

CSCI 350:

Introduction to Operating Systems

File System Implementation,
Files and Directories

Professor Tatyana Ryutov

Outline

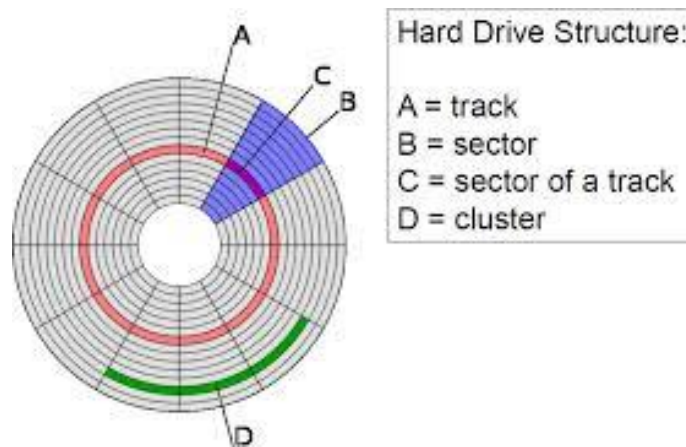
- Disc scheduling
- Flash Storage
- File-System Structure
- On-disk and in-memory structures
- File system design
 - FAT
 - FFS
 - NTFS
- Copy-on-write file systems
- Directories

File System Intro Review

- File: named collection of data
 - Structure
 - Access: sequential, direct, index
- Directory
 - Group of named files or subdirectories
 - Creates the namespace of files, provides persistent, named data
 - Organization: single-level, two-level, tree
 - Operations
- Volume is a collection of physical storage resources that form a **logical** storage device

Disk Management

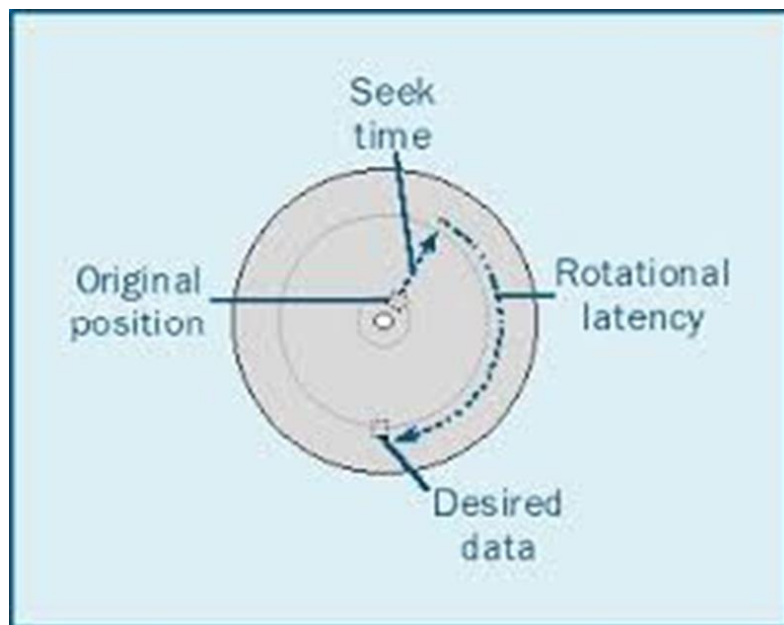
- Low-level (physical) formatting
 - Dividing a disk into sectors that the disk controller can read/write
- To use a disk to hold files, OS needs to record its own data structures on the disk
 1. Partition the disk into one or more groups of cylinders, each treated as a logical disk
 2. Logical formatting or “making a file system”
 - To increase efficiency most file systems group blocks into clusters
 - Disk I/O done in blocks
 - File I/O done in clusters - assuring more sequential-access



Disk Performance

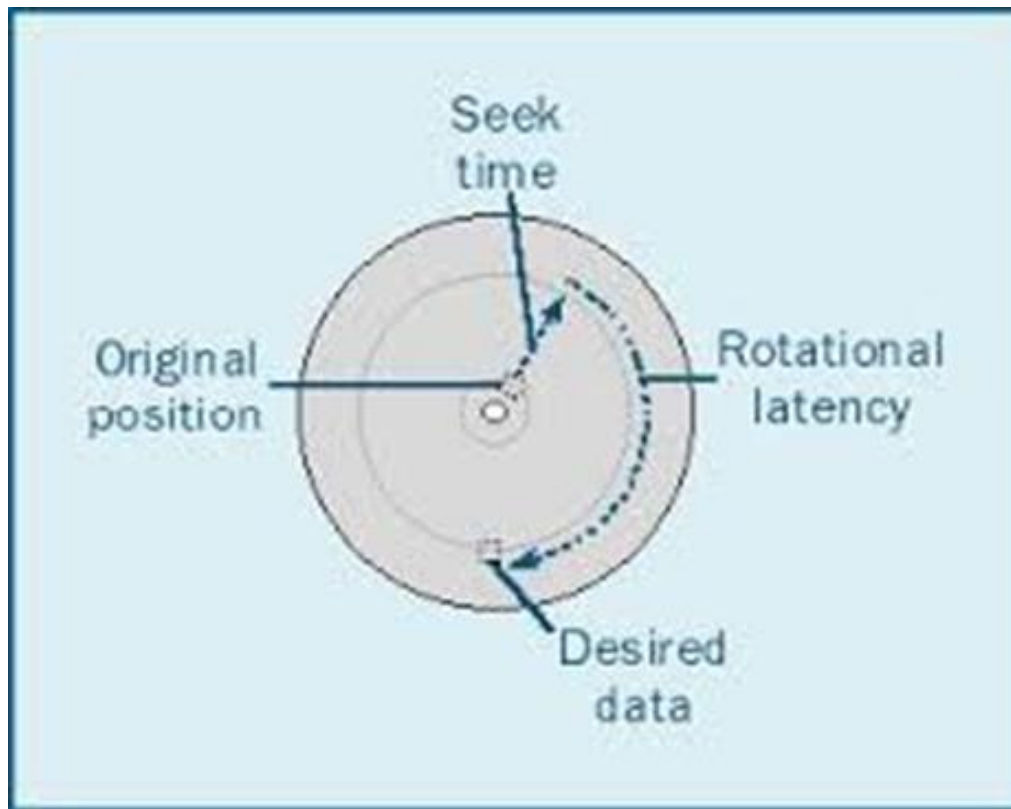
Disk Latency = Seek Time + Rotation Time + Transfer Time

- Seek Time: time to move disk arm over track (1-20ms)
- Rotation Time: time to wait for disk to rotate under head
 - Disk rotation: 4 – 15ms
- Transfer Time: time to transfer data onto/off of disk
 - Disk head transfer rate: 50-100MB/s



Disk Scheduling

- Minimize seek time
 - Seek time \approx seek distance
 - Rotational latency is the additional time for the disk to rotate the desired sector to the disk head



Disk Scheduling 2

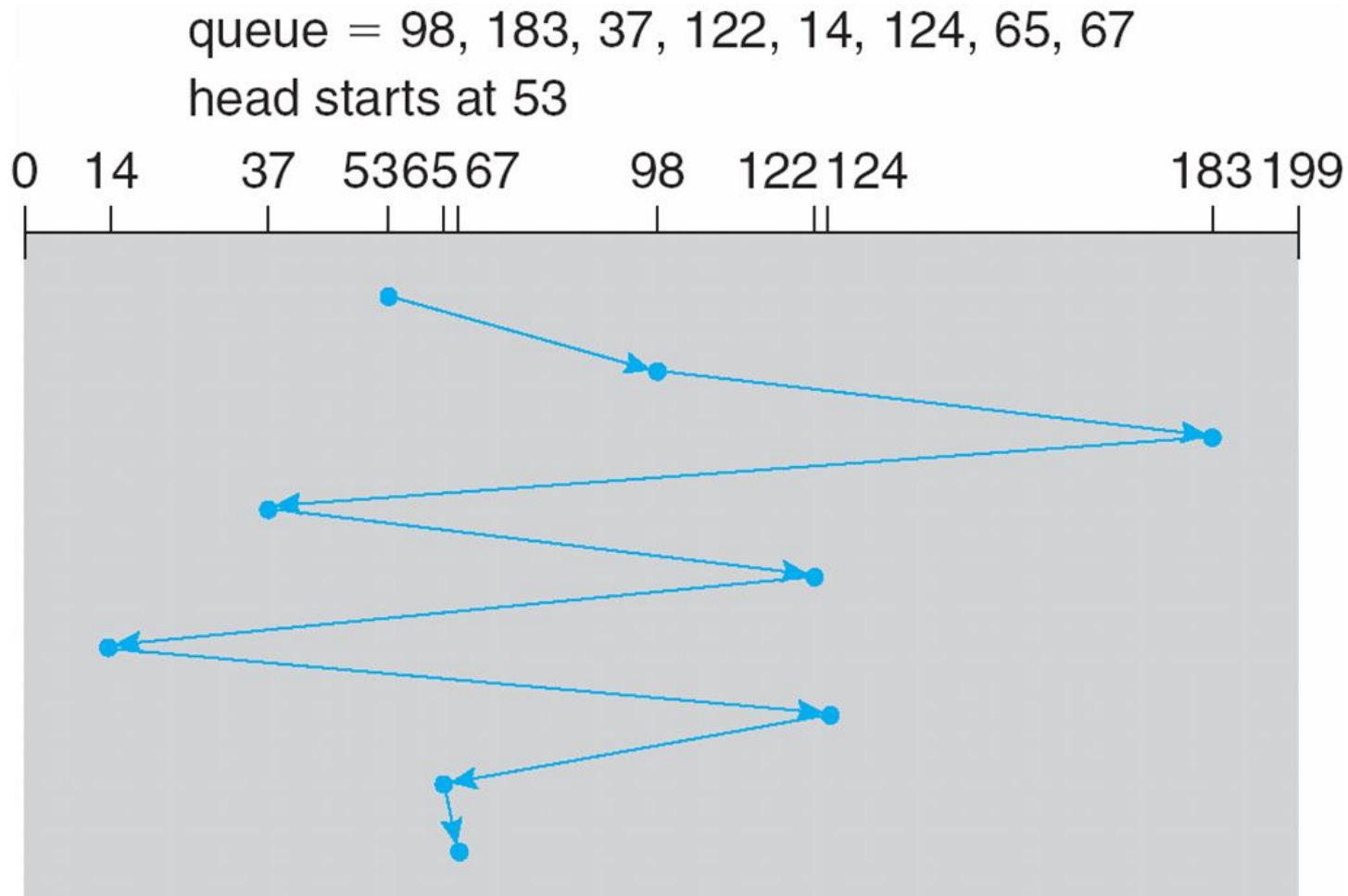
- I/O request includes: input/output mode, disk address, memory address, number of sectors to transfer
- OS maintains queue of requests, per disk or device
- Drive controllers have small buffers and can manage a queue of I/O requests
- Several algorithms exist to schedule the servicing of disk I/O
 - Illustrate scheduling algorithms with a request queue (0-199) to blocks on cylinders:

98, 183, 37, 122, 14, 124, 65, 67

- Head is initially at cylinder 53

FIFO

Illustration shows total head movement of **640** cylinders

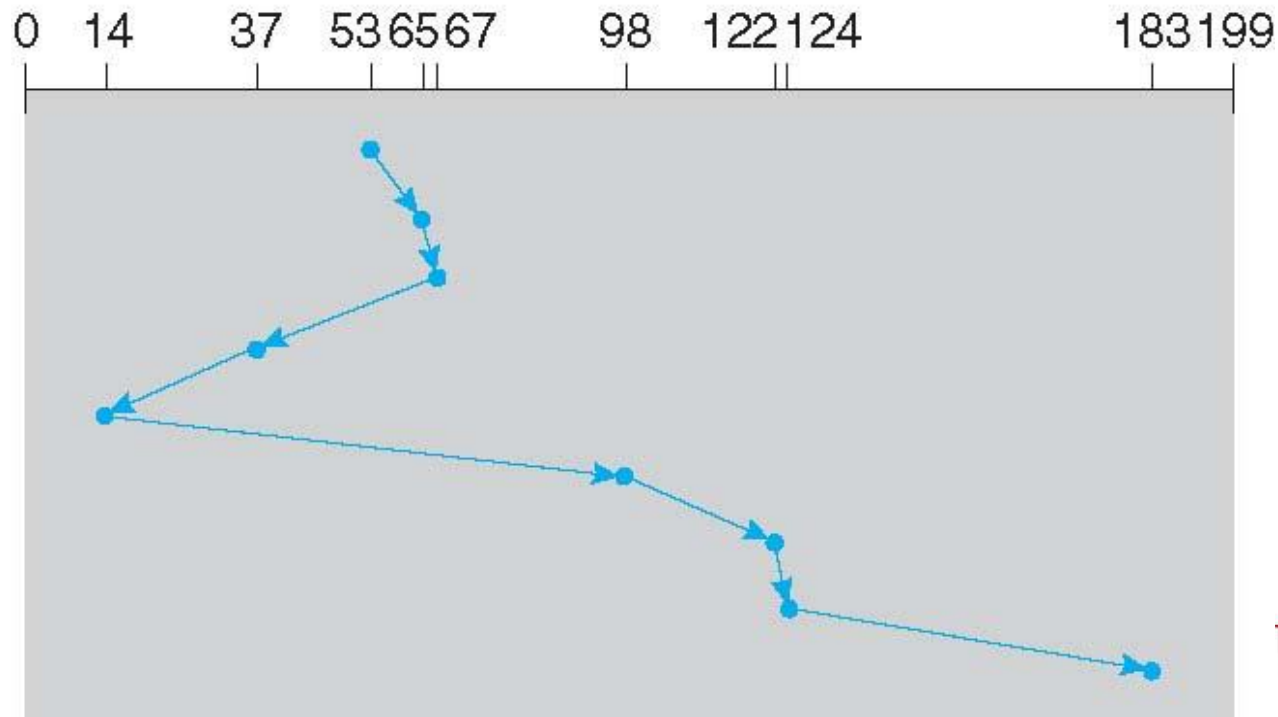


SSTF

- Shortest Seek Time First
 - selects request with the min seek time from the current head position
- Illustration shows total head movement of **236** cylinders:
65, 67, 37, 14, 98, 122, 124, 183

queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53



Disk Scheduling

- FIFO
 - Schedule disk operations in order they arrive
 - Downsides?
- Shortest seek time first (SSTF)
 - Selects the request with the minimum seek time from the current head position
 - SSTF scheduling is a form of SJF scheduling
 - may cause starvation of some requests
 - Not optimal!

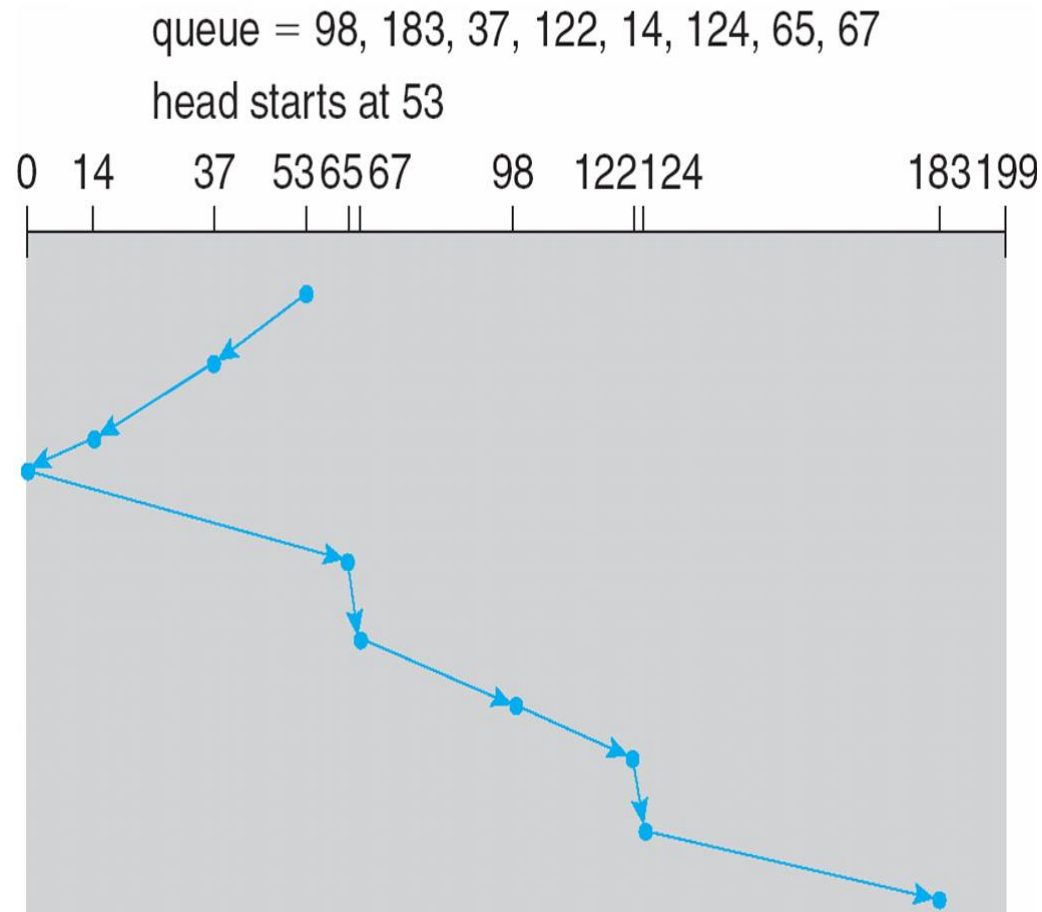
SCAN

- The disk arm starts at one end of the disk, and moves toward the other end
 - servicing requests until it gets to the other end of the disk
- SCAN algorithm
 - Sometimes called the elevator algorithm



SCAN

- Sequence:
37,14,65,67,98,122,124,183
- What if request arrives in the queue just **in front of** the head?
- What if request arrives in the queue just **behind** the head?

