CS 1550

Week 14

–

Project 4

Teaching Assistant

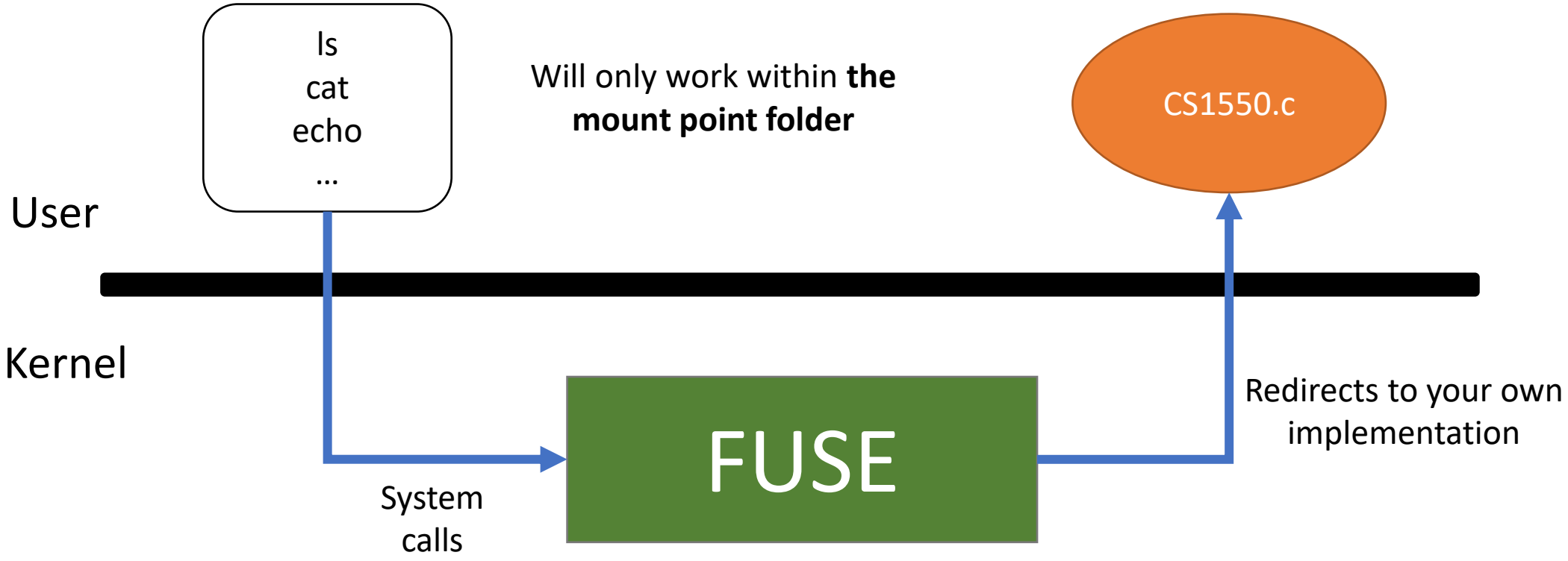
Henrique Potter

Overview

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

- Goal: Use FUSE to create our own file system

Overview: User Space File System



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

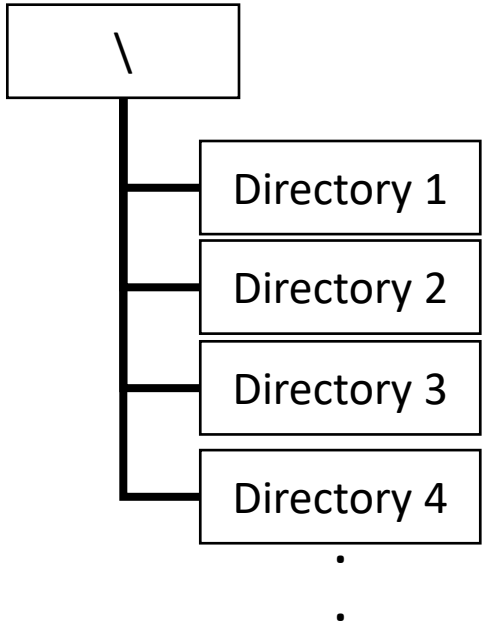
- Create the cs1550 file system as a FUSE application
- A code skeleton has been provided under the FUSE zip examples directory as cs1550.c
- Automatically built when make
- Implement **using a single file**, named **.disk 512-byte blocks**

File System

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

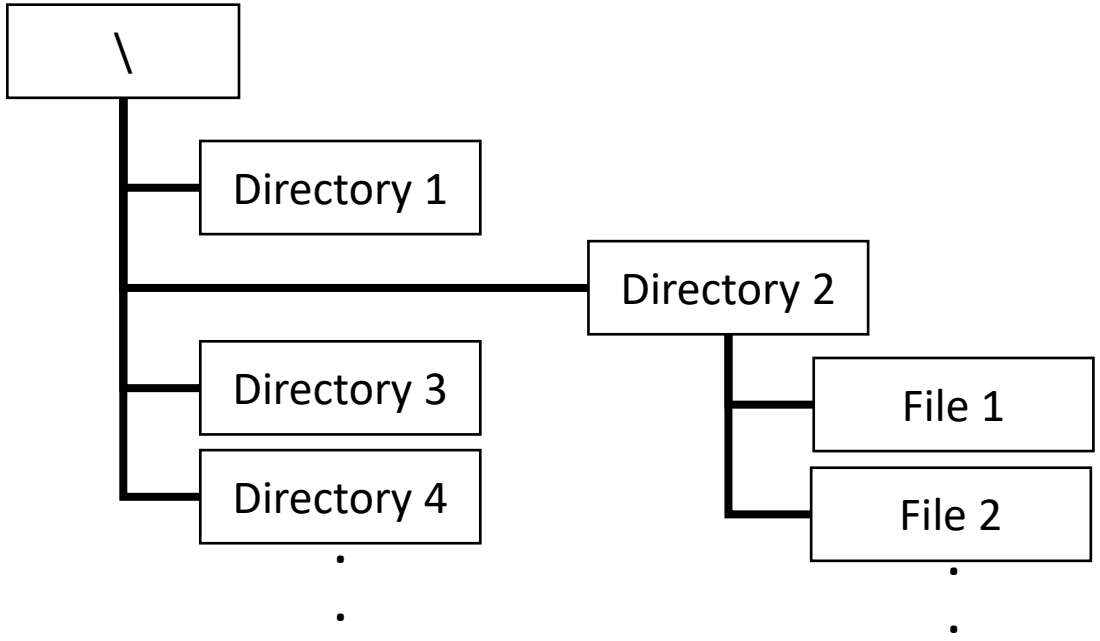
File System

- 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**.

