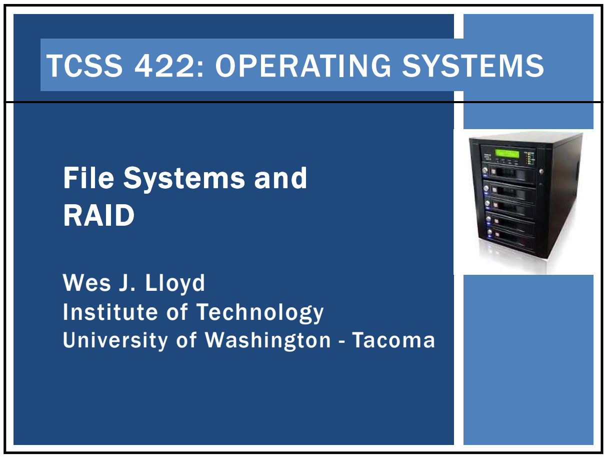
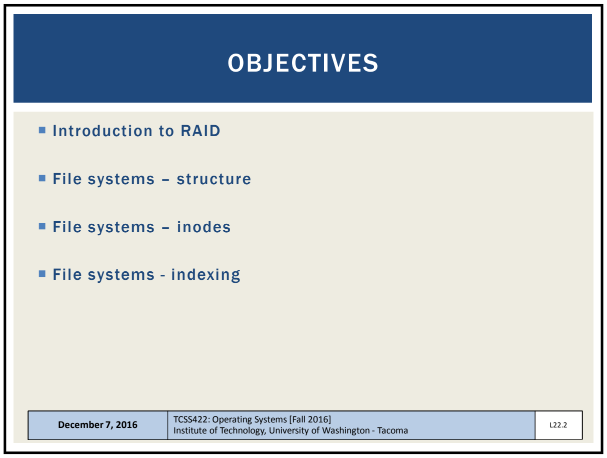


TCSS 422: OPERATING SYSTEMS

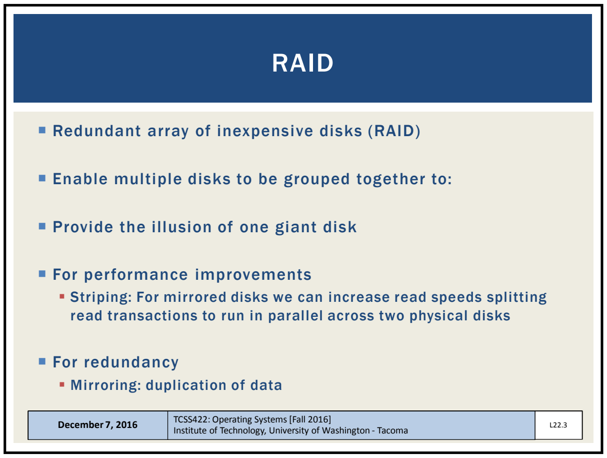
File Systems and RAID

Wes J. Lloyd
Institute of Technology
University of Washington - Tacoma



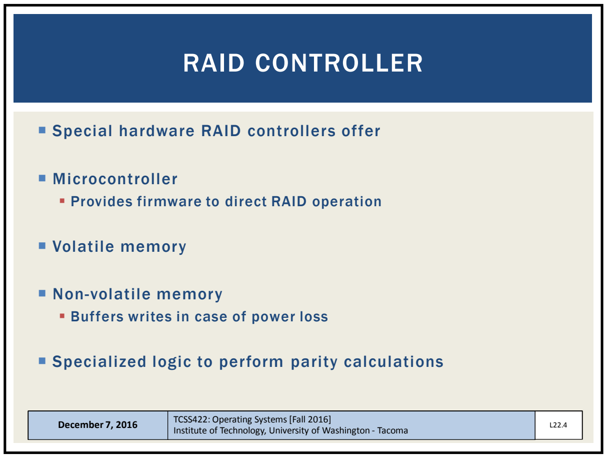
OBJECTIVES

- Introduction to RAID
- File systems – structure
- File systems – inodes
- File systems - indexing