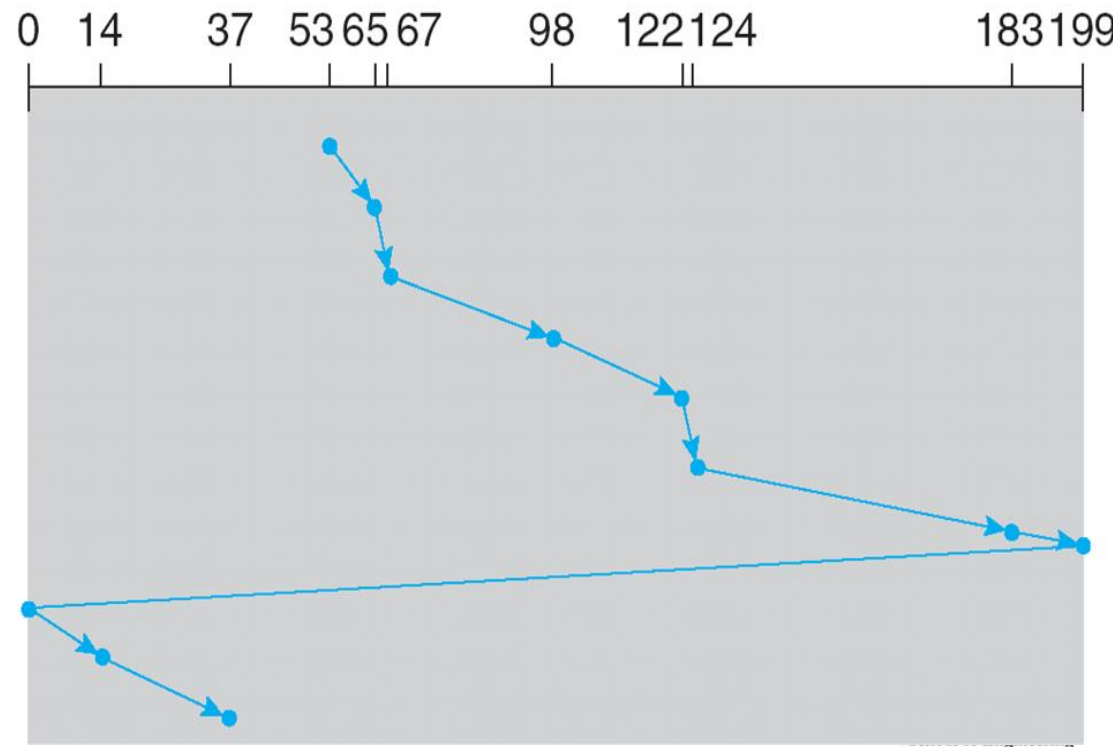


C-SCAN

- Provides a more uniform wait time than SCAN
- Move disk arm in one direction, until all requests satisfied, then start again from farthest request

queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53

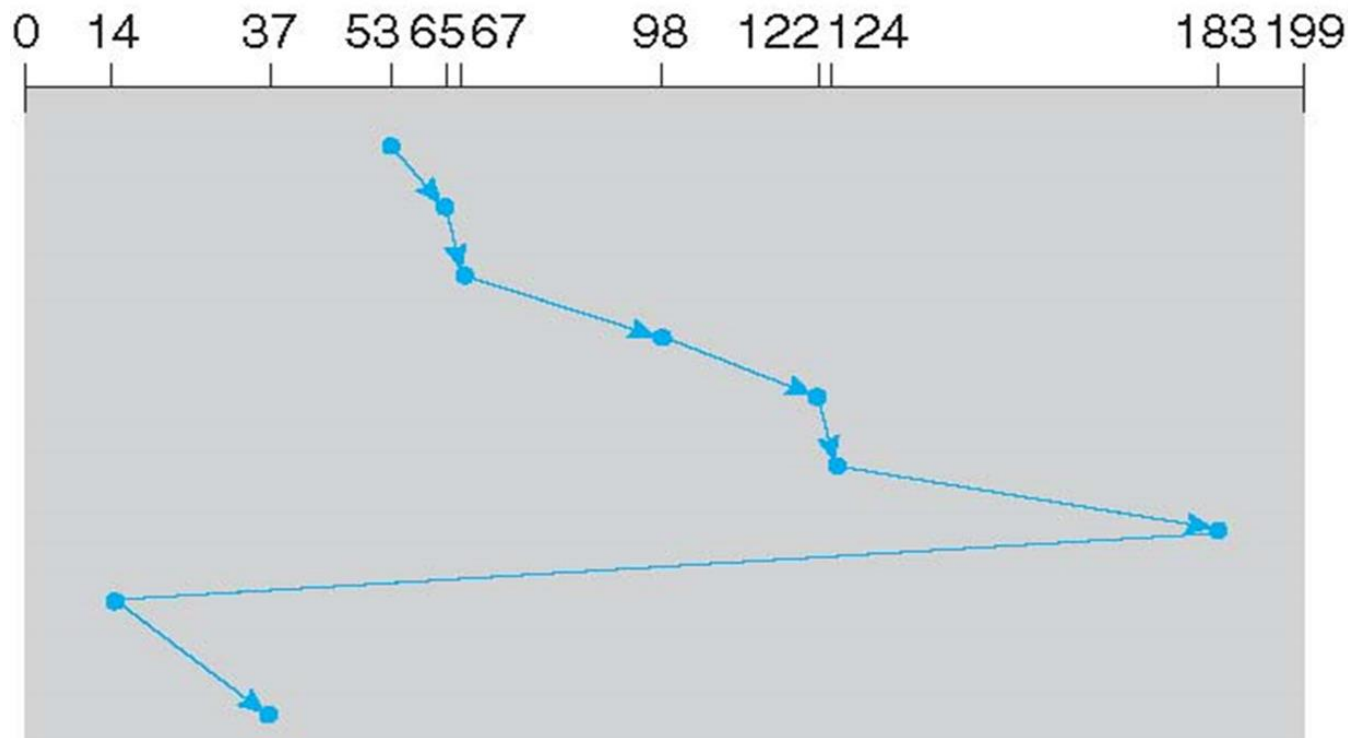


C-LOOK

- Arm only goes as far as the last request in each direction
- Look for a request before continuing to move in a given direction

queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53



Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for heavy load on disk
 - Less starvation
- Requests for disk I/O influenced by the file allocation method
 - Contiguous/linked/indexed
- A separate module of OS, replaceable with a different algorithm
- What about rotational latency?
- How does disk-based queueing effect OS queue ordering efforts?

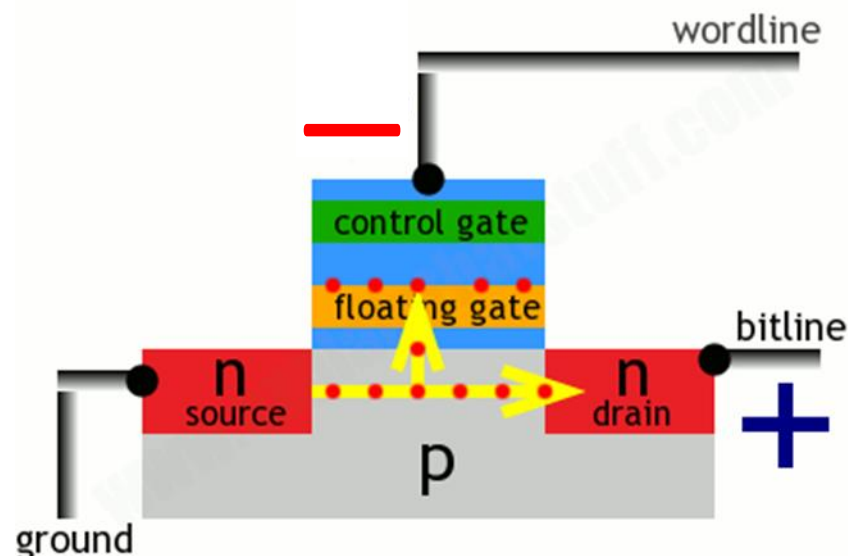
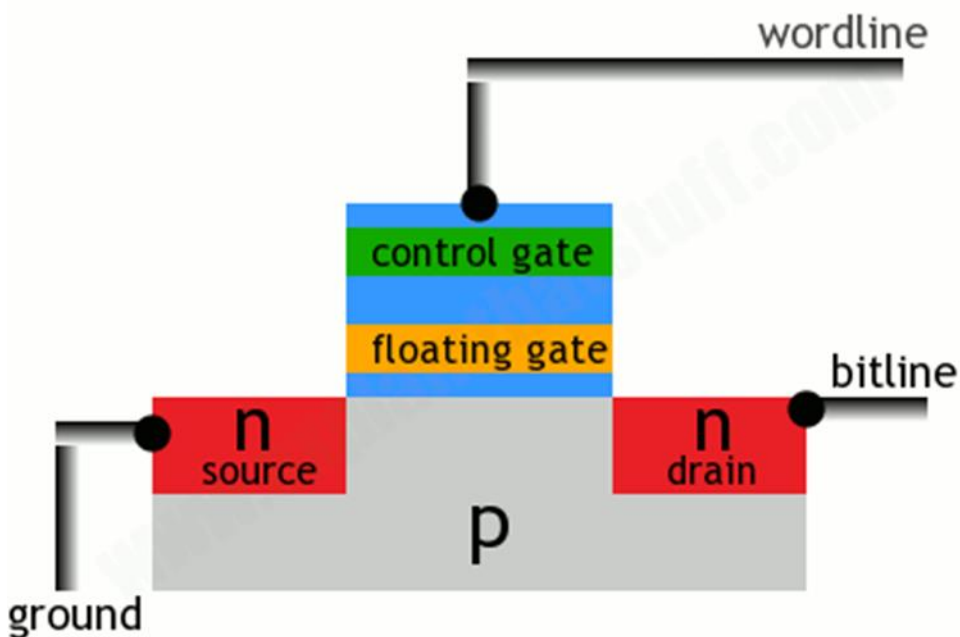
Go until Direction	Go until the last cylinder	Go until the last request
Service both directions	Scan	Look
Service in only one direction	C-Scan	C-Look

Outline

- Disc scheduling
- **Flash Storage**
- File-System Structure
- On-disk and in-memory structures
- File system design
 - FAT
 - FFS
 - NTFS
- Copy-on-write file systems
- Directories

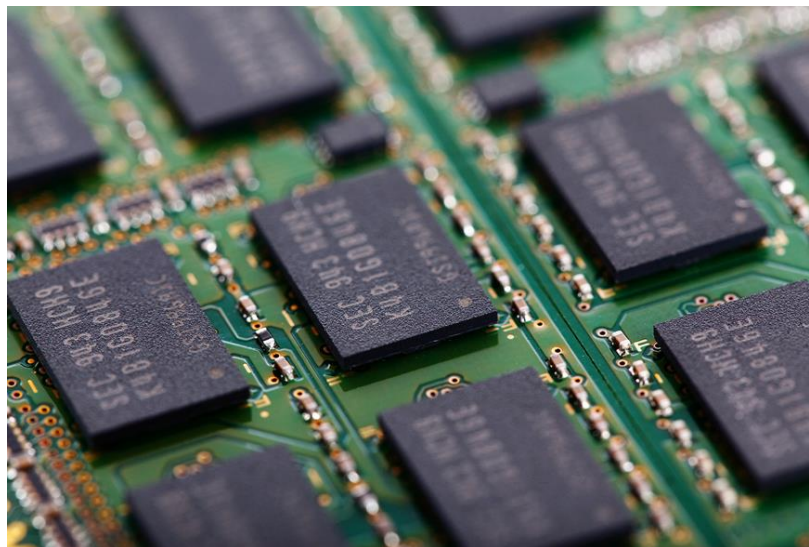
Flash Memory

- Each flash storage element is a floating gate transistor
- The presence of electrons on the floating gate is how a flash transistor stores a one
- The electrons can be flushed out by putting a negative voltage



Flash Memory

- Writes must be to “clean” cells
 - Large block erasure required before write
 - Erasure block: 128 – 512 KB
 - Erasure time: several milliseconds
- Write/read page (2-4KB)
 - 50-100 microseconds
- No updates in place – entire erasure block must be erased

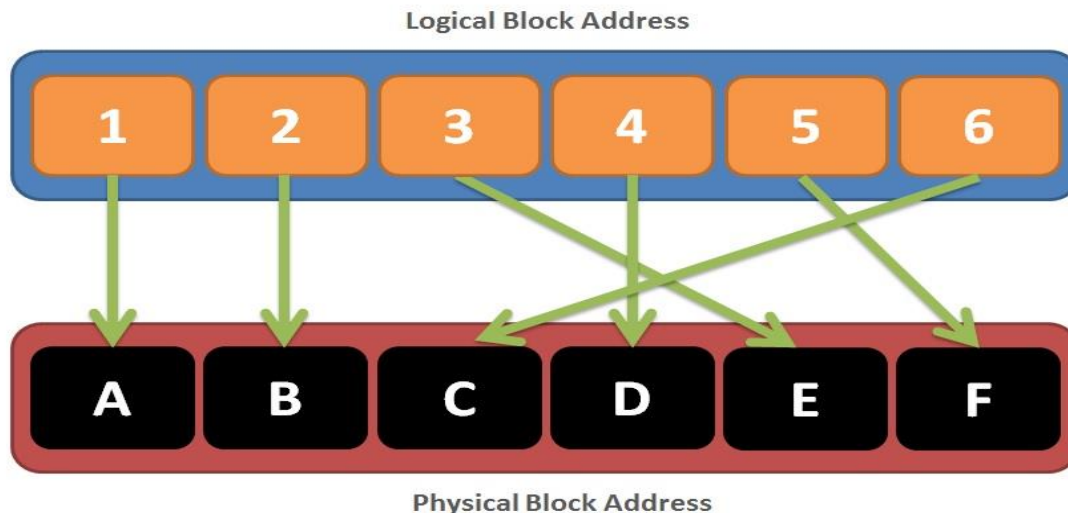


Flash Durability

- Wear out
 - High current loads from flashing and writing memory causes the circuits to degrade
- Read disturb error
 - Memory is read many times without the surrounding memory being written
- Techniques:
 - Error correcting codes
 - Bad page and bad erasure block management
 - Wear leveling (remapping a page)
 - Spare pages and erasure blocks

Flash Translation Layer (FTL)

- Flash device firmware maps logical page # to a physical location
 - Garbage collect erasure block
 - Wear-levelling
 - Remap pages that no longer work (sector sparing)
- Transparent to the device user

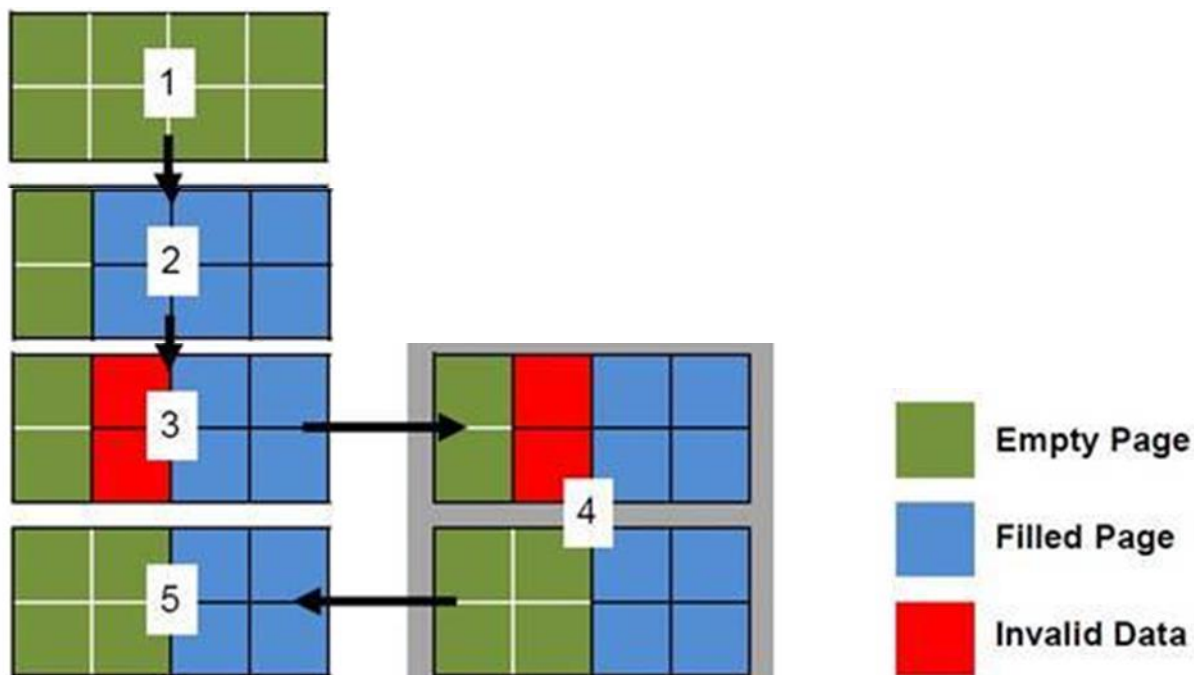


Intel 710 Flash Drive (2011)

