CSC369: Operating Systems

Fall 2014

Andrew Petersen

International Summer Research Projects

Summer research projects are an exciting opportunity to earn credit towards your degree while gaining hands-on lab experience with world-class researchers around the world.

There are spots available in Sweden (Lund), Hong Kong (CUHK), Wales (Cardiff), Singapore (NUS) and Saudi Arabia (KAUST) for Summer 2015.

All successful candidates will receive funding to cover airfare and some living expenses as a minimum.

There are established research pathways for students in Biology, Chemical & Physical Sciences, Geography, CCIT, Forensic Science, Computer Science + more.

Application deadline extended to Nov 28.

For more information, please attend one of the info sessions below or email international.utm@utoronto.ca

IEC International Education Centre

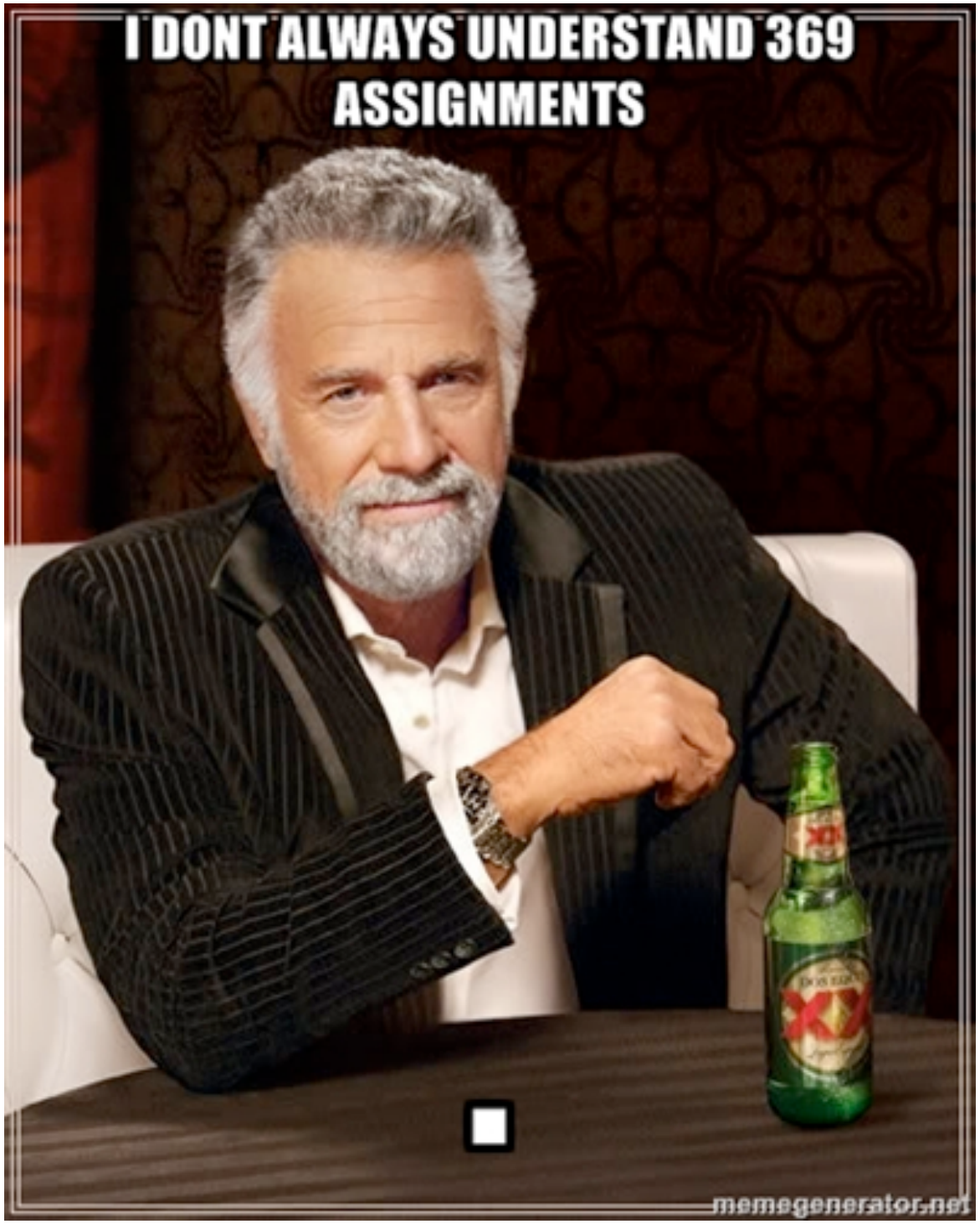
Info Sessions - Tuesday November 18	
10:00 – 11:00	DV 1160
2:00 – 3:00	DV 1158A

