SDC 20

BY Developers FOR Developers

Storage Developer Conference

September 22-23, 2020

NoLoad Filesystem: A Stacked Filesystem for NVMe Computational Storage

David Sloan, Logan Gunthorpe, Stephen Bates

EIDETICOM

Topics

- Introduction: The case for compression
- NoLoad NVMe-Based CSx
- Transparent Compression using an NVMe CSx
- Stacked Filesystems
- NoLoad Filesystem
    - Compression Method
    - Storage Architecture
    - Interface
    - Performance

Introduction

Introduction: The Case For Compression

- Data volumes are exploding
- NAND is getting cheaper but not as cheap as HDDs.
- NAND provides x1000 the performance of HDDs wrt IOPS, throughput and latency
- Compression can bridge the cost gap
- But it has to be performant, efficient and easy to consume!

EIDETICOM

NoLoad
An NVMe-Based Computational Storage Device

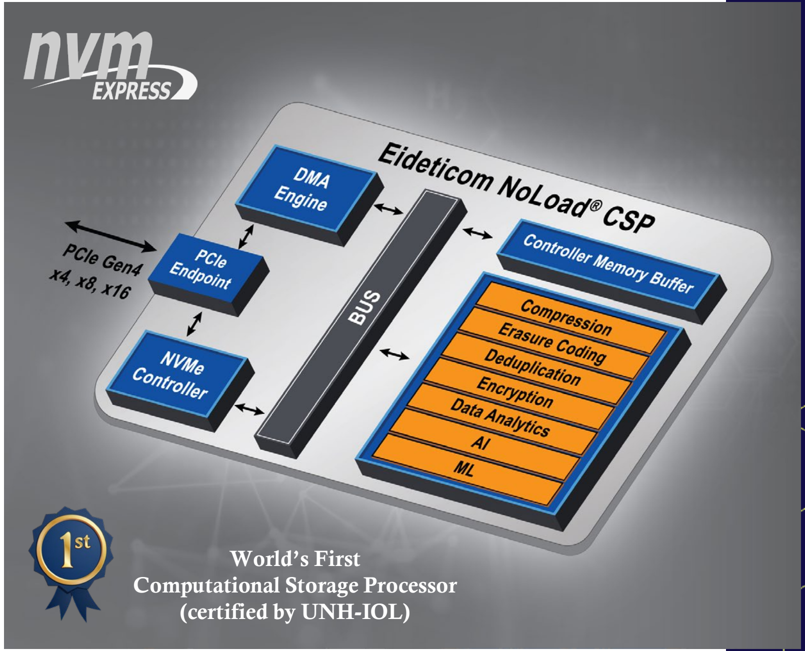
EIDETICOM

NoLoad® Computational Storage Device (CSx)

Eideticom's NoLoad® CSx

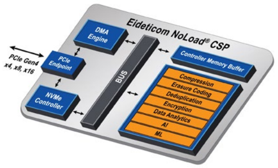
Purpose built for acceleration of storage and compute-intensive workloads

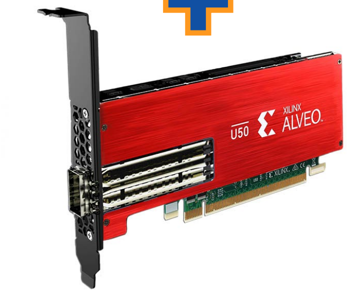
- **NoLoad Software Stack**
  - End-to-end computational storage solution providing transparent computational offload
  - Complete Software and IP core stack
- **NoLoad NVMe Front End**
  - NVMe compliant, standards-based interface
  - High performance interface tuned for computation
- **NoLoad Computational Accelerators**
  - Storage Accelerators: Compression, Encryption, Erasure Coding, Deduplication
  - Compute Accelerators: Data Analytics, Video Codec, AI and ML



SNIA Computational Storage Terminology

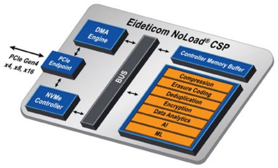
Computational Storage Processor (CSP)
A component that provides computational services to a storage system without providing persistent storage





NoLoad on Xilinx Alveo U50

Computational Storage Drive (CSD)
A component that provides persistent data storage and computational services



NoLoad on Samsung SmartSSD

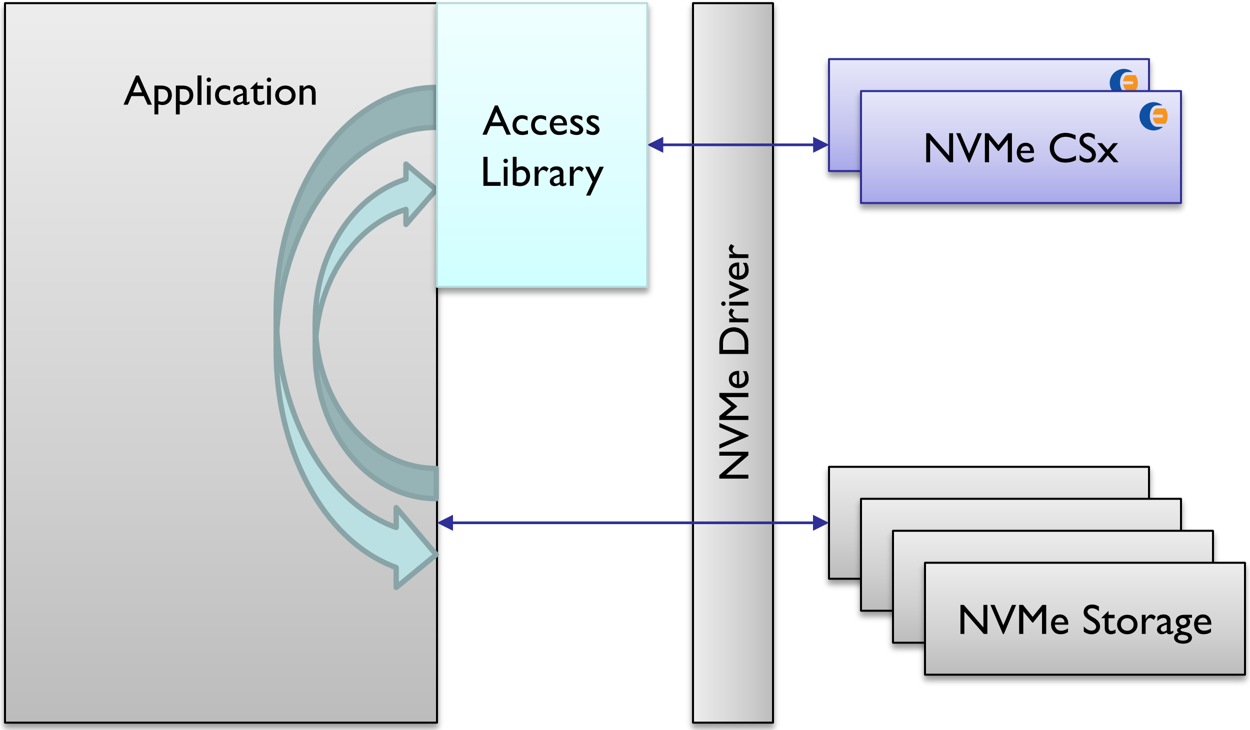
Transparent Compression using an NVMe CSx

EIDETICOM

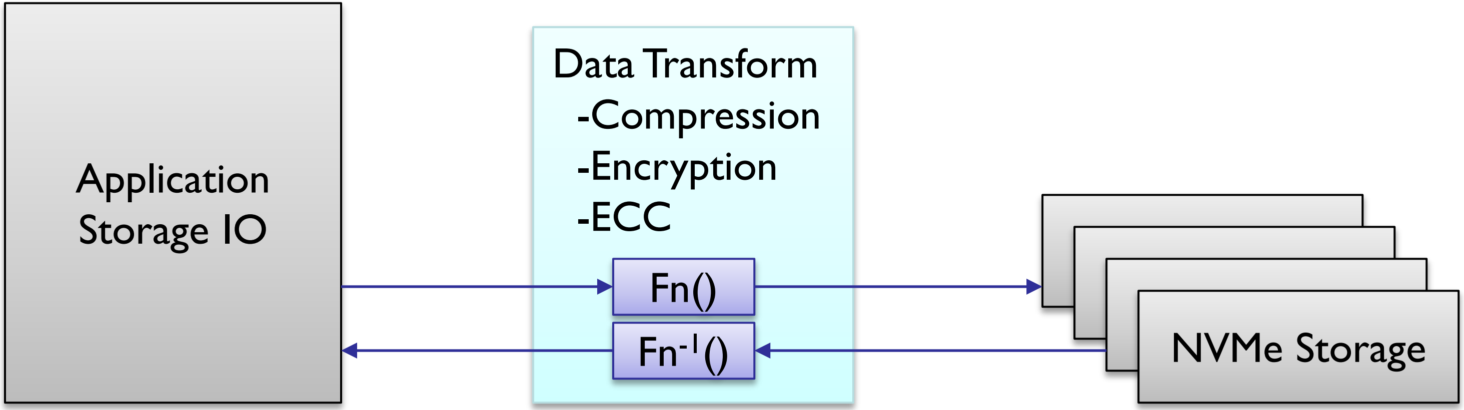
Transparent Compression

- Transparent compression is compression that **the *application* is unaware of**.
- This can happen in one of four places:
  - On the device (multiple vendors)
  - In the block layer (e.g. VDO)
  - In the filesystem (e.g. ZFS)
  - In a stacked filesystem (e.g. this work)

