Section 9: Intro to Lab 5

CSE 451 18WI

Debugging Tip: GDB Watchpoints

- Many Lab 4 bugs happen due to unintentional inconsistencies in memory
  - Corrupt code/stack page, corrupt virtual/physical metadata page, etc.
  - Finding what became corrupted is relatively trivial
- Watchpoints are *super useful* for debugging memory corruption!
  - Breaks in GDB whenever data at an address being watched changes!
- Usage Steps
  - Identify address of some data that became corrupted
    - e.g. A vpage_info/core_map_entry field that should've or shouldn't have changed
  - Without recompiling, in GDB: **watch *(data_type *) 0xaddress**

GDB Watchpoint Example

- Let's say that for a particular **struct vpi_page *info**, you learn its **next** field was set to NULL when it shouldn't have been.
- In GDB
  - **p/x &info->next**
    - Let's say it's **0xdeadbeef**
  - (without any code changes, run **make qemu-gdb** and **make gdb** again)
  - **watch *(struct vpi_page **) 0xdeadbeef**

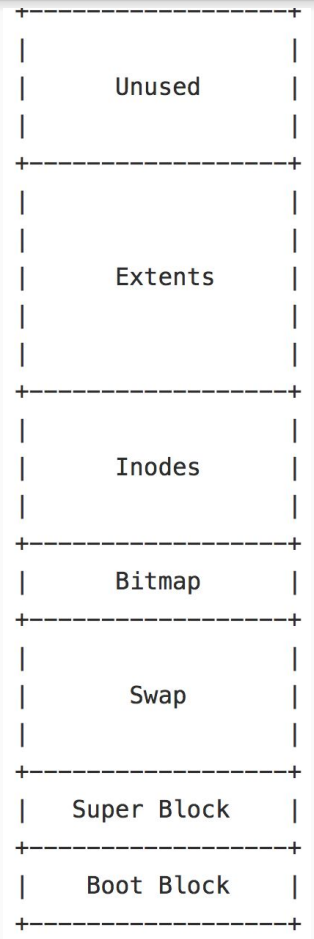
Lab 5

Two Parts

- A - Enable file creation, file writes and file appends
- B - Make the file system crash safe

Part A: Create, Write & Append

XK Disk Format



- Boot Block
  - Used by the boot loader
- Super Block
  - Describes how the disk is formatted
- Swap
  - Used for paging
- Bitmap
  - Keeps track of which blocks are free/used
- Inodes
  - Inode table holds an inode for each file (inode holds file metadata)
- Extents
  - Where file data is stored

See lab5.md for the disk diagram with block offsets included

struct dinode - inc/fs.h

```
25  // On-disk inode structure
26  struct dinode {
27    short type;           // File type
28    short devid;          // Device number (T_DEV only)
29    uint size;            // Size of file (bytes)
30    struct extent data; // Data blocks of file on disk
31    char pad[46];         // So disk inodes fit contiguosly in a block
32  };
```

struct extent - inc/extent.h

```
// represents a contiguous block on disk of data
struct extent {
  uint startblkno; // start block number
  uint nblocks;    // n blocks following the start block
};
```

struct dinode - inc/fs.h

```
25  // On-disk inode struc
26  struct dinode {
27    short type;
28    short devid;
29    uint size;
30    struct extent data;
31    char pad[46];
32  };
```

Why is there padding?

struct dinode - inc/fs.h

```
25  // On-disk inode struc
26  struct dinode {
27    short type;
28    short devid;
29    uint size;
30    struct extent data;
31    char pad[46];
32  };
```

```
// represents a contiguous block on disk of data
struct extent {
  uint startblkno; // start block number
  uint nblocks;    // n blocks following the start block
};
```

2+
2+
4+
8+
46
~~62~~

Size should be a power of 2 to ensure no dinode is split across a page

Sizeof evaluates to **64 bytes**, due to padding (2 bytes at end)

`struct inode` - `inc/file.h`

```
6  // in-memory copy of an inode
7  struct inode {
8    uint dev;  // Device number
9    uint inum; // Inode number
10   int ref;   // Reference count
11   struct sleeplock lock;
12
13   short type; // copy of disk inode
14   short devid;
15   uint size;
16   struct extent data;
17 };
```

If you modify **`struct dinode`**, make sure to update **`struct inode`** as well!

Write

- Modify **`writei`** in kernel/fs.c so an inode can be used to write to disk
- Use **`bread, bwrite, brelse`**
- See **`readi`** for an example
- Also, change **`open`** to allow **`O_RDWR`**

Append

- Need to be able to extend the size of a file
- Allocate additional space using extra block pointers or extra extent pointers

Example: Need to be able to handle the case where the user tries to append to File 1 when the disk's extent region is laid out as follows.

