

# Storage Systems (StoSys)

## XM\_0092

## Lecture 1: Welcome and Introduction

Animesh Trivedi  
Autumn 2020, Period 2



# Expectations...

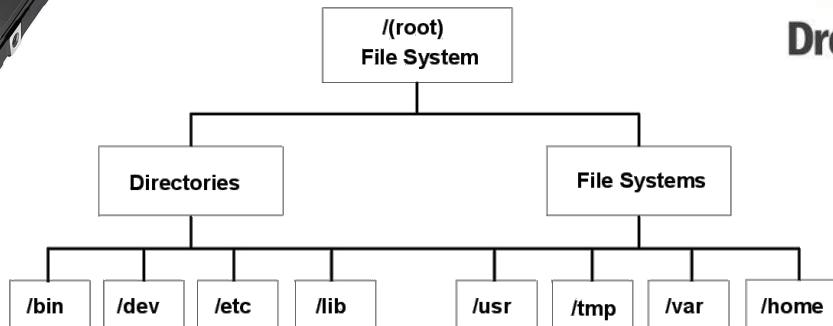
This course builds on prior knowledge from multiple courses. So please refresh your knowledge of

- **Computer Organization** : CPU, devices, interrupts, memory architecture
- **Operating Systems** : Kernel and userspace, processes, synchronization
- **Basic Storage** : See the Canvas page for background PDF reading on FS and HDD
- **Programming** : knowledge of C/C++ and tools (CMake, bash, unit tests)

Please refresh your knowledge of these topics, or consult course slides, and online resources.

Also: *this is not a course about how to write a file system*

# What do you think of: Storage Systems



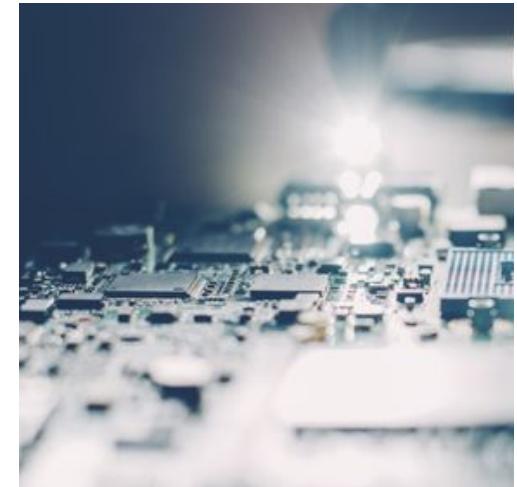
What else I am missing ?

# Storage Systems

**One of the most essential** components in the system

- Can you imagine a computer that just does “calculations” - how exciting that would be

**Probably** it is not the most exciting part of the system as we **were** not releasing headline-grabbing, Super-duper, Gigahertz, a billion Multithreaded, hundreds of Gbps storage device on a daily basis



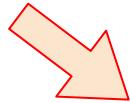
But something changed in mid-2000s that fundamentally changed storage research : Rise of **Non-Volatile Storage** in mainstream/commodity computing

At the end of this course I hope you will be as excited about it as I am

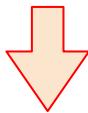
# Data essential to our society



Financial



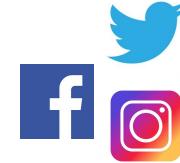
Warehouse



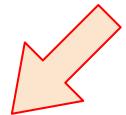
Data is the new oil!



Health



Social



Science

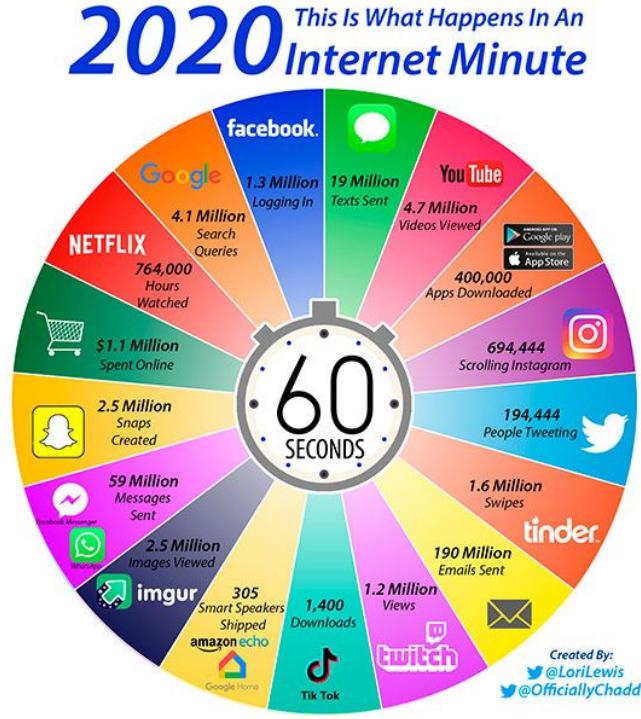
ASTRON



Mobility



# A minute on the Internet



# 200 Zettabytes

$200 \times 1,000,000,000,000,000,000,000$

A logarithmic scale diagram illustrating powers of ten. The scale is represented by a vertical dotted line with seven upward-pointing arrows at regular intervals. To the left of the line, the labels are: *sextillion*, *quintillion*, *quadrillion*, *trillion*, *billion*, *million*, and *thousand*. The labels are positioned such that they align with the arrows, indicating the magnitude of each power of ten.

<https://www.allaccess.com/merge/archive/31294/infographic-what-happens-in-an-internet-minute>  
<https://cybersecurityventures.com/the-world-will-store-200-zettabytes-of-data-by-2025/>

# If that Zettabytes does not resonate

Assume:

- 1 grain of rice is 1 byte of data



# If that Zettabytes does not resonate

Assume:

- 1 grain of rice is 1 byte of data
- **Kilobyte**: a cup of rice



# If that Zettabytes does not resonate

Assume:

- 1 grain of rice is 1 byte of data
- **Kilobyte**: a cup of rice
- **Megabyte**: 8 bags of rice



# If that Zettabytes does not resonate

Assume:

- 1 grain of rice is 1 byte of data
- **Kilobyte**: a cup of rice
- **Megabyte**: 8 bags of rice
- **Gigabyte**: 3 semi-truck



# If that Zettabytes does not resonate

Assume:

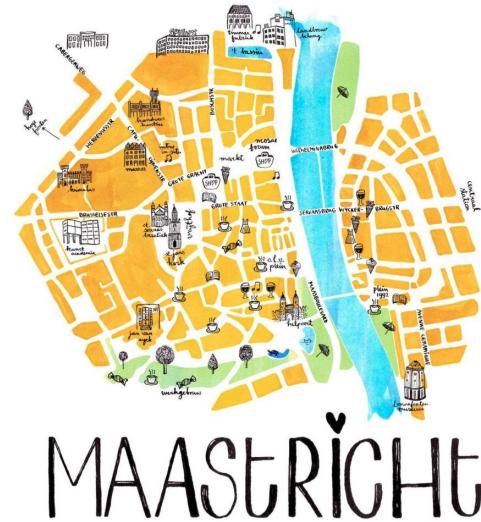
- 1 grain of rice is 1 byte of data
- **Kilobyte**: a cup of rice
- **Megabyte**: 8 bags of rice
- **Gigabyte**: 3 semi-truck
- **Terabyte**: 2 container ships



# If that Zettabytes does not resonate

Assume:

- 1 grain of rice is 1 byte of data
- **Kilobyte**: a cup of rice
- **Megabyte**: 8 bags of rice
- **Gigabyte**: 3 semi-truck
- **Terabyte**: 2 container ships
- **Petabyte**: Covers Maastricht



# If that Zettabytes does not resonate

Assume:

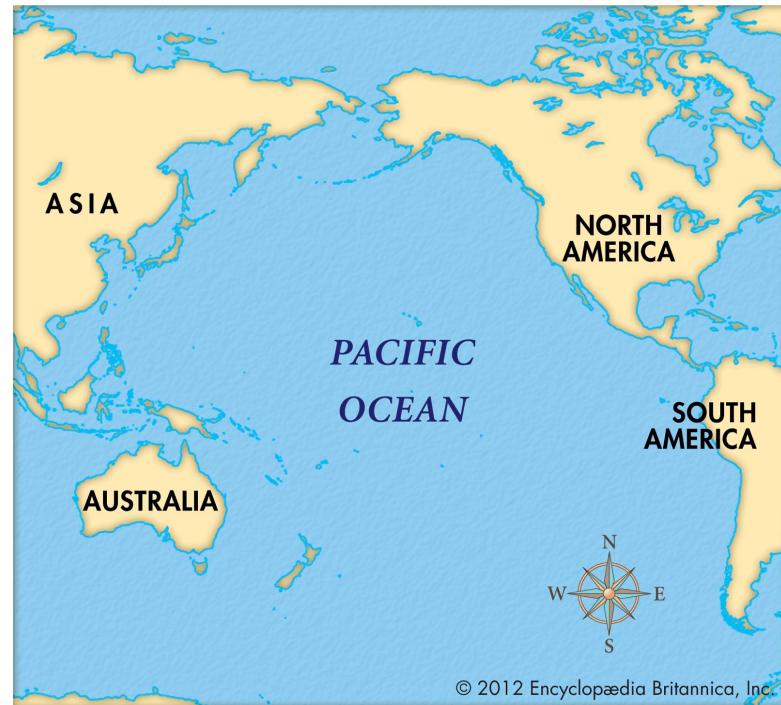
- 1 grain of rice is 1 byte of data
- **Kilobyte**: a cup of rice
- **Megabyte**: 8 bags of rice
- **Gigabyte**: 3 semi-truck
- **Terabyte**: 2 container ships
- **Petabyte**: Covers Maastricht
- **Exabyte**: Covers NL + DE + FR



# If that Zettabytes does not resonate

Assume:

- 1 grain of rice is 1 byte of data
- **Kilobyte**: a cup of rice
- **Megabyte**: 8 bags of rice
- **Gigabyte**: 3 semi-truck
- **Terabyte**: 2 container ships
- **Petabyte**: Covers Maastricht
- **Exabyte**: Covers NL + DE + FR
- **Zettabyte**: Fills up the Pacific Ocean



# If that Zettabytes does not resonate

Assume:

- 1 grain of rice is 1 byte of data
- **Kilobyte**: a cup of rice
- **Megabyte**: 8 bags of rice
- **Gigabyte**: 3 semi-truck
- **Terabyte**: 2 container ships
- **Petabyte**: Covers Maastricht
- **Exabyte**: Covers NL + DE + FR
- **Zettabyte**: Fills up the Pacific Ocean
- **Yottabytes**: An Earth size rice ball

*We are here*



# At the same time

Our needs for timely data analysis are increasing

- Microsecond/millisecond analysis
- Gigabytes, terabytes, petabytes per query
- Scalability, distributed systems
- Energy and efficiency needs

μ-scale data center and clouds

At the end of the day : ***how fast can you read and write your data from storage devices*** (you won't be able to process faster than this)

This is what we will be studying in this course

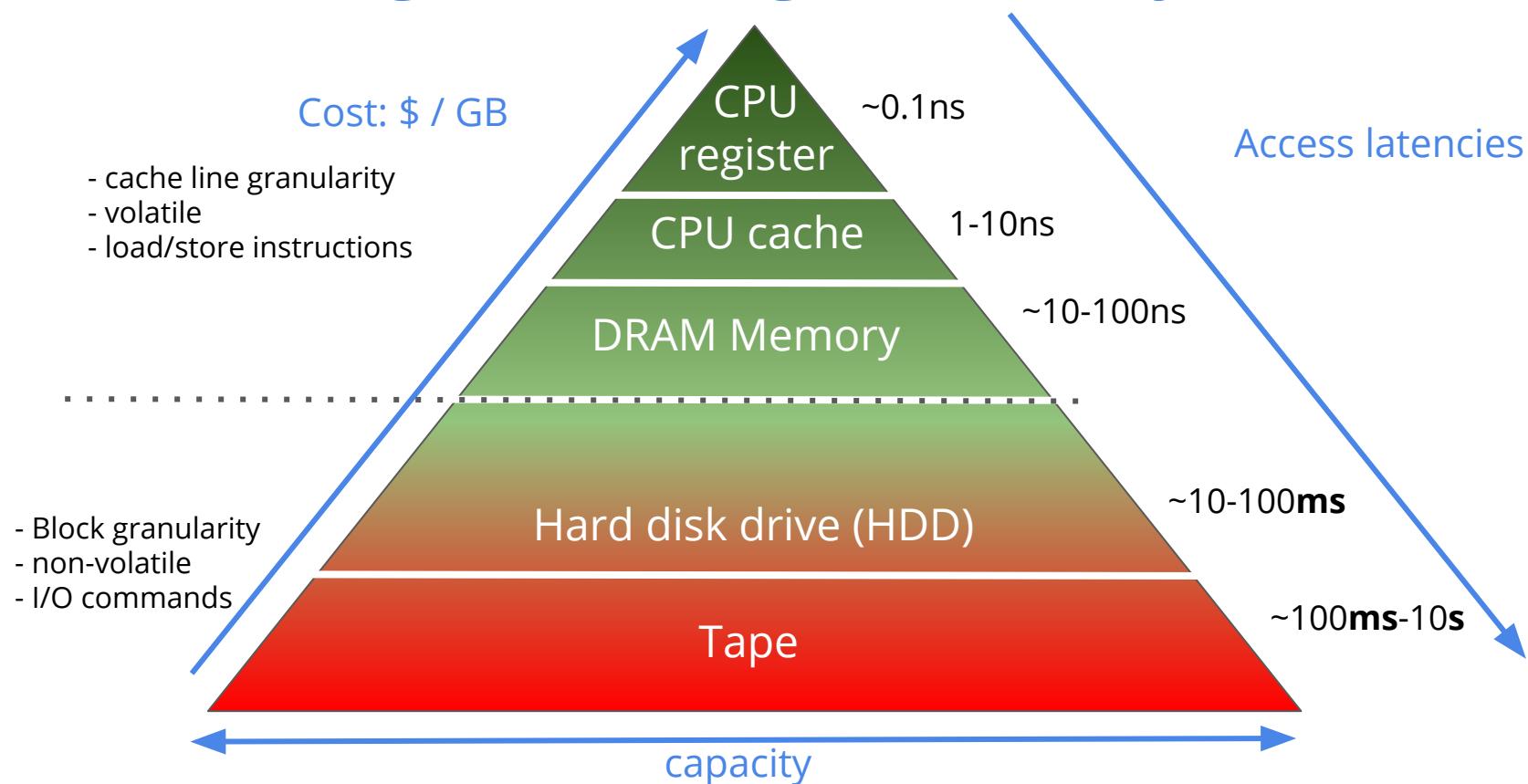
# Scope of this course

Storage systems is a really really large topic, covering ...

