Big Data Systems (CS4545/CS6545)
Winter 2021

Storage and file organization for data systems

Suprio Ray

University of New Brunswick

Acknowledgement

Thanks to N. Koudas, S. Babu, C B. Ramamurthy, A. D. Hall for material in these slides. Also thanks to numerous research papers and online articles.

Overview

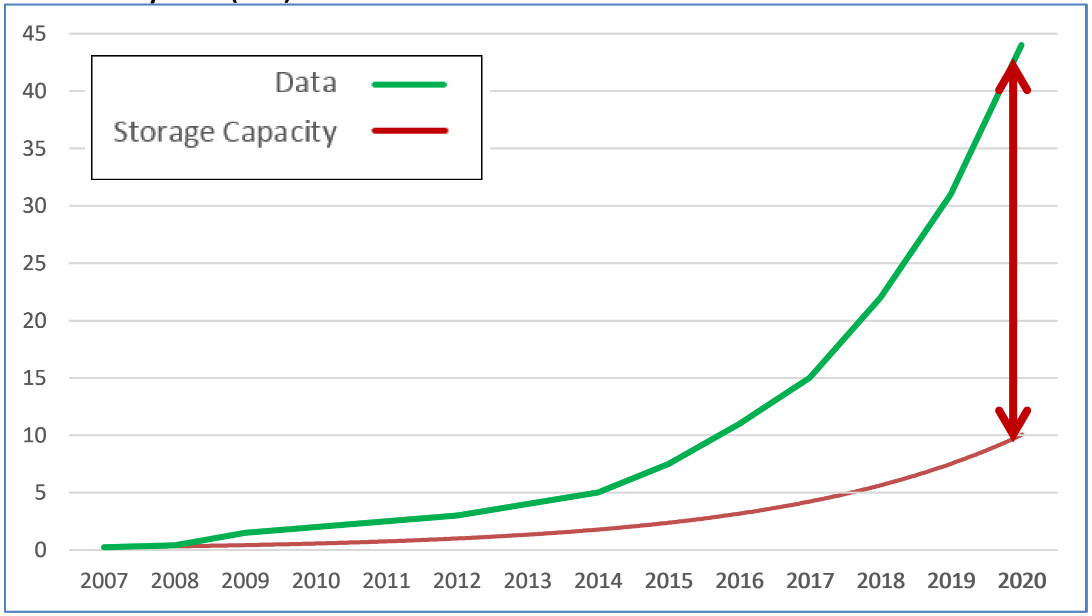
- Why storage is important for data systems?
- History of storage systems evolution
- Disks internals and data access from disks
- Software-based techniques
- Distributed storage management

RECALL: growing data capacity gap

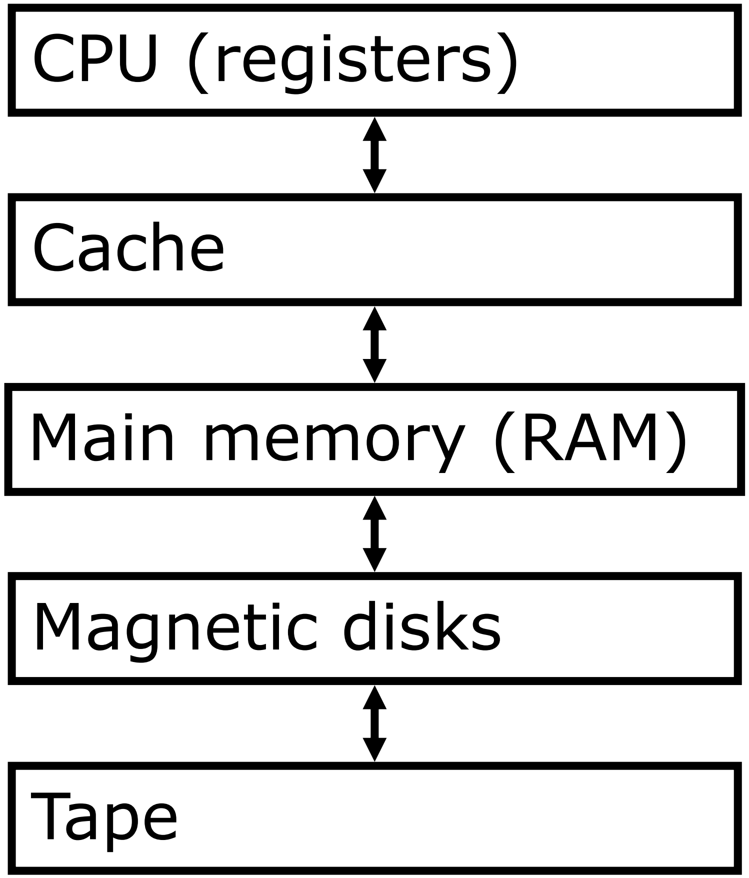
- Data deluge
  - 44 ZB by 2020

- Not enough capacity to store all generated data!
  - Let alone process all the data…

In zettabytes (ZB)

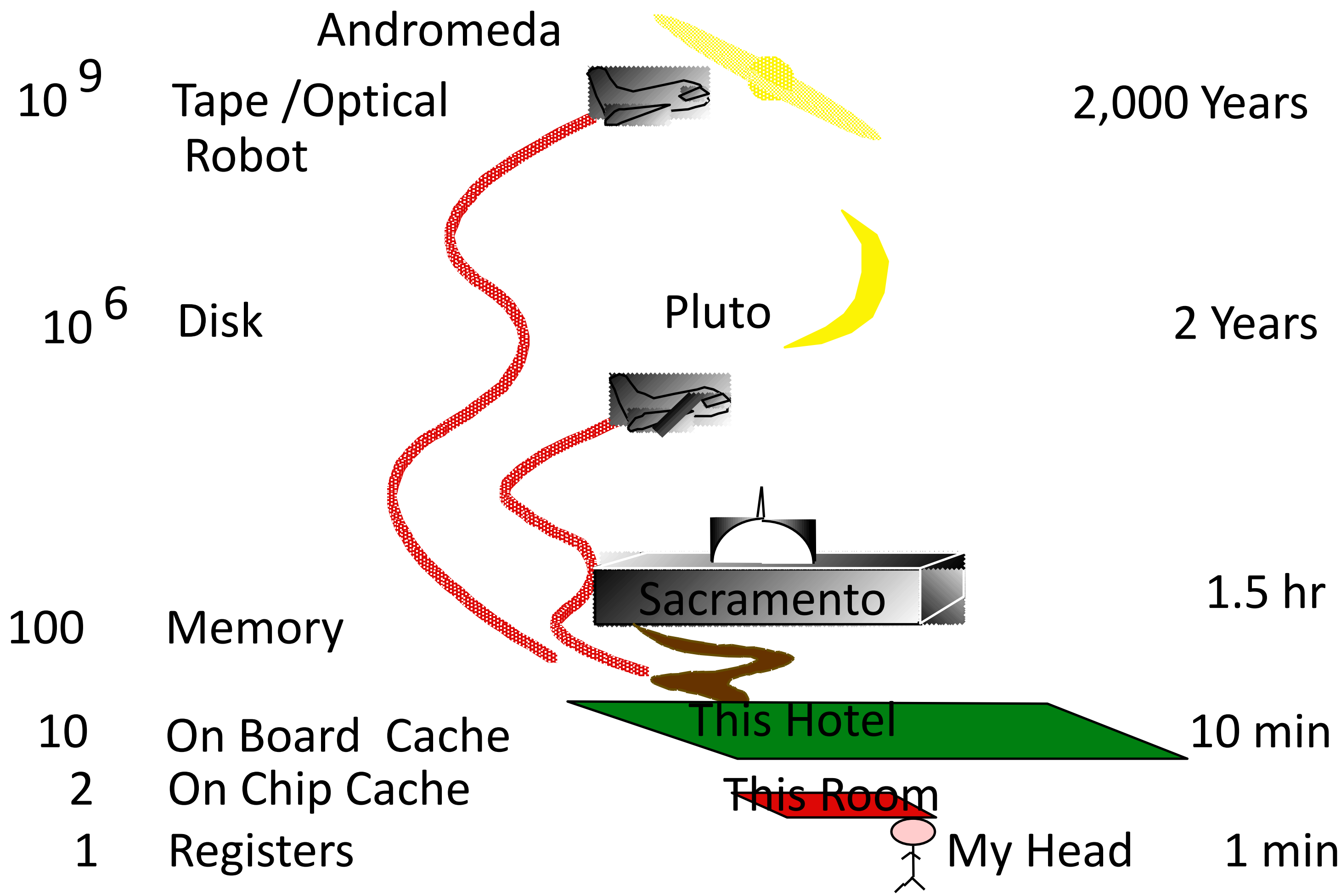


Storage alternatives – the memory hierarchy

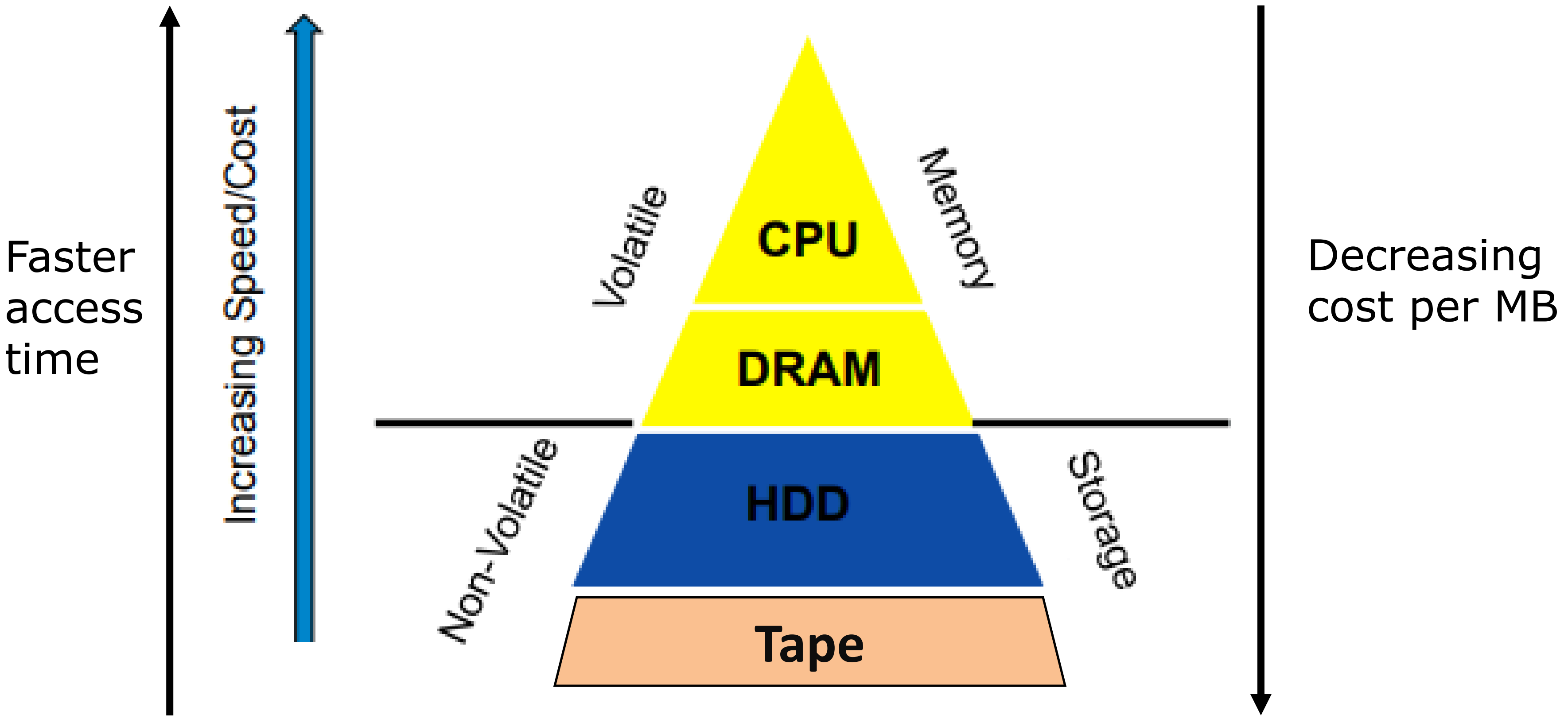


Storage alternatives – the memory hierarchy

- Main-memory is faster than disk (Jim Gray's analogy)



