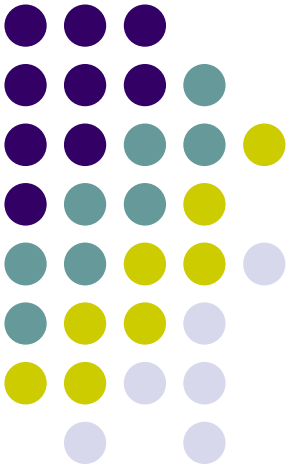
File Structures

Ch02. Fundamental File Processing Operations



2020. Spring

Instructor: Joonho Kwon

jhkwon@pusan.ac.kr

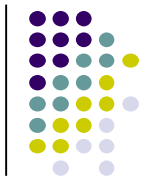
Data Science Lab @ PNU

Slides are based on the textbook.

datalab

data science laboratory

References (1/2)



- **C++ Programming Language**
  - **Stream IO and File IO**
    - https://www3.ntu.edu.sg/home/ehchua/programming/cpp/cp10_IO.html
  - **Input/output with files**
    - http://www.cplusplus.com/doc/tutorial/files/

References (2/2)

- Operating Systems: Three Easy Pieces
  - http://pages.cs.wisc.edu/~remzi/OSTEP/
    - Ch39. Files and Directories
    - Ch40. File System implementation

Contents

Files

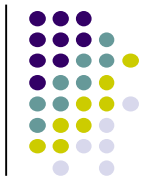
- **Sequence of bytes… nothing more, nothing less**
- **File System** resides on secondary storage (disks)

Physical files and Logical files

- Physical file
  - a file that actually exists on secondary storage
  - a file as known by OS and that appears in file directory

- Logical file
  - a file as seen by program
  - allow program to describe operations to be performed on file not knowing what physical file will be used

Connections between Physical files and Logical files



- Before the program can open a file for a use
  - OS makes a hookup between a logical file (e.g. a phone line) and some physical file or device
- History
  - At Main frame era: by Job Control Language
  - At Unix and DOS era: by the instructions within the program (O/S system calls or parts of PLs)
    - Cobol code:

```
select inp_file assign to "myfile.dat"
```

    - inp_file: logical file, myfile.dat: physical file

System calls for files

create()	
open(), close()	
read(), write()	
lseek()	move offset
stat()	get inode content

- All others are library functions
  - eg scanf(), gets(), getchar(), …..

Opening files (1/2)

- Two options
  - (1) open an existing file
  - (2) create a new file, deleting any existing contents in the physical file
- Action
  - Positioned at the beginning of the file
  - Ready to start reading or writing
  - The file contents are not disturbed by the open statement

Opening files (2/2)

fd = **open** (filename, flags [pmode]);

- Open system call
  - fd: file descriptor
  - filename: physical file name
  - flags: O_APPEND, O_CREAT, O_EXCL, O_RDONLY, O_RDWR, O_TRUNC, O_WRONLY
  - pmode: in O_CREAT, pmode is required protection mode

Pmode = 0751 = 111 101 001

{owner, group, world × read, write, execute}

Opening files: Examples

```
fd = open(fname, O_RDWR|O_CREAT,0751)
```

- Opens an existing file for reading/writing or
- Creates a new one if necessary

```
fd = open(fname,O_RDWR|O_CREAT|O_TRUNC,0751)
```

- Creates a new file for reading and writing
- If “fname” exists, its contes are truncated

```
fd = open(fname, O_RDWR|O_CREAT|O_EXCL,0751)
```

- Create a new file only if there is not “fname”
- A file exists → not opened, returns a negative value

Closing Files

- Files are closed automatically by the OS when a program terminate normally
- *CLOSE* statement is needed
  - only as protection against data loss in the event of program interruption and to free up logical filenames for reuse

```
close (fd);
```

Reading & Writing(1)

- Input / Output operation
  - low-level system call (Unix)
    - read(Source_file, Destination_addr, Size)
    - write(Destination_file, Source_addr, Size)
  - C streams (in stdio.h)
    - file = fopen(filename, type);
    - fread, fget, fwrite, fput, fscanf, fprintf

