NOTICE

- Deadlines
  - (4/20 11:59 PM) Midterm quiz 1
  - (4/27 11:59 PM) Programming assignment 2

- Others
  - PA 1 Grades will be announced today

CS 374: Operating Systems I

Part II – Files

M/W 12:00 – 1:50 PM (LINC #200)

Sanghyun Hong

sanghyun.hong@oregonstate.edu

PRELIMINARIES: UNIX AND LINUX

- *NIX: Operating Systems
  - UNIX (Paper)
    - 1969: The OS was developed by AT&T Bell Lab, written in assembly languages
    - 1972: C was developed by the same lab
      - "UNIX can run on hardware costing as little as $40,000" [1]

[1] The UNIX Time-Sharing System, *ACM SOSP* 1973

PRELIMINARIES: UNIX AND LINUX

- *NIX: Operating Systems
  - UNIX (Paper)
    - 1969: The OS was developed by AT&T Bell Lab, written in assembly languages
    - 1972: C was developed by the same lab
      - "UNIX can run on hardware costing as little as $40,000" [1]
  - Linux
    - 1991: Open-source OS, developed by Linus Torvalds
      - Studied at the University of Helsinki
      - His master's thesis: "Linux: a Portable Operating System" (Thesis) [2]

        ≫"while the Linux project has been closely associated with me personally, partly due to the name, I'd like to make it very clear that the Linux OS is a huge project done co-operatively by lots of people all over the world … Thanks to all of you."

[1] The UNIX Time-Sharing System, *ACM SOSP* 1973
[2] Linux: a Portable Operating System, Linus Torvalds

PRELIMINARIES: UNIX AND LINUX

- *NIX: Operating Systems
  - UNIX (Paper)
    - 1969: The OS was developed by AT&T Bell Lab, writ
    - 1972: C was developed by the same lab
      - "UNIX can run on hardware costing as little as $
  - Linux
    - 1991: Open-source OS, developed by Linus Torvalds
      - Studied at the University of Helsinki
      - His master's thesis: "Linux: a Portable Operatin
        - ≫"while the Linux project has been closely a to the name, I'd like to make it very clear t co-operatively by lots of people all over th
      - Linus Torvalds lives in **Oregon**

PRELIMINARIES: *NIX VS. POSIX

- *NIX: Operating Systems
  - UNIX (Paper)
  - Linux (Thesis)

- POSIX: Portable Operating System Interface (for UniX)
  - OS standard specified by IEEE
  - Defines standard interfaces for system- and user-level APIs
  - Made the applications are portable between Unix-like OSes

[1] The UNIX Time-Sharing System, *ACM SOSP* 1973
[2] Linux: a Portable Operating System, Linus Torvalds

TOPICS FOR TODAY

- Part II: Files and File System Basics
  - Provide abstractions
    - What is a file (and a directory)?
    - What is access control/permission?
  - Offer standard interface
    - How do we create/read/write a file?
    - How do we modify access/permission?
  - Manage resources
    - How does OS manage files and directories?

PROVIDE ABSTRACTION: WHAT IS A FILE?

- File
  - **Definition:** a named collection of data (*e.g.*, movie.csv containing movie data)
    - The OS does not care the content – it is just bytes
    - The application(s) are interpreting the bytes
  - Composed of *two parts*:
    - **Data** : the actual bytes
    - **Metadata**: name, size, permissions, timestamps, owner, …

PROVIDE ABSTRACTION: WHAT IS A FILE?

- File
  - **Definition:** a named collection of data (*e.g.*, movie.csv containing movie data)
    - The OS does not care the content – it is just bytes
    - The application(s) are interpreting the bytes
  - Composed of *two parts*:
    - **Data** : the actual bytes
    - **Metadata**: name, size, permissions, timestamps, owner, …
  - ***NIX OS : everything** is a file
    - Files on secondary storages, *e.g.*, disks
    - Devices (mouse, keyboard, monitor, …)
    - Network devices (network card, sockets in OS, …)
    - Inter-process communications (pipes, sockets, …)

PROVIDE ABSTRACTION: WHAT IS A FILE?

- File
  - **Definition:** a named collection of data (*e.g.*, movie.csv containing movie data)
    - The OS does not care the content – it is just bytes
    - The application(s) are interpreting the bytes
  - Composed of *two parts*:
    - **Data** : the actual bytes
    - **Metadata**: name, size, permissions, timestamps, owner, …
  - ***NIX OS : everything** is a file
    - hexdump -C /bin/ls | head # an executable is a file
    - cat /proc/cpuinfo # CPU info – read like a file
    - cat /dev/urandom | head -c 8 # A device – read like a file
    - file <your file> # OS guesses type from *magic* bytes

PROVIDE ABSTRACTION: WHAT IS A FILE?

- File metadata
  - OS standard interface
    - stat(path, &sb): read metadata by filename
    - fstat(fd, &sb) : read metadata from an open file descriptor
  - Fields
    - st_size : size in bytes
    - st_mode : file type and permission bits
    - st_nlink : number of hard links
    - st_ino : inode number
    - st_mtime : last modification time

