File Systems: FAT, FFS, NTFS

30 Jan 2025
Lecture 13

Slide Credits: David E Culler, UC Berkeley

Topics For Today

- Very simply file system
- FAT
- Inodes
- Unix Fast File System (FFS)
- NTFS

A simple File System: Ingredients

Blocks

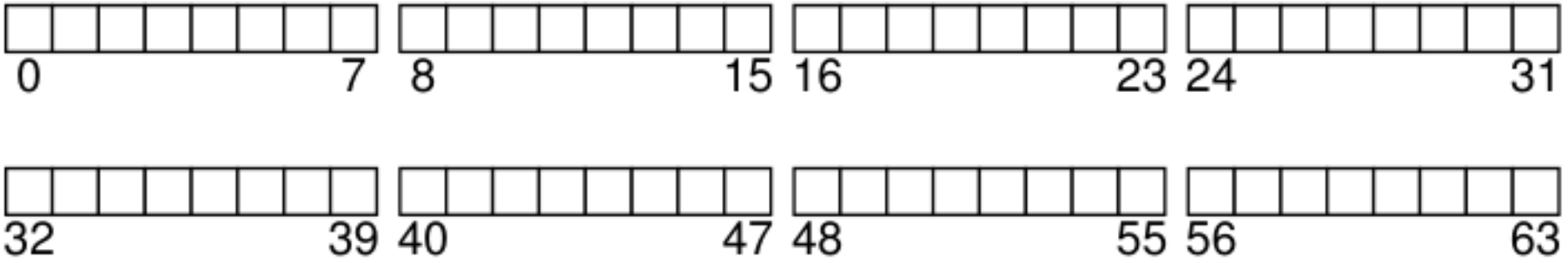
- 4 KB
- 64 total blocks
  - Numbered 0-63

Data space

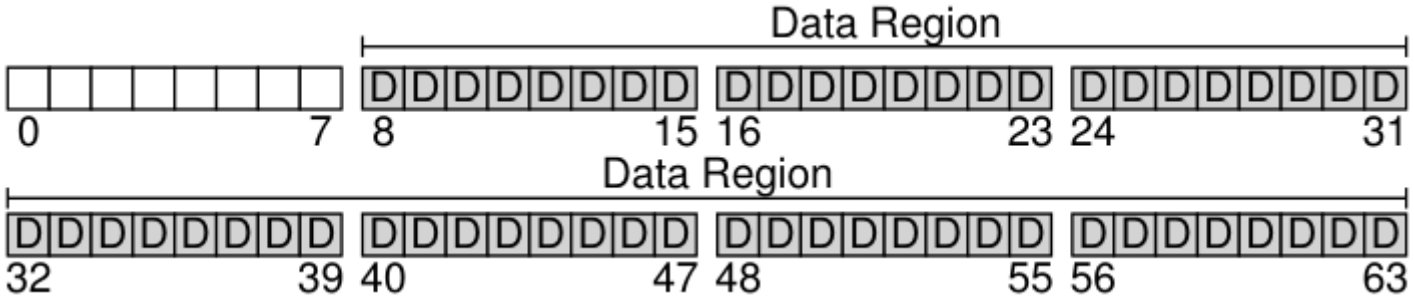
Metadata space

Superblock

Our blocks (4KB)



Data Region

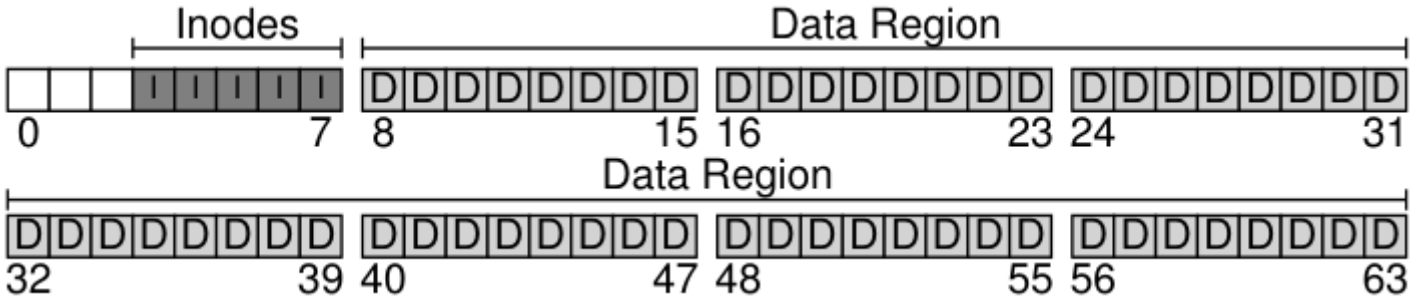


Blocks 8-63 will be data

56 total

Rest will be metadata

Inode region – What's an inode?



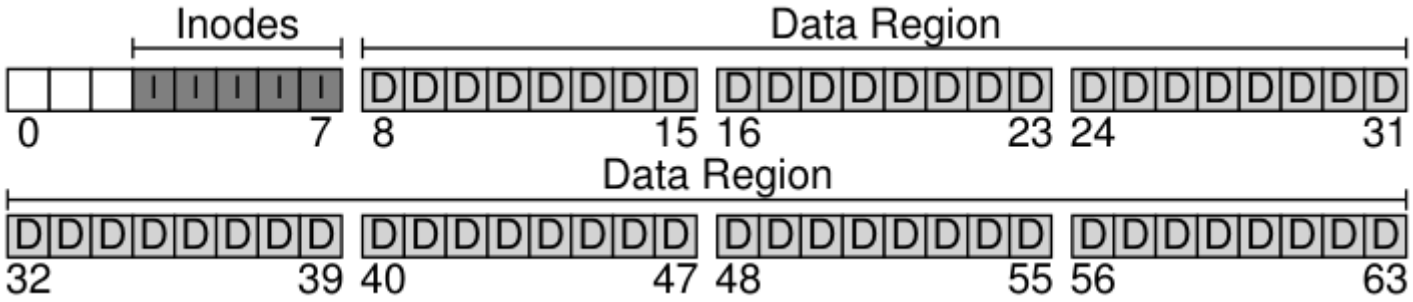
Data structure holds metadata about file

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

Typically stored in a table

- Small, say 256 bytes

Inode region



Use 5 blocks for inodes

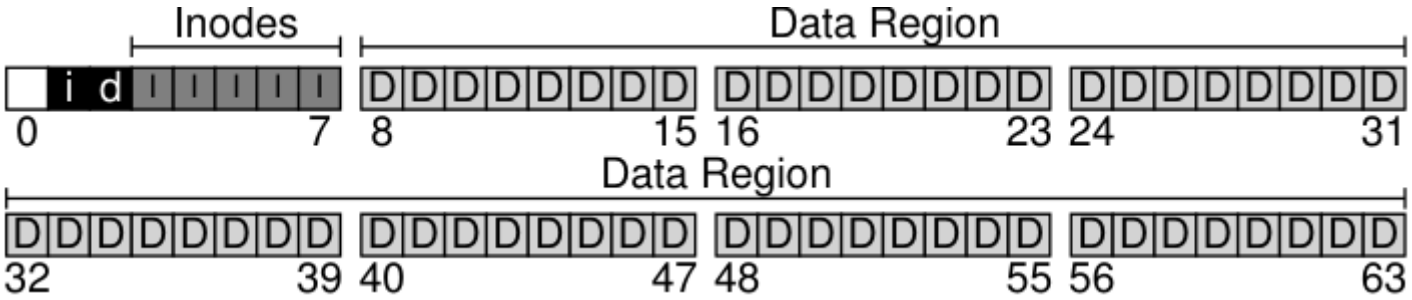
Each 4KB block holds 16 inodes

- 4096 / 256 = 16

Total 80 inodes

- Can't have more than 80 files or directories

Used/Free tracking



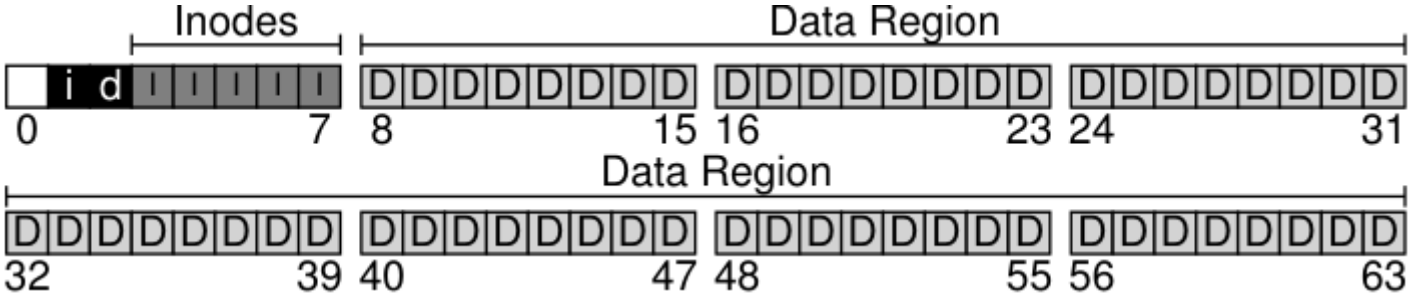
Need to track if an inode is in used

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



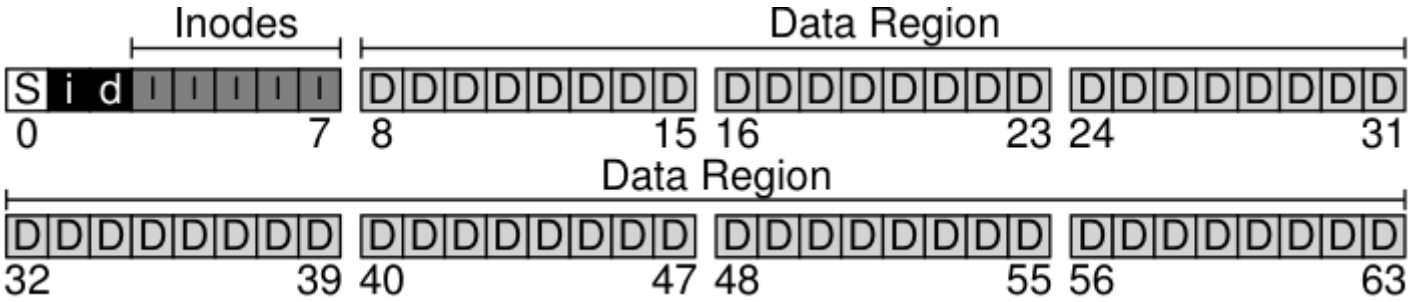
Block 1 tracks inodes

- 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



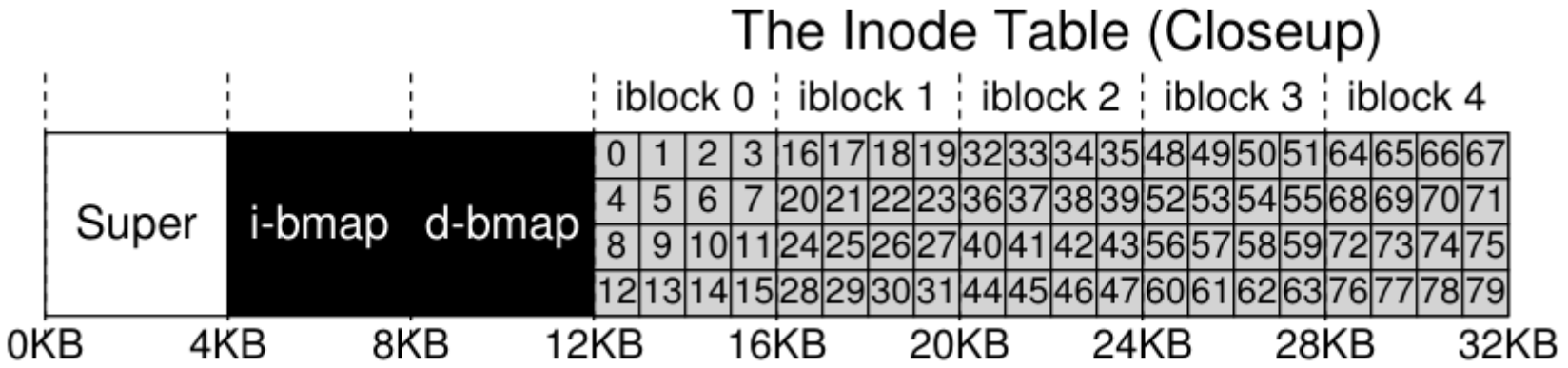
Block 0 contains basic metadata

- 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

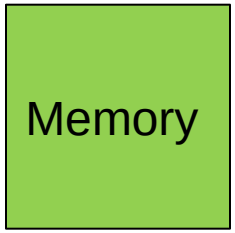
- Contains records of files and directories inside
- i-number of the file/directory
- Reclen – how many bytes in the record
- Strlen – how many bytes in the name of the file/directory
- Name – the actual name (≤ reclen)

