File System

OS 2025 LAB 4

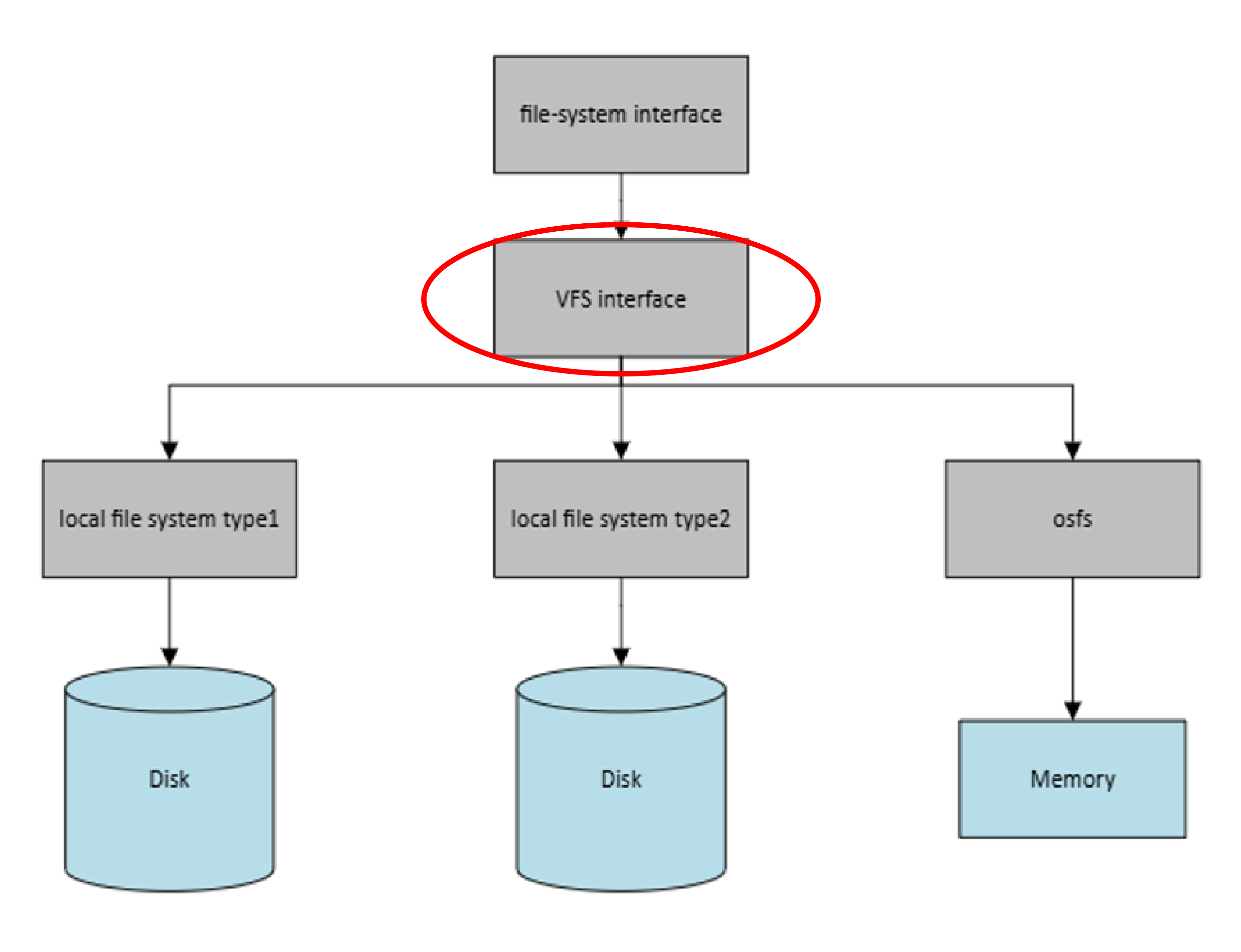
DUE DATE: 2025/01/02 17:00
(before lab 4 course finishes)

TA: NN6131037 陳彥廷、P76134846 王信智、P76144613 周安

Outline

1. Introduction to Virtual File System (VFS)
2. Layout of osfs
3. Access Path in File System
4. Requirement & Grading

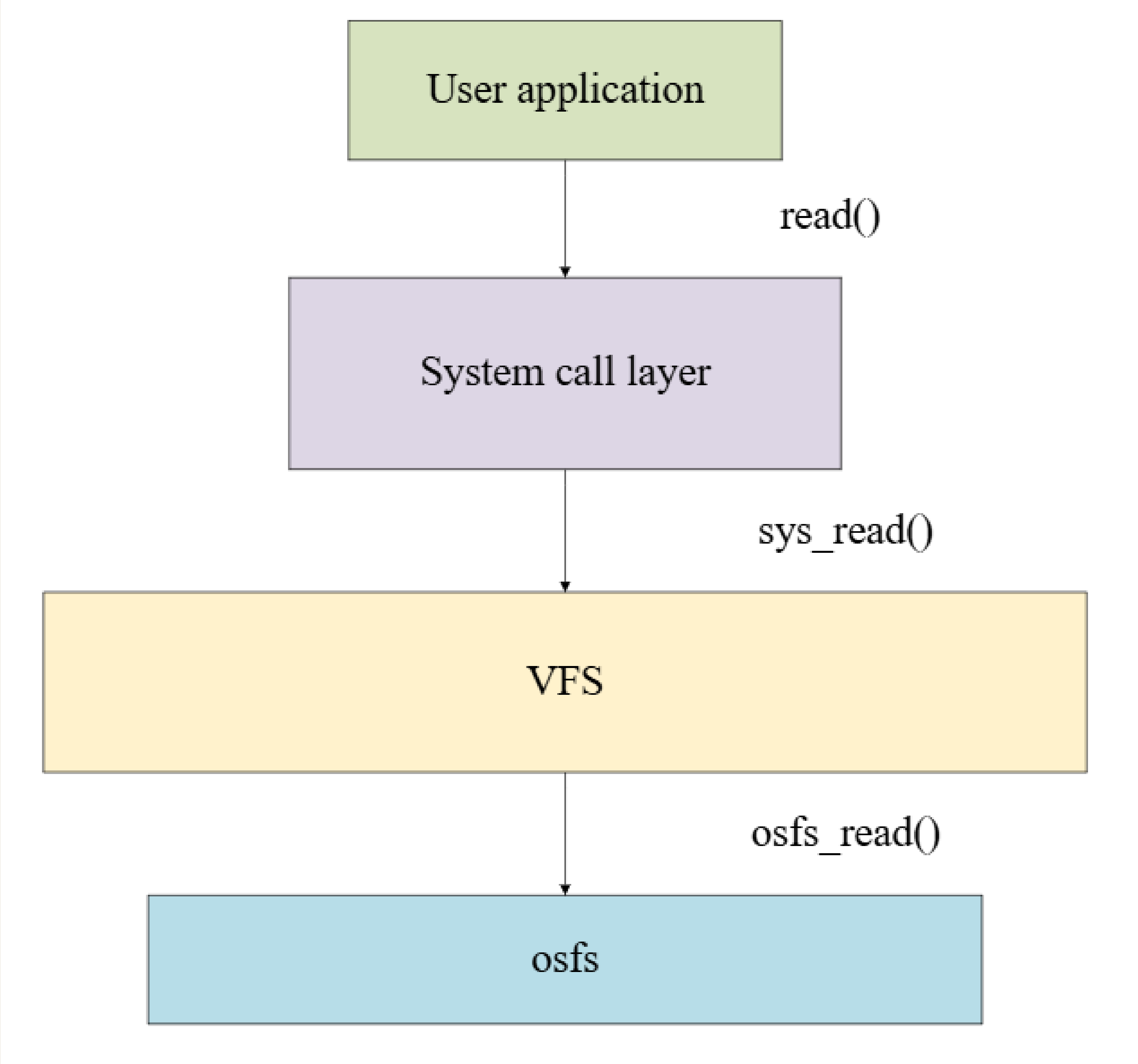
INTRODUCTION TO VFS



The VFS allows multiple file systems to coexist by separating generic functionality from specific implementations.

The file system interface interacts with VFS, which selects and invokes the appropriate file system without needing to know their internal details.

Introduction to VFS



Structure in VFS

- Superblock
- Index node (inode)
- Directory entry (dentry)

Structure in VFS – Superblock

- The superblock **stores the metadata of the filesystem**, describing its overall structure and state.

  - File system type (e.g., ext4, NTFS, etc.)
  - Total and available space
  - Data block size
  - Total number of files and directories

Structure in VFS – inode

- An inode stores the attributes for a file or directory but does not store its name or content.
- Every file or directory has a corresponding inode that stores:
  - File size
  - Permissions (read/write/execute)
  - File type (regular file, directory, etc.)(i_mode)
  - Pointers to the file's data blocks (location of actual file content)

STRUCTURE IN VFS - DENTRY

- A dentry maps file or directory names to their corresponding inodes; it can be seen as an entry in a mapping table.
- Refer to Ch.14 p.16
- Pathname lookup linux kernel document

FILE NAME	INODE NUMBER
NAME1	13579
NAME2	13688

VFS STRUCTURES IN OSFS

- Superblock
- Index node (inode)
- Directory entry (dentry)

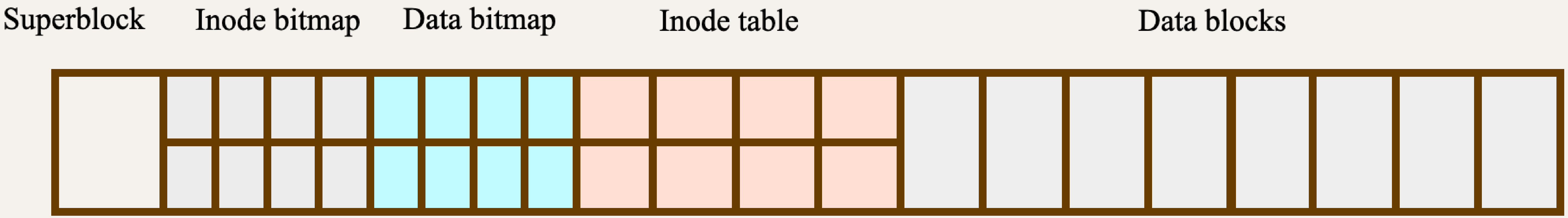
→

- osfs_sb_info
- osfs_inode
- osfs_dir_entry

OUTLINE

1. Introduction to Virtual File System (VFS)
2. Layout of osfs
3. Access Path in File System
4. Requirement & Grading

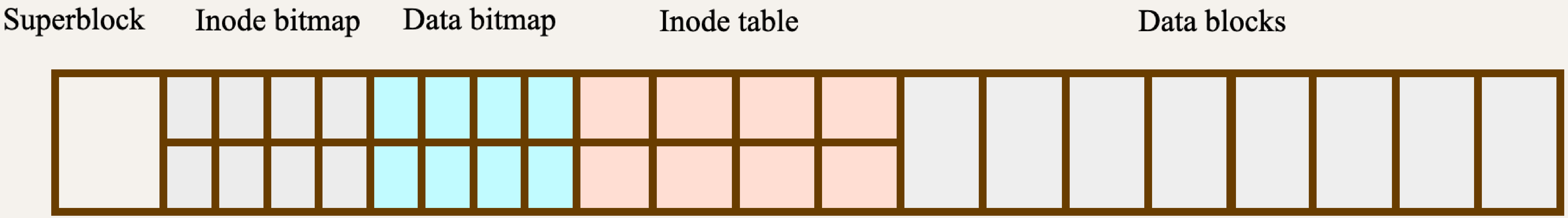
LAYOUT OF OSFS



- Memory-virtualized disk

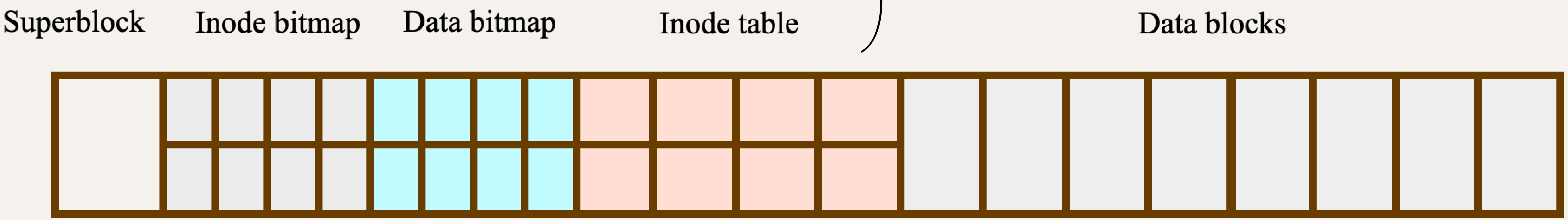
LAYOUT OF OSFS

```c
struct osfs_sb_info {
    uint32_t magic;              // Magic number to identify the filesystem
    uint32_t block_size;         // Size of each data block
    uint32_t inode_count;        // Total number of inodes
    uint32_t block_count;        // Total number of data blocks
    uint32_t nr_free_inodes;     // Number of free inodes
    uint32_t nr_free_blocks;     // Number of free data blocks
    unsigned long *inode_bitmap; // Pointer to the inode bitmap
    unsigned long *block_bitmap; // Pointer to the data block bitmap
    void *inode_table;           // Pointer to the inode table
    void *data_blocks;           // Pointer to the data blocks area
};
```



LAYOUT OF OSFS

```c
struct osfs_inode {
    uint32_t i_ino;                     // Inode number
    uint32_t i_size;                    // File size in bytes
    uint32_t i_blocks;                  // Number of blocks occupied by the file
    uint16_t i_mode;                    // File mode (permissions and type)
    uint16_t i_links_count;             // Number of hard links
    uint32_t i_uid;                     // User ID of owner
    uint32_t i_gid;                     // Group ID of owner
    struct timespec64 __i_atime;        // Last access time
    struct timespec64 __i_mtime;        // Last modification time
    struct timespec64 __i_ctime;        // Creation time
    uint32_t i_block;                   // Simplified handling, single data block pointer
};
```



OUTLINE

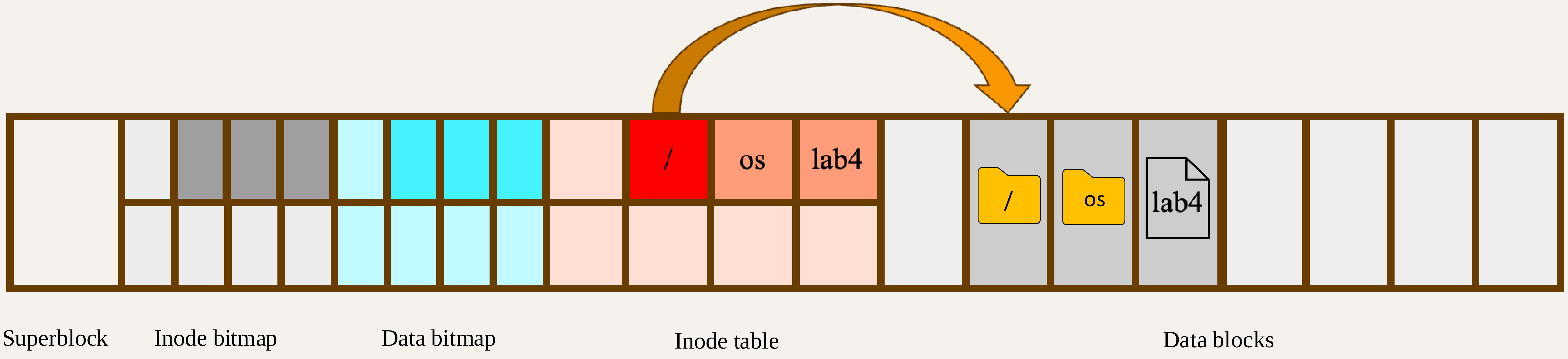
1. Introduction to Virtual File System (VFS)
2. Layout of osfs
3. Access Path in File System
4. Requirement & Grading

ACCESS PATH IN FILE SYSTEM

- We use **read()** as an example
- Assume that we want to read data in **"/os/lab4**"

Access path in file system – read()

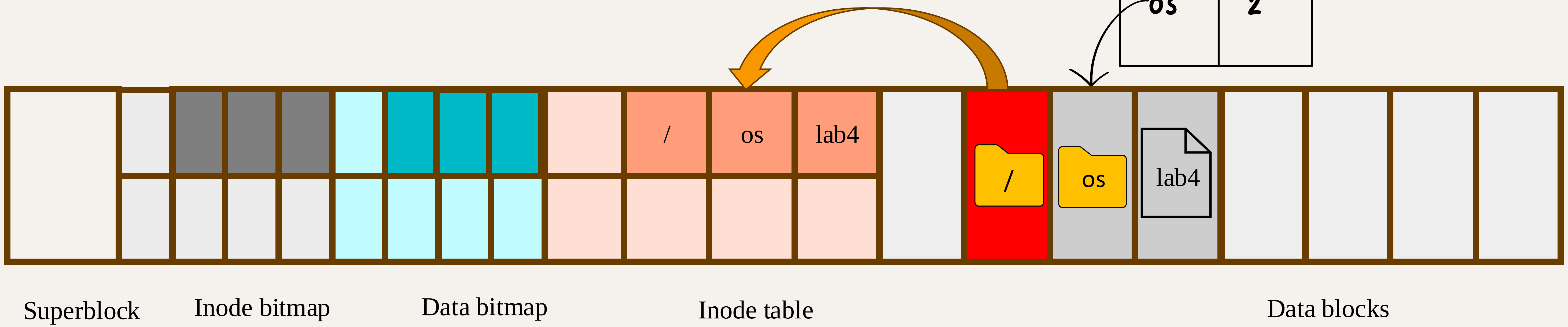
1. Read the **root** inode and locate its data block pointers



ACCESS PATH IN FILE SYSTEM – READ()

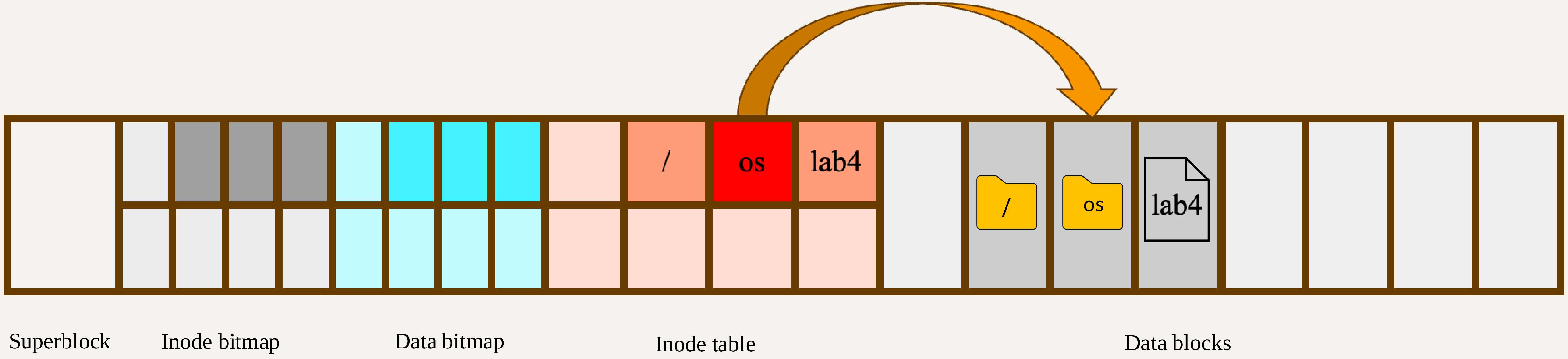
2. Read the **root** data block and locate the **os** entry

NAME	INODE
.	1
OS	2



ACCESS PATH IN FILE SYSTEM – READ()

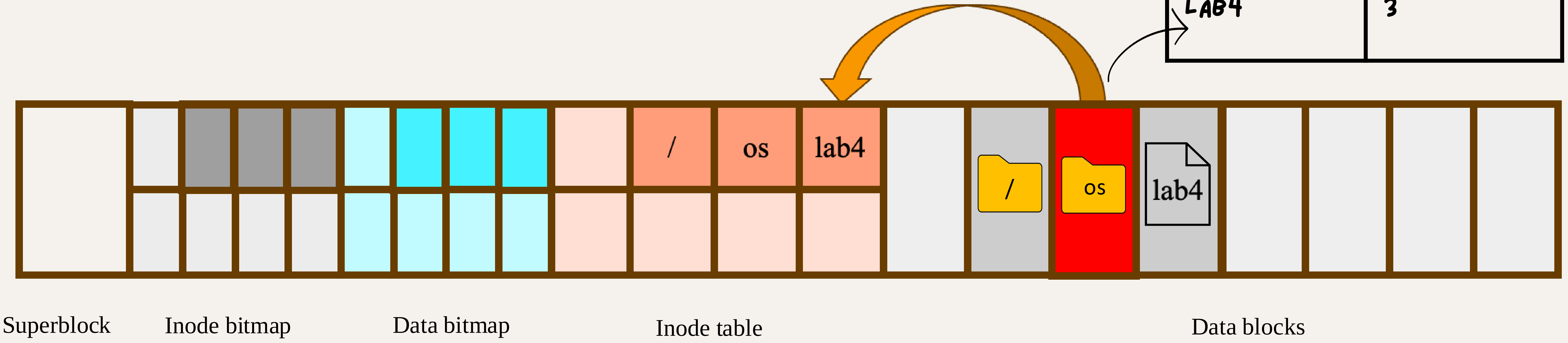
3. Read the **os** inode and locate its data block pointers



ACCESS PATH IN FILE SYSTEM – READ()

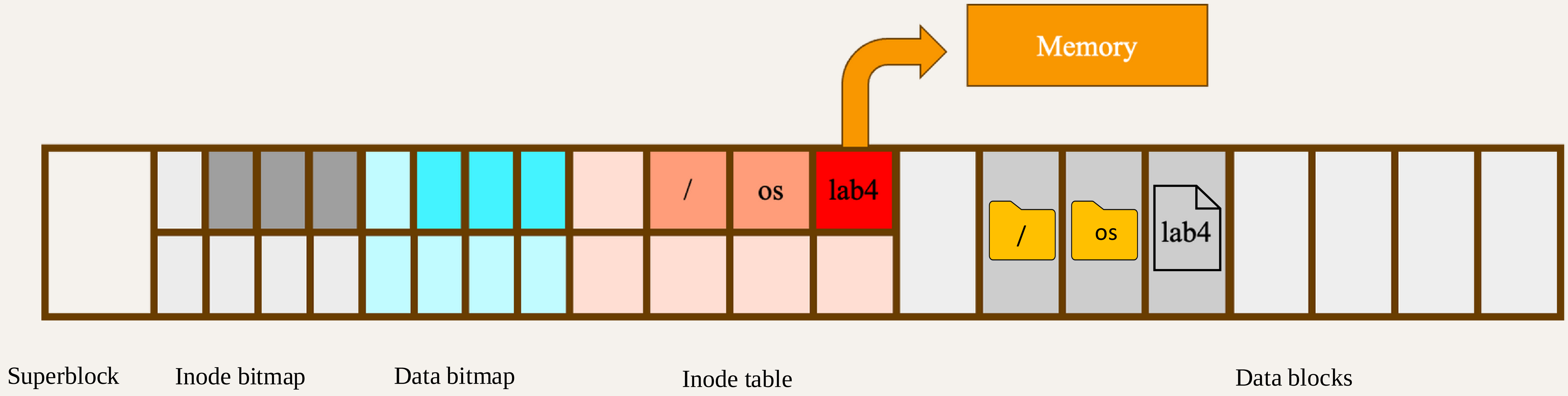
4. Read the **os** data block and locate the **lab4** entry

NAME	INODE
.	2
..	1
LAB4	3



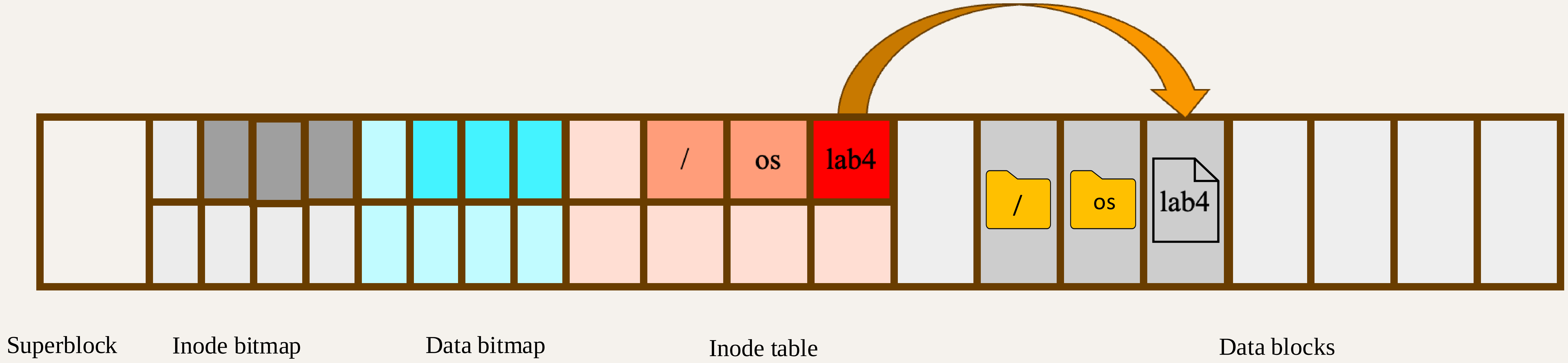
Access Path in File System – read()

5. Read the **lab4** inode into memory



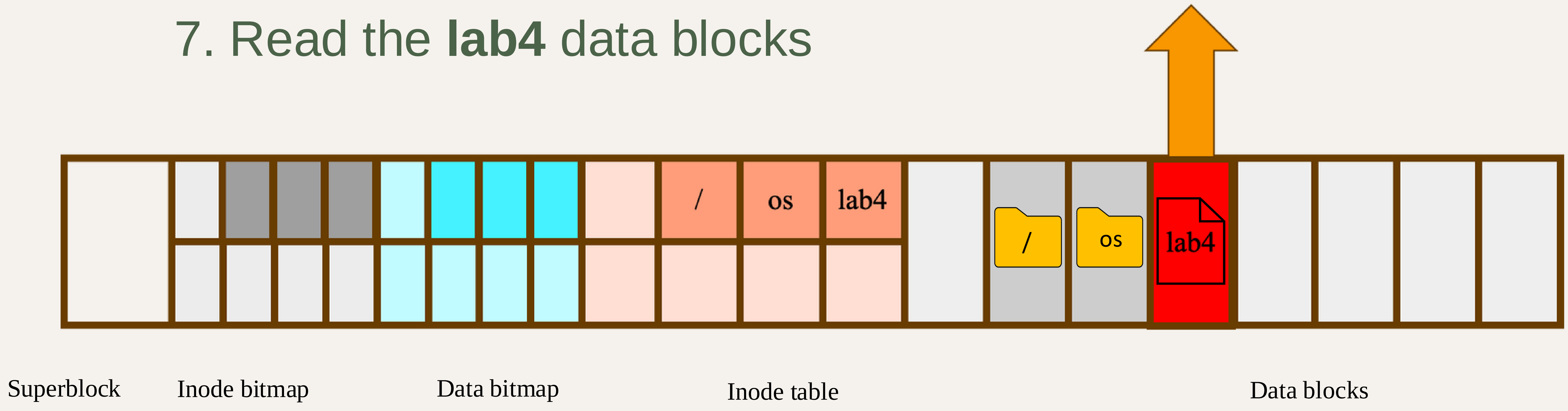
Access Path in File System – read()

6. Read the **lab4** inode and find pointers to its data blocks



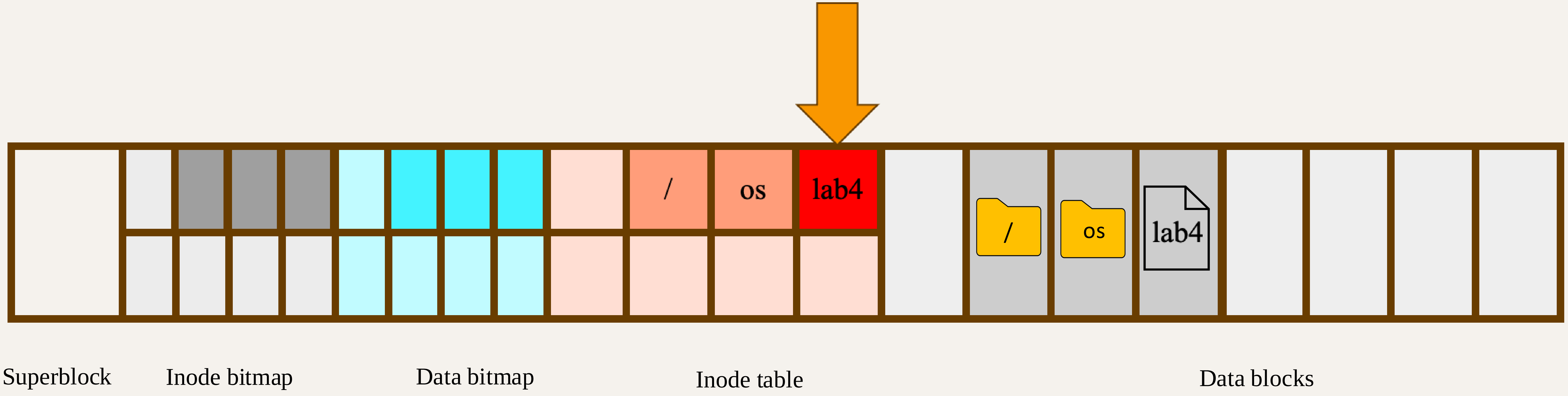
ACCESS PATH IN FILE SYSTEM – READ()

7. Read the **lab4** data blocks



ACCESS PATH IN FILE SYSTEM – READ()

8. Update the **lab4** inode (e.g., last-accessed time)



ACCESS PATH IN FILE SYSTEM – READ() TIMELINE

time

	ROOT INODE	OS INODE	LAB4 INODE	ROOT DATA	OS DATA	LAB4 DATA
OPEN(LAB4)	READ	READ	READ	READ	READ	
READ()			READ WRITE			READ

OUTLINE

1. Introduction to Virtual File System (VFS)
2. Layout of osfs
3. Access Path in File System
4. Requirement & Grading

Requirement

1. Complete **osfs_create** function in dir.c **(7%)**
2. Complete **osfs_write** function in file.c **(3%)**

Bonus

1. Modify the osfs file allocation strategy **(2%)**

e.g. (**extent-based** allocation or **multi-level indexing**)

FUNCTIONS USE IN REQUIREMENT 1 -> CREATE()

`osfs_create()`:

- `osfs_new_inode()`: Creates a new inode within the filesystem
- `osfs_add_dir_entry()`: Adds a new directory entry to a parent directory
- `d_instantiate()`: Associates a dentry with an inode in the VFS layer

HINT FOR REQUIREMENT 1 -> CREATE()

- Step 1: Parse the parent directory passed by the VFS
- Step 2: Validate the file name length
- Step 3: Allocate and initialize a VFS & osfs inode
- Step 4: Parent directory entry update for the new file
- Step 5: Update the parent directory's metadata
- Step 6: Bind the inode to the VFS dentry

FUNCTIONS USE IN REQUIREMENT 2 -> WRITE()

`osfs_write()`:

- `file_inode()`: Retrieves the inode associated with an open file
- `osfs_alloc_data_block()`: Allocates a free data block from the block bitmap
- `copy_from_user()`: Copies data from user space to kernel space

Hint for Requirement 2 -> write()

- Step 1: Retrieve the inode and filesystem information
- Step 2: Check and allocate a data block if necessary
- Step 3: Limit the write length to fit within one data block
- Step 4: Write data from user space to the data block
- Step 5: Update inode & osfs_inode attribute
- Step 6: Return the number of bytes written

Hint for Bonus

Data structure might be modified :

```
osfs_inode -> i_block
osfs_inode -> i_blocks
osfs_inode -> i_size
```

Functions might be modified:

All functions !!

How to Test & Expected Output

1. After finishing the code, build the kernel module by running `'make'`.
2. Use `'ls'` to verify what files have been built

```
vboxuser@oslab:~/oslabfs$ ls
dir.c  Dockerfile  file.c  inode.c  Makefile       Module.symvers  osfs_init.c  osfs.ko   osfs.mod.c  osfs.o   super.o
dir.o  exec.sh     file.o  inode.o  modules.order  osfs.h          osfs_init.o  osfs.mod  osfs.mod.o  super.c
```

3. Insert the module into the Linux kernel: `'sudo insmod osfs.ko'`
4. Run `'sudo dmesg'` to check if the module is inserted correctly.

```
[  637.738846] osfs: Successfully registered
```

5. Create a directory to serve as the mount point. In this example, the directory is named `'mnt'`.

```
vboxuser@oslab:~/oslabfs$ ls
dir.c  Dockerfile  file.c  inode.c  Makefile  modules.order   osfs.h       osfs_init.o  osfs.mod    osfs.mod.o  super.c
dir.o  exec.sh     file.o  inode.o  mnt       Module.symvers  osfs_init.c  osfs.ko      osfs.mod.c  osfs.o      super.o
```

6. Mount the file system: `'sudo mount -t osfs none mnt/'`
7. Run `'sudo dmesg'` to check if the mount operation was successful.

```
[ 1026.834004] osfs: Filling super start
[ 1026.834019] osfs_get_osfs_inode: Getting inode 1
[ 1026.834021] osfs: Superblock filled successfully with root inode 1
```

How to test & expected output

Now you are ready to test your write and create function
Enter the `'mnt'` directory

1. Use `'sudo touch test1.txt'` to test the create function. We expect that after running the command, a file named `'test1.txt'` will appear in the `'mnt'` directory.

```
vboxuser@oslab:~/oslabfs/mnt$ sudo touch test1.txt
vboxuser@oslab:~/oslabfs/mnt$ ls
test1.txt
```

2. Use `'sudo bash -c "echo 'I LOVE OSLAB' > test1.txt"'` to test the write function. Then, run `'cat test1.txt'.` The expected output is `'I LOVE OSLAB'.`

```
vboxuser@oslab:~/oslabfs/mnt$ sudo bash -c "echo 'I LOVE OSLAB' > test1.txt"
vboxuser@oslab:~/oslabfs/mnt$ ls
test1.txt
vboxuser@oslab:~/oslabfs/mnt$ cat test1.txt
I LOVE OSLAB
```


Thank you!