File System Data Structures Pt. 2 - ext

CSC 712 – Data Structures

Goals

- Last lecture we learned about data structures used in the FAT file system.
- But FAT is old and somewhat inefficient
- Let’s learn about a modern file system, ext!

File Systems

- *FAT*
- NTFS
- HFS
- *ext (2, 3, 4)*
- Btrfs
- …
- ReiserFS
- XFS
- JFS
- Iso9660
- NFS
- …

Problems with FAT

- In order to “seek” within a file, we must walk the whole FAT chain for that file.
- In order to find free blocks, we must scan the whole FAT

Improvements in ext

- In order to “seek” within a file, we must walk the whole FAT chain for that file.
  - Ext uses indexing! Index nodes (“inodes”) contain many pointers to file chunks
- In order to find free blocks, we must scan the whole FAT
  - Ext uses bitmaps to indicate block usage, which is much more efficient to search.

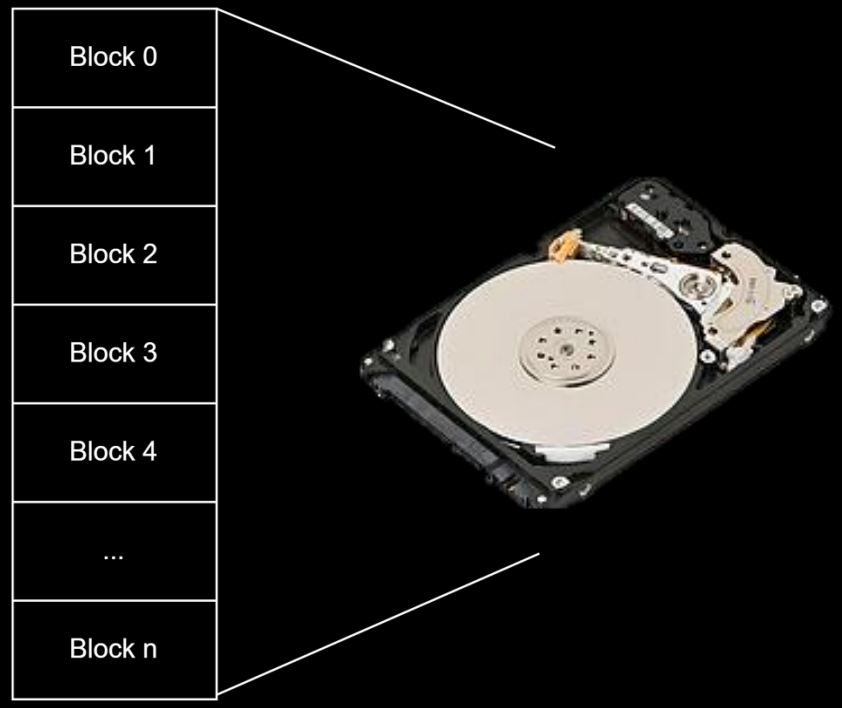
Disclaimer

- Again, I'll show you a rough approximation of how ext works
  - Conceptual ideas, not precise technical details
  - Ext has MAAAANY more features than this

Blocks

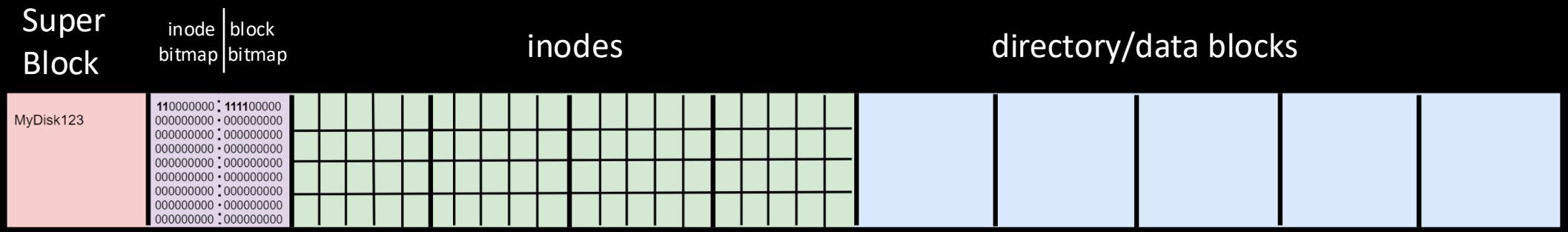
- Recall that disks are accessed by block
- Common block sizes
  - 512, 1024 (1K), 4096 (4K)

- Ext usually uses bs=4096
- We will still use bs=512 for simplicity :)



