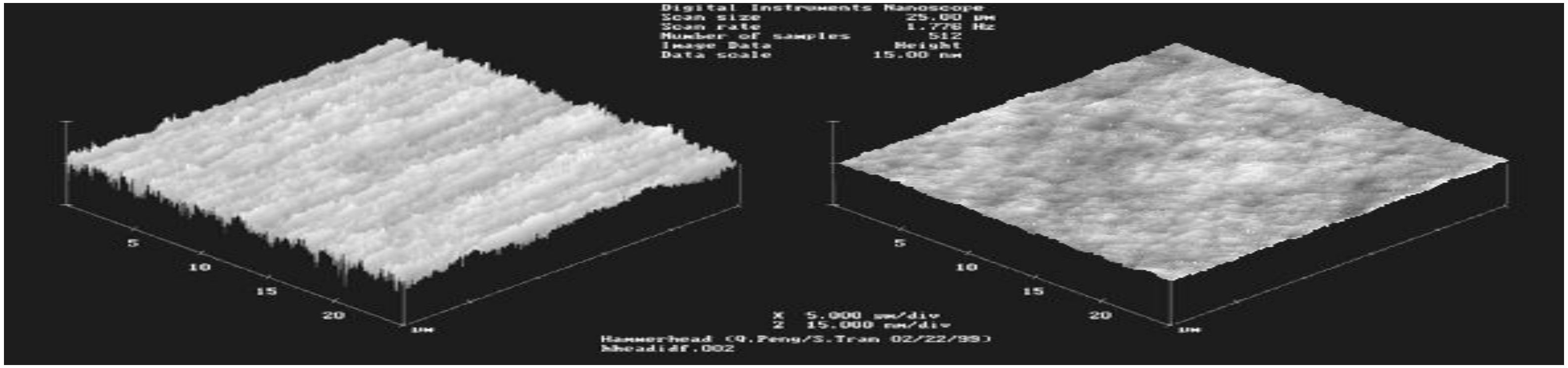
Data Model
in
Cache, Memory & Disk

By 执少（顾汉杰）
阿里云Nas文件存储团队

Agenda

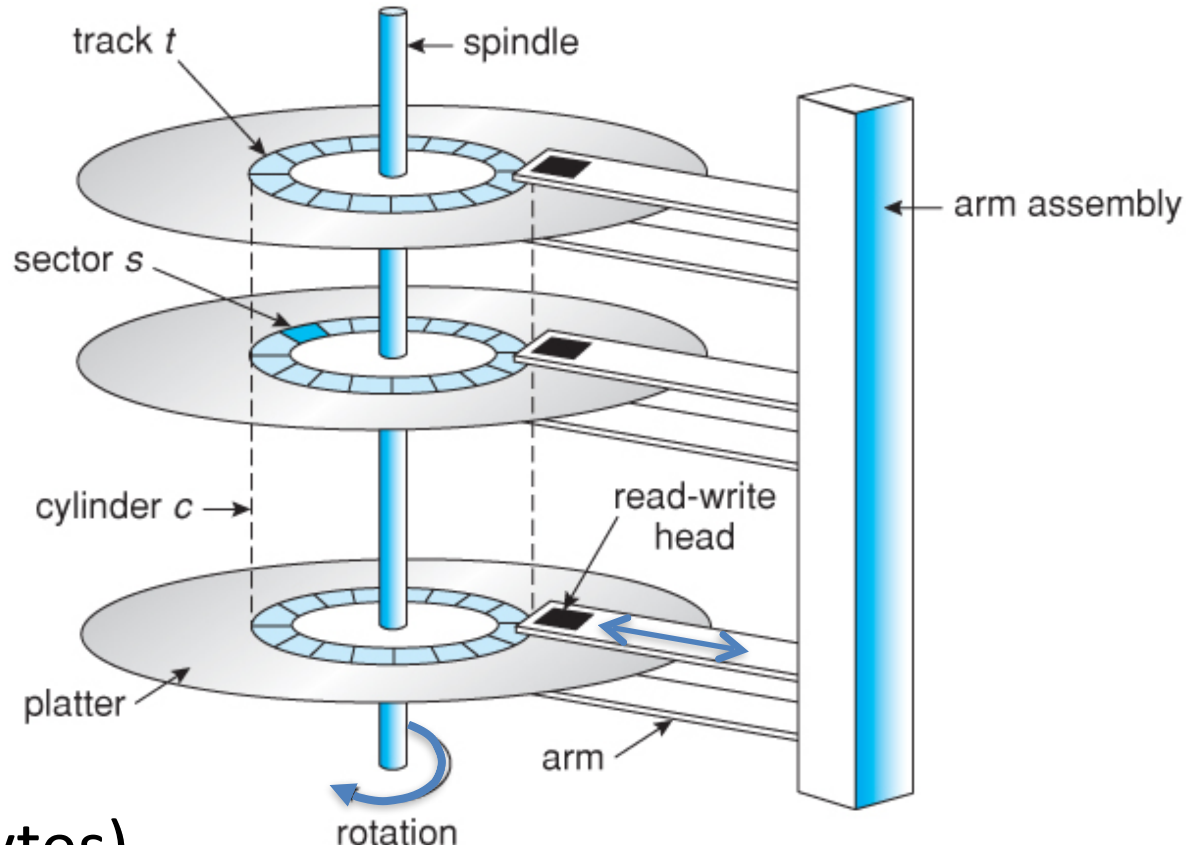
- How does disk work
- Data layout on disk
- CHS addressing
- File system on disk
- Readahead on disk
- Fragmentation on disk
- Storage model on disk(B-tree, B+tree, LSM)