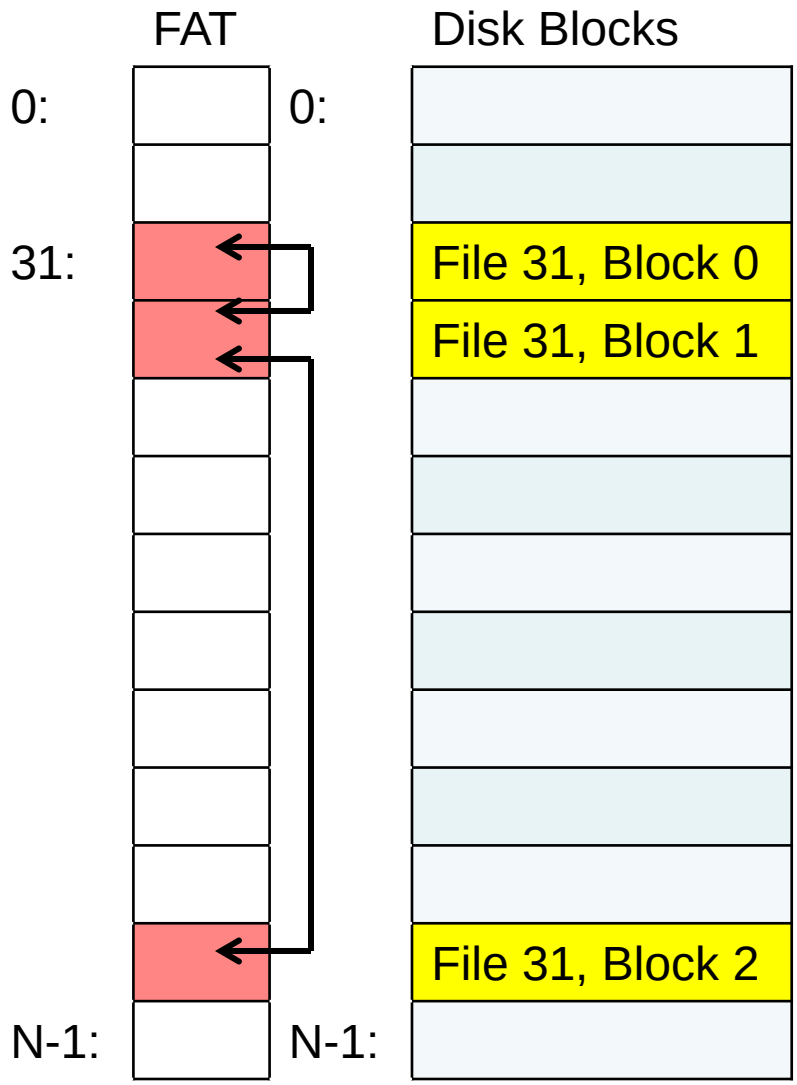
So Far

- Very simply file system
- FAT
- Inodes
- Unix Fast File System (FFS)
- NTFS

Our first filesystem: FAT (File Allocation Table)

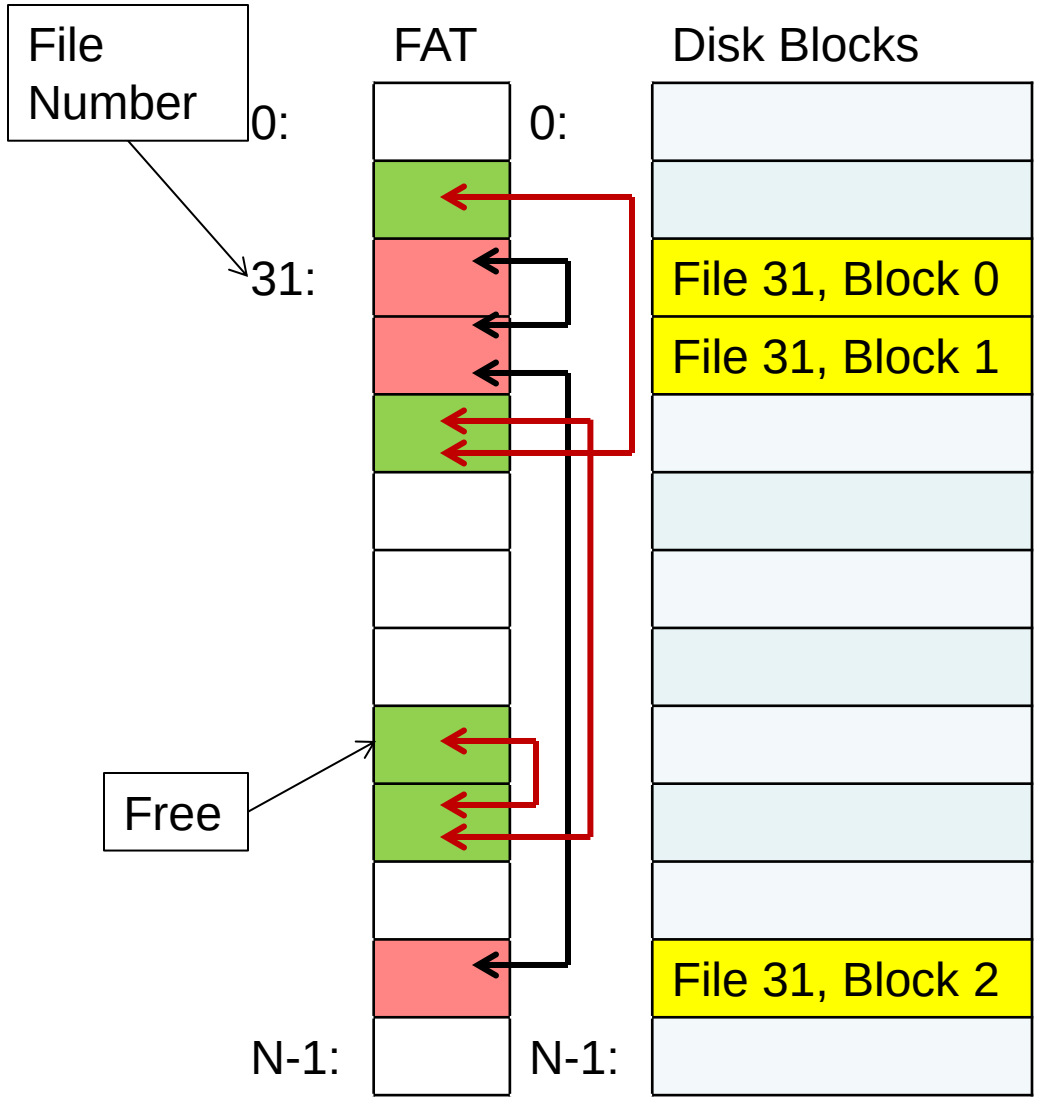
- Assume we have a way to translate a path to a "file number"
  - i.e., a directory structure
- Disk Storage is a collection of Blocks
  - Just hold file data
- Ex: `file_read` $31, \langle 2, x \rangle$
  - Index into FAT with file number
  - Follow linked list to block
  - Read the block from disk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





FAT Properties

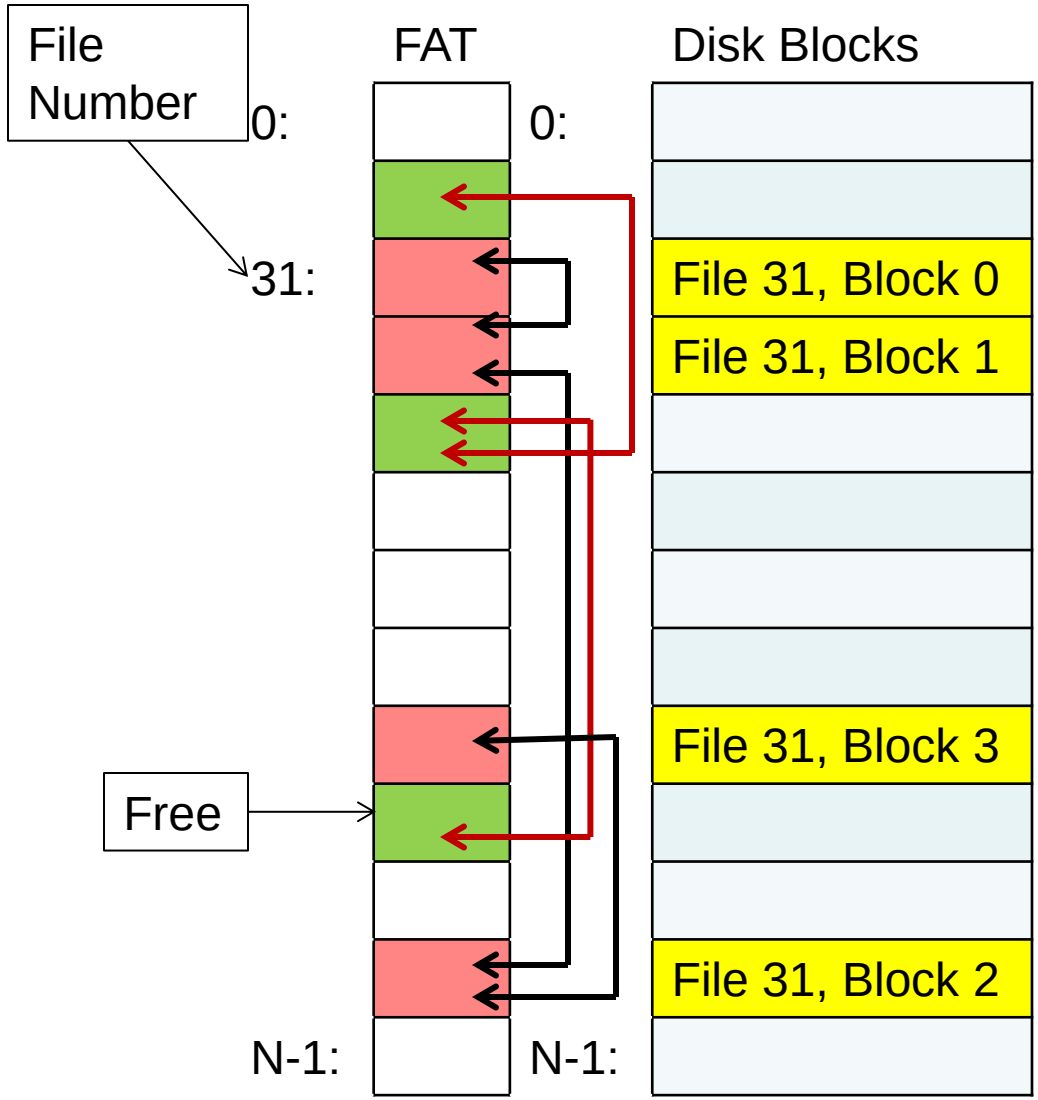
- File is collection of disk blocks
- FAT is linked list 1-1 with blocks
- File Number is index of root
  of block list for the file
- File offset ($o = B:x$)
- Follow list to get block #
- Unused blocks ⇔ FAT free list



Writing a File Block

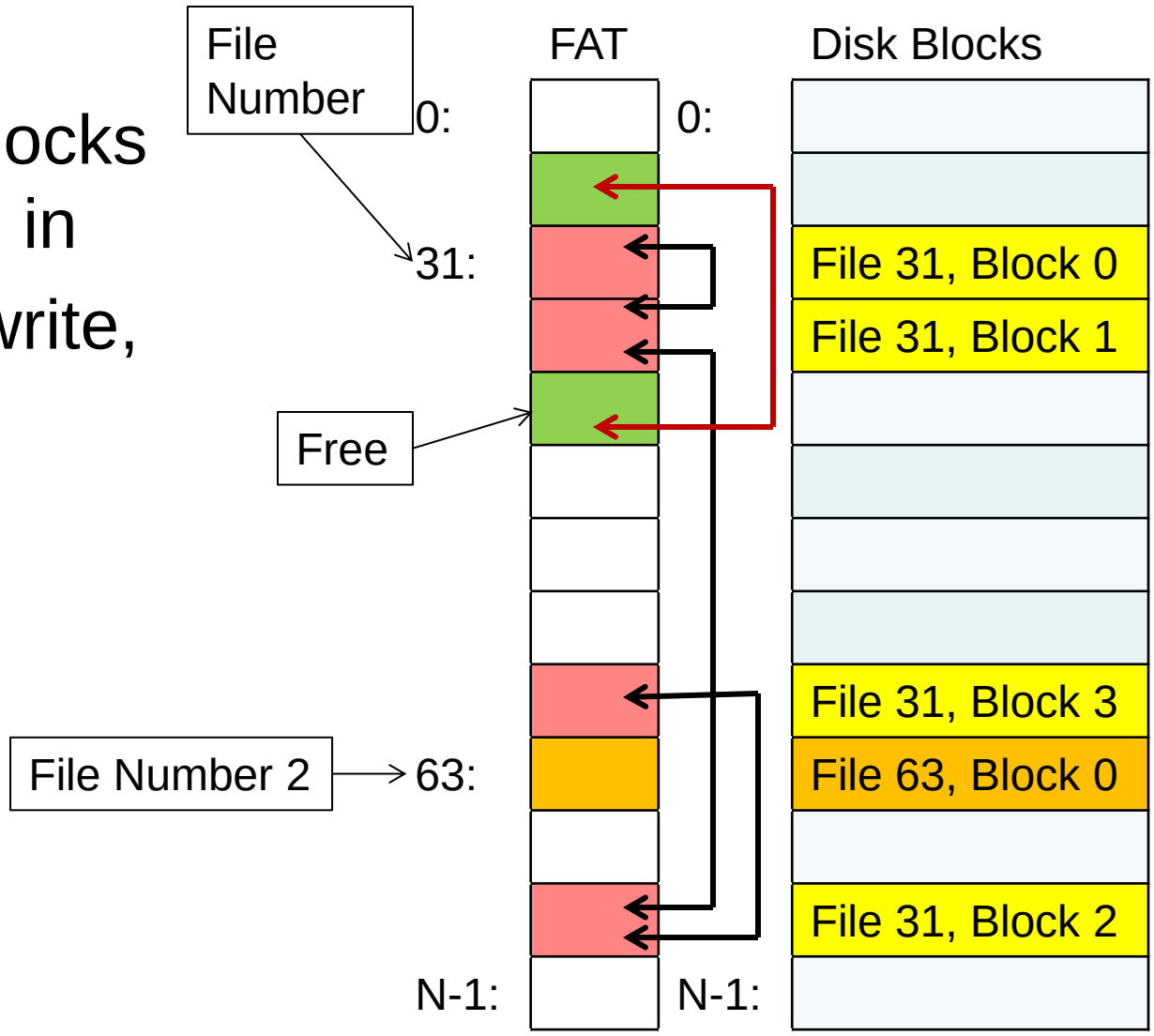
Ex: `file_write`$(51, \langle 3, y \rangle)$

