Introduction to Operating Systems

CPSC/ECE 3220 Summer 2018

Lecture Notes

OSPP Chapter 13

(adapted by Mark Smotherman from Tom Anderson’s slides on OSPP web site)

Named Data in a File System

- Directory – typically tree-structured
- Index structure – typically tree-structured
- Free space map – often a bitmap
- Locality heuristics
  - Policy in finding free space (first-fit, etc.)
  - Grouping of directories and files
  - Defragmentation
  - Optimization of writes over reads

Main Points

- File layout
- Directory layout
- Access control

File Organization Design Constraints

- For small files:
  - Small blocks for storage efficiency
  - Files used together should be stored together
- For large files:
  - Contiguous allocation for sequential access
  - Efficient lookup for random access
- May not know at file creation
  - Whether file will become small or large

File System Design

- Data structures
  - Directories: file name -> file metadata
    - Store directories as files
  - File metadata: how to find file data blocks
  - Free map: list of free disk blocks
- How do we organize these data structures?
  - Device has non-uniform performance

Design Challenges

- Index structure
  - How do we locate the blocks of a file?
- Index granularity
  - What block size do we use?
- Free space
  - How do we find unused blocks on disk?
- Locality
  - How do we preserve spatial locality?
- Reliability
  - What if machine crashes in middle of a file system op?

File Organization

- Want sequential data placement that provides efficient sequential access
- Also want placement that provides efficient random access
- But…
  - Reject contiguous storage of disk blocks – why?
  - Reject linked-list storage with links located in the disk blocks – why?

File Organization (2)

- Typically tree-structured indexing of data
  - Want to limit overheads to be efficient for small files
  - Provide scalability for large files
  - Provide a place for per-file metadata

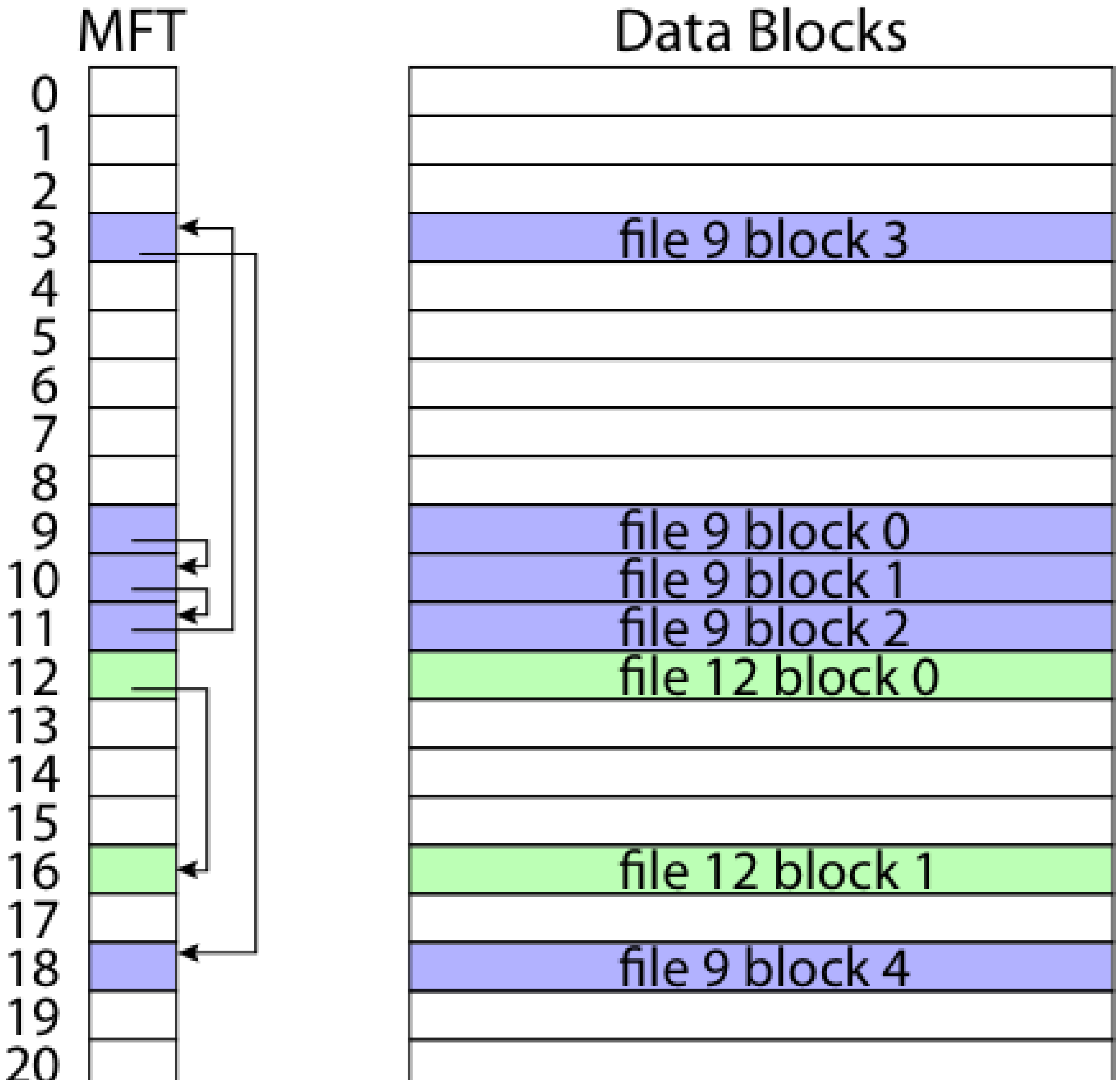
File Organization Design Options

	FAT	FFS	NTFS	ZFS
Index structure	linked list	tree (fixed, asymmetric)	tree (dynamic)	tree (COW, dynamic)
Granularity	block	block	extent	block
Free space allocation	FAT array	bitmap (fixed location)	bitmap (file)	space map (log-structured)
Locality	defrag	block groups, first fit, reserve space	extents, best fit, defrag	write-anywhere block groups

Microsoft File Allocation Table (FAT)

- Linked list index structure
  - Links are collected together in a table, not placed out in data blocks
  - Simple, easy to implement
  - Still widely used (e.g., thumb drives)
- File table
  - Linear map of all blocks on disk
  - Each file a linked list of blocks

FAT Organization



FAT Evaluation

- Pros:
  - Easy to find free block
  - Easy to append to a file
  - Easy to delete a file
- Cons:
  - Small file access is slow
  - Random access is very slow
  - Fragmentation
    - File blocks for a given file may be scattered
    - Files in the same directory may be scattered
    - Problem becomes worse as disk fills