System call v.s. Library call

in kernel	*in* a.out (user)	
system call	**library call**	
read()	scan**f**() format	tty files
	getchar() char	
	get**s**() string	
	fsacnf()	all files
	fgetc()	
	fgets()	
	fread() any number	
fd	*FILE (struct in lib)	

Read and Write Functions

- Read()
  - Source_file: logical file name for where it is read from
  - destination_addr: where to place information it reads from the input file
  - Size: a byte count

Read (source_file, destination_addr, Size);

- Write()
  - destination_file: logical file name used for sending the data
  - Source_addr: where to find information
  - Size: a byte count

Write (destination_file, Source_addr, Size);

Display the contents of a file

- Steps (Pseudo code)
  - 1. Display a prompt for the name of the input file
  - 2. Read the user's response from the keyboard into a variable called filename
  - 3. Open the file for input
  - 4. While there are still characters to be read form the input file
    - A. read a character from the file
    - B. write the character to the terminal screen
  - 5. Close the input file

Files with C Streams by example

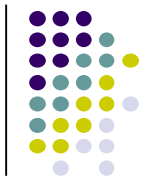
- Code

```c
#include <stdio.h>

int main()
{
    char ch;
    FILE *file;          //  pointer to file descriptor
    char filename[20];
    printf("Enter the name of the file: ");      // step1
    gets(filename);                              // step2
    file = fopen(filename, "r");                 // step3
    while ( fread (&ch, 1, 1, file) != 0)        // step 4a
        fwrite(&ch, 1, 1, stdout);               // step 4b
    fclose(file);                                // step 5
    return 0;
}
```

- Compile and execute it

```
$ gcc -o lsc listc.c
$ ./lsc
Enter the name of the file: test.txt
....
```

Files with C++ Streams classes (1/3)



- Stream classes
  - Support open, close, read and write operations
  - cin and cout
    - Predefined stream objects for standard input and standard output files
  - ofstream: Stream class to write on files
  - ifstream: Stream class to read from files
  - fstream
    - The main class for both read and write from/to to files
    - Two constructors and a wide variety of methods

Files with C++ Streams classes (2/3)

```
fstream();   // leave the stream unopened
fstream (char *filename, int mode);
int open (char *filename,  int mode);
int read (unsigned char* dest_addr, int size);
int write (unsigned char* source_addr, int size );
```

- mode
  - ios::in, ios::out, ios::binary
  - ios::ate, ios::app, ios::trunc

```
ofstream myfile;
myfile.open ("example.bin", ios::out | ios::app | ios::binary);
```

```
ofstream myfile ("example.bin", ios::out | ios::app | ios::binary);
```

Files with C++ Streams classes (3/3)

- Detecting end-of-file
  - In C, fread() returns zero when reached the end of file
  - In C++, **fstream::fail()** returns true if the previous operation on the stream failed

