STORAGE DEVELOPER CONFERENCE

SDC 22 | Fremont, CA
September 12-15, 2022

BY Developers FOR Developers

A SNIA Event

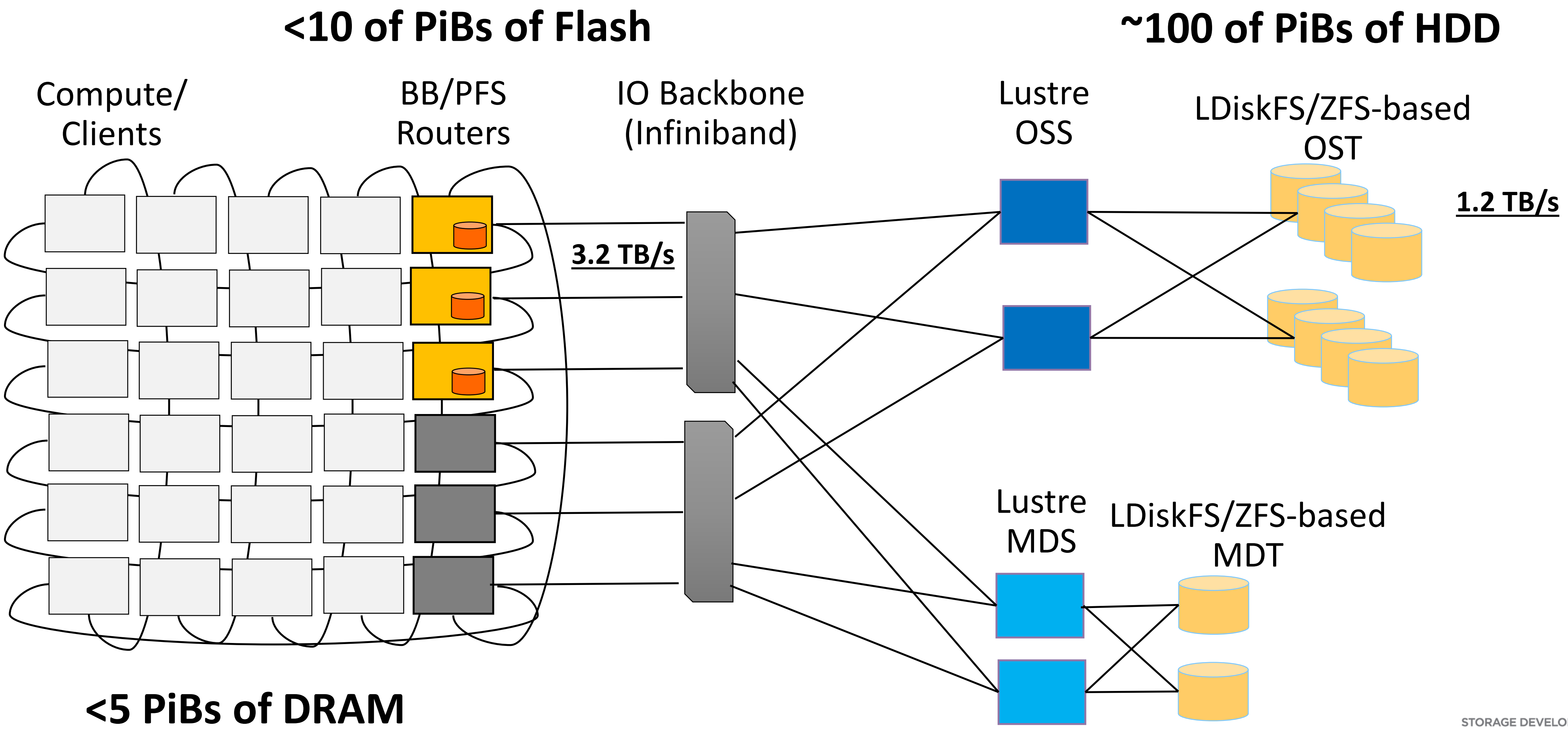
Hardware Accelerated ZFS Using Computational Storage

