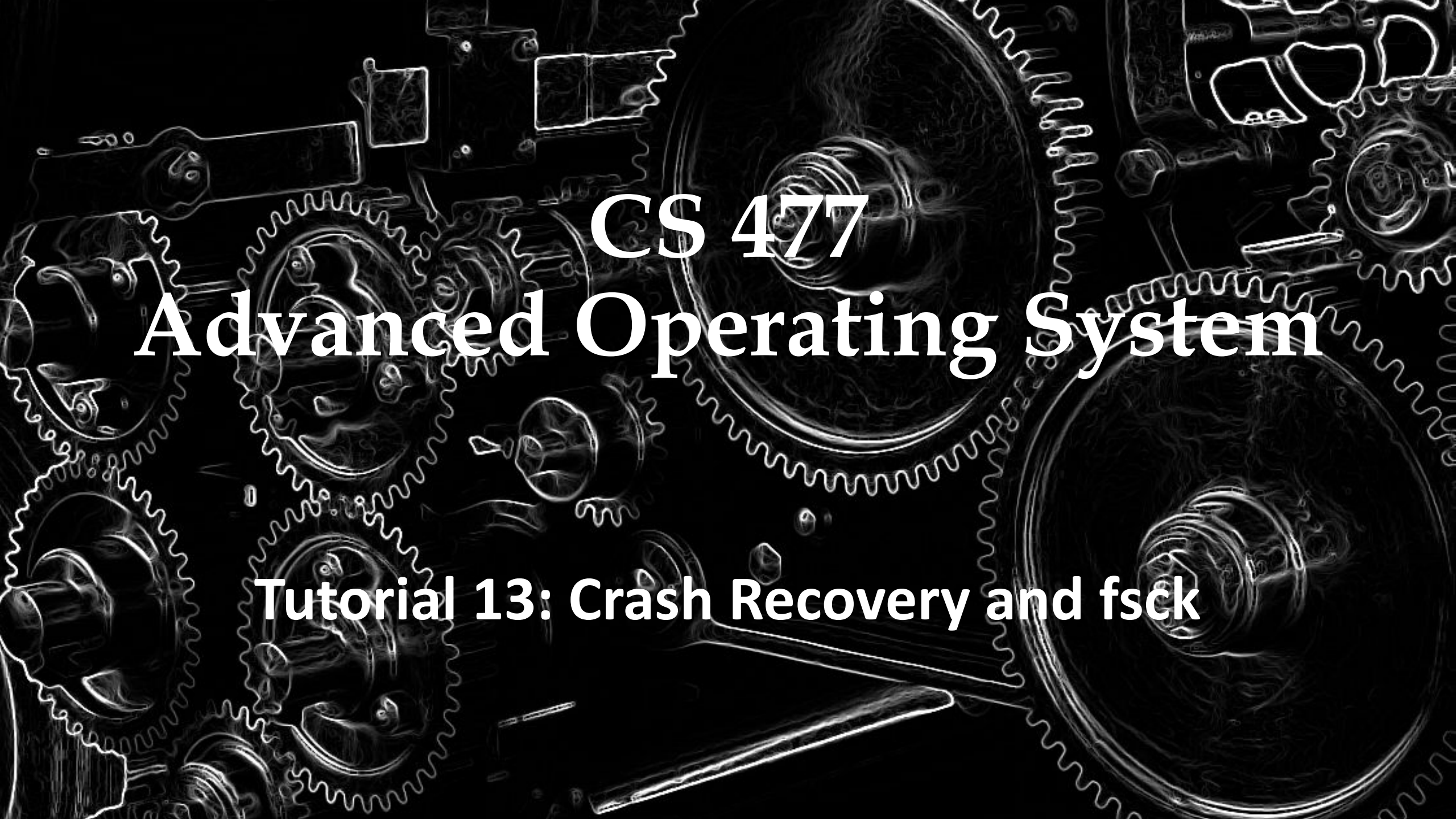


CS 477
Advanced Operating System

Tutorial 13: Crash Recovery and fsck

Today's Tutorial Outline

- Illustration on how FS metadata should agree
- What happens when metadata doesn't agree
- Using debugfs to inspect + selectively corrupt file systems
- Recovery using fsck

