File Systems

Main Points

- File layout
- Directory layout

File System 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?
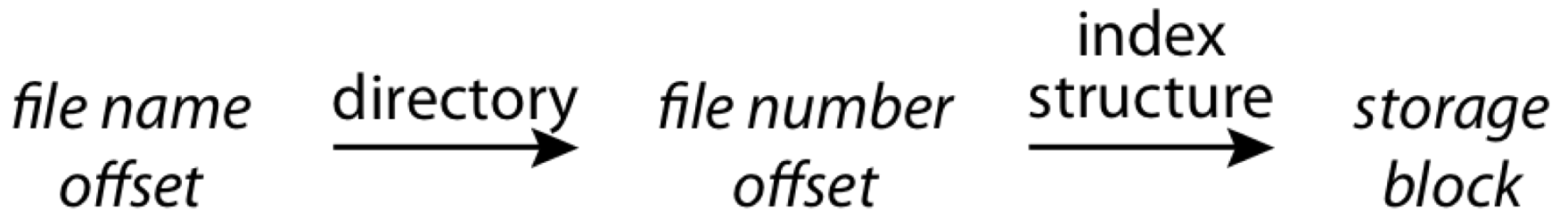 - Device has non-uniform performance

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 System Design Options

	FAT	FFS	NTFS
Index structure	Linked list	Tree (fixed, assym)	Tree (dynamic)
granularity	block	block	extent
free space allocation	FAT array	Bitmap (fixed location)	Bitmap (file)
Locality	defragmentation	Block groups + reserve space	Extents Best fit defrag

Named Data in a File System

file name offset $\xrightarrow{\text{directory}}$ *file number offset* $\xrightarrow[\text{structure}]{\text{index}}$ *storage block*
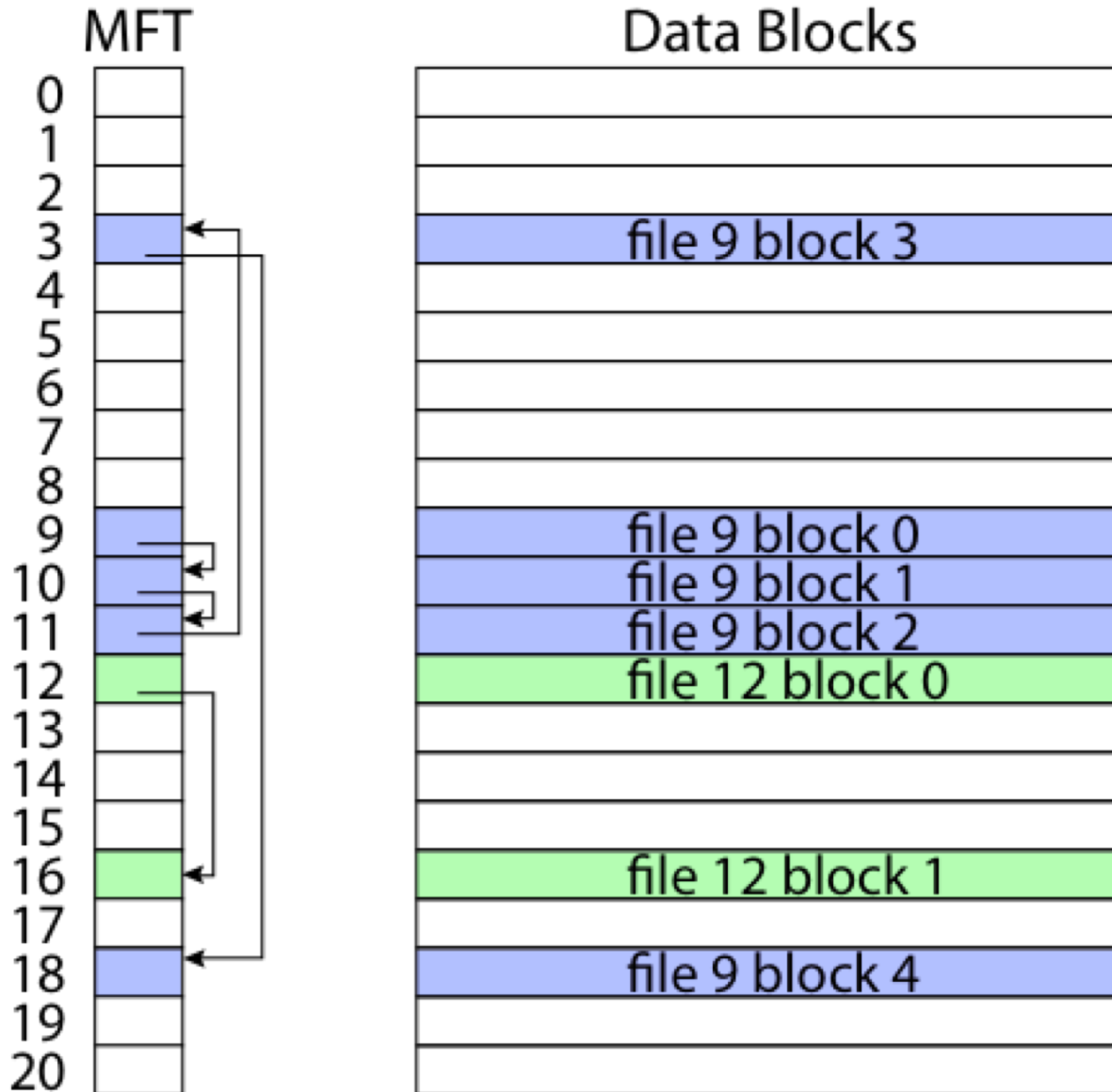
Microsoft File Allocation Table (FAT)

