



# Project 6

# File System

COS 318

Fall 2015

# Project 6: File System



- Goal: Implement a simple UNIX-like file system with a hierarchical directory structure.
- Read the project spec for the details.
- Get a fresh copy of the start code from the lab machines (</u/318/code/project6/>).
- Start as early as you can. This is a long project and you will have to write a lot of code.

# Project 6: Schedule



- Design Review:
  - No design review!
  - You will submit a pdf file with your project that describes the details about the design of your file system.
  - Look at the “Design Document” section of the project description for the content of the document that you need to write.
- Due date: Tuesday 1/12/2016 (Dean’s date), 5:00pm.

# Project 6: Overview



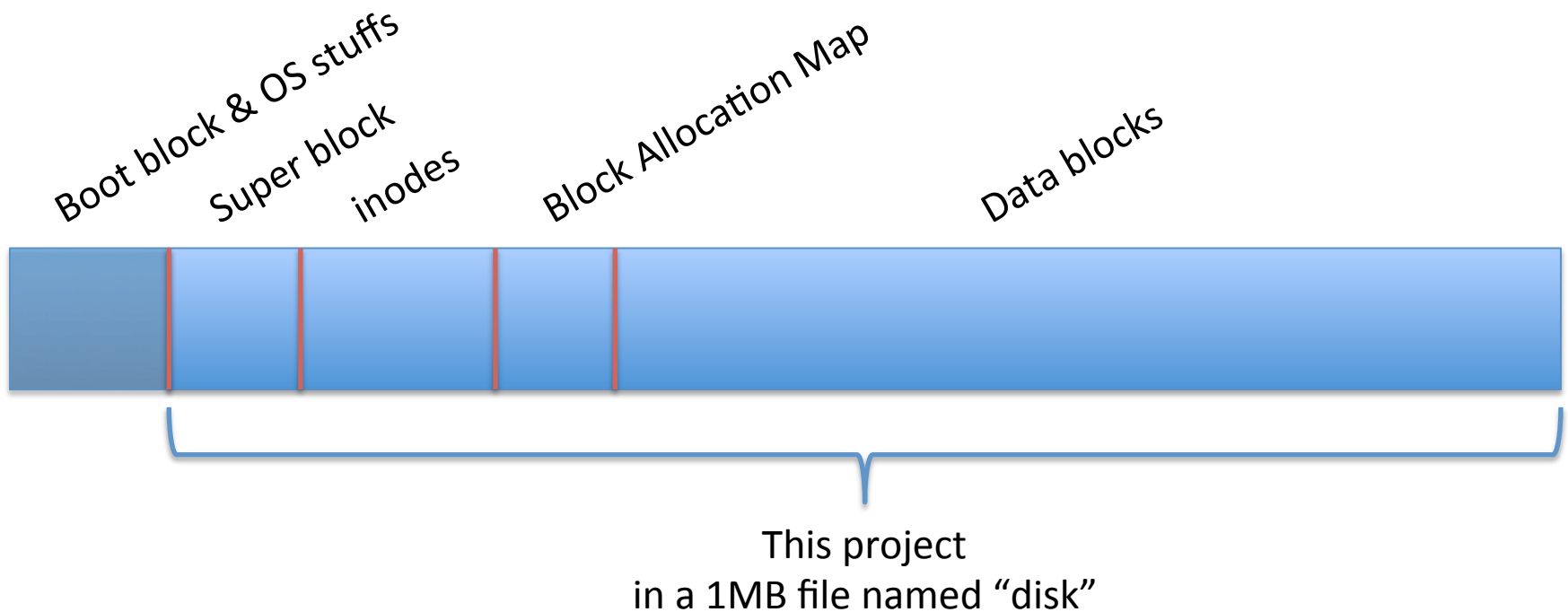
- Implement a simple UNIX-like file system with a hierarchical directory structure
- Manage disk space, as files and directories grow and shrink.
- Implement commands and system calls to browse the directory structure, create new files and directories, delete them, etc.
- Don't need to worry about concurrency, permissions, or high performance.

