CS162
Operating Systems and Systems Programming
Lecture 19

General IO (Continued) & File Systems

Professor Natacha Crooks.
Special Guest: Tux
https://cs162.org/

Slides based on prior slide decks from David Culler, Ion Stoica, John Kubiatowicz, Alison Norman and Lorenzo Alvisi

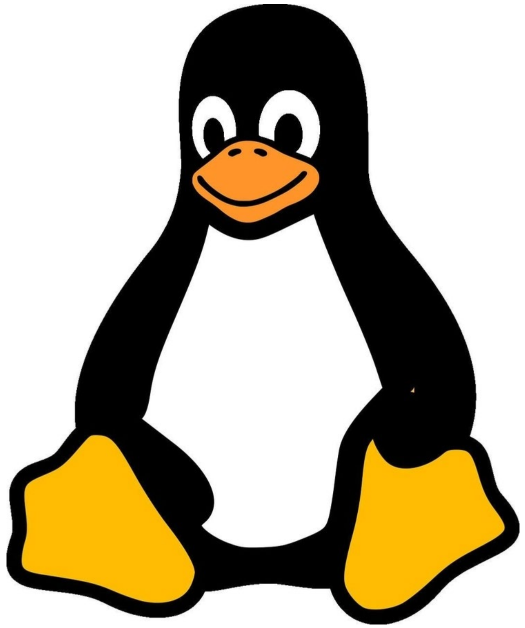
A History of Tux

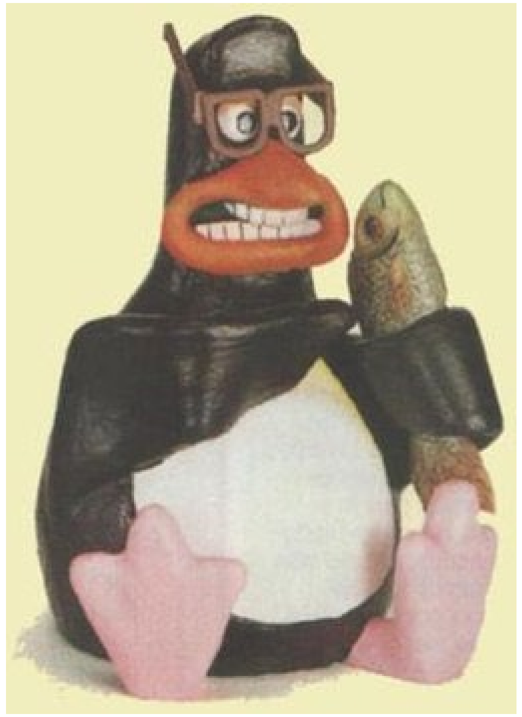
Tux, the friendly penguin has been Linux logo since 1996

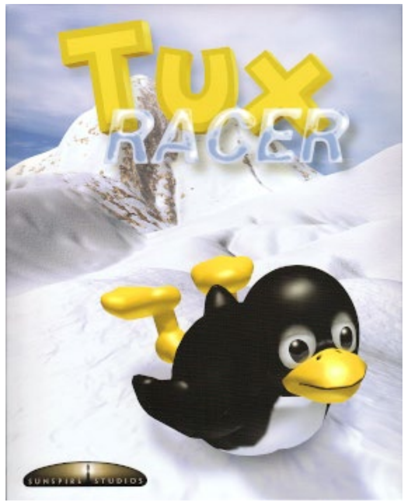
Why? Linus Torvalds … just really liked penguins despite being bitten by a penguin in Australia

By 2007, the zoo in Canberra where Torvalds was first nibbled by a penguin had erected a sign commemorating the episode, mentioning "It is our belief that the original Tux is still housed in this enclosure."

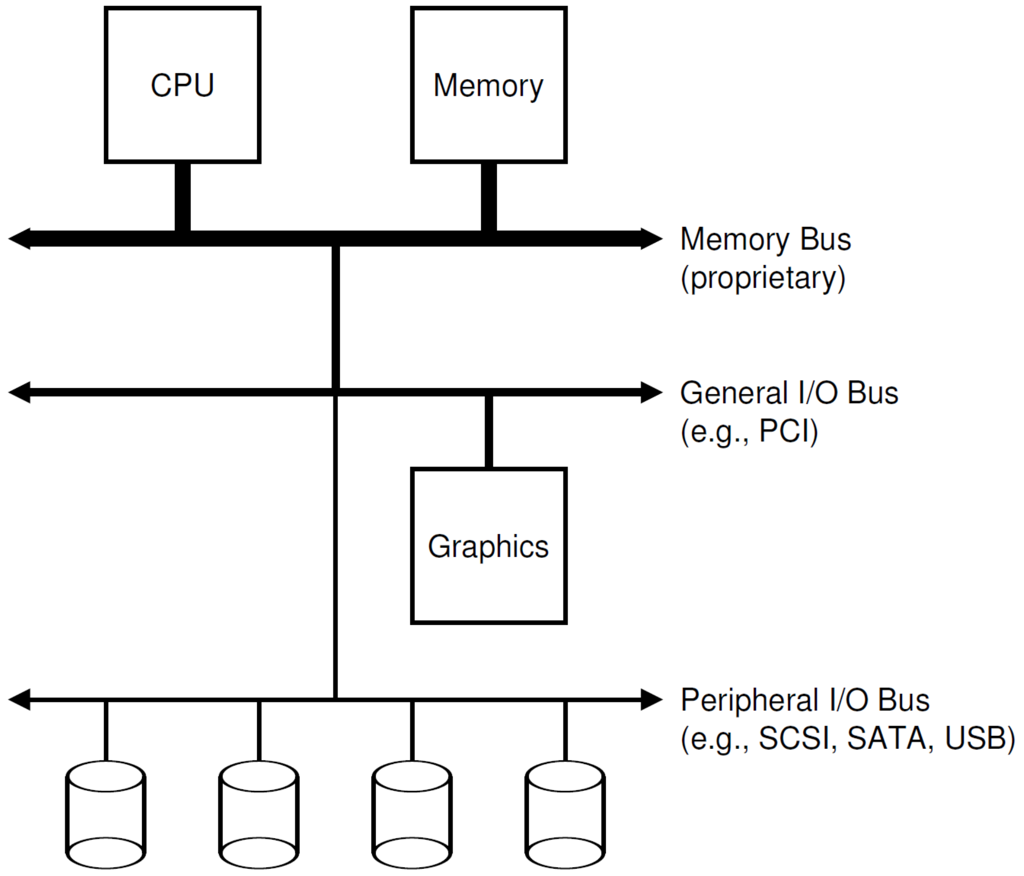
A History of Tux







Recall : Simplified IO architecture

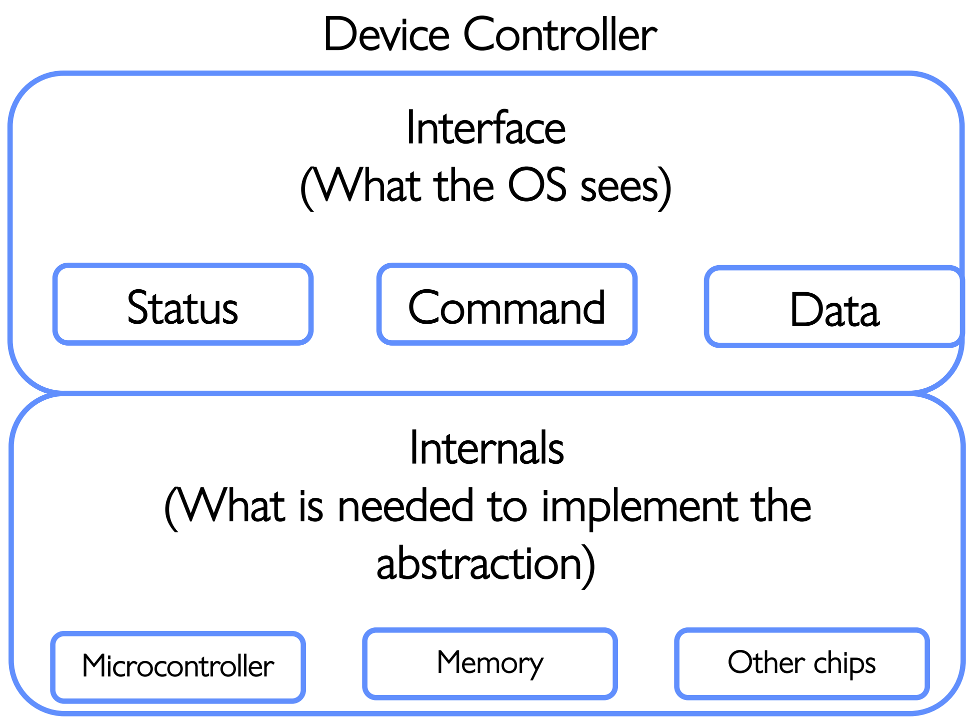


Follows a hierarchical structure because of cost:

the faster the bus, the more expensive

Recall: How does processor talk to devices?

