NVMe® Zoned Namespace SSDs & The Zoned Storage Linux® Software Ecosystem

Sponsored by NVM Express® organization, the owner of NVMe®, NVMe-oF™ and NVMe-MI™ standards

Speakers

Javier González
Principal Software Engineer, SSDR R&D Center Lead at Samsung Electronics

Damien Le Moal
Director, System Software Group at Western Digital

Agenda

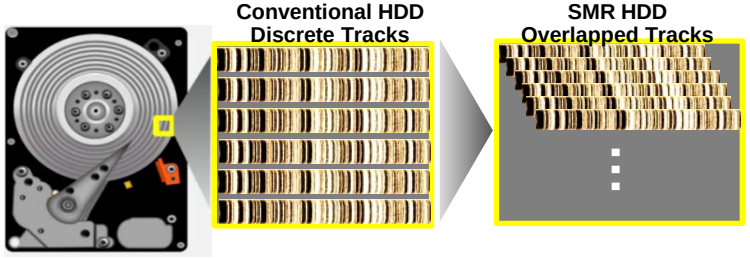
- Zoned Storage and NVMe® Zoned Namespaces
  - Specifications overview
- Linux Kernel Zoned Block Device Support
  - Zoned Block Device abstraction
  - Support history, key features and user API
- Applications zoned block device support
  - Legacy applications: File systems and device mapper
  - Optimized applications: zonefs, libzbd, libnvme, xNVME, SPDK
- Tools
  - QEMU® ZNS device emulation, nvme-cli, blkzone, fio
- Example applications
  - RocksDB, Apache Hadoop® / HDFS
- Summary

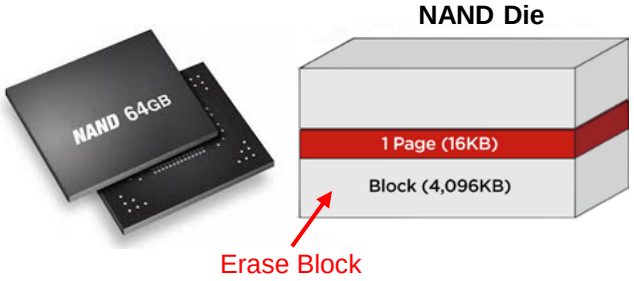
Zoned Storage and NVMe® Zoned Namespaces

New storage paradigm

Why Zoned Storage?

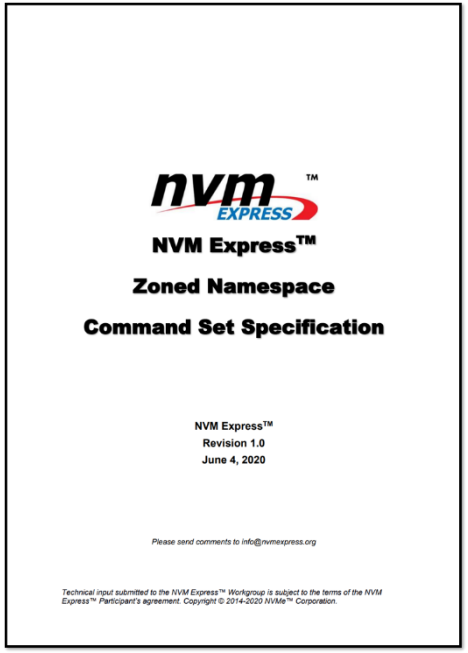
- Lower storage cost ($/TB)
  - HDDs: increased capacity with shingled magnetic recording
    - Zones of overlapped tracks
  - SSDs: reduced OP and device resources
    - E.g. DRAM
- Allowing the host to contribute to data placement simplifies the device implementation
  - Expose sequential write constraint of media
    - Zones of overlapped tracks on SMR HDDs
    - NAND die erase blocks on SSDs
- And leads to device side optimizations
  - Improved device capacity
    - Lower OP
  - Remove device data movement (GC)
    - Improved latency and throughput
  - Better support for multi-tenant workloads
    - Host data-placement





Specifications

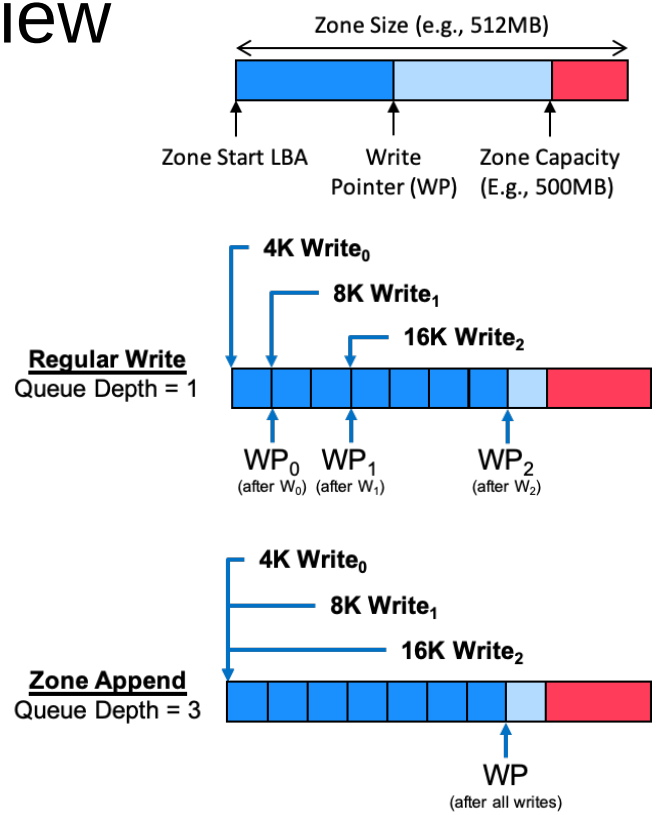
- The device storage LBA space is divided into zones with sequential write constraint
- First standardized for SMR hard-disks with ZBC for SCSI and ZAC for ATA
  - Two zone models defined
    - Host managed: sequential writes in zones are mandatory
    - Host aware: sequential writes are recommended but not mandatory
      - Random writes are accepted
- NVMe® Zoned Namespaces (ZNS) specifications define a similar interface for SSDs
  - Technical Proposal 4053 (Core ZNS specifications)
  - Technical Proposal 4056 (Namespace Types)
  - New proposals ongoing



https://nvmexpress.org/developers/nvme-specification/
(Available in the 1.4 TP package)

