Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

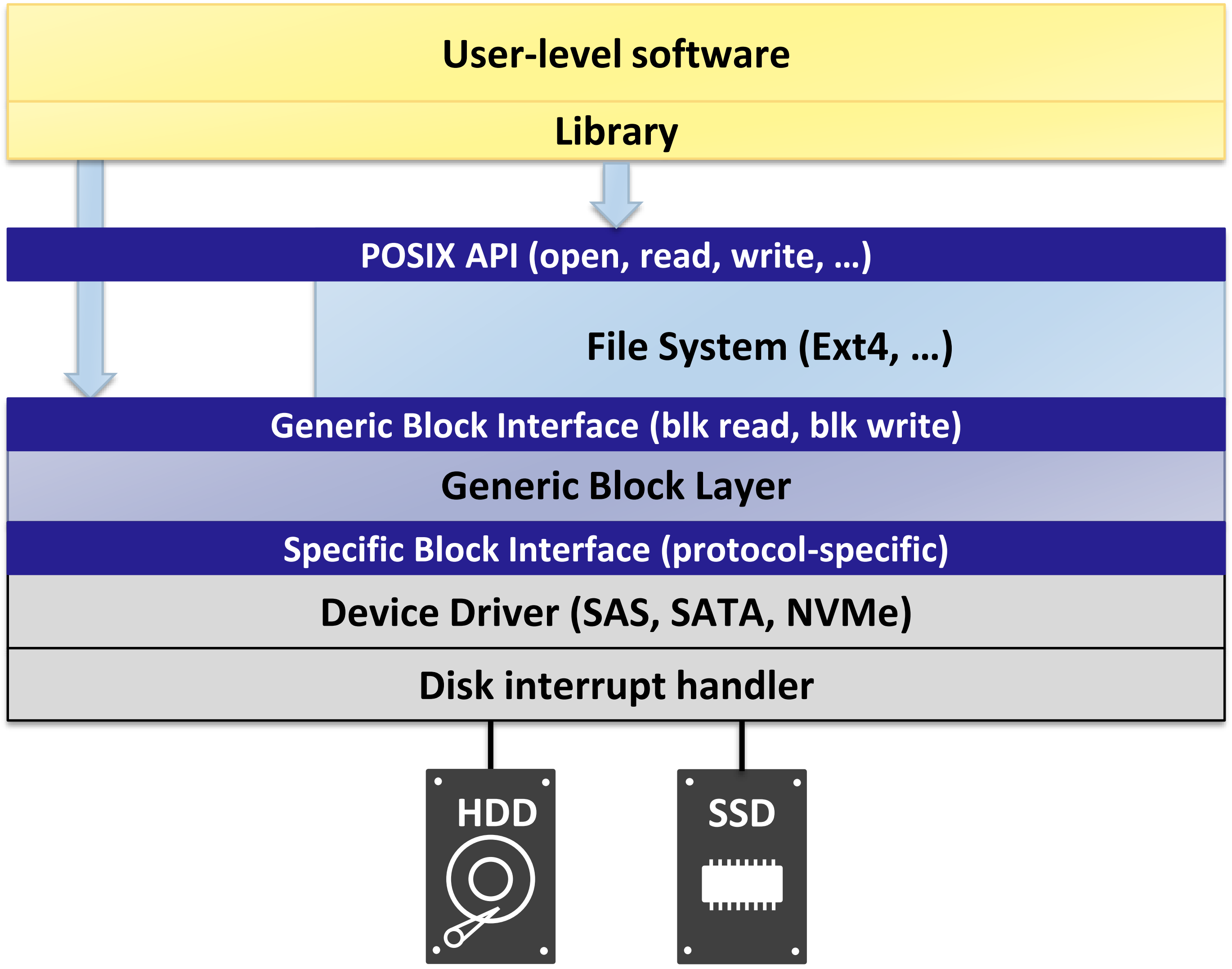
Systems Software &
Architecture Lab.