Hard Disk


Digital Instruments Nanoscope
Scan size 25.00 µm
Scan rate 1.776 Hz
Number of samples 512
Image Data Height
Data scale 15.00 nm
5
10
15
20
µm
X 5.000 µm/div
Z 15.000 nm/div
Hammerhead (Q.Peng/S.Tran 02/22/99)
hheadidf.002

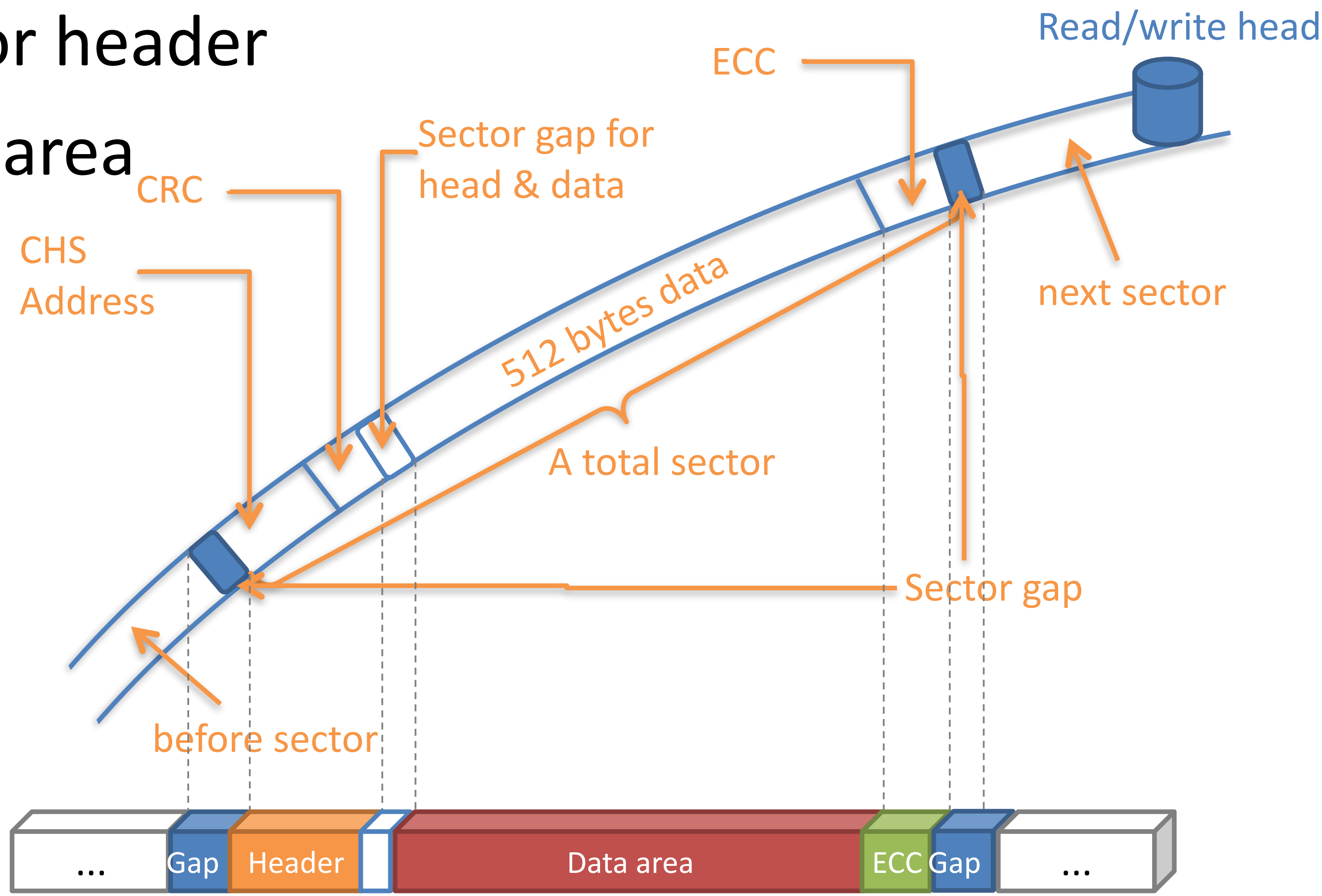
How to work?

