Persistence

Abstractions
Resource management
(isolation; efficiency)

Virtualization (CPU, memory)
Concurrency

Interaction with Devices

Disks and Persistence

- Store data users care about: persist beyond reboots
- Backing store for paging

Space multiplexing
(coexist)

Shared view of
storage!

Permissions

Intelligence in software, mostly

Motivation

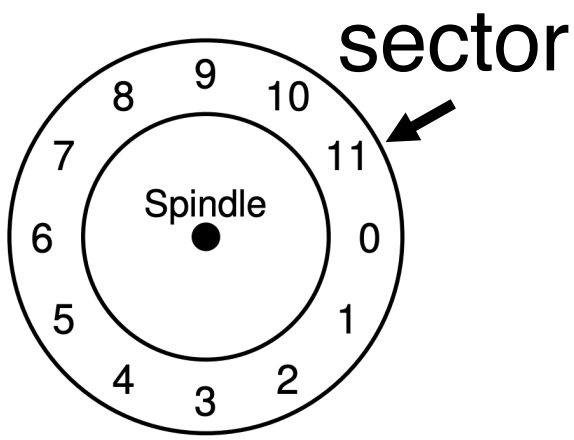
What good is a computer without any I/O devices?

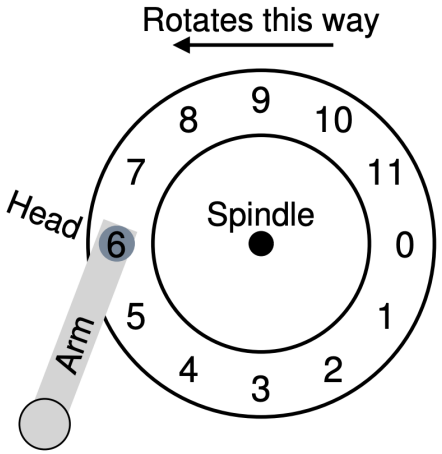
- keyboard, display, disks

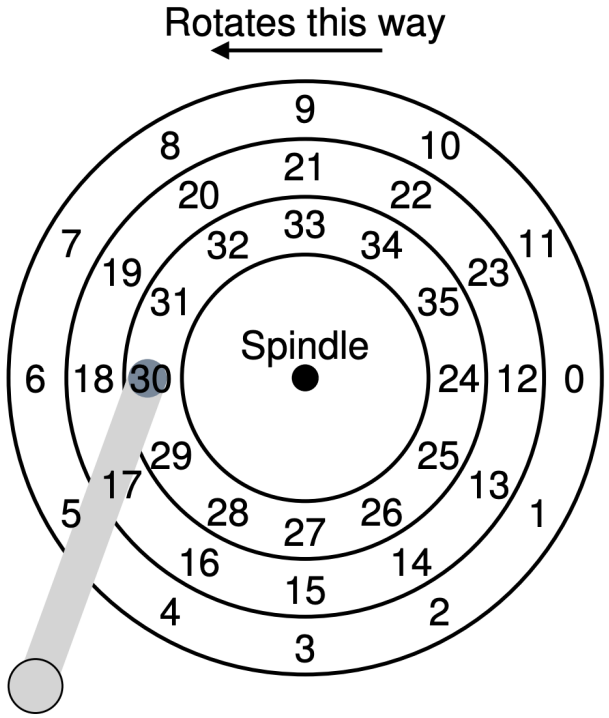
We want:

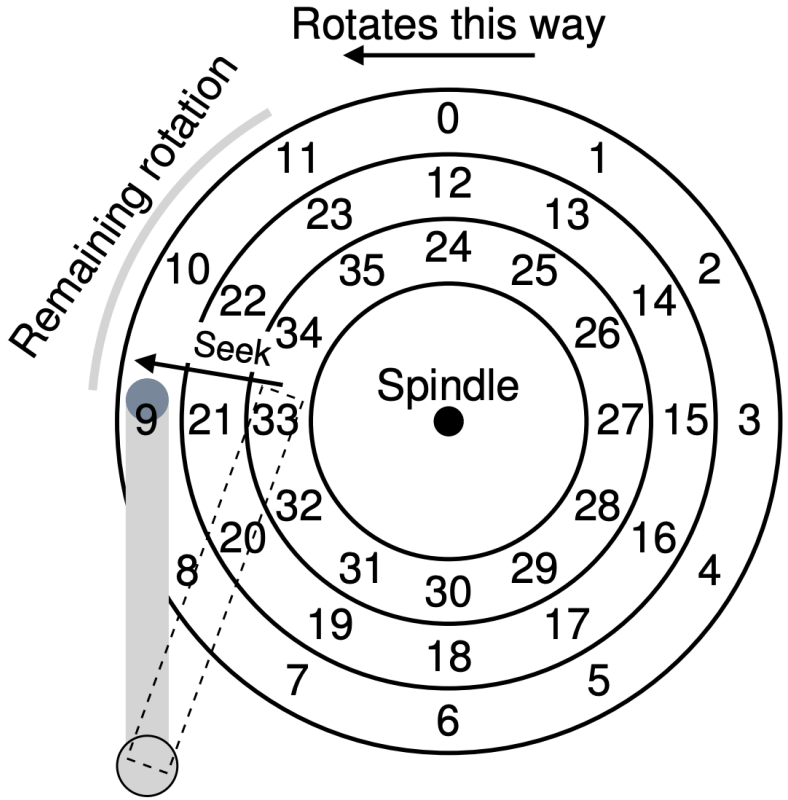
- **H/W** that will let us plug in different devices
- **OS** that can interact with different combinations

Disk


sector
Spindle
0
1
2
3
4
5
6
7
8
9
10
11


Rotates this way
Spindle
Head
Arm
0
1
2
3
4
5
6
7
8
9
10
11


Rotates this way
Spindle
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35


Rotates this way
Remaining rotation
Seek
Spindle
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

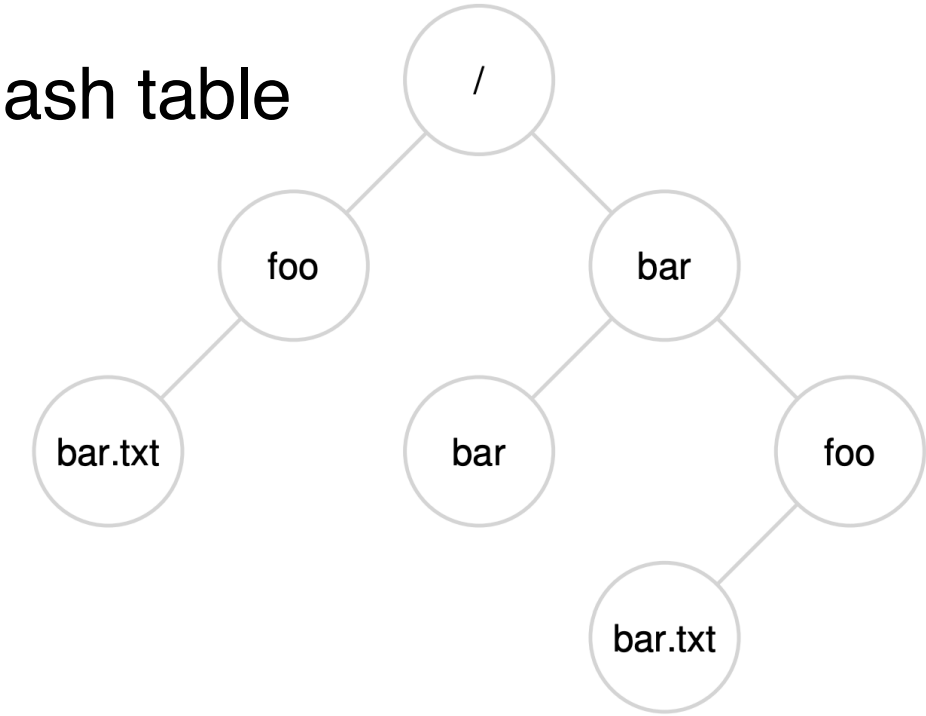
Filesystems

Abstractions over blocks of data

Flat mapping of name to blocks? E.g. hash table

Something a bit more flexible:

A hierarchy




```
   creat(), open(),
   read(), write(),
mkdir(), readdir(), …
   link(), unlink()
```

Filesystems

Why are file systems useful?

- Durability across restarts
- Naming and organization
- Sharing among programs and users

Why interesting?

- Crash recovery
- Performance
- API design for sharing
- Security for sharing
- Abstraction is useful: pipes, devices,
  - /proc, /afs, etc.

