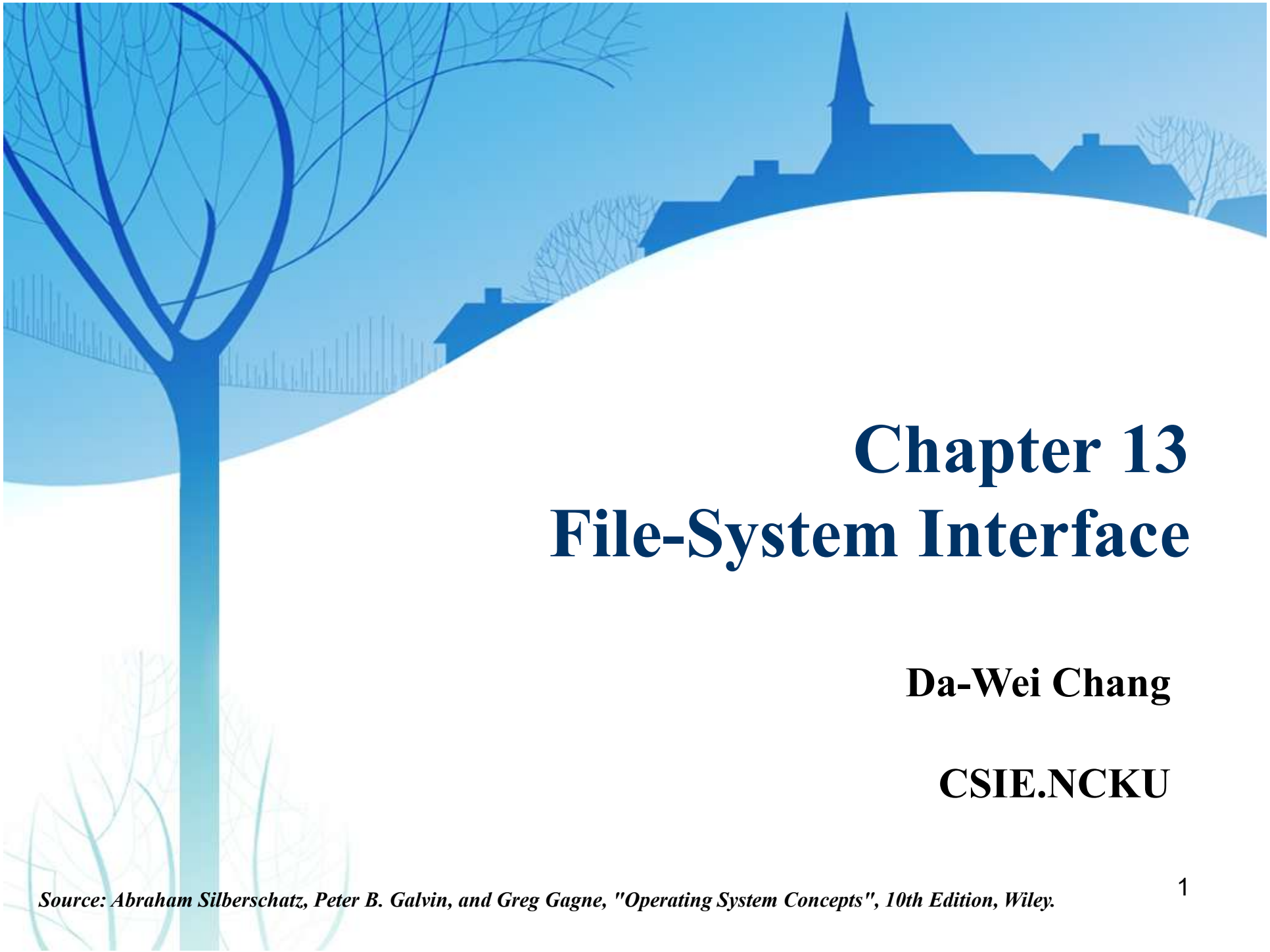
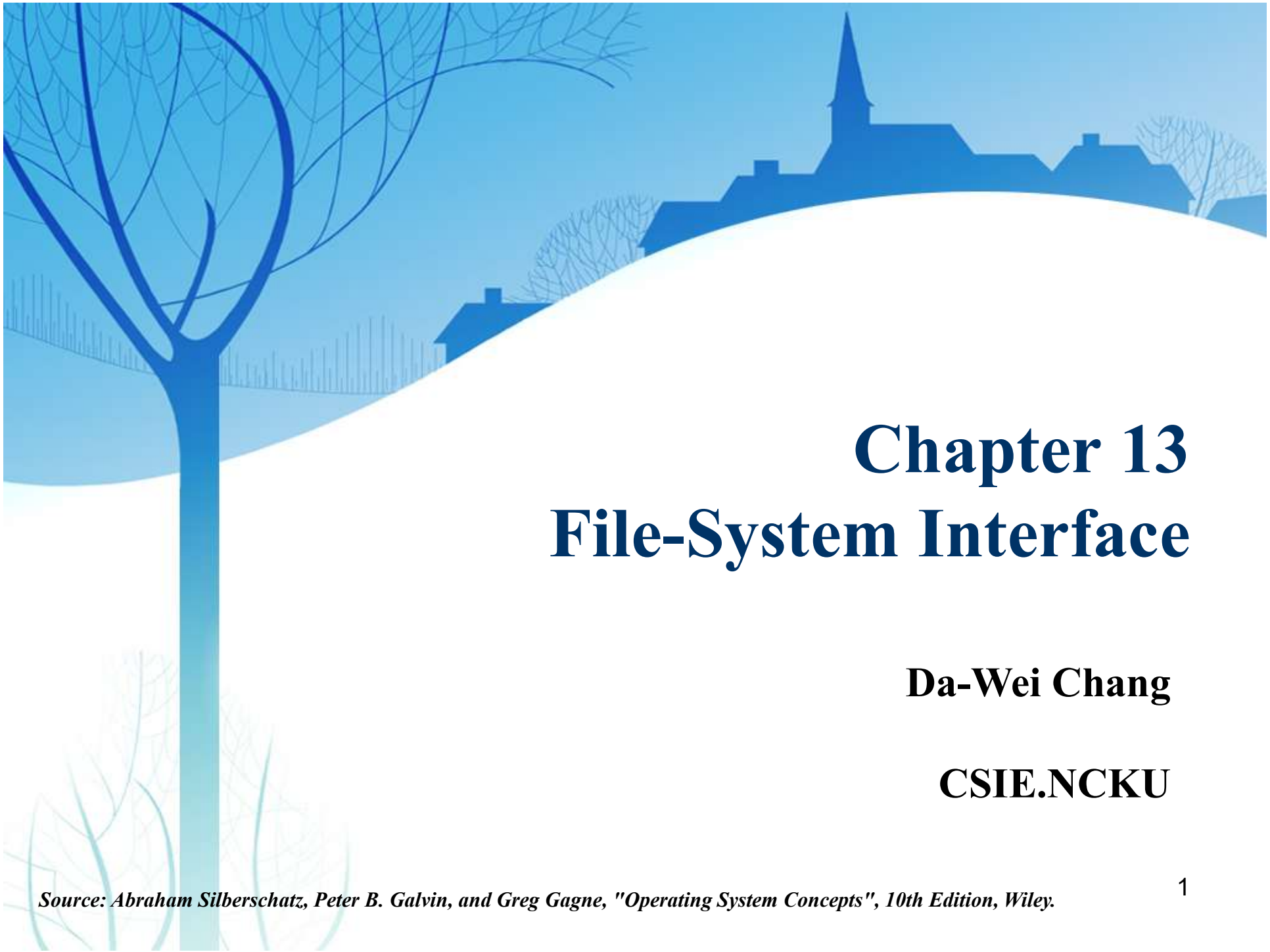


Chapter 13
File-System Interface

Da-Wei Chang

CSIE.NCKU

Source: Abraham Silberschatz, Peter B. Galvin, and Greg Gagne, "Operating System Concepts", 10th Edition, Wiley.



Chapter 13
File-System Interface

Da-Wei Chang

CSIE.NCKU

Source: Abraham Silberschatz, Peter B. Galvin, and Greg Gagne, "Operating System Concepts", 10th Edition, Wiley.