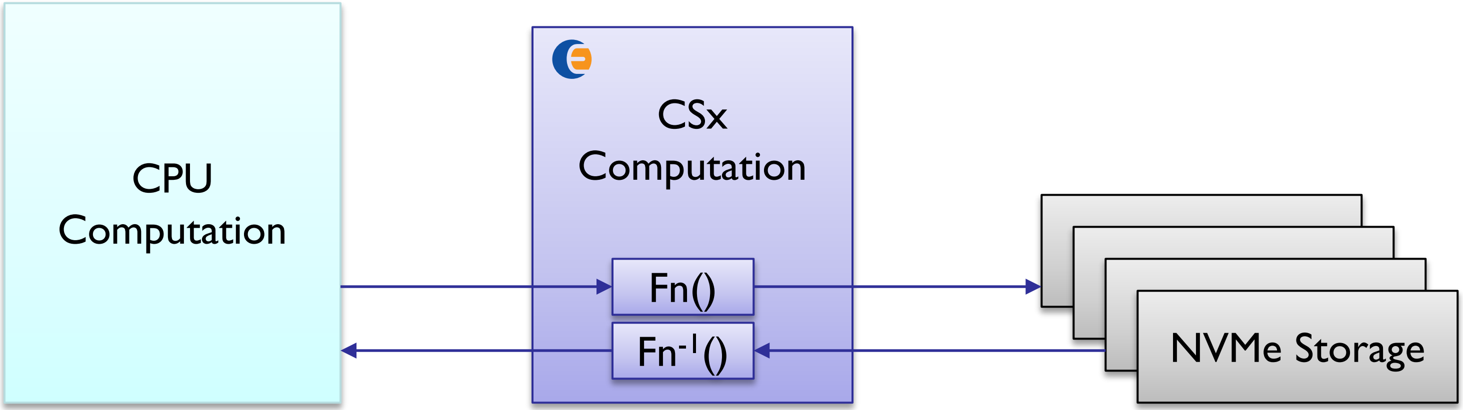
CSx as an Accelerator



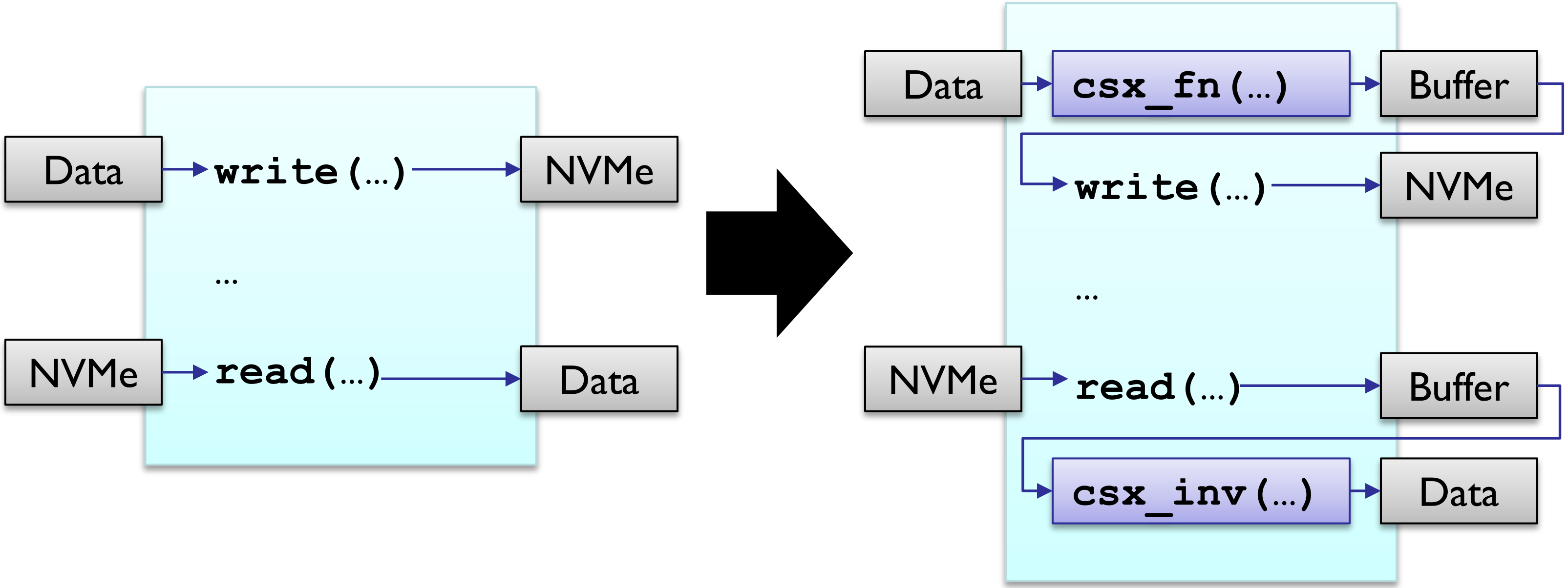
Transparent Computation



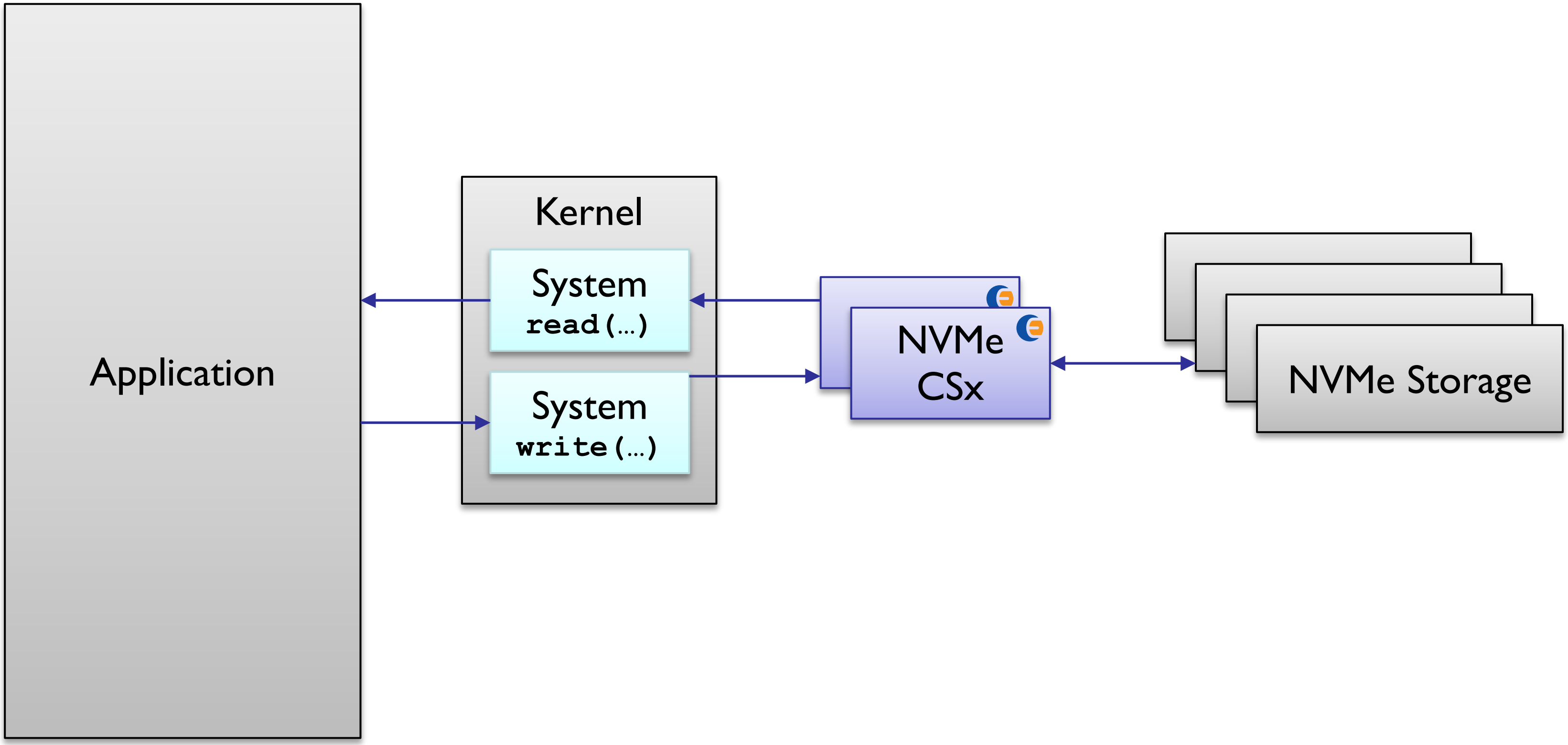
Transparent Compression



Non-Transparent Compression



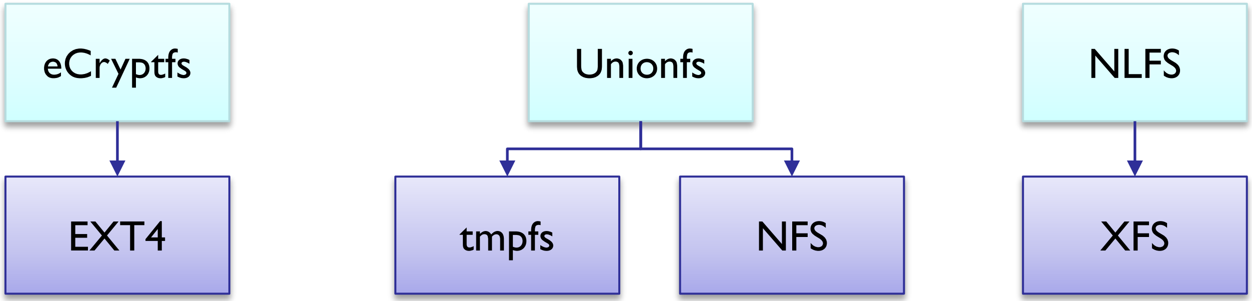
Transparent Compression