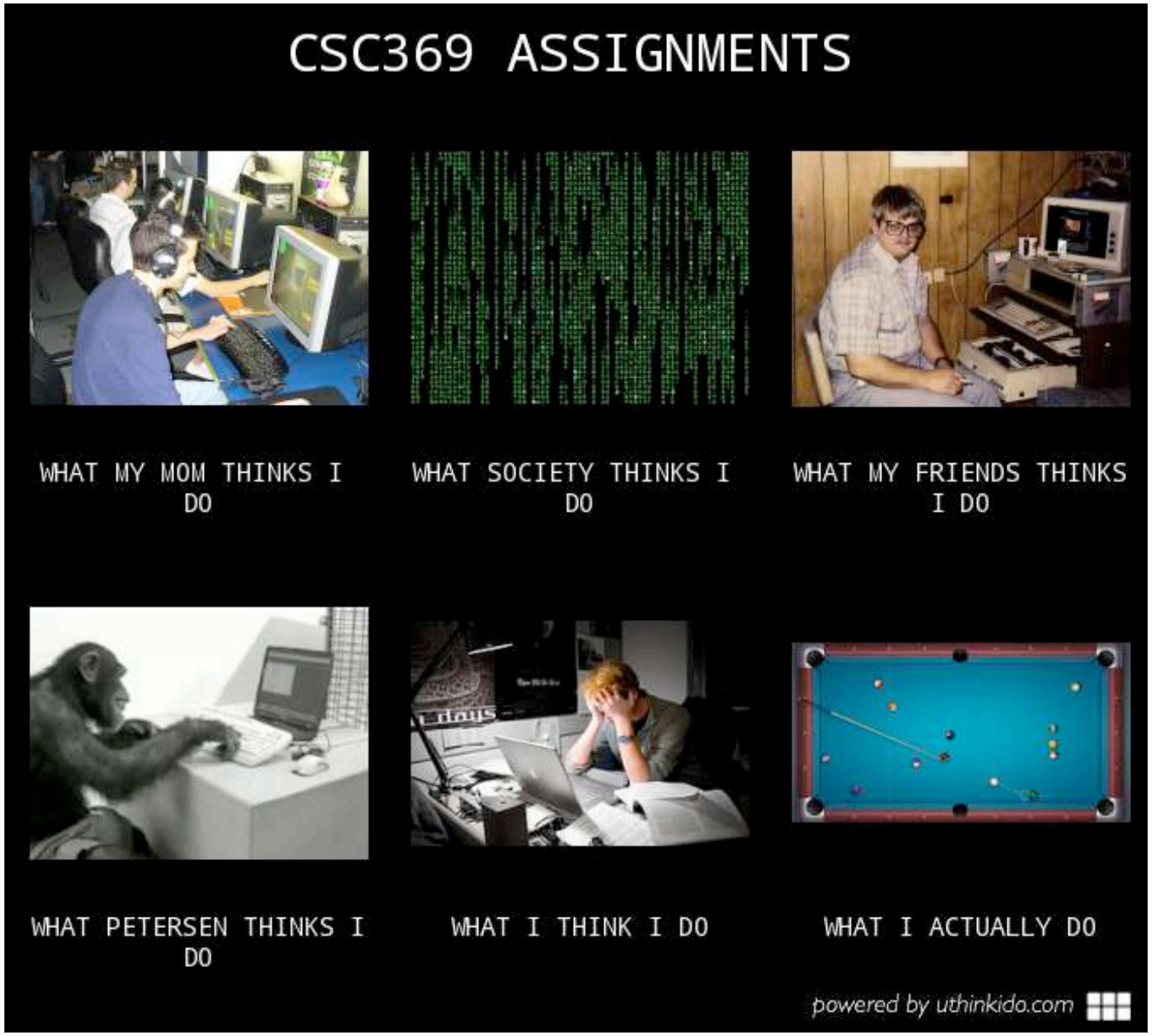
Anonymous Says ...

What is up with all the paragraph writing?

- Writing -- communicating your ideas clearly and succinctly -- is important.
- Perhaps more important than anything else we do.

Past A3 Commentary ...


I DONT ALWAYS UNDERSTAND 369 ASSIGNMENTS
memegenerator.net



Anonymous, please take a bow.

Outline

- Inodes
- Traversing Paths
- Disk layout

Making Sense of Ext2

- The file system sees storage as linear array of blocks
  - Each block has a logical block number (LBN)

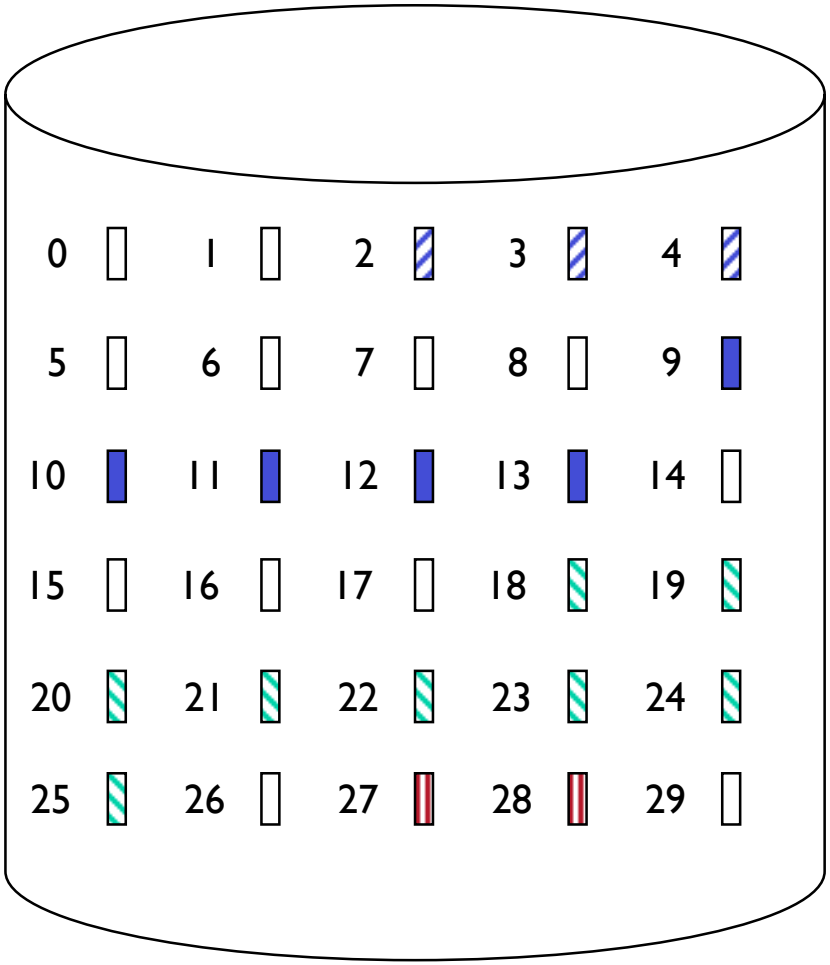
Directory Implementation

- Option 1: List
  - Simple list of file names and pointers to data blocks
  - Requires linear search to find entries
  - Easy to implement, slow to execute
    - And directory operations are frequent!
- Option 2: Hash Table
  - Create a list of file info structures
  - Hash file name to get a pointer to the file info structure in the list
  - Hash table takes space