- Linked list index structure
  - 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

MFT

Data Blocks

0	
1	
2	
3	file 9 block 3
4	
5	
6	
7	
8	
9	file 9 block 0
10	file 9 block 1
11	file 9 block 2
12	file 12 block 0
13	
14	
15	
16	file 12 block 1
17	
18	file 9 block 4
19	
20	

FAT

- 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 is used

Berkeley UNIX FFS (Fast File System)

- inode table
  - (very roughly) 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

FFS inode

- Metadata
  - File owner, access permissions, access times, …
- Set of 12 data pointers
  - With 4KB blocks => max size of 48KB files
- Indirect block pointer
  - pointer to disk block of data pointers
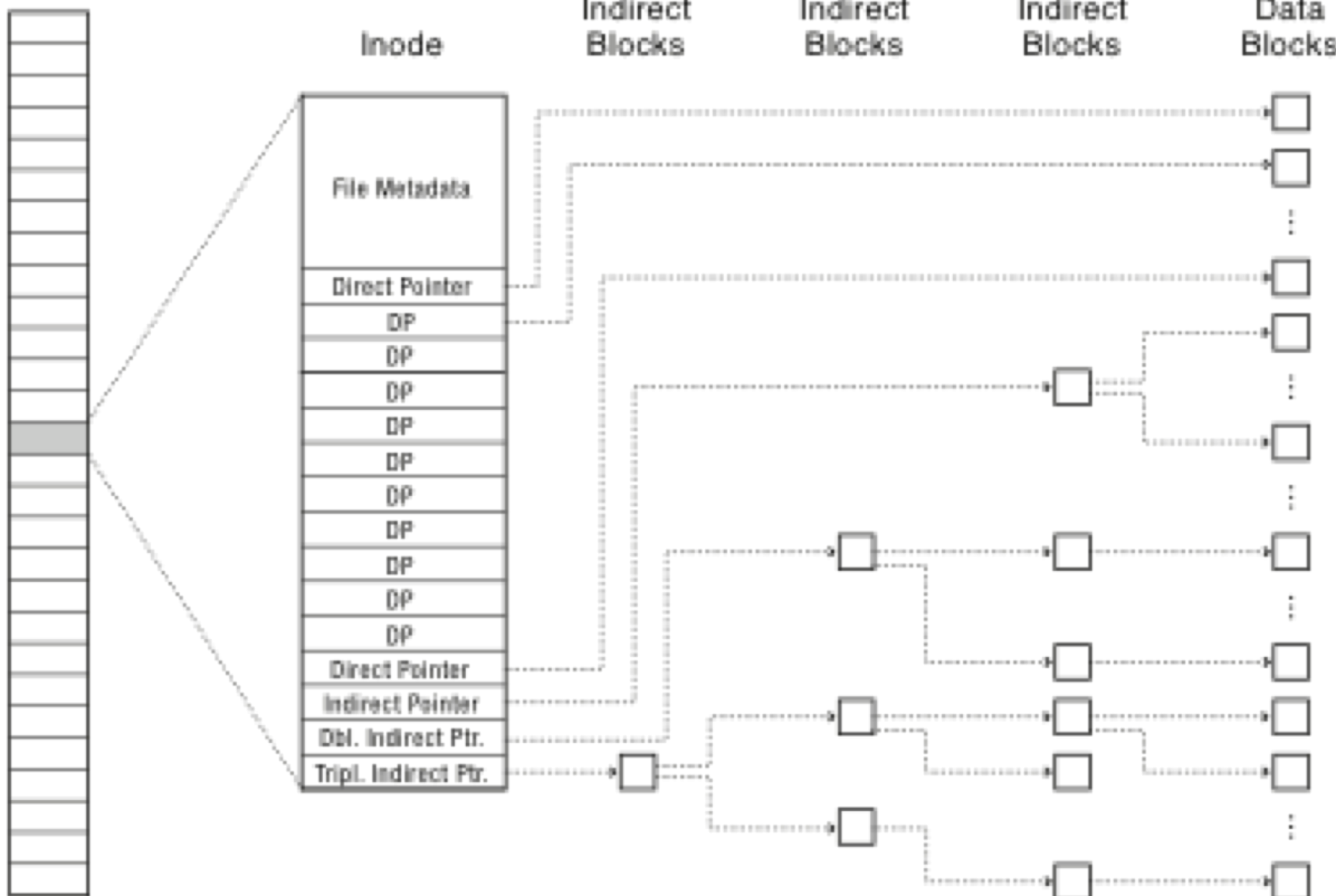- Indirect block: 1K data blocks => 4MB (+48KB)

FFS inode

- Metadata
  - File owner, access permissions, access times, …
- Set of 12 data pointers
  - With 4KB blocks => max size of 48KB
- Indirect block pointer
  - pointer to disk block of data pointers
  - 4KB block size => 1K data blocks => 4MB
- Doubly indirect block pointer
  - Doubly indirect block => 1K indirect blocks
  - 4GB (+ 4MB + 48KB)

