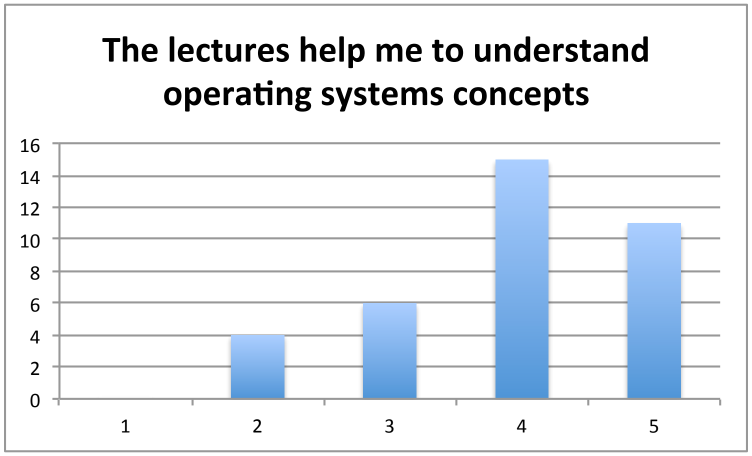
Goals for Today

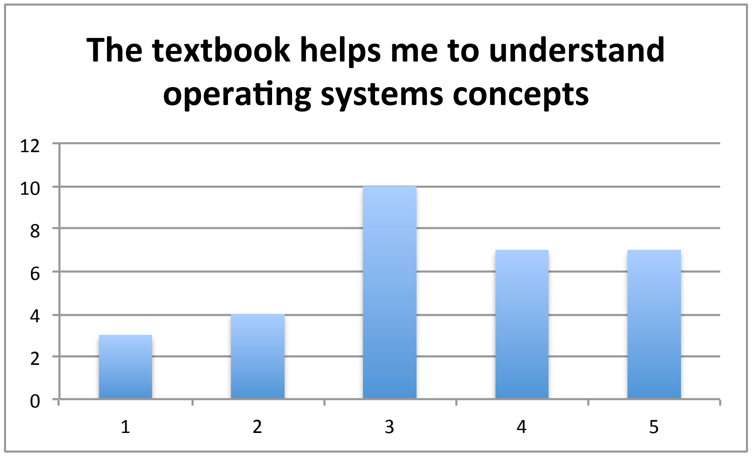
- Learning Objective:
  - Understand the challenges of file system design
- Announcements, etc:
  - C4: All remaining submissions open on Compass now
  - MP3 is out! Due **April 18th**.
  - Summary of IEF on next slide

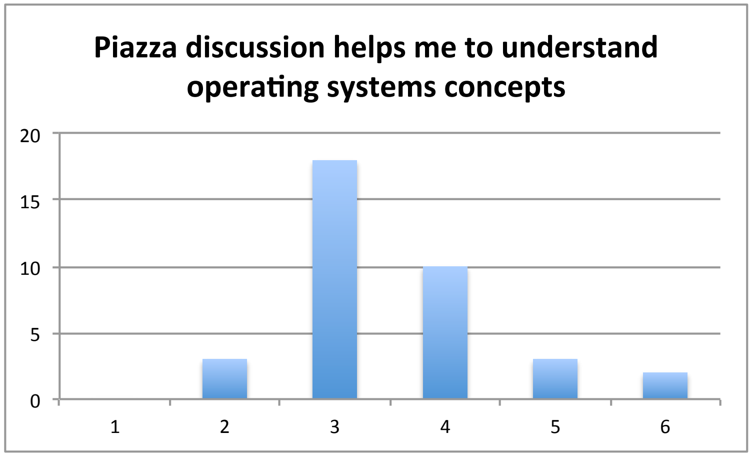
Reminder: Please put away devices at the start of class

Informal Early Feedback


The lectures help me to understand operating systems concepts

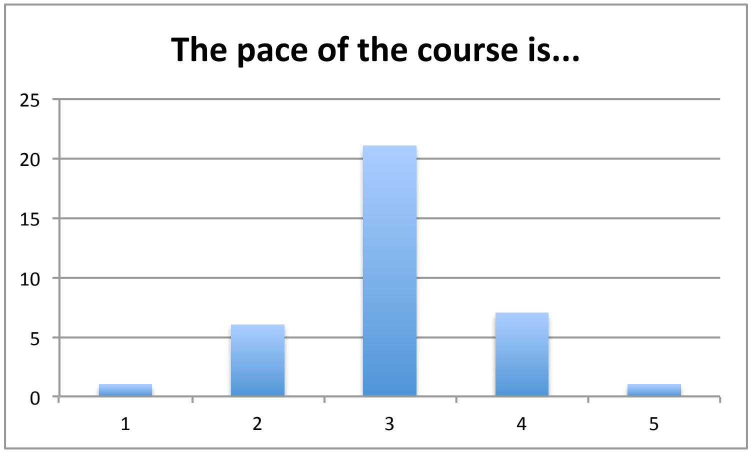
Lecture usefulness… pretty good!
Piazza/Text Usefulness…. less good.

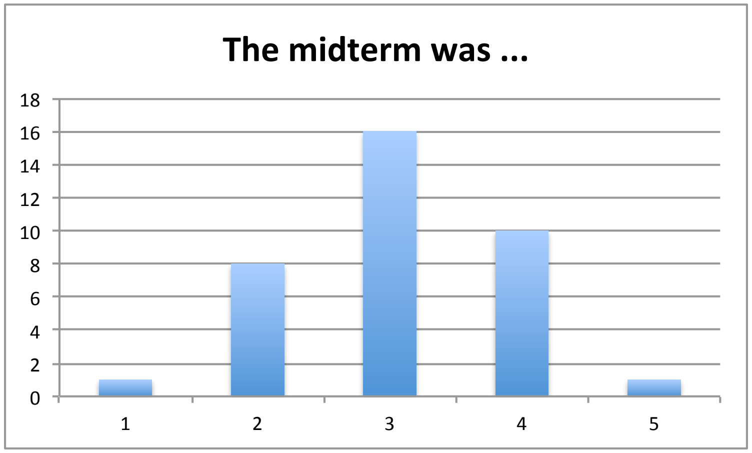

The textbook helps me to understand operating systems concepts

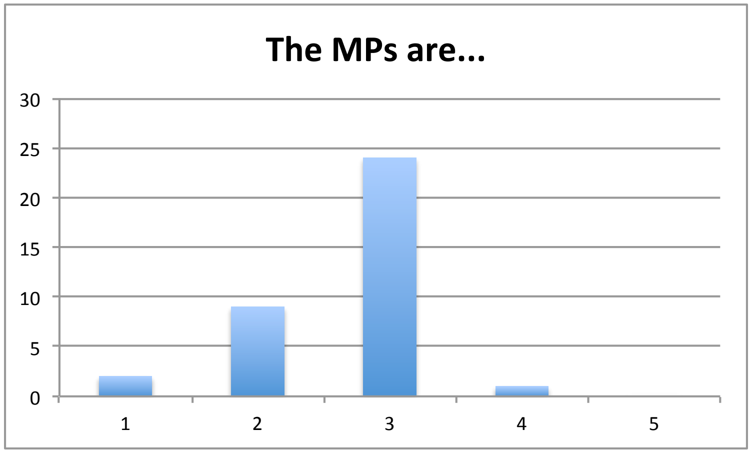

Piazza discussion helps me to understand operating systems concepts

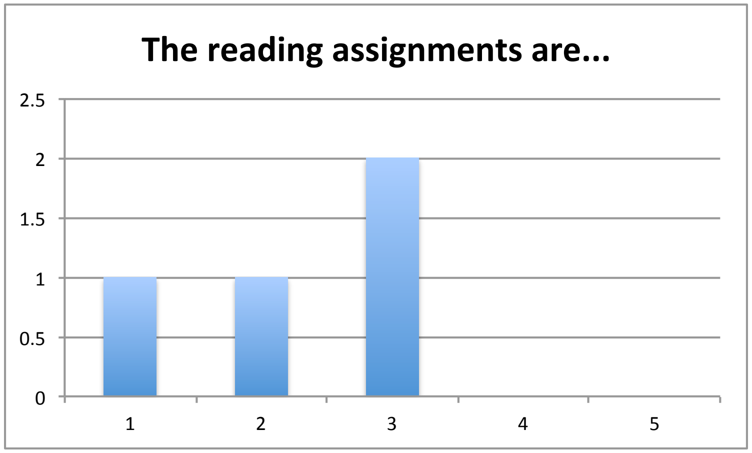
Never 1---2---3---4---5 Always

Informal Early Feedback


The pace of the course is...