Disk Layout Strategies

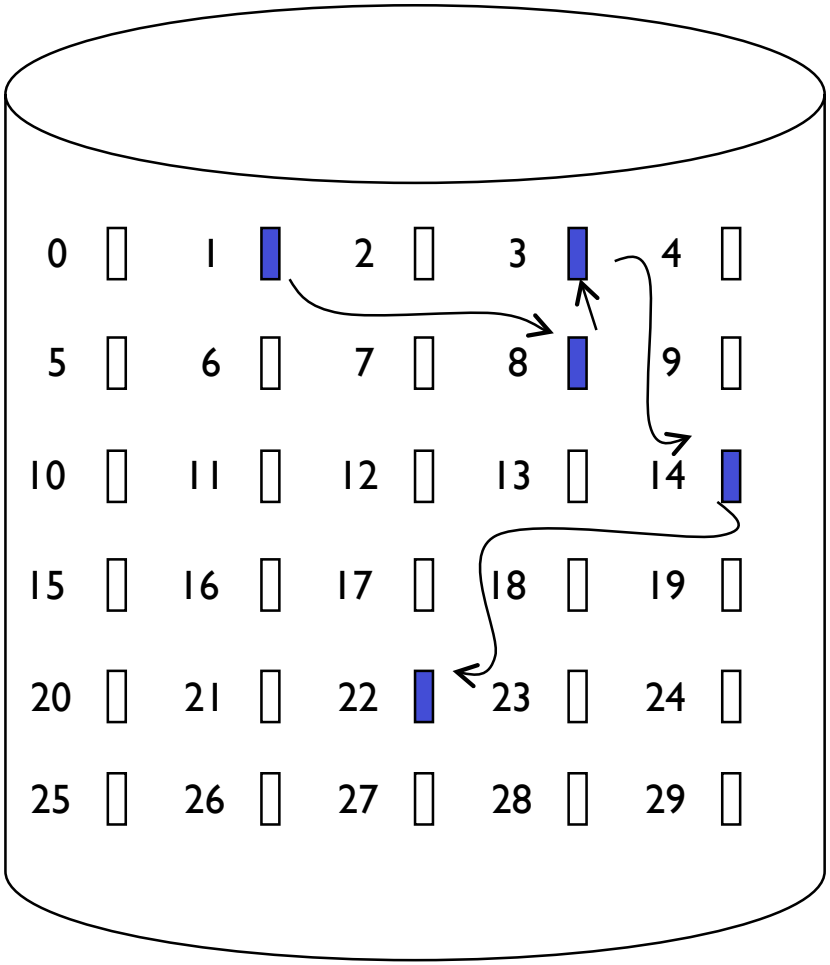
- Files often span multiple disk blocks (think "processes often span multiple pages ...")
- How do you find all of the blocks for a file?
  1. Contiguous allocation
     - Fast, simplifies directory access and allows indexing
     - Inflexible, causes external fragmentation, requires compaction
  2. Linked, or chained, structure
     - Each block points to the next, directory points to the first
     - Good for sequential (streaming) access, bad for all others
  3. Indexed structure (kind of like address translation)
     - An "index block" contains pointers to many other blocks
     - Handles random access better, still good for sequential
     - May require multiple, linked index blocks

Contiguous Allocation



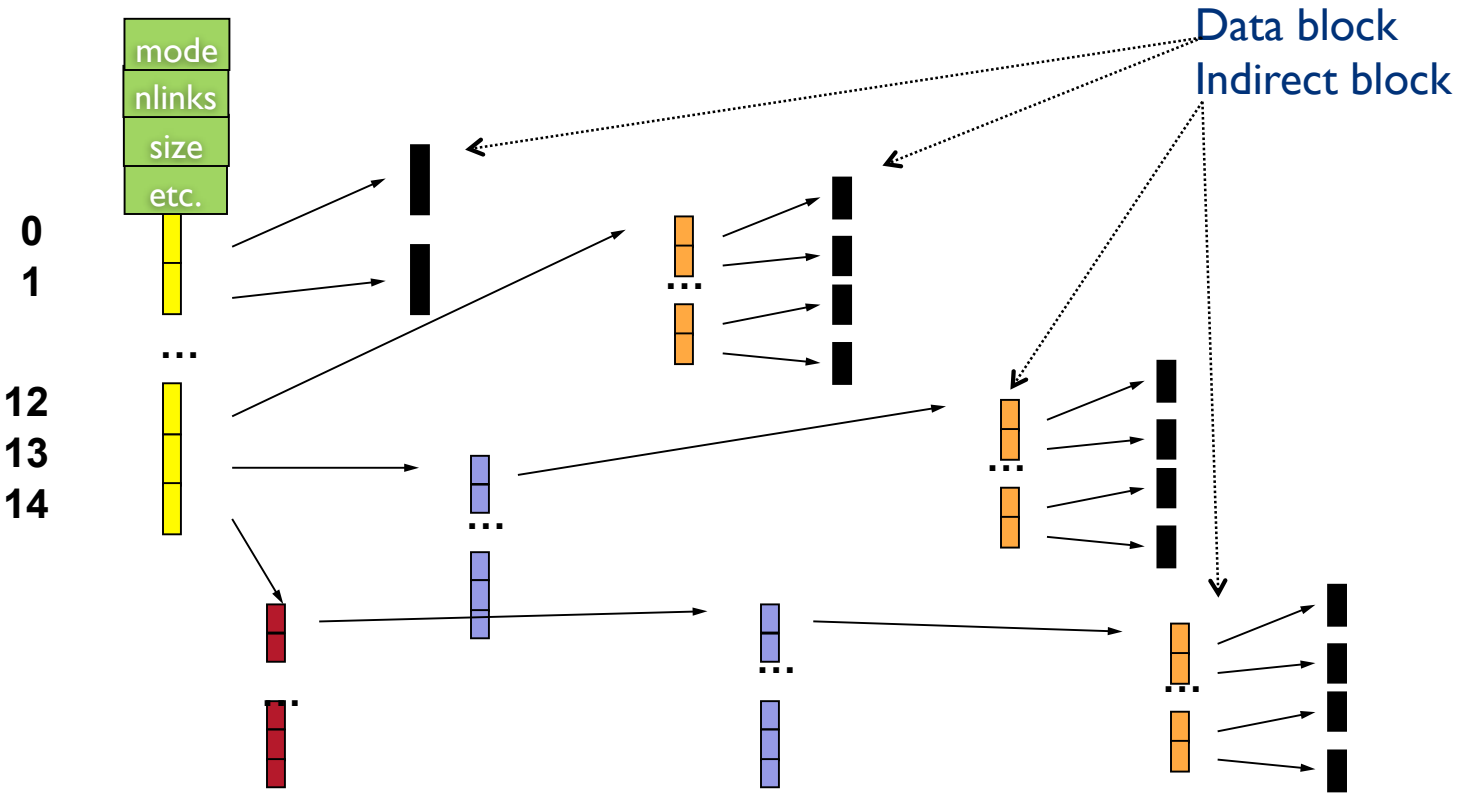
File Name	Start Blk	Length
File A	2	3
File B	9	5
File C	18	8
File D	27	2