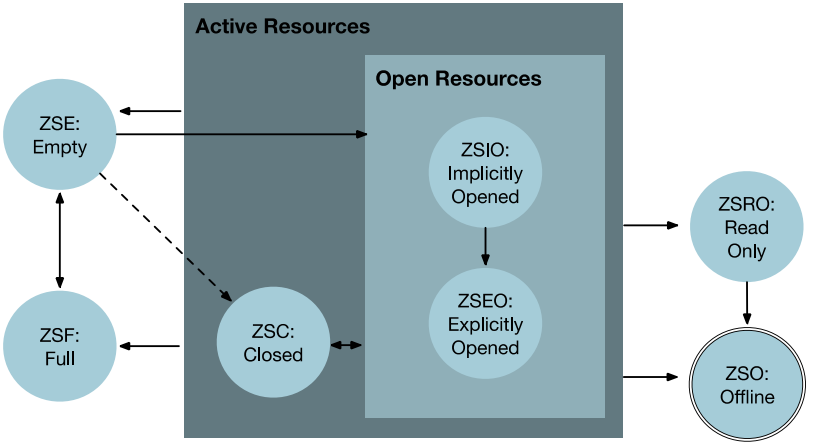
NVMe® Zoned Namespace Overview

- New “Zoned Namespace” namespace type
- Inherits the NVM Command Set
  - i.e., Read/Write/Flush commands are available
- Defines zones of fixed size to be written sequentially
  - Matches NAND media write characteristics
  - Support for different zone size configurations
  - Various schemes can be used by the drive to implements zone mapping to erase blocks
    - No scheme defined by the specifications
    - Depends on drive vendor and drive model
- Adds zone writeable capacity (zone capacity)
  - Allows the zone size to be a power of 2 number of blocks while allowing internal alignment of zones to erase blocks
- Two write methods: Regular Writes and Zone Append



NVMe® Zoned Namespace Overview

- Zone States: empty, implicitly opened, explicitly opened, closed, full, read-only, and offline
  - State transitions on writes, zone management commands, and device resets
- Zone management commands
  - Report zones, Open, Close, Finish and Reset
- Active and Open Resources associated to a zone
  - Limits on the maximum number of active and open zones
- Main differences with ZBC/ZAC
  - Zone capacity and active zones
  - Zone append command
  - Mostly compatible with ZBC/ZAC **host managed** zone model

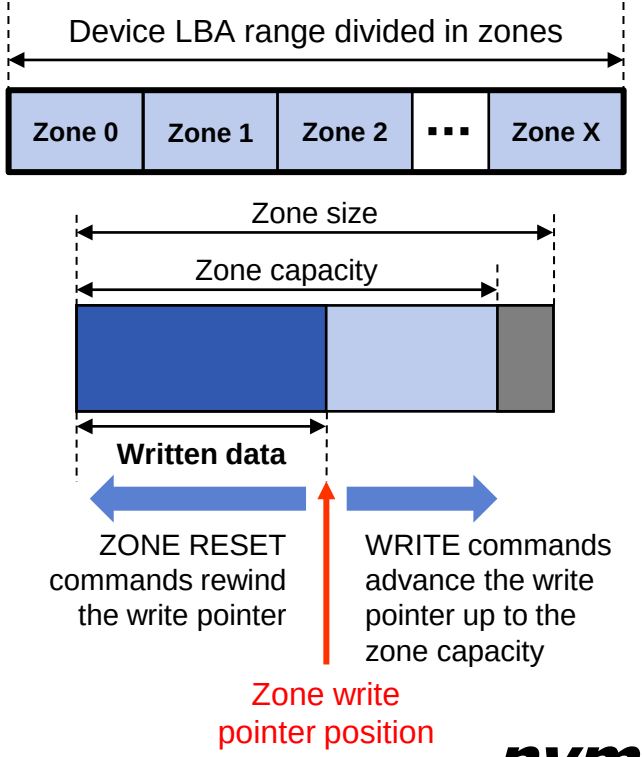


Linux® Kernel Zoned Block Device Support

History, key features, user API and applications support

Linux® Zoned Block Device Abstraction

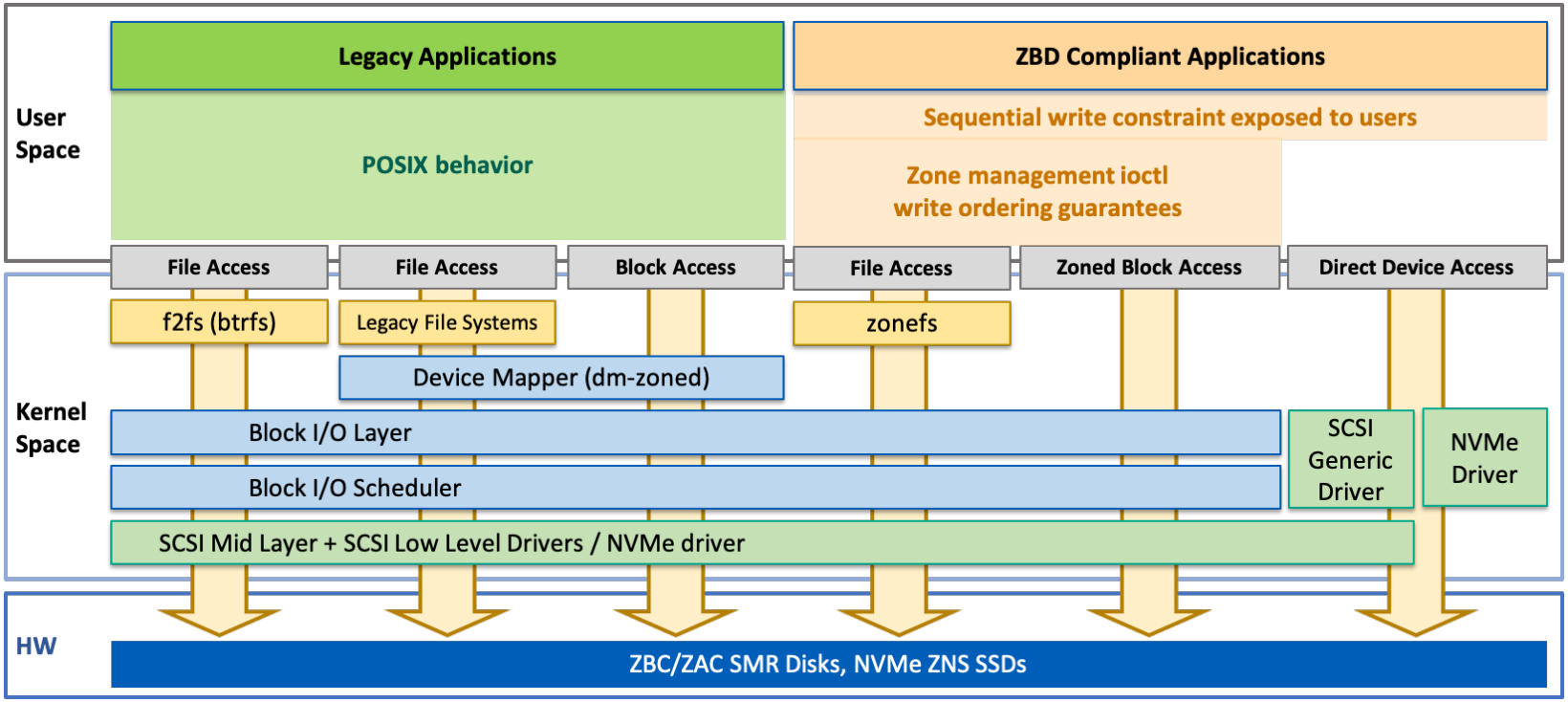
- Generic abstraction for ZBC, ZAC and ZNS devices
  - Zone types as defined by the standards
    - Conventional
      - Accept random writes, optional with SMR
    - Sequential write required
      - host-managed model only
      - SMR and ZNS
    - Sequential write preferred
      - host-aware model only
      - SMR only
  - A zone may have a useable capacity lower than the zone size (ZNS only)
- Some restrictions are added
  - All zones must have the same size
  - The zone size must be a power of 2 number of LBAs



