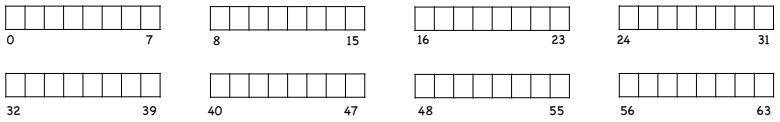
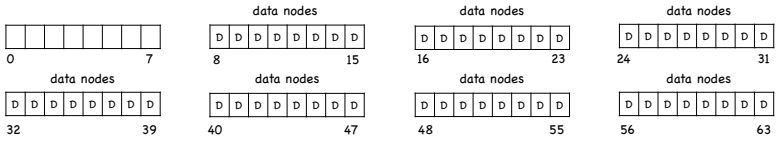
Peeking Inside

- Persistent storage modeled as a sequence of N blocks
  - from 0 to N-1
    - 4KB in this example
  - some blocks store data



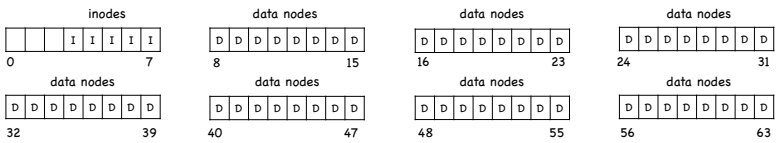
Peeking Inside

- Persistent storage modeled as a sequence of N blocks
  - from 0 to N-1
    - 4KB in this example
  - some blocks store data
  - other blocks store metadata (remember stat()?)
    - an array of inodes
      - at 256 bytes, 16 per block: file system can have up to 80 files



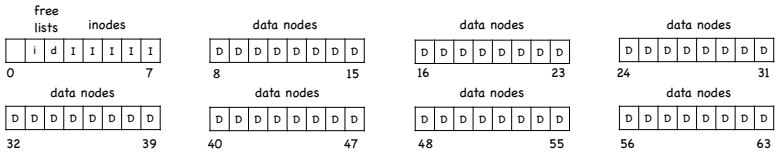
Peeking Inside

- Persistent storage modeled as a sequence of N blocks
  - from 0 to N-1
    - 4KB in this example
  - some blocks store data
  - other blocks store metadata (remember stat()?)
    - an array of inodes
      - at 256 bytes, 16 per block: file system can have up to 80 files



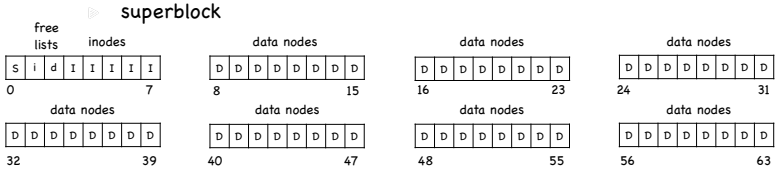
Peeking Inside

- Persistent storage modeled as a sequence of N blocks
  - from 0 to N-1
    - 4KB in this example
  - some blocks store data
  - other blocks store metadata (remember stat()?)
    - an array of inodes
      - at 256 bytes, 16 per block: file system can have up to 80 files
    - two blocks with bitmaps tracking free inodes and data blocks



Peeking Inside

- Persistent storage modeled as a sequence of N blocks
  - from 0 to N-1
    - 4KB in this example
  - some blocks store data
  - other blocks store metadata (remember stat()?)
    - an array of inodes
      - at 256 bytes, 16 per block: file system can have up to 80 files
    - two blocks with bitmaps tracking free inodes and data blocks
    - superblock

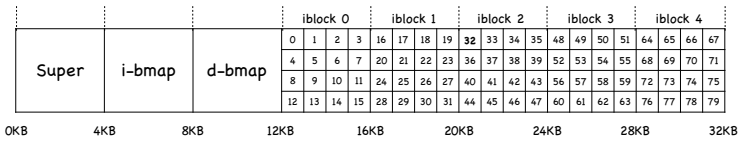


The Superblock

- One logical superblock per file system
  - at a well-known location.
  - contains metadata about the file system, including
    - how many inodes
    - how many data blocks
    - where the inode table begins
    - magic number identifying file system type
  - read first when mounting a file system

The inode

- Low-level file name
- Locating an inode on disk



  - inode 32 is on sector 40
    - can you see why?