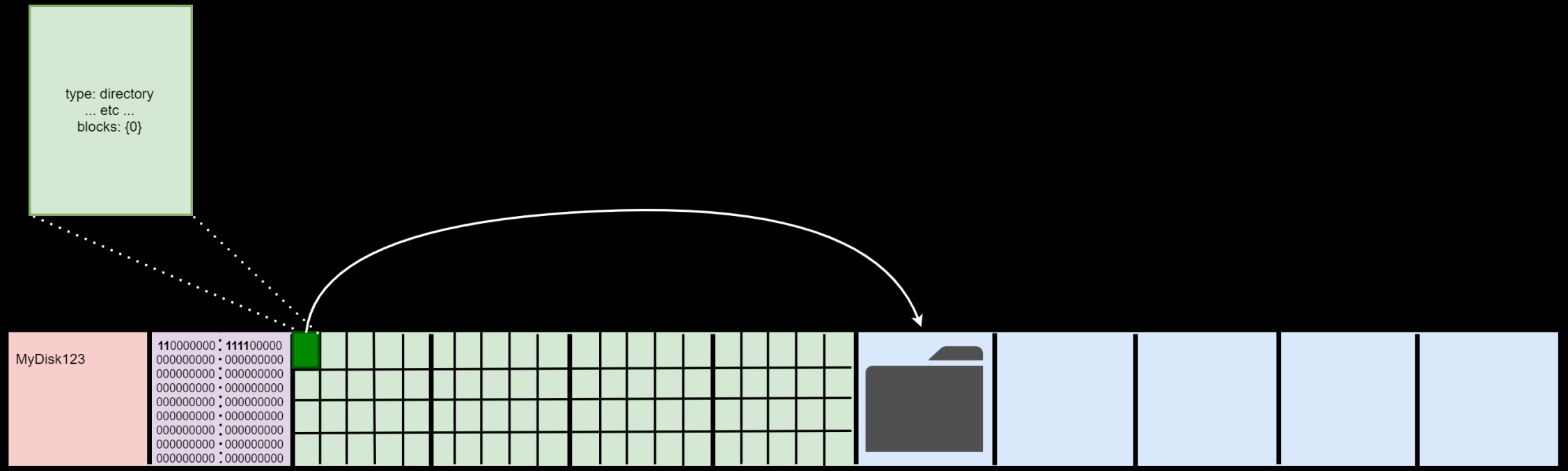
High Level ext Structure

- Superblock
  - Metadata about the disk. Label, telemetry, etc
- Bitmap block(s)
  - inode bitmap / block bitmap, showing free or used
- Inode block(s)
  - Each inode represents one file or directory, and holds its metadata
- Data blocks
  - Actual file data or directory content list



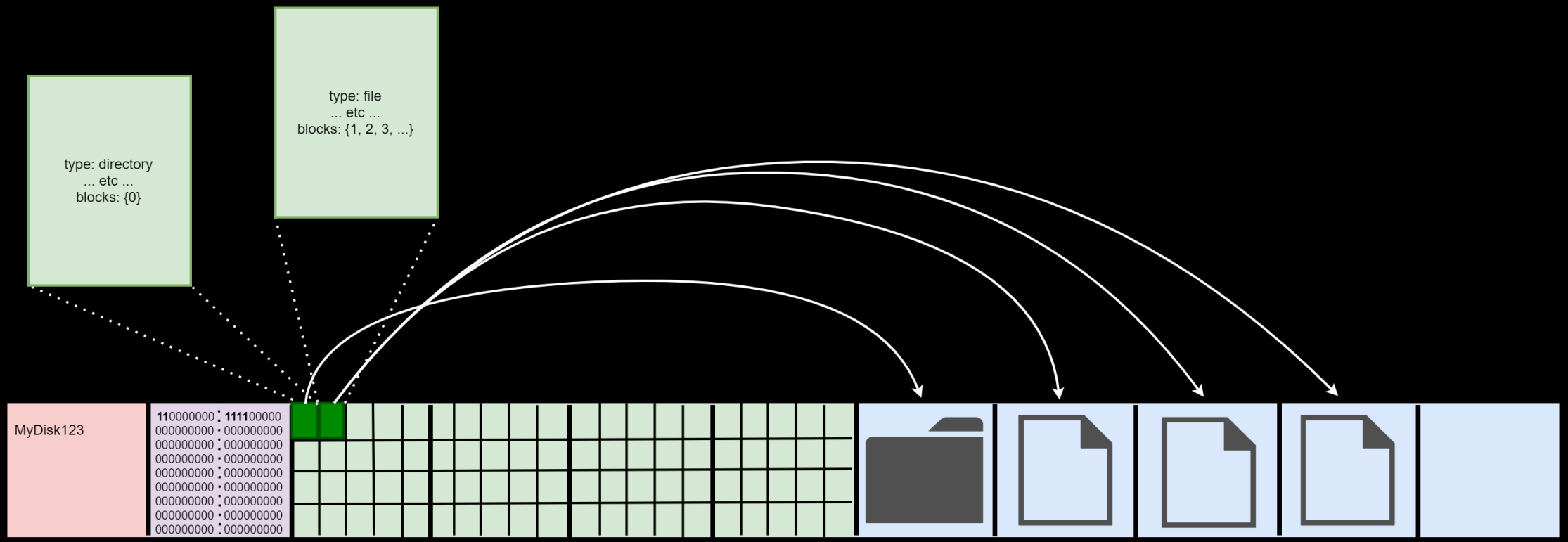
Inodes

- inodes represent files/directories
- Also contain meta-data about that file/directory