Filesystems

API example -- UNIX/Posix/Linux/xv6/&c:

- fd = open("x/y", -);
- write(fd, "abc", 3);
- link("x/y", "x/z");
- unlink("x/y");

- Plan 9 OS (Bell labs) - Attempts to structure entire OS as a filesystem

- http://plan9.bell-labs.com/plan9/

Questions for filesystems

- What **on-disk structures** to represent files and directories?
  - Contiguous, Extents, Linked, FAT, Indexed, Multi-level indexed
  - Which are good for different **metrics**?
- What disk **operations** are needed for:
- make directory
- open file
- write/read file
- close file

FS Implementation

1. On-disk structures
    - how does file system represent files, directories?

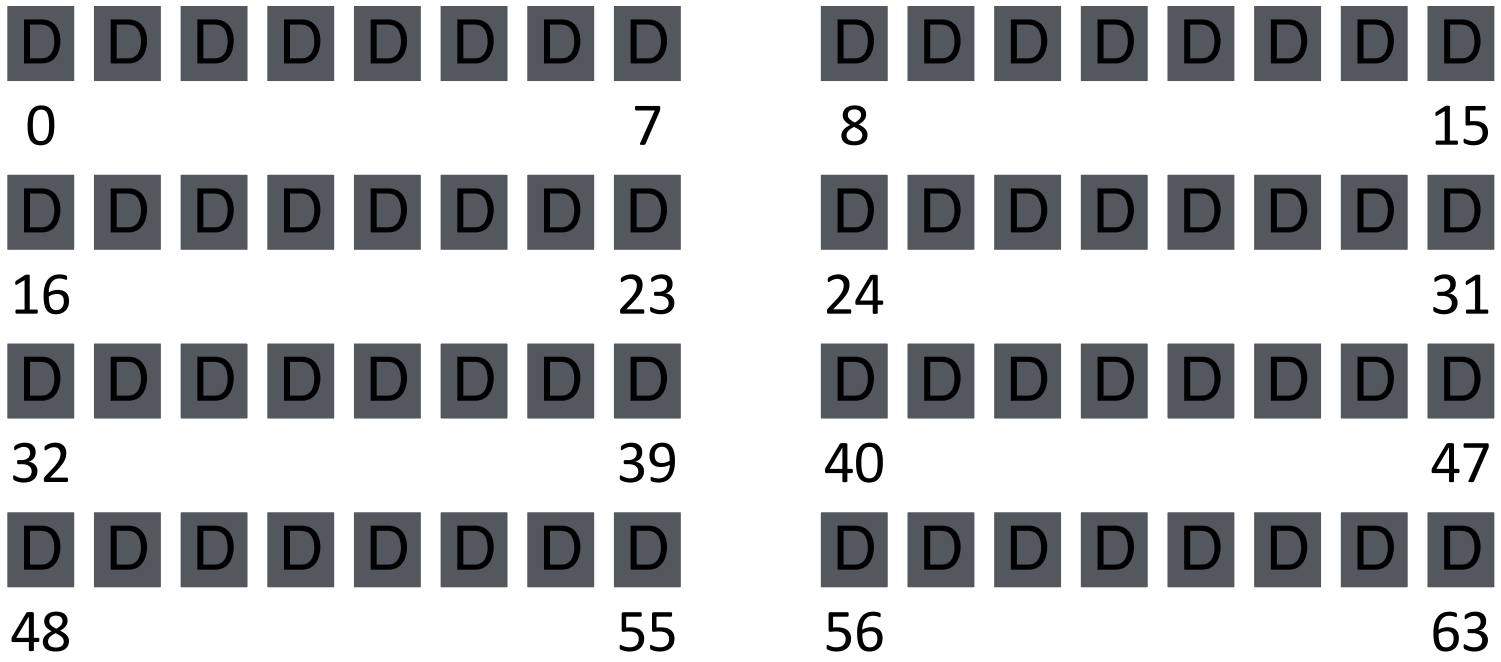
2. Access methods
    - what steps must reads/writes take?

Part 1: Disk Structures

Persistent Store

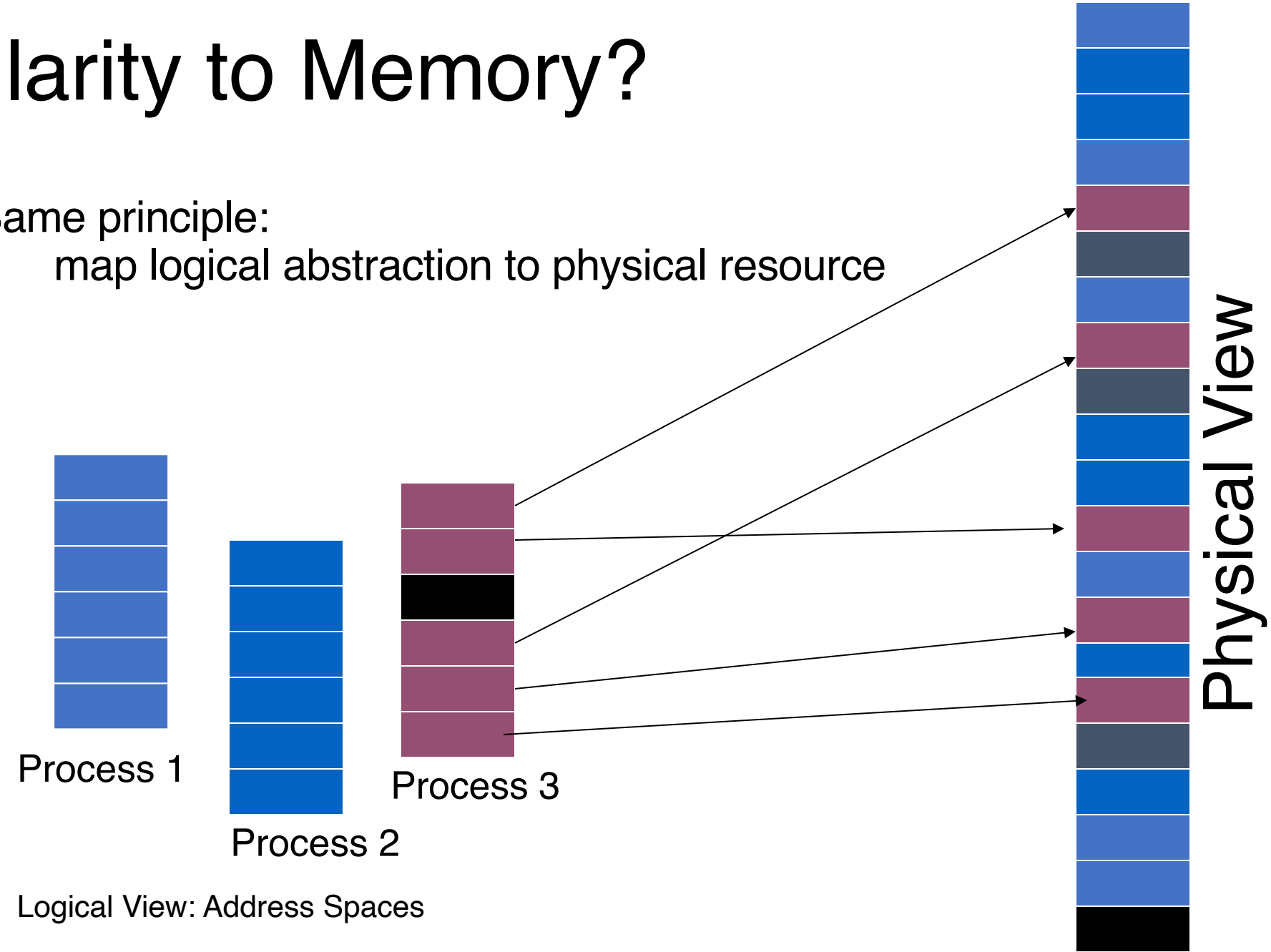
Given: large array of blocks on disk

Want: some structure to map files to disk blocks



Similarity to Memory?

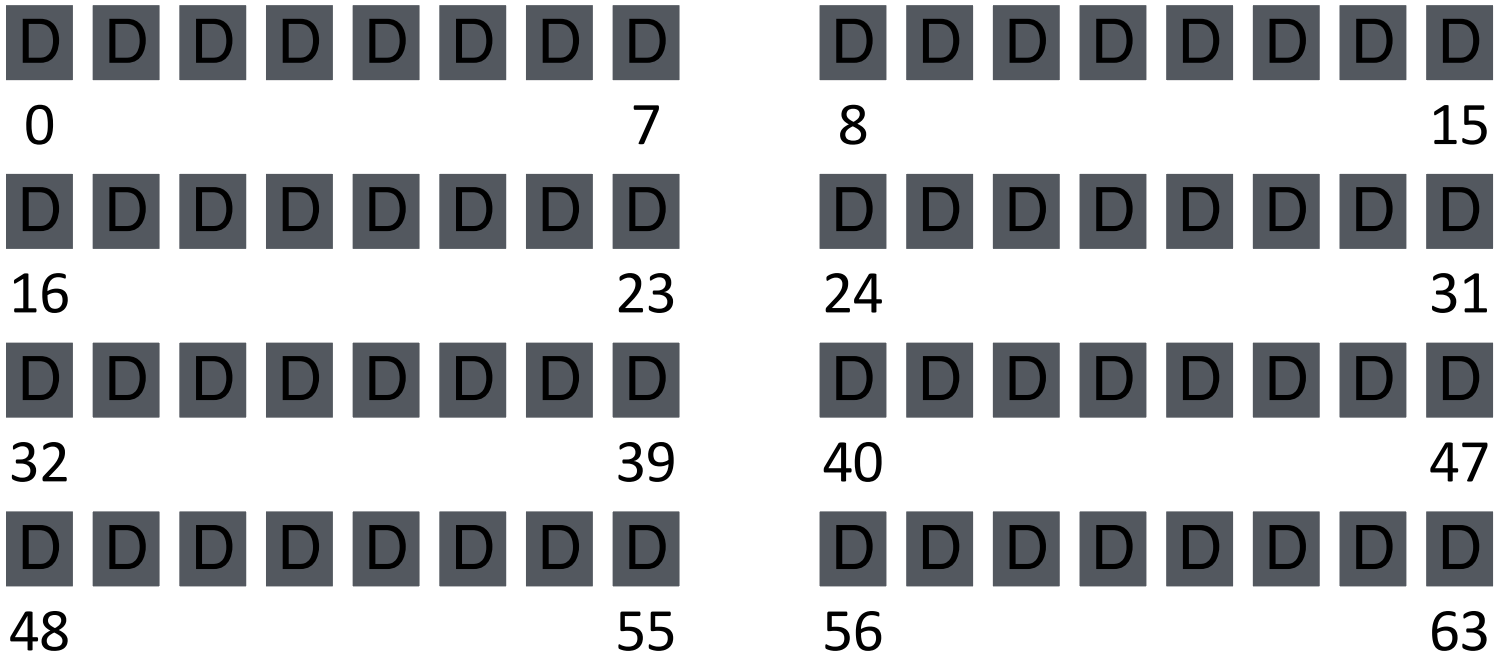
Same principle:
map logical abstraction to physical resource



