CS 1550

Week 11

–

Project 4

Teaching Assistant

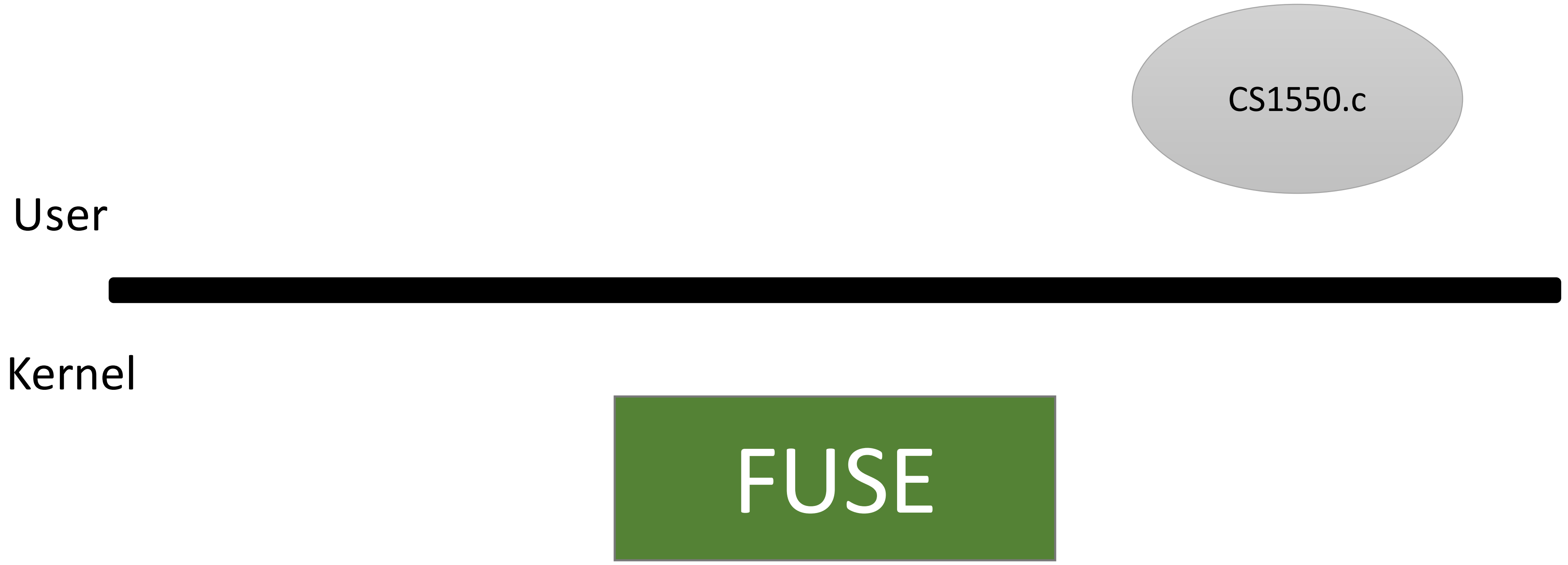
Henrique Potter

Overview

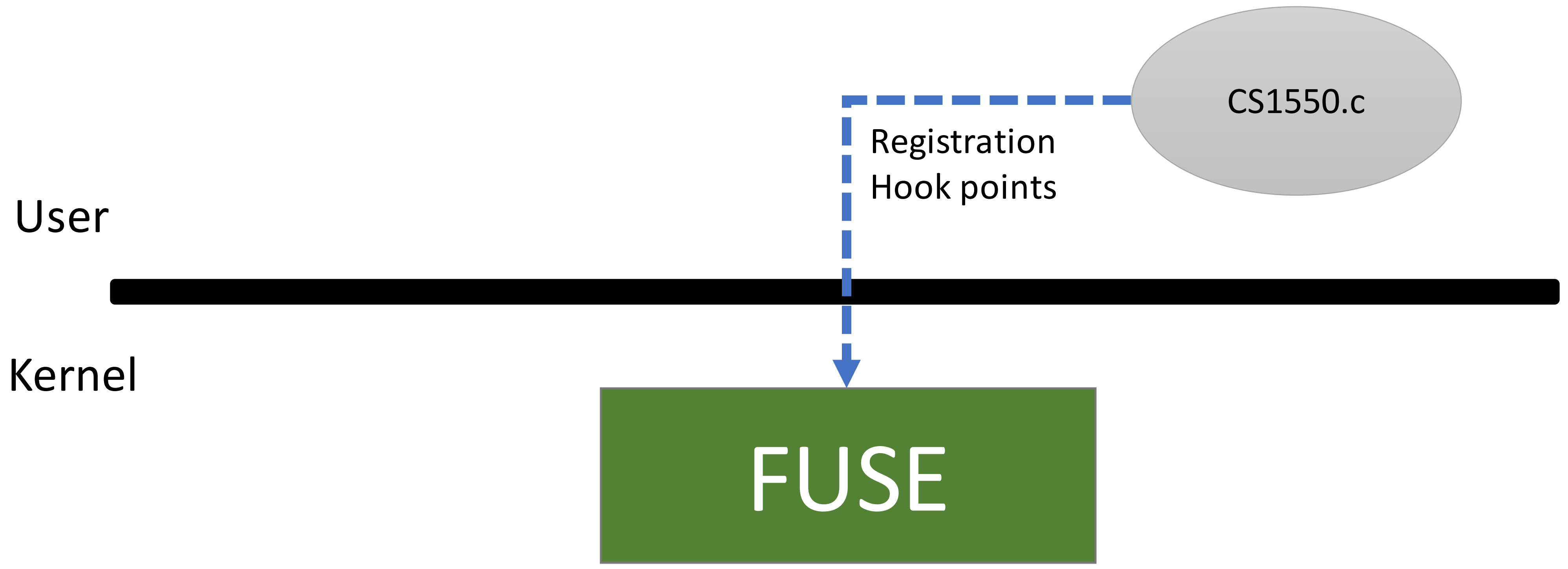
- FUSE is a **Linux kernel extension** that allows a user space program to provide the implementations for the various file-related syscalls