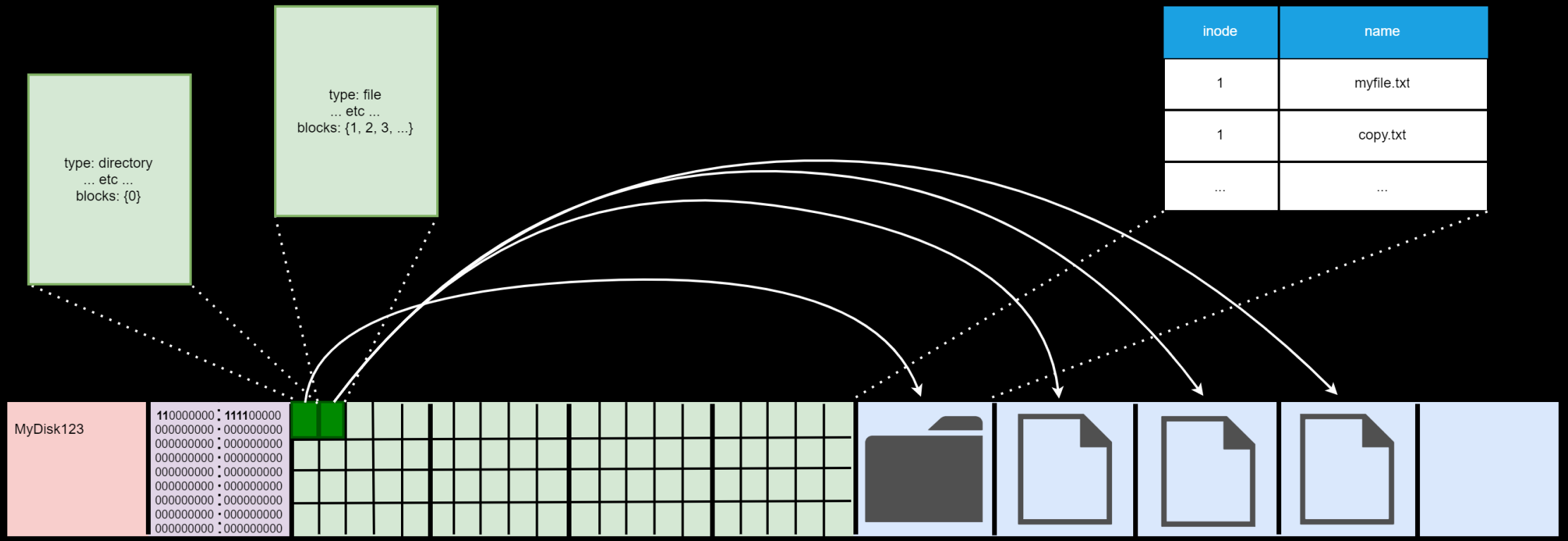
Files

- May span one or more blocks



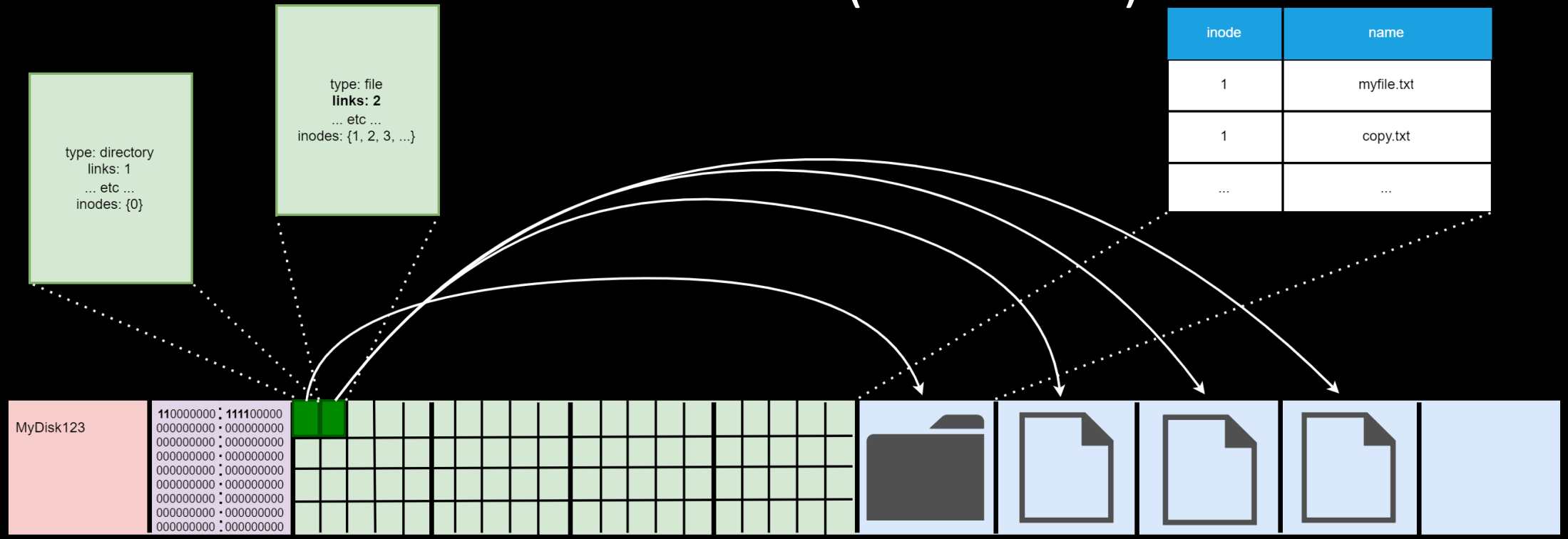
Directory Entries

- Directories hold inode:name pairs



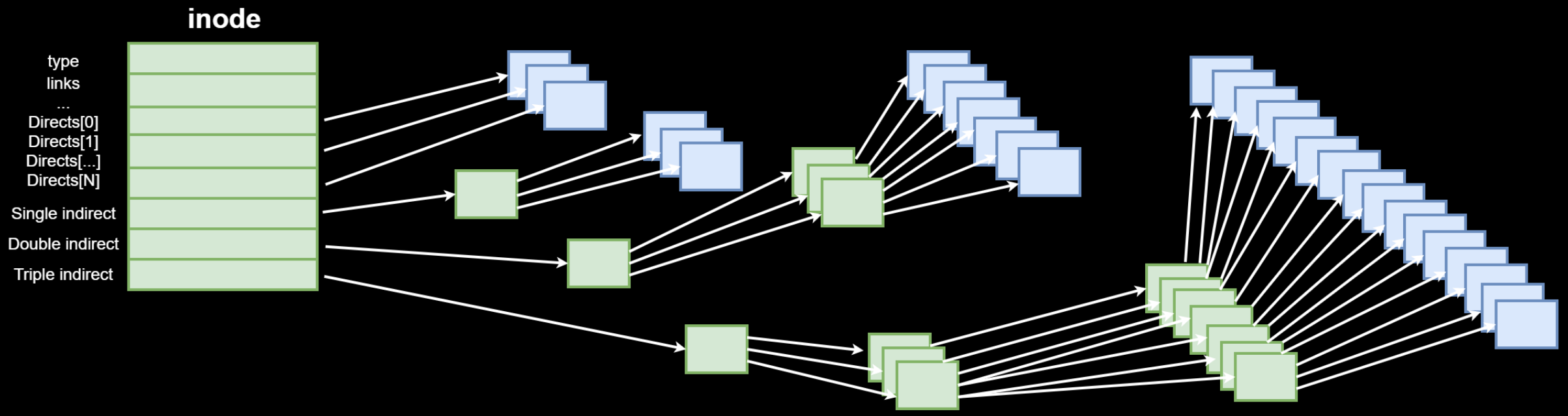
Hard Links

- Many directory listings may point to the same inode! :)
  - Need to track number of links (references)



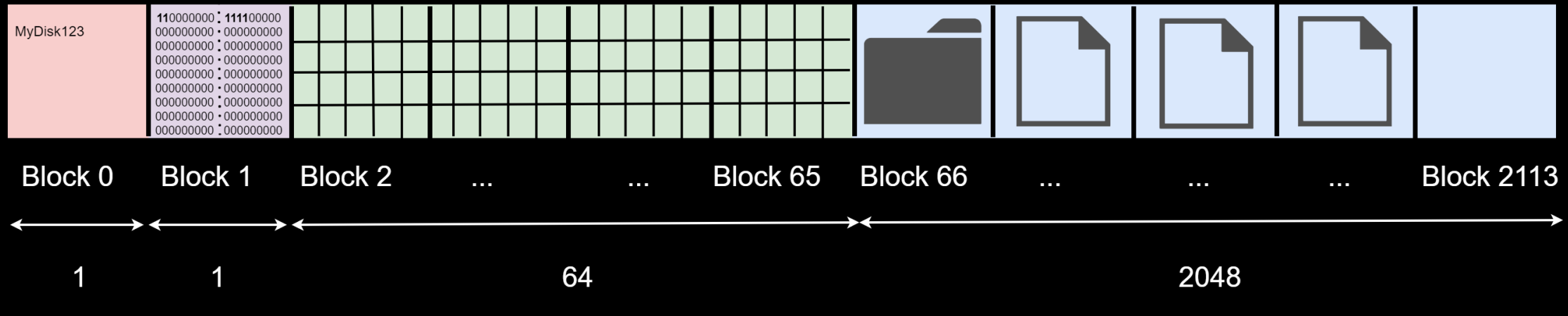
Block Indirection

- Files might be huge! How do we point to 1TB of blocks using only one inode?
  - directs = KB, single = MB, double = GB, triple = TB



Our Ext implementation (bs=512)

- Block 0: superblock
- Block 1: bitmaps 512 * 8 = 4096 (2048 inodes & 2048 data)
- Blocks 2-65: inodes size = 16 (16 * 2048 = 32768, 32768 / 512 = 64)
- Blocks 66-2113: data



Our Inodes

- Type: 2 bytes
- Links: 2 bytes
- Size: 4 bytes
- Directs[3]: 3 * 2 bytes = 6 bytes
- Indirects: 2 bytes

