STORAGE DEVELOPER CONFERENCE

SDC 23

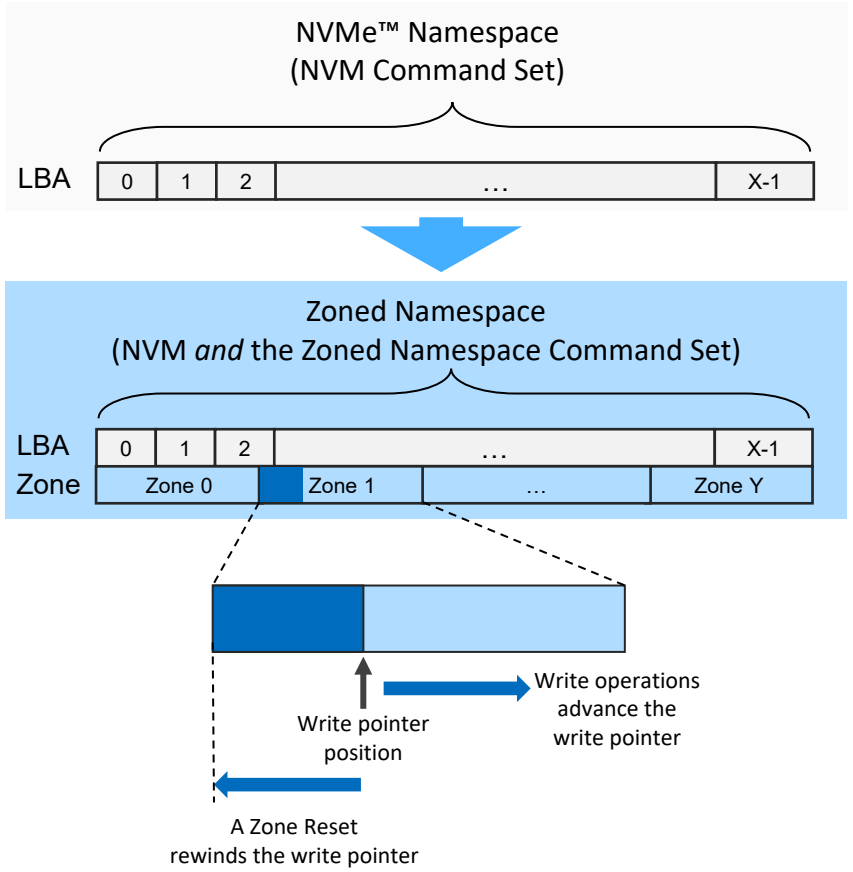
BY Developers FOR Developers

Zones and The Art of Log Structured Storage

Hans Holmberg and Dennis Maisenbacher

What is Zoned Storage?

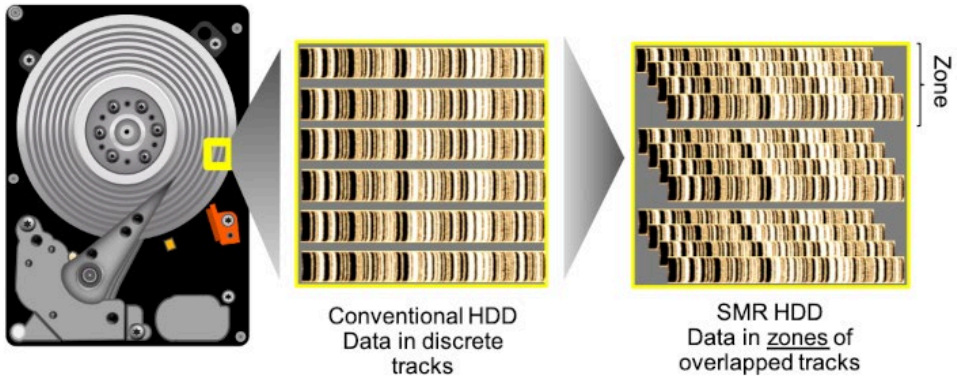
- A media independent storage interface
  - Inherits the existing concepts of logical blocks, LBAs, I/O Commands (e.g., Read and Write commands),
  - Logical blocks are divided into fixed-sized zones, which are then utilized for data placement by the host software
- Same interface for flash and HDDs
- Both conventional and zoned interfaces can be exposed by the same drive
  - E.g., useful for soft roll-outs as software is updated to take advantage of the zoned storage model



Why Zoned Storage?

HDDs

SMR HDDs offer 18% additional capacity with the newest 26TB SMR HDD over conventional CMR HDD



Flash

SSDs with Zoned Namespace support offer 3-4x higher performance, predictable latency, 7-28% higher capacity

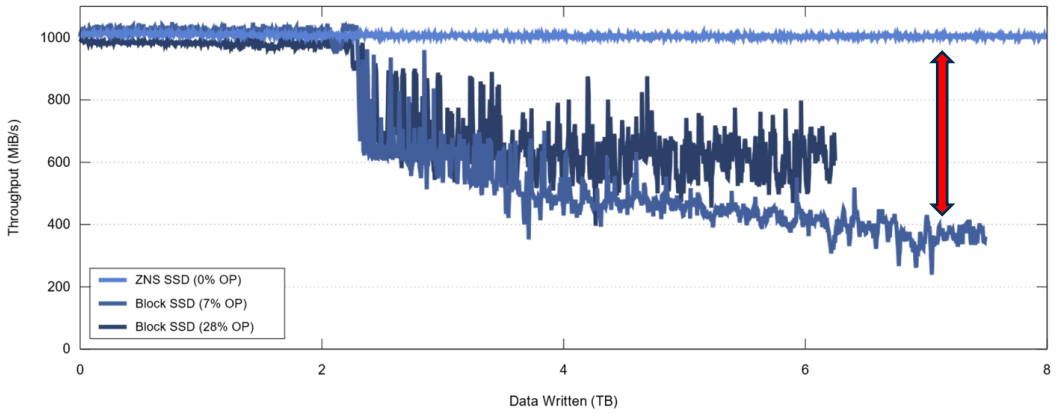


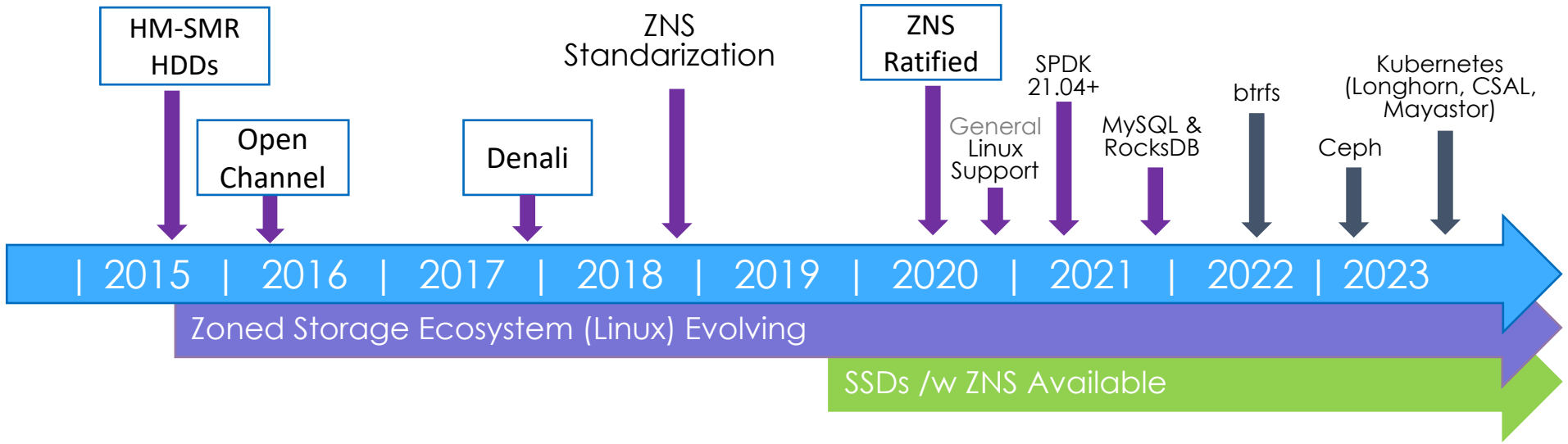
Image sources: zonedstorage.io

Zoned Storage vs Modern Workloads

- Modern storage workloads rely heavily on out-of-place sequential writes
  - Aligns with media characteristics
  - Provide higher throughput
  - Aids recovery
  - Enables easy snapshotting
- A great fit for zoned storage!
- Examples
  - Log structured merge tree data
  - Copy on write file systems

Zoned Storage Evolution

- Broad industry need for a standardized approach to direct data placement aligned to media characteristics
- The NVMe ZNS Group was formed at end 2018 to create the Zoned Namespace Command Set.
- The initial specification was ratified June 2020.
- ZNS support was added to Linux® software eco-system in June 2020, followed by SPDK support in April 2021.
- SSDs with Zoned Namespace support announced Q3 2020.
- UFS – Mobile flash
  - Driven by Google, Zoned Storage support is also being added to the UFS specification for use in mobile devices
- A single storage model across hard-drives, solid state drives, and embedded storage devices



Today's Linux® Eco-System

Official Linux® support since 2016

- Zoned API from kernel version 4.10 (Feb 2017)
- ZNS support added in kernel version 5.9 (Oct 2020)
- UFS support to be available when standardized (~2023)

5+ Linux® Distributions with Zoned Storage Support

- RHEL 9+, CentOS 7+, Fedora 33+, Debian 11+, and Ubuntu 21.04+

Two File-systems support Zoned Storage

- f2fs (client - UFS) and btrfs (enterprise - ZNS/SMR)

