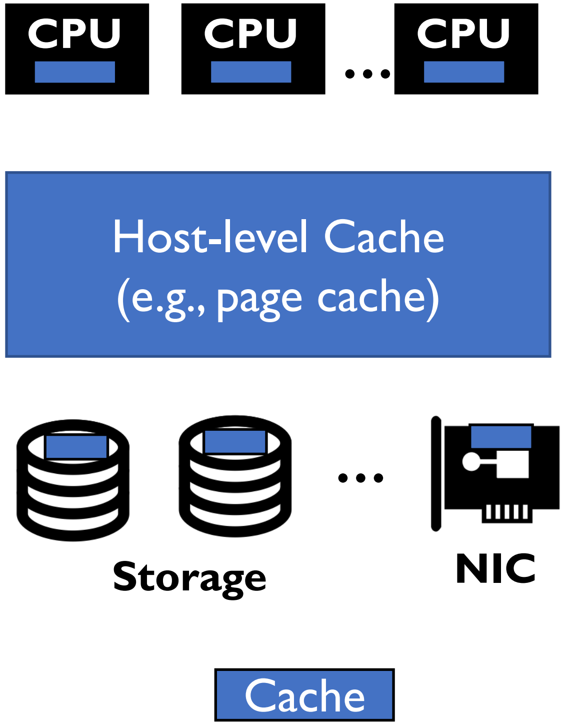
OmniCache: Collaborative Caching for Near-storage Accelerators

Jian Zhang, Yujie Ren, Marie Nguyen, Changwoo Min, Sudarsun Kannan

Caches are Ubiquitous!

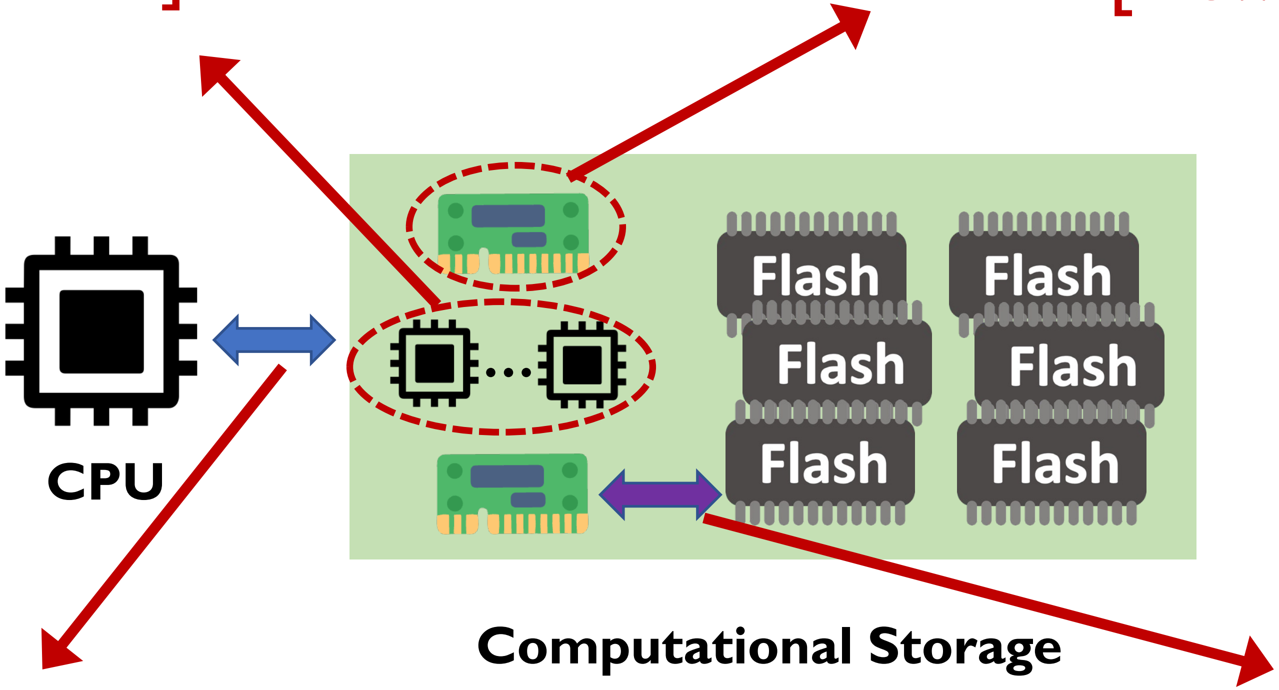
- Caches are present across different layers of the computer system
- Exploiting and combing all caches cumulatively can accelerate I/O and data processing performance
- Proposed: Combining host-level I/O cache with near-storage cache
- Besides, technologies like CXL can easily interconnect all caches
- How to build an efficient caching design for near-storage accelerators?



Outline

- Background
- Motivation
- Design
- Evaluation
- Conclusion

Hardware Trends



Multi-core compute (4-16 cores)
[ARM CSDs]

DRAM size (4-16GB)
[Newport]

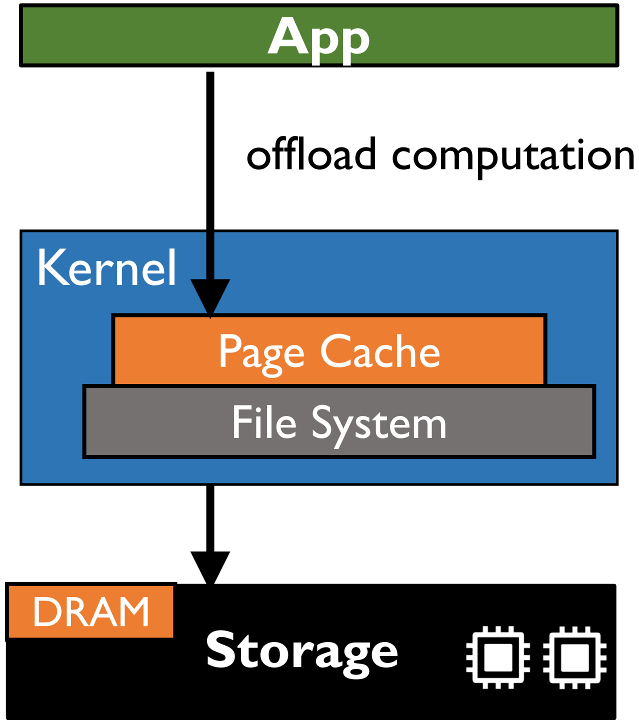
Fast remote memory access with CXL
[Samsung Memory-Semantic SSD]

High speed interconnect
[ScaleFlux]