- Components
  - platter
  - side
  - head
  - arm
- Some terms
  - Track
  - Cylinder
  - Sector (512Bytes)



Data layout on Sector

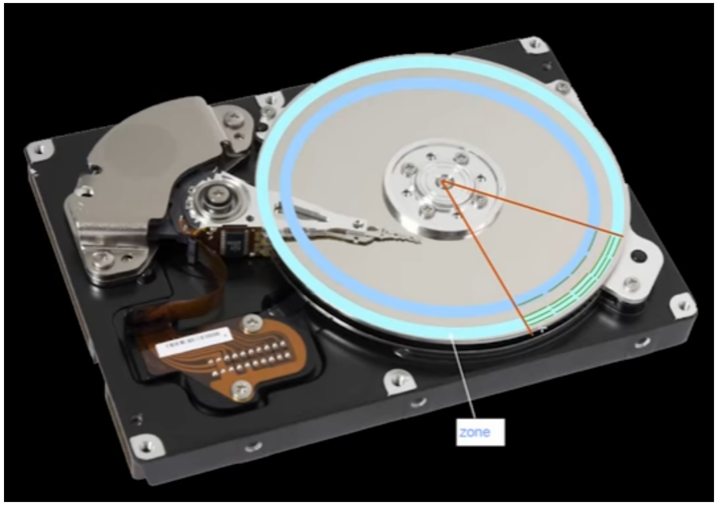
- Sector header
- Data area
- ECC
- Gaps

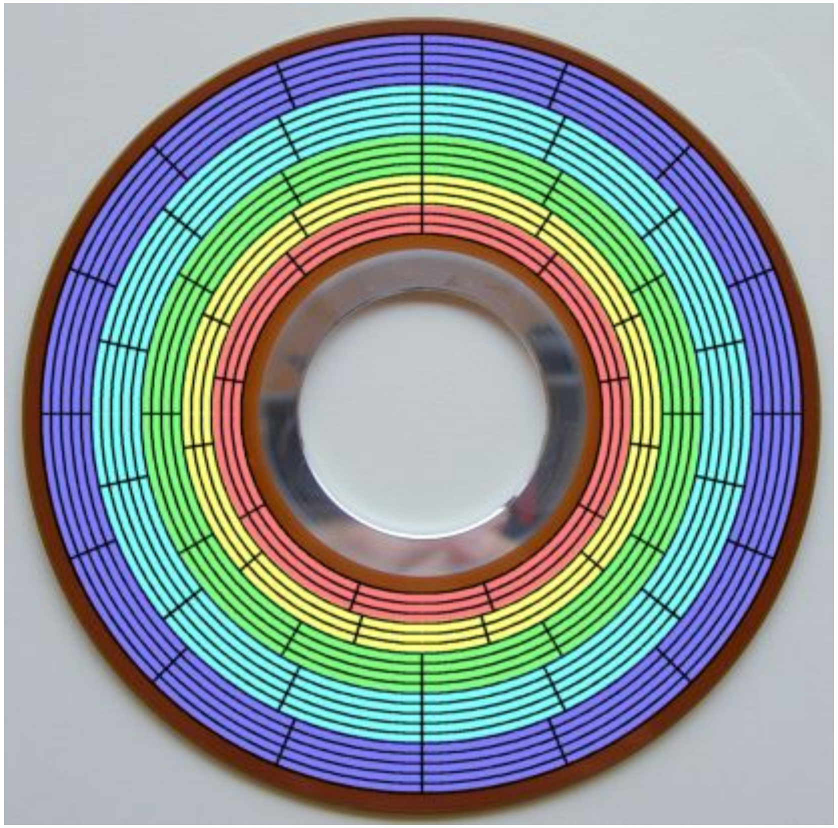


Confusing on bit density of sector

- Zone Bit Recording

zone bit recording (ZBR) is a method used by disk drives to store more sectors per track on outer tracks than on inner tracks.





How to access?

- CHS Addressing
  1. Cylinder (mechanical operation)
  2. Head (electronic choose)
  3. Sector (mechanical rotation, but so fast)
  – E.g. 1024/16/63 → 10/4/6bits

Cylinder
- From outer to inner(mech

Header
- Form up to down (electronic)

Sector
- From smaller to bigger(so

Disk IO Performance

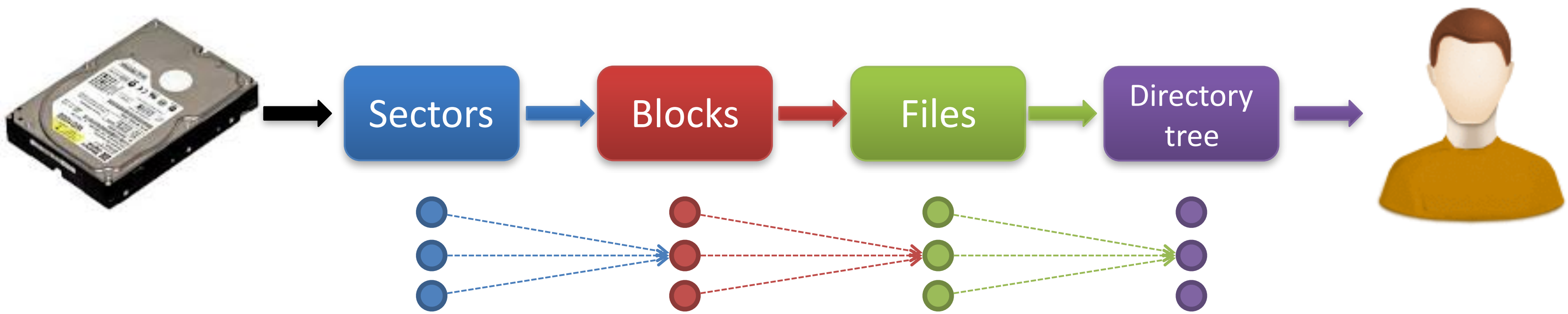
- $\mathbf{T}_{i/o} = \mathbf{t}_{seek} + \mathbf{t}_{rotate} + n*\mathbf{t}_{transfer}$
  - Seek time(0.2-0.8ms)
  - Rotational latency(5400/7200 rpm)
  - Data transfer rate(so fast)
- IOPS(Input/Output operations per second)
  - Sequential Read IOPS
  - Sequential Write IOPS
  - Random Read IOPS
  - Random Write IOPS

