File Systems: FFS, NTFS

25 January 2026
Lecture 13

Slides adapted from John Kubiatowicz (UC Berkeley)

Topics for Today

- File Systems
  - Unix Fast File System (FFS)
  - NTFS

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

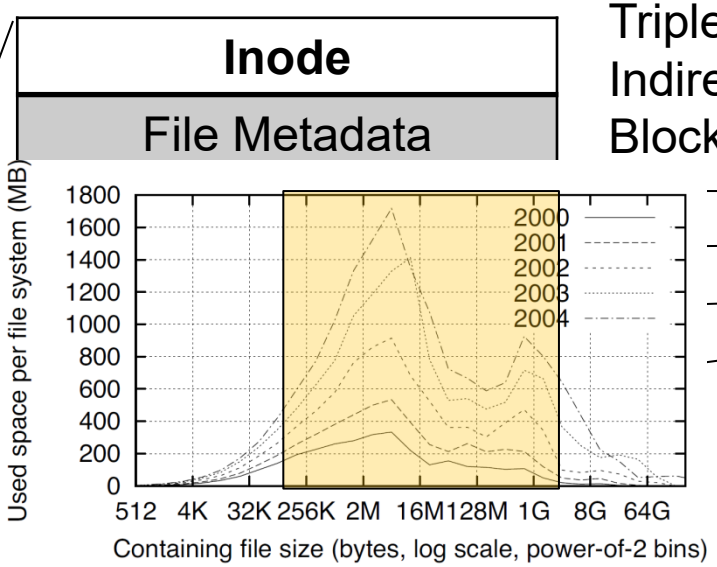
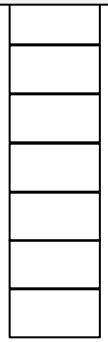
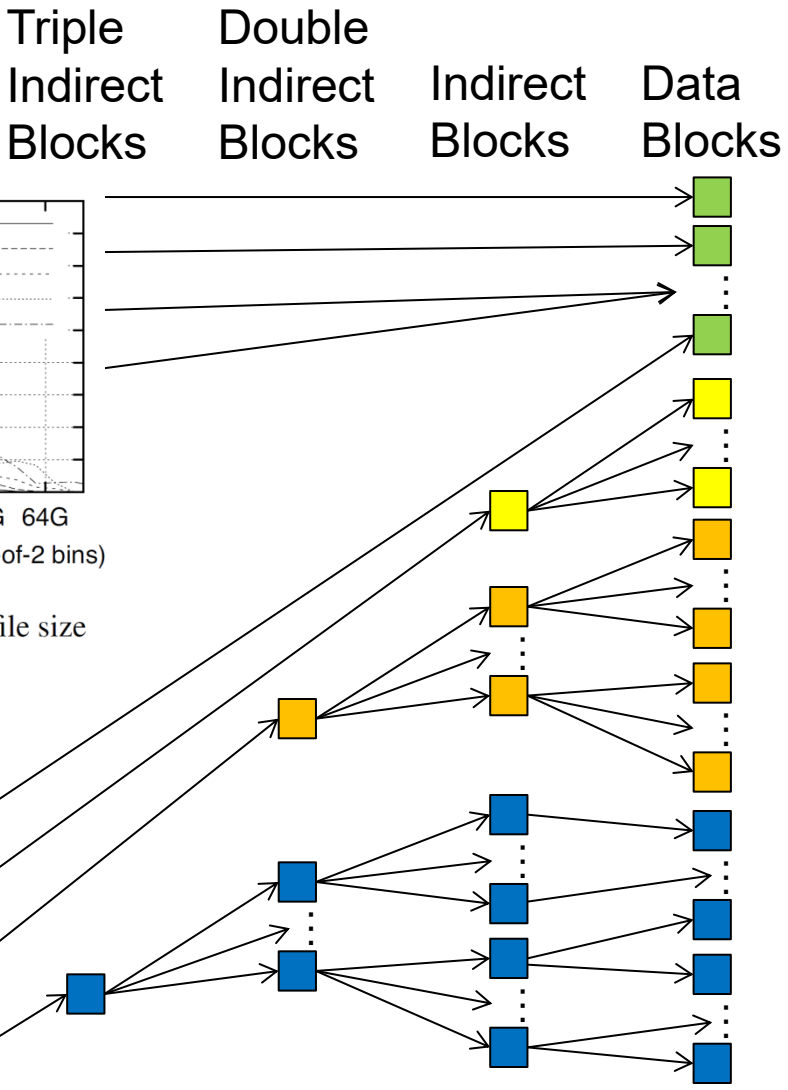


Figure 4: Histograms of bytes by containing file size





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:

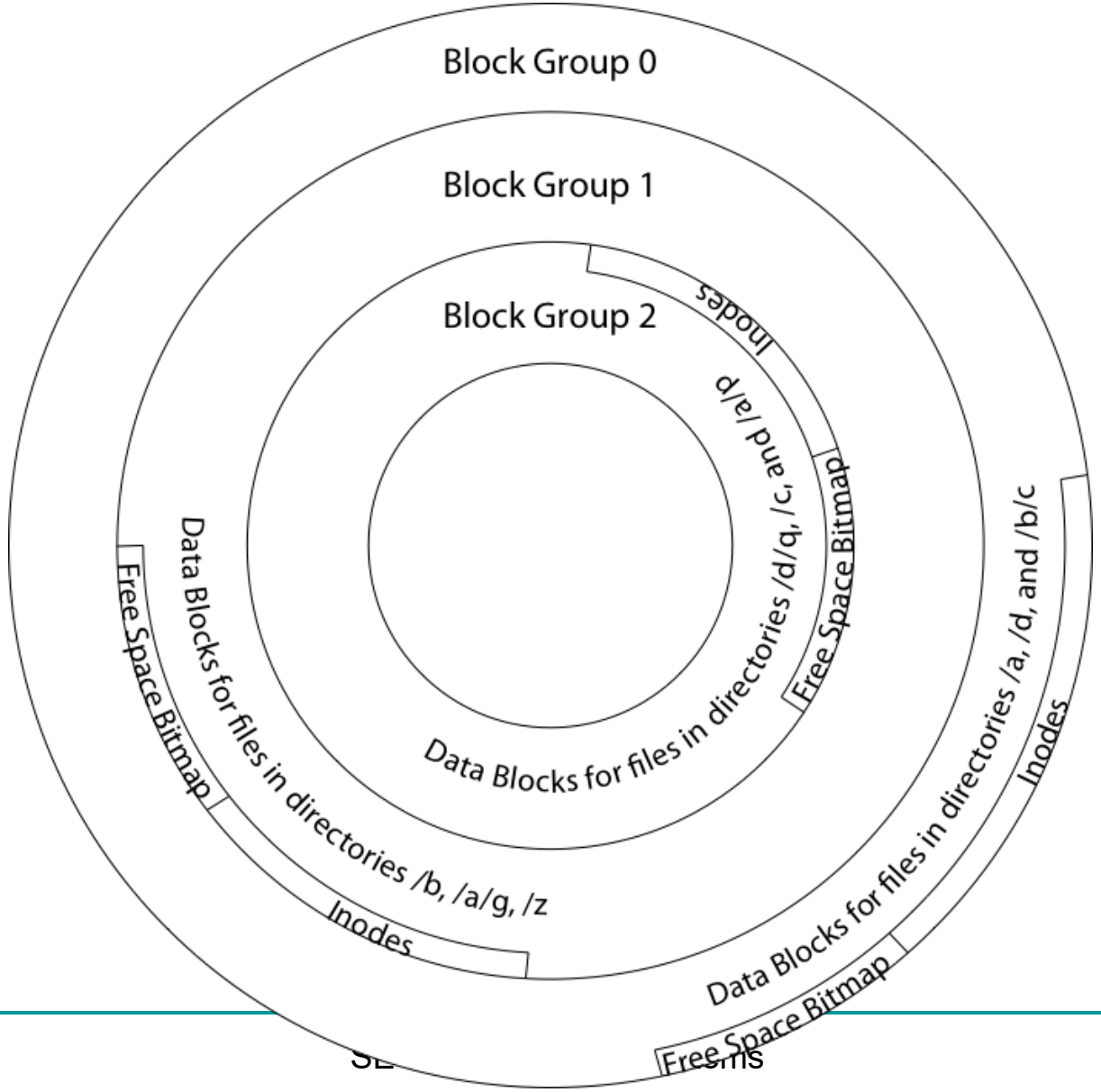
- UNIX BSD 4.2 puts a portion of the file header array on each of many cylinders. For small directories, can fit all data, file headers, etc. in same cylinder ⇒ no seeks!
- File headers much smaller than whole block (a few hundred bytes), so multiple headers fetched from disk at same time
- Reliability: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

