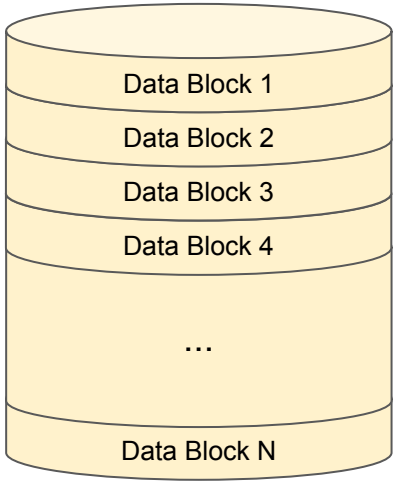
Using the page cache

Mitchell Gouzenko

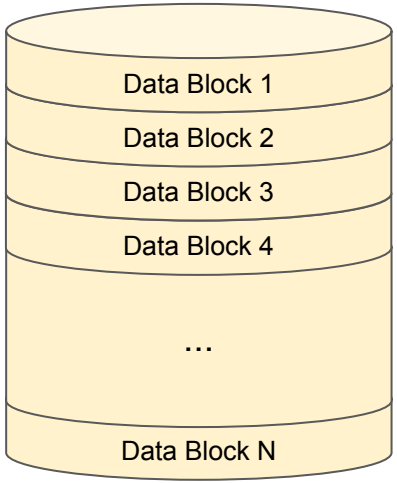
Block Devices

- Filesystem are mounted on top of block devices
    - Ex: /dev/sda1
- Each device consists of contiguous data blocks.
- If you just try to read the raw bytes, it will look like garbage with bits of files scattered around.
    - But filesystems know how to treat the structure of data on disk, such that it makes sense to that specific FS.
    - Ex: ReiserFS will not be able to make sense out of a disk laid out in the format of ext4.
- You can write to device files, and the data WILL get written to the underlying disk.
    - If /dev/sdb is a USB drive, “echo foo > /dev/sdb” will write to that drive. `dd` utility often used for doing this.



Block Devices (continued)

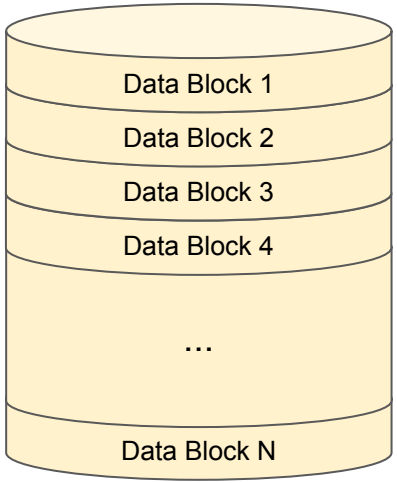
- When writing to regular file (for example, a file on an EXT4-formatted drive), data doesn't go to disk right away.
    - Note: device files like /dev/sda1 are not "regular" files.
- Data sits in kernel buffer, eventually will get synced to disk.
    - Advantage
        - Writing to buffer in kernel memory is much faster than writing to disk.
    - FS Buffer layer used to be called "Buffercache", but now the "Page Cache" (a different kernel subsystem) is used.
- Codepath for *actually* getting data to disk is lower-level than we need to worry about.
    - Involves disk scheduling and speaking to disk drivers (SATA II drivers, for instance)



Block Devices: The Page Cache

- Page cache can read individual disk blocks whose size is determined by the filesystem (must be multiple of 512).
- We will use 3 important functions to interface with page cache:

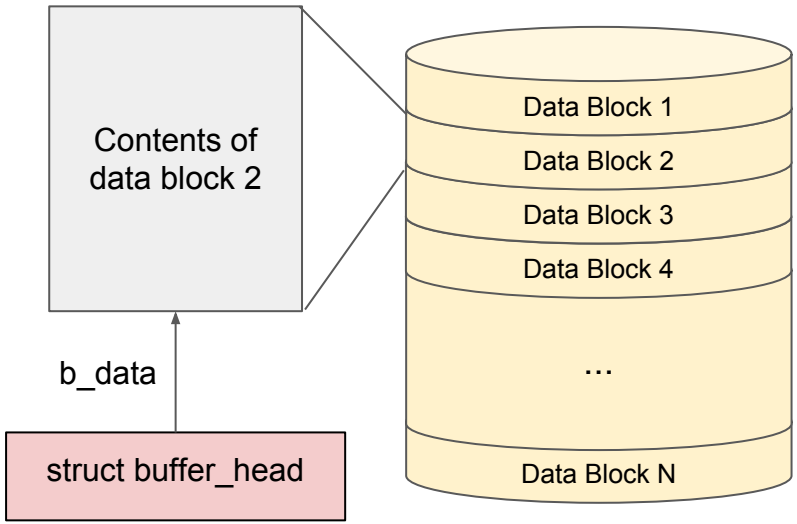
```
static inline struct buffer_head *
sb_bread(struct super_block *sb, sector_t block);

void mark_buffer_dirty(struct buffer_head *bh);

static inline void brelse(struct buffer_head *bh);
```



