File Systems

Jo, Heeseung

Today's Topics

File system basics

Directory structure

File system mounting

Basic Concepts

Requirements for long-term information storage

- Store a very large amount of information
- Survive the termination of the process using it
- Access the information concurrently by multiple processes

File

- A named collection of related information that is recorded on secondary storage
  - Persistent through power failures and system reboots
- OS provides a uniform logical view of information storage via files

File system

- Implement an abstraction for secondary storage (files)
- Organizes files logically (directories)
- Permit sharing of data between processes, people, and machines
- Protect data from unwanted access (security)

Storage: A Logical View

Abstraction given by block device drivers:

Operations

- Identify(): returns N
- Read(start sector #, # of sectors)
- Write(start sector #, # of sectors)

Source: Sang Lyul Min (Seoul National Univ.)

File System Basics (1)

For each file, we have

- File name
- File attributes (metadata)
  - File size
  - Owner, access control lists
  - Creation time, last access time, last modification time, …
- File contents (data)
  - File systems normally do not care what they are

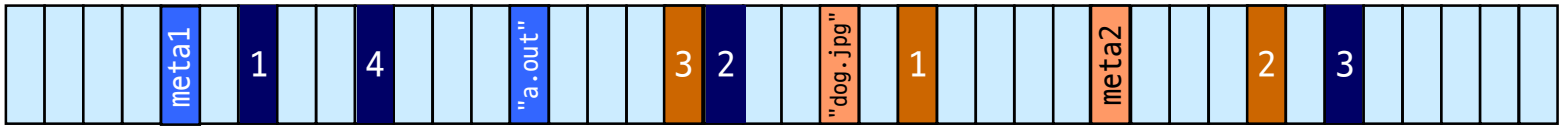
File access begins with ...

- File name
  - open ("/etc/passwd", O_RDONLY);

File System Basics (2)

File system: A mapping problem

- <filename, metadata, data> <a set of blocks>



File System Basics (3)

Goals

- Performance + Reliability

Design issues

- What information should be kept in metadata?
- How to locate metadata?
  - Mapping from pathname to metadata
- How to locate data blocks?
- How to manage metadata and data blocks?
  - Allocation, reclamation, free space management, etc.
- How to recover the file system after a crash?
- ...

File Attributes

Attributes or metadata

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

File Operations

Unix operations

```
int creat(const char *pathname, mode_t mode);
int open(const char *pathname, int flags, mode_t mode);
int close(int fd);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
off_t lseek(int fd, off_t offset, int whence);
int stat(const char *pathname, struct stat *buf);
int chmod(const char *pathname, mode_t mode);
int chown(const char *pathname, uid_t owner, gid_t grp);
int flock(int fd, int operation);
int fcntl(int fd, int cmd, long arg);
```

Directories

Directories

- For users, provide a structured way to organize files
- For the file system, provide a convenient naming interface
  - Allows the implementation to separate logical file organization from physical file placement on the disk

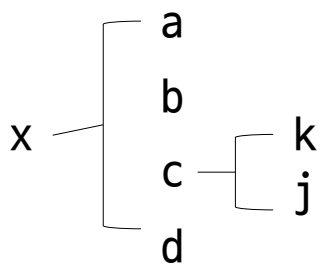
A hierarchical directory system

- Most file systems support multi-level directories
- Most file systems support the notion of a current directory (or working directory)
- Relative names (path) specified with respect to current directory
  - e.g. cd ../../../foo/bar/../bar
- Absolute names (path) start from the root of directory tree
  - e.g. cd /tmp/foo/bar

Directory Internals

A directory is …

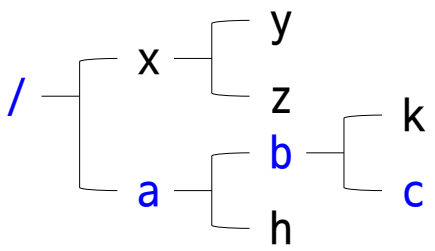
- Typically just a file that contains special metadata
- Directory = list of (file name, file attributes)
- Attributes include such things as:
  - Size, protection, creation time, access time,
  - Location on disk, etc.
- Usually unordered (effectively random)
  - Entries usually sorted by program that reads directory



Pathname Translation

open("/a/b/c", …)

- Open directory "/"
  (well known, can always find)
- Search the directory for "a",
  get location of "a"



- Open directory "a", search for "b", get location of "b"
- Open directory "b", search for "c", get location of "c"
- Open file "c"
- Of course, permissions are checked at each step