Seoul National University

Spring 2024

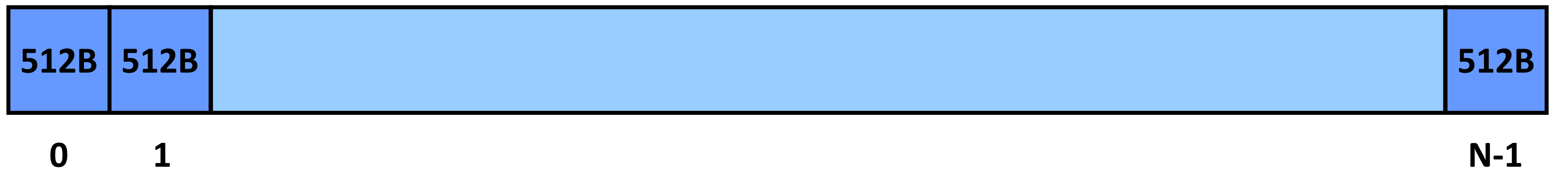
# File Systems

## File System Layers


User-level software
Library
POSIX API (open, read, write, …)
File System (Ext4, …)
Generic Block Interface (blk read, blk write)
Generic Block Layer
Specific Block Interface (protocol-specific)
Device Driver (SAS, SATA, NVMe)
Disk interrupt handler
HDD
SSD

## Storage: A Logical View

- Block interface abstraction



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

Source: Sang Lyul Min (Seoul National Univ.)

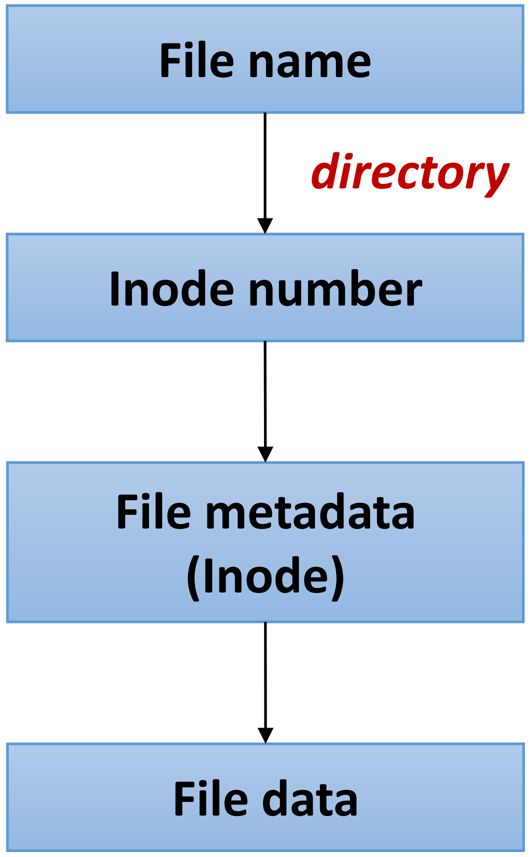
# Abstraction for Storage

- File
  - A named collection of related information that is recorded on persistent storage
  - Each file has an associated inode number (internal file ID)
  - Inodes are unique within a file system

- Directory
  - Provides a structured way to organize files
  - A special file used to map a user-readable file name to its inode number: a list of <file name, inode number>
  - Hierarchical directory tree: directories can be placed within other directories