Outline

- File Concept
- Access Methods
- Directory Structure
- File-System Mounting
- File Sharing
- Protection

File Concept

- File System
  - A collection of **files**
  - A **directory** structure

- File
  - A named collection of related information that is recorded on secondary storage

- File Types
  - Data
    - numeric
    - character
    - binary
  - Program

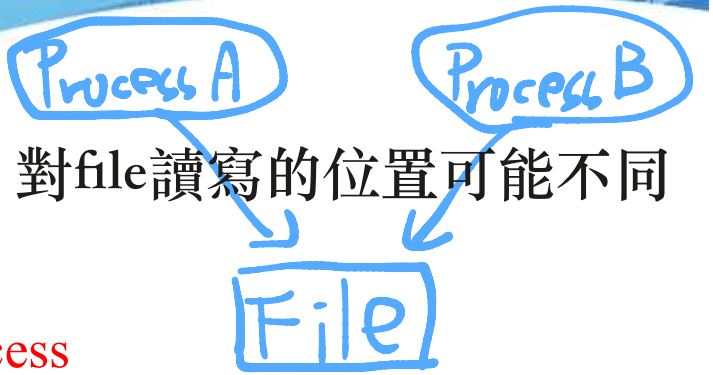
File Attributes 檔案屬性

- **Name –** file name; 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 storage 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 (or located by) the directory structure, which is maintained on the disk

File Operations

- **Common File Operations**
  - **Create**
  - **Read**
  - **Write**
  - **Reposition within file** i.e., **seek** 移動檔案的當前位置指標
  - **Delete**
  - **Truncate** 清空
  - **Open** 做所有操作前，都要先open
    - Usually invoked before the user performs operations on that file
    - Open(*F*) searches the directory structure on disk for entry *F*, and move the content of the entry (not the file content) to memory
    - Return a **file handle** for later use (e.g. both read and write use the file handle)
  - **Close**
    - Invoked when the user does not need to perform further operations on this file

Open Files



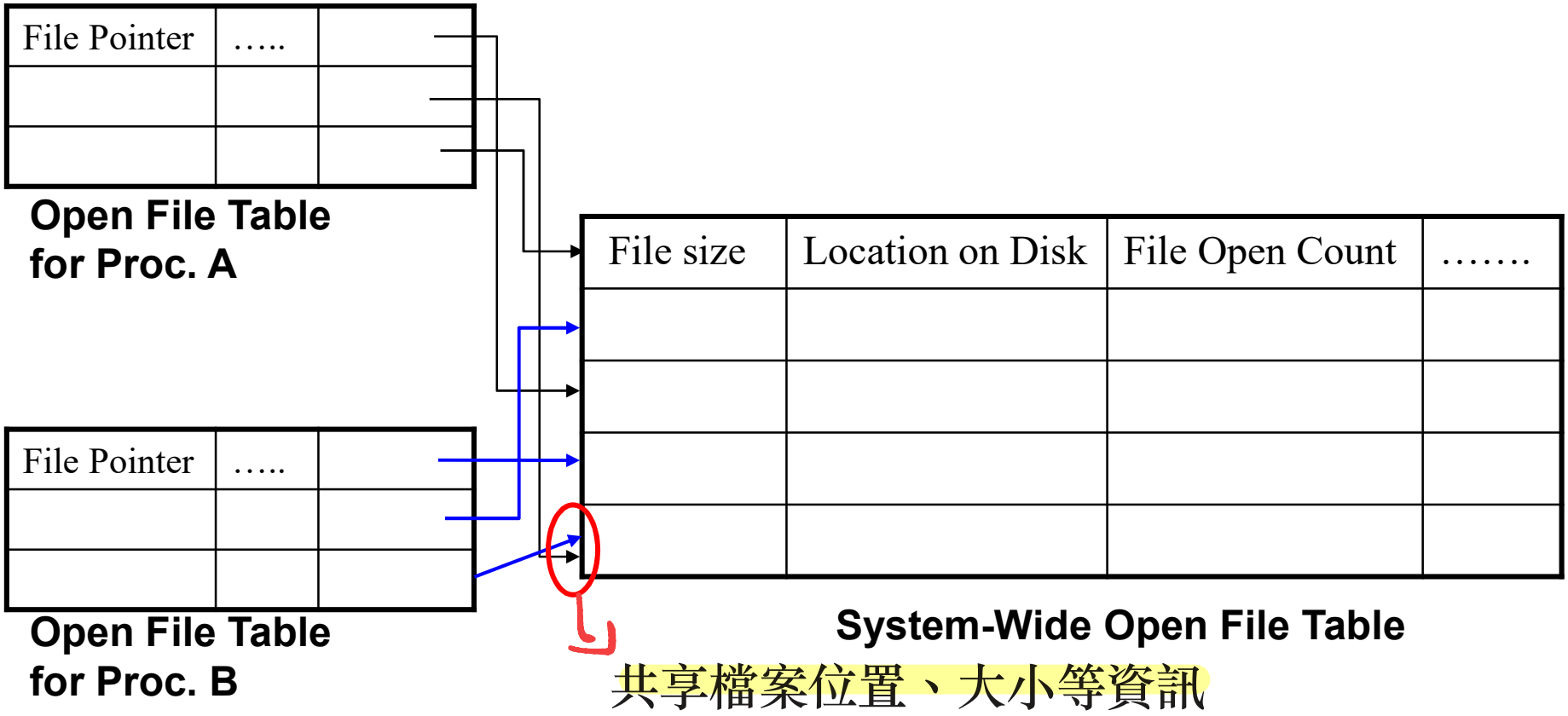
- Information kept for each open file 對file讀寫的位置可能不同
  - **File pointer**
    - pointer to last read/write location, per process
  - **File-open count** 有多少process開啟這個檔案
    - Increased/decreased by 1 for each open/close
    - allow removal of information from the open-file table (described later) when the last process closes the file
  - **Disk location** of the file
  - **Access rights**
    - per-process access mode information

Opening Files

- In a multi-user OS, several processes may open a file at the same time 多個process同時開一個file
- Two levels of open file tables
  - **Per-process table** 每個process專屬表格
    - Keep all the files opened by **the process**
    - Maintain per-process information
      - Current file pointer, access rights to the file, accounting information
    - Each entry points to the system-wide table to get further information 用來查找這張表格 看下圖
    - **File handles** are used to index the table
  - **System-wide table** 系統全域表格
    - Maintain **process-independent** information
      - Location of file on disk, access dates, file size, file open count

獨立於process的表格

Two-Level Open-File Table



Open File Locking

調解

- Mediate access to shared files
  - Prevent race condition
- Provided by some operating systems and
- Exclusive (write) or shared (read) locks
- Mandatory or advisory:

強制性 建議性

  - **Mandatory** – access (read/write/open…) may be denied depending on locks held and requested

OS管理

    - Even when another process did **not explicitly** acquire the lock
    - OS ensures the lock integrity
    - Windows OS
  - **Advisory** – processes can access the file without acquiring the lock

所有使用中的process共同管理

    - lock integrity is ensured by the cooperating processes
    - Processes should **explicitly** acquire locks to prevent race conditions
    - Unix systems

1. **互斥鎖 (Exclusive Lock):** 通常用於寫入。
   - 當行程 A 持有互斥鎖時，沒有其他人（包括想讀或想寫的）可以存取該檔案。
   - 原則：「我在改東西，閒人勿進」。
2. **共享鎖 (Shared Lock):** 通常用於讀取。
   - 當行程 A 持有共享鎖時，其他行程也可以申請共享鎖來讀取，但不能寫入。
   - 原則：「大家可以一起看，但在看完之前，誰都不准改」。

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);

            …modify file data here. . .

