NVMe based PCIe SSD Validation Challenges and Solutions

Apurva Vaidya
iGATE Global Solutions

Abstract

- PCI express Solid State Drives (PCIe SSDs) provide significant performance benefits in enterprise applications compared to traditional hard disc drives (HDDs) and SSDs with a legacy storage interface.

- The current standard (SAS, SATA) poses architectural limitations that prohibit them to deliver much desired throughput for SSD.

- Emergence of non-volatile memory express (NVMe), a scalable host controller interface specifically developed for PCIe SSDs, and a supporting ecosystem, allows SSD suppliers to transition to NVMe based PCIe SSD products.

- This presentation highlights the validation challenges such as queue depth handling, protocol validation ,asynchronous event handling and queue management along with solutions to address these challenges.

NVM Express(NVMe) Overview

- NVM Express or NVMe (also referred to as Enterprise NVMHCI) defines register interface for communication with a non-volatile memory subsystem and a standard command set for use with the NVM subsystem

- Version 1.1 of the specification was released on October 11, 2012

- NVMe advantages:
  - It is highly scalable host controller interface designed to address the needs of enterprise and client systems that utilize PCI Express based SSDs.
  - The interface provides optimized command submission and completion paths.
  - It includes support for parallel operation by supporting up to 64K I/O Queues with up to 64K commands per I/O Queue.
  - Support for many Enterprise capabilities like end-to-end data protection, error reporting and virtualization.

NVMe Key Attributes(1/2)

- Support for up to 64K I/O queues, with each I/O queue supporting up to 64K commands.
- Priority associated with each I/O queue with well-defined arbitration mechanism.
- All information to complete a 4KB read request is included in the 64 byte command itself, ensuring efficient small random I/O operation.
- Efficient and streamlined command set.
- Support for MSI/MSI-X and interrupt aggregation.

NVMe Key Attributes(2/2)

- Support for multiple namespaces.
- Efficient support for I/O virtualization architectures like SR-IOV.
- Robust error reporting and management capabilities.
- Enterprise: Support for end-to-end data protection (i.e., DIF/DIX).
- Enterprise: Support for multi-path I/O, including reservations.
- Optimized queuing interface, command set, and feature set for PCIe SSDs.