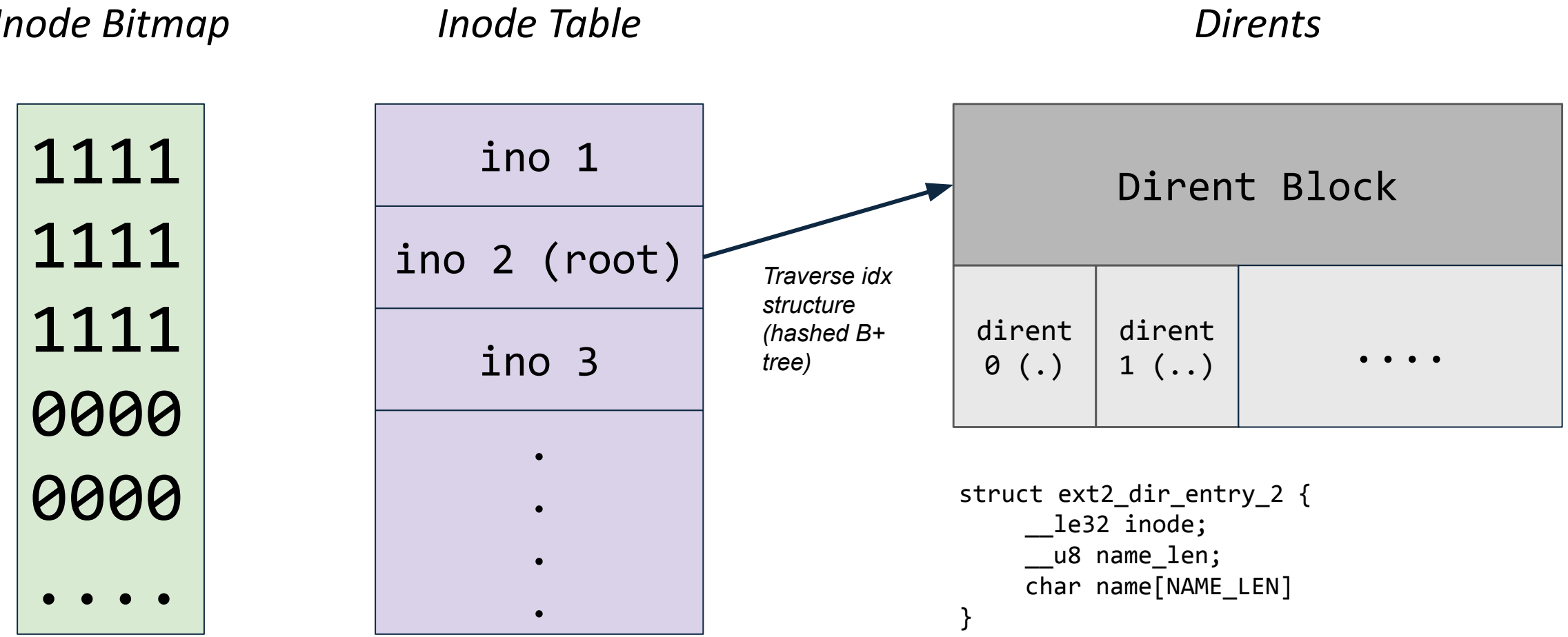
Follow along on your machine

- https://github.com/sidharth-sundar/advos-25-demo/tree/main/Week-13-Demo

- We'll trace through the scripts in the second half of the demo
- Link also posted on Moodle
  - Week 13 > Tutorial Code

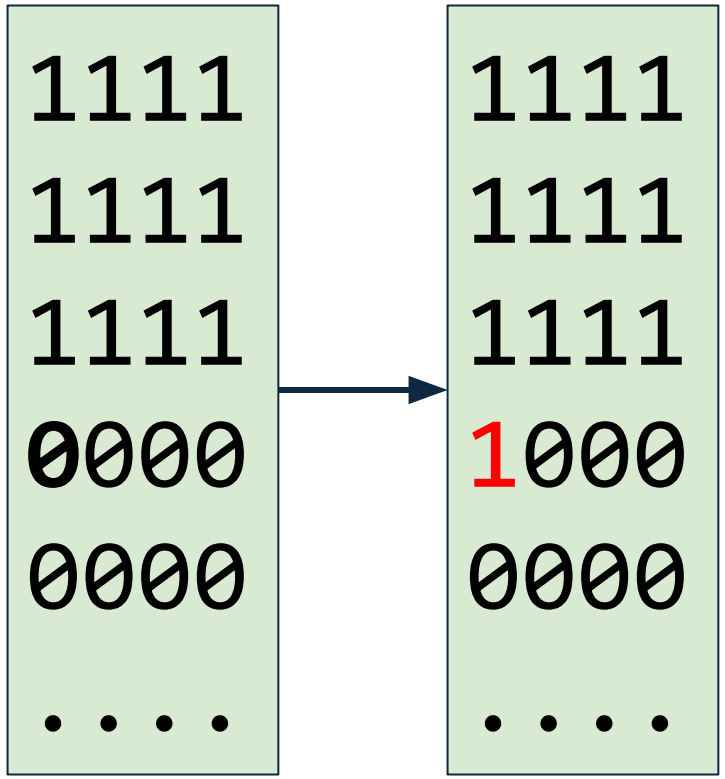
- Need `e2fsprogs` for tools shown later
  - Library of utilities for ext2/3/4
  - `$ sudo apt install e2fsprogs`

Three metadata structures in ext4

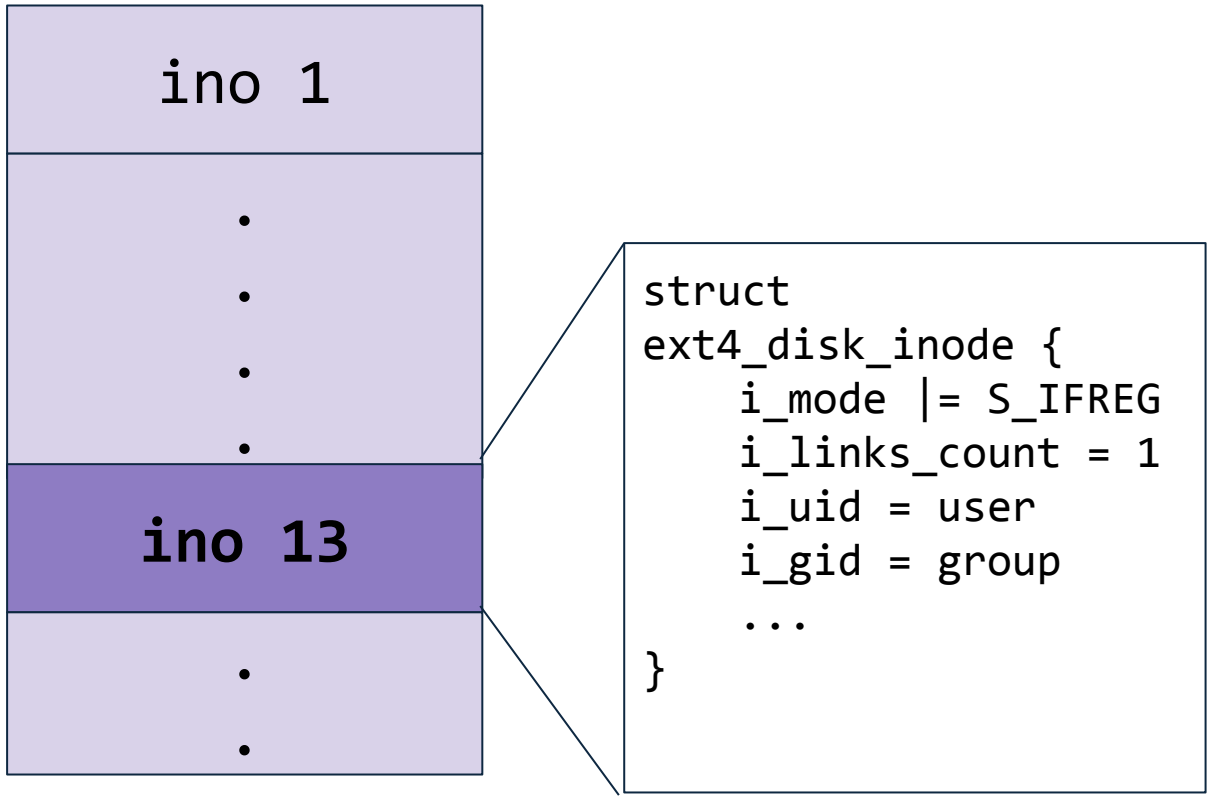


creat("/file1.txt", 0644)