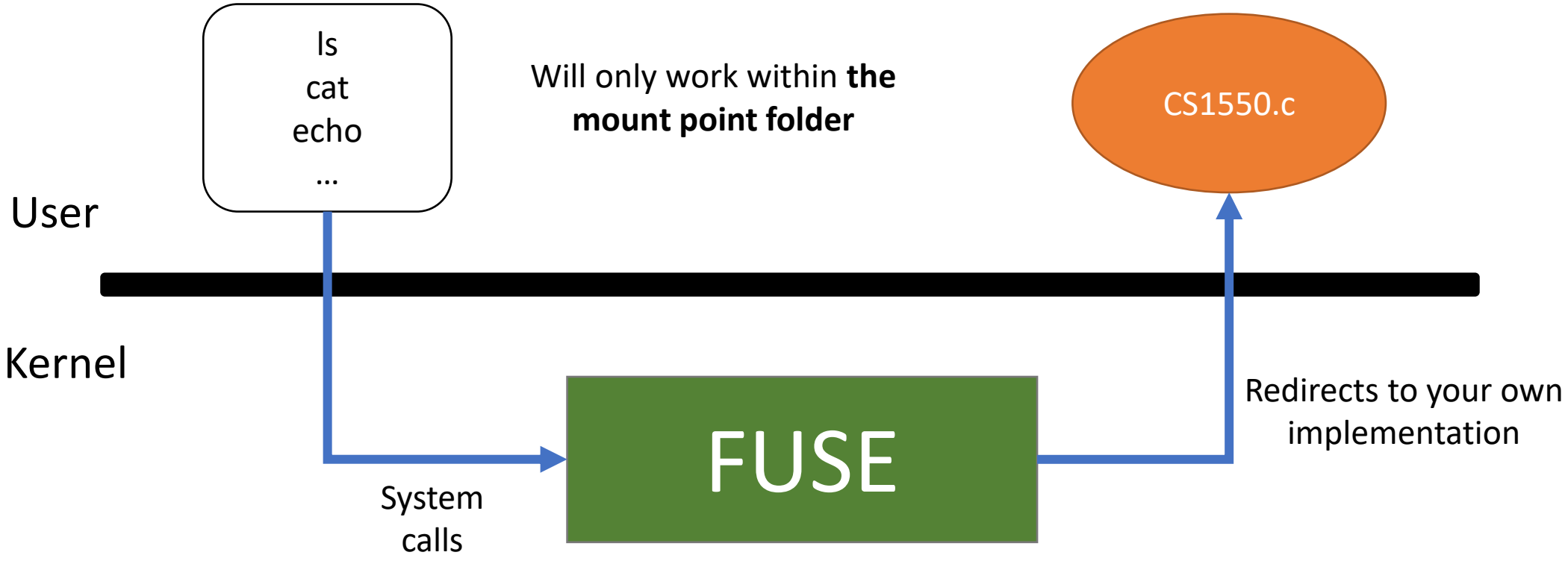
File System

- 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.

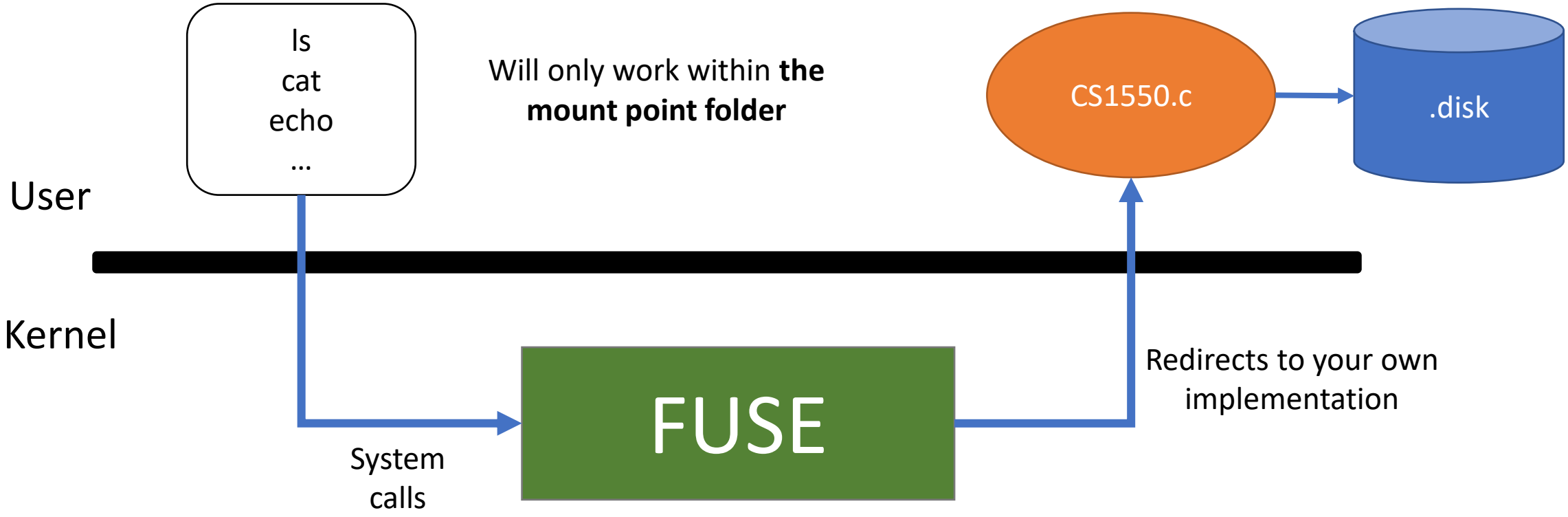
File System

- 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.
  - **The directory and file locations will be indexed.**

Overview: User Space File System



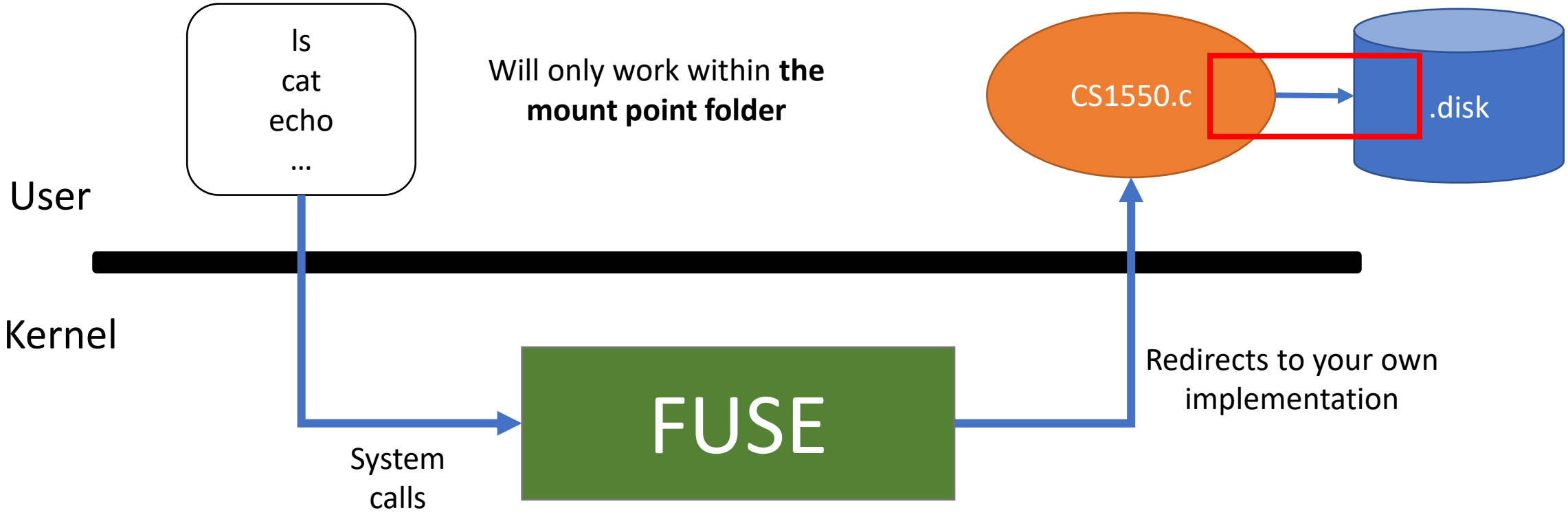
Overview: User Space File System



Overview: User Space File System

- CS1550.c
  - Implements functions to intercept file system calls allowing us to create our own implementation for each call
  - Runs in background
    - Using “./CS1550 –d testmount” to run in front
  - Communicate with .disk file to read/write file related data
- .Disk
  - Virtual Disk where data is actually kept
  - Consists of 512 bytes blocks