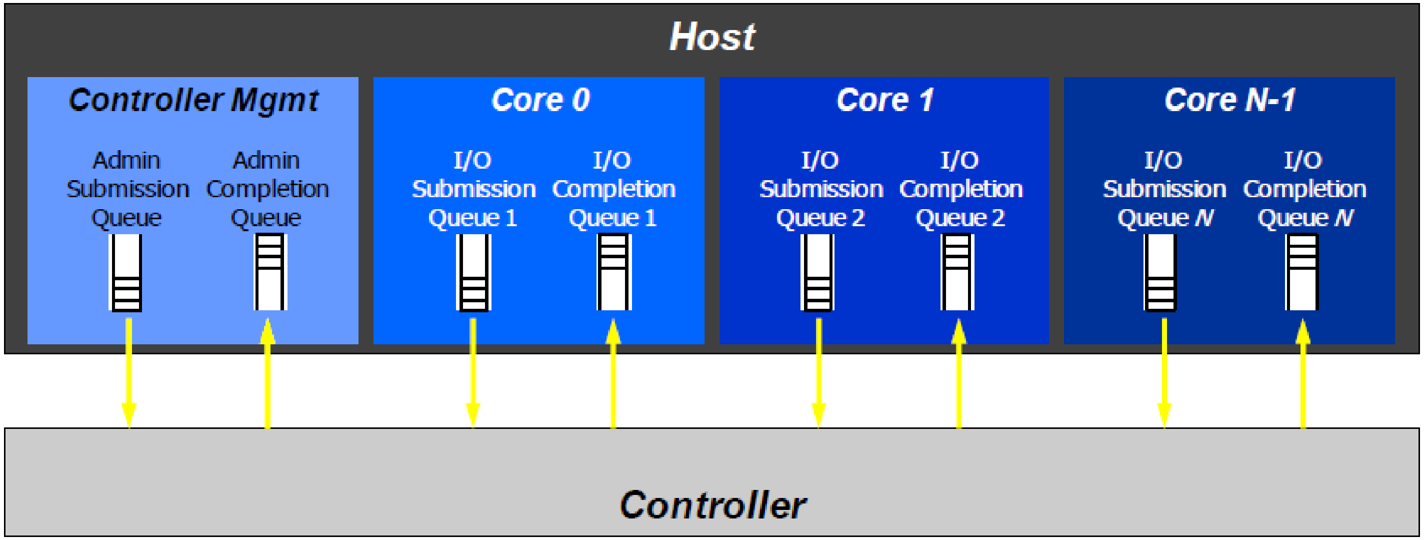
NVMe Queuing Model

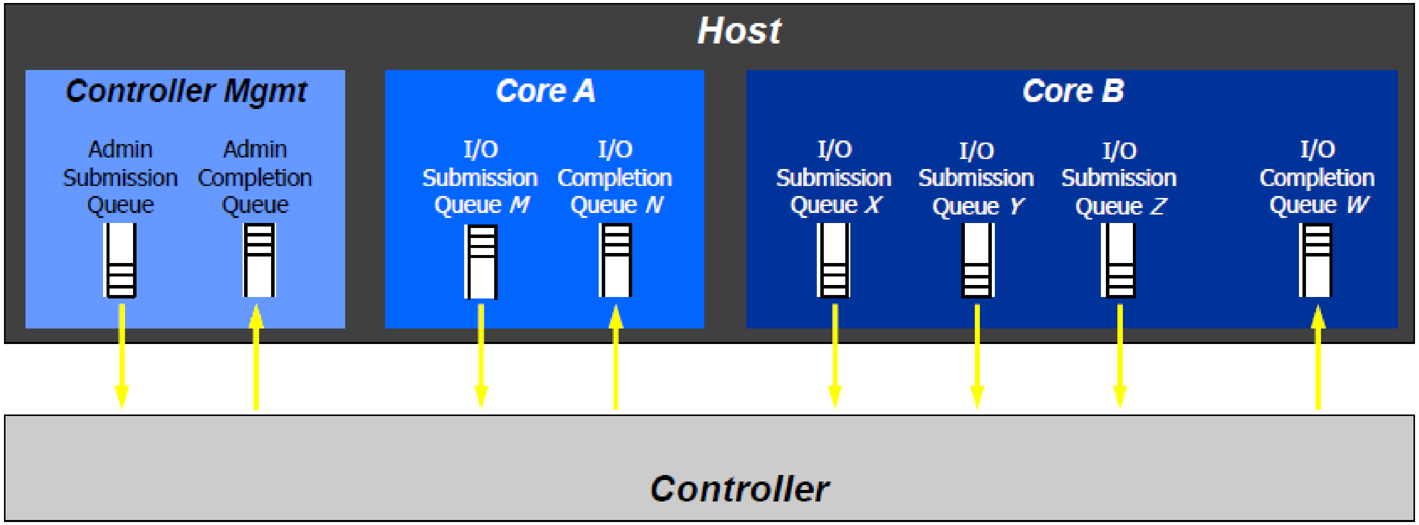
- NVM Express is based on a paired Submission and Completion Queue mechanism.
  - Commands are placed by host software into a Submission Queue.
  - Completions are placed into the associated Completion Queue by the controller.

- An Admin Submission and associated Completion Queue exist for the purpose of controller management and control
  - Creation and Deletion of I/O Submission and Completion Queues, aborting commands, etc.
  - Only commands from Admin Command Set may be submitted to the Admin Submission Queue.

- An I/O Command Set is used with an I/O queue pair. The specification defines one I/O Command Set, named the NVM Command Set.

