Not Your Grandpa's SSD:
The Era of Co-Designed Storage Devices

Alberto Lerner and Philippe Bonnet
SIGMOD'21 – Xi'an, Shaanxi, China

Motivation

- Flash memory provides better cost/performance than RAM for DB systems.
- Solid-State Drives (SSDs) are the dominant way to package Flash memory.

=> SSDs are crucial for modern DB systems, but ...

… choosing an SSD model for given workload is not trivial.

- There are so many of them. What are the design choices?
- How to navigate the design space?
- How is the design space evolving?

… designing SSDs for a given workload is now possible and desirable.

- How to design workload-specific SSDs? How to co-design DBMS and SSD?
- DBMS/SSD co-design requires the federation of expertise from Flash, Hardware, Operating System, Storage and DB experts.

This tutorial is our attempt at engaging the DB community!

Why Should You Care?

- You use SSDs in your experiments.
  - You want to better understand a device's performance level and when it fails to deliver.
- You design SSD-Based DBMS.
  - You want to better understand advanced storage protocols and devices, and be able to tell when and how to resort to them.
- You are willing to be convinced that DBMS/SSD co-design is worth your time and energy.
  - You may want to hear about open problems and existing approaches for building such protocols, devices, or both.

What This Tutorial is Not

- This tutorial is not a complete coverage of the domain.
  - The 1.5 hours format constraints both breadth and depth of our presentation.
  - We present our curated survey of the domain and refer to resources that dive in specific topics.
- This tutorial is not neutral.
  - We have been pushing for a radical evolution of SSDs and for DBMS/SSD co-design. We have significant bias.
  - We emphasize what we see as the main take-aways throughout this tutorial.
- This tutorial will not settle current open questions.
  - But we introduce them and present our views on current approaches and possible areas of further study.

Who We Are

Alberto Lerner

eXascale Infolab
University of Fribourg
Switzerland

DB2

NetAccel

Google's BigTable

Instrumented
Cosmos+

Philippe Bonnet

Data-Intensive Systems and Applications Lab
IT University of Copenhagen
Denmark

mq-blk

GeckoFTL

OCSSD

Ox

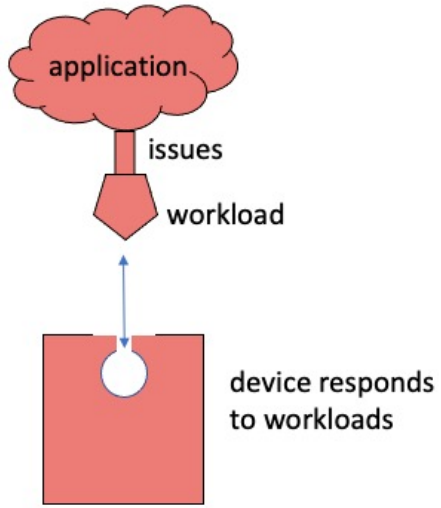
Historical Perspective

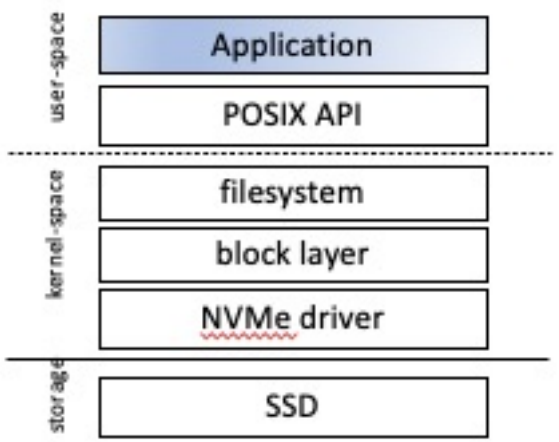
- Flash invention.
- First SSDs, compatibility.
- Diversification, Speed and NVMe.
- Entering the **era of co-Design.**
  - NVMe carries built-in evolution mechanisms.
  - Increasing application control is possible.
  - Speed and Co-designing opportunities.
    - Through workload specialization.
    - Computational and Programmable devices.

Moving away from the drop-in replacement role and into SSDs becoming an active component of the architecture.

Architectural Perspective

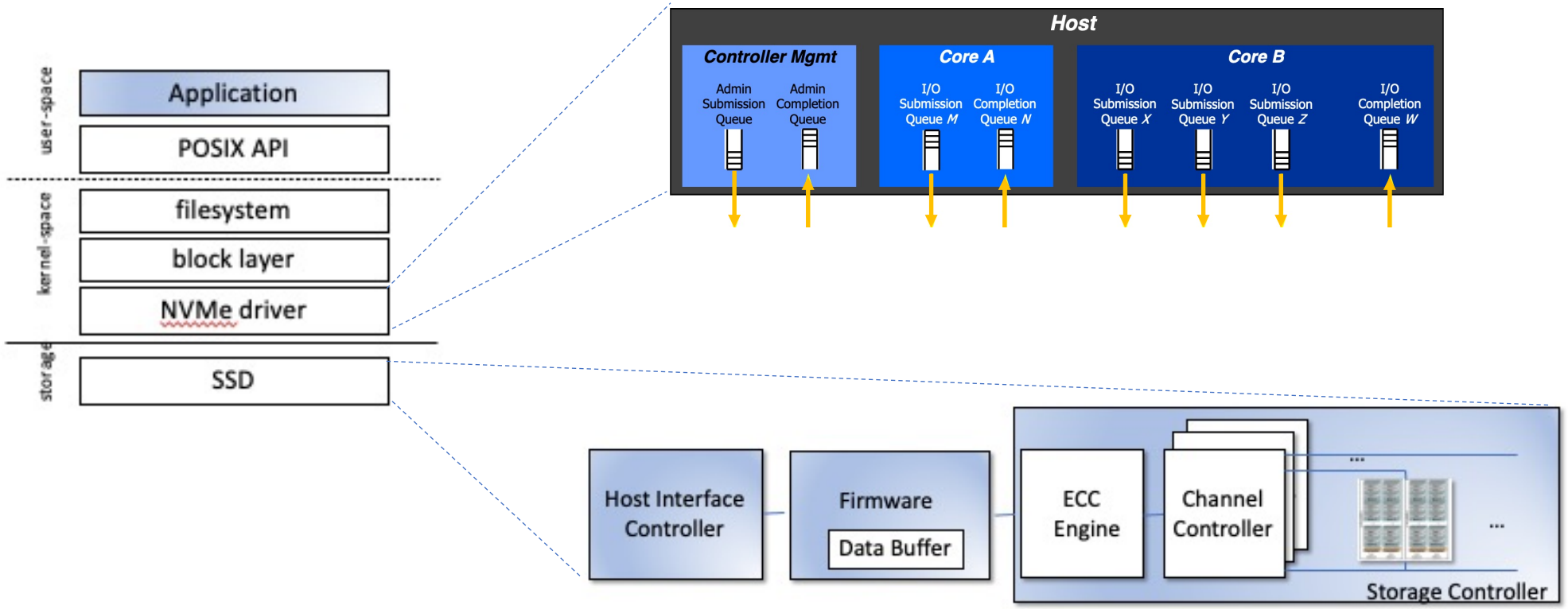
- We will be discussing how data and control move between applications and devices and will look at systems and logical view to do so





Programmer's Perspective

- But we will also talk in detail about the APIs and general *mechanics* involved in moving data around.

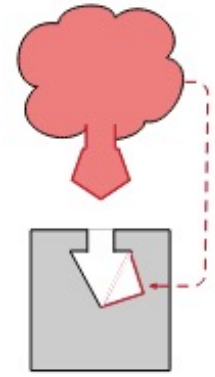


The Tutorial Message In One Slide

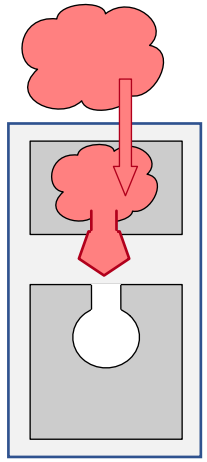
- “Grandpa” has been writing software that adapts to historical device restrictions that make little sense now.
- Some mechanisms are emerging that allow us to
  - Control the device “externally” by sending semantically richer commands
    - Higher abstractions get give control at different levels
      - OCSSD – control PUs
      - Streams – tagging of workloads
  - Control the device “internally” by interacting directly with subsystem
    - Replace a policy (e.g., scheduling or caching)
- We can use the mechanism in new device designs.

The Path to Co-Design

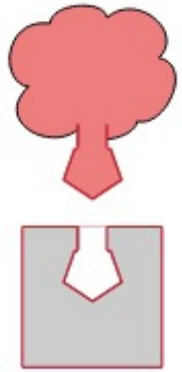
- Speaking of designs, we can think of at least four new categories of devices with different co-design approaches:
  - Configurable: devices “levers” that can in be adjusted in the field.
  - Computational: devices in which application logic be embedded
  - Programmable: devices with sub-systems controlled by the user.
  - Codesigned: the holy grail.



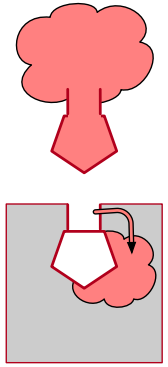
Configure the device’s behavior



Bypass API by pushing the Computational down



Tailor device behavior via Programming



Co-Designed

Tutorial Structure

- Interfaces Sections
  - P2 - Internal
  - P3 - External
- Co-designing Sections
  - P4 -Computational and Programmable Devices
  - P5 - Examples of devices
- Tooling Sections
  - P6 - Continuum
  - P7 - SPD tools: simulation, prototyping, and development

Internal Interfaces

Alberto Lerner

Tutorial Structure

- Interfaces
  - P2 - Internal
  - P3 - External
- Co-designing
  - P4 -Computational and Programmable Devices
  - P5 - Examples of devices
- Tooling
  - P6 - Continuum
  - P7 - SPD tools: simulation, prototyping, and development

From NAND-Flash to an SSD

- It is common to hear about NAND-Flash's peculiarities:
  - Cannot be overwritten without prior erasing;
  - Read and Program (write) operations take different times, etc.

These (and other) peculiarities are
the reason SSDs are built the way they are.

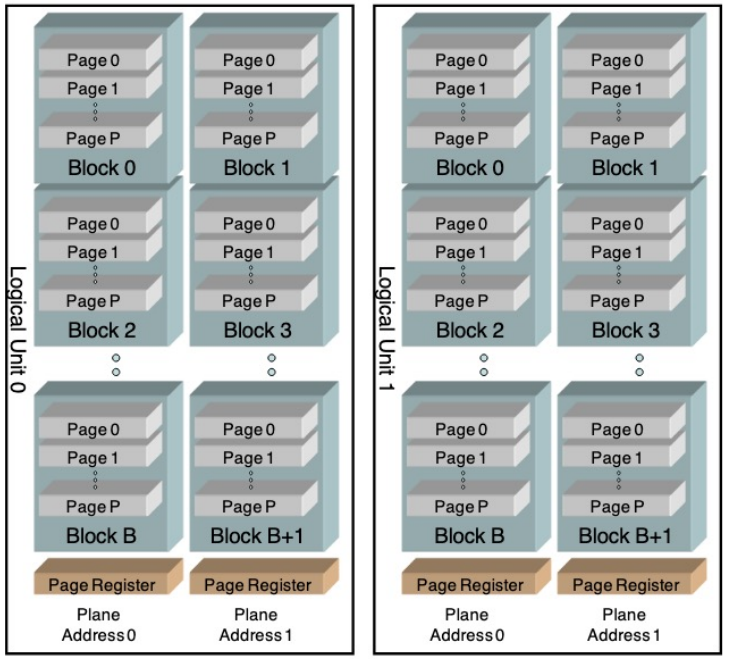
- SSDs have internal sub-systems, each contributing to (or hindering) performance.
- The sub-systems interact through interfaces that are not visible to applications.
- They implement a Data Path from user application to NAND-Flash.

The understanding of the Data Path allows for a more systematic
SSD performance (and design trade-offs) analysis.

Flash Data Path

- Flash is organized in ``Logical Units'' that independently serve a given number of pages/blocks.
- Data gets in and out of a LUN through the Data Register(s) (and optionally Cache Registers).
- These operations can be done with different throughput/latency trade-offs.

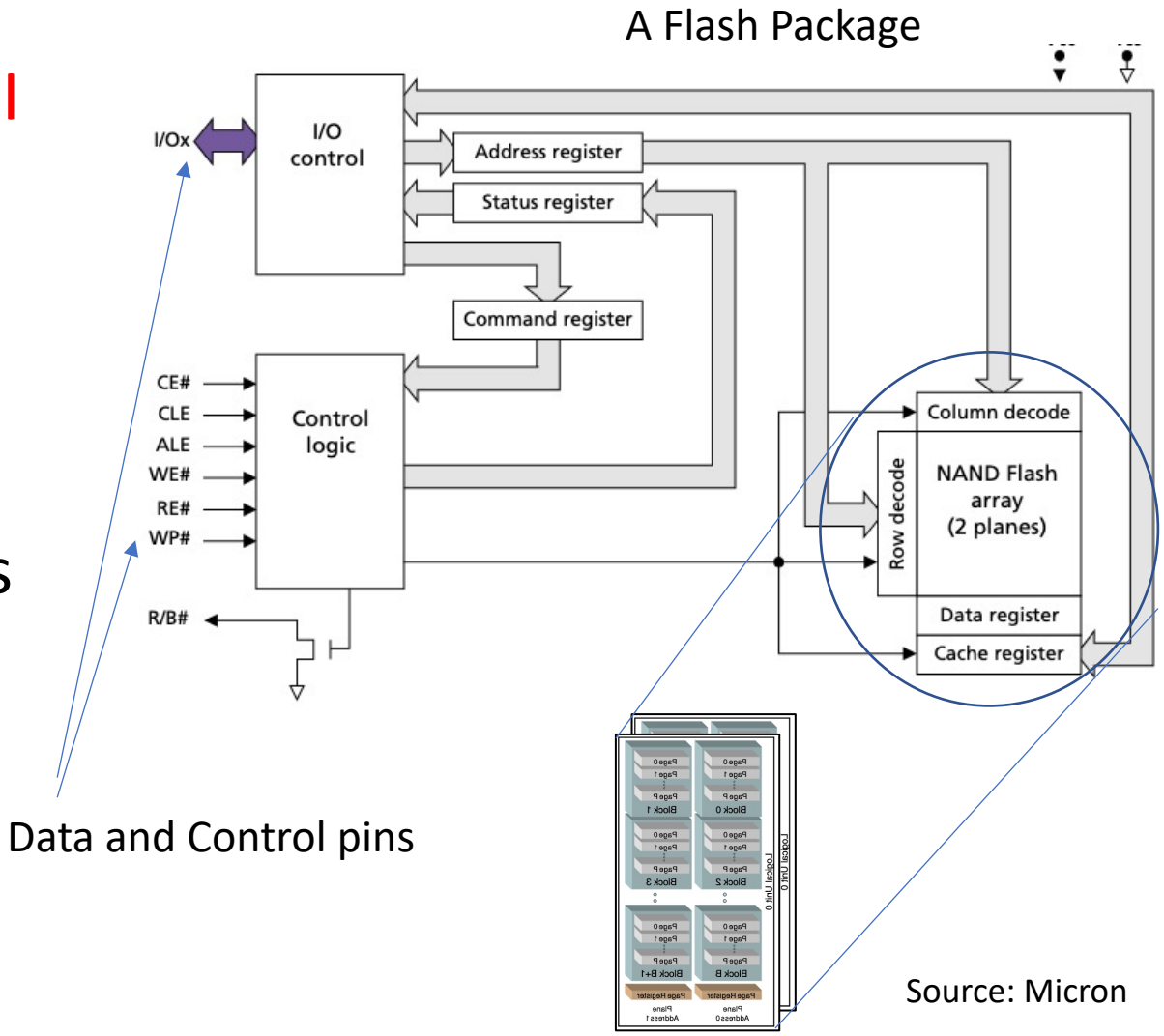
(Plug for the next module: the abstraction that sits atop of the process can expose or control the tradeoffs.)



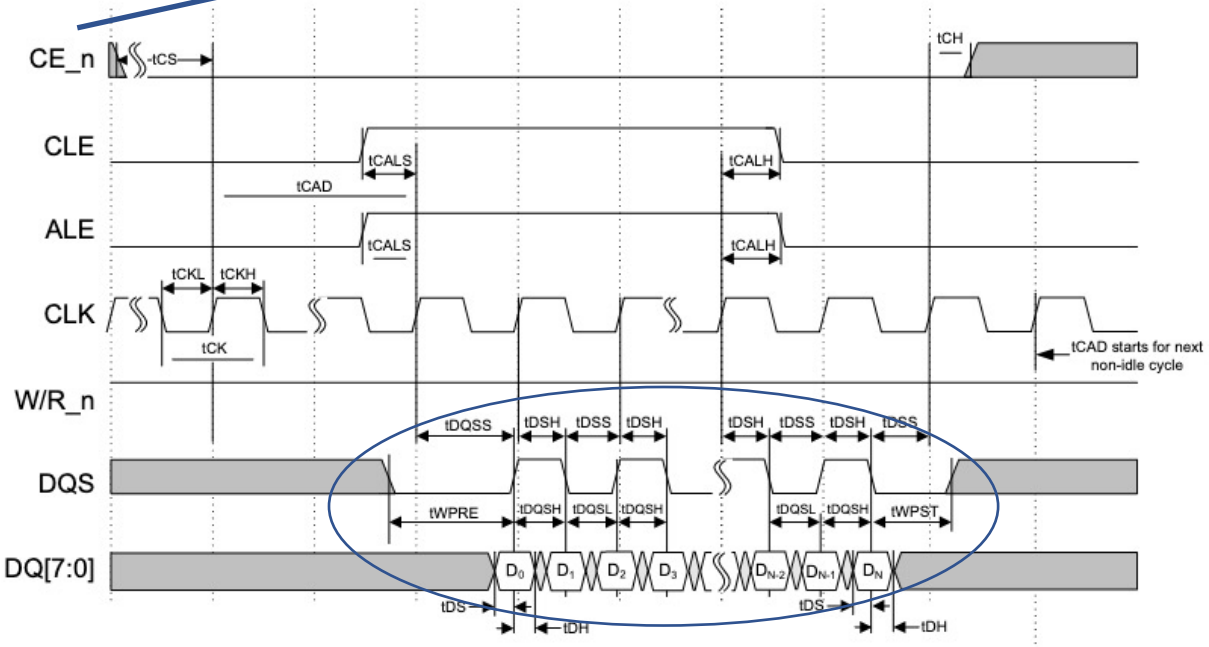
Source: ONFi Specs

ONFI and Data Movement

- There's a standard called ONFI that governs:
  - The physical interface to Flash packages at the pin level;
  - How to move data and request services.
- A flash controller orchestrates the data movement through agreed upon protocols.

