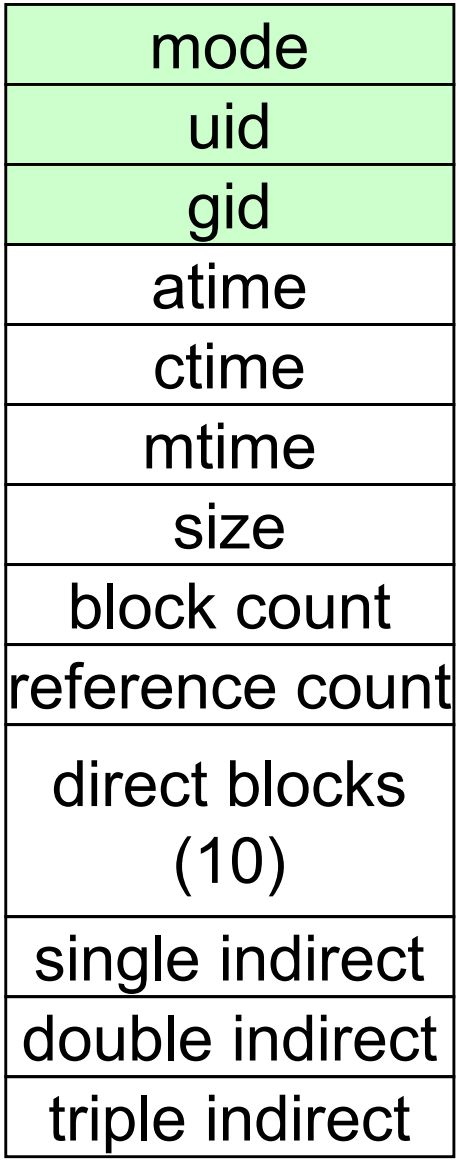
# UNIX File Management

- We will focus on two types of files
  - Ordinary files (stream of bytes)
  - Directories
- And mostly ignore the others
  - Character devices
  - Block devices
  - Named pipes
  - Sockets
  - Symbolic links

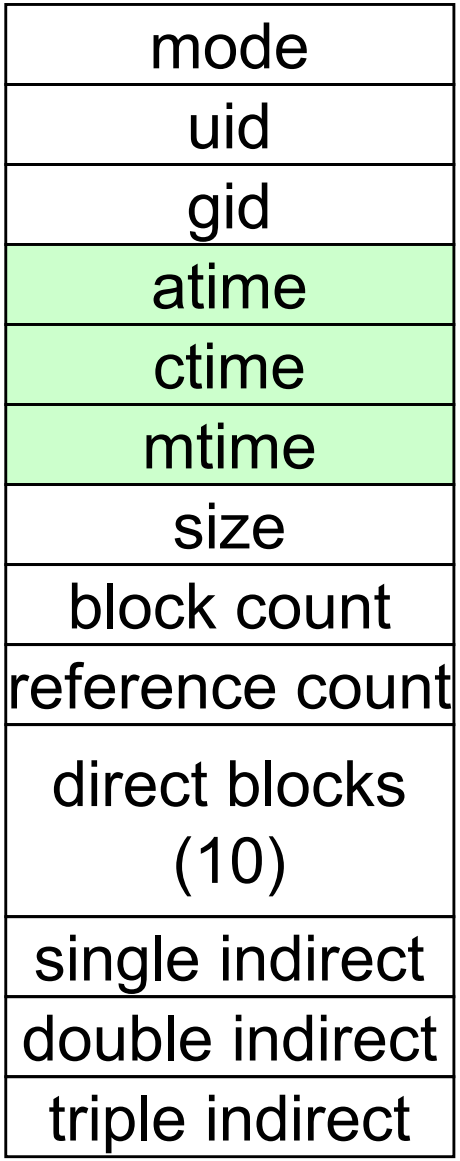
# UNIX index node *(inode)*

- Each file is represented by an Inode
- Inode contains all of a file's metadata
  - Access rights, owner,accounting info
  - (partial) block index table of a file
- Each inode has a unique number (within a partition)
  - System oriented name
  - Try 'ls –i' on Unix (Linux)
- Directories map file names to inode numbers
  - Map human-oriented to system-oriented names
  - Mapping can be *many-to-one*
    - Hard links