NVMe Queuing Model





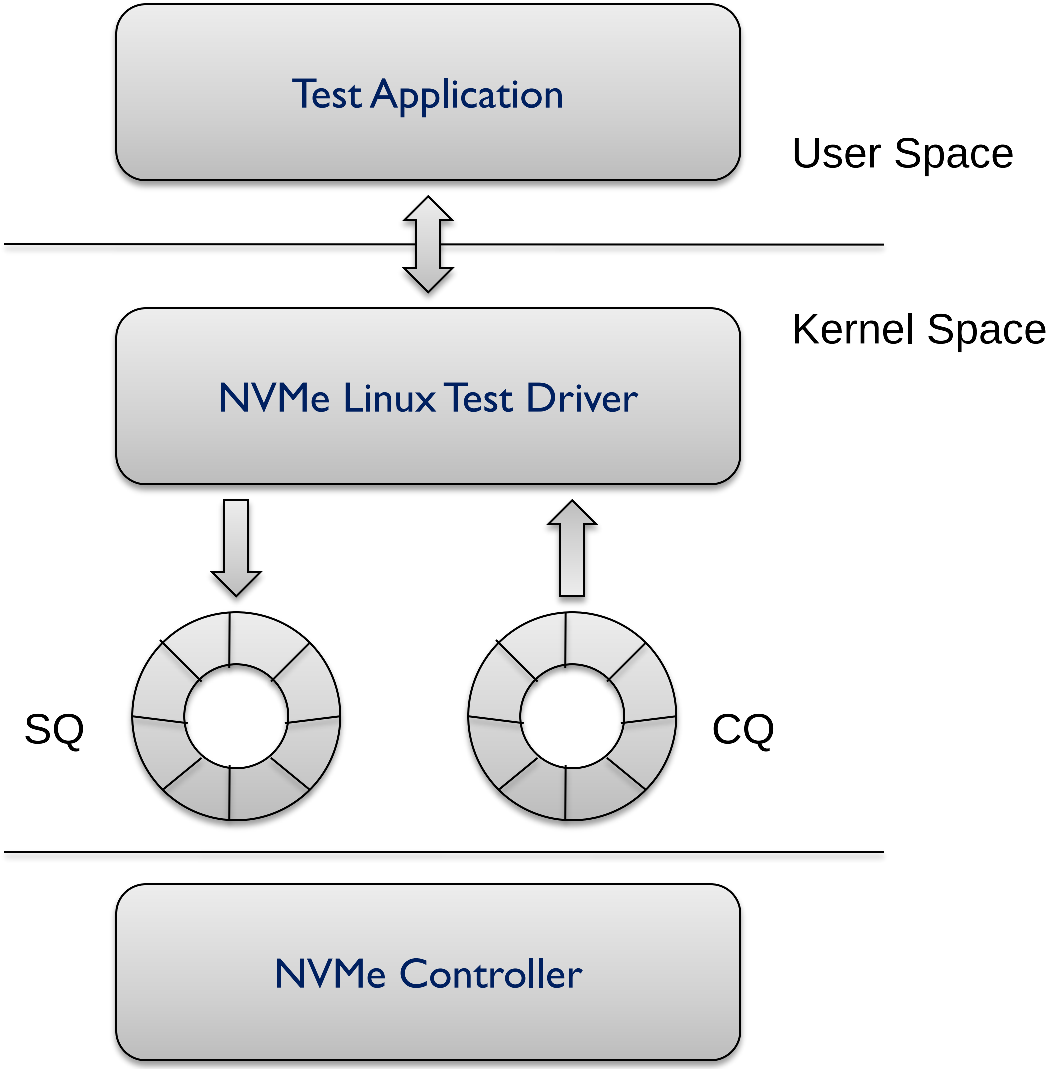
NVMe Validation Overview

- As every vendor uses the reference driver with their product, we thought of a test application that can be configured to validate the product firmware with the driver.

- Currently the test application is developed on Linux and works with modified NVMe Linux driver (NVMe Linux Test Driver).

- The aspects covered in NVMe validation are:
  - Protocol implementation Validation
  - Asynchronous Event Validation
  - Queue Management Validation
  - Queue Depth Validation

NVMe Validation - Setup

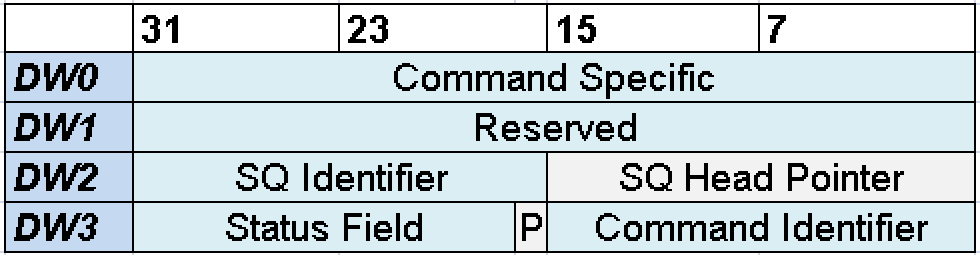


Challenge: Protocol implementation Validation

- The NVMe Specification contains Admin command set and NVM command set

- Each command(submission queue entry) is 64 byte and contains multiple Dwords (DW).

- An entry in completion queue is 16 bytes.



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

- The challenge is to test and validate the behavior of the NVMe controller for different valid/invalid commands(submission queue entries) and the response(completion queue entry).