- Grab blocks from free list
- Linking them into file



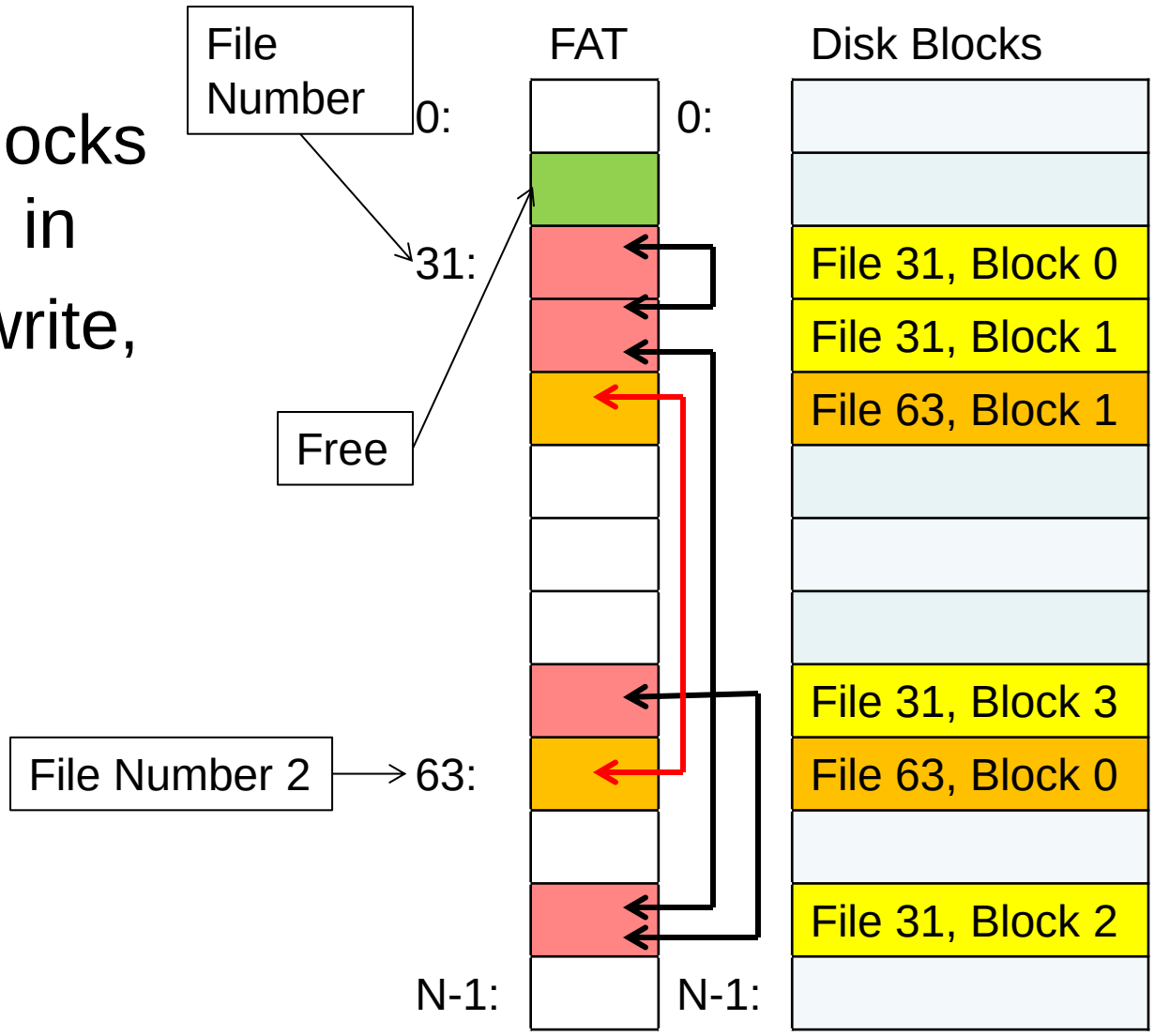
Create a New File

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



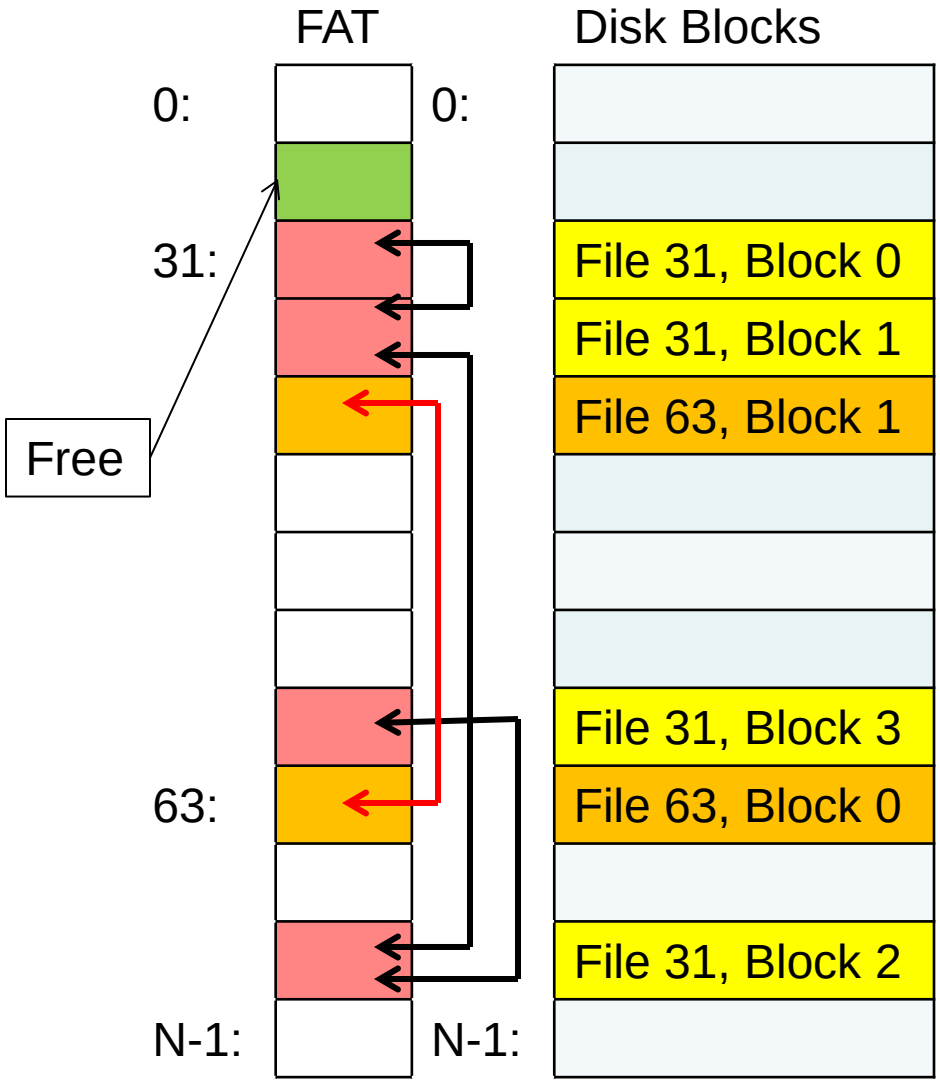
Create a New File

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



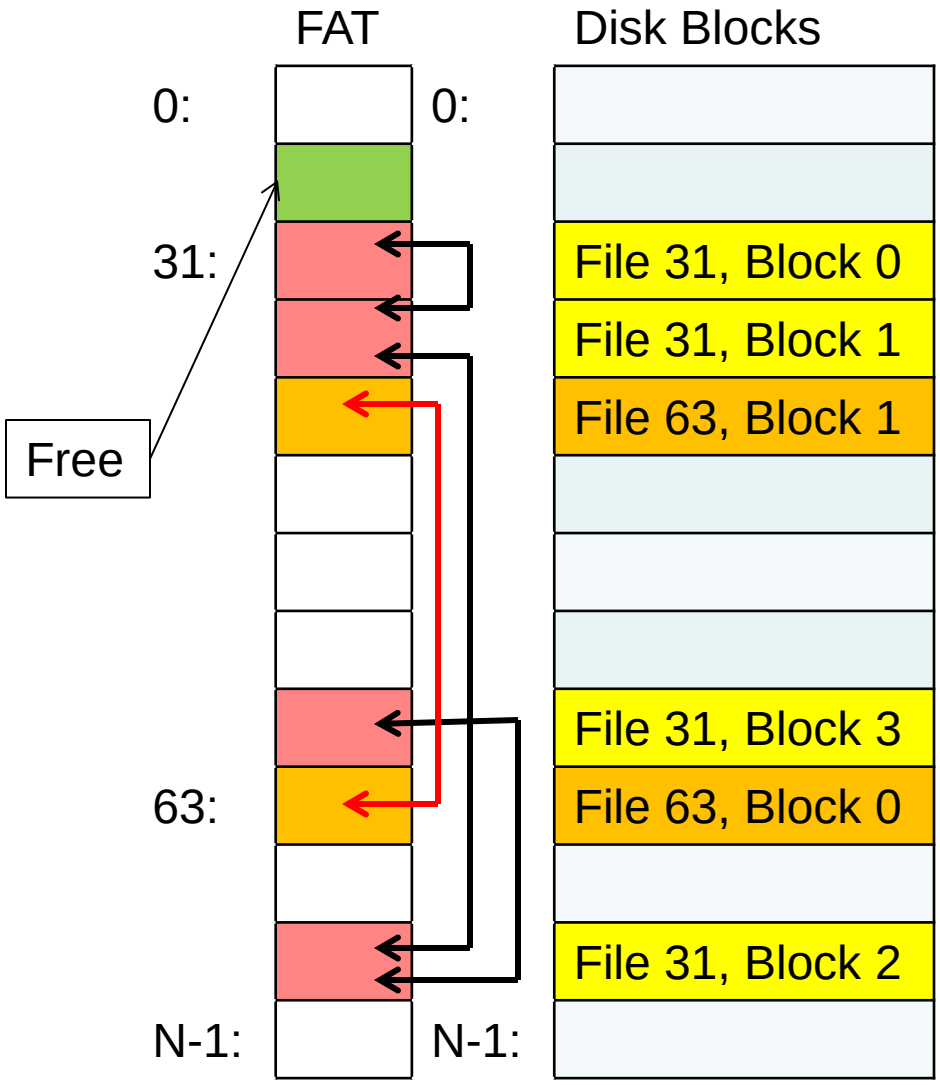
FAT Assessment

- 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

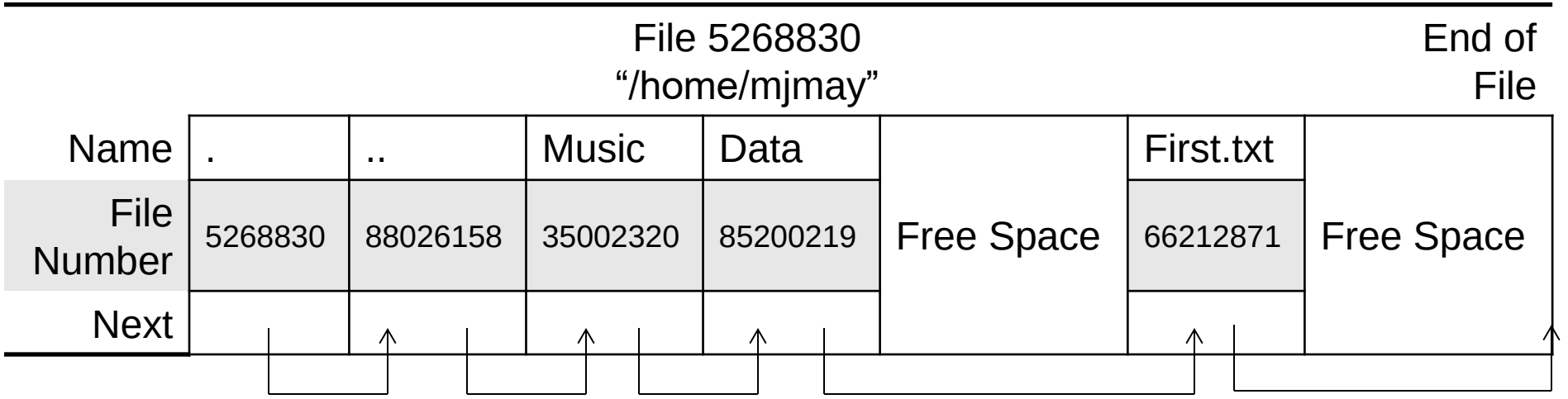


FAT Assessment

- 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?



