File Systems

18 January 2026
Lecture 12

Slides adapted from John Kubiatowicz (UC Berkeley)

Concept Review

Fragmentation

- Internal
- External

Page Table Entry

Valid/Invalid bit

Page

Page table

Multi-level page table

Topics for Today

- File Systems
  - Introduction to File Systems
  - Very simply file system
  - FAT
  - Inodes
  - Unix Fast File System (FFS)

Building a File System

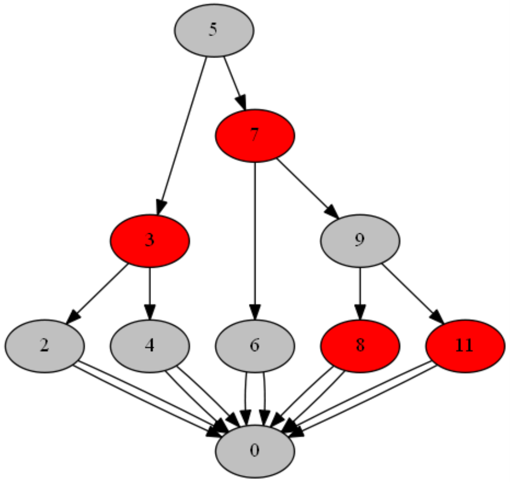
- File System: Layer of OS that transforms block interface of disks (or other block devices) into Files, Directories, etc.
  - File System Components

- Disk Management: collecting disk blocks into files
- Naming: Interface to find files by name, not by blocks
- Protection: Layers to keep data secure
- Reliability/Durability: Keeping of files durable despite crashes, media failures, attacks, etc

User vs. System View of a File

User's view:

- Durable Data Structures

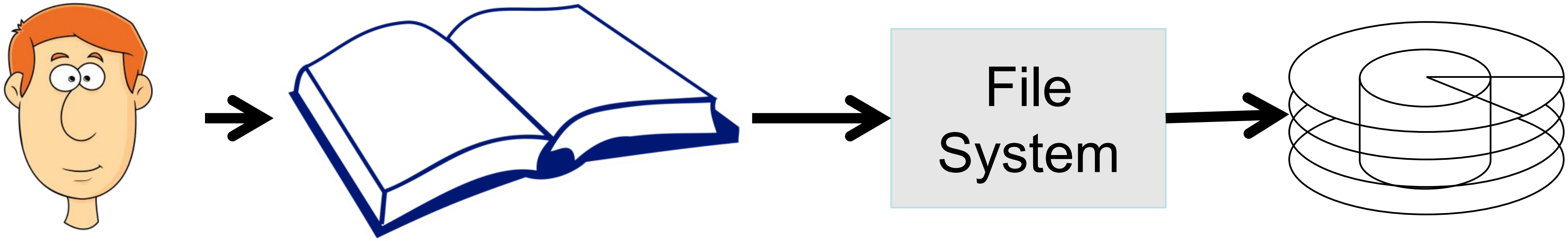


System's View:

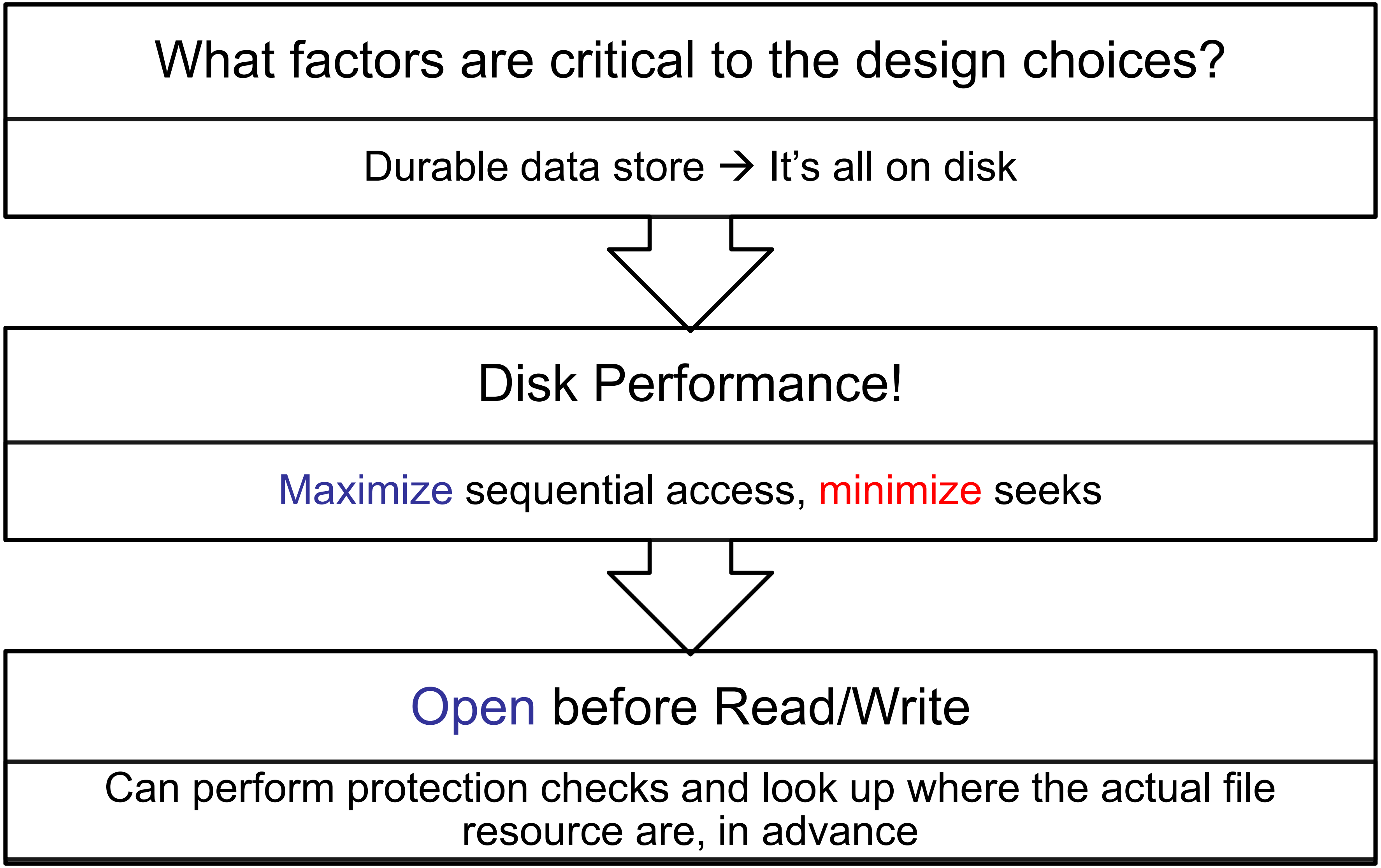
- System call interface:
  - Collection of Bytes (UNIX)
  - Doesn't matter to system what kind of data structures you want to store on disk!
- Inside OS:
  - Collection of blocks (a block is a logical transfer unit, while a sector is the physical transfer unit)
  - Block size ≥ sector size; in UNIX, block size is 4KB

Translating from User to System 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: *Write bytes 2-12?*
  - Fetch block, Modify portion, Write out Block
- Everything inside File System is in whole size blocks
  - For example, `getc()`, `putc()` ⇒ buffers something like 4096 bytes, even if interface is one byte at a time
- From now on, file is a collection of blocks



So you are going to design a file system …



What factors are critical to the design choices?

Durable data store → It’s all on disk

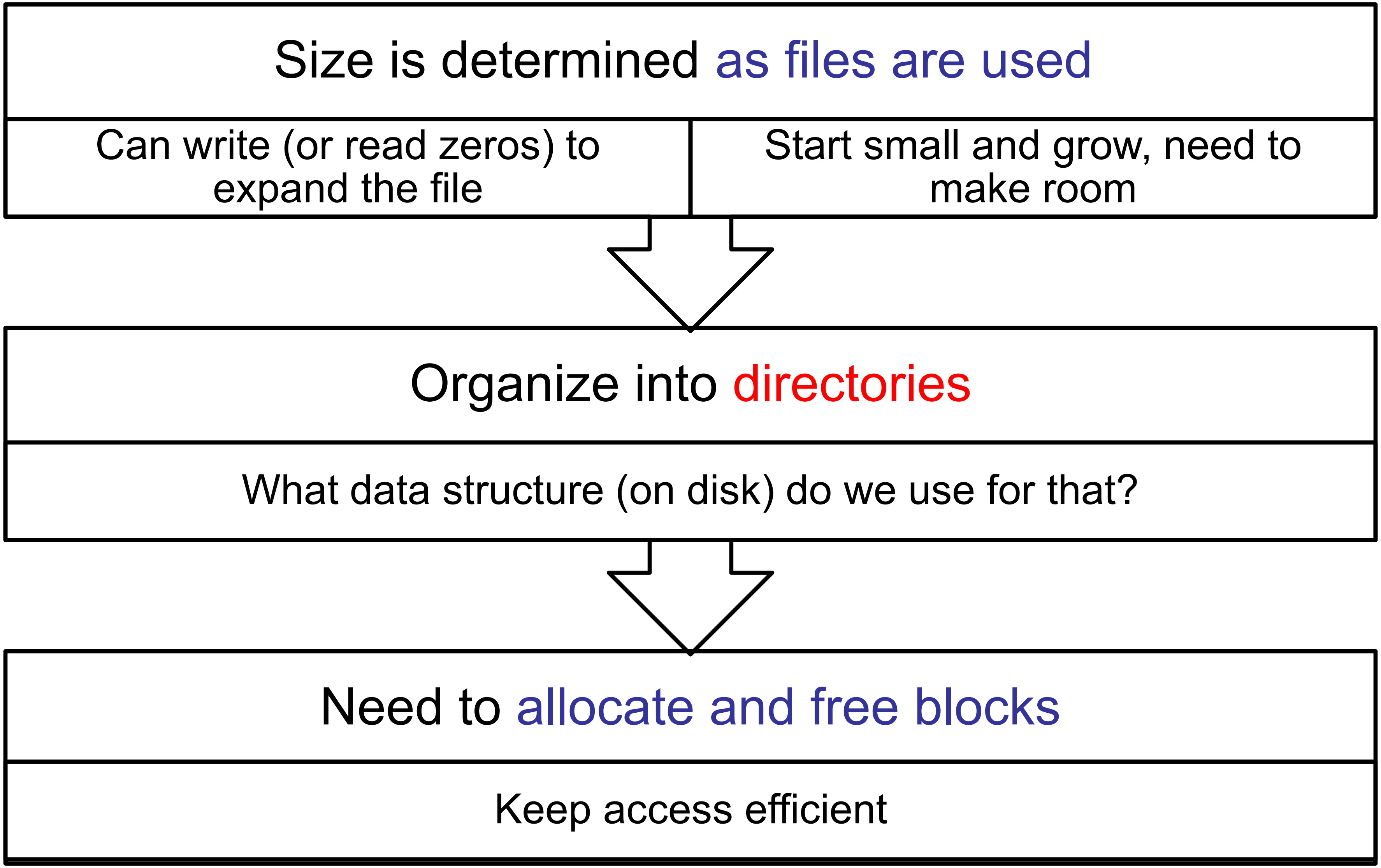
Disk Performance!

Maximize sequential access, minimize seeks

Open before Read/Write

Can perform protection checks and look up where the actual file resource are, in advance

So you are going to design a file system …



Size is determined as files are used

Can write (or read zeros) to expand the file

Start small and grow, need to make room

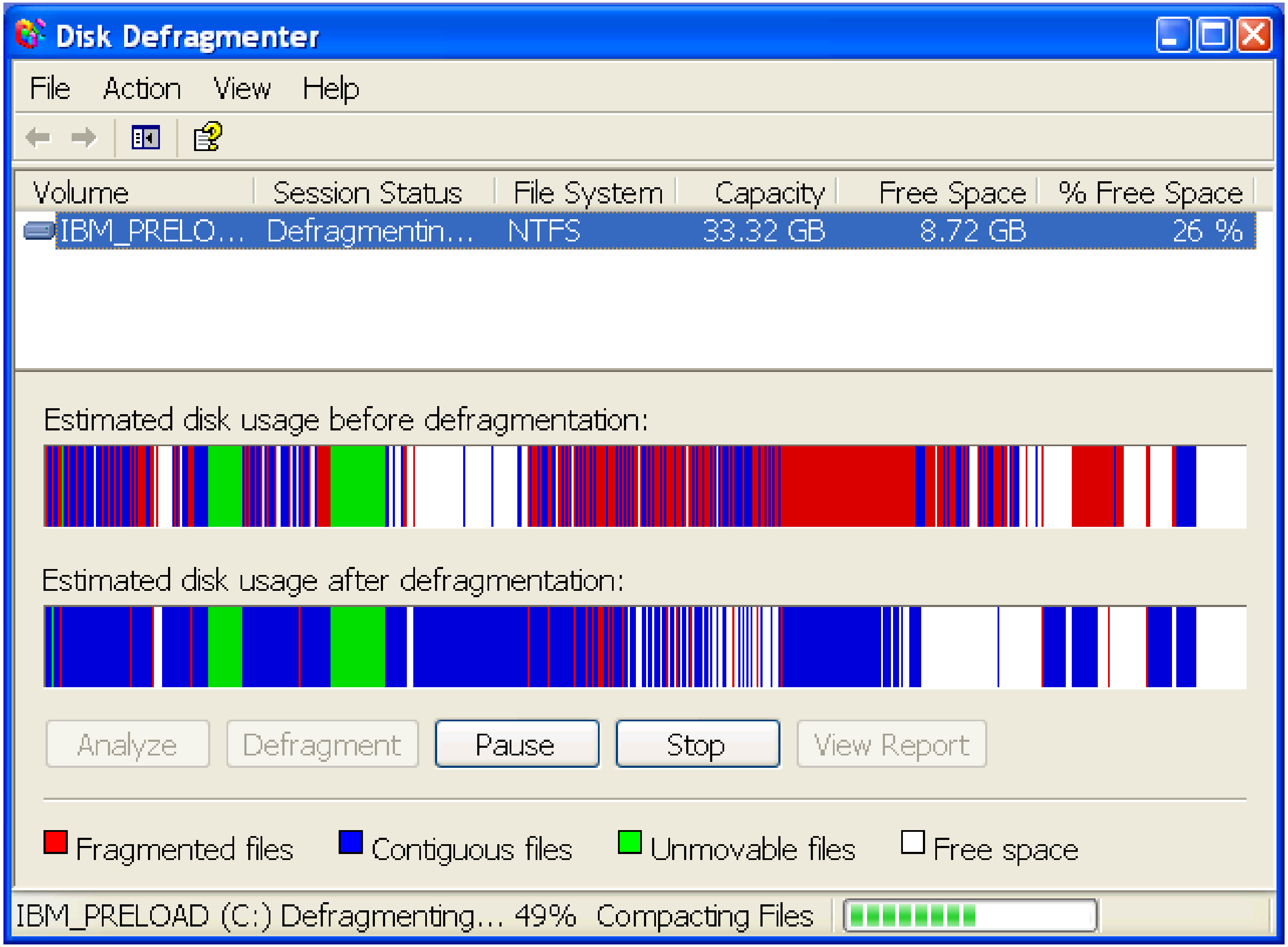
Organize into directories

What data structure (on disk) do we use for that?

Need to allocate and free blocks

Keep access efficient

Defragmenting



Volume	Session Status	File System	Capacity	Free Space	% Free Space
IBM_PRELO...	Defragmentin...	NTFS	33.32 GB	8.72 GB	26 %

Estimated disk usage before defragmentation:

Estimated disk usage after defragmentation:

🟥 Fragmented files 🟦 Contiguous files 🟩 Unmovable files □ Free space

IBM_PRELOAD (C:) Defragmenting... 49% Compacting Files

Image source: http://blog.shane-smith.com/blog:116

Disk Management Policies

- Basic entities on a disk:
  - File: User-visible group of blocks arranged sequentially in logical space
  - Directory: user-visible index mapping names to files

- Access disk as linear array of sectors.
  Two Options:
  1. Identify sectors as vectors `[cylinder, surface, sector]`. Sort in cylinder-major order. Not used much anymore.
  2. Logical Block Addressing (LBA). Every sector has integer address from zero up to max number of sectors.