Linked Allocation



File Name	Start Blk	Last Blk
…	…	…
File B	1	22
…	…	…

Indexed: UNIX Inodes



Indexed Allocation: Unix Inodes

- Unix inodes implement an indexed structure for files
- All file metadata is stored in an inode
  - Unix directory entries map file names to inodes
- Each inode contains 15 block pointers
  - First 12 are direct block pointers
    - Disk addresses of first 12 data blocks in file
  - The 13th is a single indirect block pointer
    - Address of block containing addresses of data blocks
  - Then the 14th is a double indirect block pointer
    - Address of block containing addresses of single indirect blocks
  - Finally, the 15th is a triple indirect block pointer

Outline

- Inodes
- Traversing Paths
- Disk layout

Path Name Translation

- Let's say you want to open "/one/two/three"
- What does the file system do?
  - Open directory "/" (the root, well known, can always find)
  - Search for the entry "one", get location of "one" (in directory entry)
  - Open directory "one", search for "two", get location of "two"
  - Open directory "two", search for "three", get location of "three"
  - Open file "three"

- Systems spend a lot of time walking directory paths
  - This is why open is separate from read/write
  - OS will cache prefix lookups for performance

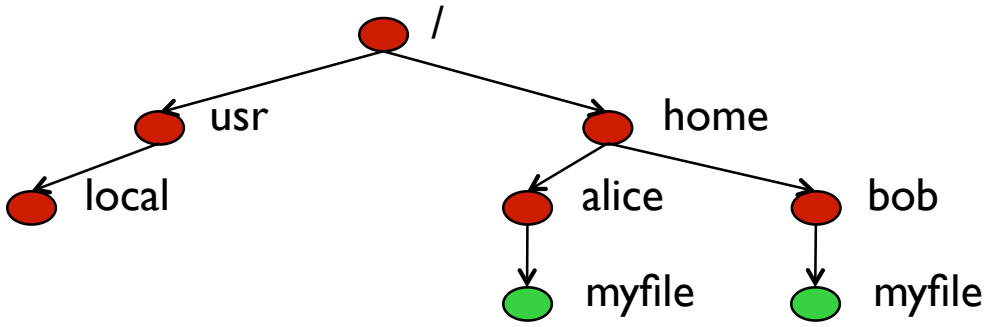
Inodes and Path Search

- Unix Inodes are not directories
  - Directories are files that impose logical organization
  - Inodes are structures that impose physical organization --where on the disk the blocks for a file are placed
- Directory entries map file names to inodes
  - To open "/one", use the master block to find the inode for "/" on disk and read the inode into memory
  - The inode allows us to find data block for directory "/"
  - Read "/", look for entry for "one"
  - This entry gives location of the inode for "one"
  - Read the inode for "one" into memory
  - The inode for "one" says where first data block is on disk
  - Read that block into memory to access the data in the file

Current Working Directory

- The current working directory is used as a base for specifying file names
  - Relative path or file names are specified with respect to current directory
  - Absolute names start from the root of directory tree
- Special names are used to navigate up and down the hierarchy: "." == current directory, ".." == parent

- If the working directory is */home/bob*, how can we refer to *myfile* in alice's directory?



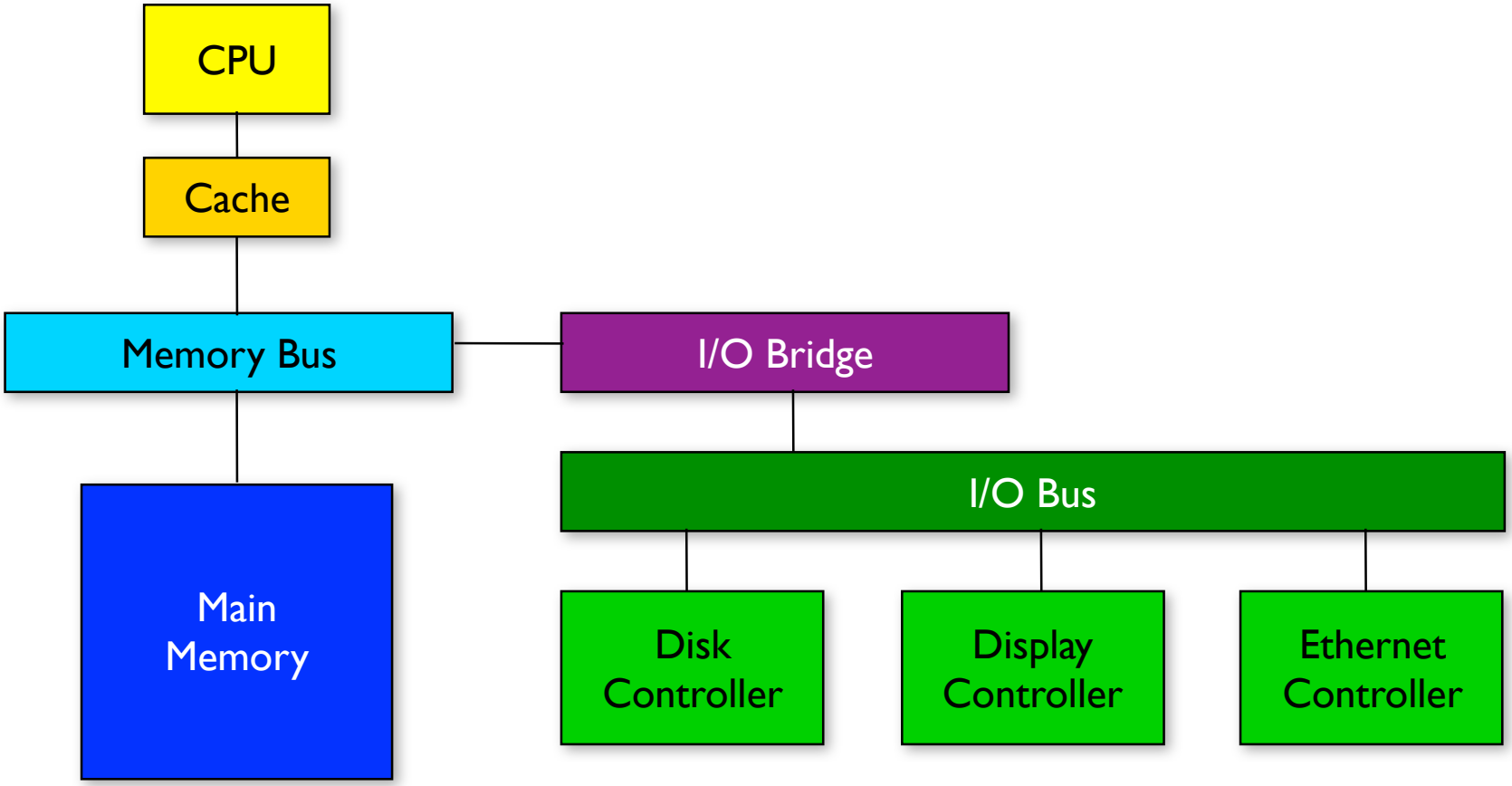
Outline

- Inodes
- Traversing Paths
- Disk layout

So … Why are Disks Slow?

