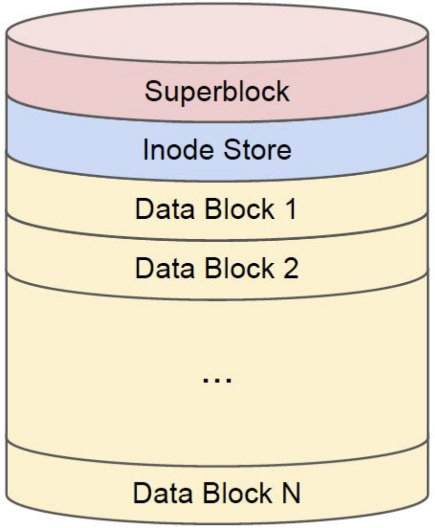
# EZFS Specification

Zijian Zhang and Emma Nieh

## EZFS Layout

- Each slice shown to the right is one block.
- Each block is 4096 bytes.



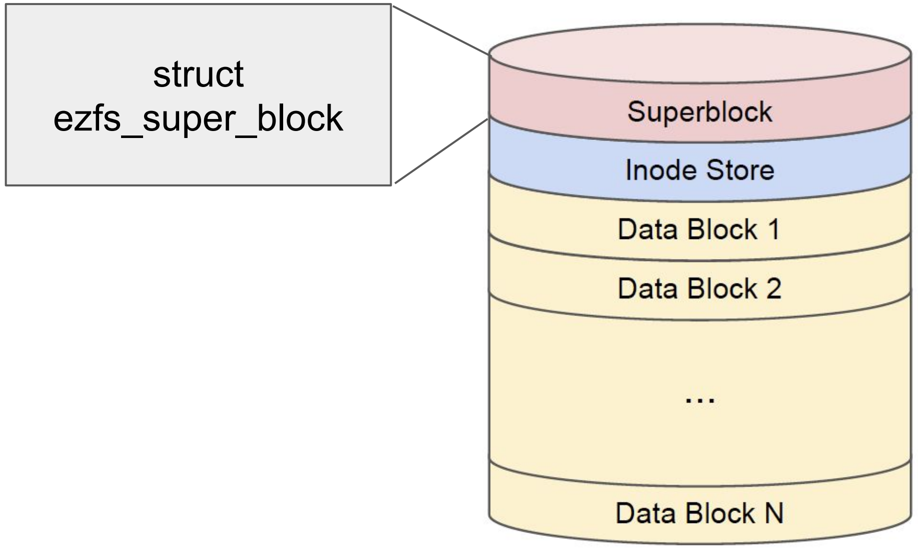
# The Superblock

- The ezfs_super_block is padded to be 4096 bytes, so it fits directly in the first block.
- Contains bit vectors representing which inodes and data blocks are free.
- Bit vectors declared by macros:

```
DECLARE_BIT_VECTOR(free_inodes, EZFS_MAX_INODES)

DECLARE_BIT_VECTOR(free_data_blocks,
EZFS_MAX_DATA_BLKS)
```

- We indicate that an inode or date block is **occupied** by setting the corresponding location in the bit vector to 1



# The Inode Store

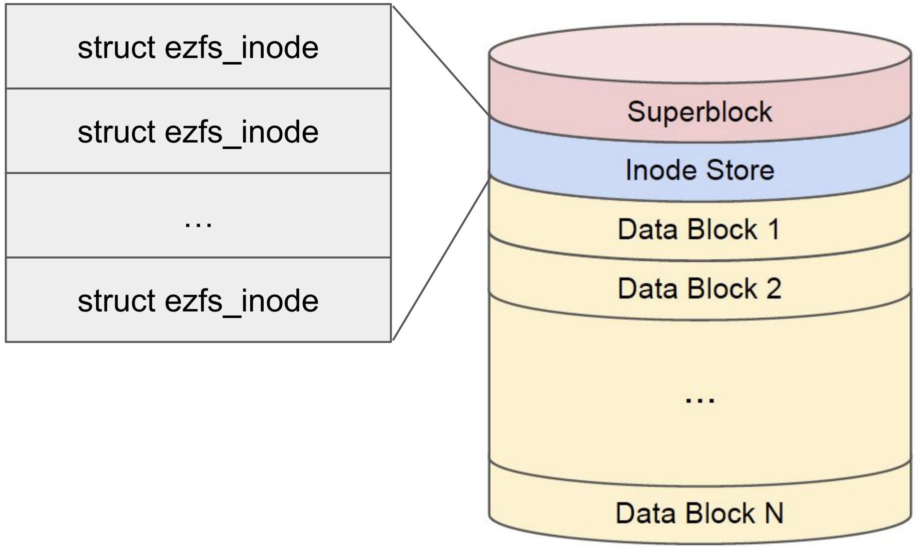
- The inode store is filled with contiguous ezfs_inodes. The maximum possible number of inodes is crammed into the inode store:

```
#define EZFS_MAX_INODES (EZFS_BLOCK_SIZE / sizeof(struct
ezfs_inode))
```

- ezfs_inode stores metadata about a file or directory (size, access times, mode, etc).