Support History

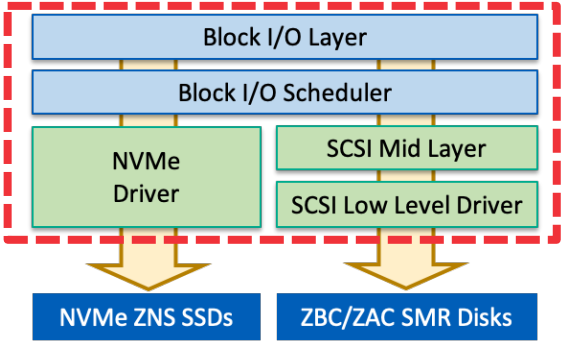
- ZBD support was introduced in 2017 with kernel version 4.10
  - For SMR disks
  - Initial release included native f2fs support
- Following kernel versions added more features
  - Device mapper support with kernel 4.13
    - dm-zoned target to abstract host managed disks as regular disks
  - Various improvements to the kernel internal API and user API
- Latest kernel version 5.9 (release candidate state) will introduce NVMe® ZNS
  - Stable release scheduled for beginning of October 2020
- Development is still on-going
  - ZBD support for *dm-crypt* added to kernel 5.9
  - btrfs native support implementation is on-going

Current Kernel Support Status



Key Features

- Provides kernel-internal and user API (ioctl commands) for all zone management operations
  - Zone report, reset, open, close and finish
- Zone append writes supported in-kernel only
  - User API is being discussed
- Provides regular write command ordering guarantees
  - Does *not* reorder writes into sequential write streams
  - Direct I/O writes (O_DIRECT) only for users
    - No guarantees of ordered page writeback for buffered writes
  - Implemented using the mq-deadline block IO scheduler
    - Using a zone write locking mechanism, resulting in at most one write per zone at any time (per specification)
- Supported devices:
  - Devices following Linux ZBD zone configuration constraints
  - ZNS namespaces with native zone append support



User API

- *ioctl()* commands for zone report, reset, open, close and finish operations are defined in *include/uapi/linux/blkzoned.h* in the kernel tree
  - Installed as */usr/include/linux/blkzoned.h*
- Report provides all zone information
  - Zone type, start sector, size and capacity
  - Zone condition and write pointer position
  - Zone flags
    - Reset recommended etc
- sysfs attribute files also available
  - Number of zones, zone size, etc

```
/**
 * Zoned block device ioctl's:
 *
 * @BLKREPORTZONE: Get zone information. Takes a zone report as argument.
 *                 The zone report will start from the zone containing the
 *                 sector specified in the report request structure.
 * @BLKRESETZONE: Reset the write pointer of the zones in the specified
 *                sector range. The sector range must be zone aligned.
 * @BLKGETZONESZ: Get the device zone size in number of 512 B sectors.
 * @BLKGETNRZONES: Get the total number of zones of the device.
 * @BLKOPENZONE: Open the zones in the specified sector range.
 *               The 512 B sector range must be zone aligned.
 * @BLKCLOSEZONE: Close the zones in the specified sector range.
 *                The 512 B sector range must be zone aligned.
 * @BLKFINISHZONE: Mark the zones as full in the specified sector range.
 *                 The 512 B sector range must be zone aligned.
 */
#define BLKREPORTZONE   _IOWR(0x12, 130, struct blk_zone_report)
#define BLKRESETZONE    _IOW(0x12, 131, struct blk_zone_range)
#define BLKGETZONESZ    _IOR(0x12, 132, __u32)
#define BLKGETNRZONES   _IOR(0x12, 133, __u32)
#define BLKOPENZONE     _IOW(0x12, 134, struct blk_zone_range)
#define BLKCLOSEZONE    _IOW(0x12, 135, struct blk_zone_range)
#define BLKFINISHZONE   _IOW(0x12, 136, struct blk_zone_range)
```