- We have asserted that writes to disk are slow
  - Specifically, writes to multiple small locations are worse than a single write
  - This is a result of the disk technology being used!
- This means that he policy decisions we're making are based on available technology
  - … and the technology is changing

I/O Diagram


CPU
Cache
Memory Bus
I/O Bridge
Main Memory
I/O Bus
Disk Controller
Display Controller
Ethernet Controller

Secondary Storage

- Secondary storage is usually:
  - Anything outside of "primary memory"
  - Anything that does not permit direct instruction execution or data fetch via machine load/store instructions
- Characteristics
  - It's large – hundreds of megabytes, gigabytes, terabytes
  - It's cheap - 500GB internal for $110 (CAN$)
  - It's persistent – data survives loss of power
  - It's slow – milliseconds to access (is a millisecond slow?)

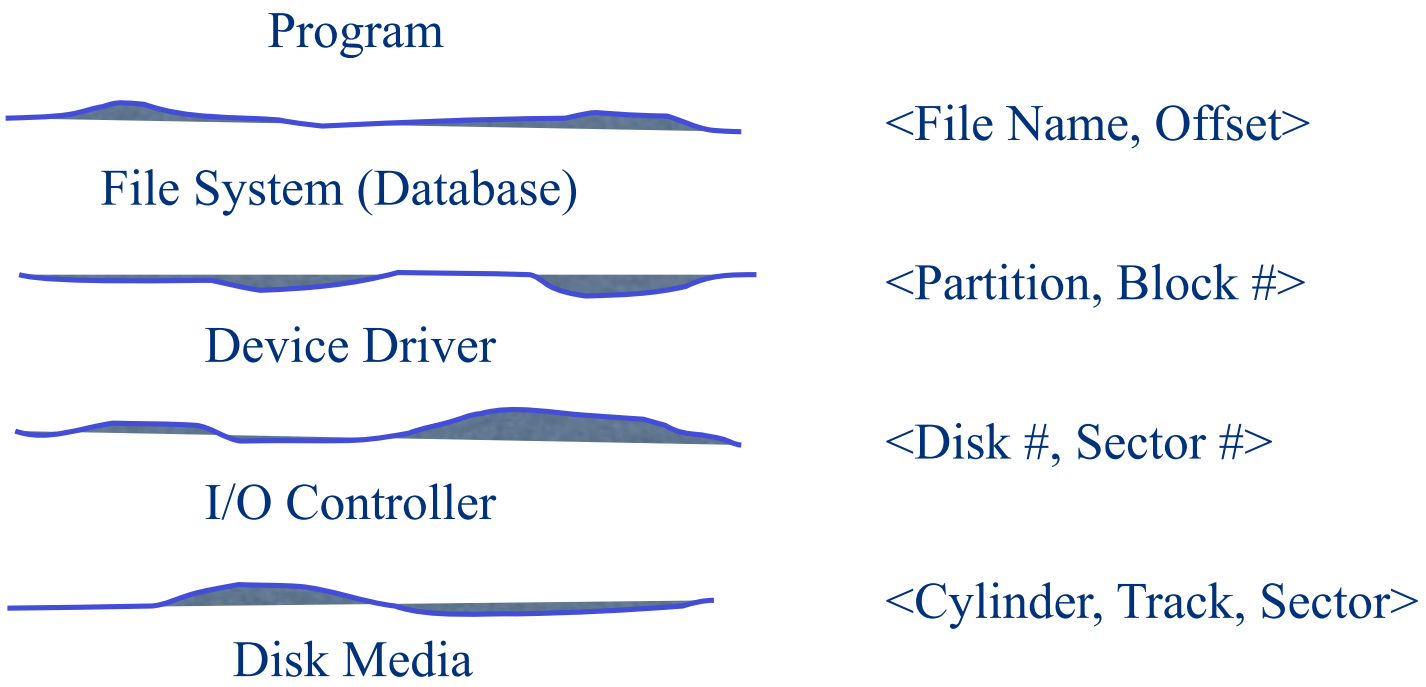
Secondary Storage Devices

- Drums: ancient history
- Magnetic disks
  - Fixed and removable (floppy -- also ancient history)
- Optical disks
  - Write-once, read-many (CD-R, DVD-R)
  - Write-many, ready-many (CD-RW)
- Solid state devices
  - These do not have many of the limitations that have caused OS policy to be as it is!
- To understand how we got to where we are, focus on the use of magnetic disks for implementing secondary storage

Disk Abstraction

- Disks are messy physical devices:
  - Errors, bad blocks, missed seeks, etc.
- The job of the OS is to hide this mess from higher level software
  - Low-level device control (initiate a disk read, etc.)
  - Higher-level abstractions (files, databases, etc.)
- The OS may provide different levels of disk access to different clients
  - Physical disk (surface, cylinder, sector)
  - Logical disk (disk block #)
  - Logical file (file block, record, or byte #)

Software Interface Layers

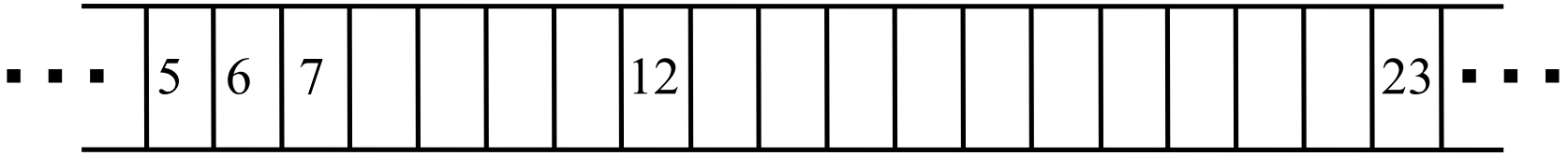


- Each layer abstracts details below it for the layers above it
  - Naming and address mapping
  - Caching and request transformations

