SEF

SOFTWARE-ENABLED FLASH™

Building Applications with Software-Enabled Flash™

Rory Bolt, Software-Enabled Flash Project TSC Chair

Agenda

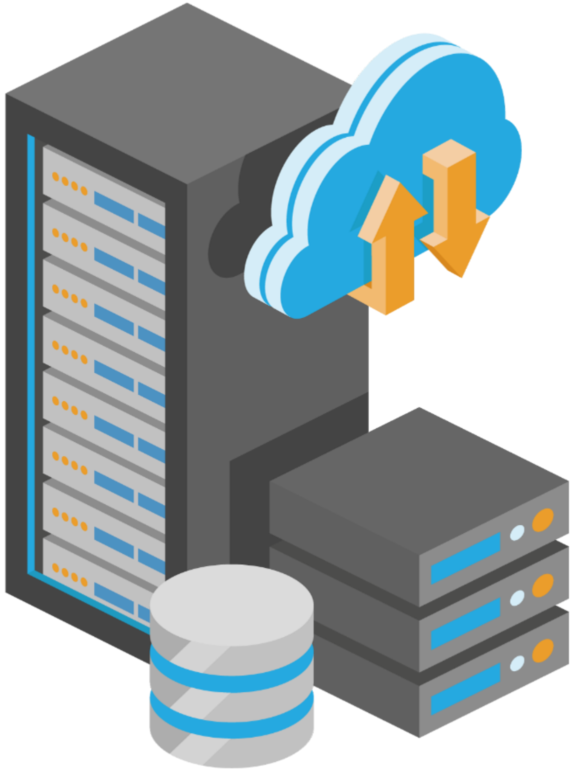
- Software-Enabled Flash™ Concepts
- The Software Stack
- Software Development Kit
- Flash Translation Layer
- I/O Through the FTL Modules
- Future Ideas
- Get Involved

A Different Way of Thinking About Flash

- Drop the HDD paradigm
- Expose full parallelism of flash
- Explicit controls over isolation, queueing modes
- Application defined latency outcomes

