# Operating Systems

Youjip Won

# 40. Filesystem Implementation

# Overview

- In this chapter, we study very simple file system (vsfs)
  - Basic on-disk structures, access methods, and various policies of vsfs
- We will study…
  - How can we build a simple file system?
  - What structures are needed on the disk?
  - What do they need to track?
  - How are they accessed?

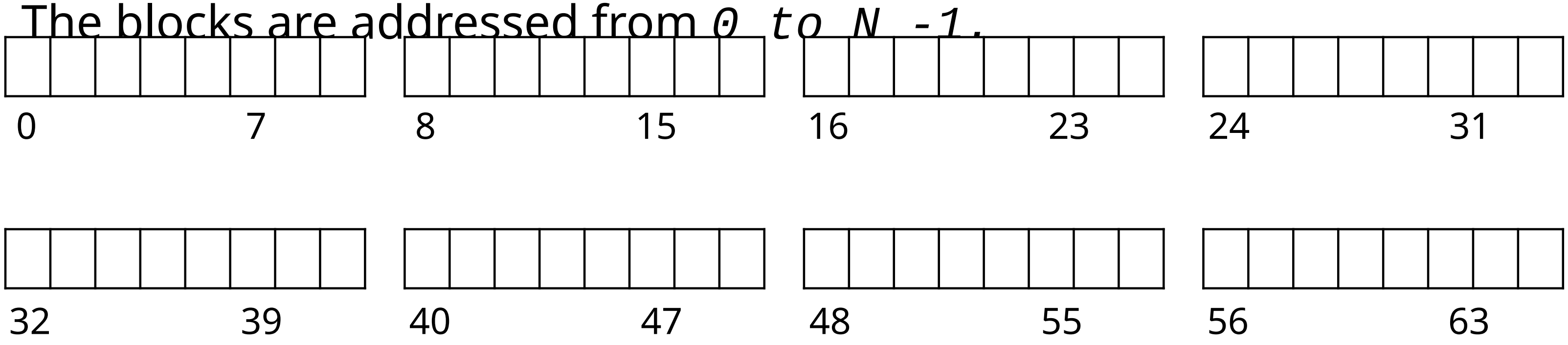
# File system Implementation

- What types of data structures are utilized by the file system?
- How file system organize its data and metadata?
- Understand access methods of a file system.
  - `open()`, `read()`,`write()`, etc.

# Overall Organization

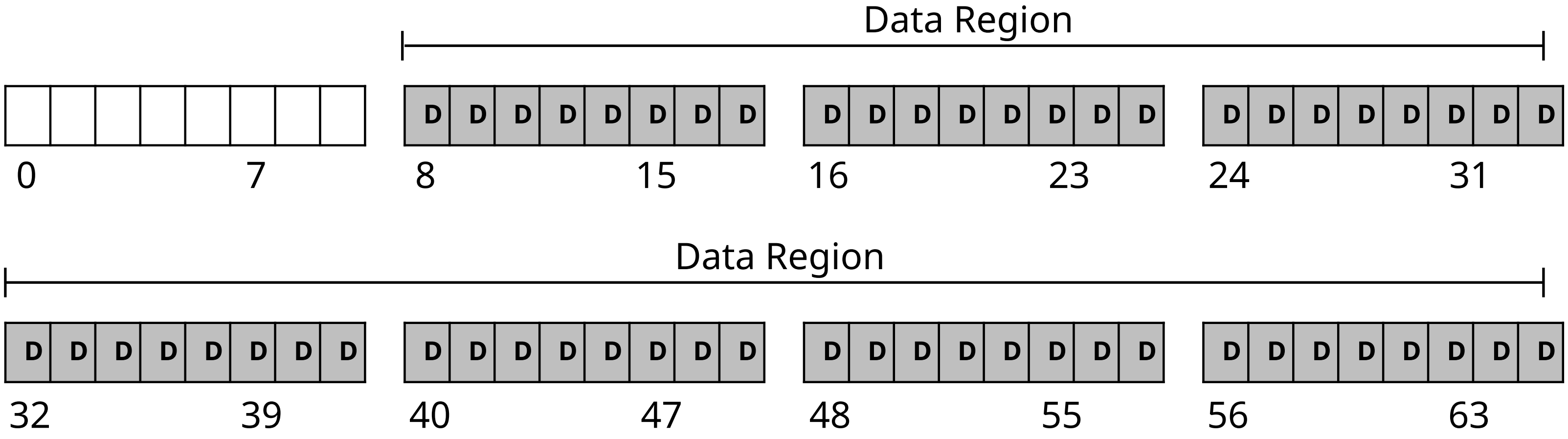
- Let’s develop the overall organization of the file system data structure.