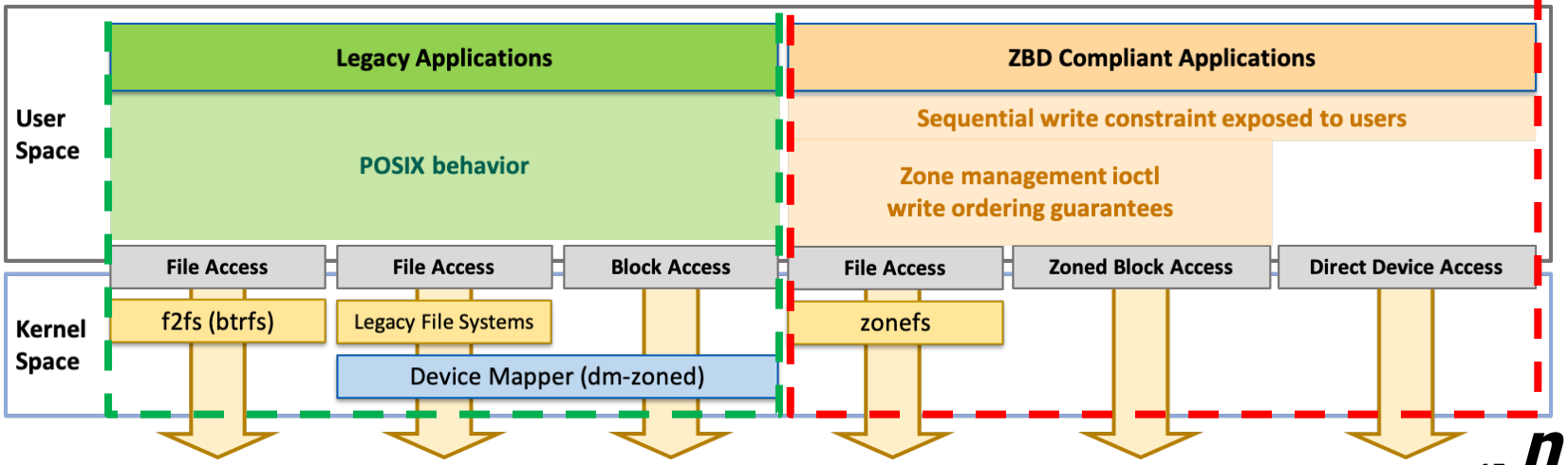
Applications Zoned Block Device Support

Regular File Interface

- Legacy applications can be used as-is
  - Sequential write constraint handled by filesystem or device mapper
- But performance may not be optimal
  - No workload-based optimization possible

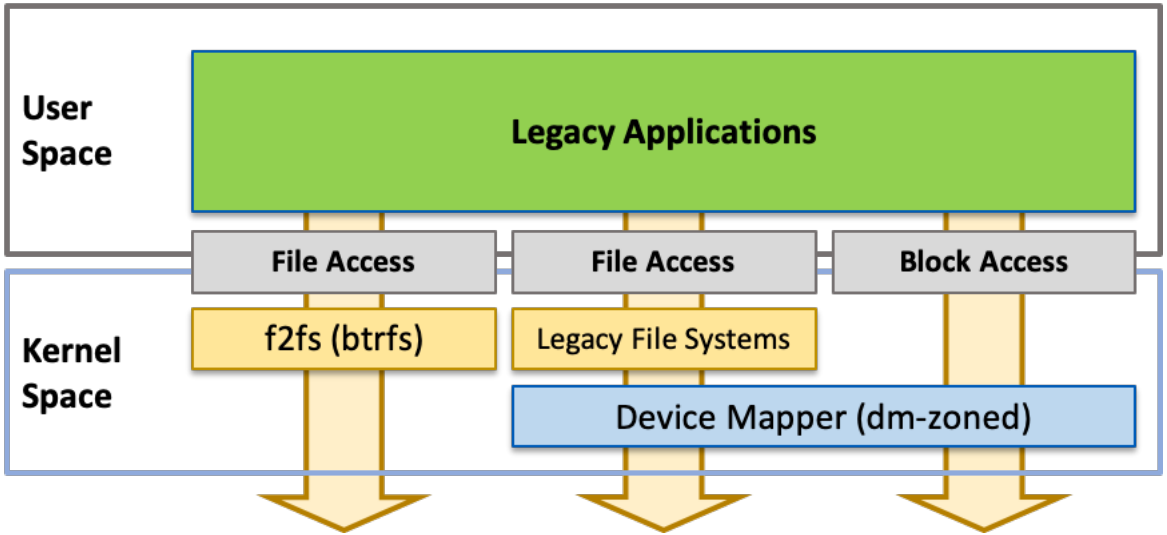
Raw block Interface

- Applications need (potentially difficult) modifications
  - Sequential writes and zone management mandatory
- But performance can be optimized
  - Based on the application access pattern



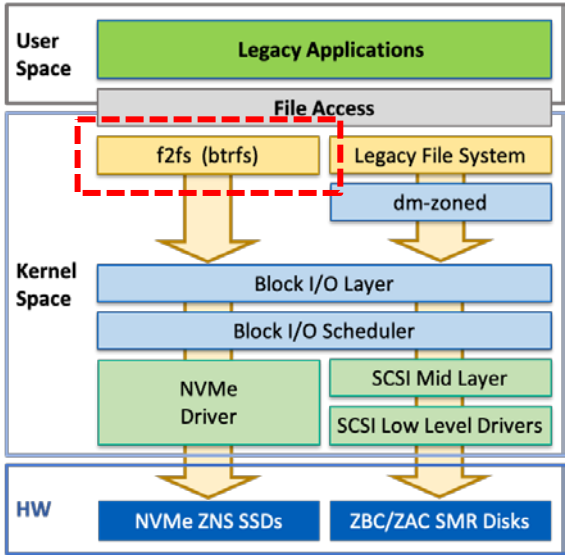
Legacy Applications

Kernel components handle ZBD sequential write constraint and zone management



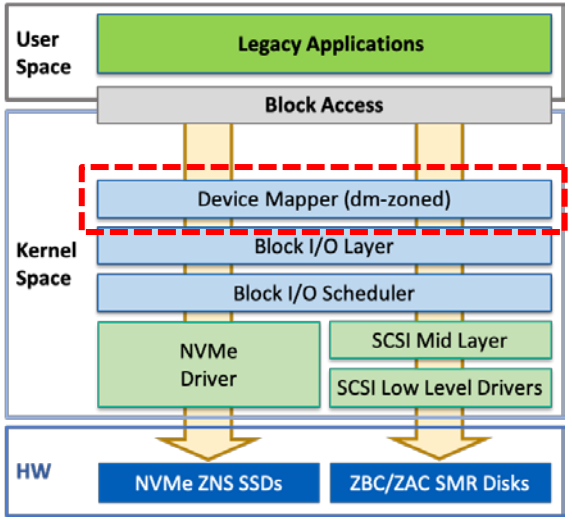
Native File System Support

- POSIX compliant file system support enables legacy applications
  - No changes required: any application can work as is
- f2fs
  - Relies on conventional zones for fixed place metadata blocks
  - NVMe® ZNS devices can be supported using a secondary device for metadata
- btrfs support in development
  - Targeting kernel version 5.11 upstream release (March 2021)
  - Uses zone append writes for data blocks
  - Does not require any conventional zones
  - 100% copy-on-write operation compatible with NVMe ZNS



