香港中文大學
The Chinese University of Hong Kong

CSCI5550 Advanced File and Storage Systems

Programming Assignment 01: In-Memory File System (IMFS) using FUSE

Outline

- **FUSE Introduction**
  - Motivation of using FUSE
  - What is inside FUSE
  - In-kernel v.s. FUSE
- **Programming with FUSE**
  - Installing FUSE
  - FUSE operations
  - An example project: LSYSFS
- **Programming Assignment 1**
  - On-Disk Organization
  - Basic Commands for Grading
  - Bonus

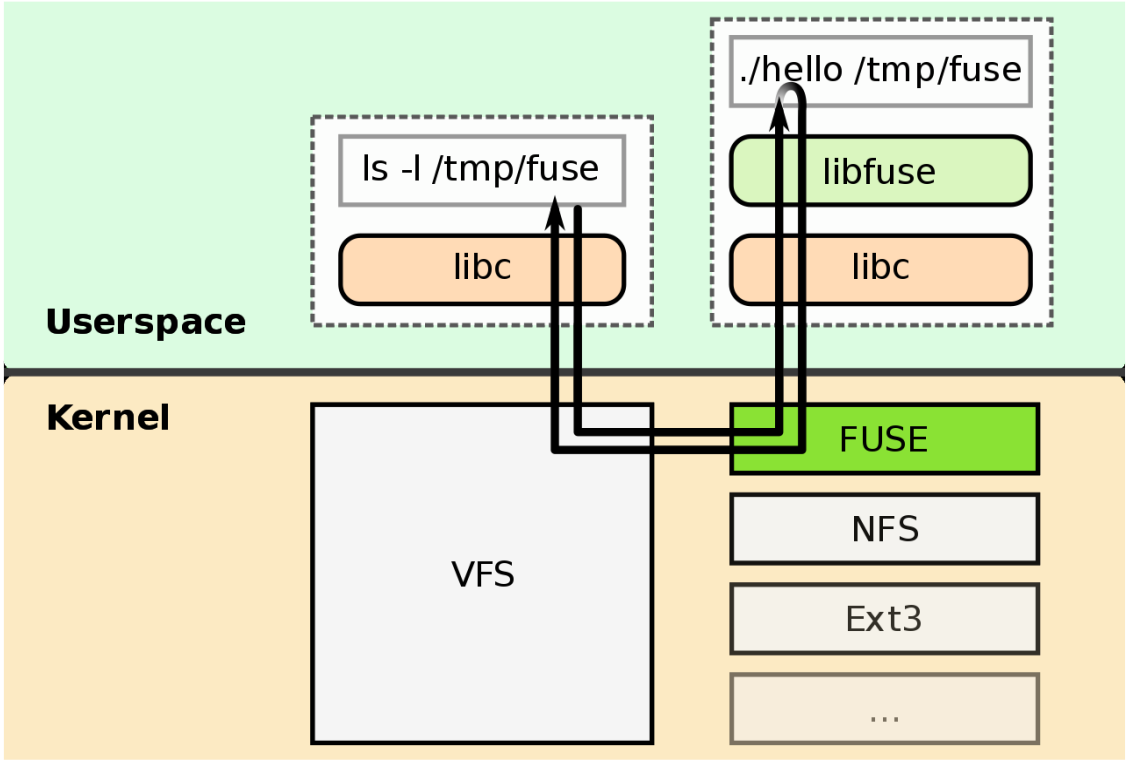
Motivation of using FUSE

- **File System in Kernel-space**
  - Very difficult to build
  - Need careful use of synchronization primitives
  - Only C language supported
  - Standard C libraries not available
  - Need root privilege
- **File System in User-space (using FUSE!)**
  - Framework to implement user-space file system
  - Easy to write: Avoid awful coding in kernel
  - Easy to test: Run like a normal user program
  - Easy to integrate libraries: Can easily deploy libraries
  - Trade performance for flexibility

What is inside FUSE?

- Kernel module
  - fuse.ko
  - file system (fusefs)
  - virtual device (/dev/fuse)
- User-space library
  - libfuse.so
  - Framework to export FUSE API

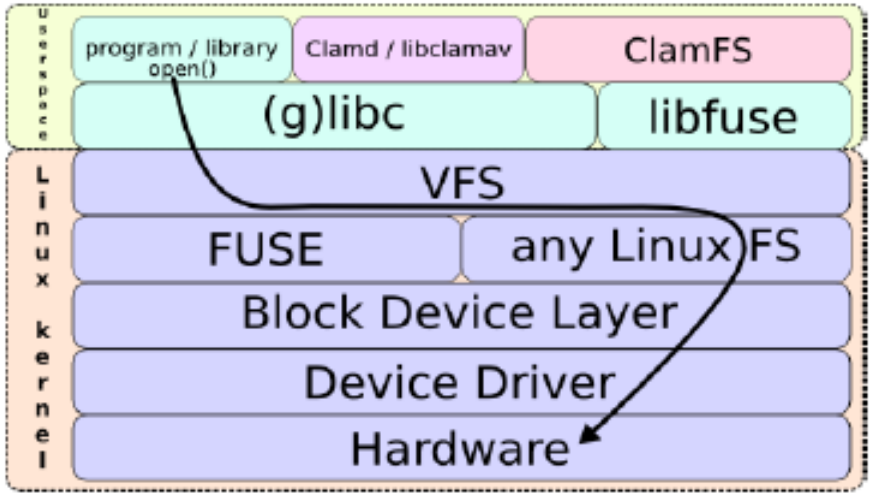
- A flow-chart diagram showing a request from user-space to list files.



