File System Design

David E. Culler

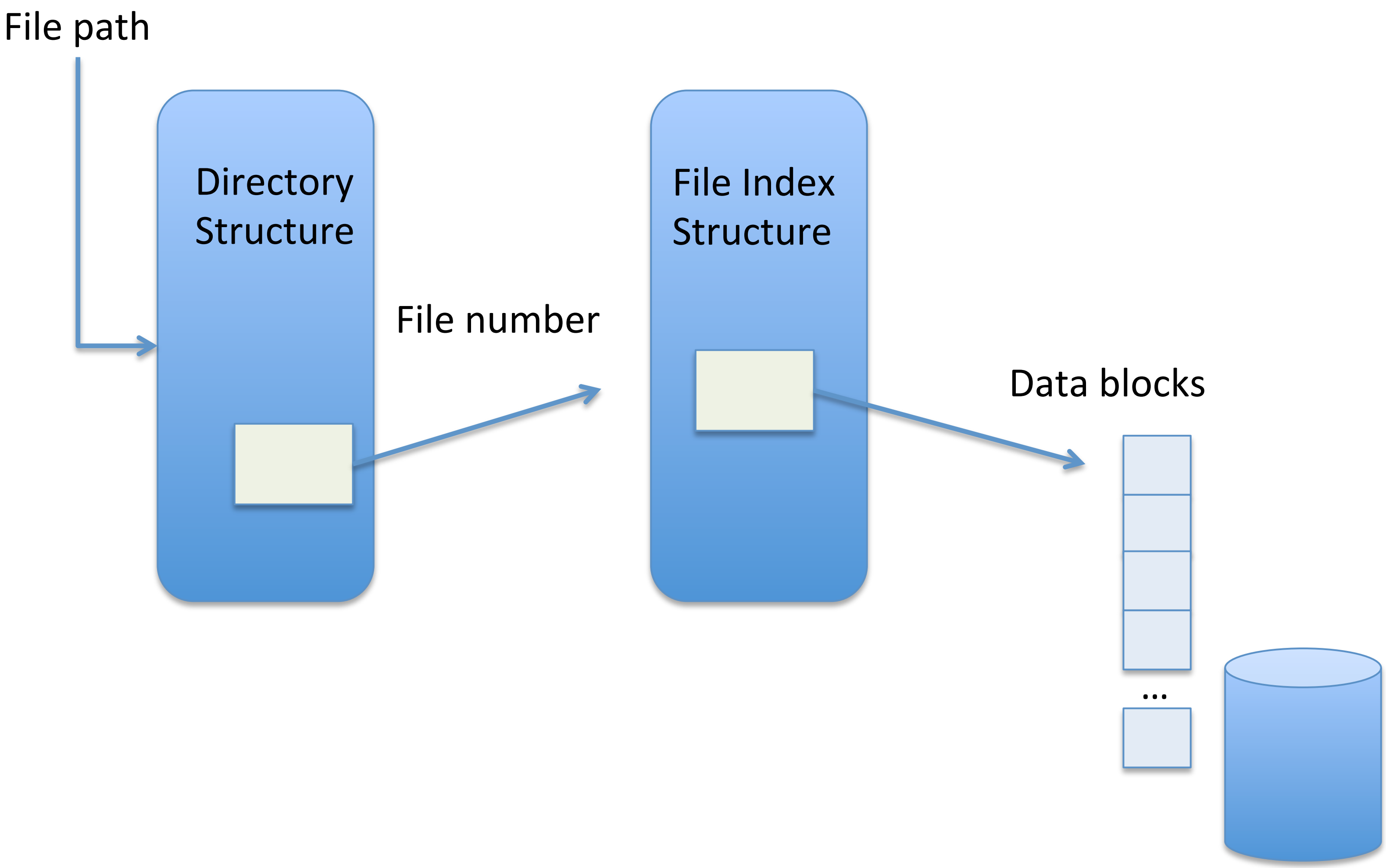
CS162 – Operating Systems and Systems Programming

Lecture 24

October 24, 2014

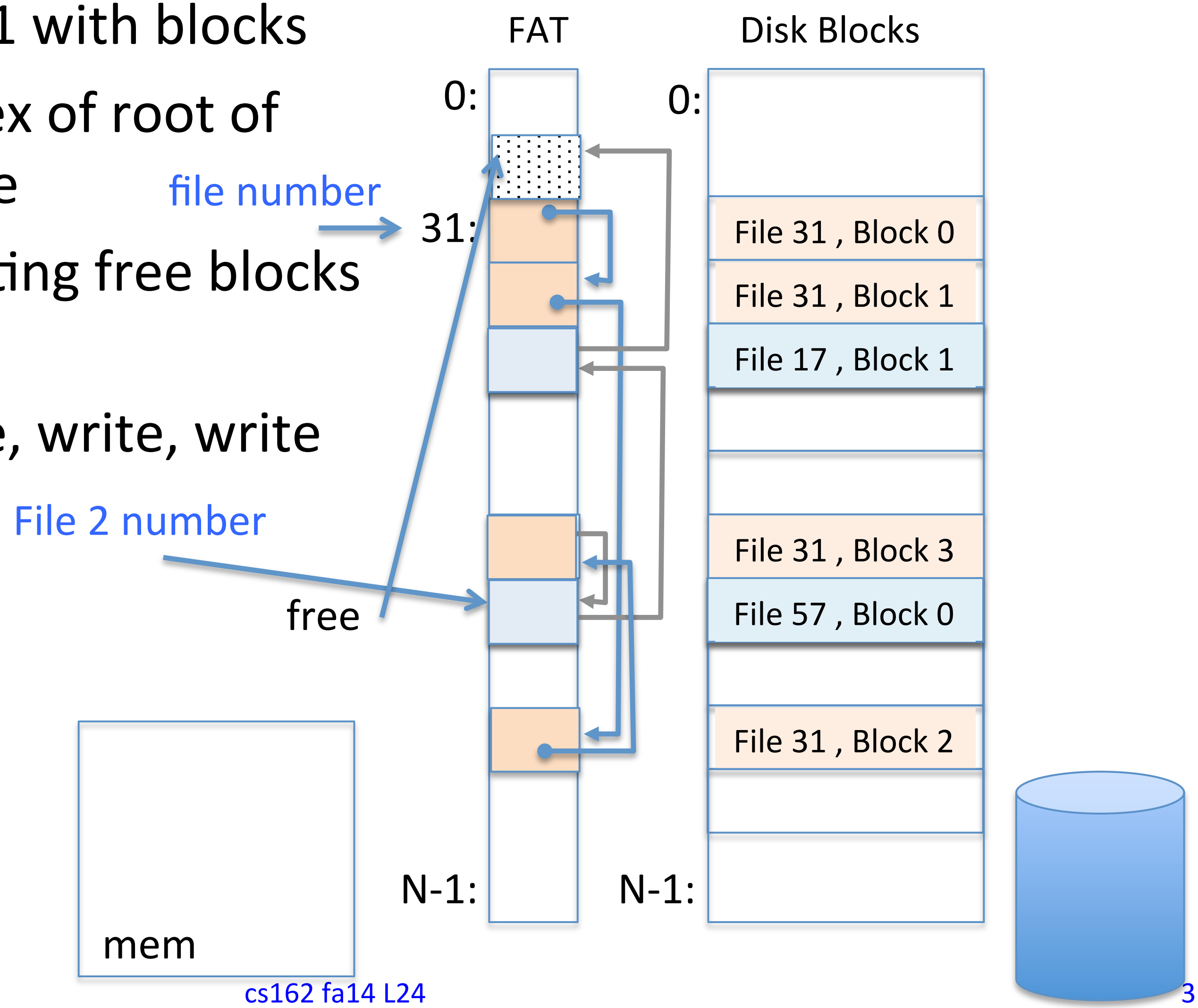
Reading: A&D 13.3
HW 4 due 10/27
Proj 2 final 11/07

Recall: Components of a File System



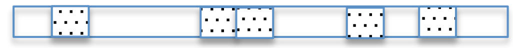
Recall: FAT (File Allocation Table)