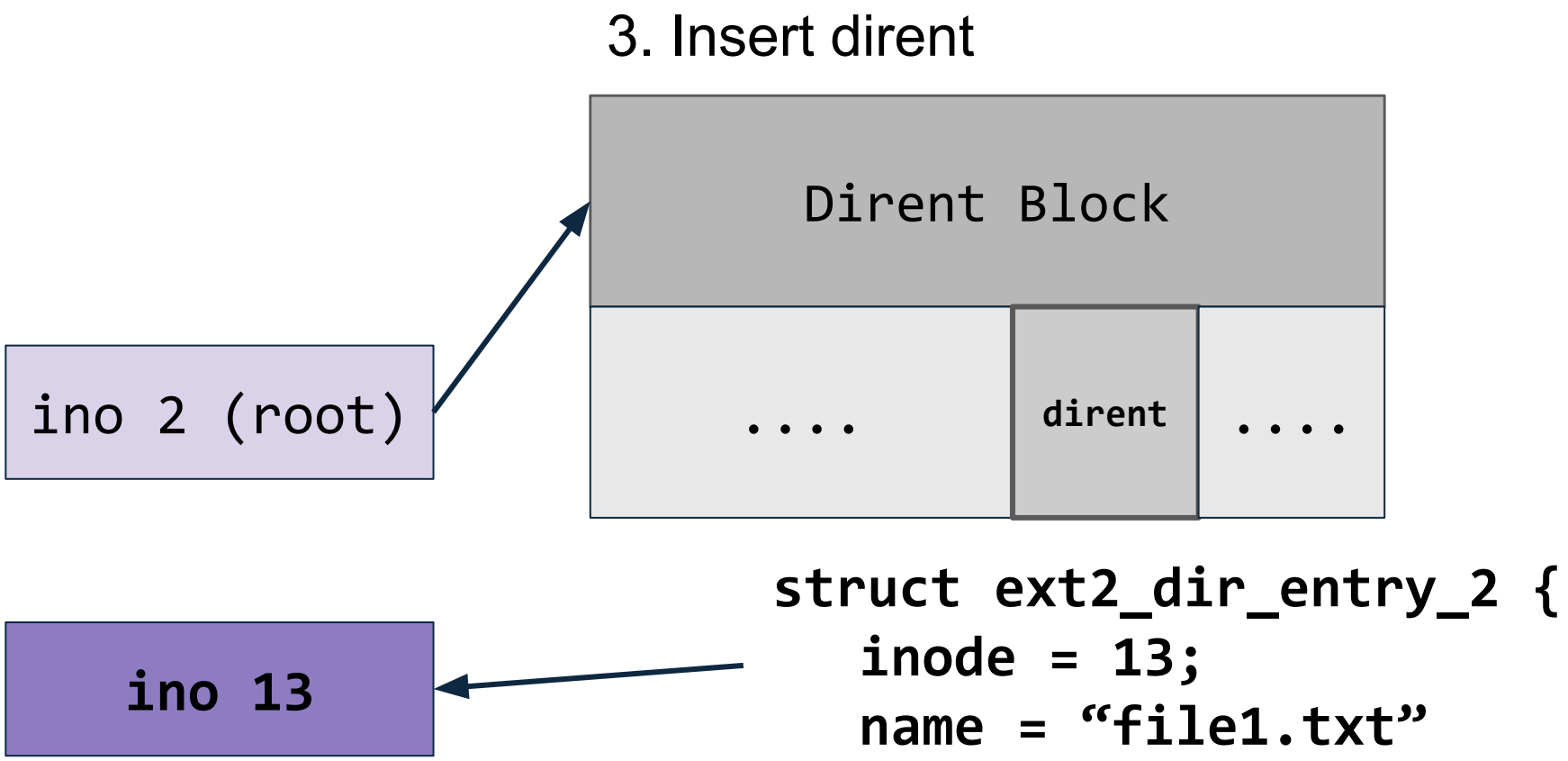
1. Find first free inode



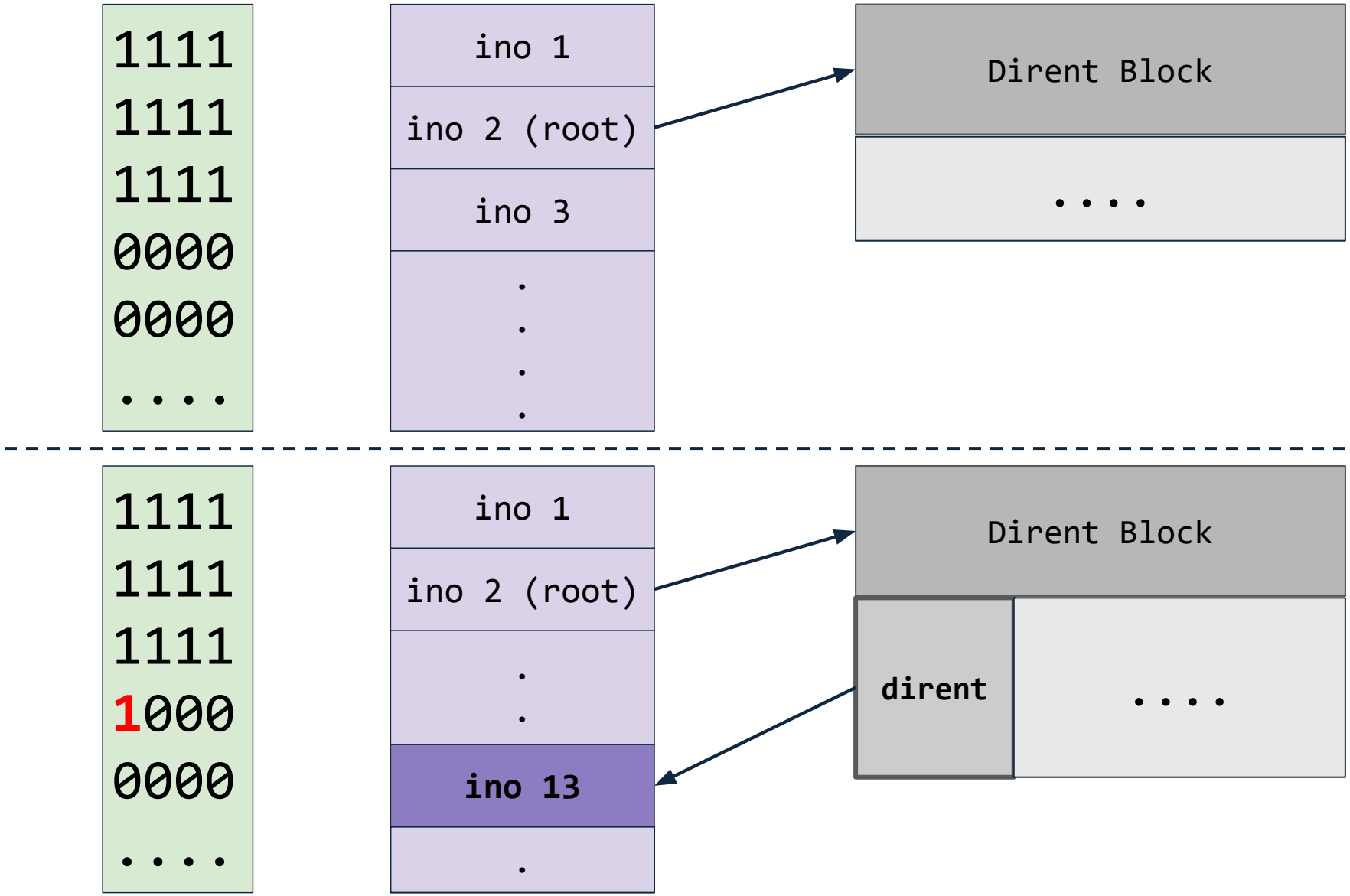
2. Initialize file inode



creat("/file1.txt", 0644)

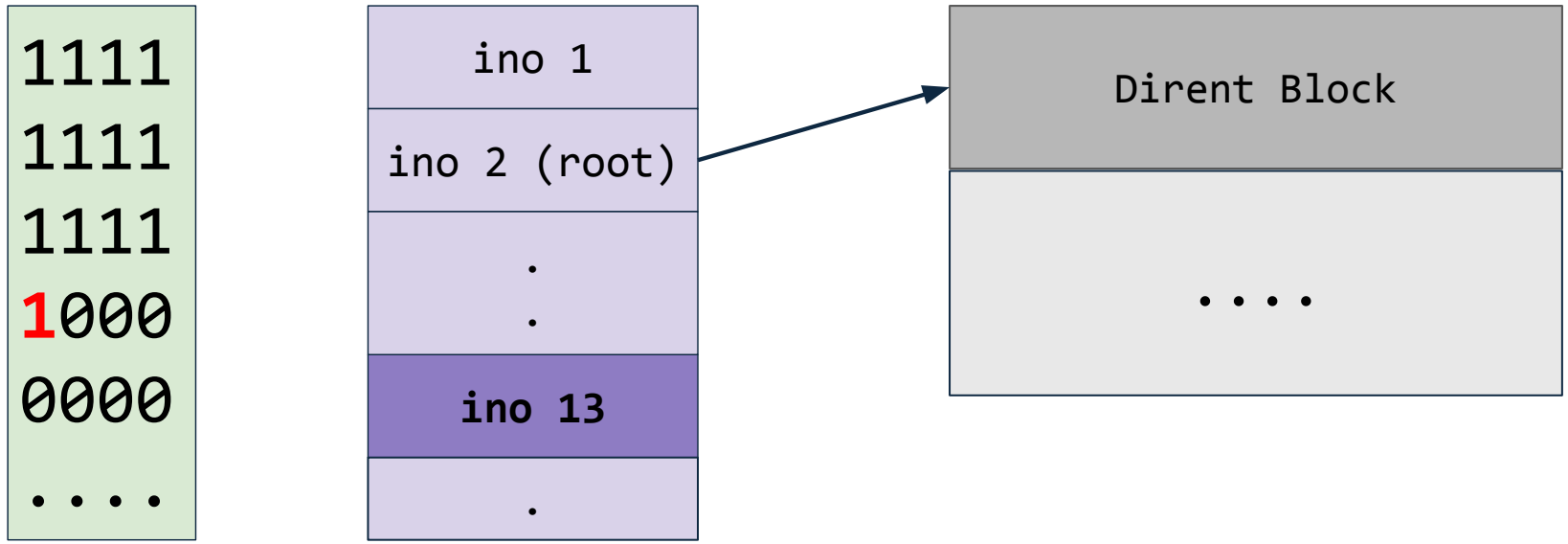


Before and After



- FS should "correctly" (***atomically***) *transition*