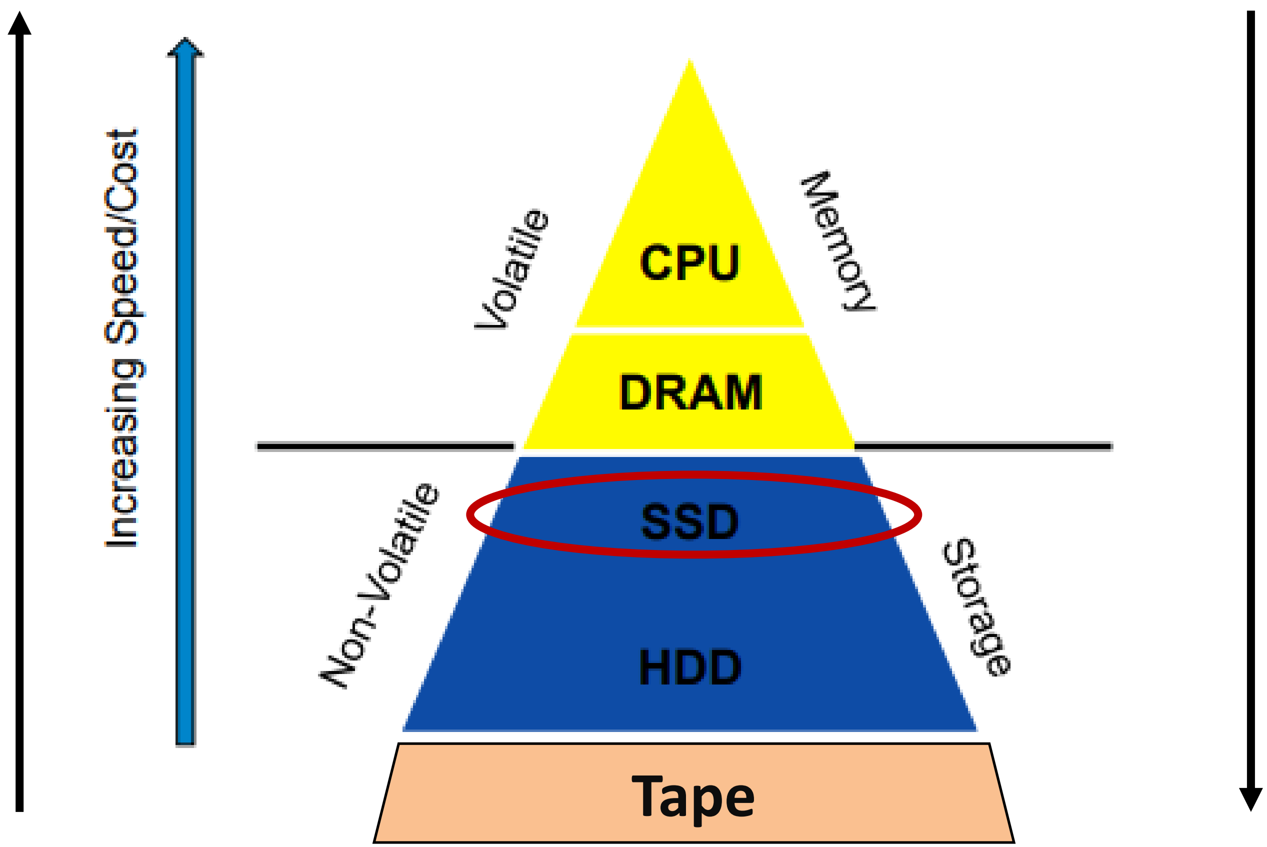
Storage alternatives – the memory hierarchy



What's missing?

Storage alternatives – the memory hierarchy

Faster access time



Decreasing cost per MB

Why Not Store Everything in Main Memory

- Typical storage hierarchy:
  - Main memory (RAM) for currently used data.
  - Disk for the persistent data store (secondary storage).

- Cost: 1TB of disk costs $70, 1TB of RAM costs over $5K

- Main memory is volatile. We want data to be saved between runs. (Obviously!)

Why disk is important for data systems?

- Storage capacity:

  disk size determines how much information we can store in a data system

- Functionality:

  disks are important for aspects such as recovery, reliability, security and archiving

Why disk is important for data systems?

- Performance:

  Since disks are the slowest component, their performance have a large impact on overall system performance

- System architecture:

  By understanding the properties of all system components, we can optimize the architecture for price / performance

Disks vs. tapes

- Tapes are cheaper and can store more data, but take much more time to read

- Disks are currently replacing or have replaced tapes as backup devices

Disks vs. RAM

- As the behavior of disks depends on the time to move the disk's arm,
  disks behavior becomes similar to sequential access tapes

- Therefore, it is expected that in the future RAM will replace disks

Disks vs. SSDs

- **Price:** SSDs are more expensive than Disks in terms of dollar per GB.

- **Speed:** SSDs offer much faster read (random access) over Disks

- **Maximum Capacity:** Disks typically have significantly more capacity (max at 10TB), than SSD (max at 1TB)

- **Form Factors:** Because HDDs rely on spinning platters, there is a limit to how small they can be manufactured.

Disks vs. SSDs (contd.)

- **Durability:** An SSD has no moving parts, so it is more likely to keep your data safe. However SSDs have limited *write cycle* (a.k.a *program/erase* or *P/E* cycle)
  - Each ***block*** contains a number of ***pages***, which are the smallest unit that can be programmed (i.e. written to)
  - Data is written on pages, but minimum unit of erasing is by ***blocks***.
  - The erase process involves hitting the flash cell with a relatively large charge of electrical energy
  - Causes the semiconductor layer on the chip to degrade over time

Typical size of a ***page*** 8-16KB, and a ***block*** 4-8MB

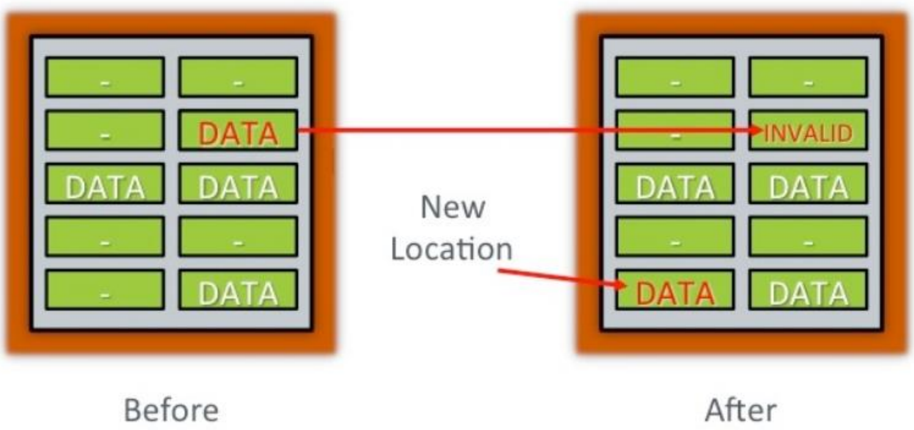
Single-level cell NAND flash supports 50,000 to 100,000 write cycles.

Operation	Area
Read	Page
Program (Write)	Page
Erase	**Block**

Src: flashdba.com

Disks vs. SSDs (contd.)

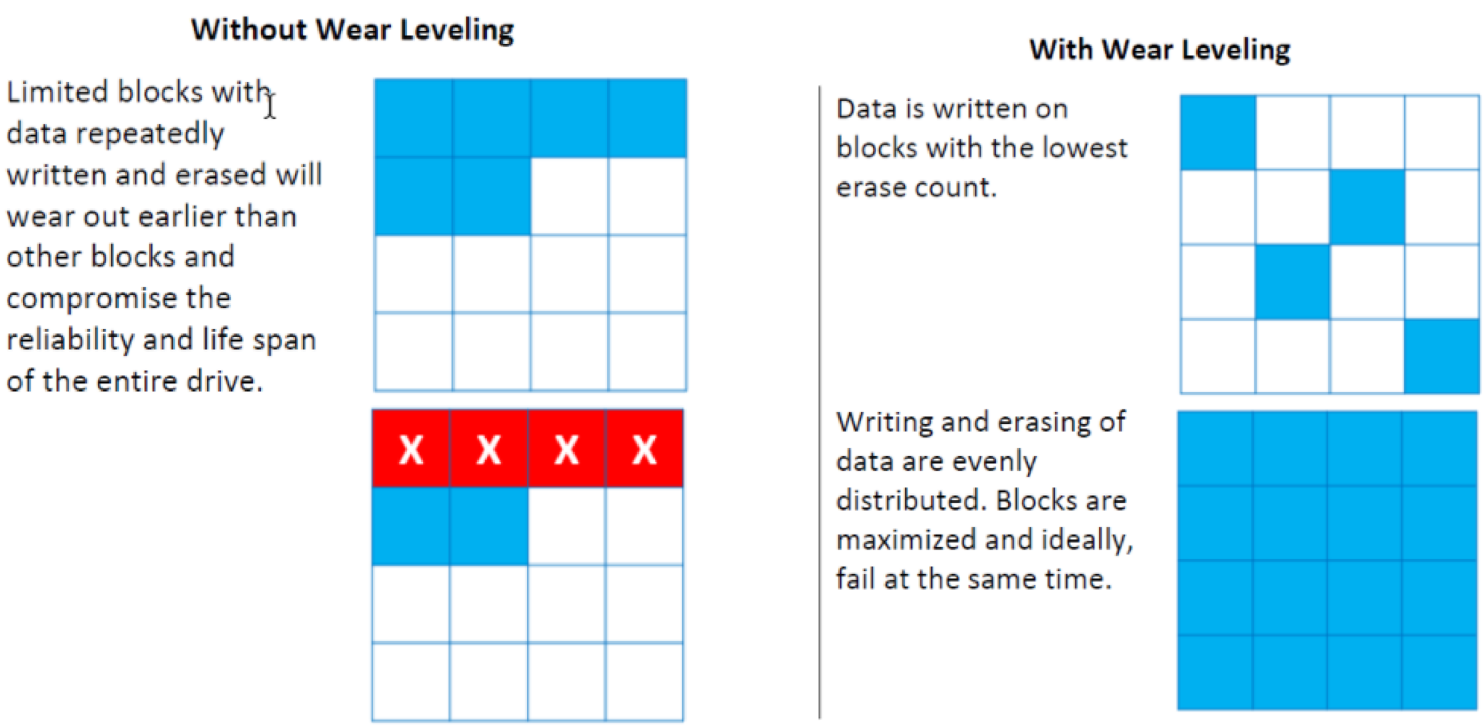
- **Durability:** SSDs have limited *write cycle* (a.k.a *program/erase* or *P/E* cycle)
  - If we were to erase a block every time we wanted to change the contents of a page, the SSD device would wear out very quickly
  - Instead, mark the old page (containing the unchanged data) as INVALID and then write the new, changed data to an empty page.
  - A mechanism is used (*flash translation layer*) to point any subsequent operations to the new page and a way of tracking invalid pages so that, at some point, they can be "recycled".



Src: flashdba.com

Disks vs. SSDs (contd.)

- **Durability:** SSDs have limited *write cycle* (a.k.a *program/erase* or *P/E* cycle)
  - ***Wear leveling*** allows the SSD device to evenly distribute the P/E cycles among all blocks.



