Los Alamos NATIONAL LABORATORY

KV-CSD: A Hardware-Accelerated Key-Value Store for Data-Intensive Applications

Inhyuk Park∗, **Qing Zheng**†, Dominic Manno†, Soonyeal Yang∗, Jason Lee†, David Bonnie†, Bradley Settlemyer‡, Youngjae Kim§, Woosuk Chung∗, Gary Grider†

∗SK hynix, †Los Alamos National Laboratory, ‡NVIDIA, §Sogang University

11/1/2023

LA-UR-23-32086

Overview

Goal

- Rapid insight generation

Problem

- Scientific analysis often slowed down by unordered, unindexed data access

Approach

- Leverage computational storage to sort and index data at rest

KV-CSD

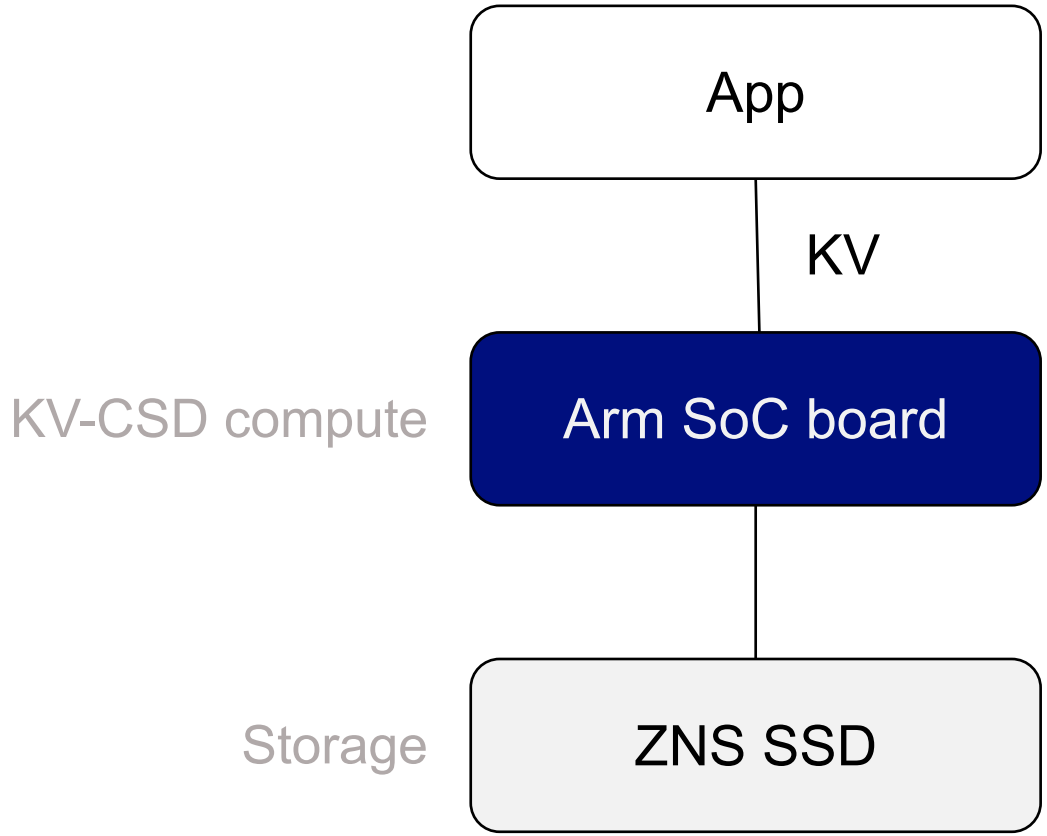
Hardware-accelerated KV storage for efficient data insertion and queries

