Files and File Systems

Major OS Themes

Virtualization

- Present physical resource as a more general, powerful, or easy-to-use form of itself
- Present illusion of multiple (or unlimited) resources where only one (or a few) really exist
- Examples: CPU, Memory, Hard drives

Concurrency

- Coordinate multiple activities to ensure correctness

Persistence

- Some data needs to survive crashes and power failures

- Need abstractions, mechanisms, policies for all

How to virtualize persistent storage?

- Recall OS Goals:
  - Convenience for the user and efficient use of machine
- Virtualization and abstraction helps with first goal
  - Files and directories abstract away the hard drive
- Efficiency:
  - File system controls when and how data is transferred to persistent storage

Outline

- Persistent storage of data
- Files
- Directories
- File Operations
- How to specify which disk blocks belong to a file

File Systems

- Provide long-term information storage
- Requirements:
  - Store very large amounts of information
  - Information must survive the termination of process using it
  - Multiple processes must be able to access info concurrently
- Two views of file systems:
  - User view – convenient logical organization of information
  - OS view – managing physical storage media, enforcing access restrictions

File Systems

- Implement an abstraction (files) for secondary storage
- Organize files logically (directories)
- Permit sharing of data between processes, people, and machines
- Protect data from unwanted access (security)

File Operations

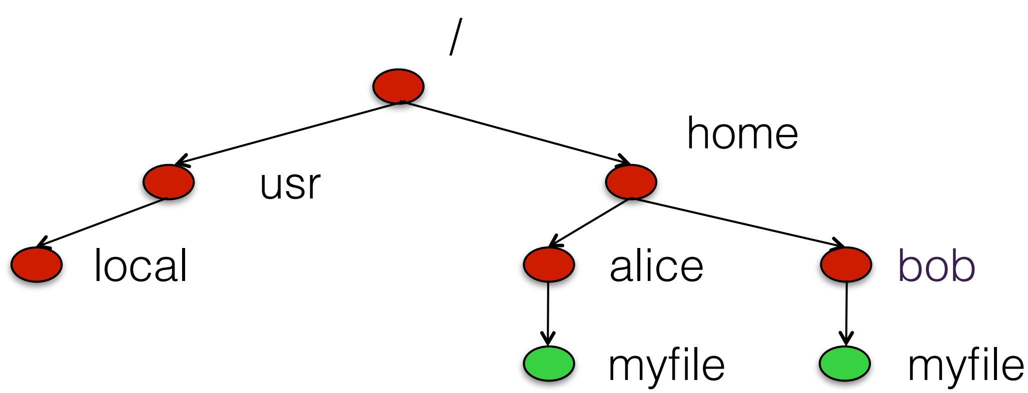
- Creation
  - Find space in file system, add entry to directory
    - map file name to location and attributes
- Writing
- Reading
  - Dominant abstraction is “file as stream”
- Repositioning within a file
- Deleting a file
- Truncation and appending
  - May erase the contents (or part of the contents) of a file while keeping attributes

File Access Methods

- General-purpose file systems support simple methods
  - Sequential access – read bytes one at a time, in order
  - Direct access – random access given block/byte number
- Database systems support more sophisticated methods
  - Record access - fixed or variable length
  - Indexed access
- What file access method(s) does Unix/Linux, Windows provide?

Directories

- Directories provide logical structure to file systems
  - For users, they provide a means to organize files
  - For the file system, they provide a convenient naming interface
    - Separates logical file organization from physical file placement
    - Stores information about files (owner, permission, etc.)
- Most file systems support multi-level directories
  - Naming hierarchies (/, /usr, /usr/local/, /home, …)



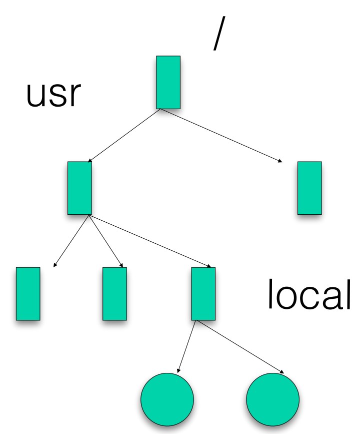
Directory Structure

- A directory is a list of entries – names and associated metadata
    - Metadata is not the data itself, but information that describes properties of the data (size, protection, location, etc.)
- List is usually unordered (effectively random)
    - Entries usually sorted by program that reads directory
- Directories typically stored in files
    - Only need to manage one kind of secondary storage unit

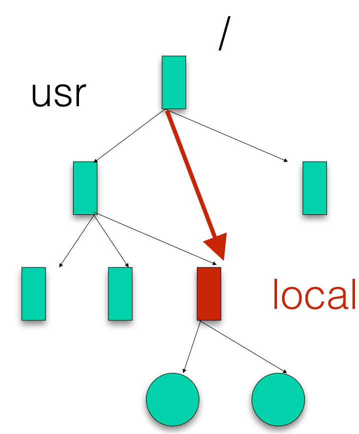
Directory Implementations

- Single-level, two-level, or tree-structured
- Acyclic-graph directories: allows for shared directories
  - The same file or subdirectory may be in 2 different directories

Tree-structured:



Acyclic graph:

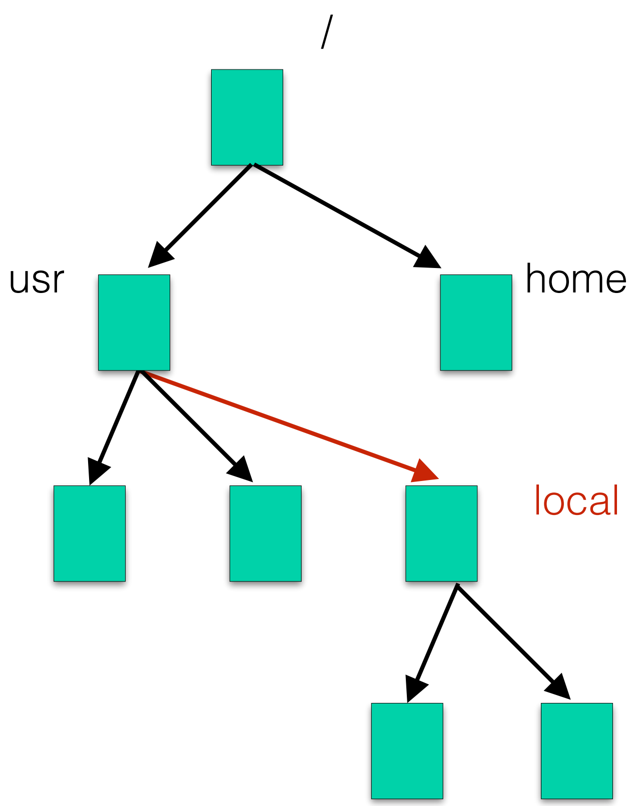


Directory Implementation

- Option 1: List
  - Simple list of file names and pointers to file metadata
  - Requires linear search to find entries
  - Easy to implement, slow to execute
    - And directory operations are frequent!
- Option 2: Hash Table
  - Create a list of file info structures
  - Hash file name to get a pointer to the file info structure in the list
  - Hash table takes space

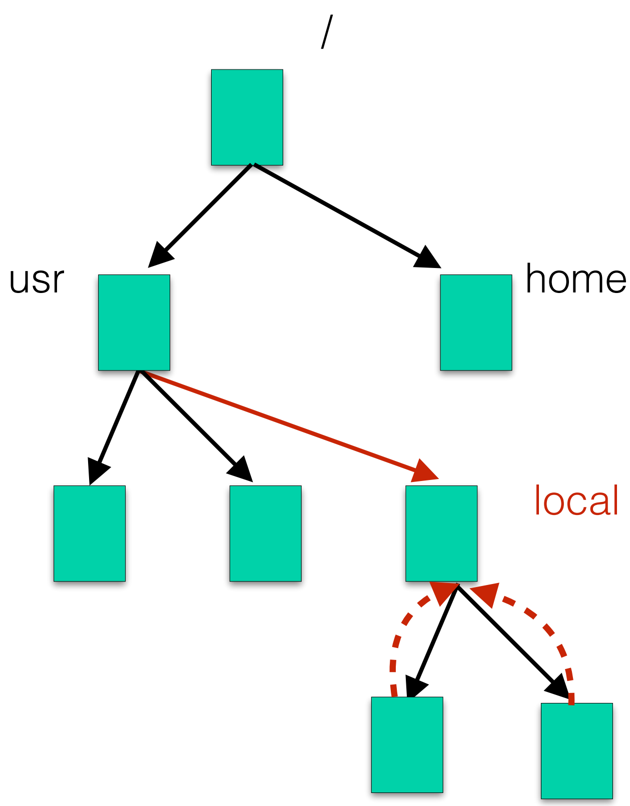
Hard Links

- In this example there is one hard link to local



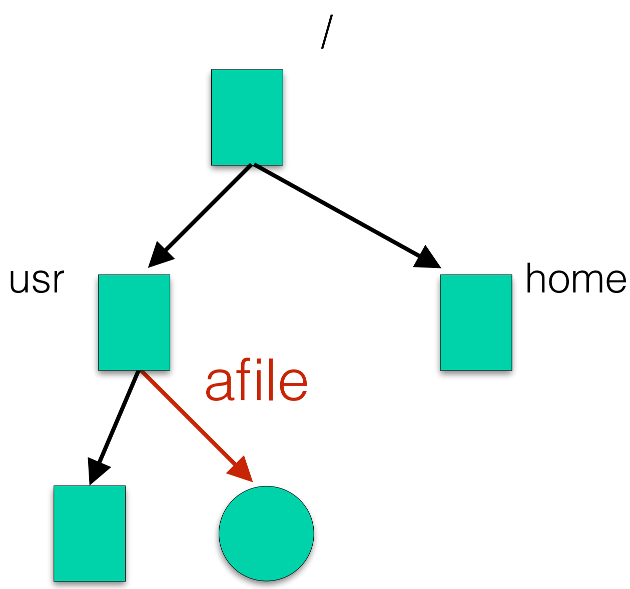
Hard Links

- Except that there really are 3 links to local because each subdirectory has a link to its parent ("..")



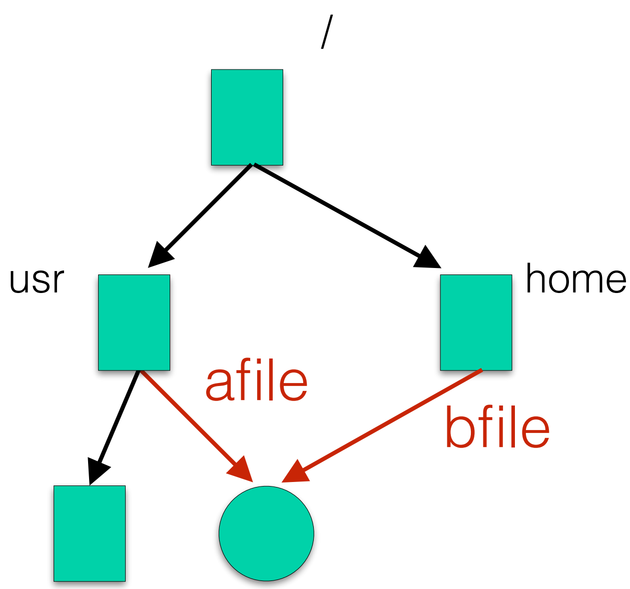
Hard Links

- The file afile has one hard link to it.



