ECE437/CS481

M06A: FILE SYSTEMS

CHAPTER 11.1-11.3 & 12.1-12.3

Xiang Sun

The University of New Mexico

File Concept

- File are stored in contiguous logical address space

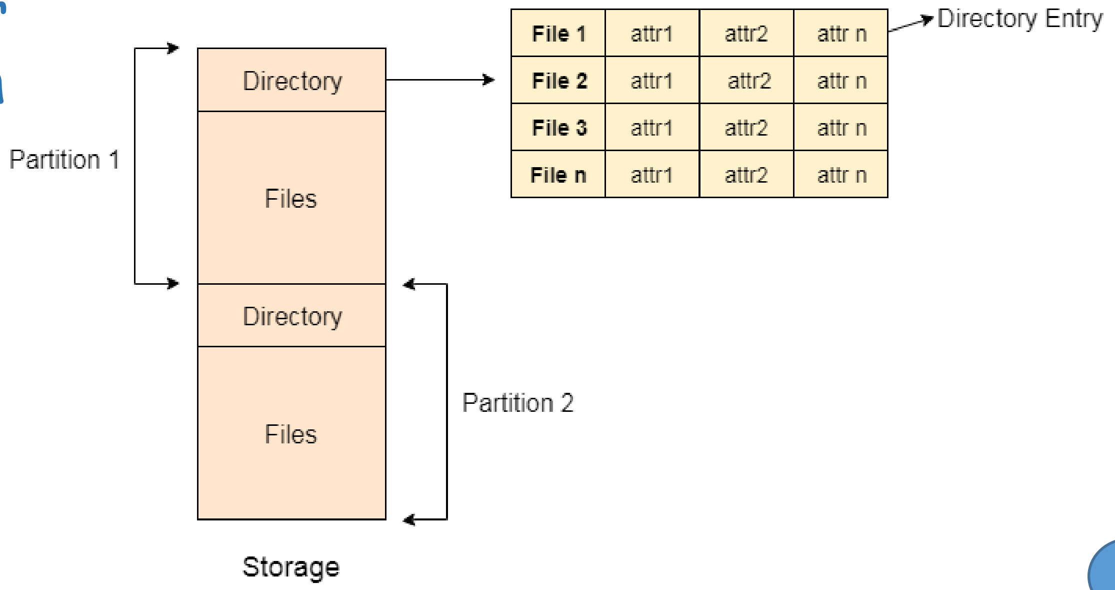
- Types of files
  - Data
    - ✓ Numeric
    - ✓ Character
    - ✓ Binary
  - Program
    - ✓ source programs
    - ✓ executable programs

File Structure

- A file can have various kinds of structure, which depends on its type.
  - A text file is a sequence of characters organized into lines.
  - A source file is a sequence of functions, each of which is further organized as declarations followed by executable statements.
  - An executable file is a series of code sections that the loader can be brought into memory and executed.

File Attributes

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies the file within the 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
- **Extended file attributes**: including character encoding of the file and security features such as a file checksum
- Information about files are kept in the directory table, which is maintained on the disk (nonvolatile device)
  - ✓ A directory table is a special type of file that represents a directory



File 1	attr1	attr2	attr n
File 2	attr1	attr2	attr n
File 3	attr1	attr2	attr n
File n	attr1	attr2	attr n

File Operations

Six Basic File Operations

- Create a file
- Write into a file: write()
- Read from a file: read()
- Deleting a file
- Repositioning within a file
- Truncating a file.
  - ✓ The user may want to erase the contents of a file but keep its attributes.

Other File Operations

- Open(Fi): search the directory structure on disk for entry Fi, and move the content of entry to the main memory (i.e., open-file table).
- Close (Fi) – remove the content of entry Fi from the main memory (i.e., open-file table).
- Copy a file---can be achieved by combining basic file operations.

Opening File Information

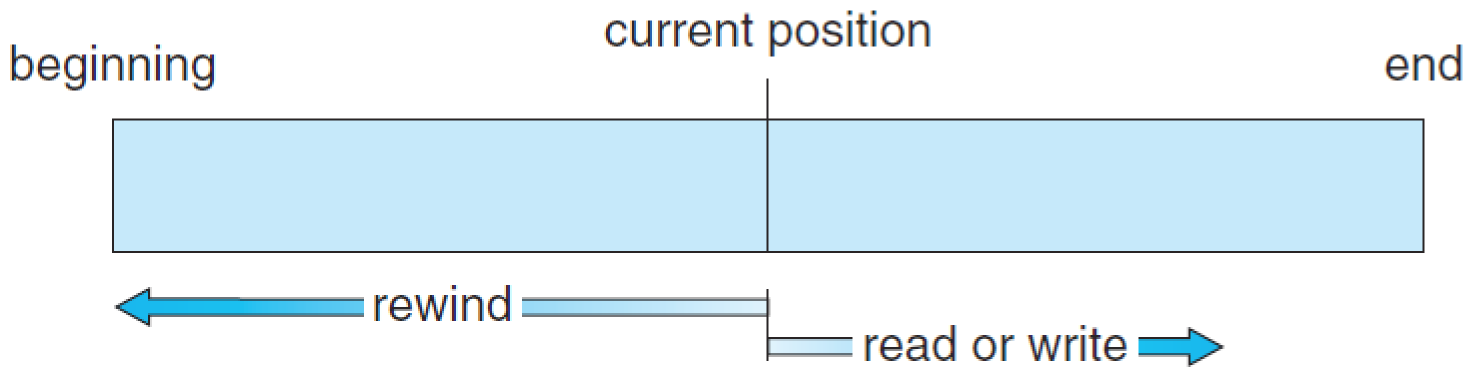
- Information associating an opening file

  - File pointer: pointer to the last read/write location. The file pointer is unique to each process operating on the file.
  - File-open count: multiple processes may have opened a file. File-open count tracks of the number of opens and closes of the file. If File-open count=0 (i.e., all the processes have closed the file), the file entry will be removed from the open-file table.
  - File locks: locking an open lock to prevent other processes from gaining access to it.

    - Mandatory locks: if a process acquires an exclusive lock on a file, the OS will prevent other processes from accessing the file.
    - Advisory locks: if a process acquires a lock on a file, the OS will NOT stop other processes from accessing it. Instead, other process may check the status of the lock (hold/release) to determine whether to access it.

File Access Methods

- File Access
  - Define the ways to read/write data from/to a file.
  - Two common access methods: Sequential Access and Direct Access
  - Sequential Access
    - ✓ Reading/writing of data records/portions in sequential order, that is, one record after the other.
    - ✓ Reset to the beginning.