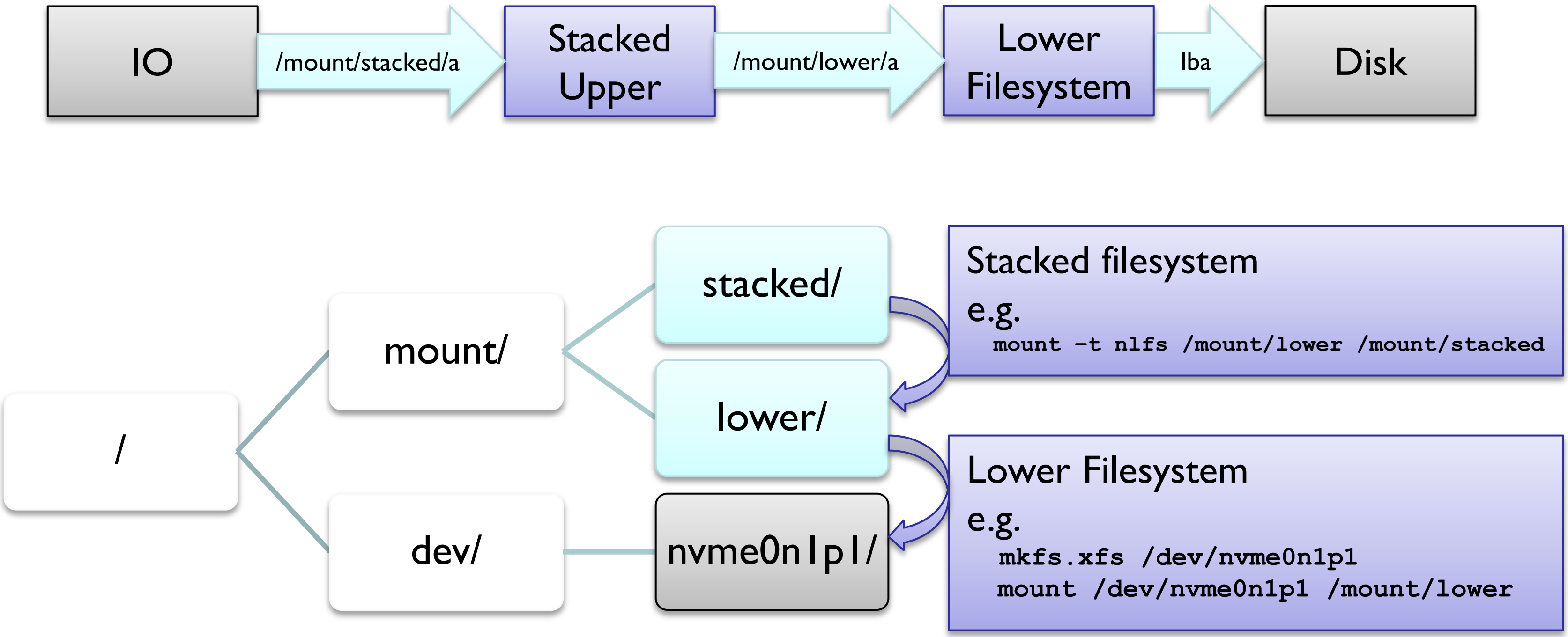
Stacked Filesystems

Stacked Filesystems

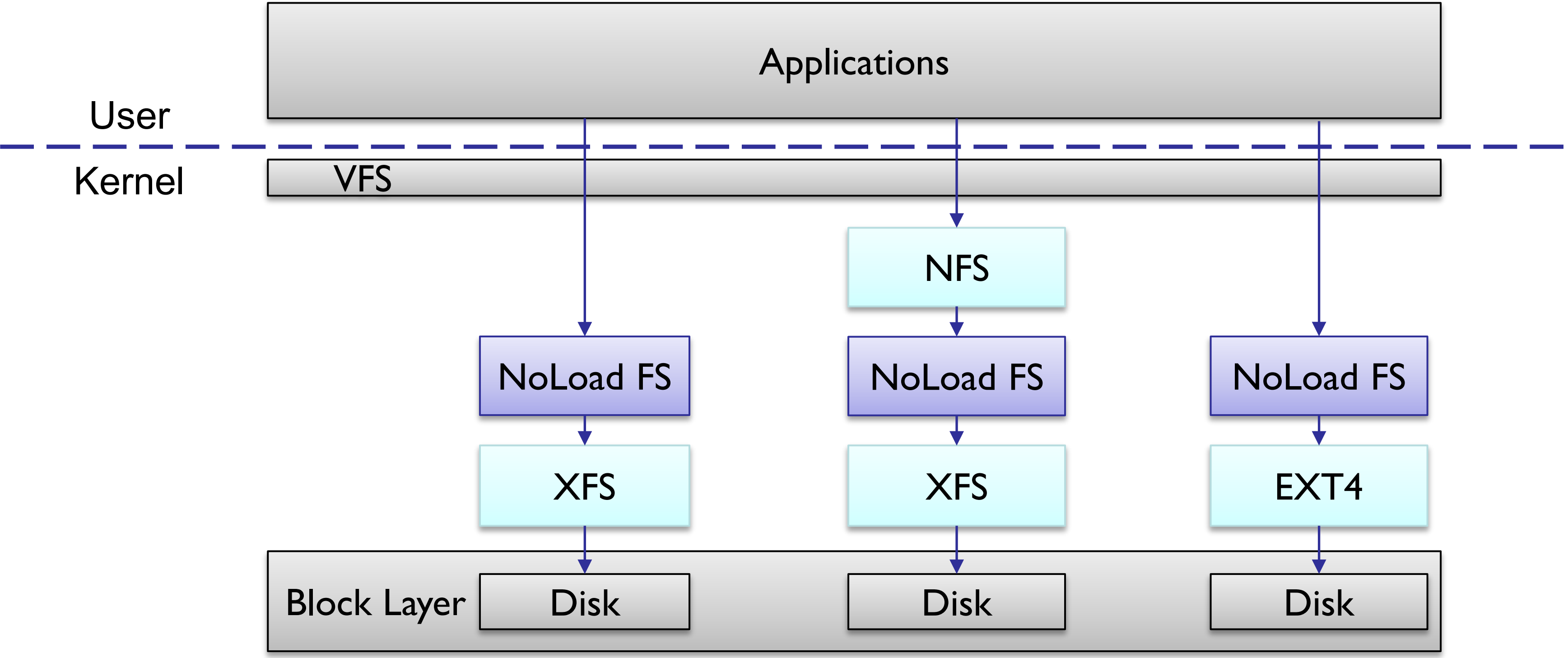
- Provide a code injection point between applications and the filesystem
- Adds functionality to existing filesystems
- Easy to integrate into existing SW stacks!



