Persistence: File System Implementation

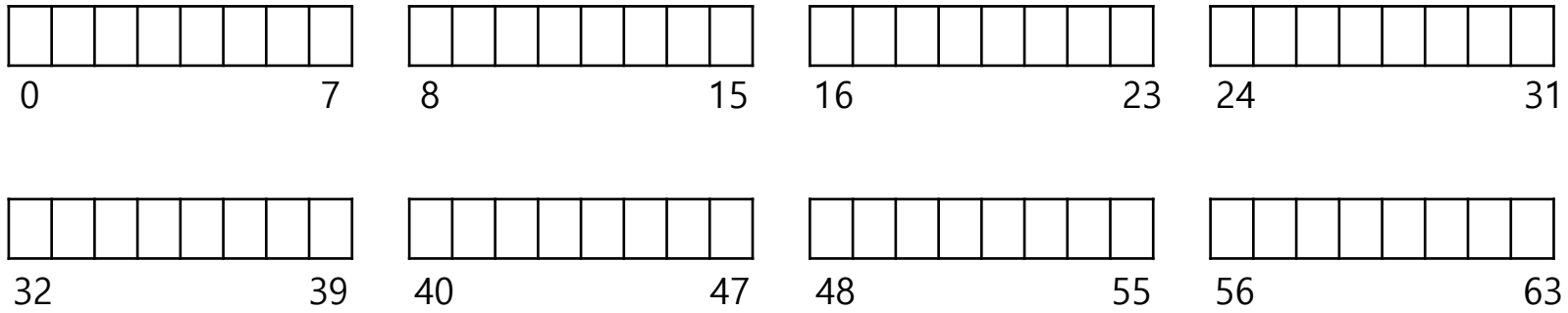
Sridhar Alagar

Review: Files

- Files: sequence of bytes with logical addresses
- Inode: contains meta data; one inode per file
- Directory: maps file name to inode number
- Per process open file table: file descriptor for every open file

File System Organization

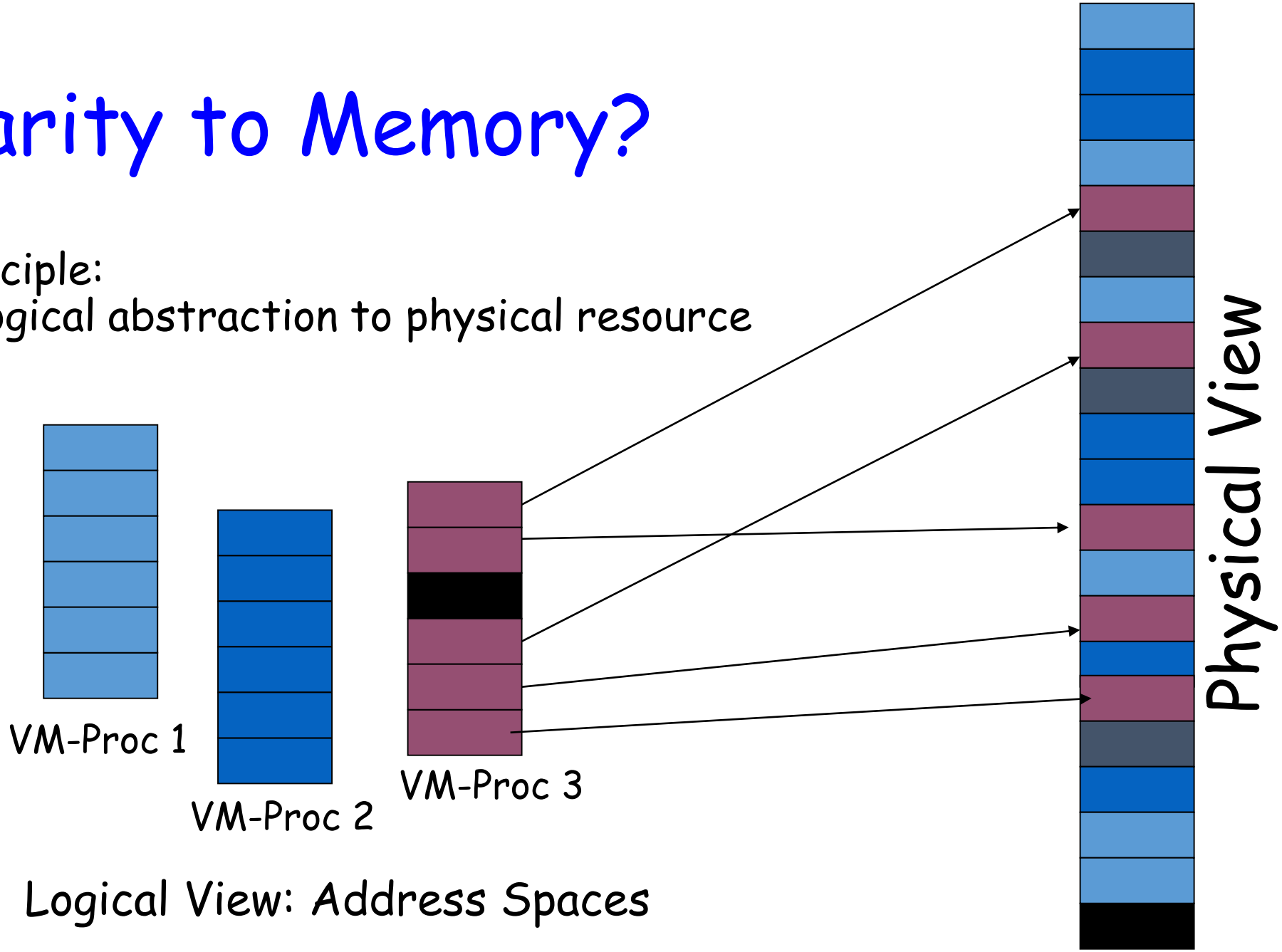
Given: disk is divided into fixed size **blocks**



Want: some structure to map files to disk blocks

Similarity to Memory?

