Module 9

File Systems - Chapters 11 and 12 (Silberchatz)

Important concepts of the chapter

- File systems
- Access methods
- Directory structures
- Protection
- File system structures
- Allocation methods
- Implementing directories
- Efficiency issues

What is a file

- **A named collection of related information saved to secondary storage**
  - Permanent nature
- **Data that is on secondary storage must be in a file**
- **Different types:**
  - Data (binary, characters….)
  - Programs

File structures

- None - byte sequences…
- Text: Lines, pages, formatted docs
- Source: classes, methods, procedures ...
- Etc.

File attributes

- **These are the properties of the file and are stored in a special file called the directory. Examples of attributes:**
  - Name:
    - to allow people to access the file
  - Identifier:
    - A number allowing the OS to identify the file
  - Type:
    - Ex: binary, or text; when the OS supports this
  - Position:
    - Indicates the disk and the address of the file on disk
  - Size:
    - In bytes or in blocks
  - Protection:
    - Determine who can write, read, execute ...
  - Date:
    - for the last modification, or last use
  - Other…

File operations: basic

- **Creation**
- **Writing**
  - Write pointer that gives the write position
- **Reading**
  - Reading pointer
- **Positioning in a file (search time)**
- **Delete a file**
  - Freeing up space
- **Truncation: resetting the size to zero while keeping the attributes**

Other operations

- **Add Info**
- **Renaming**
- **Copy**
  - can be done by renaming: two names for one file
- **Opening a file: the file becomes associated with a process which keeps its attributes, position, etc.**
- **Closing**
- **Opening and closing can be explicit (ops *open, close*)**
- **or implied**

Information related to an open file

- **File pointers**
  - For sequential access
  - Eg. for read, write
- **Opening counter**
- **Location**

File types

- **Some OS use the file name extension to identify the type.**
  - Microsoft: An executable file must have the extension .EXE, .COM, or .BAT (otherwise, the OS will refuse to execute it)
- **The type is not defined for some OS**
  - Unix: the extension is used (and recognized) only by applications
- **For some OS the type is an attribute**
  - MAC-OS: the file has an attribute that contains the name of the program that generated it (ex: Word Perfect document)

File types

file type	usual extension	function
executable	exe, com, bin or none	read 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, rrf, doc	various word-processor formats
library	lib, a, so, dll, mpeg, mov, rm	libraries of routines for programmers
print or view	arc, zip, tar	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	binary file containing audio or A/V information

Logical file structure

- **The type of a file specifies its structure**
  - The OS can then support the different structures corresponding to the types of files
    - This complicates the OS but simplifies the applications
- **Usually a file is a collection of records**
- Each record is made up of a set of fields
  - A field can be numeric or string of chars.
  - The records are of fixed or variable length (depending on the type of the file)
- **But for Unix, MS-DOS and others, a file is simply a series of "byte stream" bytes.**
  - So here, 1 record = 1 byte
  - It is the application that interprets the content and specifies a structure
  - More versatile but more work for the programmer

Access methods

Sequential
Indexed Sequential
Indexed
Direct

Access methods: 4 basic

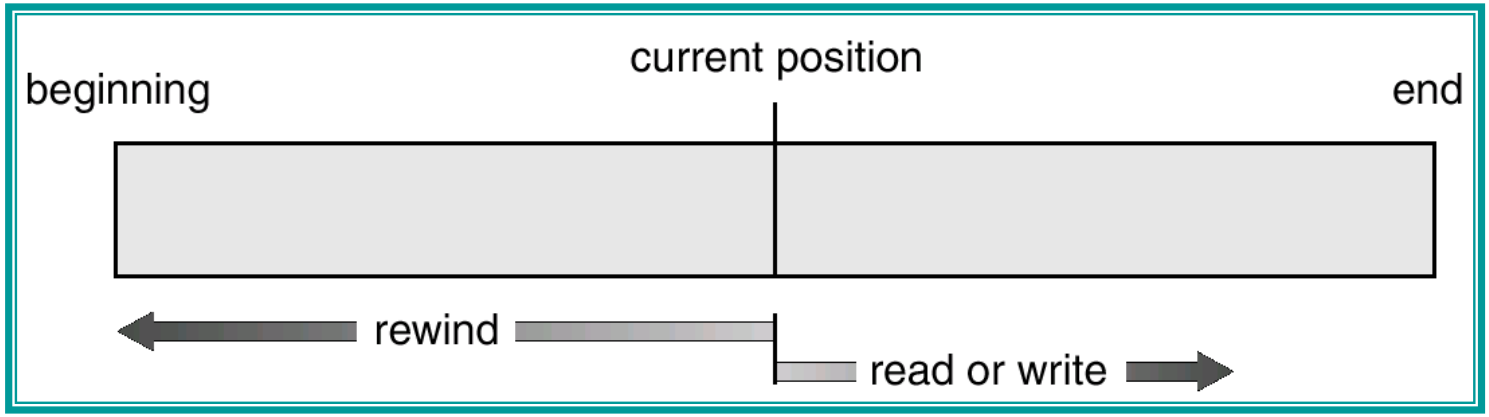
- **Sequential (tapes or disks): read or write records in a fixed order**
- **Sequential indexed (disks): sequential access or direct access (random) by using indexes**
- **Indexed: multiplicity of indexes according to needs, direct access by the index**
- **Direct or hash: direct access through hash table**
- **Not all OS support all access methods**
  - When the OS does not support them, it's up to the application to support them

File Access Methods

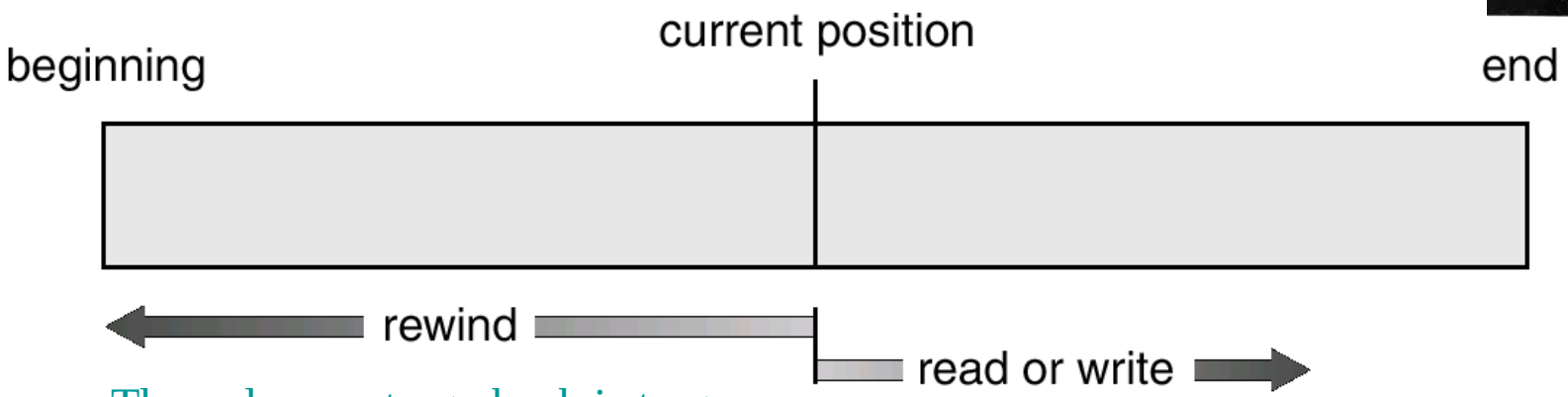
- **The logical structure of a file determines its access method**
- **Mainframe OSs generally provide several access methods**
  - Because they support several types of files
- **Several modern OS (Unix, Linux, MS-DOS…) provide a single access method (sequential) because the files are all of the same type (eg: sequence of bytes)**
  - But their file allocation method (see + below) usually allows applications to access files in different ways.
- **Ex: database management systems (DBMS) require more efficient access methods than just sequential**
  - A DBMS on a "mainframe" can use a structure provided by the OS for efficient access to records.
  - A DBMS on an OS which only provides sequential access must therefore "add" a structure to the database files for faster direct access.

Sequential Access

- **The most common method and the simplest method**
- **Records can be accessed one after the other in their sequential order of storage**
  - Based on the tape model
- **The operation read_next() reads the next record and advance the pointer to the next record**
  - Operation write_next() appends a record at the end
- **We may also come back to the beginning (rewind)**
- **It is sometimes possible to jump n records (backward or forward)**

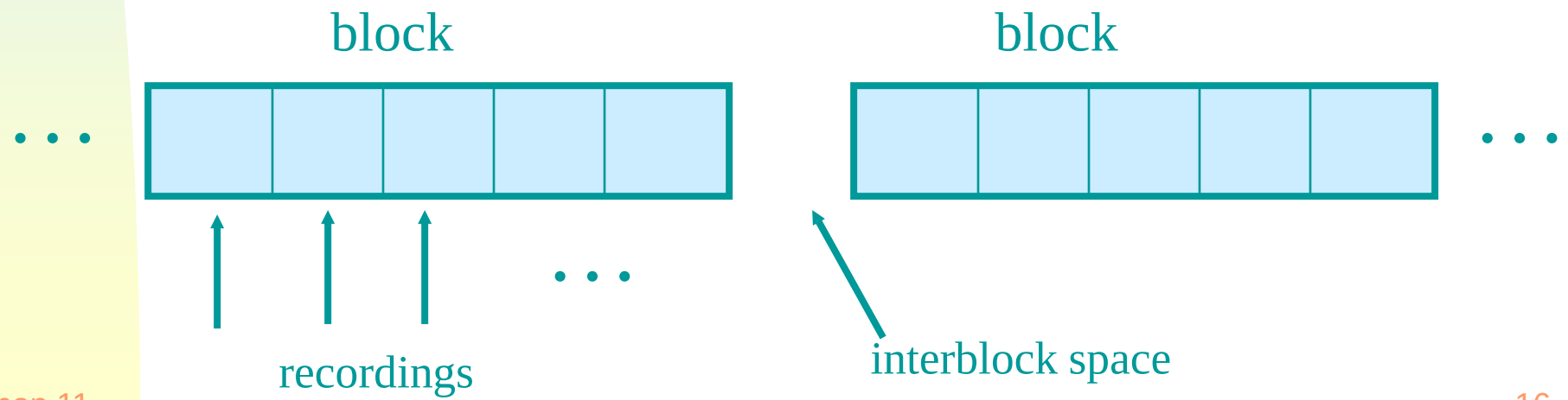


Sequential access files (Tapes)