A Quick Look

Two components: 1) An arm SoC board, and 2) A ZNS SSD

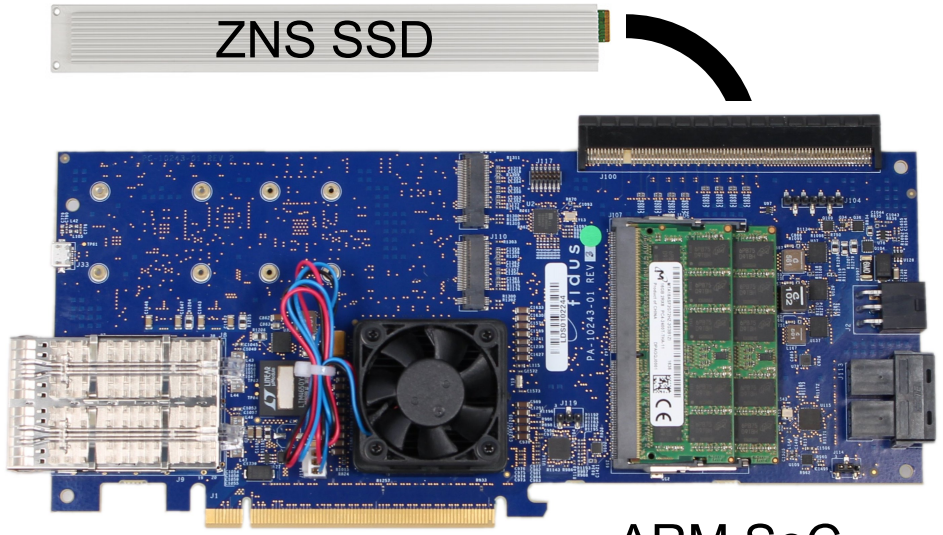
The arm board implements KV atop SSD zones

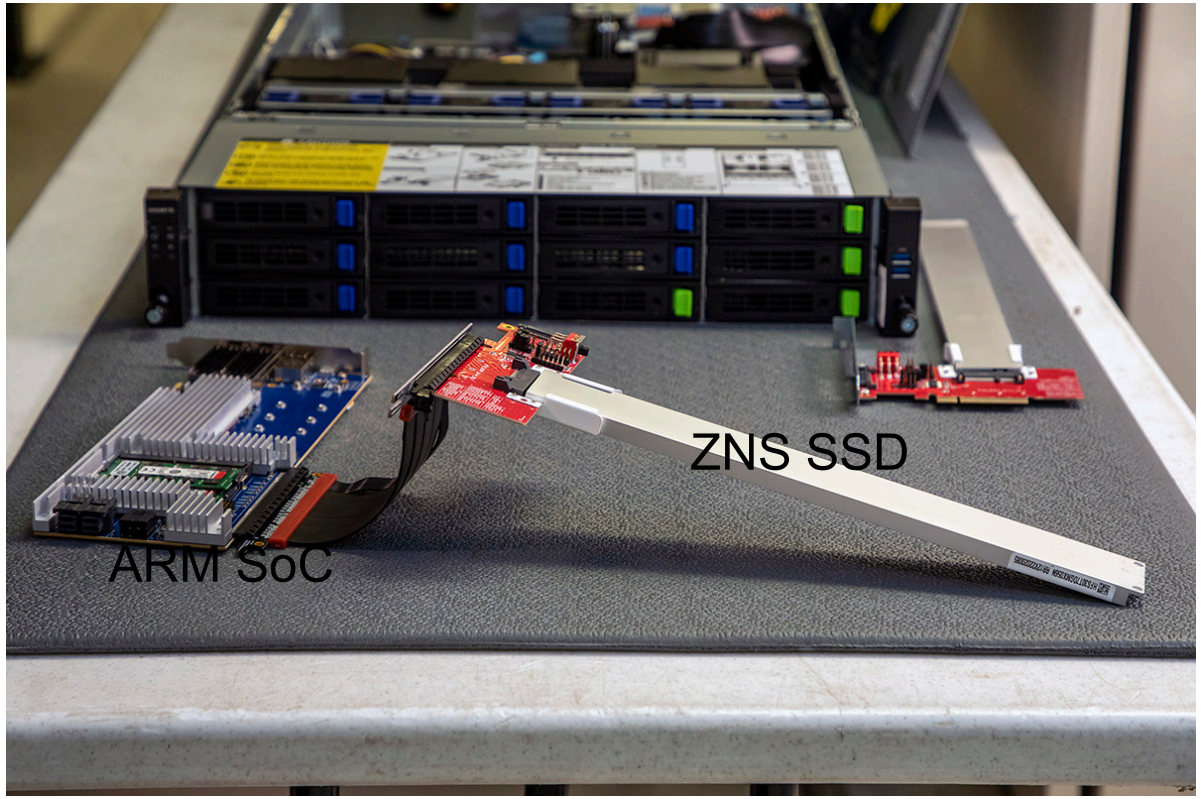
Apps use custom NVMe KV commands for bulk data insertion, index creation, and queries