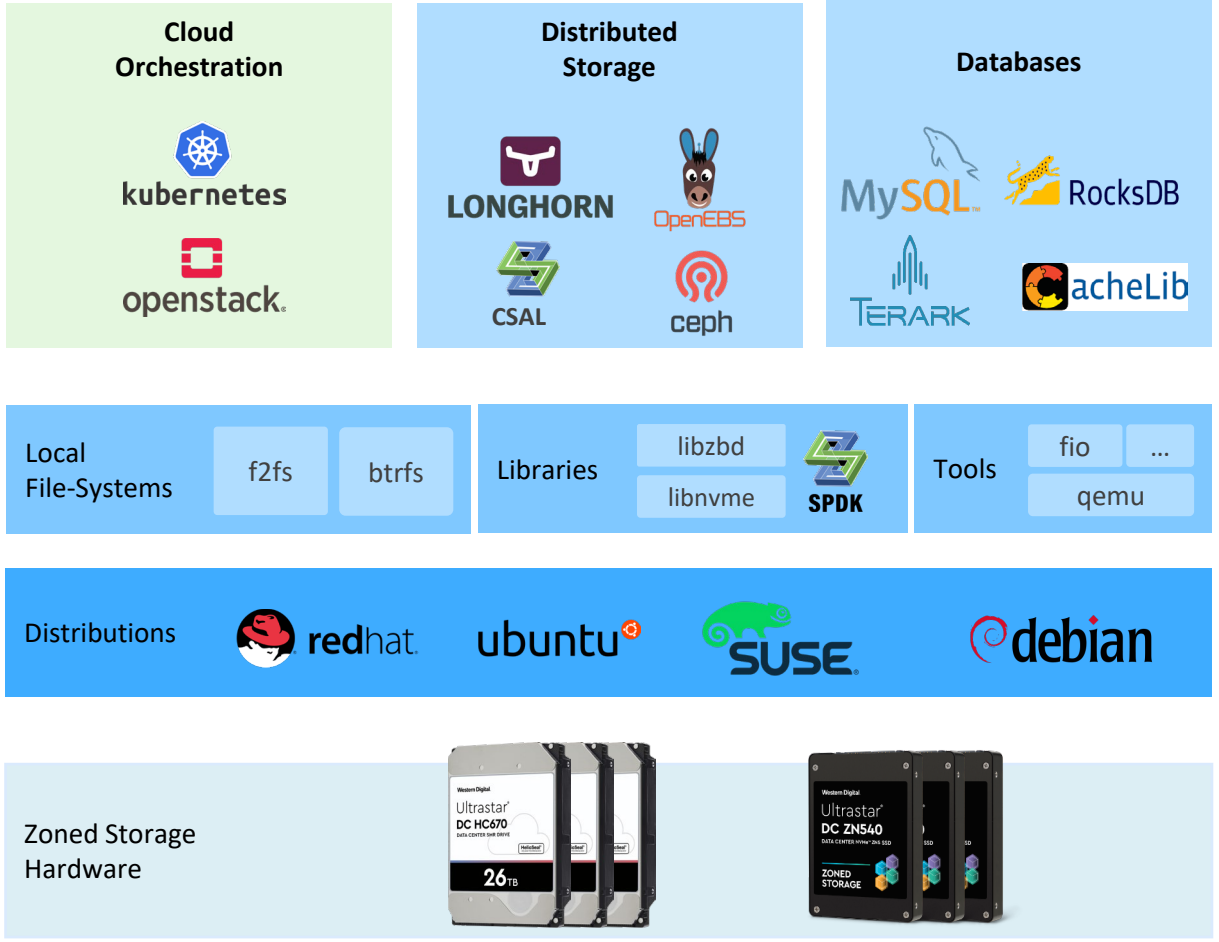
Library/Tools support

- libzbd, libnvme, SPDK, fio, qemu, blkzone, blktests, …

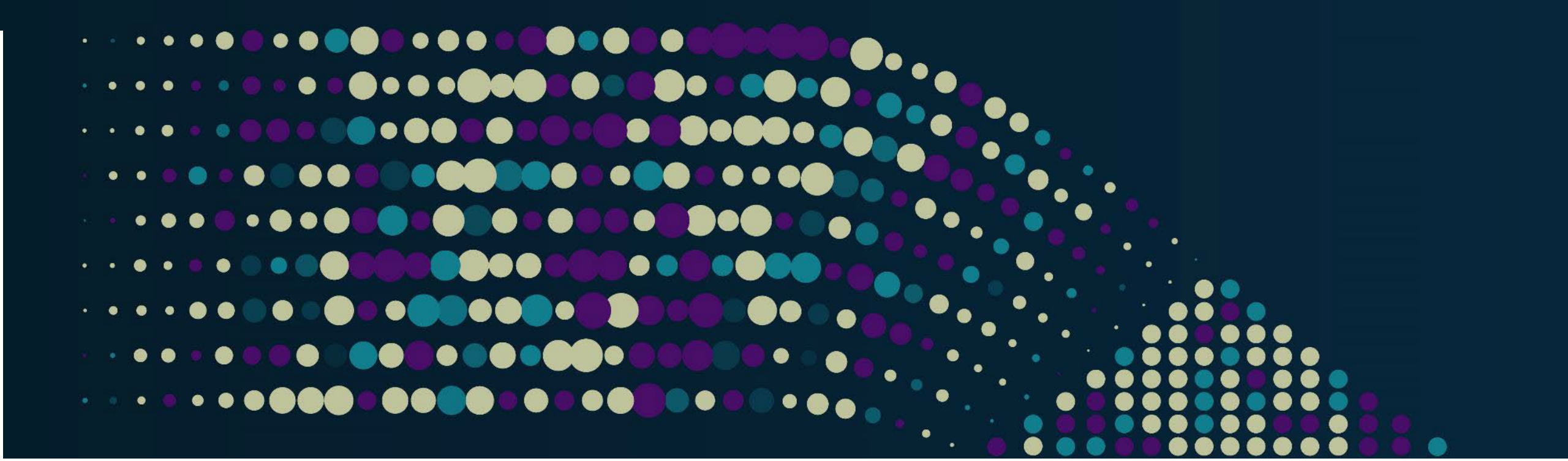
Broader Enablement

- Cloud Orchestration, Databases, Distributed Storage, Databases, Caching, Generic storage

Mature, robust, and adopted by some of the biggest consumers of storage



SDC 23

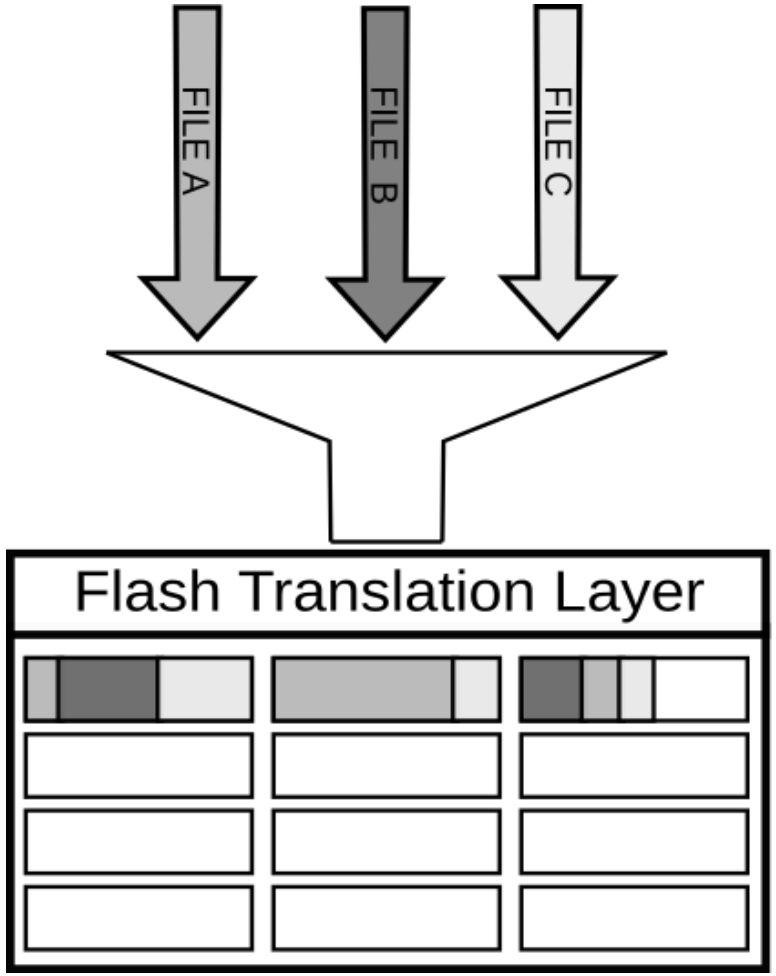


Storage stack integration

End-to-end data placement for databases

Databases

- LSM based key-value stores are designed to be flash-friendly
  - Log-structured append-only writes
  - Application defined garbage collection (compaction)
  - Large files (>128MB)
  - Few write streams (~12)
- ... but still suffers from significant amounts of device-side write amplification (2-3x)
  - Files are mixed in an SSD's erase units
  - Garbage collection is required to reclaim space

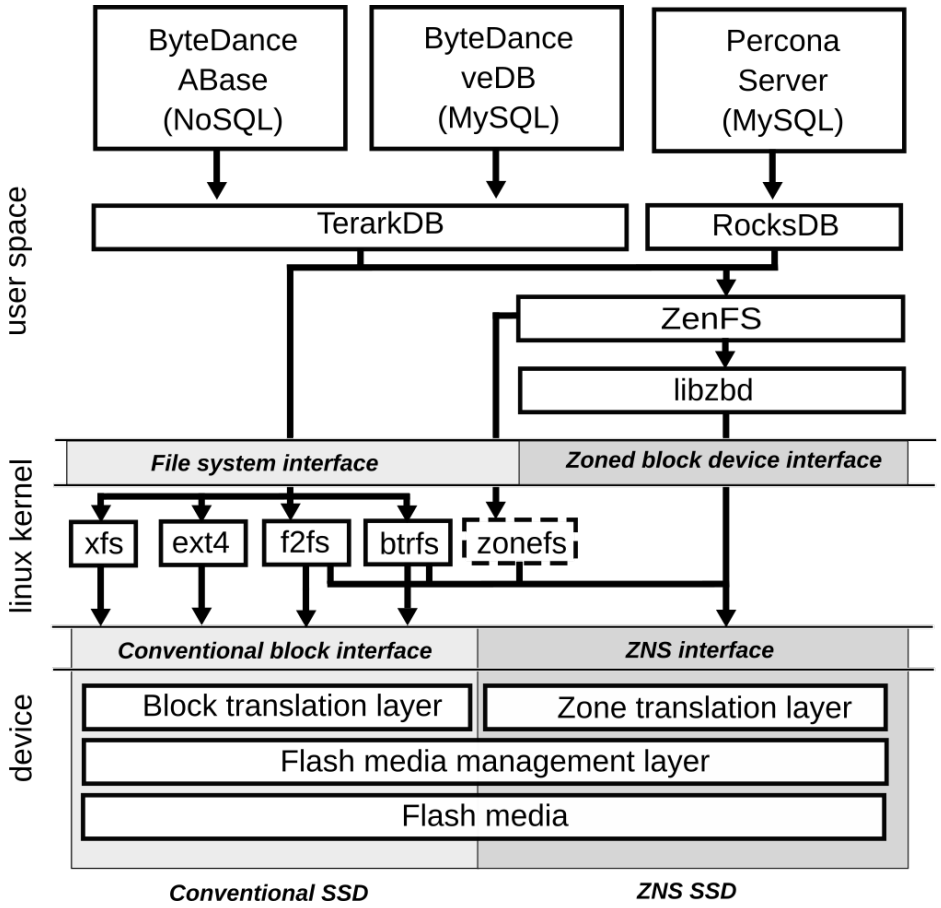


Conventional Write Amplification Mitigation

- **Over provisioning** is generally used as a method to reduce write amplification
  - **By reserving more space, write amplification decreases**
  - **Comes at a big performance cost** – to achieve the same performance, the usable disk capacity is commonly reduced with up to 50%
- Reserving 50% of SSD's storage capacity doubles the storage cost!