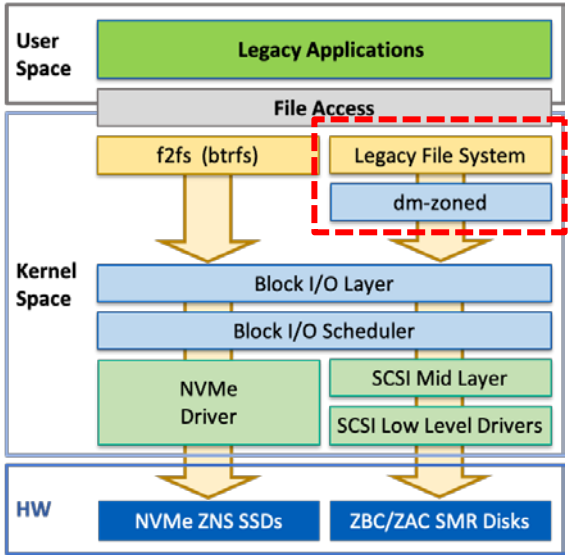
Regular Block Device Interface

- Applications relying of raw block device accesses can use the *dm-zoned* device mapper
  - Requires conventional zones for internal metadata and random write buffering
  - Conventional zones can be replaced by a regular SSD
    - Mandatory for ZNS backing devices
- Depending on the workload and setup, performance degradation may appear due to zone garbage collection
  - Use of cache device greatly improves performance
- *dm-zoned* user space tool available on github
  - https://github.com/hgst/dm-zoned-tools
  - Documentation at https://zonedstorage.io/linux/dm/#dm-zoned



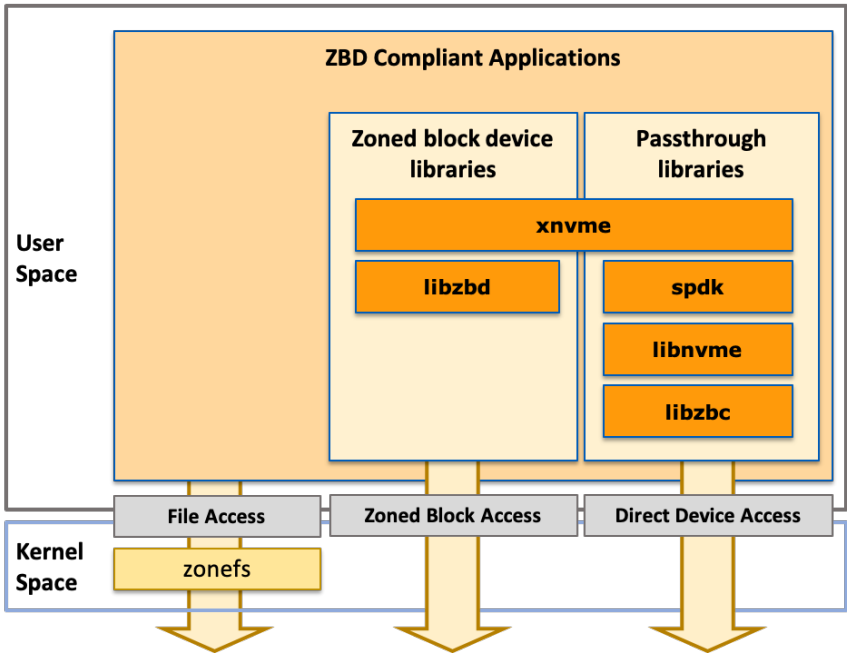
Legacy File Systems Support

- Legacy file systems (e.g. ext4 or XFS) can be enabled with the *dm-zoned* device mapper target
- Depending on the workload and setup, performance degradation may appear due to zone garbage collection
  - Weak dependency on application workload
  - Use of cache device greatly reduces the impact of write buffer zone reclaim



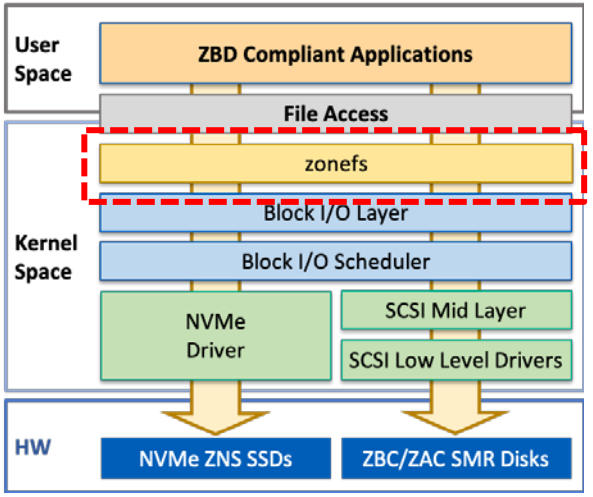
Zoned Block Device Compliant Applications

Zoned Block Device constraints exposed to the user application as-is



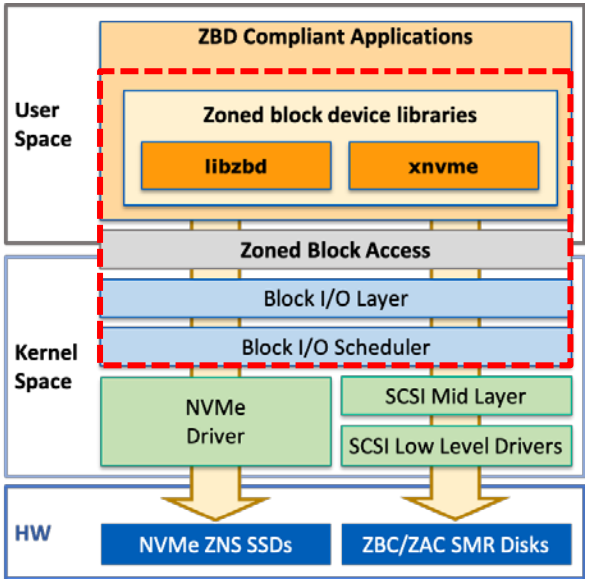
Zones as Files: zonefs

- Simplify application programming by using regular file system calls for zone access and management
  - Exposes each zone of device as a regular file
    - But random writes not possible: each zone file must be written sequentially using direct I/Os
  - Zone commands automatically executed as needed with file system calls
    - E.g. file truncate() implies a zone reset command
  - Enables different programming language bindings
    - E.g. JAVA, Python
- Zone information from the device is used as metadata
  - No journaling overhead
  - Static file tree structure representing device zones
  - Zone write pointer indicates the file size
- User space tools available on github
  - https://github.com/damien-lemoal/zonefs-tools



