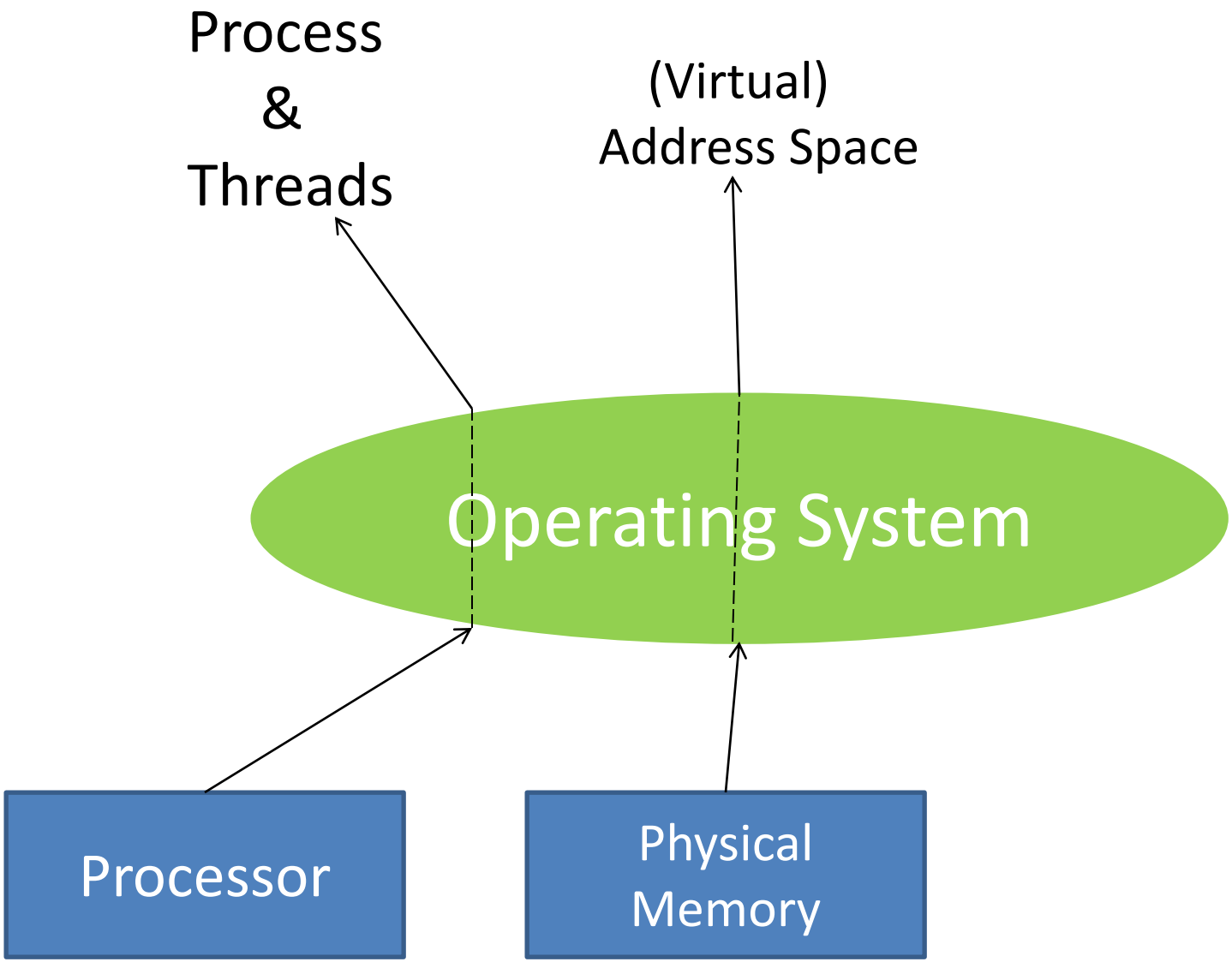
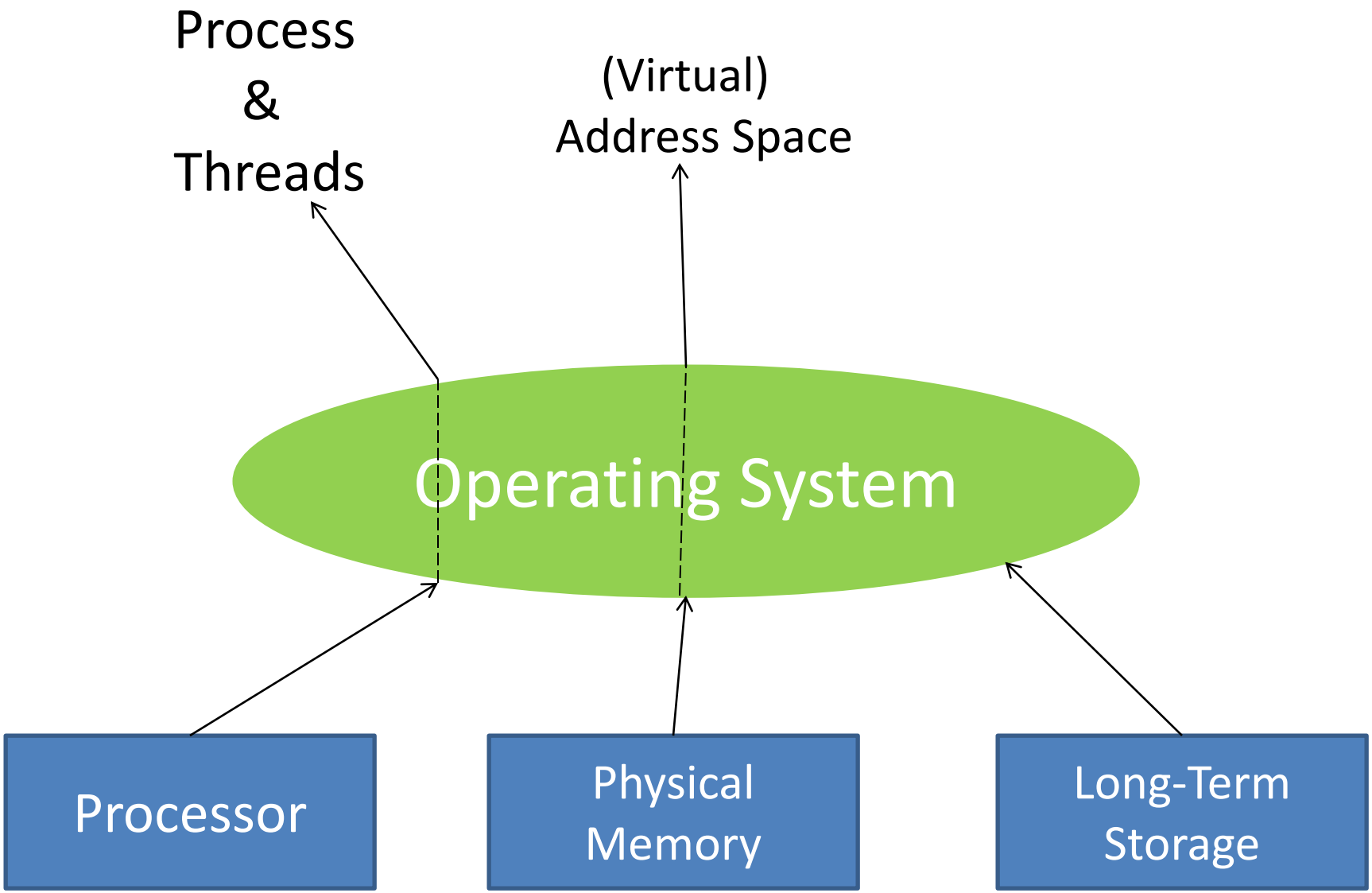