System spends a lot of time walking down directory path

- This is why open() is separate from read()/write()
  - read("/a/b/c", buf, 100); ??
- OS will cache prefix lookups to enhance performance
  - /a/b, /a/bb, /a/bbb, etc.
  - All share the "/a" prefix

Directory Operations

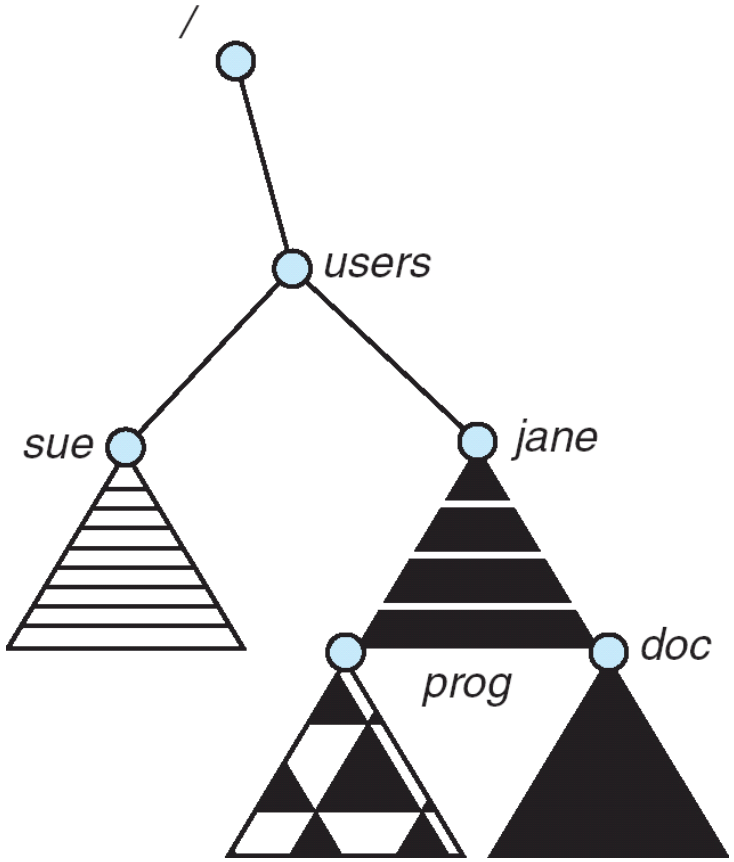
Unix operations

- Directories implemented in files
  - Use file operations to manipulate directories
- C runtime libraries provides a higher-level abstraction for reading directories
  - `DIR *opendir(const char *name);`
  - `struct dirent *readdir(DIR *dir);`
  - `void seekdir(DIR *dir, off_t offset);`
  - `int closedir(DIR *dir);`
- Other directory-related system calls
  - `int rename(const char *oldpath, const char *newpath);`
  - `int link(const char *oldpath, const char *newpath);`
  - `int unlink(const char *pathname);`

