CS 3502
Operating Systems

Storage

Kun Suo

Computer Science, Kennesaw State University

https://kevinsuo.github.io/

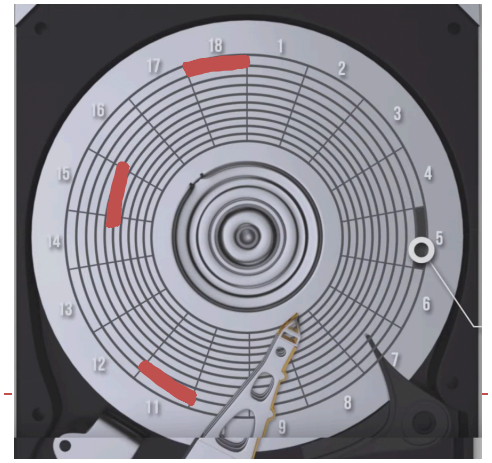
Outline

- File system reliability, consistency and performance
- Storage system structure
- Hard disk drive(HDD) vs SSD
- Case study: Distributed and Parallel File Systems

Reliability of File System

- Reliability: FS is capable of resisting or preventing various physical damages or human destroy

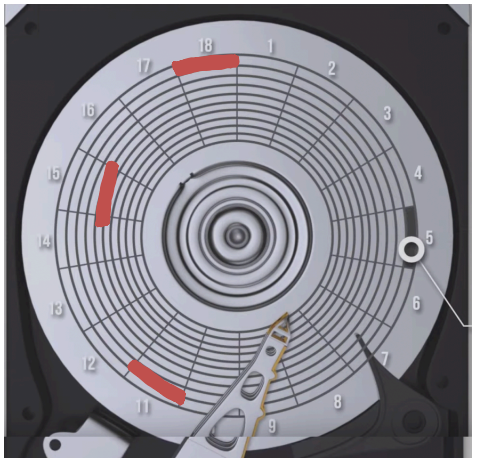
- Bad blocks, also called bad sectors, are sections of magnetic storage media (i.e., hard disks and floppy disks) that cannot be reliably used for storing and retrieving data



Reliability of File System

- Why bad blocks exist?
  - A disk will have some bad blocks during production in the factory. Yield rate (e.g., 99%)

  - New bad blocks will appear during the usage of the disk over long time.



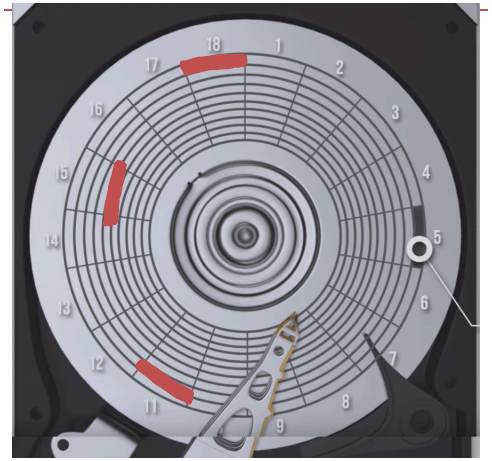
Reliability of File System

- In linux, using command *badblocks* to check bad sectors or blocks on HDD

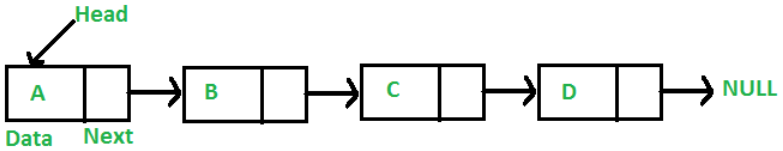
```
tecmint@TecMint ~ $ sudo badblocks -v /dev/sda10 >badsectors.txt
Checking blocks 0 to 3906559
Checking for bad blocks (read-only test):
done
Pass completed, 0 bad blocks found. (0/0/0 errors)
tecmint@TecMint ~ $
tecmint@TecMint ~ $
```

Reliability of File System

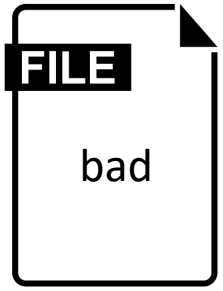
- How to deal with bad blocks?
  - Linked all bad blocks together and marked



  - Put all the bad blocks in a single file, called bad block file



  - Allocate new created bad blocks to this file and never use it



Reliability of File System

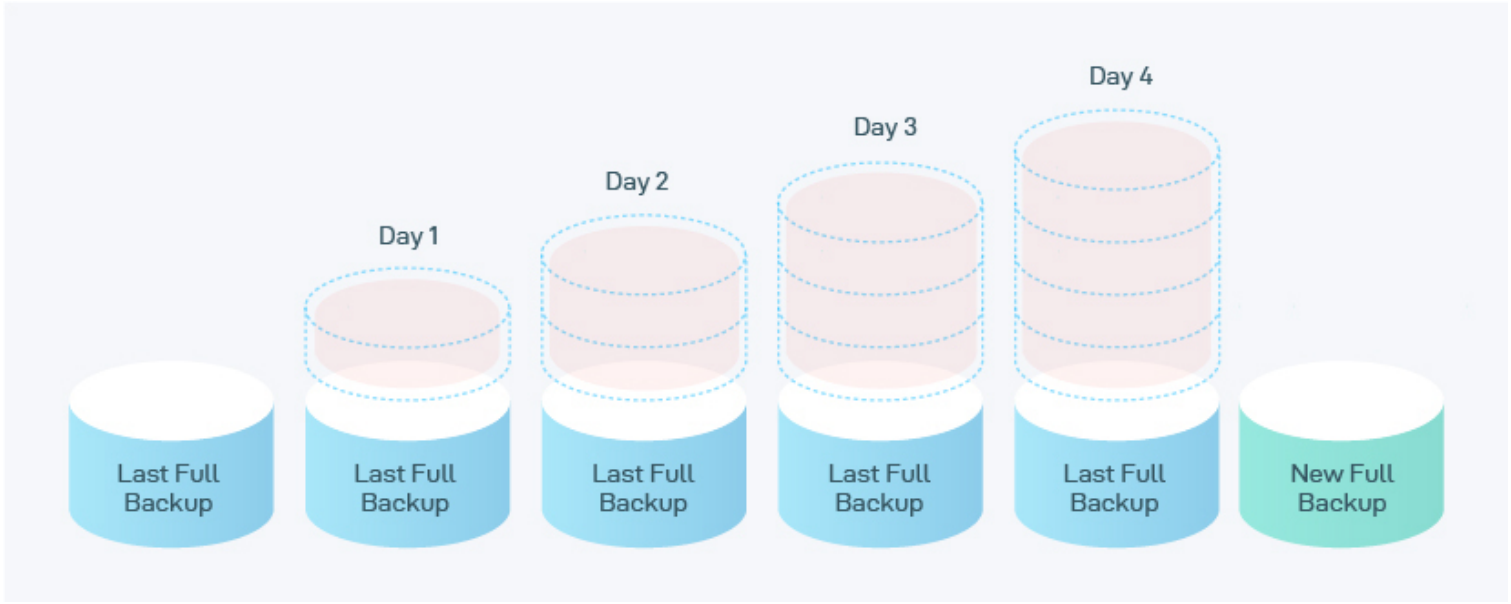
- Backup: to form multiple copies of the file through a dump operation

- Dump:
  - Full dump/backup: periodically backup all files to some storage devices, e.g., tape
  - Delta dump/incremental dump: just backup the changes between two backups

Reliability of File System

	Time for backup	Backup operation	Time for restore	Restore operation
Full dump	Long	Simple, just copy all files	Short	Restore based on the entire mirror image
Delta dump	Short	The deltas must be applied in the order they were produced. Each delta applied must have been produced with the previously applied delta as its base.	Long	Restore must be applied in the reverse order of produced deltas

Reliability of File System



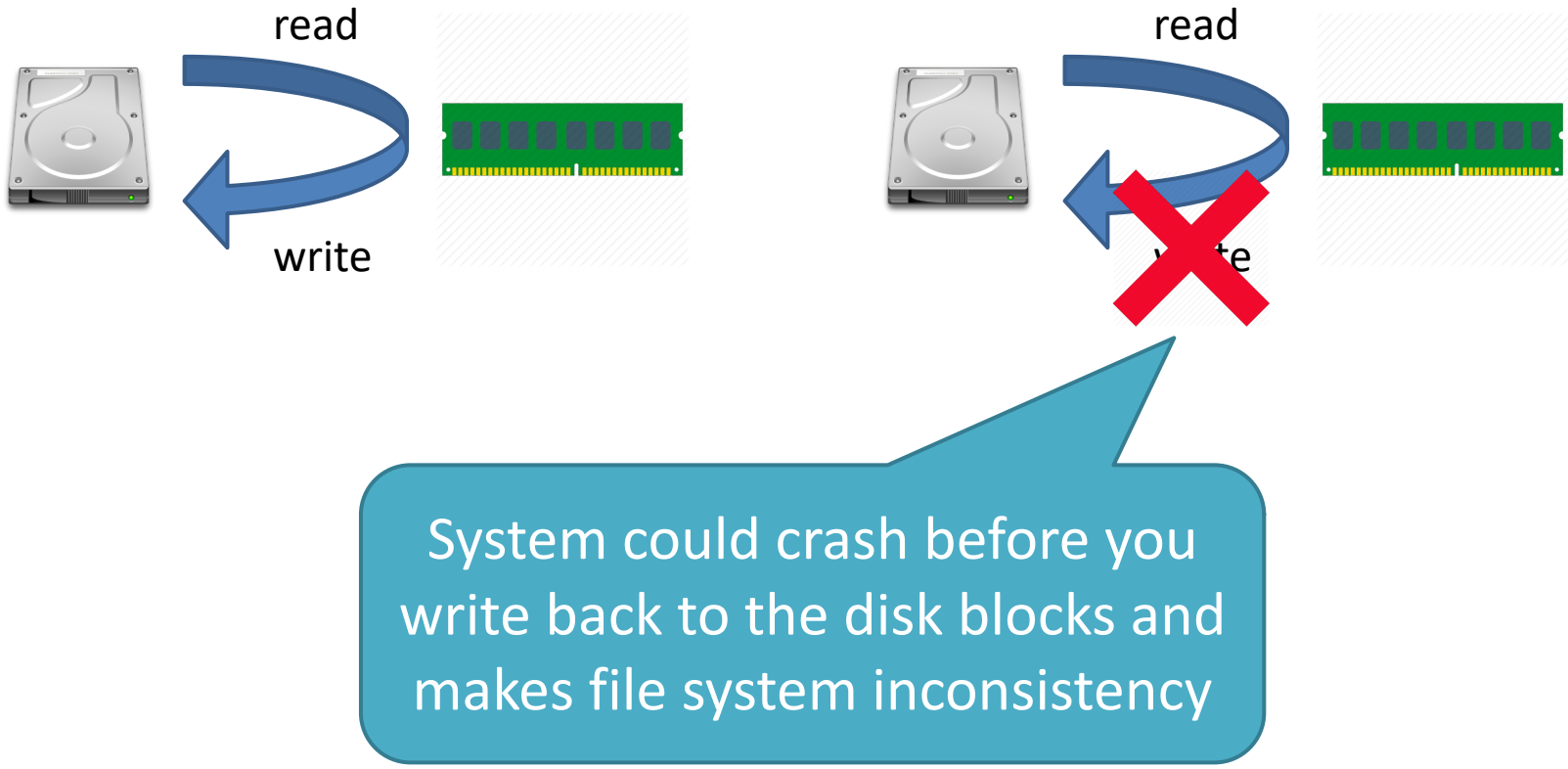
Incremental backups
between two full backups

Full backup every
N days/weeks

Full backup every
N days/weeks

Consistency of File System

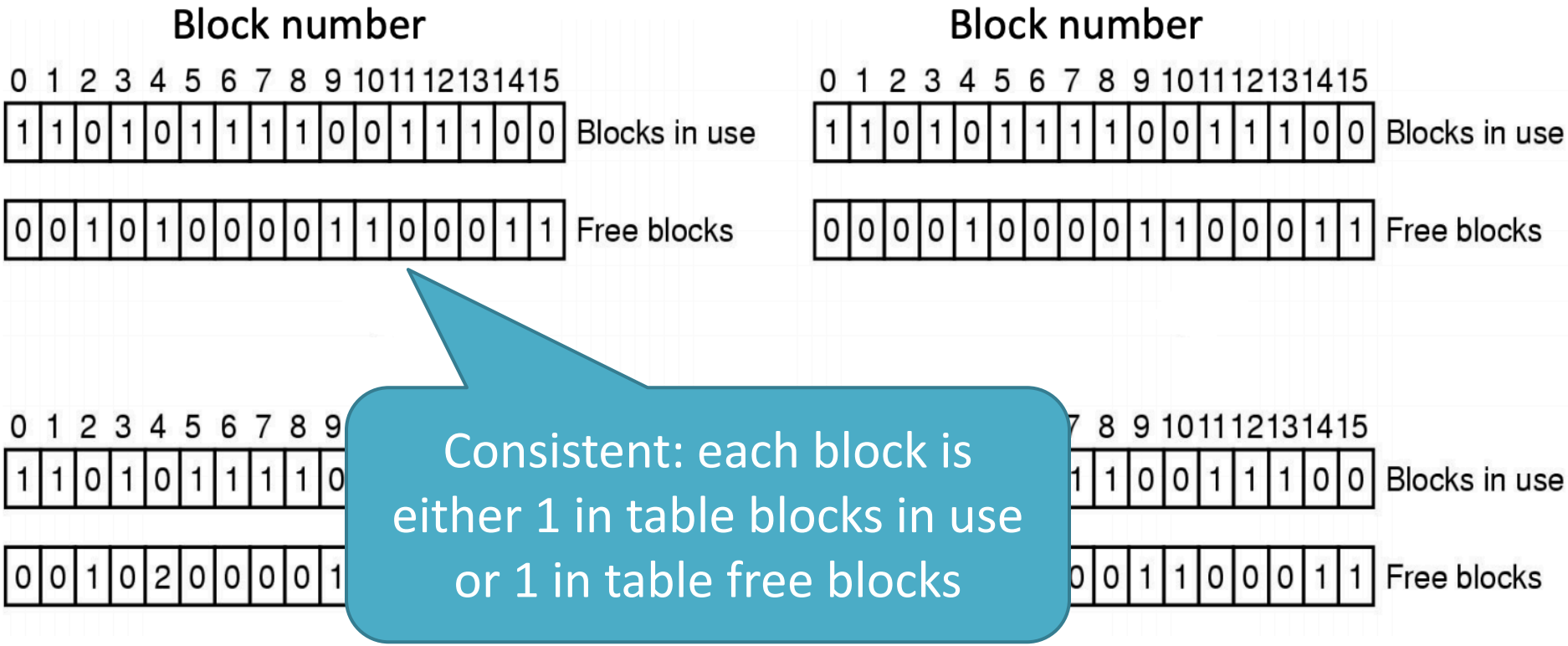
- Problem definition:



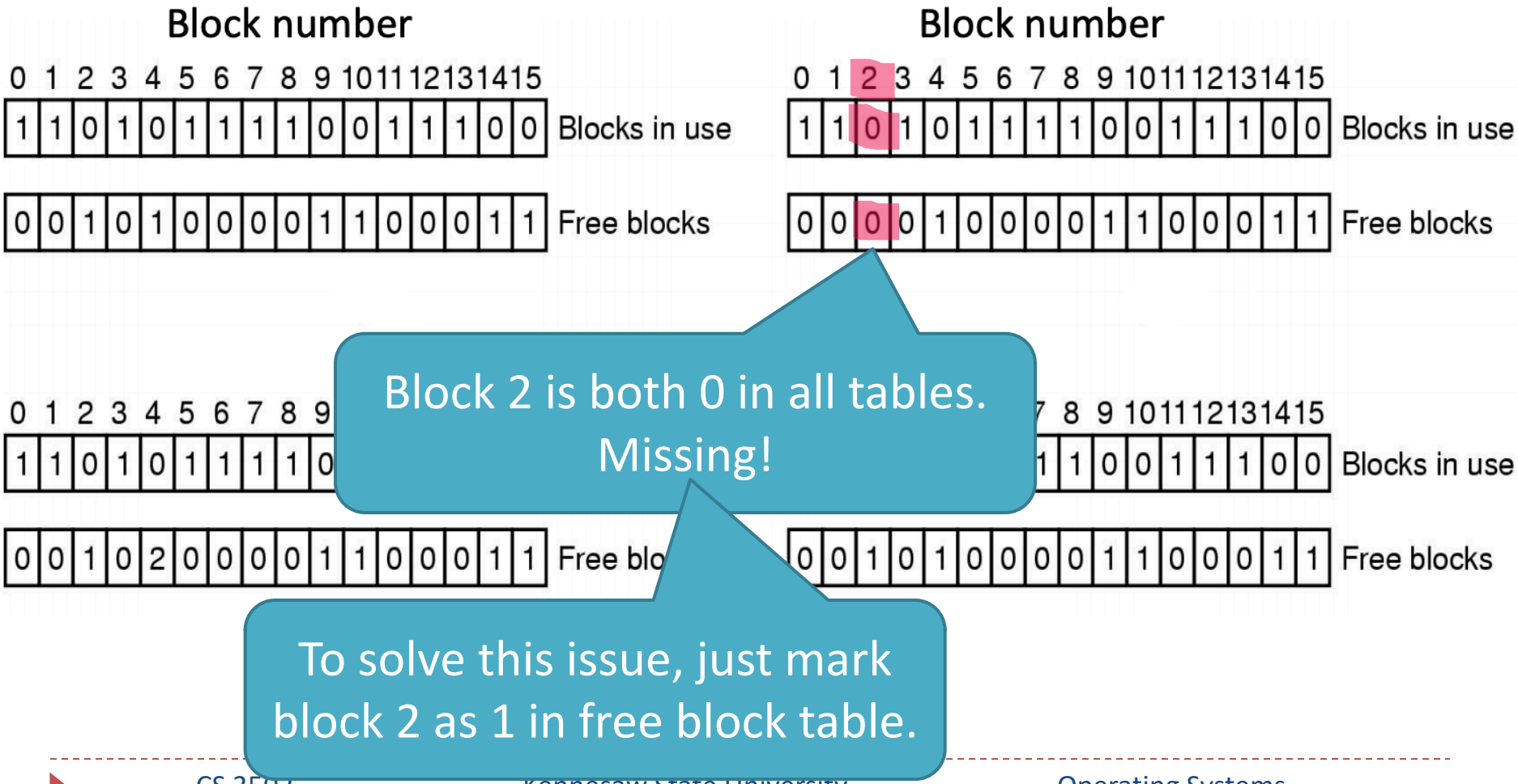
Consistency of File System

- How to deal with consistency problem in file system?
  - Design a kernel program check the block usage statistics when the OS boots
  - Maintain two tables, the item in the table is initially 0
  - Table 1 records the number of block appearing in all files
  - Table 2 records the number of block appearing in free space