- Goal: Use FUSE to create our own file system

Overview



Overview

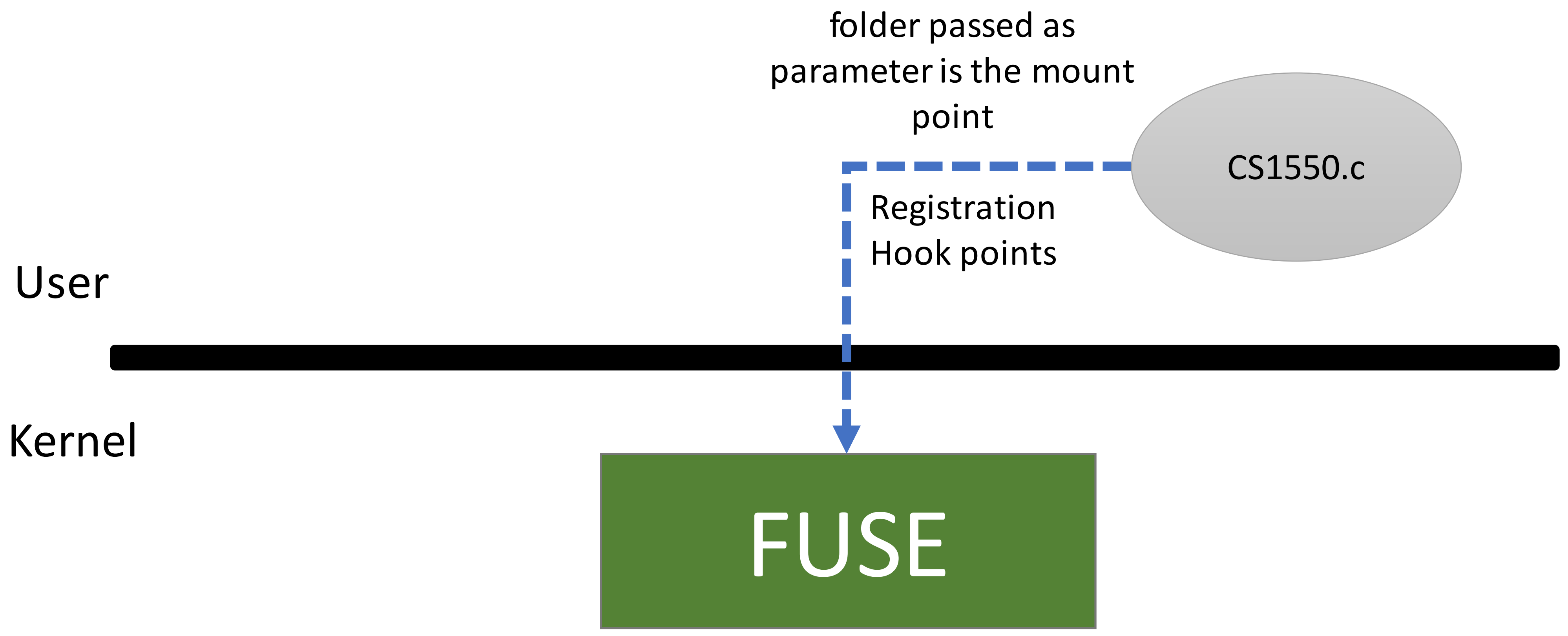


Overview

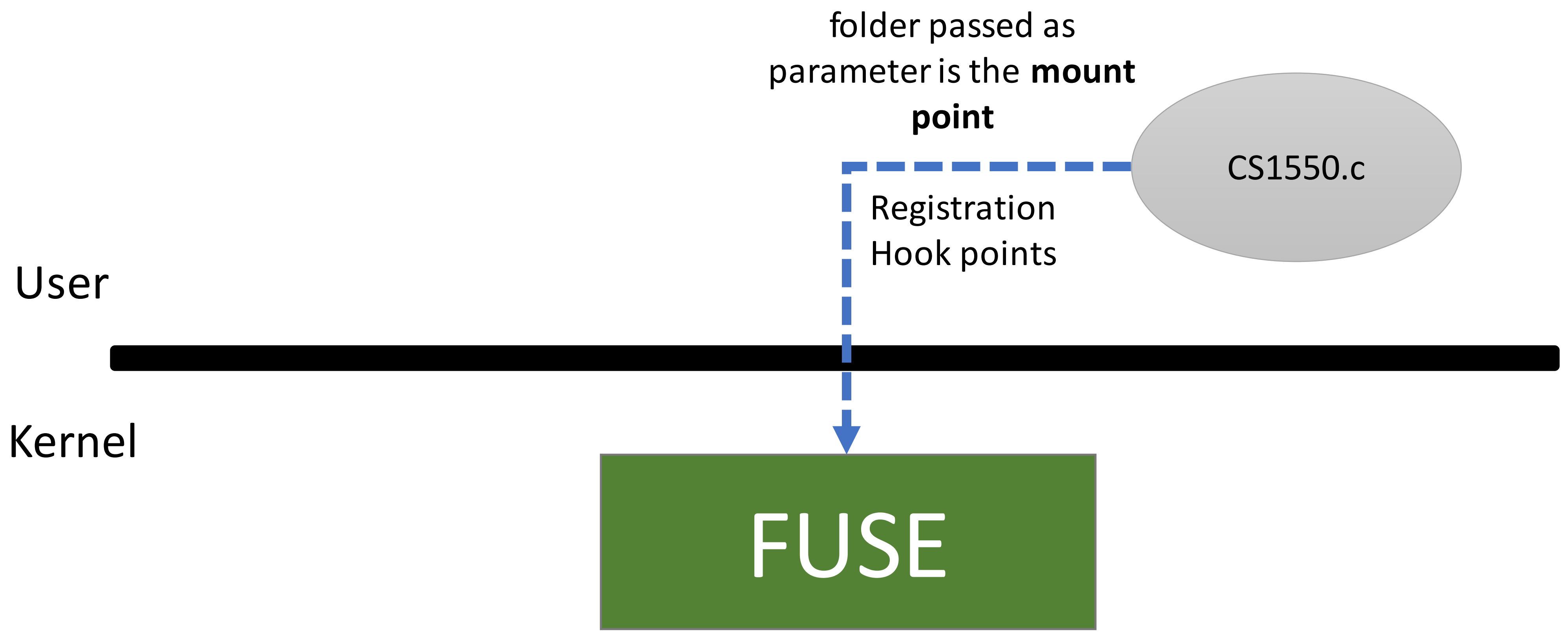
- In CS1550.c

```
static struct fuse_operations hello_oper = {
     .getattr  = hello_getattr,
     .readdir  = hello_readdir,
     .open     = hello_open,
     .read     = hello_read,
};
```