FFS inode

- Metadata
  - File owner, access permissions, access times, …
- Set of 12 data pointers
  - With 4KB blocks => max size of 48KB
- Indirect block pointer
  - pointer to disk block of data pointers
  - 4KB block size => 1K data blocks => 4MB
- Doubly indirect block pointer
  - Doubly indirect block => 1K indirect blocks
  - 4GB (+ 4MB + 48KB)
- Triply indirect block pointer
  - Triply indirect block => 1K doubly indirect blocks
  - 4TB (+ 4GB + 4MB + 48KB)

Inode Array

Inode

Triple Indirect Blocks

Double Indirect Blocks

Indirect Blocks

Data Blocks

File Metadata

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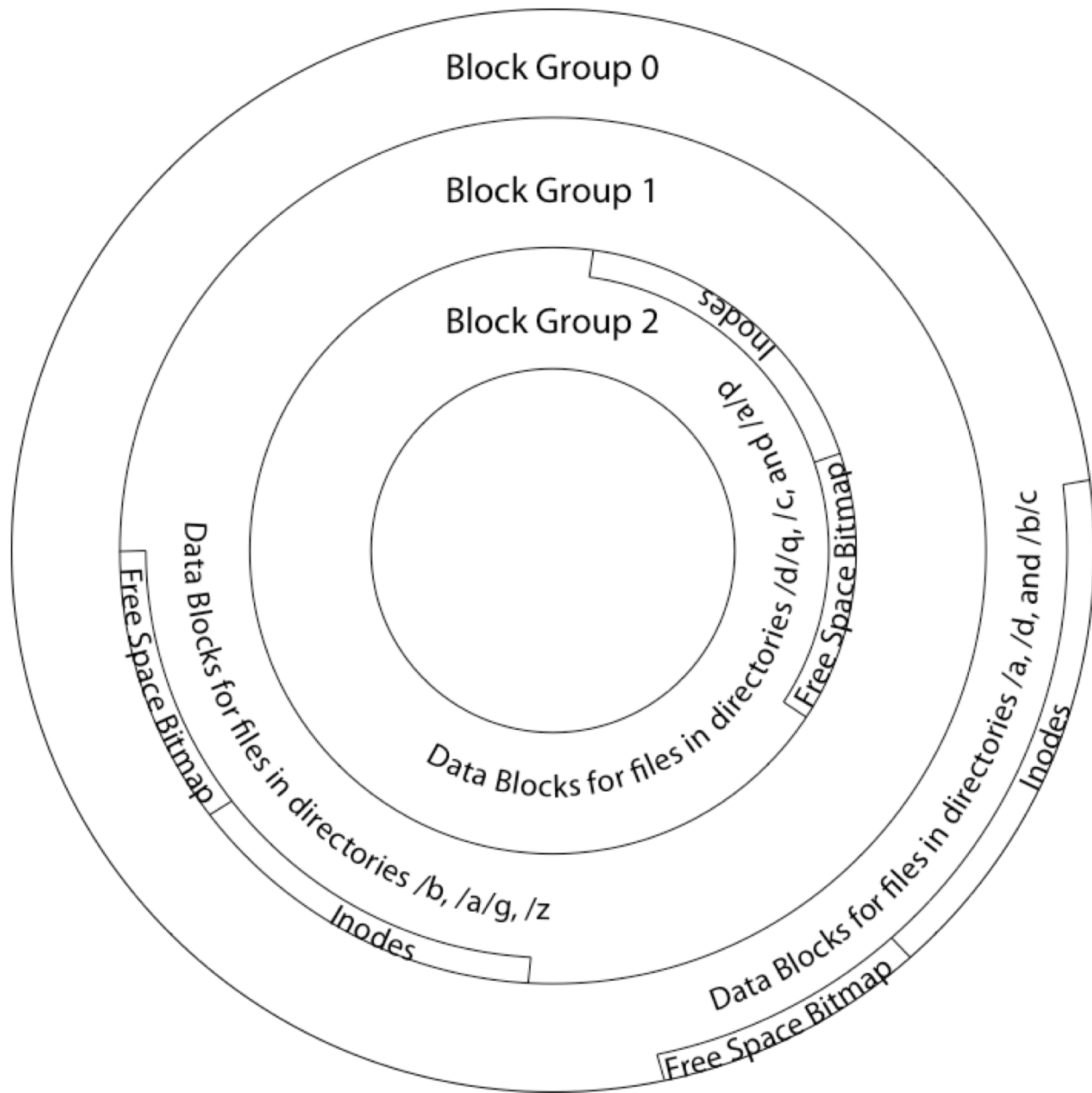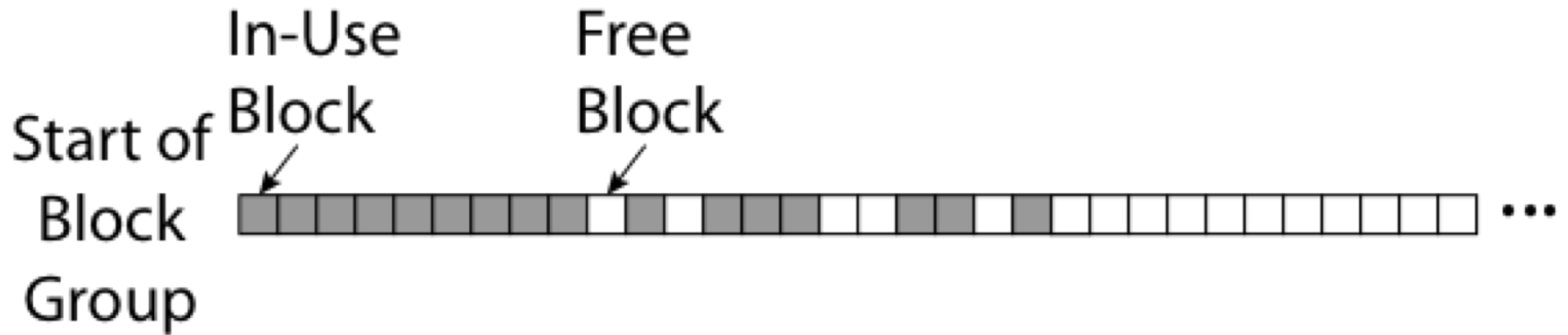Tripl. Indirect Ptr.

