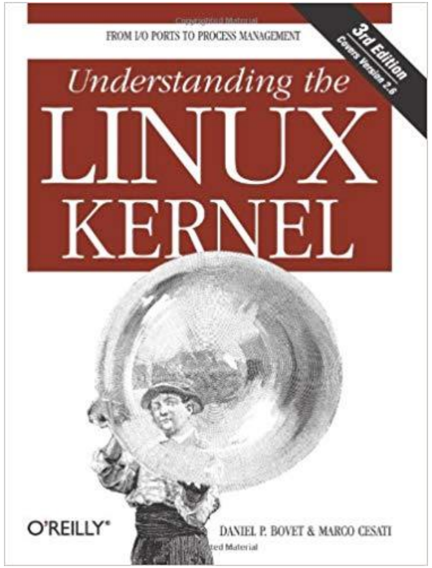
ARIZONA STATE UNIVERSITY

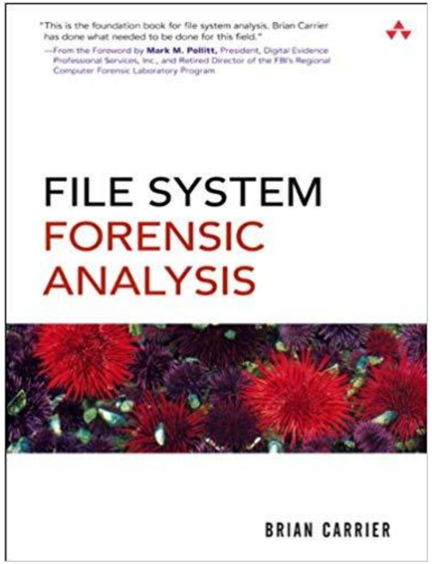
CSE 469: Computer and Network Forensics

Topic 4: File Systems

Dr. Mike Mabey | Spring 2019

My Sources





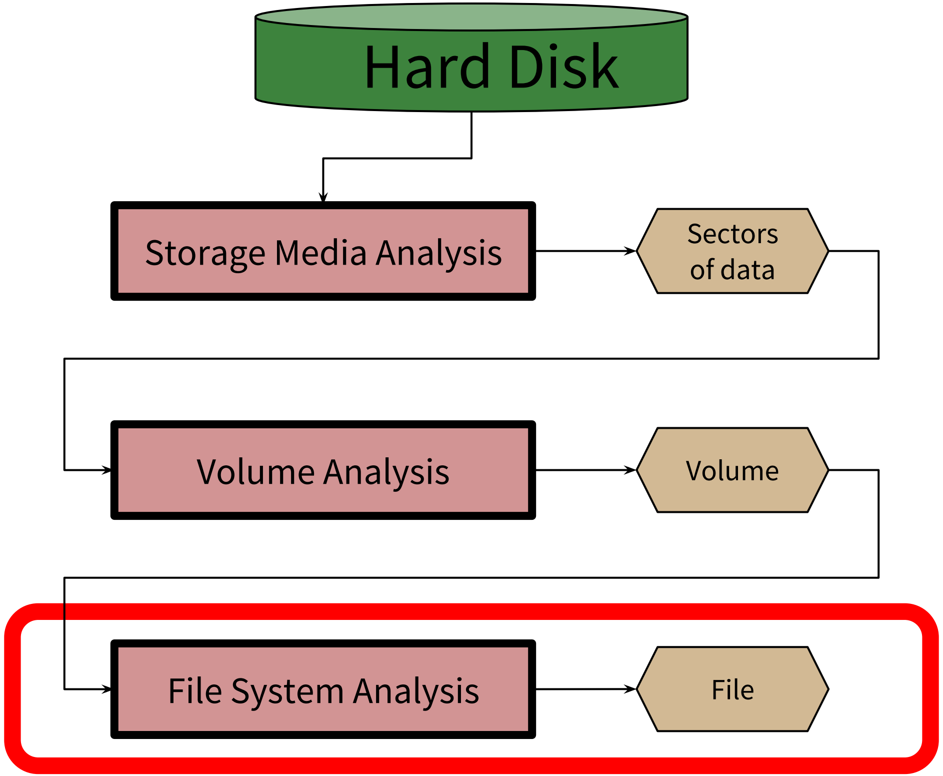
WIKIPEDIA
The Free Encyclopedia

https://smile.amazon.com/Understanding-Linux-Kernel-Third-Daniel/dp/0596005652/

https://smile.amazon.com/System-Forensic-Analysis-Brian-Carrier/dp/0321268172/

https://en.wikipedia.org/wiki/Ext4

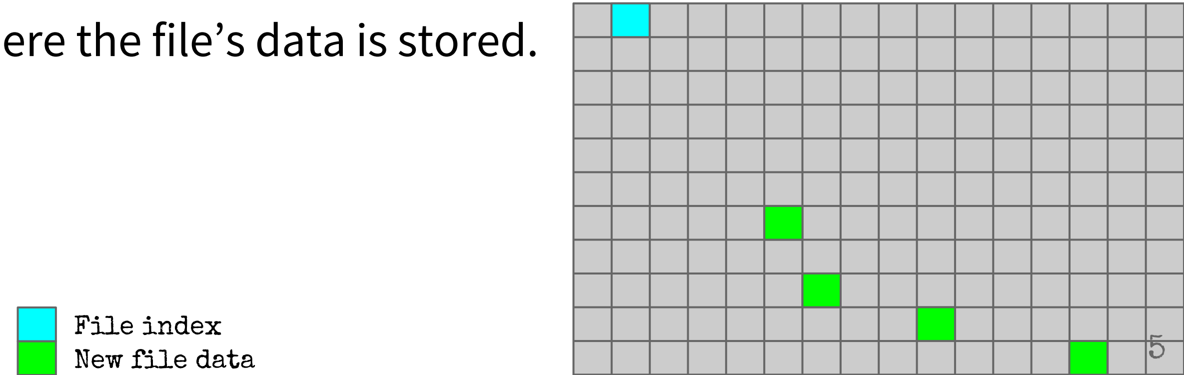
https://en.wikipedia.org/wiki/Btrfs


Hard Disk
Storage Media Analysis
Sectors of data
Volume Analysis
Volume
File System Analysis
File

Let’s Make a File System!

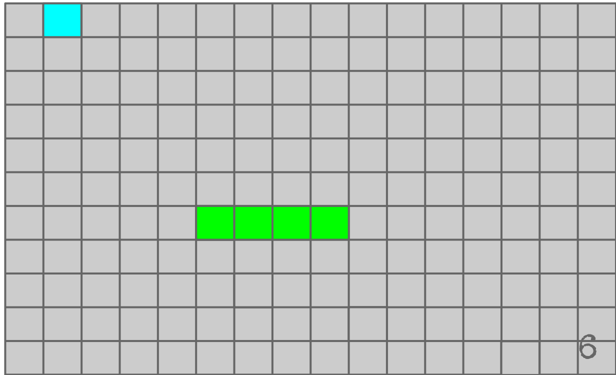
Storing a File (1)

- Scenario:
  - We want to store some data. The squares below represent discrete storage locations on the disk.
- Approach 1:
  - Just start writing data!
- Problem 1.1:
  - How do we find the information later?
- Solution 1.1:
  - Create an index of where the file's data is stored.



Storing a File (2)

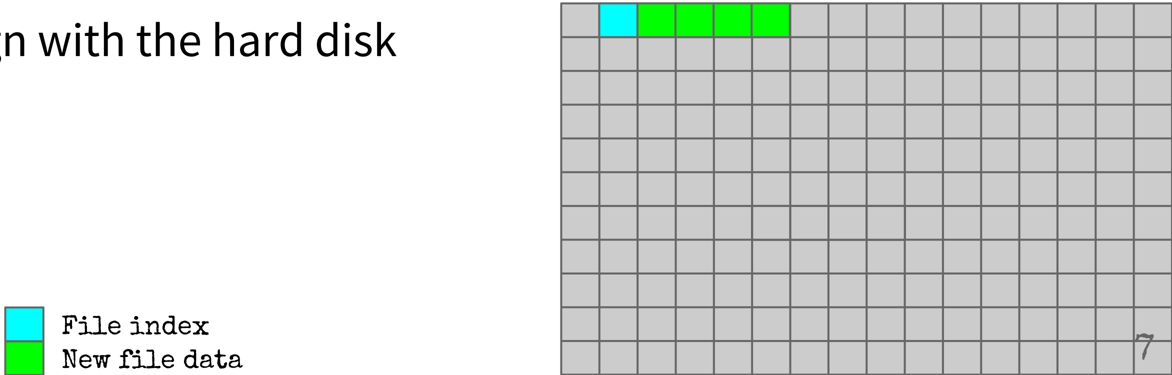
- Scenario:
  - We want to store some data. The squares below represent discrete storage locations on the disk.
- Approach 1:
  - Just start writing data!
- Problem 1.2:
  - Head seek time is unnecessarily high!
- Solution 1.2:
  - Don’t split up the file into multiple pieces, use contiguous storage space.