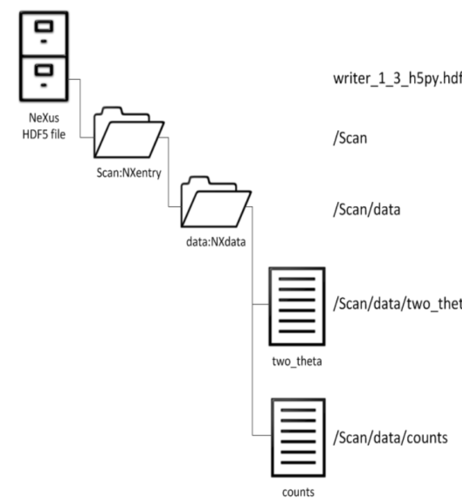
Disk/IO Scheduler

- It determines the motion of the of disk's arm and head in servicing read and write requests.
- IO Scheduler goals
  - To minimize time wasted by hard disk seeks
  - To prioritize a certain process' I/O requests
  - To give a share of the disk bandwidth to each running process
  - To guarantee that certain requests will be issued before a particular deadline
- Scheduler algorithms (choose one according by different workloads)
  - FIFO (a.k.a. FCFS)
  - SSTF (Shortest Seek Time First)
  - SCAN (a.k.a. Elevator Algorithm, LOOK, C-SCAN, C-LOOK)
  - FSCAN, N-Step-SCAN (prevents "starvation" and "arm stickiness")
  - CFQ (Completely Fair Queuing, used for desktop system)
  - AS (Anticipatory Scheduler, replaced by CFQ, some used for web server)
  - Deadline (often used for database)
  - NOOP (maybe used for non disk-based block devices, e.g. SSD)
- Scheduler implementation
  - cache/buffer
  - Read/write requests queue(s)

File system on Disk

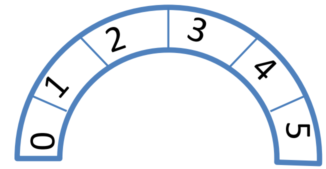
A 2G file will maintain 4,194,304 sectors(512bytes per sector)



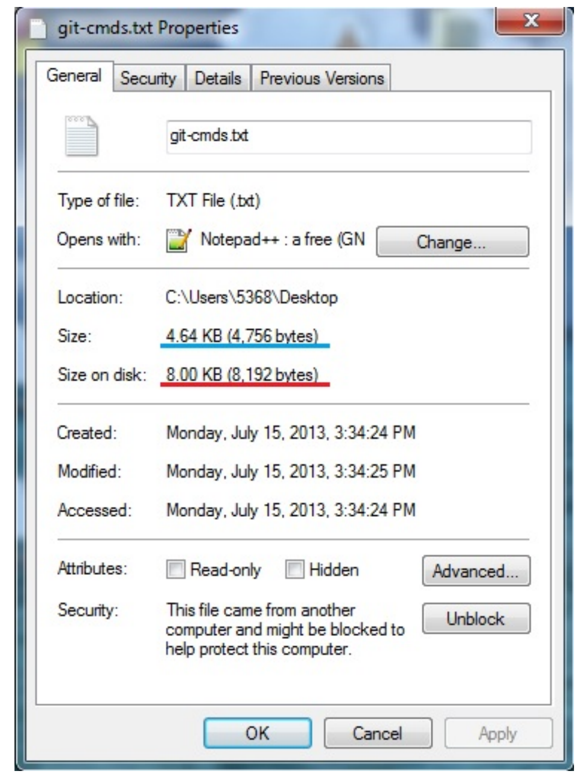


HD Sectors vs. FS Blocks

- Sectors vs. Blocks
  - Physically, data is on disk, unit is sector
  - Logically, data is on file, unit is block
  - Initialized by file system format
- LBA(Logical Block Addressing)
  - Logical block number
  - HD controller maps it to physical CHS
- CHS to LBA mapping
  - $A = (c \cdot N_{heads} + h) \cdot N_{sectors} + (s - 1),$
  - CHS(0, 0, 1) -> Block0
  - CHS(0, 0, 2) -> Block1
  - …