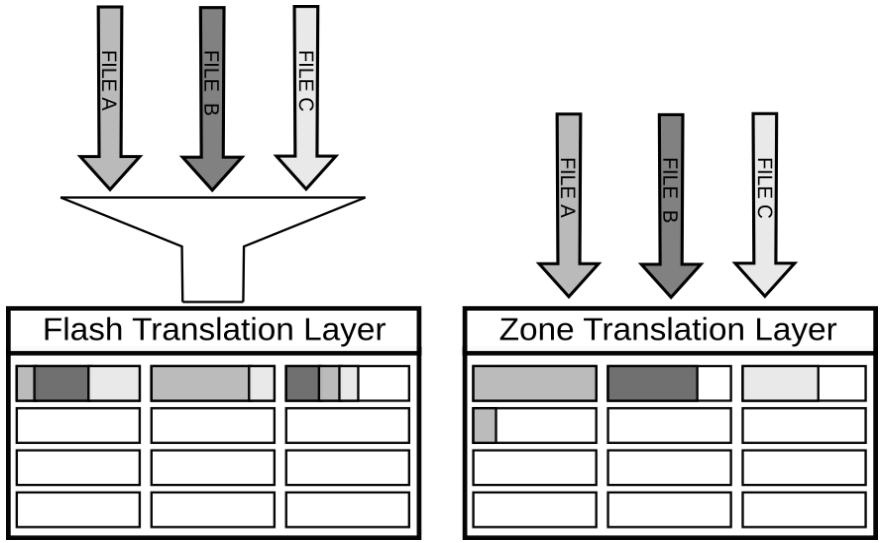
The ZenFS Filesystem

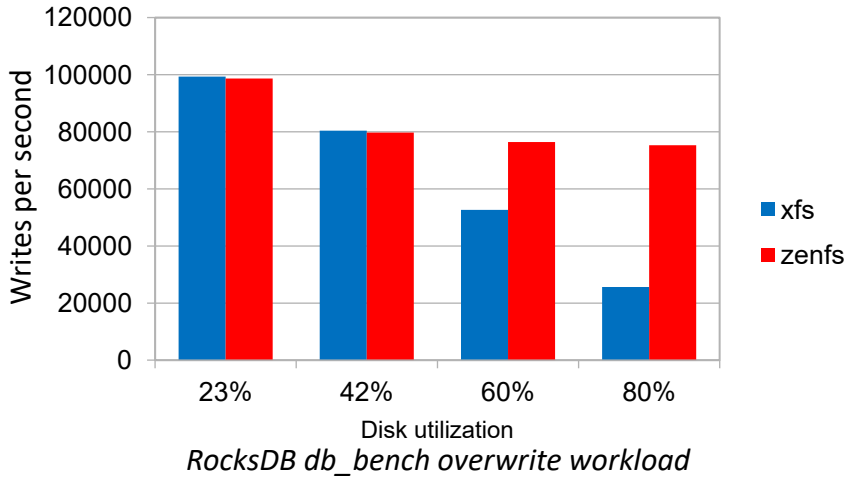
- A user space file system that is compatible with the **RocksDB** and **TerarkDB** storage backends
- Performs data placement onto zoned storage devices:
  - SSDs with ZNS support to reduce write amplification/device lifetime, improve read tail latency and throughput
  - SMR HDDs to utilize the extra available capacity
- Seamless end-to-end data placement
- Very competitive in benchmarks comparing with existing file systems on conventional SSDs



ZenFS Data Placement

- Use the Zoned Storage interface to separate files into different zones
- Co-locate files of similar lifetimes to fill up any remaining space
- Once the files occupying a zone is deleted, space can be reclaimed without any garbage collection
- No write amplification
- 3x overwrite throughput improvement at 80% space utilization





RocksDB db_bench overwrite workload

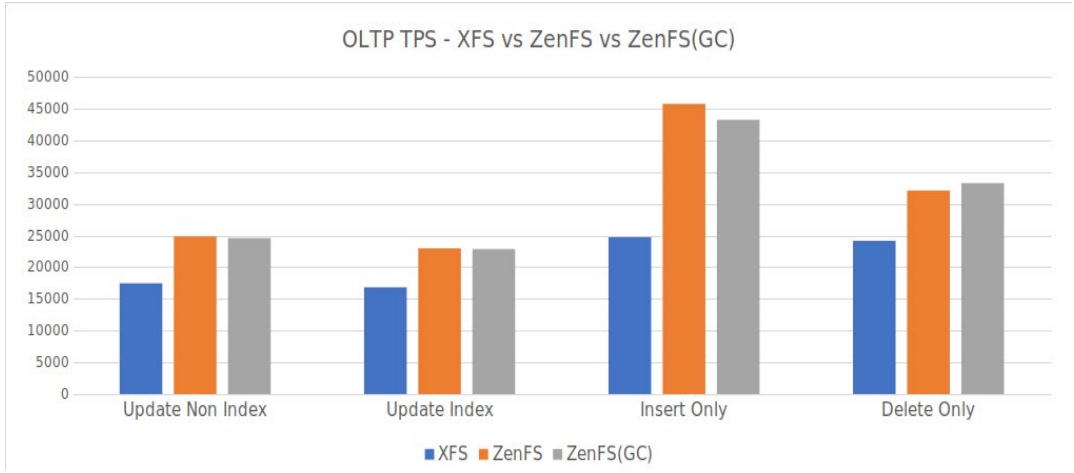
Mixed Read/Write Workloads

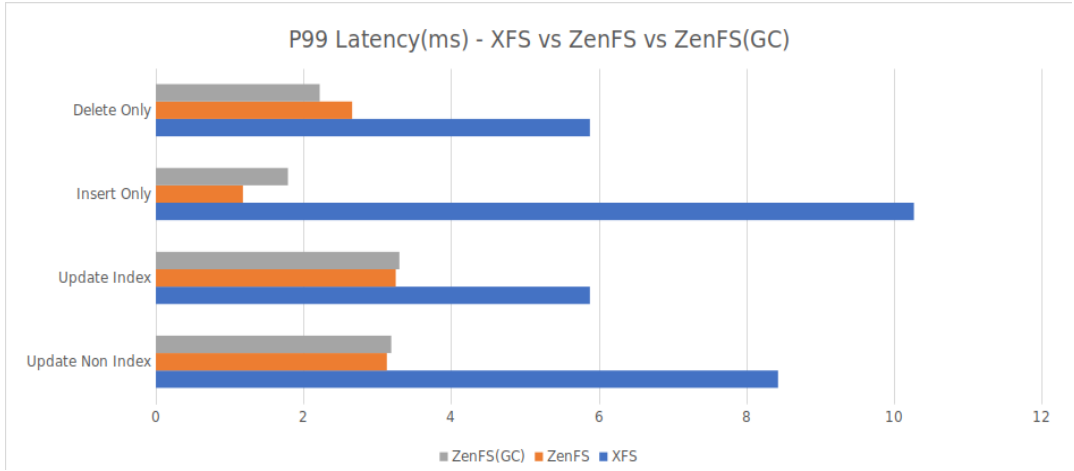
- Garbage collection requires both extra reading and writing to relocate data in order to free up space.
- Extra reading and writing affects throughput and tail latencies.
- ZenFS does not use any garbage collection per default. The tradeoff is increased space amplification.
- A small (2-3%) amount of optionally enabled GC brings **space amplification** on par with XFS on a conventional drive
- With GC enabled, ZenFS can sustain **60% more writes** while allowing **190% more reads**
- **Read tail latencies are reduced by an order of magnitude**

Percona MySQL

- Write-heavy OLTP workloads using sysbench
  - DB size: 600GB, 2h of preconditioning (overwrites), 32 threads
  - Setup:
    - XFS: 1TB SN540 (Conventional Namespace)
    - ZenFS: 1TB ZN540 (Zoned Namespace)
    - ZenFS (GC): ZenFS with Garbage collection enabled

- By using ZenFS, it is possible to achieve
  - Up to 80% Higher throughput
  - Lower tail latency by up to 10x



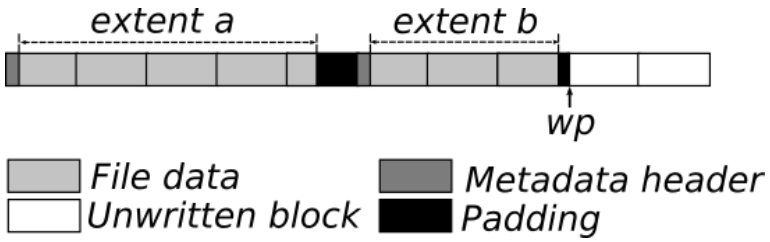


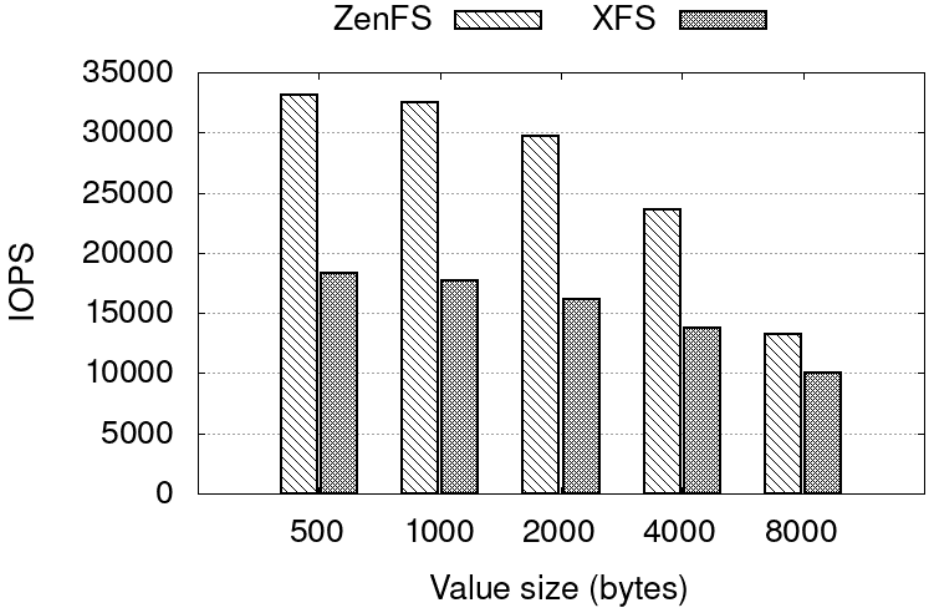
Persistent Write Optimizations

- Some workloads are dependent on transactions being committed to flash for every write
- Conventional file systems store data and metadata separately, requiring at least two writes per persisted user write

