NVMe & MI

Eng.#3 SW Eng., James Hsiao

Revision 1.0

23 - June - 2020

Agenda

- **NVM Express™**
  - Introduction
  - Theory & Architecture
  - Data Structures
  - Features
  - Directives
  - Changes in NVMe 1.4
- **NVM Express™ Management Interface**
  - Introduction
  - Theory & Architecture
  - Message Transport
  - Message Servicing Model
  - Command Set
  - Examples
- **Denali Open-Channel SSDs**
  - Introduction
  - Physical Page Address Command Set

NVM Express™

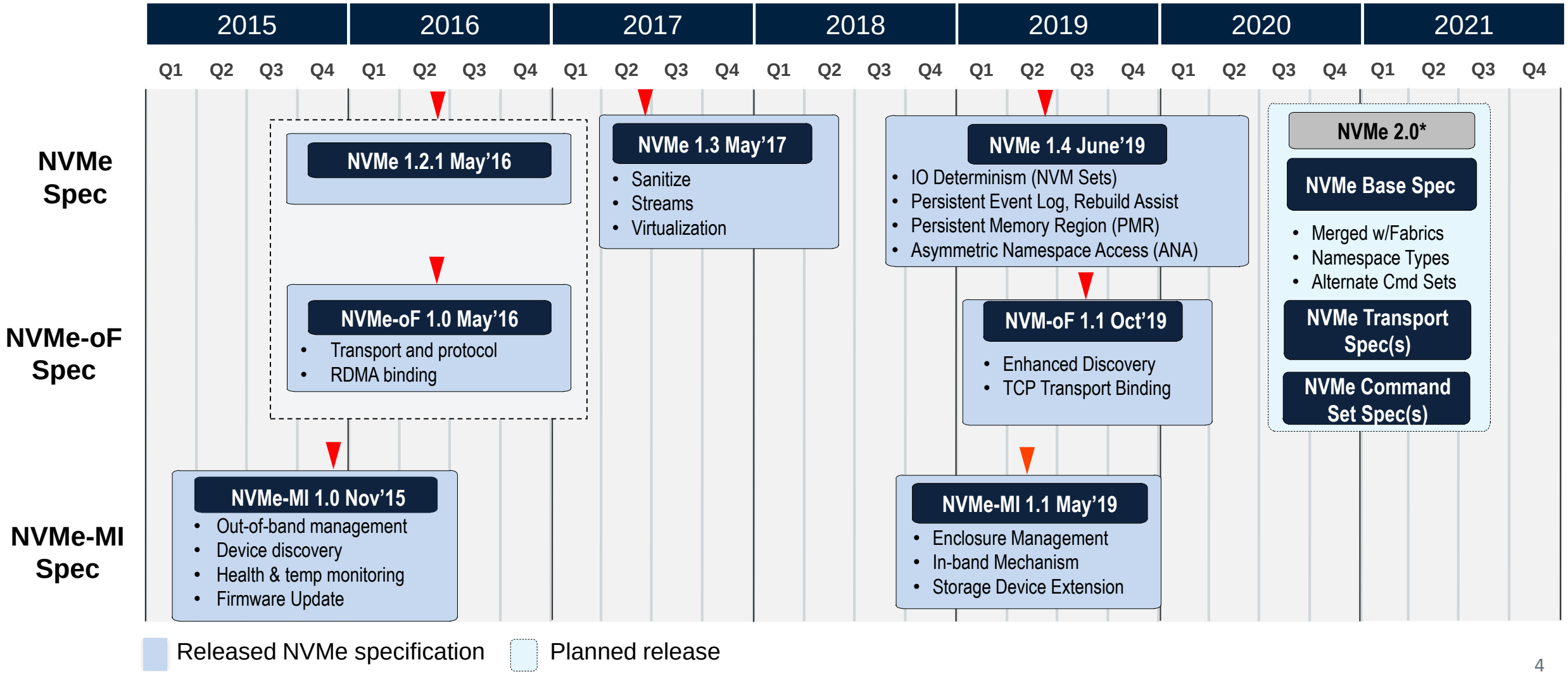
Introduction

What is NVMe™?

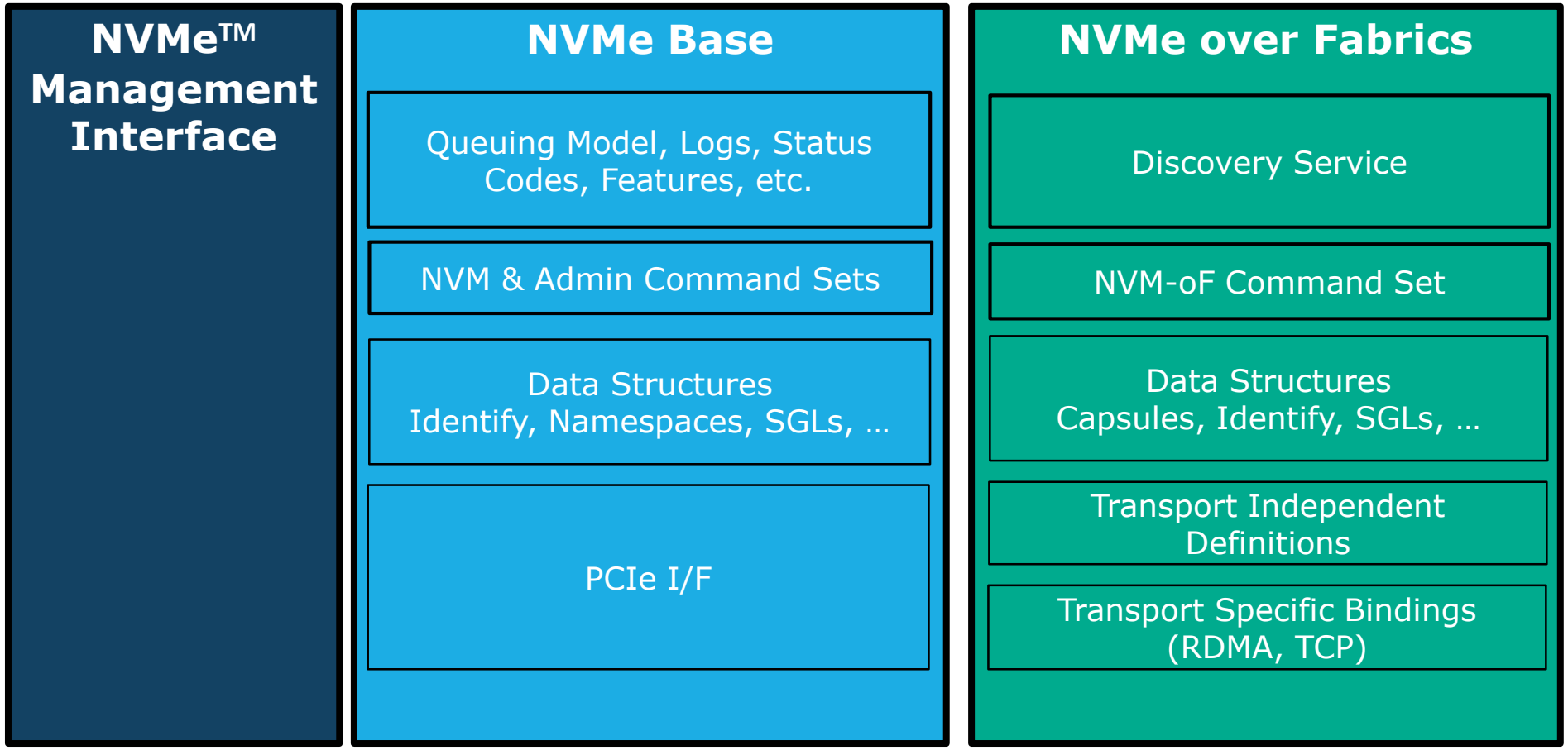
- NVM Express™ (NVMe™) is a specification defining how host software communicates with non-volatile memory across a PCI Express® (PCIe®) bus. It is the industry standard for PCIe solid state drives (SSDs) in all form factors (U.2, M.2, AIC, EDSFF). NVM Express is the non-profit consortium of tech industry leaders defining, managing and marketing NVMe technology. In addition to the NVMe base specification, the organization hosts other specifications: NVMe over Fabrics (NVMe-oF™) for using NVMe commands over a networked fabric and NVMe Management Interface (NVMe-MI™) to manage NVMe/PCIe SSDs in servers and storage systems.

- The NVMe specification was designed from the ground up for SSDs. It is a much more efficient interface, providing lower latency, and is more scalable for SSDs than legacy interfaces, like serial ATA (SATA). The first part of the specification is the host control interface. The NVMe architecture brings a new high performance queuing mechanism that supports 65,535 I/O queues each with 65,535 commands (referred to as queue depth, or number of outstanding commands). Queues are mapped to CPU cores delivering scalable performance. The NVMe interface significantly reduces the number of memory-mapped input/output commands and accommodates operating system device drivers running in interrupt or polling modes for higher performance and lower latency. The NVMe specification also contains host-to-device protocol for SSD commands used by an operating system for: read, write, flush, TRIM, firmware management, temperature, errors and others.

NVM Express™ Technology Specification Roadmap



NVMe™ spec family wasn't structured for extensibility

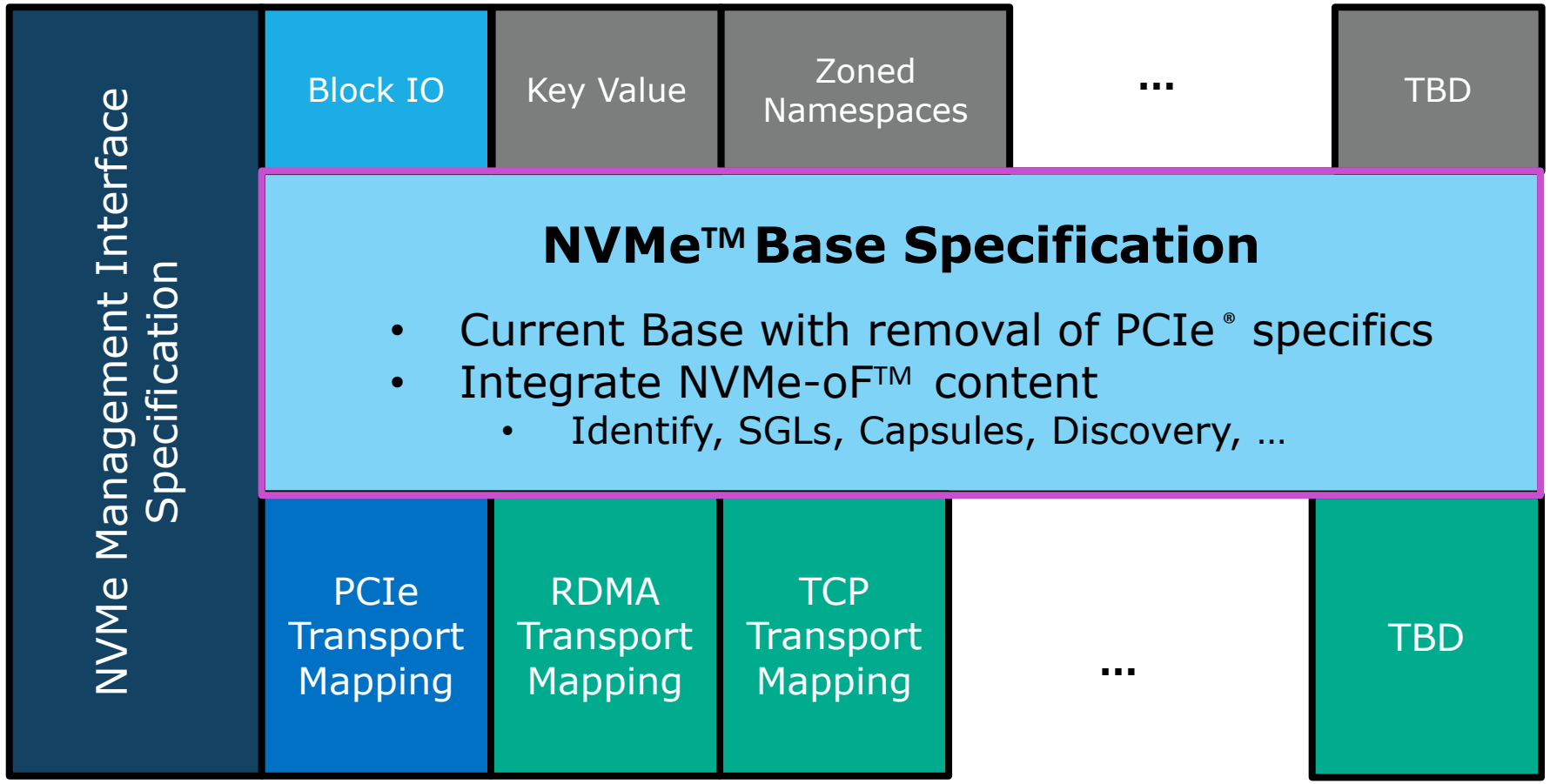


- Similar functions split between Base & Fabrics
- PCIe® transport integrated into Base
- Command Sets not layered to enable extensibility

Need a new structure to enable innovation!

Optimizing the Specifications for Evolution

Transport Separation, Command Set Extensibility, Fabrics Base Integration



- **Adds Fabrics concepts as core to NVMe**
- **Eliminates duplication in data structures**
- **Integration of NVMe and NVMe-oF base functions**
- **Separate command set specs**
- **Modular transport mapping layer, including PCIe**

Why NVMe?

NVM Express™(NVMe™) Advantages over SATA™

PCIe® for **scalable** performance, **flexible** form factors, and industry **stability**

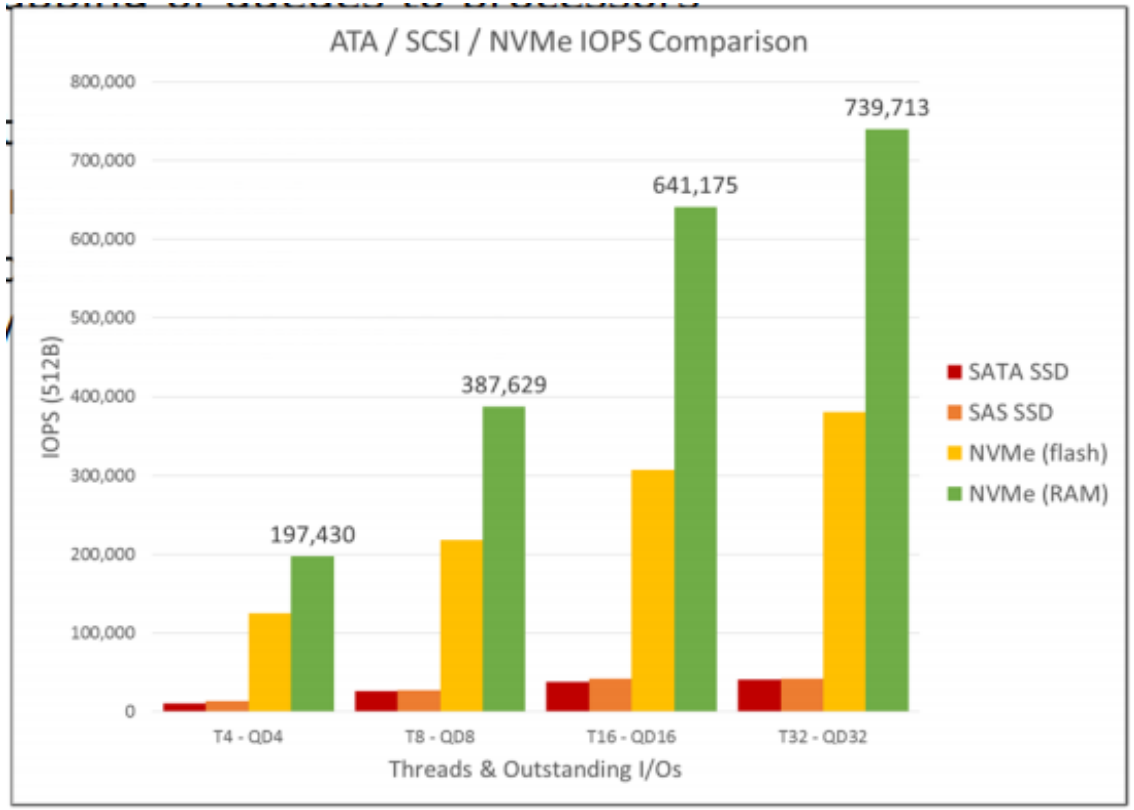
NVMe provides lower **latency** and increased **efficiency**: lower CPU utilization, lower power, lower TCO

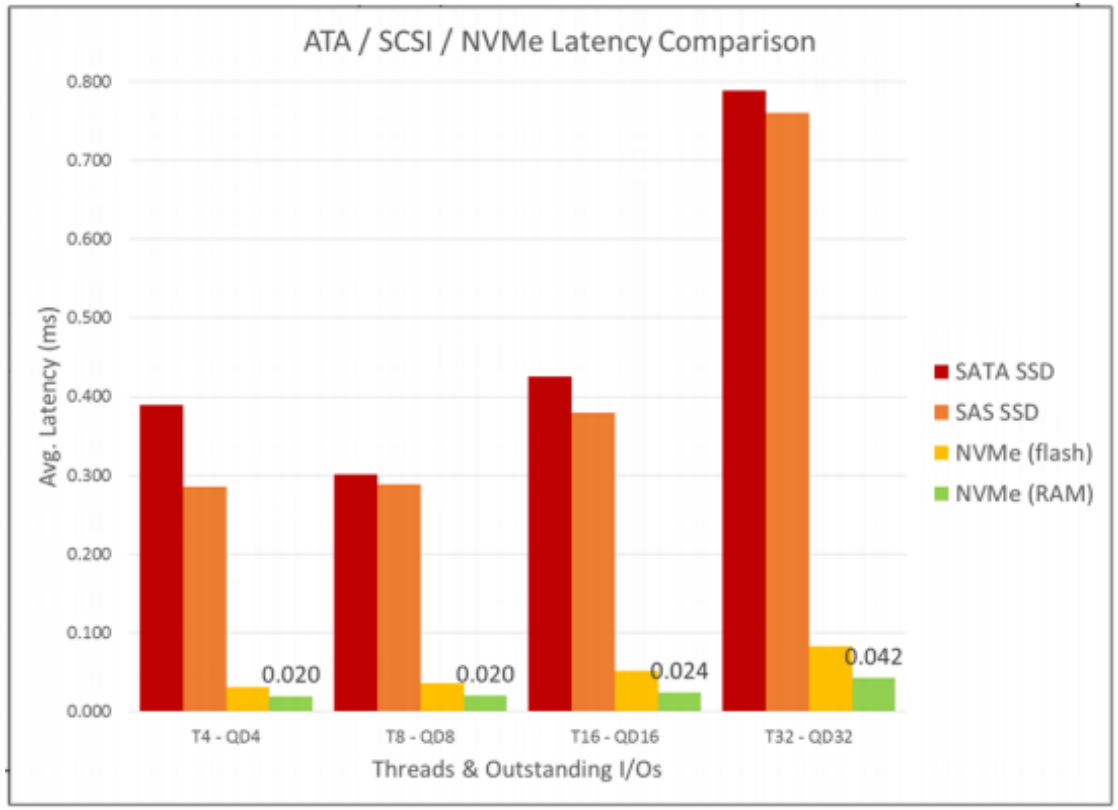
Increased **bandwidth**: 1 GB/s per lane – 1-16 lanes per drive
Directly attached to CPU, eliminate HBA cost and overhead

Low power features from both PCIe and NVMe
Security from Trusted Computing Group OPAL

Why NVMe?

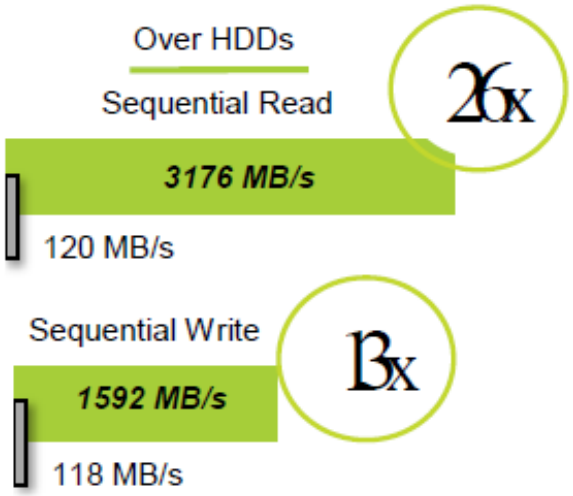
- With great IOPs and low latency



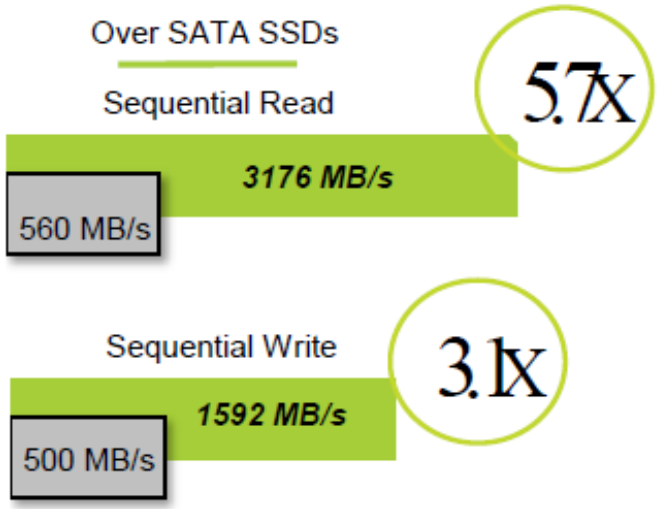


Why NVMe?

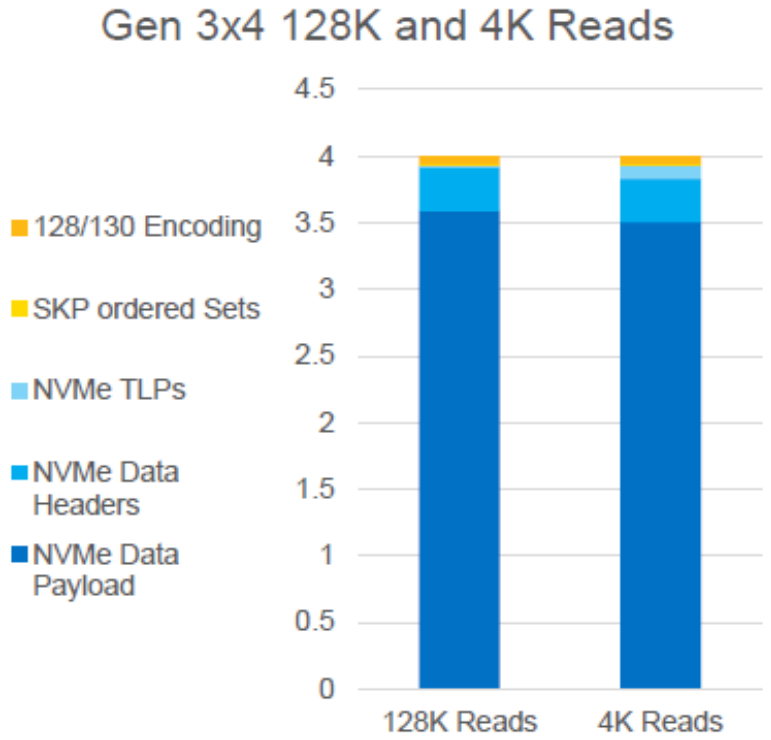
- **Removes the performance bottleneck**



Intel®SSD 7 Series
versus
WD Blue* 5400RPM 500 GB HDD



Intel®SSD 7 Series
versus
Intel®545 SATA-based SSD



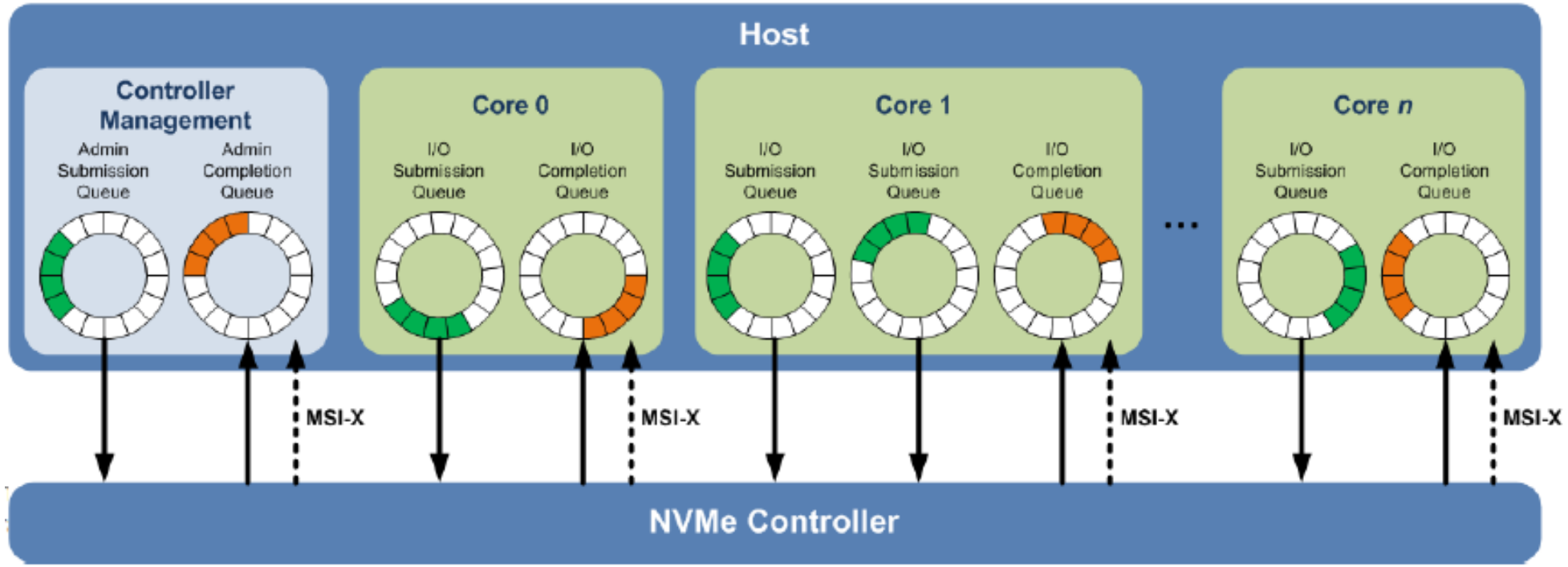
NVM Express™

Theory & Architecture

Theory of Operation

Technical Basics

- Support for up to 65,535 I/O Queues, with each I/O Queue supporting up to 65,535 outstanding commands;
- All information to complete a 4 KiB read request is included in the 64B command itself, ensuring efficient small I/O operation;
- Efficient and streamlined command set (10 required Admin Commands, 3 required I/O Commands);
- Support for MSI/MSI-X and interrupt aggregation;
- Support for multiple namespaces, multi-path I/O and namespace sharing, and efficient support for I/O virtualization architectures like SR-IOV.

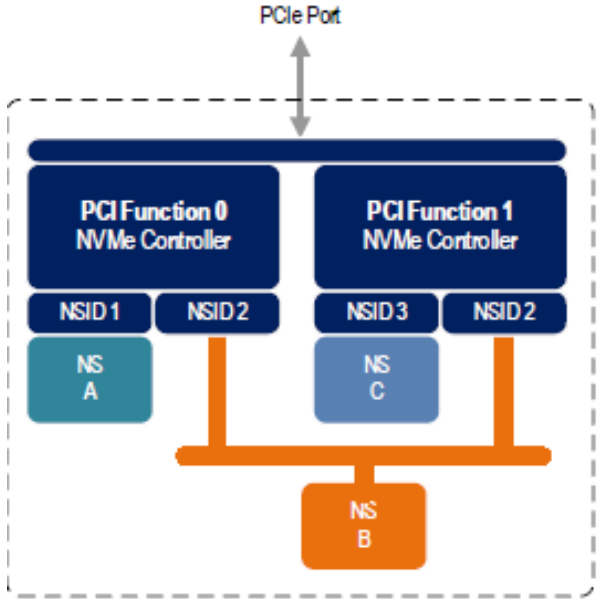


Theory of Operation

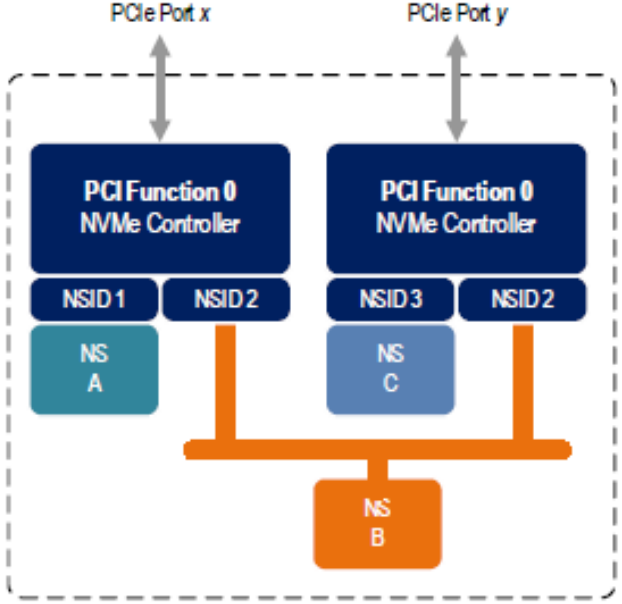
Multi-Path I/O and Namespace Sharing

- An NVM subsystem includes one or more controllers, zero or more namespaces, and one or more ports. An NVM subsystem may include a non-volatile memory storage medium and an interface between the controller(s) in the NVM subsystem and non-volatile memory storage medium.
- Multi-path I/O refers to two or more completely independent paths between a single host and a namespace while namespace sharing refers to the ability for two or more hosts to access a common shared namespace using different NVM Express controllers.
- There is a unique Identify Controller data structure for each controller and a unique Identify Namespace data structure for each namespace. The same data structure contents are returned by all controllers with access to the same shared namespace. Controllers associated with a shared namespace may operate on the namespace concurrently.

NVM Subsystem with Two Controllers and One Port



NVM Subsystem with Two Controllers and Two Ports



Controller Architecture

Controller Types

- The Controller Type (CNTRLTYPE) field in the Identify Controller data structure indicates a controller's type. Regardless of controller type, all controllers implement one Admin Submission Queue and one Admin Completion Queue. Depending on the controller type, a controller may also support one or more I/O Submission Queues and I/O Completion Queues.
- I/O Controller: An I/O controller is a general purpose controller that supports commands that provide access to an NVM subsystem's non-volatile storage medium and may support commands that provide management capabilities.
- Administrative Controller: An administrative controller is a controller whose intended purpose is to provide NVM subsystem management capabilities. Unlike an I/O controller, an administrative controller does not support commands that provide access to logical block data and metadata stored on an NVM subsystem's non-volatile storage medium.

Figure 418: NVM Subsystem with Three I/O Controllers

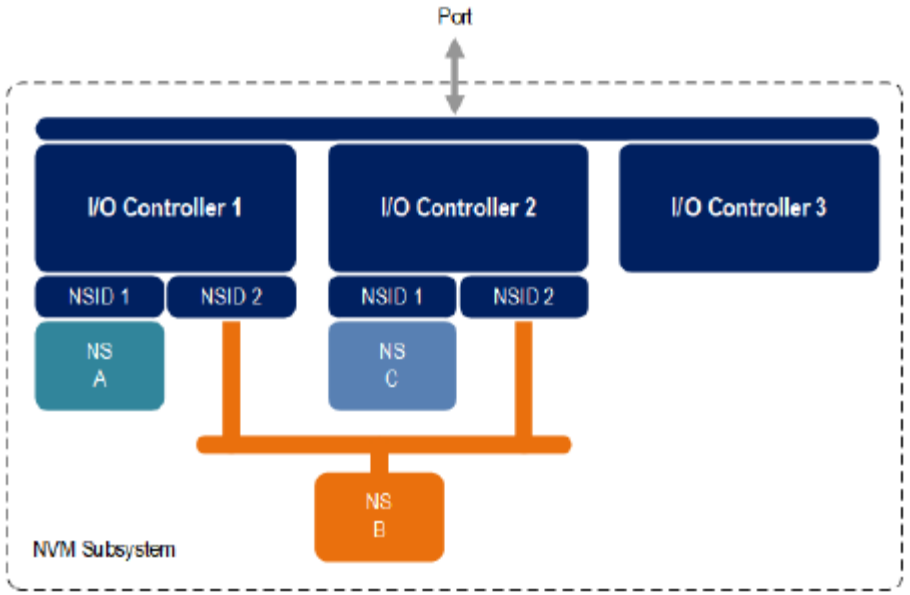
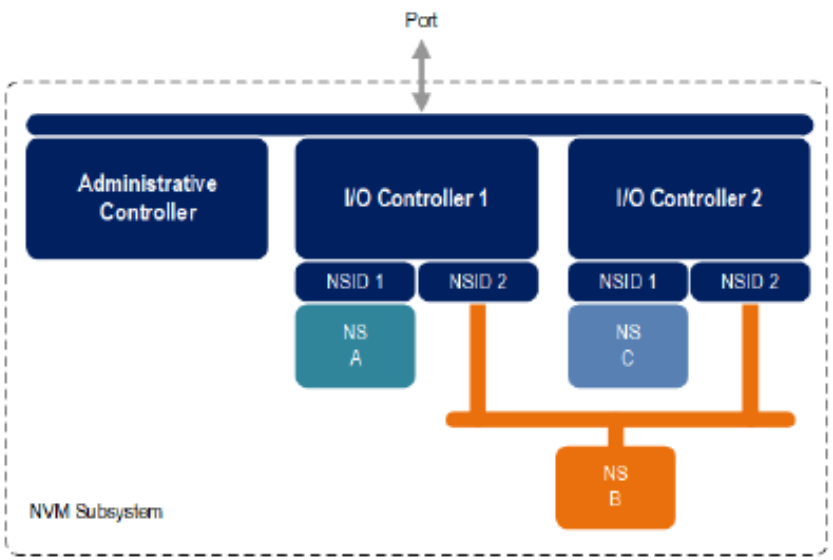
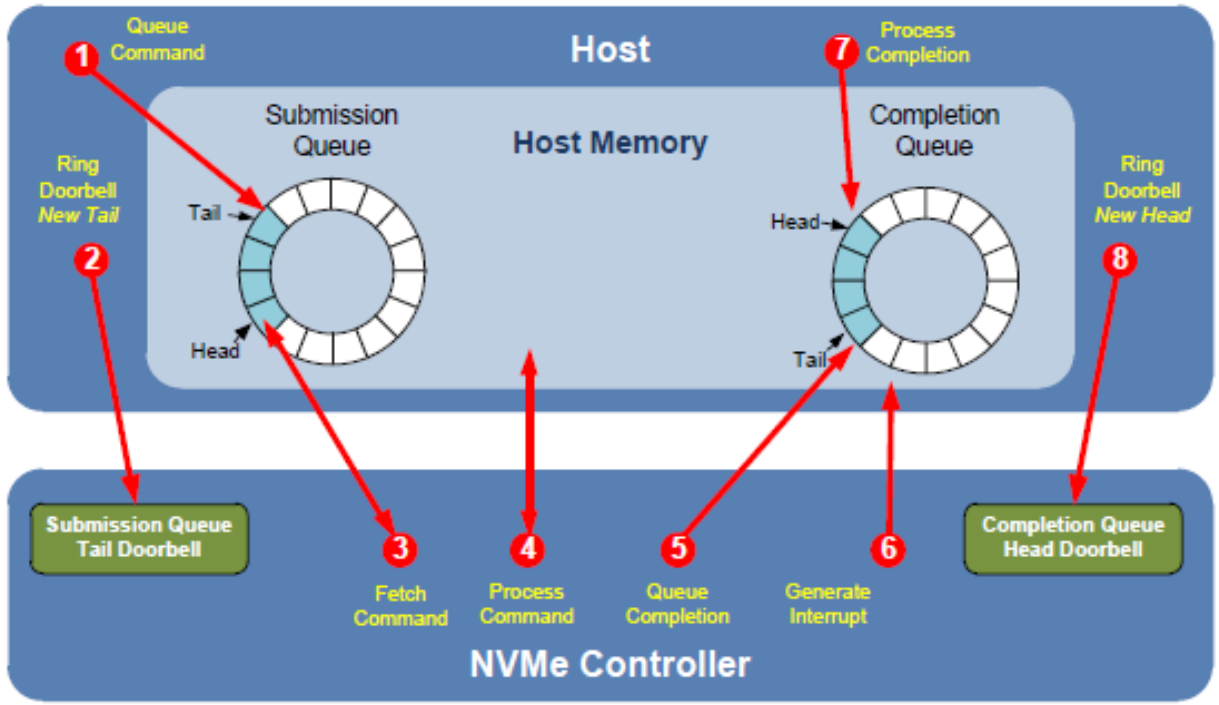


Figure 426: NVM Subsystem with One Administrative and Two I/O Controllers



Controller Architecture

- Command Submission and Completion Mechanism



Command Submission

1. Host writes command to Submission Queue
2. Host writes updated Submission Queue tail pointer to doorbell

Command Processing

3. Controller fetches command
4. Controller processes command

Command Completion

5. Controller writes completion to Completion Queue
6. Controller generates MSI-X interrupt
7. Host processes completion
8. Host writes updated Completion Queue head pointer to doorbell

Controller Architecture

Resets

NVM Subsystem Reset

- Main power is applied to the NVM subsystem;
- A value of 4E564D65h (“NVMe”) is written to the NSSR.NSSRC field;

Controller Level Reset

- NVM Subsystem Reset;
- Conventional Reset;
- Function Level Reset; and
- Controller Reset (i.e., CC.EN transitions from ‘1’ to ‘0’).