- Architecture and devices
- Algorithms and theory (encoding, compression)
- Distributed systems
- Fault-tolerance and failure management
- Application API and abstractions
- Quality of service
- Modeling, cost, and energy
- ...

In this course we will focus on major innovations that has been happening in the area of storage due to the advent of NVM storage from past 10-15 years

# The triangle of storage hierarchy



# Recap: The hard disk drive (HDD) world

Invented by IBM in 1956, in the market 1960s

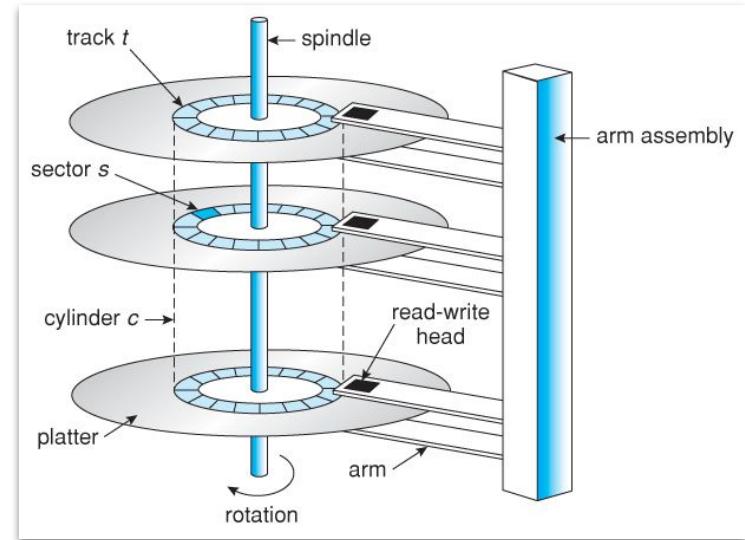
Uses ferromagnetic material to store bits

There are **sectors** (512 bytes) arranged within a **track**, on **platters**

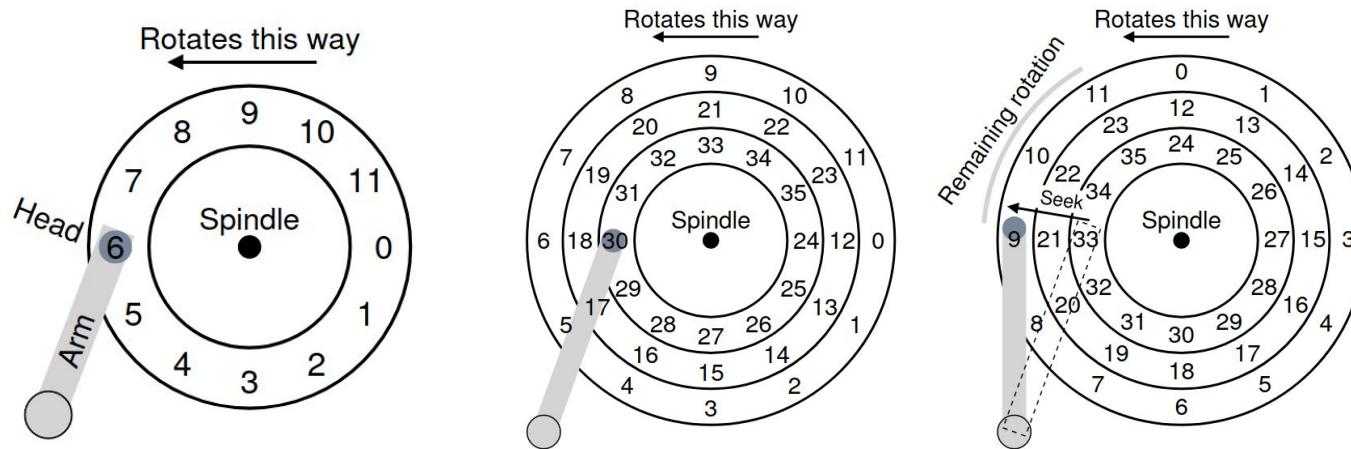
Rotational head to position the sensor on the right sector to read data (7-15K RPM)

On paper, infinite durability in terms of the number of times you can read/write a sector

Has been the primary technology to persistency store data



# HDD disk I/O basics



**Transfer time = rotation + seek + read/write time**

**Typically:**

- Sequential I/O fast, adjacent I/O fast
- Milliseconds for latency
- 100s MB/sec bandwidth (sequential)

	Cheetah 15K.5	Barracuda
Capacity	300 GB	1 TB
RPM	15,000	7,200
Average Seek	4 ms	9 ms
Max Transfer	125 MB/s	105 MB/s
Platters	4	4
Cache	16 MB	16/32 MB
Connects via	SCSI	SATA

# HDD design mantras

1. I/O happens in a sector granularity (512 bytes)
2. Read and write are symmetric - they both have the same performance
3. Random performance (both read/write) are worse than the sequential performance
4. Small I/O performance is bad - cannot amortize the seek and rotational time
5. [Not so well known] Outer vs inner track performance
  - a. Outer tracks rotate with a faster linear speed than Inner tracks
  - b. Hence, faster bandwidth (30+%) and lower latencies
  - c. See section 4.3 <http://cseweb.ucsd.edu/~gporter/papers/tritonsort-nsdi11.pdf>
6. (Theoretically) Infinite durability - magnetic field can be held indefinitely
  - a. Disregarding heating, damages, demagnetization, etc.

# Linux Tools: HDD information

```
atr@evelyn:~$ lsblk -I 8
NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda        8:0    0 238.5G  0 disk 
└─sda1     8:1    0   731M  0 part /boot
└─sda2     8:2    0    1K  0 part 
└─sda5     8:5    0 237.8G  0 part 
  └─sda5_crypt 253:0  0 237.8G  0 crypt 
    ├─ubuntu--vg-root 253:1  0 236.8G  0 lvm   /
    └─ubuntu--vg-swap_1 253:2  0   980M  0 lvm   [SWAP]
```

```
atr@evelyn:~$ sudo hdparm -g /dev/sda
/dev/sda:
geometry      = 31130/255/63, sectors = 500118192, start = 0 .
```

Cylinders / heads / sectors, the size (in sectors)

```
atr@evelyn:~$ sudo hdparm -I /dev/sda
[sudo] password for atr:

/dev/sda:

ATA device, with non-removable media
  Model Number: SAMSUNG MZ7TE256HMHP-000L7
  Serial Number: S1K7NSAG234139
  Firmware Revision: EXT09L6Q
  Transport: Serial, ATA8-AST, SATA 1.0a, SATA II Ext
Standards:
  Used: unknown (minor revision code 0x0039)
  Supported: 9 8 7 6 5
  Likely used: 9
Configuration:
  Logical      max      current
  cylinders    16383    16383
  heads        16       16
  sectors/track 63       63
  --
  CHS current addressable sectors:   16514064
  LBA   user addressable sectors:   268435455
  LBA48 user addressable sectors: 500118192
  Logical Sector size:                 512 bytes
  Physical Sector size:                512 bytes
  Logical Sector-0 offset:              0 bytes
  device size with M = 1024*1024:    244198 MBytes
  device size with M = 1000*1000:    256060 MBytes (256 GB)
  cache/buffer size = unknown
```

<https://www.kernel.org/doc/Documentation/admin-guide/devices.txt>

<https://linux.die.net/man/8/hdparm>

# HDD: Performance improvements over the years

	1983	2011	Improved
CPU Speed	1x10Mhz	4x3GHz	> 1,000x
Memory Size	$\leq$ 2MB	8GB	$\geq$ 4,000x
Disk Capacity	$\leq$ 30MB	2TB	> 60,000x
Net Bwidth	3Mbps	10Gbps	> 3,000x

Improvements in systems performance

- **CPU** - yes
- **Memory** - yes
- **Network** - yes
- **Disk** - yes

# HDD: Performance improvements over the years

	1983	2011	Improved
CPU Speed	1x10Mhz	4x3GHz	> 1,000x
Memory Size	$\leq$ 2MB	8GB	$\geq$ 4,000x
Disk Capacity	$\leq$ 30MB	2TB	> 60,000x
Net Bwidth	3Mbps	10Gbps	> 3,000x

Improvements in systems performance

- CPU - yes
- Memory - yes
- Network - yes
- Disk - yes (*is it?*)

	Mid-1980s	2009	Improvement
Disk capacity	30 MB	500 GB	16667x
Maximum transfer rate	2 MB/s	100 MB/s	50x
Latency (seek + rotate)	20 ms	10 ms	2x
Capacity/bandwidth (large blocks)	15 s	5000 s	333x worse
Capacity/bandwidth (1KB blocks)	600 s	58 days	8333x worse
Jim Gray's Rule [11] (1KB blocks)	5 min.	30 hours	360x worse

*Bw bound*

*Latency bound*

*Cost/access bound*

# HDD : Achilles heels of systems building

**Latency limits** : can not rotate faster than 10s thousands / sec

- Cannot drop latency below a few milliseconds
- Various caching, prefetching techniques in DRAM

**Random performance** very poor (a few IO operations/sec, 10 MB/sec)

- Buffering operation and write them in a single large request to amortize rotational and seek latencies

**Fundamentally:** all these limitations are physical limits (cannot be improved)

- *There is an arm sensor movement*
- *There is a rotation of spindle*

***Imagine a storage without movement***

# Enter: USB Drive (*what is inside a USB drive*)



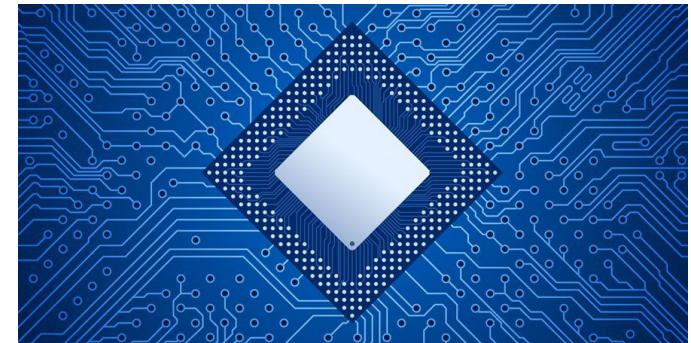
<https://www.quill.com/nxt-technologies-32gb-usb-30-flash-drive-nx27996/cbs/55362528.html>

# Non-Volatile Memory (NVM) Technology

Umbrella term for many related technologies (Solid-State Storage, Storage-Class Memory - *we are rubbish with names :P*)

Overall the idea is to use physical properties of media  
store bits and data

- Magnetic state
- **Electrical charge** ←
- Crystalline / amorphous state
- Resistance
- Optical



**No moving parts !**

# Caution / Confusion

What is a **memory** and what is a **storage**

Definition is a bit more fluid these days



- There are technologies which might have “memory” as a keyword in their name
  - You can use them to make a RAM as well (but will be slow, not efficient)
- Then there is how they are packed and used with operating systems
  - Memory - something that can be directly addressed from the CPU (load/store)
  - Storage - integrated with a block data transfer to DRAM

*Focus on understanding the key concept, without getting dragged with names*

# Flash Memory

Invented by Toshiba

- Since mid-1980s commercially available
- The basic idea of a floating gate:  
1960s with the MOSFET technology