Software Stack

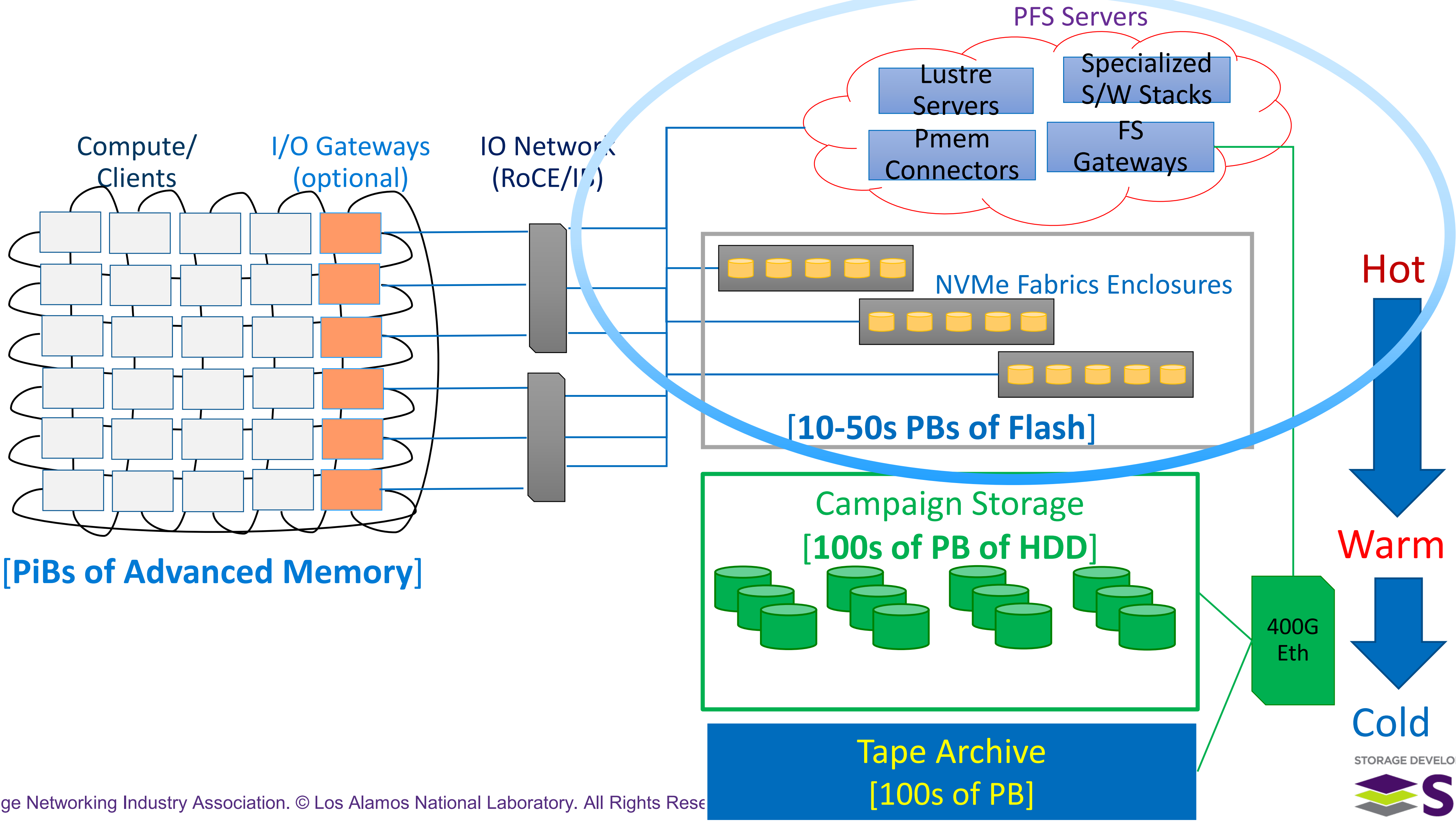
Presented by Jason Lee

LA-UR-22-28873

Current Parallel Filesystem



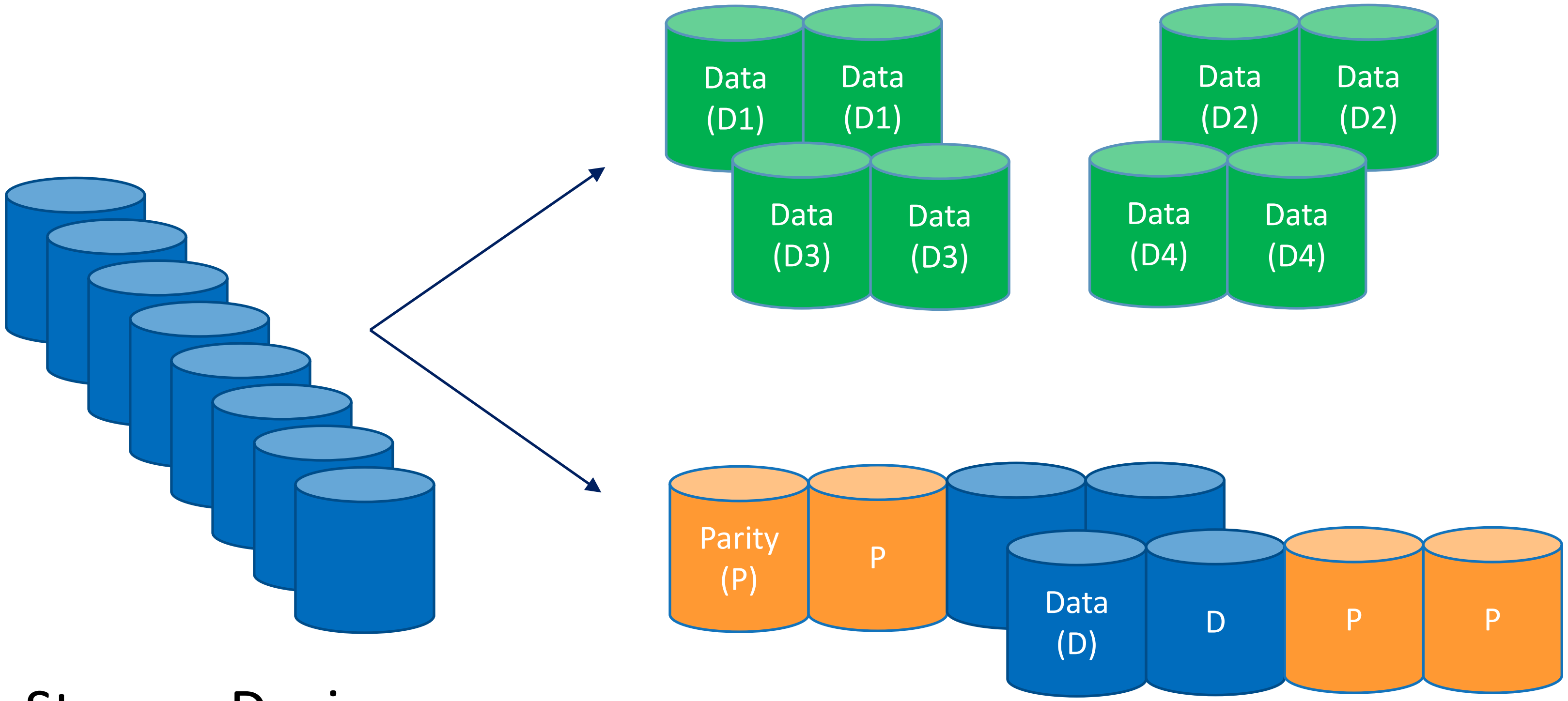
Next Generation Parallel Filesystem



Why Does LANL Care about ZFS?

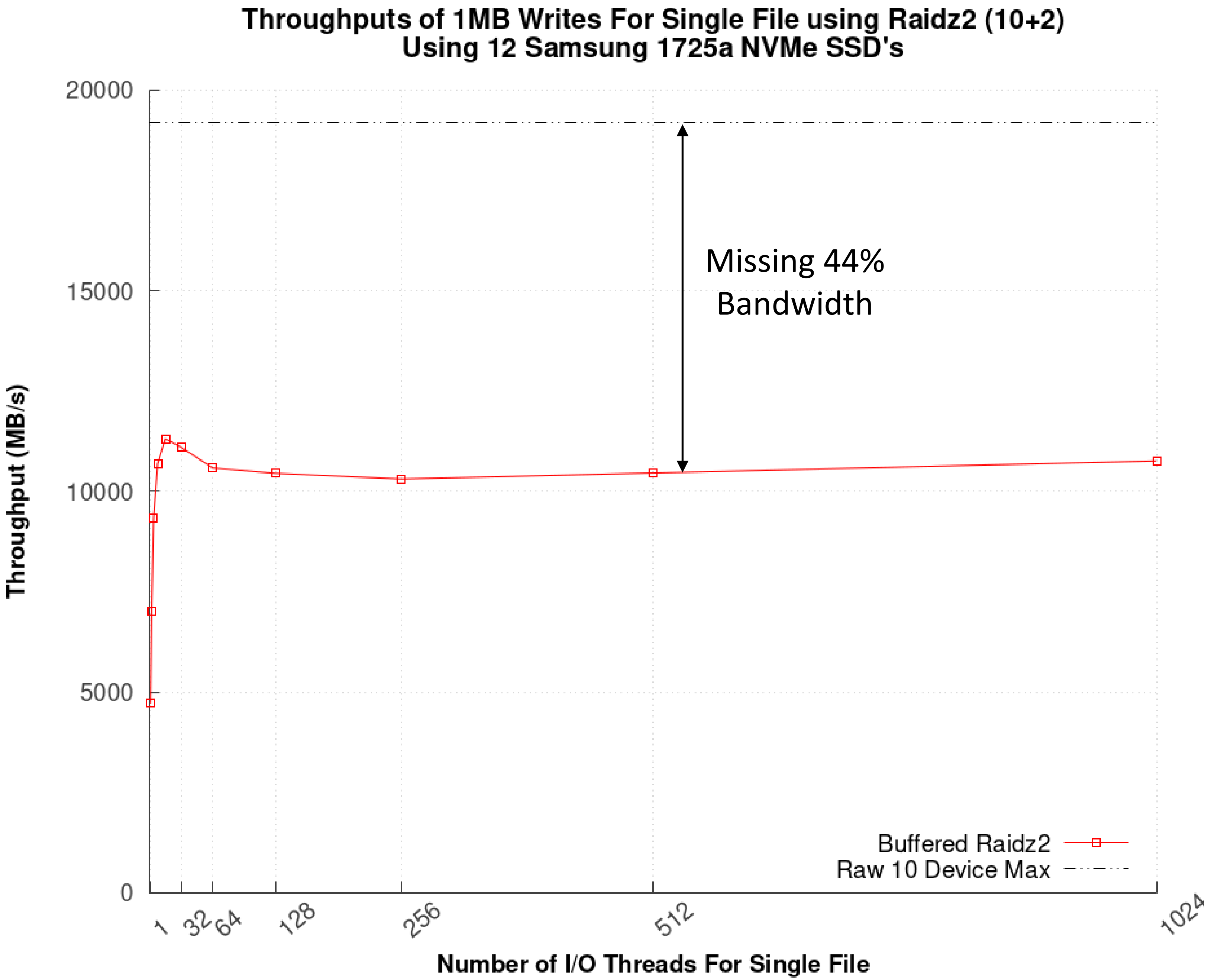
- **One of two available backing FS's for <u>Lustre</u>**
- Open source
- High integrity
  - Erasure coding (RAIDZ)
  - Mirrors
  - Checksums
  - Snapshots
- Feature rich
  - Encryption
  - Dedup
  - Compression
- Volume Manager