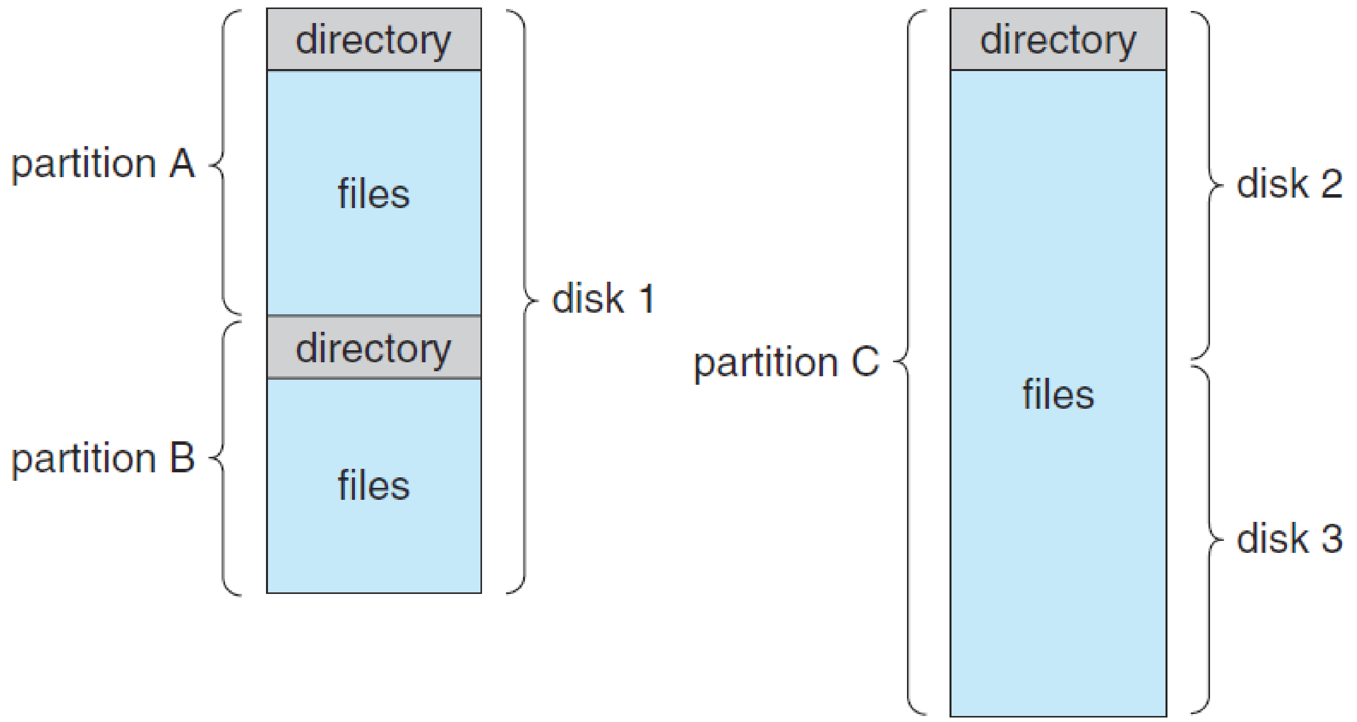


File Access Methods

- File Access
  - Two common access methods: Sequential Access and Direct Access
  - Direct Access
    - ✓ Reading/writing of data records/portions in no particular order.
    - ✓ For example, the file is viewed as a numbered sequence of blocks or records. The process may read block 14, then read block 53, and then write block 7. There are no restrictions on the order of reading/writing for a direct-access file.
    - ✓ Very useful in searching and retrieving data records over a big database

Directory Structure

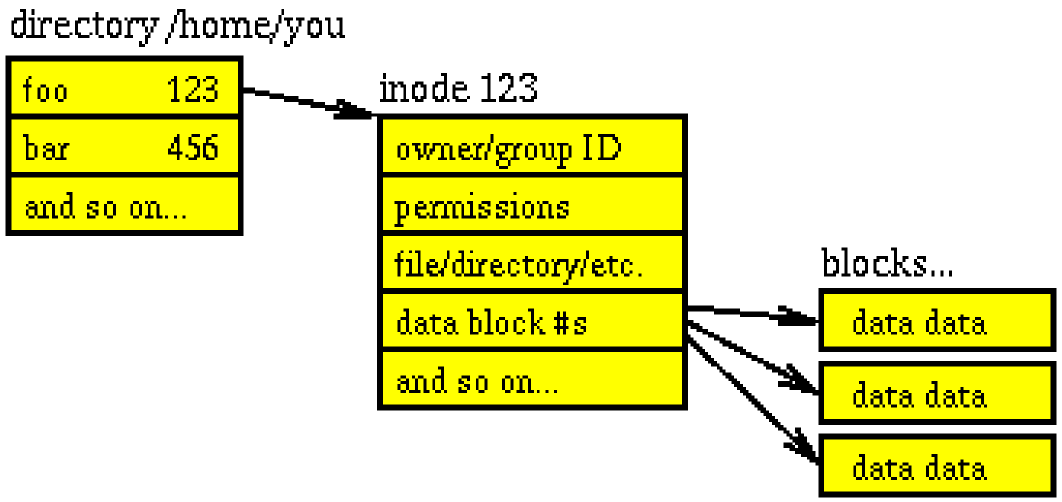
- Directory
  - Directory is a formatted record, containing various attributes of files/subdirectories.



A typical file-system organization

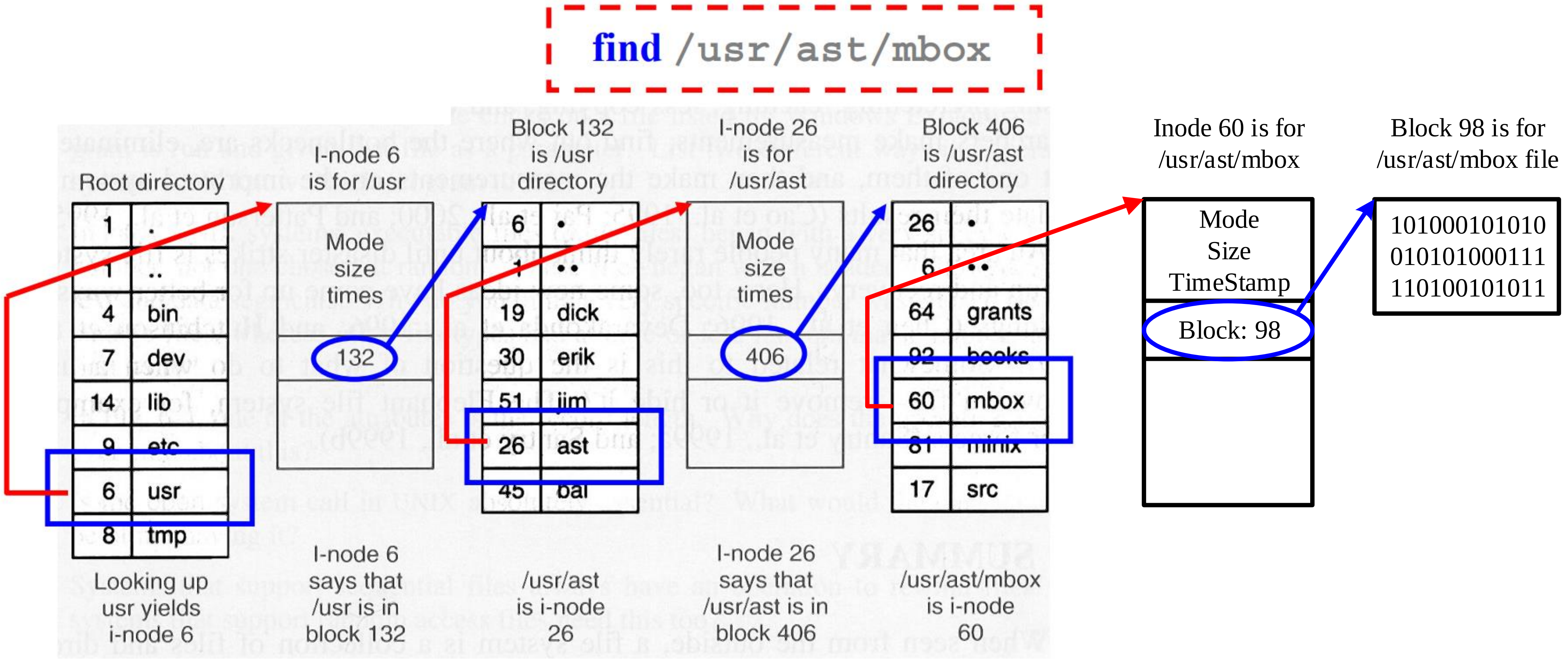
Directory Structure

- Directory in Unix
  - In Unix, the root directory (inode=0) contains names of the files/subdirectory and the mapping to the related inode.
  - inode: is a data structure that describes a file-system object, such as a file or a subdirectory. Each inode stores the attributes and/or disk block location of the object. Each inode is identified by an integer number (i.e., i-number), which is unique in a partition.