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

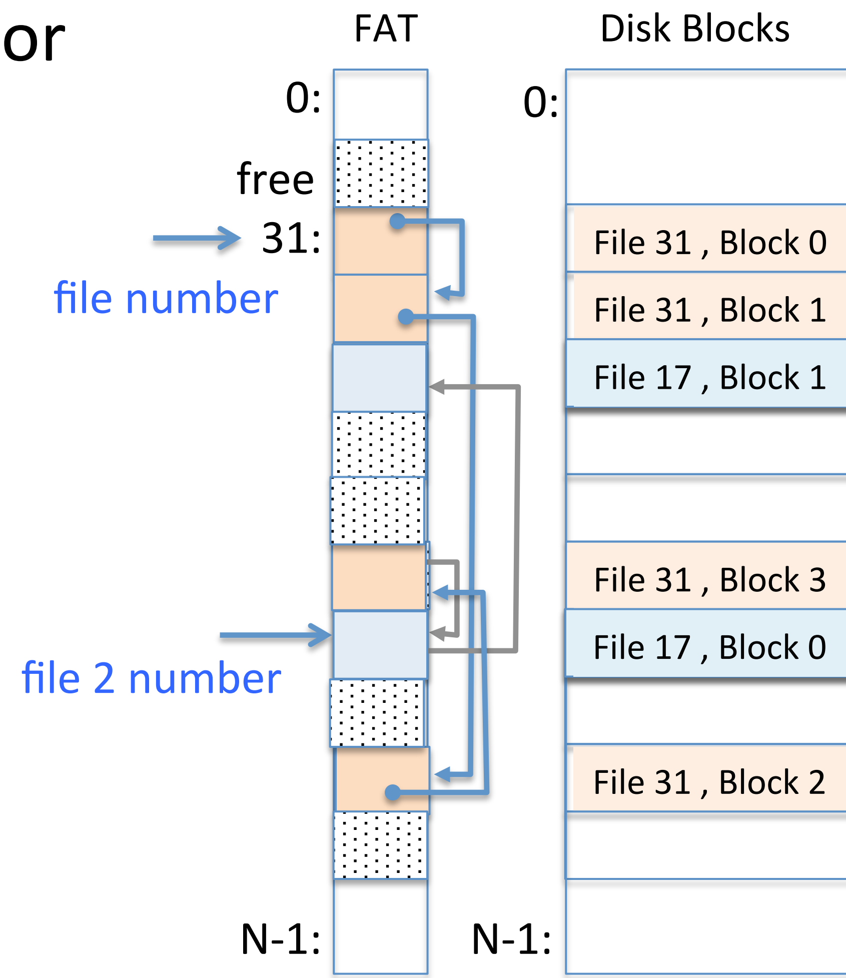


FAT Assessment

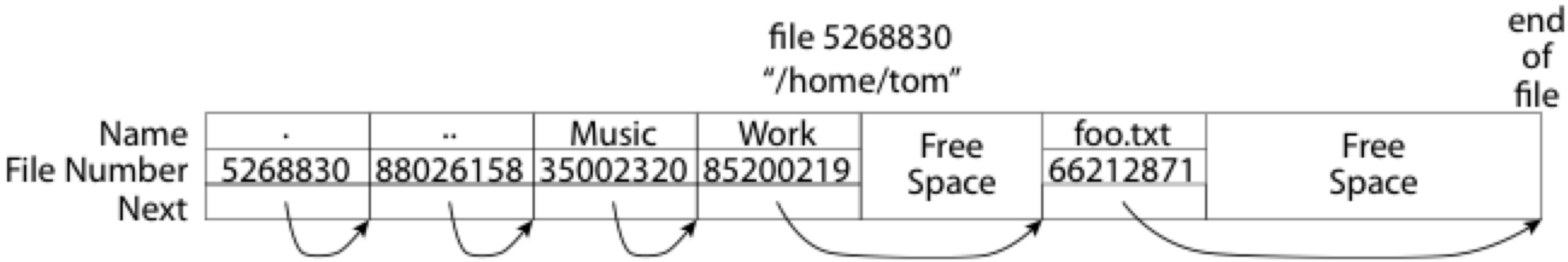
- Time to find block ??



- Free list usually just a bit vector
- Next fit algorithm
- 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 (Con't)

- 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 and index into the FAT
  – (file number = block number)

Characteristics of Files

A Five-Year Study of File-System Metadata

NITIN AGRAWAL
University of Wisconsin, Madison
and
WILLIAM J. BOLOSKY, JOHN R. DOUCEUR, and JACOB R. LORCH
Microsoft Research

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

A Five-Year Study of File-System Metadata • 9:9

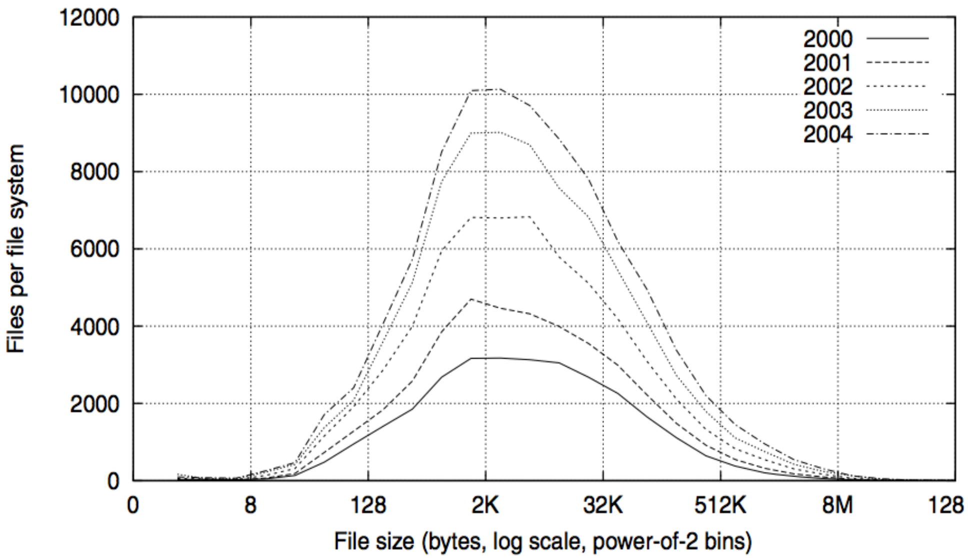


Fig. 2. Histograms of files by size.

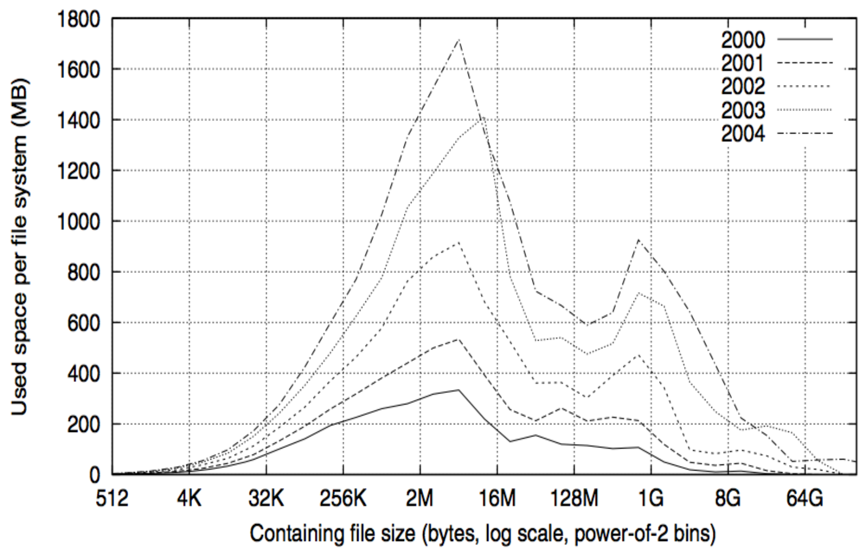
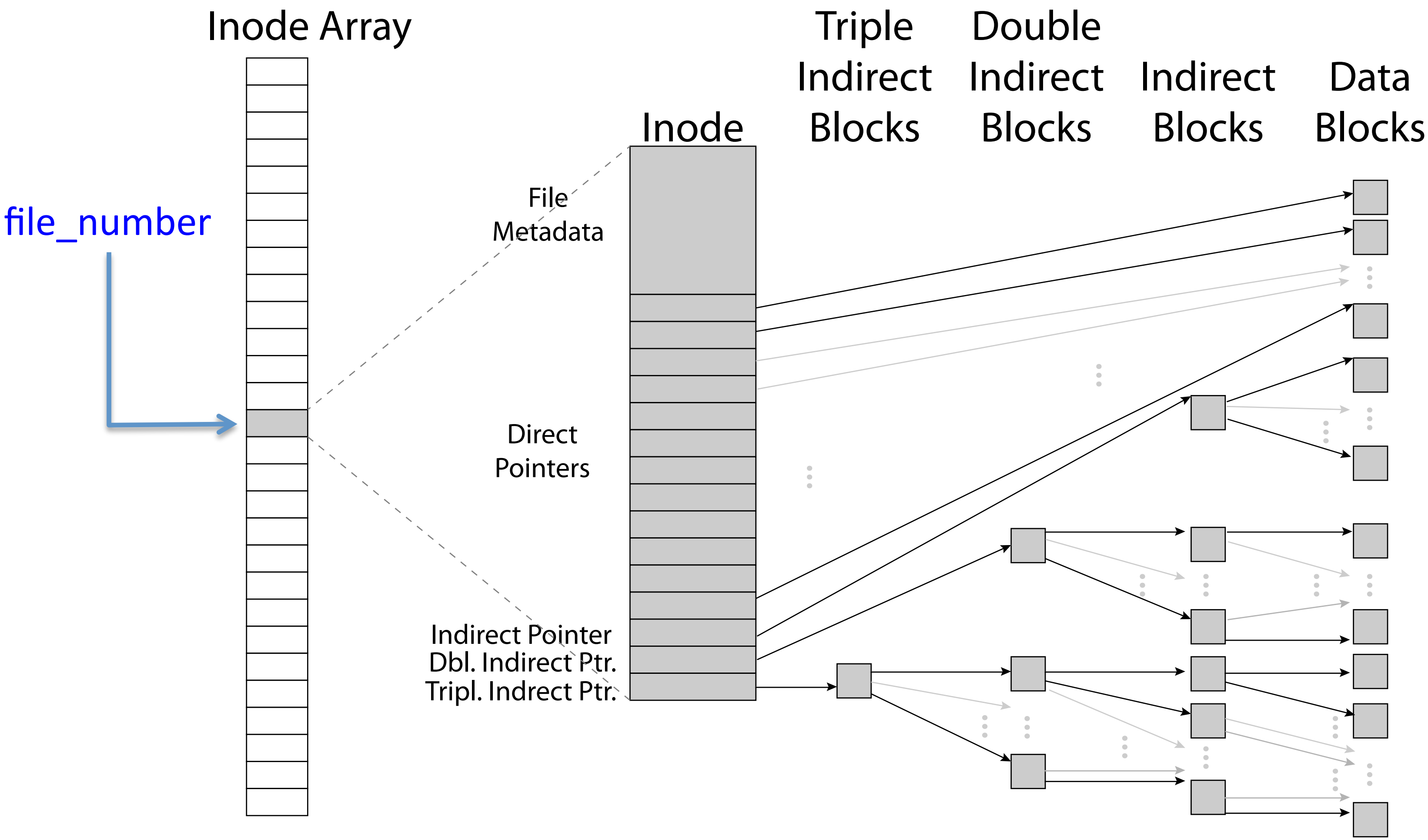