Remember, it's all about abstractions!



Hardware interface device presents to OS

Hardware interface device presents to OS

Recall: Device Drivers

Device-specific code in the kernel that interacts directly with the device hardware

Supports a standard, internal interface
Special device-specific configuration supported with `ioctl()`

Top half: accessed in call path from system calls.
Implements a set of standard, cross-device calls

Bottom half: run as interrupt routine
Gets input or transfers next block of output
May wake sleeping threads if I/O now complete

Your body is 90% water, the OS is 70% device-drivers

Ways of Measuring Performance

Latency - time to complete a task
Measured in units of time (s, ms, us, …, hours, years)

Throughput or *Bandwidth* – rate at which tasks are performed
Measured in units of things per unit time (ops/s, GFLOP/s)

Start up or Overhead – time to initiate an operation

Most I/O operations are roughly linear in *b* bytes
–Latency(b) = Overhead + b/TransferCapacity

Storage Devices

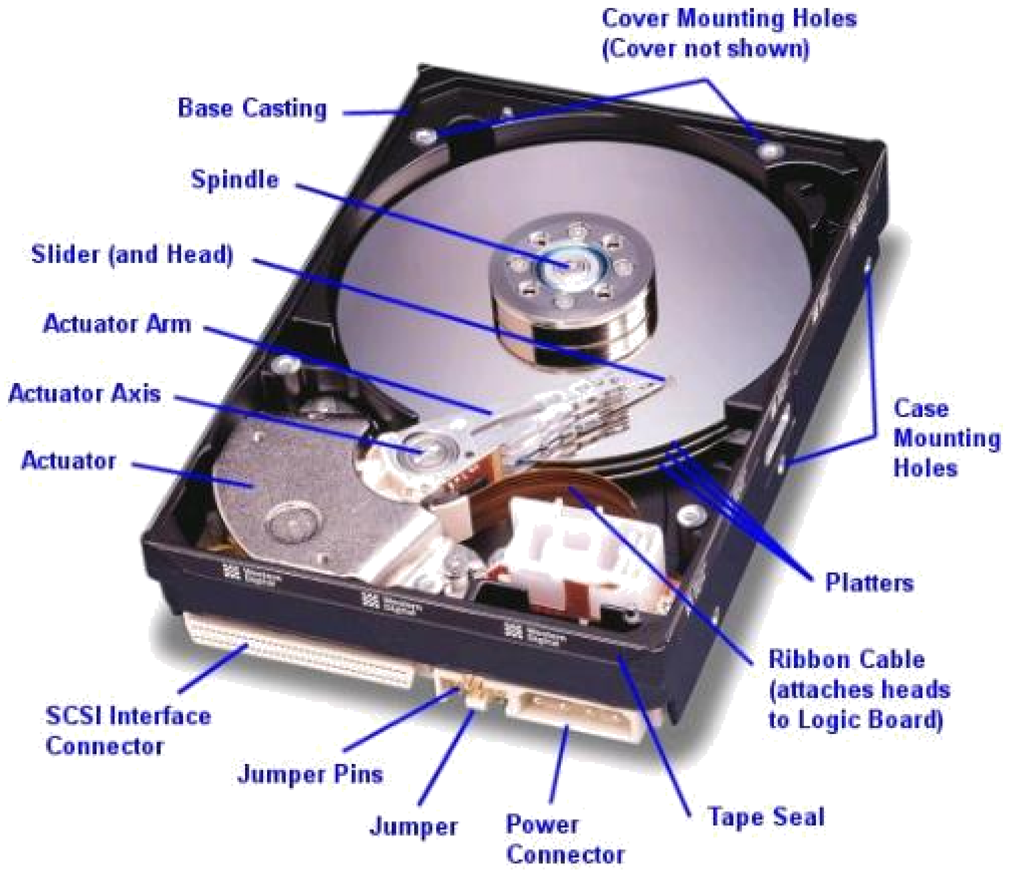
Magnetic disks

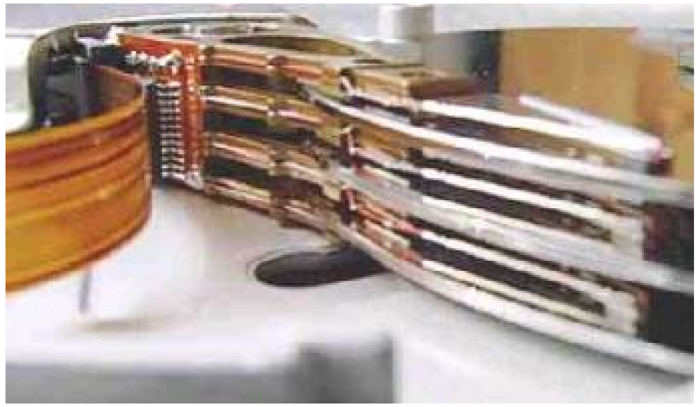
- Storage that rarely becomes corrupted
- Large capacity at low cost
- Block level random access (except for **SMR** – later!)
- Slow performance for random access
- Better performance for sequential access

Flash memory

- Storage that rarely becomes corrupted
- Capacity at intermediate cost (5-20x disk)
- Block level random access
- Good performance for reads; worse for random writes
- Wear patterns issue

Hard Disk Drives (HDDs)





