Agenda: File Systems (Chapter 4, 10.6, 11.8)

Context/Definitions

Review: files and directories

Symbolic Links

Mounts

File System Implementations

- Contiguous Allocation
- Linked List
- FAT
- inode

Secondary Storage

Advantages

- Greater capacity
- Persistent data
  - Survives loss of power
  - Exists outside of process execution
- Easily shared by multiple processes

How should data be organized in secondary storage?

Answer: Files

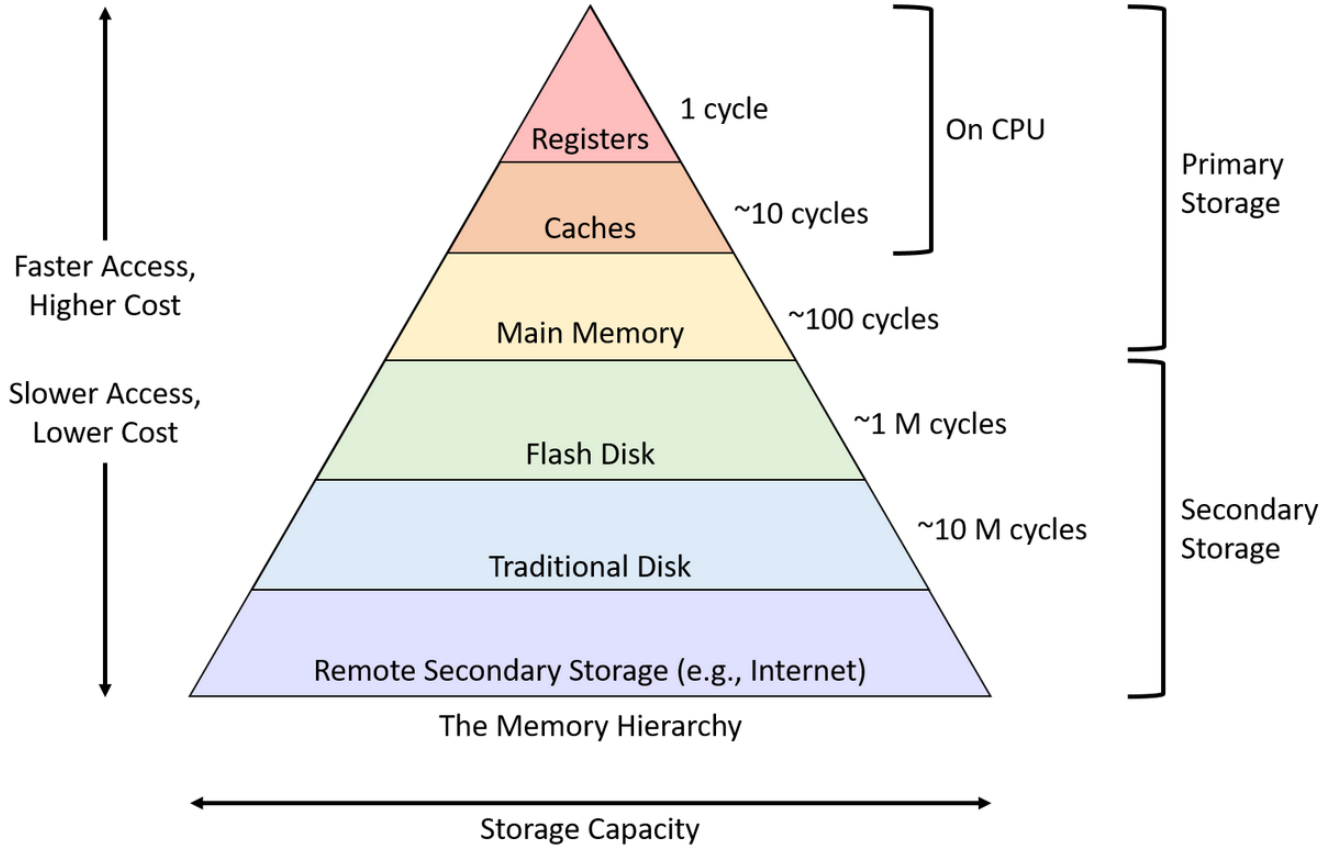


Figure 1. The memory hierarchy

From *Dive into Systems*

File Systems

A **file** is a logical collection of information

Can store text (text files)

Can store generic data (binary files)

A **file system** is the software responsible for managing files

creating, destroying, reading, writing, modifying, moving, and sucurity

GUI view of your file system:

- On windows -> windows explorer
- On Mac -> finder
- On Ubuntu -> open the home folder on the desktop

Properties of files

Names: how do we distinguish this data from others?

Format: what does the file's data mean?

- Most OS's treat files as a generic sequence of bytes (flexible, extensible)

Extension: how do we specify the format of the file?