Near-storage Devices Designs

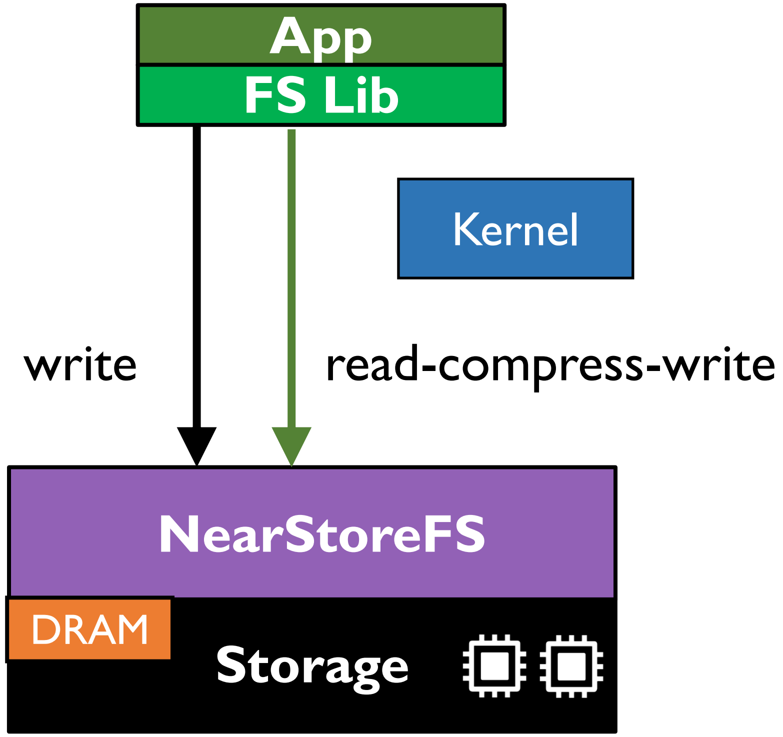
Near-storage Device with Host FS

(PolarDB [FAST '20], λ-IO [FAST '23], etc.)



Device FS

(DevFS [FAST '18], FusionFS [FAST '22])

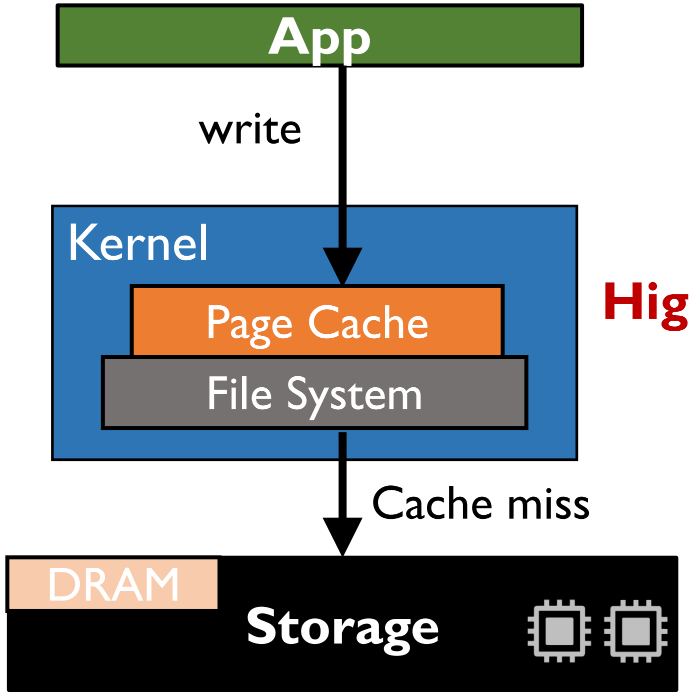


Outline

- Background
- Motivation
- Design
- Evaluation
- Conclusion

Failure to Exploit Near-storage Memory

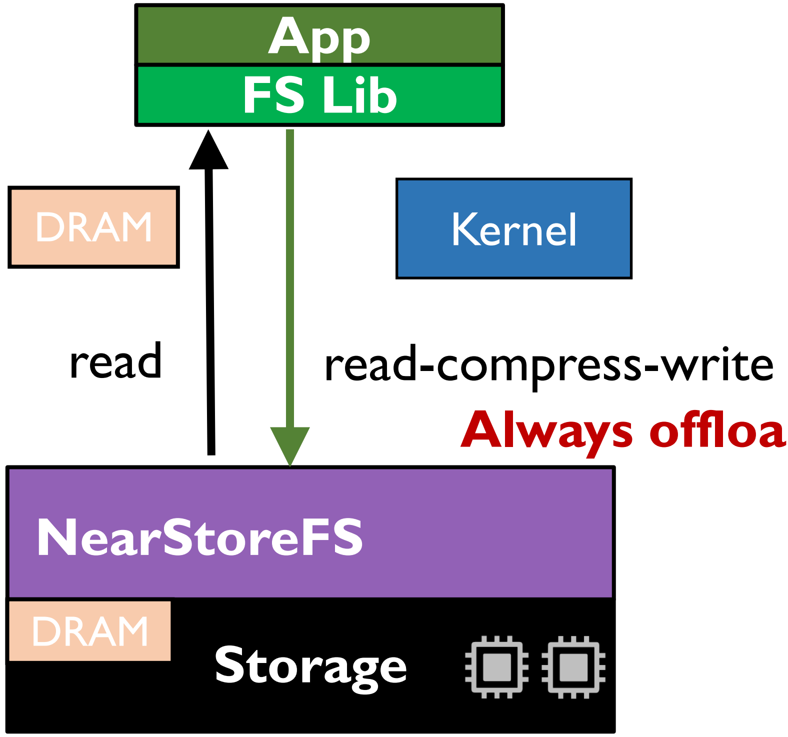
Near-storage Device with Host FS



High kernel overhead

Lack of support to use device memory

Device FS



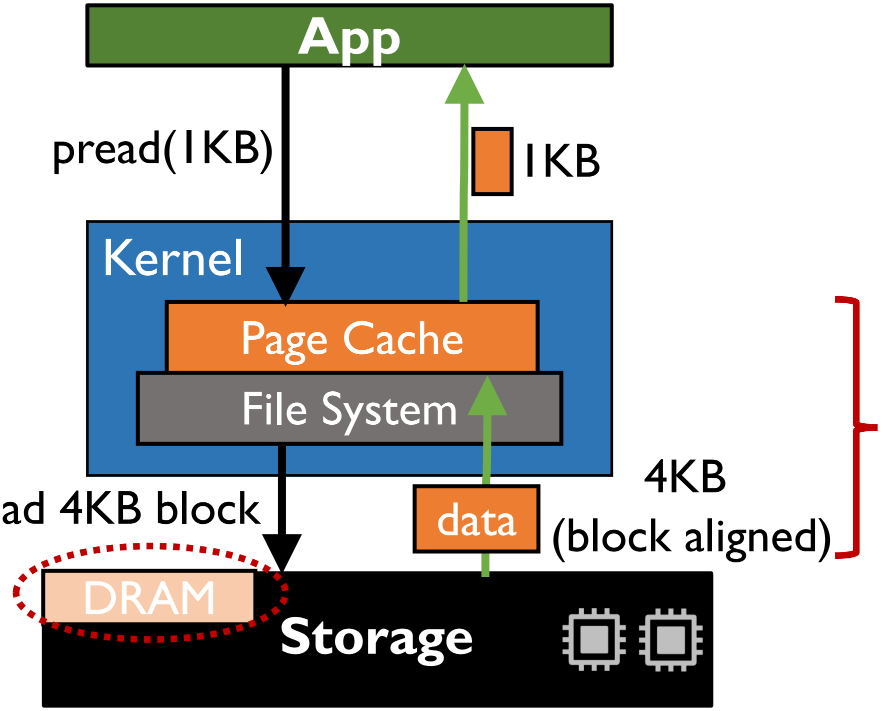
Always offload!

