SDC 20

BY Developers FOR Developers

Storage Developer Conference
September 22-23, 2020

Introduction to libnvme

Keith Busch
Western Digital

Agenda

- NVM Express on Linux: Then and Now
- A brief tour of libnvme

NVM Express and Linux

- The NVMe protocol and Linux features supporting it have grown significantly since its humble beginnings
- Open source and specification standards allow common software solutions and compatibility across a broad spectrum of devices
  - Host driver development has generally converged to common sources
  - However, user space software often reimplements the same protocol's foundational core
- Goal: provide common and reusable open source repository for Linux NVMe management software

User Facing NVMe Protocol

NVMe 1.0 (2011)

SDC 20

Specifications

Transports

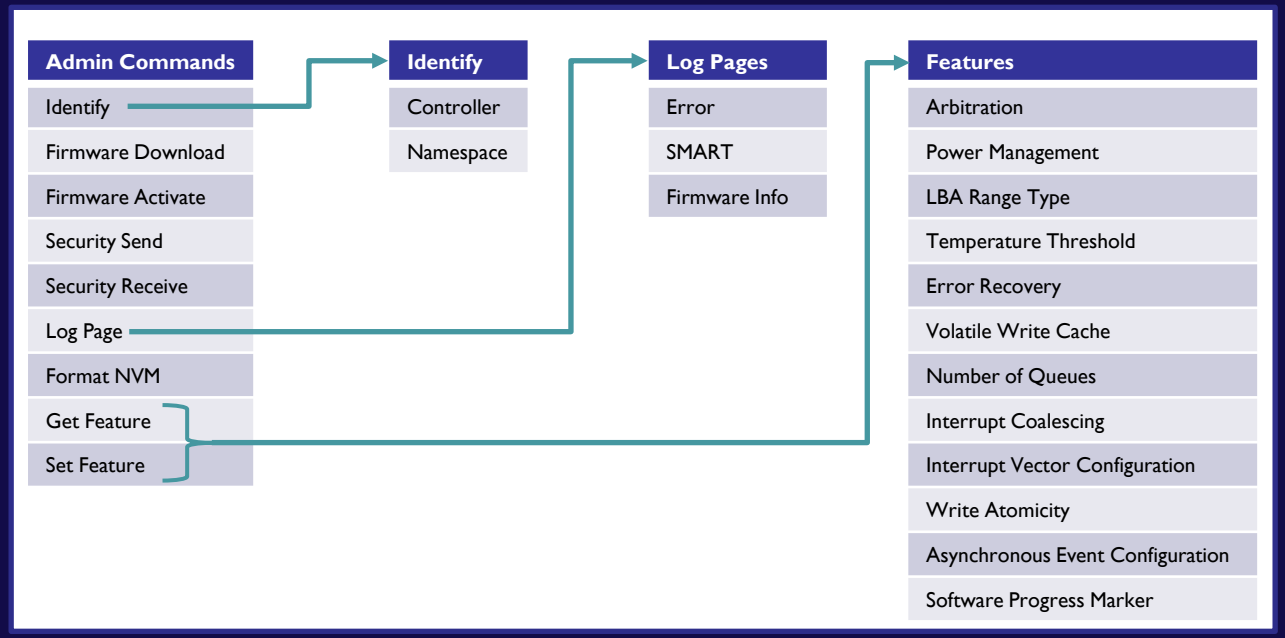
PCI EXPRESS

Commands

Admin Commands
Create IO Submission Queue
Create IO Completion Queue
Delete IO Submission Queue
Delete IO Completion Queue
Abort Command
Asynchronous Event Requests
Get Log Page
Identify
Get Feature
Set Feature
Firmware Download
Firmware Activate
Format NVM
Security Send
Security Receive

NVM Commands
Flush
Read
Write
Compare
Write Uncorrectable
Dataset Management

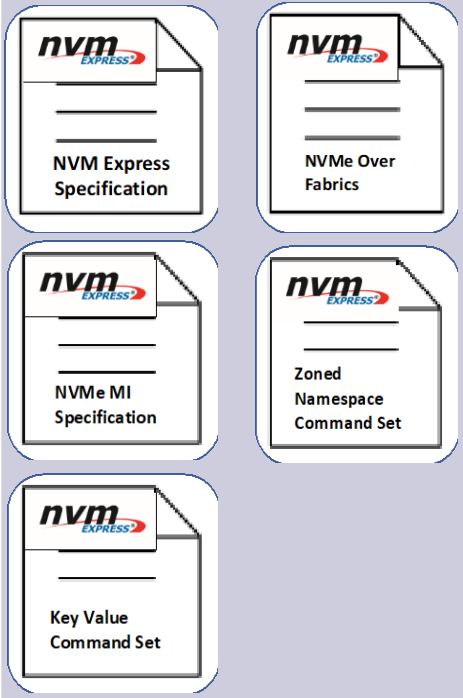
Command Subtypes



NVMe 1.4+ (today)

