CS310 Operating Systems

Lecture 42: File System Design

Ravi Mittal

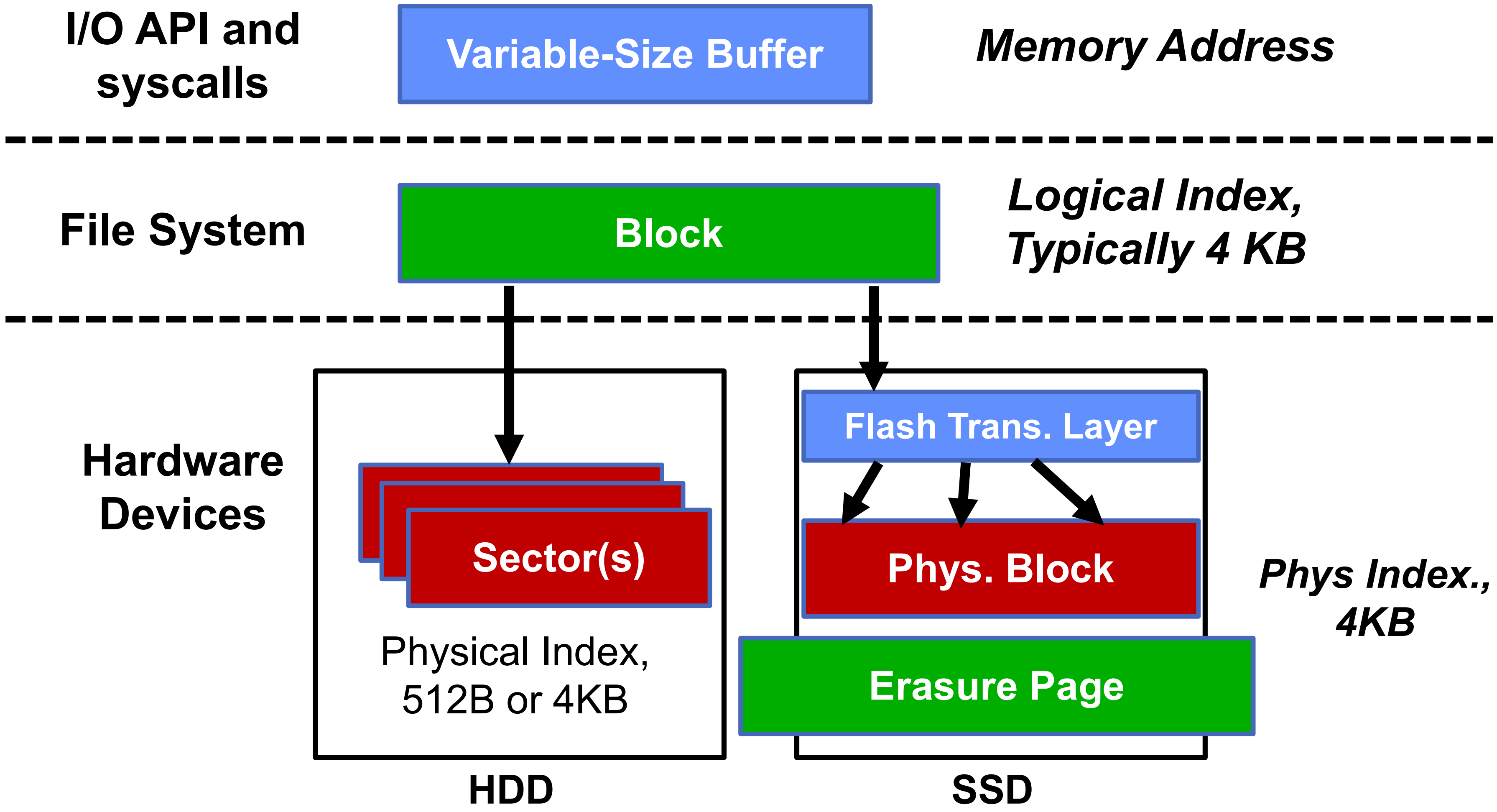
IIT Goa

Acknowledgements !

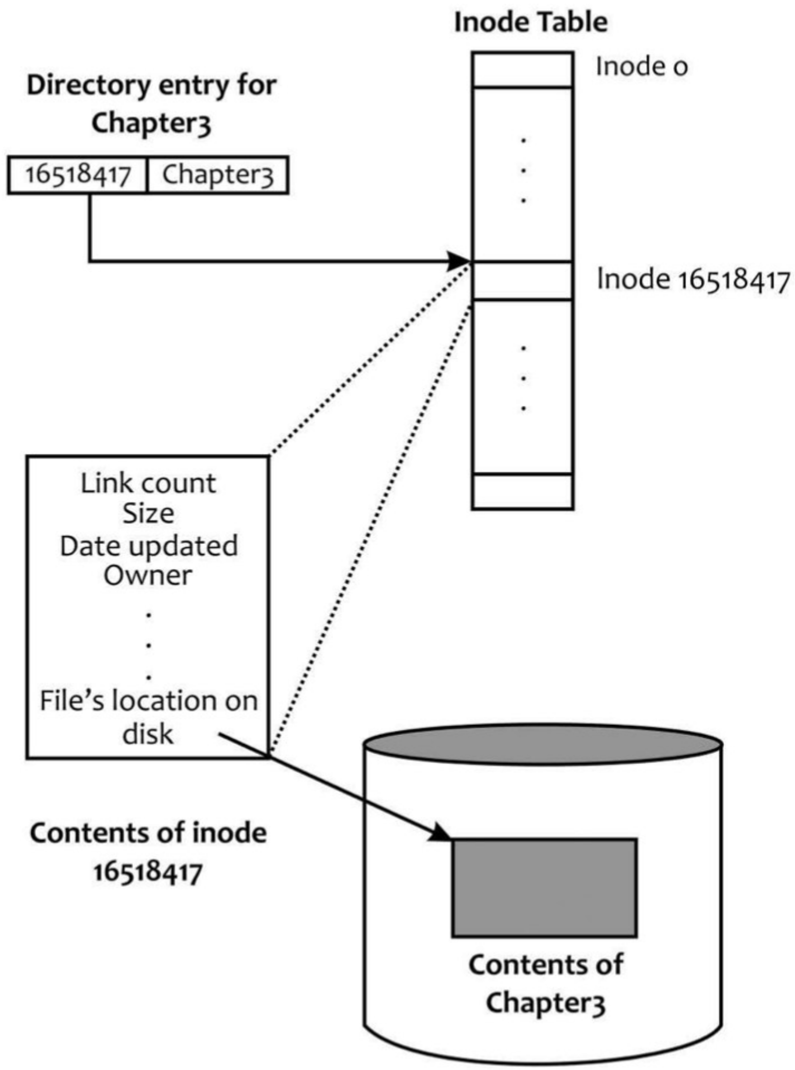
- Contents of this class presentation has been taken from various sources. Thanks are due to the original content creators:
  - CS162, Operating System and Systems Programming, University of California, Berkeley
  - Book: Modern Operating Systems by Andrew Tanenbaum and Herbert Bos
  - Book: Operating System Concepts, 10th Edition, by Silberschatz, Galvin, and Gagne

File System - Data Structure (Unix/Linux)