File index
New file data

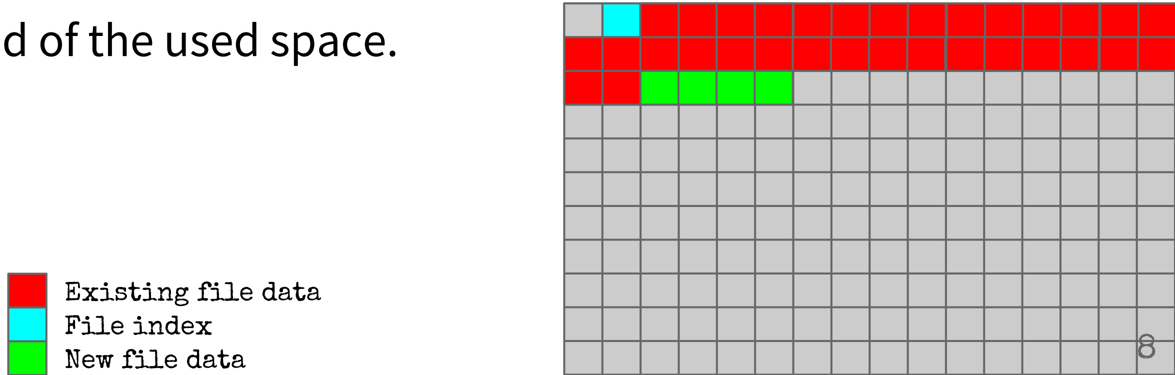
Storing a File (3)

- Scenario:
  - We want to store some data. The squares below represent discrete storage locations on the disk.
- Approach 2:
  - Write data in continuous storage locations.
- Problem 2.1:
  - Head seek time is still higher than it could be.
- Solution 2.1:
  - Use locations that align with the hard disk geometry.



Storing a File (4)

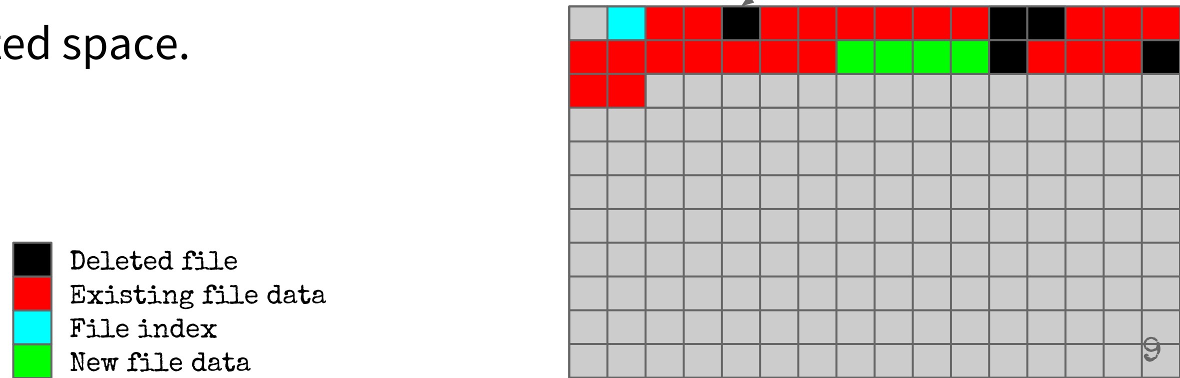
- Scenario:
  - We want to store some data. The squares below represent discrete storage locations on the disk.
- Approach 2:
  - Write data in continuous storage locations.
- Problem 2.2:
  - What if a file is already in that location?
- Solution 2.2:
  - Store the file at the end of the used space.



Storing a File (5)

- Scenario:
  - We want to store some data. The squares below represent discrete storage locations on the disk.
- Approach 2:
  - Write data in continuous storage locations.
- Problem 2.3:
  - What if some data has been deleted?
- Solution 2.3:
  - Try to reuse unallocated space.

Note: If we had started saving our file here, it would have become ***fragmented***.



Our File System

- Issues we covered while creating our file system:
  - Must keep track of where data is stored.
  - Storing data in contiguous locations improves performance when reading, writing, and updating.
  - Hard drive geometry affects read/write times.
  - Must account for existing data on the drive.
    - Fragmented files result when we don’t do a good job of predicting what space we need.
  - Must keep track of allocated/deleted areas.

Other File System Considerations

- Need a location to store metadata for each file:
  - Name
  - Times modified, accessed, created, etc.
  - Permissions
- Directory structure:
  - How to represent?
  - Where to store the information?
- Advanced features:
  - Self-healing files
  - Automatic defragmentation

For info on more advanced file system features, check out BTRFS: https://en.wikipedia.org/wiki/Btrfs

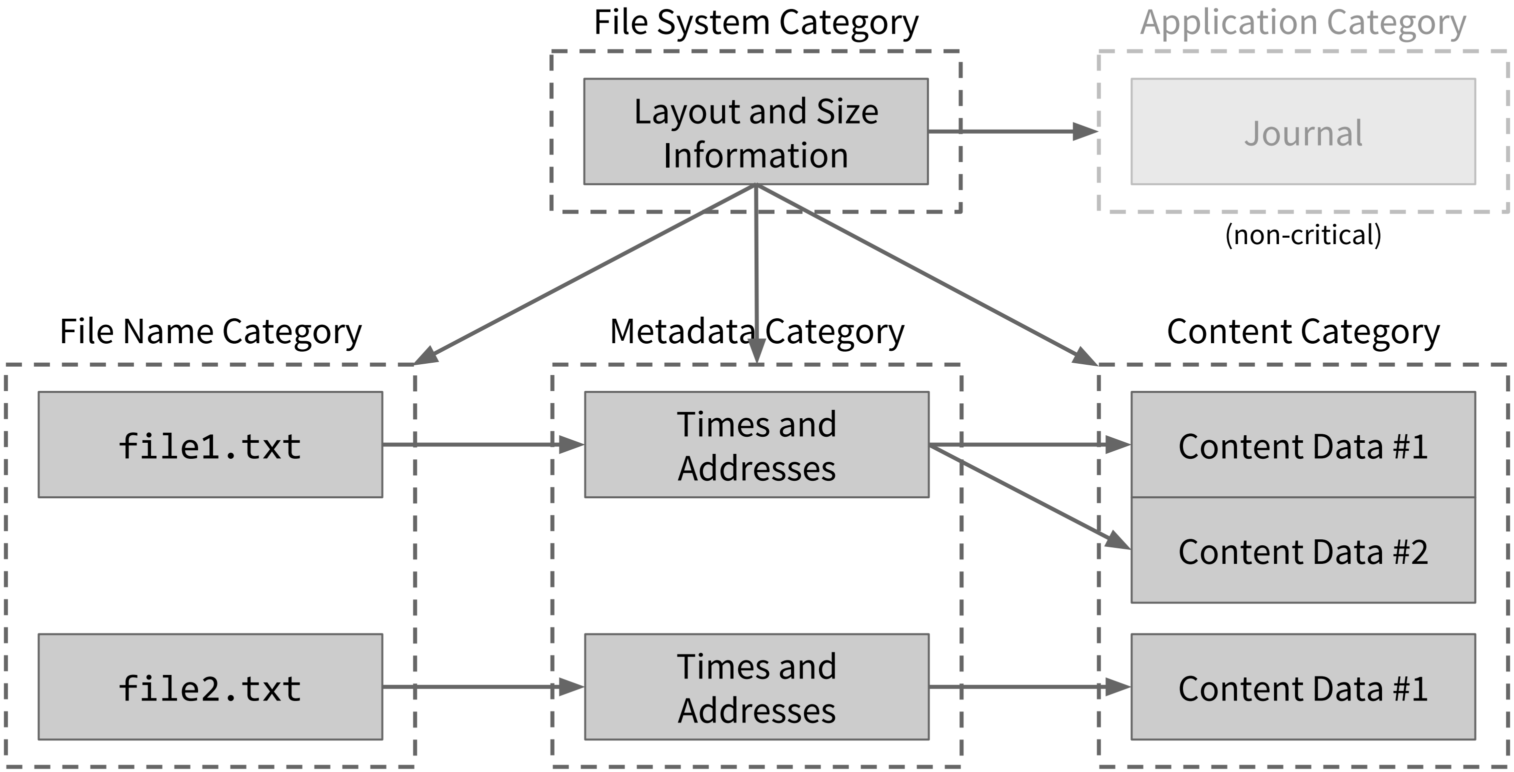
File System Reference Model

Arizona State University

Reference Model Categories

1. File system category:
   - General info about the file system.
   - Size and layout, location of data structures, size of data units.
2. Content category:
   - Data of the actual files - the reason file systems exist.
   - Organized into collections of standard-sized containers.
3. Metadata category:
   - Data that describes a file (except for the name of the file!).
   - Size, locations of content, times modified, access control info.
4. File name category:
   - a.k.a Human interface category.
   - Name of the file.
   - Normally stored in contents of a directory along with location of the file's metadata.
5. Application category:
   - Not essential to file system operations.
   - Journal.