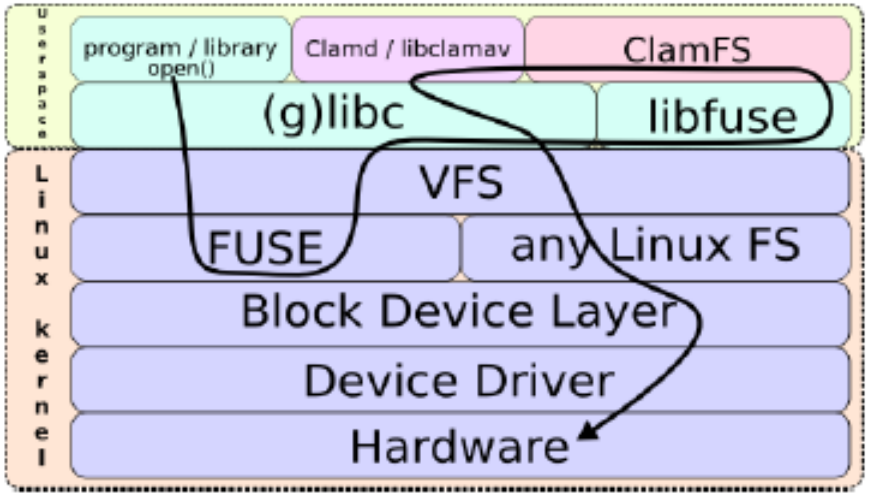
Source: https://en.wikipedia.org/wiki/Filesystem_in_Userspace

In-kernel v.s. FUSE

In-kernel



FUSE



Source: http://clamfs.sourceforge.net/

Outline

- **FUSE Introduction**
  - Motivation of using FUSE
  - What is inside FUSE
  - In-kernel v.s. FUSE
- **Programming with FUSE**
  - Installing FUSE
  - FUSE operations
  - An example project: LSYSFS
- **Programming Assignment 1**
  - On-Disk Organization
  - Basic Commands for Grading
  - Bonus

Installing FUSE

- A Linux environment is required before installing FUSE. I use the VirtualBox to install Ubuntu 16.04 (64-bit).
- Install FUSE and all the dependencies:

```
$ sudo apt-get update
$ sudo apt-get install gcc fuse libfuse-dev make cmake
```

- To check the FUSE version:

```
$ fusermount -V
fusermount version: 2.9.4
```

- Please note that FUSE version that I have is 2.9.4. However, it should be fine if your FUSE version is 2.9.x.

FUSE operations (1/2)

- FUSE uses the callback mechanism to bind the user-defined functions with file operations.
- Callbacks are a set of functions you write to implement file operations, and struct fuse_operations containing pointers to them.
- Example:

```
static struct fuse_operations operations = {
    .getattr = do_getattr,
    .readdir = do_readdir,
    .read    = do_read,
    .mkdir   = do_mkdir,
    .mknod   = do_mknod,
    .write   = do_write,
};
```

file operations user-defined functions

FUSE operations (2/2)

- Using fuse_main to pass the function pointers to FUSE module:

```
int main(int argc, char* argv[])
{
    return fuse_main(argc, argv, &operations, NULL);
}
```

- Please use the below link to check all the FUSE operations:
  - https://libfuse.github.io/doxygen/structfuse__operations.html

- Next, an example project (LSYSFS) will be used to show how to implement some basic callback functions.
  - https://github.com/MaaSTaaR/LSYSFS

LSYSFS (1/6)

- LSYSFS is an example of using FUSE to build a simple in-memory filesystem that supports creating new files and directories under root directory, but it doesn't support deleting files and directories.