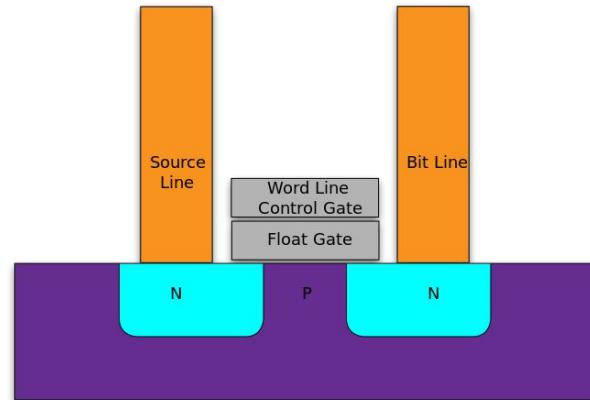
Has been around for more than 30 years

First became popular in with embedded devices

In the mainstream computing from the mid-2000s

One of the most popular technologies to store data today

A very fundamental shift how to store and manage data



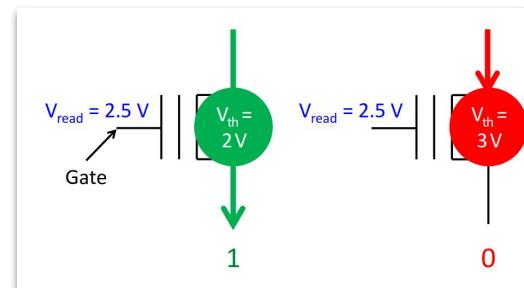
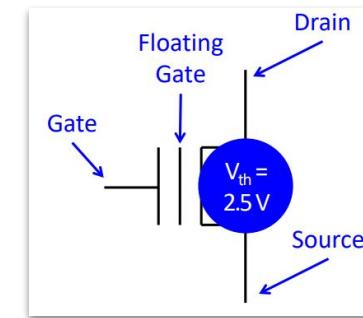
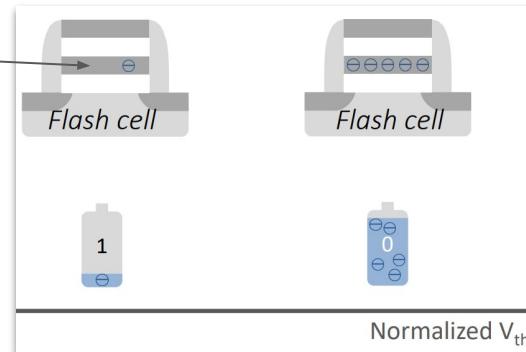
# Flash Storage - Storing data, bits by bits

The *isolated* Float Gate controls current flowing through the flash cell transistor

Electrons are trapped in FG - giving it a threshold voltage ( $V_{th}$ )

Apply a known reference voltage ( $V_{read}$ ) at the gate to find out if there are electrons trapped in the FG

- Pass through = 1
- Blocked = 0

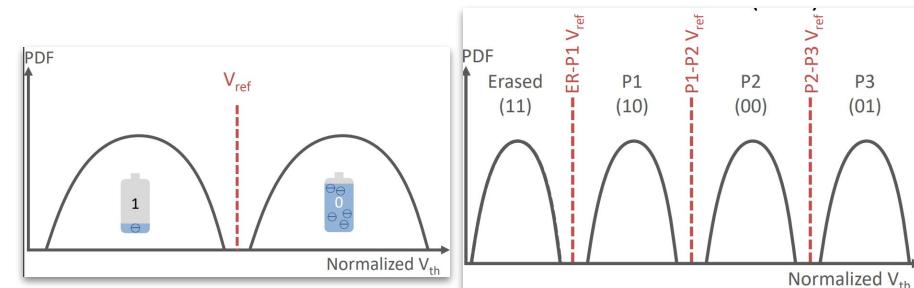


Hence, a single cell can store a zero or one

# Flash Storage - Storing data, bits by bits

Flash storage cell can be Single-Level (**SLC**), Multi-level (**MLC**) or Triple-Level (**TLC**)

- SLC = 0, presence = 1 (2 values)
- MLC = {00, 01, 10, 11} (4 values)
- TLC = {000, 001...111} (8 values)
- QLC (now available), PLC (in development)



SLC cells have the highest performance

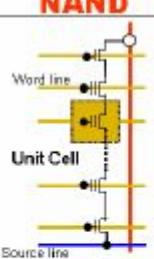
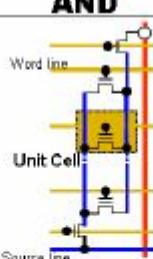
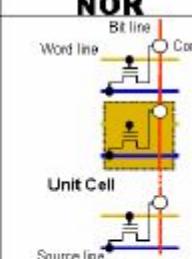
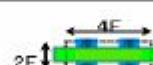
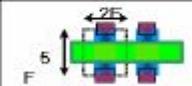
TLC cells have the highest density, better \$/byte cost

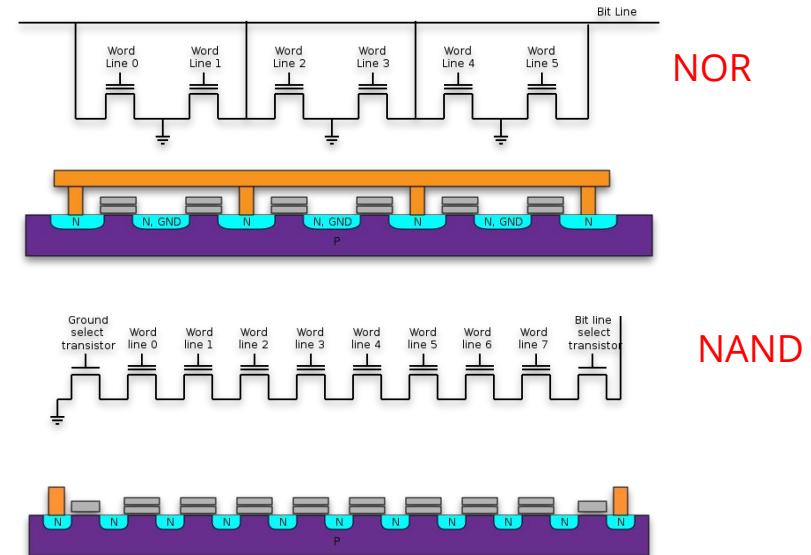
A FG must be re-programmed / erase (P/E) continuously, leading to an erosion of the isolated oxide layer - **finite P/E life cycle**

Multiple reads over the time can also erode the charge : **read disturbance**

# NAND, NOR, and AND Flash cells

There are multiple ways in which you can pack your cell calling the right trade-off between density, addressability, read and write performance, no. of connections, etc.

	<b>NAND</b>	<b>AND</b>	<b>NOR</b>
Cell Array			
Layout			
Cross-section			
Cell size	<b>4F<sup>2</sup></b>	<b>8F<sup>2</sup></b>	<b>10F<sup>2</sup></b>



# NAND and NOR

Historically there was a debate which type will win: **who won?**

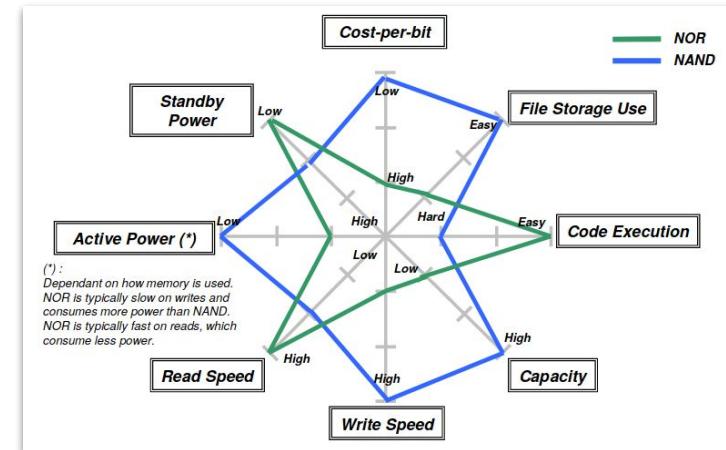
**NOR Flash** cells are often used in BIOS ROMs, for code execution

- High erase, but byte-addressable

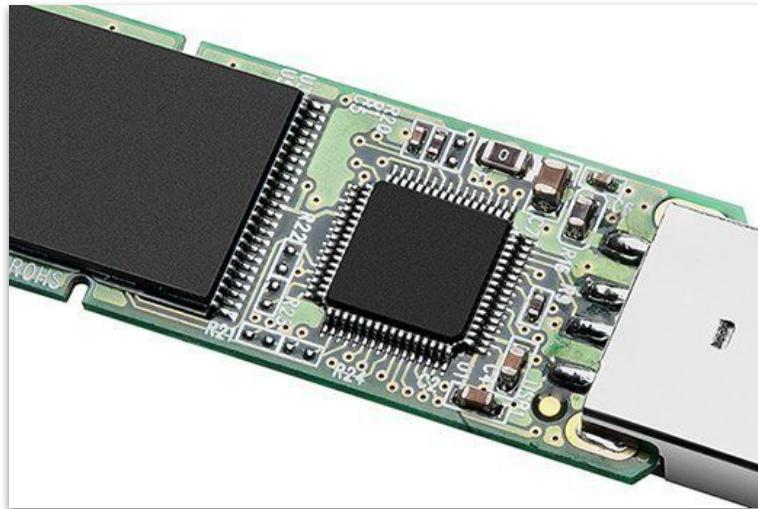
**NAND flash** cells are packed in a high density manner, used for mass storage

- High density, block addressable

	SLC NAND Flash (x8)	MLC NAND Flash (x8)	MLC NOR Flash (x16)
<b>Density</b>	512 Mbits <sup>1</sup> – 4 Gbits <sup>2</sup>	1Gbit to 16Gbit	16Mbit to 1Gbit
<b>Read Speed</b>	24 MB/s <sup>3</sup>	18.6 MB/s	103MB/s
<b>Write Speed</b>	8.0 MB/s	2.4 MB/s	0.47 MB/s
<b>Erase Time</b>	2.0 mSec	2.0mSec	900mSec
<b>Interface</b>	I/O – indirect access	I/O – indirect access	Random access
<b>Application</b>	Program/Data mass storage	Program/Data mass storage	eXecuteInPlace

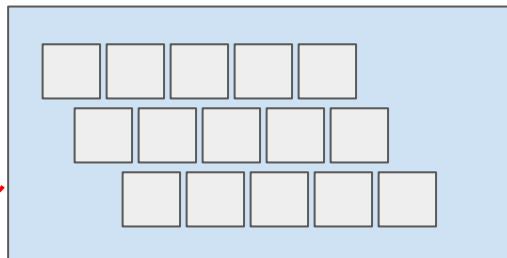


# So how things are arranged inside a drive?



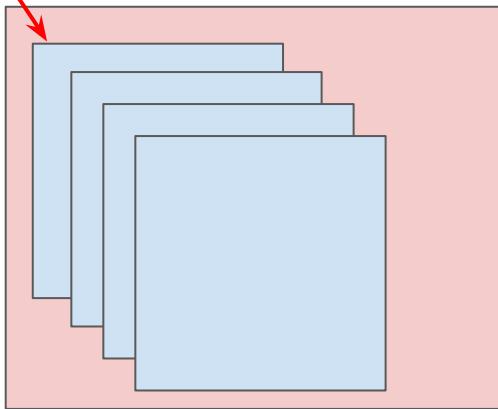
*So, why is this question important?*

# Making flash pages and blocks from flash cells



A bunch of flash cells are packed in a **page** : typically 4kB

- Typically, a unit of I/O - the page (similar to a sector size in HDD)
  - Note: different vendors/systems might use different names like sector or even block (in project Handbook: block)



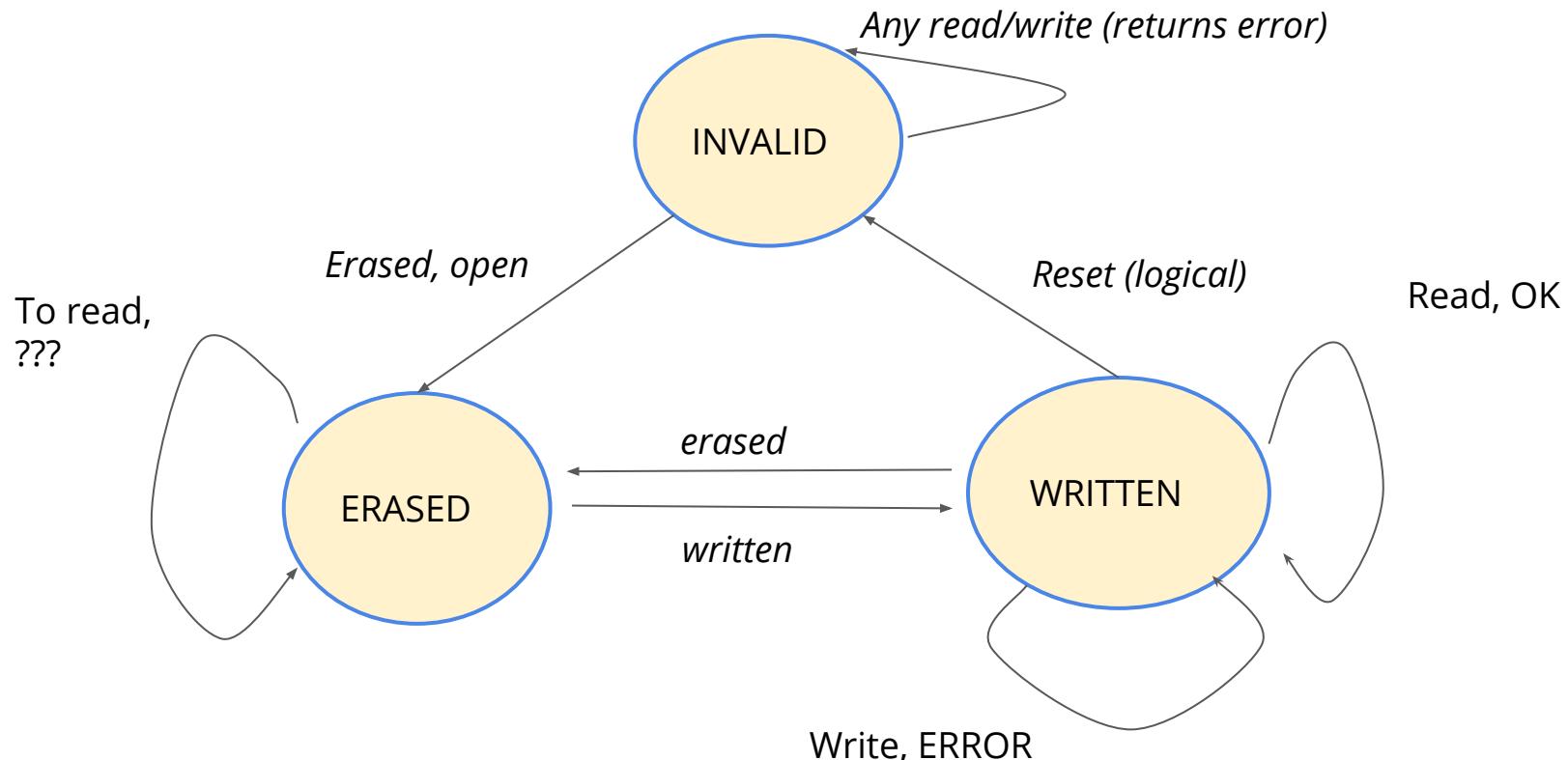
A bunch of flash pages packed together as a **block**: typically 64-128-256 pages (hence, a few Megabytes)