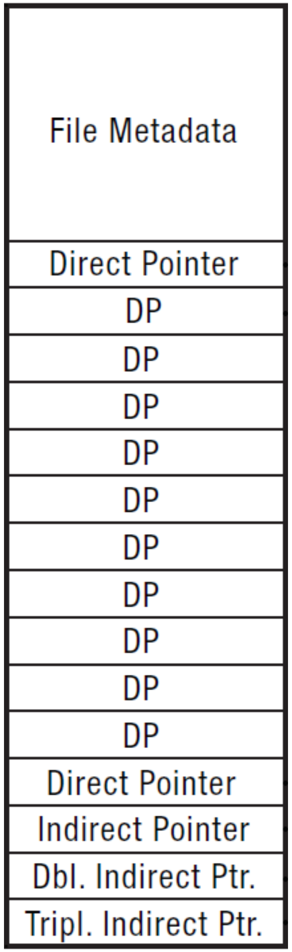
Berkeley UNIX FFS (Fast File System)

- inode table
  - Analogous to FAT table
- inode
  - Metadata
    - File owner, access permissions, access times, …
  - Set of 12 data pointers
  - With 4KB blocks => max size of 48KB files

Additional FFS inode Block Pointers

- Indirect block pointer (13^{th} pointer in inode)
  - pointer to disk block of data pointers
  - 4KB block size => 1K data blocks => 4MB
- Doubly indirect block pointer (14^{th} in inode)
  - Doubly indirect block => 1K indirect blocks
  - 4GB (+ 4MB + 48KB)
- Triply indirect block pointer (15^{th} in inode)
  - Triply indirect block => 1K doubly indirect blocks
  - 4TB (+ 4GB + 4MB + 48KB)

FFS inode

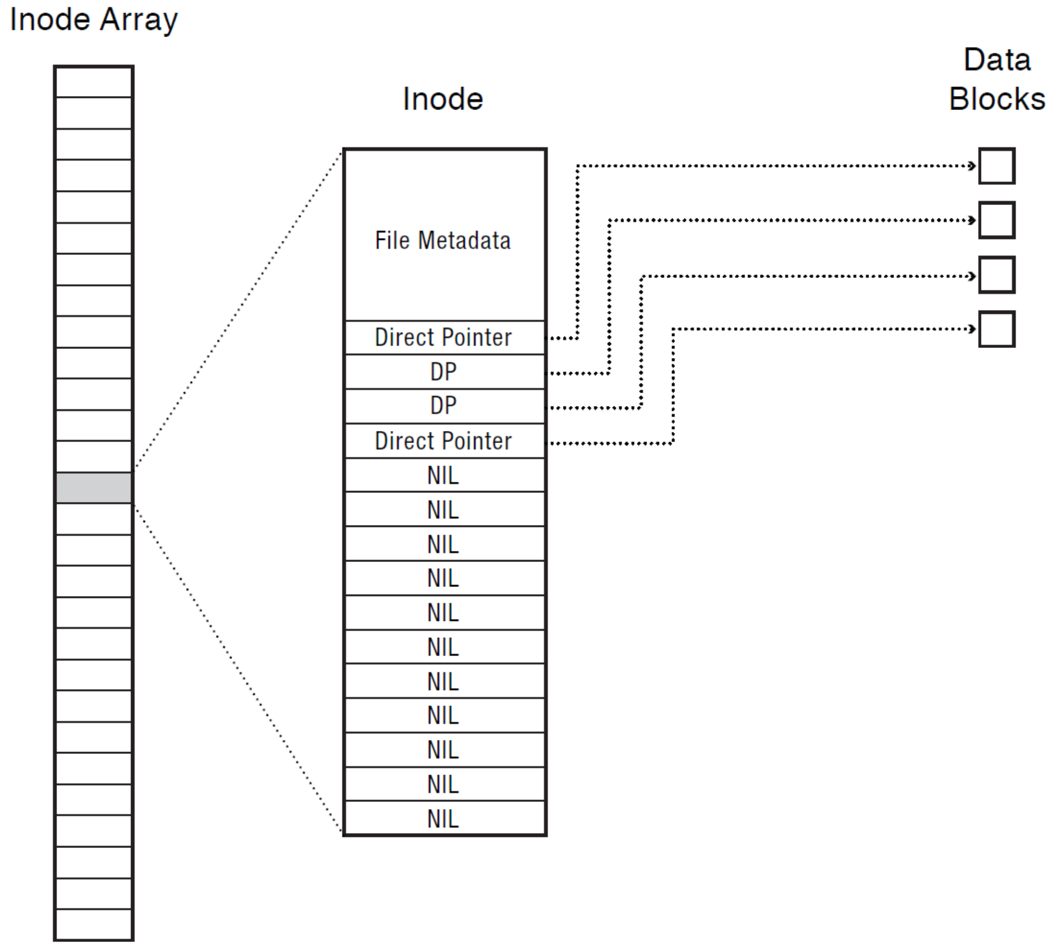


File Metadata
Direct Pointer
DP
DP
DP
DP
DP
DP
DP
DP
DP
DP
Direct Pointer
Indirect Pointer
Dbl. Indirect Ptr.
Tripl. Indirect Ptr.