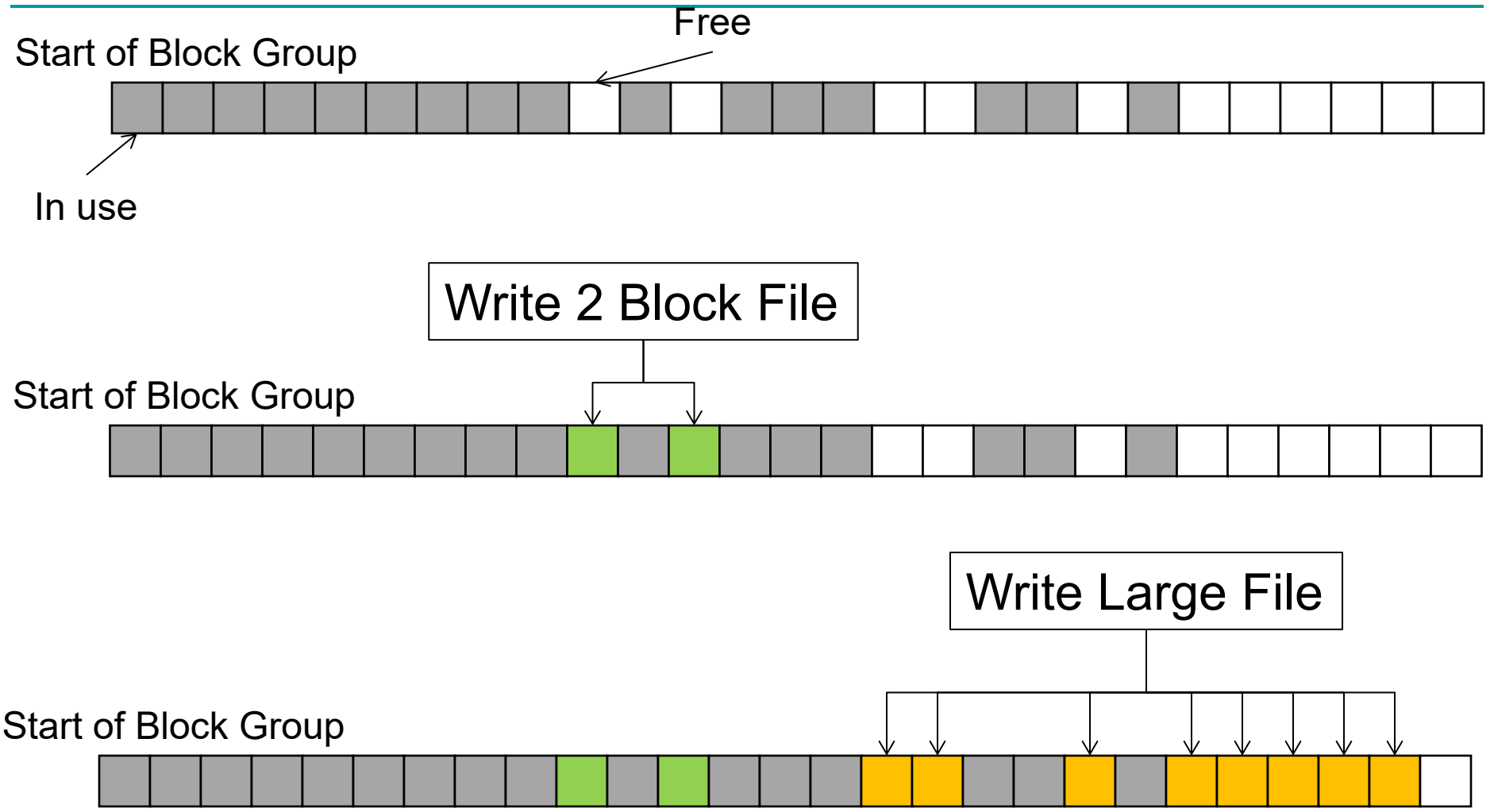
4.2 BSD Locality: Block Groups

- 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

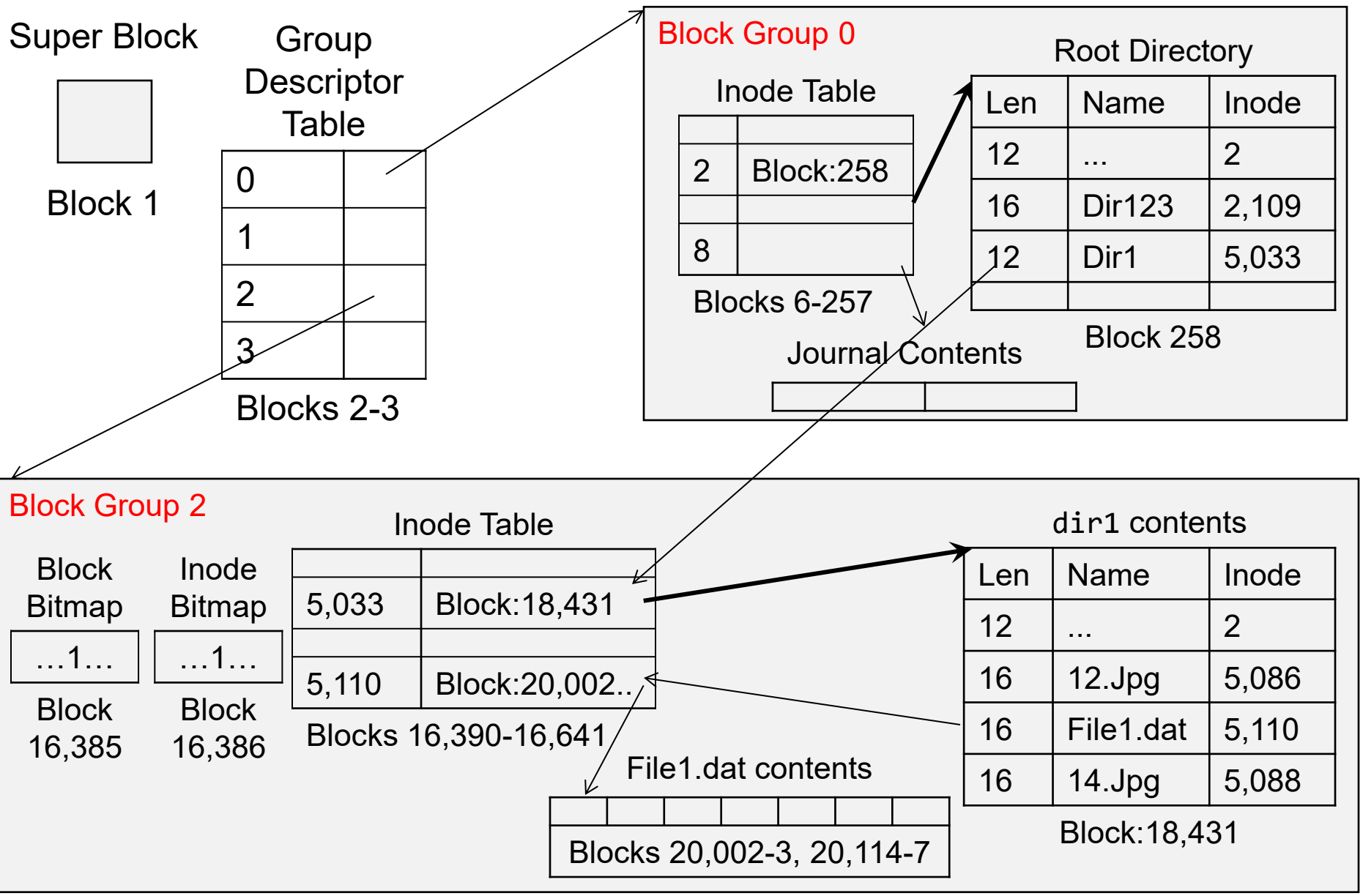
- Inefficient for tiny files (a 1 byte file requires both an inode and a data block)
- Inefficient encoding when file is mostly contiguous on disk (no equivalent to superpages)
- Need to reserve 10-20% of free space to prevent fragmentation