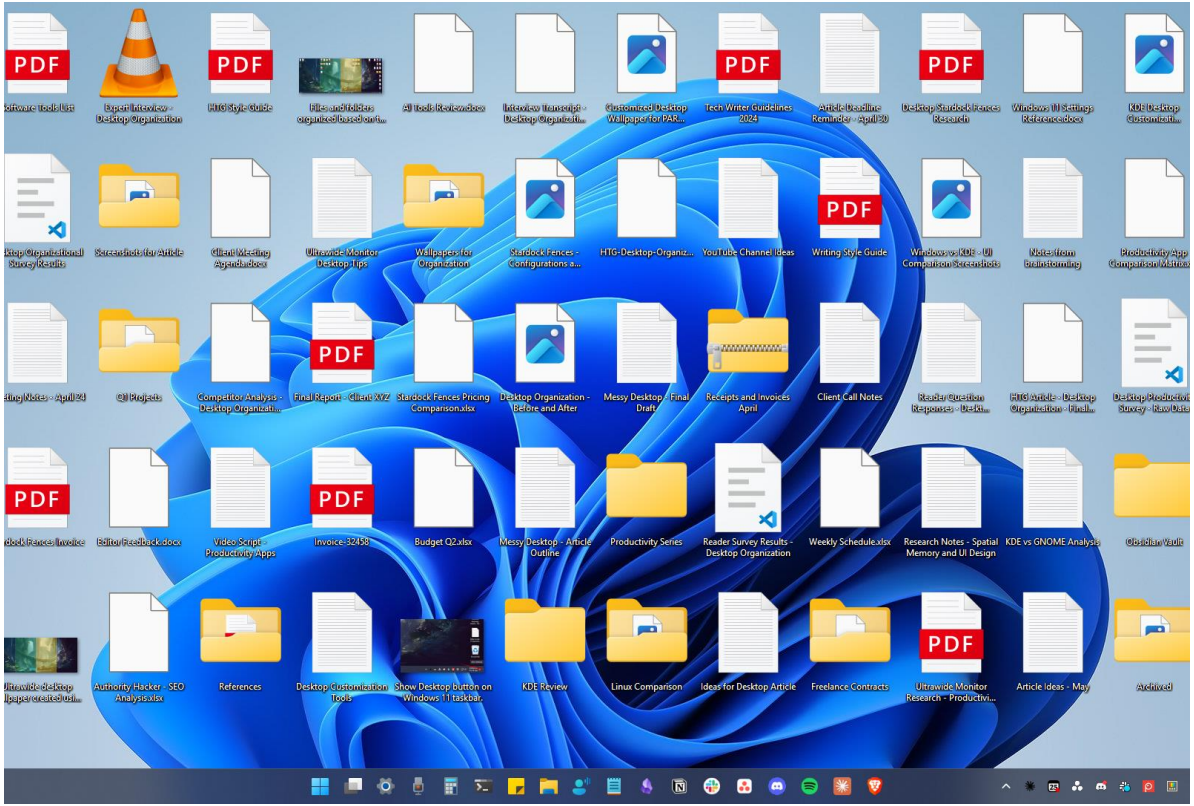
```
int main() {
  struct stat sb;
  stat("hello.txt", &sb);

  printf("Size:  %ld bytes\n",  sb.st_size);
  printf("Inode: %ld\n",       sb.st_ino);
  printf("Links: %ld\n",       sb.st_nlink);
  printf("Mode:  %o\n",        sb.st_mode & 0777);

  if (S_ISREG(sb.st_mode)) printf("Regular file\n");
  if (S_ISDIR(sb.st_mode)) printf("Directory\n");
  if (S_ISLNK(sb.st_mode)) printf("Symbolic link\n");
  return 0;
}
```

PROVIDE ABSTRACTION

- Files



PROVIDE ABSTRACTION

- Files
  - **Motivation:**
    - Scenario: one day you create 100k+ files and the next day, you want to use them
  - **Solutions :**
    - **S0:** You are Von Neumann; remember all the files
    - **S1:** Your system creates a folder containing all the files for each user
    - **S2:** Your system creates multiple folders containing the same kinds

PROVIDE ABSTRACTION: WHAT IS A DIRECTORY?

- Directories
  - **Definition:** a folder containing files and directories

PROVIDE ABSTRACTION: WHAT IS A DIRECTORY?

- Directories
  - ~~**Definition:** a folder containing files and directories~~
  - **Definition:** a special *file* whose content is a list of (name, inode#) pairs

PROVIDE ABSTRACTION: WHAT IS A DIRECTORY?

- Directories
  - ~~**Definition:** a folder containing files and directories~~
  - **Definition:** a special *file* whose content is a list of (name, inode#) pairs
    - The filename lives in the directory, not inside the file itself
    - OS's overhead for renaming file is *cheap* – why?
    - Directories form a *tree* – . (itself) and .. (parent)

PROVIDE ABSTRACTION: WHAT IS A DIRECTORY?

- Directories
  - ~~**Definition:** a folder containing files and directories~~
  - **Definition:** a special *file* whose content is a list of (name, inode#) pairs
    - The filename lives in the directory, not inside the file itself
    - OS's overhead for renaming file is *cheap* – why?
    - Directories form a *tree* – . (itself) and .. (parent)
  - **Examples**
    - ls -lai empty_dir
    - stat empty_dir # empty dir's inode number is 2 – why?

PROVIDE ABSTRACTION: PATH IN LINUX

- Path
  - **Definition:** a text string that identifies a file's location in the directory tree
  - Two kinds:
    - **Absolute path:** starts from the root / (the same regardless of where you are)
    - **Relative path :** starts from the current working directory (CWD) (depends on where you run)

PROVIDE ABSTRACTION: PATH IN LINUX

- Path
  - **Definition:** a text string that identifies a file's location in the directory tree
  - Two kinds:
    - **Absolute path:** starts from the root / (the same regardless of where you are)
    - **Relative path :** starts from the current working directory (CWD) (depends on where you run)

- Examples:
  - **Absolute path:**
    - '/nfs/stak/users/<ONID>/example' (that you can get from 'pwd' command)
  - **Relative path :**
    - './example_program'
    - Absolute path for this file: '/nfs/stak/users/<ONID>/example/example_program'
    - **Q1:** If we move to '/nfs/stak' what's its relative path?
    - **Q2:** If we move to HOME ('~/') what's its relative path?

PROVIDE ABSTRACTION: PATH IN LINUX

- Useful programming practices
  - 1) Suppose that a program will be used by multiple users
  - 2) Suppose that the program needs to read a common configuration file
  - 3) Suppose that a user who runs the program asks to read their file

Scenario 1) Absolute path.

A developer should put the program binary file (e.g., git) to a common location and users should use the absolute path to run this program in their shell.

Scenario 2) Both will be fine.

As a developer will use the same path for everyone, one can use an absolute path or a relative path from the program binary file.

Scenario 3) Relative path.

As the file shouldn't be read by any other users, the file will be located under a user's home dir. So, we can use a relative path from our home '~/' to the file.