4 x 2-Way Mirrors

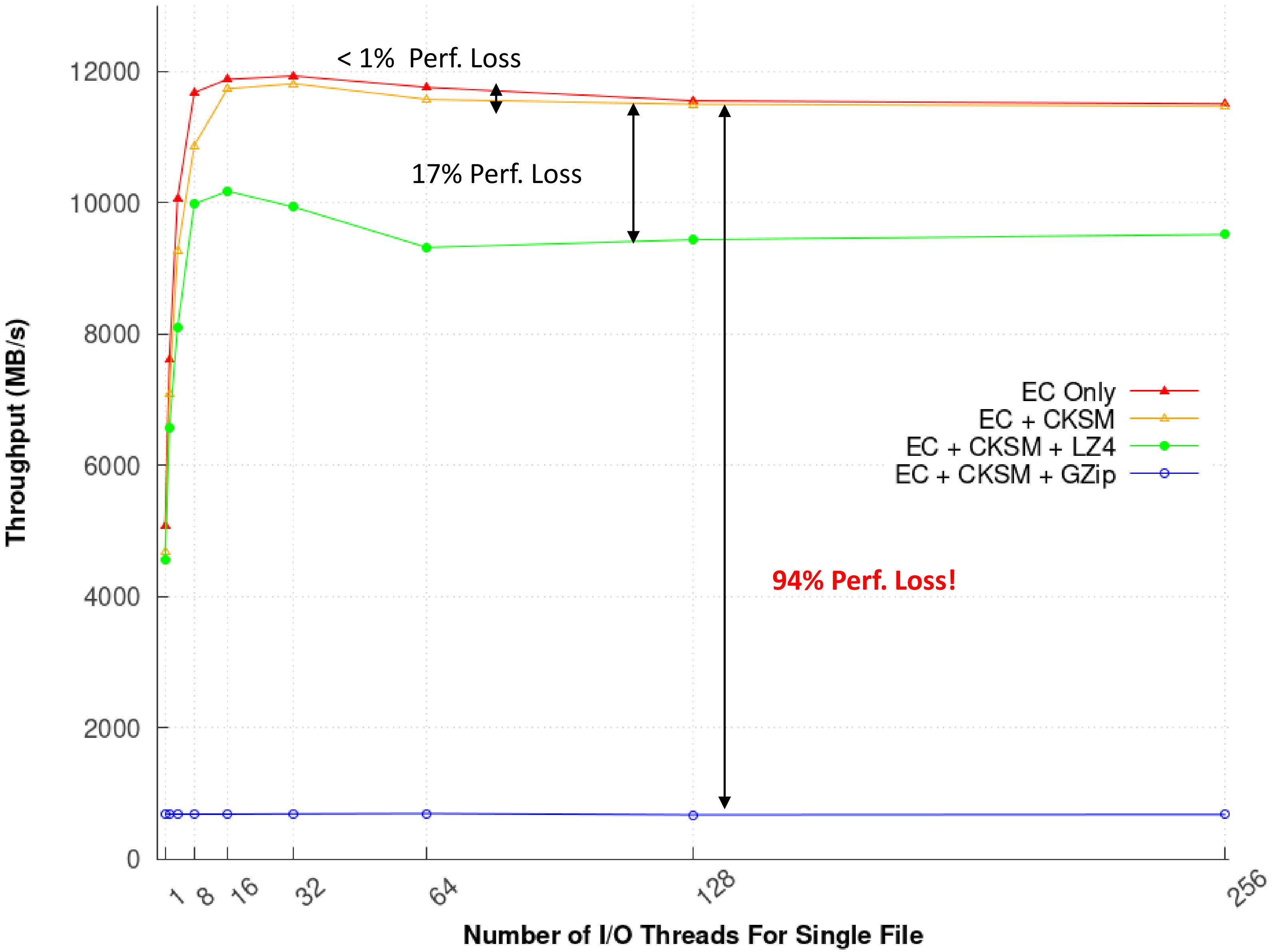


Storage Devices

2 x RAIDZ2 (2 + 2)


Throughputs of 1MB Writes For Single File using Raidz2 (10+2)
Using 12 Samsung 1725a NVMe SSD's
Throughput (MB/s)
20000
15000
10000
5000
0
1
32
64
128
256
512
1024
Number of I/O Threads For Single File
Missing 44% Bandwidth
Buffered Raidz2
Raw 10 Device Max

Throughputs of 1MB Writes For Single File Using ZFS Raidz2 (10+2) Using NVMe-oF from Host to Target