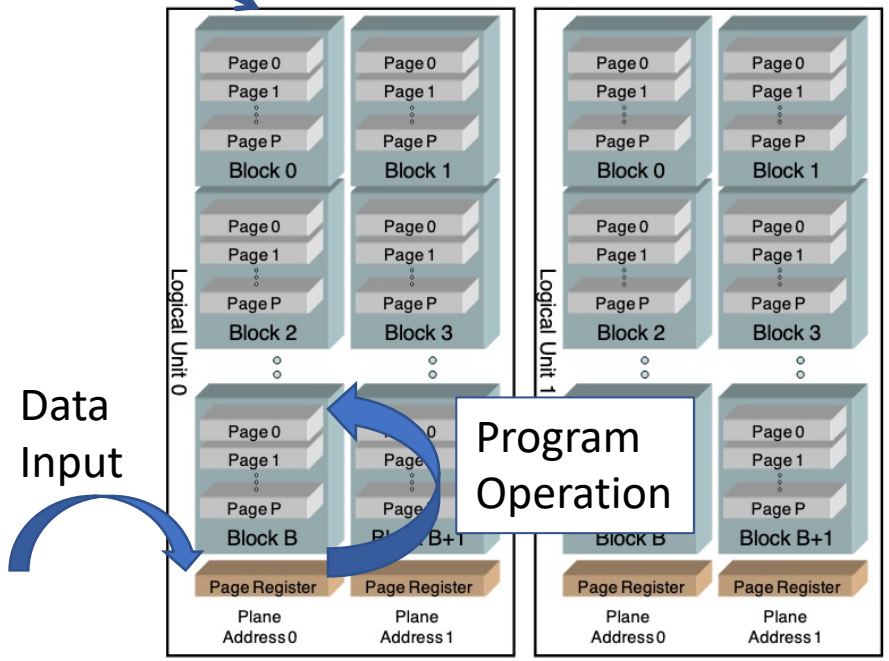


Moving Data In and Program Operation



Provides the data that to input to Page Register
(The Program operation itself is not here, just moving data in.)

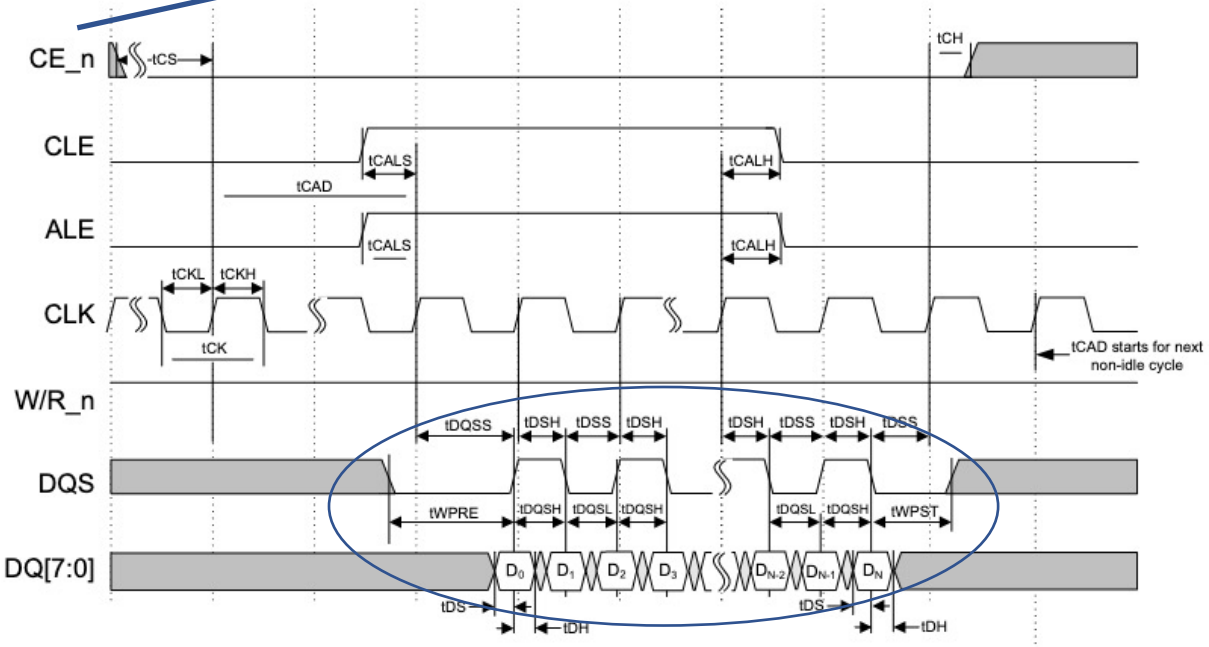
Designates which LUN will perform the operation



Delay between data input and program operation.

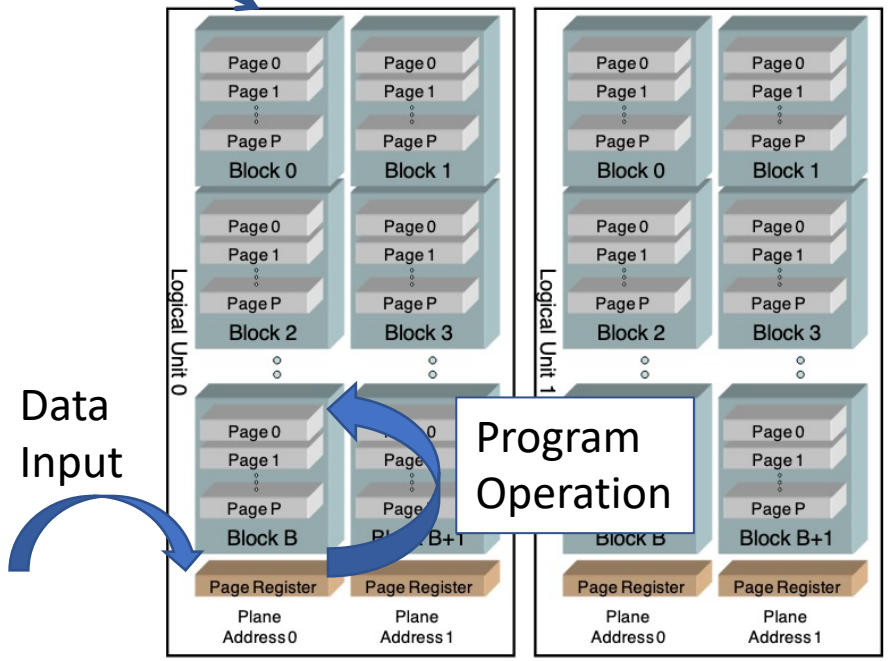
Source: ONFi Specs

Moving Data In and Program Operation



Provides the data that to input to Page Register
(The Program operation itself is not here, just moving data in.)

Designates which LUN will perform the operation

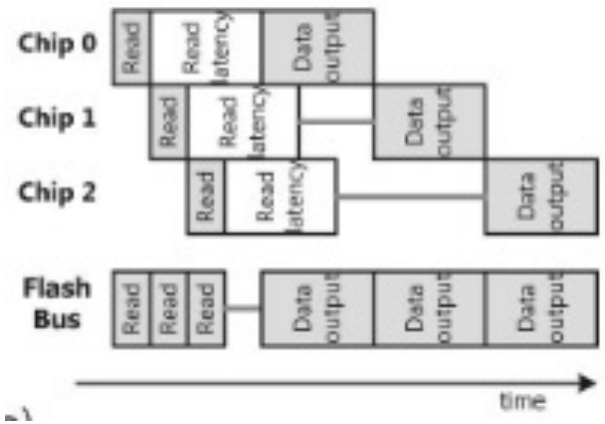


Delay between data input and program operation.

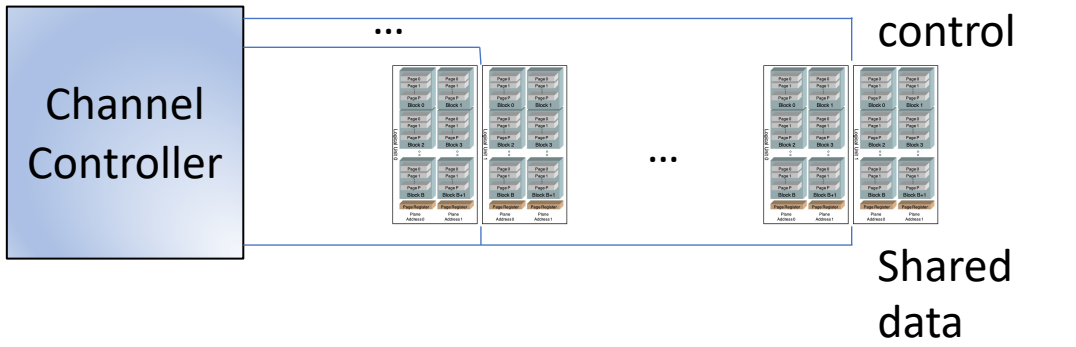
Source: ONFi Specs

“Channel” Organization

- While a LUN is performing a flash operation (to or from the Page Register), its data pins are not used.
- Device share those pins across many LUNs through “channels” but LUNs are still controlled independently.
- The SSD controller tries to maximize the time the channel is carrying data by interleaving flash operations.



Source: O3 paper

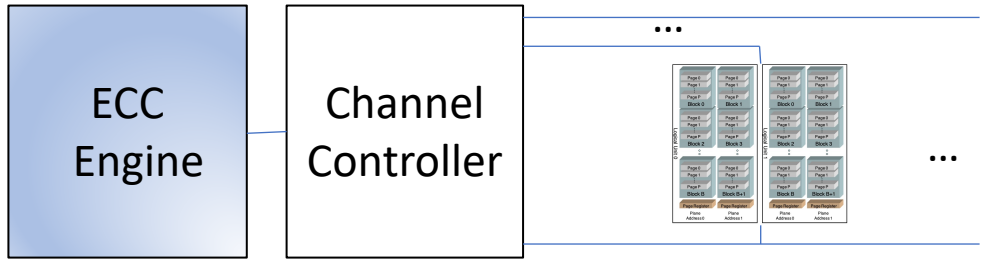


A channel hides the LUNs latency and aggregates their throughput.

Error Correction

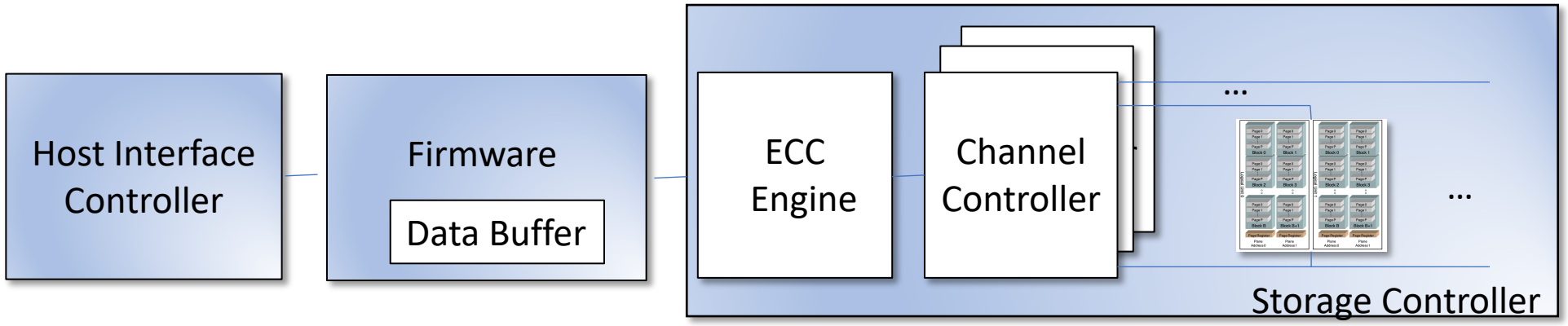
- Because Flash is extremely sensitive, it requires:
  - Scrambling data to roughly contain a balanced number of 1's and 0's;
  - Error Correction.
- These mechanisms are not entirely* supported by the LUNs; need to be built around it.

- Two main ECC techniques
  - BCH
  - LDPC



- How much and how a Flash Package deals with errors is not public information. This is one of the many reasons why, despite ONFi, different flash memory are not interchangeable.

Device Architecture



HIC

- Implements the NVMe controller.
- Performs data transfers in and out of the device.

Firmware

- Implements the FTL (page mapping, wear leveling, and GC)
- Not only FTL, but also:
  - Low-level scheduling;
  - DMA control, etc.

Storage Controller

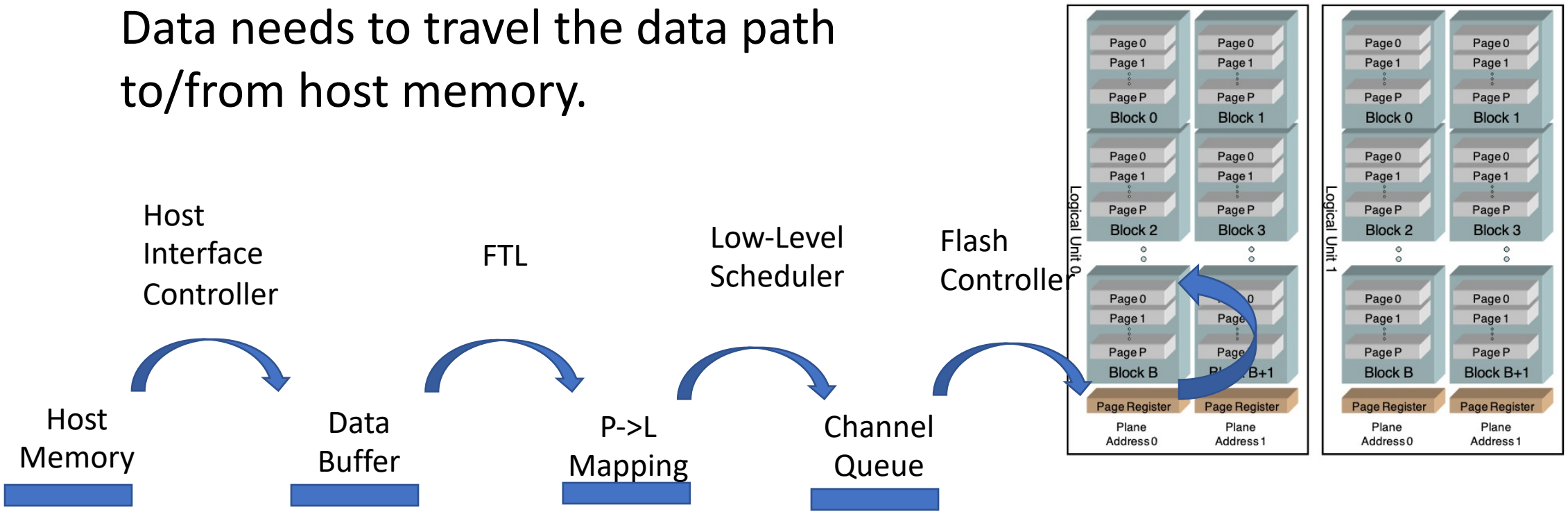
- Interfaces with Flash packages.
- Performs scrambling and ECC.

SSD Basics Mini Survey

- Agrawal et al., “Design Tradeoffs for SSD Performance,” USENIX ATC’08.
- Cai et al., “Error characterization, mitigation, and recovery in flash-memory-based solid-state drives,” Proceedings of the IEEE 105(9), September 2017.
- Dirik and Jacob, “The performance of PC solid-state disks (SSDs) as a function of bandwidth, concurrency, device architecture, and system organization,” ISCA’09.
- Grupp et al., “Characterizing Flash Memory: Anomalies, Observations, and Applications,” MICRO’09.
- Hu et al. “Performance Impact and Interplay of SSD Parallelism through Advanced Commands, Allocation Strategy and Data Granularity,” ICS’11.
- Kim, Choi, and Min, “Design Tradeoffs for SSD Reliability,” FAST’19.
- Micheloni, Crippa, and Marelli, “Inside NAND Flash Memories”, Springer, 2010.
- Nam et al., “Ozone (O3): An Out-of-Order Flash Memory Controller Architecture,” IEEE ToC 60(5), May, 2011.

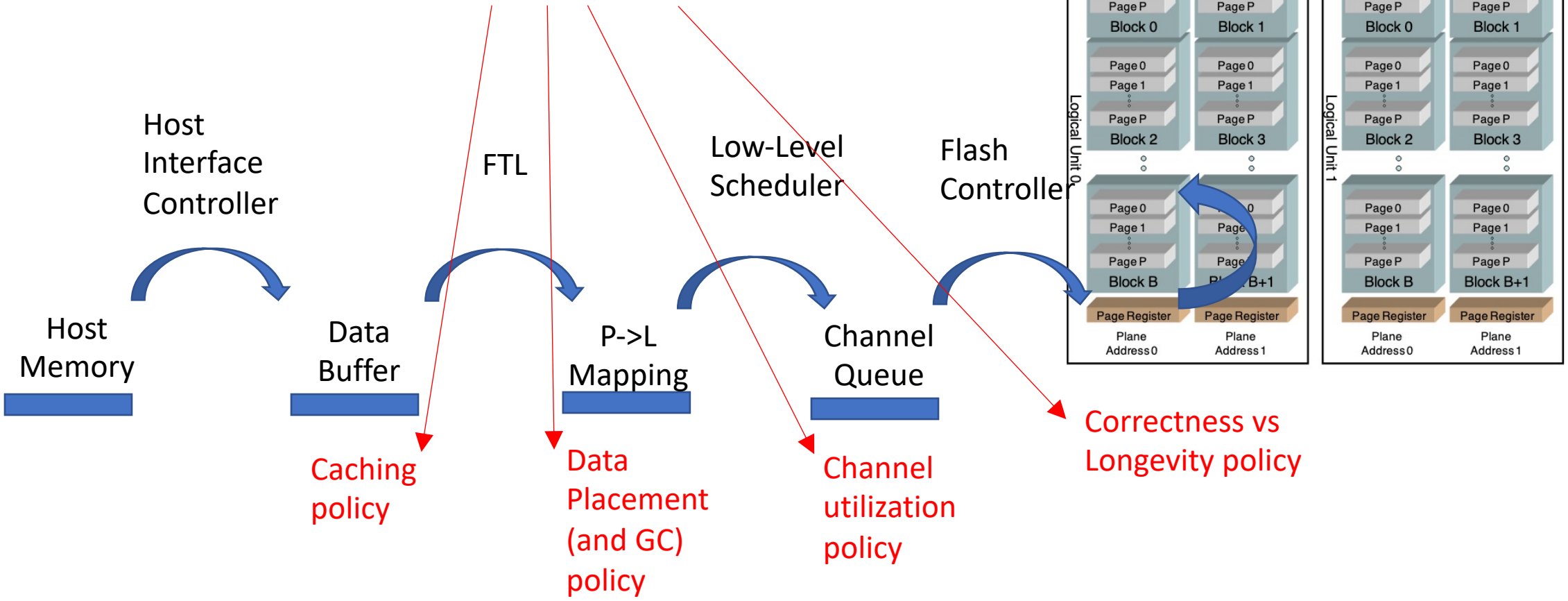
Anatomy of an NVMe Command

Data needs to travel the data path to/from host memory.