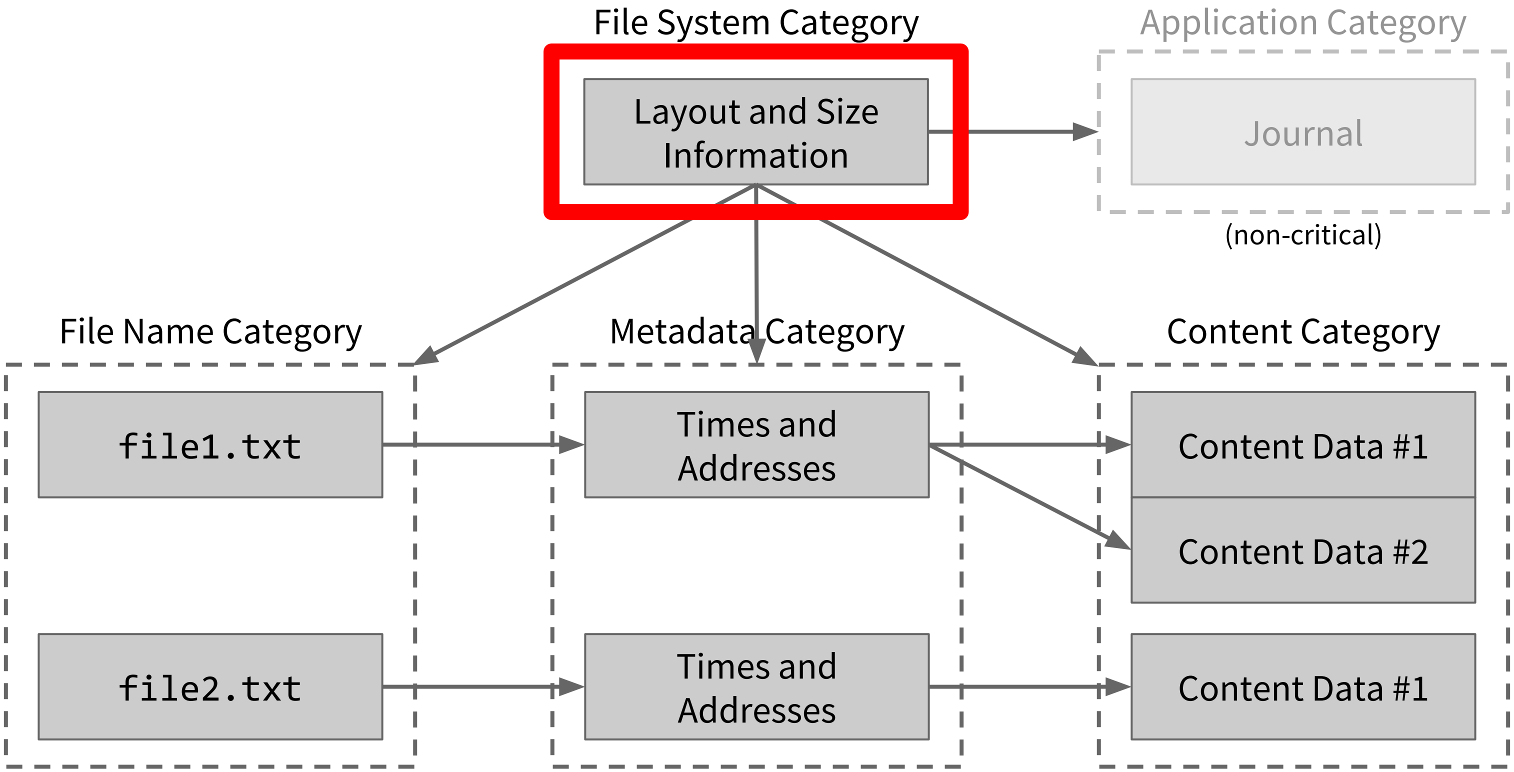
Reference Model Illustrated



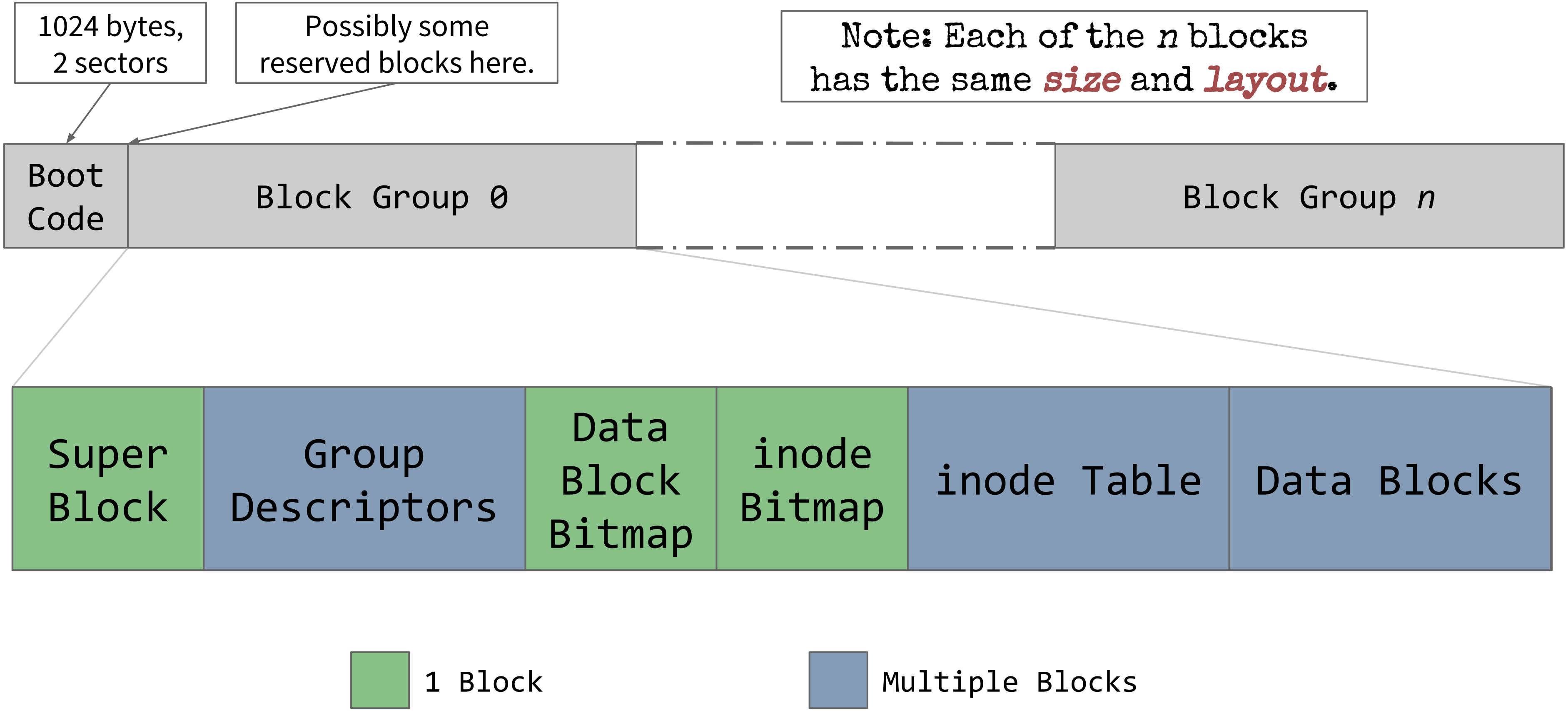
ext4

What is ext4?

- `ext` was the first file system designed for Linux.
- Organizes a disk into **blocks** and **block groups**.
  - Blocks: Groups of sectors. Called **clusters** in some *other* file systems. Blocks can be 1024, 2048, or 4096 bytes.
    - All blocks have an address, starting at 0.
    - The **smallest addressable space** in the file system.
  - Block Group: Set of blocks. Size is configurable, but always has the same structure. (More details in a couple slides!)
    - Groups are also numbered starting at 0.
    - There *may* be some reserved space before group 0.
- `ext4` was marked stable in October 2008.
- Google announced `ext4` would replace [YAFFS] as the default file system on Android devices in December 2010.


File System Category
Layout and Size Information
Application Category
Journal
(non-critical)
File Name Category
Metadata Category
Content Category
file1.txt
Times and Addresses
Content Data #1
Content Data #2
file2.txt
Times and Addresses
Content Data #1

ext4 Layout



1024 bytes,
2 sectors
Possibly some
reserved blocks here.
Note: Each of the n blocks
has the same size and layout.
Boot
Code
Block Group 0
Block Group n
Super
Block
Group
Descriptors
Data
Block
Bitmap
inode
Bitmap
inode Table
Data Blocks
1 Block
Multiple Blocks

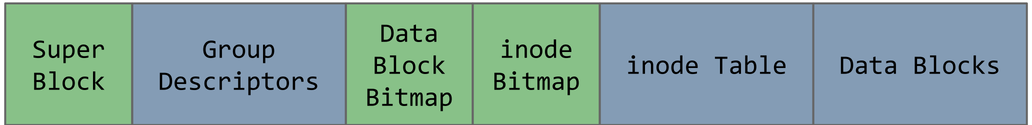
Boot Code

- If the file system has an OS kernel, first two sectors *may* have boot code.
  - Control is passed from the MBR boot code.