SDC 20

Specifications



Transports

PCI EXPRESS

RDMA

FIBRE CHANNEL

TCP

Other (loop)

Commands

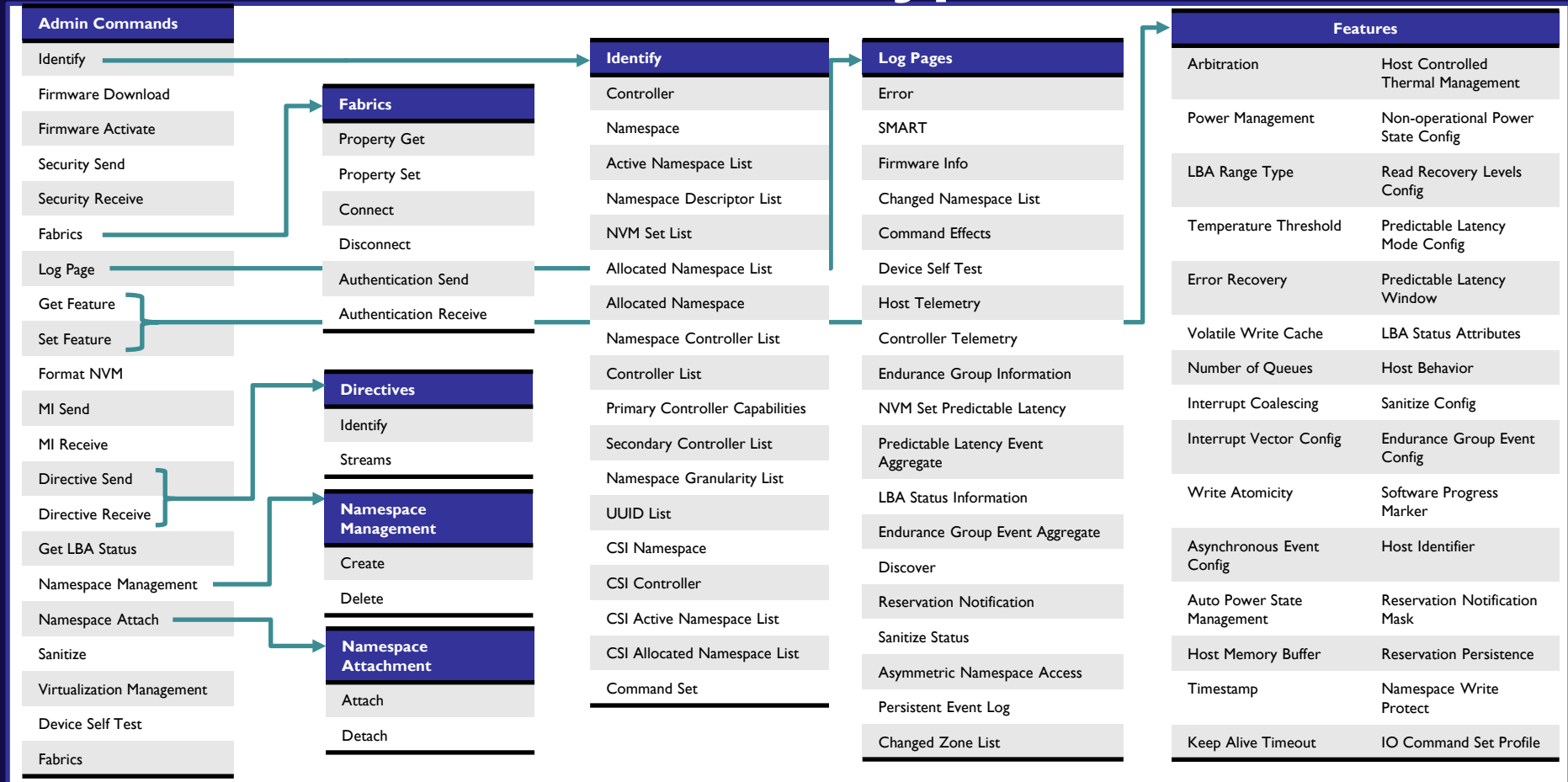
Admin Commands	
Identify	Directive Send
Firmware Download	Directive Receive
Firmware Commit	Get LBA Status
Security Send	Namespace Management
Security Receive	Namespace Attach
Log Page	Sanitize
Format NVM	Virtualization Management
Get Feature	Device Self Test
Set Feature	Fabrics
MI Send	Keep Alive
MI Receive	

NVM Commands
Flush
Read
Write
Compare
Write Uncorrectable
Dataset Management
Write Zeroes
Verify
Reservation Register
Reservation Acquire
Reservation Release
Reservation Report
Copy