Access the .disk with CS1550.c



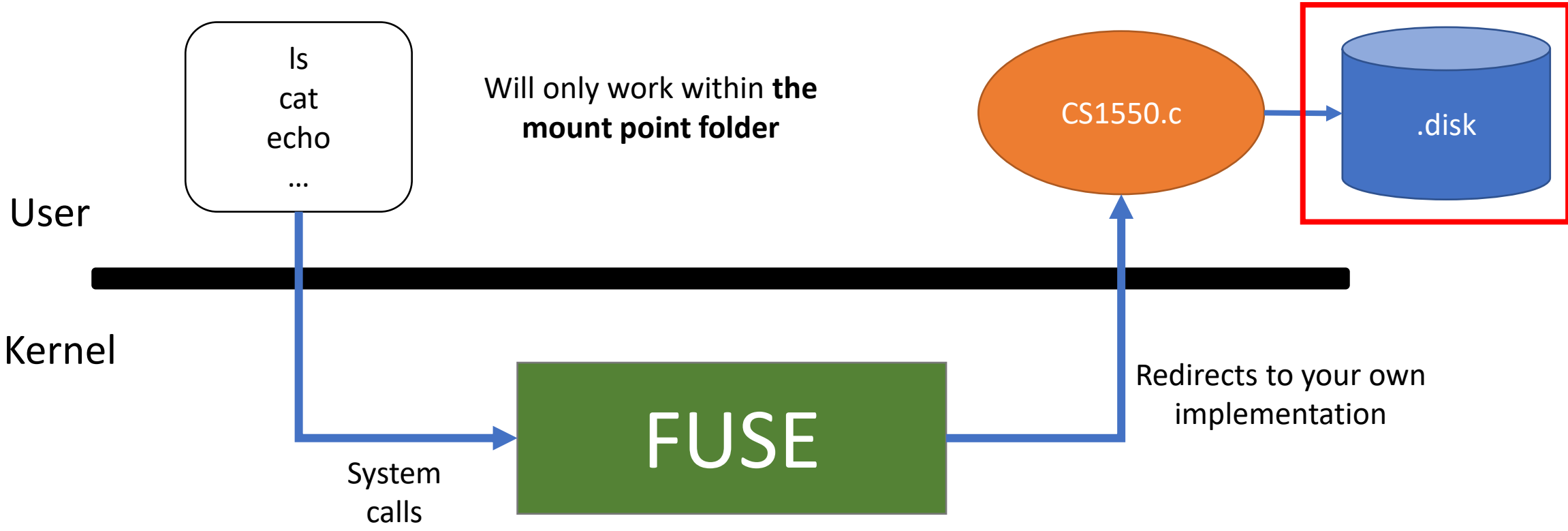
Access the .disk with CS1550.c

- CS1550.c will read data from .disk
  - Take system call requests
  - Decide location of the file
  - Using default system call (**fopen**/**fseek**/**fread**) to retrieve data needed
- Options to link to .disk
  - Open and read on-demand
    - Each time open the file and read then close
  - Open in cs1550_init()
    - Keeping file descriptor in a global variable
    - Close in cs1550_destory

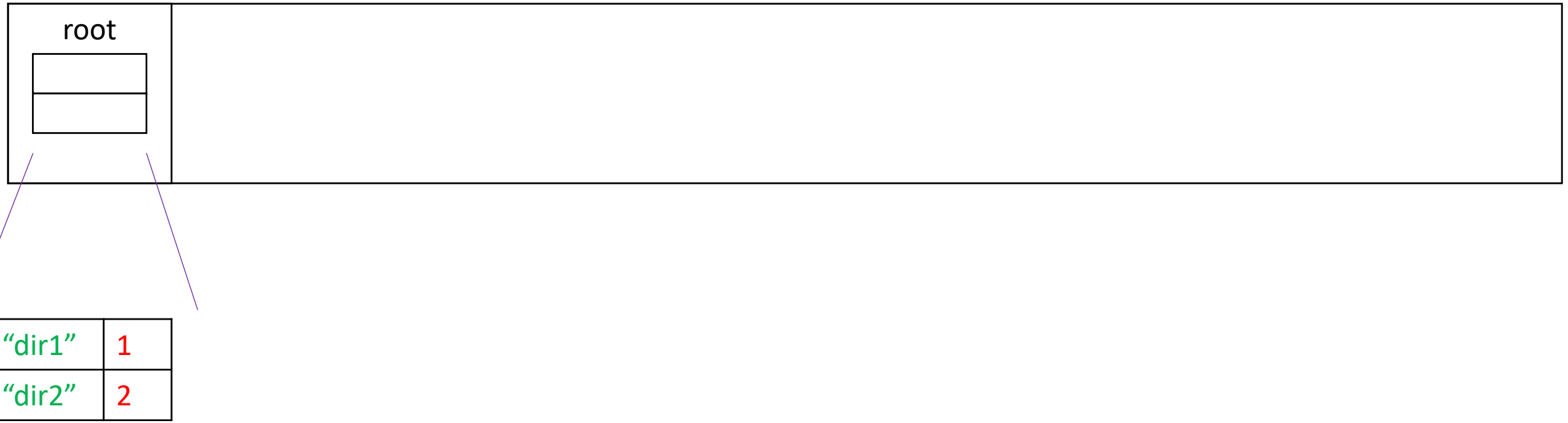
Access the .disk with CS1550.c

- CS1550.c will read data from .disk
  - Take system call requests
  - Decide location of the file
  - Using default system call (**fopen**/**fseek**/**fread**) to retrieve data needed

File System Structure



Structure



Structure: 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
  } __attribute__((packed)) directories[MAX_DIRS_IN_ROOT];  //There is an array of these

  //This is some space to get this to be exactly the size of the disk block.
  //Don't use it for anything.
  char padding[BLOCK_SIZE - MAX_DIRS_IN_ROOT * sizeof(struct cs1550_directory) - sizeof(int)];
} ;
```

Structure: 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
  } __attribute__((packed)) directories[MAX_DIRS_IN_ROOT];  //There is an array of these

  //This is some space to get this to be exactly the size of the disk block.
  //Don't use it for anything.
  char padding[BLOCK_SIZE - MAX_DIRS_IN_ROOT * sizeof(struct cs1550_directory) - sizeof(int)];
} ;
```

Number of subdir

Structure: 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
  } __attribute__((packed)) directories[MAX_DIRS_IN_ROOT];  //There is an array of these

