Operating System

Tutorial 11

File System

File systems

- Files
- Directories & naming
- File system implementation
- Example file systems

File System

- A *filesystem* is the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk.
- The word is also used to refer to a partition or disk.
- Before a partition or disk can be used as a filesystem, it needs to be initialized, and the bookkeeping data structures need to be written to the disk. This process is called *making a filesystem*.

Long-term information storage

- Must store large amounts of data
  - Gigabytes -> terabytes -> petabytes
- Stored information must survive the termination of the process using it
  - Lifetime can be seconds to years
  - Must have some way of finding it!
- Multiple processes must be able to access the information concurrently

Naming files

- Important to be able to *find* files after they're created
- Every file has at least one name
- Name can be
  - Human-accessible: "foo.c", "my photo", "Go Panthers!", "Go Banana Slugs!"
  - Machine-usable: 4502, 33481
- Case may or may not matter
  - Depends on the file system
- Name may include information about the file's contents
  - Certainly does for the user (the name should make it easy to figure out what's in it!)
  - Computer may use part of the name to determine the file type

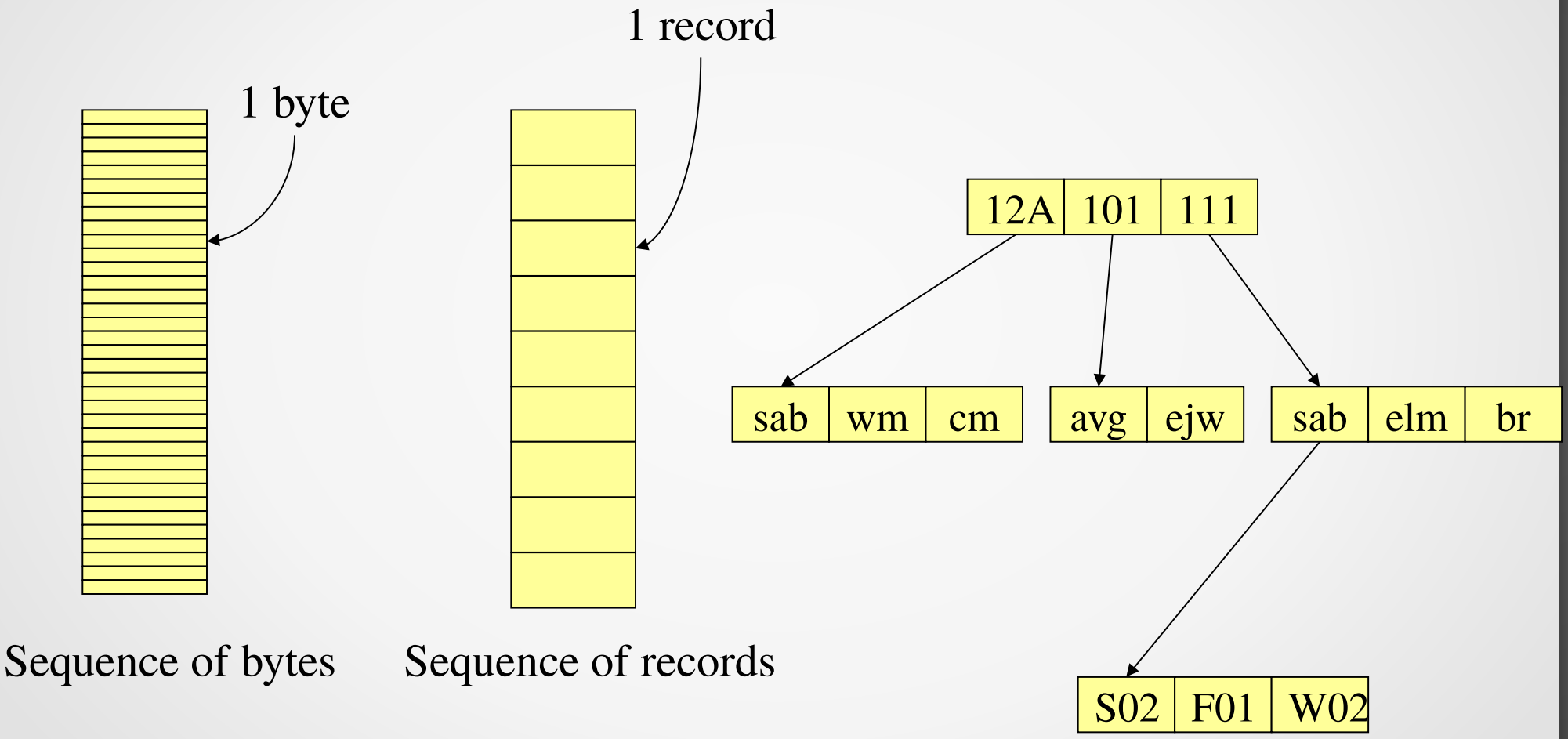
• Typical file extensions

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

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:
  - Operating system
  - Program / programmer

• File structures



File attributes

Attribute	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of the person who created the file
Owner	Current owner
Read-only flag	0 for read/write; 1 for read only
Hidden flag	0 for normal; 1 for do not display in listings
System flag	0 for normal files; 1 for system file
Archive flag	0 for has been backed up; 1 for needs to be backed up
ASCII/binary flag	0 for ASCII file; 1 for binary file
Random access flag	0 for sequential access only; 1 for random access
Temporary flag	0 for normal; 1 for delete file on process exit
Lock flags	0 for unlocked; nonzero for locked
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Date and time the file was created
Time of last access	Date and time the file was last accessed
Time of last change	Date and time the file has last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to

File operations

- Create: make a new file
- Delete: remove the file
- Open: prepare a file to be accessed
- Close: end accessing a file
- Read: get data from a file
- Write: put data to a file
- Append: like write, but only at the end of the file
- Seek: move the “current” pointer elsewhere in the file
- Get attributes: retrieve information about a file
- Set attributes: modify file information
- Rename: change a file name