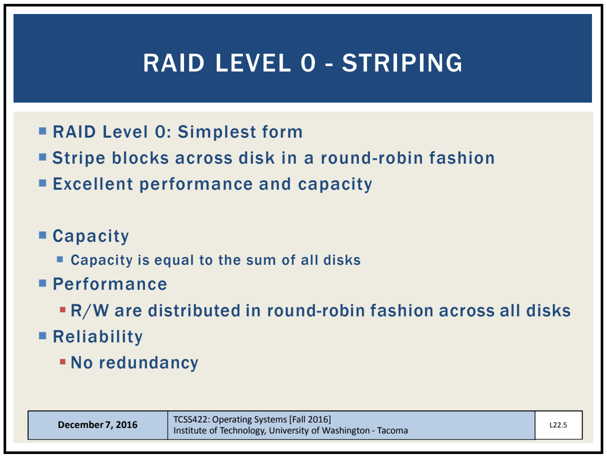
RAID

- Redundant array of inexpensive disks (RAID)
- Enable multiple disks to be grouped together to:
- Provide the illusion of one giant disk
- For performance improvements
  - Striping: For mirrored disks we can increase read speeds splitting read transactions to run in parallel across two physical disks
- For redundancy
  - Mirroring: duplication of data



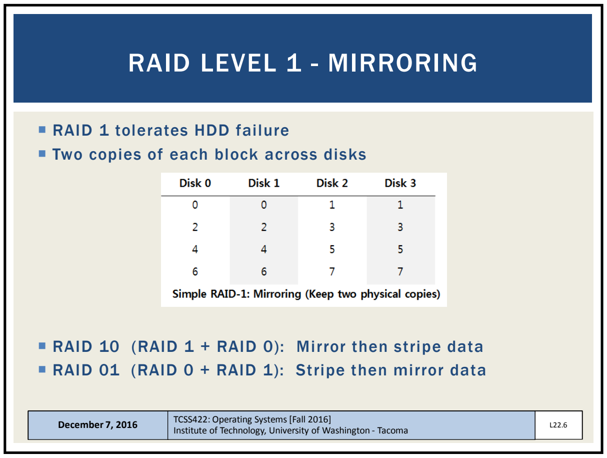
RAID CONTROLLER

- Special hardware RAID controllers offer
- Microcontroller
  - Provides firmware to direct RAID operation
- Volatile memory
- Non-volatile memory
  - Buffers writes in case of power loss
- Specialized logic to perform parity calculations



RAID LEVEL 0 - STRIPING

- RAID Level 0: Simplest form
- Stripe blocks across disk in a round-robin fashion
- Excellent performance and capacity
- Capacity
  - Capacity is equal to the sum of all disks
- Performance
  - R/W are distributed in round-robin fashion across all disks
- Reliability
  - No redundancy



RAID LEVEL 1 - MIRRORING

- RAID 1 tolerates HDD failure
- Two copies of each block across disks

Disk 0	Disk 1	Disk 2	Disk 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

Simple RAID-1: Mirroring (Keep two physical copies)

- RAID 10 (RAID 1 + RAID 0): Mirror then stripe data
- RAID 01 (RAID 0 + RAID 1): Stripe then mirror data

RAID 1 - EVALUATION

- Capacity: RAID 1 is expensive
  - The useful capacity is n/2
- Reliability: RAID-1 does well
  - Can tolerate the loss of disk(s)
  - Up to n/2 disk failures can be tolerated depending on which RAID fails
- Performance: RAID-1 is slow at writing
  - Must wait for writes to complete to all disk(s)
- **RAID is not a backup!**

RAID 5 – PARITY DISK

- Raid 5 – trades off space requirement for redundancy
- In a 5-disk array, you can only recover from the loss of 1 HDD
- Writes rotate across bit, distributing a parity bit
- To rebuild data blocks you only need data from 4 disks
  - Any drive can fail, as long as it is only 1
  - Only need:
    3 blocks + 1 parity block
    -or-
    4 blocks