- Divide the disk into blocks.
  - Block size is 4 KB.
  - The blocks are addressed from `0 to N -1`.



# Data region in file system

- Reserve data region to store user data

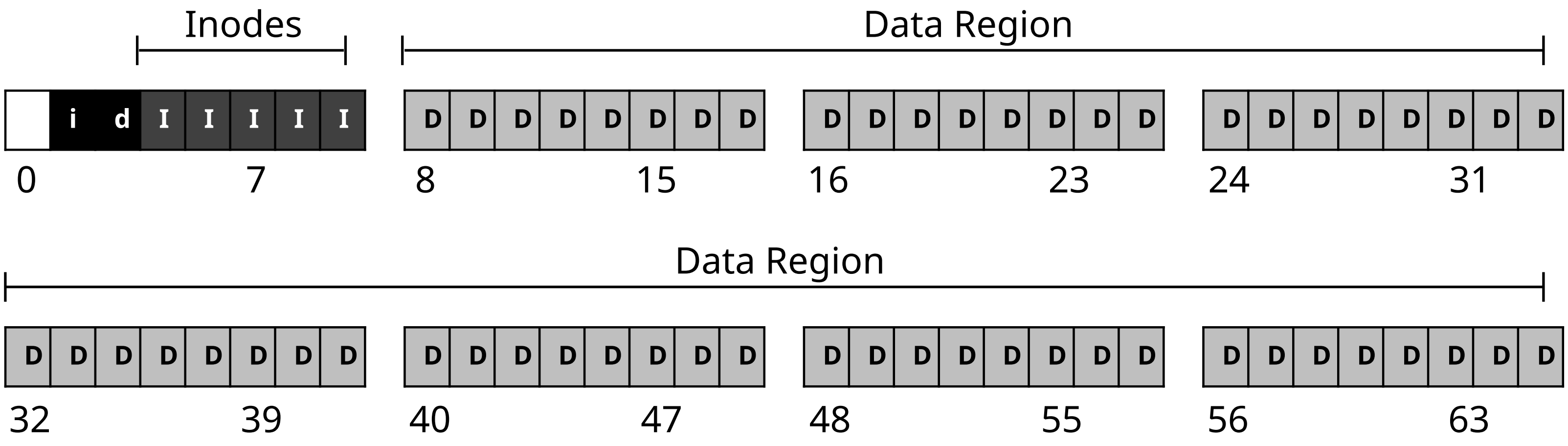


- File system has to track which data block comprise a file, the size of the file, its owner, etc.