ZNS Commands
Zone Management Send
Zone Management Receive
Zone Append

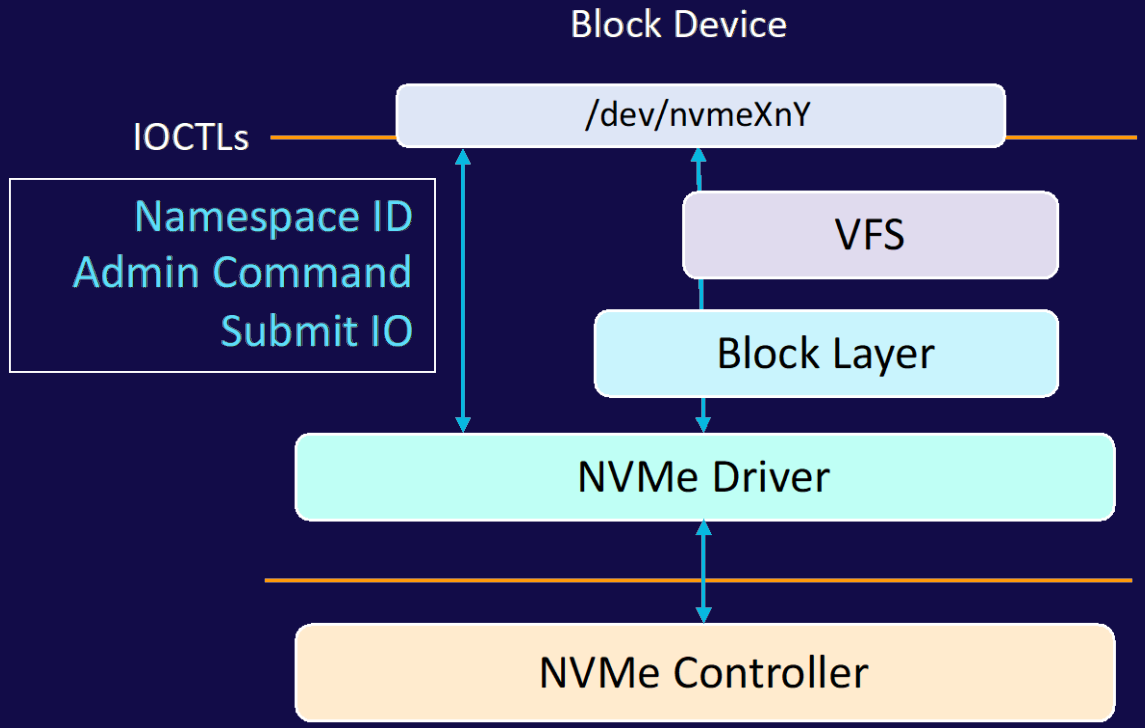
KV Commands
Store
Retrieve
Delete
Exist
List