- What happens if it doesn't?

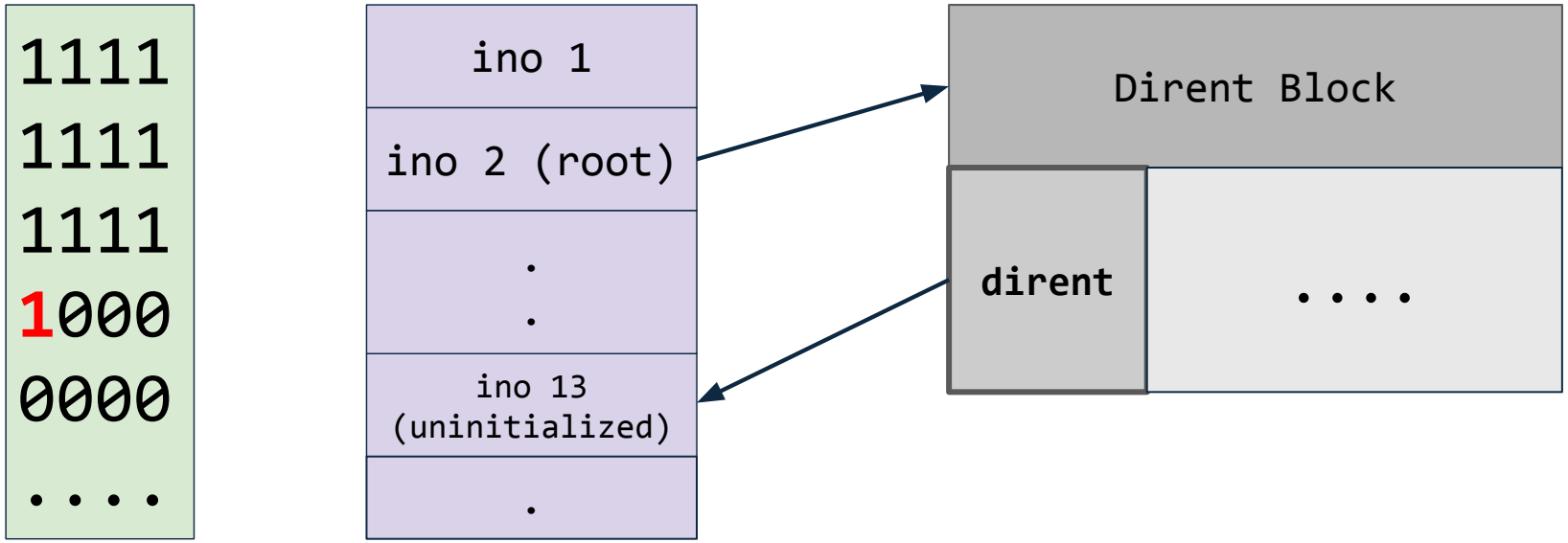
Scenario 1: Unwritten dirent



- lookup() tries to find “file1” in /
- Dirent doesn’t exist -> can’t reach file1
- But file + contents still exist, just not reachable in directory tree