  //This is some space to get this to be exactly the size of the disk block.
  //Don't use it for anything.
  char padding[BLOCK_SIZE - MAX_DIRS_IN_ROOT * sizeof(struct cs1550_directory) - sizeof(int)];
} ;
```

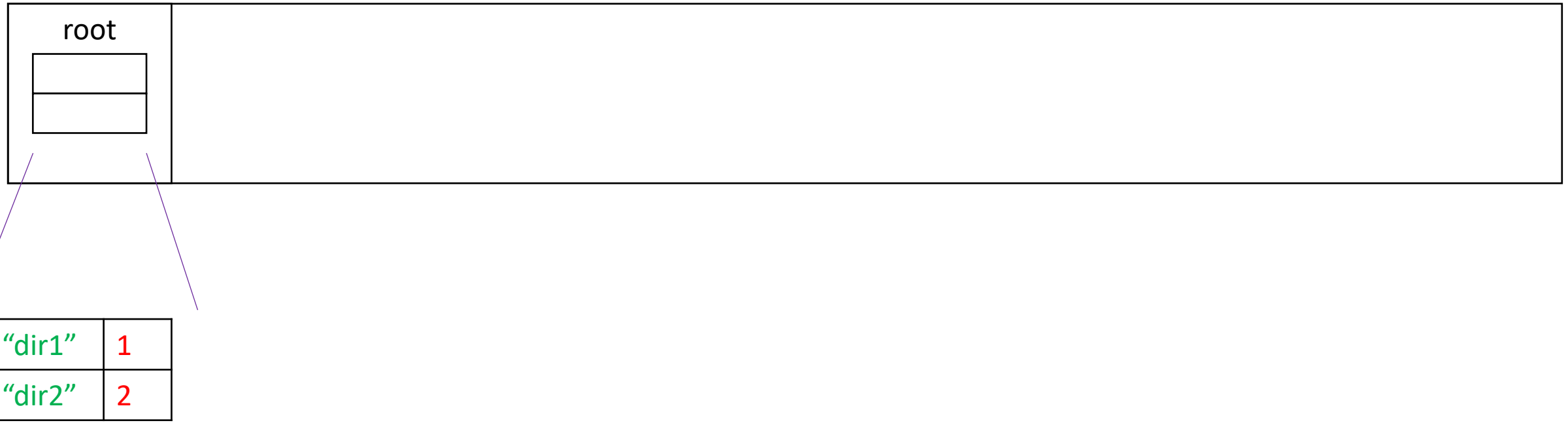
Subdirs: name, addr

Structure: Root Directory

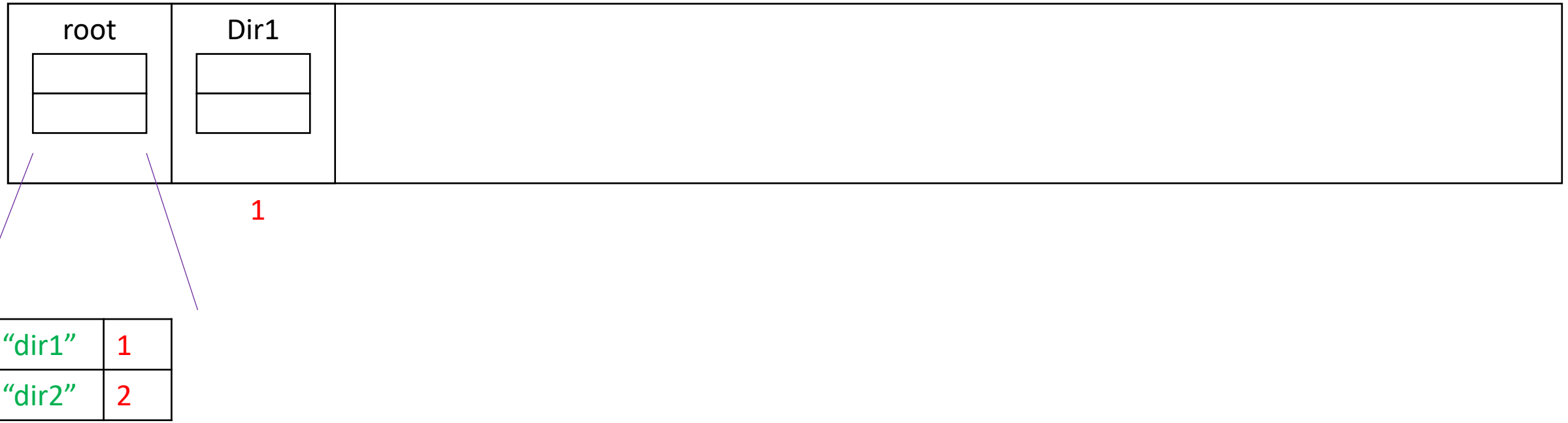
```
struct cs1550_root_directory
{
  int nDirectories; //How many subdirectories are in the root
            //Needs to be less than MAX_DIRS_IN_ROOT
  struct cs1550_directory
  {
    char dname[MAX_FILENAME + 1]; //directory name (plus space for nul)
    long nStartBlock;       //where the directory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

  //This is some space to get this to be exactly the size of the disk block.
  //Don't use it for anything.
  char padding[BLOCK_SIZE - MAX_DIRS_IN_ROOT * sizeof(struct cs1550_directory) - sizeof(int)];
} ;
```

Padding for 512 bytes blocks

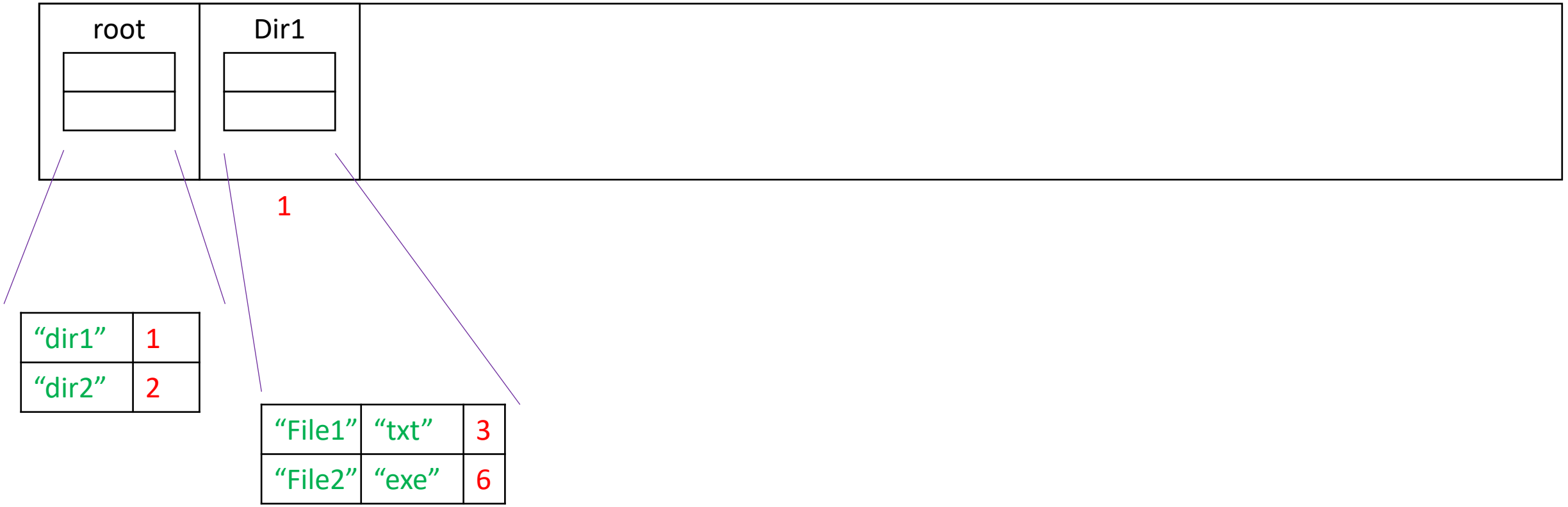
Structure



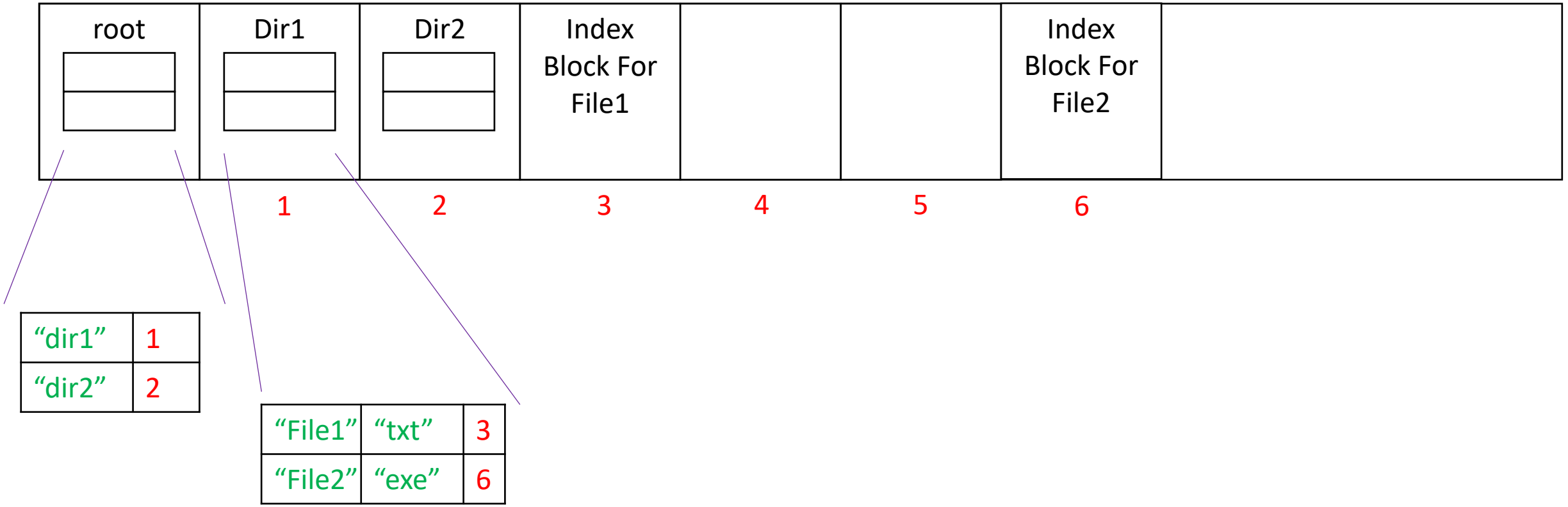
Structure



Structure



Structure


root
Dir1
Dir2
Index Block For File1
Index Block For File2
1
2
3
4
5
6
“dir1” 1
“dir2” 2
“File1” “txt” 3
“File2” “exe” 6

Structure: Subdirectory