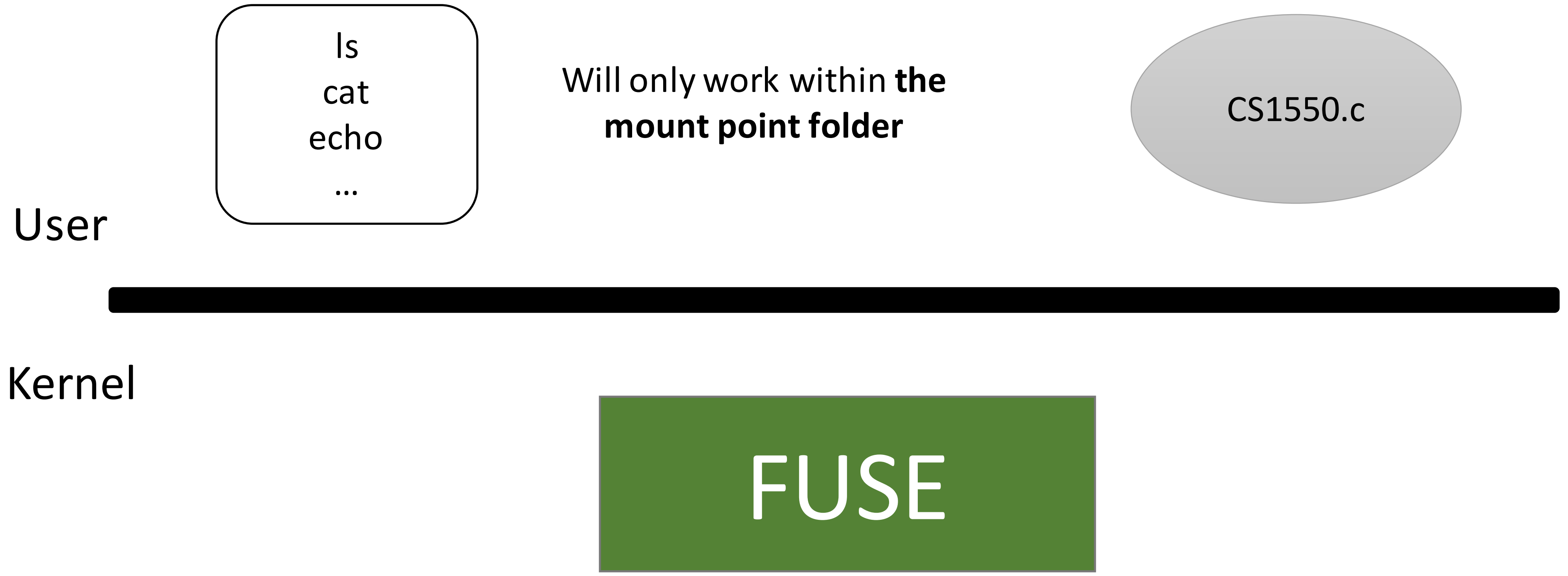
Overview


folder passed as parameter is the mount point
CS1550.c
Registration
Hook points
User
Kernel
FUSE

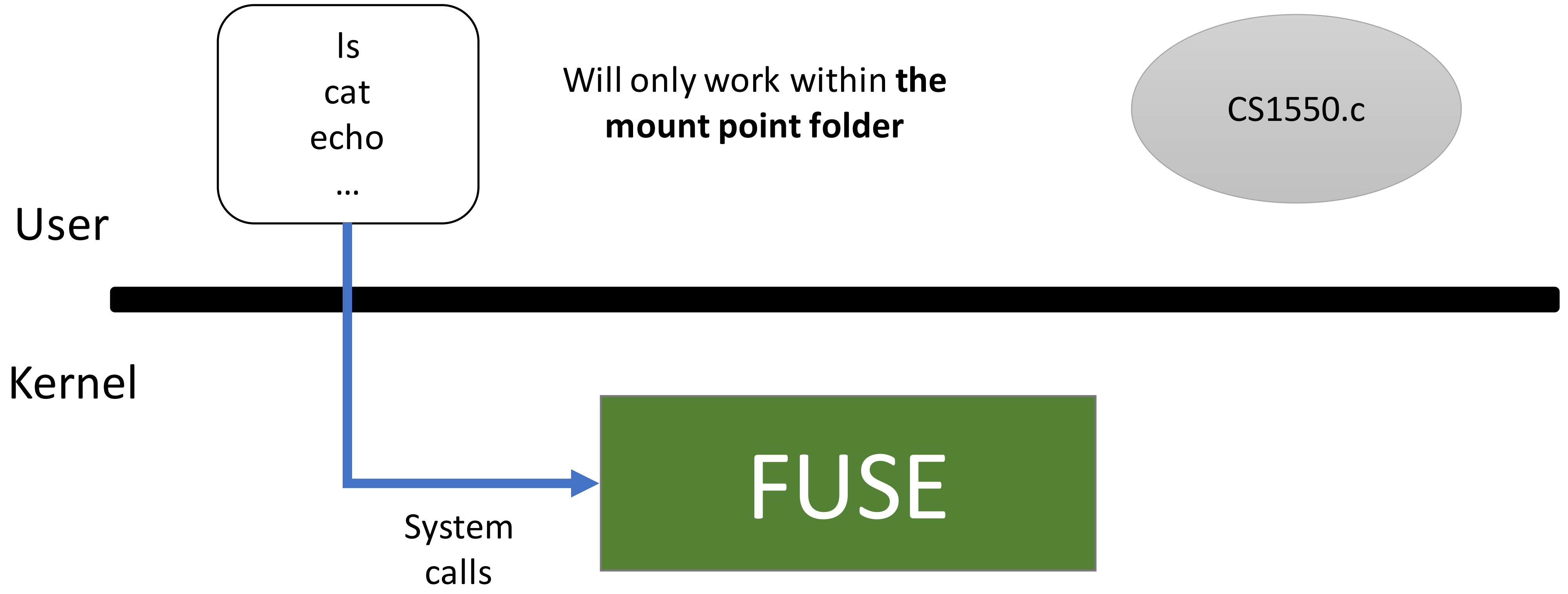
Overview


folder passed as parameter is the **mount point**
CS1550.c
Registration
Hook points
User
Kernel
FUSE

Overview

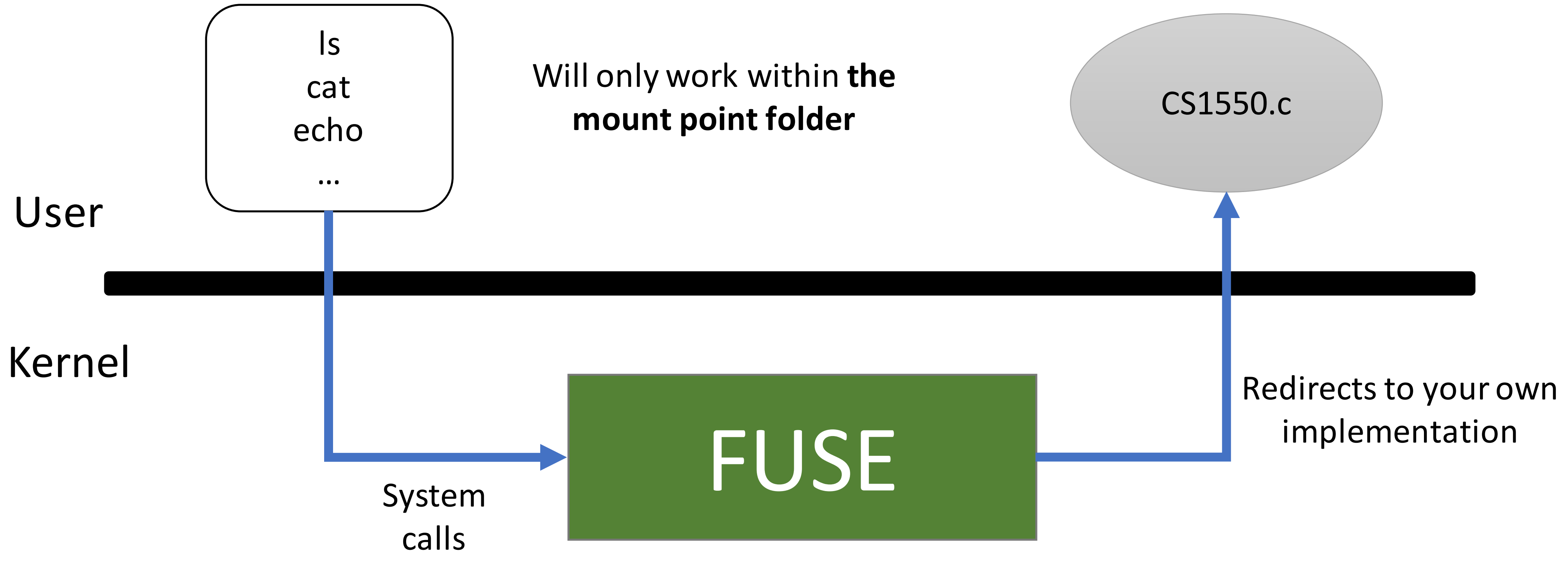

ls
cat
echo
…
Will only work within **the mount point folder**
CS1550.c
User
Kernel
FUSE

Overview



Will only work within **the mount point folder**

Overview


ls
cat
echo
…
Will only work within **the mount point folder**
CS1550.c
User
Kernel
FUSE
System calls
Redirects to your own implementation

Installation of FUSE

- Install libraries and example programs

```
cd /u/OSLab/USERNAME
cp /u/OSLab/original/fuse-2.7.0.tar.gz .
tar xvfz fuse-2.7.0.tar.gz
cd fuse-2.7.0
./configure
make
```

Installation of FUSE

- Install libraries and example programs

```
cd /u/OSLab/USERNAME
cp /u/OSLab/original/fuse-2.7.0.tar.gz .
tar xvfz fuse-2.7.0.tar.gz
cd fuse-2.7.0
./configure
make
```

This compiles the examples.