Fig. 4. Histograms of bytes by containing file size.

So what about a “real” file system

- Meet the inode



Unix Fast File System

- File Number is index into inode arrays
- Multi-level index structure
  – Great for little to large
  – Asymmetric tree with fixed sized blocks
- Metadata associated with the file
  – Rather than in the directory that points to it
- Locality Heuristics
  – Block group placement
  – Reserve space
- Scalable directory structure

An "almost real" file system

- Pintos: src/filesys/file.c, inode.c

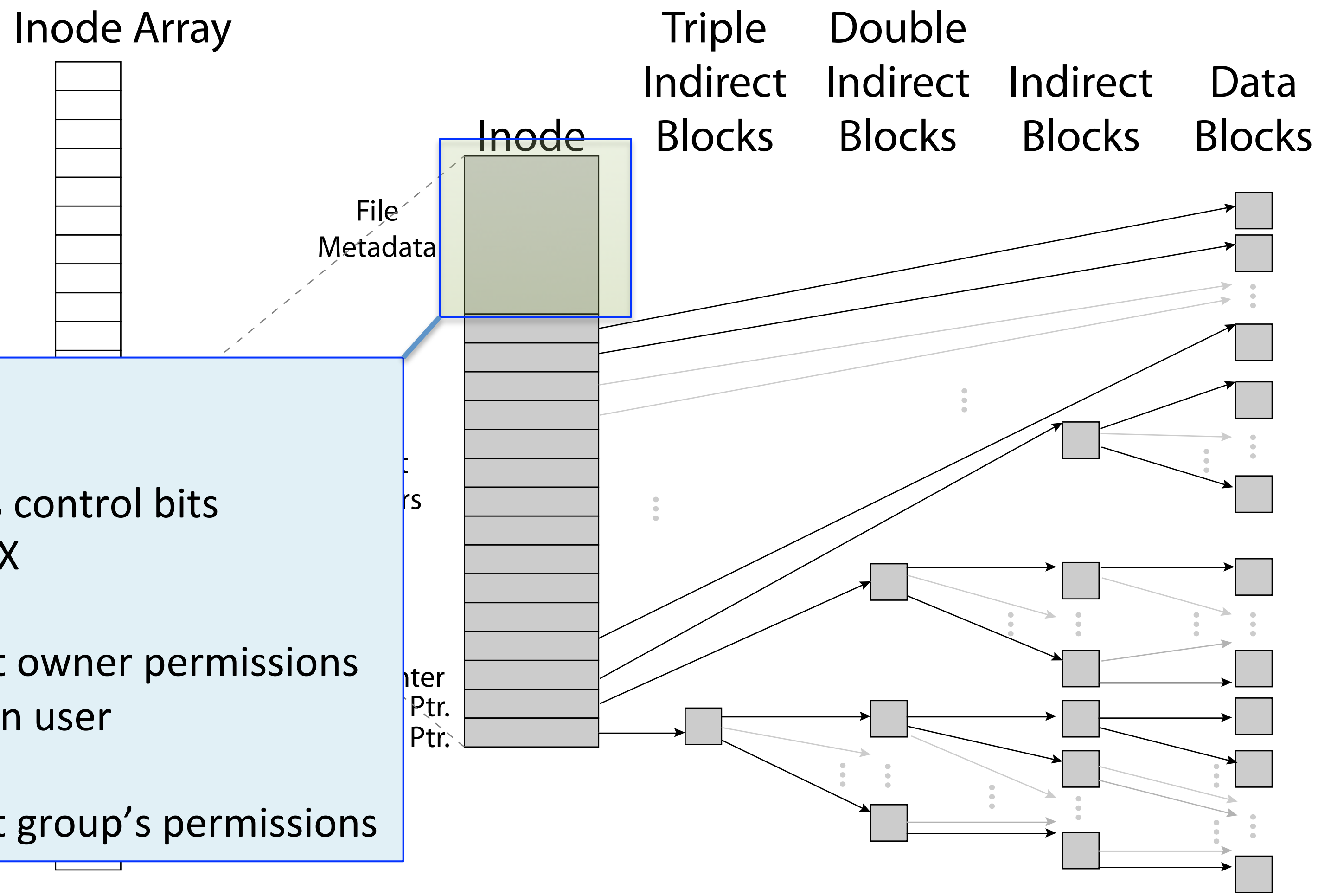
```c
/* An open file. */
struct file
  {
    struct inode *inode;        /* File's inode. */
    off_t pos;                  /* Current position. */
    bool deny_write;            /* Has file_deny_write() been called? */
  };
```

```c
/* In-memory inode. */
struct inode
  {
    struct list_elem elem;              /* Element in inode list. */
    block_sector_t sector;              /* Sector number of disk location. */
    int open_cnt;                       /* Number of openers. */
    bool removed;                       /* True if deleted, false otherwise. */
    int deny_write_cnt;                 /* 0: writes ok, >0: deny writes. */
    struct inode_disk data;             /* Inode content. */
  };
```

```c
/* On-disk inode.
   Must be exactly BLOCK_SECTOR_SIZE bytes long. */
struct inode_disk
  {
    block_sector_t start;               /* First data sector. */
    off_t length;                       /* File size in bytes. */
    unsigned magic;                     /* Magic number. */
    uint32_t unused[125];               /* Not used. */
  };
```