The midterm was ...


The MPs are...


The reading assignments are...

Too Fast 1---2---3---4---5 Too Easy

Informal Early Feedback

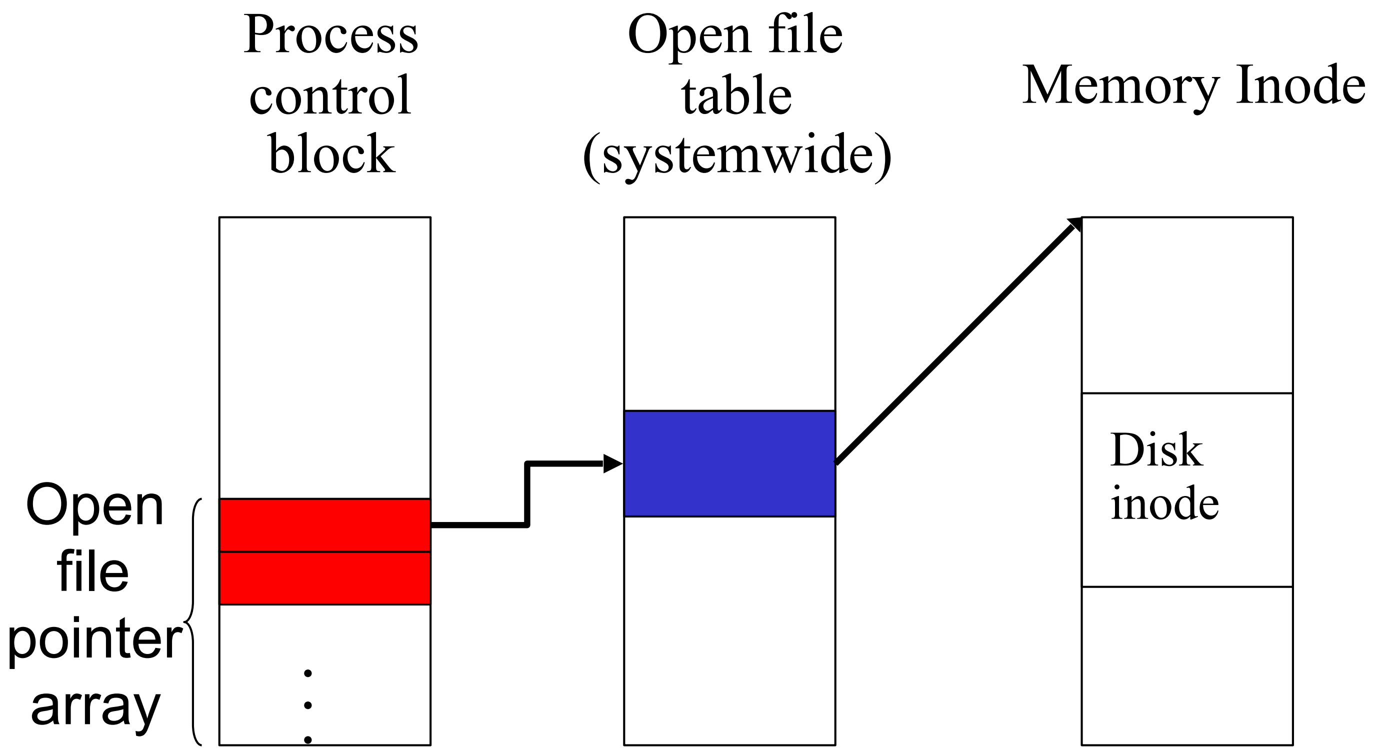
- Highlights from Comments
  - Most Popular Topics: Scheduling, Virtual Memory
  - Practical Exposure is popular, MPs considered useful
  - Concerns about Participation Points
    - Class Discussion
  - More practice (e.g., HW, Quiz) to help w/ final prep
  - Increase MPs and/or add a team project??

CS 423
Operating System Design: File System Design

Professor Adam Bates
Spring 2018

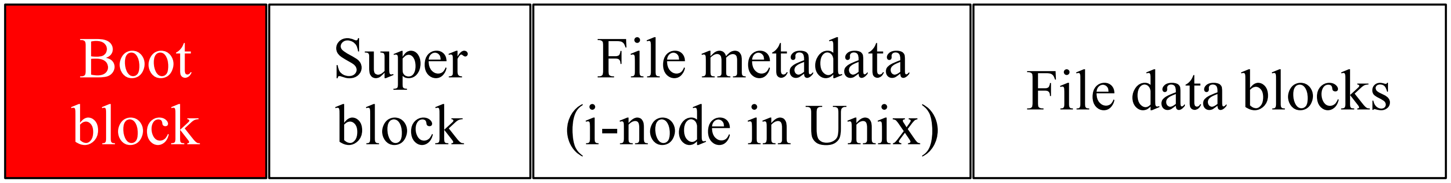
Data Structures for a FS

Data structures in a typical file system:



Disk Layout for a FS

Disk layout in a typical file system:



Boot block

Super block

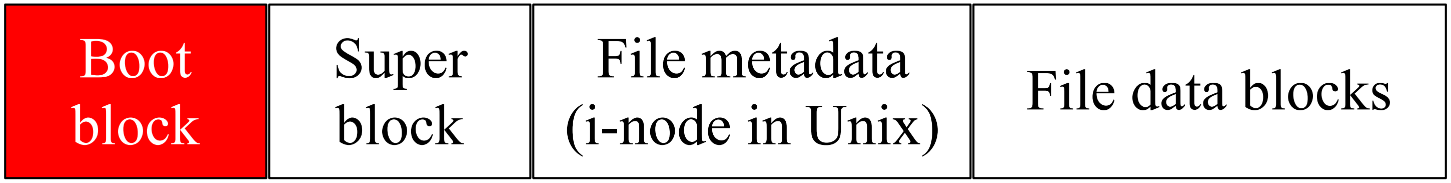
File metadata (i-node in Unix)

File data blocks

- Data Structures:
  - File data blocks: File contents
  - File metadata: How to find file data blocks
  - Directories: File names pointing to file metadata
  - Free map: List of free disk blocks

Disk Layout for a FS

Disk layout in a typical file system:



Boot block	Super block	File metadata (i-node in Unix)	File data blocks

- Superblock defines a file system
  - size of the file system
  - size of the file descriptor area
  - free list pointer, or pointer to bitmap
  - location of the file descriptor of the root directory
  - other meta-data such as permission and various times
- For reliability, replicate the superblock

Design Constraints

- How can we allocate files efficiently?
- 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
- Challenge: May not know at file creation where our file will be small or large!!

Design Challenges

- Index structure
  - *How do we locate the blocks of a file?*
- Index granularity
  - *How much data per each index (i.e., block size)?*
- 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 Allocation

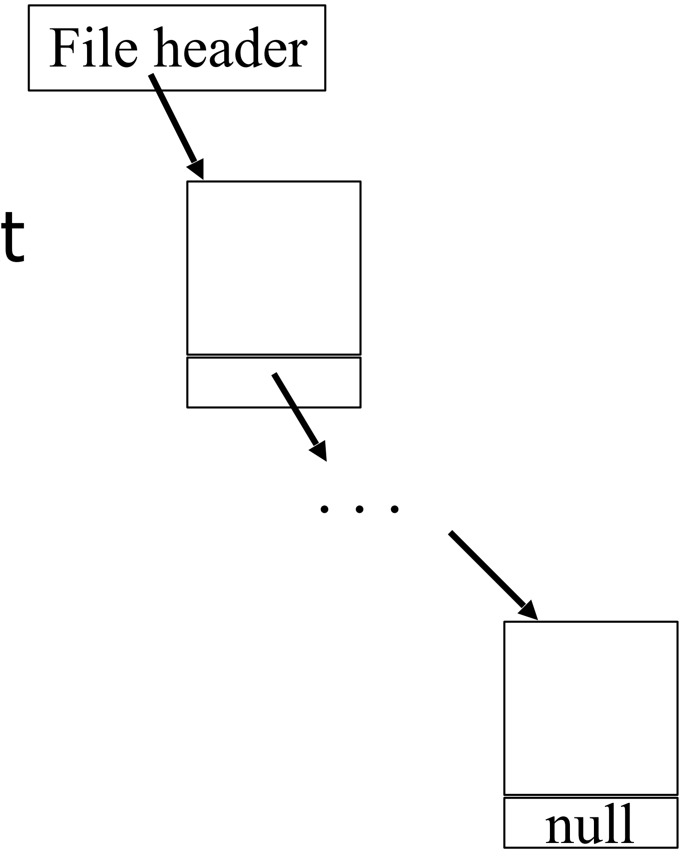
- Contiguous
- Non-contiguous (linked)
- Tradeoffs?