Solution: Protocol implementation Validation

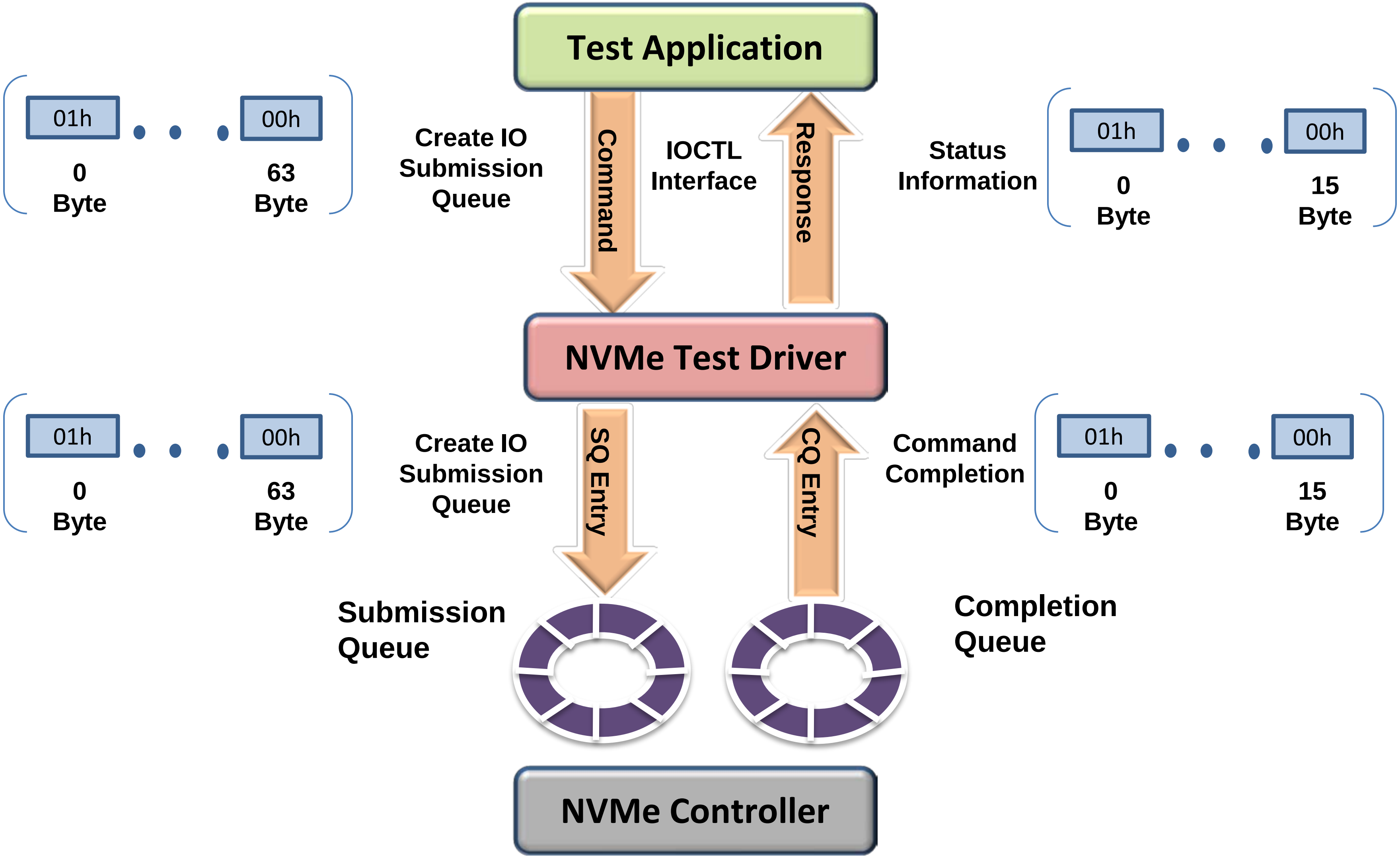
- NVMe test application provides an interface to specify value of each DW in a command.

- Addition of direct command support in the existing NVMe driver, where the driver receives the command data from the application and puts it into the submission queue.
  - Support for direct command in the driver includes interpreting the command opcode and associated handling.

    e.g. creation of a queue, deletion of a queue, etc
  - Interpreting the response and sending appropriate status information to the test application.

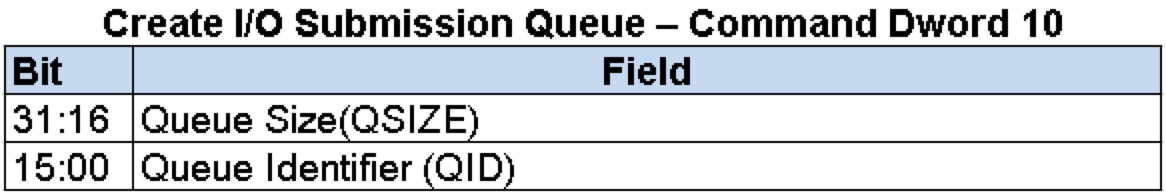
- This gives flexibility to test normal, abnormal and boundary conditions in NVM and Admin command set.

Solution: Protocol implementation Validation



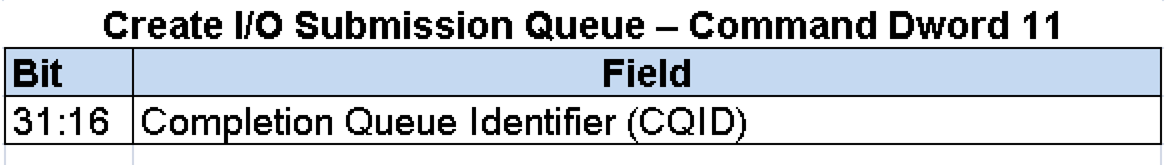
Example: Protocol Implementation Validation

- The following DWs can be set to valid/invalid values to test the response from the controller



