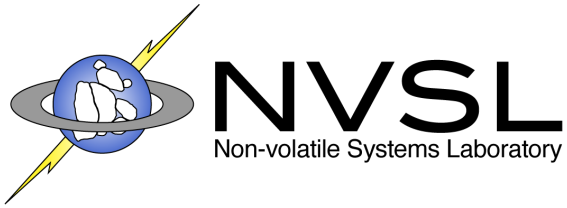
Near-Data Computation: It's Not (Just) About Performance

Steven Swanson

Non-Volatile Systems Laboratory

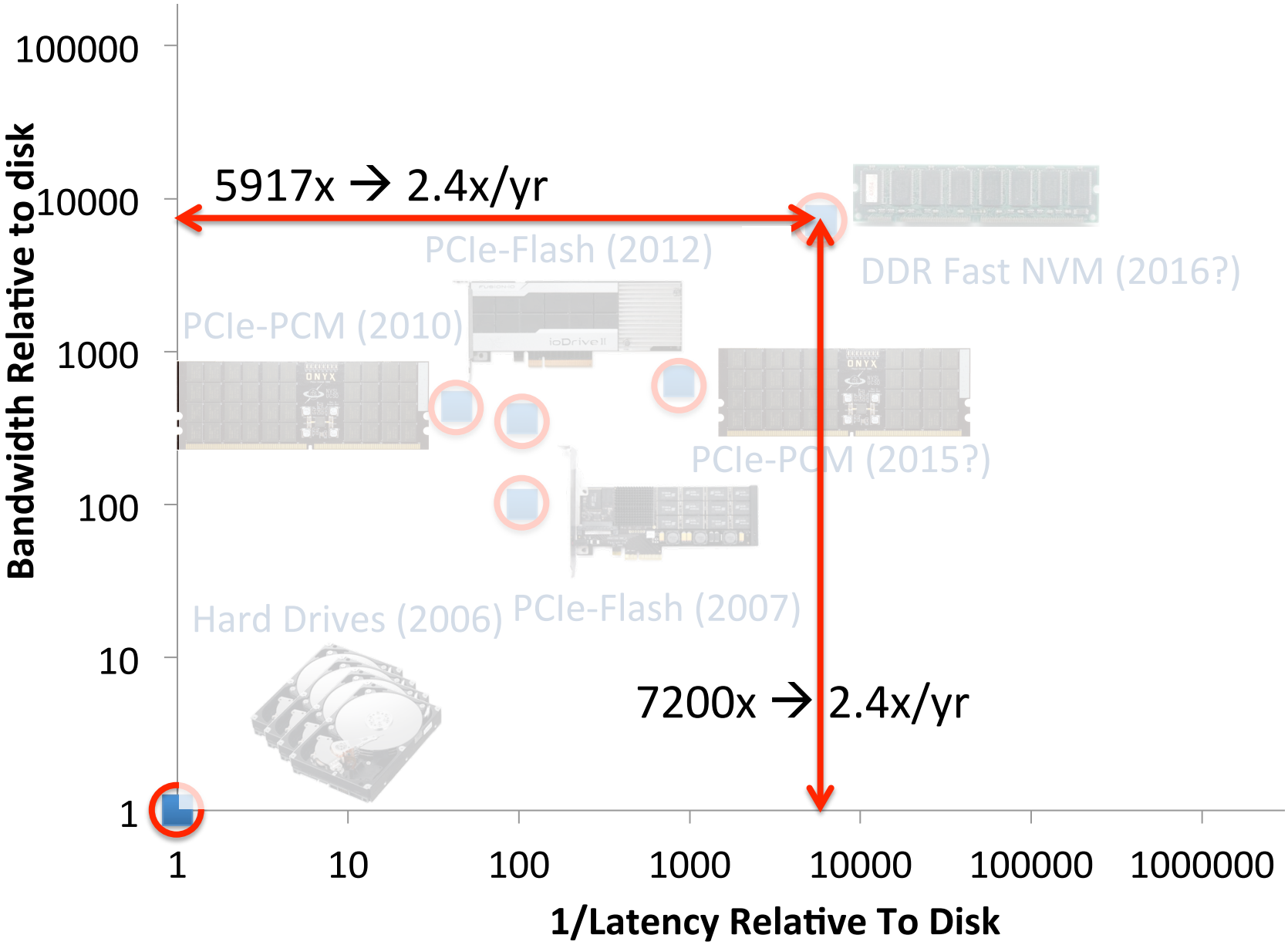
Computer Science and Engineering

University of California, San Diego



Solid State Memories

- NAND flash
  - Ubiquitous, cheap
  - Sort of slow, idiosyncratic
- Phase change, Spin torque MRAMs, etc.
  - On the horizon
  - DRAM-like speed
  - DRAM or flash-like density


100000
10000
1000
100
10
1
Bandwidth Relative to disk
5917x → 2.4x/yr
PCIe-Flash (2012)
DDR Fast NVM (2016?)
PCIe-PCM (2010)
PCIe-PCM (2015?)
PCIe-Flash (2007)
Hard Drives (2006)
7200x → 2.4x/yr
1
10
100
1000
10000
100000
1000000
1/Latency Relative To Disk

Programmability in High-Speed Peripherals

- As hardware gets more sophisticated, programmability emerges
  - GPUs – fixed function → Programmable shaders → full-blown programmability
  - NICs – Fixed function → MAC offloading → TCP offload
- Storage has been left behind
  - It hasn't gotten any faster in 40 years

Why Near-data Processing?

