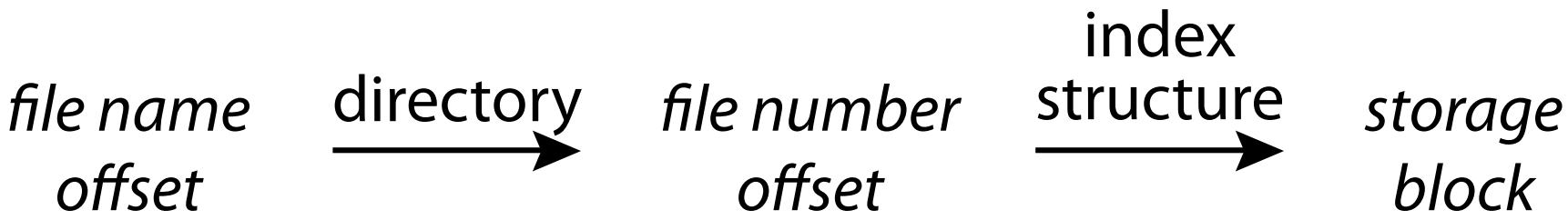
File Systems

Main Points

- File layout
- Directory layout
- Reliability/durability

Named Data in a File System



Last Time: File System Design Constraints

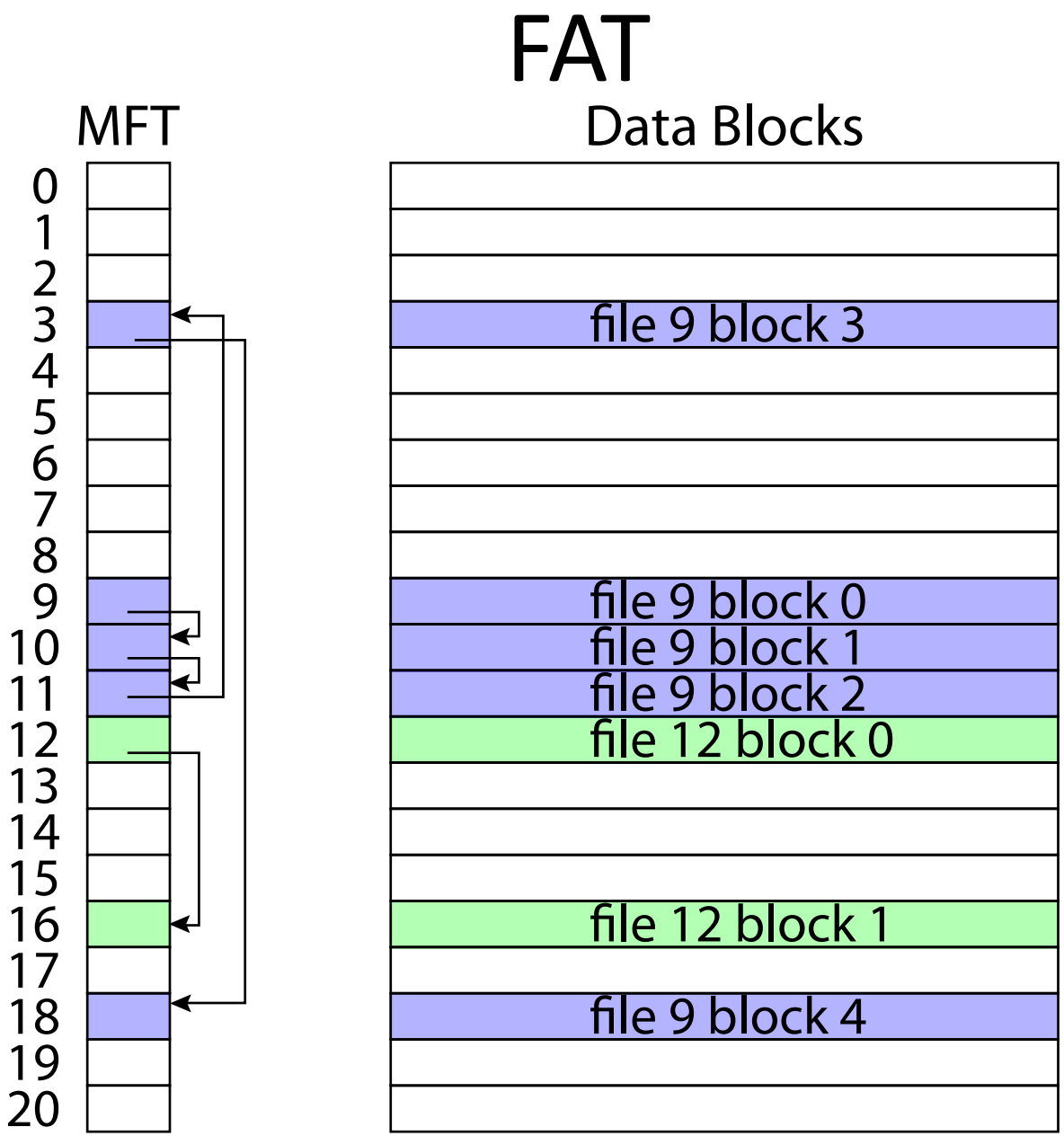
- For small files:
  - Small blocks for storage efficiency
  - Files used together should be stored together
- For large files:
  - Contiguous allocation for sequential access
  - Efficient lookup for random access
- May not know at file creation
  - Whether file will become small or large

File System Design Options

	FAT	FFS	NTFS
Index structure	Linked list	Tree (fixed, asym)	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

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

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 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

