A Software-Defined Flash Architecture Tuned for Hyperscale

Scott Stetzer, Vice President
KIOXIA America, Inc.

KIOXIA

Agenda

- Making Flash Software-Defined
- Software-Enabled Flash™ Technology Overview
- Solving Hyperscale Pain Points with Software-Defined Flash
- Getting Involved with Software-Enabled Flash OSS project

The next evolution of flash is Software-Defined

Software-Enabled Flash

The next evolution of flash is Software-Defined

- Fine-grained data placement
- Workload isolation
- Write amplification reduction
- Latency outcome control
- Advanced queueing methods
- Die-Time I/O prioritization
- Customized protocols
- Open source API and SDK

HARDWARE

- **Flash low-level management**
  - tProg,
  - tErase, etc.
  - NAND-level ECC
  - Copy offload
- **Shared resource management**
  - Flash channels
  - I/O queues and queuing (3) engines
  - Buffer memory

SOFTWARE

- **Data placement**
  - Keep correlated data in the same super block
- **FTL (or not) and garbage collection**
- **I/O Prioritization**
  - Application/hypervisor determines priority
  - Per-I/O granularity
- **Protocol**
  - Block, FDP, ZNS, object, ???

Software Defined unlocks FLASH VALUE

KIOXIA announces
the first available hardware implementing
Software-Enable Flash™

- E1.L Form Factor
- SEF Native FW
- 32TB QLC

- Die level HW Isolation
- SW defined QOS Domains
  - Finer granularity for placement/isolation
  - GC inside QoS domain
  - Encryption per domain
- pSLC enabled via SEF API

Developer samples available

SEF
SOFTWARE-ENABLED FLASH™

An Open Source Software Project
available from the Linux Foundation

Application **P**rogramming **I**nterface

Software **D**evelopment **K**it

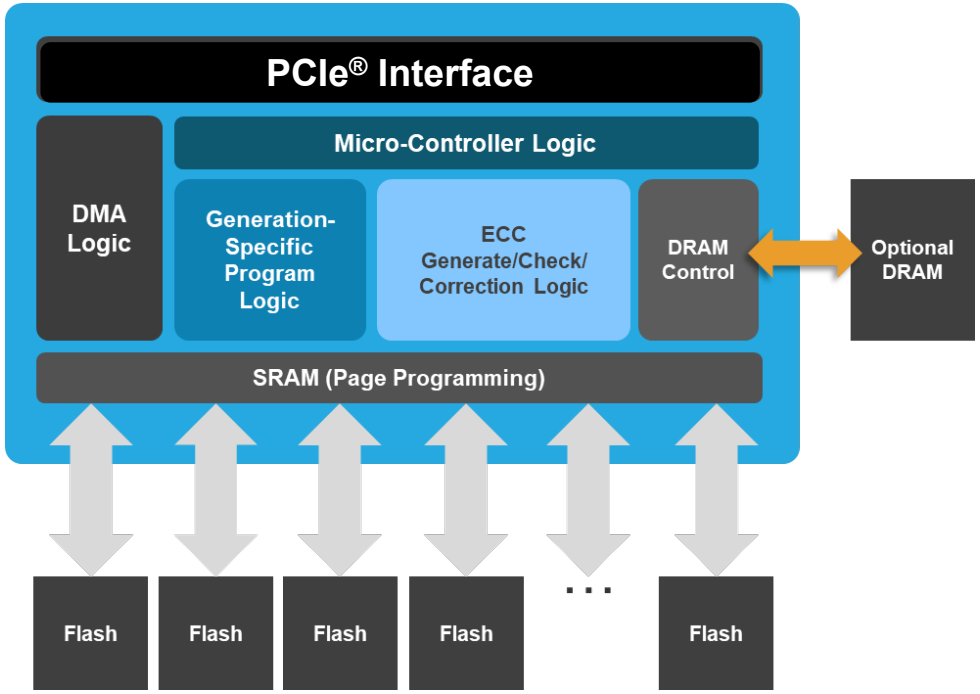
INCLUDES FULL SOURCE CODE

*** Partners and Vendors Welcome ***
SEF needs you! - Join & contribute to the project

The Hardware

Making Flash Memory Software-Defined

Configurable, Optimized Hardware



Vendor Configurable

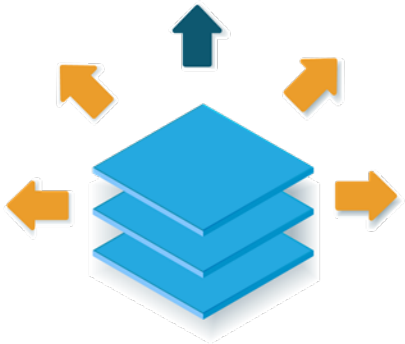
- Flash Technology
- Flash Interface Protocol
- DRAM Buffer vs. Host Buffer vs. Mixed
- Form Factor
- Power Limits