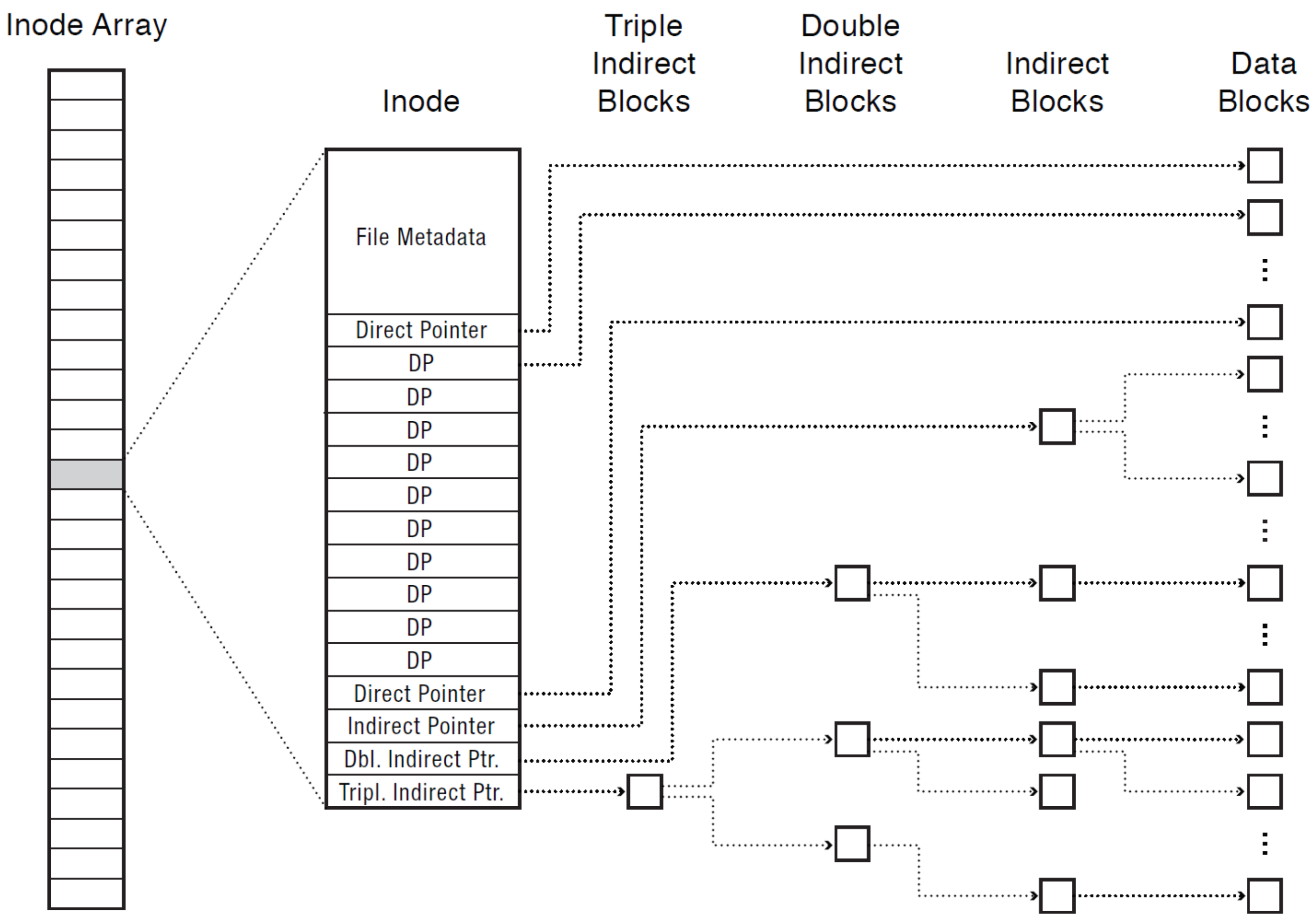
FFS Asymmetric Tree

- Small files: shallow tree
  - Efficient storage for small files
- Large files: deep tree
  - Efficient lookup for random access in large files
- Sparse files: only fill pointers if needed