- Total size = 2 + 2 + 4 + 6 + 2 = 16 bytes

Our Directory Entries

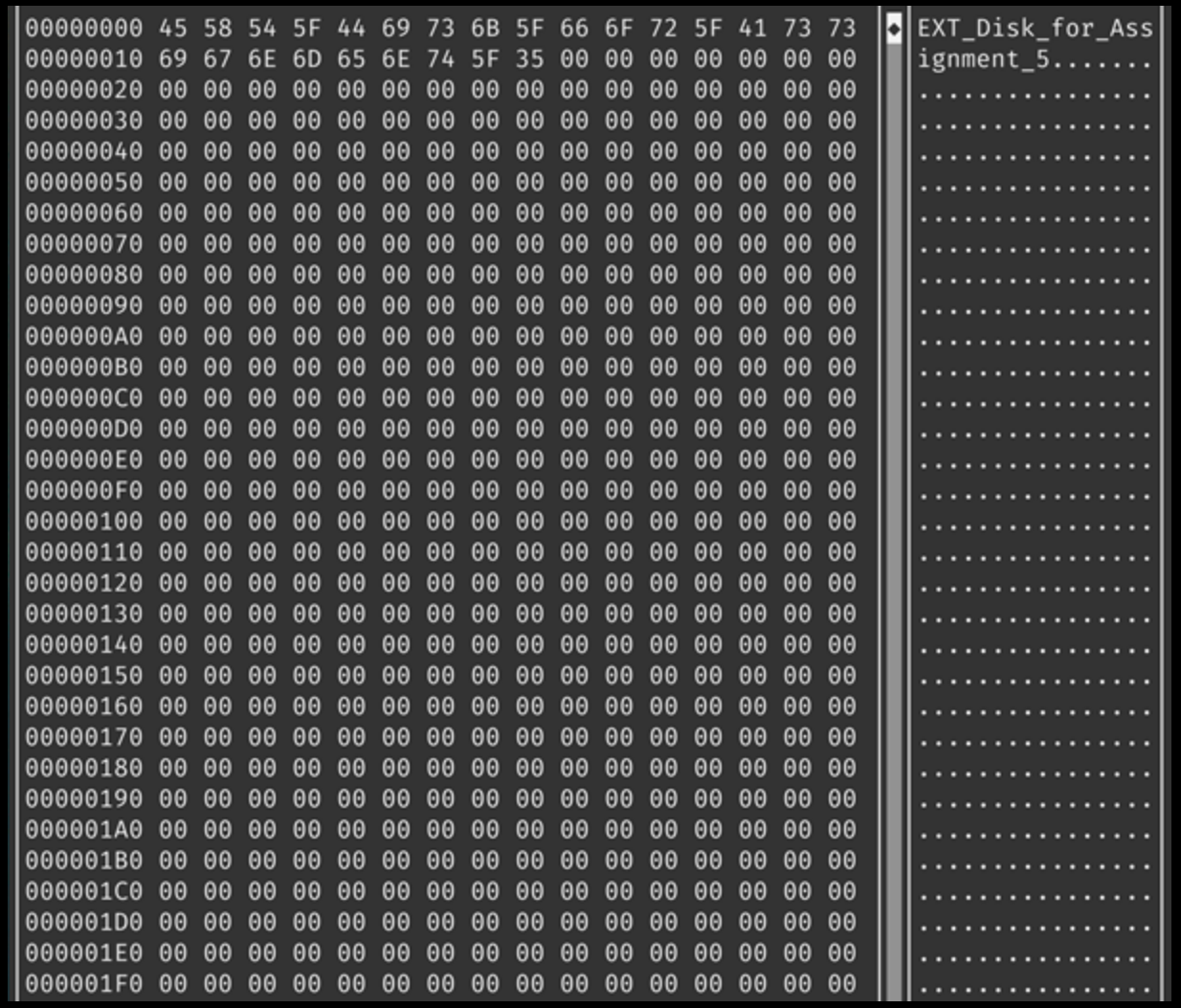
- inode: 2 bytes
- Name: 30 bytes

- Total size = 32

- 512 / 32 = 16 directory listings per block

Block 0: Superblock