KV-CSD in Real World

Current Prototype





Today's Talk

1. Why computational storage?

2. How does it work?

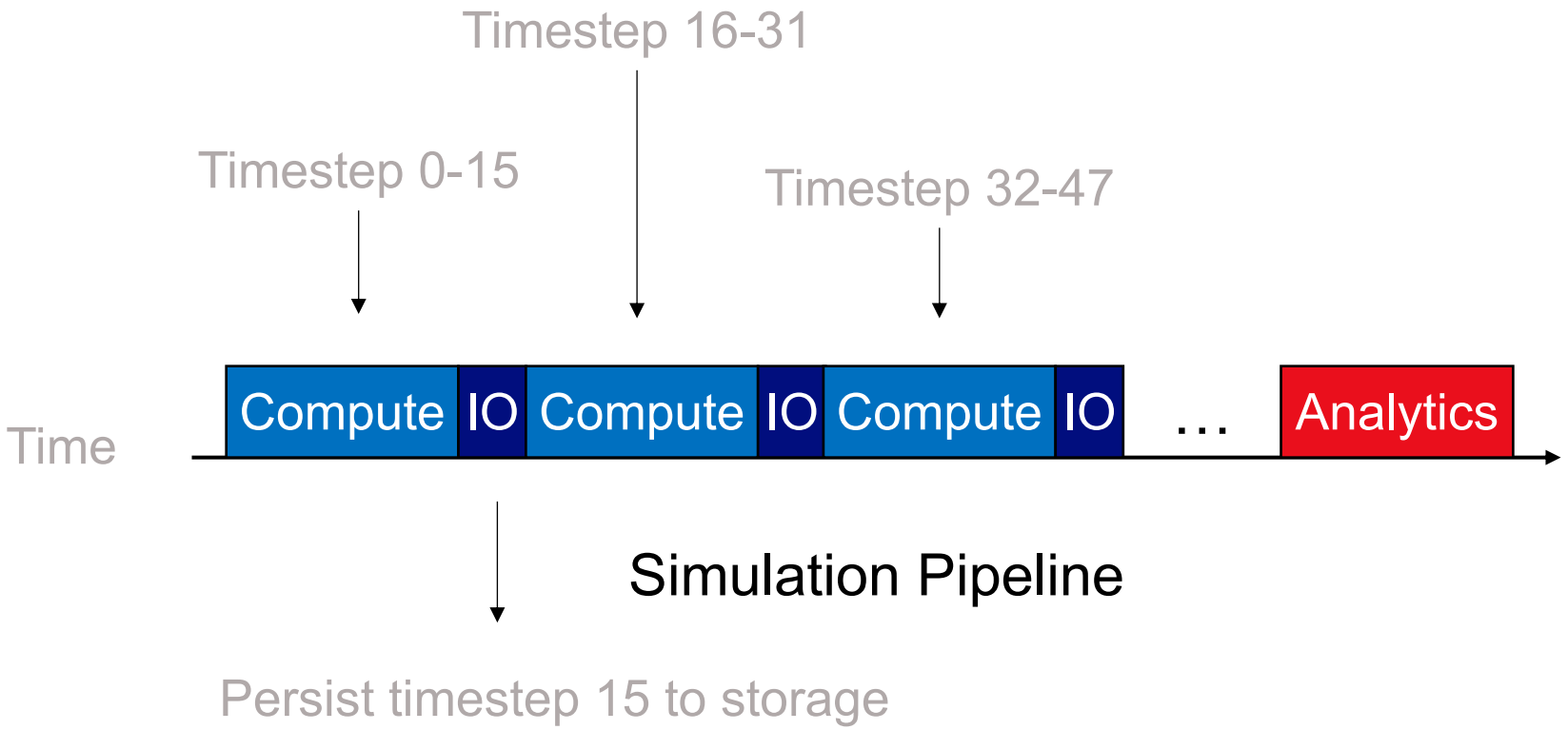
3. Results highlights

Recap: How Scientific Simulations Run

Time based bulk-synchronous parallel programs

Iterate between compute & I/O phases

Analytics occur after simulation

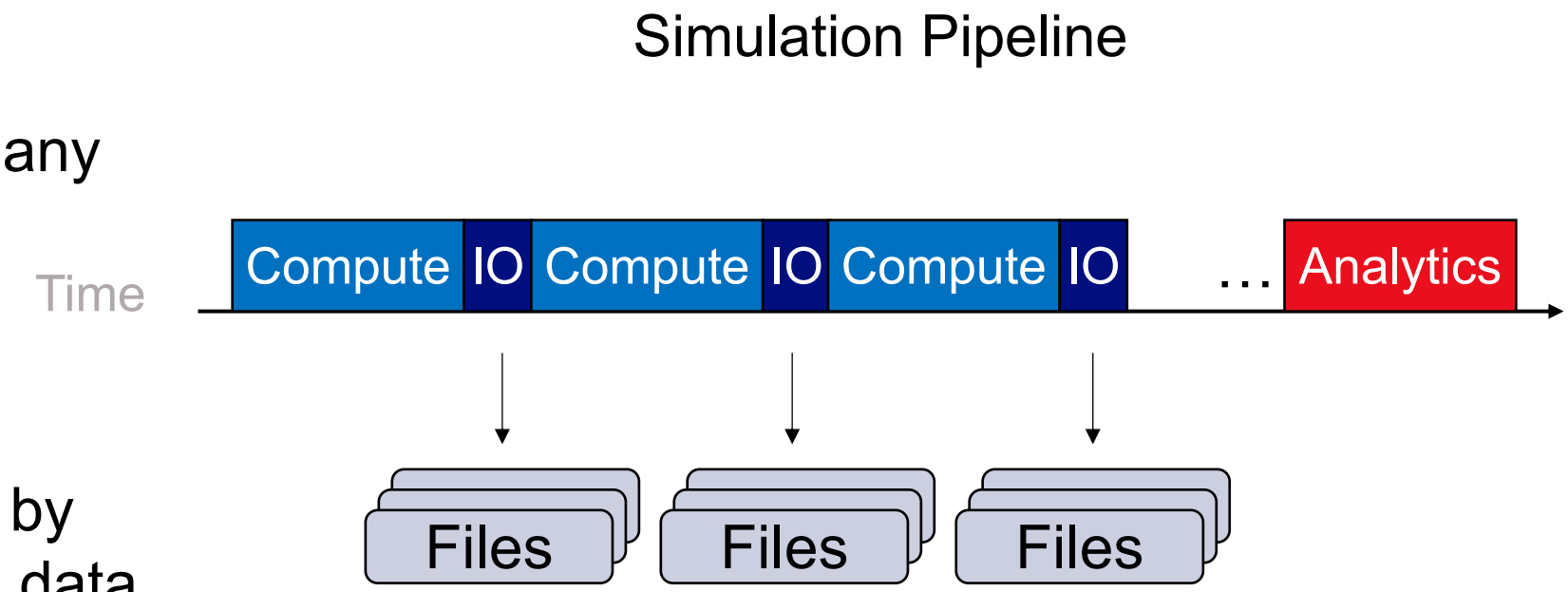


How Data is Stored Today

Through filesystems

Data stored as one big or many small files per timestep

Data typically accompanied by metadata that describes the data (type, dimension, …)



Why Analysis Can Be Slow?

Data may not be persisted in the same order as queries, leading to full data scans

Pre-sorting data prior to queries using many compute nodes can be equally inefficient