FFS Small File

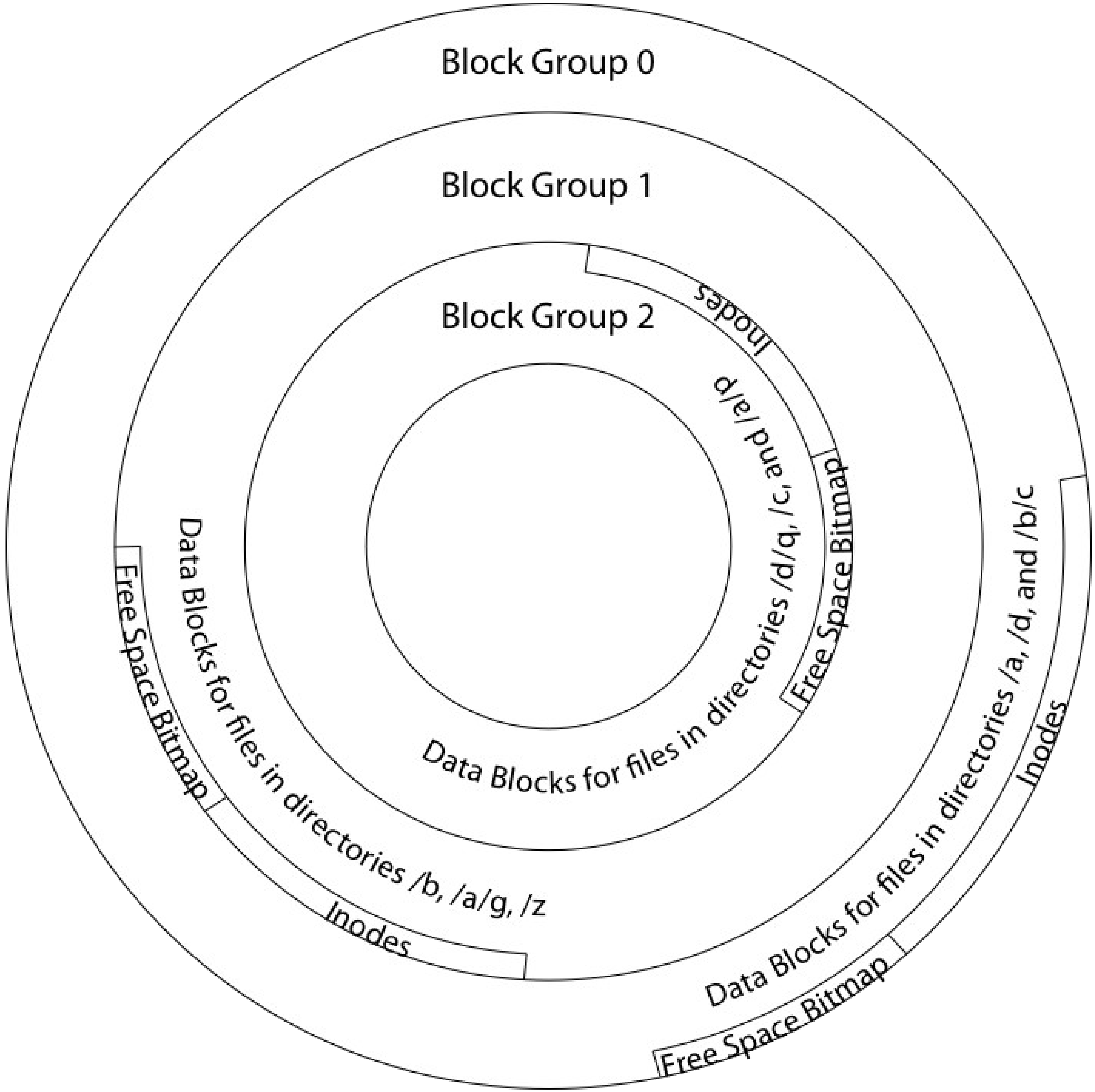


FFS Large File



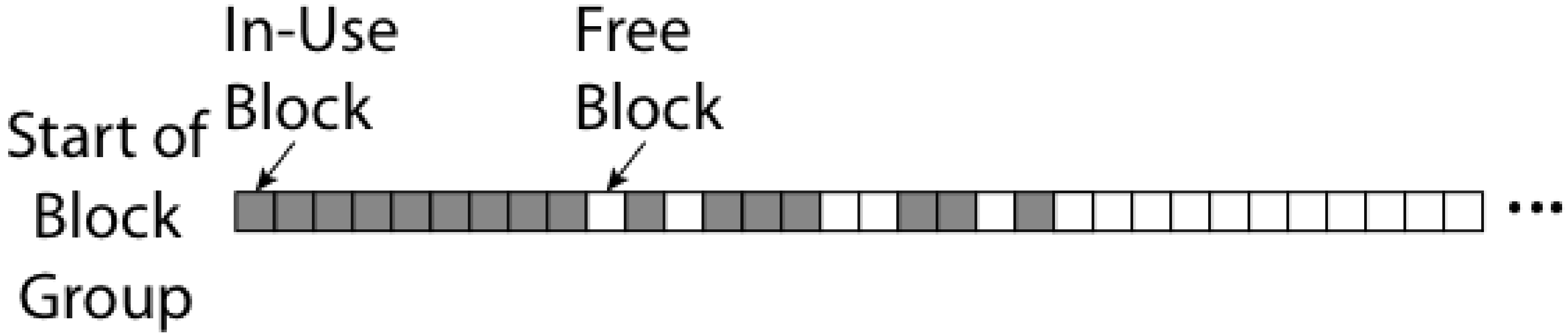
FFS Locality

- Block group allocation
  - Block group is a set of nearby cylinders
  - Files in same directory located in same group
  - Subdirectories located in different block groups
- inode table spread throughout disk
  - inodes, bitmap near file blocks
- First fit allocation
  - Small files fragmented, large files contiguous

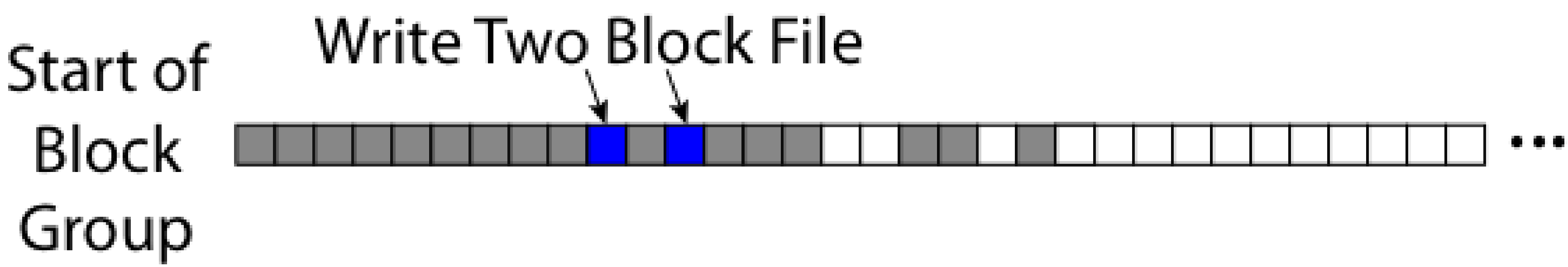


Block Group 0
Block Group 1
Block Group 2
Inodes
Free Space Bitmap
Data Blocks for files in directories /d/q, /c, and /a/p
Free Space Bitmap
Inodes
Data Blocks for files in directories /b, /a/g, /z
Inodes
Free Space Bitmap
Data Blocks for files in directories /a, /d, and /b/c