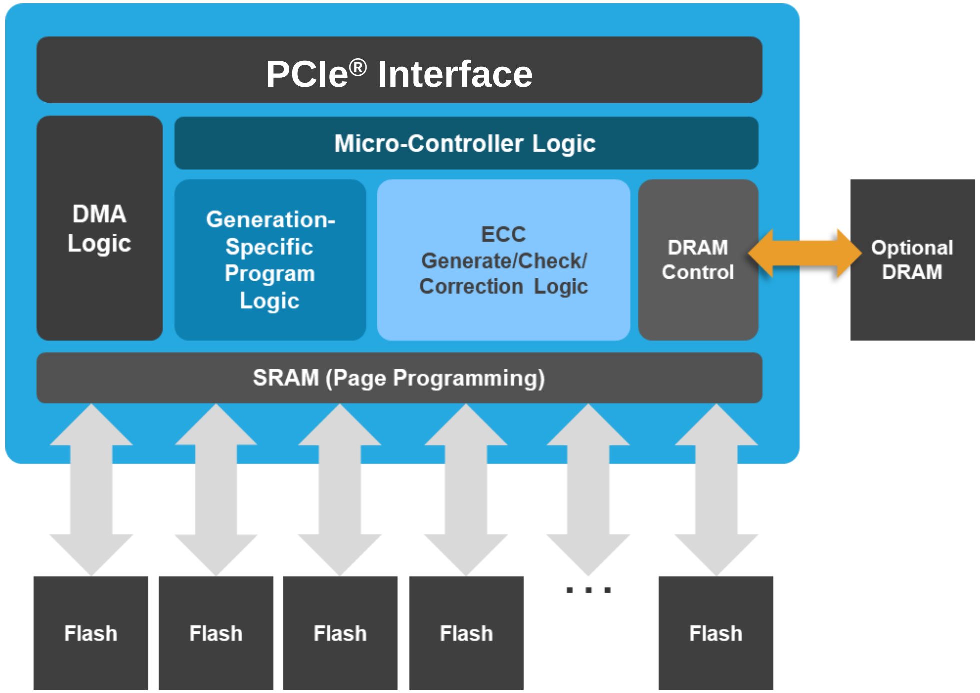
Features for Storage Developers

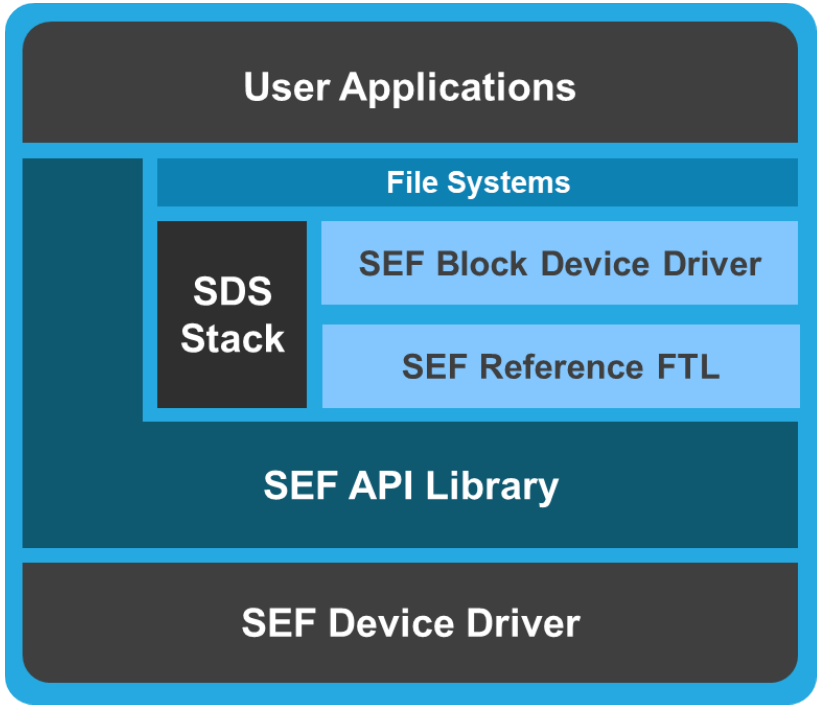
- Hardware and software-based isolation
- Advanced queueing
- Die-Time Weighted I/O prioritization
- Open source, BSD 3-clause for API and SDK