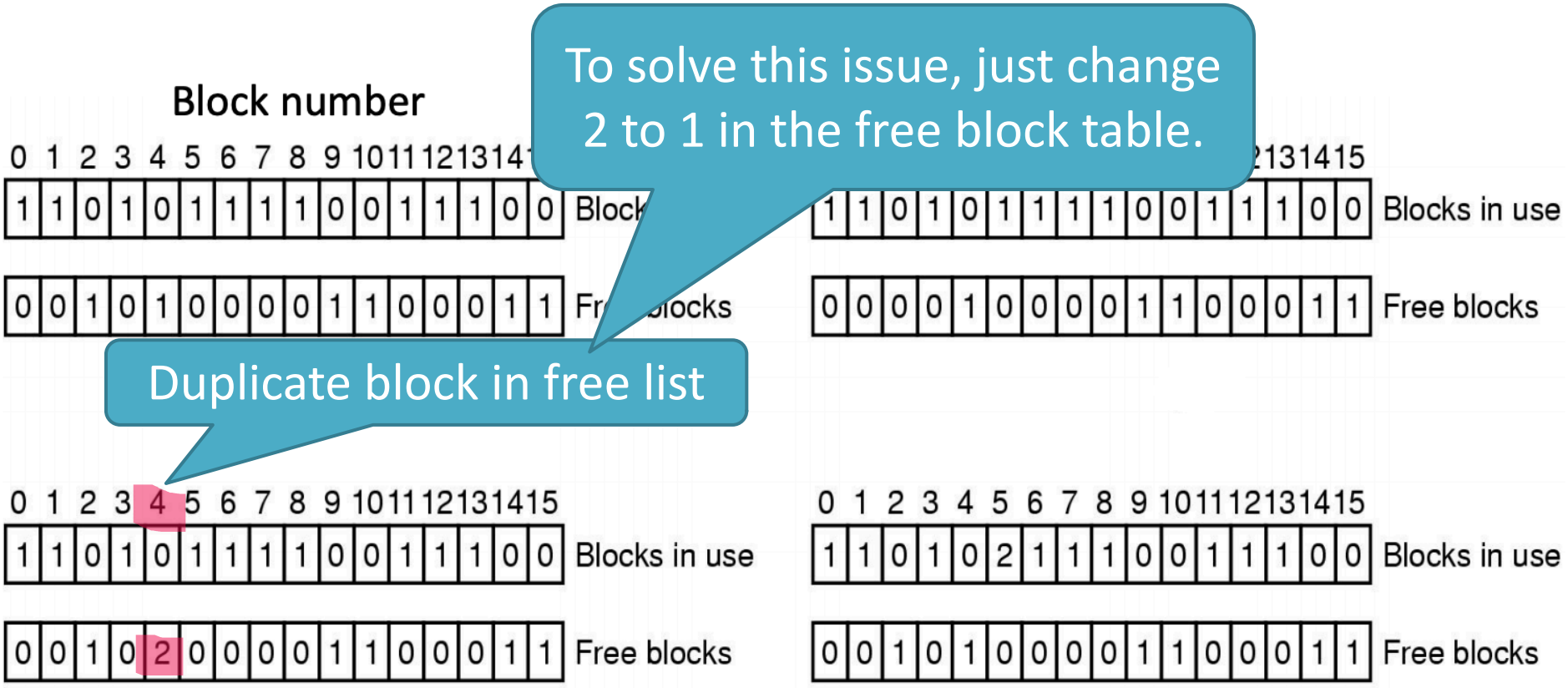
Consistency of File System



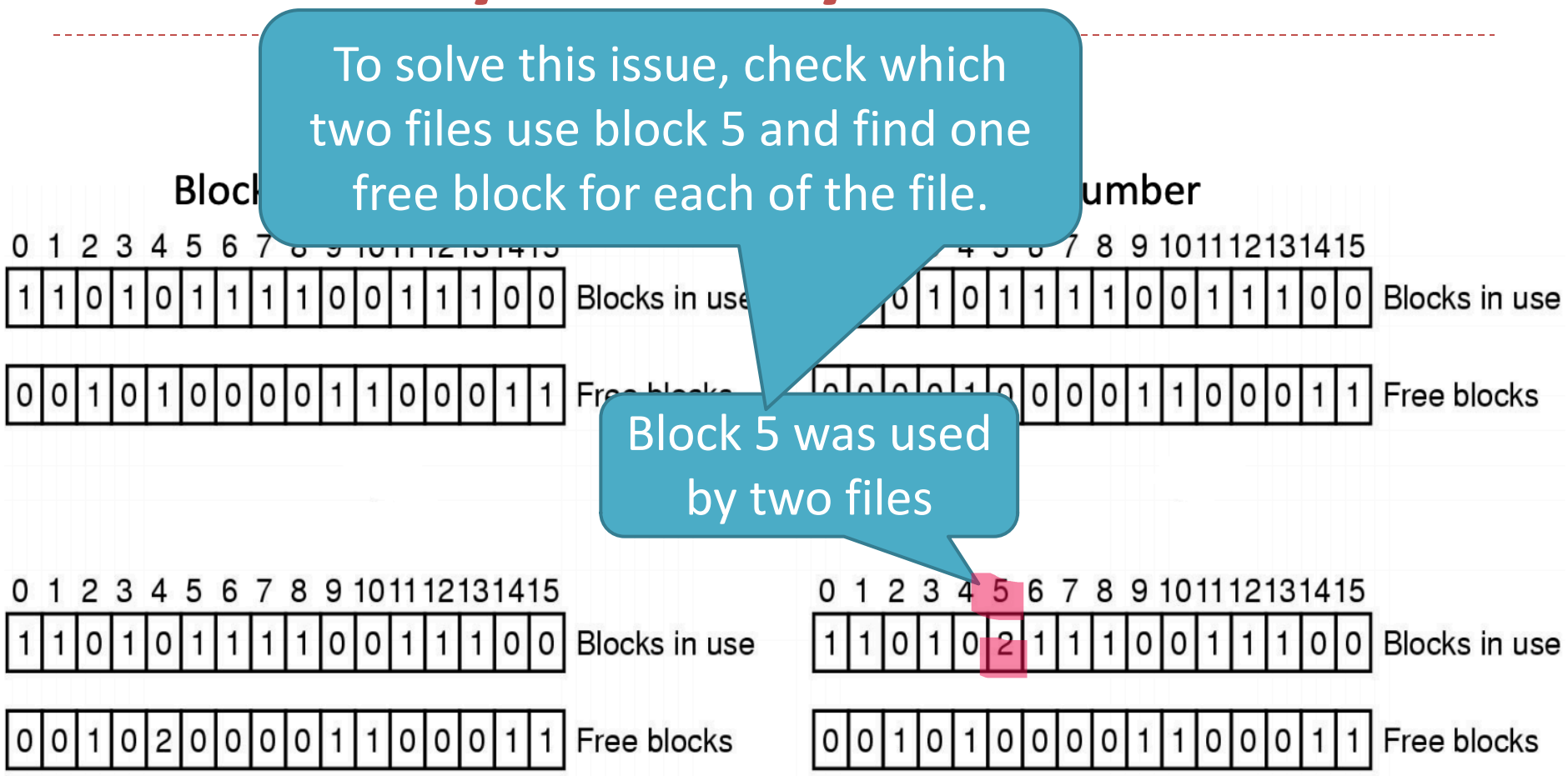
Consistency of File System



Consistency of File System

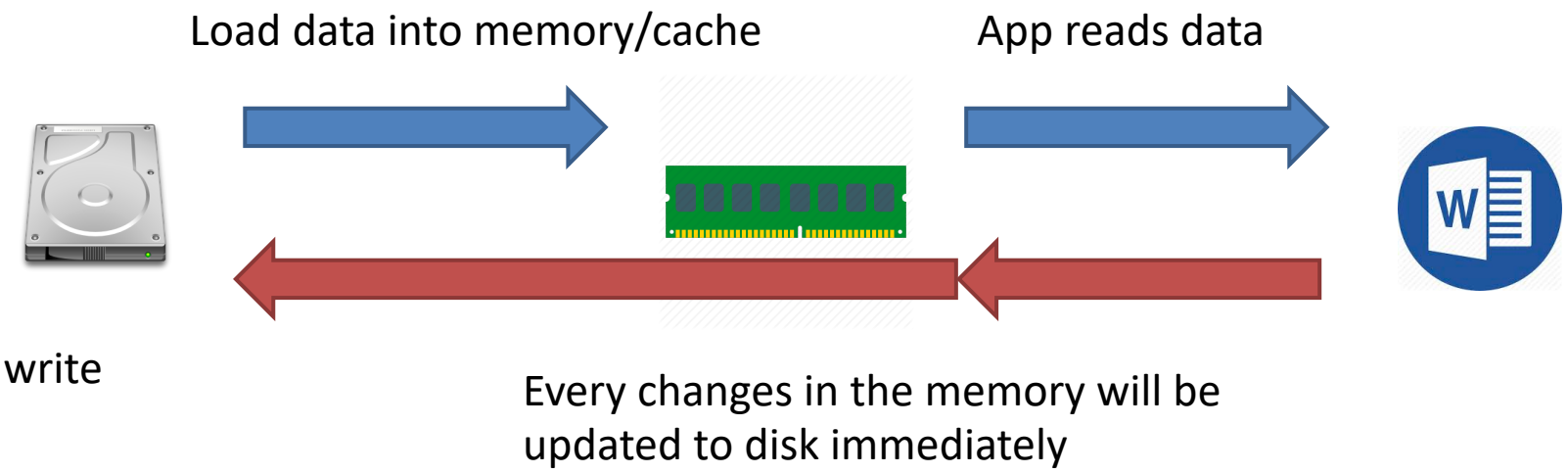


Consistency of File System

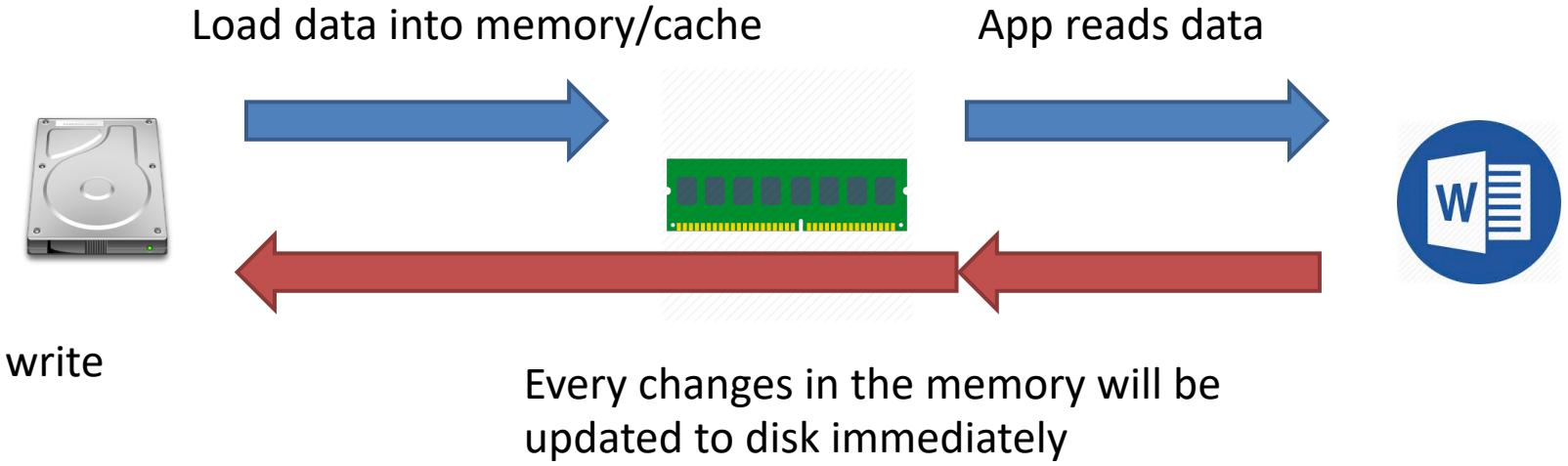


Performance of File System

- **Write through** is a storage method in which data is written into the cache and the corresponding main memory location *at the same time*



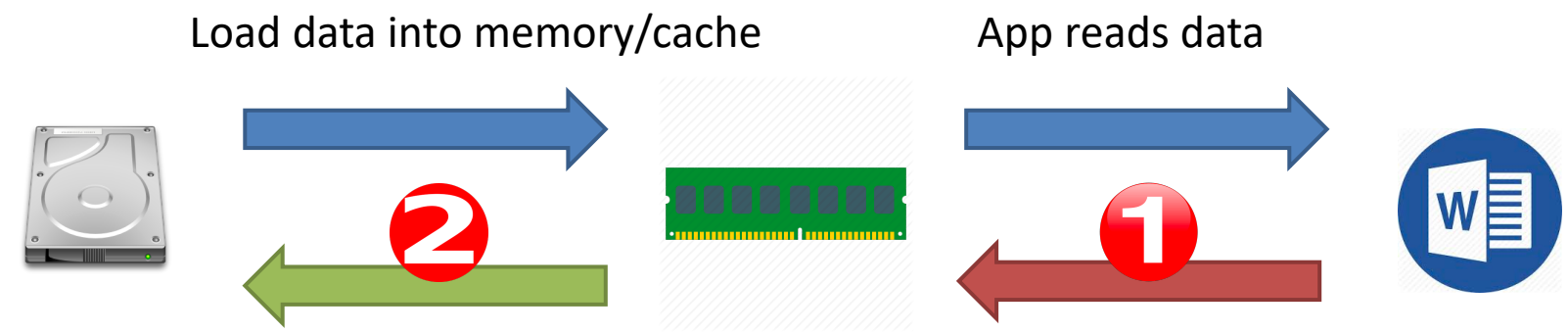
Performance of File System



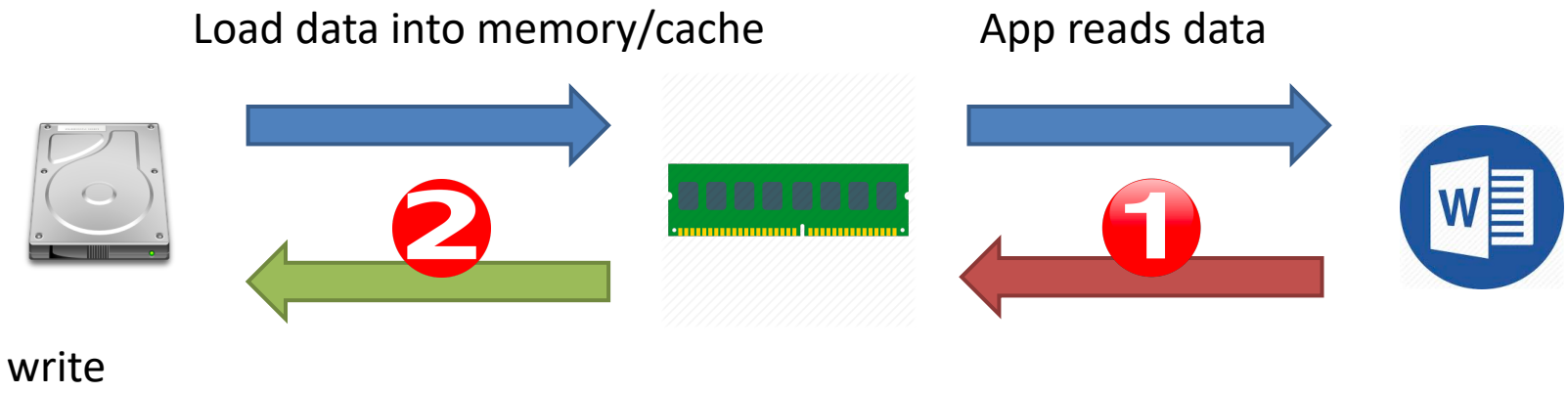
- **Adv:**
  - Good consistency, simple design and easy restore
- **Disadv:**
  - Bad performance due to frequent updates

Performance of File System

- **Write back** is a storage method in which data is written into the memory every time a change occurs, but is written into the corresponding location in disk when the data in memory is replaced



Performance of File System

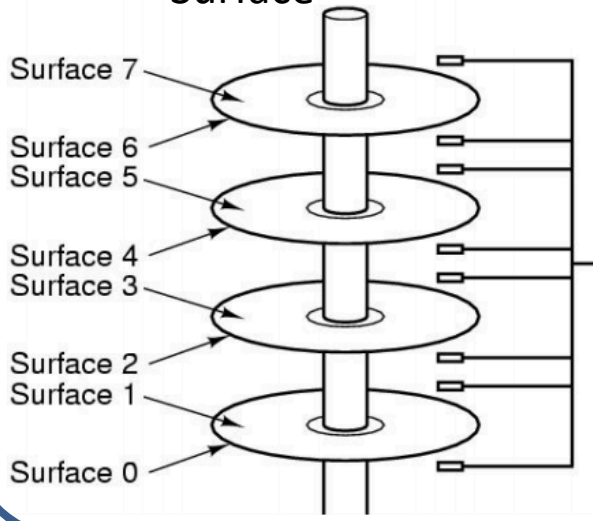


- **Adv:**
  - High performance and efficiency
- **Disadv:**
  - Complex design and implementation
  - Difficult restore

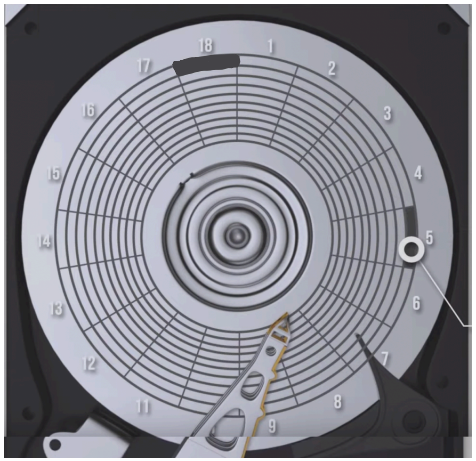
Unit of File System: Blocks

- Files and directories are allocated in blocks
- Block size is usually a multiple of the sector size