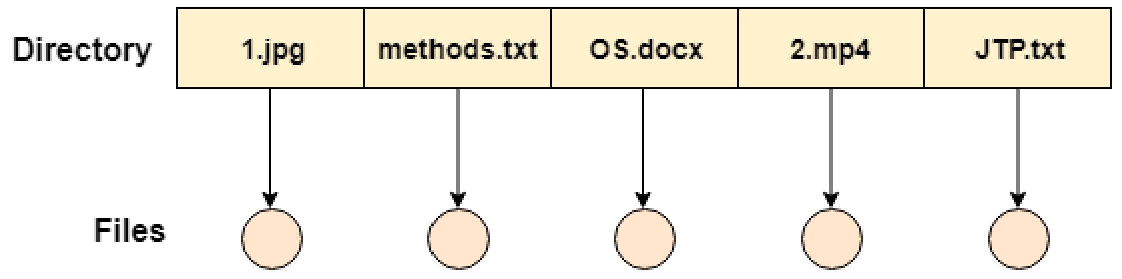
Directory Structure

- One example of directory in Unix



Directory Structure

- Single-Level Directory --- All files are contained in the same directory



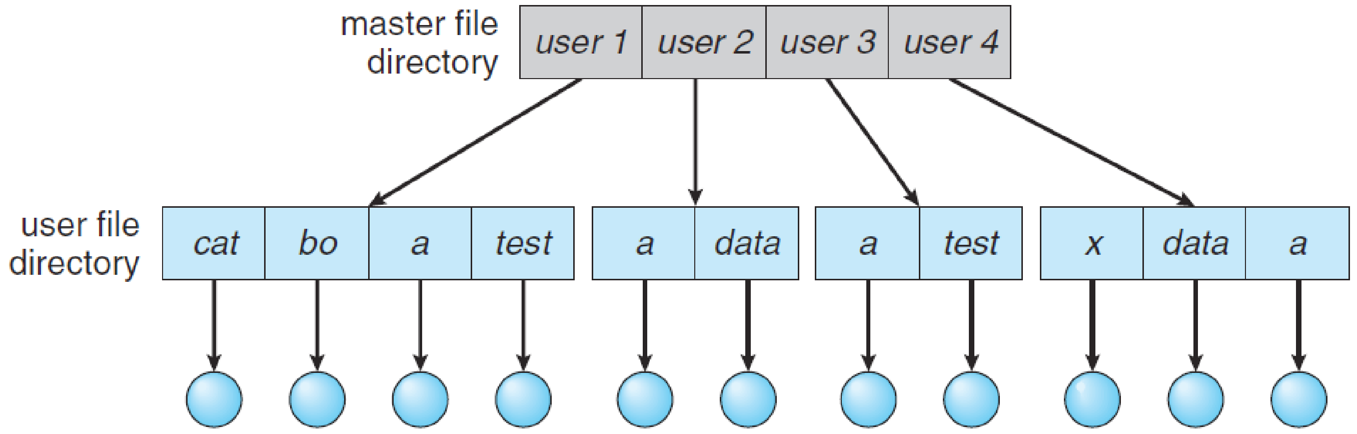
Single Level Directory

- All files are in the same directory, they must have unique names.
  - ✓ If two users call their data file as test.txt, then the unique name rule is violated.
  - ✓ Users have to check the uniqueness of the names when they try to name files.

- Searching files under single directory is not efficient.

Directory Structure

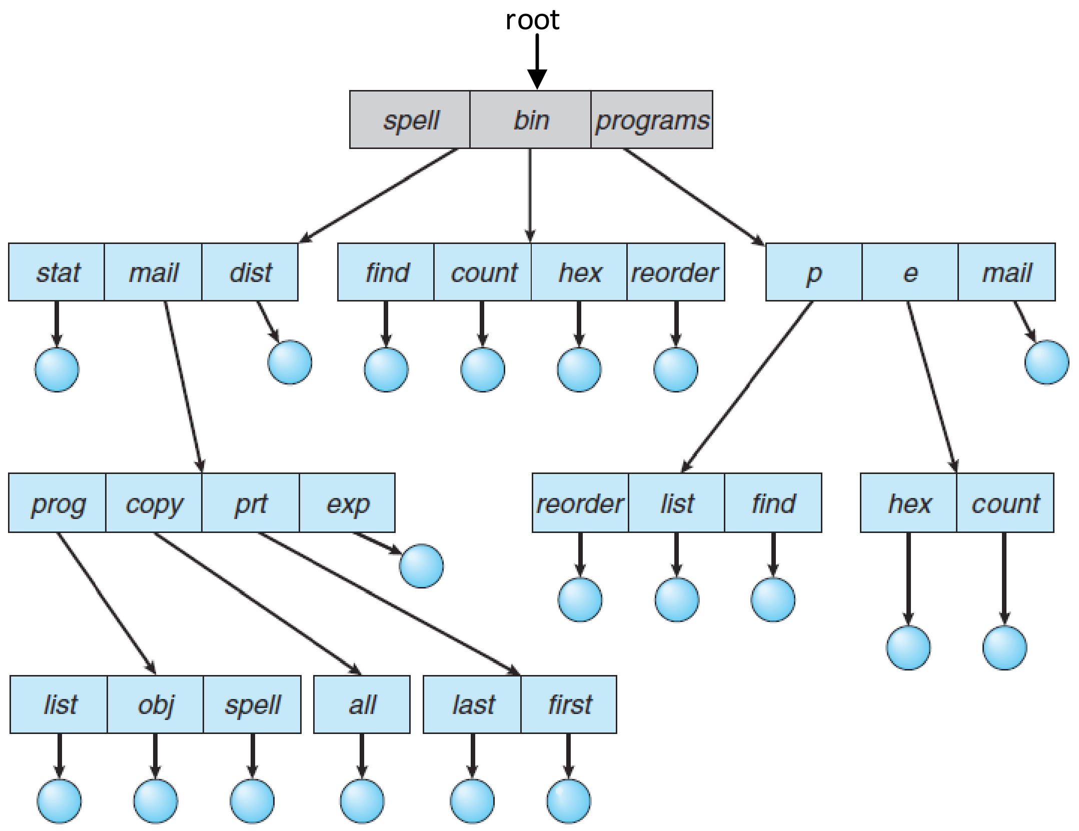
- Two-Level Directory --- Separate directory for each user



- Each user has its own user file directory (UFD).
  - ✓ Different users can choose the same file name.
- Each file can be identified by using username plus filename/UFD.
- A user searching a file is relative efficient.
- However, if the users want to cooperate on some task and to access one another's files, two-level directory is not efficient/possible to achieve it.

