Today

- File Systems
  - Files
  - Inodes
  - Directories

Review

- What are the goals of a file system?
- What does the file system abstract?
  - What details does the file system abstract?

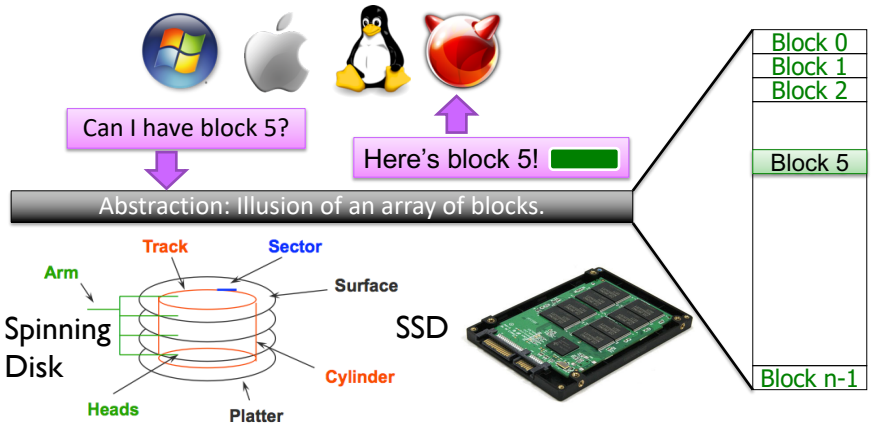
Review: File Systems

- Abstraction on top of persistent storage
  - Magnetic disk
  - SSD, USB Drive, …
- Devices provide
  - Storage that (usually) survives across machine crashes
  - Block level (random) access
  - Large capacity at low cost
  - Relatively slow performance
    - Magnetic disk read takes 10-20M processor instructions

Review: Goals for File Systems

- Long-term storage
  - Persistent: remains "forever"
- Reliable
- Large capacity, low cost
- High performance
- Named data (lookup/query)
- Controlled sharing
- Security: protecting information

Review: Storage Abstraction for File System



What is a file?

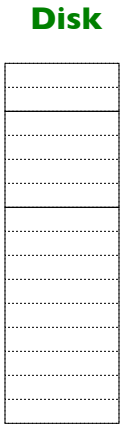
- What does the file represent?
  - What does it abstract?
- What is the state and methods associated with a file?
  - What are the requirements for the methods?

Gap in Perspective of File Systems and Disks

- User view
  - File is a named, persistent collection of data
- OS & file system view
  - File is collection of disk blocks — i.e., a container
  - File System maps file names and offsets to disk blocks

File Systems

- Disk "reality" to the OS: here's a bunch of blocks, go crazy!
- File System: Add order and structure to the blocks
  - Abstractions: files, directories, and others
  - Control how data is stored and retrieved
  - Translate high-level abstractions into low-level block requests
- There are LOTS of ways to build file systems
  - We're going to mainly focus on "traditional" structure



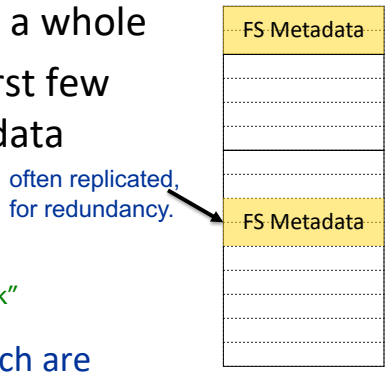
Data vs. Metadata

- **Data**: the files, directories, and other stuff for the user
  - data for the user or programs
- **Metadata**: information stored *about* the data
  - For the OS to make the file system work
  - Examples for entire FS: What type of FS is it? How large is it?
  - Example for one file: Where are the file's blocks located on disk?
- Both data and metadata stored together on disk!