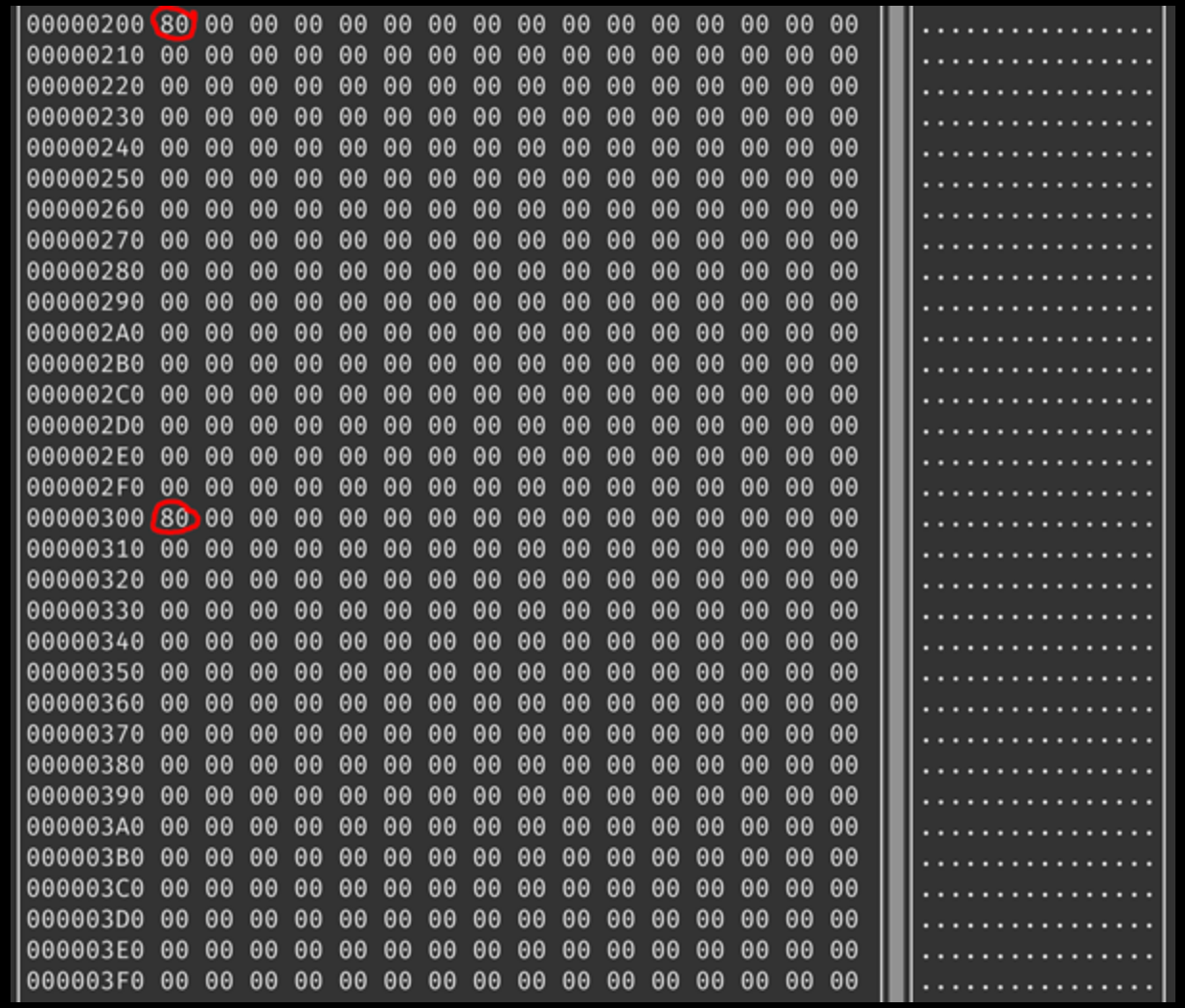
- Just contains the disk label
- Null terminated



Block 1: Bitmaps

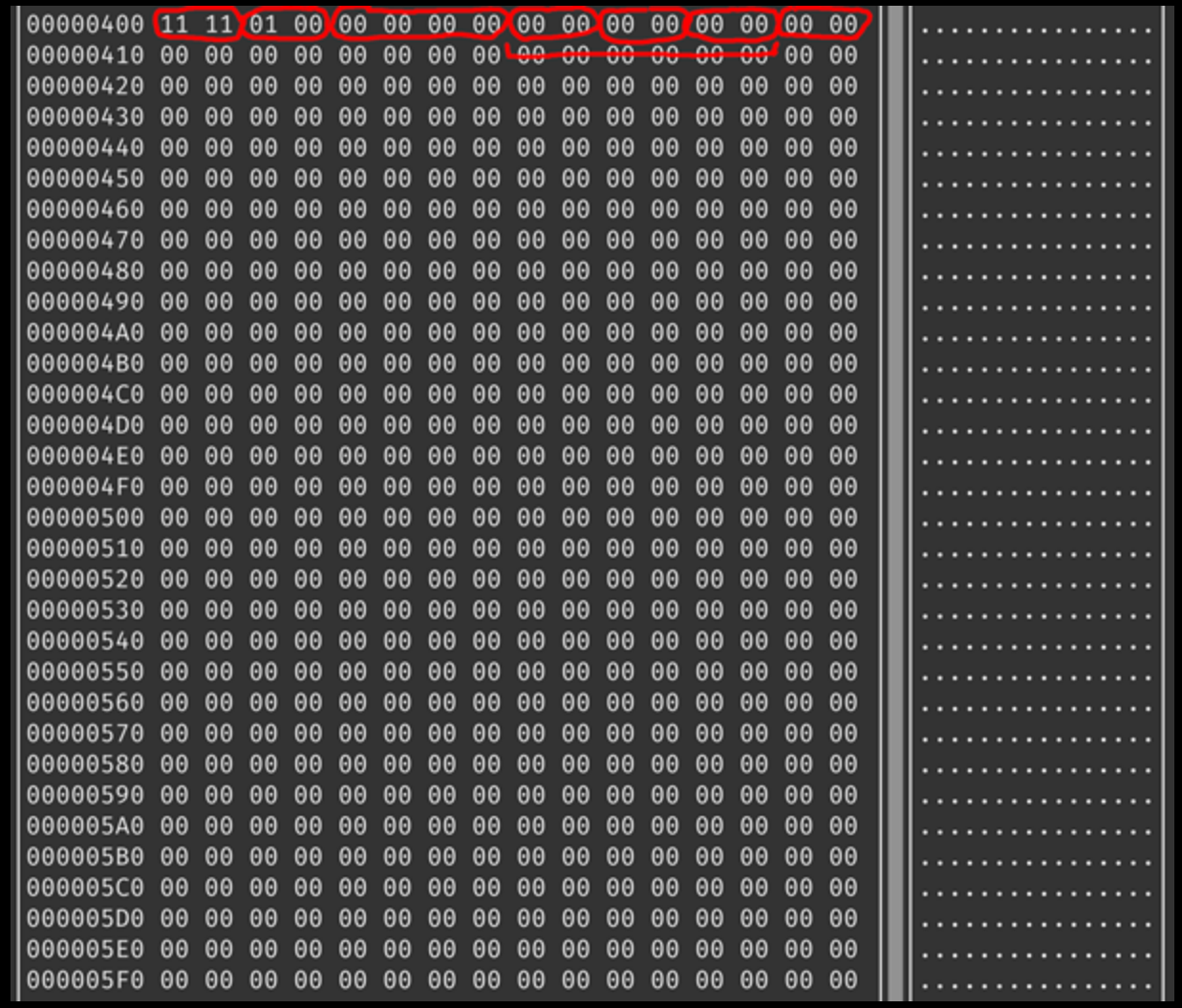
- 256 bytes (2048 bits) for inodes
- 256 bytes (2048 bits) for data