- Typically, a unit of erase (bulk erasure)
- **Recall**: you need to erase before you can write on a flash cell
  - Note: different vendors/systems might use different names like chunk (in project Handbook: chunk)

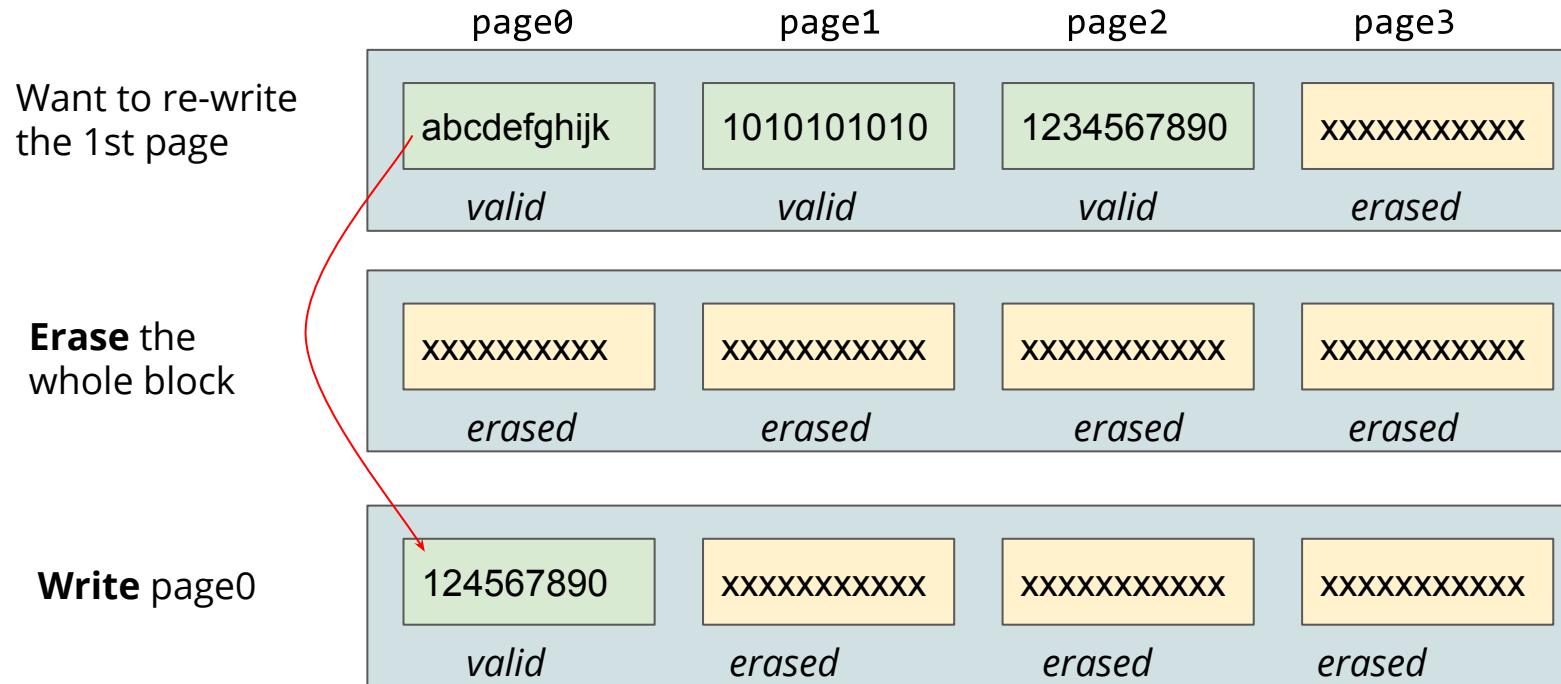
# Flash page: Basic operations

1. **Read (a page):** logically the same as reading a sector, `read(page_number)`
  - a. Internally just applying voltage and reading values
  - b. No moving parts, hence, very fast 10s of microseconds
  - c. Does it matter you are reading 1st or 1024th page? NO: hence, *uniform fast, random reads performance*
2. **Erase (a block):** reset a block (a few Megabytes)
  - a. Copy the content (if any) that you want to save to another location
  - b. Erasure the whole block (Program-Erase cycle)
  - c. Typically slow, a few to 10-100s of milliseconds
3. **Write (a page):** Once a block has been erased, `write(page_number)`
  - a. Can only write in an incremental page order inside a block
  - b. Less expensive than a erase, but more expensive than a read - typically, 50-100 microseconds
  - c. Once written, can not overwrite before an erase
  - d. Can have different read/write granularity based on the packaging

# Simplified flash page state machine (impl. dependent)



# Write - Erase Operation in Detail



Note: the content of page1 and page2 are lost, hence, if you want to save them then you have to copy out the whole block - we will come back to this problem later again

# Typical Values of Operations

	<b>Read (usec)</b>	<b>Write (usec)</b>	<b>Erase (usec)</b>	<b>P/E cycles</b>
<b>SLC</b>	25	200-300	1,500-2,000	~100,000
<b>MLC</b>	50	~600-900	~3,000	~10,000
<b>TLC</b>	75	~900-1,350	~4,500	~5,000

These are typical values for a **flash physical chips**, but a lot of progress has been made lately. So a value that you might see in your operating system (**flash storage device**) depends upon :

- The generation of the device
- How expensive it is (internal implementation)
- What cell technology it is using
- How old is it
- How much data you have written to it ....and many more

# So far: Flash vs HDD (at the cell level)

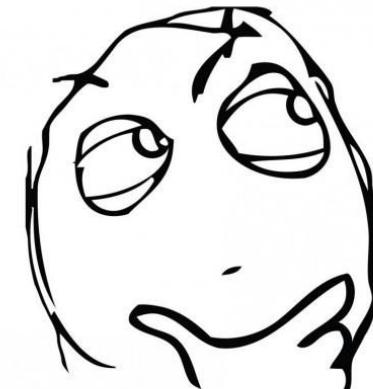
1. No moving parts inside a flash storage
2. Flash read latencies are much lower than HDDs (10s of msecs vs 10s of usecs)
3. Asymmetric read and write performances, 10 (r) vs 100 (w) useconds
  - a. HDD have the same
4. No overwriting (in-place update) the same page again (HDD can)
5. [new operation] Erase operation, must erase a whole block before writing
6. [new activity] If there are valid data pages, must copy data before erase
7. Finite number of times one can erase and program flash cells
  - a. Nonetheless, like with any electron based storage, they leak and data will be eventually lost
  - b. HDDs have (theoretically) infinite durability
8. Read performance is uniform - no difference with random or sequential ***at the cell level*** (*what you get at the device level, we will see*)
  - a. Write - *what do you think?*
9. Anything else that you could think of ?

# How do we make a device out of Flash cells?

We have seen how an individual cells/pages behave

How do we pack them as single “device” so that:

- They have sectors
- They have addressable sectors
- We can issue a read or write command
- We can attach to our computers / servers
- We can install a file system on it
- Who will do erase
- How do we rewrite the same “location” again



# Milestone work - must read

Most details from this lecture come from this paper. Though values might have changed the concepts, concerns, and trade-offs remain true still today.

**Design Tradeoffs for SSD Performance**

Nitin Agrawal\*, Vijayan Prabhakaran, Ted Wobber,  
John D. Davis, Mark Manasse, Rina Panigrahy  
*Microsoft Research, Silicon Valley*  
*\*University of Wisconsin-Madison*

**Abstract**  
*Solid-state disks (SSDs) have the potential to revolutionize the storage system landscape. However, there is little published work about their internal organization or the design choices that SSD manufacturers face in pursuit of optimal performance. This paper presents a taxonomy of such design choices and analyzes the likely performance of various configurations using a trace-driven simulator and workload traces extracted from real systems. We find that SSD performance and lifetime is highly workload-sensitive, and that complex systems problems that normally appear higher in the storage stack, or even in distributed systems, are relevant to device firmware.*

**1 Introduction**  
The advent of the NAND-flash based solid-state storage device (SSD) is certain to represent a sea change in the architecture of computer storage subsystems. These devices are capable of producing not only exceptional

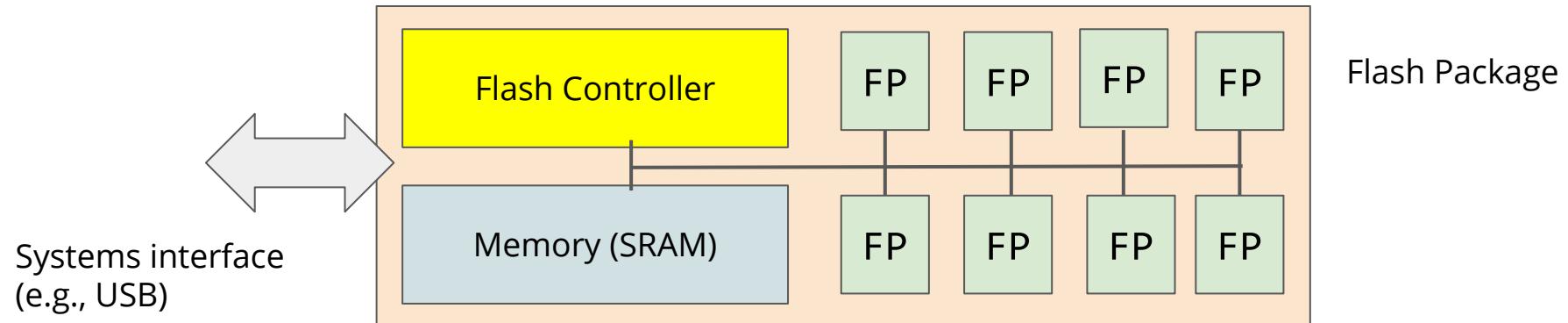
their design. Where such knowledge exists, it typically remains the intellectual property of SSD manufacturers. As a consequence, it is difficult to understand the architecture of a given device, and harder still to interpret its performance characteristics.

In this paper, we lay out a range of design tradeoffs that are relevant to NAND-flash solid-state storage. We then analyze several of these tradeoffs using a trace-based disk simulator that we have customized to characterize different SSD organizations. Since we can only speculate about the detailed internals of existing SSDs, we base our simulator on the specified properties of NAND-flash chips. Our analysis is driven by various traces captured from running systems such as a full-scale TPC-C benchmark, an Exchange server workload, and various standard file system benchmarks.