- A **magic number** embedded in the header can also be used to determine/verify the file type

Type: Files serve multiple purposes

- **Regular files** contain user information.
- **Directories** are system files that organize groups of files
- **Character special files** are related to input/output and used to model serial I/O devices, such as terminals, printers, and networks.
- **Block special files** model disk I/O

Reading/Writing Files

Early OSs only supported **sequential access**

- read bytes/records from the beginning
- cannot jump around, could rewind or back up
- convenient when medium was magnetic tape

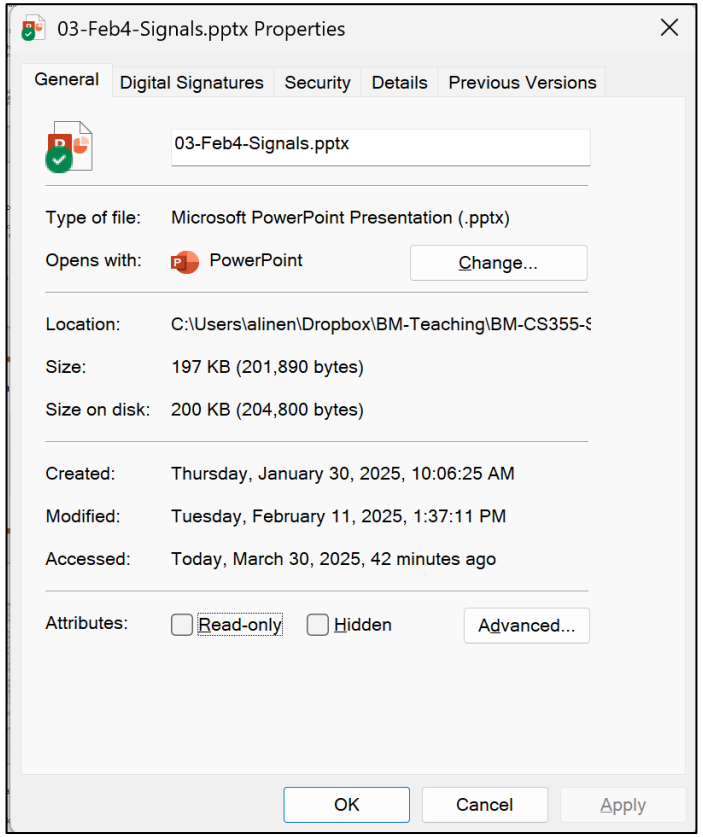
Modern OSs support **random access**

- bytes/records can be read in any order
- current point in file can be modified with **seek**

File Attributes (metadata)

In explorer, right-click and choose properties

At the command prompt, do dir (Windows) or ls (UNIX)




```
alinen@goldengate:~/cs223/class-examples/lec0$ ls -l
total 40
-rwxr-xr-x 1 alinen faculty 16696 Jan 18 14:42 a.out
-rw-r--r-- 1 alinen faculty    76 Jan 18 14:42 hello.c
-rw-r--r-- 1 alinen faculty   416 Jan 18 13:58 Hello.class
-rw-r--r-- 1 alinen faculty   104 Jan 18 13:58 Hello.java
-rw-r--r-- 1 alinen faculty   934 Jan 18 14:03 Sqrt.class
-rw-r--r-- 1 alinen faculty   197 Jan 18 14:03 Sqrt.java
```

Directories

A **directory** (or folder) contains other directories and files

The **root directory** is at the base of the file system

The **current working directory** (cwd) is the location from which the current process is running

A **path** is the exact location of a file within the file system

- **relative path:** with respect to the current working directory
- **absolute path:** with respect to the file system root

Organizing files with directories

The simplest approach is a **single-level directory system**

- All files are stored in the same directory (flat hierarchy, one parent with many children)

More common approach is a **hierarchical directory system**

- Directories and files can be nested in a tree data structure

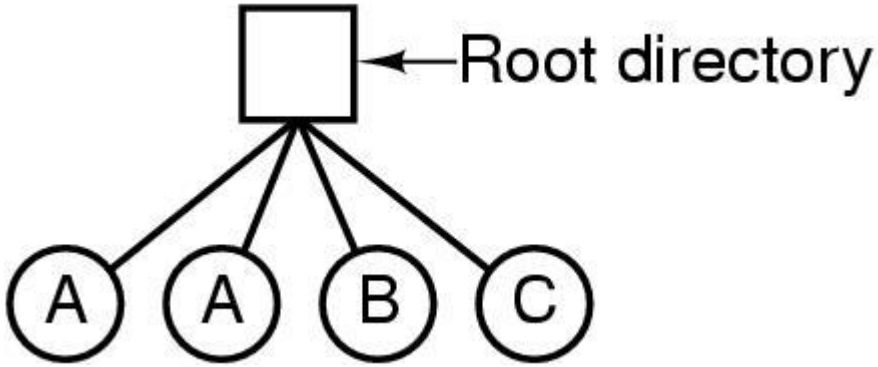
Example: Single-level directory system

A single level directory system

- contains 4 files
- owned by 3 different people, A, B, and C
- ownerships are shown, not file names

