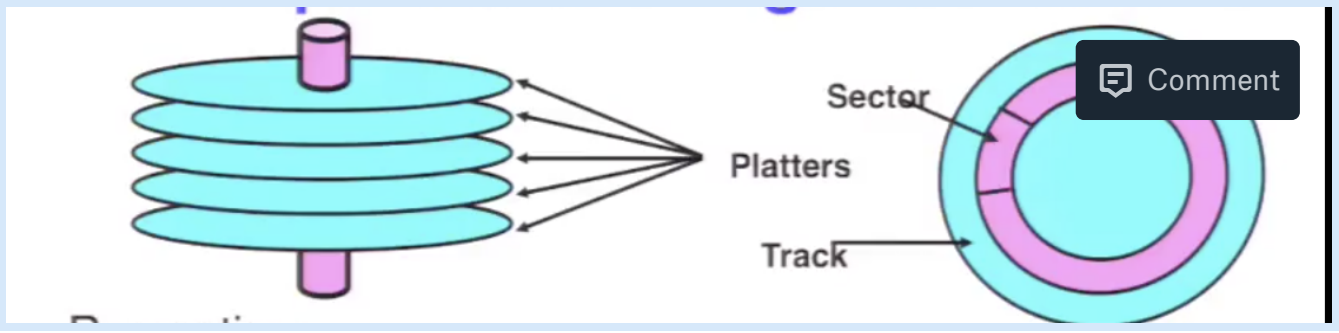
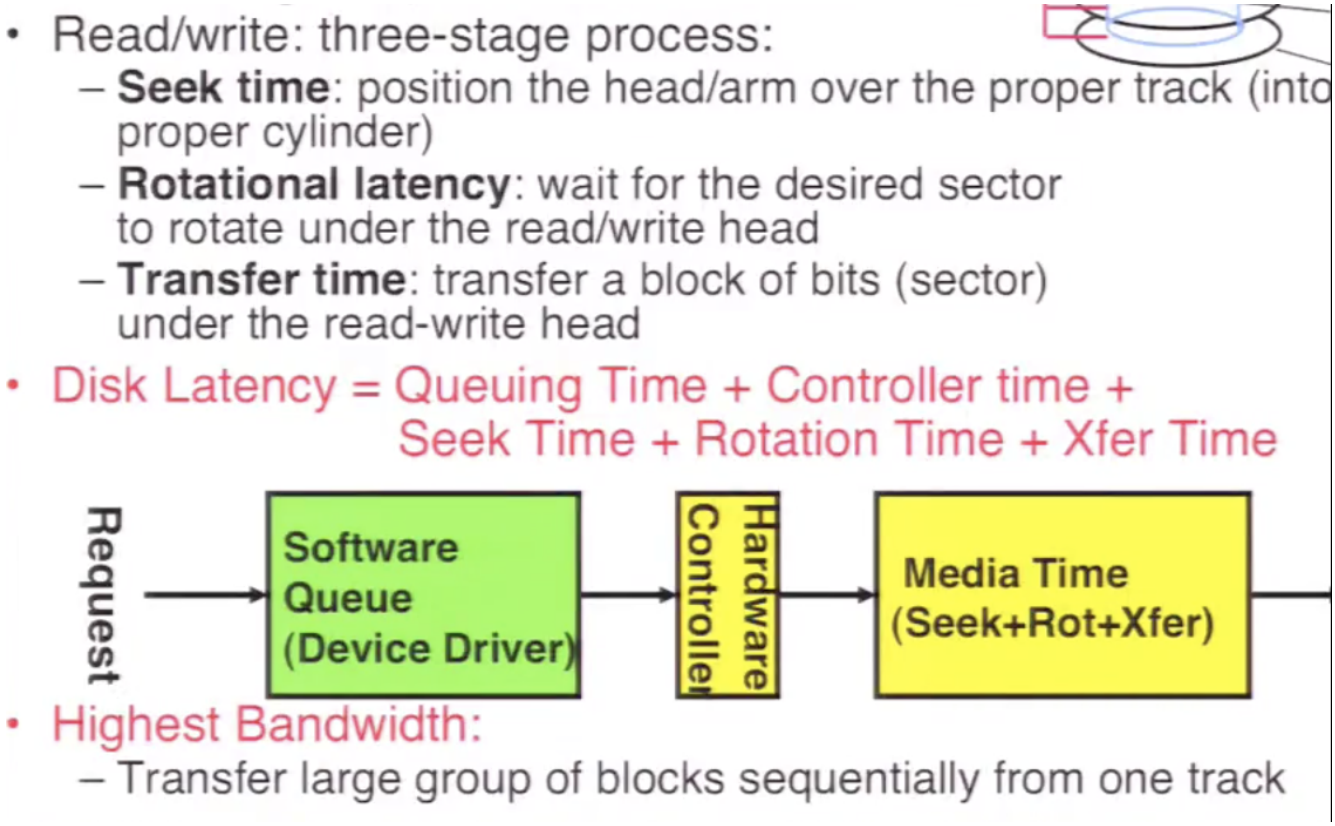
file systems & storage