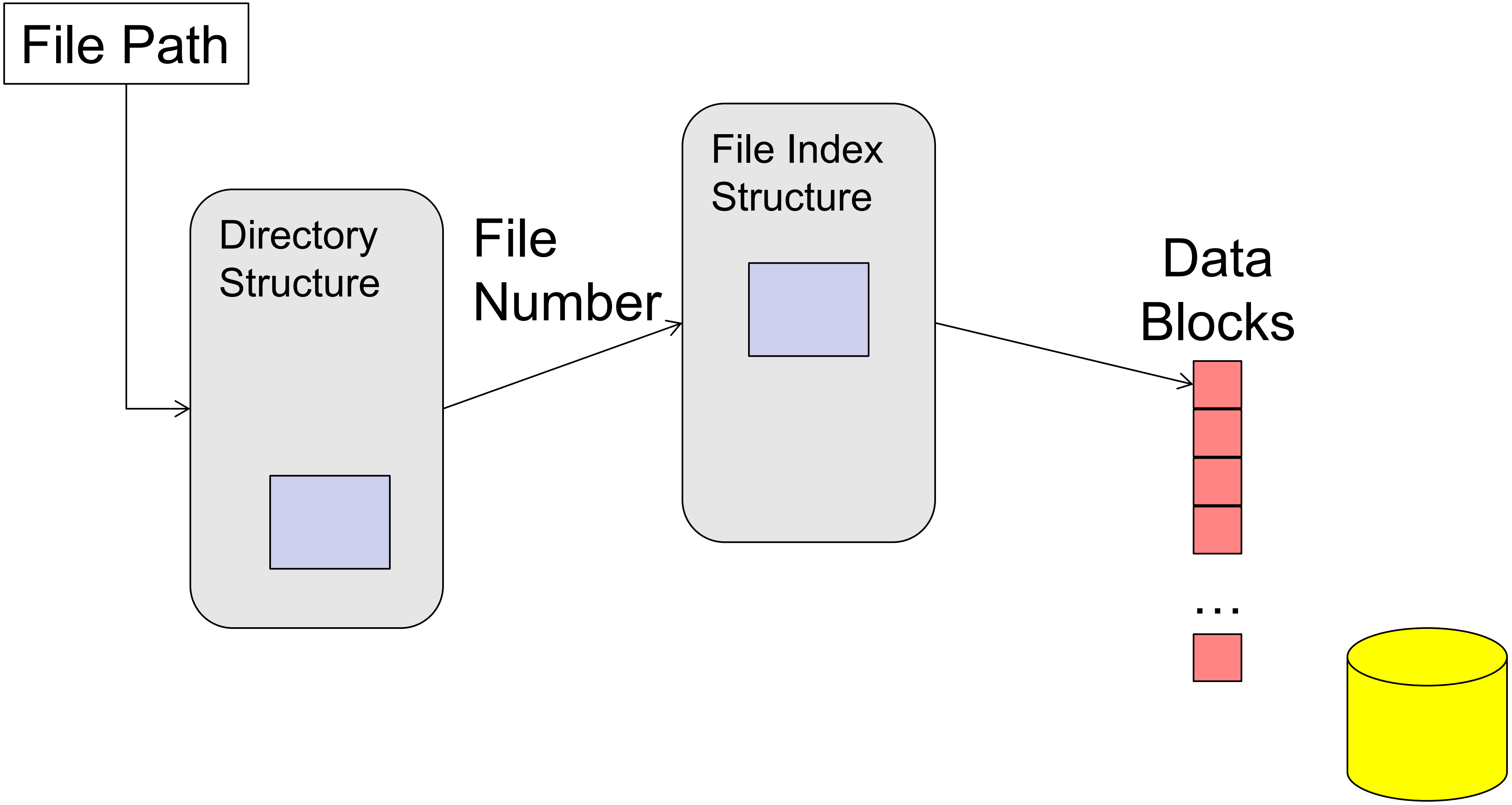
- Controller translates from address⇒physical position
  - First case: OS/BIOS must deal with bad sectors
  - Second case: Hardware shields OS from structure of disk

Disk Management Policies

- Need way to track free disk blocks
  - Link free blocks together ⇒ too slow today
  - Use bitmap to represent free space on disk

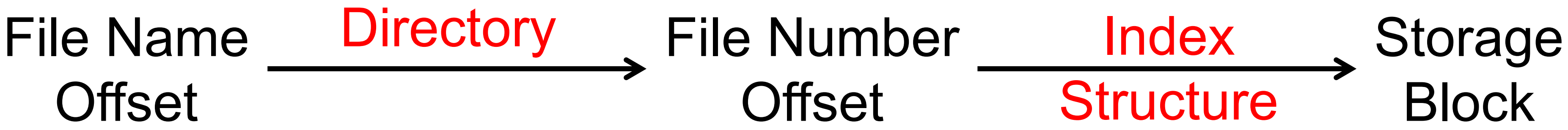
- Need a way to structure files: File Header
  - Track which blocks belong at which offsets within the logical file structure
  - Optimize placement of files' disk blocks to match access and usage patterns

Components of a File System

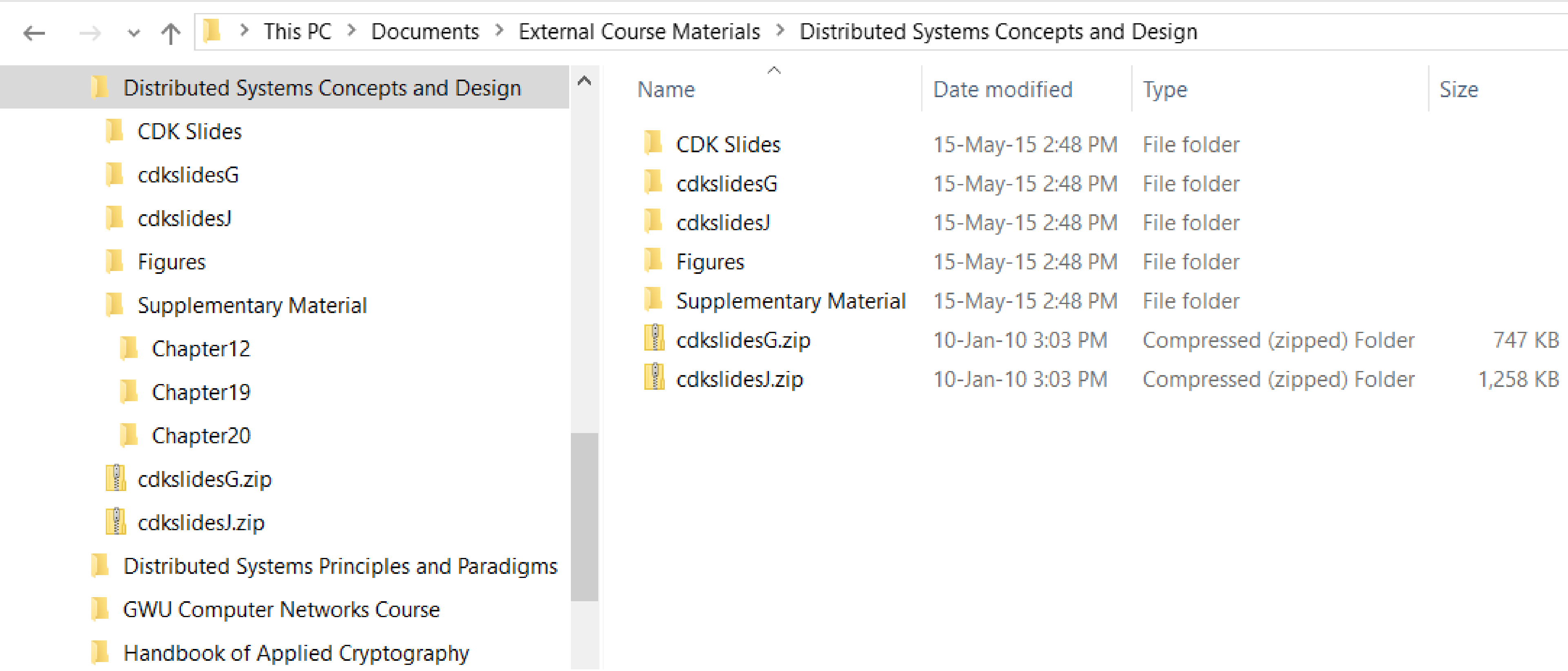


Components of a file system

- Open performs *name resolution*
  - Translates pathname into a “file number”
    - Used as an “index” to locate the blocks
  - Creates a file descriptor in PCB within kernel
  - Returns a “handle” (another `int`) to user process
- Read, Write, Seek, and Sync operate on handle
  - Mapped to **descriptor** and to **blocks**


File Name Offset → Directory → File Number Offset → Index Structure → Storage Block

Directories



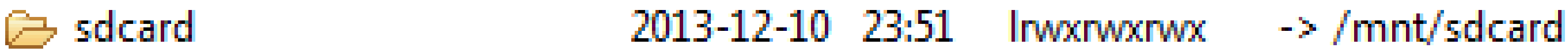
› This PC › Documents › External Course Materials › Distributed Systems Concepts and Design

- Distributed Systems Concepts and Design
  - CDK Slides
  - cdkslidesG
  - cdkslidesJ
  - Figures
  - Supplementary Material
    - Chapter12
    - Chapter19
    - Chapter20
  - cdkslidesG.zip
  - cdkslidesJ.zip
- Distributed Systems Principles and Paradigms
- GWU Computer Networks Course
- Handbook of Applied Cryptography

Name	Date modified	Type	Size
CDK Slides	15-May-15 2:48 PM	File folder	
cdkslidesG	15-May-15 2:48 PM	File folder	
cdkslidesJ	15-May-15 2:48 PM	File folder	
Figures	15-May-15 2:48 PM	File folder	
Supplementary Material	15-May-15 2:48 PM	File folder	
cdkslidesG.zip	10-Jan-10 3:03 PM	Compressed (zipped) Folder	747 KB
cdkslidesJ.zip	10-Jan-10 3:03 PM	Compressed (zipped) Folder	1,258 KB

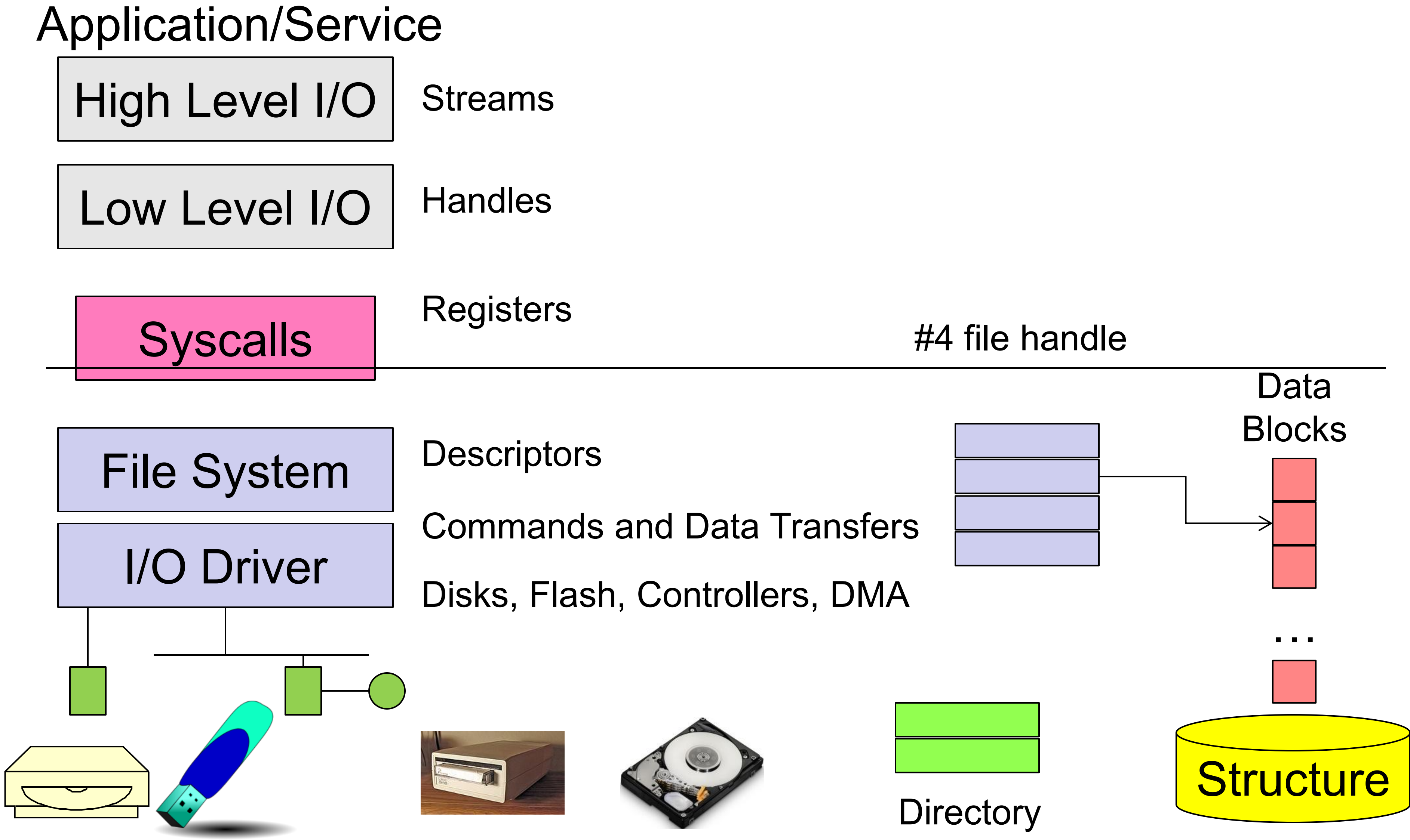
Directory

- Basically a hierarchical structure
- Each directory entry is a collection of
  - – Files
  - – Directories
    - A link to another entries
- Each has a name and attributes
  - – Files have data
- Links (hard links) make it a DAG, not just a tree
  - – Soft links (aliases) are another name for an entry



```
sdcard          2013-12-10  23:51    lrwxrwxrwx     -> /mnt/sdcard
```

I/O & Storage Levels



Application/Service

High Level I/O

Streams

Low Level I/O

Handles

Syscalls

Registers

#4 file handle

File System

Descriptors

I/O Driver

Commands and Data Transfers

Disks, Flash, Controllers, DMA

Data Blocks

…

Directory

Structure

File

- Named permanent storage

Contains

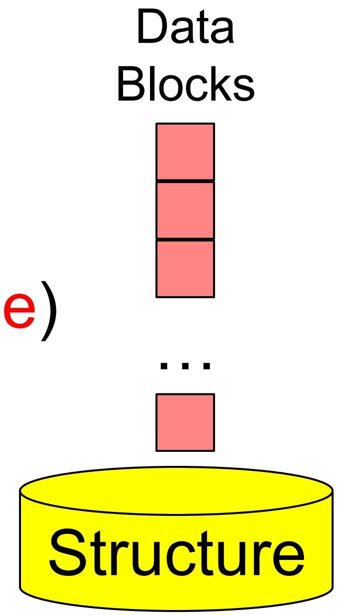
- Data
  - Blocks on disk somewhere
- Metadata (Attributes)
- Owner, size, last opened, …
- Access rights
  - R, W, X
  - Owner, Group, Other (in Unix systems)
  - Access control list in Windows system