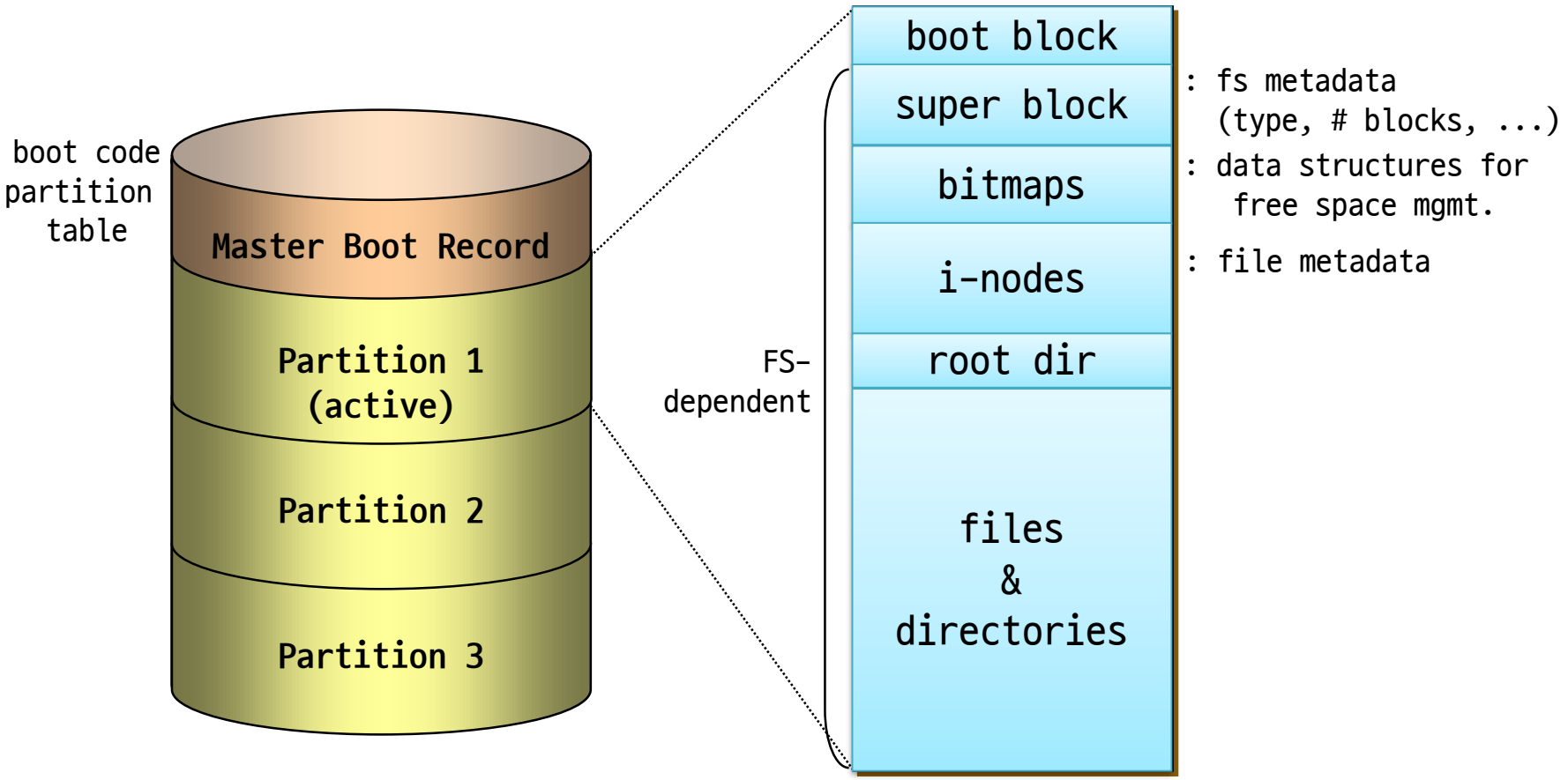
File System Mounting

Mounting

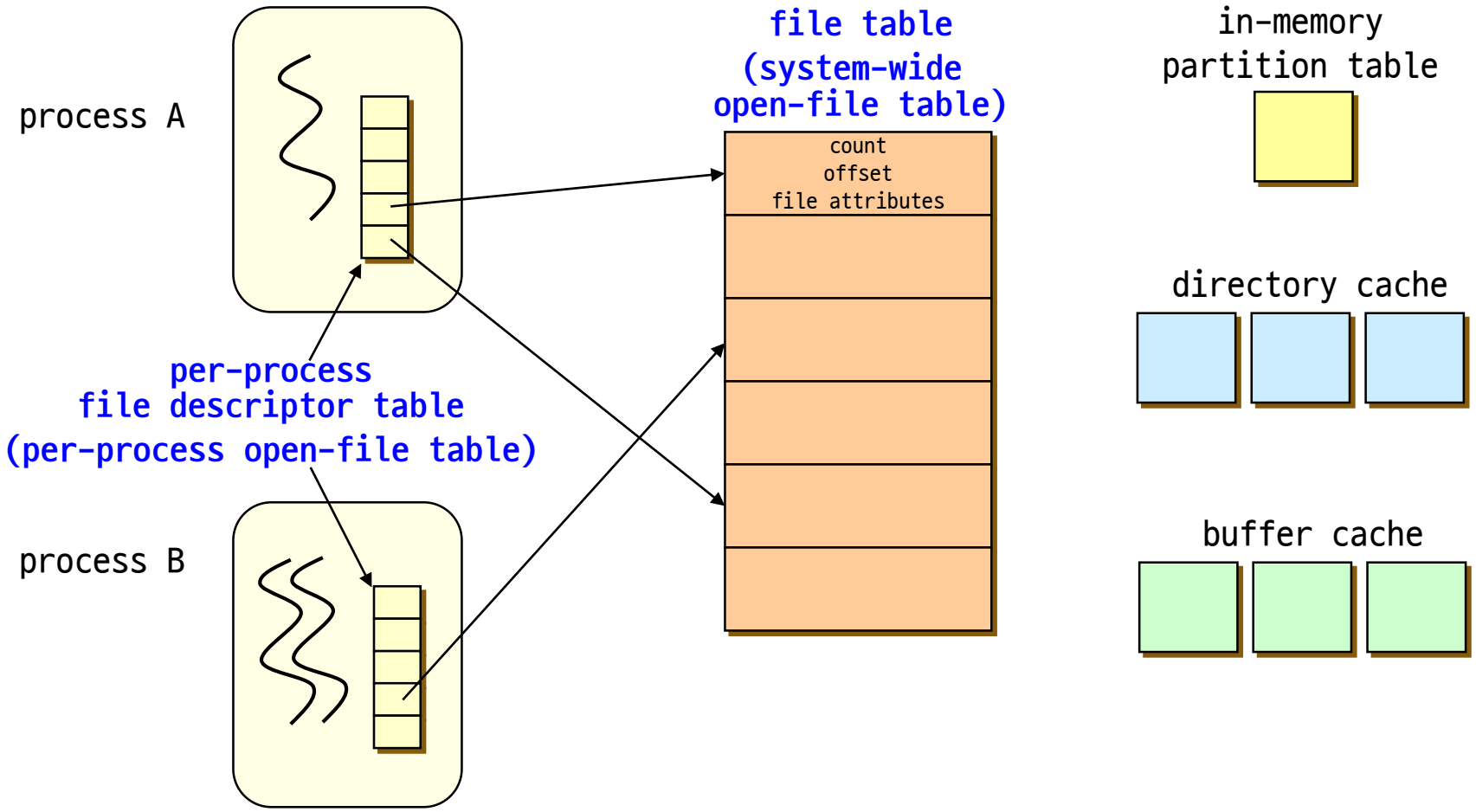
- A file system must be mounted before it can be available to processes on the system
  - Windows: to drive letters (e.g., C:\, D:\, …)
  - Unix: to an existing empty directory (= mount point)