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

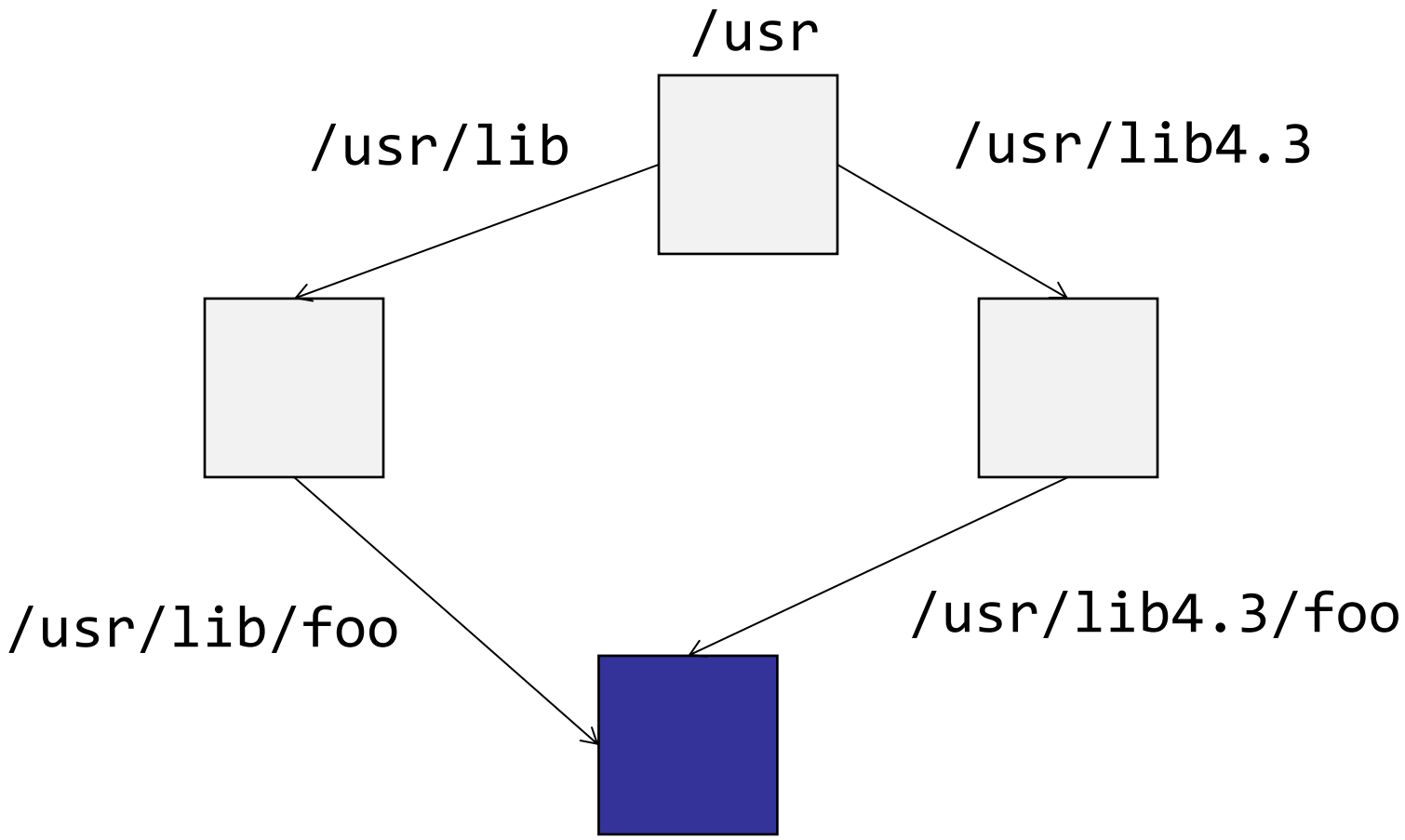
Ex: Create a `file1.dat` under `/dir1/` in ext3