- 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

- 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

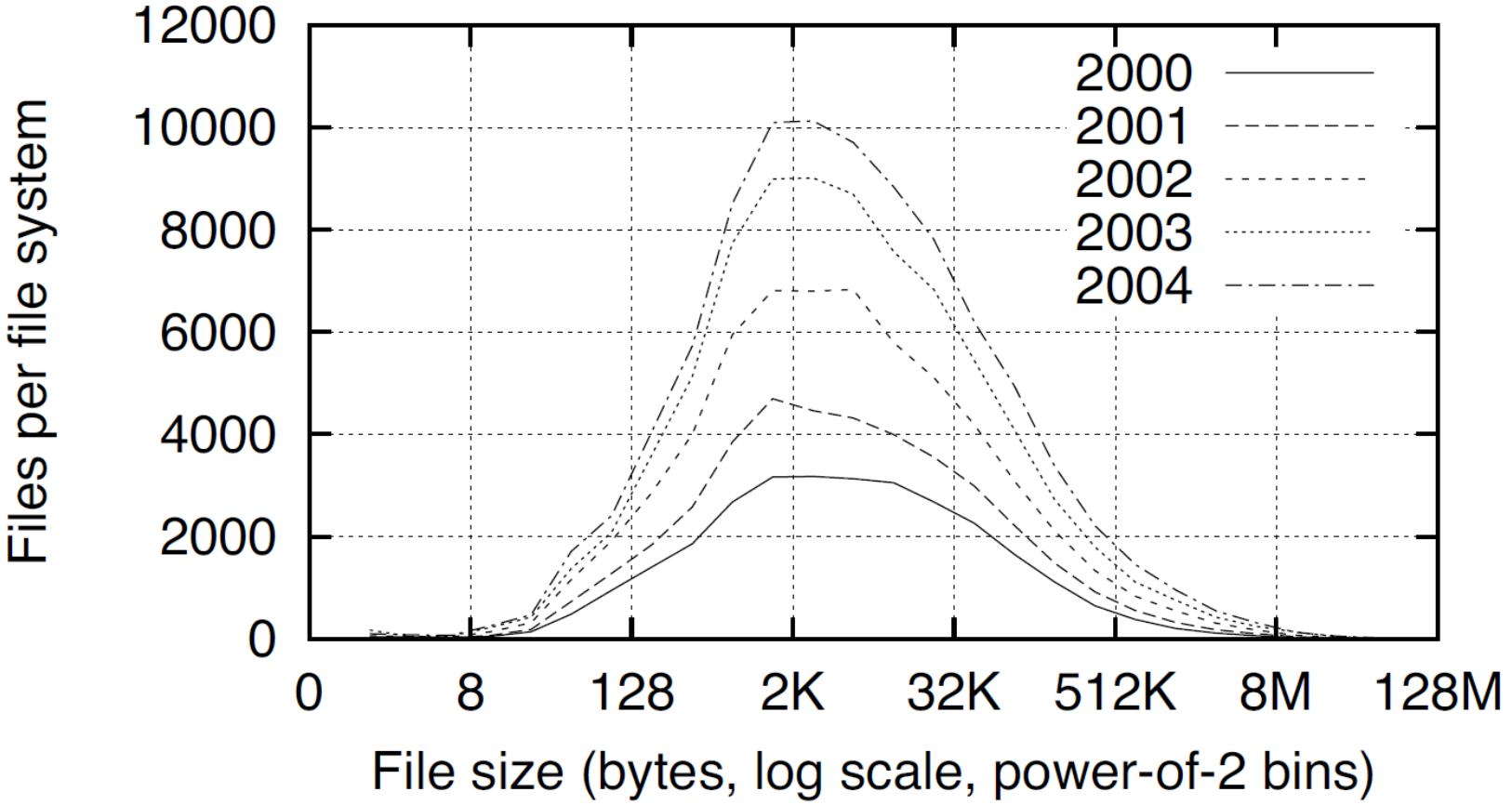


Figure 2: Histograms of files by size

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

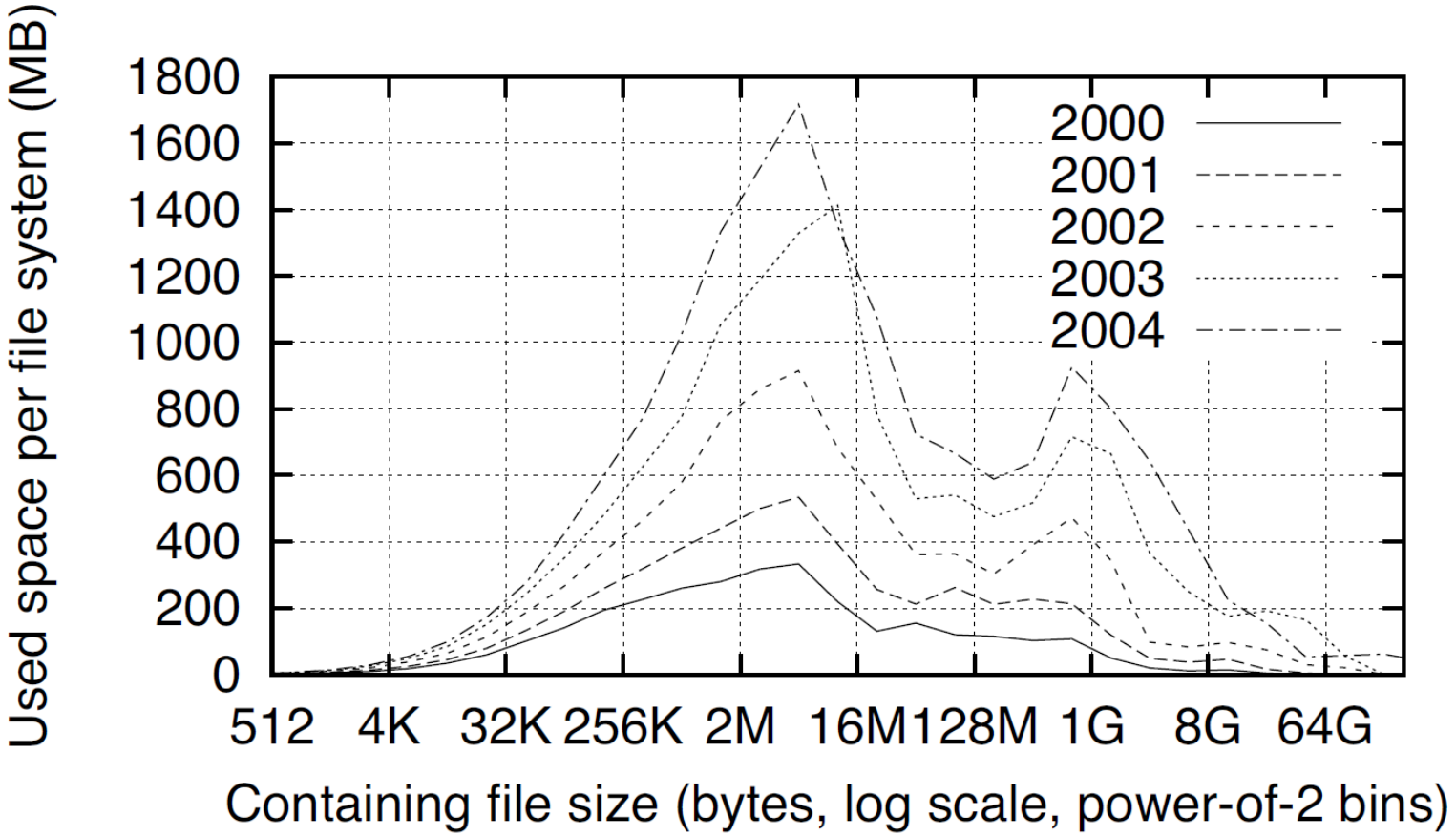
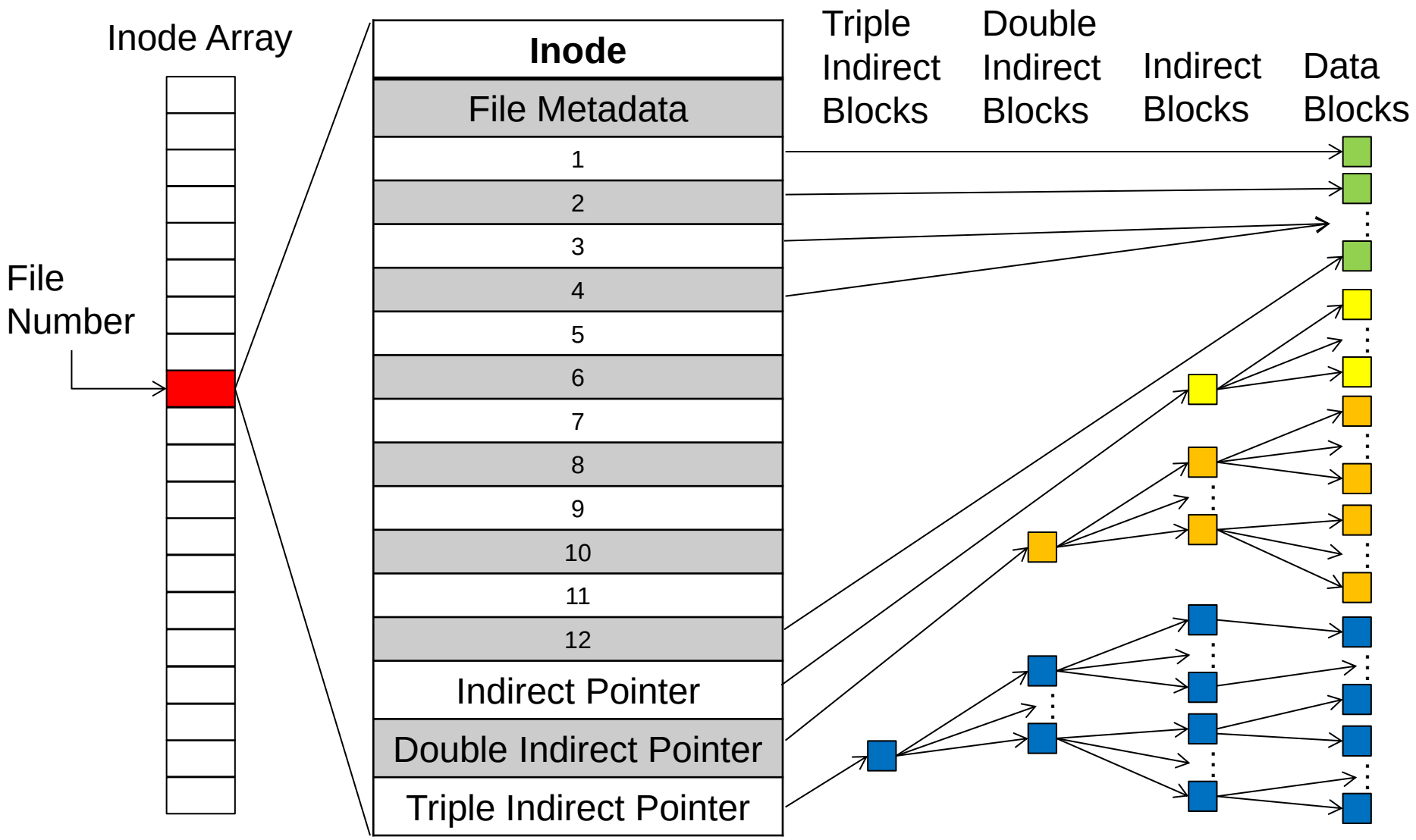