A Controller Level Reset consists of the following actions:

- The controller stops processing any outstanding Admin or I/O commands;
- All I/O Submission Queues are deleted;
- All I/O Completion Queues are deleted;
- The controller is brought to an Idle state. When this is complete, CSTS.RDY is cleared to ‘0’; and
- All controller registers and internal controller state are reset.

To continue after a Controller Level Reset, the host shall:

- Update register state as appropriate;
- Set CC.EN to ‘1’;
- Wait for CSTS.RDY to be set to ‘1’;
- Configure the controller using Admin commands as needed;
- Create I/O Completion Queues and I/O Submission Queues as needed; and
- Proceed with normal I/O operations.

Controller Architecture

Queue Management

Queue Setup and Initialization

1. Configures the Admin Submission and Completion Queues by initializing the Admin Queue Attributes (AQA), Admin Submission Queue Base Address (ASQ), and Admin Completion Queue Base Address (ACQ) registers appropriately;
2. Configure the size of the I/O Submission Queues (CC.IOSQES) and I/O Completion Queues (CC.IOCQES);
3. Submits a Set Features command with the Number of Queues attribute set to the requested number of I/O Submission Queues and I/O Completion Queues. The completion queue entry for this Set Features command indicates the number of I/O Submission Queues and I/O Completion Queues allocated by the controller;
4. Determines the maximum number of entries supported per queue (CAP.MQES) and whether the queues are required to be physically contiguous (CAP.CQR);
5. Creates I/O Completion Queues within the limitations of the number allocated by the controller and the queue attributes supported (maximum entries and physically contiguous requirements) by using the Create I/O Completion Queue command; and
6. Creates I/O Submission Queues within the limitations of the number allocated by the controller and the queue attributes supported (maximum entries and physically contiguous requirements) by using the Create I/O Submission Queue command.

Controller Architecture

Shutdown

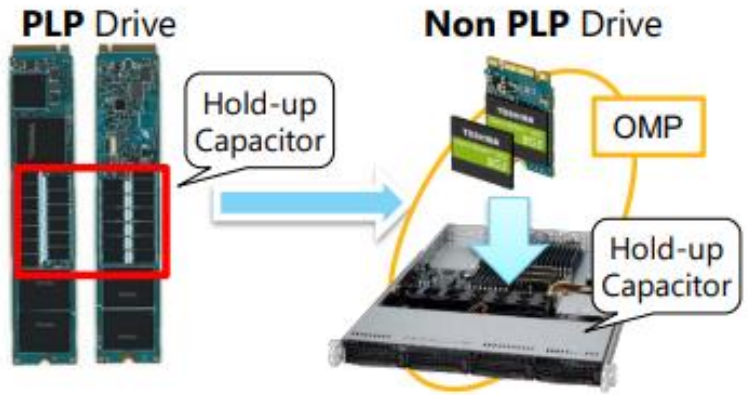
- The host should perform the following actions in sequence for a normal shutdown:

1. Stop submitting any new I/O commands to the controller and allow any outstanding commands to complete;
2. If the controller implements I/O queues, then the host should delete all I/O Submission Queues, using the Delete I/O Submission Queue command. A result of the successful completion of the Delete I/O Submission Queue command is that any remaining commands outstanding are aborted;
3. If the controller implements I/O queues, then the host should delete all I/O Completion Queues, using the Delete I/O Completion Queue command; and
4. The host should set the Shutdown Notification (CC.SHN) field to 01b to indicate a normal shutdown operation. The controller indicates when shutdown processing is completed by updating the Shutdown Status (CSTS.SHST) field to 10b.

- The host should perform the following actions in sequence for an abrupt shutdown:

1. Stop submitting any new I/O commands to the controller; and
2. The host should set the Shutdown Notification (CC.SHN) field to 10b to indicate an abrupt shutdown operation. The controller indicates when shutdown processing is completed by updating the Shutdown Status (CSTS.SHST) field to 10b.

- Power loss protection (PLP) is a mechanism to save DRAM-cached data and gracefully shut down an SSD upon an unexpected power loss condition
- PLP typically uses capacitors on the SSD to provide hold-up power until data is flushed from the DRAM to the NAND flash
- Off Module Power utilizes the host system or module to provide hold up power, eliminating the need for capacitors on the SSDs; helps reduce cost



NVM Express™

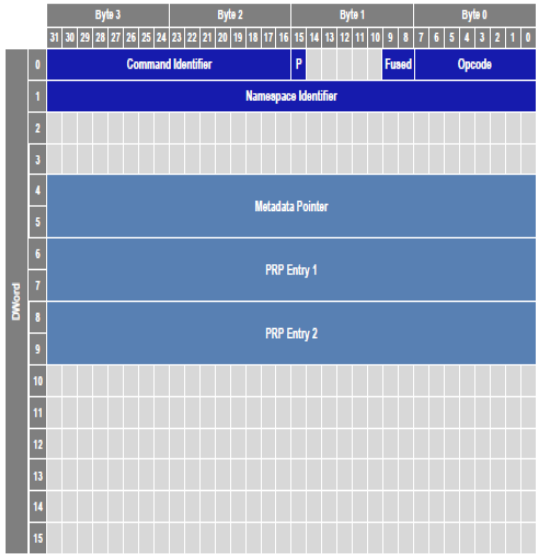
Data Structures

Data Structures

■ Submission Queue Entry

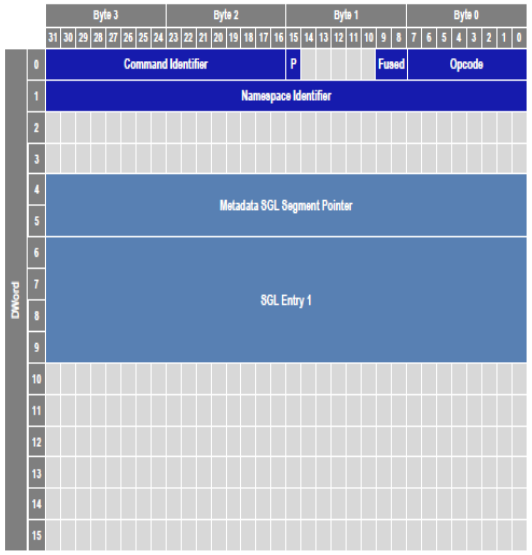
Each command is 64 bytes in size.
PRPs shall be used for all Admin commands for NVMe over PCIe implementations.
SGLs shall be used for all Admin and I/O commands for NVMe over Fabrics implementations.

Flash Memory SUMMIT Command Format (PRP)



- **Opcode** - Command to execute
- **Fused** – Indicates two commands should be executed as atomic unit
- **P** - Use PRPs or SGLs for data transfer
- **Command Identifier** - Unique ID associated with command
- **Namespace Identifier** - Namespace on which command operates
- **Metadata Pointer** – Pointer to contiguous buffer containing metadata in "DIX" mode
- **PRP Entry 1 & 2** – PRP or PRP list

Flash Memory SUMMIT Command Format (SGL)

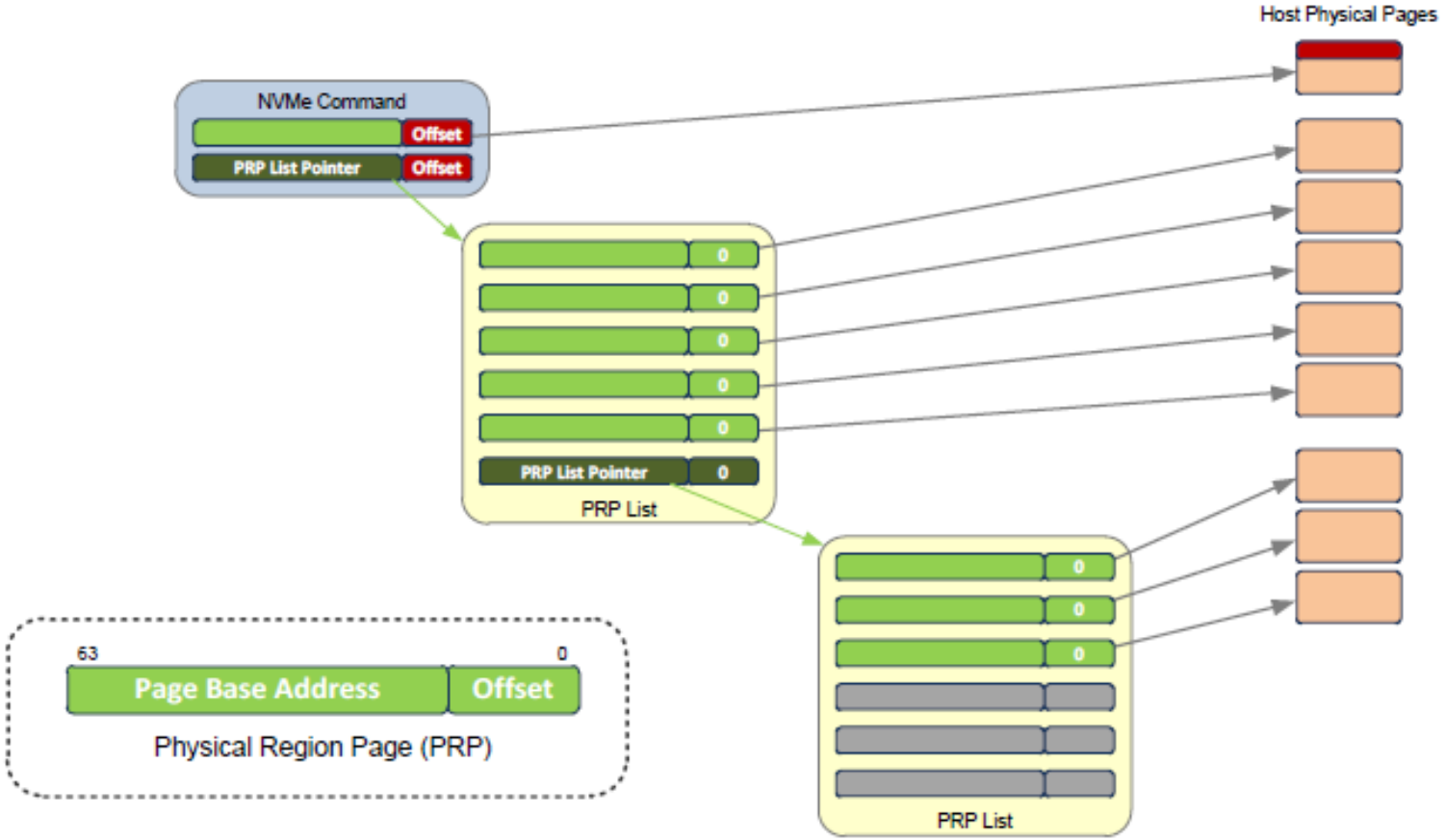


- **Opcode** - Command to execute
- **Fused** – Indicates two simpler commands should be executed as atomic unit
- **P** - Use PRPs or SGLs for data transfer
- **Command Identifier** - Unique ID associated with command
- **Namespace Identifier** - Namespace on which command operates
- **Metadata SGL Segment Pointer** – Pointer to metadata SGL segment in "DIX" mode
- **SGL Entry 1** – First SGL segment associated with data transfer

Data Structures

Physical Region Page Entry and List

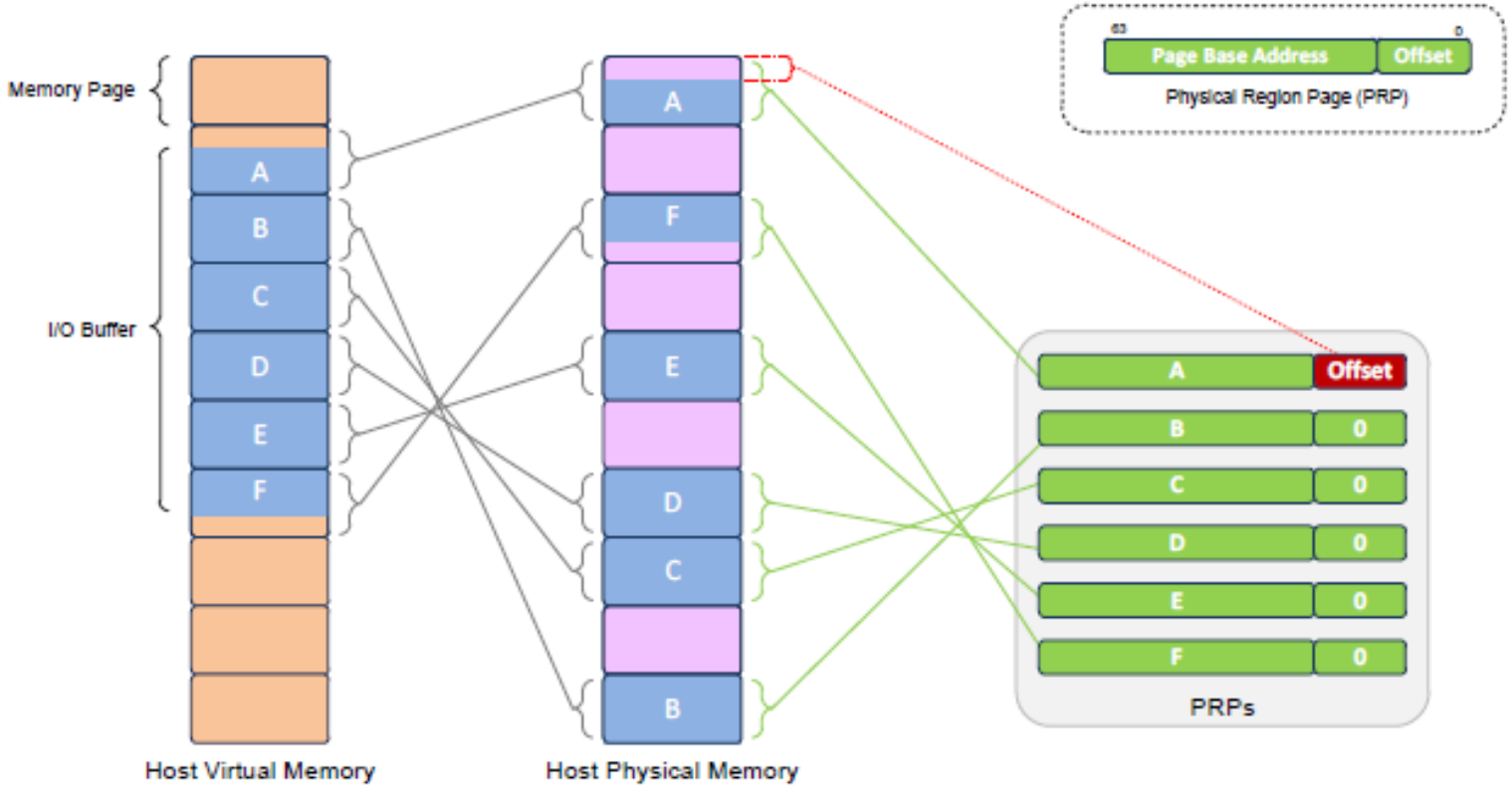
A physical region page (PRP) entry is a pointer to a physical memory page. PRPs are used as a scatter/gather mechanism for data transfers between the controller and memory. A physical region page list (PRP List) is a set of PRP entries in a single page of contiguous memory.



Data Structures

Physical Region Page Entry and List

To enable efficient out of order data transfers between the controller and the host, PRP entries are a fixed size.

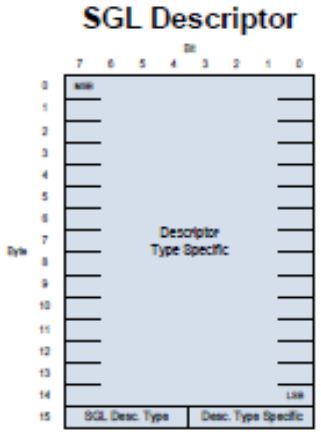


Fixed Size PRPs Accelerate Out of Order Data Delivery

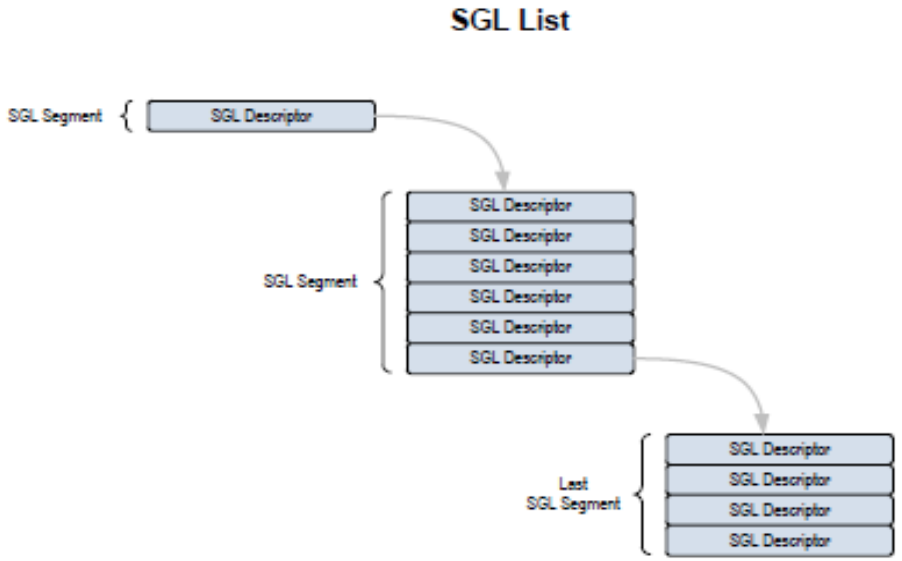
Data Structures

Scatter Gather List (SGL)

A Scatter Gather List (SGL) is a data structure in memory address space used to describe a data buffer. An SGL contains one or more SGL segments. An SGL segment is a qword aligned data structure in a contiguous region of physical memory describing all, part of, or none of a data buffer and the next SGL segment, if any. An SGL segment consists of an array of one or more SGL descriptors. Only the last descriptor in an SGL segment may be an SGL Segment descriptor or an SGL Last Segment descriptor.



Code	SGL Descriptor Type
0h	SGL Data Block
1h	SGL Bit Bucket
2h	SGL Segment
3h	SGL Last Segment
4h - Eh	Reserved
Fh	Vendor Specific



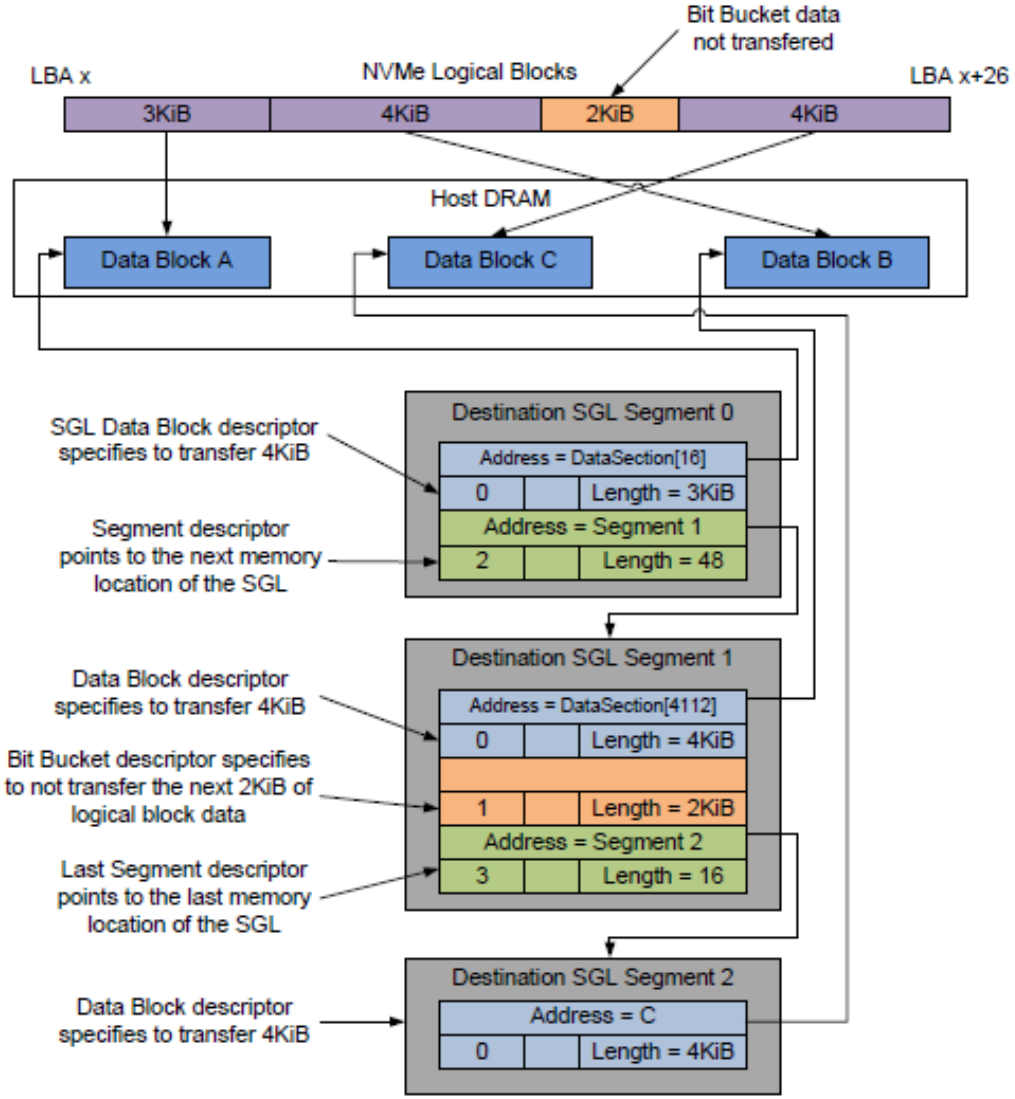
SGLs Enable Arbitrary Data Transfer Size and Byte Alignment

Data Structures

Scatter Gather List (SGL)

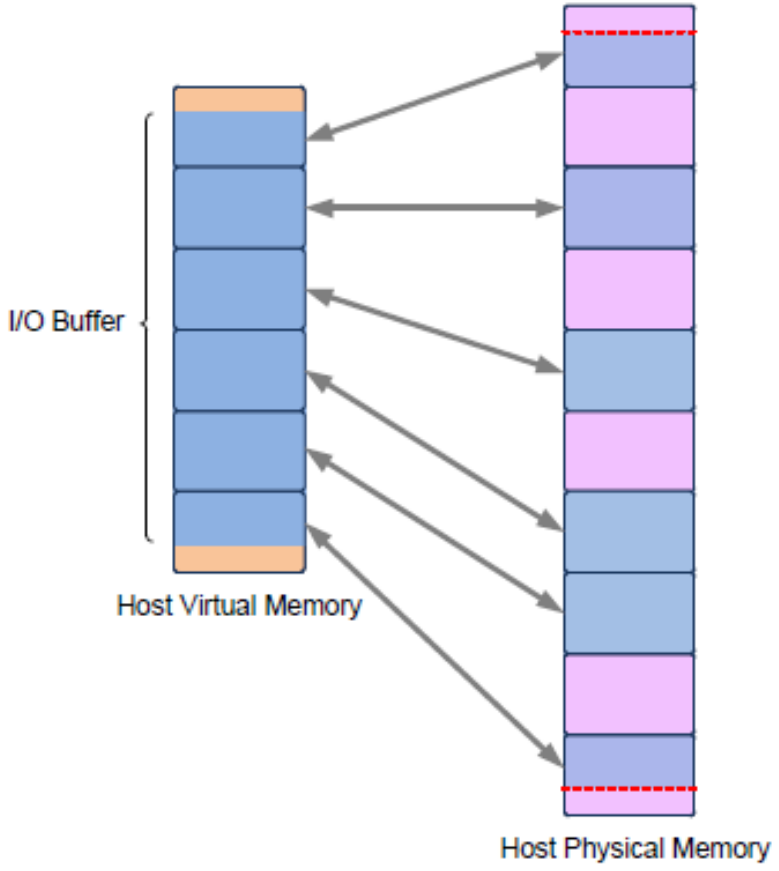
In the example, the logical block size is 512B. The total length of the logical blocks accessed is 13 KiB, of which only 11 KiB is transferred to the host. The Number of Logical Blocks (NLB) field in the command shall specify 26, indicating the total length of the logical blocks accessed on the controller is 13 KiB. There are three SGL segments describing the locations in memory where the logical block data is transferred.

The three SGL segments contain a total of three Data Block descriptors with lengths of 3 KiB, 4 KiB, and 4 KiB respectively. Segment 1 of the Destination SGL contains a Bit Bucket descriptor with a length of 2 KiB that specifies to not transfer (i.e., ignore) 2 KiB of logical block data from the NVM. Segment 1 of the destination SGL also contains a Last Segment descriptor specifying that the segment pointed to by the descriptor is the last SGL segment.

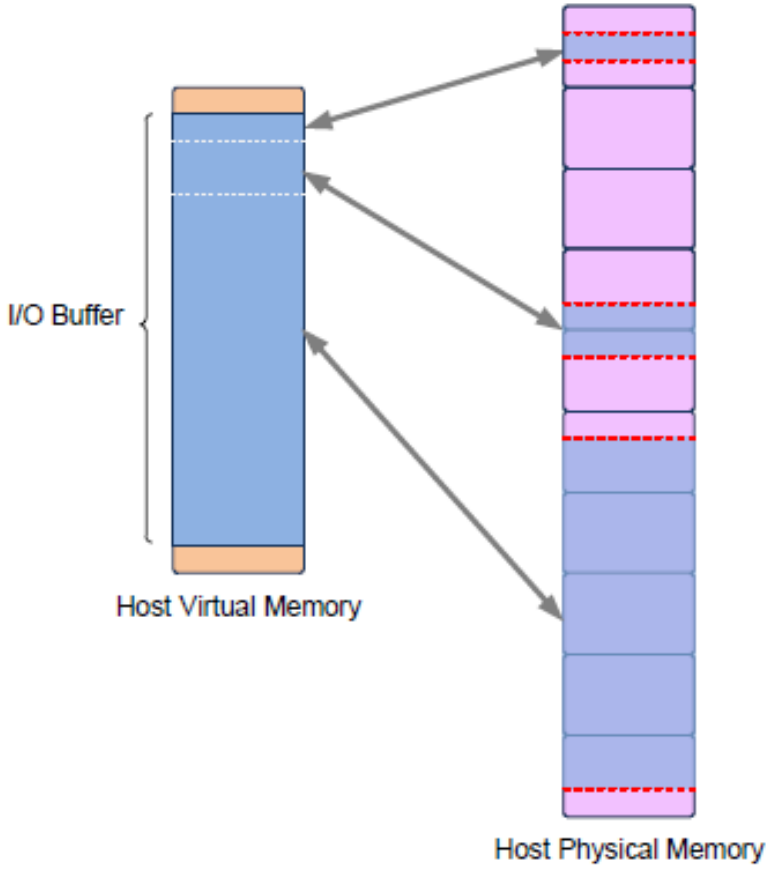


Data Structures

- Comparing SGLs with PRPs



PRP Data Transfer



SGL Data Transfer

Data Structures

■ Completion Queue Entry

An entry in the Completion Queue is at least 16 bytes in size.

□ Status Field

Indicates status for the command that is being completed. A value of 0h for the Status Field indicates a successful command completion, with no fatal or non-fatal error conditions.

- Status Code Type (SCT)
- 0h: Generic Command Status
- 1h: Command Specific Status
- 2h: Media and Data Integrity Errors
- 3h: Path Related Status
- Status Code (SC)
- 00h to 7Fh: Applicable to Admin Command Set, or across multiple command sets;
- 80h to BFh: I/O Command Set Specific status codes; and
- C0h to FFh: Vendor Specific status codes.

Figure 122: Completion Queue Entry Layout – Admin and NVM Command Set

	31	23	15	7	0
DW0	Command Specific				
DW1	Reserved				
DW2	SQ Identifier		SQ Head Pointer		
DW3	Status Field	P	Command Identifier		

Data Structures

Namespaces

A namespace is a collection of logical blocks whose logical block addresses range from 0 to the size of the namespace – 1. A namespace ID (NSID) is an identifier used by a controller to provide access to a namespace.

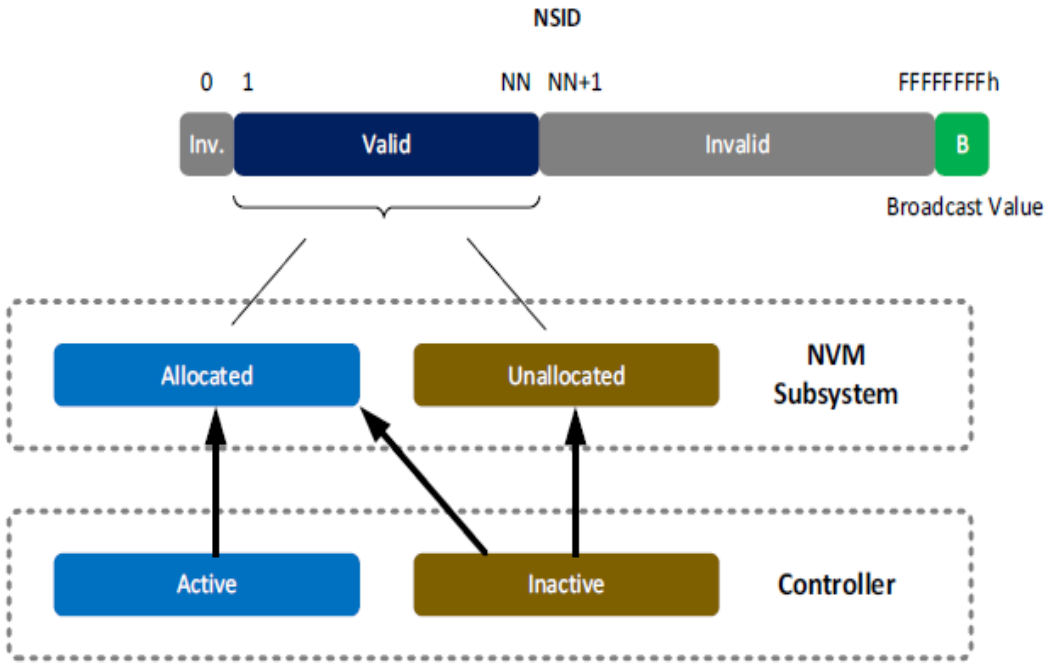
Any NSID is valid, except if that NSID is 0h or greater than the Number of Namespaces field reported in the Identify Controller data structure. NSID FFFFFFFFh is a broadcast value that is used to specify all namespaces.

Figure 349: NSID Types and Relationship to Namespace

Valid NSID Type	NSID relationship to namespace	Reference
Unallocated	Does not refer to any namespace that exists in the NVM subsystem	6.1.3
Allocated	Refers to a namespace that exists in the NVM subsystem	6.1.3
Inactive	Does not refer to a namespace that is attached to this controller[1]	6.1.4
Active	Refers to a namespace that is attached to this controller	6.1.4

NOTES:
1. If allocated, refers to a namespace that is not attached to this controller. If unallocated, does not refer to any namespace.

Figure 350: NSID Types



Data Structures

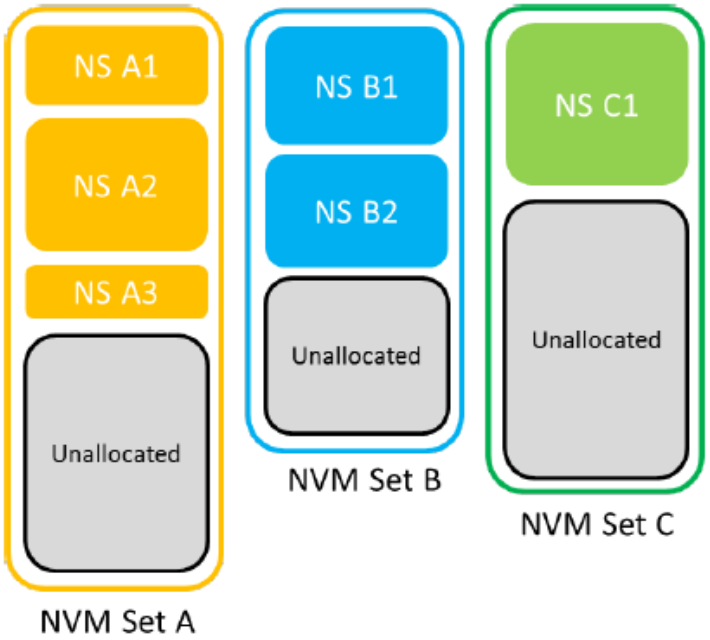
NVM Sets

An NVM Set is a collection of NVM that is separate (logically and potentially physically) from NVM in other NVM Sets. One or more namespaces may be created within an NVM Set and those namespaces inherit the attributes of the NVM Set. A namespace is wholly contained within a single NVM Set and shall not span more than one NVM Set.

For each NVM Set, the attributes include:

- an identifier associated with the NVM Set;
- the optimal size for writes to the NVM Set;
- the total capacity of the NVM Set; and
- the unallocated capacity for the NVM Set.

Figure 134: NVM Sets and Associated Namespaces



Data Structures

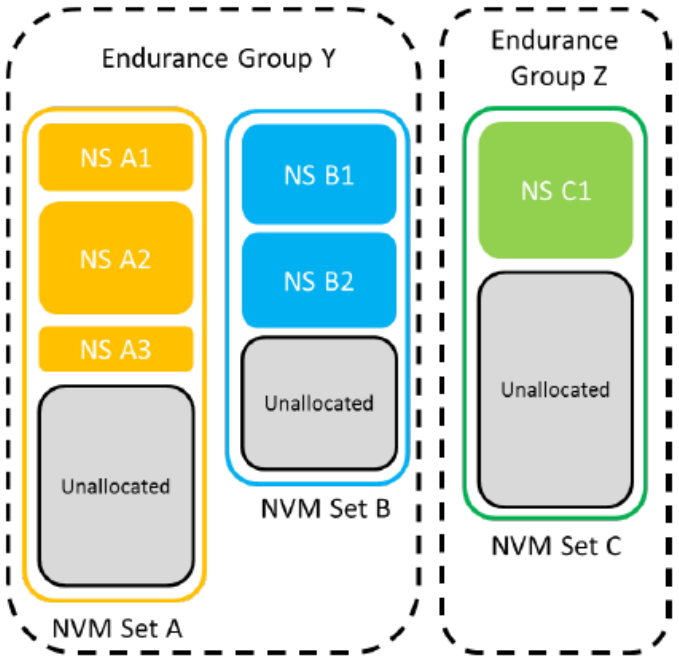
Endurance Groups

Endurance may be managed within a single NVM Set or across a collection of NVM Sets. Each NVM Set is associated with an Endurance Group. If two or more NVM Sets have the same Endurance Group Identifier, then endurance is managed by the NVM subsystem across that collection of NVM Sets.

If Endurance Groups are supported, then the NVM subsystem and all controllers shall:

- Indicate support for Endurance Groups in the Controller Attributes field in the Identify Controller data structure;
- Indicate the Endurance Group Identifier with which the namespace is associated in the Identify Namespace data structure; and
- Support the Endurance Group Information log page.