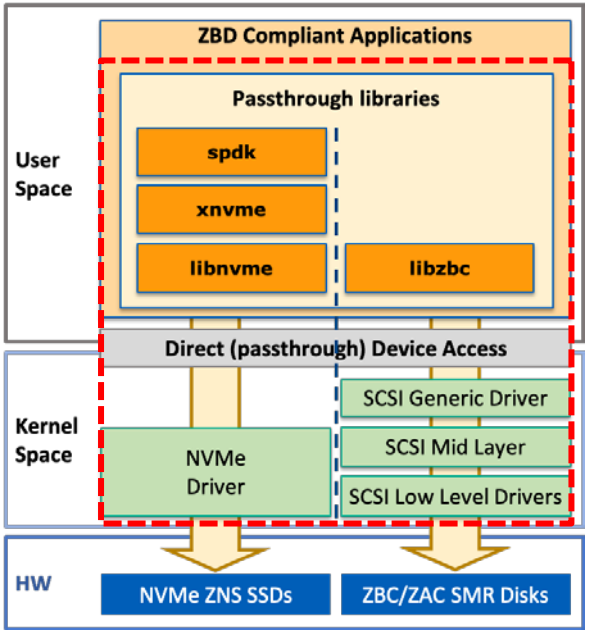
Zoned Block Accesses

- Device sequential write constraint, zone state machine, etc are all exposed as-is to the application
- Relies on kernel *ioctl()* user API for zone management operations
  - Synchronous and asynchronous I/O usable with regular system calls
- Restrictions:
  - Direct I/O (O_DIRECT) writes only
    - Buffered reads are OK
- **libzbd** is available to simplify application programming
  - https://github.com/westerndigitalcorporation/libzbd
  - Supports all zoned block device types
- **xNVME** can also be used
  - More details in following slides



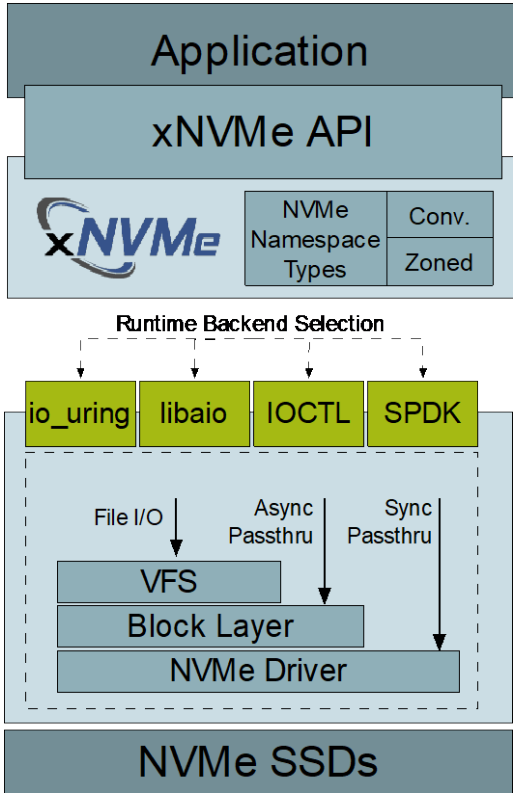
Passthrough Device Accesses

- Enable zone-aware applications with old kernels or unsupported devices
  - Also enables features not supported in Linux
- **libnvme**: NVMe® driver IOCTLs
  - Base library for nvme-cli utility
  - Synchronous passthrough using existing kernel interfaces
  - Link: https://github.com/linux-nvme/libnvme
- **SPDK**: user space NVMe® driver
  - Support for ZNS Command Set (Oct 2020)
  - Link: https://github.com/spdk/spdk
- **libzbc**: For SMR hard-disks
  - Supports ATA / ZAC and SCSI / ZBC devices
  - Link: https://github.com/hgst/libzbc



Passthrough Device Accesses

- **xNVMe:** User-space NVMe® API (https://xnvme.io)
- Provides a common API for different I/O backends
  - Synchronous and asynchronous
  - In-kernel path
    - io_uring, libaio, block device *ioctl()*
    - Leverage ongoing work on io_uring passthru
  - User-space path
    - SPDK: Use bdev passthru and complement functionality
- Same API across OSs (FreeBSD, Windows)
- Backend selection at runtime
- Link: https://github.com/OpenMPDK/xNVMe

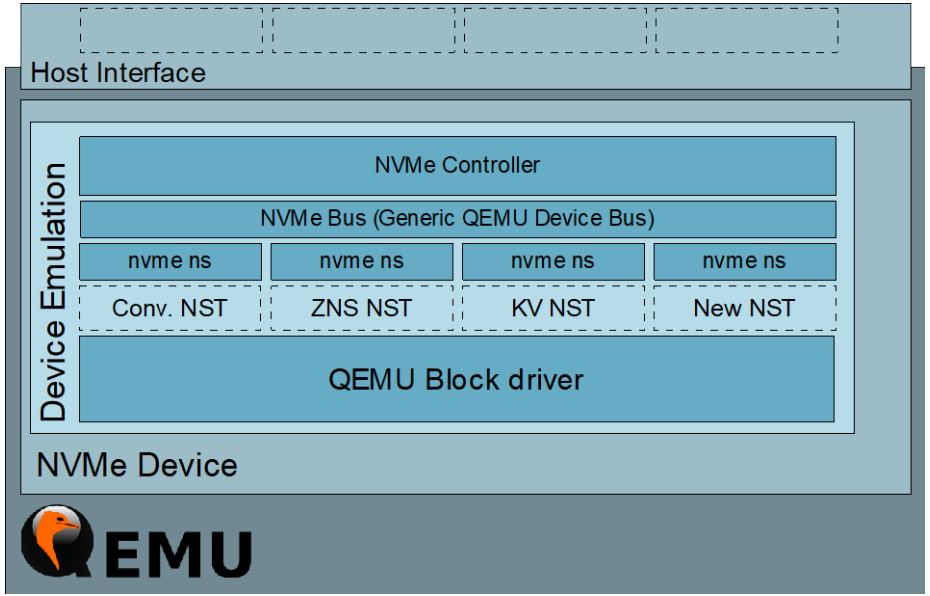


Tools

ZNS device emulation, system utilities and benchmark applications

QEMU® ZNS Device Emulation

- Enable host development without hardware
- Facilitate host debugging
  - Flexible zone configurations
  - Error injection
  - Facilitate compliance tests
    - Emulate different behaviors and device features
- Development on-going
  - v1.3 support, cleanups and refactoring merged in Q2 2020
  - SGLs, multiple namespaces (Reviewed)
  - I/O Command Sets & ZNS (Reviewing)
  - v1.4 support, extended meta, DULBE (Reviewing)
  - Other ZNS-related TPs will be supported