Figure 4: Histograms of bytes by containing file size

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

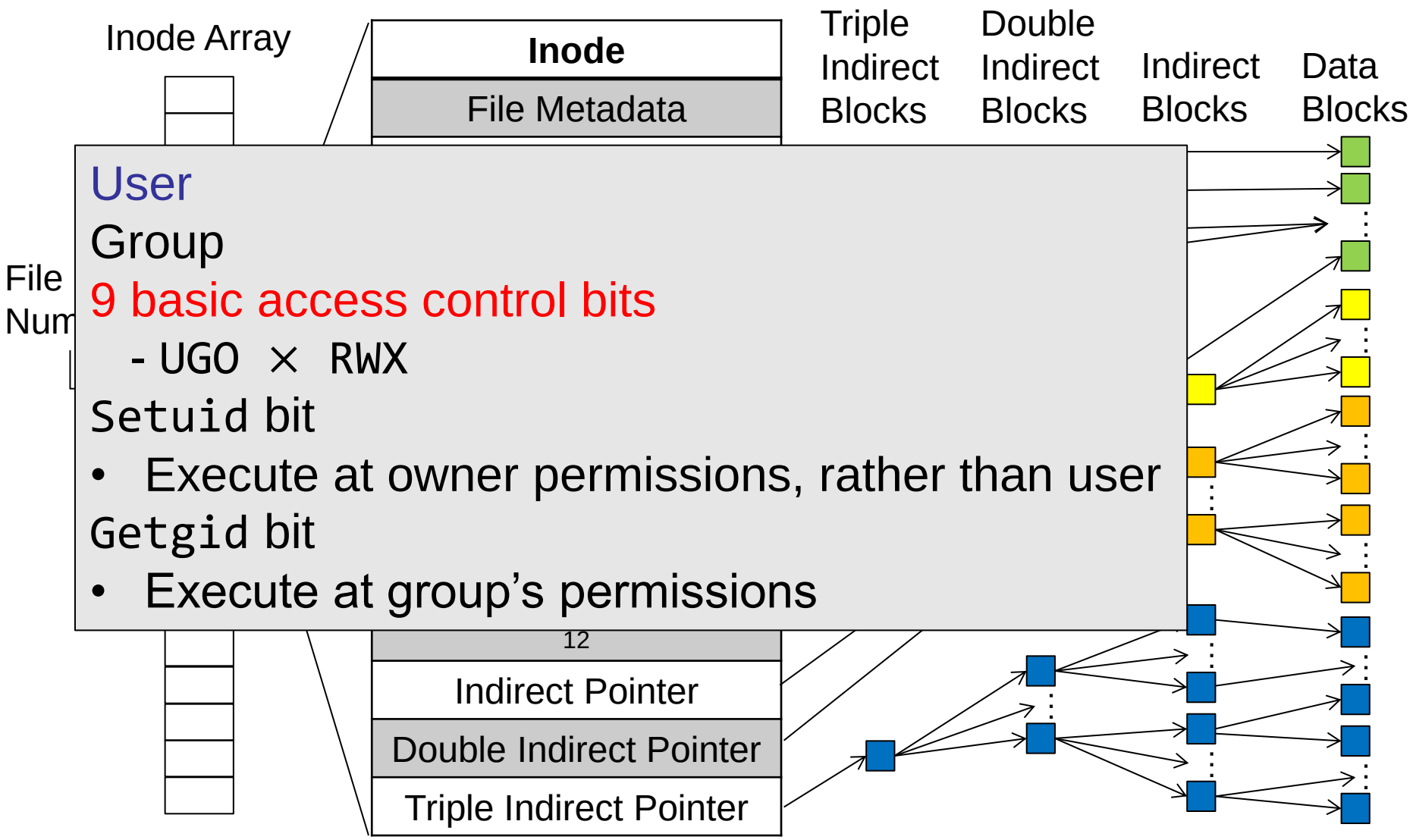


Unix Fast File System

- 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

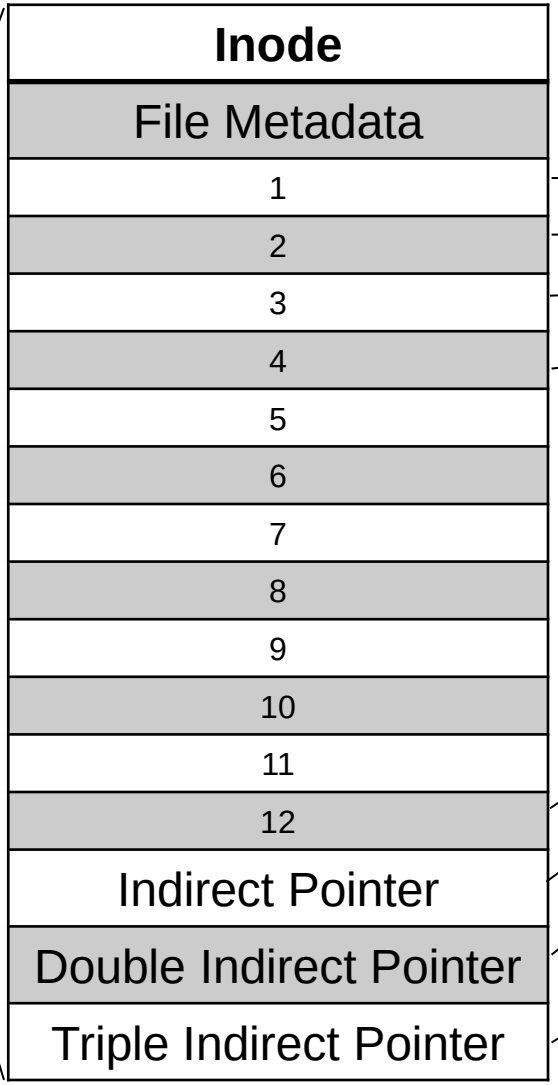
- Execute at group's permissions

Data Storage

Inode Array

Small files:
12 pointers direct to data blocks

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



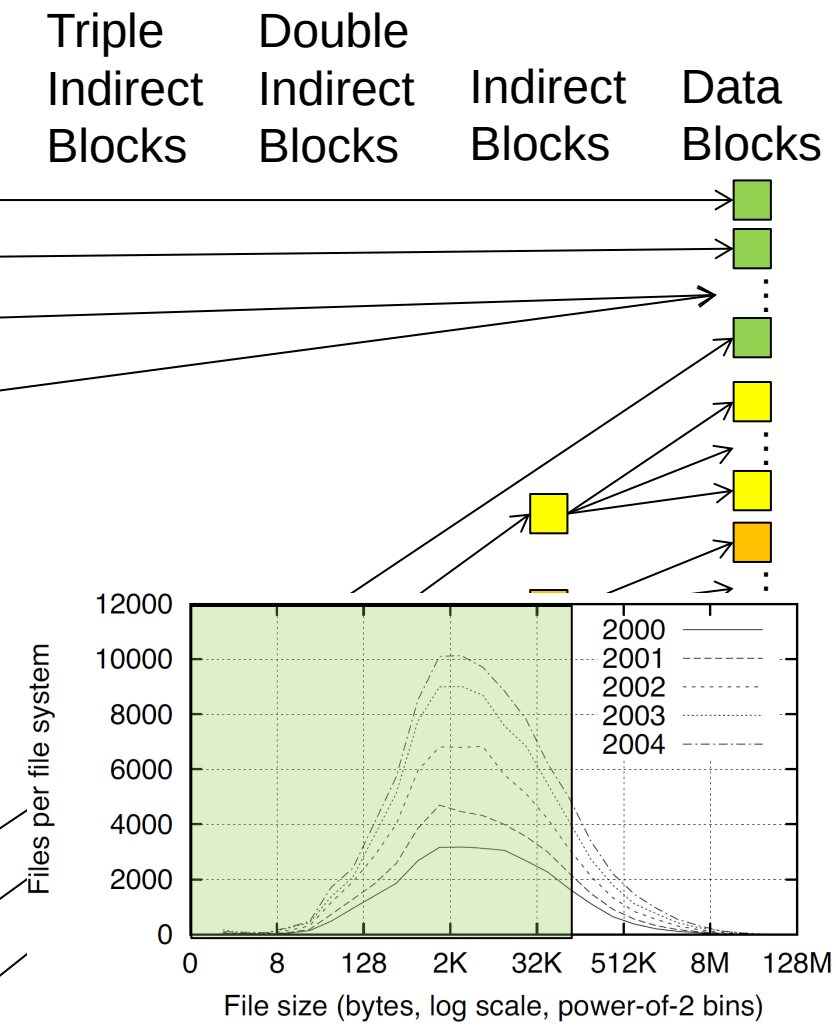


Figure 2: Histograms of files by size

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

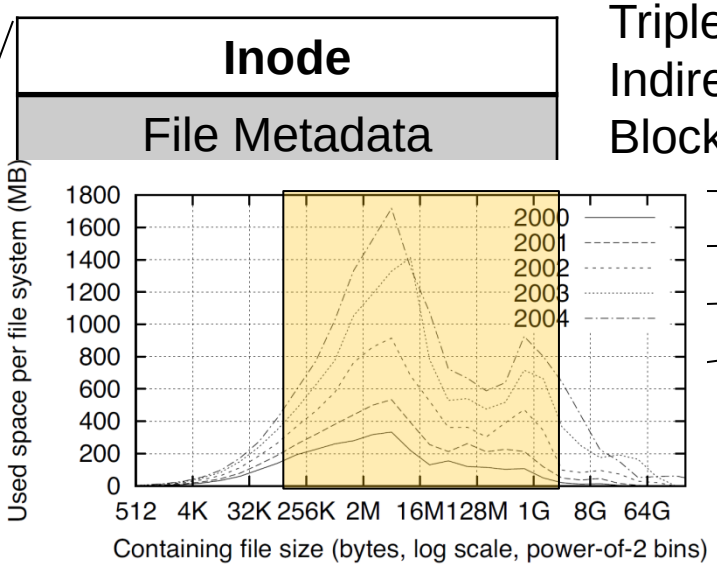
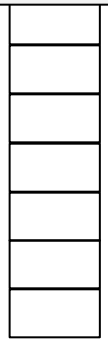
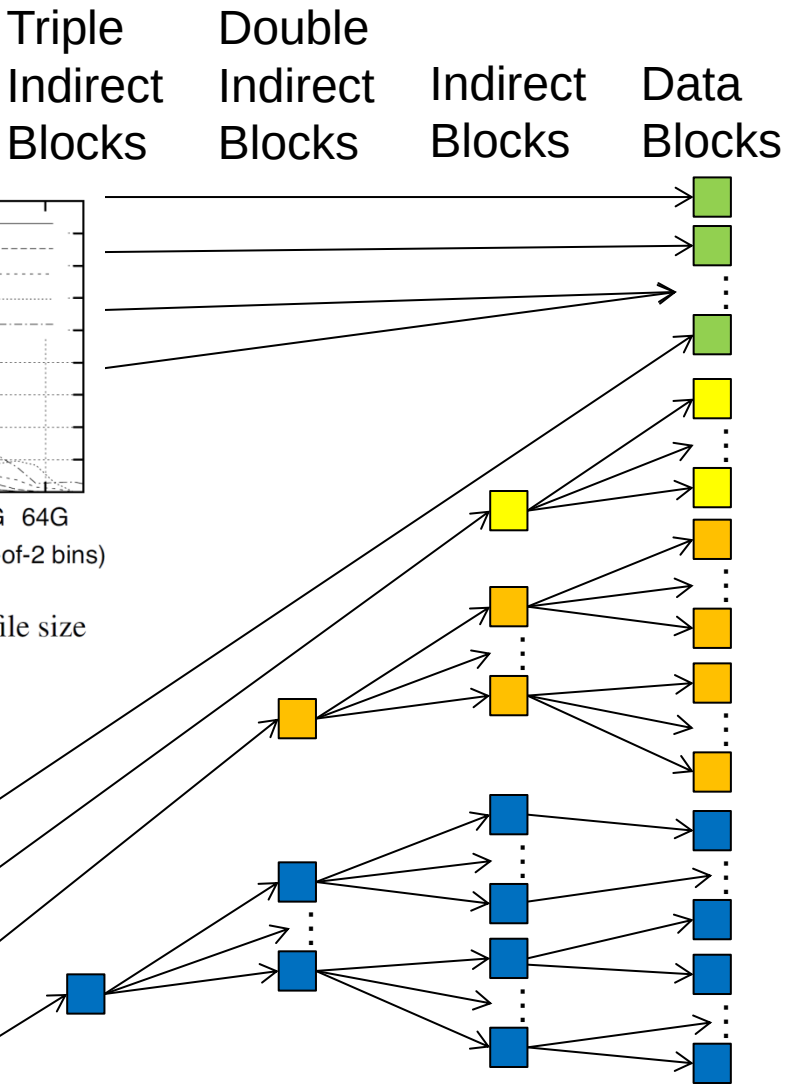


Figure 4: Histograms of bytes by containing file size





So Far

- Very simply file system
- FAT
- Inodes
- Unix Fast File System (FFS)
- NTFS

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)

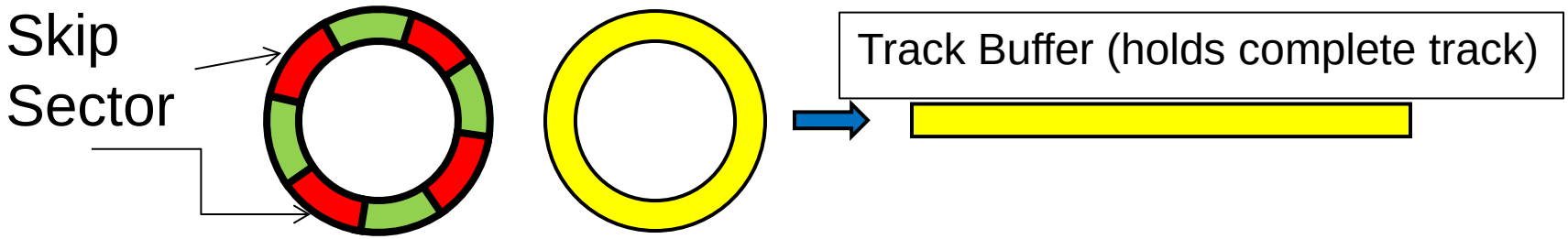
Problem 1: How big?

- 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

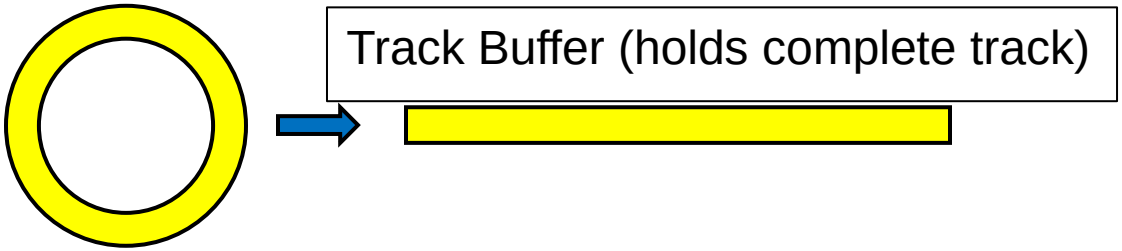
- Missing blocks due to rotational delay
  - Issue: Read one block, do processing, and read next block. In meantime, disk has continued turning: missed next block! Need 1 revolution/block!