block	sector
0	0,1
1	2,3
2	4,5



Readahead on disk

- Read ahead
  - Optimization for read performance on disk.
  - A <u>system call</u> of the Linux kernel that pre-fetches a file's more data into the <u>page cache</u> in case requests later.
  - Useful for <u>sequential access</u>, but not for random access
  - Default size 256 sectors(128KB, 2^n of page size) on Linux
  - Read Amplification(if #blockdev --setra size too large)

```
[xuchao@mdap-server4 ~]$ blockdev

Usage:
  blockdev -V
  blockdev --report [devices]
  blockdev [-v|-q] commands devices

Available commands:
        --getsz                         get size in 512-byte sectors
        --setro                         set read-only
        --setrw                         set read-write
        --getro                         get read-only
        --getss                         get logical block (sector) size
        --getpbsz                       get physical block (sector) size
        --getiomin                      get minimum I/O size
        --getioopt                      get optimal I/O size
        --getalignoff                   get alignment offset
        --getmaxsect                    get max sectors per request
        --getbsz                        get blocksize
        --setbsz BLOCKSIZE              set blocksize
        --getsize                       get 32-bit sector count
        --getsize64                     get size in bytes
        --setra READAHEAD               set readahead
        --getra                         get readahead
        --setfra FSREADAHEAD            set filesystem readahead
        --getfra                        get filesystem readahead
        --flushbufs                     flush buffers
        --rereadpt                      reread partition table
```

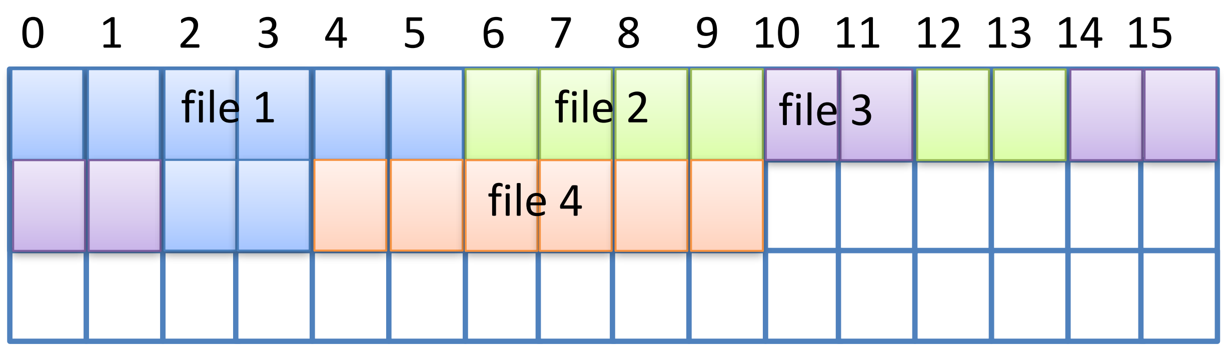
```
[xuchao@mdap-server4 ~]$ sudo blockdev --report
[sudo] password for xuchao:
RO    RA   SSZ   BSZ   StartSec            Size   Device
rw   256   512  4096          0    898319253504   /dev/sda
rw   256   512  1024         63        41094144   /dev/sda1
rw   256   512  4096      81920      2147483648   /dev/sda2
rw   256   512  1024    4276224       209715200   /dev/sda3
rw   256   512  1024    4685824            1024   /dev/sda4
rw   256   512  4096    4687872    209715200000   /dev/sda5
rw   256   512  4096  414289920    104857600000   /dev/sda6
rw   256   512  4096  619091968     52428800000   /dev/sda7
rw   256   512  4096  721494016     33554432000   /dev/sda8
rw   256   512  4096  787032064     10485760000   /dev/sda9
rw   256   512  4096  807514112    484872028160   /dev/sda10
```

Fragmentation

- In blocks

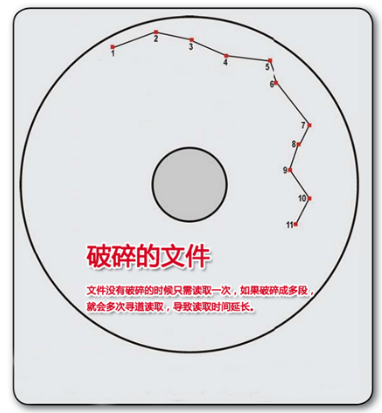
- On disk
  - As file operations(append, update, delete)

File	blocks
1	0,1,2,3,~~4,5,~~ 18,19
2	6,7,8,9 , 12,13
~~3~~	~~10,11, 14,15,16,17~~
4	20,21,22,23,24,25



Fragmentation

- Fragments In blocks
  - Can’t avoid
  - Block size depends on practical scenario
  - Capacity util% will be less
- Fragments On disk
  - It depends on file system(Fat, Ext2/3/4)
  - Head has more seek
  - IO Performance will be slow
  - Defragmentation will cause **Write Amplification**