Simple

Works well on single-user systems

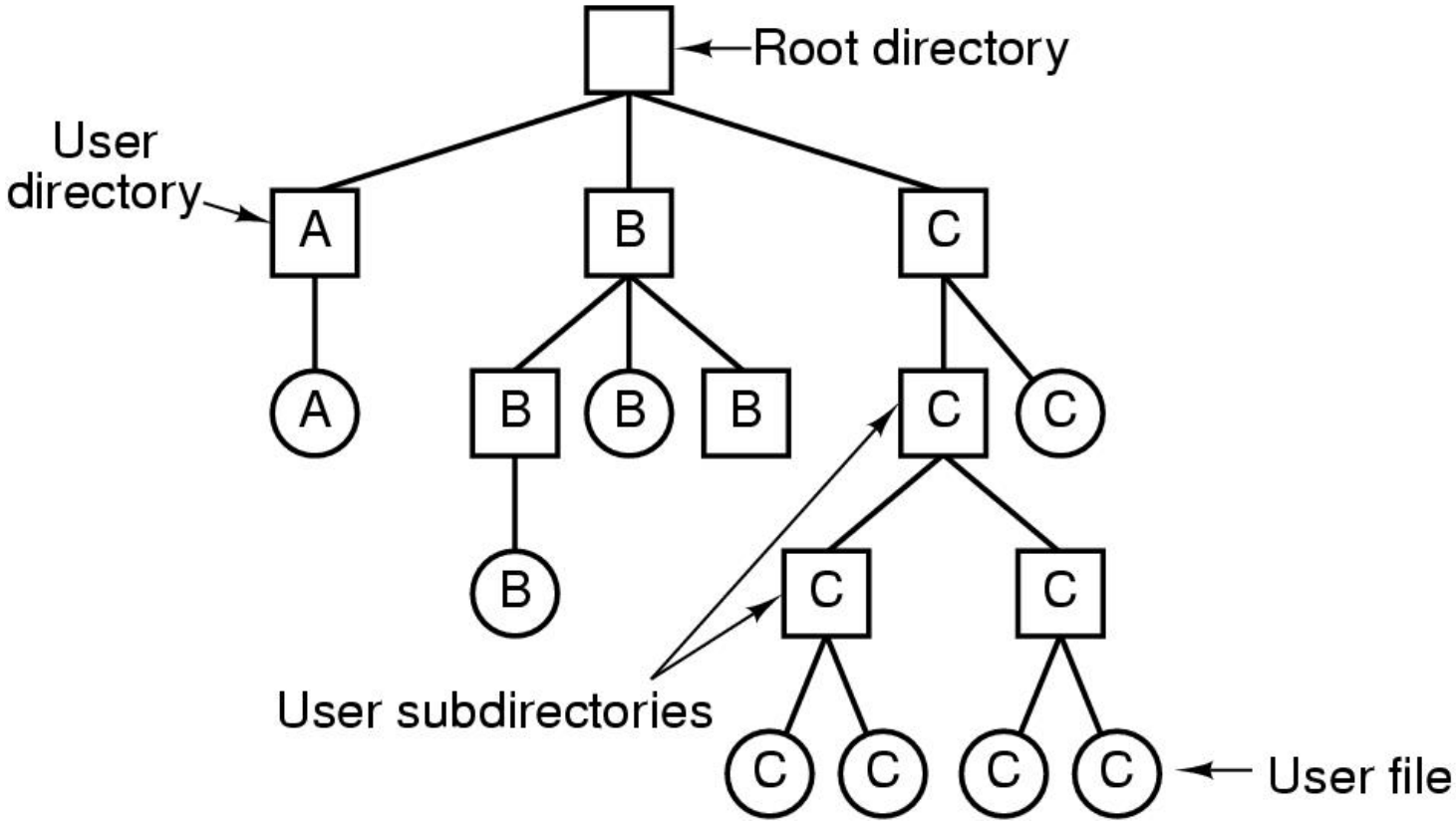


Example: Hierarchical Directory Systems

Hierarchical Directory:

- contains 8 files
- owned by 3 different people, A, B, and C
- ownerships are shown, not file names

Related files can be grouped together



Symbolic links

Useful for

- Sharing files across users without copying
- Aliasing the names for files
  - Ex: Dynamic library names
  - Ex: Referring to a file in your current working directory

Use the command: `ln –s <filename> <newname>`

Files need to maintain the number of symbolic links to a file

Demo: shared libraries

Mounts

Windows keeps partitions separate, based on names

- Example: To switch drives at the terminal, you type its name (C:)