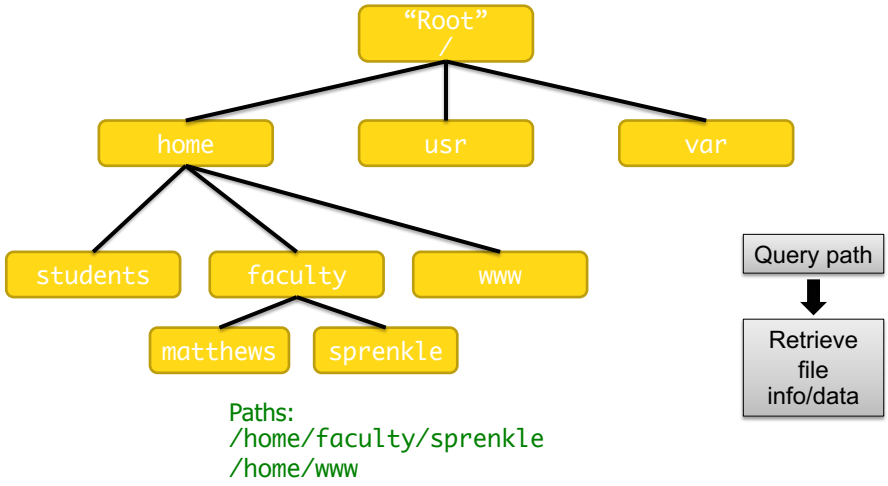
File System Metadata

- Information about the FS as a whole
- Most file systems use the first few blocks to store global metadata
- Includes:
  - Type of file system
  - Block size
  - Of the remaining blocks, which are free/in use. (% full)
    - Like the Disk Map in the project

First block: "superblock"



FS Abstraction: Hierarchical Name Space - Tree



What is a file to the file system?

- A collection of attributes:
  - Type
  - Times: creation, accessed, modified
  - Sizes: current size, maximum size
  - Structure: where is the content stored on disk?
  - Access control (permissions)
  - Others?

File – a powerful abstraction

- Documents, code
- Databases
  - Very large, possibly spanning multiple disks
- Streams
  - Input, output, keyboard, display
  - Pipes, network connections, …
- Virtual memory backing store
- Temporary repositories of OS information

- Any time you need to remember something beyond the life of a particular process/computation

What is a file?

- To the FS, a collection of attributes:
  - Type ⟸ ?
  - Times: creation, accessed, modified
  - Sizes: current size, maximum size
  - Structure: where is the content stored on disk?
  - Access control (permissions)
  - Others?

Regarding file types…

A. The OS distinguishes between file types.

B. The user distinguishes between file types.

C. Both the OS and users distinguish between file types.

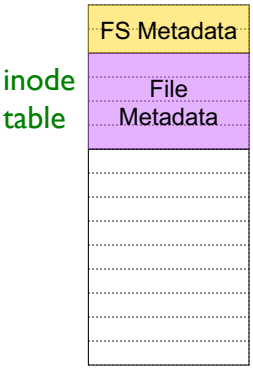
D. Nobody distinguishes between file types; files are all just files.

File Type

- To a human:
  - Is this a music file? text file? video? pdf?
  - To distinguish…
    - name the file with a suffix
    - add special "magic number" in beginning of file (e.g., Java: 0xCAFEBABE)
  - OS does NOT care what your files are. You ask for bytes, it delivers them.
- To the OS:
  - How should I interpret these bytes?
  - regular file? directory? device? FIFO (named pipe)?