Hard Links

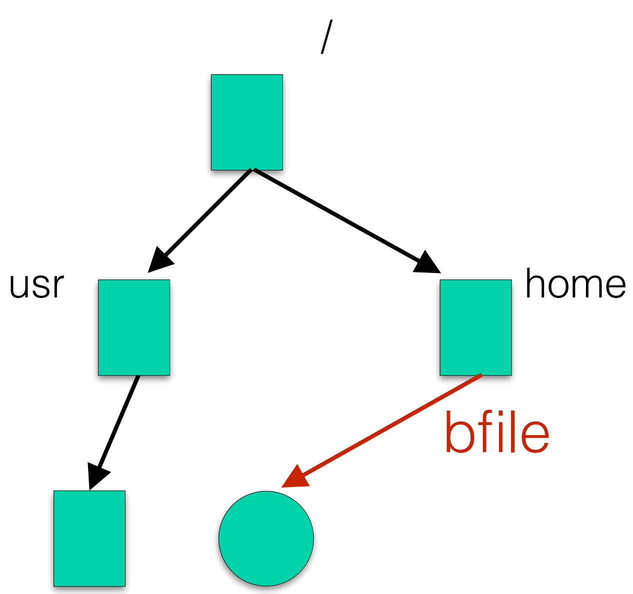
- We can create another hard link to the file using the ln command: **`ln /usr/afile /bfile`**



There is only one file, but it has two pointers (aliases) to it with potentially different names.

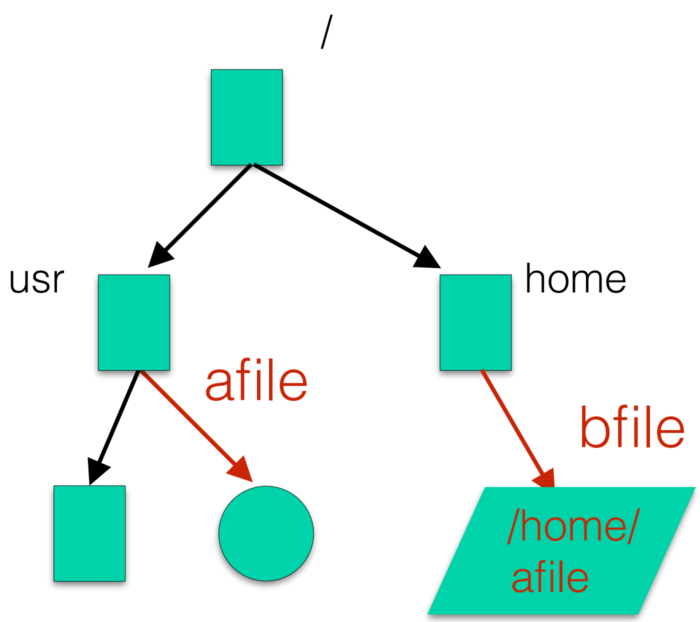
Hard Links

- If we remove afile, nothing happens to bfile.



Symbolic link

- Let's create a symbolic link instead:
  `ln -s /usr/afile /bfile`

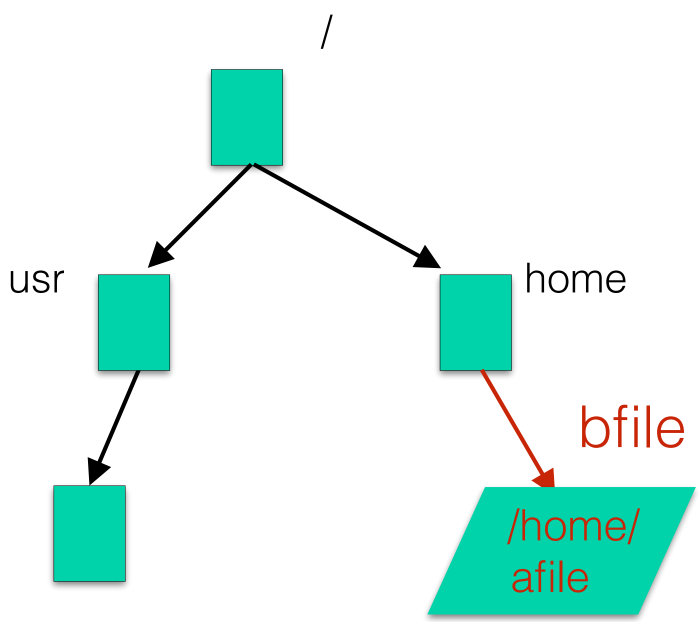


Instead of creating a new hard link, a special kind of file is created that contains the path to the linked file.

The file system code knows to use the path to find the contents.