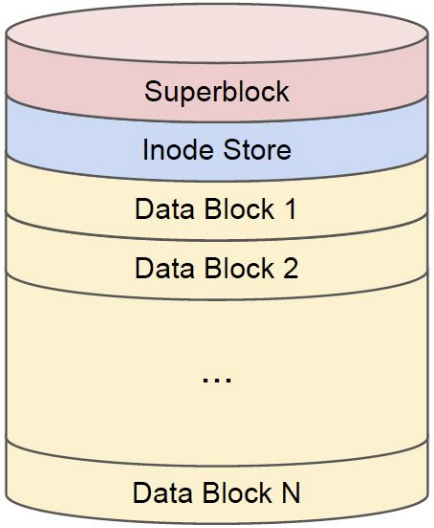
- Each ezfs_inode is associated with data blocks via data_blk_num (the first data block) and nblocks (number of data blocks). The data block, ***not the inode*** stores file or directory contents.

- Each inode is indexed by its position. The 1st inode is inode 1; the 10th inode is inode 10. Note that inodes are indexed from 1, not 0. The reason is that functions in VFS that return inodes return an unsigned int. On error, they return 0.



## The Data Blocks

- Case 1: Inode corresponds to a directory
- Case 2: Inode corresponds to a regular file



## The Data Blocks: Case 1 (directory)

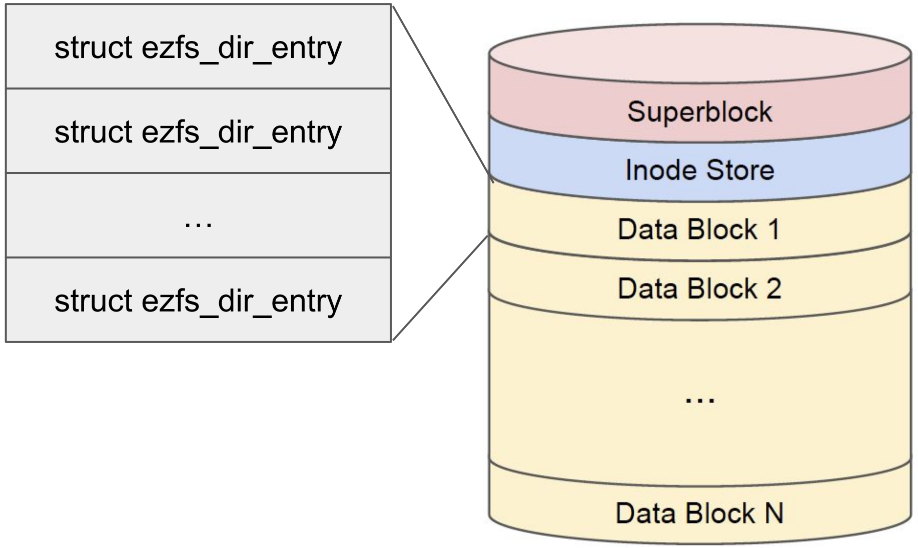
- An inode that corresponds to a directory only has **one** data block.

- The data block is a series of contiguous ezfs_dir_entry's. The maximum number of entries that can be crammed into the data block is indicated by:

```
#define EZFS_MAX_CHILDREN (EZFS_BLOCK_SIZE / sizeof(struct
ezfs_dir_entry))
```

- ezfs_dir_entry maps a filename to a inode number.

- Each ezfs_dir_entry has a flag indicating whether it's active or not. When we unlink a file (rm command), we lazily delete the file by setting flag to 0.



# The Data Blocks: Case 2 (regular file)

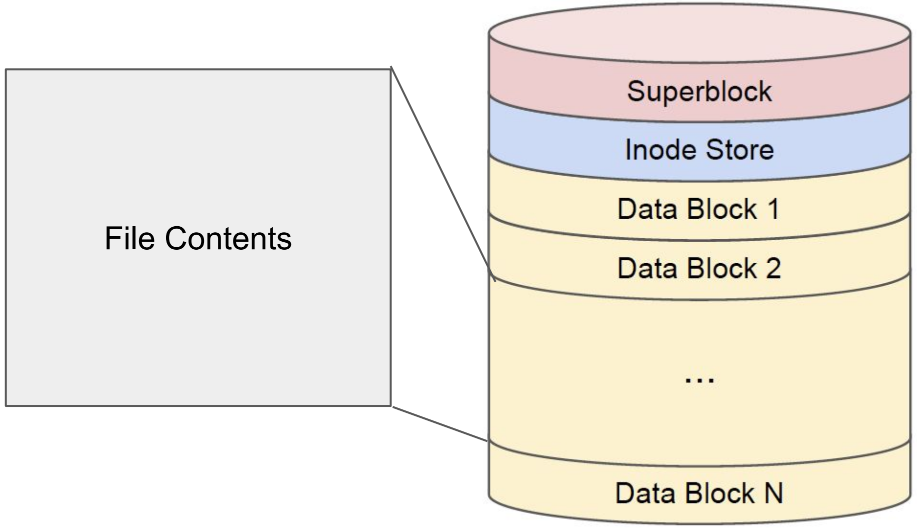
- An inode that corresponds to a regular file can have multiple **contiguous** data blocks

- Each data block is a series of bytes representing file contents.

