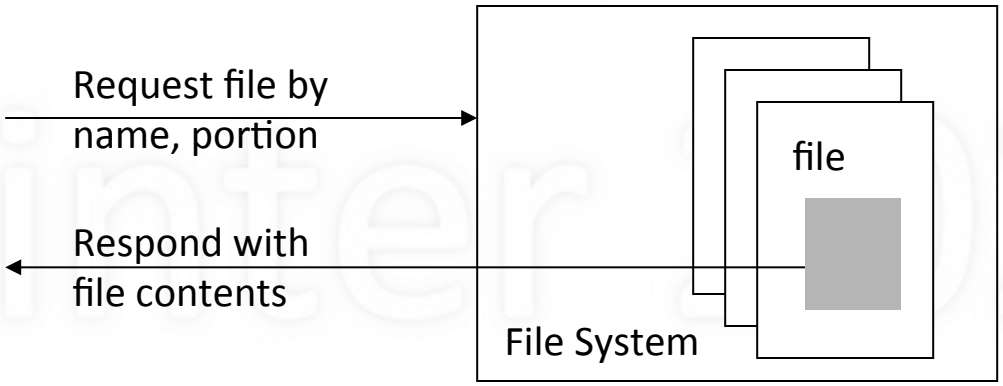
CSE 120: Principles of Operating Systems
Lecture 11: File System

Prof. Joseph Pasquale
University of California, San Diego
March 13, 2023

What is a File? File System?



- File: logical unit of storage, container of data
  - Accessed by <name, region within file>
- File System: a structured collection of files
  - Access control, name space, persistent storage

File System Abstraction

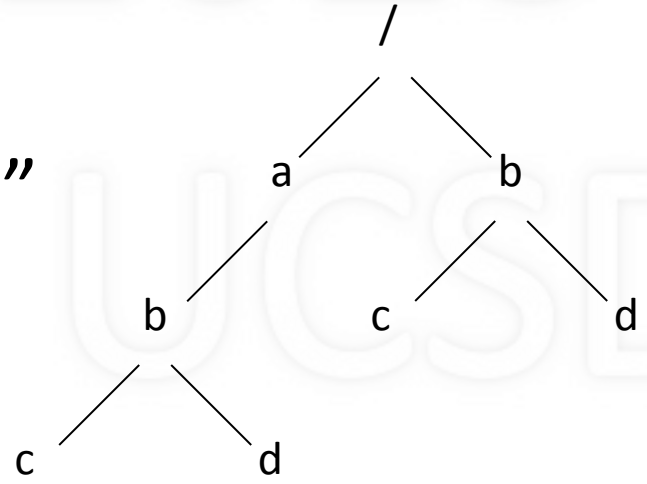
- Repository of objects
  - Objects are data, programs, for system, users
  - Objects referenced by name, to be read/written
- Persistent: remains "forever"
- Large: "unlimited" size
- Sharing: controlled access
- Security: protecting information

More than a Repository

- Any/all objects where following make sense
  - Accessed by name
  - Can be read and/or written
  - Can be protected: read-only, read-write, …
  - Can be shared
  - Can be locked
- I/O devices: disk, keyboard, display, …
- Processes: memory

Hierarchical File Name Space

- Name space is organized as a tree
  - Name has components, branches start from root
  - No size restrictions
  - Intuitive for users
- Example: UNIX “pathnames”
  - Absolute: /a/b/c
  - Relative: b/c relative to /a
  - Not strictly a tree: links