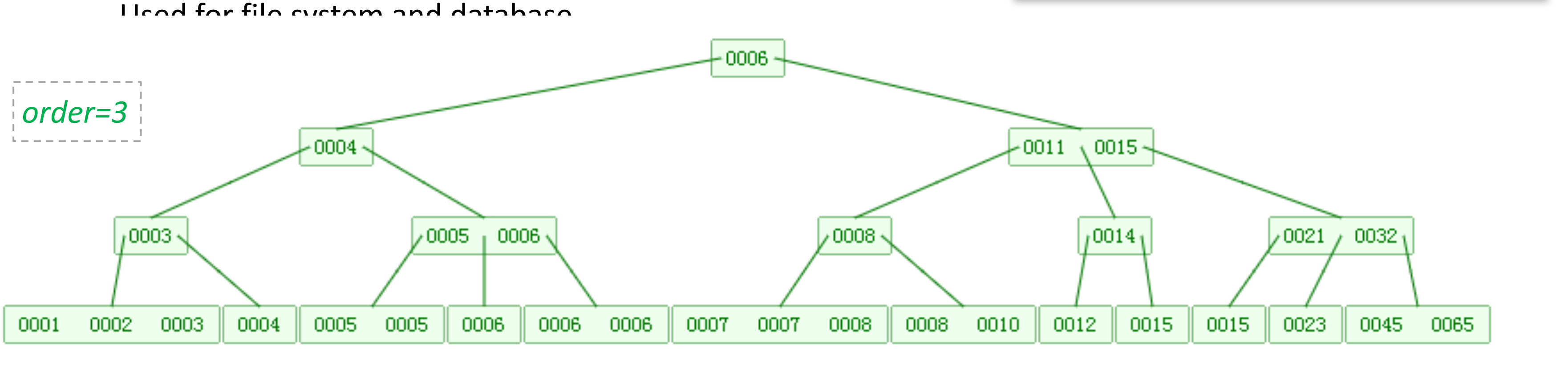
B-tree

- An order ***m*** B-tree
  - Every node has at most *m* children.
  - Every non-leaf node (except root) has at least $\lceil m/2 \rceil$ children.
  - The root has at least two children if it is not a leaf node.
  - A non-leaf node with *k* children contains $k-1$ keys.
  - All leaves appear in the same level.
- Use and advantages
  - O(log n) in searches, sequential access, insertions, and deletions
  - Perfect matched with file system & disk
  - Each node maps to a page(4KB), full loaded by readahead
  - Used for file system and database

An order ***m*** B-tree having ***N*** data:

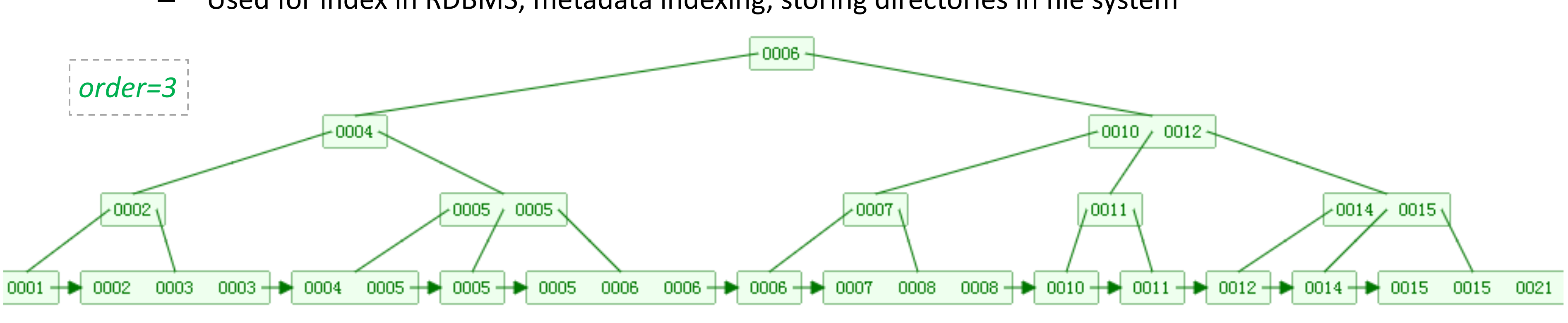
- Each non-leaf node has [*m/2 , m*] children
- The height is ($log_{m-1}N$, $log_{m/2}N$)
- If ***N*** = 62*1000000000, ***m***=1024
- then, $log_{m/2}N$ <= 4

That means, we just need **≤4** reads for each operation(search, insert, delete) in 62 billion records.

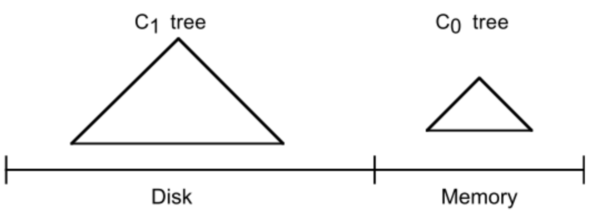


B+ tree

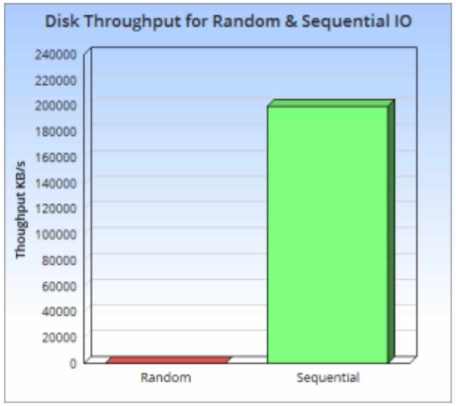
- Compare with B-tree
  - Each internal node contains only keys (not key–value pairs)
  - An additional level is added at the bottom with linked leaves
  - Only leaf nodes contain data info
- Use and advantages
  - More efficient for range queries
  - Each node contains more keys(only leaf nodes contain data)
  - Very high fanout(typically on the order of 100 or more)
  - More index keys can be loaded into memory and cache
  - More balanced(all data on leaf level)
  - Used for index in RDBMS, metadata indexing, storing directories in file system

