Overview of advanced storage technologies and storage virtualization

Pablo Pérez Trabado

Dept. of Computer Architecture

University of Malaga (Spain)

Disclaimers

- Some of the figures in the slides are taken from SNIA Education tutorials. To comply with the SNIA conditions of use for these documents, any SNIA tutorial used in the elaboration of these slides has been also provided as a handout

- Many (but not all) figures in the slides have been taken from Internet-available resources, including Wikipedia. Whenever possible by copyright restrictions, the original source has been also provided as a handout along with the slides. Also, whenever possible the original source is quoted. In any case, no ownership claims are made over images in the slides not drawn by the author himself.

Overview of disk storage technology

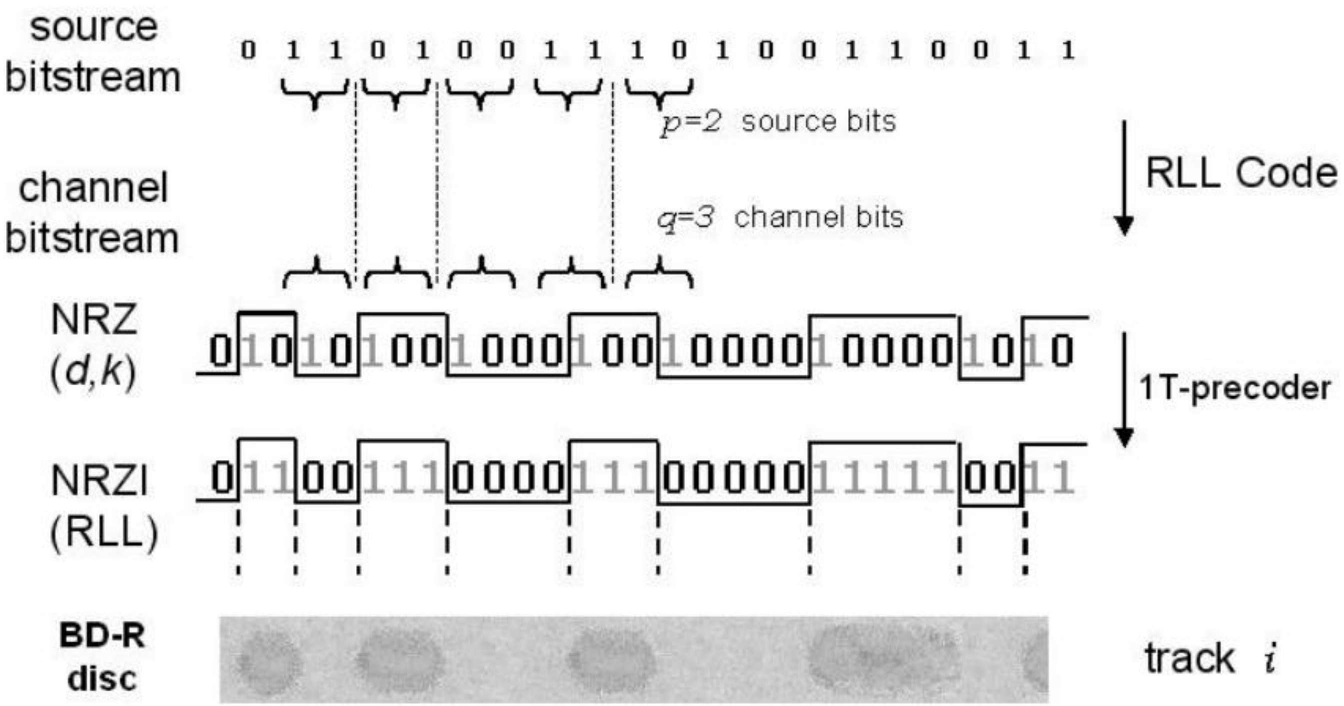
What will we learn?

- The basics of magnetic storage of data
- The physical implementation of hard disks
- The organization of data on the magnetic surface, and its impact on data
- The details of a disk seek operation, and its impact on performance
- The concept of IOPS and Transfer Rate as disk performance parameters
- The need to match application I/O requirements and disk performance parameters
- The way the OS can further shape I/O traffic

What will we learn?

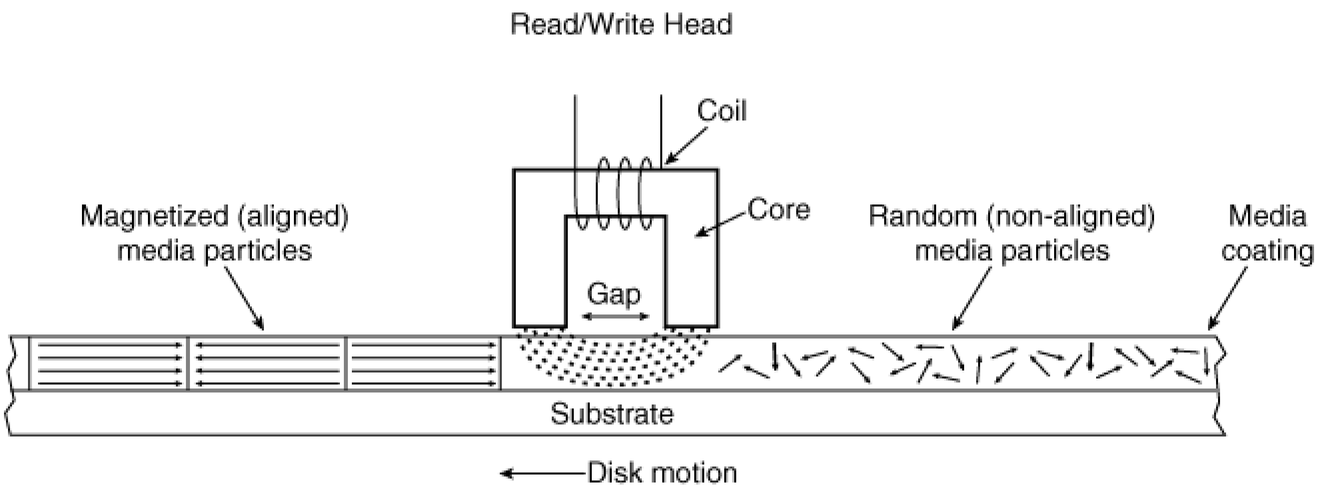
- The use of RAID configurations to improve performance and/or reliability
- The use of object-based storage to overcome the shortcomings of RAID for big data warehouses
- The internals of Solid State Disks (SSDs)
- The fortes and foibles of SSDs
- Its main role in today's storage systems

Magnetic storage of data



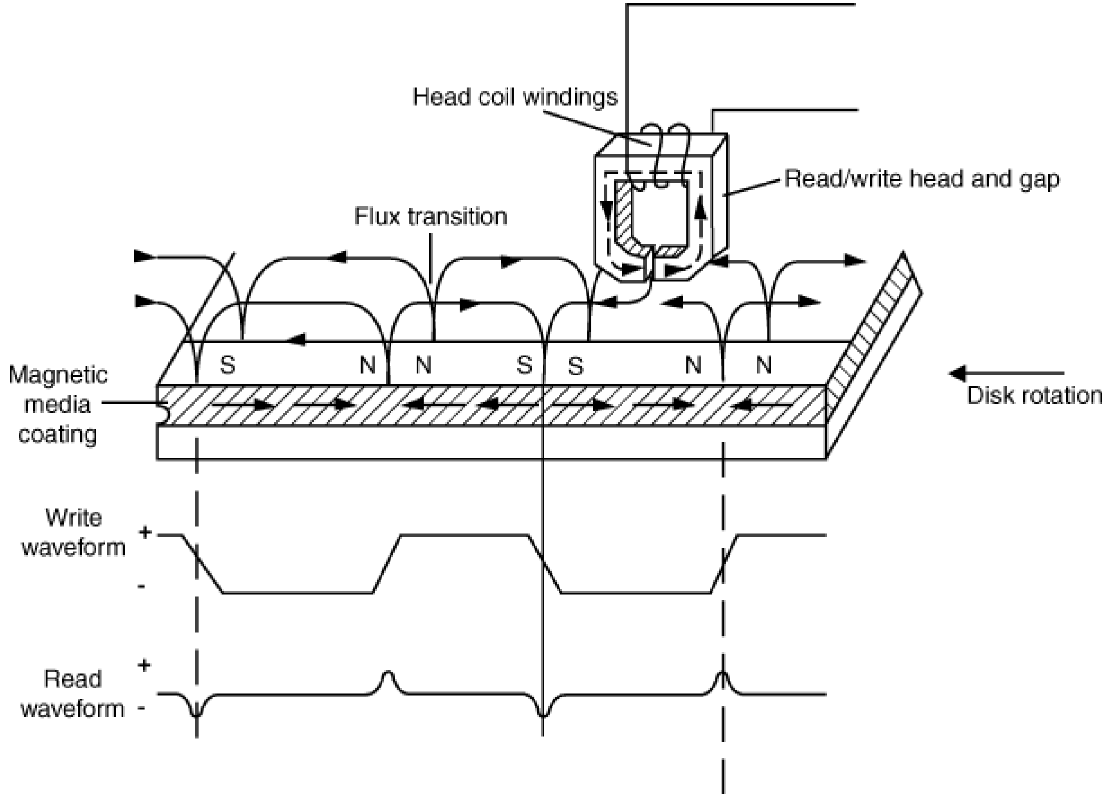
- Magnetic (and optic) storage of data uses modulation encoding:
  - Clocking info must be extracted from data itself
    - Edges in data match edges in CLK signal
  - Edges in data mark position of 1's in bit stream
  - Position of 0's guessed by receiver when no changes detected through a whole bit-length
  - Encoding of data must ensure edges neither too close not too far

Magnetic storage of data



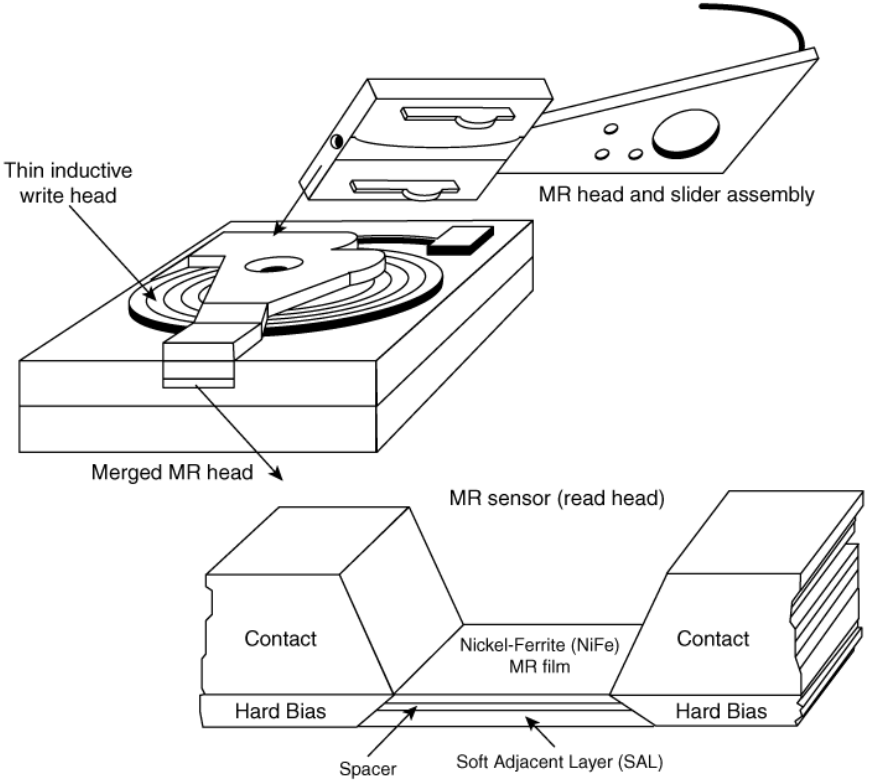
- In magnetic media, data recorded by magnetic alignment of media particles in substrate
  - Position of 1's in bitstream signaled by change in direction of magnetic domain
  - Substrate moves under writing head at a constant speed
  - Disk controller changes current applied to writing head to synchronize magnetic transitions with arrival of 1's in bitstream

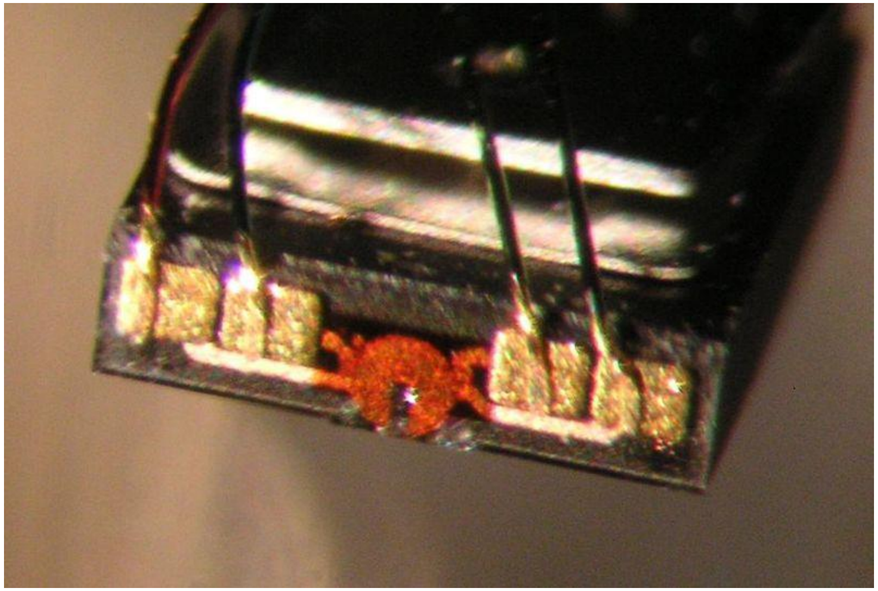
Magnetic storage of data



- Reading is done moving media under a magnetic read head
  - Changes of magnetic flux create current spikes when transitions go under head
  - Spikes mark position of 1’s
  - Position of 0’s given by controller’s data separator CLK, kept in bit-sync by transitions

Write and read heads

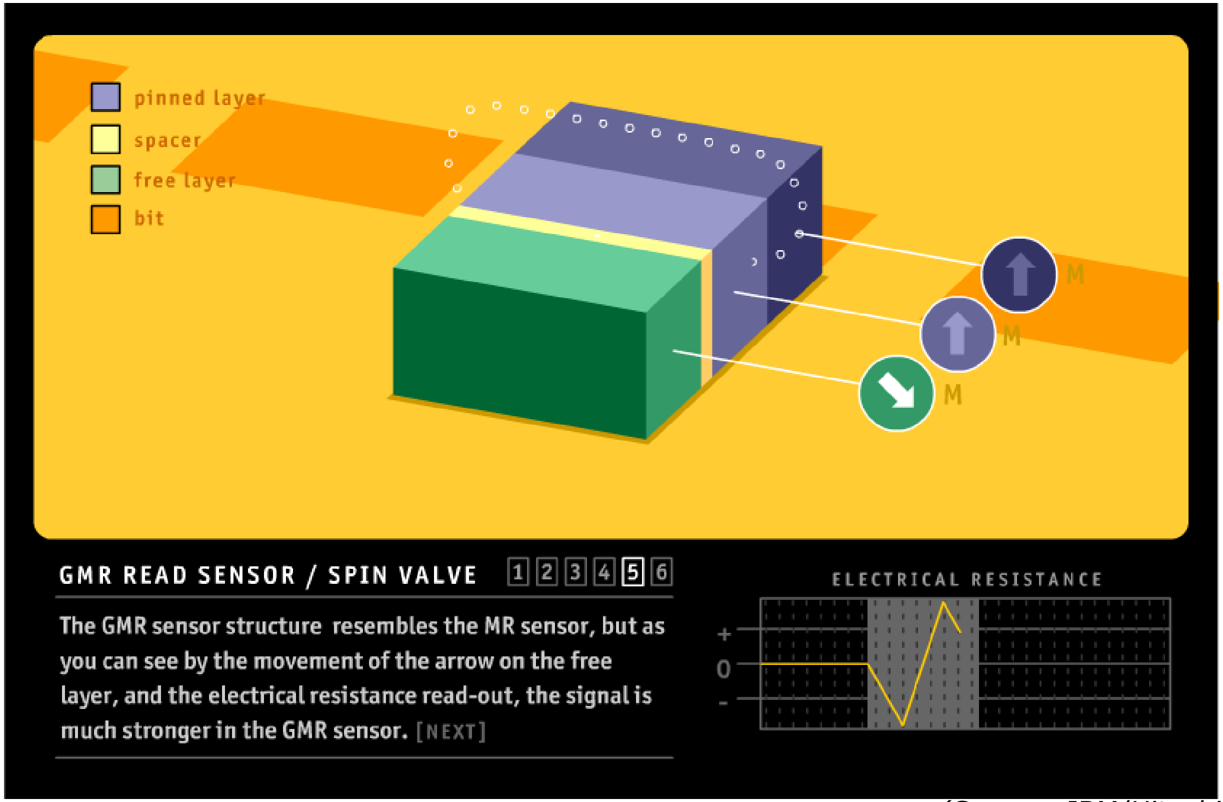




(Source: Wikipedia)

- Separate write and read heads are currently used
- For writing, Thin Film heads were used
  - Small coil and ferrite, made through photolithographic process

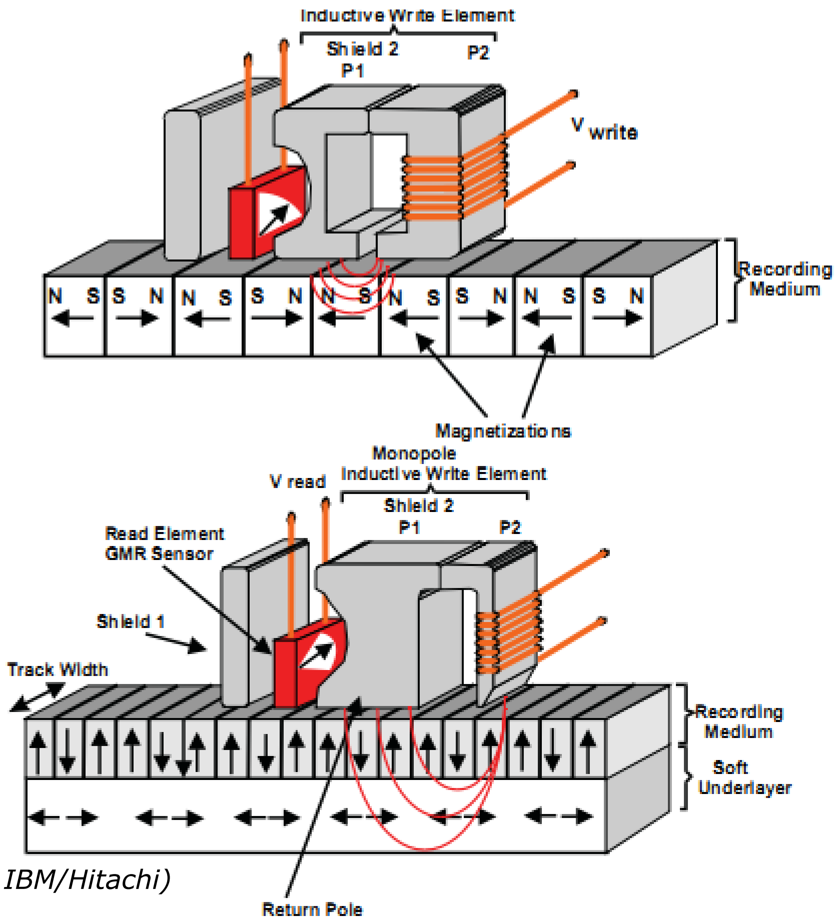
Write and read heads



(Source: IBM/Hitachi)

- Read head uses Giant Magneto Resistive technology
  - Electrical resistance of head changes when over magnetic transition
  - Generates much stronger peak signal than coil-based head
    - Allows reduction of track size, and thus increase of data density