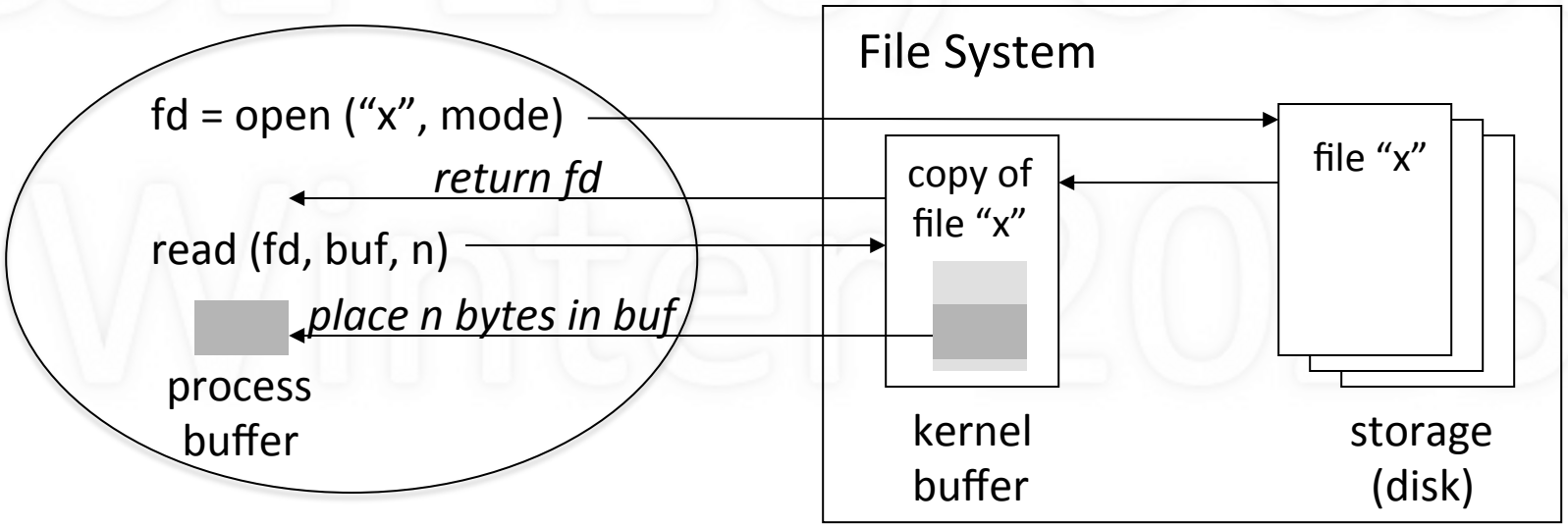
A File Has Attributes

- Type (recognized by system or users)
- Times: creation, accessed, modified
- Sizes: current size, maximum size
- Access control (permissions)

File Operations

- Creation: create, delete
- Prepare for access: open, close, mmap
- Access: read, write
- Search: move to location
- Attributes: get, set (e.g., permissions)
- Mutual exclusion: lock, unlock
- Name management: rename

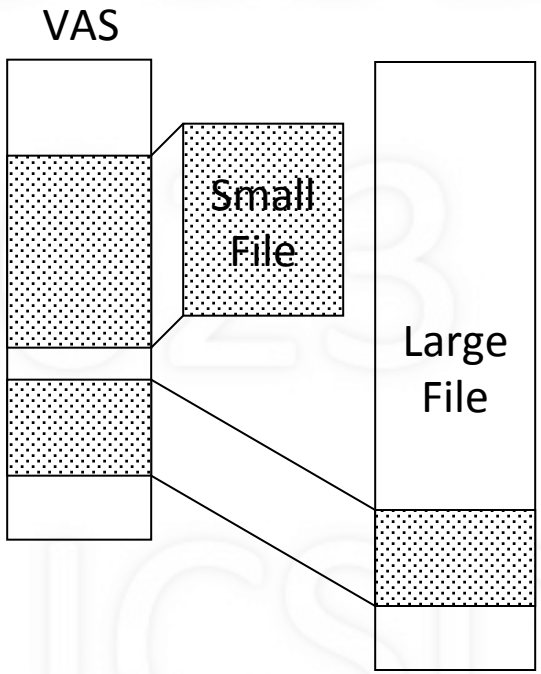
Read/Write Model



- fd = open (fname, mode)
- nr = read (fd, buf, size)
- nw = write (fd, buf, size)
- close (fd)

Memory-mapped Model

- Map file into address space
  - mmap (fd, addr, n);
  - addr = mmap (fd, NULL, n);
- Can then use memory ops
  - x = addr[5]; strcpy (addr, “hello”);
- Issues
  - Efficient for multiple processes sharing memory
  - If memory is written, how is file actually updated?



Access Control

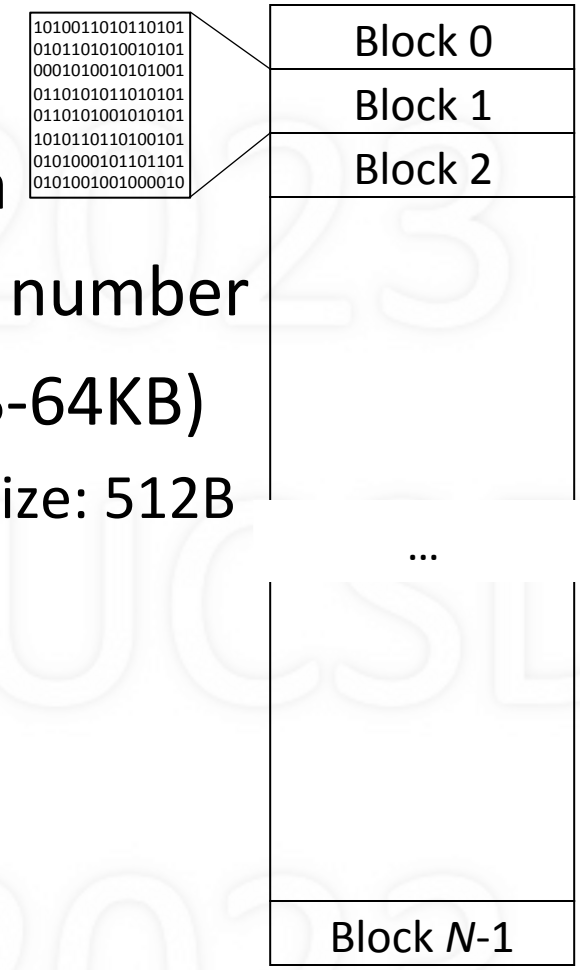
- How are files shared, to varying degrees?
- Access control
  - Who can access file
  - What operations are allowed
  - User interface must be simple and intuitive
- Example: UNIX
  - r/w/x permissions for owner, group, and everyone