	Well-studied	Worth a Closer Look
Internal bandwidth ≫ interconnect bandwidth	✓	
Moving data takes energy	✓	
NDP compute can be energy efficient	✓	
Storage latencies ≤ Interconnect Latency	✓	✓
Storage latencies ≈ CPU latency	✓	✓
NDP compute can be trusted		✓
Improved Real-time capabilities		✓

Data Dependent Accesses

Trusted Computation

Semantic Flexibility

Diverse SSD Semantics

- File system offload/OS bypass
  - [Caulfield, ASPLOS 2012]
  - [Zhang, FAST 2012]
- Caching support
  - [Bhaskaran, INFLOW 2013]
  - [Saxena, EuroSYS 2012]
- Database transactions support
  - [Coburn SOSP 2013]
  - [Ouyang HPCA 2011]
  - [Prabhakaran OSDI 2008]
- NoSQL offload
  - Samsung's SmartSSD
  - [De FCCM 2013]
- Sparse storage address space
  - FusionIO DFS

Lessons Learned

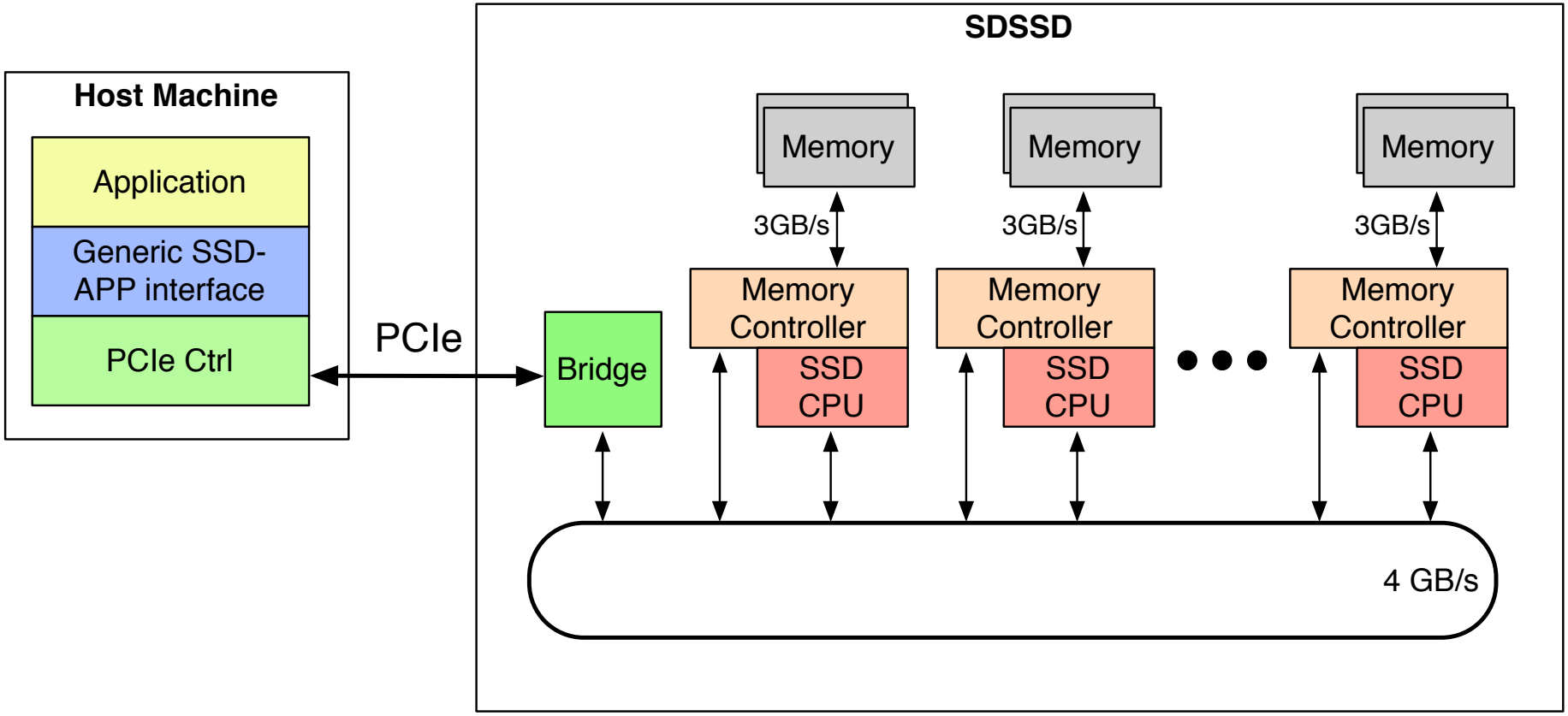
Implementation costs are high

Putting applications is uncertain

Our Goal: Make Programming an SSD Easy and Flexible.