FFS inode

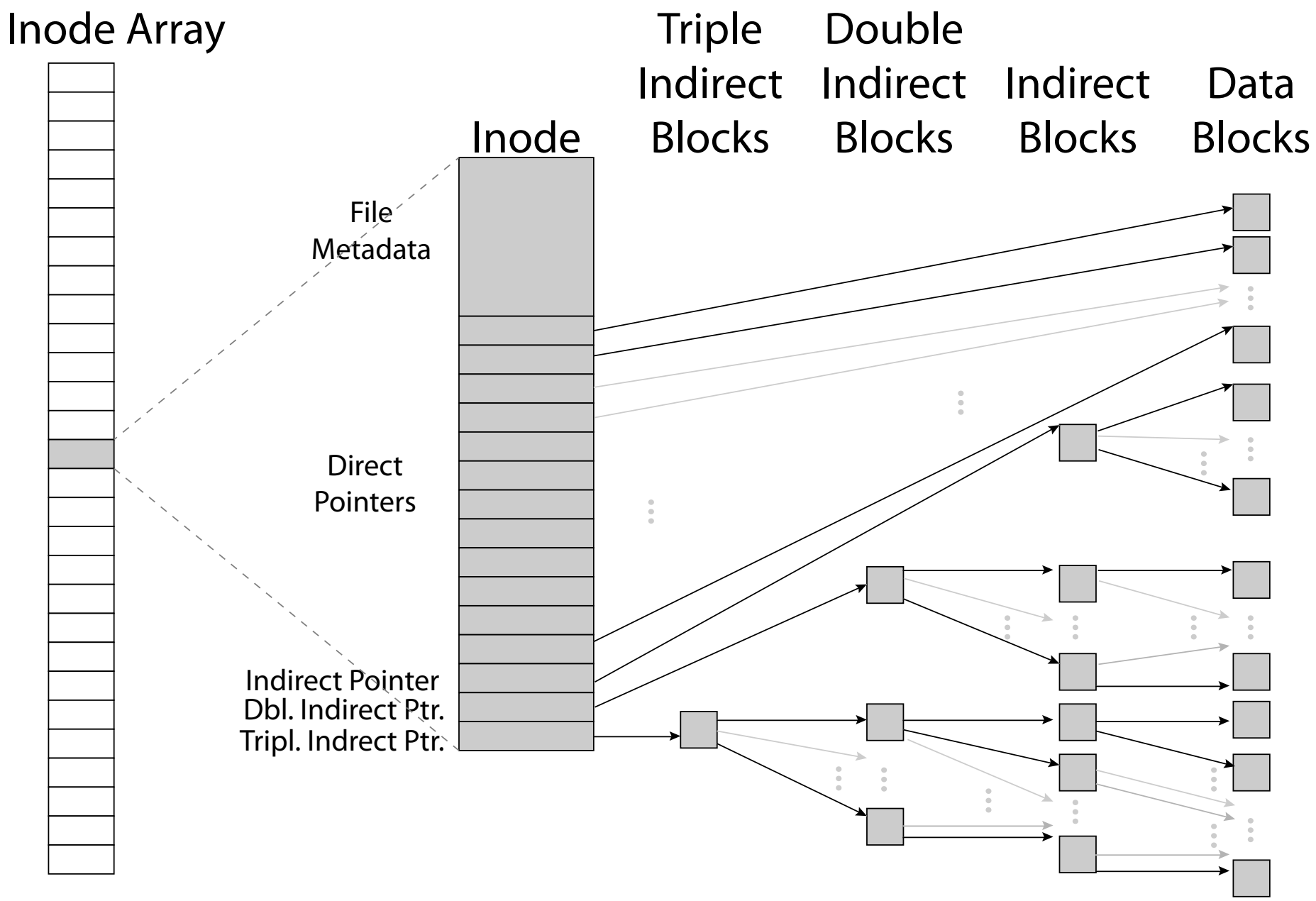
- Metadata
  - File owner, access permissions, access times, …
- Set of 12 data pointers
  - With 4KB blocks => max size of 48KB files
- Indirect block pointer
  - pointer to disk block of data pointers
- Indirect block: 1K data blocks => 4MB (+48KB)

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)