            // 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 );
                …. read file data here. . .
                // release the lock
                exclusiveLock.release();
        } catch (java.io.IOException ioe) {
                System.err.println(ioe);
        }finally {
                if (exclusiveLock != null)
                exclusiveLock.release();
                if (sharedLock != null)
                sharedLock.release();
        }
    }
}
```

File Types

副檔名

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 com-pressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

File Structures

1. None - sequence of words/bytes 通用的
2. Simple record structure
   - Lines 以行為單位
   - Fixed length
   - Variable length
3. Complex Structures
   - Formatted document 像是word檔
   - Executable file 可能有header檔之類的
4. Who decides/understands the structure
   - Operating system
   - Programs, or file creators

Access Methods

如何讀寫

循序讀寫

- **Sequential Access:** file content is accessed in order
  - *read next*
  - *write next*
  - *reset to the beginning of file*
  - *skip forward or backward **k** records*

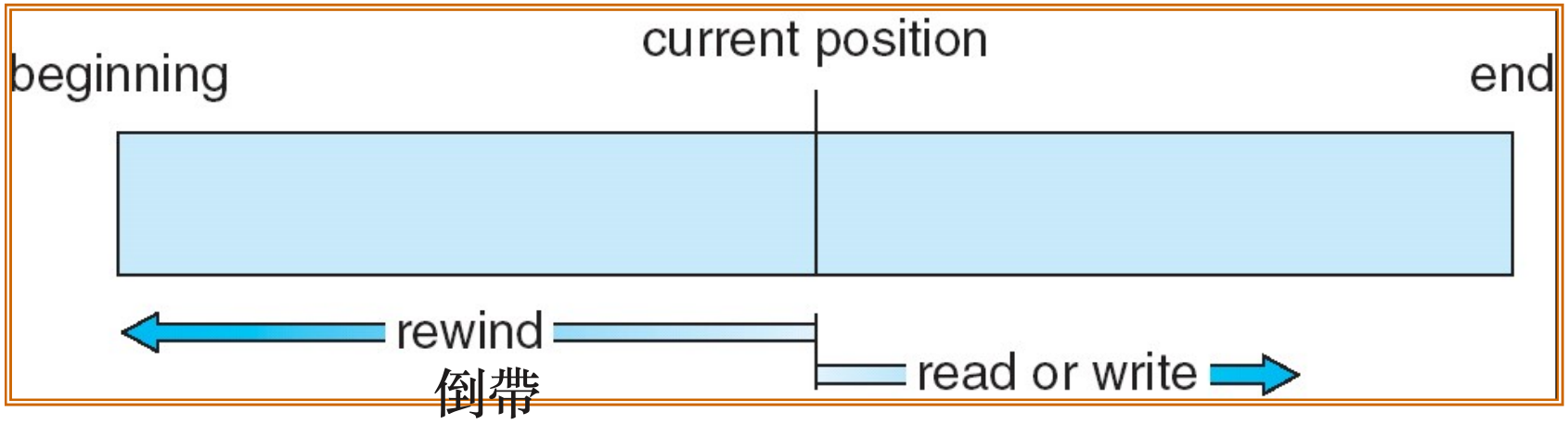
直接讀寫

- **Direct Access** (or **Relative Access):** a file is made up of fixed length logical records (or blocks), and a user can directly access block *n*
  - Model 1: include the block number as a parameter
    - *read n*
    - *write n*

    直接讀寫區塊
  - Model 2:
    - *position to n*
    - *read next*
    - *write next*

    先定位再讀寫

Sequential-access File



based on a tape model

Simulation of Sequential Access on a 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;

cp: current position
(points to the next record to be read/written)

Access Methods

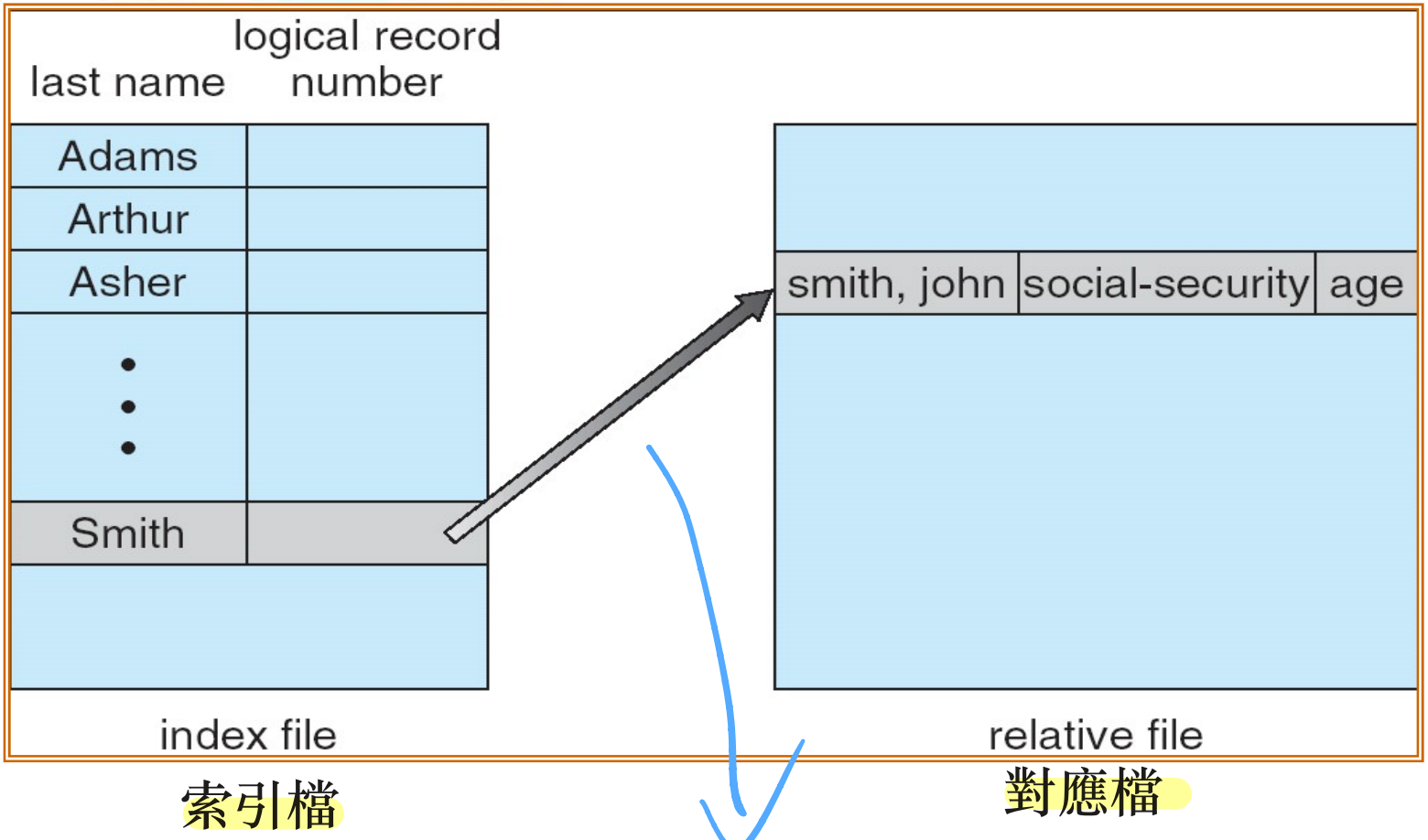
Another example: **direct access + index**

Allow us to search a large file with **little** I/O

Built on top of direct-access method

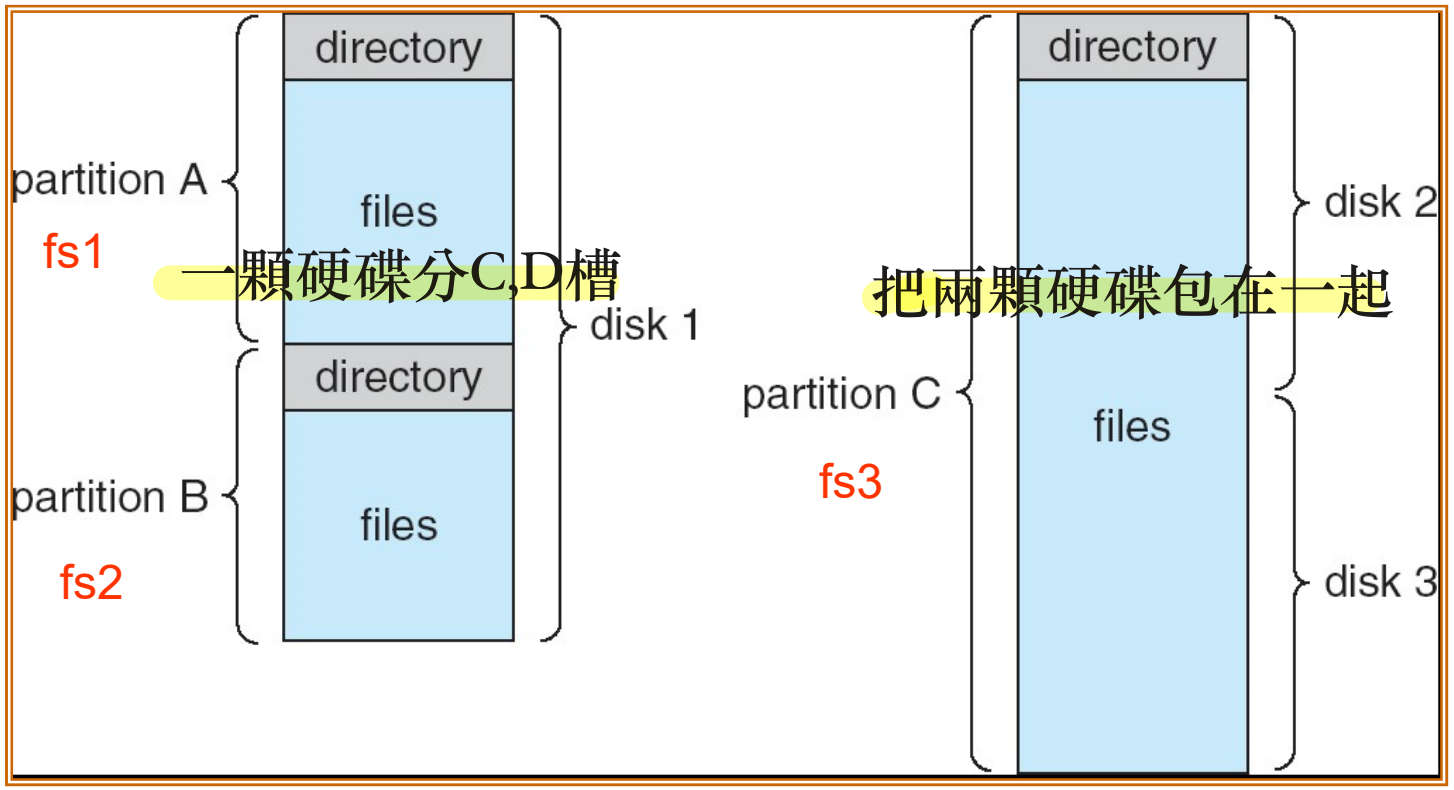