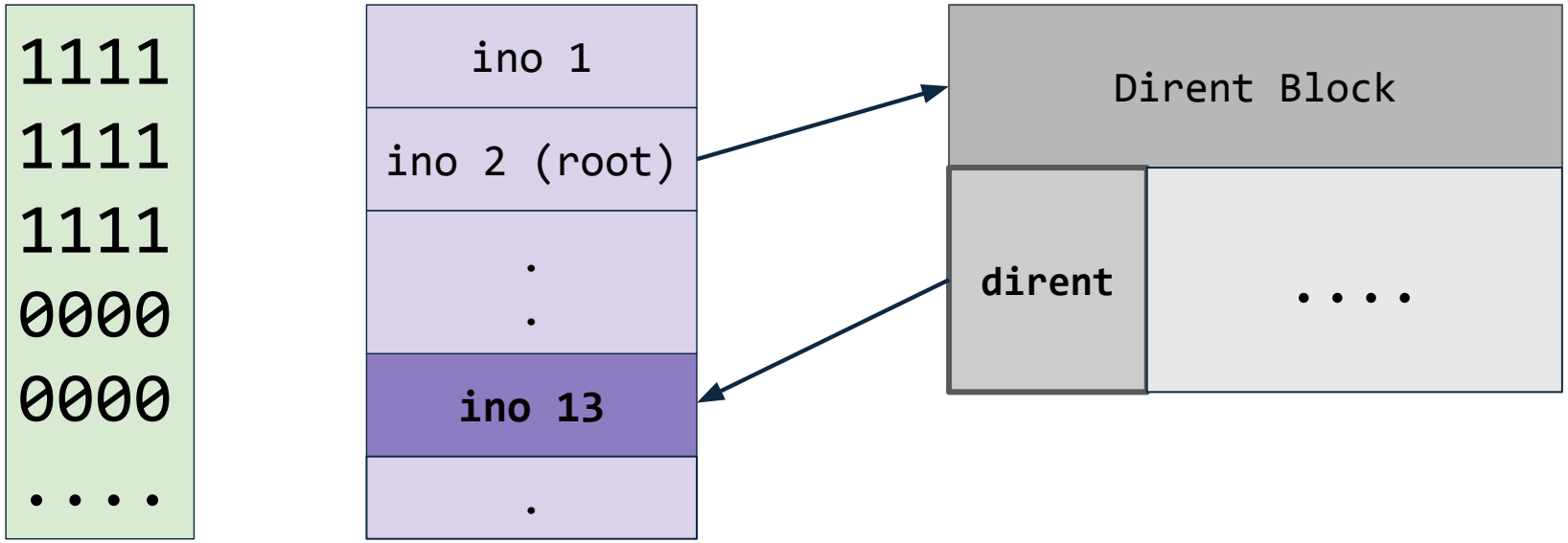
- **Orphan Inode**

Scenario 2: Unwritten inode



- We have file name, **but EVERYTHING else is lost**
  - Permissions, file contents (data blocks), …

Scenario 3: Unwritten bitmap



- We can navigate directory tree + fetch file contents just fine
- **But what happens when we try to create another file?**
  - **Double allocation**

How do FSes handle this?

- Option 1: Don’t do anything proactively. Just recover later
  - **fsck -** we’ll focus on this for the rest of the tutorial

- Option 2: Introduce proactive mechanisms to mitigate these issues
  - metadata/data journaling
    - Topic of today’s earlier lecture
    - Used by ext3/4 and xfs
  - Copy on Write
    - Used by btrfs
  - Soft Updates
    - Used to be used by Berkeley Fast File System on FreeBSD
    - Hard to maintain
    - If time permits we’ll flash the Soft Updates bonus slides up

Easy to discuss in theory - let's look at practice

- `e2fsprogs`: suite of userspace tools for managing ext2/3/4 disks
- 4 tools we cover today:
  - mkfs.ext4
    - Format drive w/ ext4 file system
  - fsck
    - Short for "File System Check"
    - Scans + repairs inconsistencies
  - dumpe2fs
    - Scans + reports on ext2/3/4 FS-wide metadata
  - debugfs
    - Inspect and modify specific files in ext2/3/4
    - Different than Linux's debugfs, which exposes generic debugging info at /sys/kernel/debug

Setting up a test image

- Source code available at https://github.com/sidharth-sundar/advos-25-demo/tree/main/Week-13-Demo

```
$ ./init.sh
```

```
#!/bin/bash
set -e

source ./header.sh

echo "[1] Creating image..."
truncate -s $SIZE $IMAGE

echo "[2] Creating ext4 filesystem..."
mkfs.ext4 -q $IMAGE

echo "[3] Initializing some files..."
sudo mount -t ext4 $IMAGE $MOUNT
cd $MOUNT
sudo sh -c "echo file1_contents > file1.txt"
sudo sh -c "echo file2_contents > file2.txt"
sudo sh -c "echo file3_contents > file3.txt"
sync
cd ..
sudo umount $MOUNT
```

- By default, init a 64MB disk image *ext4.img*
- **mkfs.ext4 ->** format disk w/ ext4 metadata
- Create 3 files w/ creat + write some contents

Inspecting global metadata with dumpe2fs

On your CLI, run: `$ dumpe2fs ext4.img`

```
Filesystem features:      has_journal ext_attr resize_inode dir_index orpha
n_file filetype extent 64bit flex_bg metadata_csum_seed sparse_super large_
file huge_file dir_nlink extra_isize metadata_csum
```

```
Inode size:                 256
Block size:                 4096
Journal inode:              8
```

```
Group 0: (Blocks 0-16383) csum 0x9964 [ITABLE_ZEROED]
  Primary superblock at 0, Group descriptors at 1-1
  Reserved GDT blocks at 2-8
  Block bitmap at 9 (+9), csum 0x7cf5efc5
  Inode bitmap at 25 (+25), csum 0x673c396c
  Inode table at 41-1064 (+41)
  14284 free blocks, 16369 free inodes, 2 directories, 16369 unused inodes
  Free blocks: 2098-2608, 2610-3120, 3122-16383
  Free inodes: 16-16384
```

- First half of output reads superblock metadata
- Second reads per-group metadata
- *Notice inodes 1-15 are allocated*