Command Subtypes

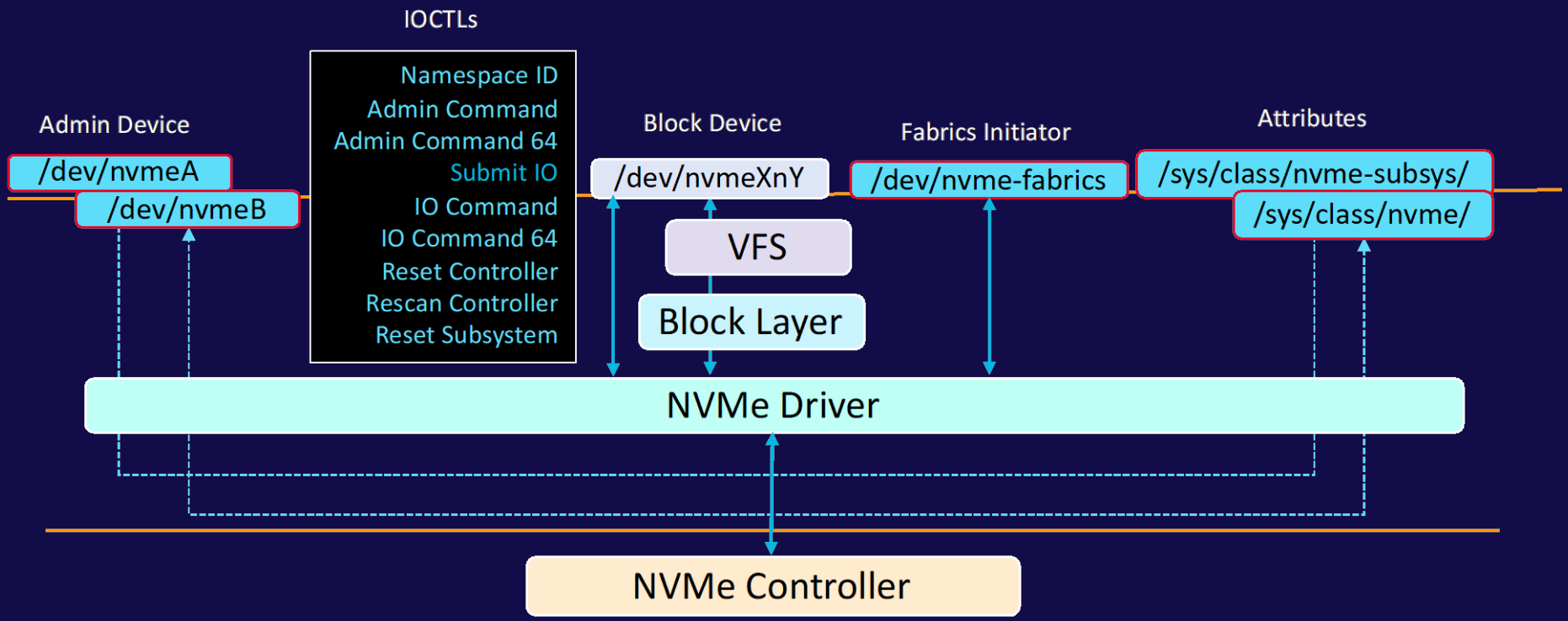


Linux NVM Express Driver

Initial Linux driver



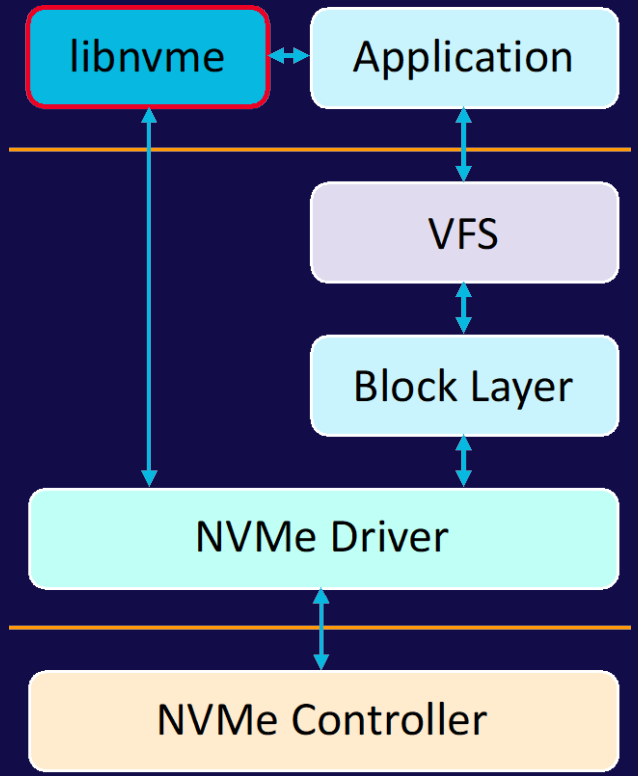
Linux NVMe Driver Interfaces Today



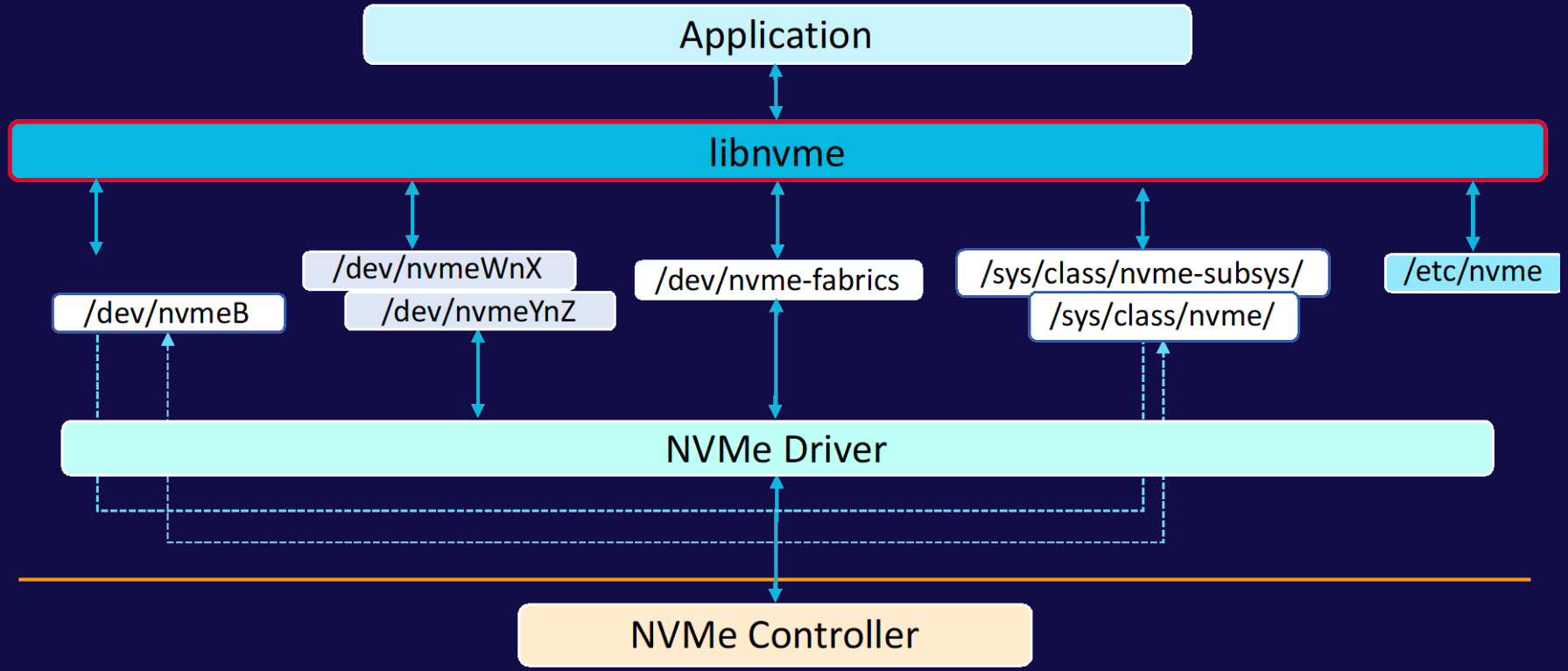
A tour of libnvme

libnvme: Where it fits

- A place for all NVMe features on Linux
- Works with the NVMe driver, not around it
  - Driver owns direct communication with the controller and command queueing
  - libnvme provides methods to scan devices enumerated by the driver, discover and connect to fabrics targets, dispatch arbitrary commands, and setup/decode data payloads