The only way to go back is to go back to the beginning (rewind, rewind)

Forward only, 1 single record. at a time



Direct Access

- **Based on disk model containing blocks of data**
- **A direct access file consists of a set of logical blocks**
  - Their size is the same as those of physical blocks
  - They are numbered from 0 to k (for a file of k+1 blocks)
    - But logical blocks are positioned on arbitrarily-chosen physical blocks on disks according to the file allocation method used (see later).
  - Each logical block can be directly (and independently) accessed
- **Each logical block consists of R records of the same size**
  - Hence, we may have internal fragmentation
  - The first logical block contains the first R records. The next group of R records is in the second logical block, an so on…
- **Access to record number N is done by first extracting logical bloc number $\lfloor N/R \rfloor$ from secondary memory and bringing it into main memory**
  - This is the logical block containing the desired record

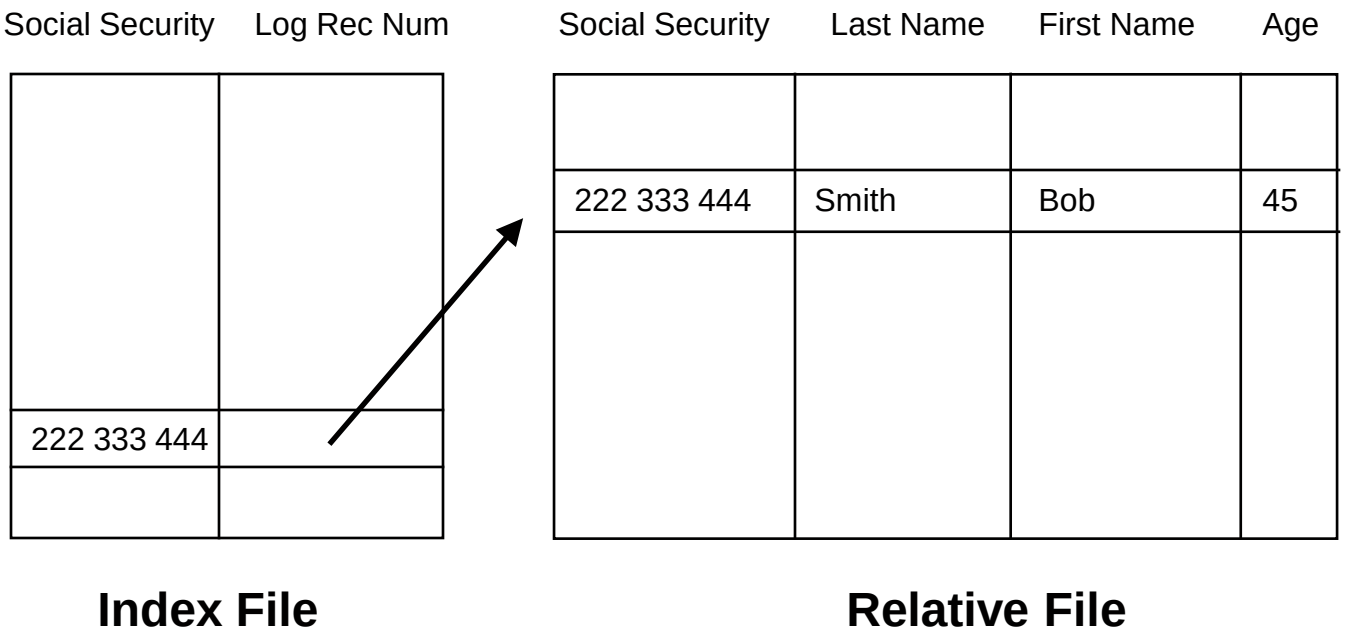
Direct and Sequential Access

- **Not all OS's offer both sequential and direct access**
  - Easy to simulate sequential access with direct access
    - Maintain a pointer *cp* that indicates the current position in a file.
  - The reverse is very difficult and inefficient.

Indexed Files

- **It is a direct access method performed by using an index**
  - Heavily used by DBMS
- **We need to use two files (per data file):**
  - A relative file : a direct access file containing the data (in logical blocks)
  - An index file : containing the indexes
- **An index consists of a key and a pointer**
  - The pointer has for value the logical block number containing the record identified by the key
  - The key consists of one the fields in record of the relative file. Its value provide a unique identifier for each record.
    - It is not permitted to have two different records of the same file having the same key value
    - Example of key: social insurance number

Indexed Files (cont.)

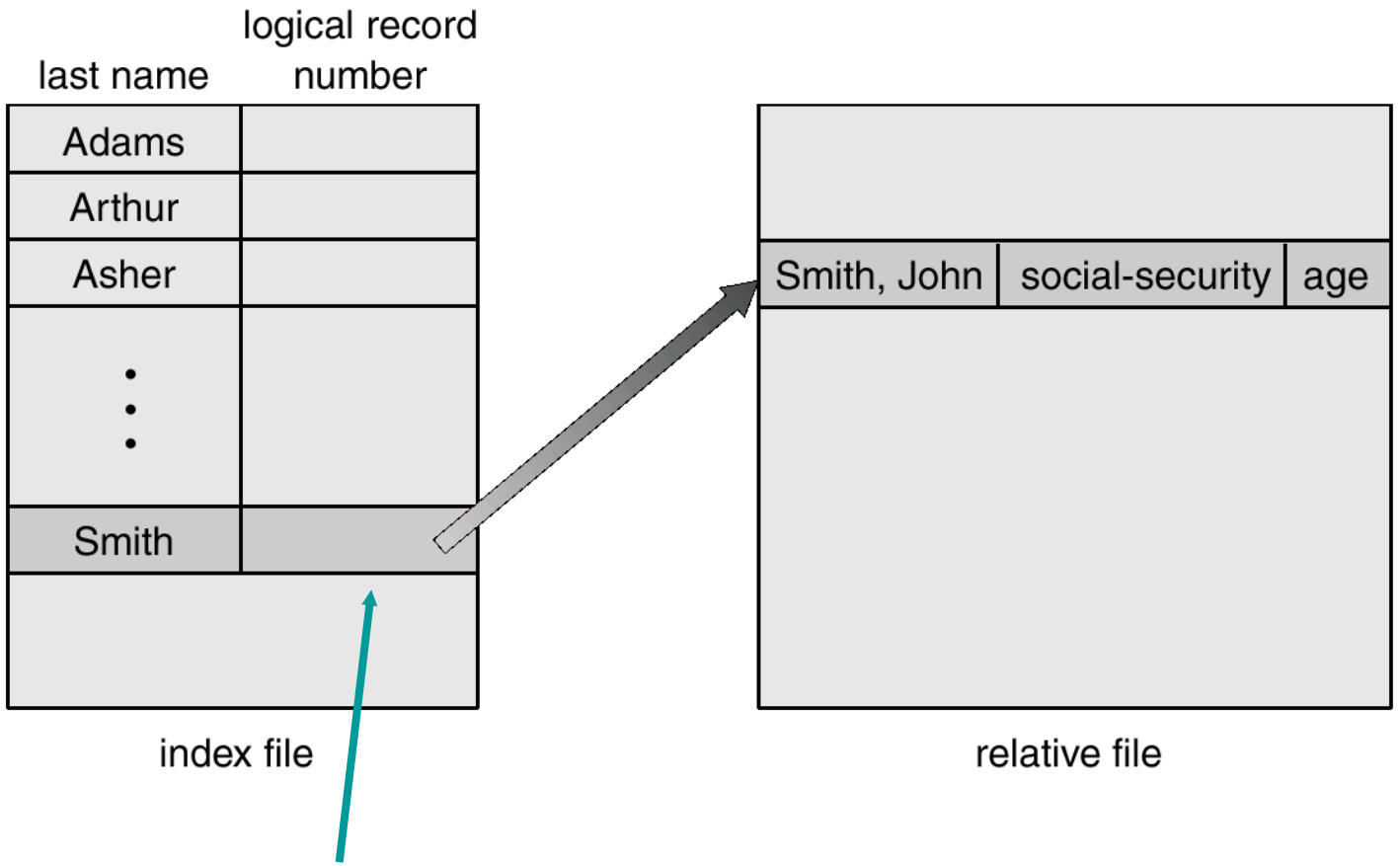


- **The index file is an ordered list (by key value)**
- **We simply perform a binary search whenever the index file can fit entirely into main memory**
  - If not, then we can create an index for the index file!
    - The primary index file is first consulted to find the logical block of the secondary index file containing the desired key value
    - That logical block is then searched to find the logical block containing the desired record

Indexed Files (continued)

- **An index can also lead to a point close to the desired record.**
  - The keys in the index file contains references (pointers) to certain important points in the relative file (for e.g., beginning of names that start with the letter S, beginning of the Smiths, beginning of serial numbers that start with 8)
  - The index file provides the means to access rapidly a point in the relative file; the search then continues from that point.
  - Such a file could also index another index file, i.e. serves as the primary file described in the previous slide.