Create I/O Submission Queue – Command Dword 10

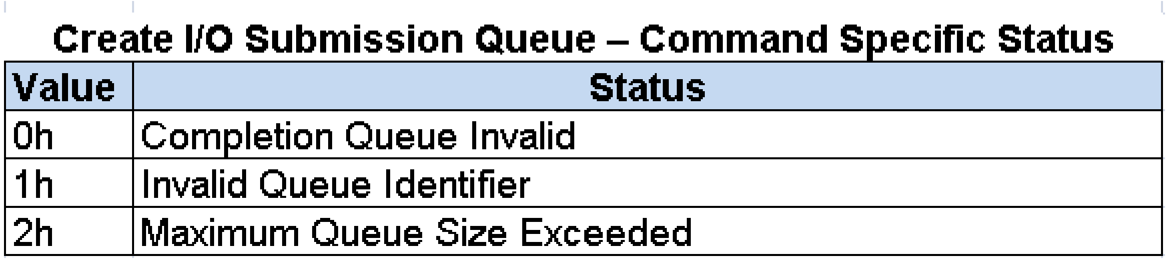
Bit	Field
31:16	Queue Size(QSIZE)
15:00	Queue Identifier (QID)



Create I/O Submission Queue – Command Dword 11

Bit	Field
31:16	Completion Queue Identifier (CQID)

- The command specific status values can be



Create I/O Submission Queue – Command Specific Status

Value	Status
0h	Completion Queue Invalid
1h	Invalid Queue Identifier
2h	Maximum Queue Size Exceeded

Challenge: Asynchronous Event Validation

- Asynchronous events are used to notify host software of status, error, and health information as these events occur.

- Host software needs to submit one or more Asynchronous Event Request commands to the controller.

- The controller specifies an event to the host by completing an Asynchronous Event Request command.

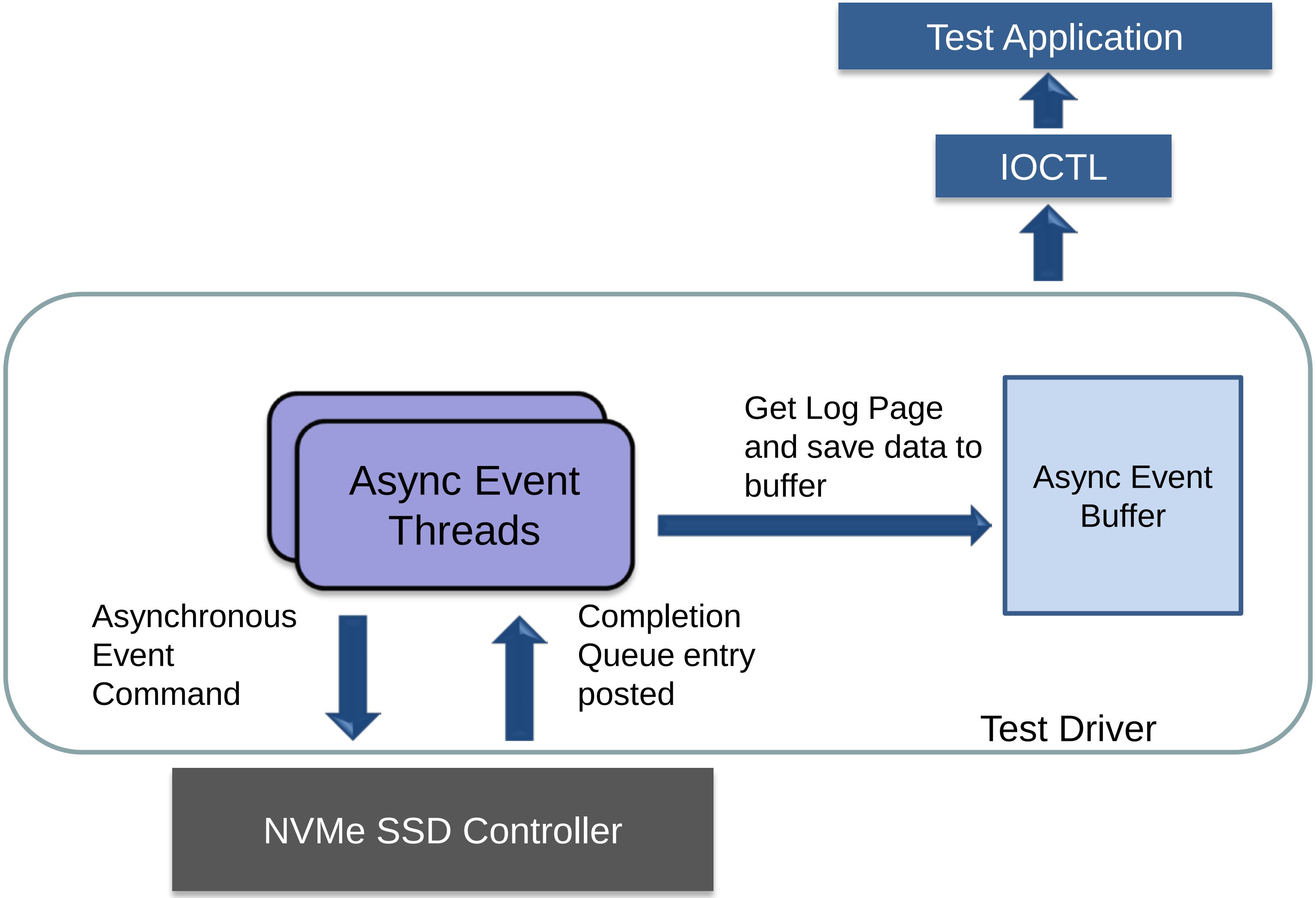
- Asynchronous events are grouped into event types. The following event types are defined:
  - Error
  - SMART / Health Status
  - I/O Command Set
  - Vendor Specific

- NVMe driver in the kernel does not support NVMe Asynchronous events.