Contiguous Allocation

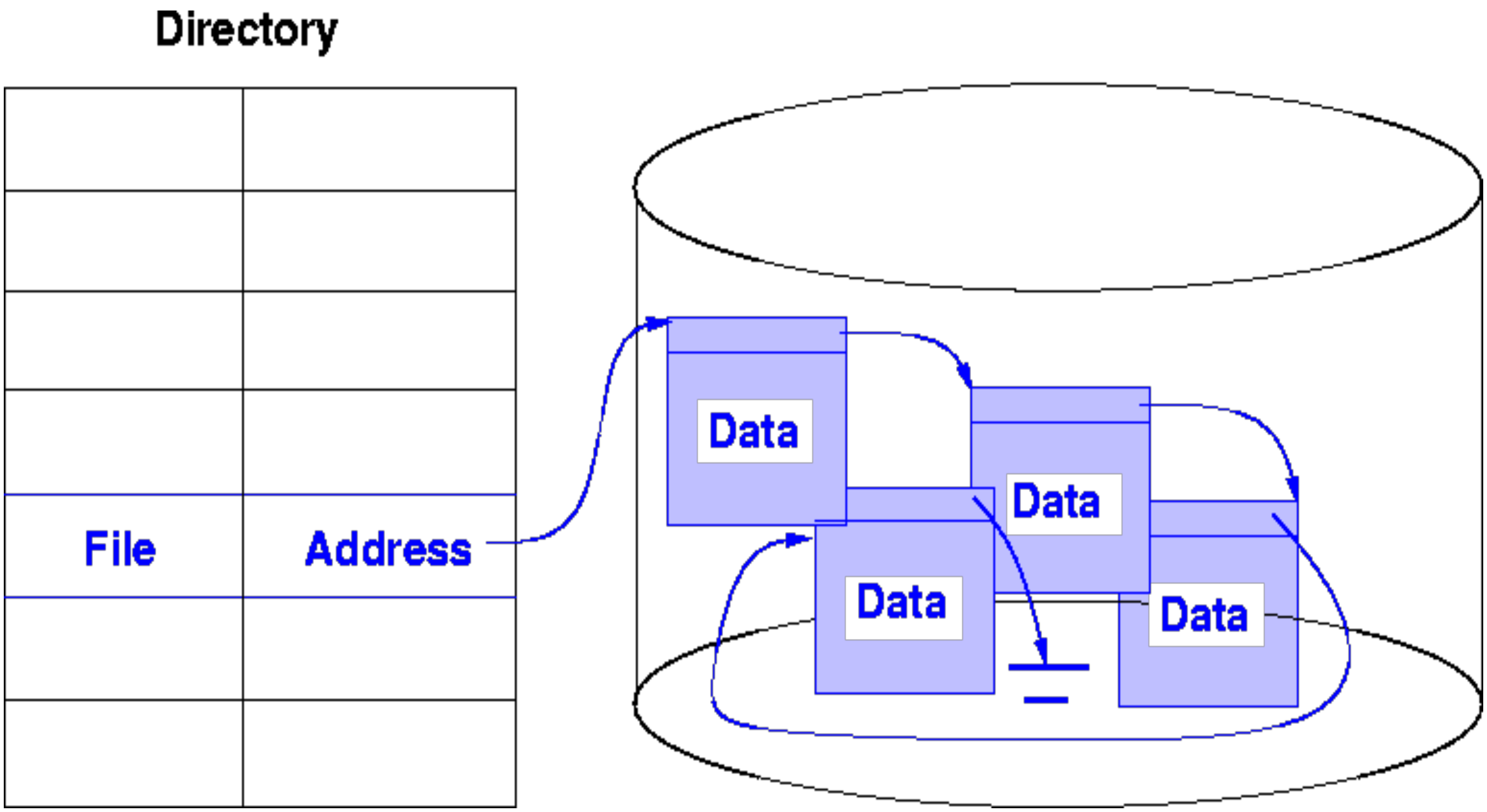
- Request in advance for the size of the file
- Search free map to locate a space
- File header
  - first sector in file
  - number of sectors
- Pros
  - Fast sequential access
  - Easy random access
- Cons
  - External fragmentation
  - Hard to grow files

Linked Files

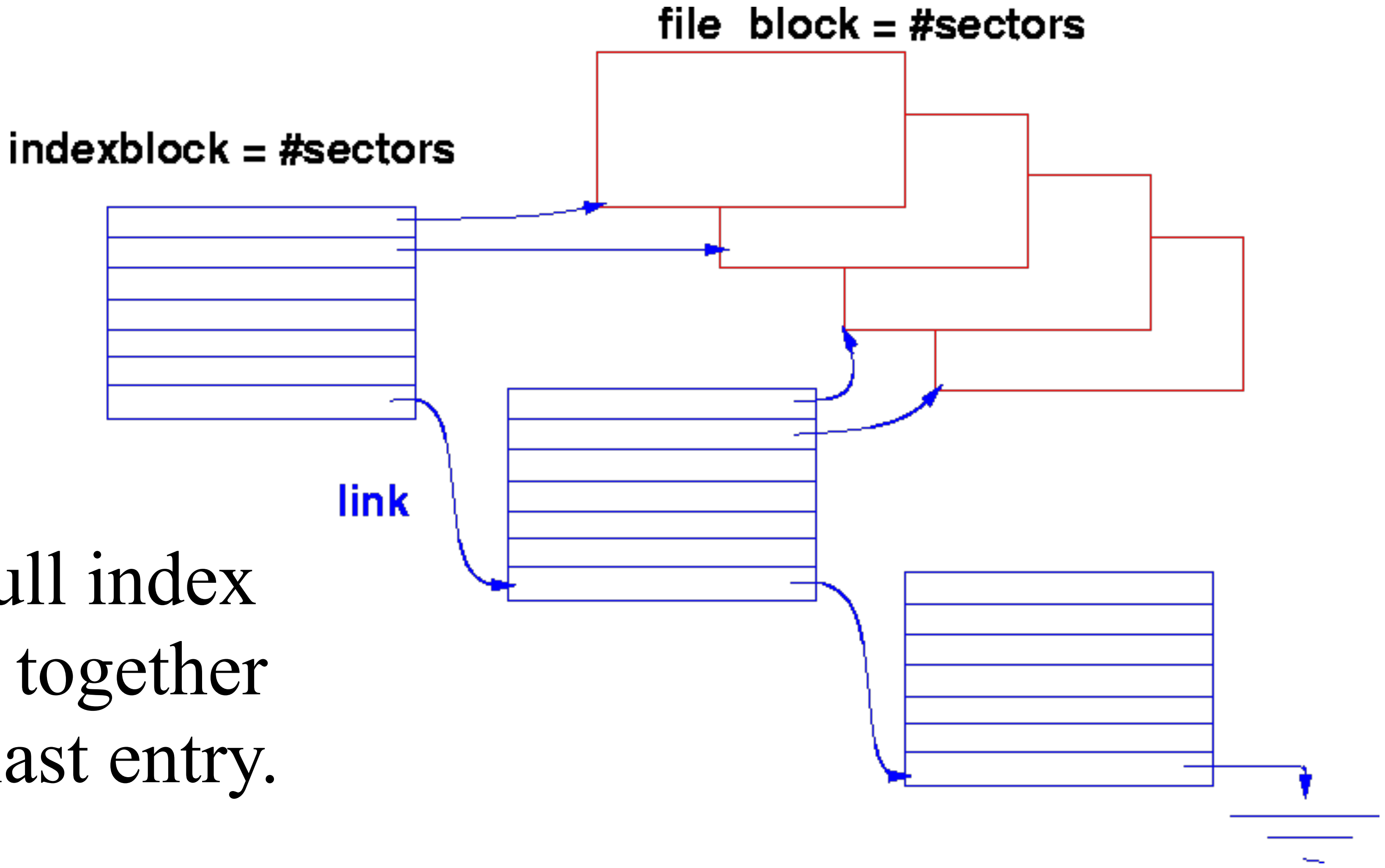
- File header points to 1st block on disk
- Each block points to next
- Pros
  - Can grow files dynamically
  - Free list is similar to a file
- Cons
  - random access: horrible
  - unreliable: losing a block means losing the rest