Create

- Create a new file when **O_CREATE** is passed to **open**

“You need to create a empty inode on disk, change the root directory to add a link to the new file, and (depending on your disk layout) change bitmap on disk. The inode file length itself will change, so don't forget to update this as well.”

Note: File deletion is not required

Part B: Crash-Safe File System

Let’s append to a file...

Simple example: Say we have a file “cat.txt”. It has a single extent that’s 1 block long. This block is half full, meaning the file size is 256 bytes. We want to append 50 bytes to the end of the file.

Need to write multiple blocks to disk:

- The block containing the inode, since we need to update the file size
- The block itself that we’re adding the 50 bytes to

Simple Example Continued

We first update the size of the file, changing 256 to 306 in the inode block. We write this change to disk. Next we get ready to write the 50 bytes to the extent block...

Simple Example Continued

CRASH

Image: https://www.petbucket.com/blog/63640/how-to-keep-your-cat-from-chewing-cables.html

XK reboots...

Oh look cat.txt is now 306 bytes long! Let's go read it!

Because we never wrote the 50 bytes to disk, that last 50 bytes we read will not be what we were expecting...

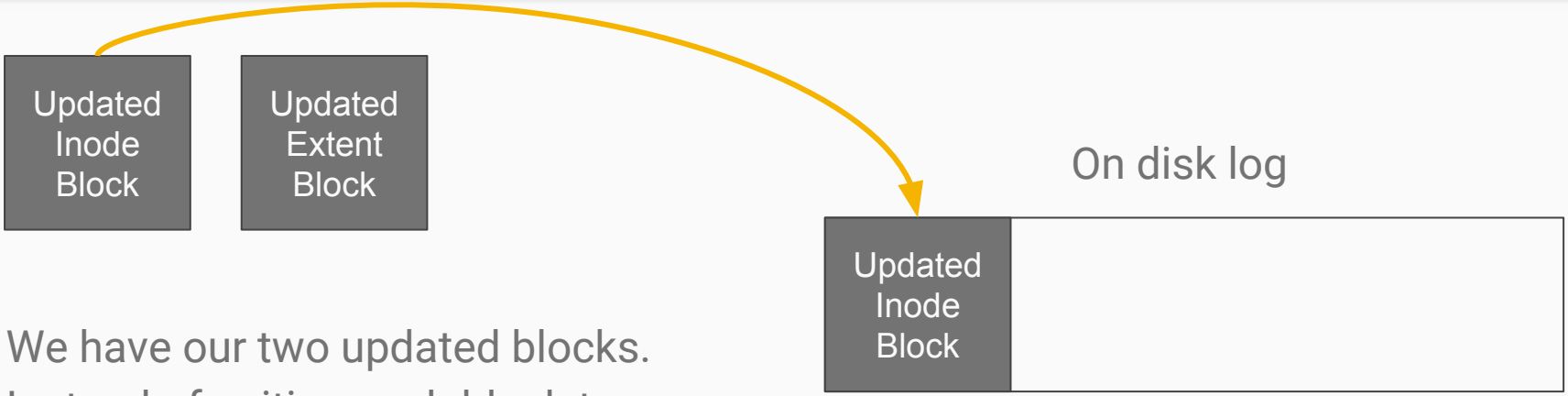
Image: https://brightside.me/wonder-animals/the-life-of-a-proud-cat-owner-expectations-vs-reality-111755/

How to make XK filesystem crash safe?

There are several different ways to do this. We recommend you implement **journaling**.

Let’s walk through the previous example, this time using journaling...

Simple Example with Journaling



We have our two updated blocks. Instead of writing each block to their respective areas on disk, we write both to the log