PCIe is a registered trademark of PCI-SIG.

Optimized Hardware Features for Software-Defined

Advanced queueing control

Control latencies at the flash operation level



Flash abstraction & management

Simplify porting between flash generations, vendors, and technologies

Low-level hardware partitioning & isolation

Maximum performance decoupling between critical workloads

Advanced on-board copy offload

Minimize CPU and bus management for data movement operations

Flash vendors focus on optimizing flash memory management

Data Placement and Isolation

HARDWARE
Virtual Device

Die-level isolation

Physical separation of data

User-configurable at deployment

SOFTWARE
Quality of Service Domain

Workload-level isolation

Separation of data by super block

Placement ID control

Isolated garbage collection, overprovisioning and encryption

Hardware Support for Software-Defined Queueing

Massively parallel I/O queues

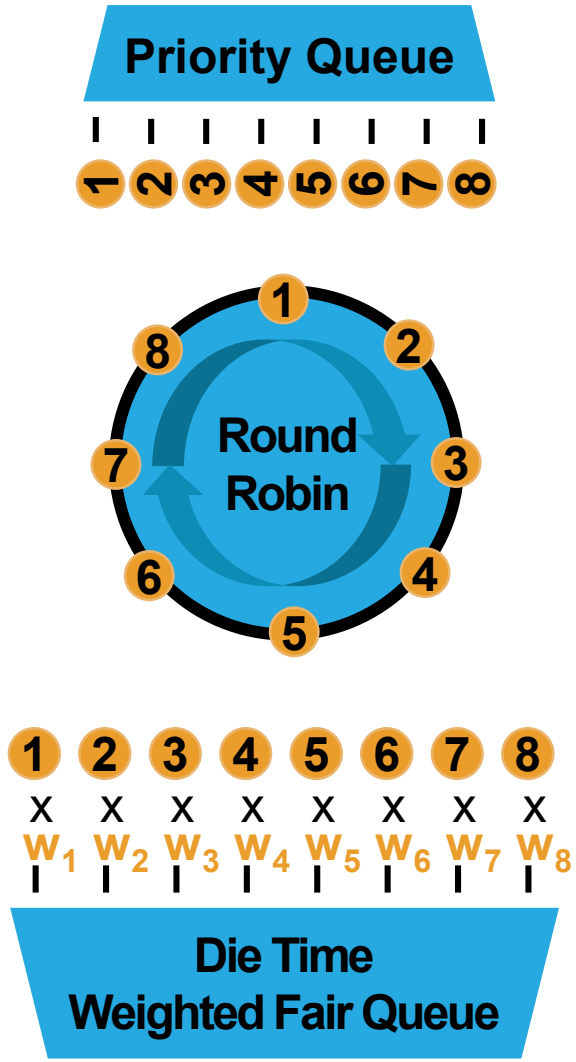
- Total separation of read and write paths
- Reads never blocked by writes
- Minimize head-of-queue blocking

Hardware-enforced I/O prioritization

- Multiple, programmable scheduling modes
- Application controlled

Die-Time Weighted Fair Queueing

- Individual erase, program, read, copy weights



The Open Source Software

Open Source API and SDKs

CLI with Python® Interpreter	FIO Test Tool	Reference Virtual Device Drivers	Reference Flash Translation Layer (FTL)
Device orchestration and management	Ported to SEF for fast and easy experimentation	No code changes to evaluate SEF in multi-tenant mode	Common block interface to SEF applications

High-Level SDK

Low-Level API

* Python is a registered trademark of the Python Software Foundation.