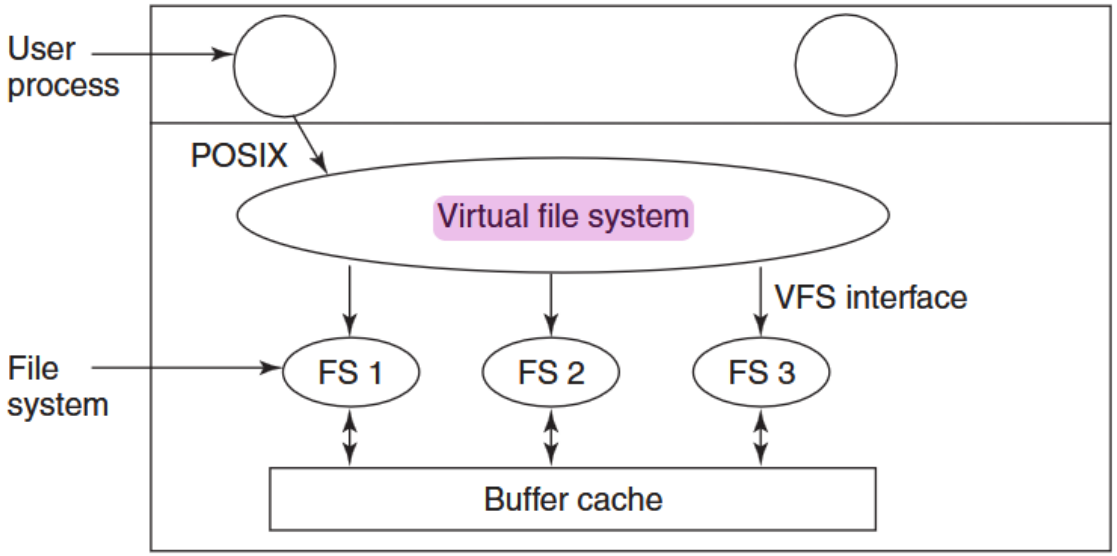
UNIX allows users to **mount** filesystems

- Goal: integrate multiple file systems into a single structure
- The mounted file system looks like a normal directory
- Example: `/mnt/c` on windows subsystem for linux (WSL)

Virtual File System (VFS)

The **virtual file system** (VFS) maintains an interface for working with files and directories

- The VFS hides the details of how the file system is stored
- The VFS might actually hide a network file system (NFS) or a local file system; the user doesn't need to know!



System calls invoke VFS

- create, unlink, write, read, stat, fctl

File System Implementation

File systems are stored on **disks**

Disks can be split into **partitions**, or big chunks of contiguous memory. Each partition can have its own file system.

For example, the C: versus D: drives on Windows

Goal: Keep track of where files are stored on disk

File System Layout

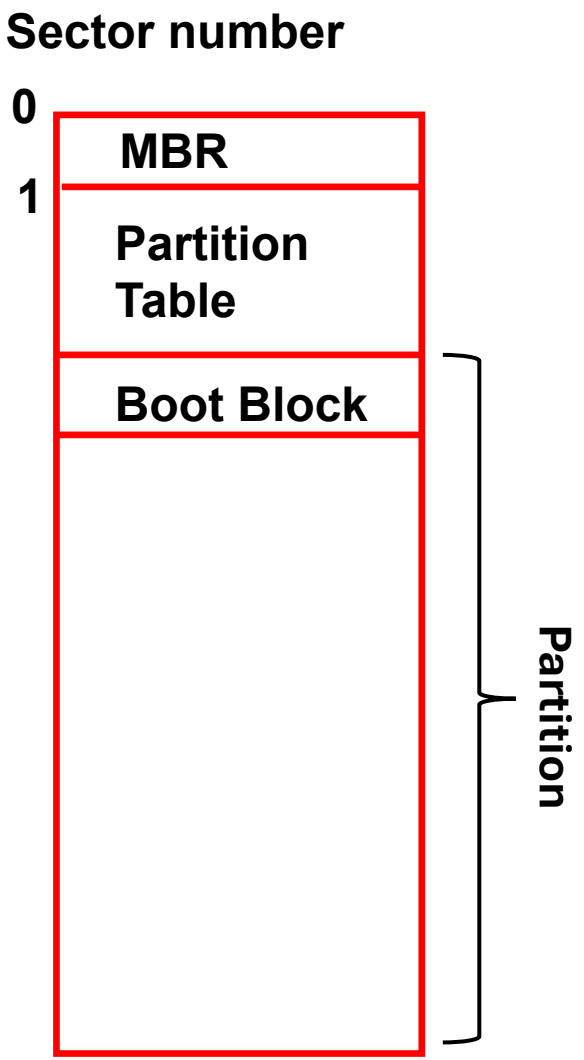
The first partition on a disk is called **Sector 0**

File Systems often adhere to the following convention to aid booting

- The **Master Boot Record (MBR)** is stored in sector 0
- The next section contains the **Partition Table**
  - Lists start/end of each partition
  - Stores the active partition

When booting your computer

- BIOS executes the MBR
- The MBR locates the active partition, reads the first block (called the **boot block)** and executes

