Operating Systems

Prof. Yongkun Li
中国科大-计算机学院 教授
http://staff.ustc.edu.cn/~ykli

Chapter 9, part 1
File Systems – Programmer Perspective

Story so far…

User Space

Processes

Operating System Kernel

File system Implementation

FAT32, EXT2/3 KV, Distributed FS, Graph System…

File System Operations

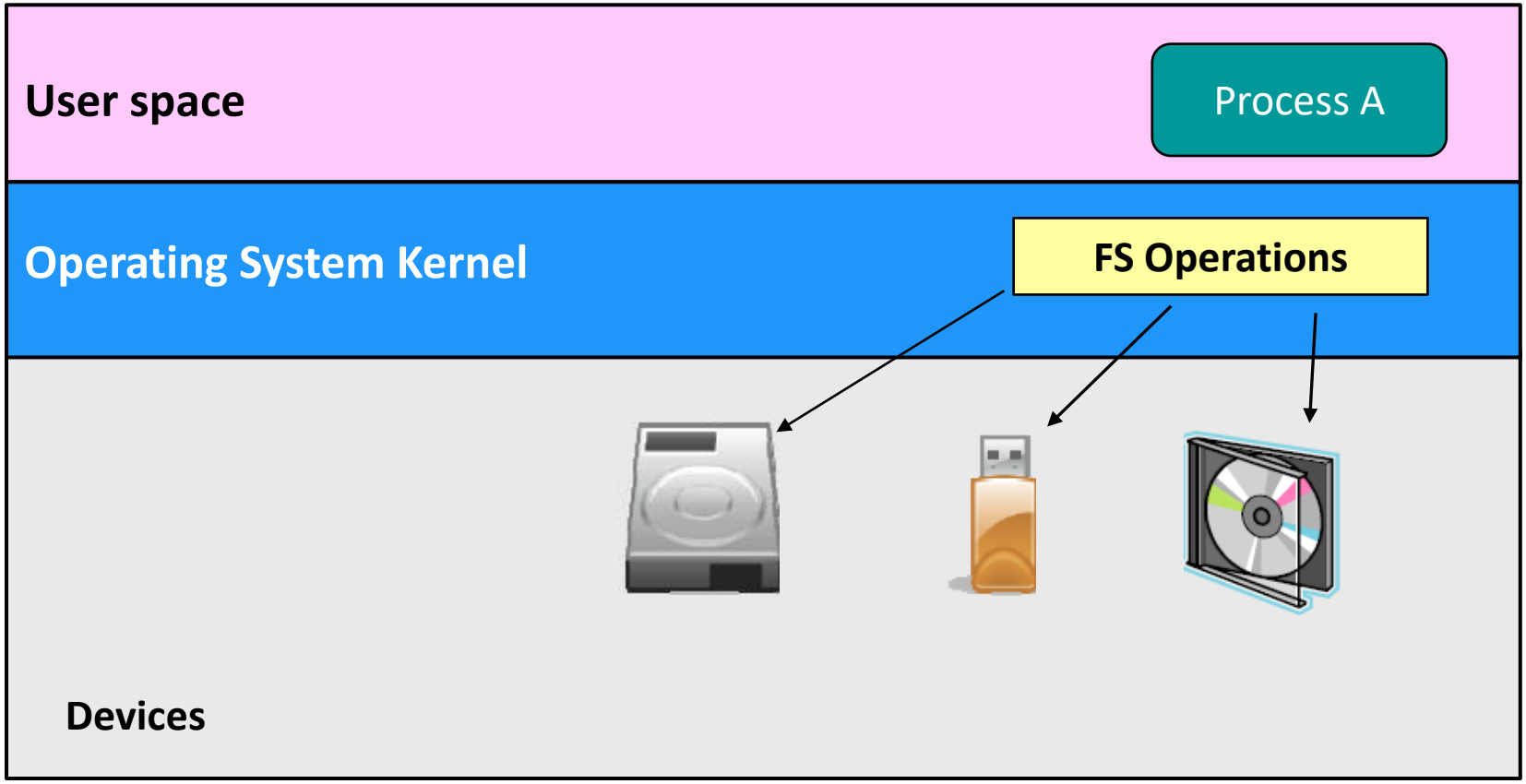
Devices

Outline

- File system introduction
- What are stored on a storage device?
  - File
  - Directory
  - Interfaces/Operations

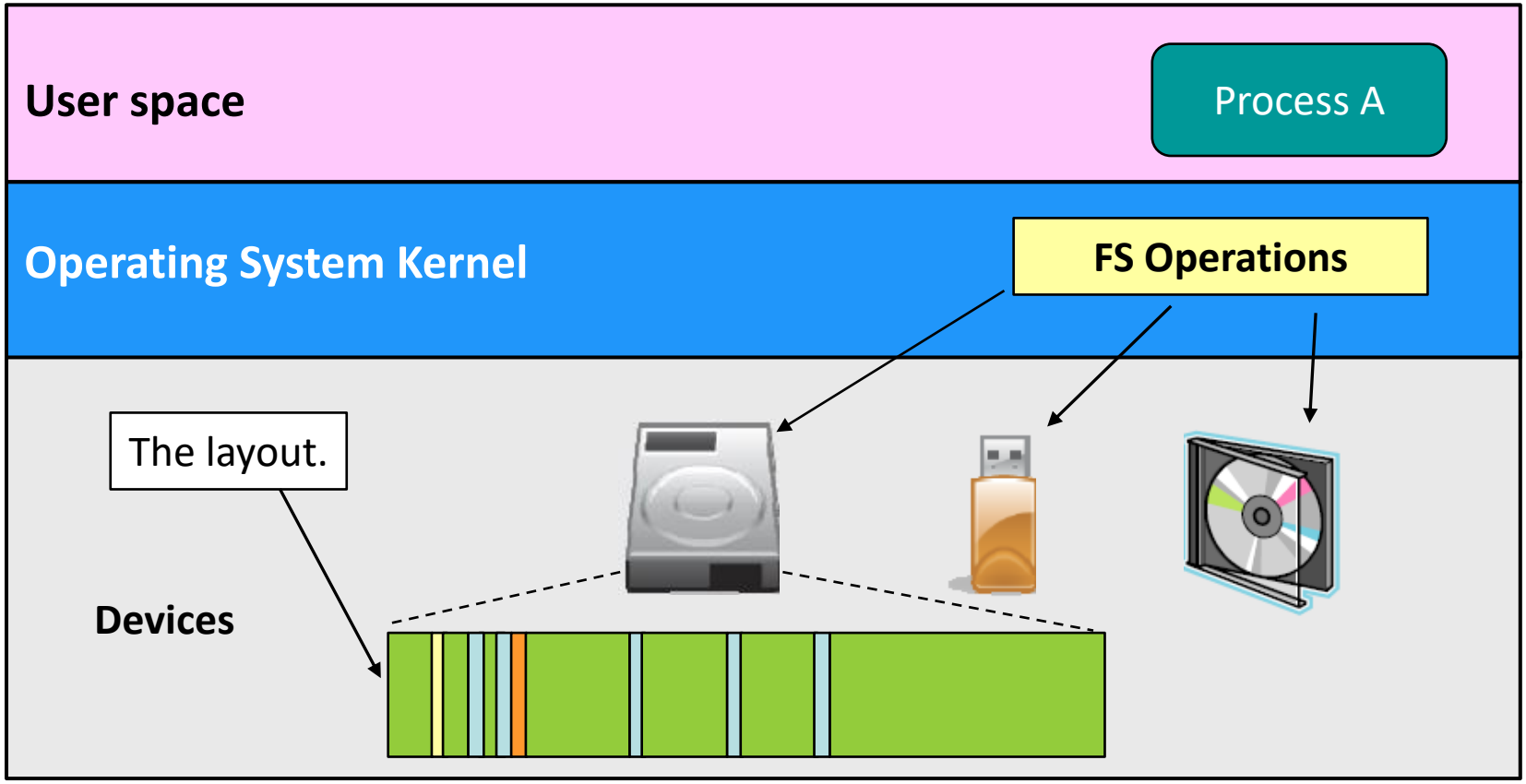
File system introduction

Introduction



- To understand what a file system (FS) is, we follow two different, but related directions:
  - **Layout** & **Operations**.

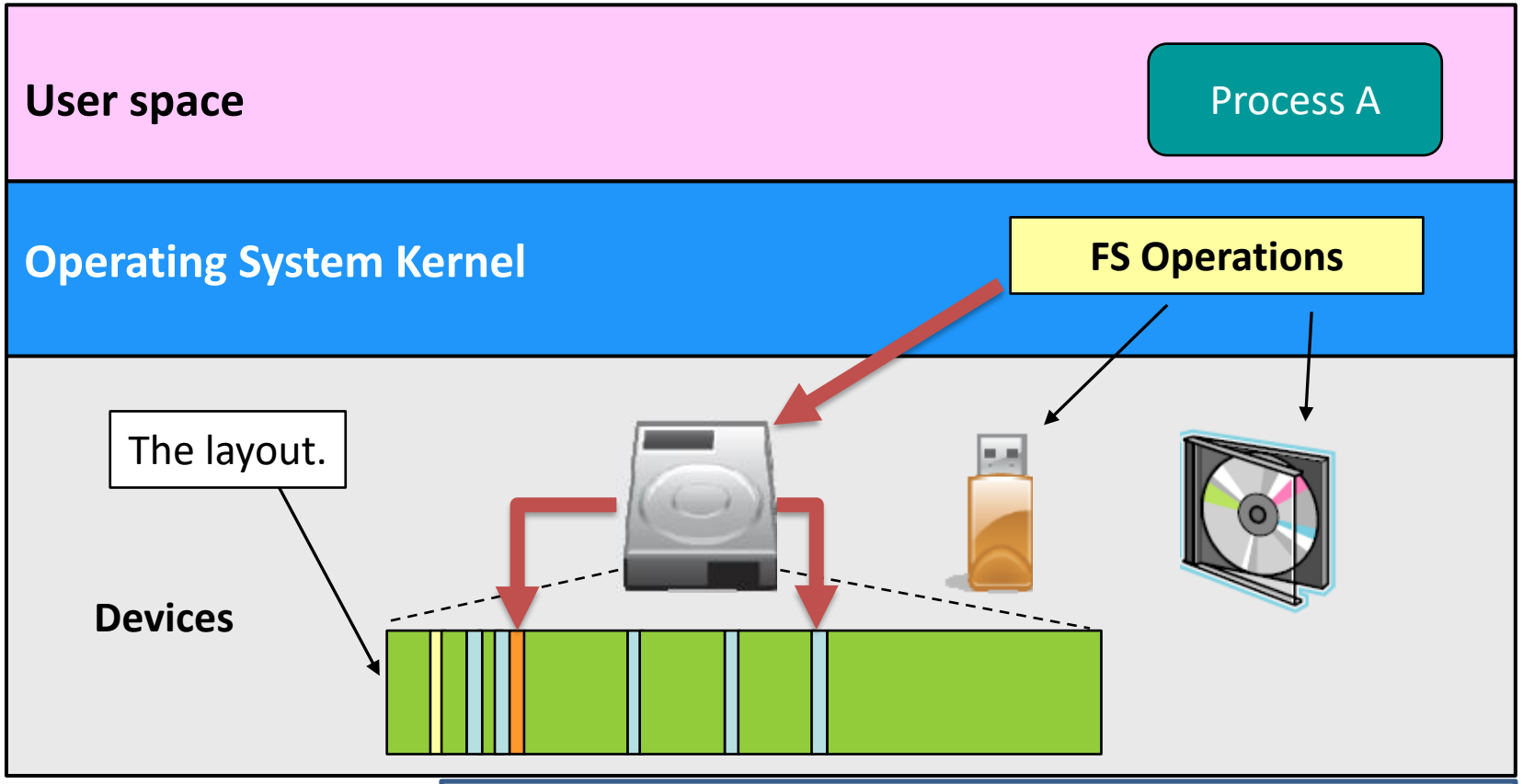
Introduction



Every FS has an unique layout on the storage device. The layout defines:

- **What** are the things stored in the device.
- **Where** the stored things are.