File Handle

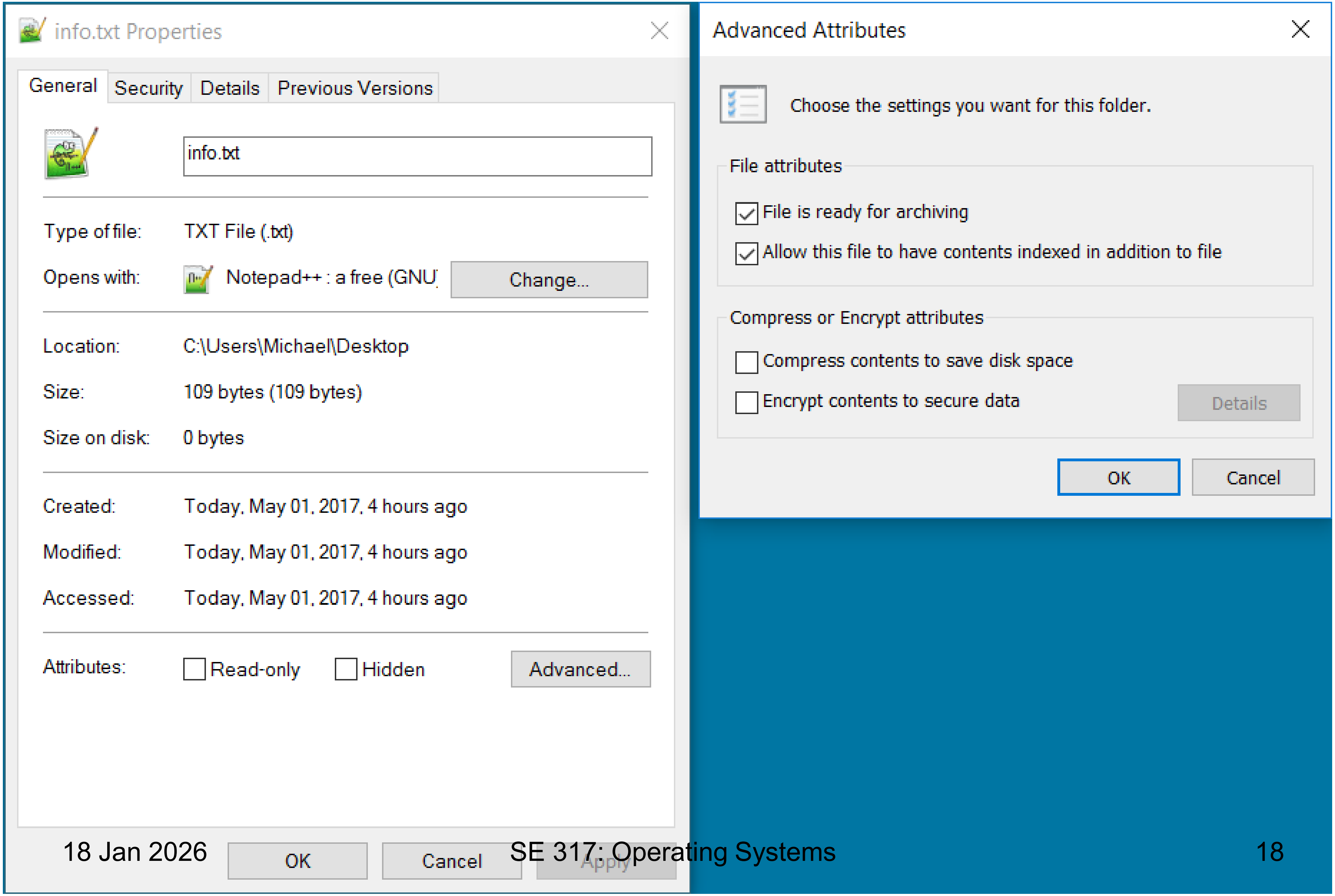
File Descriptor

FileObject (inode)

Position



File Attributes



info.txt Properties

info.txt

Type of file: TXT File (.txt)

Opens with: Notepad++ : a free (GNU

Location: C:\Users\Michael\Desktop

Size: 109 bytes (109 bytes)

Size on disk: 0 bytes

Created: Today, May 01, 2017, 4 hours ago

Modified: Today, May 01, 2017, 4 hours ago

Accessed: Today, May 01, 2017, 4 hours ago

Attributes: □ Read-only □ Hidden

Advanced Attributes

Choose the settings you want for this folder.

File attributes

☑ File is ready for archiving

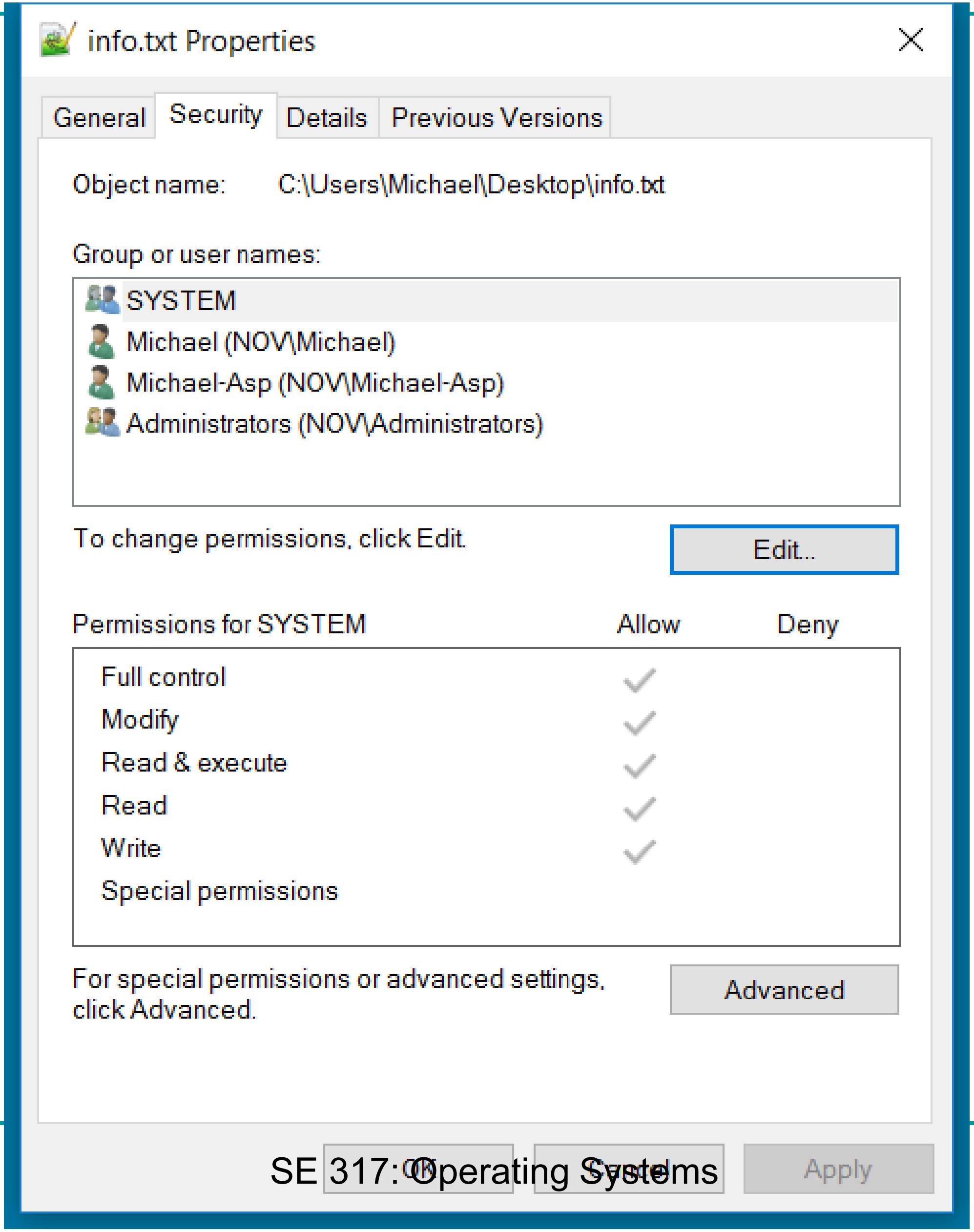
☑ Allow this file to have contents indexed in addition to file

Compress or Encrypt attributes

□ Compress contents to save disk space

□ Encrypt contents to secure data

File Attributes



Object name: C:\Users\Michael\Desktop\info.txt

Group or user names:

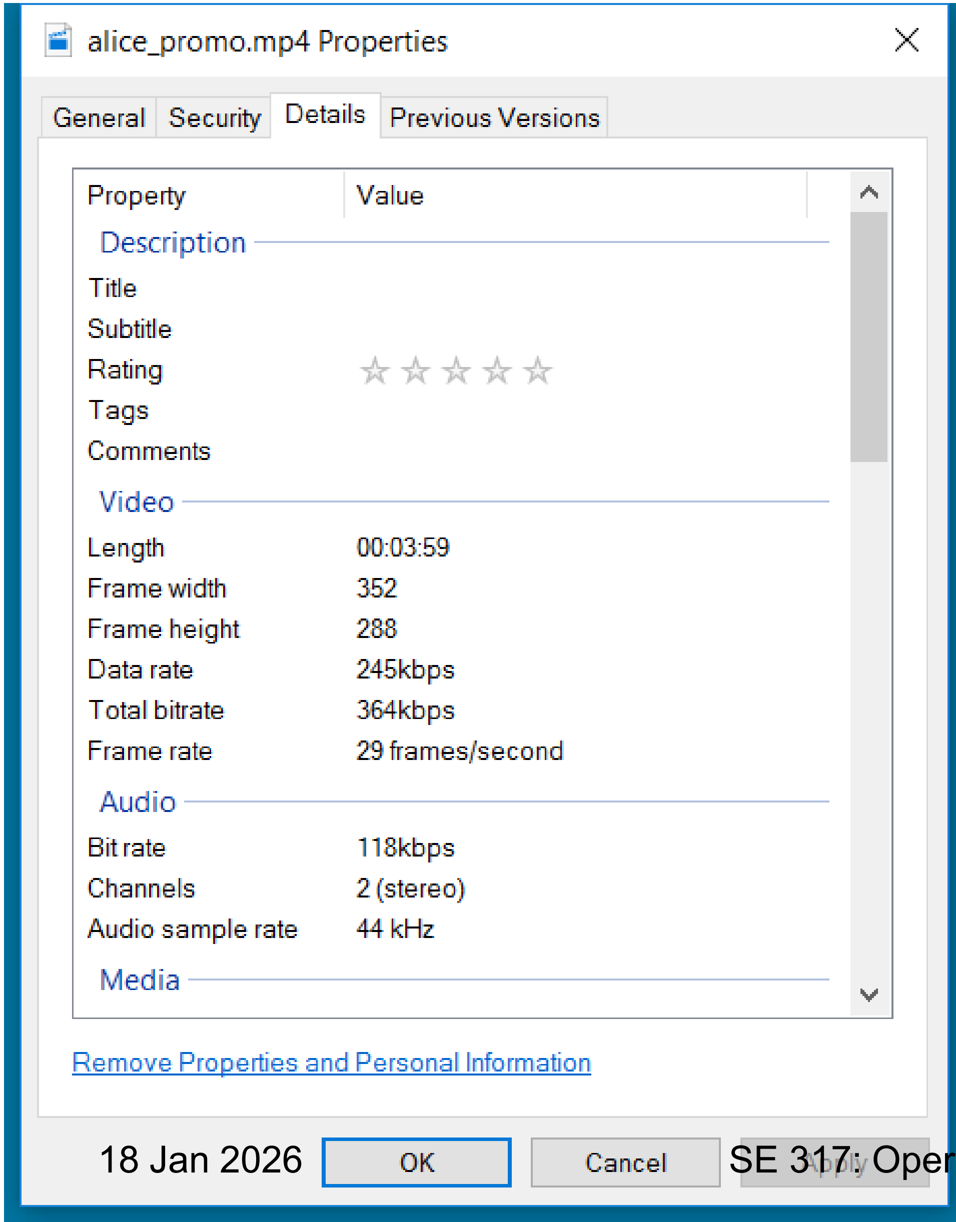
- SYSTEM
- Michael (NOV\Michael)
- Michael-Asp (NOV\Michael-Asp)
- Administrators (NOV\Administrators)

To change permissions, click Edit.

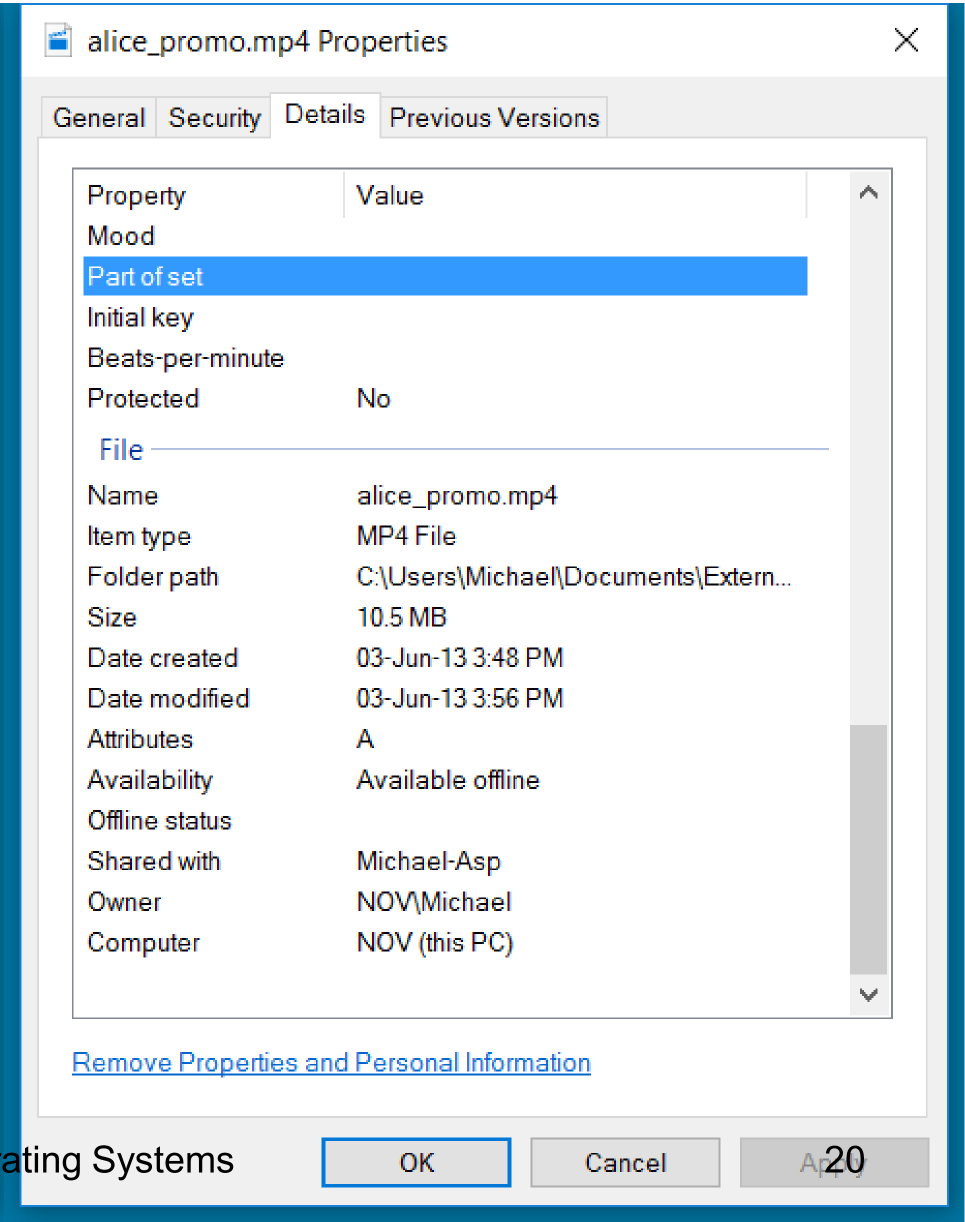
Permissions for SYSTEM	Allow	Deny
Full control	✓	
Modify	✓	
Read & execute	✓	
Read	✓	
Write	✓	
Special permissions		

For special permissions or advanced settings, click Advanced.