Explicit requests for flash behavior

Custom Hardware and Software



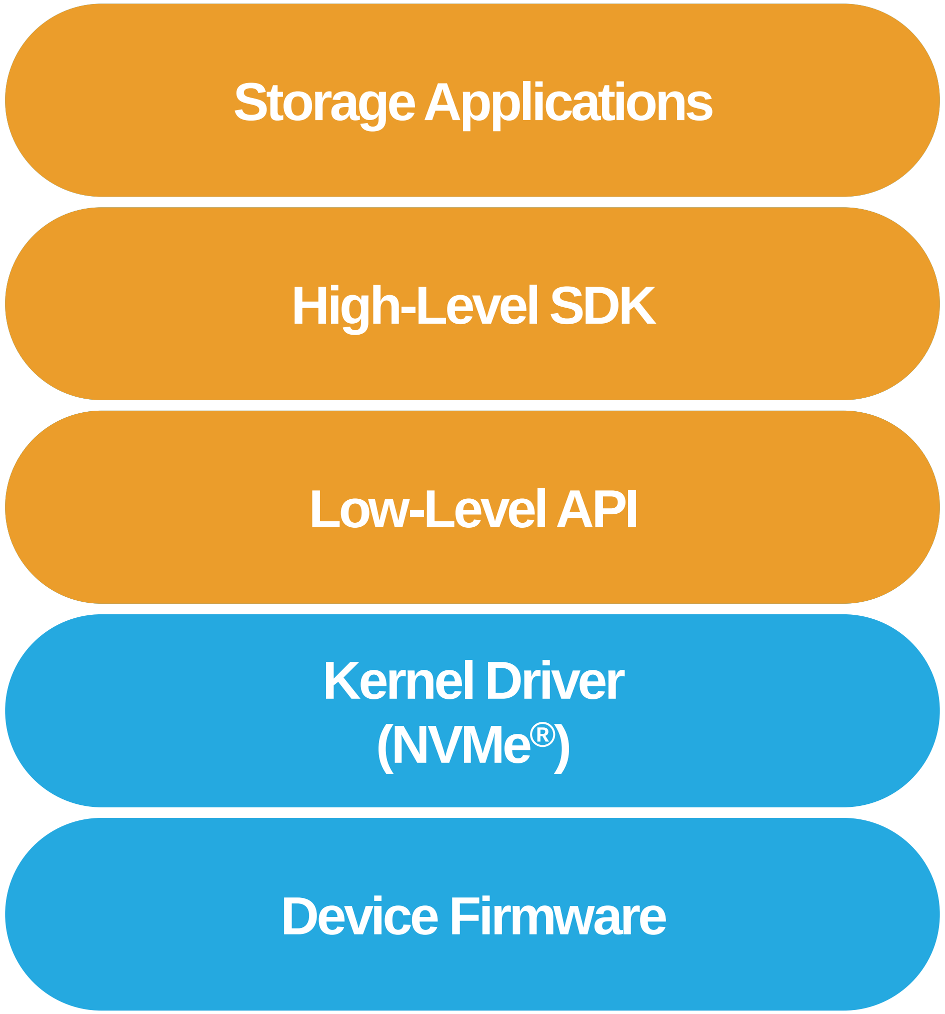


Hardware manages
the flash media

Host applications control
the storage behavior

* PCIe is a registered trademark and/or service mark of the PCI-SIG

Software Stack



Storage Applications

High-Level SDK

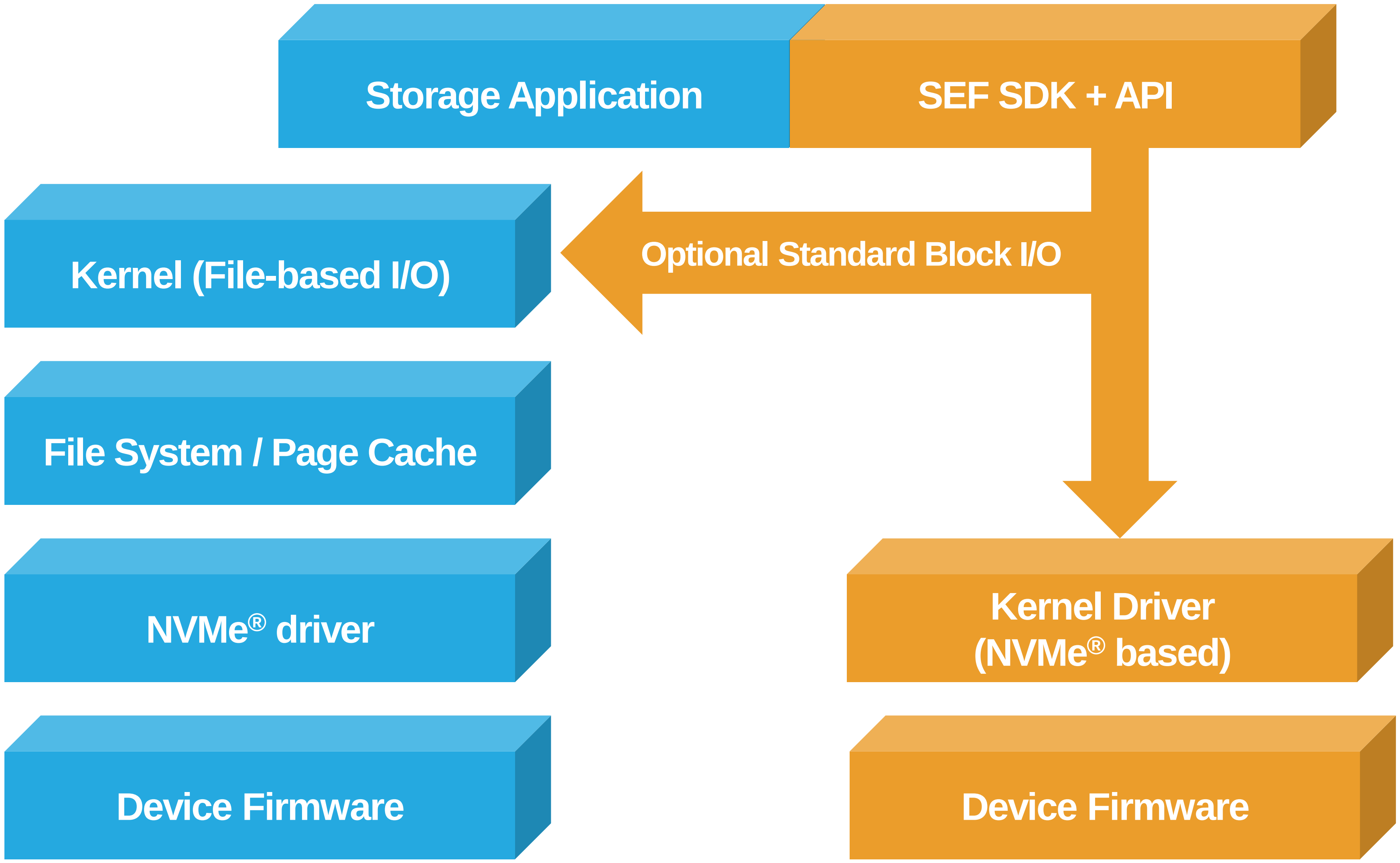
Low-Level API

Kernel Driver (NVMe®)

Device Firmware

* NVMe is a registered trademark and/or service mark of the PCI-SIG

Standard Block vs. SEF Application



Storage Application

SEF SDK + API

Optional Standard Block I/O

Kernel (File-based I/O)

File System / Page Cache

NVMe® driver

Device Firmware

Kernel Driver (NVMe® based)

Device Firmware

* NVMe is a registered trademark and/or service mark of the PCI-SIG

Application Programming Interface

- Low-level wrappers for device commands
- Exposes native "Nameless Write, Nameless Copy, Read Physical"
- Built to be multi-vendor capable