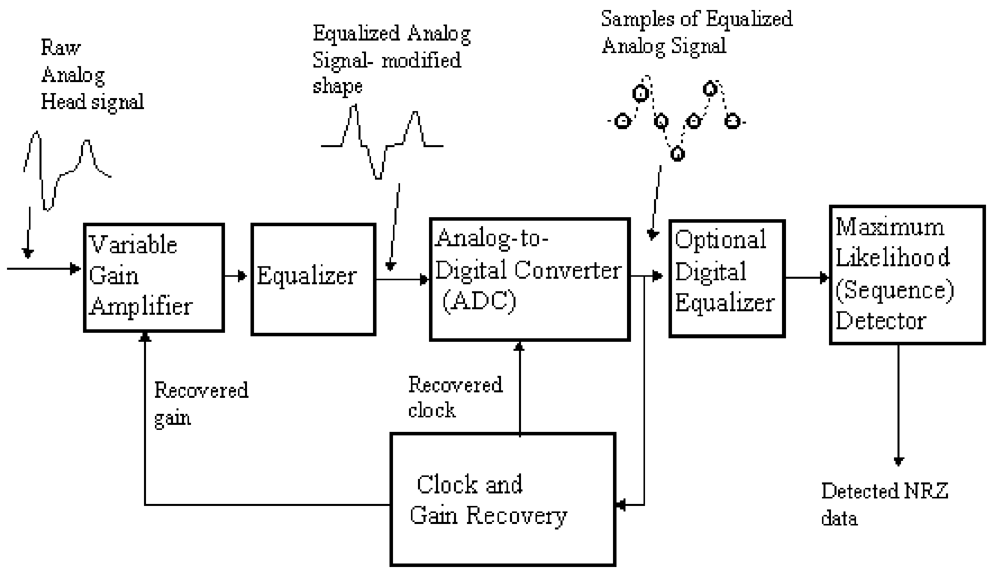
Perpendicular recording

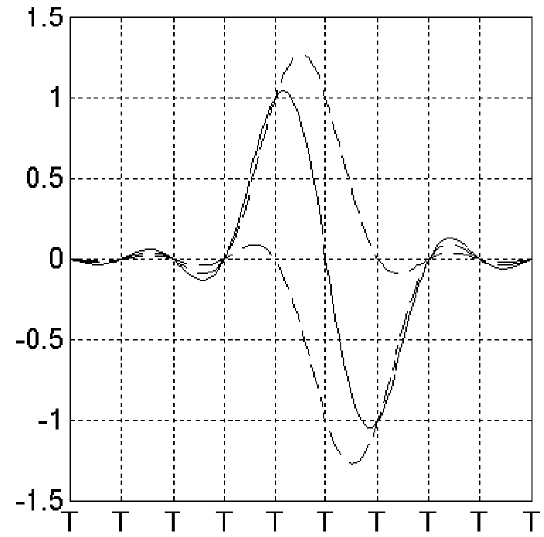


(Source: IBM/Hitachi)

- Use of perpendicular recording improves performance of write head
  - Tracks can be smaller, as substrate works as part of head
  - Magnetic domains are shorter, and data density is increased

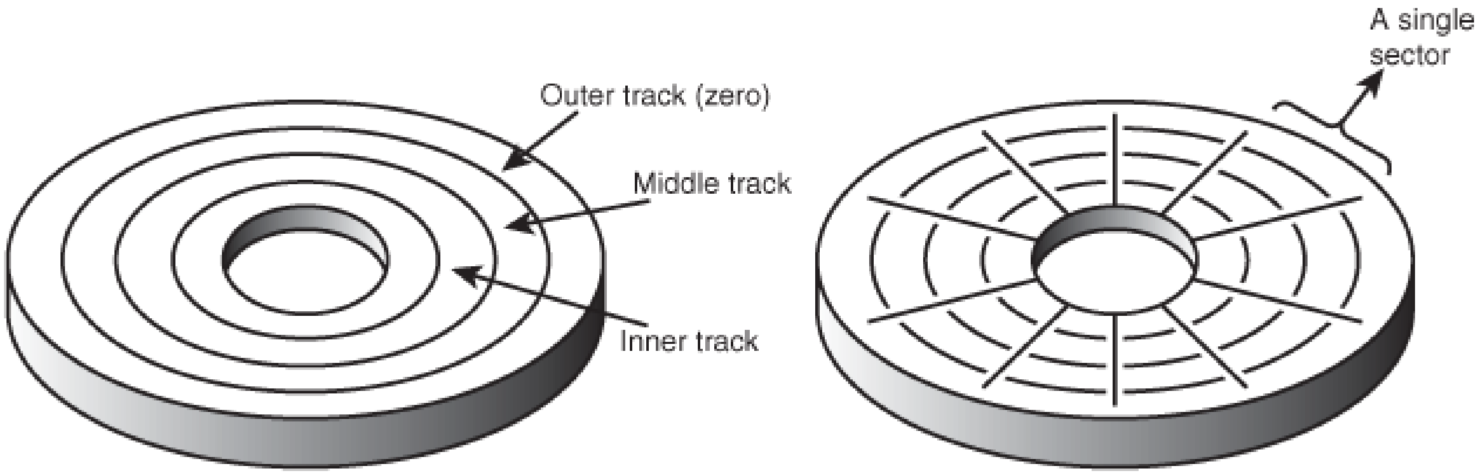
PRML decoding





- Use of PRML (Partial Response Maximum Likelihood) decoding has allowed also shorter magnetic domains
  - Shorter domains means interimpulse interference (over Nyquist limit)
  - PRML uses DSP to digitize analog read signal and analyze the samples
    - PRML guesses from digital signature of signal the sequence of pulses
    - Good likelihood or guessing right
    - I f guesses wrong, ECC is enough to correct the wrong bit

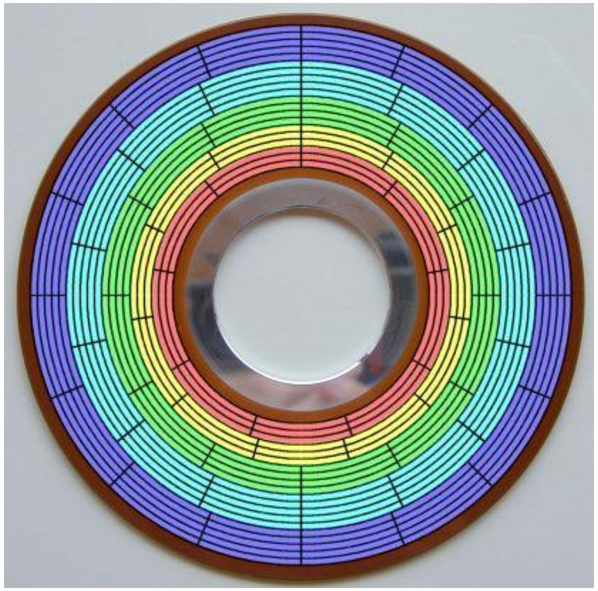
Zoning



Tracks and Sectors

- In magnetic disk, recording organized in circular tracks, split in sectors
- Classical organization of fixed number of sectors per track no longer good
  - Linear density in outer tracks much lower than inner tracks
  - Waste of capacity

Zoning



- Solution: ZBR = *Zoned Bit Recording*
  - Number of sectors per track change with radius
- Tracks organized in a number of zones
  - All tracks belonging to same zone have same number of sectors
- Transparently managed by HDD controller
  - Again, LBA addressing helps hiding the physical mapping

Zoning

- Usually, lower numbered zones belong to outer tracks
  - Example: zoning in Hitachi 7k1000 disks
- Outer zones/tracks have the greater number of sectors
  - Important when squeezing maximum performance from disks (i.e, "short stroking")

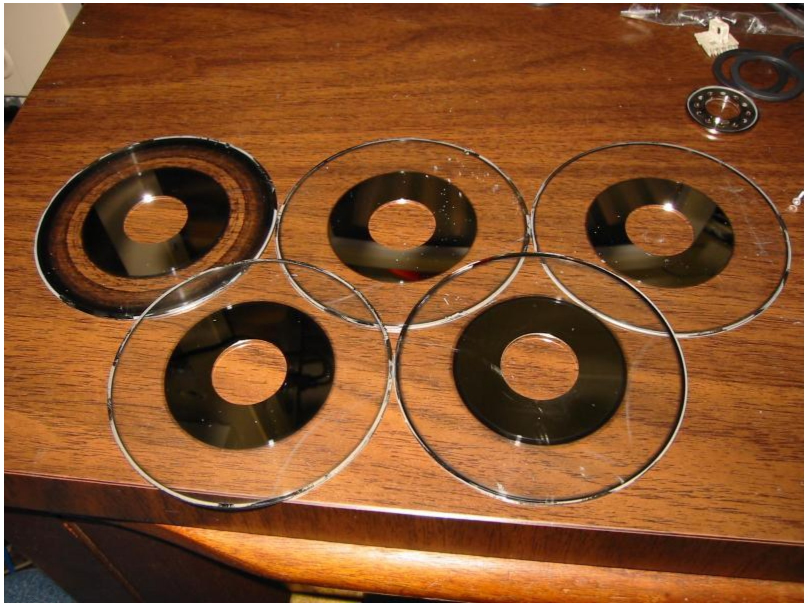
Zone	Logical cyl. (OD)	Logical cyl. (ID)	1000GB model #sec
0	0	8319	1680
1	8320	16639	1632
2	16640	24959	1620
3	24960	32639	1600
4	32640	39679	1560
5	39680	47359	1520
6	47360	51199	1520
7	51200	56319	1488
8	56320	61439	1440
9	61440	66559	1440
10	66560	71679	1392
11	71680	78719	1360
12	78720	81919	1360
13	81920	87039	1320
14	87040	92031	1280
15	92032	97023	1248
16	97024	101887	1200
17	101888	106367	1200
18	106368	110719	1152
19	110720	114175	1120
20	114176	118015	1080
21	118016	121727	1080
22	121728	125823	1040
23	125824	129919	1008
24	129920	134015	960
25	134016	137855	912
26	137856	140671	912
27	140672	143231	880
28	143232	146303	880
29	146304	147583	840

HDD anatomy

(Source: hddscan.com)

- Internally, hard disks are composed by:
  - Stacks of disks, turning at a fixed rpms
  - Head stack assembly, which pivots under control of voice coil

HDD anatomy



(Source: hddscan.com)

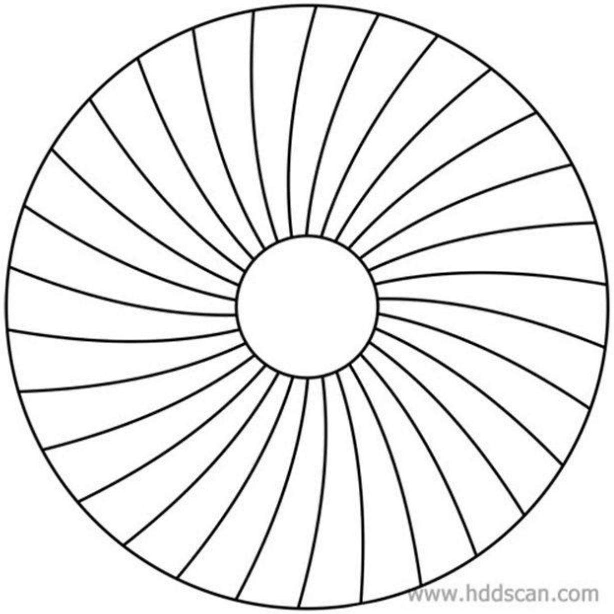
- Disk platters made of glass, or aluminium
  - Lower thermal expansion coefficient
- Disks stacked together and turning solidarily

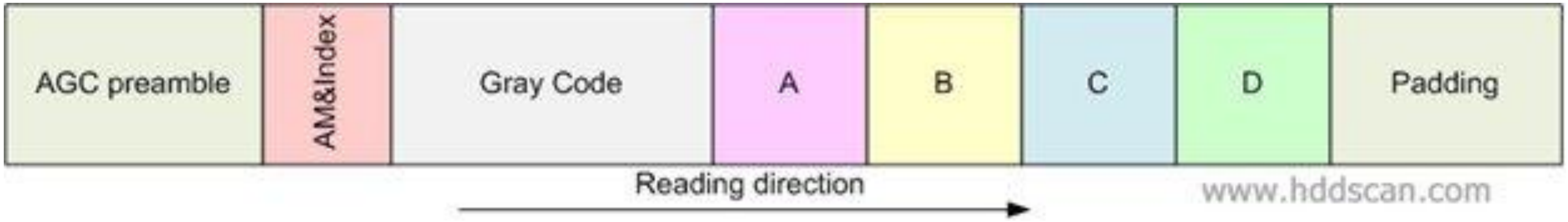
HDD anatomy

(Source: hddscan.com)

- Head stack pivots on bearing
- Movement controlled by voice coil
  - Track 0 = outer track of disk
  - Angle turned from parking area (center of disk) proportional to current in voice coil
  - Closed-loop analog control system allows for very fine positioning corrections
  - Needs positioning info in disk, to be read by head itself

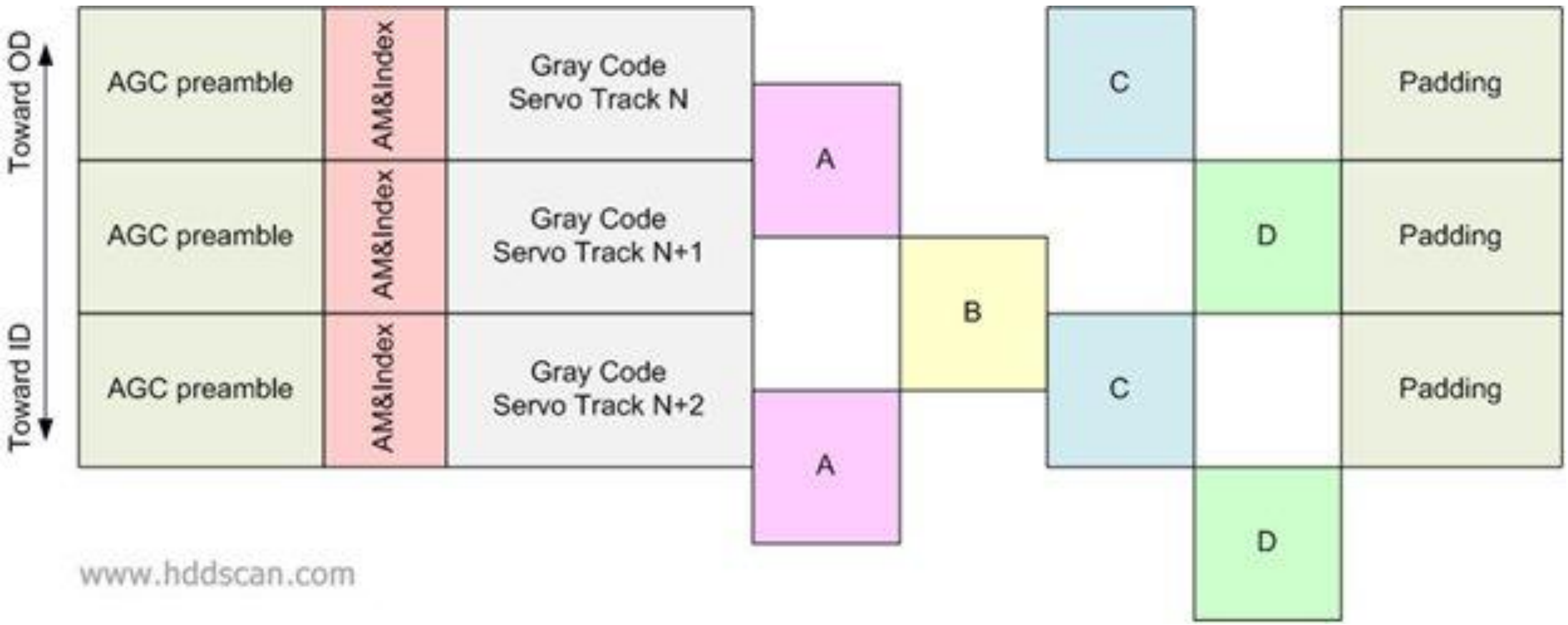
Servo track





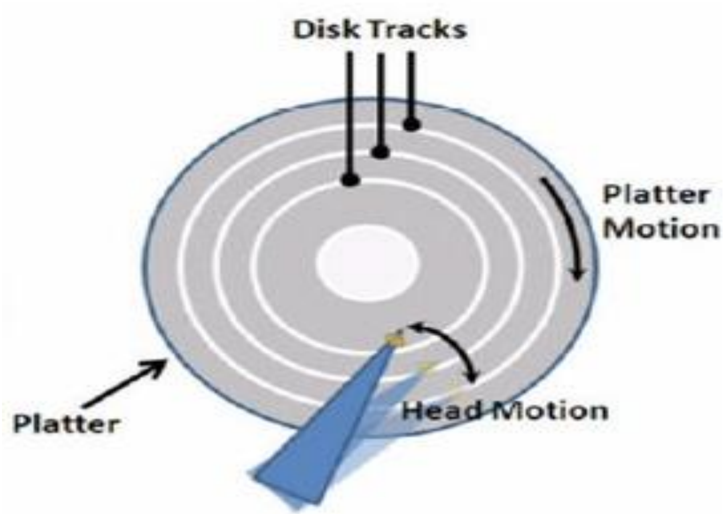
- Positioning info contained in servo wedges within disk surface

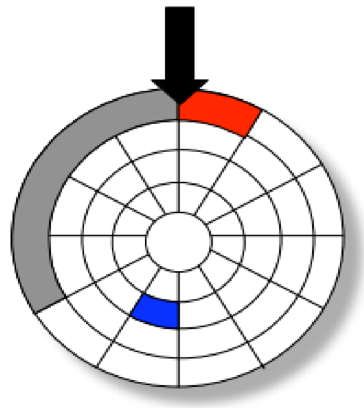
Servo track



- Once head is approximately over required track, servo information allows fine tuning of position:
  - Read head gets A-B-C-D pattern
  - Controller calculates position error correction needed
  - Current to voice coil modified accordingly
  - Repeat until no position error

HDD anatomy: seek time

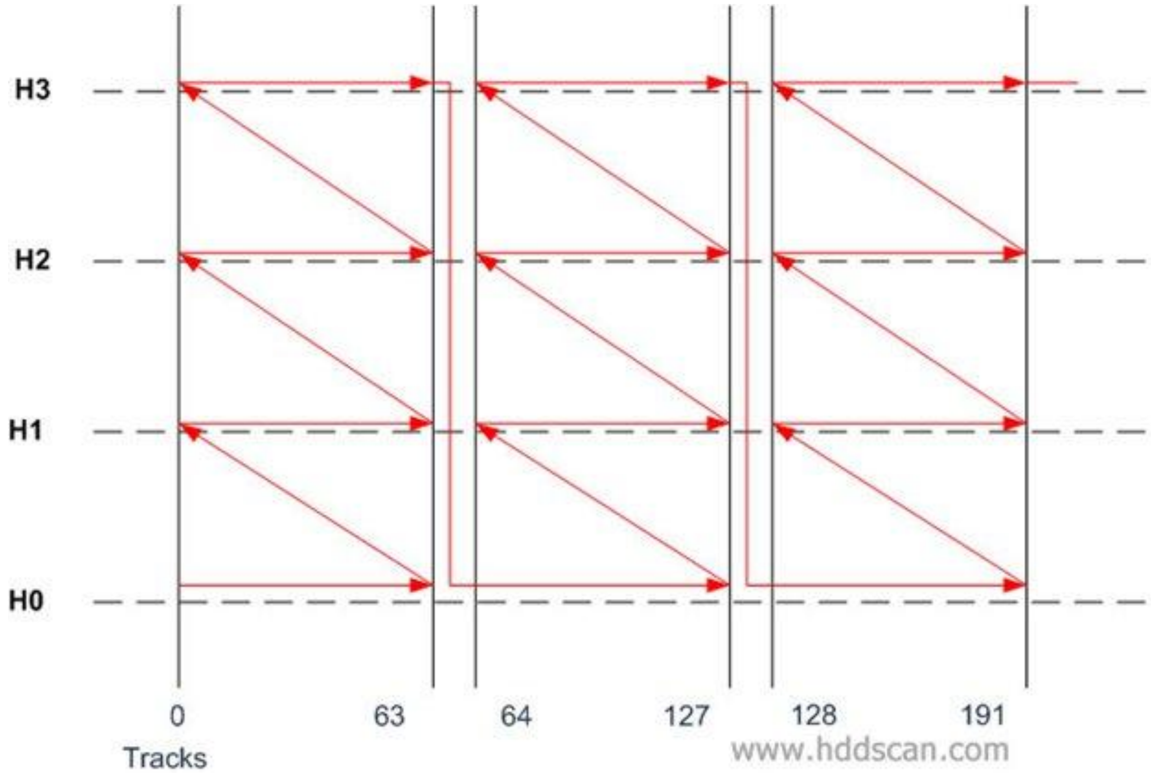




Rotational latency

- **Seek process requires:**
  - Controller receives and interprets SCSI command and gets LBA
  - Controller calculates surface, track and sector where LBA is stored
  - Controller calculates needed current and drives voice coil
  - Head stack moves to approximate position of track
  - Head reads servo info; controller calculates correction needed
  - Head waits for sector to move under head
- **Only THEN can data start being read from surface**