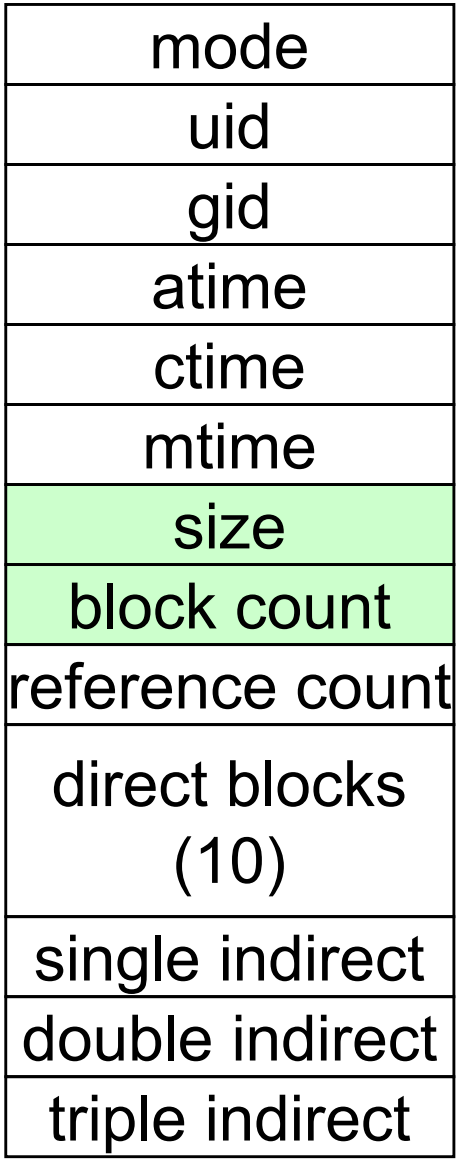

# Inode Contents

- Mode
  - Type
    - Regular file or directory
  - Access mode
    - rwxrwxrwx
- Uid
  - User ID
- Gid
  - Group ID




# Inode Contents

- atime
  - Time of last access
- ctime
  - Time when file was created
- mtime
  - Time when file was last modified