Figure 486: NVM Sets and Associated Namespaces



Data Structures

- **Persistent Memory Region (PMR)**
  - PCIe memory space on the SSD exposed to the Host
  - May be used to store command data
  - Contents persist across power cycles, resets and disabling of the PMR

- **Usage Models for PMR**
  - **Logs for SW RAID, EC & Databases**
  - **Journals for File Systems**
  - **Metadata**
  - **Staging area for data pre-processing**
  - **Network transactions**

NVM Express™

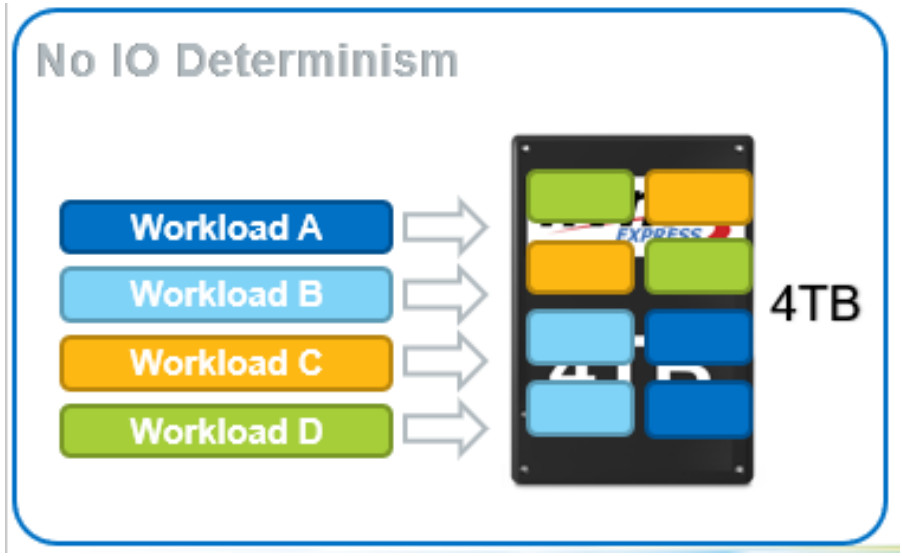
Features

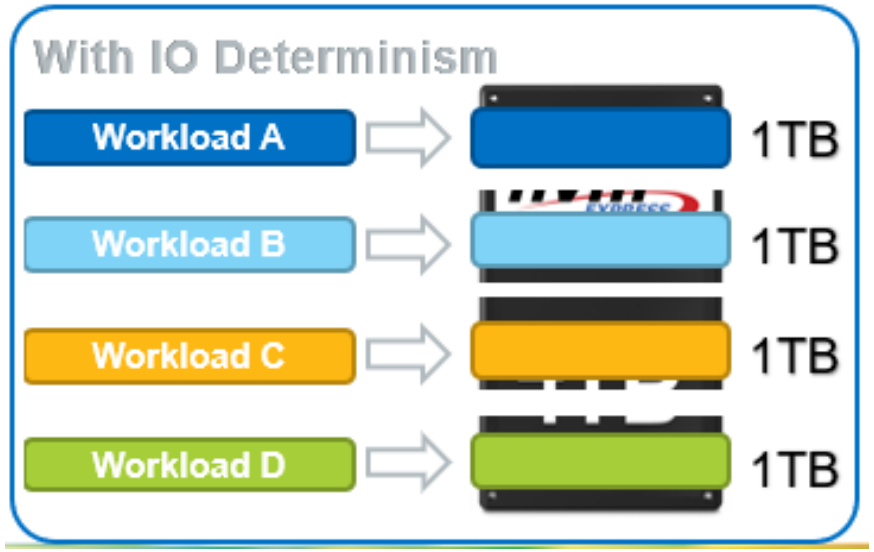
Features

IO Determinism – NVM Sets

- NVM Sets are QoS Isolated
  - Write to namespace A1 does not impact QoS associated with namespace B2
- NVM Subsystem may support one or more NVM Sets
- One or more Namespaces may be allocated to an NVM Set

IO Determinism & NVM Sets work together to provide improved QoS!





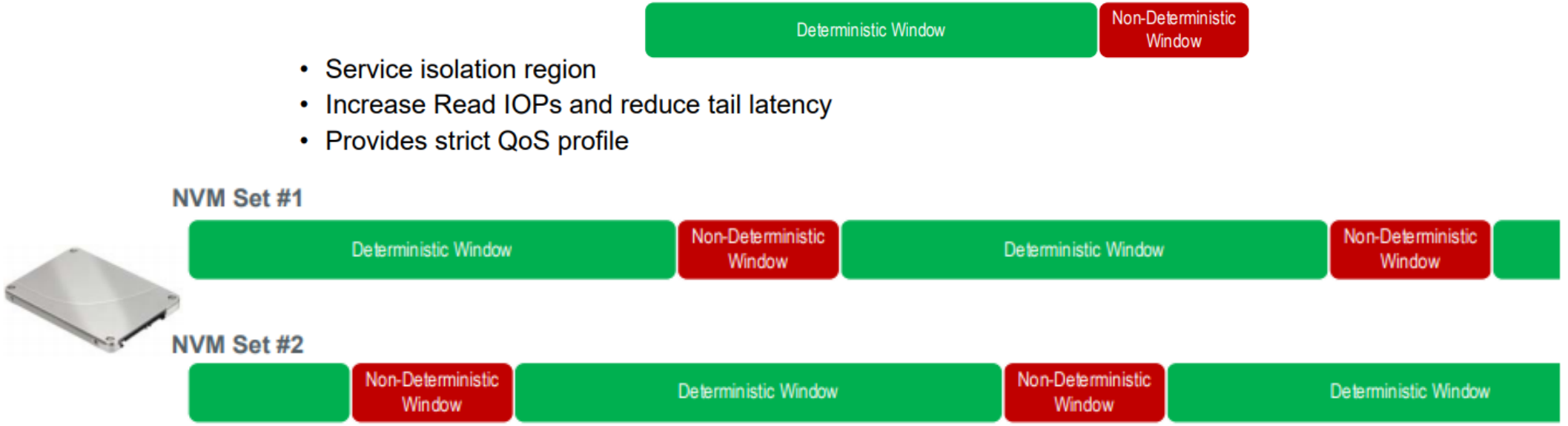
Features

■ IO Determinism – Predictable Latency Mode

Predictable Latency Mode is used to achieve predictable latency for read and write operations. When configured to operate in this mode using the Predictable Latency Mode Config Feature, the namespaces in an NVM Set provide windows of operation for deterministic operation or non-deterministic operation.

When Predictable Latency Mode is enabled:

- ☐ NVM Sets and their associated namespaces have vendor specific quality of service attributes;
- ☐ I/O commands that access NVM in the same NVM Set have the same quality of service attributes; and
- ☐ I/O commands that access NVM in one NVM Set do not impact the quality of service of I/O commands that access NVM in a different NVM Set.

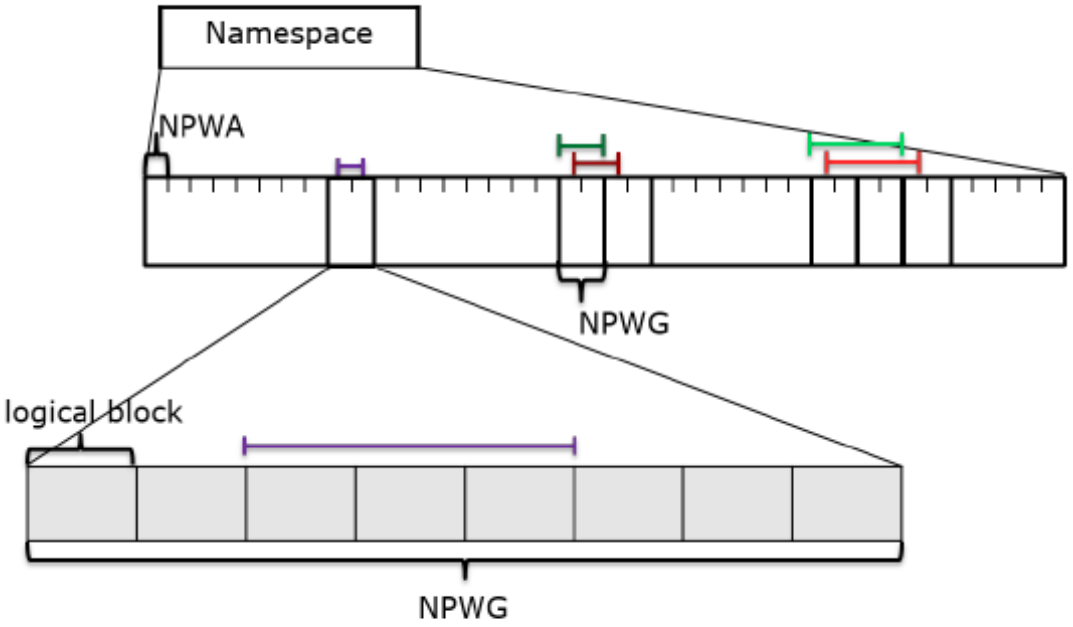


Features

■ IO Performance and Endurance Hints

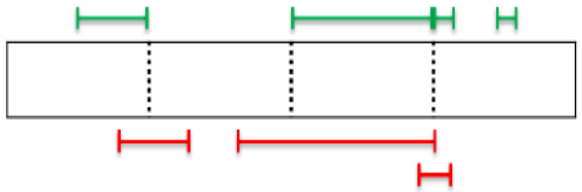
Defines performance and endurance hints enabling the controller to indicate granularities and alignments that are preferred for Write and Deallocate operations.

Figure 503: Example namespace broken down to illustrate potential NPWA and NPWG settings



Legend	
	Conformant I/O
	Non-Conformant I/O
Namespace Preferred Write Alignment (NPWA)	
Namespace Preferred Write Granularity (NPWG)	

Figure 501: An example namespace with four NOIOBs



Legend	
	Namespace
	Namespace Optimal IO Boundary (NOIOB)
	Conformant I/O
	Non-Conformant I/O

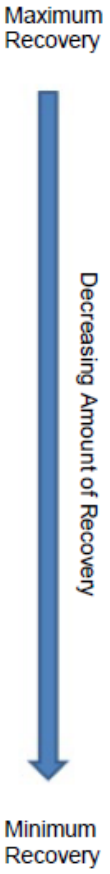
Features

■ Read Recovery Level

The Read Recovery Level (RRL) is a NVM Set configurable attribute that balances the completion time for read commands and the amount of error recovery applied to those read commands.

Figure 485: Read Recovery Level Overview

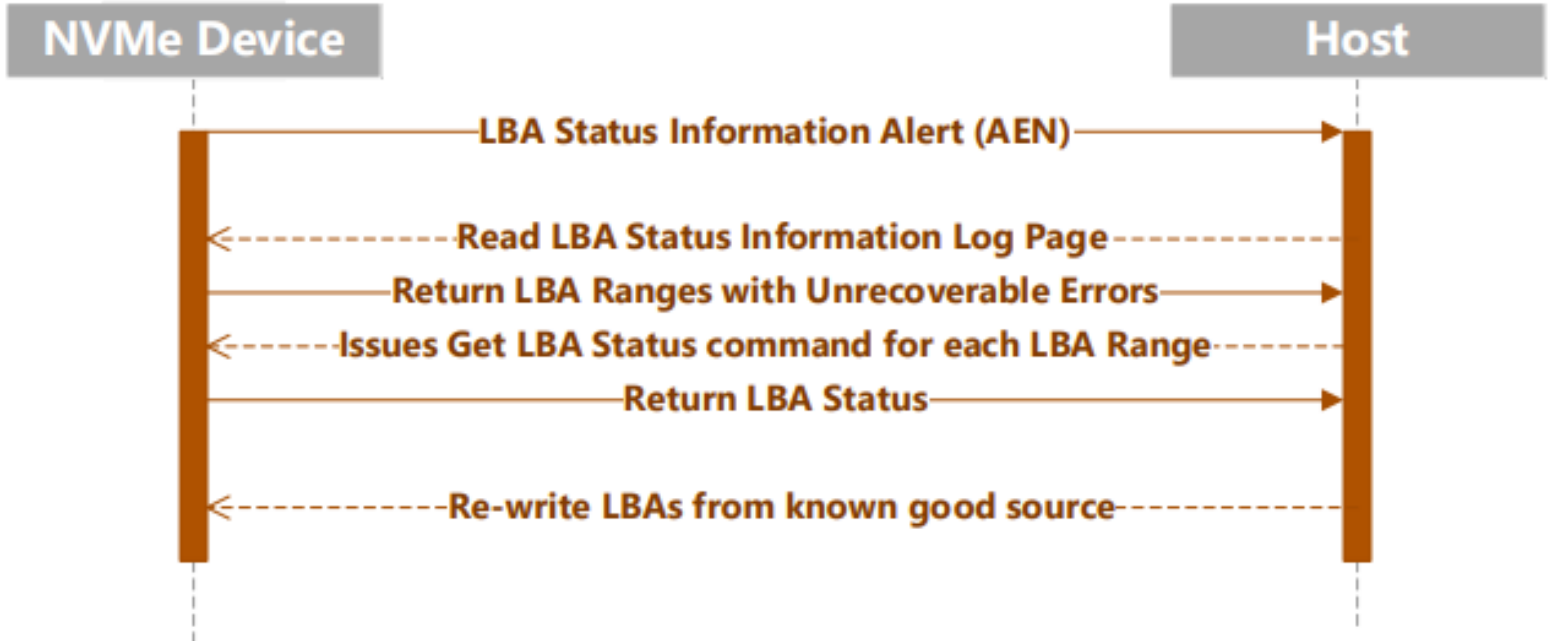
Level	O/M	Description
0	O	
1	O	
2	O	
3	O	
4	M	Default
5	O	
6	O	
7	O	
8	O	
9	O	
10	O	
11	O	
12	O	
13	O	
14	O	
15	M	Fast Fail



Features

■ Rebuild Assist

- ☐ Host is able to configure NVMe™ Device for notifications about potentially unrecoverable LBAs. This may be used to determine what LBAs on a device should be recovered by the host from another location and re-written.
- ☐ Establishes mechanism for early communication of two types of errors:
  - ● 'Tracked LBA' list – LBAs discovered to now be bad by Device through a background scan.
  - ● 'Untracked LBA' list – LBA ranges associated with a component failure by a scan originated by a Get LBA Status command.



Features

■ Verify Command

New Command to check the integrity of stored data and metadata.

- Effectively acts as Read without transferring any data or metadata to the Host
- Controller reads & discards the data – while performing equivalent Protection Information checks
- Errors are generated if data cannot be read correctly

Drive diagnostics & data scrubbing during drive operation require integrity verification, but don't require access to the actual data.

Verify significantly increases the efficiency of this type of operation!

Features

■ Persistent Event Log

The Persistent Event Log page contains information about significant events not specific to a particular command. The information in this log page shall be retained across power cycles and resets. This log page is global to the NVM subsystem.

- The Persistent Event Log defines the features necessary to build a scaffolding that enables extensible debug infrastructure that is usable at scale.
- Comprehensive set of events defined

- Health Snapshot
- Firmware Commits
- Timestamp Changes
- Power-on or Resets
- Thermal Excursions
- Vendor Specific
- TCG-defined Events
- Hardware Errors
- Changed Namespace
- Set Feature Events
- Format NVM Start & Complete
- Sanitize Start & Complete

Allows SSD customers to get consistent debug capabilities across vendors!

Allows SSD vendors an extensible framework for custom debug content!

Features

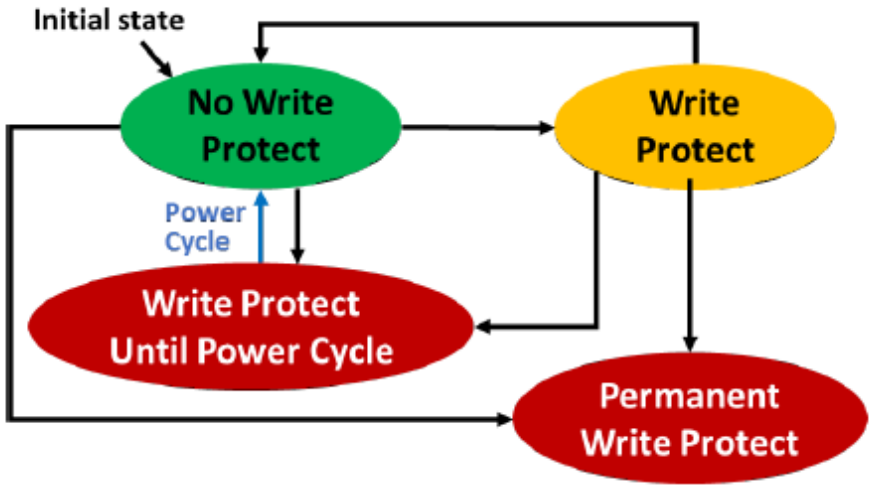
■ Namespace Write Protection

Namespace Write Protection is an optional configurable controller capability that enables the host to control the write protection state of a namespace or to determine the write protection state of a namespace.

Figure 490: Namespace Write Protection State Definitions

State	Definition	Persistent Across	
		Power Cycles	Controller Level Resets
No Write Protect	The namespace is not write protected.	Yes	Yes
Write Protect	The namespace is write protected.	Yes	Yes
Write Protect Until Power Cycle	The namespace is write protected until the next power cycle.	No	Yes
Permanent Write Protect	The namespace is permanently write protected.	Yes	Yes

Figure 491: Namespace Write Protection State Machine Model



Allowed Commands under WP

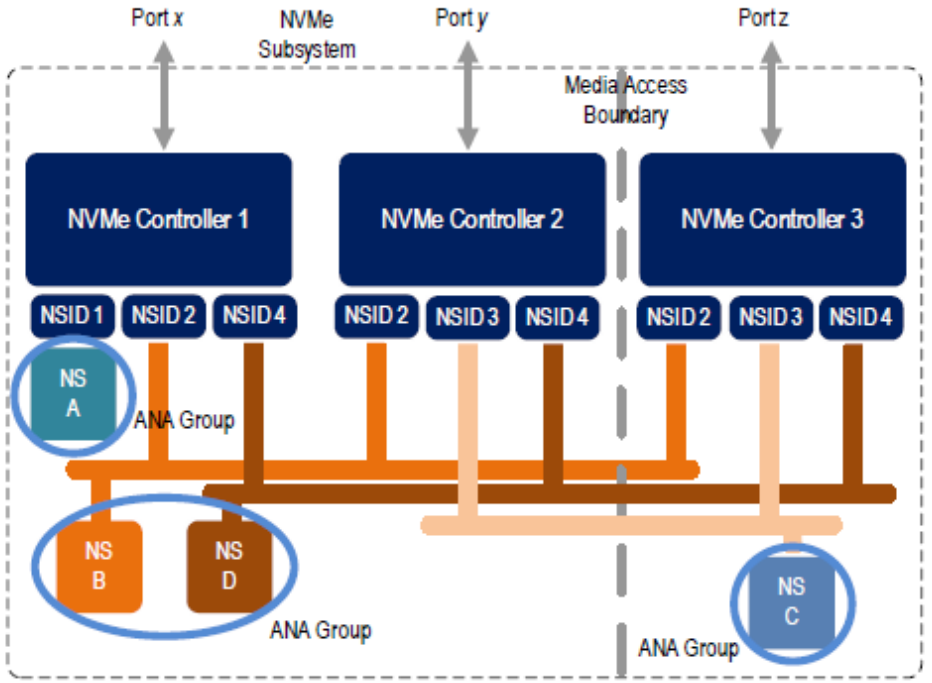
Admin Command Set	NVM Command Set
Device Self–test	Compare
Directive Send	Dataset Management
Directive Receive	Read
Get Features	Reservation Register
Get Log Page	Reservation Report
Identify	Reservation Acquire
Namespace Attachment	Reservation Release
Security Receive	Vendor Specific
Security Send	Flush
Set Features	Verify
Vendor Specific	

Features

■ Asymmetric Namespace Access Reporting

- ANA occurs in environments where namespace access characteristics may vary based on the controller used to access the namespace and the internal configuration of the NVM subsystem. ANA Reporting is used to indicate to the host information about those access characteristics.
- While commands may be sent to a shared namespace through any attached controller with asymmetric access, the characteristics may differ based on which controller is used; as a result, the host should consider those characteristics when selecting which controller to use for each command that accesses the namespace.

Figure 495: Multiple Namespace groups

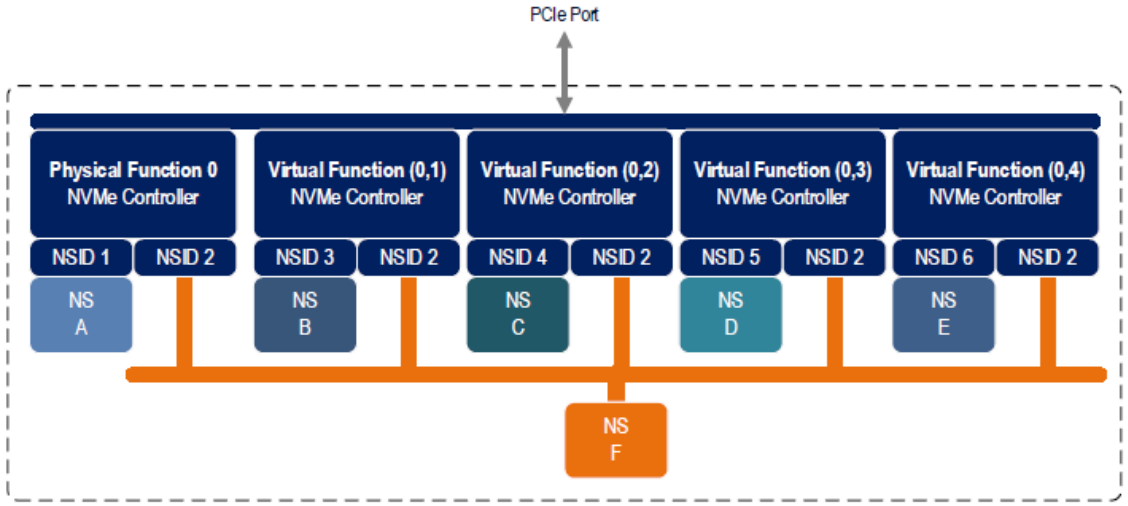


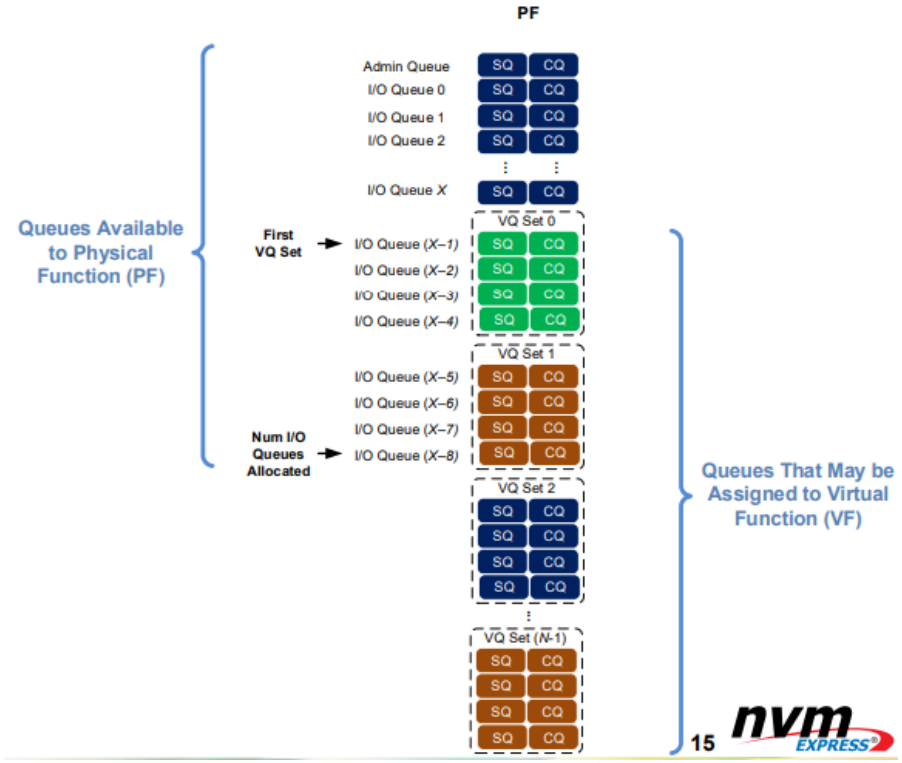
Features

■ Virtualization Enhancements

- The NVMe model has primary controllers (which may be SR-IOV Physical Functions) and secondary controllers (which may be SR-IOV Virtual Functions) that may be used to flexible assign resources, like queues, from a primary controller to a secondary controller.
- Controller resources may be assigned or removed from a controller using the Virtualization Management command issued to a primary controller. The following types of controller resources are defined:
  - Virtual Queue Resource (VQ Resource): a type of controller resource that manages one SQ and one Completion Queue (CQ)
  - Virtual Interrupt Resource (VI Resource): a type of controller resource that manages one interrupt vector.

Figure 6: PCI Express Device Supporting Single Root I/O Virtualization (SR-IOV)



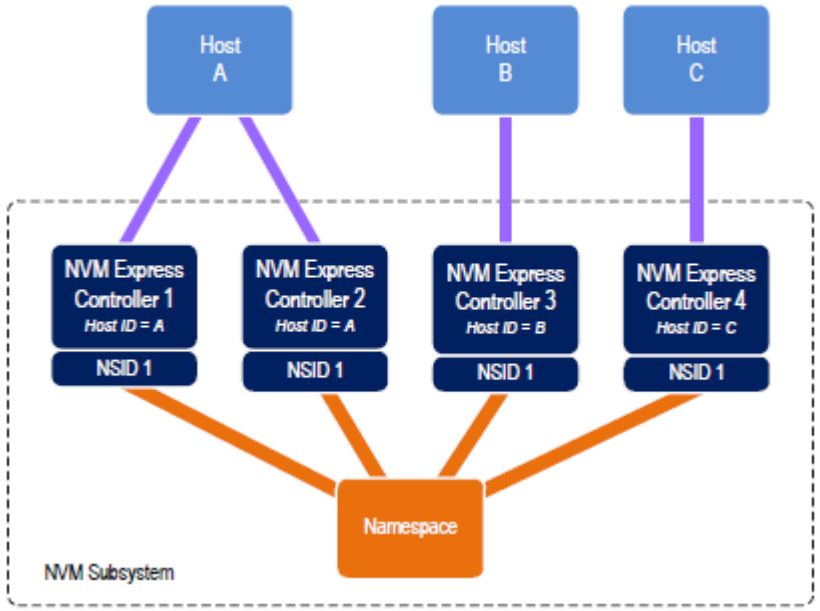


Features

■ Reservations

- ☐ Reservations provide capabilities that may be utilized by two or more hosts to coordinate access to a shared namespace.
- ☐ A reservation on a namespace restricts hosts access to that namespace. If a host submits a command to a namespace in the presence of a reservation and lacks sufficient rights, then the command is aborted by the controller with a status of Reservation Conflict.

Figure 462: Example Multi-Host System



•Example: Host A and B have read/write access and host C has read-only access to the shared namespace

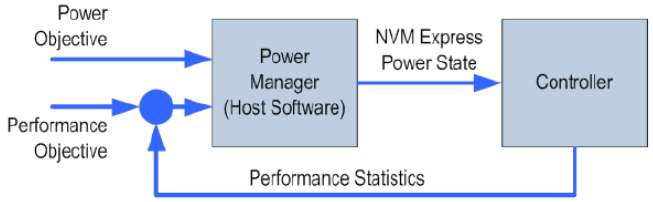
HostA-Register(NSID,Key_A) -> OK
HostB-Register(NSID,Key_B) -> OK
HostA-AcquireReservation(NSID, Reservation, WriteExclusiveRegistrantsOnly,Key_A) -> OK
HostB-AcquireReservation(NSID, Reservation, WriteExclusiveRegistrantsOnly,Key_B) -> OK
HostC-AcquireReservation(NSID, Reservation, WriteExclusiveRegistrantsOnly,Key_C) -> **Error – Reservation Conflict**
…
HostA-Read(NSID) -> OK
HostA-Write(NSID) -> OK
…
HostB-Read(NSID) -> OK
HostB-Write(NSID) -> OK
…
HostC->Read(NSID) -> OK
HostC->Write(NSID) -> **Error – Reservation Conflict**
…
HostA-ReleaseReservation(NSID, Key_A) -> OK
HostC-Write(NSID) -> OK
…

Features

■ Power Management

- ☐ The power management capability allows the host to manage NVM subsystem power statically or dynamically.
- ☐ The number of power states implemented by a controller is returned in the Number of Power States Supported (NPSS) field in the Identify Controller data structure.
- ☐ **A**utonomous **P**ower **S**tate **T**ransitions provide a mechanism for the host to configure the controller to automatically transition between power states on certain conditions without software intervention.

Figure 457: Dynamic Power Management

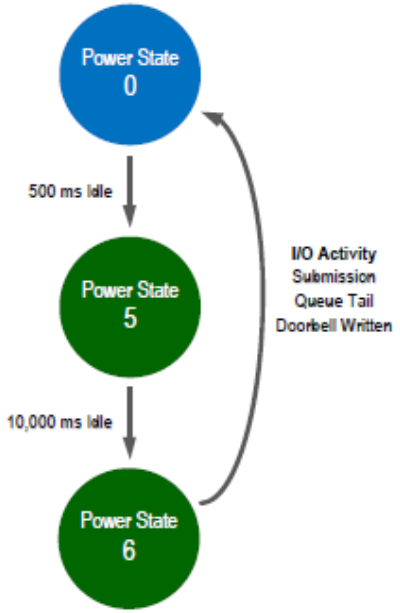


Power State Descriptor Table

Power State	Maximum Power	Operational State	Entry Latency	Exit Latency	Relative Read Throughput	Relative Read Latency	Relative Write Throughput	Relative Write Latency
0	25 W	Yes	5 µs	5 µs	0	0	0	0
1	18 W	Yes	5 µs	7 µs	0	0	1	0
2	18 W	Yes	5 µs	8 µs	1	0	0	0
3	15 W	Yes	20 µs	15 µs	2	1	2	1
4	7 W	Yes	20 µs	30 µs	1	2	3	1
5	1 W	No	100 mS	50 mS	-	-	-	-
6	.25 W	No	100 mS	500 mS	-	-	-	-

Autonomous Power State Transition Table

Idle Time Prior to Transition	Idle Transition Power State
500 ms	5
500 ms	5
500 ms	5
500 ms	5
500 ms	5
10,000 ms	6
-	-



Features

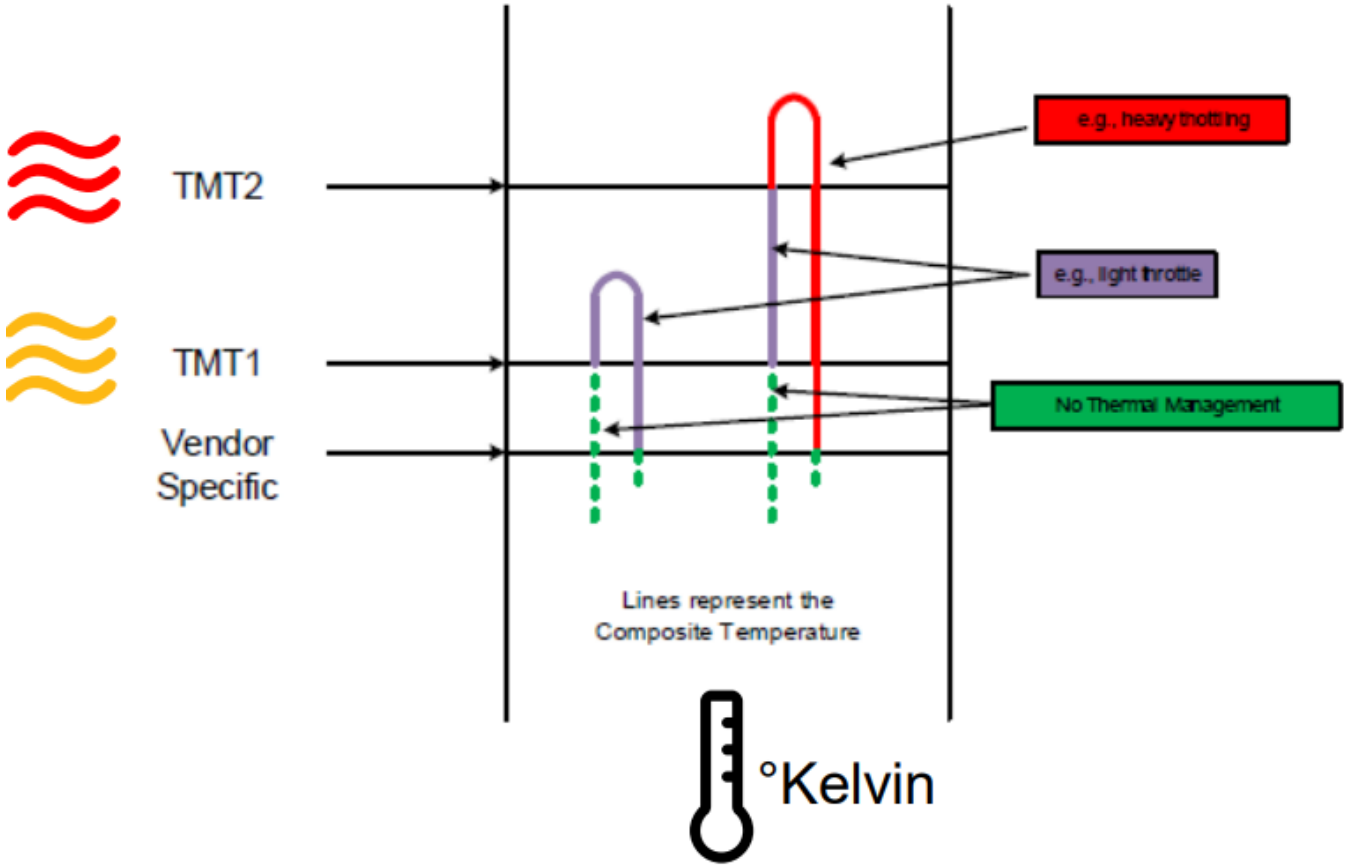
■ Host Controlled Thermal Management

Better thermal management in client systems like laptops and desktops.