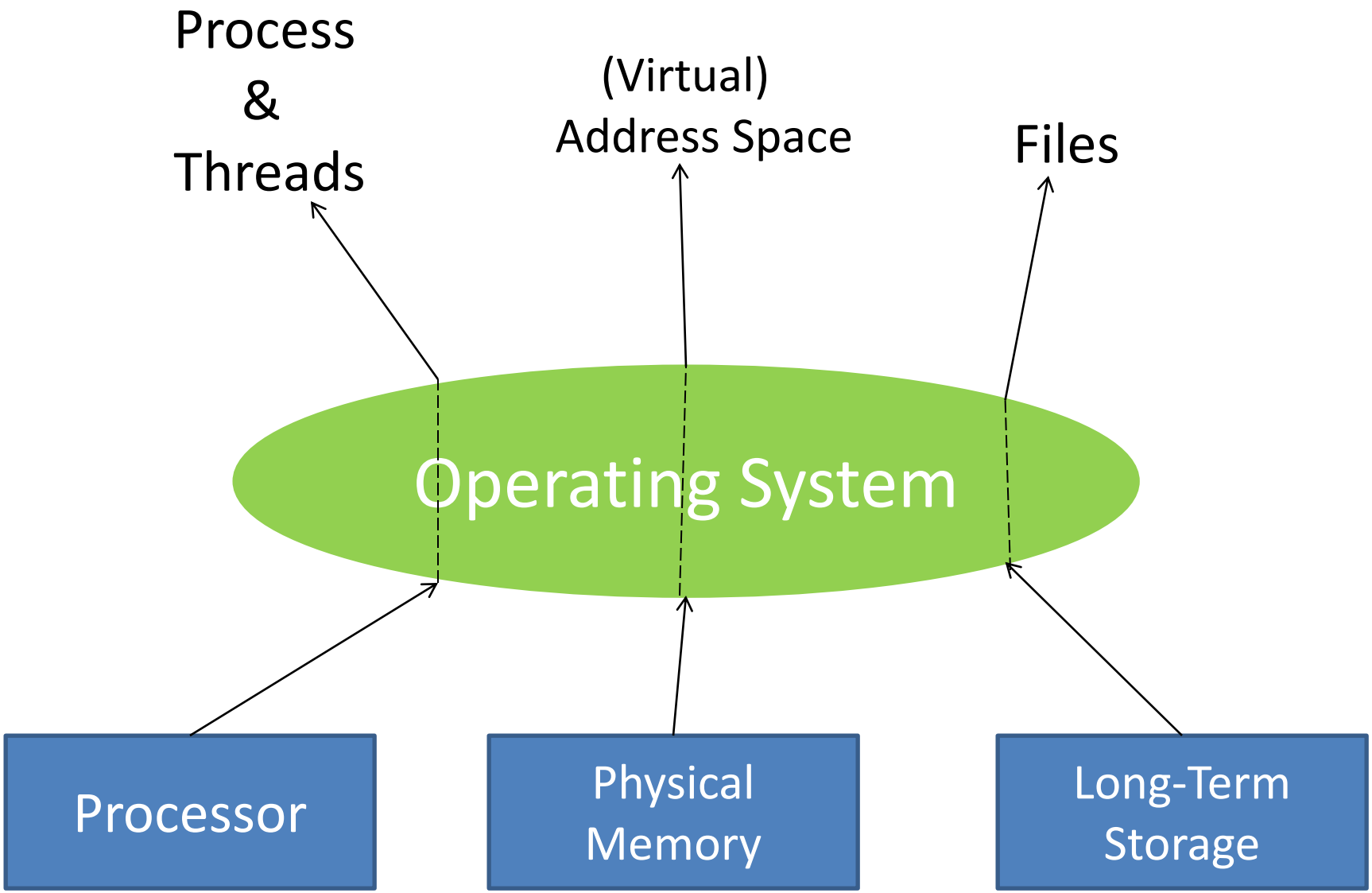
Operating Systems