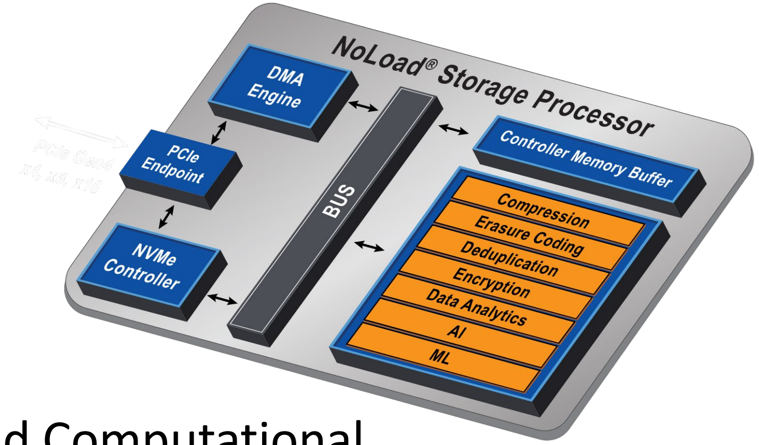
< 1% Perf. Loss

17% Perf. Loss

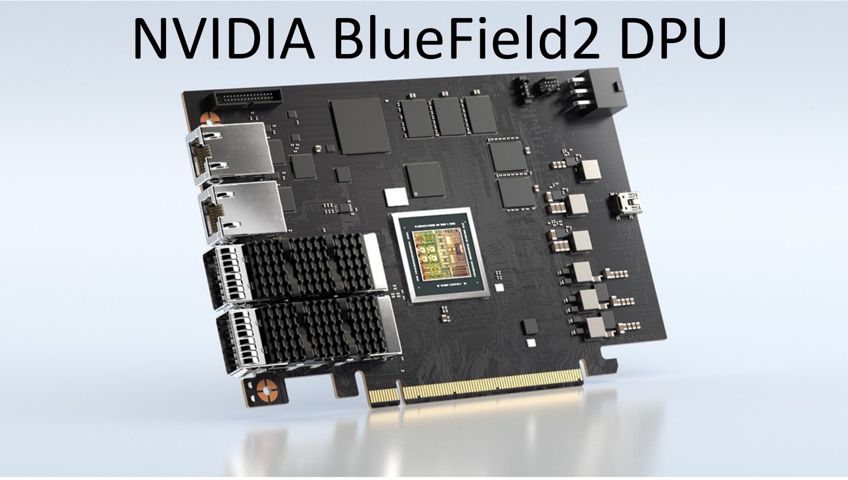
94% Perf. Loss!

What can we do to improve performance?

- Use computational storage to offload operations
  - Perform operations that are CPU/memory bandwidth intensive when run on host
  - Can be implemented with FPGAs
  - Data Processing Unit (DPU)



Eideticom Noload Computational Storage Processor (CSP)



NVIDIA BlueField2 DPU

Doesn’t ZFS already support offloading?

- Intel® QuickAssist Technology (Intel® QAT)
  - Doesn’t work on AMD machines

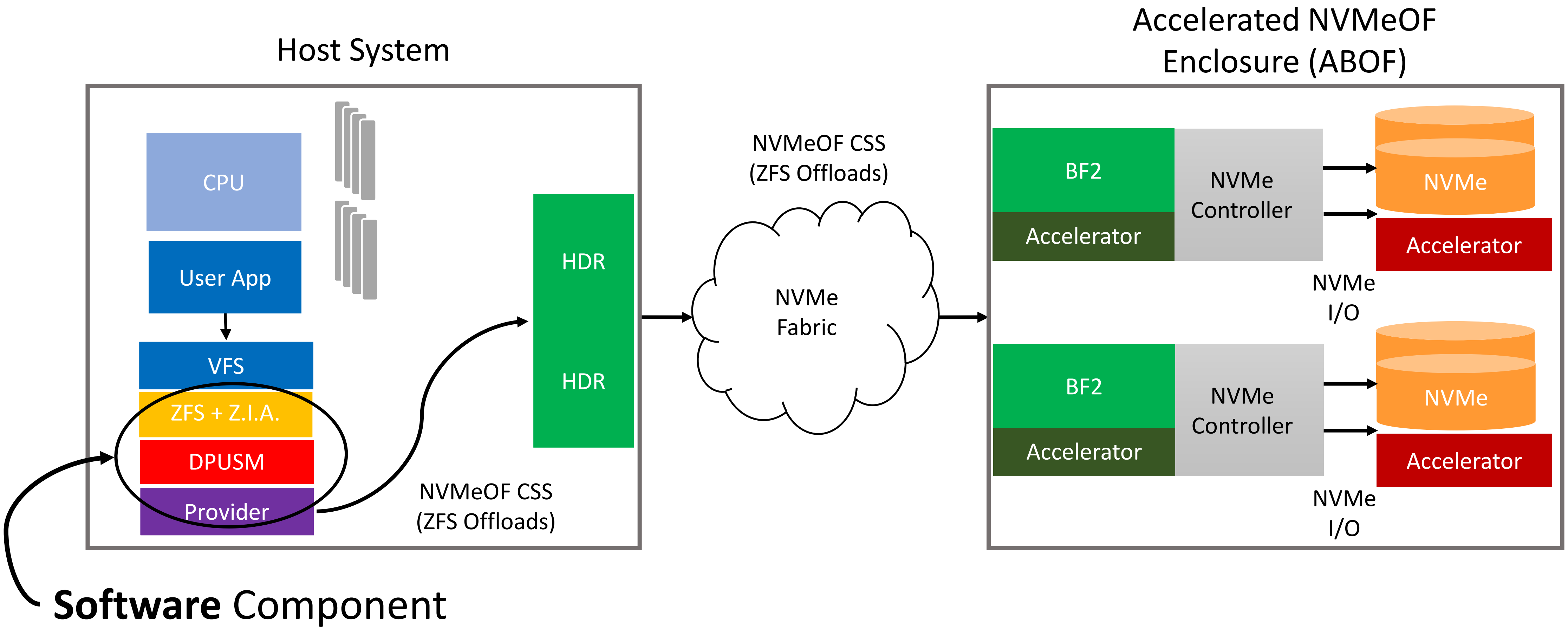
- Requires ZFS to be reconfigured

- Each offload operation is done independently of each other
  - Encryption – AES-GCM
  - Compression – GZIP
  - Checksum – SHA256