libnvme: all interfaces

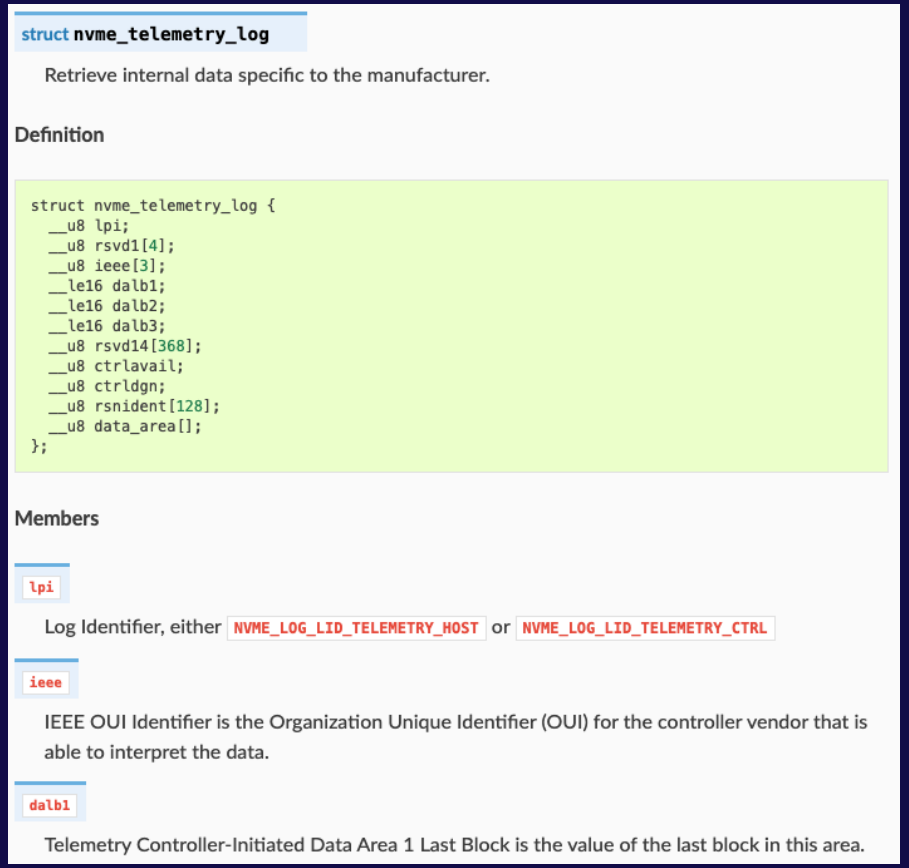


Current libnvme snapshot

- Source Code Repository: https://github.com/linux-nvme/libnvme
- Language C (with C++ compatibility)
- License: LGPL
- Code stats:
  - 20k lines of C
  - 250 exported functions
  - 200 enumerations defining 900 constants
  - 100 specification defined structures
- Install Artifacts:
  - Header files of API methods, structures, and enumerations
    - `#include <libnvme.h>`
  - Linkable objects: libnvme.a (static) and libnvme.so (dynamic)
  - Documentation in html and man pages