Example of Indexing



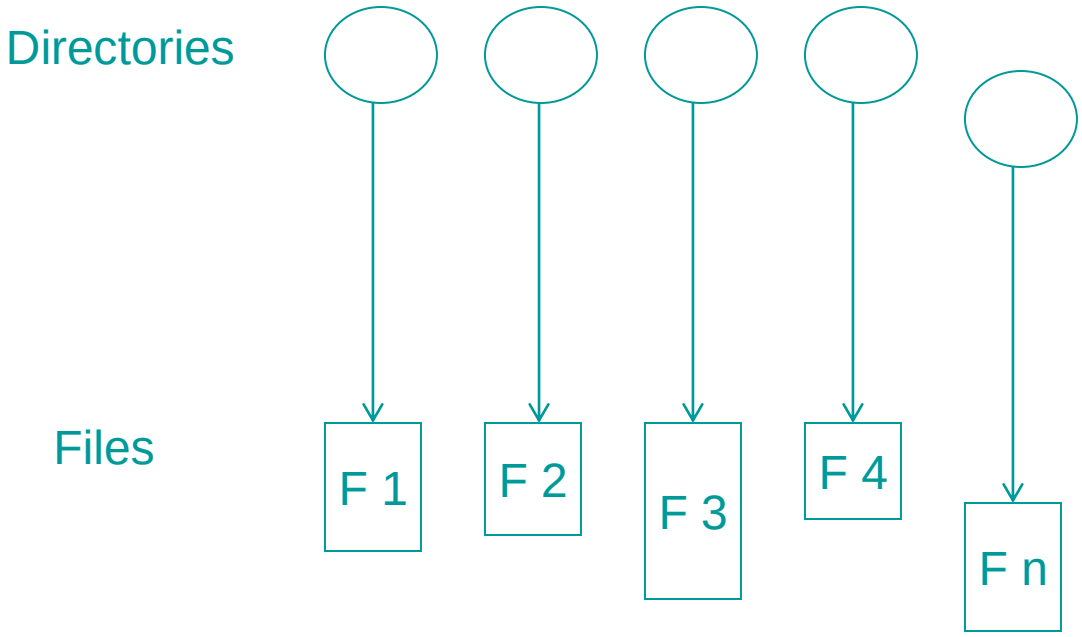
Points to the start of the Smiths (there are many)

The relative file must support both direct access and sequential access.

Directories

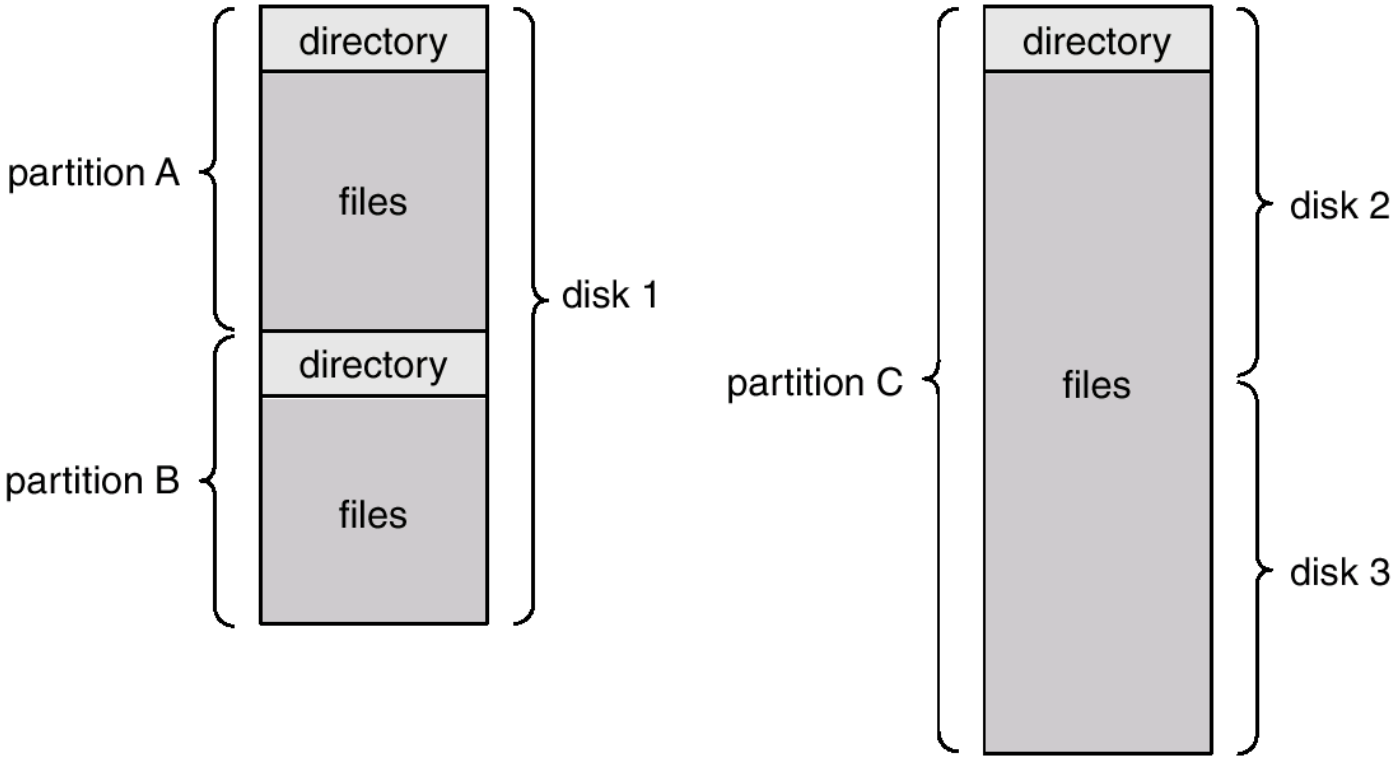
Directory structures (directories)

- **A collection of data structures containing file info.**



- **Both directories and files are on disks**
- **With the exception of one root dir. in central mem.**

Typical file system organization



Information in a Directory

- File names
- Type
- Location on disk (or other device)
- Current length
- Maximum length
- Date last accessed
- Date last modified
- Owner
- Protection

Operations Performed on Directory

What are the operations performed on a directory?

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

Why do we use directories?

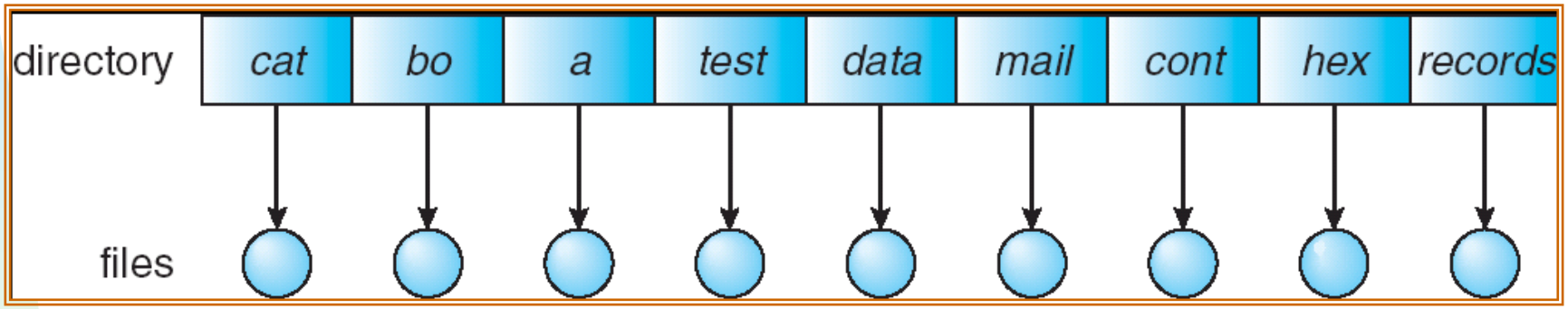
- **Efficiency – locating a file quickly**
- **Naming – convenient to users**
  - Two users can have same name for different files
  - The same file can have several different names
- **Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, …)**

Each directory holds directory entries for the files it contains

- We will discuss this in detail when we will talk about file system implementations

Single-Level Directory

- A single directory for all users

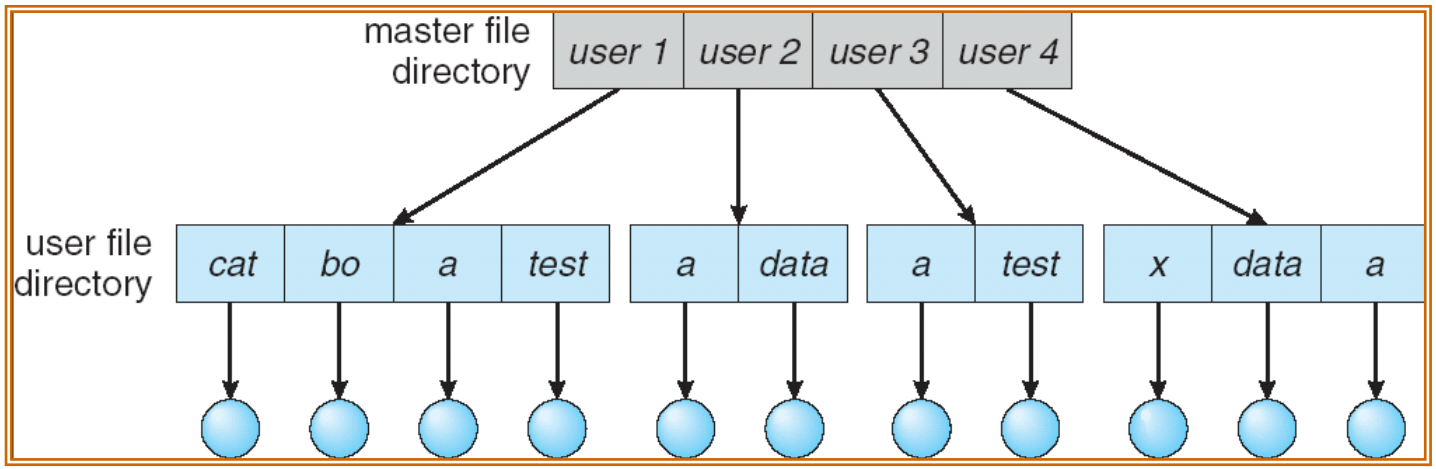


Any problems?