Host can set Thermal Management Temperature at which a device should start going into a lower power state / throttling

- TMT1 – host tells SSD what temp in degrees K it should start throttling at
- TMT2 – threshold where the SSD should start heavy throttling regardless of impact to performance

Figure 264: HCTM Example



Features

■ Device Self-test Operations

Device Self Test

Host system can request the storage device (SSD) do perform tests to ensure it is functioning properly

Short – less than 2 min

Long – will continue after reset (can send format or another DST to stop)

Figure 280: Example Device Self-test Operation (Informative)

Segment		Test Performed	Failure Criteria
1 – RAM Check		Write a test pattern to RAM, followed by a read and compare of the original data.	Any uncorrectable error or data miscompare
2 – SMART Check		Check SMART or health status for Critical Warning bits set to '1' in SMART / Health Information Log.	Any Critical Warning bit set to '1' fails this segment
3 – Volatile memory backup		Validate volatile memory backup solution health (e.g., measure backup power source charge and/or discharge time).	Significant degradation in backup capability
4 – Metadata validation		Confirm/validate all copies of metadata.	Metadata is corrupt and is not recoverable
5 – NVM integrity		Write/read/compare to reserved areas of each NVM. Ensure also that every read/write channel of the controller is exercised.	Data miscompare
Extended only	6 – Data Integrity	Perform background housekeeping tasks, prioritizing actions that enhance the integrity of stored data. Exit this segment in time to complete the remaining segments and meet the timing requirements for extended device self-test operation indicated in the Identify Controller data structure.	Metadata is corrupt and is not recoverable
7 – Media Check		Perform random reads from every available good physical block. Exit this segment in time to complete the remaining segments. The time to complete is dependent on the type of device self-test operation.	Inability to access a physical block
8 – Drive Life		End-of-life condition: Assess the drive's suitability for continuing write operations.	The Percentage Used is set to 255 in the SMART / Health Information Log or an analysis of internal key operating parameters indicates that data is at risk if writing continues
9 – SMART Check		Same as 2 – SMART Check	

Features

■ Sanitize Operations

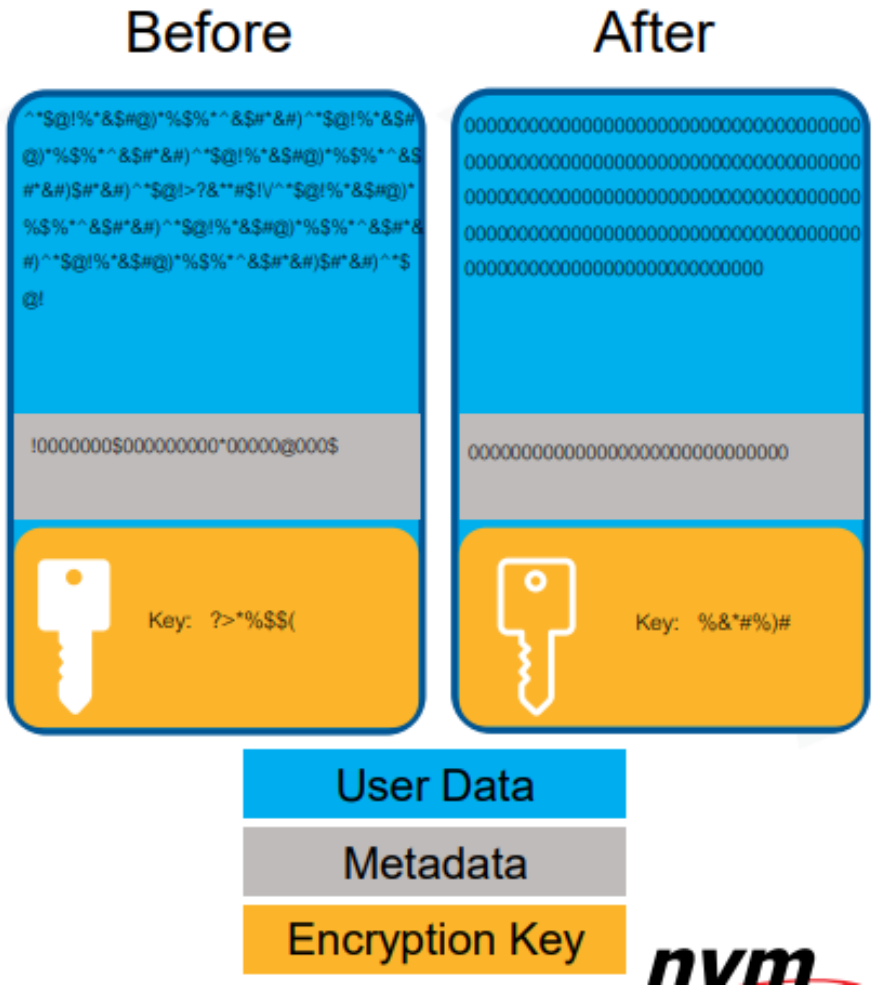
Alters user data so that is is unrecoverable by erasing media, metadata, and cache

Use when retiring SSD from use, reusing for new use case, or end of life

Modes in Sanitize

- Block Erase – low level block erase on media (physically erase NAND blocks)
- Crypto Erase - change media encryption key
- Overwrite – overwrite with data patterns (not good or recommended for NAND based SSDs due to endurance)

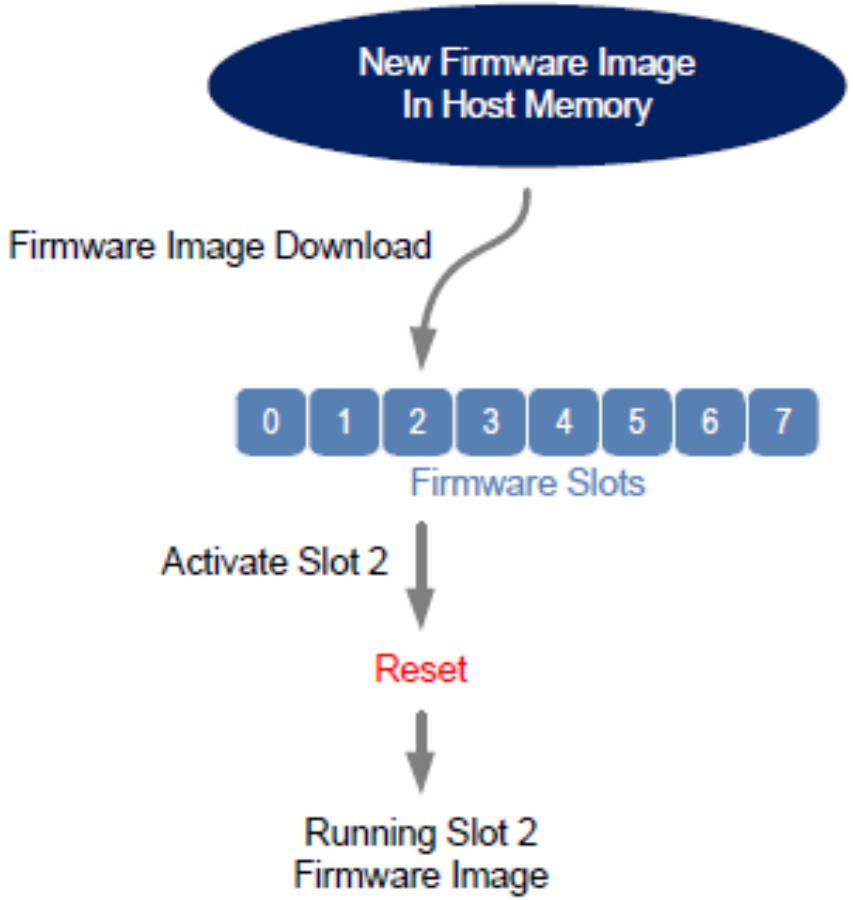
Sanitize vs Format Unit in NVMe – keeps going after reset, and erases all metadata, log pages and status during operation



Features

■ Firmware Update Process

- ☐ Firmware update to be activated by a reset
  - ● Download firmware image to firmware slot
  - ● Activate firmware image
  - ● Perform reset to cause new firmware image to run
    - • Controller reset
    - • PCIe conventional reset
    - • Subsystem reset
- ☐ Firmware update to be activated without a reset
  - ● Download firmware image to firmware slot
  - ● The host submits a Firmware Commit command with a Commit Action of 011b which specifies that the image should be activated immediately without reset.

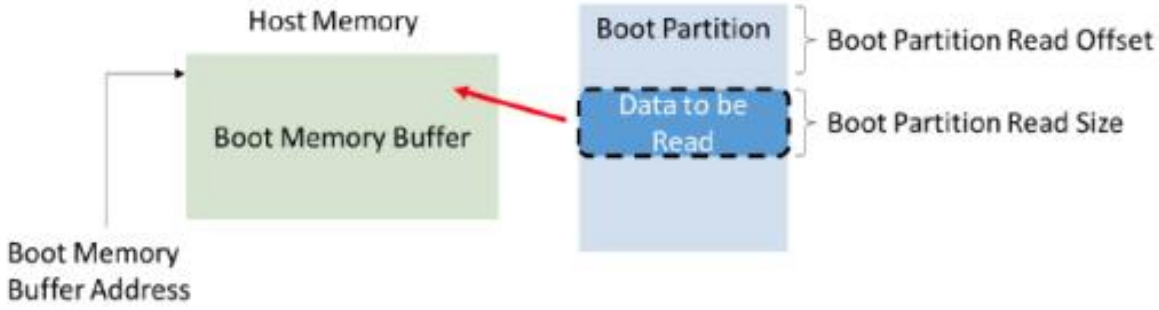


Features

■ Boot Partitions

- Optional storage area that can be read with “fast” initialization method (not standard NVMe queues). Example: UEFI bootloader
- Saves cost and space by removing the need for another storage medium (like SPI flash, EPROM)
- Write using standard NVMe Firmware Download and Firmware Commit
- Can be protected with **Replay Protected Memory Block**

Figure 281: Boot Partition Overview



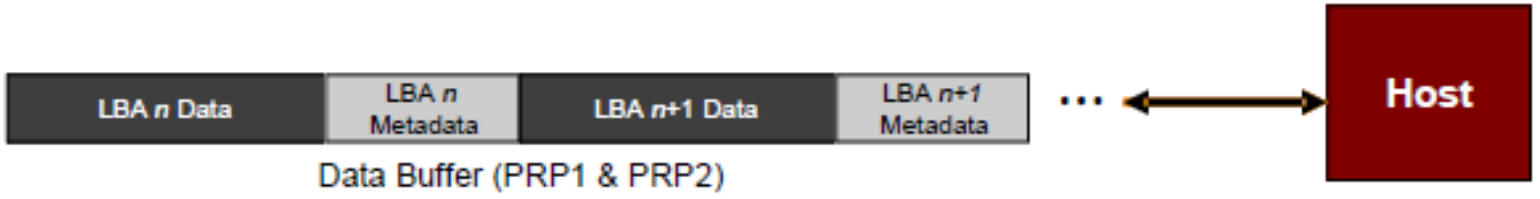
Features

■ Metadata Handling

Metadata is additional data allocated on a per logical block basis. One of the most common usages for metadata is to convey end-to-end protection information.

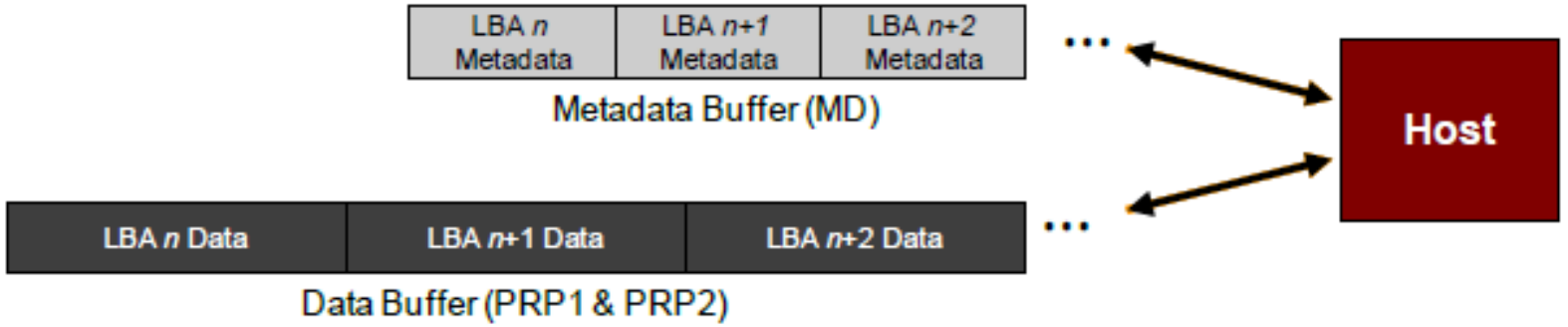
☐ Contiguous with LBA Data

Figure 451: Metadata – Contiguous with LBA Data, Forming Extended LBA



☐ Transferred as Separate Buffer

Figure 452: Metadata – Transferred as Separate Buffer



Features

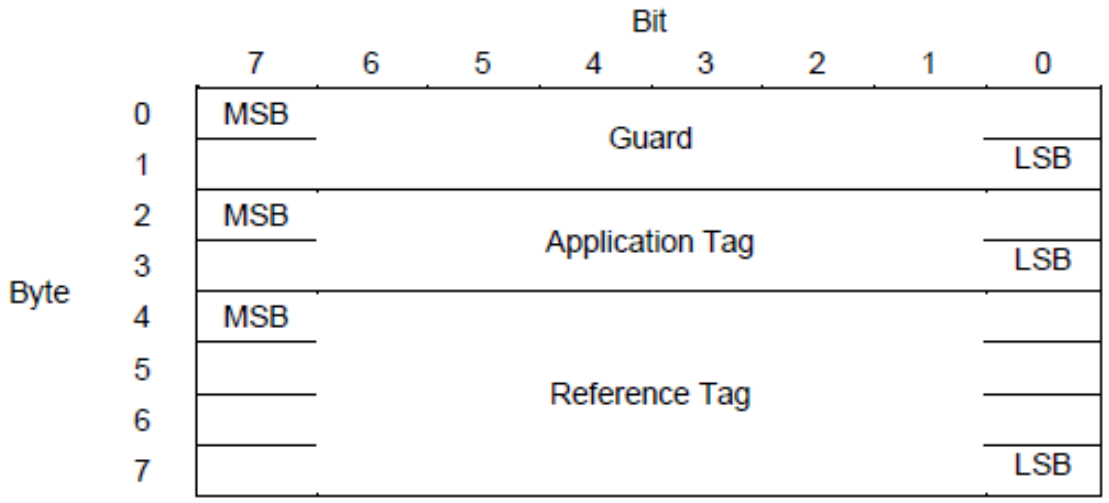
■ End-to-end Data Protection

To provide robust data protection from the application to the NVM media and back to the application itself, end-to-end data protection may be used. This additional protection information, if present, is either the first eight bytes of metadata or the last eight bytes of metadata, based on the format of the namespace.

The Protection Information format is contained in the metadata associated with each logical block.

- The Guard field contains a CRC-16 computed over the logical block data.
- The Application Tag is an opaque data field not interpreted by the controller.
- The Reference Tag associates logical block data with an address and protects against misdirected or out-of-order logical block transfer.

Figure 453: Protection Information Format

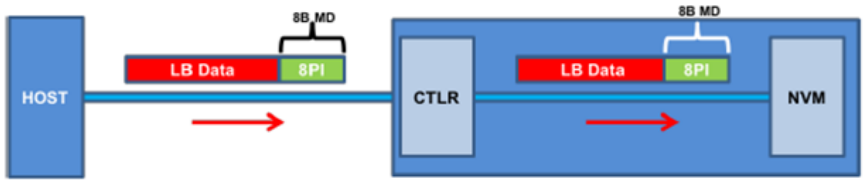


Features

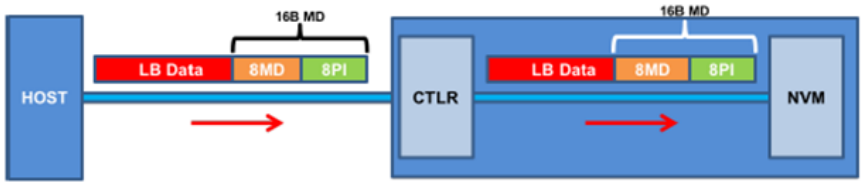
☐ The PRACT Bit

Protection Information(PRINFO) = Protection Information Action(PRACT) + Protection Information Check(PRCHK)

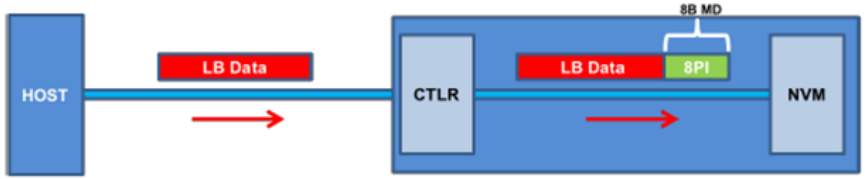
Figure 454: Write Command Protection Information Processing



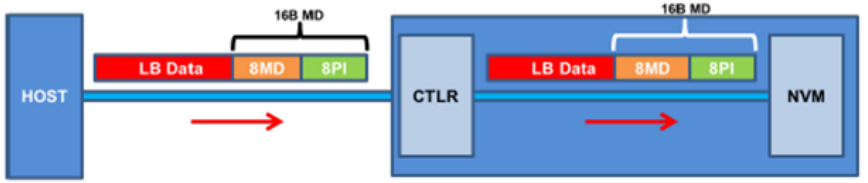
a) MD=8, PI, PRACT=0: Metadata remains same size in NVM and host buffer



b) MD>8 (e.g., 16), PI, PRACT=0: Metadata remains same size in NVM and host buffer



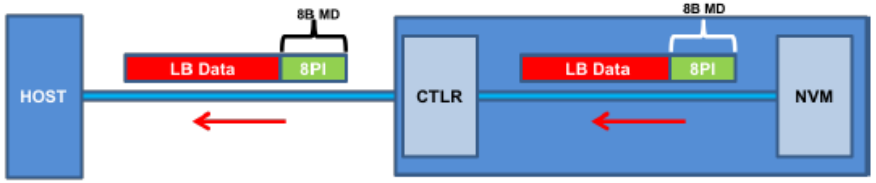
c) MD=8, PI, PRACT=1: Metadata not resident in host buffer



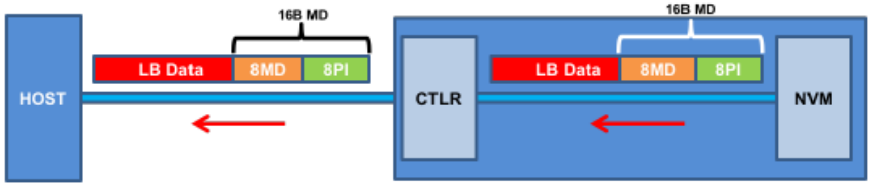
d) MD>8 (e.g., 16), PI, PRACT=1: Metadata remains same size in NVM and host buffer

NOTE: In cases (b) and (d) the Protection Information could be before or after the 8 bytes of metadata.

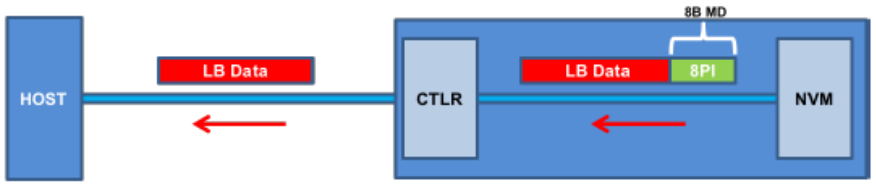
Figure 455: Read Command Protection Information Processing



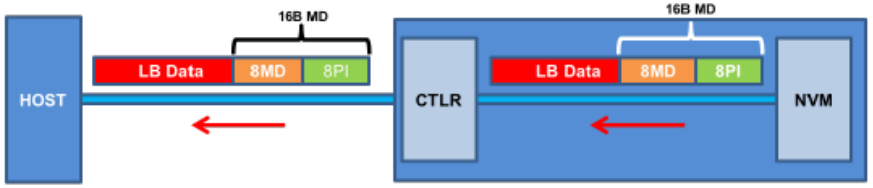
a) MD=8, PI, PRACT=0: Metadata remains same size in NVM and host buffer



b) MD>8 (e.g., 16), PI, PRACT=0: Metadata remains same size in NVM and host buffer



c) MD=8, PI, PRACT=1: Metadata not resident in host buffer



d) MD>8 (e.g., 16), PI, PRACT=1: Metadata remains same size in NVM and host buffer

NOTE: In cases (b) and (d) the PI could be before or after the 8 bytes of metadata.

NVM Express™

Directives

Directives

Directives is a mechanism to enable host and NVM subsystem or controller information exchange. The Directive Receive command is used to transfer data related to a specific Directive Type from the controller to the host. The Directive Send command is used to transfer data related to a specific Directive Type from the host to the controller. Other commands may include a Directive Specific value specific for a given Directive Type (e.g., the Write command in the NVM command set).

■ Directive Use in I/O Commands

In an I/O command, if the Directive Type is set to an I/O Command Directive, then the Directive Specific specifies the identifier of the stream associated with the data. The only I/O command that supports use of directives is the Write command.

■ Identify (Directive Type 00h)

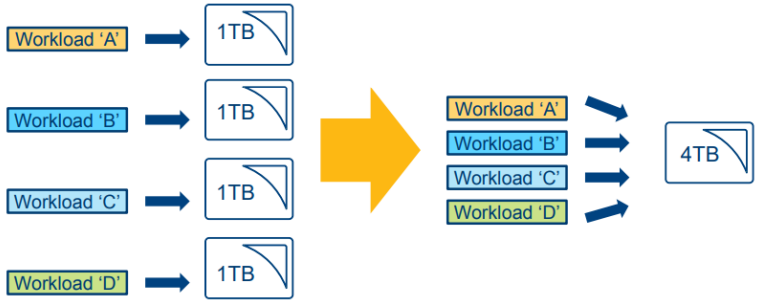
☐ Directive Receive

- The Return Parameters operation returns a data structure that contains a bit vector specifying the Directive Types supported by the controller and a bit vector specifying the Directive Types enabled for the namespace.

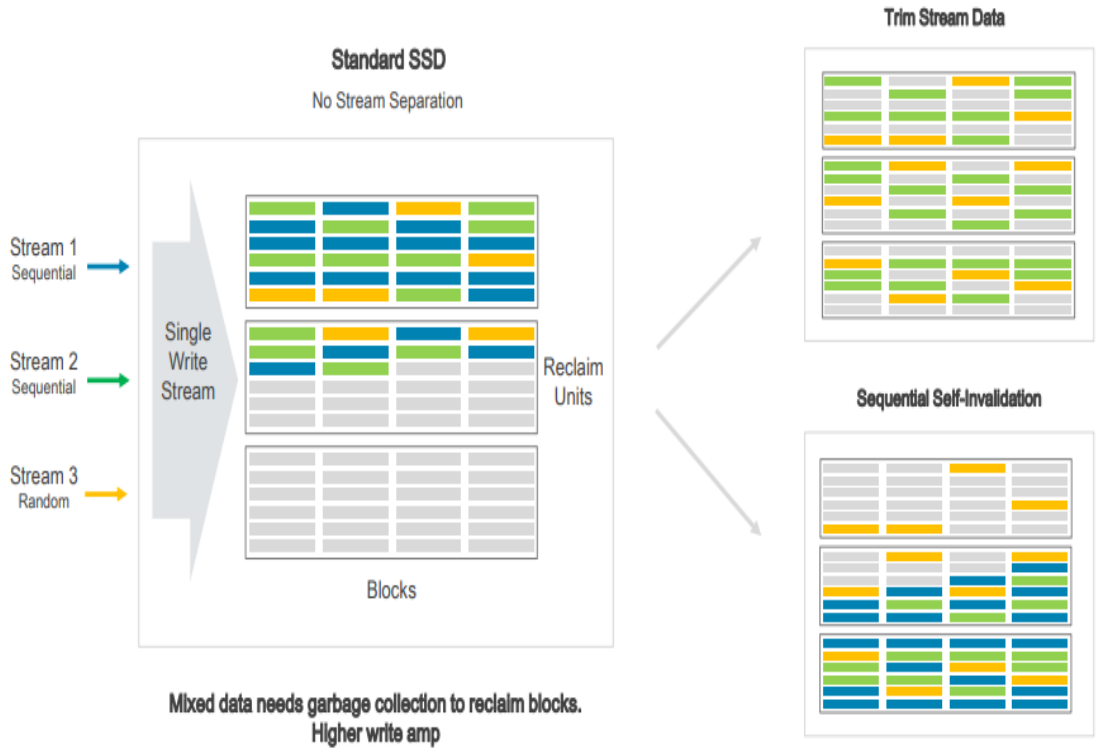
☐ Directive Send

- The Enable Directive operation is used to enable a specific Directive for use within a namespace by all controllers that are associated with the same Host Identifier.

Directives



Streams: Problem



Streams: Solution

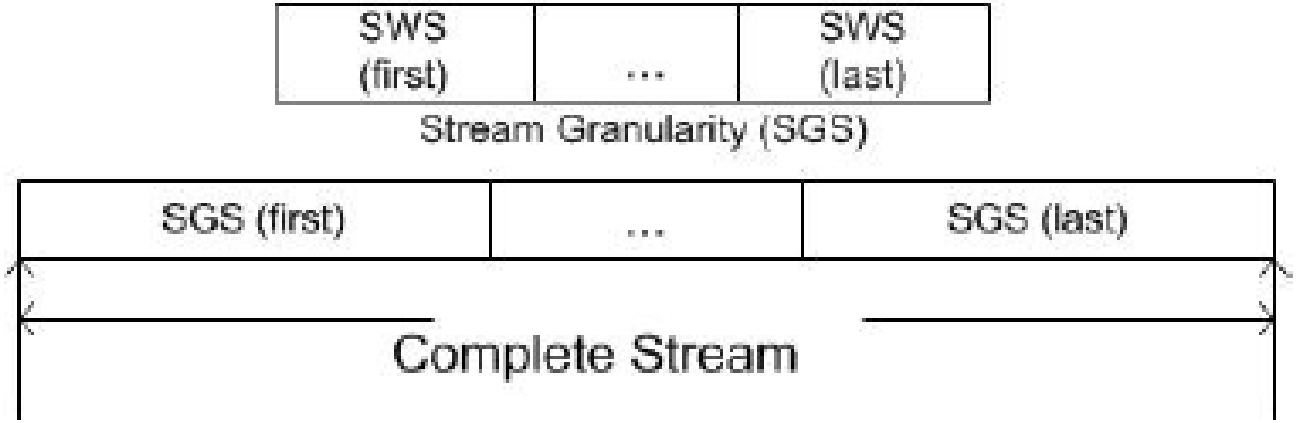
Directives

■ Streams (Directive Type 01h)

The Streams Directive enables the host to indicate (i.e., by using the stream identifier) to the controller that the specified logical blocks in a write command are part of one group of associated data. This information may be used by the controller to store related data in associated locations or for other performance enhancements.

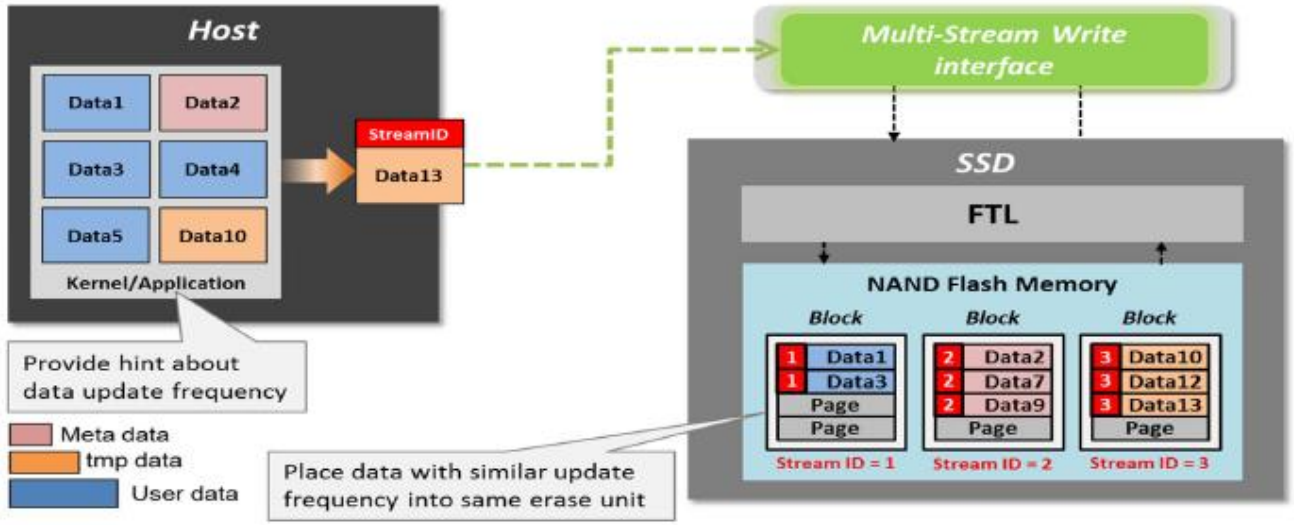
Data that is aligned to and in multiples of the Stream Write Size (SWS) provides optimal performance of the write commands to the controller. The Stream Granularity Size indicates the size of the media that is prepared as a unit for future allocation for write commands and is a multiple of the Stream Write Size. The controller may allocate and group together a stream in Stream Granularity Size (SGS) units.

Figure 513: Directive Streams – Stream Alignment and Granularity



Directives

- Directive Receive
- The Return Parameter operation returns a data structure that specifies the features and capabilities supported by the Streams Directive, including namespace specific values.
- The Get Status operation returns information about the status of currently open streams for the specified namespace and the host issuing the Get Status operation.
- The Allocate Resources operation indicates the number of streams that the host requests for the exclusive use for the specified namespace.
- Directive Send
- The Release Identifier operation specifies that the stream identifier is no longer in use by the host.
- The Release Resources operation is used to release all streams resources allocated for the exclusive use of the namespace attached to all controllers.



NVM Express™

Changes in NVMe 1.4

Changes in NVMe 1.4

- New Features (optional)
  - Persistent Memory Region
  - IO Determinism
  - Asymmetric Namespace Access
  - Namespace Write Protect
  - Persistent Event Log
  - Rebuild Assist
  - NVM Sets
  - Read Recovery Levels
  - Endurance Groups
  - IO Performance and Endurance Hints
  - Verify Command
  - Administrative Controller

Changes in NVMe 1.4

- Feature Enhancements
  - Host Memory Buffer Enhancements
  - Namespace Granularity
  - Transport SGL
  - Sanitize Enhancements
  - Shared Stream Write
  - Enhanced Command Retry
  - Controller Memory Buffer Extensions

Changes in NVMe 1.4

- Required Changes
  - New NSID value usages
  - New errors and reporting requirements
  - Temperature threshold clarifications
  - Controller Memory Buffer & Persistent Memory Region Enhancements
  - New Sanitize requirements
  - Reservation Notification Log usage
  - Clarified LBA Range feature behavior
  - Reservation Report command conflicts resolved
  - New Abort command behavior
  - Clarified Device Self-Test termination on Format scenario
  - Release of Stream Resources clarification

* Not to scale. These are ***categories*** of changes, not the full list of changes themselves

Changes in NVMe 1.4

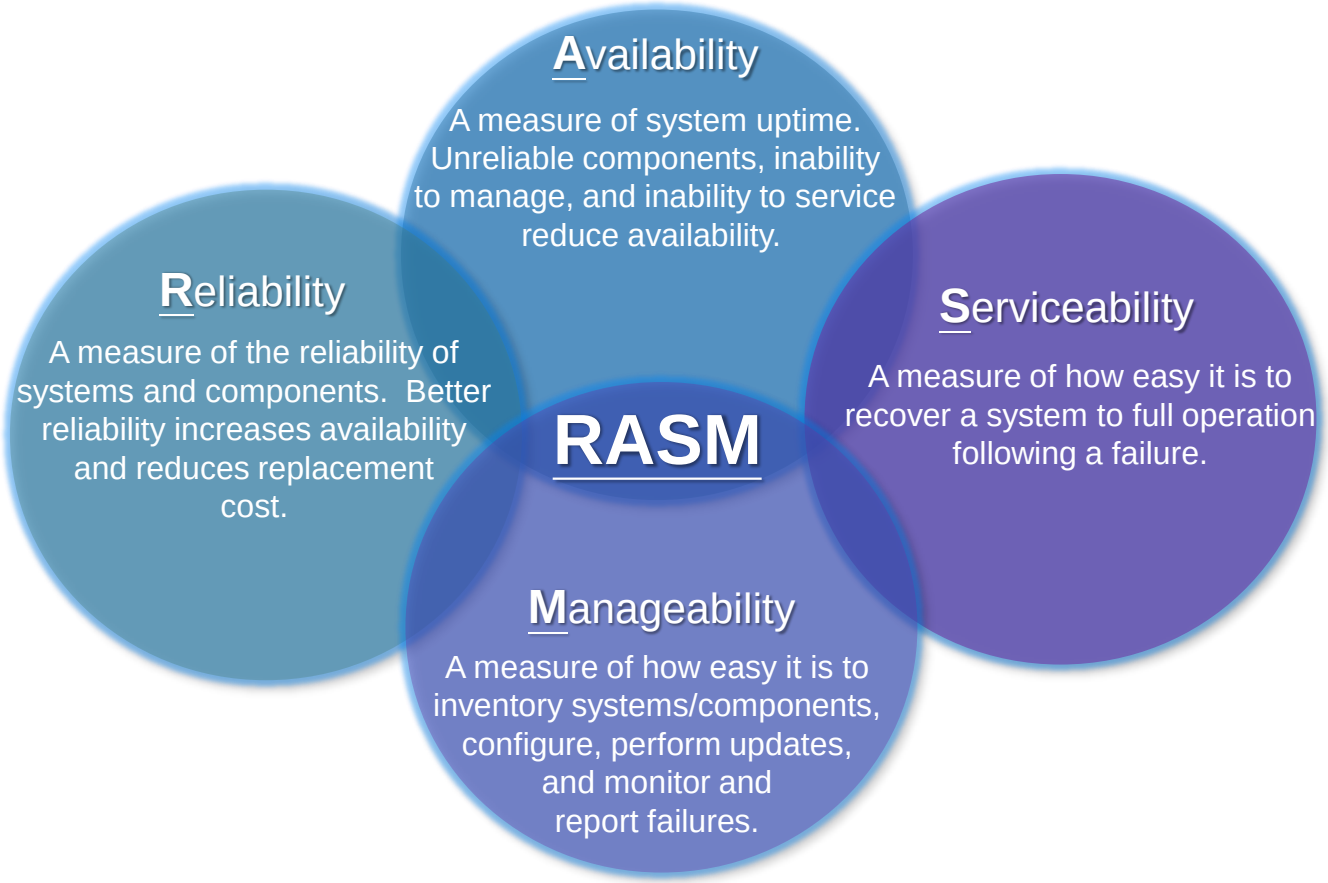
Example: Mandatory Change NSID value - FFFFFFFFh

- Overview – Namespace Identifiers
  - All usages of NSID value FFFFFFFF are now well-defined
  - Generally used to mean a broadcast action against all Namespaces
- What are the changes?
  - Clarifications in many sections: I/O Commands, Set/Get Features, Admin Commands, and Reservations
  - Explicitly defines when NSID of FFFFFFFF can be used and how to use it
- Why the changes?
  - The specification was quiet on a number of use cases
  - Need to provide consistency across Device and OS implementations
  - Improve the end-user experience and ease of NVMe™ device consumption
- Impacts of inaction
  - Inconsistent results when using devices from various hardware vendors

NVM Express™ Management Interface

Introduction

The Importance of Manageability in Servers



Better RASM = Reduced TCO

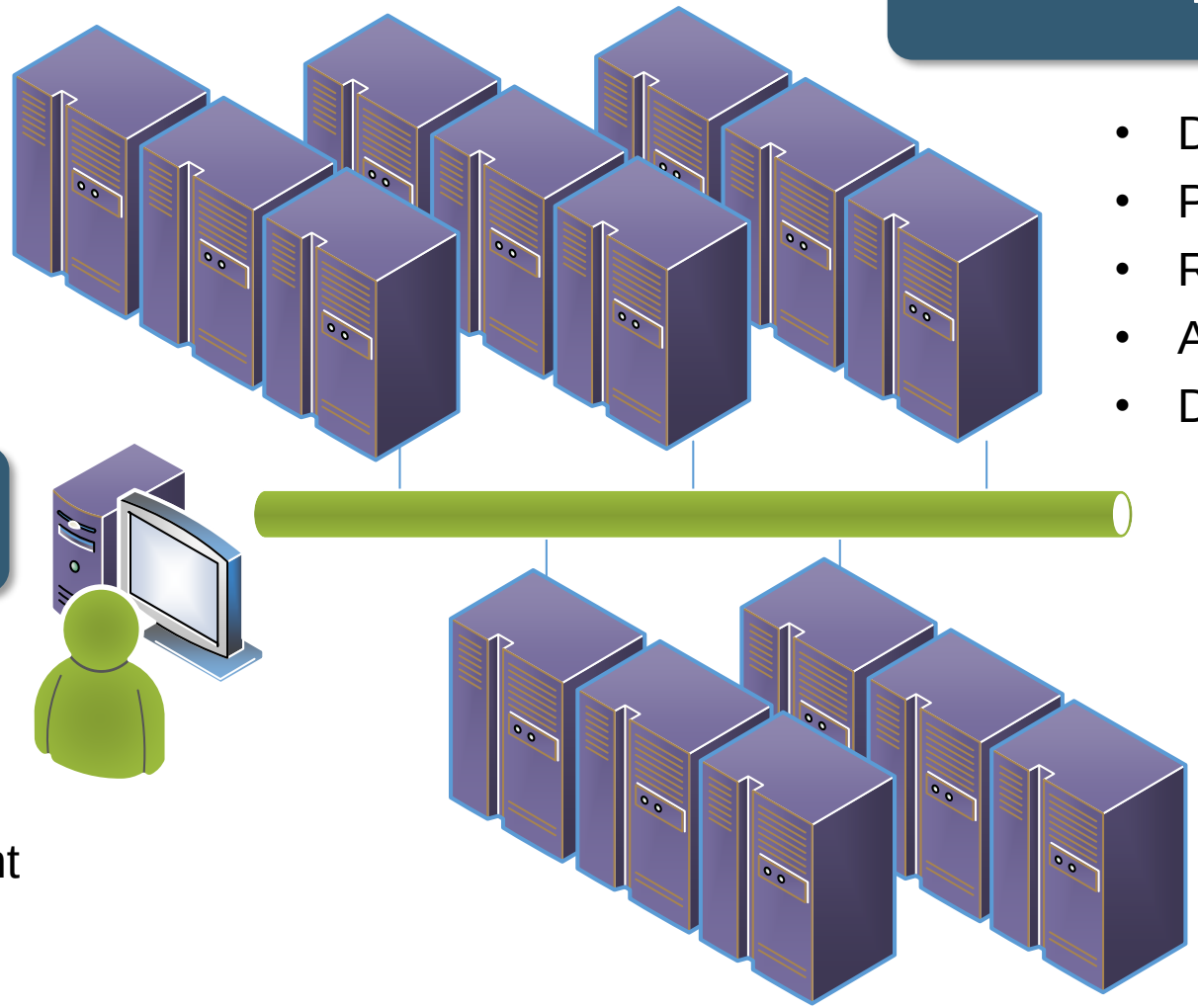
"Customers choose suppliers who provide the features that are important to them. Customers care about TCO (Total Cost of Ownership). Consequently, in the server space, MHz is not the only thing that's important: TCO is greatly affected by the RASM features of the servers. When server OEMs and users talk, their focus is RASM: Reliability, Availability, Serviceability, and Manageability. To a customer, RASM means dollars. Adding or improving on RASM reduces TCO.