Symbolic link

- Now what happens if we remove afile?



afile is removed and we have a "dangling pointer"

Hard links vs Symbolic Links

Hard Links

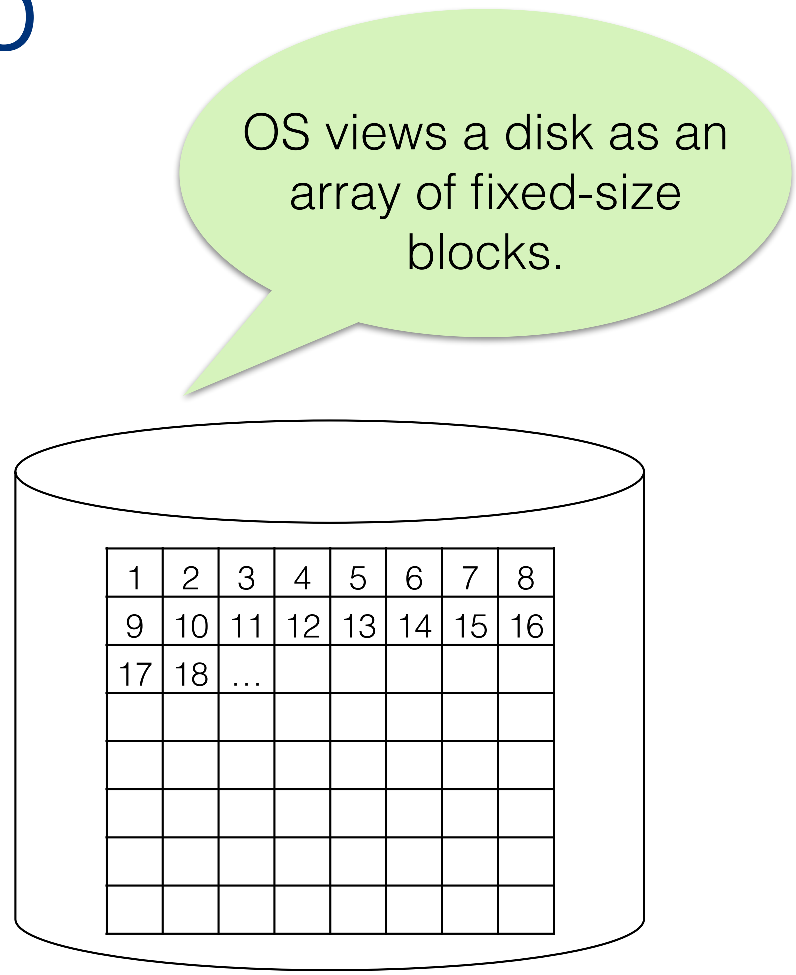
- Can’t create new hard links to directories (might create cycles)
- Can’t create hard links across partitions.
- No extra processing to follow hard links.
- Removing a hard link only removes the file if it is the last link to the file.

Symbolic Links

- Can create symbolic links to directories
- Can create symbolic links across partitions
- File system needs to look up link to follow it
- Removing a file may lead to a dangling link.

Next Up

- Implementing File Systems
  - Disk layout
  - File metadata
  - Directory data



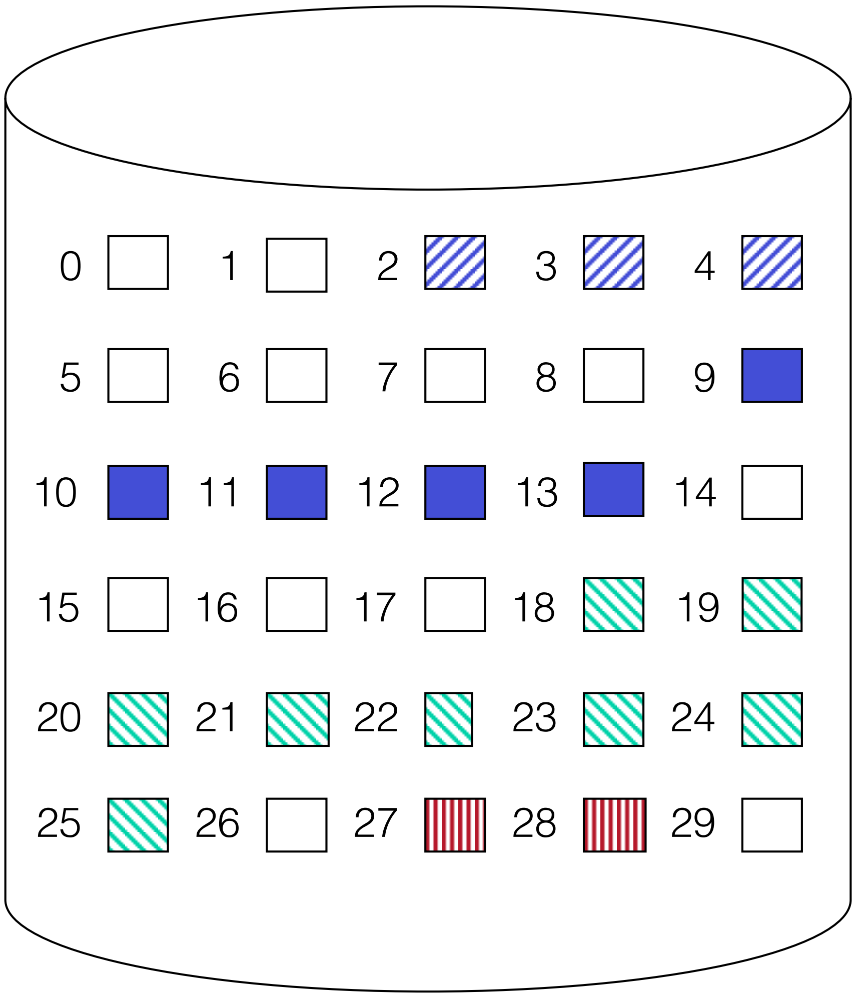
Associating Data Blocks to Files

- We need a strategy to store which disk blocks belong to a file.