Src: www.atpinc.com

UNB FREDERICTON

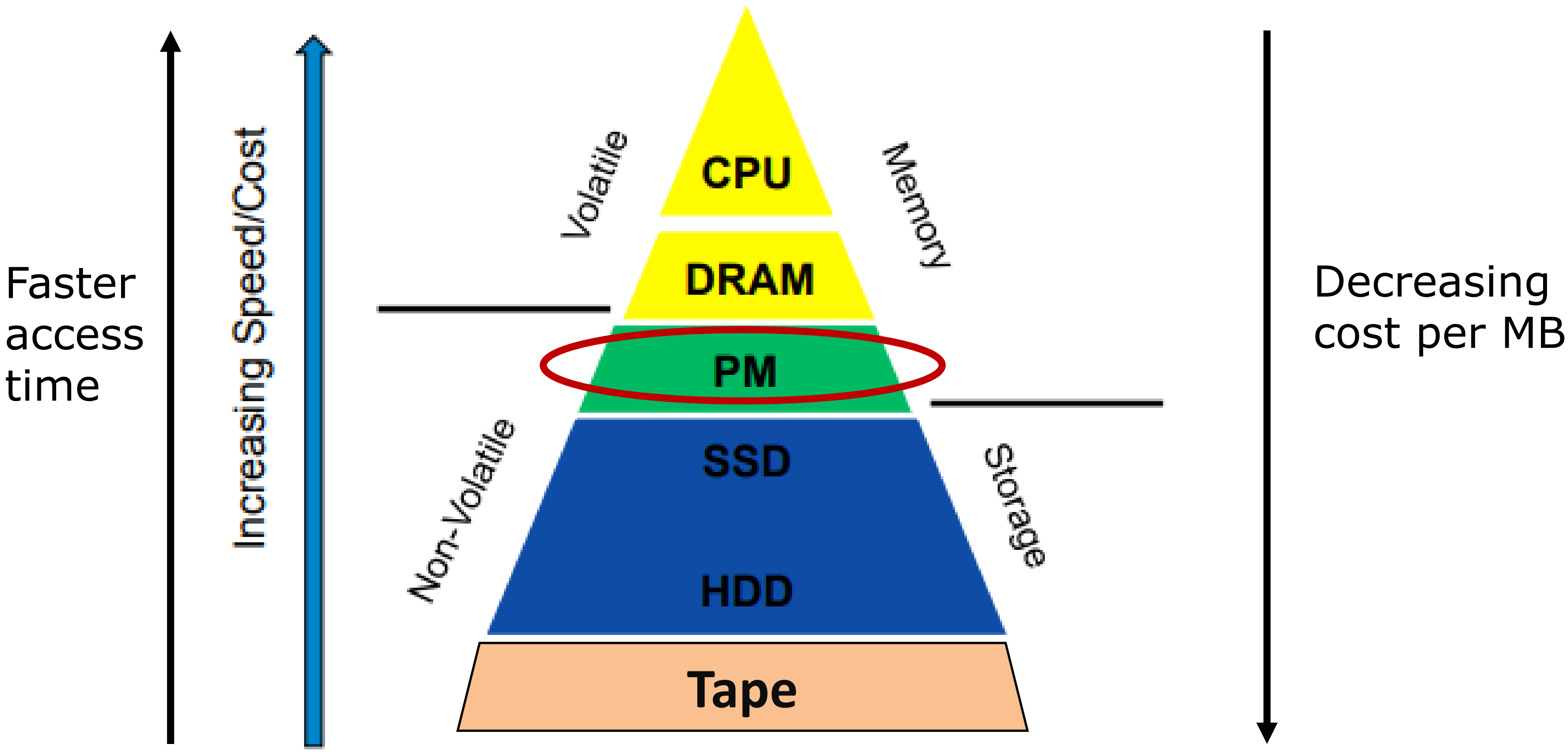
SSDs vs. hard-drives summary

Characteristic	SSD	Hard Drive
Start time	none	seconds
Random access	0.1ms to 1ms	5-10ms
Read latency	low	high
Read perf.	No change	varies
Defrag.	No benefit	benefit
Acoustics	No sound	sound
Mechanical re.	No mechanical	mechanical
Weight/size	Very light	heavy
Parallel ops	yes	yes
Write	limited	No limitation
Cost	$1-1.5/GB	$0.1/GB
Storage (max)	1~2TB	~10TB

The memory hierarchy - near future

PM (Persistent Memory) or NVDIMM: 3D XPoint (Intel), Hypervault (Netlist)

- Faster than SSD and byte-addressable
- A non-volatile, byte addressable, low latency memory



Src: adapted from M. Webb

Overview

- Why storage is important for data systems?
- History of storage systems evolution
- Disks internals and data access from disks
- Software-based techniques
- Distributed storage management

Historical evolution of disks

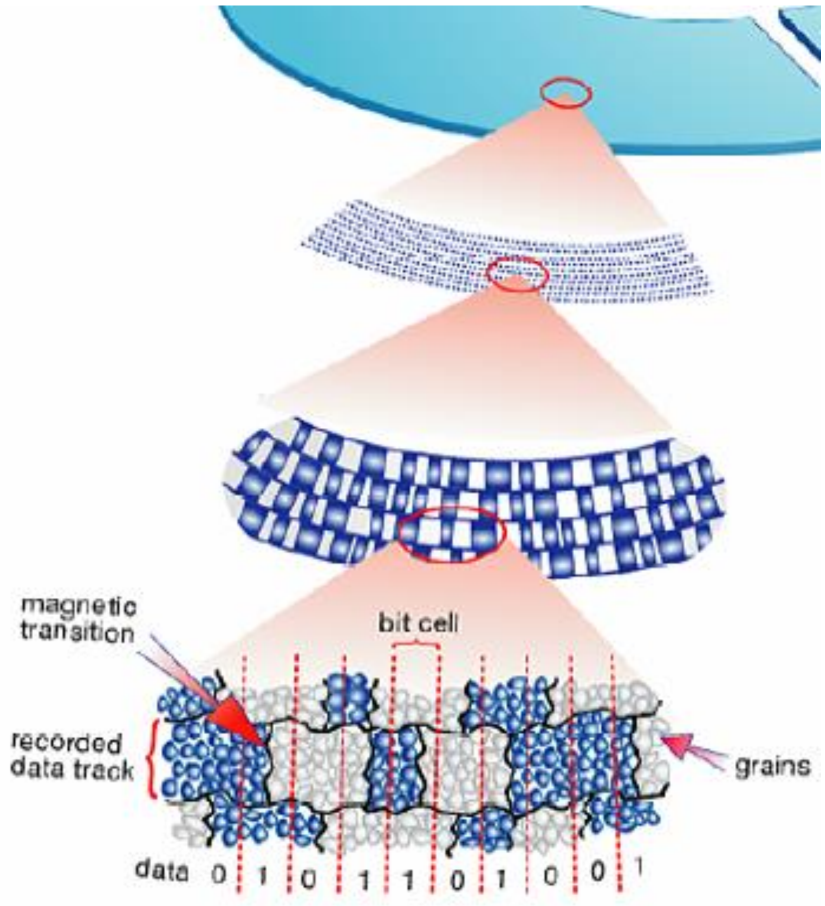
IBM RAMAC (1956)
Capacity: 3.75 MB
RPM: 1,200
(Fifty 24-inch-wide platters!)

Hitachi Deskstar 7K3000 (2011)
Capacity: 3TB
RPM: 7,200

IBM TotalStorage DS8000 series (2014)
Capacity: 192 TB
RPM: 15,000

A Magnetic 'Bit'

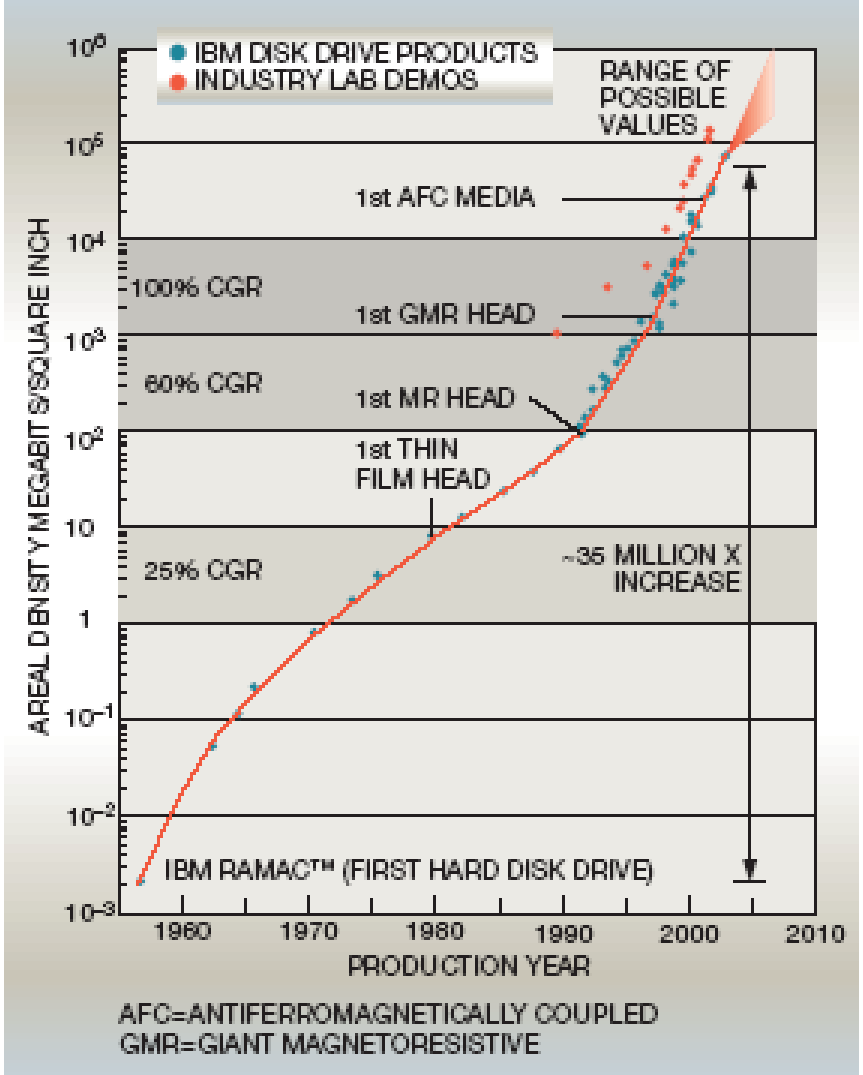
- Disk platter - Aluminum with a deposit of magnetic material
- Bit-cell composed of magnetic grains
  - 50-100 grains/bit
- '0'
  - Region of grains of uniform magnetic polarity
- '1'
  - Boundary between regions of opposite magnetization



