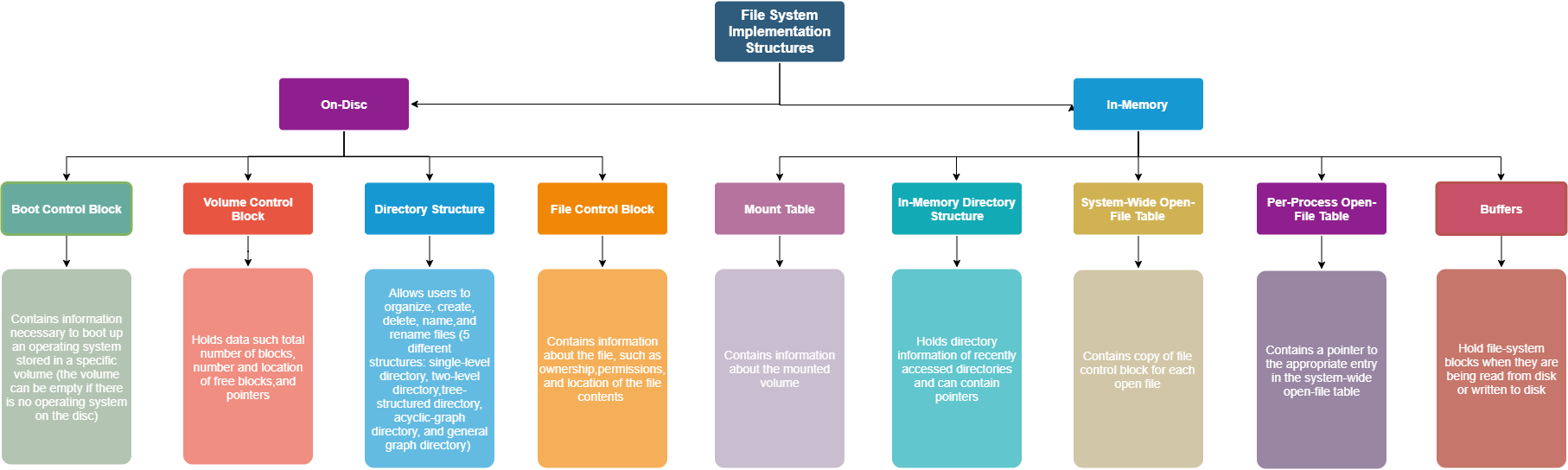
**Week 4 Interactive Assignment: Outline the objectives and functions of file systems management and the supported operations, including their reliability and performance.**

A file system is a logical storage unit that holds a collection of related information. A file system is in secondary storage, as on a disk, and serves as an interface between the user and the disk where the file system is stored. Two major objectives of the file system are keeping track of where file location, and optimization of the storage and retrieval of data (Columbia University, n.d.). According to Silberschatz, Galvin, and Gagne (2014), “The file system consists of two distinct parts: a collection of files, each storing related data, and a directory structure, which organizes and provides information about all the files in the system.”

A file is the smallest allocation of logical secondary storage. Files can store various types of data including numeric, character, and binary. The creator of the file determines the contents of a file, such as text files, source files, and executable files. In addition to data, files may also contain entire programs. A file system manages file attributes including name, identifier (a unique number used to reference files), type (directing the OS to applications or systems that support that file type), location (a pointer to a file location on secondary storage), size (amount of data stored in the file), protection (determines permissions for reading, writing, and executing files), time, date, and user identification (for usage monitoring) (Columbia University, n.d.).

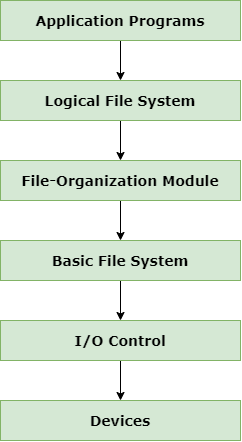
File systems also manage file operations. These operations include creating a new file, writing to a file (or pointer location), reading a file (at pointer location), repositioning within a file (seeking a specific value within the file), deleting a file, truncating a file (erasing the contents of a file, but keeping its attributes), opening a file (searching the directory structure and then pulling it into main memory), closing a file (moving file content from main memory back onto the disk) (Geeks, n.d.). To manage these file operations, the file system controls I/O transfers between the memory and disk in units of blocks (Silberschatz, Galvin, & Gagne, 2014). Specific structures in-memory and on-disk allow these transfers to occur.