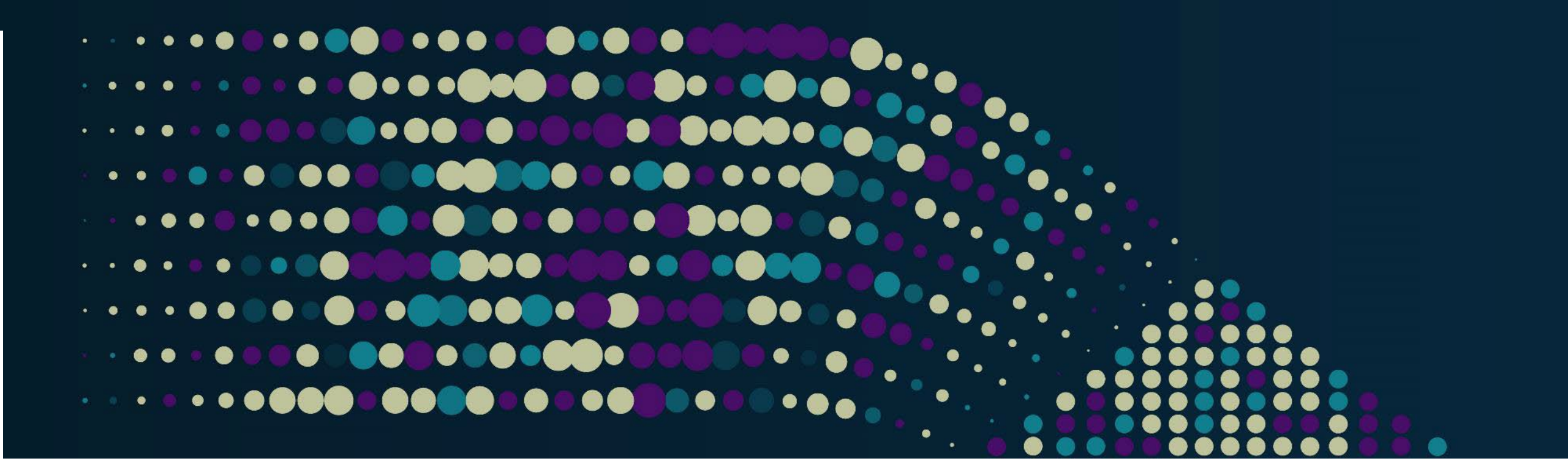
- ZenFS implements in-line metadata storage, writing file metadata together with file data, greatly improving throughput for synced writes
- ZenFS can persist user writes with a single I/O

- Up to 2X write throughput improvement





Synced write only throughput (db_bench)



Zoned Cloud Native Storage

Storage in the Cloud

- Cloud Service Providers (CSPs) are constantly challenged with large volumes of data and increasing customer demand for cost-effective storage and high performance.
  - Measurements: IOPS/TB, GB/$, TB/Rack

- To solve these challenges, CSPs provide software-defined storage solutions by exposing the data as a service that implements multiple storage tiers to provide the appropriate benefits.

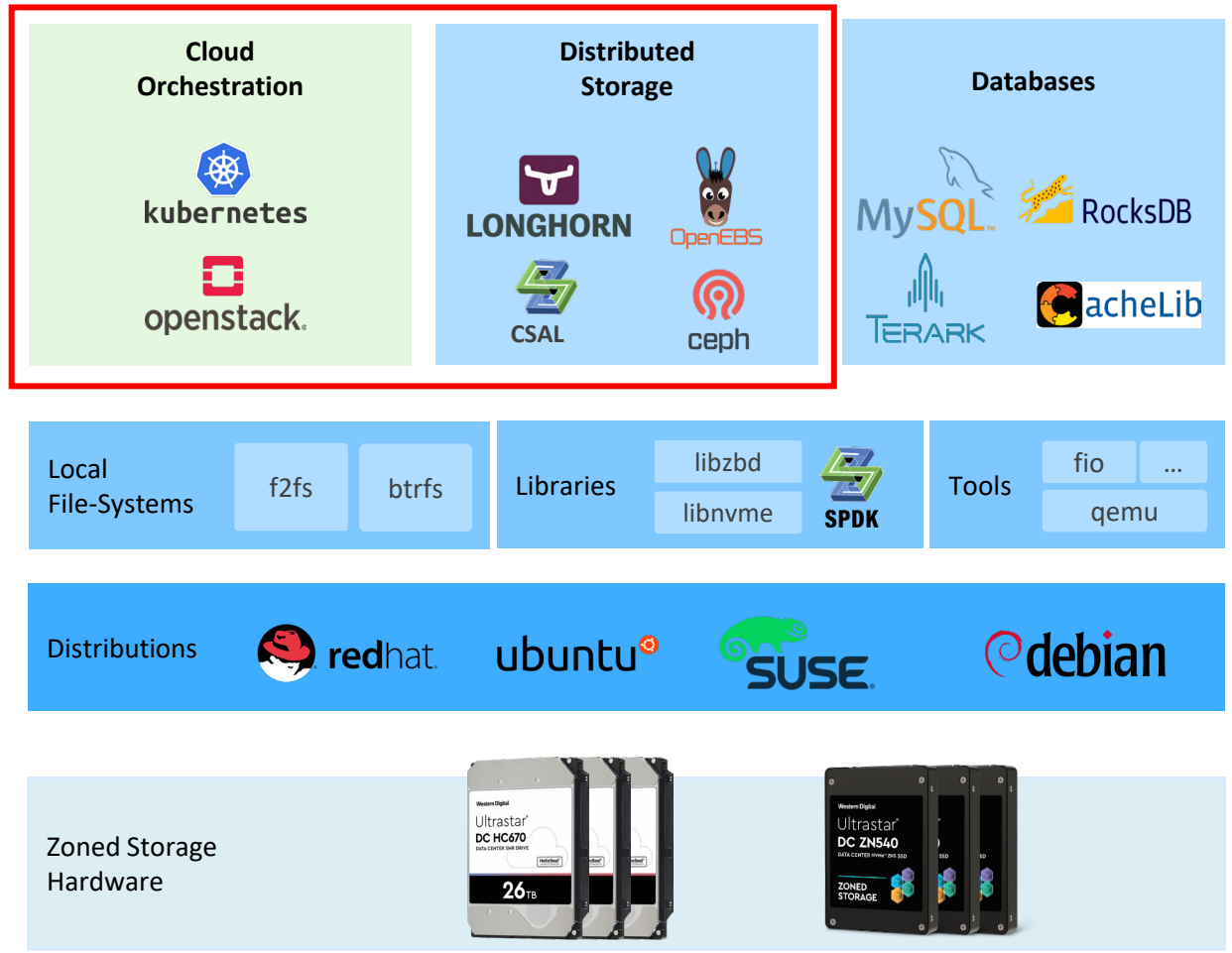
CSP's Challenges – Storage Hardware

- Storage tiers have inherent physical characteristics, which, if to be used in the cloud, need to be effectively managed
- **Flash-based SSDs (Performance Tier):**
  - Media type (TLC/QLC), WAF, DRAM, OP → **Cost (GB/$)**
  - Relatively expensive compared to HDDs. The characteristics of conventional SSDs are such that they exhibit:
    - Write amplification which impacts lifetime, performance, and overall cost.
    - Typical overheads include 2x cost (CacheLib paper) and 3-4x reduced lifetime (rw workload *) (→contradicts the requirement of deploying capacity storage (i.e., QLC) for 5-7 years)
    - Bandwidth penalties in single device - multi tenant setups.
  - ZNS SSDs solve these issues while also maintaining high performance.
- **HDDs (Capacity Tier):**
  - Rotation speed, actuator and platter count → **Cost and Throughput per GB (IOPS/GB & GB/$)**
  - **Large-scale HDD deployments** (power and space constrained) are **improved** using SMR technology (22TB vs 26TB)

Previously Missing Link for Cloud Applications

- Tight integration with the Cloud Orchestration Platform required.
- The Storage as a Service is provided by the specific CSP (Azure, AWS, GCP) which implements internal solutions to gain the advantages.
- But how to get the benefits with on-premise clouds and/or hybrid clouds?

- <u>Enablements</u>
  - **Longhorn** and **CSAL** provide a **generic block device/file-system** abstraction for containers using zoned storage
  - With **Mayastor**, we made it such that **zoned storage** devices can be **explicitly exposed** to a container!
  - **Ceph** provides conventional **block, object and file storage** in one system.



1) Longhorn – Zoned Cloud Native Storage

- Longhorn is a persistent distributed storage system with no single point of failure
  - A replica is accessing the SSD through the file system abstraction
  - Containers accesses a POSIX compatible filesystem as the k8s volume or optionally a conventional block device
- Changes necessary
  - Make use of the native support for zoned storage in BTRFS
    - Multi-year effort on enablement now makes it easy to integrate
    - Works, with minimal configuration, out of the box

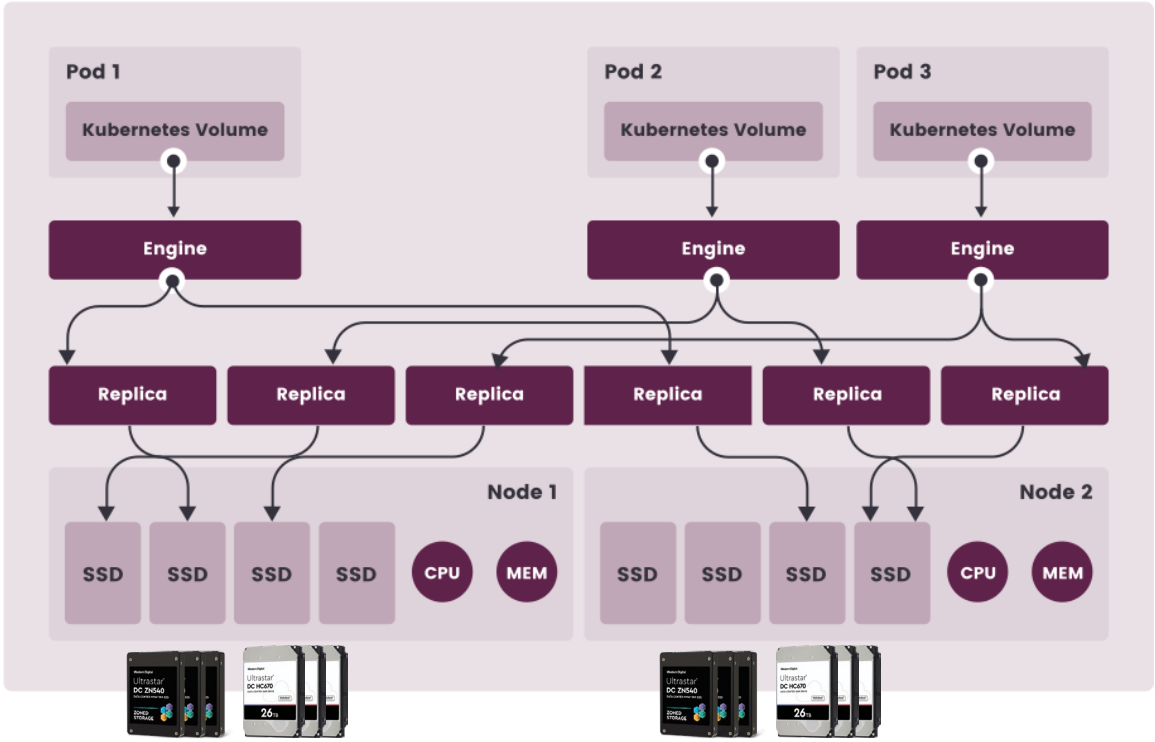


Image source: https://longhorn.io/img/diagrams/architecture/how-longhorn-works.svg