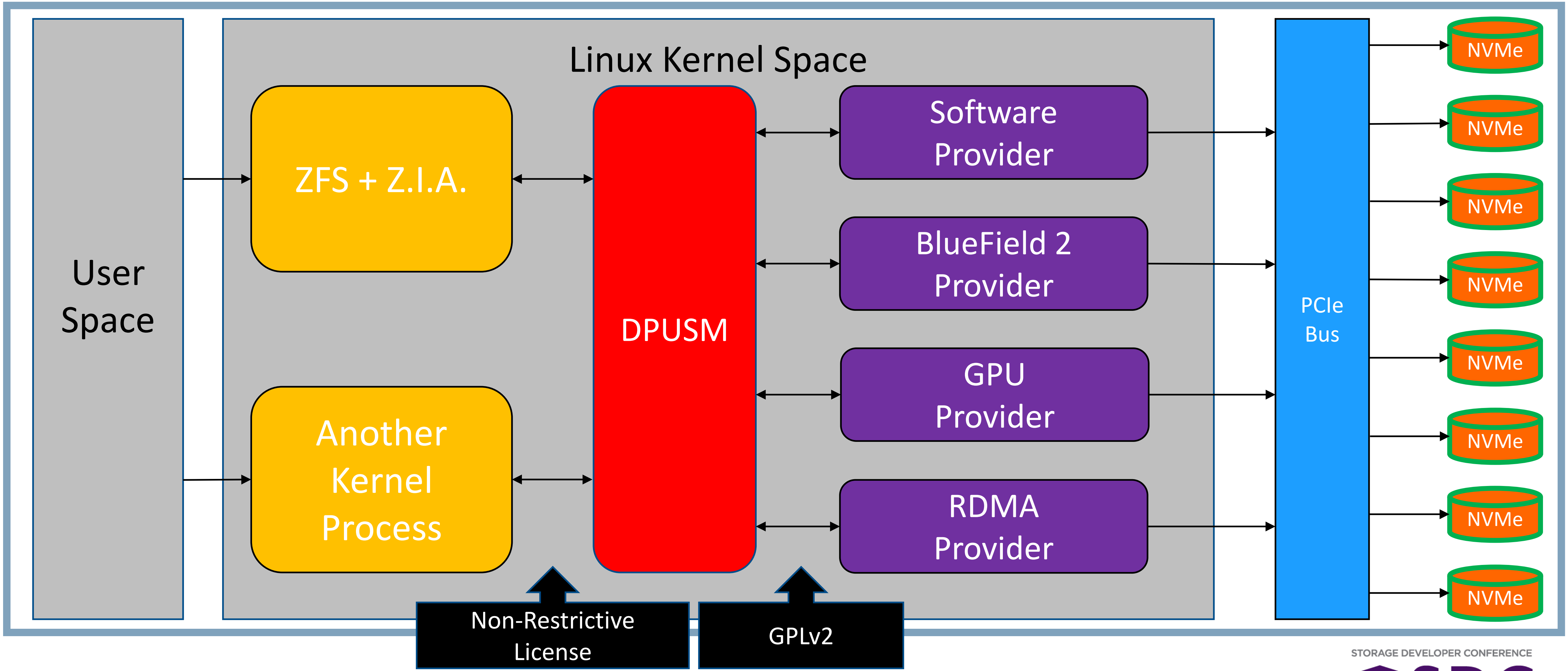
- Not extensible

Accelerated Box of Flash and ZFS Interface for Accelerators

Accelerated ZFS with Disaggregated Storage

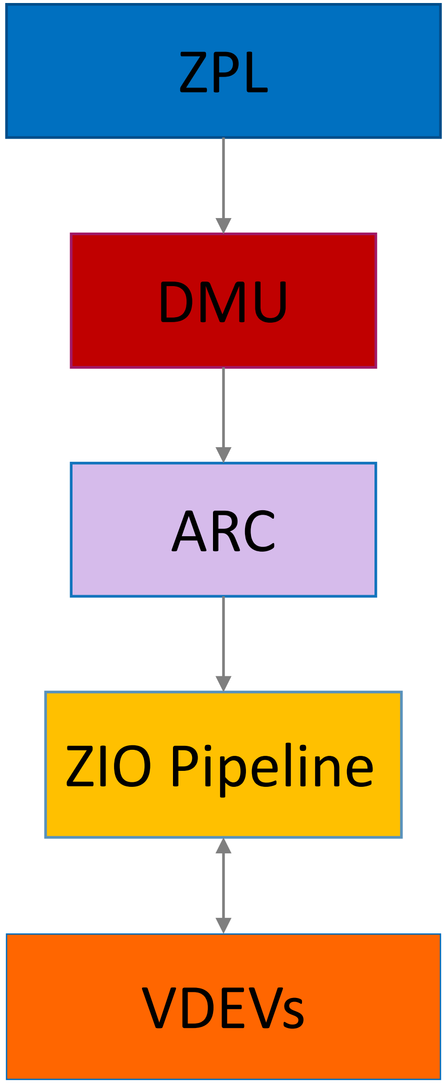


Accelerated ZFS with Converged Storage



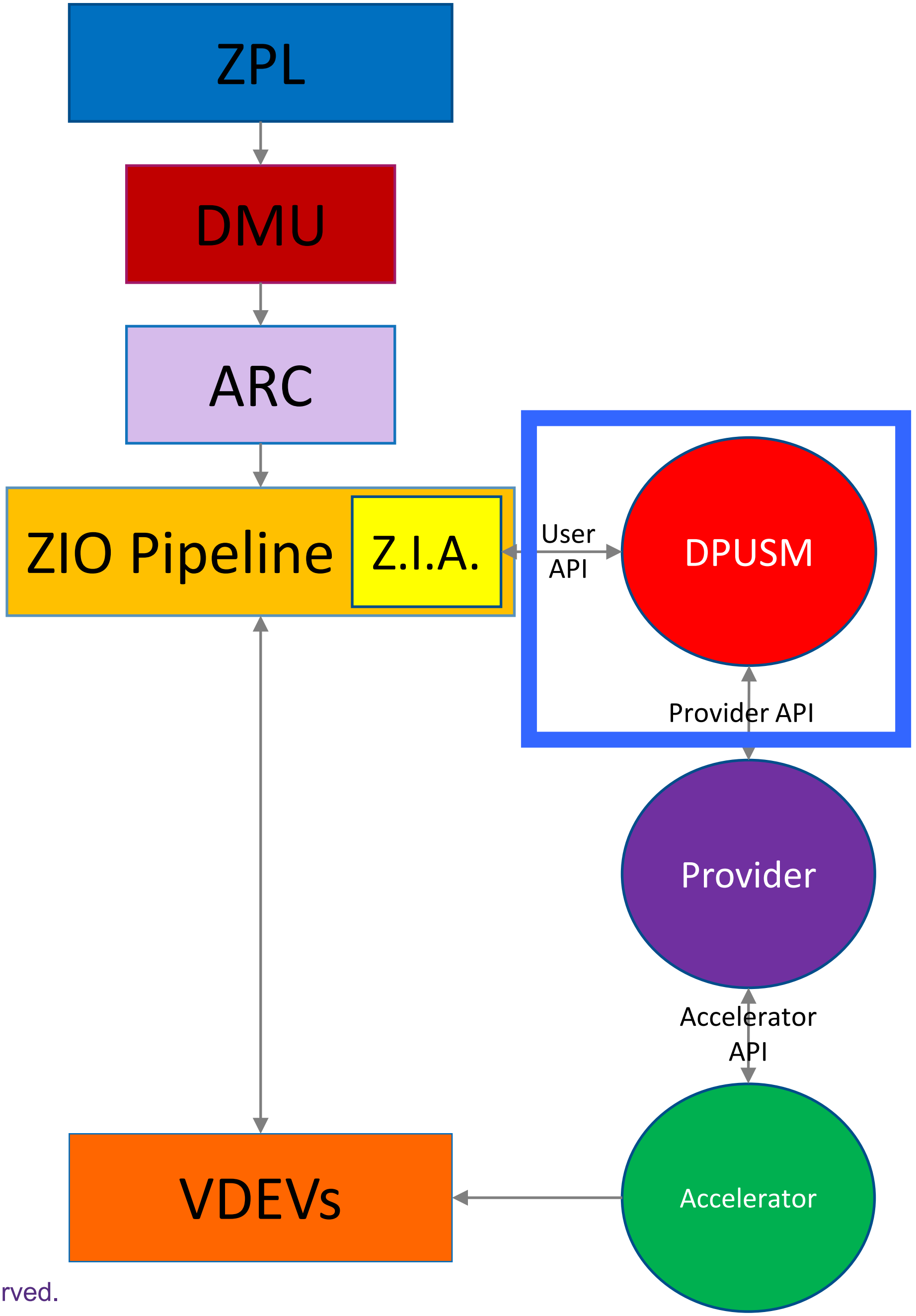
The Software

ZFS Write Pipeline



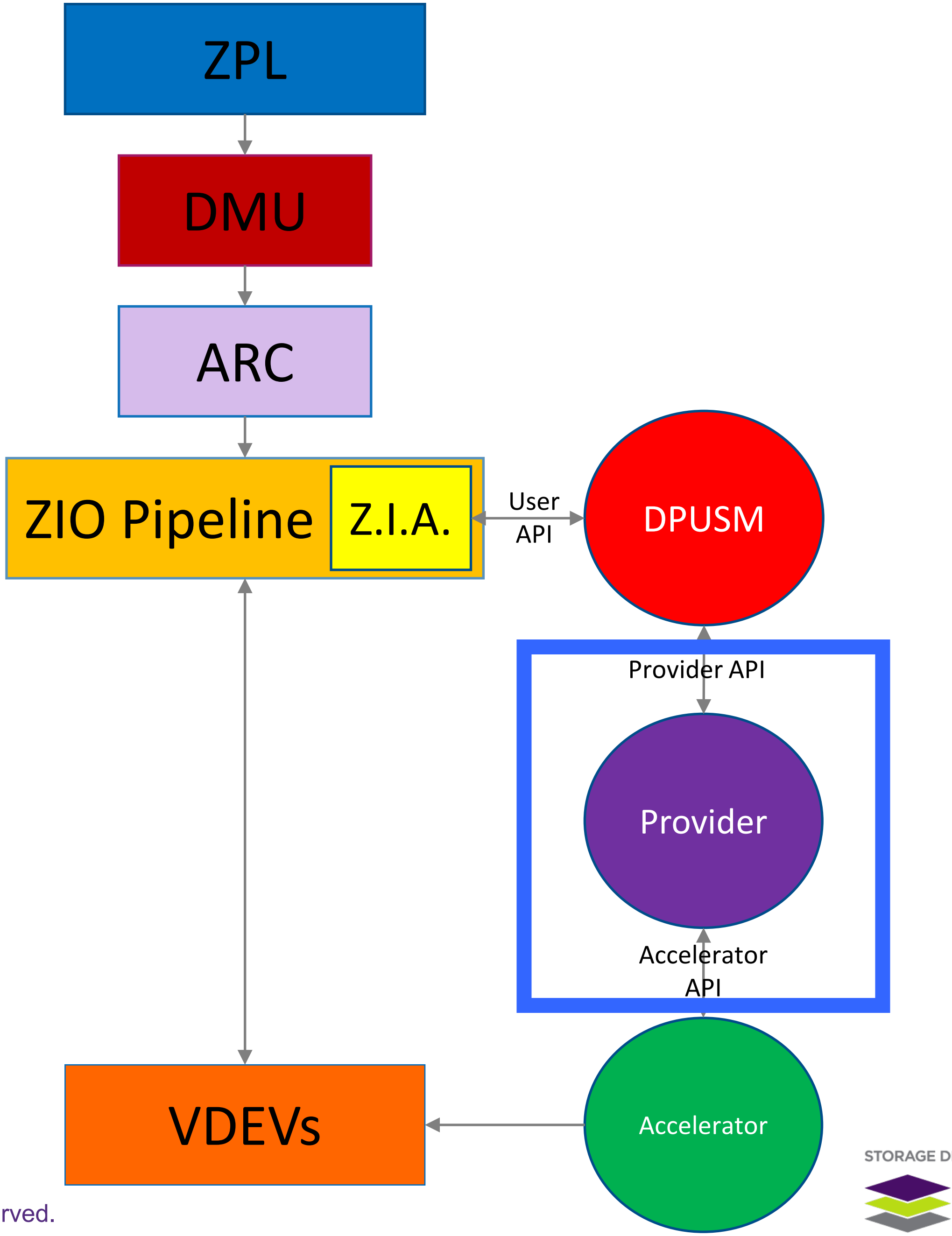
Data Processing Unit Services Module (DPUSM)

- Kernel module
- Standardized APIs for leveraging computational storage
  - Provider API
  - User API
- Acts as registry for providers



Providers

- Kernel module
- Usually implemented by accelerator vendor
- DPUSM wrapper for accelerator specific code
- Declares what the accelerator provides



Provider Implementation Basics

- `#include <accelerator_header.h>`