Software Development Kit, BSD Licensed

C-language based

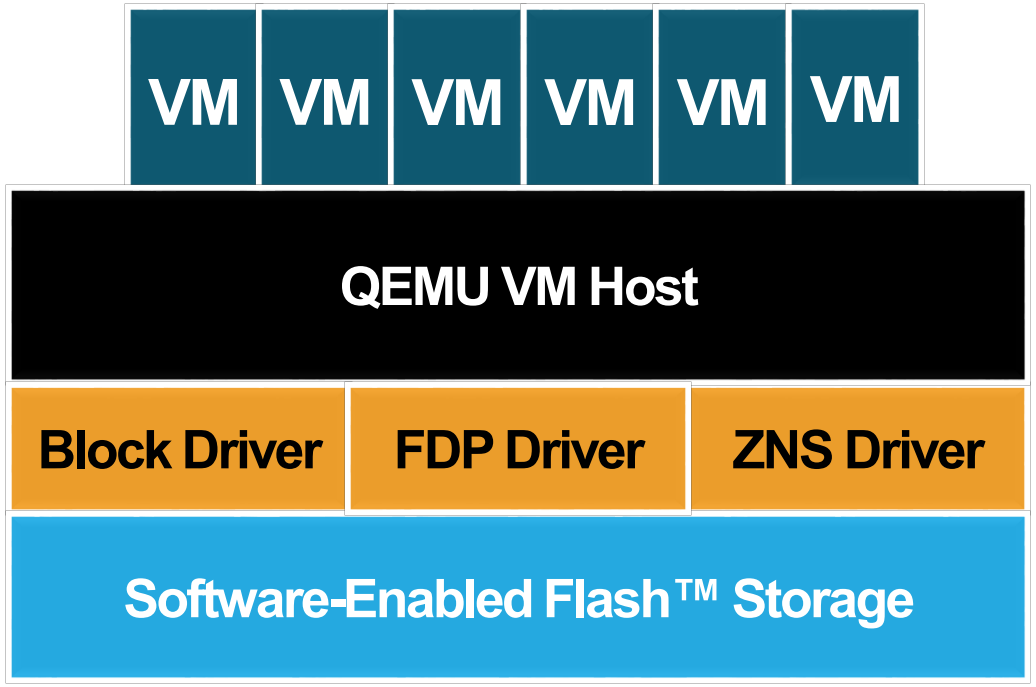
- 32 + 64 Bit with multiple CPU architectures
- Modern Linux® Kernels
- Event Driven Callbacks
- Thread Safe, Lockless Operation
- Modular, Built for Customization

Full host source code

- SEF Reference FTL
- SEF CLI
- SEF FIO
- SEF virtualized device driver
- Kernel driver / IO_URING enhancements

Reference Virtual Drivers

- No guest code changes needed
  - Fast, easy evaluation and tuning
- Customize overprovisioning per VM
  - Tune flash for write- or read-heavy workloads
- ZNS, FDP and block-based VMs on one drive
  - Simultaneously, optimized per workload
- Full data, performance isolation, queueing control
  - Orchestration layer controlled



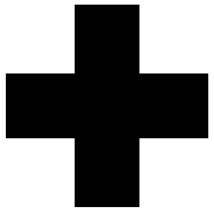
Solving Cloud & Hyperscale Pain Points with Software-Enabled Flash

Problem: Isolating Tenants and Workloads at Scale

10s to 100s of tenants in a single server

- Virtual machines
- Containers
- Bare-metal applications

…all with different performance needs



Ephemeral storage on local flash memory

- Shared resources = **Major Contention**

Unpredictable latency, latency spikes

- Inconsistent performance from “noisy neighbors”
- **inefficient** overprovisioning of servers and storage

Solving Tenant and Workload Isolation

HARDWARE

Virtual Devices

- Complete physical isolation at flash die level for critical workloads

Advanced Queueing

- Separate flash queues to avoid head-of-line blocking

SOFTWARE

Quality of Service Domains

- Software level isolation to separate workloads and data at super block level
- Isolated garbage collection within each QoS domain
- Apply unique encryption algorithms per QoS domain for tenants or workloads
- “Instant reclaim”