Source: http://www.hitachigst.com/hdd/research/storage/pm/index.html

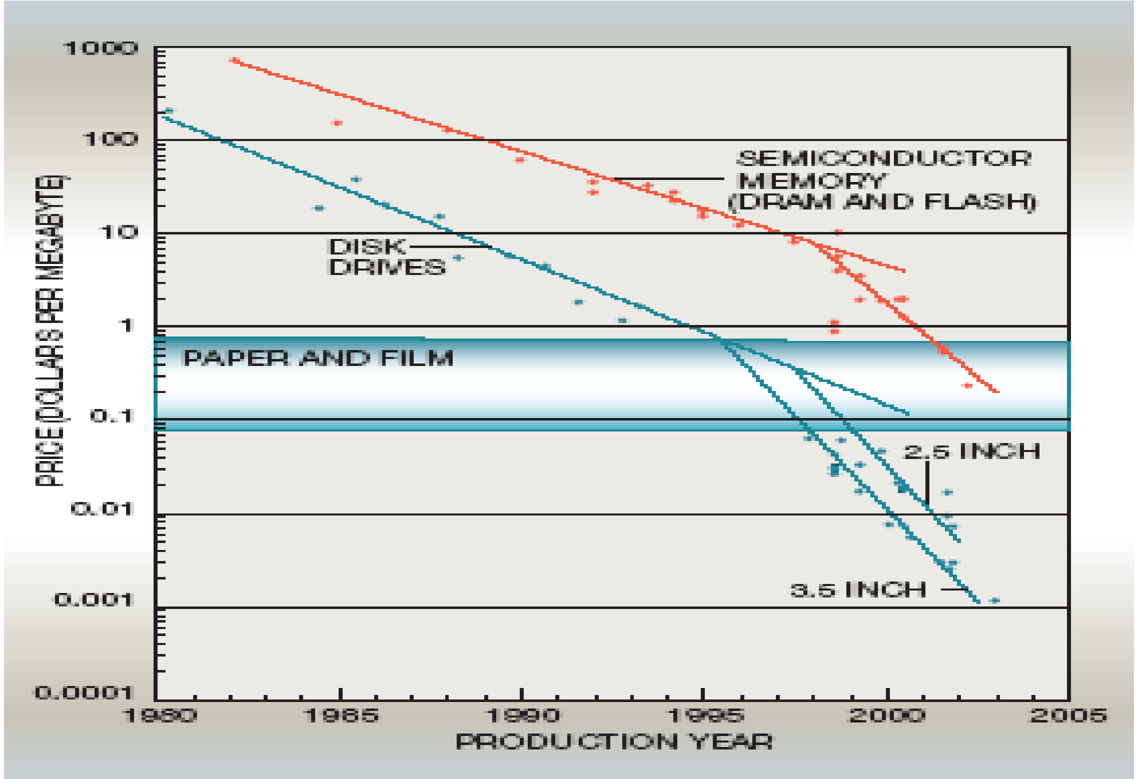
Areal density trend

- Areal density: megabit/square inch
- Current improvement rate is 100% a year!



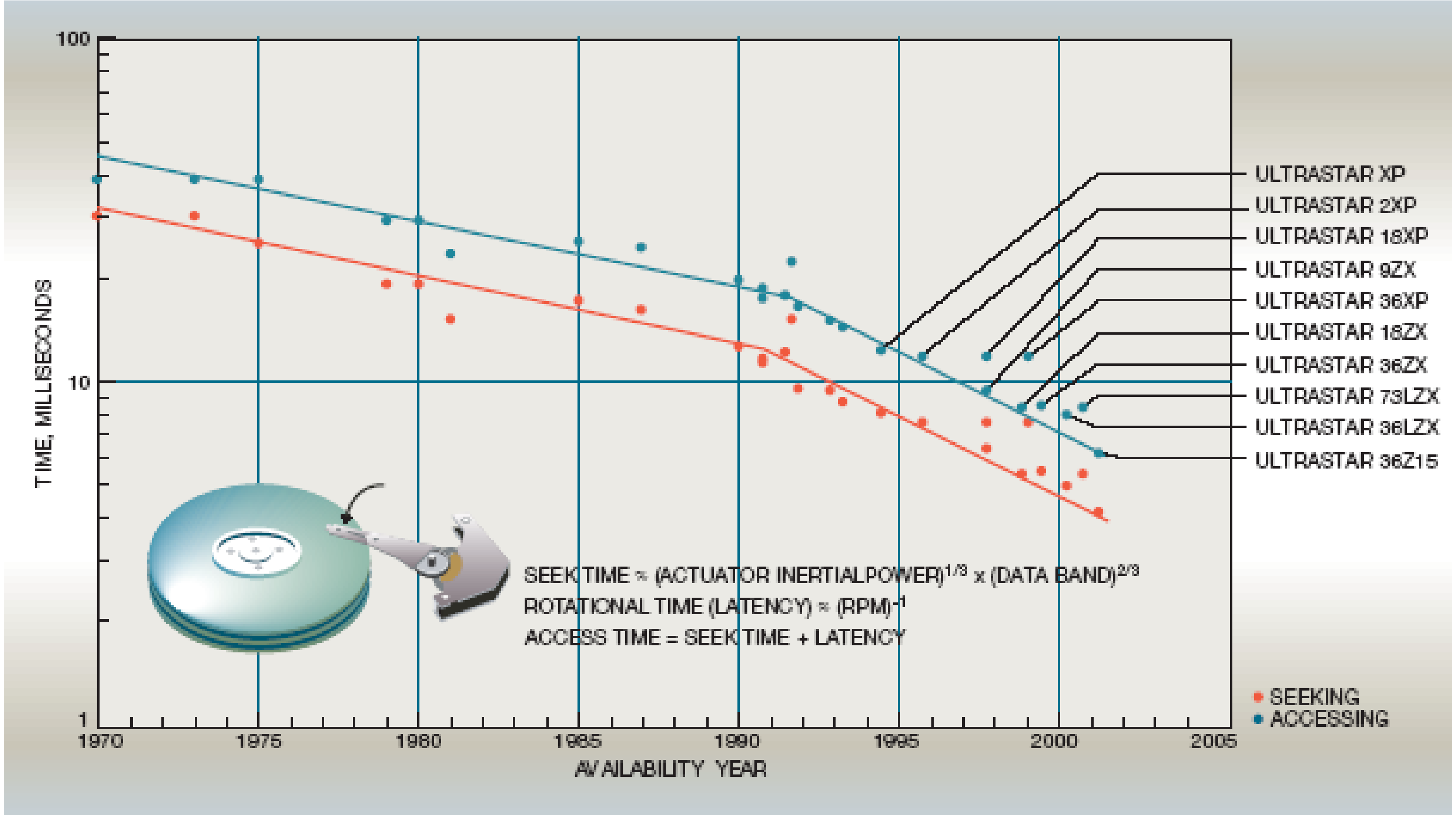
Price trends

- Price per MB decreased 37% - 50% per year



Cost of disk storage compared to paper, film, DRAM and Flash memory in $/MByte

Seek and access time trends



- Access time (milliseconds) improved only about 10% a year

Storage improvement trends summary

- Over the last decade
  - Capacity increased 60% - 100% per year
  - Price per MB decreased 37% - 50% per year
  - Access time improved about 10% a year

Overview

- Why storage is important for data systems?
- History of storage systems evolution
- Disks internals and data access from disks
- Software-based techniques
- Distributed storage management

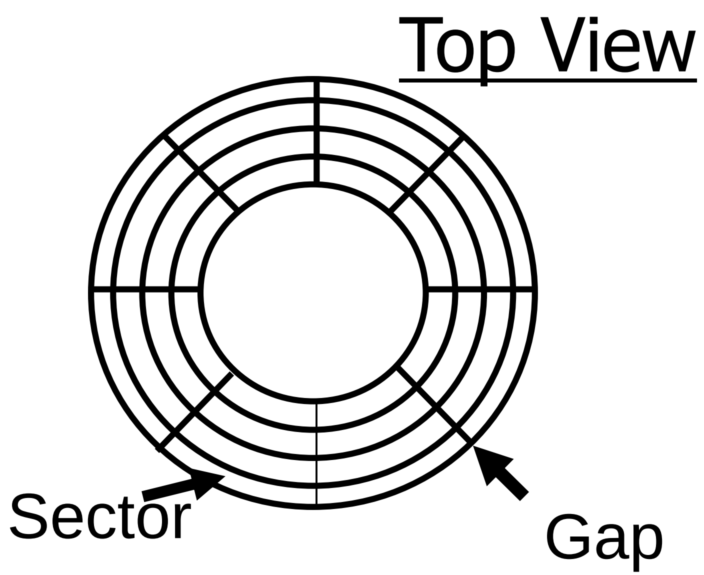
Hard Disk Drive (HDD) Components

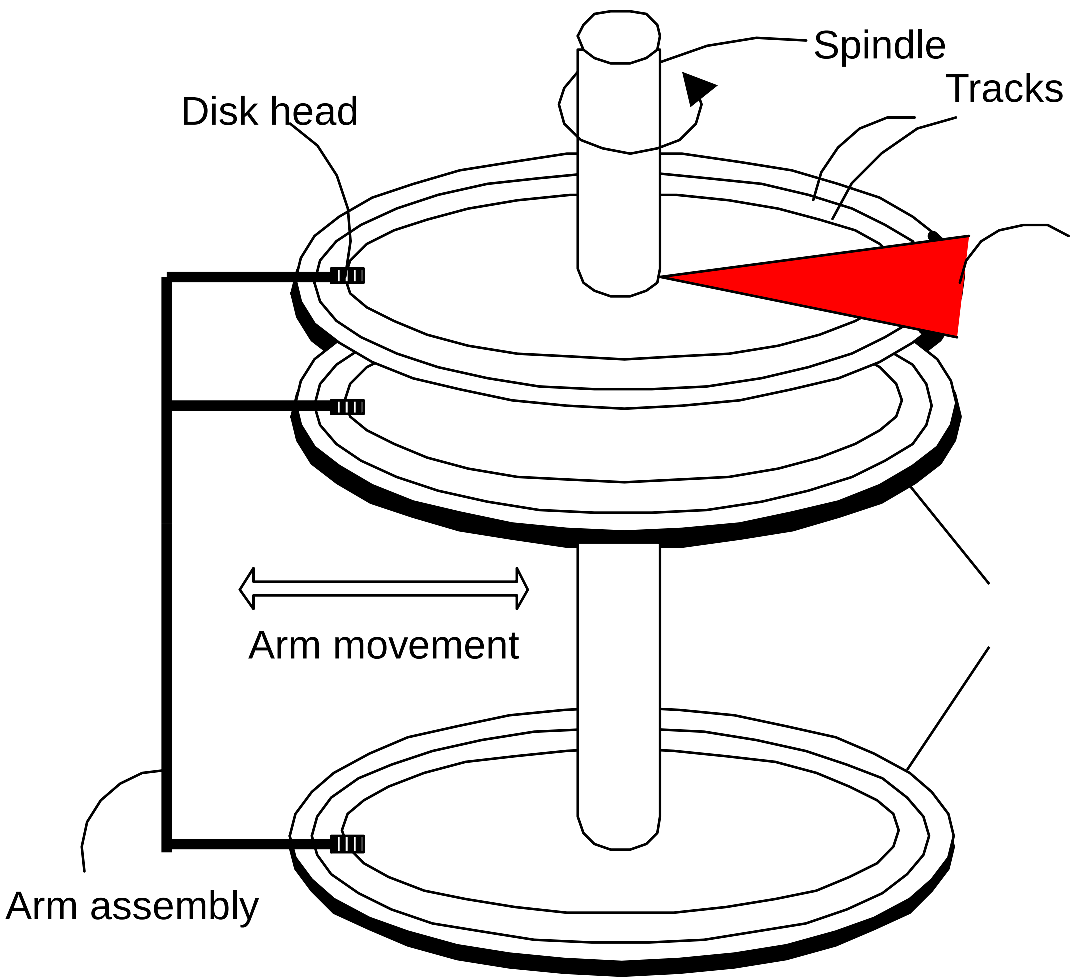
- Electromechanical
  - Rotating disks
  - Arm assembly

- Electronics
  - Disk controller
  - Cache
  - Interface controller

Accessing data from disk: Block Address

- Physical Device
- Cylinder #
- Surface #
- Start sector #

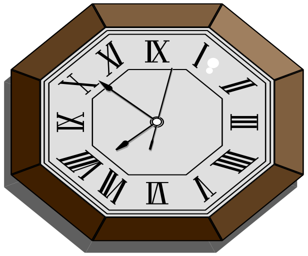




Disk Access Time (Latency)

I want
block X
from disk

→



→

block X
in memory

?