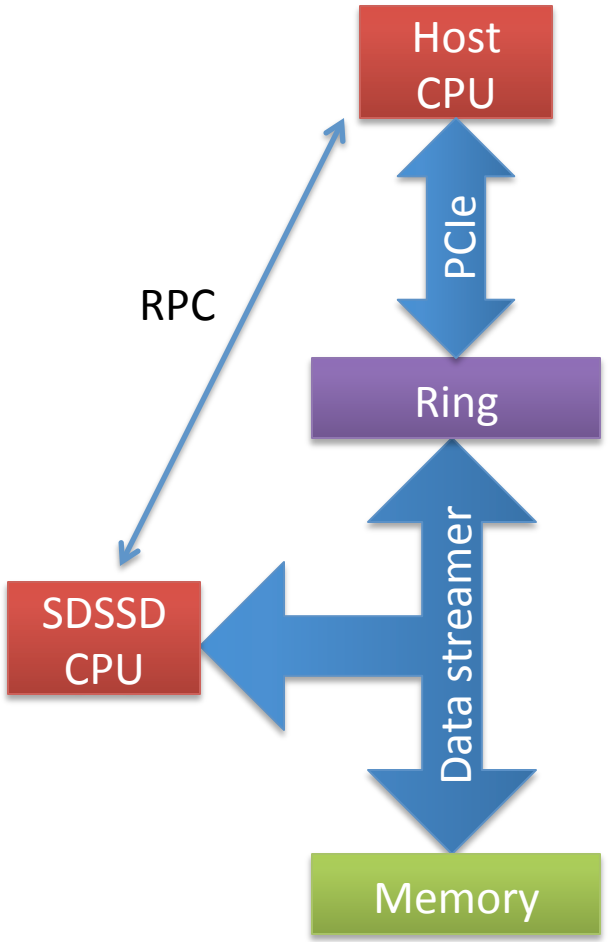
The Software Defined SSD

The Software-defined SSD



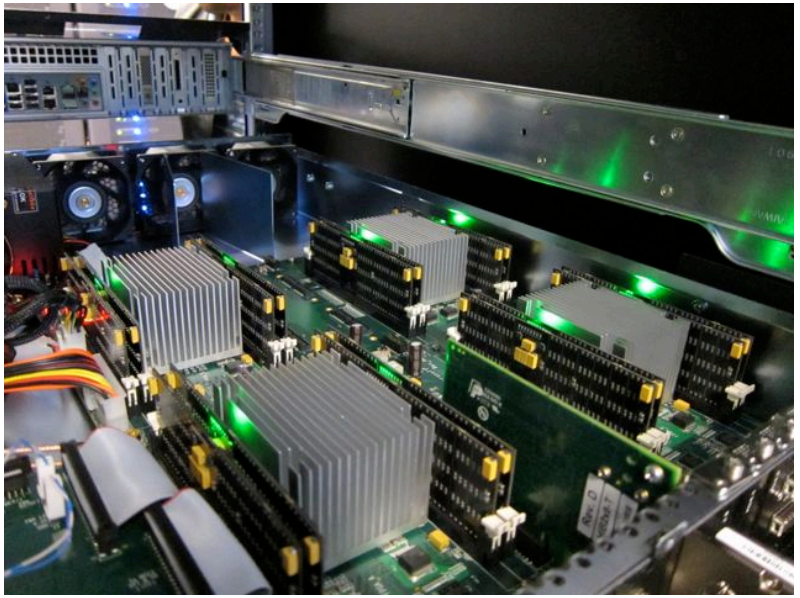
SDSSD Programming Model

- The host communicates with SDSSD processors via a general RPC interface.
- Host-CPU
  - Send and receive RPC requests
- SSD-CPU
  - Send and receive RPC requests
  - Manage access to NV storage banks



The SDSSD Prototype

- FPGA-based implementation
- DDR2 DRAM emulates PCM
  - Configurable memory latency
  - 48 ns reads, 150 ns writes
  - 64GB across 8 controllers
- PCIe: 2 GB/s, full duplex



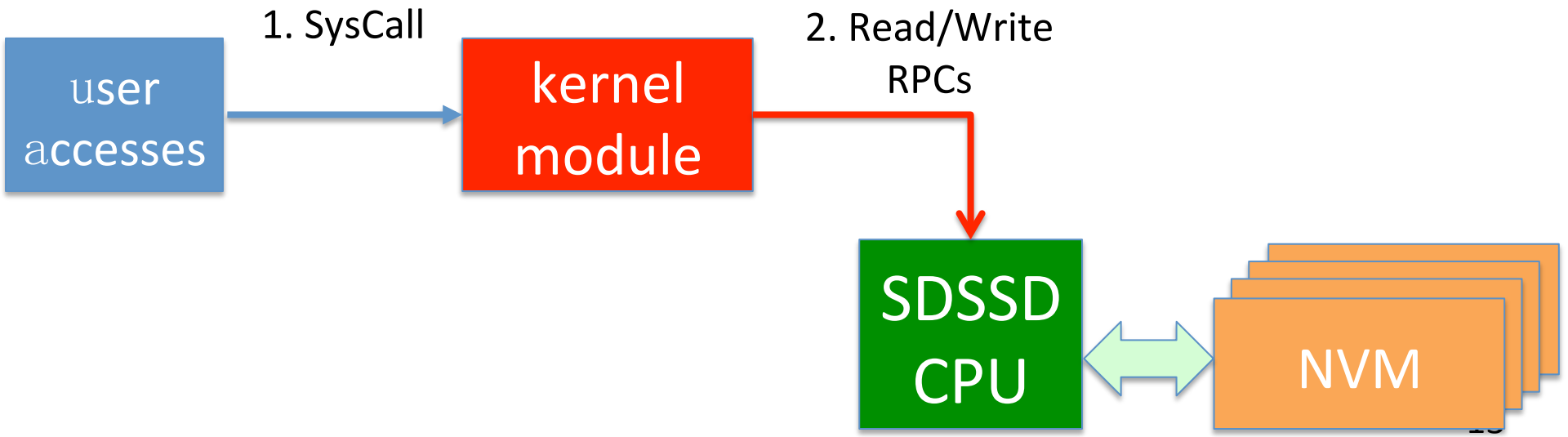
SDSSD Case Studies

- Basic IO
- Caching
- Transaction processing
- NoSQL Databases

Basic IO

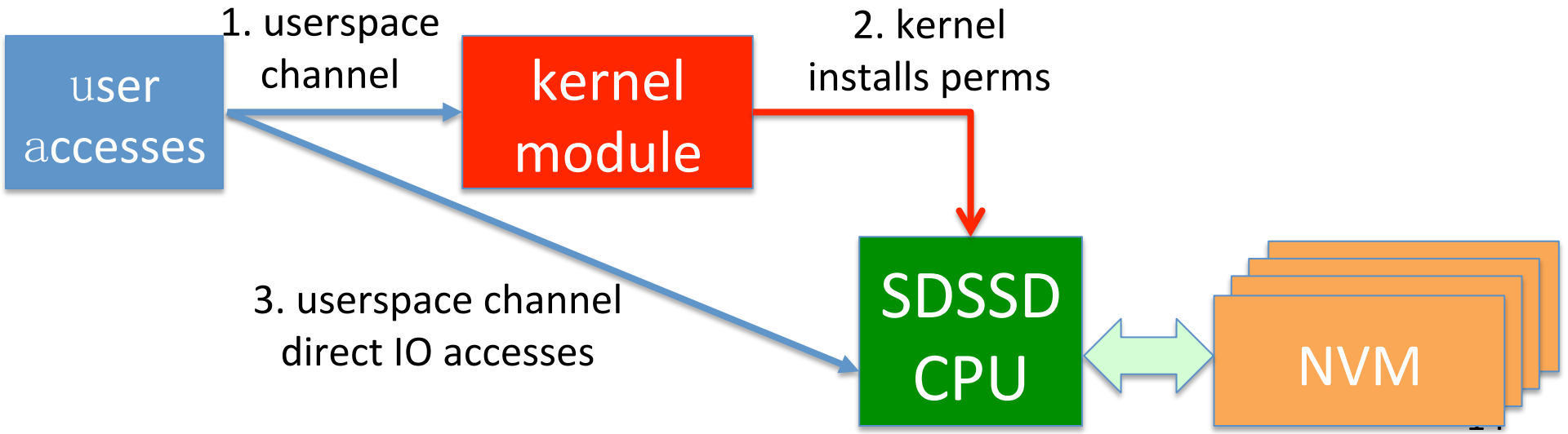
Normal IO Operations: Base-IO

- Base-IO App provides read/write functions.
- Just like a normal SSD.
- Applications make system calls to the kernel
- The kernel block driver issues RPCs