Size	
Capacity	300 GB
Page Size	4KB
Performance	
Bandwidth (Sequential Reads)	270 MB/s
Bandwidth (Sequential Writes)	210 MB/s
Read/Write Latency	75 μ s
Random Reads Per Second	38,500
Random Writes Per Second	2,000 (2,400 with 20% space reserve)
Interface	SATA 3 Gb/s
Endurance	
Endurance	1.1 PB (1.5 PB with 20% space reserve)
Power	
Power Consumption Active/Idle	3.7 W / 0.7 W

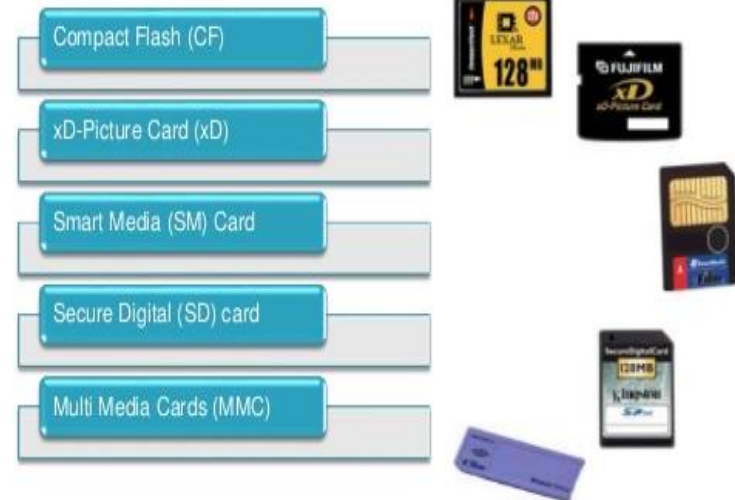
File System – Flash

- How does Flash device know which blocks are live?
 - Live blocks must be remapped to a new location during erasure
- TRIM command
 - File system tells device when blocks are no longer in use
 - reduces garbage collection overheads



Magnetic Disc vs. Flash Storage

- Magnetic disks
 - Storage rarely becomes corrupted
 - Large capacity at low cost
 - Block level random access
 - Slow performance for random access
 - Better performance for streaming access
- Flash memory
 - Storage rarely becomes corrupted
 - Shorter life spans
 - Capacity at intermediate cost
 - Block level random access
 - Good performance for random reads
 - worse for random writes
 - Lower power consumption
 - Smaller size and weight



Technology Trends

- New Technologies

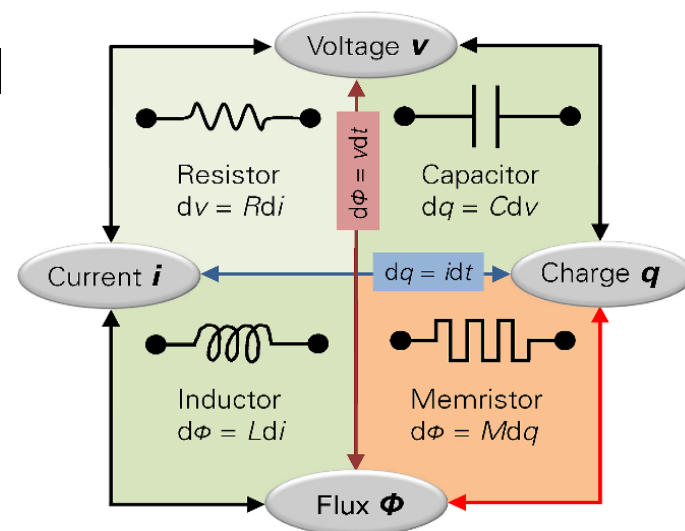
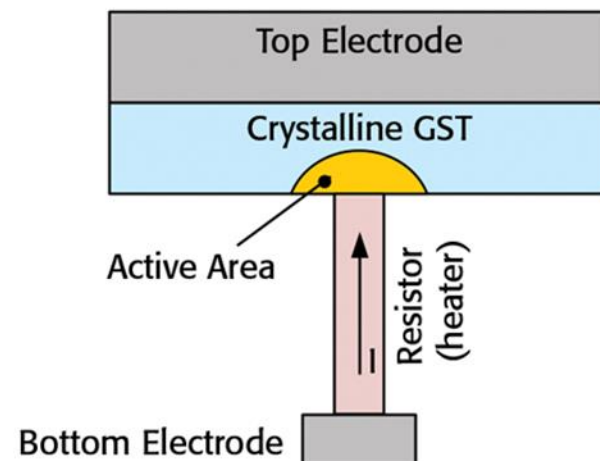
- Phase Change Memory (PCM)

- Current alters the state of chalcogenide glass between amorphous and crystalline forms; diff electrical resistance

- Memristor

- Resistance depends on the amounts and directions of currents that have flowed through it in the past

- Reliance on directories for naming and locality can change in coming years



Outline

- Disc scheduling
- Flash Storage
- **File-System Structure**
- On-disk and in-memory structures
- File system design
 - FAT
 - FFS
 - NTFS
- Copy-on-write file systems
- Directories

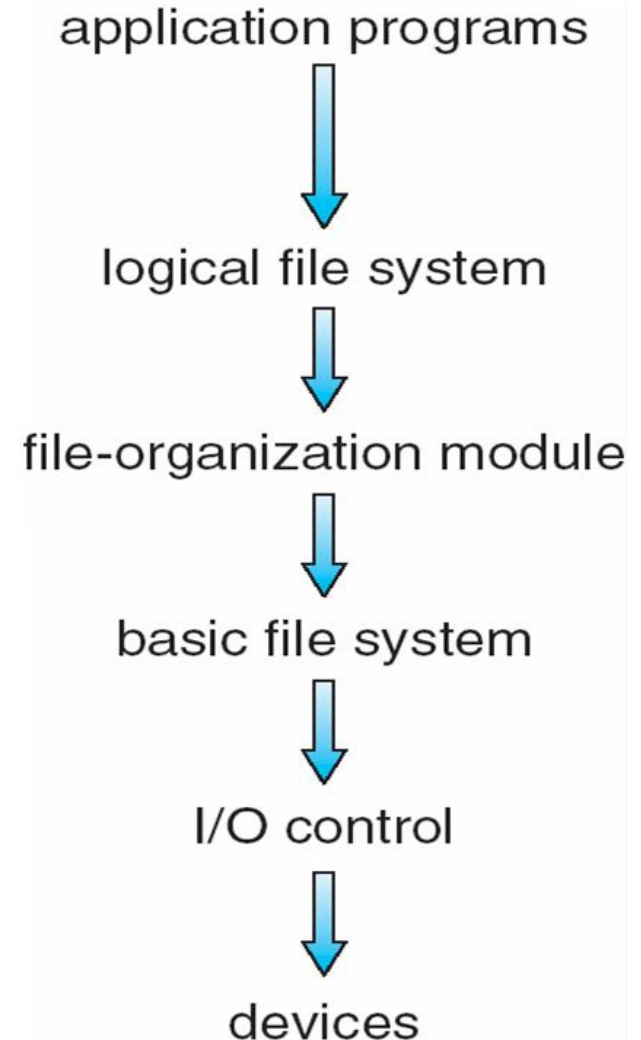
File-System Structure

- File structure
 - Logical storage unit
 - Collection of related information
 - User vs. algorithm/data structures
- File system resides on secondary storage
- File system organized into layers
- File control block – storage structure contains info about a file

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

Layered File System

- Application programs
- Logical file system
 - Manages the directory structure
 - Maintains the file structure via file control blocks
- File-organization module
 - Maps the logical blocks to the physical blocks
 - Manages the free space/blocks on the disk
- Basic file system
 - Issues generic commands to appropriate device driver to read/write physical blocks on the disk
- I/O control
 - Consists of device drivers and interrupt handlers
 - Translates high-level commands to HW-specific



Layered File System 2

Advantage of a layered structure:

- Duplication of code is minimized
- Each FS can have its own logical and file-organization modules

Disadvantages?

- Many kinds of FSs are in use today
 - UNIX uses the UNIX file system (UFS)
 - Windows NT, 2000, and XP support
 - FAT, FAT32, and NTFS
 - Linux supports over 40 different file systems
 - Distributed FSs
 - Research in FS, e.g., FUSE

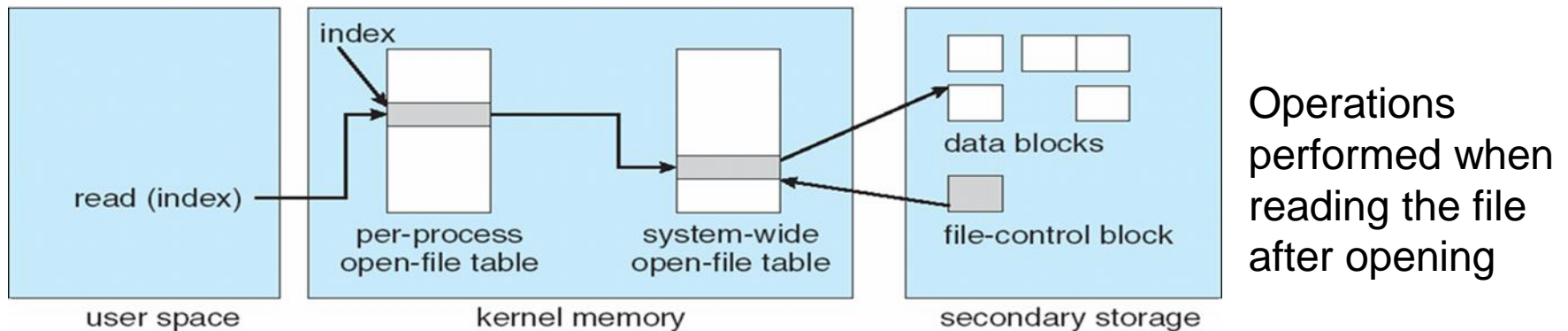
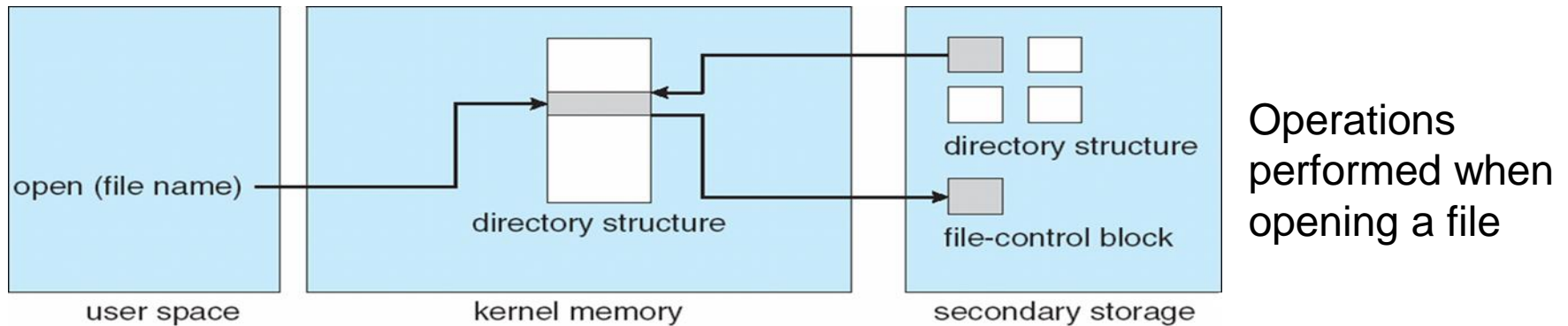
Outline

- Disc scheduling
- Flash Storage
- File-System Structure
- On-disk and in-memory structures
- File system design
 - FAT
 - FFS
 - NTFS
- Copy-on-write file systems
- Directories

FS Structures

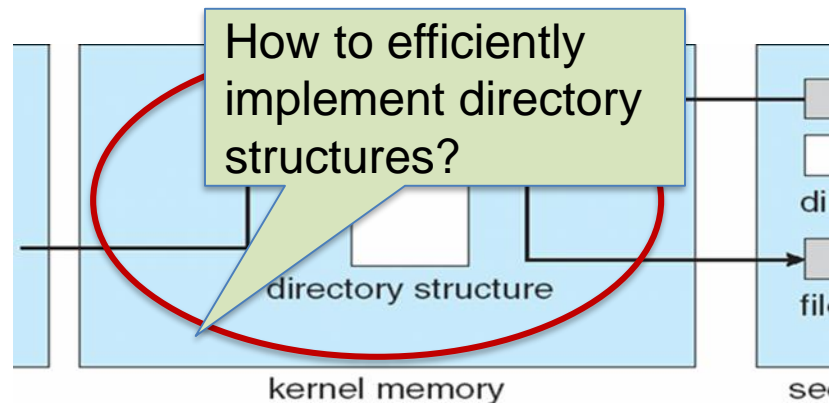
- On-disk structures:
 - Boot control block (per volume)
 - Volume control block (per volume)
 - The total number of blocks
 - The number and location of free blocks
 - The directory structure per FS
 - Per-file File Control Block
 - Individual files
- In-memory structures:
 - In-memory mount table
 - In-memory directory-structure cache
 - System-wide open-file table
 - Per-process open-file table
 - Loaded at mount time and discarded at dismount

In-Memory FS Structures



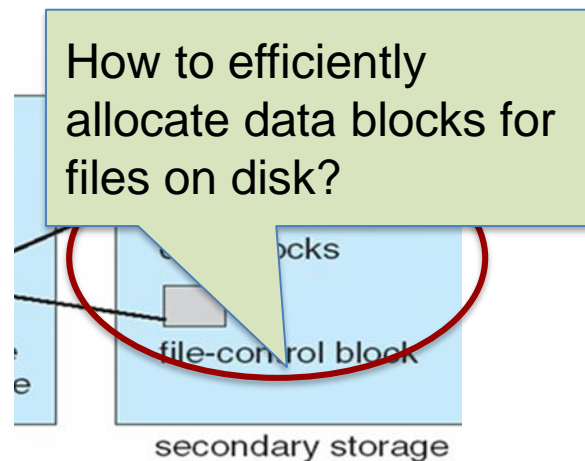
Directory Implementation

- Linear list of file names with pointer to the data blocks
 - simple to program
 - time-consuming to execute
- Hash Table – linear list with hash data structure
 - decreases directory search time
 - collisions
 - fixed size
- Non-linear structure
 - E.g., a B-tree

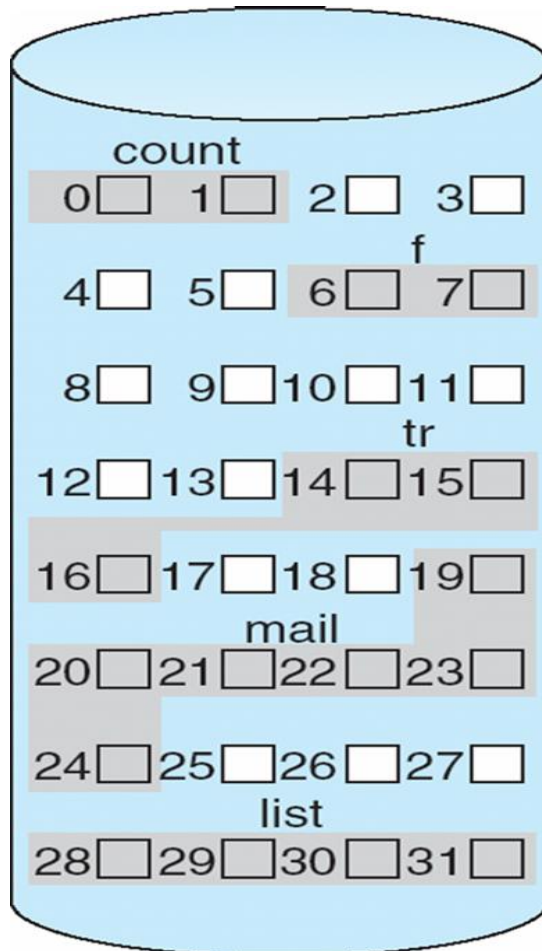


Allocation Methods

- An allocation method refers to how disk blocks are allocated for files
 - Contiguous allocation
 - Linked allocation
 - Indexed allocation
- Typically one method is used for all files within FS type