- Has an **index table**
  - The index contains pointers to various blocks of the large file
  - The index table may be placed in the memory

Example of Index and Relative Files



Search the index and then use the pointer to access the relative file
-save a lot of IO if the relative file is huge

A Typical File System Organization



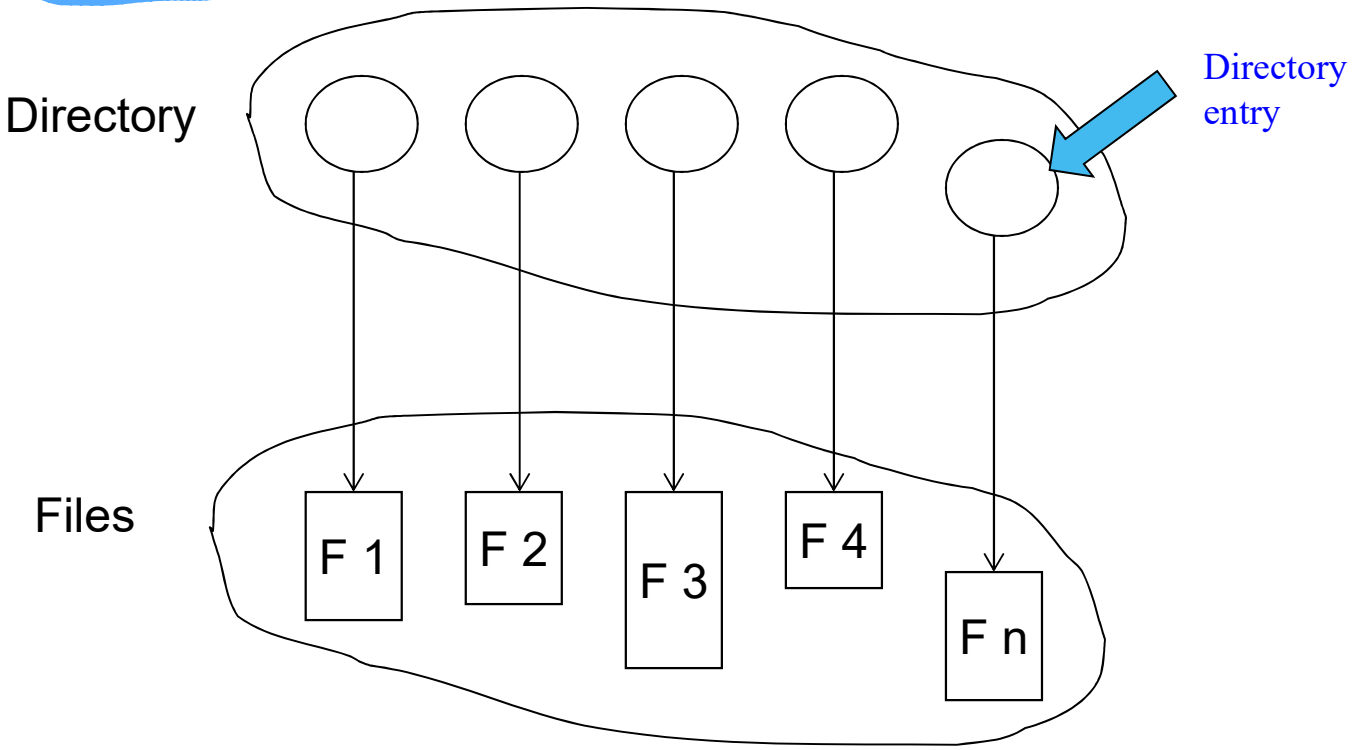
A file system consists of directories and files

A file system resides in a partition C槽 D槽

A partition can be a portion of a disk, or it can span multiple disks

Directory Structure

- A collection of entries containing information about files



Both the directory structure and the files reside on disk

Operations Performed on Directory

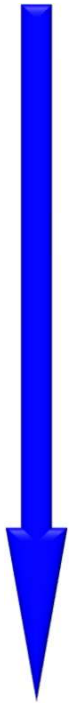
- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

Goals of a Directory Structure

- **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, …

Directory Structures

- **Directory structures**
  - Single-level directory
  - Two-level directory
  - Tree-structured directory
  - Acyclic-graph directory
  - General graph directory



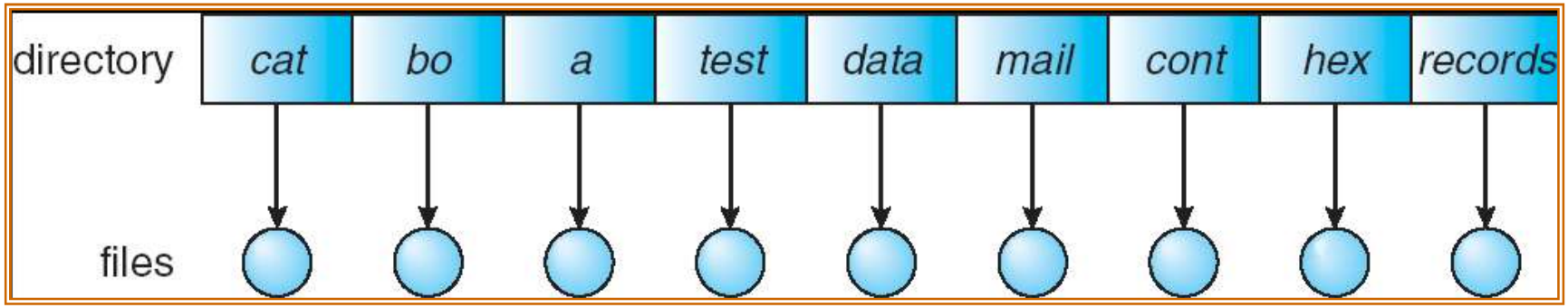
More flexible

More complicated

Single-Level Directory

所有人的檔案都放在一起

- A single directory for all users



Naming problems

. Confusion of file names among different users

. Not easy to give a new name ➔ due to name conflict

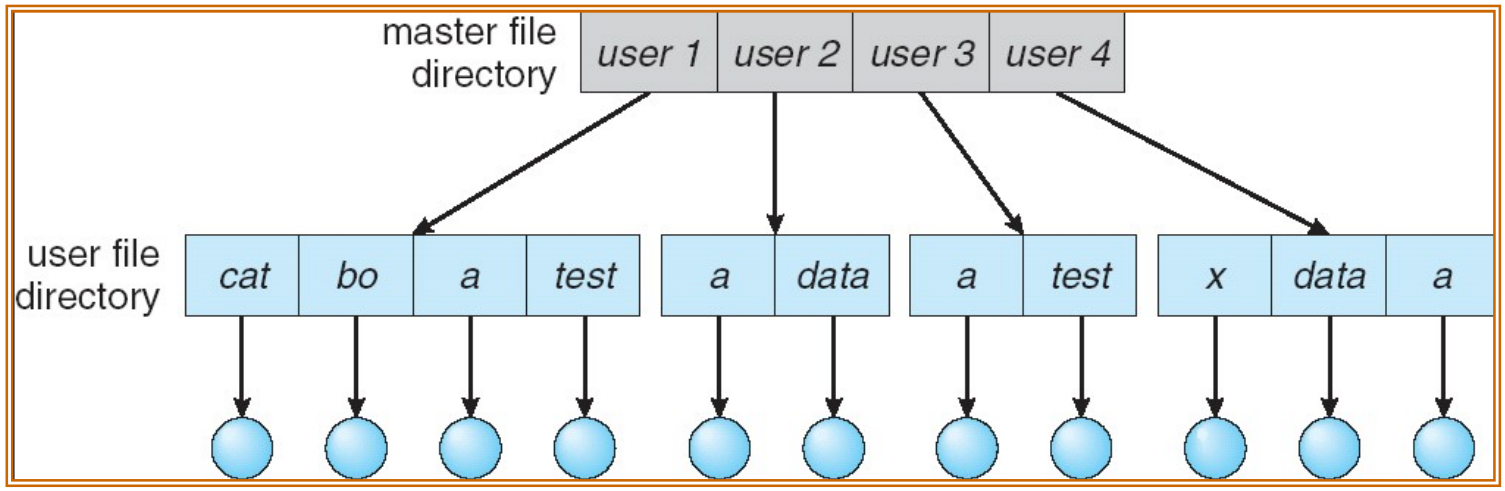
Grouping problem

. Cannot group a set of files

Two-Level Directory

每個User有自己的directory

- Separate directory for each user



Path name (e.g. /user1/cat)

Subdirectory

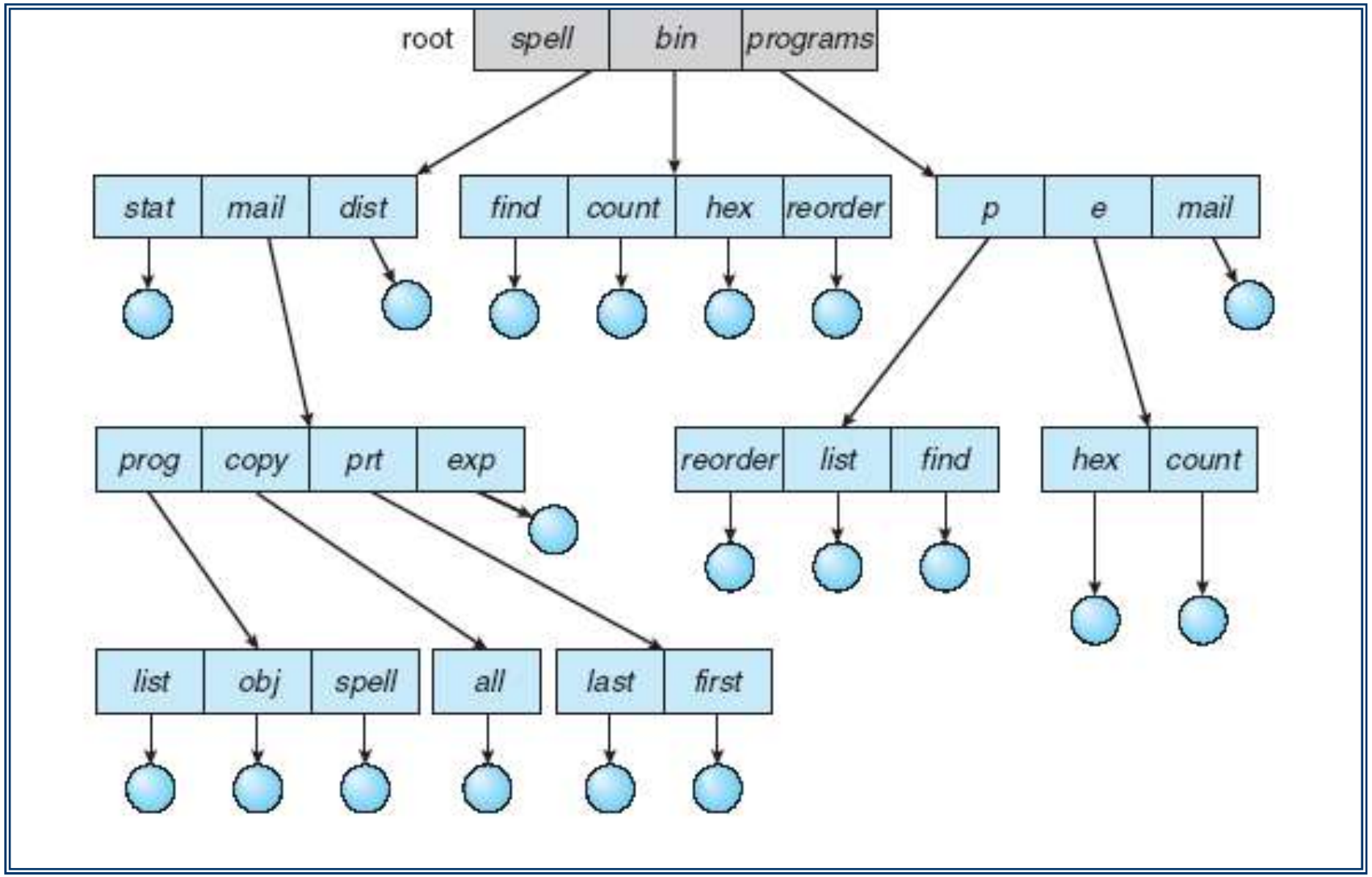
Can have the same file name for different users

Shared system files can be put in a special user directory (e.g. /user0)

Efficient searching

No grouping capability : a user **cannot group a set of files**

Tree-Structured Directories



Tree-Structured Directories (Cont)

- Efficient searching
- Grouping Capability