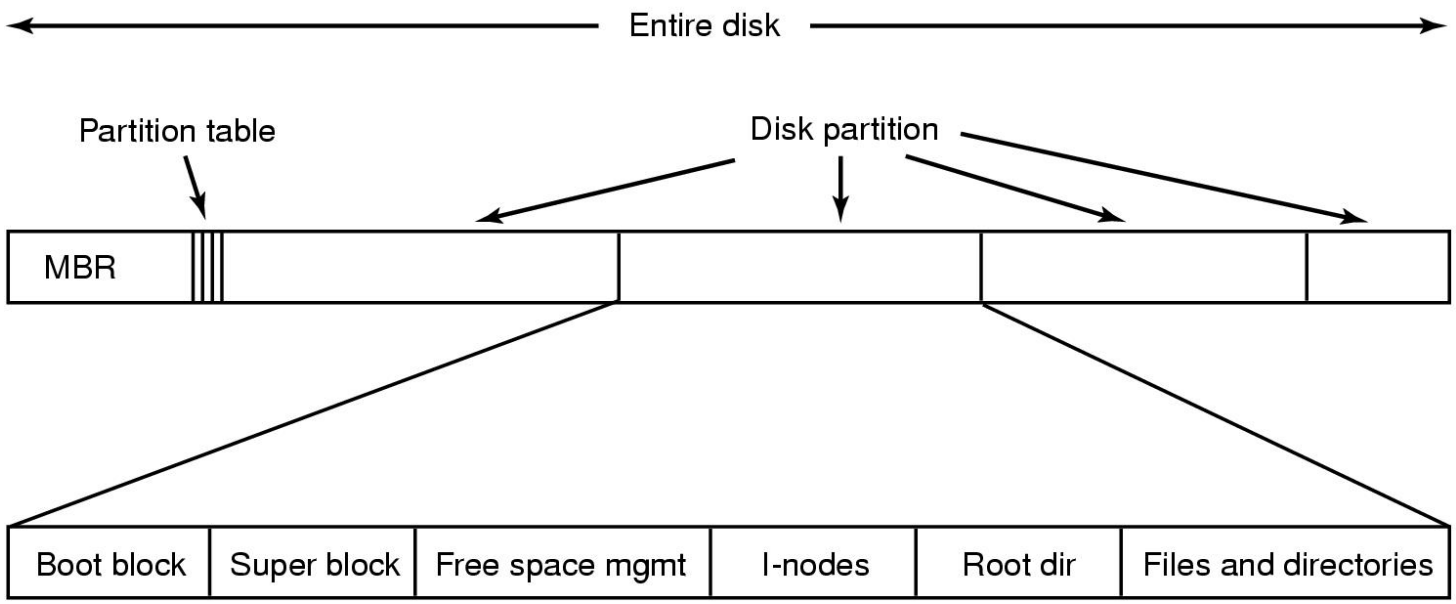


Implementing Files

Beyond the boot block, partitions can be organized in many different ways

Assume that disk storage is segmented in blocks

How should we keep track of which blocks belong to which files?



A possible file system layout w*ith a Unix partition*

Fragmentation

Like with virtual memory paging, files use memory based on blocks

Internal fragmentation occurs when memory is unused within a block

External fragmentation occurs when memory between blocks is too small for re-use

Approach #1: Contiguous Allocation

Idea: Use consecutive blocks for files

Example: Suppose we have 1 KB blocks and a 50 KB file. We would use 50 consecutive blocks

Advantages:

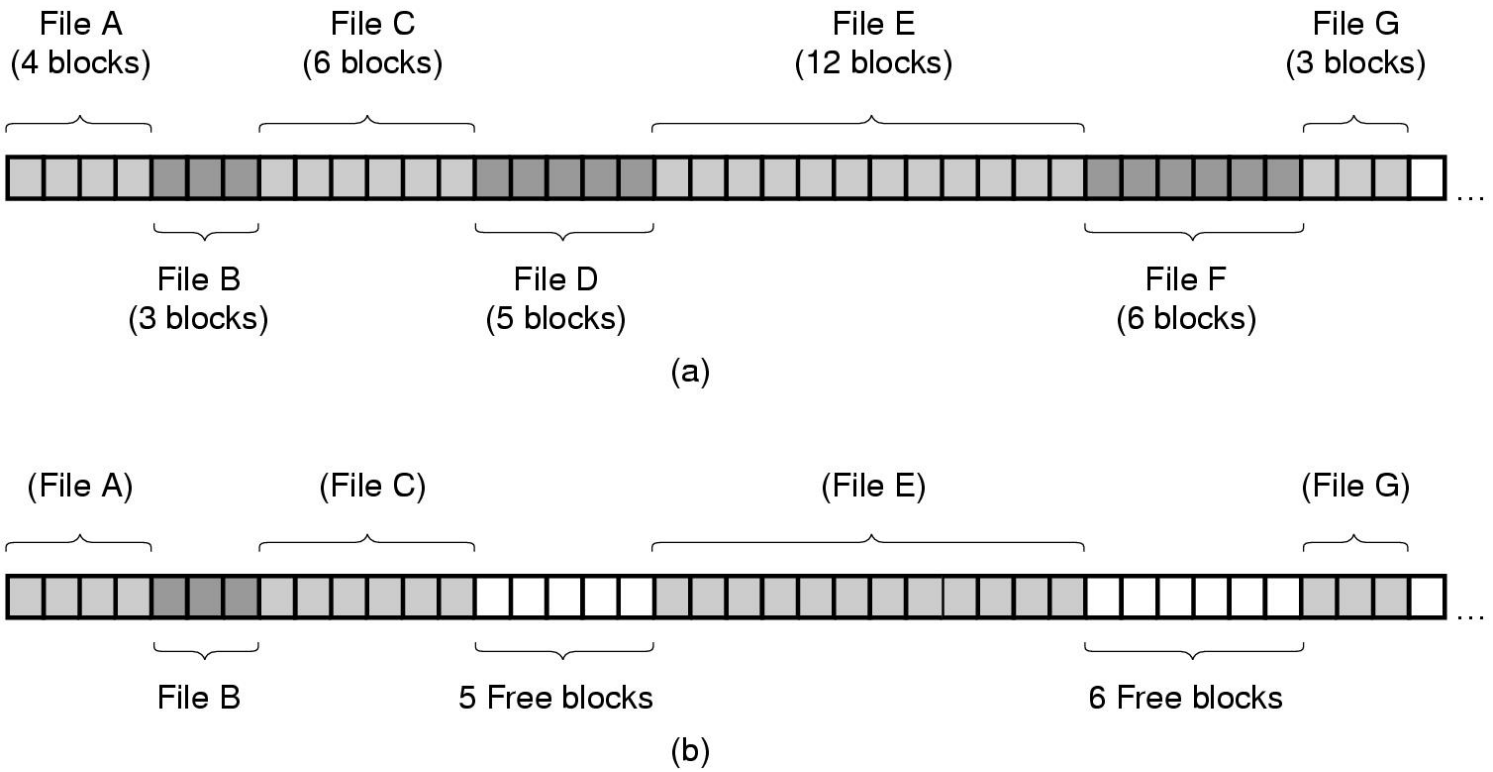
- Simple: only need to store start block and size
- Fast: can access file data sequentially