FFS: File Attributes

- Inode metadata



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

FFS: Data Storage

- Small files: 12 pointers direct to data blocks

Direct pointers

With 4kB blocks, sufficient
For files up to 48KB

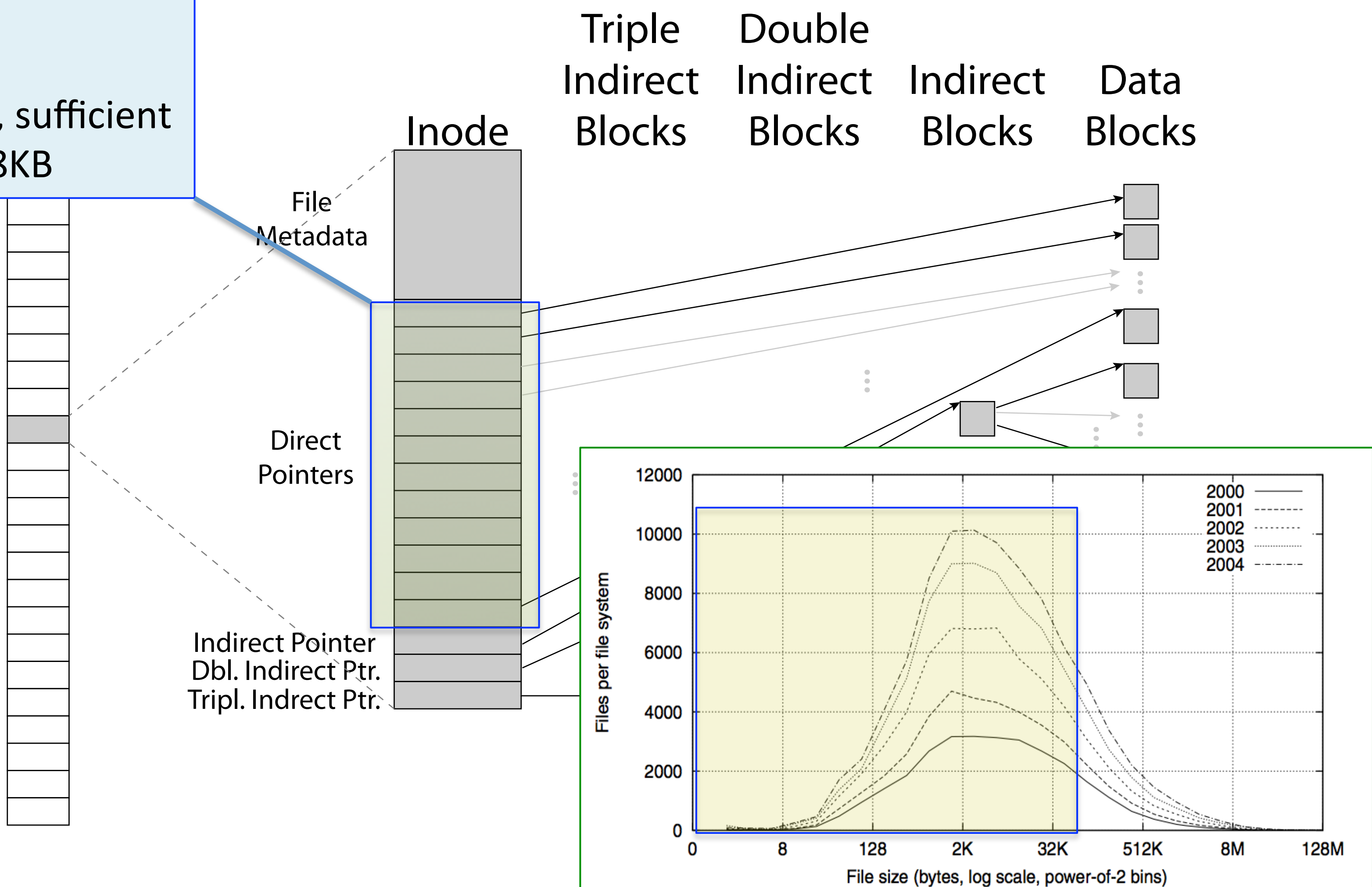
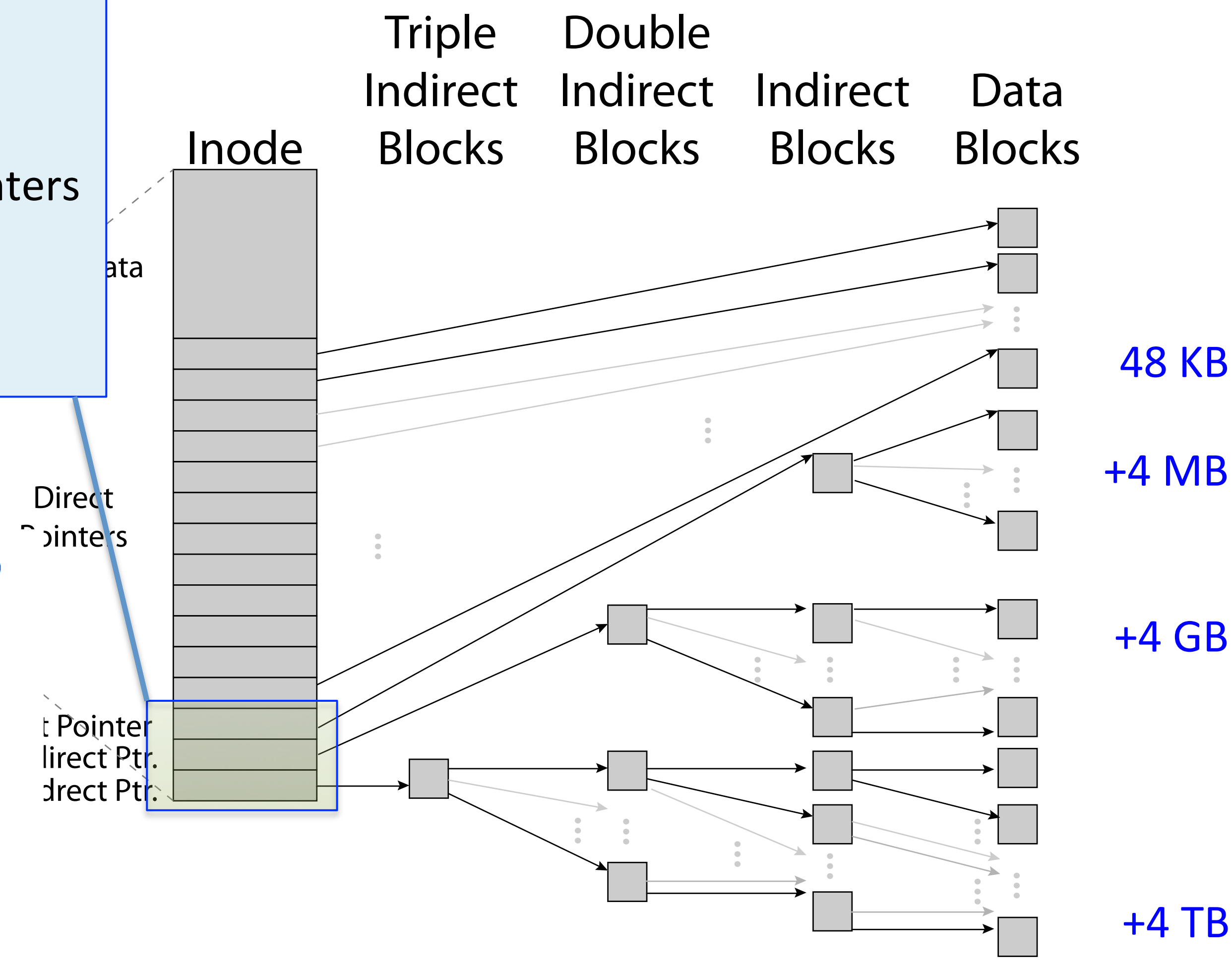


Fig. 2. Histograms of files by size.

FFS: Data Storage

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

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



A Five-Year Study of File-System Metadata • 9:9

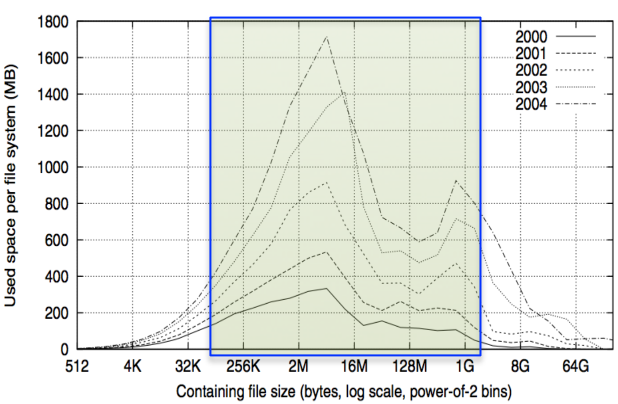


Fig. 4. Histograms of bytes by containing file size.

Freespace Management

- Bit vector with a bit per storage block
- Stored at a fixed location within the file system

Where are inodes stored?