- With many files becomes total mess
- Naming conflicts between users
- …
- Primitive and not practical

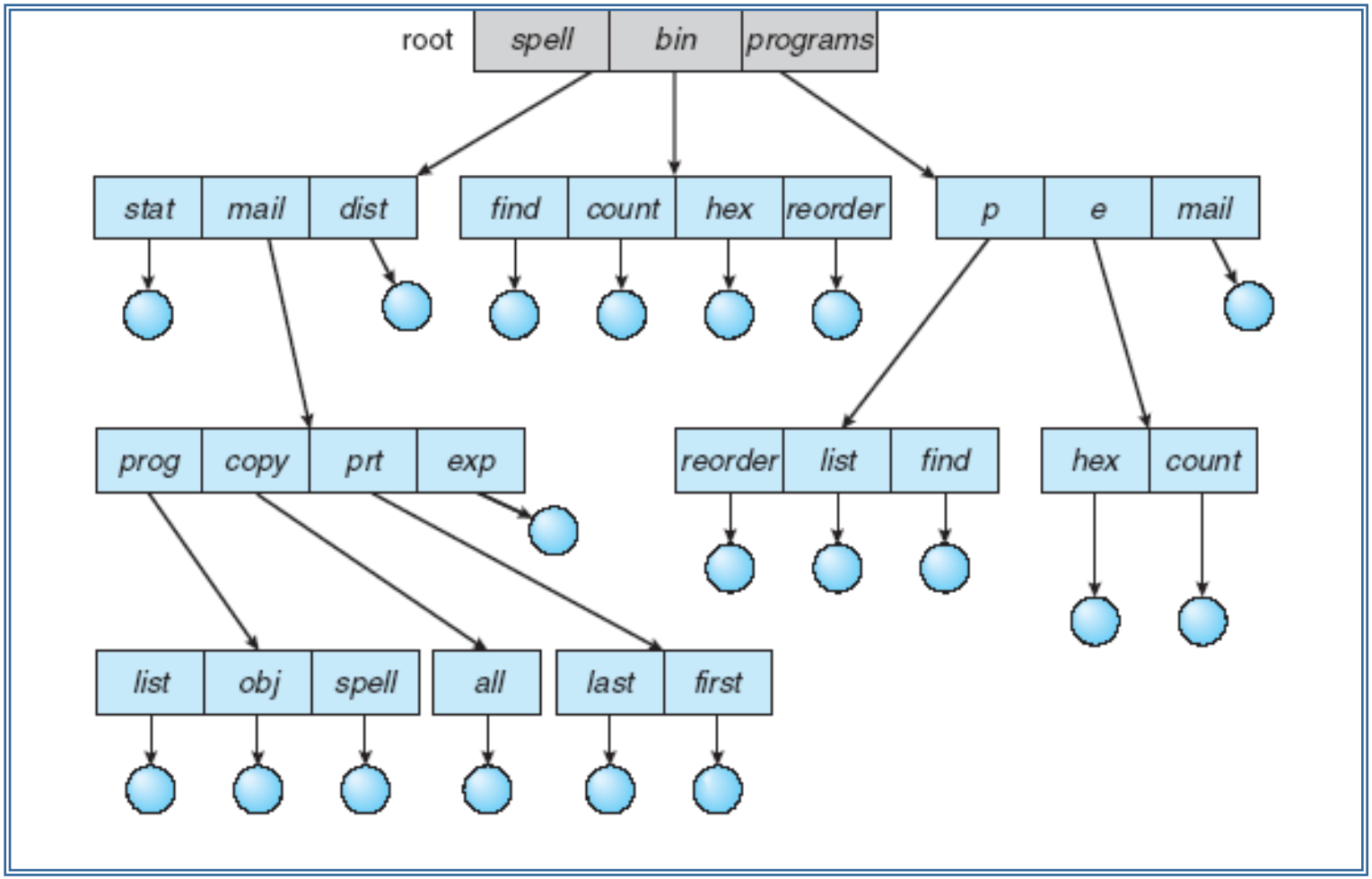
Two-Level Directory

- **Separate directory for each user**



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

Tree-Structured Directories



Tree-Structured Directories (Cont)

- **Efficient searching**
- **Grouping Capability**
- **Useful concept: current/working directory**
  - cd /spell/mail/prog
  - type list

Tree-Structured Directories (Cont)

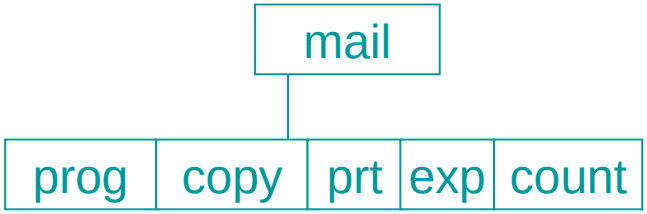
- Absolute **or** relative **path name**
- **Creating a new file is done in current directory**
- **Delete a file**

  rm <file-name>
- **Creating a new subdirectory is done in current directory**

  mkdir <dir-name>

  Example: if in current directory /mail

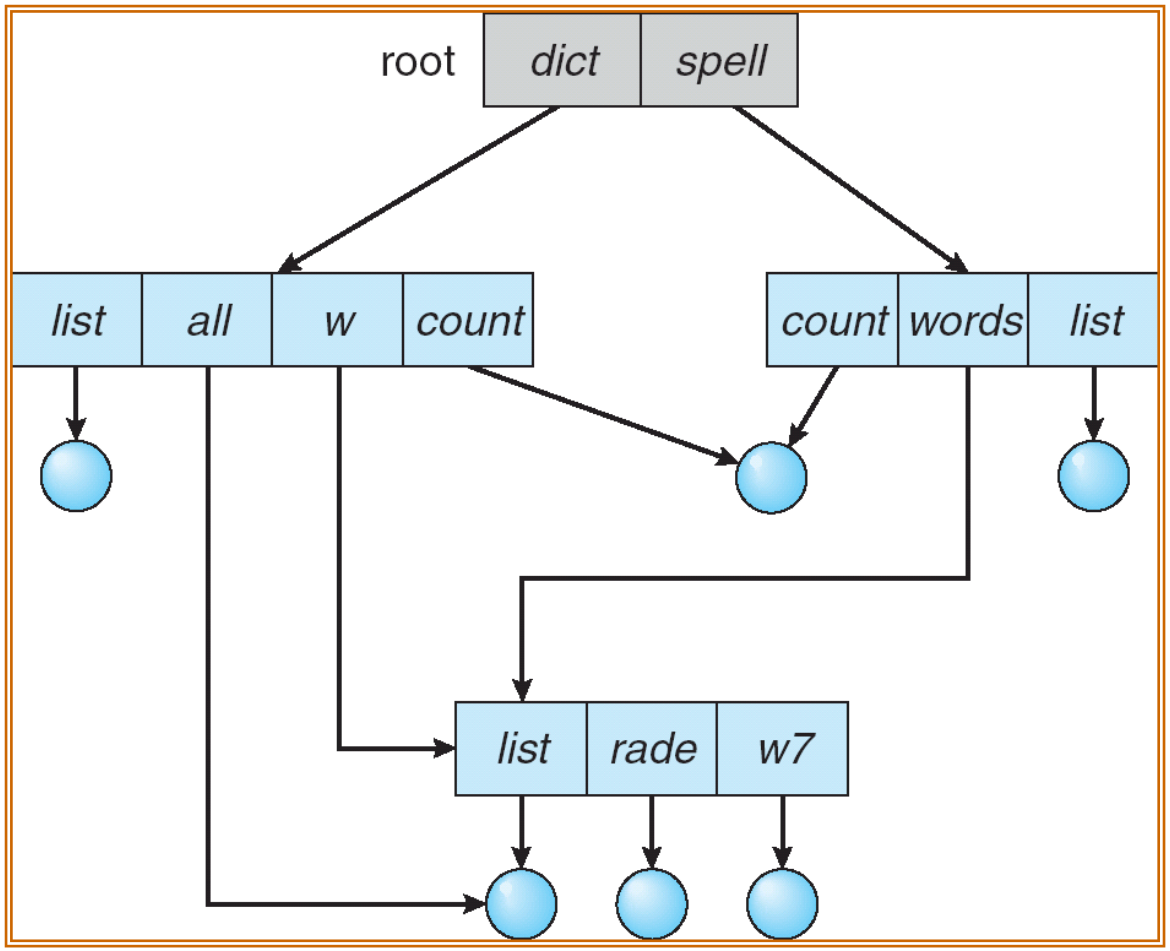
  mkdir count



Deleting “mail” ⇒ deleting the entire subtree rooted by “mail”

Acyclic-Graph Directories

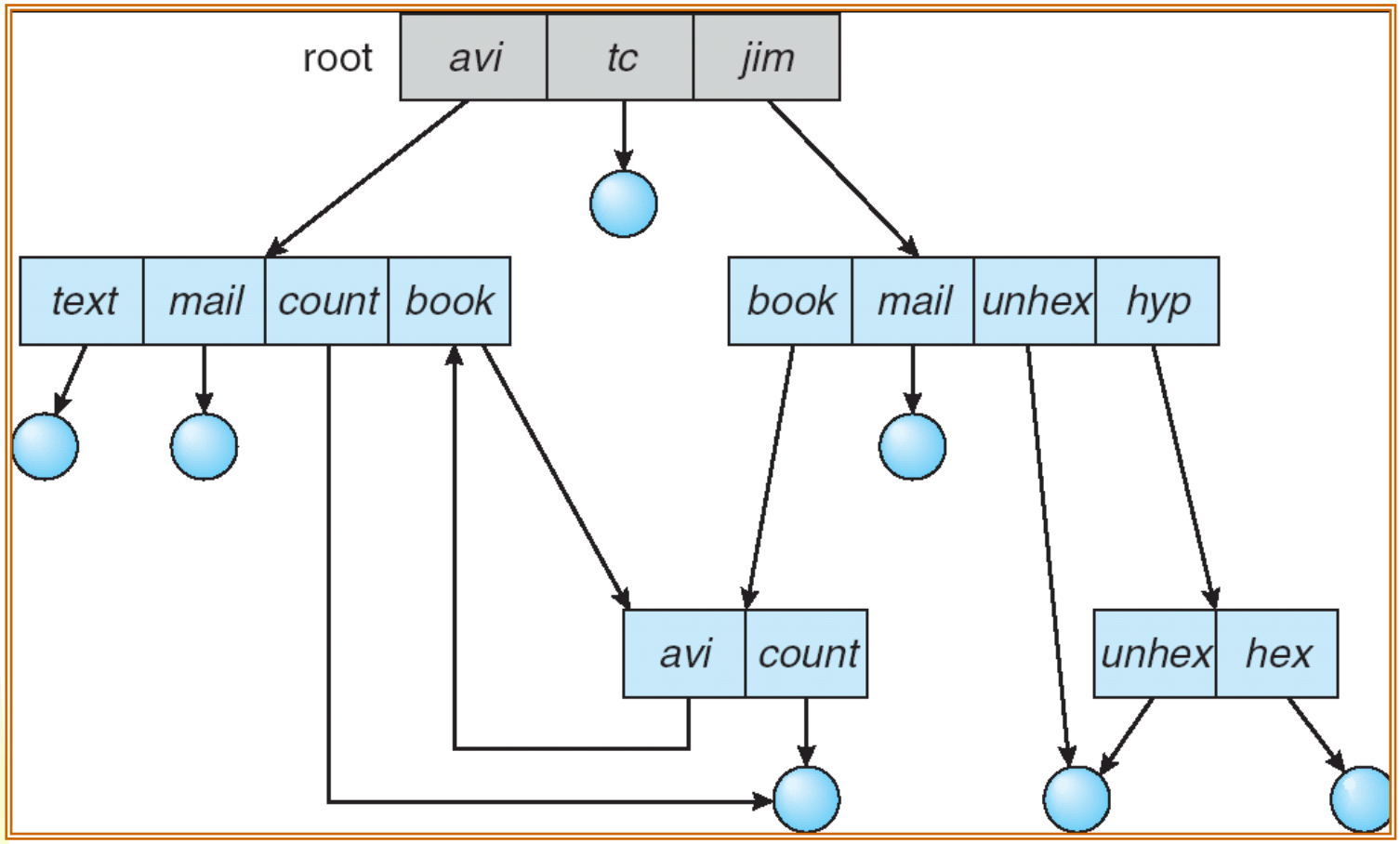
- Idea: **Have shared subdirectories and files**



Acyclic-Graph Directories (Cont.)