Surface

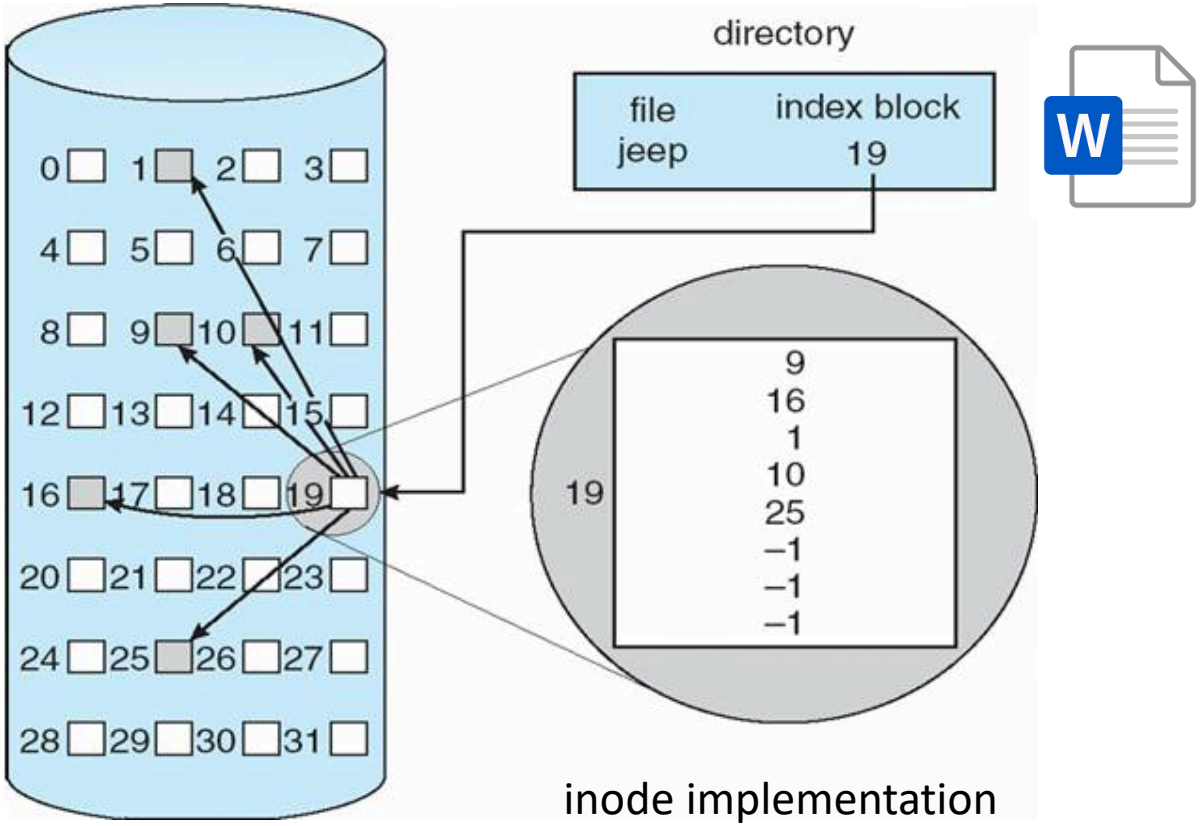


Track and sector



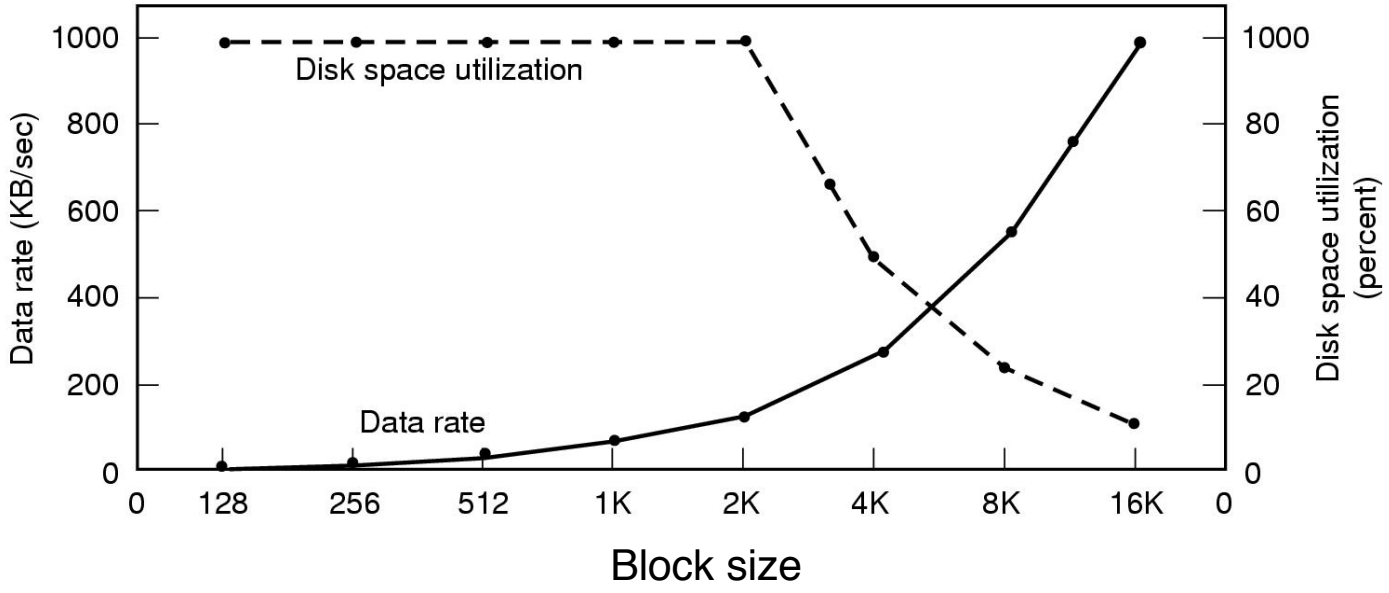
Performance of File System – Block Size

- All file systems chop files to fixed-size non-adjacent blocks



Performance of File System – Block Size

- Block size is a trade-off of space utilization and data rate



  - Dark line (left hand scale) gives data rate (read/write) of a disk
  - Dotted line (right hand scale) gives disk space efficiency

Performance of File System – Block Size

- Block size is a trade-off of space utilization and data rate
  - Block size ↗
  - Data rate ↗
  - Disk space utilization ↘

Positive correlated

Negative correlated

File size = 5

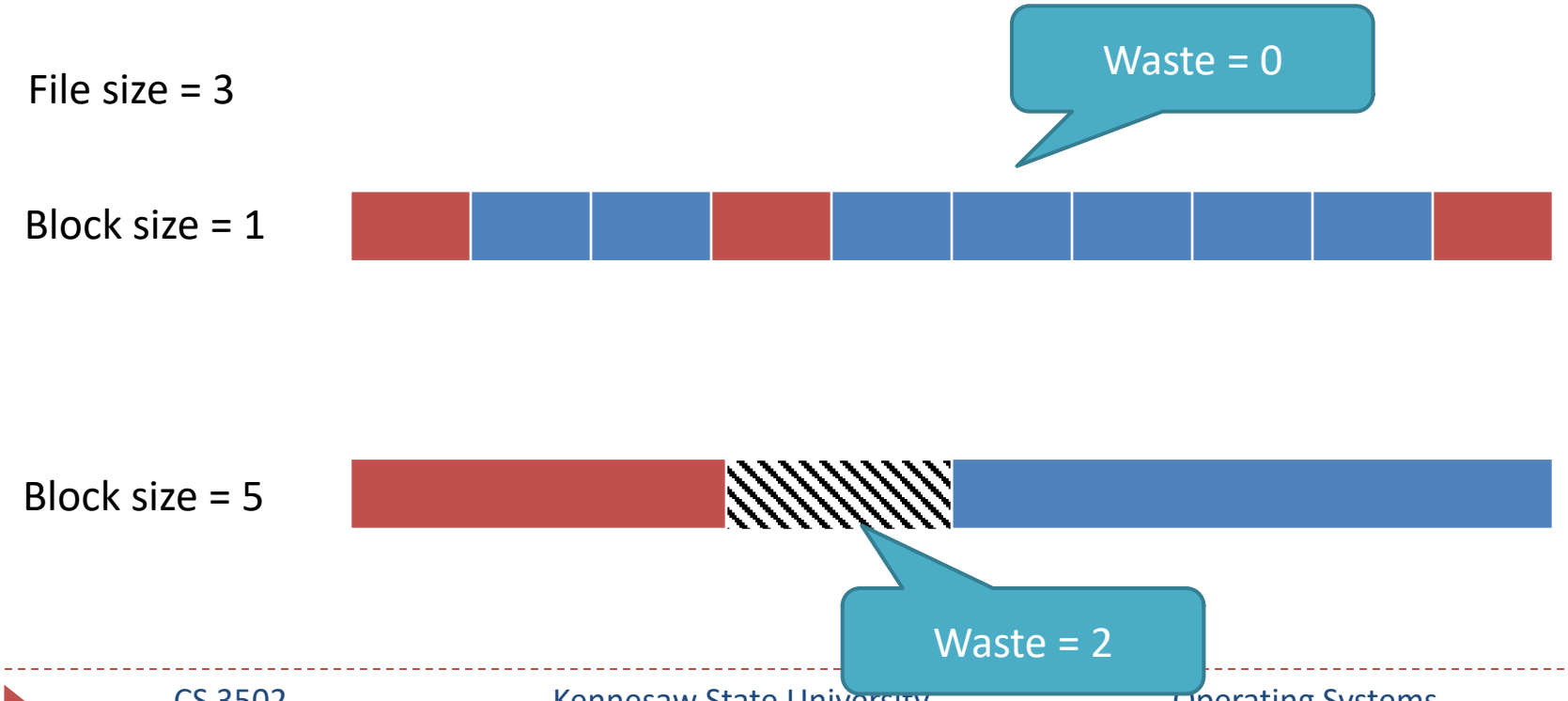
Block size = 1

Block size = 5

Read continuous data is much faster

Performance of File System – Block Size

- Block size is a trade-off of space utilization and data rate
  - Block size ↗
  - Data rate ↗
  - Disk space utilization ↘



Performance of File System – Block Size

- Block size is a trade-off of ***space utilization*** and ***data rate***
- What is best block size?
  - 60%-70% files below 4KB, 93% of the disk occupied by 10% largest files
- Block size in popular OSes:
  - Mac: 512 bytes
  - Linux: 1024 bytes
  - Windows: 4096 bytes

Outline

- File system reliability, consistency and performance
- Storage systems structure
- Hard disk drive(HDD) vs SSD
- Case study: Distributed and Parallel File Systems

Storage system vs. File system

- File system is the OS component that manages how and where data is stored, managed and accessed on the raw storage.

- File system is part of the storage system

- Besides files, we can store data into blocks, objects, etc.

https://blog.storagecraft.com/object-storage-systems/

Storage layers

User space

kernel space

hardware

Storage layers

User space

https://elixir.bootlin.com/linux/latest/source/drivers/block

https://elixir.bootlin.com/linux/latest/source/drivers/usb

kernel space

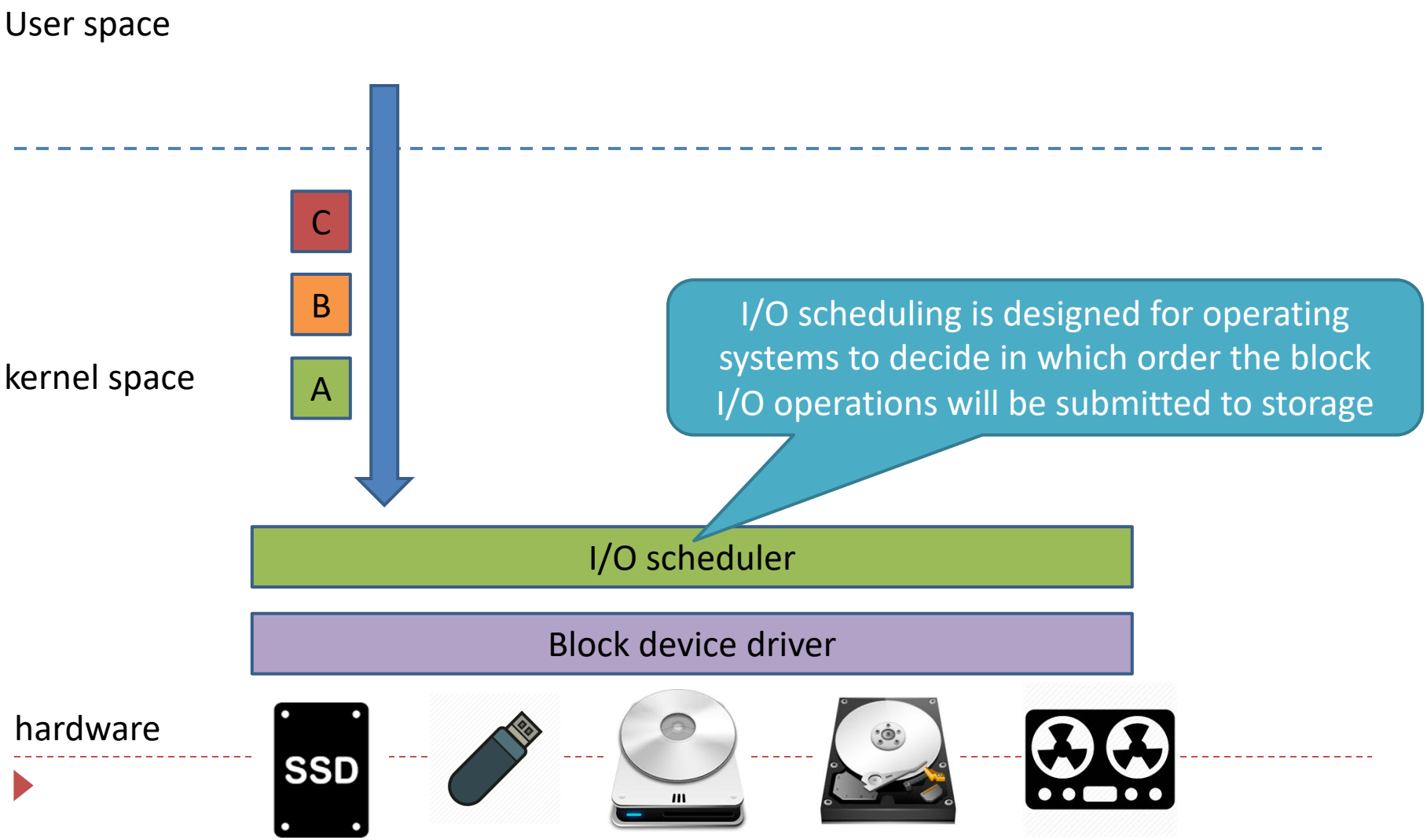
https://elixir.bootlin.com/linux/v4.8/source/drivers/cdrom

https://elixir.bootlin.com/linux/latest/source/drivers

Block device driver

hardware

Storage layers



Storage layers

User space

kernel space

Generic block layer hides the details of each hardware block device, thus offering an abstract view of the block devices

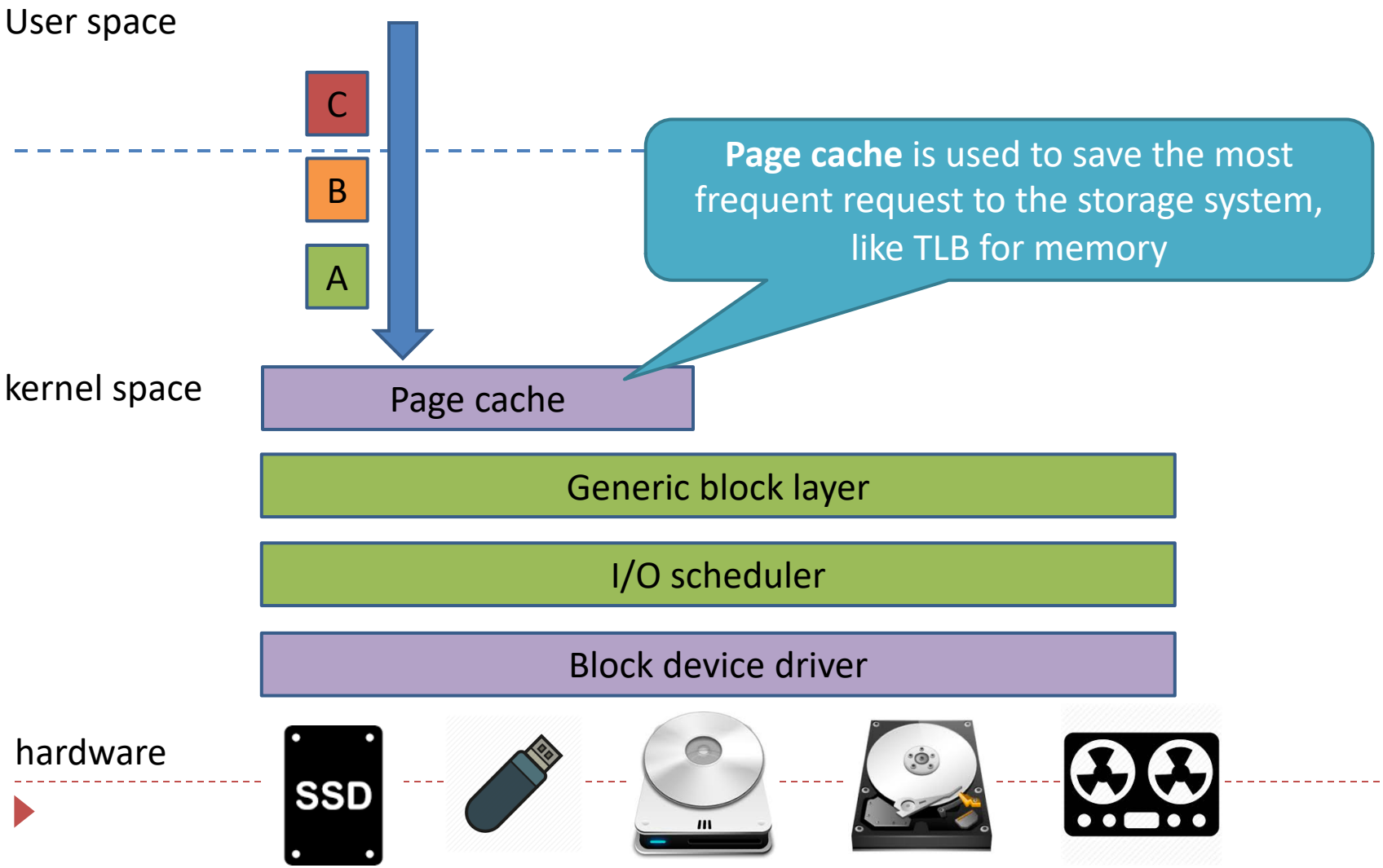
Generic block layer

I/O scheduler

Block device driver

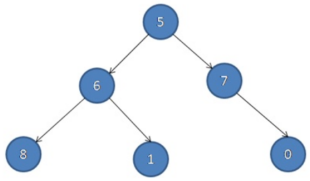
hardware

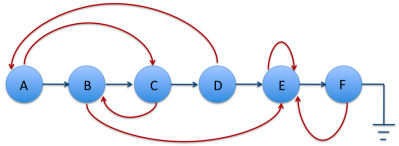
Storage layers

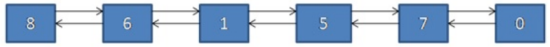


Storage layers

User space







File system is used to define how to organize data on the raw storage device

kernel space

File system (ext3, ext4, btrfs)

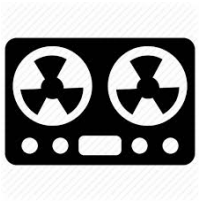
Page cache

Generic block layer

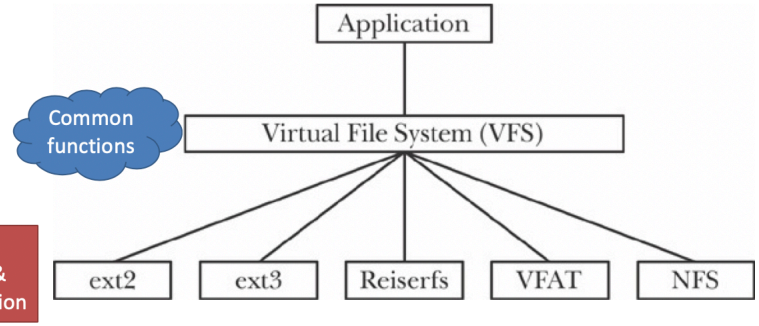
I/O scheduler

Block device driver

hardware



Storage layers



different functions & implementation

User space

An abstract layer with common functionalities supported by all the underlying concrete file systems