# API



- Disk format
- File
  - open, close, read, write, seek
  - link and unlink (delete a file)
  - stat
- Directory
  - make, remove, stat, etc.
- Shell commands
  - ls and chdir (cd)

# Disk Layout



Space between divisions is not representative of actual size.

# Superblock – Disk Metadata

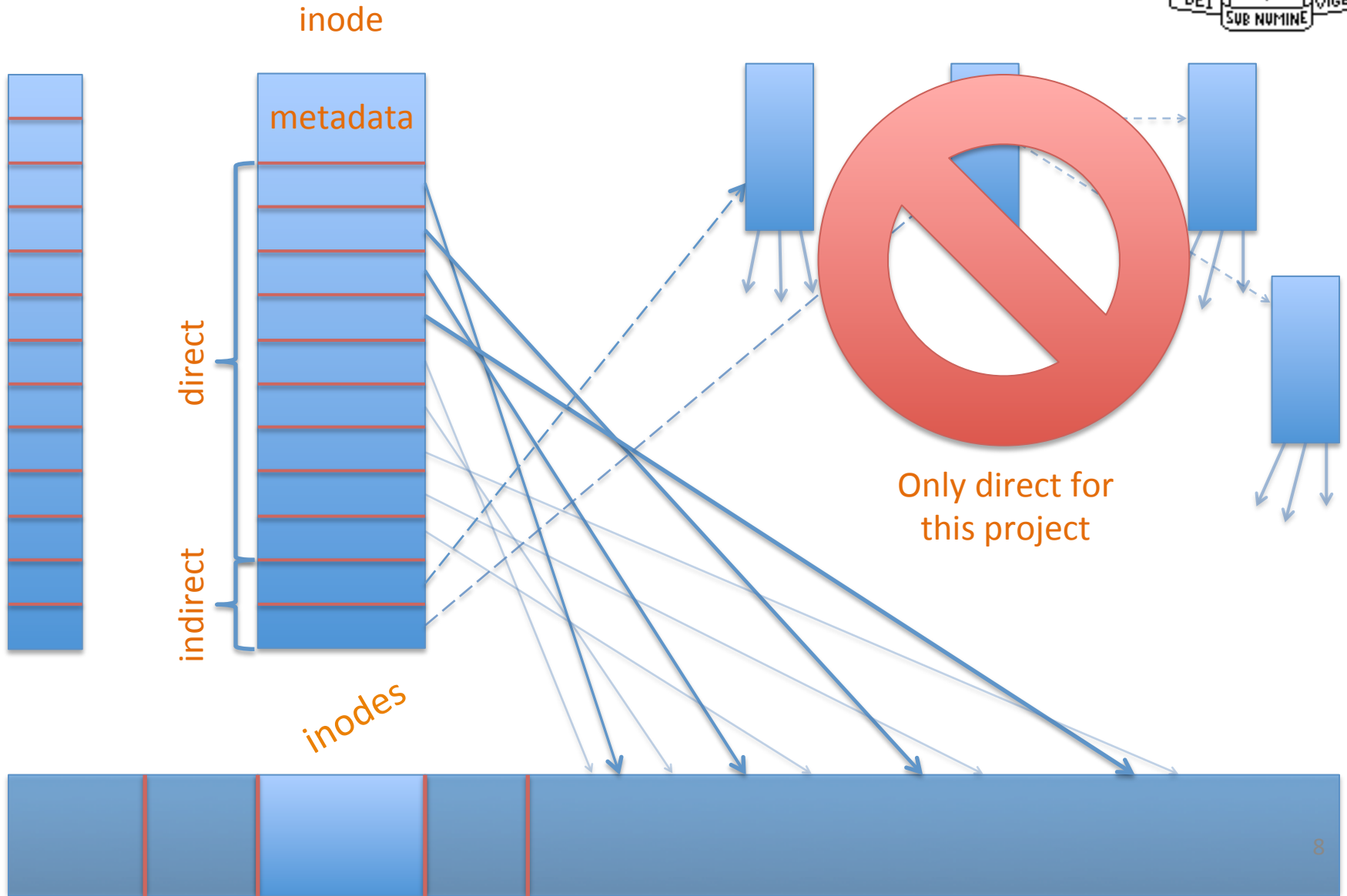


- Examples:
  - Size
  - # of inodes
  - # of data blocks
  - Where inodes start
  - Where data blocks start
  - Magic number

*Super block*



# Inodes





# Inode – Metadata



- Examples:
  - File or directory?
  - Link count
  - Size
  - Etc..

*inodes*



# fs\_init



- A “constructor” for the FS code.
- Call `block_init()` to initialize the block “device.”
- Initialize data structures and resources used by the file system.
- Format the disk or mount it if it is already formatted (create a mechanism to detect if the disk is formatted).

# fs\_mkfs



- “Makes” a file system:
  - Write the super block;
  - Mark inodes and data blocks as free;
  - Create root directory;
  - Initialize file descriptor table.

# File Creation and Deletion



- `fs_open()`, `fs_link()`, `fs_unlink()`.
- `open`: create a new file if it does not exist.
- `link`: hard link to a file
  - Create a link to an existing file
- `unlink`:
  - Delete a file if link count == 0;
  - Delete directory entry;
  - Special behavior if file is still open (look at the project description).

# File Access



- open: open an existing file (allocate file descriptor).
- read: read bytes from an open file.
- write: write bytes to an open file.
- lseek: change position in a file.
- close: free file descriptor.

# fs\_lseek() Semantics



- This project `fs_lseek()` takes only two arguments:
  - file descriptor and offset.
- Unix `lseek()` takes three arguments:
  - File descriptor, offset, whence.
- Whence: `SEEK_SET`, `SEEK_CUR`, `SEEK_END`.
- `ls_lseek()` assumes `SEEK_SET`.