File System Implementation: Goals

- Archival storage
  - Keep forever, including previous versions
- Support various storage technologies
  - Disks (different types), remote disks, …
- How to best achieve and balance
  - Performance
  - Reliability
  - Security

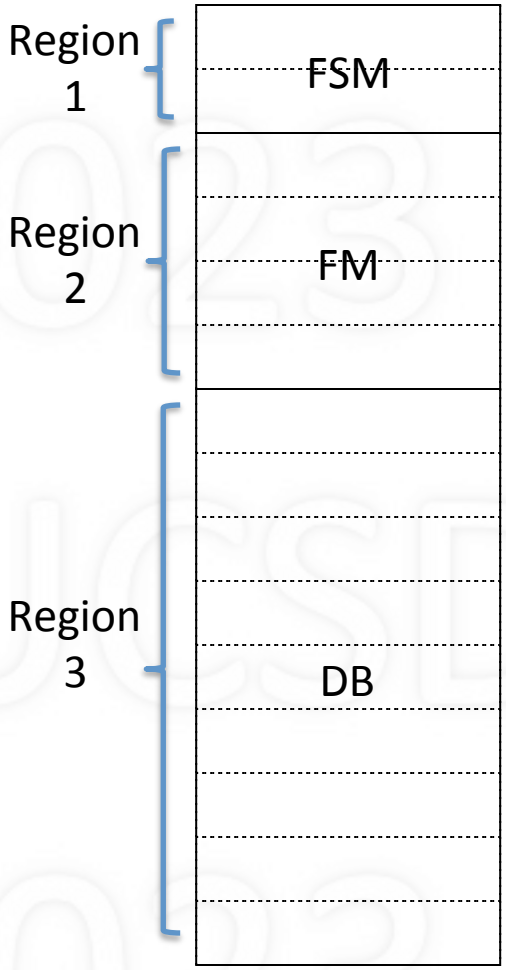
Storage Abstraction

- Hide complexity of device
  - Model as array of blocks of data
  - Randomly addressable by block number
  - Typical block size: 1KB (also 4KB-64KB)
    - Generally multiple of disk sector size: 512B
- Simple interface
  - read (block_num, mem_addr)
  - write (block_num, mem_addr)



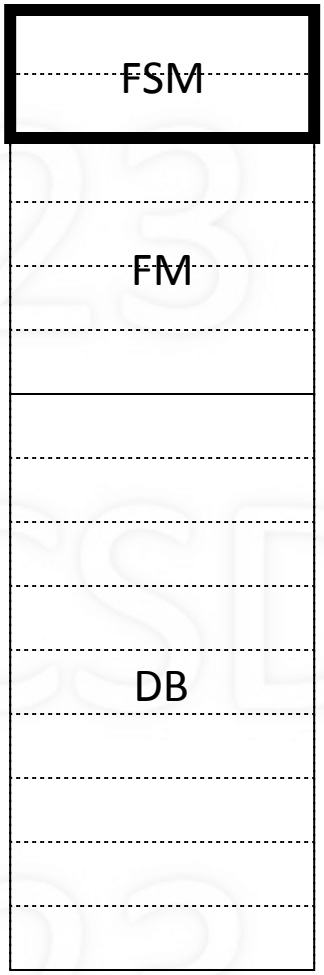
Typical Implementation Structure

- Three major regions
  - Sequence of blocks for each one
- Region 1: File System Metadata
  - Information about file system
- Region 2: File Metadata
  - File control blocks
- Region 3: Data Blocks
  - File contents



