IT308: Operating Systems

File System Implementation

File System Functionality

- String-based naming of application data (e.g., “photos/koala.jpg” instead of “the bytes between sectors 12,802 and 12,837”)

File System Functionality

- String-based naming of application data (e.g., “photos/koala.jpg” instead of “the bytes between sectors 12,802 and 12,837”)
- Automatic management of free and allocated sectors as files are created and deleted

File System Functionality

- String-based naming of application data (e.g., “photos/koala.jpg” instead of “the bytes between sectors 12,802 and 12,837”)
- Automatic management of free and allocated sectors as files are created and deleted
- Some notion of reliability in the presence of crashes

FS Abstractions: Files and Directories

- A file is associated with metadata like:
  - a user-visible name (e.g., koala.jpg)
  - a size in bytes
  - access permissions (read/write/execute)
  - statistics like last modification time
  - a seek position if open

FS Abstractions: Files and Directories

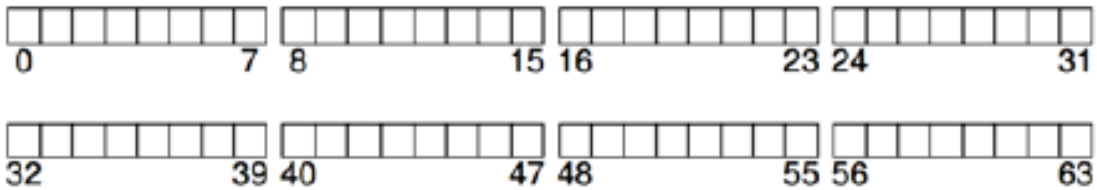
- A file is associated with metadata like:
  - a user-visible name (e.g., koala.jpg)
  - a size in bytes
  - access permissions (read/write/execute)
  - statistics like last modification time
  - a seek position if open
- A directory is a container for other files and directories
  - Typically contains pairs of <user-visible-name, low-level-name>

Blocks

- To build our file system we divide up the disk into a series of blocks (not the same as the disk sectors, which may be smaller!)
- Let's start with 4kB blocks (fairly common size)

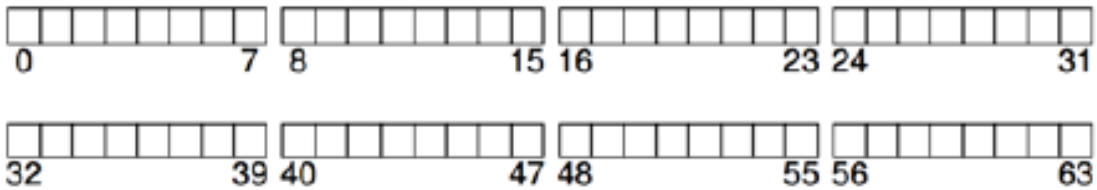
Blocks

- To build our file system we divide up the disk into a series of blocks (not the same as the disk sectors, which may be smaller!)
- Let's start with 4kB blocks (fairly common size)
- For example a small FS with 64 blocks