The cost of downtime is extremely high. According to IMEX Research, the average cost of an unplanned outage runs into the hundreds of thousand of dollars."

Management Fundamentals

Management Operational Times

- Deployment (No OS)
- Pre-OS (e.g., UEFI/BIOS)
- Runtime
- Auxiliary Power
- Decommissioning



Pillars of Systems Management

- Inventorying
- Configuration
- Monitoring
- Change Management

Sample Use Cases

Use Case	Benefit
Inventorying	Asset management. Re-provisioning systems. Track quality of components.
Health monitoring	Identify bad drives for quick replacement.
Wear monitoring	Replace drives nearing wear-out to avoid failure.
Temp. monitoring	Fan throttling reduces power, noise, and fan wear.
Power monitoring and configuration	Power throttling to save energy and cool system.
Perf. monitoring	Look for performance bottlenecks.
Configuring	Format drives for initial use. Crypto erase drives for re-provisioning or decommissioning.
Change Mgmt.	Update drive firmware for bug fixes and security patches.

What is the NVMe™ Management Interface?

- Allows a Management Controller to communicate out-of-band with an NVMe Storage Device or NVMe Enclosure over one or more external interfaces.
- The in-band tunneling mechanism which allows the NVMe-MI Management Interface Command Set to be tunneled in-band via the NVMe Admin Commands NVMe-MI Send and NVMe-MI Receive to an NVMe Storage Device or NVMe Enclosure.

- Management Controller – A device (e.g., Baseboard Management Controller) responsible for platform management that uses the NVM Express Management Interface to communicate to Management Endpoints.
- out-of-band management – A management that operates with hardware resources and components that are independent of the operating system's control.
- in-band management – A management that operates with the support of hardware components that are critical to and used by the operating system.
- NVMe Storage Device – A logical or physical component, device, or assembly that contains at least one NVM Subsystem or Expansion Connector, at least one Upstream Connector, and at least one FRU Information Device.
- NVMe Enclosure – A platform, card, module, box, rack, or set of boxes that may provide power, cooling, mechanical protection and/or external interfaces for zero or more NVMe Storage Device FRUs. An NVMe Enclosure contains one or more NVM Subsystems and one or more Enclosure Services Processes.

NVM Express™ Management Interface

Theory & Architecture

FRU Information Device

- Field-Replaceable Unit (FRU)

  A physical component, device, or assembly in a system that is able to be removed and replaced (e.g., by an end user or technician) without having to replace the entire system in which it is contained.

- FRU Information Device

  A logical or physical device used to hold the Vital Product Data (VPD). A FRU Information Device may be implemented in a variety of ways (e.g., a serial EEPROM, one-time programmable memory in silicon, etc.).

- NVMe Storage Device FRU

  An NVMe Storage Device that is able to be removed and replaced (e.g., by an end user or technician) without having to replace the entire system in which it is contained. Examples of NVMe Storage Device Field-Replaceable Units include a U.2 PCIe SSD, a PCI Express Card Electromechanical add-in card, or an M.2 module.

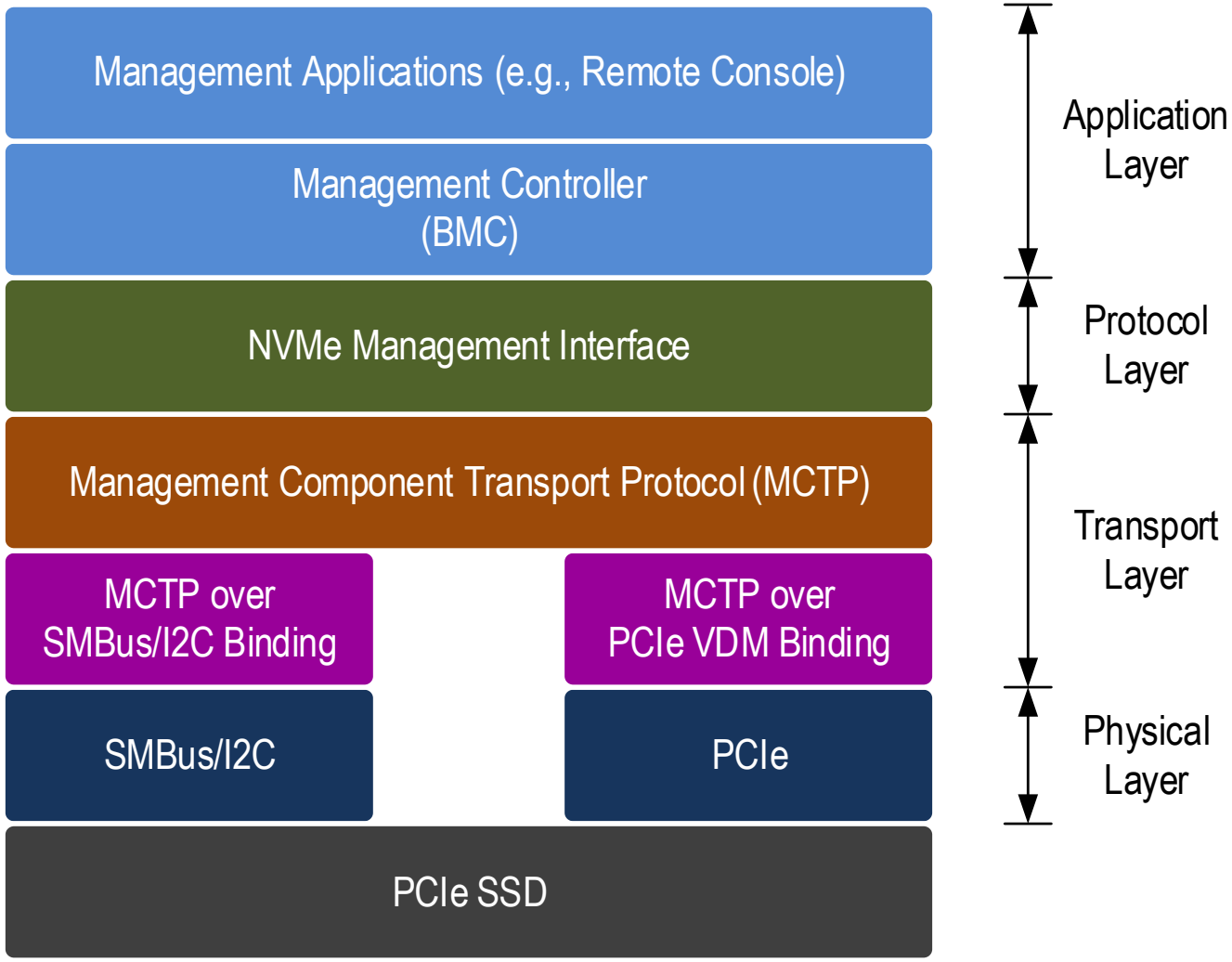
Vital Product Data (VPD)

- The Vital Product Data (VPD) is information describing an NVMe Storage Device. Each NVMe Storage Device FRU shall have a FRU Information Device with a size of 256 bytes to hold the VPD as defined in the IPMI Platform Management FRU Information Storage Definition.
- The VPD may be accessed using two methods
  - NVMe-MI commands over MCTP
  - SMBus/I2C interface using I2C operations as defined by IMPI Platform Management FRU Information Storage Definition
- The VPD for NVMe Storage Device FRUs shall contain the required elements:

Figure 144: VPD Elements

Byte	Name
7:0	Common Header
Vendor Specific	Product Info Area (Optional)
Vendor Specific	MultiRecord Info Area
Vendor Specific	Internal Use Area (Optional)
Vendor Specific	Chassis Info Area (Optional)
Vendor Specific	Board Info Area (Optional)

NVMe-MI™ Out-of-Band Protocol Layering

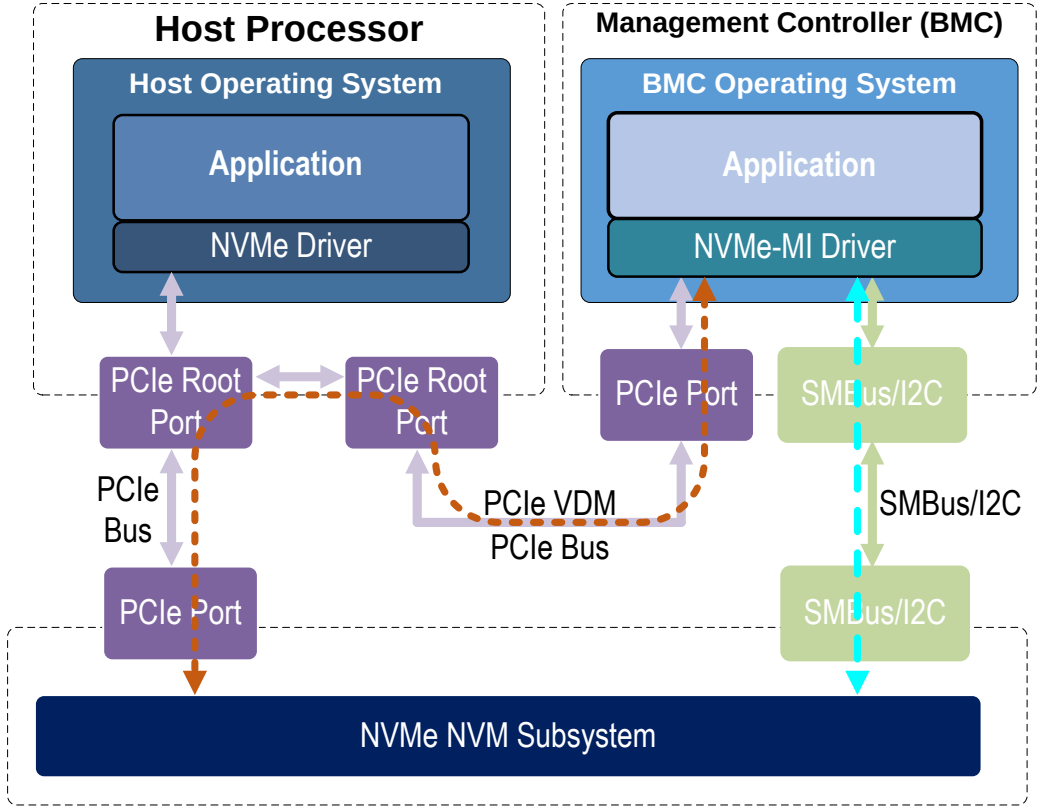


Out-of-Band Management and NVMe-MI™ in 1.0

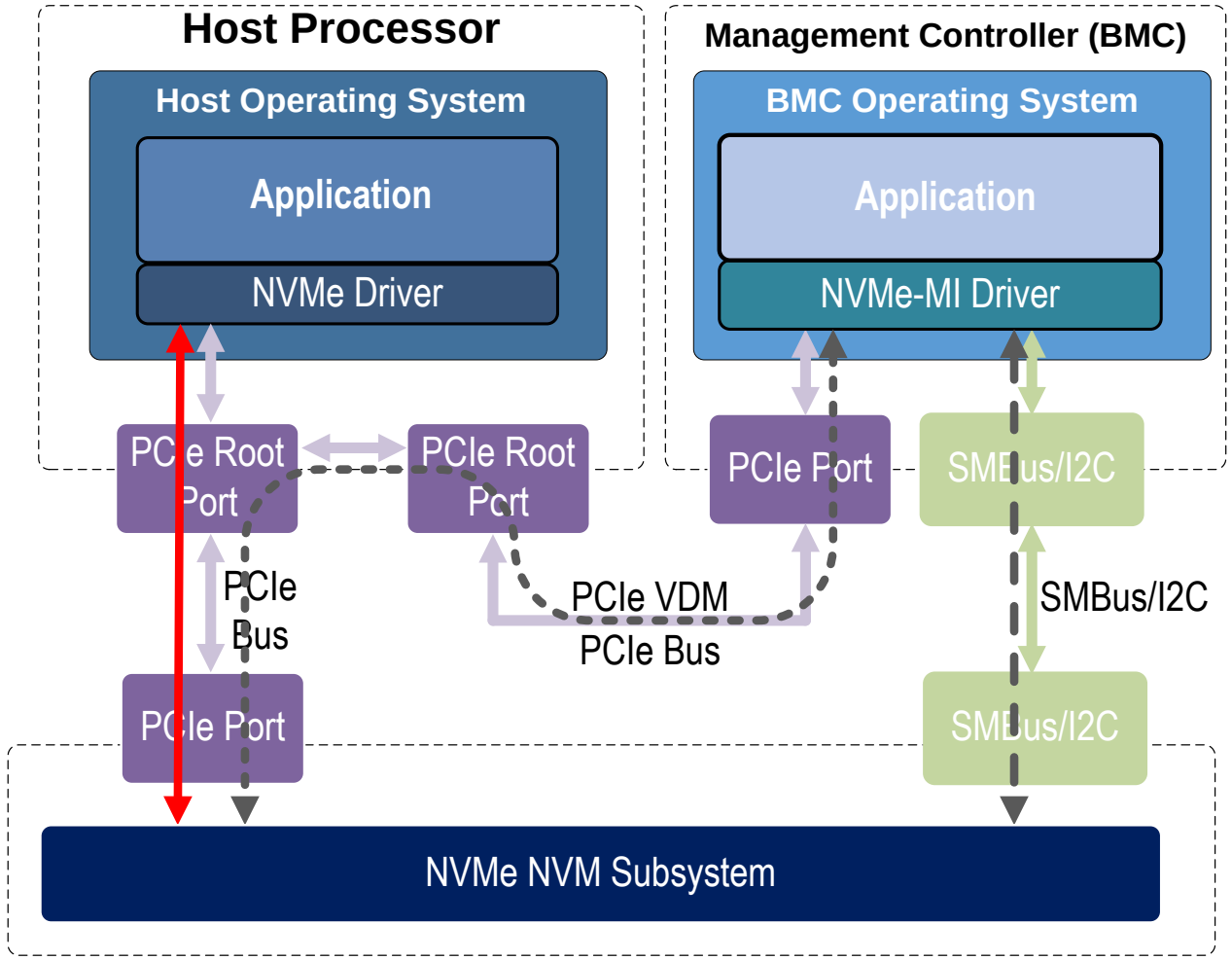
☐**Out-of-Band Management –** Management that operates with hardware resources and components that are ***independent of the host operating system control***

☐**NVMe Out-of-Band Management Interfaces**

- SMBus/I2C
- PCIe Vendor Defined Messages (VDM)



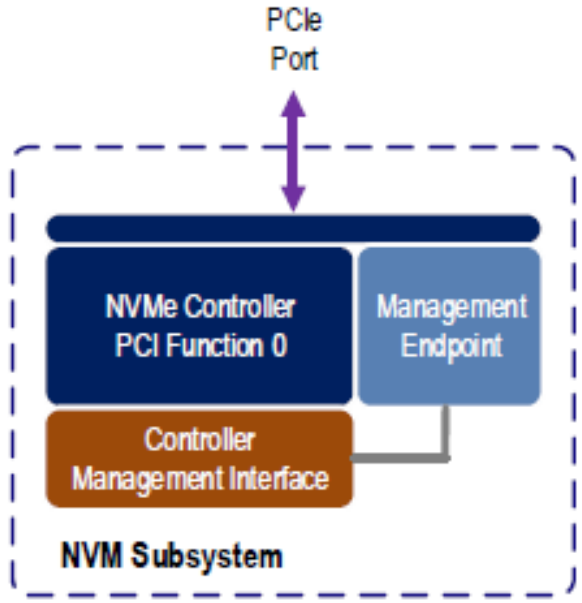
In-Band Management and NVMe-MI™ in 1.1



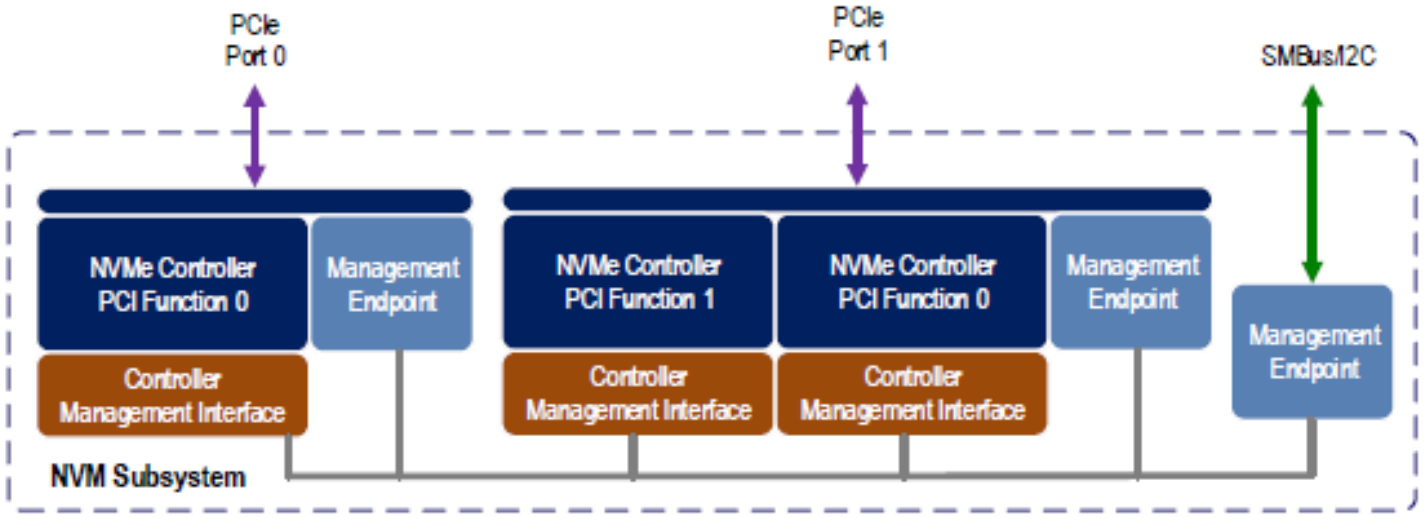
- In-band mechanism allows application to tunnel NVMe-MI commands through NVMe driver
  - Two new NVMe Admin commands
    - NVMe-MI Send
    - NVMe-MI Receive
- Benefits
  - Provides management capabilities not available in-band via NVMe commands
    - Efficient NVM Subsystem health status reporting
    - Ability to manage NVMe at a FRU level Vital Product Data (VPD) access
    - Enclosure management

NVM Subsystem Architectural Model

- **NVM Subsystem -** NVMe-MI extends the definition of an NVM Subsystem defined in the NVM Express specification (e.g., by adding a Management Endpoint, Controller Management Interface, etc.). NVMe Enclosures and NVMe Storages devices that are not Carriers have one or more NVM Subsystems. Carriers have zero or more NVM Subsystems.



NVM Subsystem Associated with Single PCIe Port

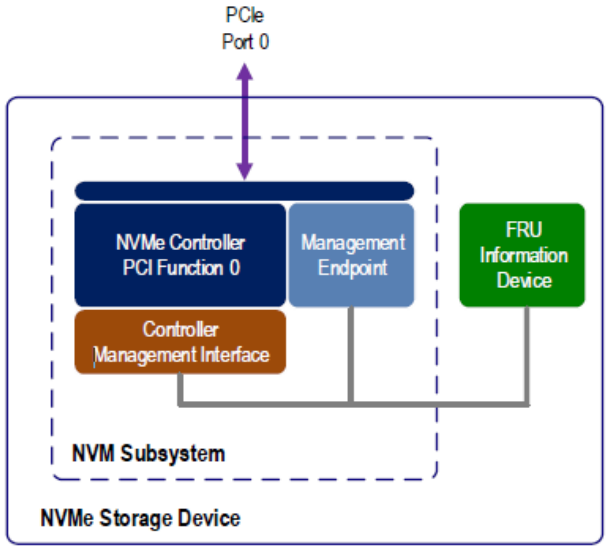


NVM Subsystem with Dual Ported PCIe Ports and an SMBus/I2C Port

NVMe Storage Device Architectural Model

- **NVMe Storage Device** – A logical or physical component, device, or assembly that contains at least one NVM Subsystem or Expansion Connector, at least one Upstream Connector, and at least one FRU Information Device.

Figure 4: Single-Port PCIe SSD



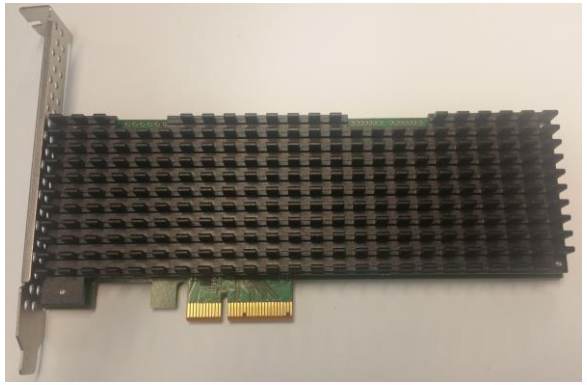
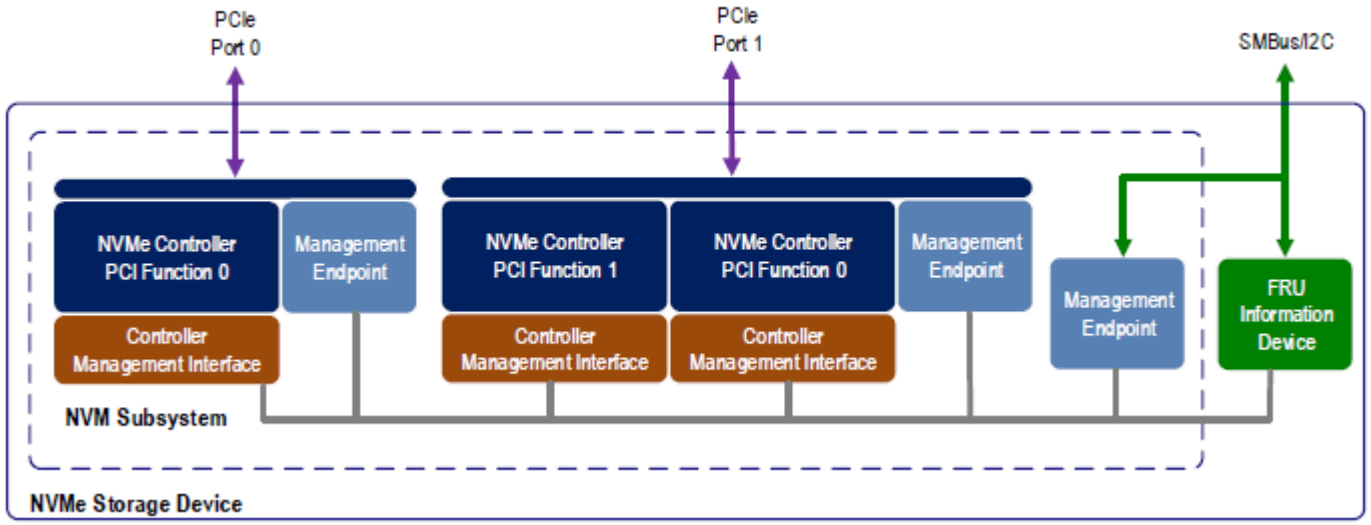


Figure 5: Dual-Port PCIe SSD with SMBus/I2C



NVMe Storage Device Architectural Model

Figure 6: NVMe Storage Device with Expansion Connectors (i.e., a Carrier)

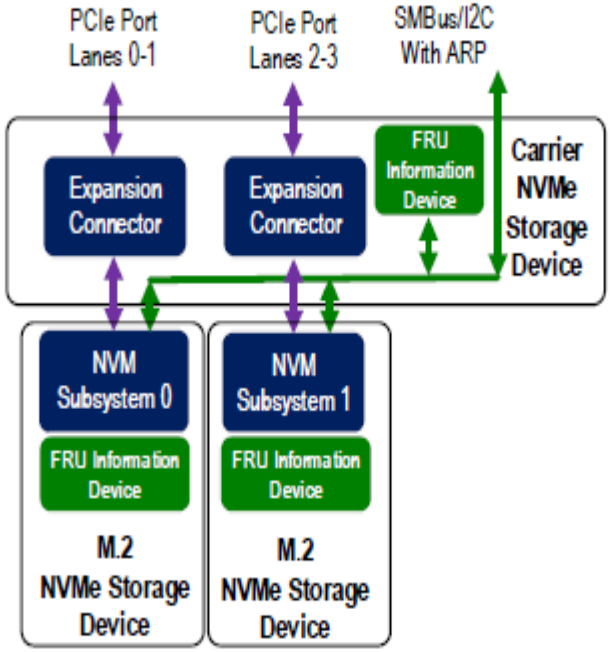
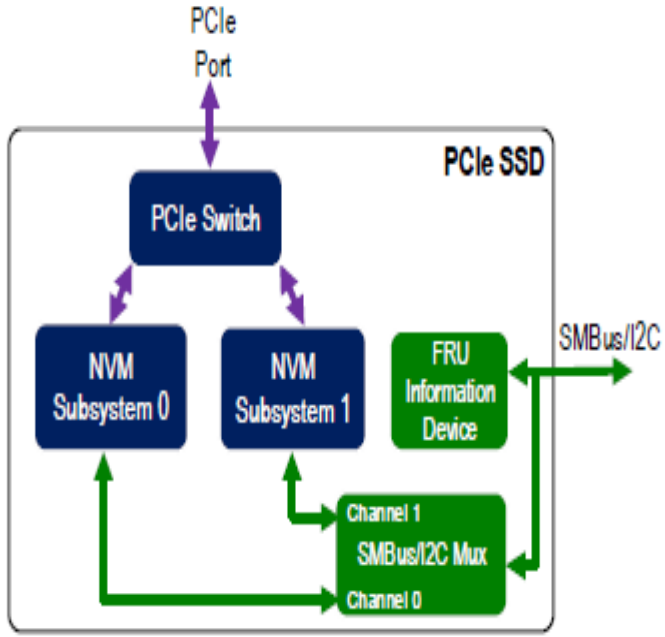
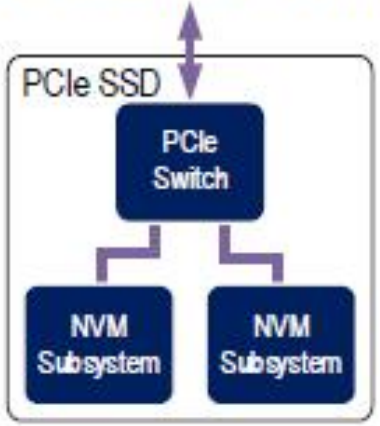


Figure 7: NVMe Storage Device with two NVM Subsystems and an SMBus/I2C Mux

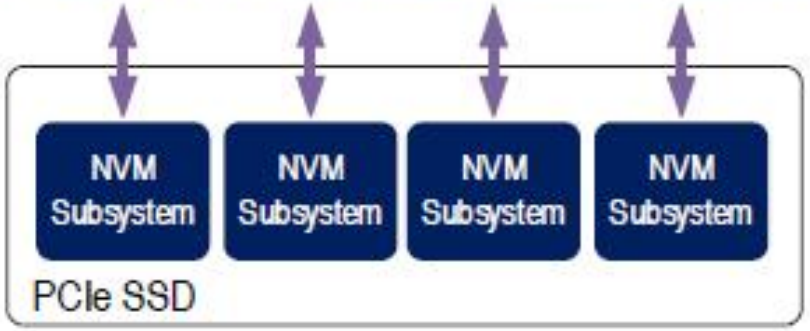


NVMe Storage Device with Multiple NVM Subsystems

M.2 Carrier Board from Amfeltec

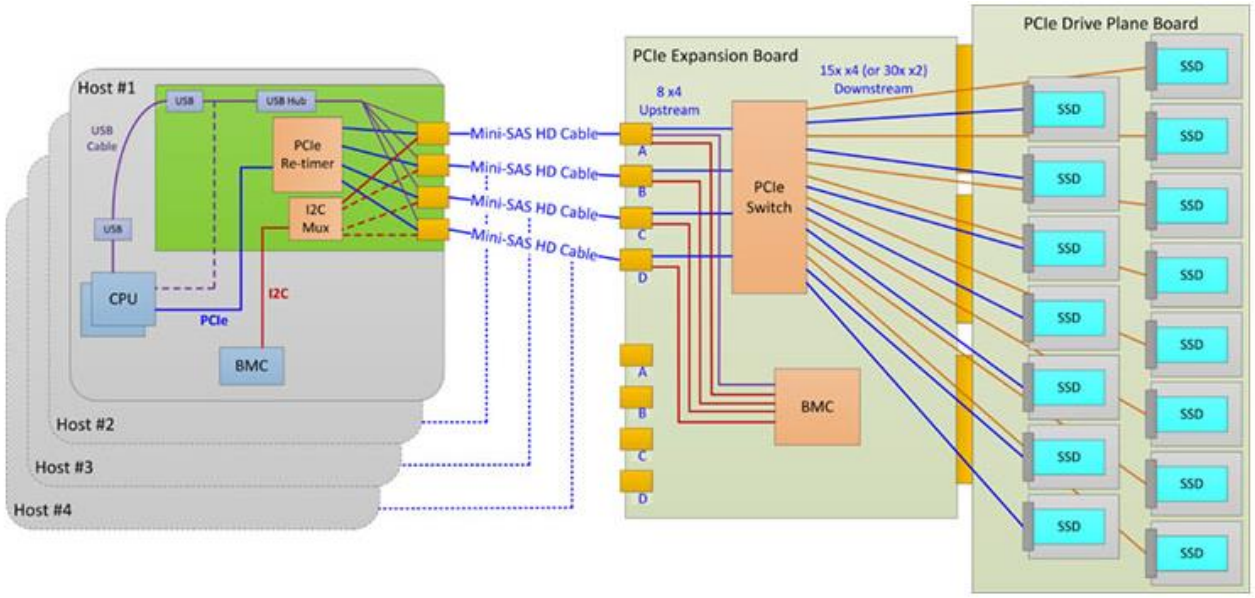


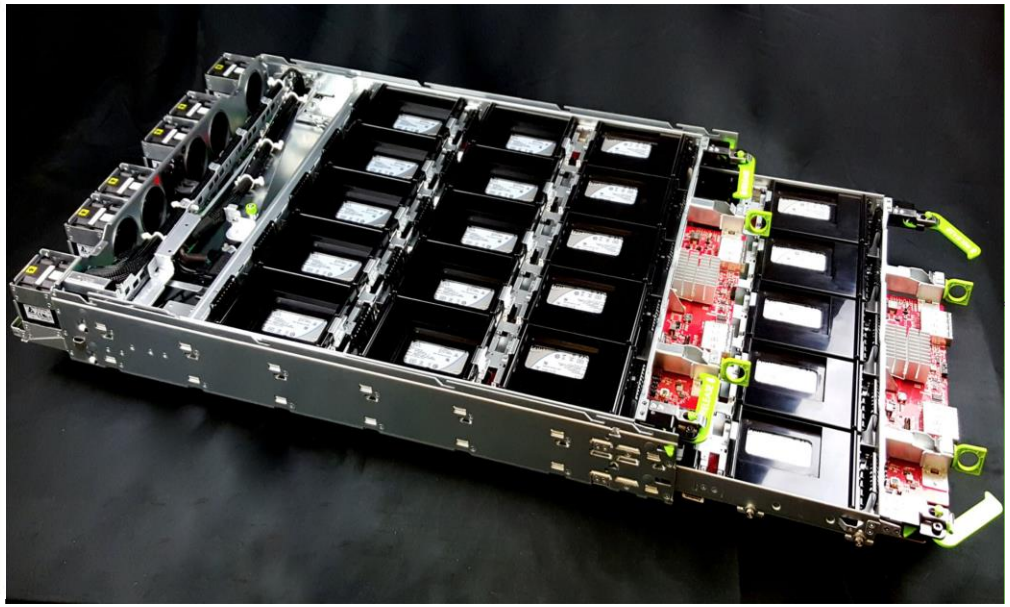
ANA Carrier Board from Facebook



NVMe JBOF (Just a Bunch Of Flash)

Facebook Lightning PCIe NVMe JBOF





- Lightning can support a variety of SSD form factors, including 2.5", M.2, and 3.5" SSDs.
- Lightning will support surprise hot-add and surprise hot-removal of SSDs to make field replacements as simple and transparent as SAS JBODs.
- Lightning will use an ASPEED AST2400 BMC chip running OpenBMC.
- Lightning will support multiple switch configurations, which allows us to support different SSD configurations (for example, 15x x4 SSDs vs. 30x x2 SSDs) or different head node to SSD mappings without changing HW in any way.
- Lightning will be capable of supporting up to four head nodes. By supporting multiple head nodes per tray, we can adjust the compute-to-storage ratio as needed simply by changing the switch configuration.

NVMe Enclosure Architectural Model

- **NVMe Enclosure** – A platform, card, module, box, rack, or set of boxes that may provide power, cooling, mechanical protection and/or external interfaces for zero or more NVMe Storage Device FRUs. An NVMe Enclosure contains one or more NVM Subsystems and one or more Enclosure Services Processes.

Figure 8: Example NVMe Enclosure

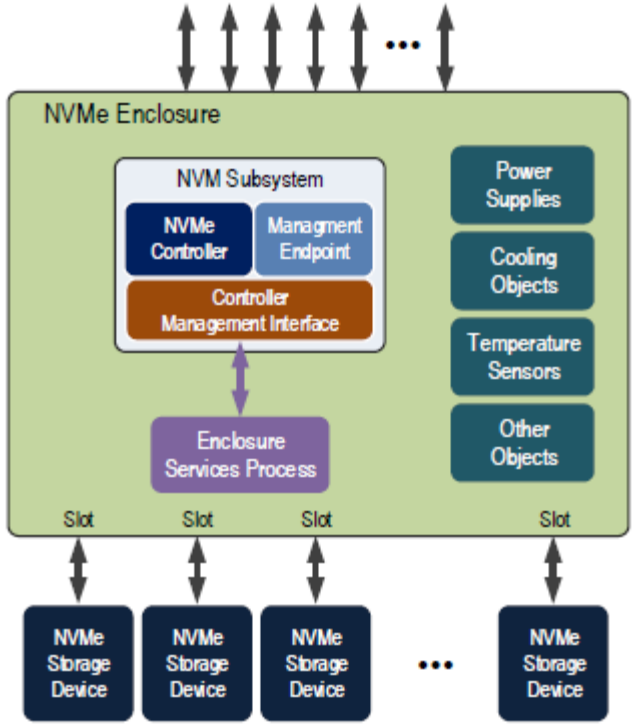
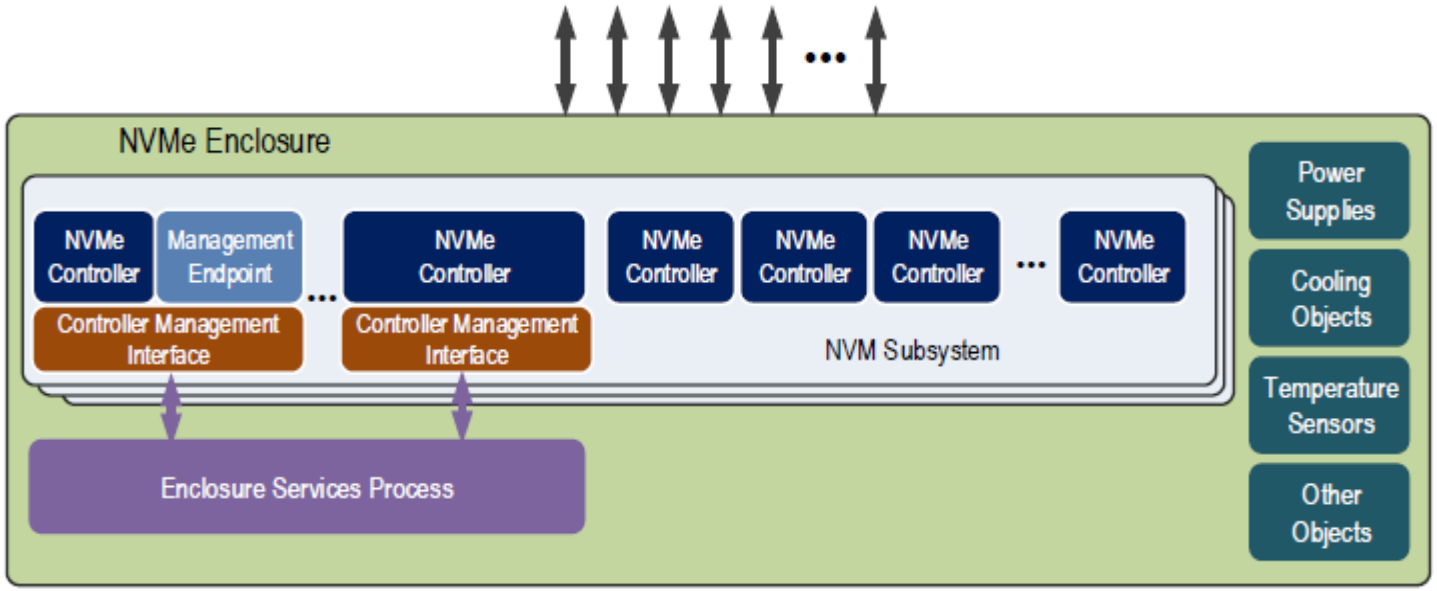


Figure 9: Example NVMe Enclosure with Multiple NVM Subsystems



NVMe Enclosure Architectural Model

- **Enclosure Services Process** – A process that implements Enclosure services for an NVMe Enclosure that supports Enclosure Management.