Computational storage offers new ways of acceleration

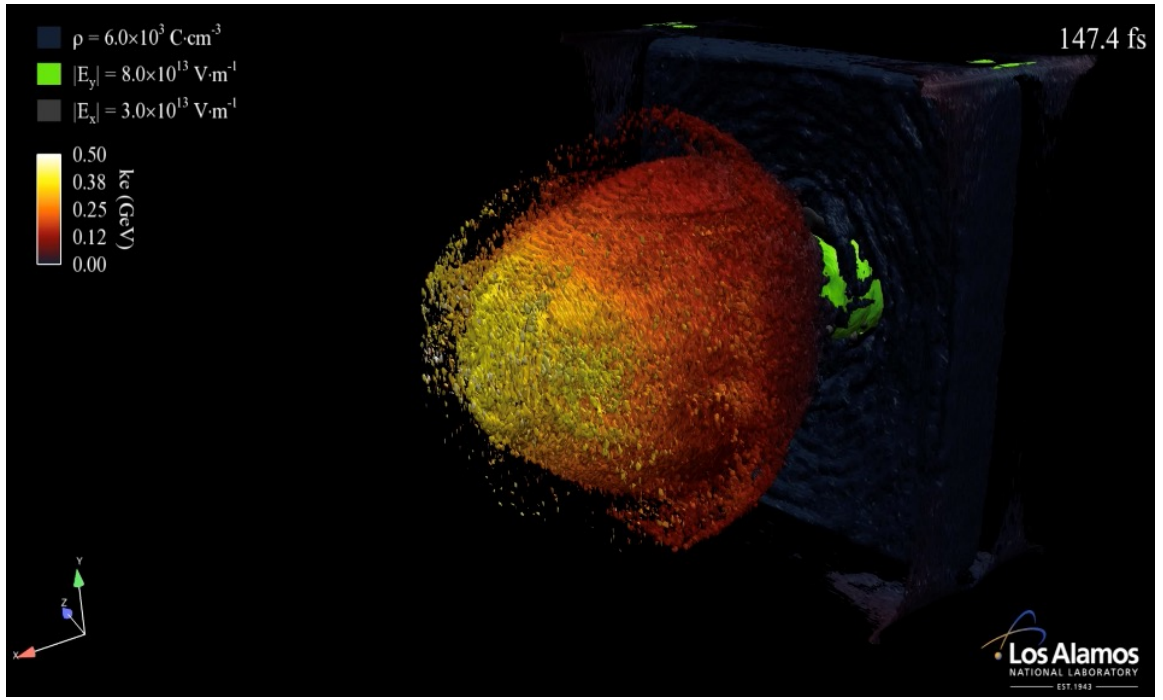


Image from LANL VPIC simulation done by L. Yin, et al at SC10

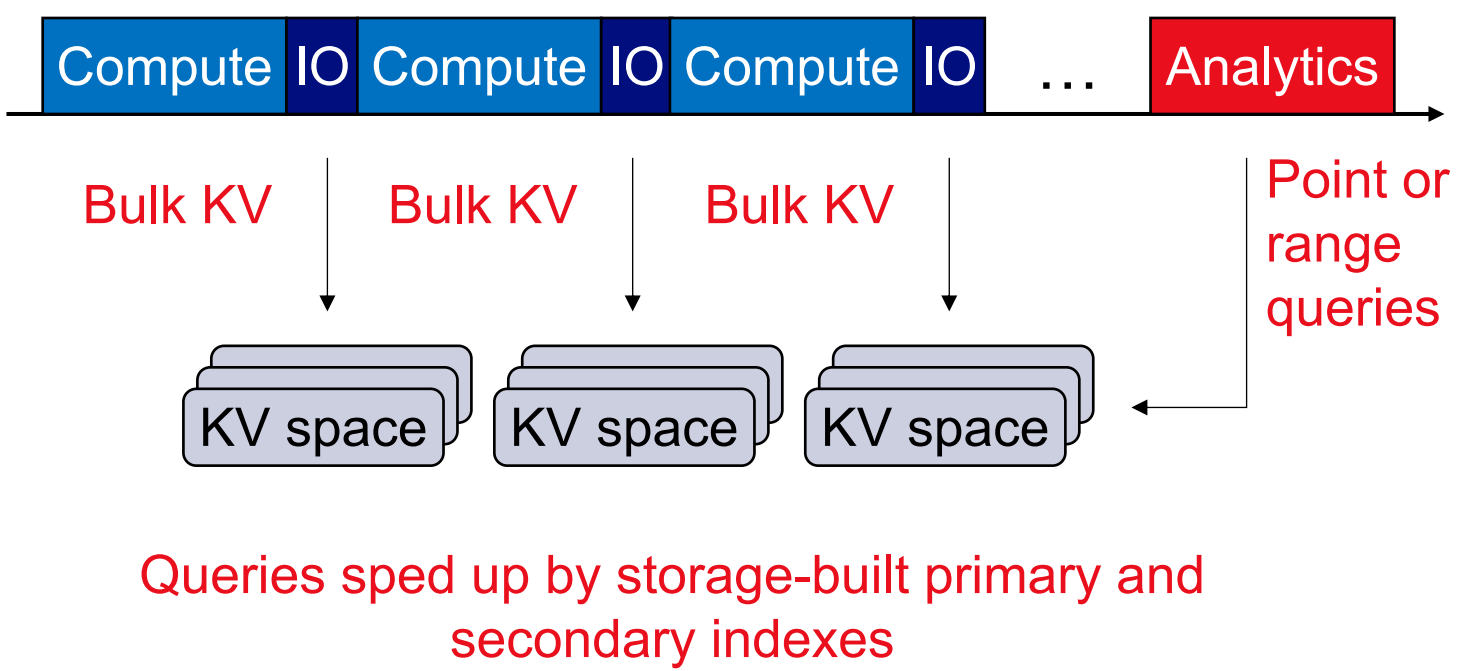
For example: a simulation may store its particles in particle ID order, but queries may target their energy levels