Simple Example With Journaling Continued

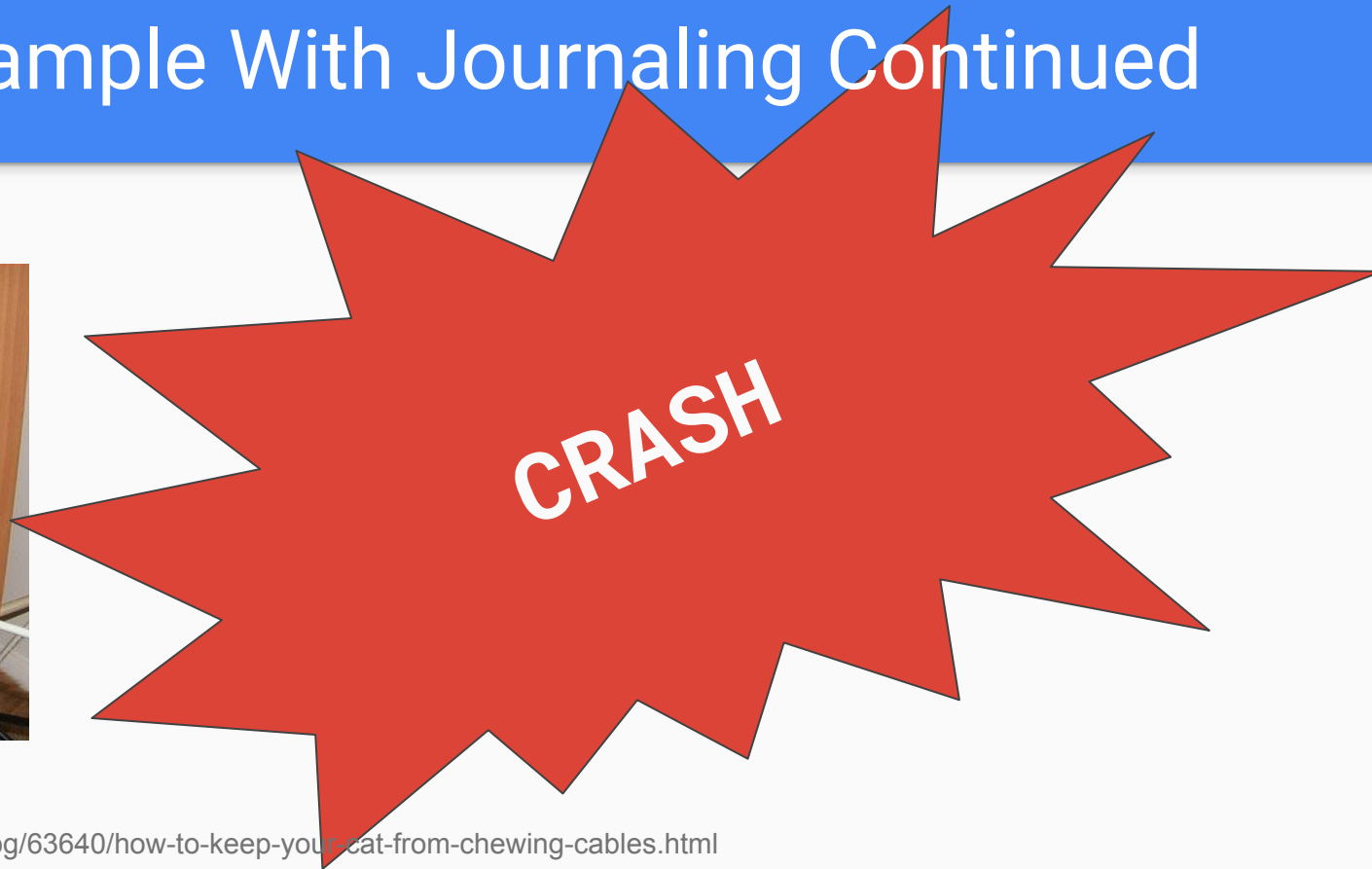


Image: https://www.petbucket.com/blog/63640/how-to-keep-your-cat-from-chewing-cables.html

XK reboots...

Oh look cat.txt is still 256 bytes long. When we read is it is as expected.

Because we only wrote to the log and not to disk, the data on disk is still valid.