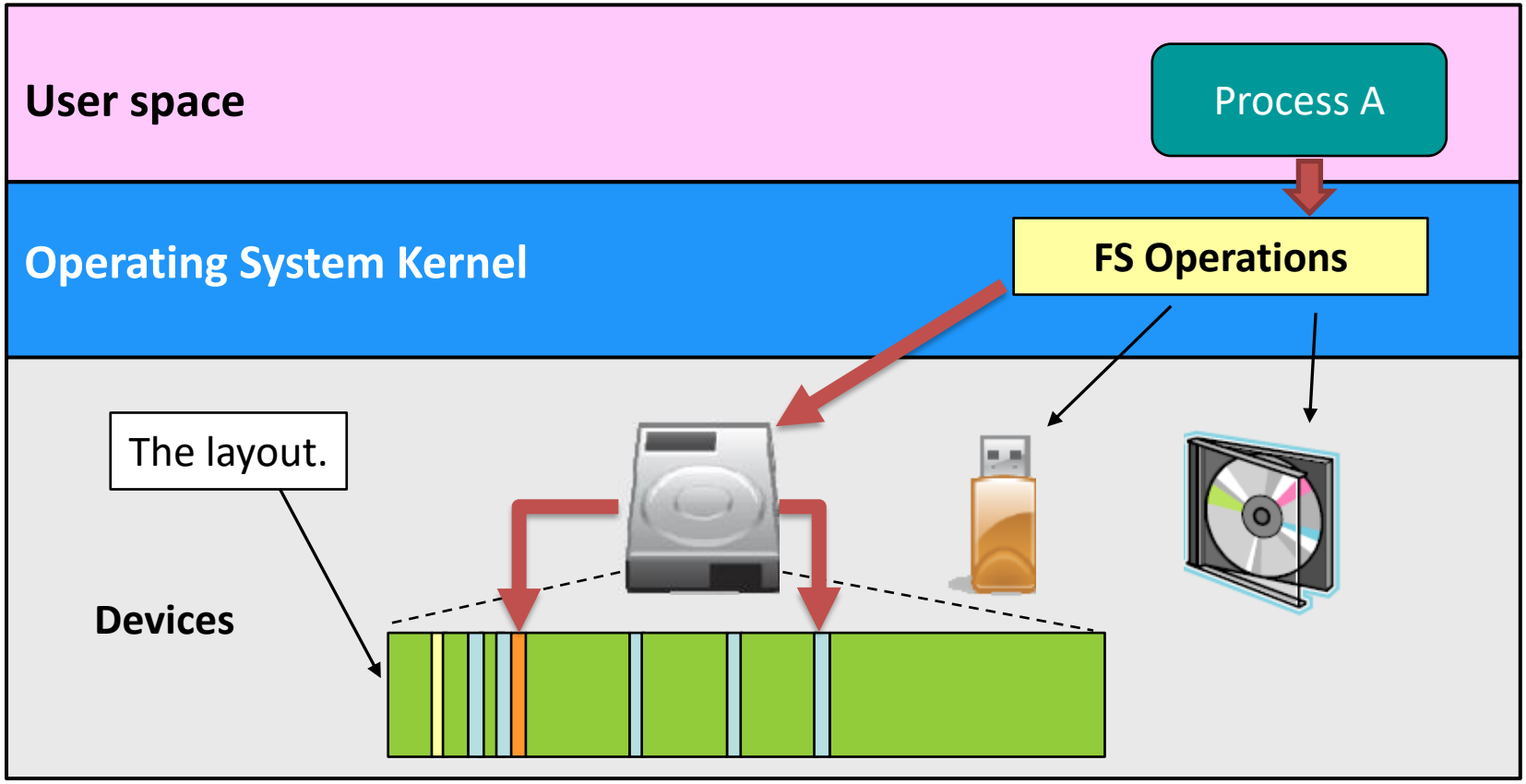
Introduction



The set of FS operations defines how the OS should work with the FS layout.

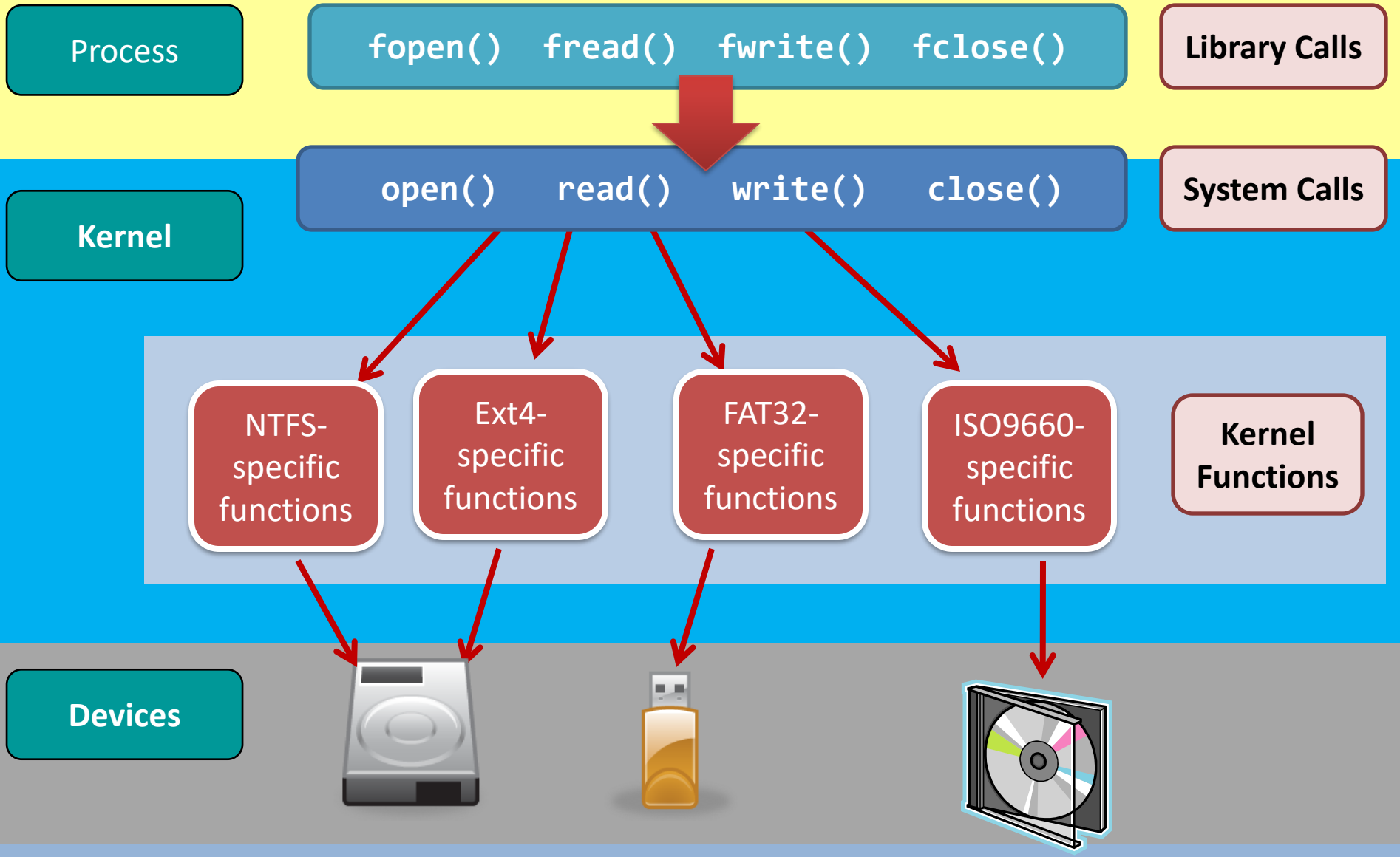
In other words, **OS knows the FS layout** and works with that layout.

Introduction



The process uses **system calls**, which then invoke the FS operations, to access the storage device.

Introduction



Summary

- Ask yourself:
  - OS = FS?
  - **Correct answer: OS ≠ FS**
  - **An OS supports a FS**

- An OS can support more than one FS.
- A FS can be read by more than one OS.

Summary

- Ask yourself:
  - Storage Device = FS?
  - **Correct answer: Storage Device ≠ FS**.

- A FS must be stored on a device.
  - But, a device may or may not contain any FS.
  - Some storage devices can host more than one FS.

- A storage device is only a dummy container.
  - It doesn't know and doesn't need to know what FS-es are stored inside it.
  - The OS instructs the storage device how the data should be stored.

Outline of topics

- There are **two basic things** that are stored inside a storage device, and are common to all existing file systems.

What are they?

– They are **Files** and **Directories**.

– We will learn what they are and some basic operations of them.

Outline of topics

- There are **two basic things** that are stored inside a storage device, and are common to all existing file systems.

How does a FS store data into the disk?

– That is, the **layout** of file systems.

– The layout affects many things:
  - The speed in operating on the file systems;
  - The reliability in using the file systems;
  - The allocation and de-allocation of disk spaces.

Outline of topics

- Other topics

  - We will look into the details of FAT32 and Ext2/3 file systems.
  - Case studies: key-value systems, distributed file systems, graph storage systems

Part1: FS – Programmer Perspective

- File
- Directory
- Operations

File

- Why do we need files?
  - Storing information in memory is good because memory is fast.
  - However, memory vanishes after process termination.

  - File provides a long-term information storage.
    - It is persistent and survives after process termination.
  - File is also a shared object for processes to access concurrently.

File

- What is a file?
  - A uniform logical view of stored information provided by OS.
  - **OS perspective**: A file is a logical storage unit (a sequence of logical records), it is an abstract data type
  - **User perspective**: the smallest allotment of logical secondary storage

  - File type (executable, object, source code, text, multimedia, archive…)
  - File attributes
  - File operations

File – what are going to be stored?

- E.g., a text file.

What can we find out in this example?

Content?	Content of the file
Filename?	Content of its parent directory
File size?	Attribute of the file

When a file is named, it becomes independent of the process, the user, and even the system

File Attributes

- Typical file attributes

Name	Human-readable form
Identifier	Unique tag (a number which identifies the file within the FS)
Type	Text file, source file, executable file…
Location	Pointer to a device and to the location of the file on the device
Size	Number of bytes, words, or blocks
Time, date	Creation, last modification, last use…
Protection	Access control information (read/write/execute)

You can try the command “ls -l”

File Attributes

- Typical file attributes

Name	Human-readable form
Identifier	Unique tag (a number which identifies the file within the FS)
Type	Text file, source file, executable file…
Location	Pointer to a device and to the location of the file on the device
Size	Number of bytes, words, or blocks
Time, date	Creation, last modification, last use…
Protection	Access control information (read/write/execute)

Some new systems also support extended file attributes (e.g., checksum)

File Attributes

- File attributes are FS dependent.
  - Not OS dependent.

The design of FAT32 does not include any security ingredients.