Toward Ordered, Computational KV Storage

App converts data to KV pairs and bulk inserts them into storage

One KV space per app process per timestep

Storage sorts data by key asynchronously and builds secondary indexes per app query needs



Why KV?

Scientific data often resembles records with keys and values

KV provides GET / SCAN primitives unavailable from filesystems

KV interface already very popular

KV provides sufficient knowledge of data without having to resort to external metadata (e.g.: no need for filename to storage LBA translation)

Why Hardware Acceleration?

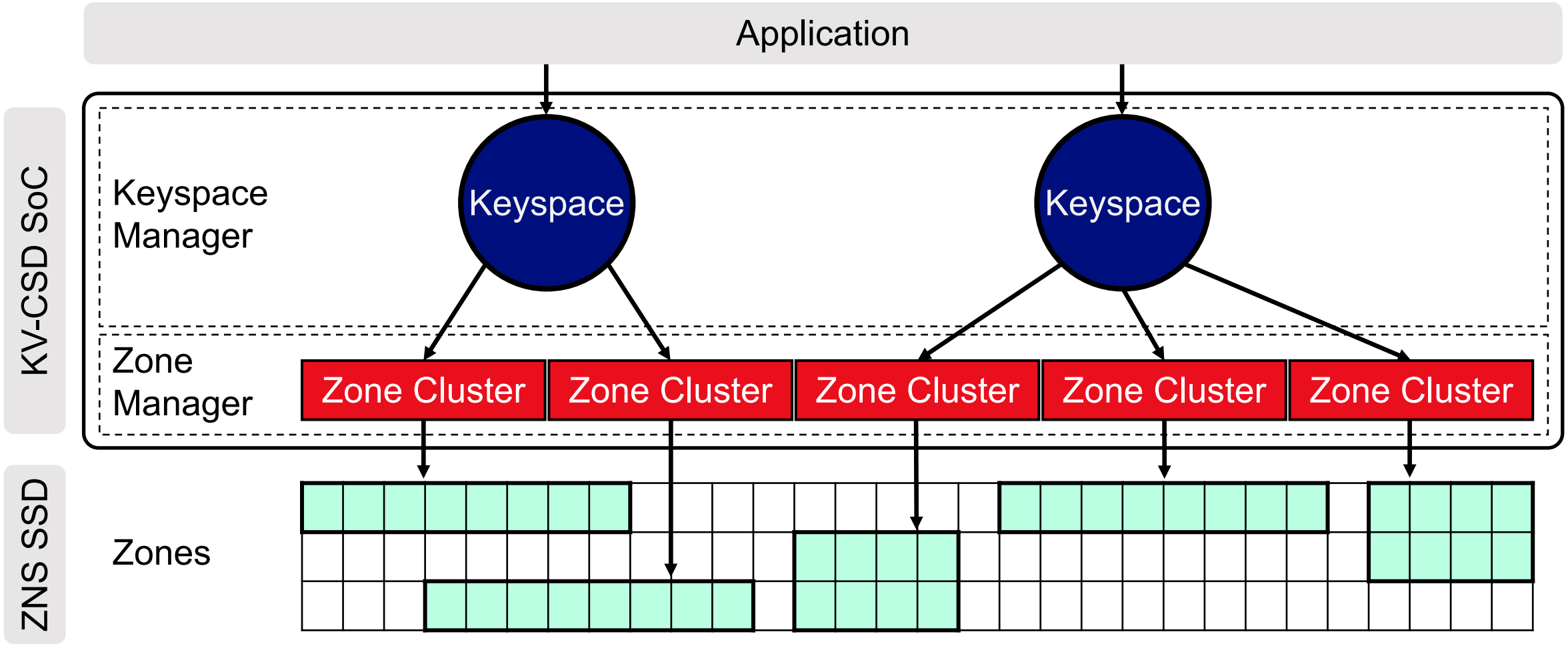
Software KV stores (such as RocksDB) rely on background processing to hide data sorting latency