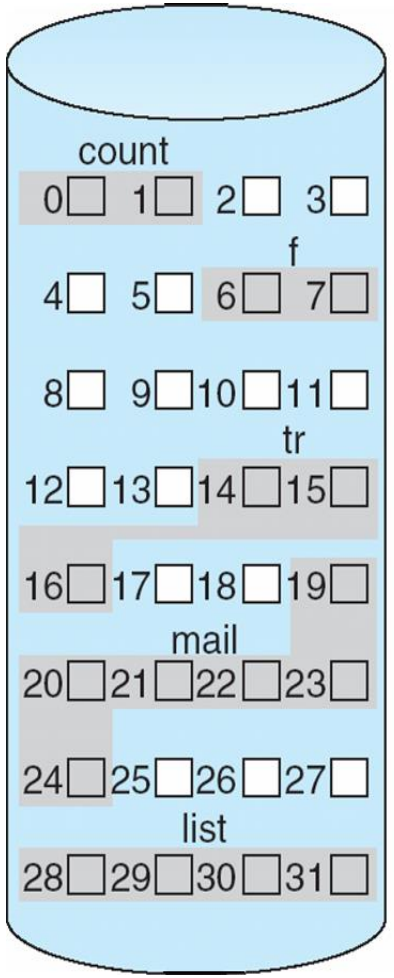
Same principle:
map logical abstraction to physical resource



Logical View: Address Spaces

Contiguous Allocation

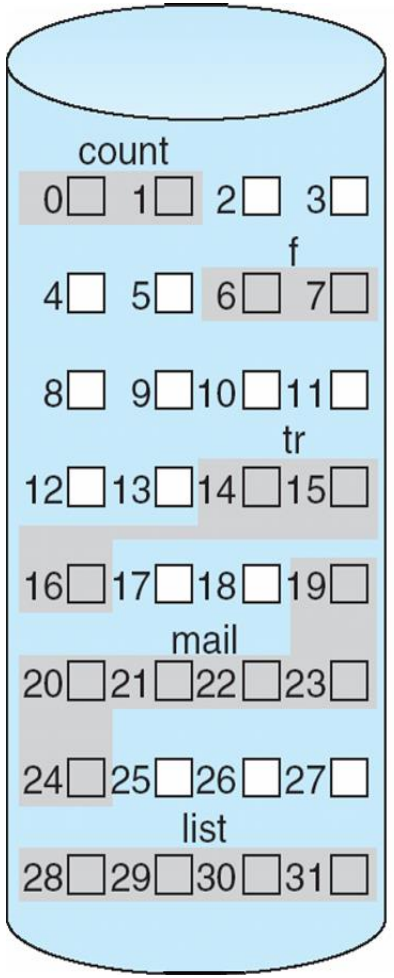
- Allocate required number of blocks(sectors) contiguously
- Simple; little overhead
- Good performance for sequential access
- Easy calculation for random access



file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Contiguous Allocation - Drawbacks

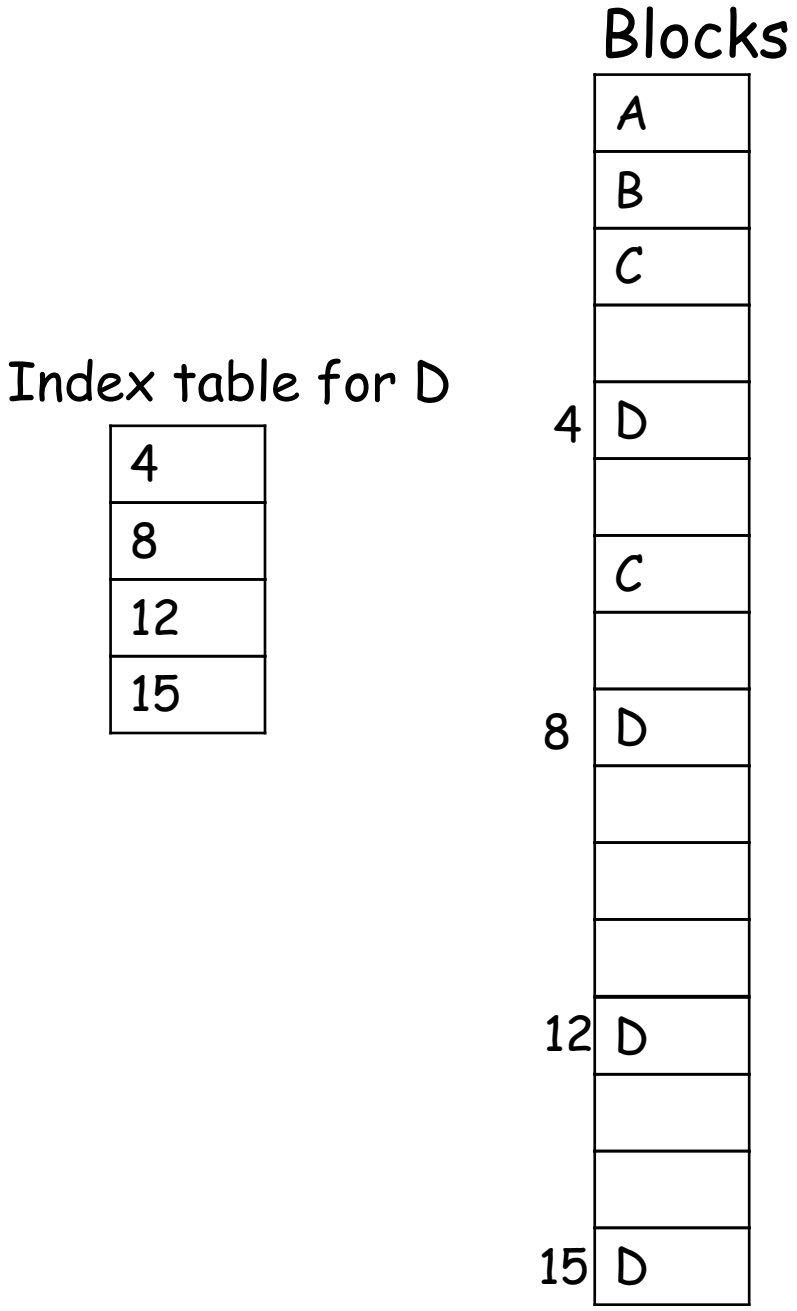
- External fragmentation
- May not be able to grow file
- Need compaction