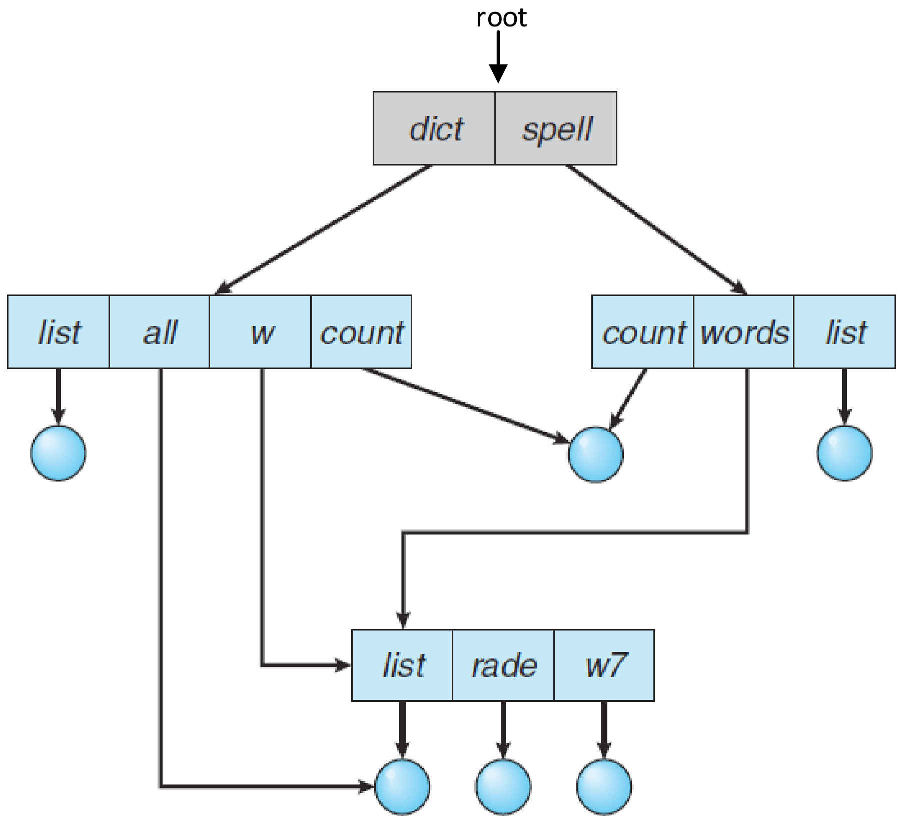
Directory Structure

- Tree-Structure Directory
  - Users are allowed to create their own subdirectories and to organize their files.
  - The tree has a root directory, and every file in the system has a unique path name.
  - Each directory (or subdirectory) contains a set of files/subdirectories.
  - A directory itself is considered as a file of the system to provide information about files.
  - A process has a current directory (working dir).
    - ✓ When the process refers to a file using a simple file name or relative path, the reference is interpreted relative to the current working directory of the process.



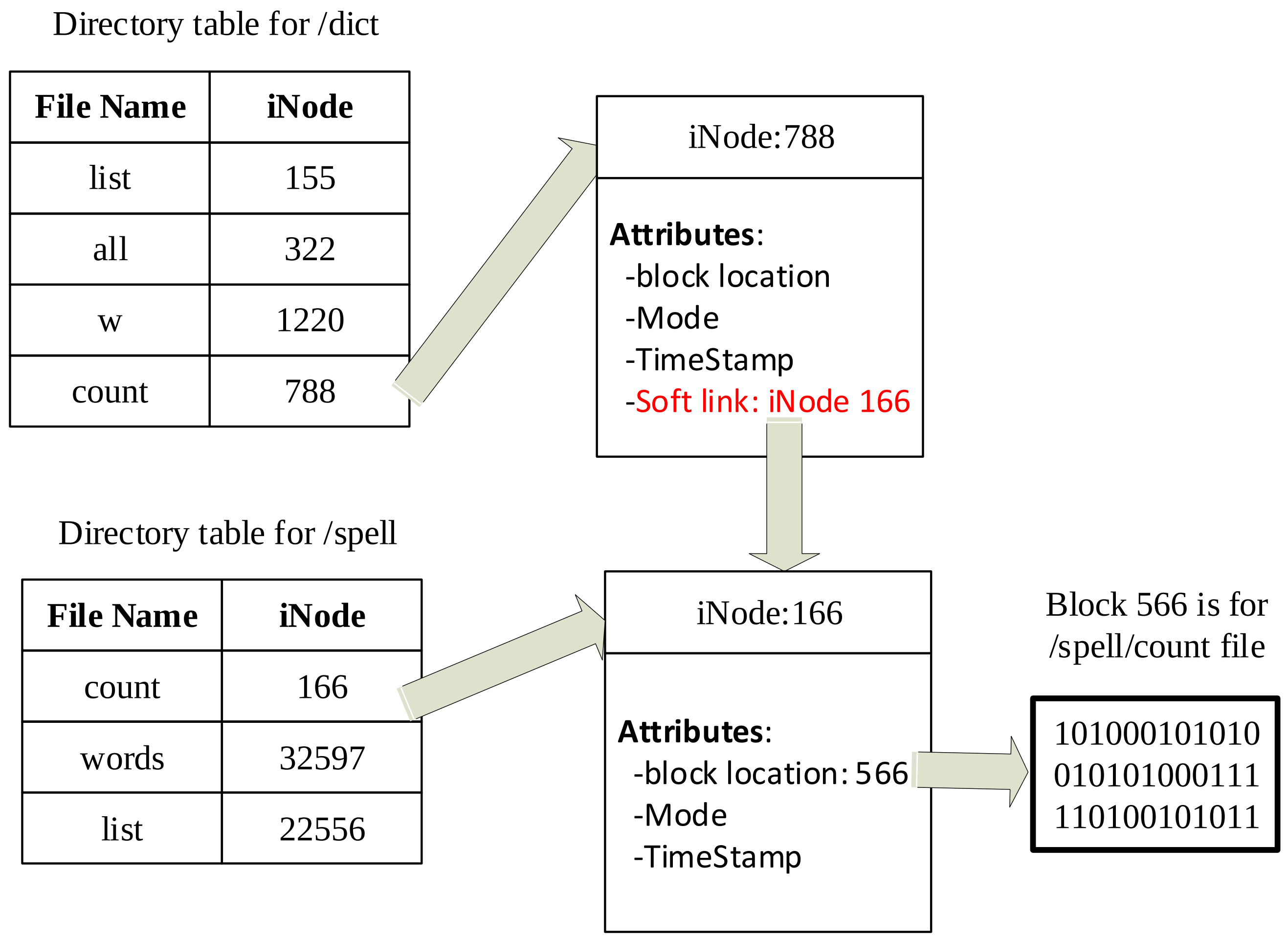
Directory Structure

- Acyclic-Graph Directory—have shared subdir and files
  - The same file or subdirectory may be in two different directories. That is, a file or subdirectory is shared by two directories
    - ✓ A "count" file is in two different directories.
  - New directory entry "link" is designed to achieve the file or subdirectory sharing.
    1. Symbolic link (soft link)
       - ✓ A symbolic link is considered as a pointer points to the pointer of a file or subdirectory.
    2. Non-symbolic link (hard link)
       - ✓ A non-symbolic link is considered as a pointer points to a file or subdirectory.



Directory Structure

- Symbolic link (soft link) in Unix
  - A file in the directory table points to it's own inode, which redirects to another inode that in turn points to a data block.
  - A symbolic link allows redirecting to a path that can cross partitions.
  - If we delete the linked file, the original file will not be deleted.
  - If we delete the original file without deleting the link, we will be left with an orphaned link.