- have **current directory** (working directory)
  - cd */spell/mail/prog*
  - cat *list*

當前目錄

Tree-Structured Directories (Cont.)

- **Absolute** or **relative** path name
- Creating/deleting a new file can be done in current directory

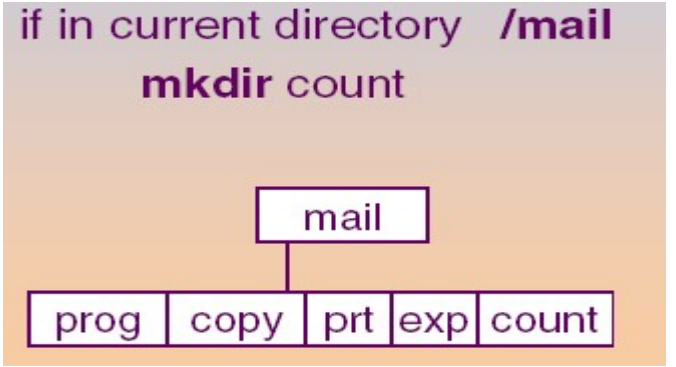
  rm <file-name>

- Creating a new subdirectory can be done in current directory

  mkdir <dir-name>

  Example: in current directory /mail

  mkdir count



Deleting "mail" ⇒ deleting the entire subtree rooted by "mail"

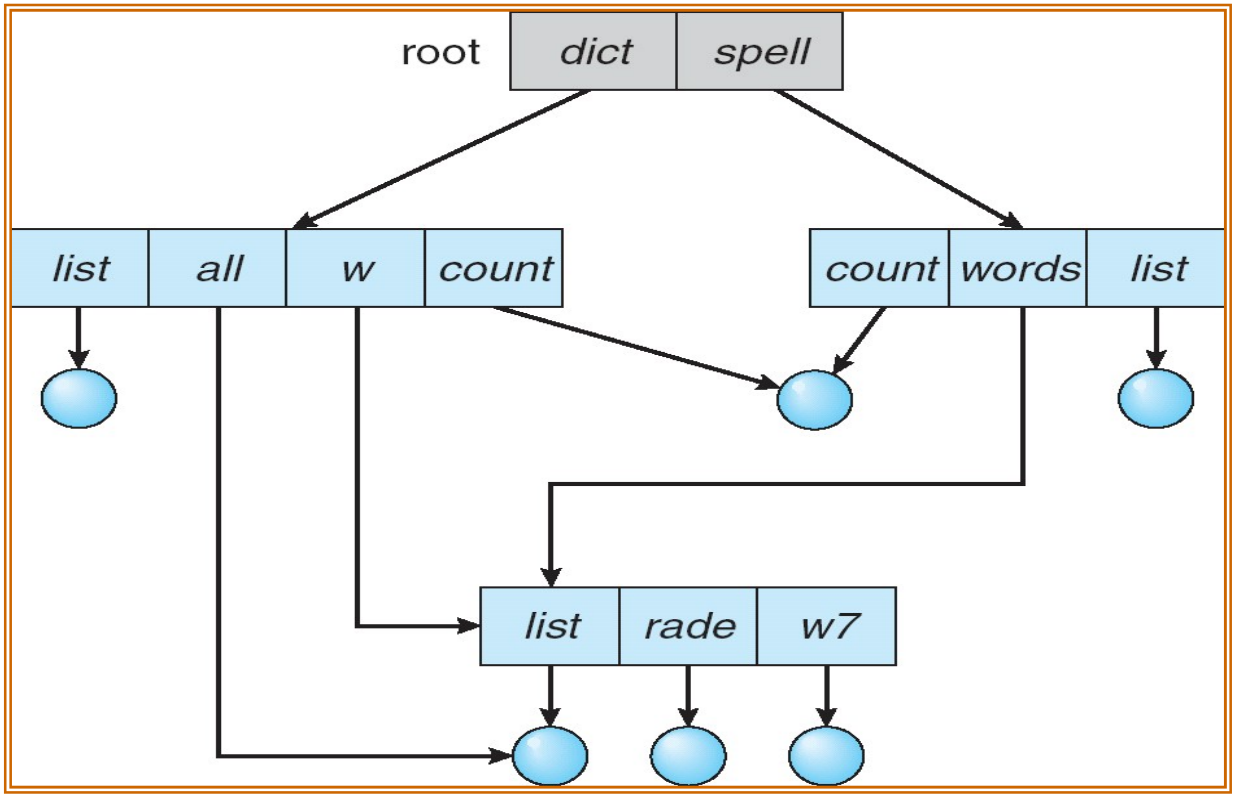
Tree structure prohibits different paths to share a file or a directory

禁止share

無迴圈

Acyclic-Graph Directories

- **Acyclic-graph**: a graph with **no** cycles
- Can have shared subdirectories and files



A file or directory can be accessed via multiple paths
e.g., /dict/all, /dict/w/list, /spell/words/list are the same file

Acyclic-Graph Directories (Cont.)

- One method of implementing shared files
  - Introducing a new directory entry type: **Link**
    - **Link** – another path name to an existing file
      - Also called **symbolic links** or **soft links**
      - Resolve the link – follow pointer to locate the file
- Problems
  - 1 – Statistics: the shared file may be counted more than once
  - 2 – File backup : the shared file may be counted more than once
  - 3 – Deletion (introduced in the next slide)

**Some operating systems, e.g. DOS, use tree structured file systems to avoid the problems*

- Maintain **acyclic** by prohibiting multiple references to **directories**

禁止對directory有多重引用

Acyclic-Graph Directories (Cont.)

- Handling of deletion
  - 1 – Link deletion ➔ just remove the link
  - 2 – File deletion
    - Delete the space of the file ➔ dangling link problem (懸空連結)
      - Search and remove all the links
      - Leaves a link until it is accessed
        - » Problem: a file with the same name could be created…
    - Leaves the space of the file until all references are removed
      - Keep the reference count