Disk Interaction

- Specifying disk requests requires a lot of info:
  - Cylinder #, surface #, track #, sector #, transfer size…
    - Older disks required the OS to know all disk parameters
- Modern disks are more complicated
    - Not all sectors are the same size, sectors are remapped, etc.
- Current disks provide a higher-level interface (SCSI, for example)
  - Kind of like an OS on the device, so disk parameters are hidden from system OS
  - The disk exports its data as a logical array of blocks [0…N]
    - Disk maps logical blocks to cylinder/surface/track/sector
  - Only need to specify the logical block # to read/write

Common Storage Device Interface

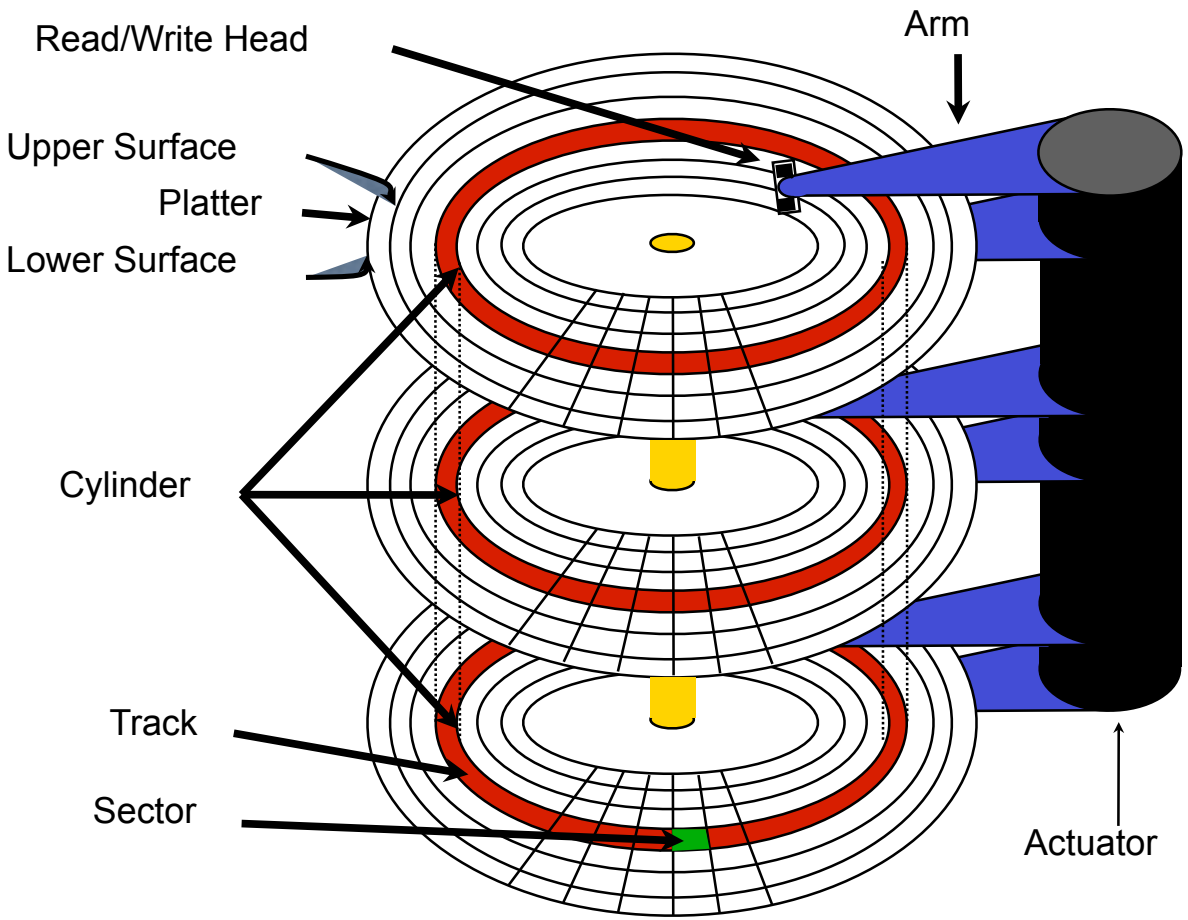


OS's view of storage device

Storage exposed as linear array of blocks

- Common block size: 512/1024/4096 bytes
- Number of blocks: device capacity / block size

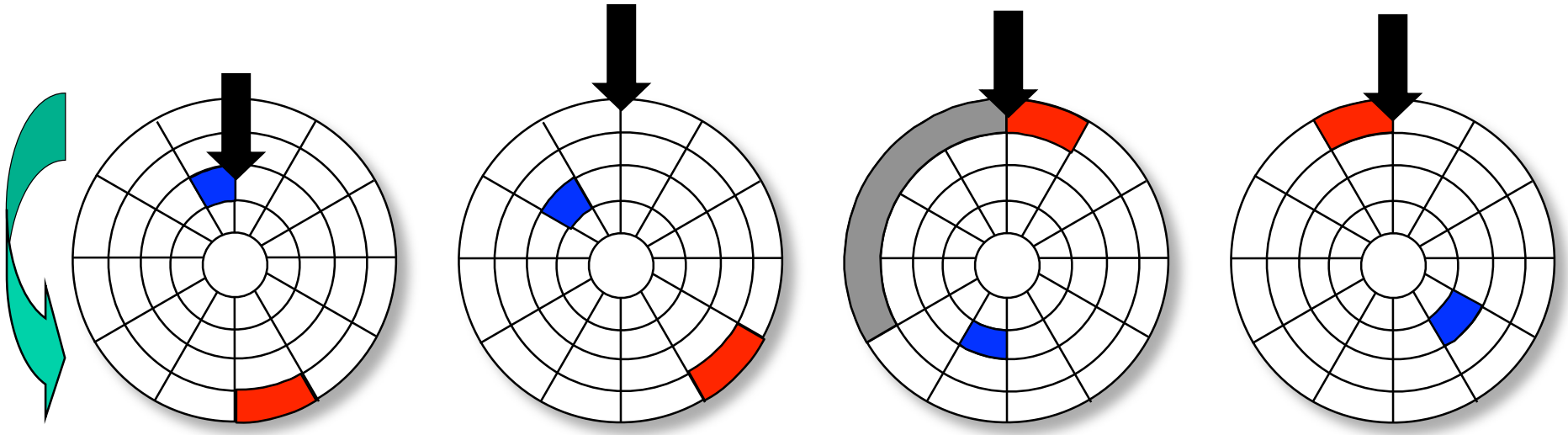
Disk Components



Disk Performance

- Disk request performance depends upon a number of steps
  - Seek – moving the disk arm to the correct cylinder
    - Depends on how fast disk arm can move (increasing very slowly)
  - Rotation – waiting for the sector to rotate under the head
    - Depends on rotation rate of disk (increasing, but slowly)
  - Transfer – transferring data from surface into disk controller electronics, sending it back to the host
    - Depends on density (increasing quickly)