Solution: Asynchronous Event Validation

- The Asynchronous Event Request command is submitted via event threads created in the driver to enable the reporting of asynchronous events from the controller.
- The event threads get blocked until there is an asynchronous event which controller wants to report.
- The controller posts a completion queue entry for this command when there is an asynchronous event. This has information about associated log page and event type.
- This unblocks a event thread which then sends Get Log Page command to get the details of event and stores it in a local buffer
- The thread again send asynchronous event request command to the device and gets blocked. This repeats in a cyclic way.
- The test application sends an IOCTL to read the event data stored in the buffer as and when required.

Solution: Asynchronous Event Validation



Test Application

IOCTL

Async Event Threads

Get Log Page and save data to buffer

Async Event Buffer

Asynchronous Event Command

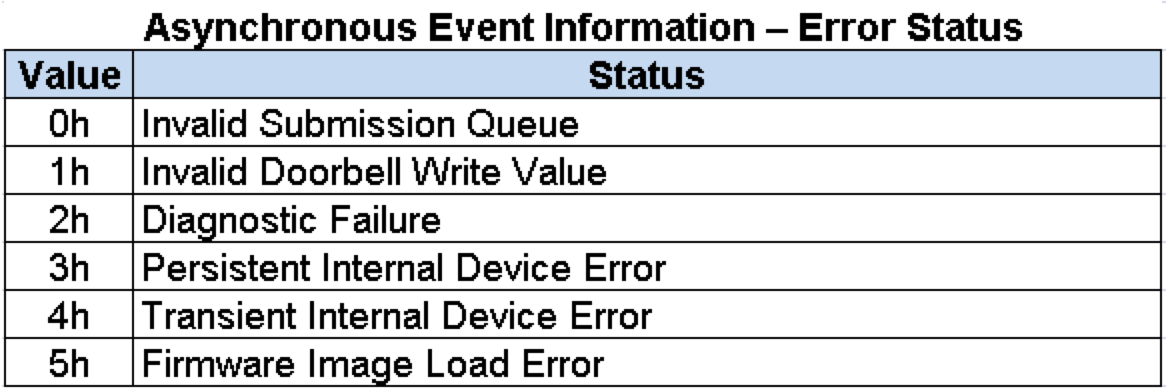
Completion Queue entry posted

Test Driver

NVMe SSD Controller

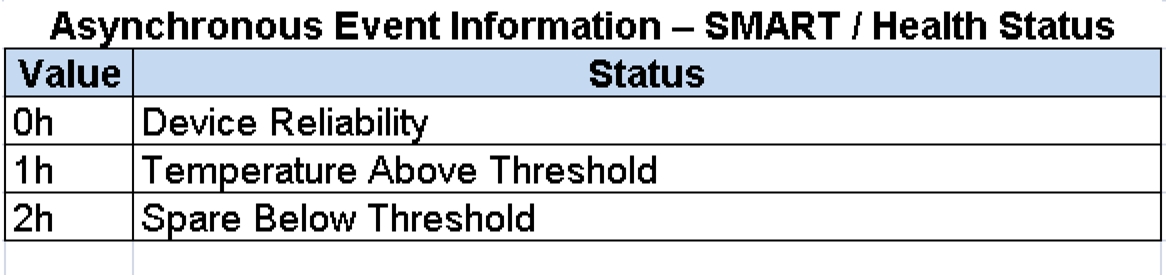
Example: Asynchronous Event Validation

- The following conditions are reported through asynchronous events and can be validated using the above solution.



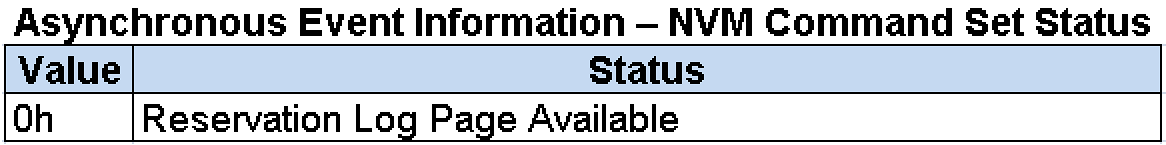
Asynchronous Event Information – Error Status

Value	Status
0h	Invalid Submission Queue
1h	Invalid Doorbell Write Value
2h	Diagnostic Failure
3h	Persistent Internal Device Error
4h	Transient Internal Device Error
5h	Firmware Image Load Error



Asynchronous Event Information – SMART / Health Status

Value	Status
0h	Device Reliability
1h	Temperature Above Threshold
2h	Spare Below Threshold



Asynchronous Event Information – NVM Command Set Status

Value	Status
0h	Reservation Log Page Available

Challenge: Queue Management

- For I/O queue management validation it is required to have a flexibility for creation and deletion of I/O queues on the fly.