2) Mayastor – Zoned Cloud Native Storage

- OpenEBS - Mayastor (written in Rust 🦀) is much like Longhorn a persistent highly available Container Attached Storage (CAS) solution
  - Utilizes SPDK → Avoiding storage related kernel modules for node uptime
  - Ongoing plumping work in fixing up the Mayastor I/O path to natively handle ZNS NVMe block devices → First proposal with replica factor 1
  - Interesting challenges with zoned replication:
    - Keep zone write pointer across replicas in sync
    - What happens on I/O errors?
- CSI driver consumer will receive a zoned block device via NVMe-oF

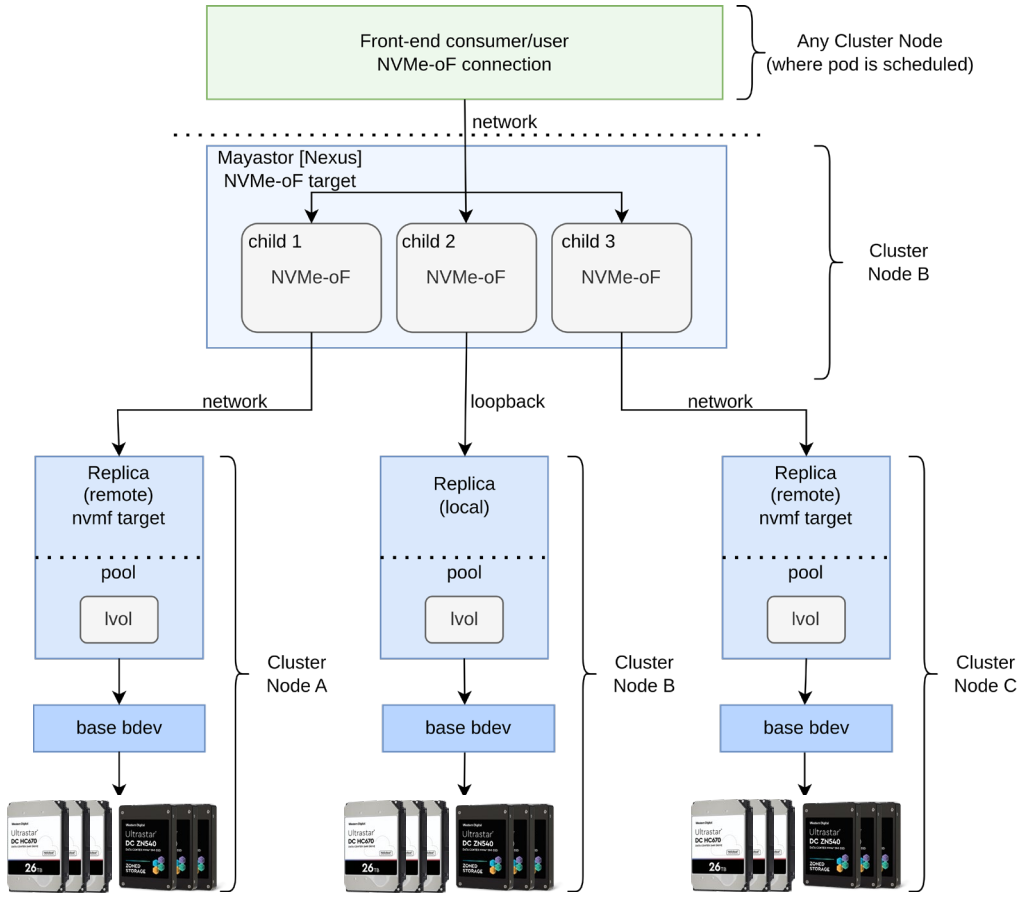
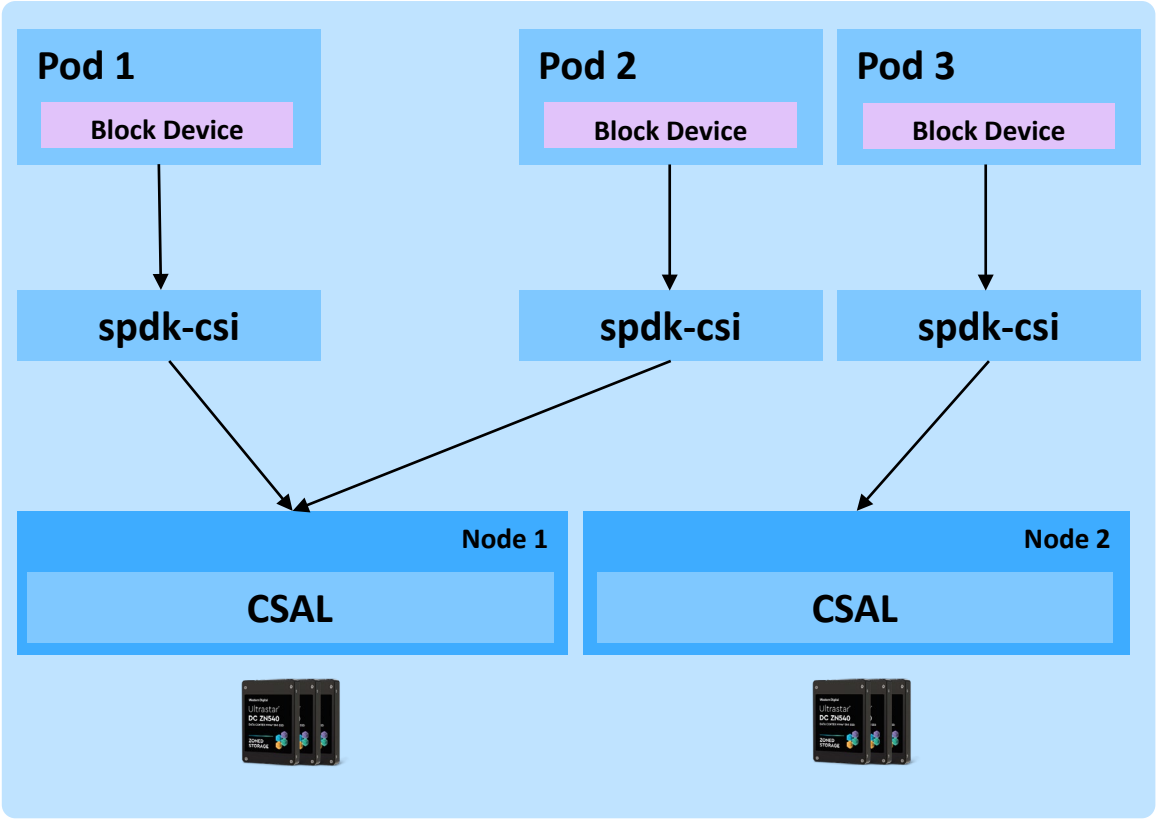


Figure inspiration: https://mayastor.gitbook.io/introduction/additional-information/i-o-path-description

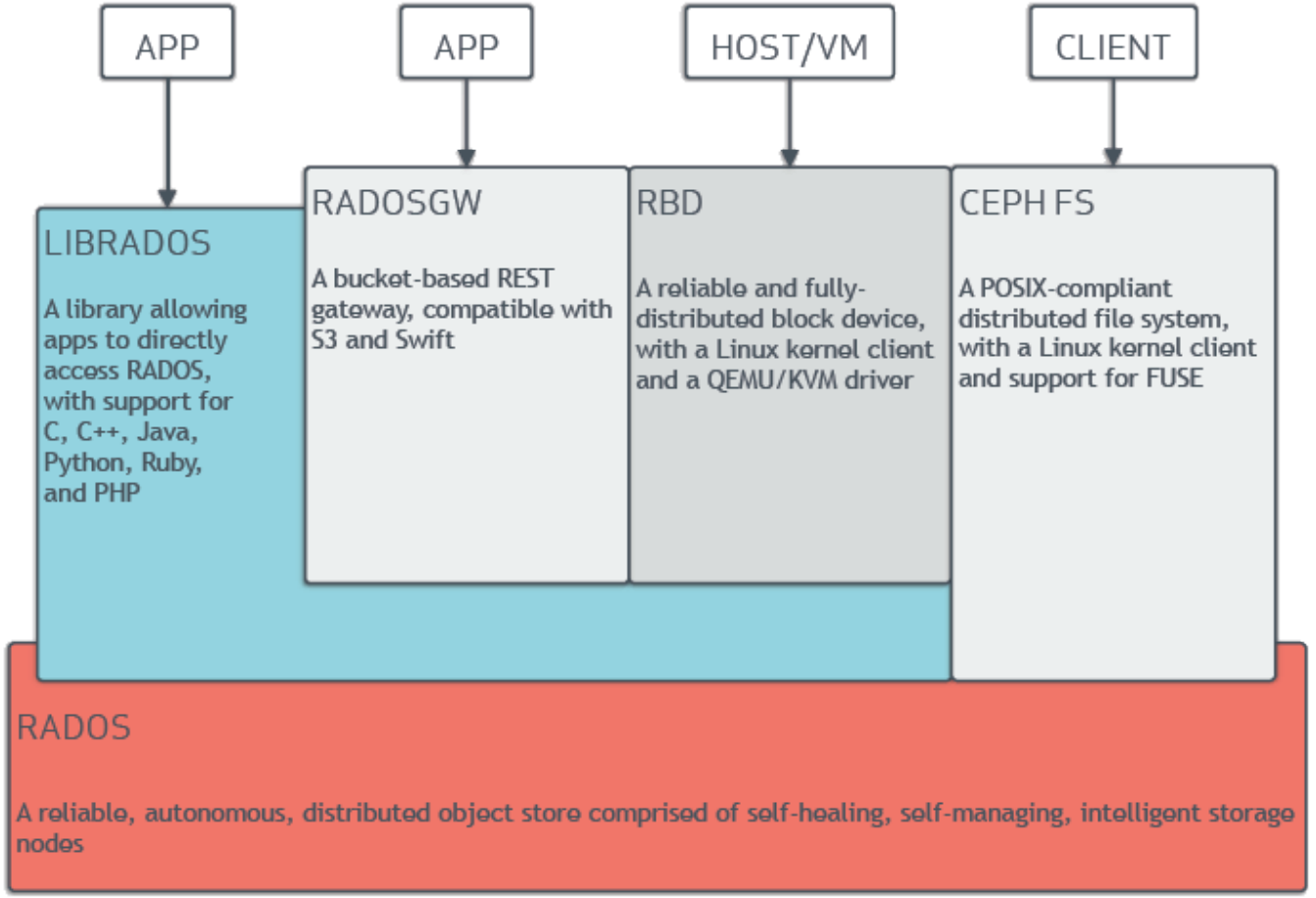
3) SPDK CSAL – Zoned Cloud Native Storage