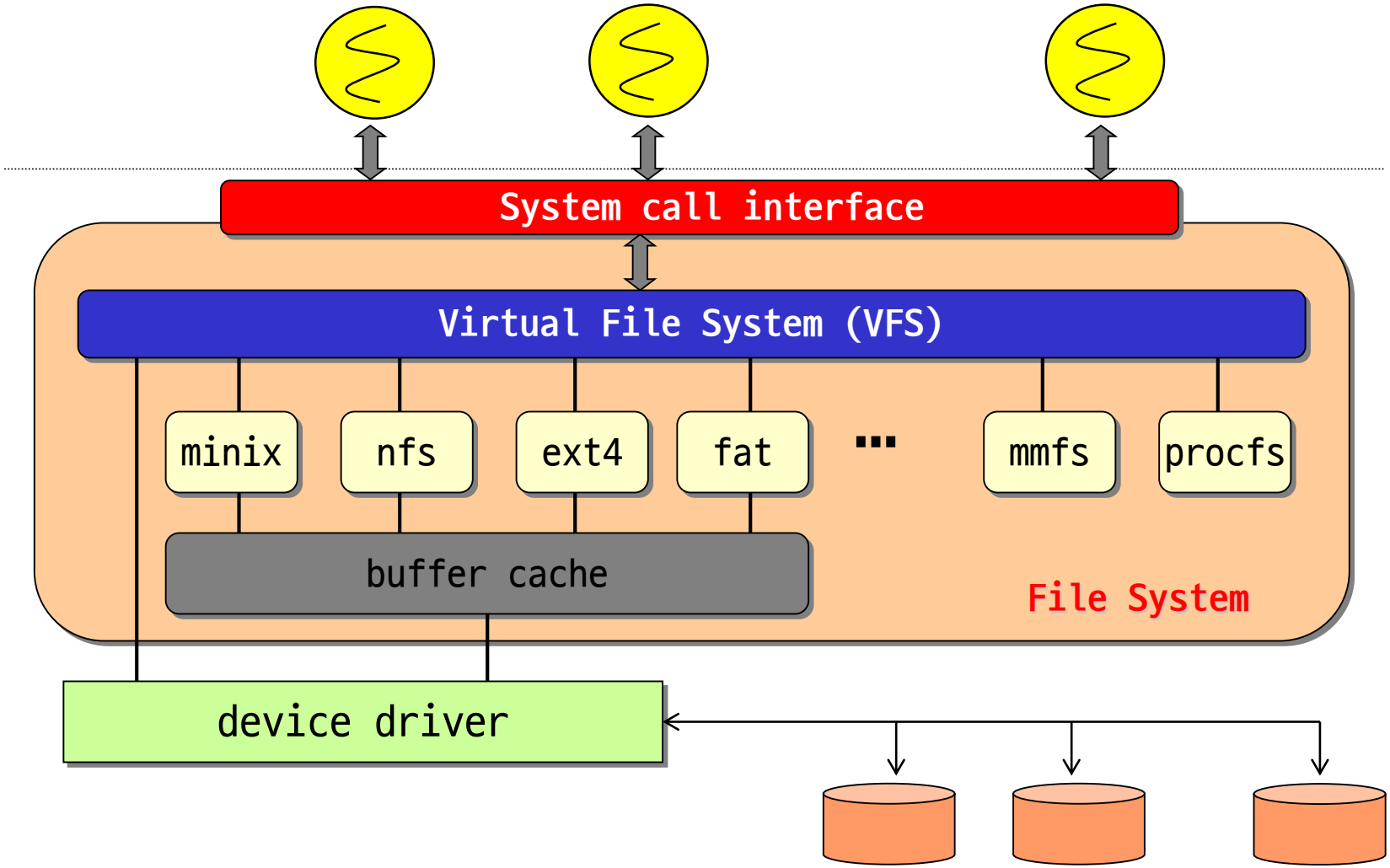
Disk Layout


boot code
partition
table
Master Boot Record
Partition 1
(active)
Partition 2
Partition 3
FS-
dependent
boot block
super block
bitmaps
i-nodes
root dir
files
&
directories
: fs metadata
(type, # blocks, ...)
: data structures for
free space mgmt.
: file metadata

In-memory Structures



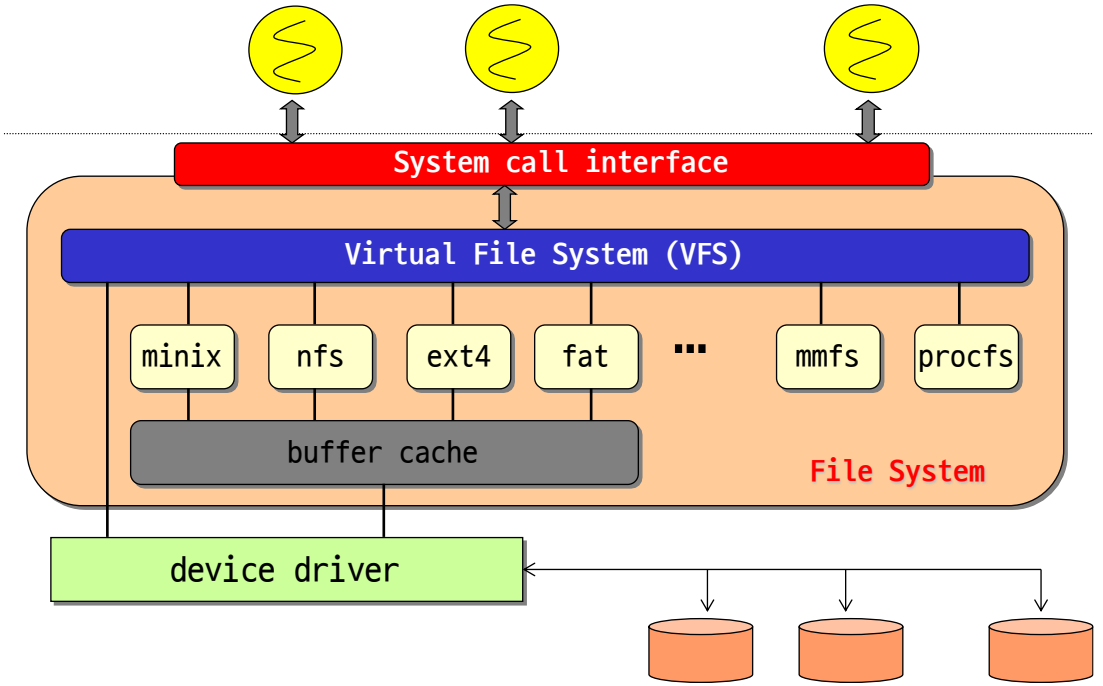
File System Internals



VFS (1)

Virtual File System

- Manages kernel-level file abstractions in one format for all file systems
- Receives system call requests from user-level
  - e.g., open, read, write, close, stat, etc.
- Interacts with a specific file system based on mount point traversal



VFS (2)

Linux: VFS common file model

- The superblock object
  - stores information concerning a mounted file system
- The dentry object
  - stores information about the linking of a directory entry with the corresponding file
- The inode object
  - stores general information about a specific file
- The file object
  - stores information about the interaction between an open file and a process