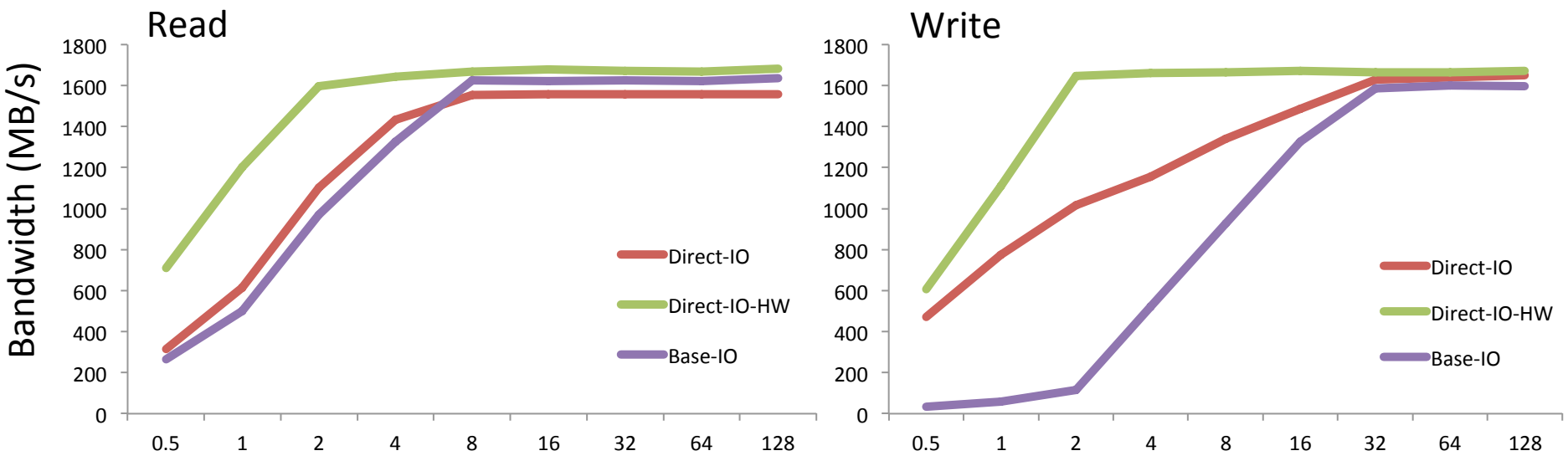


Faster IO: Direct access IO

- OS installs access permissions on behalf of a process.
- Applications issue RPCs directly to HW
- The App checks permission