- This helps in
  - Validating if the NVMe device is capable of handling multiple queues.
  - Performance testing with multiple queues

- The Linux NVMe Driver supports multiple queues which are created based on the number of cores available on the test machine.

- The challenge is to do queue management from the test application.

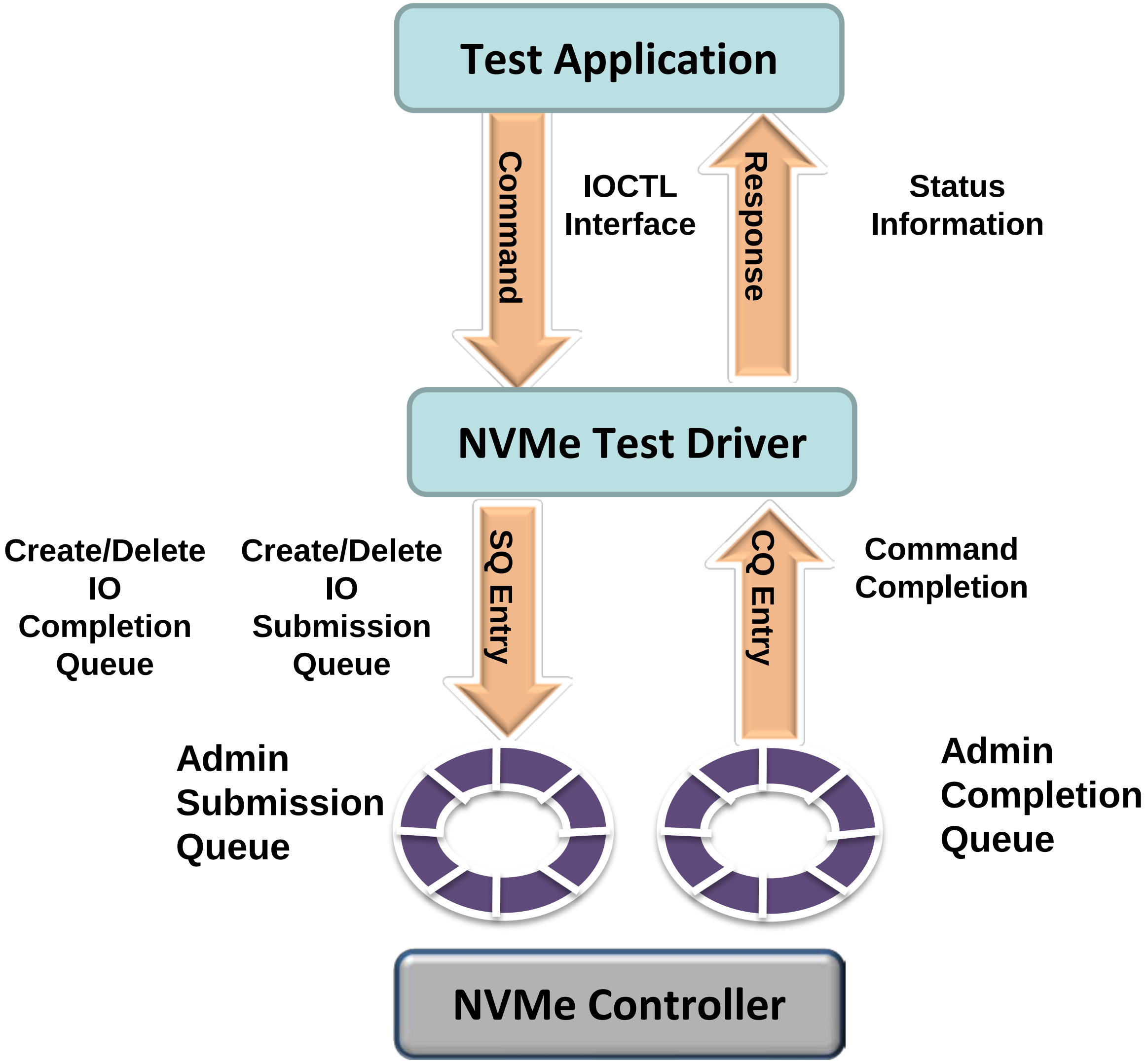
Solution: Queue Management

- Support added in the test application to send the following commands
  - Create I/O submission queue
  - Create I/O completion queue
  - Delete I/O submission queue
  - Delete I/O completion queue