Disk Access Time (Latency)

- Access Time = Seek Time +
  Rotational Delay +
  Transfer Time + Other

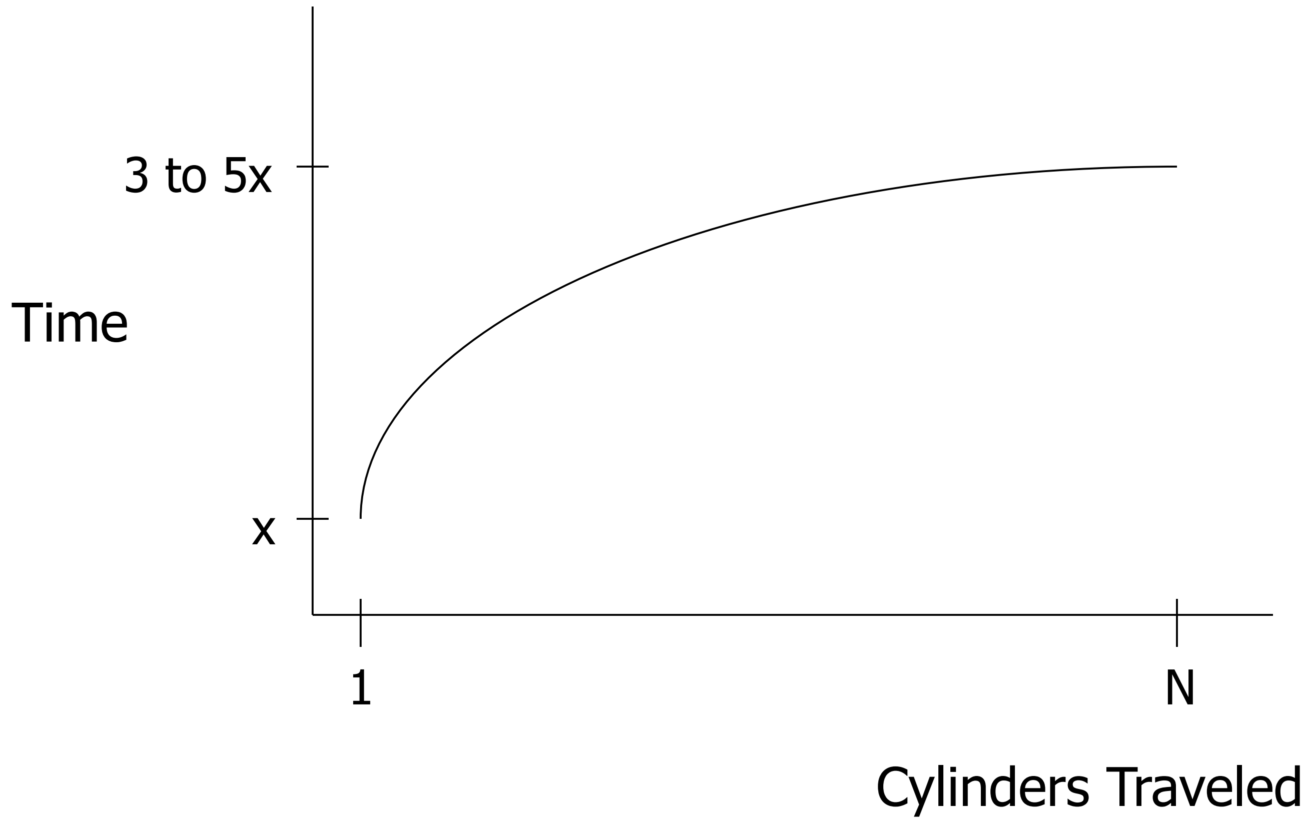
- **Seek Time**: time required to position the disk arm to the correct track (cylinder)

- **Rotational Delay**: time required for the read-write head to be at the beginning of the first sector of the requested block

- **Transfer Time**: time required to transfer the data between the disk and the main memory

Seek Time

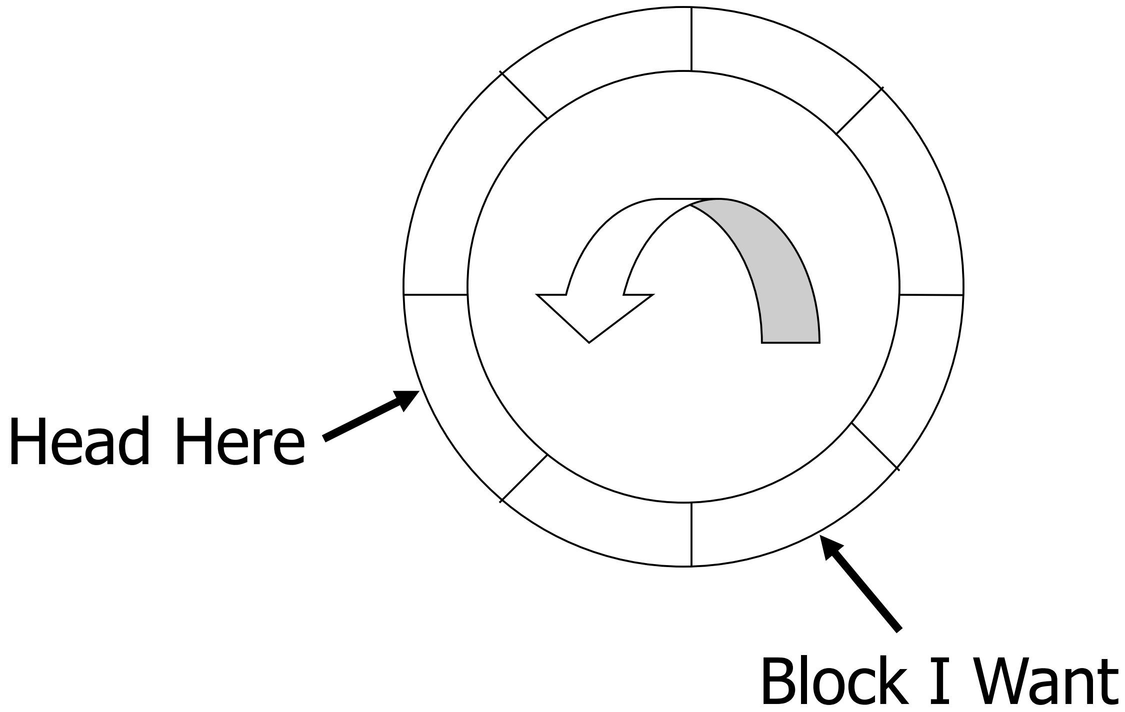
- Seek – time for the head to move to the right track



Average value: 10 ms → 40 ms

Rotational Delay

- Rotational delay – time until the sector (block) passes under the head



Average Rotational Delay

Average R = 1/2 revolution

Example: R = 8.33 ms (3600 RPM)
= 4.16 ms (7200 RPM)

Transfer Time

- Transfer rate, t: ~100 MB/second

- Transfer time: $\dfrac{\text{block size}}{t}$

- Typical block (sector) size = 4 KB
  - Transfer time for 1 block = 0.04 ms

Other Delays

- CPU time to issue I/O
- Contention for controller
- Contention for bus, memory

- “Typical” Value: 0

Disk Access Time (Latency)

- Access Time = Seek Time +
  Rotational Delay +
  Transfer Time + Other

Min Access Time = (10 + 4.16 + 0.04 + 0) ms
= 14.2 ms

Max Access Time = (40 + 4.16 + 0.04 + 0) ms
= 44.2 ms

Disk Access: Random vs. Sequential

- So far: Random Block Access
- What about: Reading “Next” block?

If we do things right …

$$\text{Time to get next block} = \frac{\text{Block Size}}{t} + \text{Negligible}$$

- skip gap
- switch track
- once in a while,
  next cylinder

Disk Access: Random vs. Sequential - summary

Rule of Thumb	Random I/O: Expensive Sequential I/O: Much less

- Ex: 1 KB Block
  - » Random I/O: ~ 20 ms.
  - » Sequential I/O: ~ 1 ms.

Disk Access: Random vs. Sequential - example

- Let's say, we have a 20 TB disk.

- If we assume 200 accesses per second reading few KB each (random), it will take a year to read the entire disk

- Sequential reading is much faster.
  Reading this disk sequentially takes a day

Reducing disk accesses

- Favoring sequential transfers to random accesses
- Using few large transfers instead of many small ones
- Using versions of RAID
- Log writes to disk
- The five minute rule

RAID

- The term "RAID" was invented by David Patterson, Garth A. Gibson, and Randy Katz in 1987, and published in SIGMOD 1988

- RAID (Redundant Array of Inexpensive Disks) is a data storage technology that combines multiple physical disk drive components into one or more logical units for the purposes of data redundancy, performance, or both.

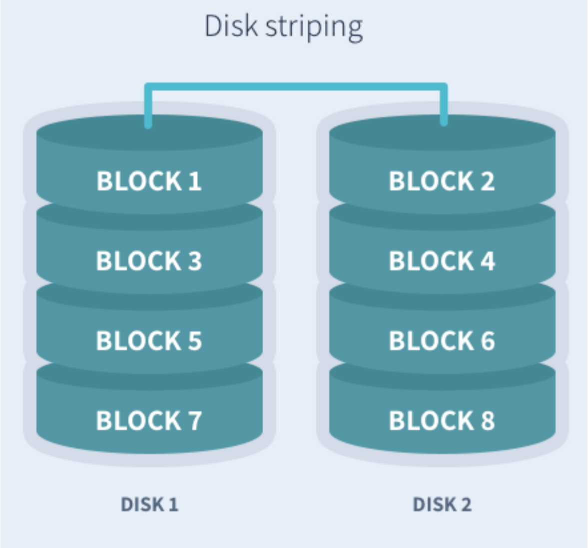
Src: wikipedia

- This was in contrast to the previous concept of highly reliable mainframe disk drives referred to as "single large expensive disk" (SLED)

- A number of standard schemes have evolved. These are called *levels*.

RAID levels: RAID0

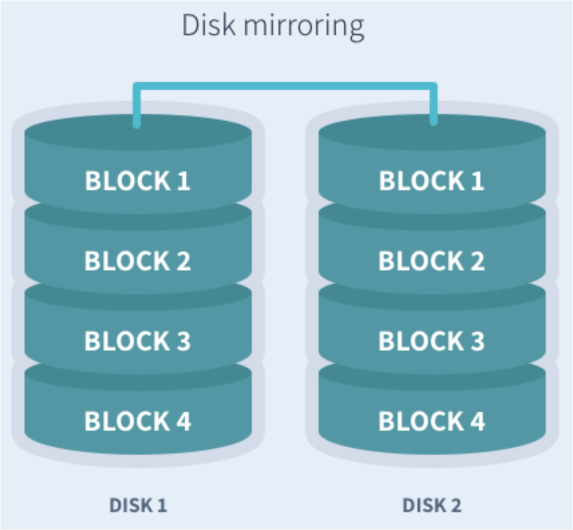
- Splits data across a number of disks allowing higher data throughput
- An individual file is read from multiple disks (giving it access to the speed and capacity of all of them.)
- However, it does not facilitate any kind of redundancy and fault tolerance as it does not duplicate data
- **Business use:** Live streaming, IPTV, VOD Edge Server



Src: Kristian Smith

RAID levels: RAID1

- Writes and reads identical data to pairs of drives
- It’s primary function is to provide redundancy.
- If any of the disks in the array fails, the system can still access data from the remaining disk(s).
- **Business use:** standard application servers where data redundancy and availability is important