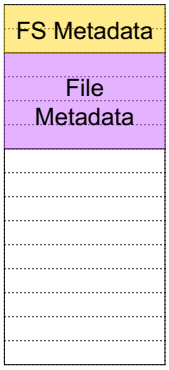
File Metadata

- Information about files
  - Everything we know about a file is encapsulated in ***inode*** structure
  - Each file has an inode
- Typically about 2-5% of blocks of disk is reserved for inodes
- Every file needs an inode. It includes:
  - Type of file (to the OS)
  - File size
  - Most recent modification time
  - Block(s) it's stored in on disk
  - Many more…
  - NOT the name of the file. (File may be linked from multiple directories with different names)



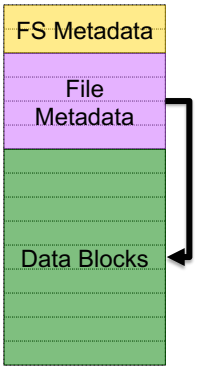
File Metadata

- Information about files
  - The inode for a file is stored on disk
    - OS/FS reads it in and keeps it in memory while the file is in active use
  - When a file is modified, the OS/FS writes changes to its inode/maps back to the disk
- Typically about 2-5% of blocks of disk is reserved for inodes
  - → There is a limit on how many inodes a file system can have → limit on how many files a file system can have



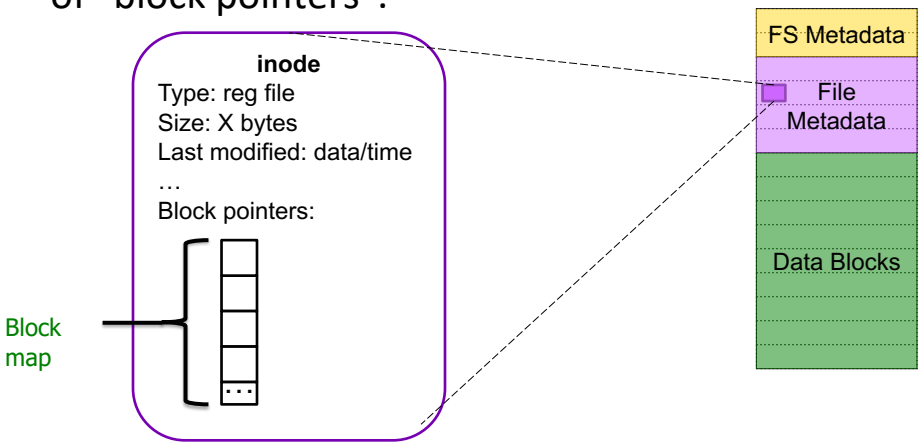
Data Blocks

- Rest of disk: data blocks
  - Stores file contents
- How do we find the data on disk?
  - File metadata (inode) stores block map
  - Once we've found the metadata for a file, it will tell us which blocks the data is in.



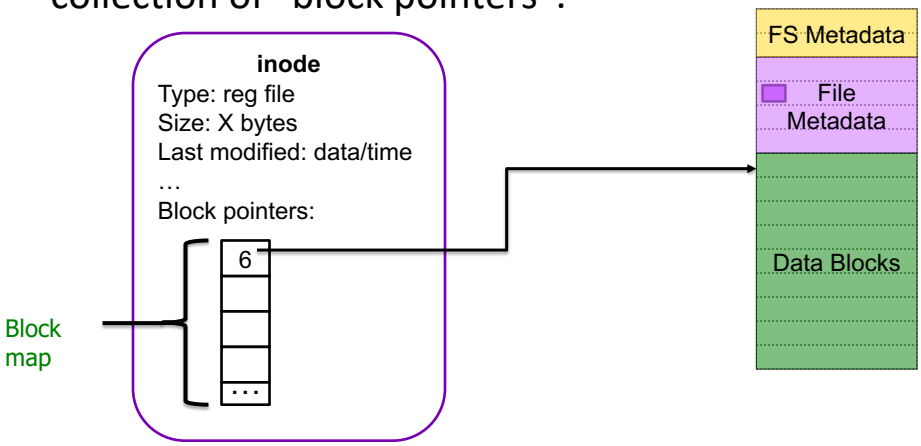
Data Blocks

- In inode structure for a file, there is a collection of "block pointers".