- Possibilities:
  - Contiguous Allocation
  - Linked Allocation (FAT)
  - Indexed Allocation (FFS, EXT2/3/4 — Inodes)
  - Extent-based Allocation (NTFS, Assignment 1)

- We also need somewhere to store the file metadata
  - permissions, last modified time, owner, type, …

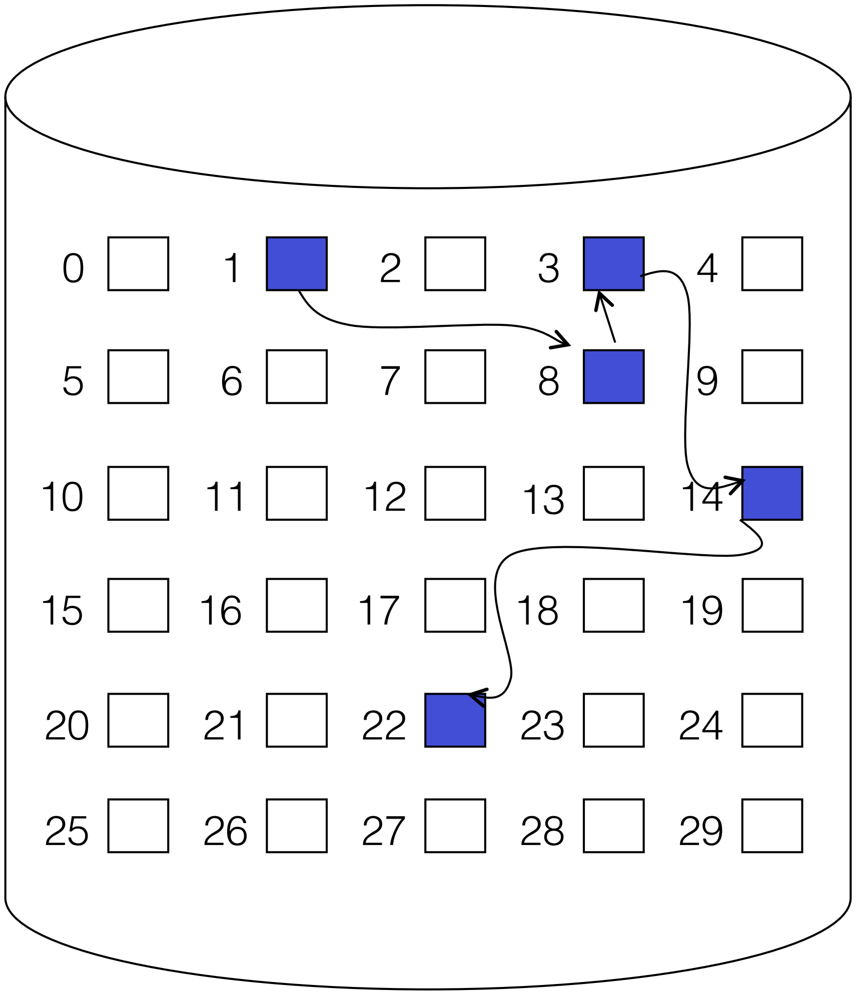
Contiguous Allocation



File Name	Start Blk	Length
File A	2	3
File B	9	5
File C	18	8
File D	27	2

Each file is given a contiguous set of blocks.
Store the starting block and length in the directory.

Linked Allocation

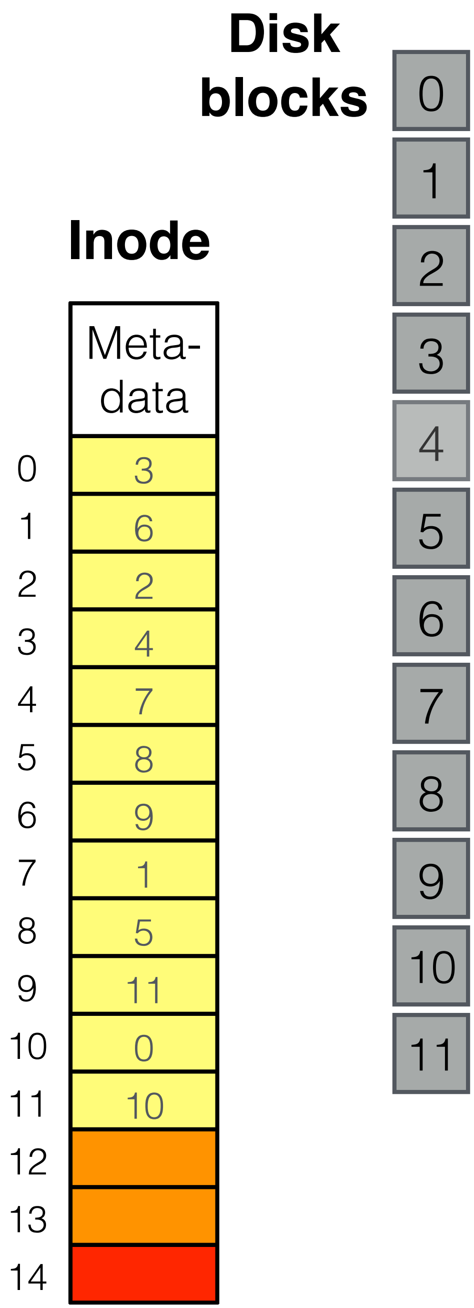


File Name	Start Blk	Last Blk
...	...	...
File B	1	22
...	...	...