Src: Kristian Smith

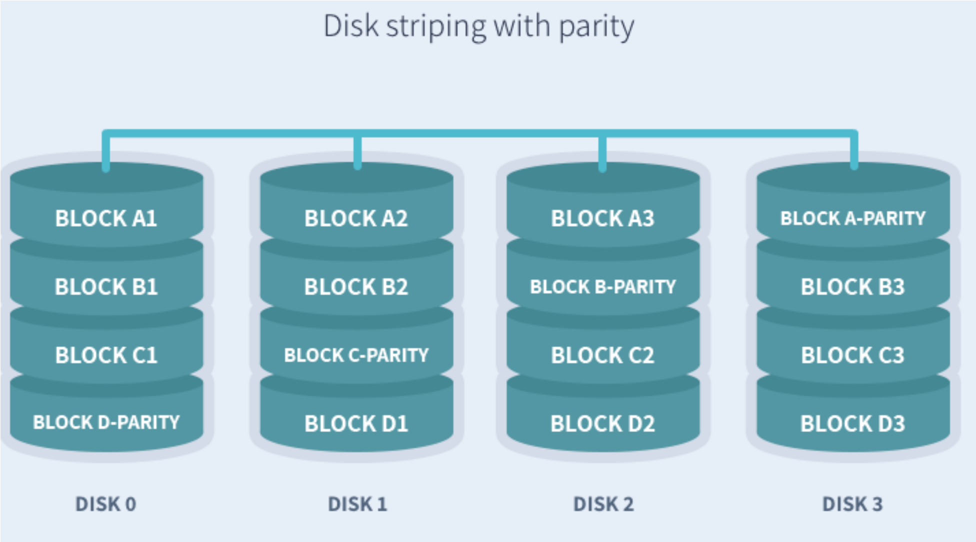
RAID levels: RAID5

- Stripes data blocks across multiple disks like RAID 0, however, it also stores parity information

- This offers both speed (data is accessed from multiple disks) and redundancy as parity data is stored across all of the disks

- However, It uses approximately one-third of the available disk capacity for storing parity

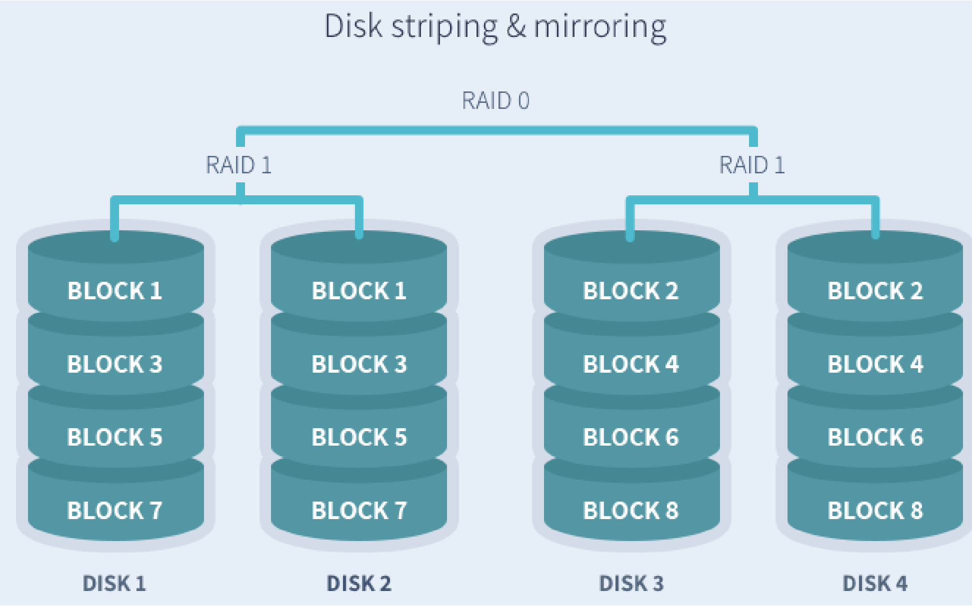
- **Business use:** file storage servers and application servers.



Src: Kristian Smith

RAID levels: RAID10

- Combines the mirroring of RAID 1 with the striping of RAID 0
- It is best suitable for environments where both high performance and security is required.
- The main drawback is lower usable capacity/high cost
- **Business use:** highly utilized database servers/ servers performing a lot of write operations



Src: Kristian Smith

Reducing disk accesses

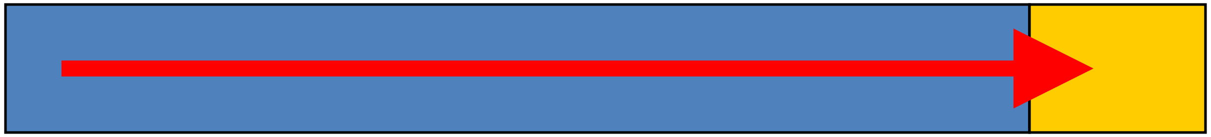
- Favoring sequential transfers to random accesses
- Using few large transfers instead of many small ones
- Using versions of RAID
- Log writes to disk
- The five minute rule

Log-structured file system

- Data velocity: OLTP workload, data from sensors & IOT
- Assumptions:
  - We can cache files in main memory
  - Performance is limited by write operations
- Idea:
  - collect changes in main memory
  - Use large IO operations

Key idea

- Write all modifications to disk sequentially in a log-like structure



  - Convert many small random writes into large sequential transfers
  - Use file cache as write buffer

Main advantages

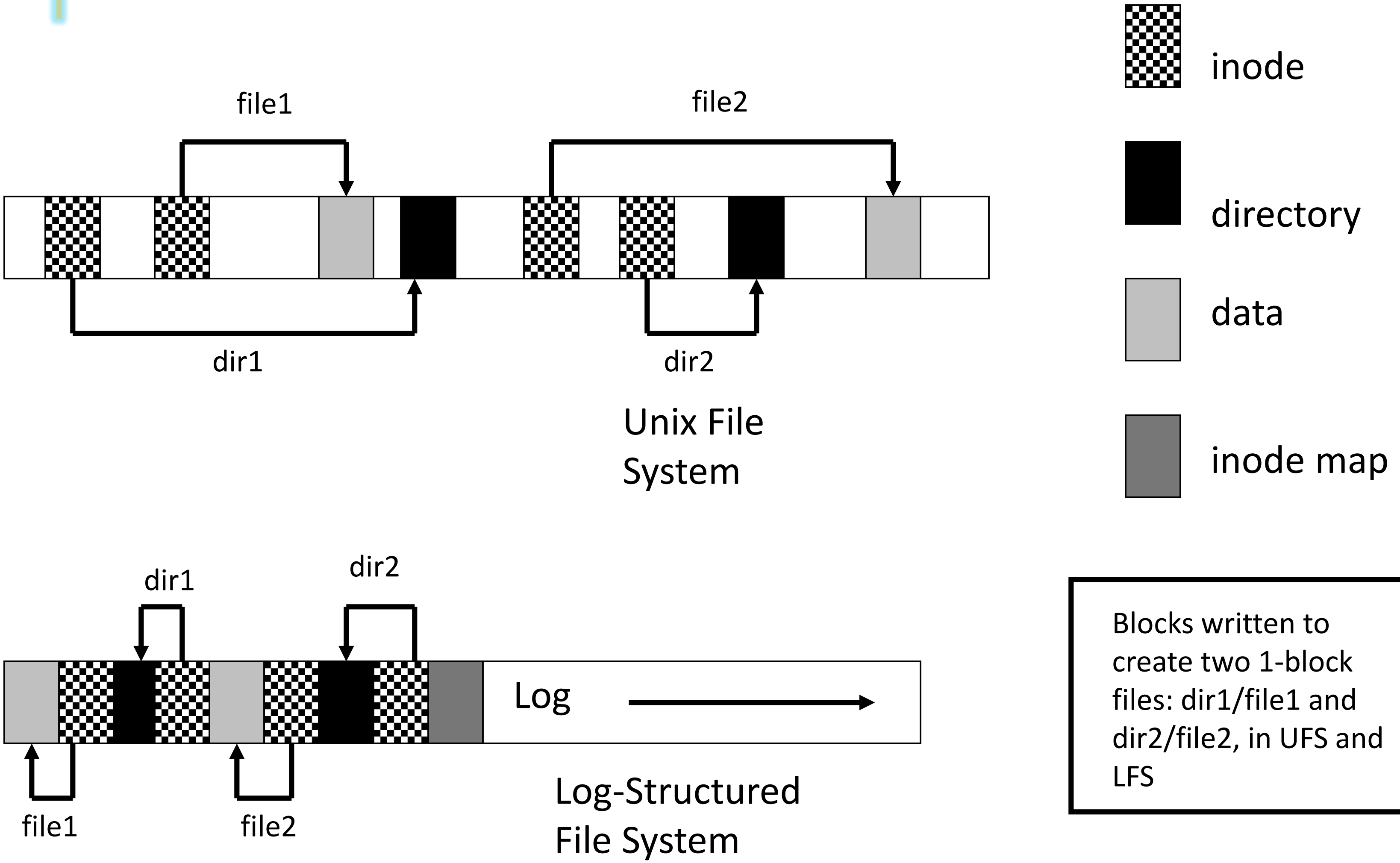
- Replaces many small random writes by fewer sequential writes

- Faster recovery after a crash

  - All blocks that were recently written are at the tail end of log

  - No need to check ***whole file system*** for inconsistencies
    - Like UNIX and Windows 95/98 do

Log-structured (LFS) vs. Unix file systems (UFS)



Blocks written to create two 1-block files: dir1/file1 and dir2/file2, in UFS and LFS

Reducing disk accesses

- Favoring sequential transfers to random accesses
- Using few large transfers instead of many small ones
- Using versions of RAID
- Log writes to disk
- The five minute rule

Economical considerations

- RAMs are an alternative to magnetic disks
- RAMs are more expensive but have faster access time
- When should information be stored on disks and when in main memory?

The five minutes rule (1987)

- In 1987, Gray and Putzolo published their now-famous five-minute rule for trading off memory and I/O capacity

- Random pages referenced every five minutes (or less) should be stored in main memory

- “..they found that the price of RAM memory to hold a record of 1 KB was about equal to the (fractional) price of a disk drive required to access such a record every 400 seconds”

Ten years later (1997)

- The five minutes rule still holds
- One minute rule:

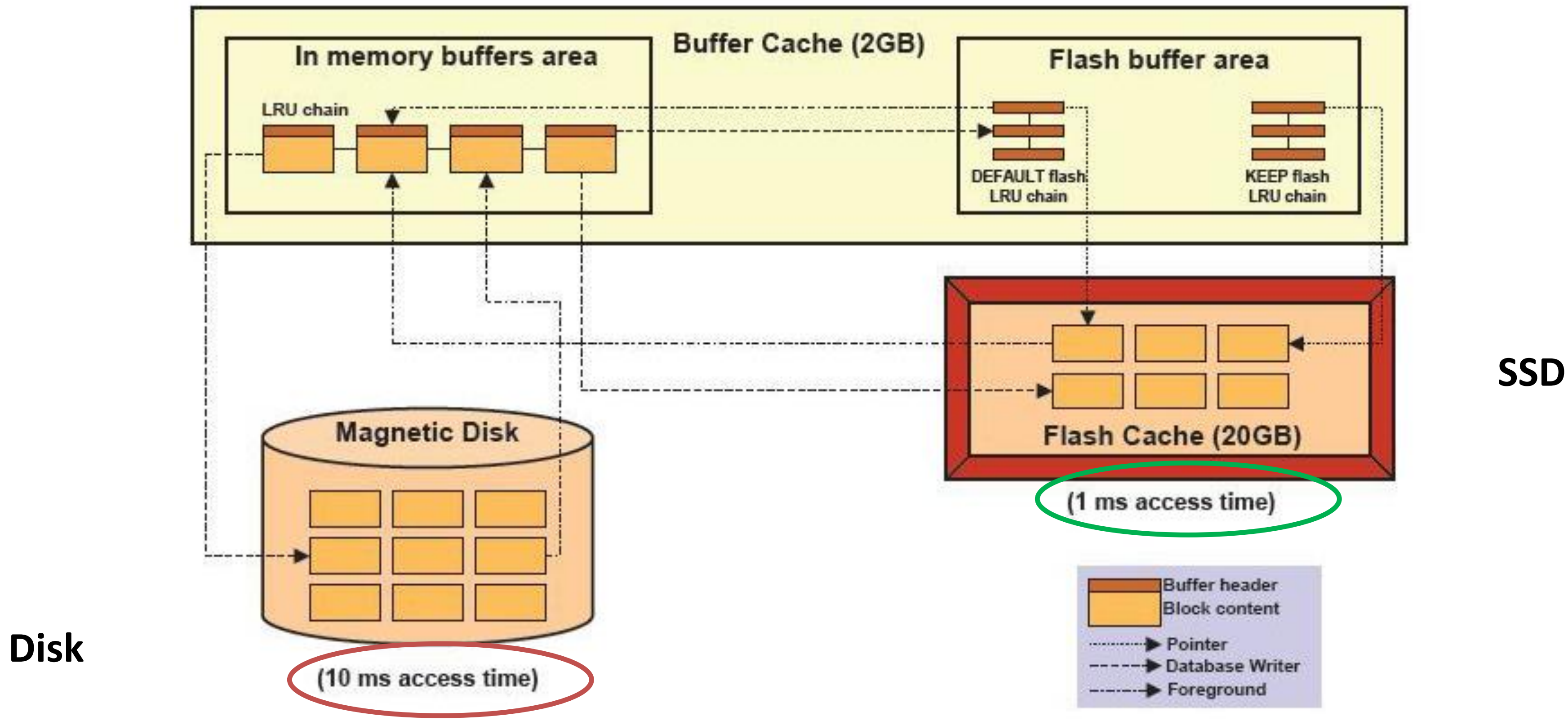
  sequential operations (such as hash-joins and sorts) should use main memory if the algorithm will re-visit the data within a minute
- Similar analysis recommends spending 1 byte of RAM to save 1 instruction per second
- Similar analysis can be carried out to compute optimal index page size