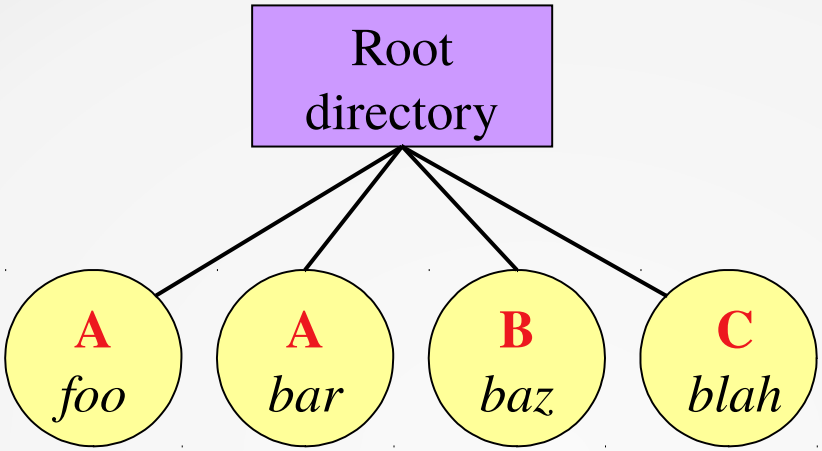
Accessing a file

- Sequential access
  - Read all bytes/records from the beginning
  - Cannot jump around
    - May rewind or back up, however
  - Convenient when medium was magnetic tape
  - Often useful when whole file is needed
- Random access
  - Bytes (or records) read in any order
  - Essential for database systems
  - Read can be …
    - Move file marker (seek), then read or …
    - Read and then move file marker

Directories

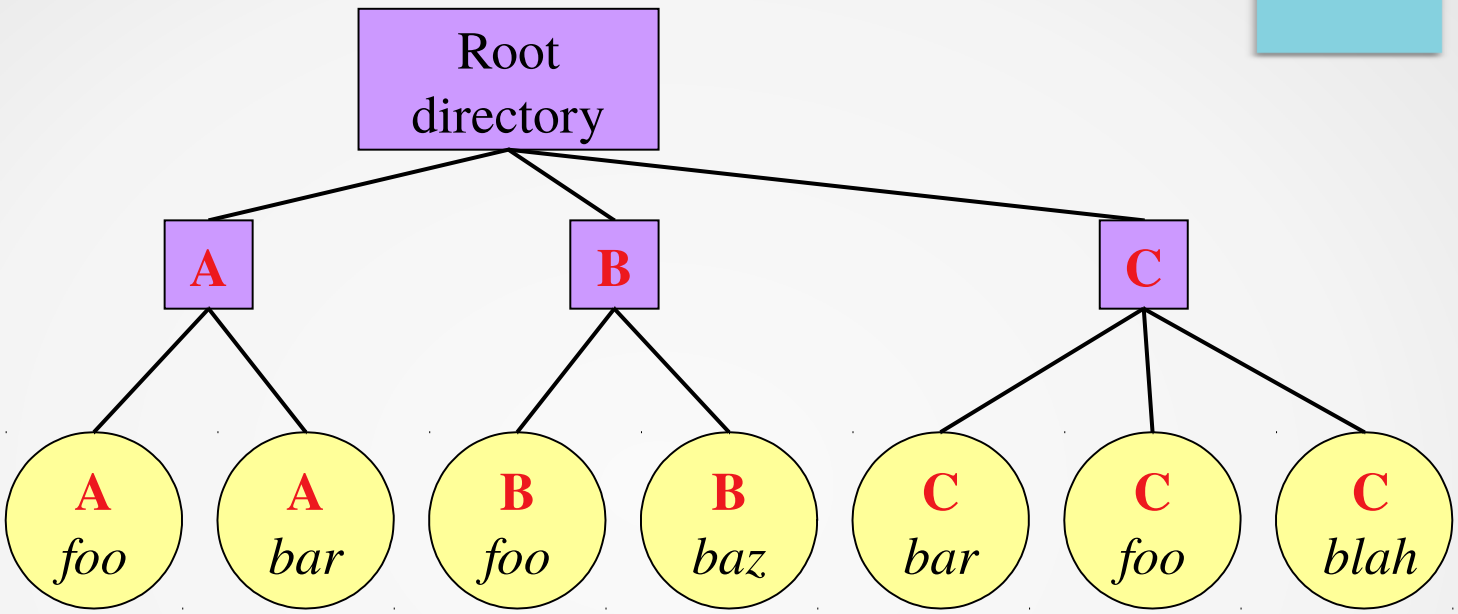
- Naming is nice, but limited
- Humans like to group things together for convenience
- File systems allow this to be done with *directories* (sometimes called *folders*)
- Grouping makes it easier to
  - Find files in the first place: remember the enclosing directories for the file
  - Locate related files (or just determine which files are related)

• Single-level directory systems



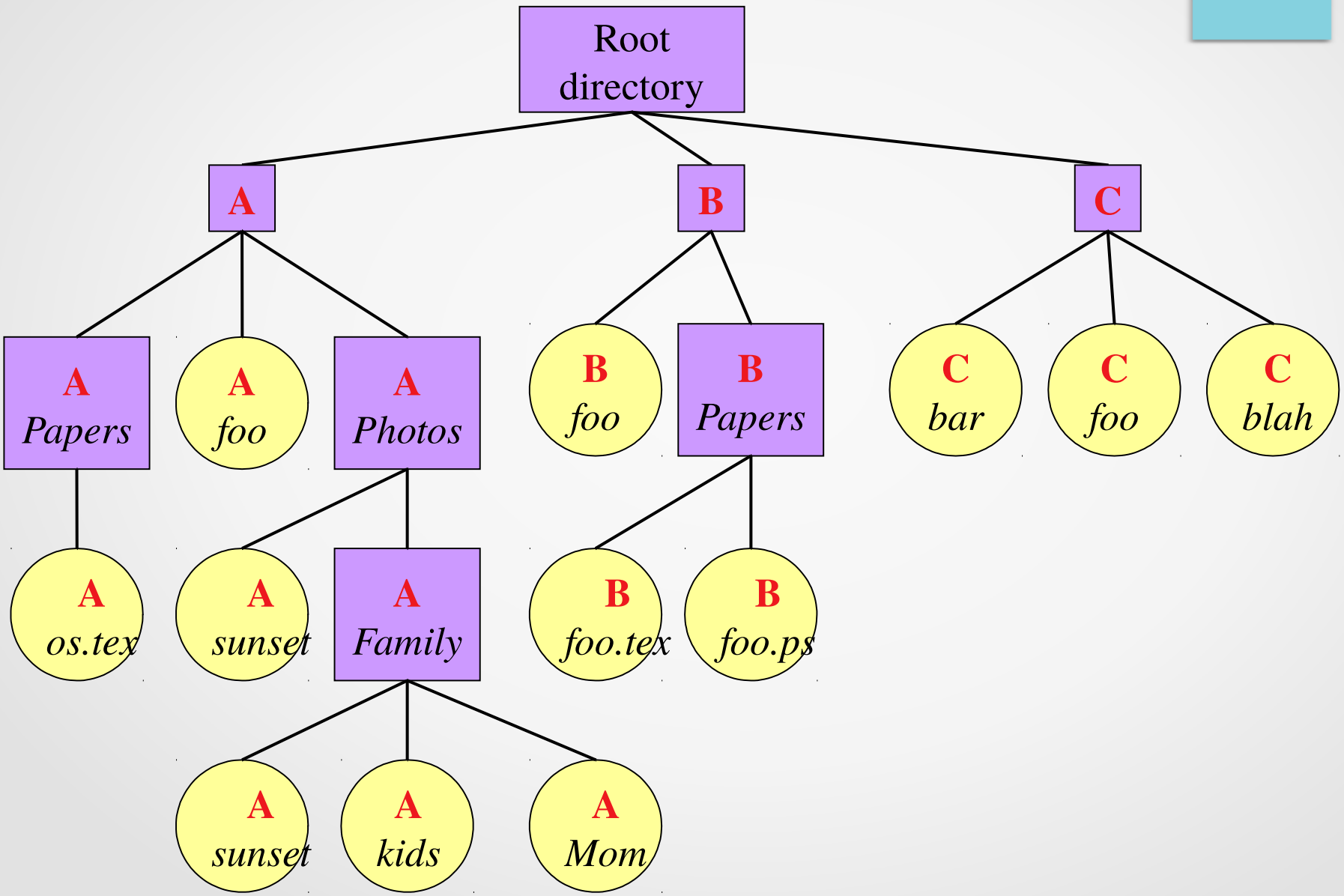
- One directory in the file system
- Example directory
  - Contains 4 files (*foo*, *bar*, *baz*, *blah*)
  - owned by 3 different people: A, B, and C (owners shown in red)
- Problem: what if user B wants to create a file called *foo*?