Store the starting block and length in the directory.
Each block in a file contains a pointer to the next block
FAT is similar — We will discuss later

Indexed Allocation (Inodes)

- Unix inodes implement an indexed structure for files
- All file metadata is stored in an inode
  - Unix directory entries map file names to inodes
- Each inode contains 15 block pointers
  - First 12 are direct block pointers
    - Disk addresses of first 12 data blocks in file
  - The 13th is a single indirect block pointer
    - Address of block containing addresses of data blocks
  - Then the 14th is a double indirect block pointer
    - Address of block containing addresses of single indirect blocks
  - Finally, the 15th is a triple indirect block pointer



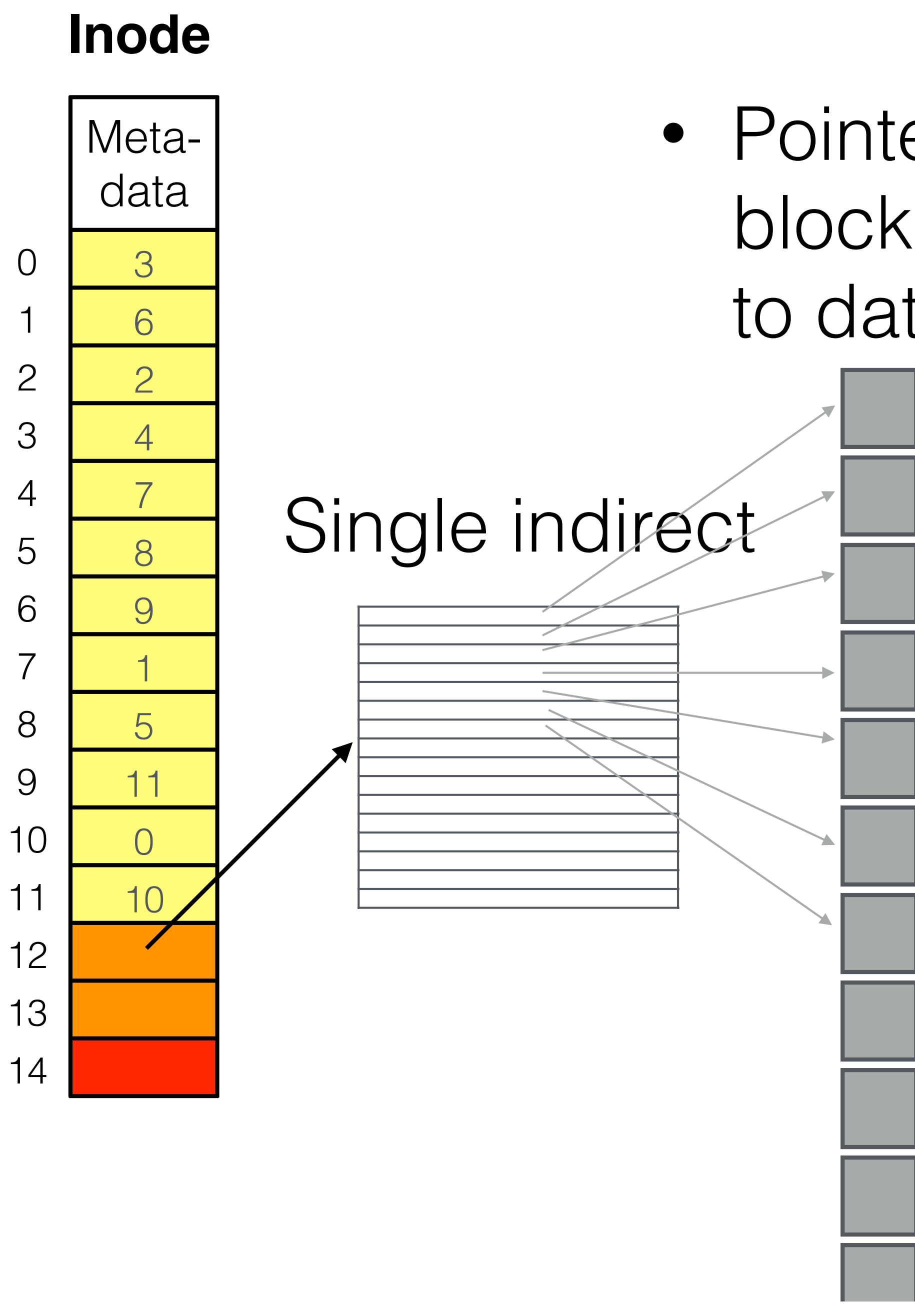
Inodes

- Each file and directory is represented by an inode
- First 12 pointers point directly to data blocks
  - 0th pointer points to the first block of the file
  - 11th pointer points to the 12th block of the file.
- The blocks do not need to be contiguous.

Remember, a disk is viewed an array of blocks. So the "pointer" is really an index into the array of disk blocks