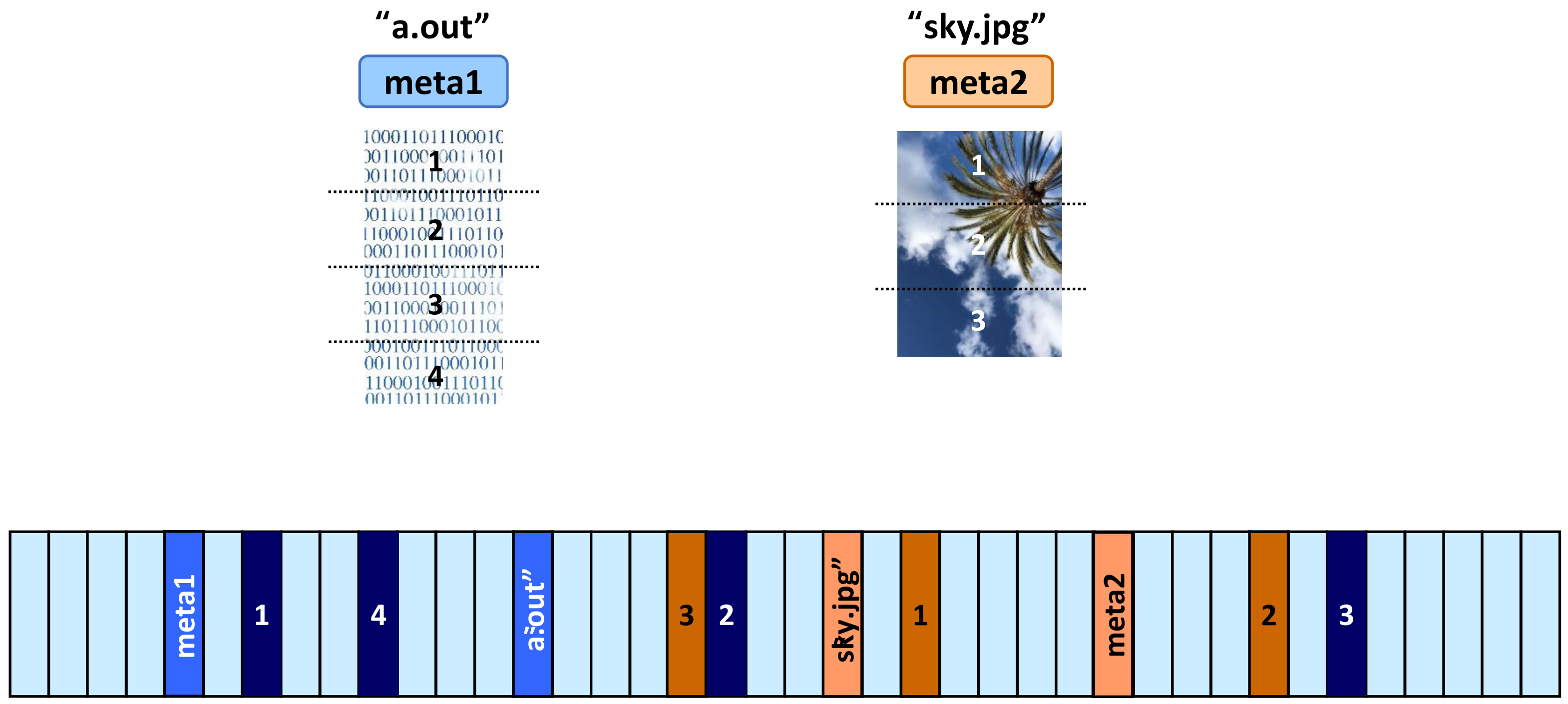
# File System Components

- File contents (data)
  - A sequence of bytes
  - File systems normally do not care what they are
- File attributes (metadata or inode)
  - File size
  - Block locations
  - Owner & access control lists
  - Timestamps, …
- File name
  - The full pathname from the root specifies a file
  - e.g., `open("/etc/passwd", O_RDONLY);`



# File System: A Mapping Problem

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



## File System Design Issues

- Goals
  - Performance, Reliability, Scalability, …
- Design issues
  - What information should be kept in metadata?
  - How to locate metadata from file name?
    - Pathname → metadata
  - How to locate data blocks?
    - <Metadata, offset> → Data block
  - How to manage metadata and data blocks?
    - Allocation, reclamation, free space management, etc.
  - How to recover the file system after a crash?
  - …

# File Attributes

- POSIX Inode
  - File type: regular, directory, char/block dev, fifo, symbolic link, …
  - Device ID containing the file
  - Inode number
  - Access permission: *rwx* for owner(*u*), group(*g*), and others(*o*)
  - Number of hard links
  - User ID and group ID of the owner
  - File size in bytes
  - Number of 512B blocks allocated
  - Time of last access (atime), time of last modification (mtime), time of last metadata change (ctime)
  - …

## File Operations

```
int open(char *pathname, int flags, mode_t mode);
int creat(char *pathname, mode_t mode);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, void *buf, size_t count);
off_t lseek(int fd, off_t offset, int whence);
int close(int fd);
int fsync(int fd);
int rename(char *oldpath, char *newpath);
int unlink(char *pathname);
int stat(char *path, struct stat *buf);
int link(char *oldpath, char *newpath);
int symlink(char *oldpath, char *newpath);
int mount(char *source, char *target, char *fstype,
          unsigned long mountflags, void *data);
int umount(char *target);
```

## Pathname Translation

- `open(“/a/b/c”, …)`
  - Open directory “/” (well known, can always find)
  - Search the directory entry 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”
  - Permissions are checked at each step
- File system spends a lot of time walking down directory paths
  - OS caches prefix lookups to enhance performance