VFS

File system (ext3, ext4, btrfs)

kernel space

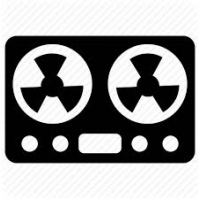
Page cache

Generic block layer

I/O scheduler

Block device driver

hardware



Storage layers

User space

Apps

System call interfaces

kernel space

VFS

File system (ext3, ext4, btrfs)

Page cache

Generic block layer

I/O scheduler

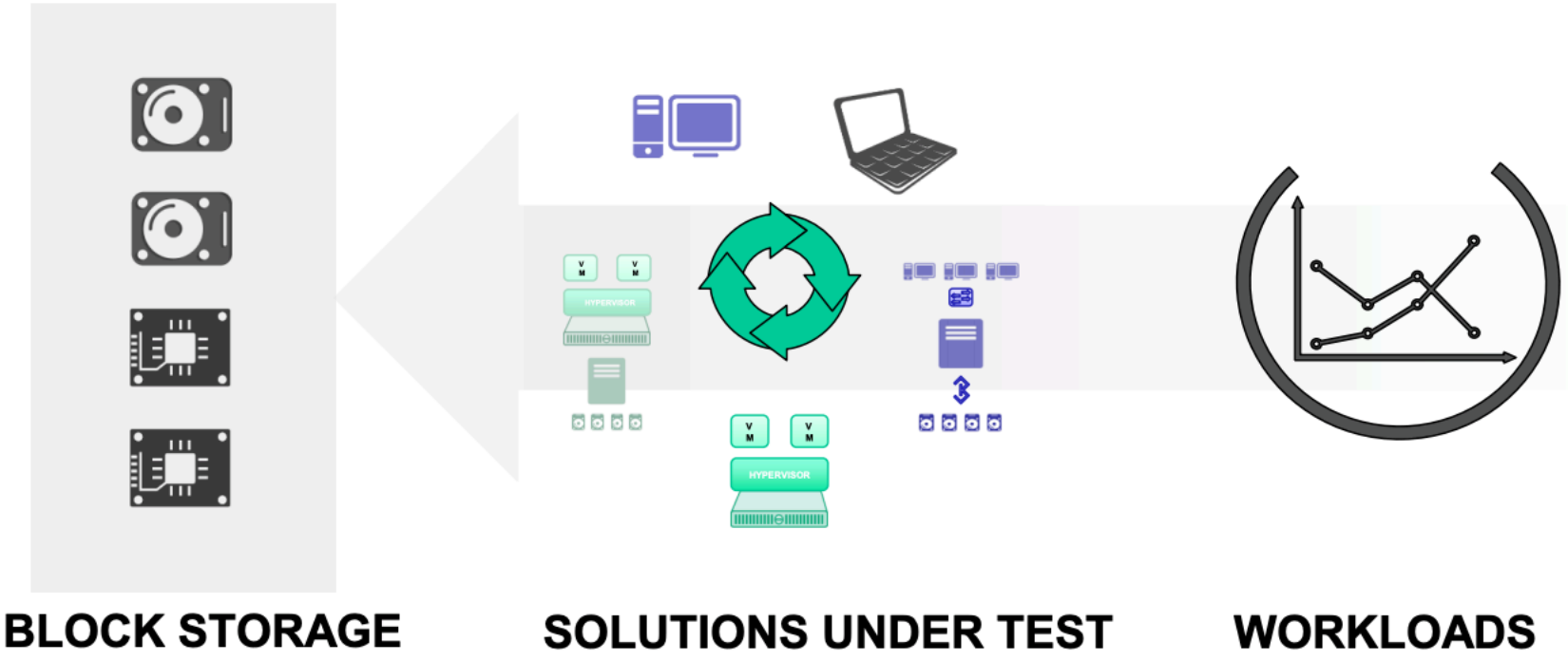
Block device driver

hardware

Outline

- File system reliability, consistency and performance
- Storage system structure
- Hard disk drive(HDD) vs SSD
- Case study: Distributed and Parallel File Systems