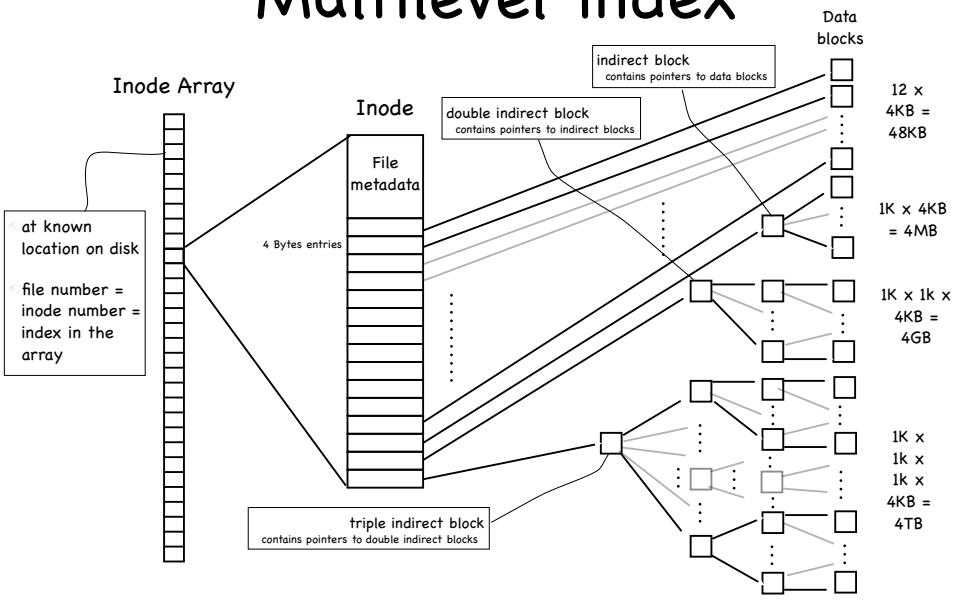
The ext2 inode (simplified)

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?
4	gid	which group does this file belong to?
2	links_count	how many hard links are there to this file?
2	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

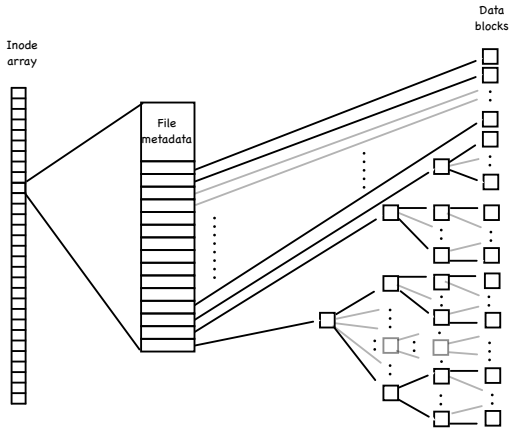
File structure

- Each file is a fixed, asymmetric tree, with fixed size data blocks (e.g. 4KB) as its leaves
- The root of the tree is the file's inode
  - contains a set of pointers
    - typically 15
    - first 12 point to data block
    - last three point to intermediate blocks, themselves containing pointers
      - 13: indirect pointer
      - 14: double indirect pointer
      - 15: triple indirect pointer

Multilevel index



Multilevel index: key ideas



- Tree structure
  - efficient in finding blocks
- High degree
  - efficient in sequential reads
    - once an indirect block is read, can read 100s of data block
- Fixed structure
  - simple to implement
- Asymmetric
  - supports efficiently files big and small

Why Unbalanced Trees? (and other fun facts)

- Most files are small
  - Roughly 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 the space
- File systems contains lots of files
  - Almost 100K on average
- File systems are roughly half full
  - Even as disks grow, file system remain about 50% full
- Directories are typically small
  - Many have few entries; most have 20 or fewer