Disadvantages:

- External fragmentation over time
- Need to know sizes of files in advance
- Compacting memory to make space (e.g. **defragmentation**) is slow
- Random access is slow (requires linear search from start)

Assume 1 KB blocks

Contiguous Allocation



(a) Contiguous allocation of disk space for 7 files

(b) State of the disk after files *D* and *E* have been removed

Approach #2: Linked-lists

Idea: Allow files to use non-contiguous blocks

How it works: Reserve first few bytes of each block to store a pointer to the next block, or NULL is the end of the file

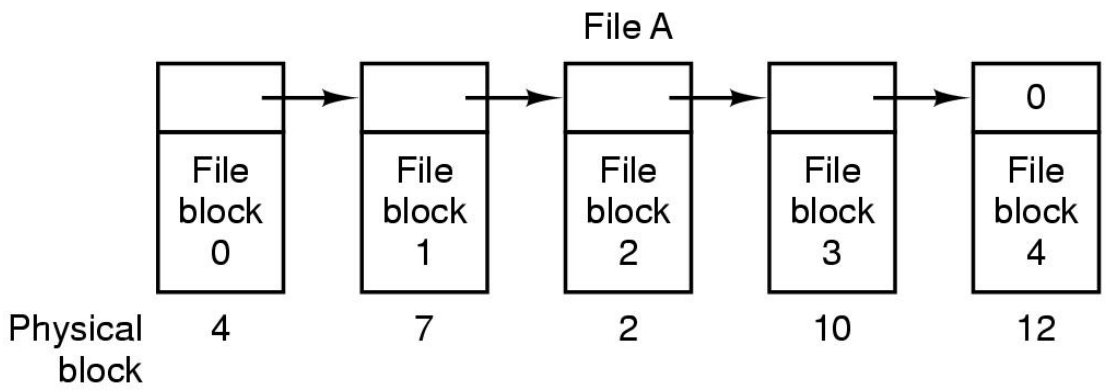
Advantages:

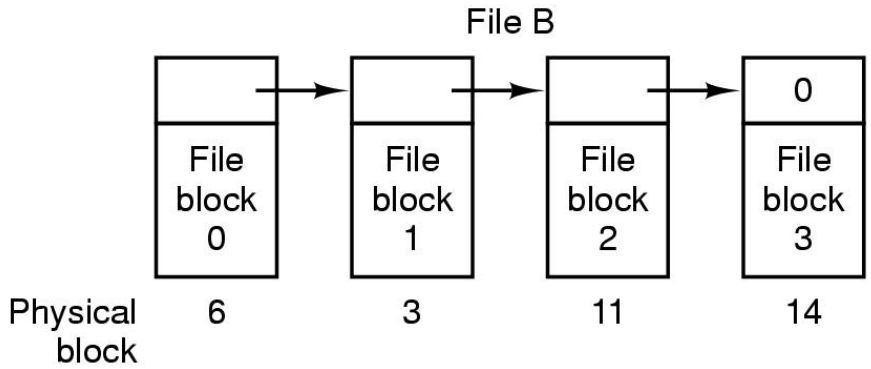
- No external fragmentation

Disadvantages:

- Random access is slow (requires linear search from start)
- Awkward storage size for each block: Blocksize – sizeof(pointer)

Exercise: Visualize the following files as blocks in memory





Approach #3: File Allocation Table (FAT)