On-disk structures that contribute to the implementation of the file system through the OS include: a boot control block (containing the information necessary to boot up an OS stored there), a volume control block (holding data such as the total number of blocks, the number and location of free blocks, and pointers), a directory structure, and a per-file file control block. The file control block contains information about the file, such as ownership, permissions, and location of the file contents (Silberschatz, Galvin, & Gagne, 2014). Regarding the purpose of in-memory structures that aid implementation of the file system, Silberschatz, Galvin, & Gagne (2014) state that, “The in-memory information is used for both file-system management and performance improvement via caching. The data are loaded at mount time, updated during file-system operations, and discarded at dismount. Several types of structures may be included.” These in-memory structures are the mount table (containing mounted volume information), an in-memory directory structure cache (holding directory information of recently accessed directories and may contain pointers), a system-wide open-file table (containing a copy of the file control block for each open file), a per-process open-file table (containing a pointer to the appropriate entry in the system-wide open-file table), and buffers (holding file-system blocks while being read from or written to disk) (Silberschatz, Galvin, & Gagne, 2014).



**Figure 1: File System Implementation Structures**

One requirement of operating systems is the ability to support various types of file systems. One way of accommodating various types would be to write directory and file routines for each type; however, according to Operating System Concepts Essentials (2nd ed.) (2014), most operating systems “use object-oriented techniques to simplify, organize, and modularize the implementation.” File system implementation is broken down into three major layers: the file-system interface (based on the open, read, write, and close calls and on file descriptors), the virtual file system layer (separating file-system-generic operations from their implementation and providing a mechanism for uniquely representing a file throughout a network), and the layer implementing the local or remote-file-system protocol. Silberschatz, Galvin, and Gagne (2014) describe the layered design of a file system in Figure 2 below, in which “Each level in the design uses the features of lower levels to create new features for use by higher levels.” (Silberschatz, Galvin, & Gagne, 2014).