Eventually, All Data Goes To Block Storage



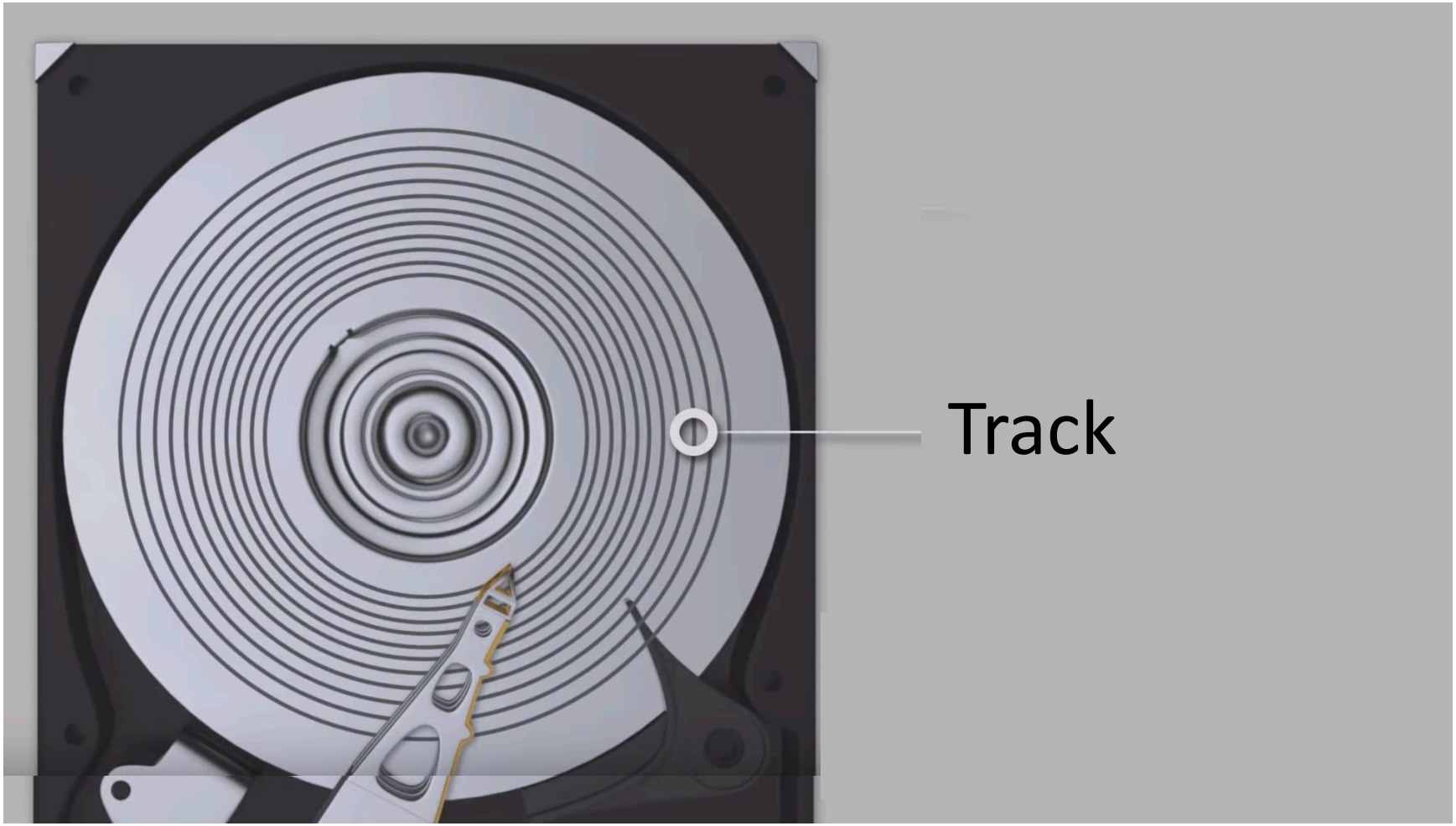
HDD vs. SSD

HDD OR DISK DRIVE

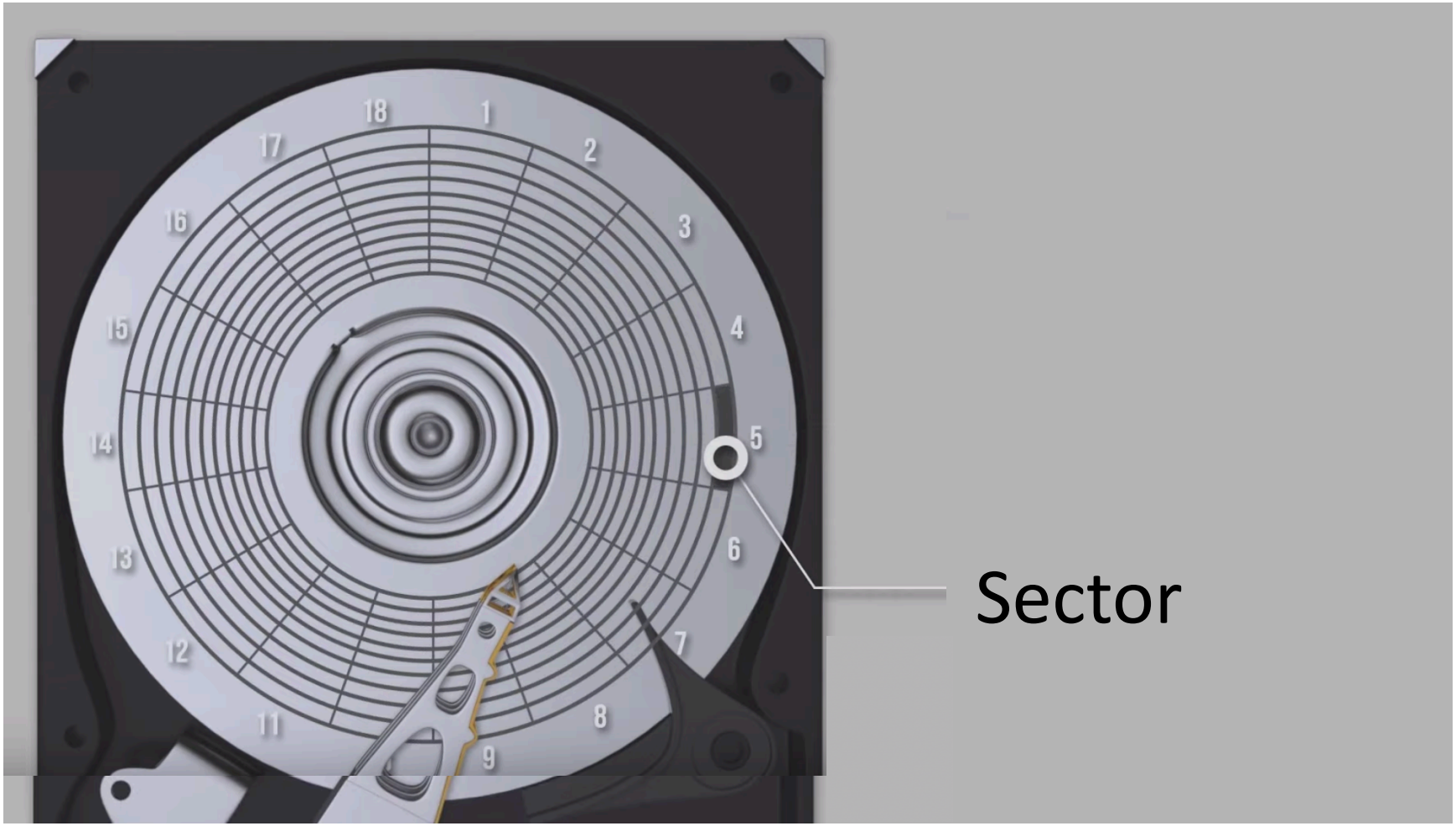
FLASH OR SSD

Hard disk drive

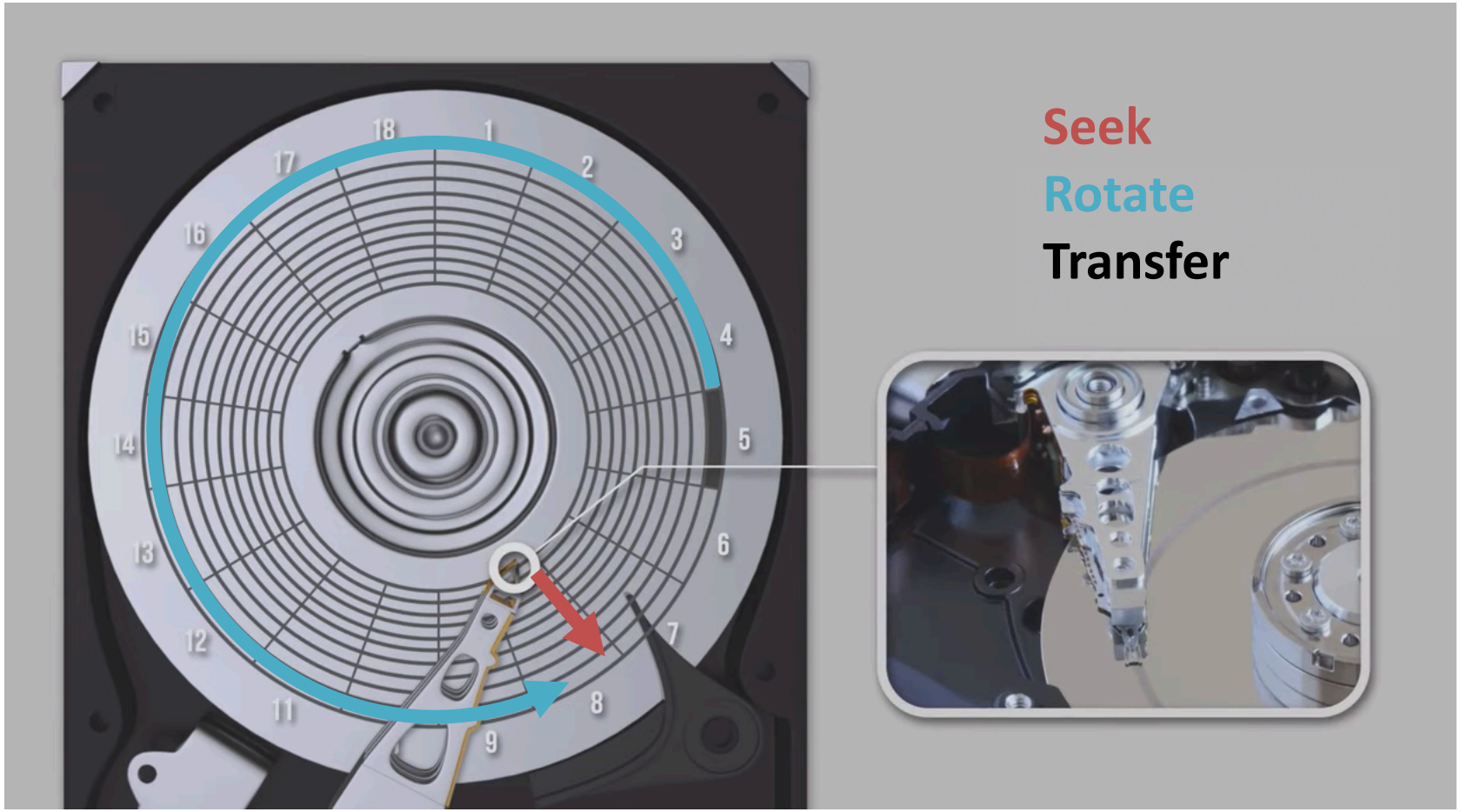
Hard disk drive



Hard disk drive

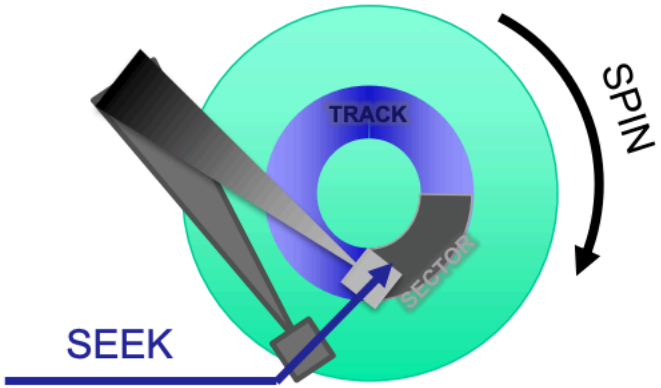


Spinning Drives Are Slow



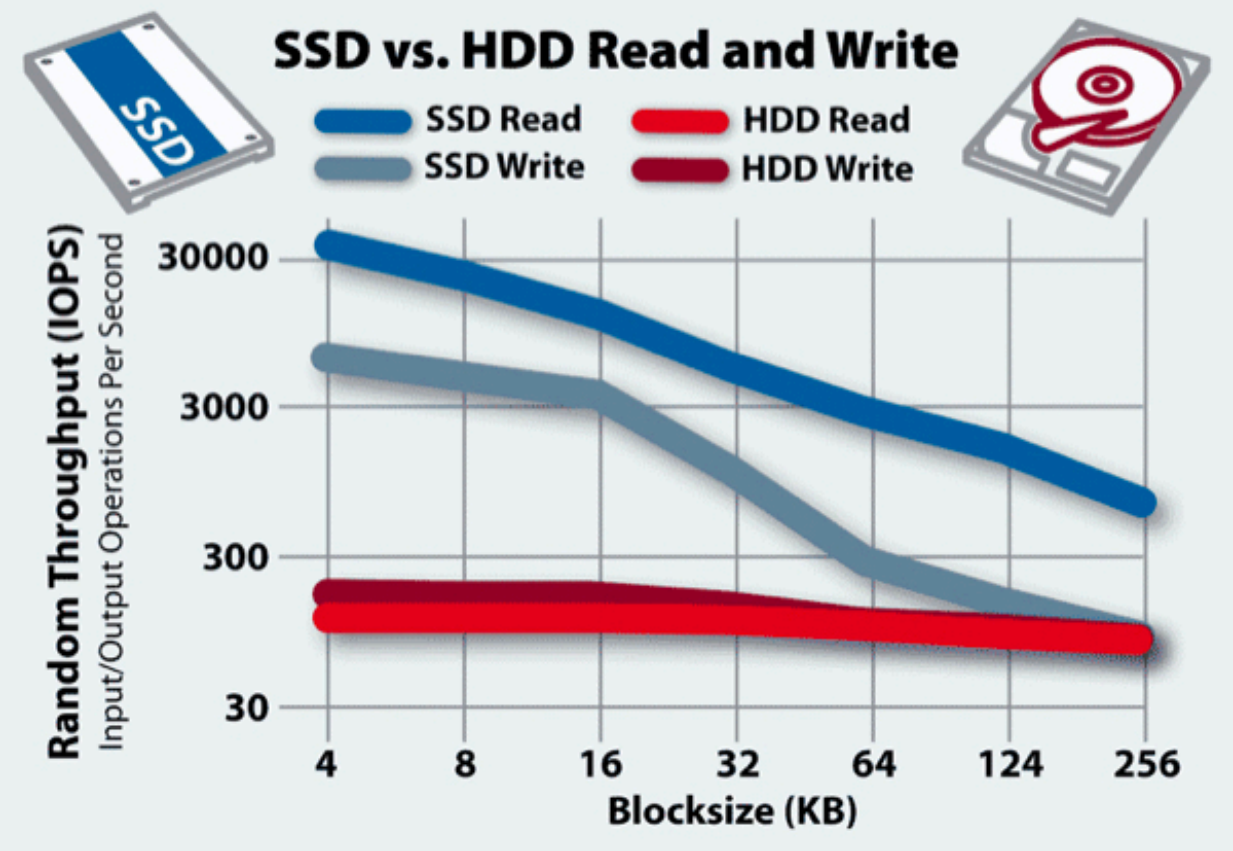
Spinning Drives Are Slow

- Random access to HDD is especially slow

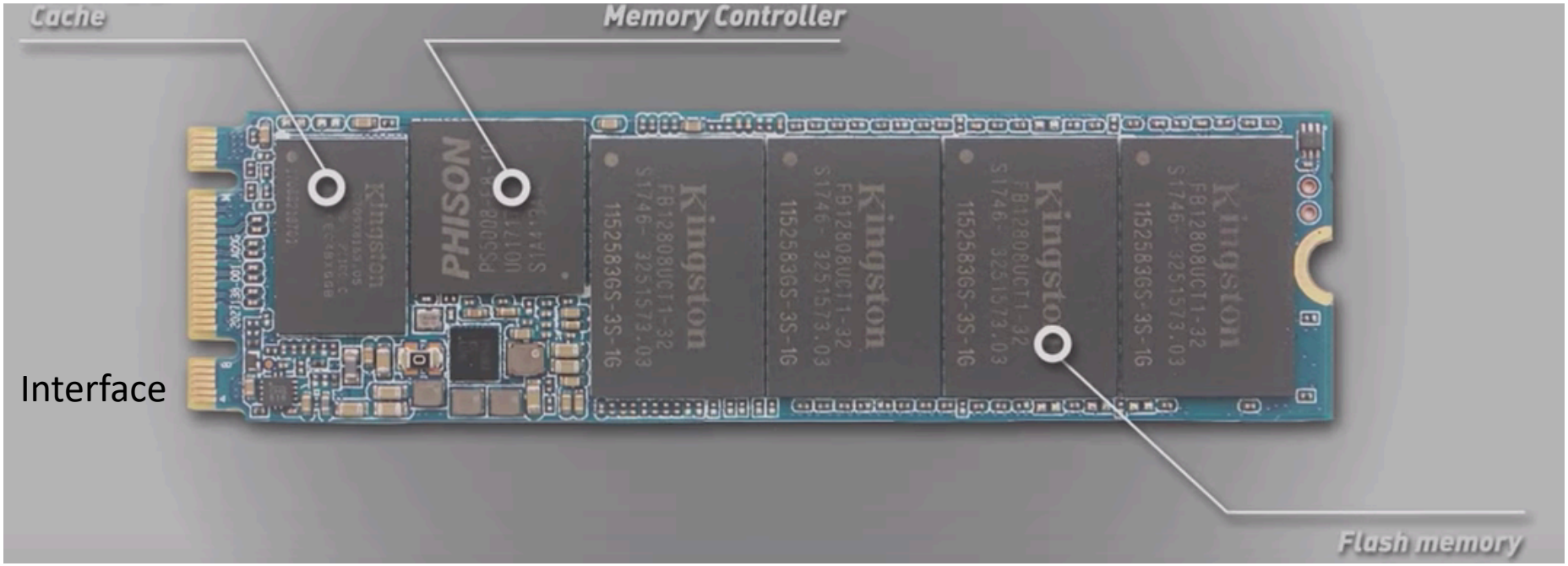


Spinning Drives Are Slow

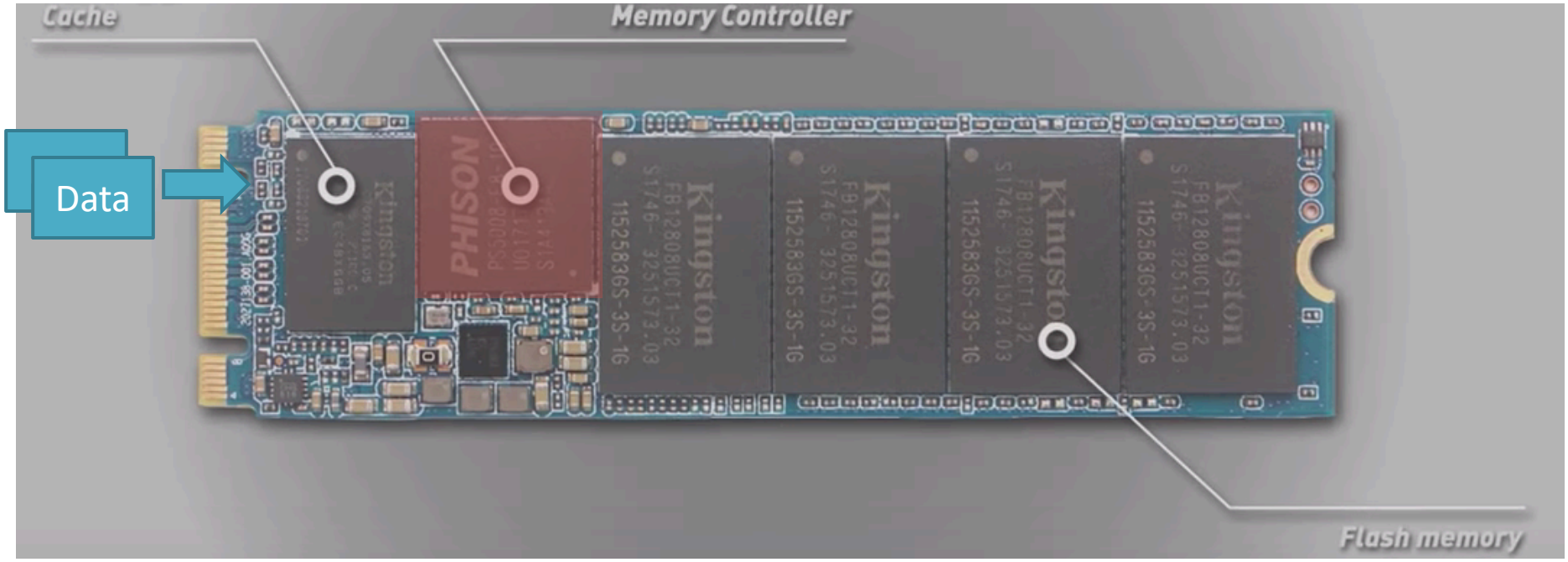
- Random access to HDD is especially slow



SSD structure

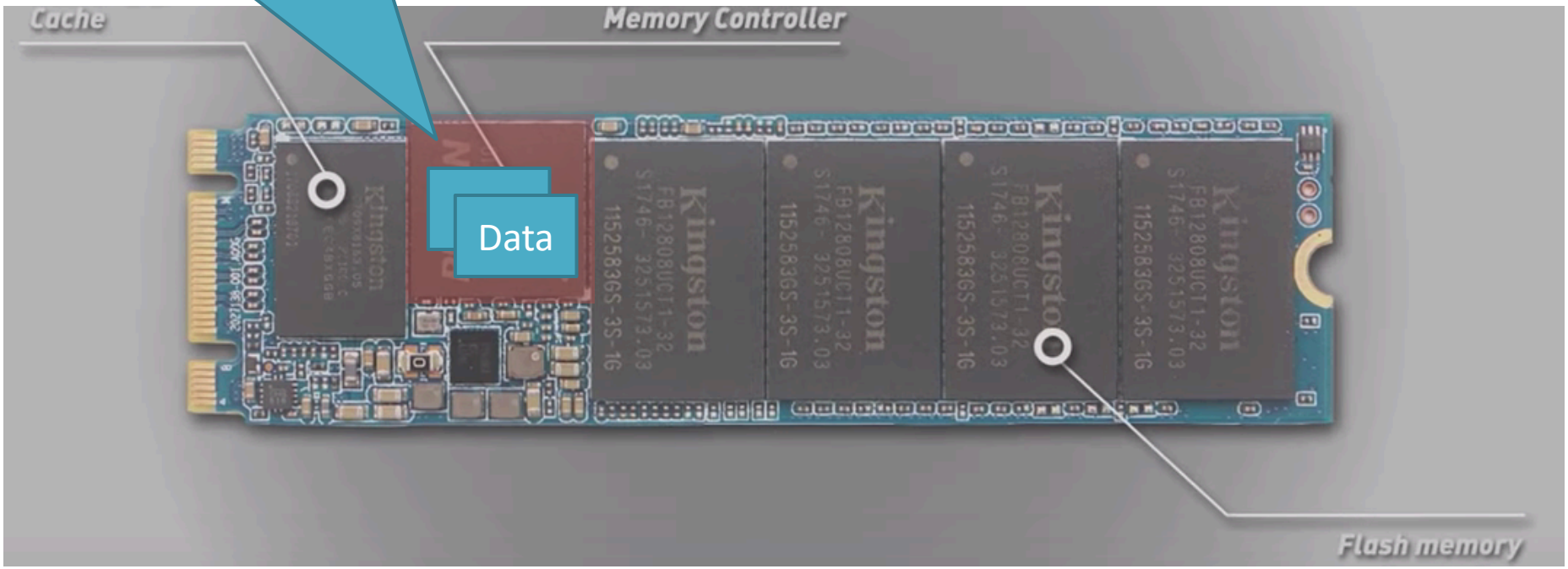


SSD structure

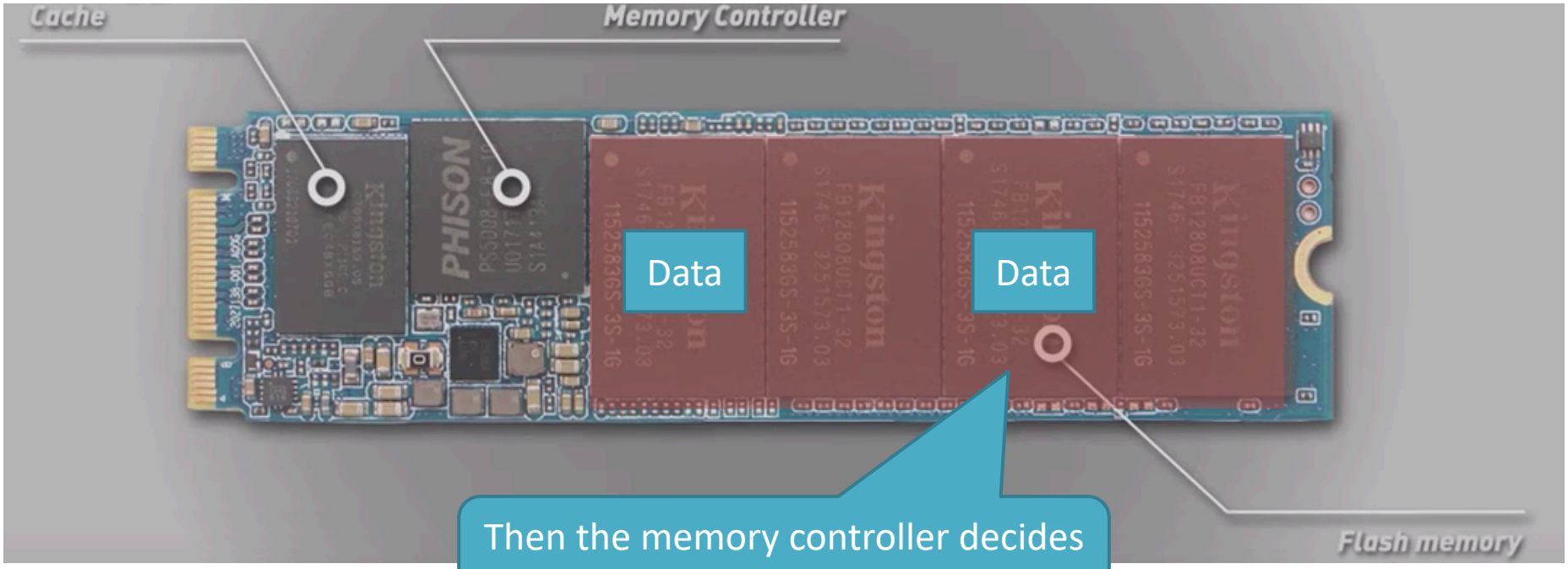


SSD structure

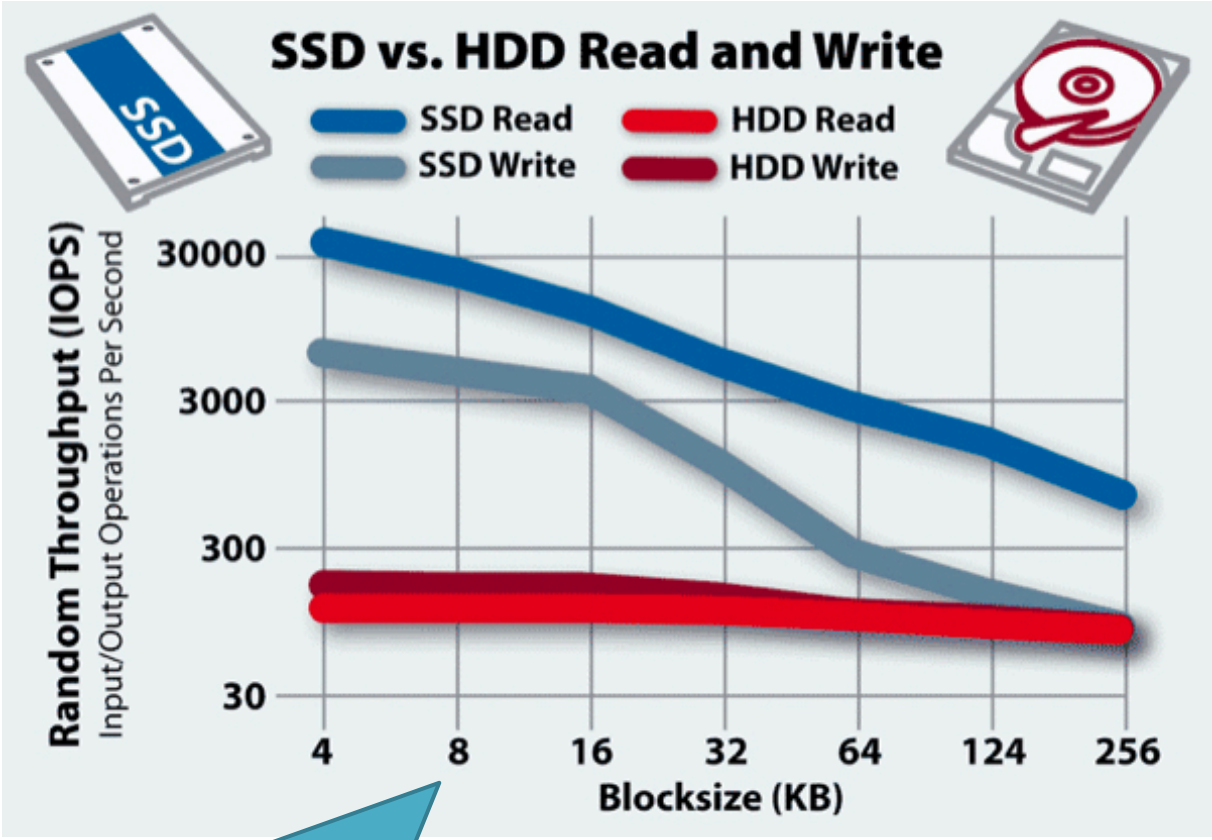
When read files, the data is first put into memory controller