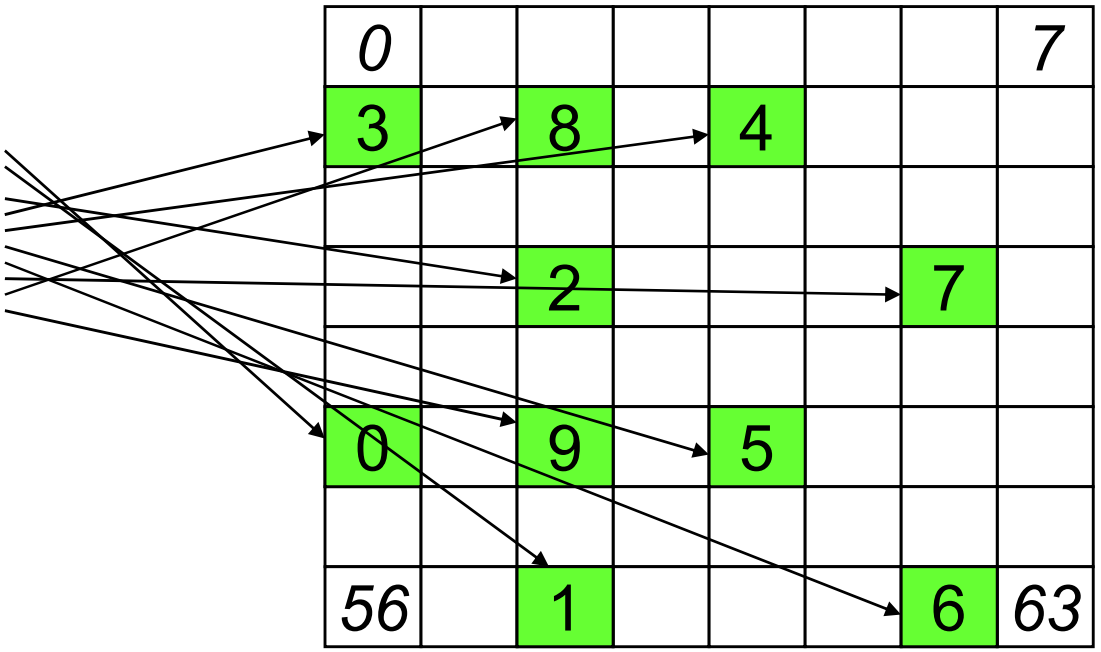

# Inode Contents

- Size
  - Size of the file in bytes
- Block count
  - Number of disk blocks used by the file.
- Note that number of blocks can be much less than expected given the file size
  - Files can be sparsely populated
    - E.g. write(f,“hello”); lseek(f, 1000000); write(f, “world”);
    - Only needs to store the start an end of file, not all the empty blocks in between.
      - Size = 1000005
      - Blocks = 2 + overheads

| mode |
| --- |
| uid |
| gid |
| atime |
| ctime |
| mtime |
| size |
| block count |
| reference count |
| direct blocks (10)<br>40,58,26,8,12,<br>44,62,30,10,42 |
| single indirect |
| double indirect |
| triple indirect |

# Inode Contents

- Direct Blocks
  - Block numbers of first 10 blocks in the file
  - Most files are small
    - We can find blocks of file *directly* from the inode




Disk

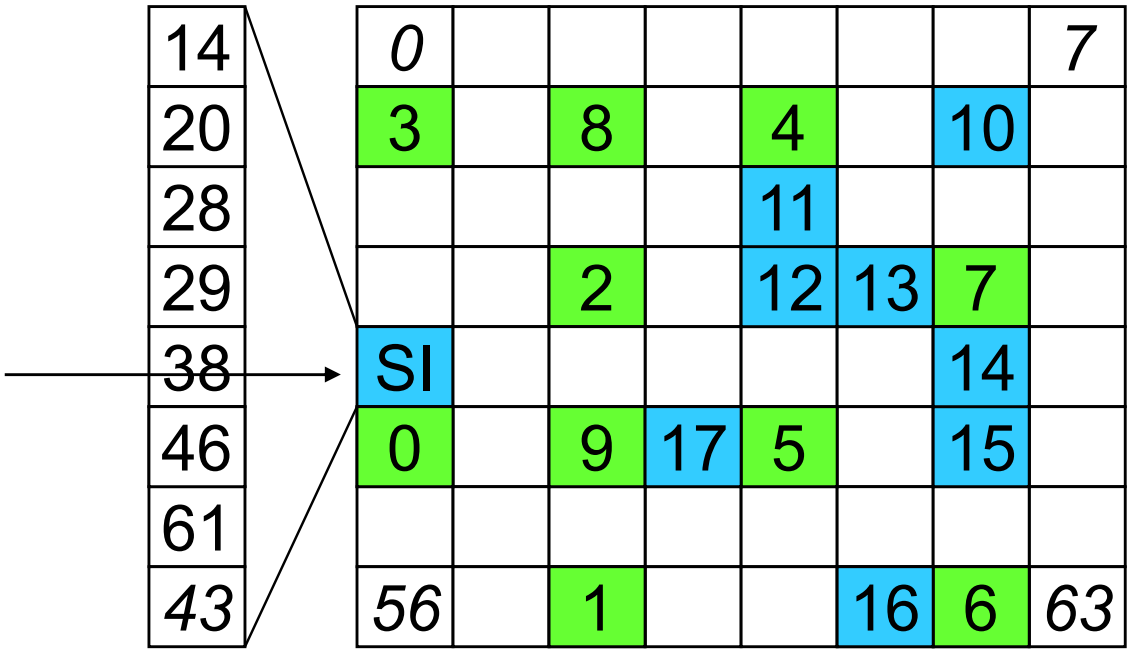
# Problem

- How do we store files greater than 10 blocks in size?
  - Adding significantly more direct entries in the inode results in many unused entries most of the time.