****

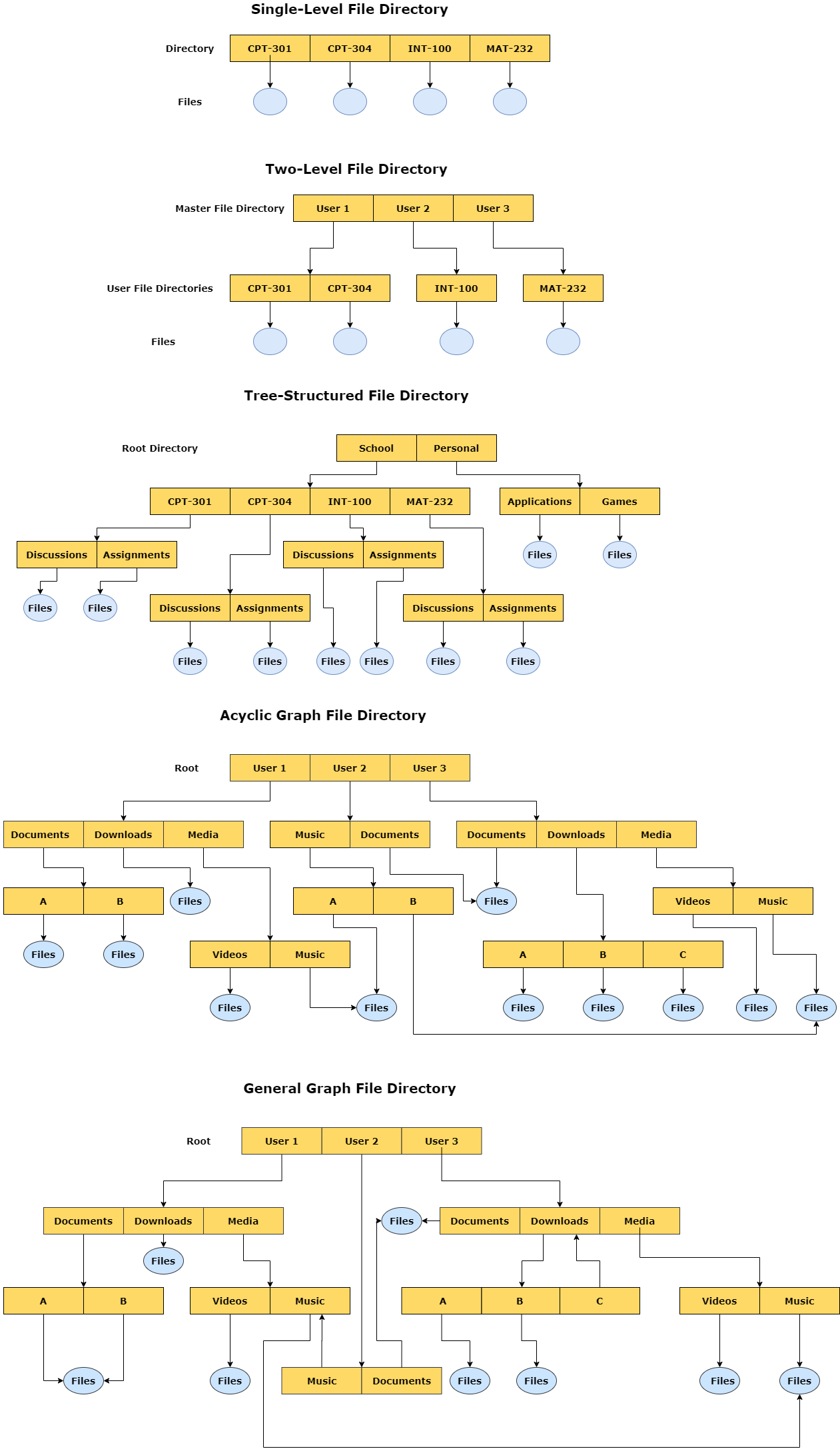
**Figure 2: Layered File System**

**Contrast different directory structures and create unique diagrams to illustrate an example of each directory structure.**

A file system consists of two parts: the collection of files and a directory structure. A file directory is an abstraction from the hardware, which allows users to organize, create, delete, name, and rename files. Users can also exploit the file directory in order to list out the contents of a directory, search for a file, and traverse the files system. There are five common schemes for structuring directories: single-level directory, two-level directory, tree-structured directory, acyclic-graph directory, and general graph directory (Silberschatz, Galvin, & Gagne, 2014). A single-level directory contains all the files in one large directory. This is the simplest scheme but becomes a problem with large numbers of files or multiple users. Since each file needs to have a unique name, and multiple users might name their files similarly, it becomes difficult to remember and locate files.

Providing a partial solution to the challenges of a single-level directory, the two-level directory contains user file directories within the master, so that each user maintains their own file directory. This scheme avoids the issue of multiple users trying to utilize the same file name; however, a disadvantage of this approach is that it isolates users from one another, preventing the sharing of files. The third directory scheme is a tree-structured directory. This approach is an extension of the single- and two-level directories in that users can create as many levels to the tree as desired through the creation of subdirectories. File paths are often used to trace the location of a file within these branching directories. According to Silberschatz, Galvin, and Gagne (2014), “A tree is the most common directory structure. The tree has a root directory, and every file in the system has a unique path name.”

The fourth file directory scheme is an acyclic-graph directory. This approach promotes collaboration and allows users to share a subdirectory in a file system by allowing the subdirectory to exist in two places at once (in both users’ independent directories). In this model, changes made to a shared file by one user will be immediately apparent to the other user when they view the shared file from their own shared subdirectory. In addition, a new file created by one user will be visible in all shared subdirectories. One of the challenges of the acyclic-graph directory is that distinct file names can refer to the same file (a file has multiple absolute path names). Another challenge lies in deletion, since pointers may remain, which then point to a non-existent file. The final directory scheme is the general graph directory. Unlike the acyclic-graph directory, this model allows cycles. While the acyclic-graph directory algorithm for traversing the graph for file references is simple, the general graph directory search algorithm is more complex in that it is designed to avoid searching the same component of the directory twice. This scheme is more flexible, but disadvantages include high cost and the need for garbage collection to manage deleted files (Geeks, n.d.).