Failure to utilize host and device memory

High Data Amplification in Host Cache Designs

- Substantial unaligned I/O request ratio in popular real-world applications
  - (> 95% requests of the 100 million total I/O requests were unaligned in both DiskANN and RocksDB)
- Unaligned I/O requests cause data amplification in host cache designs

Near-storage Device with Host FS



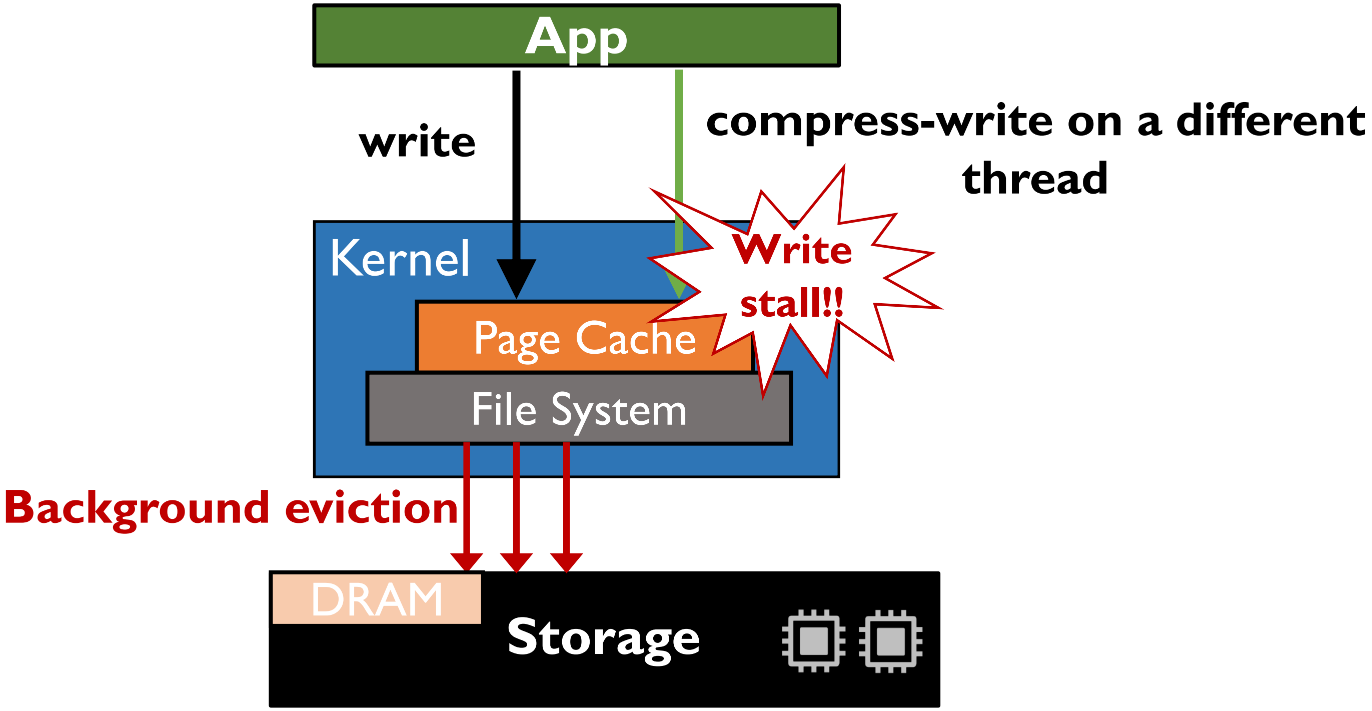
data amplification!

Device caches can reduce data amplification

Lack of Concurrent I/O and Data Processing Support

- Application threads stall frequently when host-level cache are full

Near-storage Device with Host FS

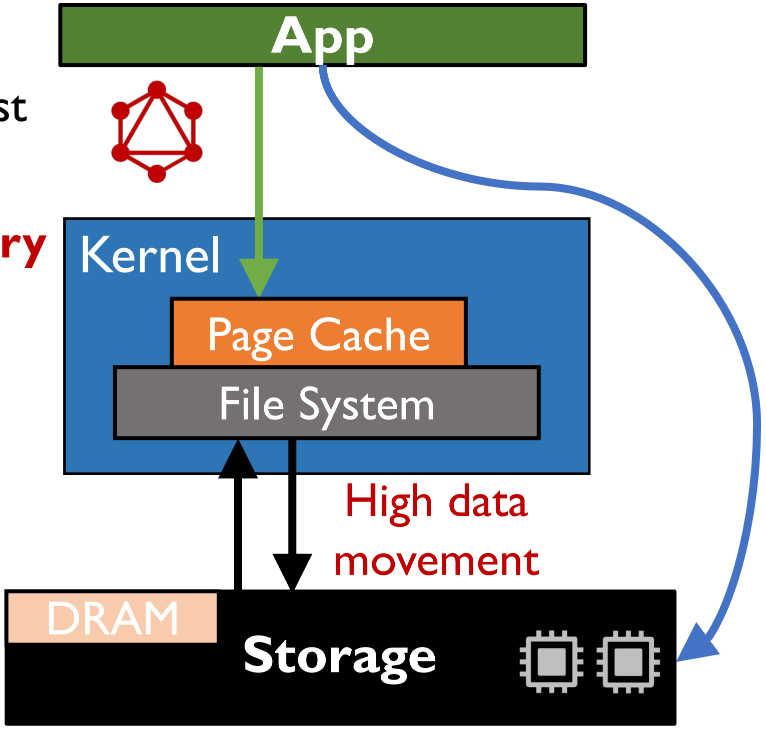


Failure to exploit device compute and memory impacts concurrent I/O and processing!

Lack of Dynamic and Concurrent Processing Support

- Current approaches lack capability to dynamically decide where to process (in the host or the device)

Near-storage Device with Host FS



Execute KNN at host
(large graph)

Not using device memory and compute

High data movement

Or offload KNN to device
(large graph)