PROVIDE ABSTRACTION: HARD LINKS VS. SYMBOLIC LINKS

- Hard link
  - **Definition:** another directory entry pointing to the same inode
    - Not a copy – shares the exact same data blocks
    - Data is deleted only when st_nlink becomes 0
    - Cannot cross filesystem boundaries

PROVIDE ABSTRACTION: HARD LINKS VS. SYMBOLIC LINKS

- Hard link and symbolic link
  - **Definition:** another directory entry pointing to the same inode
    - Not a copy – shares the exact same data blocks
    - Data is deleted only when st_nlink becomes 0
    - Cannot cross filesystem boundaries
  - **Definition:** a special file that stores a *path string*
    - Be able to point across filesystems, or to non-existent targets (dangling link)
    - stat() follows the link
    - lstat() inspects the link itself

PROVIDE ABSTRACTION: HARD LINKS VS. SYMBOLIC LINKS

- Hard link and symbolic link
  - echo "hello" > original.txt
  - ln original.txt hard.txt # create a hard link
  - ln -s original.txt soft.txt # create a symbolic link
  - ls -li original.txt hard.txt soft.txt # compare the inode number

  - rm original.txt
  - cat hard.txt # still works (st_nlink reduces to 1 from 2)
  - cat soft.txt # broken symlinks

PROVIDE ABSTRACTION: FILES AND DIRECTORIES

os1 ~/lecture/CS344-OS1$ ls -lh

total 312K

Permission	# hard-link	owner	owner-group	size (b)	last modified	name
drwxrwx---.	6	ONID	upg1xxxx	186	Apr 10 22:14	.
drwxrwx---.	3	ONID	upg1xxxx	73	Apr 5 19:58	..
drwxrwx---.	2	ONID	upg1xxxx	95	Apr 5 19:58	bufferoverflow
drwxrwx---.	2	ONID	upg1xxxx	52	Apr 4 09:02	bufferoverrun
lrwxrwxrwx.	1	ONID	upg1xxxx	22	Apr 10 22:14	home -> /nfs/stak/users/<ID>
-rw-rw----.	1	ONID	upg1xxxx	44	Apr 4 08:15	README.md
drwxrwx---.	2	ONID	upg1xxxx	79	Apr 5 20:07	thread

PROVIDE ABSTRACTION: FILES AND DIRECTORIES – CONT'D

os1 ~/lecture/CS344-OS1$ ls -alh

total 312K

Permission	# hard-link	owner	owner-group	size (b)	last modified	name
drwxrwx---.	6	ONID	upg1xxxx	186	Apr 10 22:14	.
drwxrwx---.	3	ONID	upg1xxxx	73	Apr 5 19:58	..
drwxrwx---.	2	ONID	upg1xxxx	95	Apr 5 19:58	bufferoverflow
drwxrwx---.	2	ONID	upg1xxxx	52	Apr 4 09:02	bufferoverrun
drwxrwx---.	8	ONID	upg1xxxx	299	Apr 10 21:56	.git
-rw-rw----.	1	ONID	upg1xxxx	430	Apr 5 19:56	.gitignore
lrwxrwxrwx.	1	ONID	upg1xxxx	22	Apr 10 22:14	home -> /nfs/stak/users/OID
-rw-rw----.	1	ONID	upg1xxxx	44	Apr 4 08:15	README.md
drwxrwx---.	2	ONID	upg1xxxx	79	Apr 5 20:07	thread

Hidden files!

PROVIDE ABSTRACTION: ACCESS CONTROL

- Users and groups
  - **Users** : an account, tied to actual users or that exists for specific applications
    - Physical users: Alice, Bob, …
    - Accounts for applications: root (sudo), httpd (Apache), ec2-user (AWS), …
  - **Groups:** a logical expr of an organization, tying users together for a common purpose
    - Linux services: daemon, …

PROVIDE ABSTRACTION: ACCESS CONTROL – USERS AND GROUPS

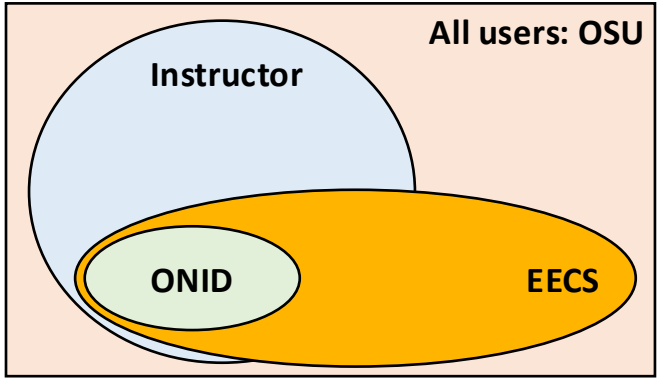
os1 ~/lecture/CS344-OS1$ ls -alh

```
total 312K
drwxrwx---.   6   ONID   upg1xxxx   186   Apr 10 22:14   .
drwxrwx---.   3   ONID   upg1xxxx   73    Apr  5 19:58   ..
drwxrwx---.   2   ONID   upg1xxxx   95    Apr  5 19:58   bufferoverflow

... <omit the entries>
```

Permission	# hard-link	owner	owner-group	size (b)	last modified	name

- Linux controls the access to files or directories based on three categories:
  - **u**ser : owner of a file or a directory
  - **g**roup : the group where users are
  - **o**thers: all the other users



PROVIDE ABSTRACTION: PERMISSION

- Permission
  - **Read** : one can read files and directories with ‘r’ permission
  - **Write** : one can write files and dirs. with ‘w’ permission
  - **Execute:** one can execute files and dirs. with ‘x’ permission

PROVIDE ABSTRACTION: PERMISSION – CONT'D

os1 ~/lecture/CS344-OS1$ ls -alh

total 312K

	… <omit the entries>					
-rw-rw----.	1	ONID	upg1xxxx	44	Apr 4 08:15	README.md
drwxrwx---.	2	ONID	upg1xxxx	79	Apr 5 20:07	thread
Permission	**# hard-link**	**owner**	**owner-group**	**size (b)**	**last modified**	**name**

- Permission representation
  - drwxrwx---

[Type] d: directory, -: file

[User] the first three letters (rwx)
[Group] the second three letters (rwx)
[Others] the last three letters (---)

PROVIDE ABSTRACTION: PERMISSION – CONT'D

os1 ~/lecture/CS344-OS1$ ls -alh

total 312K

	… <omit the entries>					
-rw-rw----.	1	ONID	upg1xxxx	44	Apr 4 08:15	README.md
drwxrwx---.	2	ONID	upg1xxxx	79	Apr 5 20:07	thread
Permission	**# hard-link**	**owner**	**owner-group**	**size (b)**	**last modified**	**name**

- Permission representation
  - drwxrwx---:
    - 770
    - 111111000

Interpretation

- Decimal #: 1st (user), 2nd (group), 3rd (others)
  + ex. 770 : 7 (user), 7 (group), 0 (others)

- Each #: Binary number
  1st (read), 2nd (write), 3rd (execute)
  + ex. 7 : 111 (rwx)
  + ex. 6 : 110 (rw)
  + ex. 600 : 110 000 000 (your ssh key)

TOPICS FOR TODAY

- Part II: Files and File System Basics
  - Provide abstractions
    - What is a file (and a directory)?
    - What is access control/permission?
  - Offer standard interface
    - How do we create/read/write a file?
    - How do we modify access/permission?
  - Manage resources
    - How does OS manage files and directories?