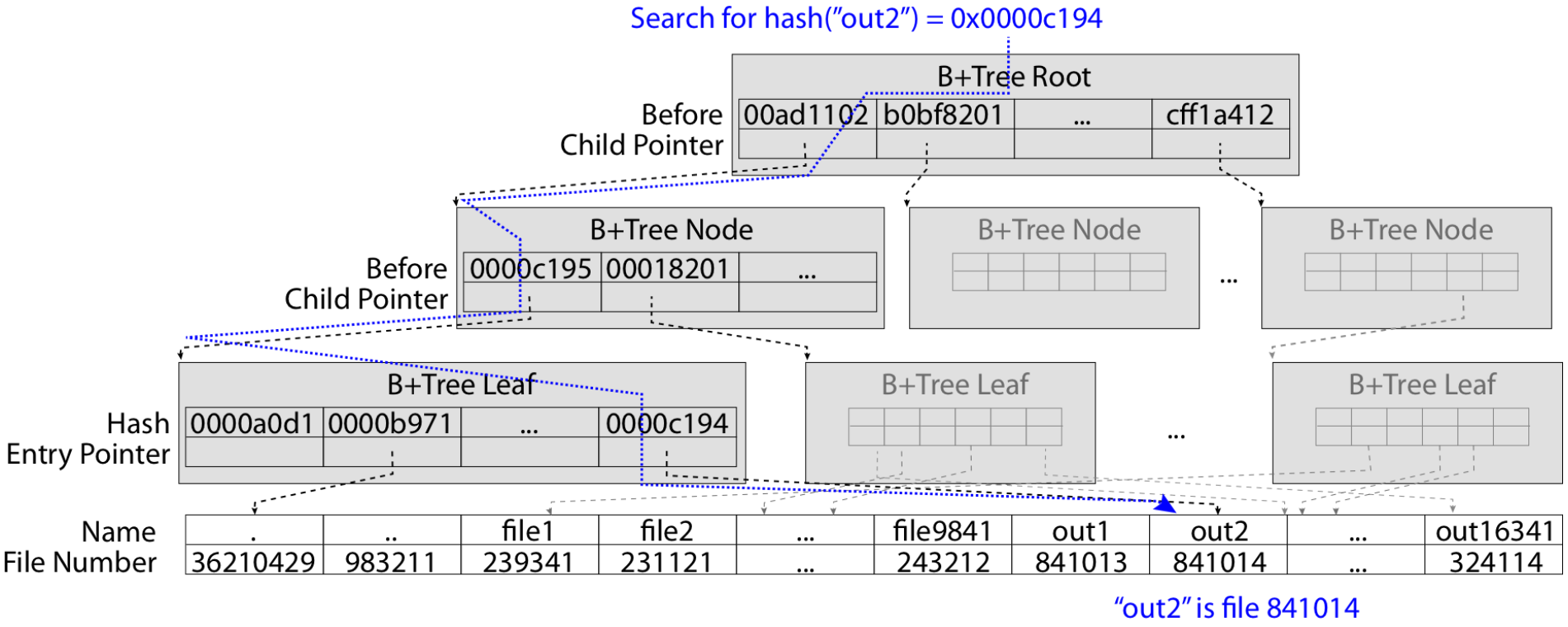
A bit more on directories

- 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

- File Systems
  - 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

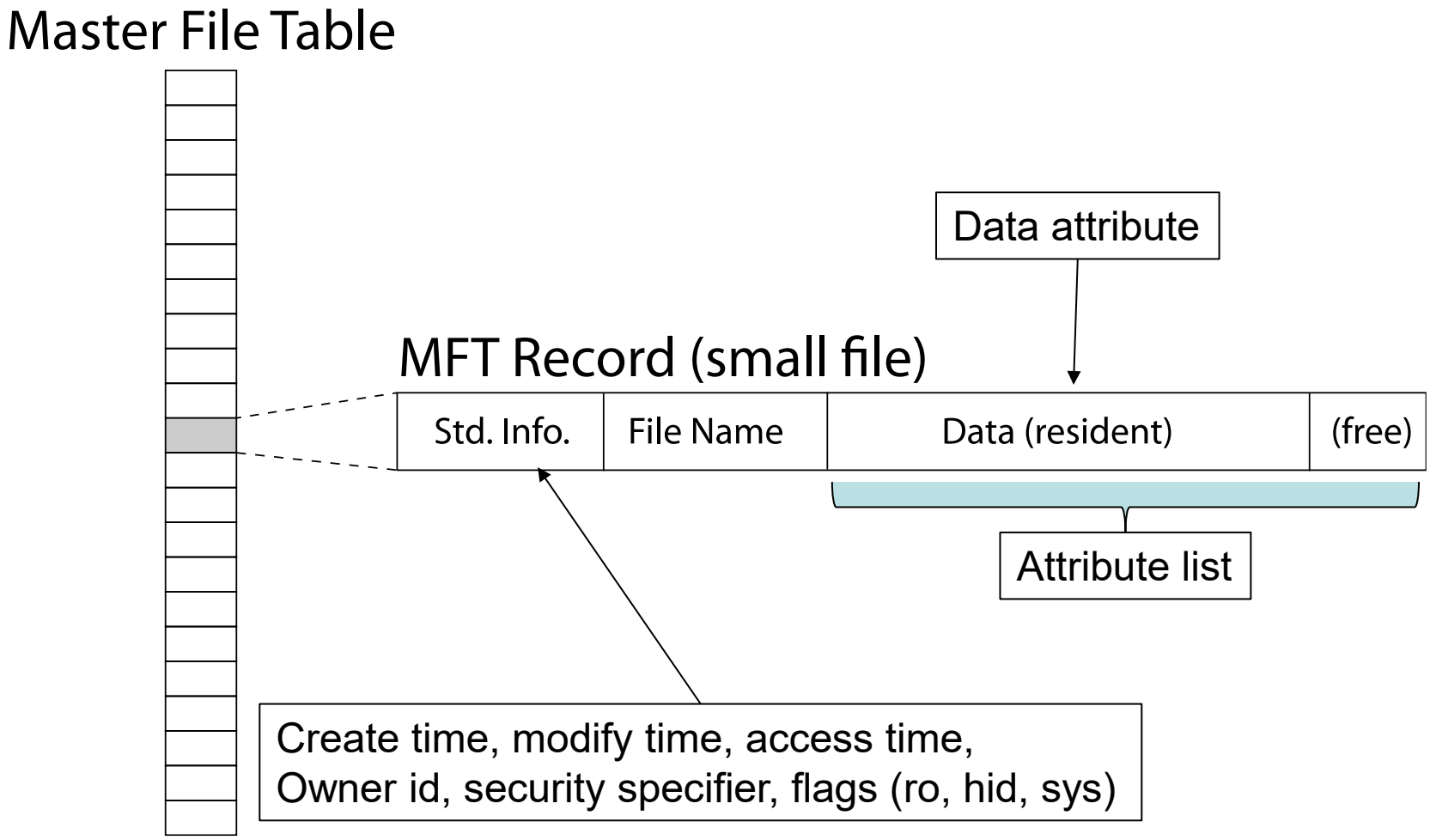
- Max 1 KB size for each table entry
- Variable-sized attribute records (data or metadata)