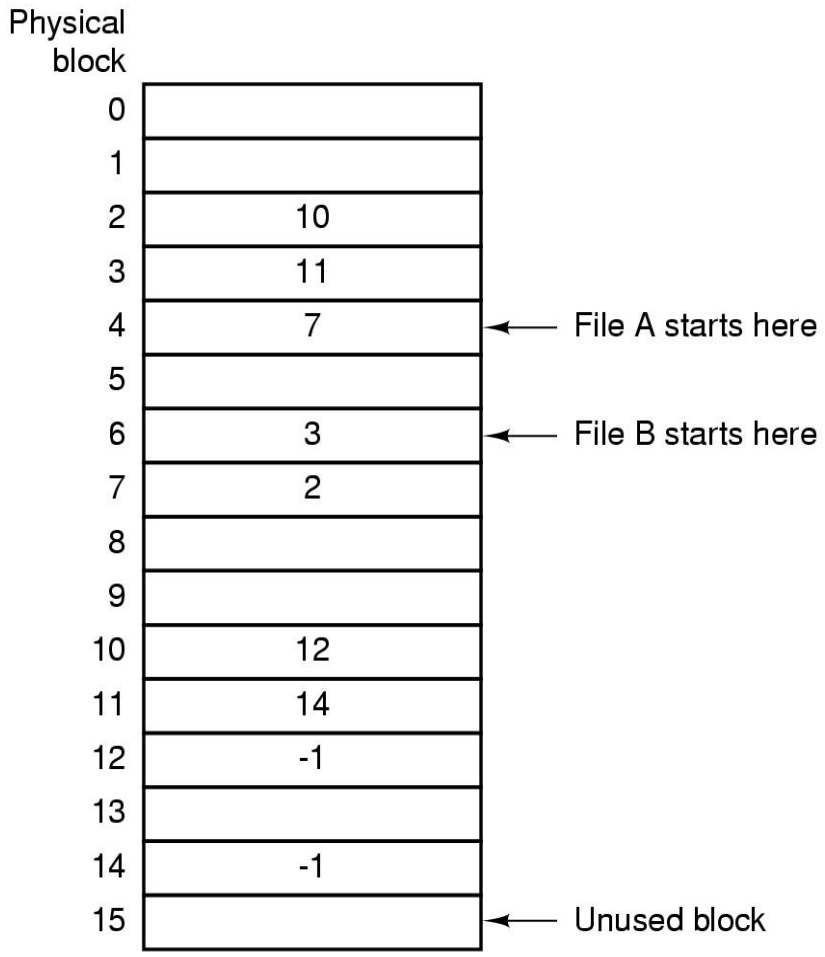
Idea: Store the block headers in their own table, separately from the data

Advantages:

- Random Access is easy/fast if table fits in memory
- Blocks no longer have an awkward size
- Still avoids external fragmentation because blocks do not need to be contiguous

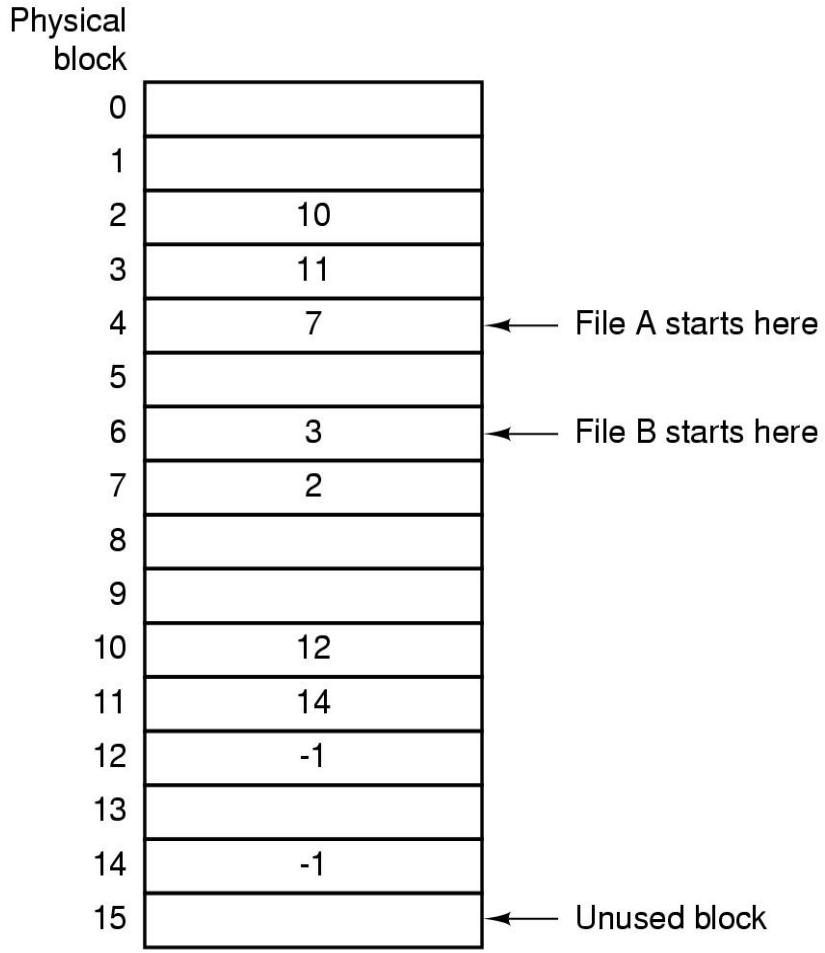
Disadvantages:

- FAT tables easily become too big
- Example: Suppose we have 1 KB blocks and 8 bytes per entry. How big would the table be for a TiB drive ?