- 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



Where are inodes stored?

- In early UNIX and DOS/Windows' FAT file system, headers stored in special array in outermost cylinders
  - Header not stored anywhere near the data blocks. To read a small file, seek to get header, seek back to data.
  - Fixed size, set when disk is formatted. At formatting time, a fixed number of inodes were created (They were each given a unique number, called an "inumber")

Where are inodes stored?

- Later versions of UNIX moved the header information to be closer to the data blocks
  - Often, inode for file stored in same cylinder group as parent directory of the file (makes an `ls` of that directory run fast).

Pros:

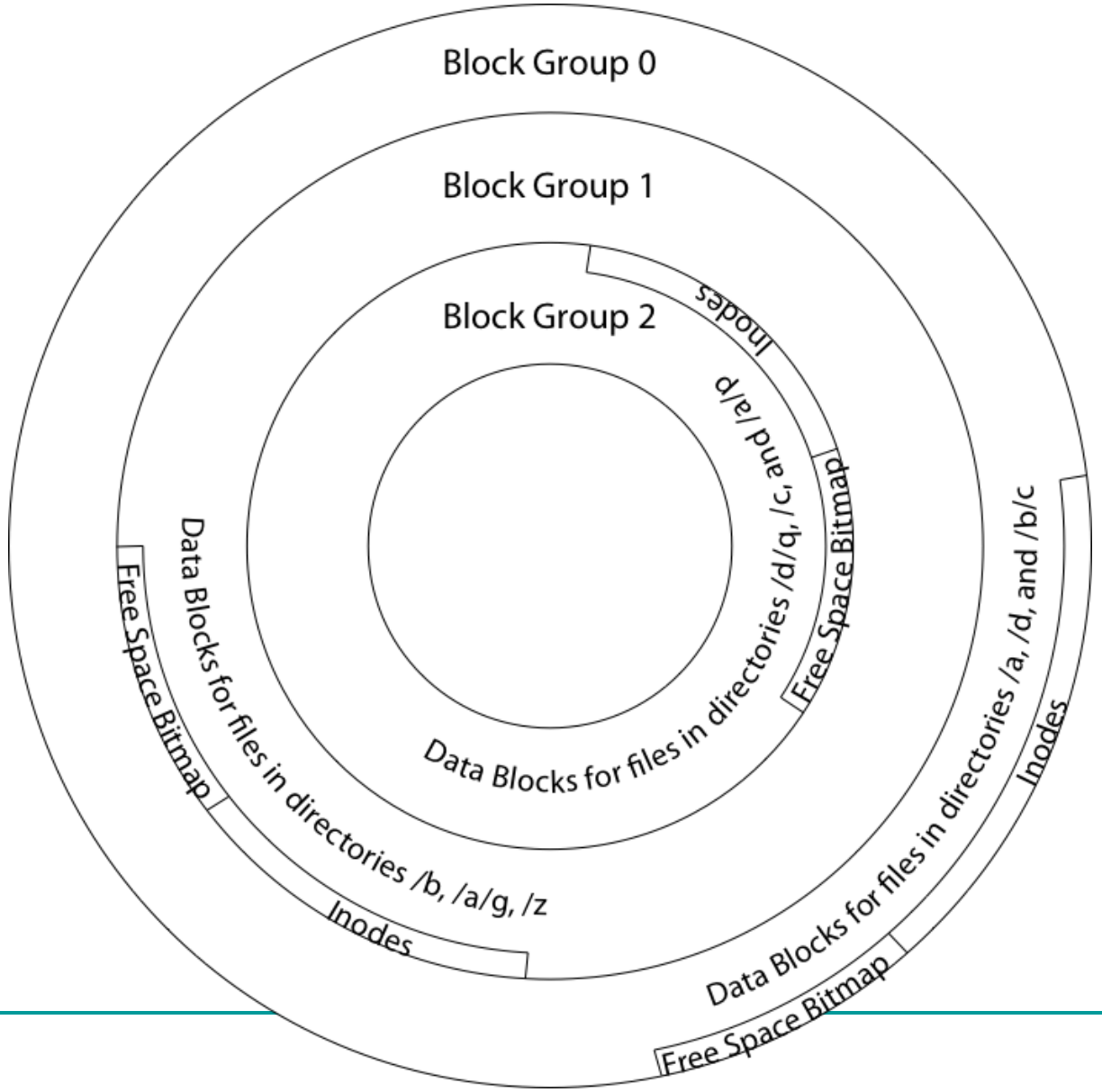
- UNIX BSD 4.2 puts a portion of the file header array on each of many cylinders. For small directories, can fit all data, file headers, etc. in same cylinder ⇒ no seeks!
- File headers much smaller than whole block (a few hundred bytes), so multiple headers fetched from disk at same time
- Reliability: whatever happens to the disk, you can find many of the files (even if directories disconnected)

  - Part of the Fast File System (FFS)
    - General optimization to avoid seeks

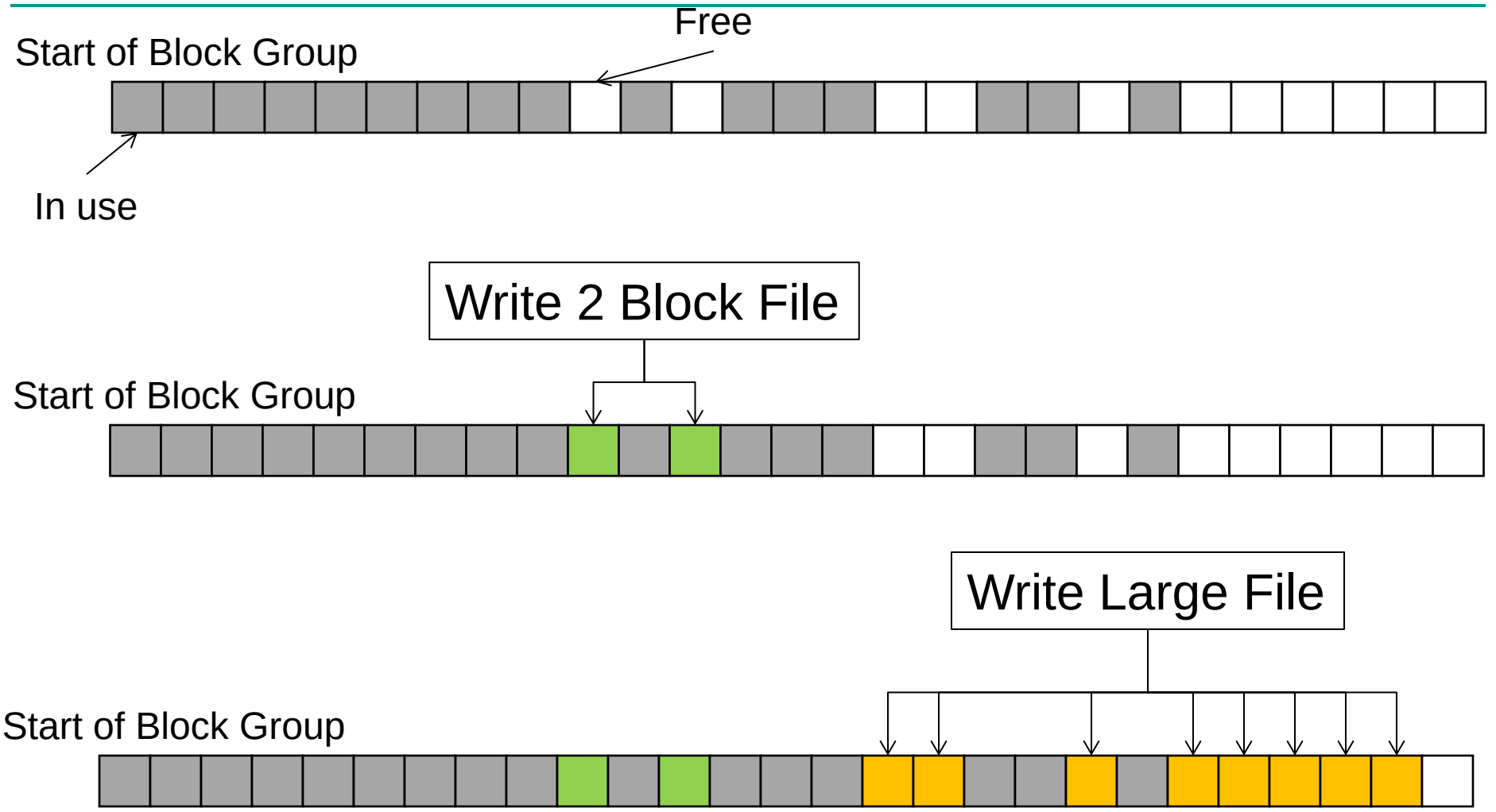
4.2 BSD Locality: Block Groups

- File system volume is divided into a set of block groups
  - Close set of tracks
- Data blocks, metadata, and free space interleaved within block group
  - Avoid huge seeks between user data and system structure
- Put directory and its files in common block group
- First-Free allocation of new file blocks
  - To expand file, first try successive blocks in bitmap, then choose new range of blocks
  - Few little holes at start, big sequential runs at end of group
  - Avoids fragmentation
  - Sequential layout for big files
- Important: keep 10% or more free!
  - Reserve space in the Background

4.2 BSD Locality: Block Groups



FFS First Fit Block Allocation



FFS

Pros

- Efficient storage for both small and large files
- Locality for both small and large files
- Locality for metadata and data

Cons

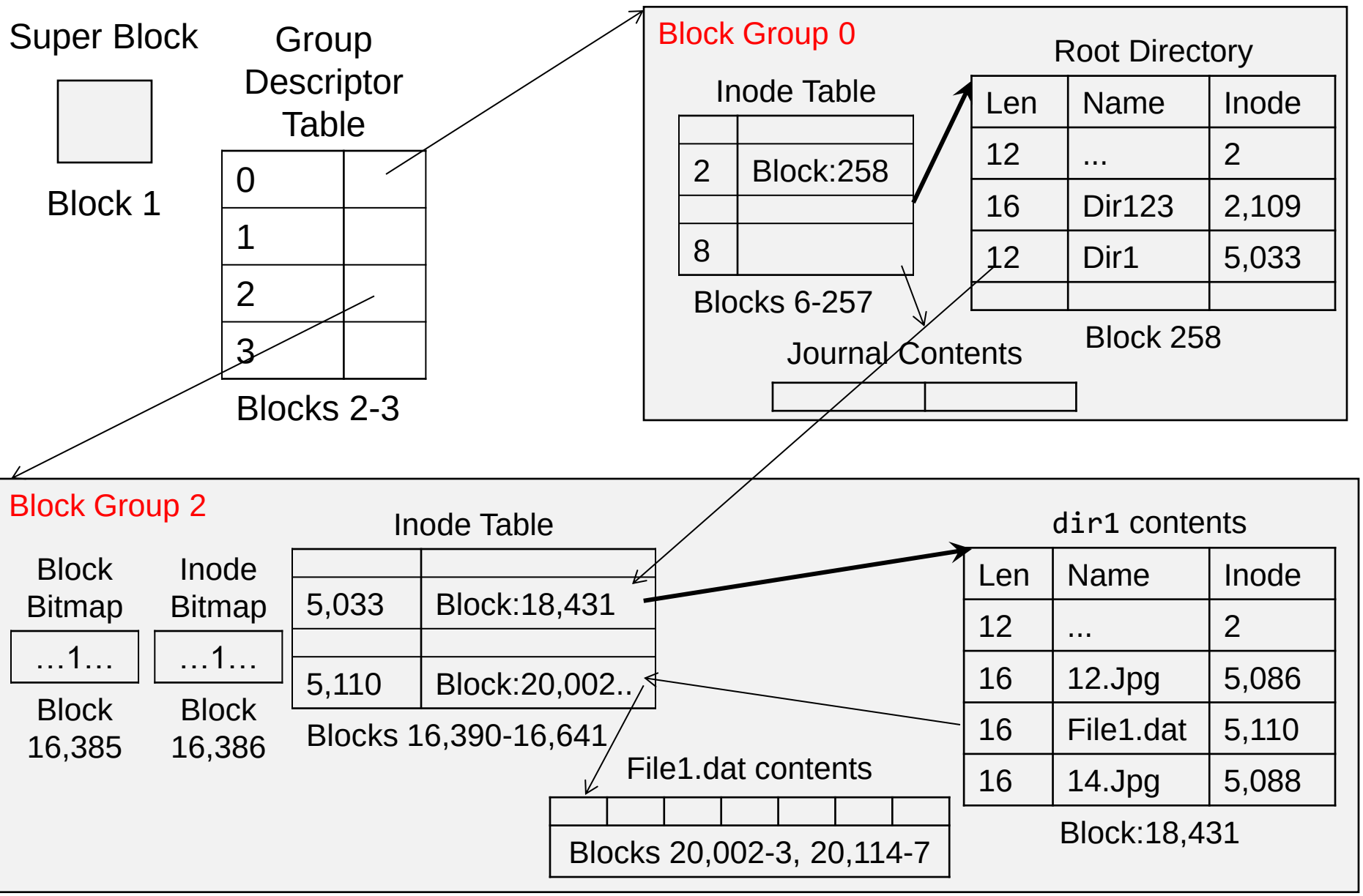
- Inefficient for tiny files (a 1 byte file requires both an inode and a data block)
- Inefficient encoding when file is mostly contiguous on disk (no equivalent to superpages)
- Need to reserve 10-20% of free space to prevent fragmentation