HDD anatomy: transfer time

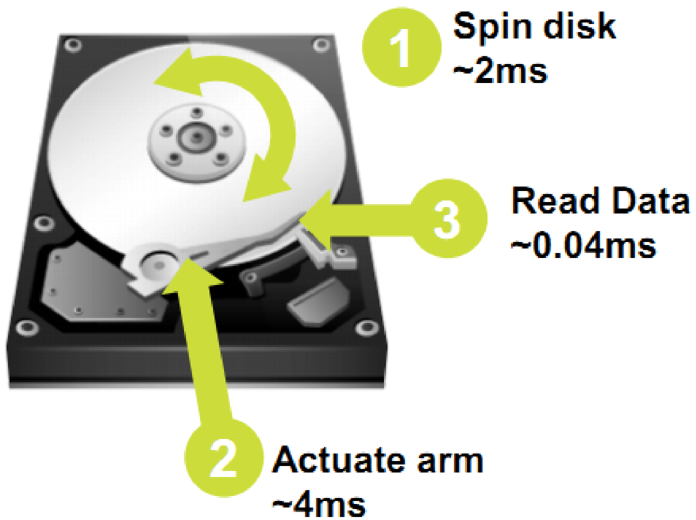


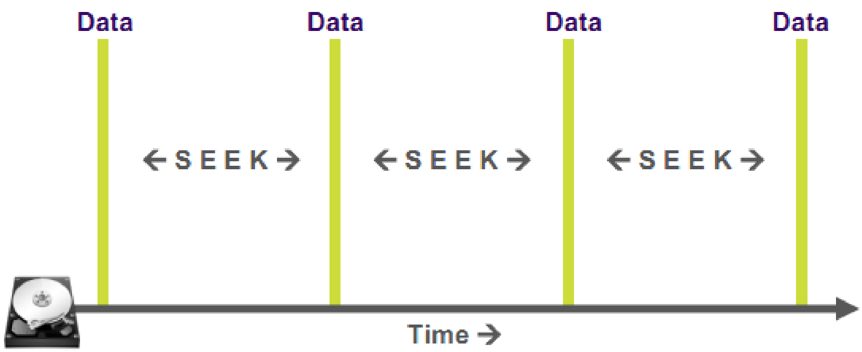
- Once read/write has started, it can go as fast as hardware cans
  - Electronic head switching avoids need to change position when track completely read/written
  - Large transfers can, then, proceed with no, or minimal, seeking delays

HDD performance: IOPS vs. Transfer Rate

- Clearly, there are two parameters which impact disk performance:
  - Seek time = Time needed to position head to LBA addressed by (SCSI) command
    - Indirectly measured by disk's IOPS = I/O Operations Per Second
  - Transfer Rate = Time needed for transfer of data to/from media, once seek is done
- Traffic pattern generated by application is critical to which one of these two parameters will control performance of our storage system

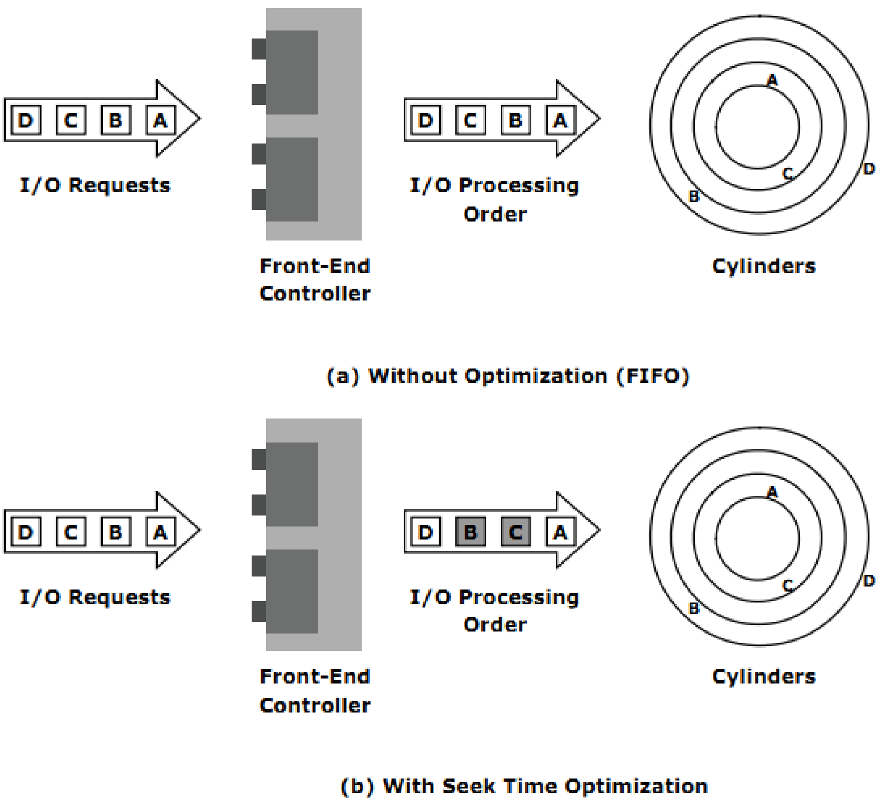
IOPS and random access





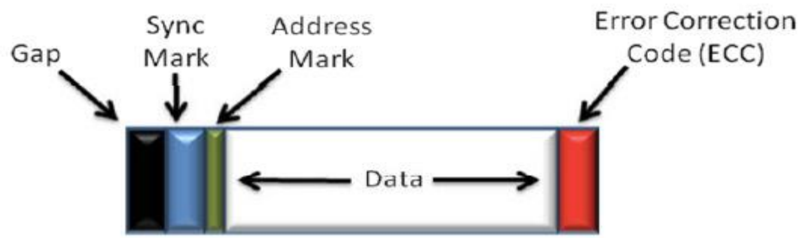
- If application generates many small read/write operations, randomly distributed over LBA range, seek time predominates
  - Typical behavior of database applications
- For this traffic pattern, critical parameter is number of IOPS disk is able to serve
- IOPS can be improved by faster (= more expensive) disks
  - 15k rpm disks (SAS, FC) may be needed, instead of slower 7.5k disks (SATA)

IOPS and random access



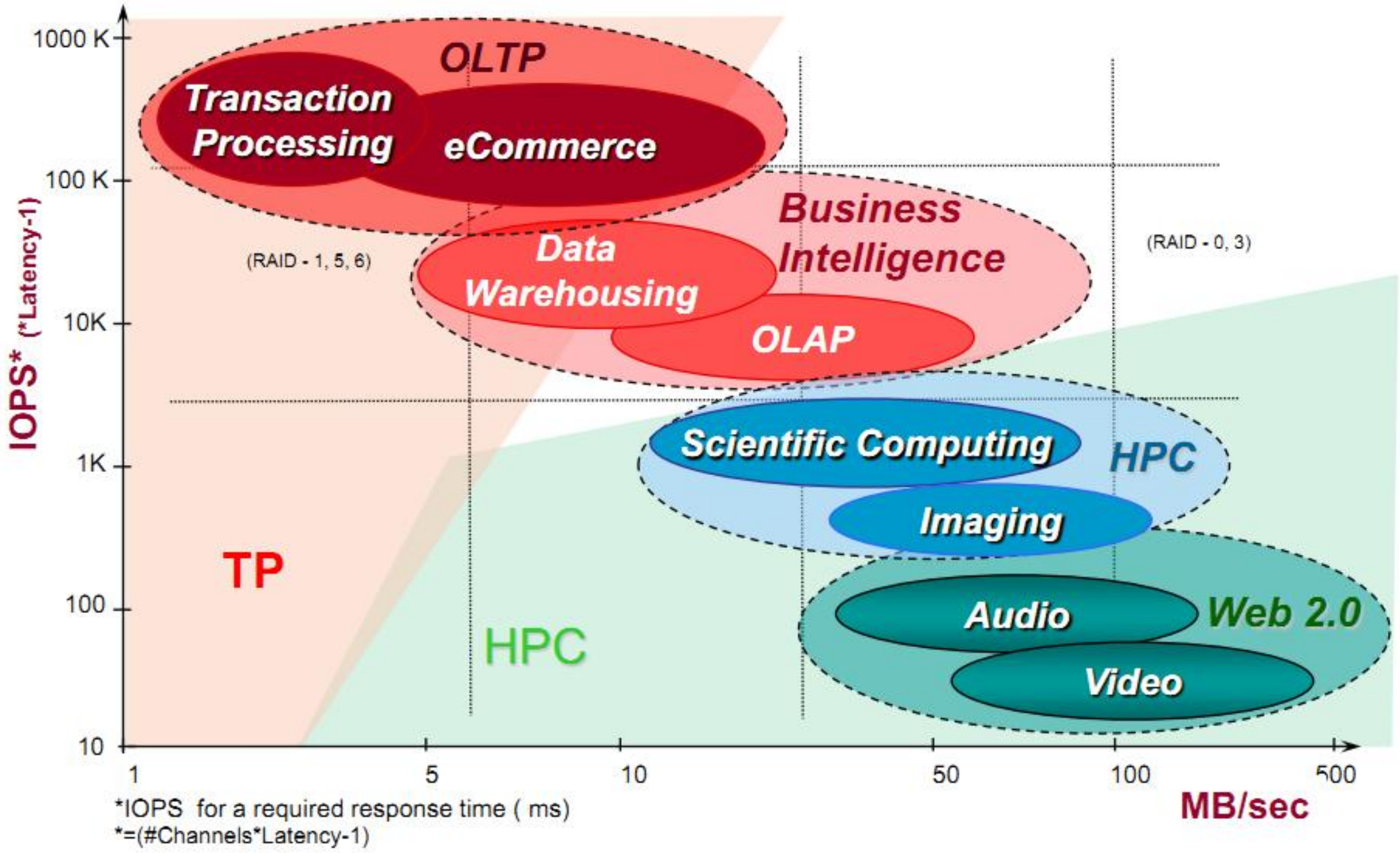
- Command queuing and reordering can also help
  - Reduce seek times by reducing head travel length
  - SCSI disks (FC, SAS) much better than SATA
    - SCSI queue deeper and better optimized than SATA's native queueing (NCQ)

Transfer rate and sequential access



- If application generates a few read/write operations with large transfer lengths, data transfer time predominates over seek times
  - Typical behavior of sequential access applications
    - Example: streaming of video/audio media over Internet
- Transfer rate depends on disk speed, track data density and which zone read
  - 15k rpm turns twice as fast as 7.5k rpm -> delivers twice the data rate
  - Inner zones contain less sectors than outer zones
    - Disk data transfer performance falls as disk gets filled !!

Traffic pattern by application

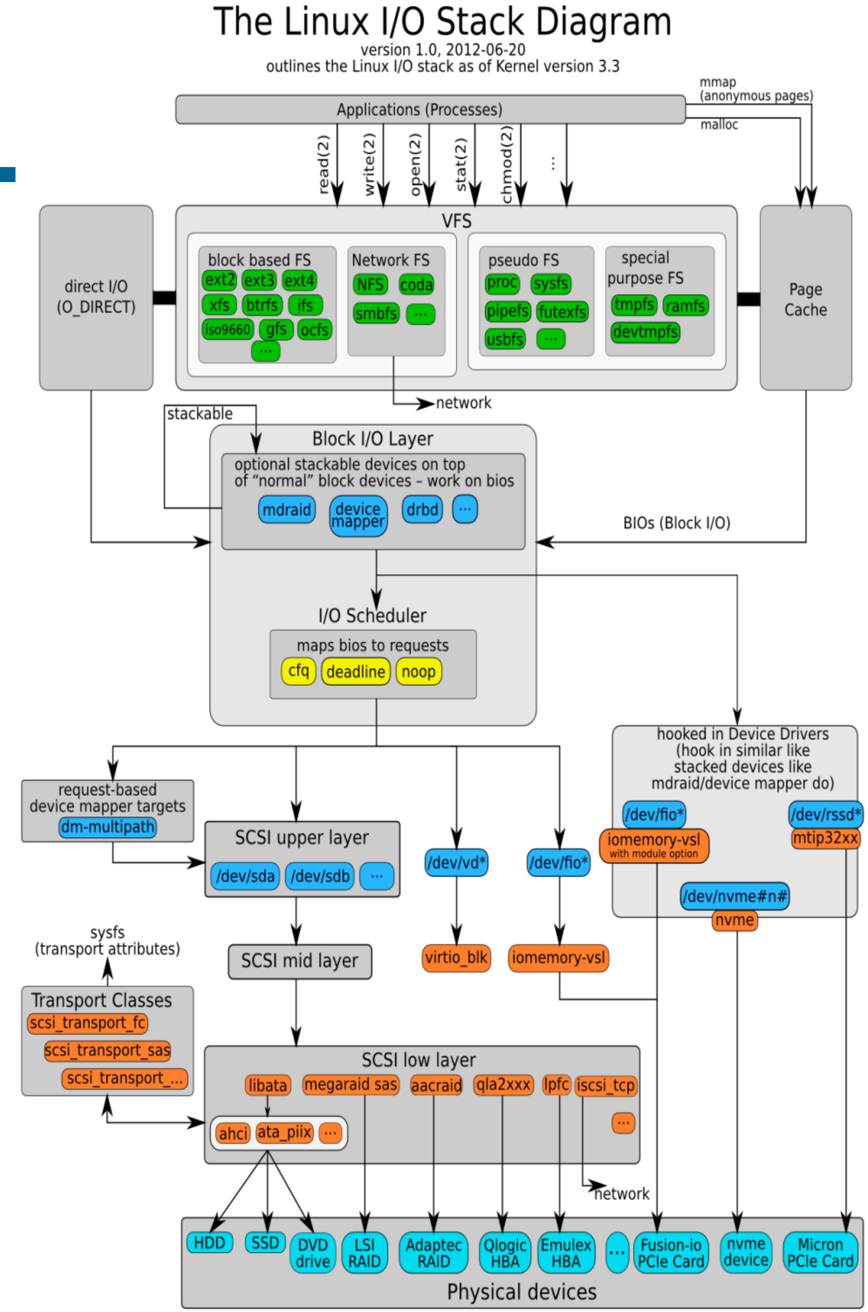


- Knowing your application will allow you to predict shape of I/O
  - Thus, predict if you will need higher IOPS or higher transfer rates

I/O shaping by OS

- OS “optimizations” can have heavy impact in I/O traffic pattern

- Example: Linux I/O scheduler
  - Maps filesystem-level block I/O to SCSI-level block I/O
  - Performs coalescing (merge) of requests to increase transfer length and reduce commands
  - Sort commands by ascending LBA, to reduce disk seeks



(Source: wikipedia)

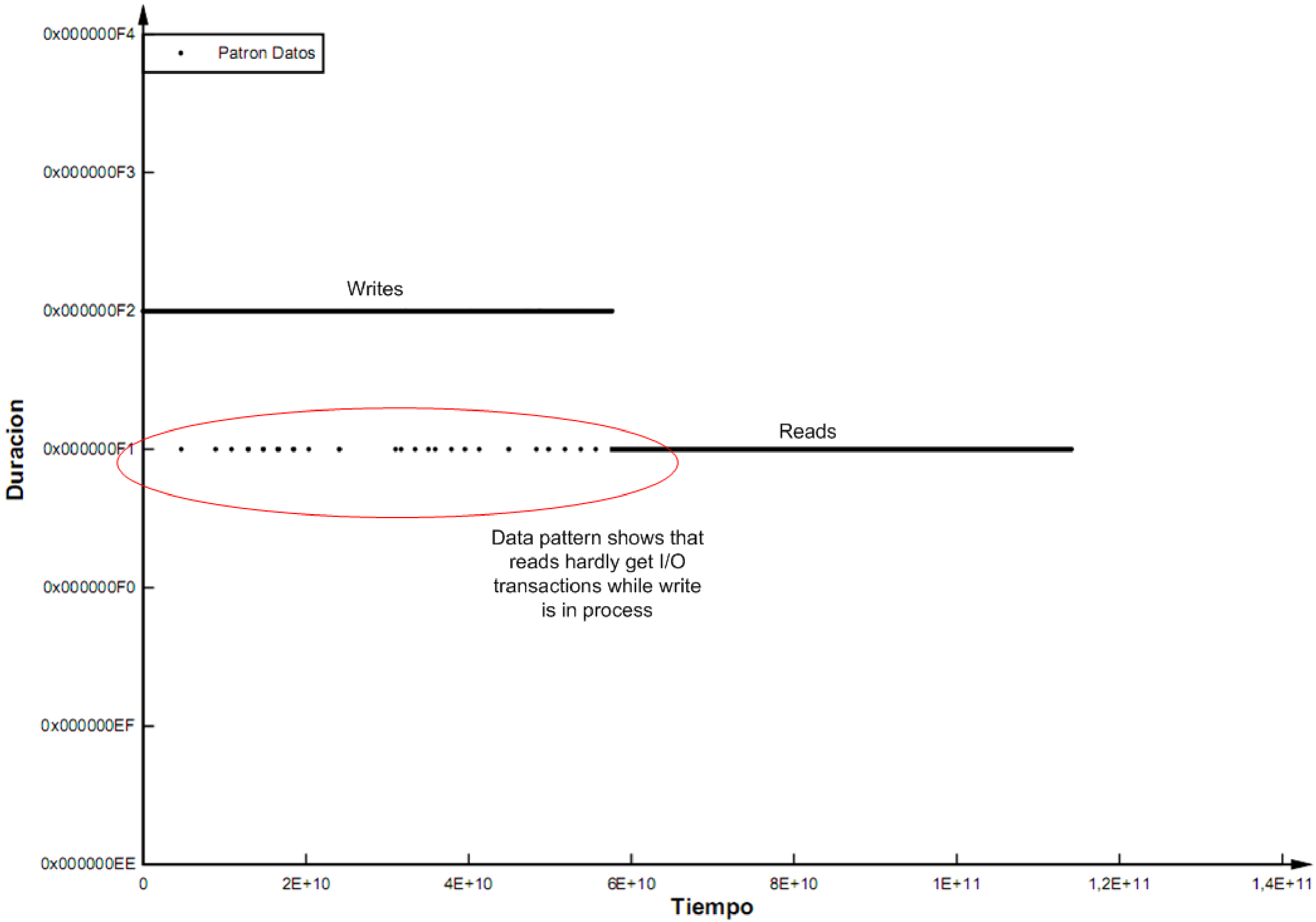
The Linux I/O Stack Diagram (version 1.0, 2012-06-20)
http://www.thomas-krenn.com/en/oss/linux-io-stack-diagram.html
Created by Werner Fischer and Georg Schönberger
License: CC-BY-SA 3.0, see http://creativecommons.org/licenses/by-sa/3.0/