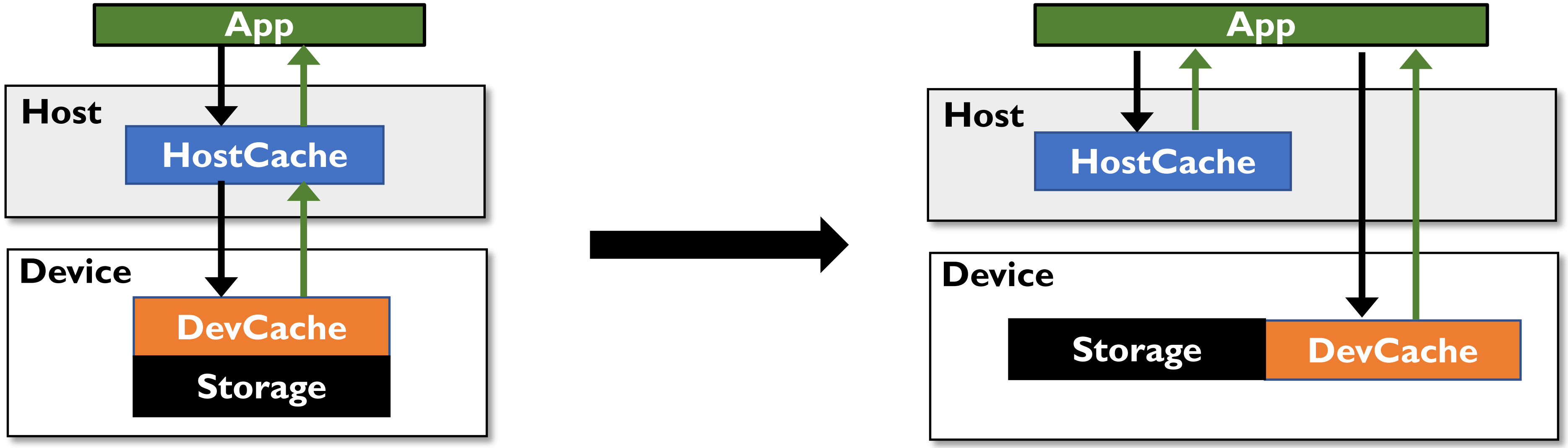
Limited Device Resources

Outline

- Background
- Motivation
- Design
- Evaluation
- Conclusion

Our Solution: OmniCache

A **horizontal caching** design to exploit the **combined capabilities** of near-storage, host compute, and their memory resources to accelerate I/O and data processing

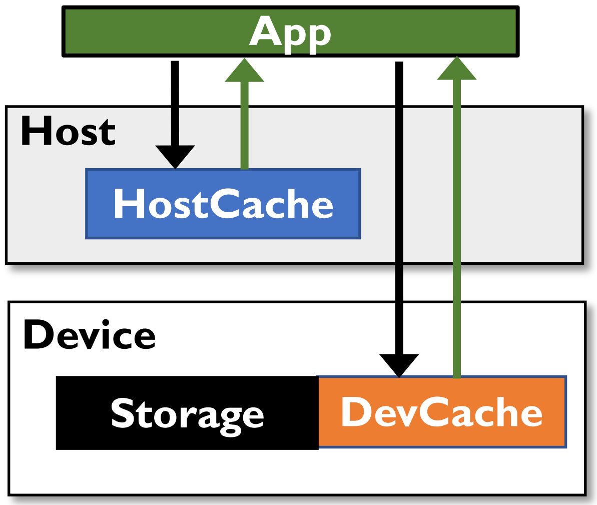


Vertical Caching

Horizontal Caching

OmniCache Overview

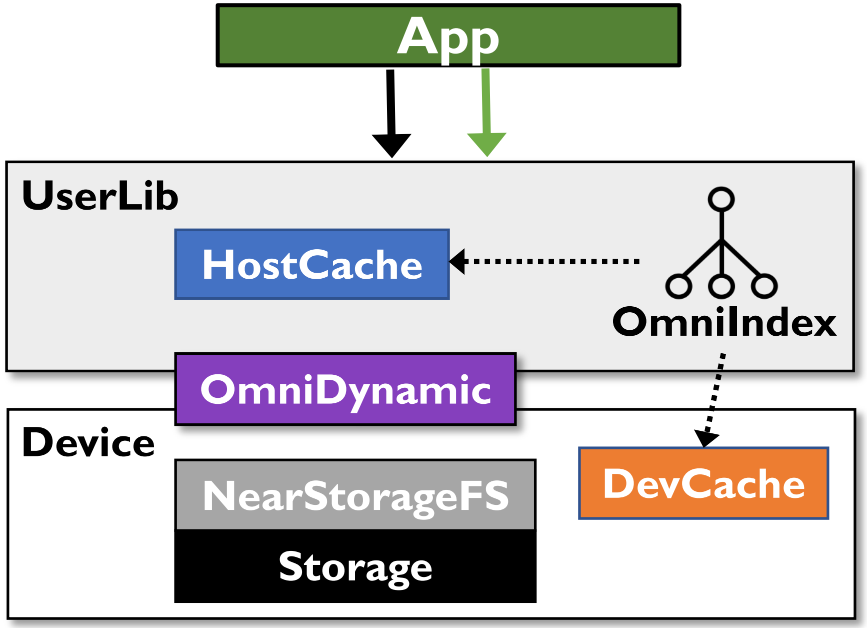
- **Horizontal** caching design to exploit host and device memory
- Accelerates I/O and data processing with **collaborative** caching
- Dynamically offloads requests to host or device
- OmniCache exploits CXL for reducing host-device communication costs