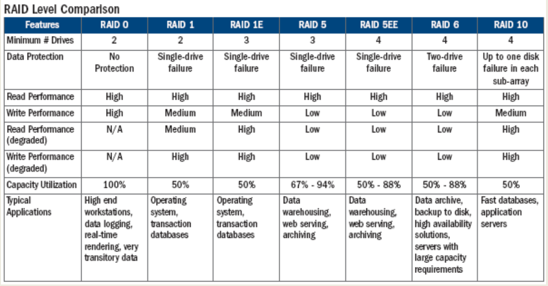
Disk 0	Disk 1	Disk 2	Disk 3	Disk 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

RAID-5 With Rotated Parity

RAID 5 – EVALUATION

- Capacity: Useful capacity is (n-1) disks
  - A HDD must be dedicated as a parity disk
- Performance
  - Writes are very slow: roughly = n/4
  - Reads are equivalent to a single disk
- Reliability
  - In RAID 5, a disk may fail, and the RAID keeps running
  - Rebuilds are slow !!!
  - Depending on disk size 8-24 hours is not unheard of
- RAID 6: Adds a second parity disk for increased resilience

RAID COMPARISON



RAID Level Comparison

Features	RAID 0	RAID 1	RAID 1E	RAID 5	RAID 5EE	RAID 6	RAID 10
Minimum # Drives	2	2	3	3	4	4	4
Data Protection	No Protection	Single-drive failure	Single-drive failure	Single-drive failure	Single-drive failure	Two-drive failure	Up to one disk failure in each sub-array
Read Performance	High	High	High	High	High	High	High
Write Performance	High	Medium	Medium	Low	Low	Low	Medium
Read Performance (degraded)	N/A	Medium	High	Low	Low	Low	High
Write Performance (degraded)	N/A	High	High	Low	Low	Low	High
Capacity Utilization	100%	50%	50%	67% - 94%	50% - 88%	50% - 88%	50%
Typical Applications	High end workstations, data logging, real-time rendering, very transitory data	Operating system, transaction databases	Operating system, transaction databases	Data warehousing, web serving, archiving	Data warehousing, web serving, archiving	Data archive, backup to disk, high availability solutions, servers with large capacity requirements	Fast databases, application servers

FILESYSTEMS

FILE SYSTEMS

- Implemented by the OS as pure software
- Provide:
  - Data structures: to describe disk content
  - Arrays of blocks, index-nodes, trees
  - Access methods: provides mapping for OS calls open(), read(), write(), etc.
  - Which structures are read? written? For each call?
  - How efficiently does the structure support file operations?
- Many available file systems (A-Z)

FILE SYSTEMS - 2

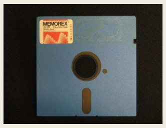
- Numerous file systems abound (A-Z)
- ADFA, AdvFS, AFS, AFS, AosFS, AthFS, BFS, BFS, Btrfs, CFS, CMDFS, CP/M, DDFS, DTFS, DOS 3.x, EAFS, EDS, ext, etx2, etx3, ext4, ext3cow, FAT, VFAT, FATX, FFS, Fossil, Files-11, Felx, HFS, HPFS, HTFS, IceFS, ISO 9660, JFS, JXFS, Lisa FS, LFS, MFS, Minix FS, NILFS, NTFS, NetWare FS, OneFS, OFS, OS-9, PFS, ProDOS, Qnx5fs, Qnx6fs, ReFS, ReiserFS, Reiser4, Reliance, Reliance Nitro, RFS, S51K, SkyFS, SFS, Soup (Apple), SpadFS, STL, TRFS, Tux3, UDF, UFS, UFS2, VxFS, VLIR, WAFL, XFS, FS, ZFS

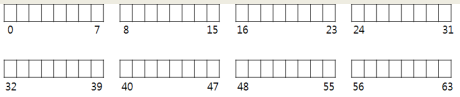
FILE SYSTEM ORGANIZATION

- Disk is divided into blocks
- Block size supported by most HDDs is 512 bytes
- Typical FS block size is 4 KB
- An instance of a file system is typically called a **partition**
- A single physical disk can have multiple partitions (file systems)

FILE SYSTEM EXAMPLE

- Consider a 64 block (4096KB block size) disk, aka. a 256 KB disk
- Legacy low density 5-¼" floppy had 160KB single side, 360KB double sided capacity