# Ensuring Persistence

- File system buffers writes into memory (“page cache”)
  - Write buffering improves performance
  - Up to 30 seconds in Linux
  - `fsync()` flushes all dirty data to disk, and tells disk to flush its write cache to the media too
  - Also flushes metadata information associated with the file
  - `fdatasync()` does not flush modified metadata

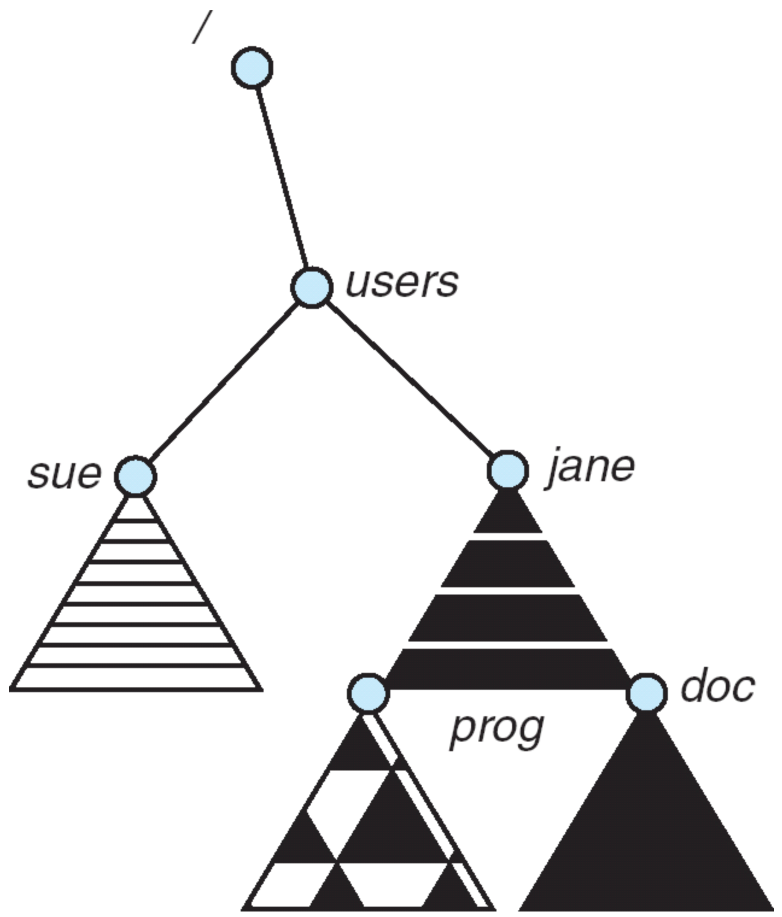
```
int fd = open(“foo”, O_CREAT | O_WRONLY | O_TRUNC);
int rc = write(fd, buffer, size);
rc = fsync(fd);
close(fd);
```

## Hard vs. Symbolic Links

- Hard link: `$ ln old.txt new.txt`
  - Both pathnames use the same inode number
  - Cannot tell which name was the "original"
  - Inode maintains the number of hard links
  - Deleting (unlinking) a file decreases the link count
  - Inode is removed only when the link count becomes 0
  - Does not work across a file system boundary
- Symbolic (or soft) link: `$ ln -s old.txt new.txt`
  - The new file contains a reference to another file or directory in the form of an absolute or relative pathname
  - "Shortcut" in Windows

# File System 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 ("______________")
  - Different file systems can be mounted in the same tree
  - Forms a large, single directory tree



## Consistency Semantics

- Unix semantics
  - Files can be shared among processes
  - Writes to an open file are visible immediately to other users that have this file open at the same time
- AFS ___________ semantics
  - Writes to an open file are not visible immediately
  - Once a file is closed, the changes made to it are visible only in sessions starting later
- Immutable-shared-files semantics
  - Once a file is declared as shared by its creator, it cannot be modified

## Summary

- Storage
  - Abstraction: a sequence of fixed-size blocks
  - read(# start sector, # of sectors to read, buffer addresses)
  - write(# start sector, # of sectors to write, buffer addresses)

- File system
  - Abstraction: a hierarchy of variable-size files and directories
  - open(pathname, flags)
  - read(file descriptor, size in bytes to read, buffer address)
  - write(file descriptor, size in bytes to write, buffer address)
  - close(file descriptor)