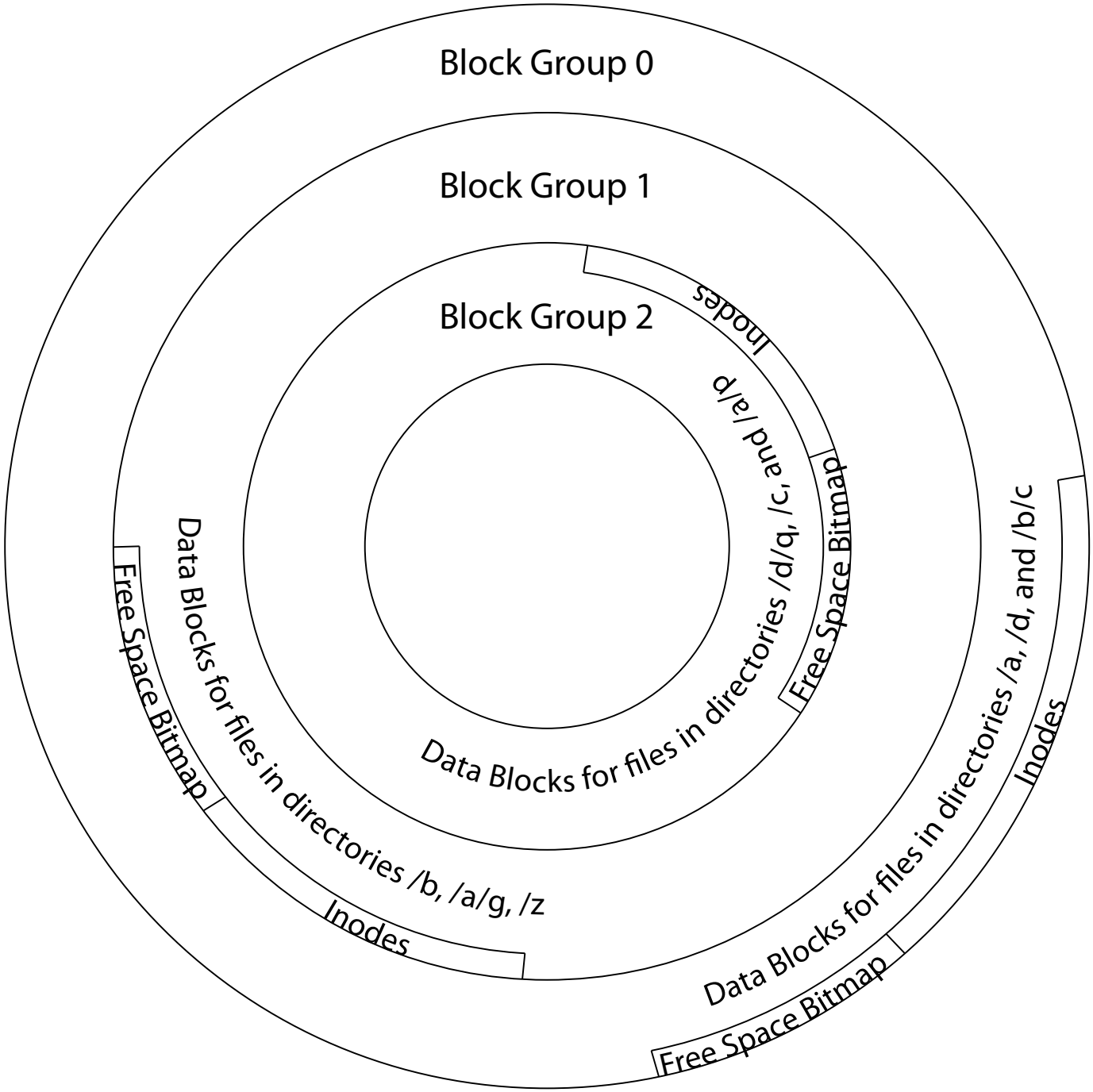

Inode Array
Inode
Triple Indirect Blocks
Double Indirect Blocks
Indirect Blocks
Data Blocks
File Metadata
Direct Pointers
Indirect Pointer
Dbl. Indirect Ptr.
Tripl. Indrect Ptr.

FFS Asymmetric Tree

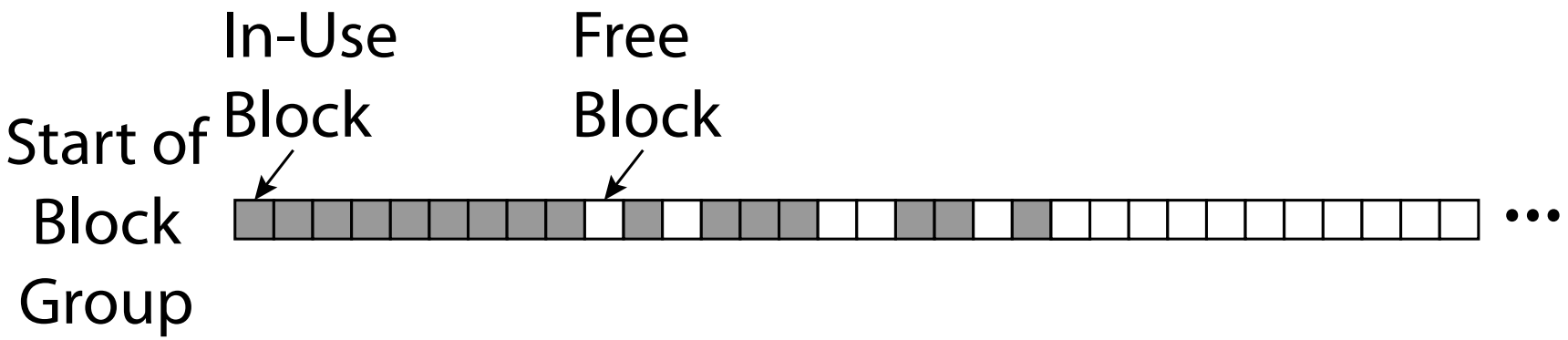
- Small files: shallow tree
  - Efficient storage for small files
- Large files: deep tree
  - Efficient lookup for random access in large files

FFS Locality

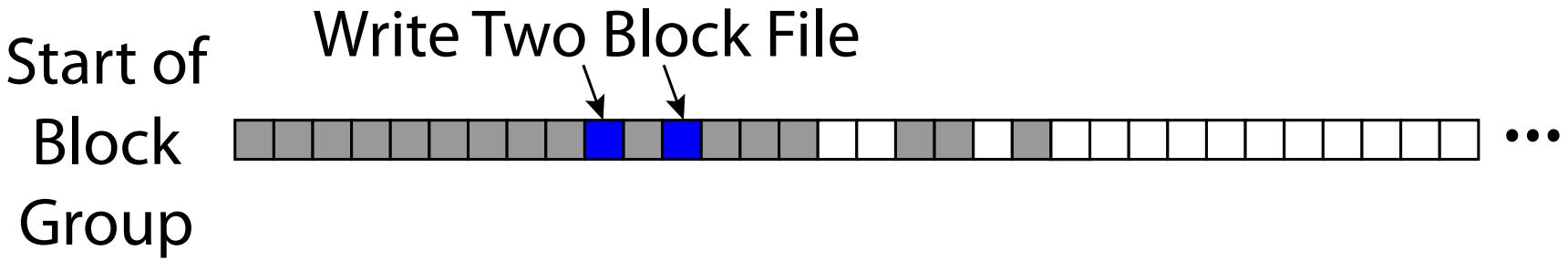
- Block group allocation
  - Block group is a set of nearby cylinders
  - Files in same directory located in same group
  - Subdirectories located in different block groups
- inode table spread throughout disk
  - inodes, bitmap near file blocks
- First fit allocation
  - Small files fragmented, large 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
Data Blocks for files in directories /b, /a/g, /z
Free Space Bitmap
Inodes
Data Blocks for files in directories /a, /d, and /b/c
Free Space Bitmap
Inodes