Files with C++ Streams

- Code
  - list.cpp

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    char ch;
    fstream file;          //  declare unattached fstream
    char filename[20];
    cout<< "Enter the name of the file: "       // step1
        << flush;                 // fource output
    cin>>filename;                                // step2
    file.open(filename, ios::in);                 // step3
    // include white space in read
    file.unsetf(ios::skipws);
    while (1)
    {
        file >> ch;                               // step 4a
        if (file.fail()) break;
        cout << ch;                               // step 4b
    }
    file.close();                                 // step 5
    return 0;
}
```

Seeking

- sequential IO
  - Every time a byte is read
    - OS moves the read/write pointer ahead
    - Ready to read the next byte
- Seeking
  - moving directly to a certain position in a file
  - lseek(Source_file, Offset, Origin) in Unix
    - Offset - the pointer moved from the start of the source_file

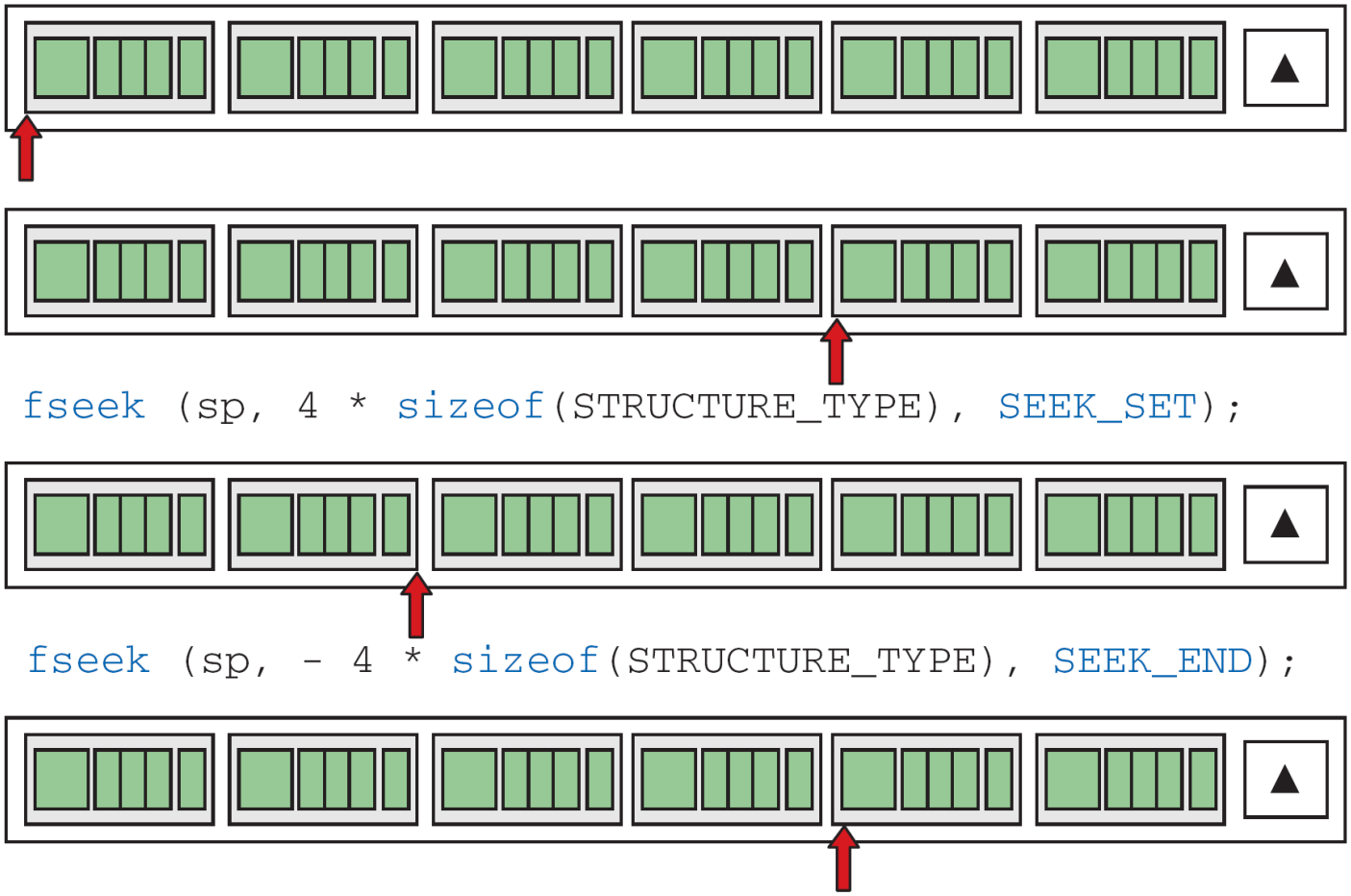
File Seek Operation in C (1/2)

- File is viewed as an array of byte in C

```
int fseek(FILE* stream, long offset, int wherefrom);
```

  - Sets the file position indicator for *stream*
  - New byte position is obtained by adding *offset* to the position specified by *wherefrom*
  - *wherefrom*
    - SEEK_CUR: The offset is computed from the current position in the file
    - SEEK_SET: The offset is computed from the beginning of the file
    - SEEEK_END: The offset is computed from the end of the file

File Seek Operation in C (2/2)