**How we store these inodes in file system?**

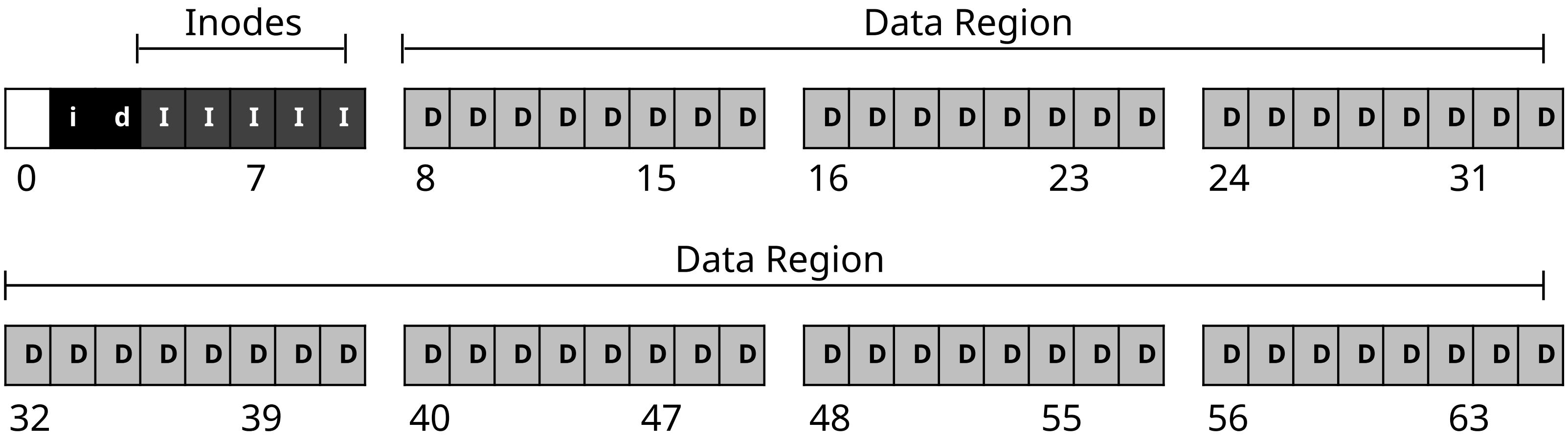
# Inode table in file system

- Reserve some space for inode table
  - This holds an array of on-disk inodes.
  - Ex) inode tables : 3 ~ 7, inode size : 256 bytes
    - 4-KB block can hold 16 inodes.
    - The file system contains 80 inodes. (maximum number of files)



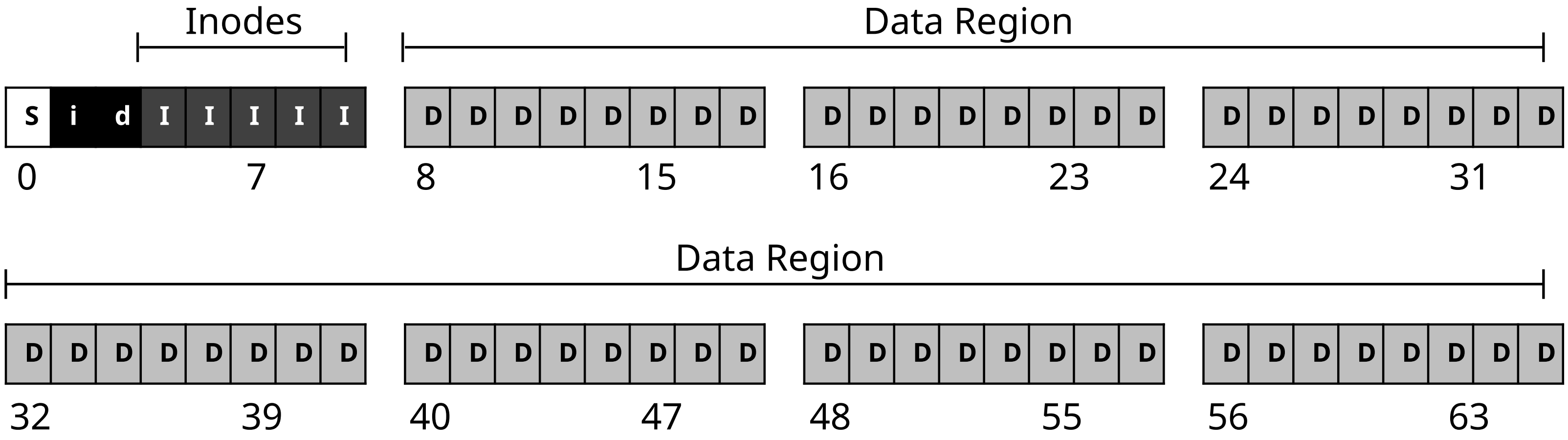
# allocation structures

- This is to track whether inodes or data blocks are free or allocated.
- Use bitmap, each bit indicates free(0) or in-use(1)
  - data bitmap: for data region for data region
  - inode bitmap: for inode table