- What if `lseek()` wants to seek past the end of the file? (look at the project description for the expected behavior)

# Directories – Part 1



- Like a file: list of files and directories:
  - Name to inode number mapping.
- Can read it like a file:
  - Use your file I/O functions (`fs_*`) to do directory manipulation.
- Always has at least two entries:
  - “.” current directory;
  - “..” parent directory.

# Directories – Part 2



- `mkdir`: make a directory.
  - create an entry in parent directory;
  - create two directories: “.” and “..”.
- `rmdir`: remove directory if empty.
- `cd`: change the current directory
  - for relative path names only.



# Example – fs\_mkdir



```
int fs_mkdir(char *file_name)
{
    if (file_name exists) return ERROR;
    // allocate inode
    // allocate data blocks
    // set directory entries for "." and ".."
    // set inode entries appropriately
    // update parent
    return SUCCESS
}
```

# Misc



- You don't need to support absolute path names.
- You don't need to support recursive directory removal.
- Implement a file system check (fsck) tool for debugging that verifies integrity of:
  - Superblock magic number;
  - Block allocations;
  - Inode allocations;
  - Block allocation map;
  - Directory content;
  - Etc.

# Implementation



- In Linux:
  - Uses a file to simulate a disk
  - Code is provided
  - Execute ./lnxsh
- Shell supports:
  - System calls for file system
  - Commands: “ls”, “cat foo”, “create foo 200”
- You will have to write a lot of code:
  - Over 1,000 lines of code.

# Testing



- A python script for testing is provided.
- Multiple tests that each:
  - execute the shell;
  - open an existing file system (or format a new one);
  - write commands to the shell (cat foo);
  - read output from the shell (ABCDEF);
  - exit.
- You should also write your own test cases:
  - submit them with your code.