hard disks



- note each platter has concentric tracks and each track is divided into many sectors. the sector is the basic addressable unit of read/write. the OS groups together sectors to form “blocks”. read/write happens in blocks.
- the platters are always spinning. we can go to any sector by moving the arm back and forth and waiting for the right sector to show up under the head.
- read/write mechanism:



- The RPM specified in the disks is speed of rotation. if they spin faster, then the rotational latency can be lesser.
- Note that modern SSDs are based on transistor storage with NAND gates as flash memory that can keep state even when powered off.

- Given the above architecture, the key is to minimize seek and rotational devices to increase throughput from disk by laying out data in a sequential manner (data continues into next sector and neighboring tracks) → basically make the head move as little as possible.
- given the above disks and a queue of requests to read from blocks (a group of sequential sectors), there are algorithms to service these requests called disk scheduling algorithms. These could be FIFO, closest block next, closest block in direction of motion of head next etc.
- But, the world is being rapidly taken over by SSDs. With SSDs, reads are really fast and all blocks are as close or as far. It is not a physical layout where the head has to physically to the block of interest.
- The downside is writes are about 10x the time of SSD reads because they have to burn the electrons thru an insulated layer (its what keeps the charge even when power is off).
- Also, you have to erase before writing to a spot because of the details of the materials of this technology. Erasing is even worse → 10x the write times! and because of this, the SSD controller (which is software that runs on the SSD microcontroller → also called firmware) periodically erases stuff and keeps empty blocks ready to be written to.
- SSD does have a small amount of on drive DRAM (which is what the processor's main memory is made of) to act as a buffer before transmission onto the bus or when received from the PCI bus.
- If you abruptly cut power supply to SSD there is a chance to corrupt the disk/lose data. best thing is to shut down the OS
- Since erases are so much more expensive than writes for SSDs, when you change the contents of a file, it actually gets written to a new block instead of erasing and writing the same block.
- The cost per gigabyte when this course was recorded was about $0.1 per GB for HDD, a magnitude more for SSD and a on more magnitude for DRAM (main memory)
- The cons with SSD is that it is more expensive and also that write throughput can be very dependent on workload and number of free blocks.