Blocks

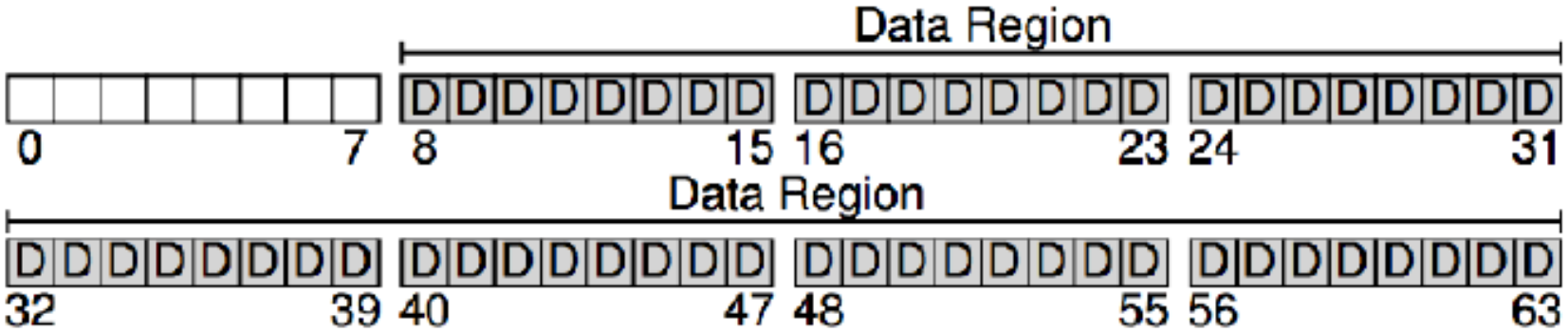
- To build our file system we divide up the disk into a series of blocks (not the same as the disk sectors, which may be smaller!)
- Let's start with 4kB blocks (fairly common size)
- For example a small FS with 64 blocks



- How big is the file system?

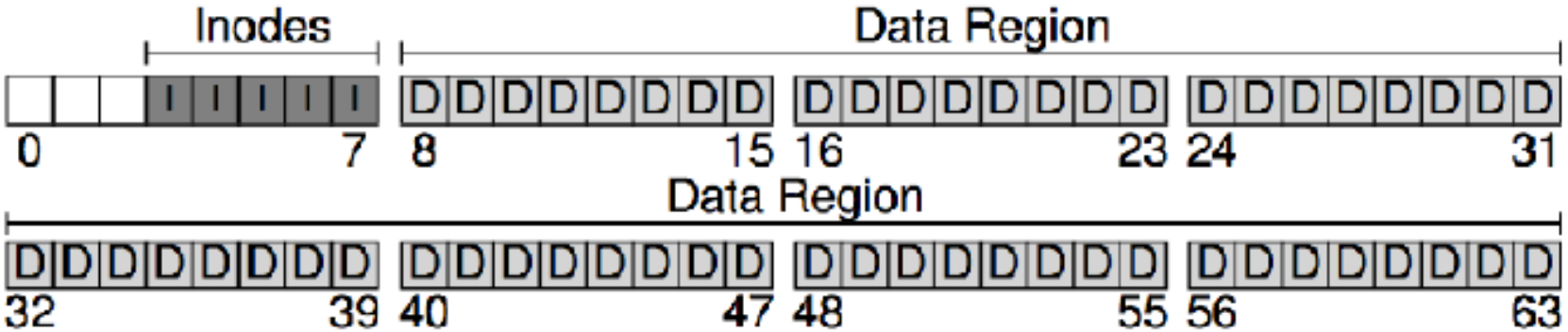
Data Region

- Let's reserve most of the file system for the users' data
- We will call that the Data Region



Inodes

- The structures that contain information about the size of the file, it's creation data, etc. (this is the metadata about the file)
- We will reserve 5 blocks for inodes



Inodes

- Inodes don't need a whole block (4kB) for the metadata for each file
- Assume 256 bytes should be good

Question

We have 5 blocks for inodes, 4kB blocks, 256byte inodes. How many files can we have on this file system?