Linux Example: ext2/3 Disk Layout

- Disk divided into block groups
  - Provides locality
  - Each group has two block-sized bitmaps (free blocks/inodes)
  - Block sizes settable at format time: 1K, 2K, 4K, 8K…
- Actual Inode structure similar to 4.2BSD with 12 direct pointers

- ext3: ext2 with Journaling
  - Several degrees of protection with more or less cost

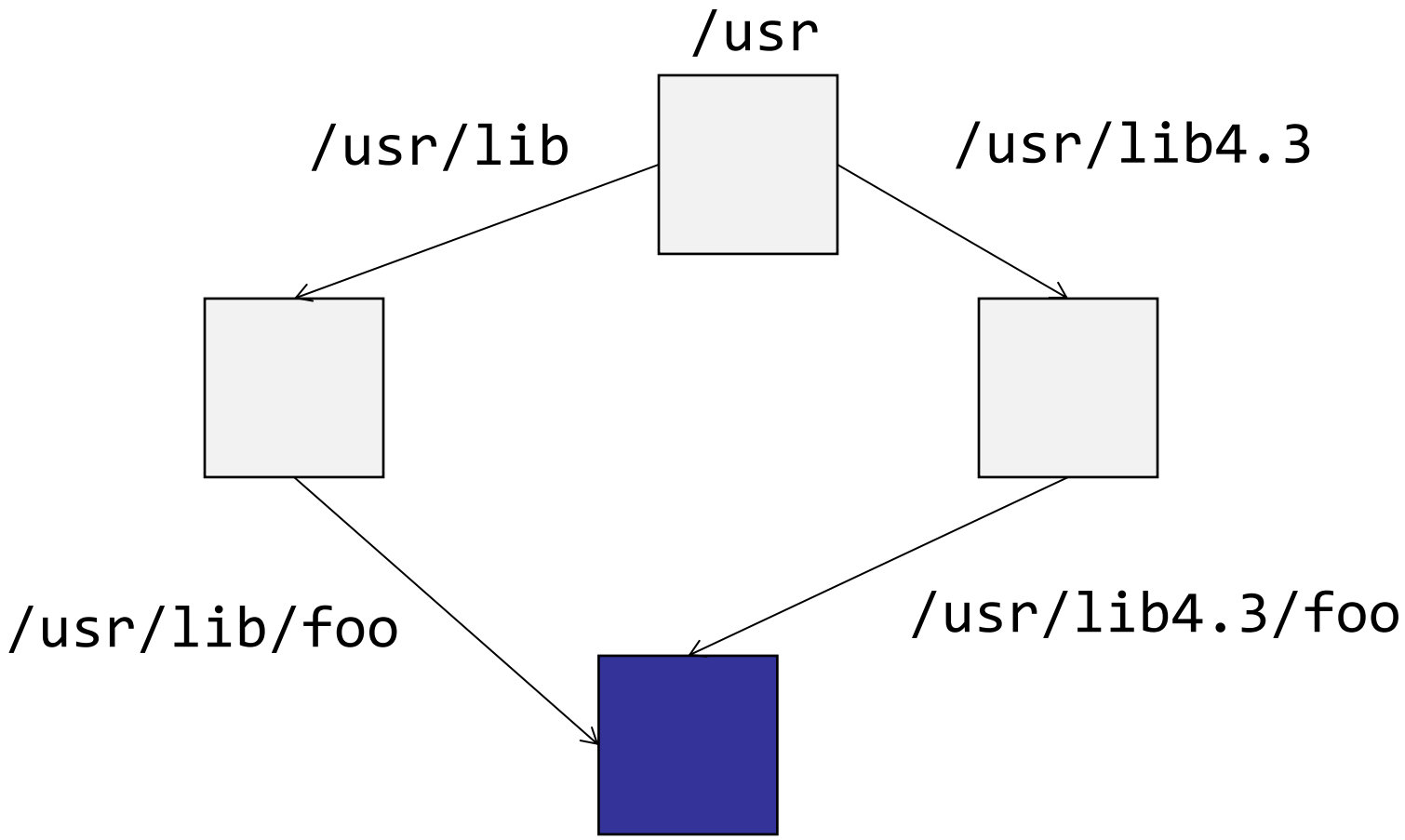
Ex: Create a `file1.dat` under `/dir1/` in ext3



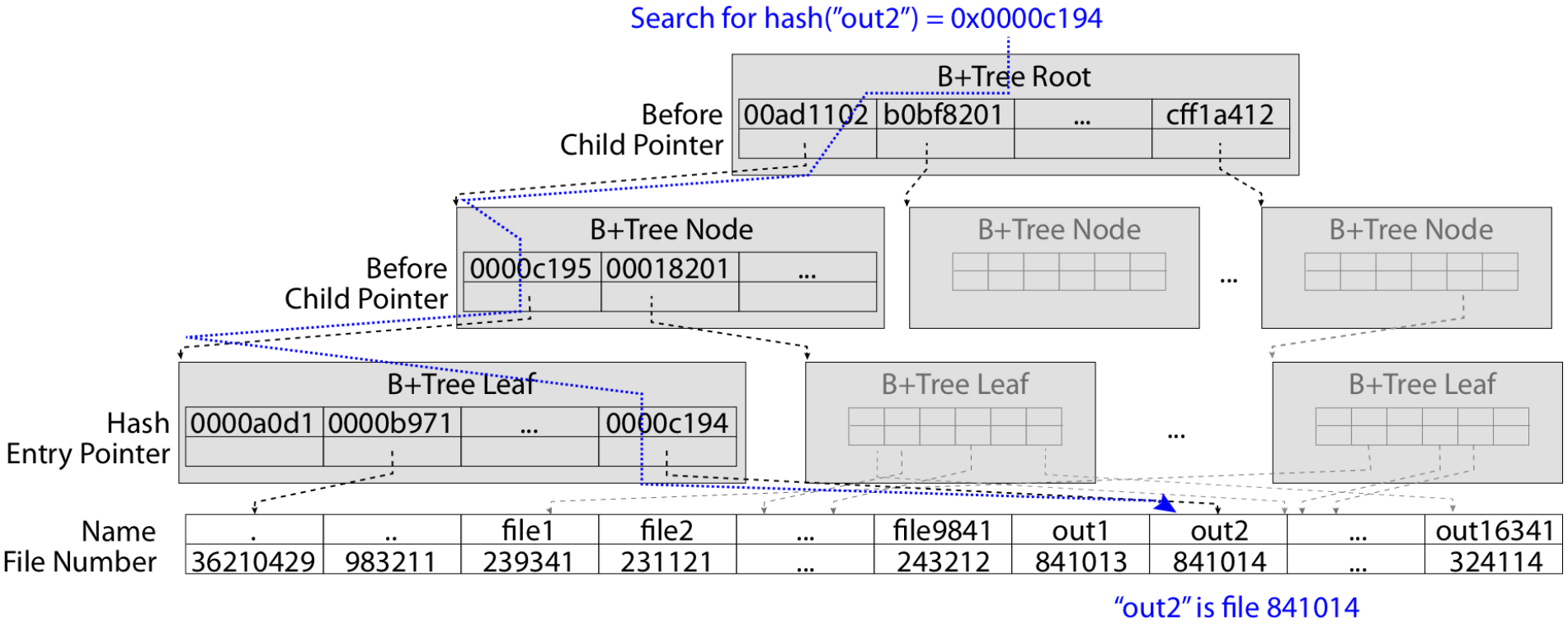
A bit more on directories

- Stored in files, can be read, but typically don't
  - System calls to access directories
  - `Open / Creat` traverse the structure
  - `mkdir /rmdir` add/remove entries
  - `Link / Unlink`
    - Link existing file to a directory (Not in FAT!)
    - Forms a DAG
- When can file be deleted?
  - Maintain reference count of links to the file
  - Delete after the last reference is gone.
- `libc` support
  - `DIR * opendir (const char *dirname)`
  - `struct dirent * readdir (DIR *dirstream)`
  - `int readdir_r (DIR *dirstream, struct dirent *entry, struct dirent **result)`

A bit more on directories



Large Directories: B-Trees (`dirhash`)



So Far

- Very simply file system
- FAT
- Inodes
- Unix Fast File System (FFS)
- NTFS

New Technology File System (NTFS)

Default on modern Windows systems

Instead of FAT or inode array: Master File Table

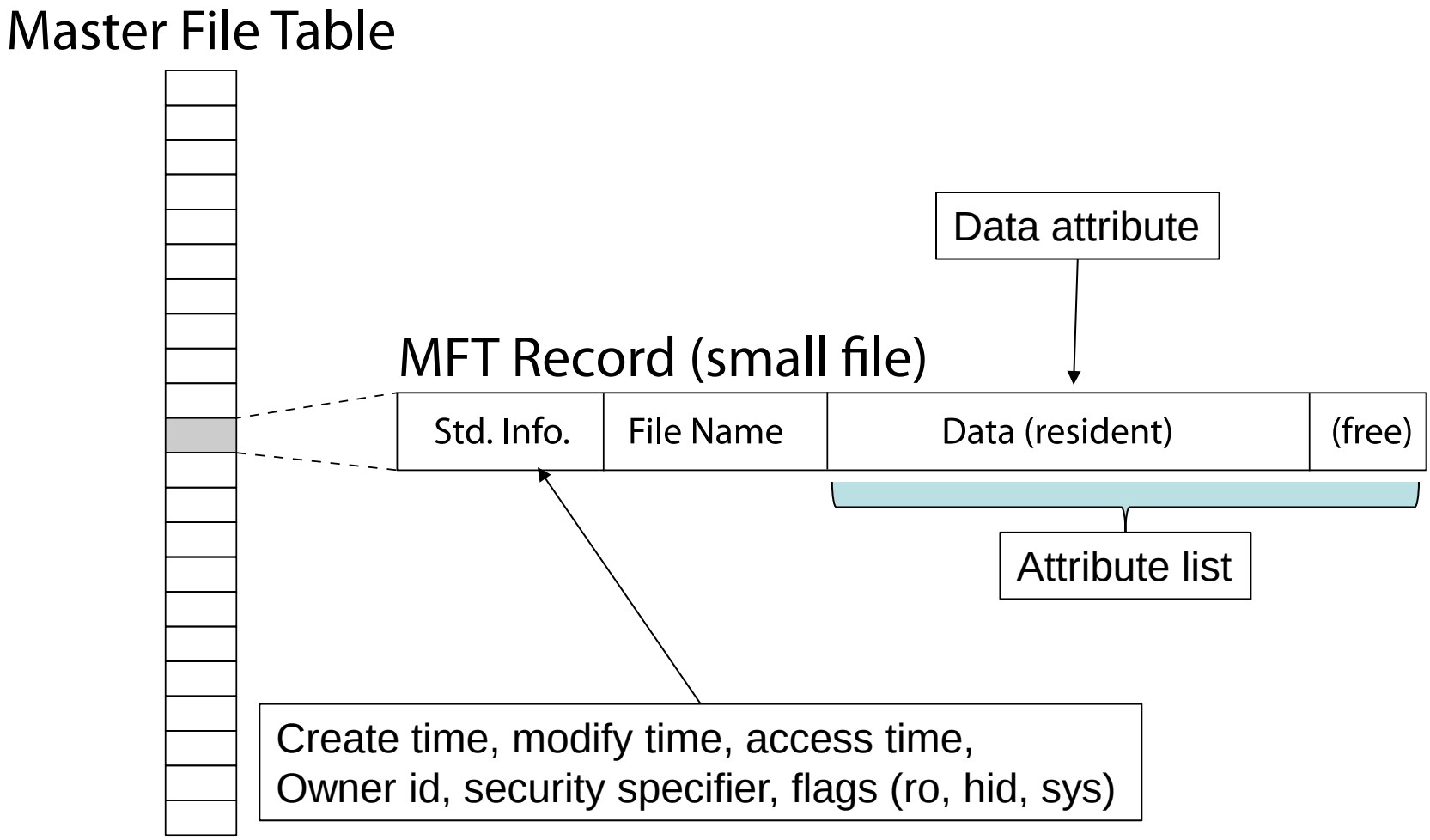
- Max 1 KB size for each table entry
- Variable-sized attribute records (data or metadata)