We find that many of the issues that arise in SSD design appear to mimic problems that have previously appeared higher in the storage stack. In solving these hard problems, there is considerable latitude for design

[https://www.usenix.org/legacy/events/usenix08/tech/full\\_papers/agrawal/agrawal.pdf](https://www.usenix.org/legacy/events/usenix08/tech/full_papers/agrawal/agrawal.pdf)

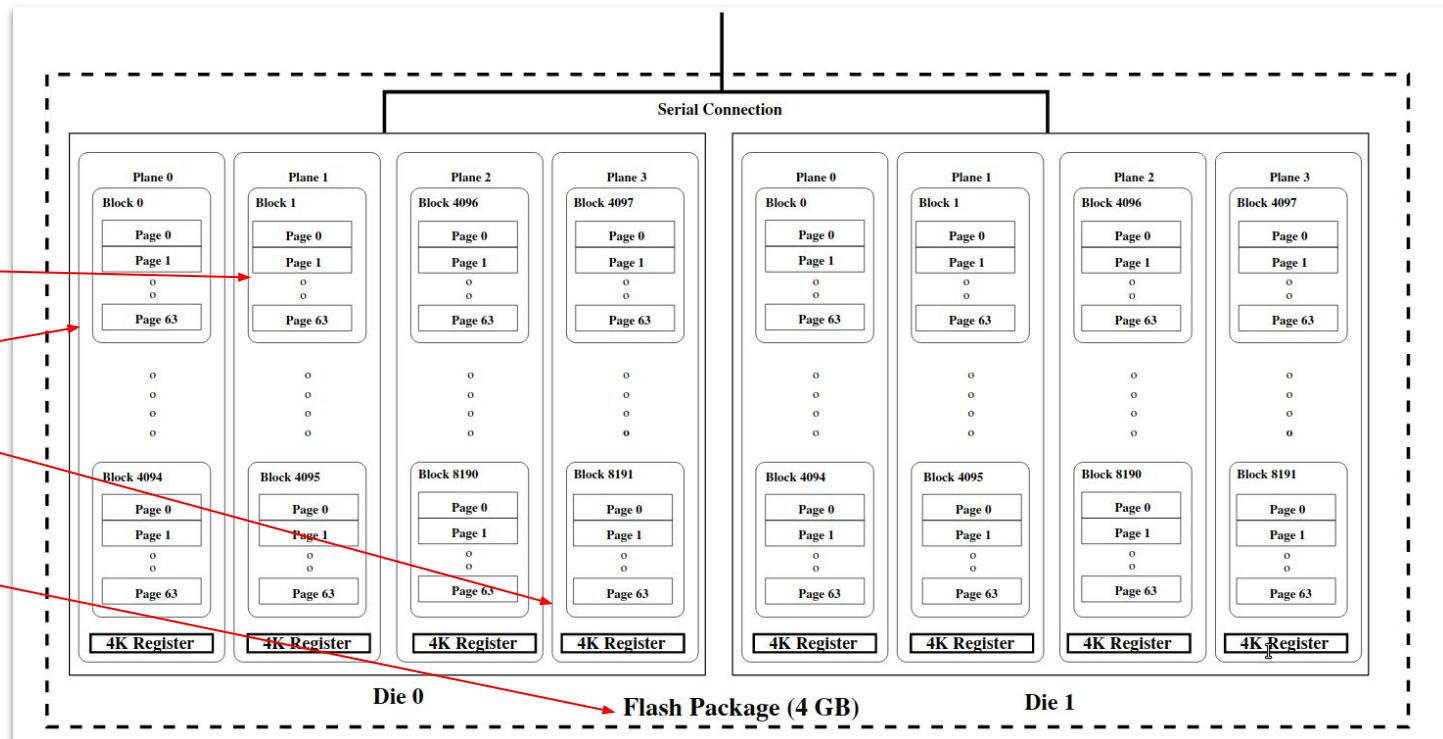
# What is inside the device



- Flash chips are arranged with a connector (could be any topology, but here a line)
- Flash controller: a microcontroller, programs the flash chips
- Memory : fast, buffer memory to hold data in transit
- System interface: can support, SATA, SAS, USB, PCIe (we will see)

# Flash Package (FP) Layout

Pages in blocks  
Blocks in planes  
Planes in dies  
Dies in packages



# Flash Package (FP) layout

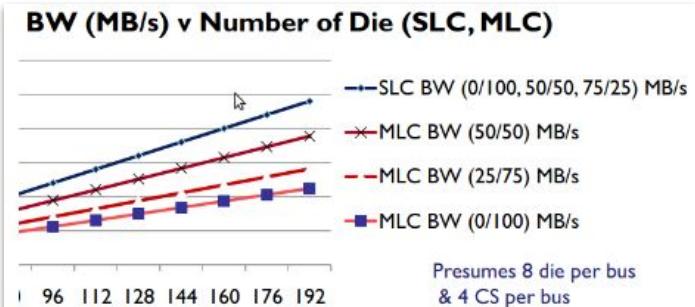
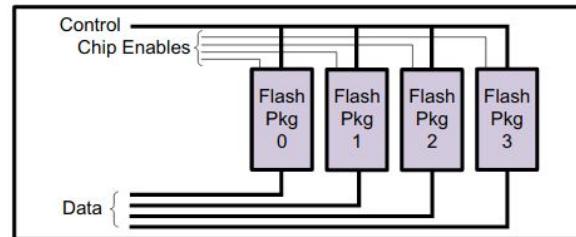
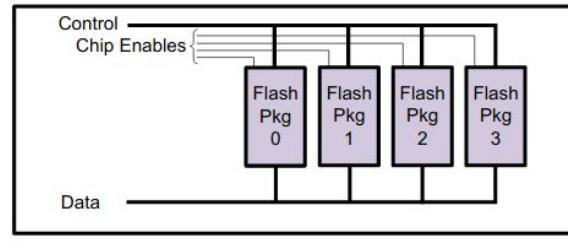
Different packages might serve I/O in parallel

Different packages might serve control in parallel

Different planes may work in parallel with a single command addressability

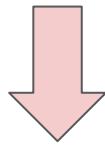
- Can move from plane 0 to 1, but not 2 (without buffering in between)
- Depends upon in the internal circuitry
  - Cost function

Multiple dies can be packaged together to push bandwidth



# The Read/Write interface?

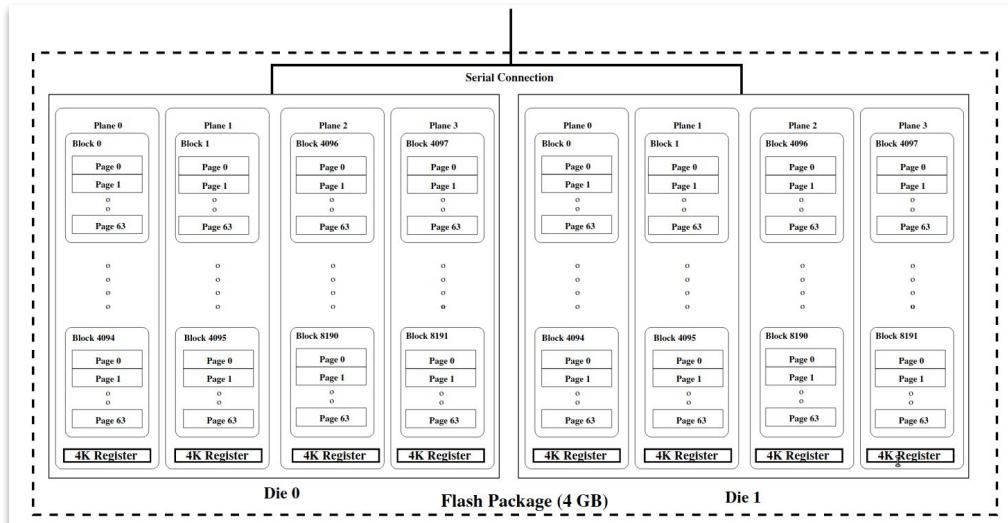
Write ( $p1, d0, b52, p211$ )?



Read ( $p1, d42, b13, p521$ )?

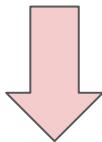


/dev/ssd0



# The Read/Write interface?

Write ( $p1, d0, b52, p211$ )?



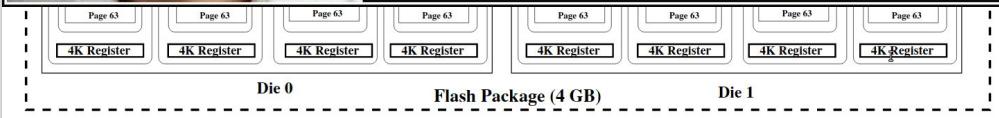
Read ( $p1, d42, b13, p521$ )?



All problems in Computer Science  
can be solved by another level of  
indirection.

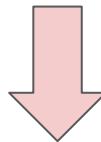
— *Butler Lampson* —

AZ QUOTES



# The Read/Write interface?

Write ( $p1, d0, b52, p211$ )?



Read ( $p1, d42, b13, p521$ )?

0

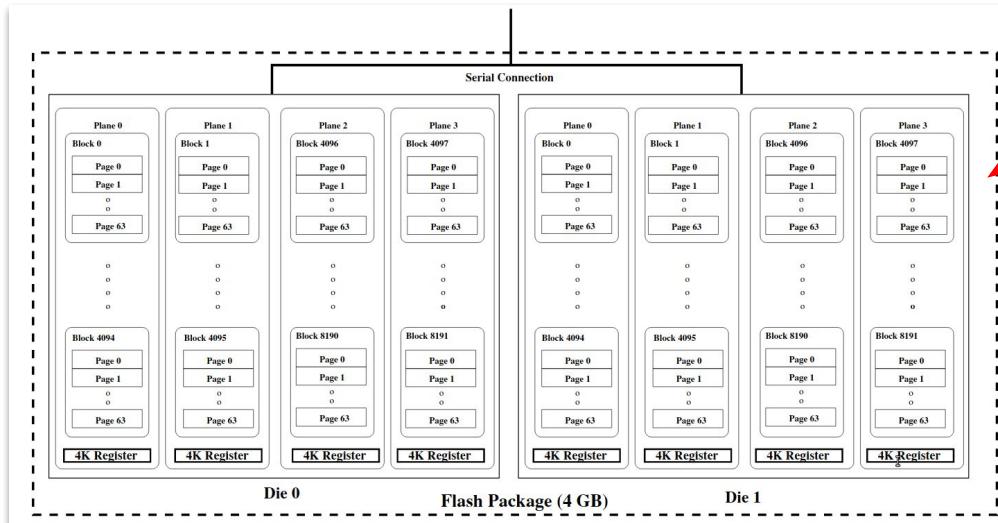
**Logical Block Address (LBA)**

(almost the same as the sector, but different size - page size typically)