Linked Allocation

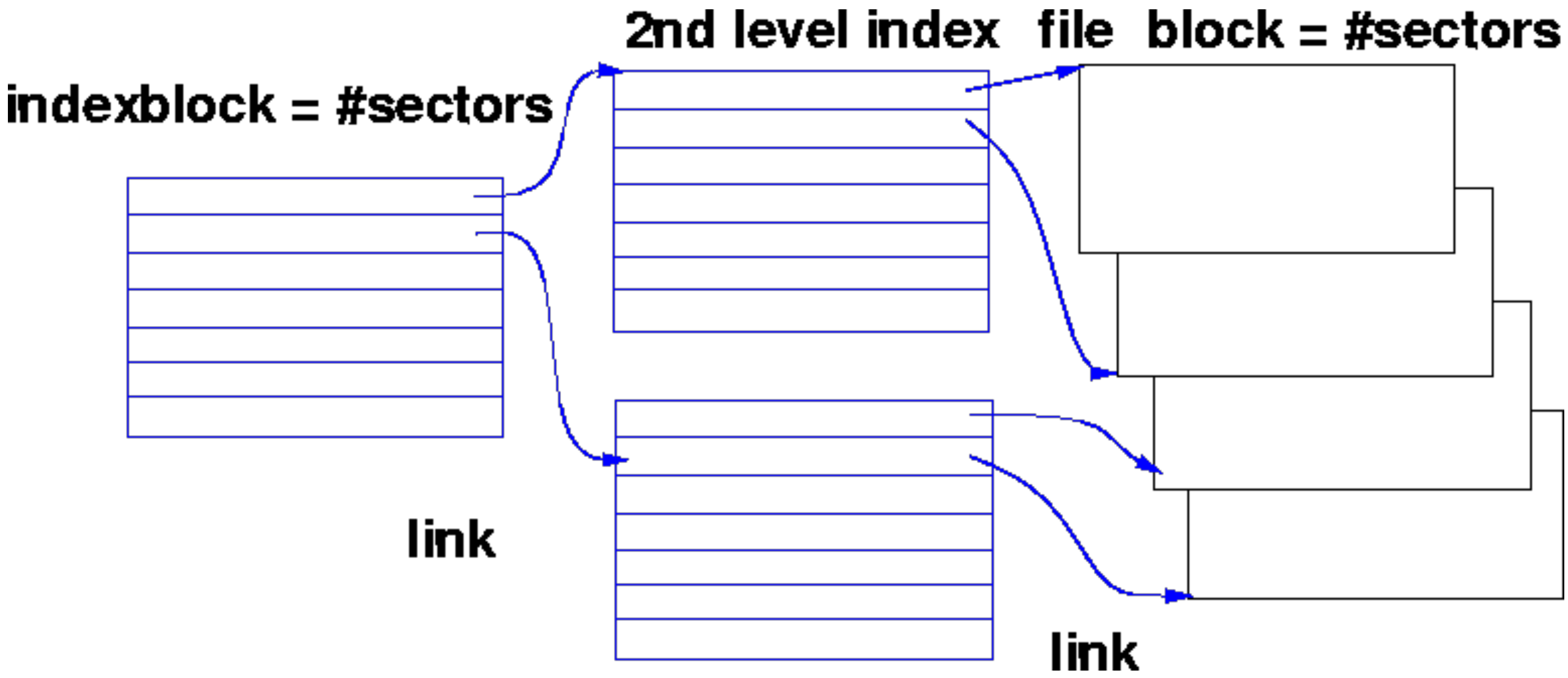


Indexed File Allocation



Link full index blocks together using last entry.

Multilevel Indexed Files



Multiple levels of index blocks

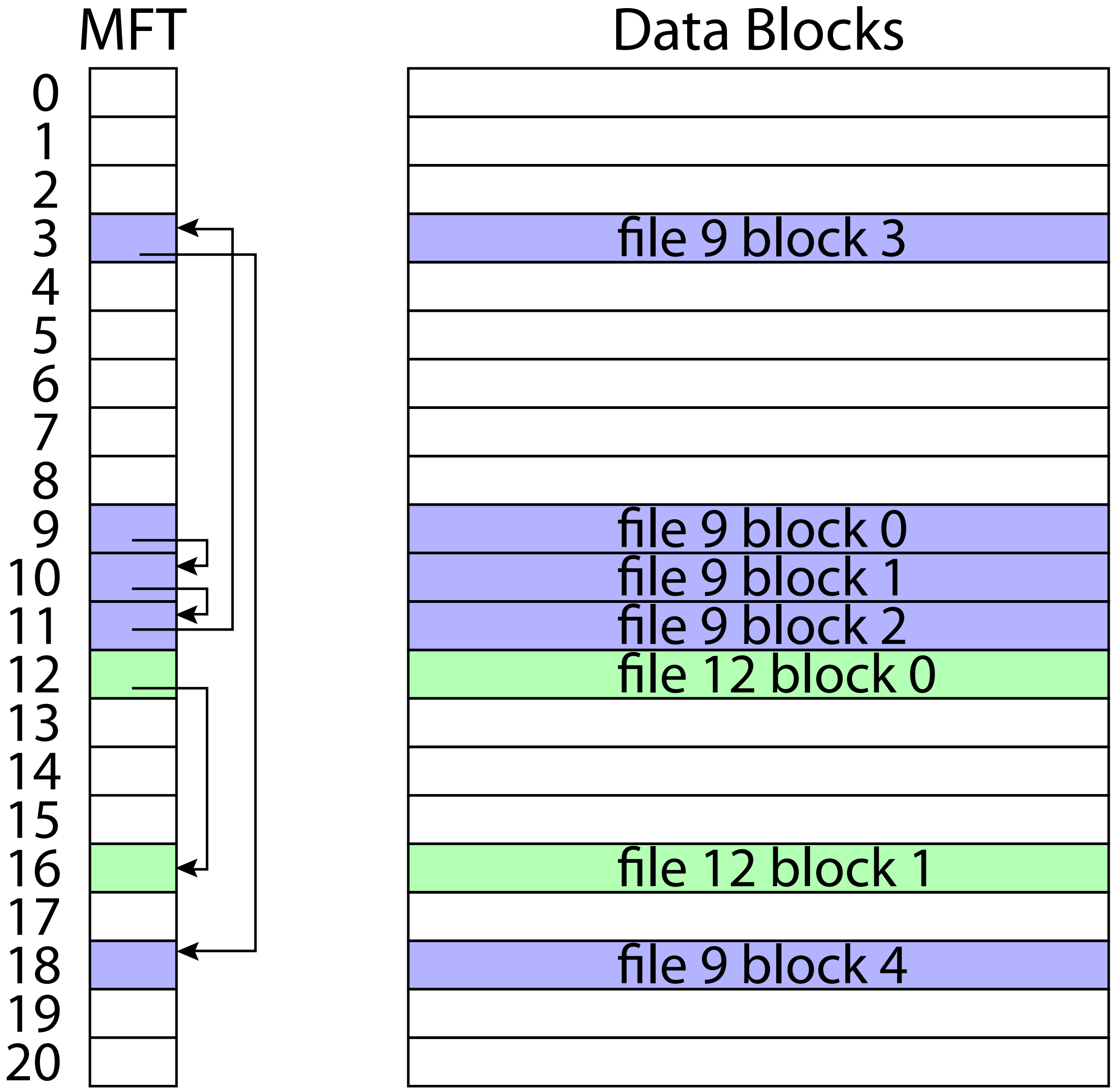
File Systems In Practice

	FAT	Berkeley FFS (Unix FS)	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

MS 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

MS File Allocation Table (FAT)