- More common scenario:
  - MBR code knows where the kernel is located and loads the kernel with no additional boot code stored by the file system.

Superblock

- Stores layout information for the file system.
- Duplicated in *every block group* in the file system.
  - Kernel only reads the superblock in group 0. The others are backup copies.
- Stores:
  - Block size
  - Total # of blocks
  - # blocks per group
  - # reserved blocks before group 0
  - # of inodes (total)
  - # of inodes per block group



Superblock Contents: Example

```
# dumpe2fs /dev/sda1
Filesystem volume name:   boot
Last mounted on:          /boot
Filesystem UUID:          79fc5ed8-5bbc-4dfe-8359-b7b36be6eed3
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal ext_attr resize_inode dir_index
Filesystem flags:         signed_directory_hash
Default mount options:    user_xattr acl
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              122160
Block count:              488192
Reserved block count:     24409
Free blocks:              376512
Free inodes:              121690
First block:              0
Block size:               4096
Fragment size:            4096
Group descriptor size:    64
Reserved GDT blocks:      238
Blocks per group:         32768
Fragments per group:      32768
Inodes per group:         8144
```

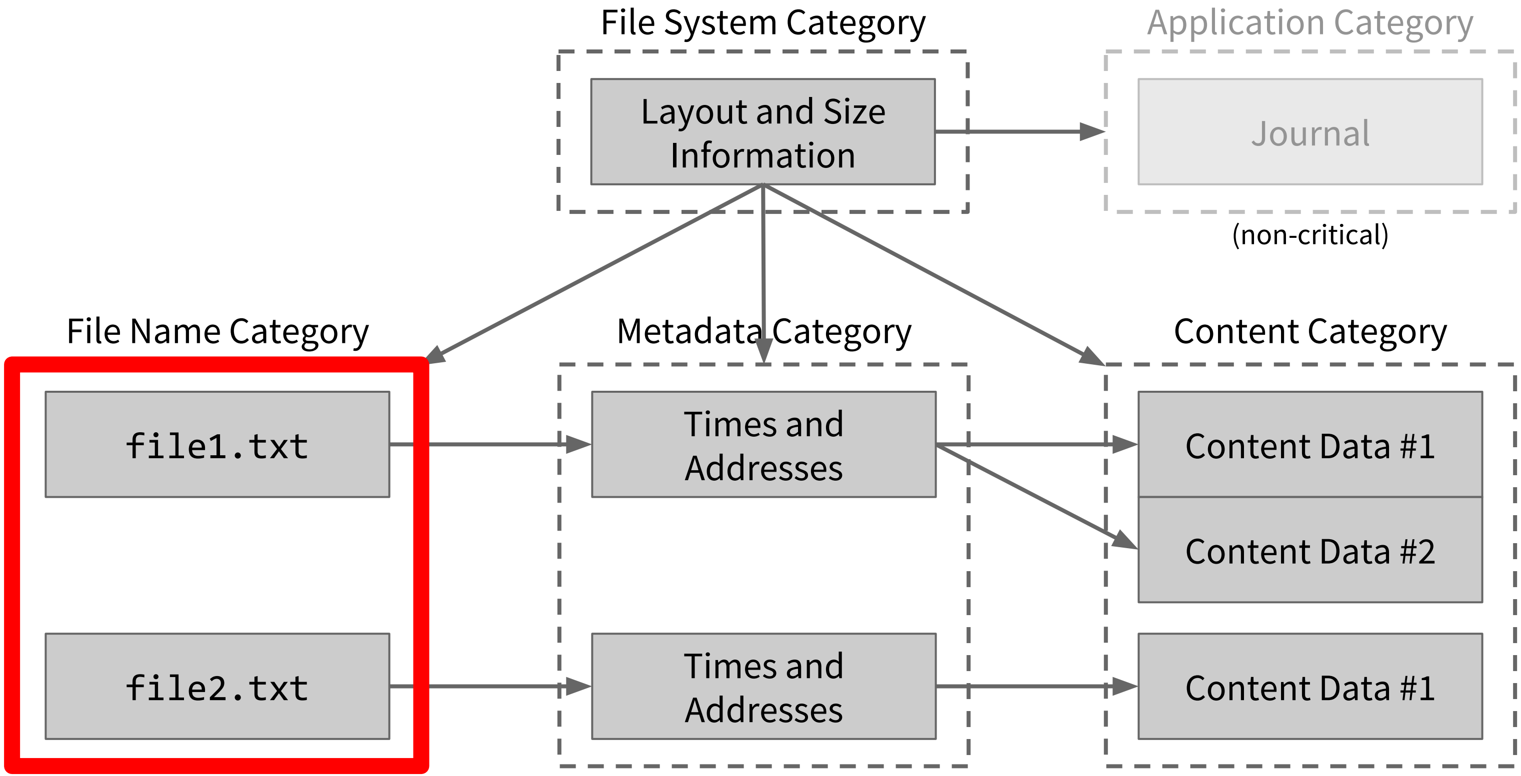
```
Flex block group size:    16
Filesystem created:       Tue Feb  7 09:33:34 2017
Last mount time:          Sat Apr 29 21:42:01 2017
Last write time:          Sat Apr 29 21:42:01 2017
Mount count:              25
Maximum mount count:      -1
Last checked:             Tue Feb  7 09:33:34 2017
Check interval:           0 (<none>)
Lifetime writes:          594 MB
Reserved blocks uid:      0 (user root)
Reserved blocks gid:      0 (group root)
First inode:              11
Inode size:               256
Required extra isize:     32
Desired extra isize:      32
Journal inode:            8
Default directory hash:   half_md4
Directory Hash Seed:      c780bac9-d4bf-4f35-b695-0fe35e8d2d60
Journal backup:           inode blocks
Journal features:         journal_64bit
Journal size:             32M
Journal length:           8192
Journal sequence:         0x00000213
Journal start:            0
```


Source: https://opensource.com/article/17/5/introduction-ext4-filesystem

Group Descriptor

- Has the following fields:
  - Block numbers of the block bitmap and inode bitmap.
  - Block number of the first inode table block.
  - Number of free blocks, free inodes, and directories in the group.
- The descriptor table contains **all** the descriptors for the whole file system.
- Duplicated in ***every block group***, just like the superblock.

Super Block	Group Descriptors	Data Block Bitmap	inode Bitmap	inode Table	Data Blocks


File System Category
Layout and Size Information
Application Category
Journal
(non-critical)
File Name Category
file1.txt
file2.txt
Metadata Category
Times and Addresses
Times and Addresses
Content Category
Content Data #1
Content Data #2
Content Data #1

ARIZONA STATE UNIVERSITY

Directory

- Just another file, but with a simple structure that identifies the files it contains.
- Always includes '.' (self) and '..' (parent) entries (even for the root directory!).
- Directory entry fields:
  - inode number
  - File name
  - File type number →

	File Type
0	Unknown
1	Regular file
2	Directory
3	Character device
4	Block device
5	Named pipe
6	Socket
7	Symbolic link

Directory Entry Example

offset	inode	rec_len	name_len	file_type	name							
0	21	12	1	2	.	\0	\0	\0				
12	22	12	2	2	.	.	\0	\0				
24	53	16	5	2	h	o	m	e	1	\0	\0	\0
40	67	28	3	2	u	s	r	\0				
52	0	16	7	1	o	l	d	f	i	l	e	\0
68	34	4028	4	2	s	b	i	n				

Deleted: → 52
There is no inode 0.