- The Cloud Storage Acceleration Layer (CSAL) which has WIP zoned storage support can be deployed as a CAS though the spdk-csi driver or Mayastor
  - Implements a caching and translation layer that transforms zoned storage to conventional storage
- CSAL uses a conventional (high-performance) block device for metadata and writes sequentially to the ZNS SSDs, thus hiding ZNS' sequential write constraint
- Exposed as a conventional block device over a NVMe-oF target



4) Ceph™ – Zoned Cloud Native Storage

- Ceph is a distributed, highly reliable, highly scalable, unified and open source storage software.
- Provides block, object and file storage in one system.
- Reliable Autonomic Distributed Object Store (RADOS) is at the core of Ceph storage clusters. This layer makes sure that stored data always remains consistent and performs data replication, failure detection and data recovery.
- Native and easy Kubernetes integration with Rook

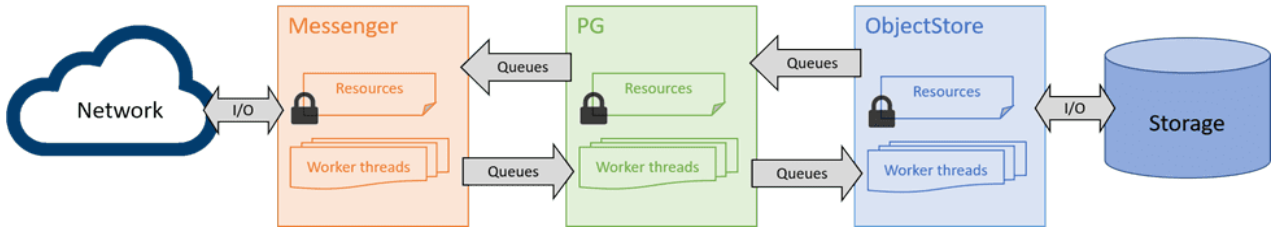


Slide credits: Aravind Remesh
Image source: https://docs.ceph.com/en/quincy/

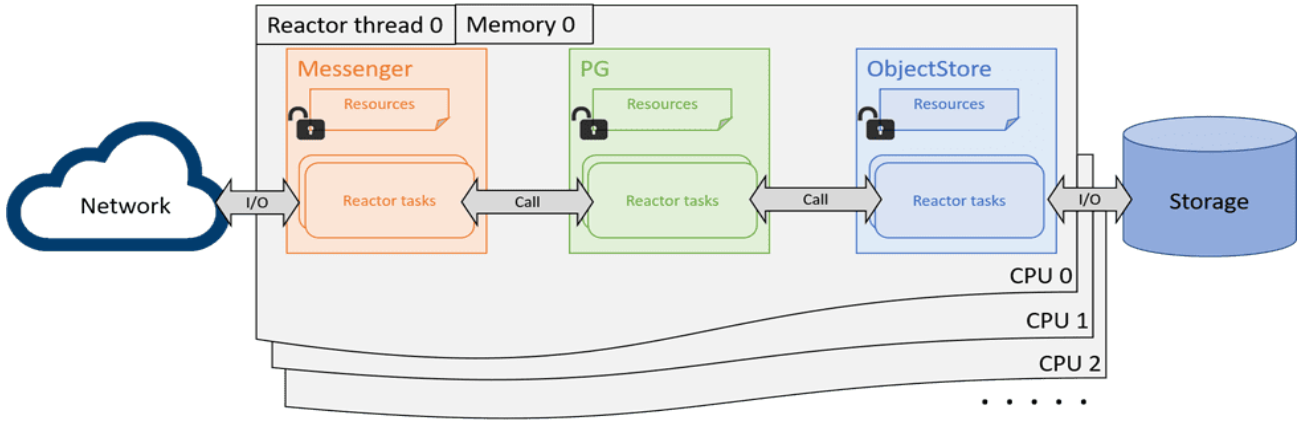
4) Ceph™ – Crimson Project

- Crimson project is working on creating a next-gen OSD, called crimson-osd.
- Classic OSD uses a threading model which involves a lot of CPU time wasted in blocking calls(serializations) and also involves cross-core communication which pollutes the caches.
- Crimson-osd - based on the event-driven async framework Seastar™ - working towards being a replacement object storage daemon(OSD) with the following goals:
  - Minimize cpu overhead
    - Minimize cycles/iop , copies
    - Minimize cross-core communication
  - Enable emerging storage technologies
    - **Zoned storage - ZNS SSDs, HM-SMR HDDs**
    - Persistent Memory

Legacy OSD



Crimson OSD



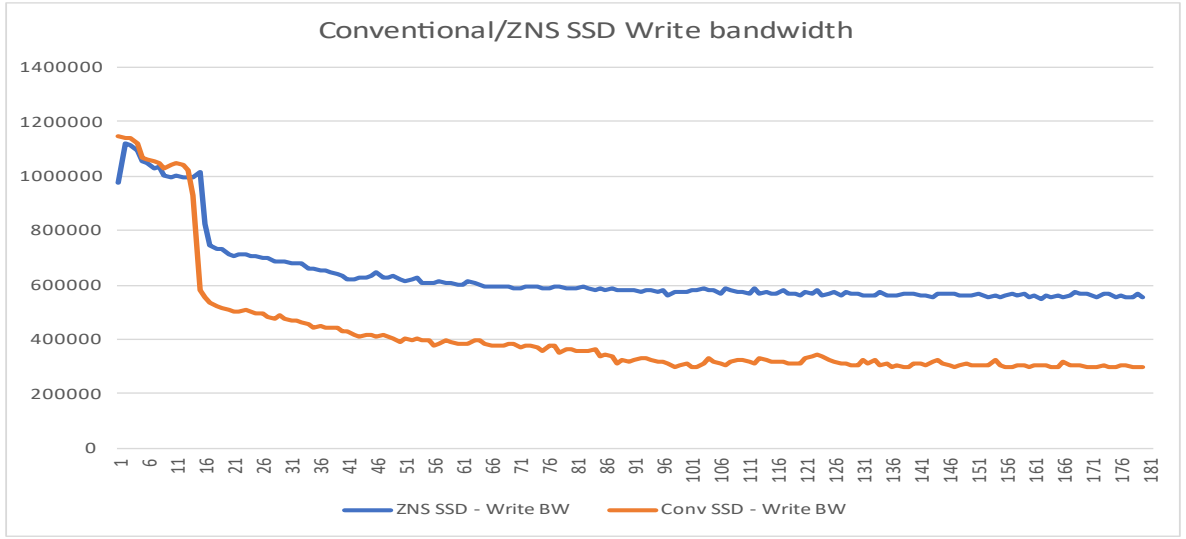
*Ref: https://www.usenix.org/conference/vault20/presentation/just

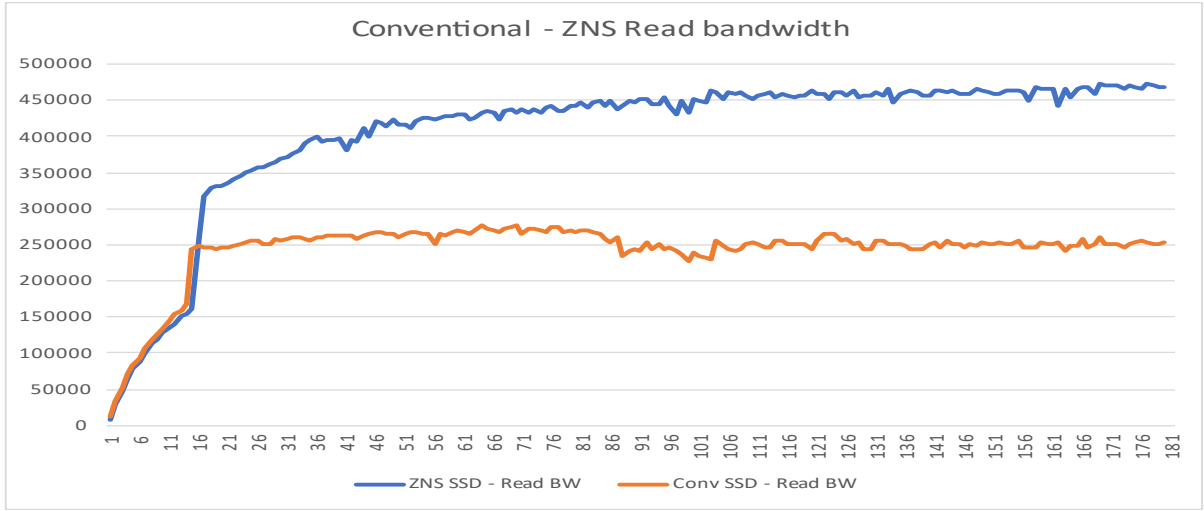
Slide credits: Aravind Remesh

4) Ceph™ – Performance improvements

- Rados block device
- Fio random read/write workload
- Read: 20%, Write: 80%
- Read: 64KiB, Write: 1024KiB
- Queue depth: 16
- Drive size: ~900GB
- Runtime: 3 hours

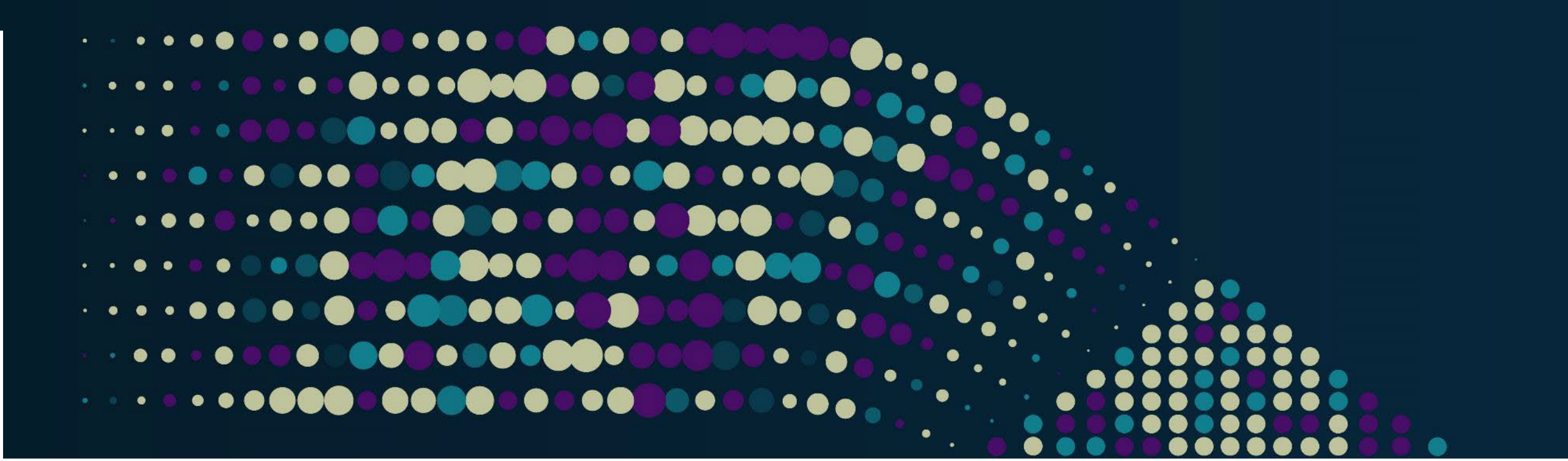
- **30% higher throughput for both reads and writes once garbage collection starts**