MS File Allocation Table (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

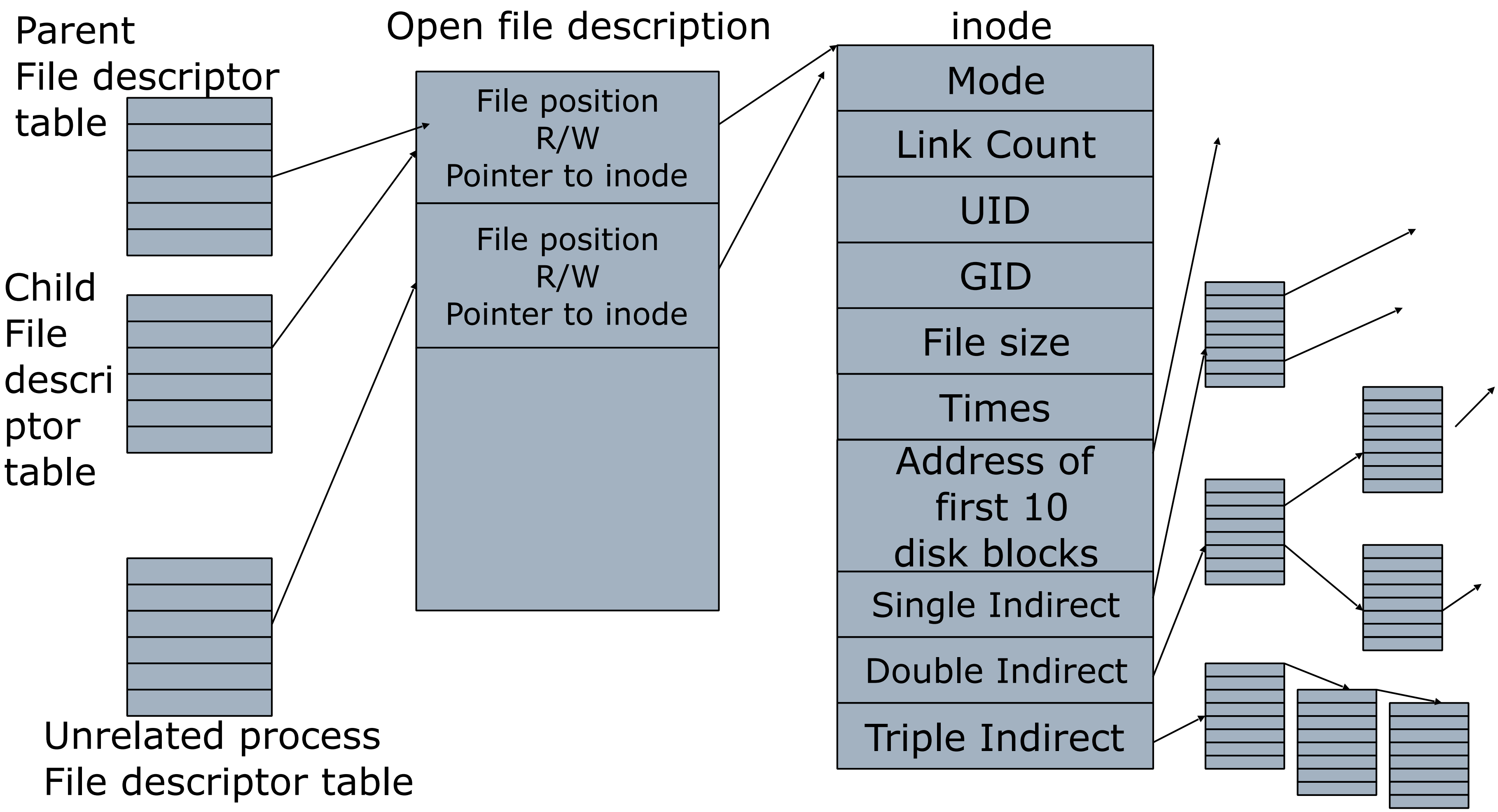
Berkeley FFS / UNIX FS

- “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
  - Indirect block pointers
    - pointer to disk block of data pointers
    - w/ indirect blocks, we can point to 1K data blocks => 4MB (+48KB)
  - … but why stop there??

Berkeley FFS / UNIX FS

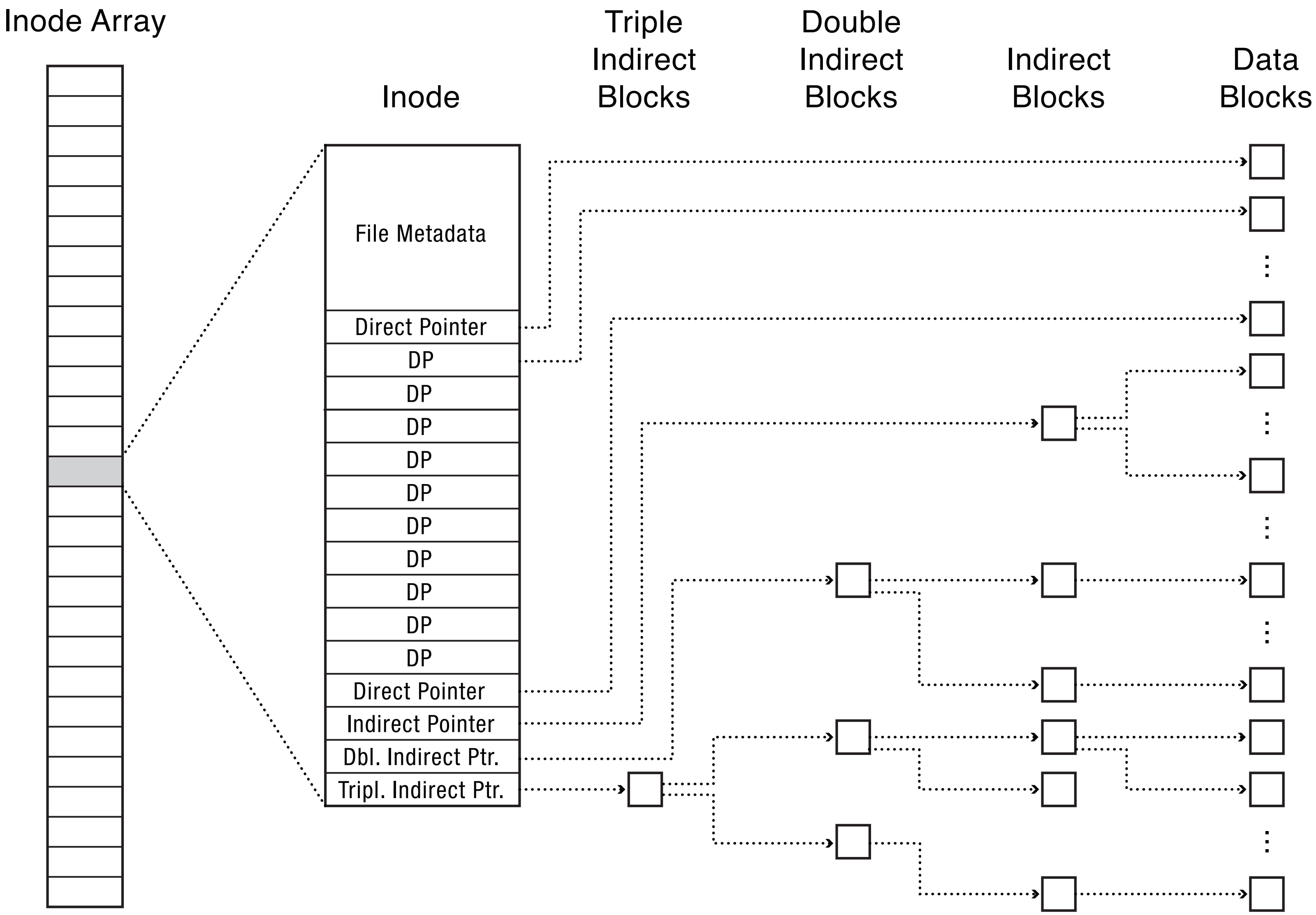
- Doubly indirect block pointer
  - w/ doubly indirect blocks, we can point to 1K indirect blocks
  - => 4GB (+ 4MB + 48KB)
- Triply indirect block pointer
  - w/ triply indirect blocks, we can point to 1K doubly indirect blocks
  - 4TB (+ 4GB + 4MB + 48KB)