Contiguous Allocation of Disk Space



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Contiguous Allocation

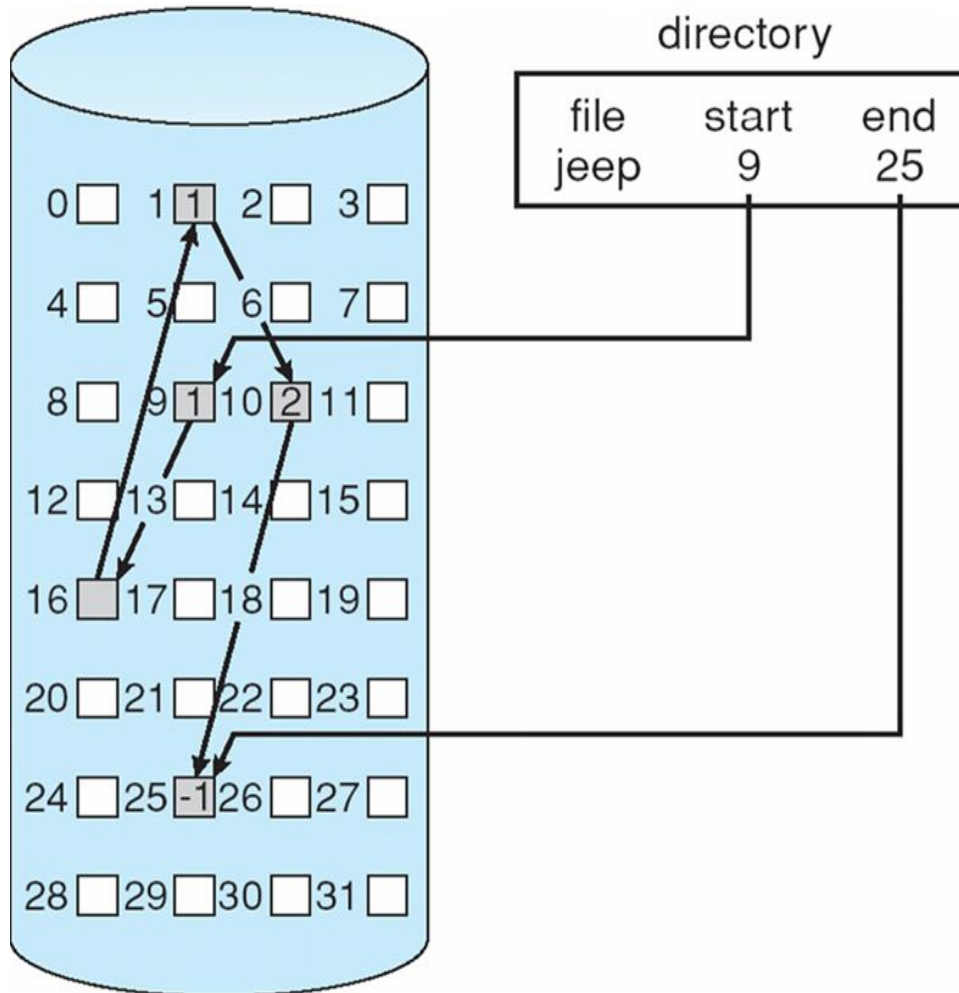
- Pros: 
 - Simple: state required per file is start block and size
 - Performance: entire file can be read with one seek
- Cons: 
 - Finding space for a new file (first fit, best fit, etc.)
 - Determining space for a file, especially if it needs to grow
 - External fragmentation
 - Need for compaction, which may require FS down time
- Used in CDRoms, DVDs

Extent-Based Systems

- Many newer file systems (i.e. Veritas File System) use a modified contiguous allocation scheme
- Extent-based file systems allocate disk blocks in **extents**
- An **extent** is a contiguous chunk of blocks
 - A file consists of one or more extents
 - location and a block count, plus a link to the first block of the next extent
- Internal and external fragmentation

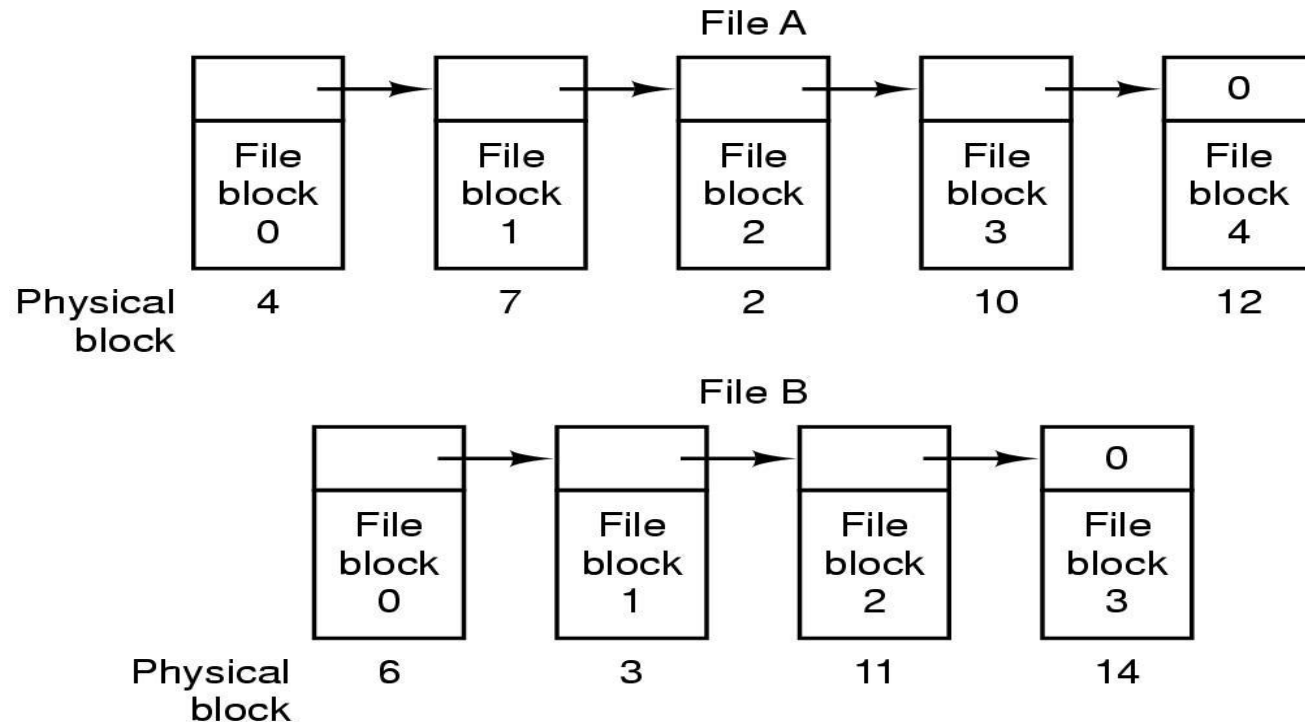
Linked Allocation

- How to implement?





Linked List Allocation

- Each file is stored as linked list of blocks
 - First word of each block points to next block
 - The rest of disk block is file data

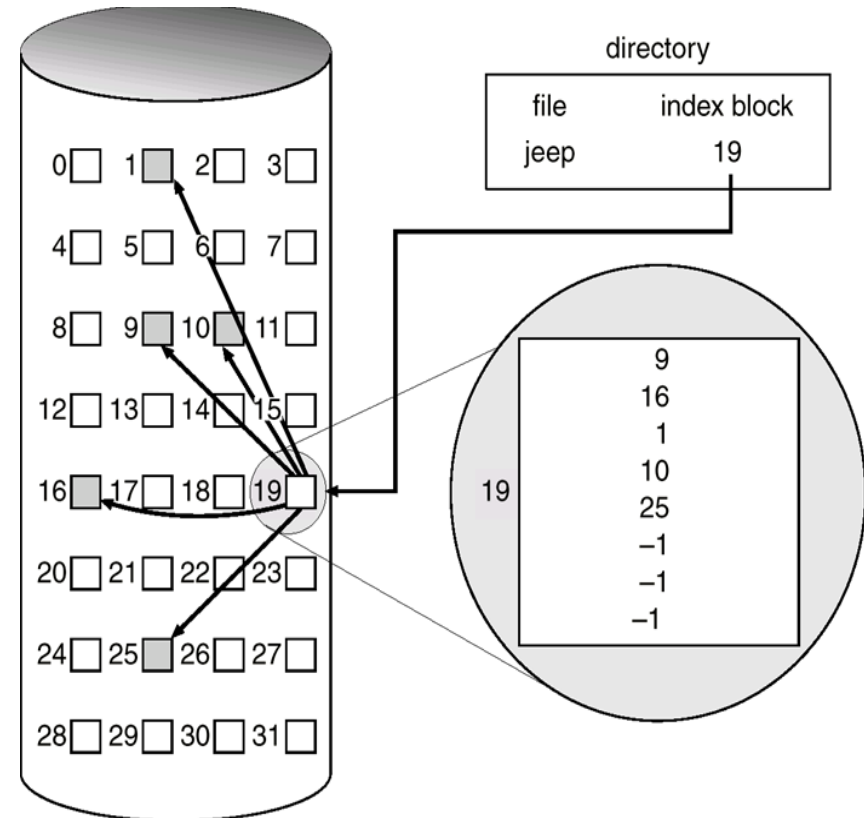


Linked List Allocation

- Pros: 
 - No space lost to external fragmentation, no compacting
 - Disk only needs to maintain first block of each file
 - A file can continue to grow if free blocks are available
- Cons: 
 - Random access is costly
 - Overheads of pointers
 - Relies on the integrity of the links

Example of Indexed Allocation

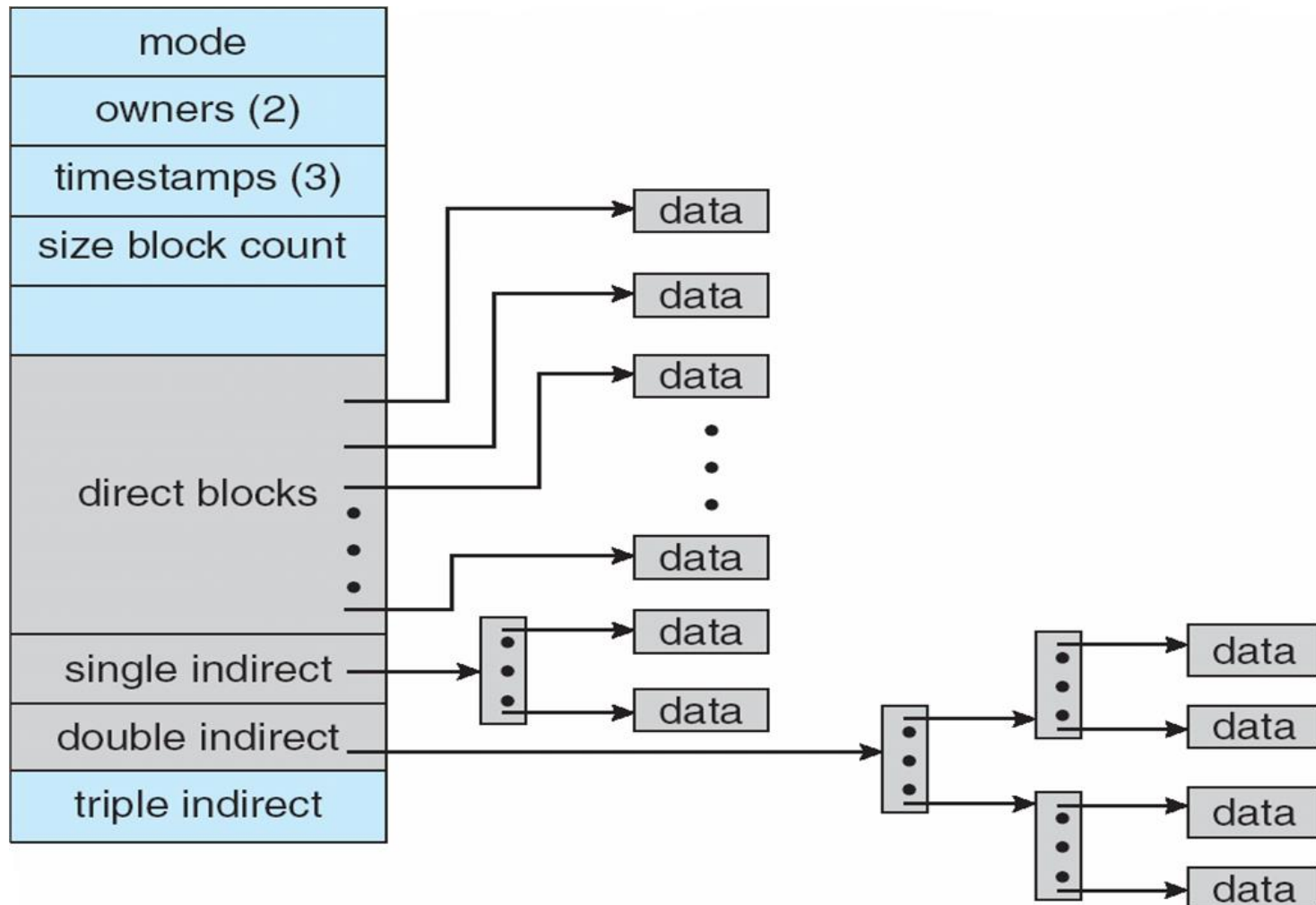
- Brings all pointers together into the **index block**
- Pros: 👍
 - Efficient random access
 - Dynamic access without external fragmentation
- Cons: 👎
 - Index table storage overhead



Sizing of the Index Block

- How large the index block should be?
- Linked scheme
 - Several index blocks can be linked together
- Multilevel index scheme
 - A first-level index block points to a set of second-level index blocks
- Combined scheme
 - A specific set of pointers points directly to file blocks
 - Special pointers point to a single indirect block, a double indirect block, and a triple indirect block
 - Used in the UNIX file system (UFS)

Combined Scheme



Free-Space Management

- Bit vector (n blocks)
 - Each disk block is represented by 1 bit



$$\text{bit}[i] = \begin{cases} 0 \Rightarrow \text{block}[i] \text{ free} \\ 1 \Rightarrow \text{block}[i] \text{ occupied} \end{cases}$$

- Easy to find space to allocate contiguous files
- Efficient only if the entire vector is kept in main memory

Linked Free Space List on Disk

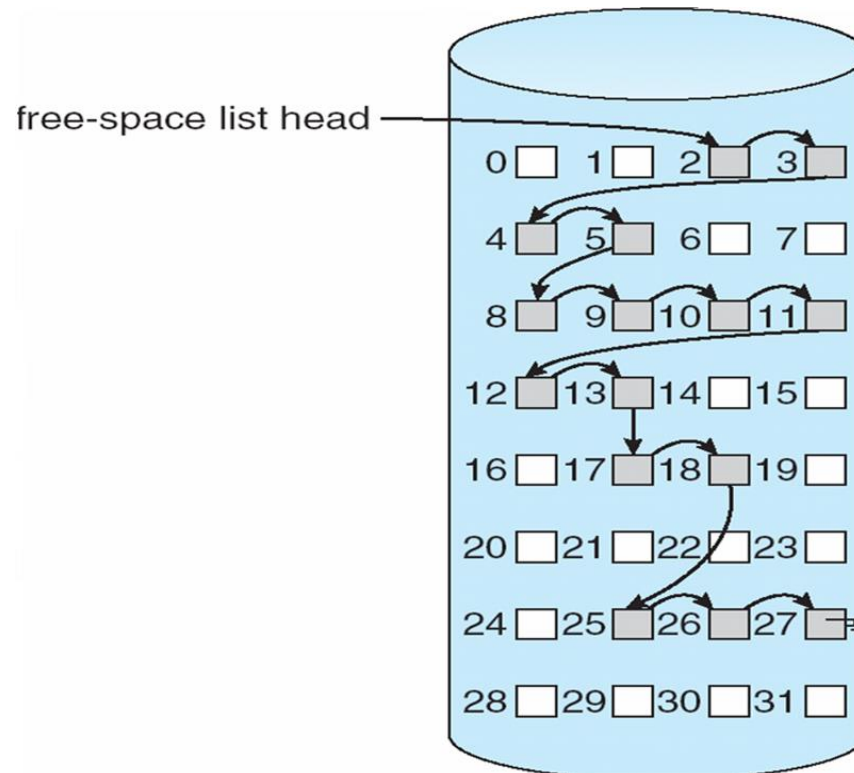
- Link together all the free disk blocks
 - keep a pointer to the first free block in a special location on the disk and cache it in memory



Pros: no waste of space

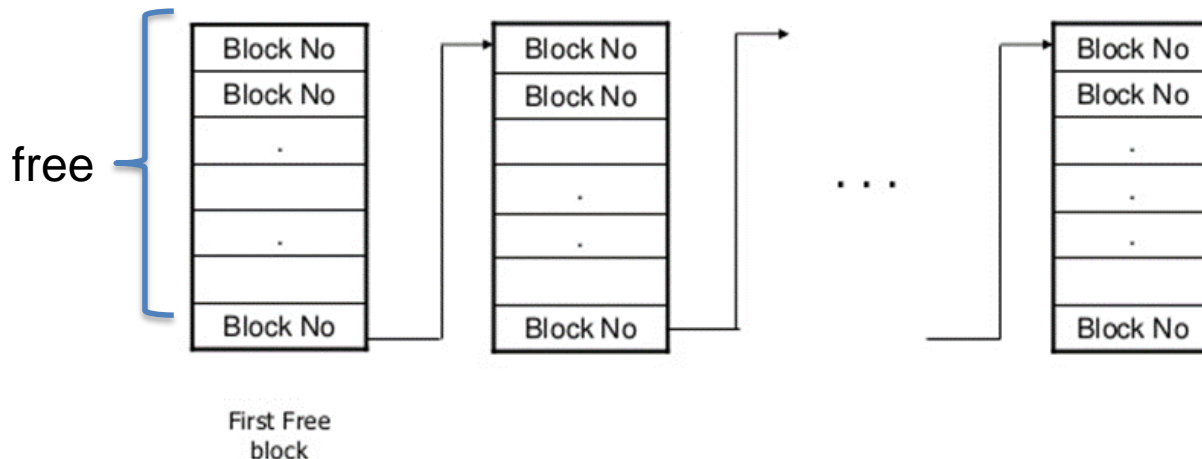


Cons: to get contiguous blocks for a file, traverse the list



Free-Space Management Approaches

- Grouping (modification of free list approach)
 - Store the addresses of n free blocks in the first free block
 - The first $n-1$ of these blocks are actually free
 - The last block contains the addresses of another n free blocks
- Counting
 - Observation: Several contiguous blocks may be allocated or freed simultaneously
 - Keep the address of the first free block and the number n of free contiguous blocks that follow the first block (count)
 - Generally shorter list vs. extra space for count

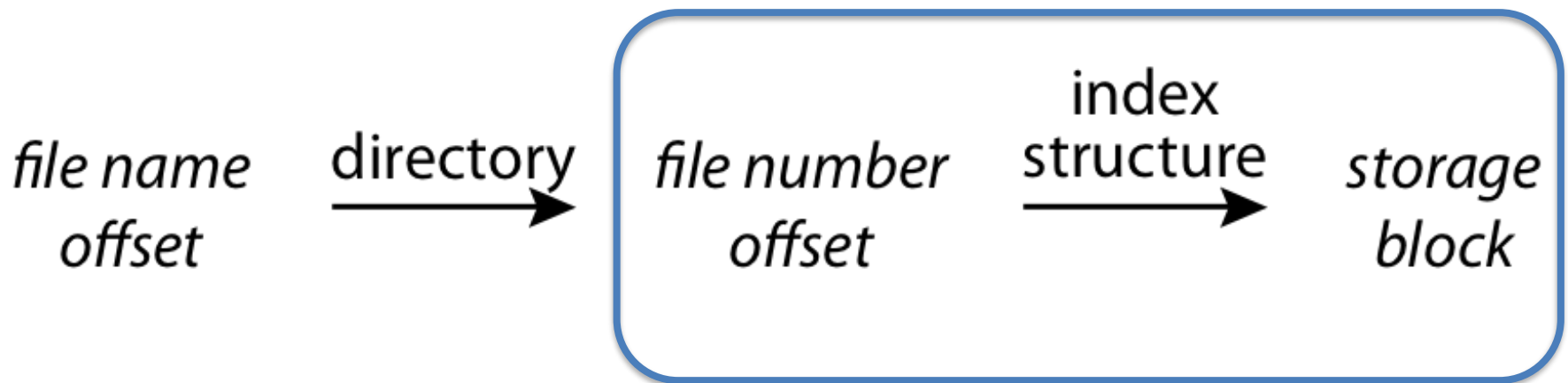


Outline

- Disc scheduling
- Flash Storage
- File-System Structure
- On-disk and in-memory structures
- **File system design**
 - FAT
 - FFS
 - NTFS
- Copy-on-write file systems
- Directories

Named Data in a File System

- How do we go from a file name and offset to a block number?
- Just a simple map?
 - How about performance/flexibility/persistence?