# Inode Contents

| mode |
|---|
| uid |
| gid |
| atime |
| ctime |
| mtime |
| size |
| block count |
| reference count |
| direct blocks (10) 40,58,26,8,12, 44,62,30,10,42 |
| single indirect: 32 |
| double indirect |
| triple indirect |

- Single Indirect Block
  - Block number of a block containing block numbers
    - In this case 8




Disk

# Single Indirection

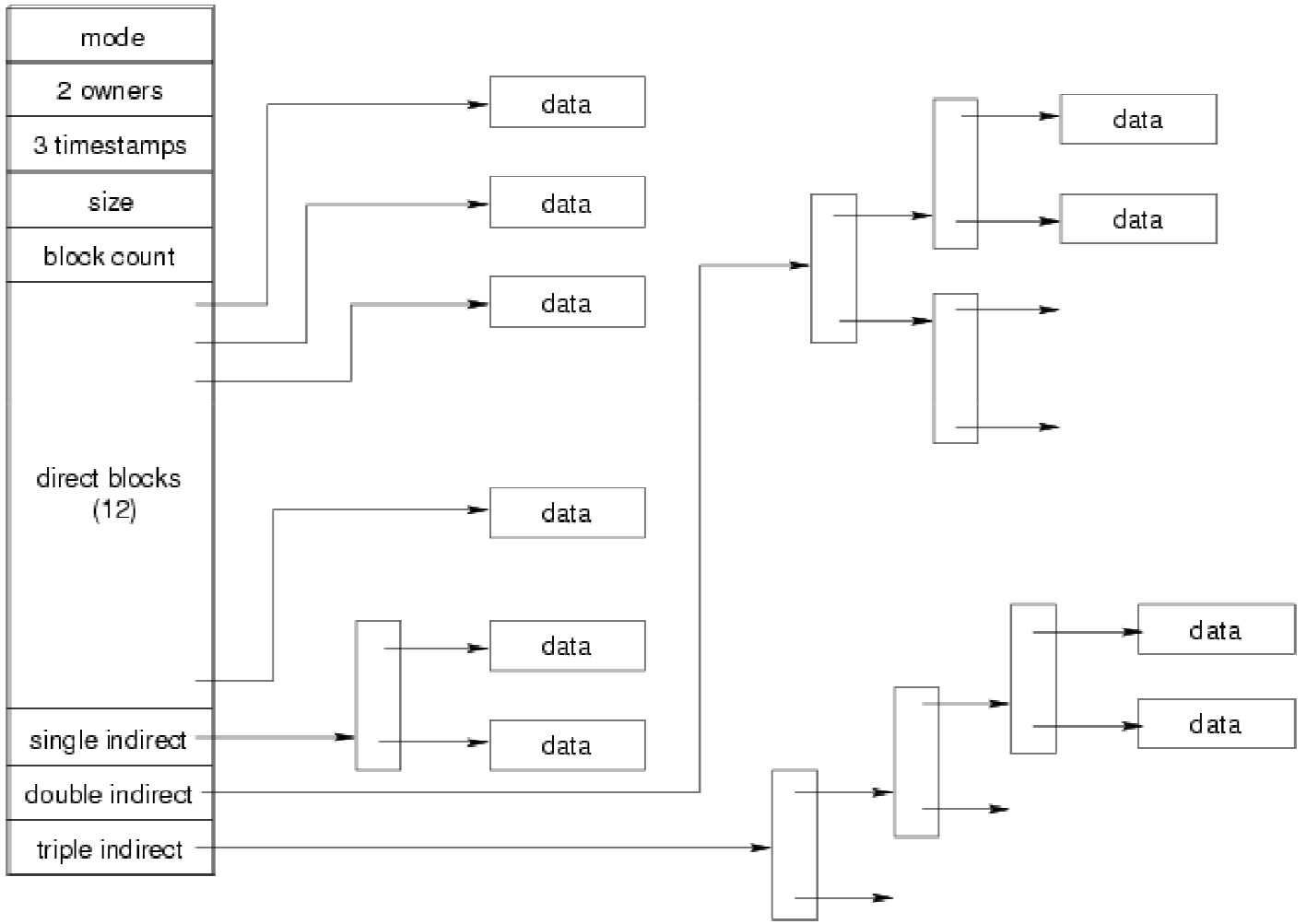
- Requires two disk access to read
  - One for the indirect block; one for the target block
- Max File Size
  - In previous example
    - 10 direct + 8 indirect = 18 block file
  - A more realistic example
    - Assume 1Kbyte block size, 4 byte block numbers
    - 10 * 1K + 1K/4 * 1K = 266 Kbytes
- For large majority of files (< 266 K), only one or two accesses required to read any block in file.