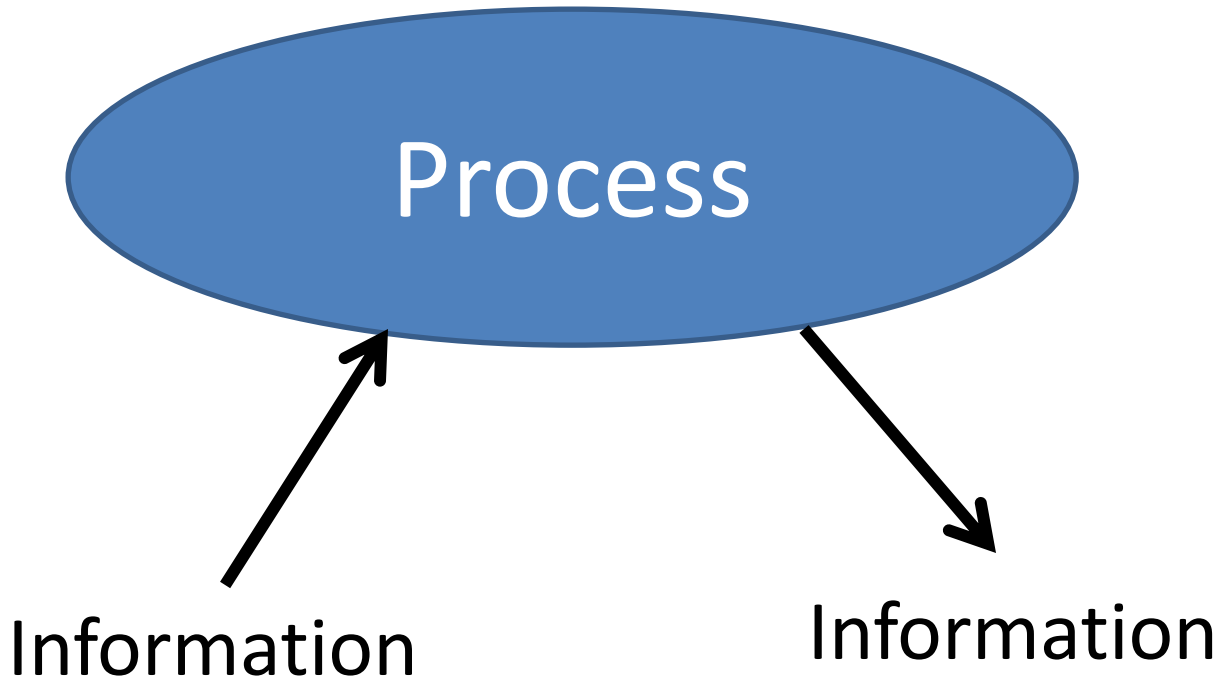
File Systems I

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com


Process
&
Threads
(Virtual)
Address Space
Operating System
Processor
Physical
Memory


Process
&
Threads
(Virtual)
Address Space
Operating System
Processor
Physical
Memory
Long-Term
Storage


Process
&
Threads
(Virtual)
Address Space
Files
Operating System
Processor
Physical
Memory
Long-Term
Storage



Is it OK to keep this information only in the process address space?

Shortcomings of Process Address Space

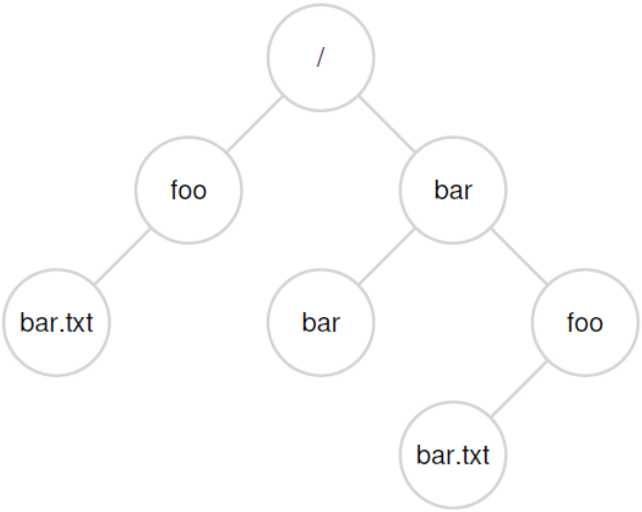
- Virtual address space may not be enough storage for all information.
- Information is lost when the process terminates, is killed, or the computer crashes.
- Multiple processes may need the information at the same time.

Requirements for Long-term Information Storage

- Store very large amount of information.
- **Persistence:** Information must survive the termination of the process using it.
- Multiple processes must be able to access the information concurrently.