Slide credits: Aravind Remesh

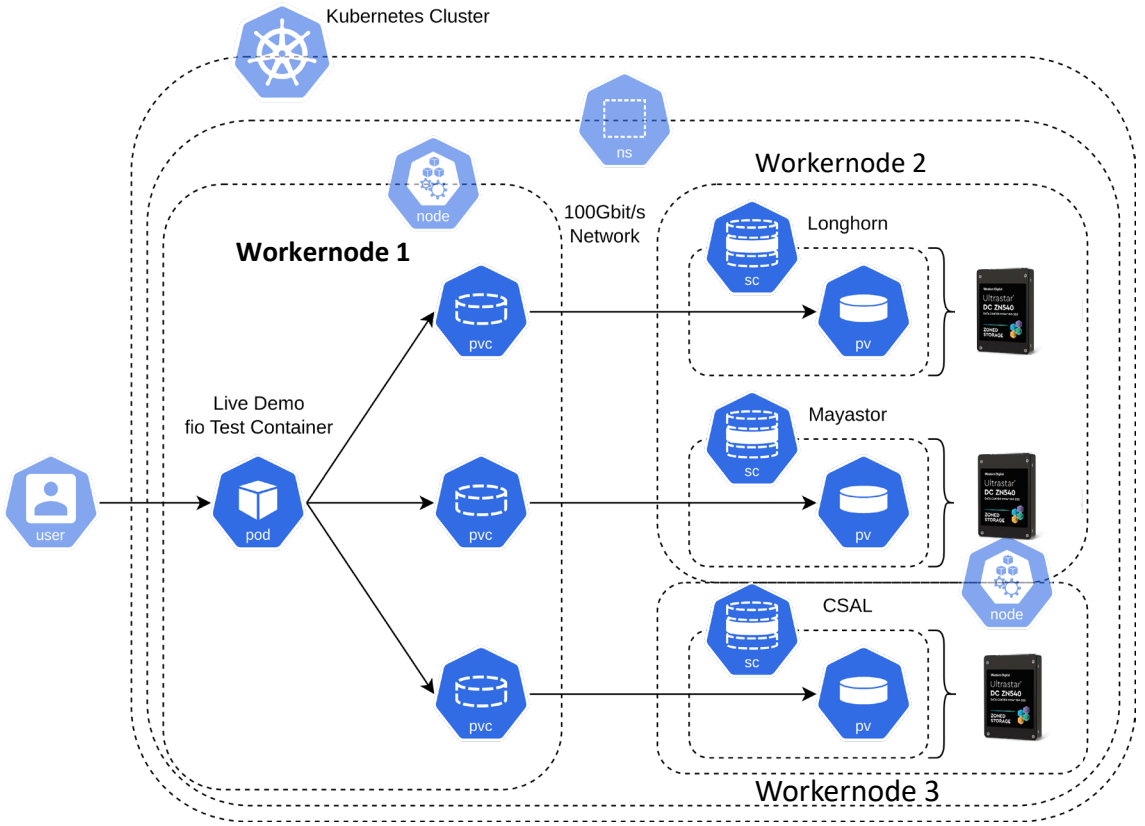
SDC 23



Showcase

Showcase Setup

- 4 Nodes (one control and three worker nodes)
  - Workernode 1
    - User workload - Container /w fio
  - Workernode 2
    - Longhorn - Exposes a generic file-system/block device backed by an Ultrastar® DC ZN540 SSD
    - Mayastor - Exposes an Ultrastar® DC ZN540 SSD natively to a container
  - Workernode 3
    - SPDK CSAL - Exposes a generic block device over NVMe-oF using an Ultrastar® DC ZN540 as its storage backend



Showcase Setup

```yaml
---
# FMS Demo Pod
apiVersion: v1
kind: Pod
metadata:
  name: fms-demo
spec:
  containers:
  - name: fms-fio
    image: dennis-k8s-masternode:32000/test/fms-fio-container:latest
    imagePullPolicy: Always
    command: [ "/bin/sleep" ]
    args: [ "365d" ]
    volumeDevices:
    - name: mayastor-zoned-bdev
      devicePath: /dev/nvme0n1
    volumeMounts:
    - mountPath: "/longhorn-vol"
      name: longhorn-vol
    - mountPath: "/spdkcsi-csal-vol"
      name: spdkcsi-csal-vol
  volumes:
  - name: longhorn-vol
    persistentVolumeClaim:
      claimName: longhorn-pvc
  - name: mayastor-zoned-bdev
    persistentVolumeClaim:
      claimName: mayastor-zoned-bdev-pvc
  - name: spdkcsi-csal-vol
    persistentVolumeClaim:
      claimName: spdkcsi-csal-pvc
```

Zoned Block Device

POSIX Filesystem on
Conv Block Device
Backed by Zoned Storage

```
ubuntu@dennis-k8s-masternode:~$ kubectl exec --stdin --tty fms-demo -- /bin/bash
root@fms-demo:/# lsblk
NAME    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
loop0     7:0    0  63.4M  1 loop
loop1     7:1    0 111.9M  1 loop
loop2     7:2    0  53.3M  1 loop
loop3     7:3    0 169.1M  1 loop
loop4     7:4    0   1.8T  0 loop
sda       8:0    0     8G  0 disk /longhorn-vol
vda     252:0    0   189G  0 disk
|-vda1  252:1    0 188.9G  0 part /etc/resolv.conf
|                                 /etc/hostname
|                                 /dev/termination-log
|                                 /etc/hosts
|                                 /spdkcsi-csal-vol
|-vda14 252:14   0     4M  0 part
`-vda15 252:15   0   106M  0 part
vdb     252:16   0     1M  0 disk
nvme0n1 259:1    0   1.8T  0 disk
nvme1n1 259:3    0    32G  0 disk
root@fms-demo:/# cat /sys/block/nvme0n1/queue/zoned
host-managed
root@fms-demo:/# fio fms-example.fio
longhorn-write: (g=0): rw=write, bs=(R) 64.0KiB-64.0KiB, (W) 64.0KiB-64.0KiB, (T) 64.0
...
spdkcsicsal-write: (g=0): rw=write, bs=(R) 64.0KiB-64.0KiB, (W) 64.0KiB-64.0KiB, (T) 6
...
mayastor-write: (g=0): rw=write, bs=(R) 64.0KiB-64.0KiB, (W) 64.0KiB-64.0KiB, (T) 64.0
...
longhorn-read: (g=1): rw=read, bs=(R) 64.0KiB-64.0KiB, (W) 64.0KiB-64.0KiB, (T) 64.0Ki
...
spdkcsicsal-read: (g=1): rw=read, bs=(R) 64.0KiB-64.0KiB, (W) 64.0KiB-64.0KiB, (T) 64.
...
mayastor-read: (g=1): rw=read, bs=(R) 64.0KiB-64.0KiB, (W) 64.0KiB-64.0KiB, (T) 64.0Ki
...
```