- Linux NVMe driver modified to support queue management from test application

- This gives the flexibility to add and delete queues at runtime and validate multiple queues handling capabilities of the NVMe device

Solution : Queue Management



Challenge: Queue Depth Validation

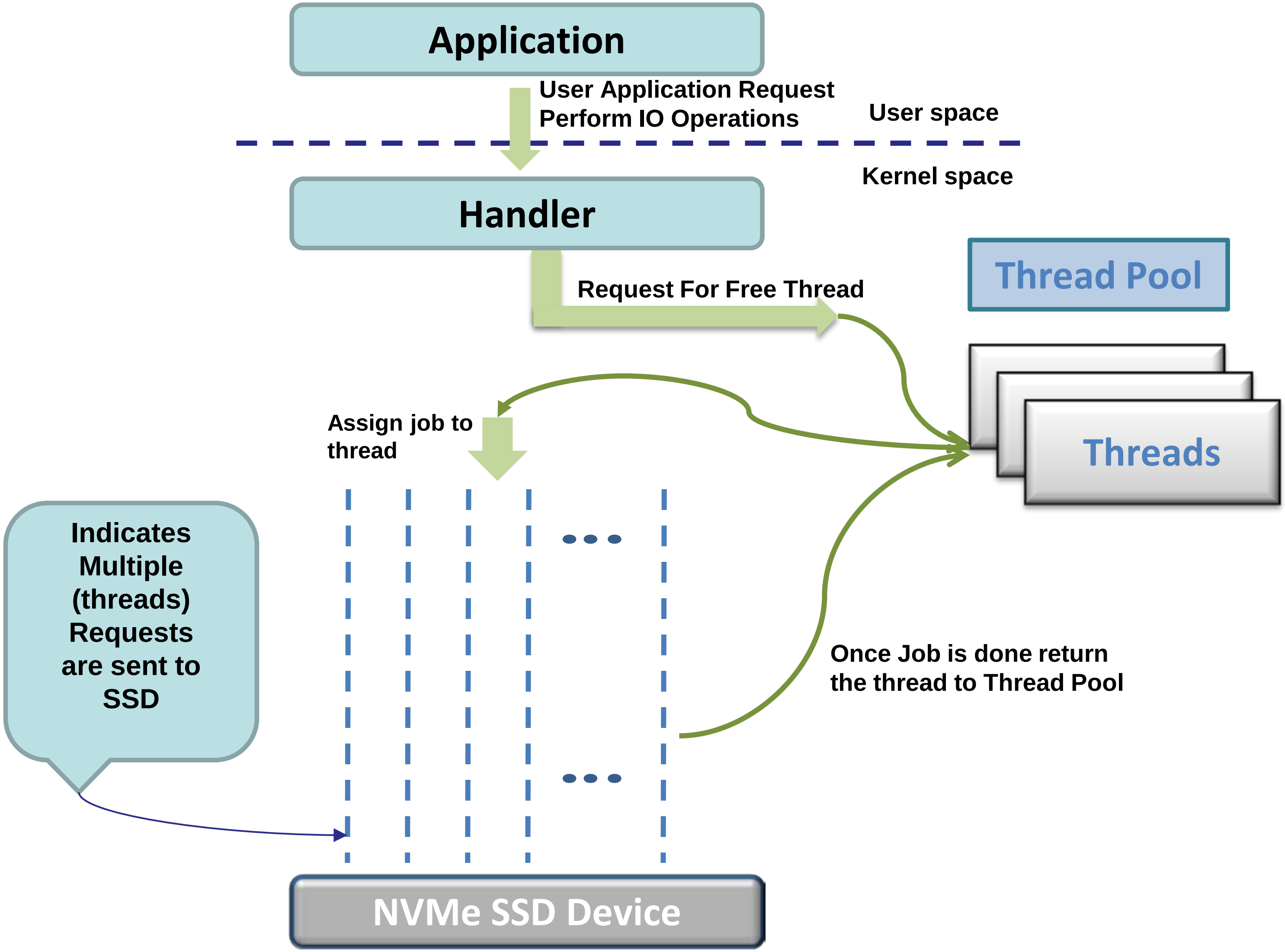
- The latency of PCIe SSD is very low, in the range of microseconds.
- The maximum queue depth is very high compared to AHCI(64K queues, 64K commands per queue)
- NVMe supports MSI-X
- The challenge is to achieve higher queue depths.

Solution: Queue Depth Validation

Creation of Thread Pool in Kernel space

- ❒ Test application delegates the work of performing I/O operations to kernel space (NVMe test driver).
- ❒ Multiple request submission is achieved using multiple kernel threads.
- ❒ A thread pool is created based on the queue depth value to save on thread creation time.
- ❒ A new thread is picked up from thread pool which sends a I/O request to the device.
- ❒ The thread waits for the response from the device and on job completion returns back to the thread pool.

Solution: Queue Depth Validation



Thank You!!