• Two-level directory system

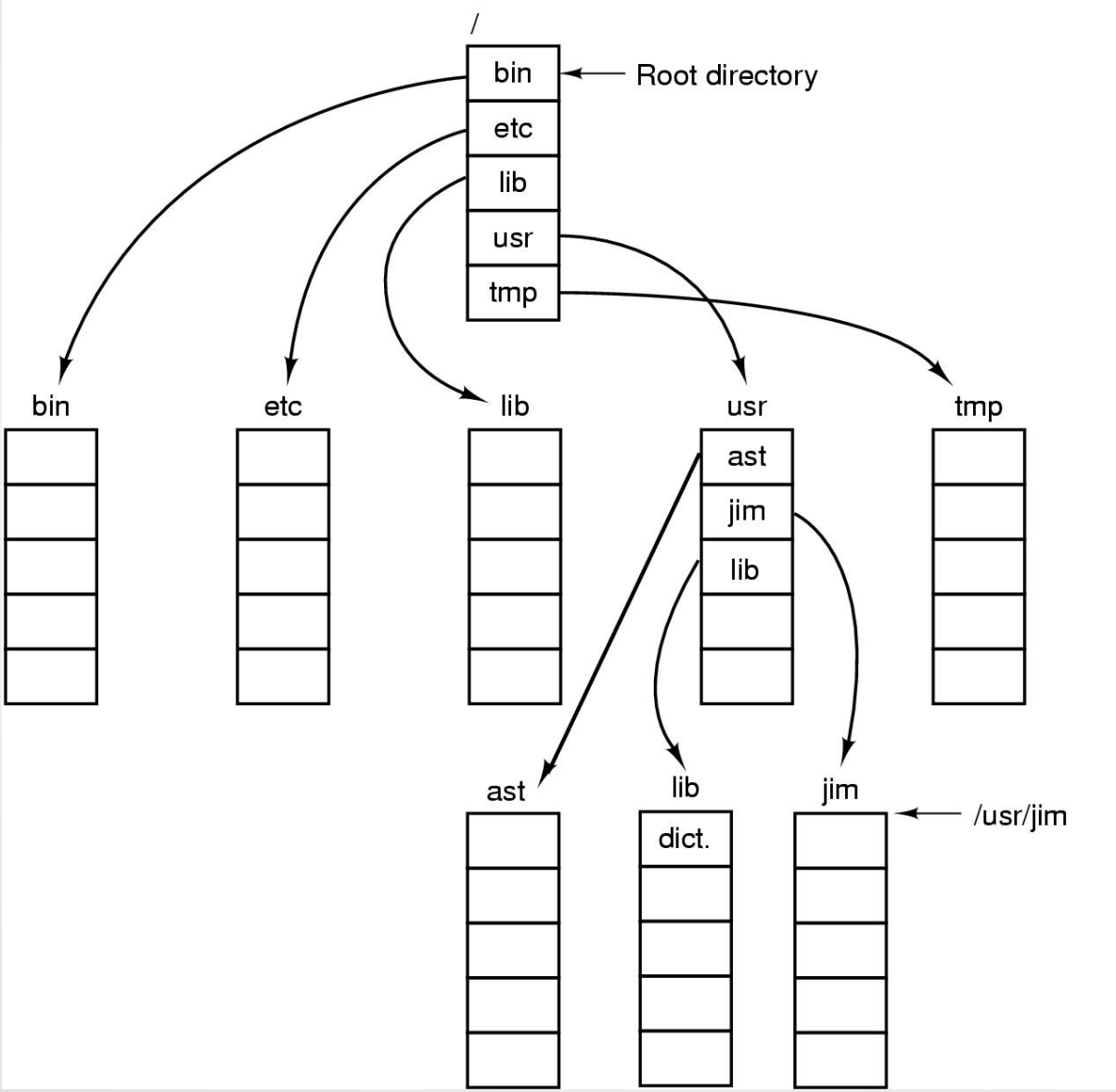


- Solves naming problem: each user has her own directory
- Multiple users can use the same file name
- By default, users access files in their own directories
- Extension: allow users to access files in others' directories