From Storage to File Systems

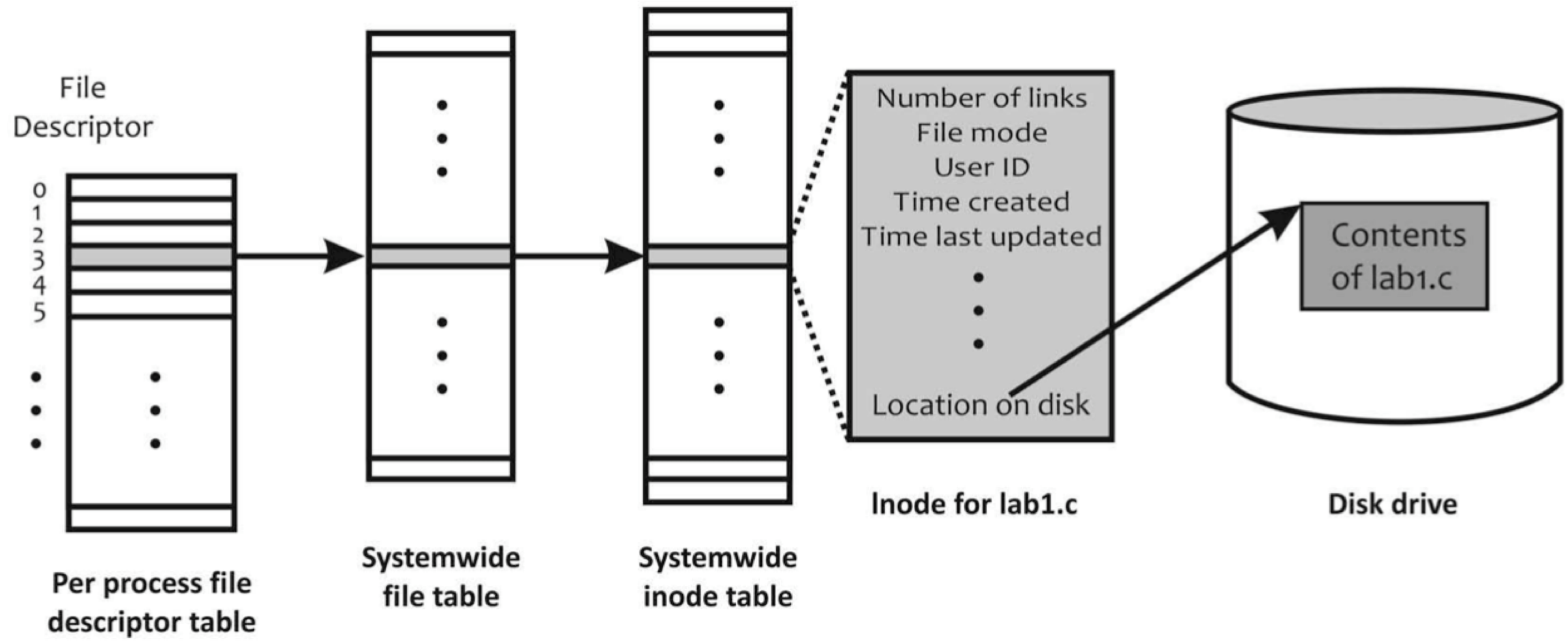


Directory Entry, inode, and file contents



Adapted from: Linux: The Textbook by Sarwar and koretsky

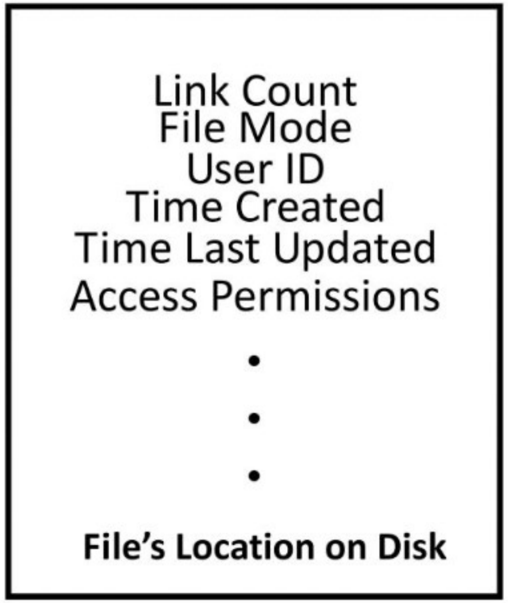
File descriptor, file table, inode, disk



vnode table (Unix)

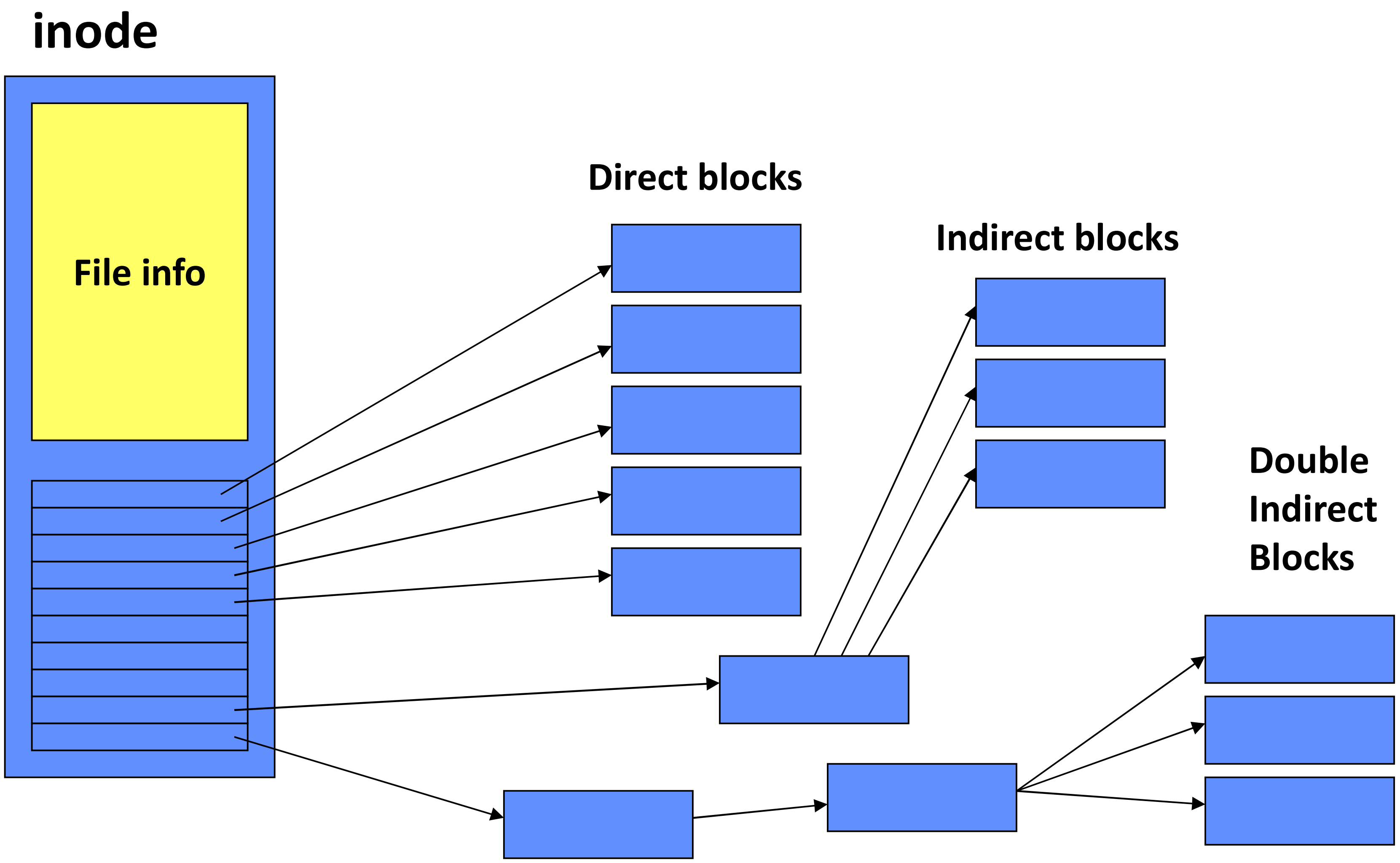
Adapted from: Linux: The Textbook by Sarwar and koretsky