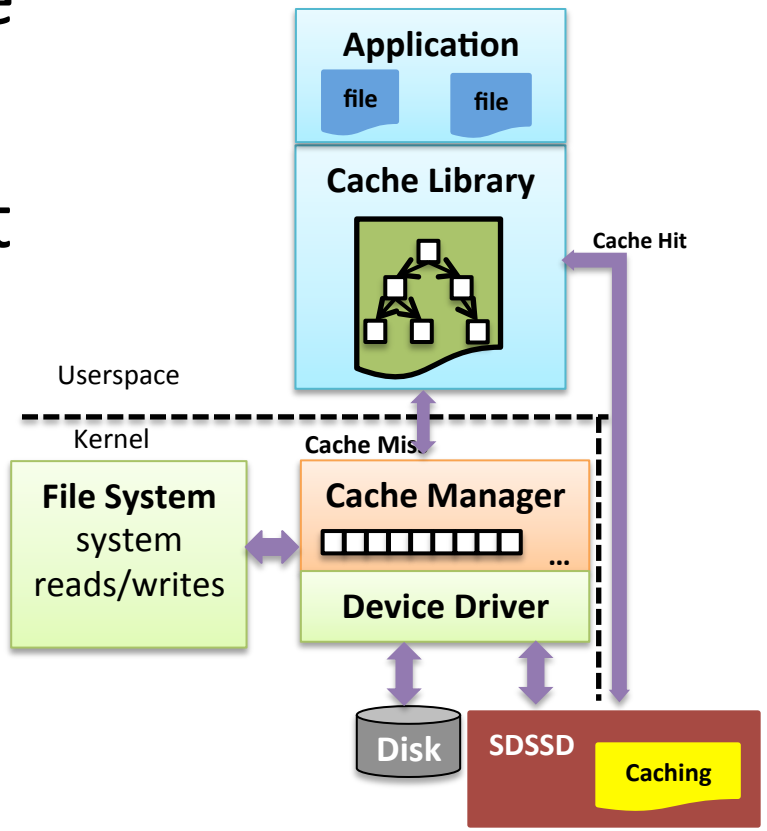
Preliminary Performance Comparison



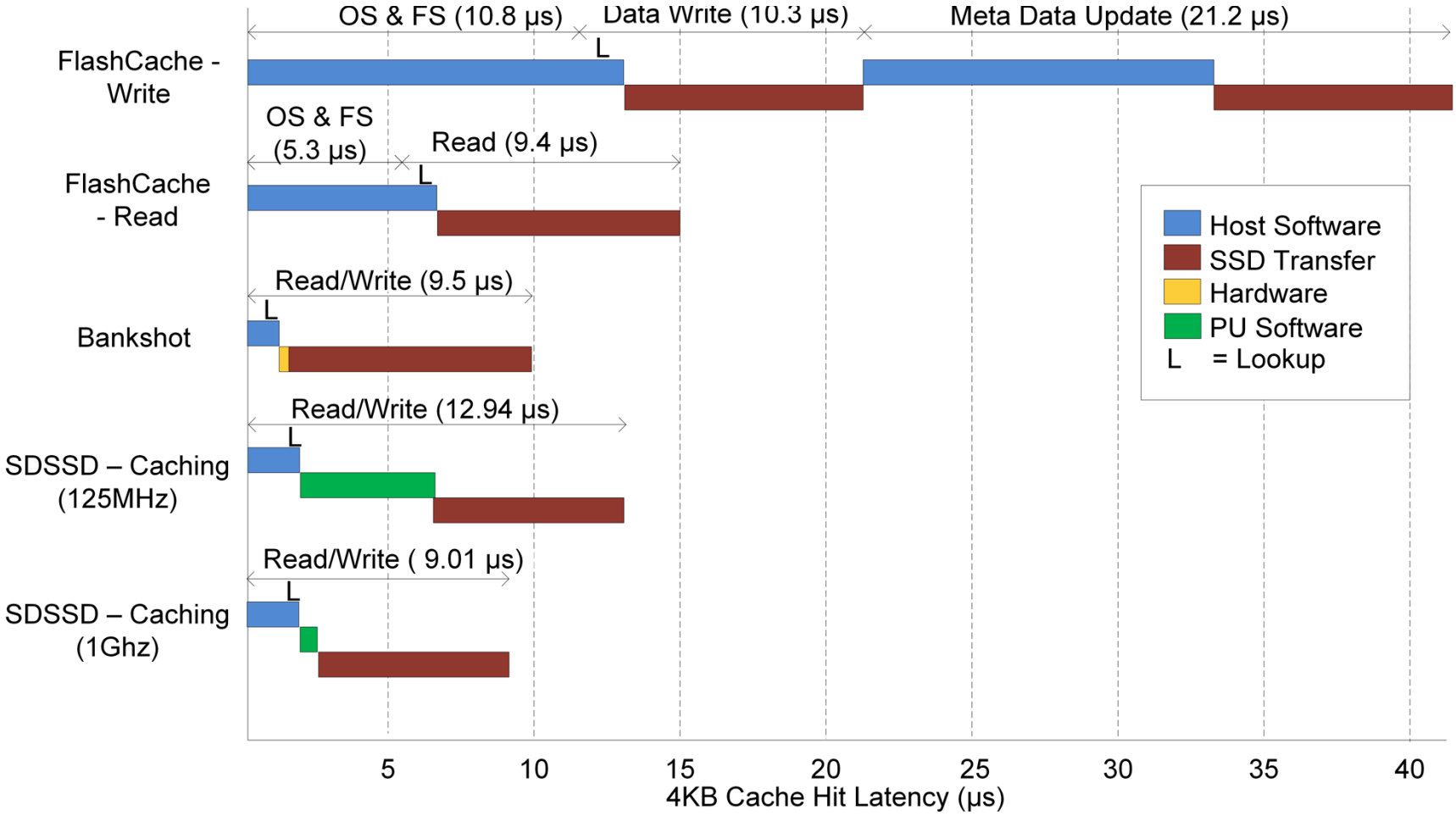
Caching

Caching

- NVMs will be caches before they are storage
- Cache hits should be as fast as Direct-IO
- Caching App
  - Tracks dirty data
  - Tracks usage statistics



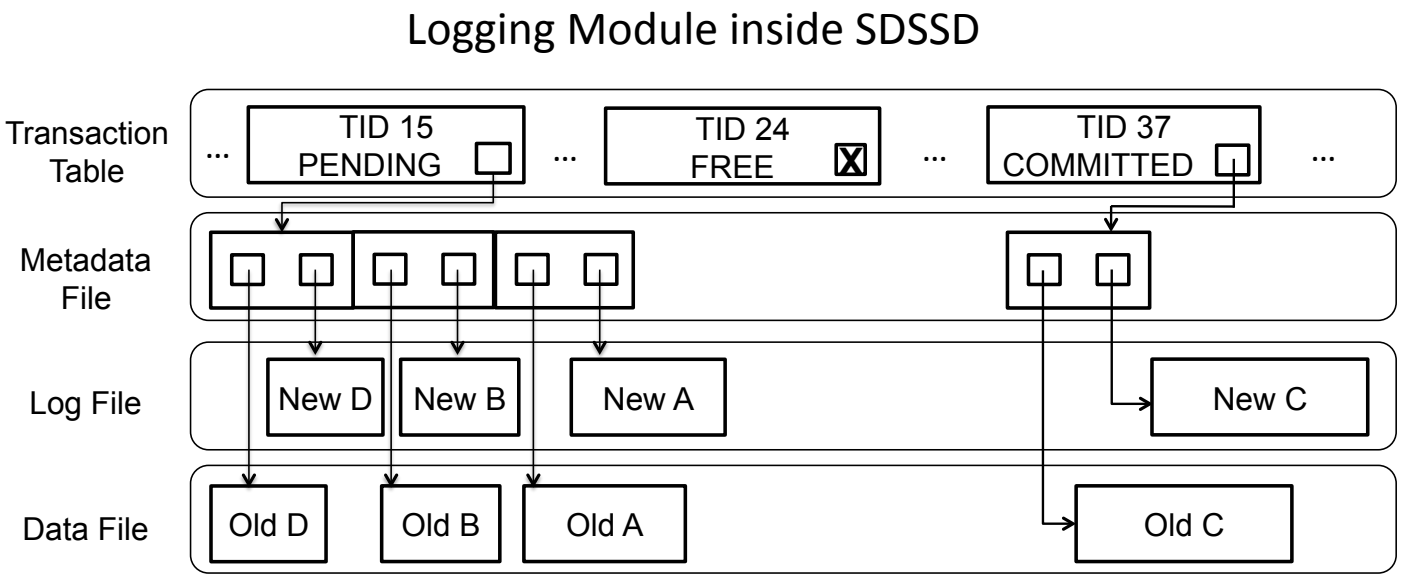