| mode |
| --- |
| uid |
| gid |
| atime |
| ctime |
| mtime |
| size |
| block count |
| reference count |
| direct blocks (10)<br>40,58,26,8,12,<br>44,62,30,10,42 |
| single indirect: 32 |
| double indirect |
| triple indirect |

# Inode Contents

- Double Indirect Block
  - Block number of a block containing block numbers of blocks containing block numbers
- Triple Indirect
  - Block number of a block containing block numbers of blocks containing block numbers of blocks containing block numbers ☺

# Unix Inode Block Addressing Scheme



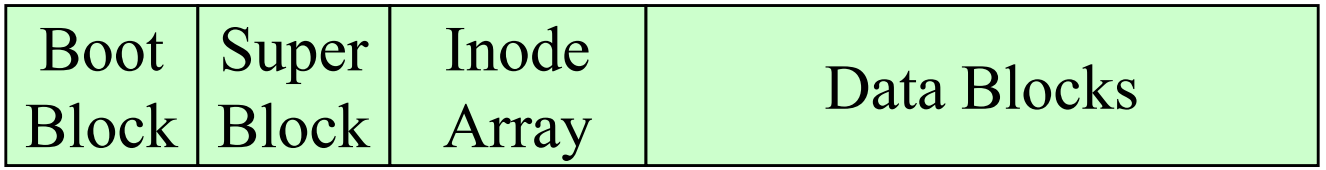
# Max File Size

- Assume 4 bytes block numbers and 1K blocks
- The number of addressable blocks
  - Direct Blocks = 12
  - Single Indirect Blocks = 256
  - Double Indirect Blocks = 256 * 256 = 65536
  - Triple Indirect Blocks = 256 * 256 * 256 = 16777216
- Max File Size
  - 12 + 256 + 65536 + 16777216 = 16843020 = 16 GB

# Inode Summary

- The inode contains the on disk data associated with a file
  - Contains mode, owner, and other bookkeeping
  - Efficient random and sequential access via *indexed allocation*
  - Small files (the majority of files) require only a single access
  - Larger files require progressively more disk accesses for random access
    - Sequential access is still efficient
  - Can support really large files via increasing levels of indirection

# Where/How are Inodes Stored




- System V Disk Layout (s5fs)
  - Boot Block
    - contain code to bootstrap the OS
  - Super Block
    - Contains attributes of the file system itself
      - e.g. size, number of inodes, start block of inode array, start of data block area, free inode list, free data block list
  - Inode Array
  - Data blocks

# Some problems with s5fs

- Inodes at start of disk; data blocks end
  - Long seek times
    - Must read inode before reading data blocks
- Only one superblock
  - Corrupt the superblock and entire file system is lost
- Block allocation suboptimal
  - Consecutive free block list created at FS format time
    - Allocation and de-allocation eventually randomises the list resulting the random allocation
- Inodes allocated randomly
  - Directory listing resulted in random inode access patterns

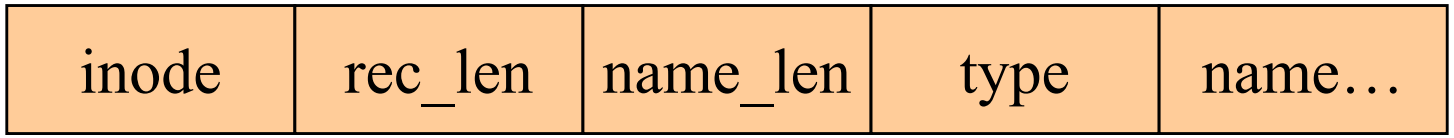
# Superblocks

- Size of the file system, block size and similar parameters
- Overall free inode and block counters
- Data indicating whether file system check is needed:
  - Uncleanly unmounted
  - Inconsistency
  - Certain number of mounts since last check
  - Certain time expired since last check
- Replicated to provide redundancy to add recoverability