Common Attributes	FAT32	NTFS	Ext2/3/4
Name	✓	✓	✓
Size	✓	✓	✓
Permission		✓	✓
Owner		✓	✓
Access, creation, modification time	✓	✓	✓

File Permissions

- E.g., in Unix system

-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5 pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 jwg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2012	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2012	program
drwx--x--x	4 tag	faculty	512	Jul 31 10:31	lib/
drwx------	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

First field: File/director

2nd /3rd /4th fields (3 bits each): controls read/write/execute for the file owner/file's group/others (e.g., 111:7,110:6)

What is the meaning of the permission 775/664?

Writing attributes?

- Can you change those attributes directly?

Common Attributes	Way to change them?	
	Command?	Syscall?
Name	`mv`	`rename()`
Size	`Too many tools to update files' contents`	`write(), truncate(), etc.`
Permission	`chmod`	`chmod()`
Owner	`chown`	`chown()`
Access, creation, modification time	`touch`	`utime()`

Part1: FS – Programmer Perspective

- File
- Directory
- Operations

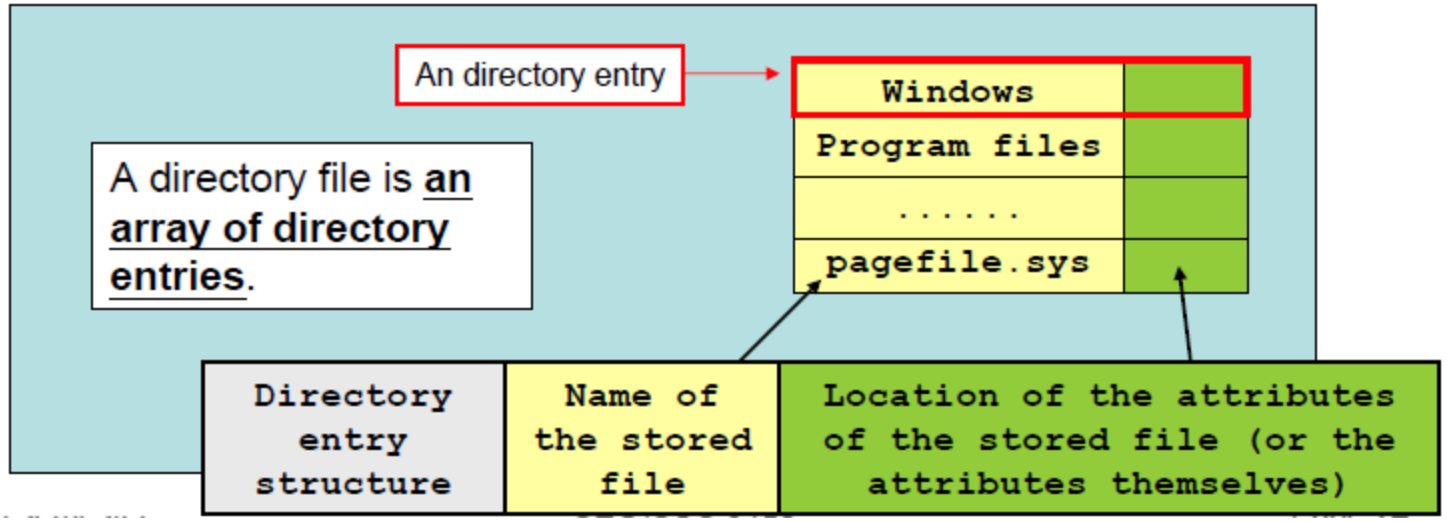
Directory

- A directory is **a file**.
  - Then, does it imply that it has **file attributes** and **file content**?

Answer: Sure

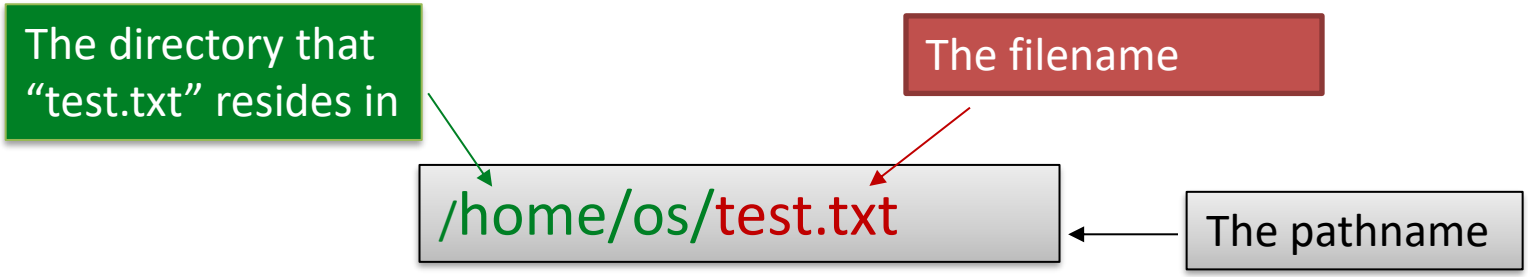
Answer: FS dependent

- How does a directory file look like?



Pathname vs Filename

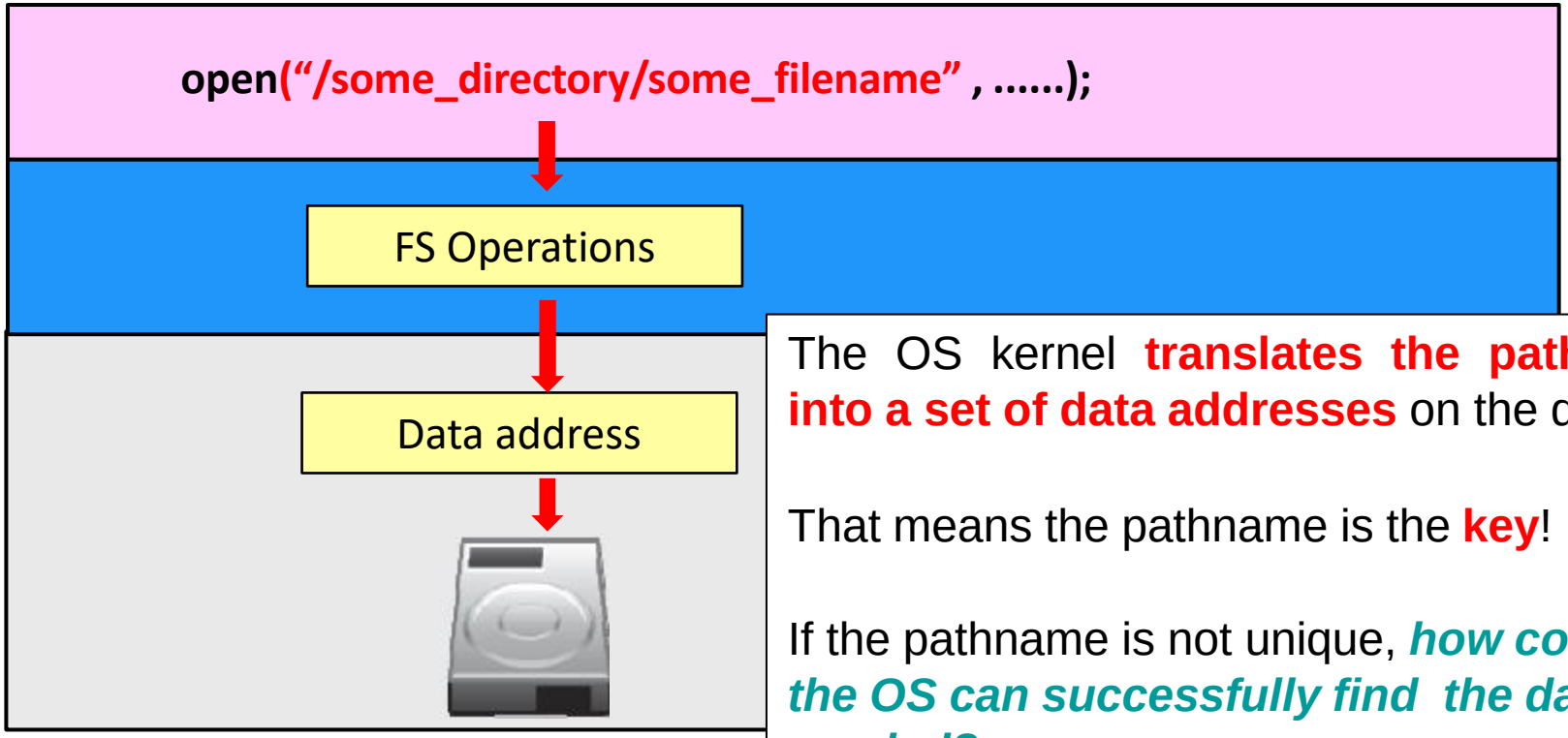
- A file can be referred to by its name, then how to achieve this?



The pathname is **unique** within the entire file system.
The filename is **not unique** within the entire file system.
The filename is only **unique within the directory** that it resides.

Pathname vs Filename

- Why do we need to consider **uniqueness**?



The OS kernel **translates the pathname into a set of data addresses** on the device.

That means the pathname is the **key**!

If the pathname is not unique, ***how come the OS can successfully find the data needed?***

Directory Traversal Process

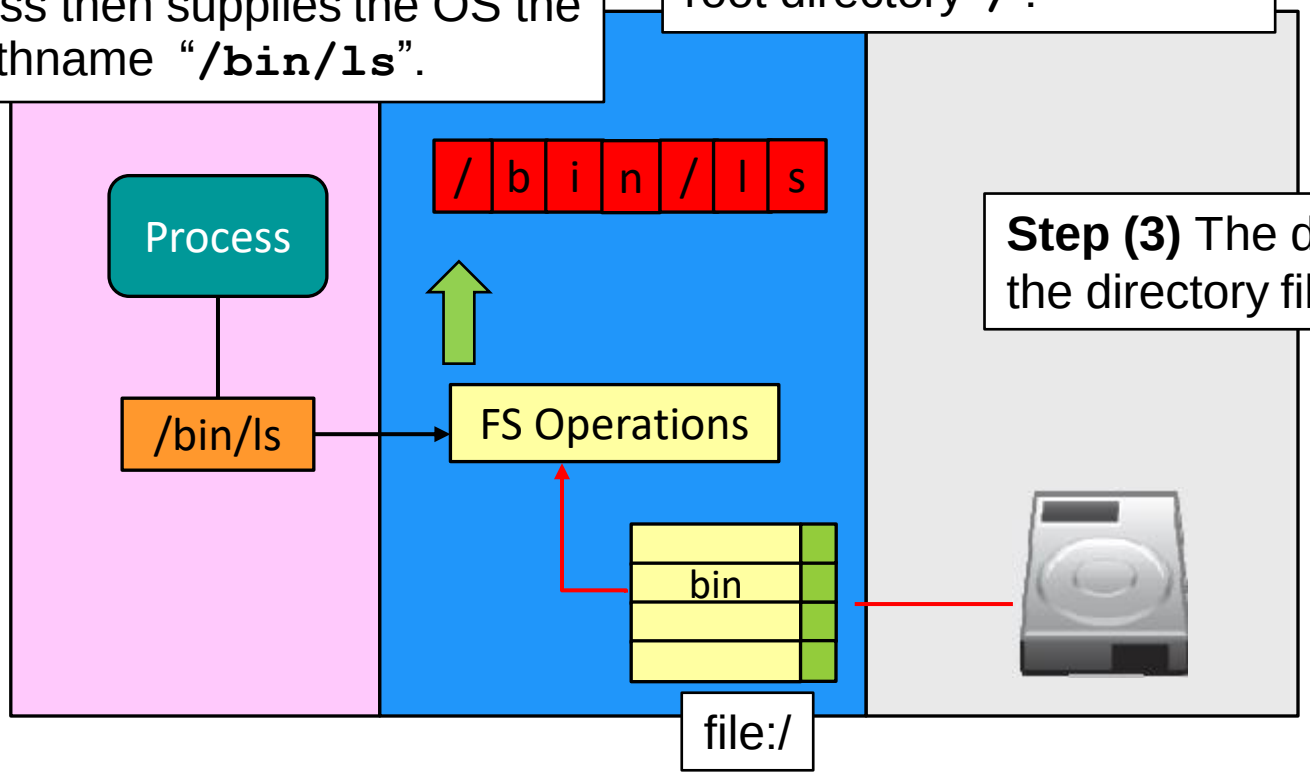
- How to locate a file using pathname?