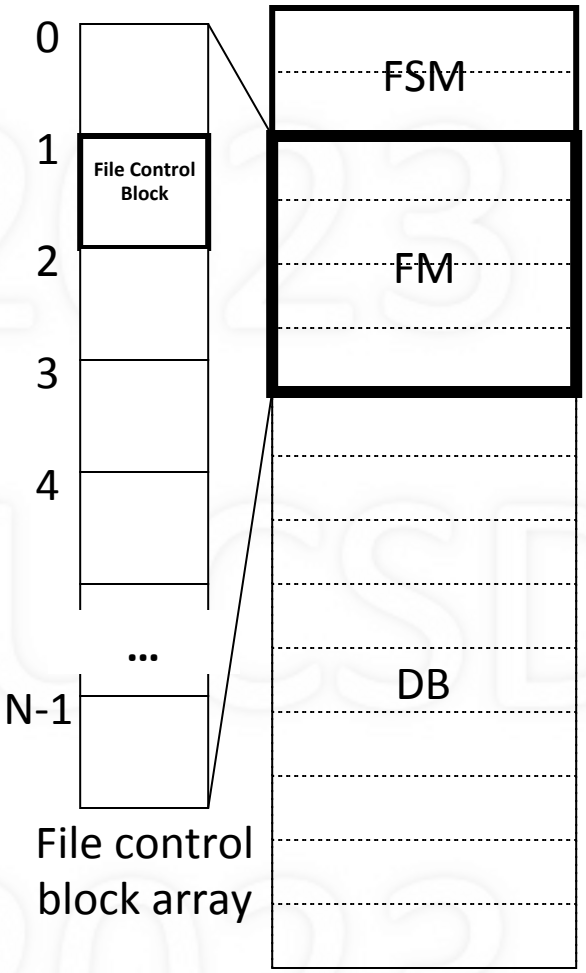
File System Metadata

- Information about the file system
- Sizes
  - Files in use, free entries
  - Data blocks in use, free entries
- Free lists (or bitmaps)
  - File control blocks
  - Data blocks



File Metadata: File Control Blocks

- Information about a file
- Referenced by number/index
- Contains
  - Attributes
    - type of file, size, permissions, …
  - References to data blocks
    - disk block map
- Note: many file control blocks may fit in single storage block

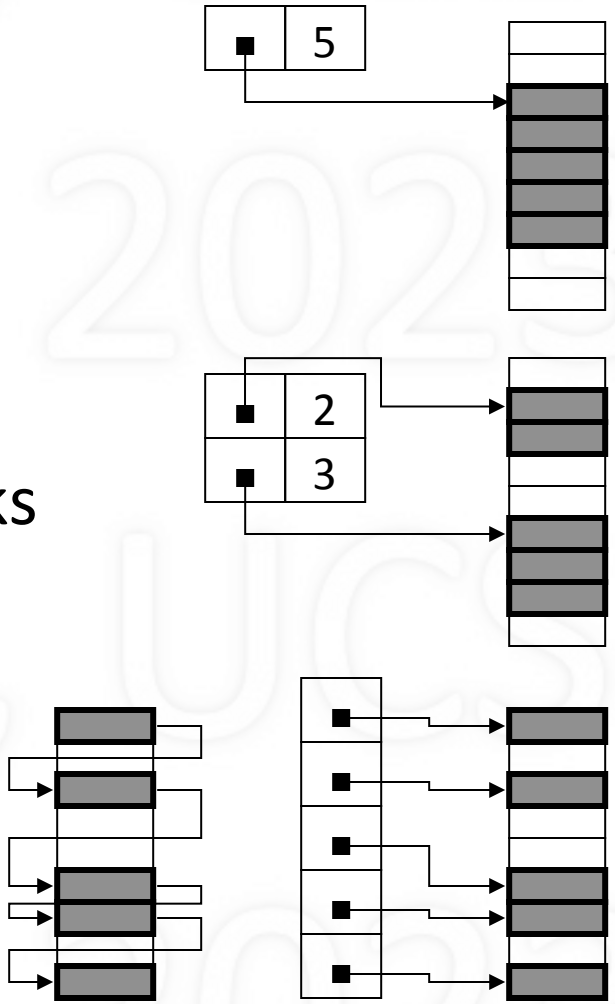


Example of File Control Block

- Number: 88 (index in file control block array)
- Size: 4096 bytes
- Permissions: rw-r--r--
- Data blocks: set of indexes into storage array
  - E.g., 567, 7076, 9201, 9248 (assuming 1KB blocks)
  - This is an example of list of non-contiguous blocks
- What about file name?