```
//The attribute packed means to not align these things
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
  } __attribute__((packed)) files[MAX_FILES_IN_DIR];  //There is an array of these

  //This is some space to get this to be exactly the size of the disk block.
  //Don't use it for anything.
  char padding[BLOCK_SIZE - MAX_FILES_IN_DIR * sizeof(struct cs1550_file_directory) - sizeof(int)];
} ;
```

Structure: Subdirectory

```
//The attribute packed means to not align these things
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
  } __attribute__((packed)) files[MAX_FILES_IN_DIR];  //There is an array of these

  //This is some space to get this to be exactly the size of the disk block.
  //Don't use it for anything.
  char padding[BLOCK_SIZE - MAX_FILES_IN_DIR * sizeof(struct cs1550_file_directory) - sizeof(int)];
} ;
```

Number of Files

Structure: Subdirectory

```
//The attribute packed means to not align these things
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
  } __attribute__((packed)) files[MAX_FILES_IN_DIR];  //There is an array of these

  //This is some space to get this to be exactly the size of the disk block.
  //Don't use it for anything.
  char padding[BLOCK_SIZE - MAX_FILES_IN_DIR * sizeof(struct cs1550_file_directory) - sizeof(int)];
} ;
```

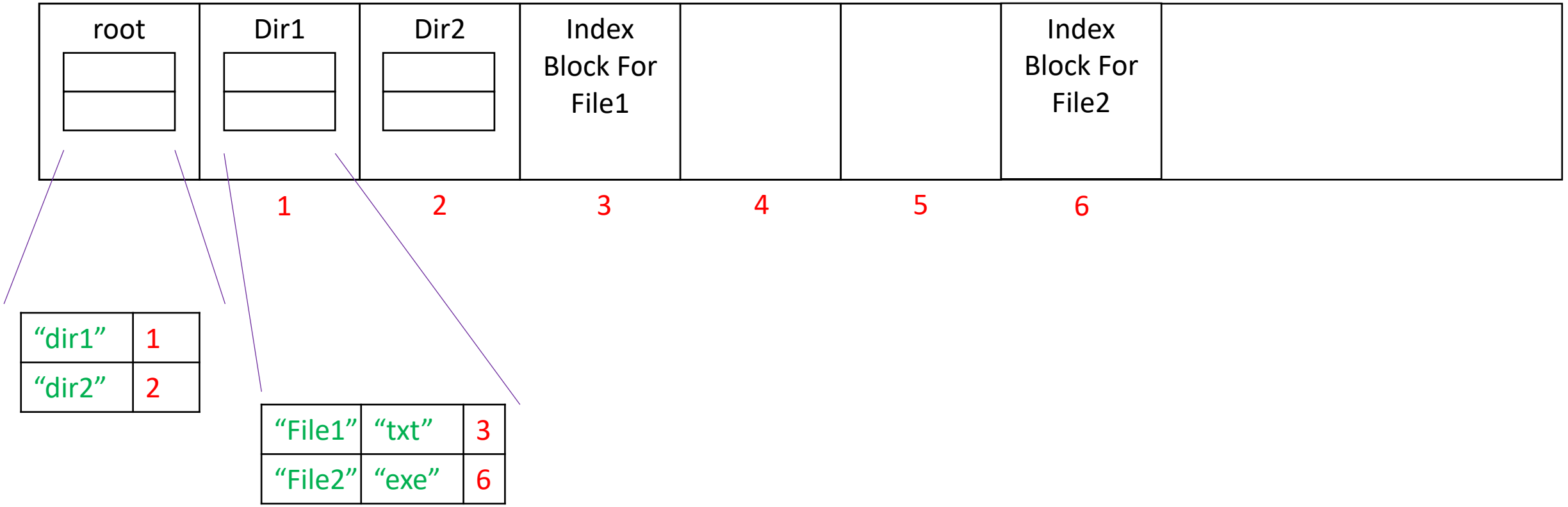
File Table

Structure: Subdirectory

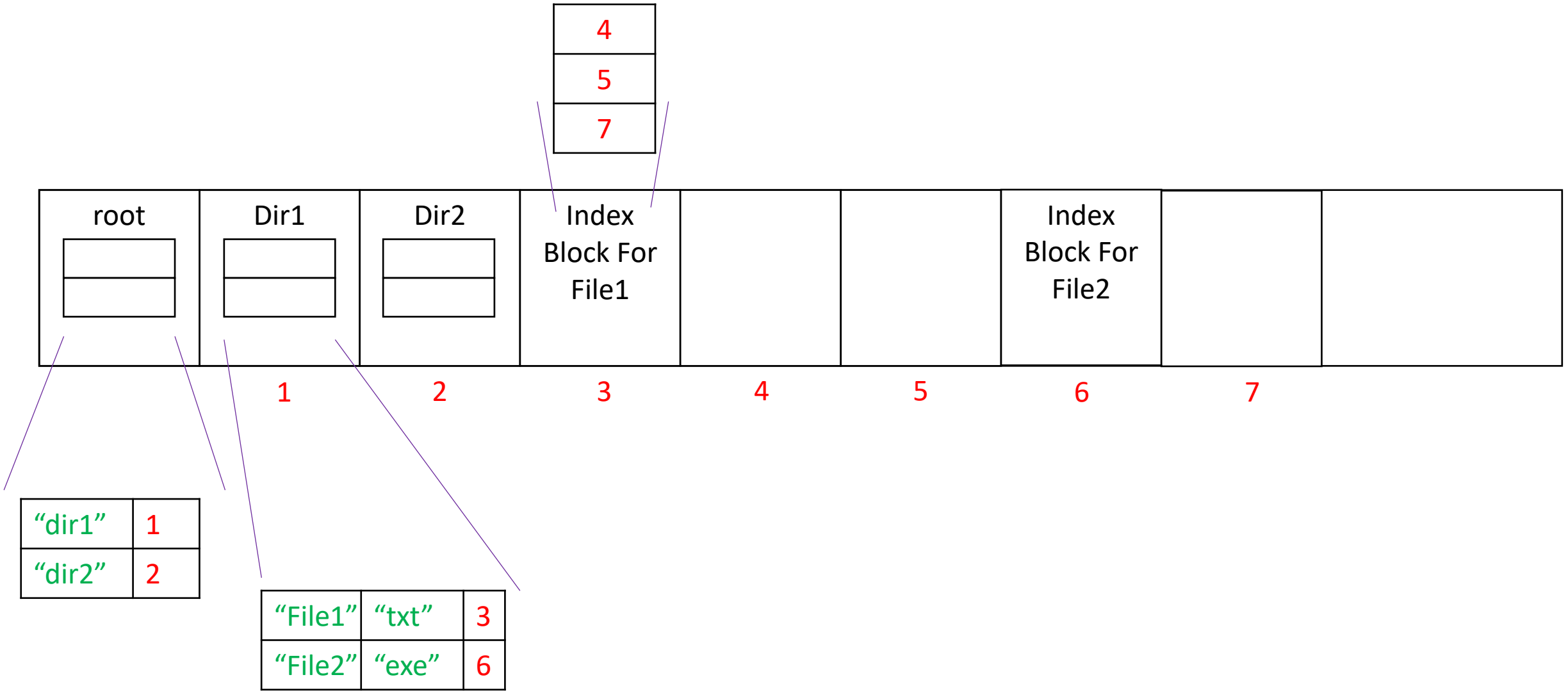
```
//The attribute packed means to not align these things
struct cs1550_directory_entry
{
  int nFiles; //How many files are in this directory.
        //Needs to be less than MAX_FILES_IN_DIR