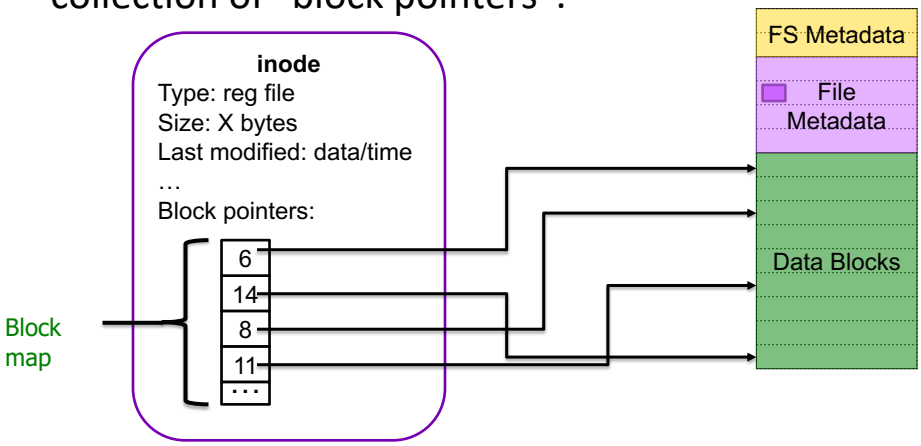
Data Blocks

- In the inode structure for a file, there is a collection of “block pointers”.



Data Blocks

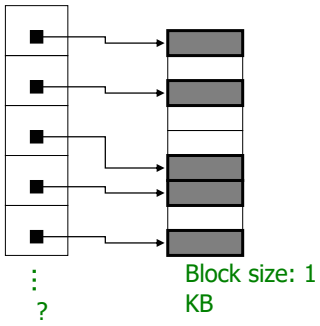
- In the inode structure for a file, there is a collection of “block pointers”.



Suppose a block is 1 KB (2^{10} bytes). How many block pointers does an inode need to support a maximum file size of 16 GB (2^{34} bytes)?

A: 2^{10} pointers B: 2^{16} pointers

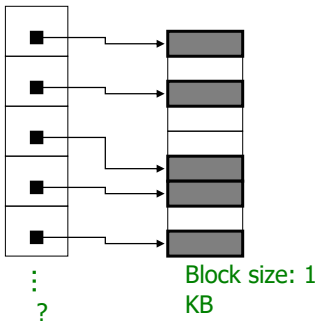
C: 2^{24} pointers D: 2^{34} pointers



Suppose a block is 1 KB (2^{10} bytes). How many block pointers does an inode need to support a maximum file size of 16 GB (2^{34} bytes)?

We need **2^{24}** block pointers.

If a pointer is 4 bytes,
that's 4 * 2^{24} = 64 MB per inode (per file)!



Most of my files are…

A. Tiny (< 1 KB)

B. Small (1 KB – 1 MB)

C. Medium (1 MB – 100 MB)

D. Large (100 MB – 1 GB)

E. Huge (> 1GB)

Discussion: Handling Dynamic File Sizes

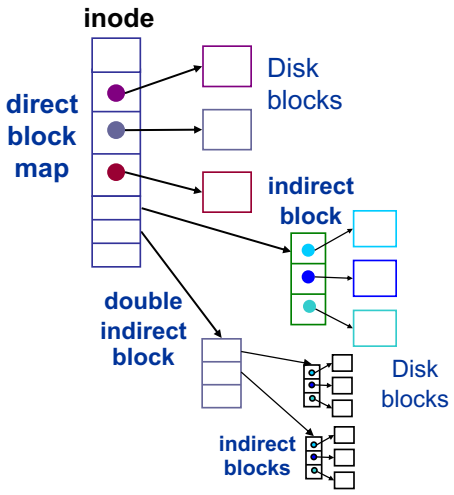
- Problem: FS doesn't know in advance how big a file is going to be, so we need a lot of block pointers in case it's big.
- Having lots of pointers makes our inodes large (e.g., 64 MB).
- Result: If we want to store a small file (e.g., 1 KB) the inode ends up wasting a lot of disk space.