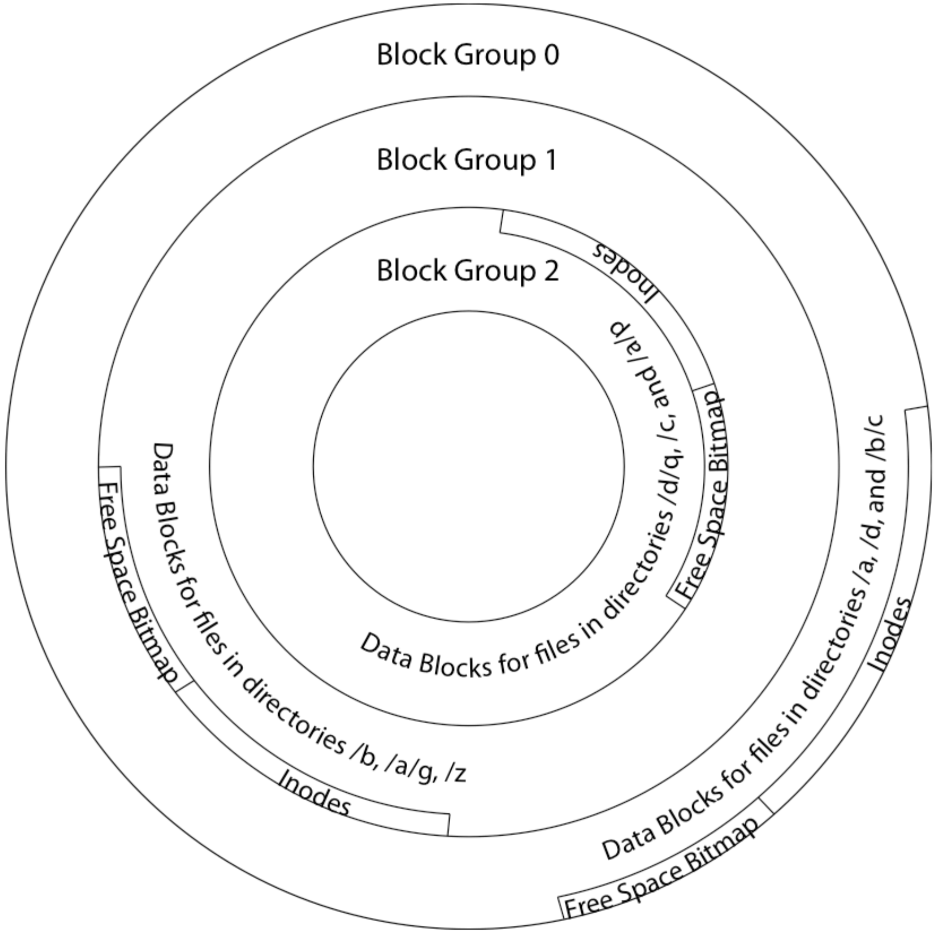
- In early UNIX and DOS/Windows' FAT file system, headers stored in special array in outermost cylinders
  - Header not stored anywhere near the data blocks. To read a small file, seek to get header, seek back to data.
  - Fixed size, set when disk is formatted. At formatting time, a fixed number of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

Where are inodes stored?

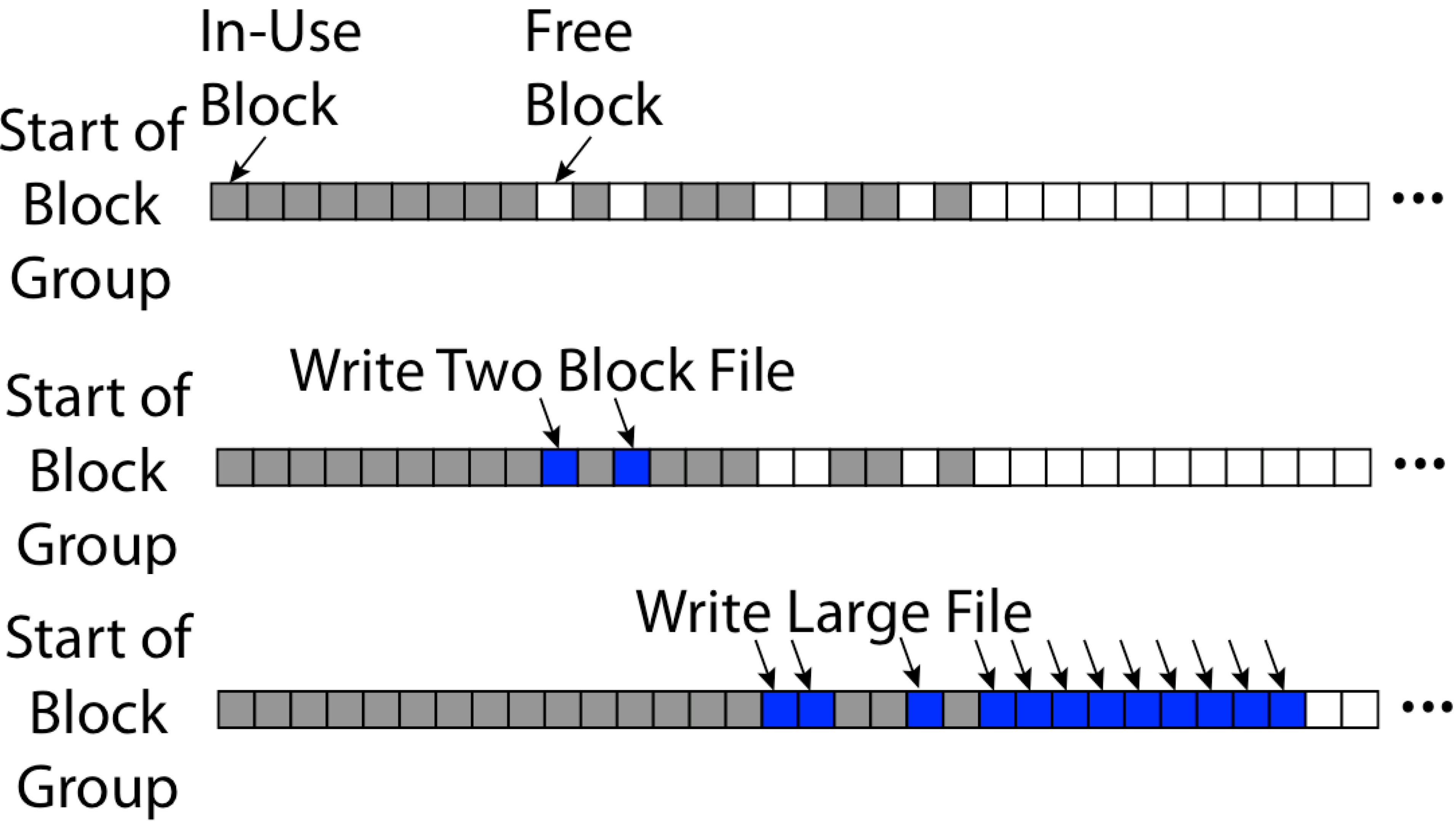
- Later versions of UNIX moved the header information to be closer to the data blocks
  - Often, inode for file stored in same "cylinder group" as parent directory of the file (makes an `ls` of that directory run fast).
  - Pros:
    - UNIX BSD 4.2 puts a portion of the file header array on each of many 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

Locality: Block Groups

- File system volume is divided into a set of block groups
  - Close set of tracks
- File data blocks, metadata, and free space are 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 block
  - Few little holes at start, big sequential runs at end of group
  - Avoids fragmentation
  - Sequential layout for big
- Reserve space in the BG



FFS First Fit Block Allocation



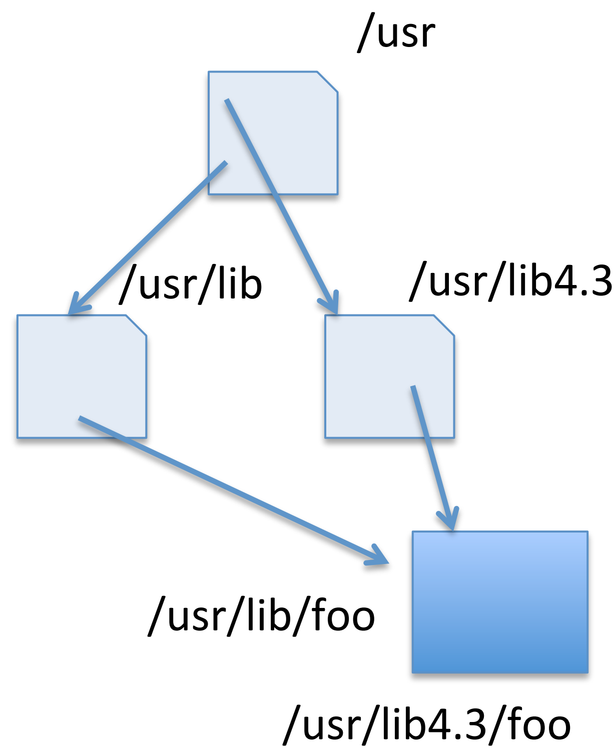
space at end

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

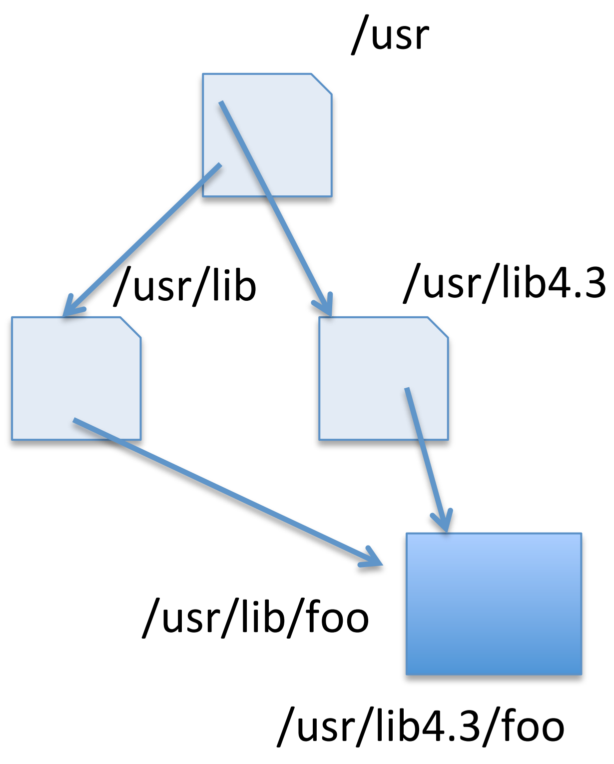
Bit more on directories

- Stored in files, can be read, but 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
- libc support
  - DIR * opendir (const char *dirname)
  - struct dirent * readdir (DIR *dirstream)
  - int readdir_r (DIR *dirstream, struct dirent *entry, struct dirent **result)



When can a file be deleted ?