- `#include <dpusm/provider_api.h>`
- Fill in DPUSM provider functions struct
  - Analogous to VFS function pointers
- Register provider with DPUSM on module initialization

1. Give user handle that references accelerator memory
2. Get user (in-memory) data into accelerator (copy, rdma, etc.) via handles
3. Accept handles for operations

- Communication with accelerator is connection protocol agnostic

Using a provider

- `#include <dpusm/user_api.h>`
- Find provider
- Use provider functions in DPUSM user functions struct

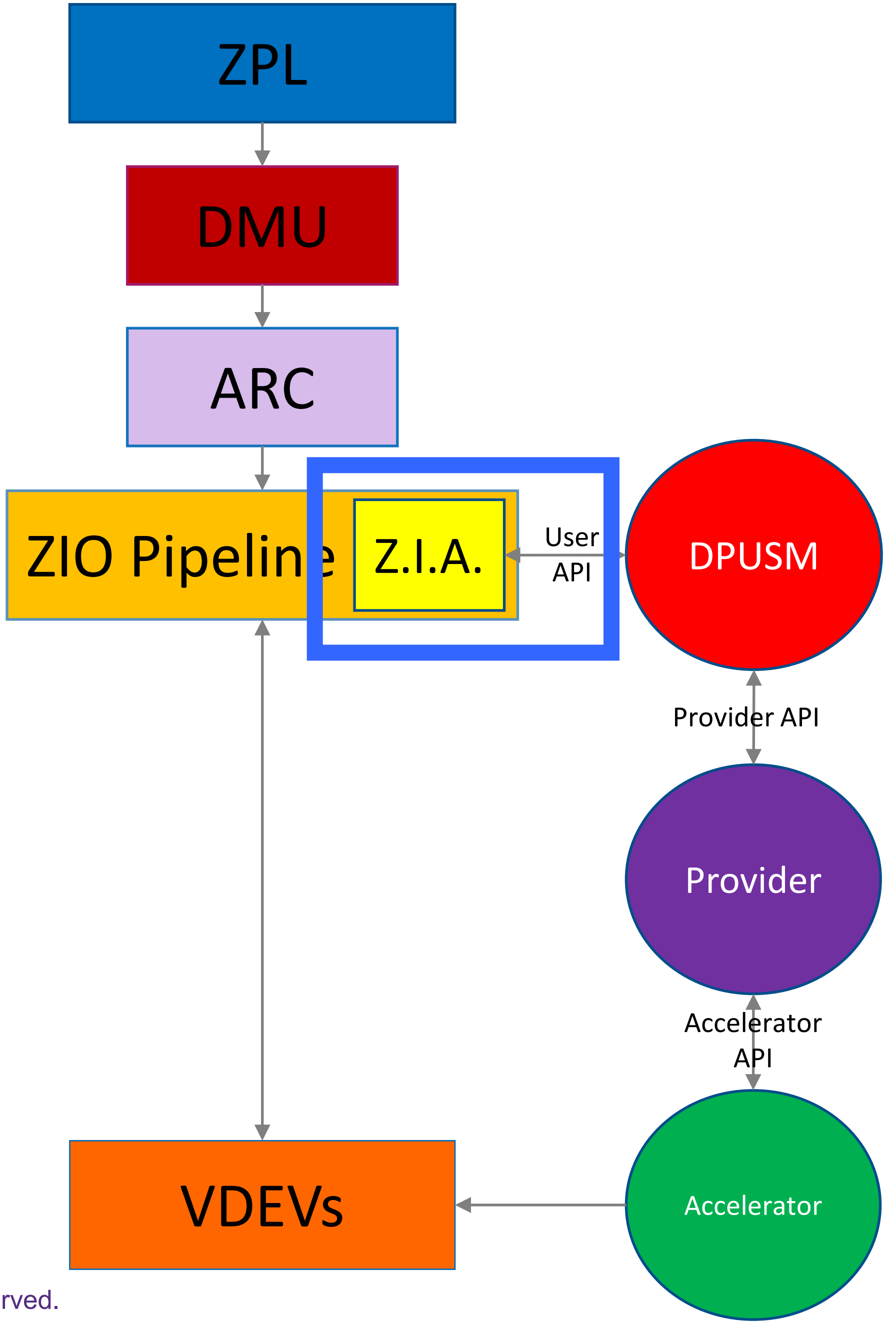
1. Get opaque handle (`void *`) to accelerator memory (wrapped by provider)
2. Get in-memory data to accelerator via handle
3. Pass handle(s) to provider functions to operate on data