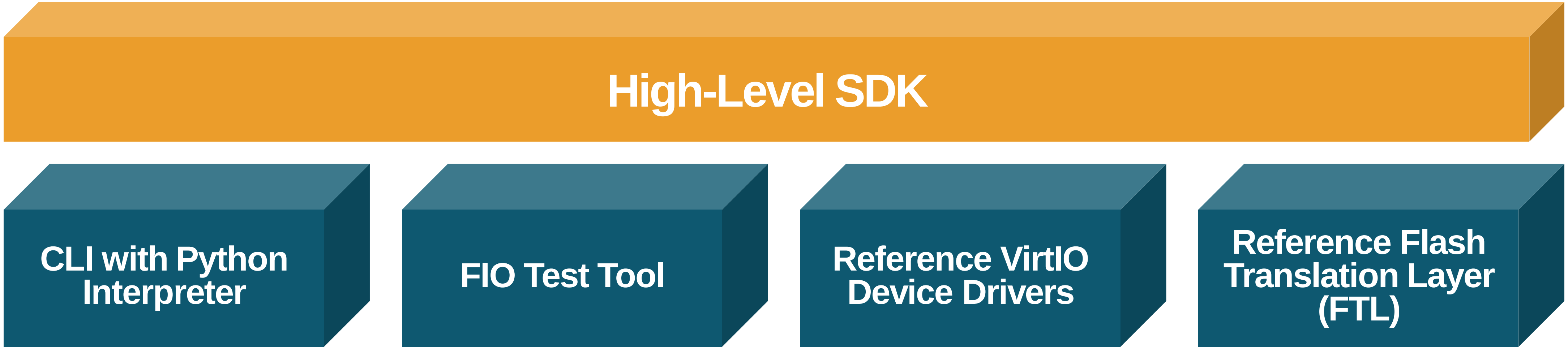
SEF
SOFTWARE-ENABLED FLASH™

Software
Development
Kit

- C language based
- 32 + 64 Bit
- Multiple architectures
- Modern Linux® kernels
- Library (shared or static)
- Event driven callbacks
- Thread safe, built for lockless operation
- Modular, built for customization

Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Software Development Kit



High-Level SDK

CLI with Python Interpreter

FIO Test Tool

Reference VirtIO Device Drivers

Reference Flash Translation Layer (FTL)

Device orchestration and management

Ported to SEF for fast and easy experimentation

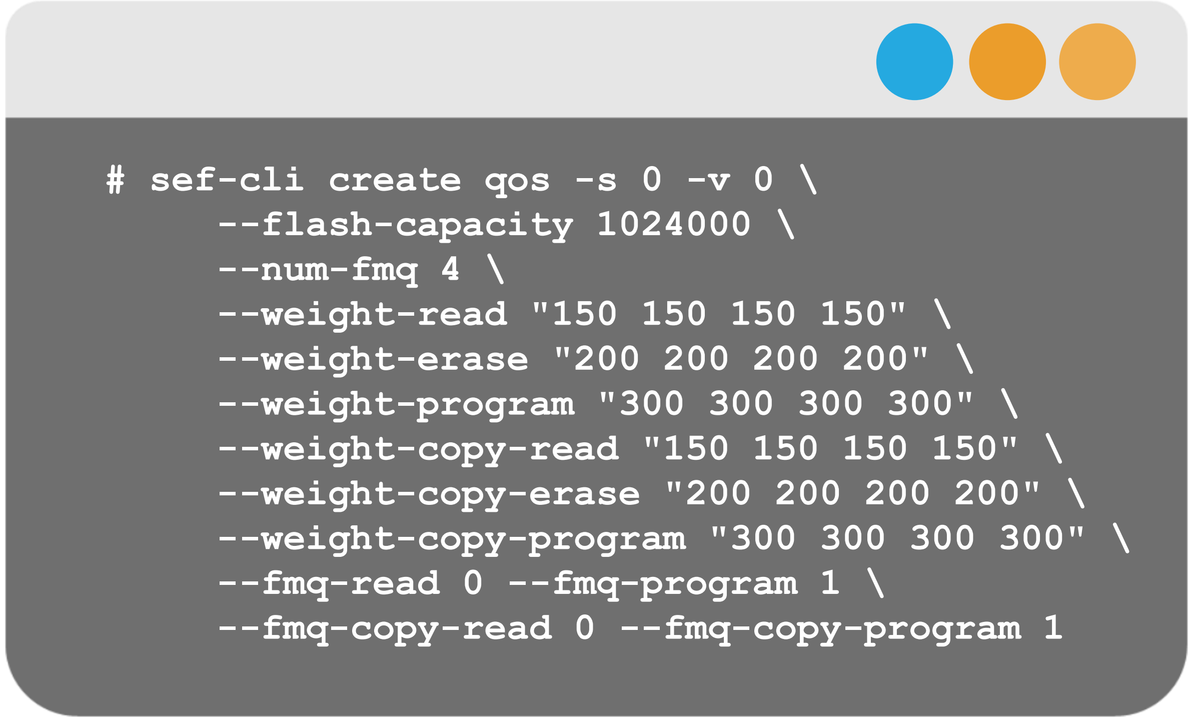
No code changes to evaluate SEF in multi-tenant mode