SEF Hardware Isolation

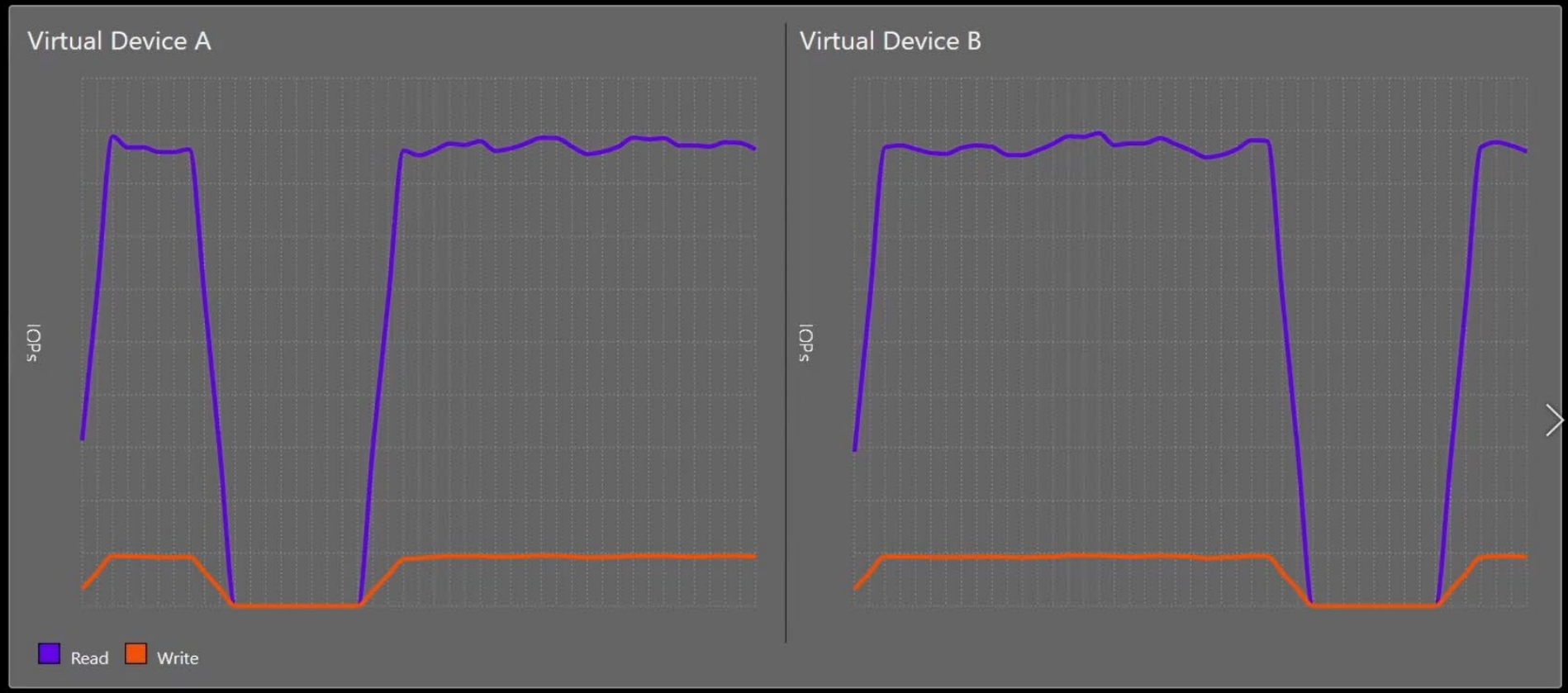
Using Self Driving Interface: Running

KIOXIA

Stopping Demo

- [x] Enable Virtual Device A
- [x] Enable Virtual Device B

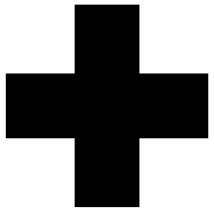
Software-Enabled Flash isolates workloads from each other while providing application-controlled latency outcomes



Problem: Prioritizing I/O at Scale

Multiple I/O streams running on a **single flash storage device**

- Drive unaware of different streams
- Data combined on-drive as-received



Unrelated data = **unrelated lifetimes**

- Garbage collection can be forced by unrelated streams
- Workload termination can invalidate large chunks of data

Inefficient flash utilization

- Lower overall storage performance
- Latency impacted by IO collision
- Unpredictable performance requiring overprovisioning
- Wasted flash lifetime on unneeded garbage collections

Solving IO Prioritization

HARDWARE

Flash Management

- Die and block level allocation
- Per flash queueing control

Advanced Queueing

- Discrete control of queuing policies and individual IO

SOFTWARE

Data Management

- Application-controlled placement ID requests
- IO and queue weight policy defined by applications