What can we do about this?

Representing Large Files

Classical Unix file systems

Map some blocks to another block map



"All problems in computer science can be solved by adding a level of indirection"

Traditional Unix FS: Block Map

- inode contains 13 block map pointers (52 bytes)
  - 10 direct: references 10 data blocks
  - 1 singly-indirect: references n data blocks
    - n = number of data block *addresses* that fit into a data block
  - 1 doubly-indirect: references n^2 data blocks
  - 1 triply-indirect: references n^3 data blocks

Traditional Unix FS: Block Map

- inode contains 13 block map pointers (52 bytes)
  - 10 direct: references 10 data blocks
  - 1 singly-indirect: references n data blocks
    - n = number of data block *addresses* that fit into a data block
  - 1 doubly-indirect: references n^2 data blocks
  - 1 triply-indirect: references n^3 data blocks
- For data block of 1024 bytes
  - Assuming pointer requires 4 bytes, then n = 256
  - Max file size: $(10 + 256 + 256^2 + 256^3) * 1024 \approx 16$ GB

Skewed Tree Block maps

- Inodes are the root of a tree-structured block map
- Small files are cheap: just need the inode to map it
  - … and most files are small
- Use indirect blocks for large files
  - If can't fit into the 10 direct referenced data blocks
  - Requires another fetch for another level of map block
  - But the shift to a high branching factor covers most large files
- Double and triple indirect blocks allow very large files
- Result: inode size scales with file size
  - Consider: If a file doubles in size, how much does the inode's size increase?

Traditional Unix FS: Block Map

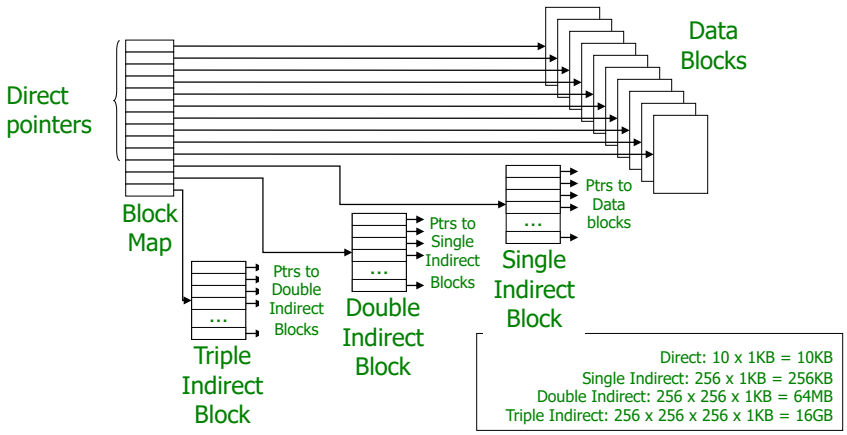
- inode contains 13 block map pointers (52 bytes per inode!)
  - 10 direct: references 10 data blocks
  - 1 singly-indirect: references n data blocks
  - 1 doubly-indirect: references n2 data blocks
  - 1 triply-indirect: references
- For data block of 1024 byte
  - Assuming pointer requires 4
  - Max file size: (10 + 256 + 25

Same max file size: 16 GB.

