File System Implementation

How to implement a simple file system?

By Matthew Tancreti
For COM S 352
Iowa State University

File System Implementation

Many possible approaches to making a file system

Need to understand the data structures for files, directories and free space and how access (e.g., read and write) is implemented

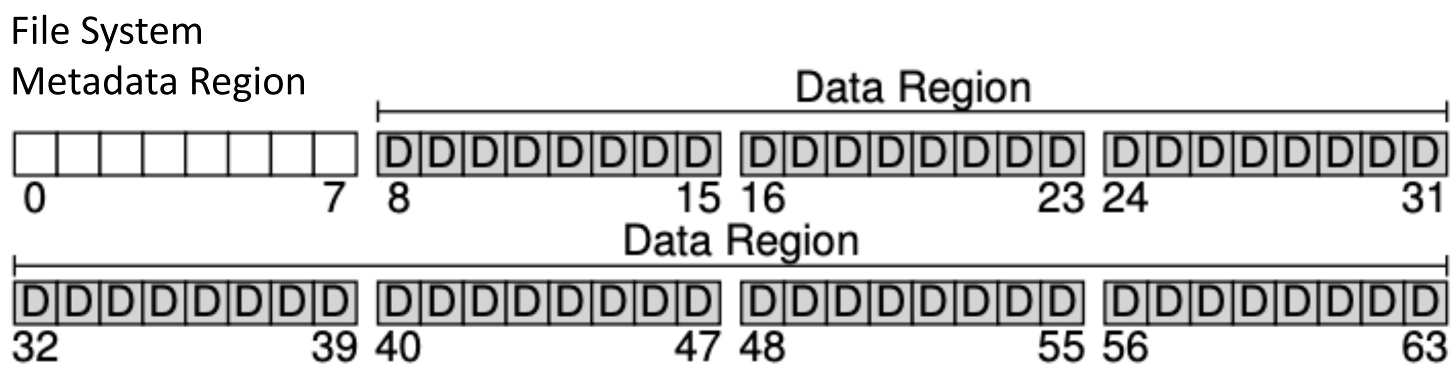
How to implement a simple file system?

Mobile shelving [source]

A Very Simple File System (VSFS)

Disk is divided into fixed sized blocks

A few blocks are reserved for file system metadata (information about the files), the rest stores the data of the files



Contiguous vs Block Allocation

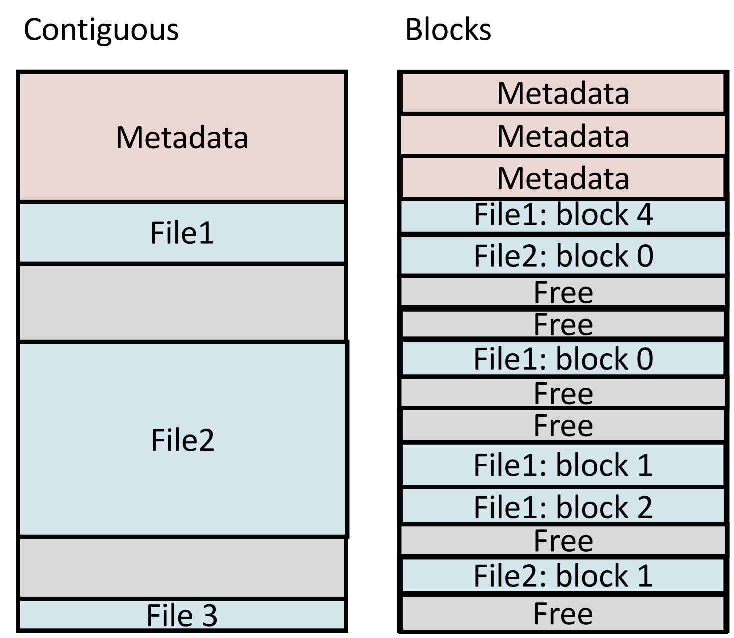
With memory we explored base and bounds (contiguous allocation) or paging

File system have a similar choice (blocks are like pages)