• Hierarchical directory system

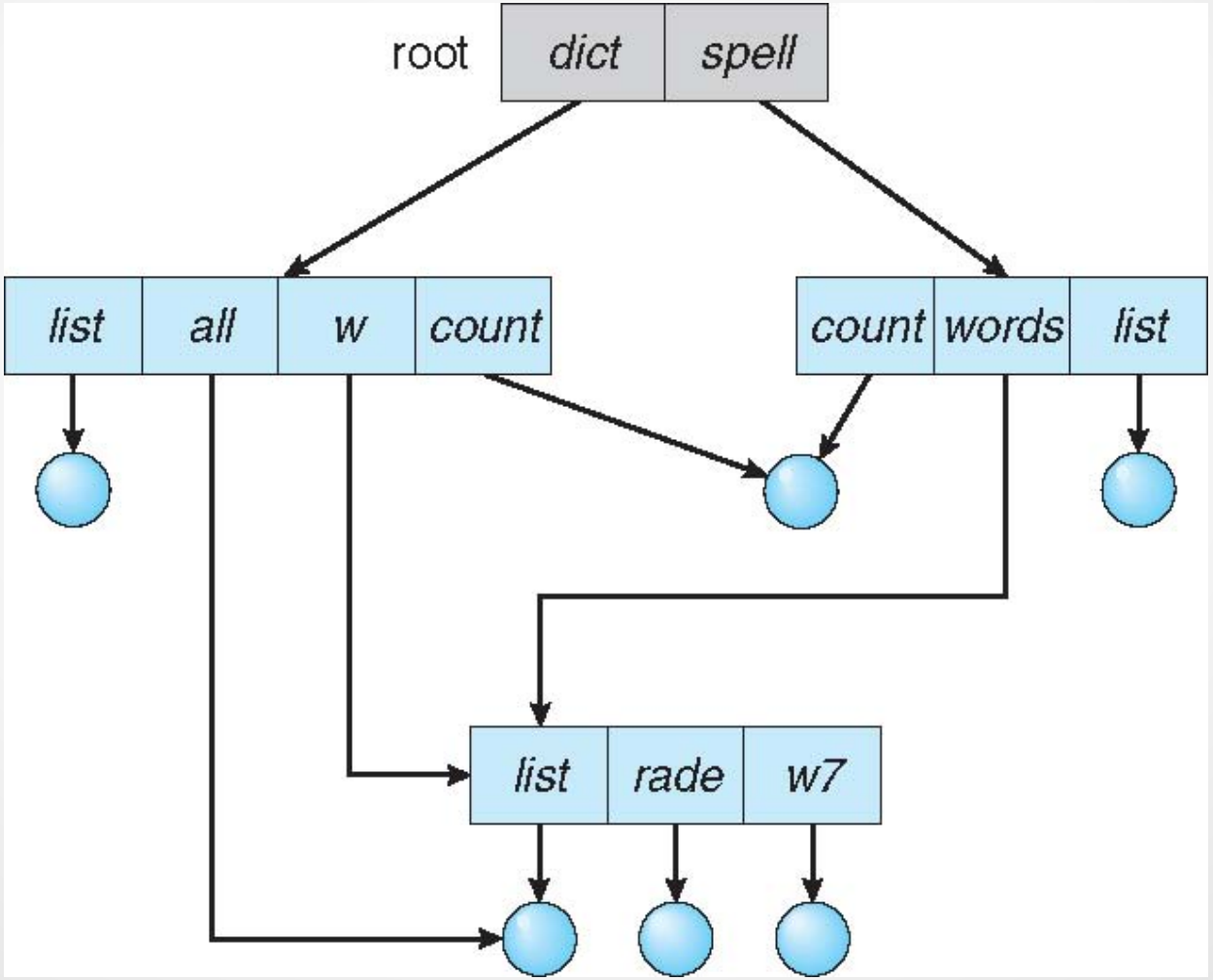


• Unix directory tree



Acyclic-Graph Directories

- Have shared subdirectories and files

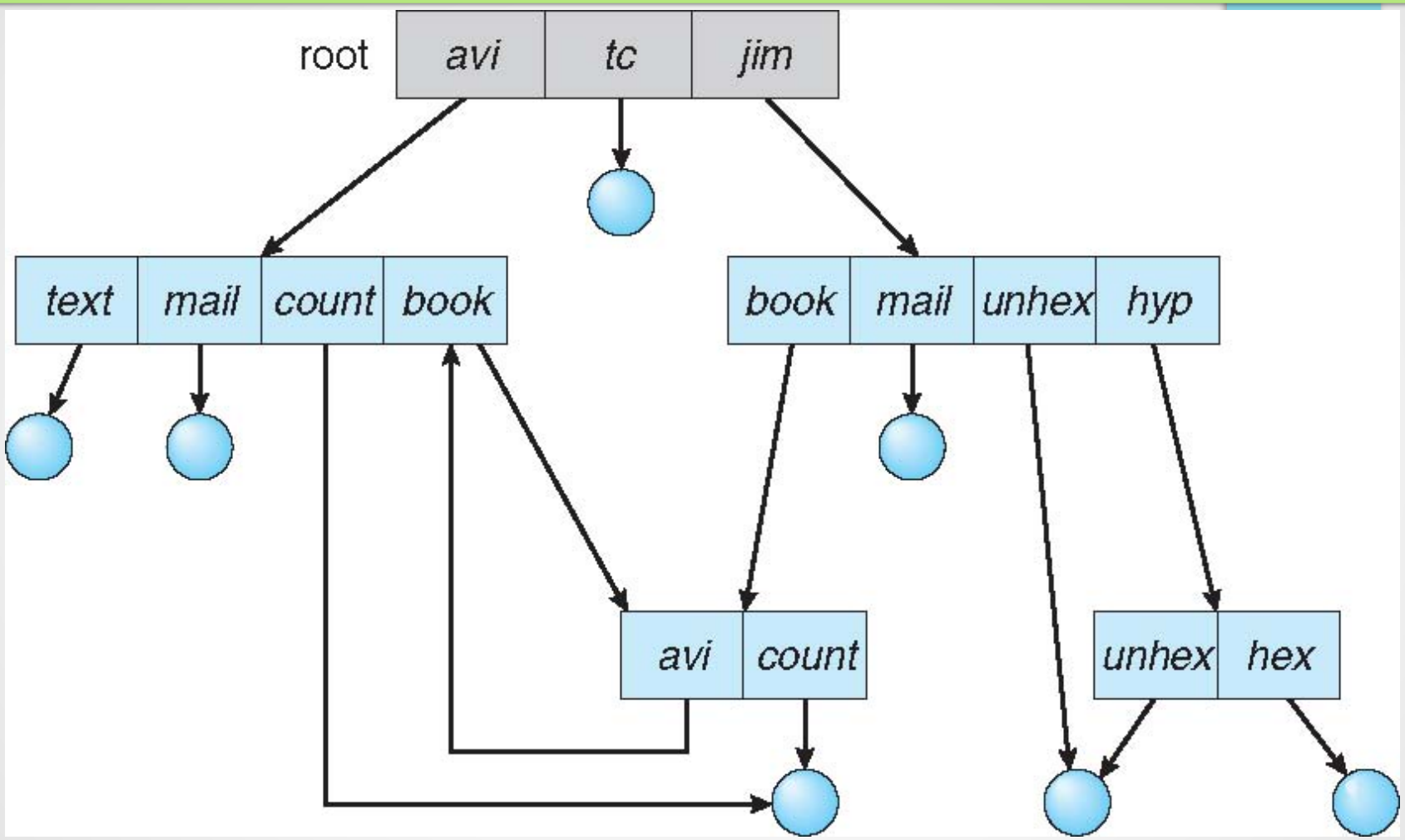


Acyclic-Graph Directories (Cont.)