Two Key Abstractions for Storage

- Files
  - Simply a linear array of bytes
  - In most systems, the OS does not know much about the structure of the file.
- Directories



Directories and files have low-level names beside the name used by the user.

Issues with long term storage

- How do you find information?
- How do you keep one user from reading another user's data?
- How do you know which blocks are free?

For now, and till we discuss disks in I/O lectures, let's assume a disk consists of a **linear sequence of fixed-size blocks** supporting two operations: read block x and write block x.

Files

Files

- A file is:
  - Data collections created by users
  - Logical unit of information created by processes
- Desirable properties of files:

Long-term existence

- files are stored on disk or other secondary storage and do not disappear when a user logs off

Sharable between processes

- files have names and can have associated access permissions that permit controlled sharing

Structure

- files can be organized into hierarchical or more complex structure to reflect the relationships among files

Files

- Used to model disks instead of RAM
- Information stored in files must be persistent (i.e. not affected by processes creation and termination)
- Managed by OS
- The part of OS dealing with files is known as the file system

Files from The User's point of View

- Files
  - Naming
  - Structure
  - Types
  - Access
  - Attributes
  - Operations
- Directories
  - Single-level
  - Hierarchical
  - Path names
  - Operations

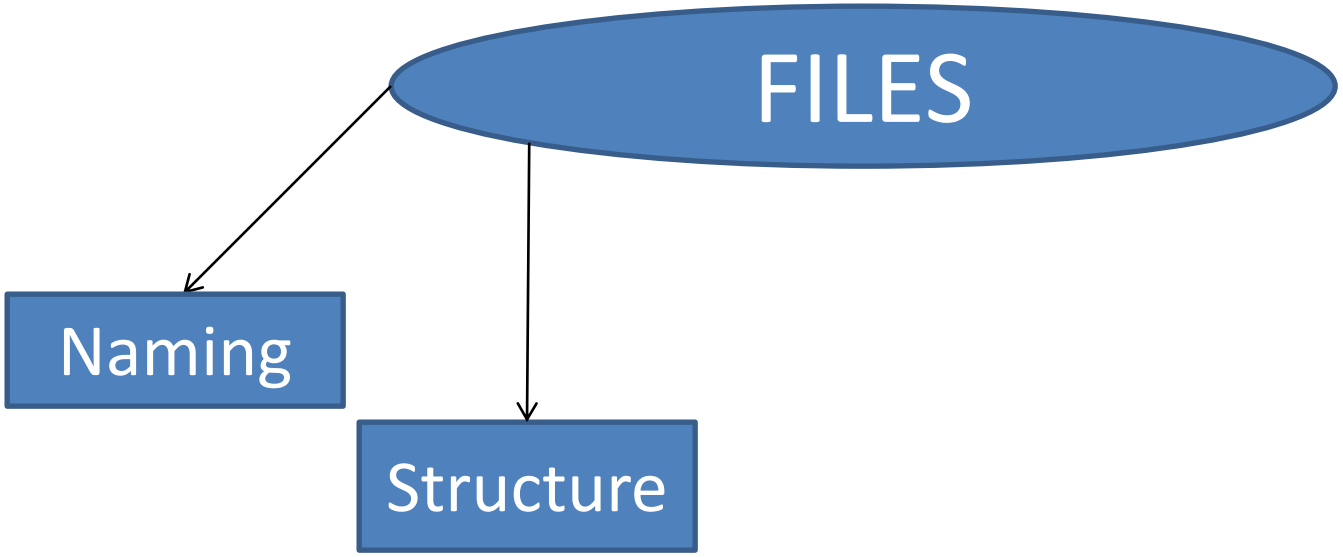
File Systems

- Provide a means to store data organized as files as well as a collection of functions that can be performed on files
- Maintain a set of attributes associated with the file
- Typical operations include:
  - Create
  - Delete
  - Open
  - Close
  - Read
  - Write