Directory table for /dict

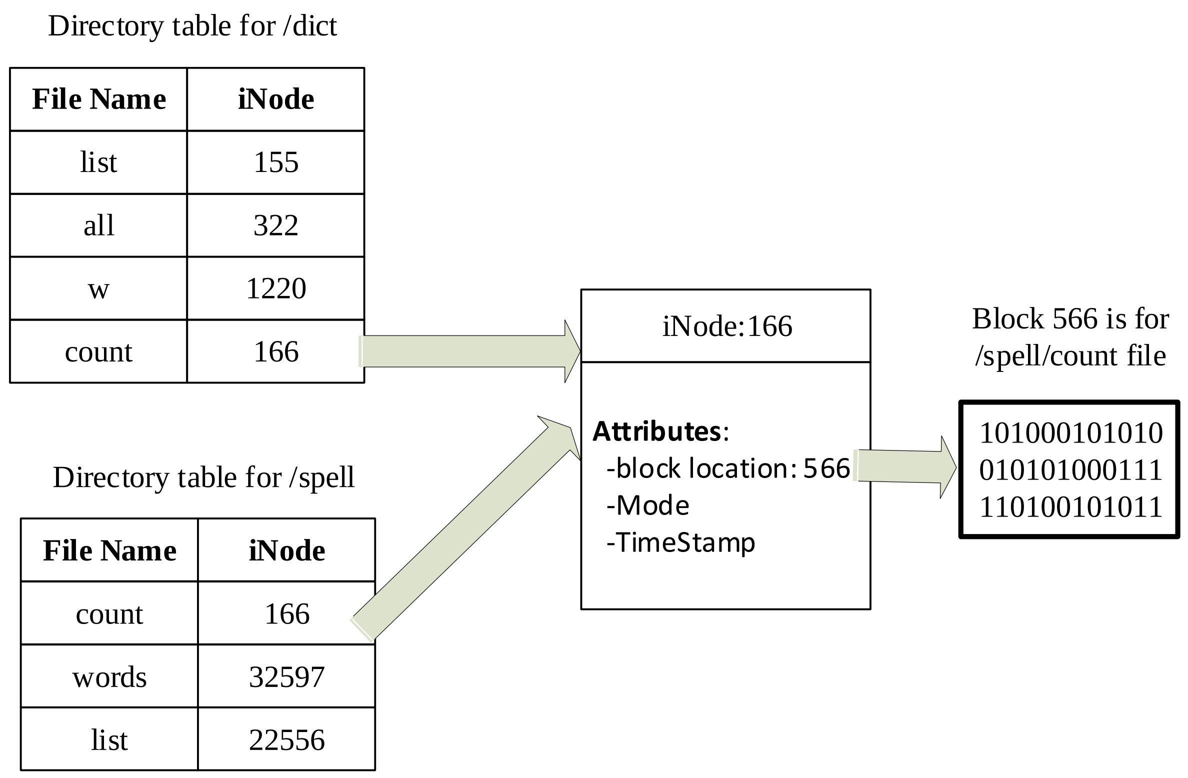
File Name	iNode
list	155
all	322
w	1220
count	788

Directory table for /spell

File Name	iNode
count	166
words	32597
list	22556

Directory Structure

- Non-Symbolic link (hard link) in Unix
  - A non-Symbolic link is the one that directly points to the inode of the shared file.
  - If delete one of the files, then the link is broken, but the inode and data block will remain until all the file names that link to the inode are deleted.
  - A non-symbolic link allows redirecting to a path that with in the same partition.



Directory table for /dict

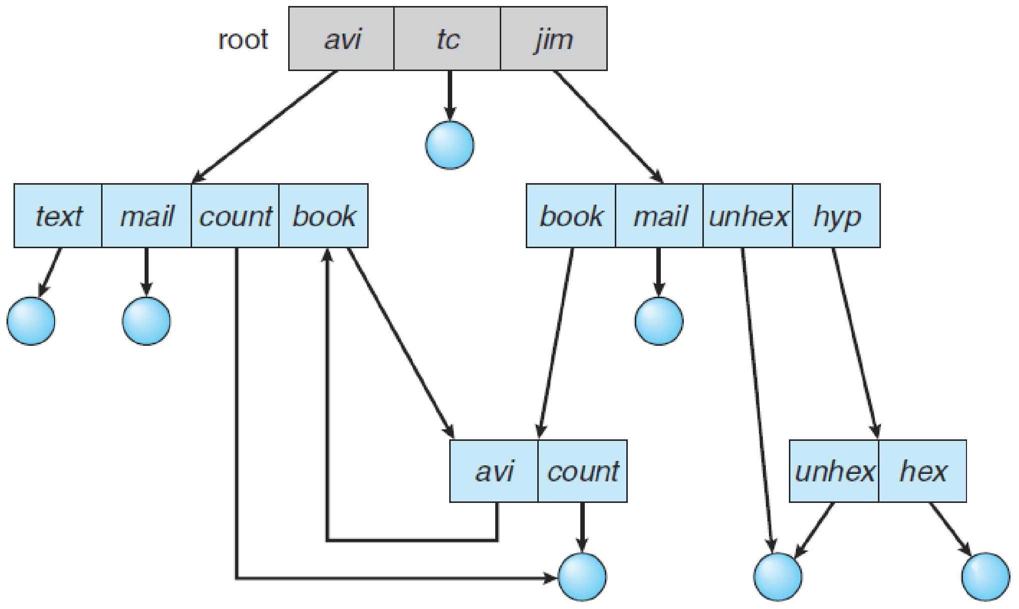
File Name	iNode
list	155
all	322
w	1220
count	166

Directory table for /spell

File Name	iNode
count	166
words	32597
list	22556

Directory Structure

- Acyclic-Graph Directory—have shared subdir and files
  - Cycles in Acyclic-Graph Directory
    - ✓ Acyclic-Graph Directory is no long a tree structure. It may lead to cycles.
    - ✓ Cycles may incur infinite loop once the system tries to search a file.
  - How to guarantee no cycle?
    - ✓ Allow only links to file not subdirectories



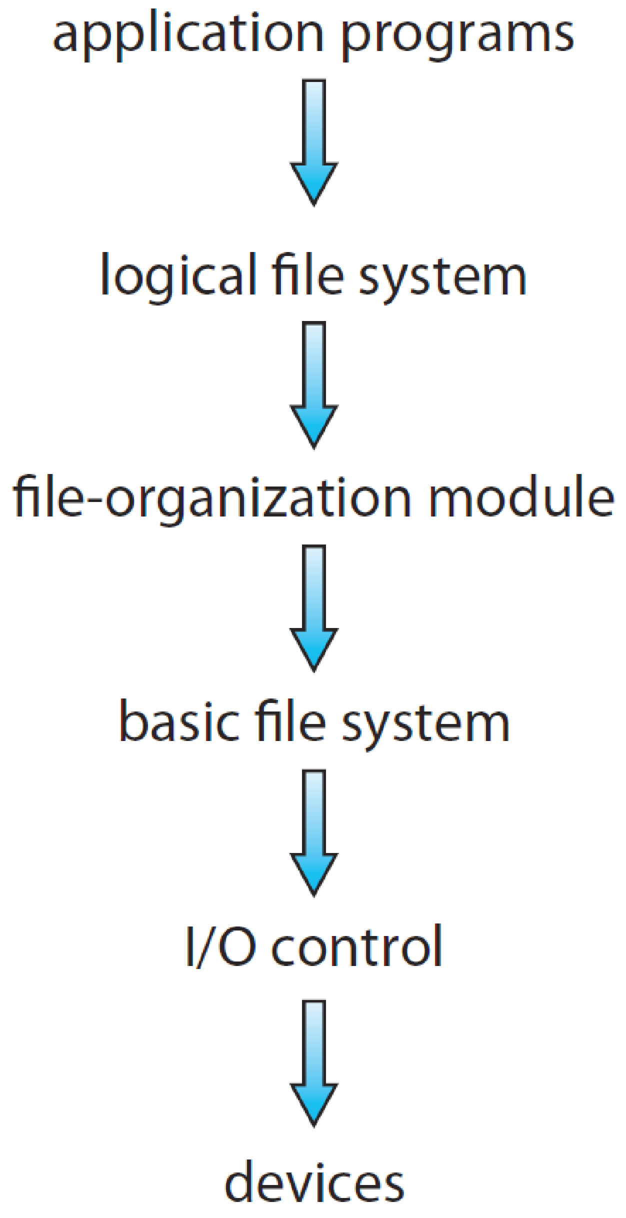
File System Structure

- A file system resides on secondary storage (e.g., disks)
  - Provides user interface to storage, mapping logical file blocks to physical blocks.
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily.