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
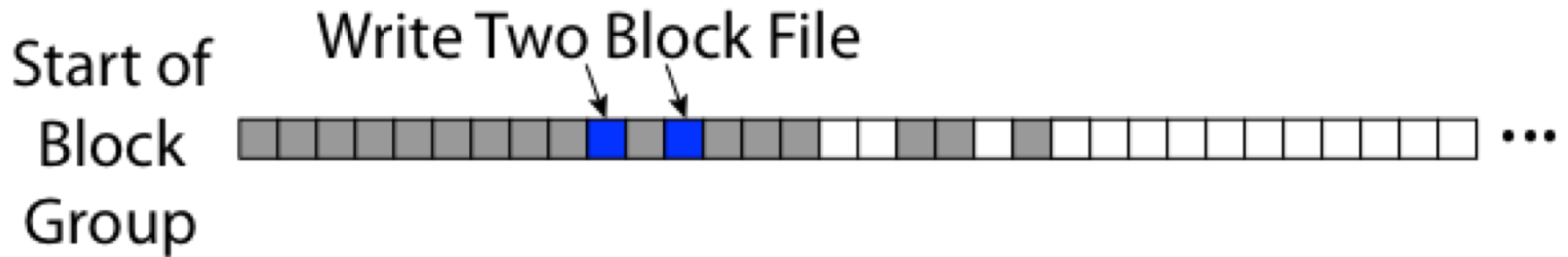
FFS Locality