  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
  } __attribute__((packed)) files[MAX_FILES_IN_DIR];  //There is an array of these

  //This is some space to get this to be exactly the size of the disk block.
  //Don't use it for anything.
  char padding[BLOCK_SIZE - MAX_FILES_IN_DIR * sizeof(struct cs1550_file_directory) - sizeof(int)];
} ;
```

Padding

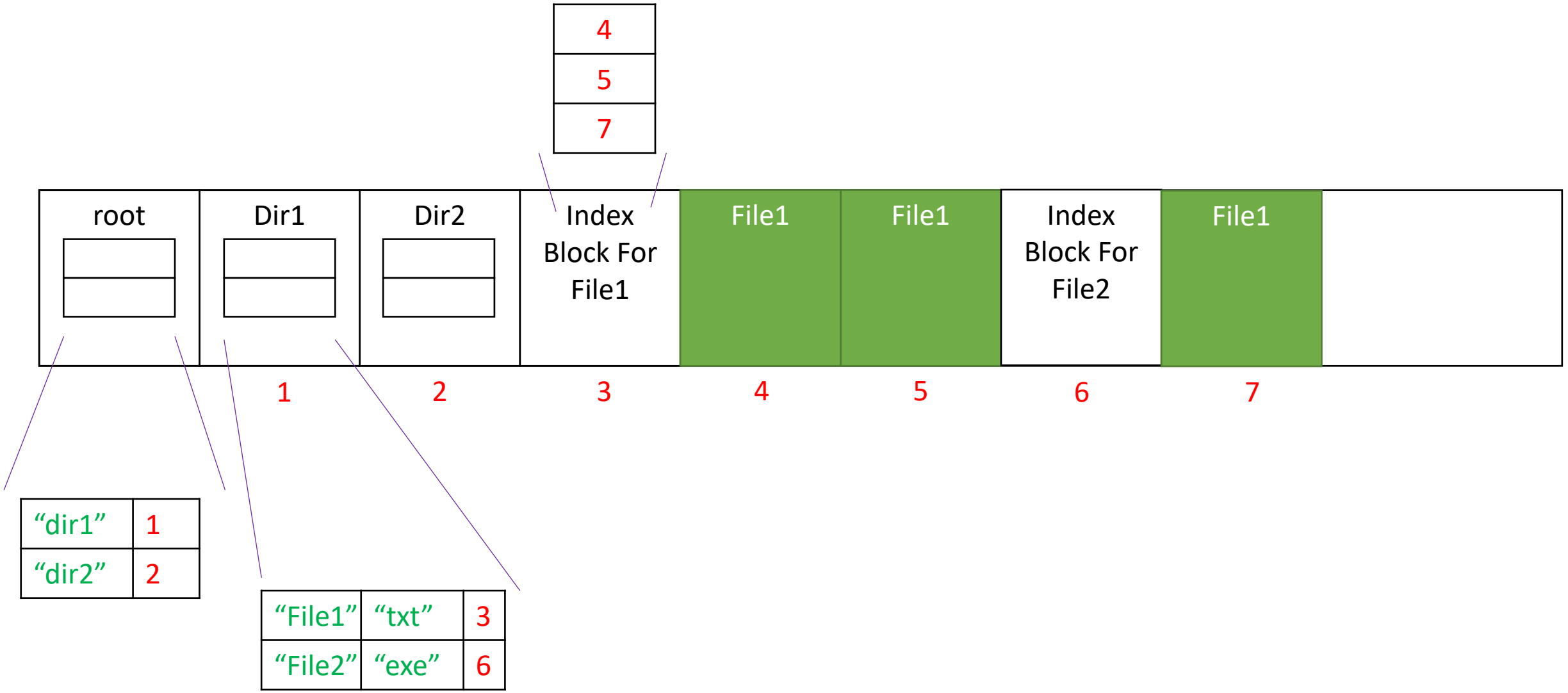
Structure


root
Dir1
Dir2
Index Block For File1
Index Block For File2
1
2
3
4
5
6
“dir1” 1
“dir2” 2
“File1” “txt” 3
“File2” “exe” 6

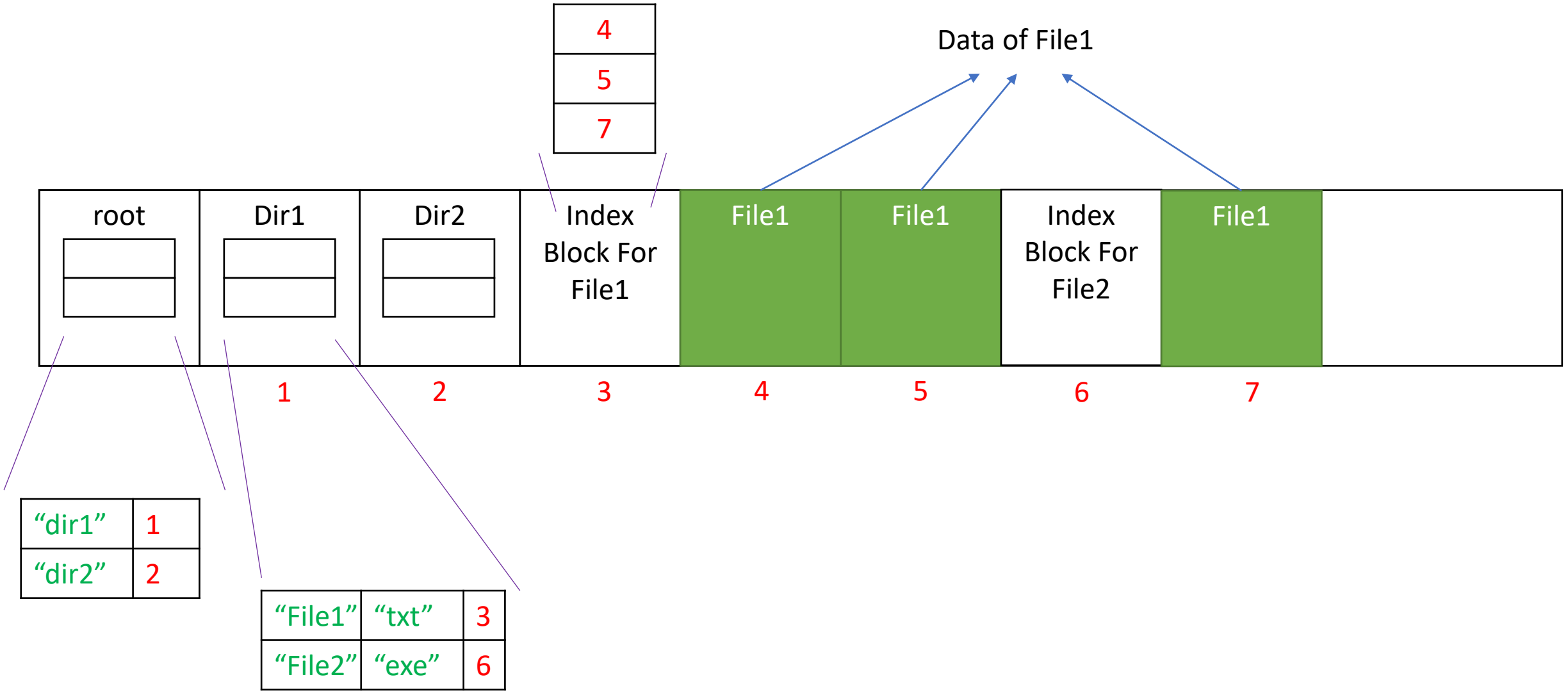
Structure


4
5
7
root
Dir1
Dir2
Index Block For File1
Index Block For File2
1
2
3
4
5
6
7
“dir1” 1
“dir2” 2
“File1” “txt” 3
“File2” “exe” 6

Structure


4
5
7
root
Dir1
Dir2
Index Block For File1
File1
File1
Index Block For File2
File1
1
2
3
4
5
6
7
“dir1” 1
“dir2” 2
“File1” “txt” 3
“File2” “exe” 6

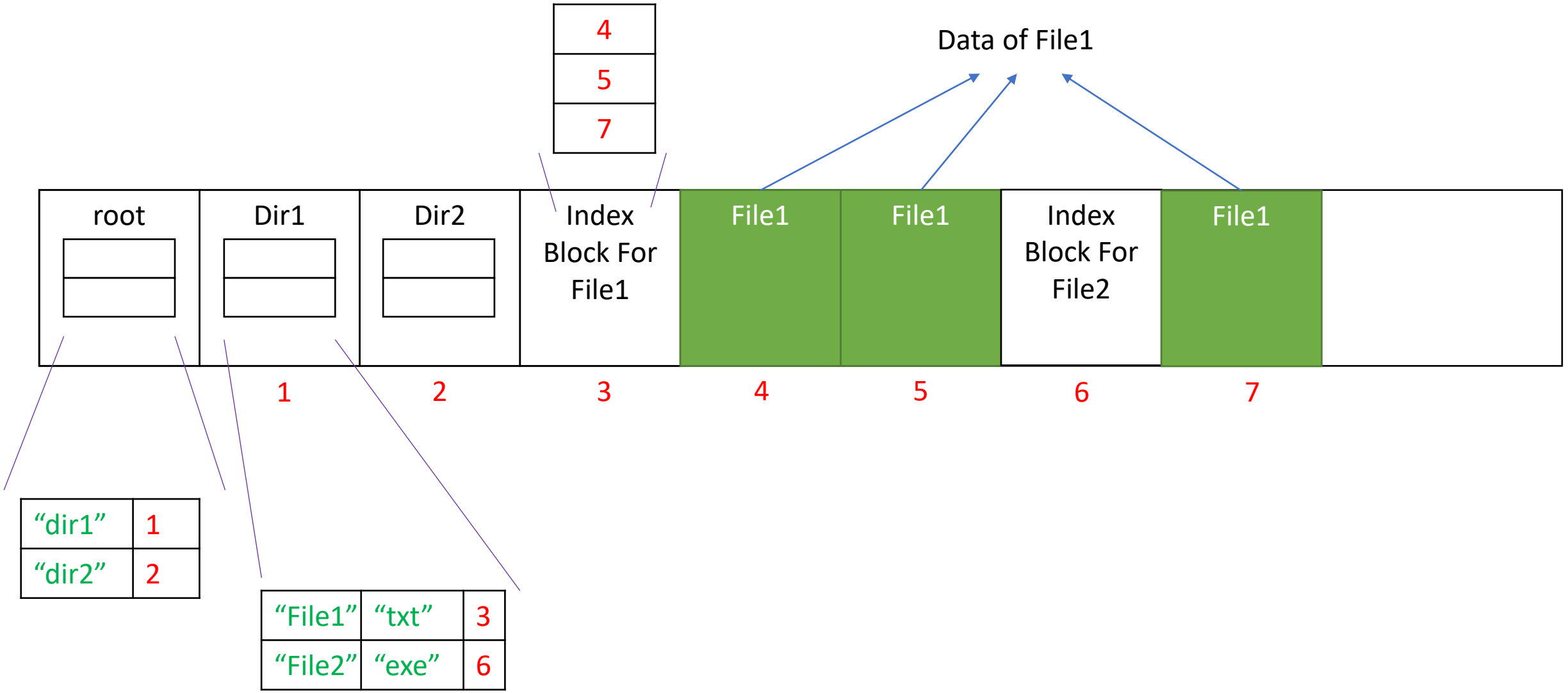
Structure


4
5
7
Data of File1
root
Dir1
Dir2
Index Block For File1
File1
File1
Index Block For File2
File1
1
2
3
4
5
6
7
“dir1” 1
“dir2” 2
“File1” “txt” 3
“File2” “exe” 6

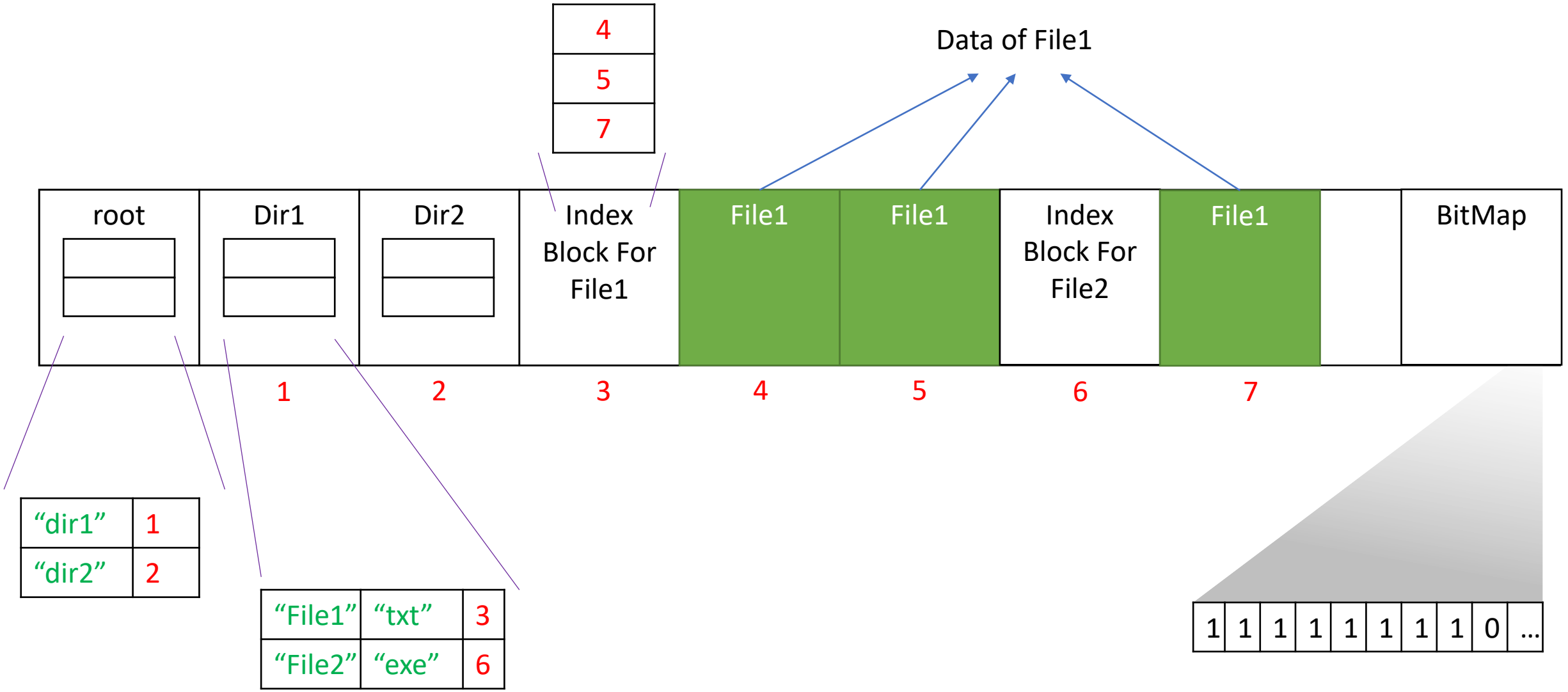
Structure: Index Block