Step (1) Suppose that the process wants to open the file "**`/bin/ls`**".

The process then supplies the OS the unique pathname "**`/bin/ls`**".

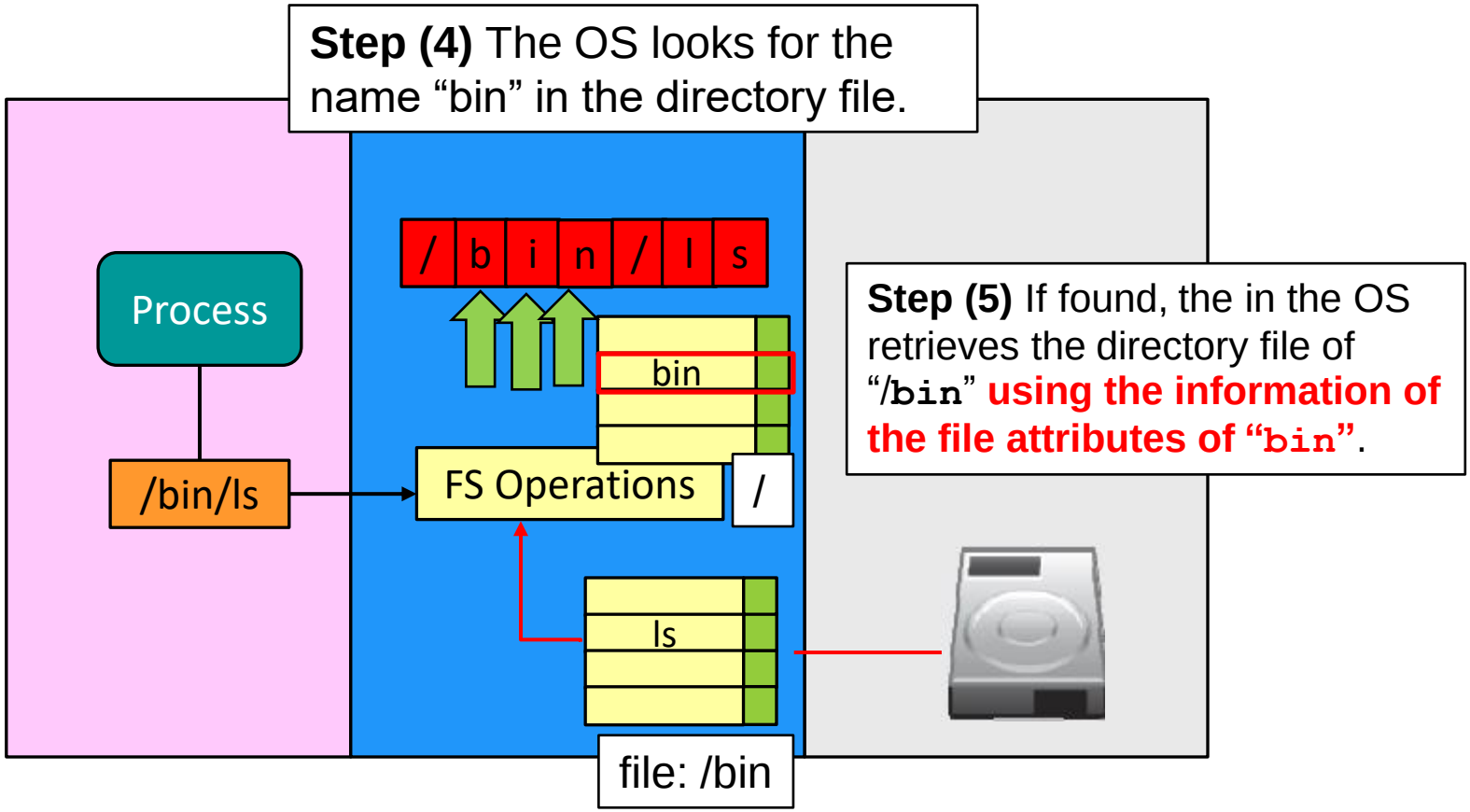
Step (2) The OS retrieves the directory file of the root directory '**`/`**'.

Step (3) The disk returns the directory file.



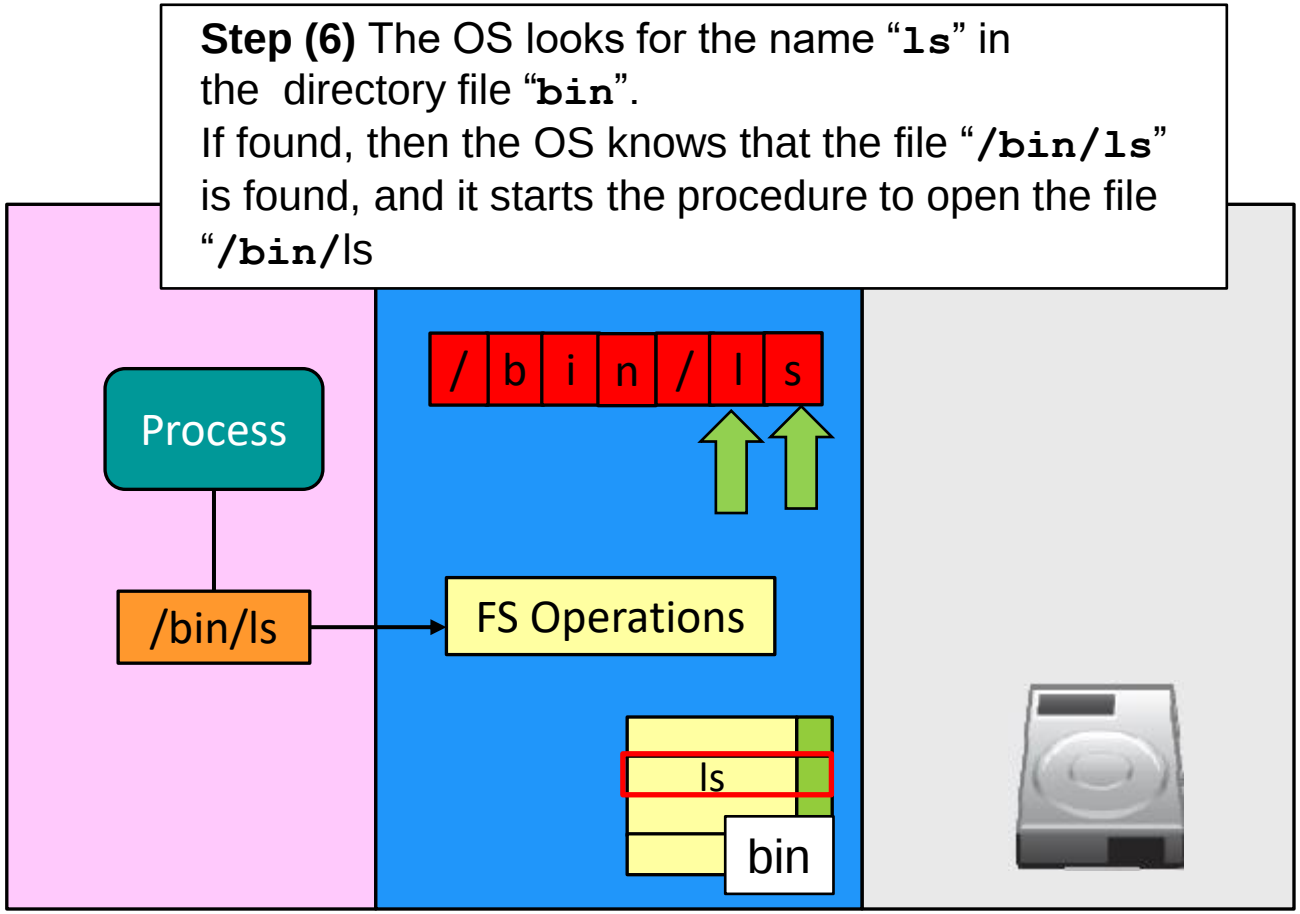
Directory Traversal Process

- How to locate a file using pathname?



Directory Traversal Process

- How to locate a file using pathname?



Short Summary

- A directory file records all the files including directories that are belonging to it.
  - So, do you understand "**`/bin/ls`**" now?
  - Locate the directory file of the target directory and to print contents out.

- Locating a file requires the **directory traversal process**

File Creation and Directory

- According to your experience, **what is the file creation?**
  - E.g., creating a file named "**`test.txt`**"?
    - "**`touch test.txt`**"?
    - "**`vim test.txt`**", then type "**`:wq`**"?
    - "**`cp [some filename] test.txt`**"?

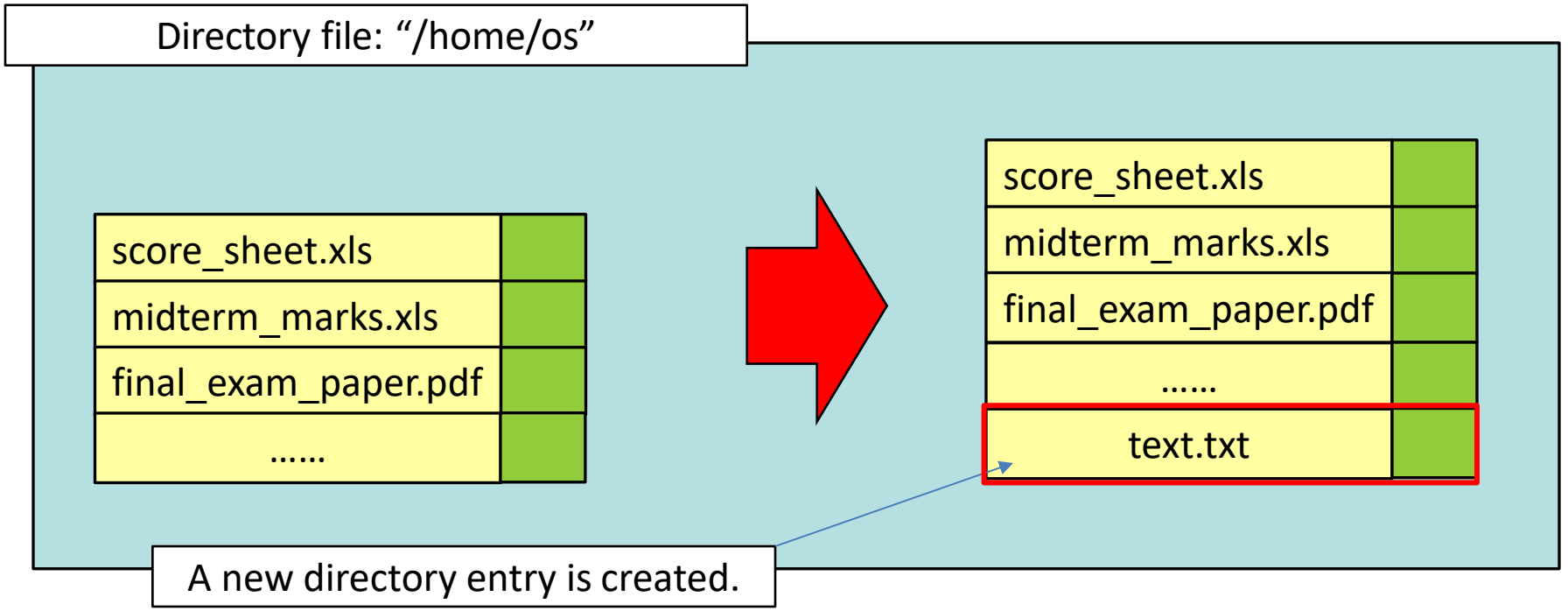
- The truth is:

  File creation == Update of the directory file

File Creation and Directory

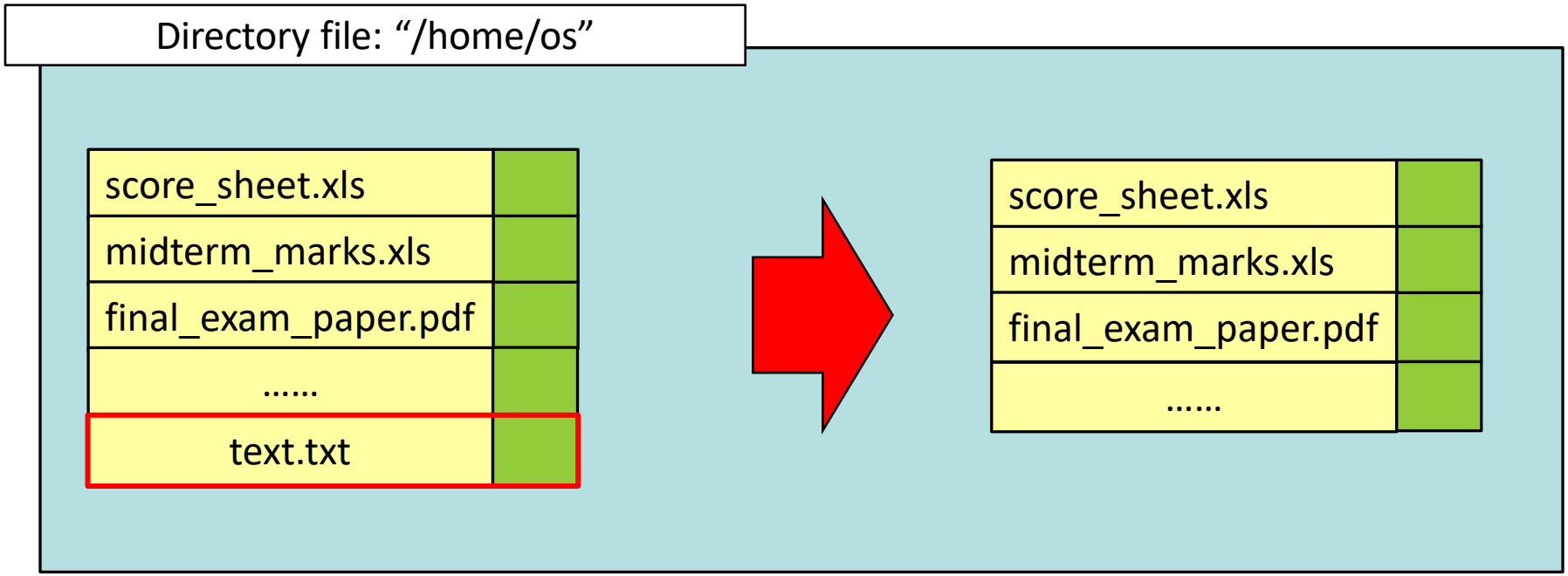
- If I type "**touch text.txt**" and "**text.txt**" does not exist, what will happen to the Directory file?

Note: "`touch text.txt`" will only create the directory entry, and there is no allocation for the file content.



File Deletion and Directory

- Removing a file is the reverse of the creation process.
  - Note that we are not ready to talk about de-allocation of the file content yet.



Updating directory file

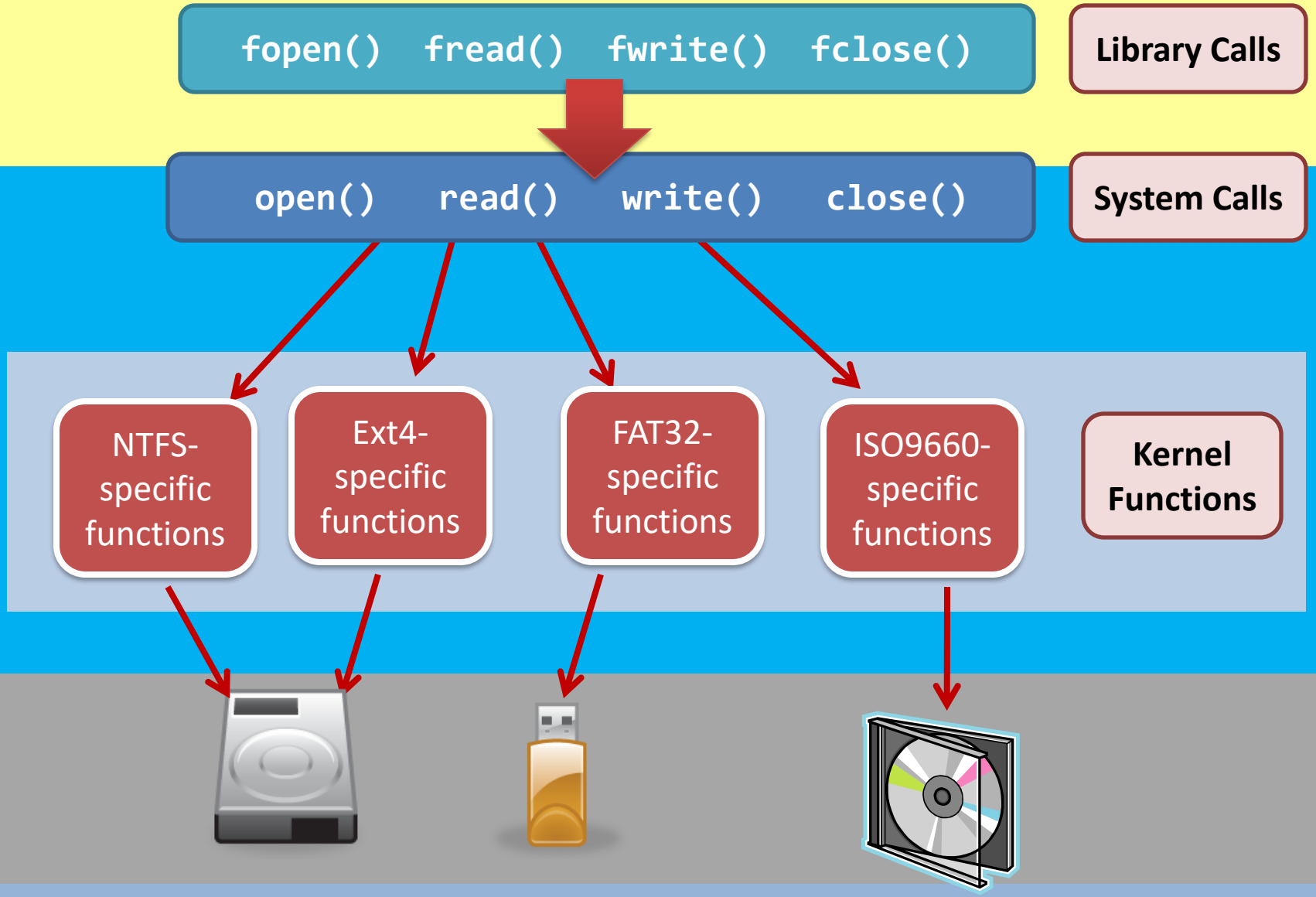
- When/how to update a directory file?

Creating a directory file	syscall - `mkdir()`; Example program - `mkdir`.
Add an entry to the directory file	syscall - `open()`, `creat()`; Example program - `cp`, `mv`, etc.
Remove an entry to the directory file	syscall - `unlink()`; Example program - `rm`.
Remove a directory file	syscall – `rmdir()`; Example program - `rmdir`.

Part1: FS – Programmer Perspective

- File
- Directory
- **Operations**

Overview


fopen() fread() fwrite() fclose()
Library Calls
open() read() write() close()
System Calls
NTFS-specific functions
Ext4-specific functions
FAT32-specific functions
ISO9660-specific functions
Kernel Functions

File operations

- The operating system should provide…

Create	Allocate space, add an entry in the directory
Write	Filename, file content (write pointer)
Read	Filename, mem location (read pointer)
Reposition	File seek (not involve actual I/O), **required for random accesses**
Delete	Release space, and erase directory entry
Truncate	Keeps attributes only

File operations

- Many operations involve searching the directory for locating the file (read/write/reposition…)
  - Can we avoid this content searching???

Open-file table

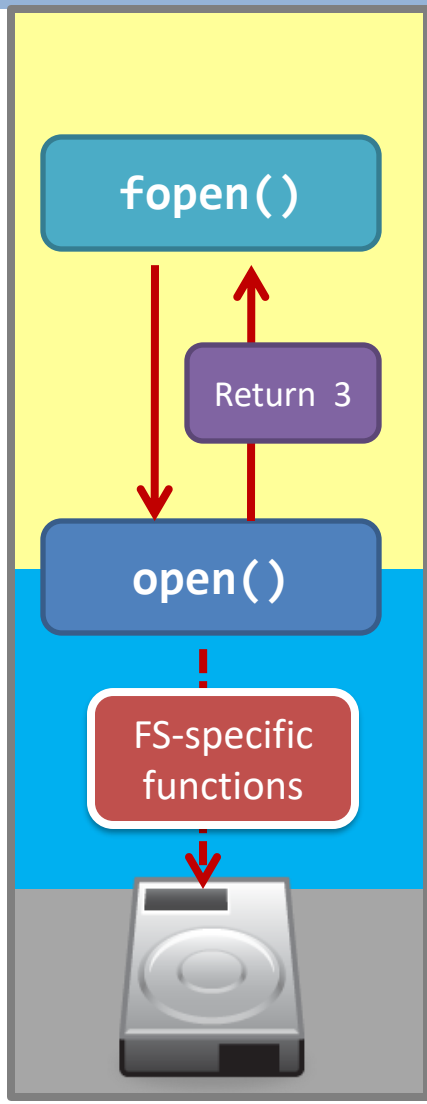
An open() system call is provided, and it is called before a file is first used

OS keeps a table containing information about all open files (per-process and system-wide table)

The file will be closed when it is no longer being actively used, using close() system call

File Open – Example

- What is **`fopen()`**?
  - First thing first, **`fopen()`** calls **`open()`**.
  - `FILE *fopen(const char *filename, const char *mode)`
- What is the type "**`FILE`**"?
  - "**`FILE`**": a structure defined in "**`stdio.h`**".
  - **`fopen()`** *creates memory* for the "**`FILE`**" structure.
    - Fact: occupying space in the area of dynamically allocated memory, i.e., `malloc()`



What is inside the "FILE" structure?

- There is a lot of helpful data in **FILE**:
  - Two important things: the **file descriptor** and **a buffer**!