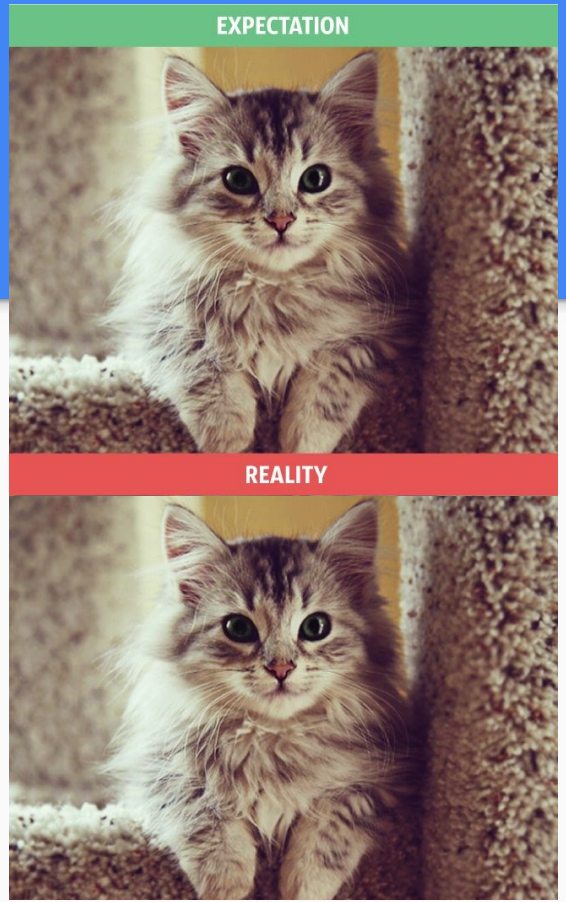
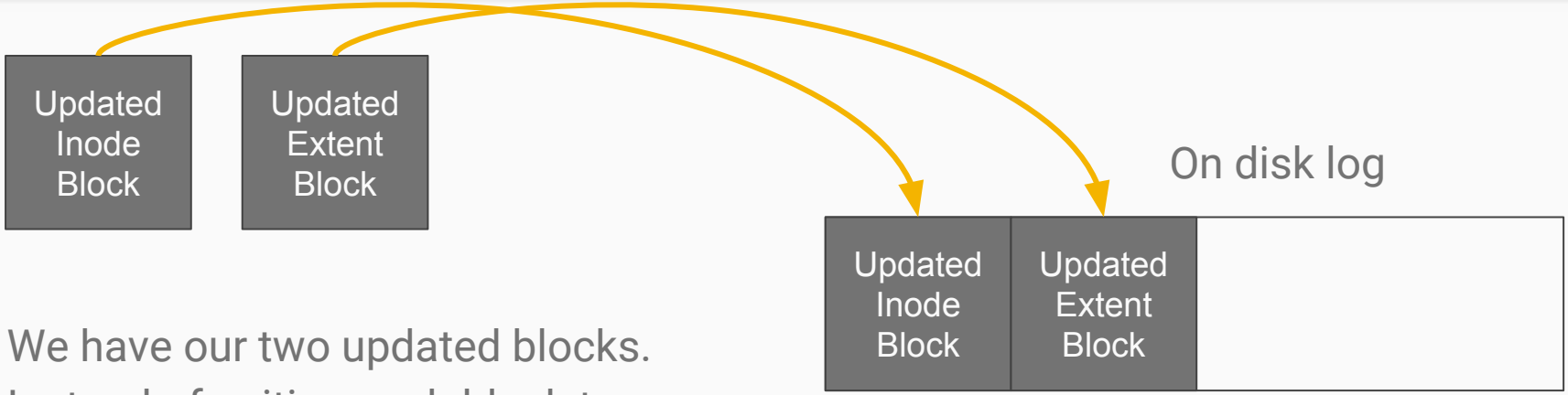


Image: https://brightside.me/wonder-animals/the-life-of-a-proud-cat-owner-expectations-vs-reality-111755/

Let’s see journaling succeed...

Simple Example with Journaling

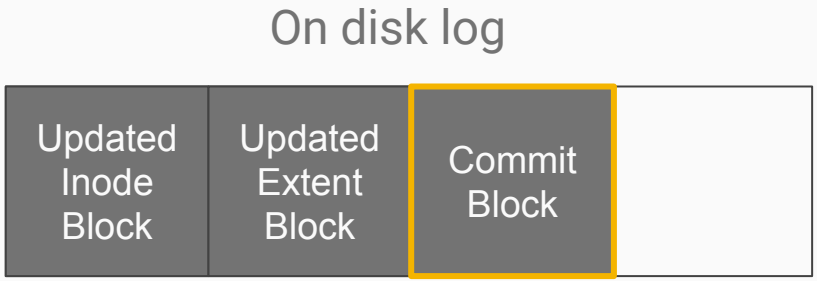


We have our two updated blocks. Instead of writing each block to their respective areas on disk, we write both to the log.

Simple Example with Journaling

Once all modified blocks have been written to the log, we need to write something that indicates all parts have been written to the log (a commit message).

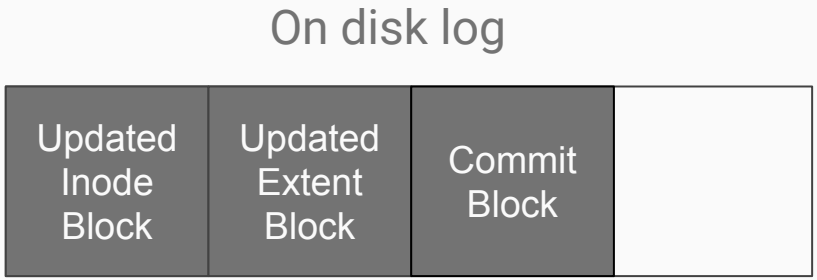
If on reboot we don't see this log commit message, we shouldn't try to apply the changes in the log.



Simple Example with Journaling

Now that all parts have been written to the log, we apply them to the proper section of disk one block at a time.

If XK crashes during this process, all necessary blocks are stored in the log so we can simply re-apply them on boot. (Applying log actions should be idempotent.)



Where to place the log?

Place the log in the metadata area before the inodes. This will allow you to use the **`bread/bwrite`** interface to interact with the log.