Berkeley FFS / UNIX FS



Berkeley FFS / UNIX FS

Alternate figure, same basic idea



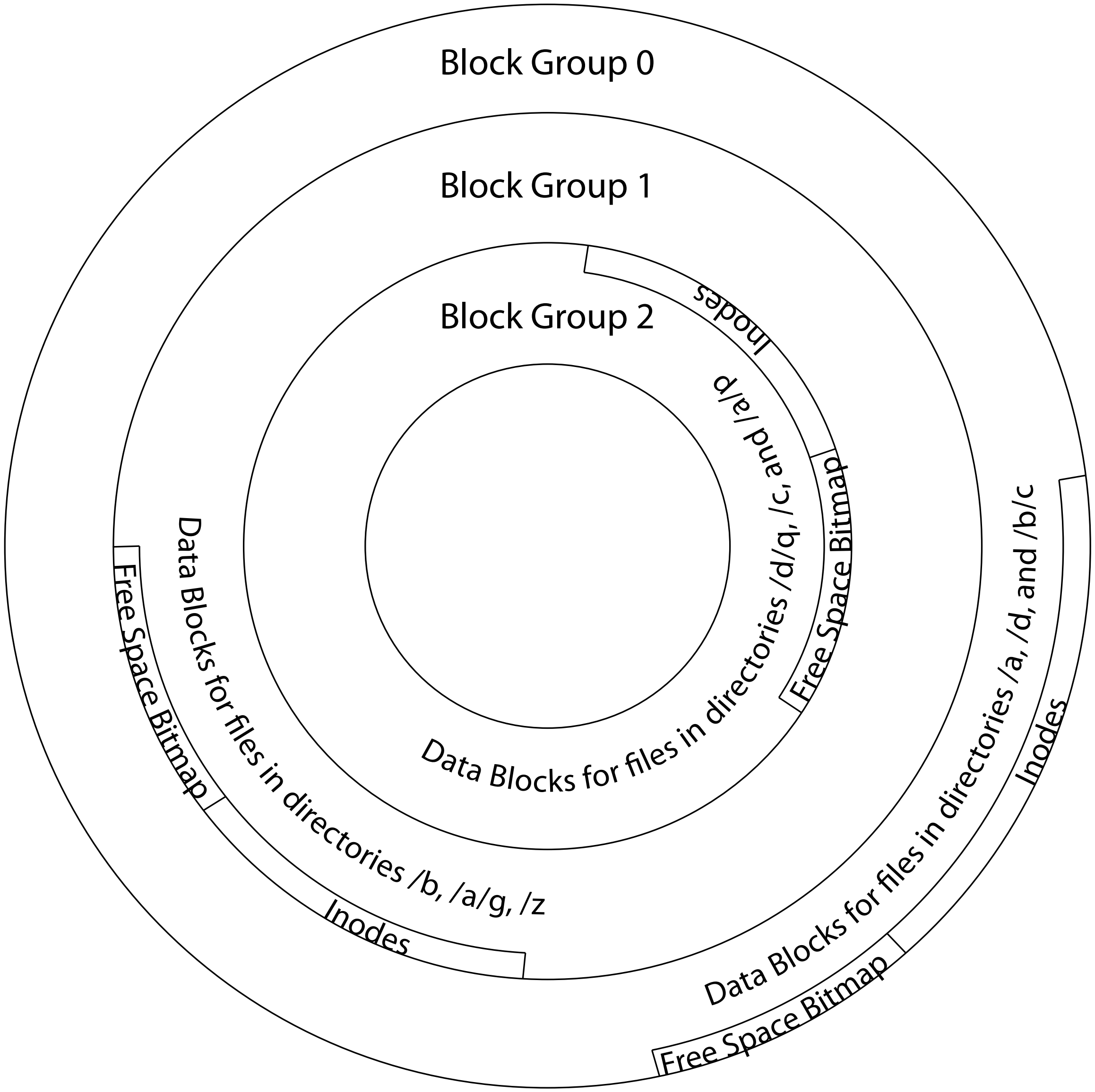
Berkeley FFS Asym. Trees

- Indirection has a cost. Only use if needed!
- 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

Berkeley FFS Locality

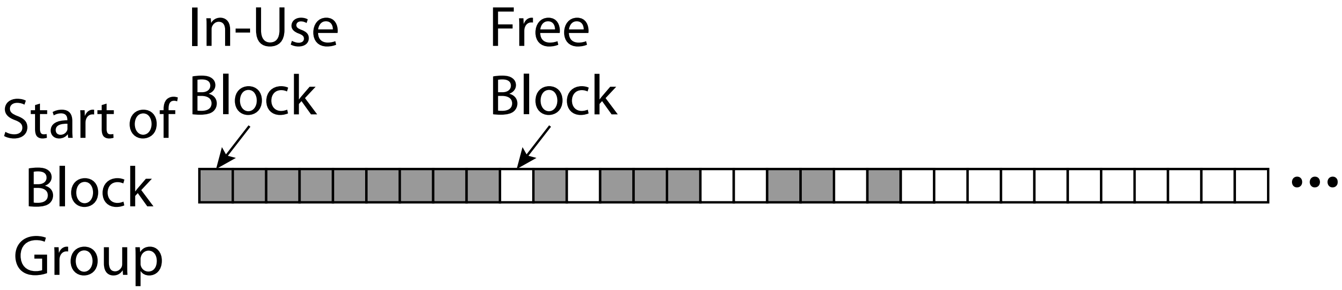
- How does FFS provide locality?
- Block group allocation
  - Block group is a set of nearby cylinders
  - Files in same directory located in same group
  - Subdirectories located in different block groups
- inode table spread throughout disk
  - inodes, bitmap near file blocks
- First fit allocation
  - Property: Small files may be a little fragmented, but large files will be contiguous

Berkeley FFS Locality


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
Inodes
Free Space Bitmap
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

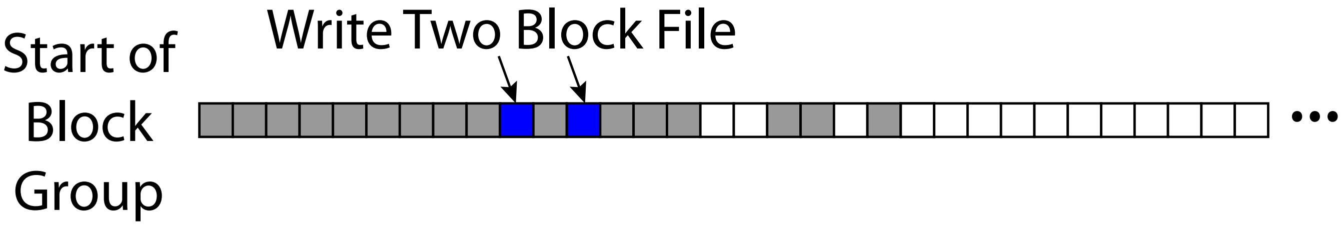
Berkeley FFS Locality

"First Fit" Block Allocation:



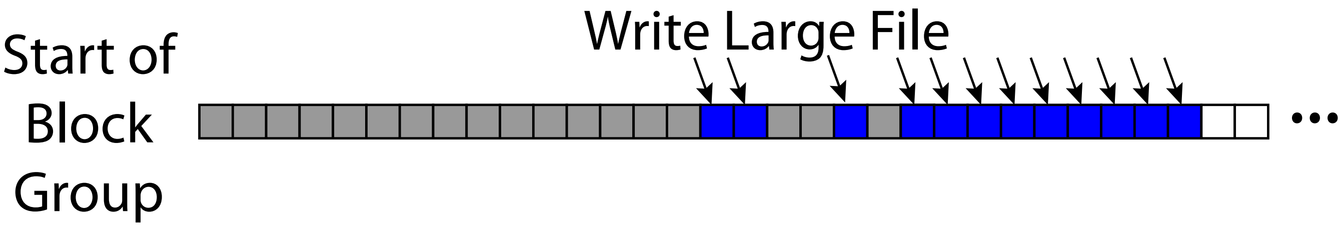
Berkeley FFS Locality

"First Fit" Block Allocation:



Berkeley FFS Locality

"First Fit" Block Allocation:



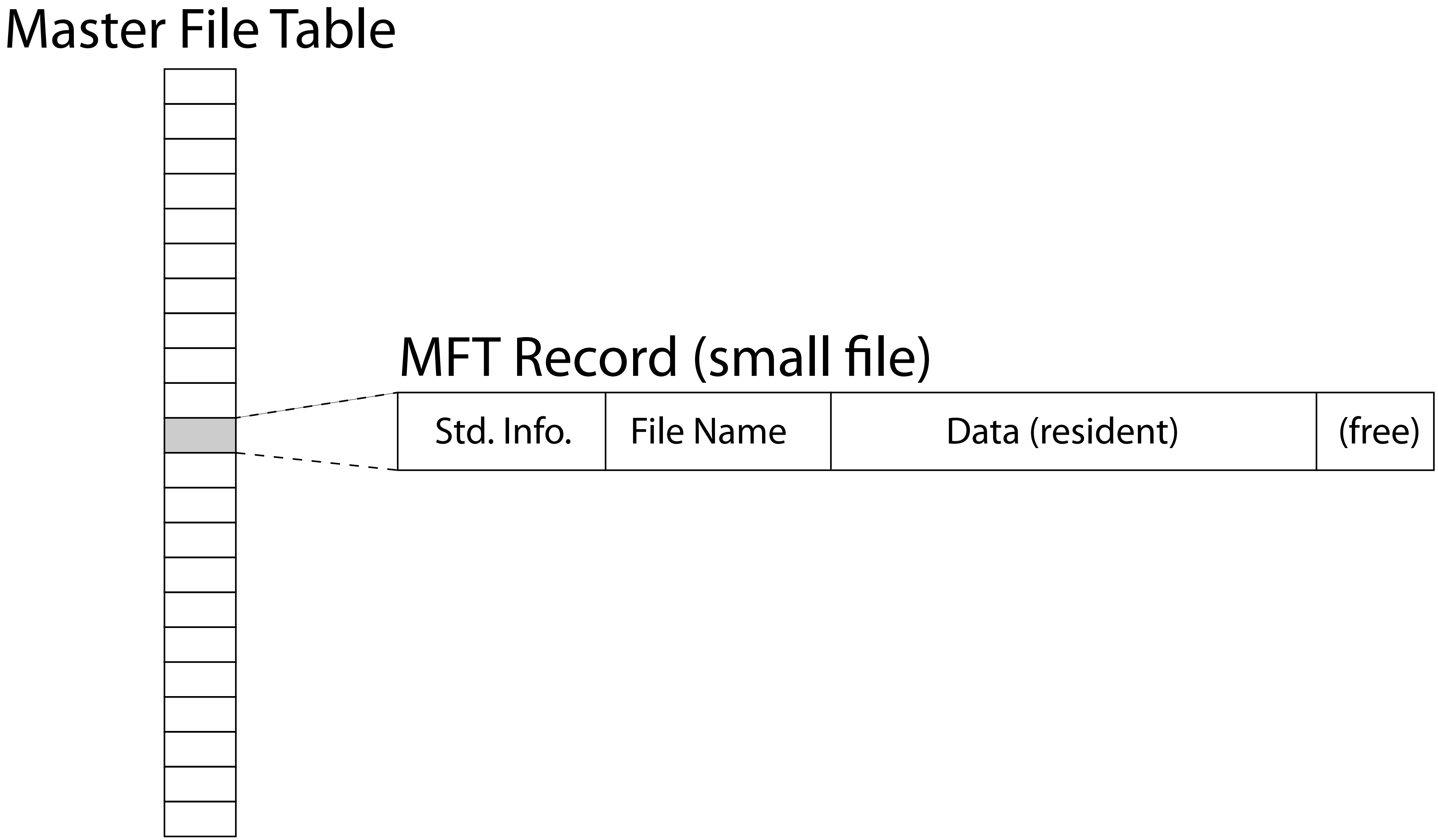
Berkeley FFS / UNIX FS

- 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

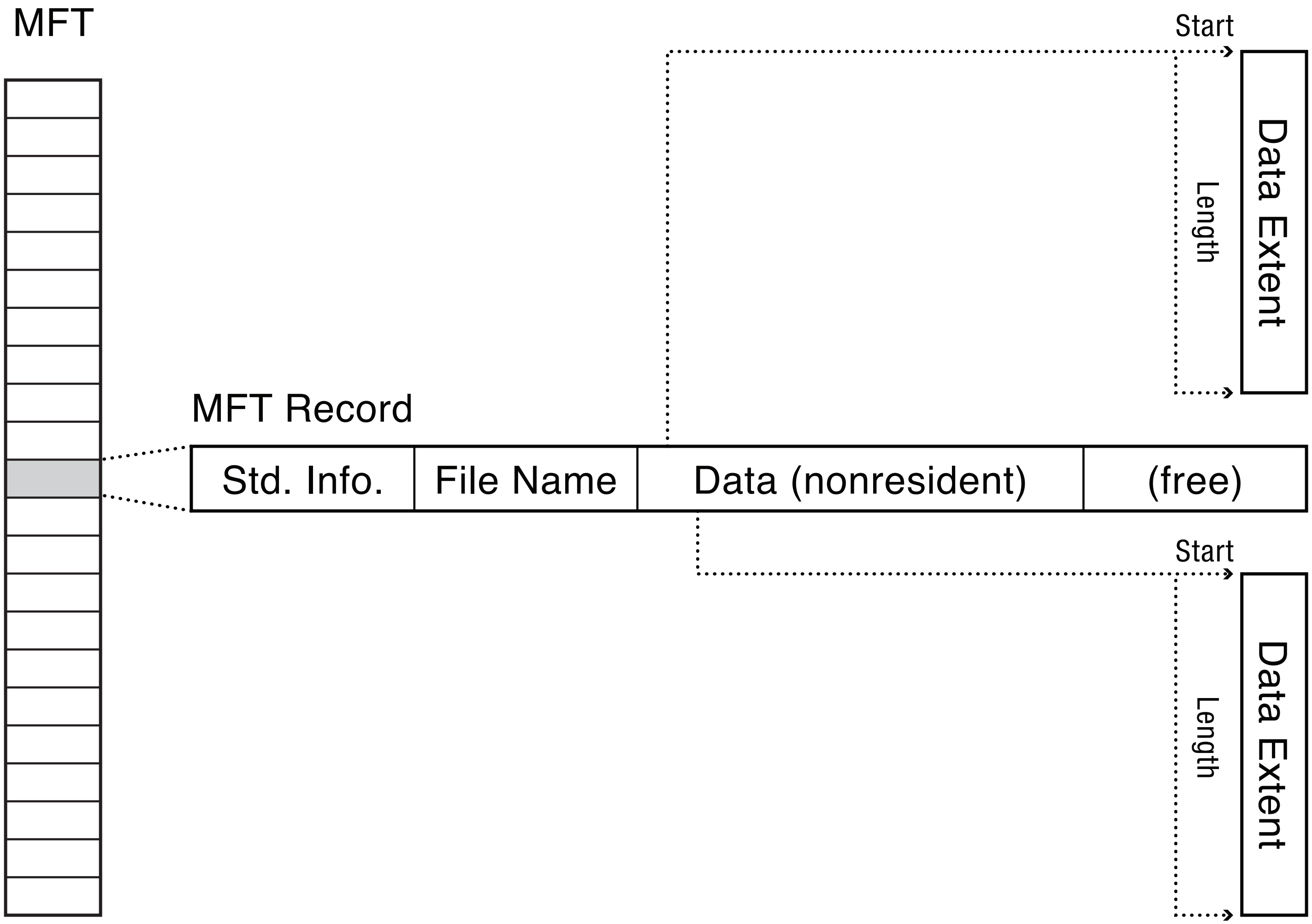
NTFS

- 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

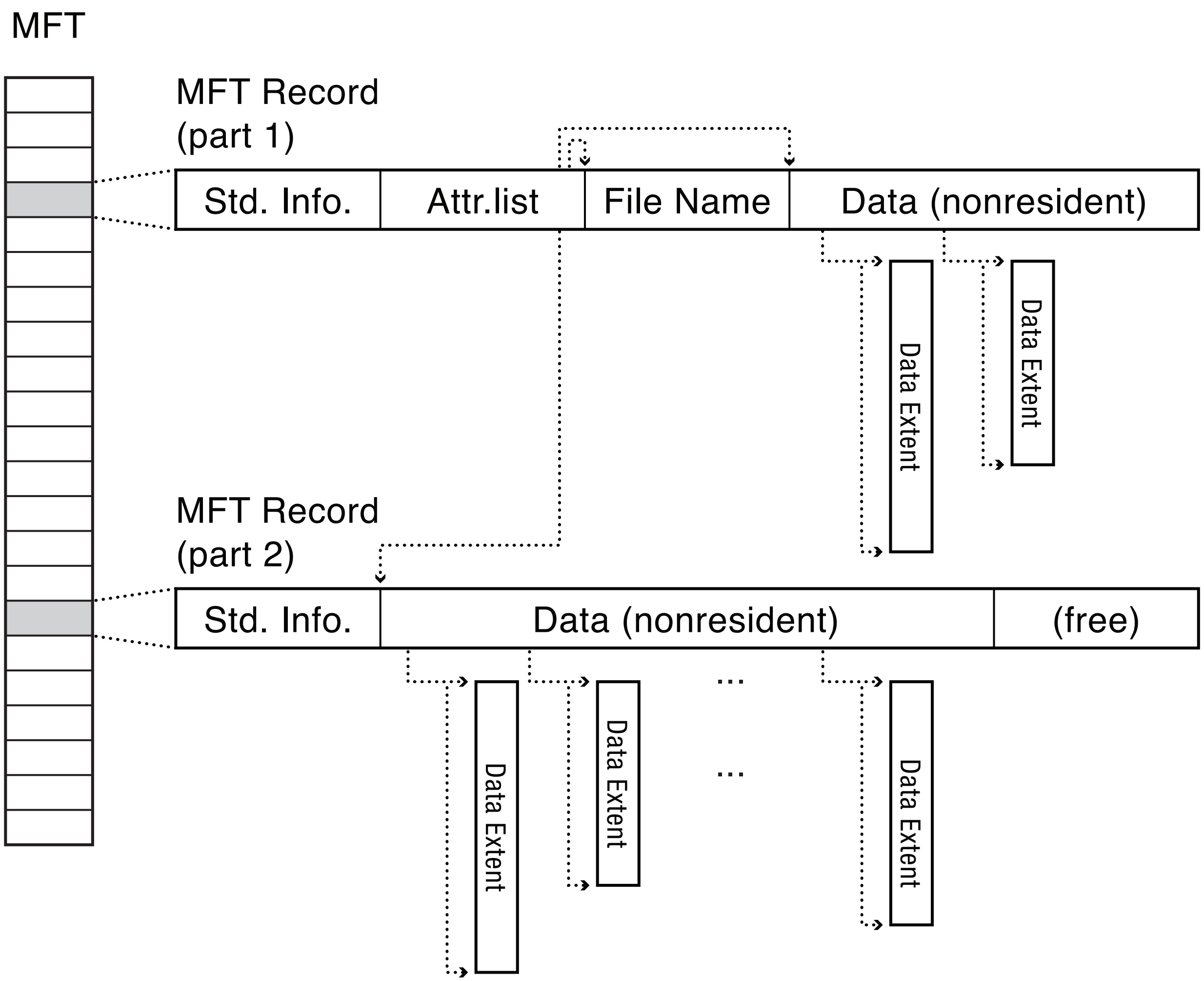
NTFS