Figure 10: Example NVMe Enclosure with Multiple Enclosure Services Processes

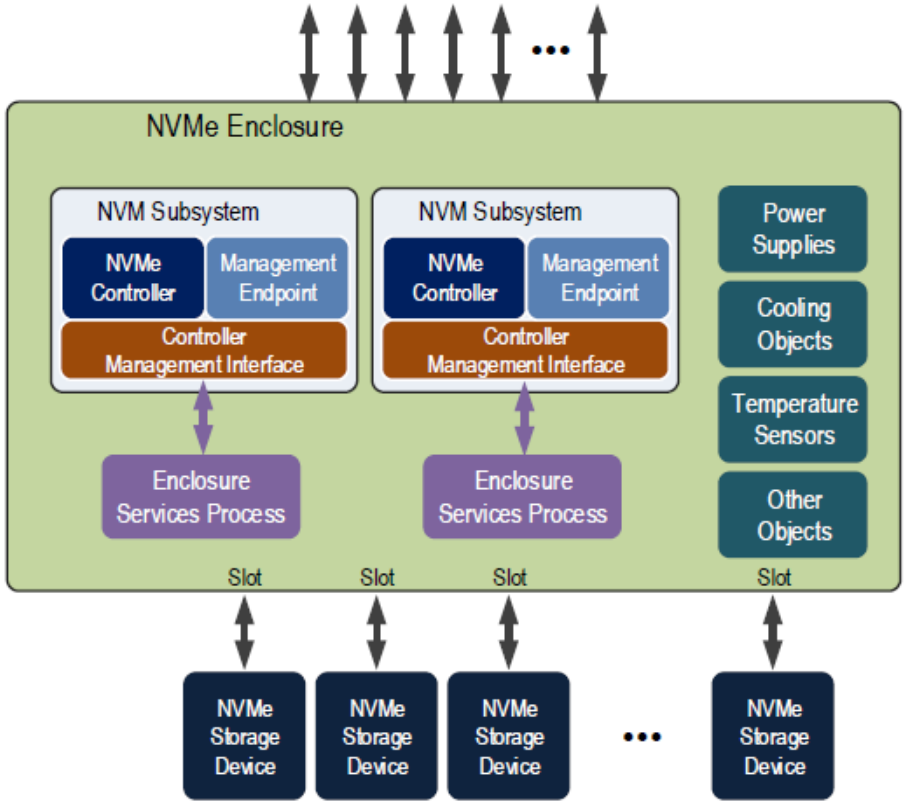
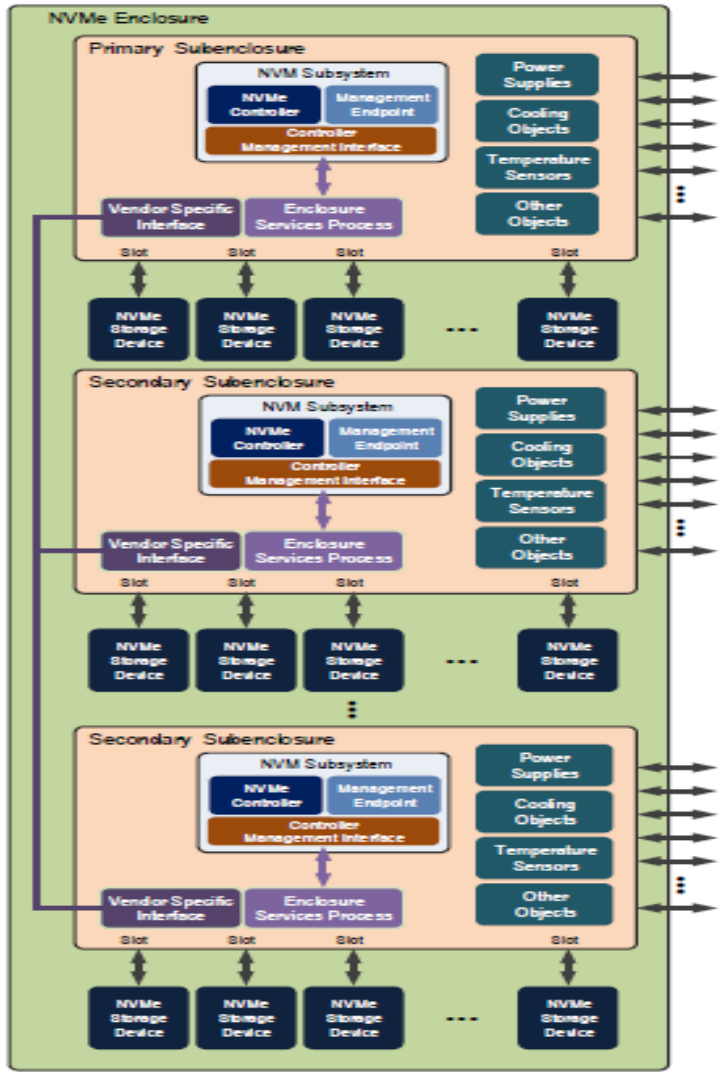


Figure 11: Example NVMe Enclosure with Subenclosures

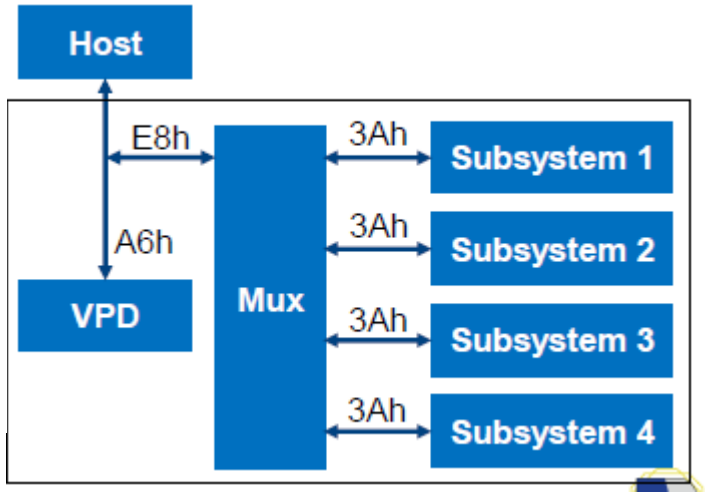


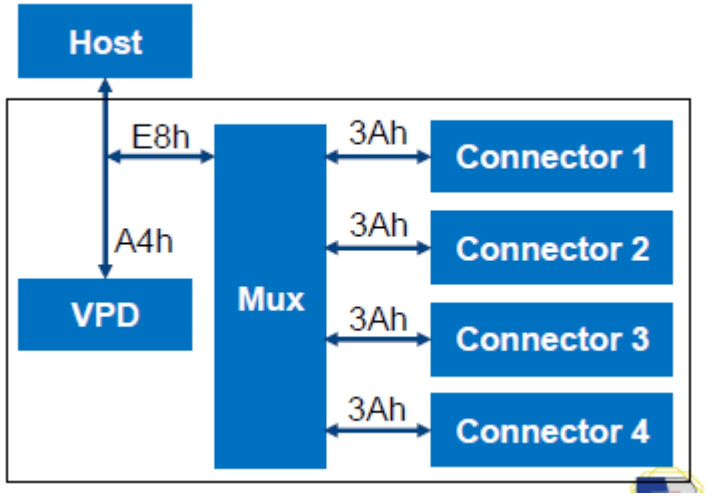
SMBus Topology for NVMe-MI

Figure 15: SMBus/I2C Elements and Requirements

SMBus/I2C Element	Default SMBus/I2C Address		SMBus ARP Support	Required Element Presence
	Hex Format	Binary Format [1]		
FRU Information Device	A6h	1010_011xb	Optional	Required on an NVMe Storage Device with *no* Expansion Connectors. Undefined on NVMe Enclosures.
FRU Information Device	A4h	1010_010xb	Optional	Required on Carriers (i.e., an NVMe Storage Device with *one or more* Expansion Connectors). Undefined on NVMe Enclosures.
SMBus/I2C Management Endpoint	3Ah	0011_101xb	Optional	Required if an NVMe Storage Device or NVMe Enclosure an SMBus/I2C Management Endpoint.
SMBus/I2C Mux	E8h	1110_100xb	Optional	For NVMe Storage Devices, required if there is more than one SMBus/I2C element on any SMBus/I2C channel with the same SMBus/I2C address that does not support ARP. Undefined on NVMe Enclosures.
Basic Management Command [2]	D4h	1101_010xb	Optional	For NVMe Storage Devices, not recommended for new designs. Undefined on NVMe Enclosures.

NOTES:
1. The x represents the SMBus/I2C read/write bit.
2. The NVMe Basic Management Command is defined in Appendix A as an informative technical note.





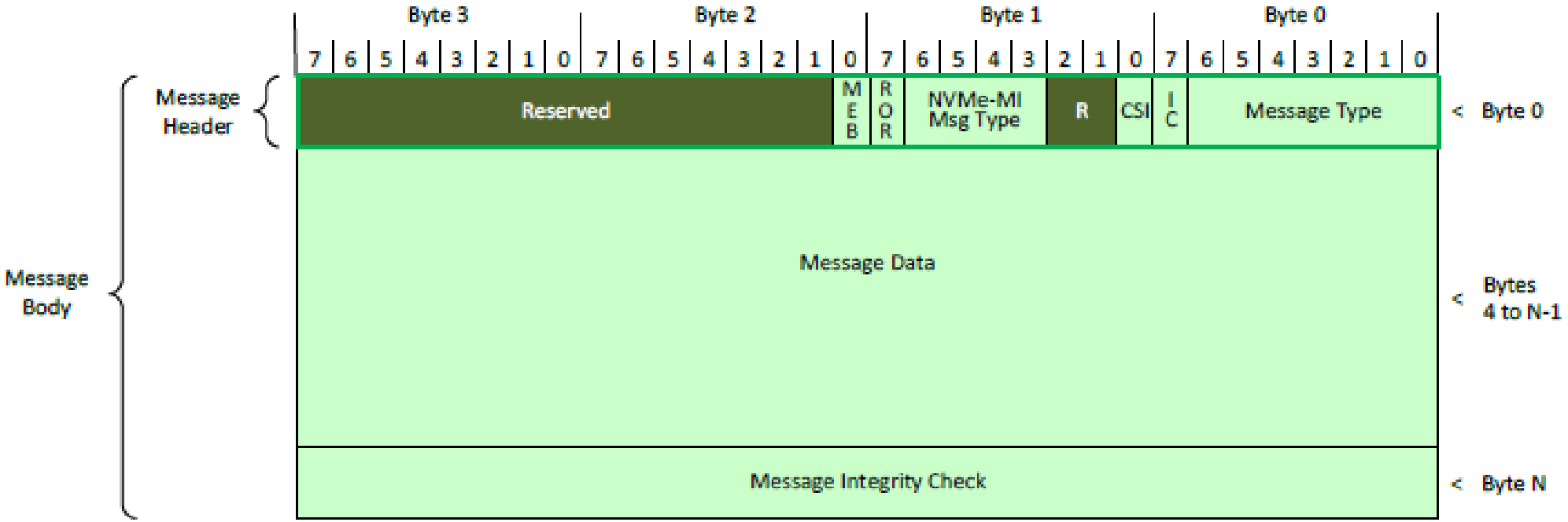
NVM Express™ Management Interface

Message Transport

Message Transport

- **NVMe-MI Messages**– In the out-of-band mechanism, an NVMe-MI Message consists of the payload of one or more MCTP packets. The maximum sized NVMe-MI Message is 4,224 bytes (i.e., 4 KiB + 128 bytes). In the in-band tunneling mechanism, NVMe-MI Messages are not split into MCTP packets and the maximum NVMe-MI message size is equal to the Maximum Data Transfer Size.

Figure 17: NVMe-MI Message



Message Transport

Figure 18: NVMe-MI Message Fields

Byte	Description
0	**MCTP Data (MCTPD):** This field contains the Message Type and Integrity Check fields as defined by the MCTP Base Specification.

Bits	Description
7	**Integrity Check (IC):** This bit is defined by the MCTP Base Specification and indicates whether the MCTP message is covered by an overall MCTP Message Integrity Check. All NVMe-MI Messages in the out-of-band mechanism shall be protected by a CRC and thus this bit shall be set to '1' in all out-of-band NVMe-MI Messages. All NVMe-MI Messages in the in-band tunneling mechanism shall not be protected by a CRC and thus this bit shall be cleared to '0' in all in-band NVMe-MI Messages.
6:0	**Message Type (MT):** This field is defined by the MCTP Base Specification for the message type. This field shall be set to 4h in all NVMe-MI Messages. Refer to the MCTP IDs and Codes specification.

Byte	Description
1	**NVMe-MI Message Parameters (NMP):** This field contains parameters applicable to the NVMe-MI Message.

Bits	Description
7	**Request or Response (ROR):** This bit indicates whether the message is a Request Message or Response Message. This bit is cleared to '0' for Request Messages. This bit is set to '1' for Response Messages.
6:3	**NVMe-MI Message Type (NMIMT):** This field specifies the NVMe-MI Message Type. Refer to the sections referenced in the table below for details about each NVMe-MI Message Type and whether they apply to the out-of-band mechanism, the in-band tunneling mechanism, or both.
2:1	Reserved
0	**Command Slot Identifier (CSI):** This bit indicates the Command Slot with which the NVMe-MI Message is associated. For Request Messages this bit indicates the Command Slot with which the Request Message is associated. For Response Messages, this bit indicates the Command Slot associated with the Request Message with which the Response Message is associated. This bit is only applicable to NVMe-MI Messages in the out-of-band mechanism. This bit is reserved for NVMe-MI Messages in the in-band tunneling mechanism.

Value	Description	Reference Section
0h	Control Primitive	4.2.1
1h	NVMe-MI Command	5
2h	NVMe Admin Command	6
3h	Reserved	-
4h	PCIe Command	7
5h to Fh	Reserved	-

Value	Description
0b	Command Slot 0
1b	Command Slot 1

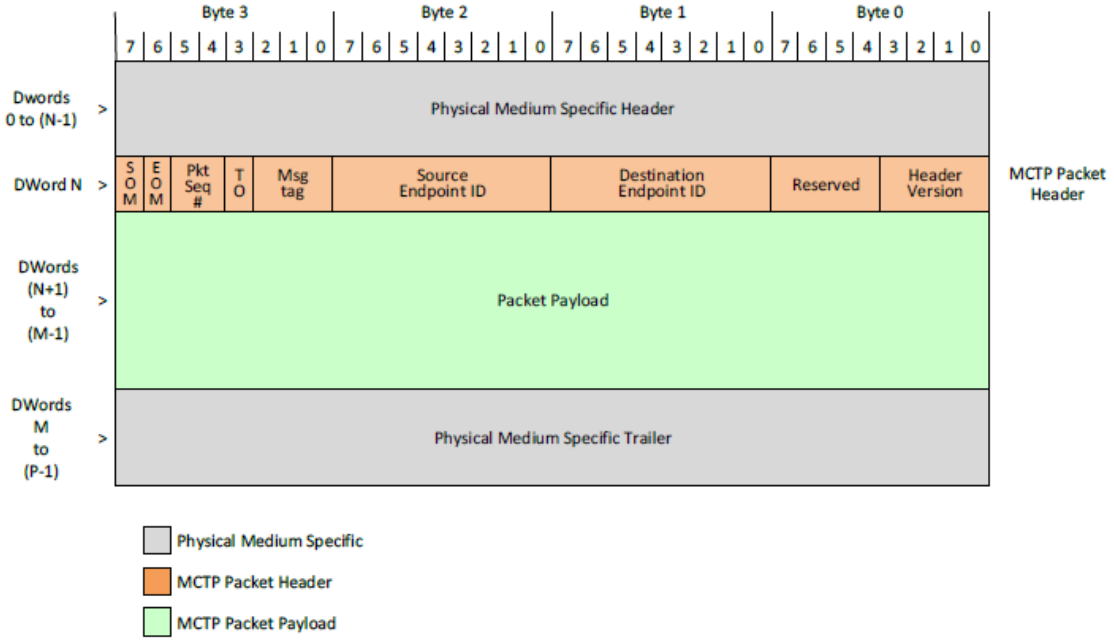
Bits	Description
7:1	Reserved
0	**Management Endpoint Buffer (MEB):** This bit indicates whether the Message Data is contained in the Message Data field of this NVMe-MI Message or in the Management Endpoint Buffer. Refer to section 3.1.

Value	Description
0b	The Message Data is contained in the Message Data of this NVMe-MI Message.
1b	The Message Data is contained in the Management Endpoint Buffer.

Byte	Description
2	
3	Reserved
N-1:4	**Message Data (DATA):** This field contains the NVMe-MI Message payload. The format of this field depends on the NVMe-MI Message Type.
N+3:N	**Message Integrity Check (MIC):** If the Integrity Check (IC) bit is set to '1', then this field contains a CRC computed over the contents of the NVMe-MI Message. Refer to section 3.1.1.1. If the IC bit is cleared to '0', then this field is not included in the NVMe-MI Message.

Message Transport

Figure 20: MCTP Packet Format



Field Name	Field Size	Description
RSVD	4 bits	(Reserved) Reserved for future definition by the MCTP base specification.
Hdr version	4 bits	(Header version) Identifies the format, physical framing, and data integrity mechanism used to transfer the MCTP common fields in messages on a given physical medium. The value is defined in the specifications for the particular medium. Note: The value in this field can vary between different transport bindings.
Destination endpoint ID	8 bits	The EID for the endpoint to receive the MCTP packet. A few EID values are reserved for specific routing. See Table 2 – Special endpoint IDs.
Source endpoint ID	8 bits	The EID of the originator of the MCTP packet. See Table 2 – Special endpoint IDs.
SOM	1 bit	(Start Of Message) Set to 1b if this packet is the first packet of a message.
EOM	1 bit	(End Of Message) Set to 1b if this packet is the last packet of a message.
Pkt Seq #	2 bits	(Packet sequence number) For messages that span multiple packets, the packet sequence number increments modulo 4 on each successive packet. This allows the receiver to detect up to three successive missing packets between the start and end of a message. Though the packet sequence number can be any value (0-3) if the SOM bit is set, it is recommended that it is an increment modulo 4 from the prior packet with an EOM bit set. After the SOM packet, the packet sequence number shall increment modulo 4 for each subsequent packet belonging to a given message up through the packet containing the EOM flag.
TO	1 bit	The TO (Tag Owner) bit identifies whether the message tag was originated by the endpoint that is the source of the message or by the endpoint that is the destination of the message. The Message Tag field is generated and tracked independently for each value of the Tag Owner bit. MCTP message types may overlay this bit with additional meaning, for example using it to differentiate between "request" messages and "response" messages. Set to 1b to indicate that the source of the message originated the message tag.
Msg tag	3 bits	(Message tag) Field that, along with the Source Endpoint IDs and the Tag Owner (TO) field, identifies a unique message at the MCTP transport level. Whether other elements, such as portions of the MCTP Message Data field, are also used for uniquely identifying instances or tracking retries of a message is dependent on the message type. A source endpoint is allowed to interleave packets from multiple messages to the same destination endpoint concurrently, provided that each of the messages has a unique message tag. When request/response message exchange is used and the Tag Owner (TO) bit is set to 1 in the request, a responder should return the same Message Tag with the Message Tag Owner bit cleared to 0 in the corresponding response Message. For messages that are split up into multiple packets, the Tag Owner (TO) and Message Tag bits remain the same for all packets from the SOM through the EOM.

Message Transport

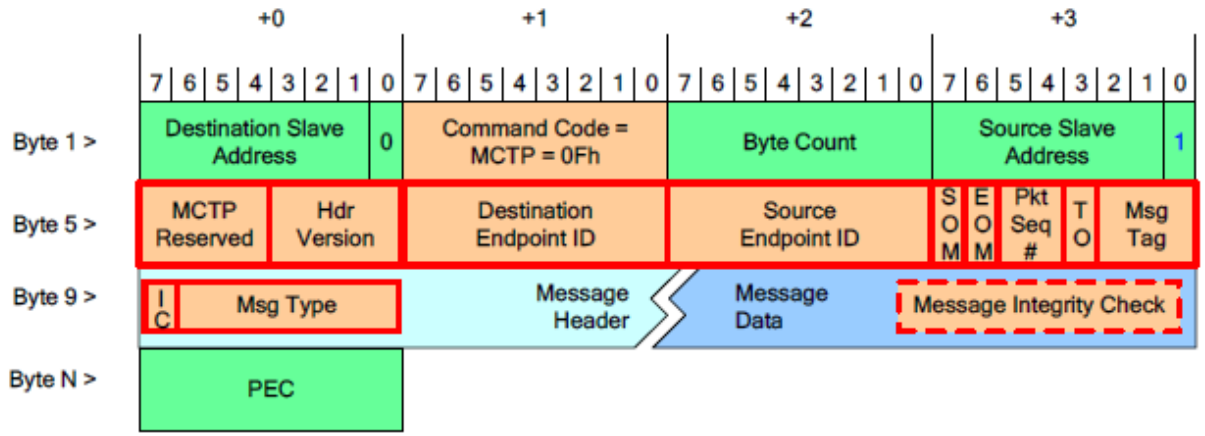


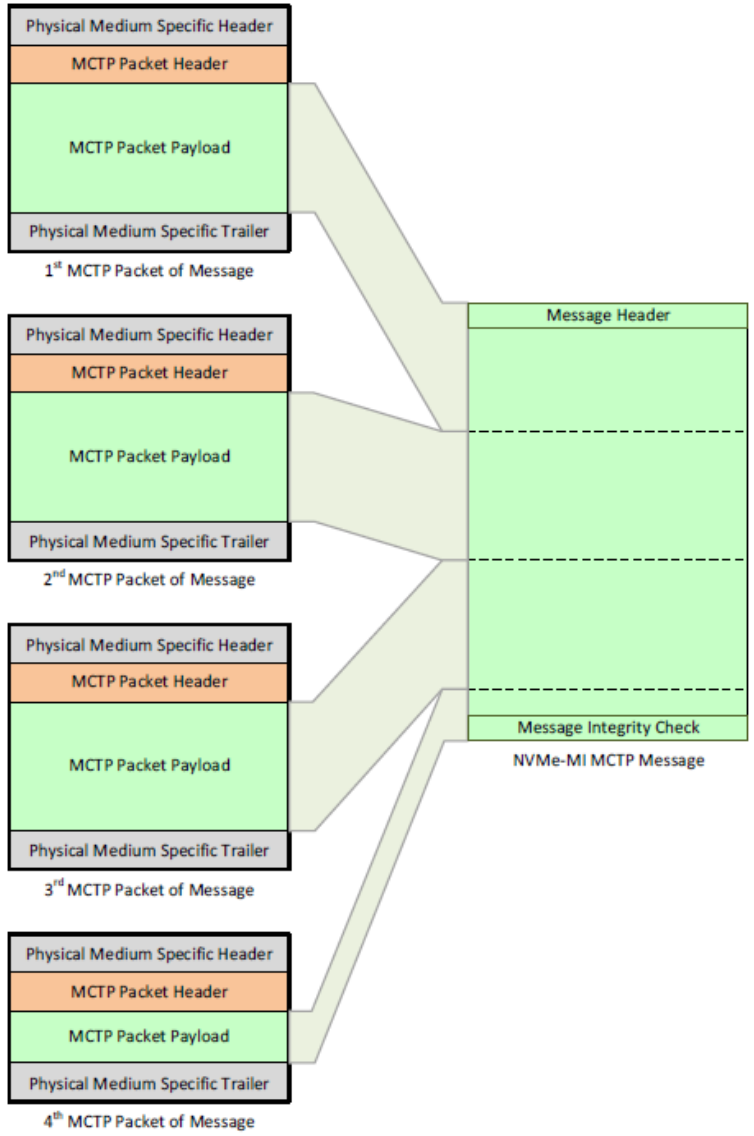
Figure 1 – MCTP over SMBus/I²C packet format

Table 1 – Packet header field descriptions

Byte	Block Write Field(s)	Description
1	Slave Address Wr	[7:1] SMBus Destination Slave Address: The slave address of the target device for the local SMBus link [0]: SMBus R/W# bit: Shall be set to 0b as all MCTP messages use SMBus write transactions.
2	Command Code	Command Code: SMBus Command Code All MCTP over SMBus messages use a command code of 0x0F.
3	Byte Count	Byte Count: Byte count for the SMBus Block Write protocol transaction that is carrying the MCTP packet content. This value is the count of bytes that follow the Byte Count field up to, but not including, the PEC byte. For example, if the MCTP packet payload length (starting with byte 9) is 64 bytes, the value in the Byte Count field would be 69. (The count of 69 accounts for 64 bytes of MCTP packet payload plus the five bytes [bytes 4 through 8, inclusive] that comprise the bytes of the SMBus-specific header and MCTP header that follow the Byte Count field.)
4	Data Byte 1	SMBus Source Slave address [7:1] : For the local SMBus link, the slave address of the source device. [0]: This bit shall be set to 1b. The value enables MCTP to be differentiated from IPMI over SMBus and IPMB (IPMI over I²C) protocols.

Message Transport

Figure 22: NVMe-MI Message Spanning Multiple MCTP Packets

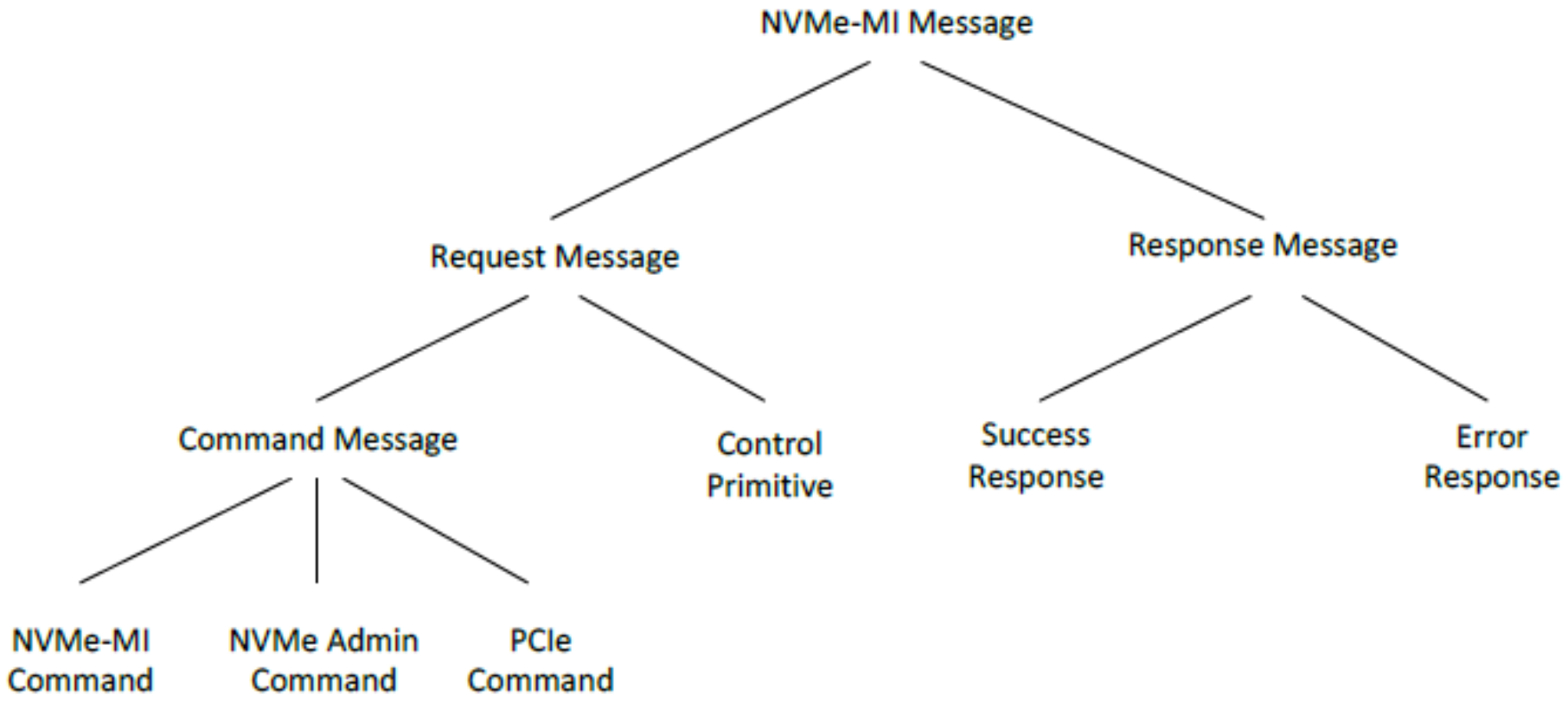


NVM Express™ Management Interface

Message Servicing Model

Message Servicing Model

Figure 23: NVMe-MI Message Taxonomy



Message Servicing Model

Figure 24: Response Message Format

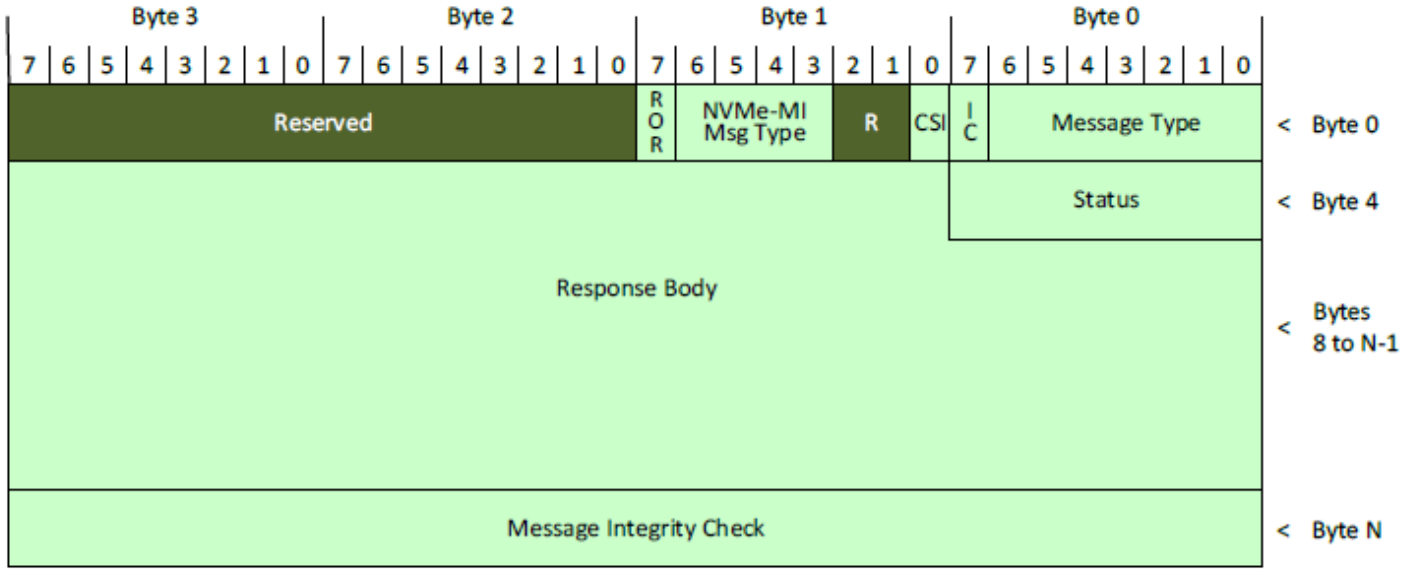
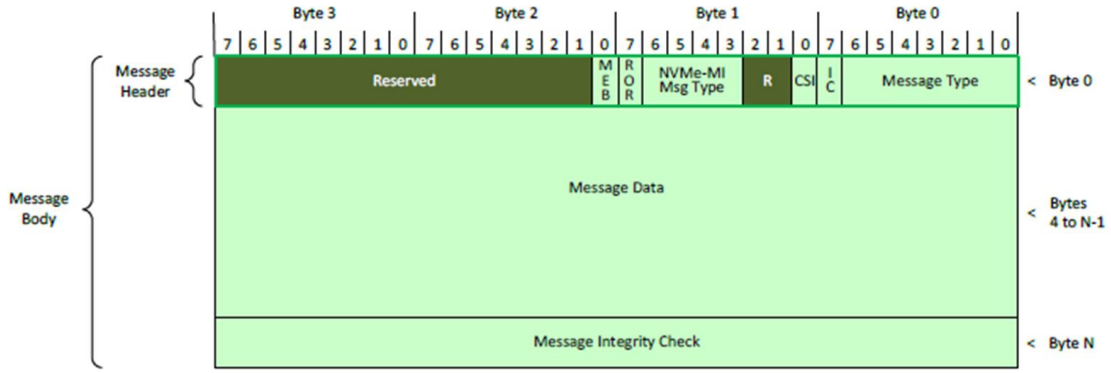


Figure 17: NVMe-MI Message



Message Servicing Model

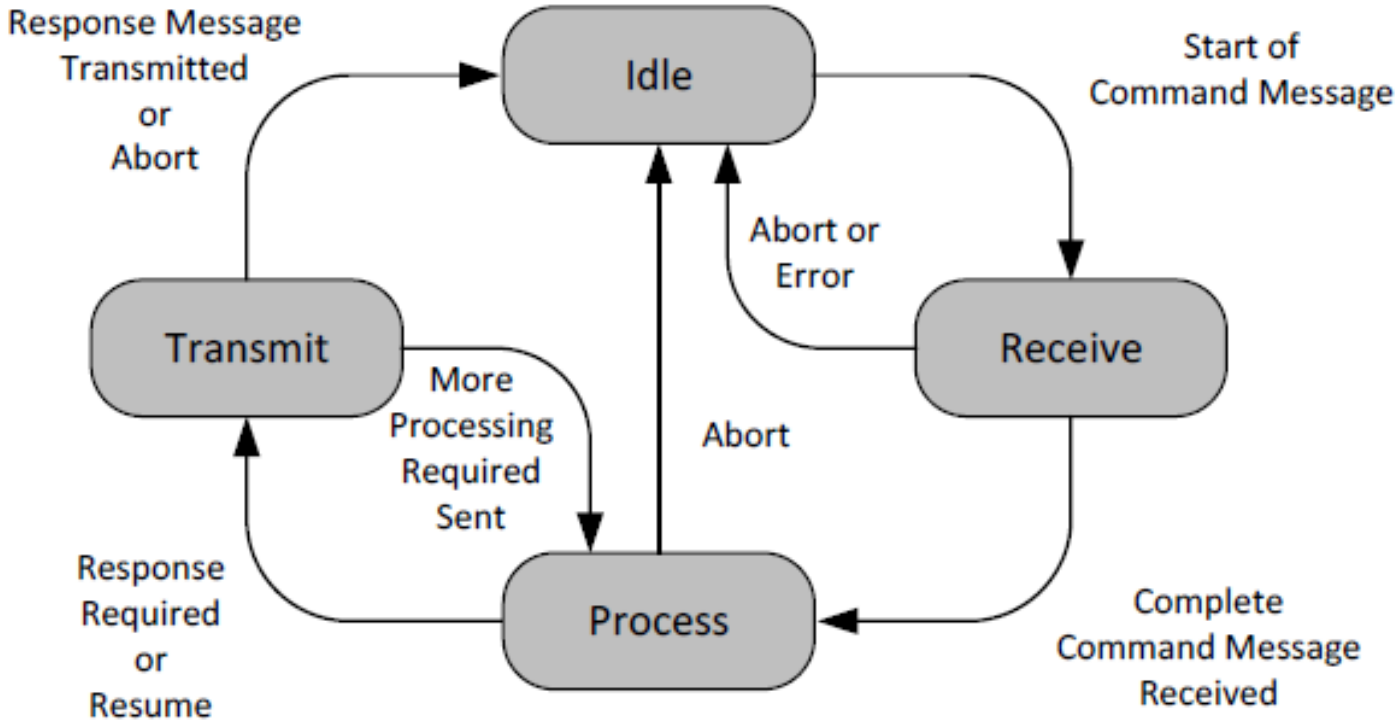
Figure 26: Response Message Status Values

Value	Description	Error Response Format Section
00h	**Success**: The command completed successfully.	4.1.2.1
01h	**More Processing Required**: The Command Message is in progress and requires more time to complete processing. When this Response Message Status is used in a Response Message, a subsequent Response Message contains the result of the Command Message. This Response Message Status shall not be sent more than once per Request Message.	4.1.2.1
02h	**Internal Error**: The Request Message could not be executed due to a vendor specific internal error.	4.1.2.1
03h	**Invalid Command Opcode**: The associated command opcode field is not valid. Invalid opcodes include reserved and optional opcodes that are not implemented.	4.1.2.1
04h	**Invalid Parameter**: Invalid parameter field value. Request Messages received with reserved values in defined fields shall be completed with an Invalid Parameter Error Response. Request Messages received with reserved or unimplemented values in defined fields shall be completed with an Invalid Parameter Error Response. Other error conditions that result in Invalid Parameter Error Response are noted elsewhere in this specification.	4.1.2.2
05h	**Invalid Command Size**: The Message Body of the Command Message was larger or smaller than that expected due to a reason other than too much or too little Request Data (e.g., the Command Message did not contain all the required parameters or no Request Data was expected but the Request Data is larger than that needed to contain the required parameters). The expected Message Body size is determined by the NVMe-MI Message Type and opcode assuming no other errors are detected (e.g., Invalid Command Opcode or Invalid Parameter).	4.1.2.1
06h	**Invalid Command Input Data Size**: The Command Message requires Request Data and contains too much or too little Request Data.	4.1.2.1
07h	**Access Denied**: A Request Message was prohibited from being processed due to a vendor specific protection mechanism.	4.1.2.1
08h to 1Fh	Reserved	-
20h	**VPD Updates Exceeded**: More updates to the VPD are attempted than allowed.	4.1.2.1
21h	**PCIe Inaccessible**: The PCIe functionality is not available at this time.	4.1.2.1
22h	**Management Endpoint Buffer Cleared Due to Sanitize**: An attempt was made access data in the Management Endpoint Buffer that was zeroed due to a sanitize operation.	4.1.2.1
23h	**Enclosure Services Failure**: The Enclosure Services Process has failed in an unknown manner.	4.1.2.1
24h	**Enclosure Services Transfer Failure**: Communication with the Enclosure Services Process has failed.	4.1.2.1
25h	**Enclosure Failure**: An unrecoverable enclosure failure has been detected by the Enclosure Services Process.	4.1.2.1
26h	**Enclosure Services Transfer Refused**: The NVM Subsystem or Enclosure Services Process indicated an error or an invalid format in communication.	4.1.2.1
27h	**Unsupported Enclosure Function**: An SES Send command has been attempted to a simple Subenclosure.	4.1.2.1
28h	**Enclosure Services Unavailable**: The NVM Subsystem or Enclosure Services Process has encountered an error but may become available again.	4.1.2.1
29h	**Enclosure Degraded**: A noncritical failure has been detected by the Enclosure Services Process.	4.1.2.1
2Ah	**Sanitize In Progress**: The requested command is prohibited while a sanitize operation is in progress. Refer to section 9.1.	4.1.2.1
2Bh to DFh	Reserved	-
E0h to FFh	Vendor Specific	Vendor Specific

Message Servicing Model

- **Out-of-Band Message Servicing Model**– The out-of-band mechanism utilizes a request and response servicing model. A Management Controller sends a Request Message to a Management Endpoint, the Management Endpoint processes the Request Message, and when processing has completed, sends a Response Message back the Management Controller.