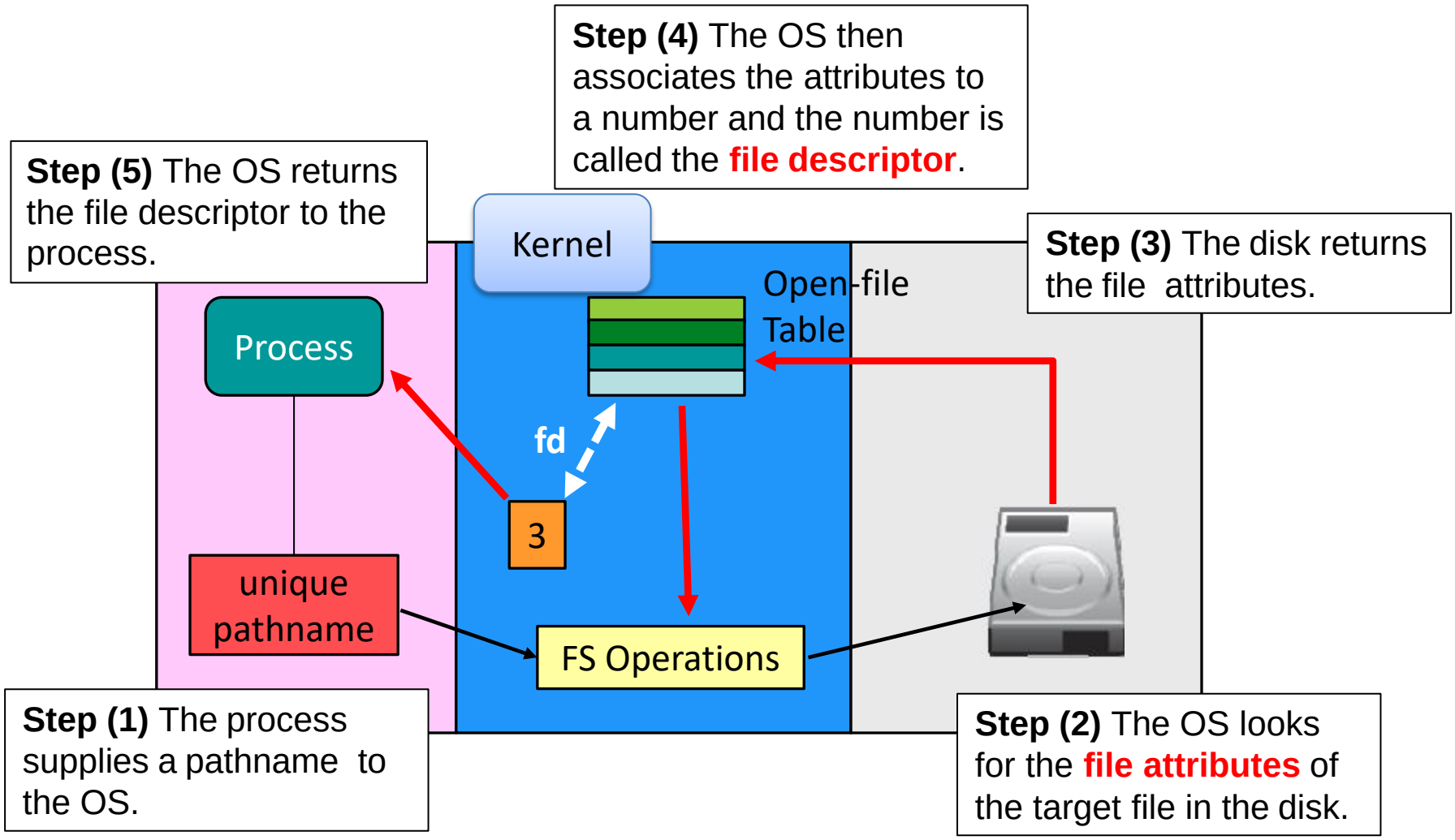
```
int main(void) {
    printf("fd of stdin  = %d\n", fileno(stdin)  );
    printf("fd of stdout = %d\n", fileno(stdout) );
    printf("fd of stderr = %d\n", fileno(stderr) );
}
```

fileno() returns the file descriptor of the FILE structure.

The type of **stdin**, **stdout**, and **stderr** is "**FILE ***"

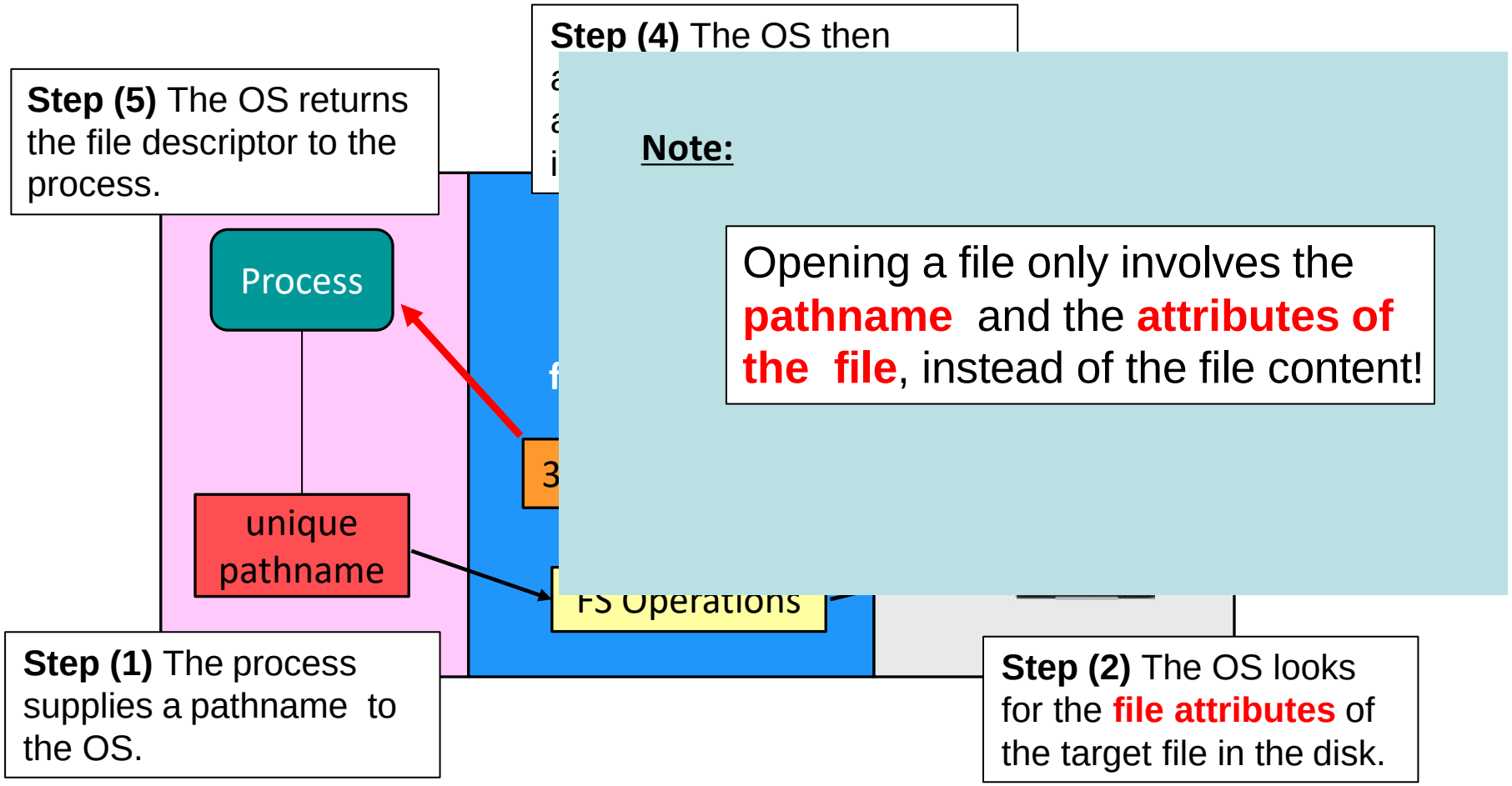
```
$ ./fileno
fd of stdin  = 0
fd of stdout = 1
fd of stderr = 2
$ _
```

The Truth of Opening a File



Note: these steps are OS-independent as well as FS-independent.

The Truth of Opening a File

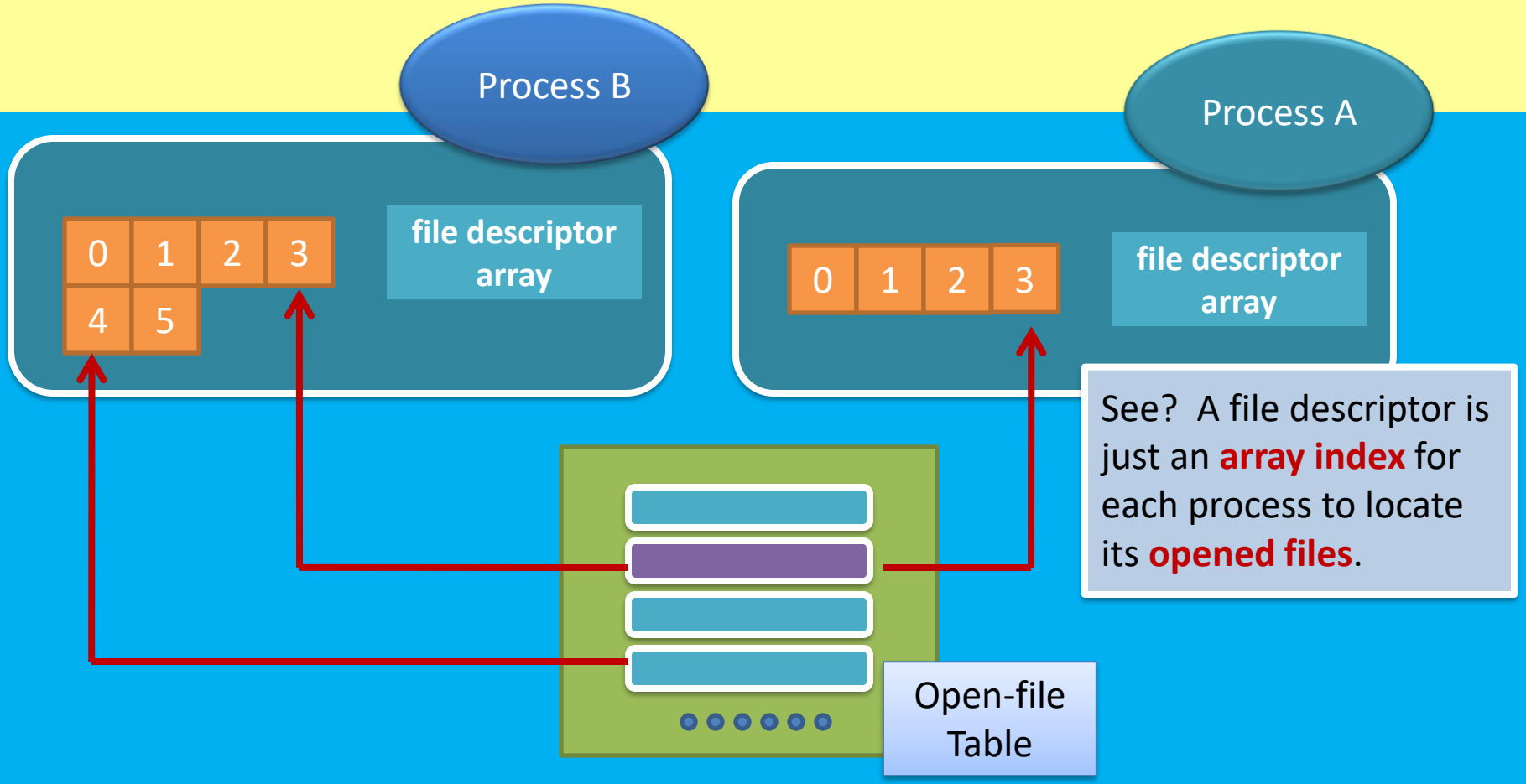


Note:

Opening a file only involves the **pathname** and the **attributes of the file**, instead of the file content!