解決懸空連結

- **懸空連結 (Dangling Link)：** 又稱為懸擺指標 (Dangling Pointer)。
- **情境：** 使用者 A 和使用者 B 都共享同一個檔案。如果 A 刪除了檔案，且系統真的把硬碟上的資料清除了。
- **問題：** 使用者 B 的目錄裡仍然保有指向那個位置的連結。當 B 試圖讀取時，指標指向的是一個無效的、或已經被回收的記憶體位址，這就是懸空連結。

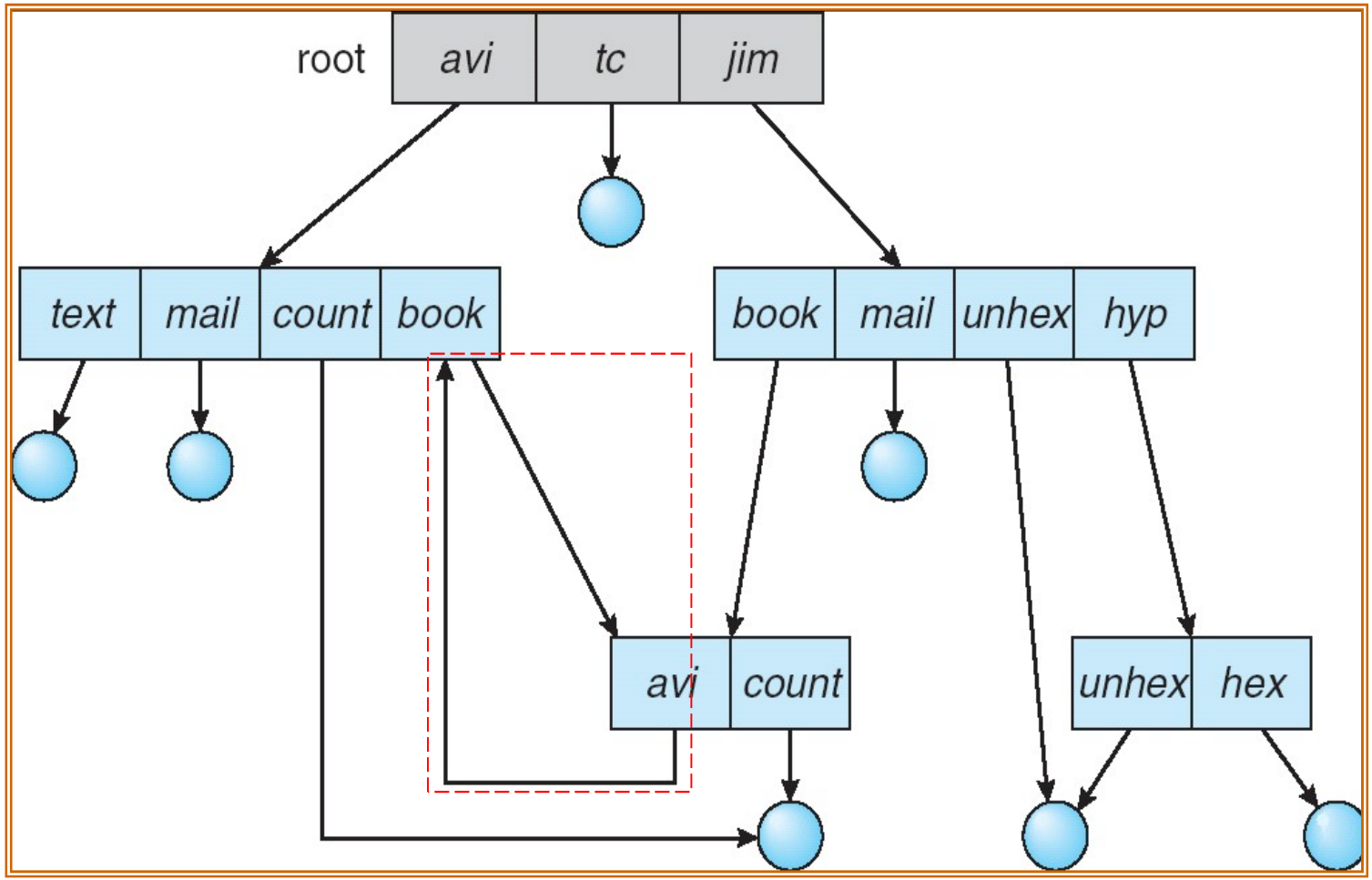
General Graph Directory

最複雜　允許迴圈

- Allow links to directory => cycles may exist
- If cycles are allowed
  - Traversal problem…
    - Infinite loop
  - Deletion is more complicated
    - A **self-referencing** directory never has its reference count as 0

    - Solutions: **Garbage collection (very expensive)**
      - **Mark** the used data and **sweep** the unused data (i.e. garbage)
      - Time consuming

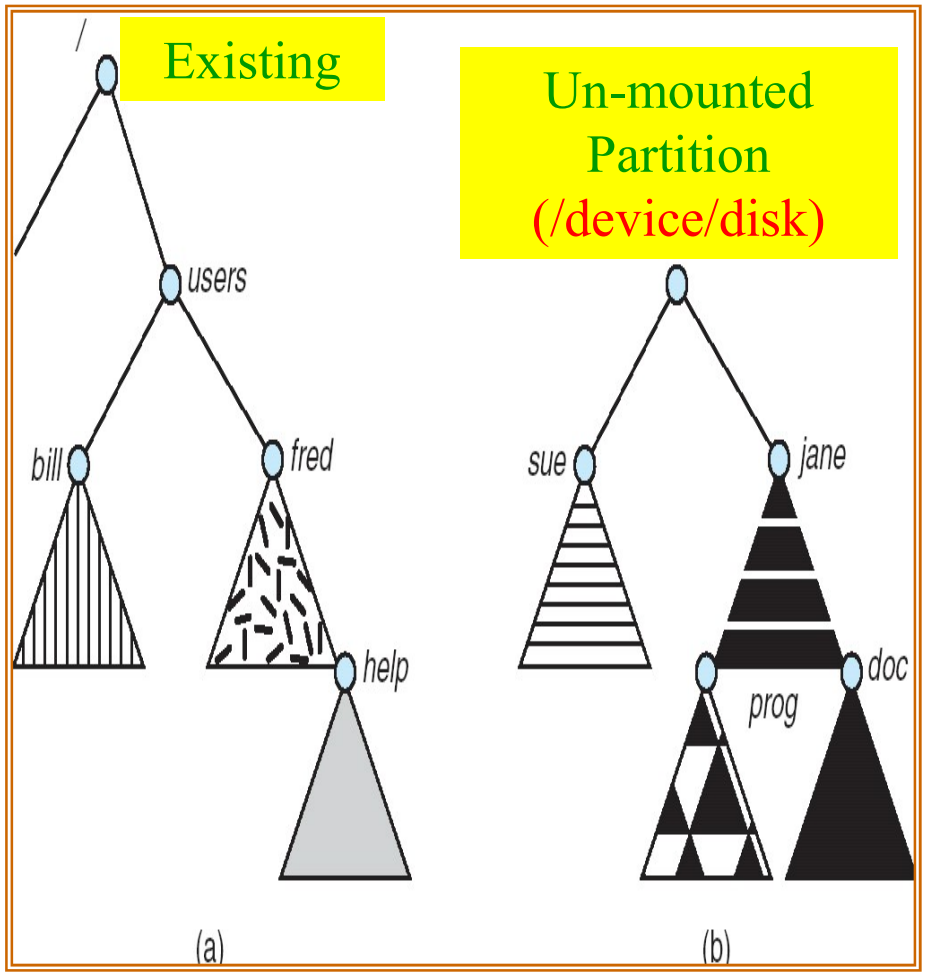
General Graph Directory

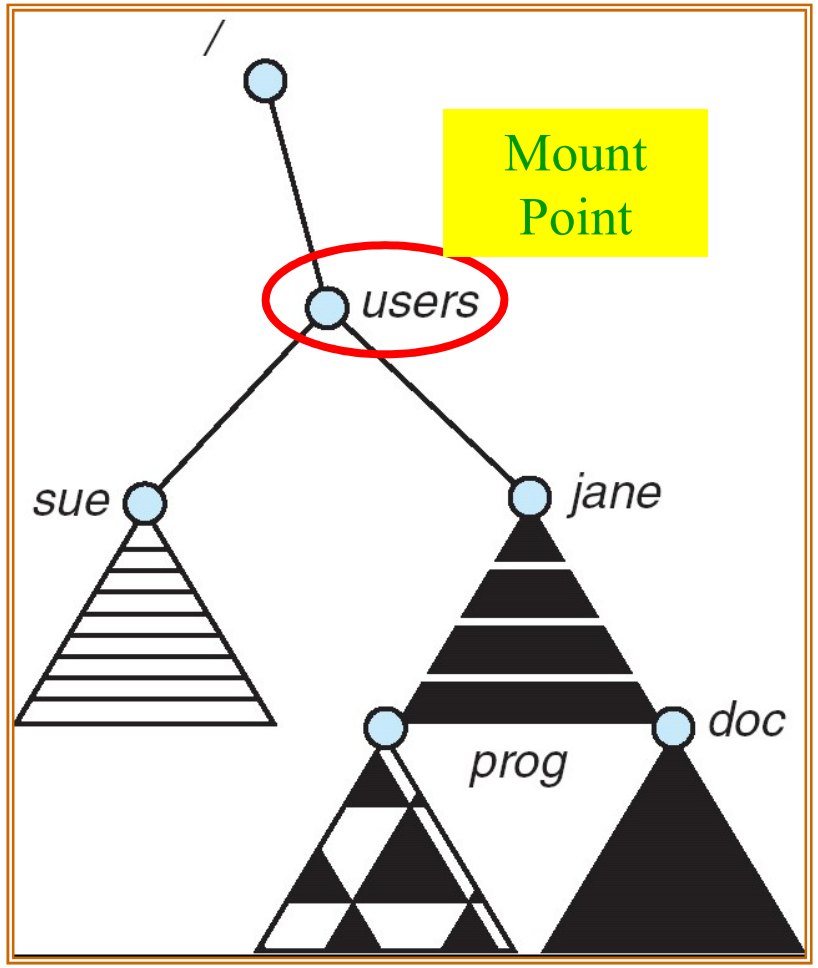


File System Mounting

- A file system must be **mounted** before it can be accessed
- A unmounted file system is mounted at a **mount point**
- Mount procedure
  - OS is given the (FS type, device name, and mount point)
    - Mount point is a **directory** (can be empty or non-empty)
    - E.g. mount -t vfat /dev/hda5 /mnt/fat
  - OS verifies that the device has a valid file system
    - The file system code reads the device and verifies that the device has the expected format
  - OS records the fact that the FS is mount on the mount point
    - Allows OS to switch among FS as appropriate during traversal

File-System Mounting





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

File Sharing

- Multiple Users
- Remote File Systems
- Failure Modes
- Consistency Semantics

File Sharing – Multiple Users

- **User IDs** identify users, allowing permissions and protections to be per-user
- **Group IDs** allow users to be in groups, permitting group access rights

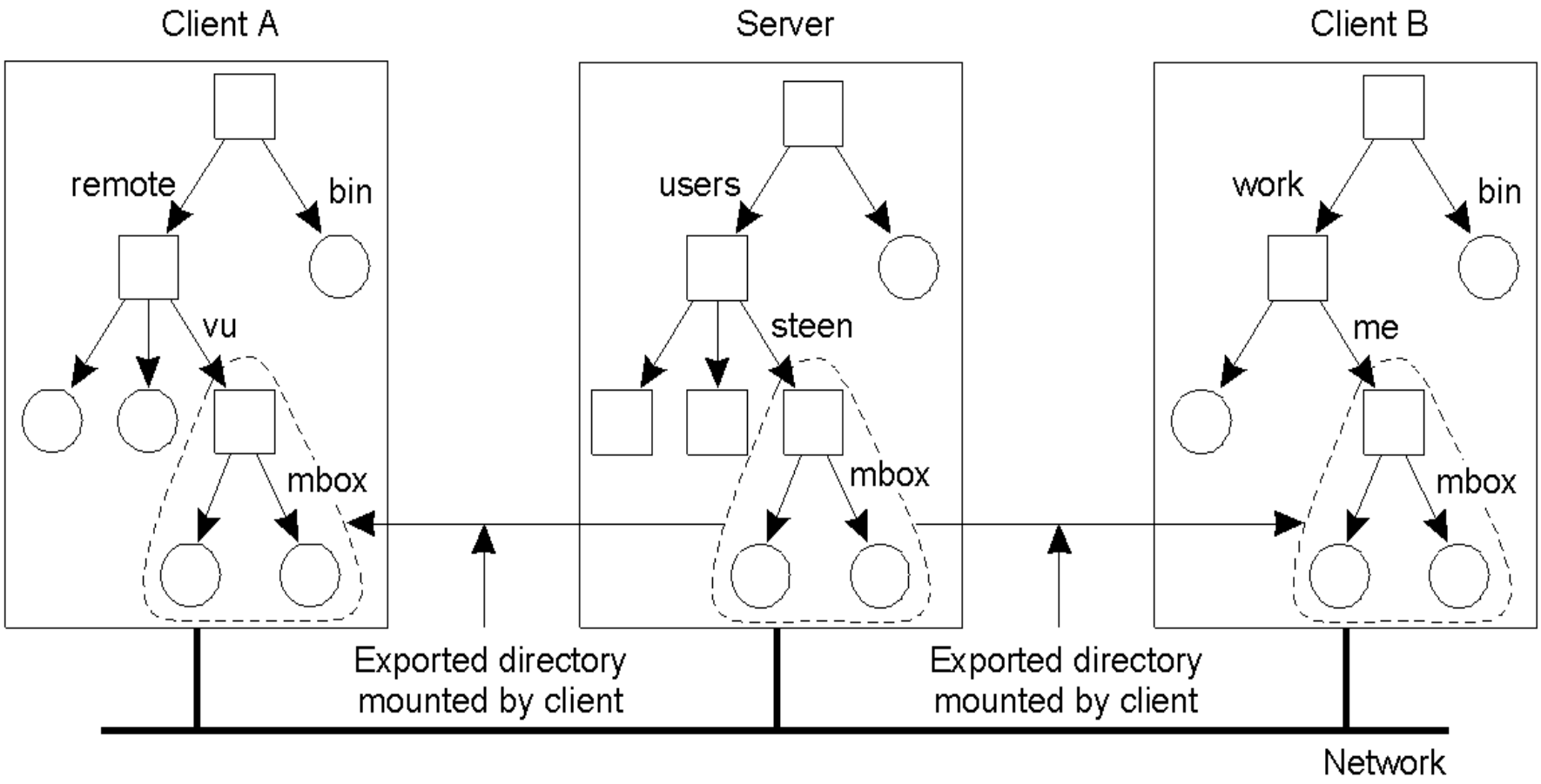
- A file has an owner who can grant access of the file to the other users
  - Who (user/group) has which (R/W…) rights?