Linked list allocation using FAT in RAM

Exercise: Visualize the blocks comprising file B



Q: Why is the FAT table often faster than LL?

Approach #4: inodes

Inodes is an abbreviation of **index nodes,** or **i-nodes.**

Idea: An inode lists the attributes and disk locations for a file, but only needs to be loaded into memory *when the file is open*.

Example: If an inode is 128 bytes (n) and 4 files (k) are open, how many bytes are needed?

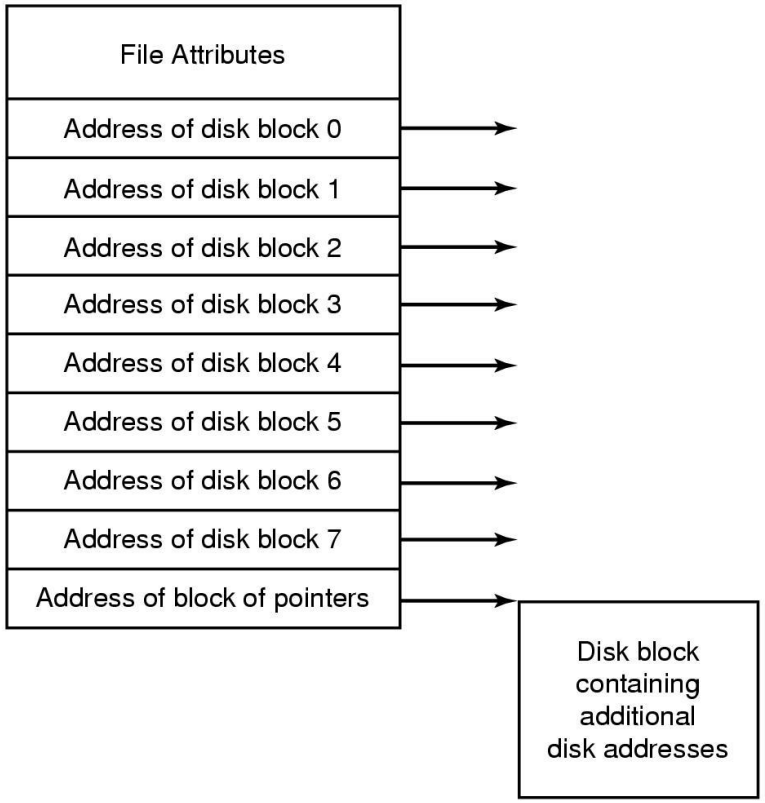
In general: n*k bytes are needed

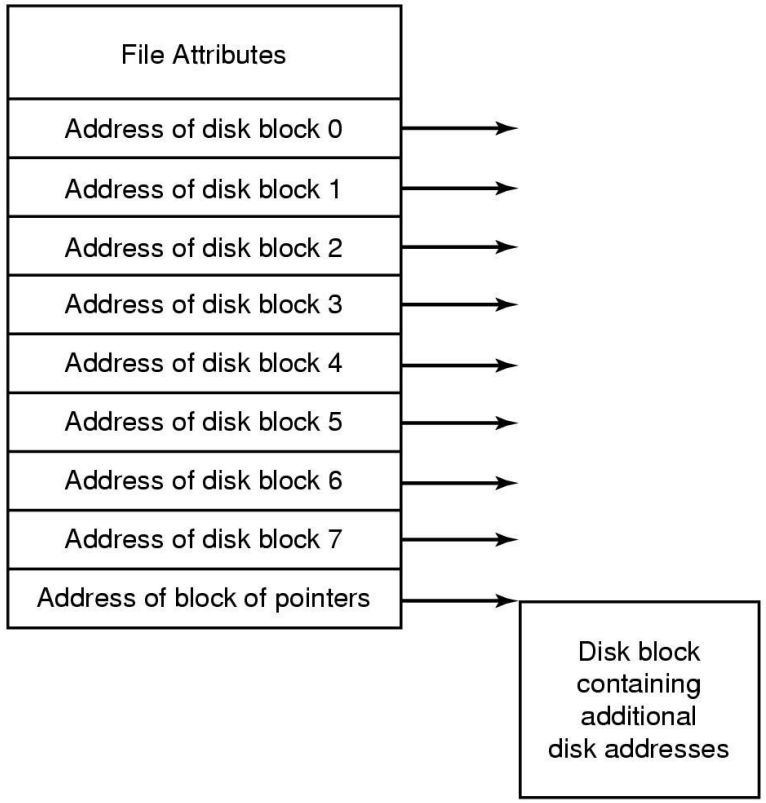
Advantages:

- Need far less space
- Random access is still easy/fast because the inode table fits in memory
- Avoids external fragmentation because blocks do not need to be contiguous

Example: inode

Visualize the inodes for the files A and B from the previous examples. Assume 1 KB blocks as before.





Summary: File System Layouts

- Goals
  - Fast sequential access
  - Fast random access
  - Ability to dynamically grow
  - Minimum fragmentation

- Standard schemes
  - Contiguous allocation
  - Linked list allocation
  - Linked list with file allocation table (FAT)
  - Linked list with Indexing (I-nodes)