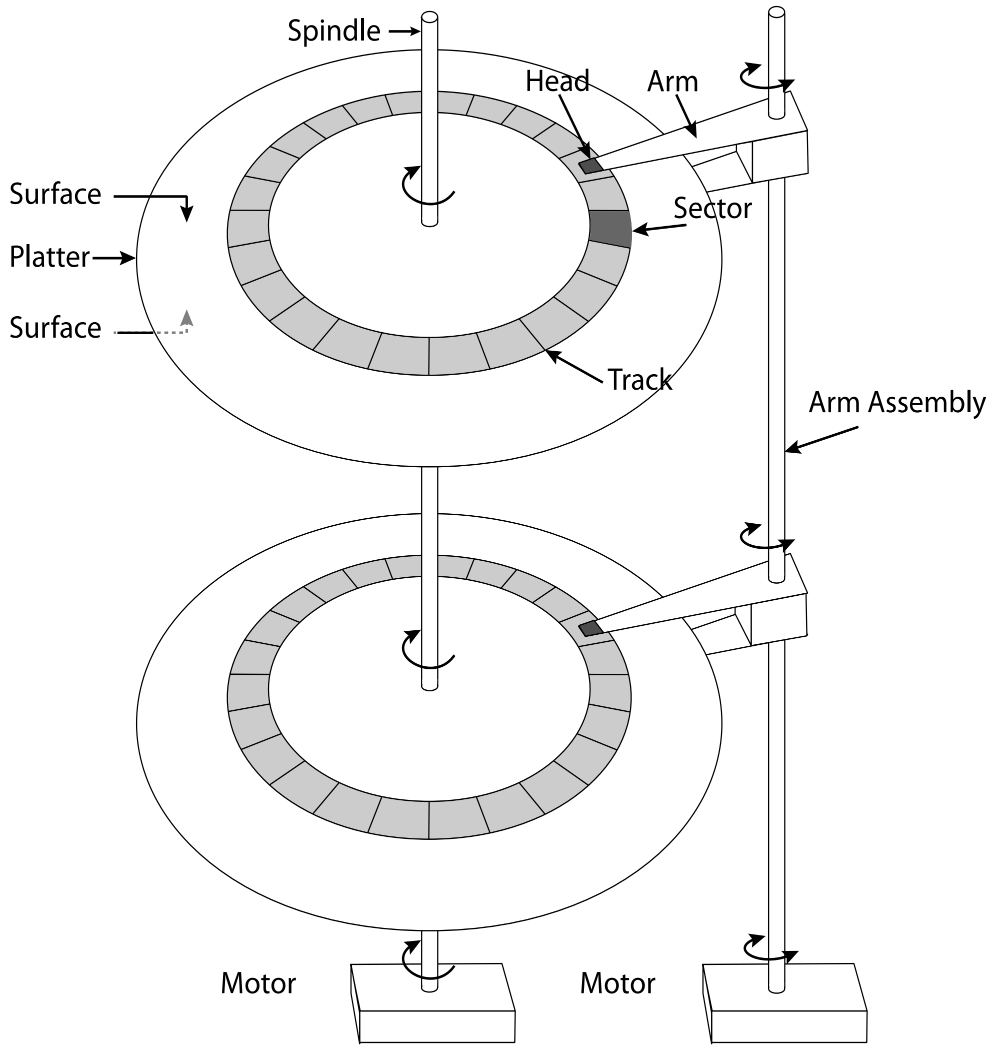
Read/Write Head Side View

IBM Personal Computer 1986

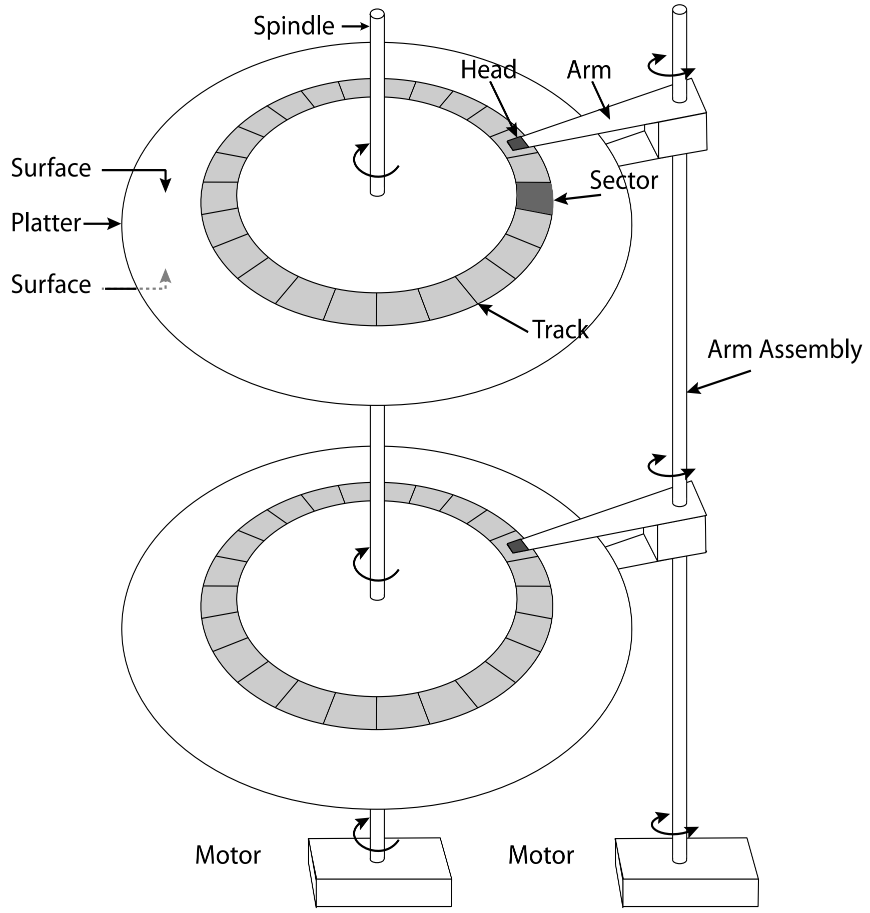
30MB Hard Disk for 500 dollars

The Amazing Magnetic Disk

Store data magnetically on thin metallic film bonded to rotating disk of glass, ceramic, or aluminum



The Amazing Magnetic Disk

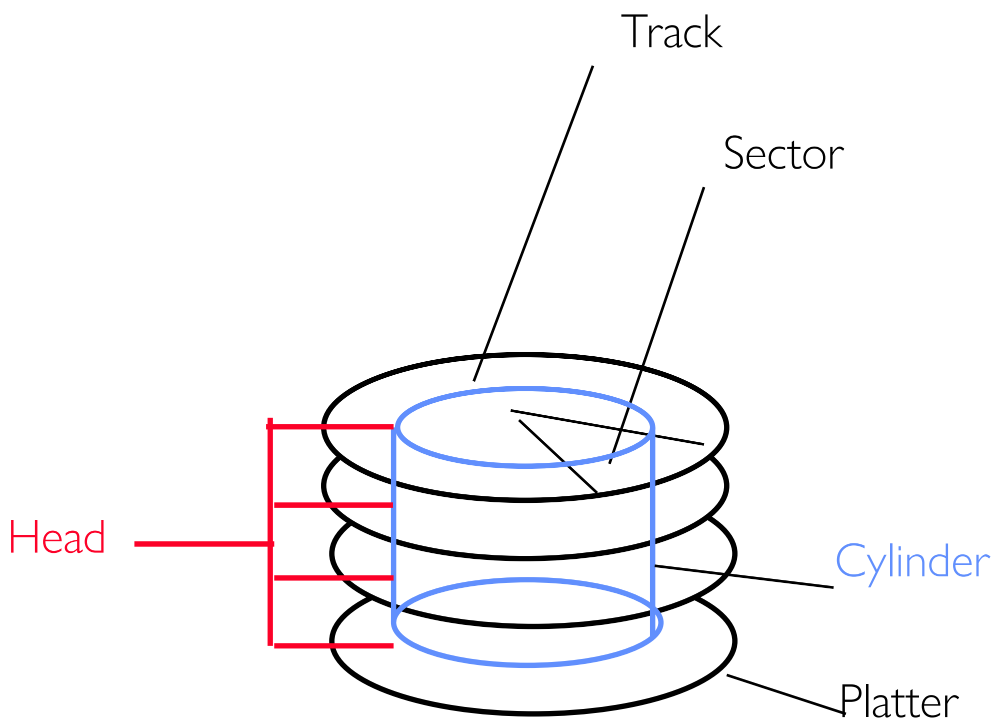


Track: concentric circle on surface

Sectors: slice of a track
Smallest addressable unit
Are units of transfers

Cylinder all the tracks under the head at a given point on all surfaces

The Amazing Magnetic Disk



Track lengths vary across disk: outside tracks have more sectors per track, higher bandwidth

Disk is organized into regions of tracks with the same number of sector/tracks

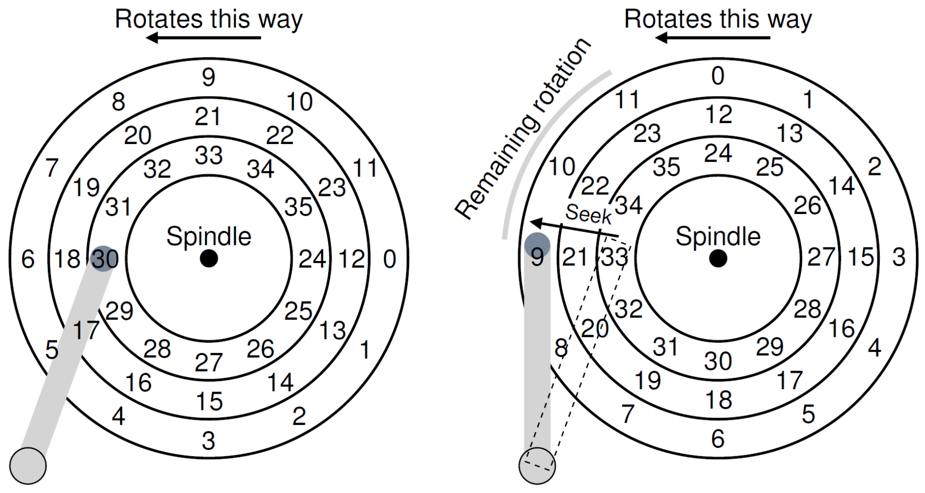
Usually, only outer half of radius is used

Reading/Writing Data

Seek time: position the head/arm over the proper track

Rotational latency: wait for desired sector to rotate under r/w head

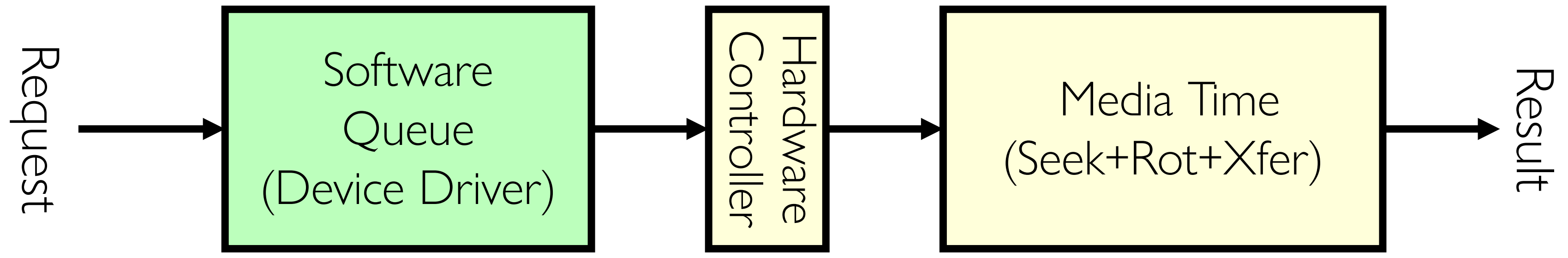
Transfer time: transfer a block of bits (sector) under r/w head