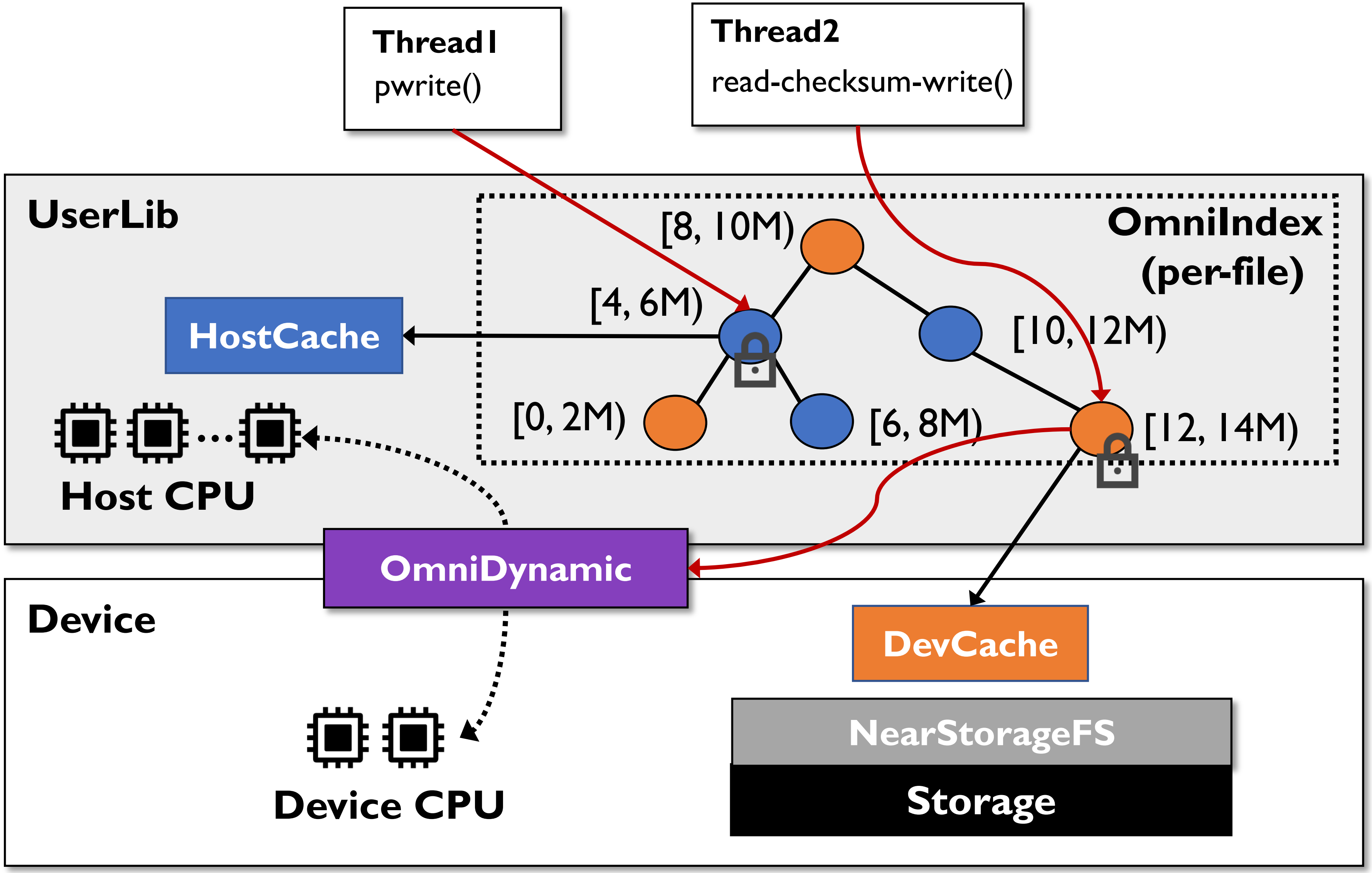
OmniCache

OmniCache Components

- **UserLib**
  - Support POSIX API
  - Provide predefined I/O and data processing functions
- **OmniIndex**
  - Provides a unified cache view across host and device
  - Delegates cache management to host
  - Fine-grained concurrency control
- **OmniDynamic**
  - Dynamically offloads requests between host and device
- **NearStorageFS**
  - Handles I/O and data processing request
  - Manages file data and metadata