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

Structure

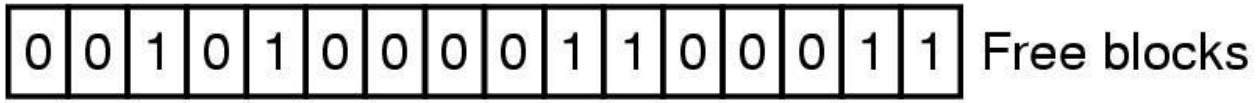

4
5
7
Data of File1
root
Dir1
Dir2
Index Block For File1
File1
File1
Index Block For File2
File1
1
2
3
4
5
6
7
“dir1” 1
“dir2” 2
“File1” “txt” 3
“File2” “exe” 6

Structure


4
5
7
Data of File1
root
Dir1
Dir2
Index Block For File1
File1
File1
Index Block For File2
File1
BitMap
1
2
3
4
5
6
7
“dir1” 1
“dir2” 2
“File1” “txt” 3
“File2” “exe” 6
1 1 1 1 1 1 1 1 0 …

Disk Management

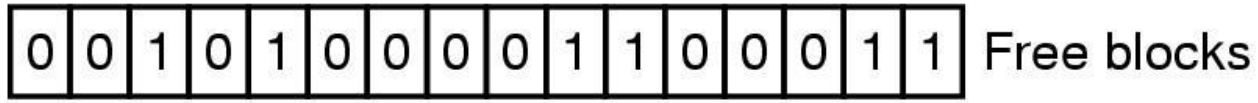
- Manage free (or empty) space using **bitmap**



(a)

Disk Management

- Manage free (or empty) space using bitmap

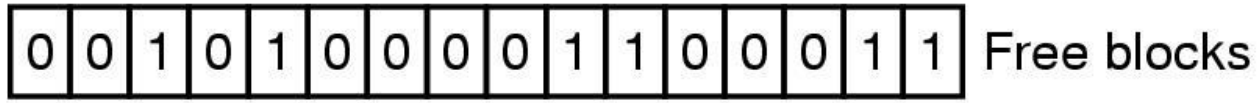


(a)

- Each block indicated by **a binary bit**

Disk Management

- Manage free (or empty) space using bitmap



(a)

- Each block indicated by **a binary bit**
- How large should it be

  Disk: 5MB Block: 512Bytes => Number of Blocks: $5M/512 = 5 * 2^{11}$

  Size of bitmap: $5 * 2^{11}$ bits => $5 * 2^{8}$ Bytes = 2.5 Blocks = 3 Blocks

Disk Management

- Initialize data map
  - The .disk will be all 0 at beginning
  - However, we have our **Root Directory, Bitmap** space occupied