- Adds ability to directly share directories between users
  - But can now have multiple absolute paths to the same file
- Two different names (aliasing)
- If *dict* deletes *list* ⇒ dangling pointer

  Solutions:
  - Backpointers, so we can delete all pointers
    Variable size records a problem
  - Entry-hold-count solution

General Graph Directory



Operations on directories

- Readdir: read a directory entry
- Mkdir: make a new directory
- Rmdir: remove a directory (usually must be empty)
- Opendir: open a directory to allow searching it
- Closedir: close a directory (done searching)
- Link: create a new entry in a directory to link to an existing file
- Unlink: remove an entry in a directory
  - Remove the file if this is the last link to this file
- Rename: change the name of a file
  - Similar to renaming a file

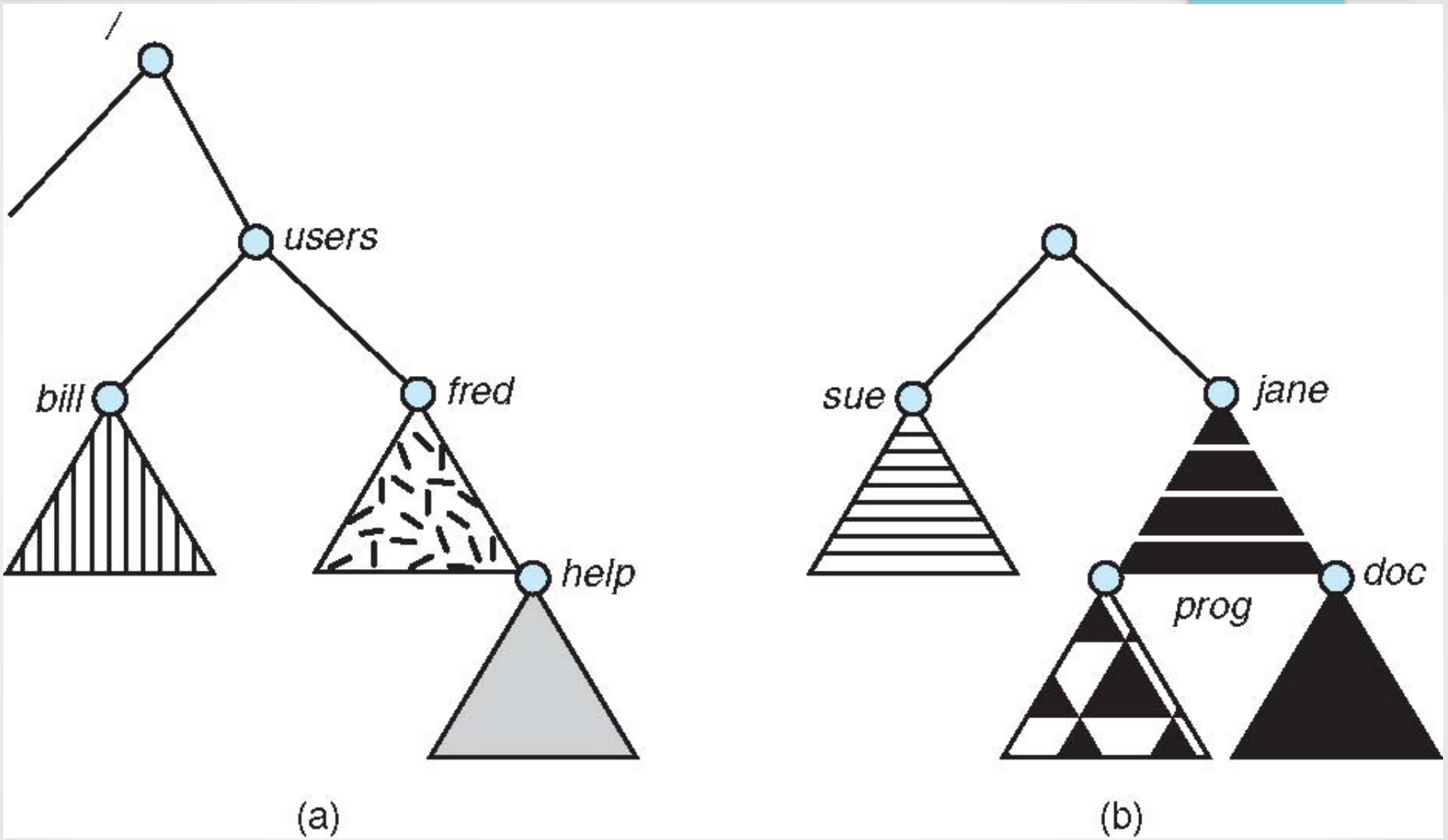
File system implementation issues

- How are disks divided up into file systems?
- How does the file system allocate blocks to files?
- How does the file system manage free space?
- How are directories handled?
- How can the file system improve…
  - Performance?
  - Reliability?