I/O shaping by OS

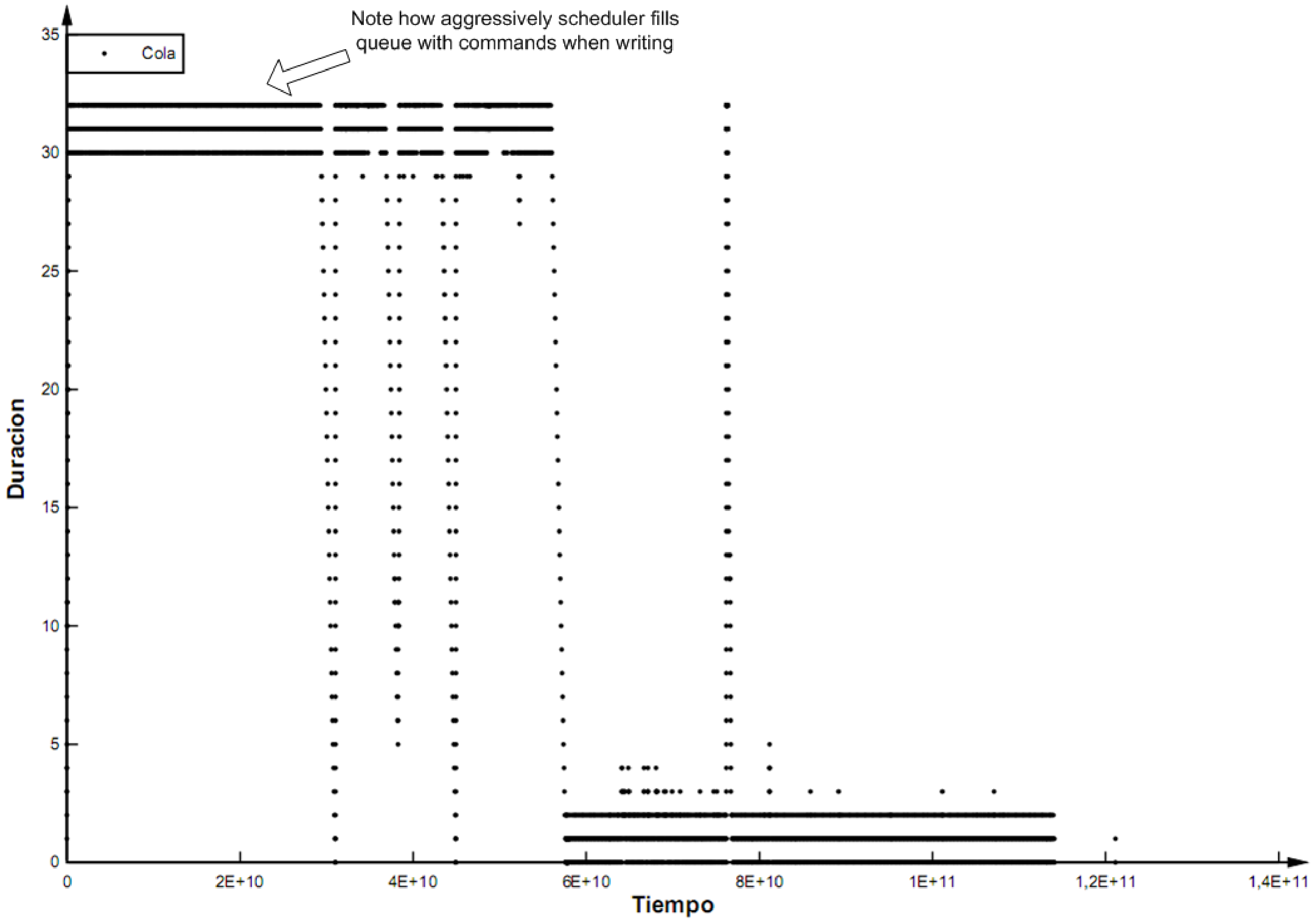
```
dt of=/scsi/prueba2 bs=512 limit=500m pattern=0x2ff22ff2 disable=verify dispose=keep

(after 3 seconds)

dt if=/scsi/prueba1 bs=512 limit=500m disable=compare

Total Statistics:
     Output device/file name: /scsi/prueba2 (device type=regular)
     Type of I/O's performed: sequential (forward)
        Data pattern written: 0x2ff22ff2 (read verify disabled)
     Total records processed: 1024000 @ 512 bytes/record (0.500 Kbytes)
     Total bytes transferred: 524288000 (512000.000 Kbytes, 500.000 Mbytes)
      Average transfer rates: 9003744 bytes/sec, 8792.719 Kbytes/sec
     Number I/O's per second: 17585.437
      Total passes completed: 1/1
       Total errors detected: 0/1
          Total elapsed time: 00m58.23s
           Total system time: 00m04.75s
             Total user time: 00m04.89s
               Starting time: Fri Nov 30 15:05:53 2012
                 Ending time: Fri Nov 30 15:06:51 2012


Total Statistics:
      Input device/file name: /scsi/prueba1 (device type=regular)
     Type of I/O's performed: sequential (forward)
           Data pattern read: 0x39c39c39 (data compare disabled)
     Total records processed: 1024000 @ 512 bytes/record (0.500 Kbytes)
     Total bytes transferred: 524288000 (512000.000 Kbytes, 500.000 Mbytes)
      Average transfer rates: 4707623 bytes/sec, 4597.288 Kbytes/sec
     Number I/O's per second: 9194.577
      Total passes completed: 1/1
       Total errors detected: 0/1
          Total elapsed time: 01m51.37s
           Total system time: 00m01.61s
             Total user time: 00m00.65s
               Starting time: Fri Nov 30 15:05:56 2012
                 Ending time: Fri Nov 30 15:07:48 2012
```

- Experiment: Simultaneous, sequential write and read of 500 MB files to parallel SCSI (SPI) disk
- OS = Linux SLES 10 SP2 (kernel 2.6.16)
- Disk’s queue depth = 32 commands

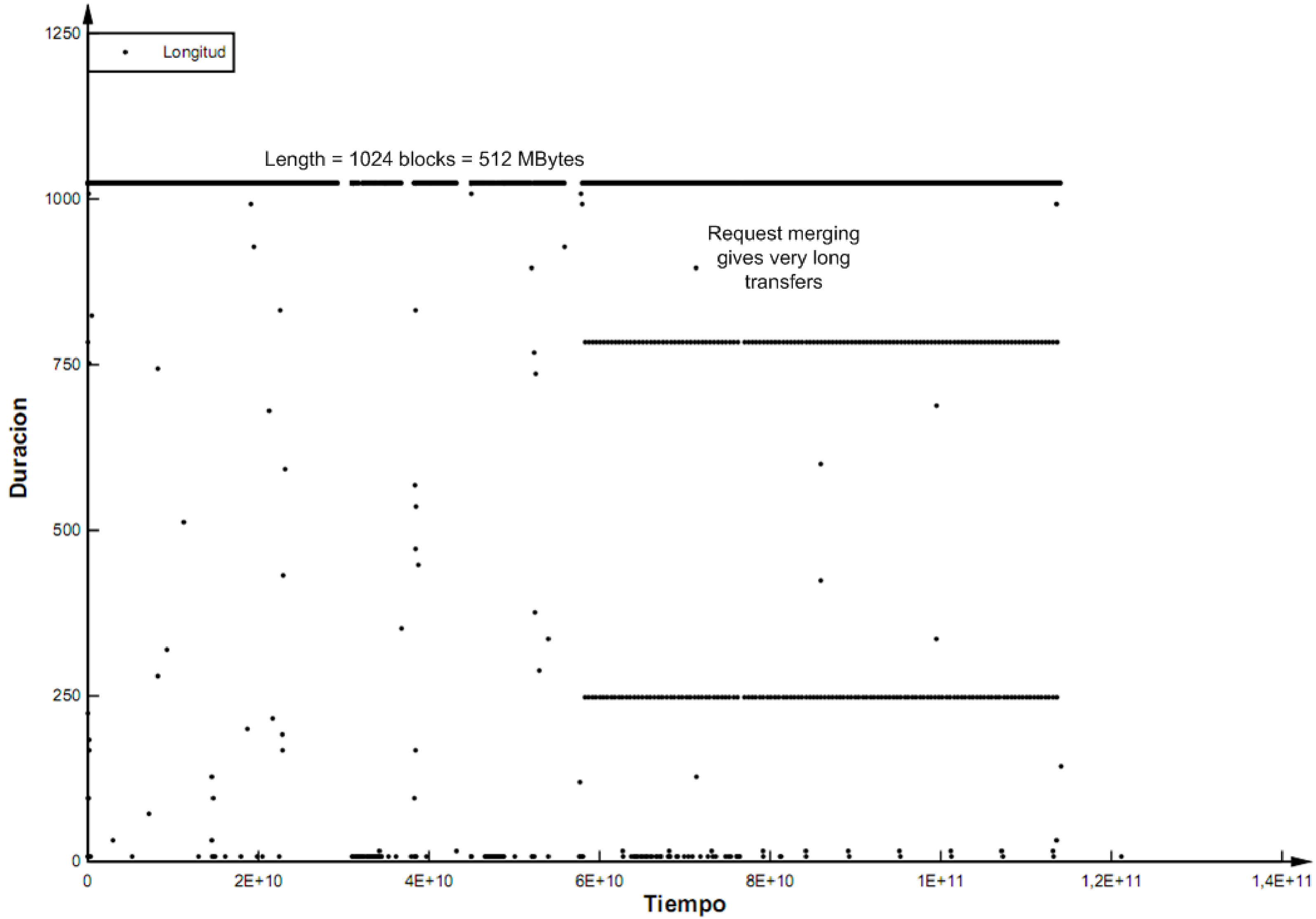
I/O shaping by OS



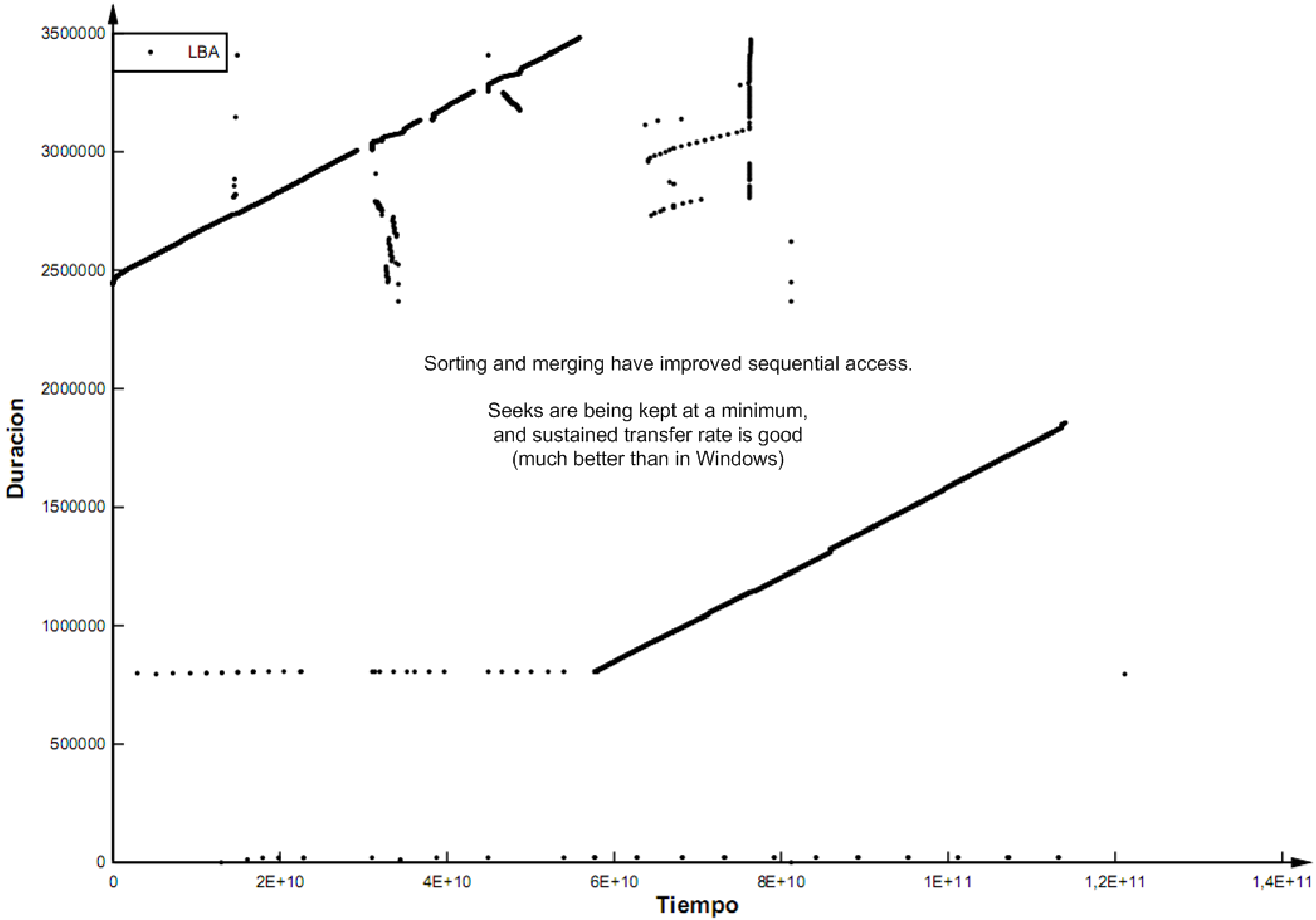
I/O shaping by OS



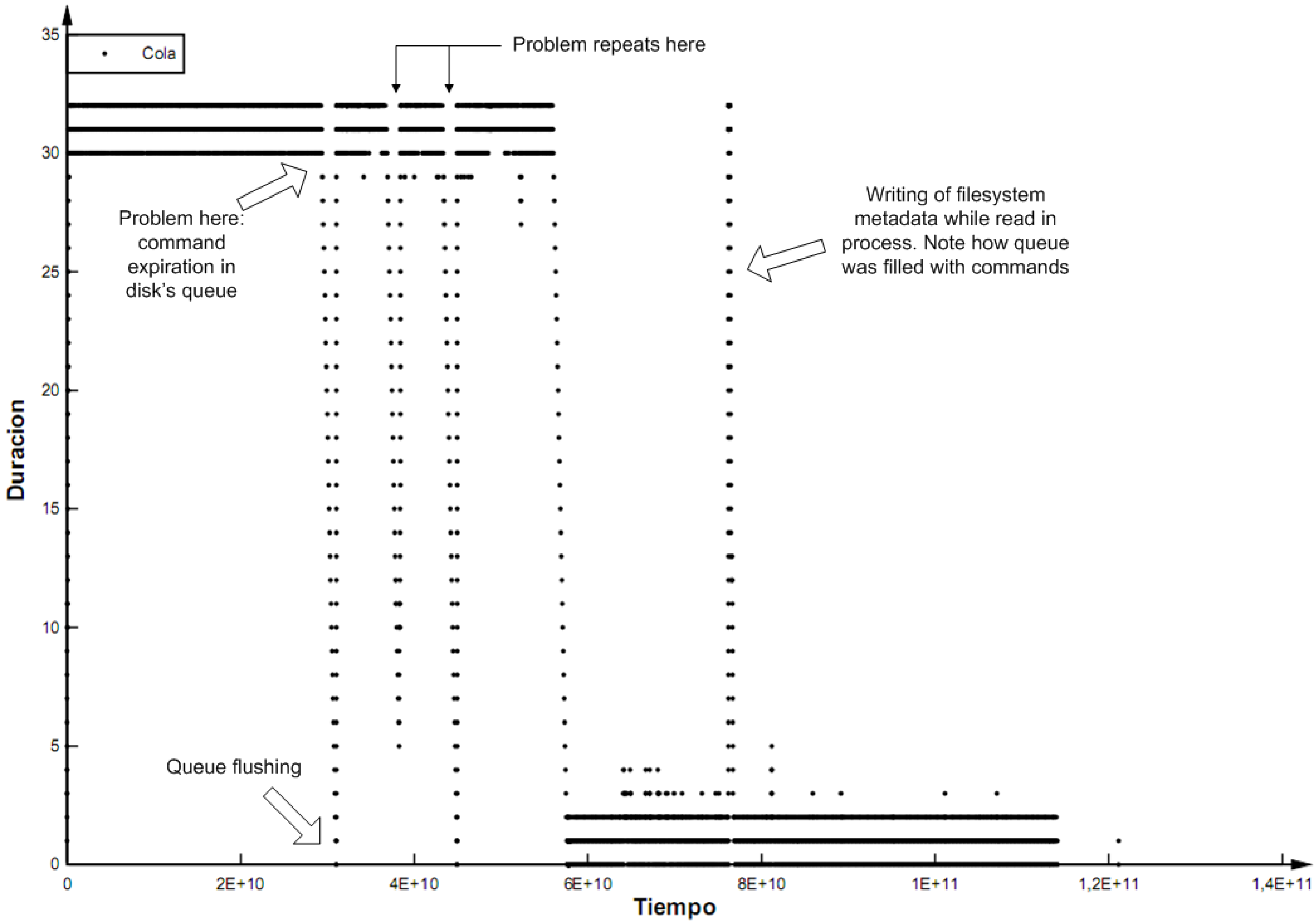
I/O shaping by OS



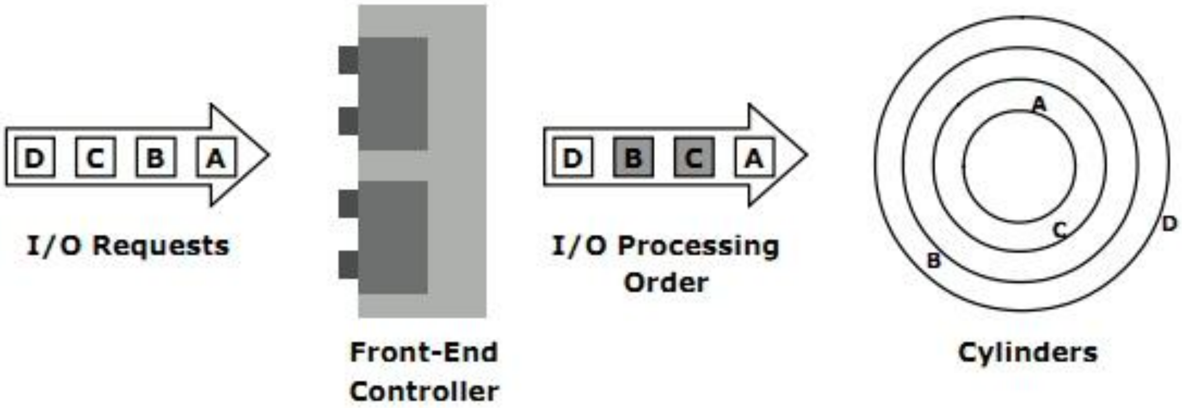
I/O shaping by OS



I/O shaping by OS



I/O shaping by OS



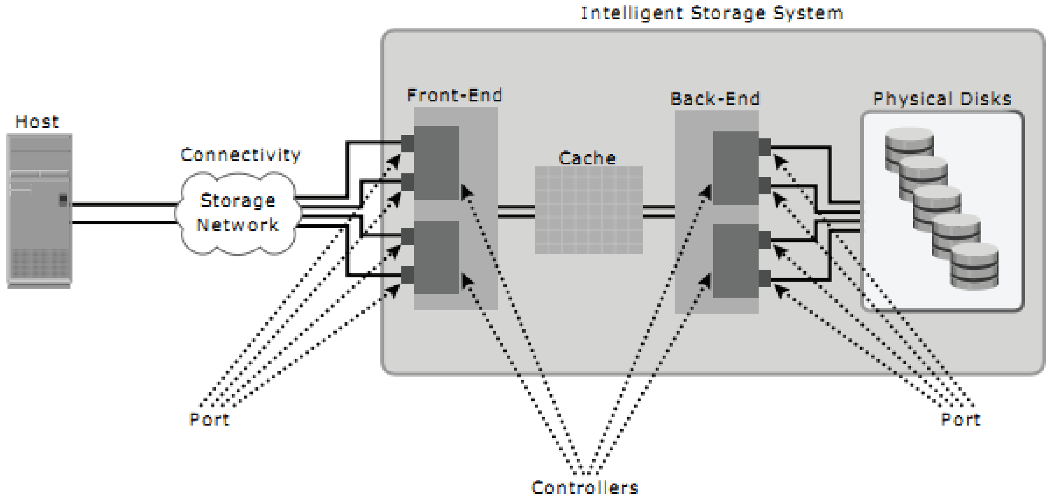
- Problem: Interaction of disk sorting algorithm with I/O scheduler sorting produces command expiration
  - In Figure, interaction of sorting algorithms moves head in sequence D-C-A
    - B is jumped, and as disk sorting algorithm won't back head, B will expire in queue
- Expiration requires inmediate execution
  - Typically translates into cascade of expirations and queue flushing
  - While flushing cache, no new commands accepted

I/O shaping by OS

```
procs -----------memory---------- ---swap-- -----io---- -system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  5    116   3744    468 225256    0    0     0 11872  289   95  7 13  0 80  0
 0  6    116   3732    468 225172    0    0     0  6560  271   74  9 10  0 81  0
 1  5    116   3660    468 225104    0    0     0 11920  289   92 10  9  0 81  0
 0  7    116   3236    464 225492    0    0     0  6160  271   82  6  7  0 87  0
 0  7    116   3200    464 225780    0    0     0     0  289   97  2  2  0 96  0
 0  6    116   3572    476 228052    0    0     8  8288  280  104 18 18  0 64  0
 0  6    116   3632    476 228052    0    0     0    56  304   59  0  0  0 100
0
 0  3    116   3868    476 228056    0    0     4   100  288   61  0  0  0 100
0
 0  3    116   4228    476 228056    0    0     0     0  290   23  0  1  0 99  0
 0  3    116   4480    476 228060    0    0     4     0  283    8  0  1  0 99  0
 0  3    116   4860    476 228064    0    0     4     0  292   26  0  1  0 99  0
 0  3    116   5284    476 228064    0    0     0     0  270   11  0  0  0 100
0
 1  6    116   3636    508 228484    0    0     4 44124  292  238 24 30  0 47  0
 1  6    116   4336    560 226612    0    0     4 18952  274  286 46 54  0  0  0
 0  3    116   3576    576 226020    0    0     0 31700  294  113 22 21  0 57  0
```

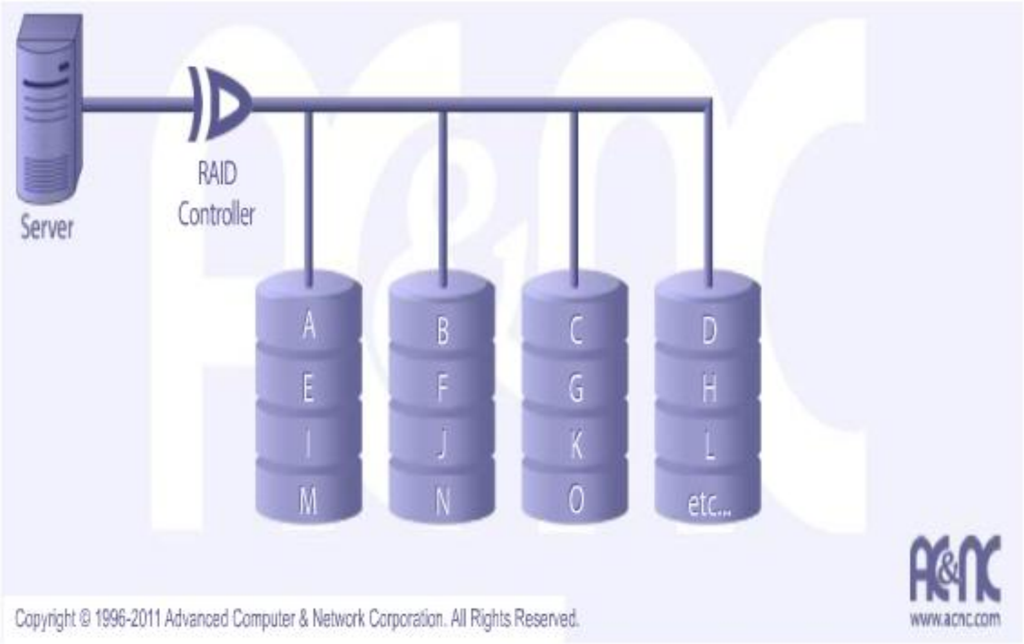
- Note performance dropout around second 30
- Disk performance has not dropped
  - Disk it’s writing as fast as it cans
- Performance has dropped at the OS level!!
  - You can’t queue new SCSI I/O requests
  - You can’t read, even if you really need it (and GUI spot-freezes) !!