Contents of *inode*

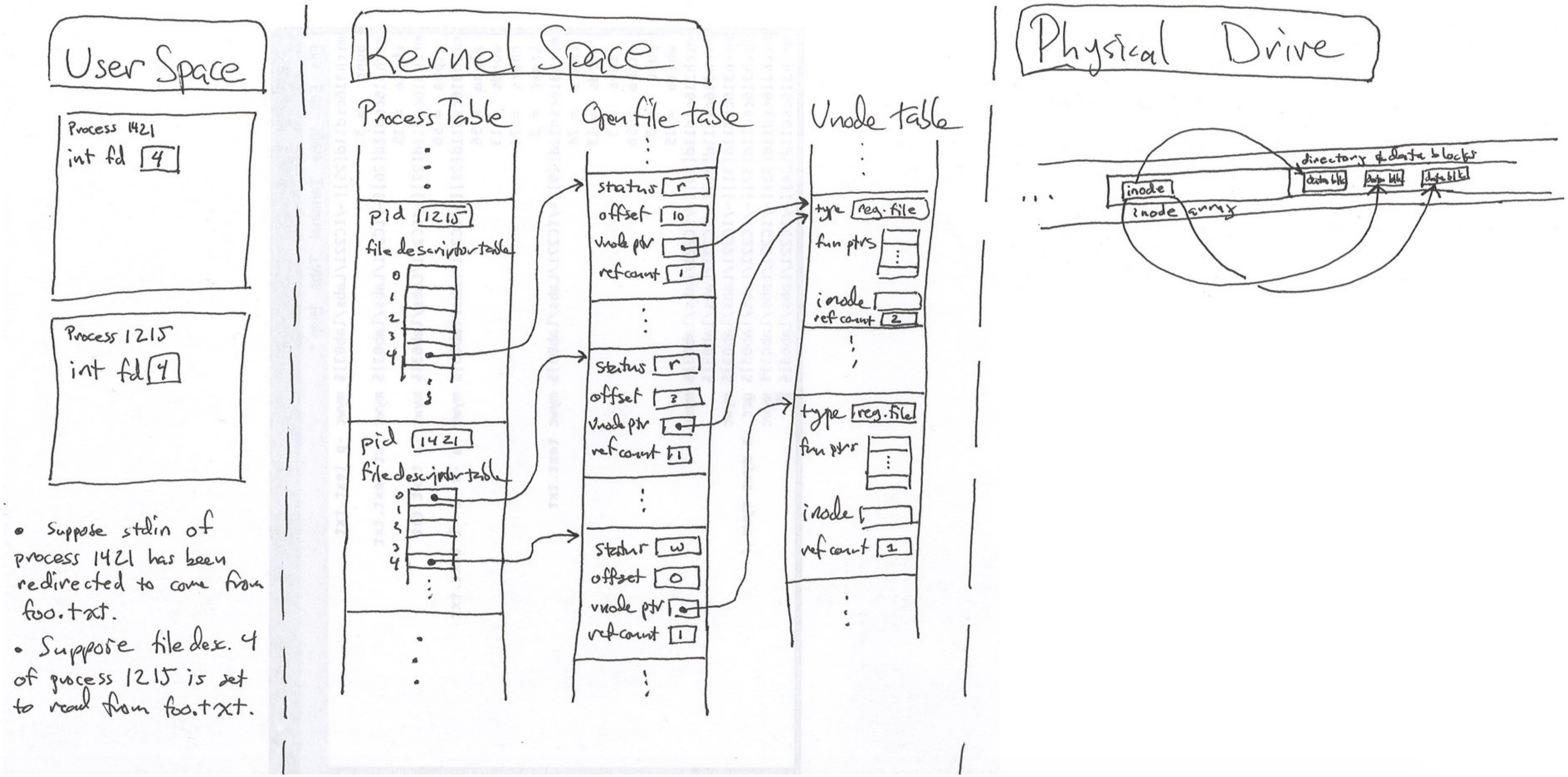


Adapted from: Linux: The Textbook by Sarwar and koretsky

Inode diagram



Big Picture



- Suppose stdin of process 1421 has been redirected to come from foo.txt.
- Suppose file desc. 4 of process 1215 is set to read from foo.txt.

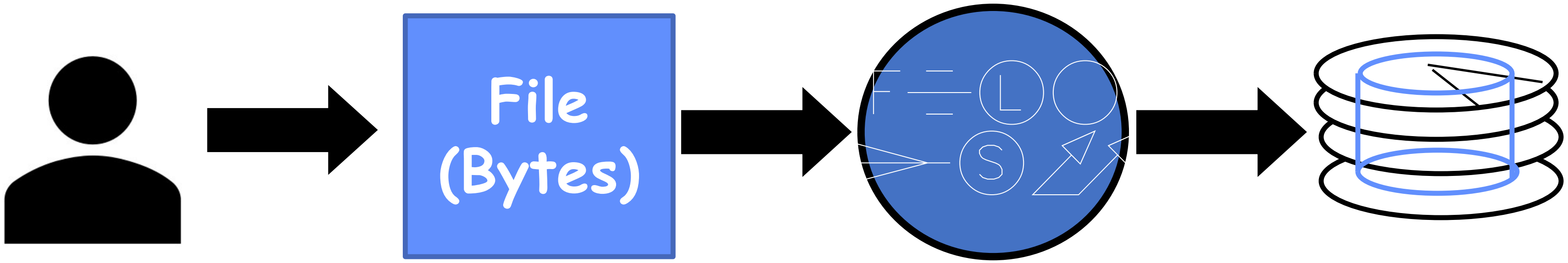
https://www.usna.edu/Users/cs/wcbrown/courses/IC221/classes/L09/Class.html

Today we will study

- File System Design Concepts
- File System Layout
- Allocation of blocks to files
- FAT: File Allocation Table
- Unix / Linux File Sytem

File System Design - Concepts

Translating from User to Sys. View



What happens if user says: "give me bytes 2 – 12?"

- Fetch block corresponding to those bytes
- Return just the correct portion of the block

What about writing bytes 2 – 12?

- Fetch block, modify relevant portion, write out block

Everything inside file system is in terms of **whole-size blocks**

- Actual disk I/O happens in blocks
- `read/write` smaller than block size needs to translate and buffer

Disk Management

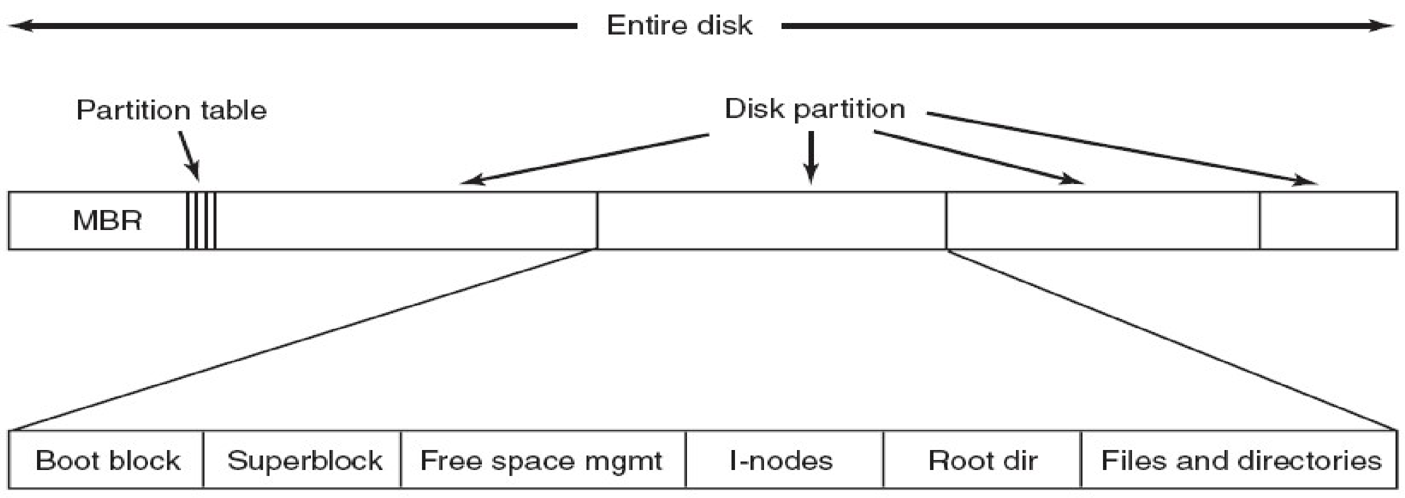
- The disk is accessed a linear array of sectors
- How to identify a sector? Two Options:
  - Physical position
    - Sector is a vector [cylinder, surface, sector]
    - This method is not used anymore
    - OS/BIOS must deal with bad sectors
  - Logical Block Addressing (LBA)
    - Every sector has an integer address
    - Control translates from address to physical position
    - Shields OS from disk structure

File System Layout

File System Layout

- File systems are stored on disks
- Disks can be divided up into one or more partitions
  - Each partition with independent file system
- Sector 0 of disk is the Master Boot Record (MBR)
  - Used to boot the computer
- After MBR is the partition table
  - It has starting and ending addresses of each partition
- One of the partition is marked as active in the master boot table
- To boot computer, BIOS reads and executes MBR
- MBR finds active partition and reads in first block (boot block)
- The program in the boot block loads the operating system contained in that partition
  - Every partition starts with boot block

A possible File System Layout



File System Layout

- Super block contains all key parameters about the file system
  - It is read into the memory when the computer is booted or file system is first touched
    - File system type, the number of blocks in the file system etc
- Free space management
  - Information about free blocks in the file system
    - Bit map or list of pointers
- i-nodes
  - An array of data structures for each file – tells about the file
- Root Directory
- Remainder of the disk contains other directories and files