File Metadata

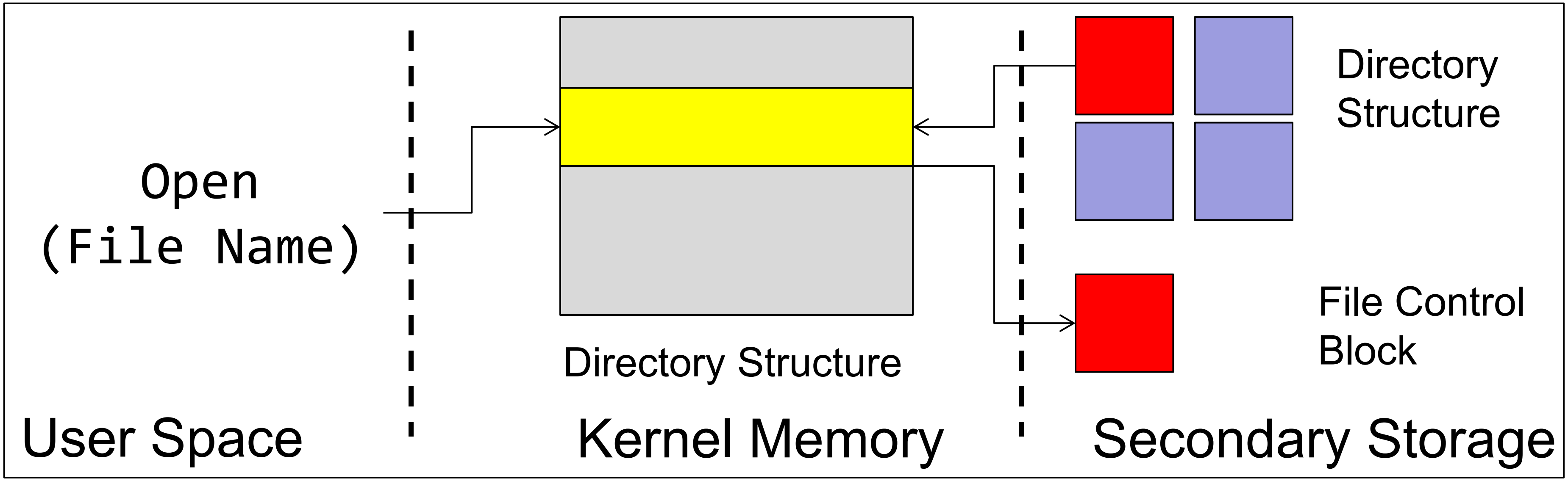

alice_promo.mp4 Properties
General
Security
Details
Previous Versions
Remove Properties and Personal Information
OK
Cancel
Apply

Property	Value
Description	
Title	
Subtitle	
Rating	☆☆☆☆☆
Tags	
Comments	
Video	
Length	00:03:59
Frame width	352
Frame height	288
Data rate	245kbps
Total bitrate	364kbps
Frame rate	29 frames/second
Audio	
Bit rate	118kbps
Channels	2 (stereo)
Audio sample rate	44 kHz
Media	


alice_promo.mp4 Properties
General
Security
Details
Previous Versions
Remove Properties and Personal Information
OK
Cancel
Apply

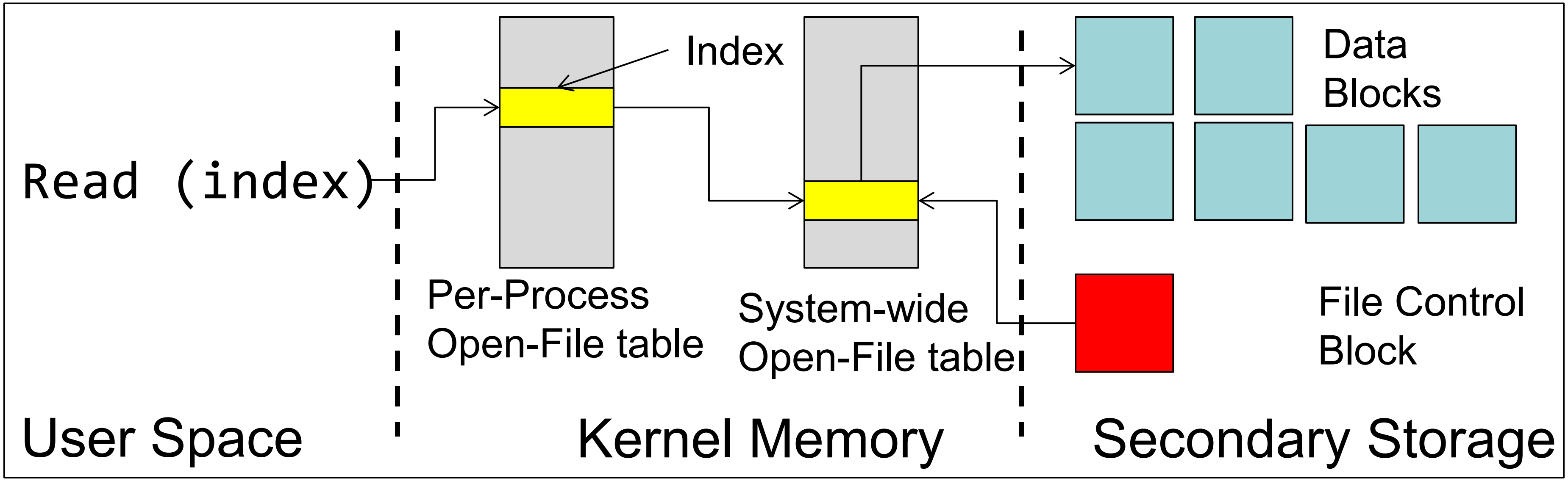
Property	Value
Mood	
Part of set	
Initial key	
Beats-per-minute	
Protected	No
File	
Name	alice_promo.mp4
Item type	MP4 File
Folder path	C:\Users\Michael\Documents\Extern...
Size	10.5 MB
Date created	03-Jun-13 3:48 PM
Date modified	03-Jun-13 3:56 PM
Attributes	A
Availability	Available offline
Offline status	
Shared with	Michael-Asp
Owner	NOV\Michael
Computer	NOV (this PC)

In-Memory File System Structures



- Open system call:
  - Resolves file name, finds file control block (inode)
  - Makes entries in per-process and system-wide tables
  - Returns index (called “file handle”) in open-file table

In-Memory File System Structures



- Read/write system calls:
  - – Use file handle to locate inode
  - – Perform appropriate reads or writes

So Far

- File Systems
  - Introduction to File Systems
  - Very simply file system
  - FAT
  - Inodes
  - Unix Fast File System (FFS)

A simple File System: Ingredients

Blocks

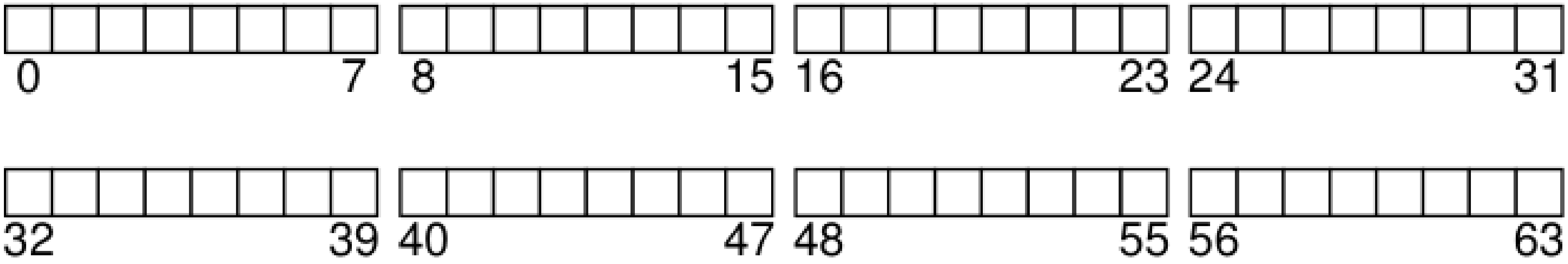
- 4 KB
- 64 total blocks
  - Numbered 0-63

Data space

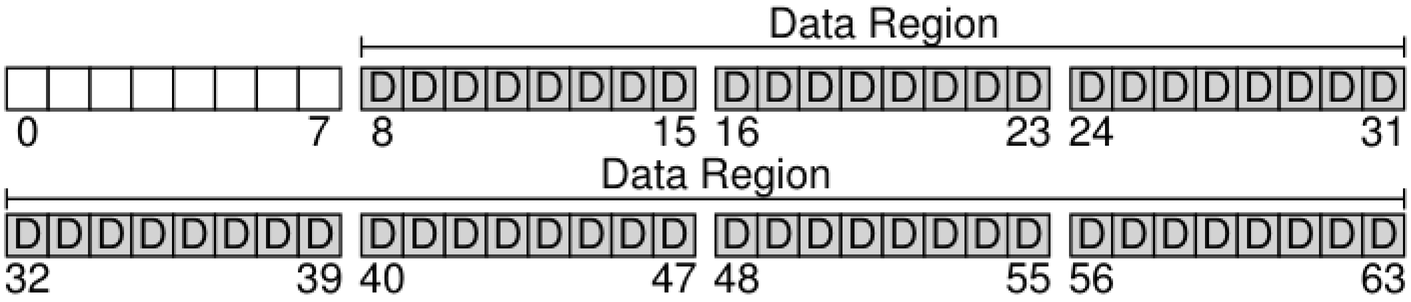
Metadata space

Superblock

Our blocks (4KB)



Data Region

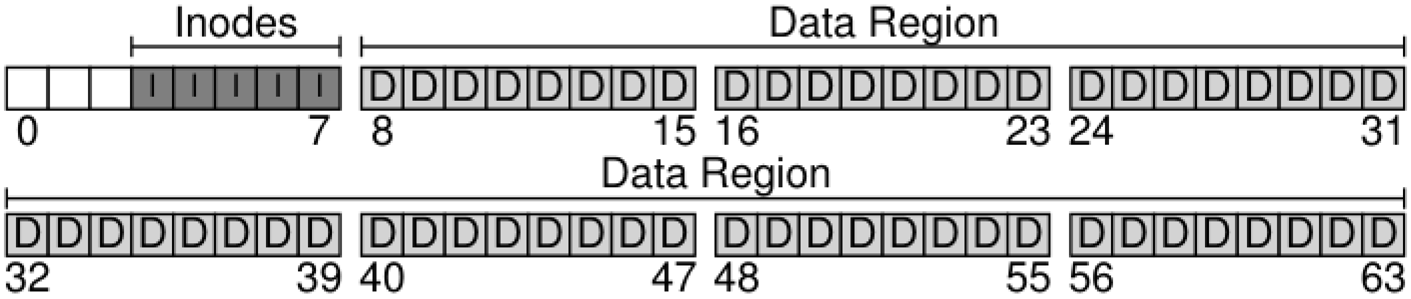

Data Region
D D D D D D D D D D D D D D D D D D D D D D D D
0
7
8
15
16
23
24
31
Data Region
D D D D D D D D D D D D D D D D D D D D D D D D D D D D D D D D
32
39
40
47
48
55
56
63

Blocks 8-63 will be data

56 total

Rest will be metadata

Inode region – What’s an inode?



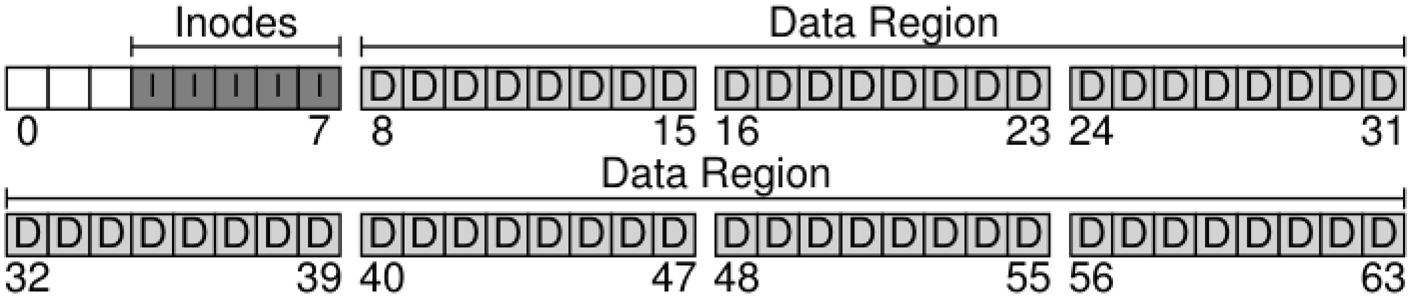
Data structure holds metadata about file