- Some important data structures in a file system
  - Volume control block—contains volume (partition) details
    - ✓ Total # of blocks, # of free blocks, free block pointers in a volume, etc.
  - Directory table/structure—contains information on how to organize files
    - ✓ For example, in Unix, directory table (e.g., containing file names and inode number) represents directory structure
  - File Control Block (FCB) —contains details (e.g., attributes) about a file
    - ✓ For example, in Unix, each inode is a file control block.

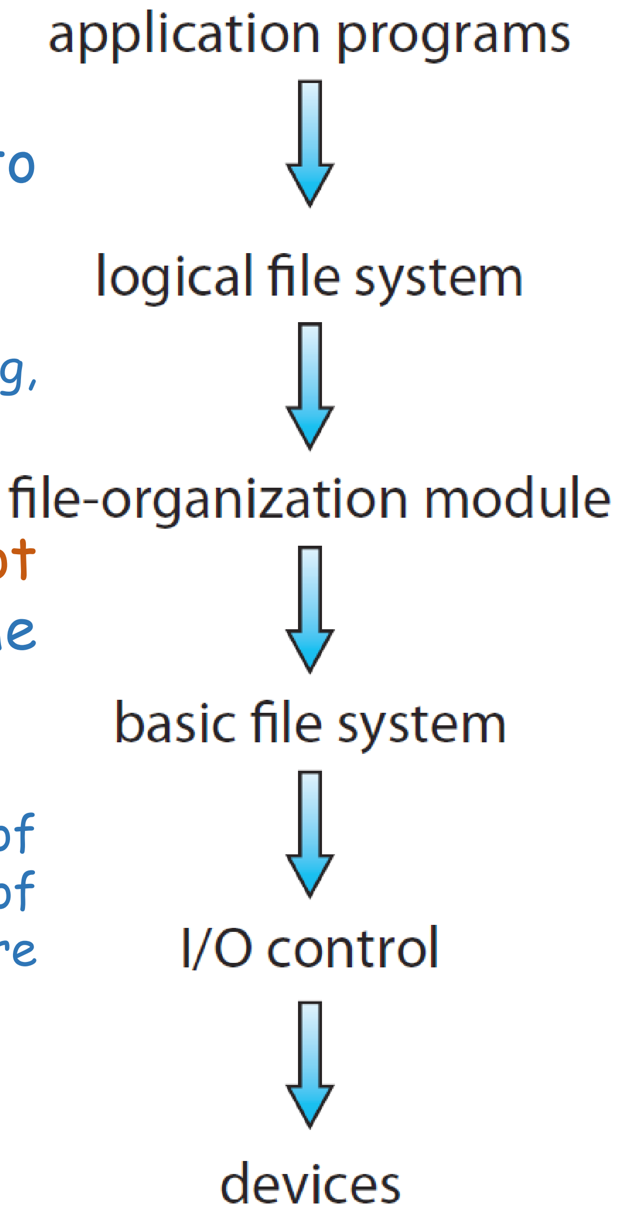
File System Structure

- File system organized into layers
  - Logical file system -manages metadata of files (directory structure)
    - ✓ Maps file names into file control blocks (e.g., inode).
    - ✓ Manages the directory tables of the file system.
    - ✓ Manages protection and security of the file system.
  - File-organization module –understands logical blocks and physical blocks
    - ✓ Mapping logical block numbers/addresses into physical block numbers/addresses.
    - ✓ Manage free space (tracks unallocated blocks).



File System Structure

- File system organized into layers
  - Basic file system -- gives commands to the device driver to read/write physical blocks on the disk
    - ✓ Also manages memory buffers and caches (allocation, freeing, replacement of buffers)
  - I/O control layer--consists of device driver and interrupt handler to transfer information between main memory and the disk.
    - ✓ A device driver can be thought of as a translator. Its input consists of high-level commands such as "retrieve block 123". Its output consists of low-level, hardware-specific instructions that are used by the hardware controller, which interfaces the I/O device to the rest of the system



File System Structure

❑ Directory Implementation

- If we create/delete/updating a file, how to find the file in the directory

- Two methods are used:
  - Linear List
  - Hash Table

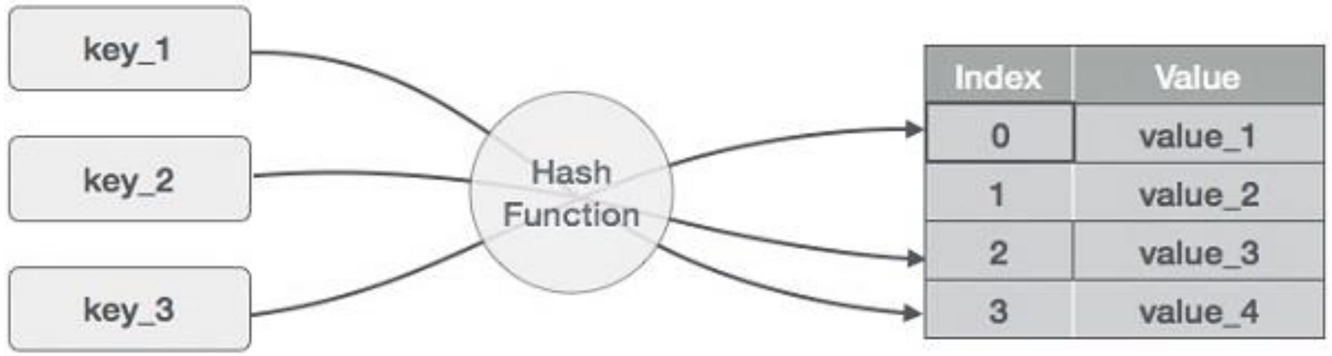
- Linear list: all the files in a directory are maintained as single lined list.

  - ✓ When a new file is created, then the entire list is checked whether the new file name is matching to an existing file name or not. In case, it doesn't exist, the file can be created at the beginning or at the end.
  - ✓ Searching for a unique name is a big concern because traversing the whole list takes time.
  - ✓ The list needs to be traversed in case of every operation (creation, deletion, updating, etc.) on the files therefore the systems become inefficient.

File System Structure

❑ Directory Implementation

➢ Hash table: A key-value pair for each file in the directory gets generated and stored in the hash table.