Keeping Track of Allocated Blocks

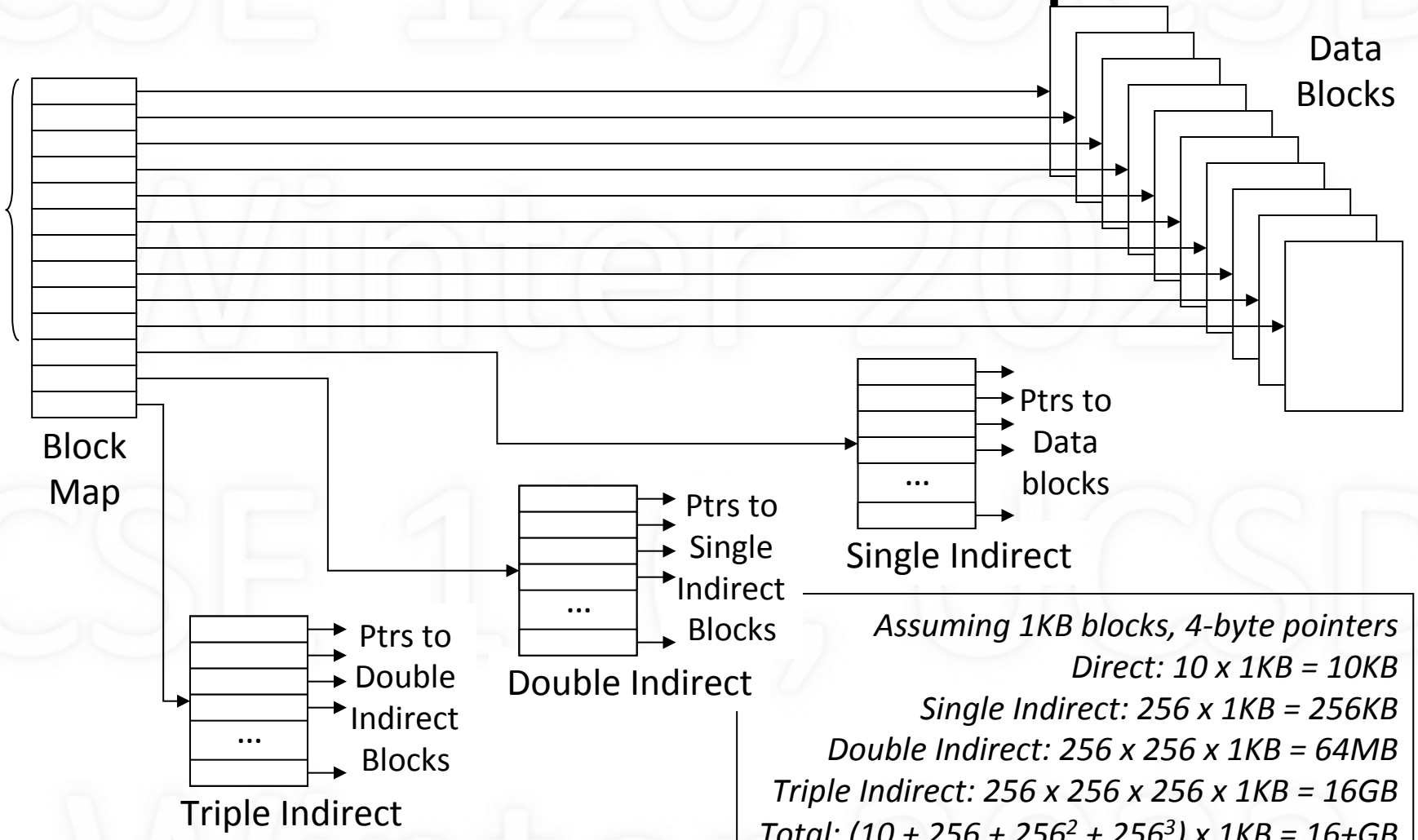
- Contiguous blocks
  - Single sequence of blocks
- Extents
  - Groups of contiguous blocks
- Non-contiguous blocks
  - Blocks individually named



Example: UNIX v.7 Block Map

- Array of pointers to data blocks
- 13 pointers
  - 10 direct: references 10 data blocks
  - 1 singly-indirect: references *n* data blocks
  - 1 doubly-indirect: references n^2 data blocks
  - 1 triply-indirect: references n^3 data blocks
- *n* depends on how many pointers fit in a block
  - Example: 256 4-byte pointers will fit in 1KB block