FUSE Example

```
cd /u/OSLab/USERNAME/
```

FUSE Example

```
cd /u/OSLab/USERNAME/
cd fuse-2.7.0/example
```

FUSE Example

```
cd /u/OSLab/USERNAME/
cd fuse-2.7.0/example
mkdir testmount (create mount point)
```

A mount point is a location in the UNIX hierarchical file system where a new device or file system is located

FUSE Example

```
cd /u/OSLab/USERNAME/
cd fuse-2.7.0/example
mkdir testmount (create mount point)
```

A mount point is a location in the UNIX hierarchical file system where a new device or file system is located

```
ls -al testmount
```

FUSE Example

```
cd /u/OSLab/USERNAME/
cd fuse-2.7.0/example
mkdir testmount (create mount point)
```

A mount point is a location in the UNIX hierarchical file system where a new device or file system is located

```
ls -al testmount
./hello testmount
```

FUSE Example

```
cd /u/OSLab/USERNAME/
cd fuse-2.7.0/example
mkdir testmount (create mount point)
```

A mount point is a location in the UNIX hierarchical file system where a new device or file system is located

```
ls -al testmount
./hello testmount
ls -al testmount
```

FUSE Example

cd /u/OSLab/USERNAME/

cd fuse-2.7.0/example

mkdir testmount (create mount point)

> A mount point is a location in the UNIX hierarchical file system where a new device or file system is located

ls -al testmount

./hello testmount

ls -al testmount

Should see . , .., hello

FUSE Example

- fusermount –u testmount

  Unmount the file system we just used when we are done, or need to make changes to the program.

  Always need to do unmount!

Setting up the Environment Variables

- cd ~
- chmod u+w .bash_profile
  Gives you the write permission to .bash_profile
- nano .bash_profile
- Scroll down to the end of the file until you see the line:
  “# Define your own private shell functions and other commends here”
- Add the following lines (spacing around ‘[‘ and ‘]’ characters need to be there!)
```
if [ “$HOSTNAME” = “thoth.cs.pitt.edu” ]; then
source /opt/set_specific_profile.sh;
fi
```
- Save the file and quit
- chmod u-w .bash_profile
- .bash_profile will not run until the next time you log in
- source /opt/set_specific_profile.sh

Debug Mode

- Testing is to launch a FUSE application with the –d option

  ```
  ./hello –d testmount
  ```

  This will keep the program in the foreground, and it will print out every message that the application receives, and interpret the return values that you're getting back.
- Open a second terminal window and try your testing procedures.
- If you do a CTRL + C in the first window, you may not need to unmount the file system.
- IMPORTANT: if your program crashes or you abort it, you definitely need to do the fusermount. Otherwise, you will get a confusing "Transport endpoint not connected" message the next time you try to mount the system.

FUSE Example cont.

cat testmount/hello

- Hello world

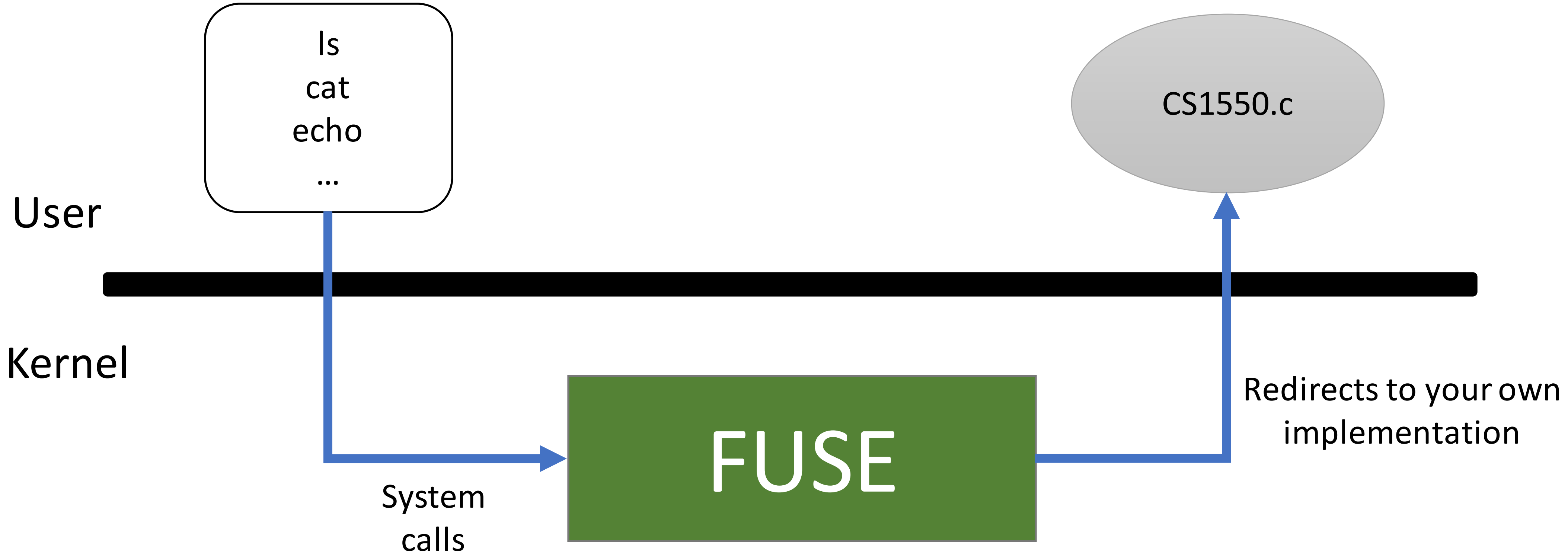
- If we cat a file that doesn’t really exist, how do we get meaningful output?

FUSE Example cont.

cat testmount/hello

- Hello world

- If we cat a file that doesn't really exist, how do we get meaningful output?



FUSE Example cont.

```
static int hello_read(const char *path, char *buf, size_t
size, off_t offset, struct fuse_file_info *fi)
{
    ...
}
```

FUSE Example cont.

```
static int hello_read(const char *path, char *buf, size_t
size, off_t offset, struct fuse_file_info *fi)
{
    ...
    if (offset < len) {
       …
       memcpy(buf, hello_str + offset, size);
    } else
       size = 0;
    return size;
}
```

FUSE Example cont.