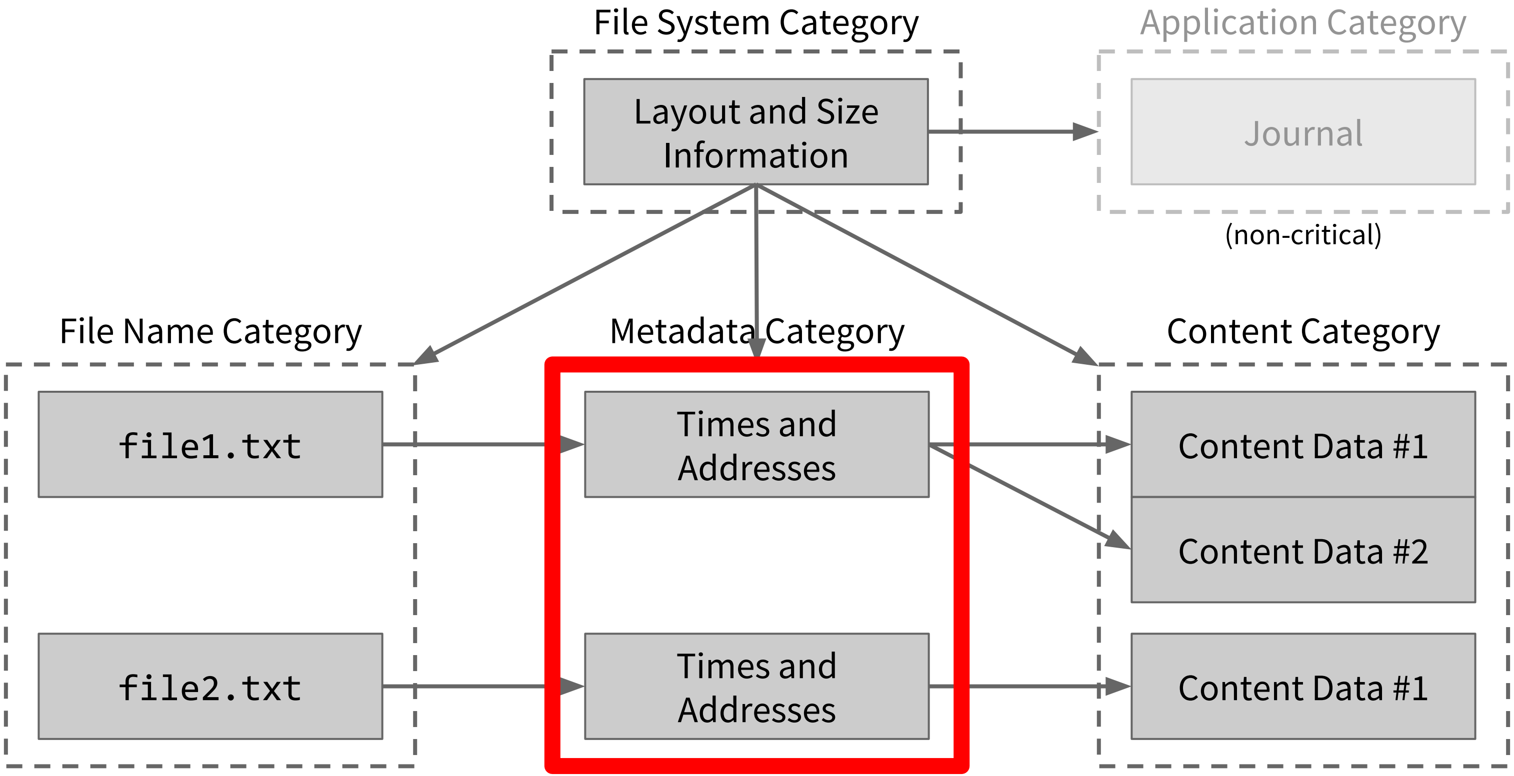
The last record needs to point to the end of the block, so it will have a length much larger than normal.

Always 8 bytes (inode, rec_len, name_len, file_type)

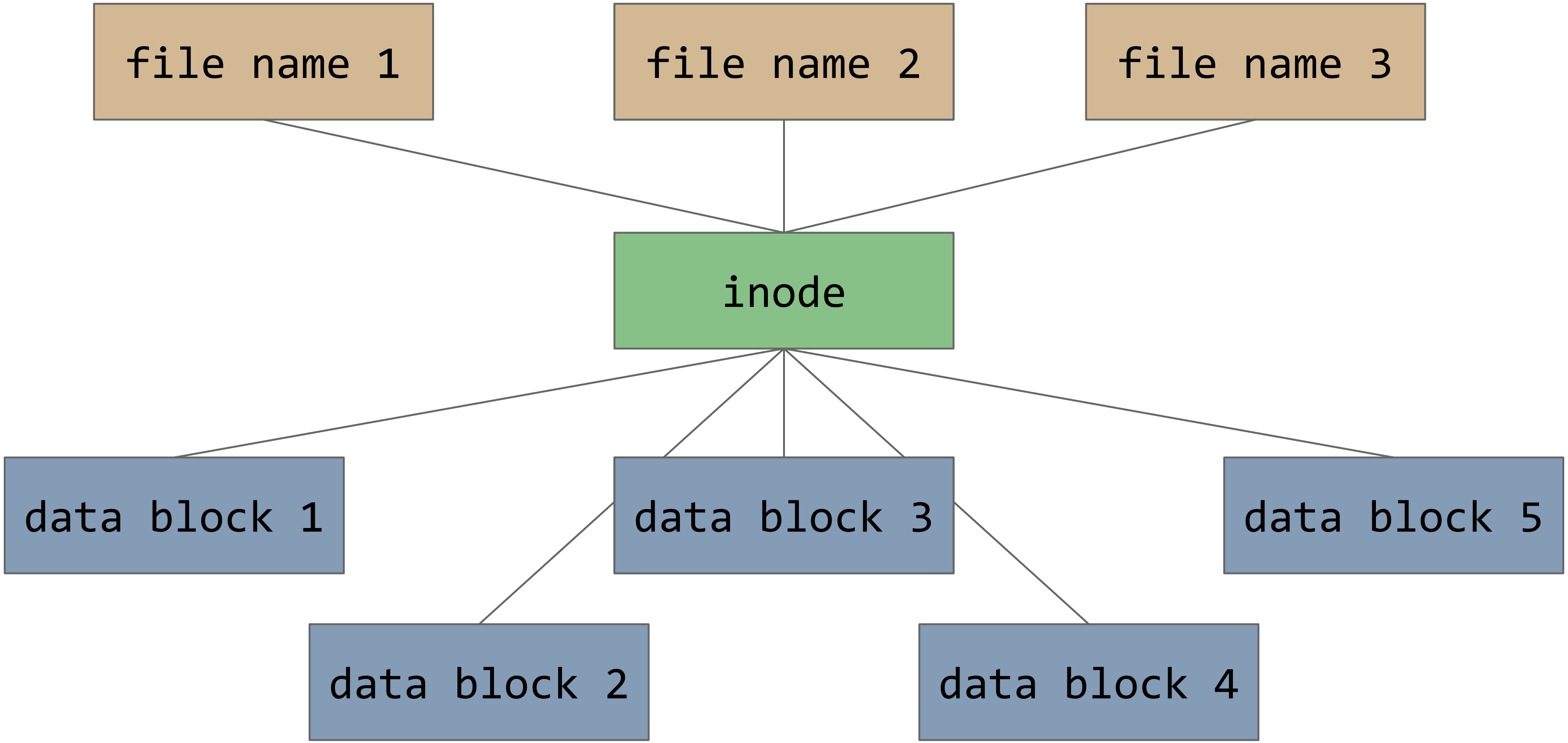
Always a multiple of 4 bytes (name)

Newer Directory Entries

- A linear array of entries isn't very efficient.
- `ext3` and `ext4` can use a balanced tree (hashed btree) keyed off a hash of the directory entry name.
- Details are beyond the scope of this class.


File System Category
Layout and Size Information
Application Category
Journal
(non-critical)
File Name Category
file1.txt
file2.txt
Metadata Category
Times and Addresses
Times and Addresses
Content Category
Content Data #1
Content Data #2
Content Data #1

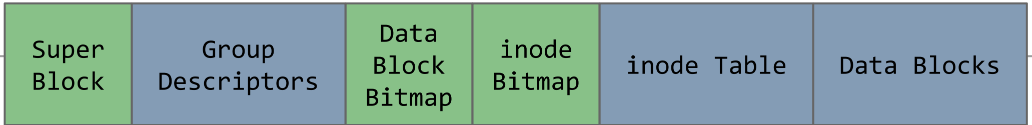
inodes


file name 1
file name 2
file name 3
inode
data block 1
data block 3
data block 5
data block 2
data block 4

ASU Arizona State University

inode Fields (Selected) (1)

Offset	Bits	Name	Description
0x0	16	i_mode	Mode (9 bits). Sticky bit, setgid, setuid (3 bits). File type (4 bits).
0x2	16	i_uid	Owner's user identifier (UID).
0x18	16	i_gid	Group identifier (GID).
0x8	32	i_atime	Last access time, in seconds since the epoch.
0xC	32	i_ctime	Last inode change time, in seconds since the epoch.
0x10	32	i_mtime	Last data modification time, in seconds since the epoch.
0x14	32	i_dtime	Deletion Time, in seconds since the epoch.
0x1A	16	i_links_count	Hard link count. With the DIR_NLINK feature enabled, ext4 supports more than 64,998 subdirectories by setting this field to 1 to indicate that the number of hard links is not known.
0x28	60	i_block	Extent tree.



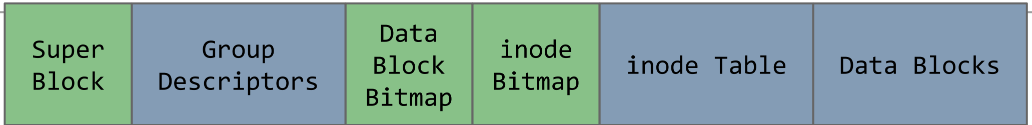
See also https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#Inode_Table

inode Fields (Selected) (2)

Offset	Bits	Name	Description
0x4	32	i_size_lo	Lower 32-bits of size in bytes.
0x6C	32	i_size_high	Upper 32-bits of file/directory size.
0x1C	32	i_blocks_lo	Lower 32-bits of "block" count.
0x74	16	i_blocks_hi	Upper 16-bits of the block count.
0x84	32	i_ctime_extra	Extra change time bits. This provides sub-second precision.
0x88	32	i_mtime_extra	Extra modification time bits. This provides sub-second precision.
0x8C	32	i_atime_extra	Extra access time bits. This provides sub-second precision.
0x90	32	i_crtime	File creation time, in seconds since the epoch. (Creation time of inode.)
0x94	32	i_crtime_extra	Extra file creation time bits. This provides sub-second precision.