Decisions, Decisions, Decisions

The decisions can be more or less effective, depending on the workload.



Speed Bumps on SSDs

- GC is slow but it is far from the only performance pitfall.
- Question: can an SSD that is servicing read operations only be slow? Yes!
  - Interference between workloads.
  - Not enough parallelism opportunities.
  - Other limitations on device design.

Generic devices are designed for non-descript applications, but many devices are used for very well understood workloads.

Why not tune the device for these workloads?

Database Workloads

- DB systems produces few but varied workloads:
  - transaction log,
  - checkpoints,
  - buffer manager reads/write,
  - data loading,
  - external sort,
  - “spills” from other operators,
  - OLAP query scans, etc.

We can characterize these workloads!

(But this is an area that needs more work.)

Workload Characterization

	Tx Log	Checkpoint	…
Sequential or random access	Seq	Seq	
Read/write/mixed operations	Write (but for recovery)	Write (but for recovery)	
Queue depth (# of outstanding operations)	1	1 or more per core	
Size of an operation	Page	Multiple Pages	
Other relevant characteristics			
Circularity	yes	no	
Latency Sensitiveness	yes	no	
Bursty-ness	yes	no	
Priority	high (user perceived)	depends on recovery goals	

Then we can reason about policies: Checkpoint is much more parallel than and can overpower Tx Log if proper scheduling is not in place.

Database Workload Characterization Mini-Survey

- Chen et al., “TPC-E vs. TPC-C: Characterizing the New TPC-E Benchmark via an I/O Comparison Study,” SIGMOD Record 39(3), September, 2010.
- Park, “Characterizing Database Workloads via a Comprehensive I/O Analysis,” MSc. Thesis, Seoul National University, 2018.
- Tuan, Cheon, and Won, “On the IO characteristics of the SQLite Transactions,” MobileSoft’16.
- Yang et al., “Don’t stack your Log on my Log,” INFLOW’14.

Interface #1 - Caching

- Data buffer in the Firmware is finite; how to better utilize it?
- Without knowing anything about the workload: LRU to pick a victim.
- Workload information can help with:
  - Determining the “utility” of different pages, not just frequency of access;
  - Establishing different policies than LRU when necessary;
  - Allow a more active caching layer, for instance:
    - Issuing pre-fetches, or
    - ML for predicting future accesses.

Interface #2 – Data Placement

- Once again, without any information, pick the next available LUN to write a page.
- Severe problems arise from this lack of information:
  - Mixing pages with different lifespans makes GC harder.
- Workload information can help with:
  - Consider how the pages a workload writes are going to be read.
    - Recall the case of the slow scan? (Hotspot on a given LUN.)

Interface #3 – Channel Utilization

- Hiding latency and maximizing channel bandwidth my lead to lack of fairness.
  - We discussed the conflict between a Transaction Log workload and a Checkpoint one.
- Perhaps the metric to optimize here should be achieving a workload's goal.
  - A low-level scheduler can keep putting Transaction Log writes in front of Checkpoints, and scheduling the latter in times when the former is not bursty.

Interface #4 – ECC Strength

- Devices assume that all data that is read should be error free.
  - ECC introduces latency to every operation.
- Certain applications may tolerate a controlled degree of error.
  - For example, bit flips in large images.

SSD Policies Examples

1. Caching
   - Park et al., “CFLRU: A Replacement Algorithm for Flash Memory,” CASES’06.
   - Kim and Ahn, “BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage,” FAST’08.
2. Placement
   - Kang et al., “The Multi-streamed Solid-State Drive,” HotStorage’14.
3. Channel Utilization
   - Wu and Re, “Reducing SSD Read Latency via NAND Flash Program and Erase Suspension,” FAST’12.
4. Longevity vs Correctness
   - Jimenez, Novo, and Ienne, “Wear Unleveling: Improving NAND Flash Lifetime by Balancing Page Endurance,” FAST’14.

Bonus: FTL Mini-Survey

FTLs combines different policies together

- Chung at al., “A survey of Flash Translation Layer,” Elsevier Journal of Systems Architecture 55, 2009.
- Ma, Feng, and Li, “A Survey of Address Translation Technologies for Flash Memories,” ACM Computing Surveys, 46(3), January, 2014.
- Shin et al., “FTL Design Exploration in ReconfigurableHigh-Performance SSD for Server Applications,” ICS’09.

Advanced Interfaces

- Memory types
  - We will see later how devices can mix different memories
    - Amount of DRAM
    - SLC + xLC Flash
    - Use of Persistent Memory (e.g., Optane)
  - But let's hold until we discuss a concrete design example.
- Channel Architecture
  - The current architecture serves the purpose of hiding Flash latency and aggregating bandwidth well
  - But would it hold if certain user computations are generating traffic inside the device?

These may require deeper architectural changes to the devices.

Advanced Interfaces Mini-Survey

- Chang, “Hybrid solid-state disks: Combining heterogeneous NAND flash in large SSDs,” Asia and South Pacific Design Automation Conference’08.
- Im and Shin, “Storage Architecture and Software Support for SLC/MLC Combined Flash Memory,” SAC’09.

Internal Interfaces Summary

Architect's Perspective

- Database systems design does not need to stop at the bottom of the Storage Manager.

Programmer's Perspective

- We gain from implementing I/O workloads that can be easily characterizable.

Data Management and SSDs General References

- Fevgas et al., “Indexing in flash storage devices: a survey on challenges, current approaches, and future trends,” The VLDB Journal 29, 2020.
- Lee et al., “A Case for Flash Memory SSD in Enterprise Database Applications,” SIGMOD’08.
- Xu et al., “Performance Analysis of NVMe SSDs and their Implication on Real World Databases,” SYSTOR’15.

External Interfaces

Philippe Bonnet

Tutorial Structure

- Interfaces
  - P2 - Internal
  - P3 - External
- Co-designing
  - P4 -Computational and Programmable Devices
  - P5 - Examples of devices
- Tooling
  - P6 - Continuum
  - P7 - SPD tools: simulation, prototyping, and development

Abstractions and Mechanisms

How to incorporate SSDs into the overall system design?

1. What are the appropriate abstractions?

   Abstractions of storage at multiple levels. From bottom up:

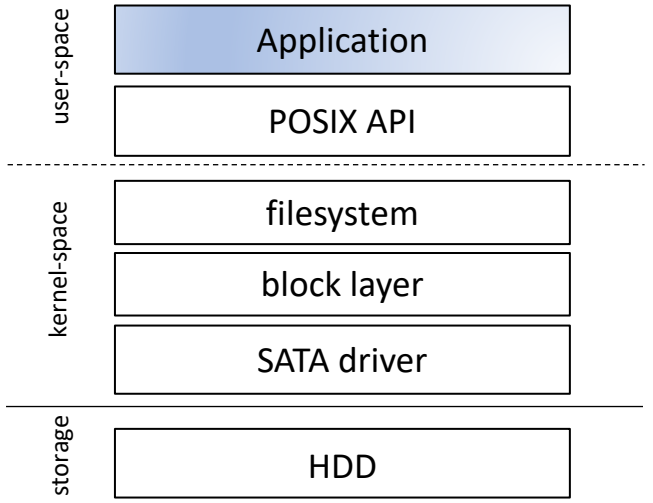
   - SSD interface hiding the device complexity
   - Storage abstraction made available to applications, hiding I/O complexity.
   - Applications map their workload onto the storage abstraction.

2. What are the I/O mechanisms providing access to SSDs?
   - I/O mechanisms provided by operating system or user-space libraries.

Grandpa Abstractions: POSIX and Blocks

For decades, in Unix sytems, applications mapped data structures onto files atop the block device interface.

- The block device interface abstracts storage as a linear array of fixed sized blocks.
  - Memory abstraction:
    payload <- read(LBA); write(LBA, payload).
- POSIX I/Os abstract storage as a collection of files.
  - Inodes map file onto collection of blocks.
  - The original POSIX specified synchronous commands, based on a buffer to avoid I/Os.
- Applications map data structures onto files
  - An ecosystem of tools and commands makes it easy to organize and manipulate files.



From HDD to SSD (*)

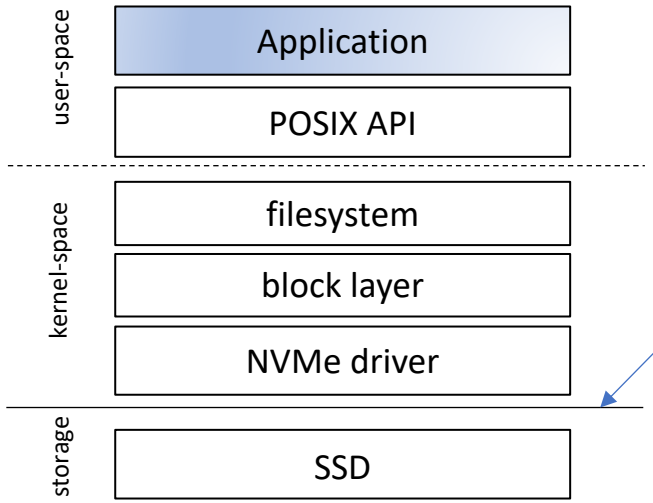
I/O Performance (2020)	I/O performance does not matter	I/O performance is crucial
HDD disk seek: 2msec 1 MiB seq. read: 718 usec	POSIX file with buffered I/O on top of Block-based HDD	Custom buffer management with direct POSIX I/Os on top of Block-based HDD
SSD (*) rand. read: 16 usec 1 MiB seq. read: 39 usec	POSIX file with buffered I/O on top of Block-based SSD	Beyond POSIX and Blocks with NVMe over PCIe-attached SSDs

(*) SSDs are not a uniform class of devices. Continuum from latency-optimized (Z-NAND) to archival.

Beyond POSIX and Blocks

- Beyond POSIX:
  - POSIX is the textbook example of a deep module. But it does not give applications control over (i) allocation and layout policies, (ii) I/O scheduling, and (iii) creates redundancies and missed optimization opportunities.
  - Need for streamlined I/O path.
- Beyond blocks:
  - The block device interface hides SSD parallelism and requires complex firmware to handle the idiosynchracies of flash.
  - Need to expose parallelism and align storage interface with SSD characteristics (as well as application needs).

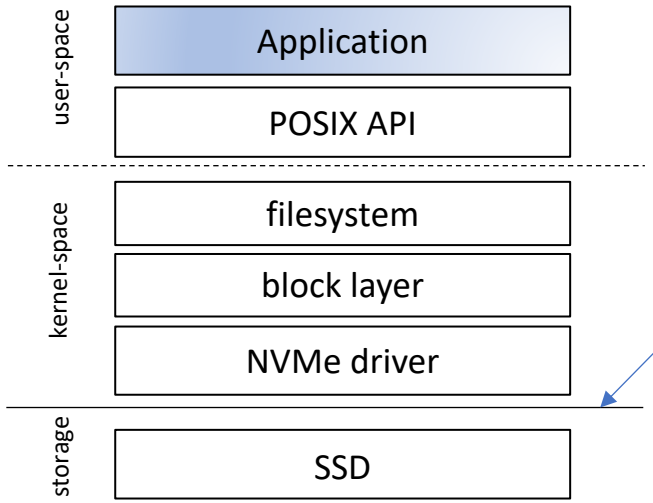
A streamlined I/O path exploiting SSD parallelism requires a deep understanding across layers.



Deeper Dive into PCIe

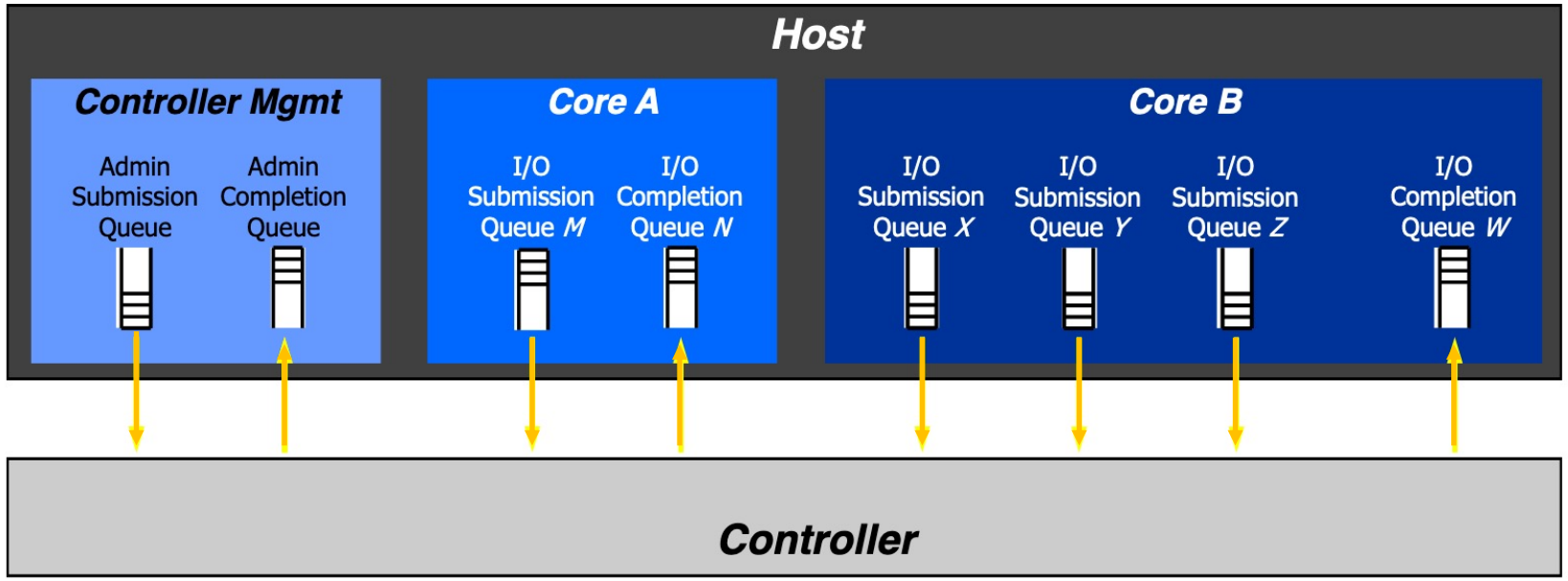
- PCIe is a layered network protocol.
  - Requests/responses (transaction layer) are exchanged atop a packet-based data link protocol (data link layer) and physical connections organized as a collection of lanes (physical layer).
  - Each lane is a pair of unidirectional, serial, point-to-point connections over short distance (max 1m).
  - PCIe fabric is organized as a tree, with a root complex and multiple endpoints connected directly or via switches.
- Each device is associated to a memory-mapped region of the host address space, defined through Base Address Registers (BAR)
- PCIe theoretical bandwidth is comparable to RAM's theoretical bandwidth.

PCIe	Gen 3 (2010)	Gen 4 (2017)	Gen 5 (2019)	Gen 6 (2021)
x4 (M.2)	4 GB/sec	8 GB/sec	16 GB/sec	32 GB/sec
x16	16 GB/sec	32 GB/sec	64 GB/sec	128 GB/sec



Deeper Dive into NVMe

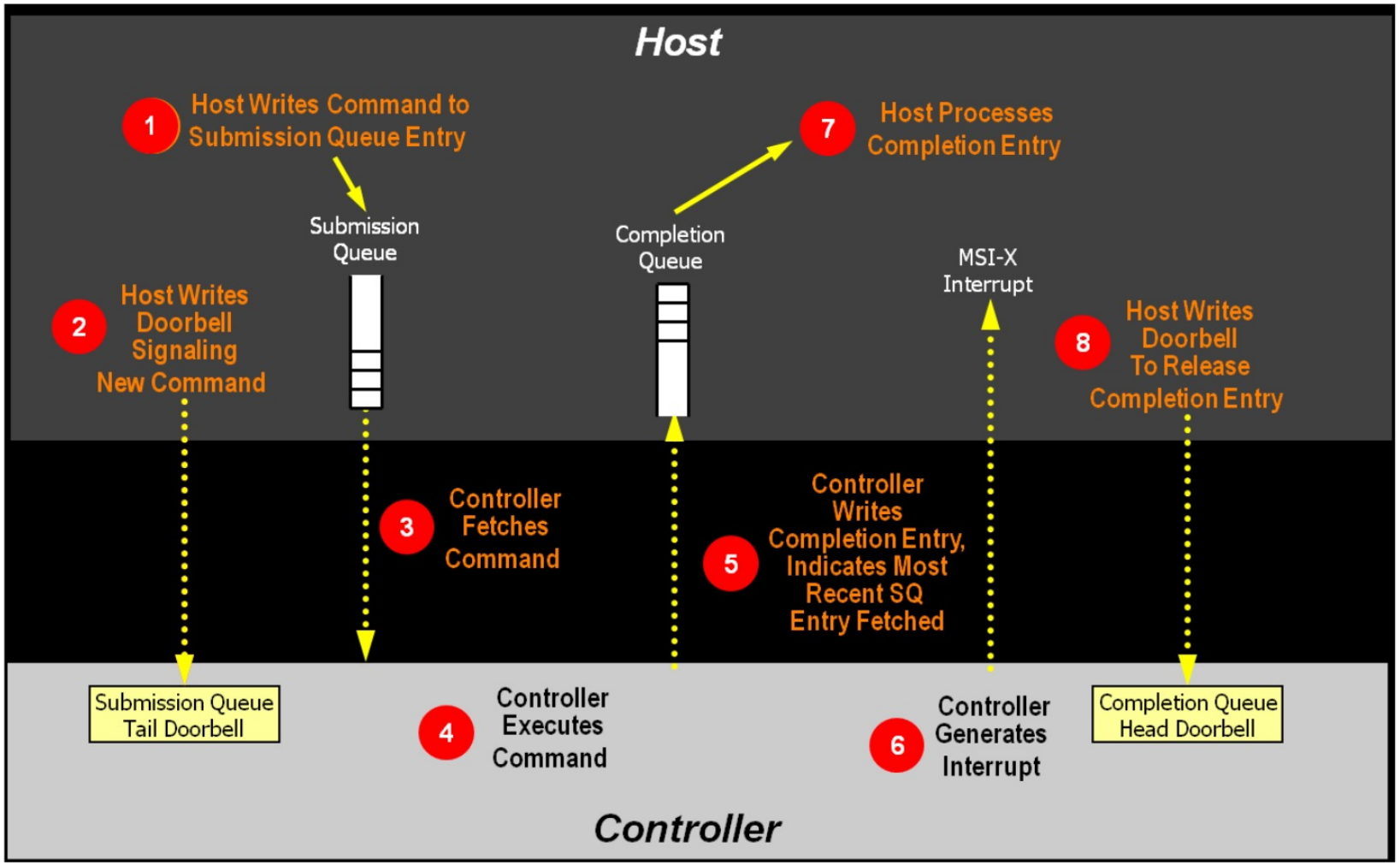
- NVMe is a host-controller interface specification
  - Designed to attach SSDs directly to the PCIe fabric.
  - First specification in 2011. Consortium led by Intel.
  - NVMe 2.0 released on 3/6/2021