Reading/Writing Data

Request Time =

Queueing Time + Controller Time + Seek + Rotational + Transfer



Typical Numbers for Magnetic Disk

Parameter	Info/Range
Space/Density	Space: 14TB (Seagate), 8 platters, in 3½ inch form factor! Areal Density: ≥ 1 Terabit/square inch! (PMR, Helium, …)
Average Seek Time	Typically 4-6 milliseconds
Average Rotational Latency	Most laptop/desktop disks rotate at 3600-7200 RPM (16-8 ms/rotation). Server disks up to 15,000 RPM. Average latency is halfway around disk so 4-8 milliseconds
Controller Time	Depends on controller hardware
Transfer Time	Typically 50 to 250 MB/s. Depends on: • Transfer size (usually a sector): 512B – 1KB per sector • Rotation speed: 3600 RPM to 15000 RPM • Recording density: bits per inch on a track • Diameter: ranges from 1 in to 5.25 in
Cost	Used to drop by a factor of two every 1.5 years (or faster), now slowing down

Disk Performance Example

Key to using disk effectively (especially for file systems) is to minimize seek and rotational delays

Do access patterns influence how fast can read/write to disk?

Avg seek time of 5ms,

7200RPM $\Rightarrow$
Time for rotation: 60000 (ms/min)/7200(rev/min) ~= 8ms

Transfer rate of 50MByte/s, block size of 4Kbyte $\Rightarrow$
4096 bytes/50×10^6 (bytes/s) = 81.92×10^{-6} sec $\cong$
0.082 ms for 1 sector

Disk Performance Example

Read block from random place on disk (random reads):

- Seek (5ms) + Rot. Delay (4ms) + Transfer (0.082ms) = 9.082ms
- Approx 9ms to fetch/put data: 4096 bytes/9.082×10^{-3} s $\cong$ 451KB/s

Read block from random place in same cylinder:

- Rot. Delay (4ms) + Transfer (0.082ms) = 4.082ms
- Approx 4ms to fetch/put data: 4096 bytes/4.082×10^{-3} s $\cong$ 1.03MB/s

Read next block on same track (sequential reads):

- Transfer (0.082ms): 4096 bytes/0.082×10^{-3} s $\cong$ 50MB/sec

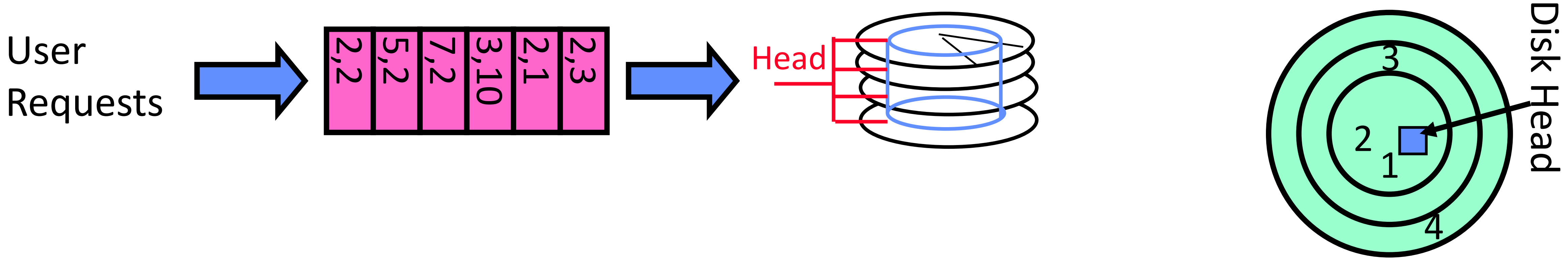
When is Disk Performance Highest?

When there are big sequential reads, or
When there is so much work to do that they can be piggy backed (reordering queues—one moment)

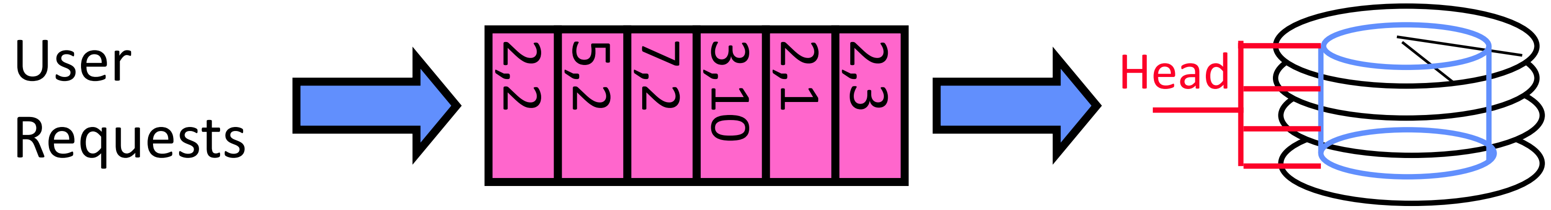
OK to be inefficient when things are mostly idle
Bursts are both a threat and an opportunity
<your idea for optimization goes here>
Waste space for speed?

Disk Scheduling (1/3)

Disk can do only one request at a time; What order do you choose to do queued requests?



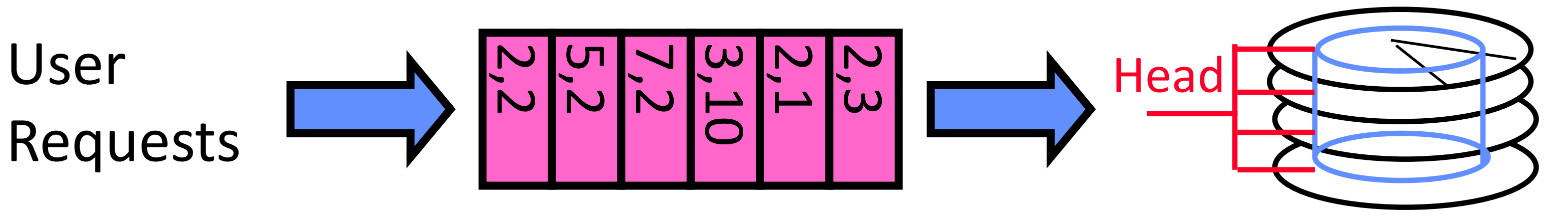
Disk Scheduling (1/3)



FIFO Order
Fair among requesters, but order of arrival may be to random spots on the disk

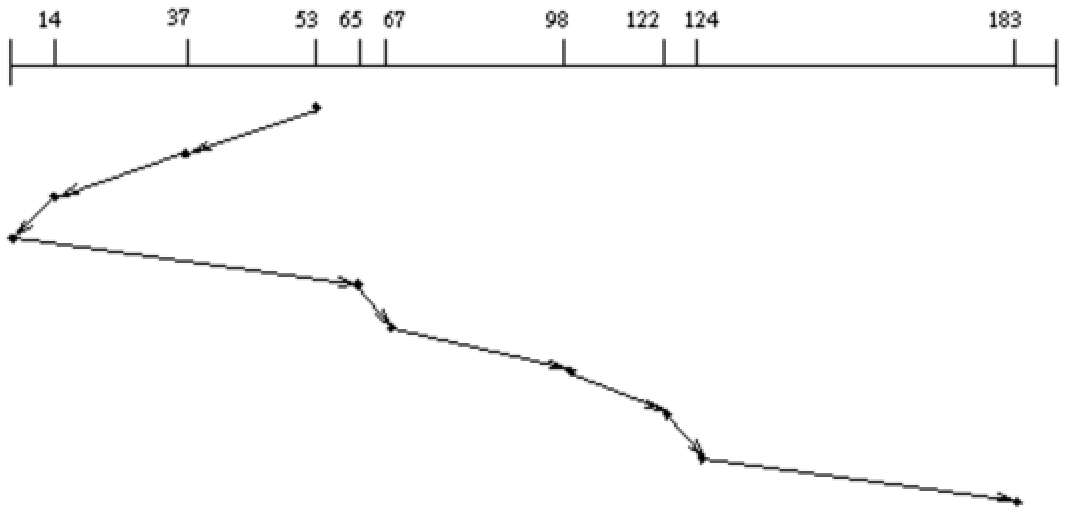
SSTF: Shortest seek time first
Pick the request that's closest on the disk
Con: SSTF good at reducing seeks, but may lead to starvation