file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Allocation - Indexed

- Allocate blocks (non-contiguously)
- Keep index table to locate the blocks of a file
- No external fragmentation
- Where is the index table stored?
  - inode

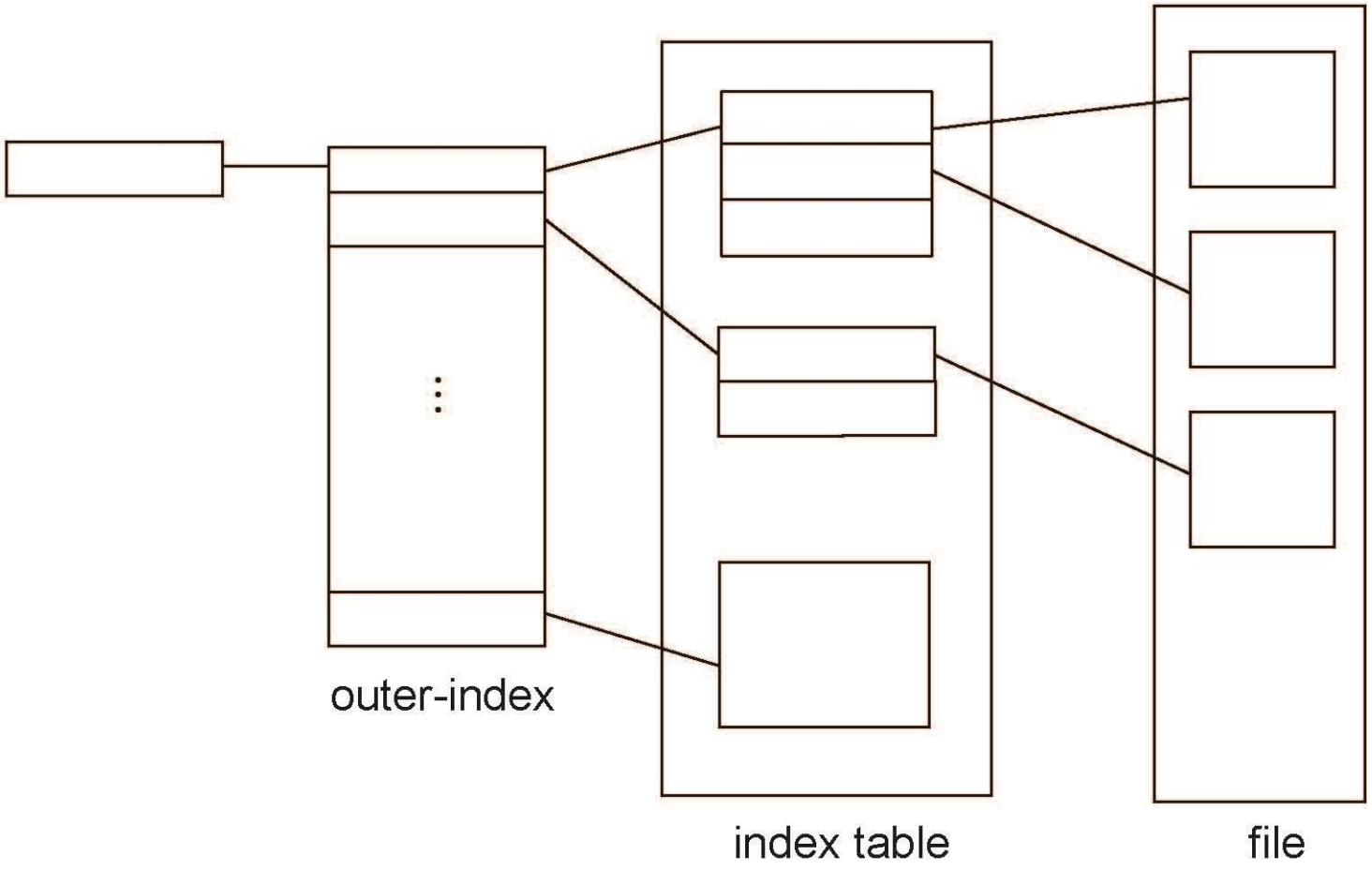


How big is the index table?

- Assume max file size 1 MB and block size 512 bytes
- What is the size of index table?