Key ideas: directories, index structures, free space maps, and locality heuristics

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?
 - find free blocks near a desired location
- Locality
 - How do we preserve spatial locality?
- Reliability
 - What if machine crashes in the middle of a FS operation?

File System Design Options

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

Outline

- Disc scheduling
- Flash Storage
- File-System Structure
- On-disk and in-memory structures
- **File system design**
 - FAT
 - FFS
 - NTFS
- Copy-on-write file systems
- Directories

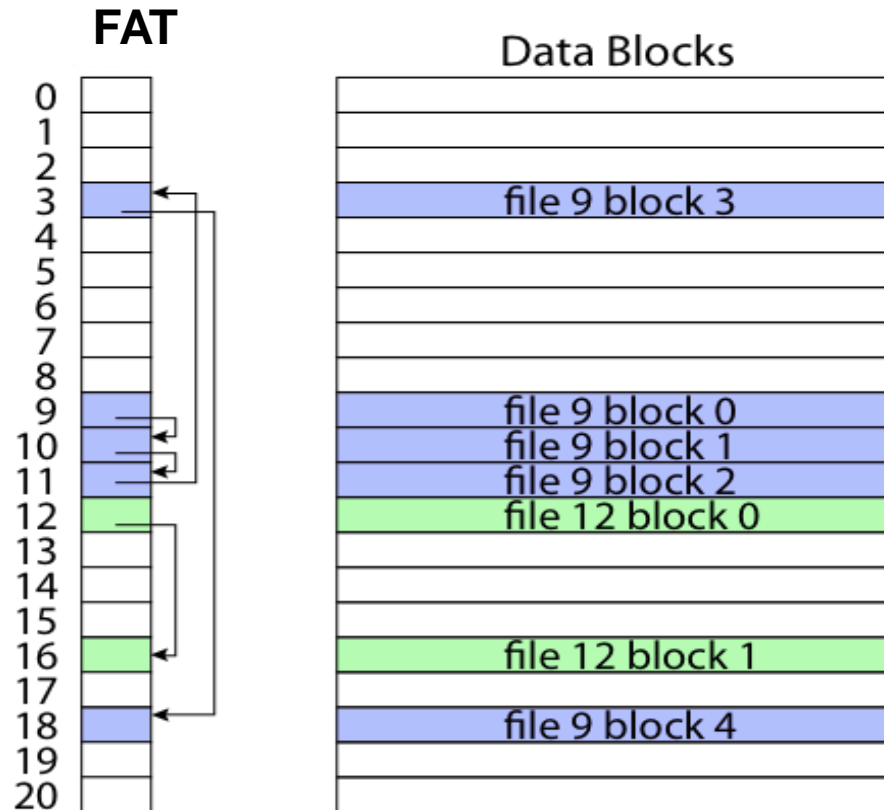
Microsoft File Allocation Table (FAT)

- Implemented in 1970s, was the main file system for MS-DOS and early versions of MS Windows
- 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 is a linked list of blocks
- Locality:
 - Allocation is simple: next fit algorithm
 - results in file fragmentation → poor locality





FAT

File's number is the index of the file's first entry in the FAT



FAT is also used for
free space tracking

FAT

- Pros: 
 - Easy to find free block
 - Easy to append to a file
 - Easy to delete a file
- Cons: 
 - Poor locality
 - 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
 - Limited metadata
 - No support for: reliability, hard links

Outline

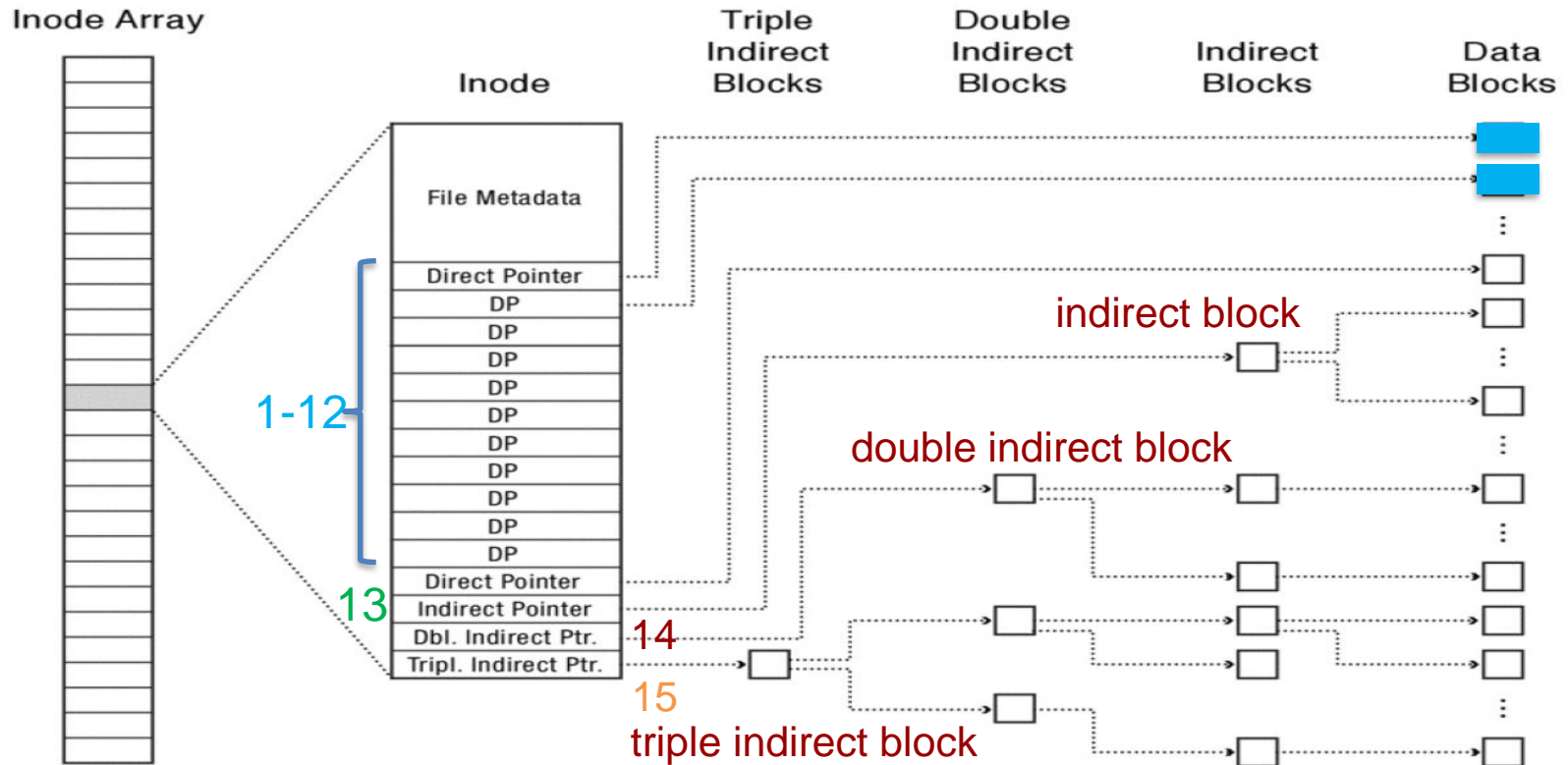
- Disc scheduling
- Flash Storage
- File-System Structure
- On-disk and in-memory structures
- **File system design**
 - FAT
 - **FFS**
 - NTFS
- Copy-on-write file systems
- Directories

UNIX FFS (Fast File System)

- Multi-level index structure
- Locality heuristics: block group placement and reserve space
- inode table (inode array)
 - Analogous to FAT table
- inode (index node) stores both metadata and the pointers to disk blocks
 - FFS inode is the root of an asymmetric tree whose leaves are the data blocks of a file

FFS: inode

FFS uses fixed, asymmetric tree called a multi-level index

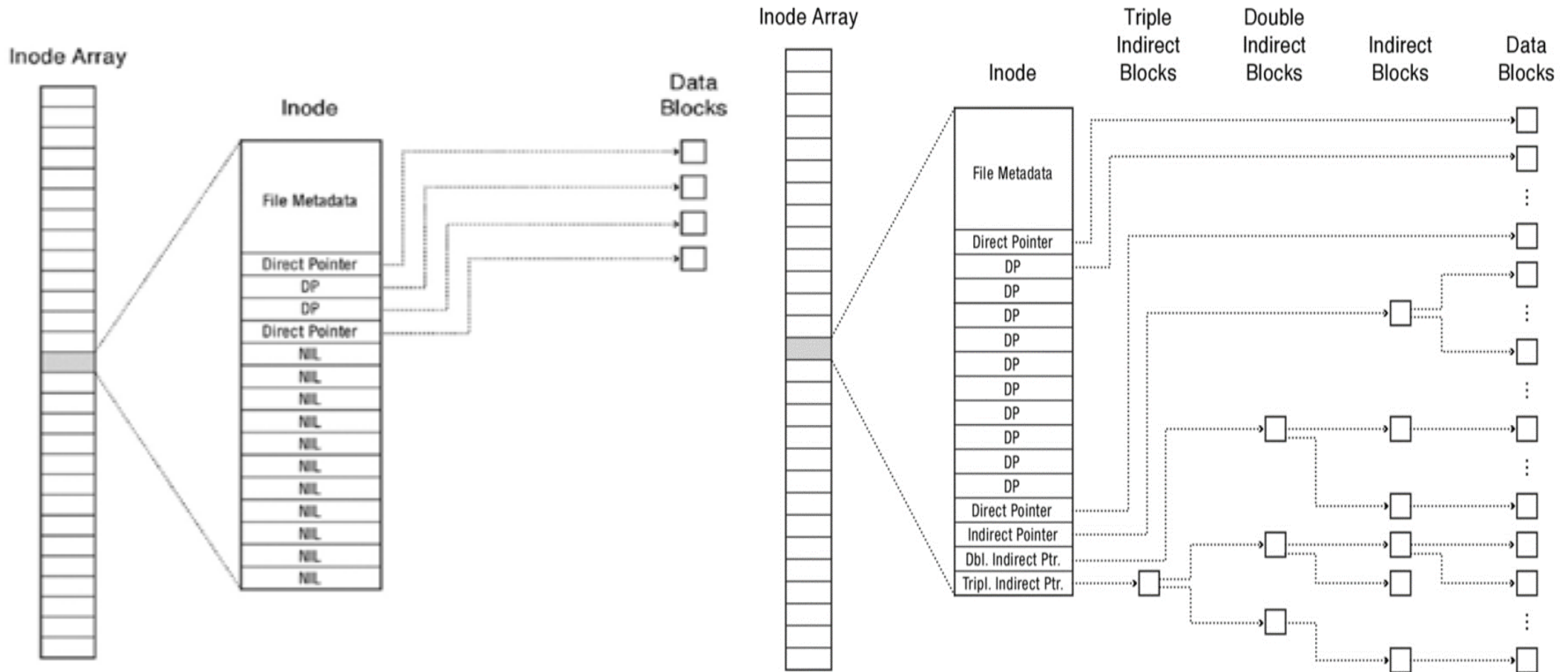


file number - inumber in FFS, is an index into the inode array

FFS Asymmetric Tree

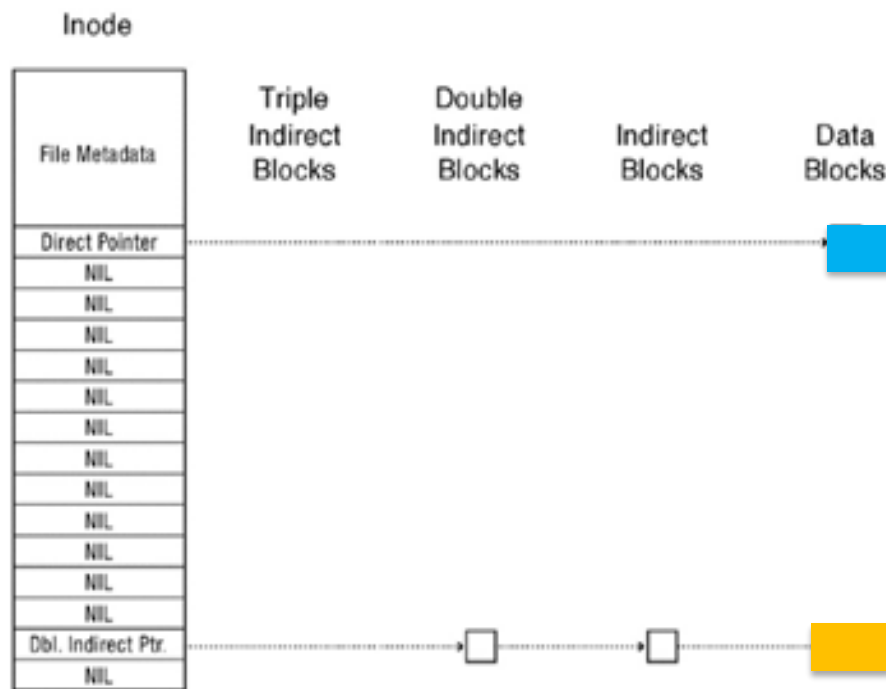
- Each file is represented as a tree
 - High degree (internal nodes with many children)
 - indirect block can contain pointers to 1024 blocks
 - improve efficiency for sequential reads/writes
 - Fixed structure
- Small files: shallow tree
 - Efficient storage for small files
- Large files: deep tree
 - Efficient lookup for random access in large files
- Sparse files: only fill pointers if needed

Small and Large FFS Trees



Sparse FFS Tree

- FFS with 4 KB blocks will only consume 16 KB - two data blocks, a double indirect block, and a single indirect block
- However, the size of the file will be 1.1GB



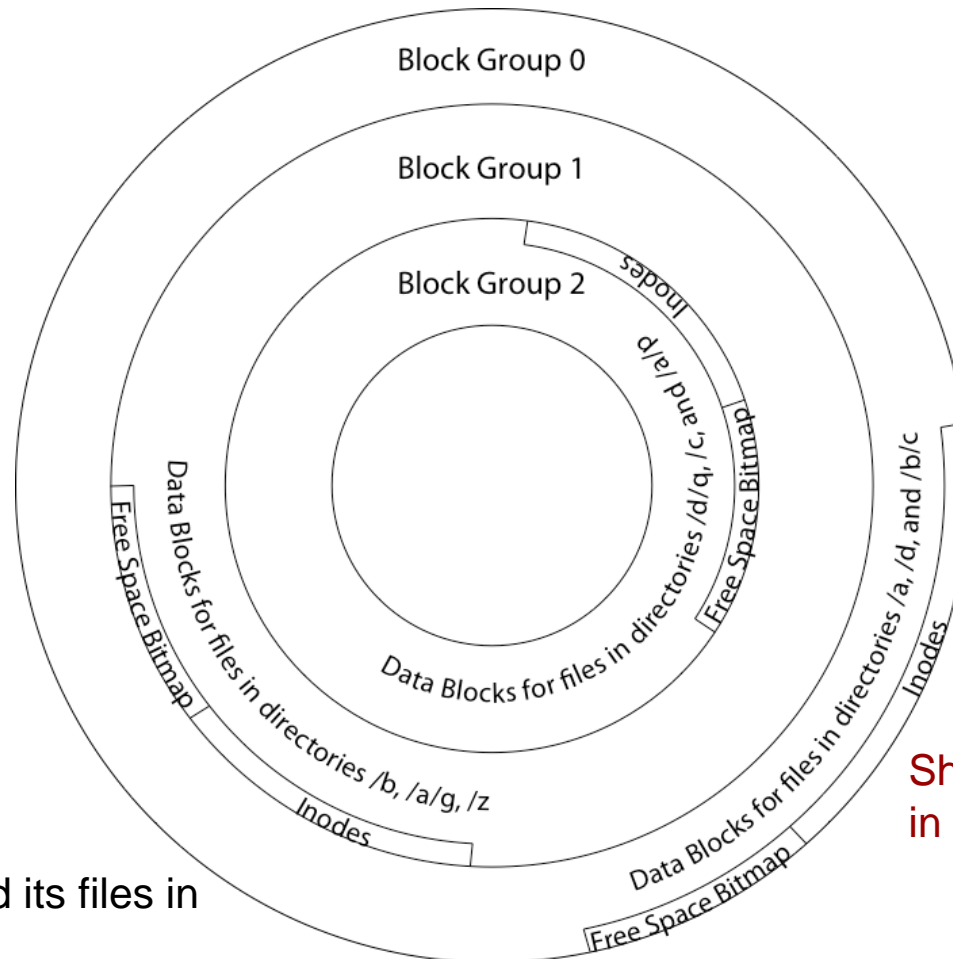
```
>ls -lgGh sparse.dat
-rwx— 1 1.1G 2012-01-31 08:45 sparse.dat*
>du -hs sparse.dat
16K sparse.dat
```