Disk Scheduling (2/3)



SCAN: Implements an Elevator Algorithm: take the closest request in the direction of travel

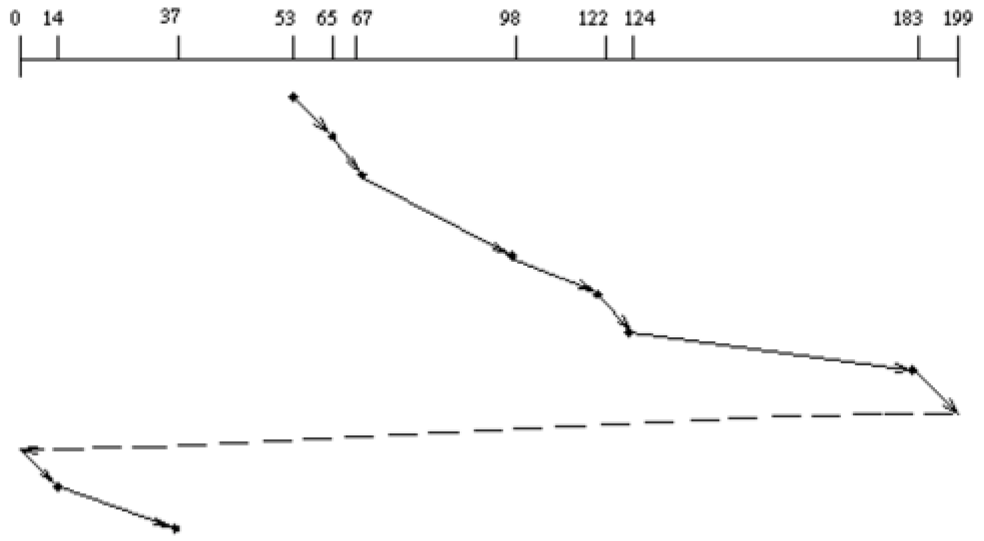
- No starvation, but retains flavor of SSTF



Disk Scheduling (3/3)

C-SCAN: Circular-Scan: only goes in one direction

- Skips any requests on the way back
- Fairer than SCAN, not biased towards pages in middle



Lots of Intelligence in the Controller

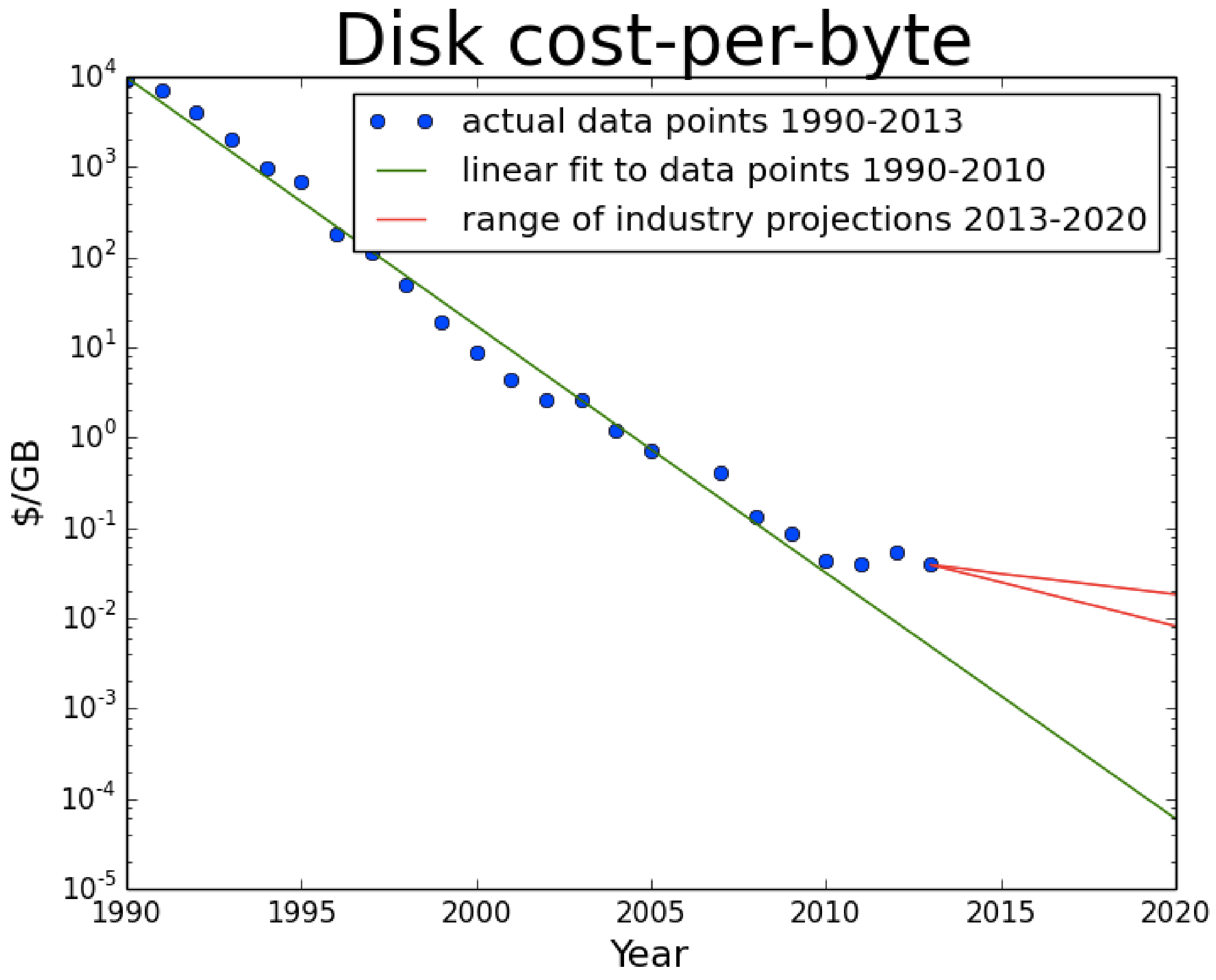
Sectors contain sophisticated error correcting codes
Disk head magnet has a field wider than track
Hide corruptions due to neighboring track writes

Sector sparing
Remap bad sectors transparently to spare sectors on the same surface

Slip sparing
Remap all sectors (when there is a bad sector) to preserve sequential behavior

Track skewing
Sector numbers offset from one track to the next, to allow for disk head movement for sequential ops

Hard Drive Prices over Time



Example of Current HDDs

Seagate Exos X18 (2020)
- 18 TB hard disk
  - » 9 platters, 18 heads
  - » Helium filled: reduce friction and power
- 4.16ms average seek time
- 4096 byte physical sectors
- 7200 RPMs
- Dual 6 Gbps SATA /12Gbps SAS interface
  - » 270MB/s MAX transfer rate
  - » Cache size: 256MB
- Price: $ 562 (~ $0.03/GB)

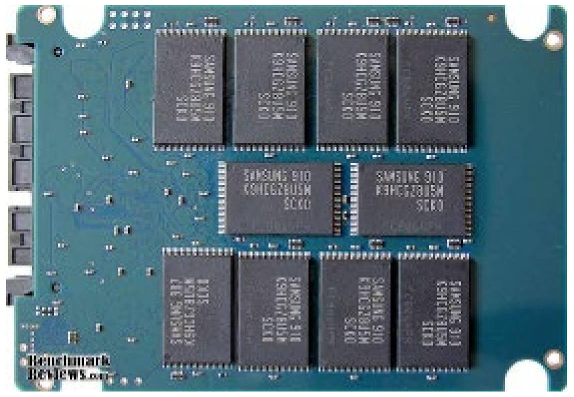
IBM Personal Computer/AT (1986)
- 30 MB hard disk
- 30-40ms seek time
- 0.7-1 MB/s (est.)
- Price: $500 ($17K/GB, 340,000x more expensive !!)