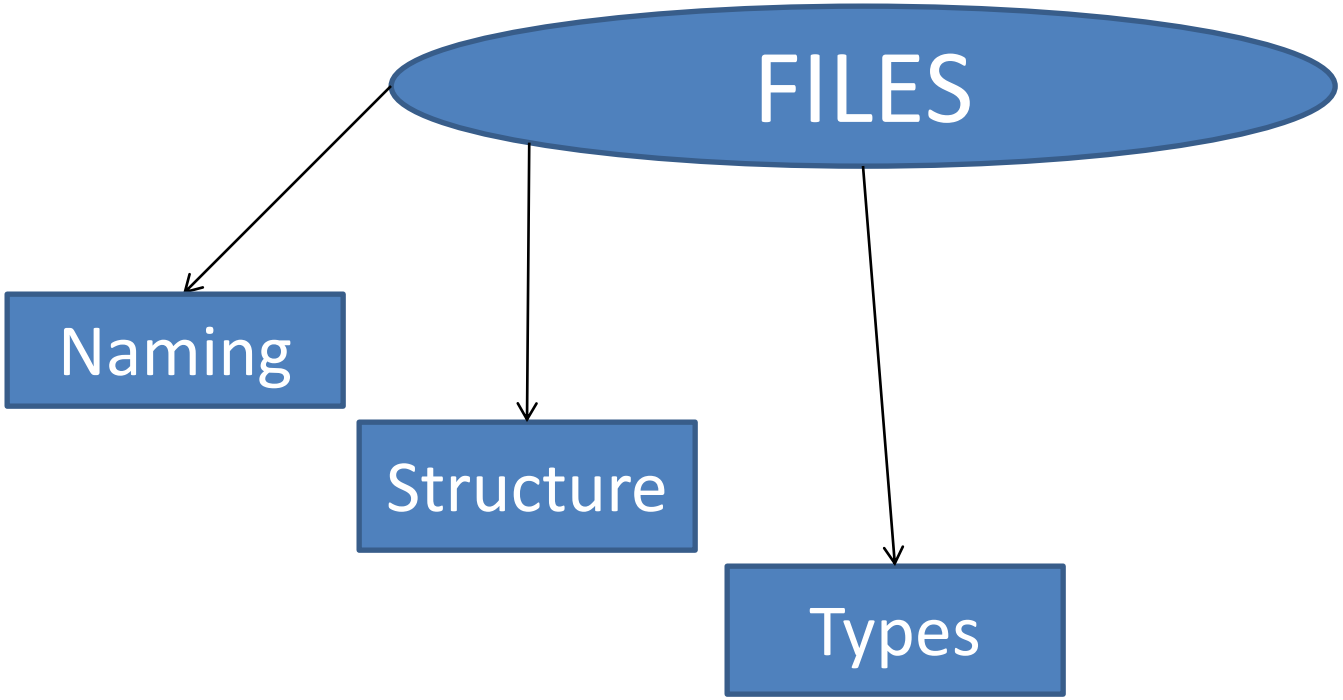
FILES

Naming

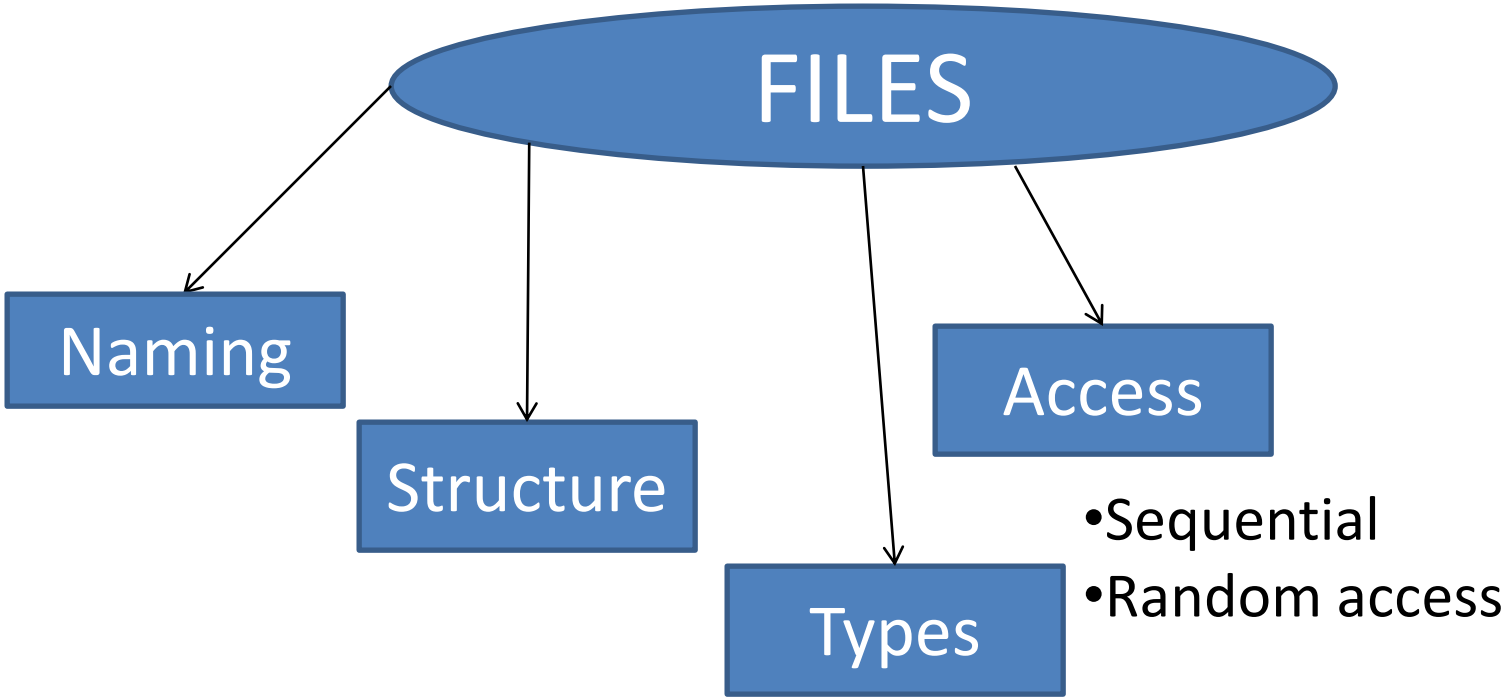
- Shields the user from details of file storage.
- In general, files continue to exist even after the process that creates them terminates.