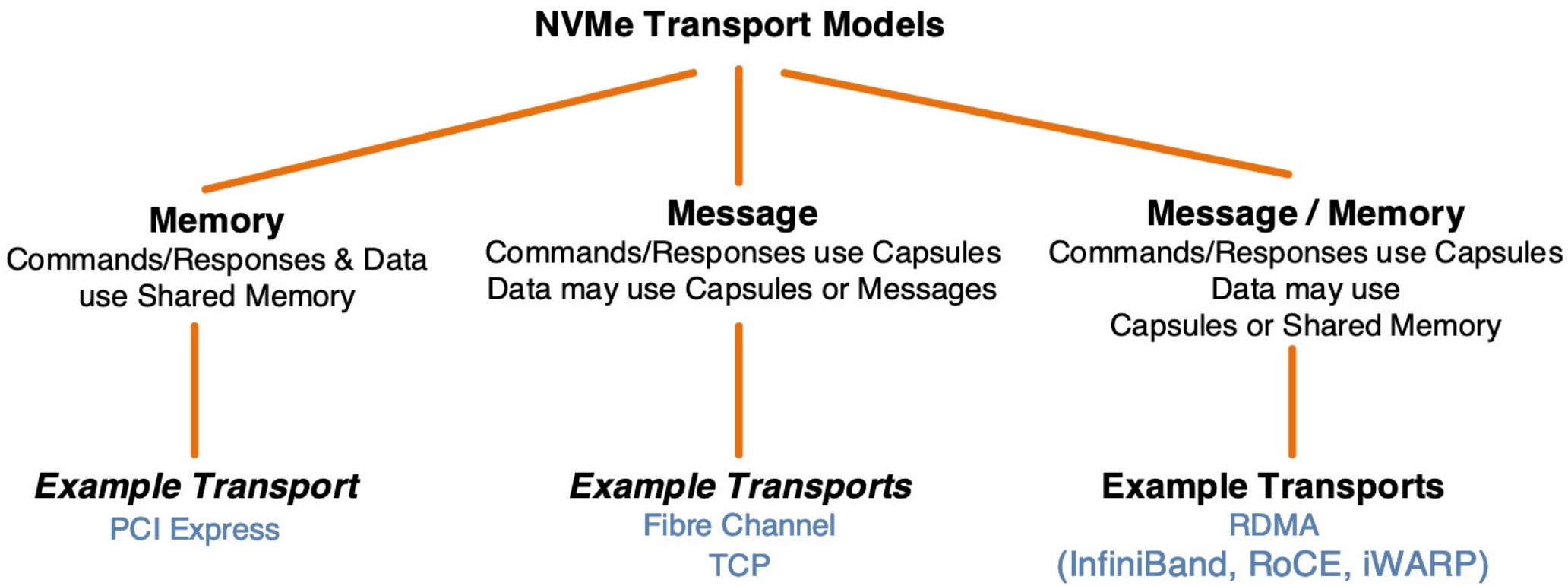
Hosts and controllers communicate through pairs of submission/ completion queues.

The queues are located in memory-mapped address space either on the host or on the device.

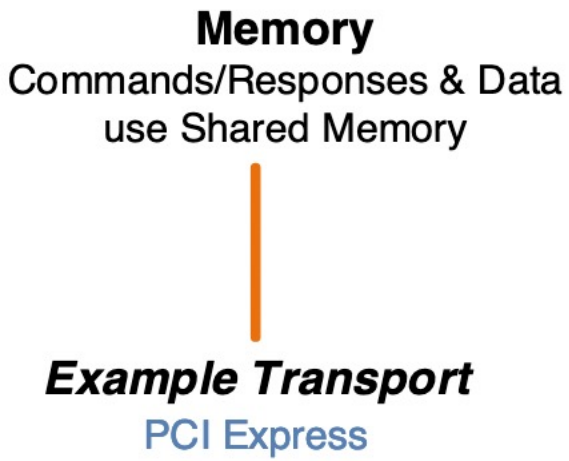
NVMe Host-Controller Interactions



NVMe 2.0 Transport Models



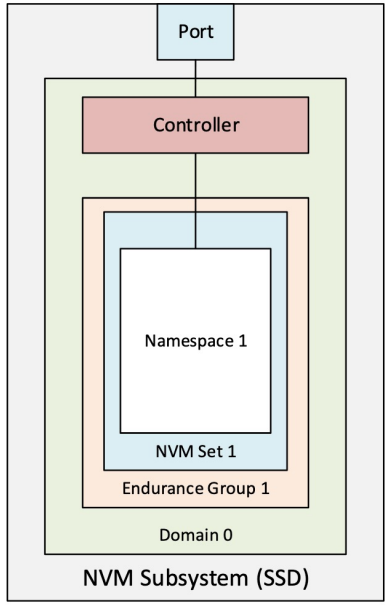
NVMe & Memory Management



1. Host memory buffer (HMB) is a portion of host memory for the exclusive use of the controller
2. Controller memory buffer (CMB) is a region of general purpose read/write memory on the controller
   - Associated to a PCI Express address (additional BAR)
   - Potentially used for peer to peer communications
3. Persistent memory region (PMR) is a region of general purpose PCI Express read/write persistent memory.

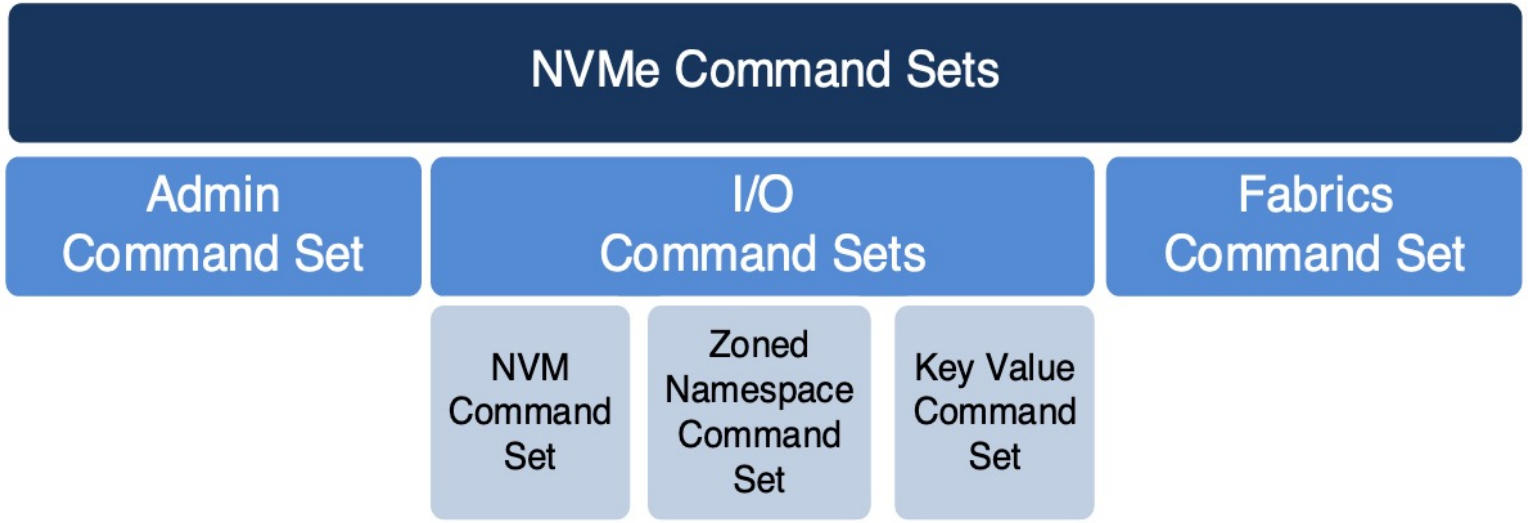
NVMe Abstractions

Namespaces

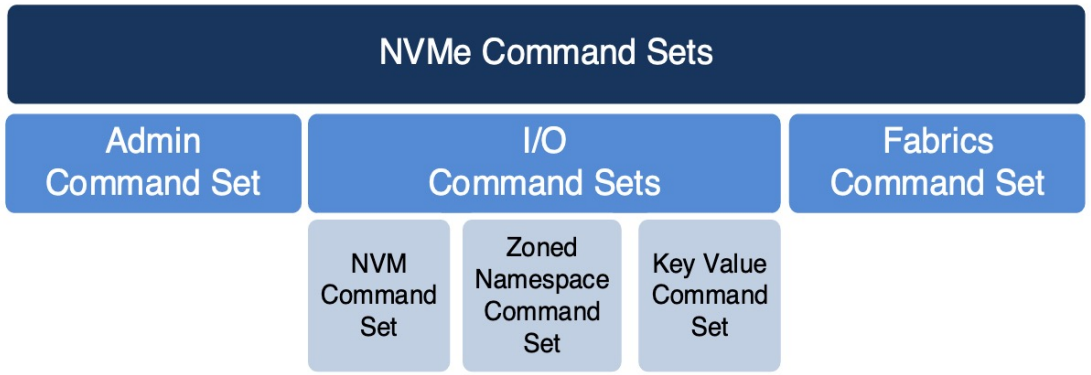


- A namespace is a formatted quantity of non-volatile memory that may be accessed by a host.
- Each namespace has an ID, a size, a capacity (max number of LBAs used), and a utilization

Command Sets



Command sets are the operations associated to a namespace



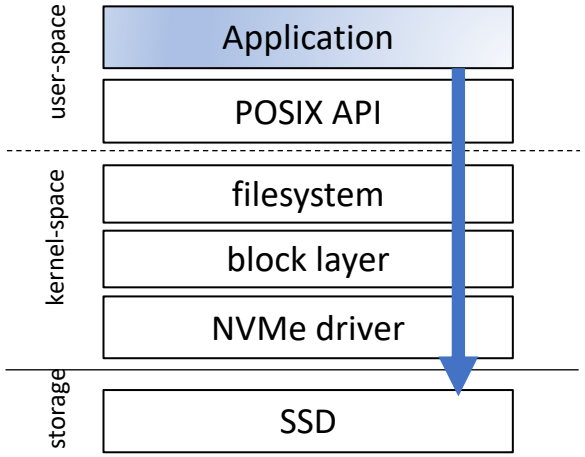
NVMe Command Sets

- Admin command set include the creation and deletion of submission/completion queues, as well as primitives for device identification, or getting log-pages, device capabilities, and features.
- Three types of I/O command sets:
  - NVM: The namespace is a collection of logical blocks, with read, write, write-zeroes commands. This is the block device abstraction.
  - Zoned: The namespace is partitioned into zones. Each zone is a collection of logical block addresses. The command set establishes that logical blocks must be written sequentially within a zone and that zones must be reset before they are written. It also defines the append command.
  - Key-Value: The namespace is organized as a collection of key-value pairs. The maximum key size is 16B. The commands supported are store/retrieve, list, exist, delete.

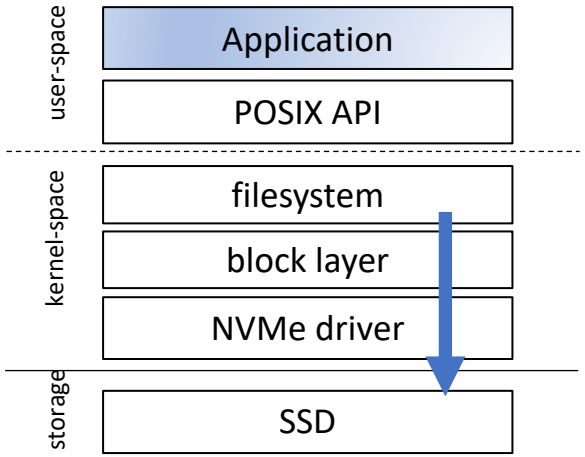
NVMe Directives

NVMe directives make it possible for hosts to cooperate with the SSD.

- I/O Command Directive are used as part of an I/O command.
  - The stream directive, only associated to writes, allows a host to indicate that LBAs belong to separate streams.
    - SSDs equipped with support for multi-streams isolate data from different streams on different blocks, thus ensuring that they can be accessed or garbage collected together.

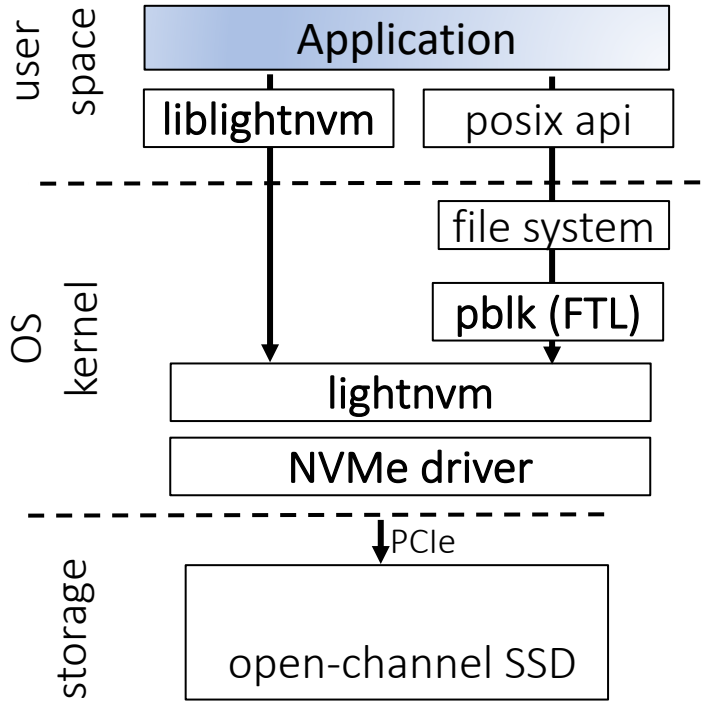


SPDK makes it possible for applications to associate directives to I/O commands.

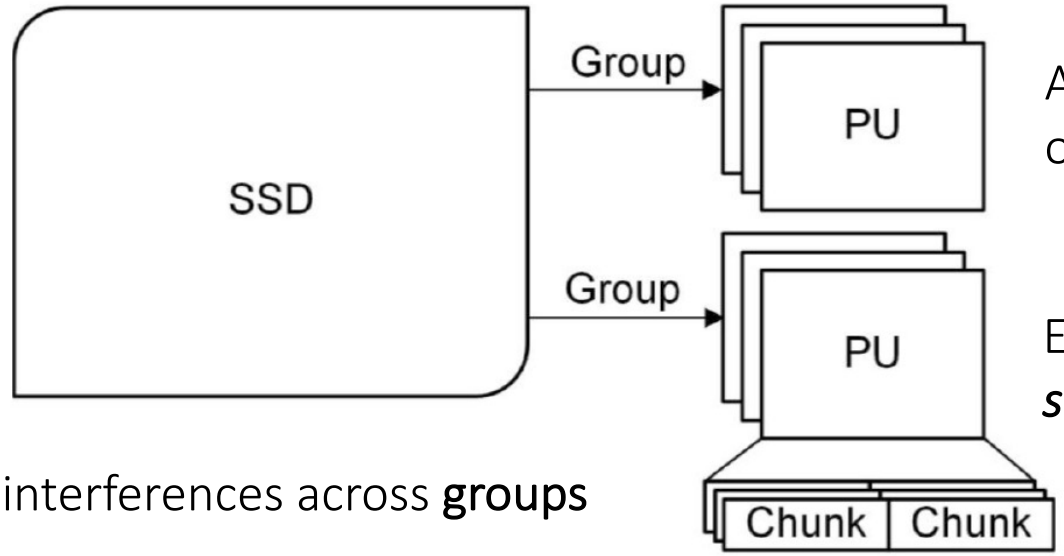


Stream directives generated from lifetime hints in the fcntl system call.

NVMe Extensions



The Open-Channel Interface specification (LightNVM): defines a hierarchical address space and let applications manage data placement and I/O scheduling.



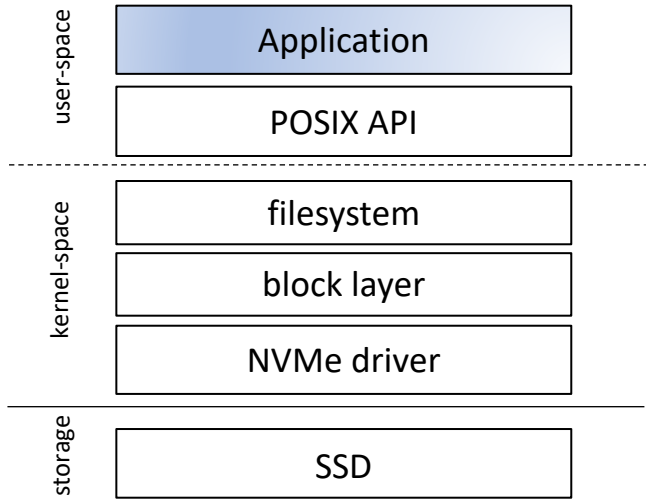
A parallel unit is a collection of ***chunks*** *(= unit of erase)*.

Each chunk is composed of ***sectors*** *(= unit of read; 4 KB)*

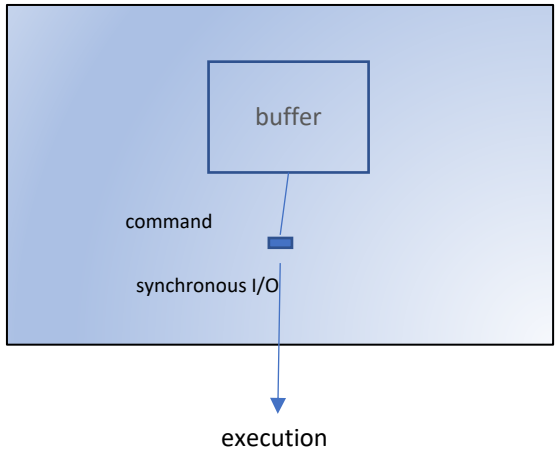
No interferences across **groups**

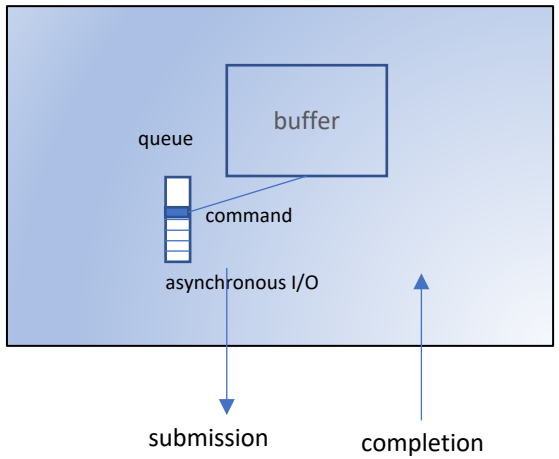
Multiple sectors constitute a **logical block** *(= unit of write):* function of flash characteristics. Logical blocks must be written sequentially within a chunk

Legacy Linux I/O Frameworks



- The original POSIX interface was based on synchronous commands
  - Exposed through C standard library (unistd.h)
- Two flavours of asynchronous I/Os introduced in mid-2000s:
  - aio is a user-space emulation of POSIX asynchronous I/Os that relies on worker threads issuing synchronous I/O calls.
  - libaio, actually issues asynchronous I/Os through the io_submit system call.
    - This system call takes as input a context (initialized through the io_setup system call), and a control block (iocb).
    - On completion, the system call io_getevent is used to reap io_event data structures that may contain a pointer to a callback function.
    - Note that libaio requires that a file be opened with O_DIRECT. It only works with the following file systems: ext2, ext3, jfs and xfs. Libaio requires two system calls per I/O.