Solid State Drives

1995 – Replace rotating magnetic media with non-volatile memory (battery backed DRAM)

2009 – Use flash memory
- Sector (4 KB page) addressable, but stores 4-64 "pages" per memory block
- Trapped electrons distinguish between 1 and 0

No moving parts (no rotate/seek motors)
- Eliminates seek and rotational delay (0.1-0.2ms access time)
- Very low power and lightweight
- Limited "write cycles"



The Flash Cell

Encode bit by trapping electrons into a cell

Single-level cell (SLC)

Single bit is stored within a transistor

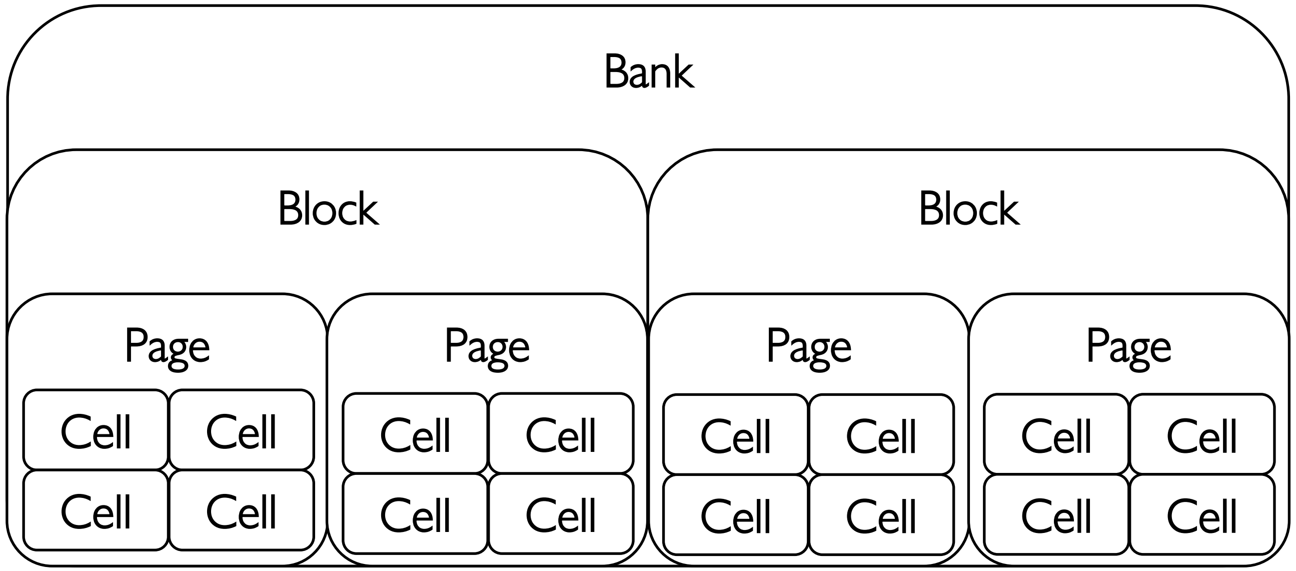
Faster, more lasting (50k to 100k writes before wear out)

Multi-level cell (MLC)

Two/three bits are encoded into different levels of charge

Wear out much faster (1k to 10k writes)

Of banks, blocks, cells



Flash chips organized in banks
- Banks can be accessed in parallel

Blocks 128 KB/256KB
- (64 to 258 pages)

Pages Few KB

Cells 1 to 4 bits

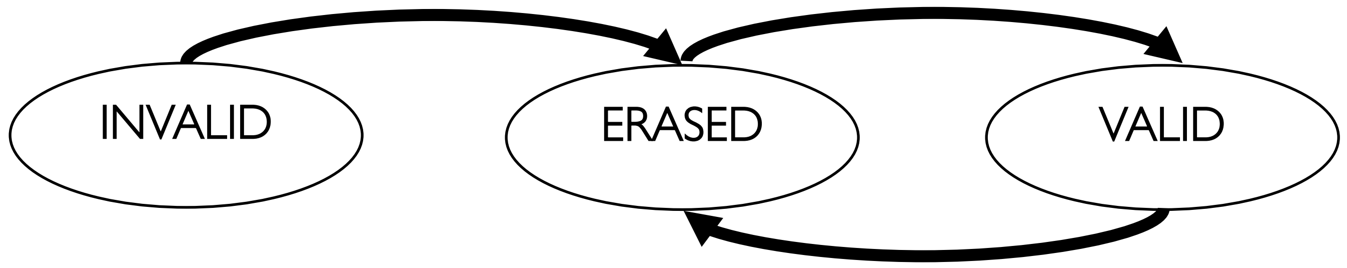
Distinction between blocks and pages important in operations!

Low-level flash operations

How do you read?
- Chip supports reading *pages*
- 10s of microseconds, independently of the previously read page

What about writing? More complicated!
- Must first *erase the block*
  - » Erase quite expensive (milliseconds)
- Once block has been erased, can then *program a page*
  - » Change 1s to 0s within a page.
  - » 100s of microseconds.
- Blocks can only be erased a limited number of times!

Low-level flash operations



		iiii	*Initial: pages in block are invalid* (i)
Erase()	→	EEEE	*State of pages in block set to erased* (E)
Program(0)	→	VEEE	*Program page 0; state set to valid* (V)
Program(0)	→	**error**	*Cannot re-program page after programming*
Program(1)	→	VVEE	*Program page 1*
Erase()	→	EEEE	*Contents erased; all pages programmable*

Low-level flash operations

Assume block of 4 pages. All valid.
Want to write Page 0

Page 0	Page 1	Page 2	Page 3
00011000	11001110	00000001	00111111
VALID	VALID	VALID	VALID

Step 1: erase full block

Page 0	Page 1	Page 2	Page 3
11111111	11111111	11111111	11111111
ERASED	ERASED	ERASED	ERASED

Step 2: program page 0

Page 0	Page 1	Page 2	Page 3
00000011	11111111	11111111	11111111
VALID	ERASED	ERASED	ERASED

SSD Architecture

Recall that SSDs uses low-level Flash operations to provide same interface as HDD

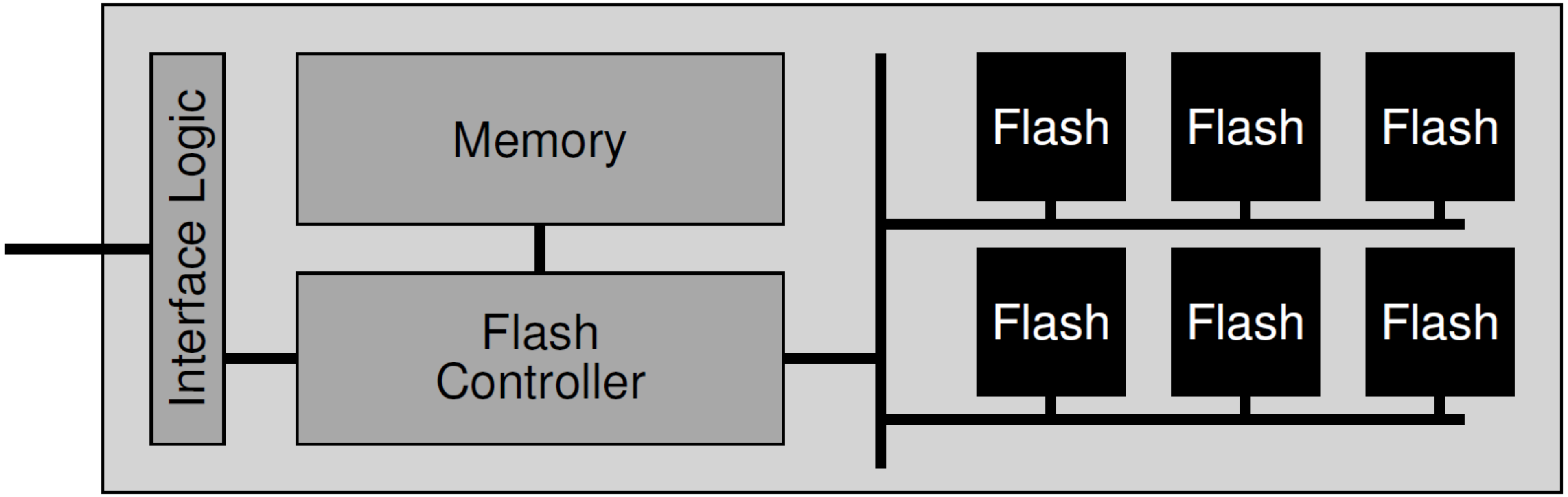
- read and write chunk (4KB) at a time

Reads are easy, but for writes, can only overwrite data one block (256KB) at a time!

Why not just erase and rewrite new version of entire 256KB block?