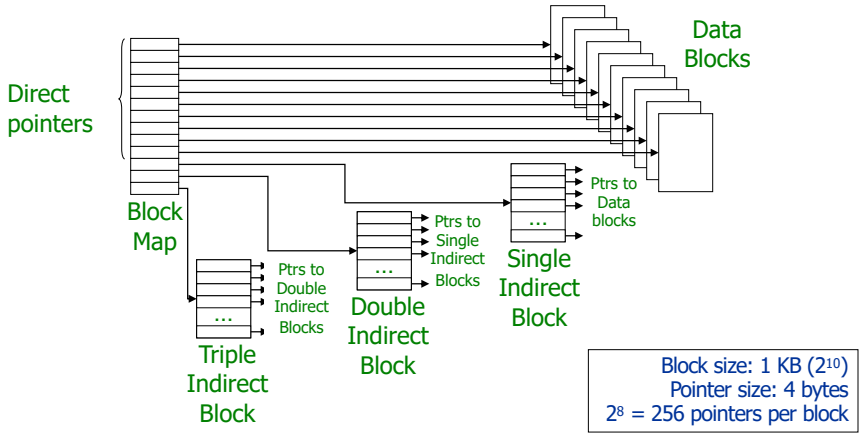
inode is now much smaller: block map is 52 bytes rather than 64 MB

Storage space for file metadata (inode's block map) now ***scales*** with file size.

Block Pointers: Multi-Level Table



Suppose we want to store a 100 KB file. How many bytes of metadata will we need for our block map?

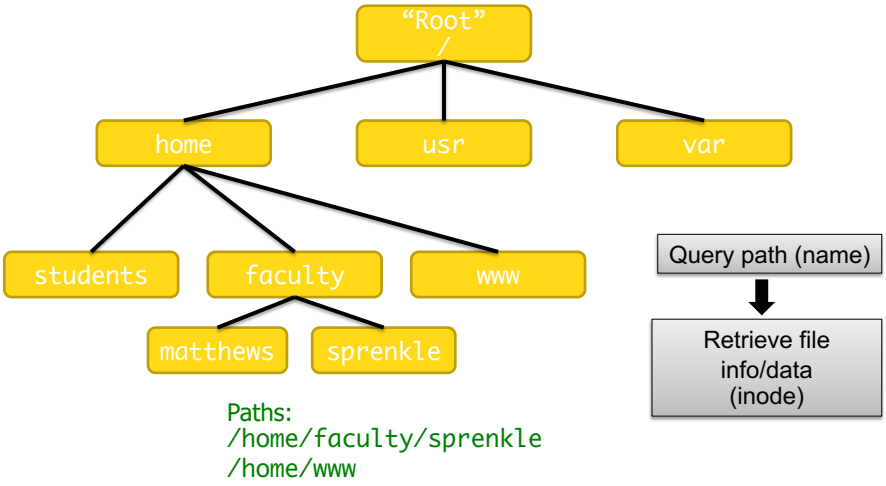


File Operations

- Create
- Write – at write pointer location
- Read – at read pointer location
- Reposition within file - seek
- Delete
- Truncate
- Open(f_i)
  - **search the directory structure** on disk for entry f_i, and move the content of entry to memory
- Close (f_i)
  - move the content of entry f_i in memory to directory structure on disk

What's in a Name?

Given the path, how do we get the inode?