POSIX Filesystem on
Conv Block Device
Backed by Zoned Storage

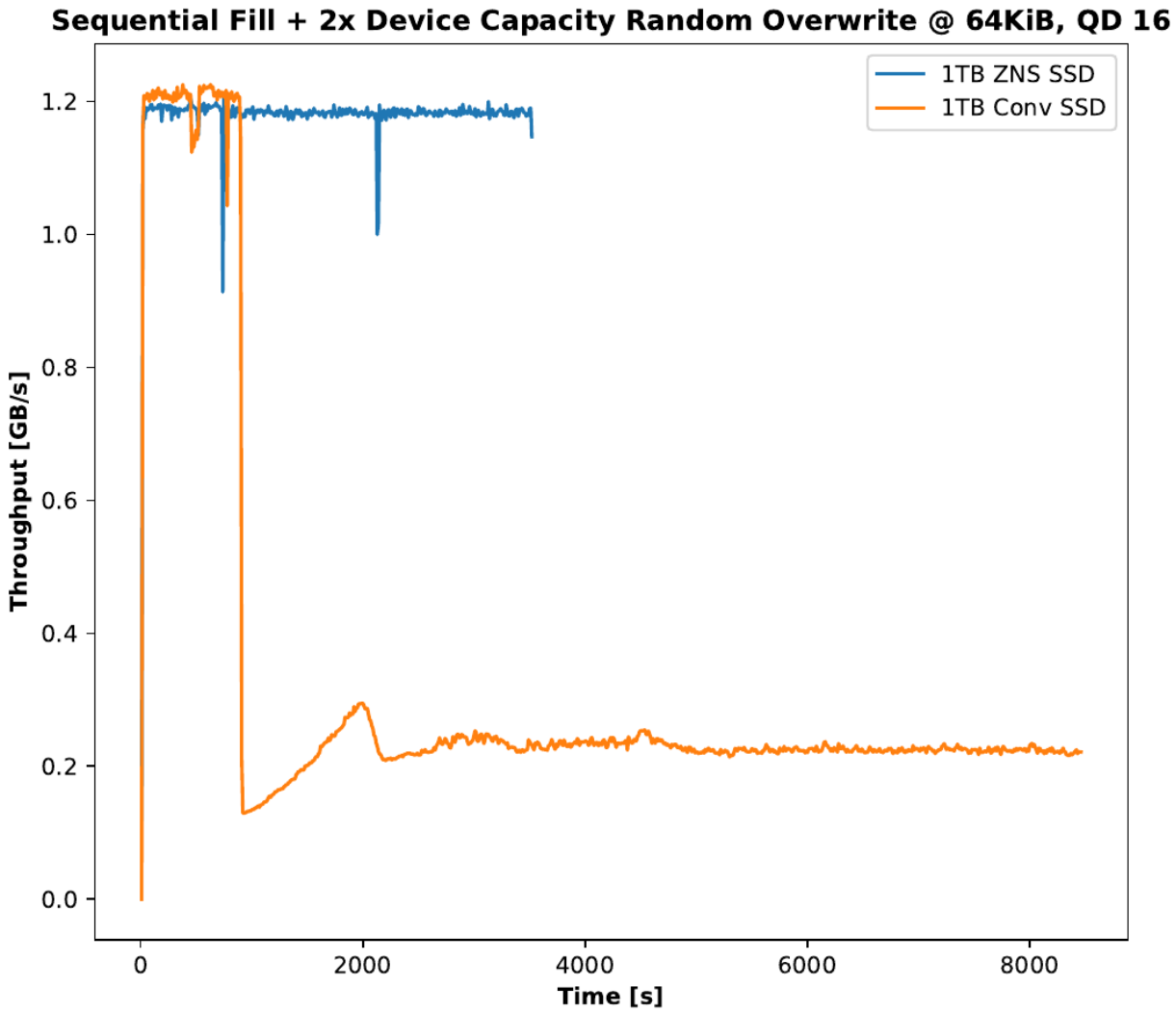
Zoned Block Device

Example workload

SDC 23

Mayastor – fio Overwrite Benchmark

- Prep filled drive capacity sequentially
- 2 full random overwrites @ 64KiB BS and QD 16
- Drives are exposed from a different node via NVMe-oF TCP as a raw block device
- 1TB drive size




```yaml
---
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: percona-server
  name: percona-server
spec:
  initContainers:
  - name: zenfs-setup
    image: docker.io/library/percona:ps-8.0.33-25
    command: ['/bin/bash', '-c', 'rm -rf /tmp-zenfs/zenfs-aux-test && zenfs mkfs --zbd=nvme0n1 --aux_path=/tmp-zenfs/zenfs-aux-test --force']
    volumeMounts:
    - name: mysql-config-directory
      mountPath: /etc/my.cnf.d
    - name: tmp-directory
      mountPath: /tmp-zenfs
    volumeDevices:
    - name: zns-block-device
      devicePath: /dev/nvme0n1
  containers:
  - name: percona-server
    image: docker.io/library/percona:ps-8.0.33-25
    ports:
    - containerPort: 3306
      name: mysql
    volumeMounts:
    - name: mysql-config-directory
      mountPath: /etc/my.cnf.d
    - name: tmp-directory
      mountPath: /tmp-zenfs
    volumeDevices:
    - name: zns-block-device
      devicePath: /dev/nvme0n1
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: "test"
    - name: INIT_ROCKSDB
      value: "yes"
  volumes:
  - name: mysql-config-directory
    hostPath:
      path: /home/debian/k8s-percona-mysql-config
      type: Directory
  - name: mysql-directory
    hostPath:
      path: /home/debian/k8s-percona-mysql
      type: Directory
  - name: tmp-directory
    hostPath:
      path: /home/debian/k8s-percona-tmp
      type: Directory
  - name: zns-block-device
    persistentVolumeClaim:
      claimName: block-device
```

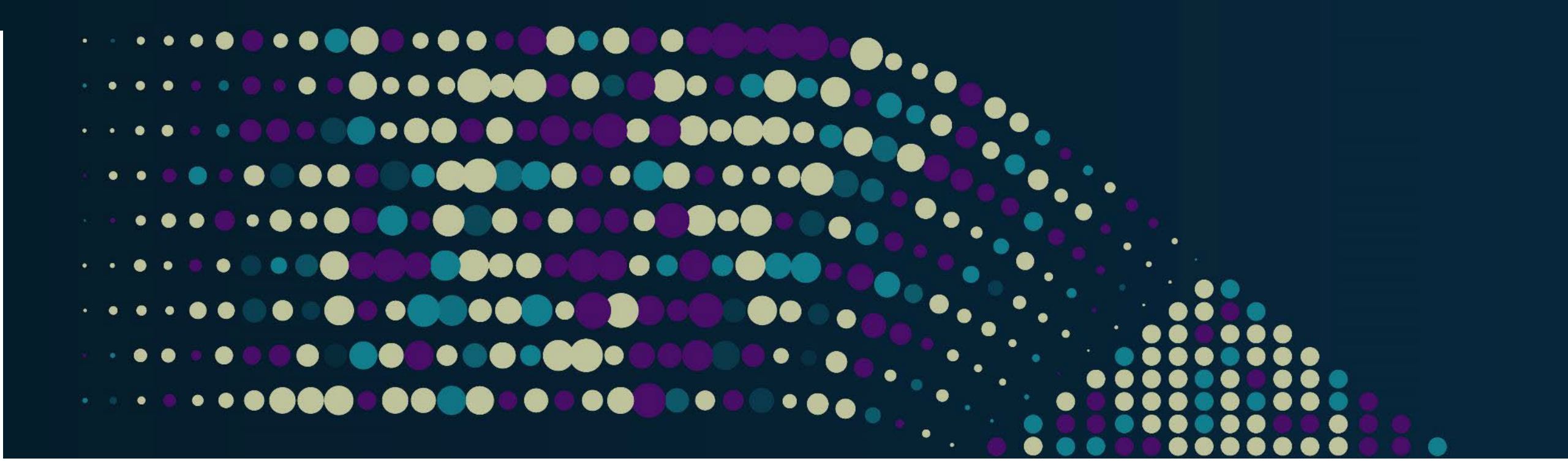
Zoned Block Device

PERCONA Server for MySQL + kubernetes

Practical Example:

- Percona Server for MySQL benefits through the ZenFS plugin from Zoned Storage in a Kubernetes environment
- Native Zoned Storage applications integrate with Zoned Cloud Native Storage solutions!

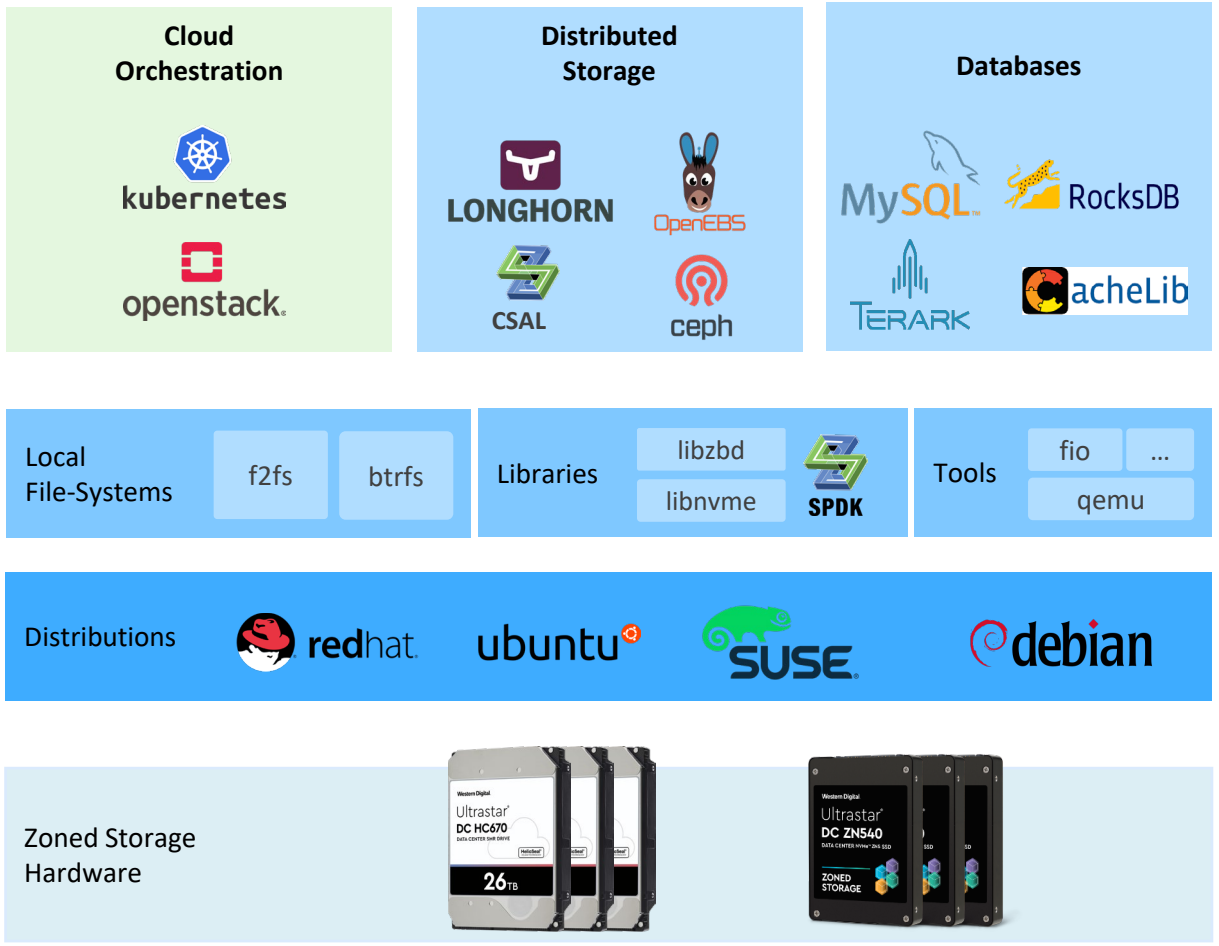
SDC 23



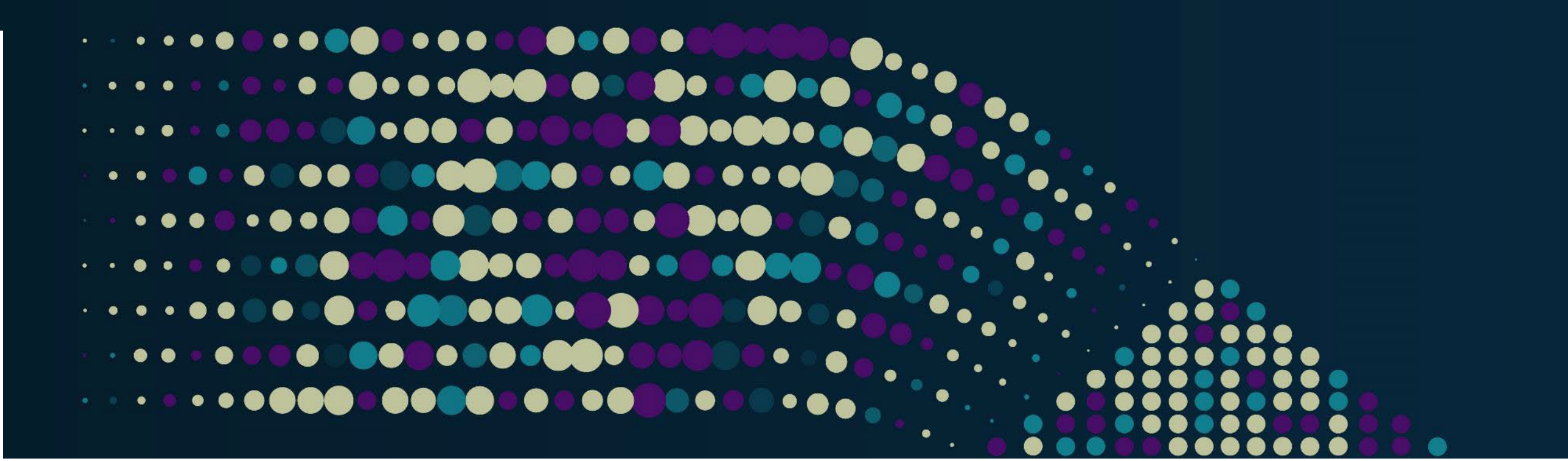
Summary

Summary

- The Linux® zoned storage ecosystem continues to mature
  - BTRFS, Longhorn, Ceph, OpenEBS - Mayastor, CSAL, end-to-end zoned applications like ZenFS
- Workloads orchestrated by Kubernetes are now able to seamlessly benefit from Zoned Storage devices as cost-effective and high-performance storage
- **No longer software stack changes required to deploy and use zoned storage at scale**



SDC 23



Please take a moment to rate this session.

Your feedback is important to us.