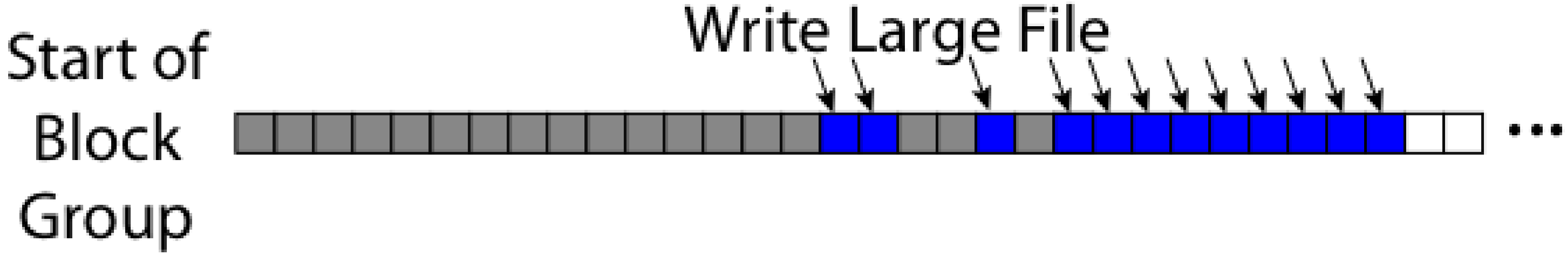
FFS First Fit Block Allocation



FFS First Fit Block Allocation



FFS First Fit Block Allocation



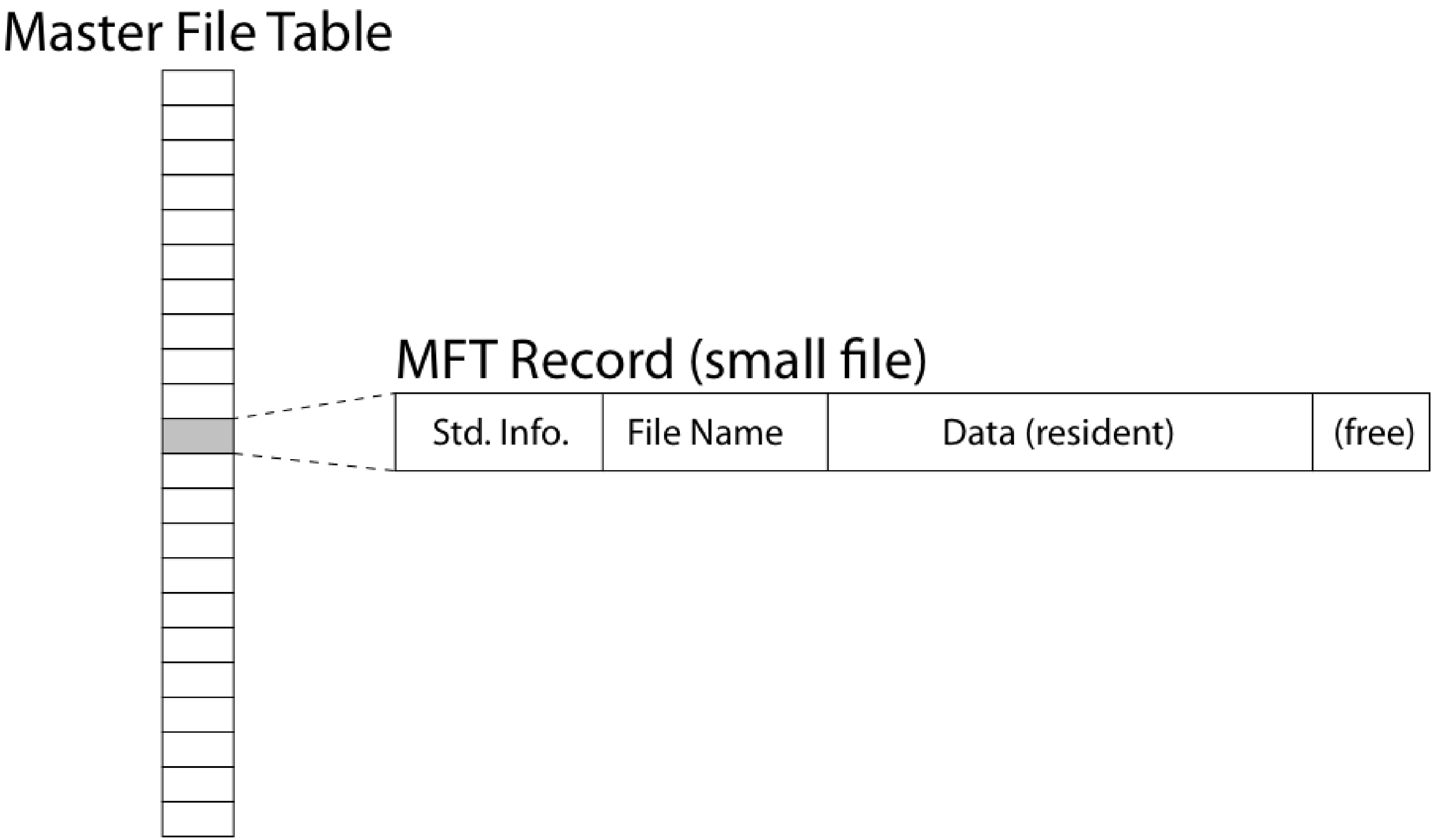
FFS Evaluation

- 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

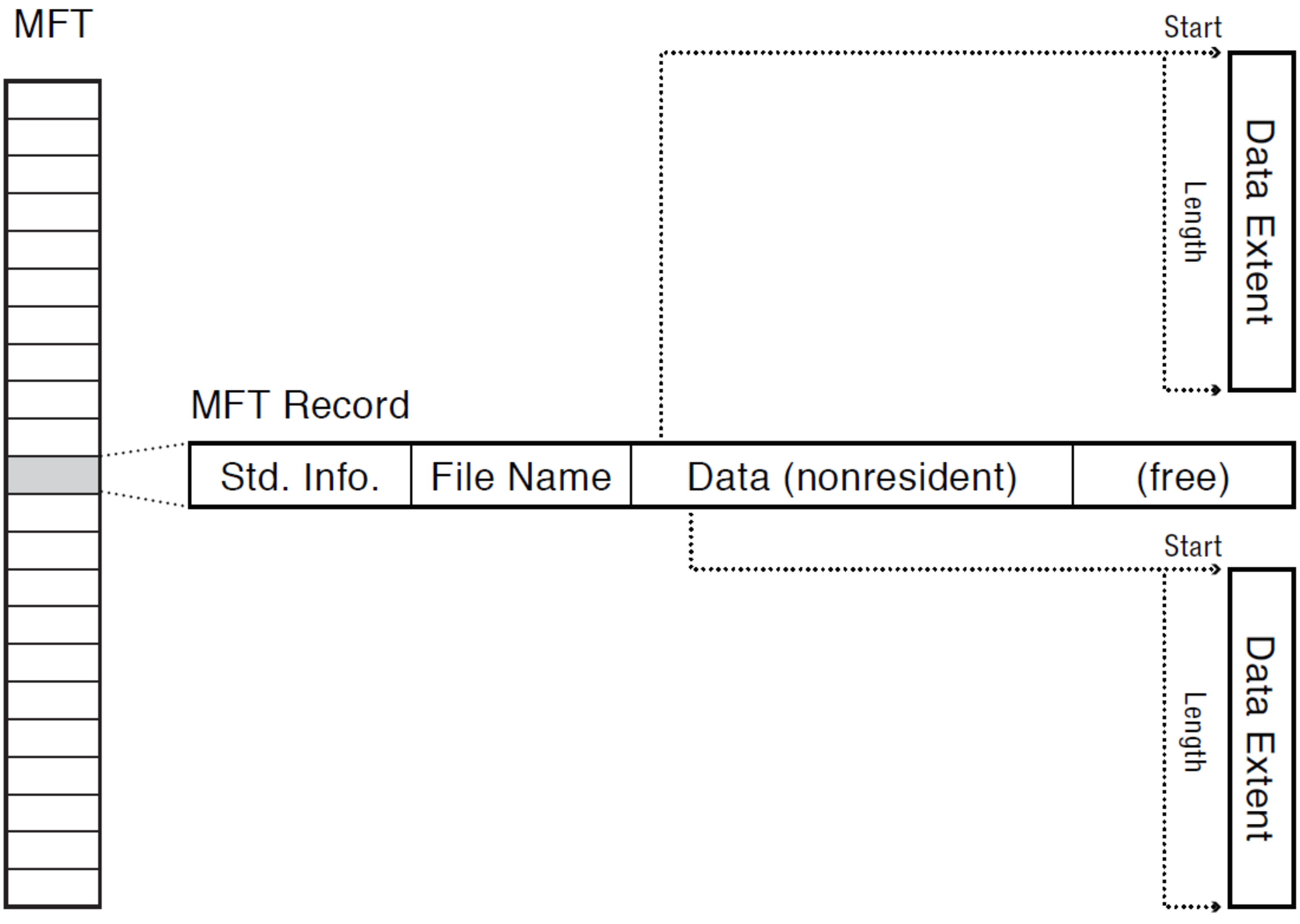
NTFS

- Master File Table
  - Flexible 1KB storage for metadata and data
- Extents
  - Block pointers cover runs of blocks
  - Similar approach in linux (ext4)
  - File create can provide hint as to size of file
- Journalling for reliability
  - Next chapter