Figure 30: Command Servicing State Diagram



Message Servicing Model

- **Control Primitives**– Control Primitives are Request Messages sent from a Management Controller to a Management Endpoint to affect the servicing of a previously issued Command Message or get the state of a Command Slot and Management Endpoint.

Figure 31: Control Primitive Request Message Format

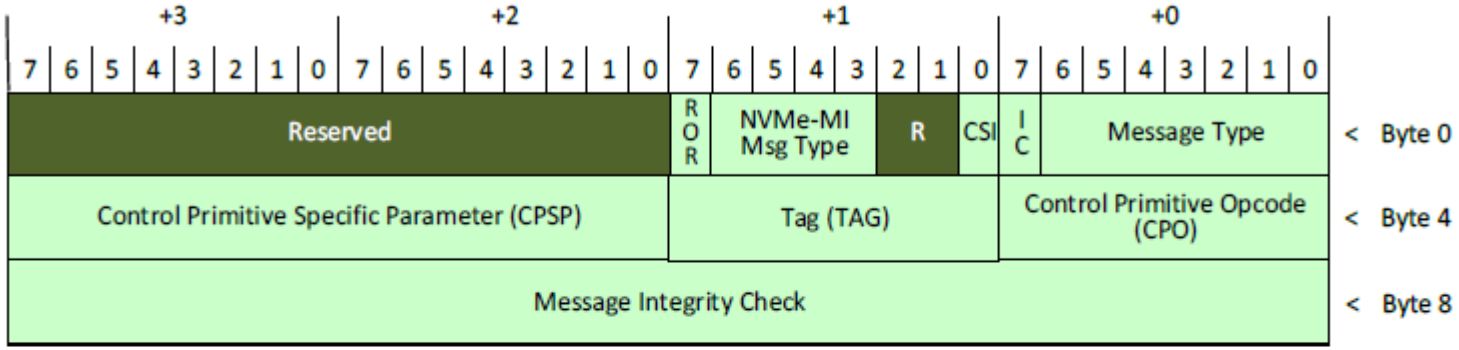


Figure 32: Control Primitive Fields

Byte	Description
03:00	**NVMe-MI Message Header (NMH):** Refer to section 3.1.
04	**Control Primitive Opcode (CPO):** This field specifies the opcode of the Control Primitive to be executed. Refer to Figure 33.
05	**Tag (TAG):** This field contains an opaque value that is sent from the Management Controller in the Control Primitive and returned by the Management Endpoint in to the associated Response Message. A Management Controller may use any value in this field.
07:06	**Control Primitive Specific Parameter (CPSP):** This field is used to pass Control Primitive specific parameter information.
11:08	**Message Integrity Check (MIC):** Refer to section 3.1.

Control Primitive	Description
Pause	Suspend transmission
Resume	Resume paused transmission
Abort	Reinitialize command slot
Get State	Retrieve state (e.g., errors) associated with a command slot
Replay	Retransmit response message for last command message processed in a command slot

Message Servicing Model

- **In-Band Tunneling Message Servicing Model**– The in-band tunneling mechanism in this specification utilizes two NVMe Admin Commands (NVMe-MI Send and NVMe-MI Receive).
  - The NVMe-MI Send command is used to tunnel an NVMe-MI Command from host software to an NVMe Controller that transfers data from the host to the NVMe Controller (similar to a write operation) or to instruct the Management Endpoint to perform an action (e.g., to reset the NVM Subsystem using the Reset command).

Figure 42: NVMe-MI Send Command Request Message to NVMe Admin Command SQE Mapping Diagram

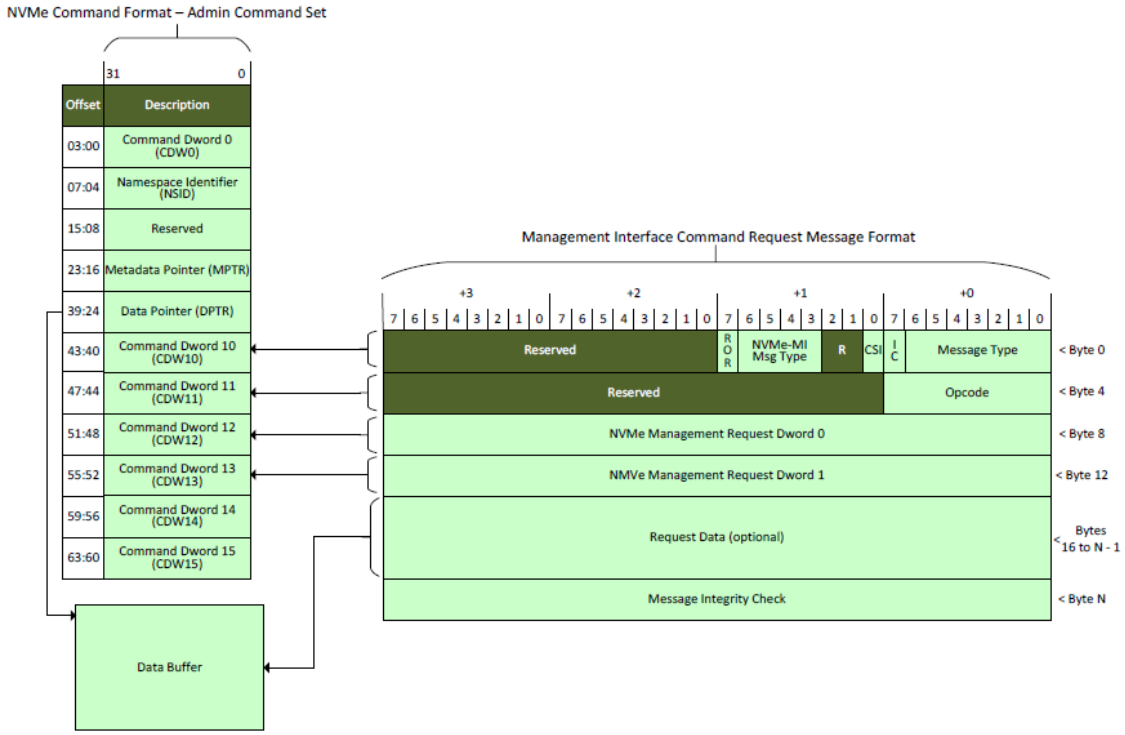
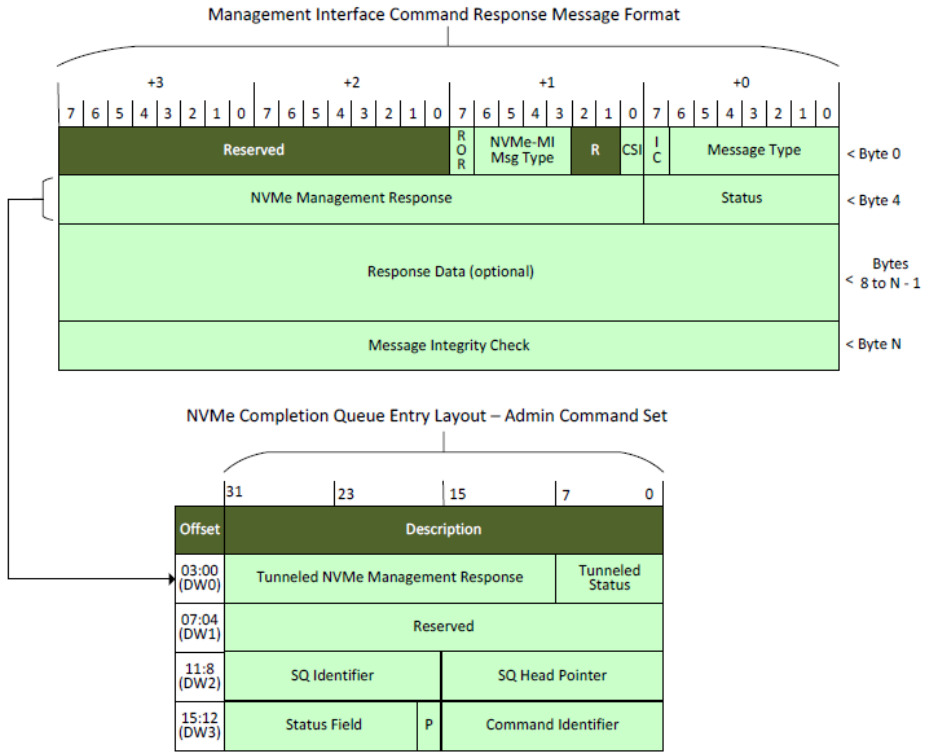


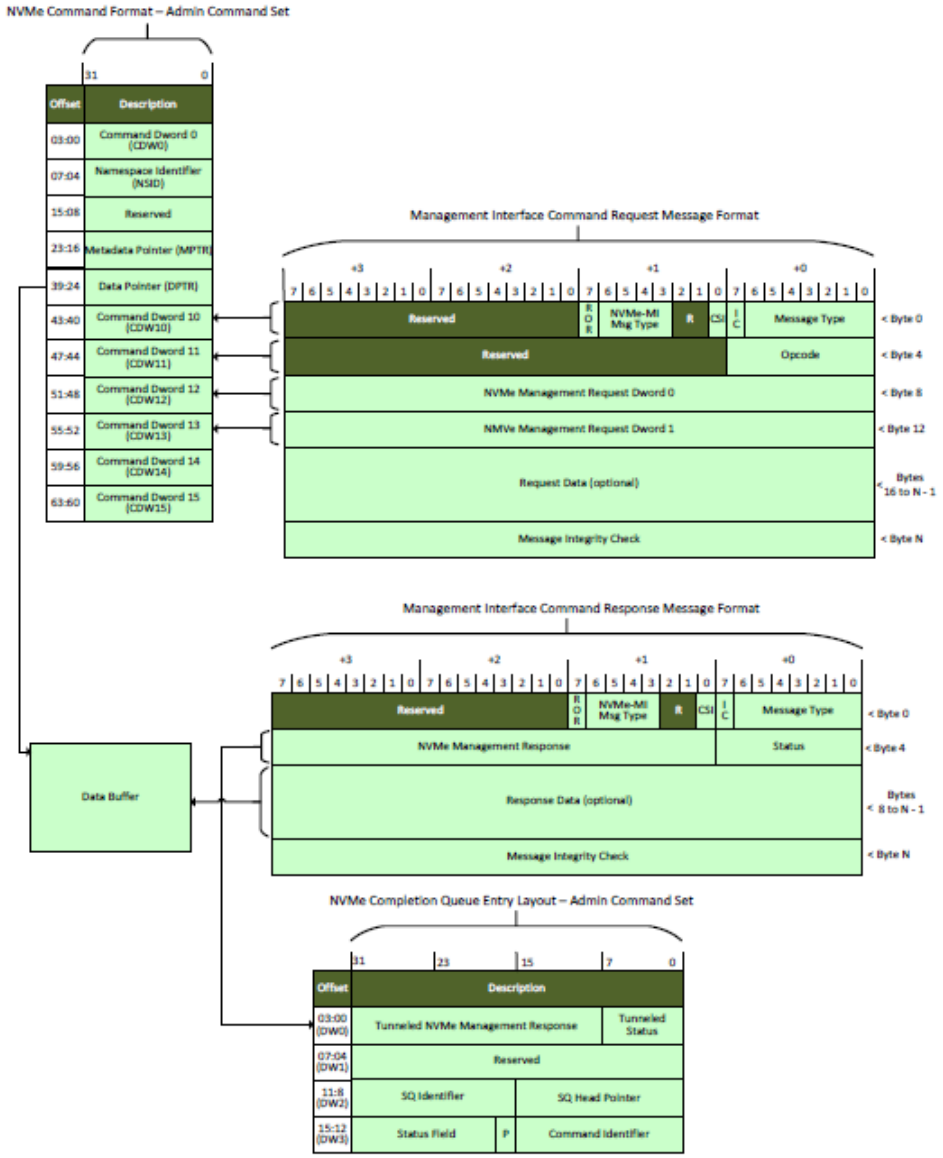
Figure 44: NVMe-MI Send Command Response Message to NVMe Admin Command CQE Mapping Diagram



Message Servicing Model

- The NVMe-MI Receive command is used to tunnel an NVMe-MI Command from a host to an NVMe Controller that transfers data from the NVMe Controller to the host (similar to a read operation).

Figure 48: NVMe-MI Receive Command Request/Response Message to NVMe Admin Command SQE/CQE Mapping Diagram



NVM Express™ Management Interface

Command Set

Management Interface Command Set

Figure 53: NVMe-MI Command Request Message Format

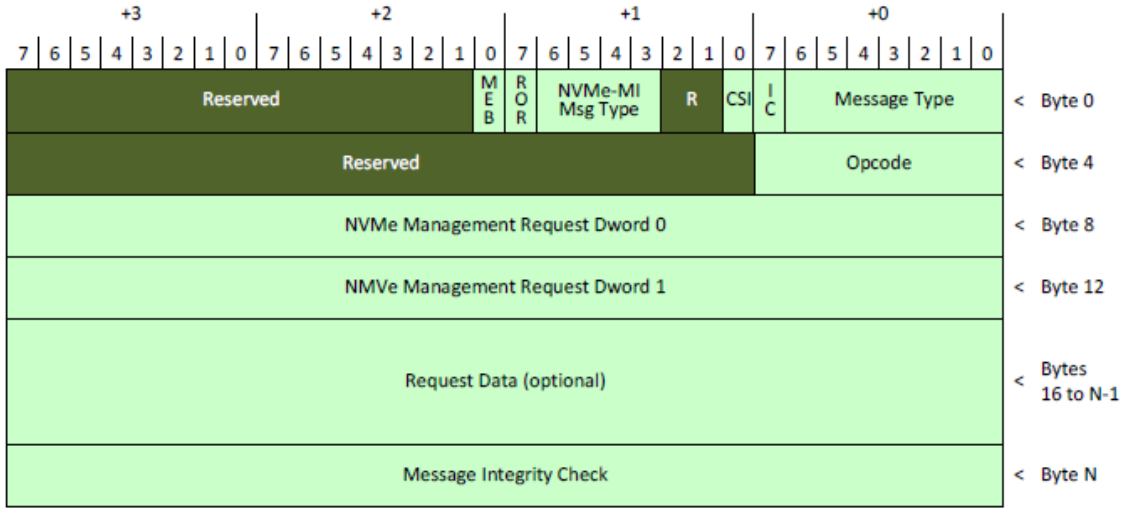
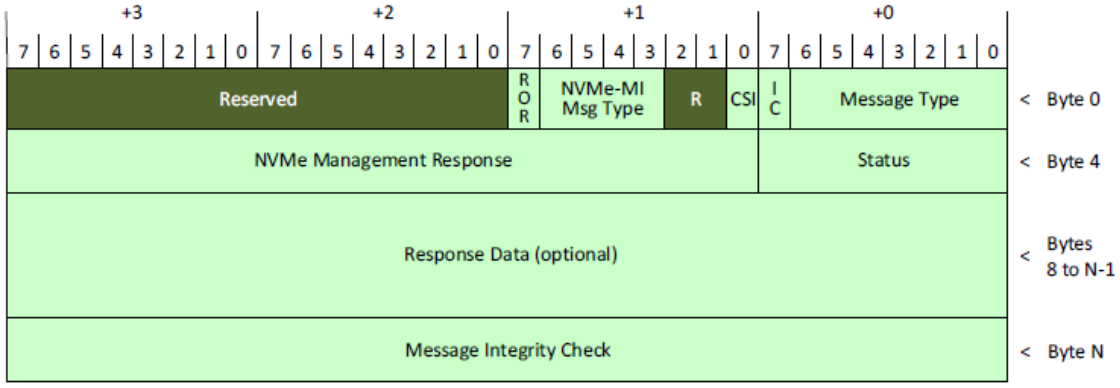


Figure 58: NVMe-MI Command Response Message Format



Command	O/M*	Description
Read NVMe-MI Data Structure	M	Retrieve information about the NVM Subsystem, Management Endpoint, or NVMe Controllers • NVM Subsystem Information • Port Information • Controller Information • Optional Commands Supported
NVM Subsystem Health Status Poll	M	Used to efficiently determine changes in health status attributes associated with the NVM Subsystem (e.g., Unrecoverable error, reset required, PCIe status, Controller SMART / Health Information, composite temperature, composite, and controller status)
Controller Health Status Poll	M	Efficiently determines changes in health status attributes associated with one or more Controllers in the NVM Subsystem
Configuration Get	M	Get NVMe-MI configuration parameter (e.g., SMBus/I2C frequency and MCTP transmission unit size)
Configuration Set	M	Set NVMe-MI configuration parameter
VPD Read	M	Read Vital Product Data (VPD)
VPD Write	M	Write Vital Product Data (VPD)
Reset	O	Reset NVM Subsystem

NVM Express Admin Command Set

Figure 111: NVMe Admin Command Request Format

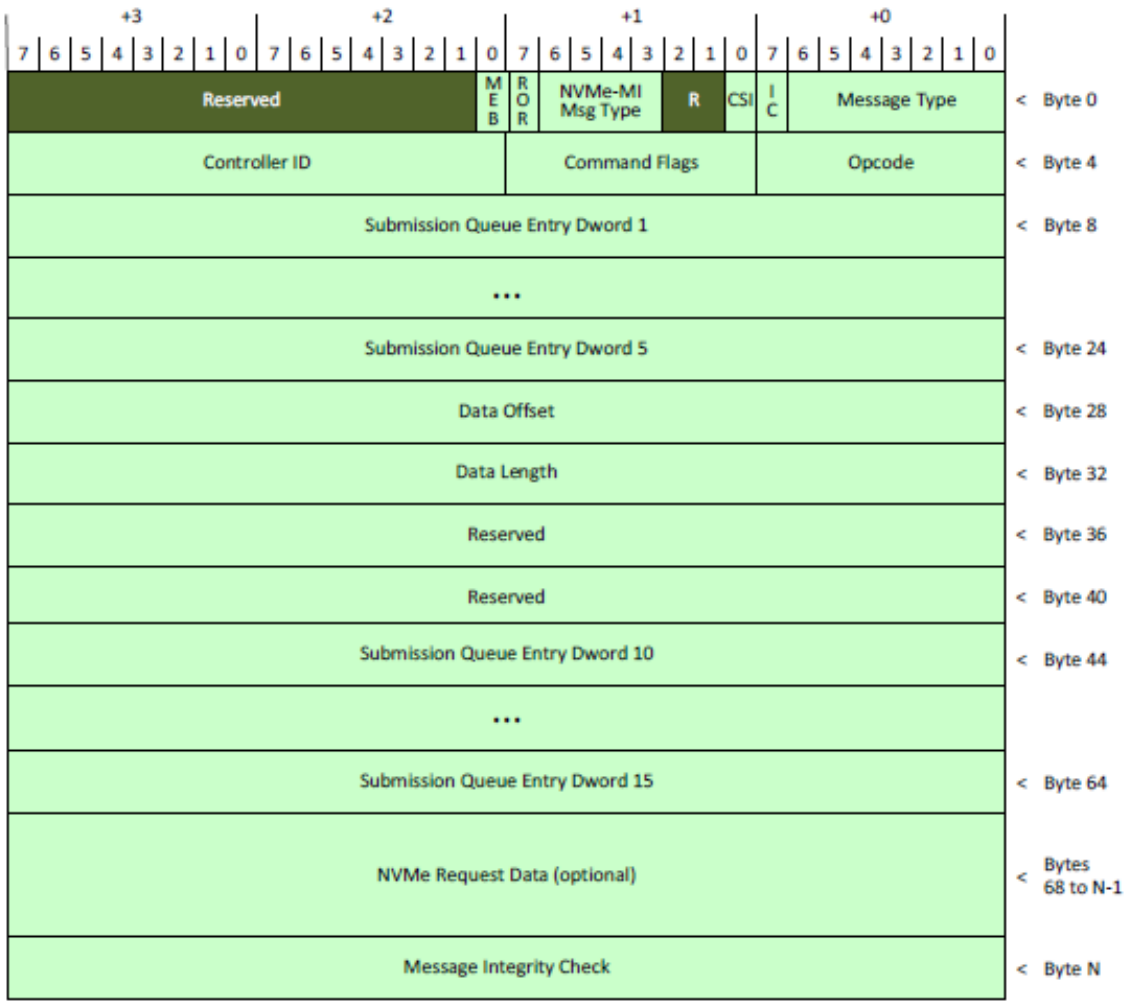
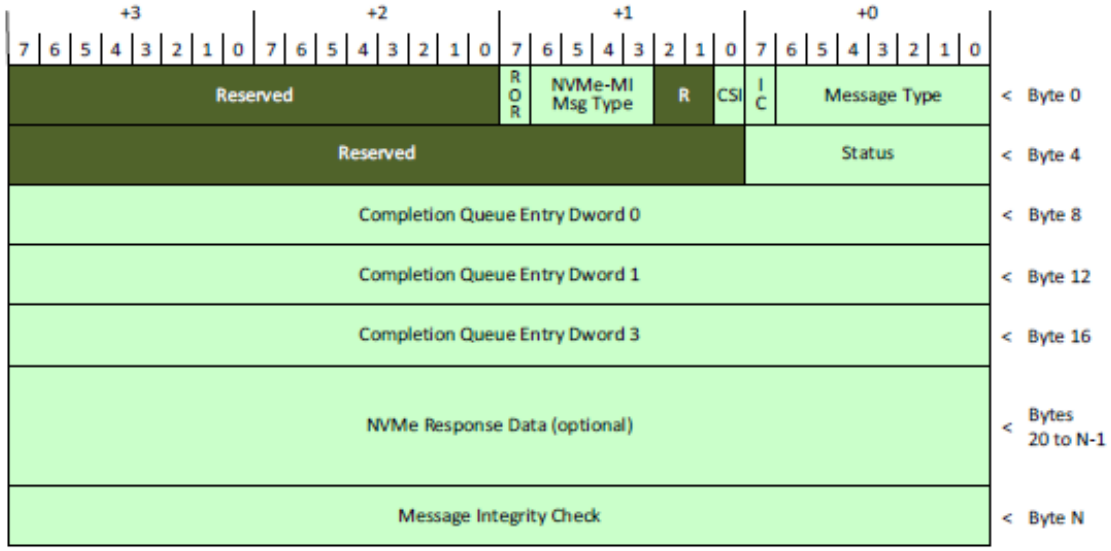


Figure 113: NVMe Admin Command Response Format



Command	O/M*	Description
Firmware Activate/Commit	O	Verifies that a valid firmware image has been downloaded and commits that revision to a specific firmware slot
Firmware Image Download	O	Download all of a portion of a firmware image for a future update to the controller
Format NVM	O	Low level format of the NVM media associated with one or more Namespaces
Get Features	M	Get NVMe configuration parameter
Set Features	O	Set NVMe configuration parameter
Get Log Page	M	Retrieve NVMe log page
Identify	M	Retrieve information about the Controllers, Namespaces, or NVM Subsystem
Namespace Management	O	Create or delete a Namespace
Namespace Attachment	O	Attach or detach a Namespace from a Controller
Security Send	O	Transfer command/data associated with security protocol
Security Receive	O	Transfer command/data associated with security protocol

PCIe Command Set

Figure 117: PCIe Command Request Format

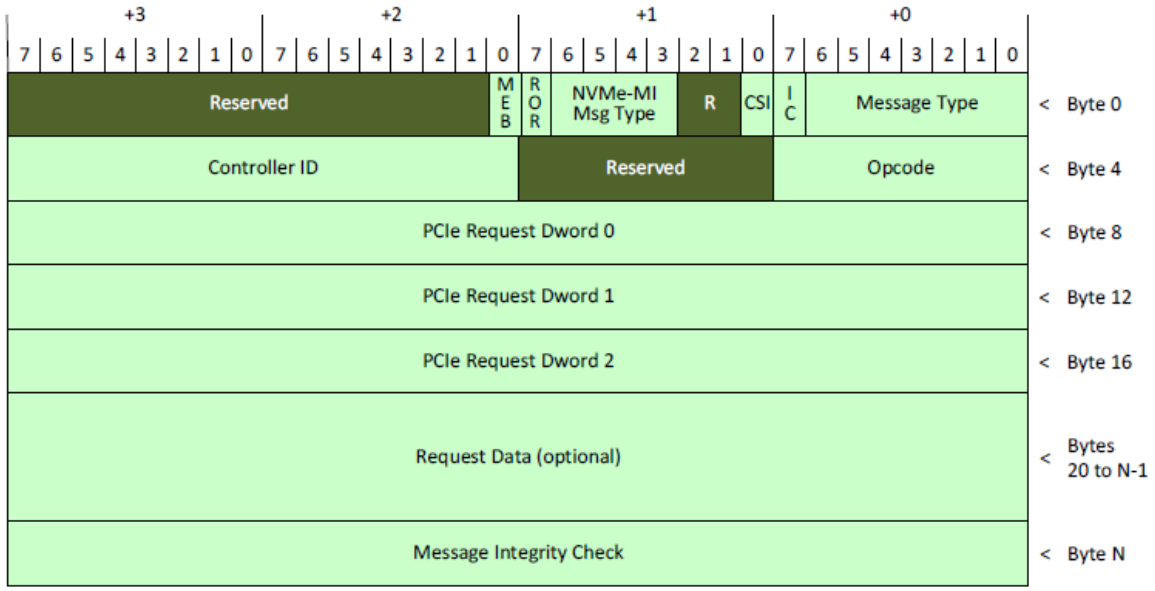
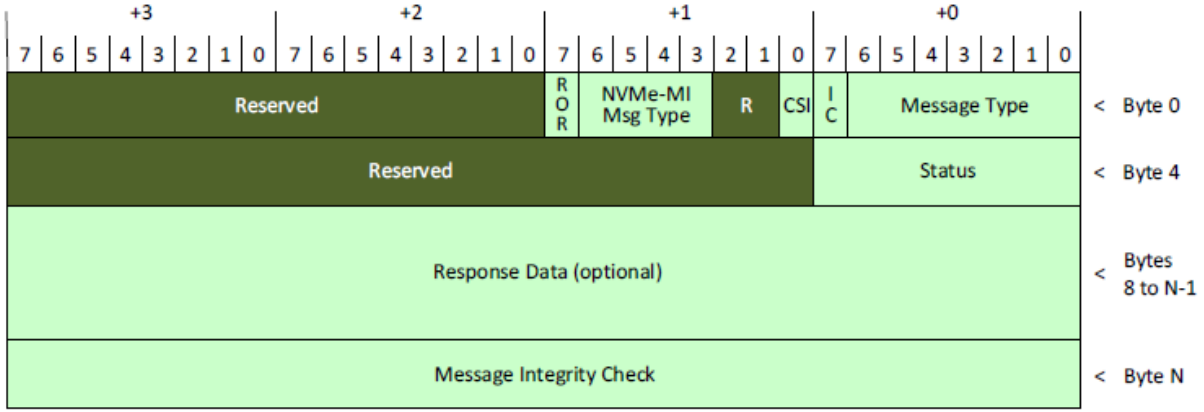


Figure 120: PCIe Command Response Format



Command	O/M*	Description
PCIe Configuration Read	O	Read PCI Express configuration space
PCIe Configuration Write	O	Write PCI Express configuration space
PCIe I/O Read	O	Read PCI Express I/O space
PCIe I/O Write	O	Write PCI Express I/O space
PCIe Memory Read	O	Read PCI Express memory space (BAR memory & MMIO)
PCIe Memory Write	O	Write PCI Express memory space (BAR memory & MMIO)

NVM Express Management Enhancements

Figure 136: NVMe Management Interface Identify Controller

Bytes	O/M[1]	Description
252:240		Reserved
253	M	**NVM Subsystem Report (NVMSR)**: This field reports information associated with the NVM Subsystem. At least one bit in this field shall be set to '1'. Bits: 7:2 Reserved 1: **NVMe Enclosure (NVMEE)**: If set to '1', then the NVM Subsystem is part of an NVMe Enclosure. If cleared to '0', then the NVM Subsystem is not part of an NVMe Enclosure. 0: **NVMe Storage Device (NVMESD)**: If set to '1', then the NVM Subsystem is part of an NVMe Storage Device. If cleared to '0', then the NVM Subsystem is not part of an NVMe Storage Device.
254	M	**VPD Write Cycle Information (VWCI)**: This field indicates information about remaining number of times that VPD contents are able to be updated using the VPD Write command. Bits 7: **VPD Write Cycle Remaining Valid (VWCRV)**: If this bit is set to '1', then the VPD Write Cycle Remaining field is valid. If this bit is cleared to '0', then the VPD Write Cycles Remaining field is invalid and cleared to 0h. Bits 6:0: **VPD Write Cycles Remaining (VWCR)**: If the VPD Write Cycle Remaining Valid bit is set to '1', then this field contains a value indicating the remaining number of times that VPD contents are able to be updated using the VPD Write command. If this field is set to 7Fh, then the remaining number of times that VPD contents are able to be updated using the VPD Write command is greater than or equal to 7Fh. If the VPD Write Cycle Remaining Valid bit is cleared to '0', then this field is not valid and shall be cleared to a value of 0h.
255	M	**Management Endpoint Capabilities (MEC)**: This field indicates the capabilities of the Management Endpoint in the Controller. Bits: 7:2 Reserved 1: **PCIe Port Management Endpoint (PCIEME)**: If set to '1', then the NVM Subsystem contains a Management Endpoint on a PCIe port. 0: **SMBus/I2C Port Management Endpoint (SMBUSME)**: If set to '1', then the NVM Subsystem contains a Management Endpoint on an SMBus/I2C port.

Figure 137: NVMe Management Interface Feature Identifiers

Feature Identifier	NVMe Storage Device O/M [1]	NVMe Enclosure O/M [1]	Persistent Across Power States and Reset [2]	Uses Memory Buffer for Attributes	Description
78h to 7Dh	-	-	-	-	Reserved
7Eh	M	M	No	Yes	Controller Metadata
7Fh	M	O	No	Yes	Namespace Metadata

Out-of-Band Operational Times

Figure 143: Operations Supported During NVM Subsystem Power States

Operation	Powered Off -All Power Rails Off	Powered On -All Power Rails On	Auxiliary Power Only [2] -Main Power Off -Auxiliary Power On	Main Power Only [2] -Main Power On -Auxiliary Power Off
SMBus/I2C VPD and SMBus/I2C Mux Access	Not Supported	Supported	Supported	Implementation Specific
SMBus/I2C MCTP Access	Not Supported	Supported	Optional [1]	Implementation Specific
PCIe MCTP Access	Not Supported	Supported	Not Supported	Supported

NOTES:
1. An implementation that supports SMBus/I2C MCTP Access during Auxiliary Power may support a subset of commands during this power state. The commands that are supported are implementation specific.
2. Auxiliary Power Only and Main Power Only columns are not applicable to form factors that do not define Auxiliary power.

NVM Express™ Management Interface

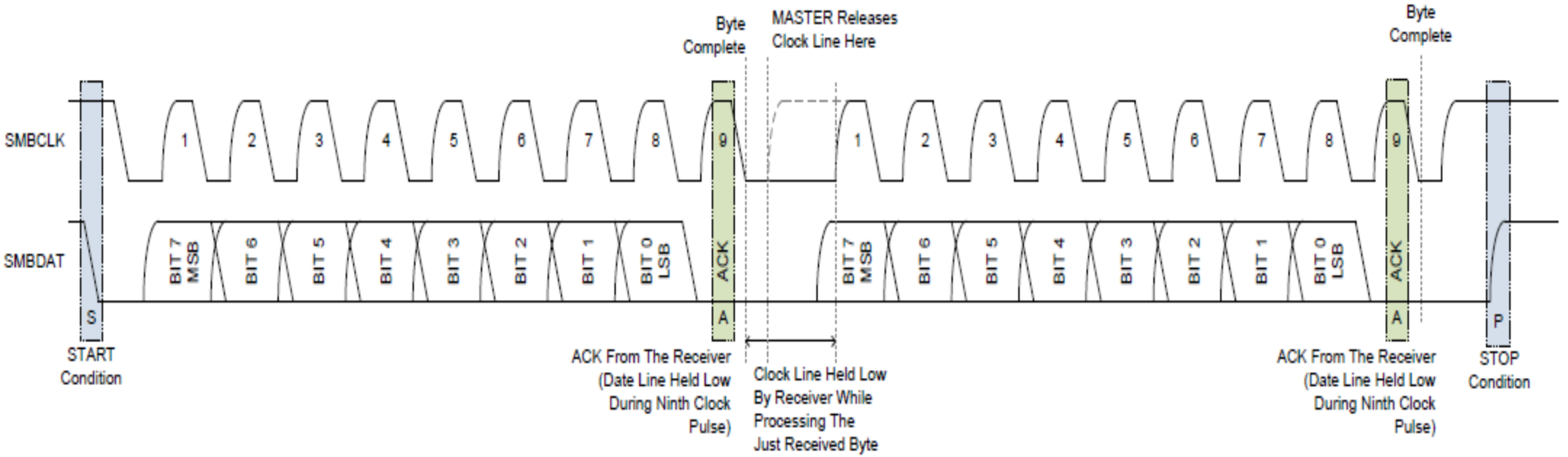
Examples

Example NVMe-MI Messages over SMBus/I2C

- The Request Messages are sent from the Management Controller to the Management Endpoint and the corresponding Response Messages are sent back from the Management Endpoint to the Management Controller.
- The examples assume the following:
  - Management Endpoint SMBus/I2C address is 3Ah;
  - Management Controller SMBus/I2C address is 20h;
  - Management Endpoint MCTP Endpoint ID is 0, examples only use SMBus/I2C address;
  - Management Controller MCTP Endpoint ID is 0, examples only use SMBus/I2C address;
  - MCTP Transmission Unit Size is 64 bytes;
  - NVMe Storage Device Composite Temperature (CTEMP) is 30 °C;
  - NVMe Storage Device Controller ID is 1; and
  - NVMe Storage Device Serial Number is AZ123456.