Inspecting per-file metadata with debugfs

On your CLI, run:

```
$ debugfs -w ext4.img # opens an interactive debugfs CLI
$ cd /
$ ls
```

```
 2  (12) .    2  (12) ..    11  (20) lost+found    13  (20) file1.txt
 14  (20) file2.txt    15  (4000) file3.txt
(END)
```

- Format is: `ino (rec_len)`
  - E.g. “.” and “..” both map to the root directory “/” with inode 2
  - Both dirents are 12B large
- You can check file inode number on your host machine using: `$ ls -i`
  - You’ll do this in the lab

Inspecting the inode table with debugfs

Within the debugfs CLI, run:

```
$ imap file1.txt
```

```
debugfs:  imap file1.txt
Inode 13 is part of block group 0
        located at block 41, offset 0x0c00
```

- Shows the block of the inode table where file1's inode is
- Shows the exact offset of the block where inode 13 starts
- We know from dumpe2fs that inode size is 256B here

Tracing the inode's indexing structure with debugfs

Within the debugfs CLI, run

```
$ bmap file1.txt 0
```

```
debugfs:  bmap
Usage: bmap [-a] <file> logical_blk [physical_blk]
debugfs:  bmap file1.txt 0
2097
```

- Logical block 0 = bytes 0-4095 of the file
- On the backing disk, located at physical block 2097
- Traverses the file inode's hashed B+tree

Selective corruption with debugfs

- Source code in `file1-nodirent.sh`, `file2-noinode.sh`, and `file3-noibit.sh`

- As each name suggests, we corrupt one of the 3 metadata structures associated with the file

What happens when we try to access the file system?

```
$ run each of the ./file<n>-*.sh scripts
$ ./mount.sh
$ cd mnt
$ ls -alih
```

```
ls: cannot access 'file2.txt': Structure needs cleaning
total 20
 ? -????????? ? ?    ?          ?                ? file2.txt
15 -rw-r--r-- 1 root root    15 Nov 28 18:14 file3.txt
11 drwx------ 2 root root 16384 Nov 28 18:14 lost+found
```

1. Why don’t we see file1.txt?
2. What happened to file2.txt?
3. Why does file3.txt seem fine?

What if we try to create a file now?

```
$ # within the mnt directory
$ sudo touch file4.txt
```

```
13/mnt2# sudo touch file4.txt
touch: cannot touch 'file4.txt': Input/output error
```

```
$ sudo dmesg | tail
```

```
[4470489.636048] EXT4-fs error (device loop9): ext4_lookup:1789: inode #2:
comm ls: deleted inode referenced: 14
[4470636.910395] EXT4-fs error (device loop9): __ext4_new_inode:1278: comm
touch: failed to insert inode 15: doubly allocated?
```

- We saw on previous slide that file3.txt has ino=15
- *But we cleared the bitmap!*

What does dumpe2fs tell us now?

```
$ ./umount.sh
$ dumpe2fs ext4.img
```

```
Filesystem state:         clean with errors
```

```
Free inodes:              0
```

```
FS Error count:           2
First error time:         Fri Nov 28 18:15:25 2025
First error function:     ext4_lookup
First error line #:       1789
First error inode #:      2
First error err:          EFSCORRUPTED
Last error time:          Fri Nov 28 18:17:53 2025
Last error function:      __ext4_new_inode
Last error line #:        1278
Last error err:           EFSCORRUPTED
```

- The filesystem has catastrophically failed!
- On lockdown - no new inodes can be created
- How do we recover?

fsck

- "File System Check"

- One per-fs
  - e2fsck (what we'll use)
  - xfs_repair
  - btrfs check

- An *offline* scanning tool for restoring a FS to a consistent state

fsck warning

DO NOT RUN FSCK ON YOUR ROOT DISK

```
$ fsck /dev/sda or fsck /dev/nvme1n1 # ABSOLUTELY DO NOT DO
```

DO NOT RUN FSCK WITHOUT ARGUMENTS

```
$ fsck # ABSOLUTELY DO NOT DO, defaults to fsck <host device>
```

Running fsck on a mounted disk can corrupt data or render the FS unusable.

ONLY run the command on the next slide

5 stages of fsck

```
$ e2fsck ./ext4.img
```

```
13# e2fsck ext4.img
e2fsck 1.47.2 (1-Jan-2025)
Group descriptor 0 has invalid unused inodes count 16369.  Fix<y>? yes
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Entry 'file2.txt' in / (2) has deleted/unused inode 14.  Clear<y>? yes
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Unattached inode 13
Connect to /lost+found<y>? yes
Inode 13 ref count is 2, should be 1.  Fix<y>? yes
Pass 5: Checking group summary information
Block bitmap differences:  -2609
Fix<y>? yes
Free blocks count wrong for group #0 (14284, counted=14285).
Fix<y>? yes
Free blocks count wrong (14284, counted=14285).
Fix<y>? yes
Inode bitmap differences:  -14
Fix<y>? yes
Free inodes count wrong for group #0 (16368, counted=16370).
Fix<y>? yes
Free inodes count wrong (0, counted=16370).
Fix ('a' enables 'yes' to all) <y>? yes

ext4.img: ***** FILE SYSTEM WAS MODIFIED *****
ext4.img: 14/16384 files (7.1% non-contiguous), 2099/16384 blocks
```

1. Check individual inodes (files/dirs).
2. Iterate inode table. If S_IFDIR set, check that all dirents are valid
3. Recursively traverse from root, ensure all directories can be reached

5 stages of fsck

```
$ e2fsck ./ext4.img
```

```
13# e2fsck ext4.img
e2fsck 1.47.2 (1-Jan-2025)
Group descriptor 0 has invalid unused inodes count 16369.  Fix<y>? yes
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Entry 'file2.txt' in / (2) has deleted/unused inode 14.  Clear<y>? yes
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Unattached inode 13
Connect to /lost+found<y>? yes
Inode 13 ref count is 2, should be 1.  Fix<y>? yes
Pass 5: Checking group summary information
Block bitmap differences:  -2609
Fix<y>? yes
Free blocks count wrong for group #0 (14284, counted=14285).
Fix<y>? yes
Free blocks count wrong (14284, counted=14285).
Fix<y>? yes
Inode bitmap differences:  -14
Fix<y>? yes
Free inodes count wrong for group #0 (16368, counted=16370).
Fix<y>? yes
Free inodes count wrong (0, counted=16370).
Fix ('a' enables 'yes' to all) <y>? yes

ext4.img: ***** FILE SYSTEM WAS MODIFIED *****
ext4.img: 14/16384 files (7.1% non-contiguous), 2099/16384 blocks
```