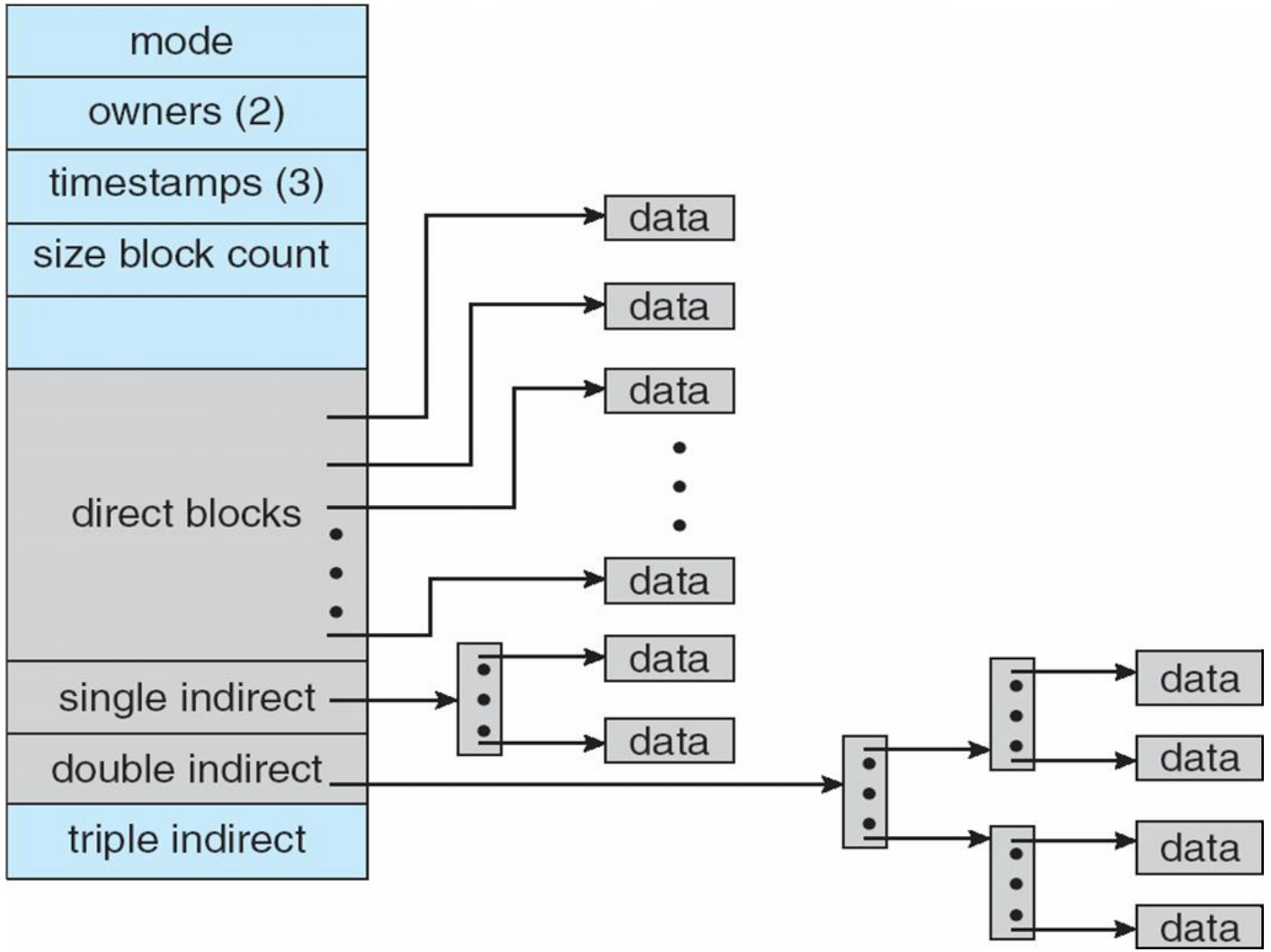
- Assume max file size 4 GB and block size 4K bytes
- What is the size of index table?