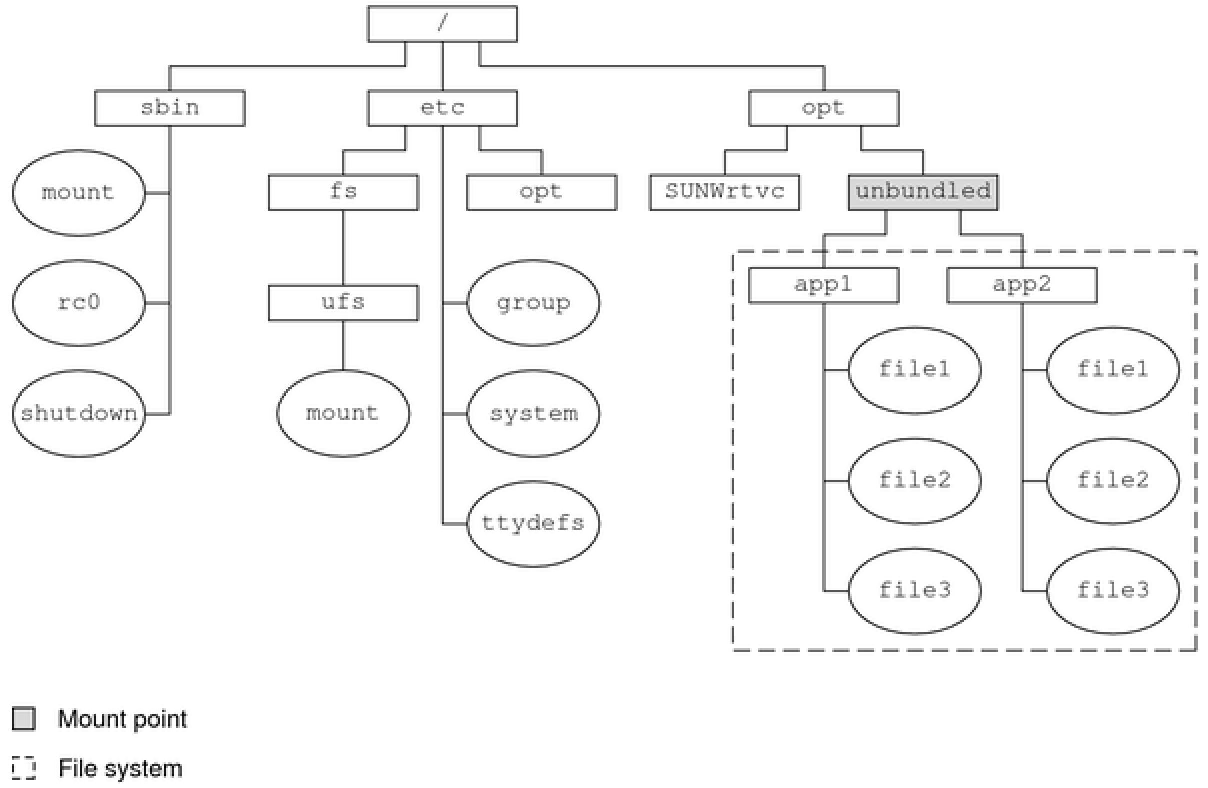


- key_1/ key_2/ key_3 are the file names
- value_1/value_2/value_3 are the inode numbers

✓ Reducing the directory searching time.
✓ May incur collision: two different file names hash to the same value.
✓ Hash tables are generally fixed size, thus reducing its scalability.

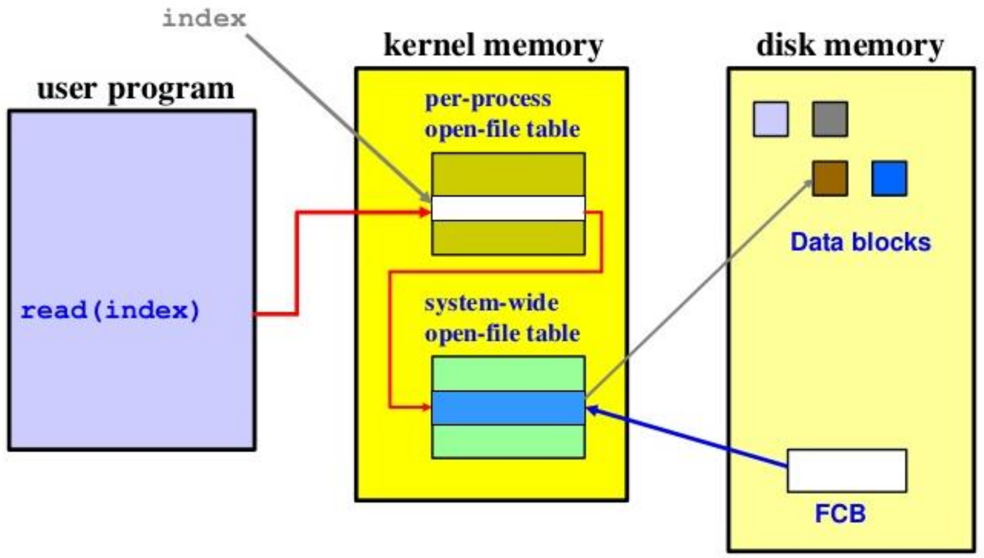
File System Mounting

- An OS may have more than one file systems (devices), and file system must be mounted before it can be available to processes on the system.
- Once a file system is mounted, the OS will give the name (e.g., device ID), mount point, and type (option) of the file system.
  - A mount point is a directory (typically an empty one) in the currently accessible filesystem on which an additional filesystem is logically attached.
  - To access a local file system, you have to
    - ✓ Create a mount point
    - ✓ Mount the local file system by using the mount command

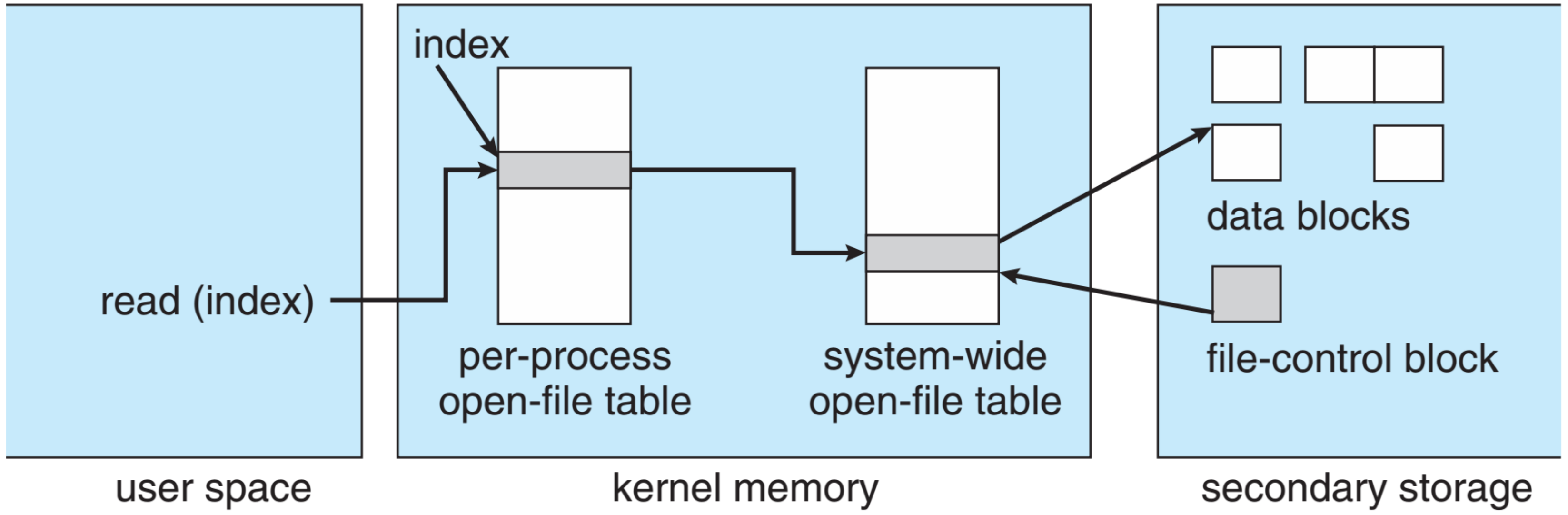


In-Memory File System

- In addition to on-disk file structures, several file system structures are maintained in memory as well.
  - A system-wide open-file table that contains copies of the FCBs (i.e., inodes) for all the open files.
  - A per-process open-file table that contains a pointer to the appropriate entry in the system-wide open-file table.
  - A mount table that stores names of mounted file systems, mount points, file system types, etc.