Disk Latency Example



After BLUE read | Seek for RED | Rotational latency | After RED read

Disk Scheduling

- The OS tries to schedule disk requests that are queued waiting for the disk
  - FCFS (no optimization)
    - Reasonable when load is low
    - Long waiting times for long request queues
  - SSTF (shortest seek time first)
    - Minimizes arm movement (seek time), maximizes request service rate
    - Favors middle blocks
  - SCAN (elevator), C-SCAN
    - Service requests in one direction until done, then reverse
    - C-SCAN only goes in one direction (like a typewriter)

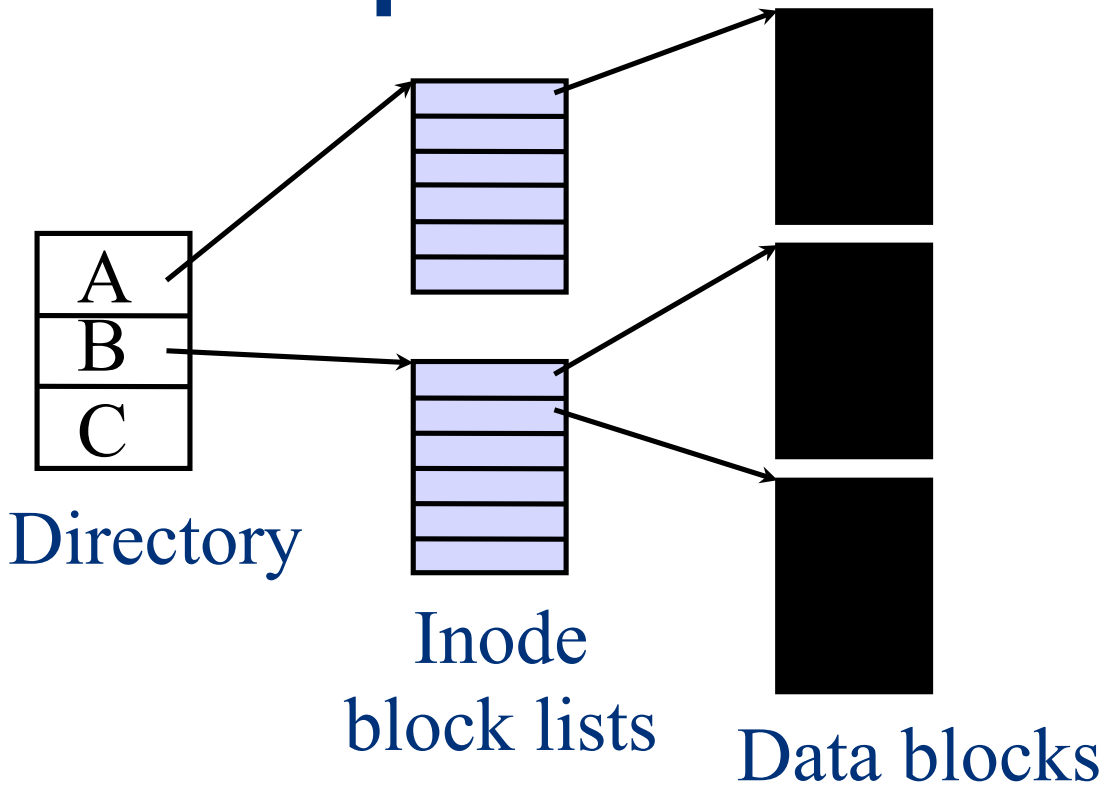
Disk Scheduling, continued

- LOOK / C-LOOK
  - Like SCAN/C-SCAN but only go as far as last request in each direction (not full width of the disk)
- In general, unless there are request queues, disk scheduling does not have much impact
  - Important for servers, less so for PCs
- Modern disks often do the disk scheduling themselves
  - Disks know their layout better than OS, can optimize better

Optimizing Disk Performance

- Done best where disk management happens
- High-level disk characteristics yield two goals:
  - Closeness
    - Reduce seek times by placing related data close
    - Benefits can be in the factor of 2 range
  - Amortization
    - Amortize each positioning delay by fetching lots of data
    - Benefits can reach into the factor of 10 range
    - Ex: prefetching

Inodes: Indirection & Independence



- File size grows dynamically, allocations are independent
- Difficult to achieve closeness and amortization

Original Unix File System

- The file system sees storage as linear array of blocks
  - Each block has a logical block number (LBN)

- Simple, straightforward implementation
  - Easy to implement and understand
  - But very poor utilization of disk bandwidth (lots of seeking)

Data and Inode Placement

Original Unix FS had two placement problems:

1. Data blocks are allocated randomly in aging file systems
   - Blocks for the same file allocated sequentially when FS is new
   - As FS "ages" and fills, need to allocate into blocks freed up when other files are deleted
   - Problem: Deleted files are essentially randomly placed, so new allocations are scattered
2. Inodes are allocated far from blocks
   - All inodes at beginning of disk, far from data
   - Traversing file name paths, manipulating files, directories requires going back and forth from inodes to data blocks

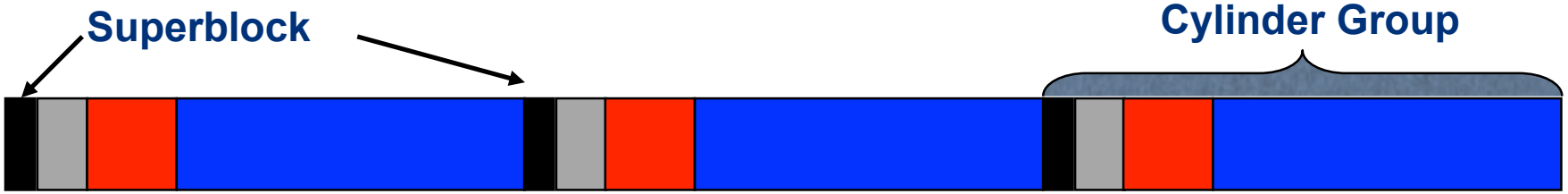
Both of these problems generate many long seeks