# super block

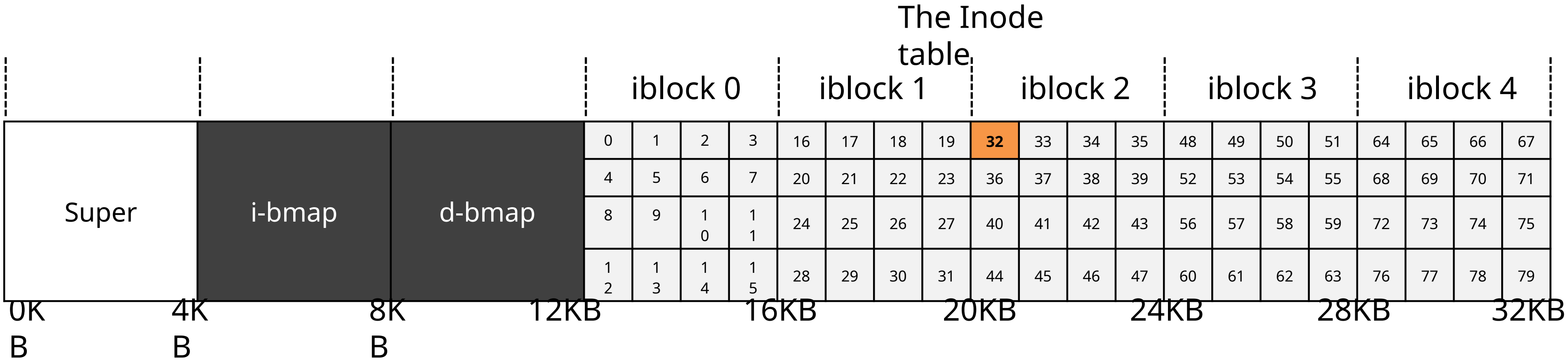
- Super block contains this information for particular file system
  - Ex) The number of inodes, begin location of inode table. etc



- Thus, when mounting a file system, OS will read the superblock first, to initialize various information.