- Erasure is very slow (milliseconds)
- Each block has a finite lifetime, can only be erased and rewritten about 10K times
- Heavily used blocks likely to wear out quickly

SSD Architecture (Simplified)



Flash Translation Layer (FTL)

- Add a layer of indirection: the flash translation layer
  - Translates request for logical blocks (device interface) to low-level Flash blocks and pages
- Reduce write amplification
  - Ratio of the total write traffic in bytes issues by the flash chip by the FTL devided by the total write traffic issued by the OS to the device
- Avoid wear out
  - A single block should not be erased too often

FTL – Two Systems Principles

FTL uses *indirection* and copy-on-write

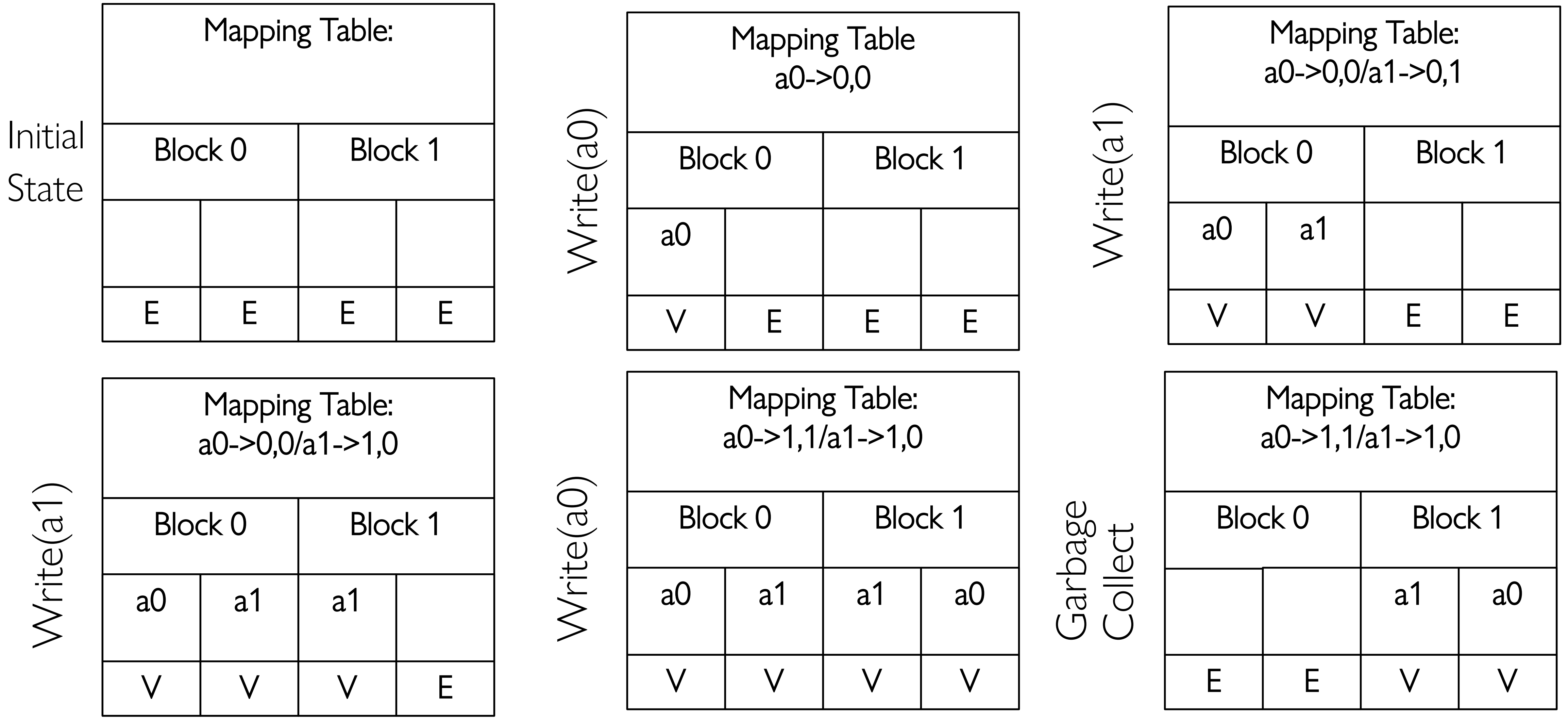
Maintains mapping tables in DRAM

- Map virtual block numbers (which OS uses) to physical page numbers (which flash mem. controller uses)
- Can now freely relocate data w/o OS knowing

Copy on Write/ Log-structured FTL

- Don't overwrite a page when OS updates its data
- Instead, write new version in a free page
- Update FTL mapping to point to new location

FTL Example



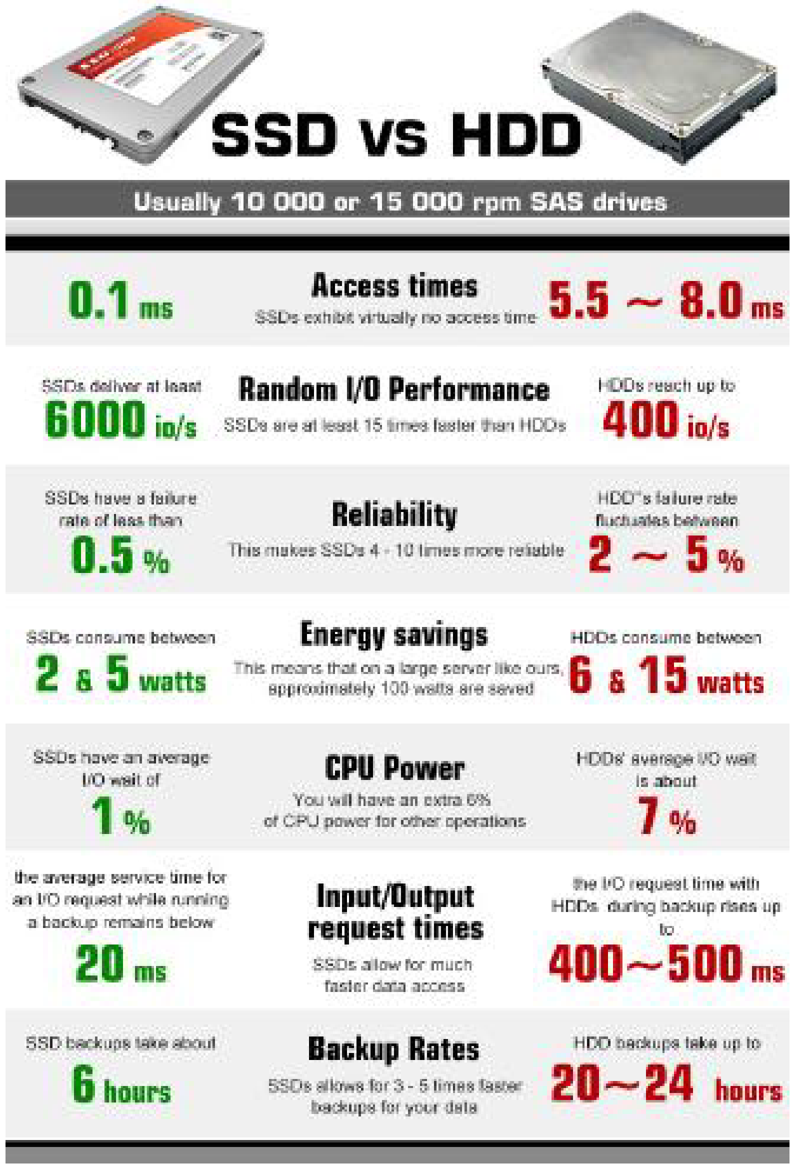
Initial State
Mapping Table:
Block 0
Block 1
E
E
E
E
Write(a0)
Mapping Table
a0->0,0
Block 0
Block 1
a0
V
E
E
E
Write(a1)
Mapping Table:
a0->0,0/a1->0,1
Block 0
Block 1
a0
a1
V
V
E
E
Write(a1)
Mapping Table:
a0->0,0/a1->1,0
Block 0
Block 1
a0
a1
a1
V
V
V
E
Write(a0)
Mapping Table:
a0->1,1/a1->1,0
Block 0
Block 1
a0
a1
a1
a0
V
V
V
V
Garbage Collect
Mapping Table:
a0->1,1/a1->1,0
Block 0
Block 1
a1
a0
E
E
V
V

Some "Current" (large) 3.5in SSDs

- Seagate Exos SSD: 15.36TB (2017)
  - Seq reads 860MB/s
  - Seq writes 920MB/s
  - Price (Amazon): $5495 ($0.36/GB)

- Nimbus SSD: 100TB (2019)
  - Seq reads/writes: 500MB/s
  - Random Read Ops (IOPS): 100K
  - *Unlimited writes for 5 years!*
  - Price: ~ $40K? ($0.4/GB)
    - » However, 50TB drive costs $125 ($0.25/GB)