- Unmount the file system

```
fusermount -u testmount
```

What You Need To Do

- Create the **cs1550 file system** as a **FUSE application**

What You Need To Do

- Create the cs1550 file system as a FUSE application
- A code skeleton has been provided **under the FUSE zip examples** directory as cs1550.c

What You Need To Do

- Create the cs1550 file system as a FUSE application
- A code skeleton has been provided under the FUSE zip examples directory as cs1550.c
- **Automatically built** when make

What You Need To Do

- Create the cs1550 file system as a FUSE application
- A code skeleton has been provided under the FUSE zip examples directory as cs1550.c
- Automatically built when make
- Implement **using a single file**, named **.disk 512-byte blocks**

CS 1550

Week 12

–

Project 4

Teaching Assistant

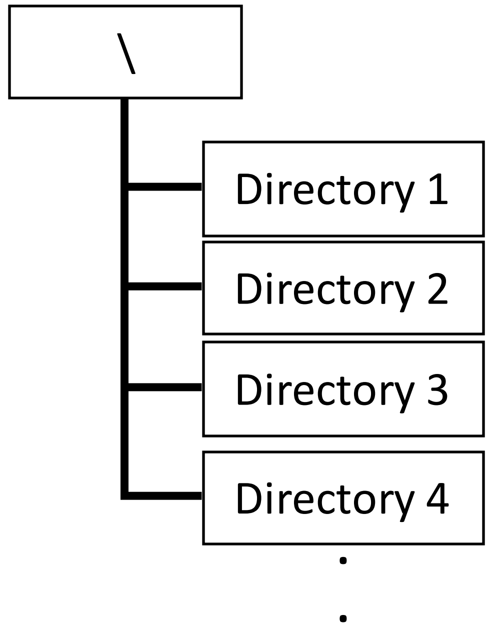
Henrique Potter

File System

- Two-level directory system
  - The root directory “\” will only contain other subdirectories, and no regular files.

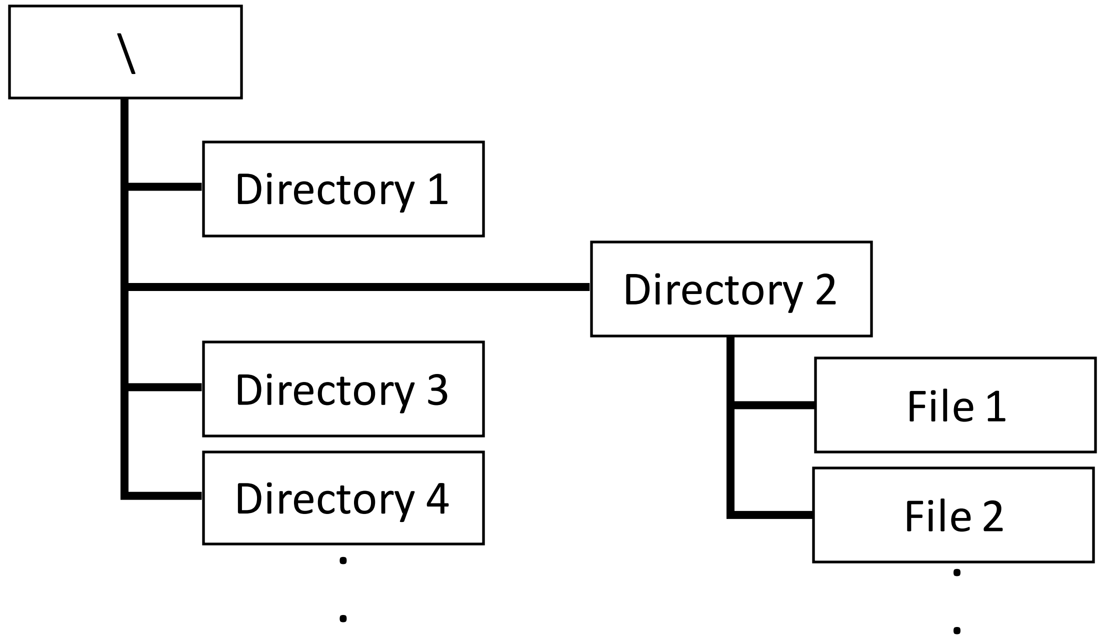
File System

- Two-level directory system
  - The root directory “\” **will only contain other subdirectories**, and no regular files.



File System

- Two-level directory system
  - The root directory “\” will only contain other subdirectories, and no regular files.
  - The subdirectories **will only contain regular files**, and no subdirectories of their own.



File System

- Two-level directory system
    - The root directory “\” will only contain other subdirectories, and no regular files.
    - The subdirectories will only contain regular files, and no subdirectories of their own.
    - All files **will be full access** with permissions to be mainly ignored.

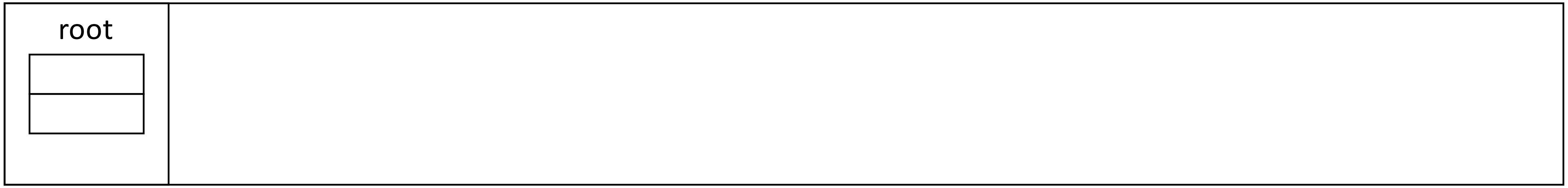
File System

- Two-level directory system
  - The root directory “\” will only contain other subdirectories, and no regular files.
  - The subdirectories will only contain regular files, and no subdirectories of their own.
  - All files will be full access with permissions to be mainly ignored.
  - Many file attributes such as creation and modification times **will not be accurately stored**.

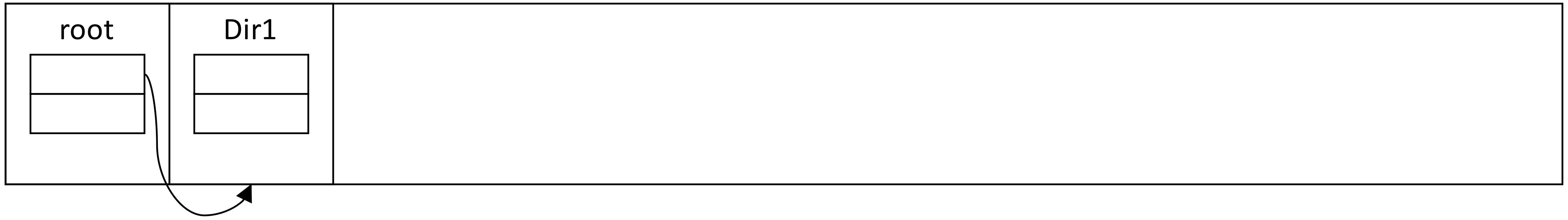
File System

- Two-level directory system
  - The root directory “\” will only contain other subdirectories, and no regular files.
  - The subdirectories will only contain regular files, and no subdirectories of their own.
  - All files will be full access with permissions to be mainly ignored.
  - Many file attributes such as creation and modification times will not be accurately stored.
  - **Files cannot be truncated**.