Note: these steps are OS-independent as well as FS-independent.

What is a file descriptor?



Although a file is opened by two different processes, the kernel uses **one structure to maintain it**!

How about read and write (**`read()`** and **`write()`** system calls)?

read() & write()

- You know, I/O-related calls will invoke system calls.

```
int   read ( int fd,  void *buffer,  int bytes_to_read )
```

From file to buffer.

```
int  write ( int fd,  void *buffer,  int bytes_to_write )
```

From buffer to file.

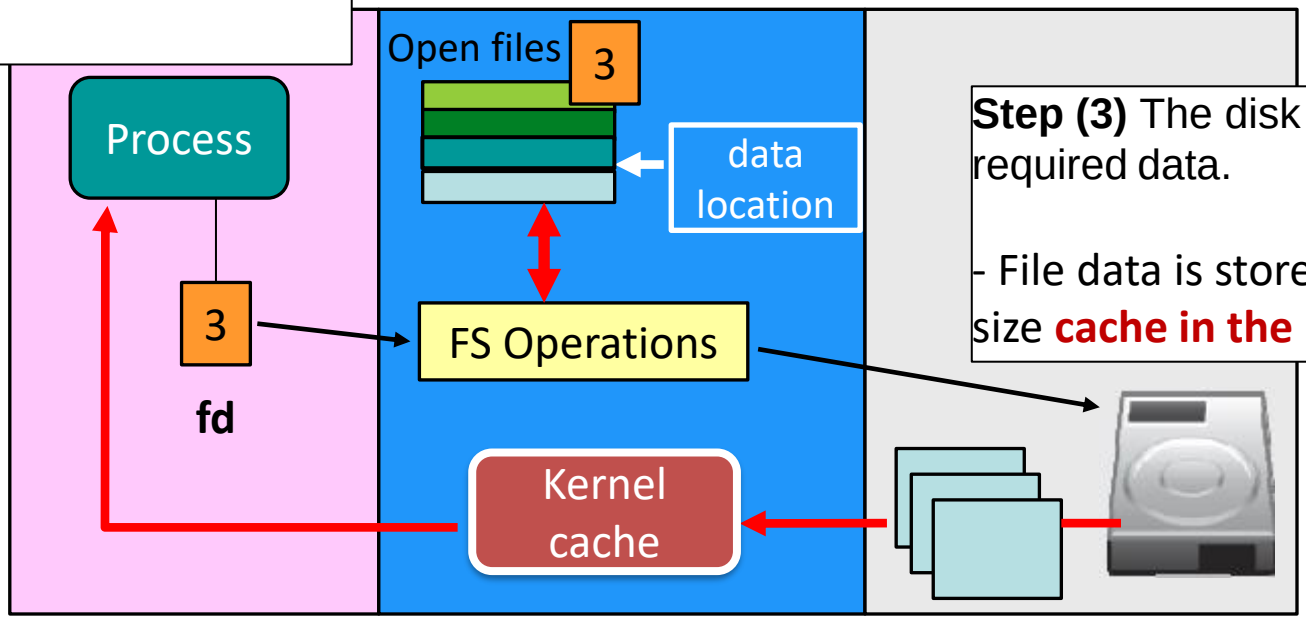
Note: I modified the function prototypes.

Library calls that eventually invoke the `read()` system call	Library calls that eventually invoke the `write()` system call
`scanf(), fscanf()`	`printf(), fprintf()`
`getchar(), fgetc()`	`putchar(), fputc()`
`gets(), fgets()`	`puts(), fputs()`
`fread()`	`fwrite()`

How to read from open files

Step (1) The process supplies a file descriptor to the OS.

Step (2) The OS reads the file attributes and uses the stored attributes to **locate the required data**.

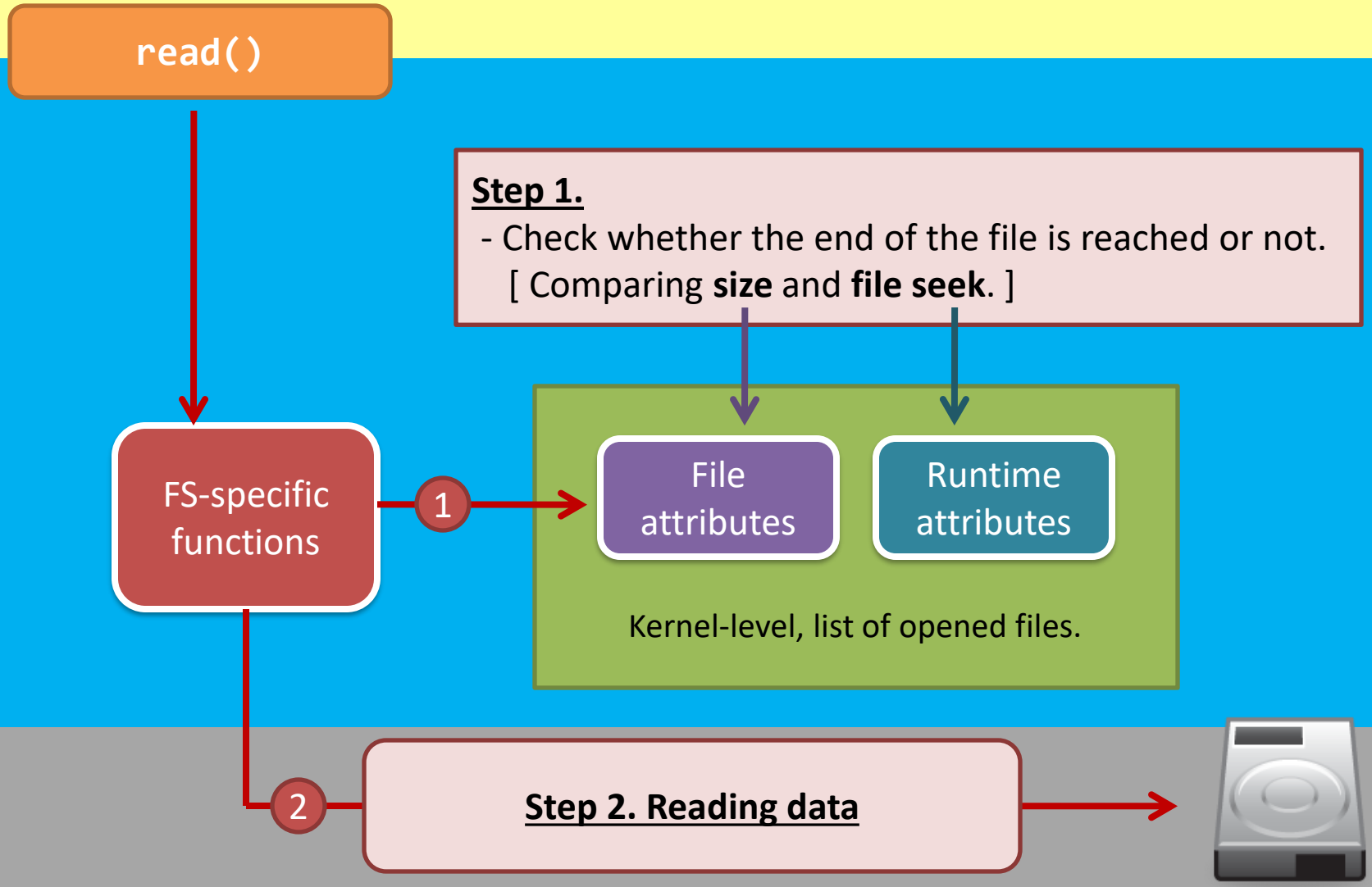


Step (3) The disk returns the required data.

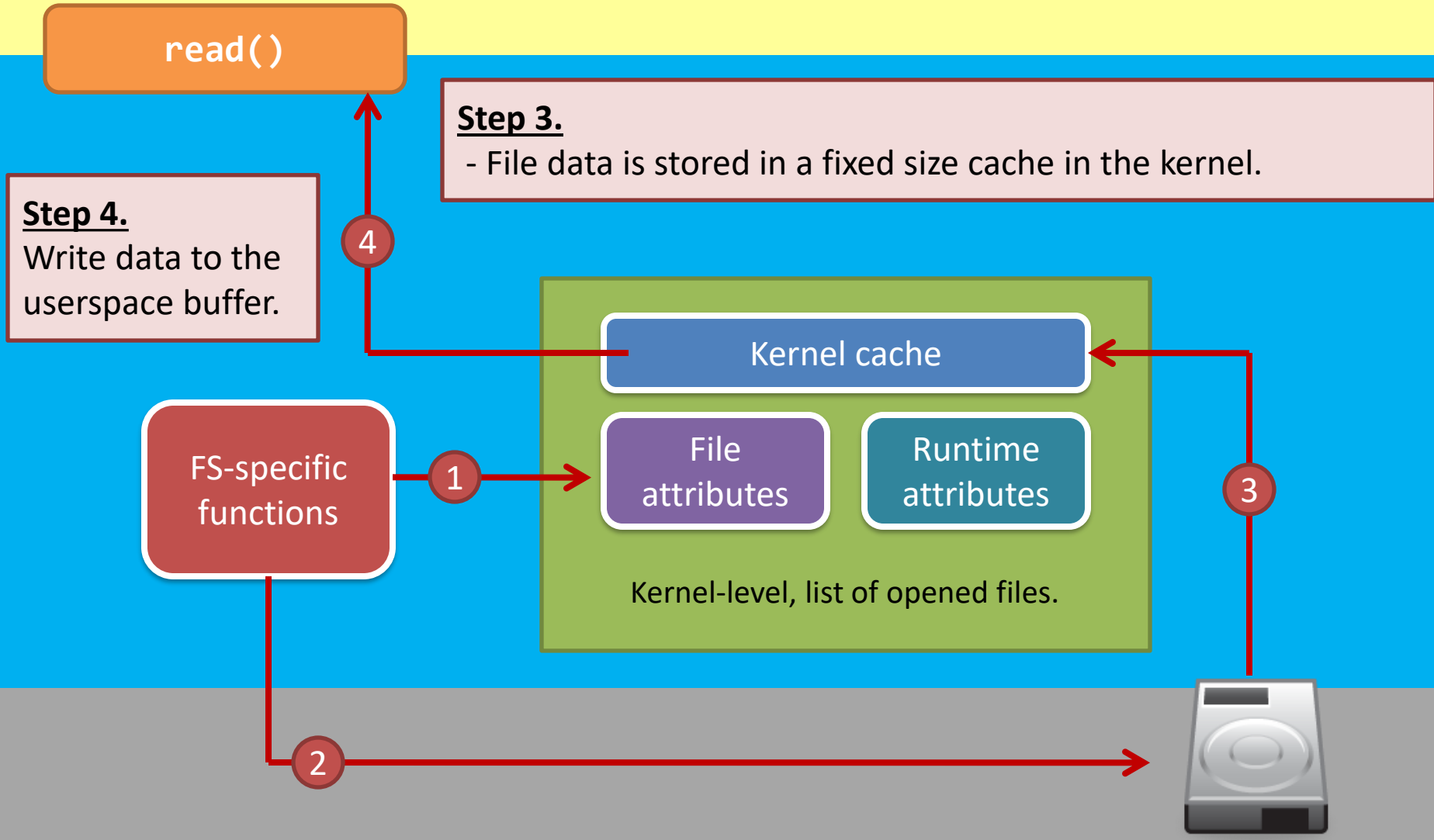
- File data is stored in a fixed size **cache in the kernel**.