REVISIT: SYSTEM CALLS

- System call
  - **Definition:** a user-level function call to request a service from the OS
  - **Example:** when we run a program "exec(<a program file>)"

- Two ways to use a system call
  - **Terminal:** run a command (that is a system call)
  - **C** : call a system call function
  - **Example**: run "exec" in Terminal or use "exec(<a program file>)" function in C

OFFER STANDARD INTERFACE: USERS AND GROUPS

- System calls (in Terminal)
  - Print the user and group IDs : "id"
  - Create/modify/delete users : "useradd" / "usermod" / "userdel"
  - Create/modify/delete groups: "groupadd" / "groupmod" / "groupdel"

- System calls (in C)
  - Print the user and group IDs : "getuid()" / "getgid()"
  - Create/modify/delete users : No C APIs; we can use "system('useradd …')"
  - Create/modify/delete groups: No C APIs; we can use "system('groupadd …')"

OFFER STANDARD INTERFACE: USERS AND GROUPS

os1 ~/lecture/CS344-OS1$ ls -alh

Permission	# hard-link	owner	owner-group	size (b)	last modified	name
total 312K						
drwxrwx---.	6	ONID	upg1xxxx	186	Apr 10 22:14	.
drwxrwx---.	3	ONID	upg1xxxx	73	Apr 5 19:58	..
drwxrwx---.	2	ONID	upg1xxxx	95	Apr 5 19:58	bufferoverflow
… <omit the entries>						

- An example of 'id' system call

os1 ~/lecture/CS344-OS1$ id

uid=1xxxxx (ONID)
My user ID

gid=4xxxx (upg1xxxx)
My group ID

groups=4xxxx (upg1xxxx), 3xxx (cs-faculty)
Groups that I am associated with

OFFER STANDARD INTERFACE: PERMISSION

- System calls (in Terminal)
  - Change the ownership : “chown -R <user>:<group>”
  - Change the permission: “chmod -R <mode to set>”

- System calls (in C)
  - Change the ownership : “chown(const char *_path_, uid_t _owner_, gid_t _group_)”
  - Change the permission: “chmod(const char *_pathname_, mode_t _mode_)”

OFFER STANDARD INTERFACE: PERMISSION

os1 ~/lecture/CS344-OS1$ ls -alh

total 312K

… <omit the entries>

Permission	# hard-link	owner	owner-group	size (b)	last modified	name
-rw-rw----.	1	ONID	upg1xxxx	44	Apr 4 08:15	README.md
drwxrwx---.	2	ONID	upg1xxxx	79	Apr 5 20:07	thread

- Examples of "chown" and "chmod" commands
  - $ chown -R <someone>:<upg1xxxx> thread
  - $ chmod 644 README.md
  - $ chmod o+rw README.md
  - $ chmod 700 thread
  - $ chmod g-rwx thread

Rules
- Use a number, e.g., 644
- Use a string: user/group/others +/- perm.
  + ex. u+x (user can execute the file/dir)
  + ex. g-wx (group cannot write or execute it)

OFFER STANDARD INTERFACE: SOME USEFUL SYSTEM CALLS

- System calls (frequently used)
  - Get the details about a file : $ stat <file/dir name>
  - Create an empty file : $ touch <file/dir name>
  - Total size of a directory : $ du -alh <file/dir name>
  - Total filesystem size and info : $ df -h

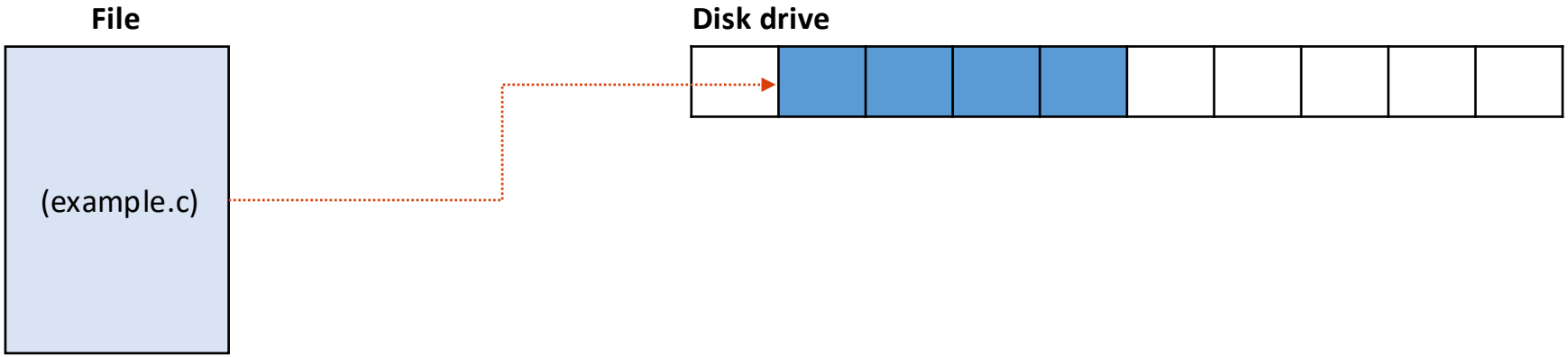
    …

TOPICS FOR TODAY

- Part II: Files and File System Basics
  - Provide abstractions
    - What is a file (and a directory)?
    - What is access control/permission?
  - Offer standard interface
    - How do we create/read/write a file?
    - How do we modify access/permission?
  - Manage resources
    - How does OS manage files and directories?