- Size
- Owner
- Access rights
- Access/modify times

Typically stored in a table

- Small, say 256 bytes

Inode region



Use 5 blocks for inodes

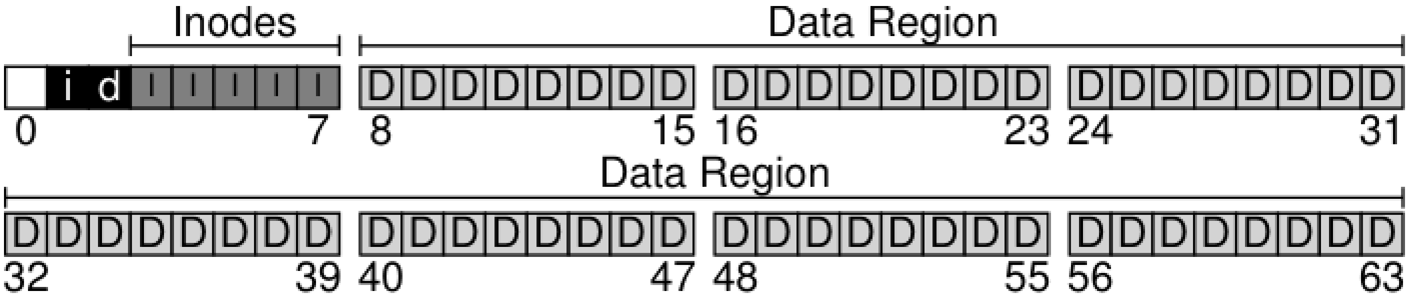
Each 4KB block holds 16 inodes

- 4096 / 256 = 16

Total 80 inodes

- Can’t have more than 80 files or directories

Used/Free tracking



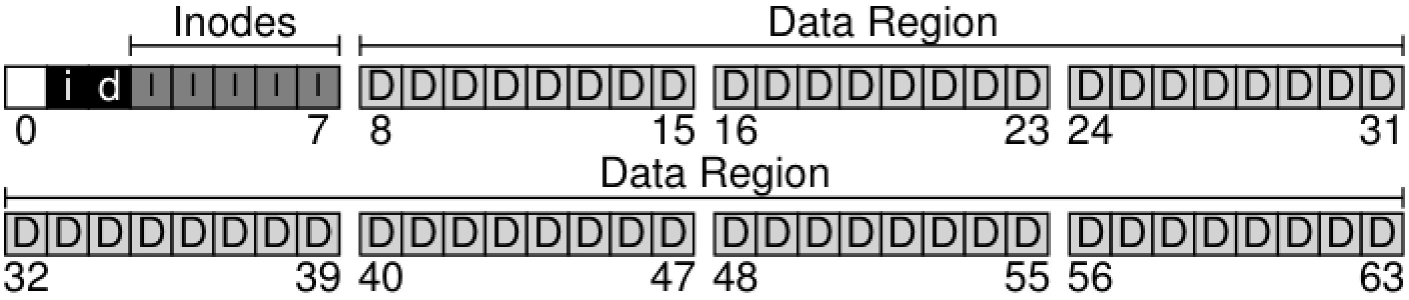
Need to track if an inode is in used

Need to track if a data block is in use

Use a **bitmap** for each

- 0 if inode/block is free
- 1 if inode/block in use
- Example: 1011 0011

Used/Free tracking



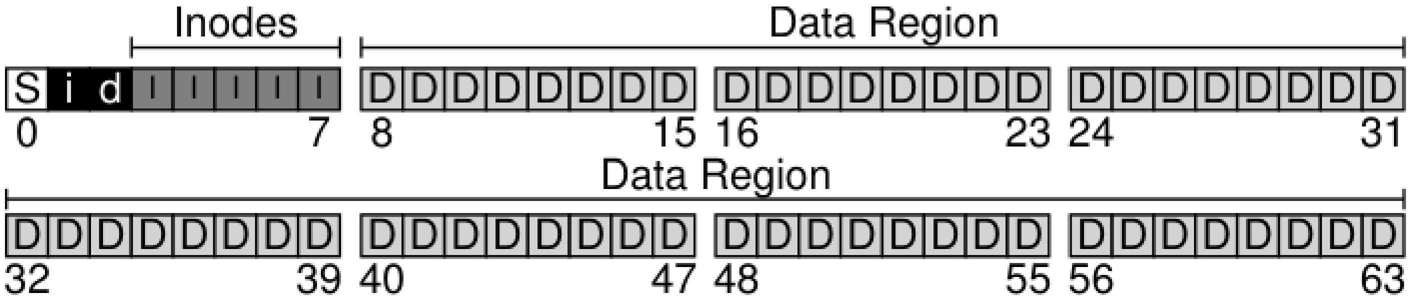
Block 1 tracks inodes

- Could really track 32K inodes, but let’s be simple

Block 2 tracks data blocks

- Could really track 32K blocks, but let’s be simple

Superblock



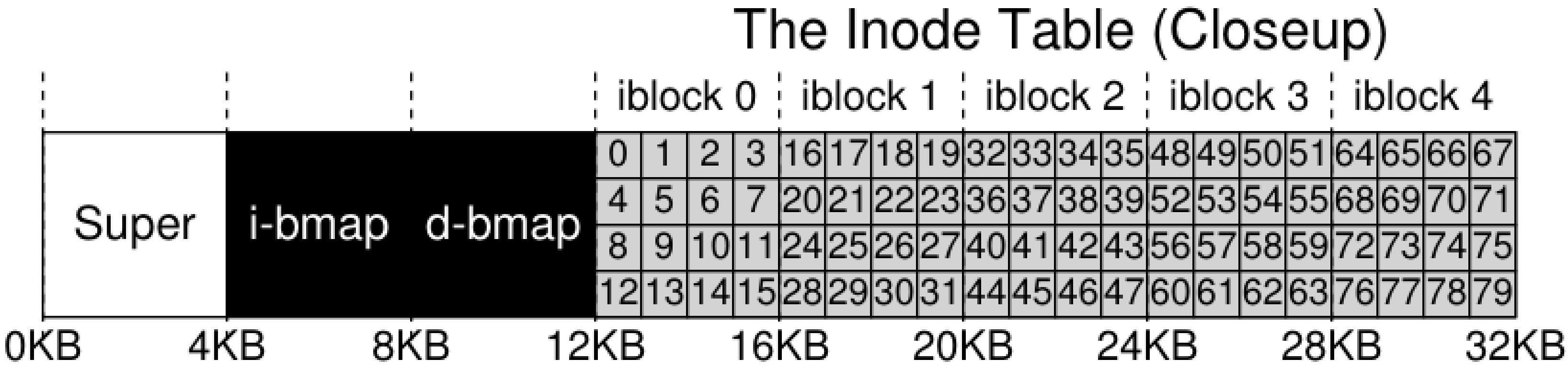
Block 0 contains basic metadata

- Called Superblock

Contains info about the file system

- How many inodes
- How many blocks
- Where things are
- IDs, etc.

Inode drill down



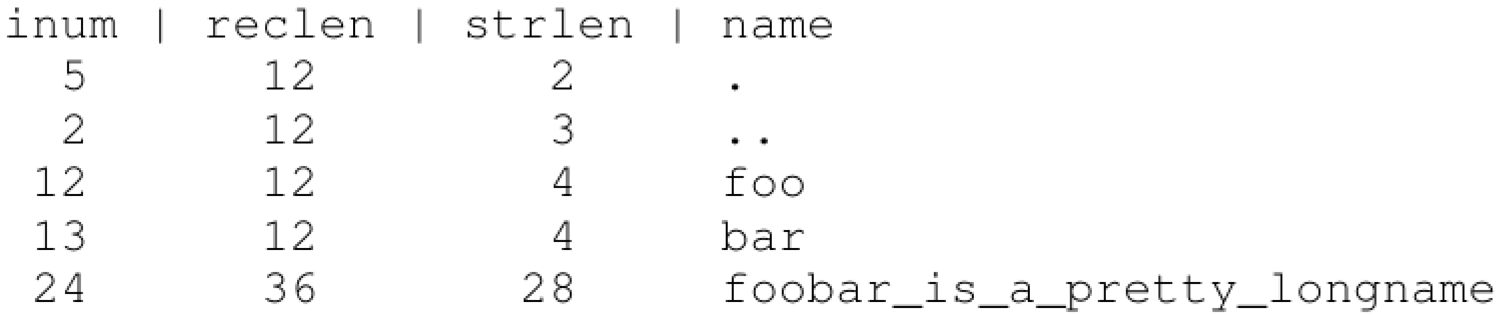
Each inode has an implicit number

- i-number

Can index the file by taking i-number X sizeof(inode)

- To read inode 32, start at byte 32 X sizeof(inode) = 8192

What is a directory?



```
inum | reclen | strlen | name
  5      12        2     .
  2      12        3     ..
 12      12        4     foo
 13      12        4     bar
 24      36       28     foobar_is_a_pretty_longname
```

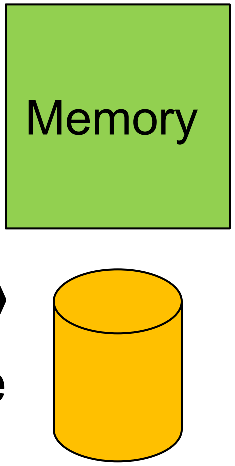
- Contains records of files and directories inside
- i-number of the file/directory
- Reclen – how many bytes in the record
- Strlen – how many bytes in the name of the file/directory
- Name – the actual name (≤ reclen)

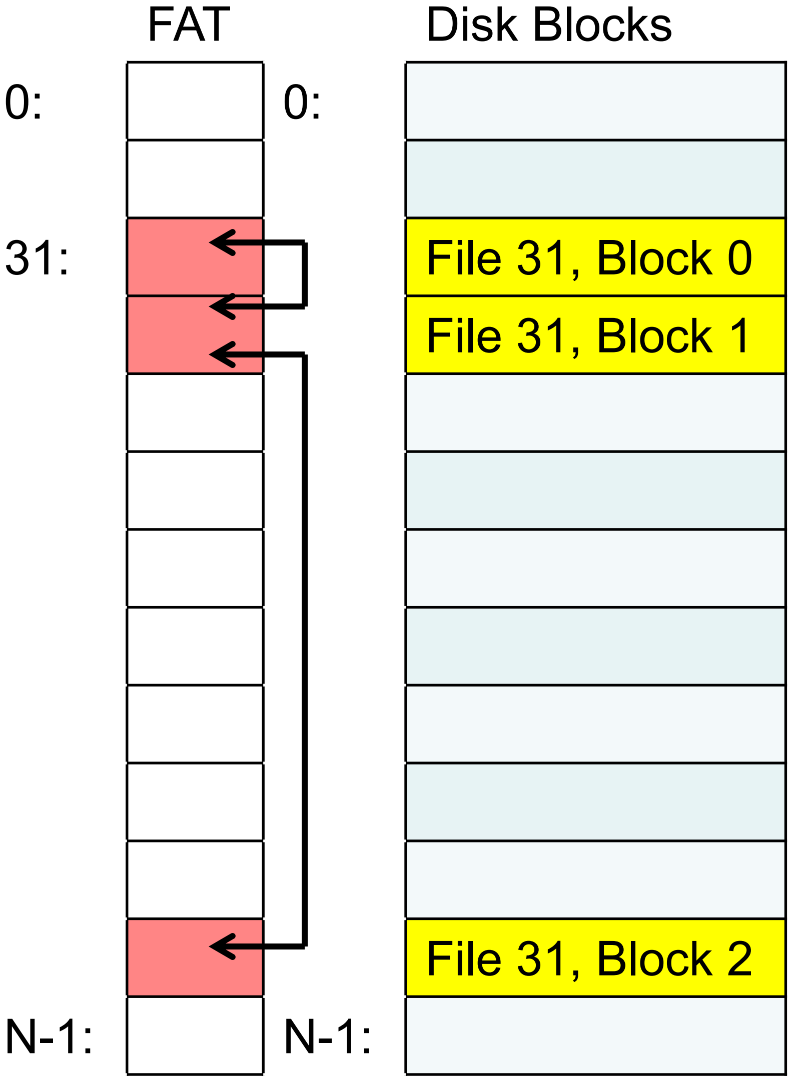
So Far

- File Systems
  - Introduction to File Systems
  - Very simply file system
  - FAT
  - Inodes
  - Unix Fast File System (FFS)

Our first filesystem: FAT (File Allocation Table)

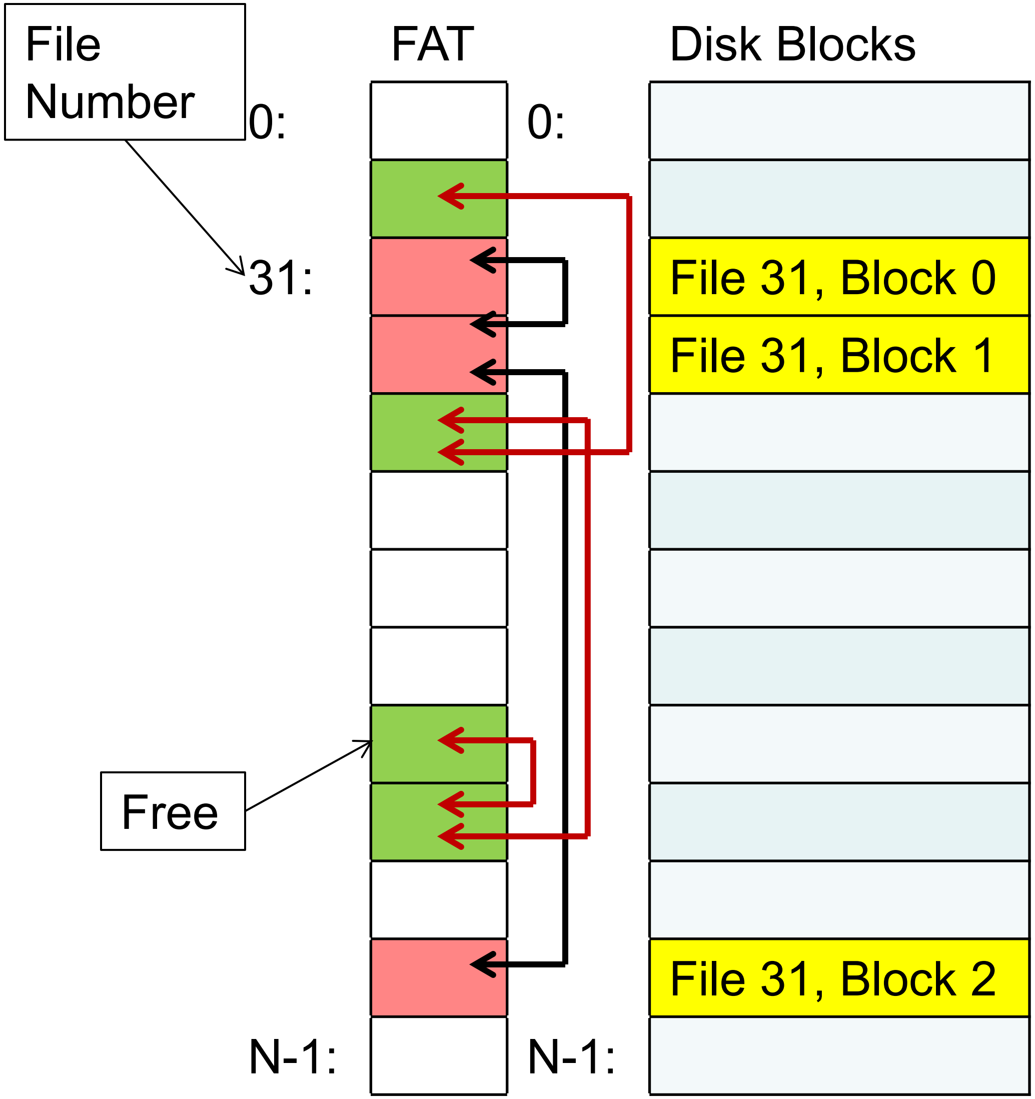
- Assume we have a way to translate a path to a "file number"
  - i.e., a directory structure
- Disk Storage is a collection of Blocks
  - Just hold file data
- Ex: `file_read` $31, \langle 2, x \rangle$
  - Index into FAT with file number
  - Follow linked list to block
  - Read the block from disk into mem





FAT Properties

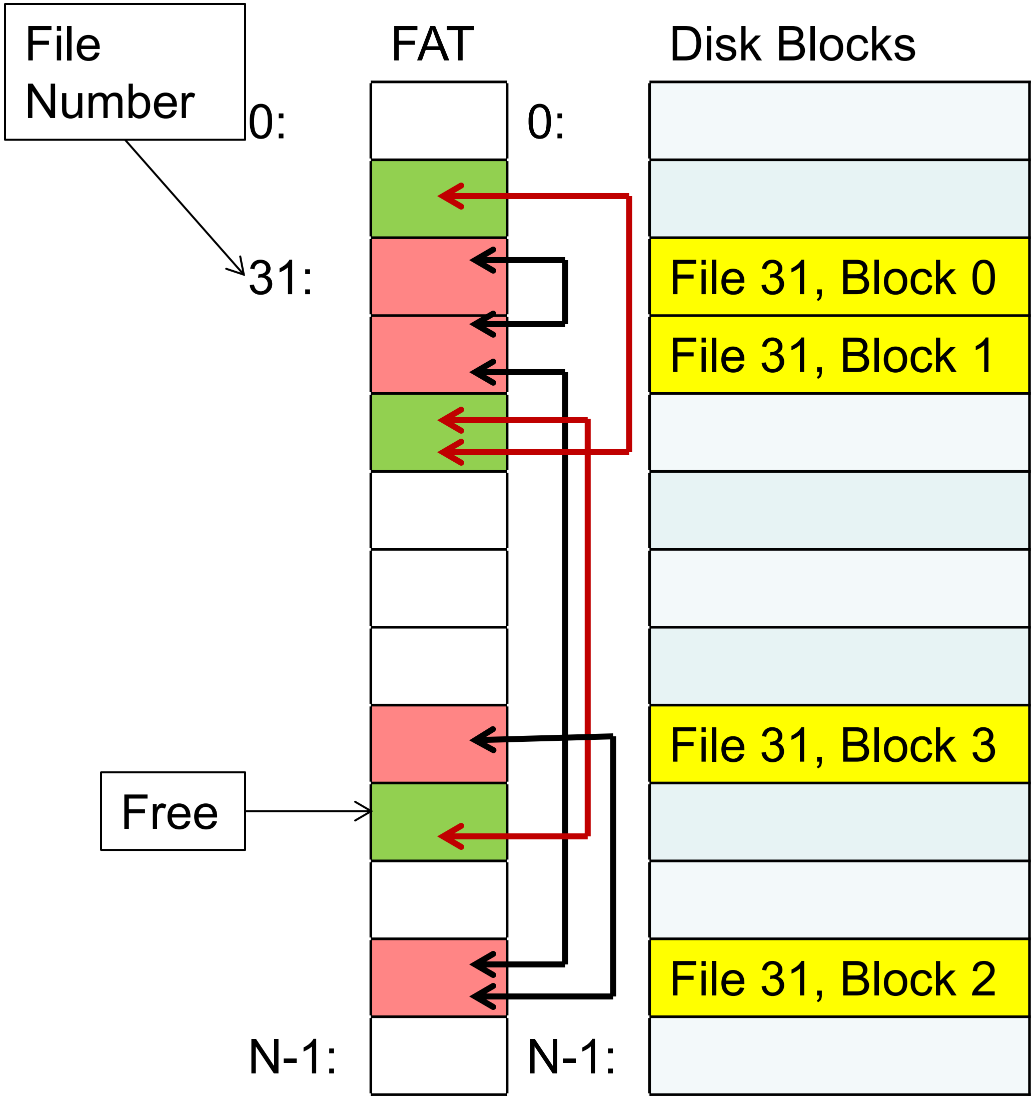
- File is collection of disk blocks
- FAT is linked list 1-1 with blocks
- File Number is index of root of block list for the file
- File offset ($o = B{:}x$)
- Follow list to get block #
- Unused blocks ⇔ FAT free list