- 0x80 = 1 0 0 0 0 0 0 0 ...
  - Root directory entry



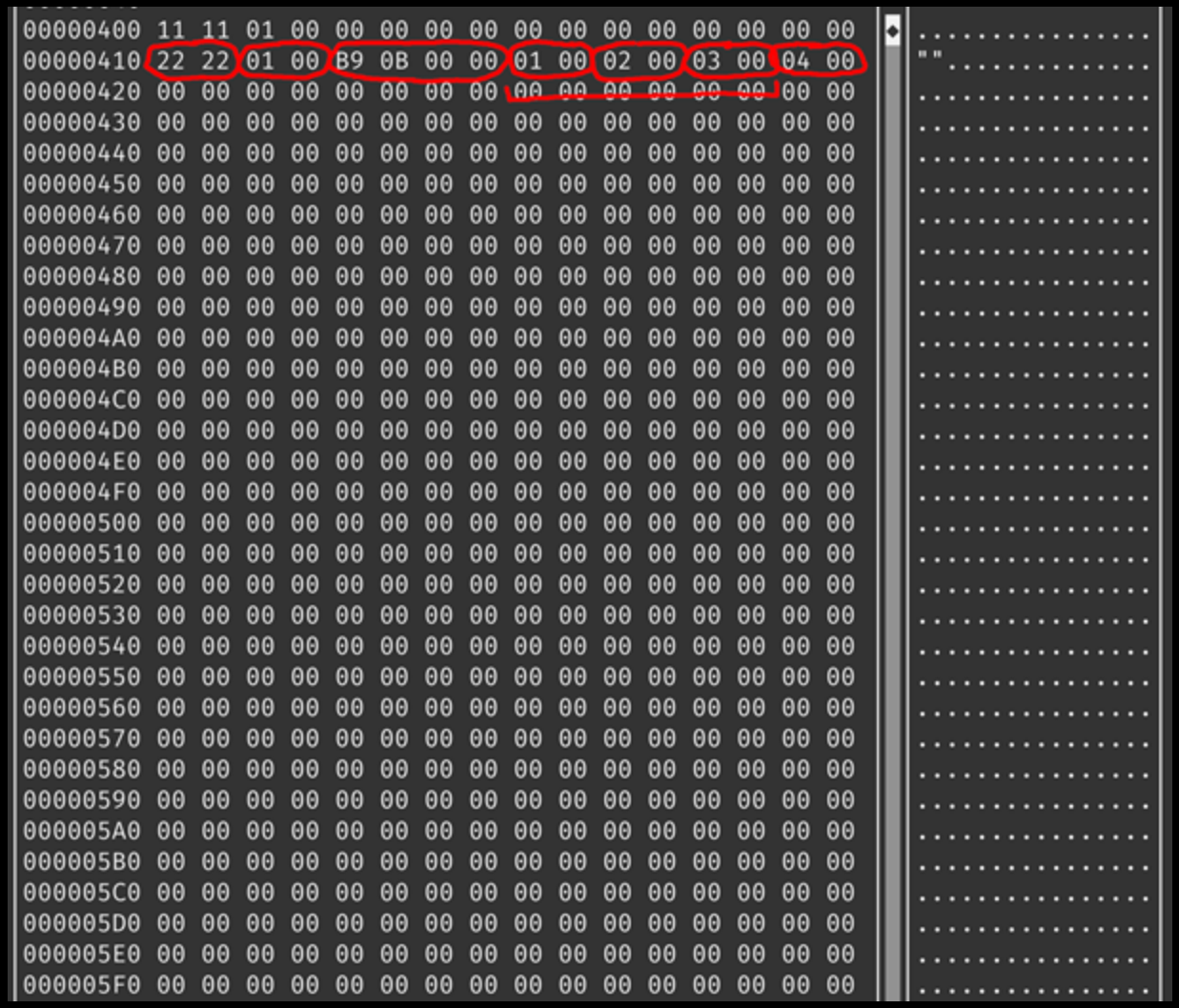
Blocks 2-65 (64 total): inodes

- 0x1111 = TYPE_DIR
- 0x0001 = links
- 0x00000000 = size
- Directs[3]:
  - 0x0000: data 0 (block 66)
  - Unused
  - Unused
- Indirects: unused