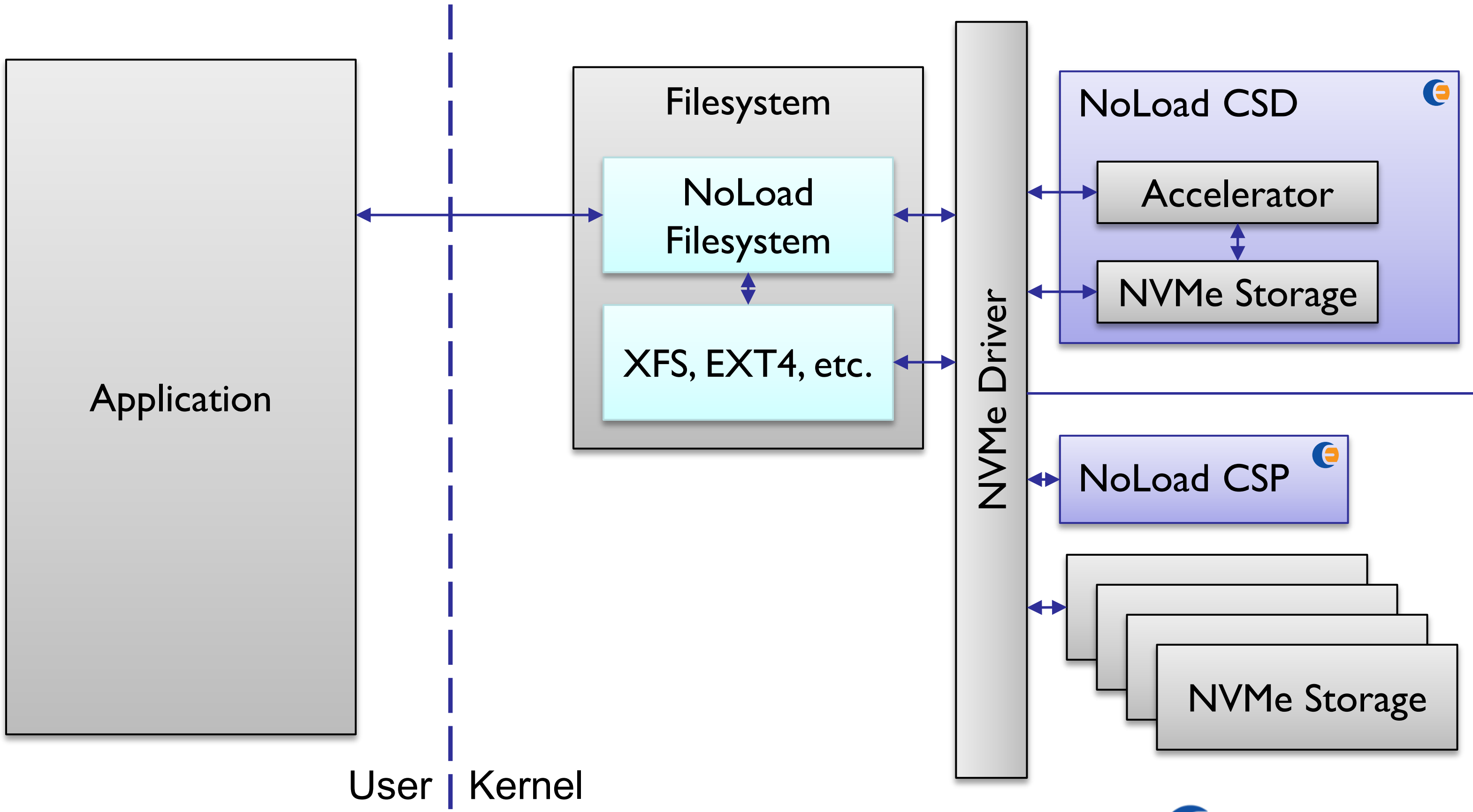
Stacked Filesystem



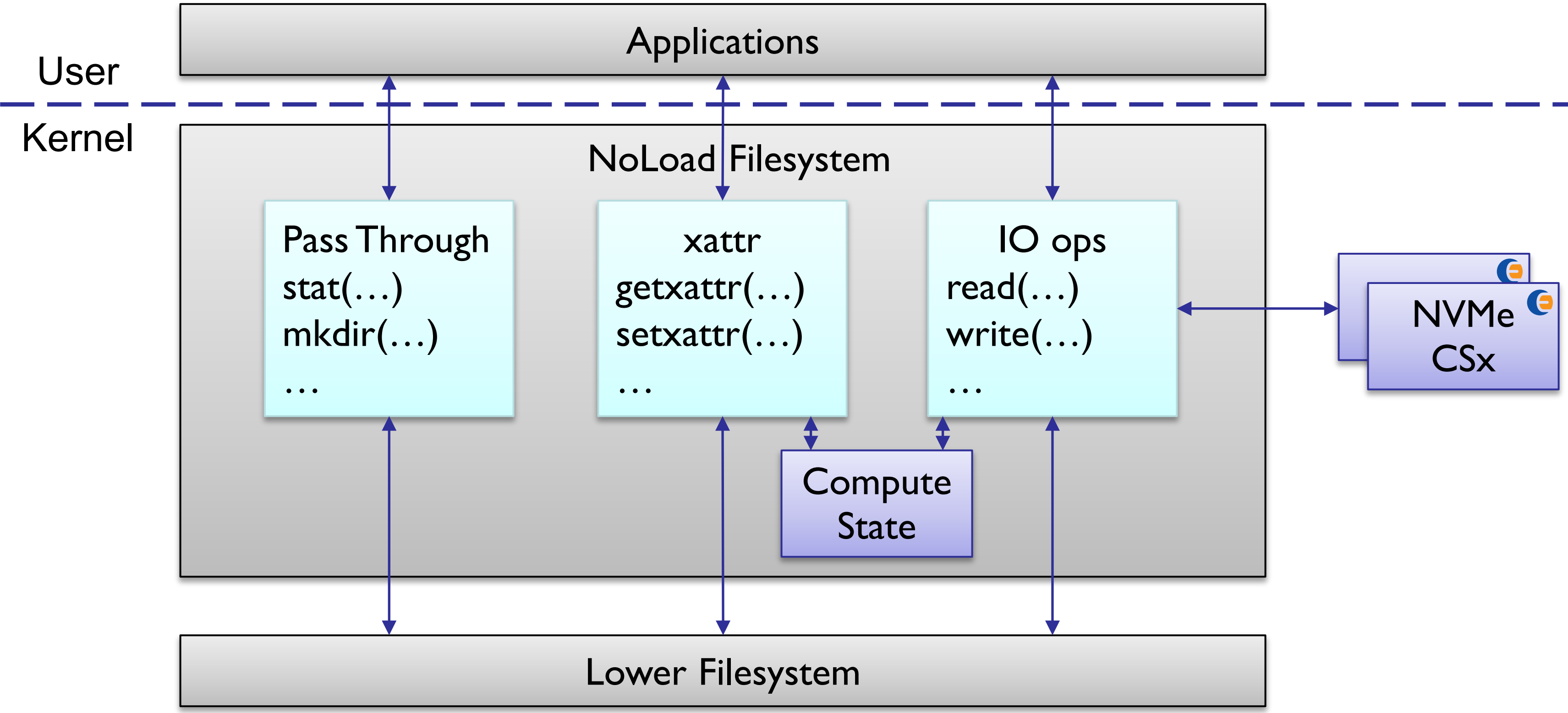
Stacked Filesystems



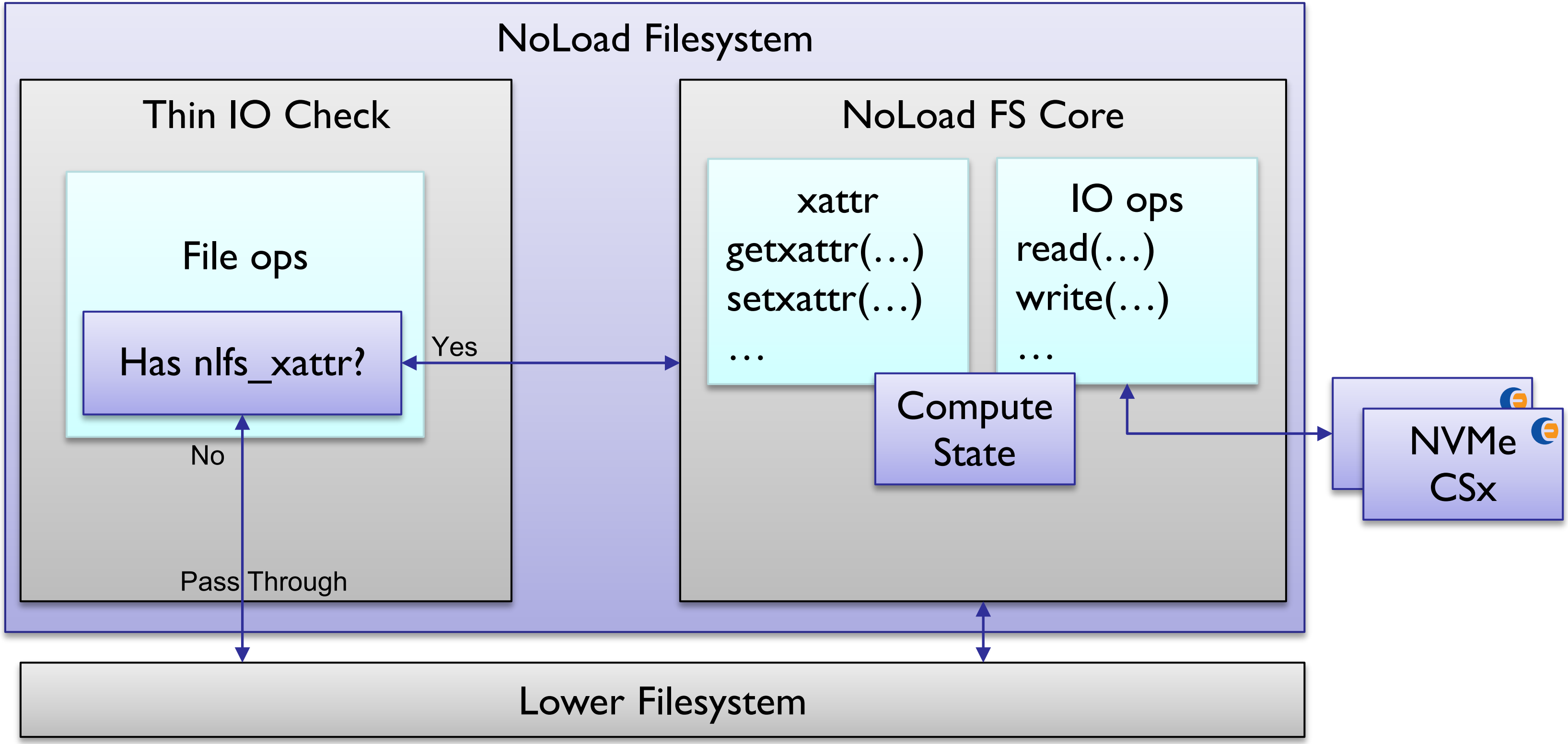
NoLoad Filesystem



NoLoad Filesystem



NoLoad Filesystem

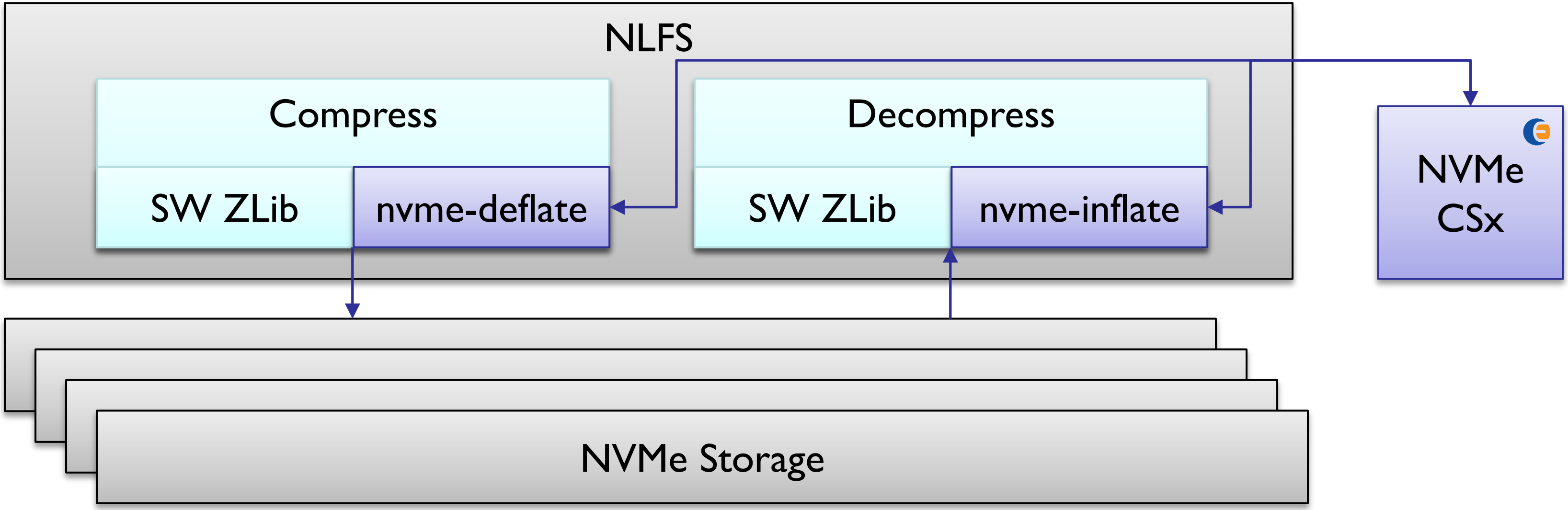


NoLoad FS: Compression Method