- 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

/b/a/ and ,c/ ,q/d/ directories in files for Blocks Data

Free Space Bitmap

Data Blocks for files in directories /a, /d, and /b/c

Inodes

Free Space Bitmap

Data Blocks for files in directories /b, /a/g, /z

Inodes

FFS First Fit Block Allocation

FFS First Fit Block Allocation

FFS First Fit Block Allocation
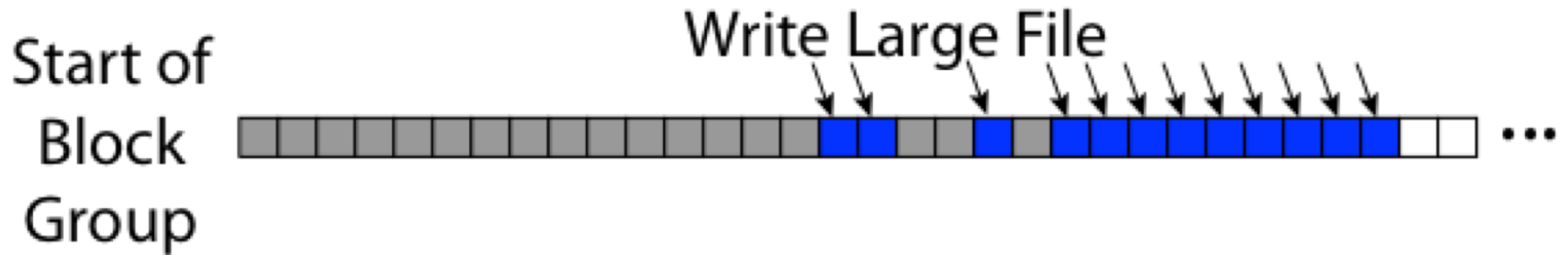
FFS

- 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
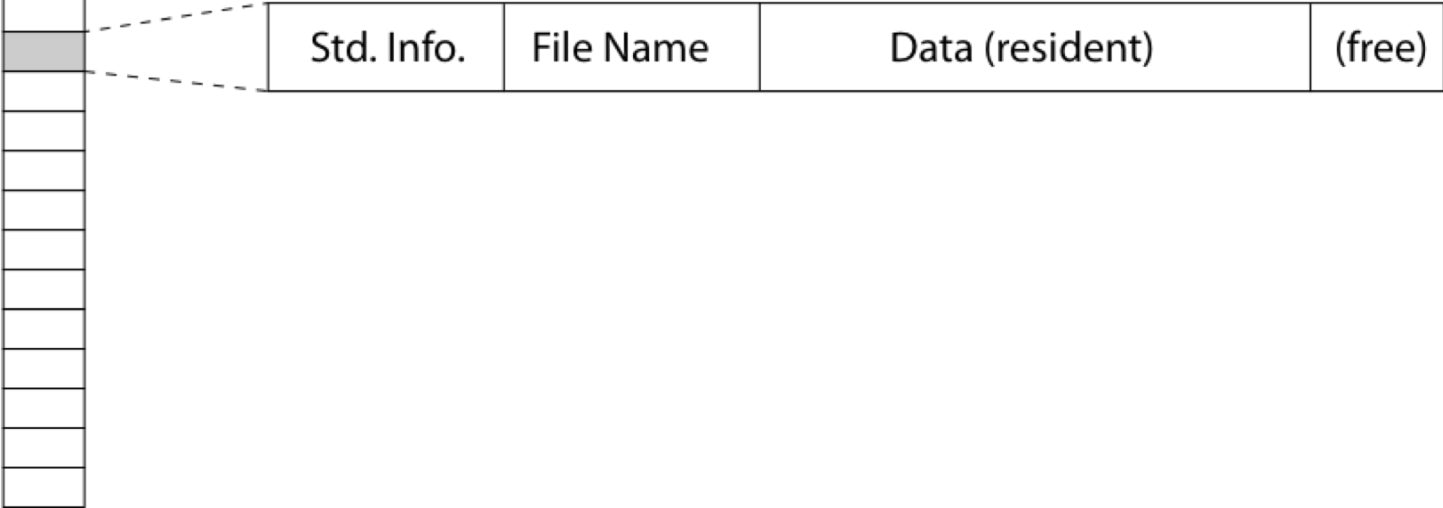
NTFS