```
fseek (sp, 4 * sizeof(STRUCTURE_TYPE), SEEK_SET);
```

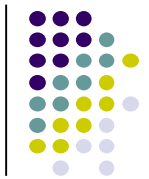
```
fseek (sp, - 4 * sizeof(STRUCTURE_TYPE), SEEK_END);
```

```
fseek (sp, 2 * sizeof(STRUCTURE_TYPE), SEEK_CUR);
```

File seek in C++ (1/2)

- fstream has two pointers
  - the *get* and the *put position*
  - **tellg() and tellp()**
    - Return the current get position (in the case of tellg) or the put position (in the case of tellp)
  - **seekg() and seekp()**
    - change the location of the get and put positions

File seek in C++ (2/2)



seekg(byte_offset, origin)

seekp(byte_offset, origin)

- origin – ios::beg, ios::cur, ios::end

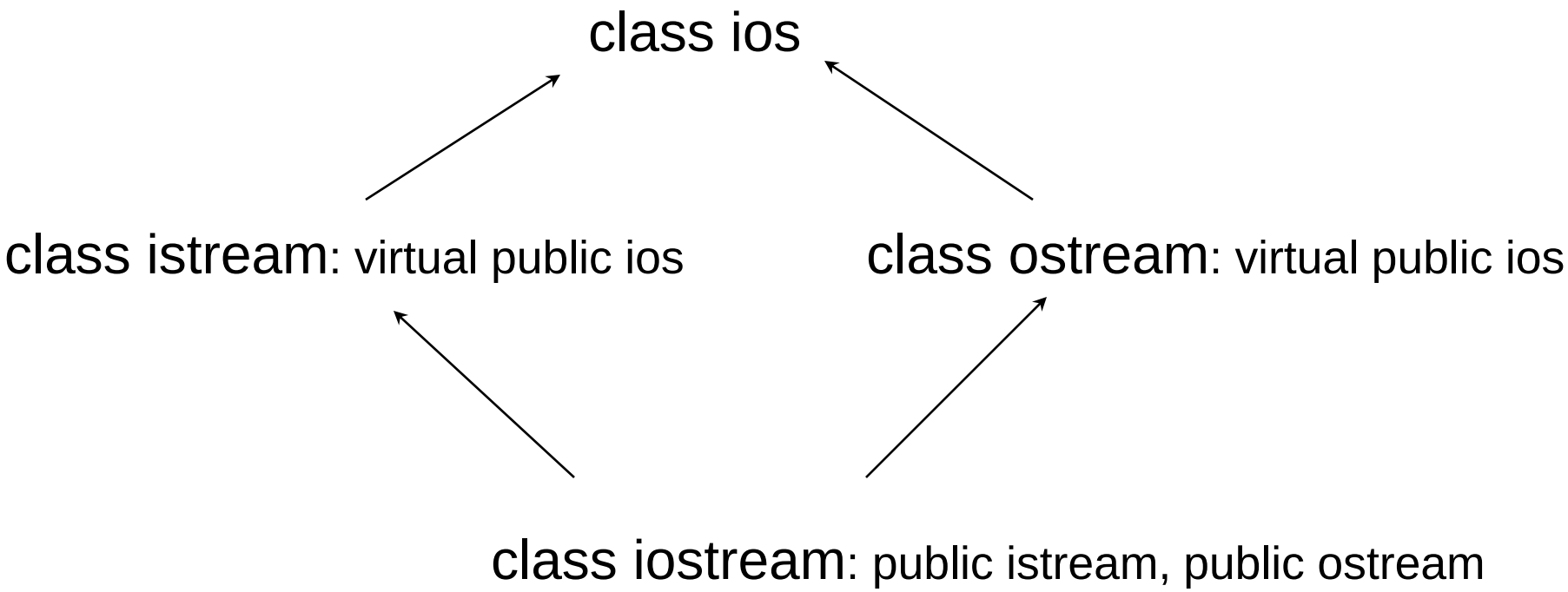
```
file.seekg(373, ios::beg);
file.seekp(373, ios::beg);
```

File related header files

- C streams
  - Stdio.h
- Many unix operations
  - fctnl.h, file.h

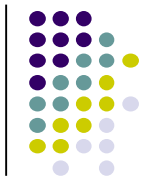
- C++ streams
  - iostream.h
  - fstream.h

iostream.h



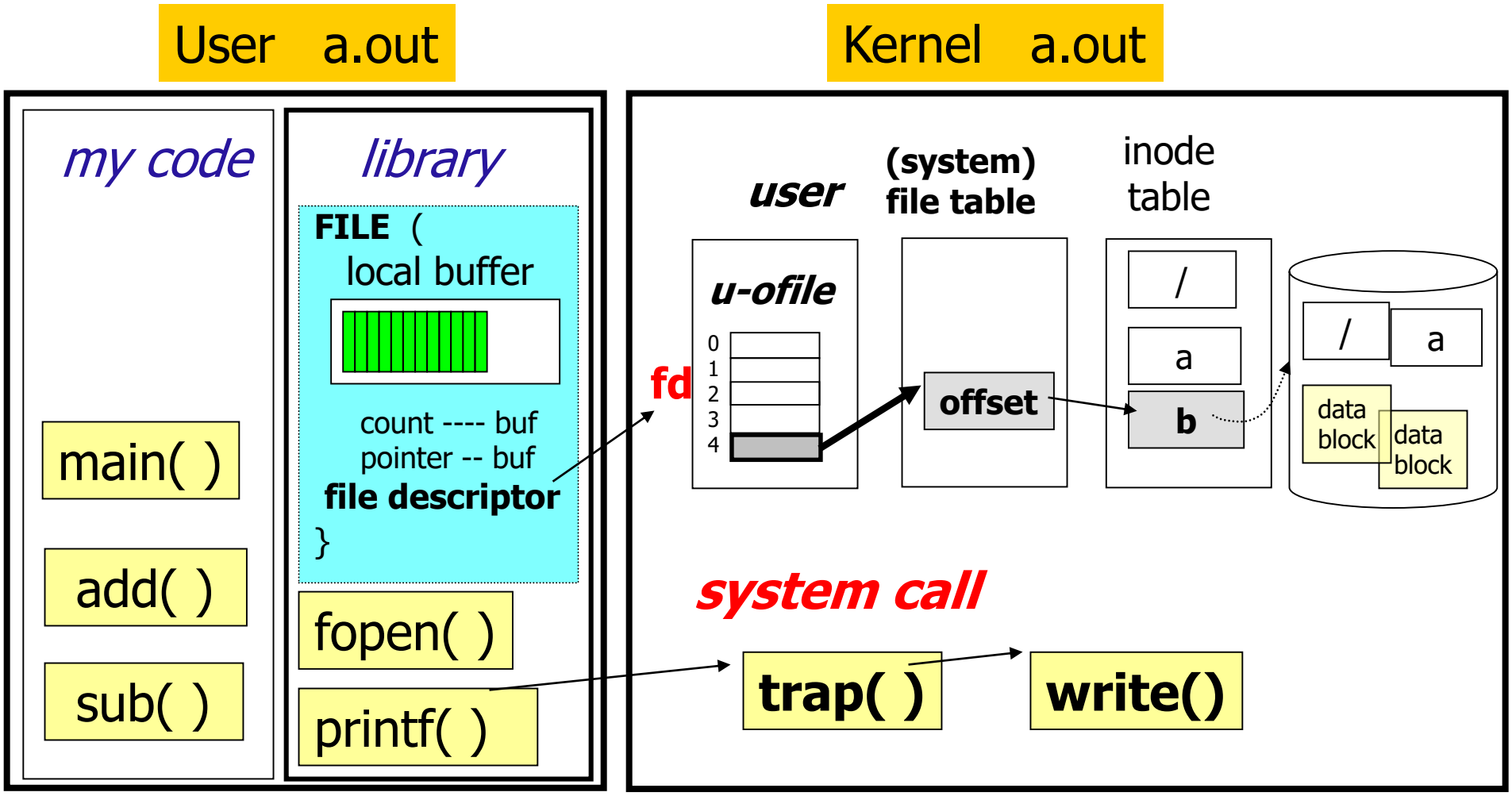
class hierarchy

Copy program in c++ : mycp.cpp