# Thus far…

- Inodes representing files laid out on disk.
- Inodes are referred to by number!!!
  - How do users name files? By number?
  - Try ls –i to see how useful inode numbers are….

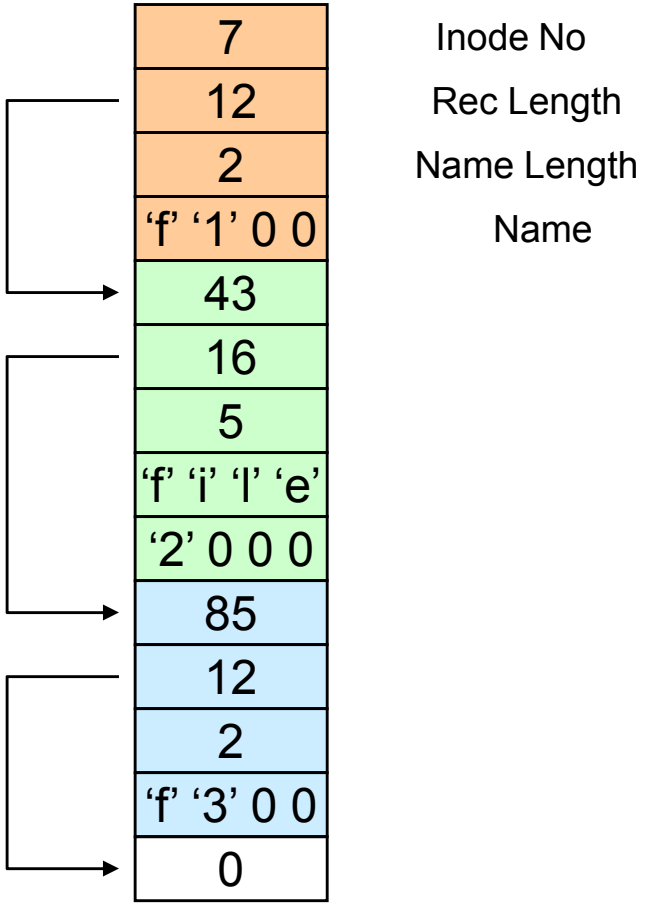
# Ext2fs Directories




- Directories are files of a special type
  - Consider it a file of special format, managed by the kernel, that uses most of the same machinery to implement it
    - Inodes, etc…
- Directories translate names to inode numbers
- Directory entries are of variable length
- Entries can be deleted in place
  - inode = 0
  - Add to length of previous entry
  - use null terminated strings for names