- 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
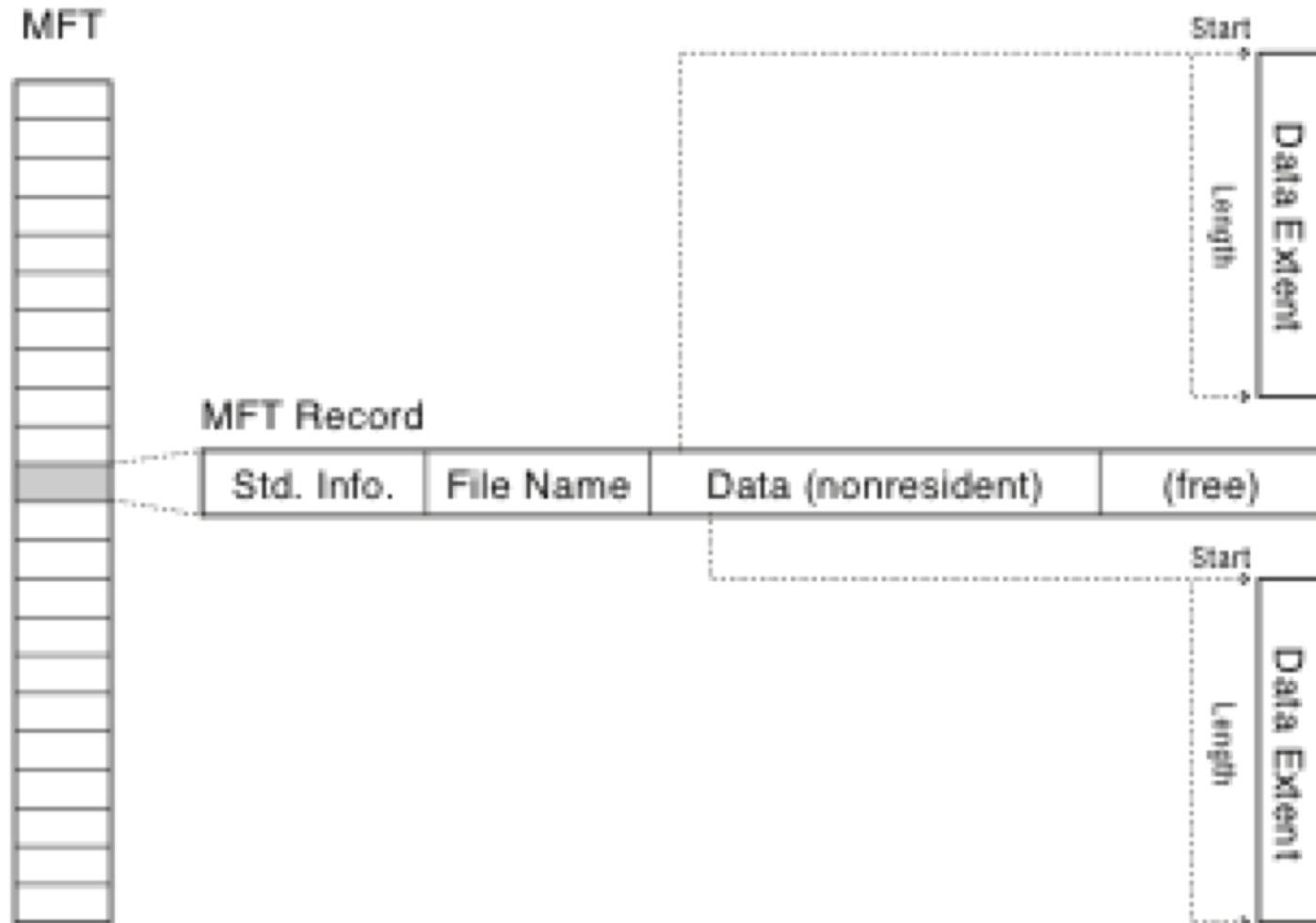- Journalling for reliability
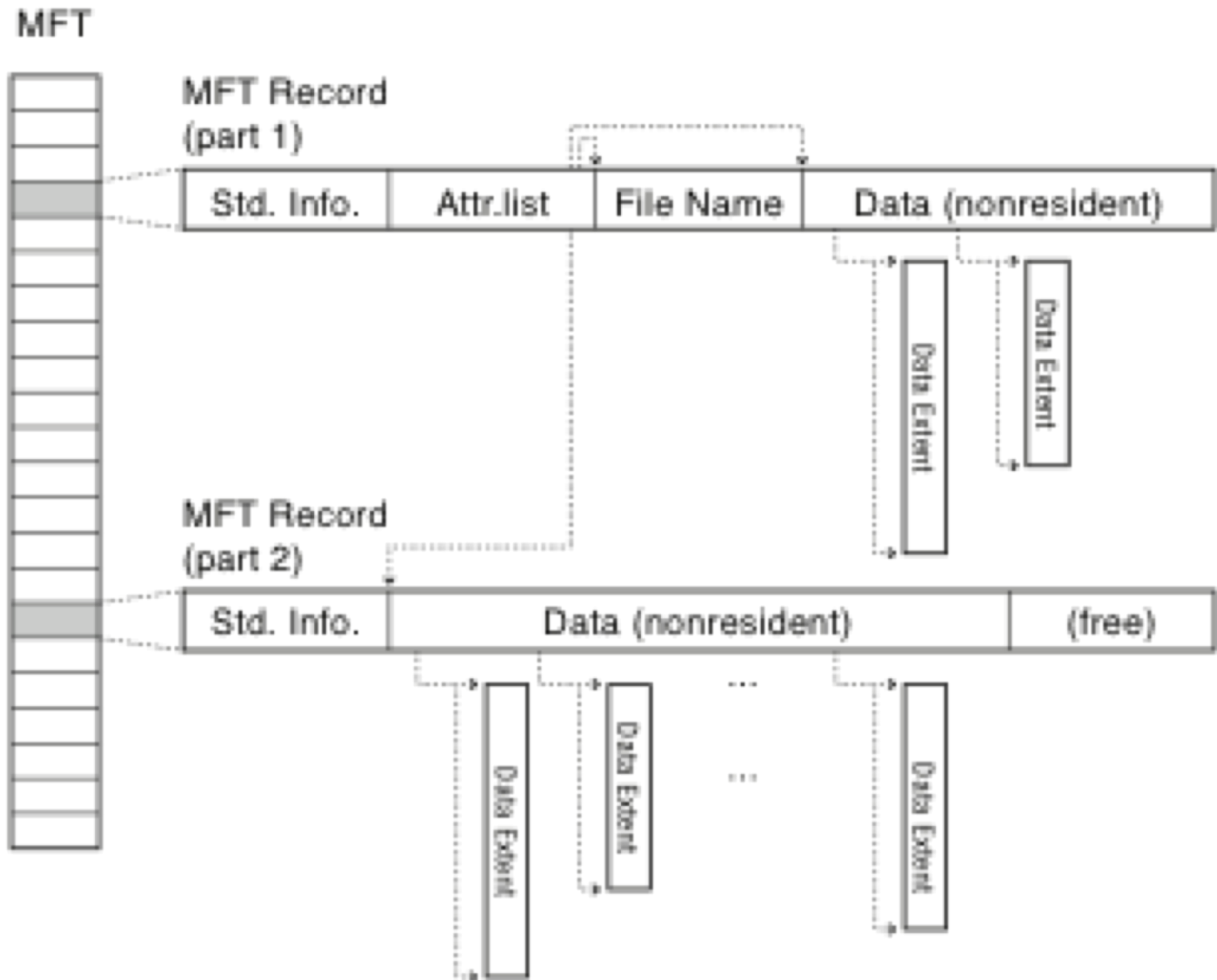 - Next chapter