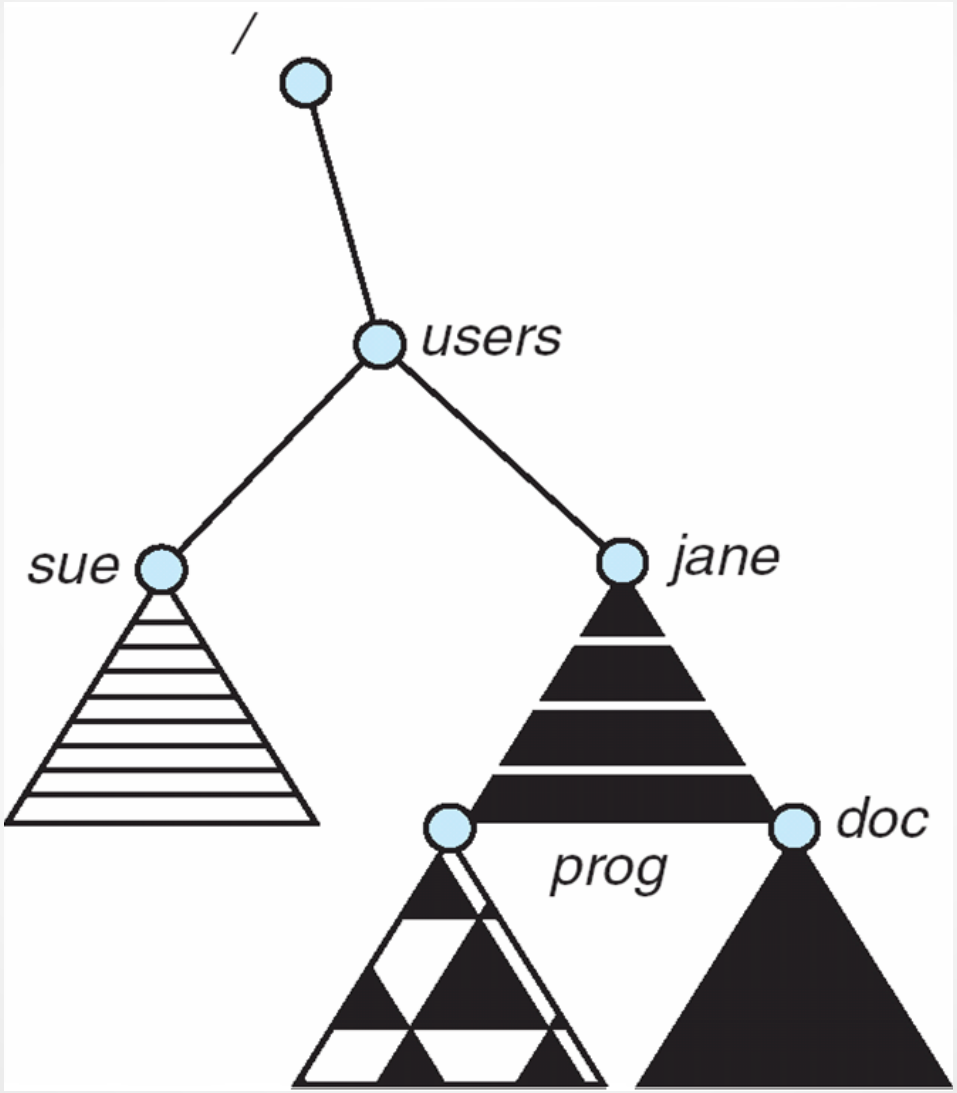
File System Mounting

- File system must be *mounted* before it can be available to processes on the system.
- Sometime, the directory structure may be built out of multiple volumes, which must be mounted to make them available within the file-system name space.
- Mounting is:
  - Privileged operation
  - First check for valid file system on volume
  - Kernel data structure to track mount points