Internally we can map from LBA to physical locations

/dev/ssd0

dev\_capacity

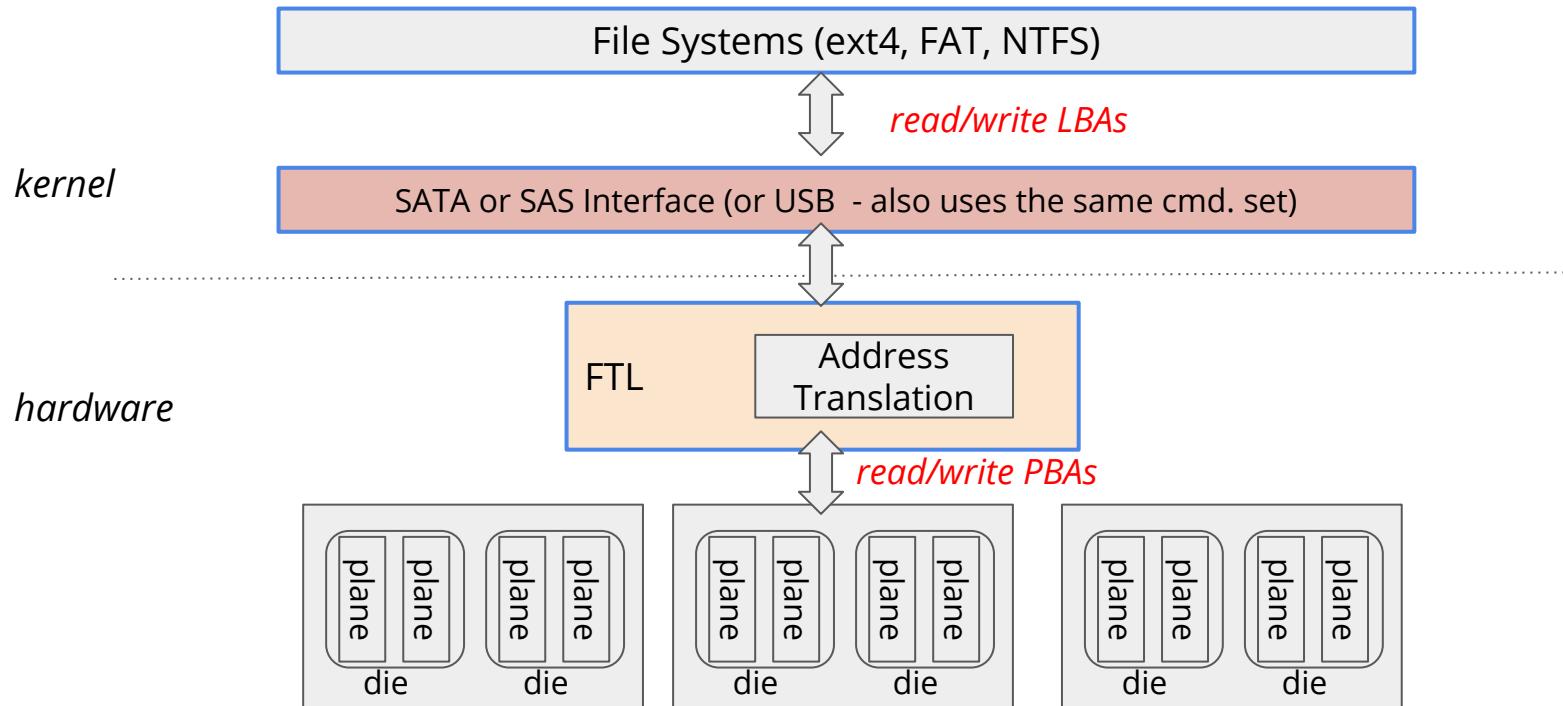


**Physical Block Address (PBA)**

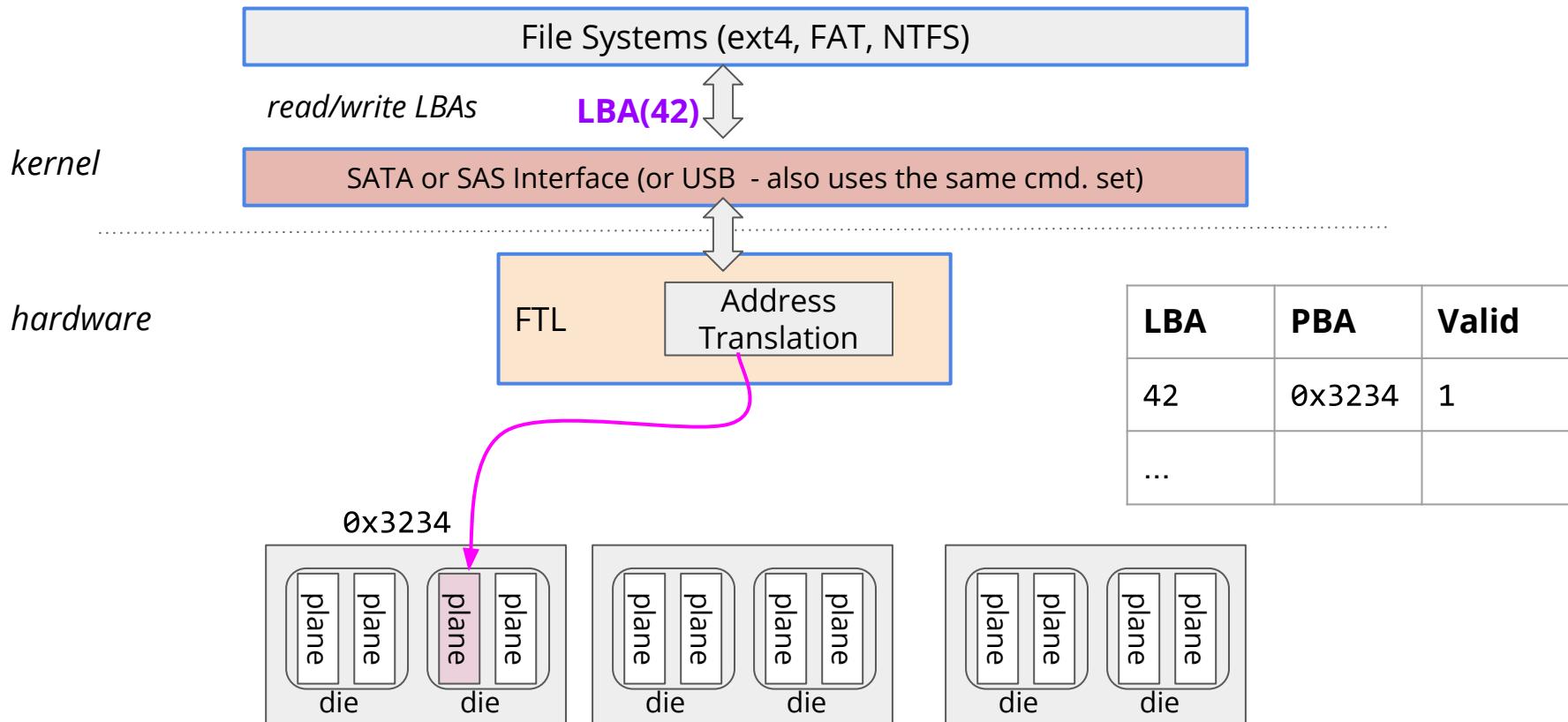
Internal device enumeration, their own business

# Flash Translation Layer (FTL)

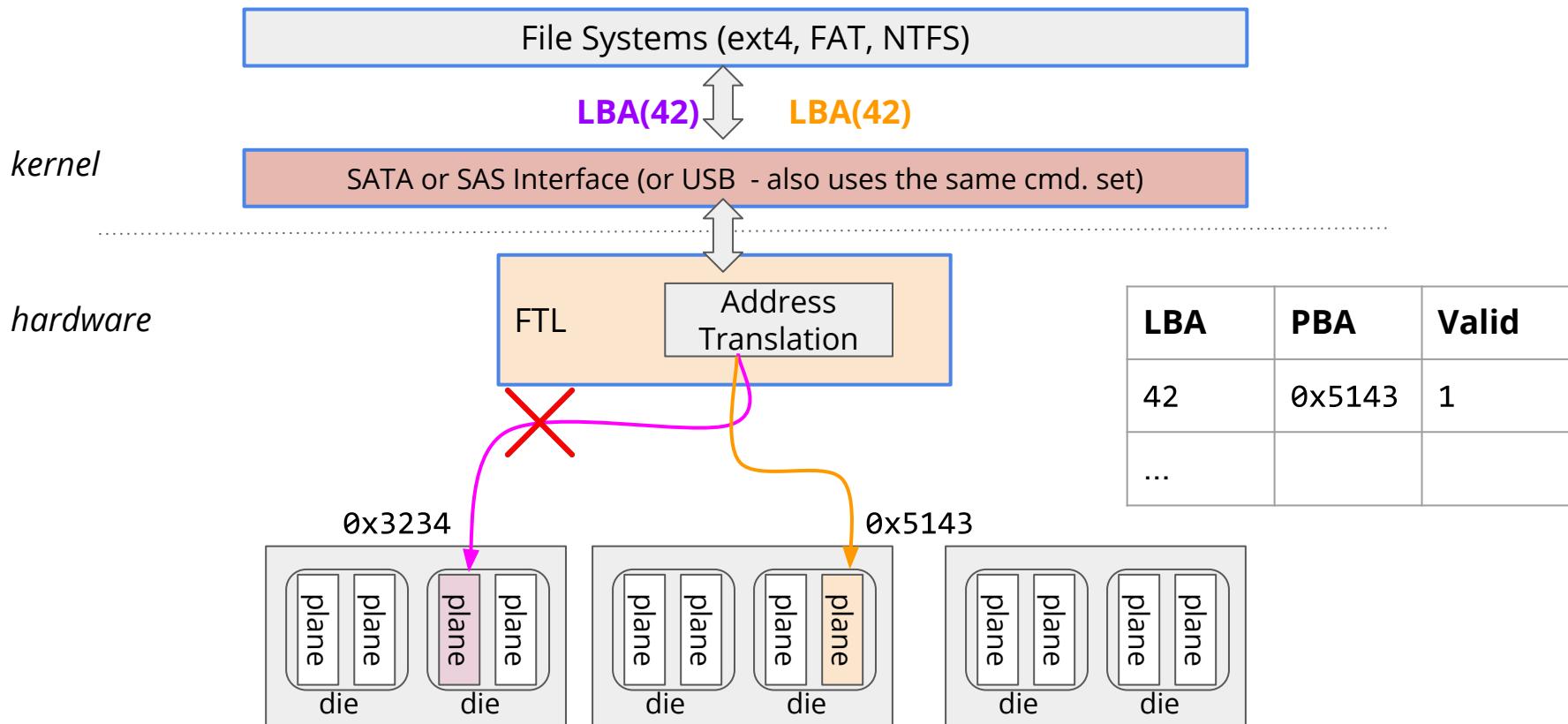
A piece of logic that runs on the controller and is responsible for : **Address Translation**