Note: Every field with an offset >=0x80 is an ***extended field***, meaning it was introduced in ext4 and is not backwards compatible with ext2/3.

See also https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#Inode_Table



Mode

- ext4 stores file permissions for the **user** (the owner of the file), the **group** the file is a part of, and all **others** (world).
- 3 bits for each ↑ represent the ***read***, ***write***, and ***execute*** permissions: 1 means they can, 0 means they can't.

Example Mode: `0754`

`0`: Means number is displayed in octal

`111`
`1`: Owner can read
`1`: Owner can write
`1`: Owner can execute

`101`
`1`: Group can read
`0`: Group cannot write
`1`: Group can execute

`100`
`1`: World can read
`0`: World cannot write
`0`: World cannot execute

Arizona State University

File Types

0. Unknown
1. Regular file
2. Directory
3. Character device
4. Block device
5. Named pipe
6. Socket
7. Symbolic link

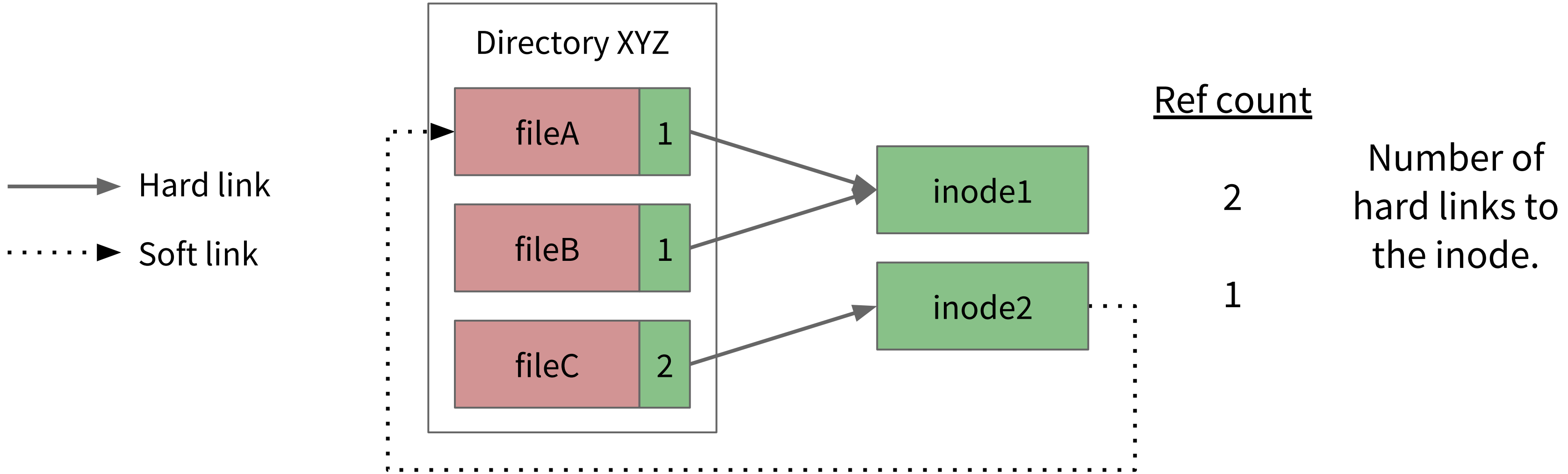
Regular file, Directory: The only 2 types that allocate data blocks in the file system (except symbolic links, sometimes).

Block device: Require all read/write operations to work on an entire block at a time.

Symbolic link: Contents of the file are the path to the file pointed to. Path is stored in inode if <60 characters, uses a data block otherwise.

Hard and Soft Links

- Hard link: A **filename** that points to an **inode**.
  - *Everything* has a hard link to it.
- Soft link: An **inode** that points to a **filename**.
  - Optional.



Time Attributes

- Allow an investigator to develop a timeline of the incident
- M-A-C
  - **m**time: Modified time
    - Changed by modifying a file's content.
  - **a**time: Accessed time
    - Changed by reading a file or running a program.
  - **c**time : changed time
    - Keeps track of when the meta-information about the file was changed (e.g., owner, group, file permission, or access privilege settings).
    - Can be used as approximate ***dtime*** (deleted time).

This slide is from
Topic 1: Forensics Intro

ext4: Extra Time Attributes

- ext4 introduces two additional time attributes:
  - **d**time: deletion time
  - **cr**time: creation time
- ext4 extends the time values from 32 bits to 64.
  - Overcomes the 2038 problem (puts it off until 2446).
    - 32 bits is a signed int to allow referencing dates *before* January 1, 1970 by using negative numbers.
  - Does not apply to dtime (remains 32 bits).

64-bit Time Values in ext4

Extra time field: 32 bits

Original time field: 32 bits

0001010010100101001010010100100 1   1001010010100100110010100101001 0

Number of seconds since the epoch (Jan 1, 1970 UTC)

New whole-second value:

February 16, 2185 00:22:42 ← 6788794962 == 0110010100101001001100101001010010

Nanoseconds means 9 decimal places

Nanosecond value:

000101001010010100101001010010 == 86592082   0.086592082

Don't forget you have to convert the bytes from Little Endian first!

Final date value:

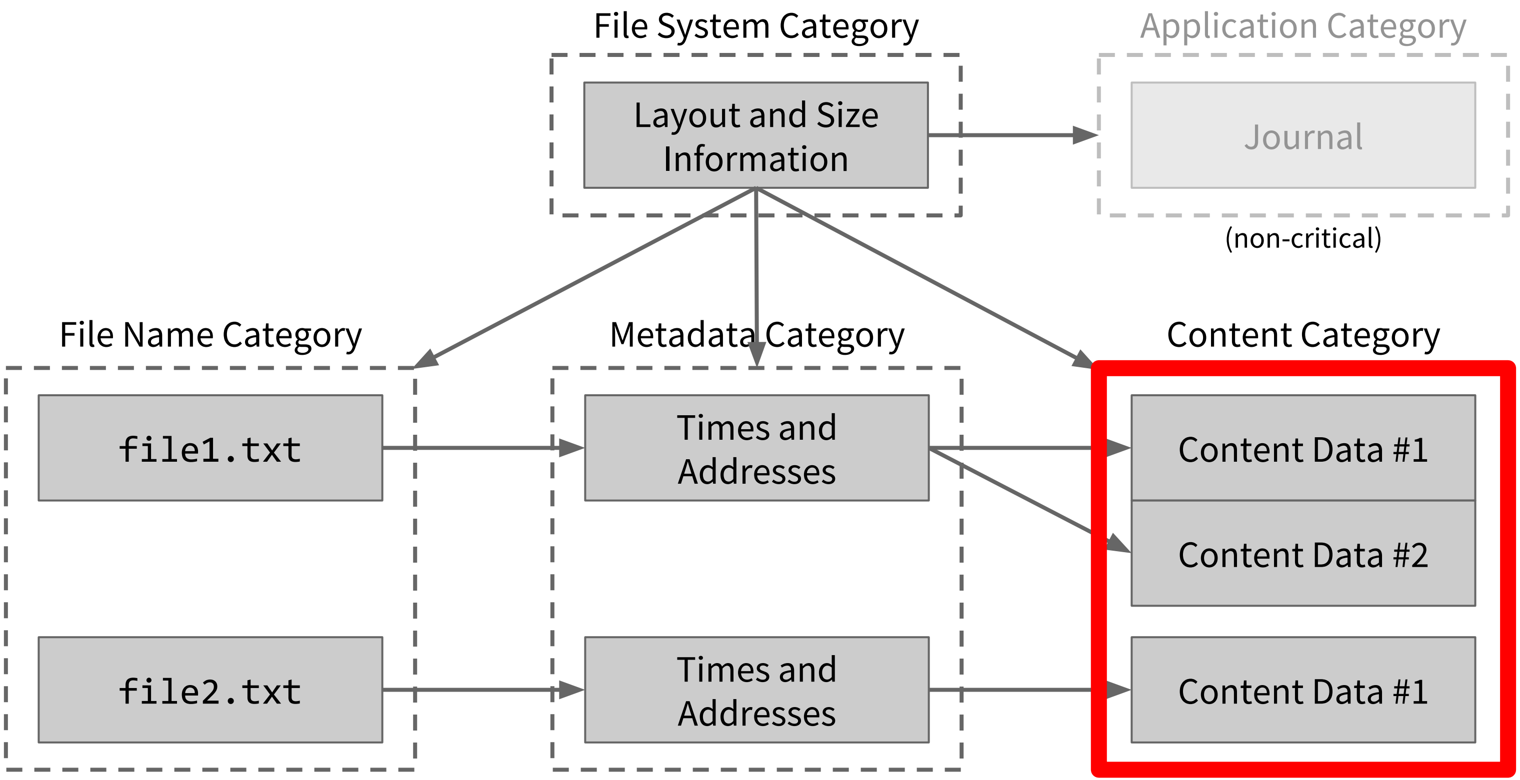
February 16, 2185 00:22:42.086592082

Effects of Deleting a File in ext4

- Changes to the inode: (assuming ref_count is 1)
  - The file size value is set to zero.
  - The number of extents value in the extent header is likewise zeroed.
  - The extent itself is also cleared.
- Changes to the directory:
  - inode number is set to 0.
  - Previous entry lengthened to cover the deleted file's entry in the directory.
    - Linear directory entries only!
- Changes to the block group(s):
  - inode bitmap set to 0 for freed inode(s).
  - Data block bitmap set to 0 for freed data block(s).