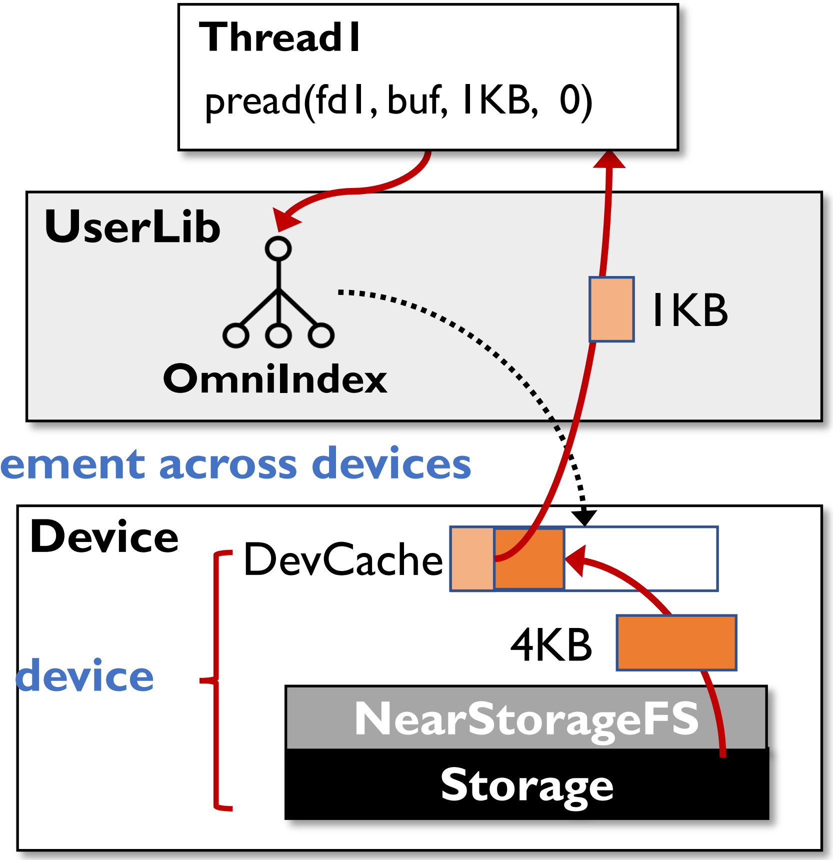


Concurrent I/O and Processing Example



Reduce Read/Write Amplification

- OmniCache reduces amplification by increasing data access on the nearest cache



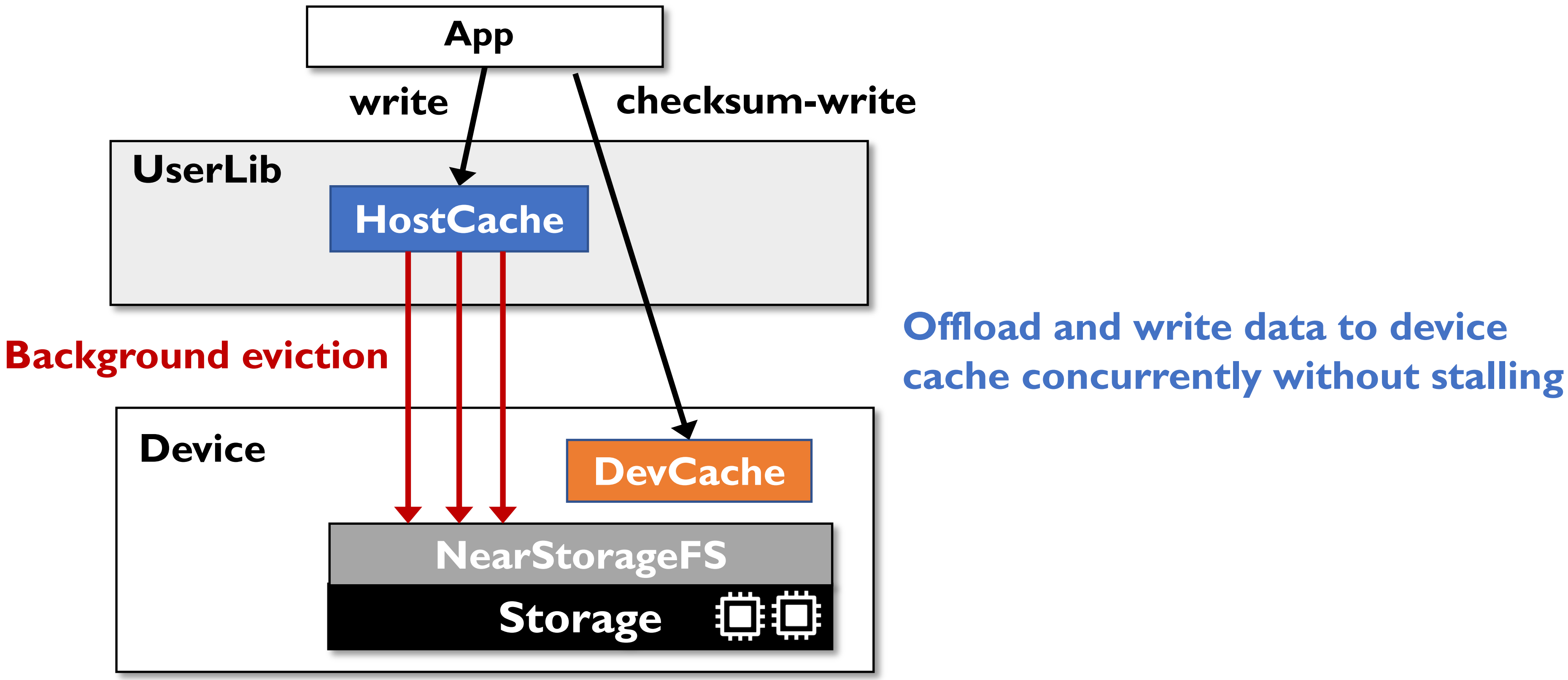
Only 1KB data movement across devices

Low access latency within the device

Near-storage cache overcomes alignment-related amplification issues

Concurrent I/O and Data Processing Support

- Minimizes application stalls by collaboratively using HostCache and DevCache

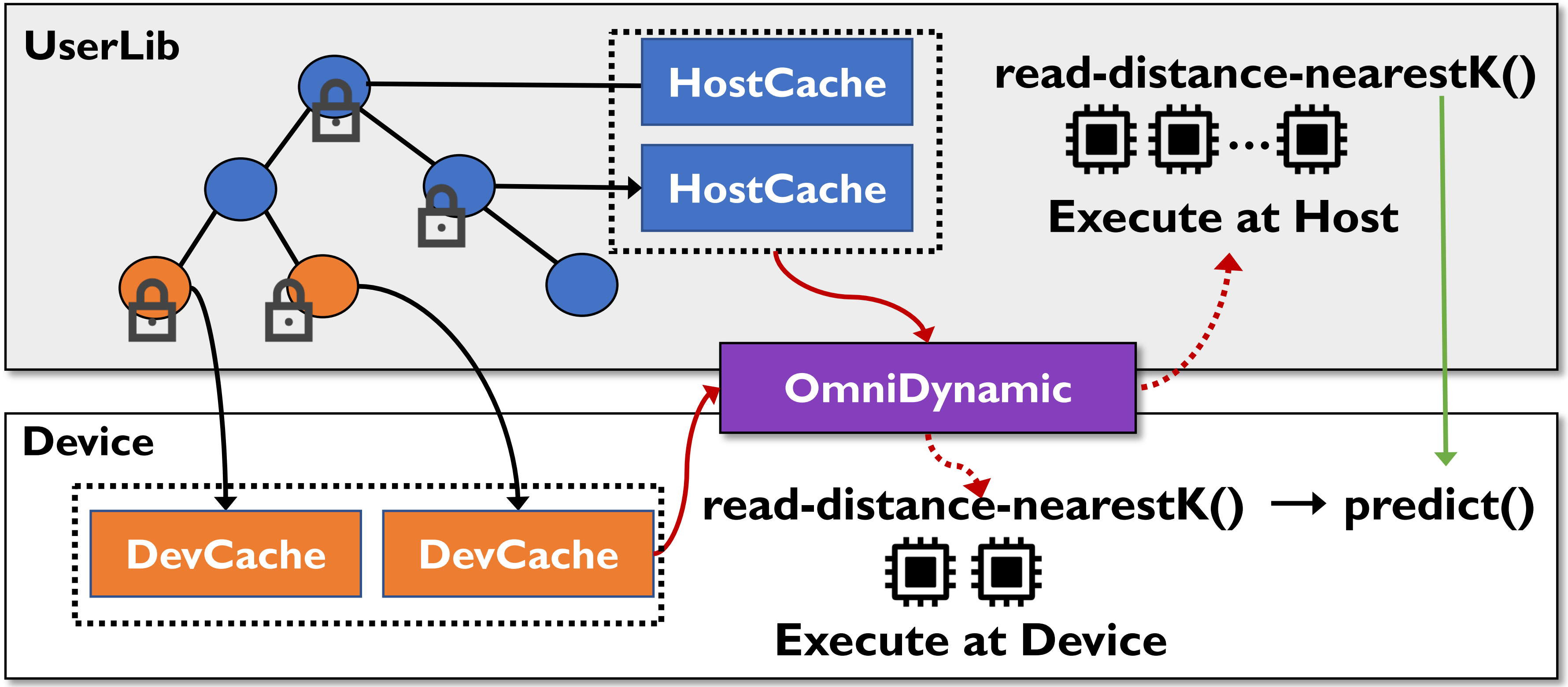


Offload and write data to device cache concurrently without stalling

Horizontal paradigm for caches can reduce frequent application stalls!

Collaborative Data Processing

- OmniCache collaboratively uses HostCache and DevCache for processing



How to dynamically determine where to process?

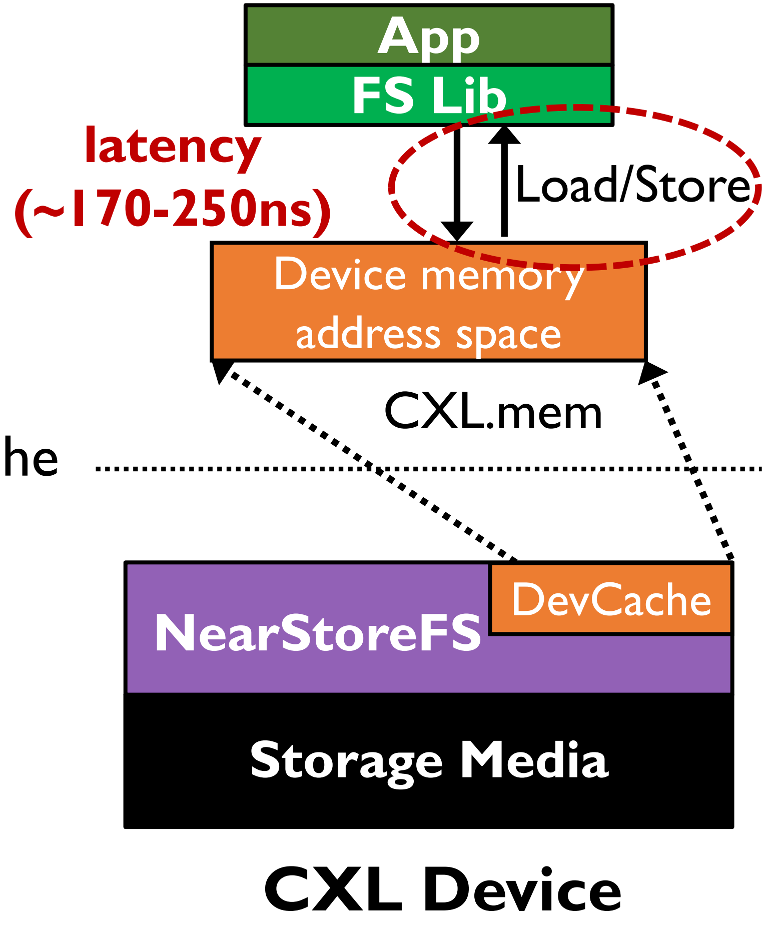
Model-driven Dynamic Offloading

- Processing speed depends on factors such as data ratio, processor cache, queuing delay
- OmniCache continuously monitors host/device resources before offloading
- Model estimates processing time across host and device to identify a request's location
- ***Model = Data Transfer Cost + Queuing Latency + Execution Time***

Please see paper for more details!