Writing a File Block

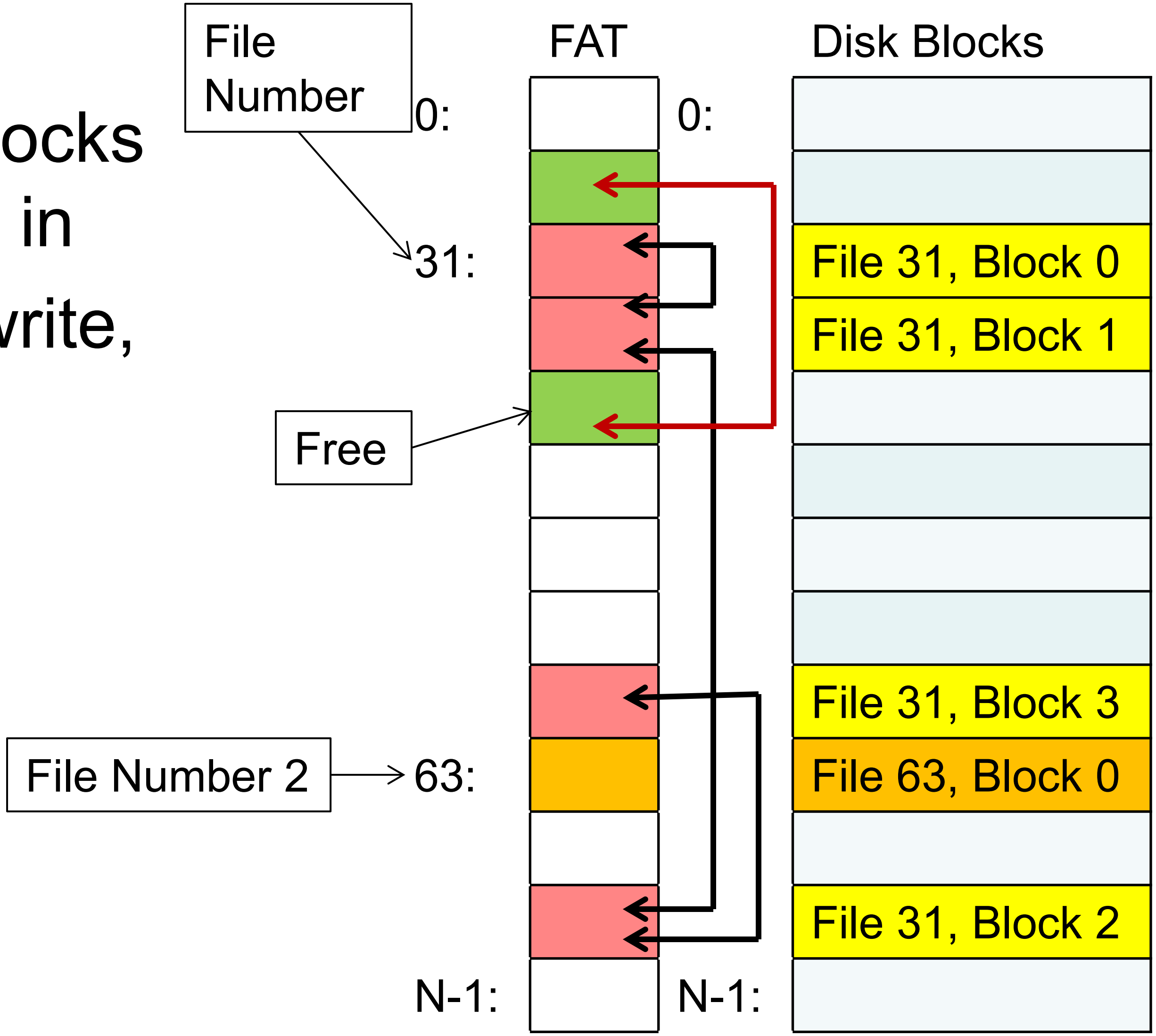
Ex: `file_write`$(51, \langle 3, y\rangle)$

- Grab blocks from free list
- Linking them into file



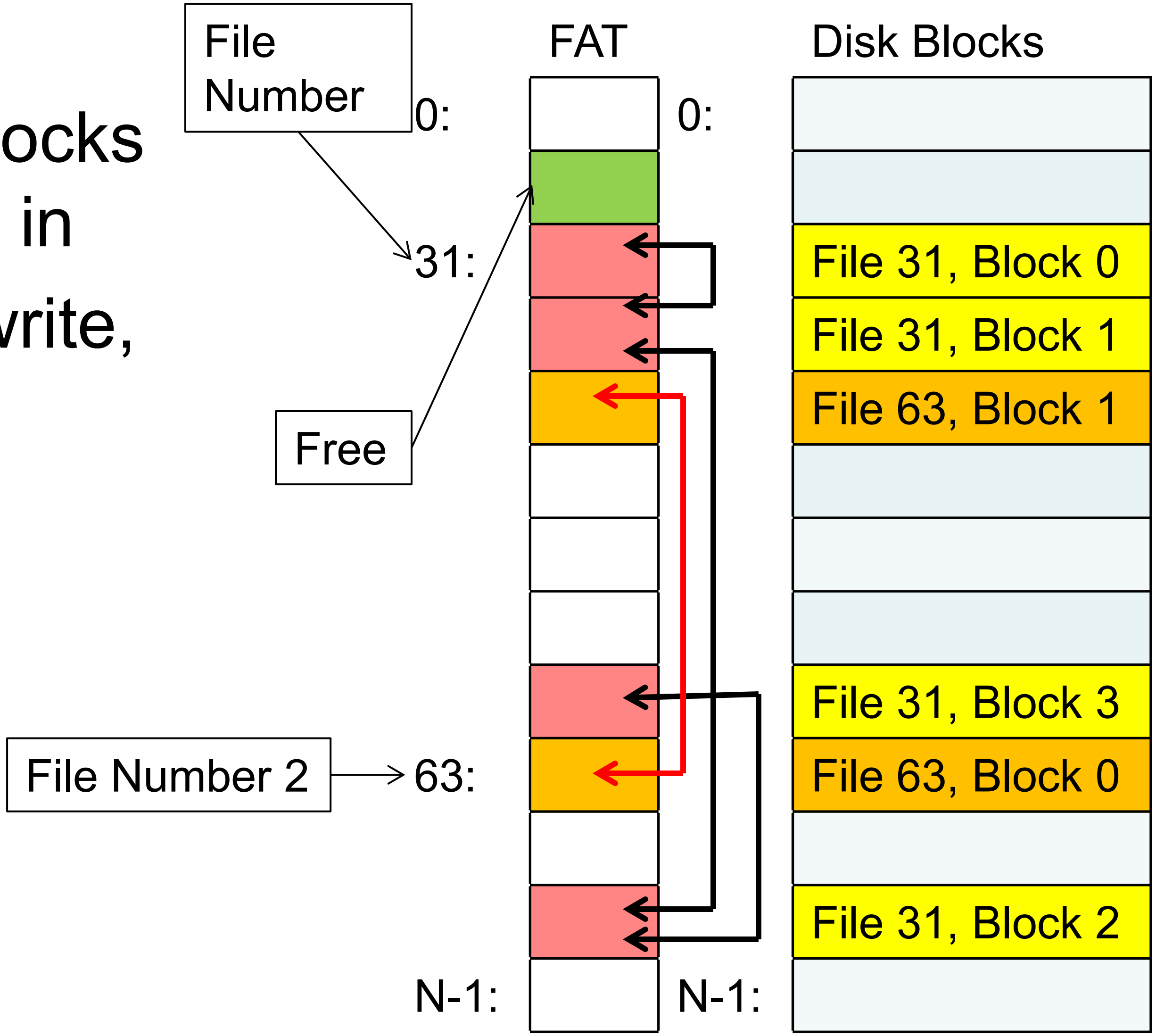
Create a New File

- Grow file by allocating free blocks and linking them in
- Ex: Create file, write, write



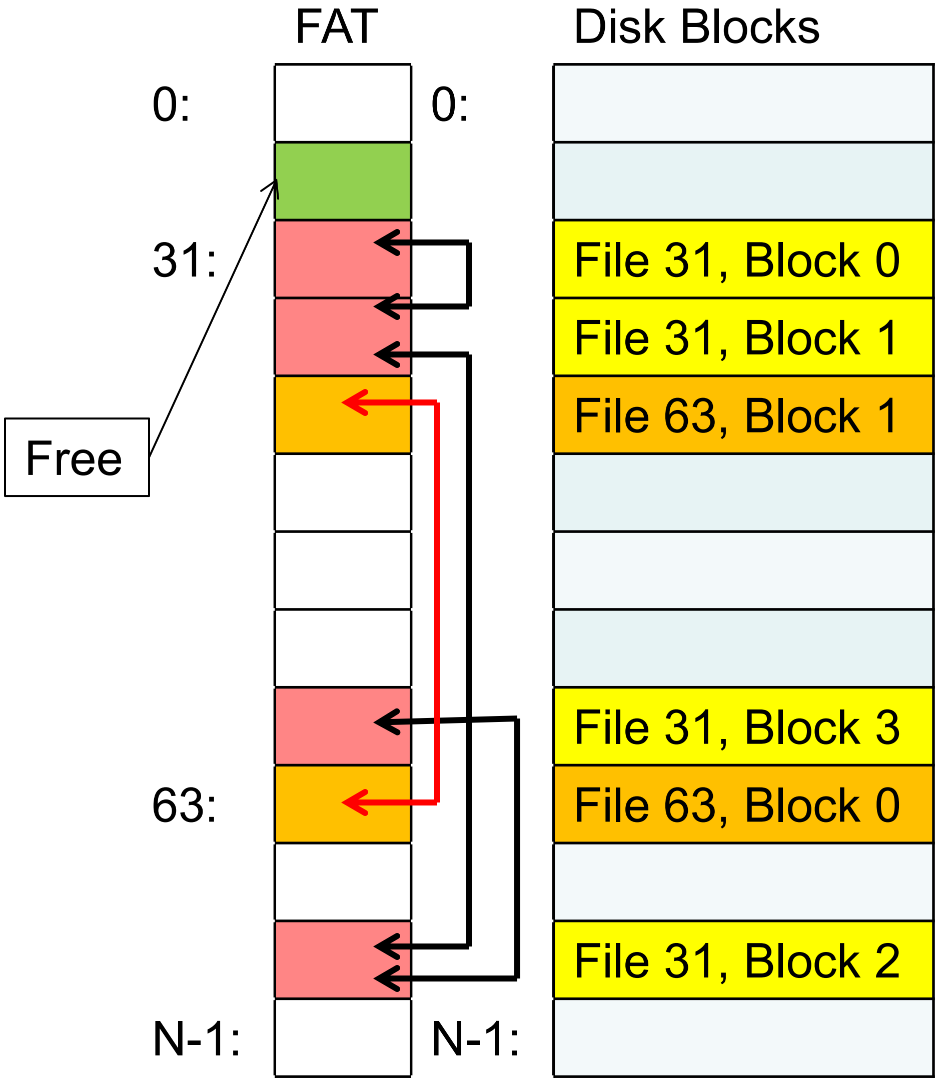
Create a New File

- Grow file by allocating free blocks and linking them in
- Ex: Create file, write, write



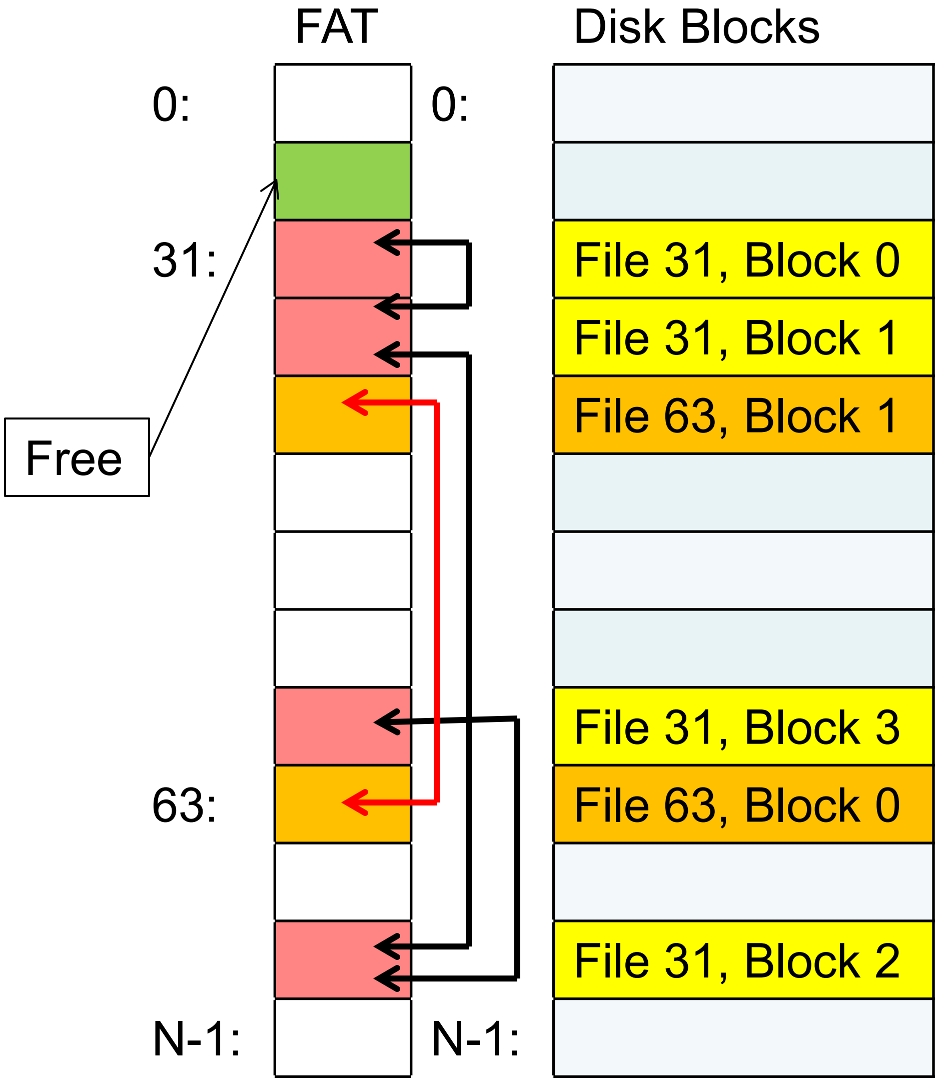
FAT Assessment

- Used in DOS, Windows, USB drives, …
- Where is FAT stored?
  - On disk, restore on boot, copy in memory
- What happens when you format a disk?
  - Zero the blocks, link up the FAT free-list
- Simple

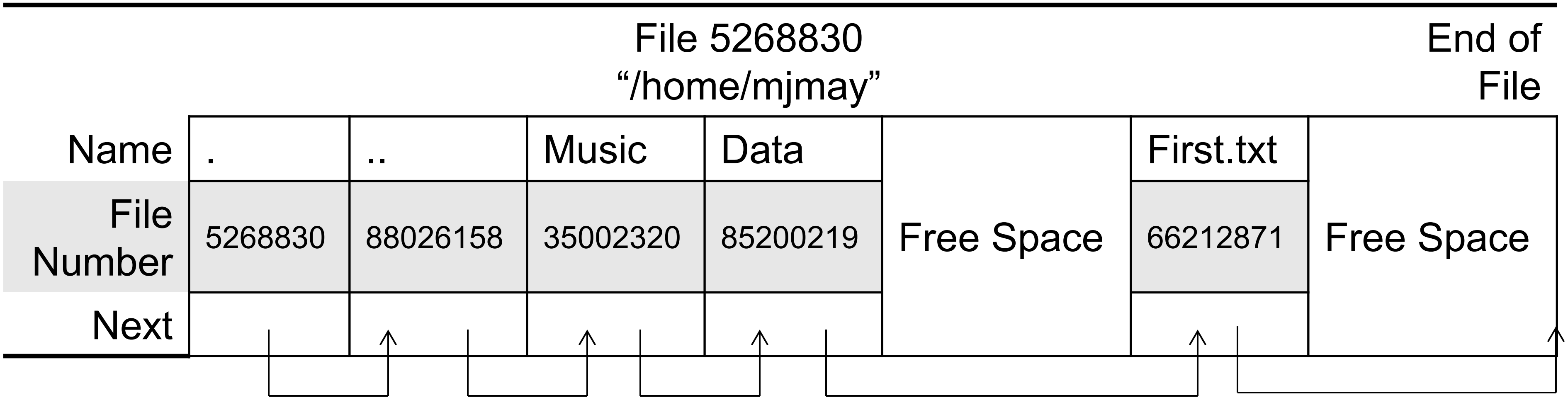


FAT Assessment

- Time to find block (large files)?
- Free list usually just a bit vector (here's it's a linked list).
- Block layout for file?
- Sequential Access?
- Random Access?
- Fragmentation?
- Small files?
- Big files?