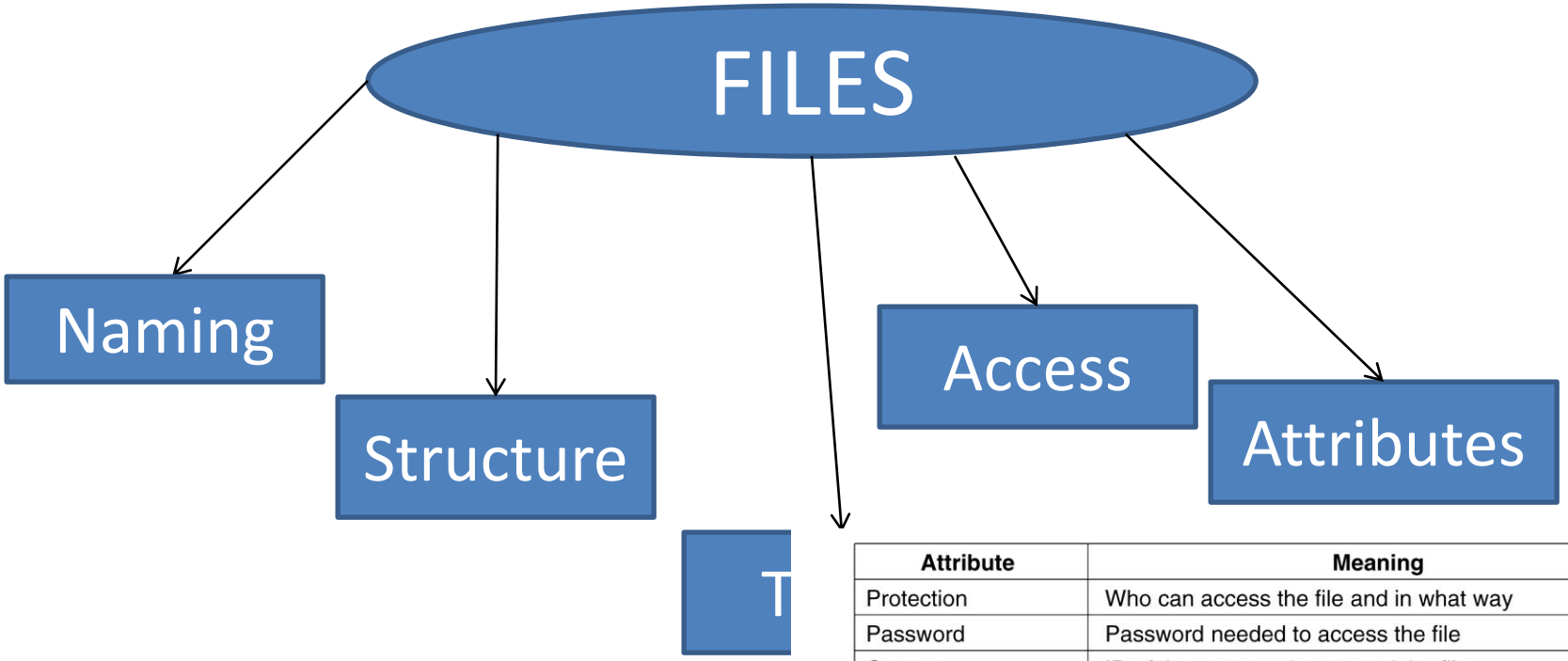


- OS sees it as a linear stream of bytes.
- Users can have different views (e.g. record, …)
- Programs can impose structures (e.g. jpg, mp3, …)