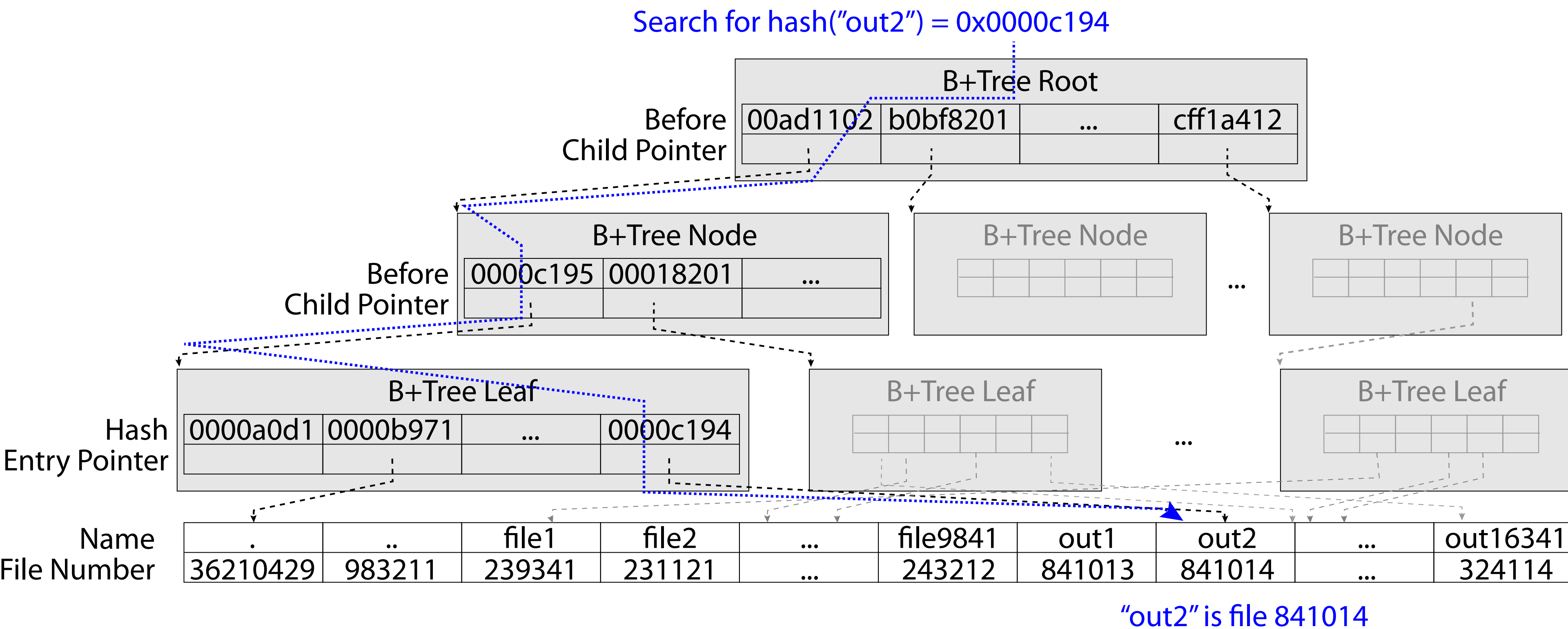
- Maintain reference count of links to the file.
- Delete after the last reference is gone.



Links

- Hard link
  - Sets another directory entry to contain the file number for the file
  - Creates another name (path) for the file
  - Each is “first class”
- Soft link or Symbolic Link
  - Directory entry contains the name of the file
  - Map one name to another name