NLFS: Compressed Filesystem

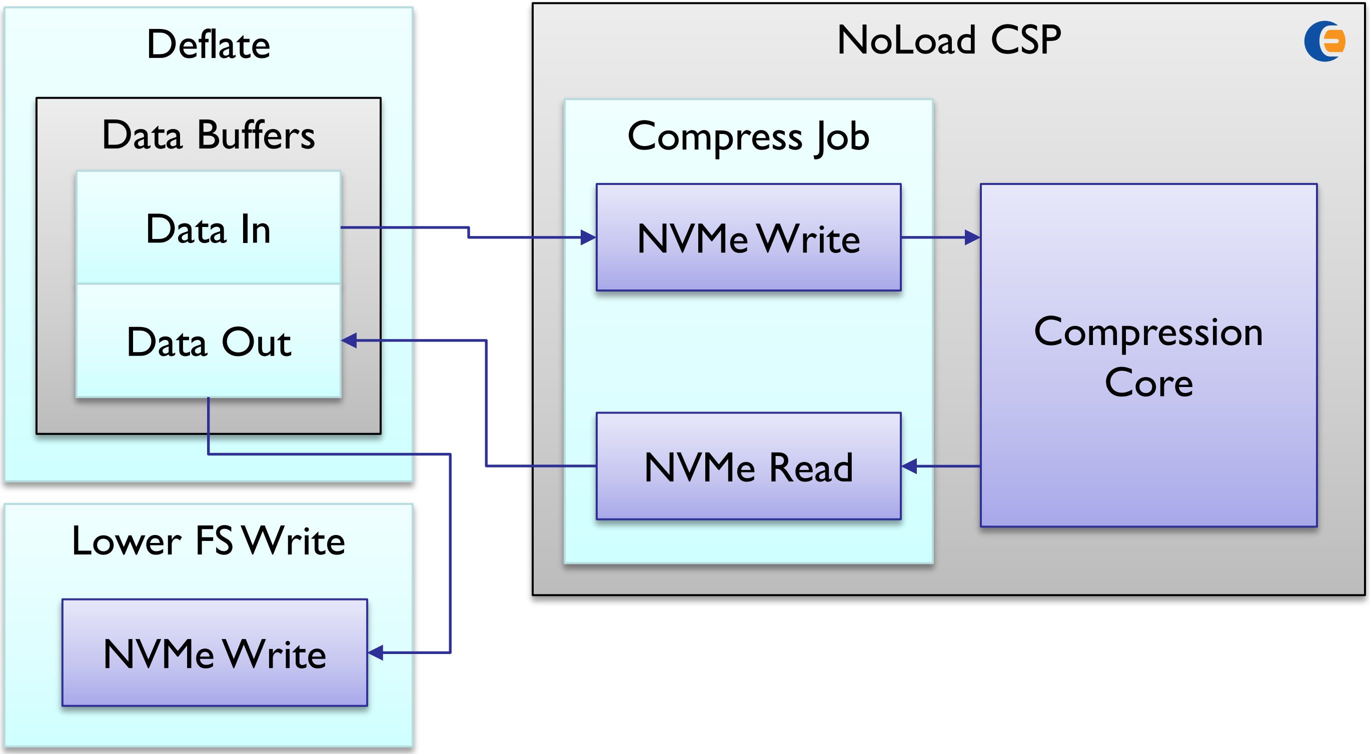
- Use existing infrastructure, Don’t reinvent the wheel
- SW Compatible
  - Z-Lib encoding
- Record-based

ZLib-Encoding



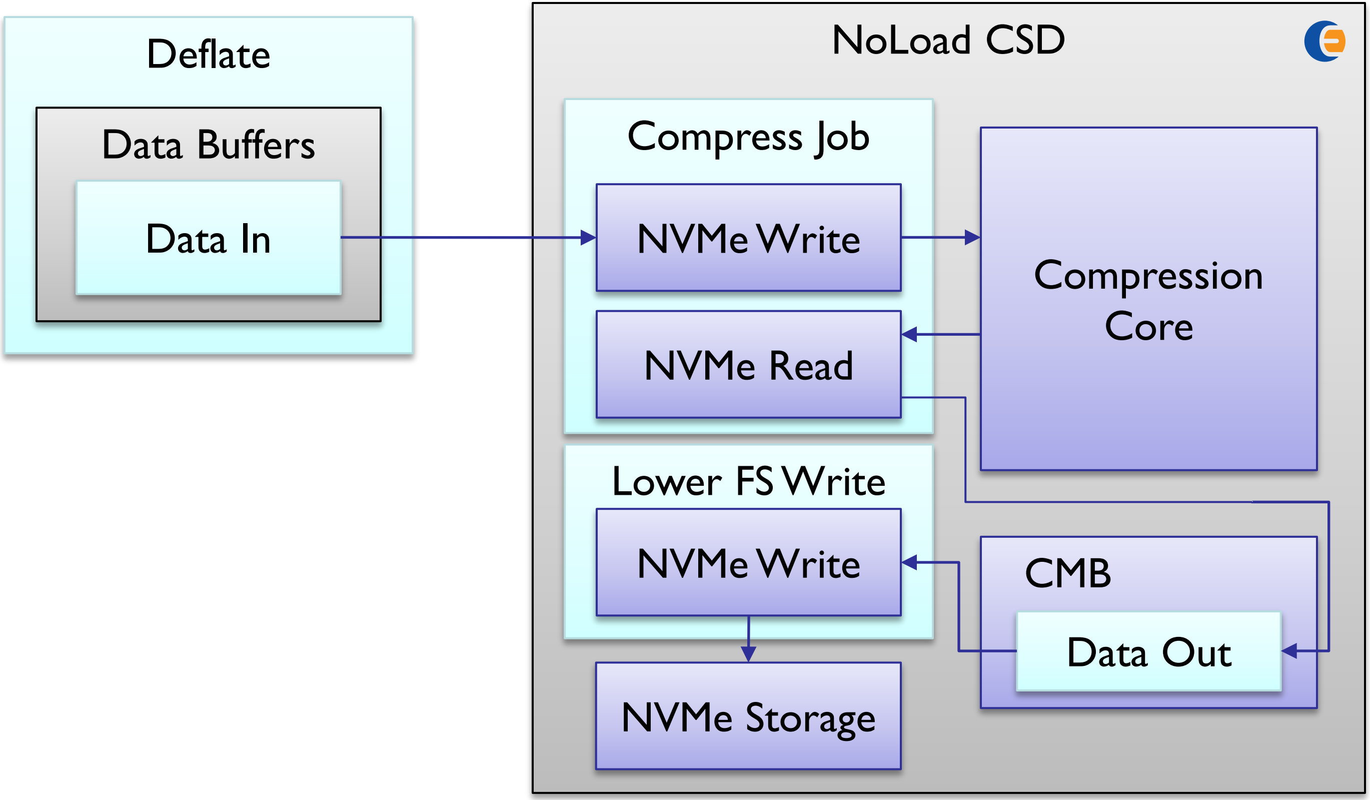
- SW Recovery
- Asymmetric topologies
- Use Existing Kernel Infrastructure