RAID: increasing performance and reliability



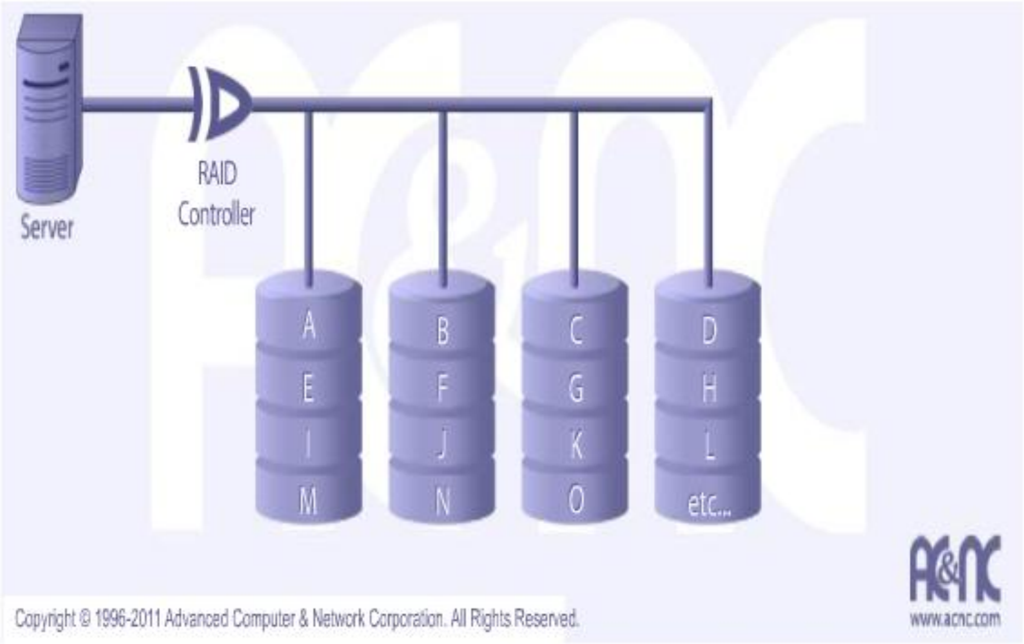
- Frequently, IOPS, transfer rate or capacity requirements are larger than anything a single disk can serve
- Solution: use a smart controller to:
  - Exercise simultaneously more than one drive
  - Hide this parallel operation to server, showing just a SCSI LUN from front-end
  - Calculate and insert redundant ECC parity info in writes to protect info
- Solution known as RAID (Redundant Array of Independent Disks)

RAID0: Raw performance, no redundancy



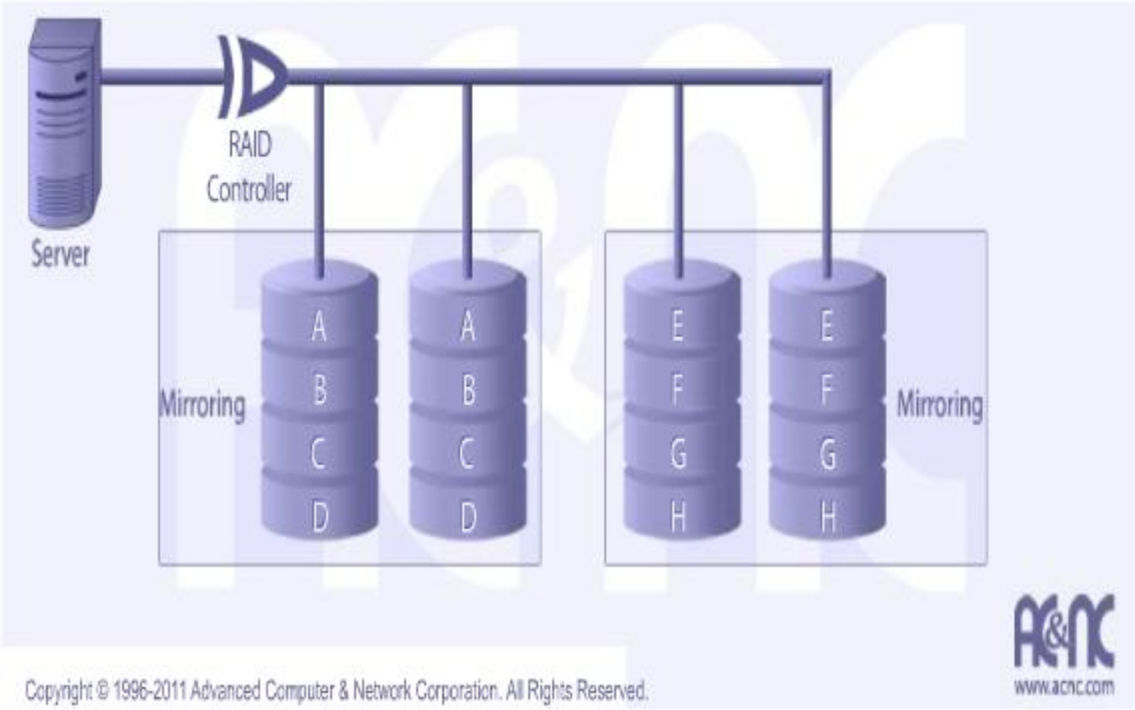
- RAID0 makes striping over several disks
  - Large transfers read/written simultaneously over all disks
  - Transfer speed greatly increased
- Small transfers still exercise only one disk
  - Carefully choosing (small) stripe size, random I/O will distribute IOPS over all disks

RAID0: Raw performance, no redundancy



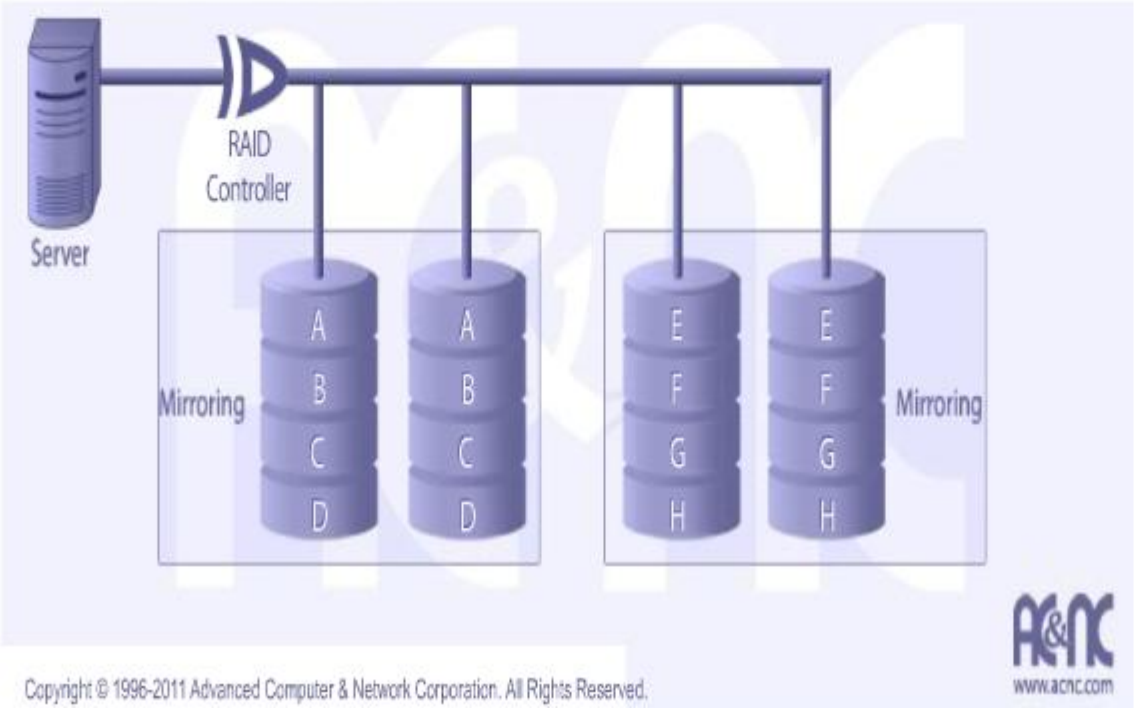
- Problem: NO redundancy
  - Just a single disk failure means all data is lost

RAID1: Redundancy for HA



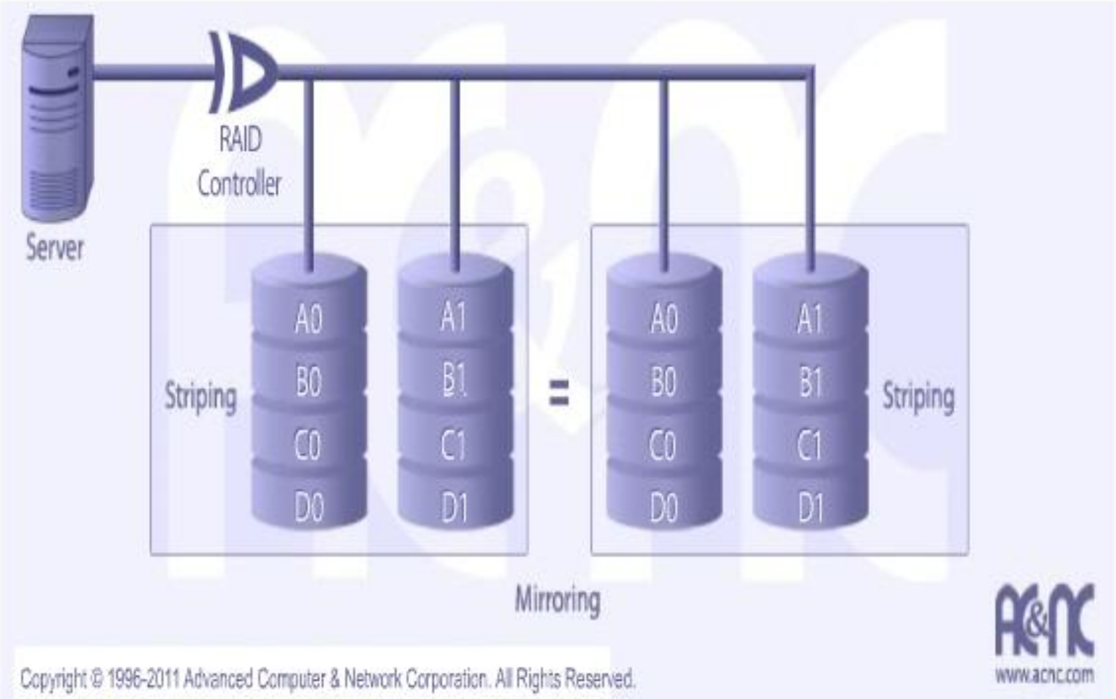
- RAID1 mirrors every disk
  - Reads performed only on active disk
  - Writes performed simultaneously on both disks
  - Failover switching on drive failure is almost instantaneous

RAID1: Redundancy for HA



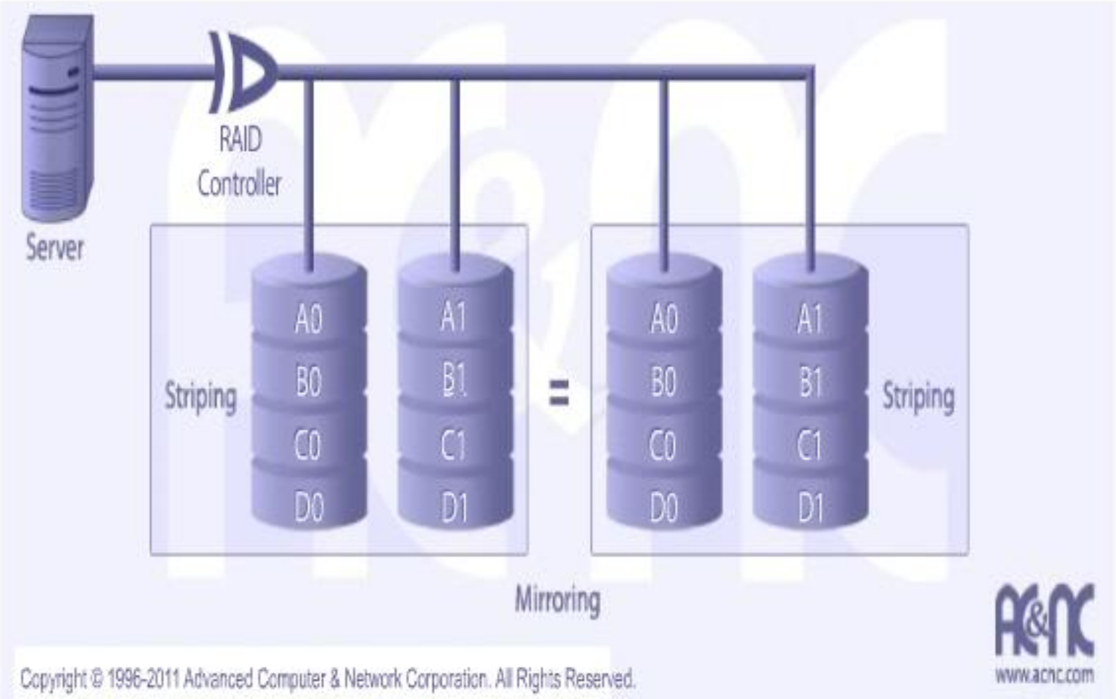
- Problems
  - Expensive, as loses half the disks to redundancy
  - No advantage for IOPS or transfer rate
    - No parallel, stripped read o writes

RAID0+1: Mirrored stripping



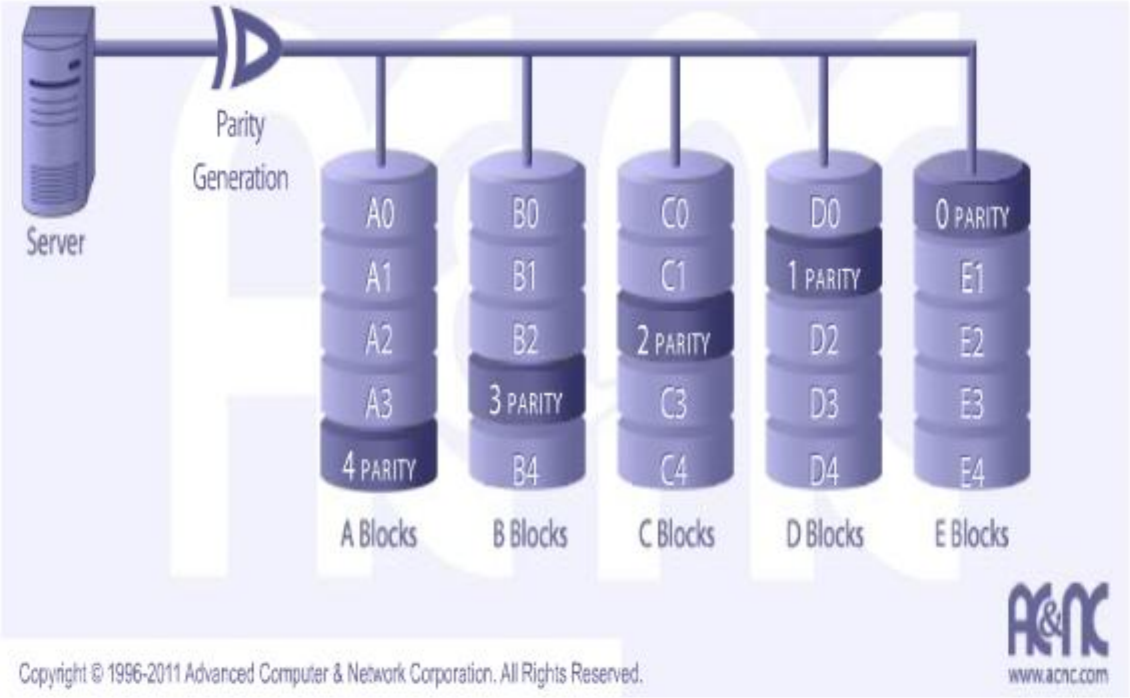
- RAID0+1 combines mirroring and striping
  - Mirroring over striped arrays
  - Offers RAID0 performance and RAID1 protection
  - Twice as expensive as RAID0

RAID10: Striping over mirror



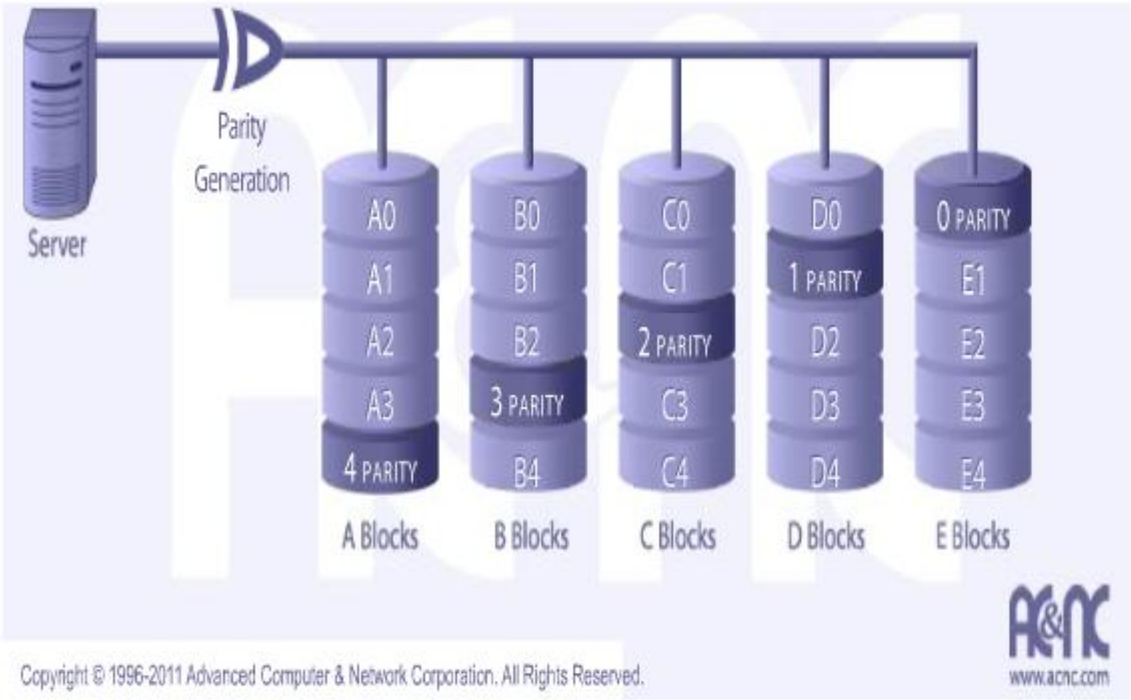
- RAID10 combines mirroring and striping
  - Striping over mirrored arrays
  - Offers RAID0 performance and RAID1 protection
  - Twice as expensive as RAID0