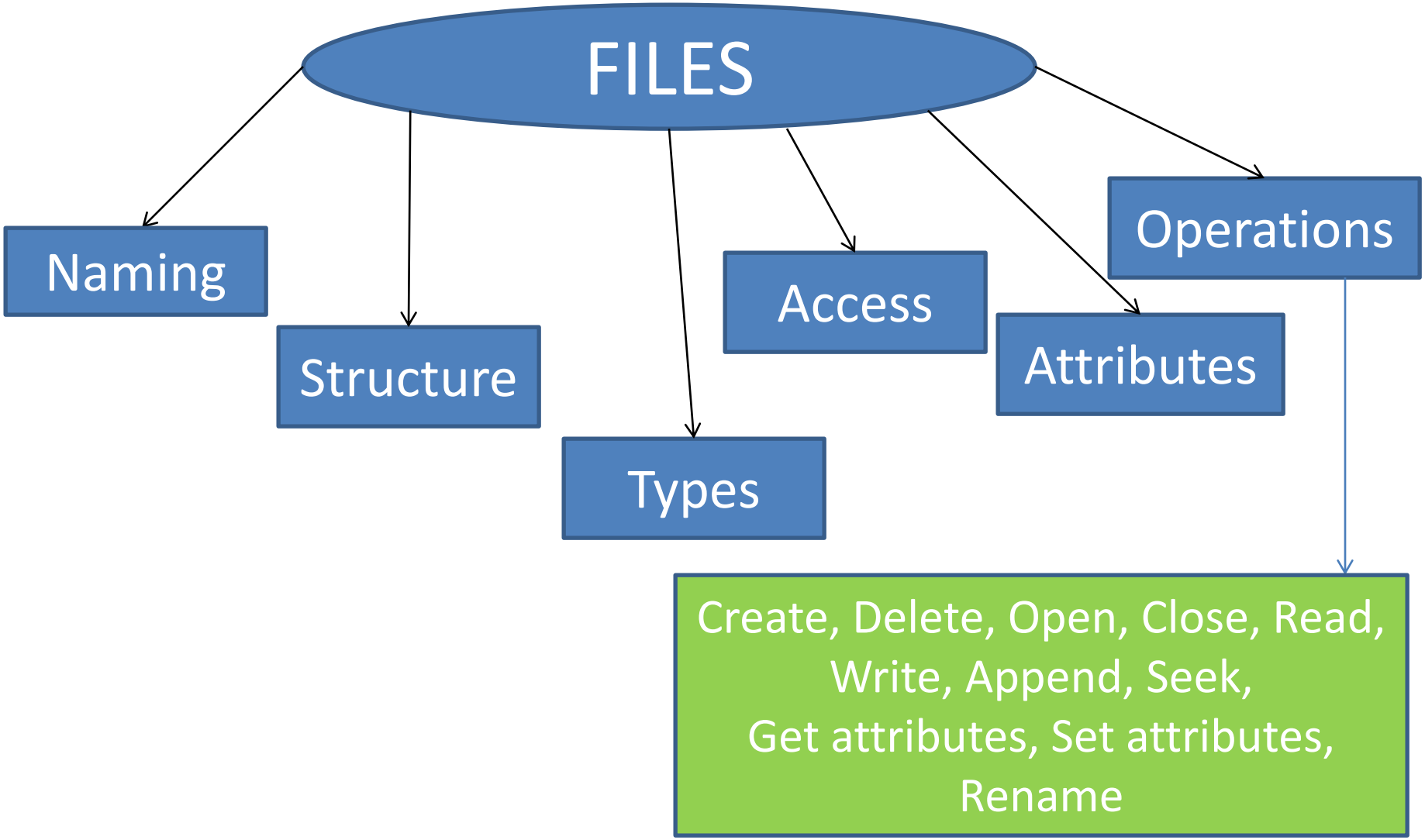


- Regular files
  - ASCII
  - Binary
- Character special
  - to model serial devices (printers, networks, …)
- Block special
  - to model disks


FILES
Naming
Structure
Types
Access
•Sequential
•Random access



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 was last changed
Current size	Number of bytes in the file
Maximum size	Number of bytes the file may grow to


FILES
Naming
Structure
Types
Access
Attributes
Operations
Create, Delete, Open, Close, Read, Write, Append, Seek, Get attributes, Set attributes, Rename

Dealing with Files in C
(As an Example)

First step

- Declaration:

```
FILE *fptr1, *fptr2 ;
```

Opening Files

- The statement:

  fptr1 = **fopen** (“filename", "r") ;

  would open the file filename for input (reading).

  - r: read
  - w: write
  - a: append
  - … there are some more

Testing for Successful Open

- If the file was not able to be opened, then the value returned by the *fopen* routine is NULL.
- For example, let's assume that the file *mydata* does not exist. Then:

```
FILE *fptr1 ;
fptr1 = fopen ( "myfile", "r") ;
if (fptr1 == NULL)
{
        printf ("File 'mydata' did not open.\n") ;
}
```

Reading From Files

- In the following segment of C language code:

```
int a, b ;
FILE *fptr1, *fptr2 ;
fptr1 = fopen ( "mydata", "r" ) ;
fscanf ( fptr1, "%d%d", &a, &b) ;
```

the *fscanf* function would read values from the file "pointed" to by *fptr1* and assign those values to *a* and *b*.

End of File

- The end-of-file indicator informs the program when there are no more data (no more bytes) to be processed.
- There are a number of ways to test for the end-of-file condition. One is to use the *feof* function which returns a true or false condition:

```
fscanf (fptr1, "%d", &var) ;
if ( feof (fptr1) )
{
        printf ("End-of-file encountered.\n");
}
```

- Another (better) way of testing EOF:

```
while(fscanf(fp,"%d ", &current) == 1)
{

}
```

Writing To Files

```
int a = 5, b = 30;
FILE  *fptr2 ;
fptr2 = fopen ( “filename", "w" ) ;
fprintf ( fptr2, "%d %d\n", a, b ) ;
```

the fprintf functions would write the values stored in *a* and *b* to the file "pointed" to by *fptr2*.

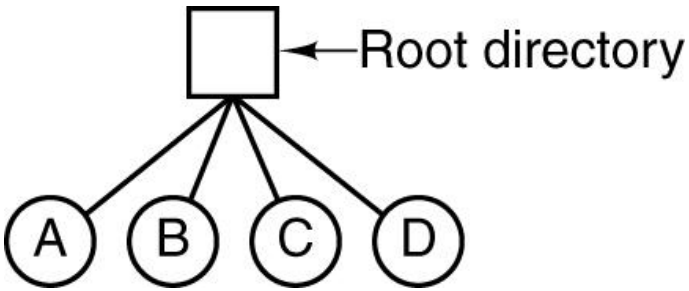
Closing Files

```
fclose ( fptr1 ) ;
```

Once the files are open, they stay open until you close them or end the program (which will close all files.)