Master File Table
MFT Record (small file)
Std. Info.
File Name
Data (resident)
(free)

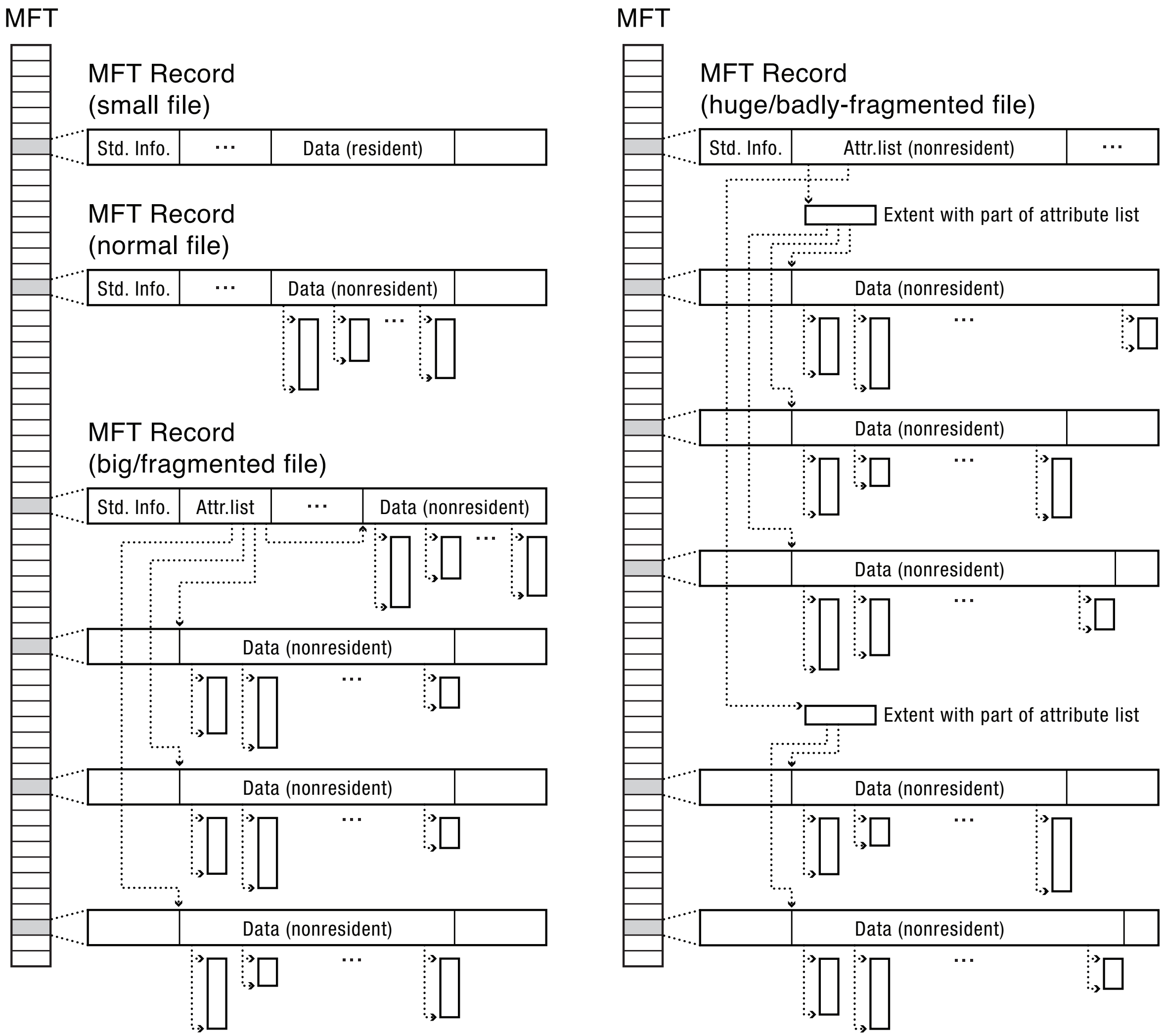
NTFS


MFT
MFT Record
Std. Info.
File Name
Data (nonresident)
(free)
Start
Length
Data Extent
Start
Length
Data Extent

NTFS Indirect Block



NTFS


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)