  - For a given file, the owner's user and group ids are stored in the file attributes
  - Permissions are usually also stored in the file attributes

File Sharing – Remote File Systems

別人的資料夾掛載到我的目錄上，好像是在用我自己的

- Remote file system allow a computer to mount file systems from remote machines
  - **mount os.ncku.edu.tw:/OS /david/os**
    指令 remote的IP 掛載的 掛載點
  - Using the client-server model
    - The machine containing files is the **server**
    - The machine seeking access is the **client**
  - Protocols between clients and servers
    - **NFS**: standard UNIX client-server file sharing protocol
    - **CIFS**: standard Windows protocol
  - Standard operating system file calls are translated into **remote calls**

Mount in NFS


Client A
Server
Client B
remote
bin
users
work
bin
vu
steen
me
mbox
mbox
mbox
Exported directory
mounted by client
Exported directory
mounted by client
Network

File Sharing – Failure Modes

- Remote file systems add new failure modes, due to network failure, server failure

- The client can
  1. – Terminates the operations
  2. – Delay the operations until the network or server is reachable
     - Most remote FS protocols enforce or allow delaying operations

當機後 強制執行 或 等一下

- Recovery in the case of server failure
  1. – If the server maintains state information (i.e., stateful server)
     - E.g., open files, connections, read/write positions…
     - How to maintain server state upon server failure? Not easy…

兩種情況

  2. – Stateless implementation
     - Like NFS (before version 4)
     - Each request is **self-contained**
       - E.g., read (file name, offset)
     - Easy to implement and recover
     - But, longer message and processing time

File Sharing – Consistency Semantics

- **Consistency semantics**
  - Related to how multiple users access a shared file ***simultaneously*** 多個使用者同時共享檔案
- When modifications of data by one user can be observed by other users 其中一人修改檔案其他人會看到

File Sharing – Consistency Semantics

- **UNIX Semantics** 一個檔案只對應到一個映像