ZFS Interface for Accelerators (Z.I.A.)

- Modifications to the ZFS **write** pipeline

- Transparent acceleration of CPU and memory intensive ZFS write operations with accelerators
  - Compression
  - Checksum
  - RAIDZ (Generation and Reconstruction)
  - I/O

- User data access not affected
  - During write
  - Afterwards



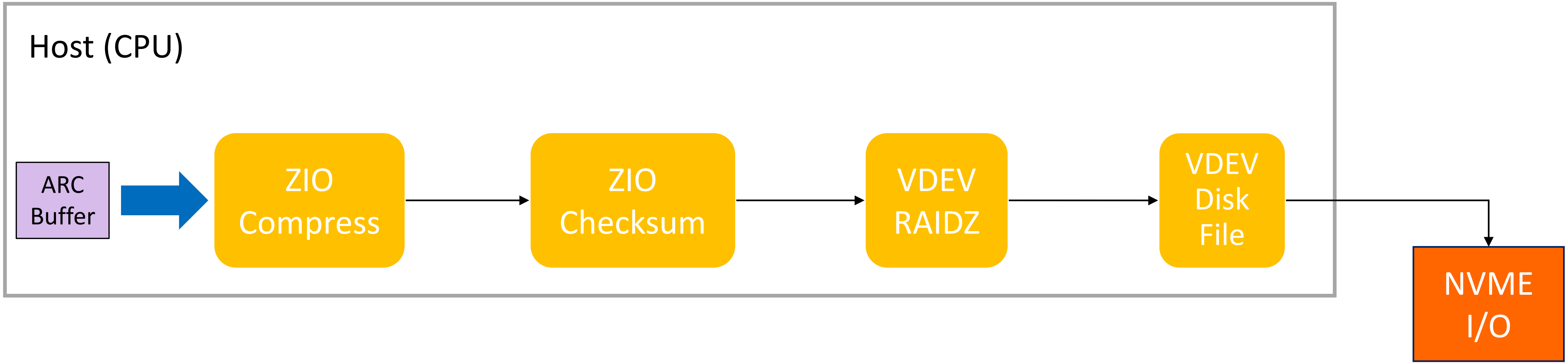
Z.I.A. Usage (Admins)

- Currently need to reconfigure ZFS with `--with-zia=<DPUSM Root>`
  - Expect that ZFS will always compile Z.I.A. once merged
  - Z.I.A. will not cause issues if DPUSM is not found at load time

- Select a provider
  - `zpool set zia_provider=<provider name> <zpool>`

- Enable offloading
  - `zpool set zia_<property>=on <zpool>`
  - Offloading only occurs if the ZFS stage is enabled