4KB Cache Hit Latency



Transaction Processing

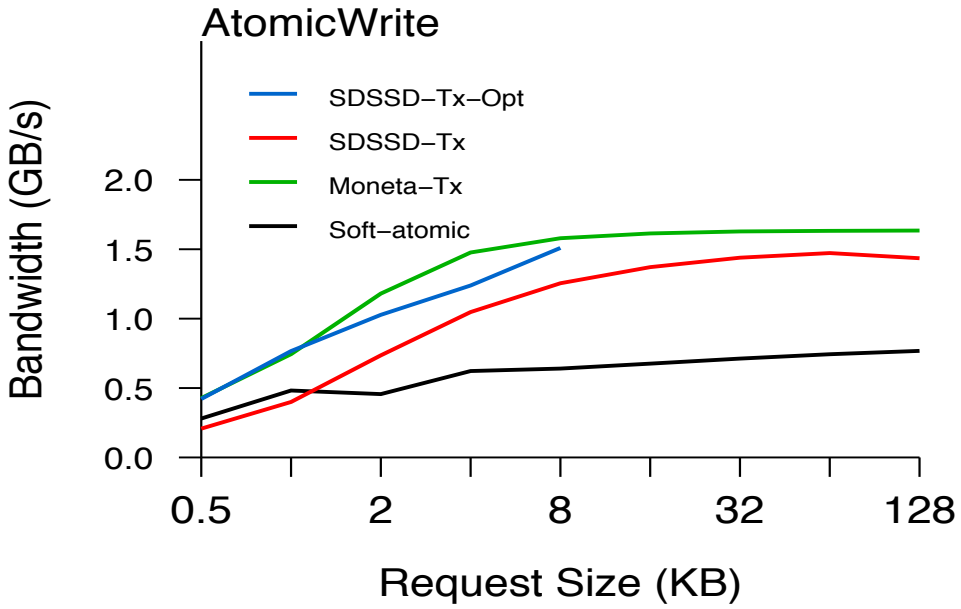
Atomic Writes

- Write-ahead logging guarantees atomicity
- New interface for atomic operations
  - LogWrite(TID, file, file_off, log, log_off)
- Transaction commit occurs at the memory interface → full memory bandwidth