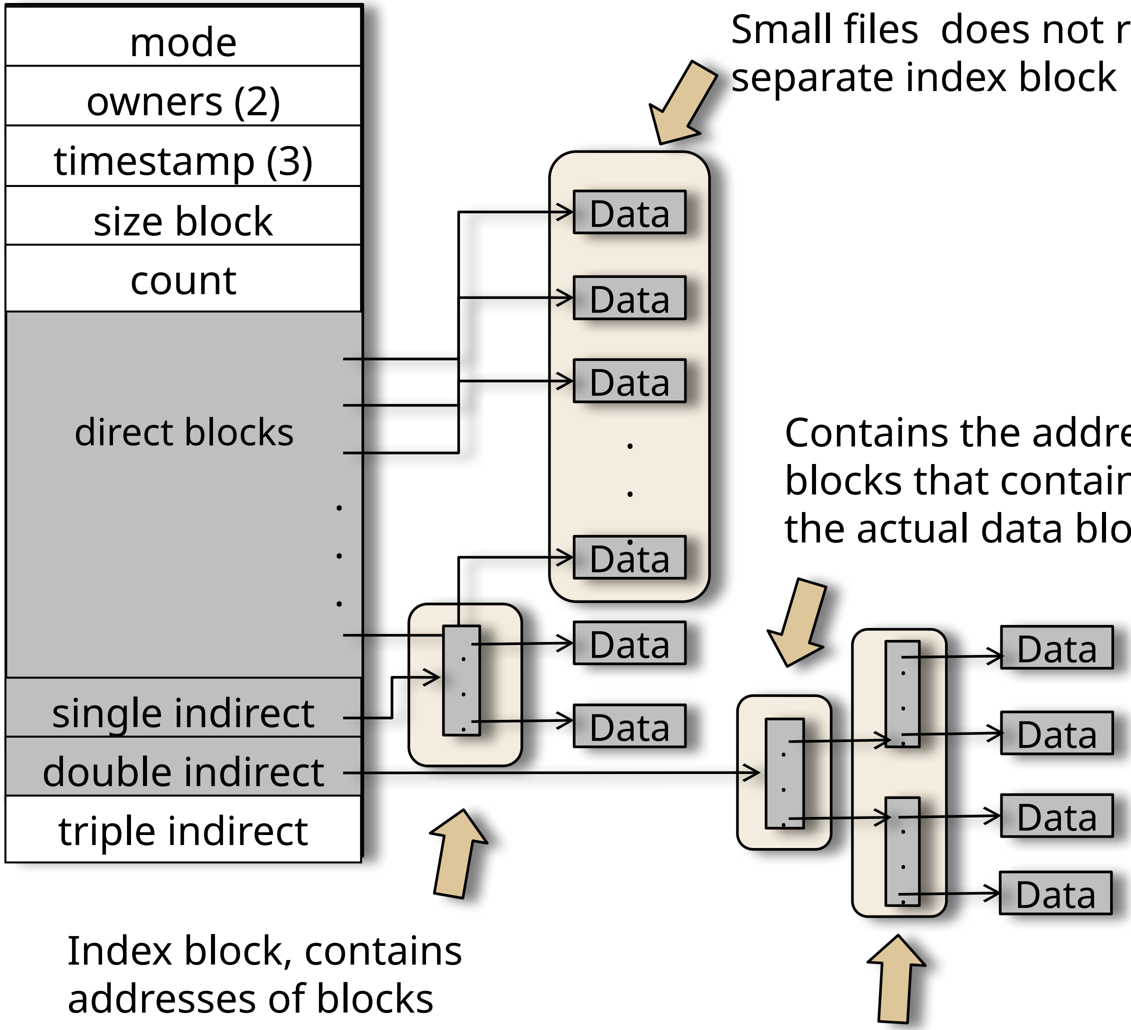
# File Organization: The inode

- Each inode is referred to by inode number.
  - by inode number, File system calculate where the inode is on the disk.
  - Ex) inode number: 32
    - Calculate the offset into the inode region (32 x sizeof(inode) (256 bytes) = 8192
    - Add start address of the inode table(12 KB) + inode region(8 KB) = 20 KB



| iblock 0 | | | | iblock 1 | | | | iblock 2 | | | | iblock 3 | | | | iblock 4 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 16 | 17 | 18 | 19 | **32** | 33 | 34 | 35 | 48 | 49 | 50 | 51 | 64 | 65 | 66 | 67 |
| 4 | 5 | 6 | 7 | 20 | 21 | 22 | 23 | 36 | 37 | 38 | 39 | 52 | 53 | 54 | 55 | 68 | 69 | 70 | 71 |
| 8 | 9 | 10 | 11 | 24 | 25 | 26 | 27 | 40 | 41 | 42 | 43 | 56 | 57 | 58 | 59 | 72 | 73 | 74 | 75 |
| 12 | 13 | 14 | 15 | 28 | 29 | 30 | 31 | 44 | 45 | 46 | 47 | 60 | 61 | 62 | 63 | 76 | 77 | 78 | 79 |

# File Structure: Indexed Allocation



Small files does not require separate index block

Contains the addresses of blocks that contain pointers to the actual data blocks

Index block, contains addresses of blocks that contains data

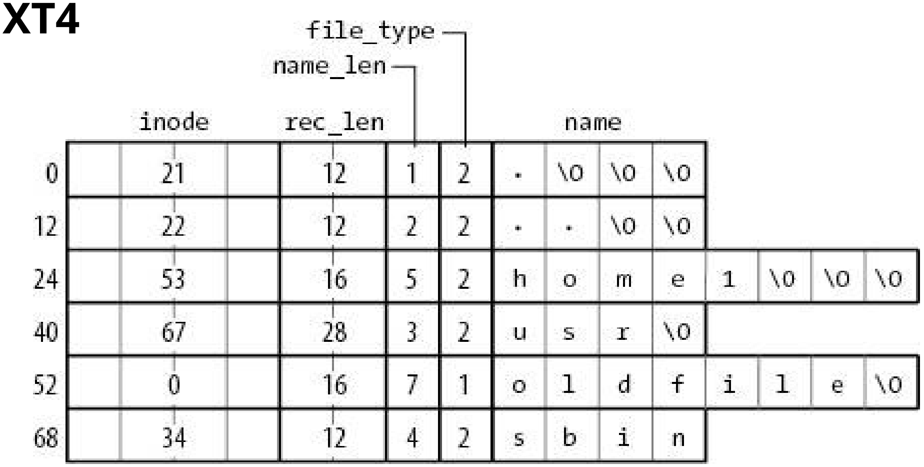
Pointers to the actual data blocks

# Directory Structure

## VSFS

| inum | reclen | strlen | name |
|---|---|---|---|
| 5 | 4 | 2 | . |
| 2 | 4 | 3 | .. |
| 12 | 4 | 4 | foo |
| 13 | 4 | 4 | bar |
| 24 | 8 | 7 | foobar |