The Page Cache: `sb_bread`

```
static inline struct buffer_head *
sb_bread(struct super_block *sb, sector_t block);
```

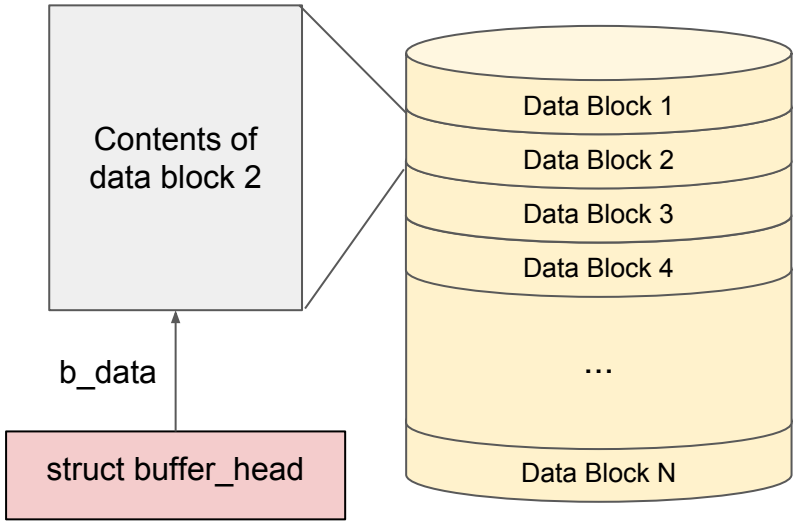
- VFS superblock struct has the name of the device upon which filesystem is mounted.
- sb_bread reads the corresponding `block` from the device specified in `sb` and stores it in a buffer.
- Example shown in diagram: `sb_bread(sb, 2)`
- Note: if data and buffer_head for block 2 are still in memory, sb_bread doesn't need to read from the disk. It just returns the buffer_head.



The Page Cache: `mark_buffer_dirty`

```
void mark_buffer_dirty(struct buffer_head *bh);
```

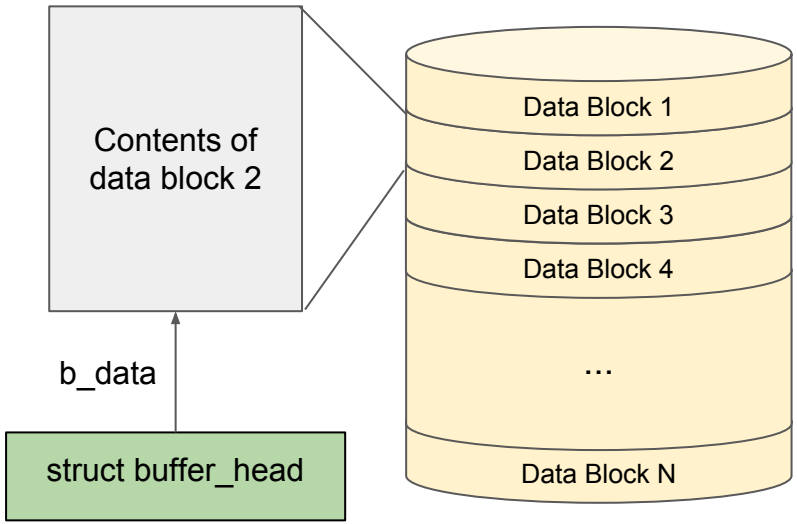
- b_data points to a contiguous chunk of bytes that is backed by Block 2 on device.
- If you modify these bytes, you want the changes to get onto the device eventually.
- This function flags the `buffer_head` as dirty.
- Dirty `buffer_heads` need their data to be synced to disk eventually!



The Page Cache: `brelse`

```
static inline void brelse(struct buffer_head *bh);
```

- After `sb_bread`, `buffer_head` and data block contents are pinned in memory. The page cache won't remove them.
- brelse will “release” the `buffer_head`
  - Kernel may or may not free the `buffer_head`.
    - Only frees when there is memory pressure.
    - This is why computers seem to use up all the RAM. Most available memory is used to hold caches of file contents to speed up I/O.
  - If kernel decides to free `buffer_head`, it will sync its data to disk, **but only if `buffer_head` marked dirty**!
- Note: from now on, I represent all `buffer_heads` in red. When the `buffer_head` is released with `brelse()`, I will color it green.



The Page Cache: Example Write

```
// Read 3rd data block
bh = sb_bread(sb, 3);

// Copy 10 bytes from src to data buffer.
memcpy(bh->b_data, src, 10);

// Mark the buffer_head dirty so that kernel will
// eventually sync it to disk.
mark_buffer_dirty(bh);

// Release the buffer. Kernel will decide to
// either keep buffer_head around, or free
// the buffer_head and flush its data.
brelse(bh);
```