…Challenges

- Capacity doubles each year
- Access time improves by 10% annually
- Implication:
  - Disk performance remains a critical bottleneck
  - Keeping data close to memory is getting even more important!
  - SSDs will become important for read-heavy workloads

SSD as a cache

- Storing disk blocks in the SSD cache



DB Smart Flash Cache in Oracle 11g

Overview

- Why storage is important for data systems?
- History of storage systems evolution
- Disks internals and data access from disks
- Software-based techniques
- Distributed storage management

Software-based Optimizations (in Disk controller, OS, or DBMS Buffer Manager)

- Prefetching blocks
- Choosing the right block size
- 5 minutes rule and its variations
  - Random pages referenced <u>every</u> five minutes (or less) should be stored in main memory

Prefetching Blocks

- **Idea**: get a block into memory from disk before it is needed
- Exploits locality of access
  - Ex: relation scan
- Improves performance by hiding access latency
- Need to keep track of the access pattern

Block Size Selection?

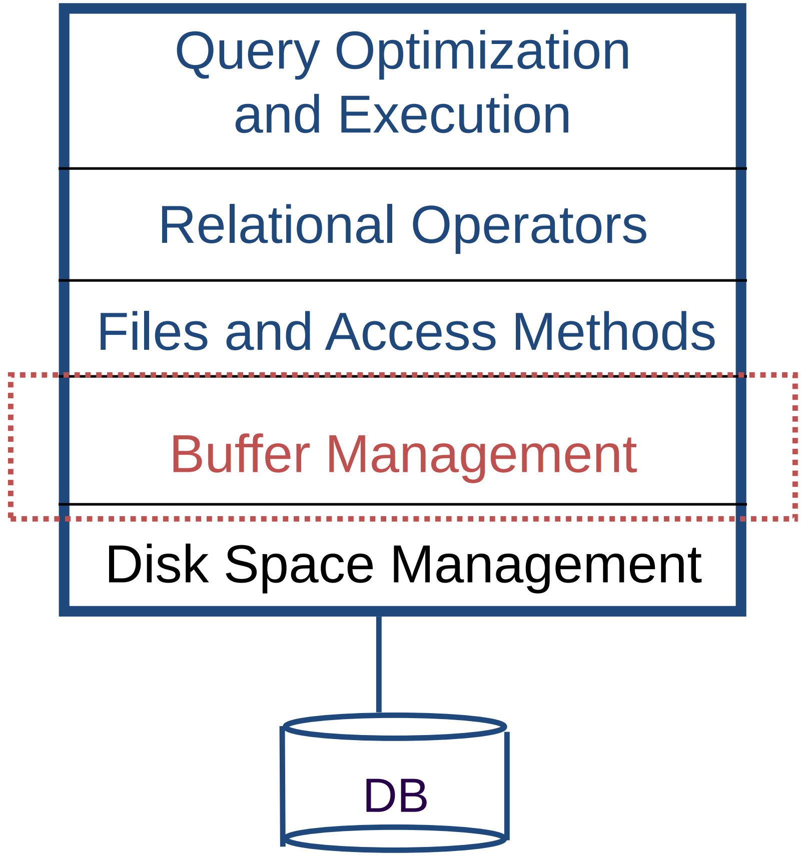
- Big Block $\rightarrow$ Amortize I/O Cost

Unfortunately...

- Big Block $\Rightarrow$ Read in more useless stuff!

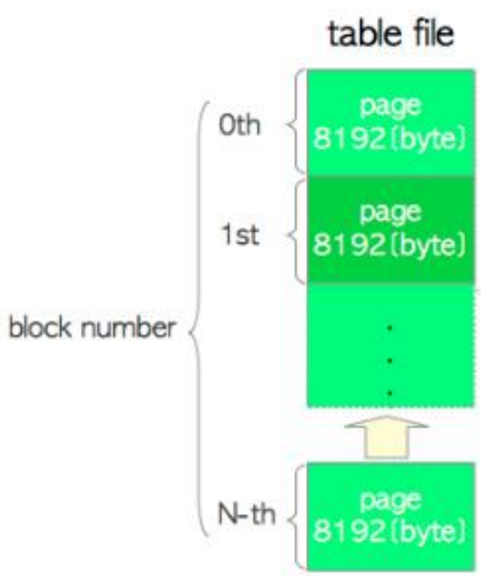
- Small Block $\rightarrow$ Too many seeks

Context - DBMS

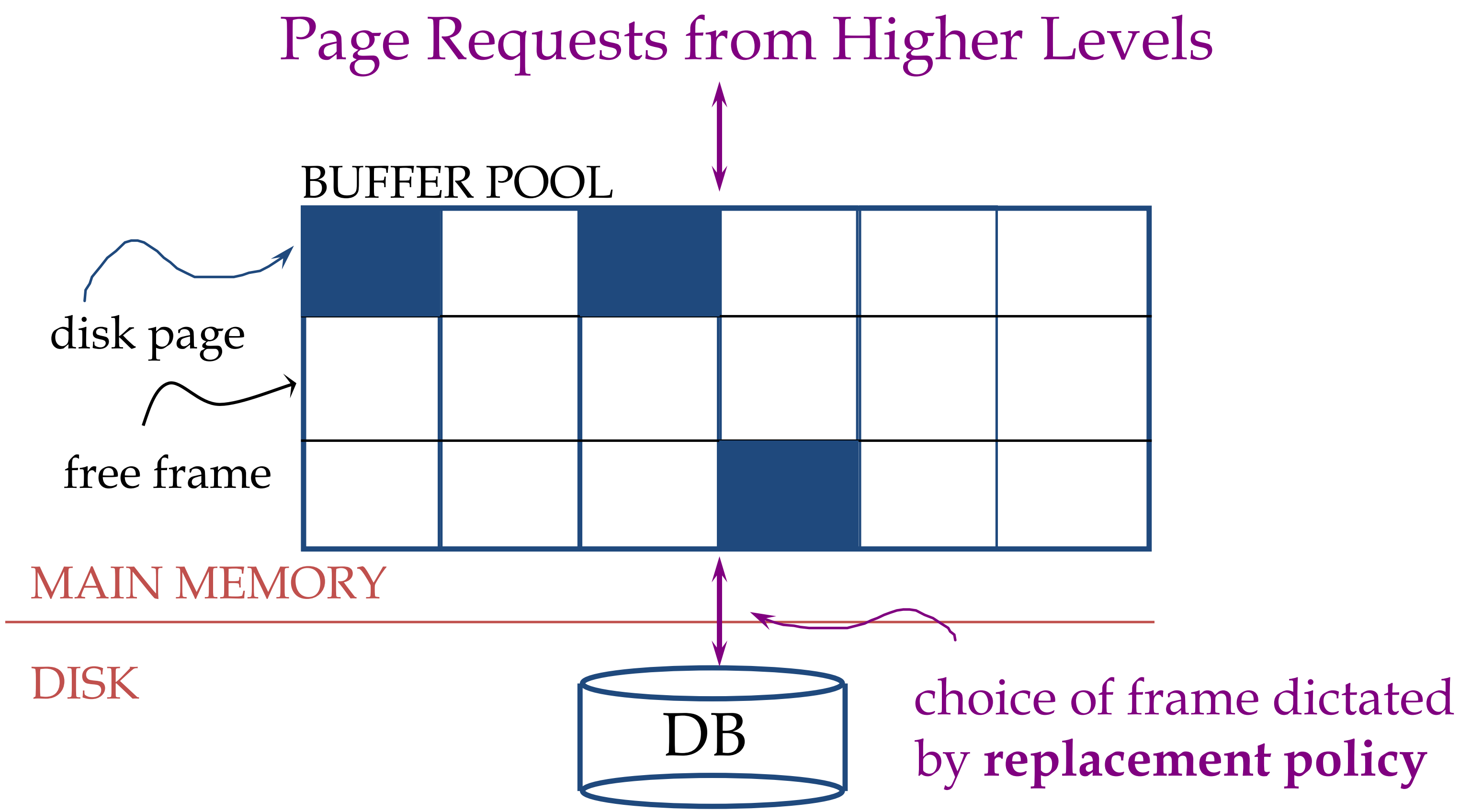


Internal layout of a heap table file in PostgreSQL

- Inside the data file (heap table), it is divided into **pages** (or **blocks**) of fixed length, the default is 8192 byte (8 KB).
- Those pages within each file are numbered sequentially from 0, and such numbers are called as **block numbers**.
- If the file has been filled up, PostgreSQL adds a new empty page to the end of the file to increase the file size.



Buffer Management in a DBMS



- Data must be in RAM for DBMS to operate on it!
- Buffer Mgr hides the fact that not all data is in RAM

Buffer Manager

- Manages buffer pool: the pool provides space for a limited number of pages from disk.

Needs to decide on page replacement policy.

Enables the higher levels of the DBMS to assume that the needed data is in main memory.

When a Page is Requested ...

- Buffer pool information table contains:
  <frame#, pageid, pin_count, dirty>

- If requested page is not in buffer pool:
  - Choose a frame for *replacement.*
    Only "un-pinned" pages are candidates!
  - If frame is "dirty", write it to disk
  - Read requested page into chosen frame

- *Pin* the page and return its address.

Overview

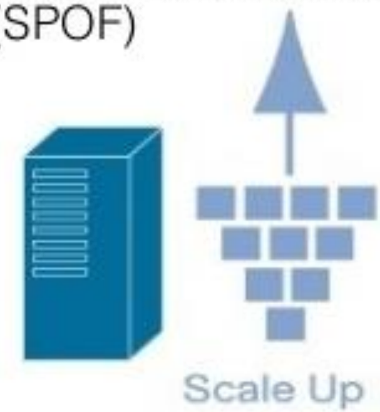
- Why storage is important for data systems?
- History of storage systems evolution
- Disks internals and data access from disks
- Software-based techniques
- Distributed storage management

Scaling vertically vs. horizontally

<u>Vertical Scaling</u>
Increases the power of existing system by adding more powerful hardware.