Multi-level Index Table

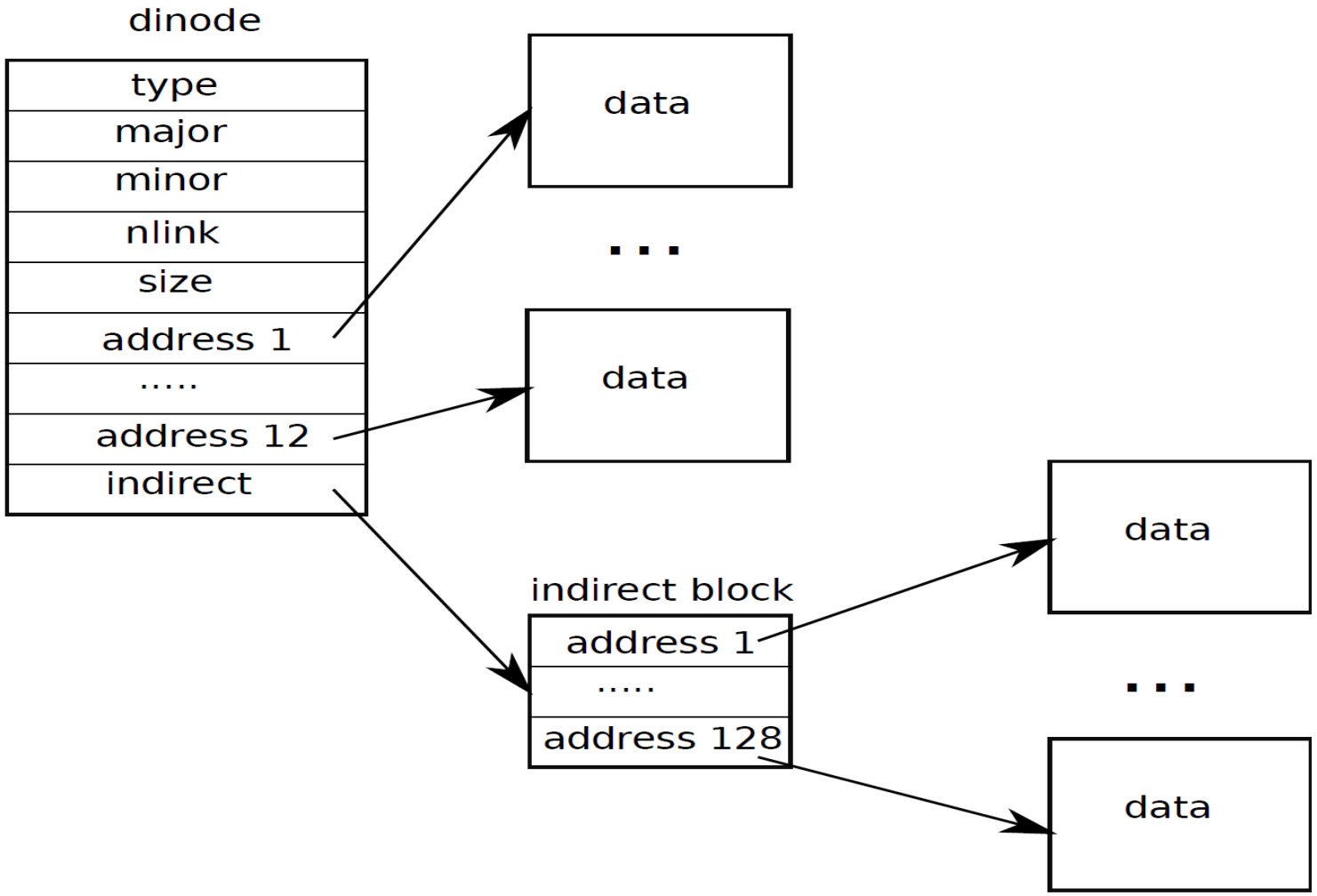


Dynamic Allocation

- Overhead (blocks for indirect pointers) allocated only when the file size grows
- For small file size direct blocks would suffice