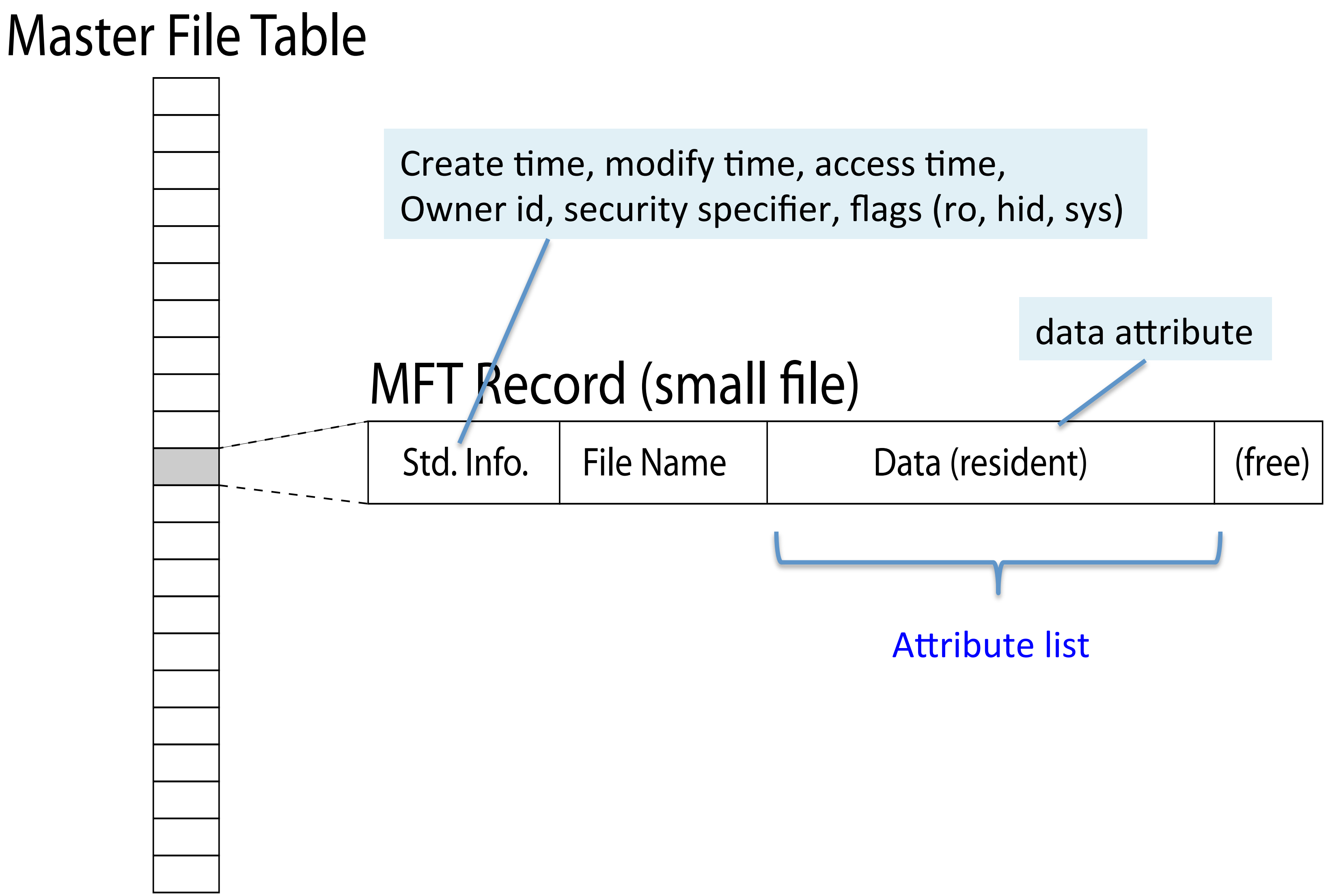
Large Directories: B-Trees



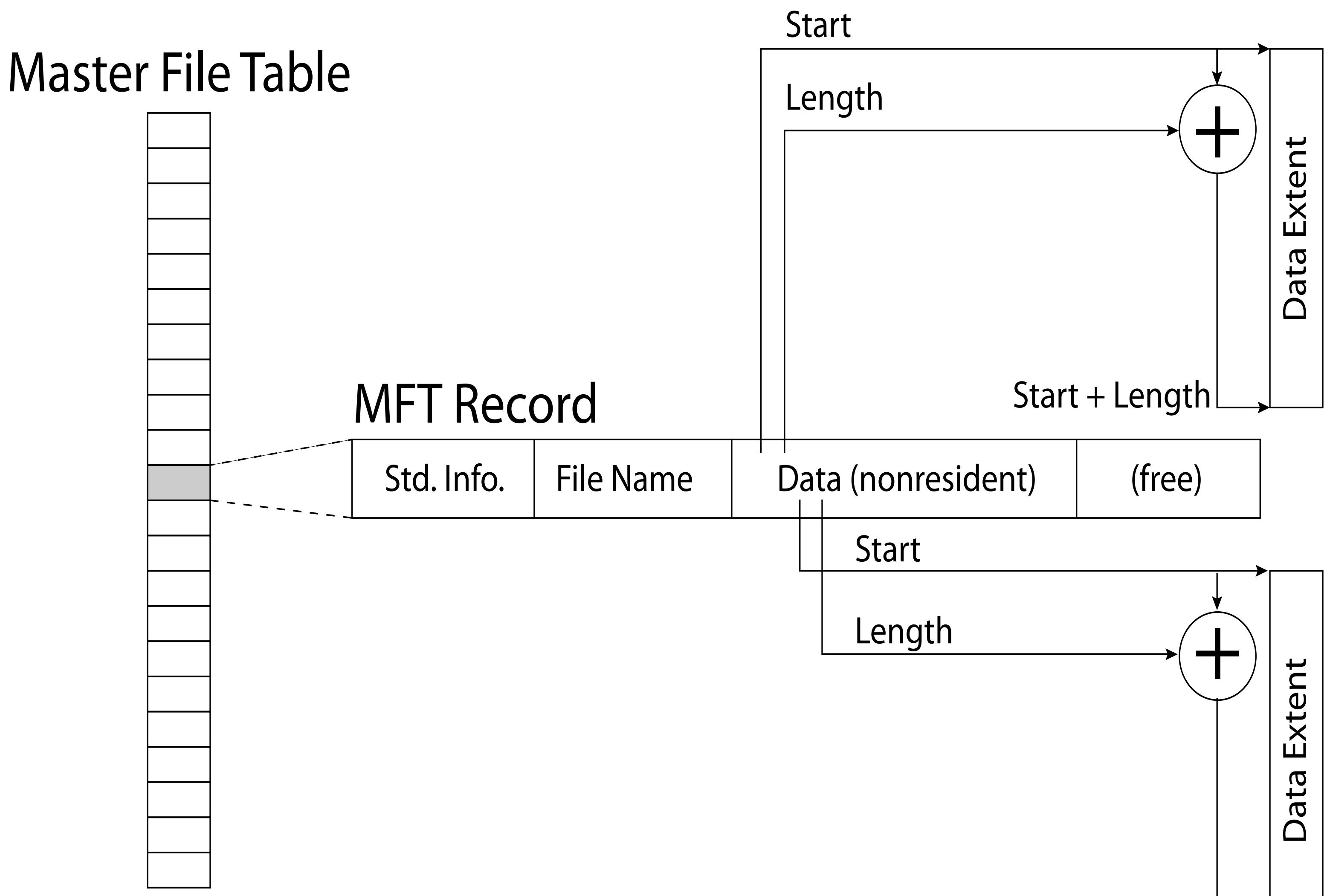
NTFS

- Master File Table
  - Flexible 1KB storage for metadata and data
  - Variable-sized attribute records (data or metadata)
  - Extend with variable depth tree (non-resident)
- Extents – variable length contiguous regions
  - Block pointers cover runs of blocks
  - Similar approach in linux (ext4)
  - File create can provide hint as to size of file
- Journalling for reliability
  - Discussed next week