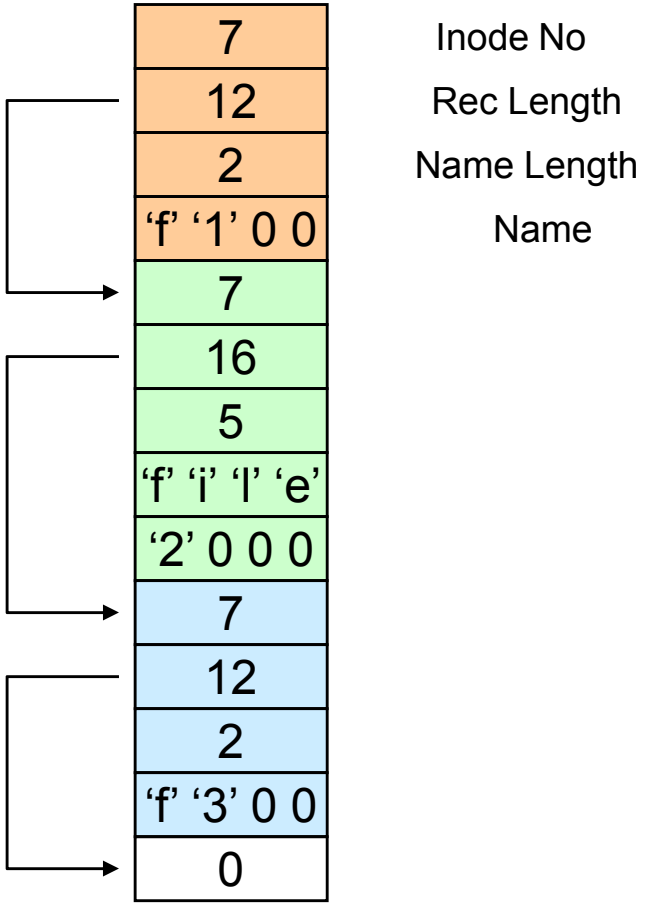
# Ext2fs Directories

- “f1” = inode 7
- “file2” = inode 43
- “f3” = inode 85



# Ext2fs Directories

- Note that inodes can have more than one name
  - Called a *Hard Link*
  - Inode (file) 7 has three names
    - "f1" = inode 7
    - "file2" = inode 7
    - "f3" = inode 7



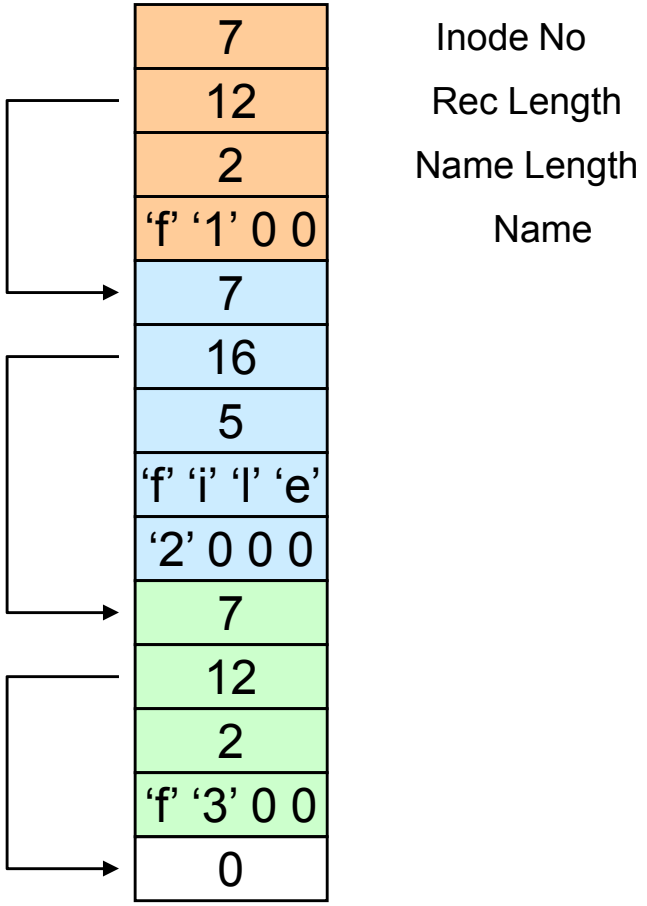
| mode |
| --- |
| uid |
| gid |
| atime |
| ctime |
| mtime |
| size |
| block count |
| reference count |
| direct blocks (10)<br>40,58,26,8,12,<br>44,62,30,10,42 |
| single indirect: 32 |
| double indirect |
| triple indirect |

# Inode Contents

- We can have many name for the same inode.
- When we delete a file by name, i.e. remove the directory entry (link), how does the file system know when to delete the underlying inode?
  - Keep a *reference count* in the inode
    - Adding a name (directory entry) increments the count
    - Removing a name decrements the count
    - If the reference count == 0, then we have no names for the inode (it is unreachable), we can delete the inode (underlying file or directory)

# Ext2fs Directories

- Deleting a filename
  - rm file2



# Ext2fs Directories

- Deleting a filename
  - rm file2
- Adjust the record length to skip to next valid entry