- **How do we achieve sharing?**
  - Have a special directory entry, not containing the file attributes, but containing the name of another file
    - Called symbolic (soft) link
  - Having two different directory entries that reference the same file attribute information (i.e. the file control block – FCB)
    - Called hard link (FCB contains a count of links to the file)
- **What happens when a file being referenced by a symbolic link is deleted?**
  - Dangling pointer
- **What happen when a file referenced by a hard link is deleted?**
  - The link is deleted, other directory entries can still be used via the remaining hard links

General Graph Directory



General Graph Directory (Cont.)

- **Adding links to subdirectory leads to many complications, we really want to avoid it**
  - The tree-structured (easily searched) in the directory is destroyed
    - When searching the directory, can search sub-directories multiple times.
  - Cycles in the graph can occur.
    - When searching the directory, can end up with looping (infinite searching).
  - When a link to a directory is removed, how to release the files contained in the subdirectory (link count is non zero)
- **But how to allow or omit cycles?**
  - Allow only links to files, not subdirectories
  - Every time a new link to a subdirectory is added use a cycle detection algorithm to determine whether it is OK
    - Complex and costly.
  - Allow only symbolic links to subdirectories and do not follow symbolic links
  - Use garbage collection to clean up the directory when a subdirectory is deleted (this is expensive).

Combining Several File Systems

File system

- Directory tree residing on the particular device/partition

Why combining?

- Several Hard Disks partitions, floppy/ZIP disks, CDROM, network disks
- Uniform view/access to them

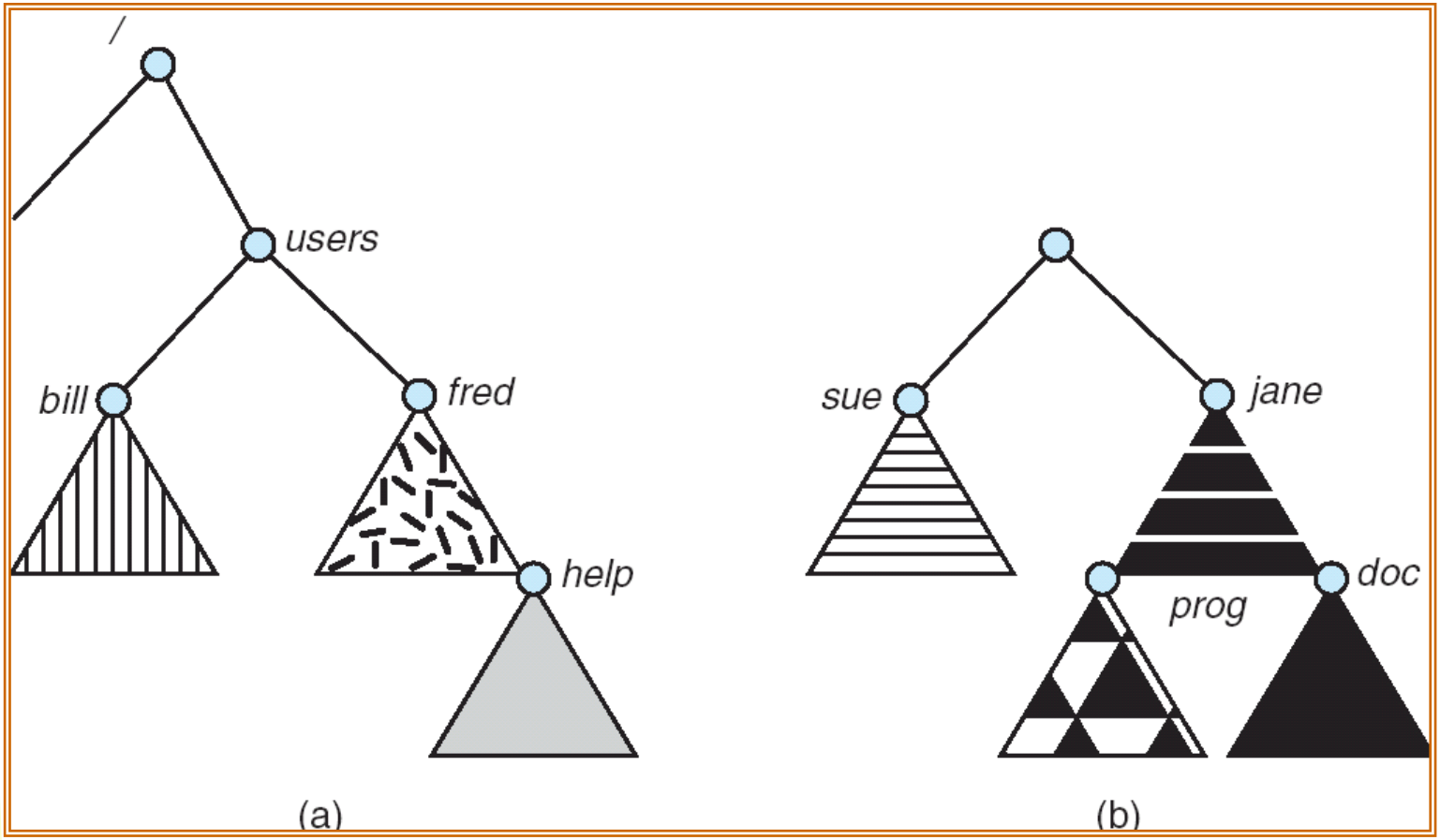
How?

- Mount a file system into particular node of the directory tree
- Windows: 2 level system – automatically mounts into drive letters
- Unix: explicit mount operation, can mount anywhere

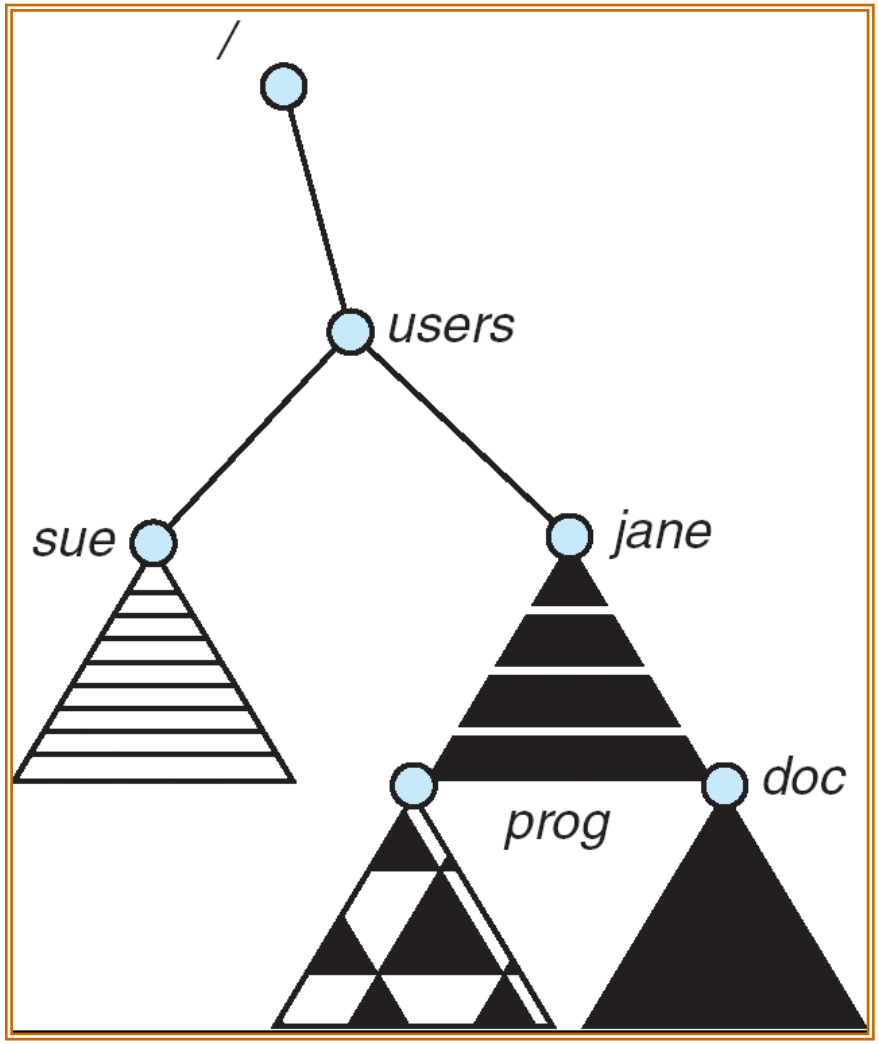
File System Mounting

- **A file system must be** mounted **before it can be accessed**
- **A unmounted file system (i.e. Fig. 11-11(b)) is mounted at a** mount point