Allocation of Blocks to Files

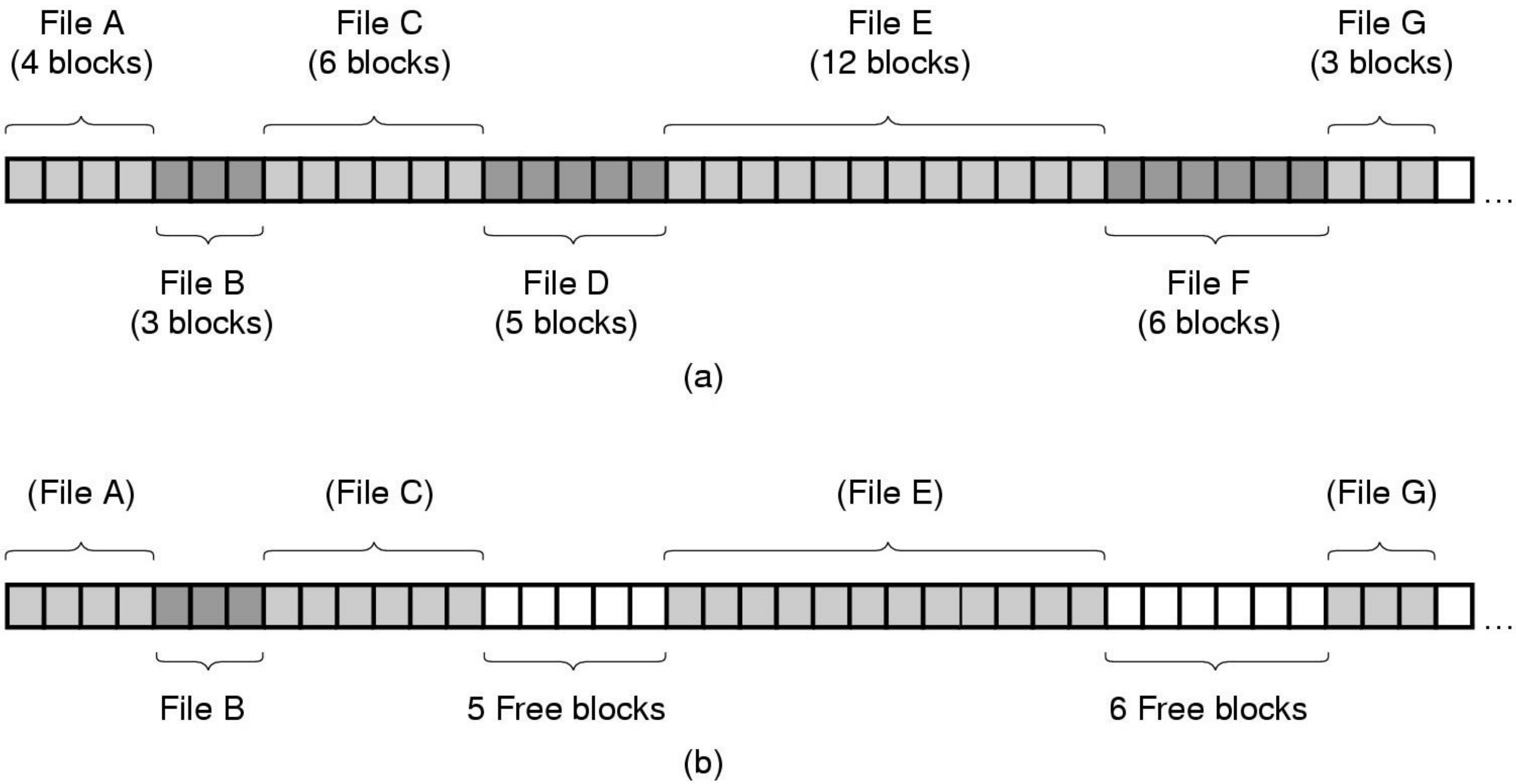
Allocating Blocks to Files

- Contiguous Allocation
- Linked List Allocation
- Linked list using table
- i-nodes

Allocating Blocks to Files

- Contiguous Allocation
- Linked List Allocation
- Linked list using table
- i-nodes

Contiguous Allocation



(a) Contiguous allocation of disk space for 7 files

(b) The state of the disk after files D and F have been removed.

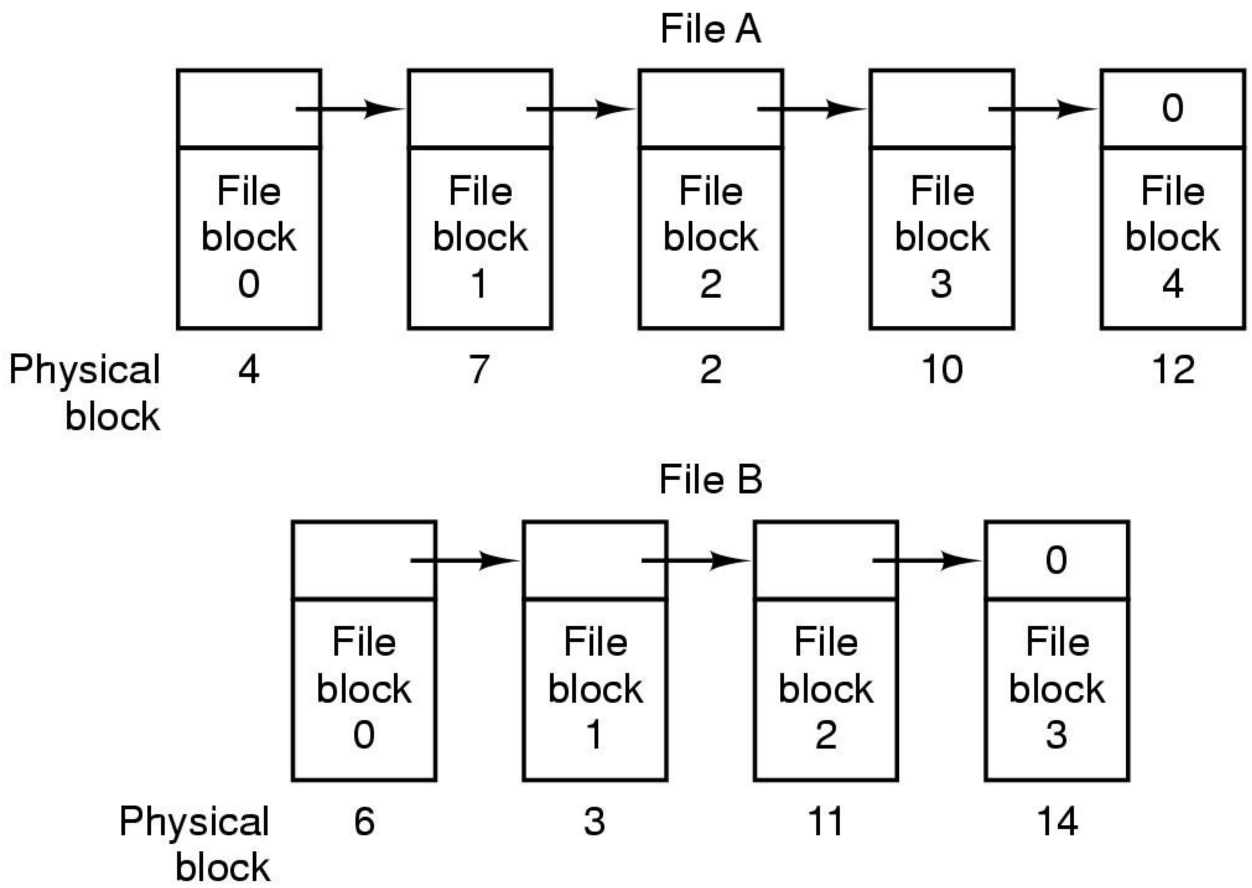
Contiguous Allocation

- Fragmentation – the main issue
- Used in CD-ROMs

Allocating Blocks to Files

- Contiguous Allocation
- Linked List Allocation
- Linked list using table
- i-nodes

Linked List Allocation



- Storing a file as a linked list of disk block
- Slow random access
- No fragmentation

Allocating Blocks to Files

- Contiguous Allocation
- Linked List Allocation
- Linked list using table – File Allocation Table
- i-nodes