Super Block	Group Descriptors	Data Block Bitmap	inode Bitmap	inode Table	Data Blocks

See also: https://digital-forensics.sans.org/blog/2010/12/20/digital-forensics-understanding-ext4-part-1-extents


File System Category
Layout and Size Information
Application Category
Journal
(non-critical)
File Name Category
Metadata Category
Content Category
file1.txt
file2.txt
Times and Addresses
Times and Addresses
Content Data #1
Content Data #2
Content Data #1

Block Bitmap / inode Bitmap

- 0 == available.
- 1 == in use.
- One bit per block/inode.
  - Denotes ***allocation status***.
- Number of **data blocks in a group** is always equal to the number of **bits in a block**.
- Far fewer inodes than blocks per group.
  - User-configurable.
  - Makes sense since most files will occupy more than one block, only need one (initial) inode per file.

Super Block	Group Descriptors	Data Block Bitmap	inode Bitmap	inode Table	Data Blocks

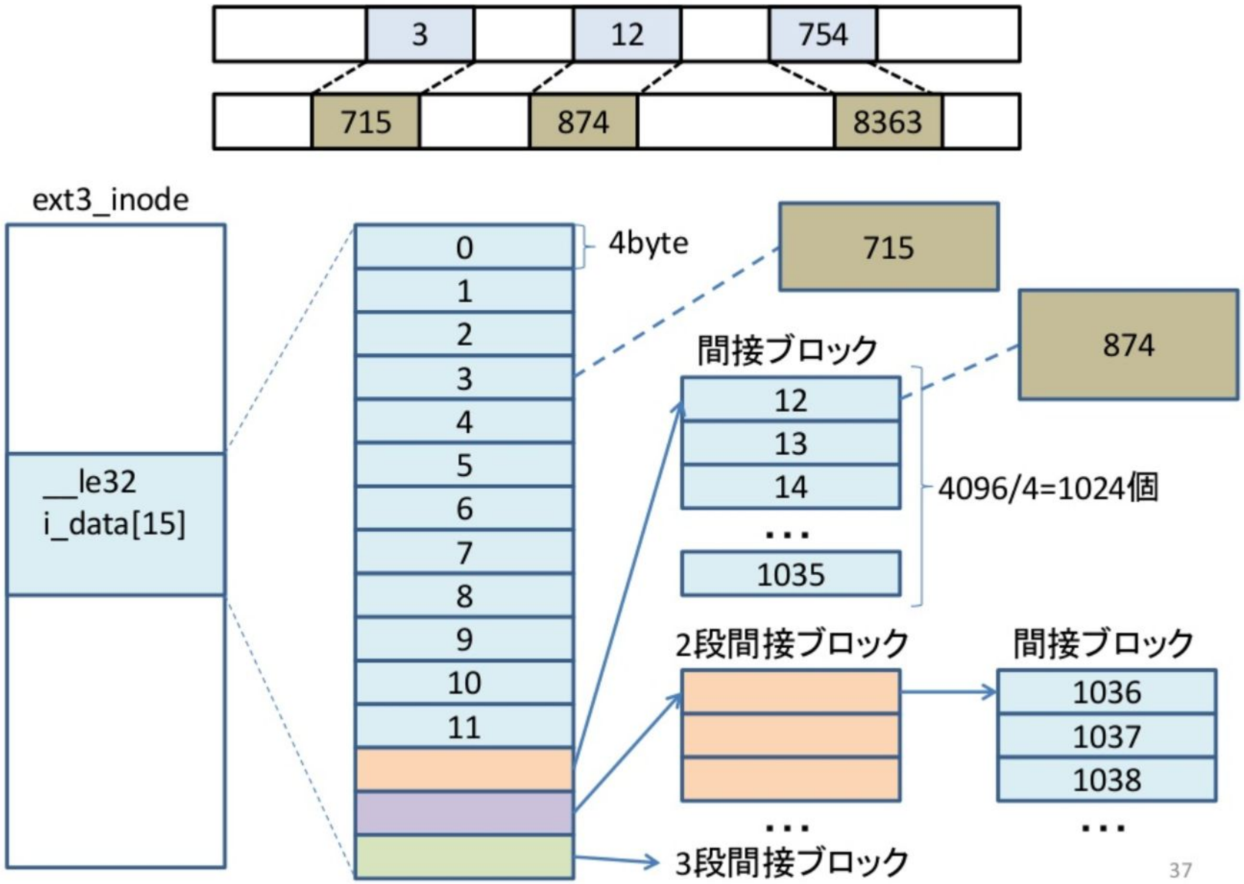
Extents

- The unit of allocation in `ext4`.
  - Described by its **starting** and **length** in blocks.
  - One file fragment only uses one extent.

- Previous "block mapping" scheme (<=`ext3`) stored each block address used by the file.

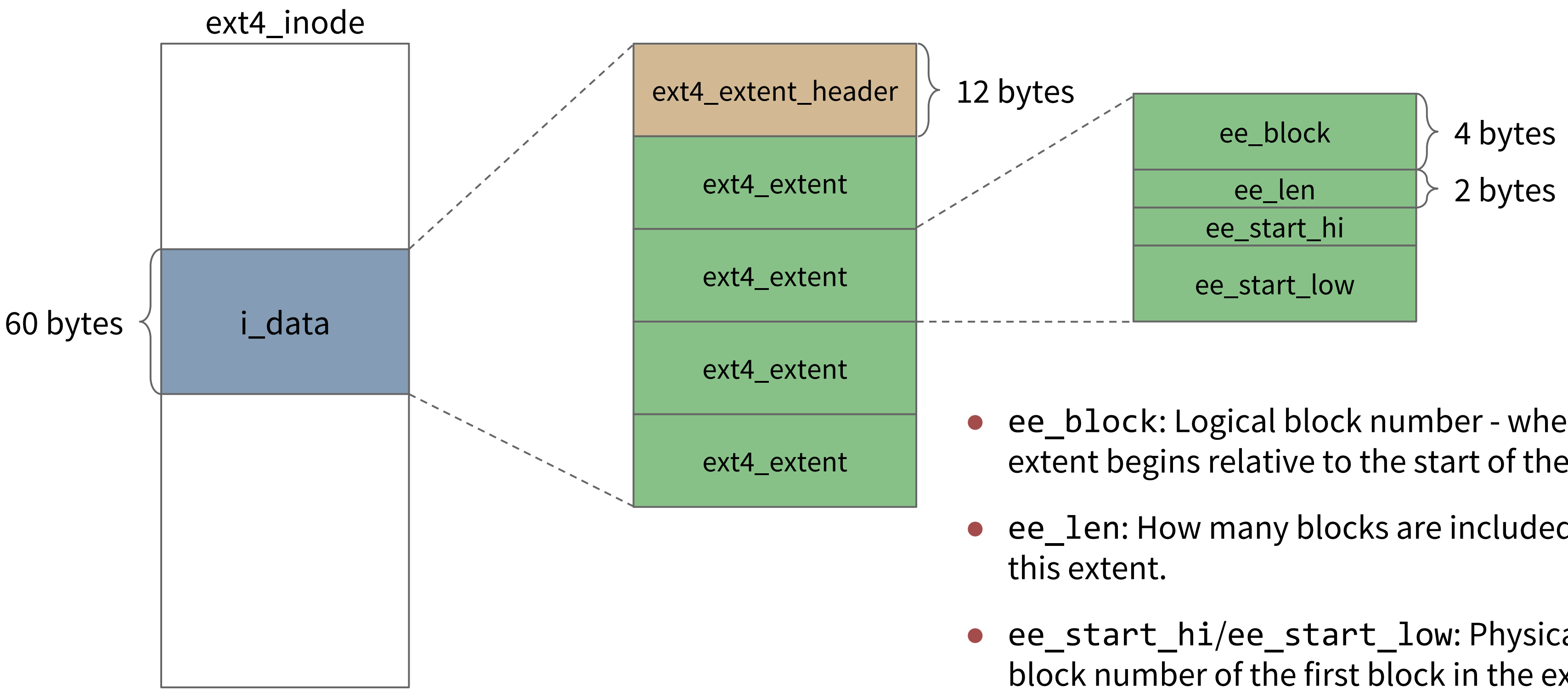
Super Block	Group Descriptors	Data Block Bitmap	inode Bitmap	inode Table	Data Blocks