## EXT4



|  | inode | rec_len | name_len | file_type | name |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 21 | 12 | 1 | 2 | . | \0 | \0 | \0 |  |  |  |  |
| 12 | 22 | 12 | 2 | 2 | . | . | \0 | \0 |  |  |  |  |
| 24 | 53 | 16 | 5 | 2 | h | o | m | e | 1 | \0 | \0 | \0 |
| 40 | 67 | 28 | 3 | 2 | u | s | r | \0 |  |  |  |  |
| 52 | 0 | 16 | 7 | 1 | o | l | d | f | i | l | e | \0 |
| 68 | 34 | 12 | 4 | 2 | s | b | i | n |  |  |  |  |

# File Read

| | data bitmap | inode bitmap | root inode | foo inode | bar inode | root data | foo data | bar data[0] | bar data[1] | bar data[2] |
|---|---|---|---|---|---|---|---|---|---|---|
| open(bar) | | | read | | | | | | | |
| | | | | | | read | | | | |
| | | | | read | | | | | | |
| | | | | | | | read | | | |
| | | | | | read | | | | | |
| read() | | | | | read | | | | | |
| | | | | | | | | read | | |
| | | | | | write | | | | | |
| read() | | | | | read | | | | | |
| | | | | | | | | | read | |
| | | | | | write | | | | | |
| read() | | | | | read | | | | | |
| | | | | | | | | | | read |
| | | | | | write | | | | | |

# File Creation

| | data bitmap | inode bitmap | root inode | foo inode | bar inode | root data | foo data | bar data[0] | bar data[1] | bar data[2] |
|---|---|---|---|---|---|---|---|---|---|---|
| create (/foo/bar) | | | read | | | | | | | |
| | | | | | | read | | | | |
| | | | | read | | | | | | |
| | | | | | | | read | | | |
| | | read | | | | | | | | |
| | | write | | | | | | | | |
| | | | | | | | write | | | |
| | | | | | read | | | | | |
| | | | | | write | | | | | |
| | | | | write | | | | | | |
| write() | | | | | read | | | | | |
| | read | | | | | | | | | |
| | write | | | | | | | | | |
| | | | | | | | | write | | |
| | | | | | write | | | | | |