  - a file is associated with **a single image**
  - Writes to an open file **visible immediately to other users** that open the same file 有人修改其他人都看得到
  - One mode allows users to share the file pointer
    - File pointer advances will be visible by others
- **Session Semantics** 一個檔案對應到多個映像
  - a file is associated temporarily with **multiple** images
  - Session: **between open() and close()** 在open到close這段時間大家各自修改各自的
  - Writes to an open file by a user are not visible immediately to other users that have the same file open Close之後才看到別人修改了什麼
  - Writes only **visible to sessions starting after the file is closed**
- **Immutable-Shared-Files Semantics**
  - Once a file is declared as shared, it **cannot be modified**. 如果檔案是共享就不能修改

Protection

- File owner should be able to control
  - what operations can be done on this file (i.e. type of access)
  - by whom

- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List** (names and attributes)

Access Control Lists and Groups

- Access Control List (ACL) 存取控制清單
  - A list of (user names/ids, allowed access types)
  - Associated with each file (or directory)
  - **The list can be very long ….**
    - If we want to allow everyone to read file A….
  - System's users may change
    - File A's ACL needs to be updated when a new user is created
  - File attributes will become a variable-sized entity
- Use a **condensed version of the ACL**
  - e.g., in UNIX 直接歸類為三大類
    - Mode of access: read, write, execute
    - Three classes of users: **owner**, **group**, **public**

Access Control Lists in UNIX

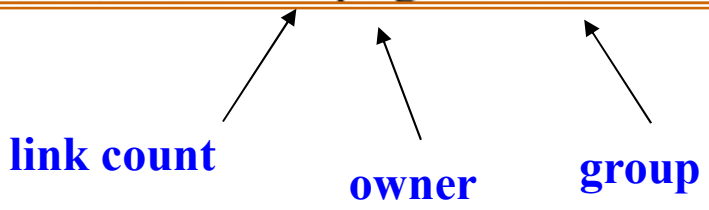
			RWX
a) **owner access**	7	⇒	1 1 1
b) **group access**	6	⇒	RWX 1 1 0
c) **public access**	1	⇒	RWX 0 0 1

Attach a group to a file

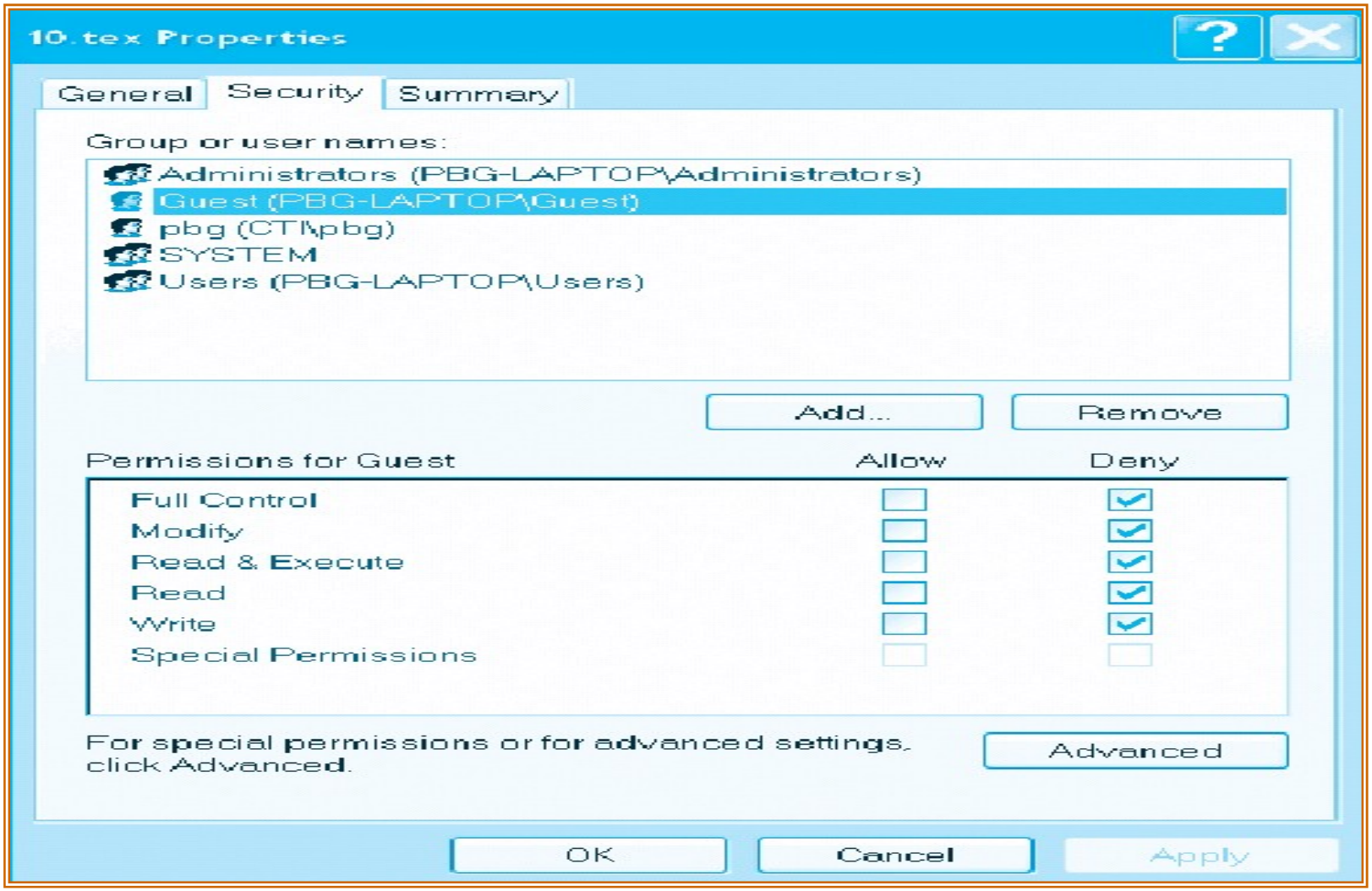
chgrp G game

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/



Access Control List Management in Windows XP



user "guest" is denied access to file *10.tex*