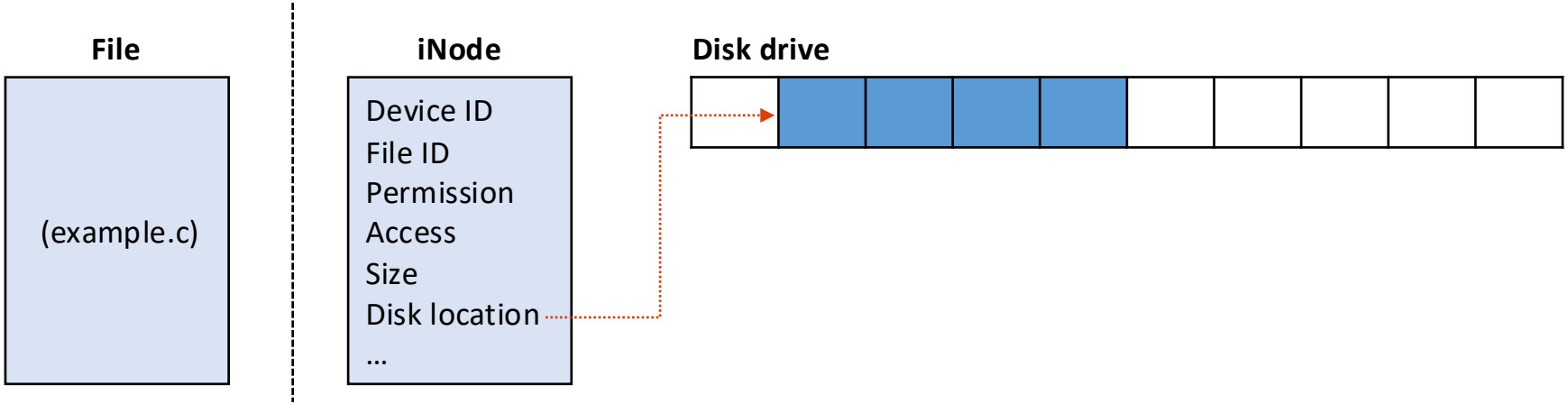
PROBLEM: HOW TO STORE FILES TO STORAGE

- Scenario 1: store a file to a disk drive
  - File: a sequence of data



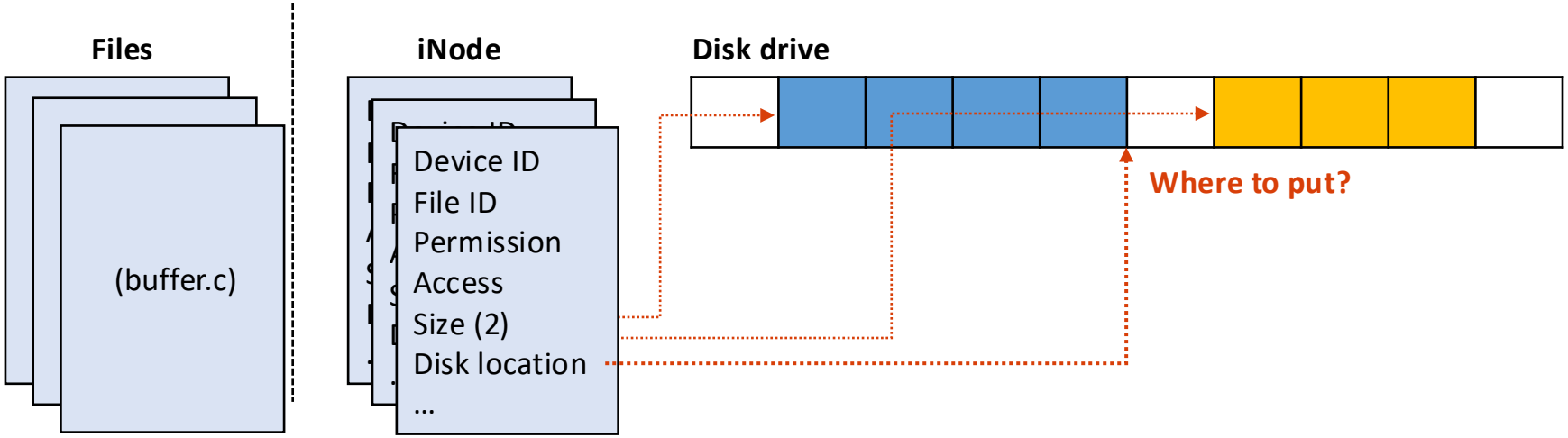
MANAGE RESOURCES: INODE STRUCTURE

- Scenario 1: store a file to a disk drive
  - **i(ndex)Node:** a data-structure that describes a file-system object, e.g., a file/dir.



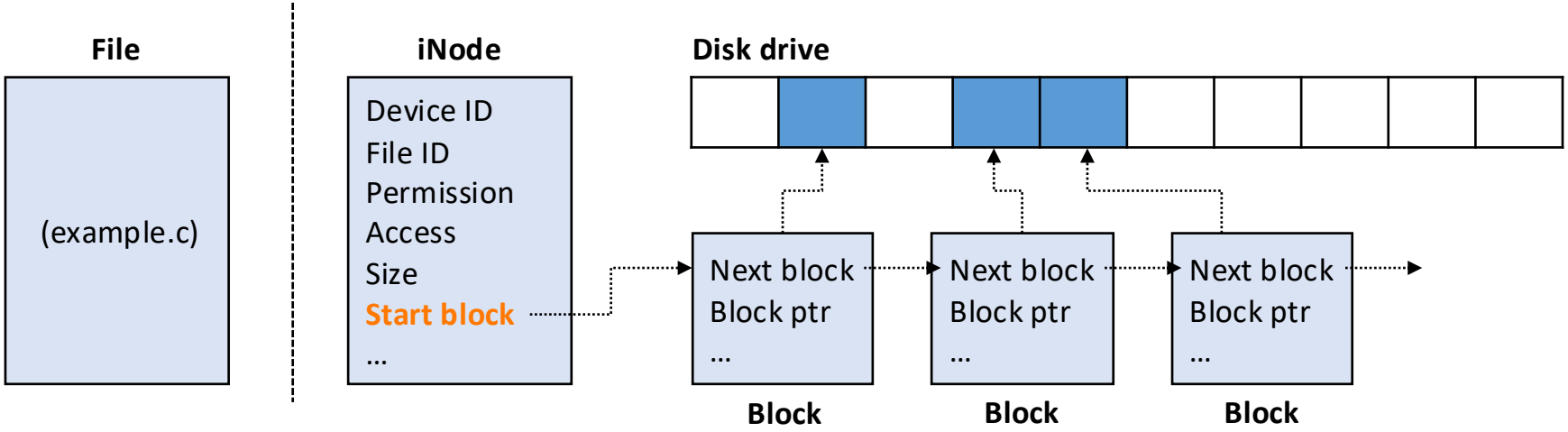
MANAGE RESOURCES: INODE STRUCTURE – CONT'D

- Scenario 2: store **multiple (> 2+)** files to a disk drive
  - **i(ndex)Node:** a data-structure that describes a file-system object, e.g., a file/dir.