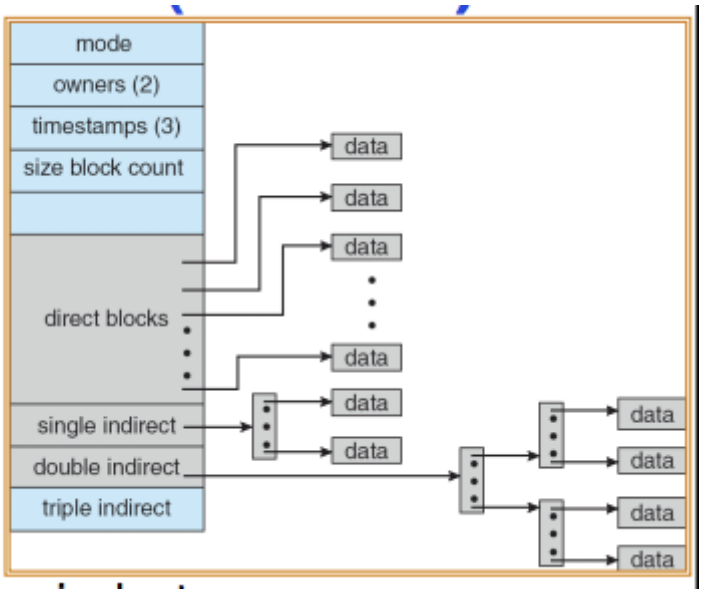
file systems

- it basically transforms the interface of disks and blocks into file and directories and provides on top of that naming functionality for files and directories, protection and permissions and reliability/durability
- a file is basically a collection of blocks, which are made of pages (pages is the SSD equivalent for sectors on HDDs)
- modern hard drives provide an interface to the OS' file system where it can address individual pages/sectors, identified by a unique key. this is the logical view for the OS.
- each file has a thing called a file header, which stores all the blocks that are relevant to that file. this is why lots of very small files might be bad.
- With HDDs, file systems had to think about how to layout files on sectors sequentially to enable fast access cuz all blocks of a file are often used with the other blocks. but with SSDs, this spatial property is gone!
- access patterns for files: (fast random access of disk is very important for virtual memory paging which needs to be super fast, so we shouldn't have to seek all bytes to get to the required bytes.)

Designing the File System: Access Patterns

- Sequential Access: bytes read in order ("give me the next X bytes, then give me next, etc.")
  - Most of file accesses are of this flavor
- Random Access: read/write element out of middle of array ("give me bytes i—j")
  - Less frequent, but still important, e.g., mem. page from swap file
  - Want this to be fast – don't want to have to read all bytes to get to the middle of the file
- Content-based Access: ("find me 100 bytes starting with JOSEPH")
  - Example: employee records – once you find the bytes, increase my salary by a factor of 2
  - Many systems don't provide this; instead, build DBs on top of disk access to index content (requires efficient random access)
  - Example: Mac OSX Spotlight search (do we need directories?)

- one of the most common file system in the world is the File Allocation Table (FAT) file system. Several billion devices support this including Android phones that use SD cards.
- the FAT file system is laid out so that the meta data table maps file names to the id of the first block in the table. the next few IDs are laid out sequentially. so with a spinning disk technology, the page/sector IDs are laid out next to each other but the seeks will not be sequential. for a SSD based technology like SD card this is fine since everything is equally far apart.
- the meta-data table → inode: An inode is a data structure on a filesystem on Linux and other Unix-like operating systems that stores all the information about a file except its name and its actual data.
- a different format of file header originally developed for the BSD (Berkeley Standard Distribution) filesystem: notice the mode bits, owners, created, modified & accessed timestamps, size block count and the layers of pointers to data blocks that get increasingly indirect.



- the part with the pointers to data blocks is called the inode. the top part that has metadata is the file metadata table (header)
- why have this increasing hierarchy of pointers to data blocks?

    - if our file is small, we will only use the first couple.
    - if our file is big we will use up the more indirect ones and scale to larger files.
- each file has a inumber that identifies that file → in addition to header + inode above.
- what are directories?
    - directories are just implemented as files under the hood
    - can contain other directories
- hard links are different names for the same file → they point to the same inumber/inode
    - if you delete a hard link, the other link will still work; the file system reference counts
- soft links/symbolic link are like “shortcut” pointers to files by encoding the file names
    - if you move around the underlying file name that is pointed to it or delete it the soft link won’t work
    - don’t have to reference count
    - and works across file systems since its just pointing to a path. hard link is file system dependent because it points to a inumber/inode
- name resolution in a file system involves given a path, traversing the directory files of each directory (remember directories are files) until we get to the leaf file. in a globally distributed file system like GFS, this can involve jumping between machines until we get to a unique logical file. (might be physically replicated under the hood)
- with physical drives its good to layout so that data that is likely to be accessed together (like inode block and data block) is laid out close to each other. SSD makes this easier cuz everything is equally close