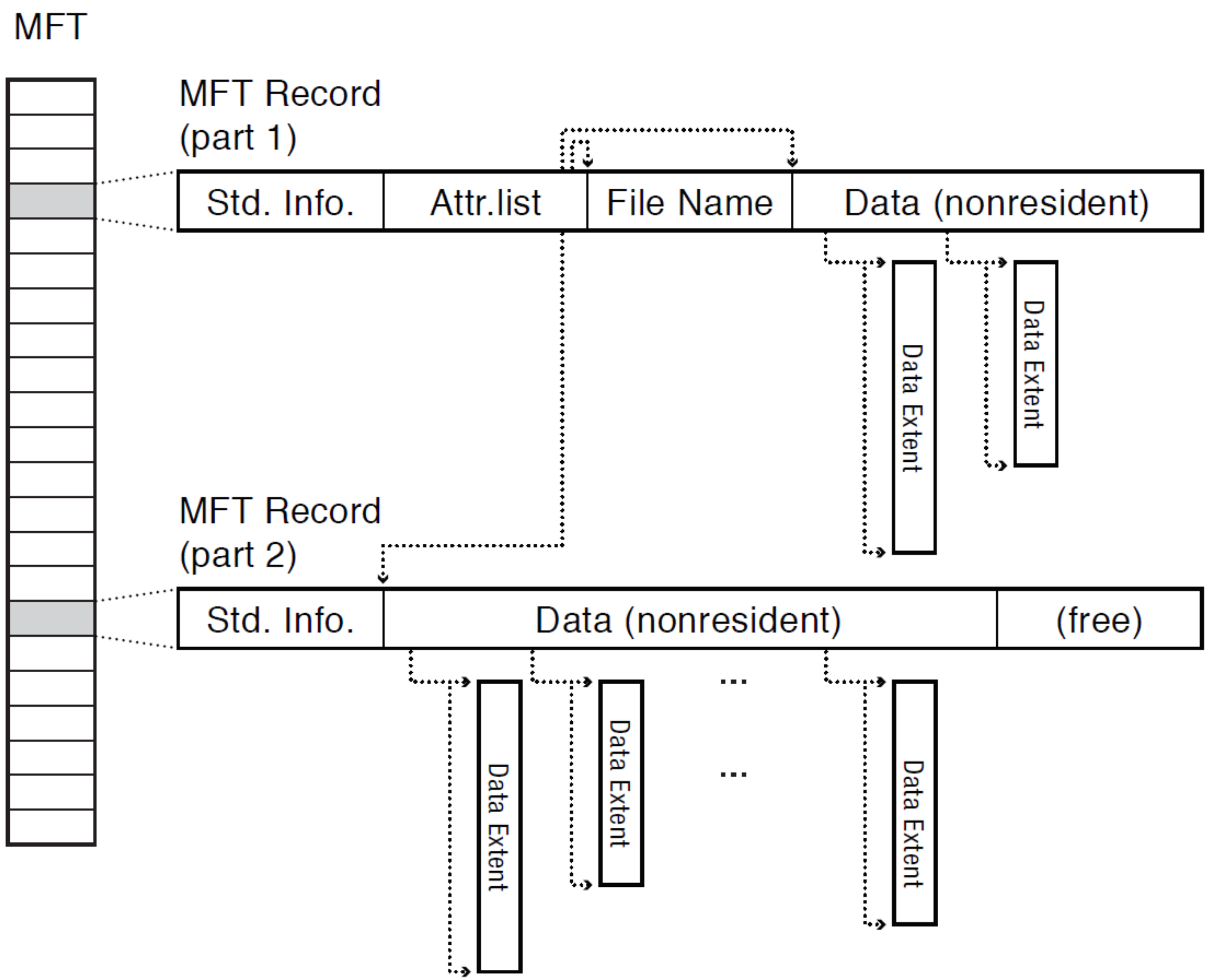
NTFS Small File

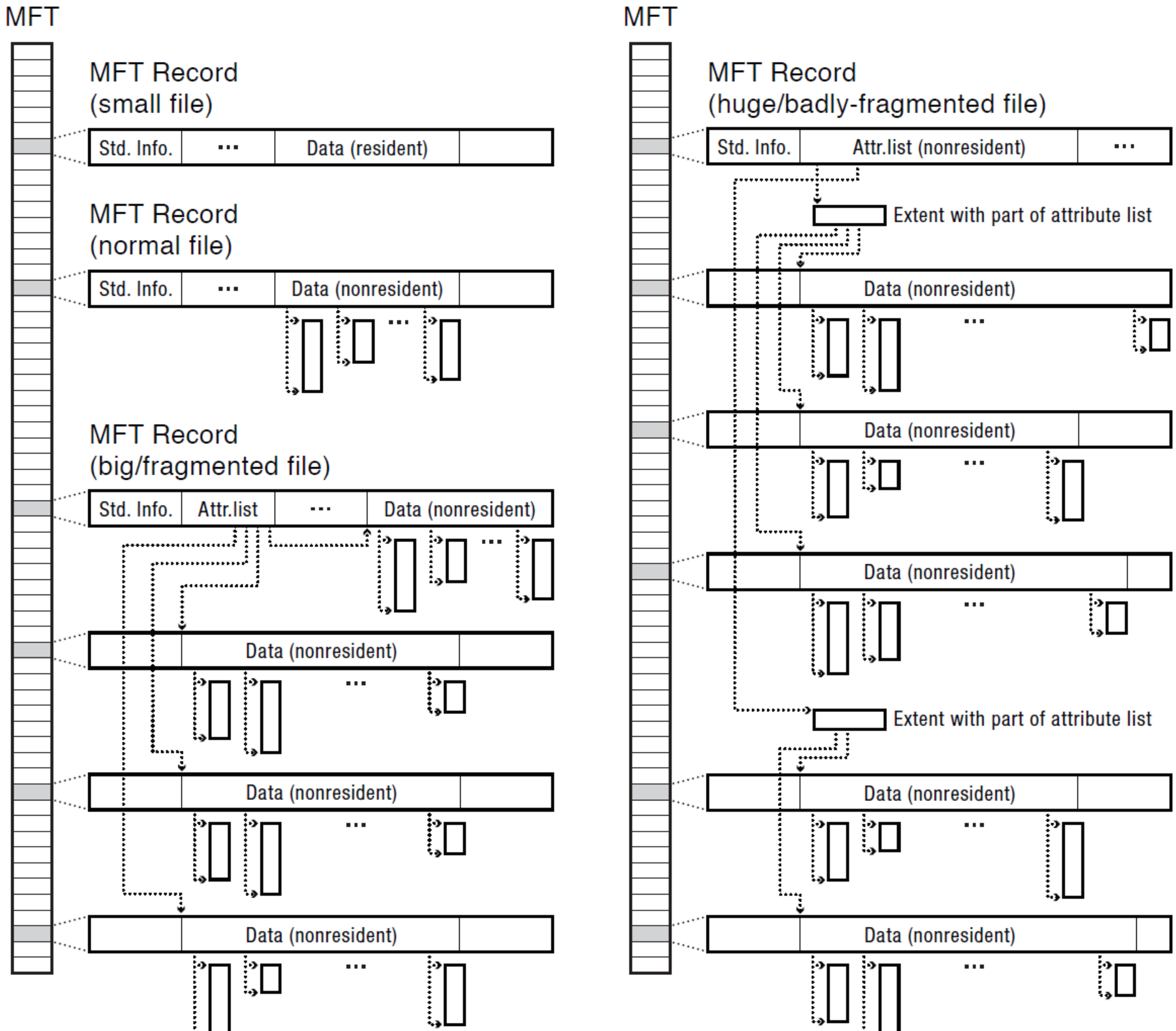


NTFS Medium-Sized File

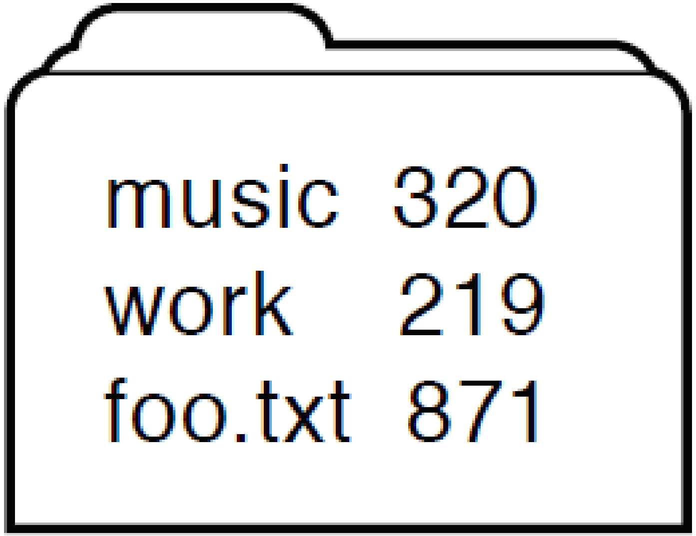


NTFS Indirect Block

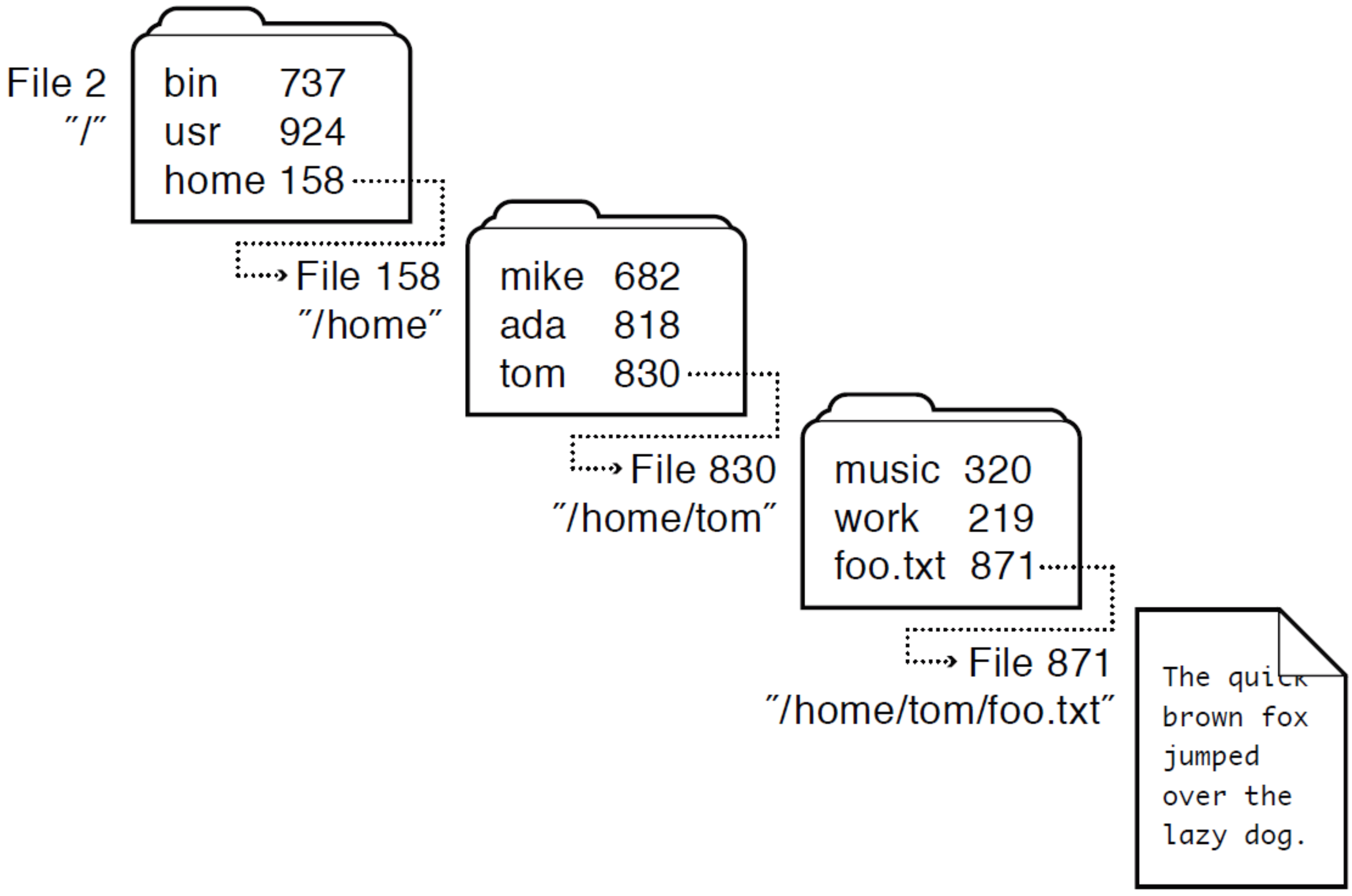



MFT
MFT Record
(small file)
Std. Info.
...
Data (resident)
MFT Record
(normal file)
Std. Info.
...
Data (nonresident)
MFT Record
(big/fragmented file)
Std. Info.
Attr.list
...
Data (nonresident)
Data (nonresident)
Data (nonresident)
Data (nonresident)
MFT
MFT Record
(huge/badly-fragmented file)
Std. 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)