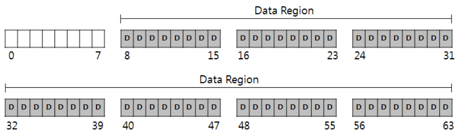


FILE SYSTEM STORAGE

- File system is stored using blocks on the disk
- This is considered a "reserved" region of the disk
- Corruption of the reserved region can destroy the file tables causing data on the disk to by unaddressable
- File system tracks:
  - Which blocks comprise a file
  - Where the blocks reside (are they contiguous?)
  - The size of files
  - The owner of files
  - File permissions (e.g. R/W/X)

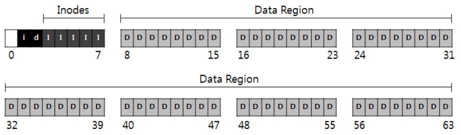
FILE SYSTEM LOCATION

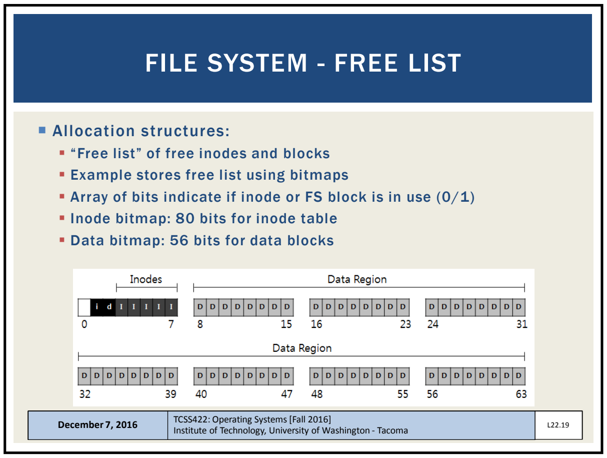
- Below the reserved region is at the front on a 64-block disk partition
- There are 56 blocks to track using "index-nodes" (inodes)



FILE SYSTEM EXAMPLES - 2

- Consider 256kb disk, with 56 free data blocks
- i-node size 256 bytes each
- 4KB block can contain 16 inodes
- Minimum of 4 – 4KB blocks required
- Here reserve 5 – 4KB blocks for files
- Provides some spare inodes

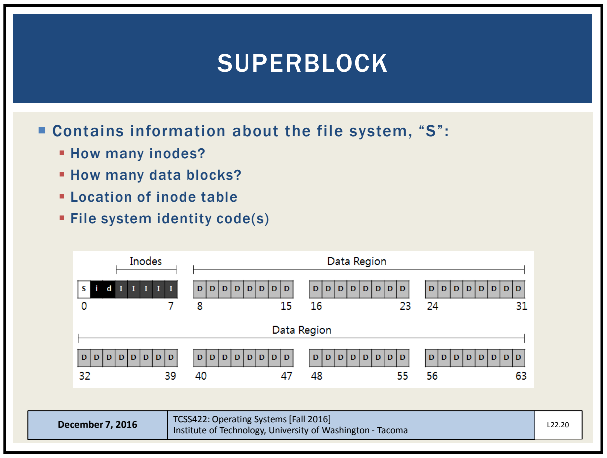




FILE SYSTEM - FREE LIST

- Allocation structures:
  - "Free list" of free inodes and blocks
  - Example stores free list using bitmaps
  - Array of bits indicate if inode or FS block is in use (0/1)
  - Inode bitmap: 80 bits for inode table
  - Data bitmap: 56 bits for data blocks

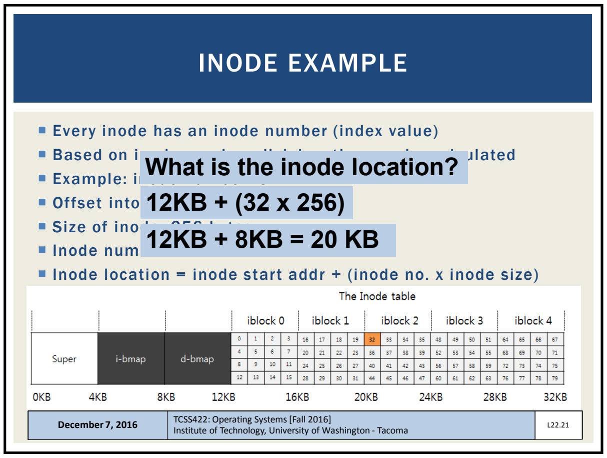
December 7, 2016 | TCSS422: Operating Systems [Fall 2016]
Institute of Technology, University of Washington - Tacoma | L22.19