On-Disk Structures

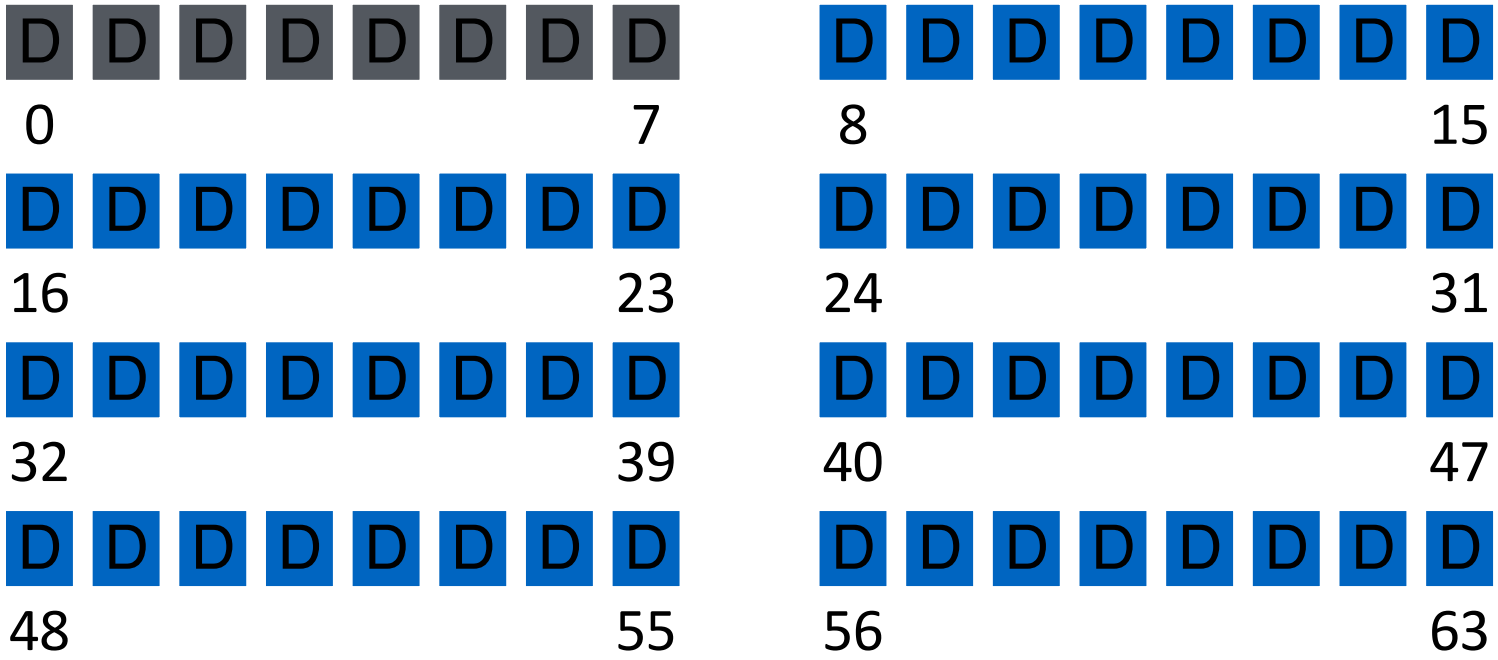
- data block
- inode table
- indirect block
- directories
- data bitmap
- inode bitmap
- superblock

FS Structs: Empty Disk

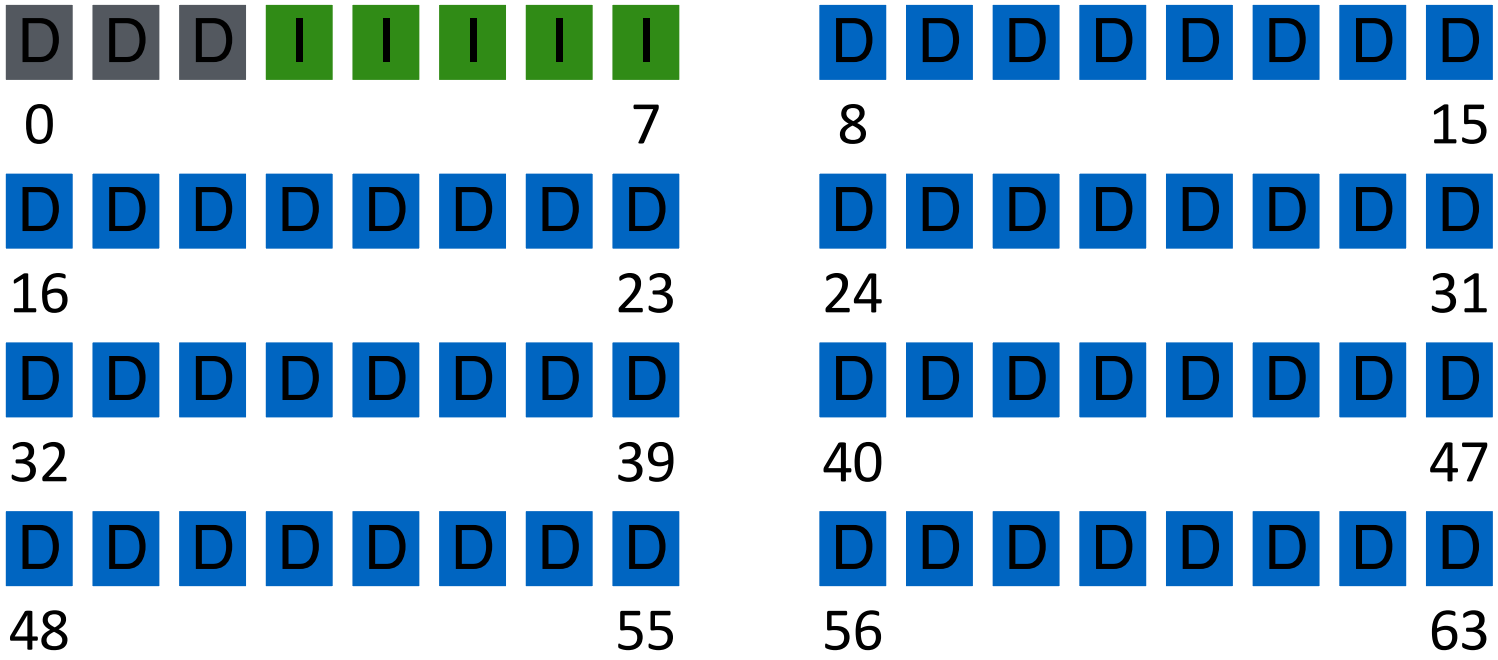


Assume each block is 4KB

Data Blocks


D D D D D D D D
0 7
D D D D D D D D
8 15
D D D D D D D D
16 23
D D D D D D D D
24 31
D D D D D D D D
32 39
D D D D D D D D
40 47
D D D D D D D D
48 55
D D D D D D D D
56 63

Inodes


D D D I I I I I
0 7
D D D D D D D D
8 15
D D D D D D D D
16 23
D D D D D D D D
24 31
D D D D D D D D
32 39
D D D D D D D D
40 47
D D D D D D D D
48 55
D D D D D D D D
56 63

One Inode Block

Each inode is typically 256 bytes (depends on the FS, maybe 128 bytes)