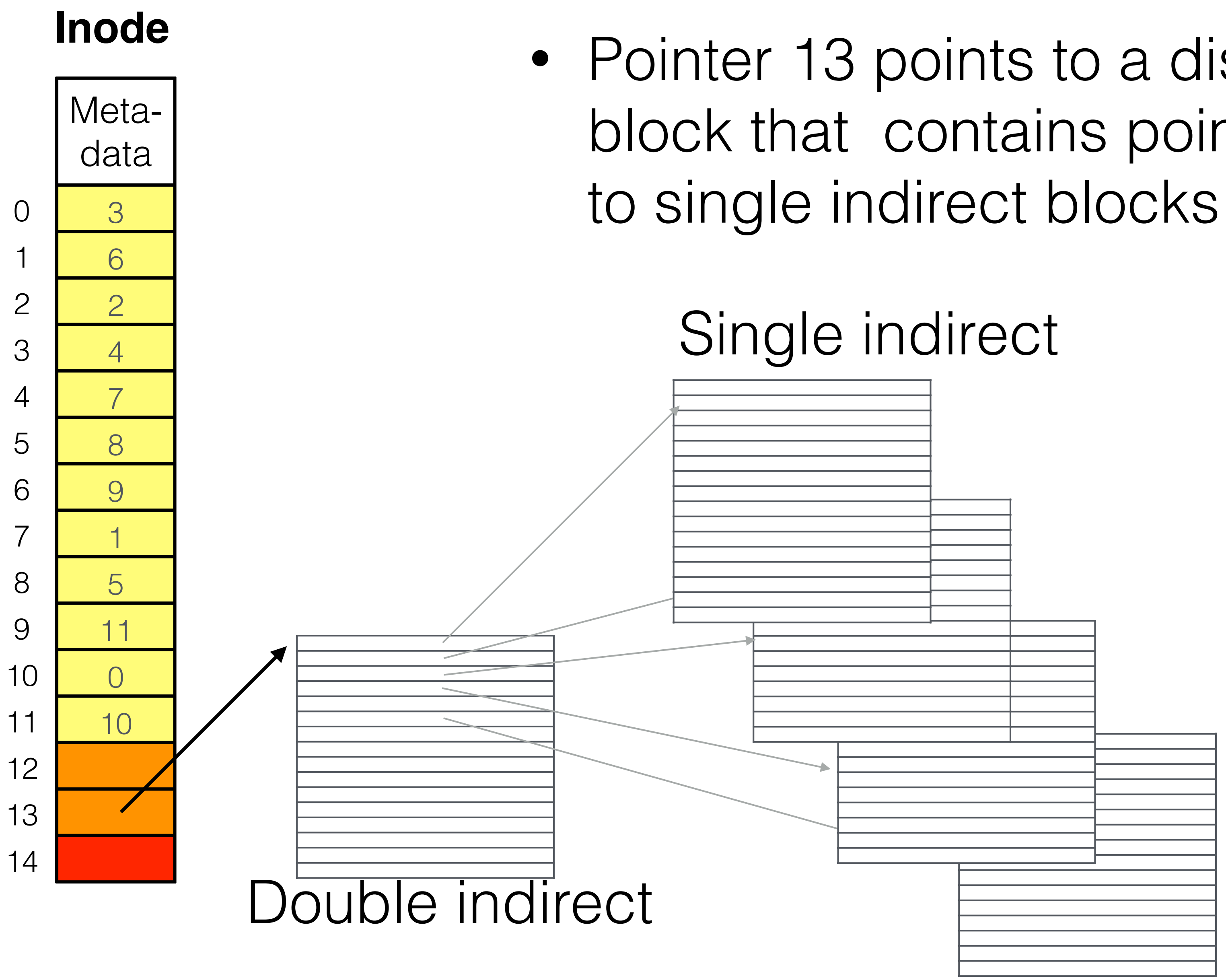
Inodes

- Pointer 12 points to a disk block that contains pointers to data blocks.