FAT: File Allocation Table

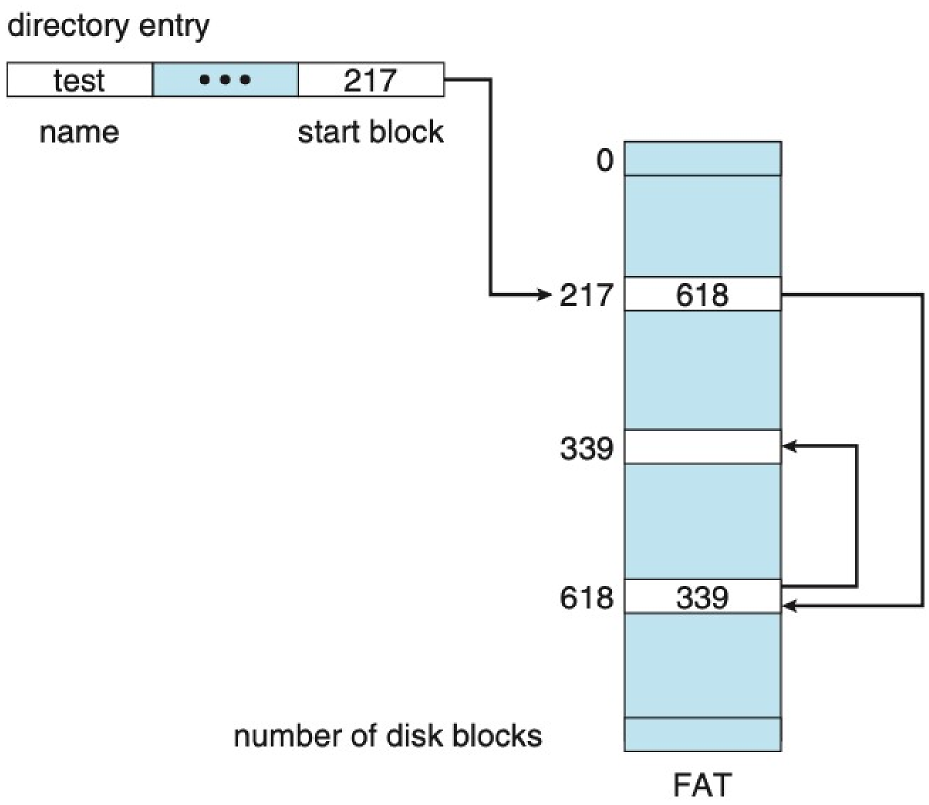
- MS-DOS, 1977
- Still widely used!

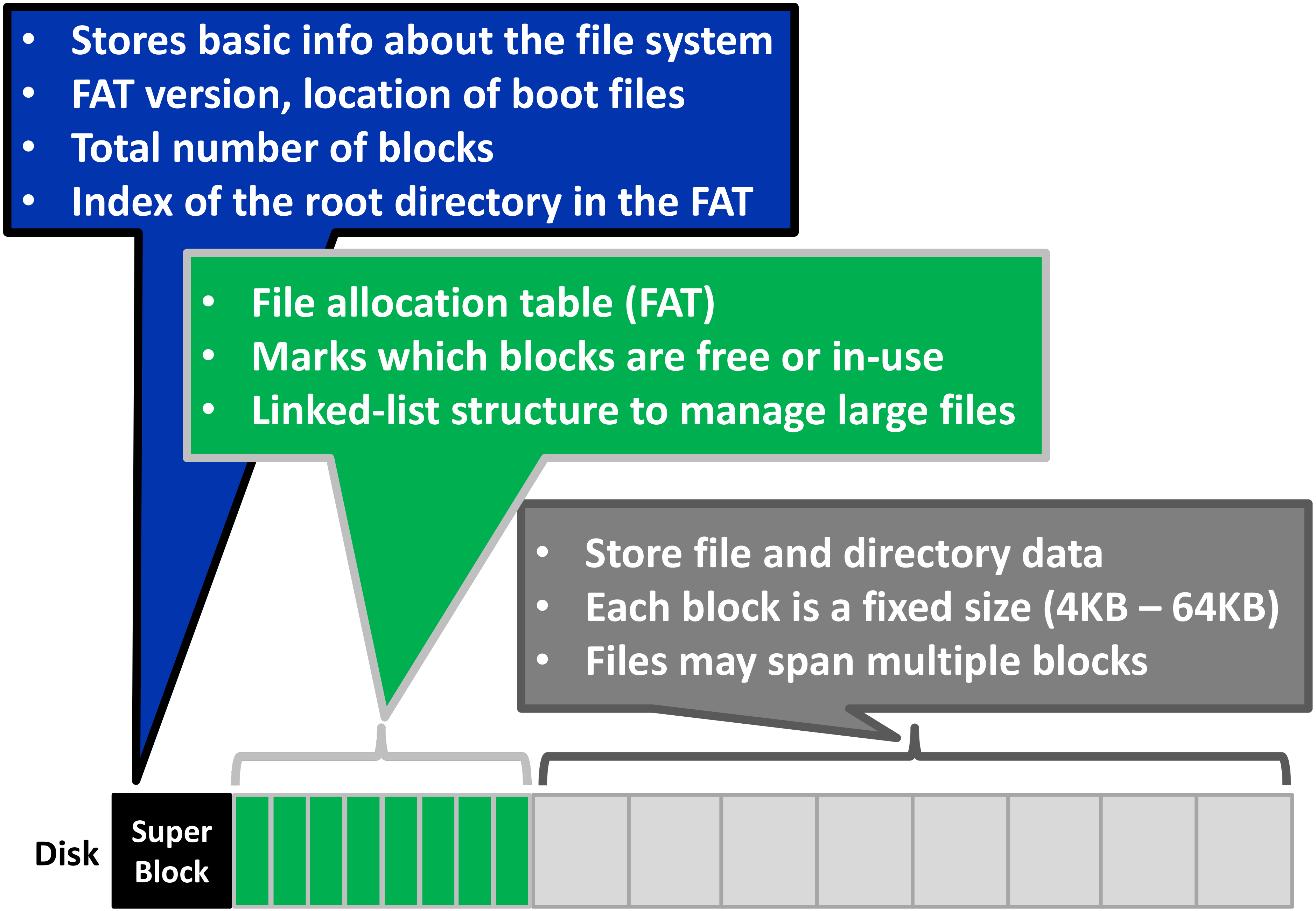
File Allocation Table (FAT)

- Simple file system popularized by MS-DOS
  - First introduced in 1977
  - Most devices today use the FAT32 spec from 1996
  - FAT12, FAT16, VFAT, FAT32, etc.
- Still quite popular today
  - Default format for USB sticks and memory cards

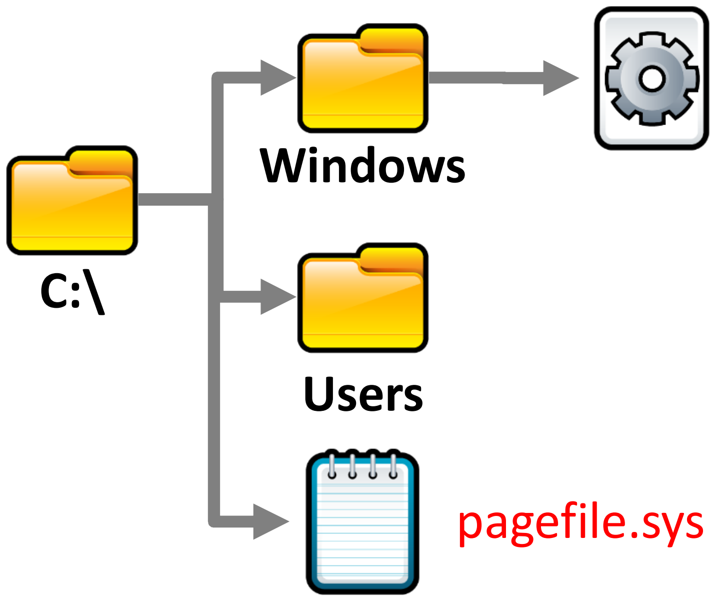
Linked List using a table in memory - FAT