Quality of Service Domains

- Per I/O queueing priority
- Separation of data and overhead per-workload

SEF Weighted Fair Queueing

Using Self Driving Interface: Running

KIOXIA

Stopping Demo

Die Time Balance

Read Write

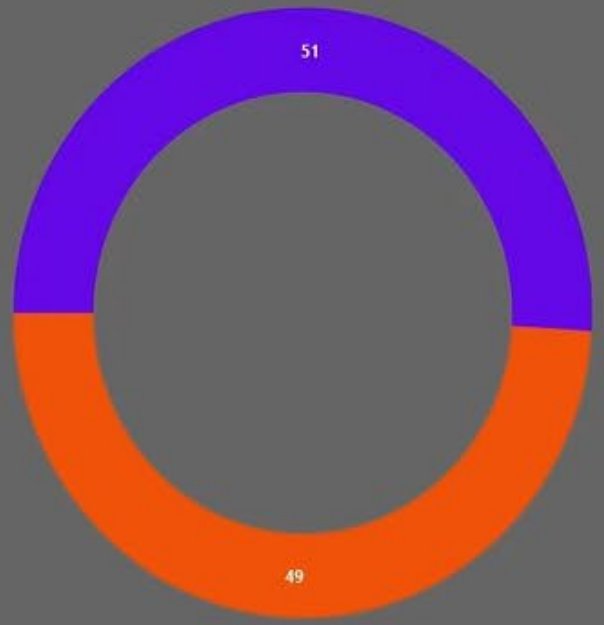
90/10 50/50 10/90

Enable Read Enable Write

Software-Enabled Flash allows control over read and write priorities (die times) while preserving the full performance of the device

R/W IOPS Ratio: 9.27

Percent Die Time



Read Write

Percent Die Time History

Problem: Innovation for TODAY and TOMORROW

Servers and applications in use today are deployed for **many years.**

- Immutable hardware
- Choose today, live with for ages

+

Workloads change over time.
The hardware doesn't

- Legacy defined storage devices
- With fixed interface protocols
- and locked overprovisioning levels, etc.

=

Locked HW can tend to slow down innovation and potentially delay deployment of new applications

Innovate at the Speed of Software

HARDWARE

Flash Management

- HW fully **abstracted** by the API
- Each device handles low level flash activity for read, write, erase
- Host offload of critical functions (copy, GC, WL) under SW control
- Endurance managed by HW
- ECC managed in HW
- Multi-vendor support

SOFTWARE

Software-Defined

- Any storage protocol
  -Block, FDP, ZNS, Object, etc
- QoS domains enable deployment of multi-protocols by workload or tenant