Bring more common block interface to SEF applications

Built to be customized

Command Line Interface

- Full lifecycle management
- Python® scriptable
- Dynamic provisioning
  - Per-application
  - Per-virtual machine
  - Per-container basis



```
# sef-cli create qos -s 0 -v 0 \
     --flash-capacity 1024000 \
     --num-fmq 4 \
     --weight-read "150 150 150 150" \
     --weight-erase "200 200 200 200" \
     --weight-program "300 300 300 300" \
     --weight-copy-read "150 150 150 150" \
     --weight-copy-erase "200 200 200 200" \
     --weight-copy-program "300 300 300 300" \
     --fmq-read 0 --fmq-program 1 \
     --fmq-copy-read 0 --fmq-copy-program 1
```

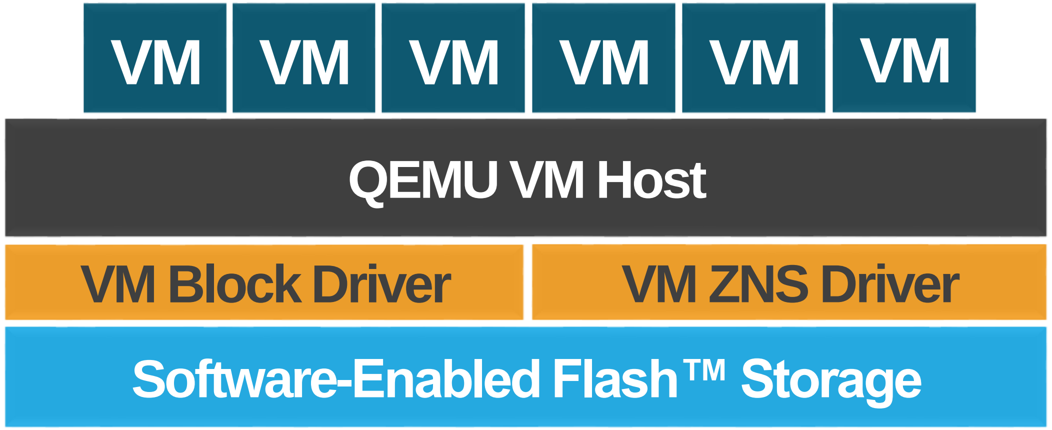
* Python is a registered trademark of the PSF.

FIO Testing Tool

- Explore configuration options
- Test latency and isolation controls
- Prototype system performance
- Full sources included in SDK

Reference VirtIO Device Drivers

- NO GUEST CODE CHANGES
- Customize overprovisioning per VM
- Run ZNS and block-based VMs on single drive
- Full data, performance isolation, queueing control