What about the Directory?



File 5268830
“/home/mjmay”

End of
File

Name	.	..	Music	Data	Free Space	First.txt	Free Space
File Number	5268830	88026158	35002320	85200219		66212871	
Next							

- Essentially a file containing `<file_name:file_number>` mappings
- Free space for new entries
- In FAT: attributes kept in directory (!)
- Each directory a linked list of entries
- Where do you find root directory (“/”)?

Directory Structure

- How many disk accesses to resolve "`/my/book/count`"?
  - Read in file header for root (fixed spot on disk)
  - Read in first data block for `root`
    - Table of file name/index pairs. Search linearly – ok since directories typically very small
  - Read in file header for "`my`"
  - Read in first data block for "`my`"; search for "`book`"
  - Read in file header for "`book`"
  - Read in first data block for "`book`"; search for "`count`"
  - Read in file header for "`count`"
- Current working directory: Per-address-space pointer to a directory (inode) used for resolving file names
  - Allows user to specify relative filename instead of absolute path (say CWD="`/my/book`" can resolve "`count`")

Big FAT security holes

- FAT has no access rights
- FAT has no header in the file blocks
- Just gives an index into the FAT
  - – (file number = block number)

So Far

- File Systems
  - Introduction to File Systems
  - Very simply file system
  - FAT
  - Inodes
  - Unix Fast File System (FFS)

Characteristics of Files

- Most files are small
- Most of the space is occupied by the rare big ones

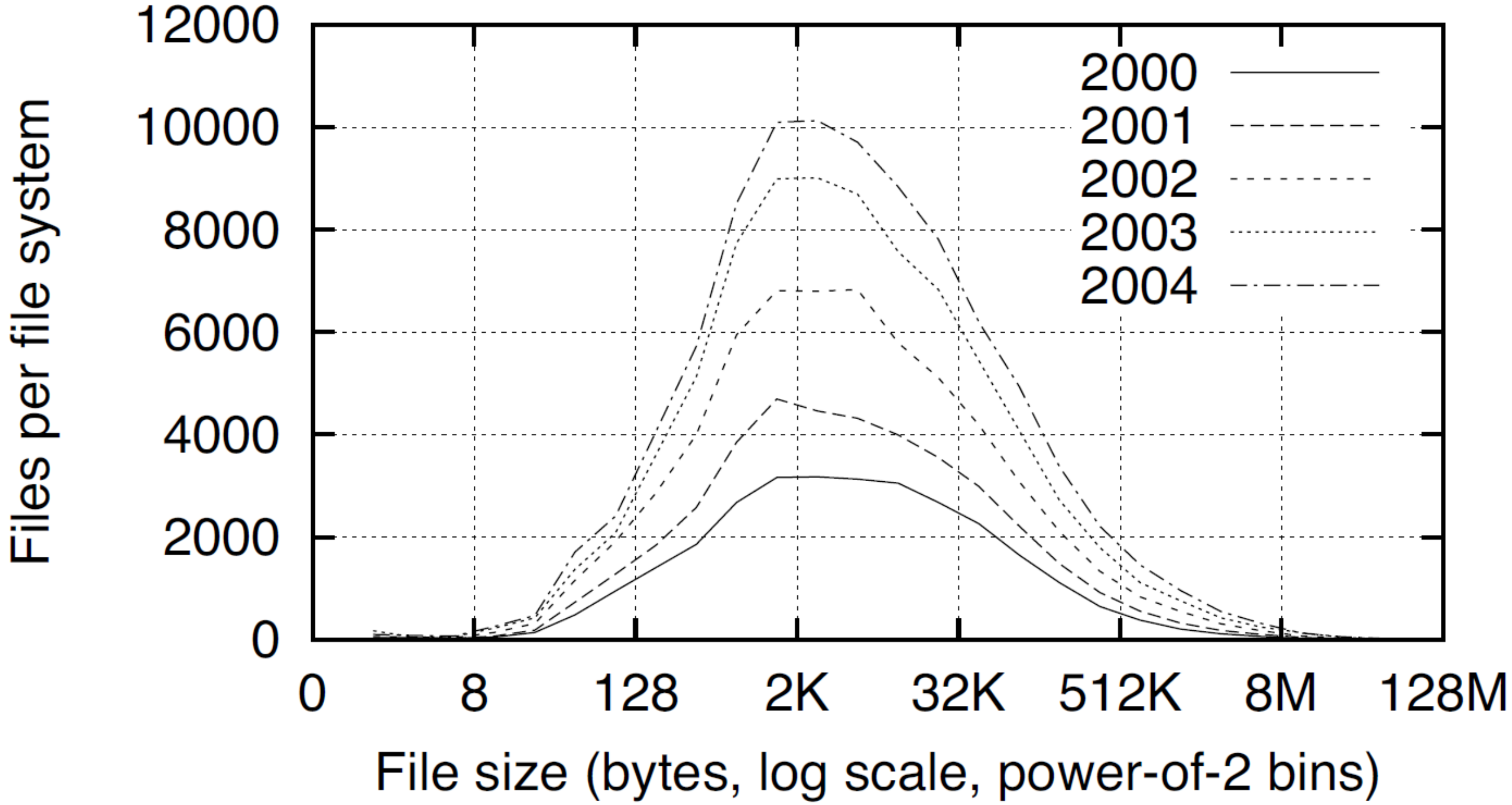


Figure 2: Histograms of files by size

AGRAWAL, N., BOLOSKY, W. J., DOUCEUR, J. R., AND LORCH, J. R. 2007. A five-year study of file-system metadata. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, San Jose, CA, 31–45.

Characteristics of Files

- Most files are small
- Most of the space is occupied by the rare big ones

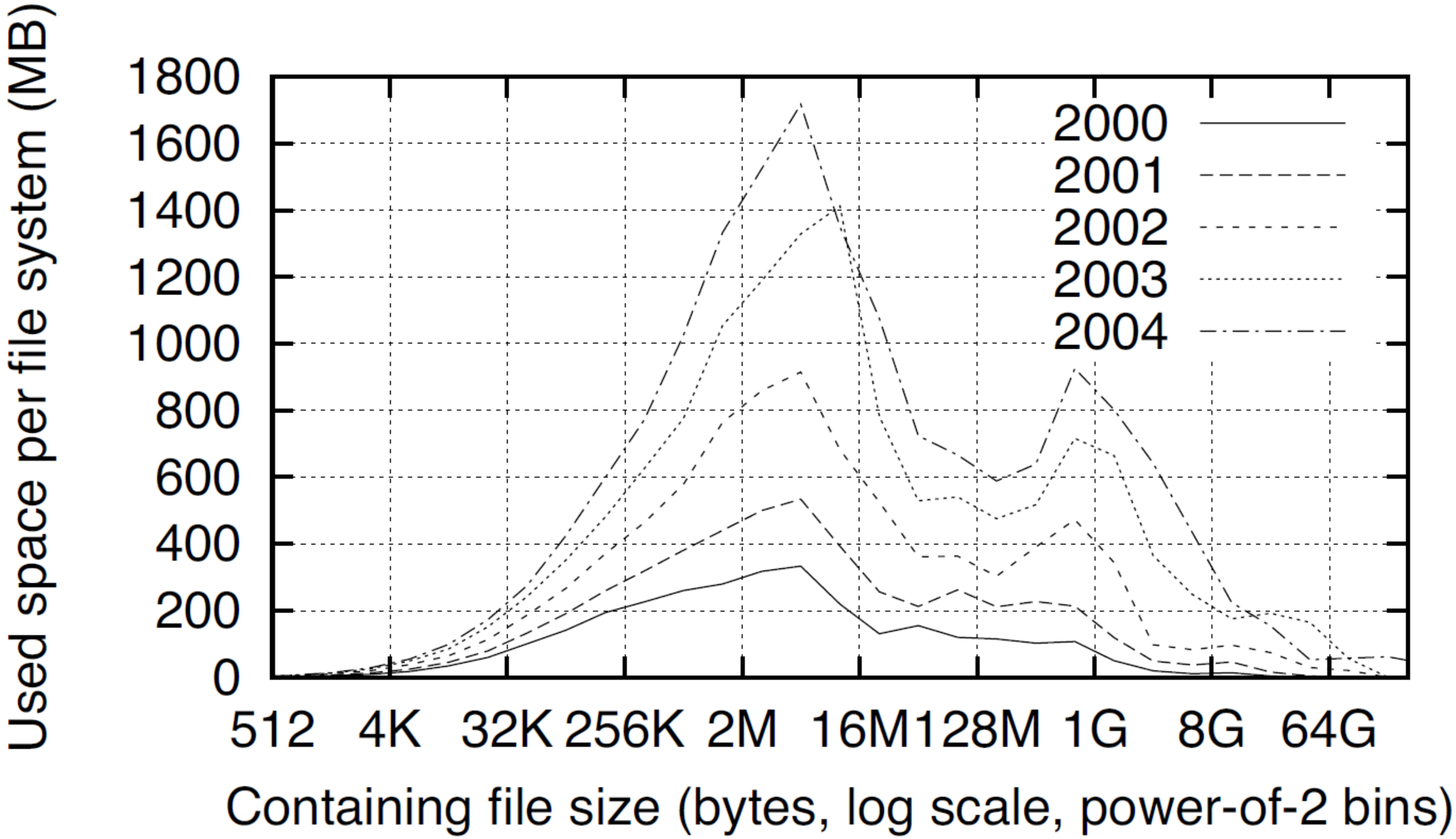
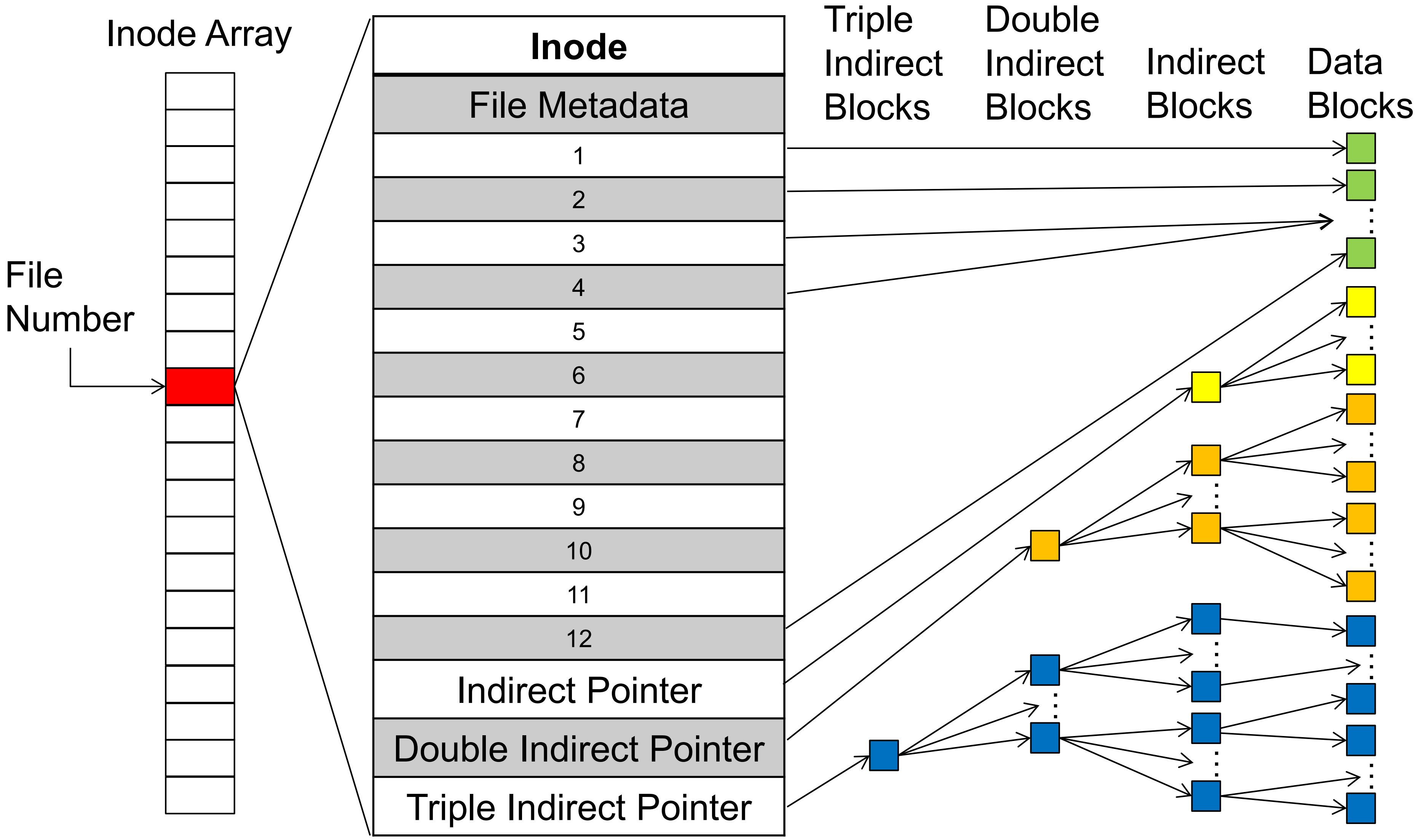


Figure 4: Histograms of bytes by containing file size

Agrawal, N., Bolosky, W. J., Douceur, J. R., and Lorch, J. R. 2007. A five-year study of file-system metadata. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, San Jose, CA, 31–45.