Modern Linux I/O Framework

- io_uring was introduced by Jens Axboe in 2019
  - It relies on pairs of circular submission/completion queues shared between user-space and kernel-space.
  - The queues are single producer and single consumer. Submission queue entries are 64B, while completion queue entries are 16B.
  - System calls are available for (a) setting up queues, (b) registering application memory referenced in submission queue entries and (c) initiating/completing a number of asynchronous I/Os. By default, io_uring requires a single system call for multiple I/O submissions and completions.
  - It is also possible to setup a queue pair with a flag (SETUP_SQPOLL) so that io_uring starts a kernel thread that polls the shared submission queue for entries.
- io_uring is available through liburing that defines a range of helper functions for setting up queues, manipulating buffers and generating submission queue entries.
- io_uring can be used with any file system with buffered or unbuffered I/Os. It recently added support for submitting ioctls.

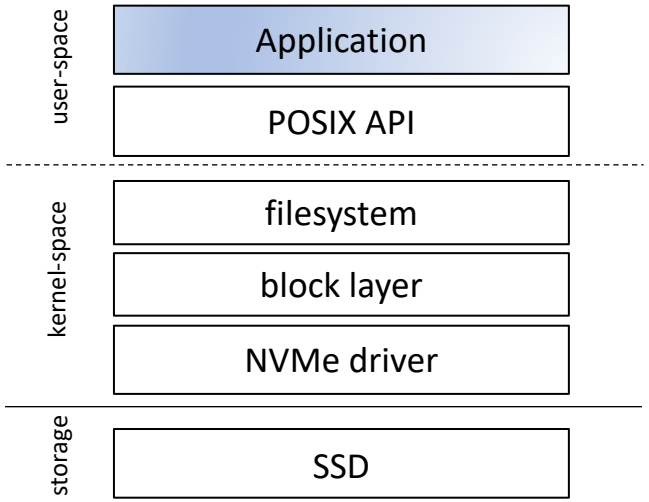
Design Choices

- Bypassing the buffer cache
  - Using the O_DIRECT flag.
- Bypassing the file system
  - After opening a device file, read and write operations at a given file offset are passed on to the block layer without interference.
  - The block layer receives *bio* requests. It may reorganize them, independently on each core, in software queues. Bio requests are then dispatched to the NVMe driver via hardware queues.
  - The block layer provides various mechanisms to deal with completion: signal-based, continuous polling or hybrid polling.
- Bypassing the block layer
  - ioctl give applications direct access to the NVMe driver via synchronous commands.
- Bypassing the kernel
  - SPDK framework based on a user-space NVMe driver, packaged as a library that maps PCIe BARs directly into the application process thus supporting zero-copy.
  - Functions are provided to allocate the queue pairs used for I/O submission and completion, as well as payload buffers in this DMA-transferrable memory.
  - NVMe SSDs can be accessed directly via a C API (nvme.h for block devices nvme_zns.h for ZNS drives).
  - The application must ensure that a single thread submits I/O and that this thread polls for completion.

I/O frameworks Trade-Offs

- SPDK supports zero-copy and enables trading higher CPU utilization for lower I/O latency, but exposes applications to PCIe/NVMe limitations (number of queues, number of entries per queues, buffer region size, single thread handling submission/completion).
- io_uring isolates applications from PCIe and NVMe, supports a range of targets (raw device, file, ioctl) and tuning options (sqpoll) with a single system call for a collection of I/Os. But it traverses multiple software layers, requires a copy between software and hardware queues in the block layer and exposes significant complexity to applications.
- xNVMe is a user-space library that abstracts I/O mechanisms across a range of operating systems. It is a means to design portable, simple, yet efficient NVMe-based software (xnvme.io).

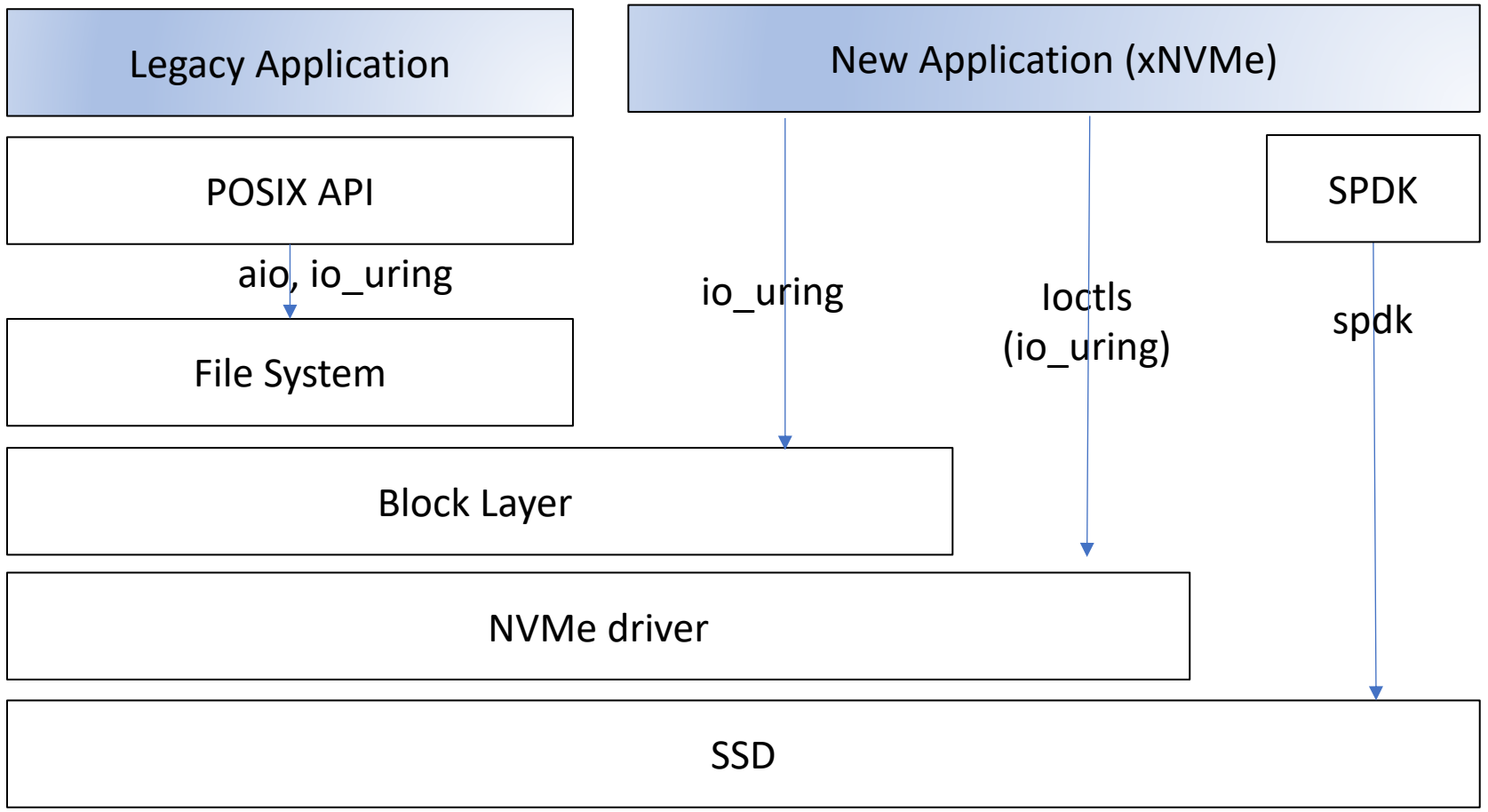
Linux I/O Frameworks



File system extensions for ZNS
In F2FS, XFS

mlq-blk for NVMe since 2013

No support for KV in NVMe driver yet



External Interfaces Summary

Architect's Perspective

- NVMe is the narrow-waist of the storage stack.

  => Transport-based storage stack (PCIe, fabric). What is the impact of memory-centric interconnects CXL/Gen-Z?

- Variety of abstractions, but no sweet-spot for database workloads.
  - NVM: log-on-log.
  - ZNS: no placement or scheduling
  - KV: no caching.
  - Open-Channel: unknown internal policies.

  => Is computational storage the answer?

Programmer's Perspective

- NVMe directives as cross-layer communication mechanism
- New abstractions and mechanisms require a storage manager re-design

External Interfaces General References

- Jens Axboe. Efficient IO with io_uring. https://kernel.dk/io_uring.pdf, 2019
- MatiasBjørling. From Open-Channel SSDs to Zoned Namespaces. USENIX Vault, 2019.
- Matias Bjørling et al. LightNVM: The Linux Open-Channel SSD Subsystem. In USENIX Conference on File and Storage Technologies, 2017.
- IvanLuizPicoli et al. Open-Channel SSD (What is it Good For). In Conference on Innovative Data Systems Research (CIDR), 2020.
- Javier Gonzalez. Zoned Namespaces: Standardization and Linux Ecosystem, SNIA. SDC EMEA, 2020.
- Myoungsoo Jung. OpenExpress: Fully Hardware Automated Open Research Framework for Future Fast NVMe Devices. In USENIX Annual Technical Confer- ence, 2020.
- Jeong-Uk Kang et al. The Multi-streamed Solid-State Drive. In USENIX Workshop on Hot Topics in Storage and File Systems, HotStorage 14, 2014.
- NVMe 2.0 Base Specification. https://nvmexpress.org/wp-content/uploads/NVM-Express-Base-Specification-2_0-2021.06.02-Ratified-5.pdf
- PCIe specification. https://pcisig.com/specifications

Computational, Programmable and Co-Designed SSDs

Philippe Bonnet

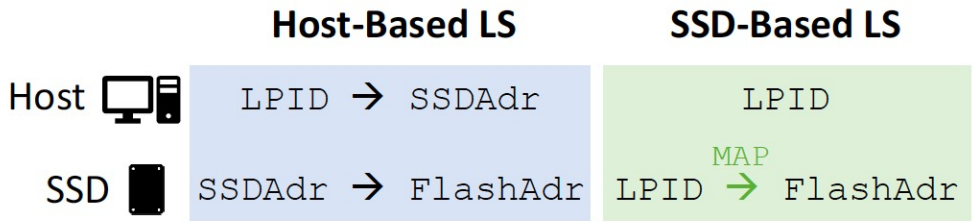
Tutorial Structure

- Interfaces
  - P2 - Internal
  - P3 - External
- Co-designing
  - P4 -Computational and Programmable Devices
  - P5 - Examples of devices
- Tooling
  - P6 - Continuum
  - P7 - SPD tools: simulation, prototyping, and development

The case for SSD Interfaces Adapted to Database Workloads

Functionality Argument

- Reduce complexity of host software with higher-level storage abstractions
- Example: OX-Batch
  - Transactional interface
    - Avoids log-on-log
  - Batch objects writes
    - No application-level mapping
    - Avoids round-trips



Performance Argument

- Latency:
  - Streamlined software on I/O Path to avoid duplications and leverage optimization opportunities (e.g., hardware acceleration).
- Cost:
  - Reduce CPU load
- Energy consumption:
  - Reduce data movement
- Throughput
  - Leverage internal SSD throughput

New SSD Interfaces

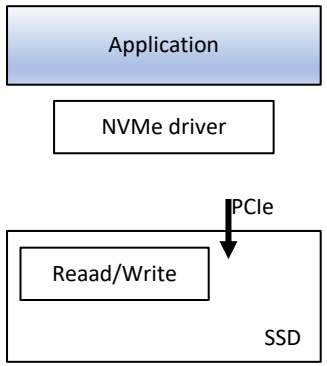
Providing new SSD interfaces adapted to database workloads requires a transitions from standard memory (read/write) interfaces to a communication (send/receive) interface

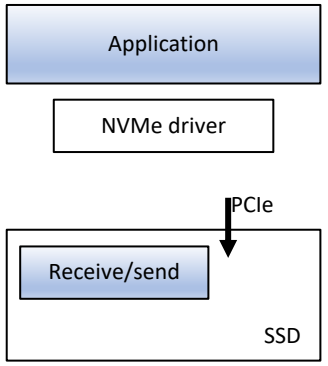
This transition requires

(i) computation on SSD side to process messages sent by host,

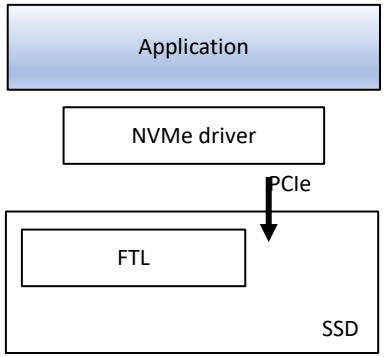
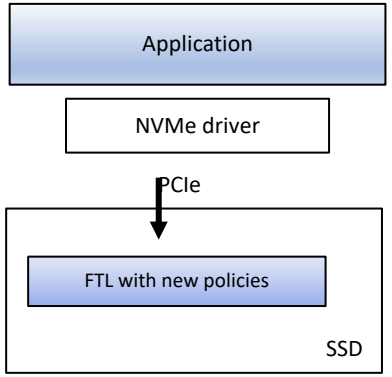
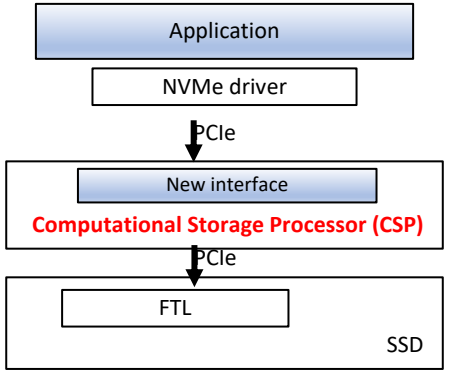
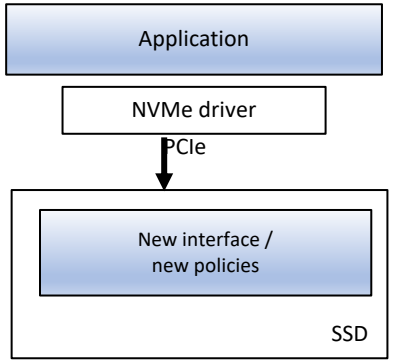
(ii) a communication protocol, and

(iii) integration of computation and SSDs within storage.

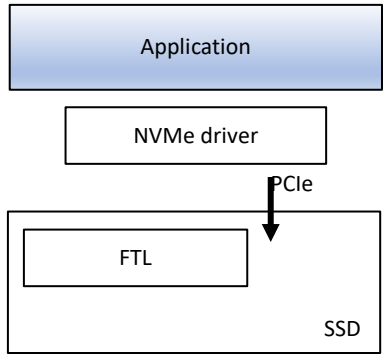
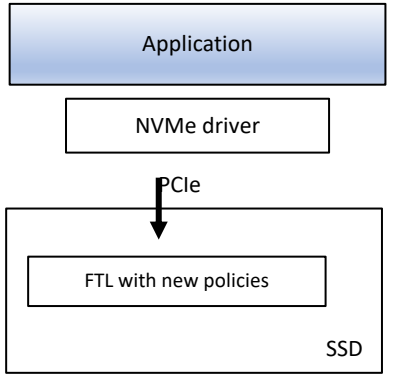
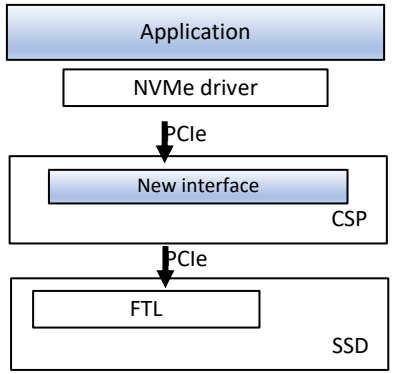
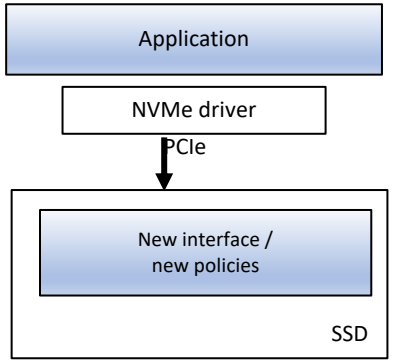




SSD Nomenclature

	SSD as Black box	SSD as White Box
Standard External Interface	Standard SSD 	Programmable SSD 
New External Interface	Computational SSD 	Co-Designed SSD 

SSD Nomenclature

	SSD as Black Box	SSD as White Box
Standard External Interface	Standard SSD 	Programmable SSD 
New External Interface	Computational SSD 	Co-Designed SSD 

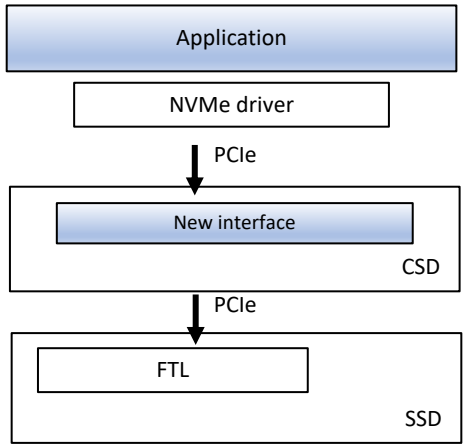
Decades of Research on SSD internals

Decades of Research on near-data processing

Programmoing CSD for deployments on cloud and HPC

Database SSDs

Computational Storage: 1 Slide Survey



- Longer Surveys review the state-of-the-art and identify open issues.
- Prototypes tailored for the need of a specific system on a given hardware platform:
  - FPGA-based, ARM-based, x86-based computational storage devices.
  - Research prototypes of a database system, a dataflow processing engine, a machine learning platform, a similarity search algorithm or a scientific computing application.
  - Deployments on Public clouds (Alibaba, AWS) and HPC clusters (Los Alamos National Lab).