SUPERBLOCK

- Contains information about the file system, "S":
  - How many inodes?
  - How many data blocks?
  - Location of inode table
  - File system identity code(s)

December 7, 2016 | TCSS422: Operating Systems [Fall 2016]
Institute of Technology, University of Washington - Tacoma | L22.20



INODE EXAMPLE

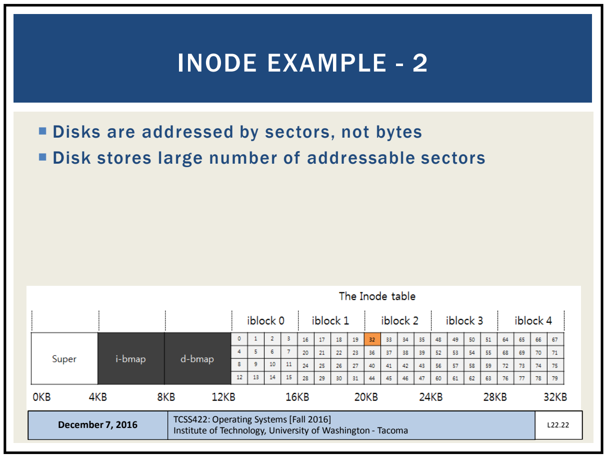
- Every inode has an inode number (index value)
- Based on i ulated
- Example: i
- Offset into
- Size of ino
- Inode num
- Inode location = inode start addr + (inode no. x inode size)

What is the inode location?

12KB + (32 x 256)

12KB + 8KB = 20 KB

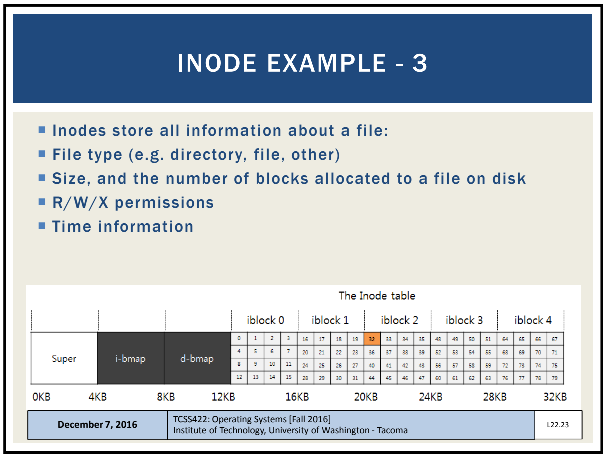
December 7, 2016 | TCSS422: Operating Systems [Fall 2016]
Institute of Technology, University of Washington - Tacoma | L22.21



INODE EXAMPLE - 2

- Disks are addressed by sectors, not bytes
- Disk stores large number of addressable sectors

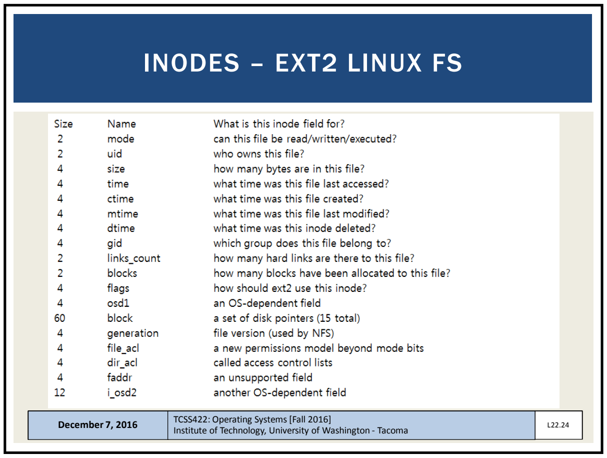
December 7, 2016 | TCSS422: Operating Systems [Fall 2016]
Institute of Technology, University of Washington - Tacoma | L22.22