System Utilities

- blkzone
  - Part of util-linux
    - Primary Linux tool for zone device management
    - git@github.com:karelzak/util-linux.git
  - Common support for all zoned devices and zone management operations
    - Zone report, reset, open, close and finish
- nvme-cli
  - Primary Linux tool to manage NVMe® devices
    - https://github.com/linux-nvme/nvme-cli.git
  - ZNS extensions
    - ZNS Identification: Controller & Namespace
    - ZNS Management: State transitions, send / receive, descriptors
    - ZNS Report: Zone Information & Log pages
    - ZNS I/O (zone append)

```
Usage:
 blkzone <command> [options] <device>

Run zone command on the given block device.

Commands:
 report       Report zone information about the given device
 reset        Reset a range of zones.
 open         Open a range of zones.
 close        Close a range of zones.
 finish       Set a range of zones to Full.

Options:
 -o, --offset <sector>  start sector of zone to act (in 512-byte sectors)
 -l, --length <sectors> maximum sectors to act (in 512-byte sectors)
 -c, --count <number>   maximum number of zones
 -f, --force            enforce on block devices used by the system
 -v, --verbose          display more details

 -h, --help             display this help
 -V, --version          display version
```

```
usage: nvme zns <command> [<device>] [<args>]

The '<device>' may be either an NVMe character device (ex: /dev/nvme0) or an
nvme block device (ex: /dev/nvme0n1).

Zoned Namespace Command Set

The following are all implemented sub-commands:
  id-ctrl             Retrieve ZNS controller identification
  id-ns               Retrieve ZNS namespace identification
  zone-mgmt-recv      Sends the zone management receive command
  zone-mgmt-send      Sends the zone management send command
  report-zones        Retrieve the Report Zones report
  close-zone          Closes one or more zones
  finish-zone         Finishes one or more zones
  open-zone           Opens one or more zones
  reset-zone          Resets one or more zones
  offline-zone        Offlines one or more zones
  set-zone-desc       Attaches zone descriptor extension data
  zone-append         Writes data and metadata (if applicable), appended to the
  changed-zone-list   Retrieves the changed zone list log
  version             Shows the program version
  help                Display this help
```

Benchmarking

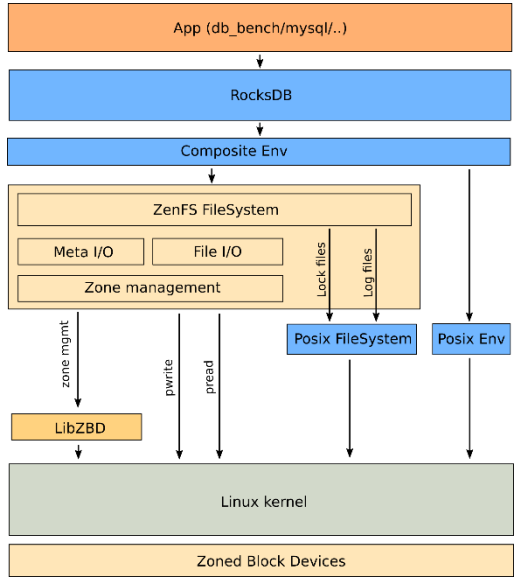
- fio: Flexible I/O Tester
  - De-facto storage benchmarking tool in Linux
  - https://github.com/axboe/fio
- Zoned support added in fio version 3.9
  - Initially for SMR disks
  - Now common support for all zoned devices
    - Zone capacity support
  - zonemode=zbd enables zoned I/O mode
  - Other options:
    - zonerange, zoneskip, read_beyond_wp
    - max_open_zones, zone_reset_threshold/frequency
- Improvements for ZNS on-going
  - Zone append writes and Simple Copy support
    - Depend on kernel interface

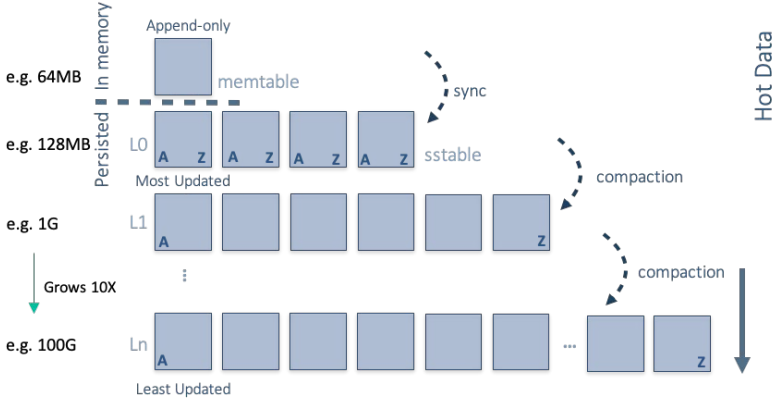
Example Applications

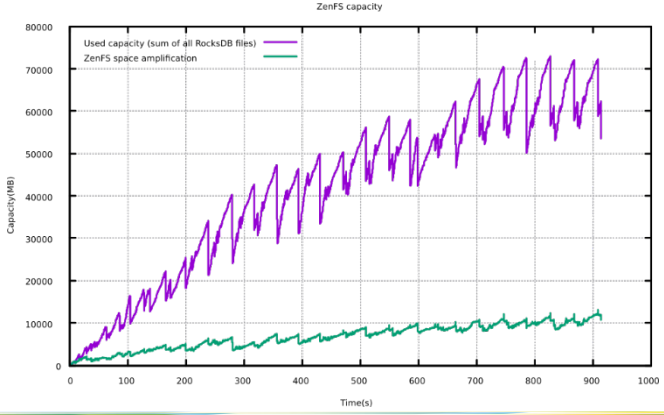
RocksDB, Hadoop/HDFS

RocksDB

- LSM-tree based Key-value store
- ZBD support implemented using new ZenFS storage backend
  - Uses libzbd
  - Maps Zones to SSTables
  - Result in ~1X device write amplification
- Patches posted upstream, review in progress
  - https://github.com/facebook/rocksdb/pull/6961