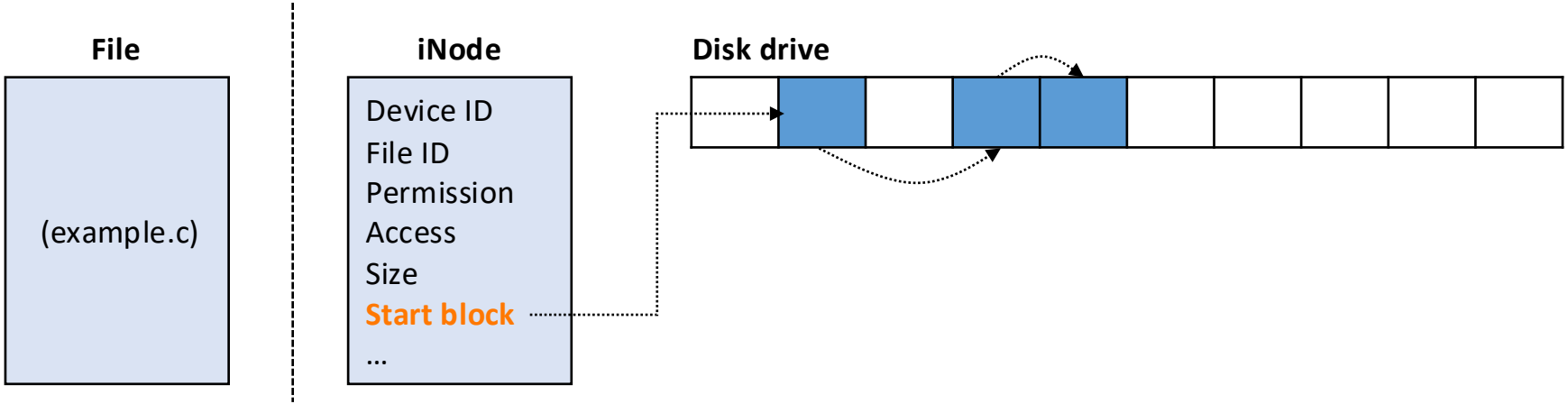
MANAGE RESOURCES: BLOCK STRUCTURE (FAT FILESYSTEM)

- Scenario 2: store **multiple (> 2+)** files to a disk drive
  - **i(ndex)Node:** a data-structure that describes a file-system object, e.g., a file/dir.
  - **Block** : a small(est) unit of data storage, defined by OS (4KB in Linux)



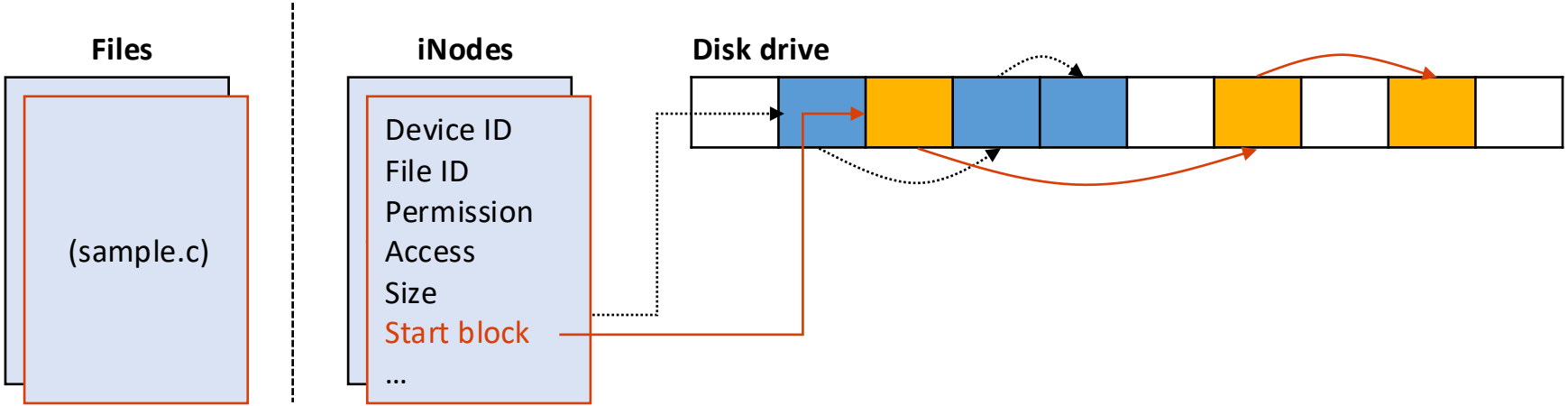
MANAGE RESOURCES: BLOCK STRUCTURE (FAT FILESYSTEM)

- Scenario 2: store **multiple (> 2+)** files to a disk drive
  - **i(ndex)Node:** a data-structure that describes a file-system object, e.g., a file/dir.
  - **Block** : a small(est) unit of data storage, defined by OS (4KB in Linux)