NVMe Base Types

- Defines all data structures, enumerations, fields and bitfields
- Updated to match the specification and ratified proposals
- All types are documented and cross linked to related functions, structures, and enumerations
- Example: Telemetry Log



struct **nvme_telemetry_log**

Retrieve internal data specific to the manufacturer.

Definition

```
struct nvme_telemetry_log {
  __u8 lpi;
  __u8 rsvd1[4];
  __u8 ieee[3];
  __le16 dalb1;
  __le16 dalb2;
  __le16 dalb3;
  __u8 rsvd14[368];
  __u8 ctrlavail;
  __u8 ctrldgn;
  __u8 rsnident[128];
  __u8 data_area[];
};
```

Members

`lpi`

Log Identifier, either `NVME_LOG_LID_TELEMETRY_HOST` or `NVME_LOG_LID_TELEMETRY_CTRL`

`ieee`

IEEE OUI Identifier is the Organization Unique Identifier (OUI) for the controller vendor that is able to interpret the data.

`dalb1`

Telemetry Controller-Initiated Data Area 1 Last Block is the value of the last block in this area.

Passthrough Commands

- Provides ioctl definitions for Linux kernel's user API
  - Except NVME_IOCTL_SUBMIT_IO: Kernel should deprecate that
- Flexible enough to send any NVMe command and payload, including future and vendor specific commands
  - Can't do fused operations: unsupported by kernel
- Abstract/generic parameters are not very coder friendly

```
int nvme_admin_passthru(int fd, __u8 opcode, __u8 flags, __u16 rsvd,
                        __u32 nsid, __u32 cdw2, __u32 cdw3, __u32 cdw10, __u32 cdw11,
                        __u32 cdw12, __u32 cdw13, __u32 cdw14, __u32 cdw15,
                        __u32 data_len, void *data, __u32 metadata_len, void *metadata,
                        __u32 timeout_ms, __u32 *result);
```

Passthrough Commands

- NVMe opcode type parameterized with fields and types specific to that command
  - More coder friendly than bits and bytes in generic parameters
  - Provides additional type safety
- The library does not provide operations disruptive to driver
  - Examples include: Create/Delete SQ/CQ, Abort, Asynchronous Event Notification, Keep Alive, Connect/Disconnect
  - You can use the library to construct such commands anyway, but you will probably not be happy with the results!

Command Example: Get Log

SDC 20

int **nvme_get_log**(int *fd*, enumnvme_cmd_get_log_lid *lid*, __u32 *nsid*, __u64 *lpo*, __u8 *lsp*, __u16 *lsi*, bool *rae*, __u8 *uuidx*, __u32 *len*, void * *log*)

NVMe Admin Get Log command

Parameters

`int fd`

File descriptor of nvme device

`enum nvme_cmd_get_log_lid lid`

Log page identifier, see `enum nvme_cmd_get_log_lid` for known values

`__u32 nsid`

Namespace identifier, if applicable

`__u64 lpo`

Log page offset for partial log transfers

`__u8 lsp`

Log specific field

Passthrough Commands

- Many commands have variable parameters based on command specific dwords
  - APIs provided for for all variations of the following opcodes: Get Log, Identify, Get/Set Features, Get/Set Directives
- Currently defines 150 functions

Example: Controller Telemetry Log

SDC 20

int **nvme_get_log_telemetry_ctrl**(int *fd*, bool *rae*, __u64 *offset*, __u32 *len*, void * *log*)