Meet the Inode

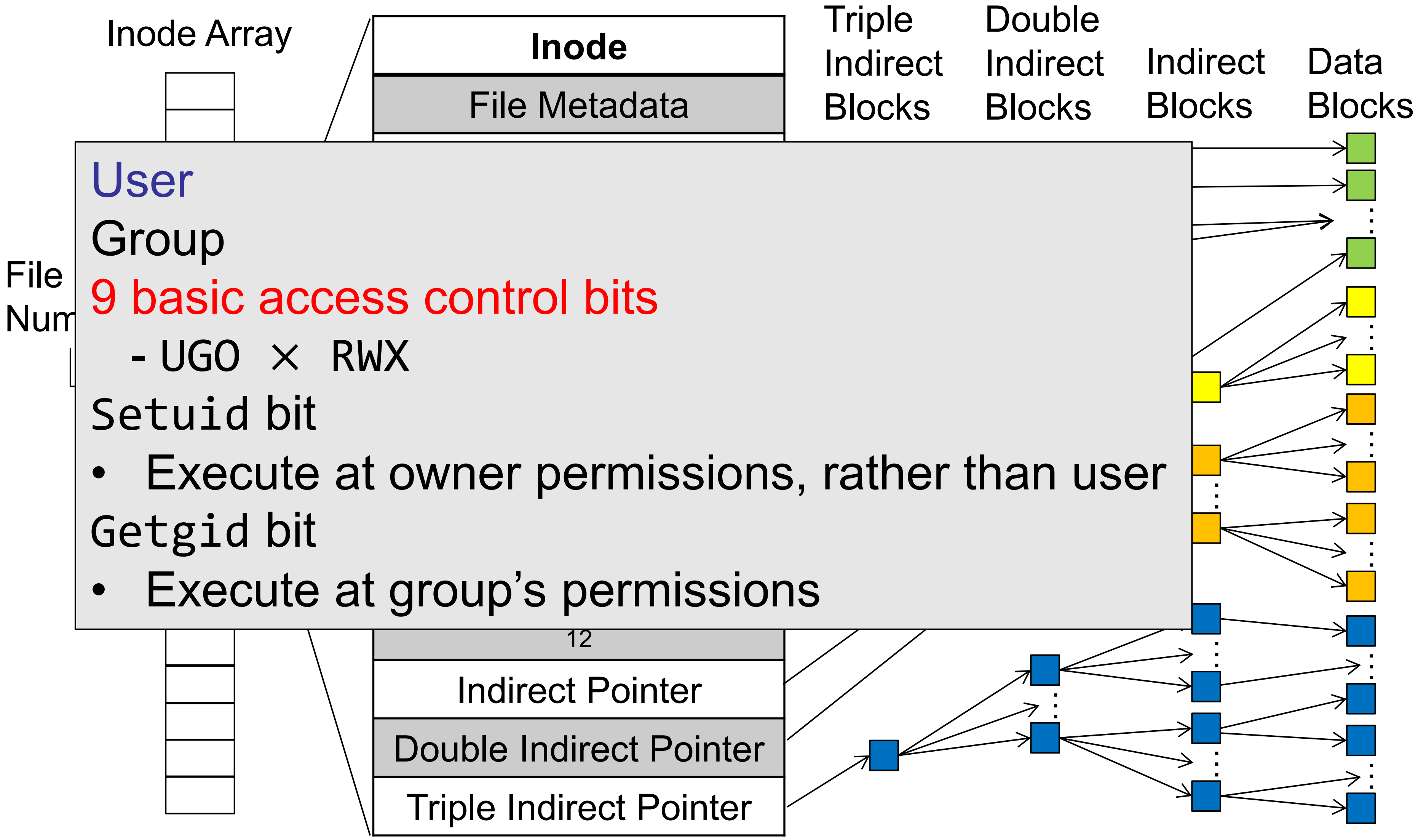


Unix Fast File System

- Original inode format appeared in BSD 4.1 (1981)
  - *Berkeley Standard Distribution Unix*
  - Similar structure for Linux ext2/3
- File Number is index into inode arrays
- Multi-level index structure
  - Great for small and large files
  - Asymmetric tree with fixed sized blocks
- Metadata associated with the file
  - Rather than in the directory that points to it (FAT)
- UNIX FFS: BSD 4.2 (1983): Locality Heuristics
  - Block group placement
  - Reserve space
- Scalable directory structure

Image Source: https://en.wikipedia.org/w/index.php?curid=1193267

File Attributes



User

Group

9 basic access control bits

- UGO × RWX

Setuid bit

- Execute at owner permissions, rather than user

Getgid bit

- Execute at group’s permissions

Data Storage

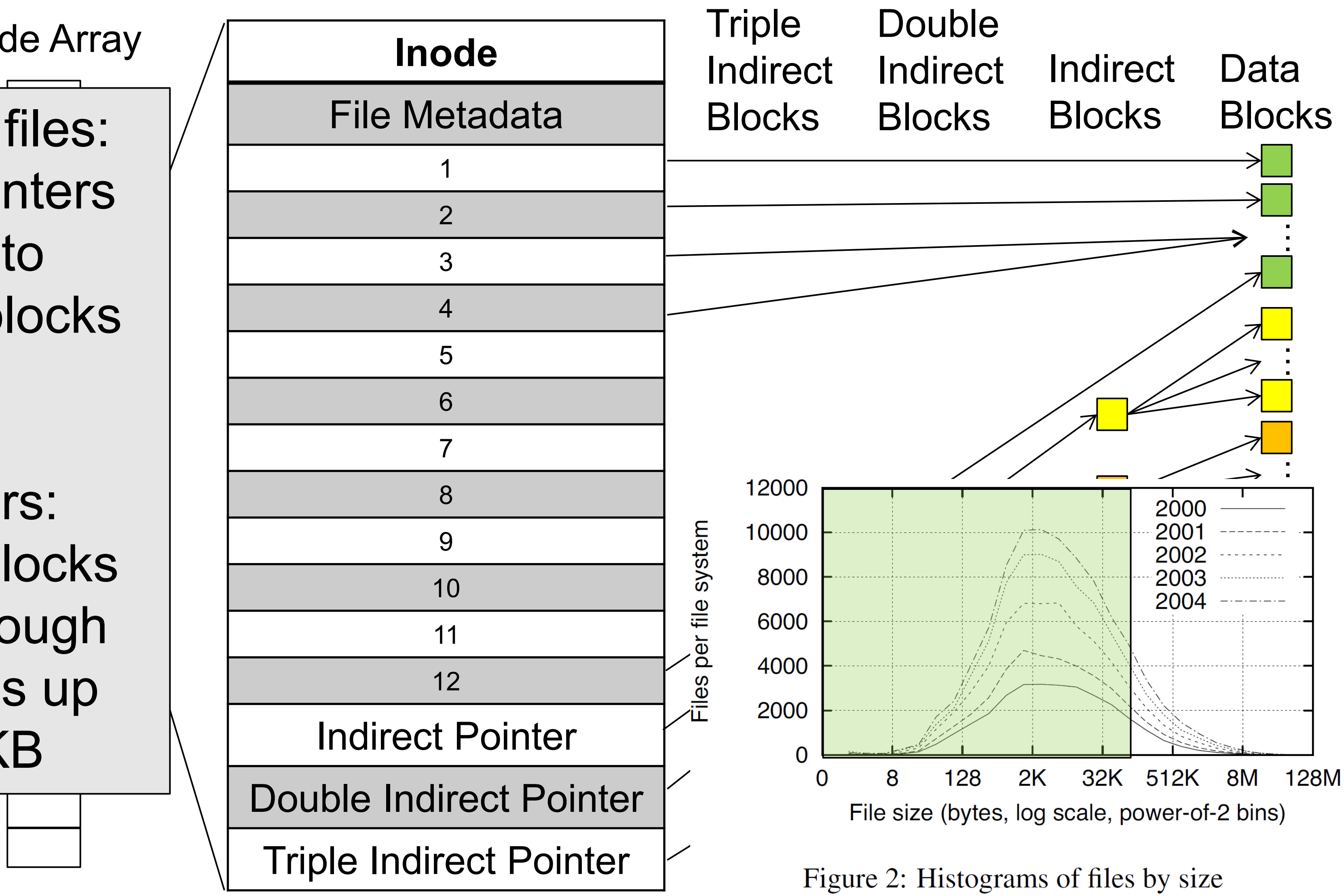


Figure 2: Histograms of files by size

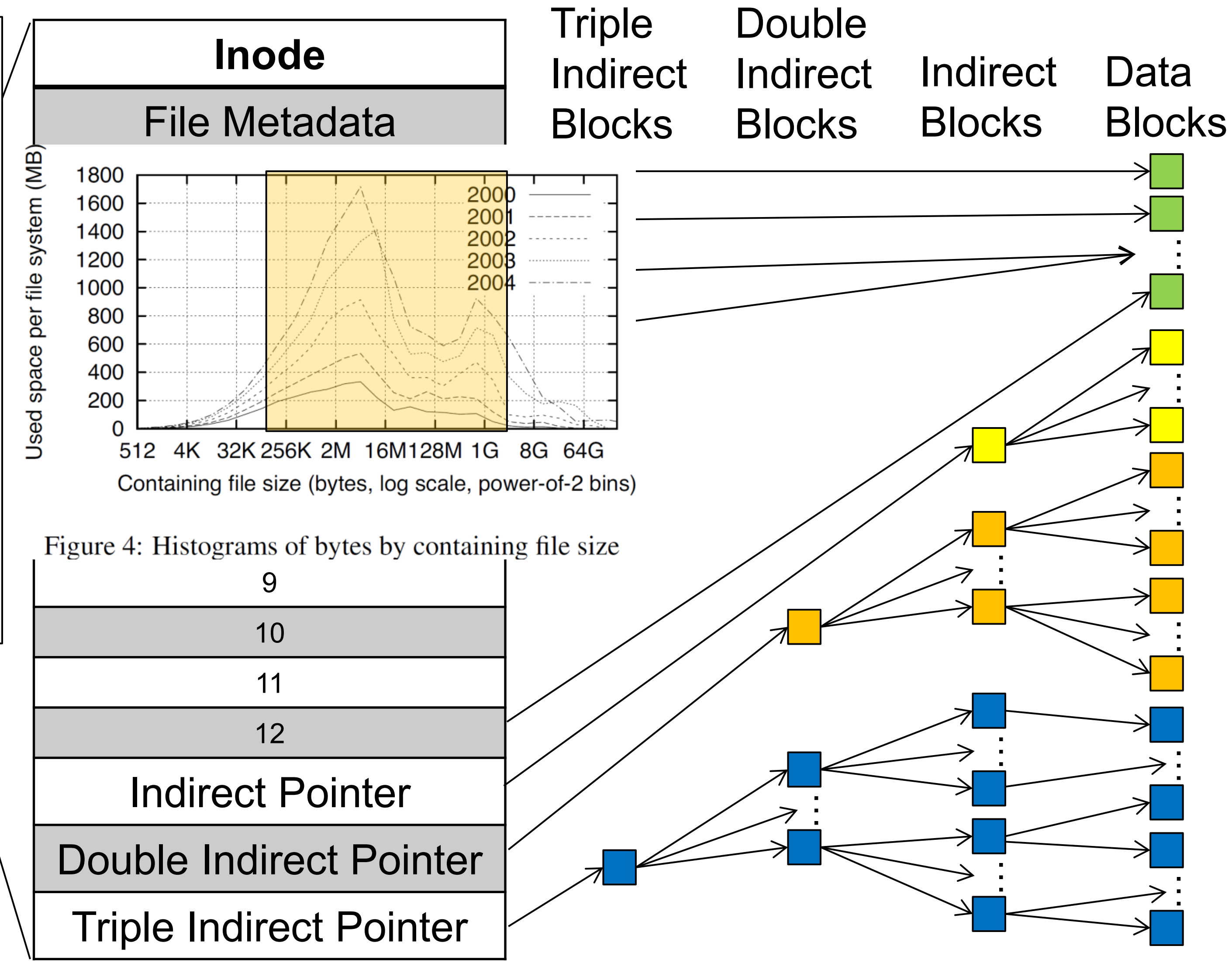
Small files: 12 pointers direct to data blocks

Direct pointers: 4KB blocks ➔ Enough for files up to 48KB

Data Storage

Indirect pointers

- Point to a disk block containing only pointers
- 4KB blocks ➔1024 pointers
  - 4 MB at Level 2
  - 4GB at Level 3
  - 4TB at Level 4



Inode

File Metadata

9

10

11

12

Indirect Pointer

Double Indirect Pointer

Triple Indirect Pointer

Triple Indirect Blocks

Double Indirect Blocks

Indirect Blocks

Data Blocks

Used space per file system (MB)
1800
1600
1400
1200
1000
800
600
400
200
0
512
4K
32K
256K
2M
16M
128M
1G
8G
64G
Containing file size (bytes, log scale, power-of-2 bins)
2000
2001
2002
2003
2004

Figure 4: Histograms of bytes by containing file size

So Far

- File Systems
  - Introduction to File Systems
  - Very simply file system
  - FAT
  - Inodes
  - Unix Fast File System (FFS)

UNIX BSD 4.2 (1983)

- Same as BSD 4.1 (same file header and triply indirect blocks), except incorporated ideas from Cray DEMOS:
  - Uses bitmap allocation in place of freelist
  - Attempt to allocate files contiguously
  - 10% reserved disk space
  - Skip-sector positioning (soon)

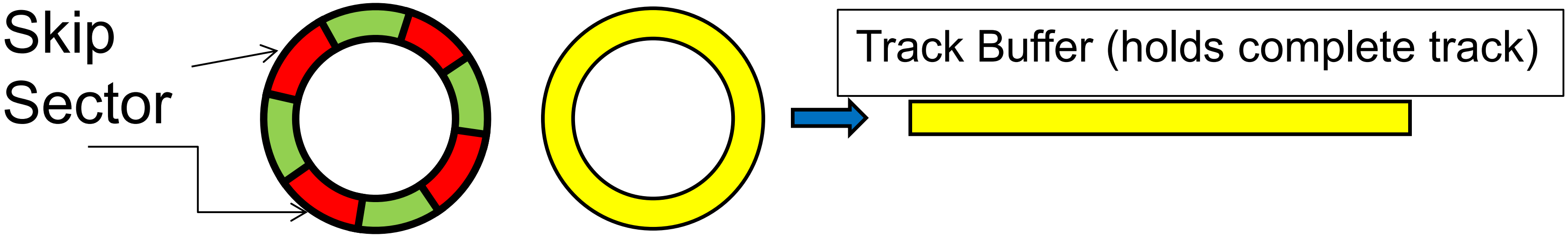
By Clemens PFEIFFER - Own work, CC BY 2.5, https://commons.wikimedia.org/w/index.php?curid=1441453

Problem 1: How big?

- When create a file, don't know how big it will become (in UNIX, most writes are by appending)
  - How much contiguous space do you allocate for a file?
  - In BSD 4.2, just find some range of free blocks
    - Put each new file at the front of different range
    - To expand a file, you first try successive blocks in bitmap, then choose new range of blocks
  - Also in BSD 4.2: store files from same directory near each other
- Fast File System (FFS)
  - Allocation and placement policies for BSD 4.2

Problem 2: Rotational Delay

- Missing blocks due to rotational delay
  - Issue: Read one block, do processing, and read next block. In meantime, disk has continued turning: missed next block! Need 1 revolution/block!



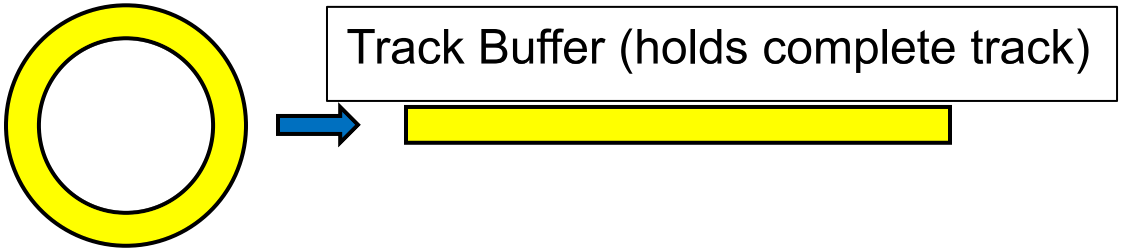
Skip Sector

Track Buffer (holds complete track)

- Solution 1: Skip sector positioning ("interleaving")
  - Place the blocks from one file on every other block of a track: give time for processing to overlap rotation
- Solution 2: Read ahead: Read next block right after first, even if application hasn't asked for it yet.
  - This can be done by the OS (read ahead) - OR -
  - By the disk itself (track buffers). Many disk controllers have internal RAM that allows them to read a complete track

Problem 2: Rotational Delay

- Important Aside: Modern disks and controllers do many complex things “under the covers”
  - Track buffers, elevator algorithms, bad block filtering



Track Buffer (holds complete track)

Conclusion

- File Systems
  - Introduction to File Systems
  - Very simply file system
  - FAT
  - Inodes
  - Unix Fast File System (FFS)