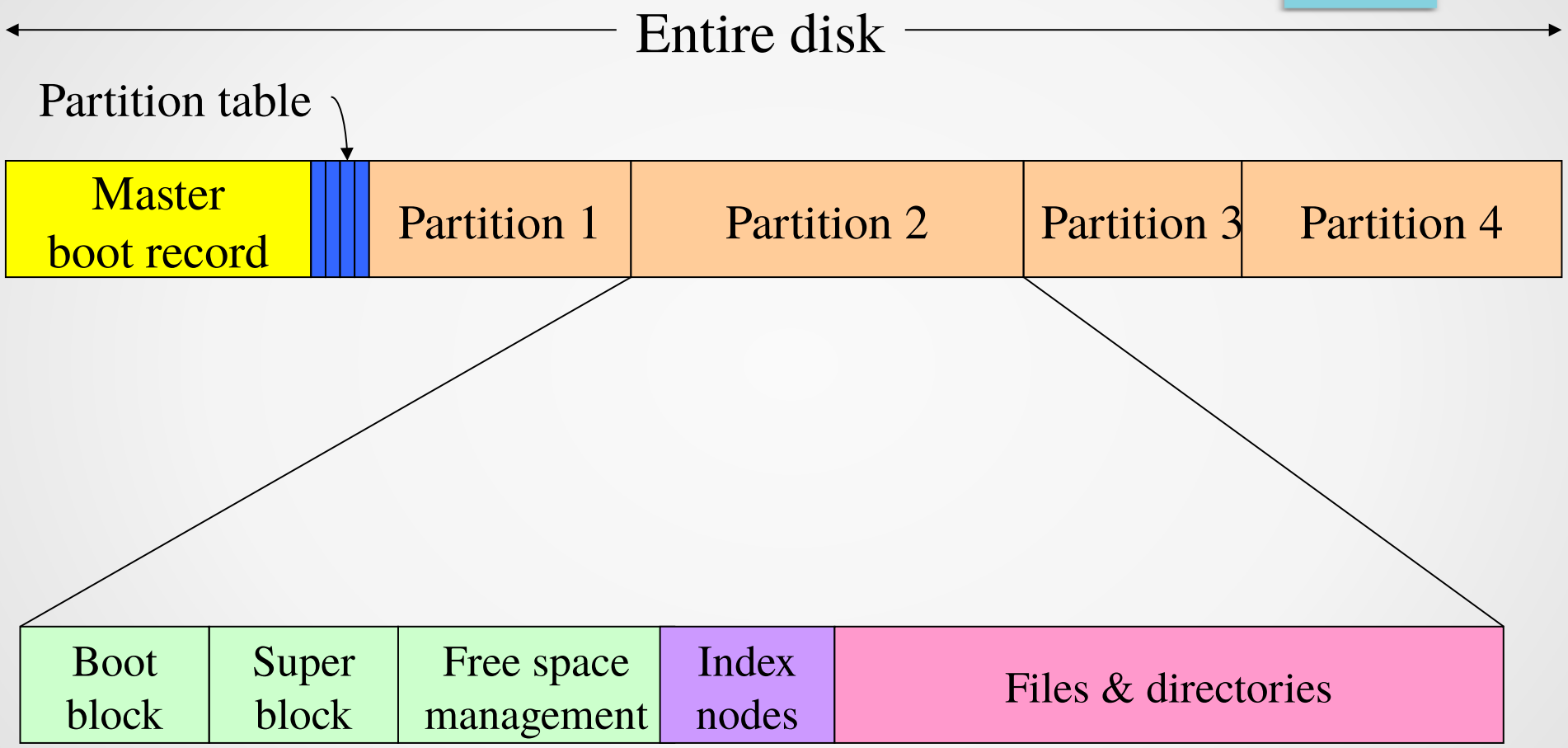
(a) Existing (b) Unmounted Partition



- Mount Point



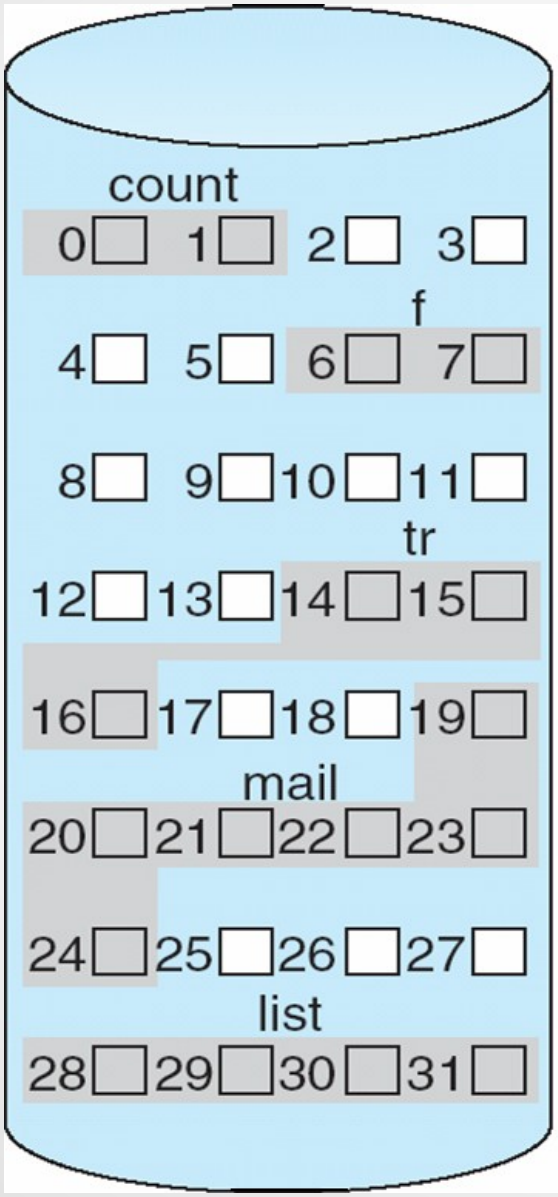
• Carving up the disk



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 of Disk Space



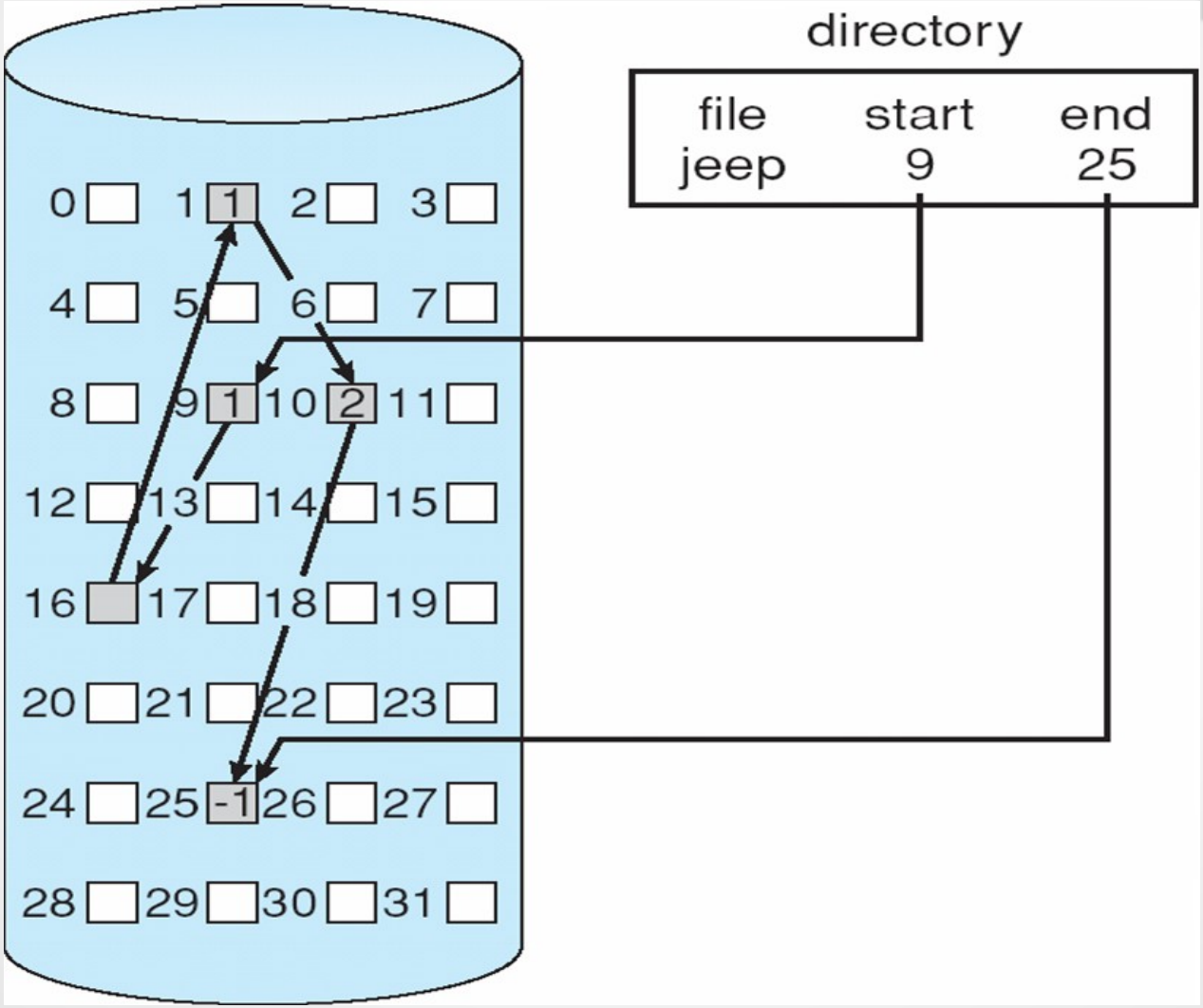
directory

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

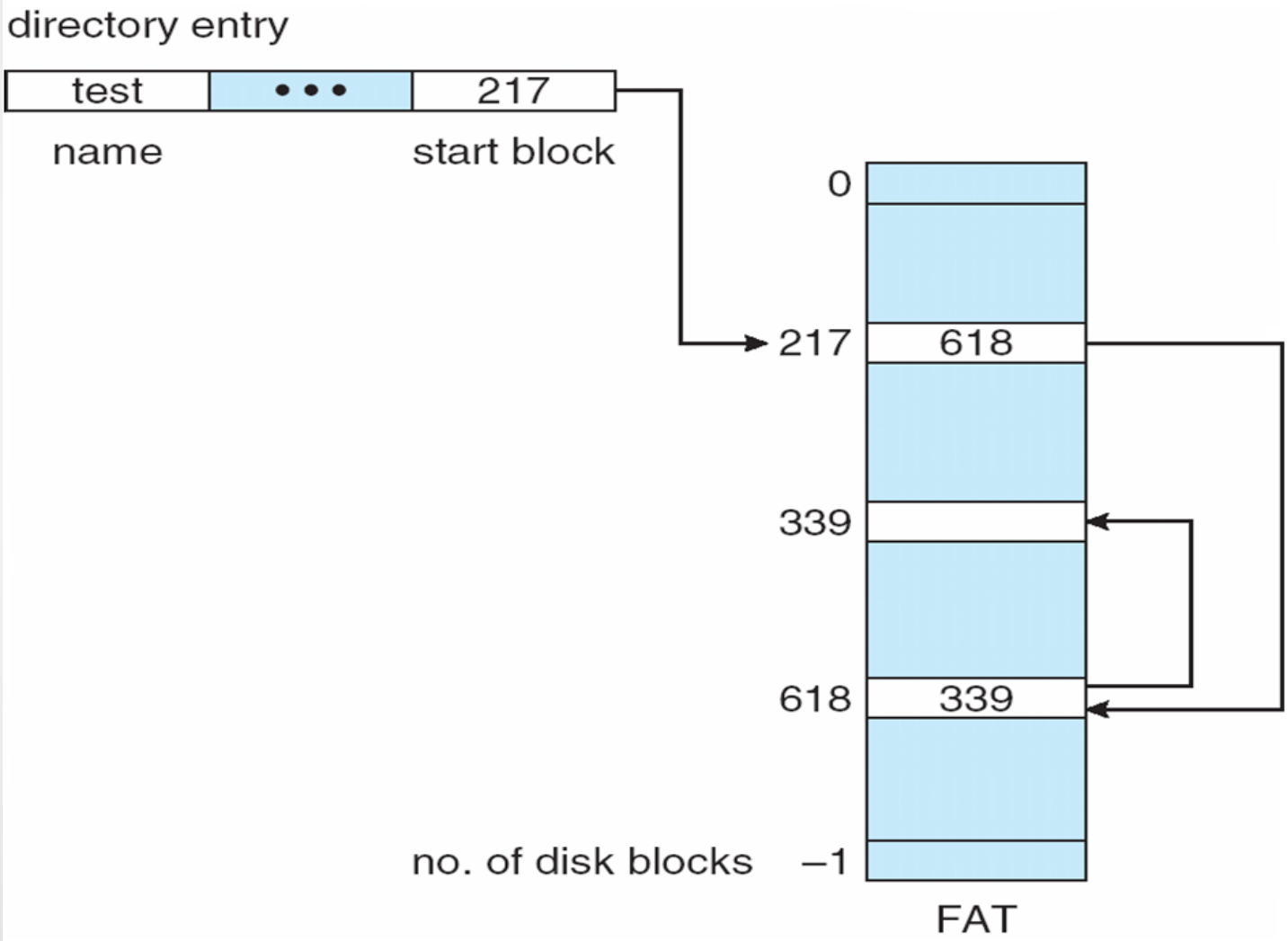
- Linked Allocation



FAT

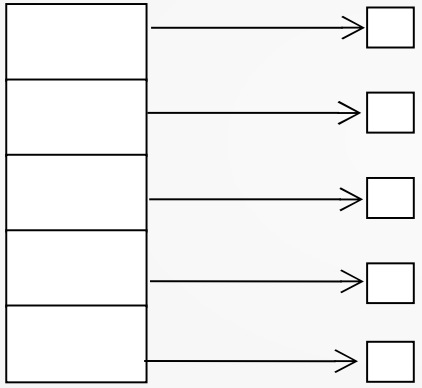