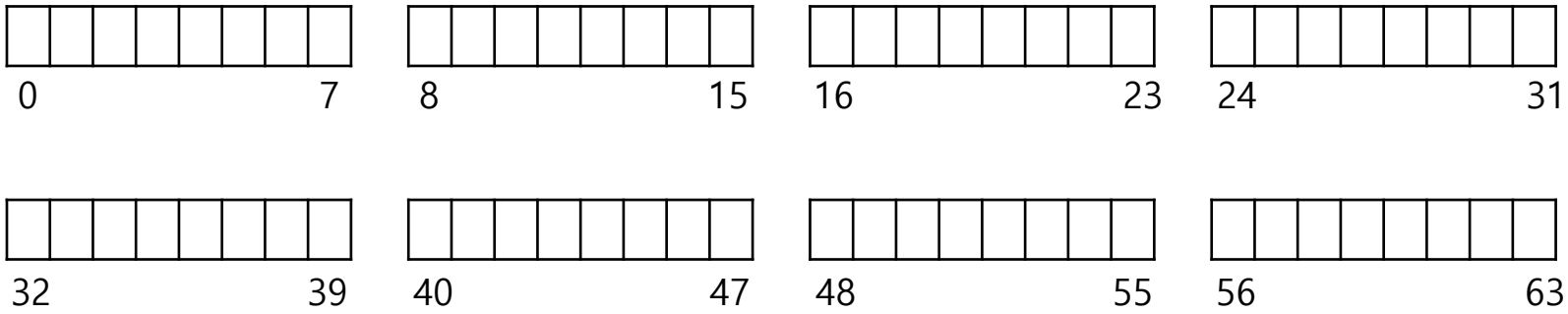


XV6 - Index table



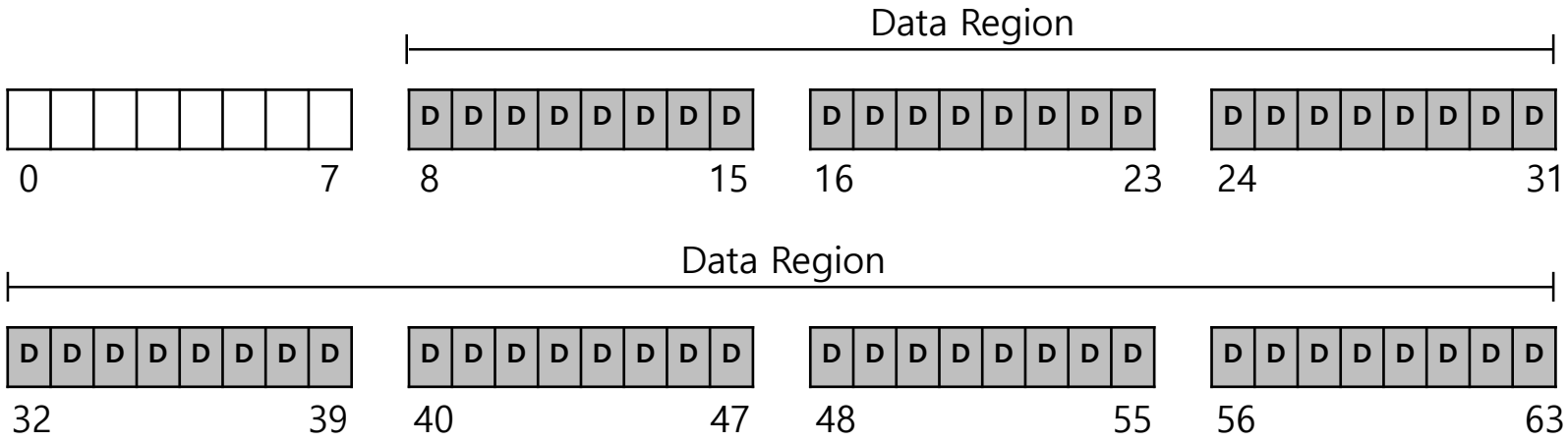
A simple File System

- 64 blocks of size 4 KB each
- Indexed allocation is used



Data region in file system

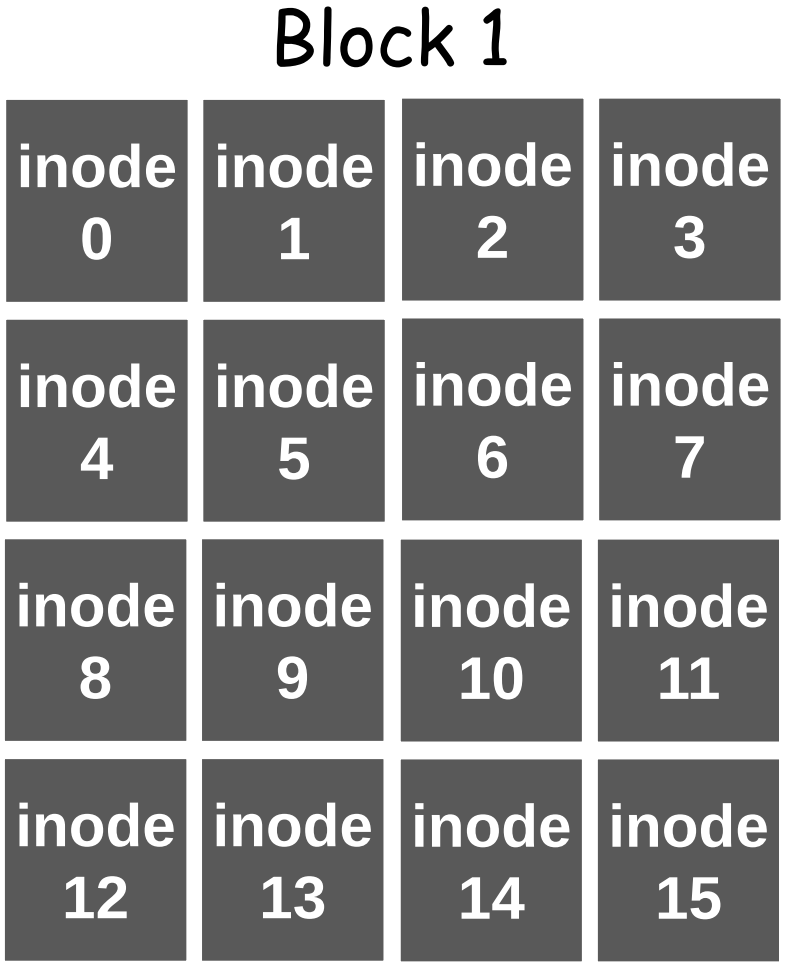
- Reserve blocks (data blocks) to store user data



Where to store inodes in the file system?

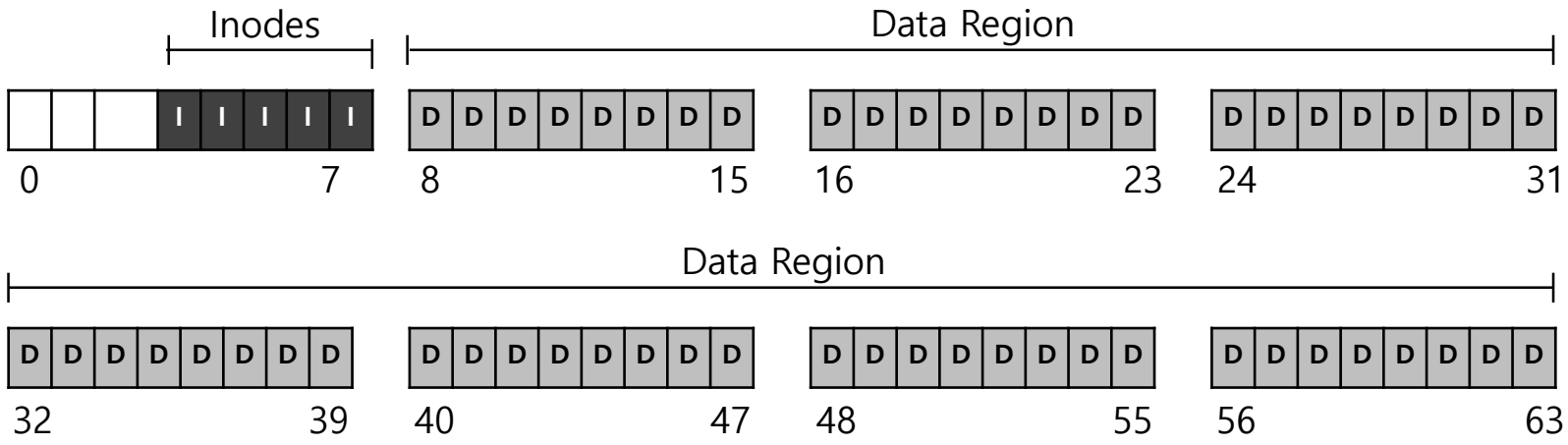
How many inodes in one block?

- Each inode is typically 256 bytes (depends on the FS)
- 4KB disk block
- 16 inodes per inode block.