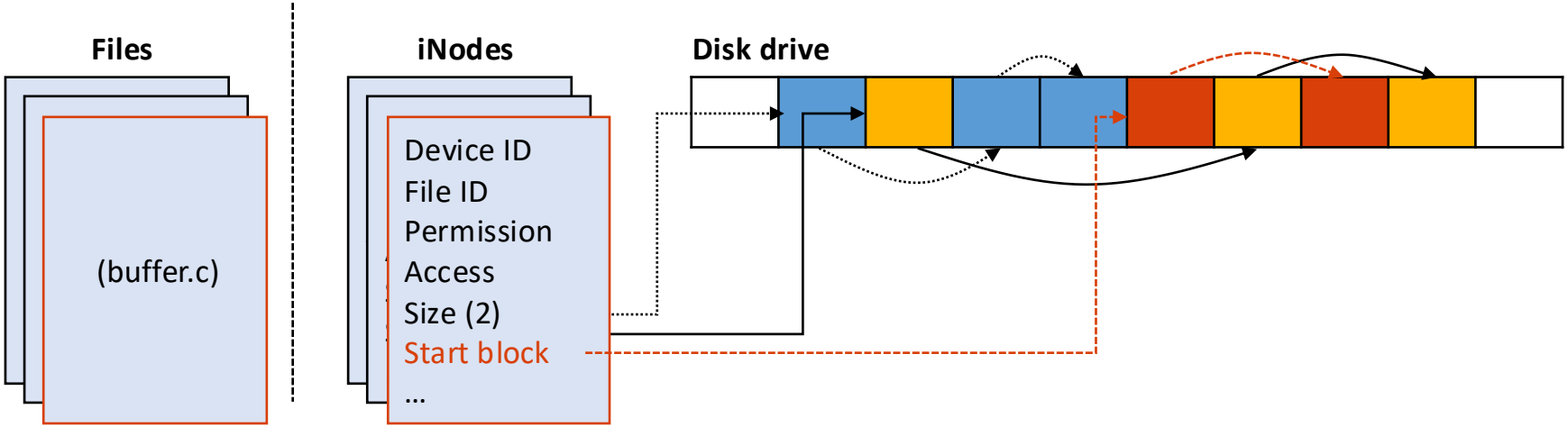
MANAGE RESOURCES: BLOCK STRUCTURE (FAT FILESYSTEM)

- Scenario 2: store **multiple (> 2+)** files to a disk drive
  - **i(ndex)Node:** a data-structure that describes a file-system object, e.g., a file/dir.
  - **Block** : a small(est) unit of data storage, defined by OS (4KB in Linux)



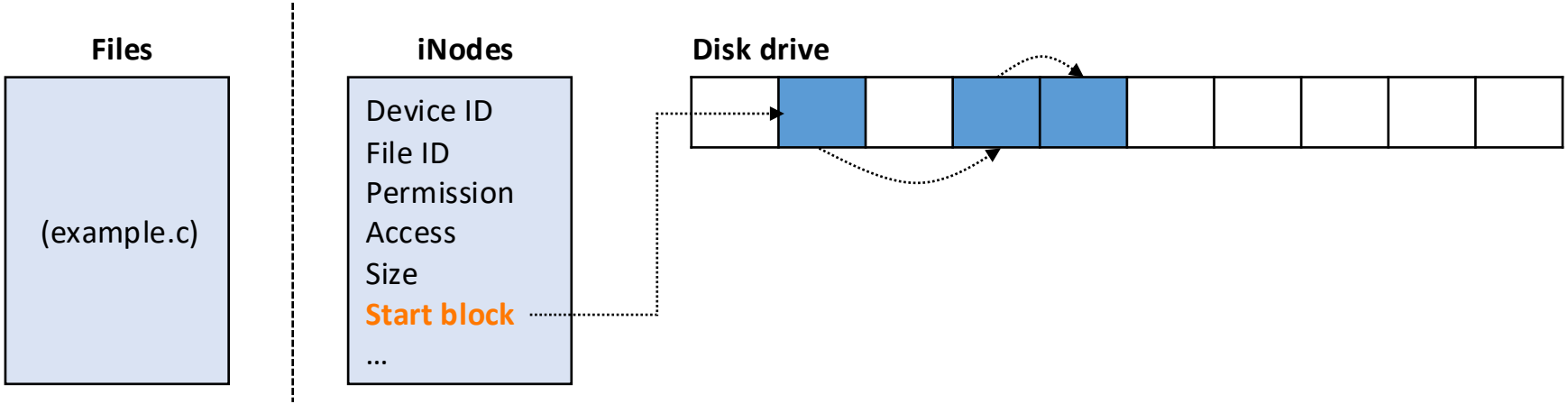
MANAGE RESOURCES: BLOCK STRUCTURE (FAT FILESYSTEM)

- Scenario 2: store **multiple (> 2+)** files to a disk drive
  - **i(ndex)Node:** a data-structure that describes a file-system object, e.g., a file/dir.
  - **Block** : a small(est) unit of data storage, defined by OS (4KB in Linux)



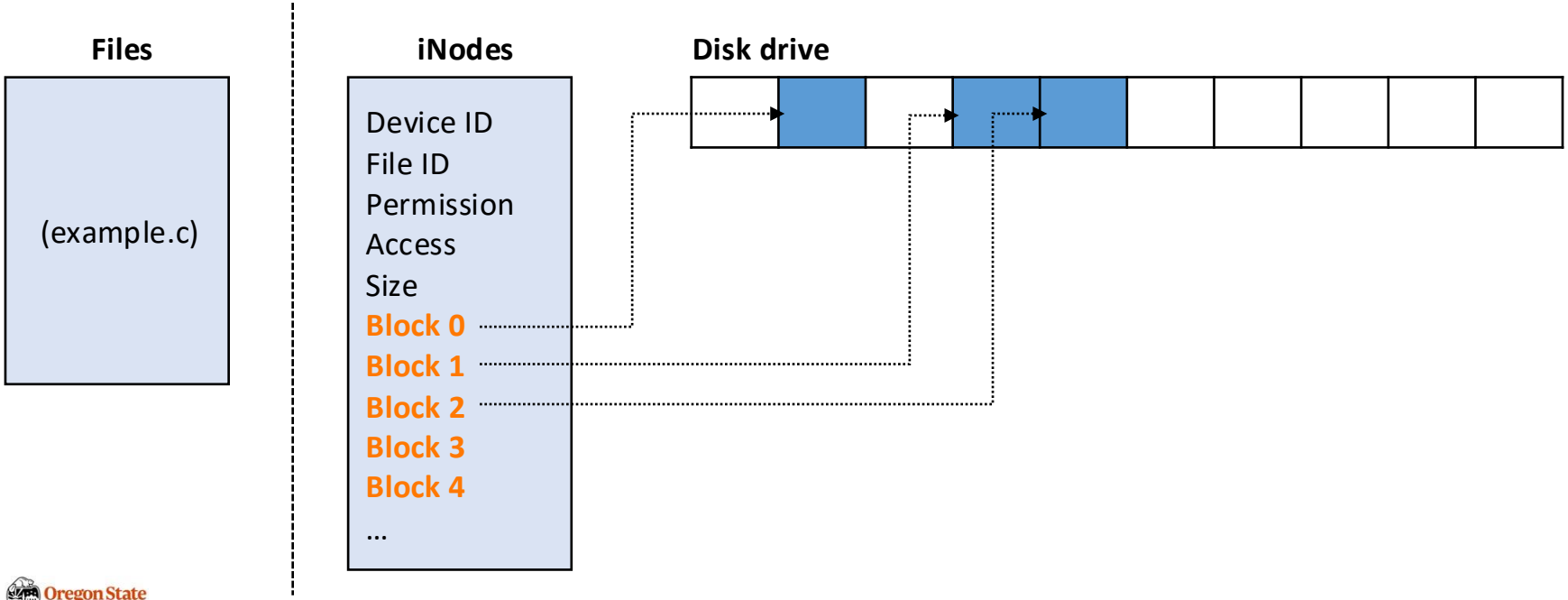
MANAGE RESOURCES: BLOCK STRUCTURE FOR *ACCESS EFFICIENCY*