Step (4) The OS fills the **buffer provided by the process** with the data. **Write data to the userspace buffer.**

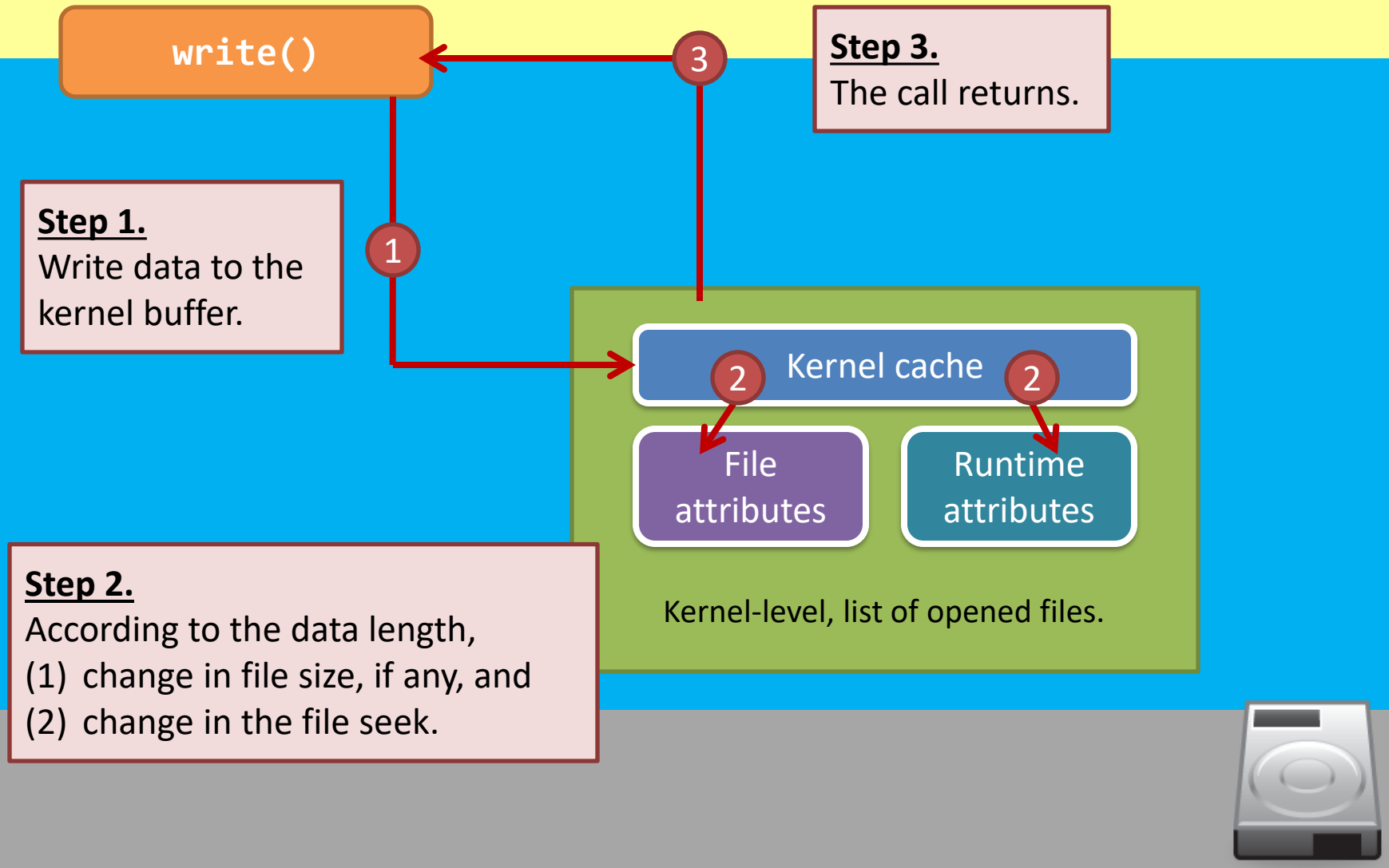
`read()` system call



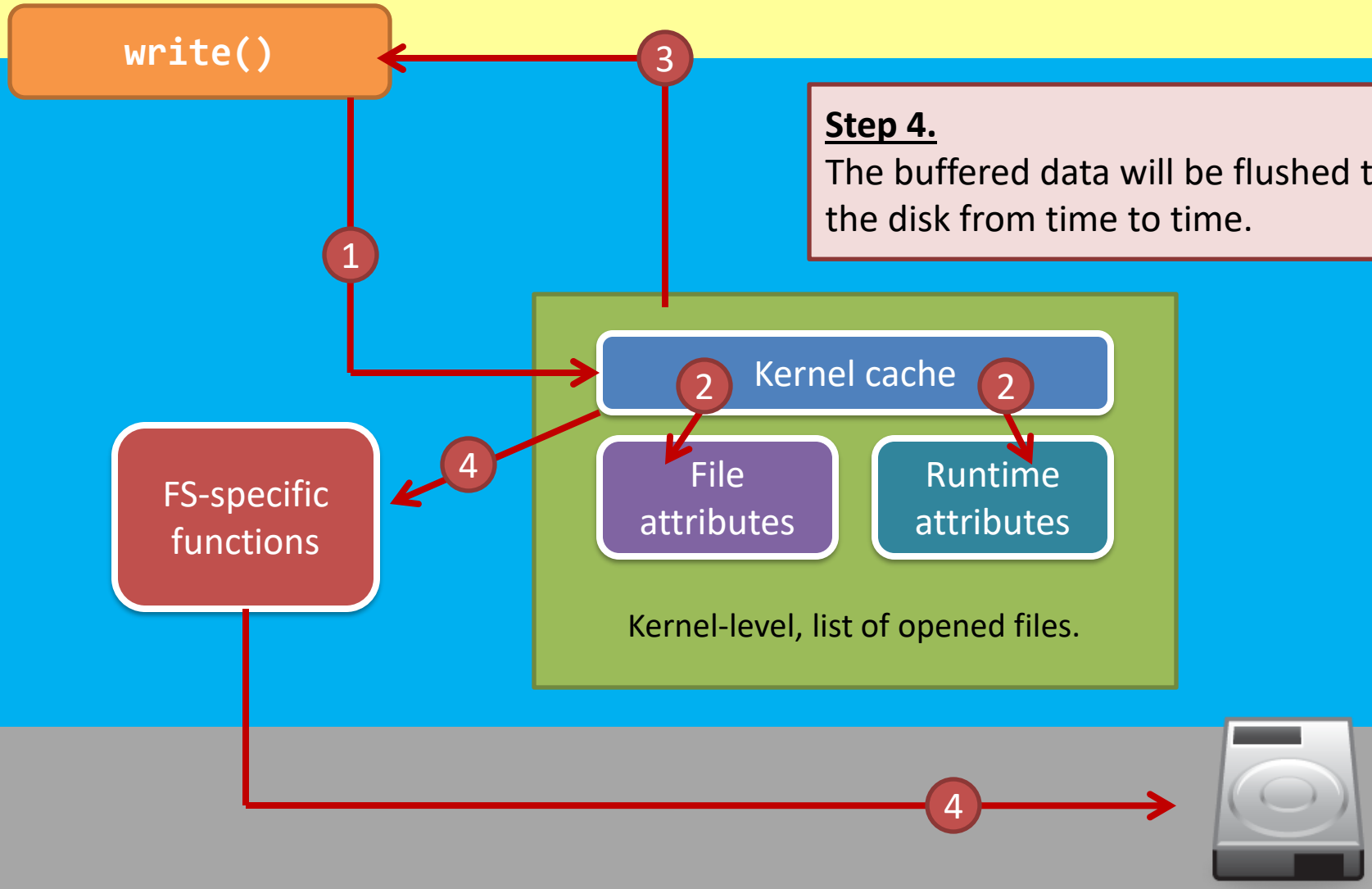
`read()` system call



`write()` system call



`write()` system call



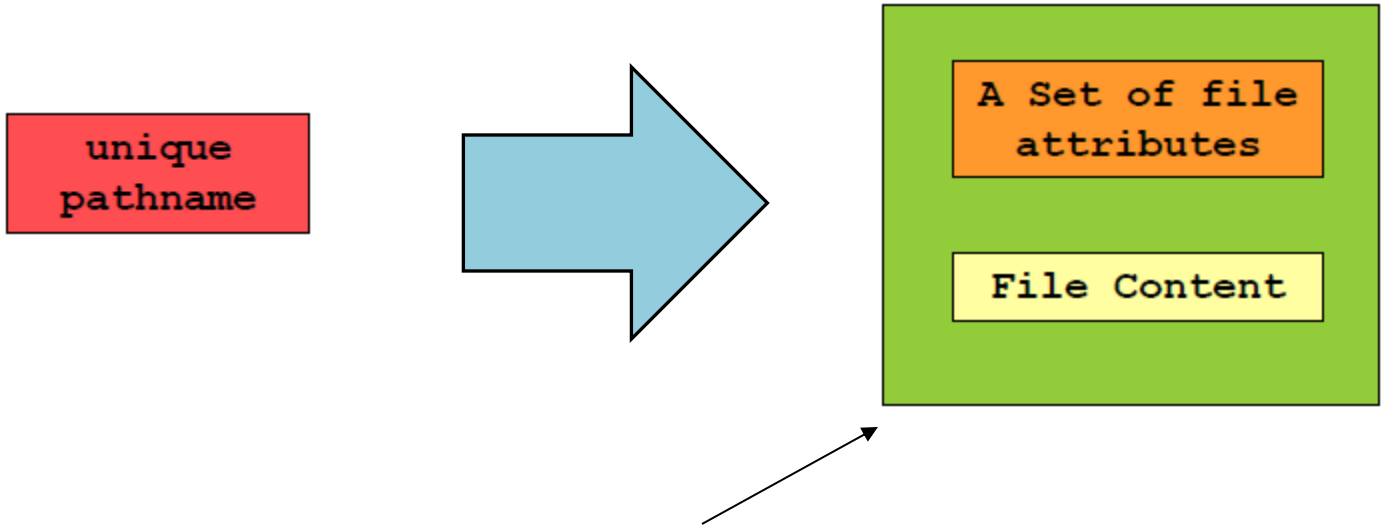
The kernel buffer cache implies…

- Performance
  - Increase reading performance?
  - Increase writing performance?

- Problem
  - Can you answer me why **you cannot press the reset button**?
  - Can you answer me **why you need to press the "eject" button before removing USB drives?**

Short Summary

- Every file has its unique pathname.
  - Its pathname leads you to its attributes and the file content.



A file has **two** important components! Plus, there are usually stored **separately**.

Short Summary

- We only introduce the read/write flow:
  - File writing involves **disk space allocation**; but…
  - The allocation of disk space is highly related to the design of the layout of the FS.
  - Also, the same case for the de-allocation of the disk space…

Summary of part 1

- In this part, we have an introduction to FS
  - File and directory
  - The truth about the calls that we usually use,
  - We learned: The content of a file is not the only entity, but also the file attributes.

- In the next part, we will go into the disk:
  - How and where to store the file attributes?
  - How and where to store the data?
  - How to manage a disk?