<u>Issues</u>:

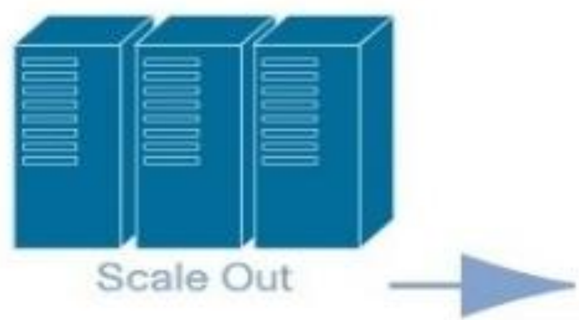
- Additional Investment
- Single point of failure (SPOF)



<u>Horizontal Scaling</u>
Adds extra identical boxes to server.

<u>Issues</u>:

- Requires Load balancer for managing connection.
- Distribution of work within the units becomes overhead.
- Additional investment.



Issues with ultra-scale data

- How to store the large amount of data?
  - On a large number of commodity hardware (horizontal scaling)

Google's data center in Oklahoma

Issues with ultra-scale data

- How to store the large amount of data?
  - On a large number of commodity hardware (horizontal scaling)

- How to manage the replication and the health of the large number of devices?

- More importantly, how to partition the large scale data to store in these storage devices (nodes)?

- Large storage implies large number of devices to store them.
  - How to address (mean time to) failure?

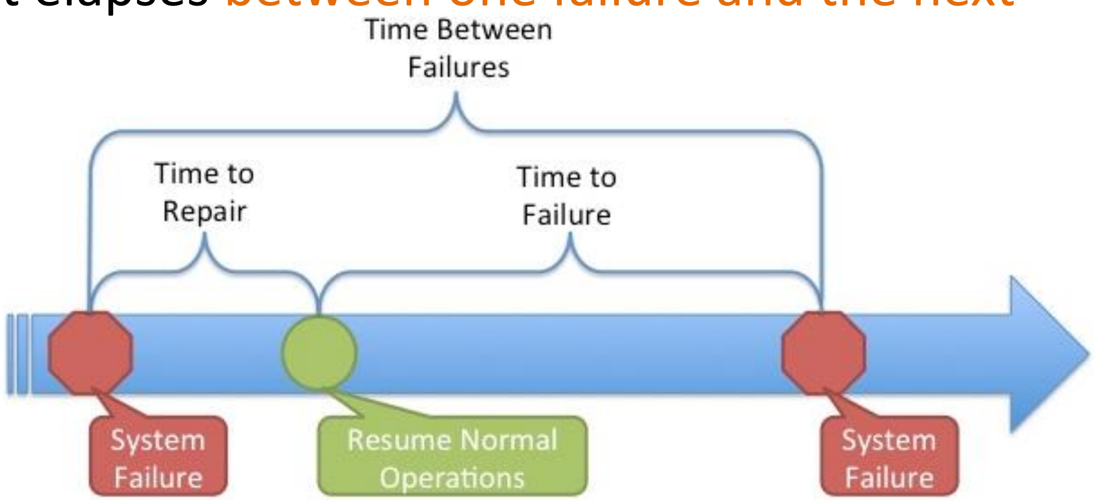
Reliability of a System (mainly disks)

- Metrics
  - Mean time to failure (MTTF): the length of time that a system is online between outages or failures. Often known as "uptime"

More disks mean more chances of failures? Or less?

o

  - Mean time between failure (MTBF) = MTTF + MTTR: amount of time that elapses between one failure and the next



Src: Stephen Foskett

Failure probability in a set of disks

- Metric
  - Mean time to failure of ***n*** disks ($MTTF_n$) = $MTTF_1$ / n

- Example (disks are assumed to be identical and independent)
  - If mean time to failure of a disk drive $MTTF_1$ is 50,000 hours (5.7 years)
  - $MTTF_n$ of 100 identical disks drops to 500 hours (i.e., 21 days)

Issues with ultra-scale data (contd.)

- How to store the large amount of data?
  - On a large number of commodity hardware (horizontal scaling)

- How to manage the replication and the health of the large number of devices?

- More importantly, how to partition the large scale data to store in these storage devices (nodes)?

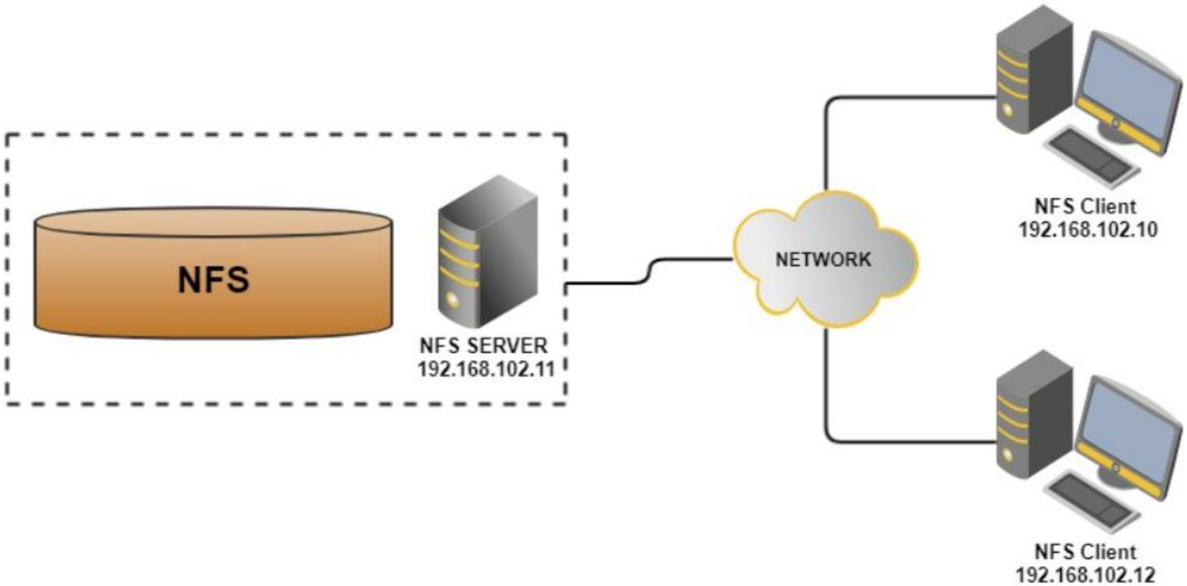
- Large storage implies large number of devices to store them.
  - How to address shortening MTTF (Mean time to failure)?
  - How to realize “fault tolerance”?
  - Redundancy/replication is a solution

Distributed file system (DFS)

- A dedicated server manages the files for a compute environment
- DFS addresses various transparencies: location transparency, sharing, performance etc.
- Examples: NFS, AFS (Andrew FS)
- Is DFS good for Internet-scale applications?

NFS

- Network File System (NFS)
  - A distributed file system protocol developed by Sun Microsystems in 1984
  - Allows a user on a client computer to access files over a computer network much like local storage is accessed.
  - NFS, like many other protocols, builds on the Remote Procedure Call (RPC)



Src: https://linuxhowtoguide.blogspot.com

NFS Limitations

- Limited storage capacity
  - The files reside on a single machine. this means that it will only store as much information as can be stored in one machine.
- Limited reliability
  - It does not provide any reliability guarantees if that machine goes down.
- Single point of failure
  - All the clients must go to this machine to retrieve their data. This can overload the server if a large number of clients must be handled. Clients must also always copy the data to their local machines before they can operate on it.

On to Google File

- Internet applications introduced a new challenge in the form web logs, web crawler’s data: large scale “peta scale”

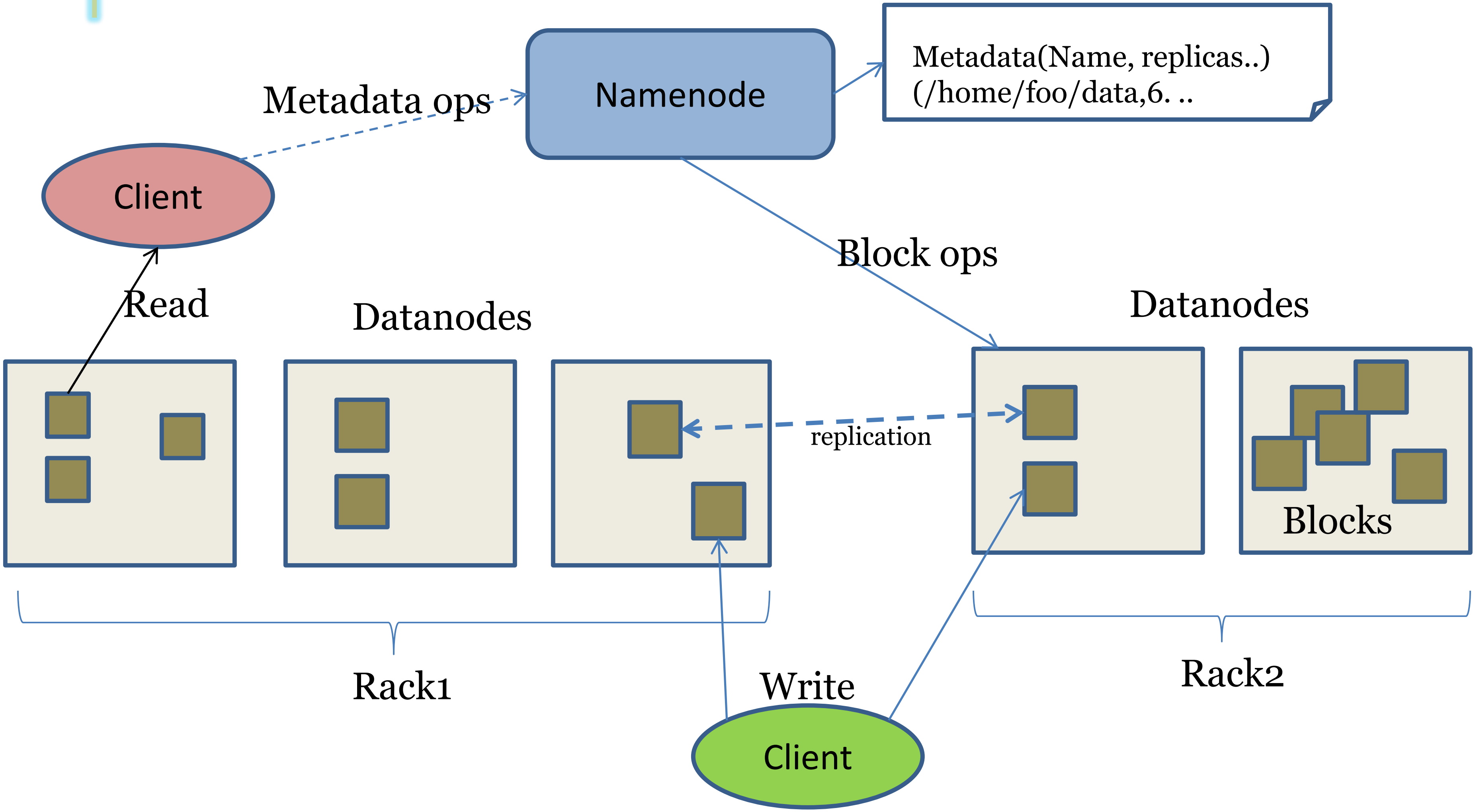
- But observe that this type of data has an uniquely different characteristic than your transactional or the “order” data on amazon.com: “write once”

- Google exploited this characteristics in its Google file system (GFS)

Hadoop Distributed File System (HDFS)

- HDFS is an open-source version of the GFS
- HDFS is a distributed file system for large scale data
- Advantages
  - Designed to store a very large amount of information. This requires spreading the data across a large number of machines.
  - It also supports much larger file sizes than NFS.
  - HDFS should store data reliably. If individual machines in the cluster malfunction, data should still be available.
  - Provides fast, scalable access to this information.
  - Integrate well with Hadoop MapReduce, allowing data to be read and computed upon locally when possible.

HDFS Architecture



Src: C B. Ramamurthy

Next

- Query processing review