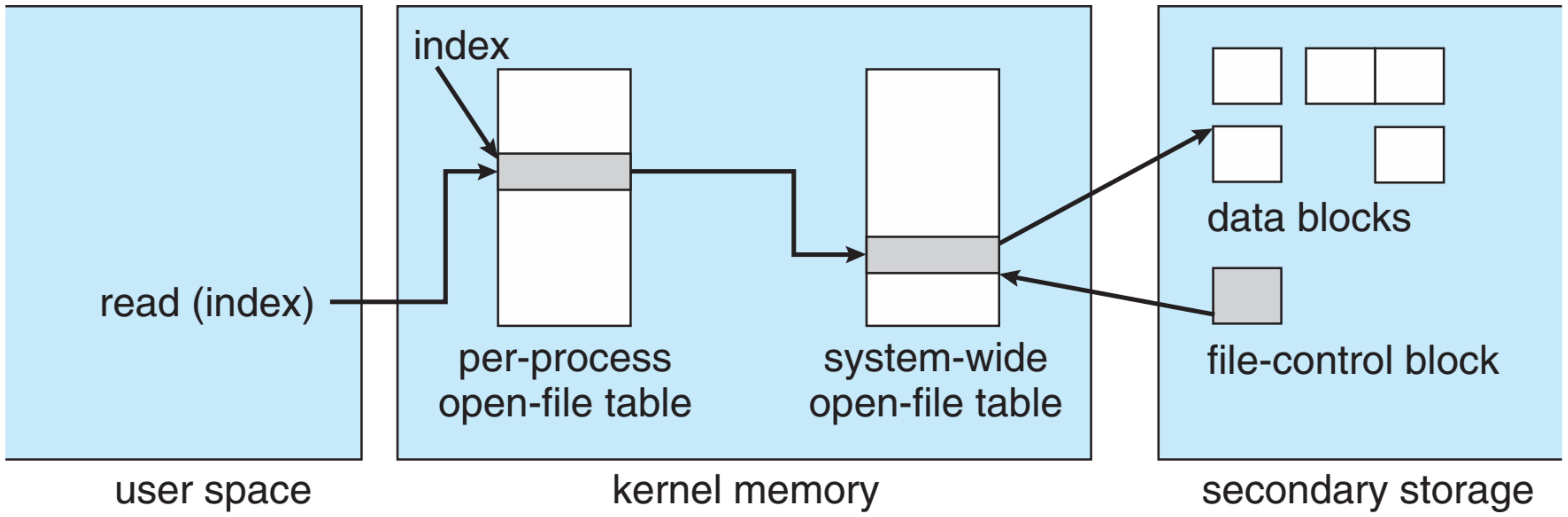


In-Memory File System – Open Files



- The open() call passes a pathname to the logical file system, which first searches the system-wide open-file table to see if the file is already in use by another process.
  - ✓ If yes, a per-process open-file table entry is created, and points to the existing system-wide open-file table.
  - ✓ If no, the directory structure is searched for the given file name. Once the file is found, the FCB is copied into a system-wide open-file table in memory. A per-process open-file table entry is created and points to the existing system-wide open-file table.
  - ✓ System-wide open-file table not only stores FCBs but also file-open count, which tracks the number of the processes that have the file open.
- A file descriptor will be returned, if open() is successful. The file descriptor will be used in subsequent system calls (e.g., write() and read()).

In-Memory File System – Read/Write/Close Files

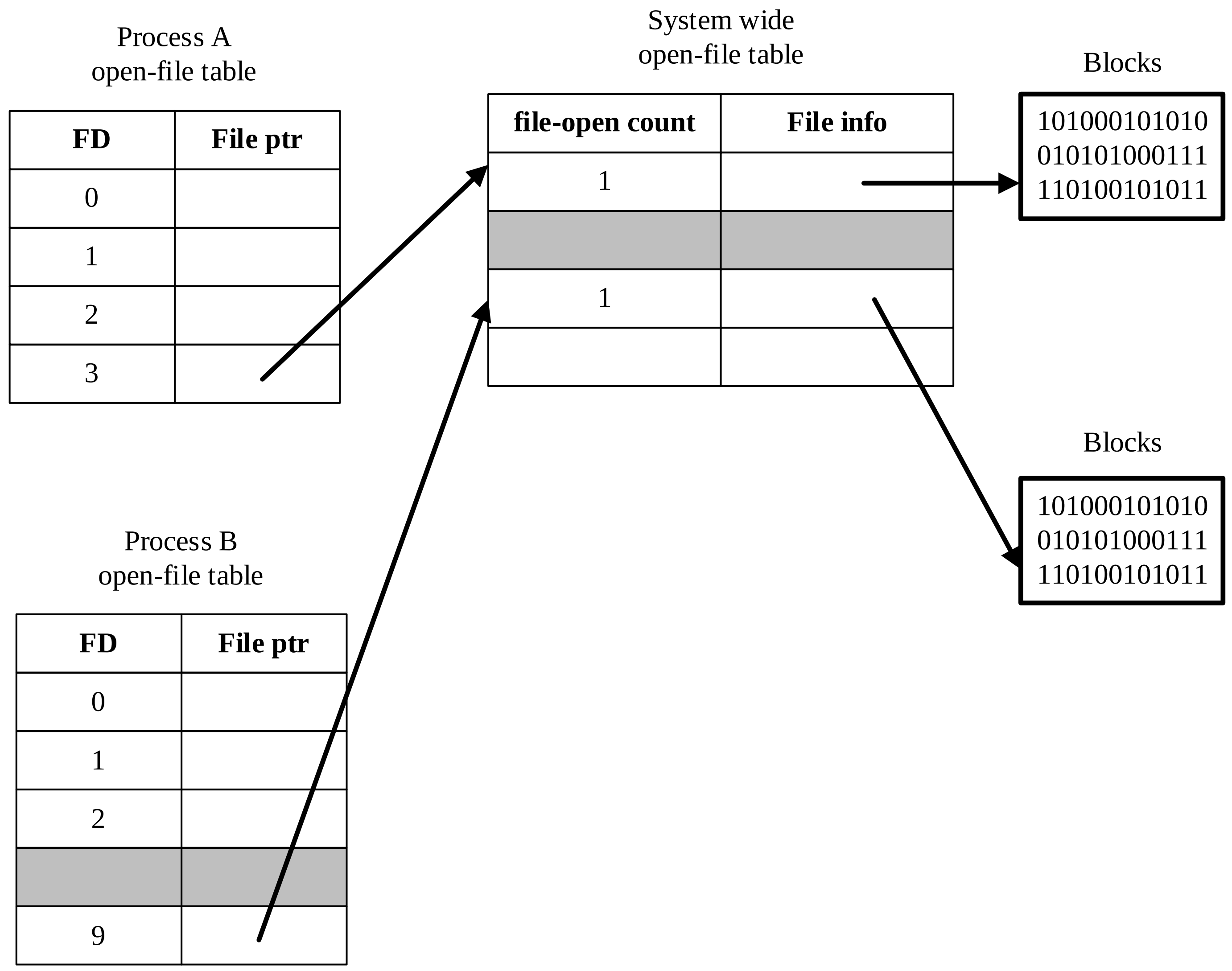


- When a process closes the file, the per-process open-file table entry is removed, and the related file-open count in the system-wide open-file table is decremented.
- When all the processes (which previously opened the file) has closed the file (i.e., file-open count=0), the related entry in the system-wide open-file table is removed.

In-Memory File System - Read/Write/Close Files

❑ More about file descriptor

- A file descriptor is a non-negative integer.
- File descriptor is used to identify the opened files by a process. Basically, it is the index of the open-file table.
- Three file descriptors (i.e., 0, 1, 2) are reserved for stdin, stdout, stderr. They are automatically generated when a process is created.


Process A
open-file table
FD
File ptr
0
1
2
3
System wide
open-file table
file-open count
File info
1
1
Blocks
101000101010
010101000111
110100101011
Process B
open-file table
FD
File ptr
0
1
2
9
Blocks
101000101010
010101000111
110100101011