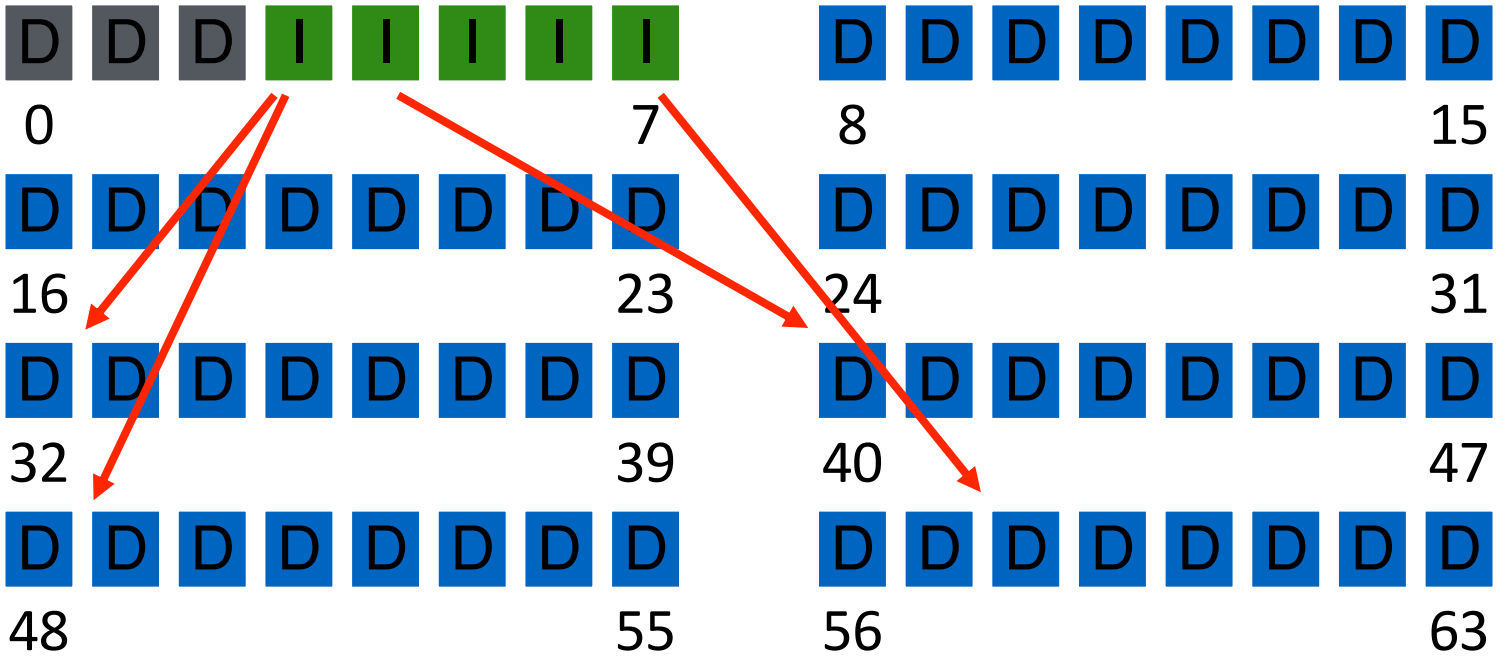
4KB disk block

16 inodes per inode block.

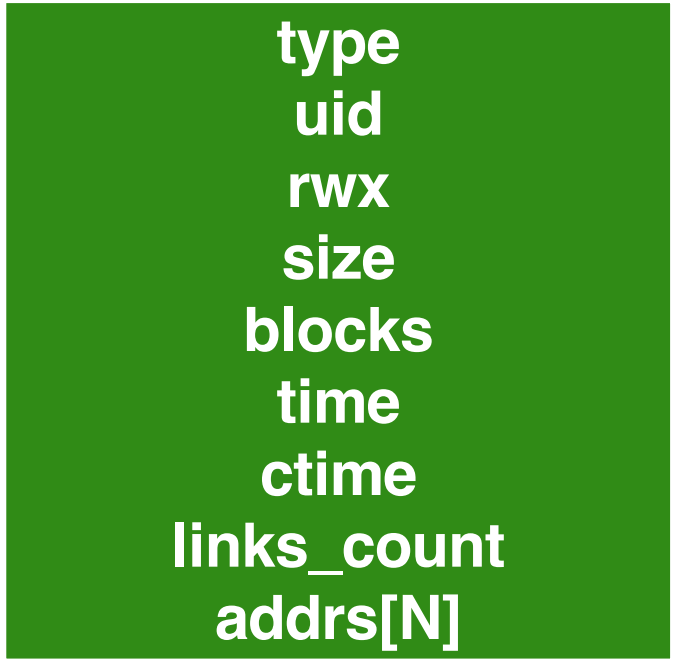
Inode

type (file or dir?)
uid (owner)
rwx (permissions)
size (in bytes)
Blocks
time (access)
ctime (create)
links_count (# paths)
addrs[N] (N data blocks)

Inodes


D D D I I I I I D D D D D D D D
0 7 8 15
D D D D D D D D D D D D D D D D
16 23 24 31
D D D D D D D D D D D D D D D D
32 39 40 47
D D D D D D D D D D D D D D D D
48 55 56 63