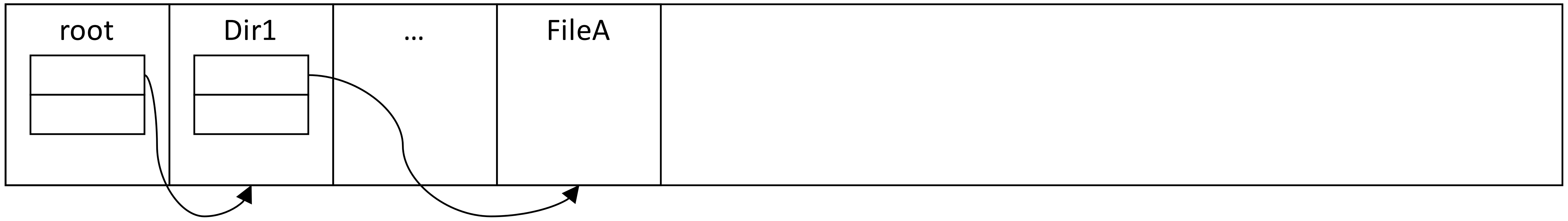
Structure



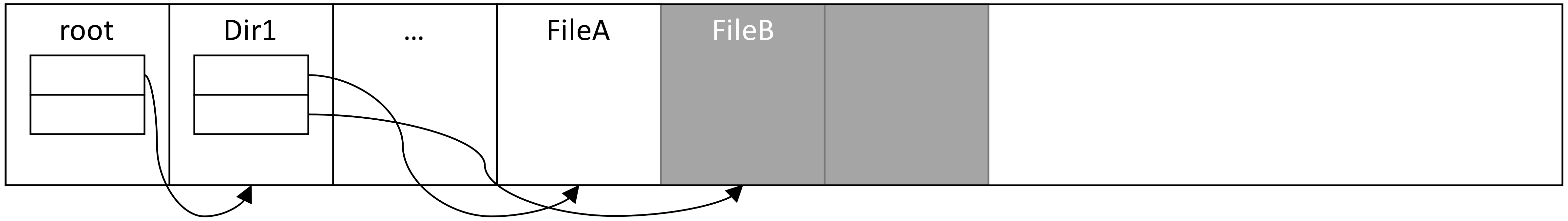
Structure



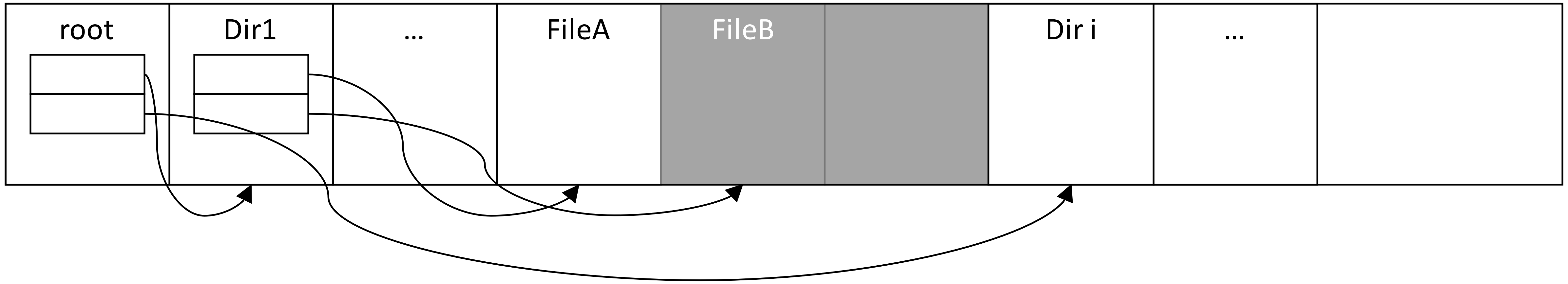
Structure



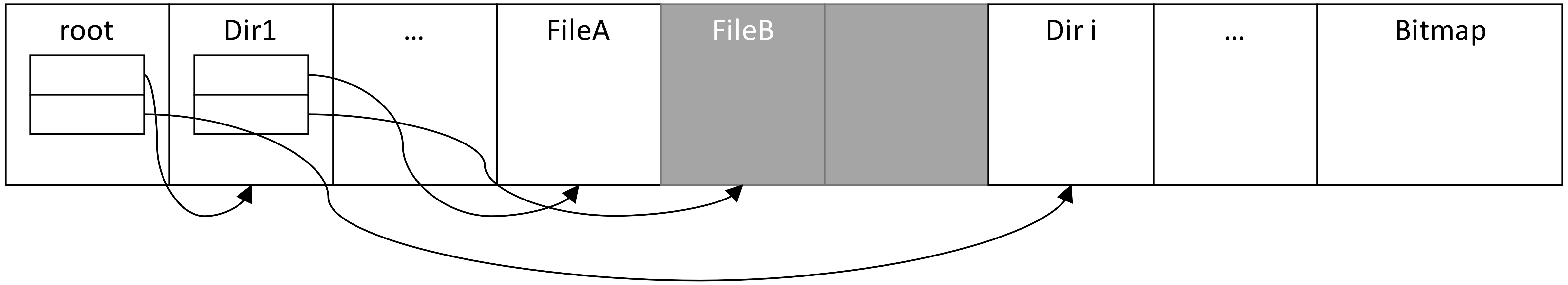
Structure



Structure

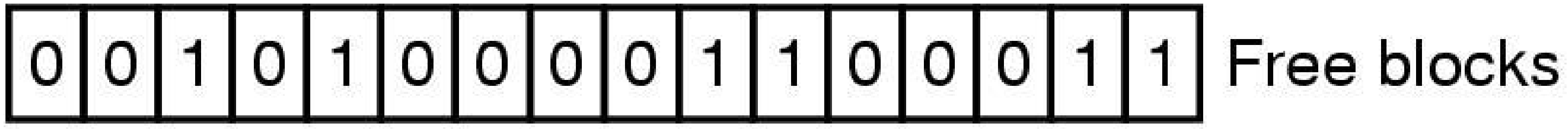


Structure



Disk Management

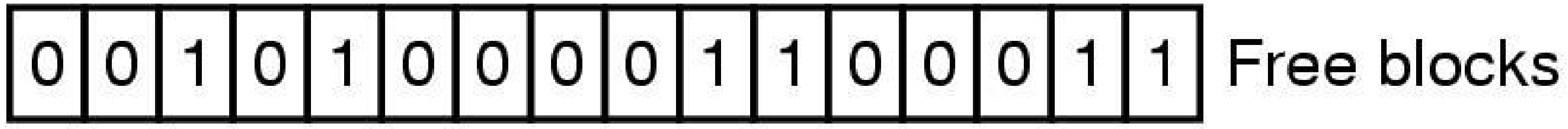
- Manage free (or empty) space using **bitmap**



(a)