SSD structure



Performance comparison

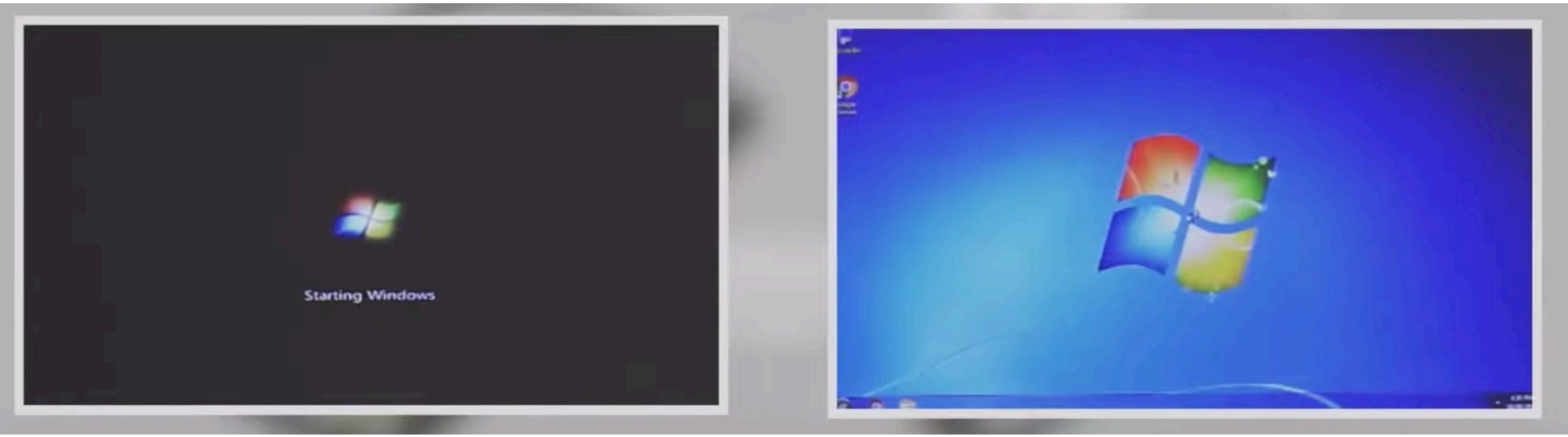


SSD is more than 10x than HDD

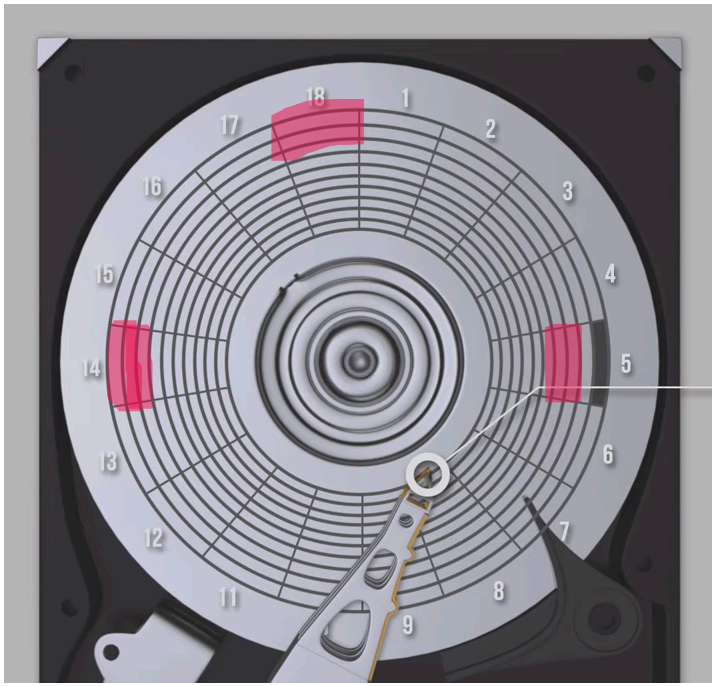
Performance comparison

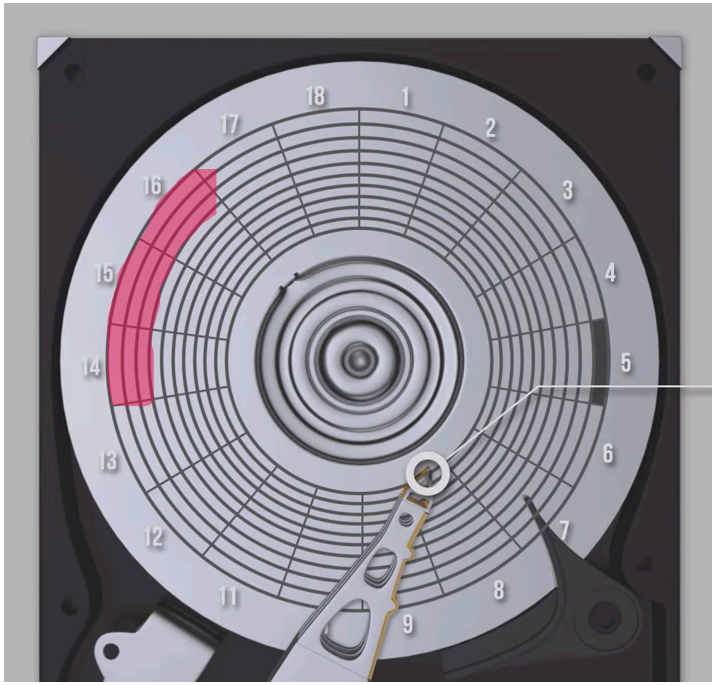
Windows Boot on HDD

Windows Boot on SSD



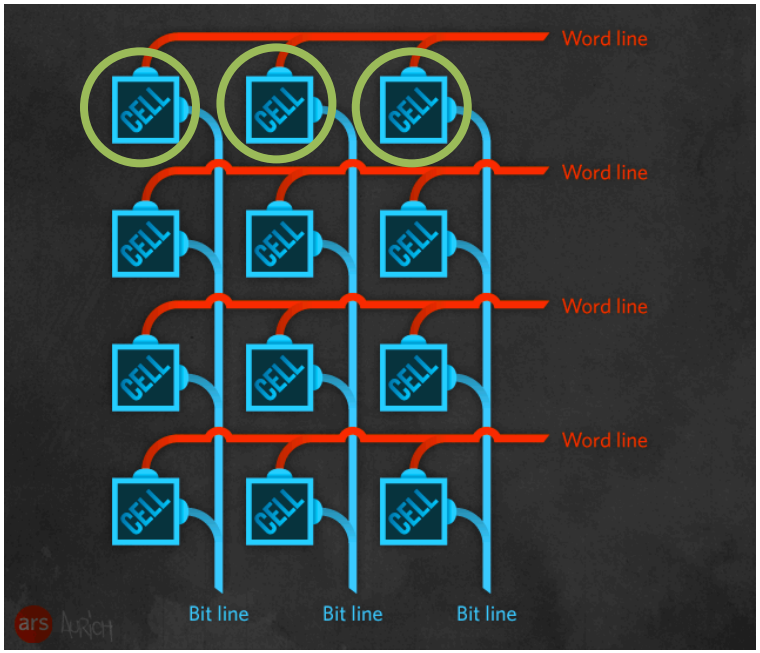
Sequential vs. Random on SSD/HDD

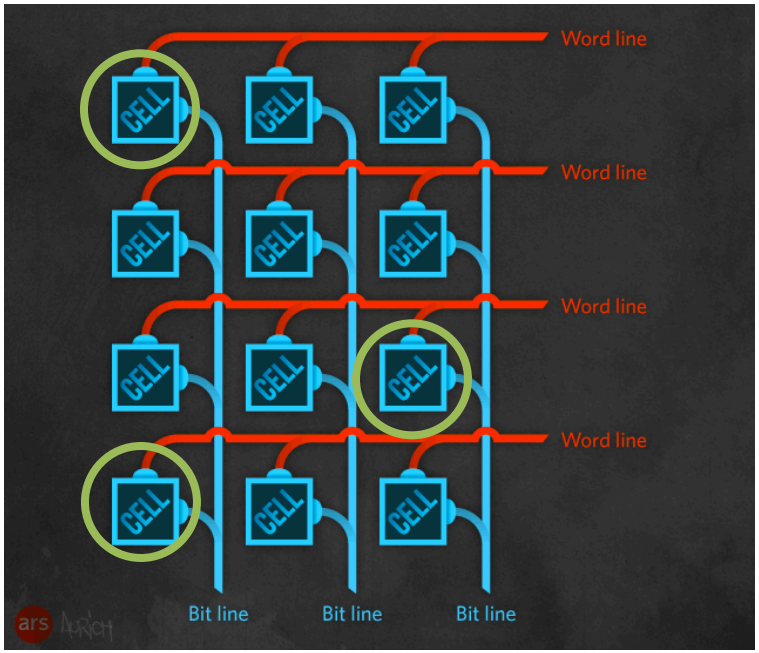




Due to mechanical structure, sequential operation is much faster than random operation on HDD

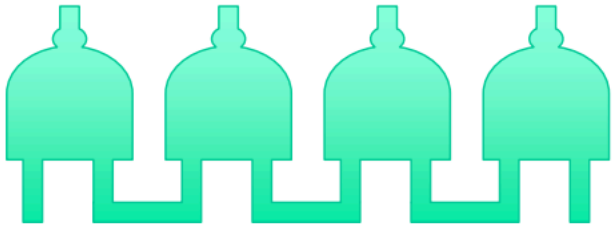
Sequential vs. Random on SSD/HDD



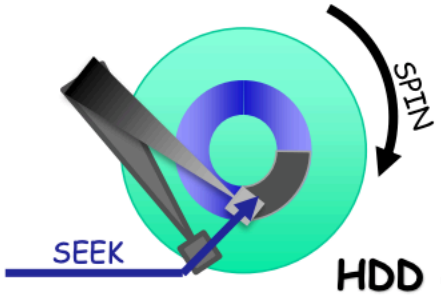


SSD is in semiconductor flash, accessing different cells takes almost equal time

Sequential vs. Random on SSD/HDD



SSD OR FLASH



HDD OR DISK DRIVE

	WRITE	READ
EVERYTHING IS RANDOM IO FOR FLASH	ERASE + WRITE	READ

	WRITE	READ
SEQUENTIAL	WRITE	READ
RANDOM	SEEK/SPIN + WRITE	SEEK/SPIN + READ

SLOWER PERFORMANCE

Everything is random on SSD

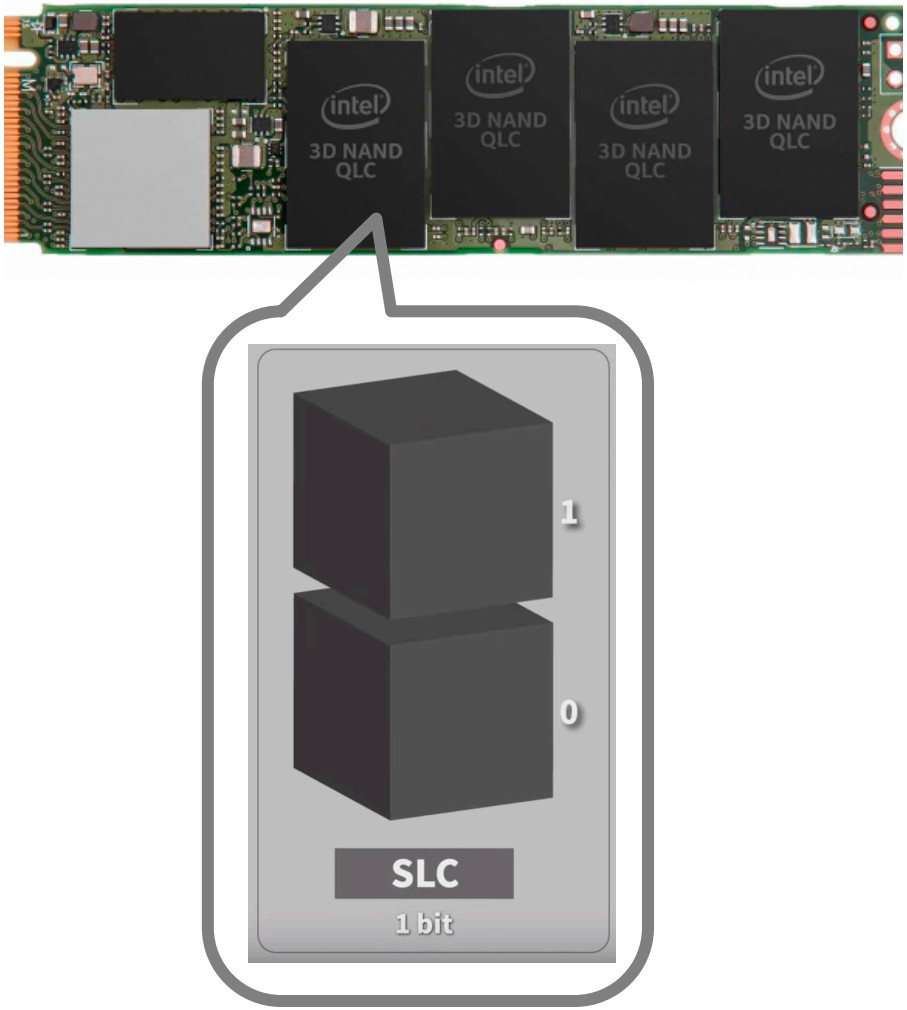
Sequential operation is much faster than random operation on HDD

SSD Storage is Emerging

- Solid-state drives (SSDs) are widely adopted in data centers
  - Examples: EMC XtremIO Array, NetApp Sandisk, Micron P420m
- Pros of SSDs:
  - High I/O throughput
  - low power
  - high reliability
- Cons of SSDs:
  - Wear-out, much shorter lifespan
  - Expensive

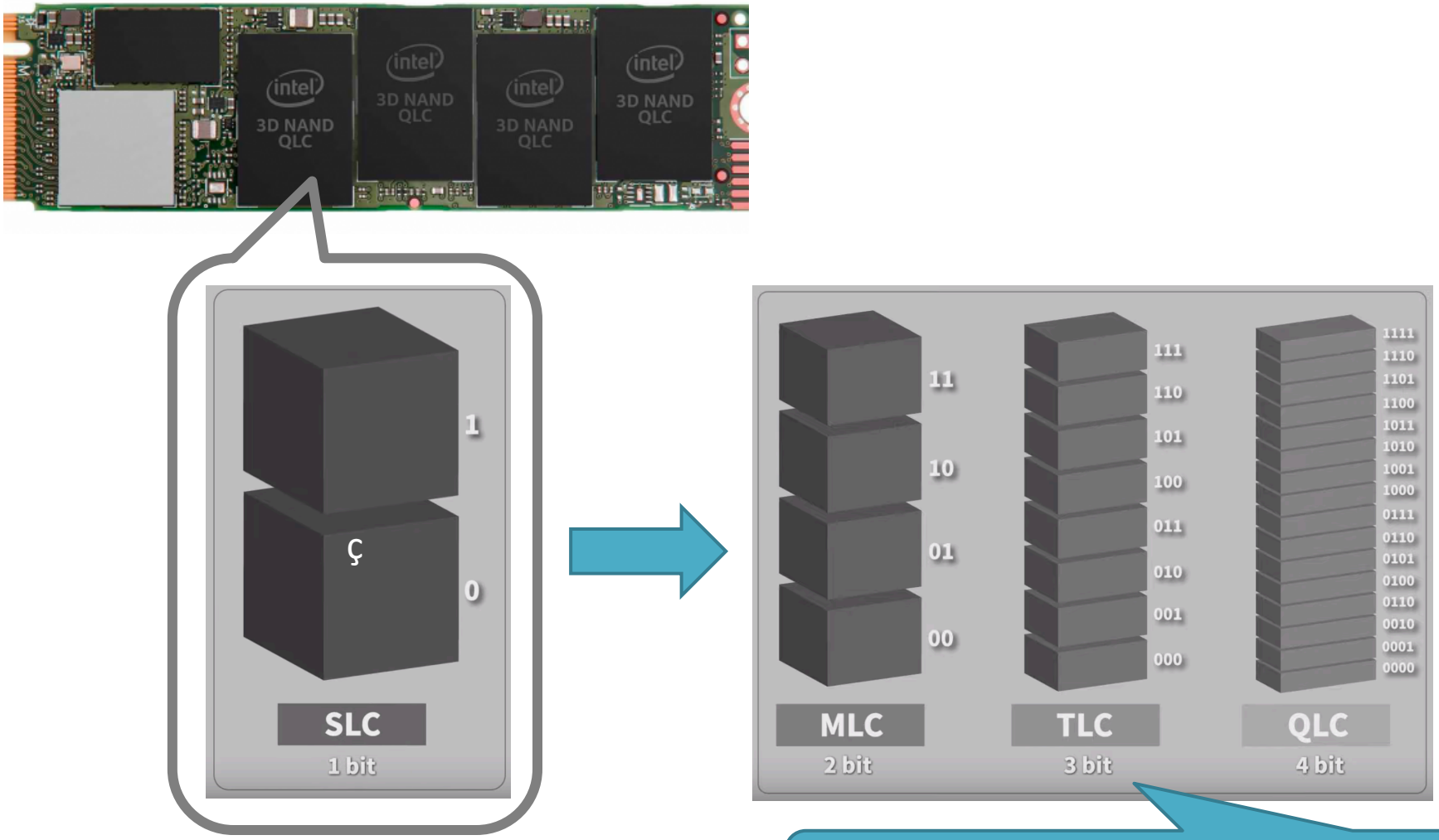
EMC XtremIO
[Source: http://www.crn.com]

Unit in early SSD



- The basic storage unit in SSD is called single level cell (SLC)
- One SLC can store 1 bit data (0 or 1)

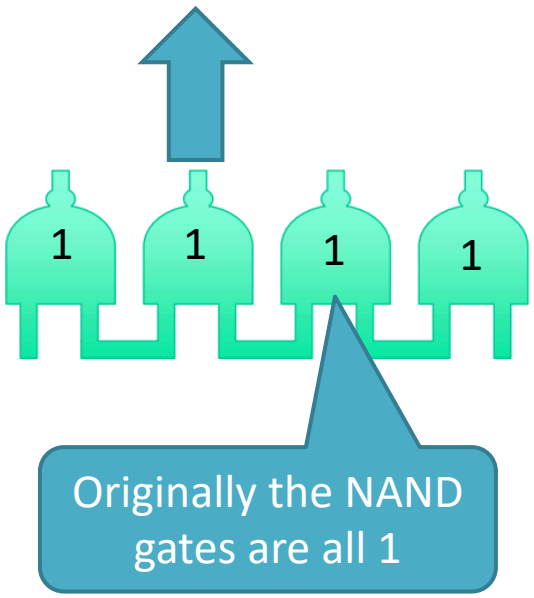
Unit in modern SSD



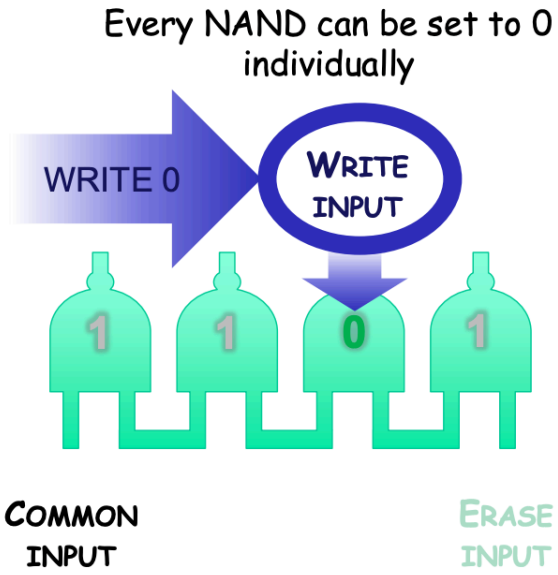
Single unit in SSD can store more data, making SSD price more affordable