Inode

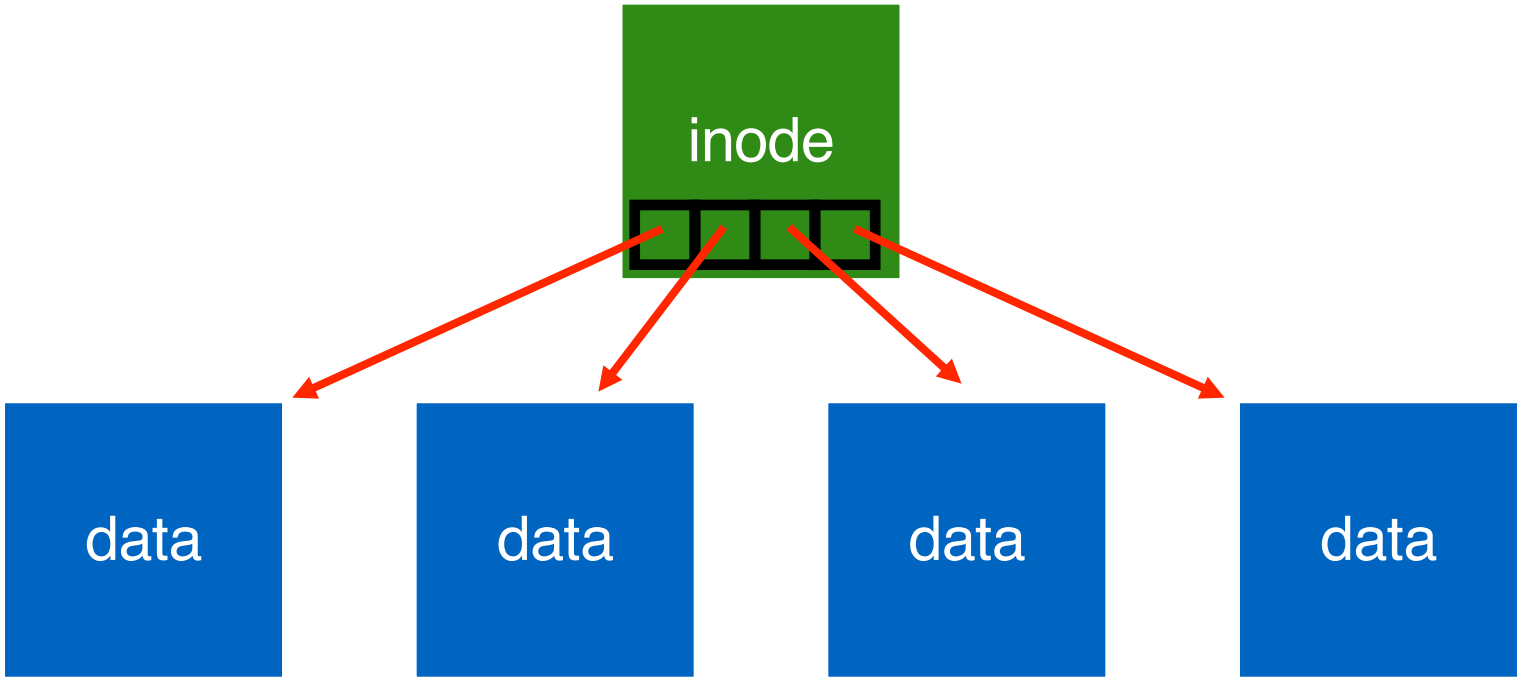


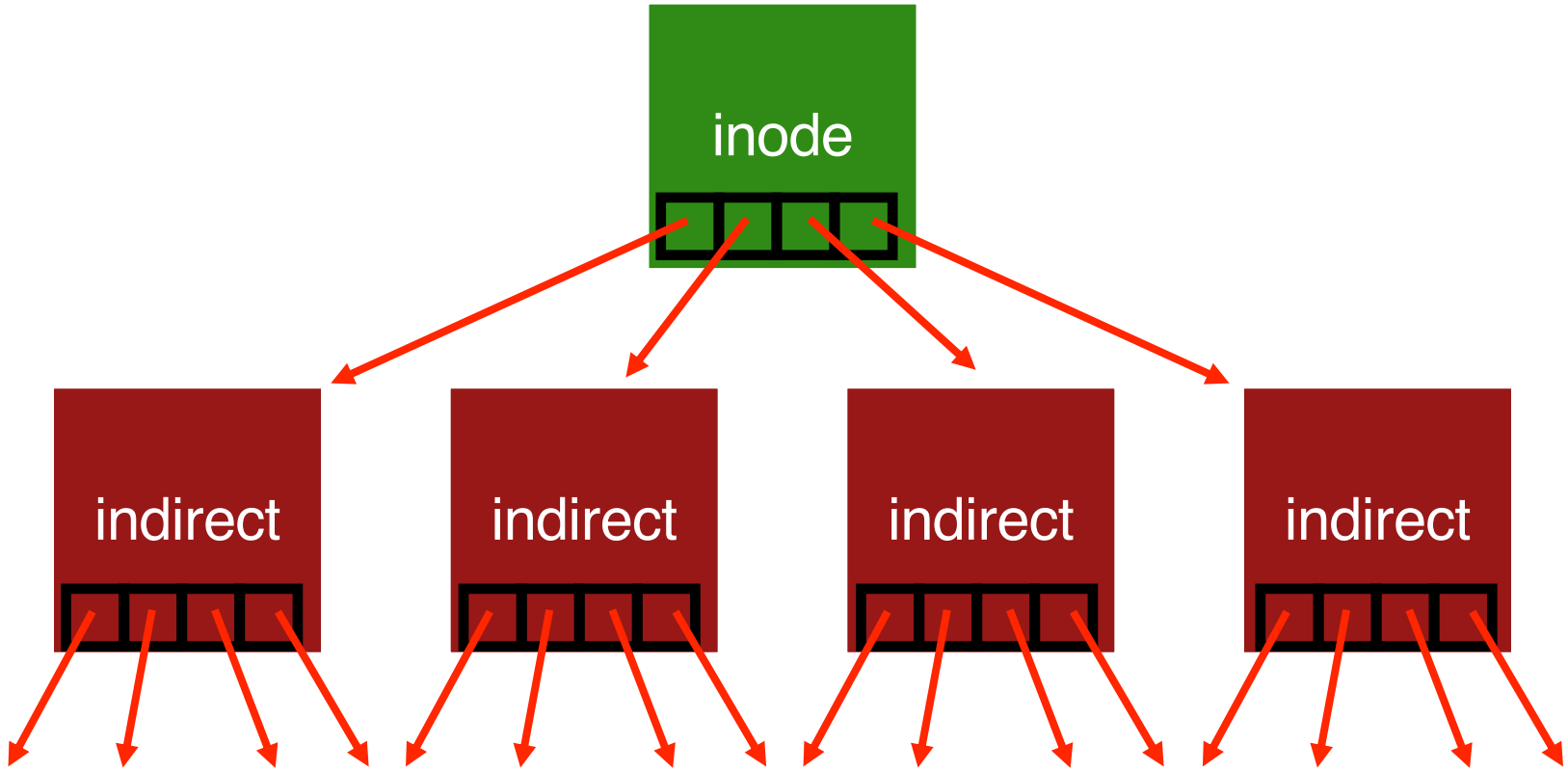
Assume single level (just pointers to data blocks)

What is max file size?
Assume 256-byte inodes (all can be used for pointers)
Assume 4-byte addrs

How to get larger files?

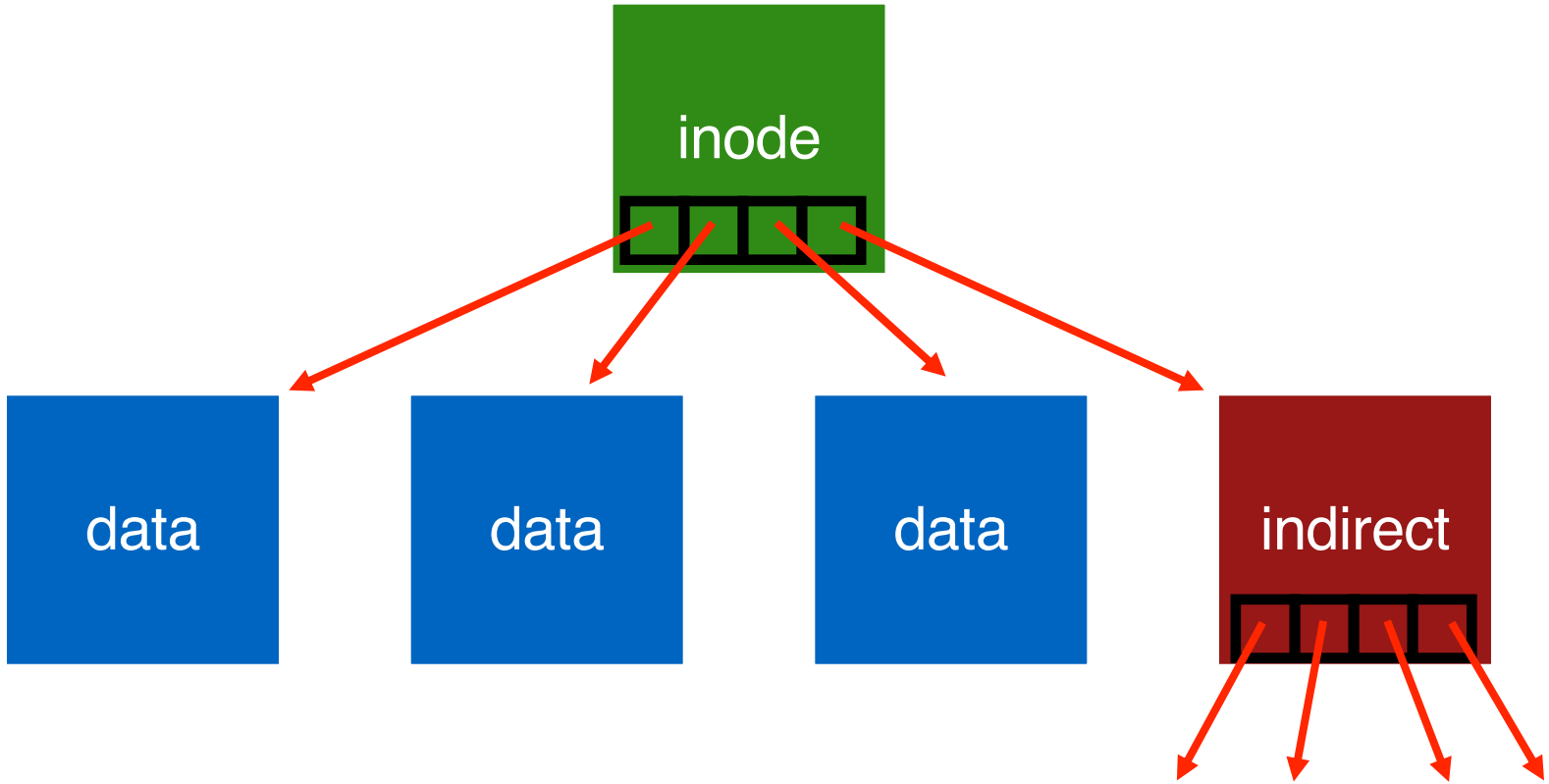
256 / 4 = 64
64 * 4K = 256 KB!


inode
data
data
data
data



Indirect blocks are stored in regular data blocks.

what if we want to optimize for small files?