Disk Management

- Manage free (or empty) space using bitmap

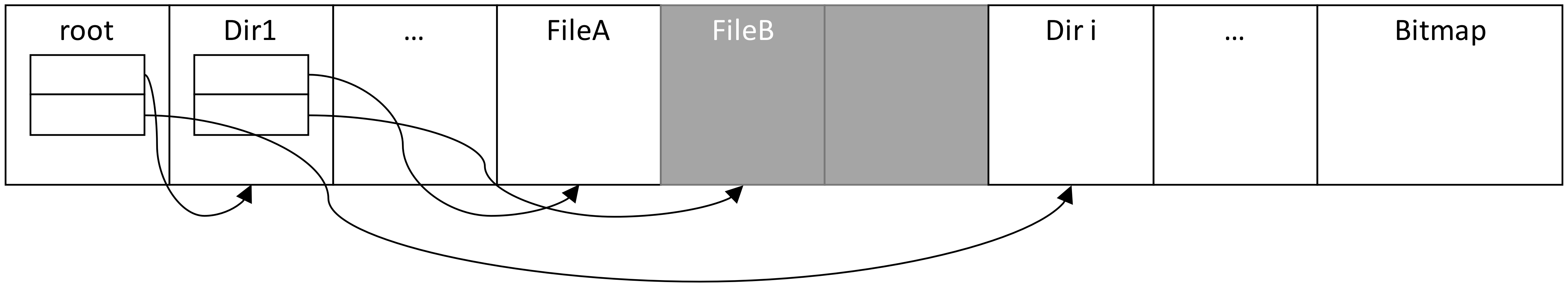


(a)

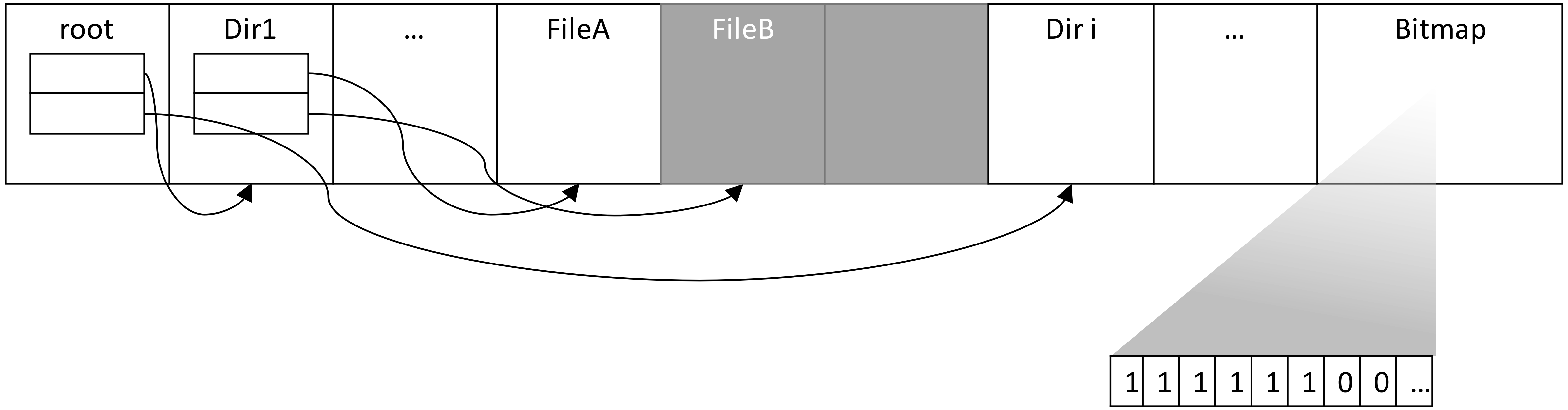
- **Create** a 5MB disk image

```
dd bs=1K count=5K if=/dev/zero of=.disk
```