Block Mapping



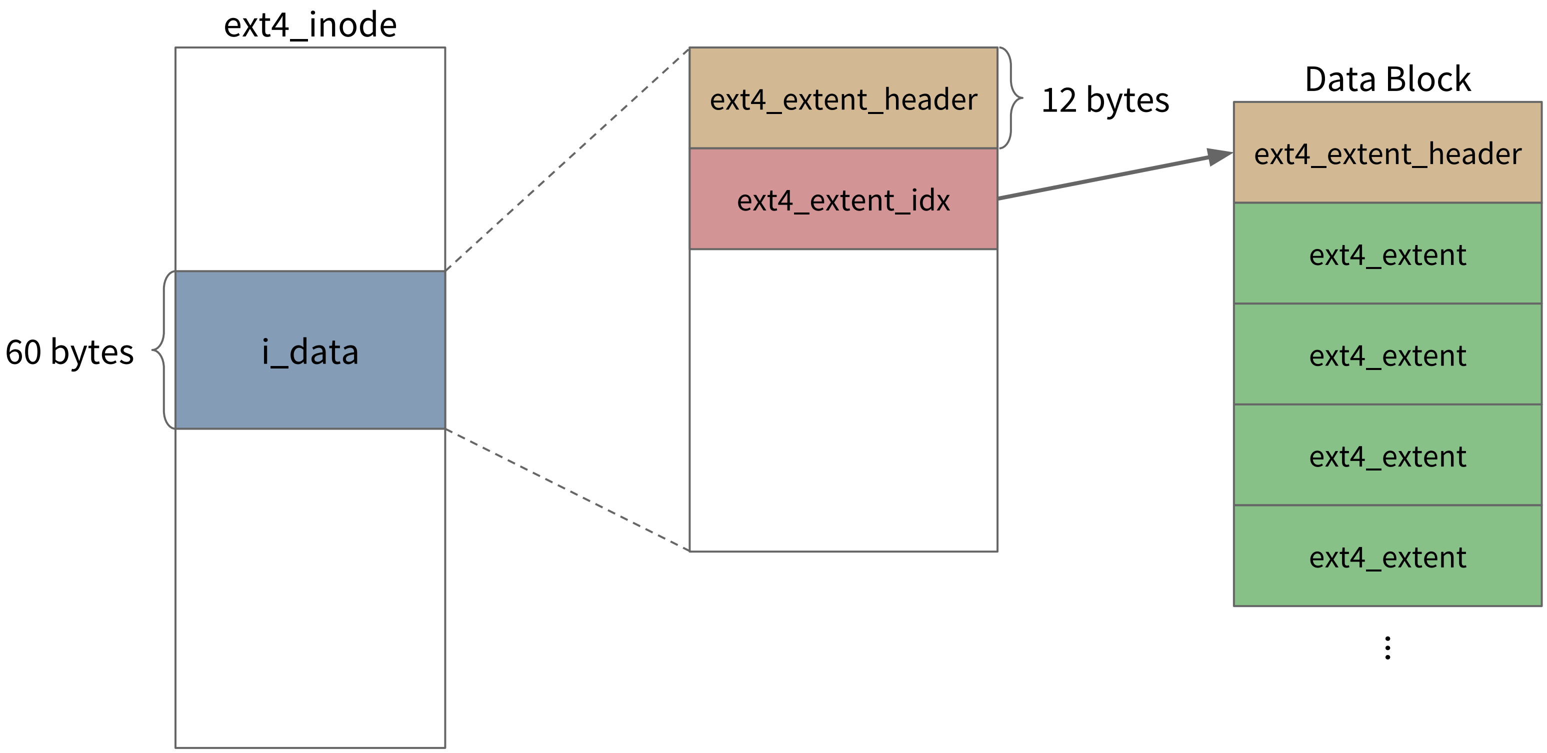
Source: https://www.slideshare.net/YoshihiroYunomae/f-36905134

Extent Structure



- `ee_block`: Logical block number - where this extent begins relative to the start of the file.
- `ee_len`: How many blocks are included in this extent.
- `ee_start_hi/ee_start_low`: Physical block number of the first block in the extent- i.e., where the extent actually begins on disk.

Extent Tree



Has the same structure as ext4_extent on the previous slide.

If a file needs more than 4 extents, `ext4` makes what is called an "extent tree".

See also: https://digital-forensics.sans.org/blog/2011/03/28/digital-forensics-understanding-ext4-part-3-extent-trees

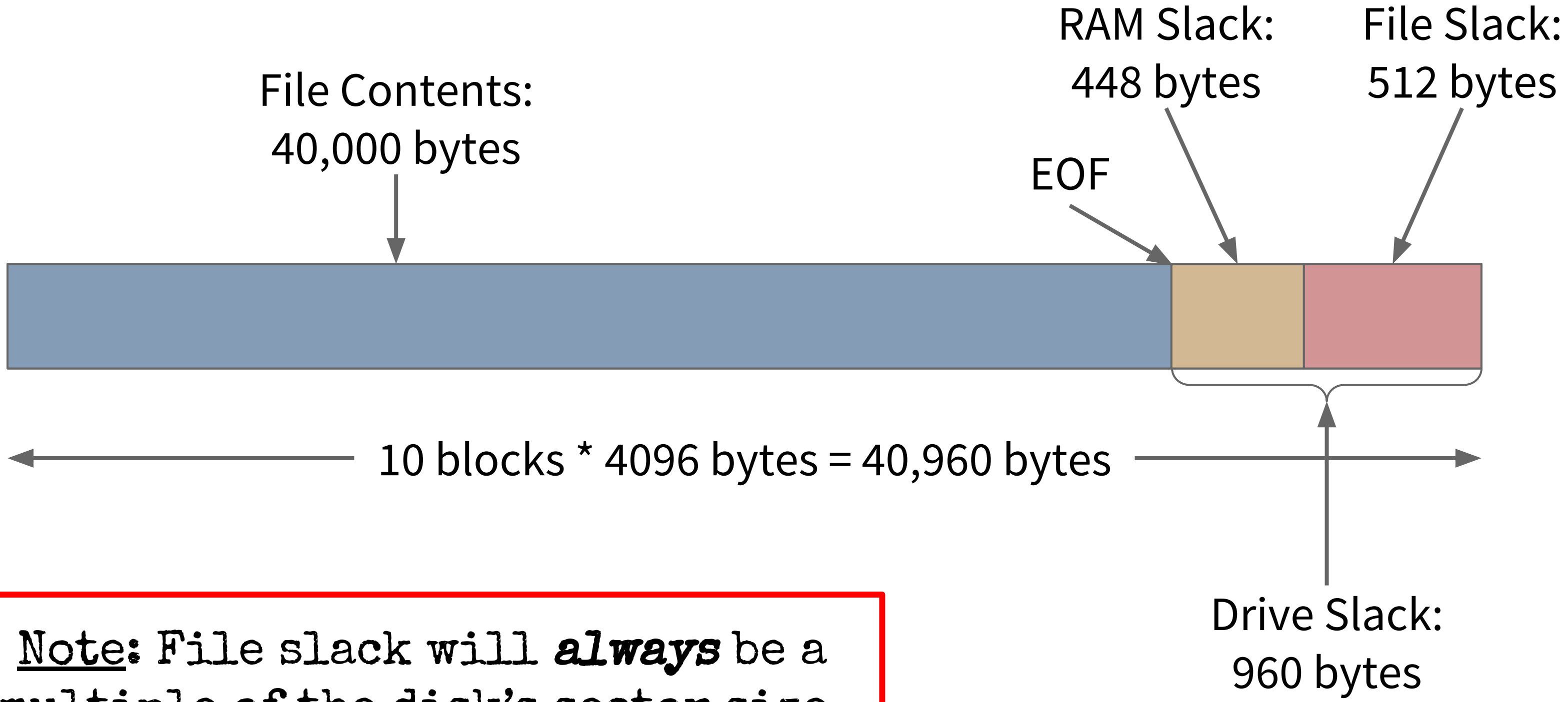
Drive Slack

- Drive Slack: The area on a disk that is ***allocated*** to a file, but doesn't store any of the file's data.
- Example:
  - File system with 4K blocks on a disk with 512 byte sectors.
  - File that is 40,000 bytes long occupies 10 blocks.
  - 10 blocks * 4096 bytes = 40,960 bytes allocated for the file.
  - The excess space of 960 bytes is called **drive slack**.
- Drive slack is divided into two parts: File slack and RAM slack.

File and RAM Slack

- Block devices: Require all read/write operations to work on an **entire block** at a time.
  - Cannot read/write a character at a time the way ***character devices*** do.
- Legacy operating systems used to read an entire block of data from RAM when writing to disk, *whether or not the entire block was part of the file being written!*
  - This is **RAM slack**. The size of the RAM slack is determined by how much of the disk's sector is leftover after writing the file.
  - The part of drive slack that isn't RAM slack is **file slack**.
- RAM slack Could be anything stored in memory: logon IDs, passwords, file fragments, … anything!

Slack: Illustrated



<u>Note</u>: File slack will ***always*** be a multiple of the disk's sector size.