NoLoad CSP



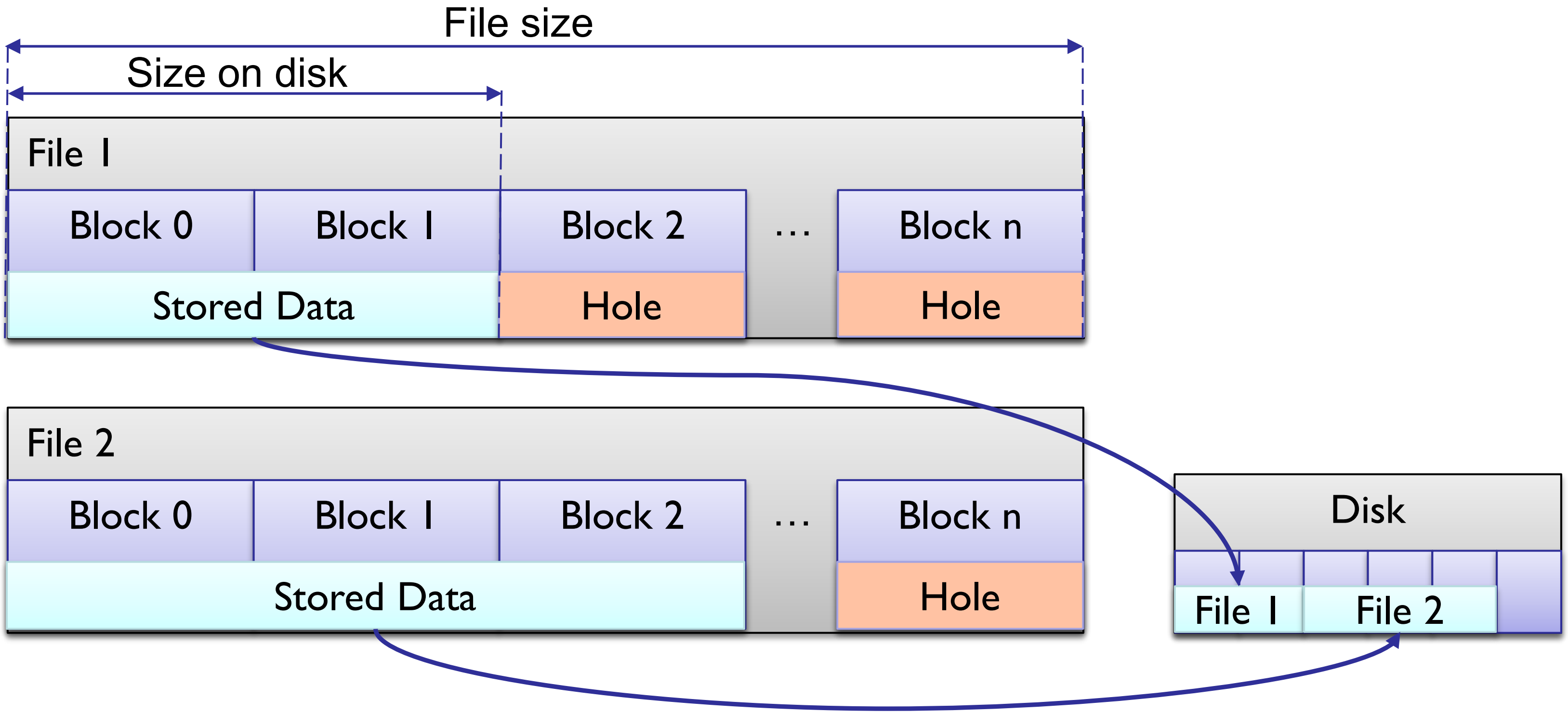
Deflate
Data Buffers
Data In
Data Out
NoLoad CSP
Compress Job
NVMe Write
NVMe Read
Compression Core
Lower FS Write
NVMe Write

NoLoad CSD


Deflate
Data Buffers
Data In
NoLoad CSD
Compress Job
NVMe Write
NVMe Read
Compression Core
Lower FS Write
NVMe Write
CMB
Data Out
NVMe Storage

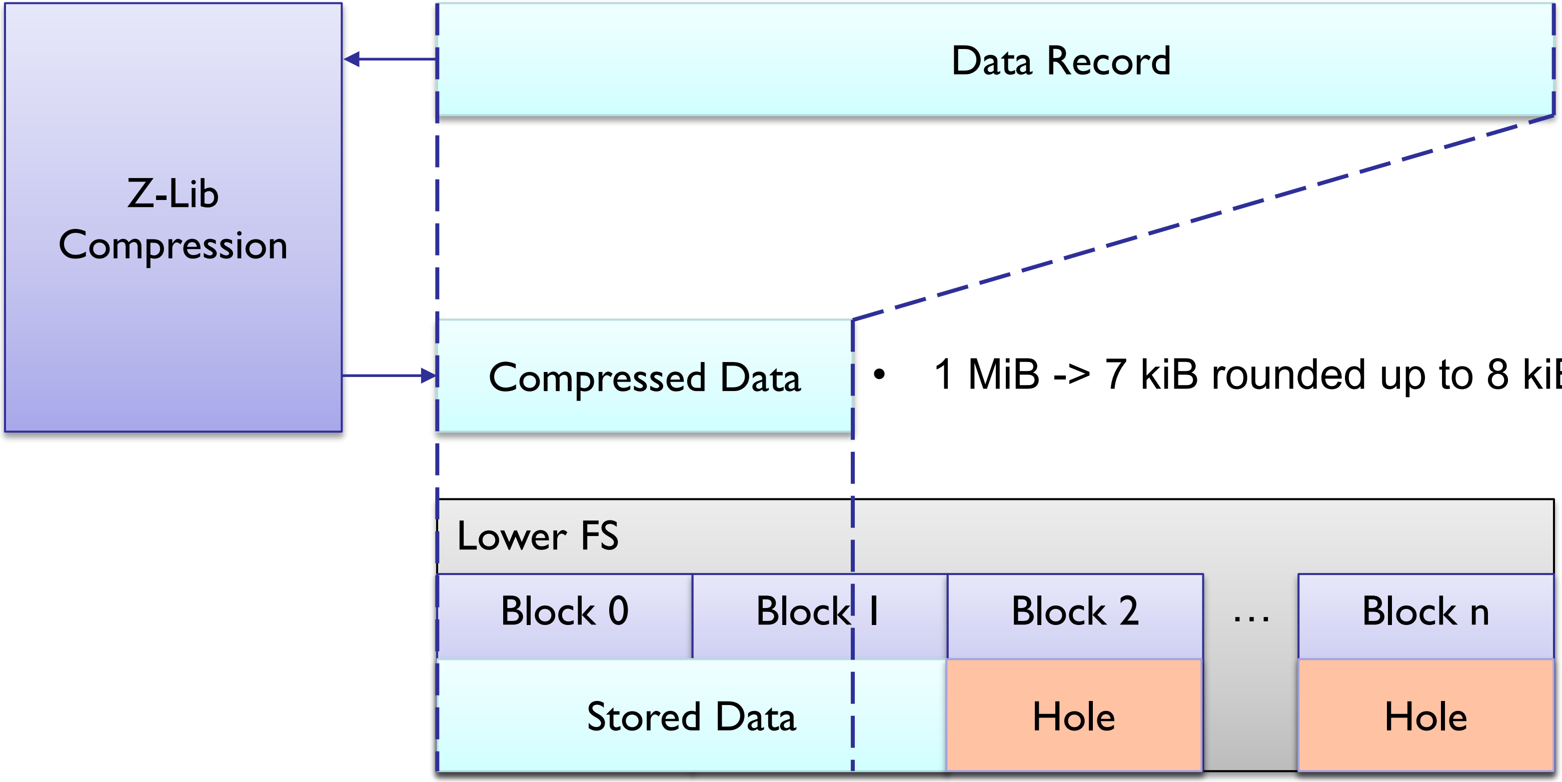
NoLoad FS: Storage Architecture

Storing Compressed Data



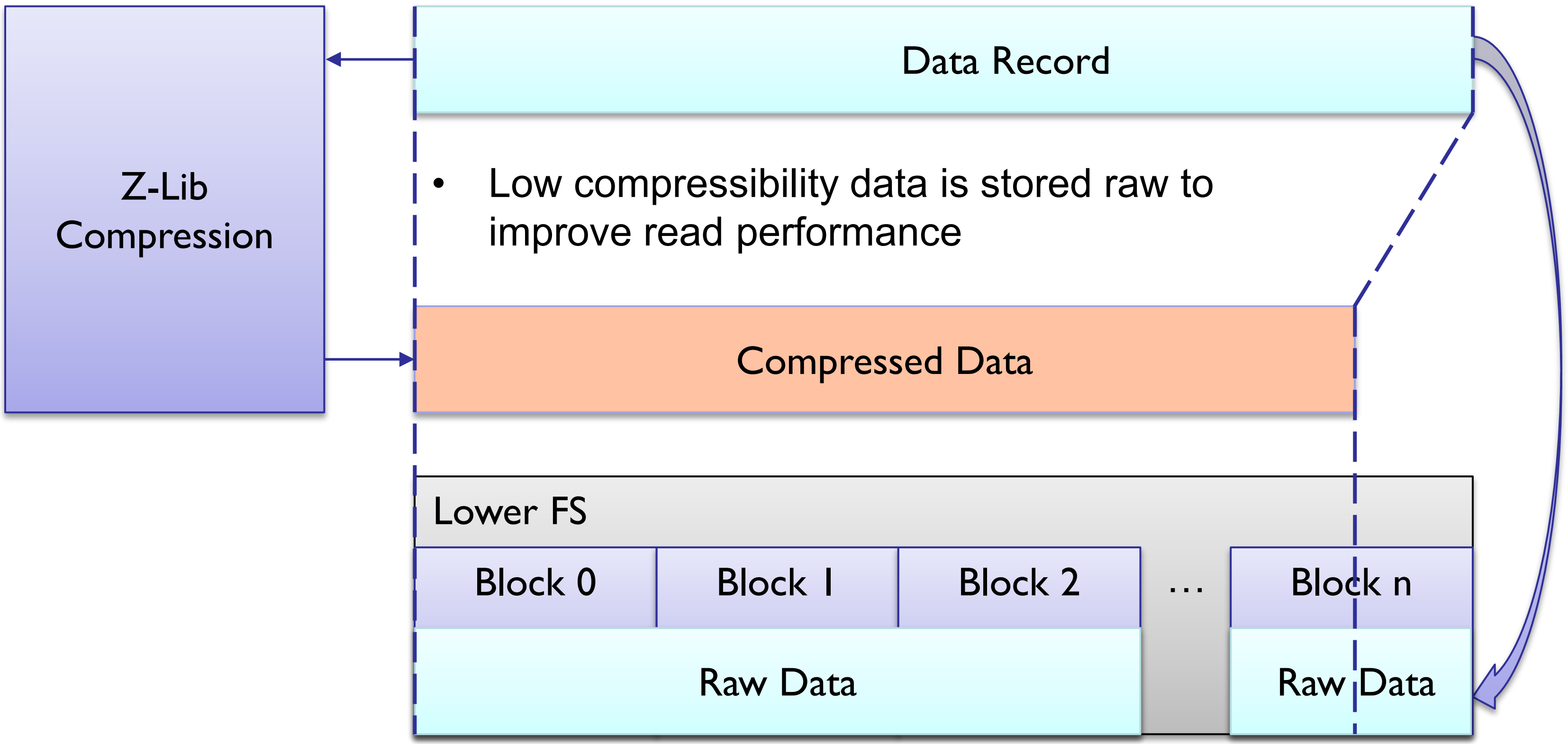
- Hole punching informs lower fs where data can be omitted