```
#include <iostream>
#include <fstream>

using namespace std;

void error(char *s, char *s2 = ""){
    cerr << s << ' ' << s2 << '\n';
    exit(1);
}

int main(int argc, char *argv[])
{
    if( argc != 3) error("wrong number of arguments");

    ifstream src(argv[1]);   //input file stream
    if (!src) error("cannot open input file", argv[1]);
    ofstream dest(argv[2]);    //output file stream
    if(!dest) error("cannot open output file", argv[2]);

    char ch;
    while( src.get(ch) ) dest.put(ch);

    if(!src.eof() || dest.bad())
        error("something strange happened");
    return 0;
}
```

Detour: FILE vs fd



When the local buffer (in FILE) becomes empty, Read() system call fills this buffer again

Detour: Functions for file handling

- So, you usually use library…

  printf() for formating (such as %s, %d)

  getchar() for performance

  ….

  But all library I/O functions end up asking system call

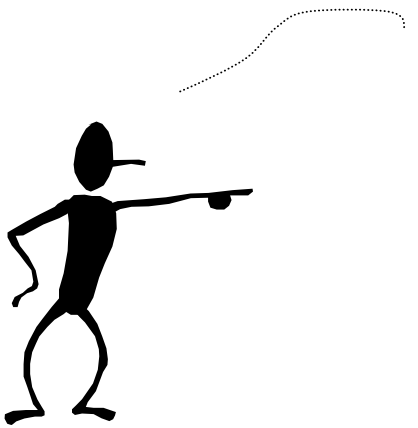
  (Library functions are " user" code & cannot do I/O directly)

  They are front-end and provide you with convenience,

  performance …

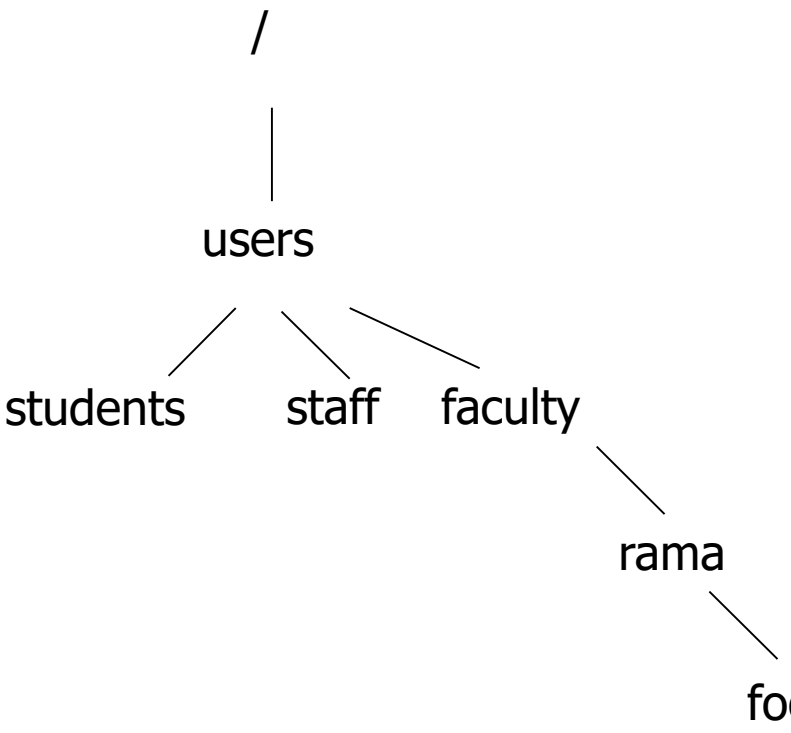
  Many library functions may exist

  But there's only one system call for read()



Detour: Unix Directory Structure

- UNIX file system : tree organization
- File : identified by its absolute value
  - ex) /usr/mydir/addr
  - ‘.’ current directory, ‘..’ the parent of ‘.’



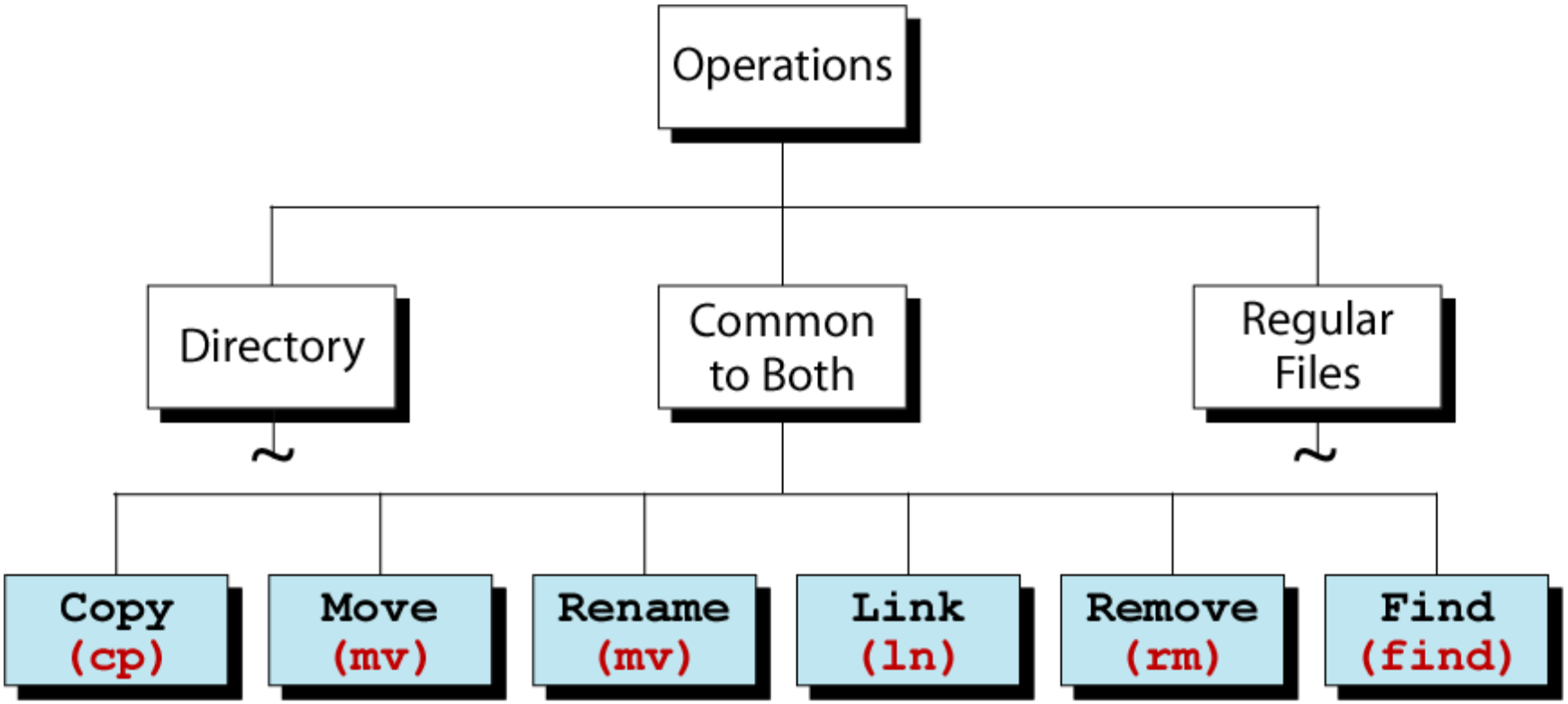
Detour: Physical and Logical files in UNIX

- UNIX views both physical devices and disk files as file.
  - ex) Keyboard(STDIN), Console(STDOUT), Error-file(STDERR)

- I/O redirection, pipes
  - Shortcuts for switching between standard I/O and regular file I/O
    - ex) list > myfile /* I/O redirection */
    - ex) program1 | program2 /* pipe */

Detour: Unix File system commands

- File and Directory commands



Q&A