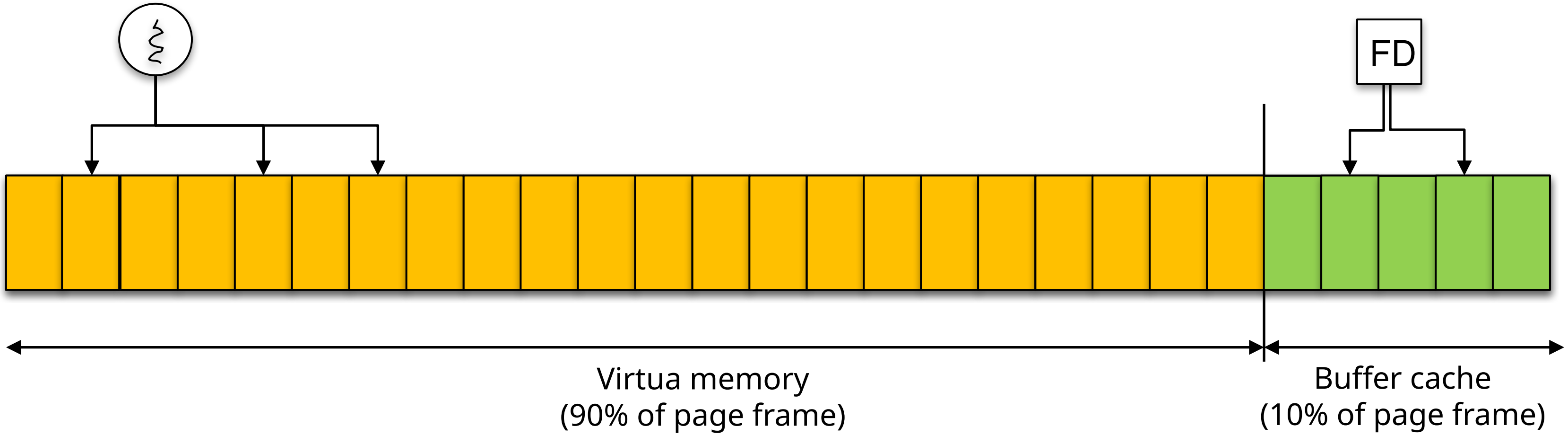
…

# File Creation (Cont.)

| | data bitmap | inode bitmap | root inode | foo inode | bar inode | root data | foo data | bar data[0] | bar data[1] | bar data[2] |
|---|---|---|---|---|---|---|---|---|---|---|
| … | | | | | | | | | | |
| | | | | | read | | | | | |
| | read | | | | | | | | | |
| write() | write | | | | | | | | | |
| | | | | | | | | | write | |
| | | | | | write | | | | | |
| | | | | | read | | | | | |
| | read | | | | | | | | | |
| write() | write | | | | | | | | | |
| | | | | | | | | | write | |
| | | | | | write | | | | | |

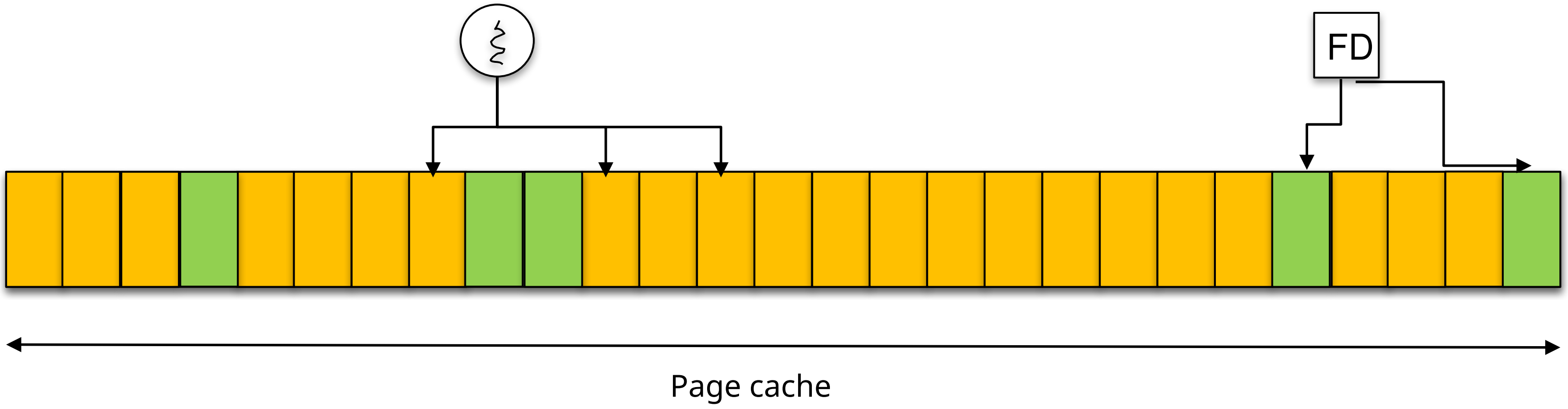
# Caching and Buffering

- Reading and writing can very IO intensive.
  - File open: two IO for each directory component and one read for the data.
- Buffer Cache
  - cache the disk blocks to reduce the IO.
  - LRU replacement
  - Static partitioning: 10% of DRAM, inefficient usage



# Caching and Buffering

- Page Cache
  - Merge virtual memory and buffer cache
  - A physical page frame can host either a page in the process address space or a file block.
    - Process uses page table to map a virtual page to a page frame.
    - A file IO uses “`address_space`”(Linux) to map a file block to a physical page frame.
  - Dynamic partitioning



# Summary

- Requirements for building filesystem
  - File information: inode
  - File structure: indexed file
  - Directory (name→inode-number): array of <inode #, name>'s
  - Free block information: Bitmap
- All are flexible.