Atomic Writes

- SDSSD atomic-writes outperform pure software approach by nearly 2.0x
- SDSSD atomic-writes achieve comparable performance to a pure hardware implementation

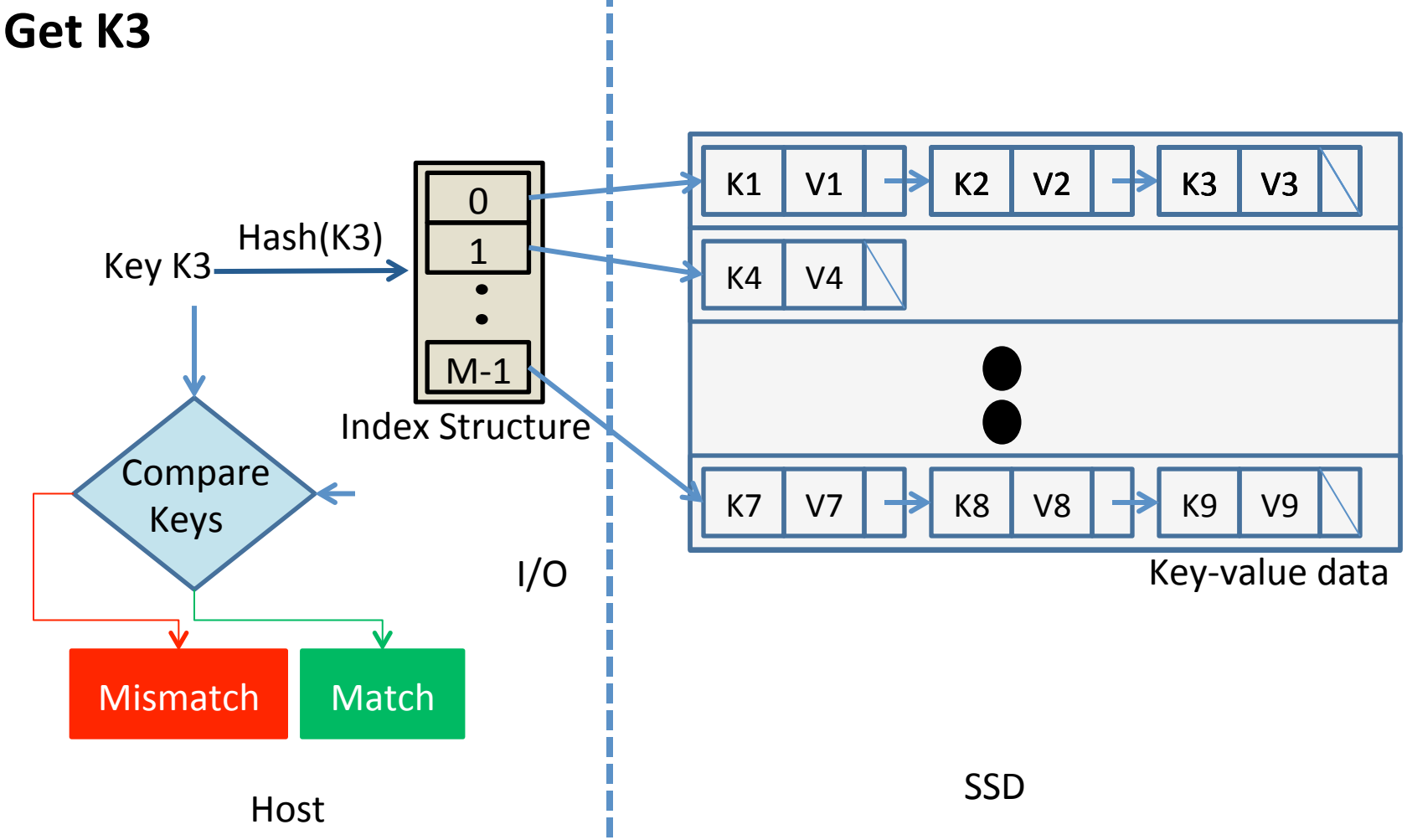


Key Value Store

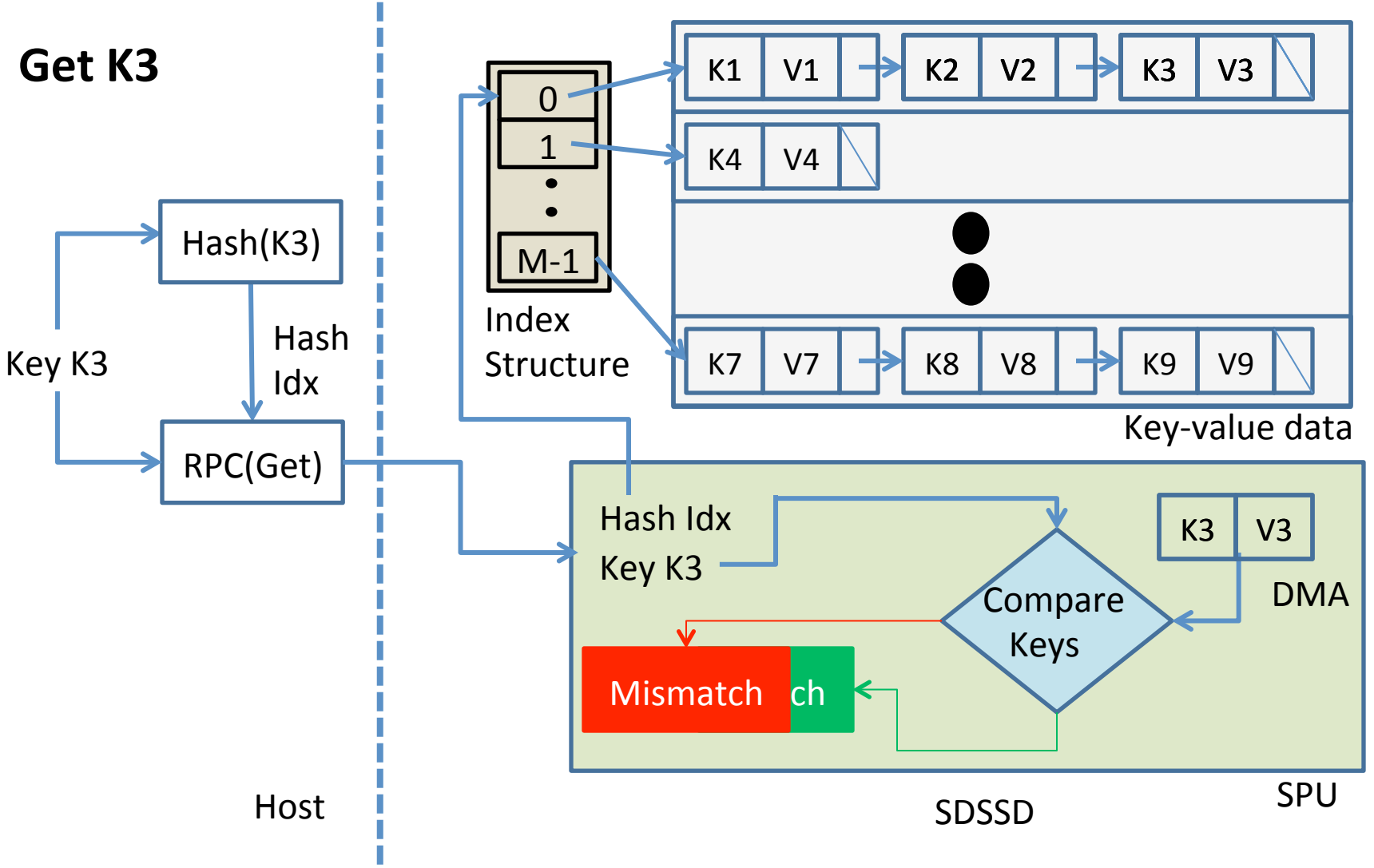
Key-value store

- Key-value store is a fundamental building block for many enterprise applications
- Supports simple operations
  - *Get* to retrieve the value corresponding to a key
  - *Put* to insert or update a key-value pair
  - *Delete* to remove a key-value pair
- Open-chaining hash table implementation.