Contiguous allocation has problem of **external fragmentation**

Block allocation has problem of **internal fragmentation**

Block allocation usually wins because internal fragmentation is less of a problem and block allocation is much more efficient in allocating and resizing files



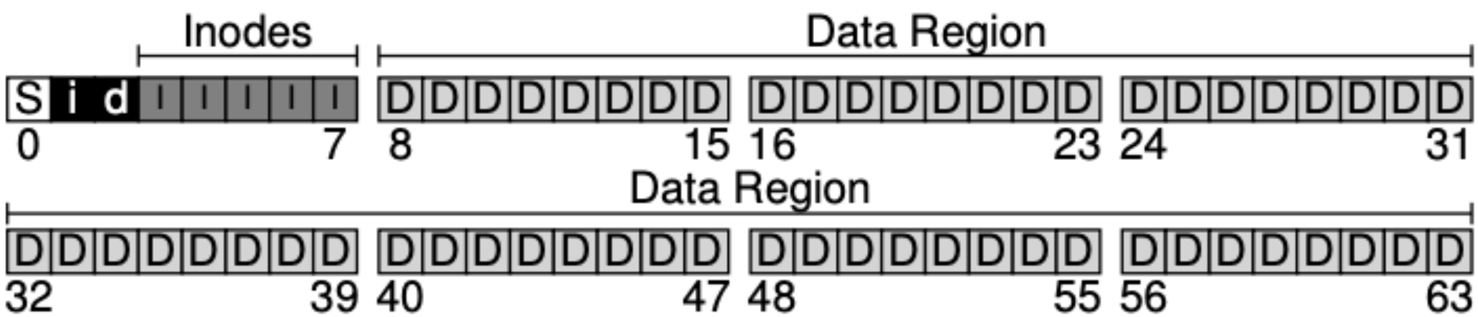
File System Metadata

inodes (I) contains information about a particular file

data bitmap (d) stores which blocks are free/used in the data region

inode bitmap (i) stores which blocks are free/used in the metadata region

Super block (S) contains information about the file system



inode

inode (index node) contains metadata (information) about a file

Example: Ext2 file system 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

points to other inode or data blocks

Direct Indexing

In **direct indexing** the inode for a file has pointers to the data blocks of the file

Suppose an inode has 15 block pointers and blocks are 4KB, what is the largest possible file size?

15 * 4KB = 60KB maximum file size

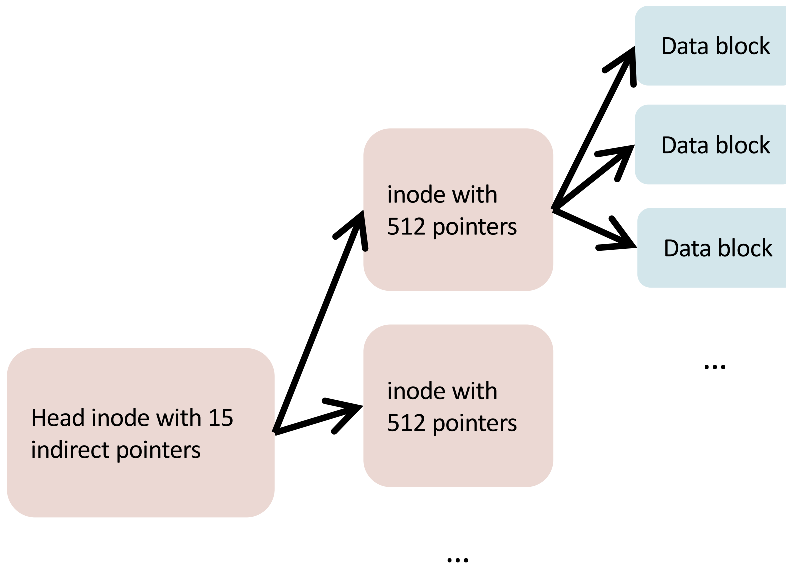
Indirect indexing

An **indirect pointer** points to an inode with more pointers

Suppose pointers are 8 bytes and block size is 4KB, then a block can hold 512 pointers

With one level of indirection the previous example has:

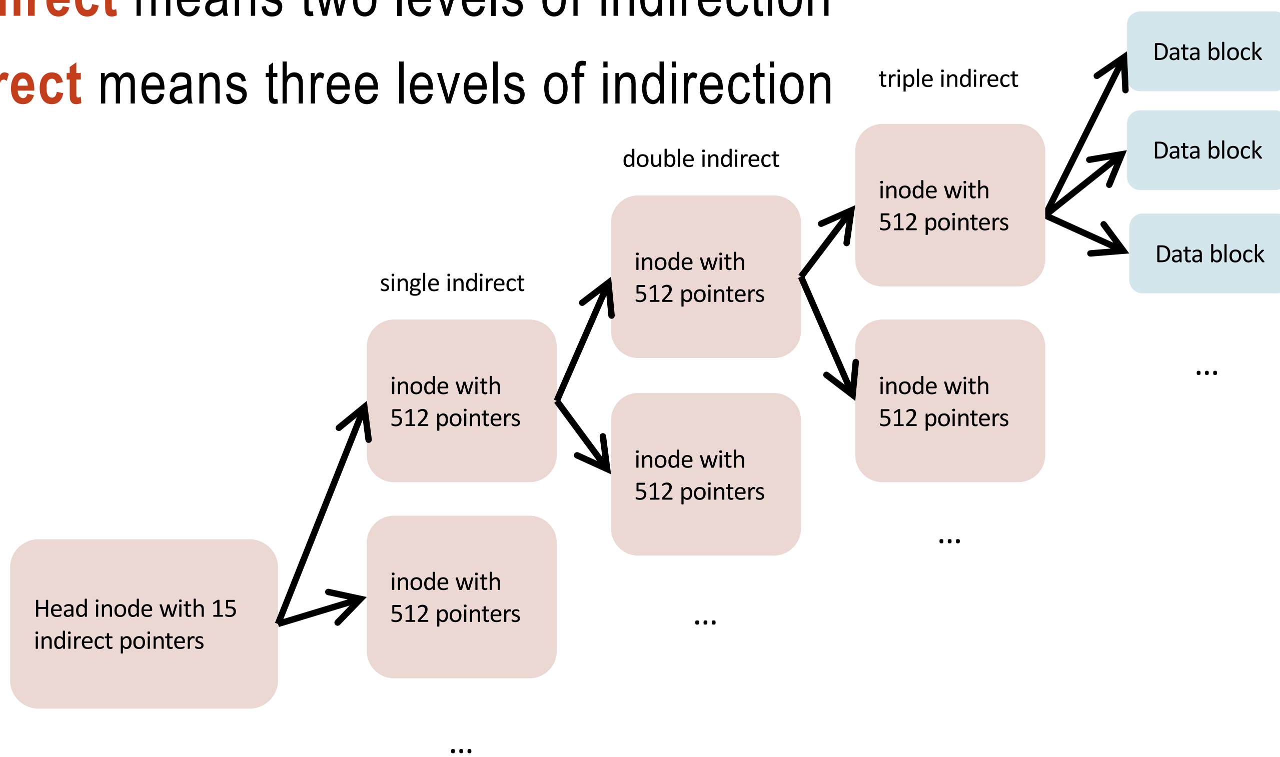
15 * 512 * 4KB = about 30MB maximum file size