A. 5

B. 2^{12}

C. 2^{8}

D. 2^{20}

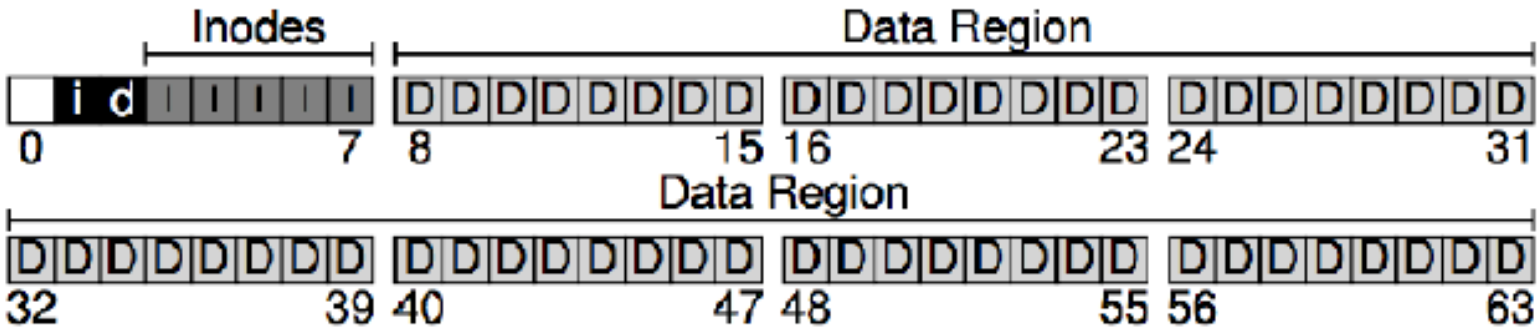
E. 80

Free Space

- We need to track which inodes and which data blocks are free/used
- A simple structure to track such things is a bitmap

Free Space

- We need to track which inodes and which data blocks are free/used
- A simple structure to track such things is a bitmap
- One bit per inode (80 bits), and one bit per data block (56 bits). But lets be lazy and use a whole block for each

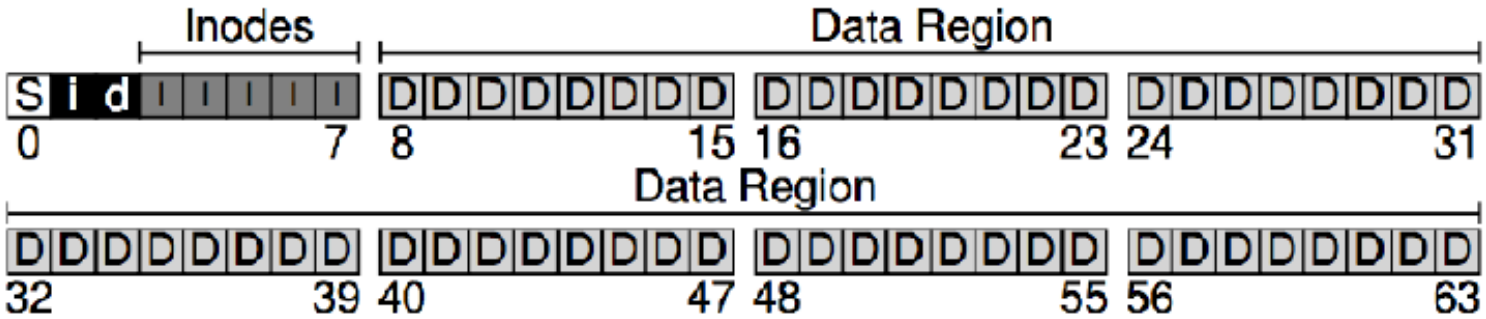


Superblock

- When mounting a file system, the OS needs to know which kind of file system is on the disk, how many inodes, etc.
- This is the metadata about the file system itself