NTFS Small File
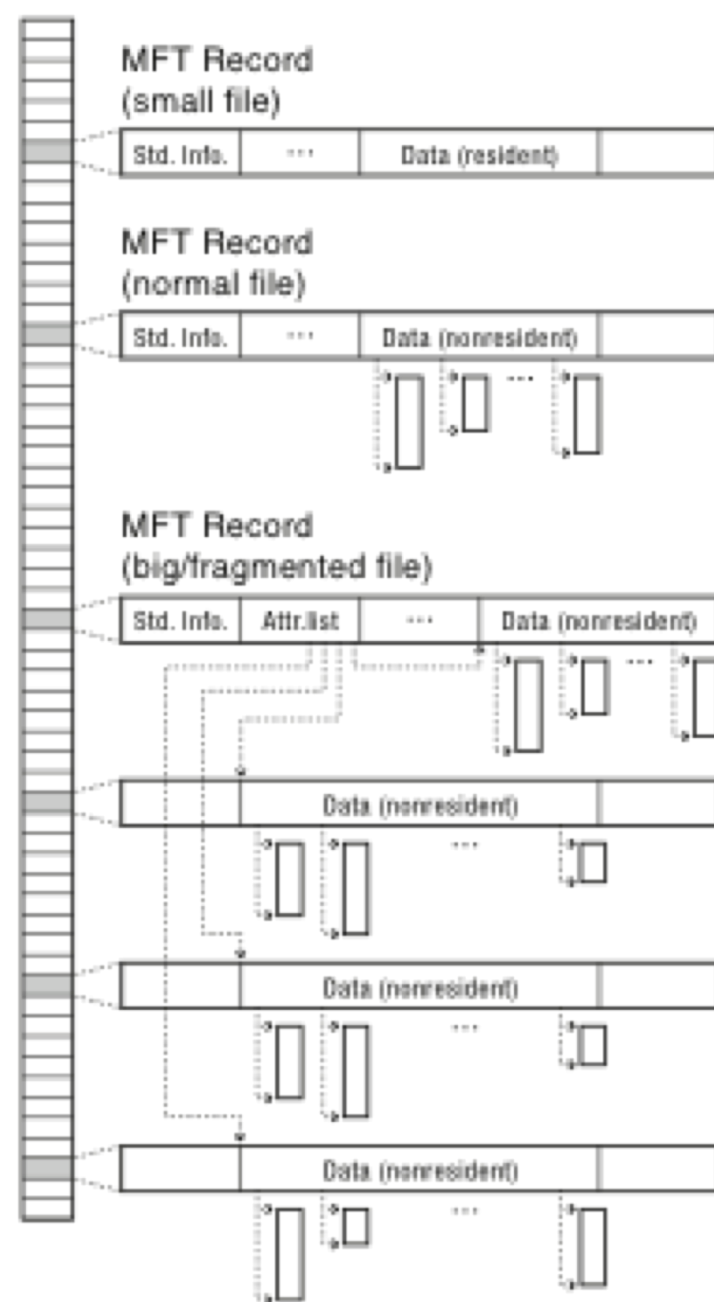
Master File Table

MFT Record (small file)

Std. Info.	File Name	Data (resident)	(free)

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)

Named Data in a File System

file name
offset
$\xrightarrow{\text{directory}}$
file number
offset
$\xrightarrow[\text{structure}]{\text{index}}$
storage
block
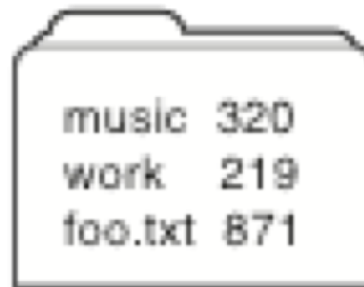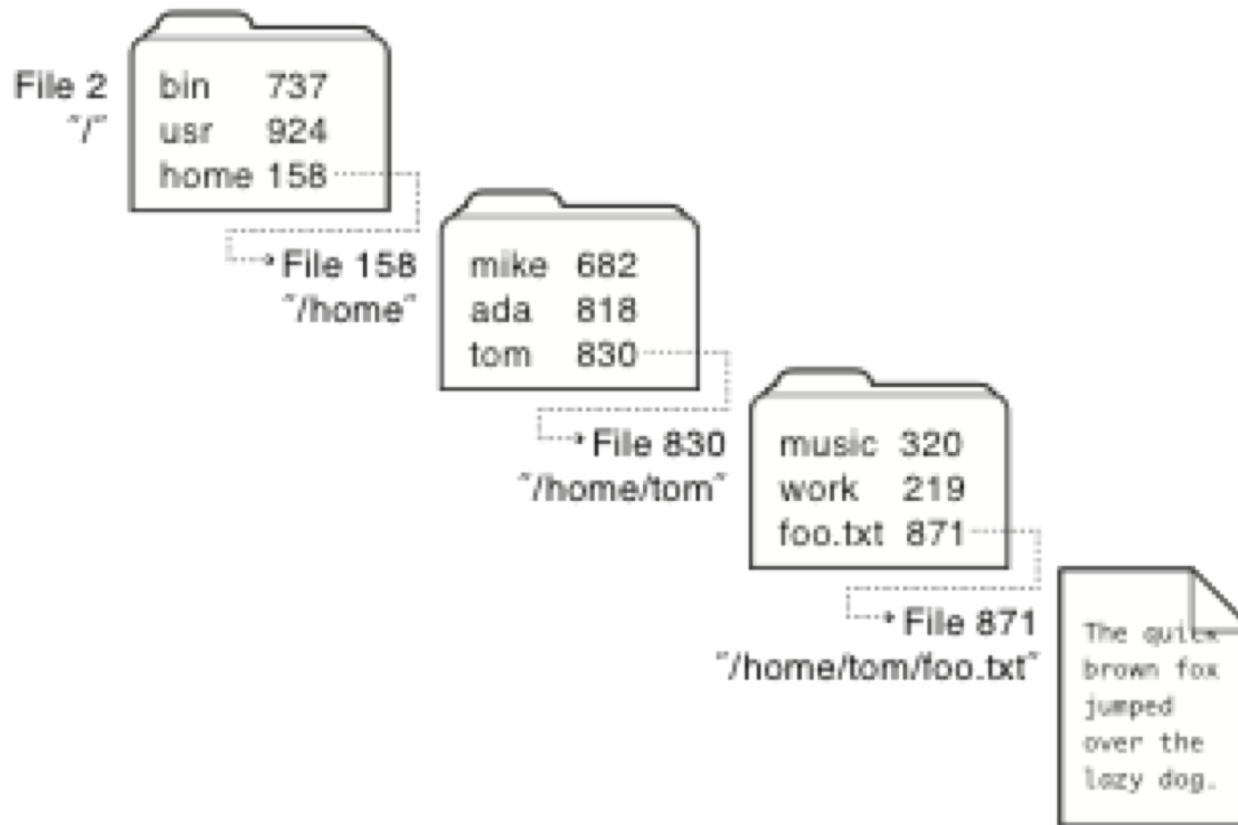
Directories Are Files