# What Happens when you overwrite a LBA



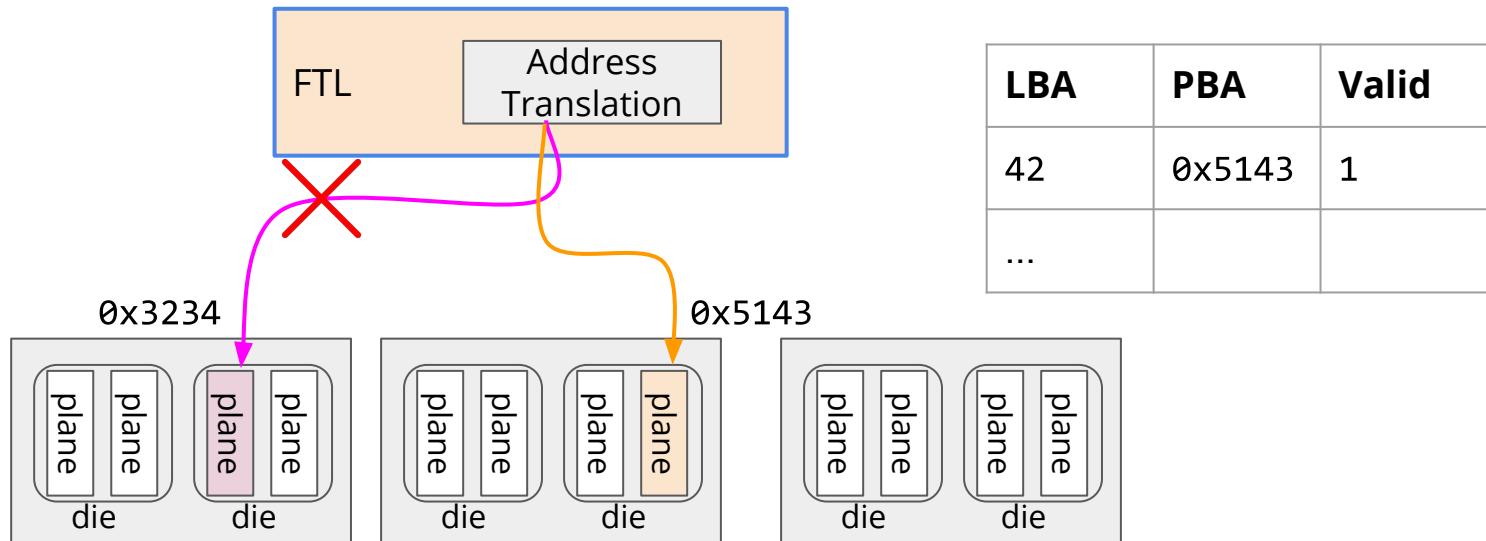
# What Happens when you overwrite a LBA



# The basic FTL address translation

Can give a way to rewrite the same address again and again

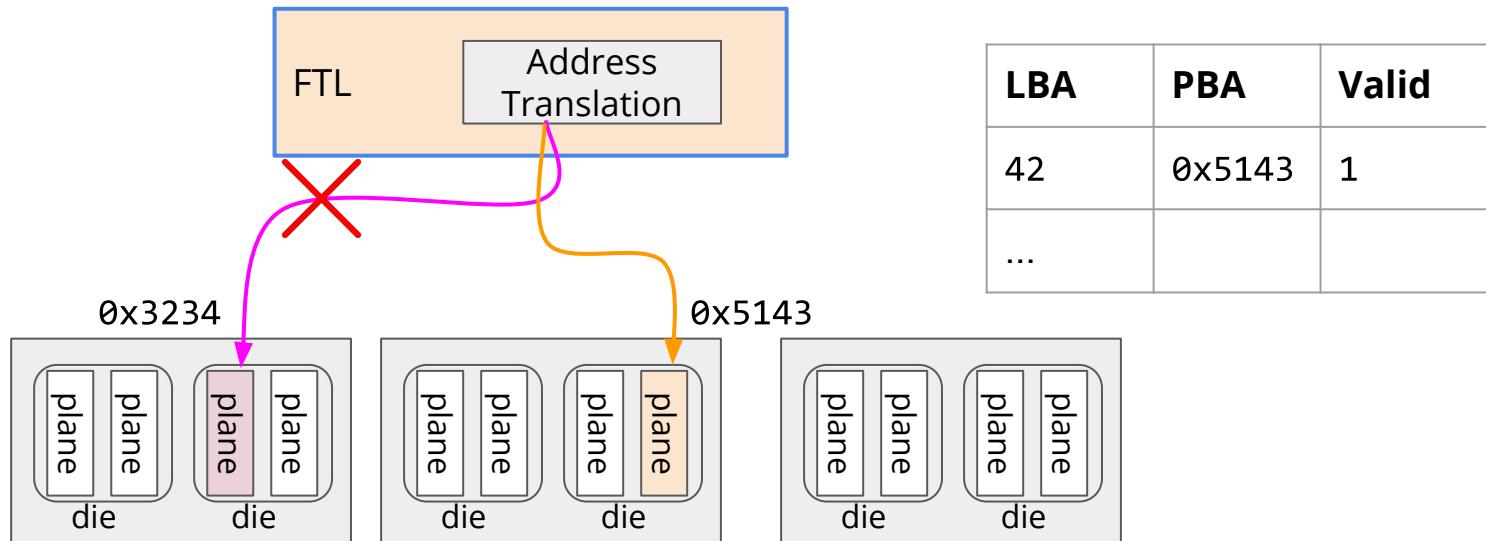
- *Is that all?*



# The basic FTL address translation

Can give a way to rewrite the same address again and again

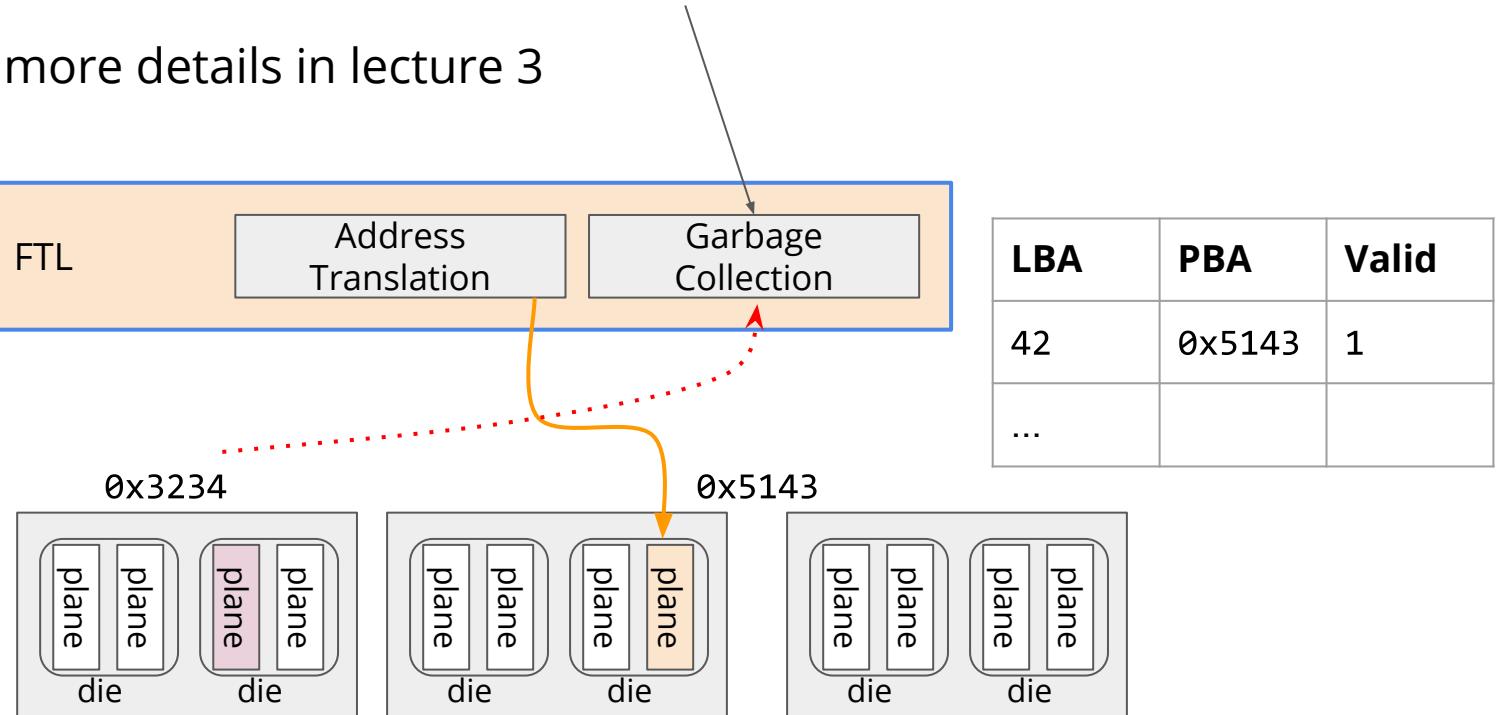
- *What happened with the 0x3234 page?*
- *What if there are no free ready blocks available?*



# Garbage collection (GC)

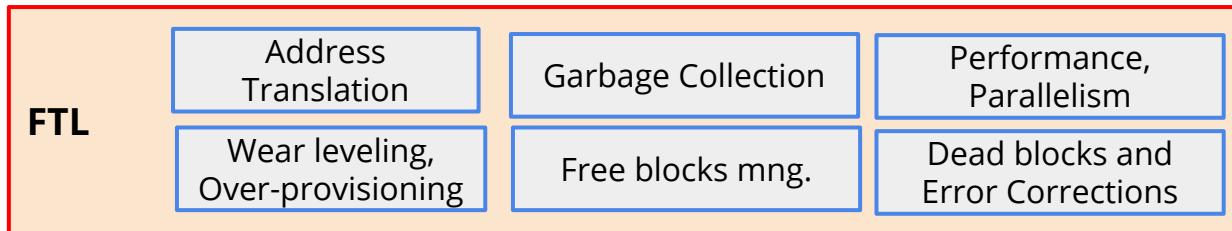
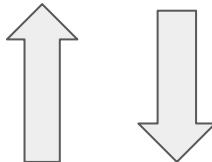
The old blocks need to keep track of and **garbage collected** for erasing

We will see more details in lecture 3

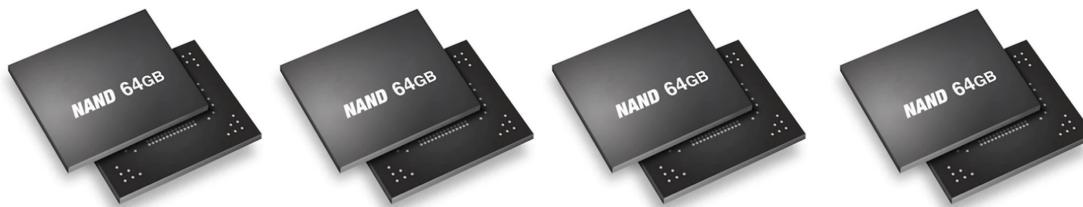


# Flash Translation Layer does a lot more ...

*Reads writes from the software (OS, application, fs)*

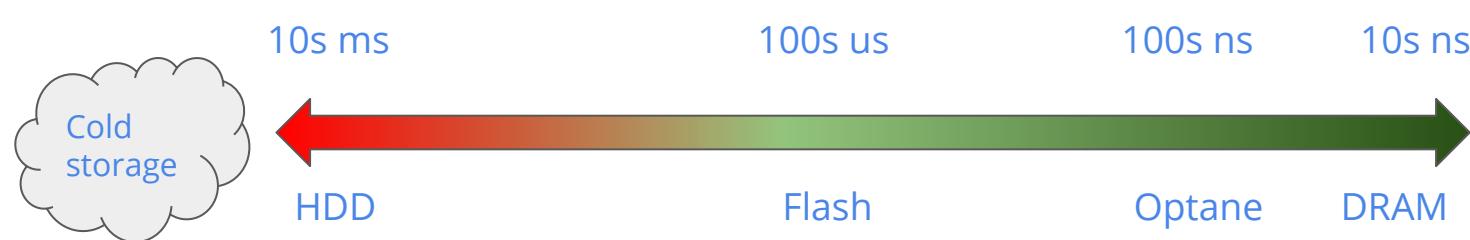
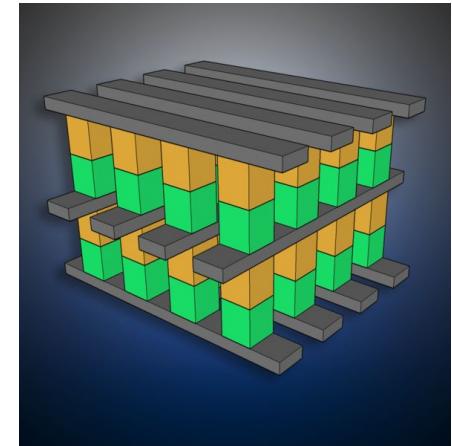


***FTL is the secret sauce***



# Optane: A new class of NVM technology

- Jointly developed by Intel and Micron
- Uses a new type of material
  - Not publicly disclosed - but thought to be a resistive bulk material
  - Hard to guess internals, but we are trying ;)
- Much faster than flash, and can be packaged as byte-addressable memory
  - Latencies in 100s of nanoseconds
  - Bandwidths in 10s of GBytes/sec



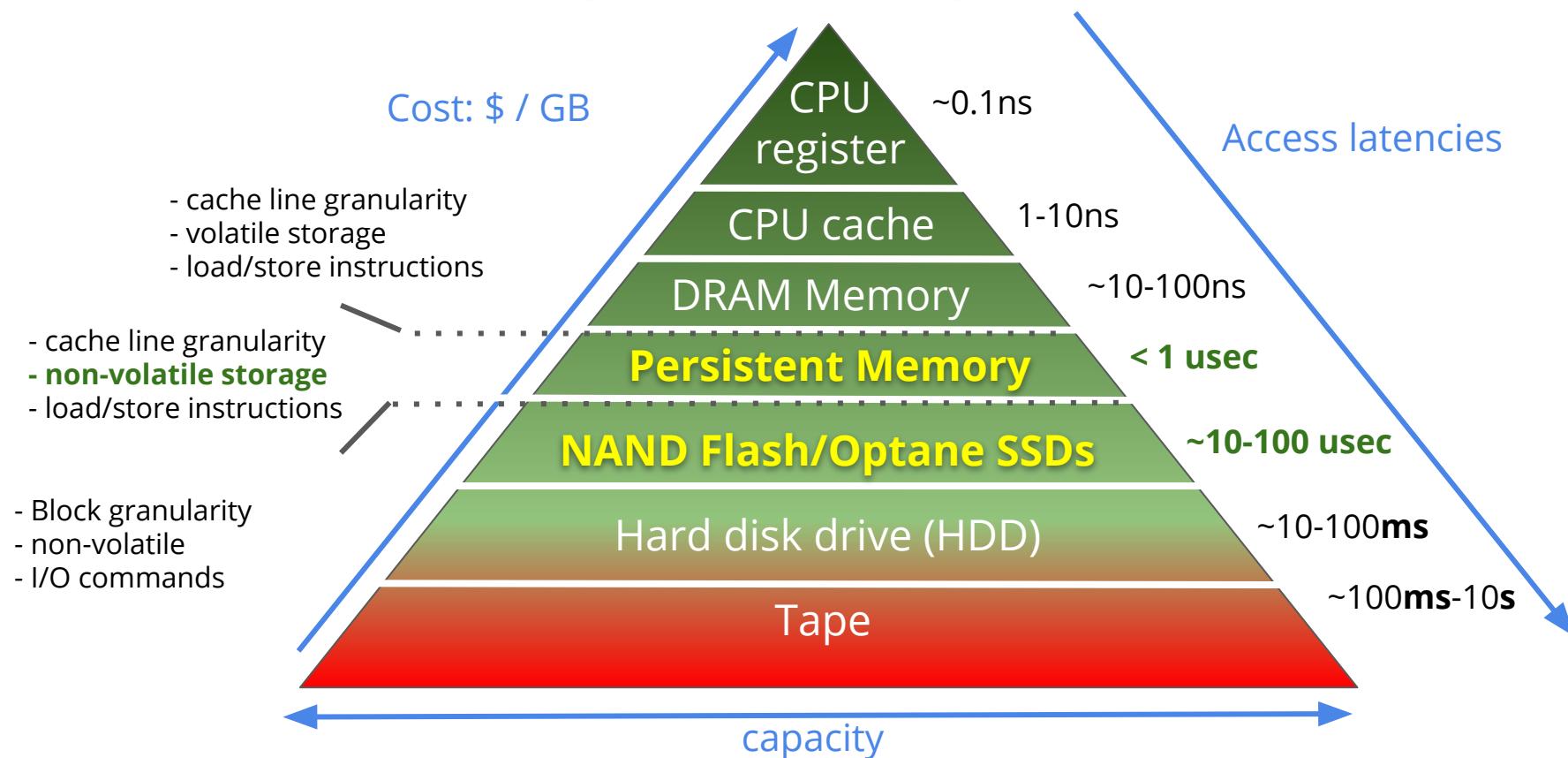
# Optane: Memory and storage form factors

Today available in storage as well as memory form factor

- Optane DIMMs (memory/cache) : 128GB, 256GB and 512GB DIMMs
  - Do direct load/store from the CPU
  - Treat it like a persistent memory
- Optane SSDs (as storage /dev/optane)
  - Make a file system, and do file I/O, mmap read and write



# The (new) triangle of storage hierarchy



# Funny enough ...the Architecture Bible

## ***Computer Architecture: A Quantitative Approach*** ***John Hennessy and David Patterson***

5.4	Protection: Virtual Memory and Virtual Machines	315
5.5	Crosscutting Issues: The Design of Memory Hierarchies	324
5.6	Putting It All Together: AMD Opteron Memory Hierarchy	326
5.7	Fallacies and Pitfalls	335
5.8	Concluding Remarks	341
5.9	Historical Perspective and References	342
	Case Studies with Exercises by Norman P. Jouppi	342

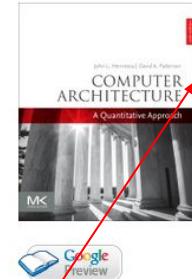
### **Chapter 6 Storage Systems**

6.1	Introduction	358
6.2	Advanced Topics in Disk Storage	358
6.3	Definition and Examples of Real Faults and Failures	366
6.4	I/O Performance, Reliability Measures, and Benchmarks	371
6.5	A Little Queuing Theory	379
6.6	Crosscutting Issues	390
6.7	Designing and Evaluating an I/O System—The Internet Archive Cluster	392
6.8	Putting It All Together: NetApp FAS6000 Filer	397
6.9	Fallacies and Pitfalls	399
6.10	Concluding Remarks	403
6.11	Historical Perspective and References	404
	Case Studies with Exercises by Andrea C. Arpaci-Dusseau and Remzi H. Arpaci-Dusseau	404