CXL Extensibility for OmniCache

- CXL can enable host compute to directly access device memory!
- CXL.mem maps device memory to the host as a NUMA node
- Reduces I/O queuing delays, and CPU polling overheads for OmniCache



Outline

- Background
- Motivation
- Design
- Evaluation
- Conclusion

Experimental Setup

- Hardware platform
    - Dual-socket 64-core Xeon Scalable CPU @ 2.6GHz
    - 512GB Intel Optane DC NVM

- Emulated in-storage FS (no programmable storage H/W)
    - Dedicate device threads for handling I/O requests
    - Add PCIe latency for all I/O operations
    - Reduce CPU frequency for device CPUs (and memory bandwidth)

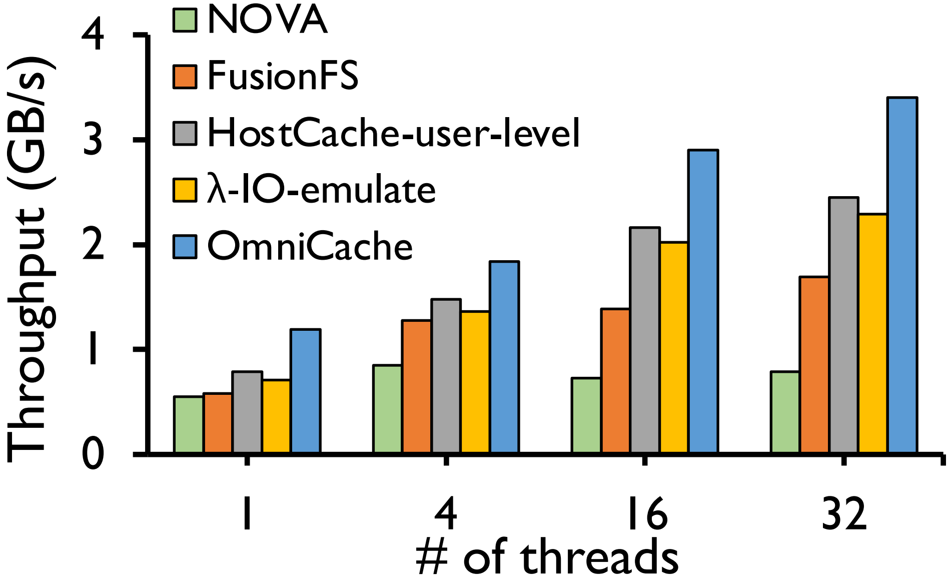
- State-of-the-art designs
    - NOVA [FAST' 16] (Kernel-level FS)
    - FusionFS and User-level host cache atop FusionFS [FAST '22] (Device-level FS)
    - Emulated λ-IO without FPGA but with OS caching [FAST '23] (near-storage design)

Evaluation Goals

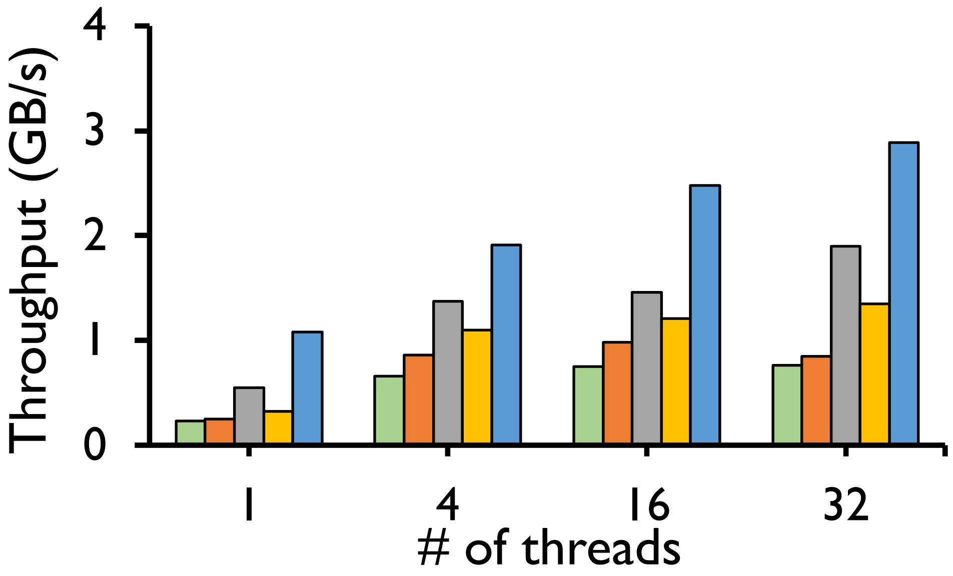
- Understand effectiveness of OmniCache for reducing I/O overheads
- Study and validate the benefits of OmniDynamic
- Understand the performance by using CXL.mem
- Discuss overall real-world application impact

Microbench

- Each thread issues 1KB I/O requests, resulting in a total workload size of 64GB
  - HostCache-user-level and lambda-IO-emulate employ 20GB host DRAM cache
  - OmniCache uses 16GB HostCache and 4GB DevCache



Random Read



Random Write

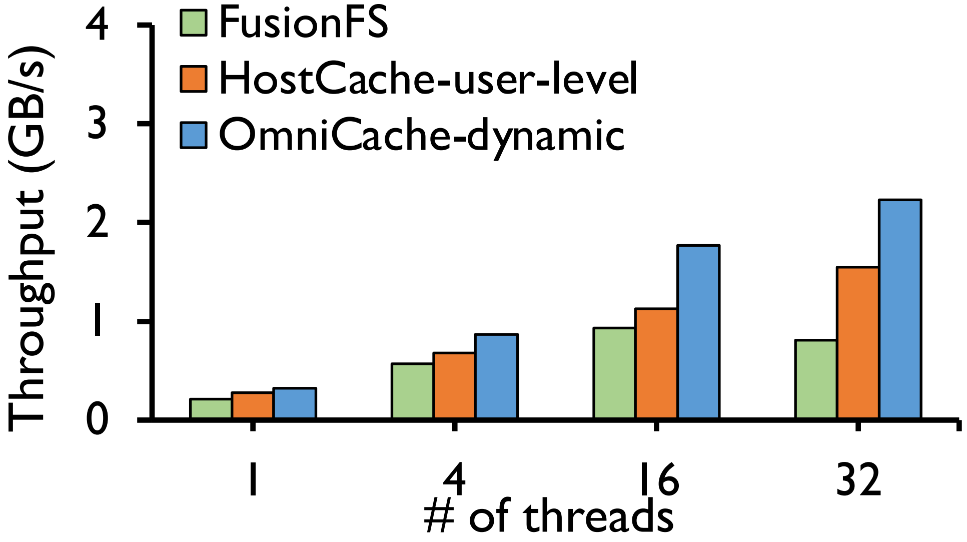
OmniCache significantly reduces data movement and eviction stalls

Evaluation Goals

- Understand effectiveness of OmniCache for reducing I/O overheads
- Study and validate the benefits of OmniDynamic
- Understand the performance of using CXL.mem
- Discuss overall real-world application impact