Compression Records



- 1 MiB -> 7 kiB rounded up to 8 kiB

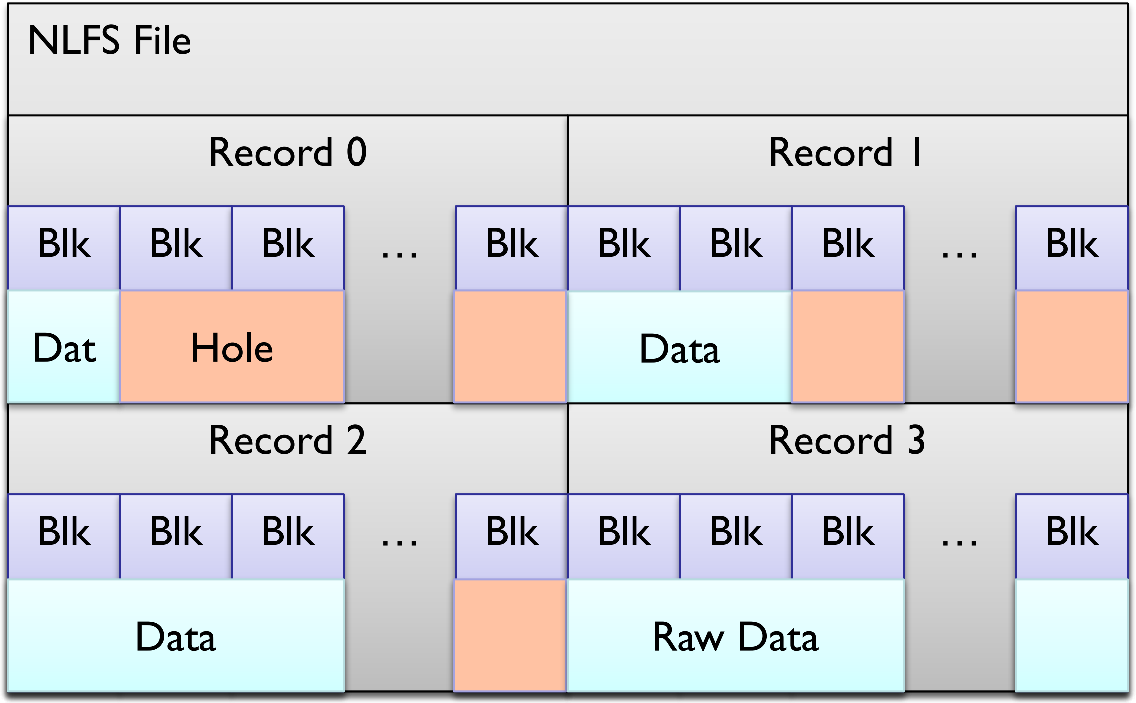
Compression Records



- Low compressibility data is stored raw to improve read performance

EIDETICOM

Compression Records



NoLoad FS: Interface

System Stats

```
$ls -lhs /nlfs/
total 128
128K -rw-r--r-- 1 dummy dummy 1.0M Aug 20 10:58 rand_ac.dat
```

Size on Disk

File Size

- Report compression info using built in tools
  - ls's -s option displays stored size, which can be less than the file size

Implementation Specific Stats

```
$xattr -l rand_ac.dat
user.nlfs_recordsize: 65536
user.nlfs_recordstat:
0000 … 8129.4340.65536.
…
00f0 … 8129.4340.65536.
```

Stored Size   Record Size

- Allows the reporting and modification of implementation specific settings on a per-file basis

EIDETICOM

NLFS Debug Info

NLFS Mount ID

```
$ls /sys/kernel/debug/nlfs/0
compress_records_actions_zlib     compress_ratio
compress_records_actions_nvme     compress_records_write
compress_records_actions_read     compress_records_read
decompress_records_actions_zlib   lower_path   raw_records_read
                                  mnt_path     raw_records_write
```

- compress_ratio
  - Compression ratio of mount as a whole (file size/size on disk)
- [lower|mnt]_path
  - Paths of lower fs and nlfs mount respectively

NLFS Debug Info

```
$ls /sys/kernel/debug/nlfs/0
compress_records_actions_nvme     compress_ratio
decompress_records_actions_nvme   compress_records_write
compress_records_actions_zlib     compress_records_read
decompress_records_actions_zlib   lower_path   raw_records_read
                                  mnt_path     raw_records_write
```

- [de]compress_records_actions_x
  - Number of actions taken for record type
    - Records processed using CSx
    - Records processed using SW

NLFS Debug Info

```
$ls /sys/kernel/debug/nlfs/0
compress_records_actions_zlib     compress_ratio
compress_records_actions_nvme     compress_records_write
compress_records_actions_read     compress_records_read
decompress_records_actions_zlib   lower_path   raw_records_read
                                  mnt_path     raw_records_write
```

- compress_records_x
  - Number of compressed records read/written to disk
- raw_records_x
  - Number of raw records read/written to disk

Usage Example

```
$ls /xfs/
raw.dat
$mount -t nlfs /xfs /nlfs -o recordsize=1M
$touch /nlfs/sequential.dat
$touch /nlfs/rand_ac.dat

$xattr -w user.nlfs_recordsize 65536 /nlfs/rand_ac.dat
$cp /nlfs/raw.dat /nlfs/sequential.dat
$cp /nlfs/raw.dat /nlfs/rand_ac.dat
$sync

$ls -lhs /nlfs/
total 1.2M
128K -rw-r--r-- 1 dummy dummy 1.0M Aug 20 11:23 rand_ac.dat
8.0K -rw-r--r-- 1 dummy dummy 1.0M Aug 20 11:23 sequential.dat
1.0M -rw-r--r-- 1 dummy dummy 1.0M Aug 20 11:22 raw.dat
```