VM VM VM VM VM VM

QEMU VM Host

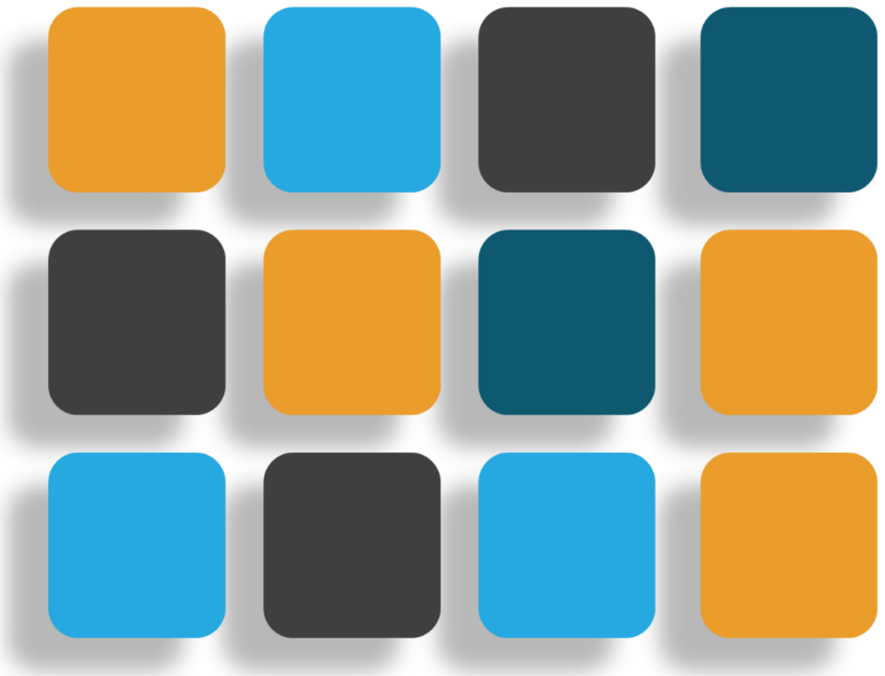
VM Block Driver

VM ZNS Driver

Software-Enabled Flash™ Storage

Reference FTL

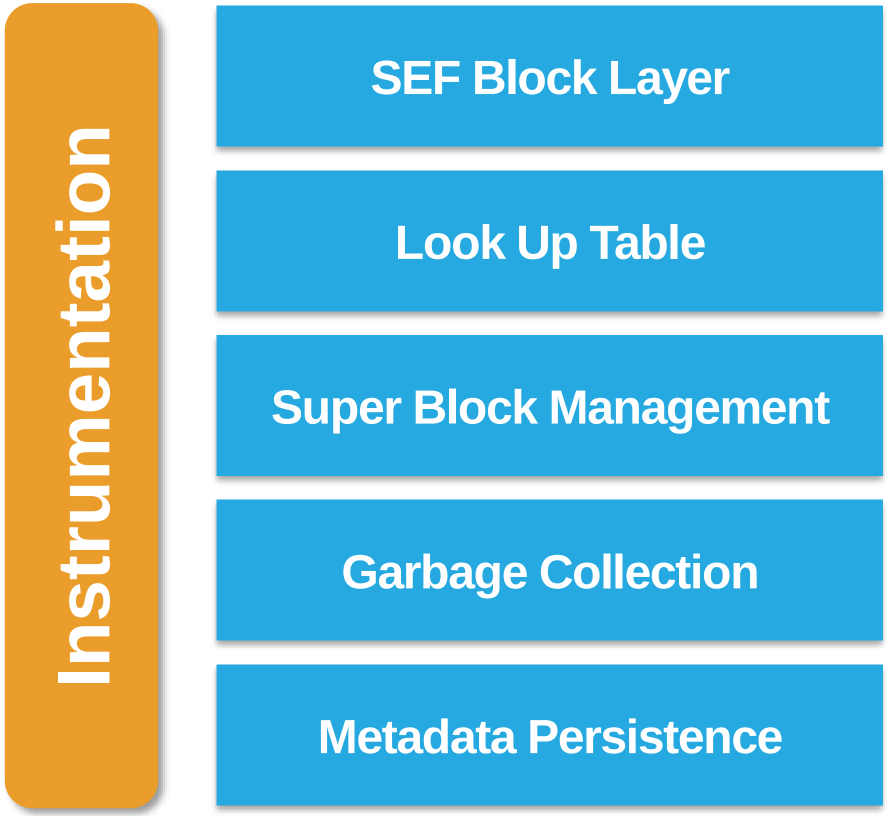
- Full Flash Translation Layer (FTL)
- Provides block-like interface to applications
- Built for modularity, expandability



FTL Tasks

- Handle user I/O requests
- Manage per-placement ID write buffers
- Protect against WAR, RAW, etc. hazards
- Map from logical to physical via look up table
- Tracks used super blocks and their states
- Recover from power loss
- Tombstoning old blocks
- Managing garbage collection, patrol reads, etc.