RAID5: Distributed parity



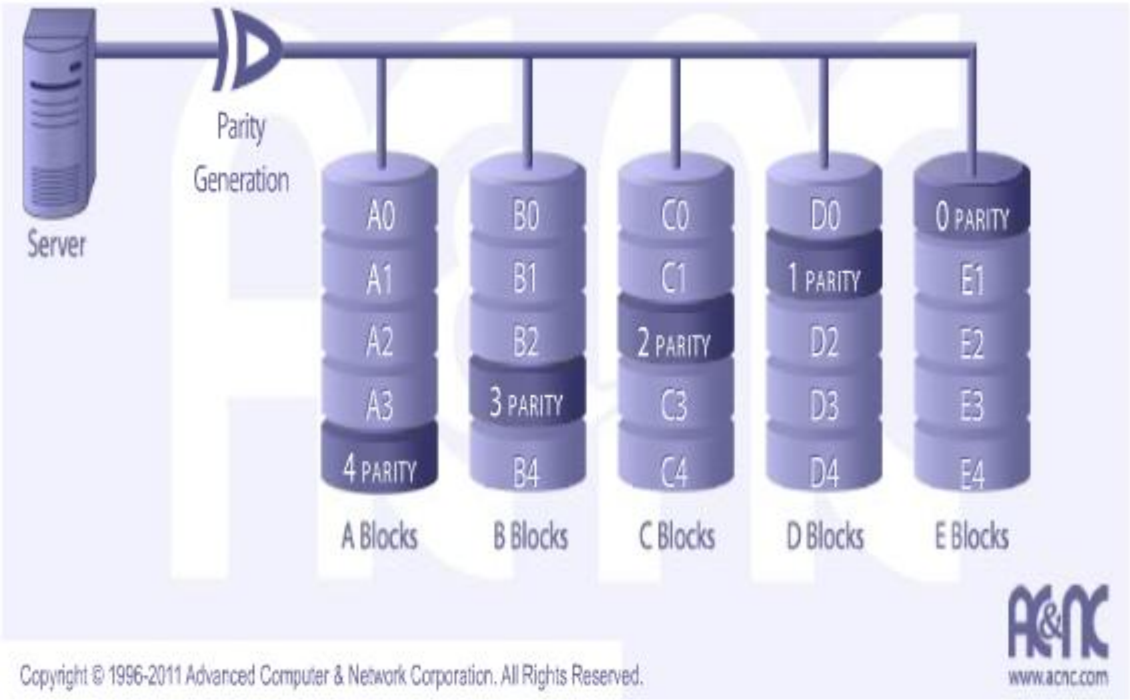
- RAID5 is compromise between performance, redundancy and cost
  - Controller stripes data over disks as in RAID0
  - Controller calculates and writes on stripe additional parity block
    - Parity blocks spreads across all disks
    - If all parity blocks are written on same disk, we have RAID3

RAID5: Distributed parity



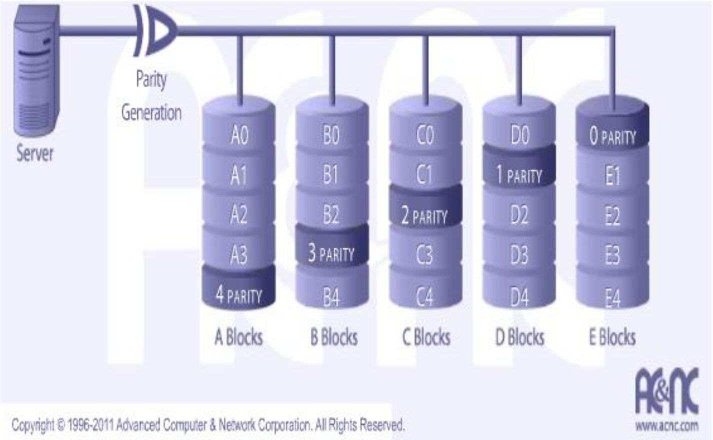
- RAID5 can survive a single disk failure
  - Requires hot spare (unused disk, kept in array waiting for failure)
  - Controller uses parity to reconstruct data in failed disk, and writes it onto spare
    - Can take hours on busy arrays

RAID5: Distributed parity



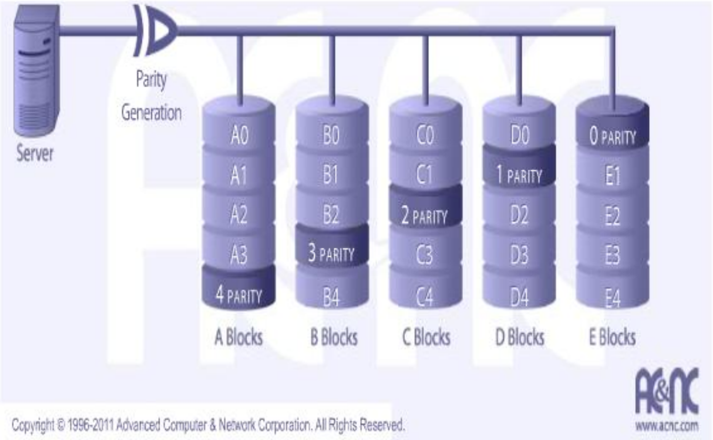
- In RAID5, a second disk failure while recovering data from first disk failure loses all data
  - Problem for modern RAIDs with large capacity (2TB) SATA disks
    - Reconstruction for 2TB while array is in production takes very long
    - And probably there is another disk of same batch in array!!

RAID5: Compromise on performance



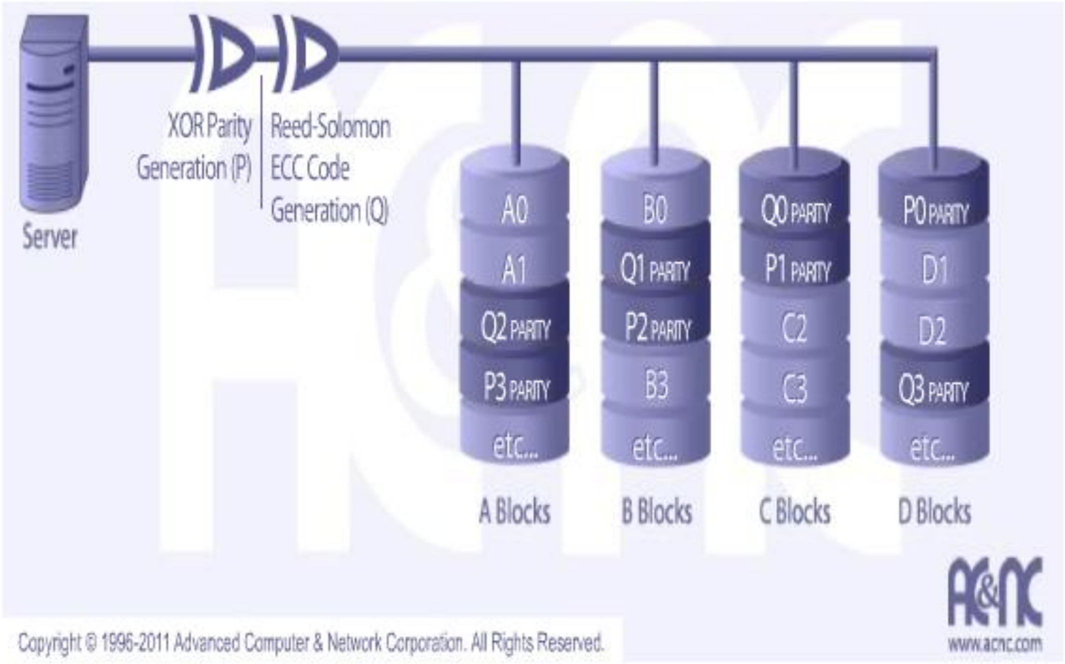
- RAID5 offers data protection at lower cost than RAID1
  - Loses equivalent capacity of only 1 disk
- But RAID5 must compromise on performance
  - Parallel striped read makes RAID5 read performance equal to RAID0
  - However, write requires a Read-Modify-Write cycle
    - Controller reads data block to be modified, and parity block for stripe
    - Controller recalculates parity block
    - Controller writes modified data block and modified parity block
    - Total: 4 IOPS to write a single block

RAID5: Compromise on performance



- Performance of RAID5 can thus be troublesome for small random writes
  - Require high IOPS disks to be effective for heavy-duty database use
  - With cheaper SATA disks, may require *short-stroking*
    - Raise IOPS by partitioning disks using just outer zones
    - Reduces disk head travel, reducing thus seek times
    - Uses disk in zones where transfer speed is maximal

RAID6: Double parity

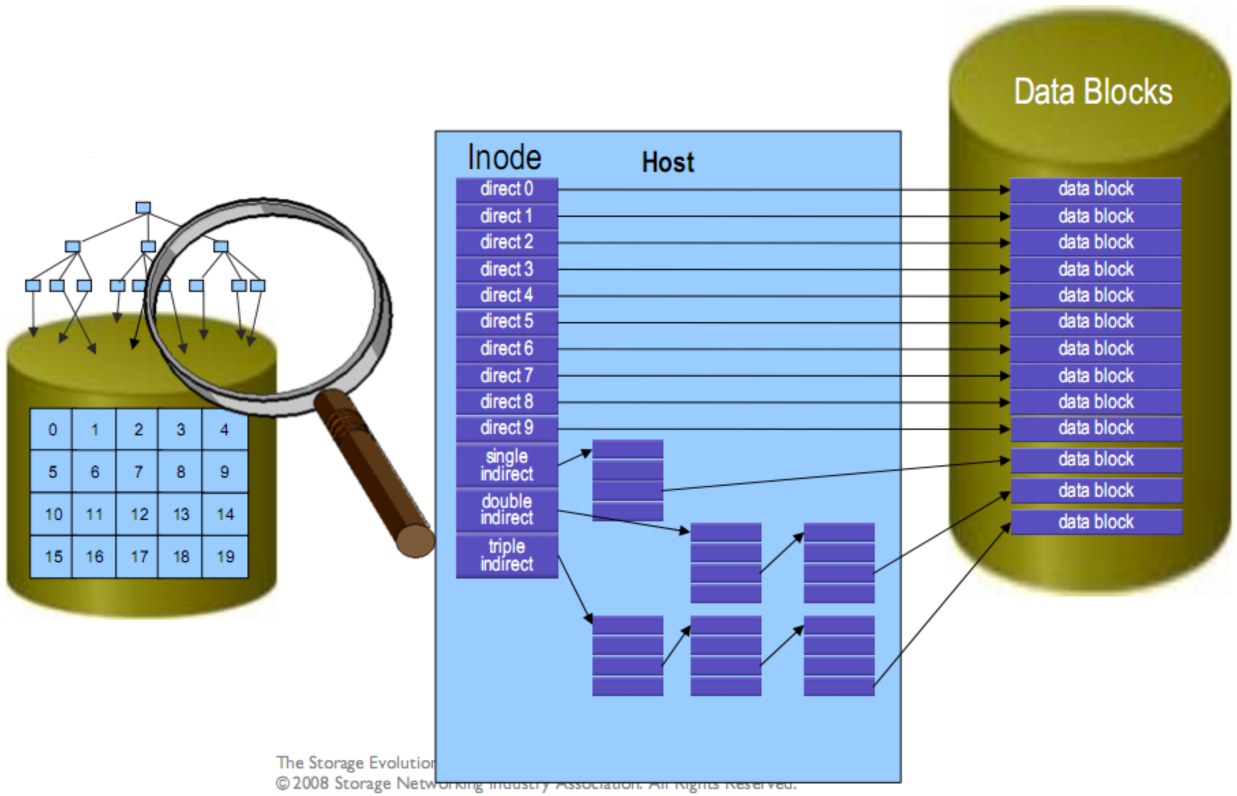


- RAID6 is RAID5 with a second, additional parity block
- Can survive two disk failures without losing data
  - Required by large reconstructions times for large capacity SATA disks
- Loses equivalent capacity of two disks to hold parity blocks
- Writing performance slower than RAID5 (6 IOPS per write)

Object storage

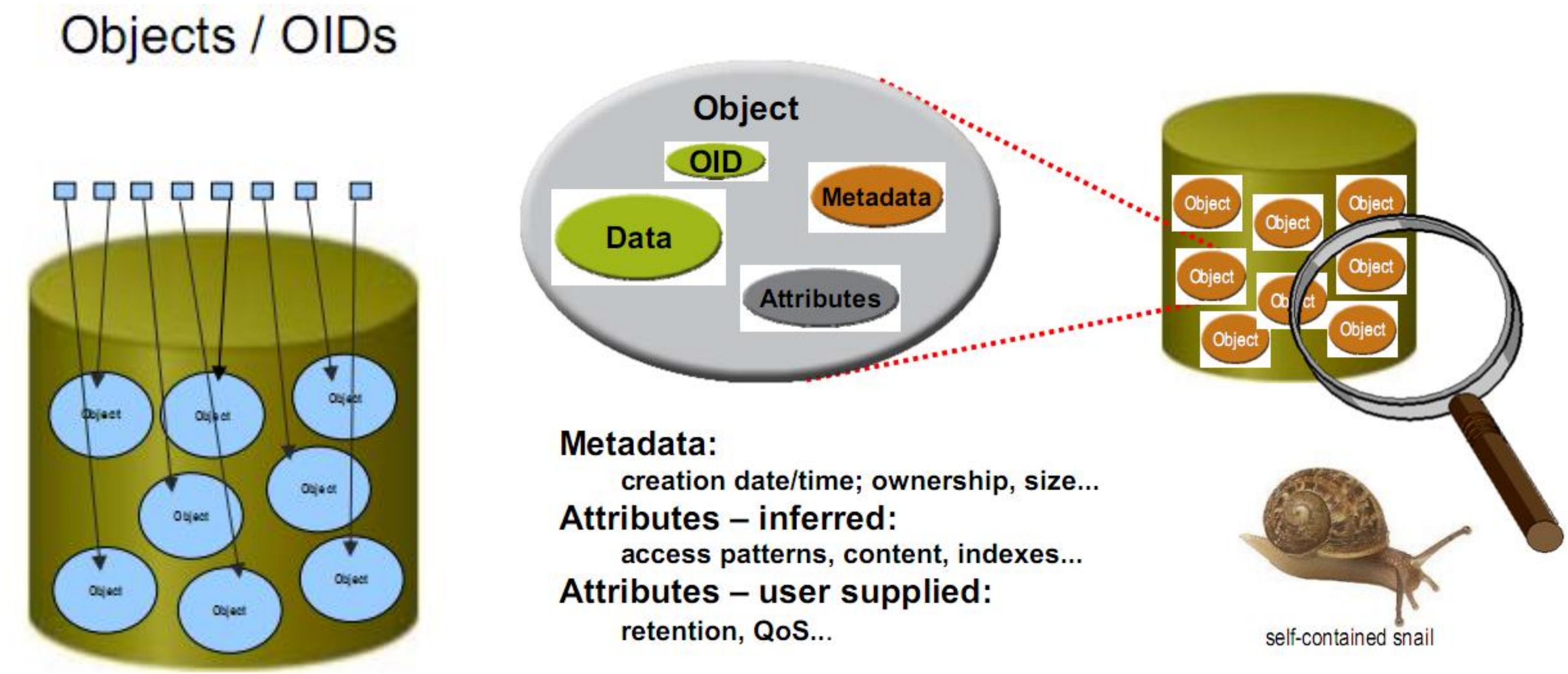
- Today, RAID is somehow reaching its limits
  - Advent of large capacity (2+ TB) SATA disks means reconstruction takes forever
    - Fear of second failure while reconstructing
    - Heavy impact on performance, both if RAID6 and when reconstructing
- Also, cost of RAID controllers and physical rack is not trivial
- Object-based storage provides alternative to RAID

Object storage



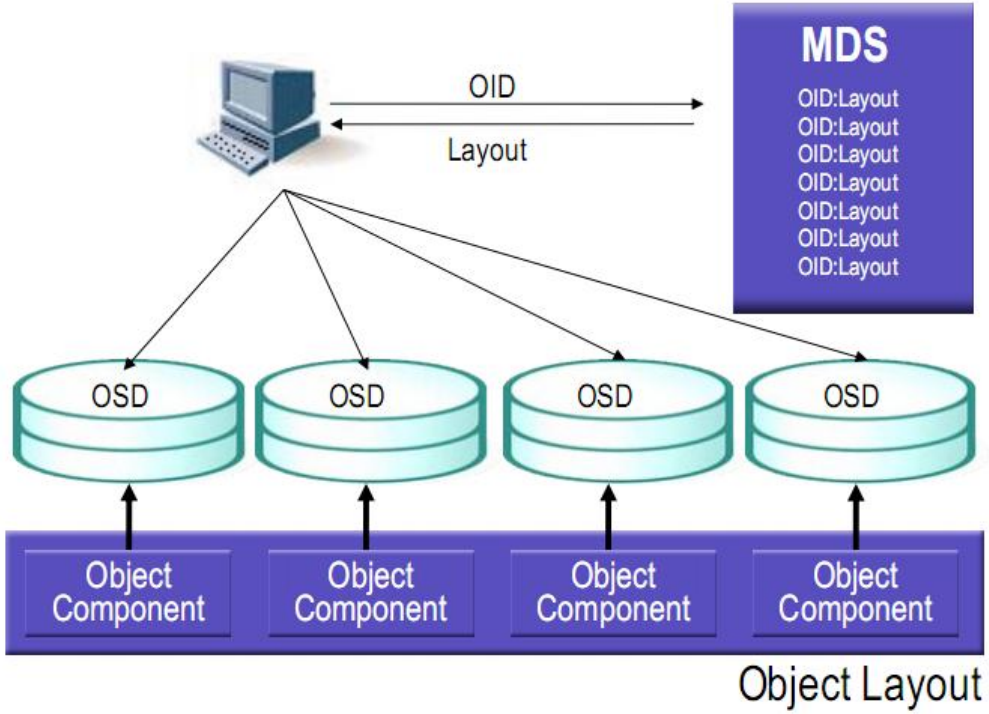
- Traditional inode-based filesystem stores block addresses as metadata
  - Addresses are then used as LBAs in block I/O commands

Object storage



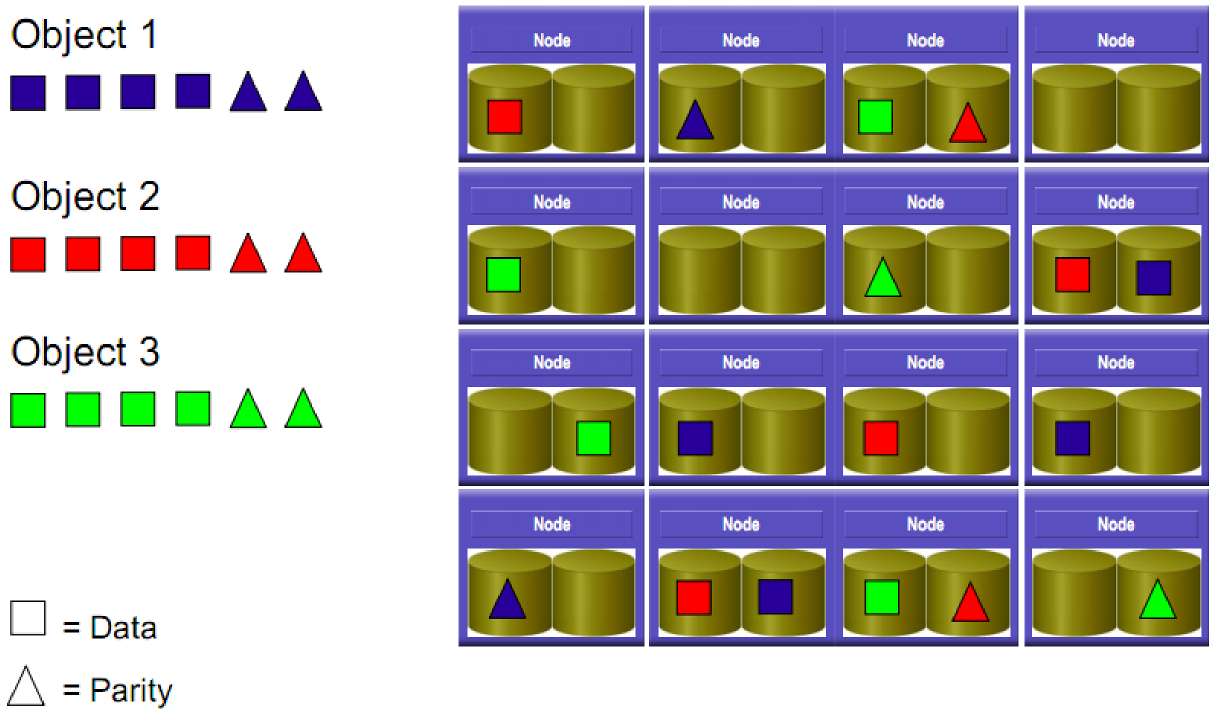
- In object-oriented storage, filesystem no longer knows about blocks, but about objects
  - Object has unique ID
  - Object is self-contained
    - Holds physical storage layout as metadata
- Requires use of object database
  - Metadata about objects contained in metadata server (MDS)