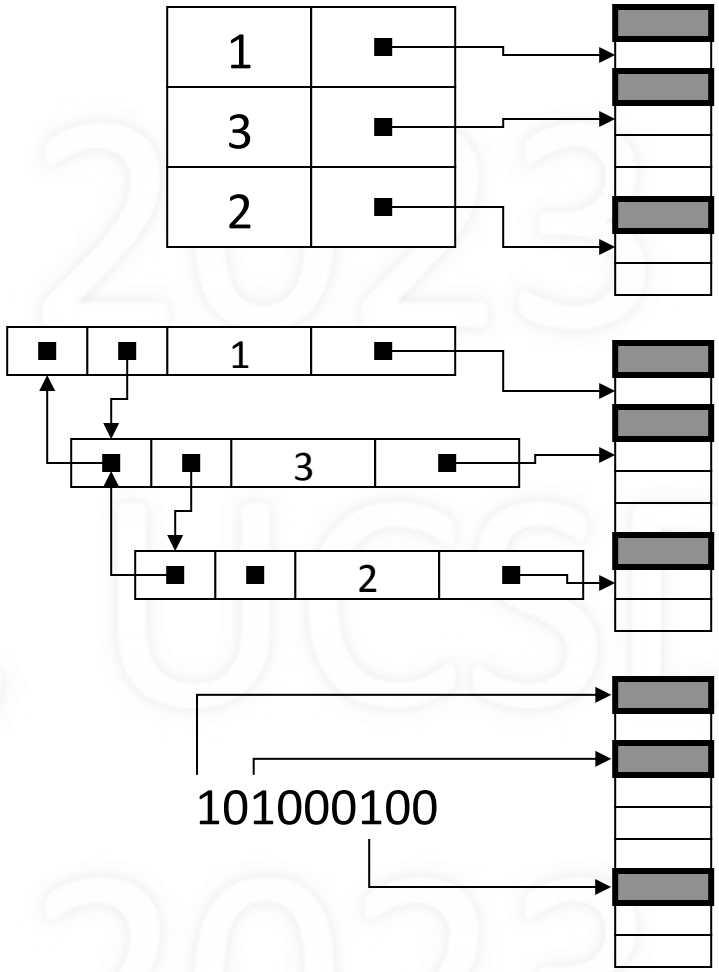
UNIX v.7 Block Map



Assuming 1KB blocks, 4-byte pointers
Direct: 10 x 1KB = 10KB
Single Indirect: 256 x 1KB = 256KB
Double Indirect: 256 x 256 x 1KB = 64MB
Triple Indirect: 256 x 256 x 256 x 1KB = 16GB
Total: $(10 + 256 + 256^2 + 256^3)$ x 1KB = 16+GB

Keeping Track of Free Blocks

- Free Block Map
  - Compact if lots of free regions of space
- Doubly Linked List
  - Easy to keep ordered due to fast inserts and deletes
- Bit Map
  - Fixed size regardless of fragmentation



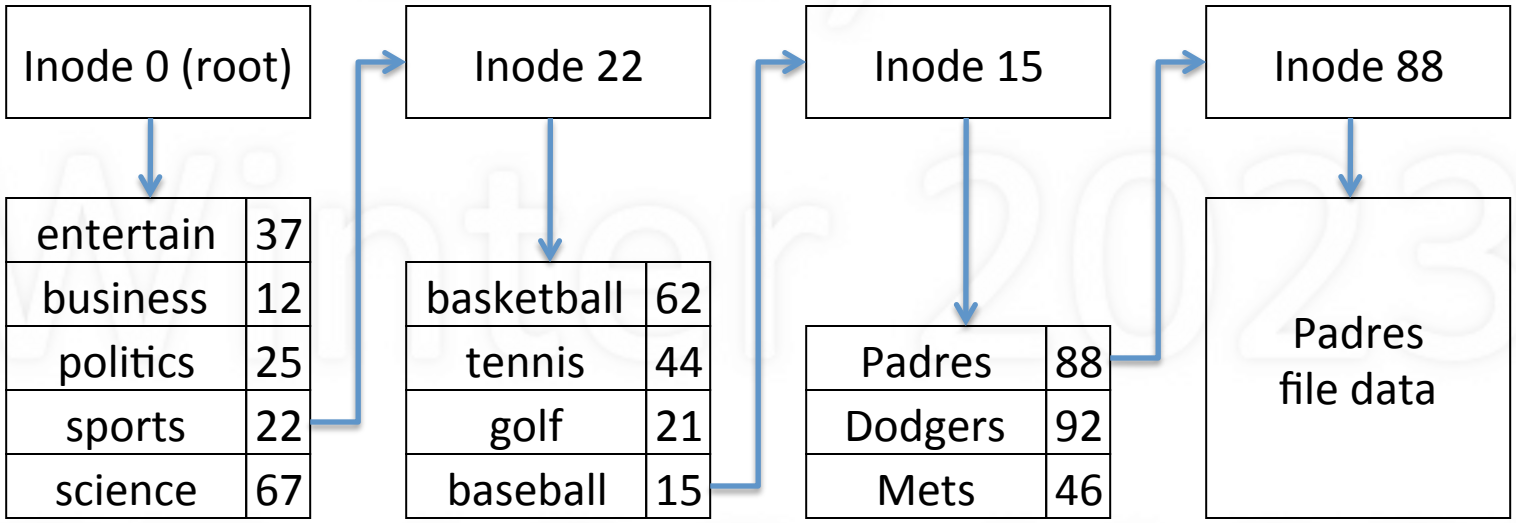
File Name to File Control Block

- Users access files using file names
- Problem: how to translate
  - from file name: "/sports/baseball/Padres"
  - to file control block number: 88
- Must parse file name
- Each branch corresponds to a directory/folder
- Each directory/folder may itself be a file