### **Appendix A Pipelining: Basic and Intermediate Concepts**

A.1	Introduction
-----	--------------

**5th edition**



Computer Architecture

6th Edition

A Quantitative Approach

Write a review

Authors: John Hennessy, David Patterson

eBook ISBN: 9780128119068

Paperback ISBN: 9780128119051

Imprint: Morgan Kaufmann

Published Date: 23rd November 2017

Page Count: 936

[View all volumes in this series: The Morgan Kaufmann Series ...](#)

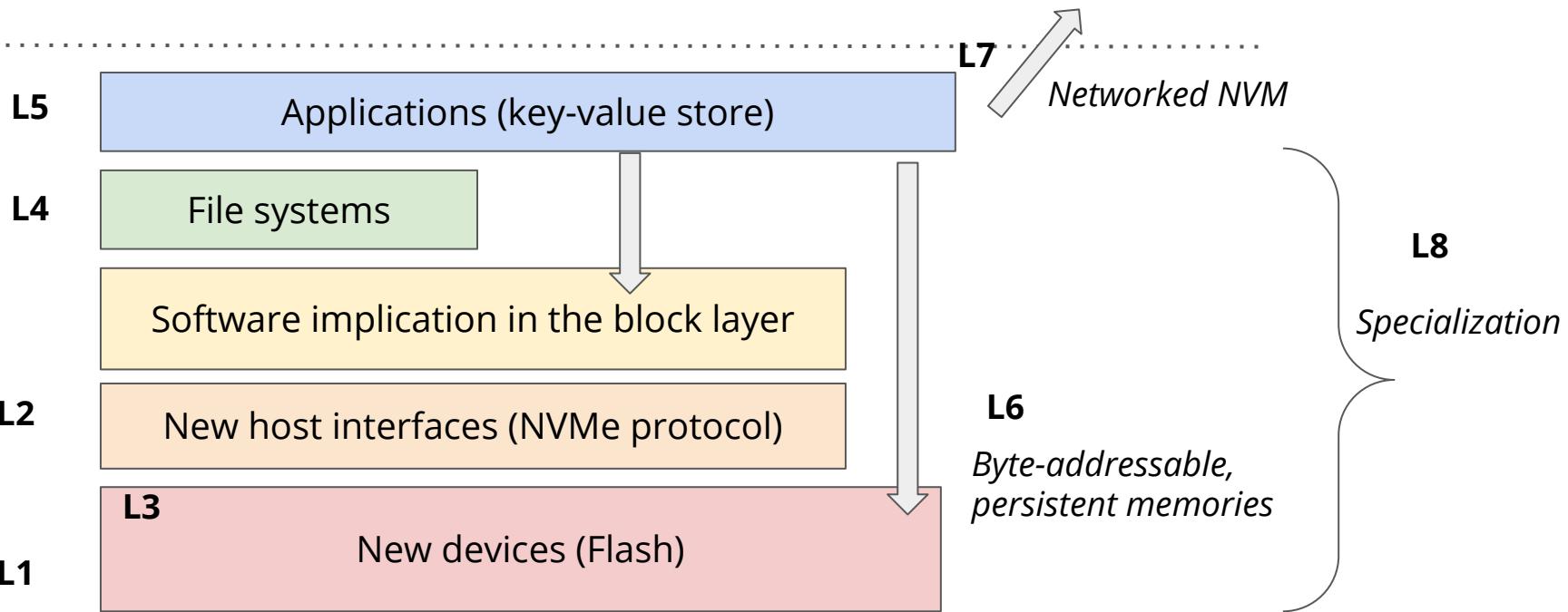
### Table of Contents

#### Printed Text

1. Fundamentals of Quantitative Design and Analysis
2. Memory Hierarchy Design
3. Instruction-Level Parallelism and Its Exploitation
4. Data-Level Parallelism in Vector, SIMD, and GPU Architectures
5. Multiprocessors and Thread-Level Parallelism
- 6. The Warehouse-Scale Computer**
7. Domain Specific Architectures
  - A. Instruction Set Principles
  - B. Review of Memory Hierarchy
  - C. Pipelining: Basic and Intermediate Concepts

# The layered approach in the lectures

Distributed Systems L9-L10

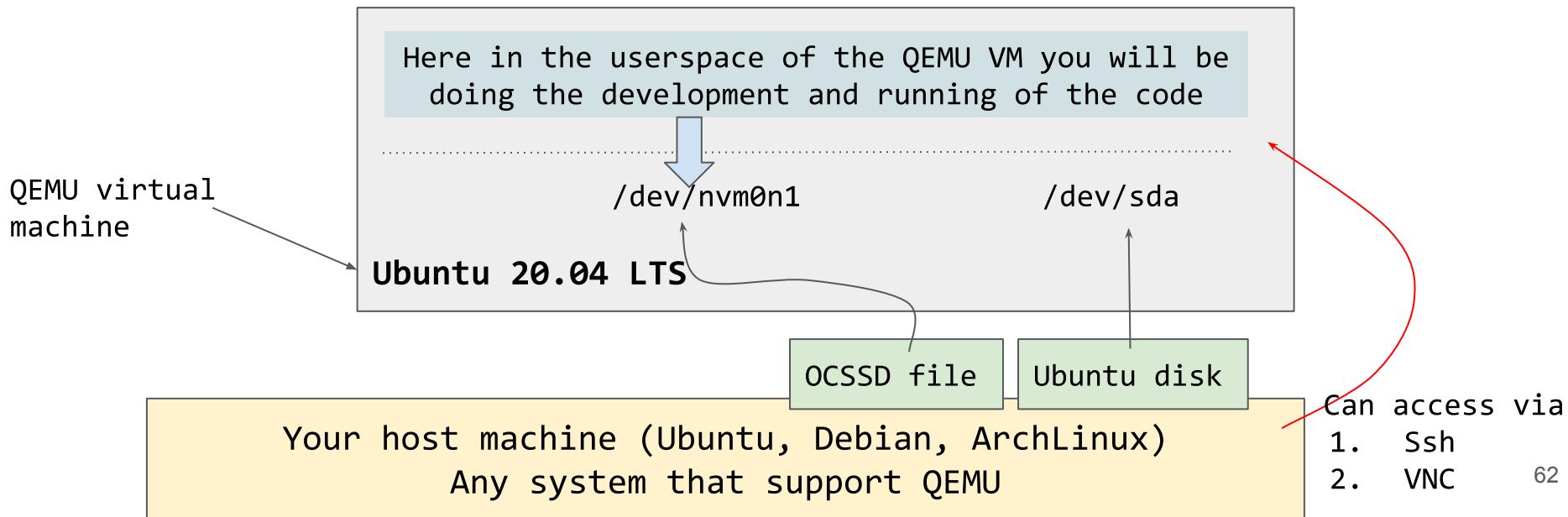


# Syllabus outline

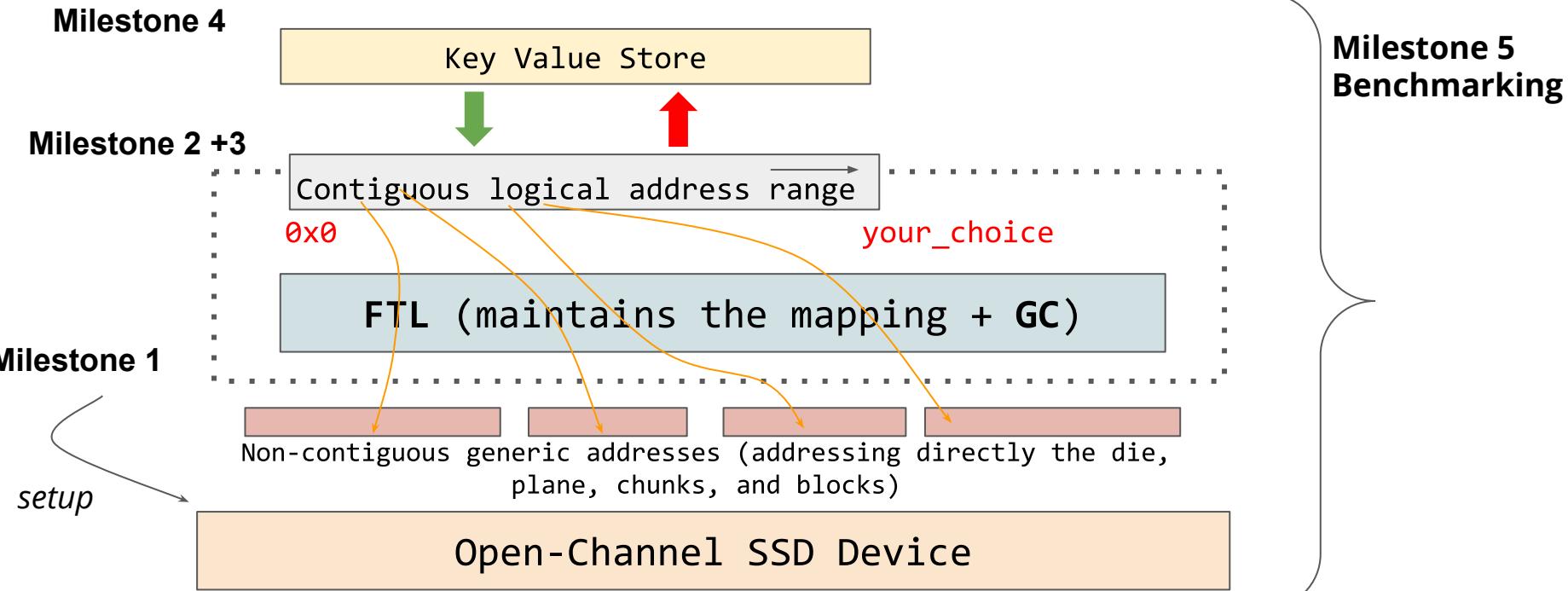
1. **Welcome and introduction to NVM (today)** 
2. Host interfacing and software implications
3. Flash Translation Layer (FTL) and Garbage Collection (GC)
4. NVM Block Storage File systems
5. NVM Block Storage Key-Value Stores
6. Emerging Byte-addressable Storage
7. Networked NVM Storage
8. Trends: Specialization and Programmability
9. Distributed Storage / Systems - I
10. Distributed Storage / Systems - II

# Project overview

You will be doing coding to understand internals and challenges associated with modern storage stacks (for now just “functional” not “performance”)



# Project milestones



# On Canvas - read the project handbook



## Storage Systems (StoSys) Course Project Handbook

(XM\_0092)

P2 (November - December), 2020

Version: 1.1

Animesh Trivedi ([a.trivedi@vu.nl](mailto:a.trivedi@vu.nl))  
Giulia Frascaria ([g.frascaria@student.vu.nl](mailto:g.frascaria@student.vu.nl))  
Sacheendra Talluri ([s.talluri@vu.nl](mailto:s.talluri@vu.nl))

*If you find  
mistakes do  
let us know*

# Warning

This is an **experimental** course

- You are a part of the experiment
- We will build upon your feedback

This is a **coding-heavy** course

- If you have not done C/C++ programming this may be a tough course
  - *Use of structs, pointers, file I/O, thread synchronization, locks*
- Start coding early, there will be plenty of surprises (but fun as well)

*There's a relatively little flexibility with the deadlines (as they all dependent on each other)*



# Recap: From this lecture

1. **Understand: The difference between storage and memory is blurring**
2. You should know what is NVM
3. You should know flash internals, and how they differ from HDDs
4. What are SLC, MLC and TLC flash cells, what do they offer
5. What is an FTL (the idea of internal mapping)
6. What are LBAs and PBAs
7. What is Optane memory
8. General performance ballpark numbers of NVM technologies

# Lecture Reading List

1. Michael Cornwell. 2012. Anatomy of a Solid-state Drive: While the ubiquitous SSD shares many features with the hard-disk drive, under the surface they are completely different. Queue 10, 10 (October 2012), 30–36. DOI:<https://doi.org/10.1145/2381996.2385276>
2. Design Tradeoffs for SSD Performance,  
[https://www.usenix.org/legacy/events/usenix08/tech/full\\_papers/agrawal/agrawal.pdf](https://www.usenix.org/legacy/events/usenix08/tech/full_papers/agrawal/agrawal.pdf)
3. Operating System Implications of Fast, Cheap, Non-Volatile Memory,  
[https://www.usenix.org/legacy/events/hotos11/tech/final\\_files/Bailey.pdf](https://www.usenix.org/legacy/events/hotos11/tech/final_files/Bailey.pdf)
4. Flash-based SSDs, Operating Systems: Three Easy Pieces,  
<http://pages.cs.wisc.edu/~remzi/OSTEP/file-ssd.pdf>
5. Nand flash solid state storage: Performance and capability, an in-depth look  
[https://www.snia.org/sites/default/education/tutorials/2009/spring/solid/JonathanThatcher\\_NandFlash\\_SSS\\_PerformanceV10-nc.pdf](https://www.snia.org/sites/default/education/tutorials/2009/spring/solid/JonathanThatcher_NandFlash_SSS_PerformanceV10-nc.pdf)
6. Data Storage Research Vision 2025, <https://dl.acm.org/doi/book/10.5555/3316807>  
<https://par.nsf.gov/servlets/purl/10086429>