HDD vs. SSD Comparison



SSD vs HDD

Usually 10 000 or 15 000 rpm SAS drives

SSD		HDD
0.1 ms	**Access times** SSDs exhibit virtually no access time	5.5 ~ 8.0 ms
SSDs deliver at least 6000 io/s	**Random I/O Performance** SSDs are at least 15 times faster than HDDs	HDDs reach up to 400 io/s
SSDs have a failure rate of less than 0.5 %	**Reliability** This makes SSDs 4 - 10 times more reliable	HDD"s failure rate fluctuates between 2 ~ 5 %
SSDs consume between 2 & 5 watts	**Energy savings** This means that on a large server like ours, approximately 100 watts are saved	HDDs consume between 6 & 15 watts
SSDs have an average I/O wait of 1 %	**CPU Power** You will have an extra 6% of CPU power for other operations	HDDs' average I/O wait is about 7 %
the average service time for an I/O request while running a backup remains below 20 ms	**Input/Output request times** SSDs allow for much faster data access	the I/O request time with HDDs during backup rises up to 400~500 ms
SSD backups take about 6 hours	**Backup Rates** SSDs allows for 3 - 5 times faster backups for your data	HDD backups take up to 20~24 hours

HDD	SDD
Require seek + rotation	No seeks
Not parallel (one head)	Parallel
Brittle (moving parts)	No moving parts
Random reads take 10s milliseconds	Random reads take 10s microseconds
Slow (Mechanical)	Wears out
Cheap/large storage	Expensive/smaller storage

Recall: I/O and Storage Layers

High Level I/O — *Streams*

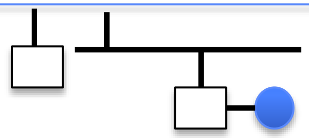
Low Level I/O — *File Descriptors*

Syscall — *open(), read(), write(), close(), …*

Open File Descriptions

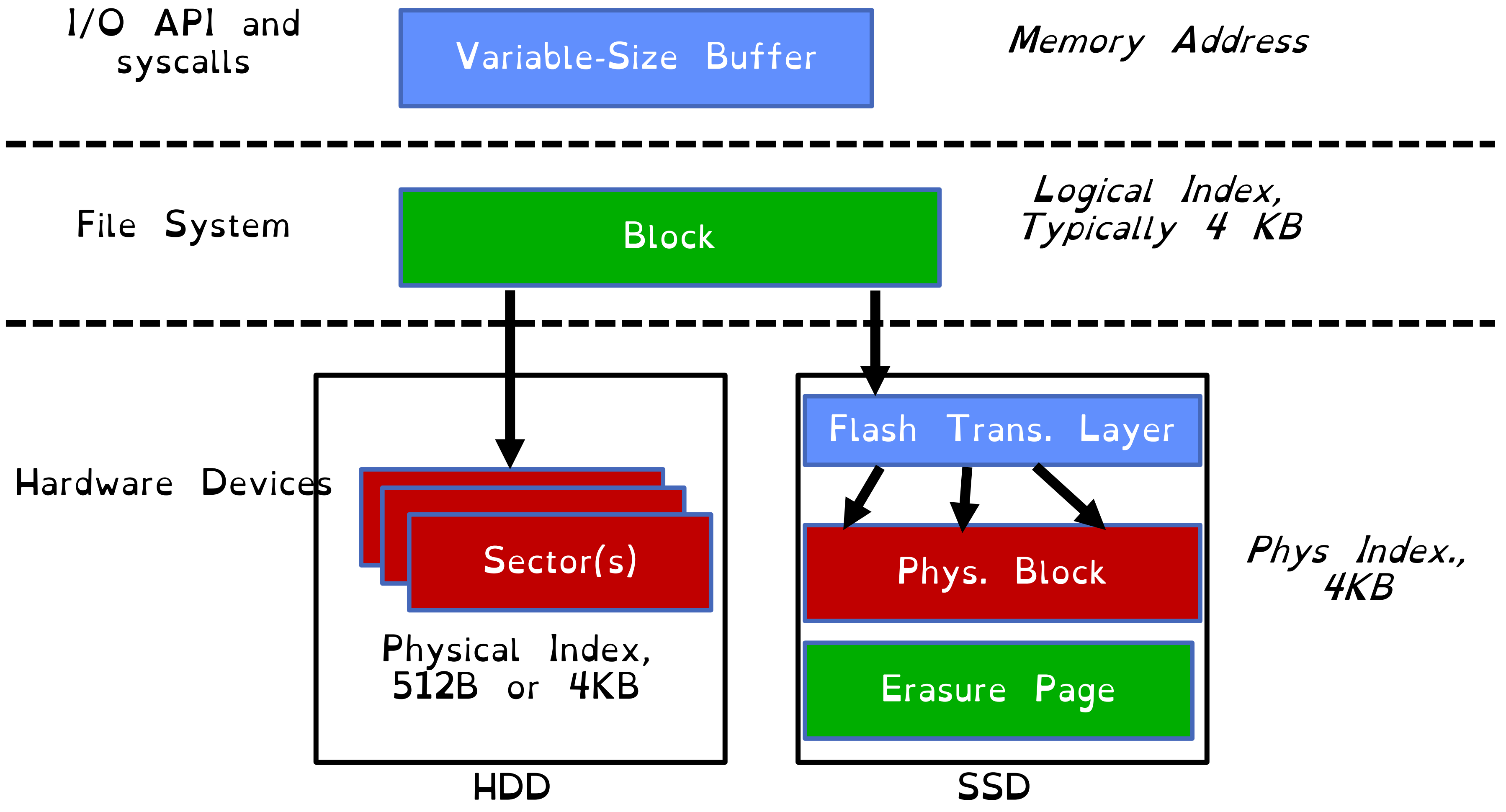
File System — *Files/Directories/Indexes*

I/O Driver — *Commands and Data Transfers*



Disks, Flash, Controllers, DMA

From Storage to File Systems



Building a File System

Layer of OS that transforms block interface of disks (or other block devices) into Files, Directories, etc.

Building a File System

OS as an illusionist:
Take limited hardware interface (array of blocks) and provide a more convenient/useful interface with:

Naming: Find file by name, not block numbers

Organize file names with directories

Organization: Map files to blocks

Protection: Enforce access restrictions

Reliability: Keep files intact despite crashes, failures, etc.

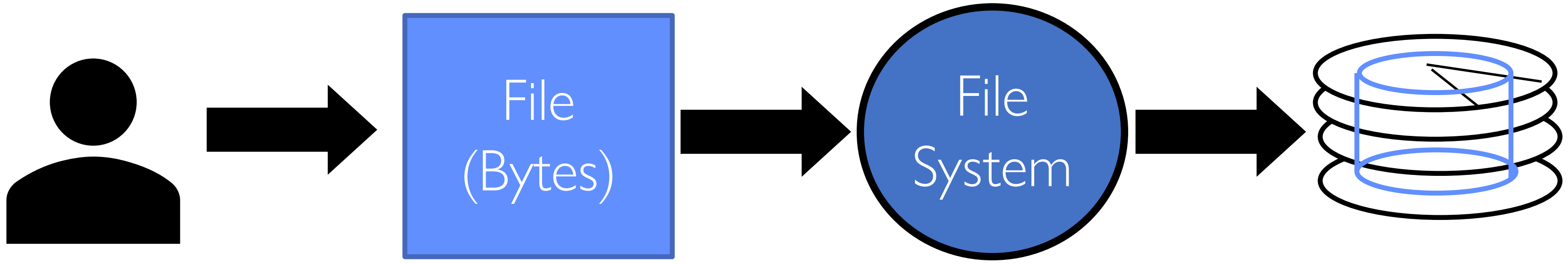
User vs. System View of a File

User’s view:
- Durable Data Structures

System’s view (system call interface):
- Collection of Bytes (UNIX)
- Doesn’t matter to system what kind of data structures you want to store on disk!

System’s view (inside OS):
- Collection of blocks (a block is a logical transfer unit, while a sector is the physical transfer unit)
- Block size ≥ sector size; in UNIX, block size is 4KB

Translation from User to System View



What happens if user says: "give me bytes 2 – 12?"
- Fetch block corresponding to those bytes
- Return just the correct portion of the block

What about writing bytes 2 – 12?
- Fetch block, modify relevant portion, write out block

Everything inside file system is in terms of whole-size blocks

Disk Management

Basic entities on a disk:

File: user-visible group of blocks arranged sequentially in logical space

Directory: user-visible index mapping names to files

The disk is accessed as linear array of sectors

Old: Physical Position [cylinder, surface, sector]

New: Logical Block Addressing (LBA)

Every sector has integer address

Controller translates from address $\Rightarrow$ physical position

Shields OS from structure of disk

What Does the File System Need?

Track free disk blocks

– Need to know where to put newly written data

Track which blocks contain data for which files

– Need to know where to read a file from

Track files in a directory

– Find list of file's blocks given its name

Where do we maintain all of this?

– Somewhere on disk