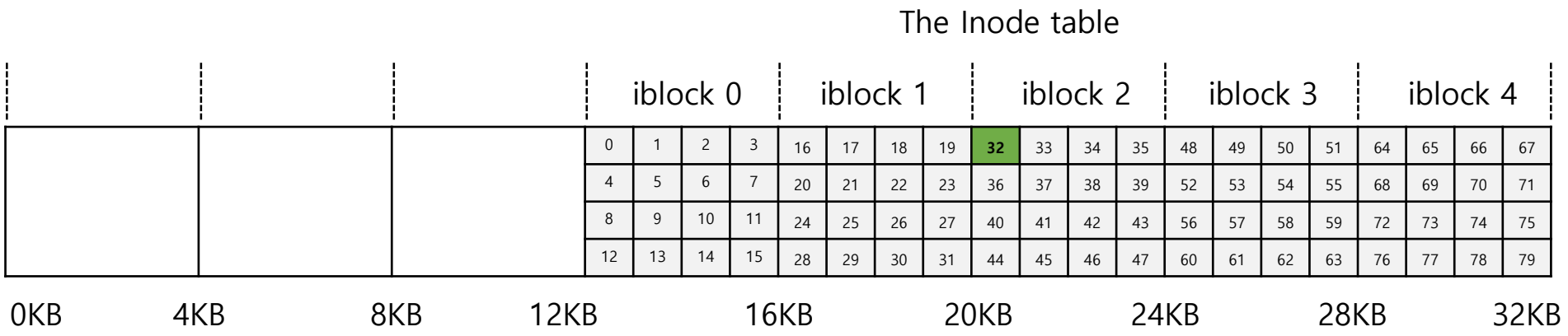
Inode table in file system

- Reserve some blocks (3 – 7) for inodes



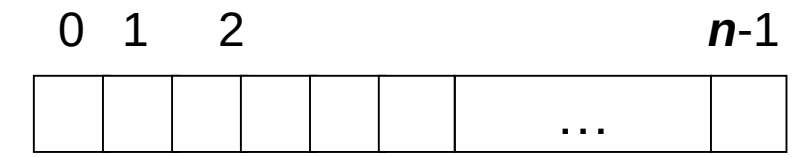
FS Organization: The inode

- How to find an inode, say, 32?



How to find free data blocks/inodes?

- Use bit vector or bit map. 1 bit per block



$$\text{bit}[i] = \begin{cases} 1 \Rightarrow \text{block i free} \\ 0 \Rightarrow \text{block I occupied} \end{cases}$$

Block number = (number of bits per word) *(number of 0-value words) + offset of first 1 bit

CPUs have instructions to return offset of first “1” bit in a word

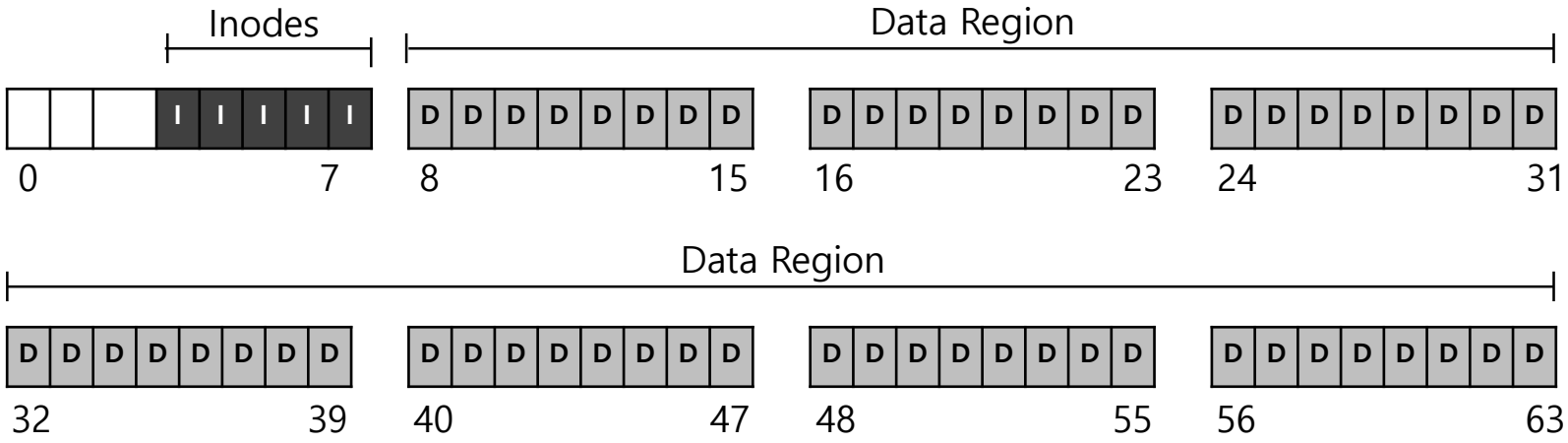
How big is bitmap?

- In our example: 56 bits data blocks, and 80 for inodes

- Disk size = 1 TB, block size = 4KB. Bitmap size?
  - block size = 4KB = 2^{12} bytes
  - disk size = 2^{40} bytes (1 terabyte)
  - $2^{40}/2^{12} = 2^{28}$ bits (or 32MB)
  - 2^{13} = 8192 blocks

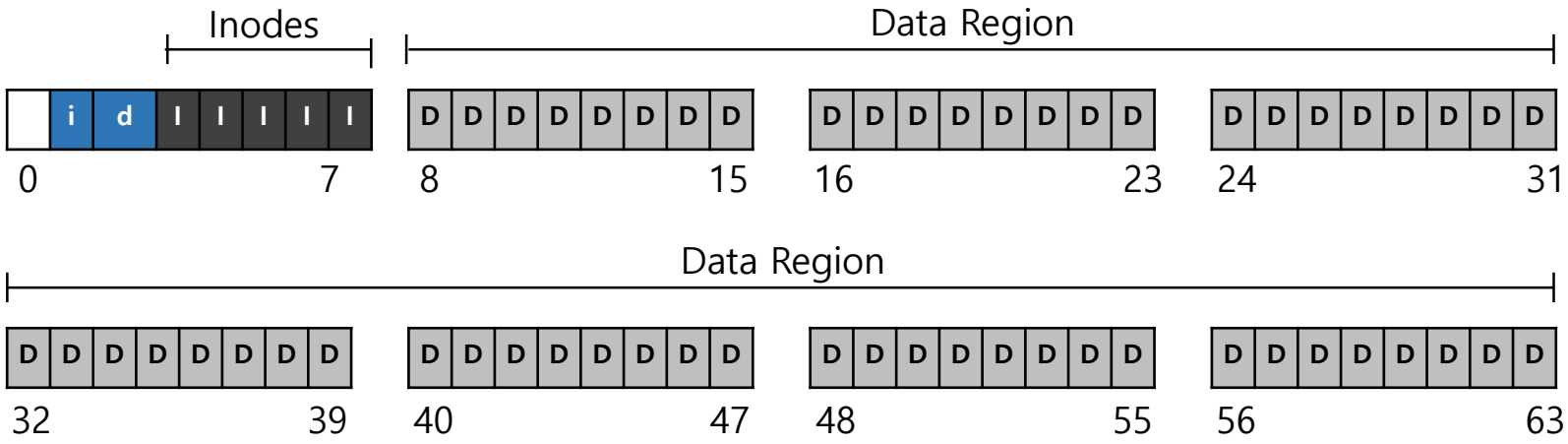
Where are bitmaps stored?