Directories

Directories: Single-Level Directory Systems



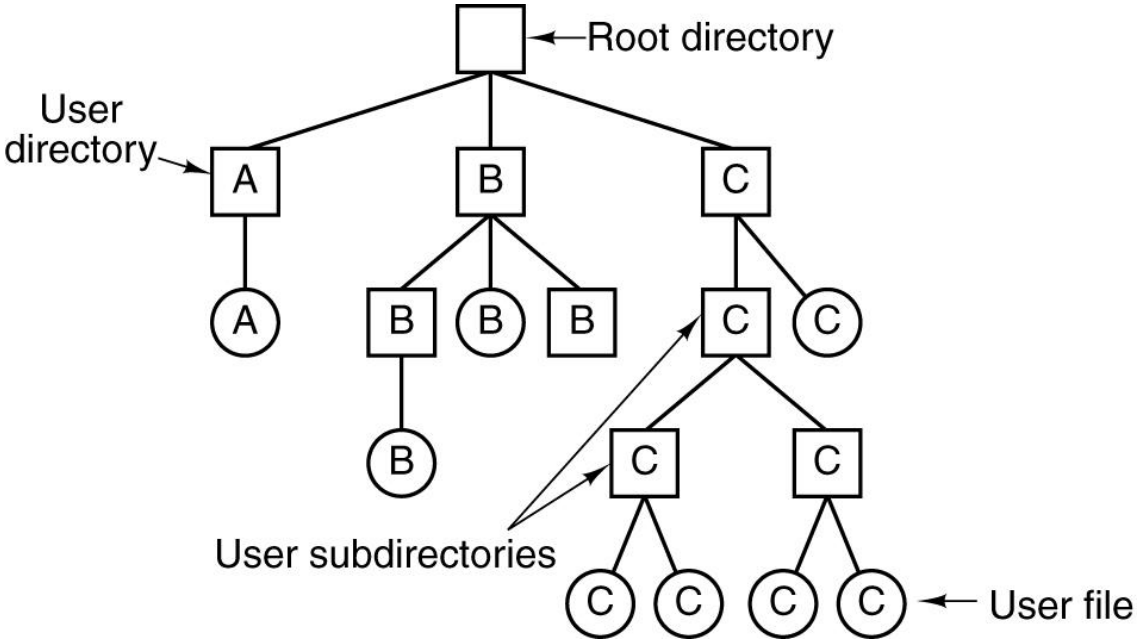
+ Simplicity

-Not adequate for large number of files.

Used in simple embedded devices

Directories: Hierarchical Directory Systems

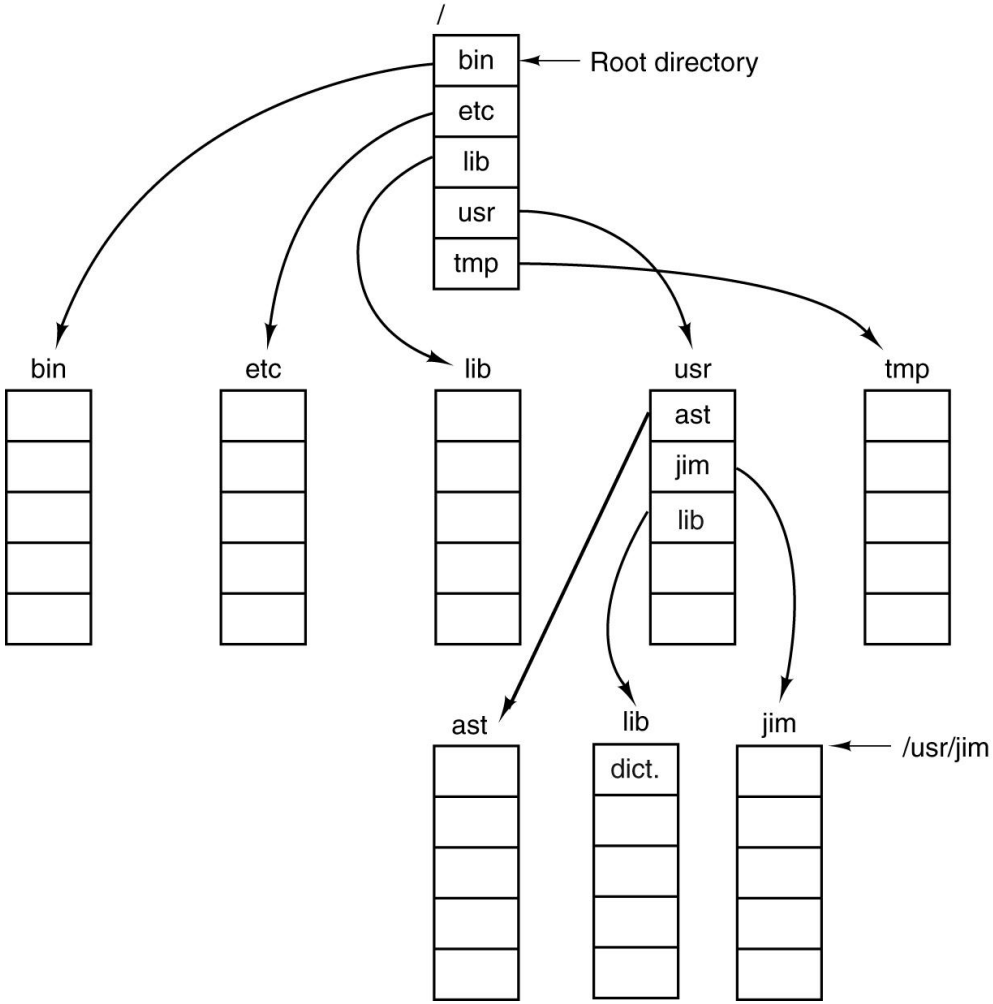
- Group related files together
- Tree of directories



Directories: Path Names

- Needed when directories are used
- Absolute path names
  - Always start at the root
  - A path from the root to the specified file
  - The first character is the separator
- Relative path names
  - Relative to the working directory
  - Each process has its own working directory

Directories: Path Names



Directories: Operations

- More variations among OSes than file operations
- Examples (from UNIX):
  - Create, delete
  - Opendir, closedir
  - Readdir
  - Rename
  - Link, unlink

Conclusions

- Files are OS abstraction for storage, same as address space is OS abstraction for physical memory, and processes (& threads) are OS abstraction for CPU.
- In this lecture we discussed files from user perspective. Next lecture we will discuss the implementation.