- Below is the data structure that LSYSFS uses:

```
char dir_list[ 256 ][ 256 ];
int curr_dir_idx = -1;
```

The first index is for directory index
The second index for directory name

```
char files_list[ 256 ][ 256 ];
int curr_file_idx = -1;
```

The first index is for file index
The second index for file name

```
char files_content[ 256 ][ 256 ];
int curr_file_content_idx = -1;
```

The first index is for file index (same as above)
The second index for file content

LSYSFS (2/6)

- **gettar**
  - **gettar** is the most important function among all. It is in charge of reading the metadata of a given path, and it is always called before any operation made.

```
static int do_getattr( const char *path, struct stat *st ){
    st->st_uid = getuid();
    st->st_gid = getgid();
    st->st_atime = time( NULL );
    st->st_mtime = time( NULL );
    if ( strcmp( path, "/" ) == 0 || is_dir( path ) == 1 ){
        st->st_mode = S_IFDIR | 0755;
        st->st_nlink = 2;
    }
    else if ( is_file( path ) == 1 ){
        st->st_mode = S_IFREG | 0644;
        st->st_nlink = 1;
        st->st_size = 1024;
    }
    else{
        return -ENOENT;
    }
    return 0;
}
```

LSYSFS (3/6)

- **readdir**
  - **readdir** will be invoked when ls is given. That is, **readdir** will return all the names under the current directory.

```
static int do_readdir( const char *path, void *buffer,
        fuse_fill_dir_t filler, off_t offset, struct fuse_file_info *fi )
{
    filler( buffer, ".", NULL, 0 ); // Current Directory
    filler( buffer, "..", NULL, 0 ); // Parent Directory

    if ( strcmp( path, "/" ) == 0 ){
        for ( int curr_idx = 0; curr_idx <= curr_dir_idx; curr_idx++ )
            filler( buffer, dir_list[ curr_idx ], NULL, 0 );

        for ( int curr_idx = 0; curr_idx <= curr_file_idx; curr_idx++ )
            filler( buffer, files_list[ curr_idx ], NULL, 0 );
    }
    return 0;
}
```

LSYSFS (4/6)

- **mkdir** & **mknod**
  - **mkdir** will be used when there is a creation of a directory, and **mknod** will be used when a new file is created.

```
static int do_mkdir( const char *path, mode_t mode ){
    path++; // eliminate "/" to get dir name
    curr_dir_idx++;
    strcpy( dir_list[ curr_dir_idx ], path );

    return 0;
}
static int do_mknod( const char *path, mode_t mode, dev_t rdev ){
    path++; // eliminate "/" to get file name
    curr_file_idx++;
    strcpy( files_list[ curr_file_idx ], path );

    curr_file_content_idx++;
    strcpy( files_content[ curr_file_content_idx ], "" );

    return 0;
}
```

LSYSFS (5/6)

- **write** & **read**
  - **write** is for writing new content to a file, and **read** is for reading the file content.

```
static int do_write( const char *path, const char *buffer,
        size_t size, off_t offset, struct fuse_file_info *info ){
    int file_idx = get_file_index( path );
    if ( file_idx == -1 ) // No such file
        return;
    // new content is in buffer
    strcpy( files_content[ file_idx ], buffer );

    return size;
}
static int do_read( const char *path, char *buffer,
        size_t size, off_t offset, struct fuse_file_info *fi ){
    int file_idx = get_file_index( path );
    if ( file_idx == -1 )
        return -1;
    char *content = files_content[ file_idx ];
    // using buffer to return file content
    memcpy( buffer, content + offset, size );

    return strlen( content ) - offset;
}
```

LSYSFS (6/6)

- To compile LSYSFS, modify the fifth line of Makefile and the type “make”:

```
$(COMPILER) $(FILESYSTEM_FILES) -o lsysfs `pkg-config fuse --cflags --libs`
```

```
$(COMPILER) -D_GNU_SOURCE $(FILESYSTEM_FILES) -o lsysfs `pkg-config fuse --cflags --libs`
```

- To run LSYSFS:
  - Open one terminal to start LSYSFS: `$ ./lsysfs –f MOUNT_POINT`
  - Open another terminal and go to the mount point to test it!

- LSYSFS should work well with following commands:
  - cd, ls, mkdir, echo “string” >> file, touch, and cat

Outline

- **FUSE Introduction**
  - Motivation of using FUSE
  - What is inside FUSE
  - In-kernel v.s. FUSE
- **Programming with FUSE**
  - Installing FUSE
  - FUSE operations
  - An example project: LSYSFS
- **Programming Assignment 1**
  - On-Disk Organization
  - Basic Commands for Grading
  - Bonus

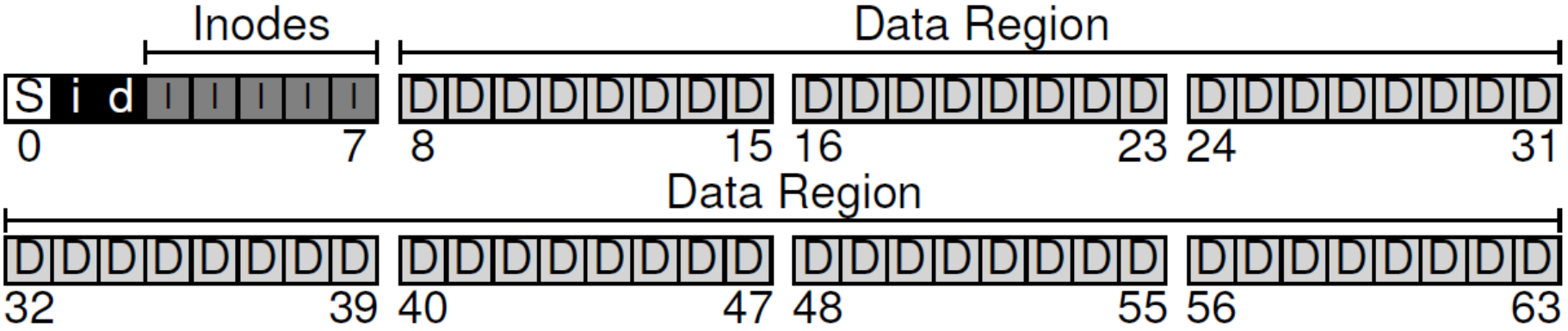
Programming Assignment 1

- In this programming assignment, you are required to build a simple in-memory filesystem, called IMFS, using FUSE.

- Requirements:
  - Follow the On-Disk Organization: Superblock, Metadata, and Data region;
  - Keep all the data structures (including metadata and data) in the memory;
  - Work well with basic shell commands.

Overall Organization

- On-Disk Organization: A series of blocks, each of size 4 KB, is addressed from 0 to N −1.
  - Metadata Region
    - Superblock (S): containing special information about IMFS
    - Inode Bitmap (i): indicating the availability of Inodes (I)
    - Data Bitmap (d): indicating the availability of Data Blocks (D)
    - Inodes (I): accommodating inodes
  - Data Region
    - Data Blocks (D): persisting user data

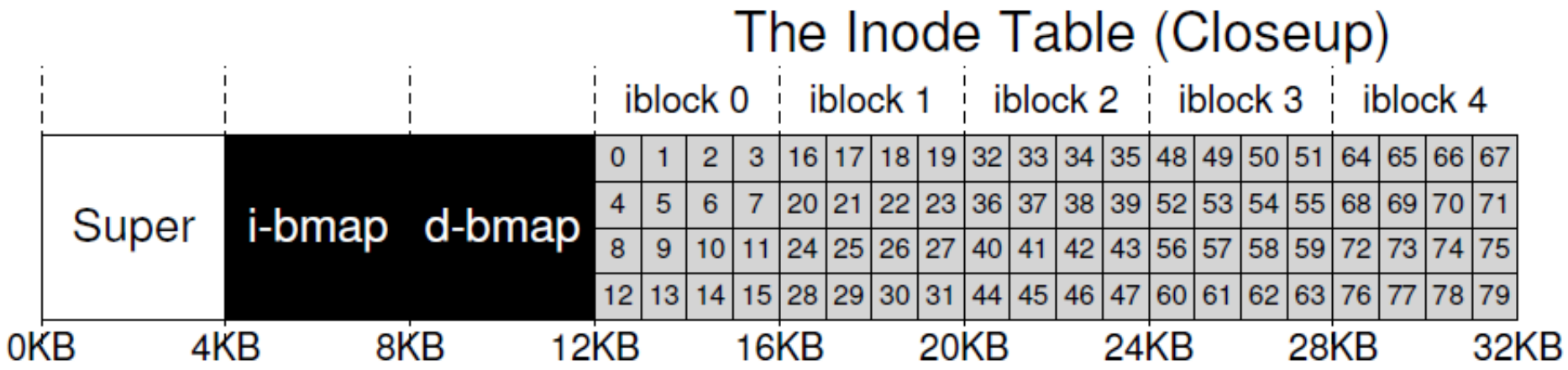


Metadata Region (1/3)

- Superblock (S)

```
struct superblock {
    unsigned int size_ibmap;
    unsigned int size_dbmap;
    //unsigned int size_inode; -> not used in IMFS
    unsigned int size_per_data_region;
    unsigned int size_filename;
    unsigned int root_inum;
    unsigned int num_disk_ptrs_per_inode;
};
superblock SB;
```

All the units of size will be counted in bytes

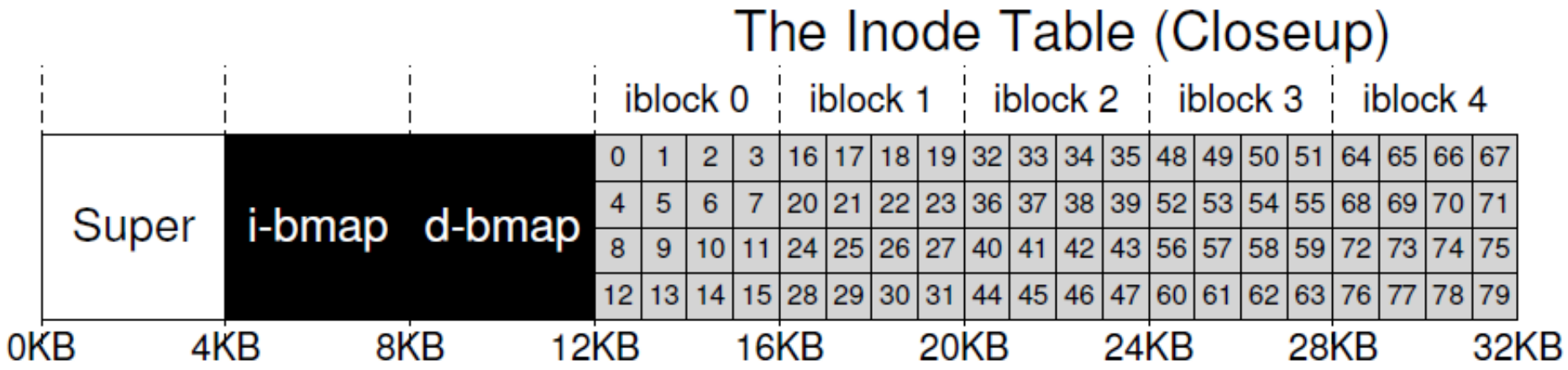


Metadata Region (2/3)

- Inode Bitmap (i) / Data Bitmap (d)

```
bool inode_bitmap[SB.size_ibmap];
bool data_bitmap[SB.size_dbmap];
```

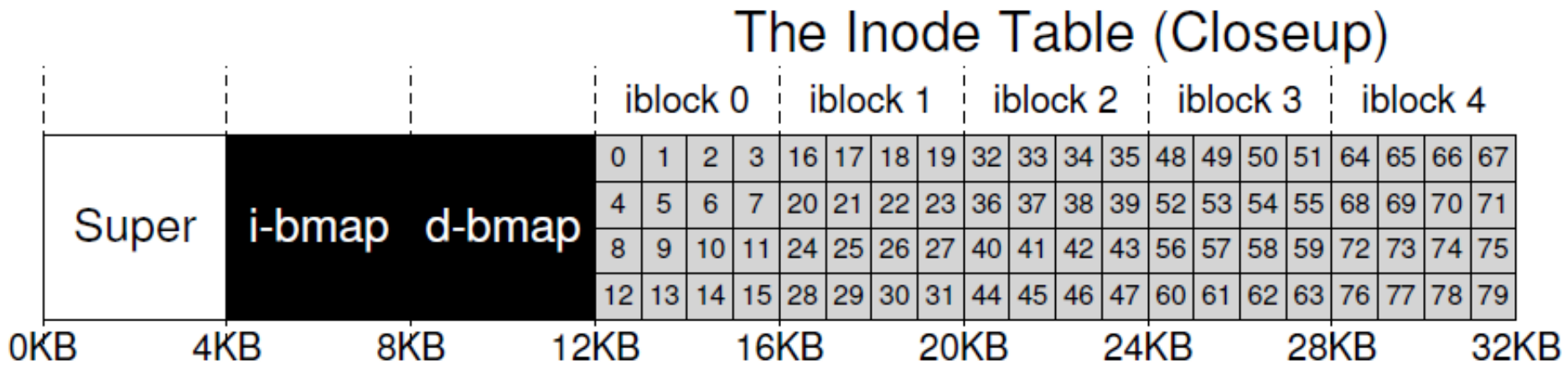
inode_bitmap: indicating the availability of inodes.
data_bitmap: indicating the availability of data blocks.



Metadata Region (3/3)

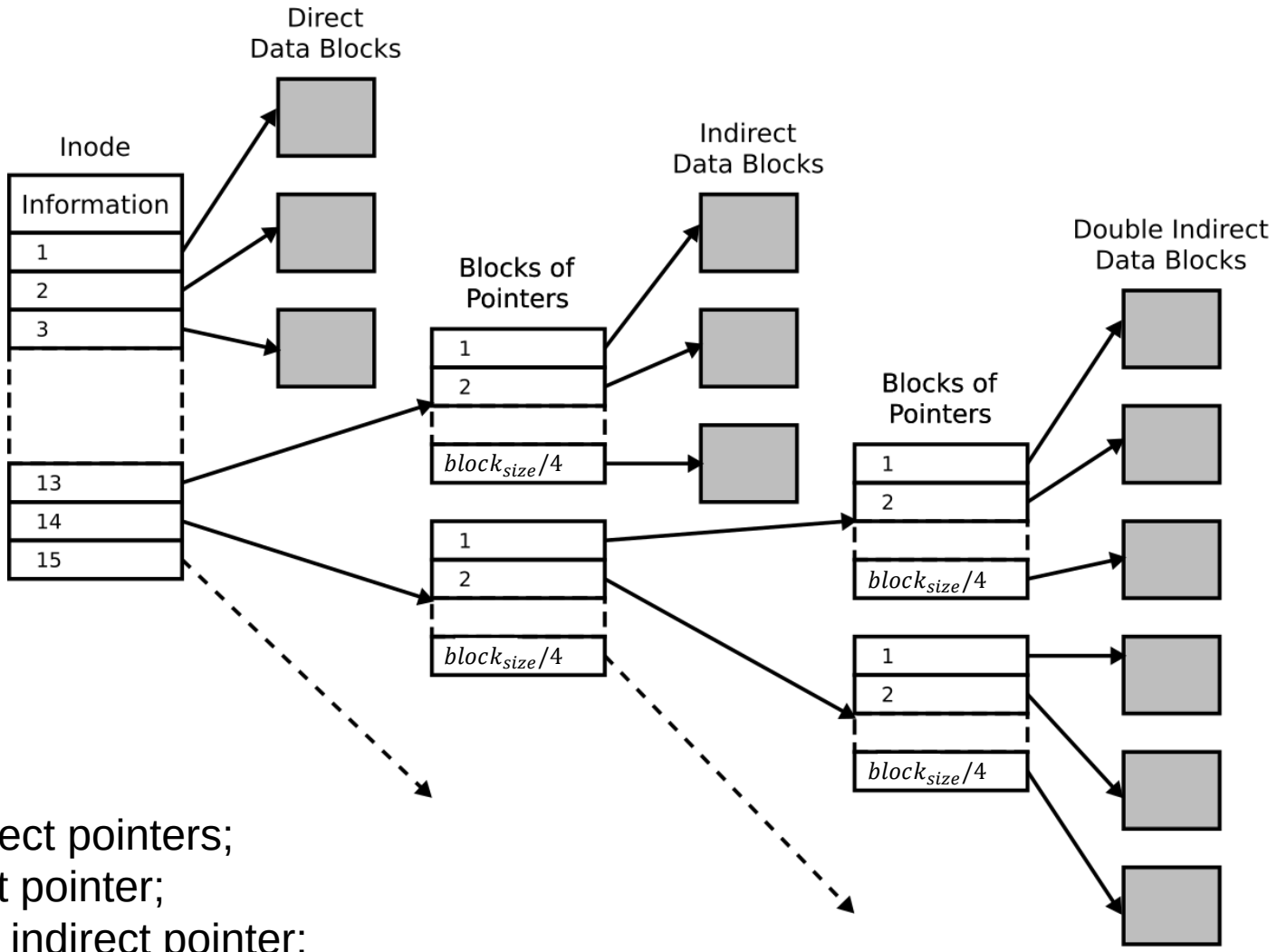
- Inodes (I)

```
struct inode {
    int flag;
        // indicating the type of file of this inode
        // (regular file or dir or something else)
    int blocks; // how many blocks have been used
    int block[SB.num_disk_ptrs_per_inode];
                    // a set of inum points to data_region
    int links_count; // # of hard links to this file
};
inode inode_table[SB.size_ibmap];
```



Metadata Region (3/3)

- Below is an example showing how indirect pointers work.



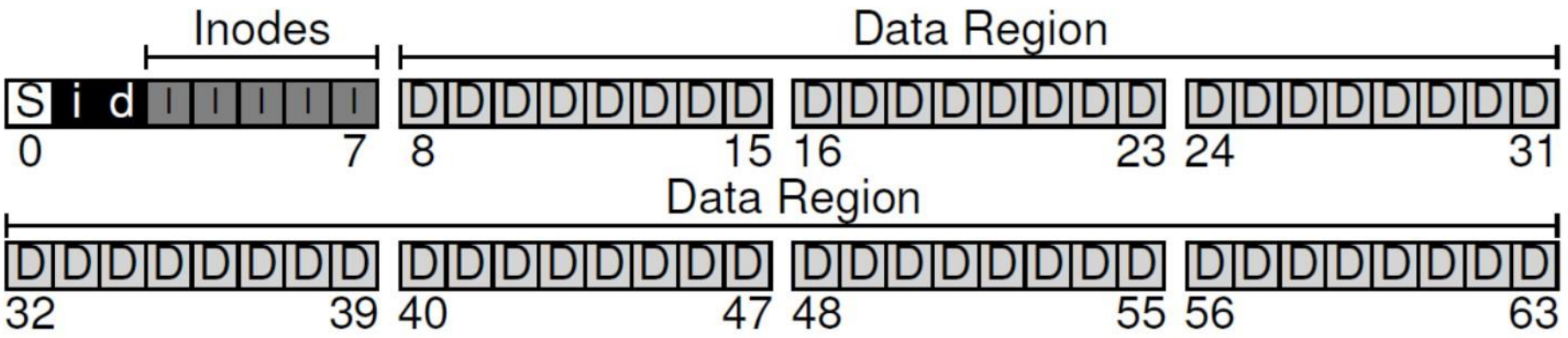
Each IMFS inode
M disk pointers:

- **0~M-3** are for direct pointers;
- **M-2** is for indirect pointer;
- **M-1** is for double indirect pointer;

Data Region (1/3)

- Directory and file organization share the same data region

```
struct data_region {
    char space[SB.size_per_data_region];
};
data_region d_reg[SB.size_dbmap];
```

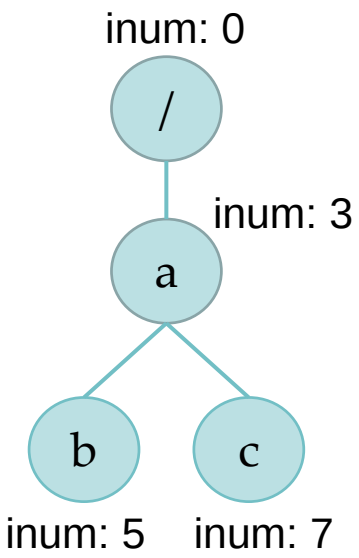


Data Region (2/3)

- Directory Organization

sizeof(**inum**) = 4, and sizeof(**file_name**) = sizeof(SB.size_filename)
As a result, each entry will consume 4 + sizeof(SB.size_filename) bytes

Use the space of data region to store **inum** and **file_name**.



The data_region of directory a:

inum	file_name
0	..
3	.
5	b
7	c

The **file_name** can be regular file name or directory name

Data Region (3/3)

- File Organization
  - Directly store file content into the space of data region.

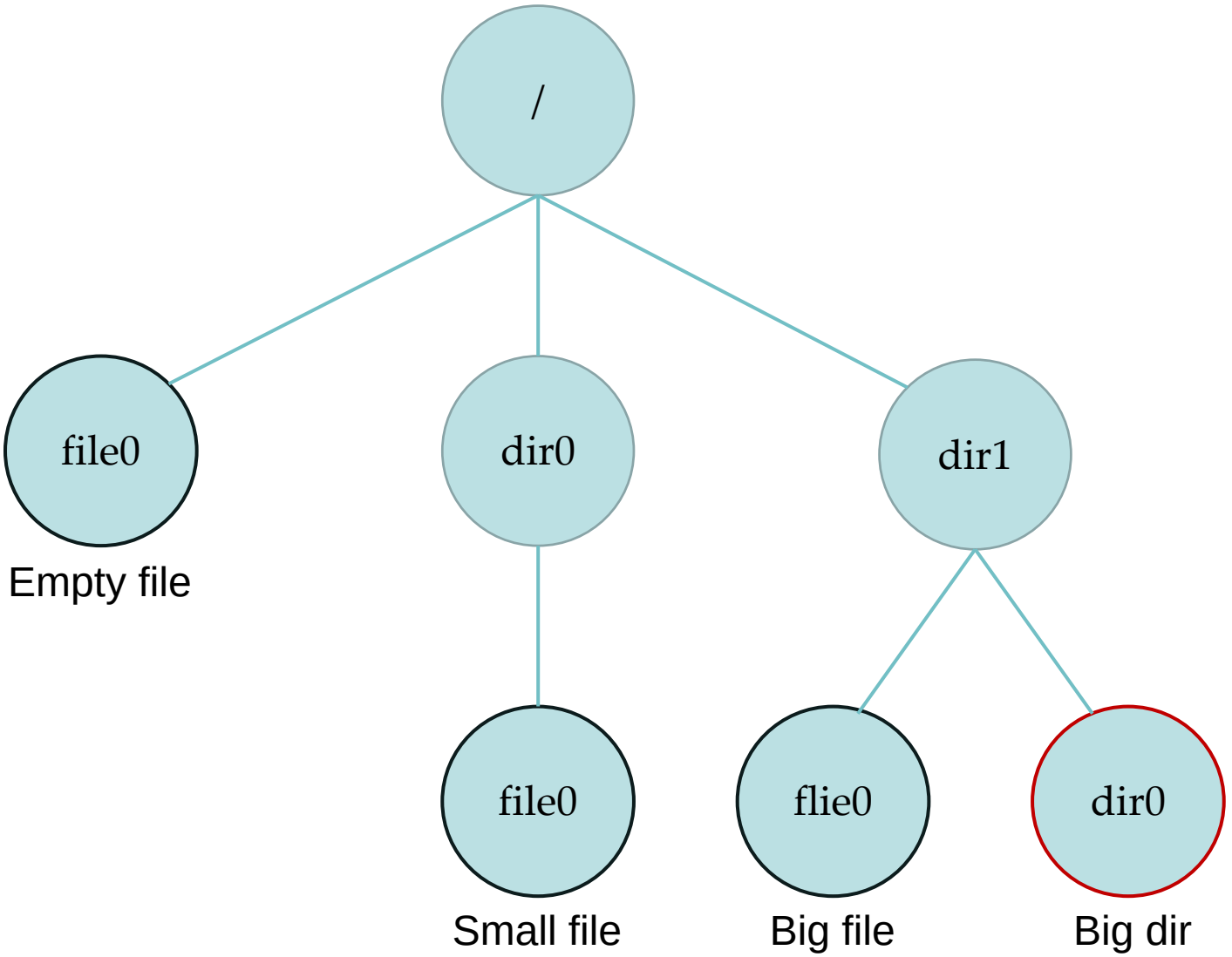
- Bigger Directories or Files: Multi-Level Index
  - Indirect Pointer
  - Double Indirect Pointer

Parameter Setting for Assessment

- `size_ibmap = 32`
- `size_dbmap = ~~32~~` → **`512`**
- `size_per_data_region = 64`
- `size_filename = 12`
- `root_inum = 0`
- `num_disk_ptrs_per_inode = 4`
  - **2** direct pointers;
  - **1** indirect pointer;
  - **1** double indirect pointer.

- *Note: For ease of testing, the values of the parameters of IMFS are set to be small.*

Directory Tree for Assessment



Basic Requirements (100%)

- **(5%)** **`cd`**: changing the current directory
- **(5%)** **`ls`**: listing all files & directory under the current directory
- **(15%)** **`mkdir`**: creating a directory
- **(15%)** **`touch`**: for creating a file
- **(10%)** **`echo "string" >> file`**: writing a string to a file
- **(15%)** **`cat`**: reading a file
- **(15%)** **`rmdir`**: removing a directory
  - *Note:* **`rmdir`** *will recursively remove everything under that directory.*
- **(10%)** **`rm`** for deleting a file
- **(5%)** Support of "big directory" (having a large number of files)
- **(5%)** Support of "big file" (containing a very long string)

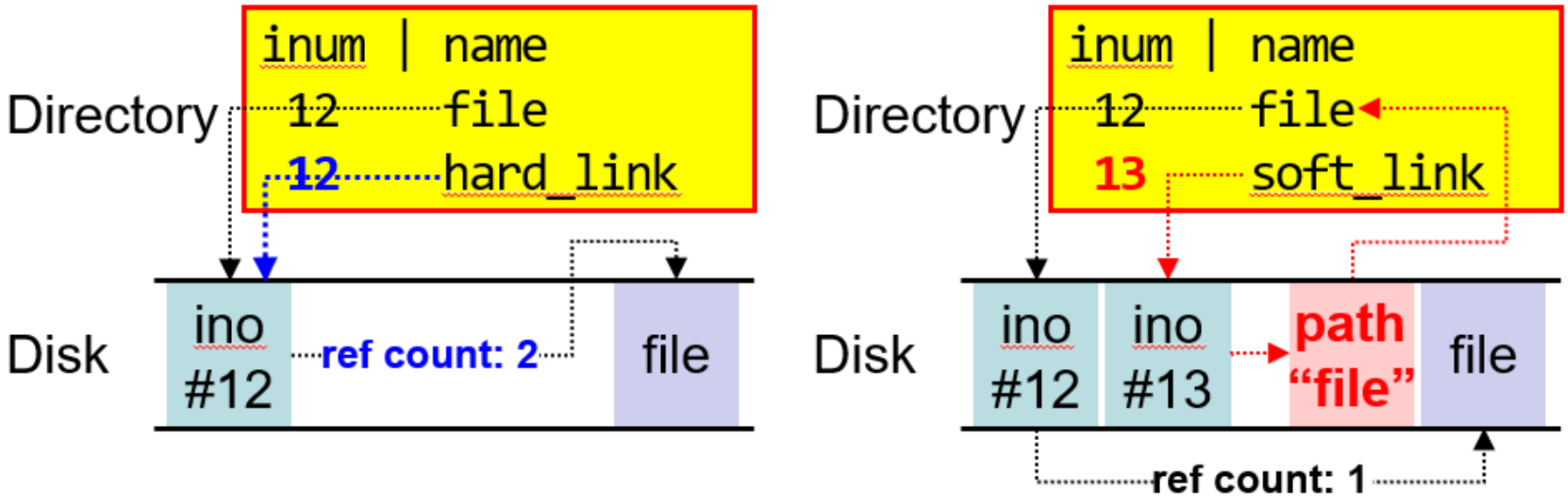
- *Note: No need to handle the boundary condition(s) that exceed the provided parameter setting.*

Bonus (10%): Hard Link & Soft Link

- Support of hard link and soft link in IMFS:

(5%) **`ln file1 link1.hardlink`**: Creating a hard link

(5%) **`ln -s file1 link1.softlink`**: Creating a soft link



- *Note: All the basic commands should be completed before getting the bonus.*

Submission

- **Submission Deadline**: **9:30am on March 23, 2020**

- Please submit two things to CUHK Blackboard:
  - ① **The whole package of your project**
    - ✓ Including the source code(s), Makefile, etc.
    - ✓ Naming the package of your IMFS project after your **student ID**
  - ② **A short report**
    - ✓ Showing how to run your project
    - ✓ Indicating the commands that are supported in your IMFS
    - ✓ Providing the screen shots of the results to prove the implemented commands are functioned well

- Discussion is allowed, but no **plagiarism**
  - Your code(s) will be cross-checked.

Reference

- https://libfuse.github.io/doxygen/structfuse__operations.html
- https://github.com/MaaSTaaR/LSYSFS
- http://clamfs.sourceforge.net/
- https://en.wikipedia.org/wiki/Filesystem_in_Userspace