Multiple Levels of Indirection

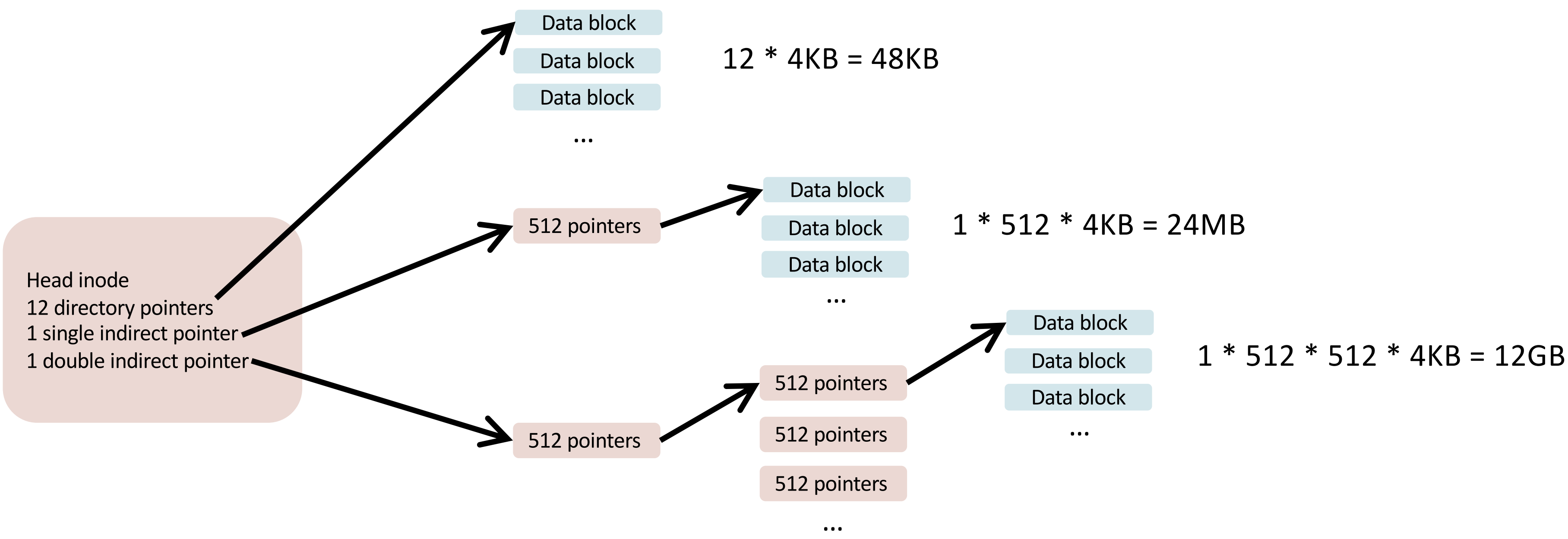
Double indirect means two levels of indirection

Triple indirect means three levels of indirection



Multi-Level Indexing

Head inode may combine multiple levels of indirection



Common Observations of File Systems

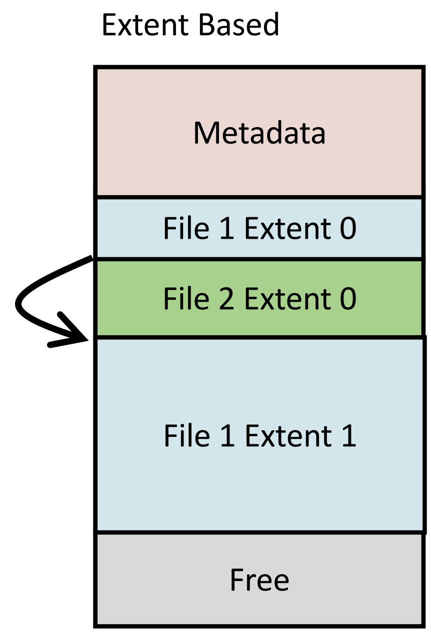
Most files are small	˜2K is the most common size
Average file size is growing	Almost 200K is the average
Most bytes are stored in large files	A few big files use most of space
File systems contains lots of files	Almost 100K on average
File systems are roughly half full	Even as disks grow, file systems remain ˜50% full
Directories are typically small	Many have few entries; most have 20 or fewer

Alternative Approach: Extents

Extent-based file systems allocate variable sized blocks called **extents**

When extent cannot grow further, a new extent is added

Can use array or linked list data structure to hold pointers to extents



Example Reading a File

Assume read from the file /foo/bar

	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**					
								read		
					write					
read()					**read**					
									read	
					write					
read()					**read**					
										read
					write					

Example Creating a Writing File

Assume writing to file /foo/bar

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

Basic Performance Improvements

Caching – holds popular blocks to decrease number of times blocks are read from disk

Write buffering - batch multiple updates into a smaller set of I/O operations