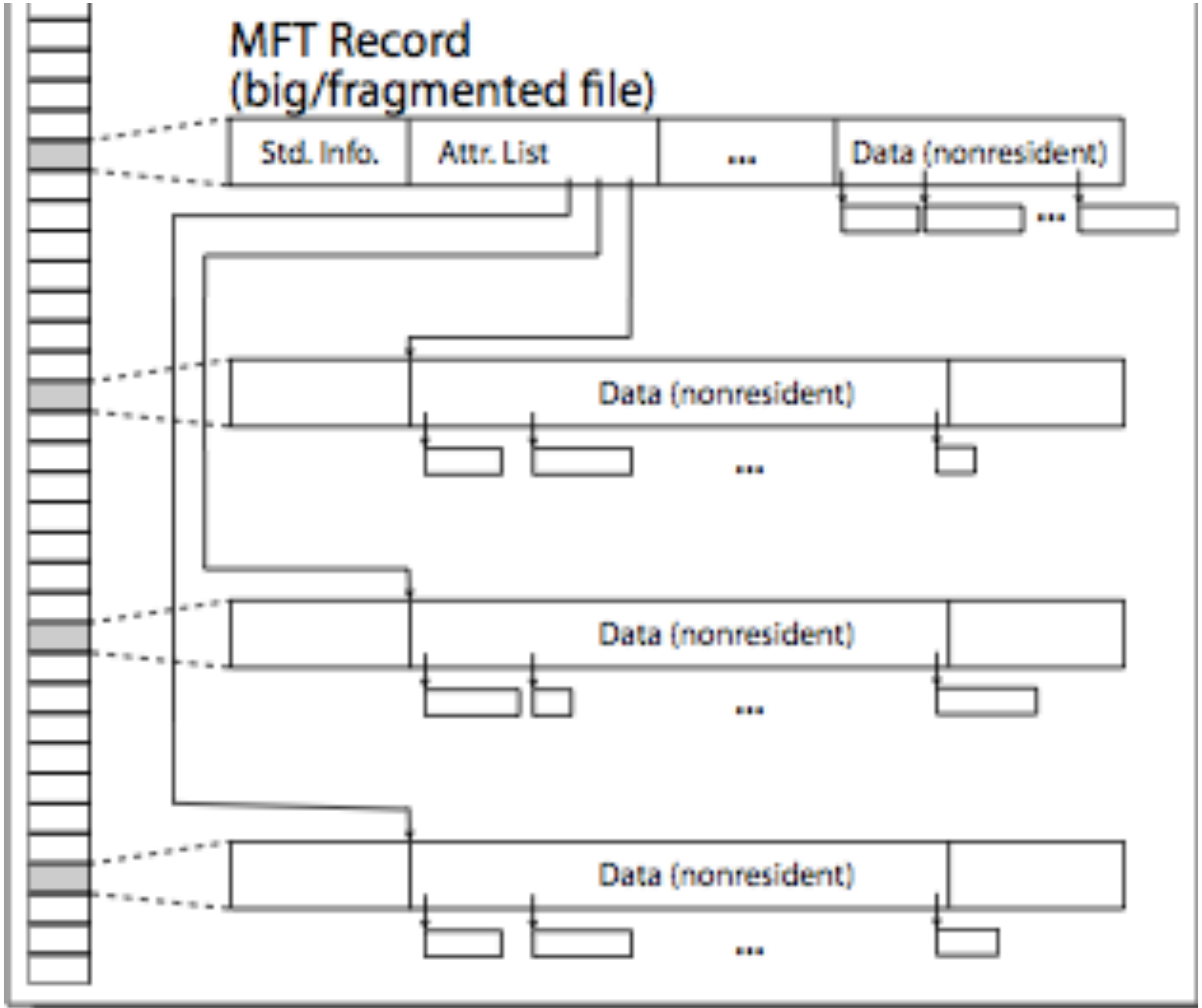
NTFS Small File

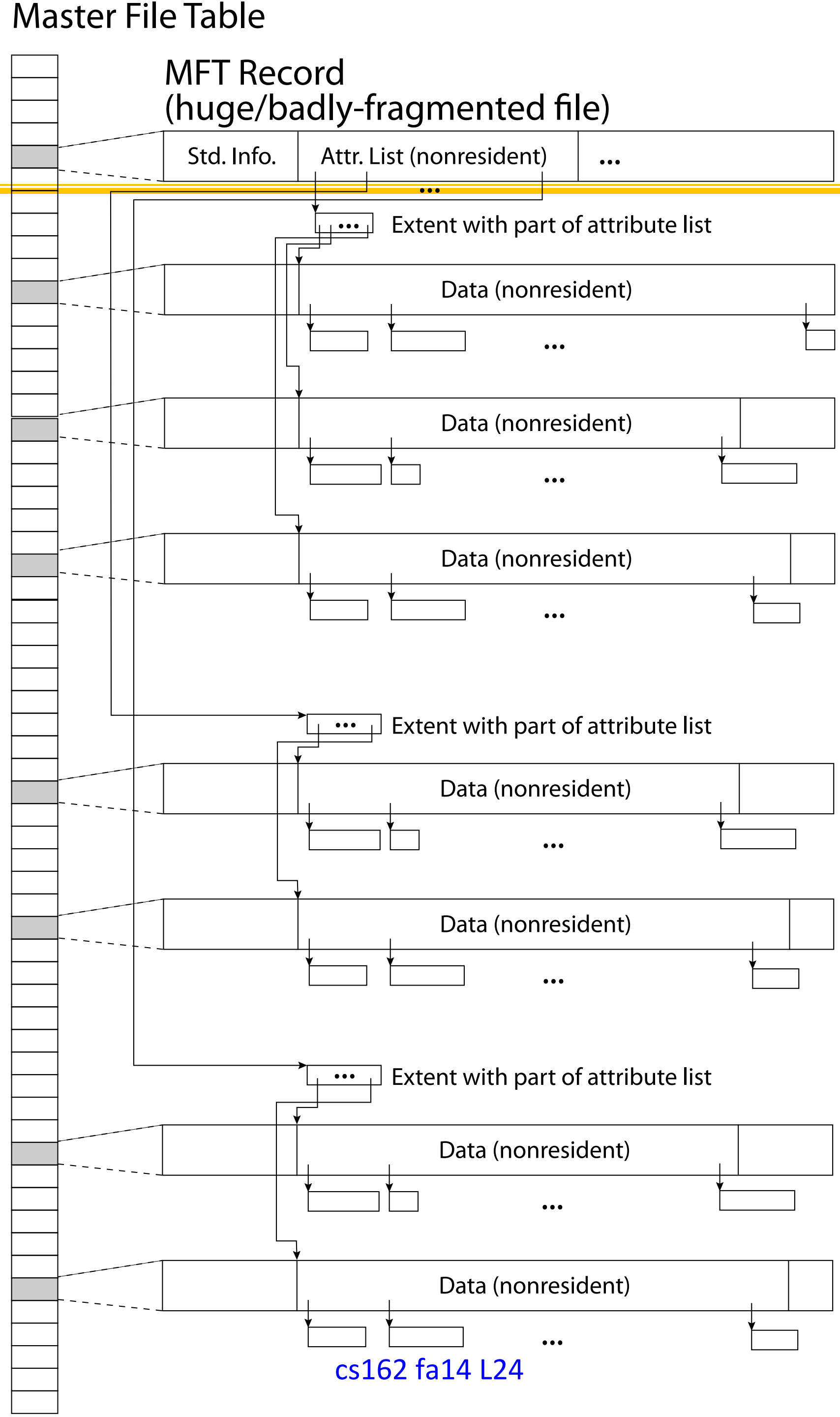


NTFS Medium File



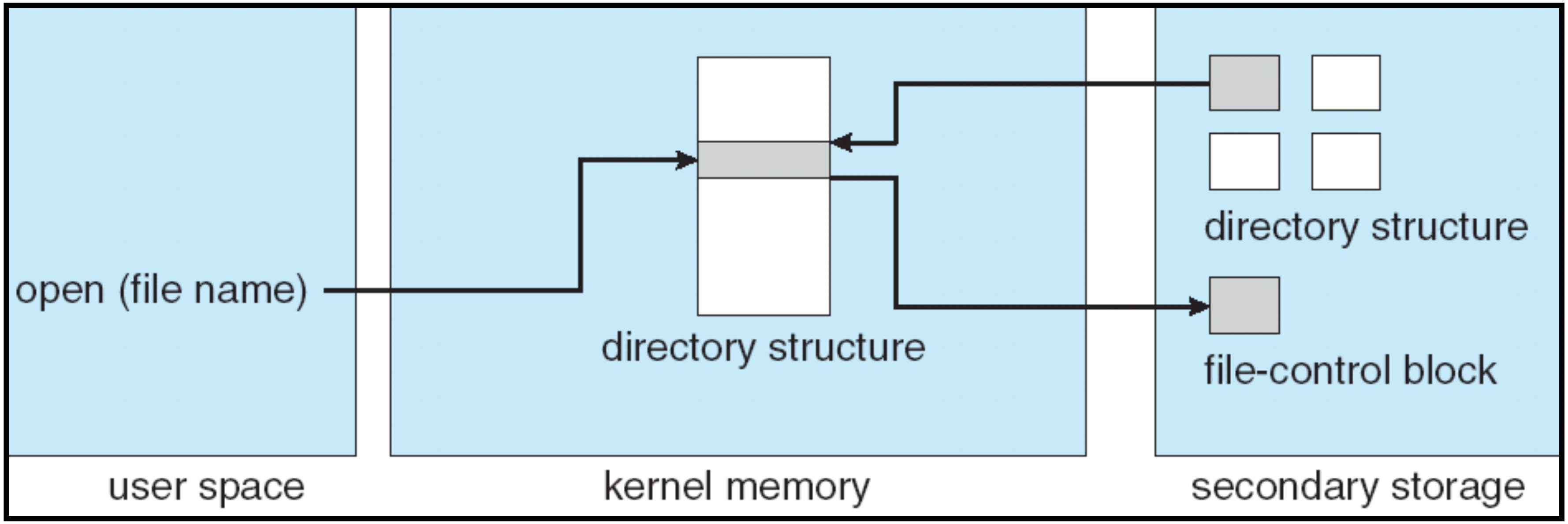
NTFS Multiple Indirect Blocks





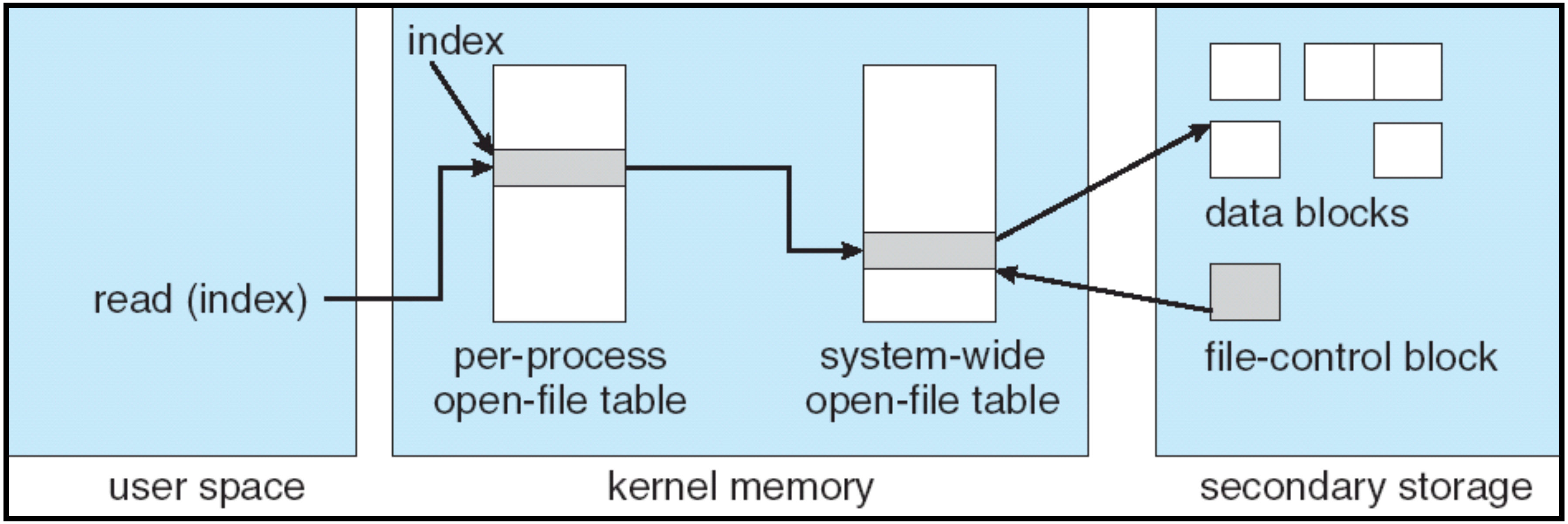
Master File Table
MFT Record
(huge/badly-fragmented file)
Std. Info.
Attr. List (nonresident)
...
Extent with part of attribute list
Data (nonresident)
Data (nonresident)
Data (nonresident)
Extent with part of attribute list
Data (nonresident)
Data (nonresident)
Extent with part of attribute list
Data (nonresident)
Data (nonresident)

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

Quizzie: File Systems

- Q1: True _ False _ A hard-link is a pointer to other file
- Q2: True _ False _ inumber is the id of a block
- Q3: True _ False _ Typically, directories are stored as files
- Q4: True _ False _ Storing file headers on the outermost cylinders minimizes the seek time

Quizzie: File Systems

- Q1: True _ False _**X** A hard-link is a pointer to other file
- Q2: True _ False _**X** inumber is the id of a block
- Q3: True _**X** False _ Typically, directories are stored as files
- Q4: True _ False _**X** Storing file headers on the outermost cylinders minimizes the seek time

File System Summary (1/2)

- File System:
  - Transforms blocks into Files and Directories
  - Optimize for access and usage patterns
  - Maximize sequential access, allow efficient random access
- File (and directory) defined by header, called "inode"
- Multilevel Indexed Scheme
  - Inode contains file info, direct pointers to blocks,
  - indirect blocks, doubly indirect, etc..

File System Summary (2/2)

- 4.2 BSD Multilevel index files
  - Inode contains pointers to actual blocks, indirect blocks, double indirect blocks, etc.
  - Optimizations for sequential access: start new files in open ranges of free blocks, rotational Optimization
- Naming: act of translating from user-visible names to actual system resources
  - Directories used for naming for local file systems