FFS Free Space Allocation

- Free space management:
 - bitmap one bit per storage block
- First fit allocation
 - Small files fragmented, large files contiguous
- Optimize for the common case:
 - Block group allocation
 - Block **group** is a set of nearby cylinders
 - Files in the same directory located in same group
 - Subdirectories located in different block groups

FFS Disk Layout

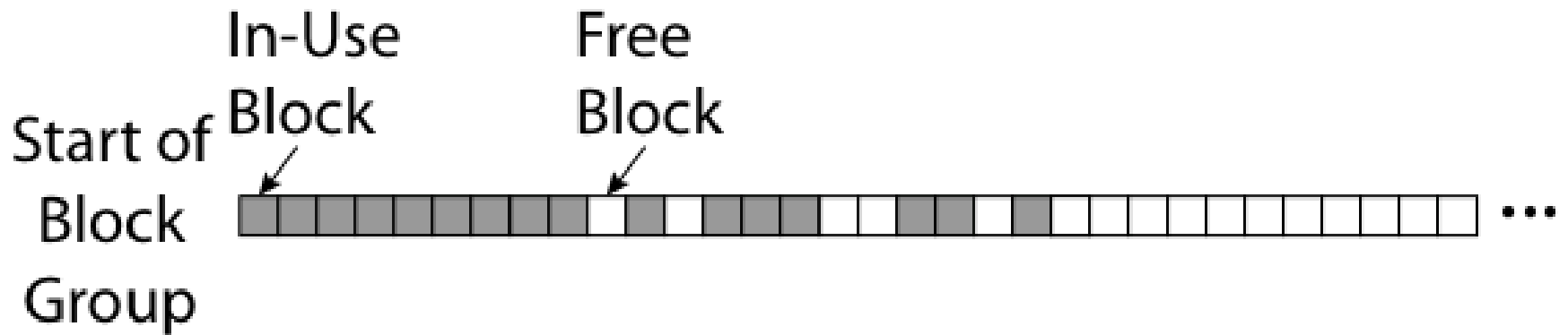
inode array and free space bitmap are conceptually arrays of records, stored at a well-known location, however split into pieces and distributed across disk



Should we put a subdirectory in the same block group?

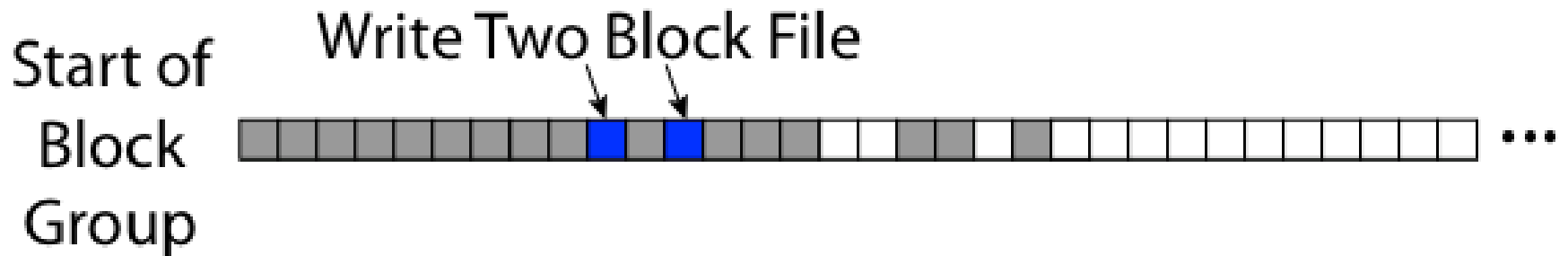
FFS puts directory and its files in the same block group

FFS First Fit Block Allocation



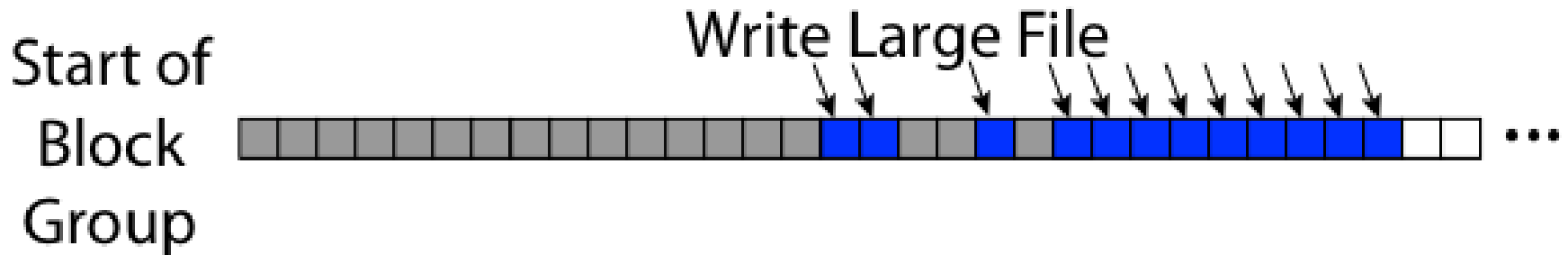
FFS's block placement heuristic is to put each new file block in the first free block in that file's block group

FFS First Fit Block Allocation



Small files fill holes near start of block group

FFS First Fit Block Allocation





Large files fill holes near start of block group and then write most data to sequential range blocks

Reserved Space

- Block group heuristic does not work when disk is almost full
- FFS reserves some disk space (about 10%) and reports only the rest as free
- Memory waste vs. faster disk access



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 some free space to prevent fragmentation

Outline

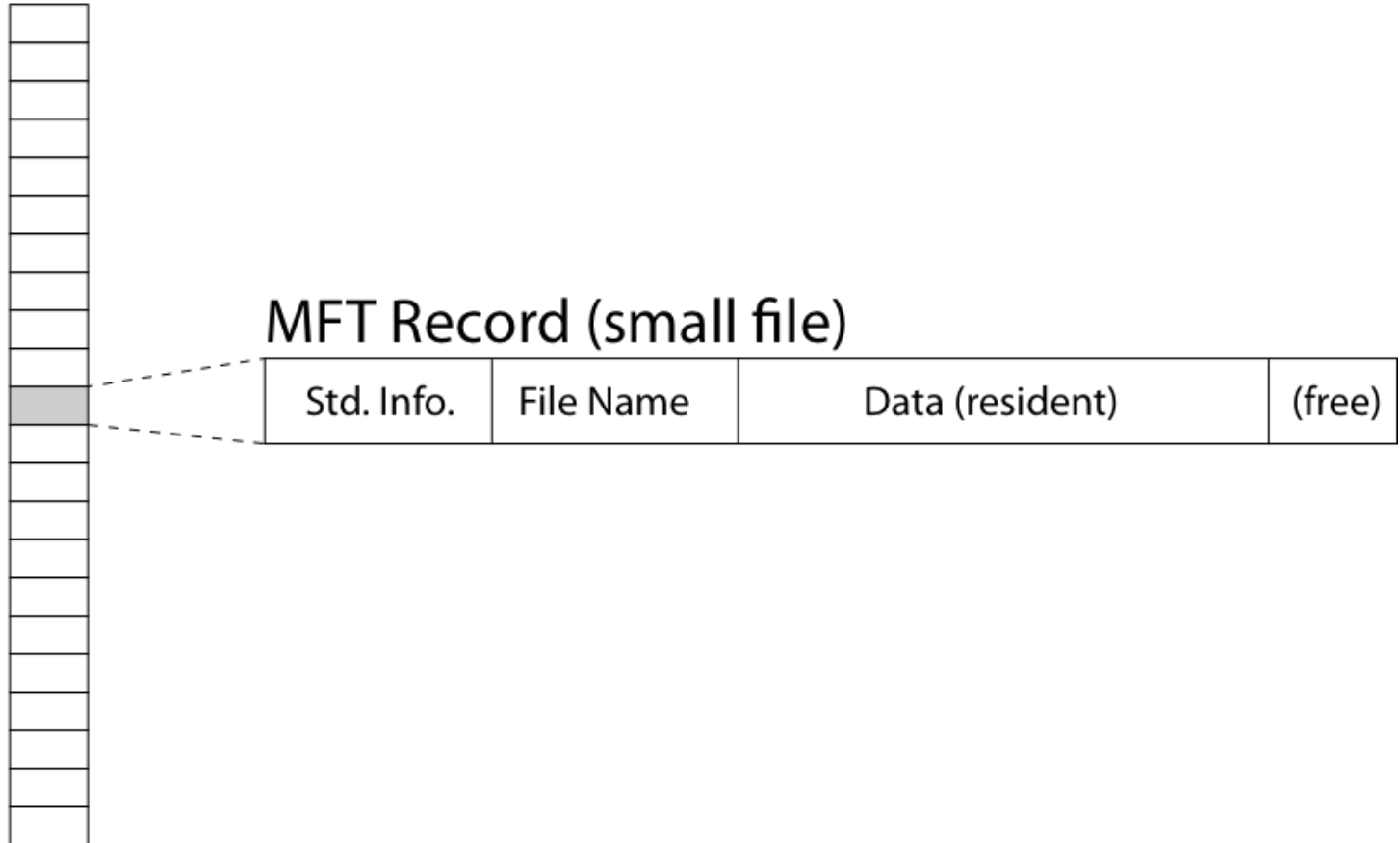
- Disc scheduling
- Flash Storage
- File-System Structure
- On-disk and in-memory structures
- **File system design**
 - FAT
 - FFS
 - **NTFS**
- Copy-on-write file systems
- Directories

Microsoft New Technology File System NTFS

- Extents
 - Flexible tree with extents
 - variable depth, deeper if file is fragmented
- Master File Table (MTF)
 - Flexible 1KB storage for metadata and data (attributes)
 - Attributes can be resident and non-resident
- Flexible tree and master file table
 - Each file is represented by a variable depth tree
 - The extent pointers for a file with a small number of extents can be stored in a shallow tree, even if the file is large
- Journaling for reliability

NTFS Small File

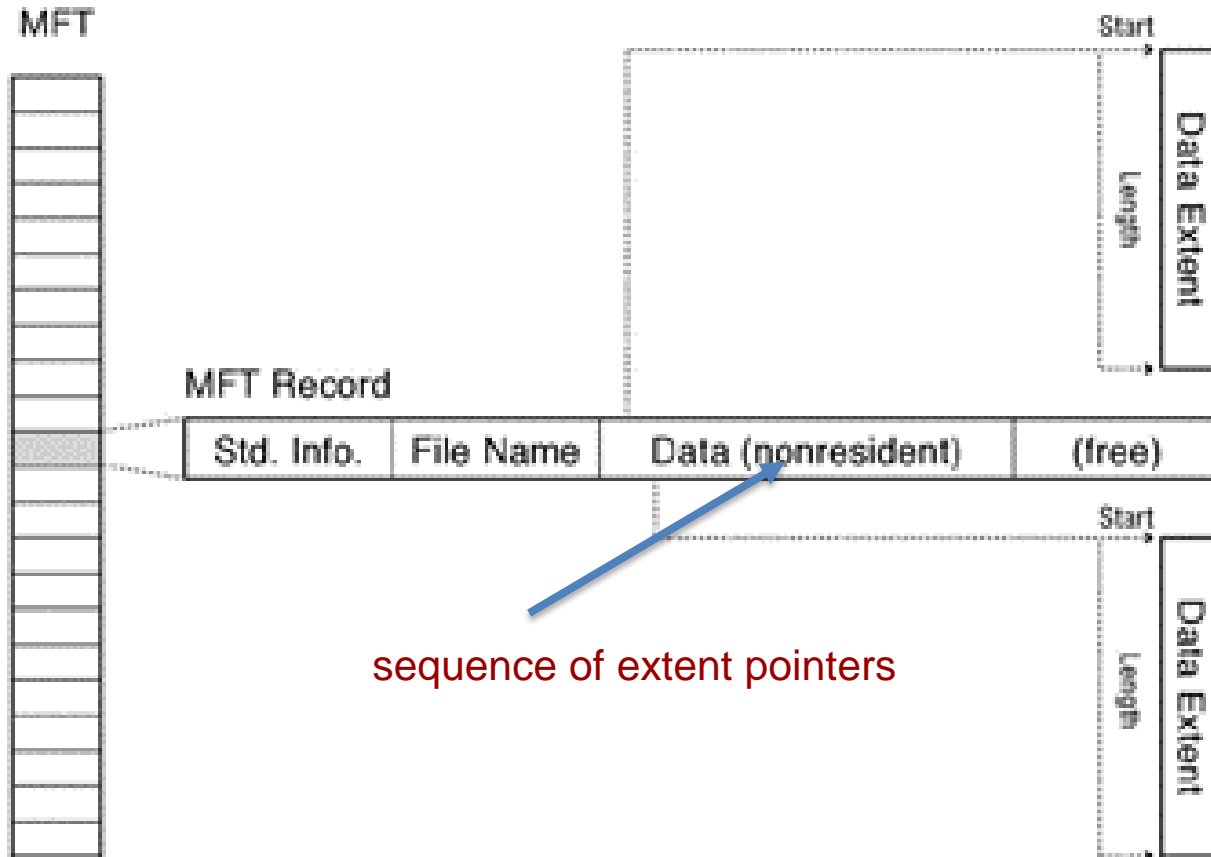
Master File Table



MFT record can store different attributes: standard info, file name, attribute list

