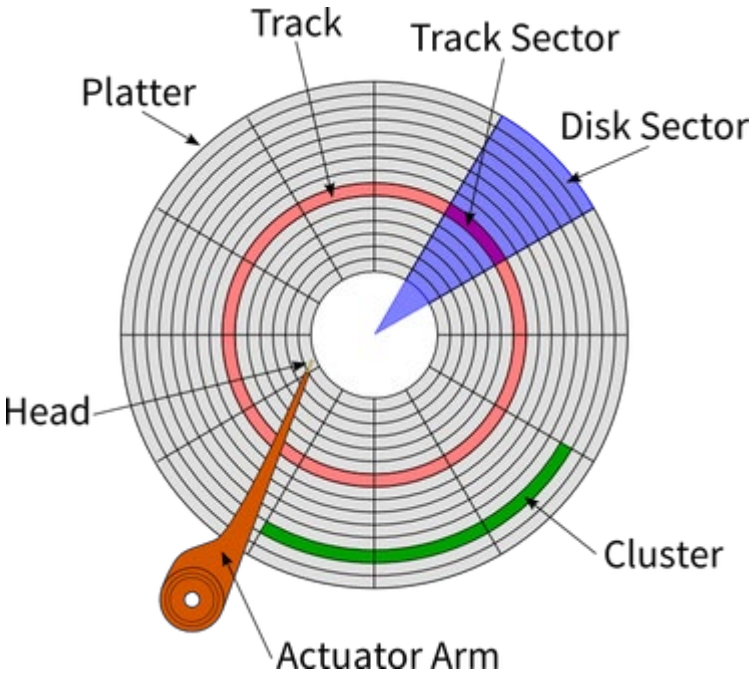
Difference between using OS / DBMS

I. Specialized Prefetching
II. Control over buffr replacement policy
III. Control over thread/process scheduling
    1. A lock convoy occurs when multiple threads of equal priority contend repeatedly for the same lock.
IV. Control over flushing data to disk

Disk Composition

I. Arm assembly moved in or out to position, with “Tracks” under it
II. Only one head reads/write at any one time
III. Block/Page size is a multiple of fixed sector size
IV. Time Composition
    A. Seek Time (i.e., Moving arms to position disk head on the correct track
        1. For small cylinders, the only need is to reposition the head instead of the arm movement. So the seek time of small # cylinders seek time is dominated by head position
    B. Rotational Delay (i.e., the time it takes the platters to rotate the required sector into position underneath the head
    C. Transfer Time
V. **Settle Time** (i.e., how long the head assembly takes to fully lock on to a track and stop oscillating, so it can decelerate and com to a rest for beginning reading)
    A. With increase disk track density, it requires a longer time to decelerate and start reading
VI. **Adjacent Blocks**
    A. “Next” means the next block that it will access after rotating for the degree it will do during settle time.
VII. **Disk Anatomy**
    A. A **track** is that portion of a disk which passes under a single stationary head during a disk rotation, a ring 1 bit wide. A **cylinder** is comprised of the set of tracks described by all the heads (on separate platters) at a single seek position. Each cylinder is equidistant from the center of the disk. A track is divided into **segments of sectors**, which is the basic unit of storage.

VIII.



IX. Suppose we have a 10000 RPM disk has 8 heads and 480 cylinders. It is divided into 120- cylinder zones with the cylinders in different zones containing 200, 240, 280, and 320 sectors. As- sume each sector contains 4096 bytes and a seek time between adjacent cylinders of 2 msec.
    A. To get the capacity of the disk, multiply the total number of sectors by the sector size. The total number of sectors is 120*200 + 120*240 + 120*280 + 120*320 or 124800, so the total disk capacity should be 124800 * 4096, or 511,180,800 bytes.
    B. To get the transfer rate, first, pick the cylinder zone with the largest number of sectors (320). This is the cylinder zone that will yield the greatest transfer rate. It's probably near the outside of the disk (since there's more surface area to work with).
    C. Now, multiply the number of sectors by the number of bytes per sector (320 * 4096) and you get 1310720. This is the number of bytes you can retrieve *per disk rotation.*
    D. The number of rotations per second is 10000/60, approximately 166.67. Multiply that by 1310720 bytes per rotation, and you get 218453333, or about 208 megabytes per second.
    E. Since you need 166.67 cylinders to get a full second's worth of data, you should subtract the amount of data that cannot be read while the head is seeking between cylinders (2 ms per seek).
X. https://www.ntfs.com/hard-disk-basics.htm

Implications on Memory Access

I. With growing capacity of memory, memory access will become the new bottleneck, and no longer uniform random access model (NUMA) can be assumed for memory (i.e., Time to access each piece of data is assumed to be the same)

Cache

II. **Translation lookaside buffer** refers to memory cache that is used to reduce the time taken to access user memory location
III. **Cache Line:** The unit of data transfer between the cache and main memory. Typically the cache line is 64 bytes. The processor will read or write an entire cache line when any location in the 64 byte region is read or written.
IV. **Performance Improved Factor**
    A. Cache Capacity
    B. Locality (Temporal and Spatial)
        1. Non-Random Access (Clustering to cache line, squeeze more operations into a cache line)
        2. Random Access
        3. Compress Data => Require less space in Cache => Need time to decompress

Cache Trees

I. **CSS Tree**
    A. Fixed number of fan-out and array size (i.e., for chile nodes)
    B. Slightly improvement in search performance than B+ tree
    C. NO child pointers, so only use half of the B+ tree space
II. **Why does the CSB+ tree (or full CSB+ tree) have a worse search performance than the CSS tree?**
    A. Because it allocates fixed sized arrays which may not be full, hence degrading performance.

Spatial Data Check

I. So each grid/layer stores all data and the approach essentially checks a cell on a specific level if it sufficiently overlaps with the query.