- FAT (File Allocation Table) variation
  - Beginning of volume has table, indexed by block number
  - Much like a linked list, but faster on disk and cacheable
  - New block allocation simple

File-Allocation Table



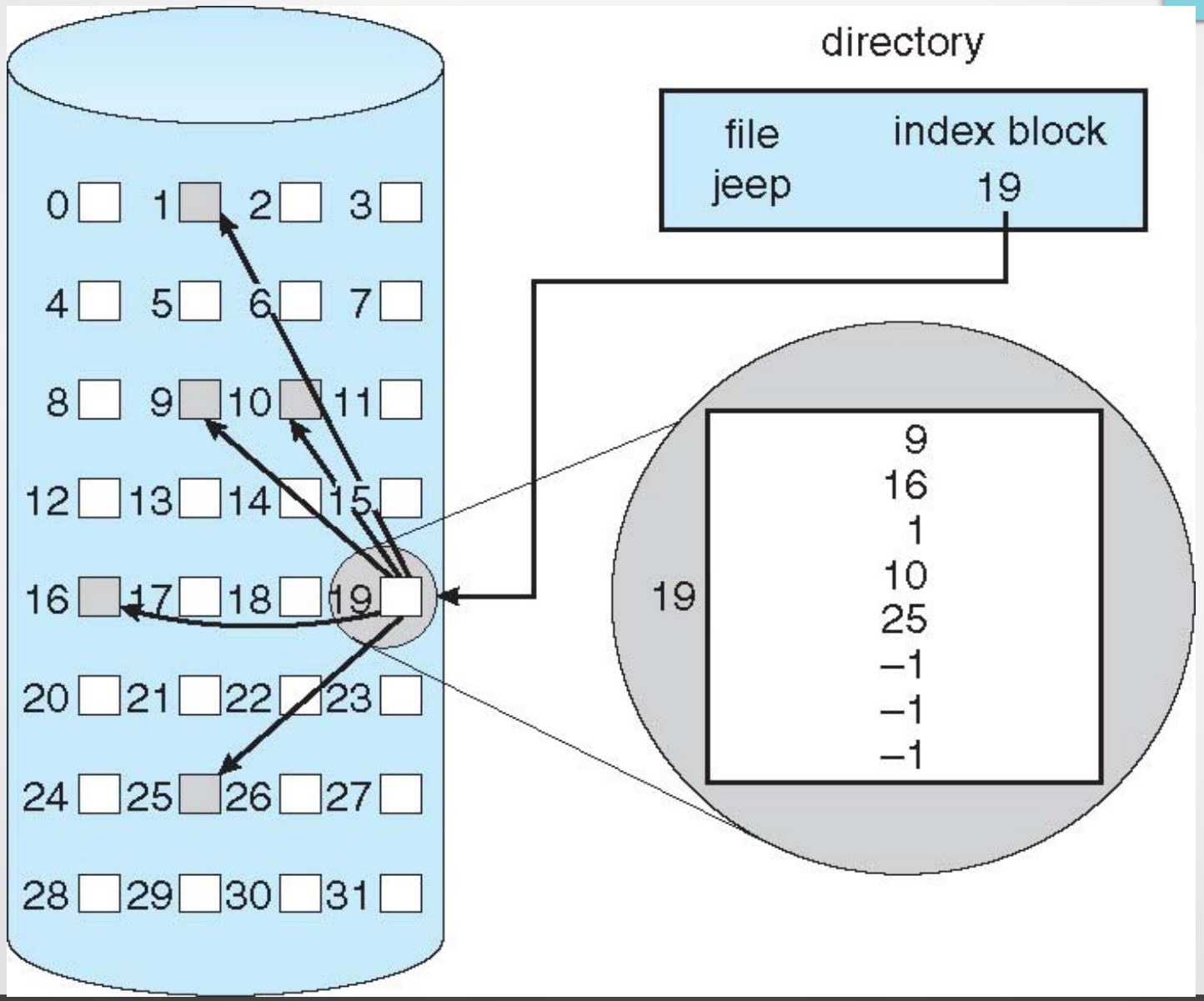
Allocation Methods - Indexed

- **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

• MS-DOS File Allocation Table

Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB