IOWA STATE UNIVERSITY
Department of Electrical and Computer Engineering

Lecture 31:
File System Implementation

Agenda

- Recap
- File System Implementation
  - Make & Mount FS
  - Developing a basic FS

Recap

- File
  - A linear array of bytes
  - A **file system** (FS) is responsible for managing and storing files persistently on disk
    - data structures
    - implementations of file operations
  - Each file has a unique, low-level name called **inode number** in the file system
- Directory
  - contains a list of (user-readable name, low-level name) pairs.
    - each entry refers to either *files* or other *directories*

Recap

- Creating file

```
int fd = open("foo", O_CREAT | O_WRONLY | O_TRUNC);
```

- Reading & writing file

```
prompt> strace cat foo
…
open("foo", O_RDONLY|O_LARGEFILE)  = 3
read(3, "hello\n", 4096)           = 6
write(1, "hello\n", 6)             = 6 // file descriptor 1: standard out
hello
read(3, "", 4096)                  = 0 // 0: no bytes left in the file
close(3)                           = 0
…
prompt>
```

Recap

- Change file offset

```
off_t lseek(int fildes, off_t offset, int whence);
```

- Sync file

```
off_t fsync(int fd)
```

- Rename file

```
rename(char* old, char *new)
```

- Remove file

```
prompt> strace rm foo
…
unlink("foo")                = 0      // return 0 upon success
…
```

Recap

- Make a directory
  - `mkdir()`

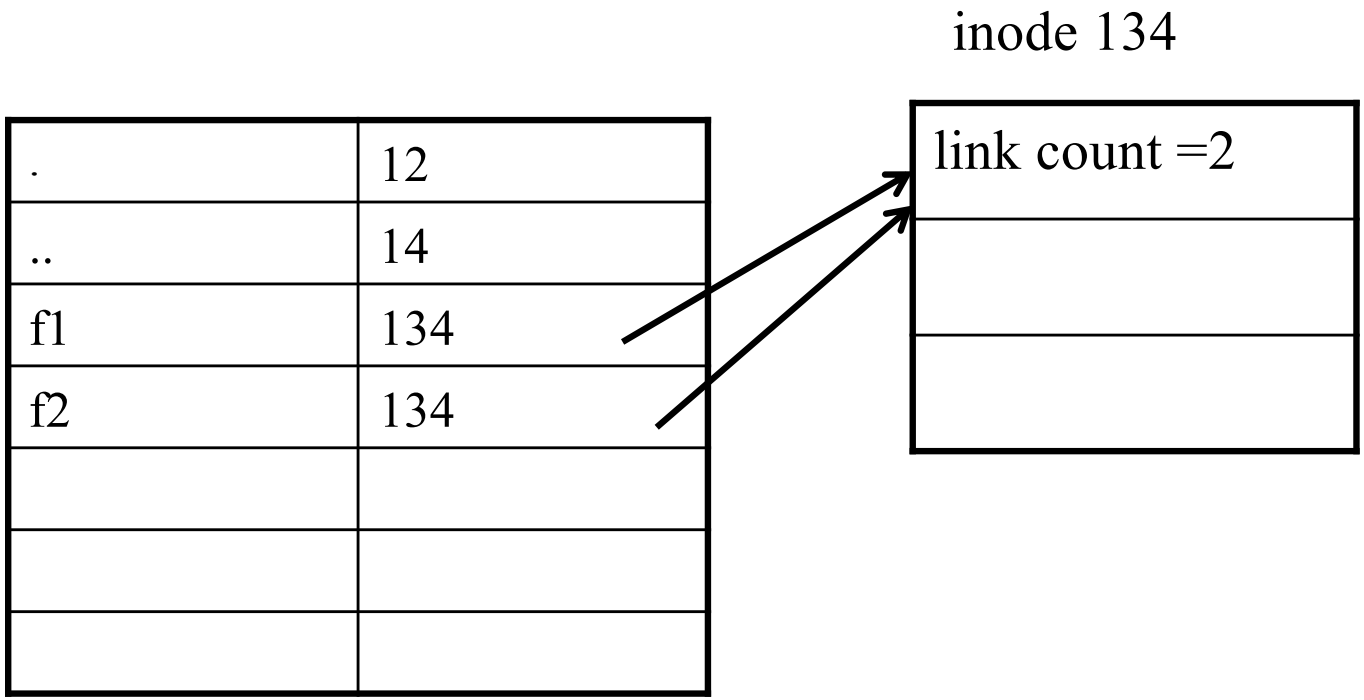
```
prompt> strace mkdir foo
…
mkdir("foo", 0777)                    = 0
prompt>
```

  - Each directory have two default entries
    - . (itself), .. (parent)

```
prompt> ls -a
./       ../
prompt> ls -al
total 8
drwxr-x---  2 remzi remzi    6 Apr 30 16:17 ./
drwxr-x--- 26 remzi remzi 4096 Apr 30 16:17 ../
```

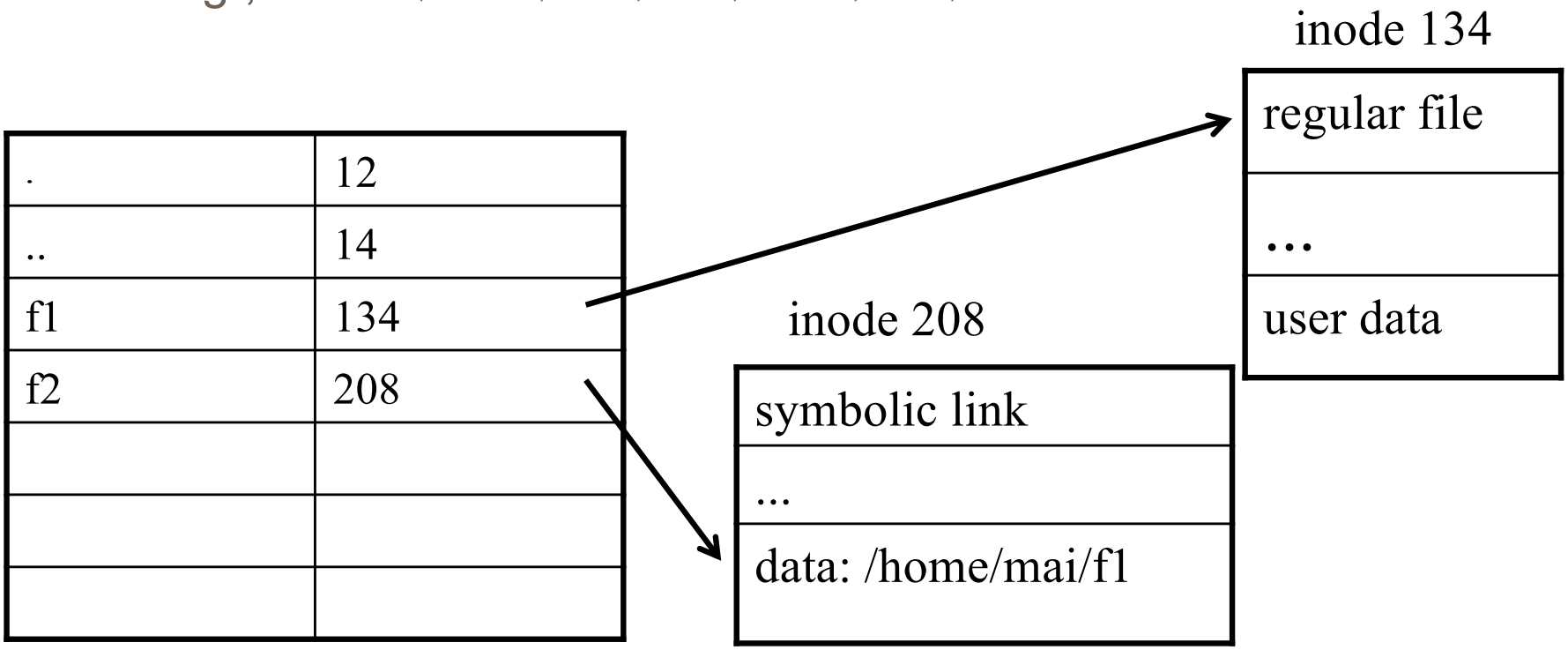
Recap

- Hard link
  - Both files map to the same inode
    - e.g., `ln /home/mai/f1 /home/mai/f2`



Recap

- Symbolic link
  - A symbolic link has its own inode number
    - e.g., `ln -s /home/mai/f1 /home/mai/f2`



Agenda

- ~~Recap~~
- File System Implementation
  - Make & Mount FS
  - Developing a basic FS

Create an FS

- `mkfs` tool : Make a file system
  - Write an empty file system, starting with *a root directory*, on to a disk partition.
  - Input:
    - A device (such as a disk partition, e.g., `/dev/sda1`)
    - A file system type (e.g., `ext3`)

Mount an FS

- `mount()`
  - Take an existing directory as a target **mount point**.
  - Essentially paste a new file system onto the directory tree at that point
    - E.g., the pathname `/home/users/` refers to the root of the newly-mounted directory.

```
prompt> mount –t ext3 /dev/sda1 /home/users
prompt> ls /home/users
a b
```

Agenda

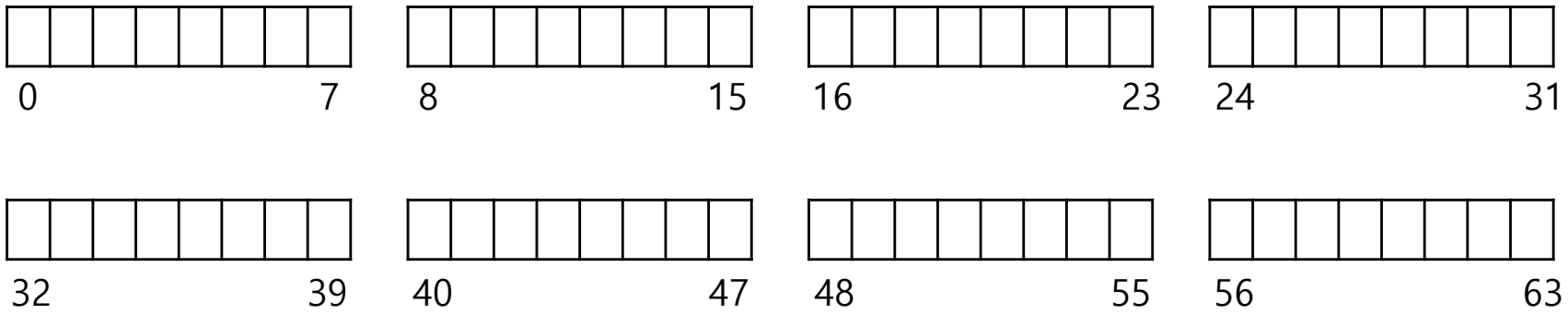
- ~~Recap~~
- File System Implementation
  - ~~Make & Mount FS~~
  - Developing an example FS

The Way to Think

- Two different perspectives
  - **Data structures**
    - What types of on-disk structures are utilized by the file system to organize its data and metadata
  - **Access methods**
    - How does it map the calls made by a process
      - e.g., `open()`, `read()`, `write()`
    - Which structures are used during the execution of a particular system call

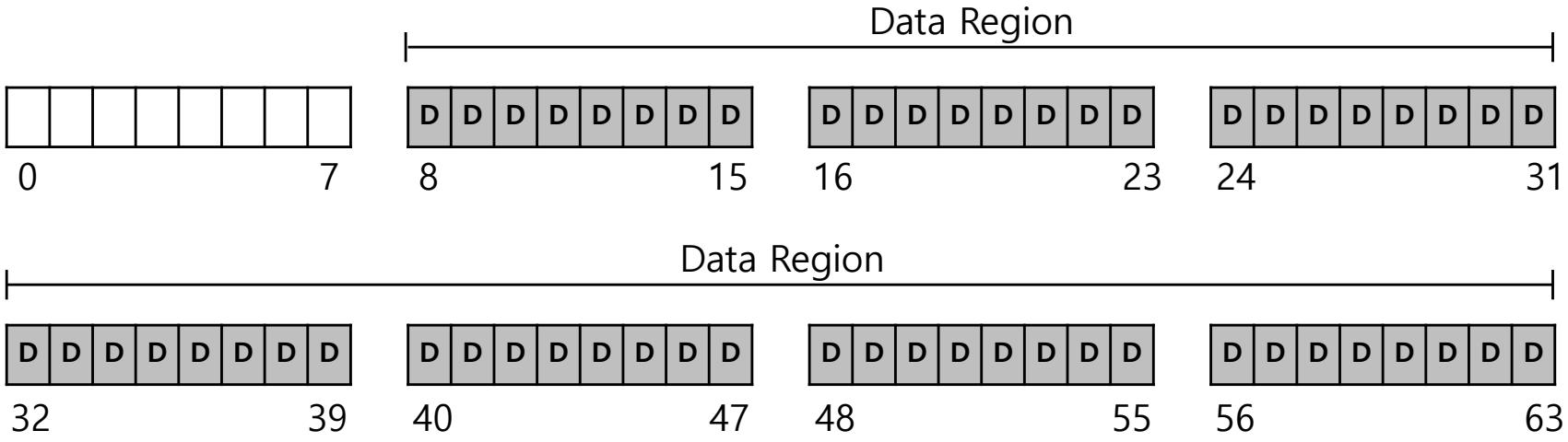
Overall Organization

- Divide the raw disk into **blocks**
  - Minimum management unit on the disk
    - e.g., Block size is 4 KB
  - The blocks are addressed from `0 to N -1.`



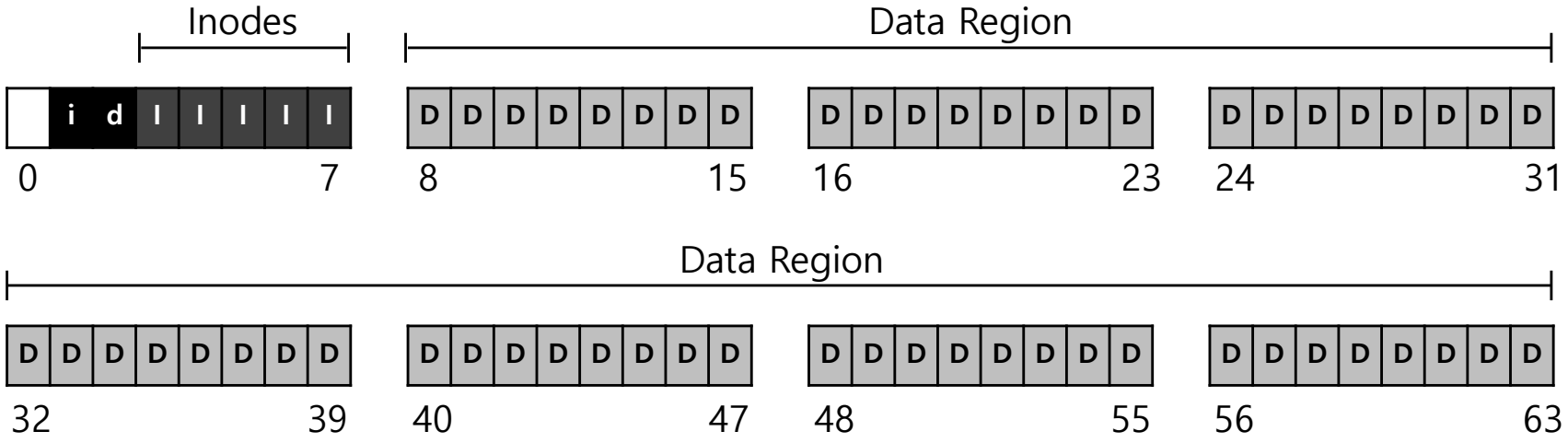
Data Region

- Reserve data region to store user data
  - Minimum management unit on disk
    - e.g., 4KB block size
  - The blocks are addressed from `0 to N -1.`
    - e.g., 0 - 63 block



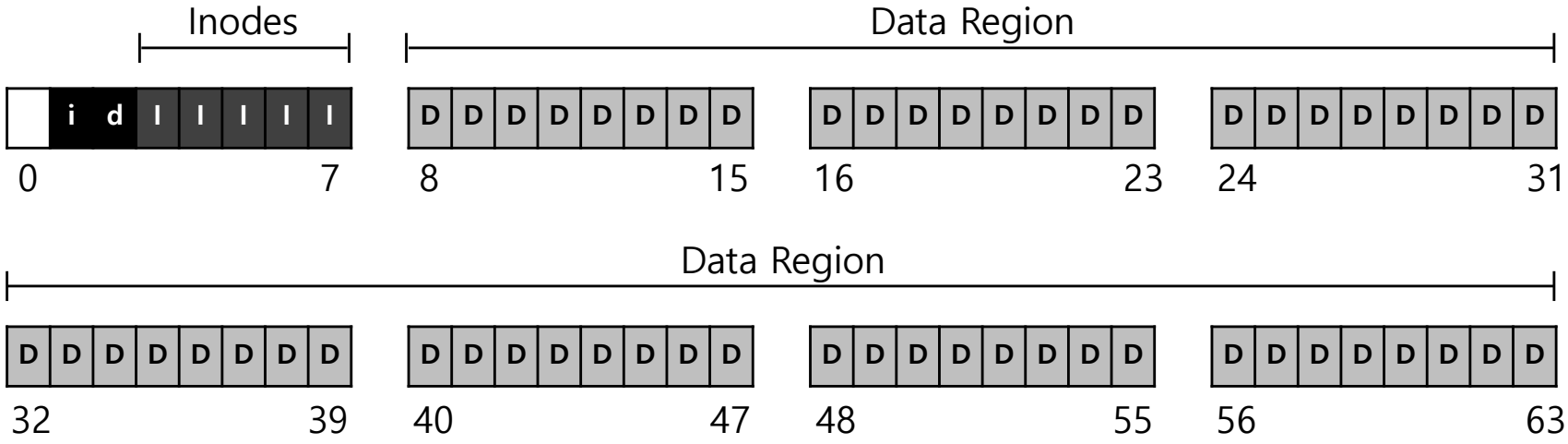
Metadata Region

- Need metadata to track which data blocks comprise a file, the size of the file, its owner, etc.
  - Maintain an **inode** data structure for each file
  - Store all inodes in an **inode table**
    - e.g., inode tables : 3 ~ 7, inode size : 256 bytes
      - 4-KB block can hold 16 inodes
      - The FS contains 80 inodes



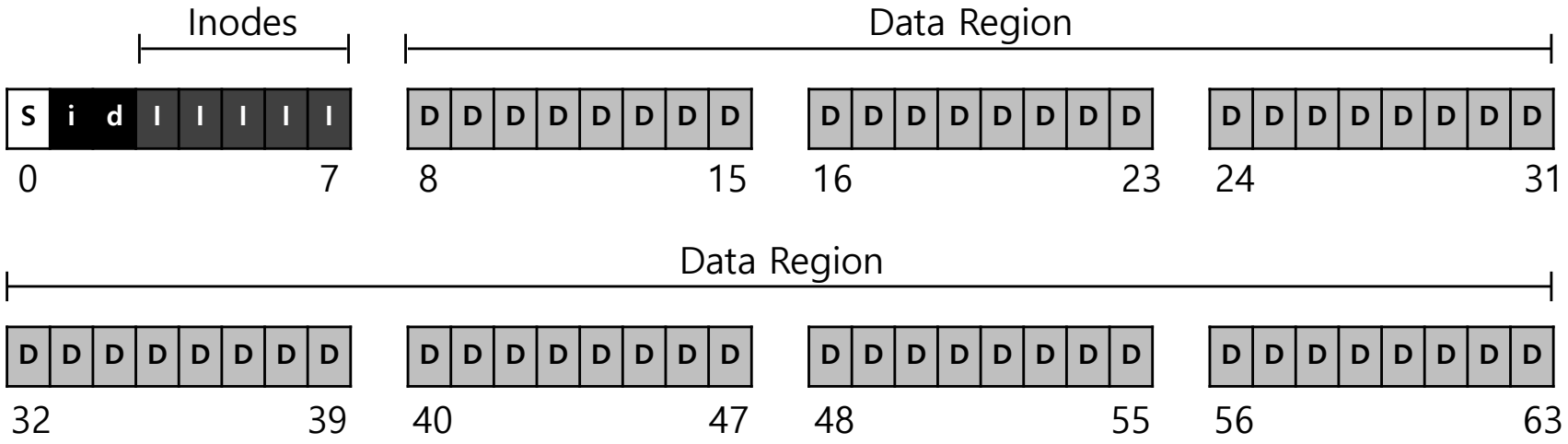
Allocation Structure

- Track whether inodes or data blocks are free or used
- A common structure is **bitmap**
  - each bit indicates free(0) or in-use(1)
  - **data bitmap**: for data region (d)
  - **inode bitmap**: for inode table (i)



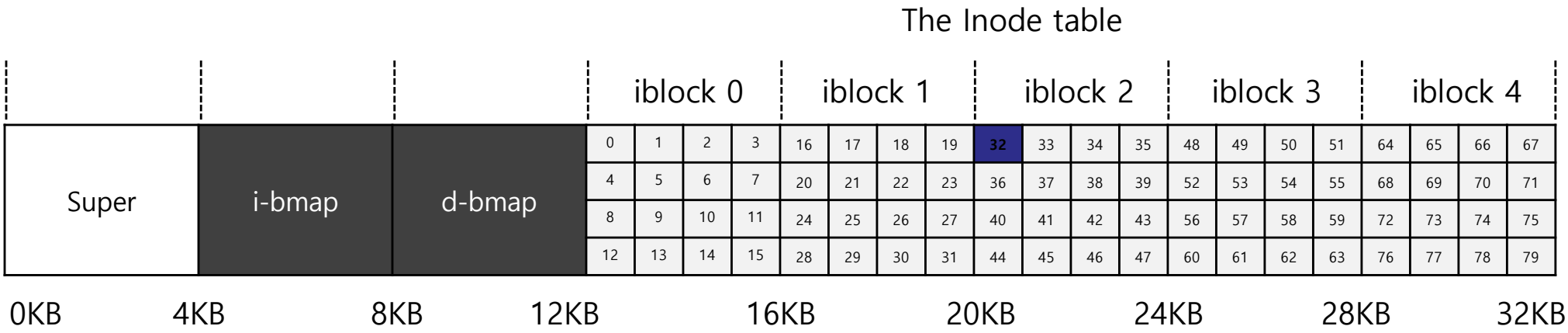
Superblock

- contains the overall information for an FS
  - e.g., the number of inodes, begin location of inode table, etc
  - when mounting an FS, OS will read the superblock first, to initialize various structures in memory



File Organization: inode

- Each inode is referred to by inode number.
  - FS calculates where the inode is on the disk based on the inode number
  - e.g., inode number: 32
    - Calculate the offset into the inode region (32 x sizeof(inode))= 32 x 256 = 8192 = 8K
    - Add to start address of the inode table: 12KB + 8KB = 20 KB



File Organization: inode

- `inode` contains all information about a file
  - File type
    - regular file, directory, symbolic link, etc
  - Size
    - In bytes
    - In blocks: the number of data blocks allocated to it
  - Addresses of the data blocks belonging to the file
  - Protection information
    - who owns the file, who can access, etc
  - Time information
  - ...

File Organization: inode

- E.g., a simplified Ext2 inode

Size	Name	What is this inode field for?
2	mode	can this file be read/written/executed?
2	uid	who owns this file?
4	size	how many bytes are in this file?
4	time	what time was this file last accessed?
4	ctime	what time was this file created?
4	mtime	what time was this file last modified?
4	dtime	what time was this inode deleted?
4	gid	which group does this file belong to?
2	links_count	how many hard links are there to this file?
2	blocks	how many blocks have been allocated to this file?
4	flags	how should ext2 use this inode?
4	osd1	an OS-dependent field
60	block	a set of disk pointers (15 total)
4	generation	file version (used by NFS)
4	file_acl	a new permissions model beyond mode bits
4	dir_acl	called access control lists
...	...	...

Multi-Level Index

- inode contains pointers of data blocks of the file
  - point to the location of the block on disk
- inode have fixed number of **direct pointers** (e.g., 12) and a **single indirect pointer**
  - a direct pointer points to a data block directly
  - a single indirect pointer points to a block that contains direct pointers
  - If a file grows large enough, an indirect block is allocated; the indirect pointer field of the inode is set to point to the allocated indirect block
  - Max file size: $(12 + 1024)$ x 4 KB or 4144 KB

Multi-Level Index

- **Double indirect pointer** points to a block that contains single indirect blocks.
- **Triple indirect pointer** points to a block that contains double indirect blocks.
- E.g., 12 direct pointers, a single and a double indirect block.
  - over 4GB in size (12+1024+ 1024^2) x 4KB
- Many file system use a multi-level index.
  - e.g.. Linux Ext2/3, NetApp's WAFL
    - Ext4 use extents instead of simple pointers

Agenda

- ~~Recap~~
- ~~File System Implementation~~
  - ~~Make & Mount FS~~
  - ~~Developing a basic FS~~

Questions?

*acknowledgement: slides include content from “Modern Operating Systems” by A. Tanenbaum, “Operating Systems Concepts” by A. Silberschatz etc., “Operating Systems: Three Easy Pieces” by R. Arpaci-Dusseau etc., and anonymous pictures from internet.