- Software frameworks for computational storage:

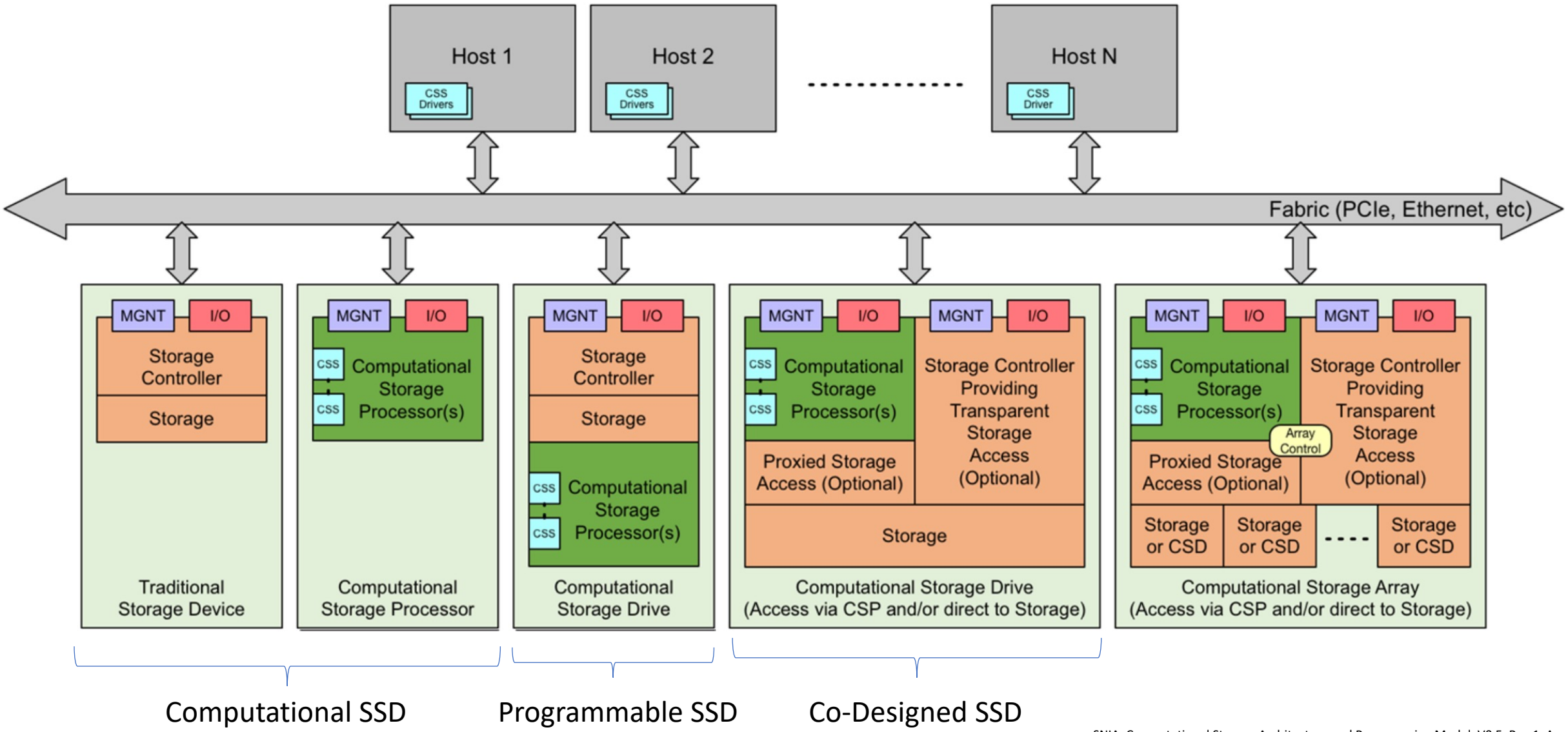
  Willow: SSD apps called via RPC, cannot be composed.

  Biscuit: SSD Task Pipelines programmed in C++, compiled on the host and shipped to CSD.

Computational Storage: Industry Viewpoint

- Standardization efforts to integrate computational storage into existing storage ecosystem.
  - SNIA: Fixed vs. Programmable Computational Storage Services
    - Fixed services: Pipeline, Database filter, Scatter-Gather, Compression, Data deduplication
    - Programmable services: OS, container, bitstream, BPF
  - NVMe: New command set for computational storage
    - Task force led by Eideticom, Intel, Samsung with deadline early 2022
    - Educated guess:
      - Command set based on load/unload/execute abstraction
      - Data transferred with existing command sets

SNIA Architecture

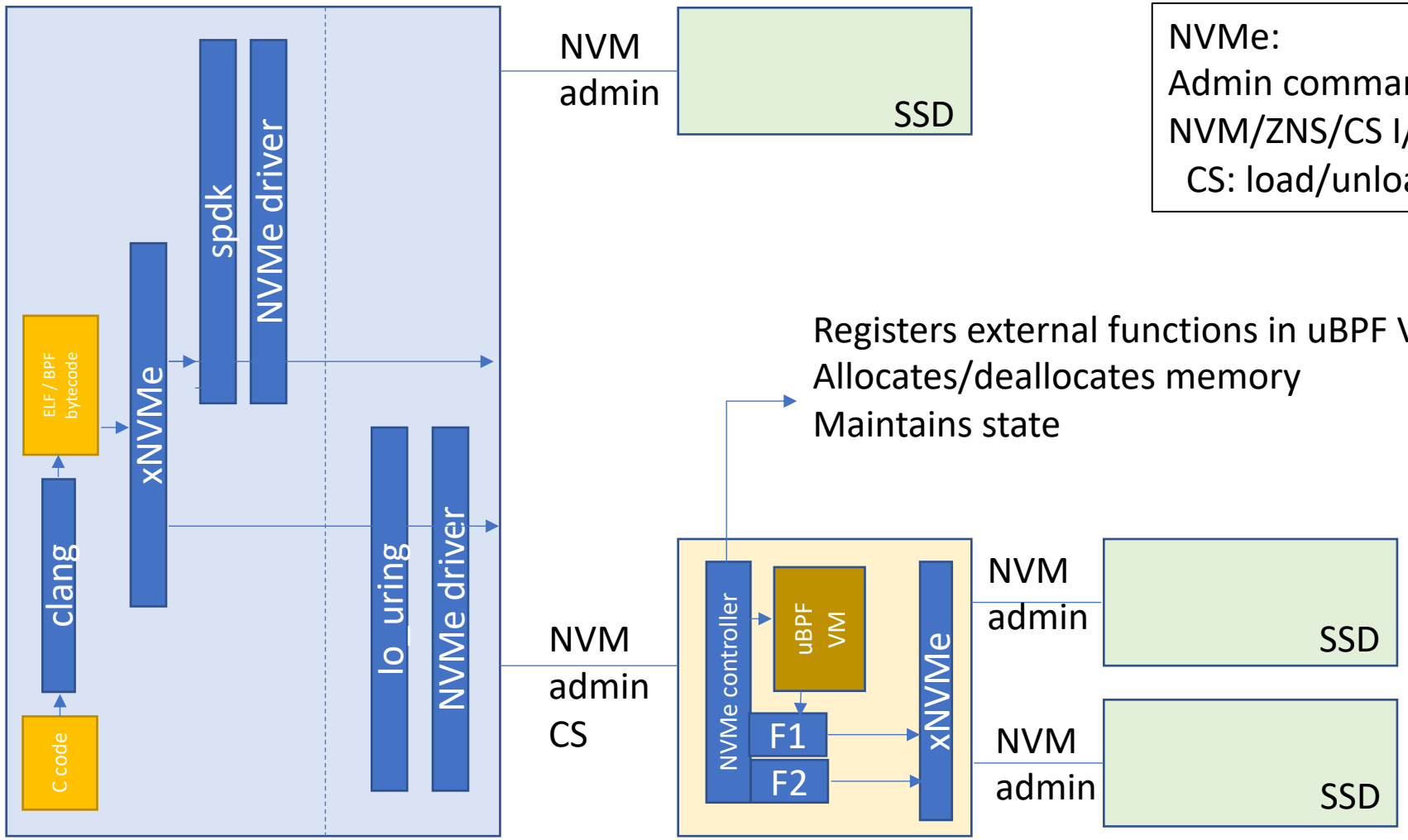


SNIA. Computational Storage Architecture and Programming Model. V0.5, Rev 1. Aug 2020.

Role of BPF

- BPF is a vendor-neutral Instruction Set Architecture (ISA)
  - Assembly-like byte code
  - Standardized in Linux kernel, JIT on variety of controllers.
  - BPF backends for clang and gcc.
- NVMe is extensible, however …
  - Defining a new extension for each new storage interface is cumbersome.
  - BPF as generic mechanism to call computational storage services from host
  - BPF programs as means to compose and orchestrate function calls on CSP.

Handling New Storage Interfaces with BPF



NVMe:
Admin command set
NVM/ZNS/CS I/O command sets
CS: load/unload/execute

Source: N.Hedam et al. uBPF and Computational Storage: A Reality Check. 2021.

Computational Storage: Open Issues

- SSD Integration
  - How to support NVMe access from CSP?
  - How to deal with data movement across multiple devices?
    - CMB provides support for peer-to-peer transfers, but those transfers need to be orchestrated and streamlined.
- Computational Storage Processor
  - How to handle namespaces from CSP programs? Need for data virtualization.
  - How to orchestrate multiple offloaded programs?
  - How to isolate multiple tenants?
  - What to hardware accelerate? How?
  - What is the attack surface? How to protect it? How to verify BPF on CSD?
- System Design
  - What to offload?
  - What services to call? How to program CSP?
  - How to place data and processing?
  - How to deploy CSP software?

Co-Designed SSDs: 1 Slide Survey

- Framework for offloading computation on co-designed SSD
  - **INSIDER**: FPGA-based reconfigurable controller as computational storage processor. INSIDER provides a file system abstraction. Operations on computational storage are exposed as operations on files, internally organized as a pipeline of sub-programs. INSIDER manages the isolation and scheduling of offloaded programs. They rely on a customized I/O stack to access stored data.

Co-Design SSDs: Research Agenda

How can database system experts
program a co-designed SSD that is tailored for their workload?

- What abstractions to support the definition of new storage interfaces?
- How to define new policies that support new storage interfaces?

Computational & Programmable SSDs Summary

Architect's Perspective

- Computational storage soon part of NVMe ecosystem.

Programmer's Perspective

- Many open issues programming and deploying CSPs at scale for a range of different data-intensive systems and applications:
  - Who programs CSP? How?
  - Target workloads/use-cases?
  - How to integrate CSP with overall system design?

What are appropriate abstractions for programming Co-Designed SSDs?

$\Rightarrow$Today, it is possible to explore this question experimentally (P7: Tooling)

Computational Storage General References

- Ivan Luiz Picoli., OX: Deconstructing the FTL for Computational Storage. PhD thesis, 2019.
- J. Do et al., Programmable solid-state storage in future cloud datacenters. Communications of the ACM, 62(6), 2019.
- J. Do et al., Better Database Cost/Performance via Batched I/O on Programmable SSD, *VLDB Journal*, 2021.
- W. Cao *et al.*, POLARDB Meets Computational Storage: Efficiently Support Analytical Workloads in Cloud-Native Relational Database', 2020.
- N. Borić et al., 'Unified Spatial Analytics from Heterogeneous Sources with Amazon Redshift', in *Proceedings of the 2020 ACM SIGMOD*, Jun. 2020.
- 'Los Alamos announces details of new computational storage deployment'. https://www.lanl.gov/discover/news-release-archive/2020/November/1116- computational-storage.php
- SNIA Draft Technical Work available for Public Review | SNIA'. https://www.snia.org/tech_activities/publicreview
- A. Barbalace et al., Computational Storage: Where Are We Today?, CIDR 2021.
- R. Balasubramonian *et al.*, 'Near-Data Processing: Insights from a MICRO-46 Workshop', *IEEE Micro*, vol. 34, no. 4, Jul. 2014.
- Woods et al. (2014). Ibex-An intelligent storage engine with support for advanced SQL offloading. Proceedings of the VLDB Endowment. 7. 963-974.
- A. HeydariGorji et al., 'STANNIS: Low-Power Acceleration of Deep Neural Network Training Using Computational Storage', *DAC* 2020
- J. Do et al., 'Cost-effective, Energy-efficient, and Scalable Storage Computing for Large-scale AI Applications', ACM Transactions on Storage, 2021.
- M. Torabzadehkashi, et al., 'Accelerating HPC Applications Using Computational Storage Devices', HPC *2019*
- Z. Ruan et al., 'INSIDER: Designing In-Storage Computing System for Emerging High-Performance Drive'
- B. Gu *et al.*, 'Biscuit: A Framework for Near-Data Processing of Big Data Workloads', in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture*
- S. Seshadri et al. 2014. Willow: a user-programmable SSD. In *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation* (*OSDI'14*).
- Yuhong Zhong, et al., BPF for storage: an exokernel-inspired approach. *HotOS '21*

Example of Programmable Devices

Alberto Lerner

Tutorial Structure

- Interfaces
  - P2 - Internal
  - P3 - External
- Co-designing
  - P4 -Computational and Programmable Devices
  - P5 - Examples of devices
- Tooling
  - P6 - Continuum
  - P7 - SPD tools: simulation, prototyping, and development

Examples of Advanced Devices

Programmable device

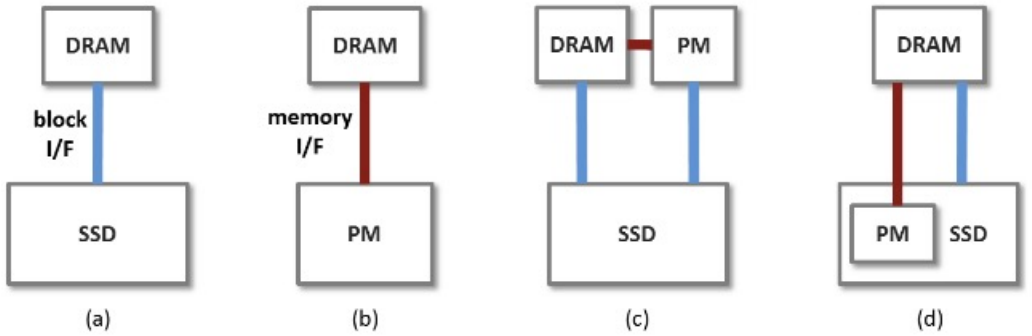
- The 2B-SSD creates a device with a dual (sided) interface : the traditional one and a byte-addressable one.
- The byte-addressable interface has the same functionality as Persistent Memory but is managed by the device.
- The application can control the “side” in which a given page resides and benefit from that side's advantages momentarily.

Computational Device

- Moves the workload generation into the device.
- The Graph-SSD creates a command set to access graph data and moves certain graph computations into the device.

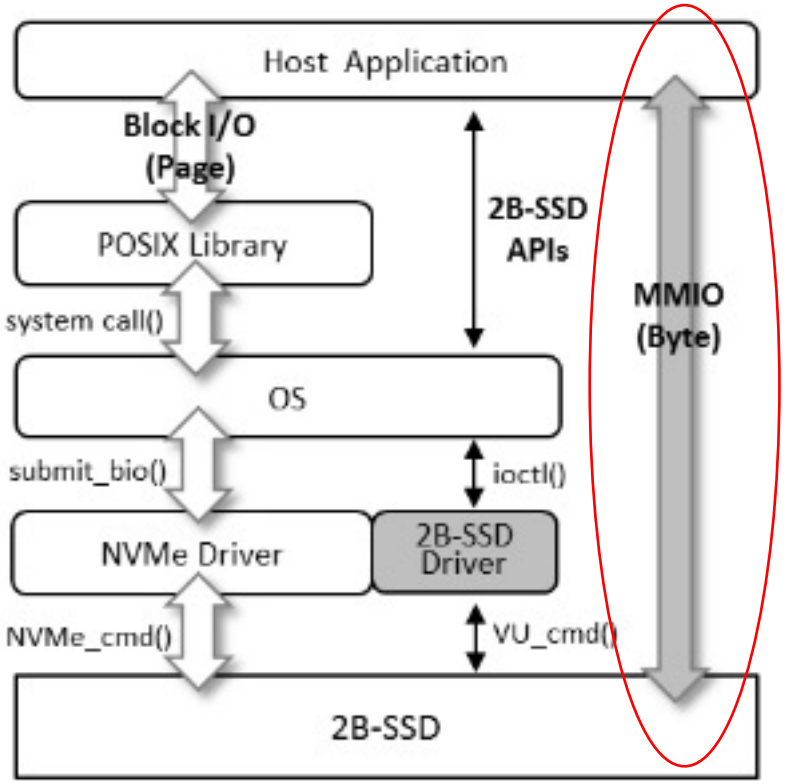
A Hybrid PM-SSD Device

- There are various possible storage hierarchies in a system:
  a) A two-level hierarchy of DRAM and Flash (via SSD devices);
  b) A two-level hierarchy of DRAM and Persistem Memory (with a CPU-base PM Controller);
  c) A DRAM and PM store controlled via software;
  d) [2B-SSD] A hybrid PM/NAND-Flash SSD.



An Alternative Data Path

- An application can write directly to the fast, byte-addressable “side” of the 2B-SSD:
  - Just write against an established address region.
- The region can be backed by a LBA (logical block address) determined by the application.
- Meanwhile, other LBA can be read/written to via NVMe.
- 2B-SSD changes some subsystem (page mapping) using internal interfaces.



Byte-addressable interface

Source: Bae 2018

Software API

- The application can move an LBA between “sides” of the device
- WAL can be implemented using the “fast” side
- Results:
  - 1.74x to 2.71x faster in Postgres and RocksDB workloads

- BA-PIN

  LBA: Flash->PM managed
- BA-FLUSH

  LBA: Flash content == PM content
- BA-SYNC

  LBA: Flash<-PM managed

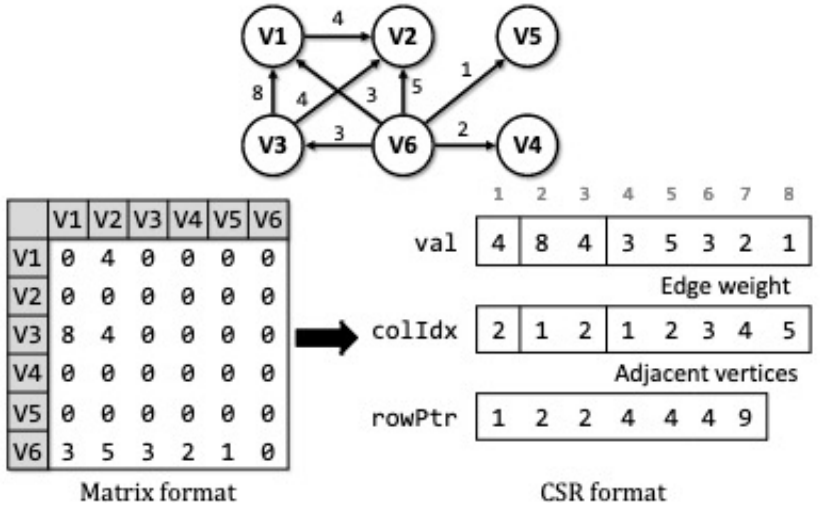
Let’s talk about PM

- 2B-SSD implements persistent memory via NVDIMM (DRAM + capacitors).
- Nothing prevents this design from using another type of byte addressable memory such as Optane.
  - As we’ve seen, NVMe allows for a Persistent Memory Region in the device.