(a) Existing. (b) Unmounted Partition



Mount Point



Protection

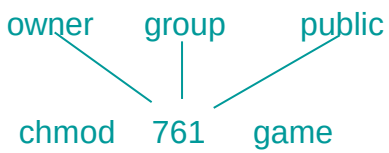
- **File owner/creator should be able to control:**
  - what can be done to the file
  - by whom
- **Types of access**
  - Read
  - Write
  - Execute
  - Append
  - Delete
  - List (list names and attributes in a subdirectory)

Access Lists and Groups - UNIX

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

			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

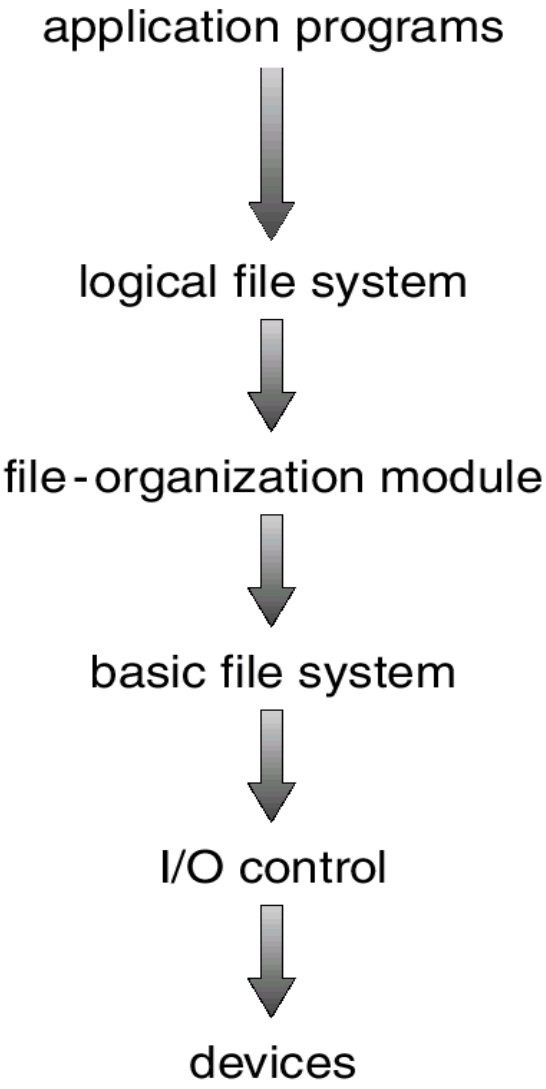
chgrp G game

Allocation methods

File system structures

- **File structure:** **two ways to view a file:**
  - space allocation unit
  - collection of related information
- **The file system resides in secondary memory: disks, tapes ...**
- **File control block: data structure containing information about a file**

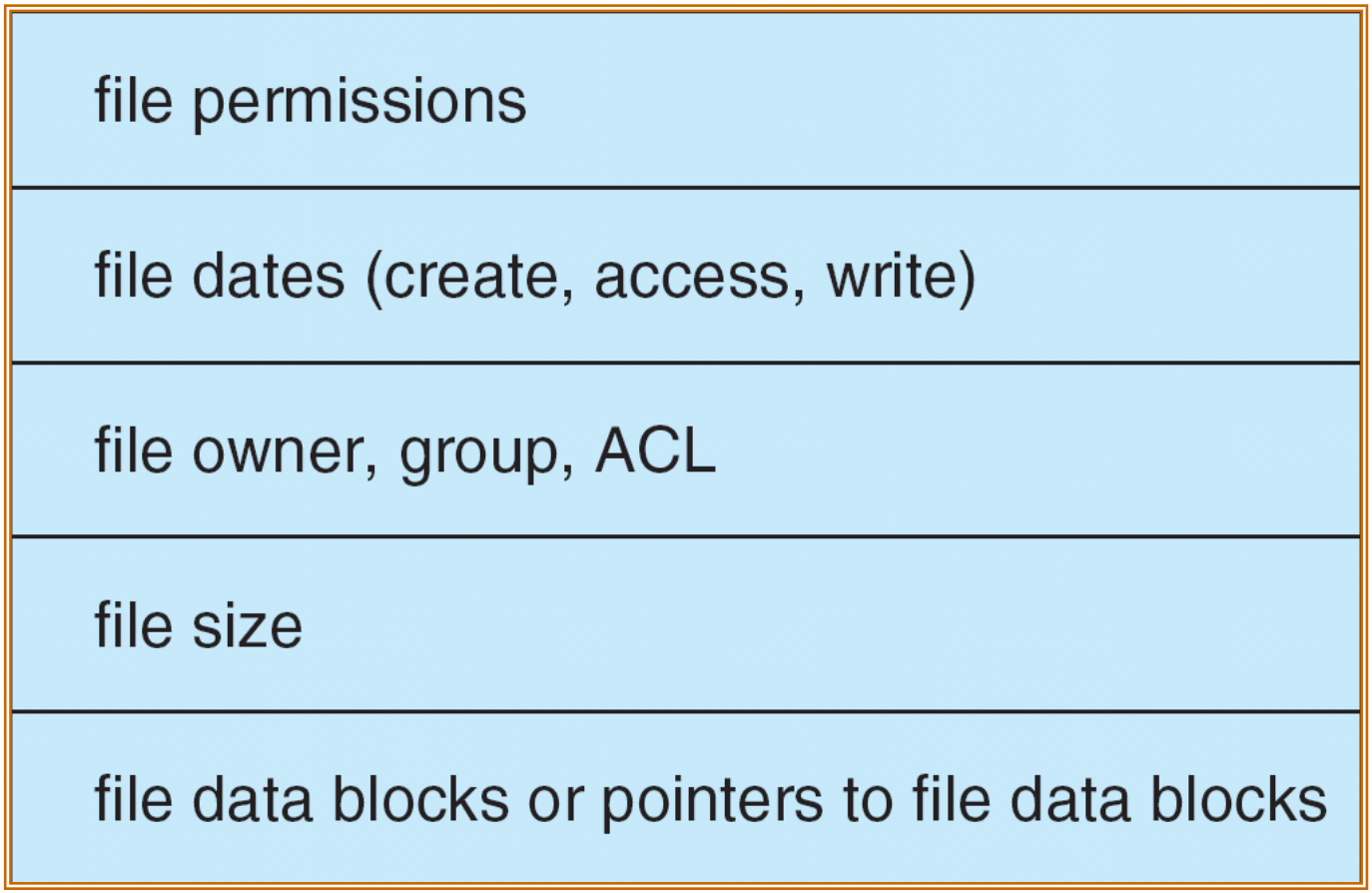
Layered file systems



Physical structure of files

- **The secondary memory is subdivided into blocks and each I/O operation is performed in units of blocks**
  - Tape blocks are of variable length, but disk blocks are of fixed length
  - On disk, a block is made up of a multiple of contiguous sectors (ex: 1, 2, or 4)
    - the size of a sector is usually 512 bytes
- **It is therefore necessary to insert the records in the blocks and extract them later**
  - Simple when each byte is a record by itself
  - More complex when the records have a structure (eg: "IBM main-frame")
- **Files are allocated in units of blocks. The last block is therefore rarely filled with data**
  - Internal fragmentation

A typical “File Control Block”



The directory implementation

- Linear list of file names with pointers to data blocks.
  - **Easy to program**
  - **Take a longer run time**
- Hash table - linear list with hashed data structure.
  - **Reduces directory search time**
  - **Collisions - situation where two filenames is hashed in the same place**
  - **Table size is fixed**

Three methods of file allocation

- Contiguous allocation
- Linked allocation
- Indexed allocation

Contiguous Allocation

Idea: **Each file occupies a set of contiguous blocks on the disk**

How much to allocate when a file is created?

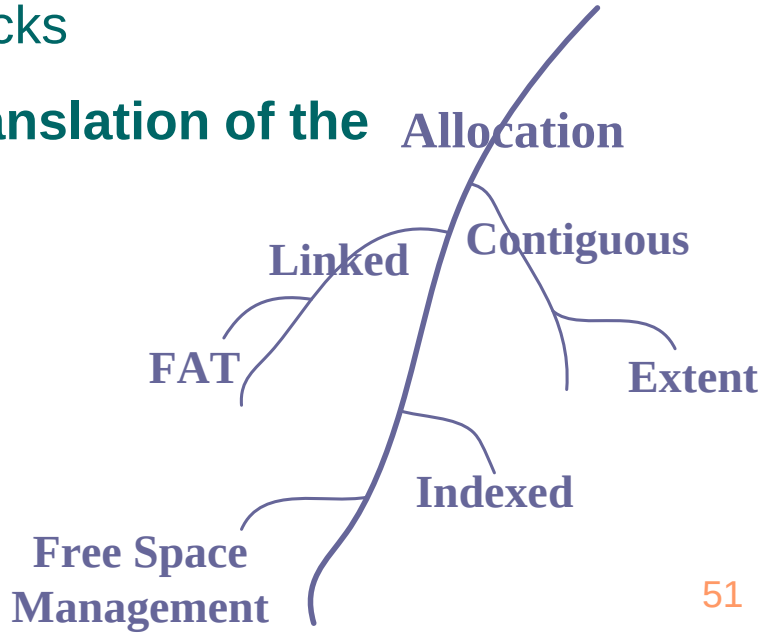
- difficult to guess in general
- too little – difficult to grow, to much – wasting space

How to grow a file?

- into preallocated free space, free adjacent blocks, relocate into bigger hole of free blocks

What information is needed to allow translation of the logical to physical address?