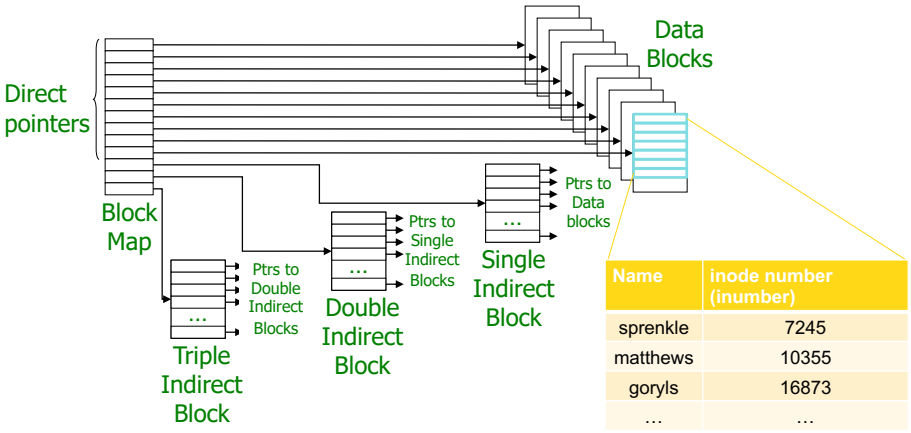


Paths:
/home/faculty/sprenkle
/home/www

Directories

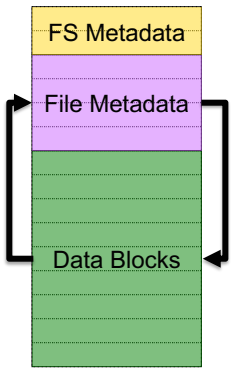
- A directory is a file
  - Just like a regular file, it has an inode with attributes and a block map
- What's different/special about a directory?
- The OS interprets the data of a directory differently
  - Rather than ignoring the data and just handing to users as it would with regular files…
  - Directory contains a collection of mappings: name → inode number

Directory

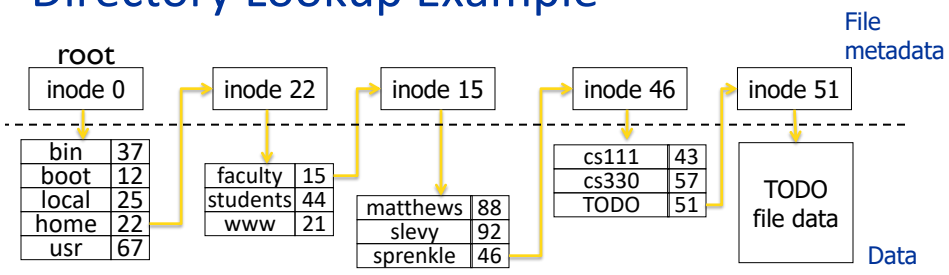


Data Blocks

- Rest of disk: data blocks
- Stores user file and directory content
  - Once we've found the metadata for a file, it will tell us which blocks the data is in
  - If the data is a *directory*, it will refer to other file metadata



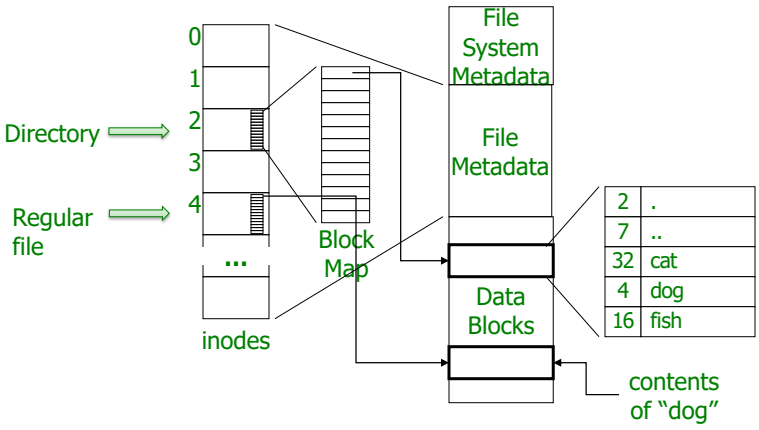
Directory Lookup Example



- Given pathname: /home/faculty/sprenkle/TODO
  1. Inode 0 block map points to data block(s) of root directory
  2. Look up "home" in root directory to get inode 22
  3. Inode 22 block map points to data block(s) of home directory
  4. Look up "faculty" in home directory to get inode 15

     …

The Big Picture



Summary

- Modern disk interface lets OS read/write numbered blocks.
- File system's goal is to add nice user abstractions on top of blocks.
- We focused on two main file types (to the OS):
  - Regular file: user data, OS doesn't care what it contains
  - Directory: maps names to inodes in the file system

Looking Ahead

- Synchronization Assignment due tonight
- Project 4 due in two Mondays
  - Monday after Thanksgiving