Each entry in MFT contains metadata plus

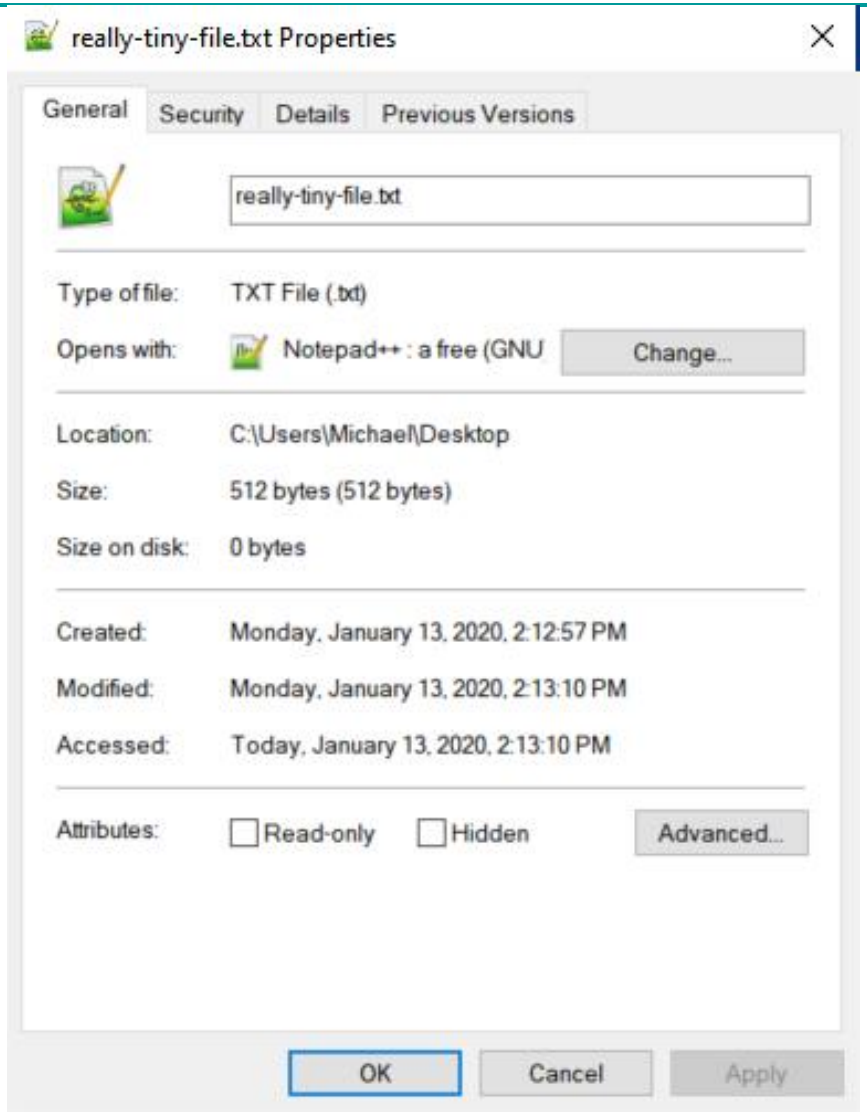
- File's data directly (for small files)
- A list of *extents* (start block, size) for file's data
- For big files: pointers to other MFT entries with *more* extent lists

Add transactions for file system changes (journaling)

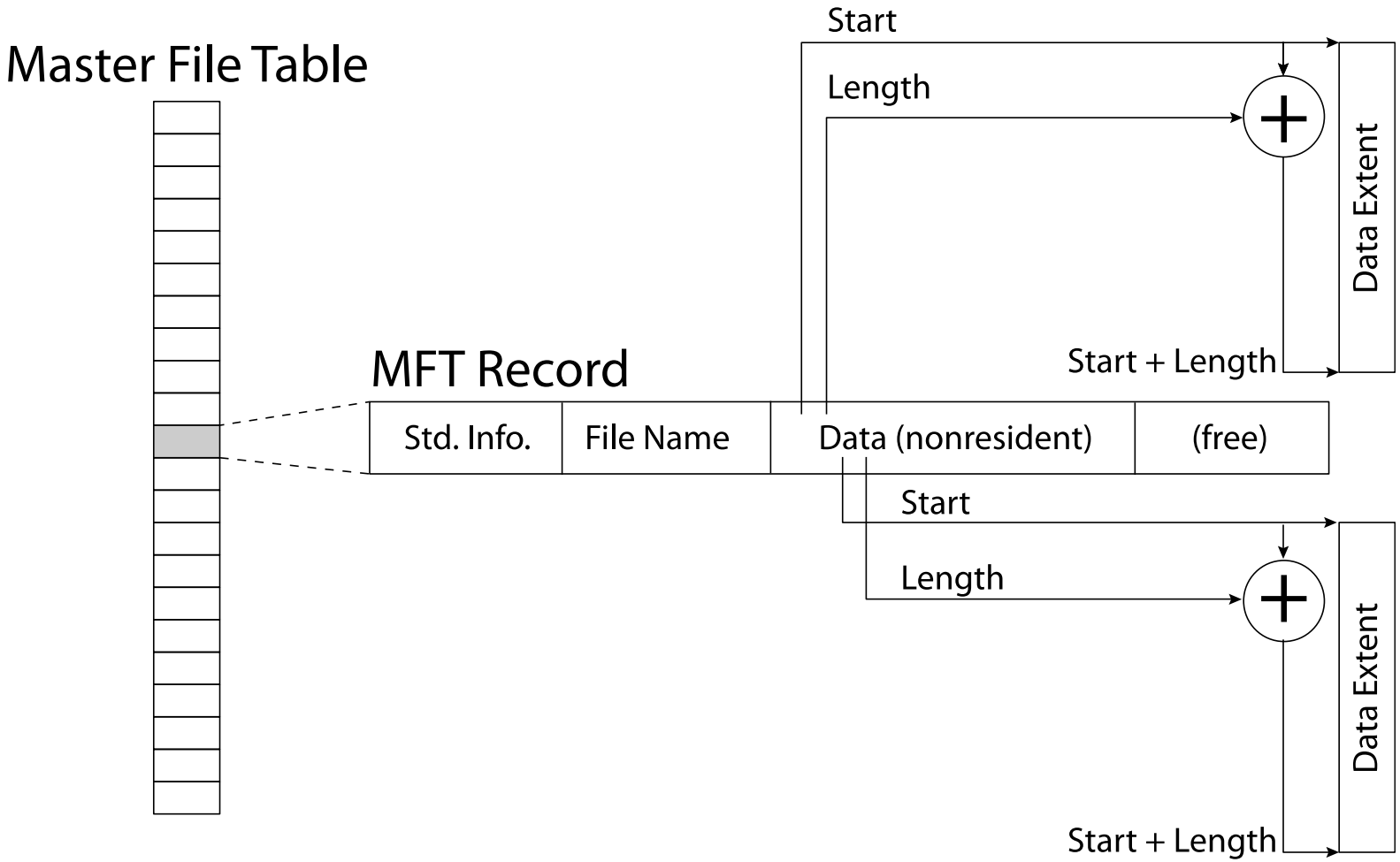
NTFS Small File



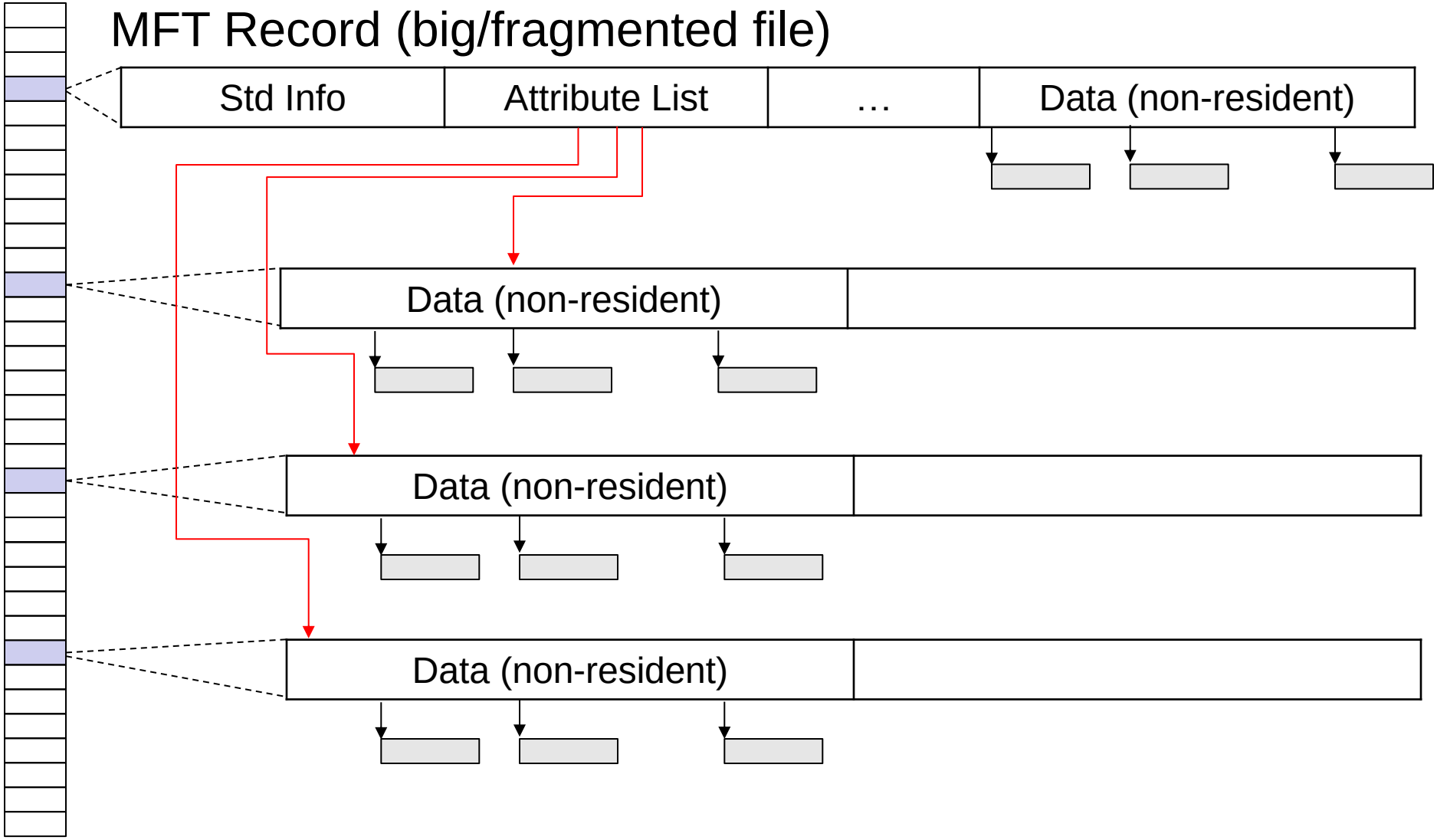
NTFS Small File



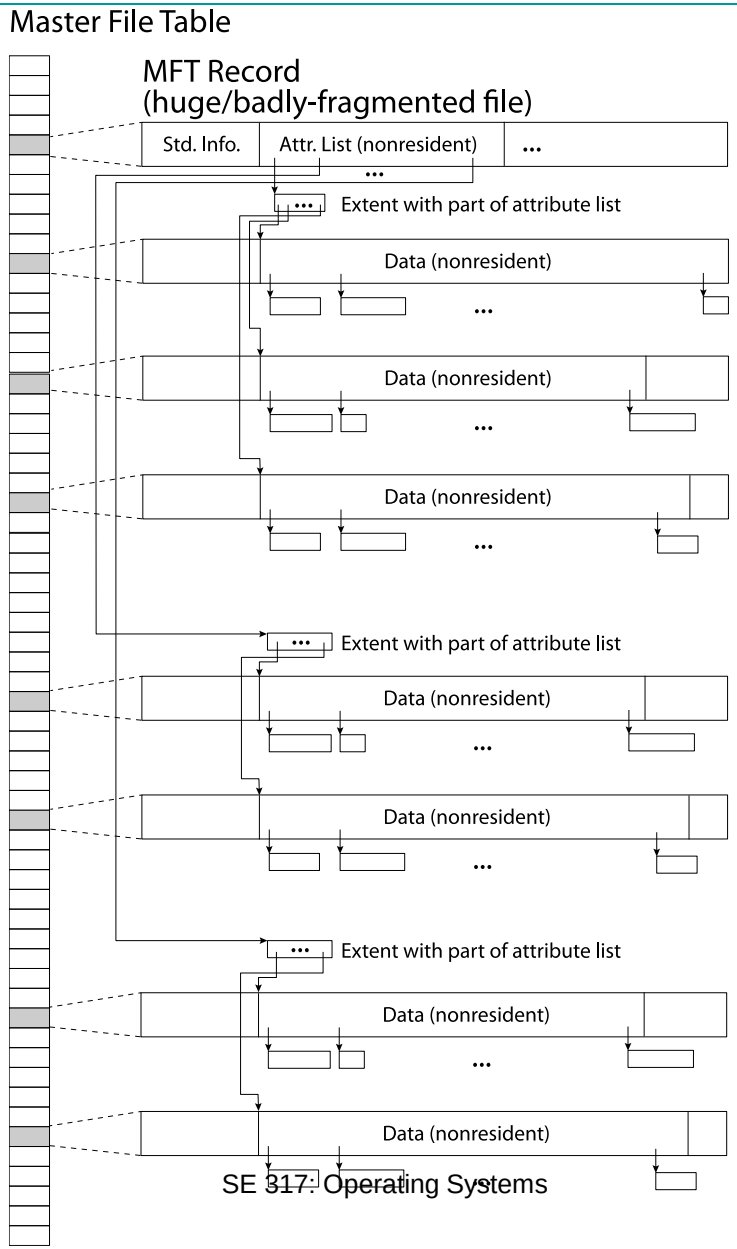
NTFS Medium File



NTFS Multiple Indirect Blocks



Huge/Badly Fragmented File



NTFS Directories

Directories implemented as B* Trees

File's number identifies its entry in MFT

MFT entry always has a file name attribute

- Human readable name, file number of parent dir

Hard link? Multiple file name attributes in MFT entry

Conclusion

- Very simply file system
- FAT
- Inodes
- Unix Fast File System (FFS)
- NTFS