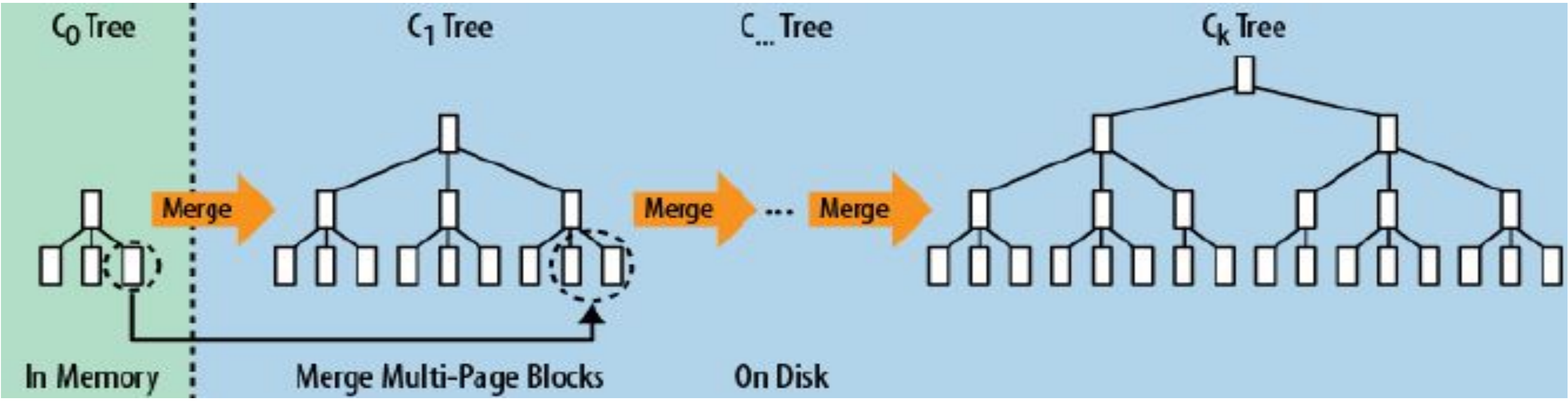


LSM tree



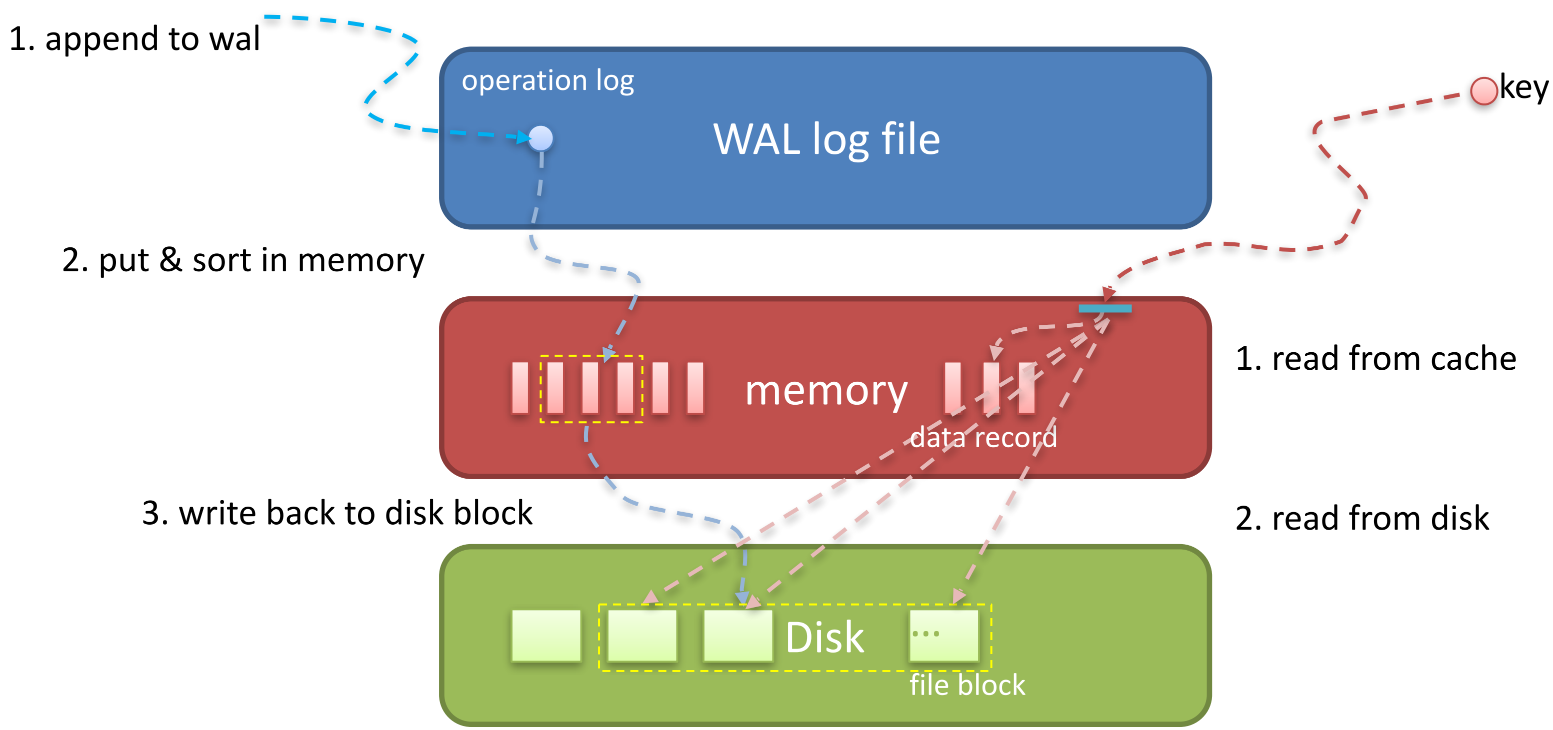
- Log-Structured Merge-tree
  - Sequential IO performance >>> Random IO performance
  - Maintaining data in two or more tree-like component data structures underlying different storages
  - Data is wrote back to disk in rolling batches in some latency.
  - Write ahead log for guarantee data safety
  - Put and sort in memory temporarily(preventing I/O cost)
  - Write back to multi-page blocks on disk sequentially
  - Periodically compact files, split or merge regions
  - Used for HBase, LevelDB, BigTable, MongoDB, RocksDB, Cassandra