Directories Are Files

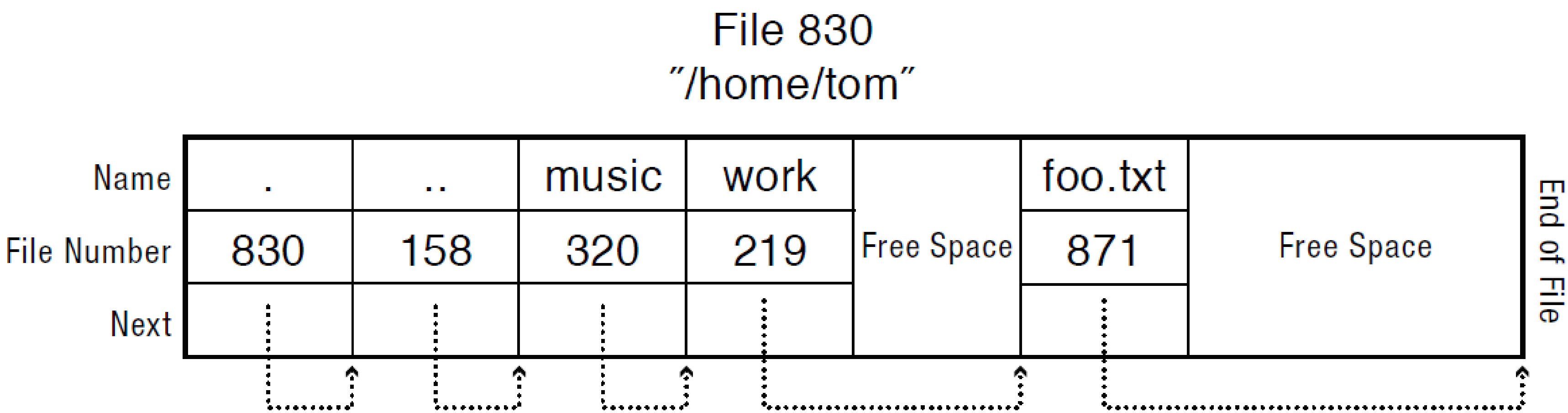


Recursive Filename Lookup

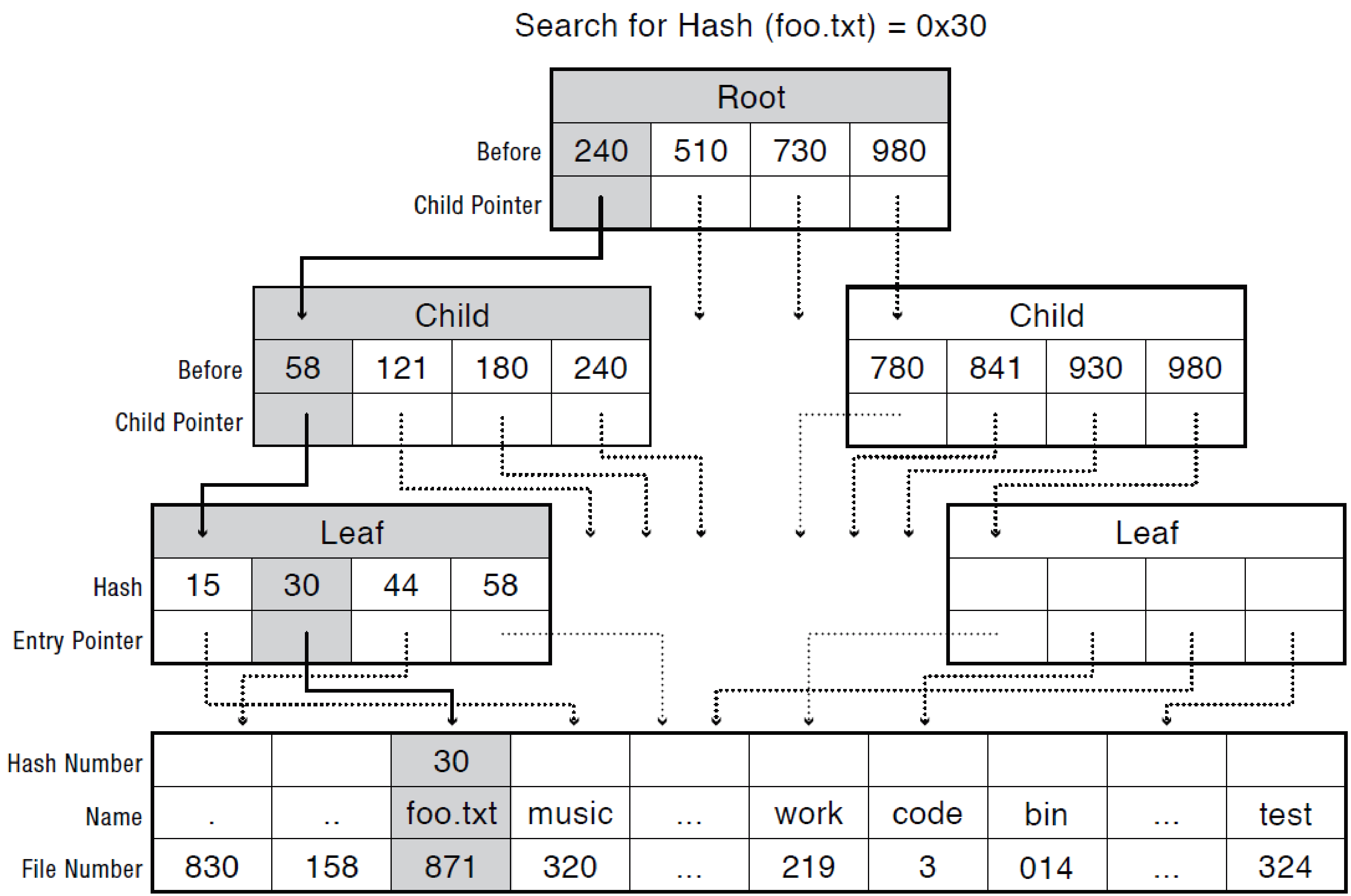


Directory Layout

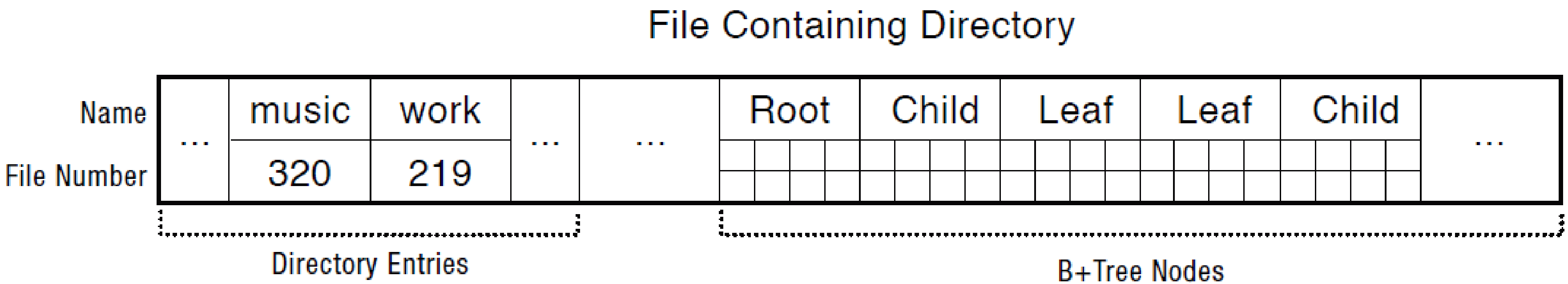
irectory stored as a file
near search to find filename (small directorie



Large Directories: B Trees



Large Directories: Layout



Access Control

- Access control matrix
  - Row for each user (or application)
  - Column for every protected resource
  - Sparse
- Row-oriented approach
  - Capability list
  - Keep with user
- Column-oriented approach
  - Access control list (ACL)
  - Keep with resource

Capability List

- Example: process keeps a table of open files with allowed type of access
- Example: smartphone app keeps a list of resources with allowed type of access

Access Control List

- For every protected resource, list of who is permitted to do what
- Example: Windows ACL
  - List of access control entries (ACEs)
  - Each ACE contains a 32-bit access rights mask and a security identifier (SID)
- Example: compressed ACL for UNIX files
  - Owner, group, world: read, write, execute
  - Setuid: program will run using the permissions of user who installed it
  - File type to indicate if file is a directory