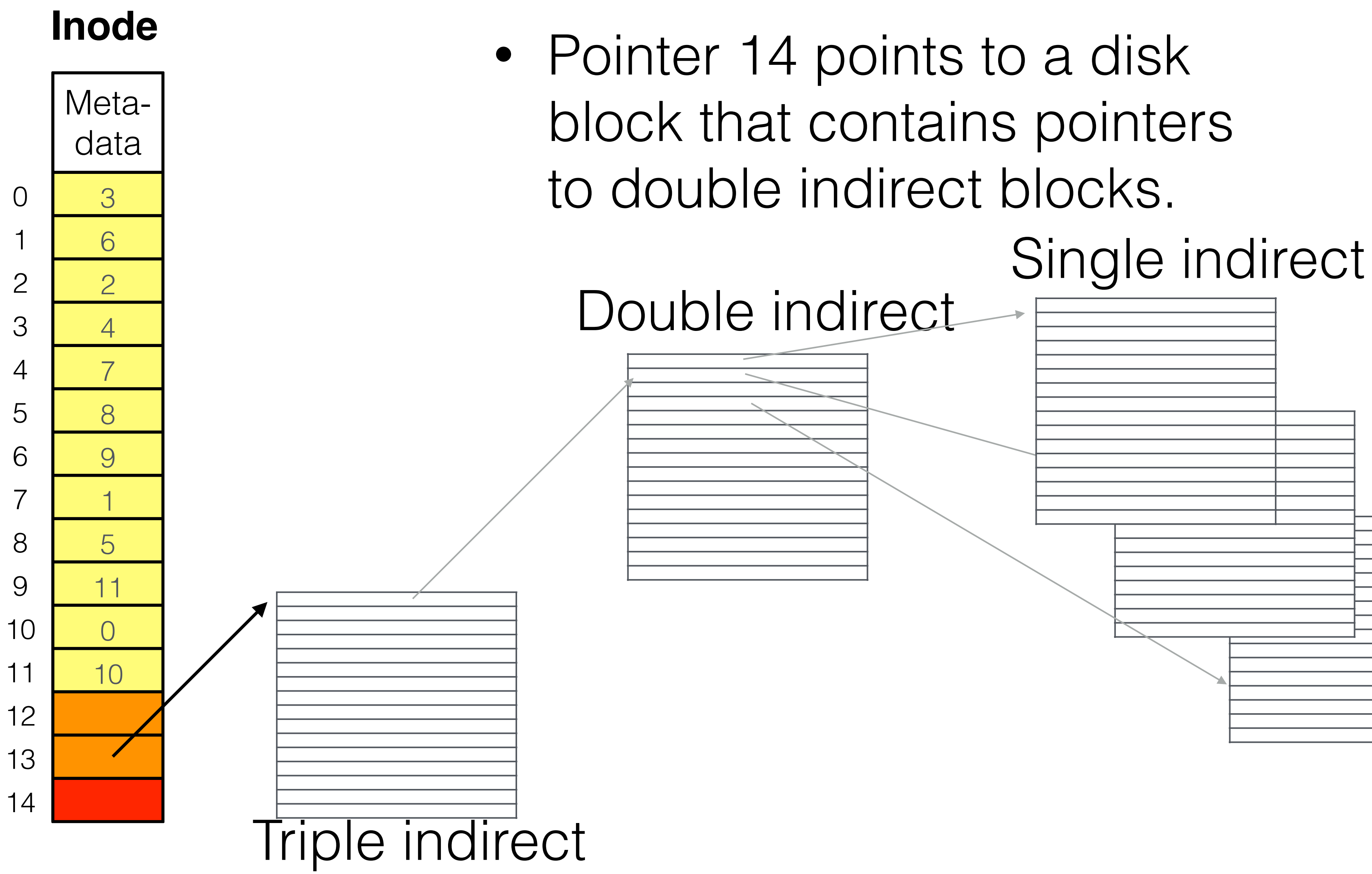
Inodes

- Pointer 13 points to a disk block that contains pointers to single indirect blocks.



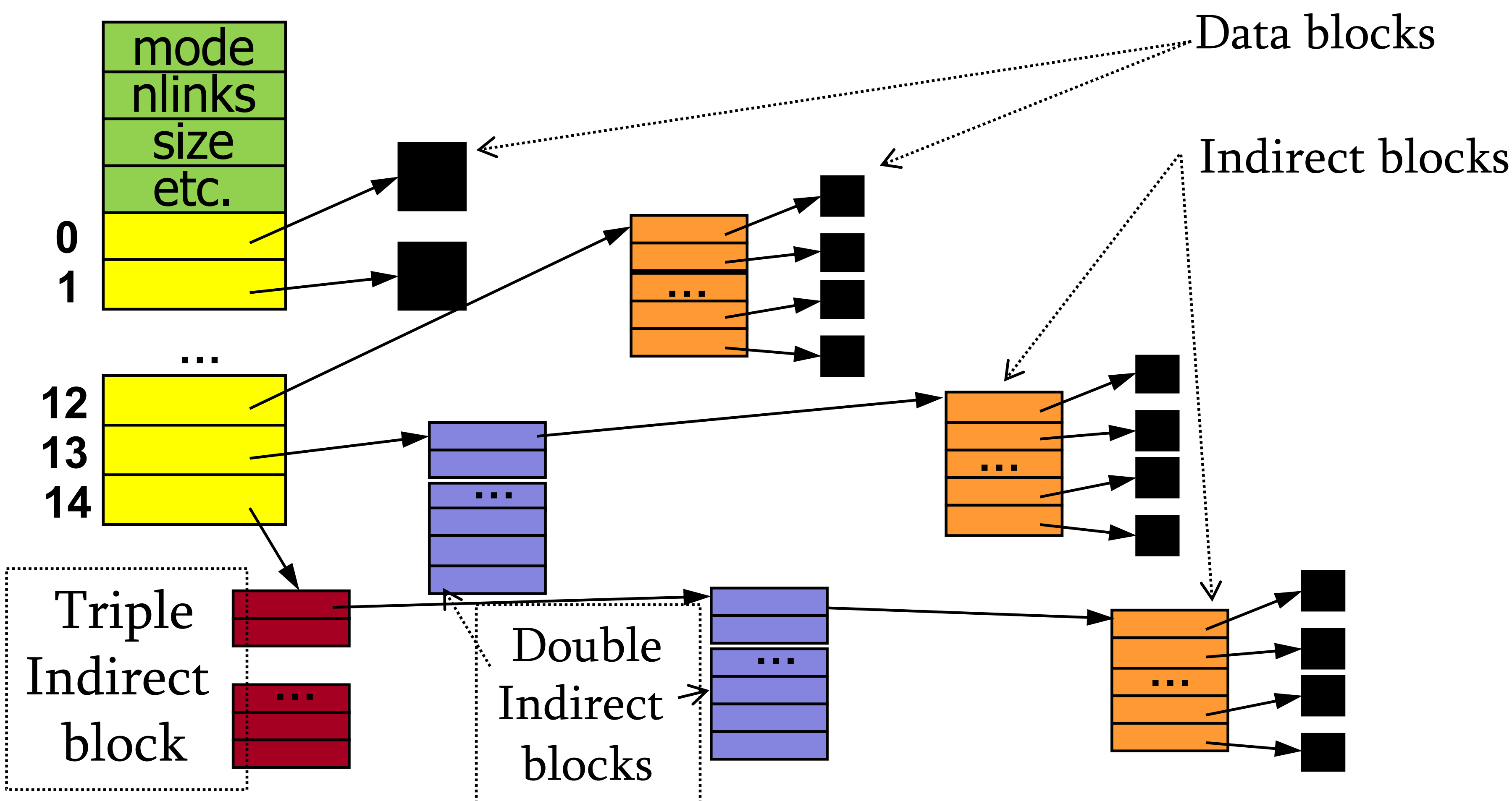
Inodes

- Pointer 14 points to a disk block that contains pointers to double indirect blocks.



Putting it all together

- Ext2 Linux file system Inodes are 128 bytes



Disk Layout Strategies

- Files often span multiple disk blocks
- How do you find all of the blocks for a file?
  1. Contiguous allocation
     - Fast, simplifies directory access and allows indexing
     - Inflexible, causes external fragmentation, requires compaction
  2. Linked, or chained, structure
     - Each block points to the next, directory points to the first
     - Good for sequential (streaming) access, bad for all others
  3. Indexed structure (kind of like address translation)
     - An “index block” contains pointers to many other blocks
     - Handles random access better, still good for sequential
     - May require multiple, linked index blocks