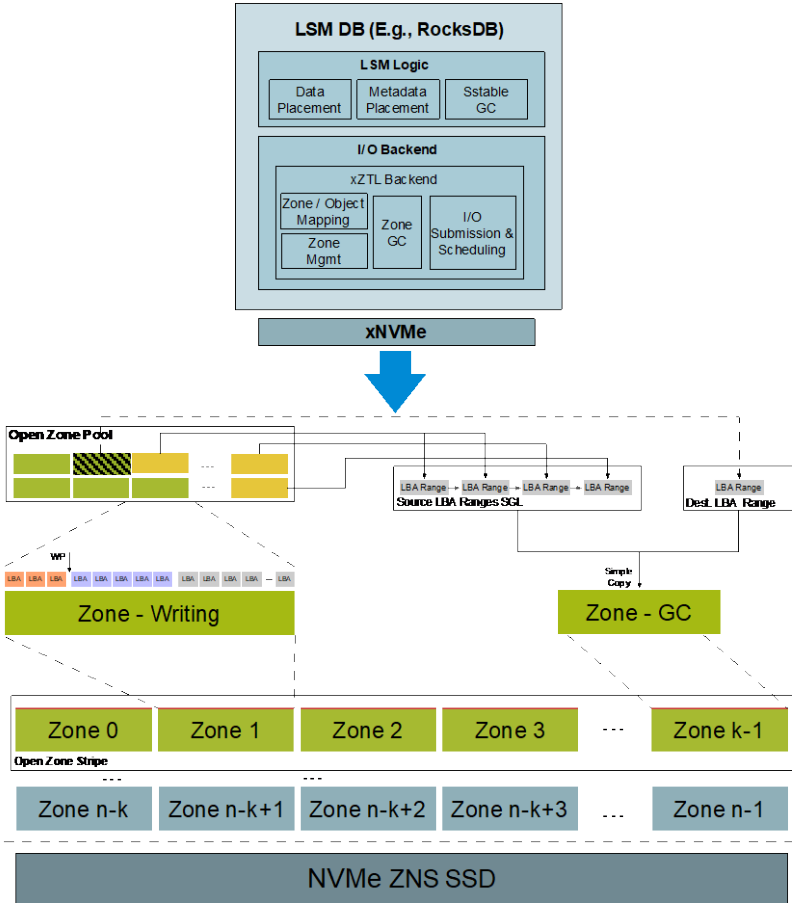






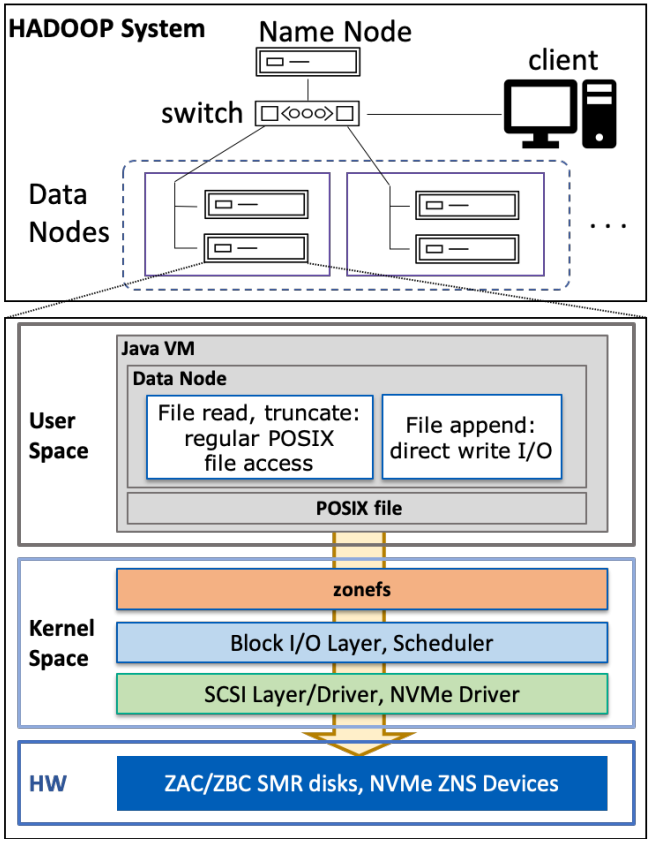
xZTL: LSM-based Databases

- Enable RocksDB ZNS with thin backend: ~1000 LOC
- Add zone logic for LSM DBs in xZTL
  - Easy to port other DBs (e.g., Cassandra)
  - Transparent support for several ZNS architectures and I/O models
    - Append: Large zones
    - Stripping: Small zones
  - xZTL: https://github.com/OpenMPDK/xZTL
  - RocksDB w/ xZTL: https://github.com/ivpi/rocksdb
    - Patches to be sent after ZenFS review
- Tight integration with xNVMe
  - Leverage changes in application to run on multiple OSs and I/O backends
- Write amplification of ~1X
  - ~2X reduction in SW



Apache Hadoop® HDFS

- Zoned block device support from JAVA implemented using zonefs
  - Simplifies development by keeping most POSIX semantic used with regular files
    - Allows reusing many code relying on regular POSIX system calls (read(), truncate(), etc) for chunk file management
  - zonefs use also naturally enable support for efficient zone append write path where possible
- Early results show significant performance improvements for large datasets
  - Smaller datasets benefit from regular file systems caching



Summary

- Linux® ZBD base ecosystem is mature
  - Benefits from long work on SMR HDDs
  - Solid sequential write constraint support
- Many tools and libraries provide help for developers
  - libnvme, nvmecli, libzbd, xnvme
- Work is still on-going
  - Zone append user interface semantic and implementation
  - ZNS passthrough
    - NVMe® command passthrough support for unsupported devices
      - ZNS devices without zone append implemented
    - Async passhtrough using io_uring
    - SPDK
  - File systems support
    - f2fs single device, btrfs, ZFS
  - Optimized application support
    - RocksDB, Cassandra, Hadoop/HDFS, Ceph


nvm
EXPRESS®
Architected for Performance