  - Blocks holding our **Root Directory, Bitmap** must be set to 1

Disk Management

- Initialize data map
  - The .disk will be all 0 at beginning
  - However, we have our **Root Directory, Bitmap** space occupied
  - Blocks holding our **Root Directory, Bitmap** must be set to 1
- Options
  - Do it in main function
  - Do it in CS1550_init

Syscalls

- **cs1550_getattr**
- **cs1550_mkdir**
- **cs1550_readdir**
- cs1550_rmdir
- **cs1550_mknod**
- **cs1550_write**
- **cs1550_read**
- cs1550_unlink
- cs1550_truncate
- cs1550_open
- cs1550_flush
- cs1550_init
- cs1550_destory

Syscalls

- **cs1550_getattr**
- **cs1550_mkdir**
- **cs1550_readdir**
- cs1550_rmdir
- **cs1550_mknod**
- **cs1550_write**
- **cs1550_read**
- cs1550_unlink
- cs1550_truncate
- cs1550_open
- cs1550_flush
- cs1550_init
- cs1550_destory

Return errors based on project description

Requirements and submission

- Well-commented cs1550.c
- Rubric

Item	Grade
cs1550_getattr	15%
cs1550_mkdir	15%
cs1550_readdir	15%
cs1550_mknod	15%
cs1550_write	15%
cs1550_read	15%
File System works correctly	10%

CS 1550

Week 14

–

Project 4

Teaching Assistant

Henrique Potter