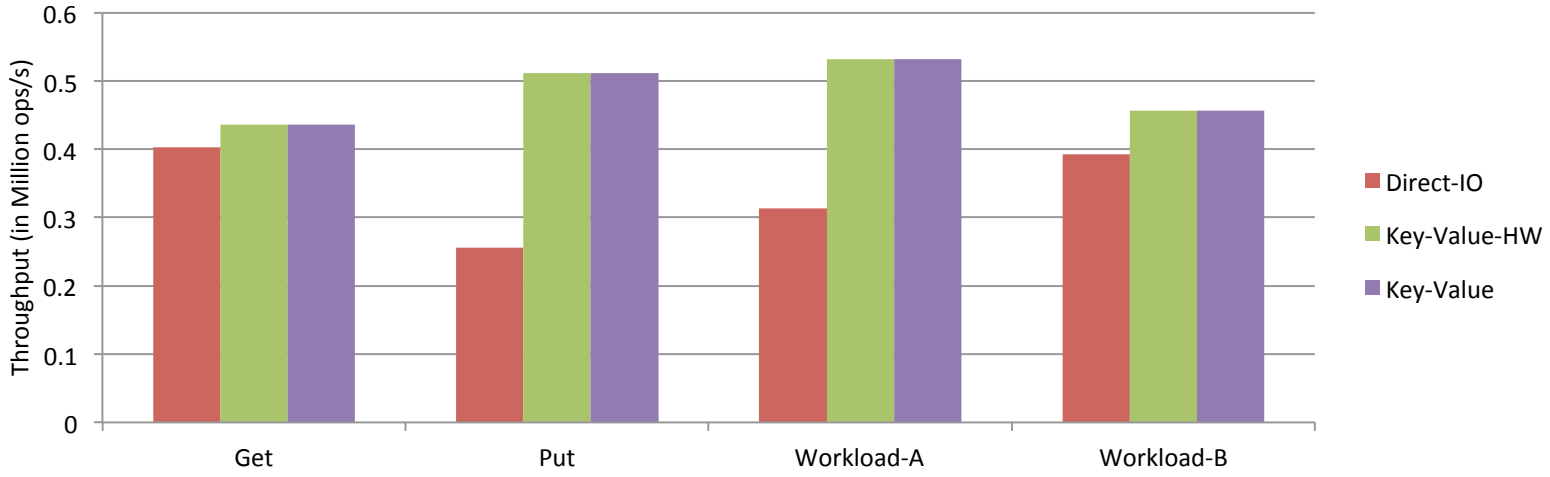
Direct-IO based Key-Value Store

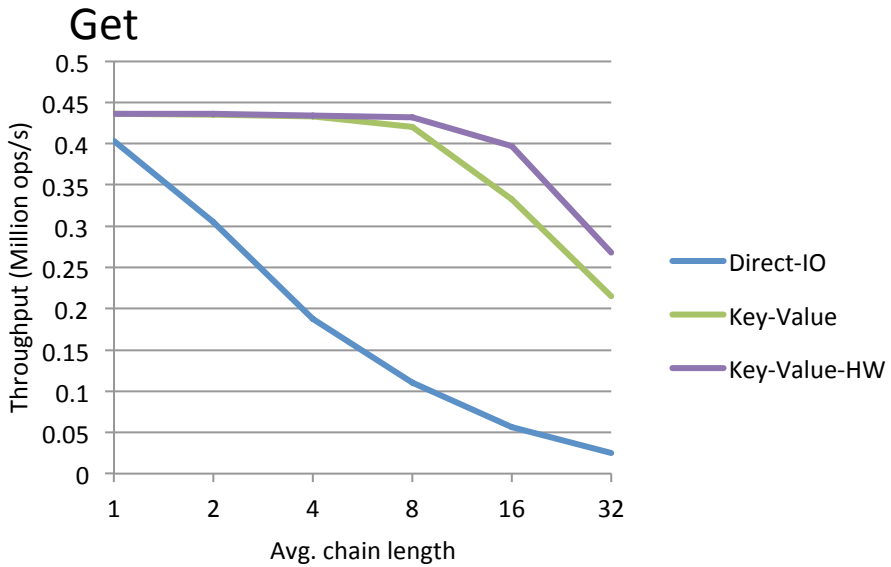


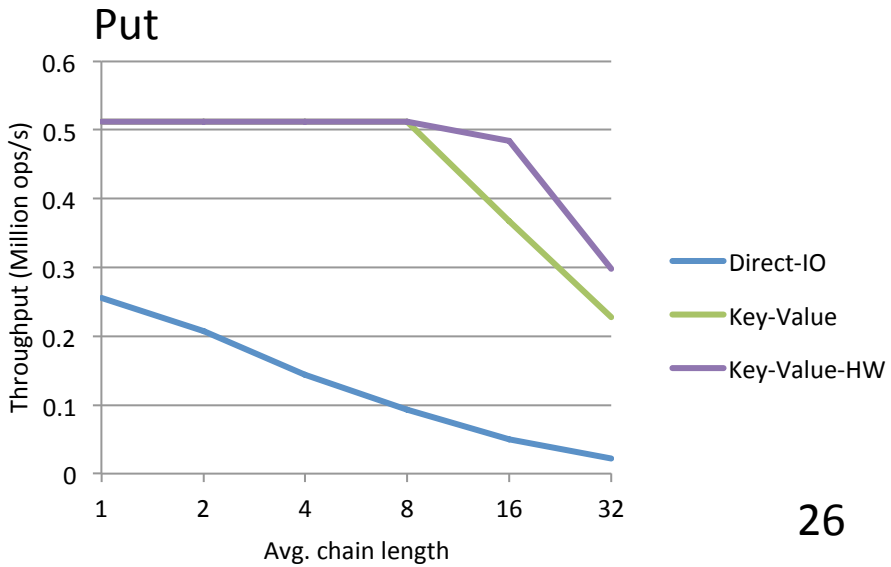
SDSSD Key-Value Store App



MemcacheDB performance







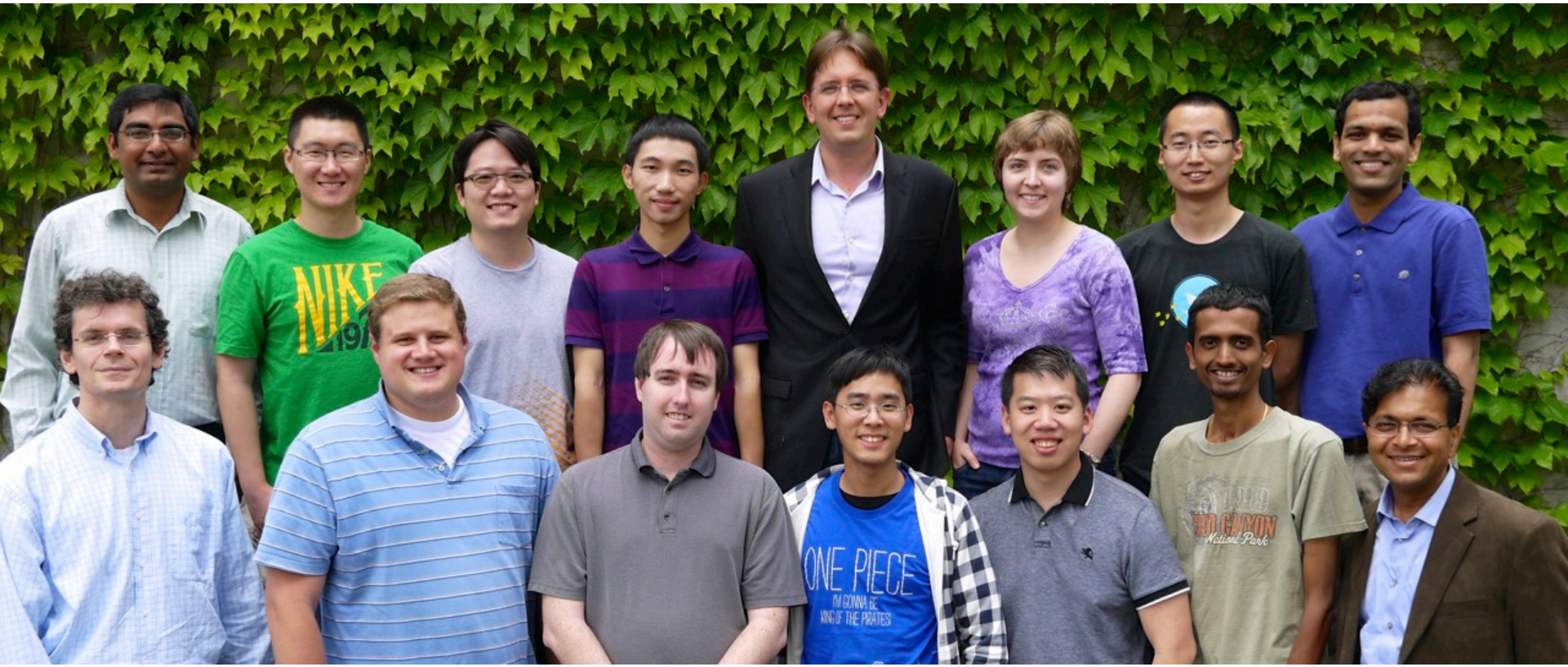
Ease-of-Use

Description	Name	Fixed-function		SDSSD	
		LOC (Verilog)	Devel. Time (Person-months)	LOC (C)	Devel. Time (Person-months)
Simple IO operations [8].	`Base-IO`	n/a	n/a	1500	1
Virtualized SSD interface with OS bypass and permission checking [9].	`Direct-IO`	3000	36	1524	1.2
Atomic writes tailored for scalable database systems based on [12].	`Atomic-Write`	1372	18	901	1
Direct-access caching device with hardware support for dirty data tracking [6]	`Caching`	1336	12	728	1
SSD acceleration for the persistent key-value store, MemcacheDB [11].	`Key-Value`	1750	6	834	1

Conclusion

- New memories argue for near-data processing
  - But not just for bandwidth!
  - Trust, low-latency, and flexibility are critical too!
- Semantic flexibility is valuable and viable
  - 15% reduction in performance compared to fixed-function implementation
  - 6-30x reduction in implementation time
  - Every application could have a custom SSD

Thanks!



NVSL
Non-volatile Systems Laboratory

Questions?