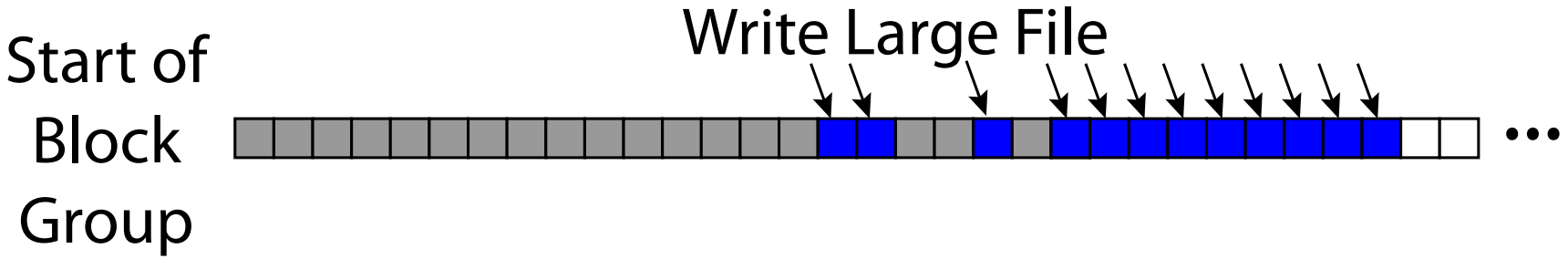
FFS First Fit Block Allocation



FFS First Fit Block Allocation



FFS First Fit Block Allocation



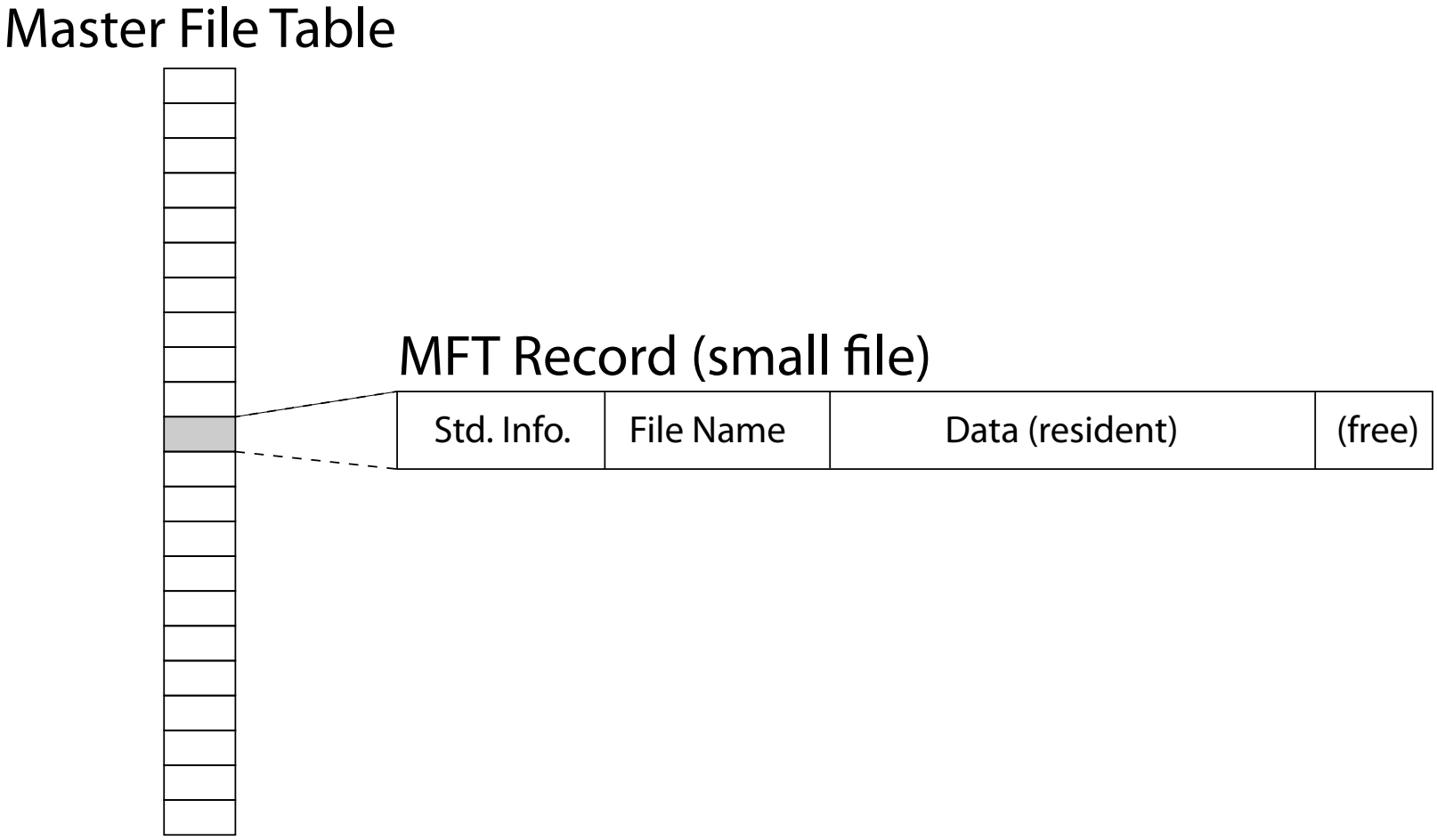
FFS

- 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