Evaluate and validate OmniDynamic

- Each thread randomly reads 4KB data blocks, calculates checksum, and writes it back
  - OmniCache-dynamic uses 16GB HostCache and 4GB DevCache



I/O + Data Processing (Read-CRC-Write)

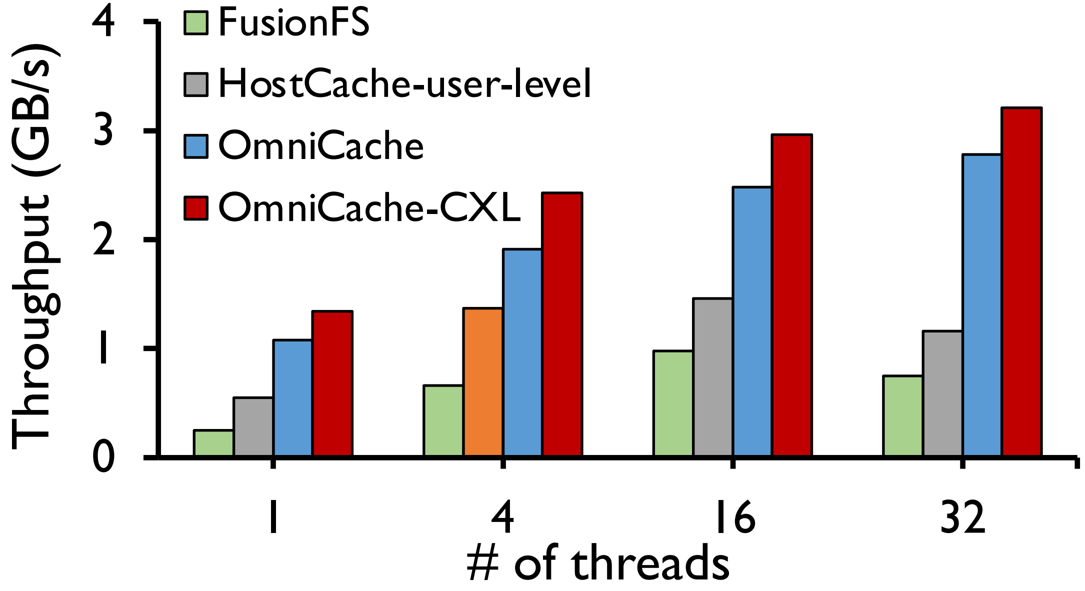
Improves performance by considering hardware/software factors such as queue delays, host and device load, near-data access

Evaluation Goals

- Understand effectiveness of OmniCache for reducing I/O overheads
- Study and validate the benefits of OmniDynamic
- Understand the performance of using CXL.mem
- Discuss overall real-world application impact

Understand the performance with CXL

- CXL emulation: we map device memory as a remote NUMA socket to the host CPUs, but local to the device CPUs



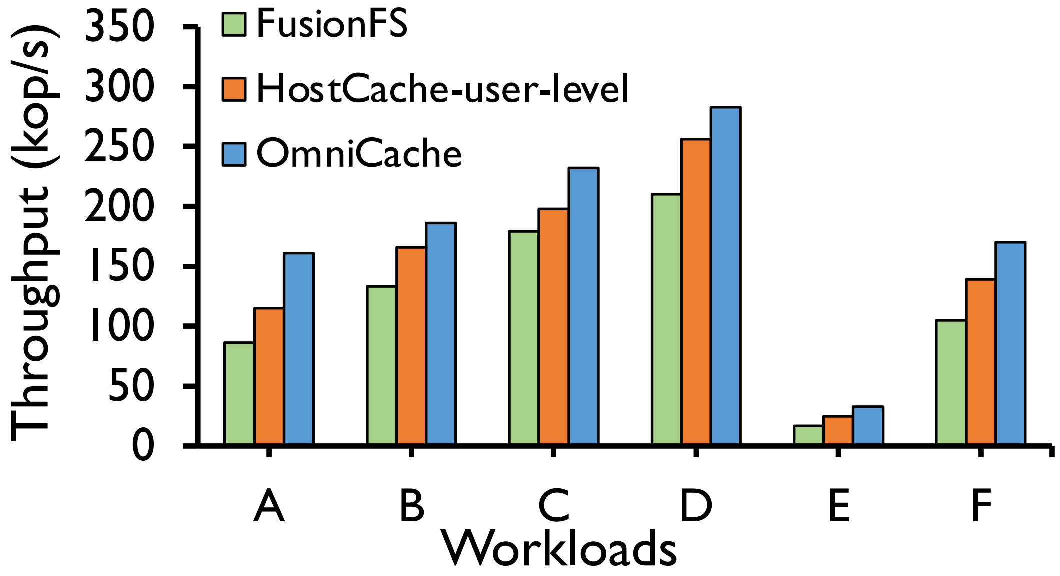
Random Write

OmniCache with CXL improves performance by avoiding I/O queuing delays and CPU polling cost

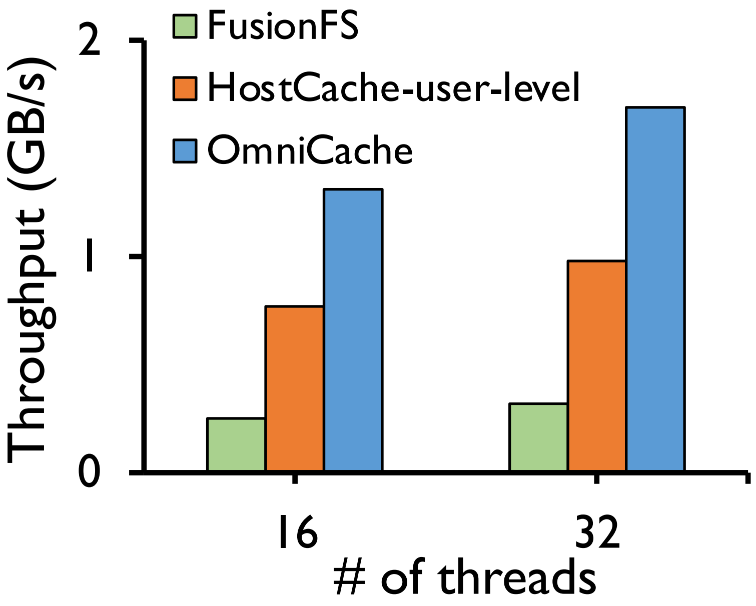
Evaluation Goals

- Understand effectiveness of OmniCache for reducing I/O overheads
- Study and validate the benefits of OmniDynamic
- Understand the performance by using CXL.mem
- Discuss overall real-world application impact

Real-world Application



YCSB on LevelDB



KNN

OmniCache also shows high throughput gains on real-world applications

Outline

- Background
- Motivation
- Design
- Evaluation
- Conclusion

Conclusion

- It’s critical to use all compute and memory resources in the system
  - Our approach: OmniCache, a **horizontal** and scalable caching design

- OmniCache effectively reduces data amplification by collaboratively using host and device caches

- OmniCache accelerates I/O and data processing by concurrently using host and device CPUs

- CXL provides substantial benefit with the unified caching design

https://github.com/RutgersCSSystems/omnicache-fast24-artifacts