Read, Write and Erase on SSD

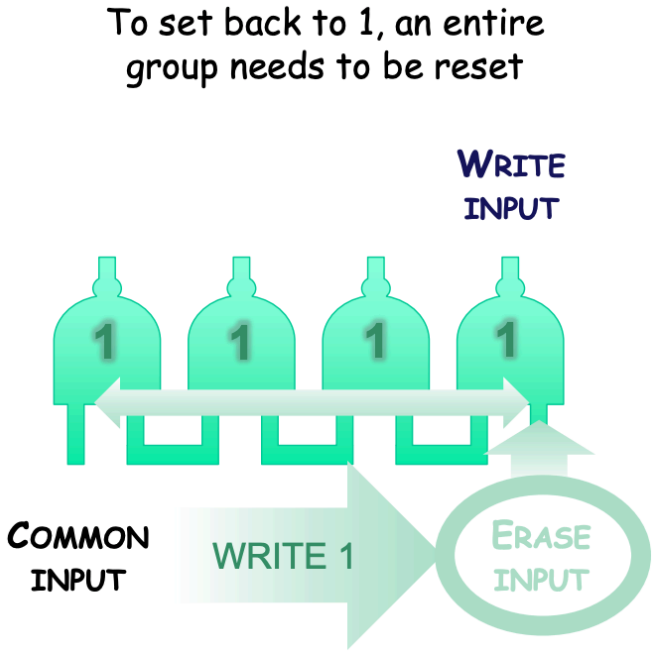
Read



Write

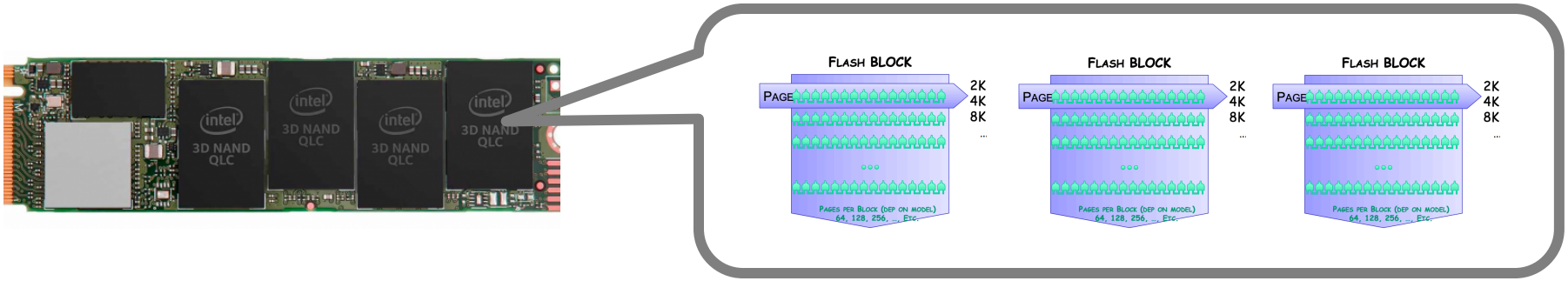


Erase



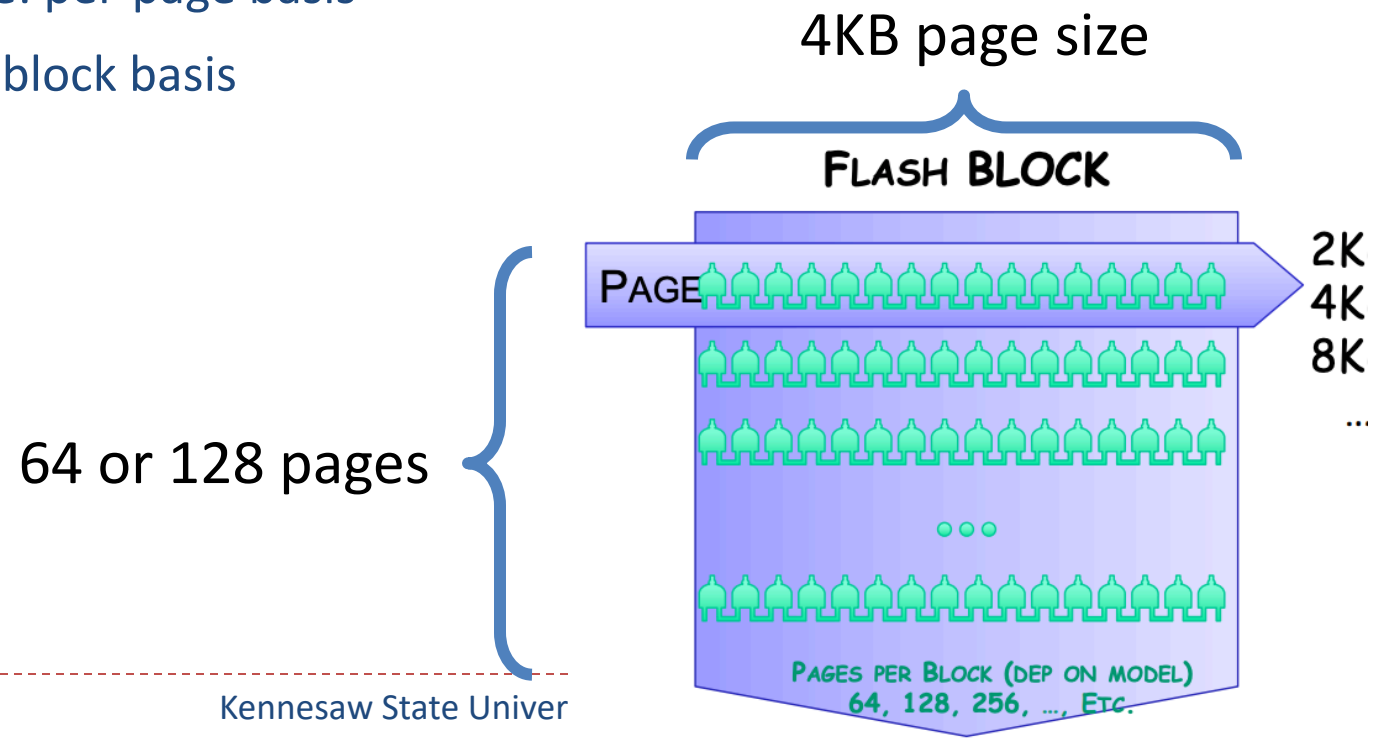
How SSDs Work?

- Organized into blocks
- Each block has 64 or 128 pages of size (e.g., 4KB each)
- Three basic operations: read, write, erase
  - Read, write: per-page basis
  - Erase: per-block basis



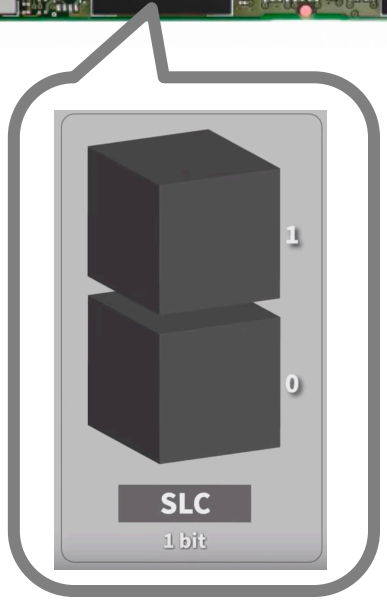
How SSDs Work?

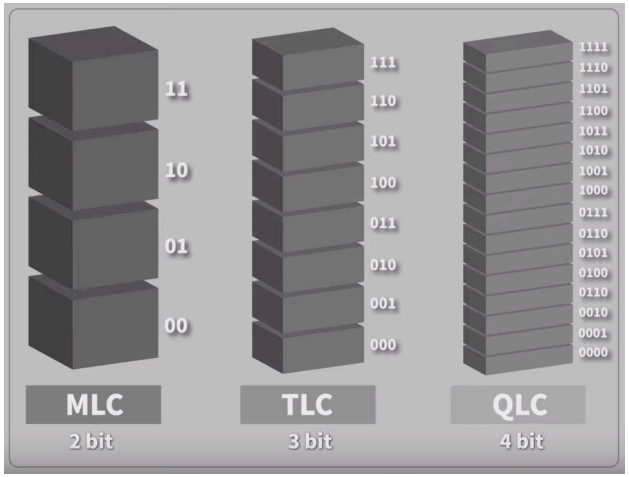
- Organized into blocks
- Each block has 64 or 128 pages of size (e.g., 4KB each)
- Three basic operations: read, write, erase
  - Read, write: per-page basis
  - Erase: per-block basis



Number of Erasing for Units

The lifespan of SSD is negatively correlated with the size





100K times

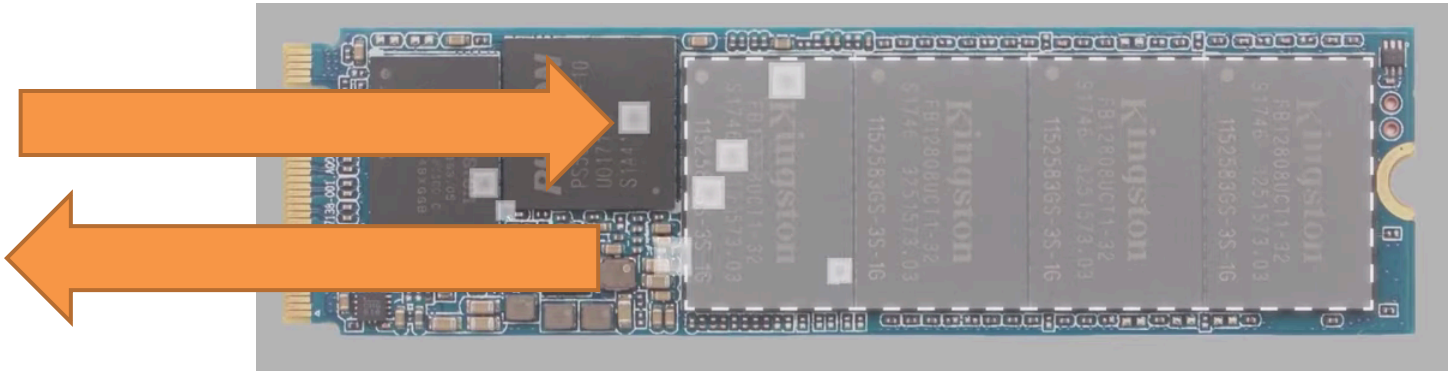
5K times

2K times

1K times

SSD Lifespan

- The lifespan of SSD is around 1k-100k write depending on the SSD materials
- Read on SSD does not shorten the SSD lifespan



SSD Lifespan

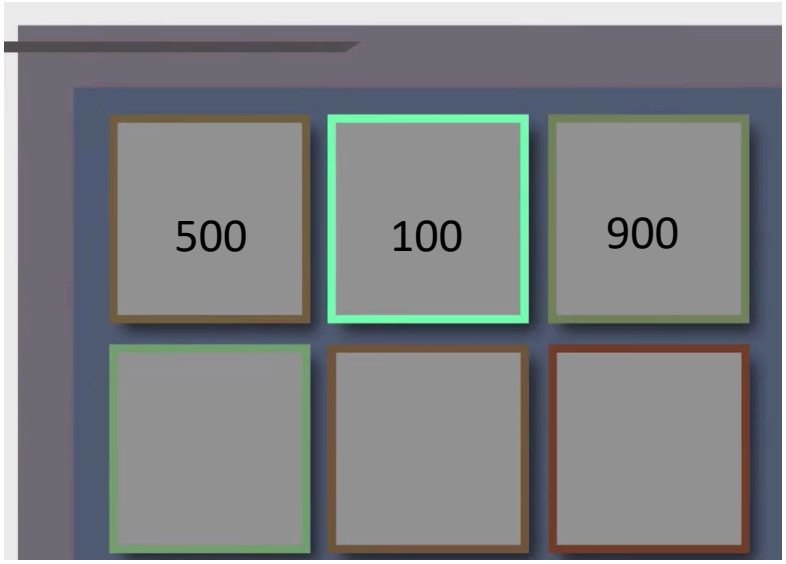
In SSD, some blocks might have more writes than the other blocks. If not handled properly, the blocks might be broken more easily.

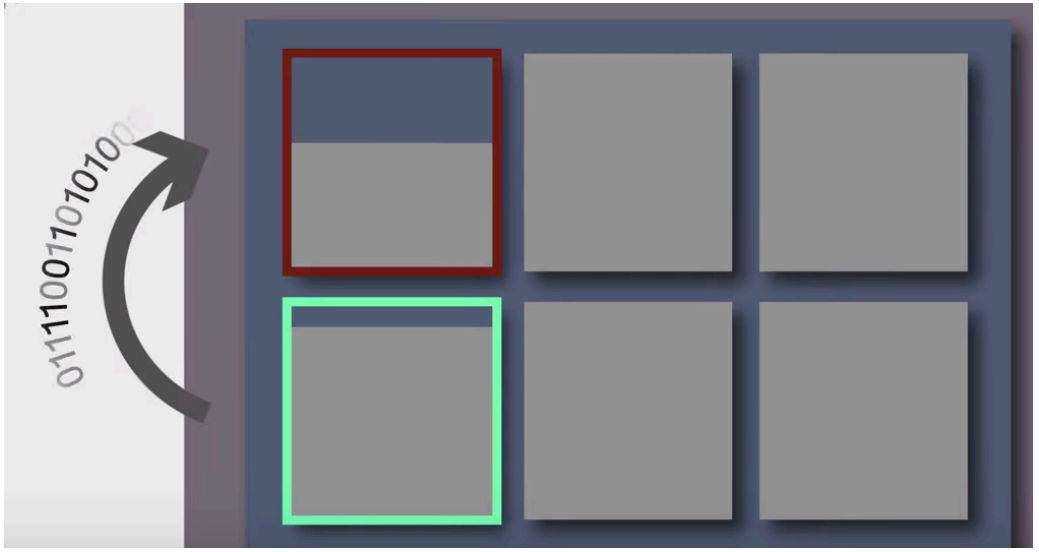
This would make the SSD efficient volume smaller and the entire speed of SSD slower

SSD Lifespan: Dynamic balanced control

The memory controller can select the “younger” blocks for more writes instead of the “older” blocks

The memory controller will move the cold data from the ”younger” blocks to the “older” blocks





HDD vs SSD reliability comparison

HDD

Mechanical structure

SSD

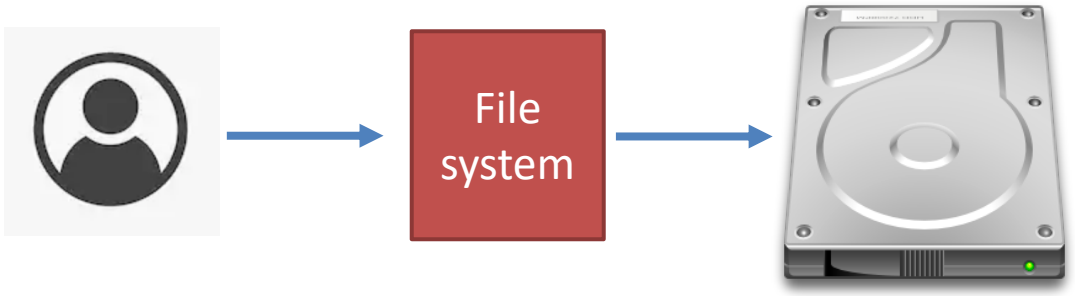
Electronical structure

Outline

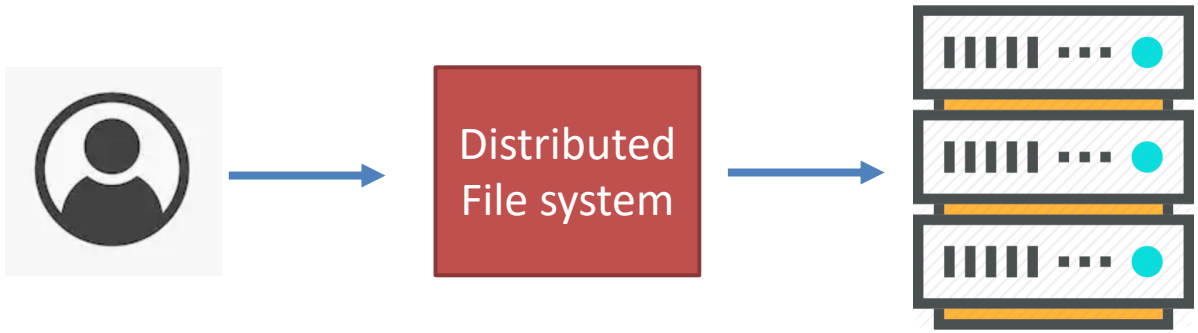
- File system reliability, consistency and performance
- Storage system structure
- Hard disk drive(HDD) vs SSD
- Case study: Distributed and Parallel File Systems

Distributed and Parallel File Systems

File system on single machine



FS manages how and where data is stored, managed and accessed on the raw storage.

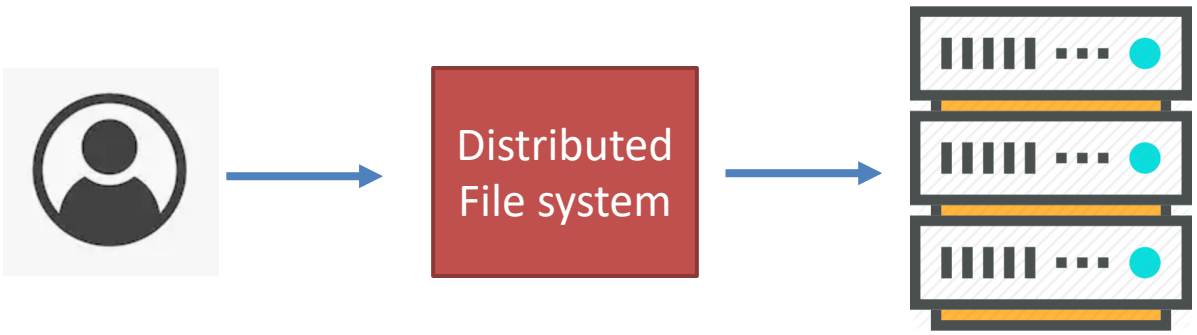


DFS/PFS provide similar abstractions of data on *multiple* machines

Distributed and Parallel File Systems

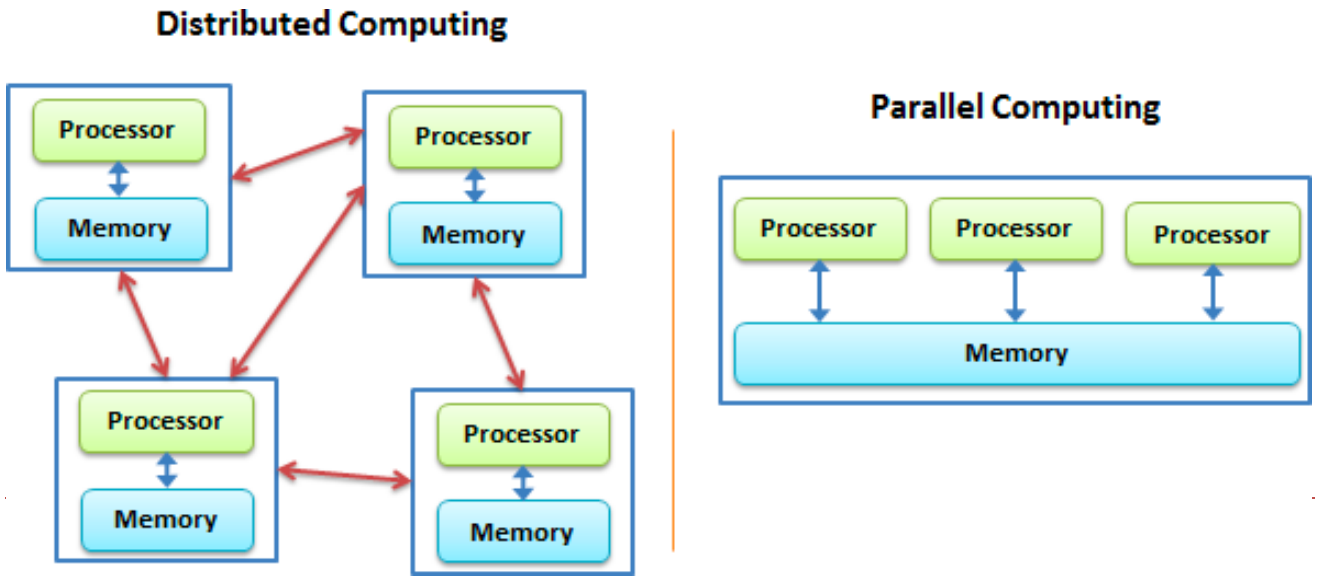
- Namespace: path name → machine ID: disk block address

- Management: locate and manage data on one disk → placement of files on which machines, efficiency, load balance, reliability, etc.