Object storage



- Data stored as chunks in storage nodes
  - Named OSD = *Object-based Storage Devices*
- Client gets physical layout of chunks from metadata server MDS
  - Layout used to do block I/O to data in OSD nodes

Object storage

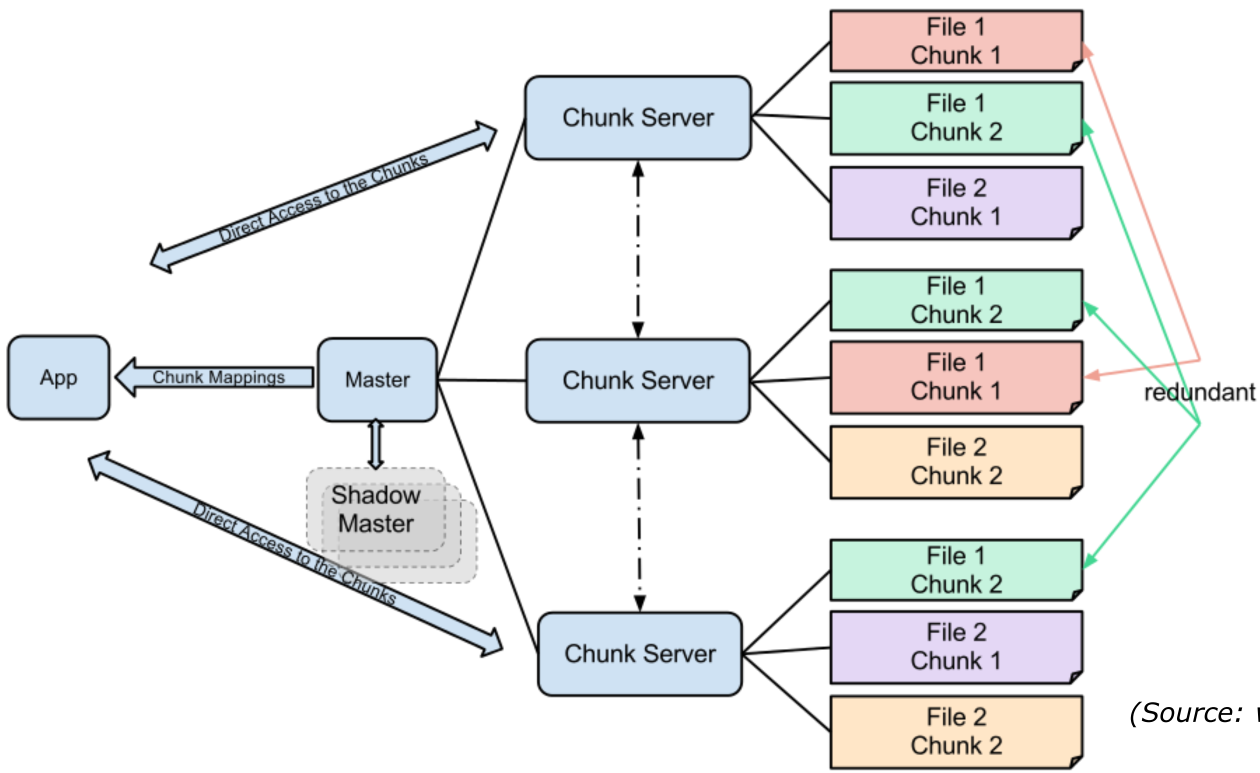


- Distribution to improve IOPS and replication can be done just storing several copies of each chunk in separate OSD nodes
  - LOCKSS model = *LOts of Copies Keep Stuff Safe*
- Individual nodes can use direct-attach storage (SAS, SATA)
  - Nodes way cheaper than networked SAN/NAS storage

Object storage: RAIN

- Model based on replication on OSD nodes known as *RAIN*
  - Redundant Array of Inexpensive/Independent Nodes

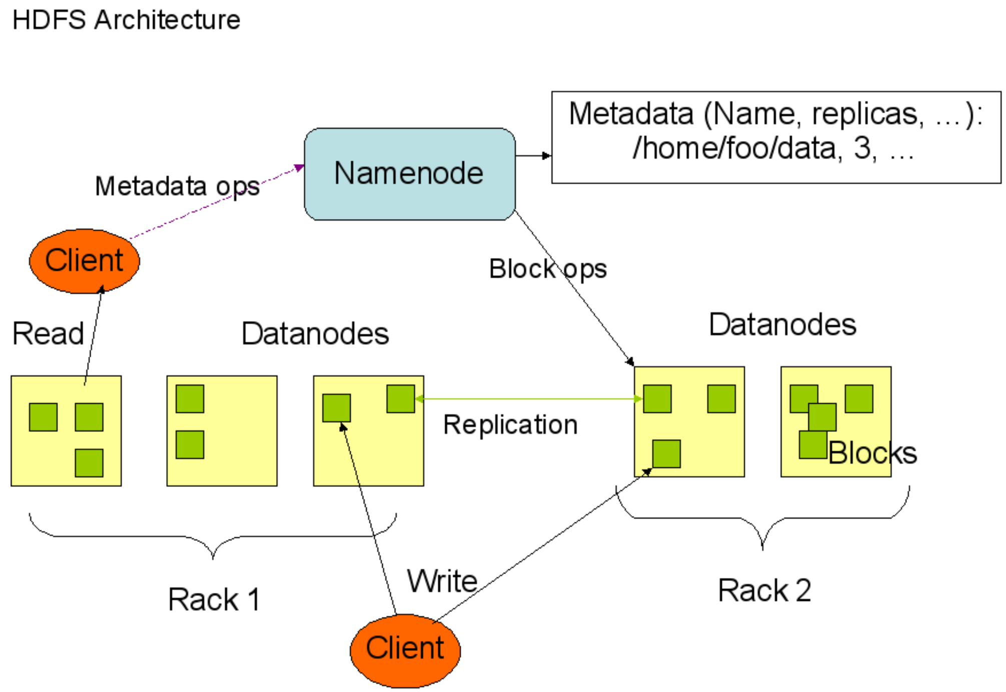
Object storage: GFS and HDFS



(Source: wikipedia)

- This storage paradigm is used by the Google File System (GFS) and the Hadoop Distributed File System (HDFS)
  - Figure shows GFS architecture

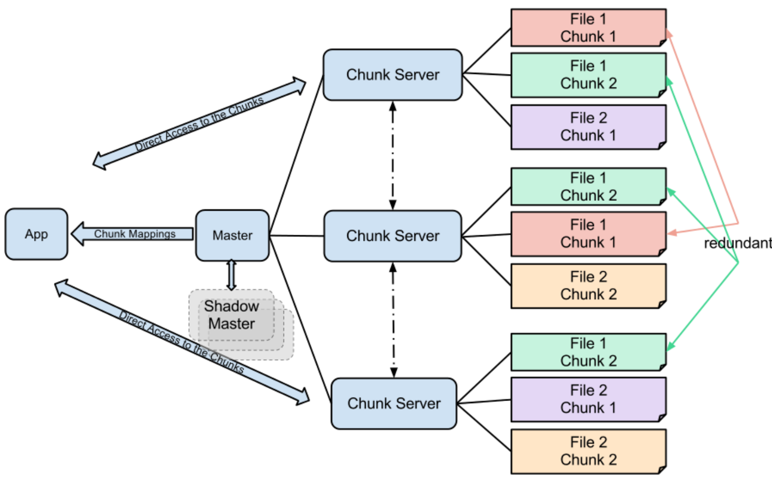
Object storage: GFS and HDFS

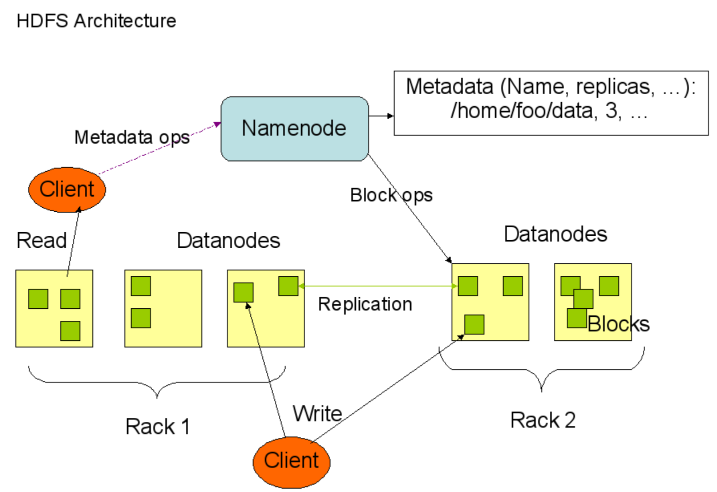


(Source: wikipedia)

- HDFS is open-source development based on GFS

Object storage: GFS and HDFS



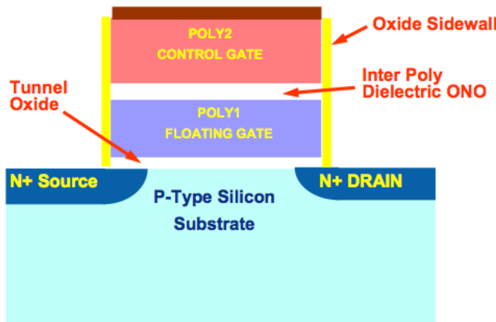


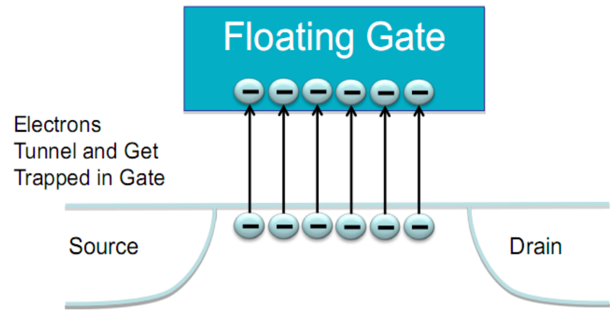
- GFS or HDFS allow scaling to big data with low hardware costs
  - Work best when filesystem needs to store just a moderate number of large files, updated frequently and concurrently by multiple nodes
  - Resilient to frequent breakdown of nodes
    - Event all too common in Google's operation

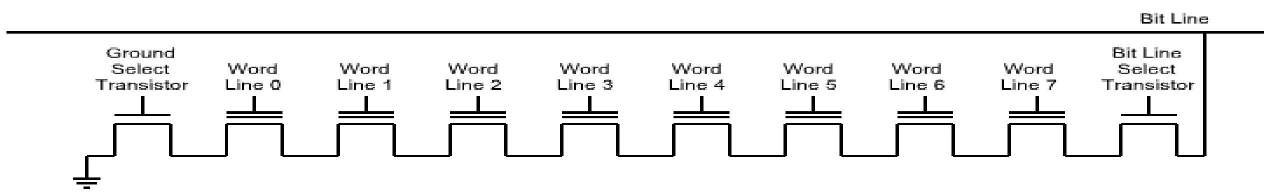
SSD: Solid-State Drives

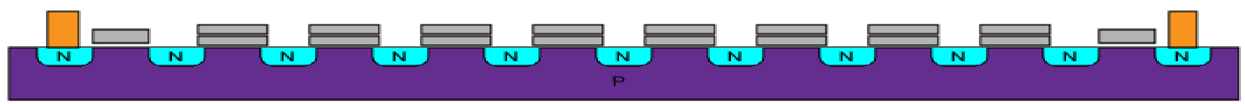
- Flash memory-based disk drives are becoming increasingly popular
  - Are a must for consumer-grade products like tablets and laptops
  - Also increasingly vital for use on professional storage systems
    - Allow overcoming the not-enough-IOPS problem

SSD internals





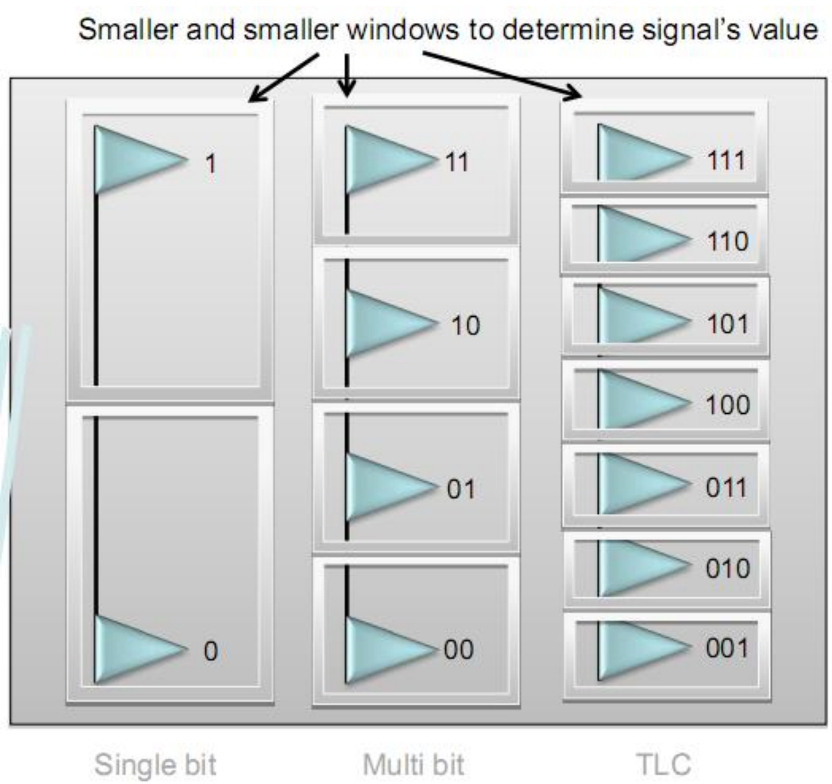


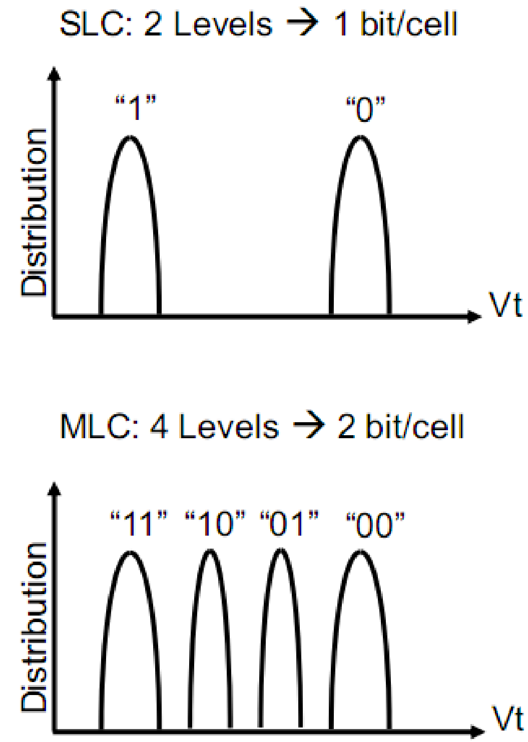


(Source: wikipedia)

- Flash cell stores information as charge in floating gate
- Programming the cell will inject electrons in gate
- State of bit checked applying V to all control gates, except that bit
  - Bit line is pulled down is gate is programmed

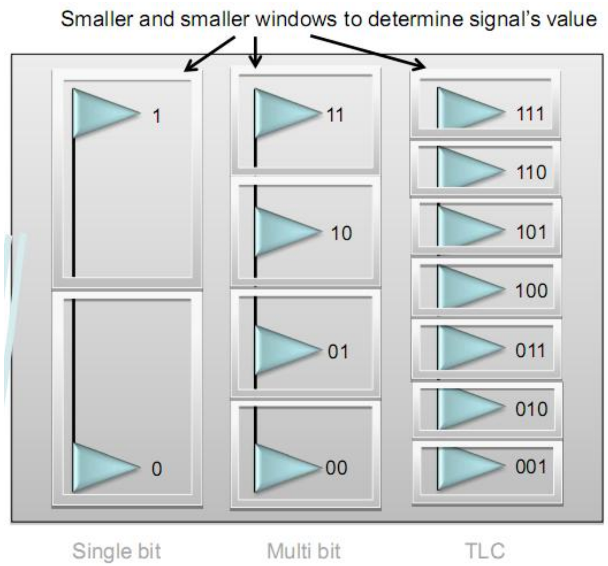
SSD internals

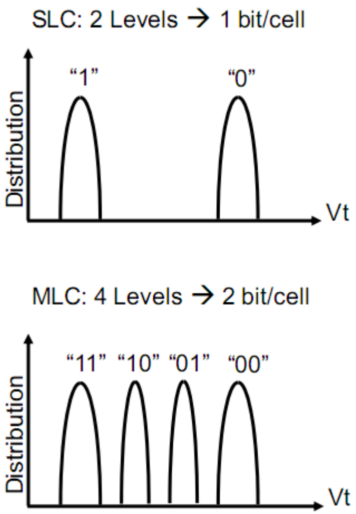




- Cells can hold more than one bit
  - Just discriminate more than two levels when reading charge status

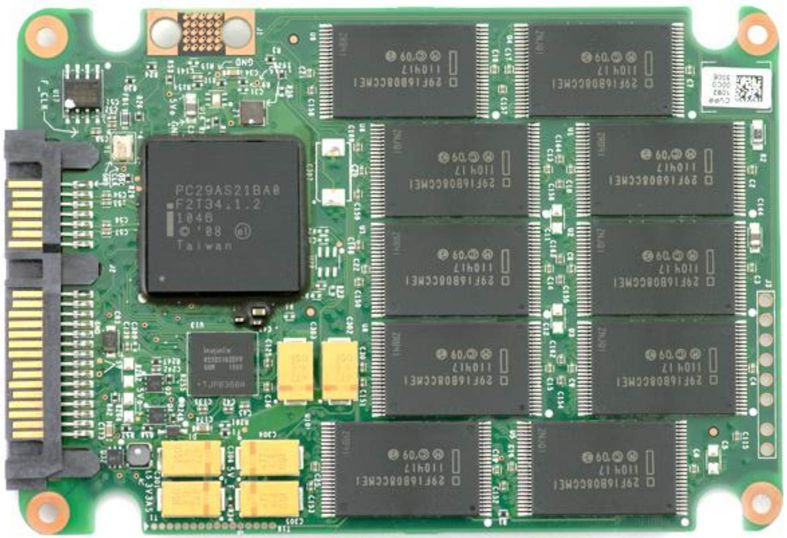
SSD internals

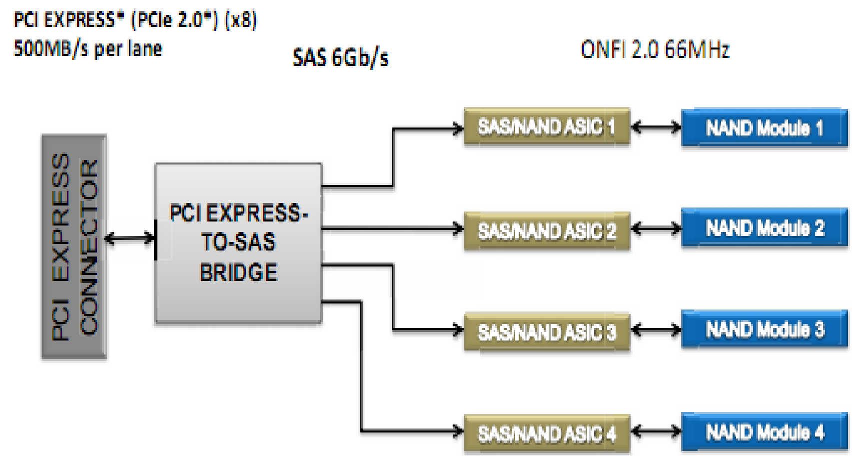




- SLC = Single Level Cell
  - Just two levels = 1 bit/cell (cost ~ 7 x MLC)
- MLC = Multi Level Cell
  - Four levels = 2 bits/cell
- TLC = Triple Level Cell
  - Eight levels = 3 bits/cell

SSD internals





- Reading/writing cell in a single NAND module is VERY fast
  - Typical read/write latencies around 60 – 90 μs
- Disk design further improves transfer speed and IOPS with parallel access to multiple NAND modules