We don’t believe in PM vs. Flash; PM + Flash in a device is another design region that can help certain workloads better than separate access to PM and Flash.

Motivation for A “Graph-Friendly” Device

- Large graphs can occupy entire devices.
- Sparse graphs are represented in compressed adjacency matrices such as CSR [paper].
- In this scenario, the main workload fetches the data necessary for the host to calculate the neighbors of a vertex.
- What if the device knew how to navigate CRS files?



Source: Matam 2019

A Graph API

- Commands are passed through the NVMe interface
  - Although using reserve bits rather than new command sets
- Can leverage page placement and caching internal interfaces, as it understands the CPR format and the primitives ran atop of it.
- Results:
  - Between 1.29x to 1.85x faster in several basic graph primitive benchmarks.

Category	Commands
Graph read commands	GetAdjacencyList, GetEdgeWeight
Graph update commands	AddEdge, AddVertex, DeleteEdge, DeleteVertex, UpdateEdge, UpdateVertex
Graph initialization	GraphInitialize

Table 1: GraphSSD APIs

An Actual SSD Implementation

- The GraphSSD is operational.
  - Implemented on top of the OpenSSD, which we discuss in the Tooling Section.
  - None of the authors belong to an SSD manufacturing company; they were all in Academia.
  - None of the authors was from the OpenSSD project.

Discussion

- Programmable and Computational devices represent two powerful axes for creating specialized devices.
- The specialization aims at servicing a given workload (or a set thereof) better than a workload-agnostic device would.
- A question may arise as to why go to all this trouble to obtain, at best, 1.8x and 2.7x acceleration? 10x or 100x would look better.
  - Unfortunately, these works do not quantify the CPU savings they create.
  - They do so with devices that are not too different than COTS ones.
  - The true question is why would you peg your CPU (in this *post-Moore* era of ours) with tasks that can be offloaded to a device?
- Our job is to create the means to make this offloading more accessible.

The Co-Design “Continuum” of Techniques

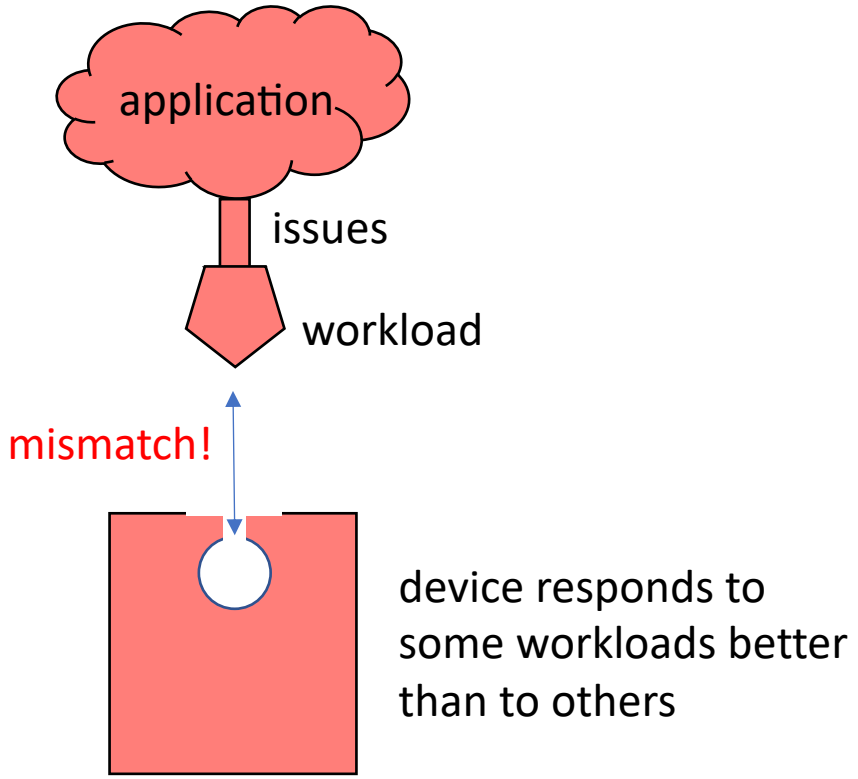
Alberto Lerner

Tutorial Structure

- Interfaces
  - P2 - Internal
  - P3 - External
- Co-designing
  - P4 -Computational and Programmable Devices
  - P5 - Examples of devices
- Tooling
  - P6 - Continuum
  - P7 - SPD tools: simulation, prototyping, and development

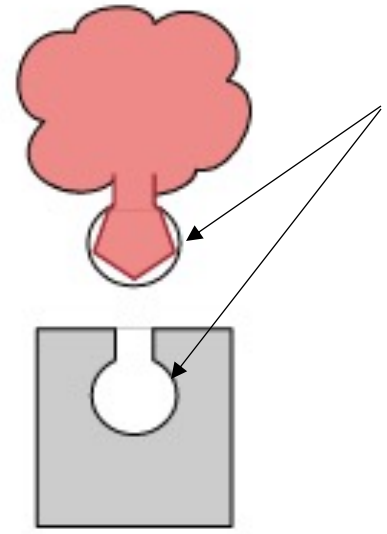
Application-Device “Pairing”

- We characterized applications before by the workload they generate.
- Different devices would perform differently under a given workload.
- If we understand the workload issued and the ideal workload for a device, we can determine application-device “compatibility.”



“Grandpa” Pairing Approach

- We often assume that the application is the one that should adapt to the device. But how?
- Shape the workload to fit the devices “unwritten contract”:
  - issue large requests, in parallel if possible;
  - prefer locality of access;
  - favor sequential Requests;
  - group requests by life-time;
  - prefer uniform life-times.



Perform the IO slightly differently to obtain better response from the device

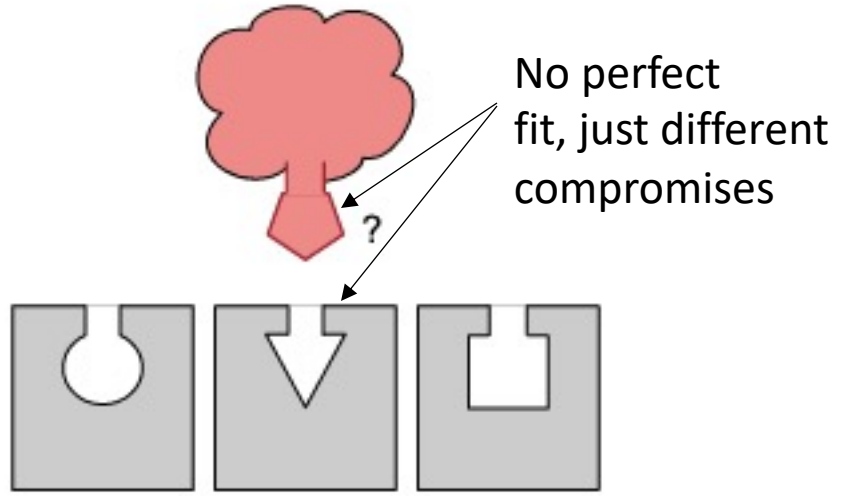
Why tailor only the application?!

“Unwritten Contract” Mini-Survey

- Desnoyers, “Analytic Modeling of SSD Write Performance,” SYSTOR’12.
- He at al., “The Unwritten Contract of Solid State Drives,” EuroSys’17.
- Kakaraparthy et al., “Optimizing Databases byLearning Hidden Parameters of Solid State Drives,” VLDB, 13(4), 2019.
- Wu, Arpaci-Dusseau, and Arpaci-Dusseau, “Towards an Unwritten Contract of Intel Optane SSD,” HotStorage’19.
- Zuck et al., “Why and How to Increase SSD Performance Transparency,” HotOS’19.

Choice of Device Variation

- As we said before, different devices perform differently under a different workload.
- That's because they make different design decisions along a few axes:
  - Read- vs write-oriented
    - Type of cell used (SLC/MLC/…)
  - Low-latency at QD1
    - Optane and Z-NAND
  - Parallelism Degree
    - Number of planes or number of channels
  - Write performance variance
    - Over-provisioning level

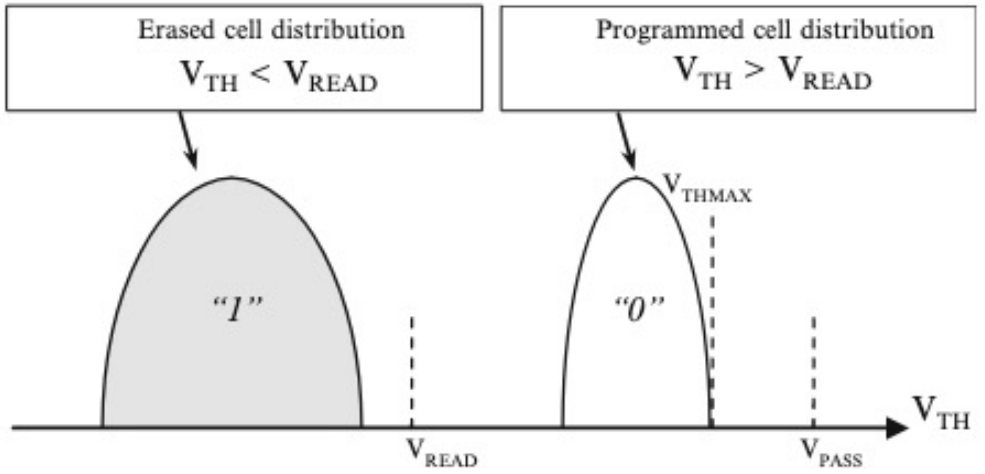


Devices as different points in the design space

Type of cell is arguably on of the biggest sources of performance variation!

Single-Level Flash Cells

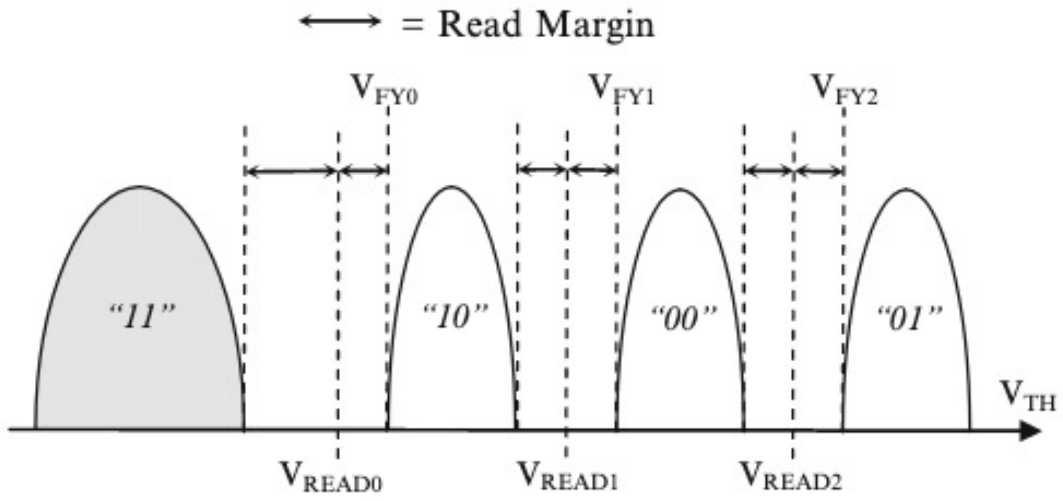
- Simply put, a NAND-flash cell traps voltage.
- In Single-Level Cells, a cell has two voltage levels.
- If the voltage trapped is below the V_{READ} that's a 1, otherwise 0.
- To *program* a 0, the voltage level is increased using a method called *Incremental Step Pulse Programming.*



Source: Micheloni 2013

Multi-level Flash Cells

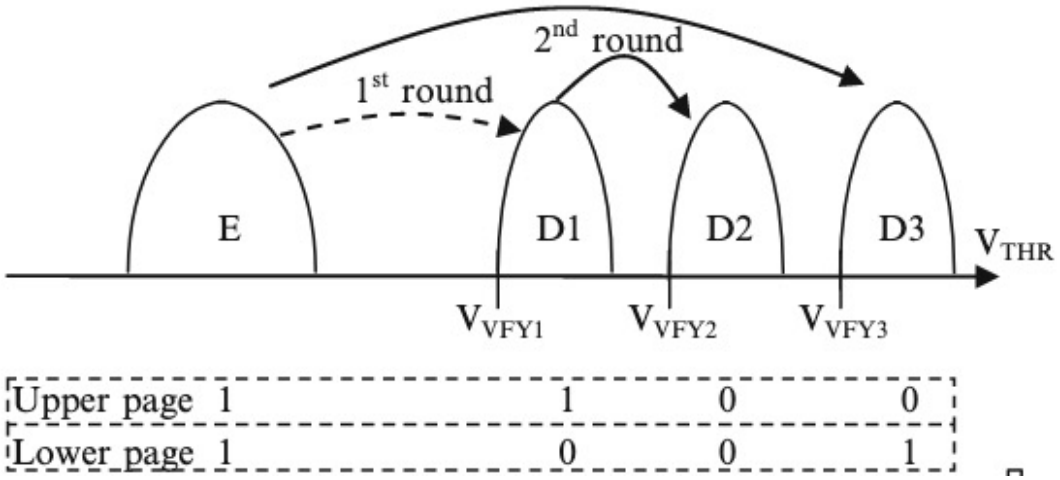
- In Multi-Level Cells, there are more than two possible voltage levels.
- Devices encode bits of two or more pages in a single cell, the Lower and Upper page (or Least Significant Bit or Most Significant Bit)
- The figure is for an MLC cell of 2-bits but TLC and QLC devices exist of 3- and 4- bits, respectively.



Source: Micheloni 2013

Programming LSB and MSB

- Note, however, that voltages can only be increased.
  - We must write the lower page (LSB) in a 1st round, before writing the upper page (MSB).
  - The 2nd round is more time consuming the first round.
- Writing to the upper page is slower than to the lower one!



Source: Micheloni 2013

Understand the Design of a Device

- Check whether your device is SLC/MLC/TLC/QLC or 3D
  - 1-, 2-, 3-, 4-bit cells or stacked
  - An example of a mixed SLC-MLC device

Latency	Read	SLC mode		99 μs
		MLC mode	LSB page	58 μs
			MSB page	90 μs
			Average	74 μs
	Write	SLC mode		486 μs
		MLC mode	LSB page	481 μs
			MSB page	2,295 μs
			Average	1,388 μs

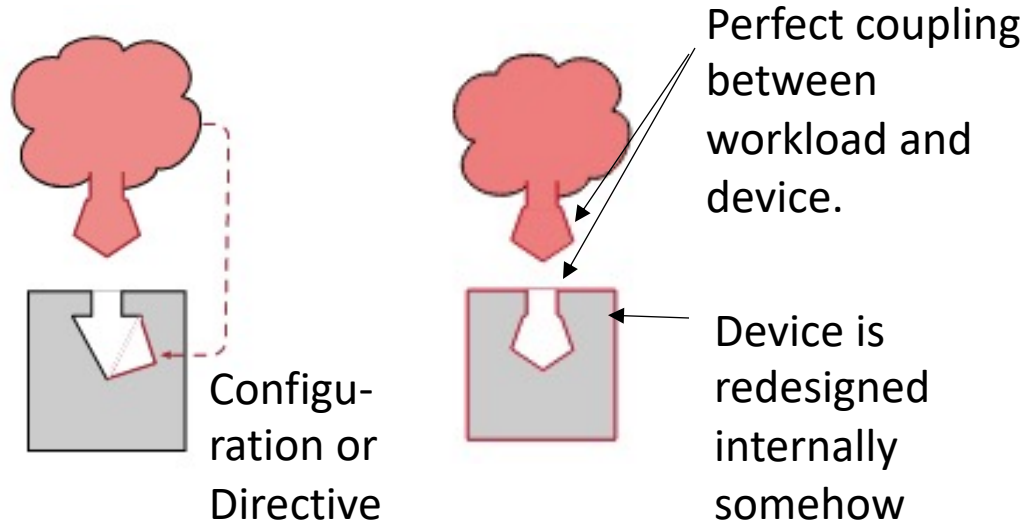
MSB is 4.7x slower!!

- Modern devices can implement several optimizations
  - Some cells in the device may be “downgraded” to single-bit
  - Writes may go to (fast) downgraded SLC cells

Source: Kwak 2020

Configurable and Programmable Devices

- Configurable ones use current NVMe directives:
  - Queue Priorities;
  - Streams.
- Programmable ones introduce new semi-specialized devices by replacing components:
  - Specialty caching such as GALRU and 2R;
  - Specialty scheduling such as Flin;
  - Specialty placement such as DC-Store;
  - Specialty ECC such as the Approximate SSD;
  - Multiple components as in X-FTL and 2B-SSD.



Programmable Devices References

- Bae et al., “2B-SSD: the case for dual, byte-and block-addressable solid-state drives,” ISCA’18.
- Kang et al., “X-FTL: transactional FTL for SQLite databases,” SIGMOD’13.
- Kang et al., “2R: efficiently isolating cold pages in flash storages,” VLDB, 13(11), 2020.
- Kwak et al., “GALRU: A Group-Aware Buffer Management Scheme for Flash Storage Systems,” IEEE Access, 8, 2020.
- Kuon et al., “DC-Store: Eliminating Noisy Neighbor Containers using Deterministic I/O Performance and Resource Isolation,” FAST’20.
- Li et al., “ASCache: An Approximate SSD Cache for Error-Tolerant Applications,” DAC’19.
- Tavakkol et al. “FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives,” ISCA’18.

Computational Devices

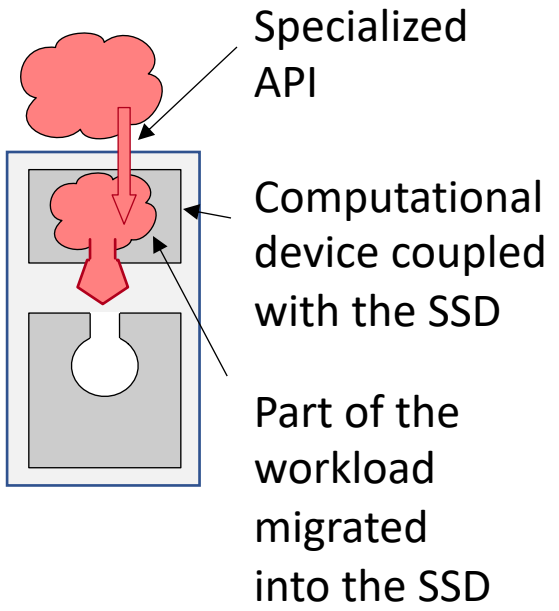
- Move the workload generation into the device.