Each entry in MFT contains metadata plus

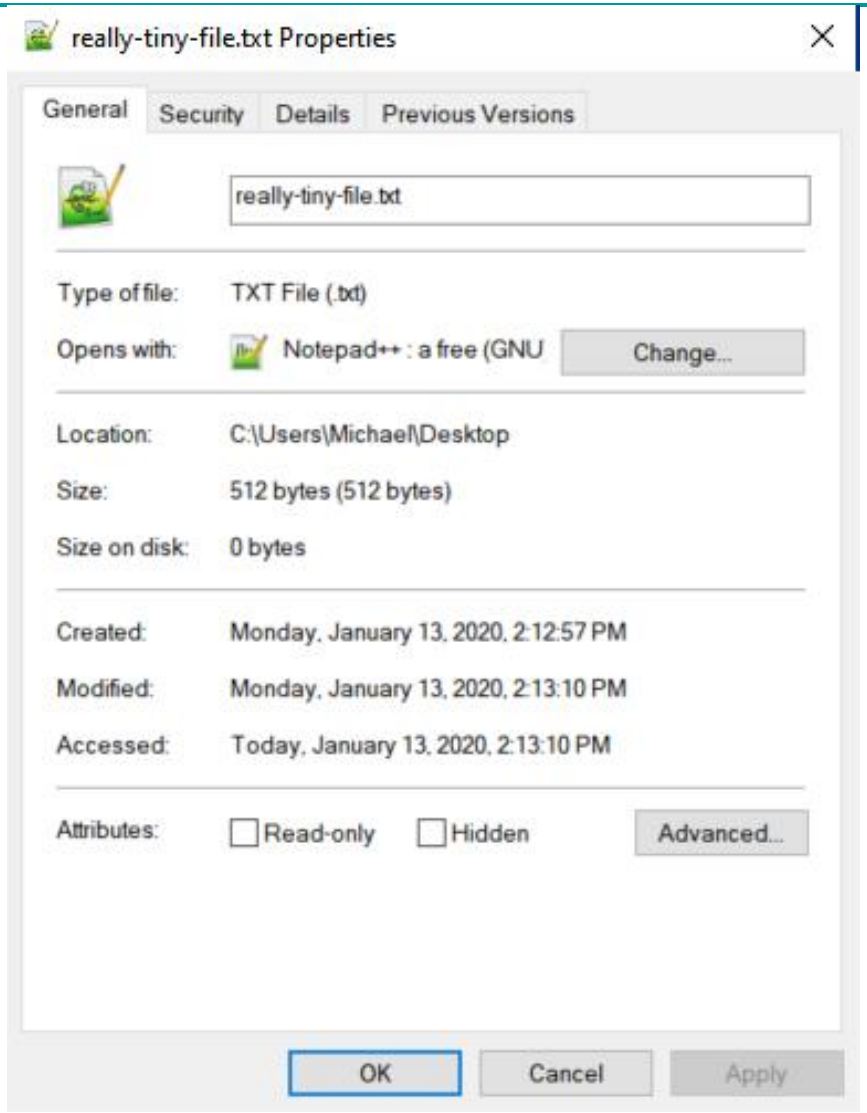
- File's data directly (for small files)
- A list of *extents* (start block, size) for file's data
- For big files: pointers to other MFT entries with *more* extent lists

Add transactions for file system changes (journaling)