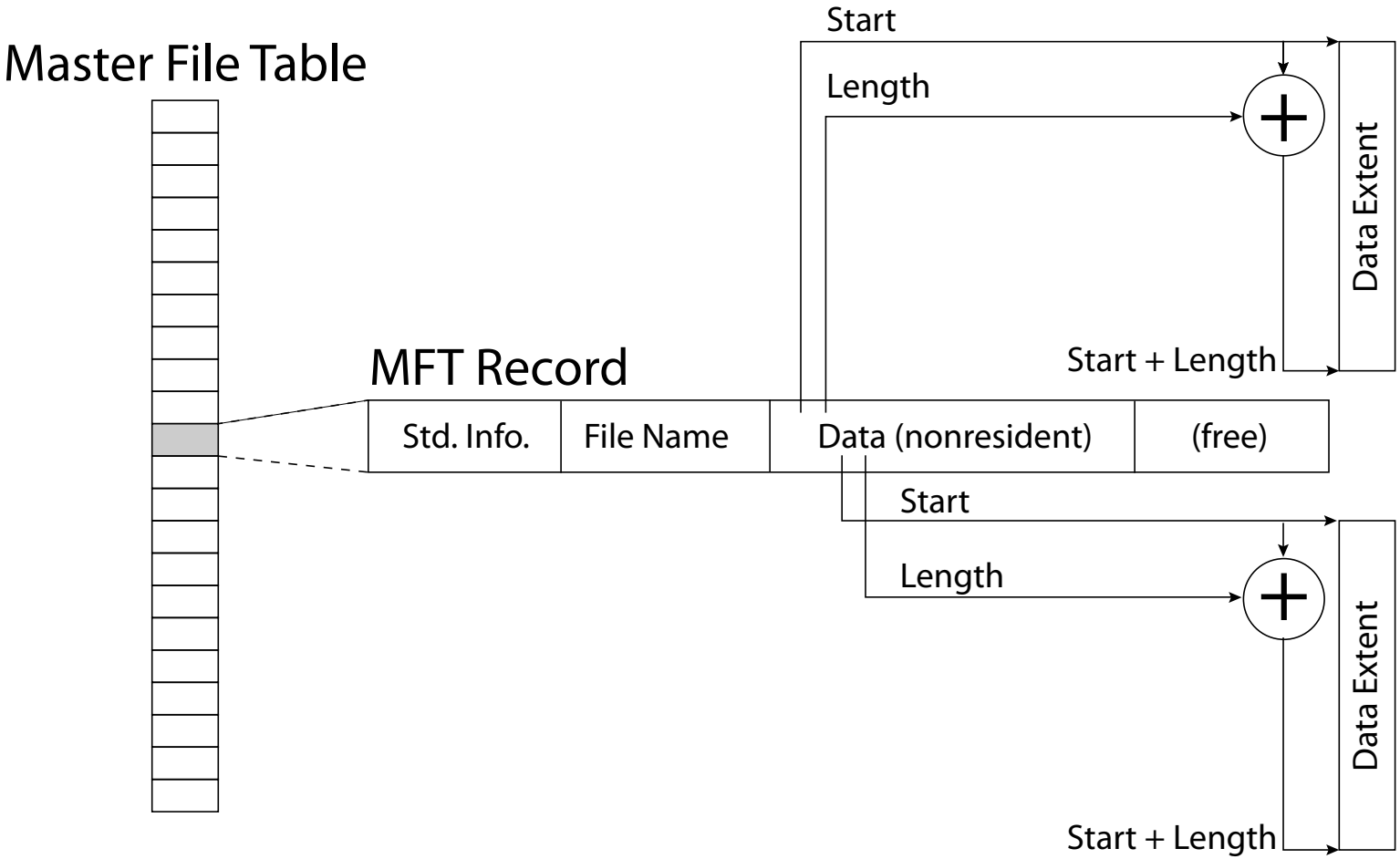
NTFS

- 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
  - Discussed next time

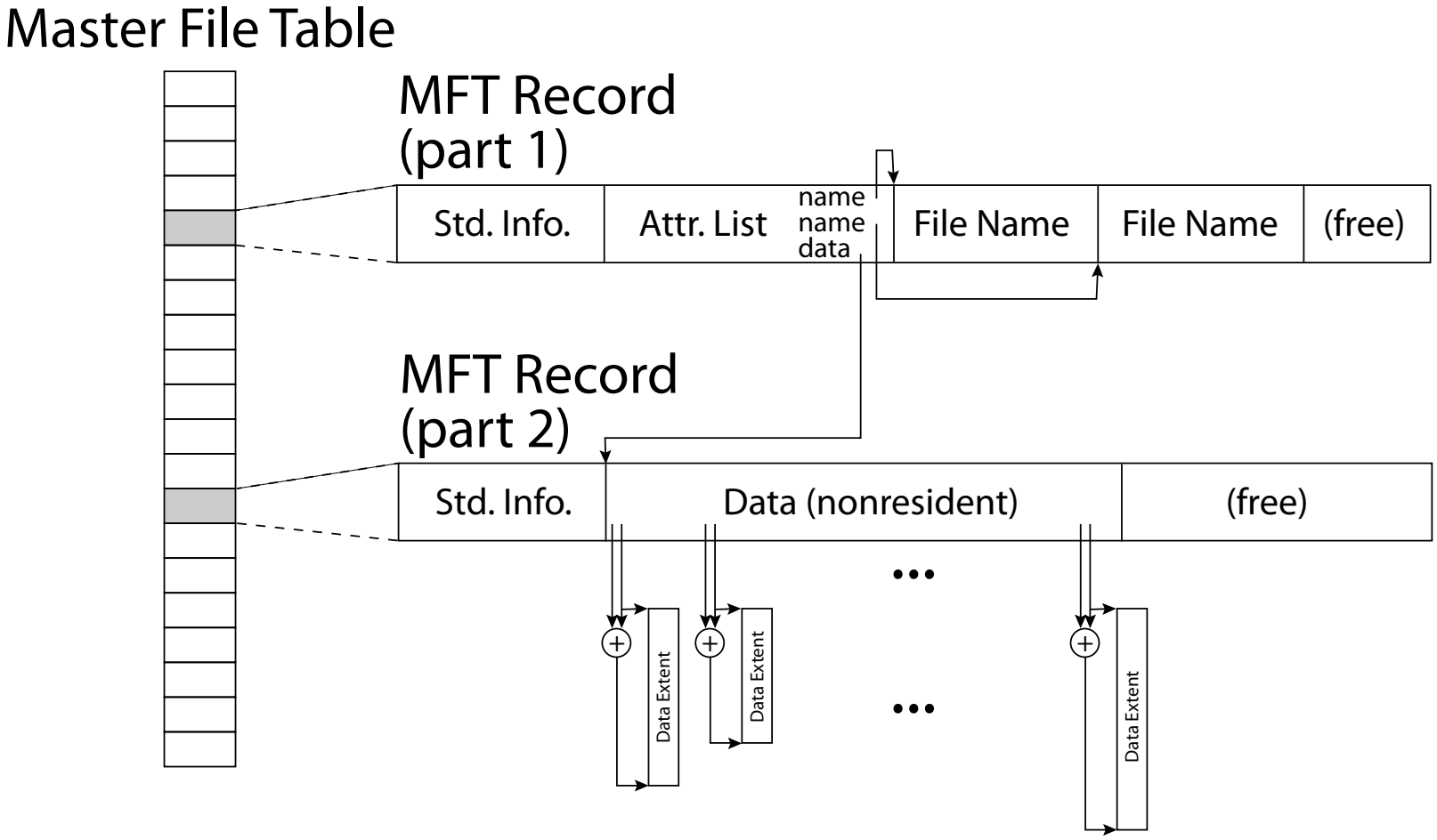
NTFS Small File



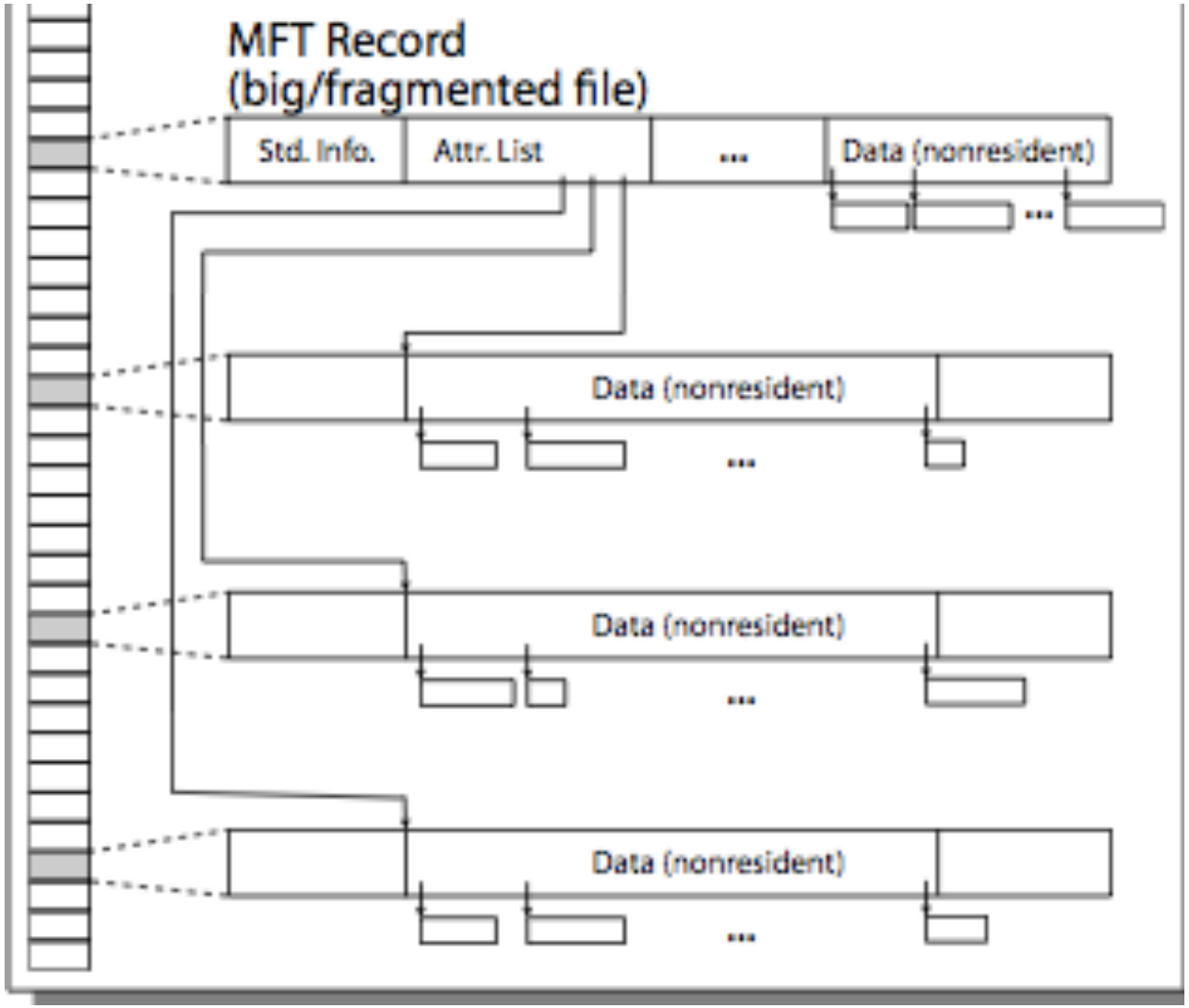
NTFS Medium File

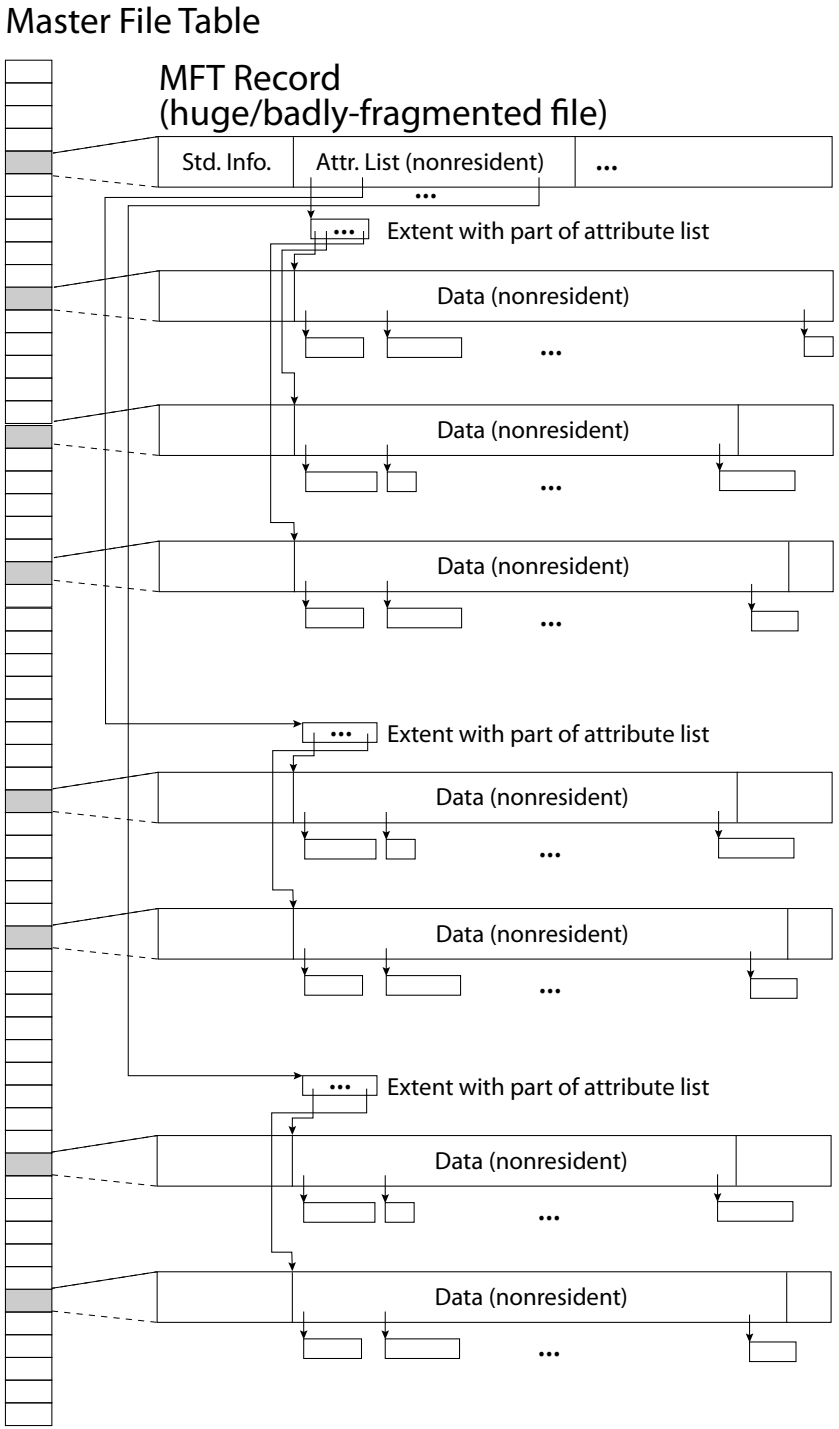