- initial block and size, in the FCB



Contiguous Allocation

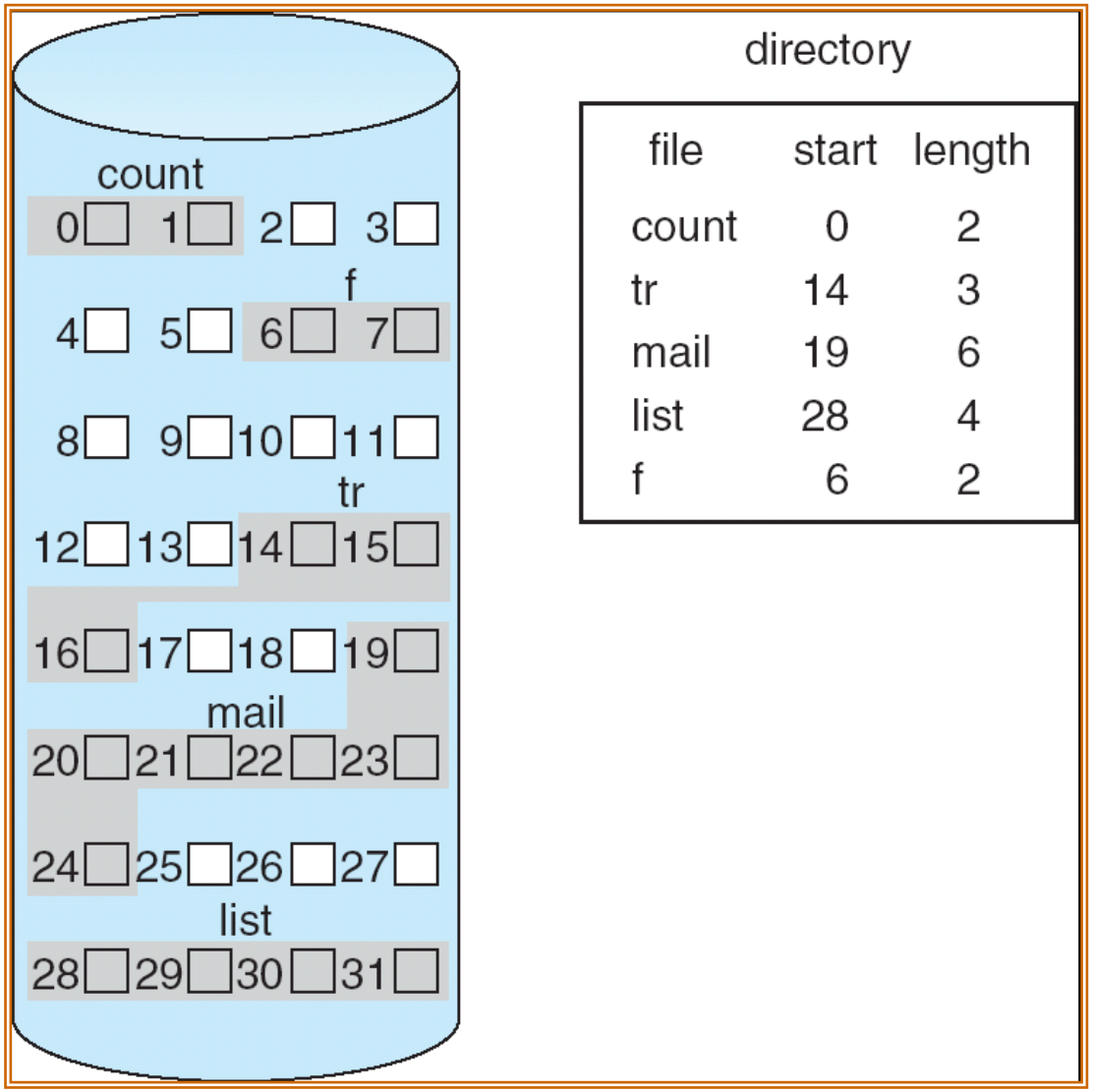
Benefits:

- **Simple to implement**
  - ph. block = start block + logical address/block size
  - offset = logical address modulo block size
- **Efficient random access**
- **Good locality of reference**

Drawbacks:

- **Wasteful of space (fragmentation)**
- **Files cannot grow**
- **Cannot easily add to data in the middle of the file**
- **Needs compaction**

Contiguous Allocation Example



Linked Allocation

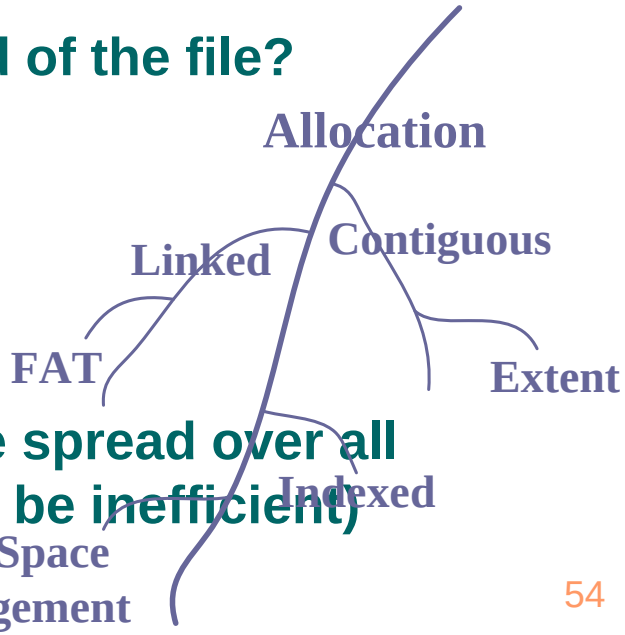
Idea: **Each block of the file contains a pointer pointing to the next block of that file. The directory entry of the file points to the first block of the file.**

Benefits:

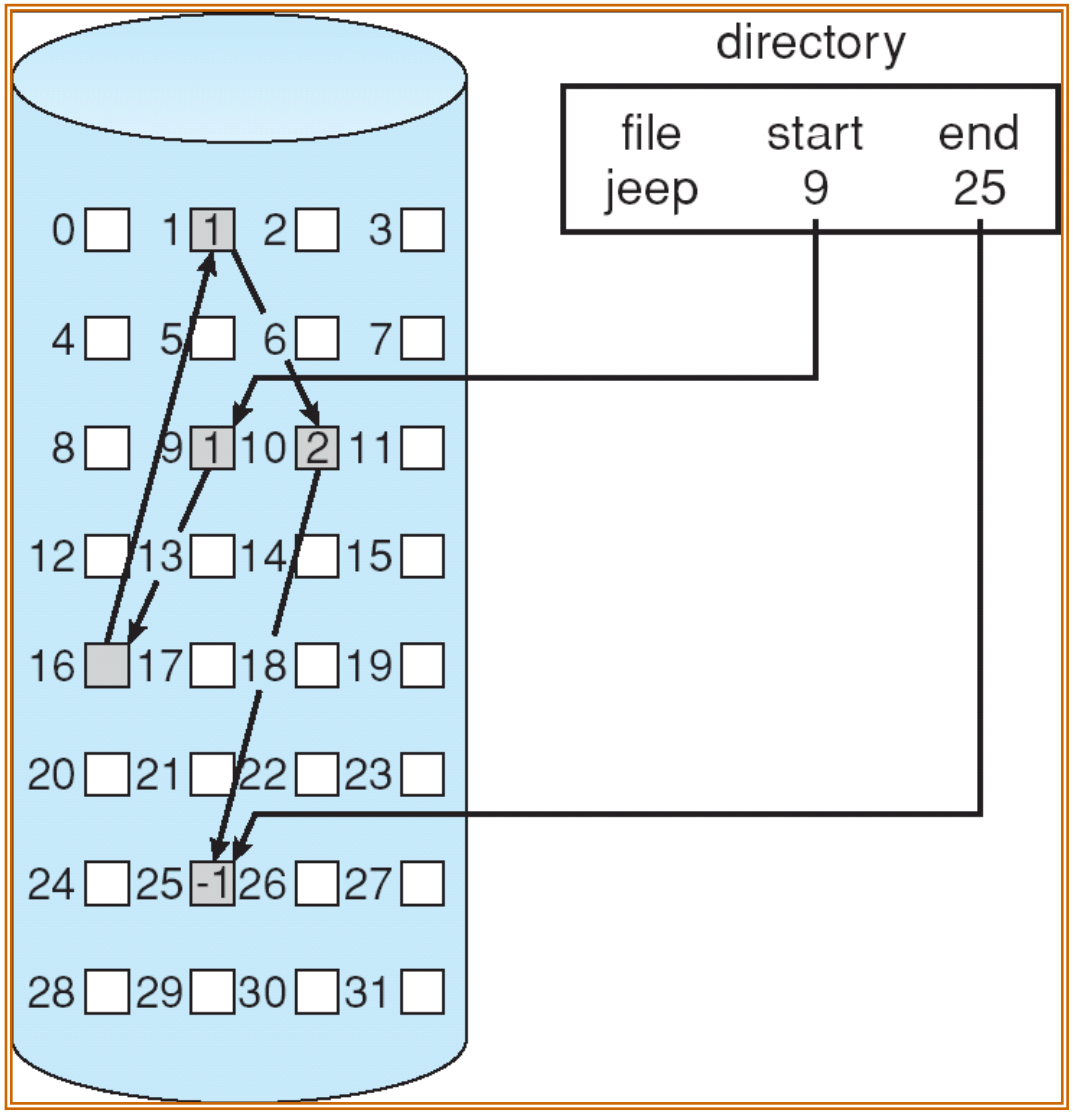
- **simple**
- **no external fragmentation**
- **easy to grow files (a free block anywhere on the disc is linked to the end of the file)**

Drawbacks:

- **What happens if you want to reach the end of the file?**
  - Has to read it all, to follow the pointers
- **Very inefficient random access**
- **Wasted space for pointers**
  - Using larger bloc size helps
- **Poor locality of reference (the file might be spread over all disc, so even reading it sequentially might be inefficient)**

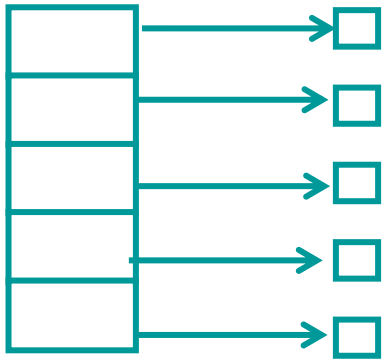


Linked Allocation

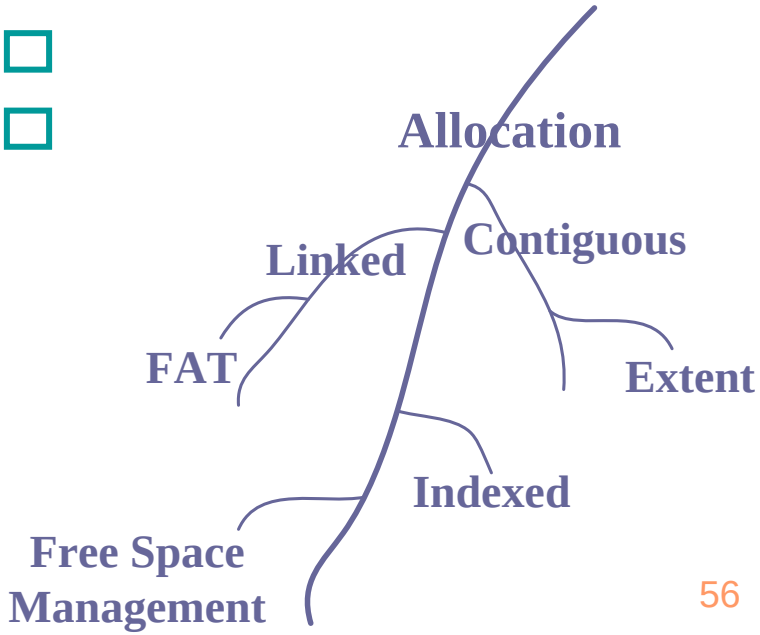


Indexed allocation – similar to paging

- All pointers to blocs (bloc numbers) are placed in a table (index bloc).