Implementing UNIX Directories

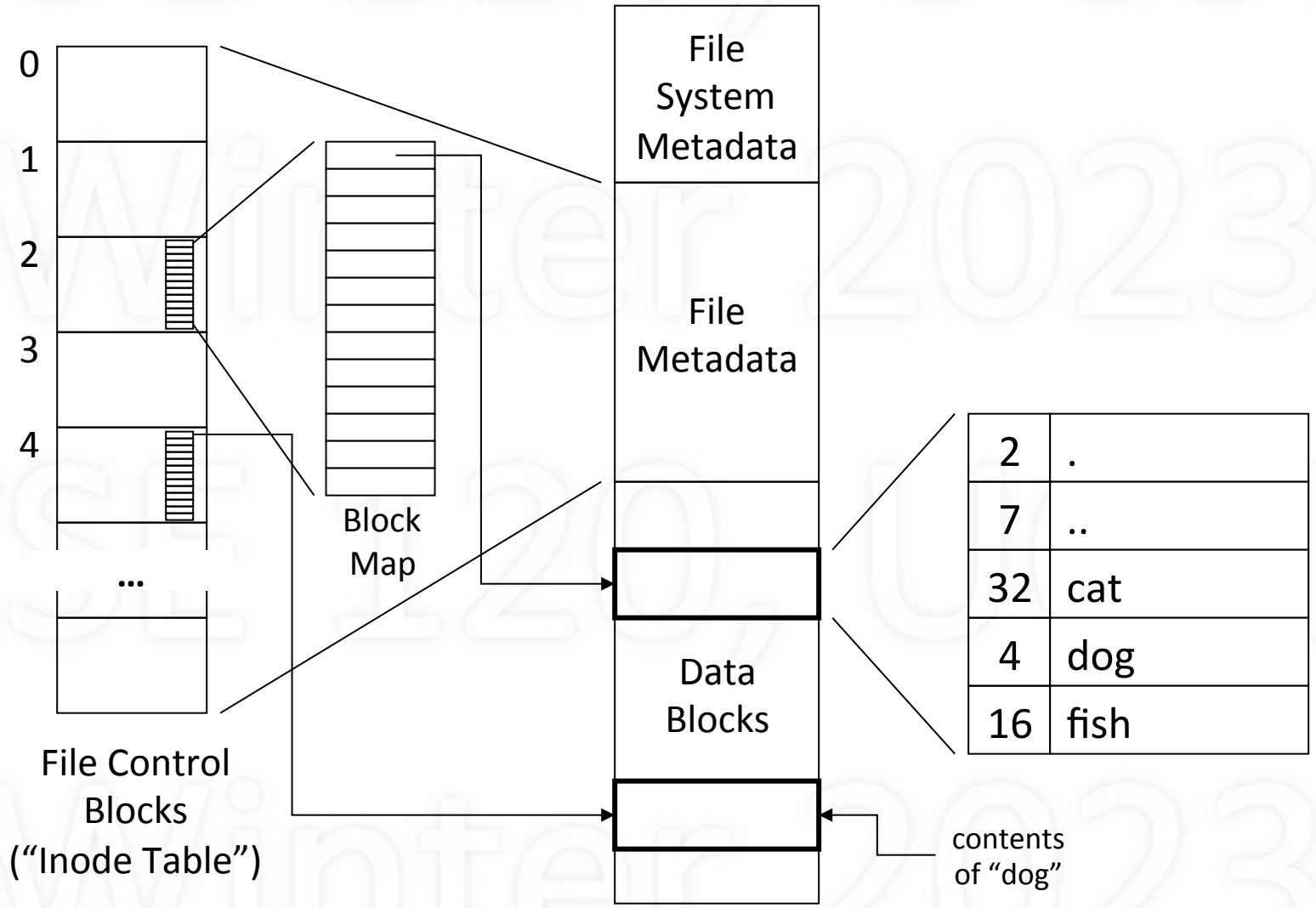
- Table where each entry contains
  - name and attributes
  - name and pointer to file control structure
- Unix (name and pointer) – pre-BSD
  - Each entry: branch name (14), i-node number (2)
  - Berkeley Unix uses a more complex scheme to support long names