- The range of data blocks can be calculated by data_blk_num and nblocks.

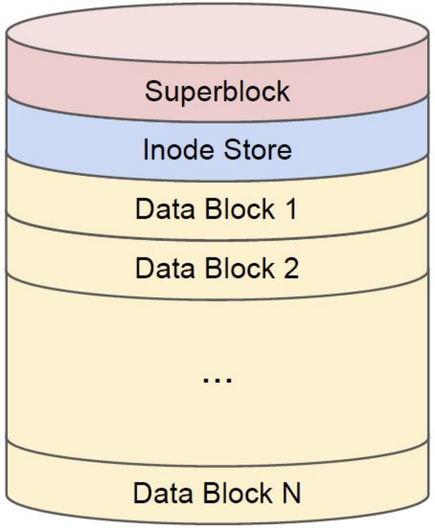
[data_blk_num, data_blk_num + nblocks - 1]

- **Empty regular file should have 0 data blocks.**



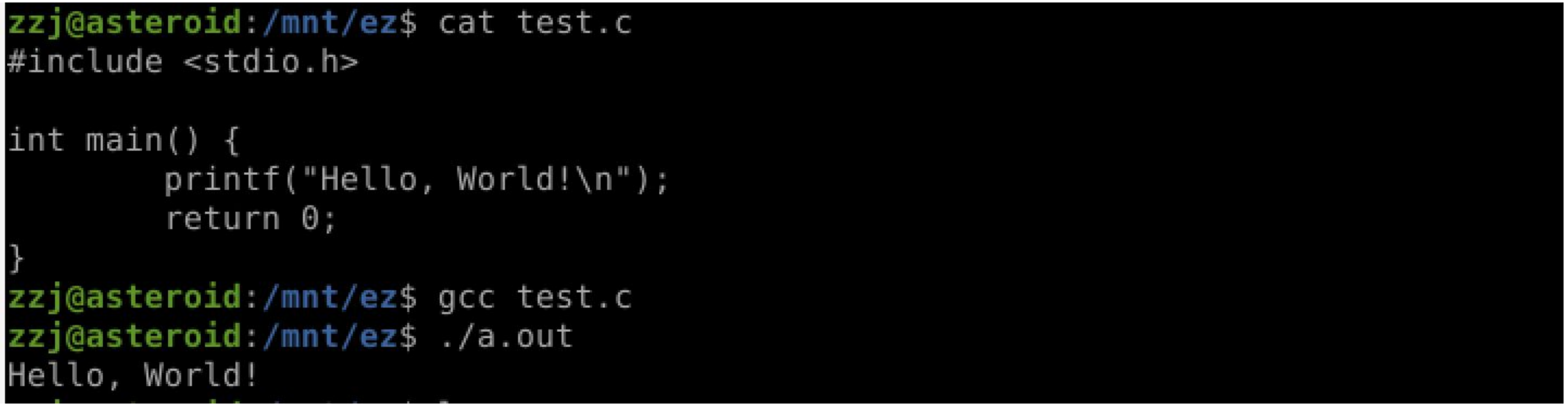
# EZFS Data Block Contiguous Allocation Policy

- As the size of a regular file extends, it might request for more data blocks.
- If the file is empty before the request, assign an empty block with the lowest available data block number to it.
- Otherwise
  - If the data block following the last existing data block of this file is empty, allocate this block to it.
  - Else, in order to maintain the continuum of data blocks, try to move the existing blocks along with the new block to another position with enough space.
  - If there is no such space, return with proper errno.



## EZFS Additional Features

- EZFS supports mmap, and thus executable files.



```
zzj@asteroid:/mnt/ez$ cat test.c
#include <stdio.h>

int main() {
        printf("Hello, World!\n");
        return 0;
}
zzj@asteroid:/mnt/ez$ gcc test.c
zzj@asteroid:/mnt/ez$ ./a.out
Hello, World!
```

## EZFS Limitations

- Although each inode can have multiple contiguous blocks, the size of a EZFS file cannot be more than the total size of all data blocks.
- A directory only has one data block, thus it can only have EZFS_MAX_CHILDREN children.

```
#define EZFS_MAX_CHILDREN (EZFS_BLOCK_SIZE / sizeof(struct ezfs_dir_entry))
```

- The number of inodes is limited by the macro EZFS_MAX_INODES. Therefore, at any given time:

# files + # directories ≤ EZFS_MAX_INODES

recall that EZFS_MAX_INODES is defined as:

```
#define EZFS_MAX_INODES (EZFS_BLOCK_SIZE / sizeof(struct ezfs_inode))
```

## Reference

- https://cs4118.github.io/pantryfs/pantryfs-spec.pdf, Mitchell Gouzenko