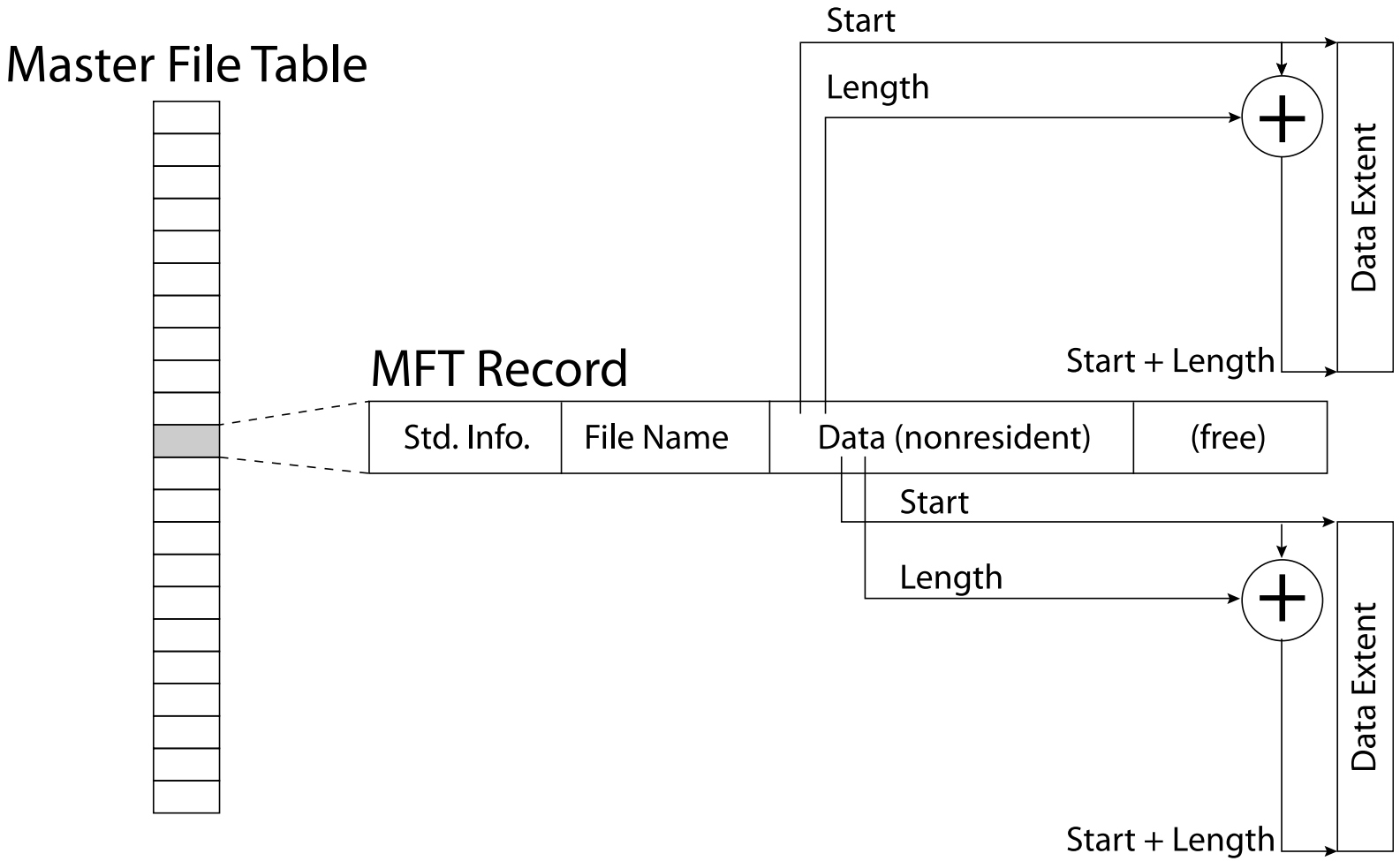
NTFS Small File



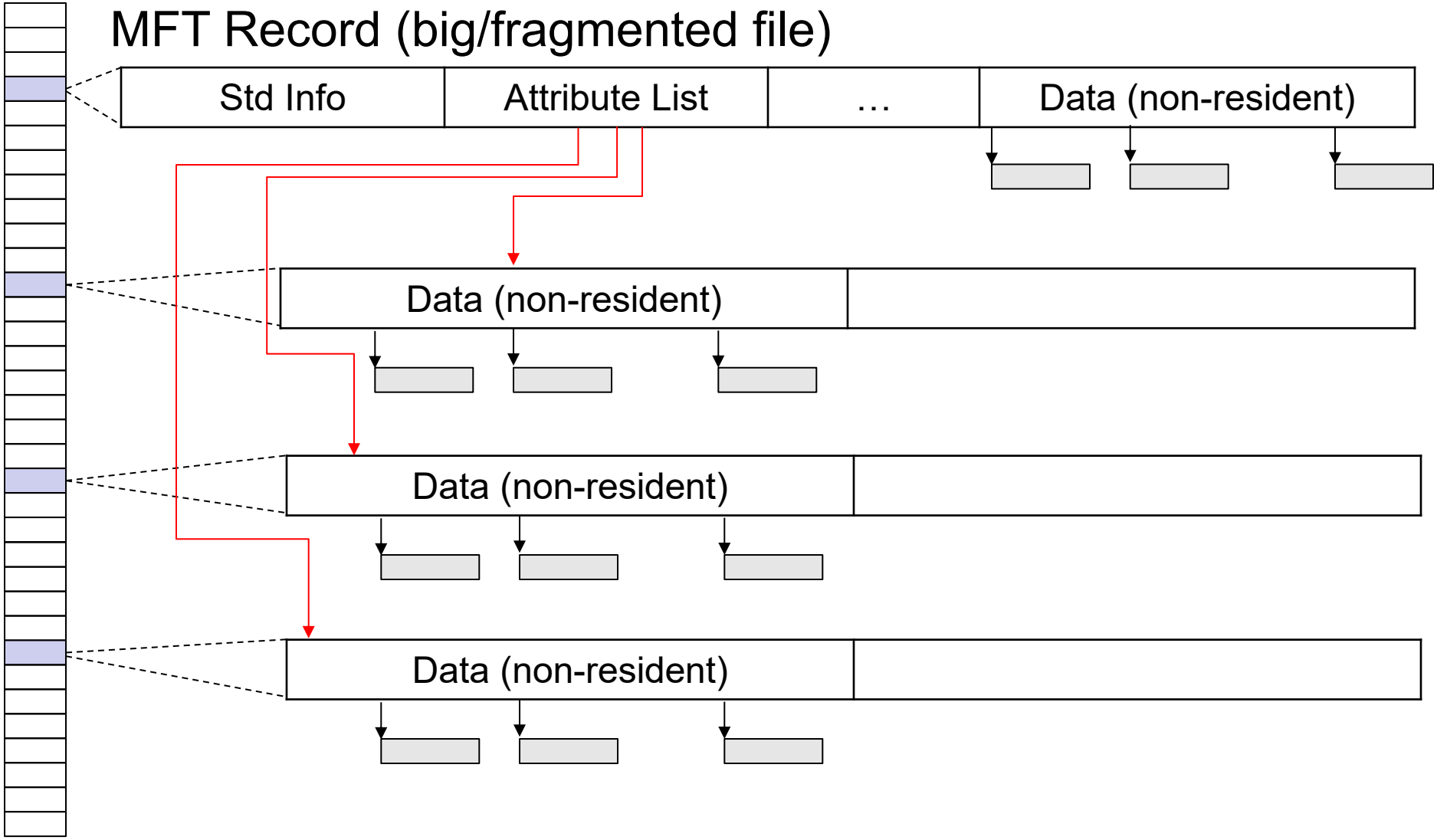
NTFS Small File



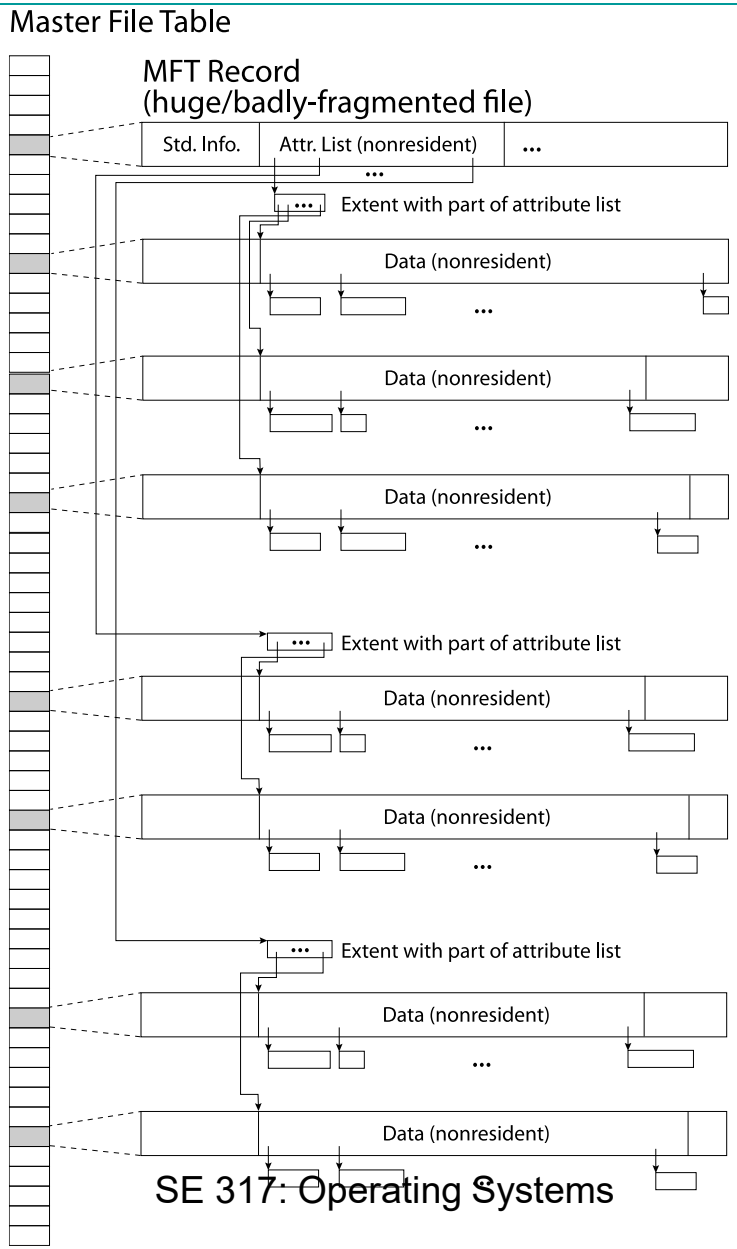
NTFS Medium File



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

- File Systems
  - Unix Fast File System (FFS)
  - NTFS