FFS

- BSD Unix did a redesign (early-mid 80s?) that they called the Fast File System (FFS)
  - Improved disk utilization, decreased response time
  - McKusick, Joy, Leffler, and Fabry, ACM TOCS, Aug. 1984
- Now the FS from which all other Unix FS's have been compared
- Good example of being device-aware for performance

Cylinder Groups

- BSD FFS addressed placement problems using the notion of a cylinder group (allocation group)
  - Disk partitioned into groups of cylinders
  - Data blocks in same file allocated in same cylinder group
  - Files in same directory allocated in same cylinder group
  - Inodes for files allocated in same cylinder group as file data blocks



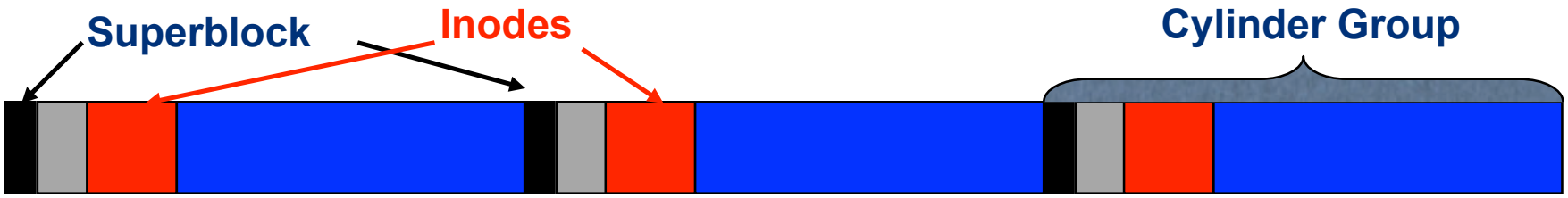
Cylinder group organization

Cylinder Groups

- Allocation in cylinder groups provides closeness
  - Reduces number of long seeks
- Free space requirement
  - To be able to allocate according to cylinder groups, the disk must have free space scattered across cylinders
  - 10% of the disk is reserved just for this purpose
    - Only used by root – this is why it is possible for "df" to report >100%
  - If preferred cylinder group is full, allocate from a "nearby" group

FFS: Consistency Issues

- Inodes: fixed size structure stored in cylinder groups



- Metadata updates should be atomic operations:
  - Write newly allocated inode to disk before its name is entered in a directory.
  - Remove a directory name before the inode is deallocated
  - Write a deallocated inode to disk before its blocks are placed into the cylinder group free list.

Solving Consistency Problems

- If the server crashes during any of these operations, then the file system is in an inconsistent state.
- Solution:

  Keep a log of updates to enable roll-back or roll-forward.

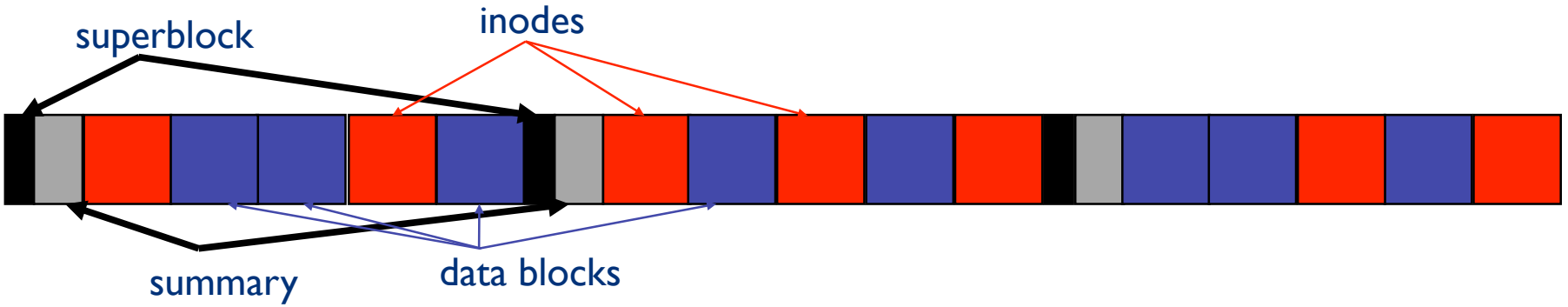
Are Reads the Problem?

- Observation: Most of the reads that go to the disk are the first read of a file. Subsequent reads are satisfied in memory by the file buffer cache.

- There is no performance problem with reads (except the first). Writes are the problem.
- Writes are not well-clustered, as they include modifying inodes and data blocks.

Log Structured File System (LFS)

- Developed by Ousterhout in 1989
- Idea: Write ***all*** file system data in a continuous log
- Uses inodes and directories from FFS
- Needs an inode map to find the inodes
- Cleaner reclaims space from overwritten or deleted blocks.



LFS Reads

- If the writes are easy, what happens to the reads?
- To read a file from disk:
  1. Read the superblock to find the index file
  2. Read the index file and find the inode-map
  3. Get the file's inode
  4. Use the inode as usual to find the file's data blocks
- Remember, we expect reads to hit in memory most of the time … after we open the file in the first place.

In practice?

- Cleaning turns out to be quite complicated
  - The "tail" of the log must always be moved up to give the "head" space to read
  - Depending on when cleaning happens there may be significant performance loss.
- LFS inspired the logging features of journaling file systems in use today. e.g., Ext3.
  - Old and new versions of data are kept on disk until update is complete
  - For undo capabilities write old data to log, and for redo, write new data to the log
  - A commit operation releases data from the log
  - Each file operation is a transaction

Summary

- Secondary storage is generally large, persistent, and relatively slow
- File system policies have been influenced by the limitations of the hardware implementation
  - Even after those limitations are removed, some of the policies remain
- We focused on magnetic disks, since those drove the development of advanced file allocation policies
  - The key: Accessing non-contiguous blocks is slow!
  - FFS and LFS are attempts to reduce non-contiguous access time