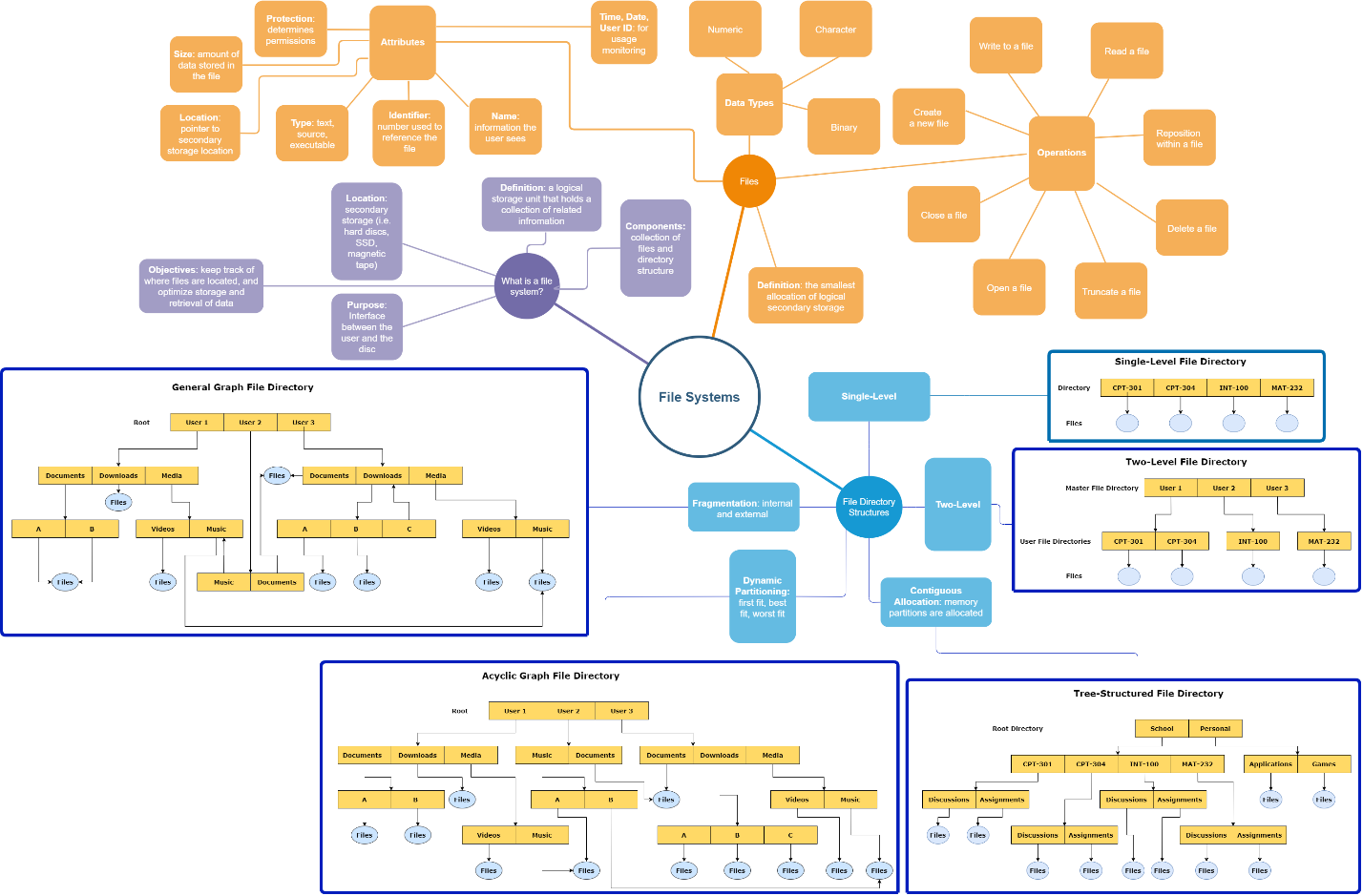


**Figure 3: Directory Structures**

**Describe different types of input/output devices, distinguishing between the hardware and software layers and summarizing the integration across I/O and memory components.**

The operating system provides an abstraction for the two main functions of a computer, I/O and processing. Regarding I/O, the operating systems manages I/O operations and devices. Since I/O devices vary greatly, the methods to control them form an I/O subsystem of their own in the kernel. Even though the range of I/O devices has grown over time, the software/hardware interface across systems has become more standardized. While it seems difficult to integrate new devices with an older system, device drivers help resolve this issue by providing a standardized interface between devices and the I/O subsystem, and system calls provide an interface between applications and the operating system (Silberschatz, Galvin, & Gagne, 2014).

Most I/O devices either serve as storage (CD-ROM’s), modes of transmission (Bluetooth), or user interface devices (mouse, keyboard, etc.). Other I/O devices are specialized. The hardware components of I/O devices consist of buses, device controllers, I/O ports, registers, and the devices themselves. Buses are a group of wires that send messages via electrical voltages according to a specified protocol. Device controllers are electronics that control a port, bus, or device. A port is the connection point between a device and computer. The registers contain data and control signals that allow the processor and controllers to communicate with one another. This is where the hardware/software interface come into play. Specialized I/O instructions initiate buses that select the desired device and move bits in or out of device registers (Silberschatz, Galvin, & Gagne, 2014). Device controllers can employ memory-mapped I/O in which “device-control registers are mapped into the address space of the processor” (Silberschatz, Galvin, & Gagne, 2014). Then I/O requests are executed by the CPU. Computers communicate with I/O devices through signals exchanged either in the air or through cables.



**Figure 4: File Systems Concept Map**

**References**

Columbia University. (n.d.) Unit 4: Application and System Software. Retrieved October 4, 2018 from http://www.columbia.edu/itc/hs/medical/dbmi\_hit\_cert/ONC\_Comp\_4/comp4\_unit4-3\_lecture\_handout.pdf

Geeks for Geeks. (n.d.). File Systems: Operating Systems. Retrieved October 4, 2018 from https://www.geeksforgeeks.org/file-system-operating-systems/

Silberschatz, A., Galvin, P. B., & Gagne, G. (2014). Operating system concepts essentials (2nd ed.). Retrieved from https://www.vitalsource.com