INODE EXAMPLE - 3

- Inodes store all information about a file:
- File type (e.g. directory, file, other)
- Size, and the number of blocks allocated to a file on disk
- R/W/X permissions
- Time information

December 7, 2016 | TCSS422: Operating Systems [Fall 2016]
Institute of Technology, University of Washington - Tacoma | L22.23



INODES – EXT2 LINUX FS

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
4	faddr	an unsupported field
12	i_osd2	another OS-dependent field

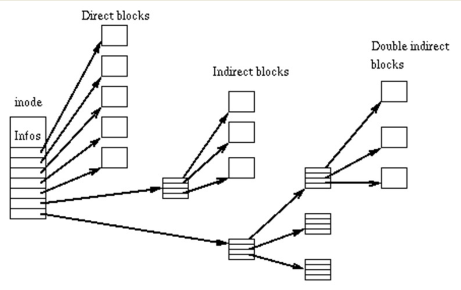
December 7, 2016 | TCSS422: Operating Systems [Fall 2016]
Institute of Technology, University of Washington - Tacoma | L22.24

MULTI-LEVEL INDEX

- Inodes use multi-level index
- First level: include 12-direct block pointers
- Second level: include 1 indirect block pointer
- Points to an entire block (4096 KB / 4 bytes) = 1,024 block pointers

- **Indirect pointer**: one level
- Maximum file size
- (12 + 1,024) * 4KB = (1,036 x 4KB) = 4,144 KB

MULTI-LEVEL INDEX - 2



MULTI-LEVEL INDEX - 3

- **Double indirect pointer**
- First level: include 12-direct block pointers
- Second level: include 1 indirect block pointer
- Third level: include 1 indirect block pointer
- Maximum file size:
- 12 + 1,024 + (1,024 x 1,024) * 4KB
- 1,049,612 x 4KB = 4,198,448 KB
  - > 4GB

- **Triple indirect pointer**
- Maximum file size:
- 12 + 1,024 + (1024^2) + (1024^3) * 4KB = > 4TB

EXTENTS

- Extents have a pointer with a stored length
- Each file has multiple extents
- A single extent would require contiguous file allocation

- In contrast to block pointers
- Extents conserve space better than multi-level indexes, but are less agile at representing file allocations scattered across the disk
  - Multi-level indexes excel for scattered file allocations

- File indexing presents a *space vs. flexibility* tradeoff

FILE INDEXING

- Multi-level indexing
  - Ext2, ext3

- Extents
  - Ext4 (default Ubuntu 16.04), XFS (default CentOS 7)
  - NTFS, Btrfs (b-tree fs)

- Exhaustive file systems feature comparison
  - https://en.wikipedia.org/wiki/Comparison_of_file_systems

COMMON FILE CHARACTERISTICS

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 ~50% full
Directories are typically small	Many have few entries; most have 20 or fewer

File System Measurement Summary

DIRECTORIES

- Directory contains file name and i number (index)
- Extra files for the **parent dir** and **pwd**
- Can store dirs as linear list, often stored in inodes
- XFS uses B-trees to eliminate sequential search of filenames for duplicates when creating a new file

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

on-disk for dir

FILE I/O - READ

- Consider reading a file called "/foo/bar"
- Traverse starting at root "/" (inumber = 2) to find file
- Read each inode to dereference file block location on disk

FILE I/O – READ OPERATIONS

- 3 block file: 11 reads, 3 writes (last access 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					
								read		
					write					
read()					read					
									read	
					write					
read()					read					
										read
					write					

File Read Timeline (Time Increasing Downward)

FILE I/O - WRITE

- At least Five I/Os to update an existing file
  - one to read the data bitmap
  - one to write the bitmap (to reflect its new state to disk)
  - two more to read and then write the inode
  - one to write the actual block itself.