NTFS Indirect Block

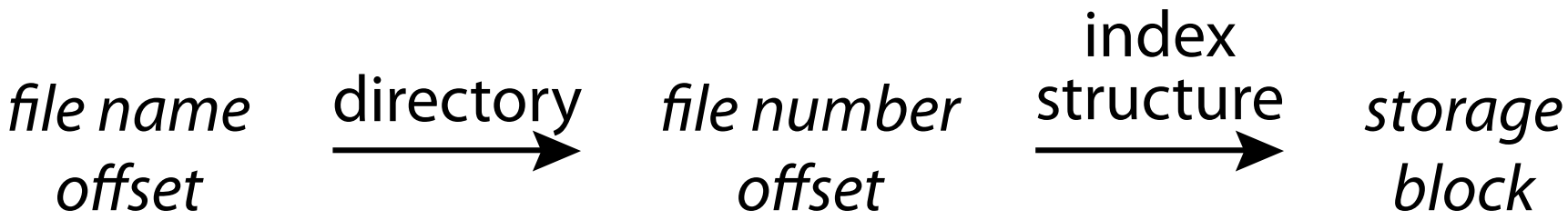


NTFS Multiple Indirect Blocks

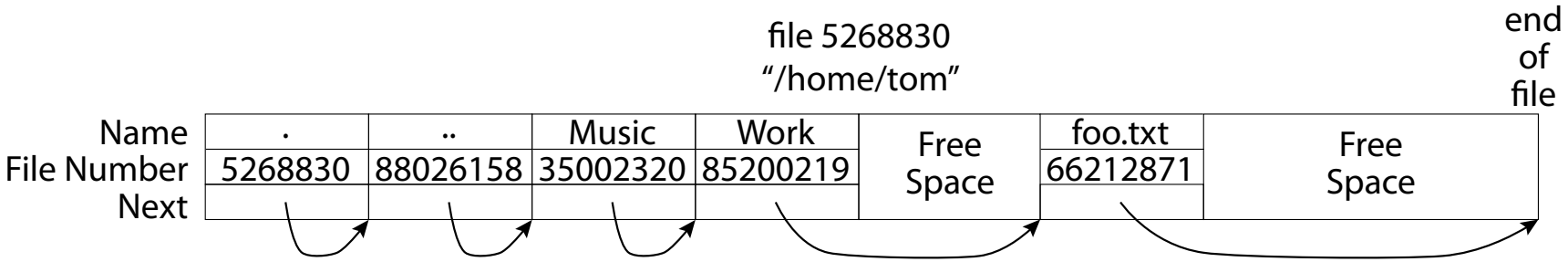



Master File Table
MFT Record
(huge/badly-fragmented file)
Std. Info.
Attr. List (nonresident)
...
Extent with part of attribute list
Data (nonresident)
Data (nonresident)
Data (nonresident)
Extent with part of attribute list
Data (nonresident)
Data (nonresident)
Extent with part of attribute list
Data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