NTFS: file with two data extents

starting block and length in blocks of an extent of data

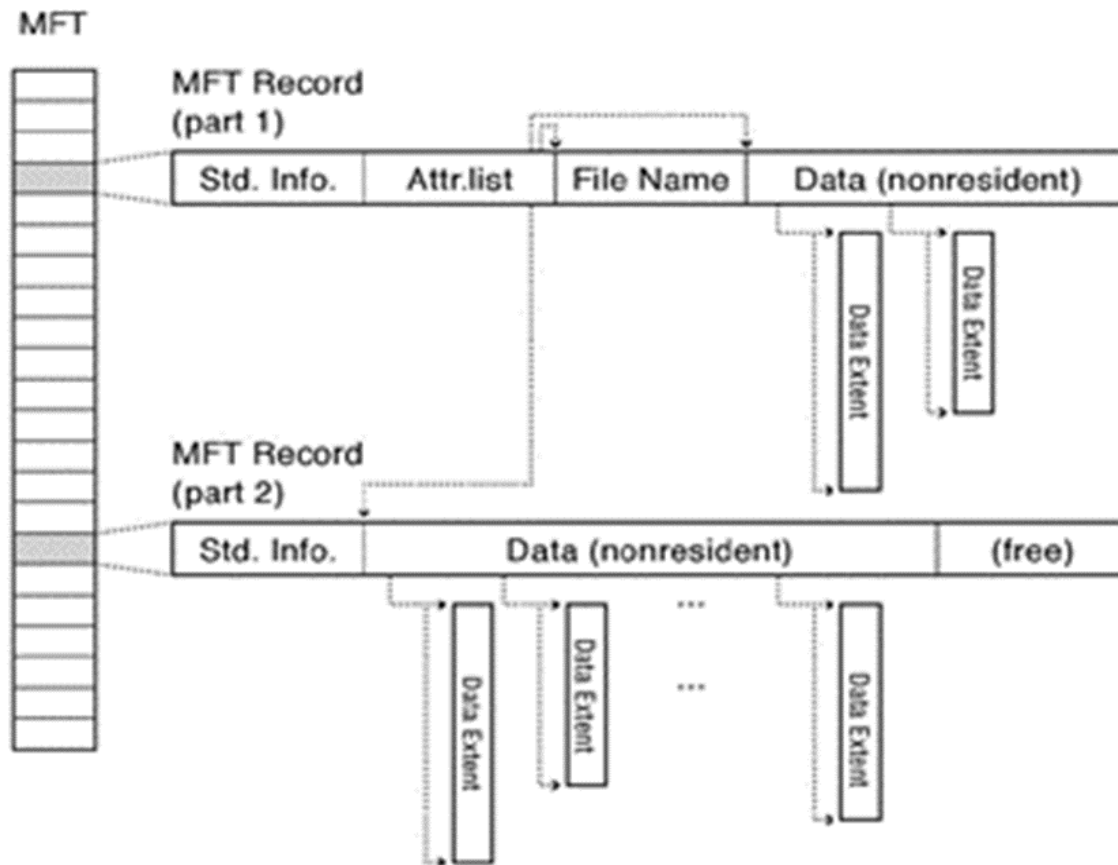


extents can hold variable numbers of blocks

NTFS: Attribute List

Attributes can grow to span multiple records

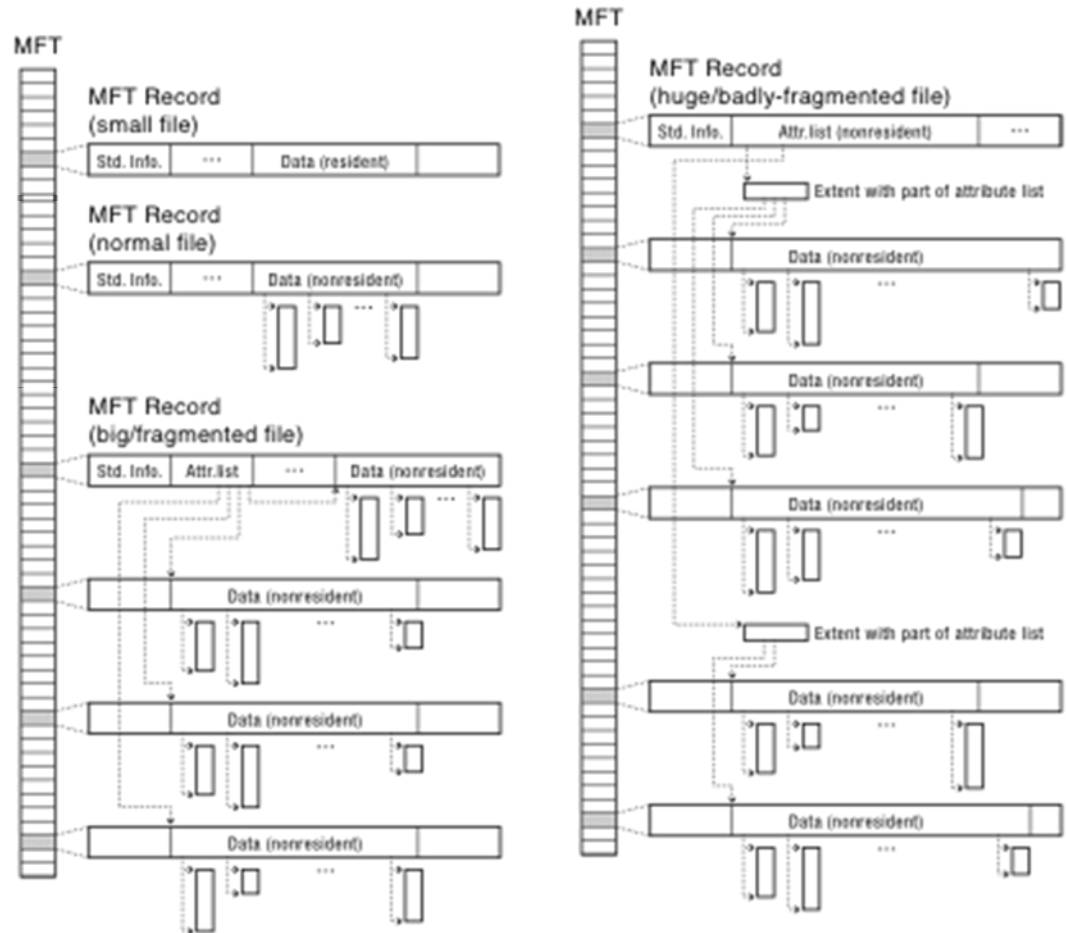
- the first MFT record includes an attribute list: pointers to attribute records



NTFS Attributes

Four stages of file growth:

1. All content is resident
2. Small number of extents in single non-resident attribute
3. Extent pointers in multiple records, resident attribute list
4. Non-resident data attributes, tracked by a non-resident attribute list



NTFS Locality

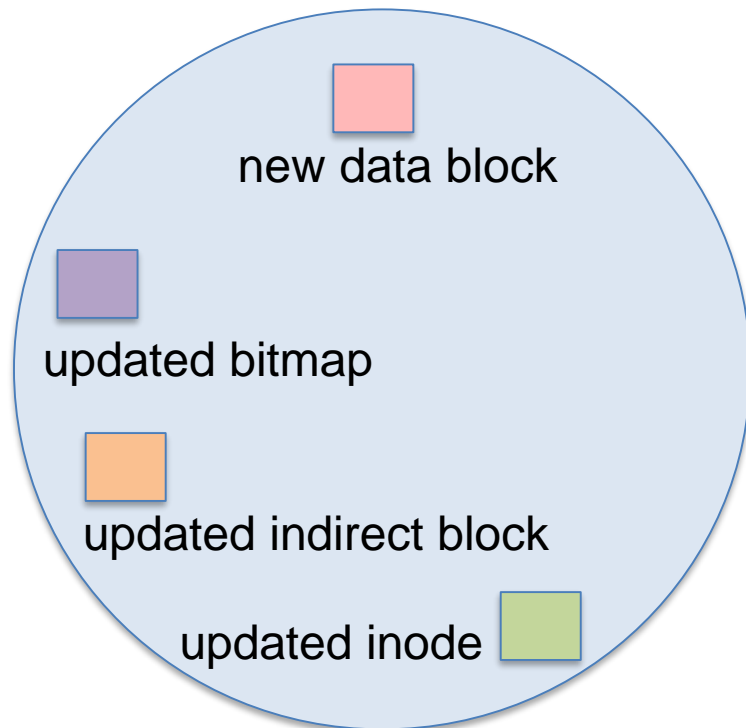
- Variation of best fit
 - smallest free region that is large enough
- Not all bitmap is in memory
 - system caches the allocation status for a smaller region of the disk
- Start of a volume is reserved for MFT expansion
 - to avoid MFT fragmentation
 - does not place file blocks in the reserve area until the non-reserved area is full
 - halves the size of the MFT reserve area and continues
- Defragmentation utility

Outline

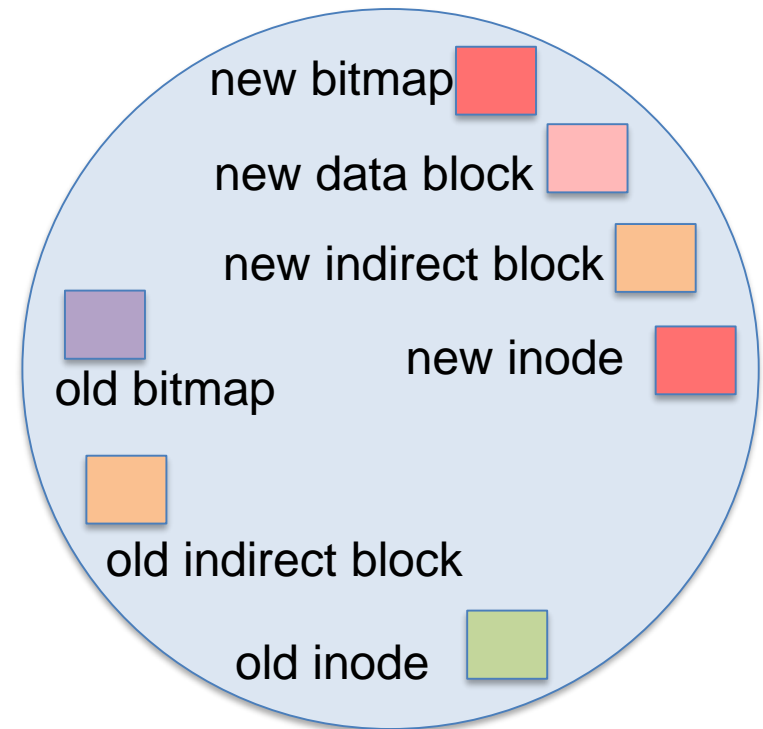
- Disc scheduling
- Flash Storage
- File-System Structure
- On-disk and in-memory structures
- File system design
 - FAT
 - FFS
 - NTFS
- **Copy-on-write file systems**
- Directories

Copy-On-Write (COW) File Systems

- COW file systems do not overwrite the existing data
 - transform random I/O updates to sequential one

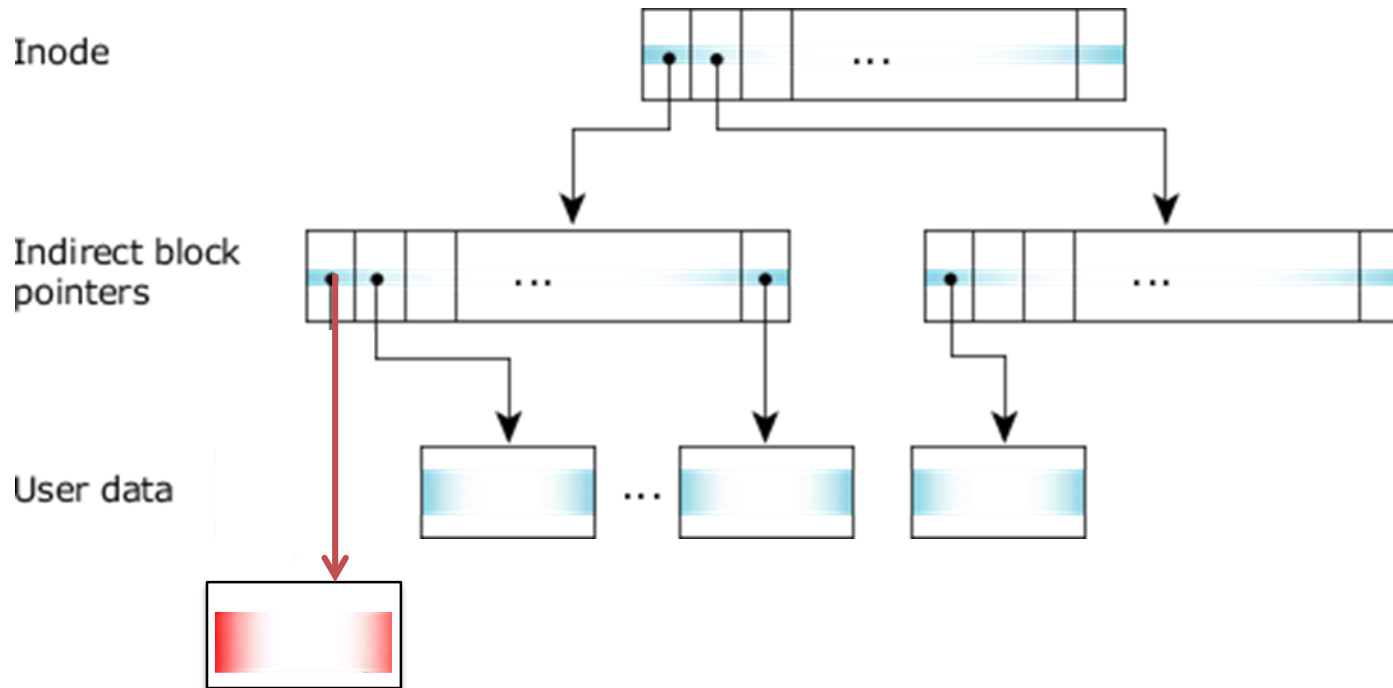


update-inplace FS

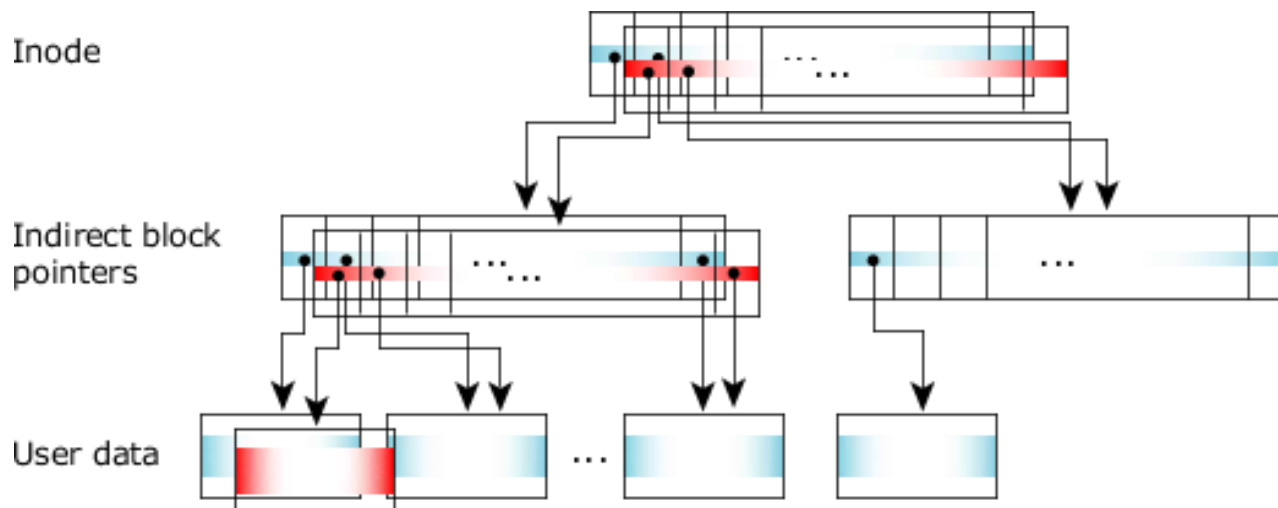
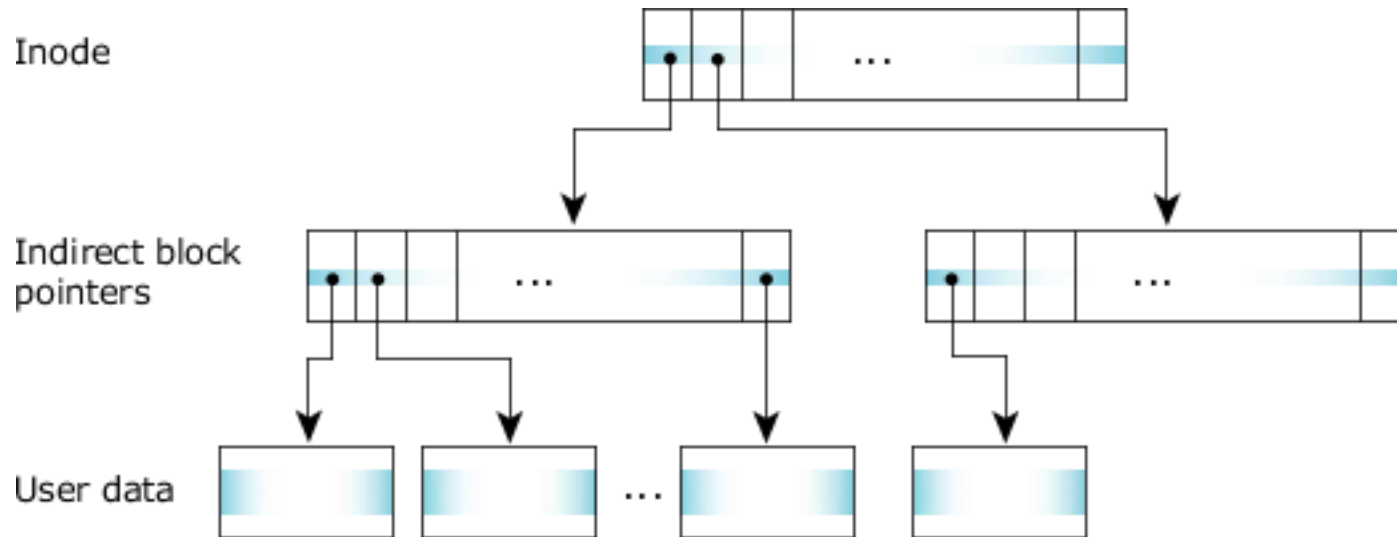