FTL Components



Instrumentation

SEF Block Layer

Look Up Table

Super Block Management

Garbage Collection

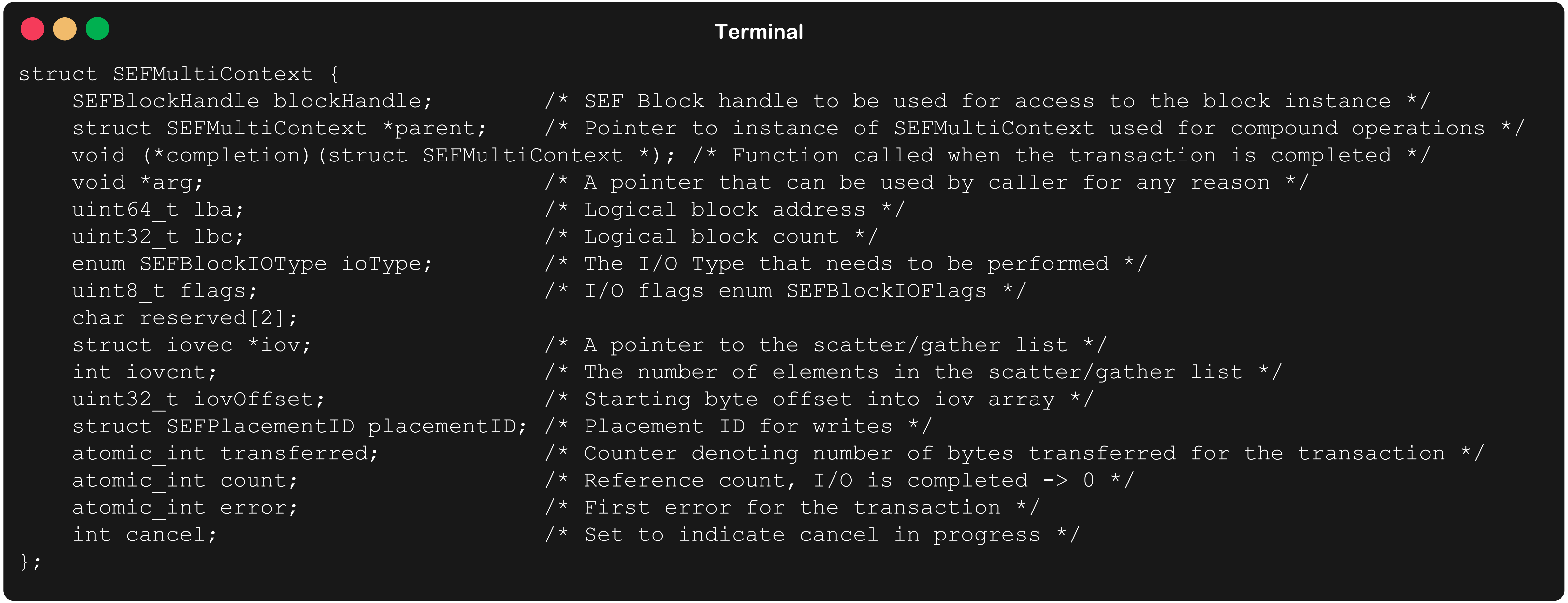
Metadata Persistence

Let’s walk through an I/O to examine each of these layers…

Application Sending I/O to Block Layer

SEF Block Layer

SefBlockIO(SefMultiContext *ctx)



Terminal

```
struct SEFMultiContext {
    SEFBlockHandle blockHandle;        /* SEF Block handle to be used for access to the block instance */
    struct SEFMultiContext *parent;    /* Pointer to instance of SEFMultiContext used for compound operations */
    void (*completion)(struct SEFMultiContext *); /* Function called when the transaction is completed */
    void *arg;                         /* A pointer that can be used by caller for any reason */
    uint64_t lba;                      /* Logical block address */
    uint32_t lbc;                      /* Logical block count */
    enum SEFBlockIOType ioType;        /* The I/O Type that needs to be performed */
    uint8_t flags;                     /* I/O flags enum SEFBlockIOFlags */
    char reserved[2];
    struct iovec *iov;                 /* A pointer to the scatter/gather list */
    int iovcnt;                        /* The number of elements in the scatter/gather list */
    uint32_t iovOffset;                /* Starting byte offset into iov array */
    struct SEFPlacementID placementID; /* Placement ID for writes */
    atomic_int transferred;            /* Counter denoting number of bytes transferred for the transaction */
    atomic_int count;                  /* Reference count, I/O is completed -> 0 */
    atomic_int error;                  /* First error for the transaction */
    int cancel;                        /* Set to indicate cancel in progress */
};
```

Look Up Table (LUT)

Look Up Table

- Contains mapping of LBA to a physical flash address

- 64-Bits per entry for support of Massive Capacities
  - 2GiB RAM per 1 TiB flash
  - Host-based DRAM use

- Different use cases could optimize
  - Object storage
  - Zoned Namespace-like accesses
  - Compression (start, extent, etc.)
  - Split between host RAM and drive flash

* GiB refers to gibibyte, or 2^30. TiB refers to tibibyte, or 2^40

Super Block Management

- Device responsible for choosing “best” super block to allocate
- Super Block module keeps track of allocated blocks
  - Identifiers (opaque, give by device)
  - Current state (open for write, open for copy, closed, etc.)
  - Placement ID associated
  - Number of allocated ADU (~sector)
  - Bitmap of valid ADUs
  - Etc.
- Provides information to garbage collection as needed
- Minimal RAM requirements