ZFS Write Pipeline

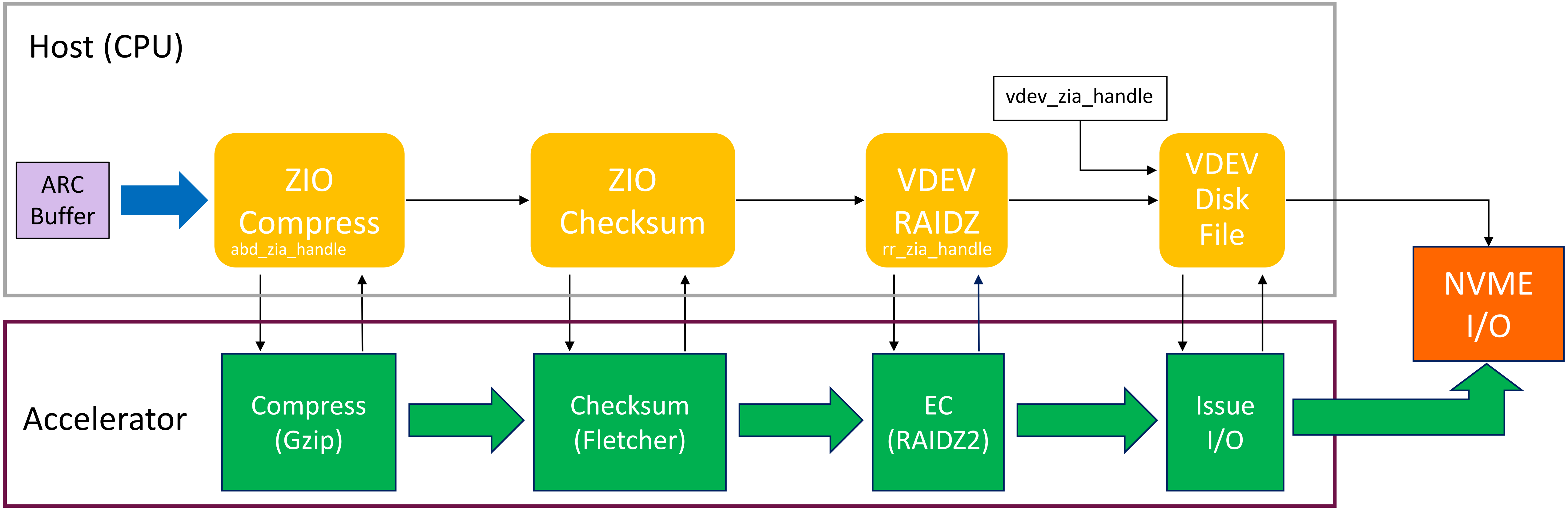


General Description of Changes

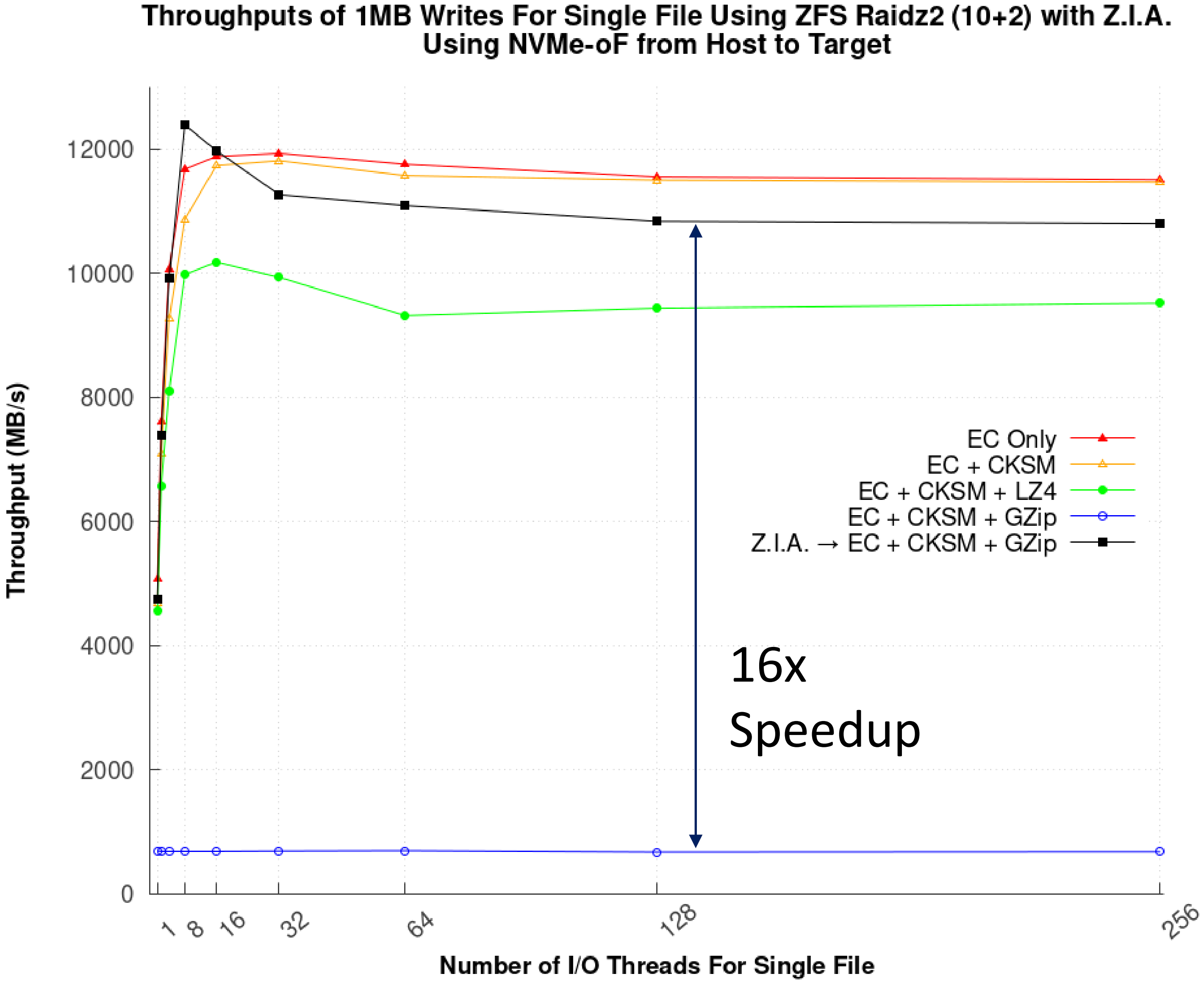
- If data is not offloaded at start of stage, offload it
- Run the operation
- Return status code (not data)

- If Z.I.A. fails, bring data back to memory, fall back to running operation in software
- If offloaded data cannot be returned to memory, restart write pipeline
  - A copy of the original data is still available in ZFS
  - Not implemented yet

Z.I.A. Write Pipeline



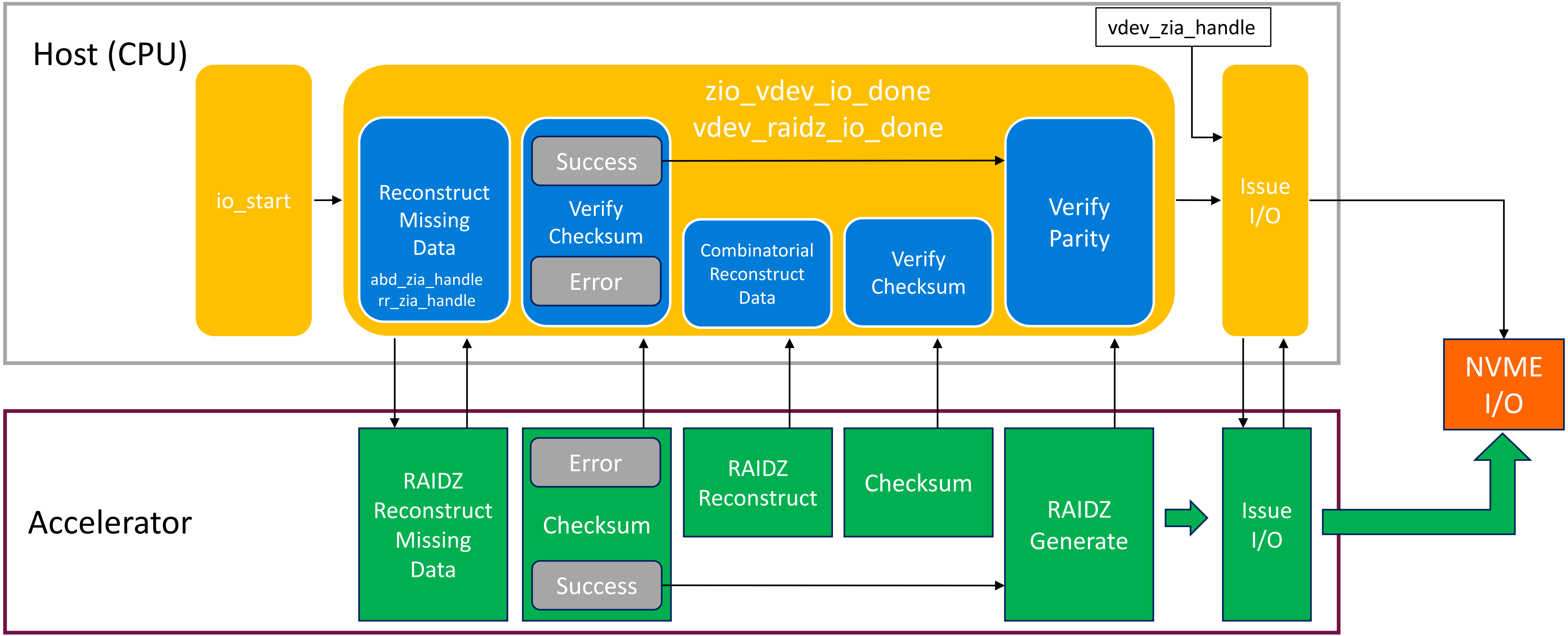
Z.I.A. Performance with Eideticom NoLoad



Throughputs of 1MB Writes For Single File Using ZFS Raidz2 (10+2) with Z.I.A.
Using NVMe-oF from Host to Target

16x
Speedup

Z.I.A. Resilver



More Information

- Z.I.A. Pull Request
  - https://github.com/openzfs/zfs/pull/13628

- Data Processing Unit Services Module
  - https://github.com/hpc/dpusm

- Direct I/O Pull Request
  - https://github.com/openzfs/zfs/pull/10018

Please take a moment to rate this session.

Your feedback is important to us.