- File Allocation Table
- Table has one entry for each block

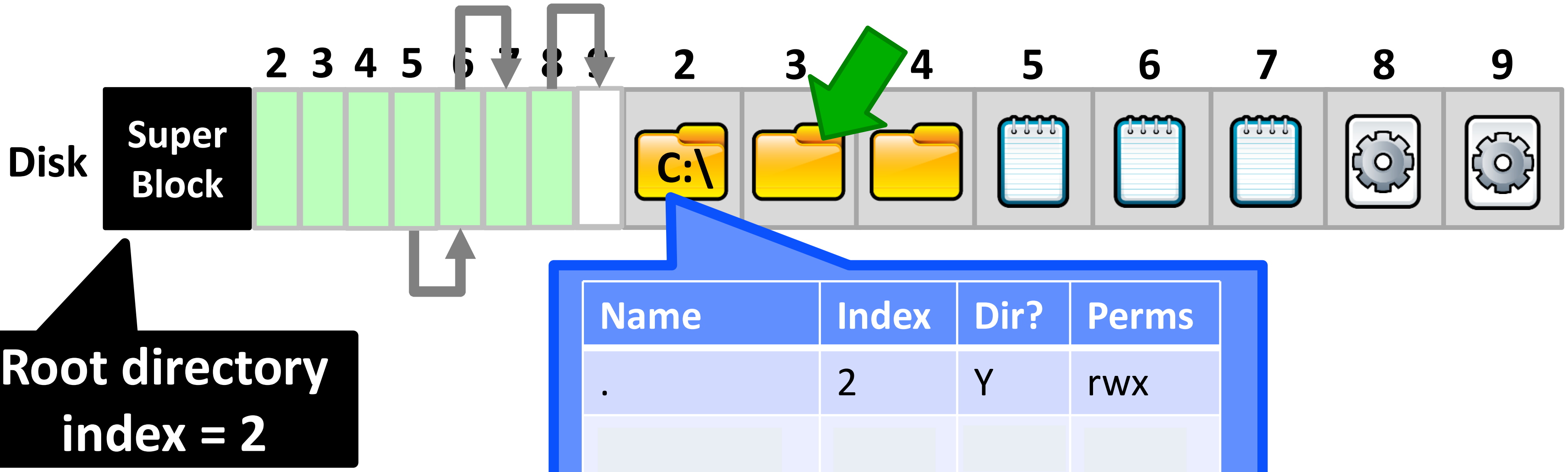




Stores basic info about the file system
FAT version, location of boot files
Total number of blocks
Index of the root directory in the FAT
File allocation table (FAT)
Marks which blocks are free or in-use
Linked-list structure to manage large files
Store file and directory data
Each block is a fixed size (4KB – 64KB)
Files may span multiple blocks
Disk
Super Block



- Directories are special files
  - File contains a list of entries inside the directory
- Possible values for FAT entries:
  - 0 – entry is empty
  - 1 – reserved by the OS
  - 1 < N < 0xFFFF – next block in a chain
  - 0xFFFF – end of a chain



Name	Index	Dir?	Perms
.	2	Y	rwx

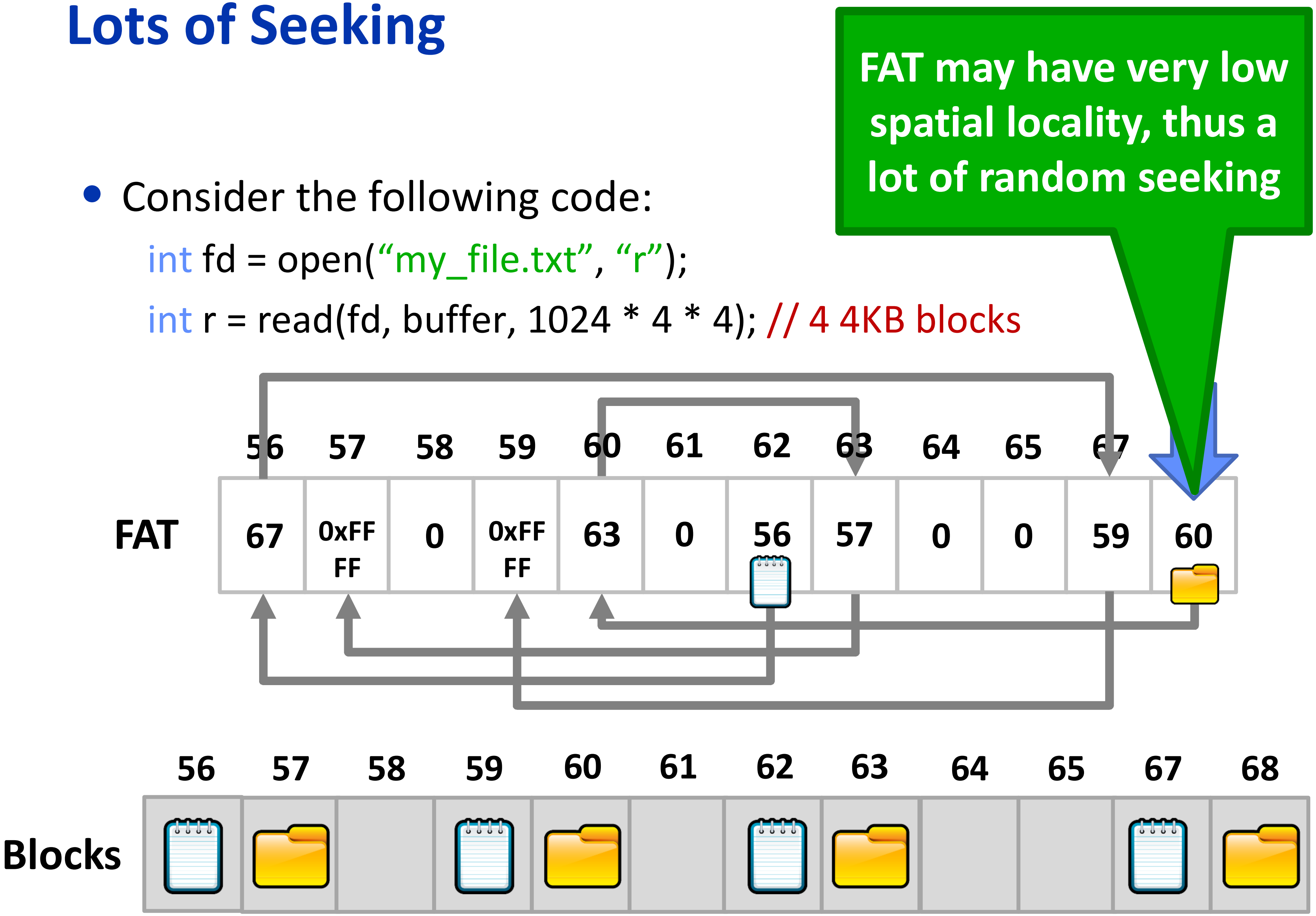
FAT - Properties

- The Good –
  - Hierarchical tree of directories and files
  - Variable length files
  - Basic file and directory meta-data
- The Bad
  - At most, FAT32 supports 2TB disks
  - Locating free chunks requires scanning the entire FAT
  - Prone to internal and external fragmentation
    - Large blocks → internal fragmentation
  - **Reads require a lot of random seeking**

Lots of Seeking

- Consider the following code:

```
int fd = open("my_file.txt", "r");
int r = read(fd, buffer, 1024 * 4 * 4); // 4 4KB blocks
```



Allocating Blocks to Files

- Contiguous Allocation
- Linked List Allocation
- Linked list using table
- i-nodes

Modern File system for Linux

1st XFS 10/10

Web: https://xfs.wiki.kernel.org **Licence:** GPL
Version: –
Its power is in the absence of any notable shortcomings.

2nd Ext4 10/10

Web: https://ext4.wiki.kernel.org **Licence:** GPL
Version: –
A super-fast and reliable filesystem, although a bit slow on flash drives.

3rd Btrfs 8/10

Web: https://btrfs.wiki.kernel.org **Licence:** GPL
Version: –
Very robust, with lots of features, but we doubt it has enough failure tolerance.

4th Reiser5 6/10

Web: https://sourceforge.net/projects/reiser4/files/v5-unstable/ **Licence:** GPL **Version:** 5
Amazing features for logical volumes, yet it's very unreliable for daily use.

5th NTFS 4/10

Web: https://github.com/tuxera/ntfs-3g **Licence:** LGPLv2
Version: 2021.8.22
Very solid and production ready, but also shows abysmal speed rates.

Unix Fast File System (Berkeley FFS)

i-node (Linux File System)

- Efficiency of FAT is very low
  - Lots of seeking over file chains in FAT
  - Only way to identify free space is to scan over the entire FAT
- Linux file system uses more efficient structures
  - Extended File System (ext) uses index nodes (inodes) to track files and directories

Size Distribution of Files

- FAT uses a linked list for all files
  - Simple and uniform mechanism
  - … but, it is not optimized for short or long files
- Question: are short or long files more common?
  - Studies over the last 30 years show that short files are much more common
  - 2KB is the most common file size
  - Average file size is 200KB (biased upward by a few very large files)
- Key idea: optimize the file system for many small files