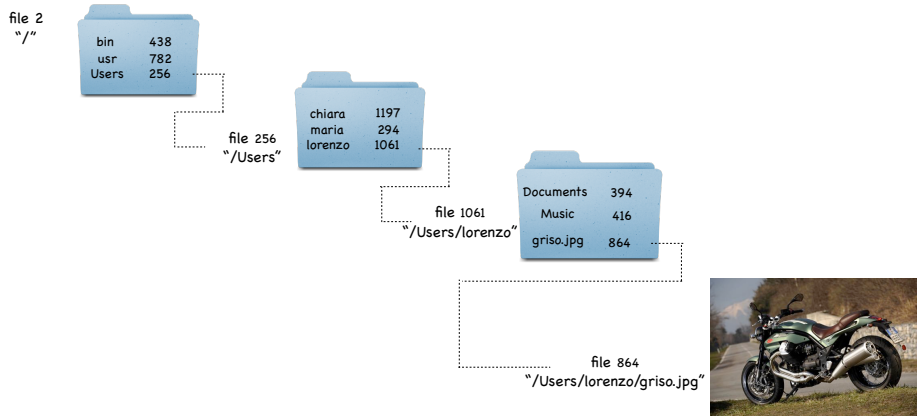
Directory

- A file that contains a collection of mapping from file name to file number

- To look up a file, find the directory that contains the mapping to the file number
- To find that directory, find the parent directory that contains the mapping to that directory's file number...
- Good news: root directory has well-known number (2)

Looking up a file

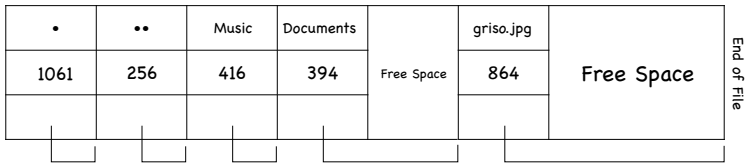
- Find file /Users/lorenzo/griso.jpg



Directory Layout

- Directory stored as a file
  - Linear search to find filename (small directories)

File 1061
/Users/lorenzo



- Larger directories use B trees
  - searched by hash of file name

Reading a File

- First, must open the file
  - open("/CS4410/roster", O_RDONLY)
  - Follow the directory tree, until we get to the inode for "roster"
  - Read that inode
    - do a permission check
    - return a file descriptor fd
- Then, for each read()
  - read inode
  - read appropriate data block (depending on offset)
  - update last access time in inode
  - update file offset in in-memory open file table for fd

Read first 3 data blocks from /CS4410/roster

	data bitmap	inode bitmap	root inode	CS4410 inode	roster inode	root data	CS4410 data	roster data[0]	roster data[1]	roster data[2]
			read()							
						read()				
open(CS4410)				read()						
							read()			
					read()					
					read()					
read()								read()		
					write()					
					read()					
read()									read()	
					write()					
					read()					
read()										read()
					write()					

Writing a File

- Must open the file, like before
- But now may have to allocate a new data block
  - each logical write can generate up to five I/O ops
    - reading the free data block bitmap
    - writing the free data block bitmap
    - reading the file's inode
    - writing the file's inode to include pointer to the new block
    - writing the new data block
- Creating a file is even worse!
  - read and write free inode bitmap
  - write inode
  - (read) and write directory data
  - write directory inode

and if directory block is full, must allocate another block

Read first 3 data blocks from /CS4410/roster

	data bitmap	inode bitmap	root inode	CS4410 inode	roster inode	root data	CS4410 data	roster data[0]	roster data[1]	roster data[2]
			read()							
						read()				
				read()						
							read()			
		read()								
create (/CS4410/roster)		write()								
							write()			
					read()					
					write()					
				write()						
					read()					
	read()									
write()	write()									
								write()		
					write()					
					read()					
write()	read()									
	write()									
									write()	
					write()					
					read()					
write()	read()									
	write()									
										write()
					write()					

Caching

- Reading a long path can cause a lot of I/O ops!
- Cache aggressively!
  - early: fixed sized cache for popular blocks
    - static partitioning can be wasteful
  - current: dynamic partitioning via unified page cache
    - virtual memory pages and file system blocks in a single cache