Named Data in a File System

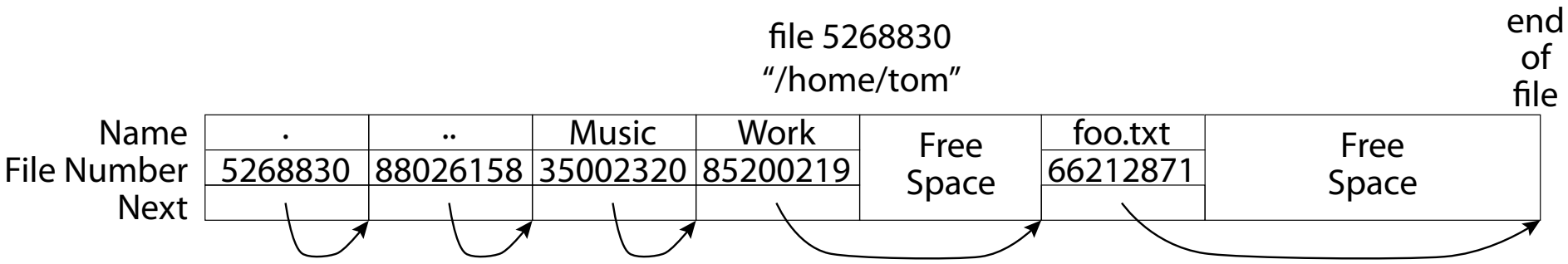


Directories

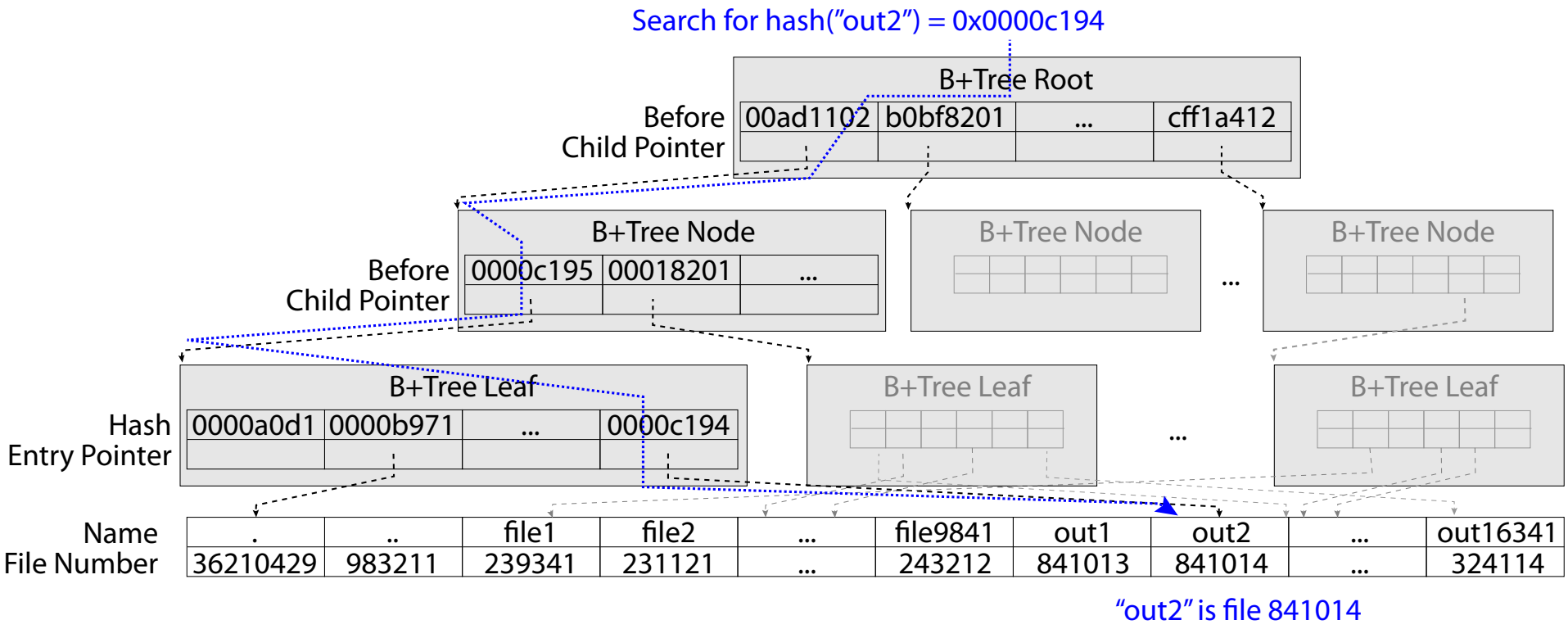


Directories

- Directories can be files
  - Map file name to file number (MFT #, inode num)
- Table of file name -> file number
  - Small directories: linear search



Large Directories: B-Trees



File System Reliability

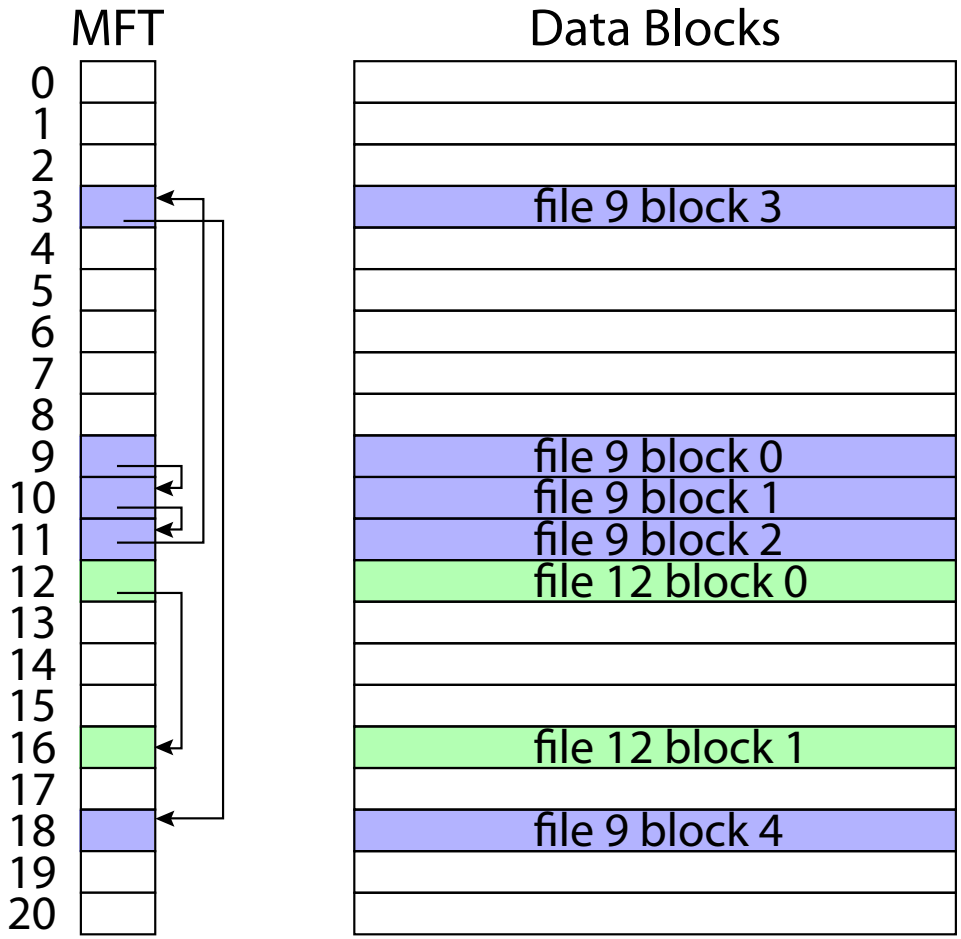
- What can happen if disk loses power or machine software crashes?
  - Some operations in progress may complete
  - Some operations in progress may be lost
  - Overwrite of a block may only partially complete
- File system wants durability (as a minimum!)
  - Data previously stored can be retrieved (maybe after some recovery step), regardless of failure

Storage Reliability Problem

- File operations often involve updates to multiple disk blocks
  - inode, indirect block, data block, bitmap, …
- At a physical level, operations complete one at a time
  - Hard to guarantee that they will all complete
- Many disk devices have an extra capacitor to allow writes on current track to complete

FAT Reconsidered

- To append to a file:
  - Add data block
  - Add pointer to data block
  - Update access time for entry at head of file



Reliability Approach #1

- Sequence operations in a specific order
- Careful design to allow sequence to be interrupted safely
  - Write data block before updating FAT entry
- Requires post-crash recovery
  - Write file before updating directory
  - May leave file created, without being in any directory