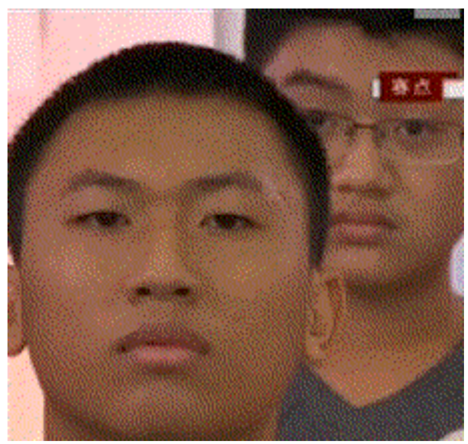


LSM tree

- Read/Write path



What about SSD?



Optimizations for storage system

- 总结的一些常用优化手段：
  - 利用数据的局部性原理和磁盘预读机制来减少数据加载时的磁盘IO次数；
  - 利用顺序写/顺序读来降低磁盘的寻道时间；
  - 利用批量写来降低操作系统的IO调用次数，从而降低内核态与用户态之间的上下文切换开销；
  - 利用操作系统的虚拟内存技术，通过内存映射来降低数据在内核态与用户态之间的拷贝开销；
  - …

Reference

- https://computing.llnl.gov/tutorials/parallel_comp/
- https://en.wikipedia.org/wiki/CPU_cache
- https://en.wikipedia.org/wiki/Cache_memory
- http://www.hardwaresecrets.com/how-the-cache-memory-works/
- https://www.quora.com/How-does-the-cache-memory-in-a-computer-work
- http://ark.intel.com/products/64594/Intel-Xeon-Processor-E5-2620-15M-Cache-2_00-GHz-7_20-GTs-Intel-QPI
- https://superuser.com/questions/196143/where-exactly-l1-l2-and-l3-caches-located-in-computer
- https://en.wikipedia.org/wiki/Disk_sector
- https://en.wikipedia.org/wiki/Cylinder-head-sector
- https://en.wikipedia.org/wiki/Zone_bit_recording
- http://www.tldp.org/LDP/sag/html/hard-disk.html
- https://www.youtube.com/watch?v=Cj8-WNjaGuM&list=PLlVZ1eXuYslrb9G6xm4SKV51SO-KAFrCw&index=1&t=12s
- http://blog.csdn.net/hguisu/article/details/7408047
- http://www.pcguide.com/ref/hdd/geom/tracksZBR-c.html
- http://cn.linux.vbird.org/linux_basic/0230filesystem.php
- https://www.cs.uic.edu/~jbell/CourseNotes/OperatingSystems/10_MassStorage.html
- https://en.wikipedia.org/wiki/Hard_disk_drive_performance_characteristics
- https://en.wikipedia.org/wiki/I/O_scheduling
- https://www.howtogeek.com/115229/htg-explains-why-linux-doesnt-need-defragmenting/
- https://en.wikipedia.org/wiki/Readahead
- https://en.wikipedia.org/wiki/B-tree
- https://en.wikipedia.org/wiki/B%2B_tree
- http://www.cnblogs.com/yangecnu/p/Introduce-B-Tree-and-B-Plus-Tree.html
- http://www.cs.umb.edu/~poneil/lsmtree.pdf
- http://www.cnblogs.com/siegfang/archive/2013/01/12/lsm-tree.html