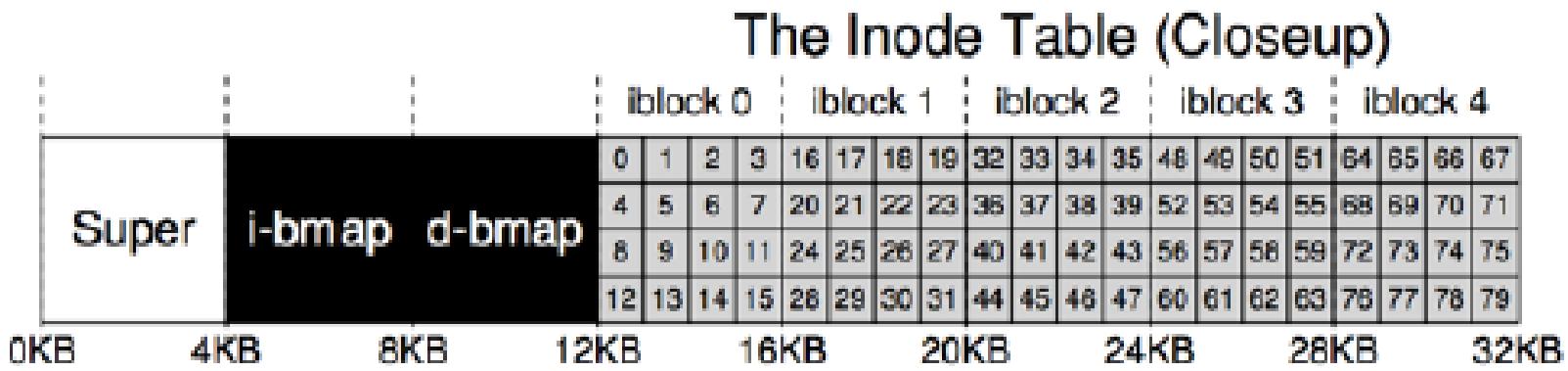
Superblock

- When mounting a file system, the OS needs to know which kind of file system is on the disk, how many inodes, etc.
- This is the metadata about the file system itself
- It is stored in the 0th block. It is the superblock



Finding an inode

- Each inode on the system has a number
- To locate inode 32 you need to compute the address on disk:
  - $32 \star \text{sizeof(inode)} + \text{start of inode region} = 8\text{kB} + 12\text{kB} = 20\text{kB}$



Disk Sector

- Recall that disks are not byte addressable
- You have to read a whole sector, typically 512 bytes
- So what sector is address 20kB in?
  - (20*1024/512)=40

What is stored in the inode?

Size	Name	What is this inode field for?
2	mode	can this file be read/written/executed?
2	uid	who owns this file?
4	size	how many bytes are in this file?
4	time	what time was this file last accessed?
4	ctime	what time was this file created?
4	mtime	what time was this file last modified?
4	dtime	what time was this inode deleted?
2	gid	which group does this file belong to?
2	links_count	how many hard links are there to this file?
4	blocks	how many blocks have been allocated to this file?
4	flags	how should ext2 use this inode?
4	osd1	an OS-dependent field
60	block	a set of disk pointers (15 total)
4	generation	file version (used by NFS)
4	file_acl	a new permissions model beyond mode bits
4	dir_acl	called access control lists

- Notice: size, and block pointers

Direct Pointers

- If the inode has a fixed number of pointers, this fixes the max size of the file: pointers $\star$ block size
- If there are 12 direct pointers: 48kB files

Indirect Pointers

- If we need bigger files we can use an additional indirect pointer which is a pointer to a data block, filled with pointers
- A 4kB block with 4 byte pointers = 1024 pointers in a block
- So (12+1024) $\star$ 4kB = 4144kB

Storing Directories

- Directories are often treated like special files
- They are just a set of data entries that are names + inode numbers
- Usually we find the inode number of a directory in its parent directory

inum	reclen	strlen	name
5	4	2	.
2	4	3	..
12	4	4	foo
13	4	4	bar
24	8	7	foobar

Question

How can you find the inode for the root directory?

A. Look in the superblock or in a well-known inode

B. Scan all the nodes

C. Scan all the data blocks

Example: Reading

- Let's try and read a file: /foo/bar
- Assume file is 12 kB (3 data blocks)
- We have to traverse directories and inodes
- We have to also write the last accessed time

	data bitmap	inode bitmap	root inode	foo inode	bar inode	root data	foo data	bar data[0]	bar data[1]	bar data[2]
open(bar)			read			read				
				read						
							read			
					read					
read()					read					
					write			read		
read()					read					
					write					read
read()					read					
					write					

Example: Creating/Writing a File

- When creating we have to do lots of writes!
- We have to write to the inode allocation bitmap and to the directory, etc.
- We also have to allocate data blocks for the file we want to write and update the inode with that mapping as we go.