Parameters

`int fd`

File descriptor of nvme device

`bool rae`

Retain asynchronous events

`__u64 offset`

Offset into the telemetry data

`__u32 len`

Length of provided user buffer to hold the log data in bytes

`void * log`

User address for log page data

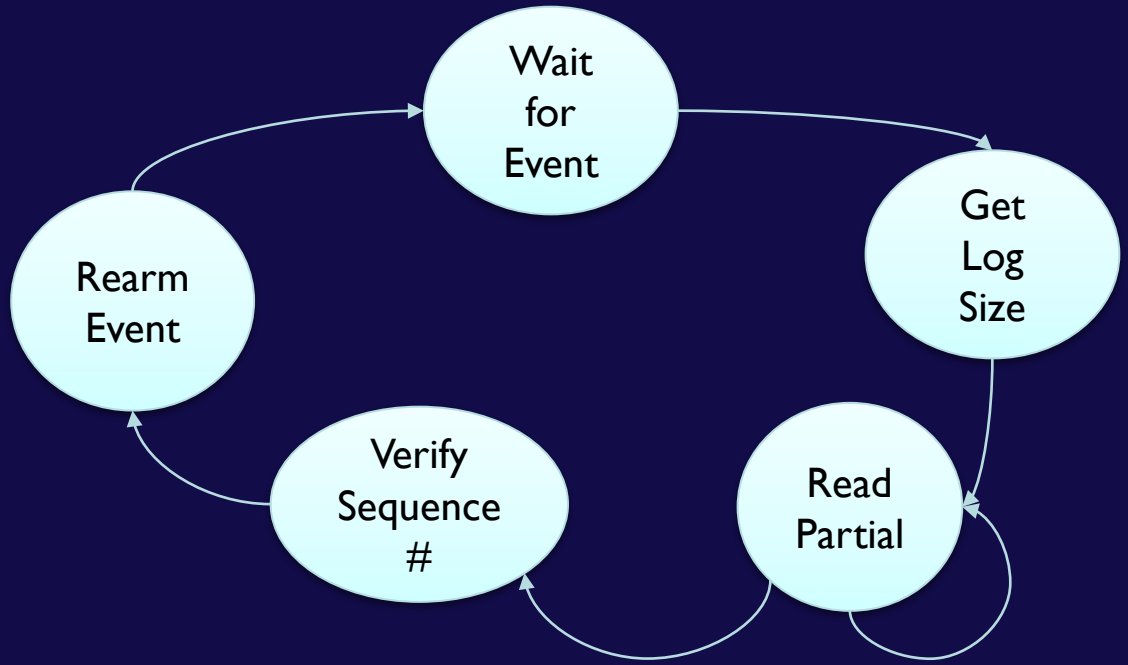
Multi-Step commands

SDC 20

- Some commands require multiple steps to successfully transfer data
  - Ex: Log pages, firmware download
- An incorrect sequence may end up sending or receiving incomplete or incorrect data
- libnvme provides helper functions to manage some multi-step sequences

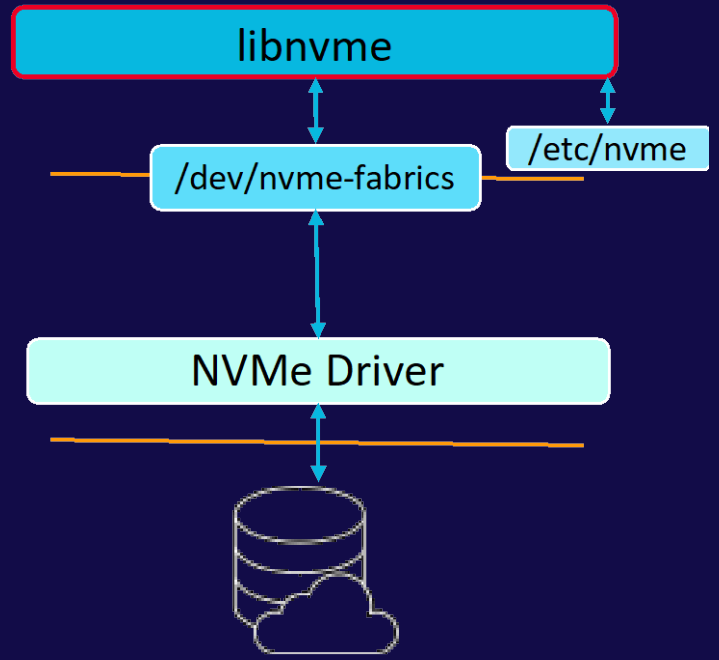
Example: Controller Telemetry Log

```
int nvme_get_ctrl_telemetry(int fd, struct nvme_telemetry_log **log);
```



Fabrics

- The kernel provides a special device handle to initiate connections to NVMeOF targets
  - Kernel's UAPI: Write magic strings
- libnvme helps:
  - Generate and submit the magic strings
  - Recursively discover and connect to targets
  - Decoding host and target configuration files in /etc/nvme/
- Synchronizes with driver's fabric interface



sysfs

- libnvme provides methods to scan sysfs to establish the nvme topology
  - Enumerate, search, and filter the topology, retrieve attributes, and submit commands
  - Filtering examples:
    - Find all controllers from vendor “ACME”
    - Find all RDMA controllers
- Ongoing maintenance synchronizes with driver’s sysfs interface as needed