music   320
work   219
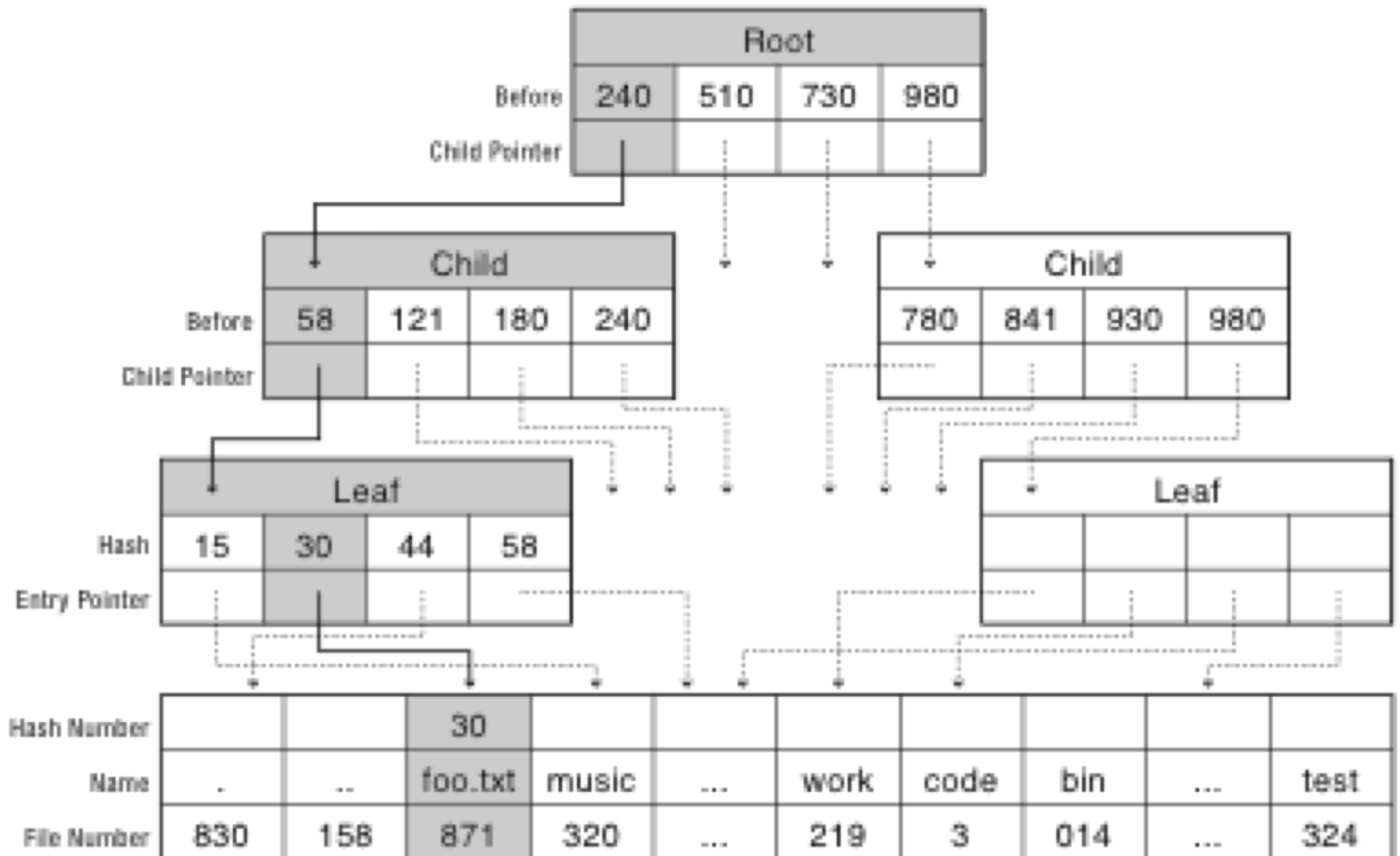foo.txt 871

Recursive Filename Lookup

Directory Layout

Directory stored as a file
Linear search to find filename (small directories)

Large Directories: B Trees

Search for Hash (foo.txt) = 0x30

Root				
Before	240	510	730	980
Child Pointer				

Child				
Before	58	121	180	240
Child Pointer				

Child				
	780	841	930	980

Leaf				
Hash	15	30	44	58
Entry Pointer				

Leaf				

Hash Number			30							
Name	.	..	foo.txt	music	...	work	code	bin	...	test
File Number	830	158	871	320	...	219	3	014	...	324

Large Directories: Layout