4. Check inode link count (# of directories a file is in vs what's reported in the inode) to find orphaned files

5. Check general metadata
   - Block bitmap
   - Inode bitmap
   - Superblock
   - Group descriptors

How does fsck fix issues with the 3 files?

fsck: file1.txt

```
$ e2fsck ./ext4.img
```

```
13# e2fsck ext4.img
e2fsck 1.47.2 (1-Jan-2025)
Group descriptor 0 has invalid unused inodes count 16369.  Fix<y>? yes
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Entry 'file2.txt' in / (2) has deleted/unused inode 14.  Clear<y>? yes
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Unattached inode 13
Connect to /lost+found<y>? yes
Inode 13 ref count is 2, should be 1.  Fix<y>? yes
Pass 5: Checking group summary information
Block bitmap differences:  -2609
Fix<y>? yes
Free blocks count wrong for group #0 (14284, counted=14285).
Fix<y>? yes
Free blocks count wrong (14284, counted=14285).
Fix<y>? yes
Inode bitmap differences:  -14
Fix<y>? yes
Free inodes count wrong for group #0 (16368, counted=16370).
Fix<y>? yes
Free inodes count wrong (0, counted=16370).
Fix ('a' enables 'yes' to all) <y>? yes

ext4.img: ***** FILE SYSTEM WAS MODIFIED *****
ext4.img: 14/16384 files (7.1% non-contiguous), 2099/16384 blocks
```

No dirent pointing to file (link_count = 0), but inode reports link_count = 1

All file contents can still be accessed via inode

Moves file to lost+found, lets user manually recover after

fsck: file2.txt

```
$ e2fsck ./ext4.img
```

```
13# e2fsck ext4.img
e2fsck 1.47.2 (1-Jan-2025)
Group descriptor 0 has invalid unused inodes count 16369.  Fix<y>? yes
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Entry 'file2.txt' in / (2) has deleted/unused inode 14.  Clear<y>? yes
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Unattached inode 13
Connect to /lost+found<y>? yes
Inode 13 ref count is 2, should be 1.  Fix<y>? yes
Pass 5: Checking group summary information
Block bitmap differences:  -2609
Fix<y>? yes
Free blocks count wrong for group #0 (14284, counted=14285).
Fix<y>? yes
Free blocks count wrong (14284, counted=14285).
Fix<y>? yes
Inode bitmap differences:  -14
Fix<y>? yes
Free inodes count wrong for group #0 (16368, counted=16370).
Fix<y>? yes
Free inodes count wrong (0, counted=16370).
Fix ('a' enables 'yes' to all) <y>? yes

ext4.img: ***** FILE SYSTEM WAS MODIFIED *****
ext4.img: 14/16384 files (7.1% non-contiguous), 2099/16384 blocks
```

The inode was zeroed out, *so we don't know what blocks were mapped to the file*

No way to recover, so fsck deletes it

fsck: file3.txt

```
$ ./cleanup.sh
$ ./init_fs.sh
$ ./corrupt-ext4-full.sh
$ cat fsck.txt
```

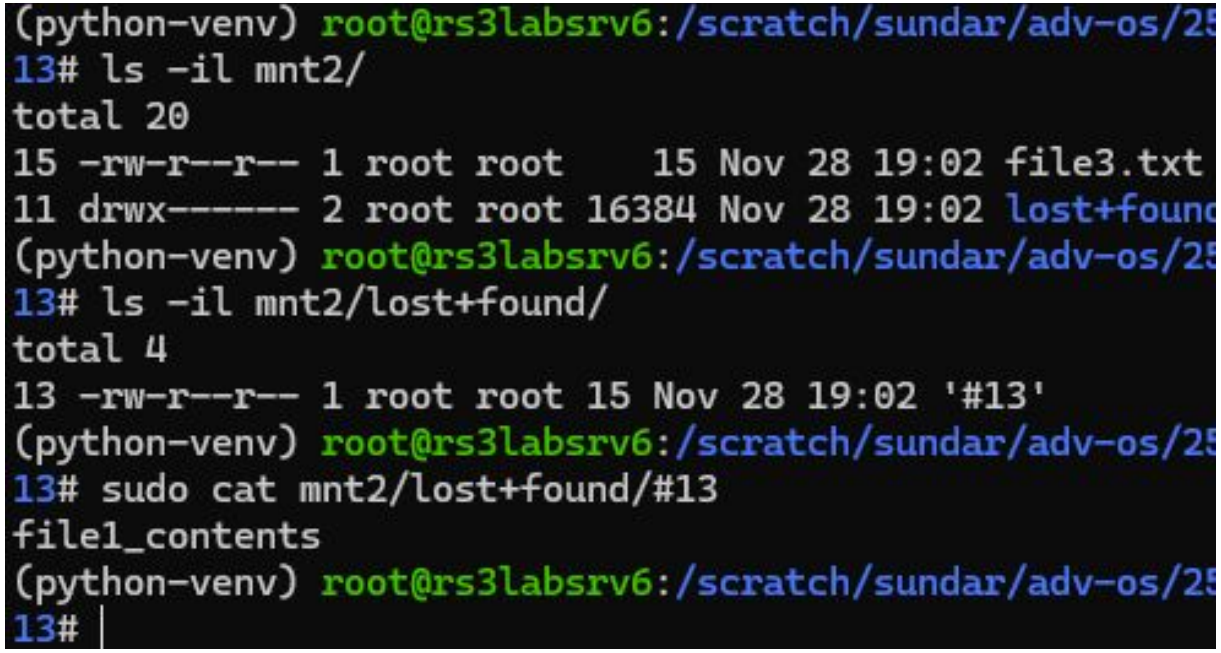
(attempting to create a file in slide 19 set the bitmap, so we need to rerun w/o creating a new file)

fsck can detect if an inode is active w/ an unset bitmap, so it just sets it

```
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Entry 'file2.txt' in / (2) has deleted/unused inode 14.  Clear? yes

Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Unattached inode 13
Connect to /lost+found? yes

Inode 13 ref count is 2, should be 1.  Fix? yes

Pass 5: Checking group summary information
Block bitmap differences:  -2609
Fix? yes

Free blocks count wrong for group #0 (14284, counted=14285).
Fix? yes

Free blocks count wrong (14284, counted=14285).
Fix? yes

Inode bitmap differences:  -14 +15
Fix? yes

Free inodes count wrong for group #0 (16369, counted=16370).
Fix? yes

Free inodes count wrong (16369, counted=16370).
Fix? yes
```