Ibex

KV SSD

Graph SSD

SSD for Search Engines



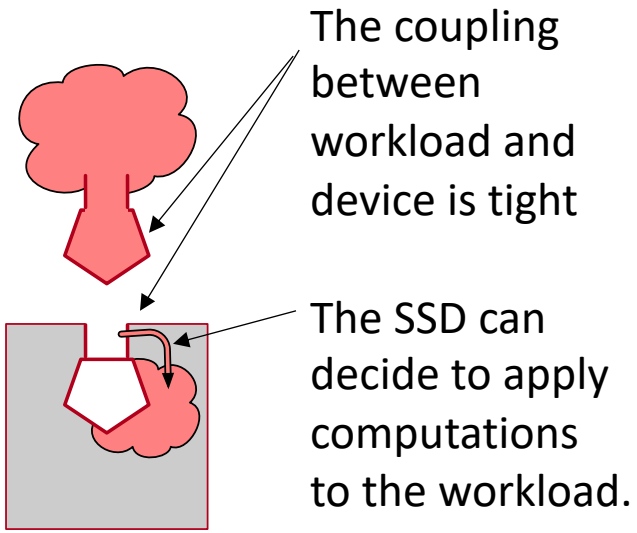
- There is still work required to allow a generic workload to be moved to the device.
  - Borrowing from the BPF work is only one out of several possible approaches.

Computational Devices References

- Bisson et al., “Crail-KV: A high-performance distributed key-value store leveraging native KV-SSDs over NVMe-oF,” IPCCC’18.
- Matam et al., “Graphssd: graph semantics aware ssd,” ISCA’19.
- Wang et al., “SSD In-Storage Computing for Search Engines,” IEEE ToC, 2016.
- Woods, István, and Alonso, “Ibex: An intelligent storage engine with support for advanced sql offloading,” VLDB, 7(11), 2014.

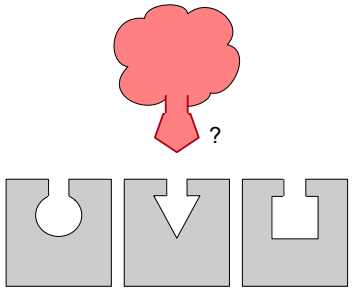
Fully Co-Designed Devices

- Inherit abilities from both Computational and Programmable devices.
- The workload is sent through the standard interfaces, but the device distills application's computations from the workload, sparing the former from performing those.
- The device is designed so that the computational device and the SSD blend perfectly.
  - The computational portions has access to the SSD internal interfaces.



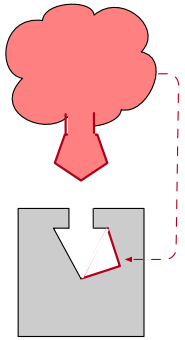
Data conversions is an example application.

Navigating the Co-Design Space

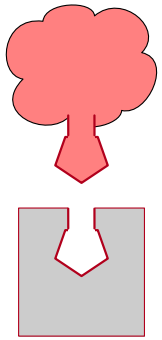


Perform device selection

Configure device to the workload



Tailor the device to the workload



Shift workload generation to the device`

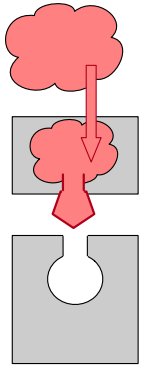
Configurable

Programmable

Computational

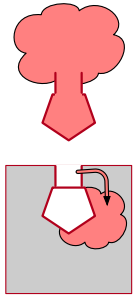
Co-designed

Co-Design



generic computations (standard)

Database SSDs



Additional References

- Do, Sengupta, and Swanson, “Programmable solid-state storage in future cloud datacenters,” CACM, 62(6), June, 2019.
- Kim et al., “In-storage processing of database scans and joins,” Elsevier Information Sciences, 327, 2016.

Simulation, Prototyping, and Development SSD Platforms

Alberto Lerner

Tutorial Structure

- Interfaces
  - P2 - Internal
  - P3 - External
- Co-designing
  - P4 -Computational and Programmable Devices
  - P5 - Examples of devices
- Tooling
  - P6 - Continuum
  - P7 - SPD tools: simulating, prototyping, and development

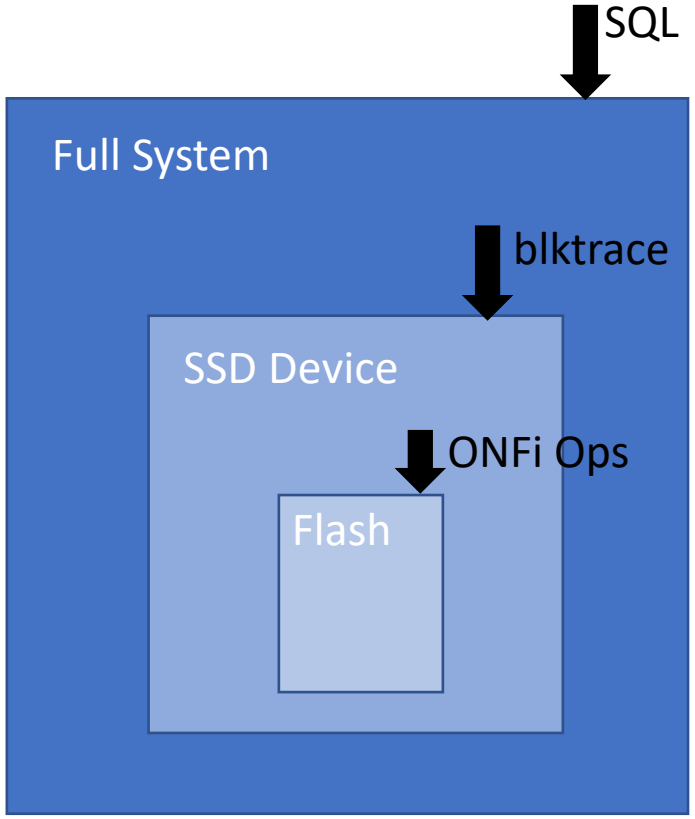
SPD Techniques

- What to use if we want to build or alter an SSD?
- There are at least three alternatives:
  - Work entirely on a software-Simulation of the hardware;
  - Work entirely on Prototyping the actual hardware;
  - Work on hardware-assisted Development.

- Each has pros and cons we we discuss here.

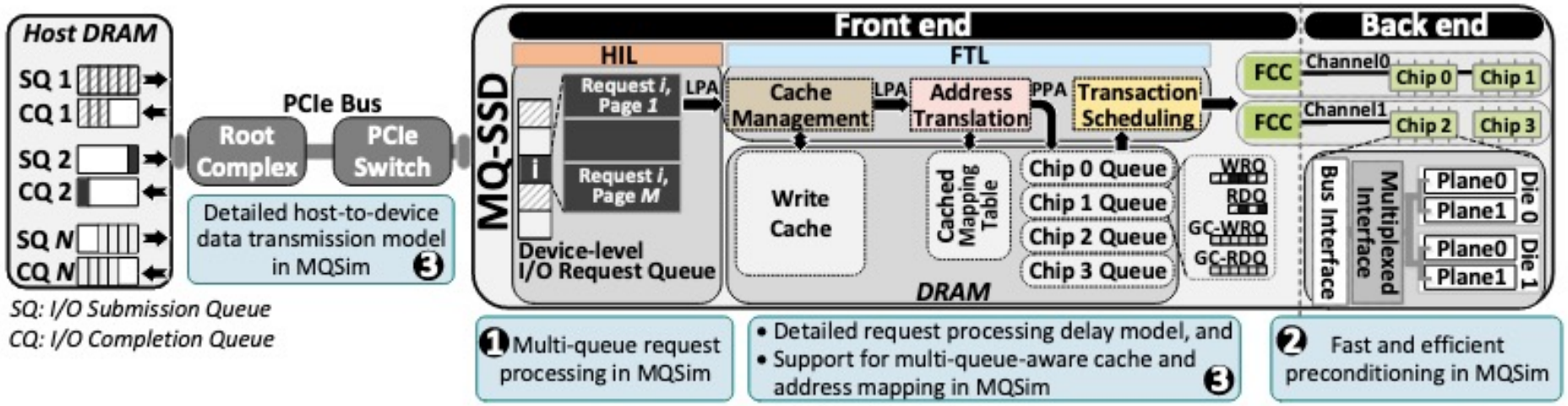
Software-based Simulation Techniques

- Use cycle-accurate software to simulate different portions of a system
  - Flash alone
    - FlashSim
  - Inside a device
    - MQSim
  - Inside a system
    - Gem5
- Workloads should be compatible with the simulation level



Example of an SSD Simulator

- MQSIM can be used stand-alone or inside a system-wide simulator such as Gem5.
- Allows access to three (out of four) internal interfaces we discussed.
- Can be used to implement new NVMe directives or command sets.



Source: Tavakkol 2018

Simulation Trade-offs

Pros

- 100% software.
- Easy learning curve to use certain simulators.
- The sky is the limit as to what new hardware can be modeled.
  - As long as it reflects a reasonable datasheet of such hardware (which is not always available).

Cons

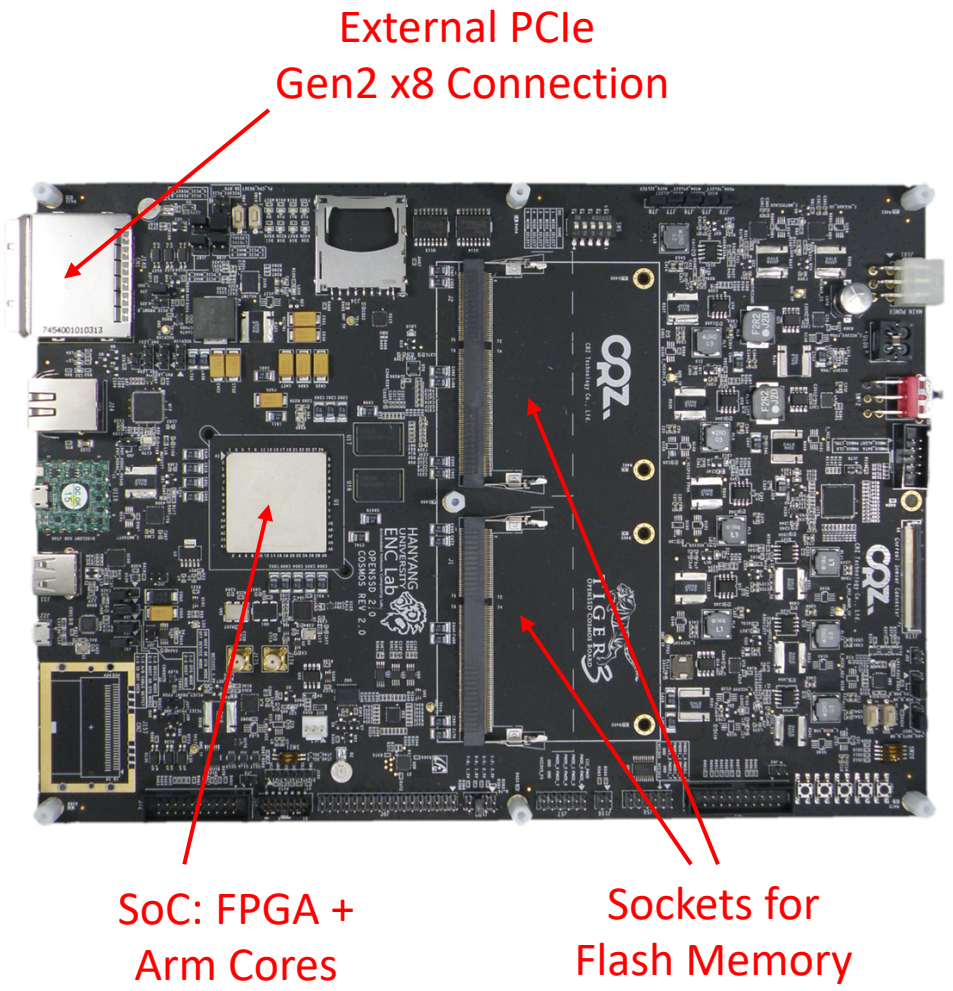
- Can’t capture aspects that are not modeled:
  - non-determinism in general, e.g., flash errors, flash aging.
- Slow execution:
  - Gen5 is equivalent to a machine running at a few MHz.

Simulation References

- Binkert et al., “The gem5 simulator,” SIGARCH Computer Architecture News, 39(2), 2011.
- Jung et al., “NANDFlashSim: Intrinsic latency variation aware NAND flash memory system modeling and simulation at microarchitecture level,” MSST’12.
- Gouk et al., “Amber: Enabling precise full-system simulation with detailed modeling of all SSD resources,” MICRO’18.
- Tavakkol et al., “Mqsim: A framework for enabling realistic studies of modern multi-queue SSD devices,” FAST’18.

Hardware-Based Prototyping Platforms

- OpenSSD Family of Devices
  - 3rd generation of devices using actual Flash memory.
  - Full-fledged SSD, 100% compatible with Linux/Windows NVMe drive
  - Large functionality set implemented as firmware (C coding) running on ARM cores
- Daisy Family
  - Infomal 4th generation of OpenSSDs
  - Commercial spin-off backed by CRZ in Korea



Prototyping Trade-Offs

Pros

- Real flash with real (at times idiosyncratic) behavior.
- Many examples of programmable devices.
- Growing community.

Cons

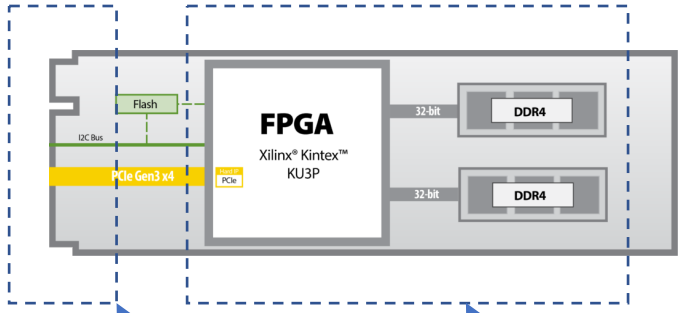
- Software-based features can add latency.
- Not many options of Flash and channel designs.
- Changing features close to the Flash require a steep learning curve.

Hardware Prototyping References

- Kwak et al., “Cosmos+ OpenSSD: Rapid prototype for flash storage systems,” ACM ToS, 16(3), 2020.
- Lee and Song, “Experimental Results of Implementing NVMe-based Open Channel SSDs,” Flash Memory Summit, 2017.
- Lerner et al., “It Takes Two: Instrumenting the Interaction between In-Memory Databases and Solid-State Drives,” CIDR’20.

Hardware-Assisted Development Platform

- Full-fledged NVMe device with emulated Flash
  - Carrier: any recent FPGA with PCIe and FMC connectors
  - OpenExpress NVMe controller (open for Academia)
  - Flash Emulator running on M.2 form factor FPGA (on the right)



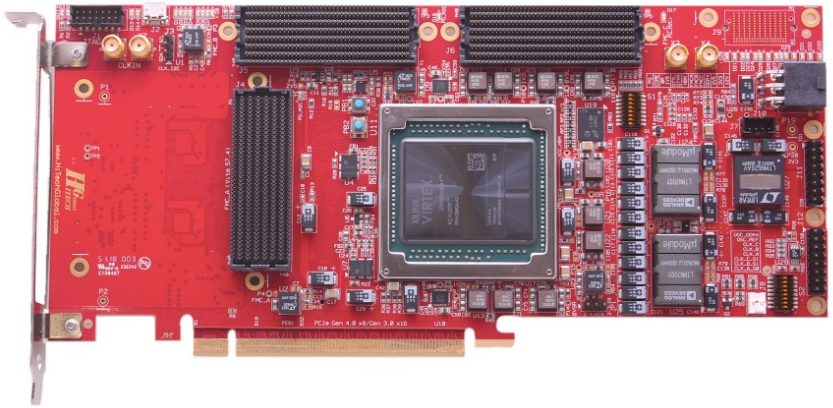
M.2 Interface

Flash emulation

"ONFi over PCIe?"

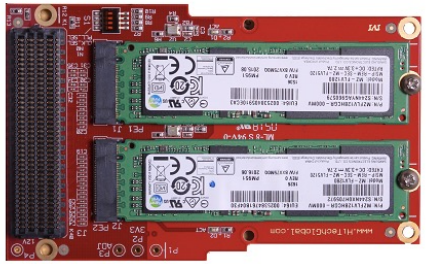
Example Setting

- Carrier



Hitech Global HTG-937
(technically a base network card)

- Flash Module Emulator



FMC-M2 Adapter and Bittware 250-M2D
(technically an OpenCompute Module
on an M.2 package)

Hardware-Assisted Development Trade-Offs

pros

- Can be as fast as a “mem disk”
- Can emulate different kinds of Flash and Persistent Memory
- Can potentially emulate different channel architectures

cons

- Flash emulator is currently still under development
- M.2 capacity is limited (32GB)

Hardware Development References

- Jung, “OpenExpress: Fully Hardware Automated Open Research Framework for Future Fast NVMe Devices,” USENIX ATC’20.
- Jung, Jung, and Song, “Architecture exploration of flash memory storage controller through a cycle accurate profiling,” IEEE ToCS, 57(4), 2011.
- Shen at al., “M.2 Accelerator Module Hardware Specification V1.0,” Open Compute Project, 2020.

Summary of the Approaches

	Development Speed	Execution Speed	Accuracy of results	Hardware Available
Software simulation	Fastest	Low MHz	Within a margin or error	Anything that has a datasheet available
Hardware prototyping	Slow	GHz	Exact results	Very limited
Hardware-assisted simulation	Slow to medium	100’s of MHz to GHz	Close to exact results	As wide as software when it comes to Flash memories

Conclusion

Take Away #1

- Making SSD behavior a part of a Database's stack is desirable and, currently, possible.
  - The design space for Database algorithms is free to explore beyond the current devices' limitations.
  - Computing paths and resource usage can become more efficient.
  - Devices can perform part of Database computations.

Take Away #2

- The changes made to a device should guided by a workload you understand, i.e., that you can characterize:
  - You can design a device that can adapt to the workload,
  - you can program a device to serve only that workload, or
  - you can move the logic that generates the workload into the device.

Take Away #3

- You have the external and internal interfaces at your disposal to export and implement your changes.
  - Your device can be 100% compatible with NVMe if you wish!

Take Away #4

- There are several alternative platforms to implement and evaluate your changes, ranging from pure software to pure hardware.

Thank you

Q&A