Insertion is suspended when background jobs cannot keep up

Hardware acceleration allows for more aggressive latency hiding

By deferring background work until after insertion concludes and by performing it within a computational storage device

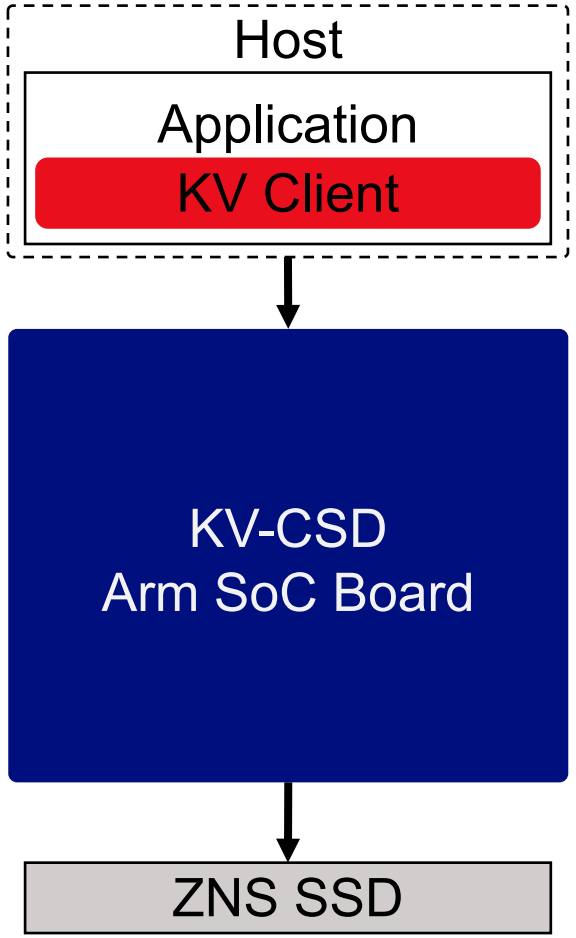
A Closer Look at the Device



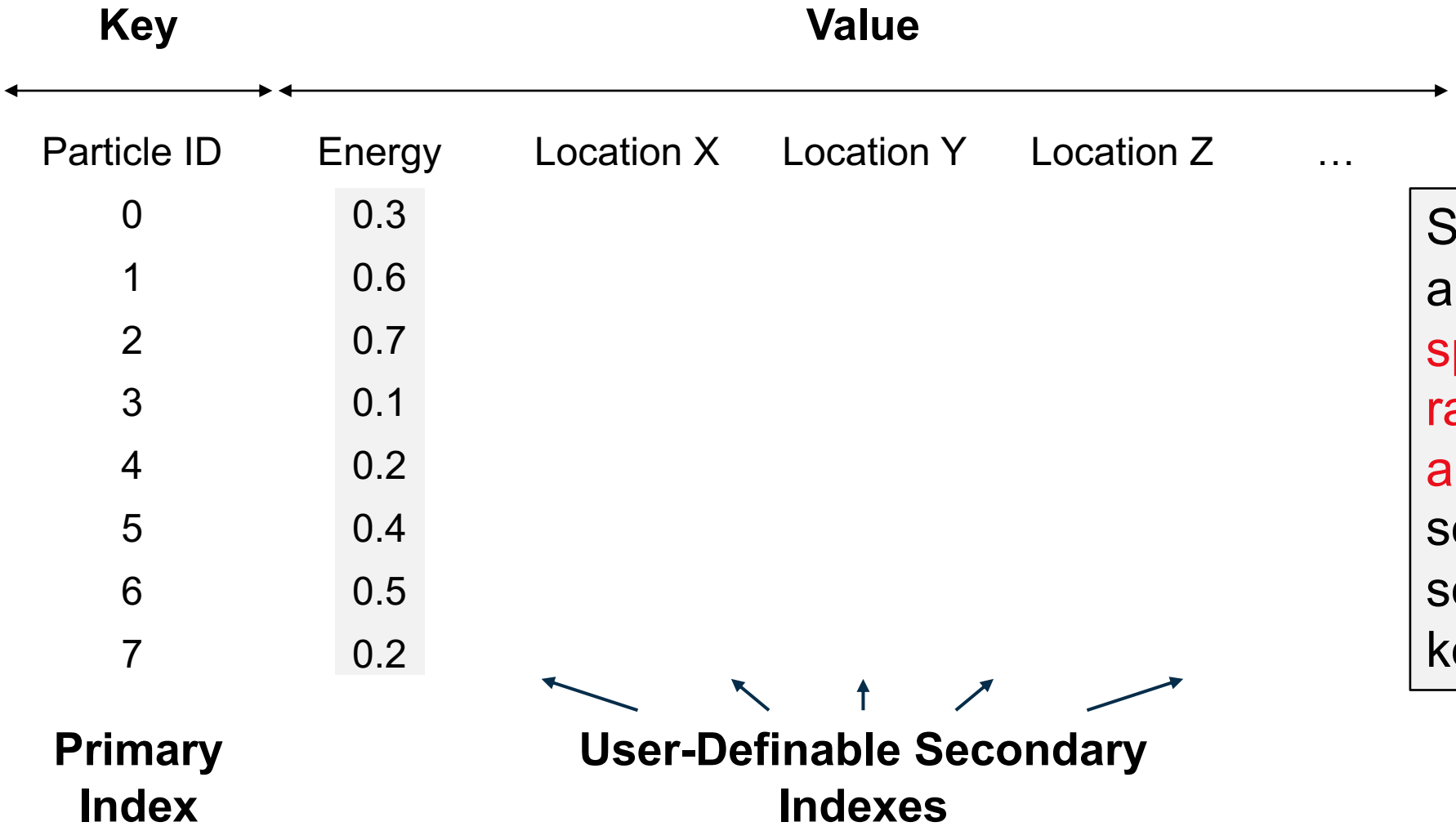
Each cell represents a zone

Keyspace API

	Keyspace State			
	New	Writable	Indexing	Indexed
Keyspace Info	✓	✓	✓	✓
KV insertion		✓		
Query				✓
Keyspace Deletion	✓	✓	✓	✓



Primary and Secondary Indexes



Secondary indexes are defined by users specifying the byte range and the type of a portion of value to serve as the secondary index keys

Result Highlights: More Details in Paper

	Filesystem (Base approach)	RocksDB (State-of-the-art)	KV-CSD (This paper)
Simulation I/O Path	Fast	Slow	Fast
Analytics Path	Slow	Fast	Fast

Both KV-CSD and RocksDB allow efficient reads, but KV-CSD does so without potentially significantly slowing down writes

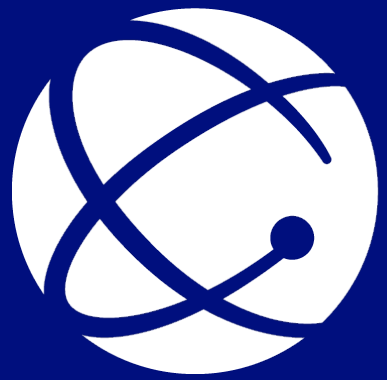
Conclusion

Efficient data retrieval matters

Hardware KV stores enable analytics-friendly primitives while better hiding background work latency (than software solutions), leading to better time-to-insights

KV-CSD is tailored for scientific simulation pipelines, at the cost of being more restrictive than regular KV stores (see more discussion in paper)

Computational storage more practical now than it was 30 years ago (though more R&D is needed for production deployment)



Los Alamos

NATIONAL LABORATORY