Distributed File Systems vs. Parallel File Systems

- Design objectives
  - Fault-tolerance vs. Concurrent performance
- Workload
  - Loosely coupled, distributed apps vs. coordinated HPC apps for scientific research
- The boundary is blurring



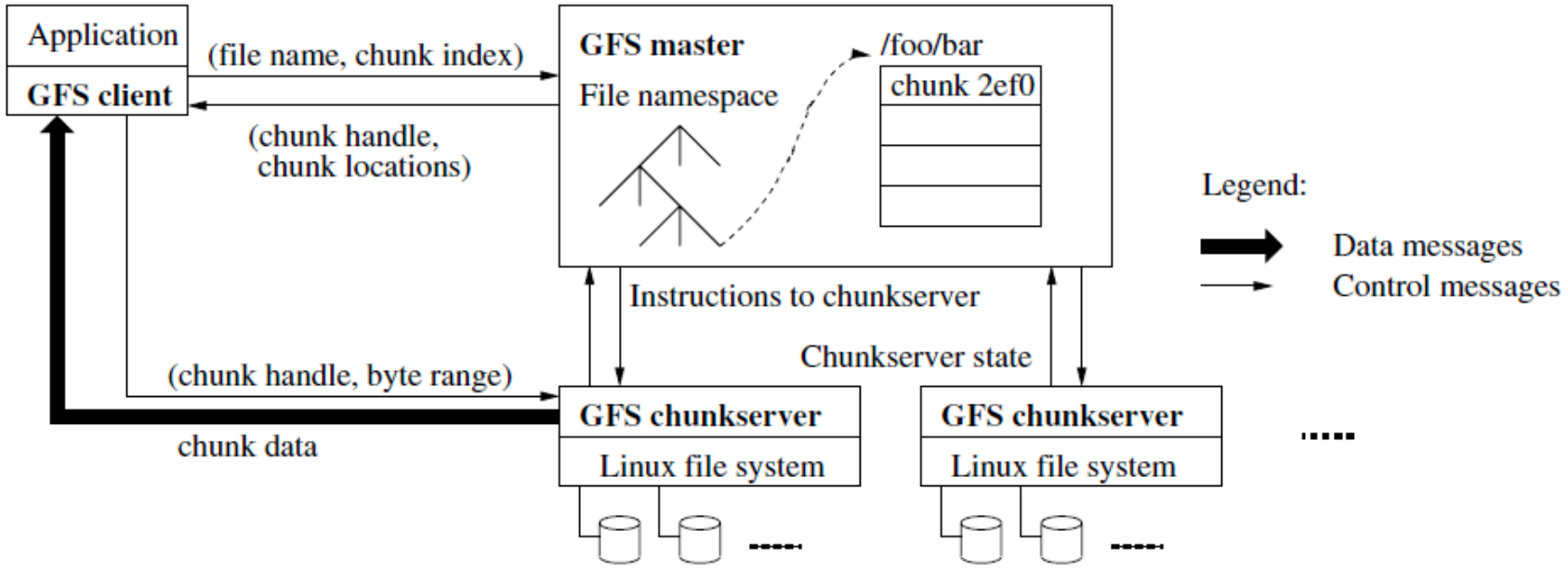
Examples

- Distributed File Systems
  - NFS, GFS (Google File System), HDFS (Hadoop Distributed File System), GlusterFS
- Parallel File Systems
  - PVFS (Parallel Virtual File System), Lustre, OCFS2, GPFS

A Case Study: Google File System (GFS)

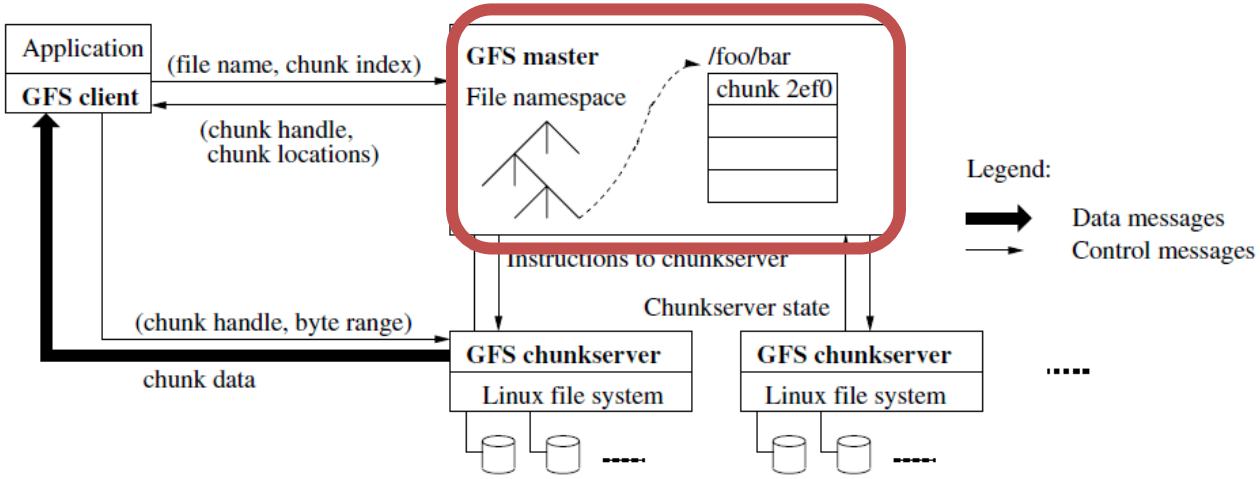
- Google needed a good distributed file system
  - Redundant storage of massive amounts of data on cheap and unreliable computers
- Why not use an existing file system?
  - Google's problems are different from anyone else's
  - Different workload and design priorities
  - GFS is designed for Google apps and workloads
  - Google apps are designed for GFS

GFS Architecture



Each chunk is identified by an immutable and globally unique 64 bit chunk handle assigned by the master at the time of chunk creation.

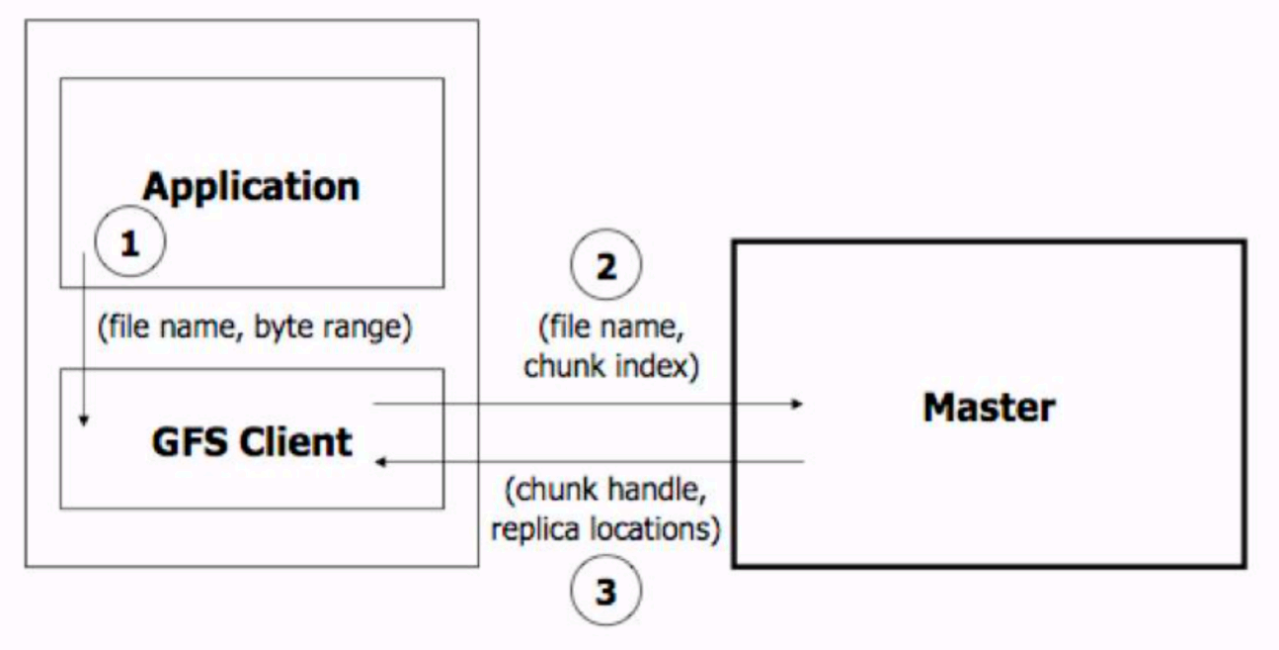
Master's Responsibilities



- Metadata storage
- Namespace management/locking
- Periodic communication with chunkservers
  - give instructions, collect state, track cluster health
- Chunk creation, re-replication, rebalancing
  - balance space utilization and access speed
  - re-replicate data if redundancy falls below threshold
  - rebalance data to smooth out storage and request load
- More …

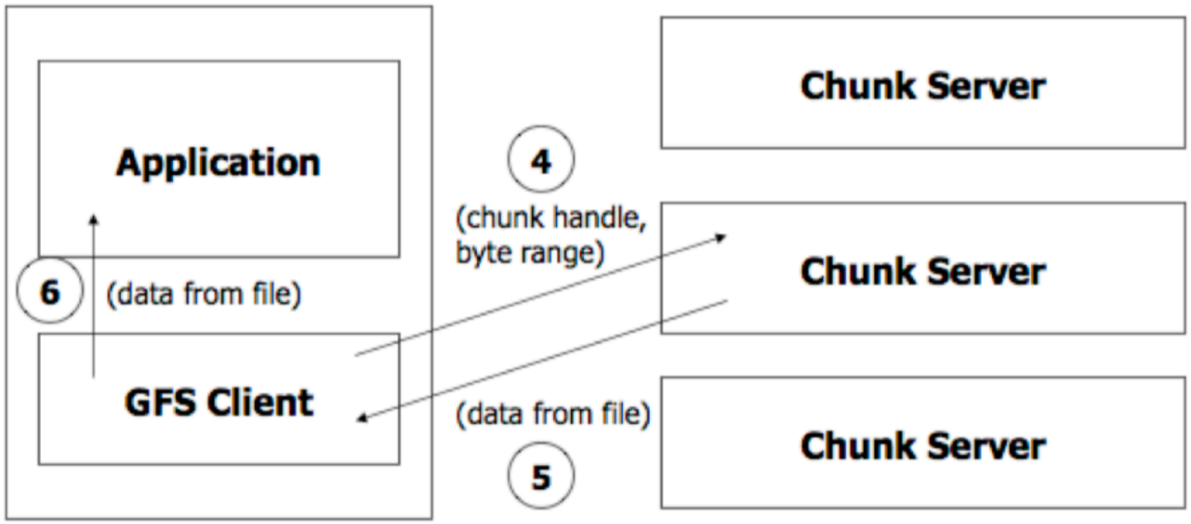
Read on GFS

1. Application originates the read request
2. GFS client translates request and sends it to master
3. Master responds with chunk handle and replica locations



Read on GFS

4. Client picks a location and sends the request
5. Chunkserver sends requested data to the client
6. Client forwards the data to the application

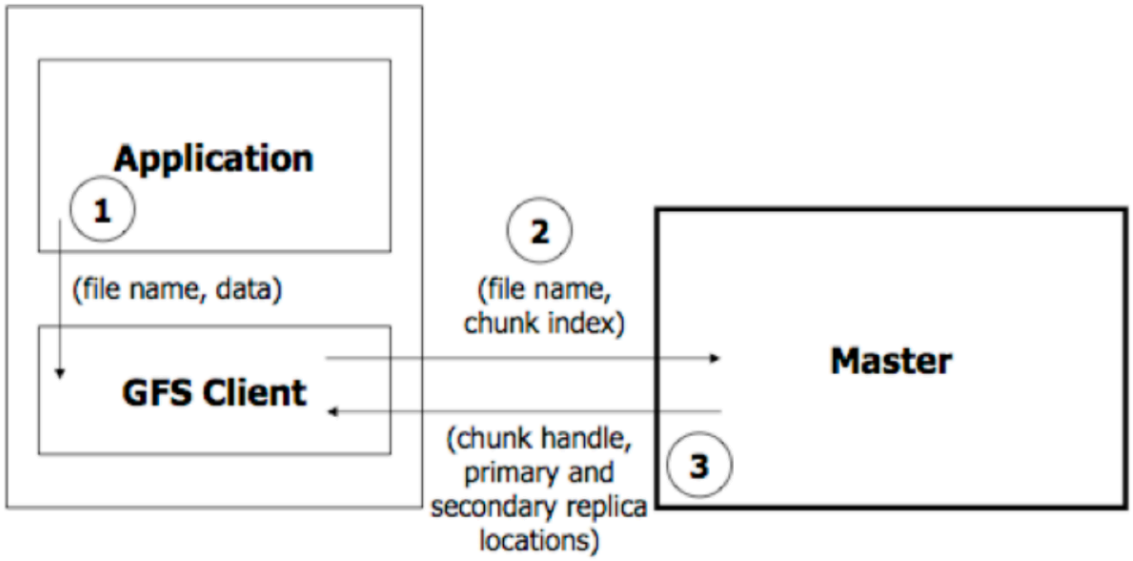


Write on GFS

1. Application originates the request

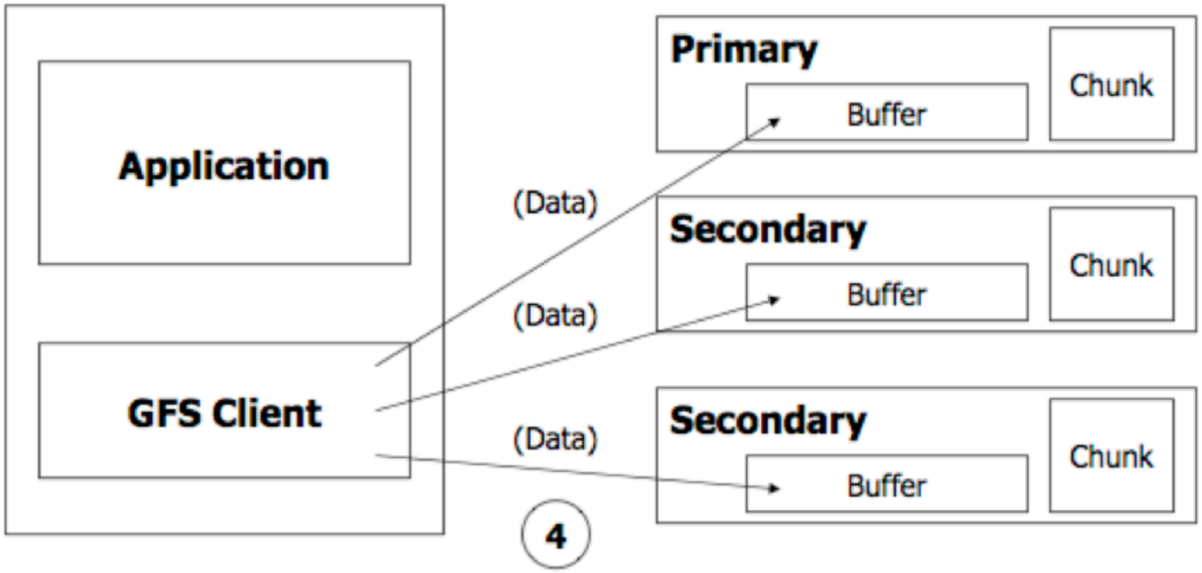
2. GFS client translates request and sends it to the master

3. Master responds with chunk handle and replica locations, which are cached by the client.



Write on GFS

4. Client pushes write data to all locations. Data is stored in chunkserver's internal buffers

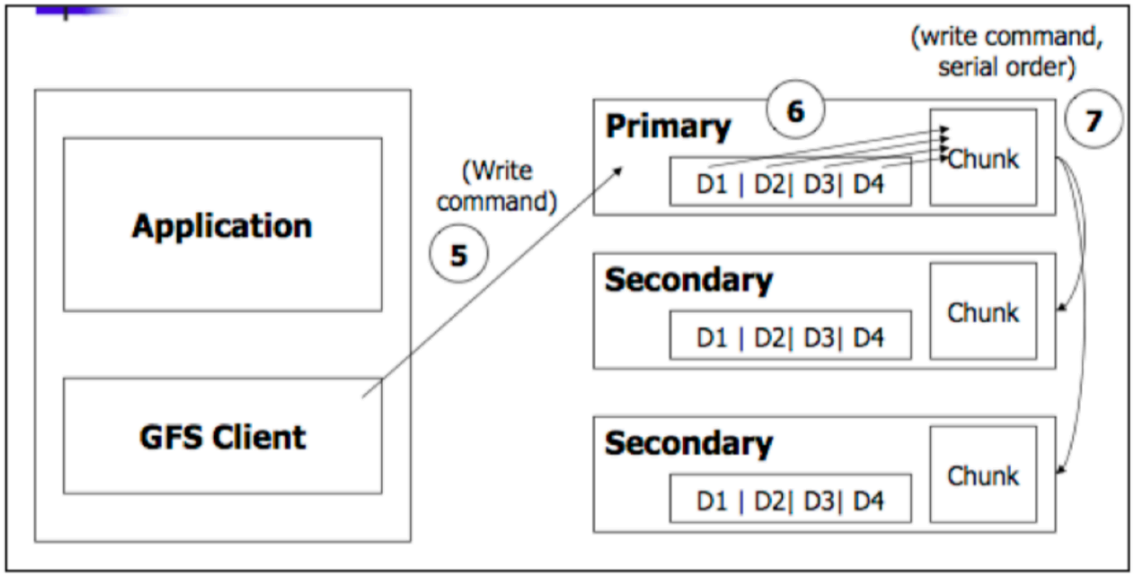


Write on GFS

5. Client sends write command to primary

6. Primary determines serial order for data instances in its buffer and writes the instances in that order to the chunk

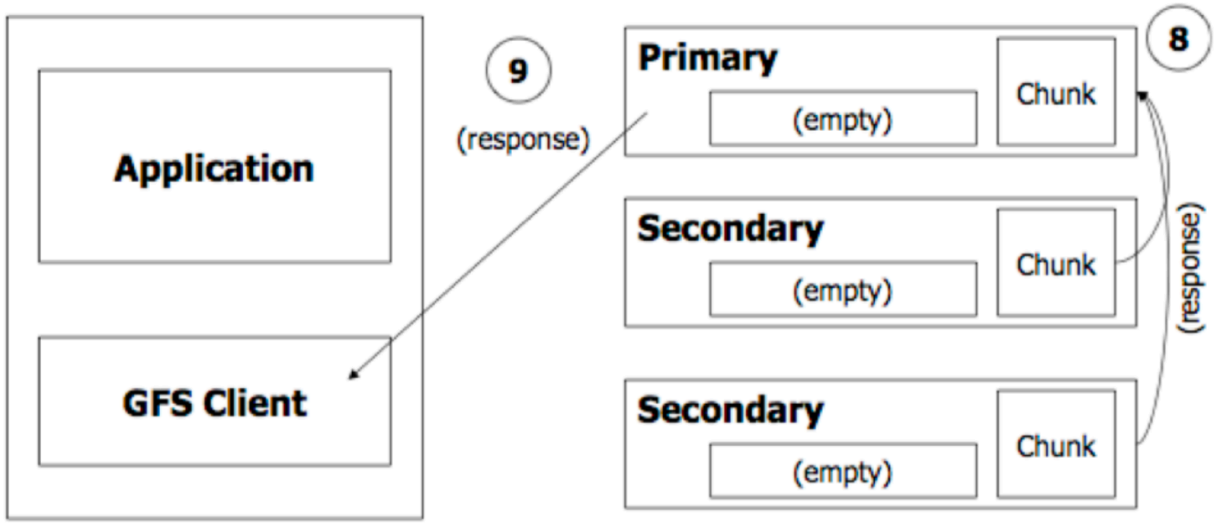
7. Primary sends the serial order to the secondaries and tells them to perform the write



Write on GFS

8. Secondaries respond back to primary

9. Primary responds back to the client



More about GFS

- http://static.googleusercontent.com/media/research.google.com/en/us/archive/gfs-sosp2003.pdf

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google*

ABSTRACT

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore rad-

1. INTRODUCTION

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

More about Distributed and Parallel File Systems

- Finding a needle in Haystack: Facebook's photo storage
  - https://www.usenix.org/legacy/event/osdi10/tech/full_papers/Beaver.pdf
- Flat Datacenter Storage
  - https://www.usenix.org/system/files/conference/osdi12/osdi12-final-75.pdf
- Ceph: A Scalable, High-Performance Distributed File System
  - https://www.ssrc.ucsc.edu/Papers/weil-osdi06.pdf

Conclusion

- File system reliability, consistency and performance
- Storage system structure
- Hard disk drive(HDD) vs SSD
- Case study: Distributed and Parallel File Systems