Compressible file on lower fs

Optimize file for random access

Create new compressed files

Flush any cached data to disk

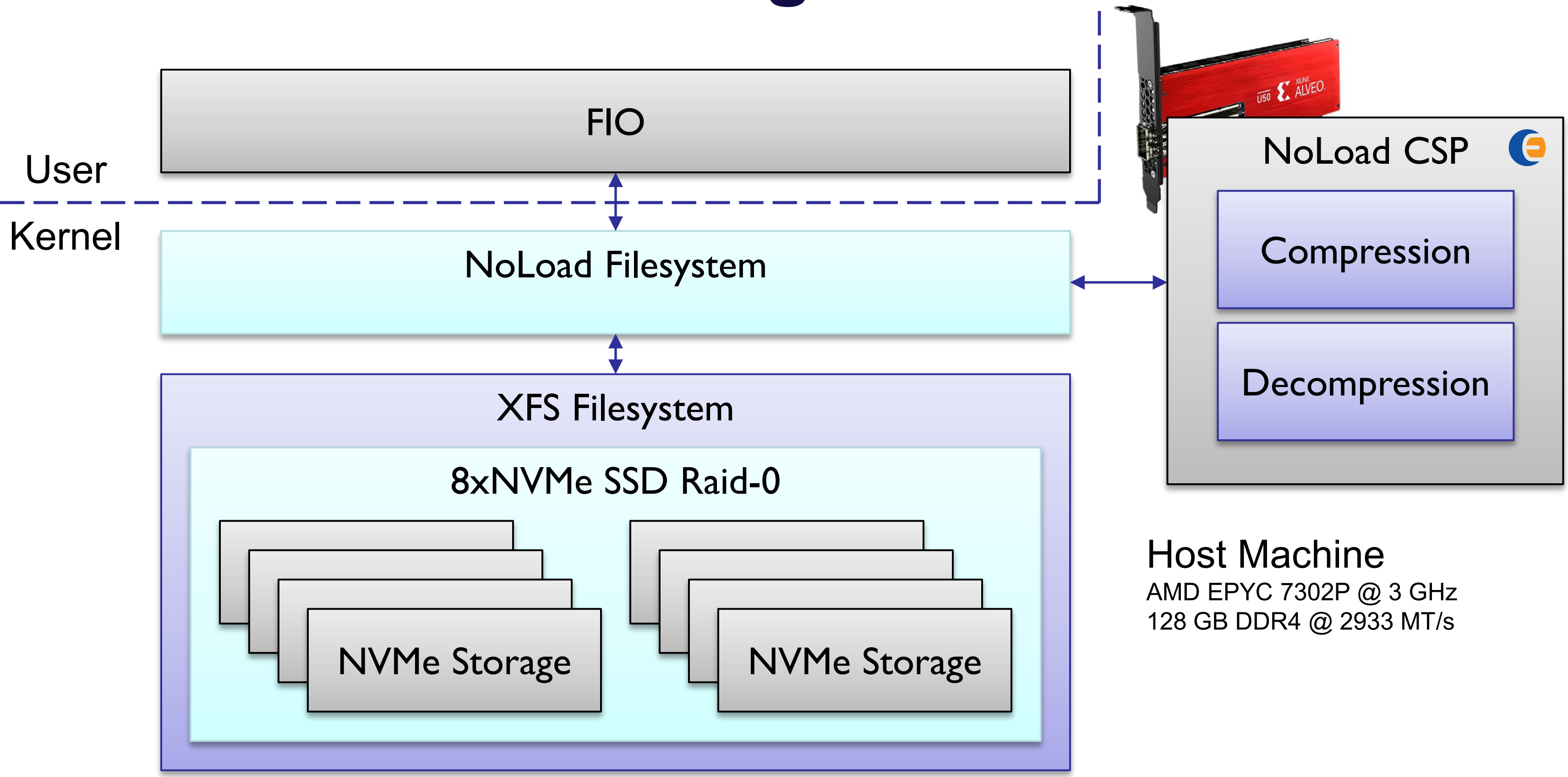
Small record file

Large record file

Pre-existing raw file

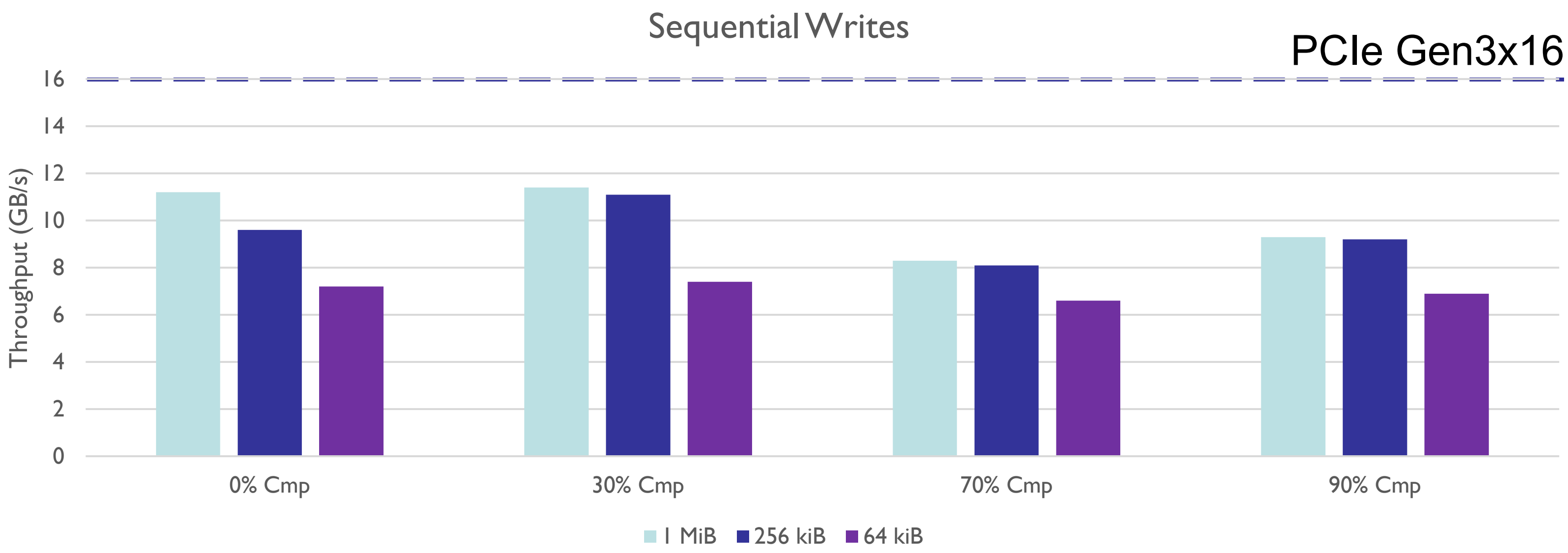
NoLoad FS: Performance

Configuration

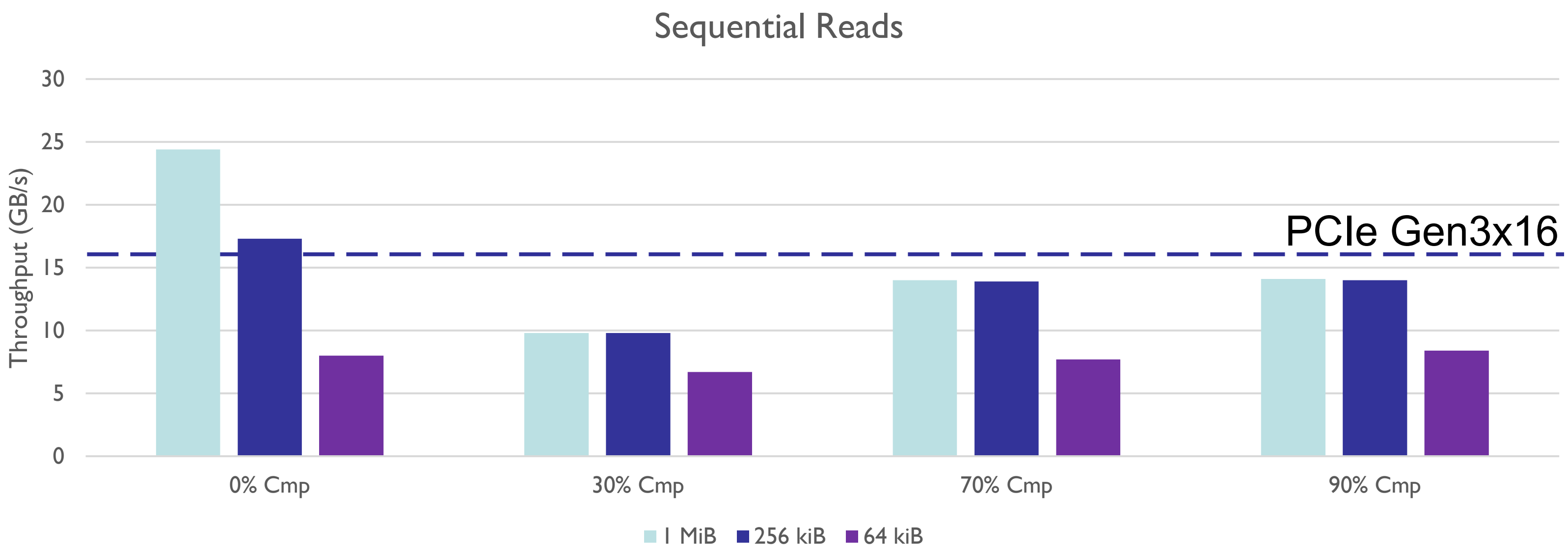


Host Machine
AMD EPYC 7302P @ 3 GHz
128 GB DDR4 @ 2933 MT/s

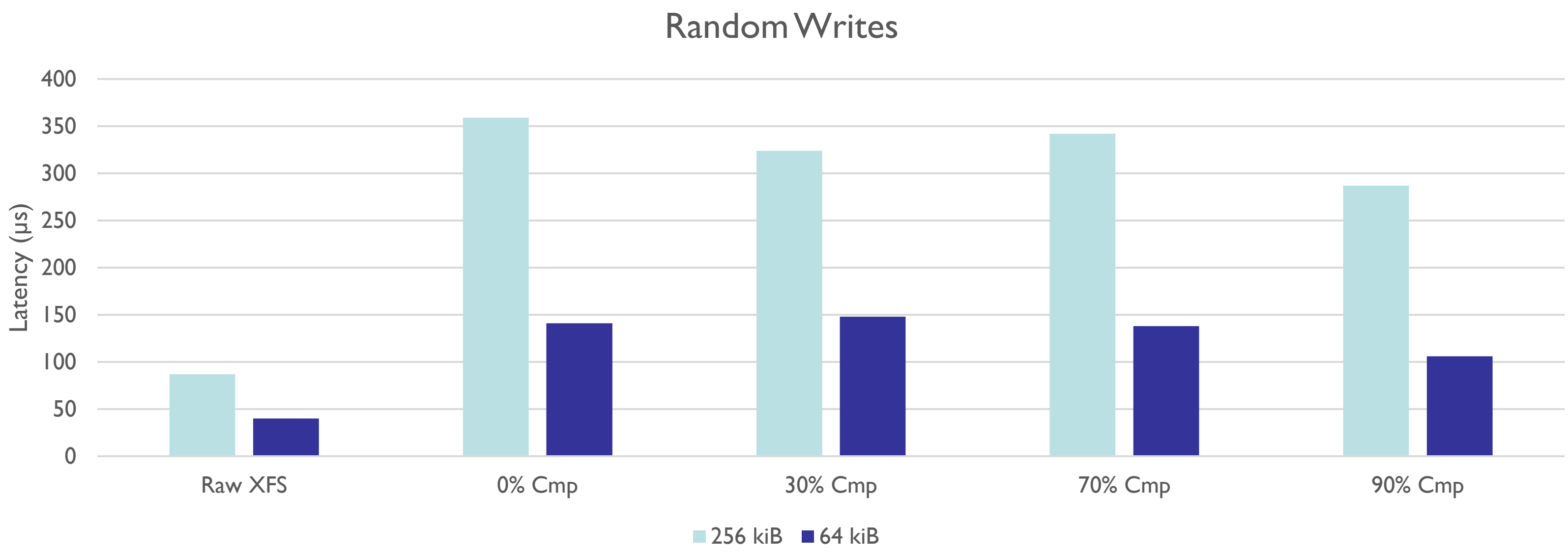
Write Throughput vs Record Size



Read Throughput vs Record Size



Write Latency vs Record Size



Read Latency vs Record Size

Conclusions

- Data volumes are exploding, and NAND is getting cheaper. Hardware accelerated compression can help bridge the gap
- NVMe CSx devices allow high speed injection of HW accelerated data transforms into applications
- Stacked filesystems allow for greater configuration flexibility and leverage existing filesystem optimizations
- A Stacked filesystem can be inserted into existing infrastructure with minimal effort

Please take a moment
to rate this session.

Your feedback matters to us.

EIDETICOM