Garbage Collection Module

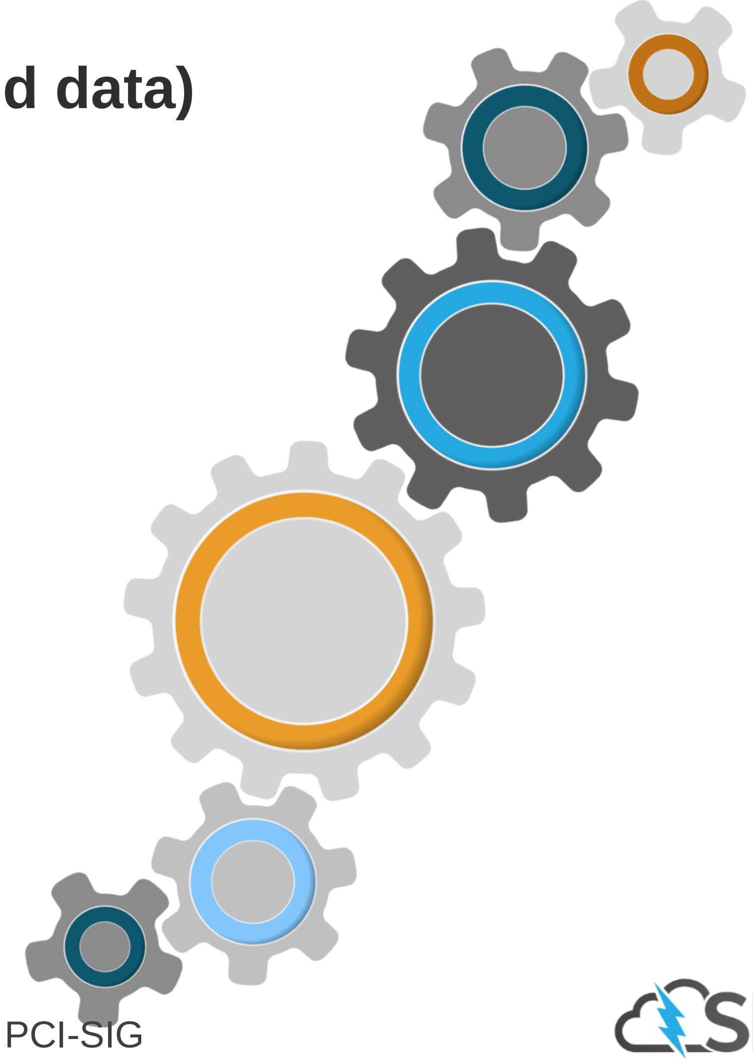
Garbage Collection

- Automatic and application initiated
  - Free super blocks drop below defined threshold
  - Application decides “now is a good time”
- Runs in its own thread
- Supports full SEF offload and queueing
  - Copy offload (nameless copy) fully implemented
  - Can be assigned to any specific queue to run at higher or lower priority on the device
- Can be customized or replaced by developer

Garbage Collection Procedure

Garbage Collection

- While (still work to do)
  - Get list of collectable superblocks (ones w/invalid data)
  - Sort by # of invalid ADUs(~sectors)
  - Determine placement id with most invalid data
  - Allocate destination super blocks
  - Send nameless copy bitmaps (from Super BlockTracking)
  - Perform copy in-drive, no host CPU or DRAM or PCIe® bus bandwidth
  - Update Flash Translation with new mappings
  - Discard read-out super blocks



* PCIe is a registered trademark and/or service mark of the PCI-SIG

(Metadata) Persistence

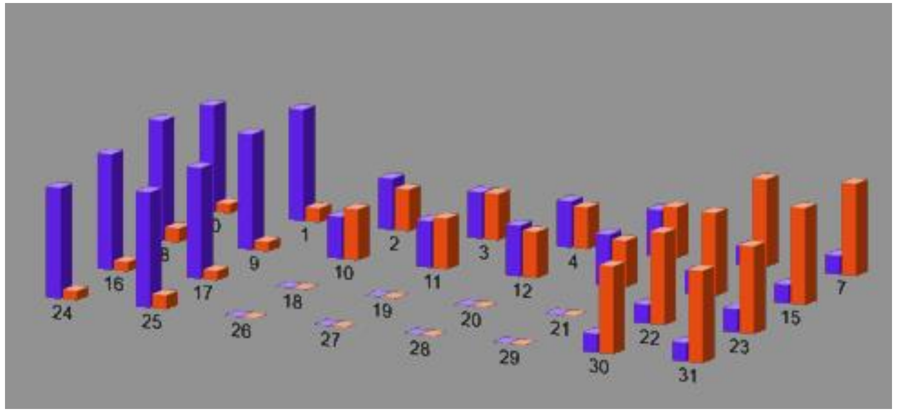
Metadata Persistence

- Keeps track of metadata
  - FTL look up tables
  - Super Block state
  - Placement IDs
  - Etc.
- Uses “Root Pointer” feature of SEF hardware
  - Provides a well-known area for data storage
- Enables restart of FTL after unclean shut down

Instrumentation

- SEF operations invisible to standard I/O tracking tools
  - IOstat, etc. will not register any I/O

- Dynamic enable and disable
  - Sample counters without restarting application
  - Avoid overhead of tracking if not needed



- Controlled via named UNIX sockets
  - Can dump JSON format for easy use

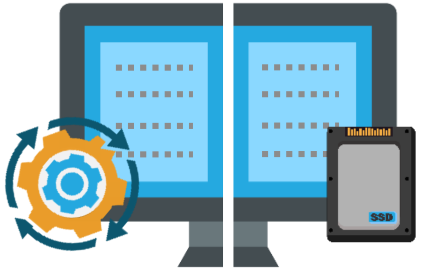
Future SEF SDK Ideas

EXT4-on-SEF file system

- Directly links EXT4 inodes into SEF
- Applications could use standard file system interface, get SEF benefits

SEF on Data Processing Unit (DPU) or Computational Storage

- ARM® processor support already enabled
- Minimize host resource impact on virtualized systems



Distributing write buffers between host & drive RAM

- SEF hardware specification allows for flexibility in design

* Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

Summary

SEF provides fine-grained control over flash to applications.

SEF SDK makes it easier to use.

SEF Reference FTL is modular and extendable.

Get Involved

- **Get source code at GitHub**
  - https://github.com/SoftwareEnabledFlash/
- **Read and watch more content**
  - https://softwareenabledflash.org
- **Join the mailing list**
  - https://lists.softwareenabledflash.org/g/sef-dev/join
- **Sign up for the Software-Enabled Flash Project**
  - https://enrollment.lfx.linuxfoundation.org/?project=sef

THE LINUX FOUNDATION