Example of Parsing Names in UNIX

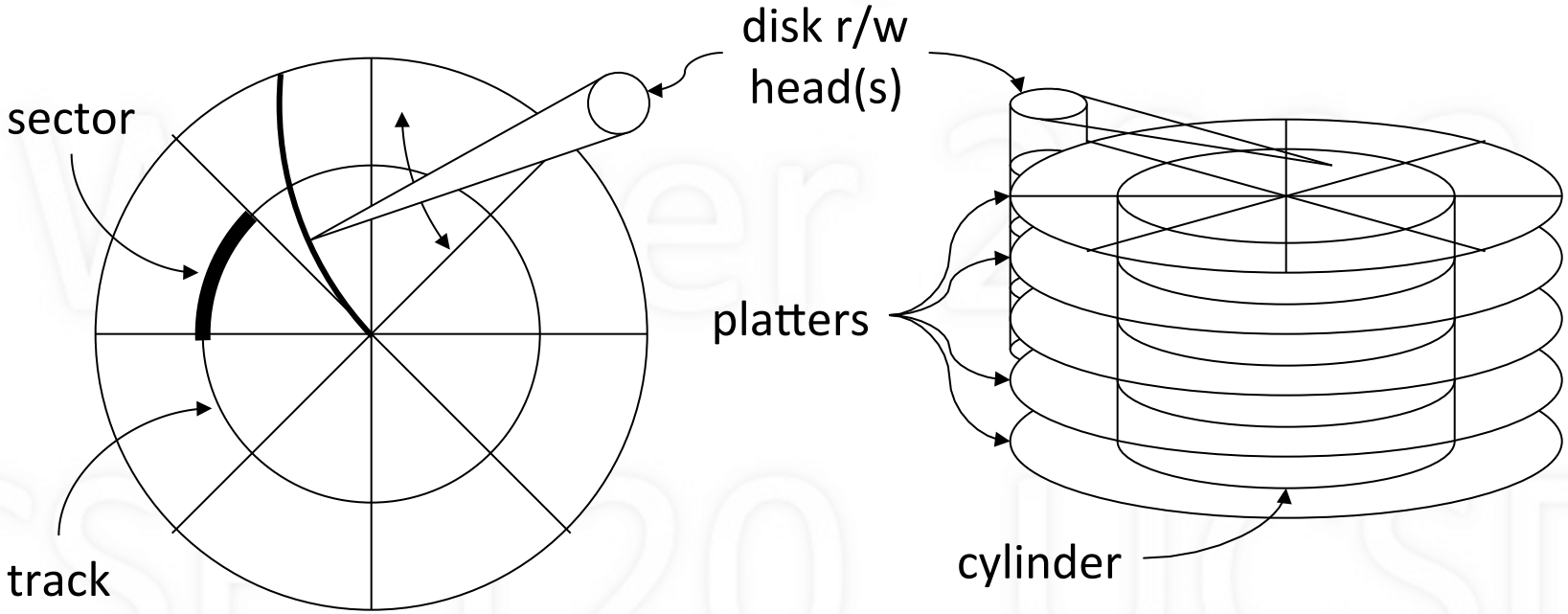


- Given pathname: /sports/baseball/Padres
  - Inode 0 block map points to data block(s) of root directory
  - Look up "sports" in root directory to get inode 22
  - Inode 22 block map points to data block(s) of sports directory
  - Look up "baseball" in sports directory to get inode 15
  - …

The Big Picture



File Systems Use Disks for Storage



- Why disks: persistent, random access, cheap
- But, disks are slow, because they are mechanical

File System Performance

- Disk accesses are time expensive: 5-20 msec!
  - Rotational latency: 2-6 msec (5400-15000 RPM)
  - Seek time: 3-13 msec
  - Transfer rate: 100+ MB/sec
- Reduce accesses by
  - reading multiple blocks in one access (read ahead)
  - maintaining a block cache
- Cluster related blocks to reduce seek time

Solid State Drives (SSD)

- NAND-based flash memory, non-volatile
- Unaffected by shock, magnetic fields; no noise
- Limited number of writes, wears out with age

Attribute	Solid State Drive	Hard Disk Drive	SDD vs HDD
Access time	0.1 msec	10 msec	** 100x **
Throughput	500 MB/sec	100 MB/sec	5x
Power	< 2 watts	5-6 watts	1/2x
Capacity	64 GB – 1 TB	500 GB – 8 TB	1/10x
Cost	$50-100 for 100 GB	$5-10 for 100 GB	10x

Performance: Caching

- Data blocks of files
- File system metadata (keep in memory)
- File metadata
  - Currently active files
  - Recently used
- Block maps
- File names
  - Name to file metadata translations

Performance: Clustering

- Blocks that exhibit locality of reference
  - Directory, and files within that directory
  - The inodes of the directory and files
- Strategy
  - Place related blocks close to each other: clustering
  - Reduces disk head movement, and thus seek time

Performance: Block Size

- What should the block size be?
  - The larger the block, the better the throughput
  - The smaller the block, the less wasted space
- Technology trends
  - Disk density is increasing faster than disk speed
  - Make disk blocks larger: 1 KB → 8 KB, 64 KB, 1 MB

Reliability: Consistency

- Buffer cache reduces disk accesses
- But if system crashes, block modifications lost
- To improve file system consistency
  - Write out modified blocks (how often?)
  - Write out critical blocks (write-through)
- Critical blocks: file system meta-data
  - Directories, i-nodes, free block lists

Reliability: Journaling

- Journal: log of file (or file system) updates
- For every update, create log entry
- Write log entry out to disk (part of journal)
- If there is a crash
  - Look at journal entries
  - Check if mods properly reflected in file system
  - Update appropriately

Summary

- File system: abstraction for object repository
- Includes both static and dynamic objects
- Name space organized hierarchically
- Access models: read/write, memory-mapped
- Supports access control: read, write, sharing
- Implementation structure: FSM, FM, Data
- Performance (caching, clustering), reliability

Textbook

- OSP: Chapter 11
- OSC: Read Chapters 13, 14
  - Lecture-related: 13.1-13.6, 14.1-14.6, 14.9
  - Recommended: 14.7-14.8, 15.1-15.9