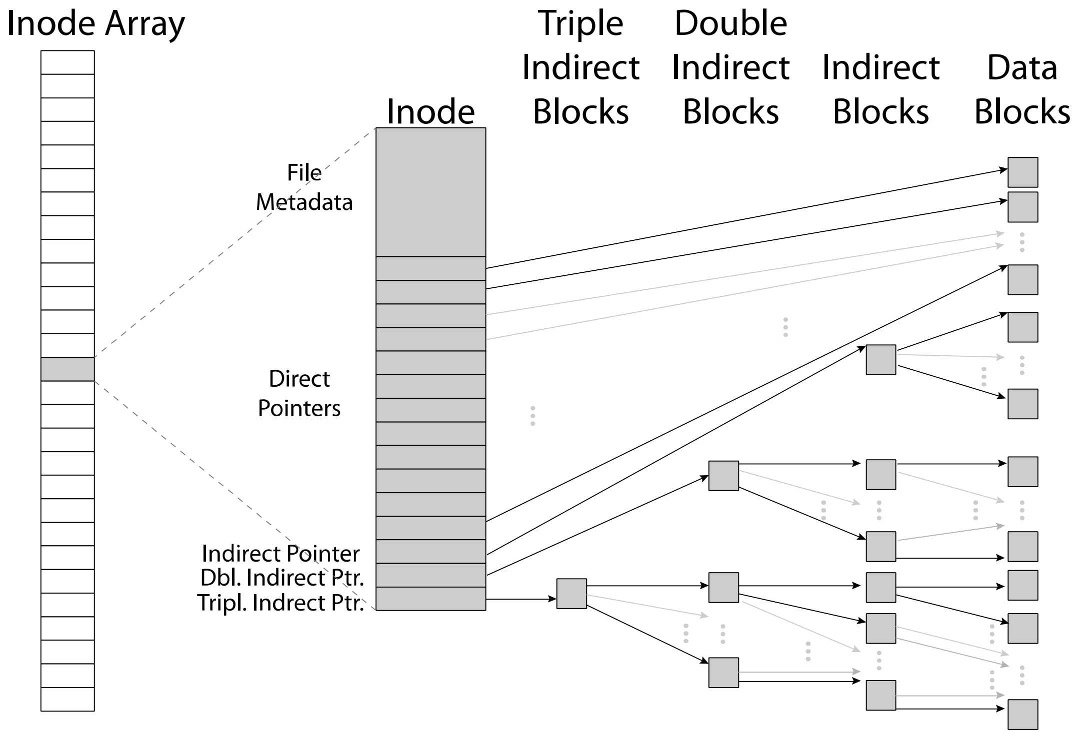
Inodes in Unix (Including Berkeley FFS)

- Original ***inode*** format appeared in BSD 4.1 (more following)

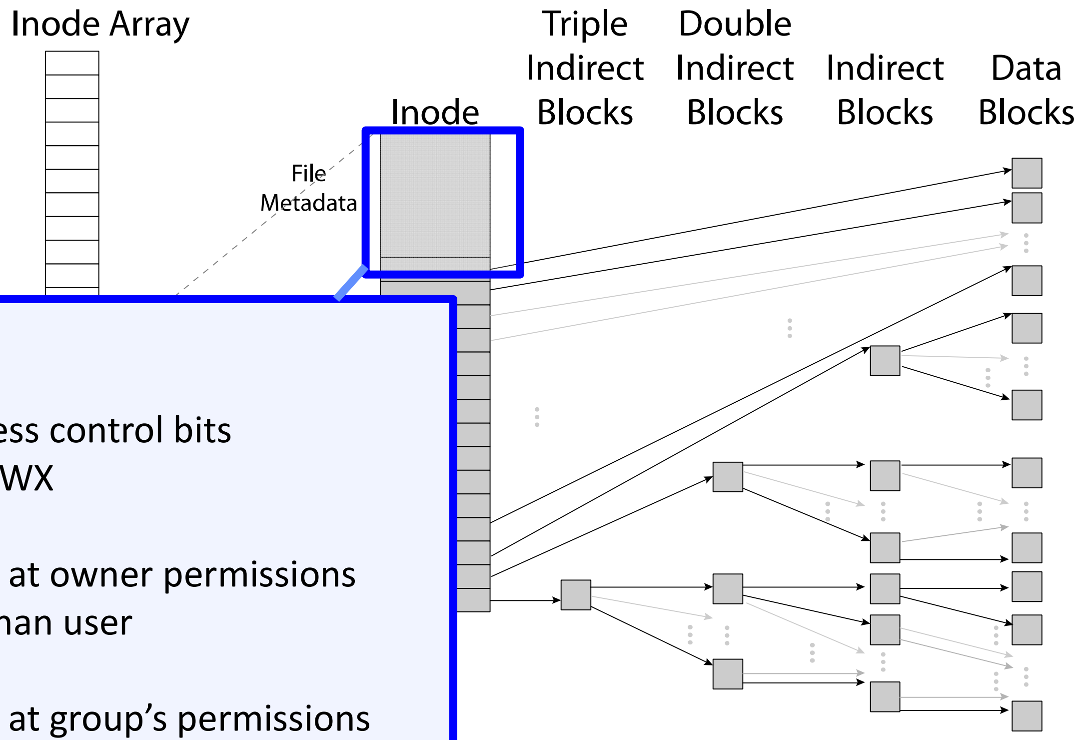
- Index structure is an array of *inodes*
  - File Number (inumber) is an index into the array of inodes
  - Each inode corresponds to a file and contains its metadata

- Inode maintains a multi-level tree structure to find storage blocks for files
  - Great for little and large files
  - Asymmetric tree with fixed sized blocks

Inode Structure



Inode Structure



User
Group
9 basic access control bits
- UGO x RWX
SetUID bit
- execute at owner permissions
rather than user
SetGID bit
- execute at group's permissions

Small Files: 12 Pointers Direct to Data Blocks

Direct pointers

4kB blocks ⇒ sufficient for files up to 48KB

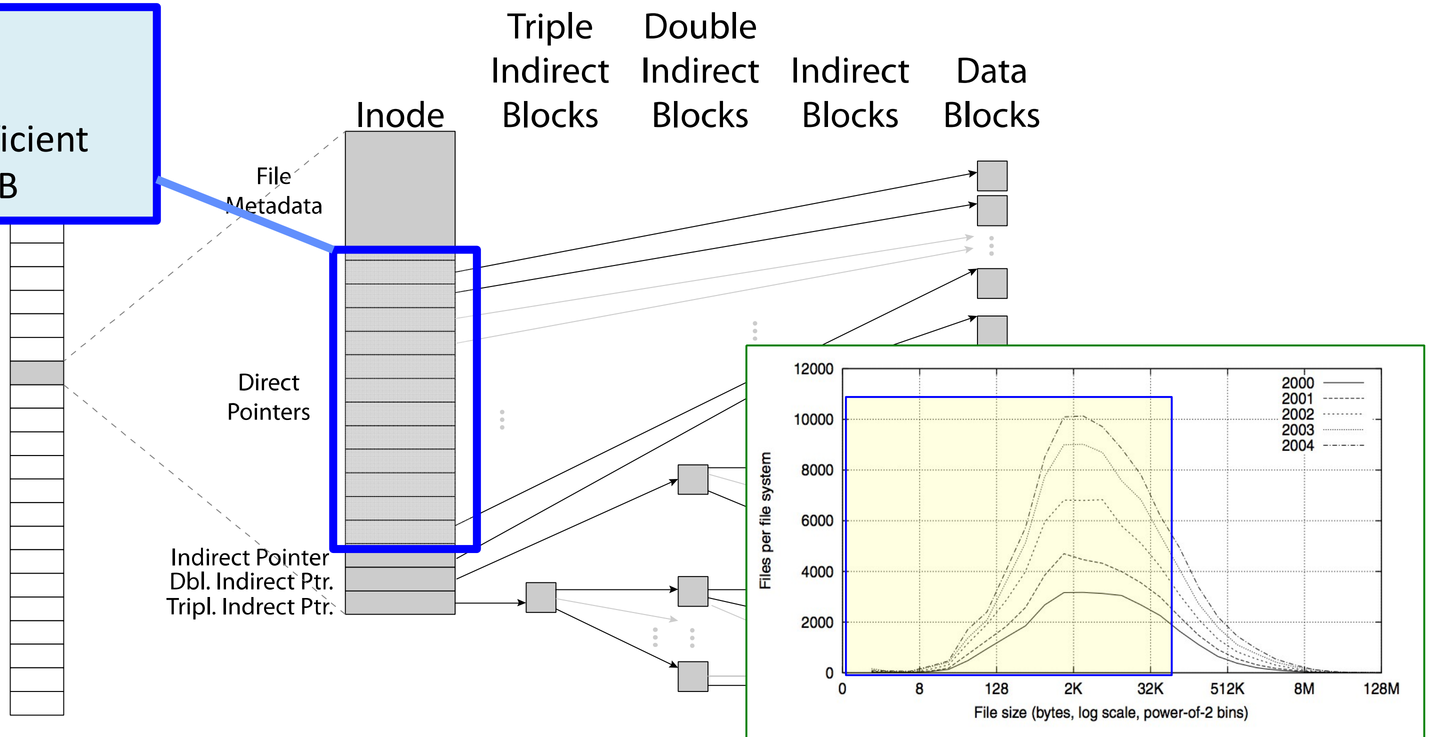


Fig. 2. Histograms of files by size.