SSD internals

Bandwidth[4]	Sustained Sequential Reads (up to) ▪ 40GB: 200 MB/s ▪ 80GB: 270 MB/s ▪ 120GB: 270 MB/s ▪ 160GB: 270 MB/s ▪ 300GB: 270 MB/s ▪ 600GB: 270 MB/s	Sustained Sequential Writes (up to) ▪ 40GB: 45 MB/s ▪ 80GB: 90 MB/s ▪ 120GB: 130 MB/s ▪ 160GB: 165 MB/s ▪ 300GB: 205 MB/s ▪ 600GB: 220 MB/s
Read Latency[5]	75 µs	
Write Latency[5]	90 µs	
Random I/O Operations per Second (IOPS)[1]	Random 4KB Reads (up to) ▪ 40GB : 30,000 IOPS ▪ 80GB: 38,000 IOPS ▪ 120GB: 38,000 IOPS ▪ 160GB: 39,000 IOPS ▪ 300GB: 39,500 IOPS ▪ 600GB: 39,500 IOPS	Random 4KB Writes (up to) ▪ 40GB: 3,700 IOPS ▪ 80GB: 10,000 IOPS ▪ 120GB: 14,000 IOPS ▪ 160GB: 21,000 IOPS ▪ 300GB: 23,000 IOPS ▪ 600GB: 23,000 IOPS

- Lack of moving parts and low latency gives blazingly fast device

SSD internals

Random Read and Write Input/Output Operations Per Second (IOPS)

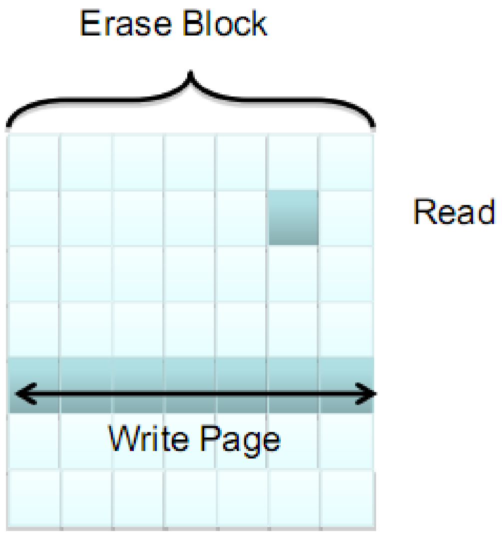
Specification[1]	Unit	400 GB	800 GB (Default)	800 GB (Max Performance[2])
Random 4 KB Read (up to)	IOPS	90,000	180,000	180,000
Random 4 KB Write (up to)	IOPS	38,000	75,000	75,000

Maximum Sustained Sequential Read and Write Bandwidth

Specification[1]	Unit	400 GB	800 GB (Default)	800 GB (Max Performance[2])
Sequential Read (up to)	MB/s	1000	2,000	2,000
Sequential Write (up to)	MB/s	750	1,000	1,500

- Enterprise-grade SSDs provide IOPS and sustained transfer BW in another league
  - Typical IOPS of single HDD ~ 100 – 140 IOPS
- No need to turn platter or move heads, so power requirements also minimal
  - Heat dissipation thus very low

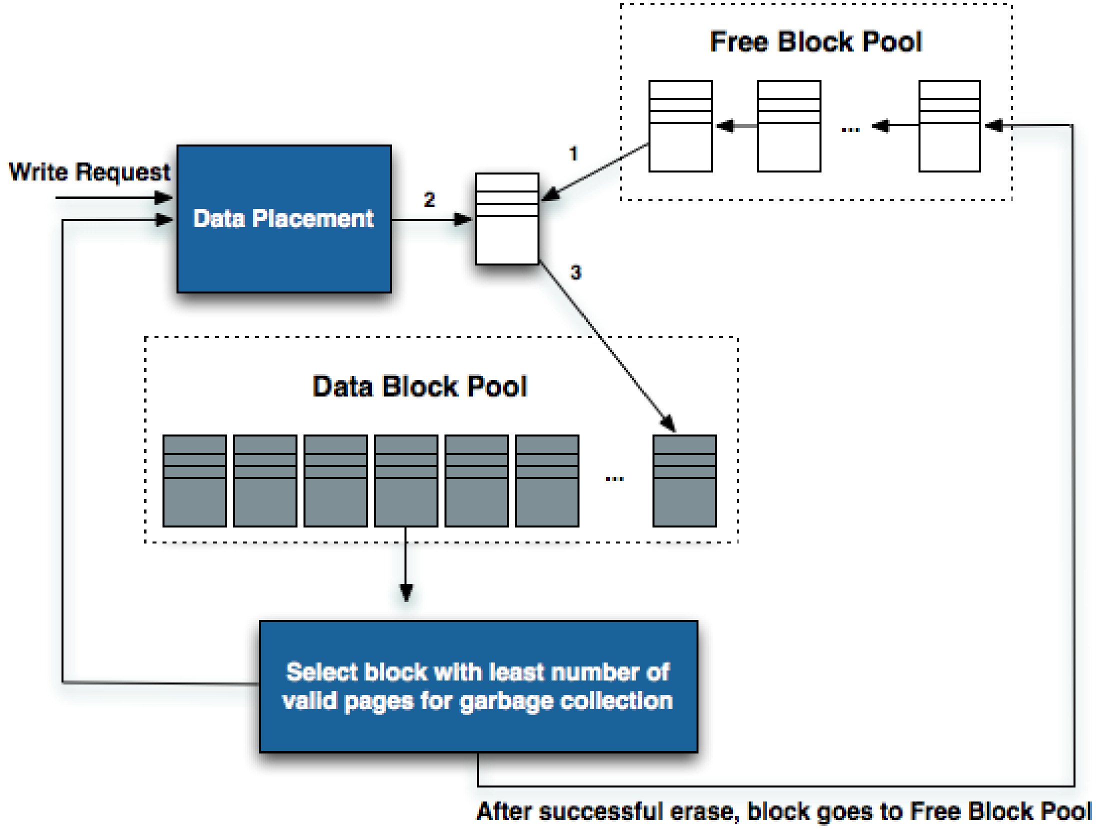
SSD: So, where's the catch?



Typical Specification	SLC	MLC
Bits per Cell	1	2
Page Size (K)	4	4
Pages/Block	64	128
Page Program (us)	250	900
Random Read (us)	25	50
Block Erase (ms)	2	2

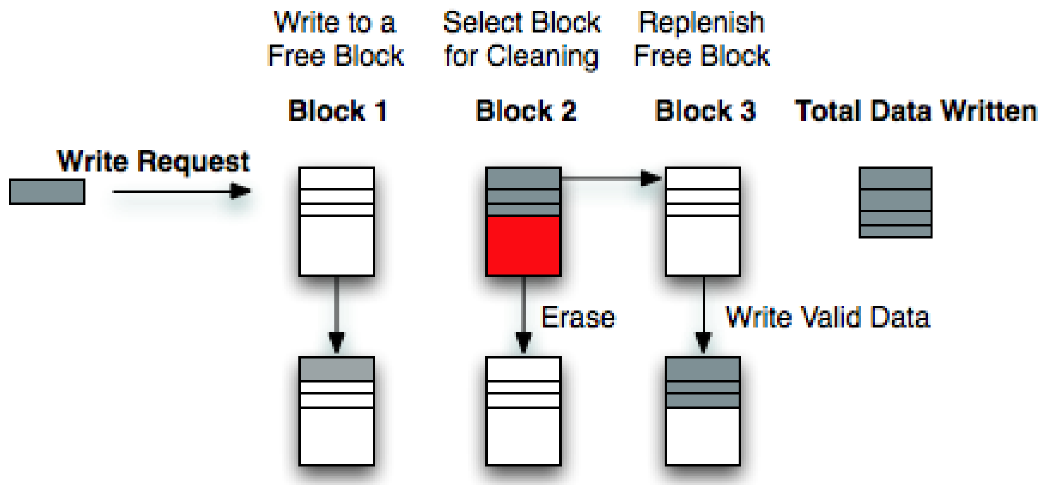
- Page is smallest structure which can be read or written
  - Typically, 4K
- To program a cell, it must be first erased
  - Block is smallest structure which can be erased
  - Typically 1 block = 64 or 128 pages = 256 or 512 KB
  - Erase time around 1-2 ms
  - If filesystem deletes content, page is marked non-valid, but **unusable until block is erased**

Catch: write amplification



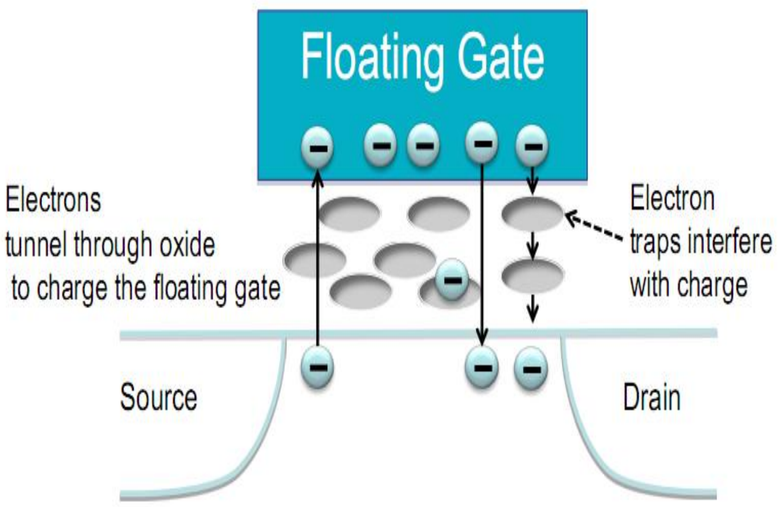
- Once you run out of clean blocks, you need to free unused, non-valid space to refill Free Block Pool
  - Known as *Garbage collection*

Catch: write amplification



- Write request triggers garbage-collecting and page migration
  - Valid pages are written on empty block to free old block for erasing
    - Possible because maker overprovisions memory (real size > nominal size)
  - Controller hides real placement of data, dynamically re-mapping real cells to SCSI-level LBAs
  - Write amplification: data written to flash / data written by host
    - Typical values between 1.1 and 3
  - Write amplification slows down performance

SSD: So, where's the catch?



Typical Specification	SLC	MLC
Bits per Cell	1	2
Page Size (K)	4	4
Pages/Block	64	128
Page Program (us)	250	900
Random Read (us)	25	50
Block Erase (ms)	2	2
Typical Program/ Erase Cycles	100,000	10,000

- Flash Nand wears with every erase cycle
  - If enough wear, cell can no longer reliably store info
- SCL flash wears in 100.000 writing cycles
  - Much more expensive (3 – 7 times price of MLC)
  - Used in enterprise-grade SSDs
- MLC flash wears in just 3.000 – 10.000 cycles

SSD: write endurance

Write Endurance Specifications

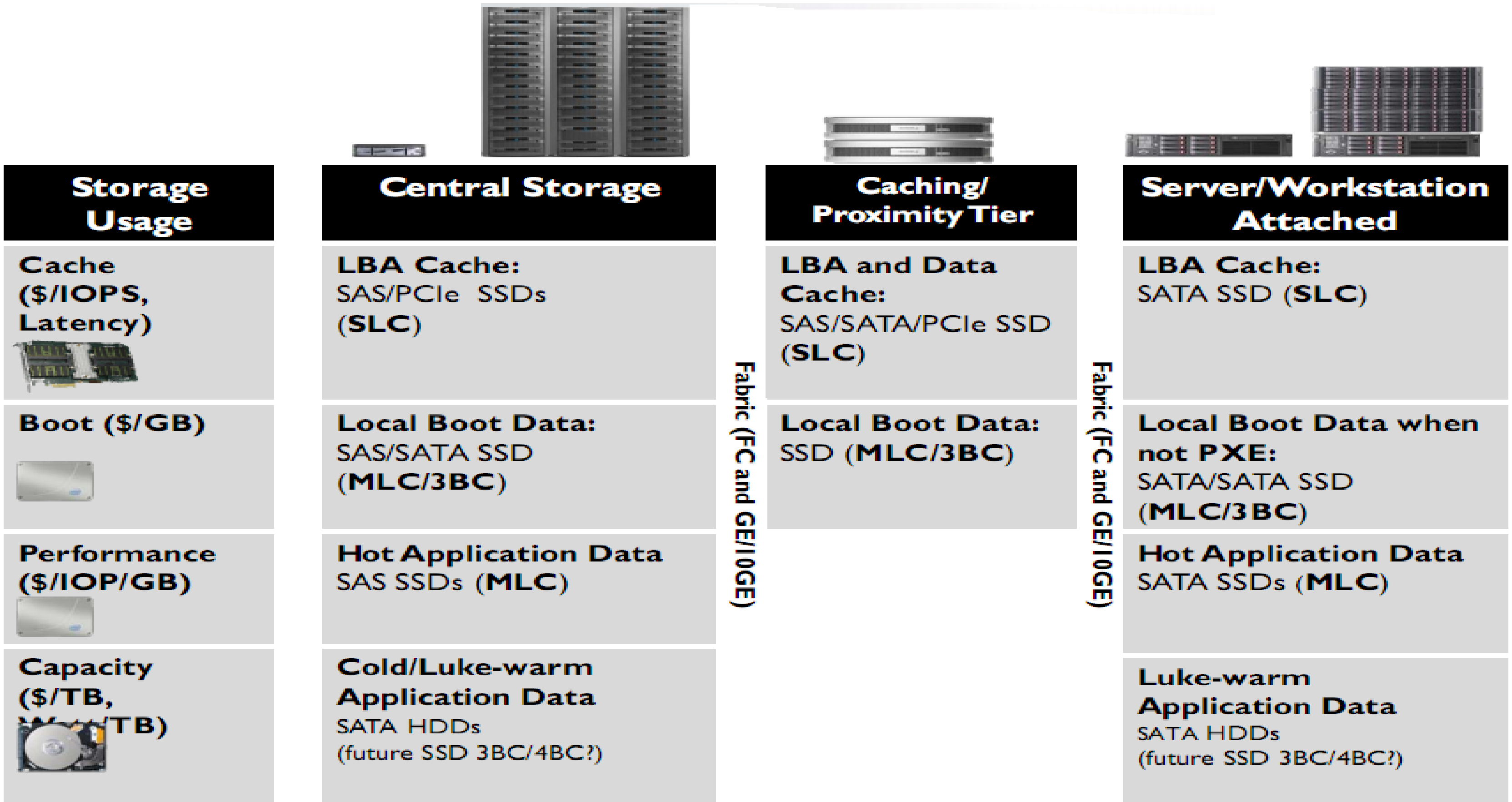
Intel SSD 910 Series	4 KB Writes	8 KB Writes
400 GB	Up to 5 PB (2.5 PB per NAND module)	Up to 7 PB (3.5 PB per NAND module)
800 GB	Up to 10 PB (2.5 PB per NAND module)	Up to 14 PB (3.5 PB per NAND module)

- Worst case for wear is random writing of small data blocks
  - Write amplification exacerbates wear
- Serious makers specify rated life of product in amount of TB/PB written before failure arrives

SSD: Summary

- SSDs can provide huge gain on IOPS and transfer speed
  - Dramatical performance improvement in apps
- Example: SETAO (Société d'Exploitation pour les Transports de l'Agglomération Orléanaise)
  - Traffic simulation software response time changed from 2 hours (SATA disks) to nearly instantaneous on solid-state drives
    - Running more simulations per day allowed saving of 1 million Euros
  - Provisioning/booting 200 virtual desktops changed from 20 minutes (SATA drives) to 5 seconds with SSD
- However, if used with the wrong application, SSDs can wear real fast
  - Need to have a very clear understanding of our app read/write requeriments

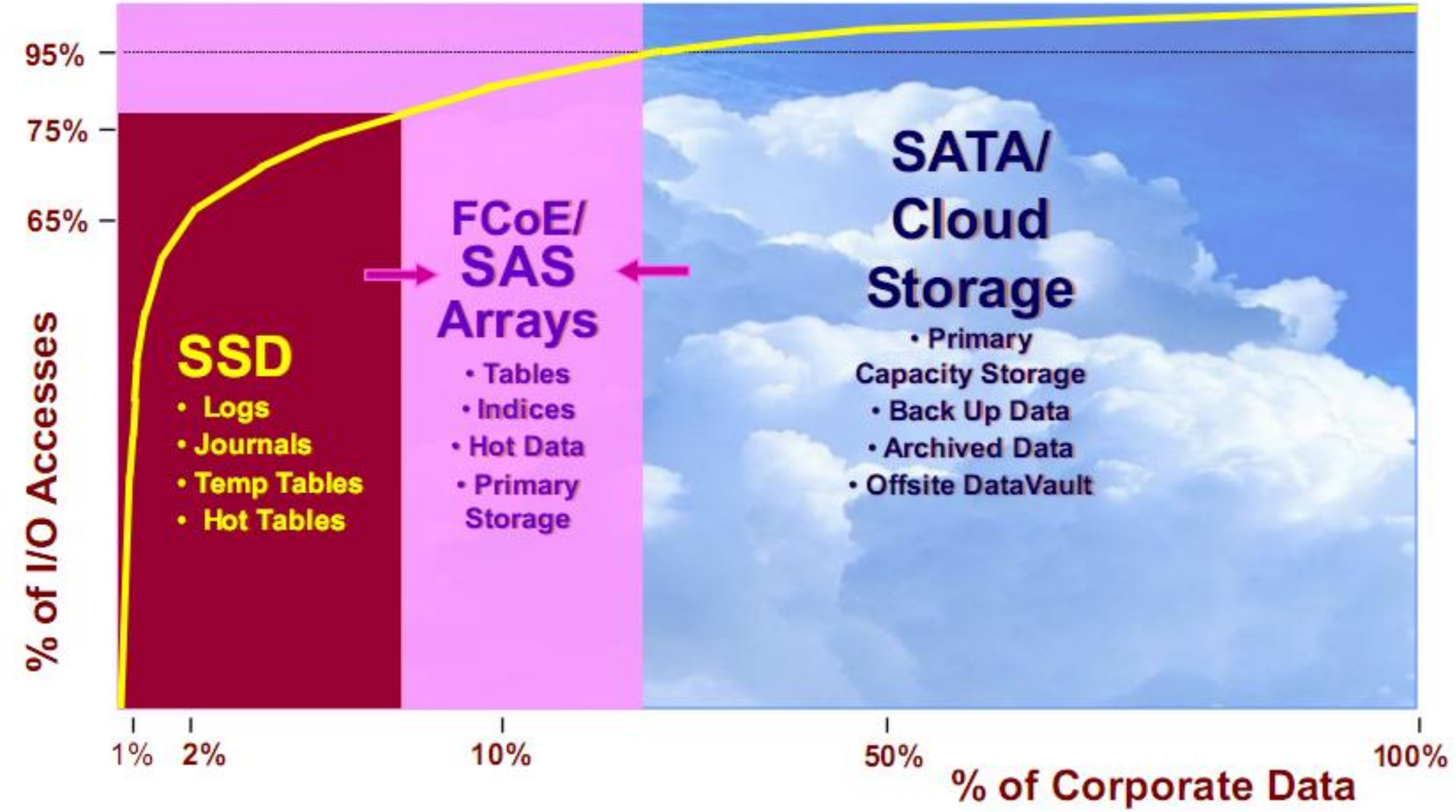
SSDs: Role



Storage Usage	Central Storage	Caching/ Proximity Tier	Server/Workstation Attached
Cache ($/IOPS, Latency)	**LBA Cache:** SAS/PCIe SSDs (**SLC**)	**LBA and Data Cache:** SAS/SATA/PCIe SSD (**SLC**)	**LBA Cache:** SATA SSD (**SLC**)
Boot ($/GB)	**Local Boot Data:** SAS/SATA SSD (**MLC/3BC**)	**Local Boot Data:** SSD (**MLC/3BC**)	**Local Boot Data when not PXE:** SATA/SATA SSD (**MLC/3BC**)
Performance ($/IOP/GB)	**Hot Application Data** SAS SSDs (**MLC**)		**Hot Application Data** SATA SSDs (**MLC**)
Capacity ($/TB, Watt/TB)	**Cold/Luke-warm Application Data** SATA HDDs (future SSD 3BC/4BC?)		**Luke-warm Application Data** SATA HDDs (future SSD 3BC/4BC?)

- Typically, SSDs used as caches or Tier 0 levels, substituting fast HDDs (15k rpm) in disk arrays

SSDs: Role



- Tiering storage in racks allows good compromise of performance / capacity / reliability /cost

What's next?

- By now we have already seen how the SCSI interface allows use of a logical model of the storage, in which physical details are hidden behind a LUN and a linear array of LBAs, to which physical storage is transparently mapped

- Now, we will see how this concept has been used in the implementation of highly flexible and powerful storage architectures for virtual machines