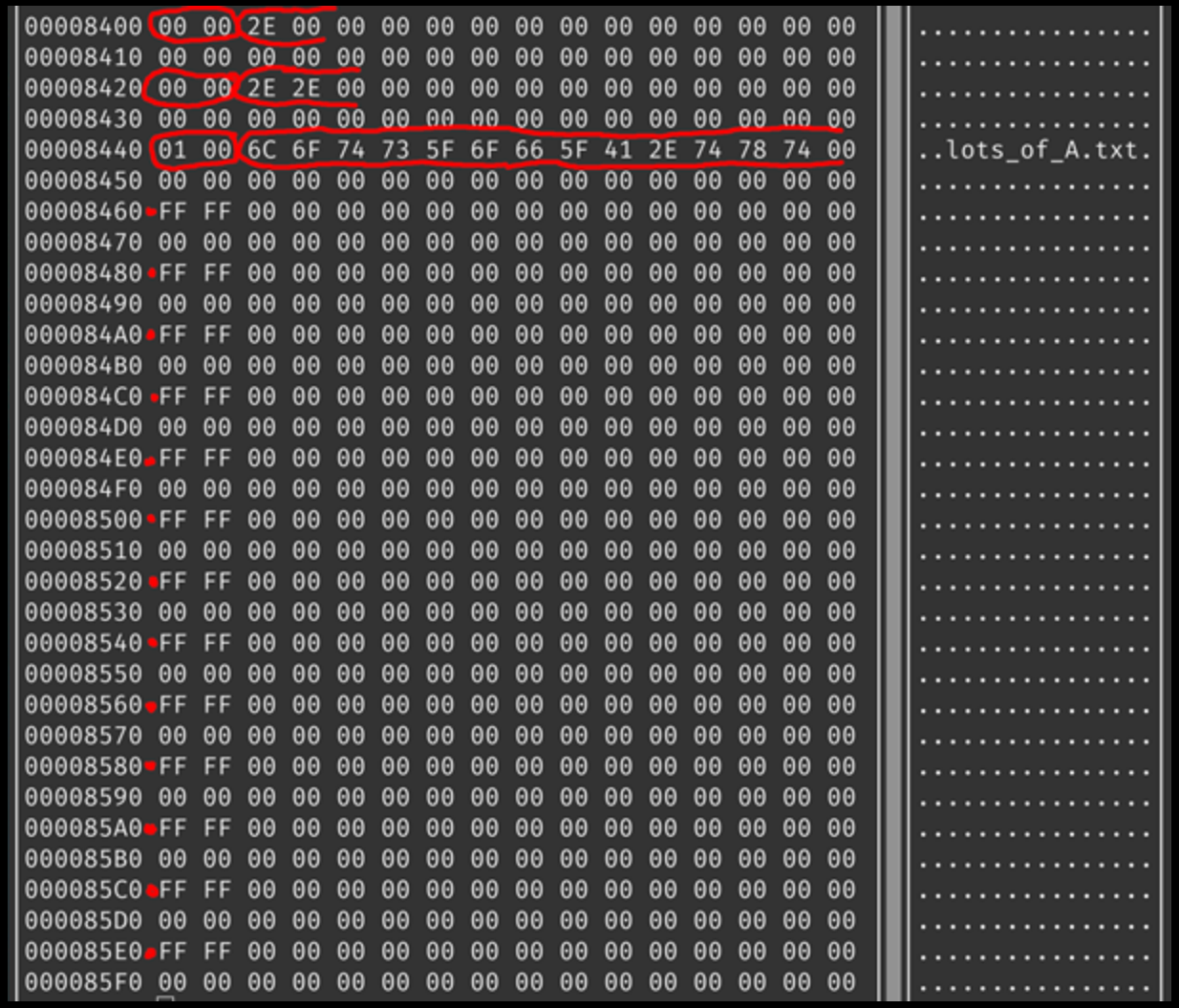
Blocks 2-65 (64 total): inodes

- 0x2222 = TYPE_FILE
- 0x0001 = links
- 0x00000BB9 = size (3001)
- Directs[3]:
  - 0x0001: data 1 (block 67)
  - 0x0002: data 2 (block 68)
  - 0x0003: data 3 (block 69)
- Indirects: 0x0004
  - Data 4 (block 70) contains an array of 2-byte datablock numbers



Blocks 66-2113 (2048 total): data

- Entry 1
  - 0x0000: inode 0
  - 0x2e 0x00: “.”
- Entry 2
  - 0x0000: inode 0
  - 0x2e 0x2e 0x00: “..”
- Entry 3
  - 0x0001: inode 1
  - 0x67 0x6f … : “lots_of_A.txt”
- Entry 4 - 15
  - 0xFFFF: unused



Further Reading

http://web.mit.edu/tytso/www/linux/ext2intro.html

https://www.kernel.org/doc/html/latest/filesystems/ext4/index.html

Live Demo