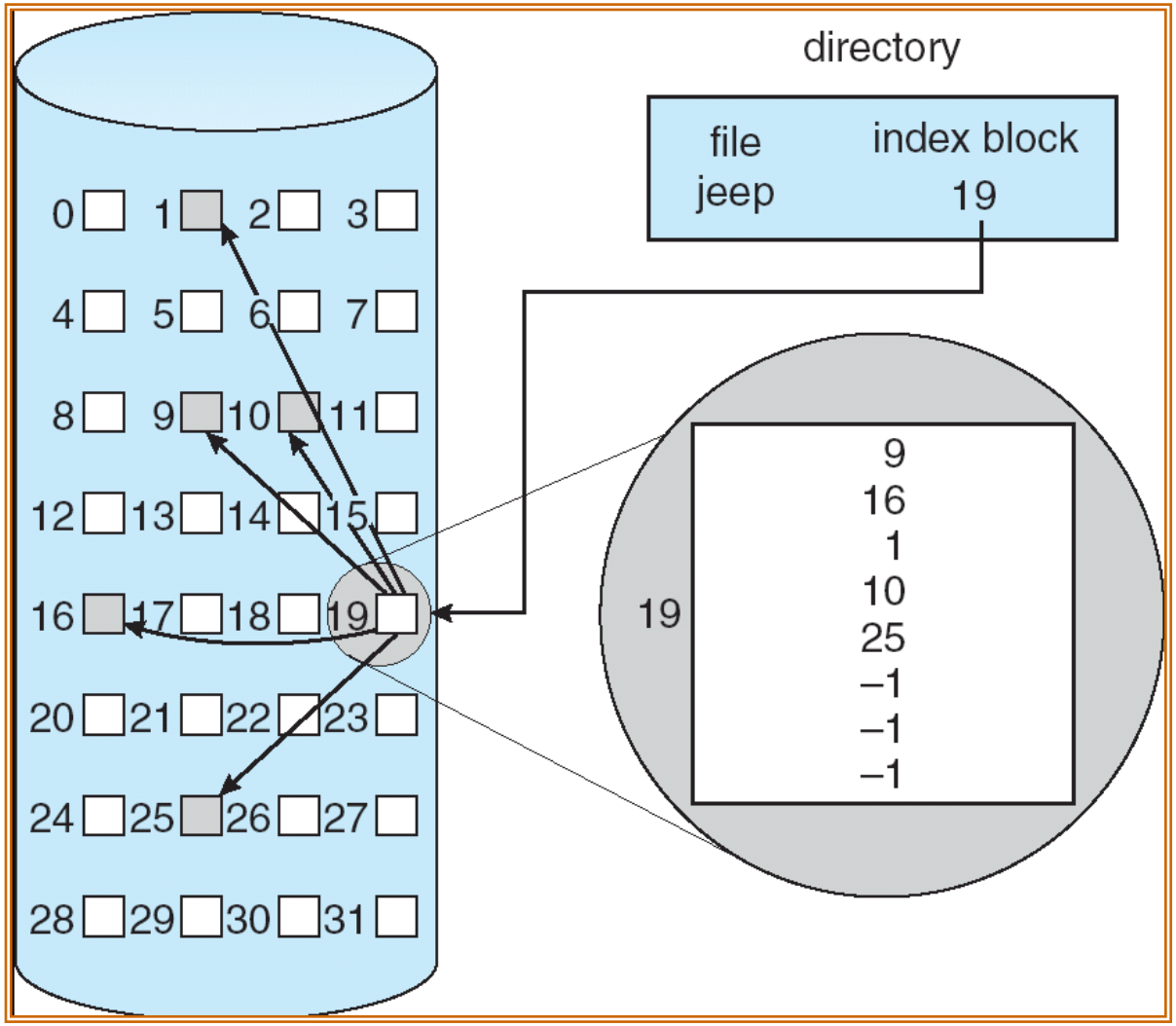


index table



Indexed Allocation

Idea: **Use an *index block* to store all the block pointers of the file**



Indexed Allocation (Cont.)

Benefits:

- **easy random access**
- **no external fragmentation**

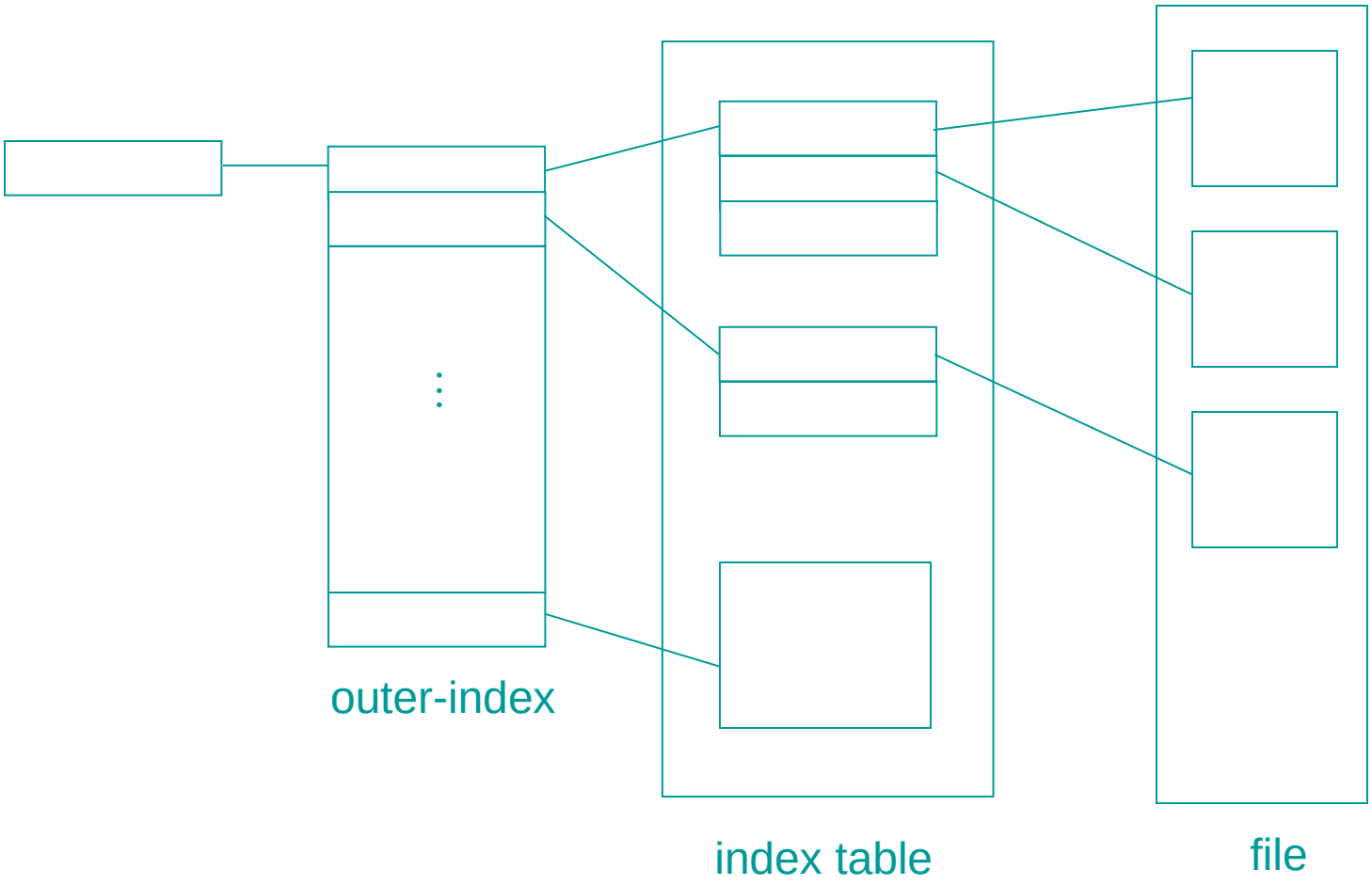
Drawbacks:

- **need index table blocks**
- **poor locality of reference**

What if the file needs more blocks than the index block can reference?

- **i.e. 512 byte block with 256 entries, 256x512 ⇒ maximum files size of 128 Kbytes**
- **Implement Index Table as**
  - linked list of blocks
    - poor for random access
  - hierarchical index tables
    - too much overhead for small files
  - combined scheme

Hierarchical Indexed Allocation



Implementation of directories

- **Linear list of file names with pointers to data blocks**
  - sequential access
  - easy to program
  - time needed to browse list
- **Hash tables: calculated tables**
  - fast search time
  - collision problem
  - fixed table dimension

Efficiency and performance

- **Efficiency depends on:**
  - directory allocation and organization method
- **To increase performance:**
  - Make efficient access to frequently visited blocks
    - Dedicate memory buffers that contain the image of the most frequently used info
  - Optimize sequential access if it is often used: free behind and read ahead


Thank You!
Ευχαριστώ
ขอบคุณ
Vielen Dank
Merci
Teşekkürler
DMnvwd
شكراً
متشكرم
Gracias
Grazie
Bedankt
Dankie
THANK YOU
Köszönettel
Obrigado!
شکریا
Díky
謝謝
GADDA GUEY
WAD MAHAD
SAN TAHAY
감사합니다
Urakoze