Reference FTLs

- Full source code / ready to customize

Simultaneous Multiple Software-Defined Protocols with Isolation

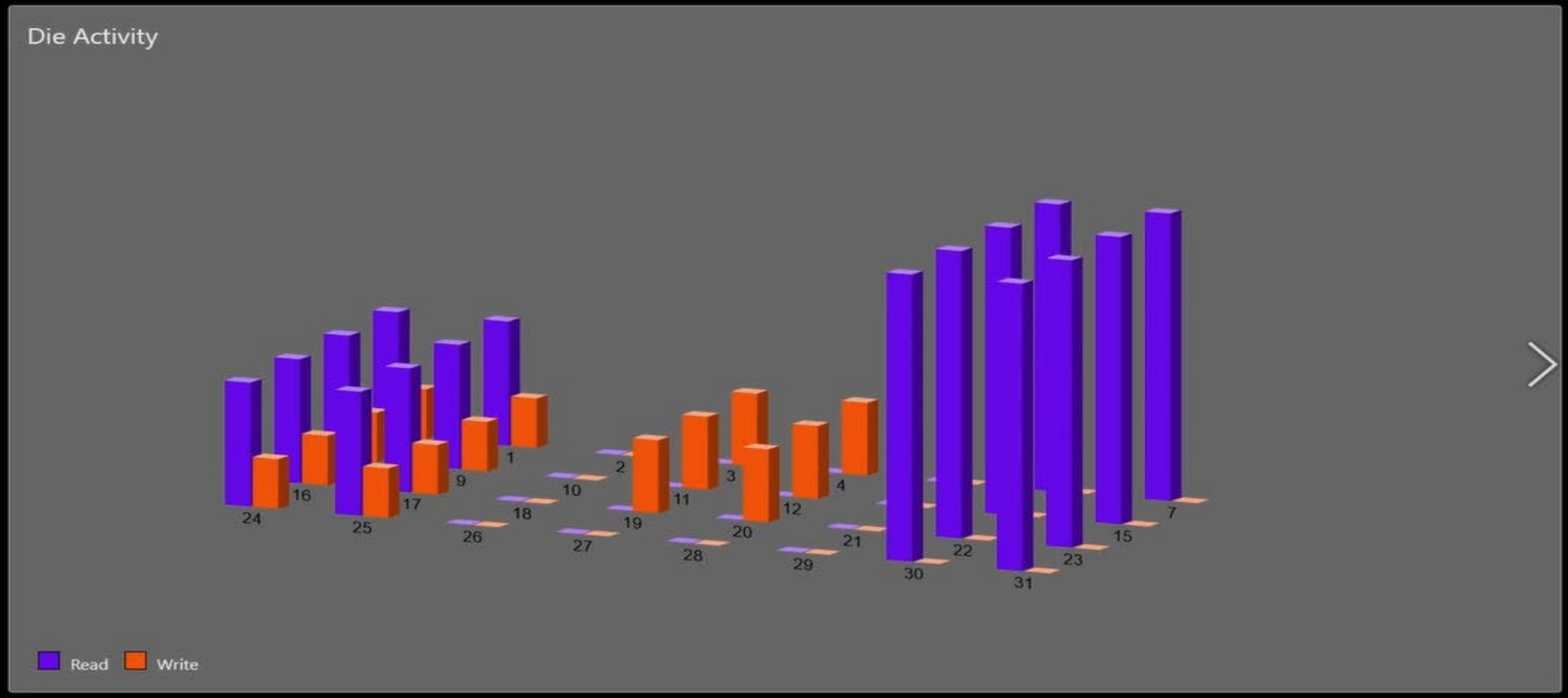
SEF
Using Self Driving Interface: Running

KIOXIA

Stop Demo

- [x] Read/Write Workload (FDP)
- [x] Write Workload (ZNS)
- [x] Read Workload (Block)

Software-Enabled Flash supports standard and application-defined protocols while isolating workloads to individual flash dies for complete control and isolation



All three workloads are isolated using separate flash dies

Software-Enabled Flash™ Project

An Open, Linux Foundation® managed organization

Open Source, Vendor Neutral

Managed under the Linux Foundation

Open Source Governance Processes

Open Technical Steering Committee (TSC) meetings

Specification built for multiple implementations

Flash technology independent
Controller technology independent

Open Source API

Available on GitHub®

SDK coming soon

Antitrust Policy

- Linux Foundation meetings involve participation by industry competitors, and it is the intention of the Linux Foundation to conduct all of its activities in accordance with applicable antitrust and competition laws. It is therefore extremely important that attendees adhere to meeting agendas, and be aware of, and not participate in, any activities that are prohibited under applicable US state, federal or foreign antitrust and competition laws.
- Examples of types of actions that are prohibited at Linux Foundation meetings and in connection with Linux Foundation activities are described in the Linux Foundation Antitrust Policy available at http://www.linuxfoundation.org/antitrust-policy. If you have questions about these matters, please contact your company counsel, or if you are a member of the Linux Foundation, feel free to contact Andrew Updegrove of the firm of Gesmer Updegrove LLP, which provides legal counsel to the Linux Foundation.

The Software-Enabled Flash™ Project Needs You!

https://softwareenabledflash.org

Definition of capacity: KIOXIA defines a megabyte (MB) as 1,000,000 bytes, a gigabyte (GB) as 1,000,000,000 bytes and a terabyte (TB) as 1,000,000,000,000 bytes. A computer operating system, however, reports storage capacity using powers of 2 for the definition of 1GB = 2^30 = 1,073,741,824 bytes and therefore shows less storage capacity. Available storage capacity (including examples of various media files) will vary based on file size, formatting, settings, software and operating system, such as Microsoft Operating System and/or pre-installed software applications, or media content. Actual formatted capacity may vary.

All company names, product names and service names may be trademarks of their respective companies.

Images are for illustration purposes only.