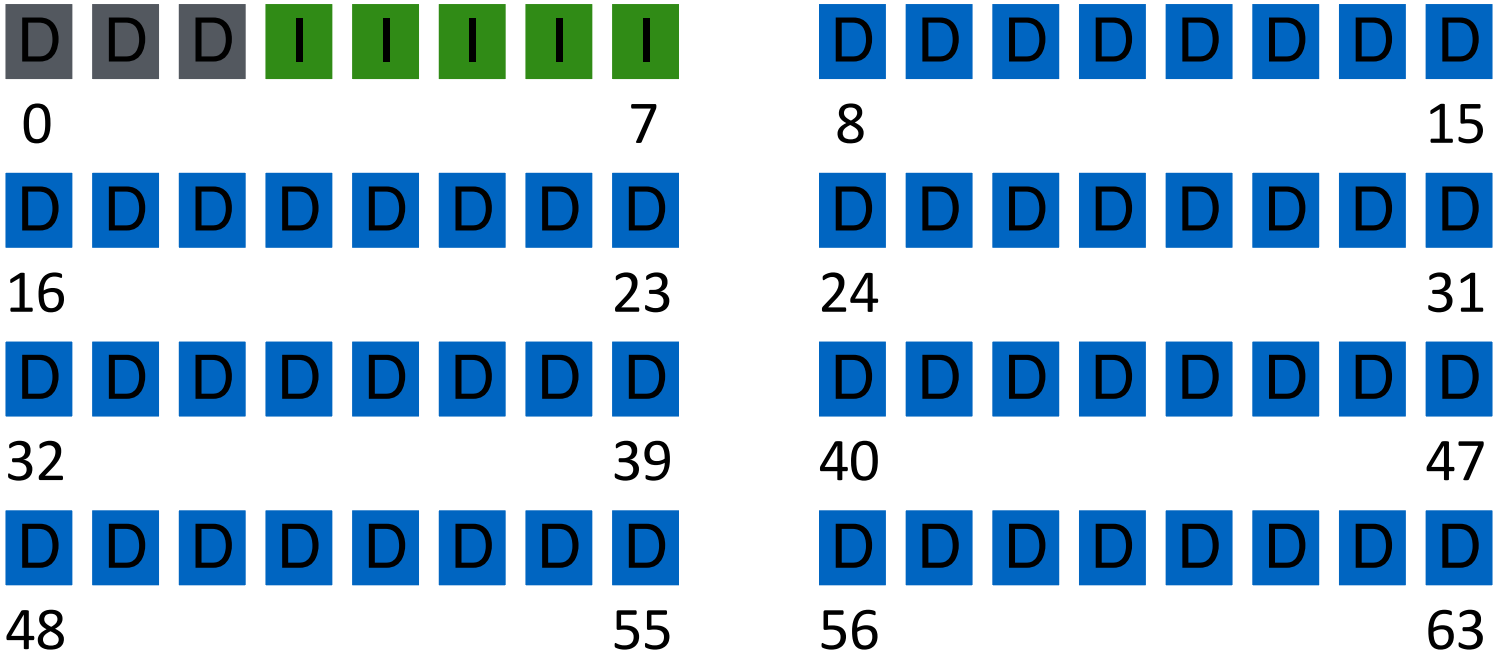


Better for small files

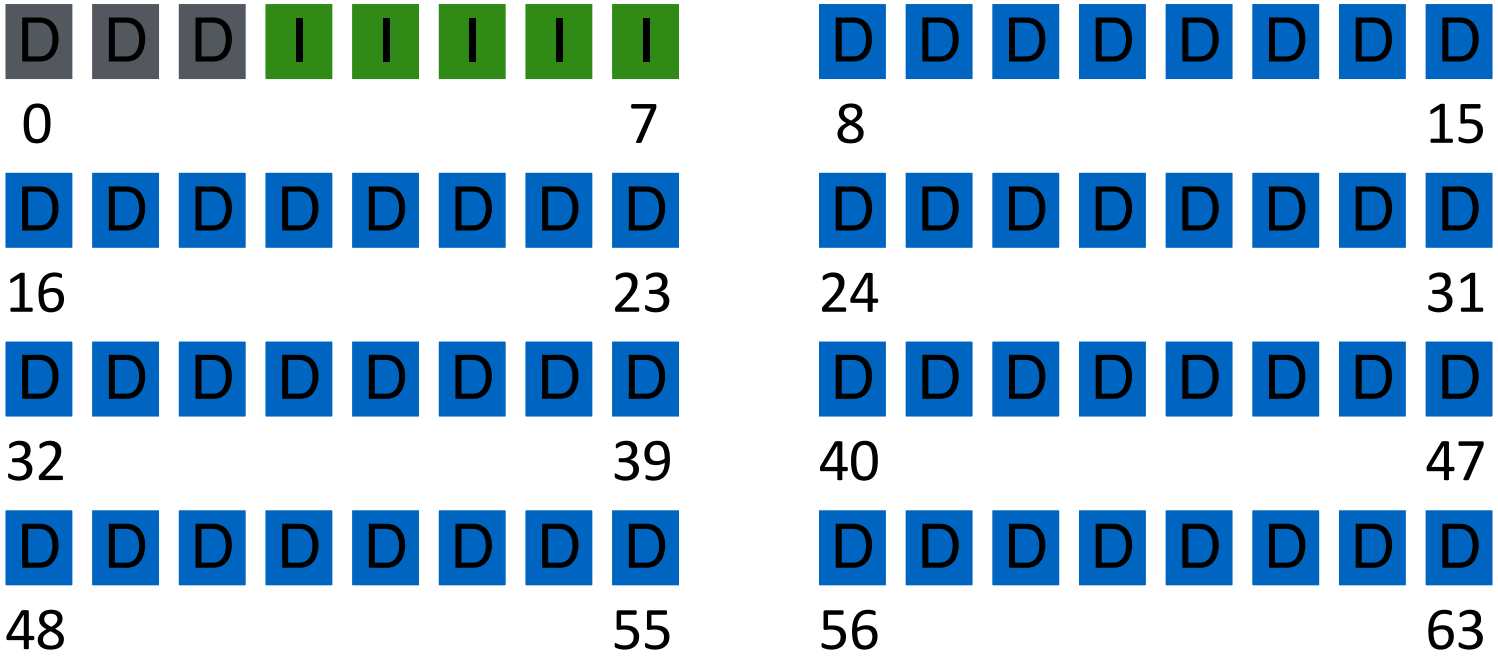
Inode

type
uid
rwx
size
Blocks (optional)
time
ctime
links_count
direct_ptr[N]
indirect_ptr[N+X]
//Some stat structure

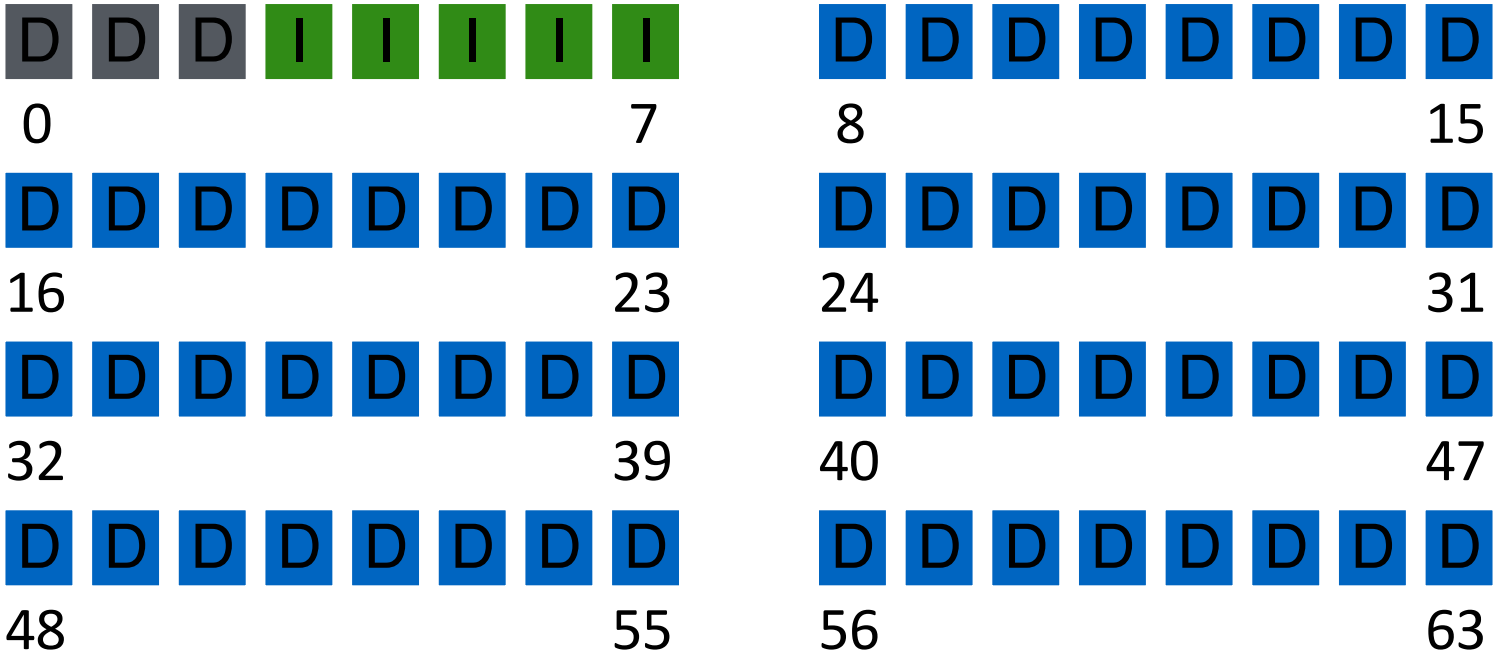
Assume 256 byte inodes (16 inodes/block).
What is offset for inode with number 0?



Assume 256 byte inodes (16 inodes/block).
What is offset for inode with number 4?

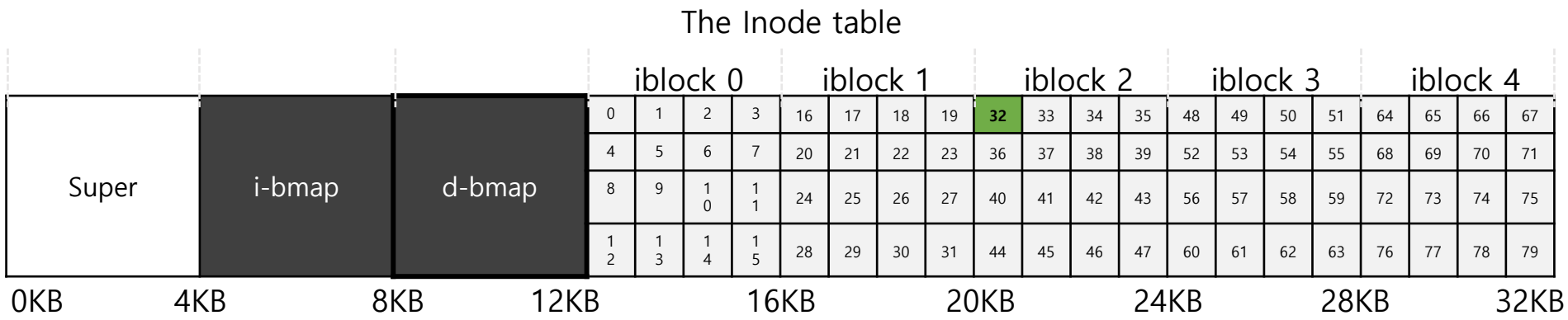


Assume 256 byte inodes (16 inodes/block).
What is offset for inode with number 40?