COW FS

CoW File System: Updating a File



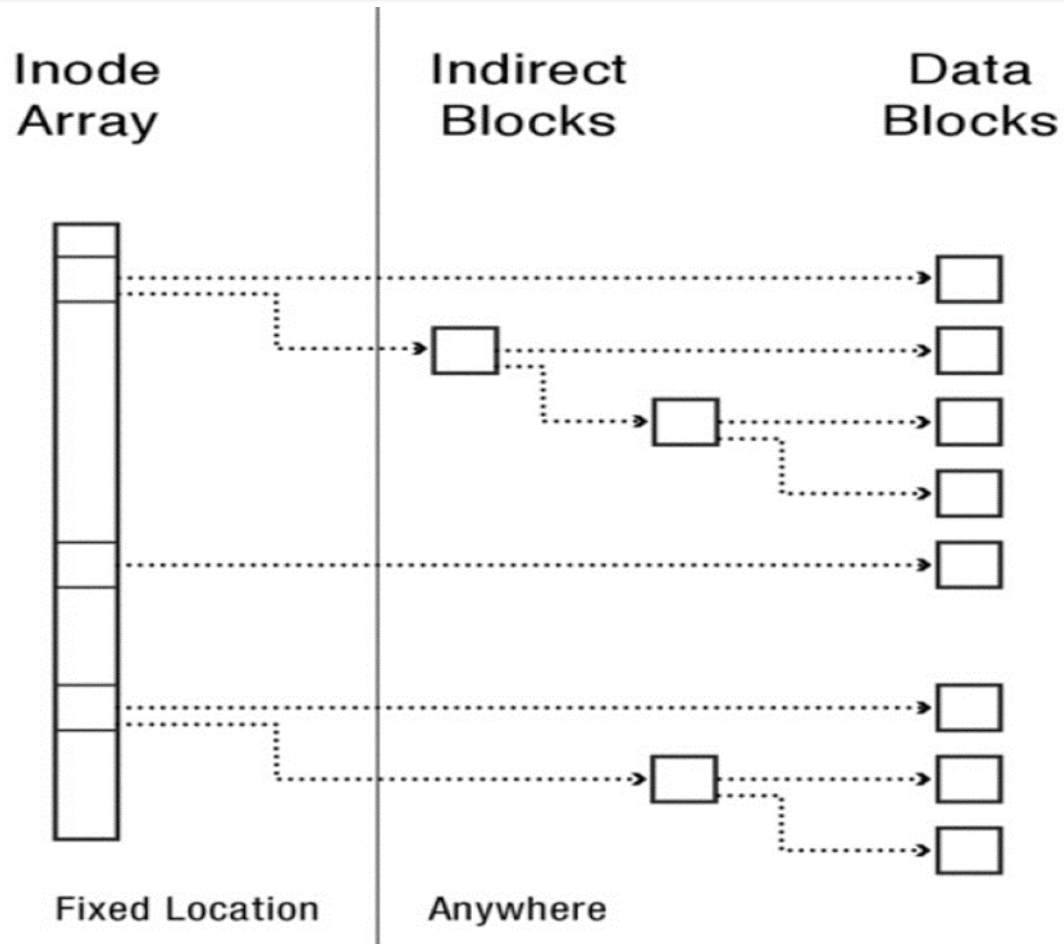
CoW File System: Updating a File



Trends Driving Adoption of COW FSs

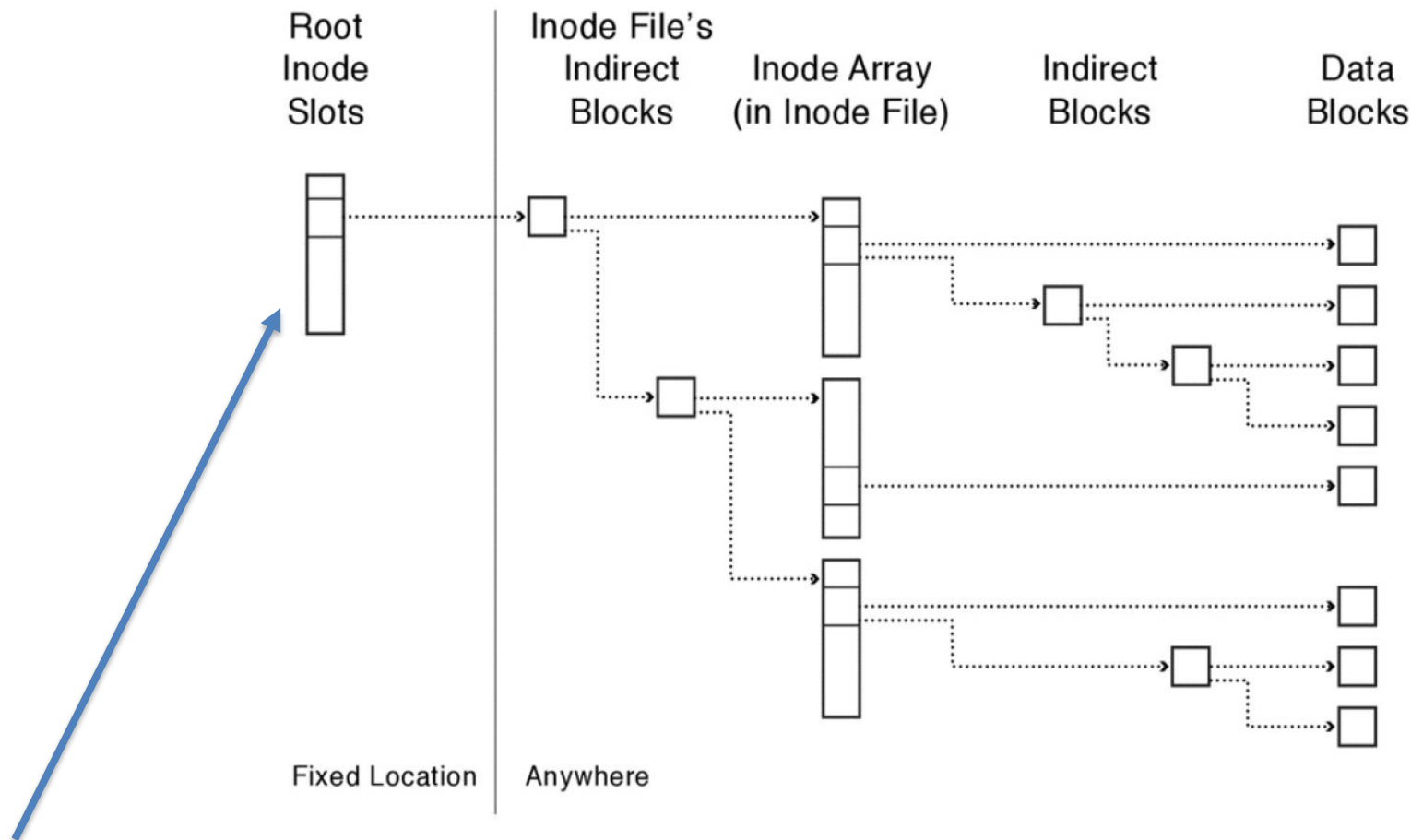
- Small writes are expensive
 - Especially on RAID
- Widespread adoption of flash storage
 - To write small page (4KB) need to clear large (512KB) erasure block
 - Wearing out due to frequent write/erase; wear leveling
- Growing capacity enables versioning
 - updates in COW do not overwrite old data; versioning

A Traditional Update-in-Place FS



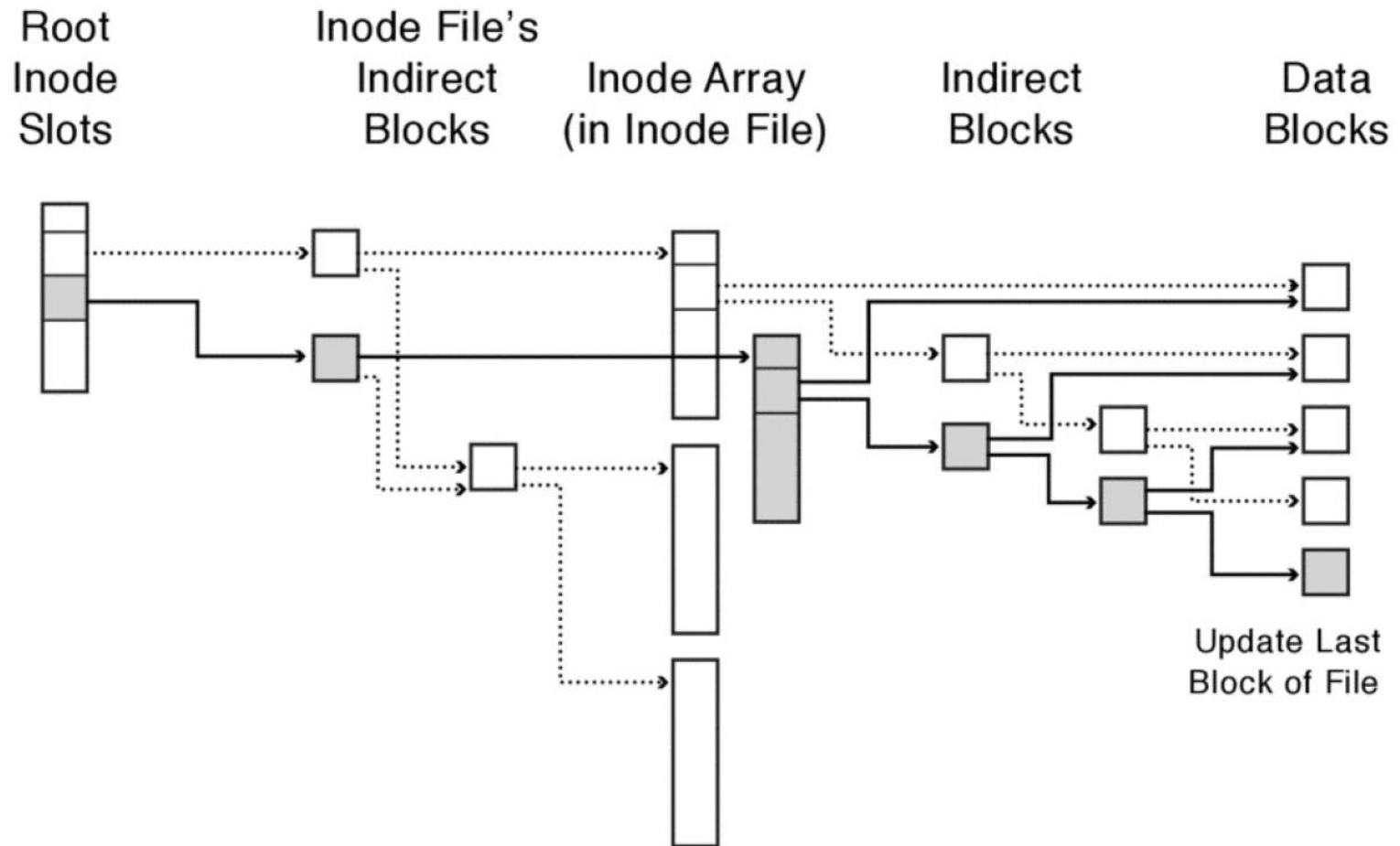
- File's indirect nodes and data blocks can be located anywhere on disk
- Given a file's inumber, we can find its inode in a fixed location on disk

Simple COW File System



- Include a monotonically increasing version number and checksum in the root inode
- Keep a small array of slots for the current and recent root inodes, updating the oldest one

COW: Writing a Block

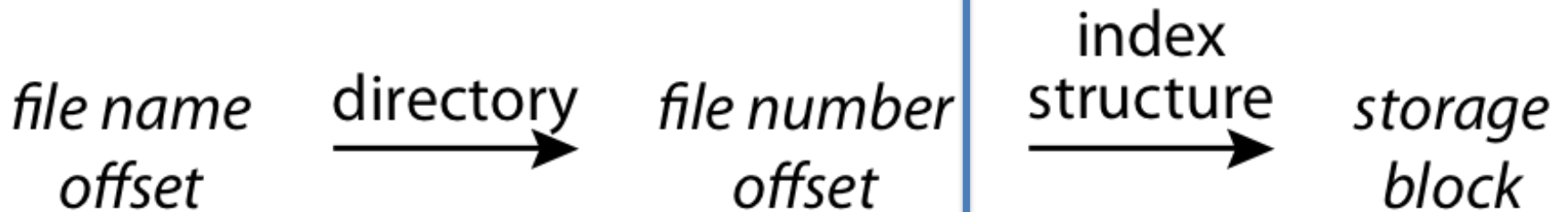


Write block and all of the blocks on the path from it to the root to new locations

Outline

- Disc scheduling
- Flash Storage
- File-System Structure
- On-disk and in-memory structures
- File system design
 - FAT
 - FFS
 - NTFS
- Copy-on-write file systems
- **Directories**

Recall: Named Data in a File System

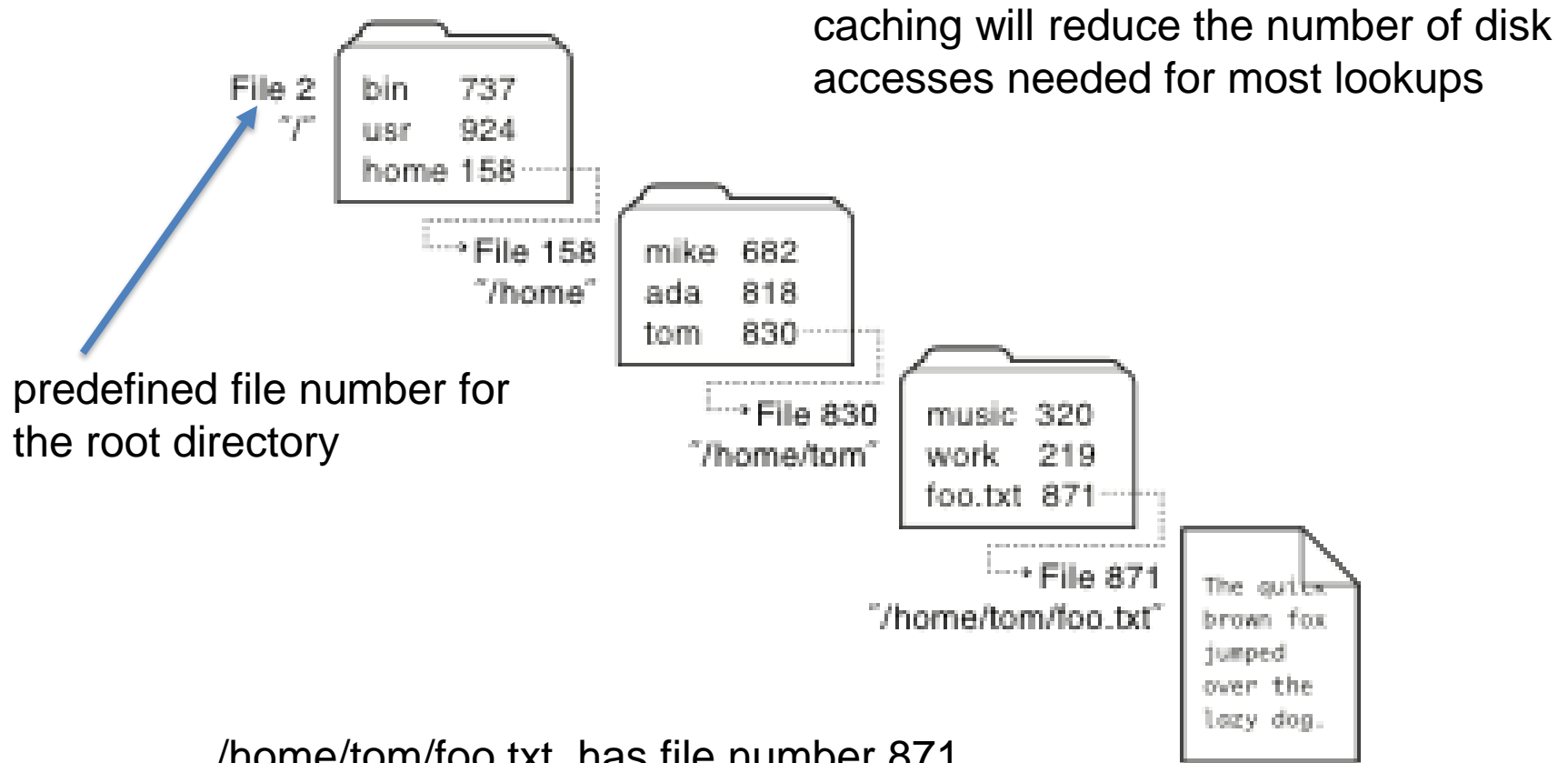


Directories Are Files



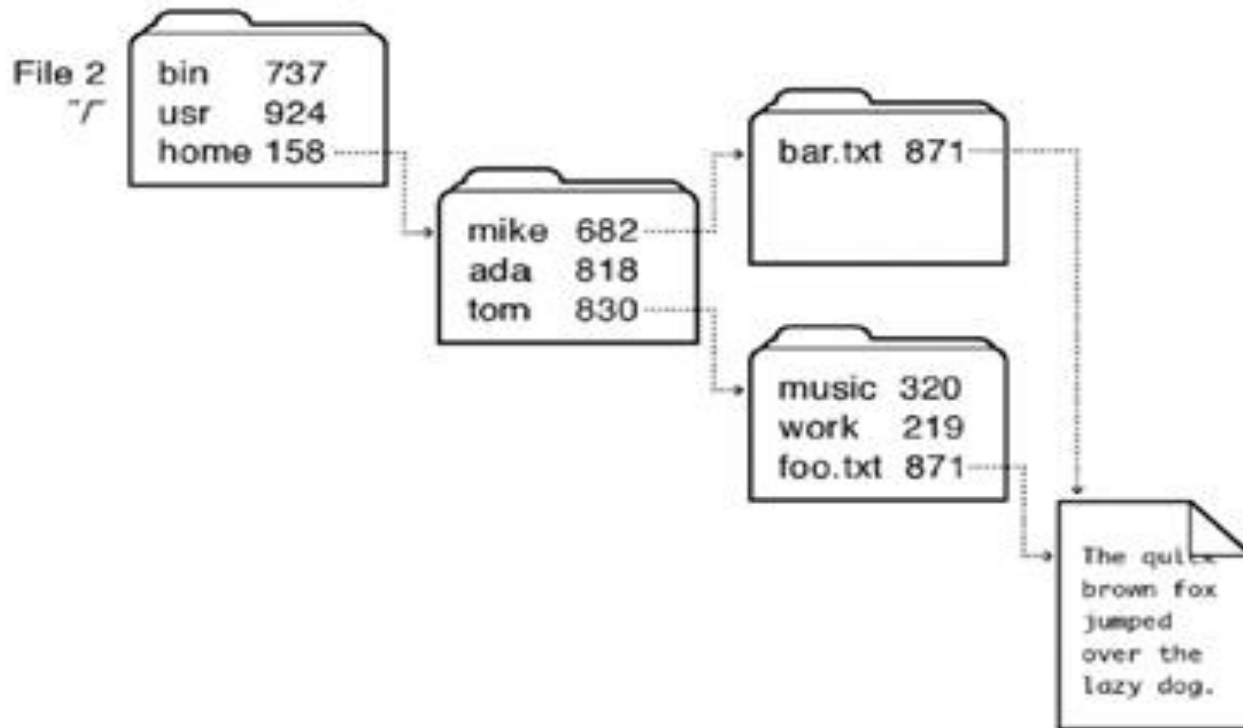
use files to store directories!

Recursive Filename Lookup



Hard and Soft Links

Example of a directed acyclic graph directory with multiple hard links to a file



Hard links map different path names to the same file number


Soft or symbolic links are directory entries that map one name to another name

Small Directory Layout

Directory stored as a file: linear list of file name/file number
Linear search to find filename (small directories)

File 830
"/home/tom"

Name	.	..	music	work		foo.txt		End of File
File Number	830	158	320	219	Free Space	871	Free Space	
Next								



Large Directories: Logical View

Tree node contains an array of (hash key, file offset) - a pointer to child node containing entries with keys smaller than the hash key