Example NVMe-MI Messages over SMBus/I2C

- Data Transfers On SMBus: Every byte consists of 8 bits. Each byte transferred on the bus must be followed by an acknowledge bit. Bytes are transferred with the most significant bit (MSB) first.



- Packet Error Checking: The Packet Error Checking mechanism improves reliability and communication robustness, whenever applicable, is implemented by appending a Packet Error Code (PEC) at the end of each message transfer. The PEC is a CRC-8 error-checking byte, calculated on all the message bytes (including addresses and read/write bits). The PEC is appended to the message by the device that supplied the last data byte.

Example NVMe-MI Messages over SMBus/I2C

- Example 1: In this example, a Management Controller issues an Identify Command to read the Serial Number (bytes 23:04 of the Identify Controller Data Structure) of an NVMe Storage Device. The NVMe Storage Device's response is shown in the Example 2.

Start	SSD Addr 0	Ack	Protocol= MCTP	Ack	Length	Ack	BMC Addr 1	Ack	MCTP Version	Ack	SSD EID	Ack	BMC EID	Ack	flags,seq, own, tag	Ack	Type = NVMe-MI	Ack	NVMe Admin	Ack	Rsvd	Ack	Rsvd	Ack
	3Ah		0Fh		45h		21h		01h		00h		00h		8Bh		84h		10h		00h		00h	

Opcode= Identify	Ack	Flags= Len+ Off	Ack	Cntrl Id LSB	Ack	Cntrl Id MSB	Ack	Dword1 LSB	Ack	Dword1	Ack	Dword1	Ack	Dword1 MSB	Ack	Dword2 LSB	Ack	Dword2	Ack	Dword2	Ack	Dword2 MSB	Ack
06h		03h		01h		00h		00h		00h		00h		00h		00h		00h		00h		00h	

Dword3 LSB	Ack	Dword3	Ack	Dword3	Ack	Dword3 MSB	Ack	Dword4 LSB	Ack	Dword4	Ack	Dword4	Ack	Dword4 MSB	Ack	Dword5 LSB	Ack	Dword5	Ack	Dword5	Ack	Dword5 MSB	Ack
00h		00h		00h		00h		00h		00h		00h		00h		00h		00h		00h		00h	

Offset LSB	Ack	Offset	Ack	Offset	Ack	Offset MSB	Ack	Length LSB	Ack	Length	Ack	Length	Ack	Length MSB	Ack	Dword8 LSB	Ack	Dword8	Ack	Dword8	Ack	Dword8 MSB	Ack
04h		00h		00h		00h		14h		00h		00h		00h		00h		00h		00h		00h	

Dword9 LSB	Ack	Dword9	Ack	Dword9	Ack	Dword9 MSB	Ack	Dword10 LSB	Ack	Dword10	Ack	Dword10	Ack	Dword10 MSB	Ack	Dword11 LSB	Ack	Dword11	Ack	Dword11	Ack	Dword11 MSB	Ack
00h		00h		00h		00h		01h		00h		00h		00h		00h		00h		00h		00h	

Dword12 LSB	Ack	Dword12	Ack	Dword12	Ack	Dword12 MSB	Ack	Dword13 LSB	Ack	Dword13	Ack	Dword13	Ack	Dword13 MSB	Ack	Dword14 LSB	Ack	Dword14	Ack	Dword14	Ack	Dword14 MSB	Ack
00h		00h		00h		00h		00h		00h		00h		00h		00h		00h		00h		00h	

PEC	Ack	Stop
B2h		

Start	SSD Addr 0	Ack	Protocol= MCTP	Ack	Length	Ack	BMC Addr 1	Ack	MCTP Version	Ack	SSD EID	Ack	BMC EID	Ack	flags,seq, own, tag	Ack	Dword15 LSB	Ack	Dword15	Ack	Dword15	Ack	Dword15 MSB	Ack
	3Ah		0Fh		0Dh		21h		01h		00h		00h		5Bh		00h		00h		00h		00h	

CRC32C LSB	Ack	CRC32C	Ack	CRC32C	Ack	CRC32C MSB	Ack	PEC	Ack	Stop
4Ah		C3h		2Ch		FAh		EFh		

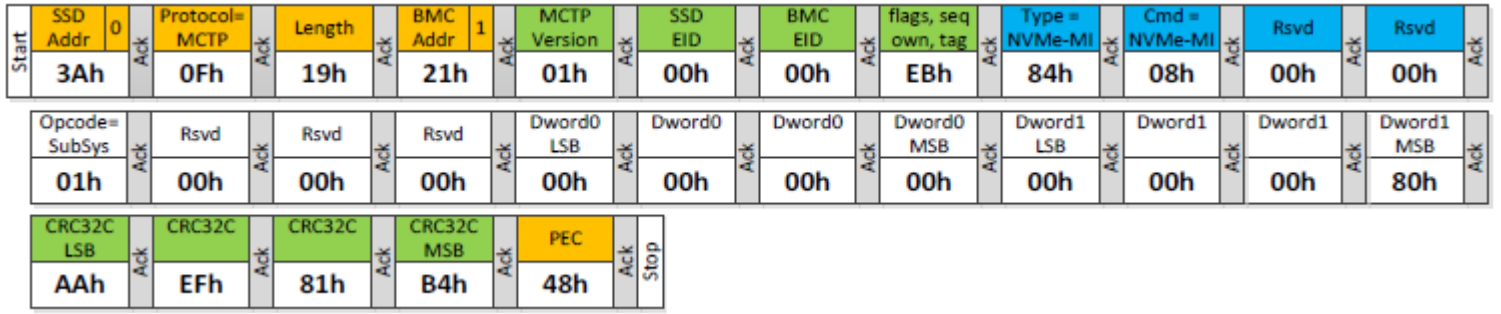
Example NVMe-MI Messages over SMBus/I2C

- Example 2: This example shows an NVMe Storage Device's Response Message to the Identify Command from Example 1. This message is small enough to fit in a single packet so both MCTP SOM and EOM flags are set. The NVM Express specification defines the format (Dwords 0, 1, and 3) of the Identify Controller Data Structure bytes that are returned.

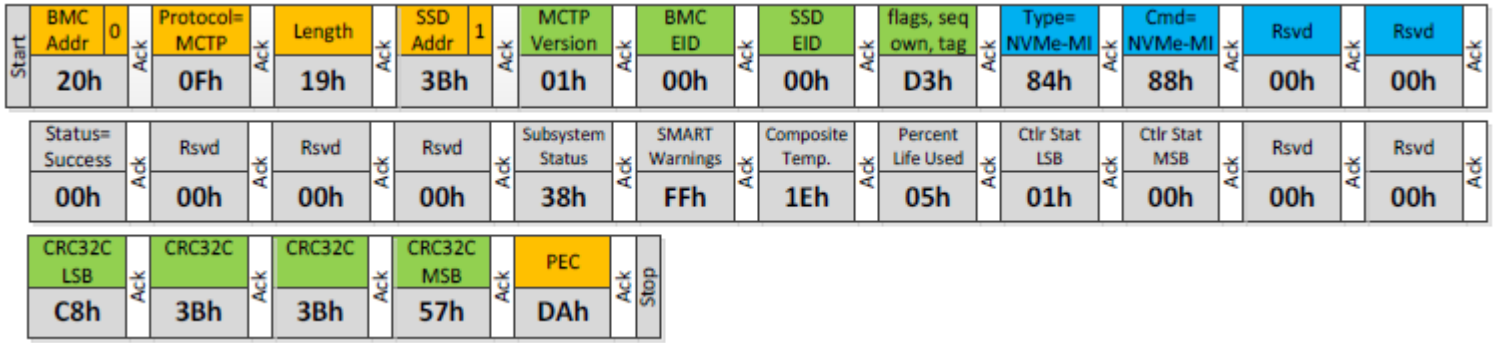
Start	BMC Addr | 0	Ack	Protocol= MCTP	Ack	Length	Ack	SSD Addr | 1	Ack	MCTP Version	Ack	BMC EID	Ack	SSD EID	Ack	flags, seq own, tag	Ack	Type= NVMe-MI	Ack	NVMe Admin	Ack	Rsvd	Ack	Rsvd	Ack
	20h		0Fh		31h		3Bh		01h		00h		00h		C3h		84h		90h		00h		00h	
	Status= Success	Ack	Rsvd	Ack	Rsvd	Ack	Rsvd	Ack	Dword0 LSB	Ack	Dword0	Ack	Dword0	Ack	Dword0 MSB	Ack	Dword1 LSB	Ack	Dword1	Ack	Dword1	Ack	Dword1 MSB	Ack
	00h		00h		00h		00h		00h		00h		00h		00h		00h		00h		00h		00h	
	Dword3 LSB	Ack	Dword3	Ack	Dword3	Ack	Dword3 MSB	Ack	Response Data 'A'	Ack	Response Data 'Z'	Ack	Response Data '1'	Ack	Response Data '2'	Ack	Response Data '3'	Ack	Response Data '4'	Ack	Response Data '5'	Ack	Response Data '6'	Ack
	00h		00h		00h		00h		41h		5Ah		31h		32h		33h		34h		35h		36h	
	Response Data ''	Ack	Response Data ''	Ack	Response Data ''	Ack	Response Data ''	Ack	Response Data ''	Ack	Response Data ''	Ack	Response Data ''	Ack	Response Data ''	Ack	Response Data ''	Ack	Response Data ''	Ack	Response Data ''	Ack	Response Data ''	Ack
	20h		20h		20h		20h		20h		20h		20h		20h		20h		20h		20h		20h	
	CRC32C LSB	Ack	CRC32C	Ack	CRC32C	Ack	CRC32C MSB	Ack	PEC	Ack	Stop													
	7Ah		1Fh		C4h		7Bh		48h															

Example NVMe-MI Messages over SMBus/I2C

- Example 3: In this example, a Management Controller issues an NVM Subsystem Health Status Poll command and clears the Composite Controller Status.

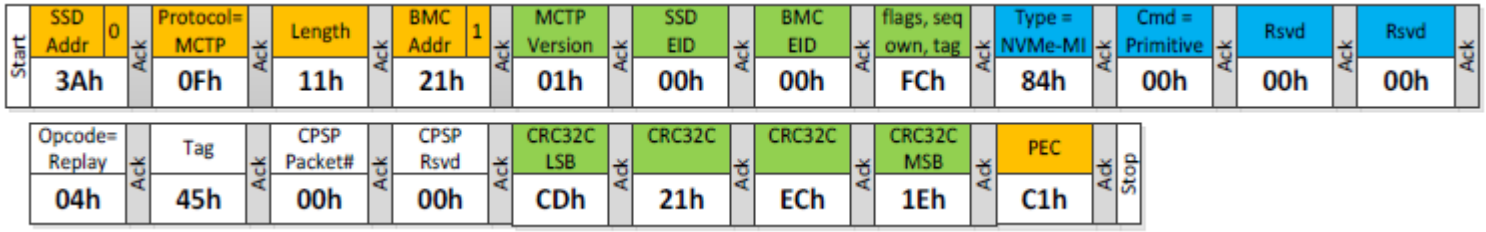


- Example 4: This example shows an NVMe Storage Device's response to the NVM Subsystem Health Status Poll command from Example 3.

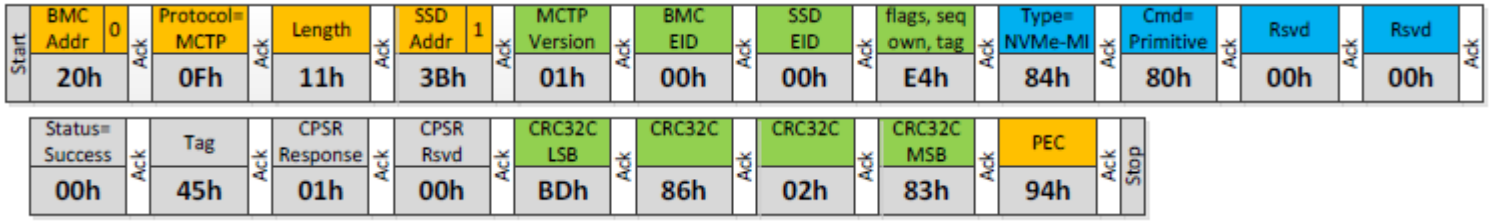


Example NVMe-MI Messages over SMBus/I2C

- Example 5: This example shows a Management Controller issuing a Replay Control Primitive.



- Example 6: This example shows an NVMe Storage Device sending an acknowledgement Response Message to the Replay Control Primitive and then sending a second Response Message that replays the previous Response Message from specified offset of zero.



Denali Open-Channel SSDs

Introduction

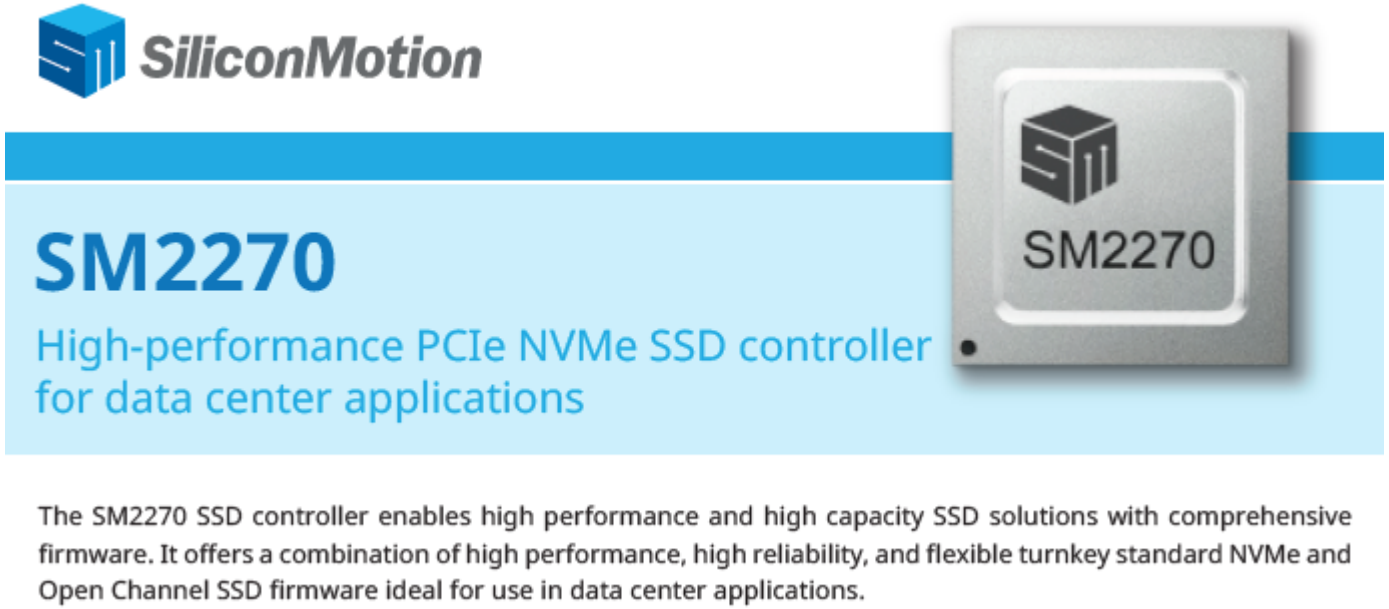
Introduction

SM2270

Silicon Motion Announces New Dual-Mode Enterprise Class SSD Controller Solution at 2018 Flash Memory Summit:

SM2270 SSD controller is designed with standard NVMe™ and Open Channel capabilities for enterprise and data center storage

The dual-mode SM2270 is a complete SSD controller solution with customer specific or turnkey firmware that can support Open Channel storage implementations as well as standard NVMe protocols.
"The SM2270 is the world's first PCIe SSD controller that supports standard NVMe and Open Channel technology to be available in production. It draws on Silicon Motion's leadership position in flash controller technology and unrivaled knowledge of the data center SSD storage systems," said Wallace C. Kou, President and CEO of Silicon Motion. "This combination makes the SM2270 the ideal controller in SSDs for demanding cloud server and data center applications."



Introduction

Open-Channel

A new class of SSDs has been developed known as Open-Channel SSDs. Open-Channel SSDs differ from a traditional SSD in that they expose the internal parallelism of the SSD to the host and allows it manage it. This allows Open-Channel SSDs to provide three properties to the host:

- I/O Isolation

I/O isolation provides a method to divide the capacity of the SSD into a number of I/O channels that map the parallel units of the device. This enables an Open-Channel SSD to be used in multi-tenant applications without tenants interfering with each other.

- Predictable latency

Predictable latency is achieved by having control in the host over when, where and how I/O are submitted to the SSD.

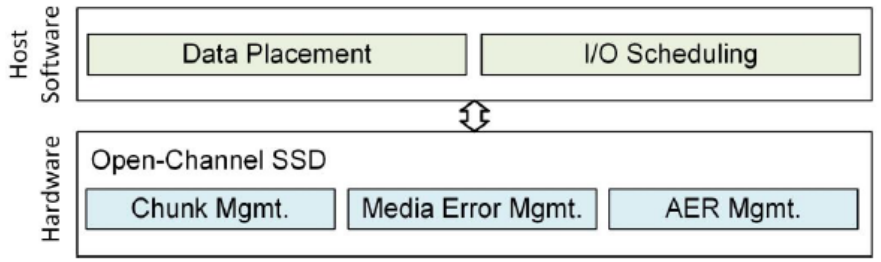
- Software-Defined Non-Volatile Memory

By integrating the SSD flash translation layer into the host, workload optimizations can be applied either within a self-contained flash translation layer, file-system integration or applications themselves.

Host-Managed Non-Volatile Memory Management

- Enables the host to control data placement, I/O scheduling and implement tight integrations with file-systems and applications.

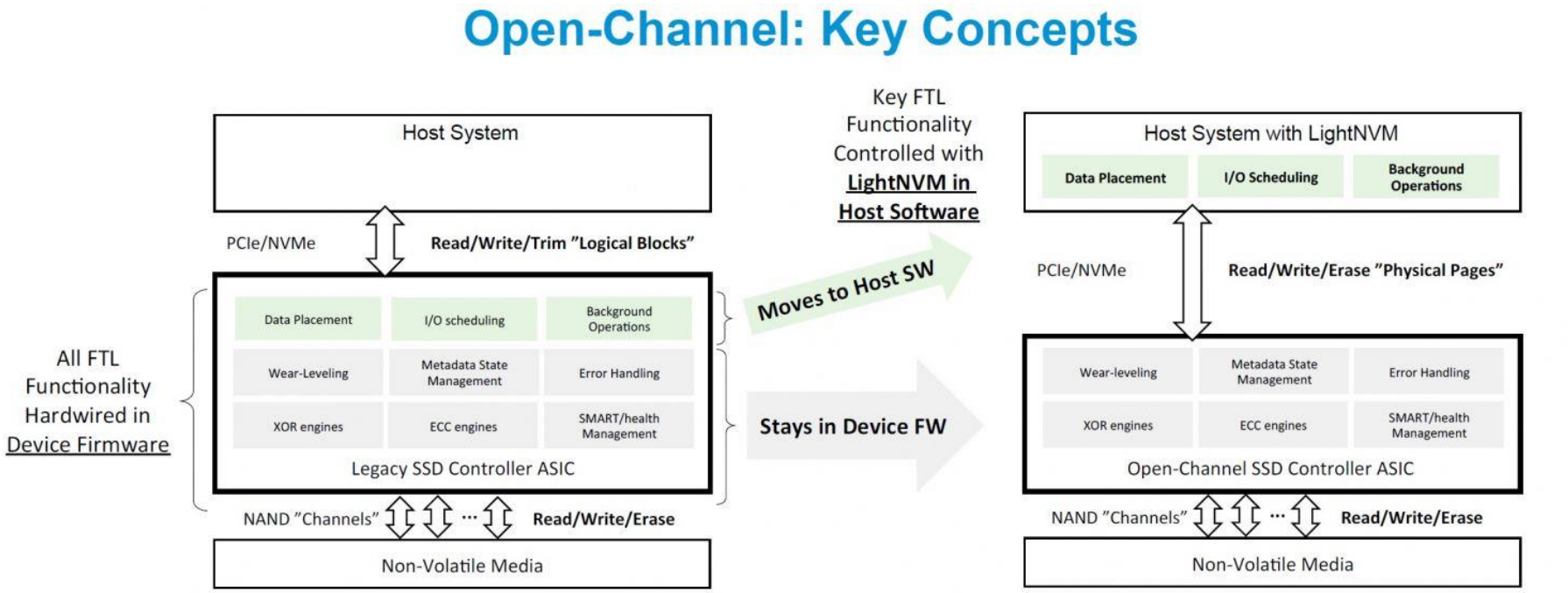
Figure 1: Responsibility of an Open-Channel SSD



Introduction

Open-Channel

An open-channel solid state drive is a solid-state drive which does not have a firmware Flash Translation Layer implemented on the device, but instead leaves the management of the physical solid-state storage to the computer's operating system. The interface used by the operating system to access open-channel solid state drives is called LightNVM.



Traditional SSD

- Logical Block Addressing (LBA) on Device
- FTL controlled by Device Firmware ("Black-Box")
- Fixed functionality & performance

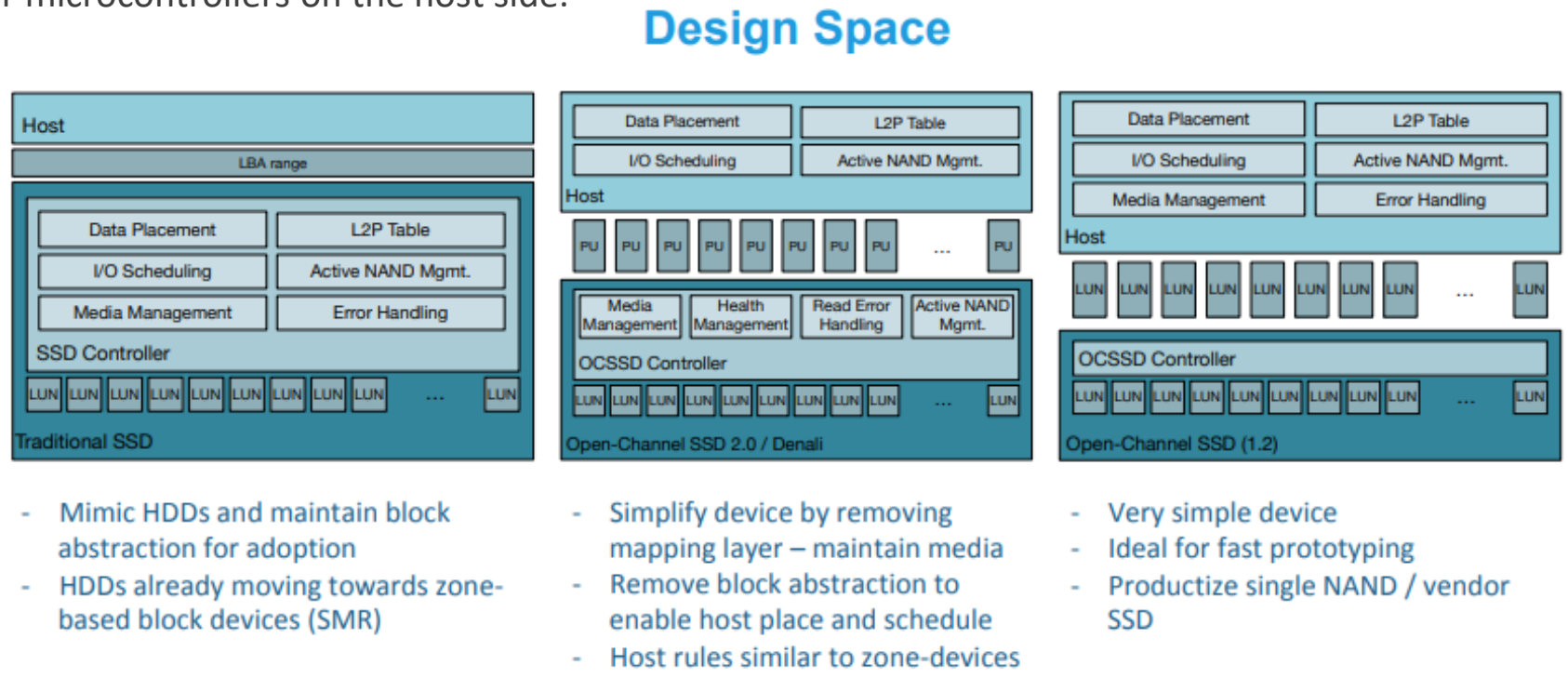
Open-Channel SSD

- Physical Page Addressing (PPA) Command Set
- Key FTL functions exposed to LightNVM on Host
- Flexible for application-specific performance

Introduction

Microsoft Denali

Project Denali is a standardization and evolution of Open Channel that defines the roles of SSD vs. that of the host in a standard interface. Media management, error correction, mapping of bad blocks and other functionality specific to the flash generation stays on the device while the host receives random writes, transmits streams of sequential writes, maintains the address map, and performs garbage collection. Denali allows for support of FPGAs or microcontrollers on the host side.



Introduction

Theory of Operation

- A group is a shared bus within the Open-Channel SSD, where bandwidth and transfer overhead are shared between the parallel units attached. A parallel unit is the unit of parallelism within the SSD, and its access is fully independent of other parallel units. Note that the representation is not necessarily the physical media attachments and can be a logical representation.
- Within Parallel units, there is a set of chunks. The chunks represent a linear range of logical blocks. The chunks require that writes to them are issued sequentially and that a chunk is reset before being written to again.
- This hierarchical representation allows the host to control which parallel unit is accessed independently from other parallel units. The SSD firmware should not be optimized for any particular workload, as the host will manage access to the parallel units and which chunks are accessed, such that the host can manage the SSD characteristics with respect to parallelism.

Figure 2 shows the internal representation of an open-channel SSD. Its structure is divided into Groups, Parallel Units (PUs), Chunks, and Logical blocks.

Figure 2: Logical parallelism in an SSD

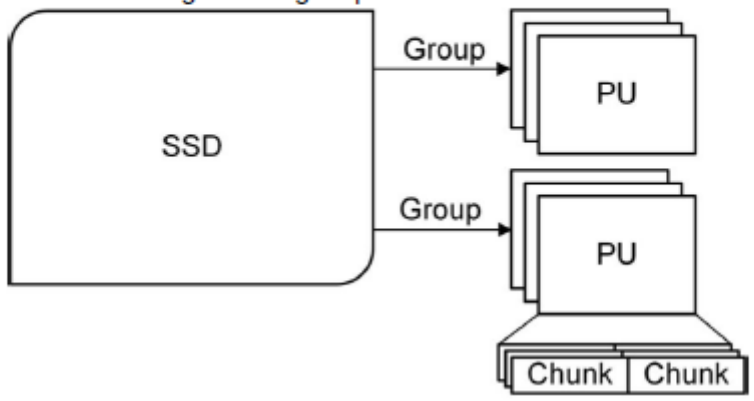
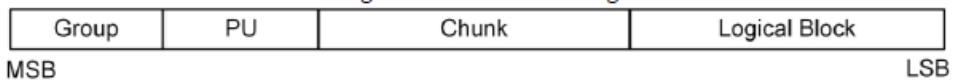


Figure 3: LBA Encoding



Introduction

Theory of Operation

- There are three I/O command types: Read, Write and Reset. Data can be retrieved from a chunk by issuing reads. The minimum read size is the logical block size. To write data, the host must write sequentially within a chunk, and each command must fulfill the minimum write unit size. After a chunk is fully written, the chunk must be reset by the host before it can be written again.
- Each chunk maintains at least four fields: A start LBA (SLBA), number of logical blocks (NLB), write pointer (WP), and chunk state (CS). They are used to communicate the start of the chunk, its size, when a chunk is open, and the number of logical blocks that have been written within a chunk.
- The state field defines the state of a chunk. It can be either (i) free, (ii) open, (iii) closed, or (iv) offline. During the lifetime of a chunk, it cycles through (i-iii), whereas, at the end of its lifetime, its state becomes (iv) offline.

Figure 4: Chunk Fields

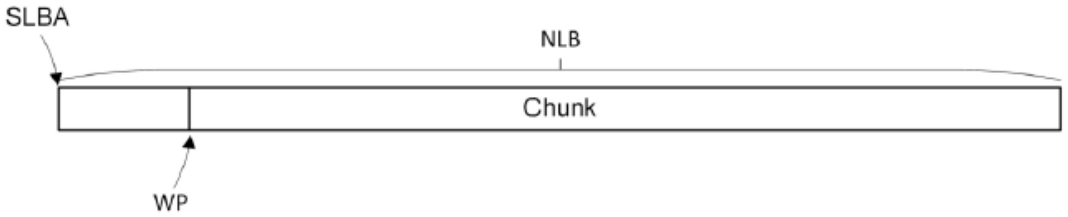
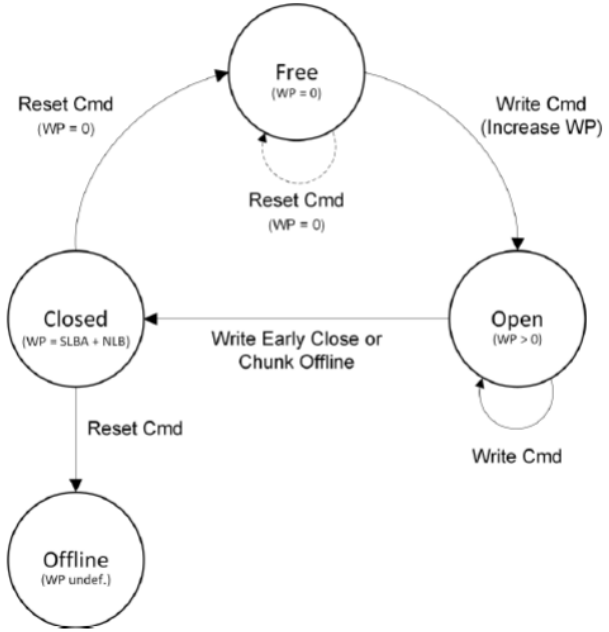


Figure 5: Chunk State Diagram



Denali Open-Channel SSDs

Physical Page Address Command Set

Physical Page Address Command Set

Admin Commands

- Geometry: For the host to issue I/O commands to the device, the boundaries of groups, parallel units, and chunks must be known. The geometry command communicates these boundaries to the host.
- Get Log Page - Chunk Information: The Chunk Information log page communicates the per-chunk description of all chunks on the device. The information provided is over the life of the controller and is retained across power cycles.
- Media Feedback (Feature Identifier CAh): This feature controls read command feedback.

Figure 9: Opcodes for Admin Commands

Opcode (07)	Opcode (06:02)	Opcode (01:00)	NVMe Opcode	O/M	Namespace Identifier Used	Command
Generic	Function	Data Transfer				
1b	110 00b	10b	E2h	M	Yes	Geometry
0b	000 00b	10b	02h	M	Yes	Get Log Page - Chunk Information
0b	000 10b	01b	09h	M	Yes	Set Features - Media Feedback
0b	000 10b	10b	0Ah	M	Yes	Get Features - Media Feedback

Physical Page Address Command Set

I/O Commands

- Vector Chunk Reset: This command issues a reset of a list of chunks. If a metadata region is set, the reset command shall update the metadata buffer with the new chunk descriptor of the reset chunks.
- Vector Chunk Write: The vector chunk write command writes data and if the namespace supports it, metadata, to the LBA list indicated.
- Vector Chunk Read: The vector chunk read command reads data and if the namespace supports it, metadata, to the LBA list indicated.
- Vector Chunk Copy: The vector chunk copy command copies data and user metadata from one set of LBAs to another set of LBAs using internal device buffers. The destination addresses must follow the write requirements of the drive.

Figure 19: Opcodes for Data Commands

Opcode (07) Generic	Opcode (06:02) Function	Opcode (01:00) Data Transfer	NVMe Opcode	O/M	Command
0b	000 00b	01b	01h	M	Write (From NVMe 1.3 specification)
0b	000 00b	10b	02h	M	Read (From NVMe 1.3 specification)
0b	000 10b	01b	09h	M	Dataset Management (From NVMe 1.3 specification)
1b	001 00b	00b	90h	O	Vector Chunk Reset
1b	001 00b	01b	91h	O	Vector Chunk Write
1b	001 00b	10b	92h	O	Vector Chunk Read
1b	001 00b	11b	93h	O	Vector Chunk Copy

Physical Page Address Command Set

Asynchronous Event Information

An Open-Channel SSD may implement an out-of-band feedback mechanism that informs the host about the state of its media. The feedback mechanism is used to allow the host to perform data management based on media-centric feedback.

Figure 33: Asynchronous Event Information Completion Queue Entry

Bytes	Description
31:24	Reserved
23:16	"D0" - Log Page Identifier. The log page needs to be read by the host to clear the event.
15:08	See the "Chunk Notification Log Information" table.
07:03	Reserved
02:00	"7h" - Vendor specific

Figure 34: Log Page Identifier Chunk Notification

Value	Description
D0h	**Chunk Notification Log (CNL):** Report the state of the block. See log page definition in Figure 35 for further information.

Physical Page Address Command Set

Wear-leveling

- Since actions taken to distribute wear evenly affect the drive's isolation guarantee, maintaining an even wear across chunks is the responsibility of the host.

Figure 11: Device Geometry Data Structure

Byte	Description
32	**Wear-level Index Delta Threshold (WIT):** Value to maintain wear-level indices within. All chunks wear-level index delta should be plus or minus this value.

Figure 16: Chunk Information - Chunk Descriptor Table

2	**Wear-level Index (WLI):** Reports the amount of endurance this chunk has spent. This is a monotonically increasing value from 0 to 255. 0 being beginning of chunk lifetime, and 255 being at the chunk end of life.

- For example, if the threshold is 10 and the average chunk wear is 100, the wear-level index delta is allowed to be within the range of 90-110. If host does not wear-level index delta according to the threshold the device may signal it through a Chunk Notification Log event.

Figure 35: Get Log Page - Chunk Notification Log Entry (Log Identifier D0h)

21:20	**State (S):** Field that indicate the state of the block.

Bits	Description
15:09	Reserved
08	If set to '1' then the chunk's wear-level index is outside the average wear-level index threshold defined by the controller.

Physical Page Address Command Set

LBA format

- The LBA format encodes the geometry into the LBA by dedicating parts of the LBA to be used as an index for corresponding parts of the geometry.

- As an example, assume the following configuration:
  - 16 groups - 4 bits required.
  - 4 PUs within each group - 2 bits required.
  - 1004 chunks within each PU - 10 bits required - Max 1024.
  - 4096 logical blocks within a chunk. 12 bits required.

Each bit of the LBA therefore correspond to:

- Bits 11-0 specify a logical block within a chunk.
- Bits 21-12 specify a chunk within a PU
- Bits 23-22 specify a PU within a group
- Bits 27-24 specify a group
- Bits 63-28 are unused.

Figure 12: LBA format

Byte	Description
0	**Group bit length:** Contiguous number of bits assigned to Group addressing.
1	**PU bit length:** Contiguous number of bits assigned to PU addressing.
2	**Chunk bit length:** Contiguous number of bits assigned to Chunk addressing.
3	**Logical block bit length:** Contiguous number of bits assigned to logical blocks within Chunk.
7:4	Reserved

NVMe & MI

Q & A


www.atpinc.com
INDUSTRIAL
ONLY