File Organization: The inode

- Each inode is referred to by inode number.
  - by inode number, File system calculate where the inode is on the disk.
  - Ex: inode number: 32
    - Calculate the offset into the inode region (32 x sizeof(inode) (256 bytes) = 8192
    - Add start address of the inode table(12 KB) + offset into inode region = 20 KB



Directories

File systems vary

Common design:

- Store directory entries in data blocks
- Large directories just use multiple data blocks
- Use bit in inode to distinguish directories from files

Various formats could be used

- lists
- b-trees

Simple Directory List Example

valid	name	inode
1	.	134
1	..	35
1	foo	80
1	bar	23

unlink(“foo”)

Hard links and Soft (symbolic) links

Hard Link :

- A hard link acts as a copy (mirrored) of the selected file. It accesses the data available in the original file.
- If earlier selected file is deleted, the hard link to the file will still contain the data of that file.

ln /path/to/source /path/to/link

Soft Link :

- A soft link (also known as symbolic link) acts as a pointer or a reference to the file name. It does not access the data available
- in the original file. If the earlier file is deleted, the soft link will be pointing to a file that does not exist anymore

ln -s /path/to/source /path/to/link

Allocation

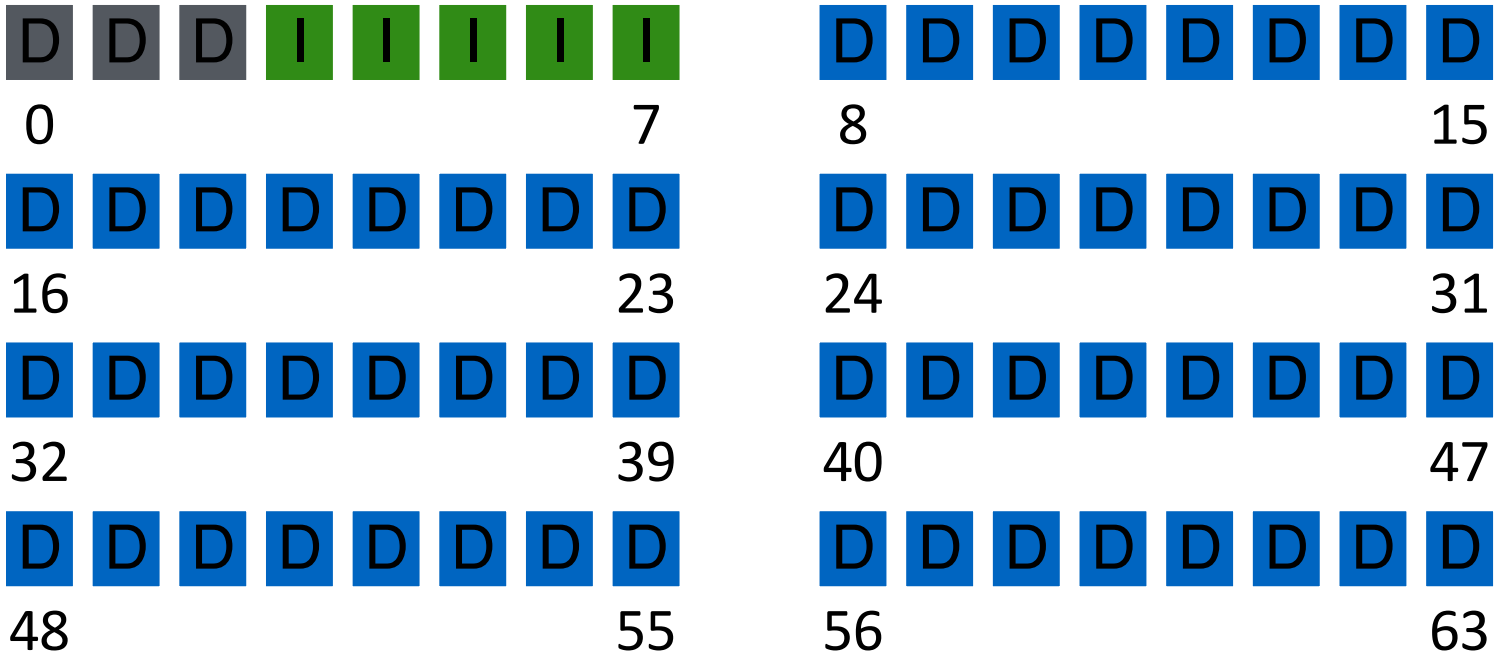
How do we find free data blocks or free inodes?

Free list

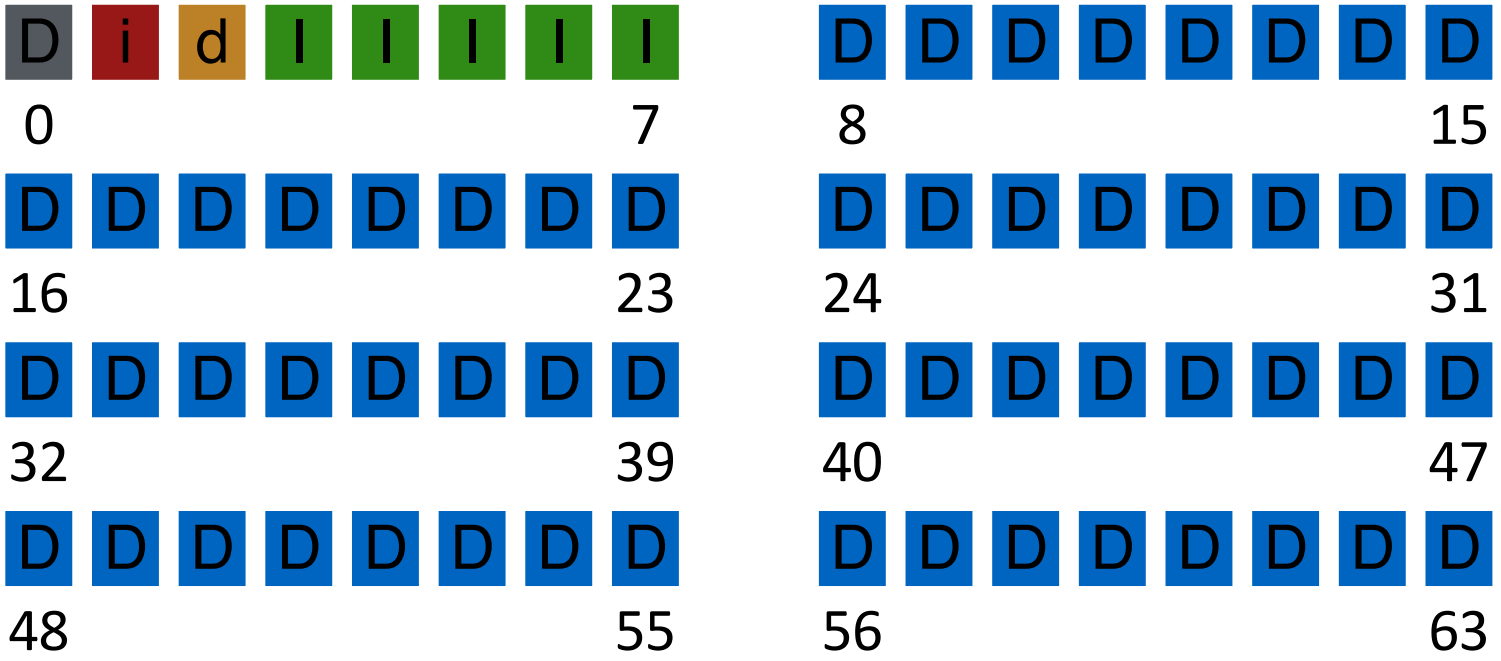
Bitmaps

Tradeoffs!

Bitmaps?


D D D I I I I I
0 7
D D D D D D D D
8 15
D D D D D D D D
16 23
D D D D D D D D
24 31
D D D D D D D D
32 39
D D D D D D D D
40 47
D D D D D D D D
48 55
D D D D D D D D
56 63

Opportunity for Inconsistency



(Need file system checking)

Superblock

Need to know basic FS configuration metadata, like:

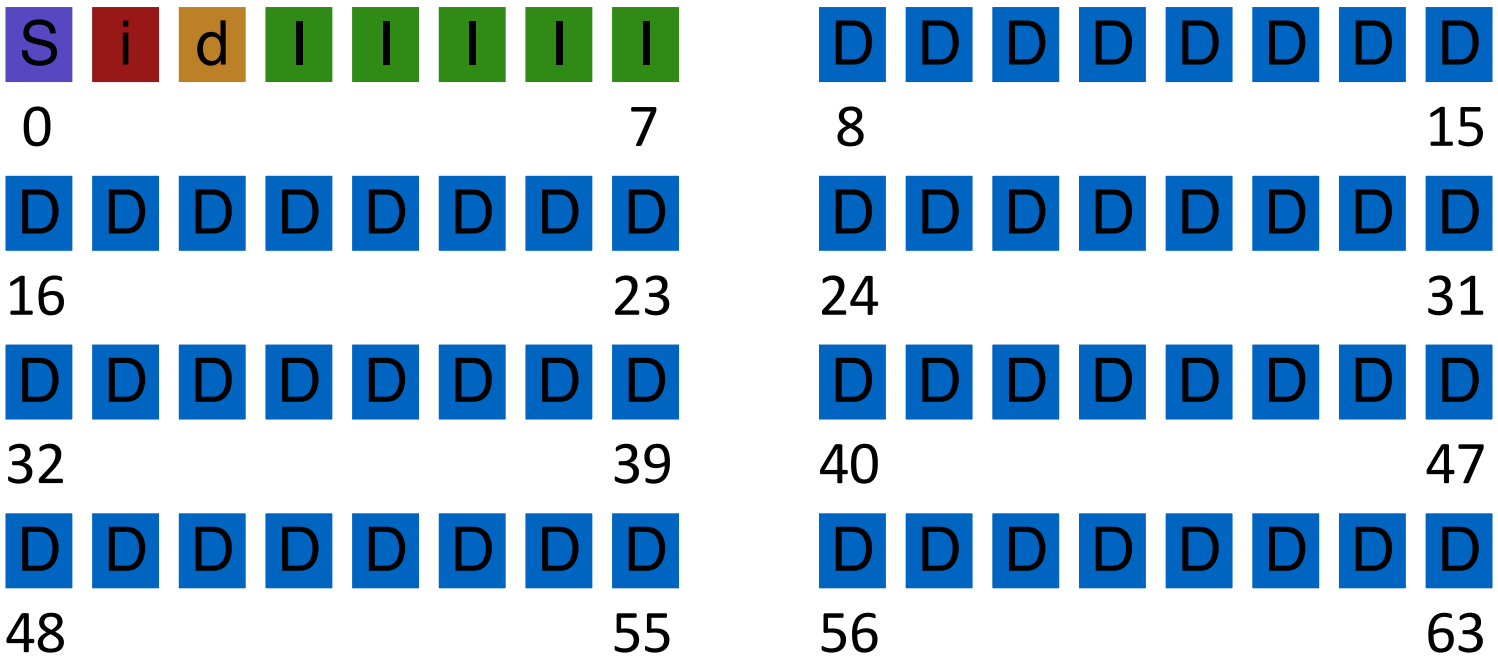
- block size
- # of inodes

Store this in superblock

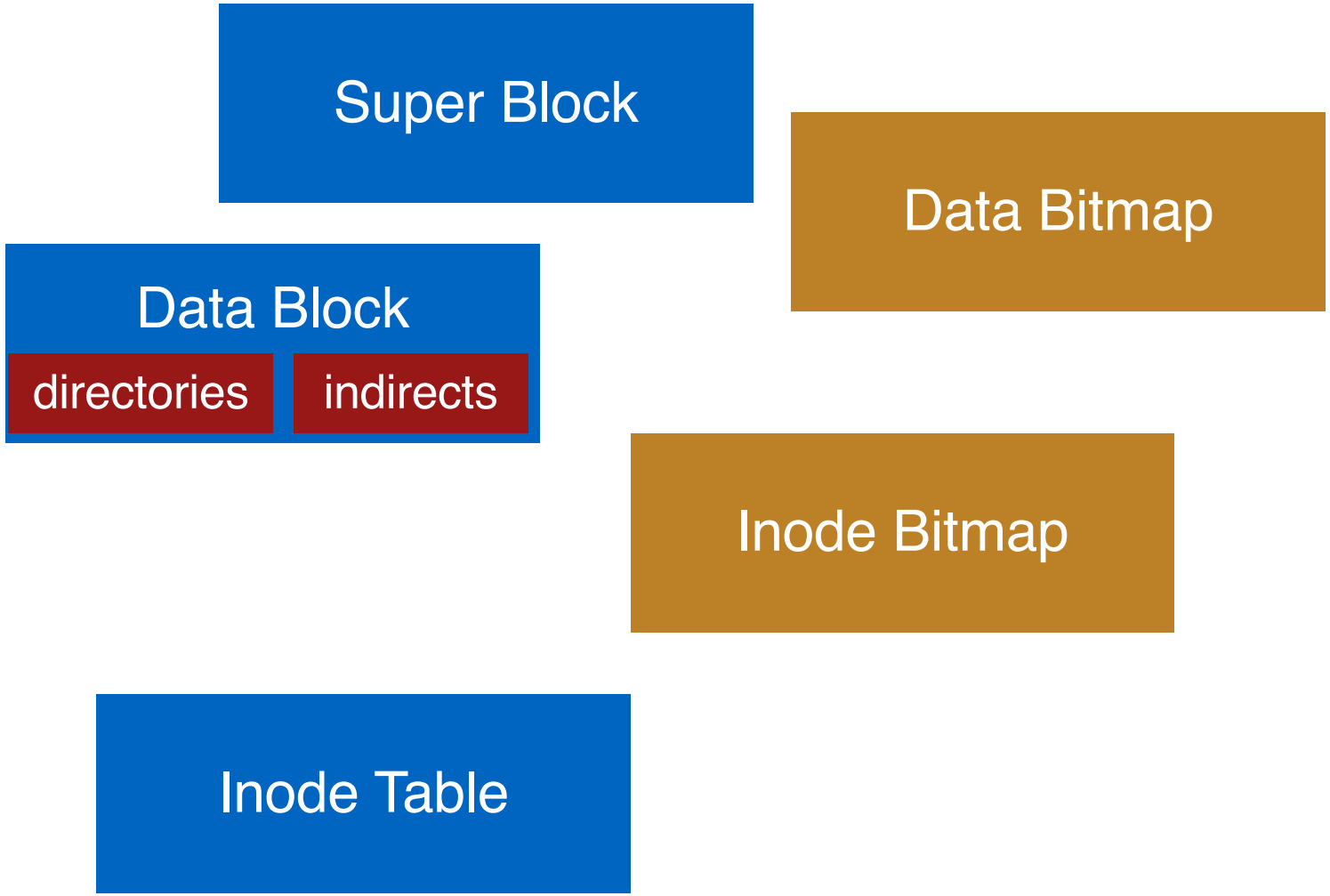
Superblock – Real FS (also FUSE)

```
Struct superblock{
    start address of inode bitmap
    start address of data block bitmap
    start address of inode region
    start address of data block region
    //Anything else that is required
}
```

Superblock


S i d I I I I I
D D D D D D D D
0 7 8 15
D D D D D D D D
D D D D D D D D
16 23 24 31
D D D D D D D D
D D D D D D D D
32 39 40 47
D D D D D D D D
D D D D D D D D
48 55 56 63

On-Disk Structures



Part 2 : Operations

- create file
- write
- open
- read
- close

How do they affect the data structures in the filesystem?

create /foo/bar

data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data
		read				
					read	
			read			
						read
	read write					
						write
				read write		
			write			

What needs to be read and written?

open /foo/bar

data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data
		read					
					read		
			read				
						read	
				read			

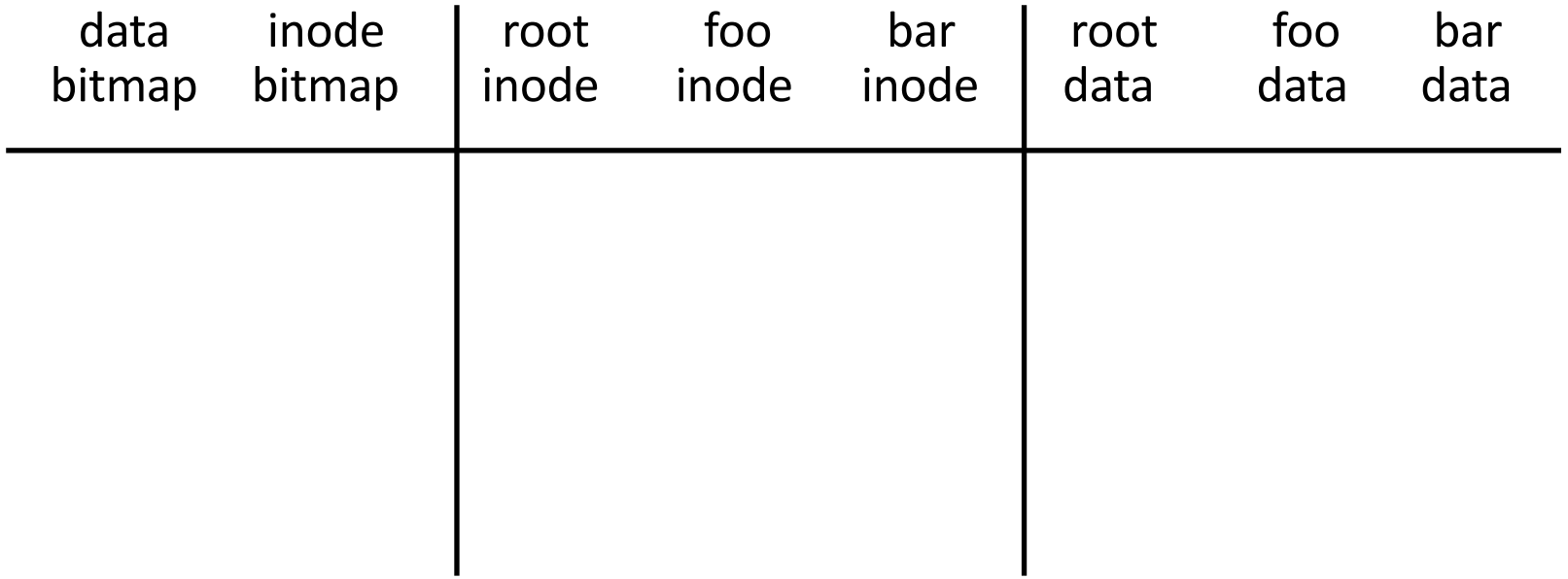
write to /foo/bar (assume file exists and has been opened)

data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data
				read			
read							
write							
							write
				write			

read /foo/bar – assume opened

data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data
				read			
							read
				write			

close /foo/bar



nothing to do on disk!