Inspecting the repaired fs

```
$ ./mount.sh
$ ls -il mnt
$ sudo ls -il mnt/lost+found
$ sudo cat mnt/lost+found/#13
```



- We can locate file3
- We've lost file1's name, but its contents are preserved in lost+found
- file2 is gone

Closing notes on fsck

- fsck is sometimes **more** powerful than journaling

- Why don't we just run fsck every time we mount a FS and not journal?
  - *It's too slow*
  - You have to scan *all* the metadata multiple times over
  - Example from personal experience
    - Accidentally corrupted data on a lab server running RAID0 over 2 SSDs totaling 1.7TB
    - fsck took **7 hours** to complete
  - Better to pay the up-front cost of journaling

Interesting Readings

- Copy-On-Write
  - BTRFS: The Linux B-Tree Filesystem: https://dl.acm.org/doi/pdf/10.1145/2501620.2501623
  - B-trees, Shadowing, and Clones: https://dl.acm.org/doi/pdf/10.1145/1326542.1326544
- Soft Updates
  - “Soft updates, hard problems”: https://lwn.net/Articles/339337/
  - *1999 soft updates paper [PDF]*
  - SquirrelFS: using the Rust compiler to check file-system crash consistency: https://www.usenix.org/conference/osdi24/presentation/leblanc

Interesting Readings

- File System Fuzzing
  - Janus: https://taesoo.kim/pubs/2019/xu:janus.pdf
  - Hydra: https://dl.acm.org/doi/10.1145/3341301.3359662
  - Monarch: https://www.usenix.org/conference/atc24/presentation/lyu
  - CrashMonkey: https://www.cs.utexas.edu/~vijay/papers/osdi18-crashmonkey.pdf

Soft Updates Bonus Slide

- Good article, titled “Soft updates, hard problems”: https://lwn.net/Articles/339337/
- Here’s our favorite part

- *“Page 7 of the 1999 soft updates paper [PDF] begins the descriptions of specific kinds of update dependency structures and their relationships to each other. I've read this paper at least 15 times, and each time I when get to page 7, I'm feeling pretty good and thinking, "Yeah, okay, I must be smarter now than the last time I read this because I'm getting it this time," - and then I turn to page 8 and my head explodes.”*

Page 8

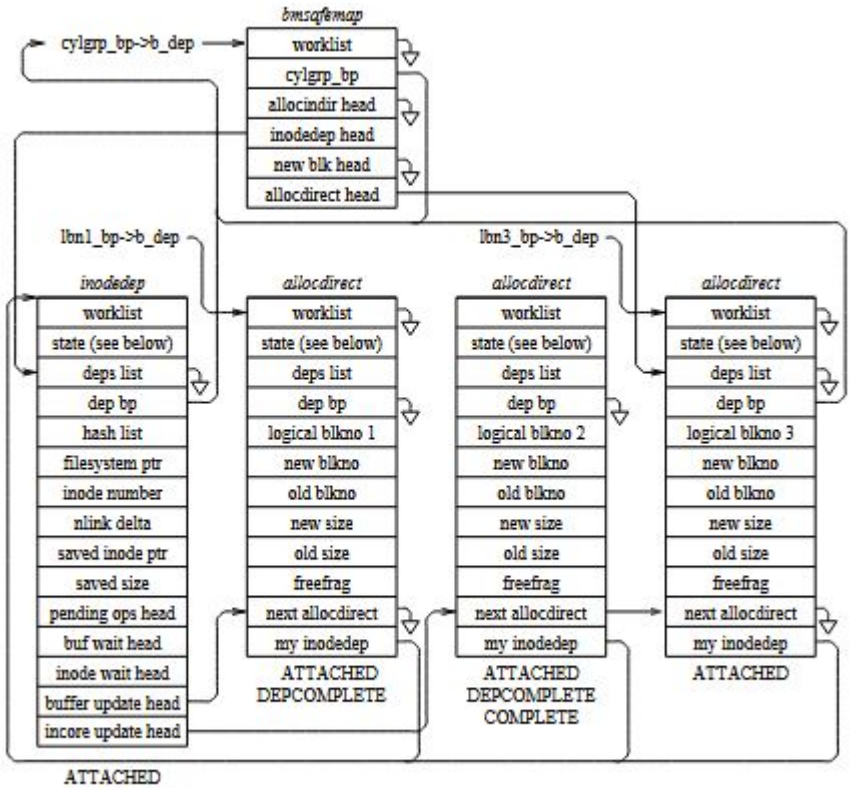


Figure 4: *Direct Block Allocation Dependencies*

3.5. Indirect Block Dependency Tracking

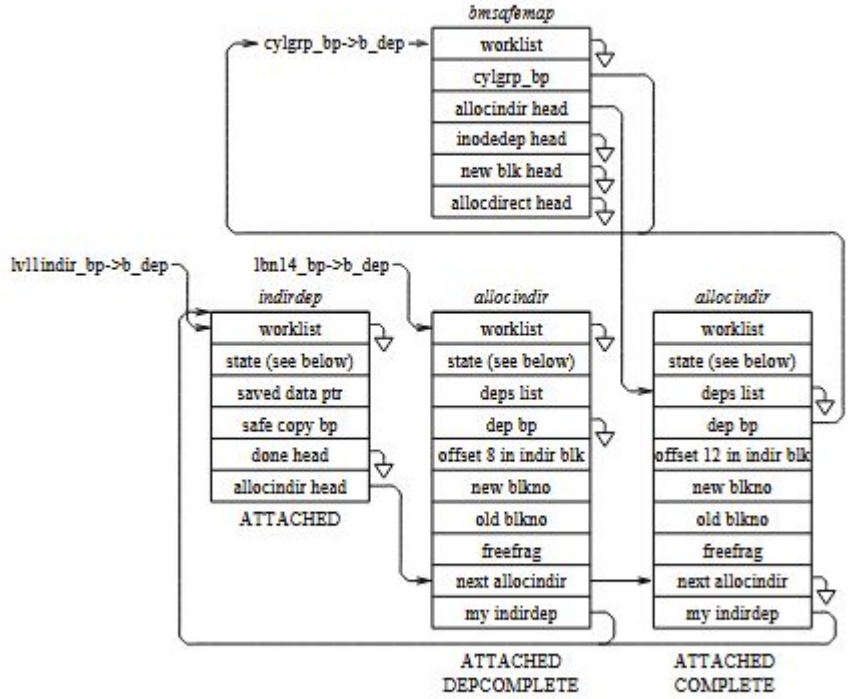


Figure 5: *Indirect Block Allocation Dependencies*

So, you can imagine why this is only maintained in FFS as a legacy option these days