- Bitmap for data blocks
- Bitmap for inodes



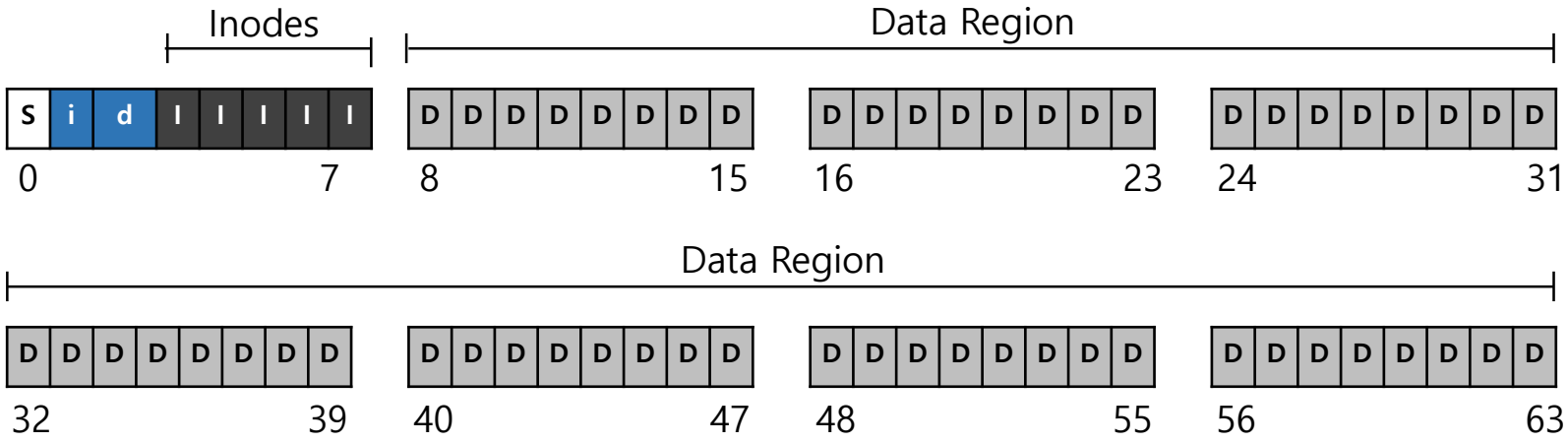
Where are bitmaps stored?

- Bitmap for data blocks is stored in block 1
- Bitmap for inodes is stored in block 2

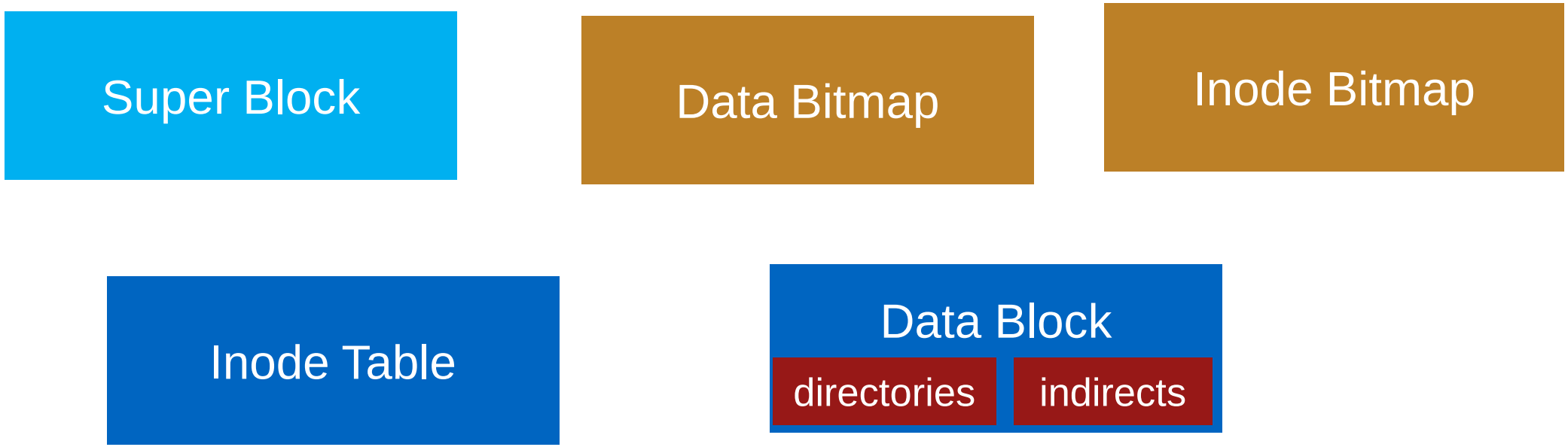


Superblock

- Superblock contains information about the file system
  - disk size, block size, # inodes, beginning of inode table, etc.
- While mounting, superblock is read first



On-Disk Structure



Operations

- What on-disk structures these operations change?
  - Create()
  - Open()
  - Write()
  - Read()
  - Close()

create /foo/bar: What needs to be read and written?

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

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

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

nothing to do on disk!

Disclaimer

- Some of the materials in this lecture slides are from the lecture slides by Prof. Arpaci, Prof. Youjip, and other educators. Thanks to all of them.