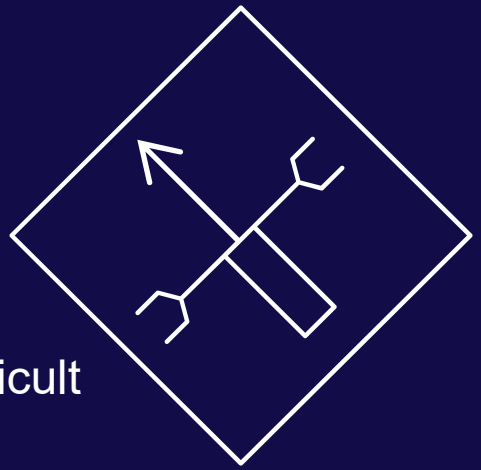
```
|-- nvme-subsys5 - NQN=testnqn
|   |-- nvme5n1 lba size:512 lba max:488397168
|   |-- nvme7 loop  live
|   |   `-- nvme5c7n1 optimized
|   |-- nvme6 loop  live
|   |   `-- nvme5c6n1 optimized
`-- nvme-subsys0 - NQN=nqn.2018-01.com.wdc:nguid:19190E802017-0001-001B448B44AF756C
    `-- nvme0 pcie 0000:09:00.0 live
        `-- nvme0n1 lba size:512 lba max:488397168
```

IO Example: FIO libnvme ioengine?!

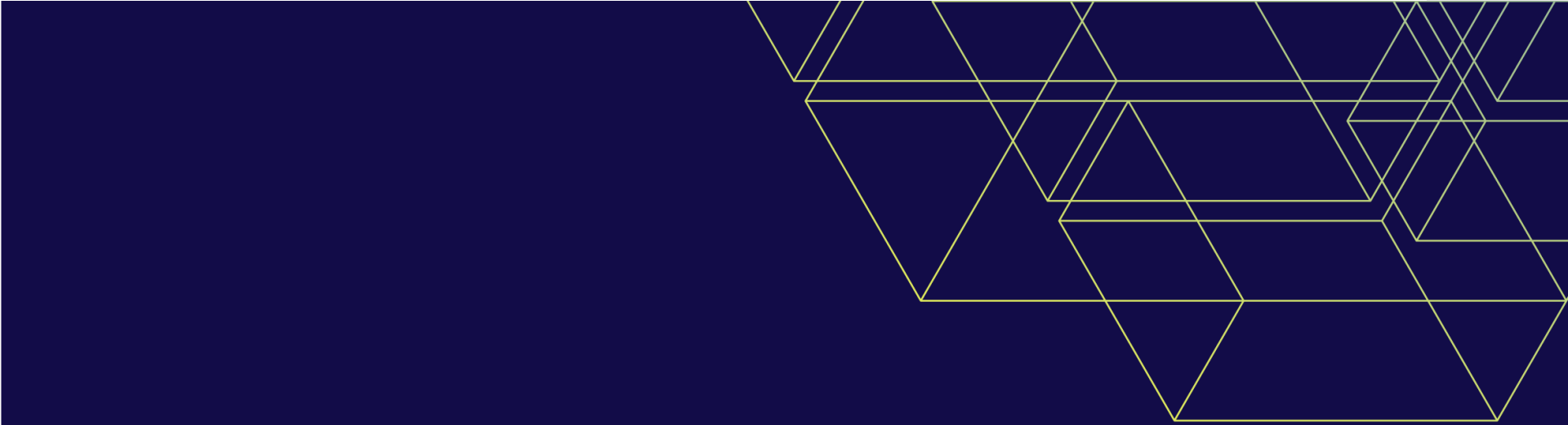
```
#include <libnvme.h>
#include "../fio.h"

static int nvme_open_file(struct thread_data *td, struct fio_file *f) {
            nvme_ns_t n = nvme_ns_open(f->file_name);
            f->fd = nvme_ns_get_fd(n);
            f->engine_data = n;
            return 0;
}

static enum fio_q_status nvme_queue(struct thread_data *td, struct io_u *io_u) {
            struct fio_file *f = io_u->file;
            nvme_ns_t n = f->engine_data;

            switch (io_u->ddir) {
                        case DDIR_READ:
                                    io_u->error = nvme_ns_read(n, io_u->xfer_buf, io_u->offset, io_u->xfer_buflen);
                                    break;
                        case DDIR_WRITE:
                                    io_u->error = nvme_ns_write(n, io_u->xfer_buf, io_u->offset, io_u->xfer_buflen);
                                    break;
            }
            return FIO_Q_COMPLETED;
}
```

Remaining Work

- A few specification features remain to be implemented
  - Persistent event log
  - Management Interface
  - Key/Value Command Set
- Some features are still untested
  - Need to verify with real hardware
  - Finding target support for some features remains difficult
- New NVMe specifications feature are always coming!
- Complete integration with nvme-cli
  - Release and request package support with Linux distributions



Thank You!



Please take a moment
to rate this session.

Your feedback matters to us.