Large Files: 1-, 2-, 3-level indirect pointers

Indirect pointers
- point to a disk block containing only pointers
- 4 kB blocks => 1024 ptrs
  - => 4 MB @ level 2
  - => 4 GB @ level 3
  - => 4 TB @ level 4

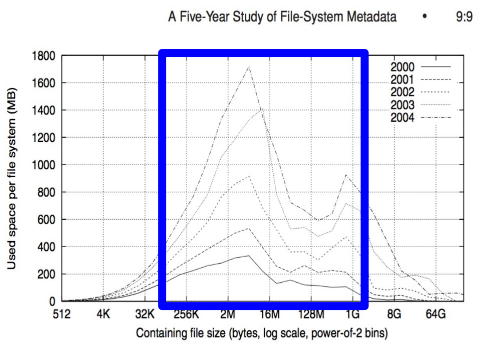
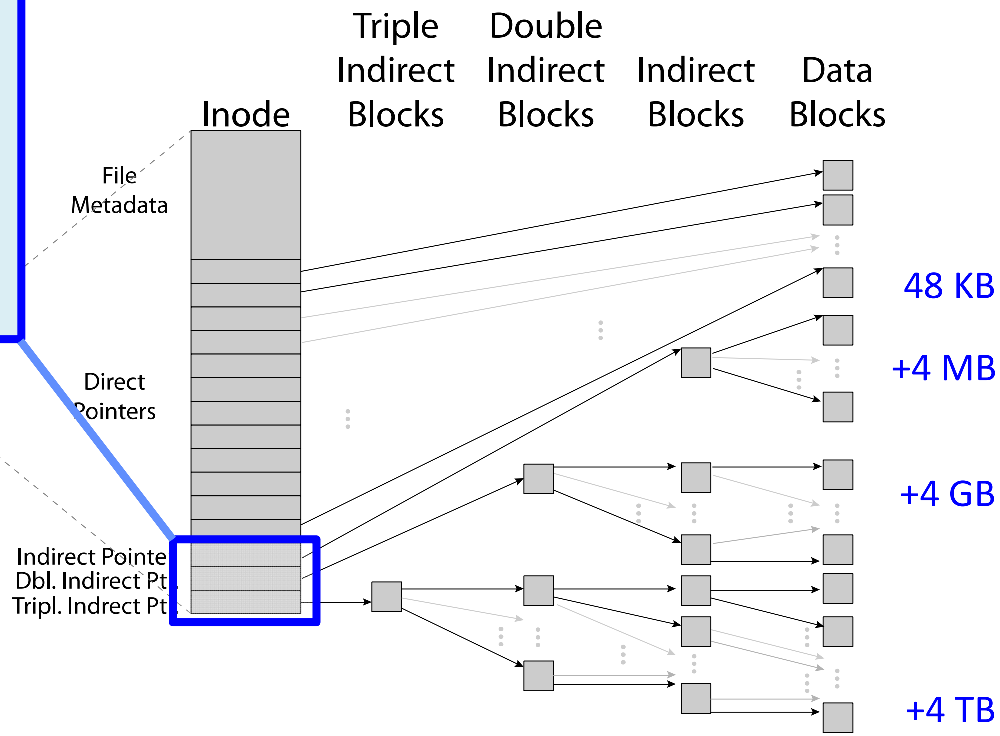


Fig. 4. Histograms of bytes by containing file size.

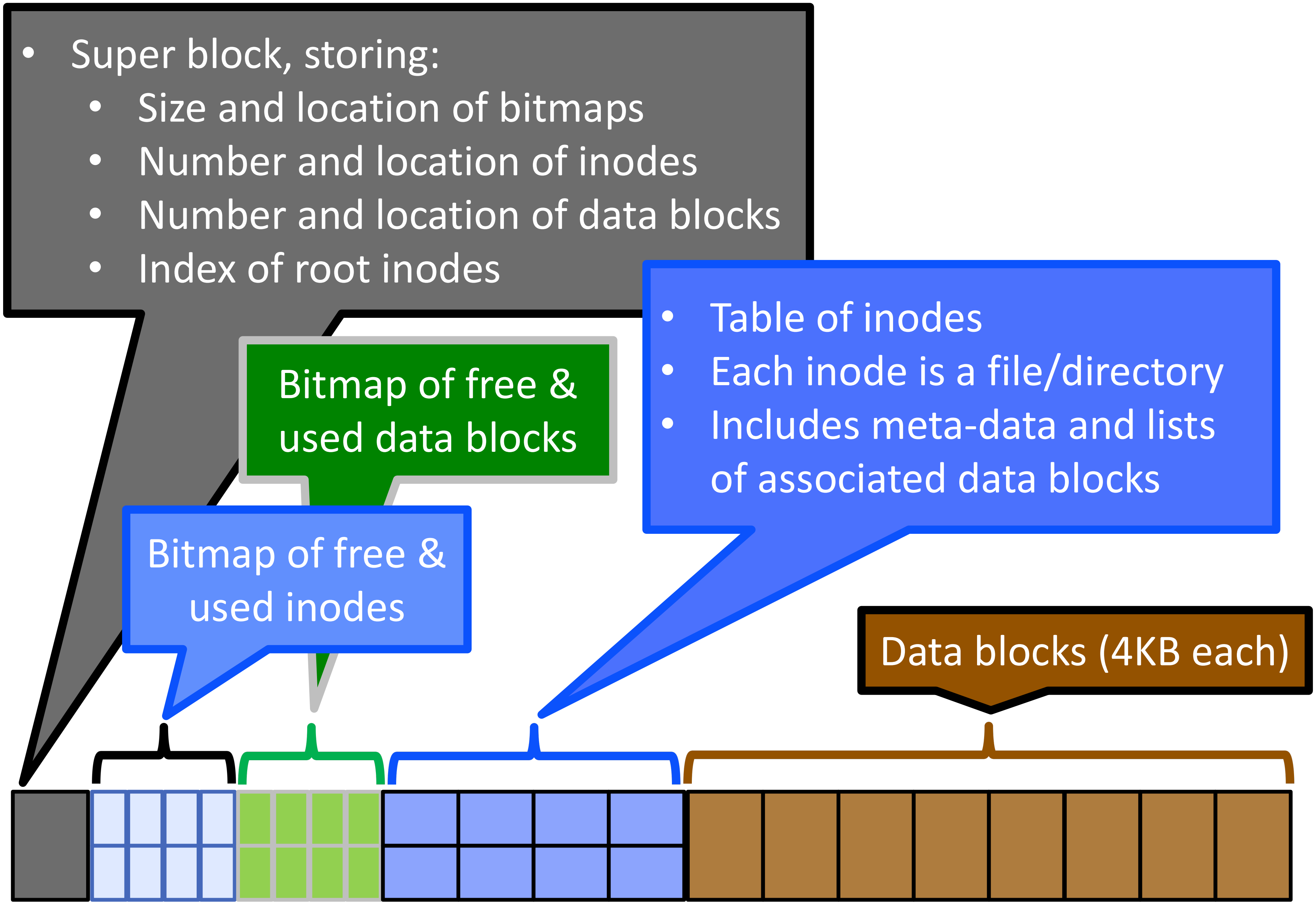


Advantages of inodes

- Optimized for file systems with many small files
  - Each inode can directly point to 48KB of data
  - Only one layer of indirection needed for 4MB files
- Faster file access
  - Greater meta-data locality → less random seeking
  - No need to traverse long, chained FAT entries
- Easier free space management
  - Bitmaps can be cached in memory for fast access
  - inode and data space handled independently

Lecture Summary

- File System:
  - Transforms blocks into Files and Directories
  - Optimize for size, access and usage patterns
  - Maximize sequential access, allow efficient random access
- FAT: File Allocation Table
  - Very popular (DOS)
  - File indexes into simple table, find blocks by traversing linked list
  - Simple; table size may become very big for large disk (eg 2-4 TB); Lots of seeks
- FFS (Linux/Unix)
  - `inode` file index structure
  - Asymmetric, multi-level tree
  - One block per leaf of tree


• Super block, storing:
• Size and location of bitmaps
• Number and location of inodes
• Number and location of data blocks
• Index of root inodes
Bitmap of free & used data blocks
Bitmap of free & used inodes
• Table of inodes
• Each inode is a file/directory
• Includes meta-data and lists of associated data blocks
Data blocks (4KB each)