- Scenario 3: Users access a certain block(s)
  - **i(ndex)Node:** a data-structure that describes a file-system object, e.g., a file/dir.
  - **Block** : a small(est) unit of data storage, defined by OS (4KB in Linux)

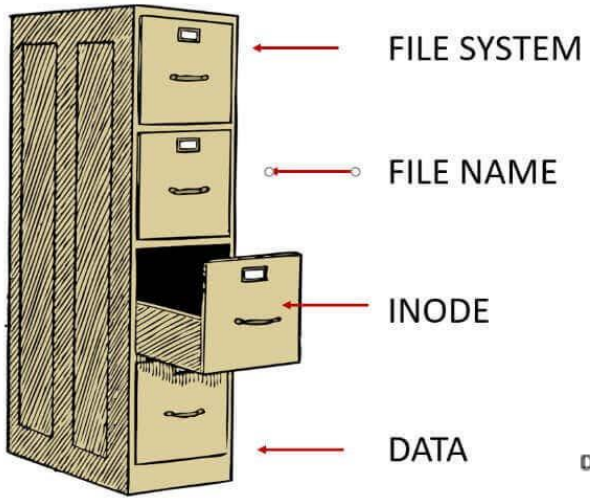


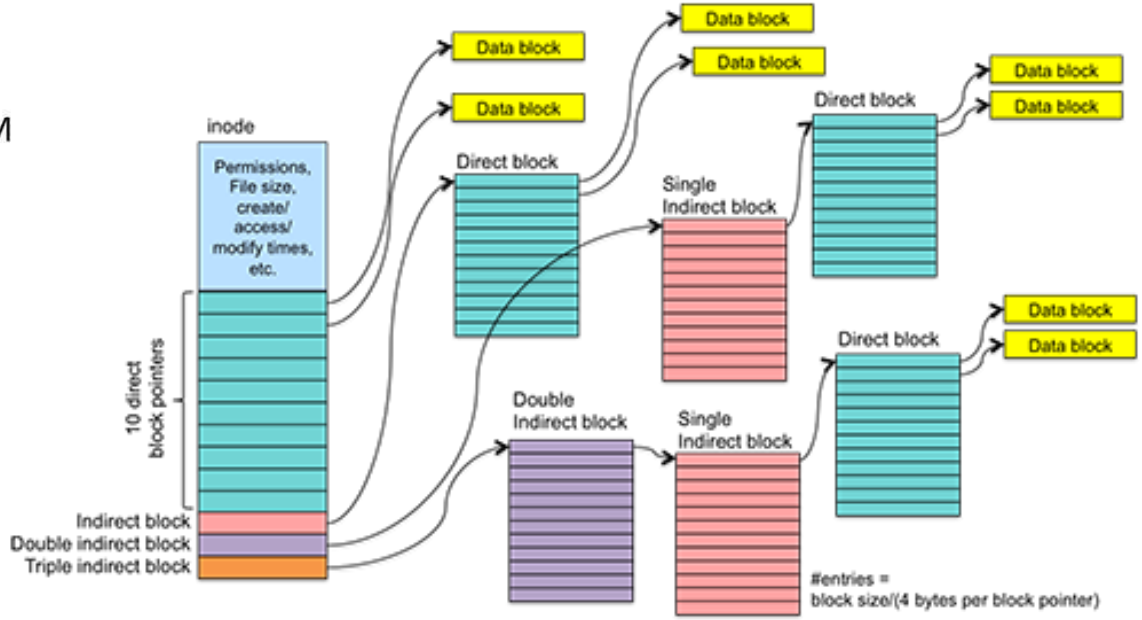
MANAGE RESOURCES: BLOCK STRUCTURE FOR *ACCESS EFFICIENCY* (EXT2/3)

- Scenario 3: Users access a certain block(s)
  - **i(ndex)Node:** a data-structure that describes a file-system object, e.g., a file/dir.
  - **Block** : a small(est) unit of data storage, defined by OS (4KB in Linux)



MANAGE RESOURCES: FILESYSTEM STRUCTURE OVERVIEW





[1] What Are inodes in Linux and How Are They Used? https://helpdeskgeek.com/linux-tips/what-are-inodes-in-linux-and-how-are-they-used/
[2] File System Design Case Studies, https://people.cs.rutgers.edu/~pxk/416/notes/13-fs-studies.html

TOPICS FOR TODAY

- Part II: Files and File System Basics
  - Provide abstractions
    - What is a file (and a directory)?
    - What is access control/permission?
  - Offer standard interface
    - How do we create/read/write a file?
    - How do we modify access/permission?
  - Manage resources
    - How does OS manage files and directories?

Thank You!

M/W 12:00 – 1:50 PM (LINC #200)

Sanghyun Hong

sanghyun.hong@oregonstate.edu