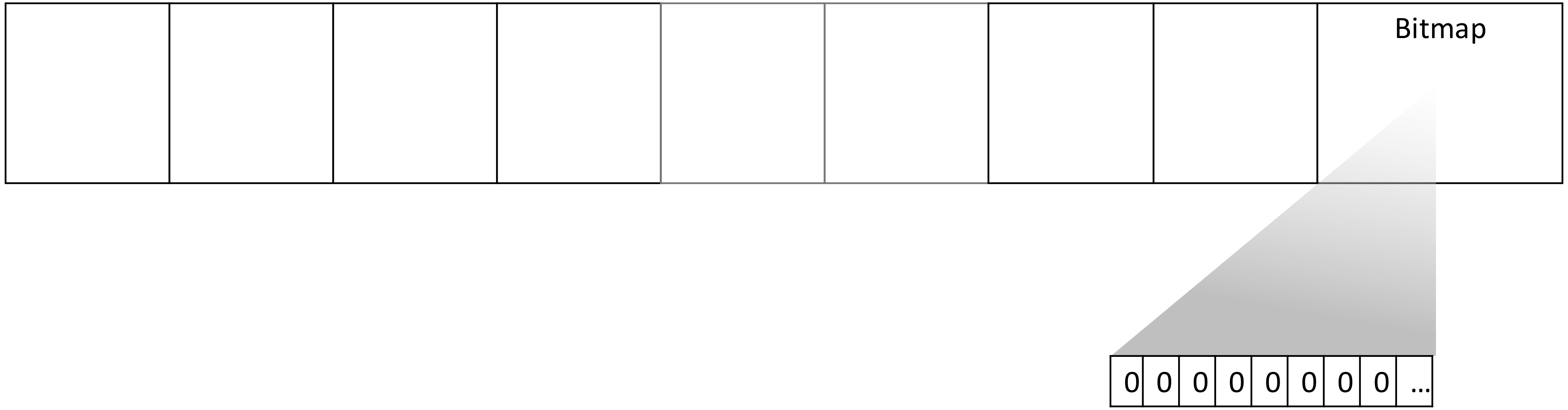
Disk Management



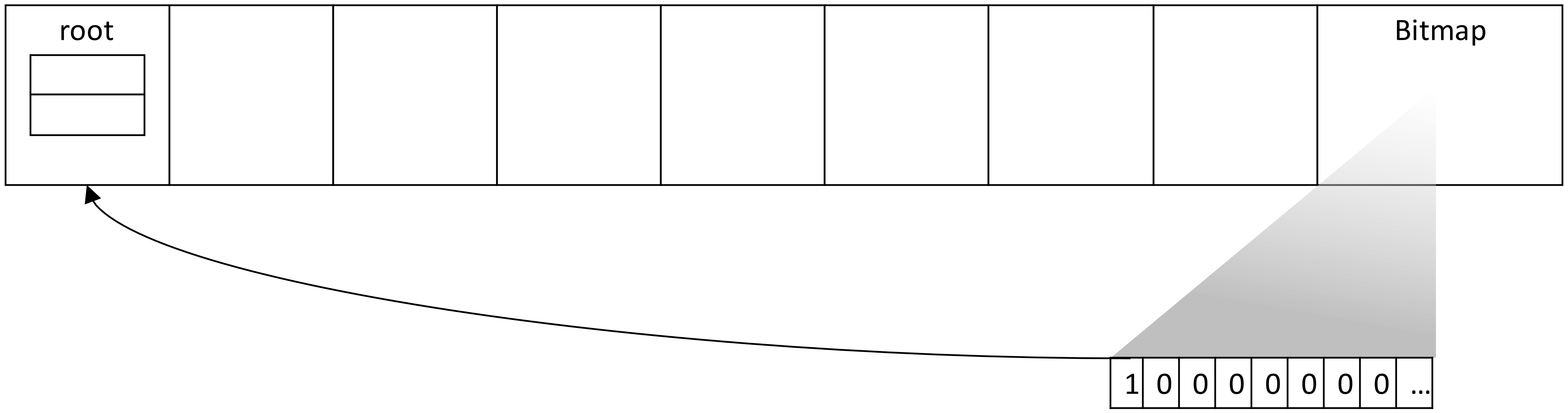
Disk Management



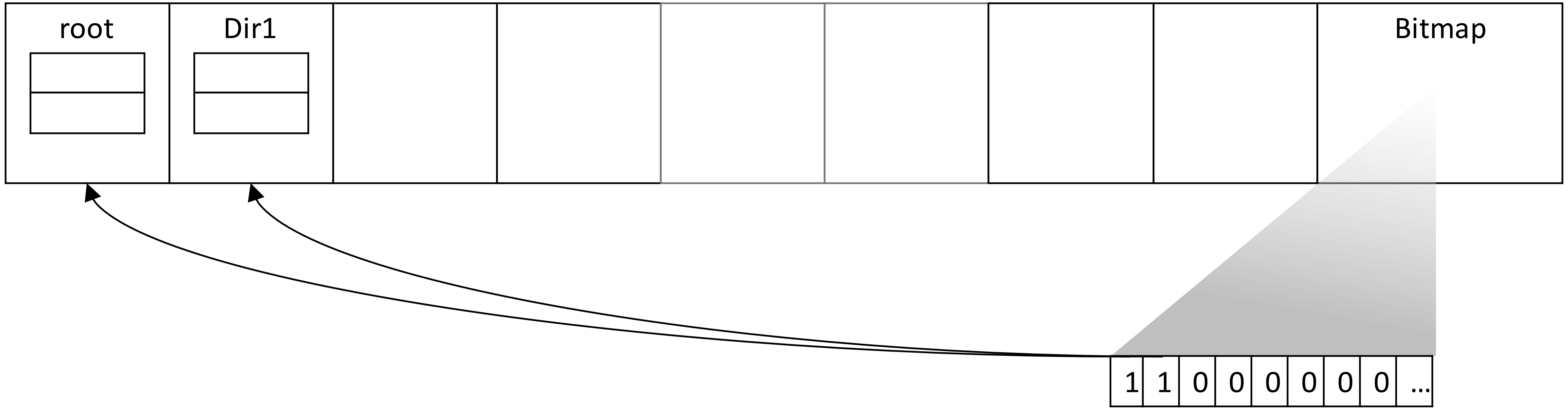
Disk Management



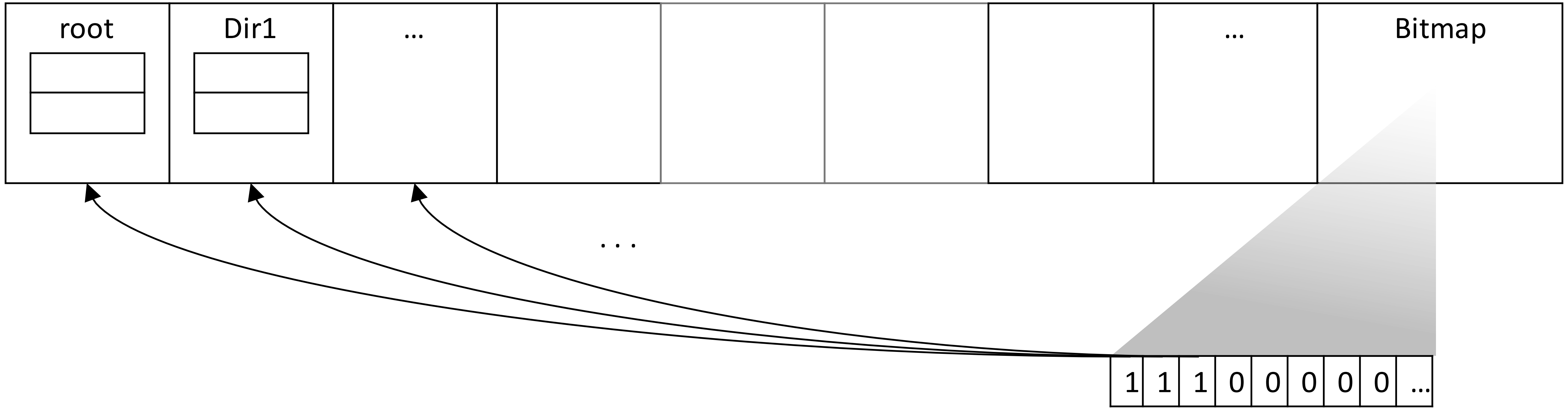
Disk Management



Disk Management



Disk Management



Root Directory

```
struct cs1550_root_directory {
	int nDirectories;	//How many subdirectories are in the root
				//Needs to be less than MAX_DIRS_IN_ROOT
	struct cs1550_directory
	{
		char dname[MAX_FILENAME + 1]; //directory name (plus 
space for nul)
		long nStartBlock; //where the directory block is on disk
	} directories[MAX_DIRS_IN_ROOT]; //There is an array of these
};
```

Root Directory

```
struct cs1550_root_directory {
	int nDirectories;	//How many subdirectories are in the root
				//Needs to be less than MAX_DIRS_IN_ROOT
	struct cs1550_directory
	{
		char dname[MAX_FILENAME + 1]; //directory name (plus
space for nul)
		long nStartBlock; //where the directory block is on disk
	} directories[MAX_DIRS_IN_ROOT]; //There is an array of these
};
```

Subdirectories

```
struct cs1550_directory_entry
{
        int nFiles;         //How many files are in this directory.
                            //Needs to be less than MAX_FILES_IN_DIR
        struct cs1550_file_directory
        {
                char fname[MAX_FILENAME + 1];       //filename (plus space for nul)
                char fext[MAX_EXTENSION + 1];       //extension (plus space for nul)
                size_t fsize;                       //file size
                long nStartBlock;                   //where the first block is on disk
        } files[MAX_FILES_IN_DIR];                  //There is an array of these
};
```

Subdirectories

```
struct cs1550_directory_entry
{
        int nFiles;         //How many files are in this directory.
                            //Needs to be less than MAX_FILES_IN_DIR
        struct cs1550_file_directory
        {
                char fname[MAX_FILENAME + 1];      //filename (plus space for nul)
                char fext[MAX_EXTENSION + 1];      //extension (plus space for nul)
                size_t fsize;                      //file size
                long nStartBlock;                  //where the first block is on disk
        } files[MAX_FILES_IN_DIR];                 //There is an array of these
};
```

Files

```
struct cs1550_disk_block {
	//All the space in the block can be used for actual data
	//storage.
	char data[MAX_DATA_IN_BLOCK];
};
```

Syscalls

- **cs1550_getattr**
- **cs1550_mkdir**
- cs1550_readdir
- cs1550_rmdir
- **cs1550_mknod**
- cs1550_write
- cs1550_read
- cs1550_unlink
- cs1550_truncate
- cs1550_open
- cs1550_flush

Syscalls

- **cs1550_getattr**
- **cs1550_mkdir**
- cs1550_readdir
- cs1550_rmdir
- **cs1550_mknod**
- cs1550_write
- cs1550_read
- cs1550_unlink
- cs1550_truncate
- cs1550_open
- cs1550_flush

No delete calls need to be written so
you don’t need to solve fragmentation

When there is no space left, return an error