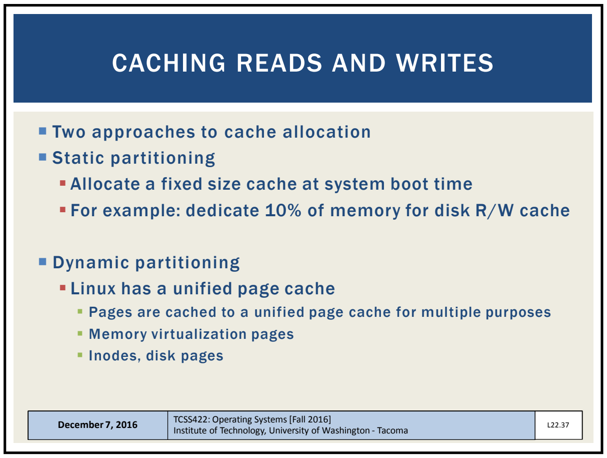
FILE I/O – WRITE - 2

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

File Creation Timeline (Time Increasing Downward)

FREE SPACE MANAGEMENT

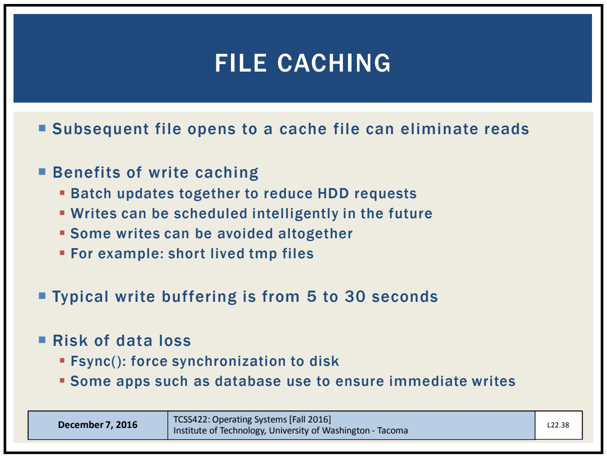
- Free Lists
  - Linked list of free blocks
  - Head node tracks first free block, each subsequent block is linked with a pointer
  - Bitmaps
    - Bit-wise arrays of free blocks
  - B-trees (XFS)
    - Represents free list in a more compact form, with better search performance
- Free list design impacts efficiency of finding free blocks



CACHING READS AND WRITES

- Two approaches to cache allocation
- Static partitioning
  - Allocate a fixed size cache at system boot time
  - For example: dedicate 10% of memory for disk R/W cache
- Dynamic partitioning
  - Linux has a unified page cache
    - Pages are cached to a unified page cache for multiple purposes
    - Memory virtualization pages
    - Inodes, disk pages

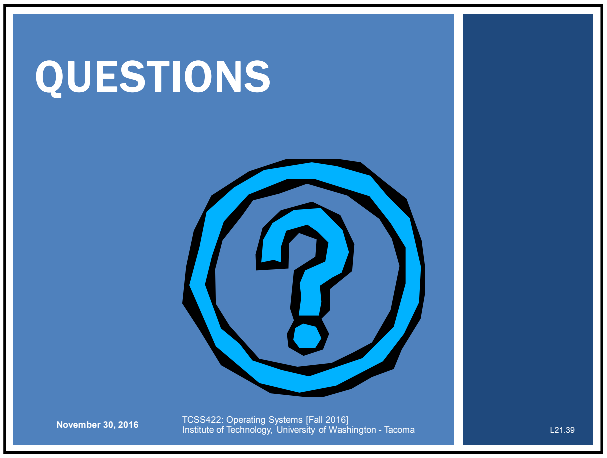
December 7, 2016 | TCSS422: Operating Systems [Fall 2016] Institute of Technology, University of Washington - Tacoma | L22.37



FILE CACHING

- Subsequent file opens to a cache file can eliminate reads
- Benefits of write caching
  - Batch updates together to reduce HDD requests
  - Writes can be scheduled intelligently in the future
  - Some writes can be avoided altogether
  - For example: short lived tmp files
- Typical write buffering is from 5 to 30 seconds
- Risk of data loss
  - Fsync(): force synchronization to disk
  - Some apps such as database use to ensure immediate writes

December 7, 2016 | TCSS422: Operating Systems [Fall 2016] Institute of Technology, University of Washington - Tacoma | L22.38



QUESTIONS

November 30, 2016 | TCSS422: Operating Systems [Fall 2016] Institute of Technology, University of Washington - Tacoma | L21.39