RUTGERS SAMSUNG EPFL

Context-aware Prefetching for Near-Storage Accelerators

Jian Zhang, Marie Nguyen, Sanidhya Kashyap, Sudarsun Kannan

Caching and Prefetching are Important

- Caching and prefetching are both widely explored to enhance I/O performance

- OS page cache (host-level caching) supports prefetching data from the device to the host to reduce cache misses
  - Prefetching system call in Linux (e.g., readahead, fadvise)

- Prefetching techniques often overlooked for near-storage accelerators!
  - Challenge of near-storage accelerators: limited memory for caching and prefetching

Outline

- Background
- Motivation
- Design
- Evaluation
- Conclusion

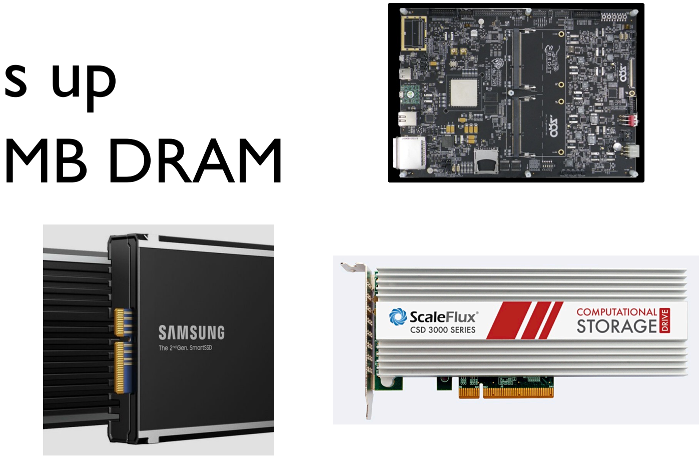
Hardware Trends

- Near-storage accelerators (e.g., CSDs) can improve I/O performance and reduce data movement costs
  - E.g., Samsung SmartSSD, Newport CSDs and ScaleFlux CSDs

- Hardware resources have up to 4-16 cores and around 4GB DRAM

- Unfortunately, device DRAM is still limited and frequently fills up
  - reserved for internal tasks (e.g., internal storage software or FTL: 1MB DRAM per 1GB)

- Careful management of device-level memory is crucial

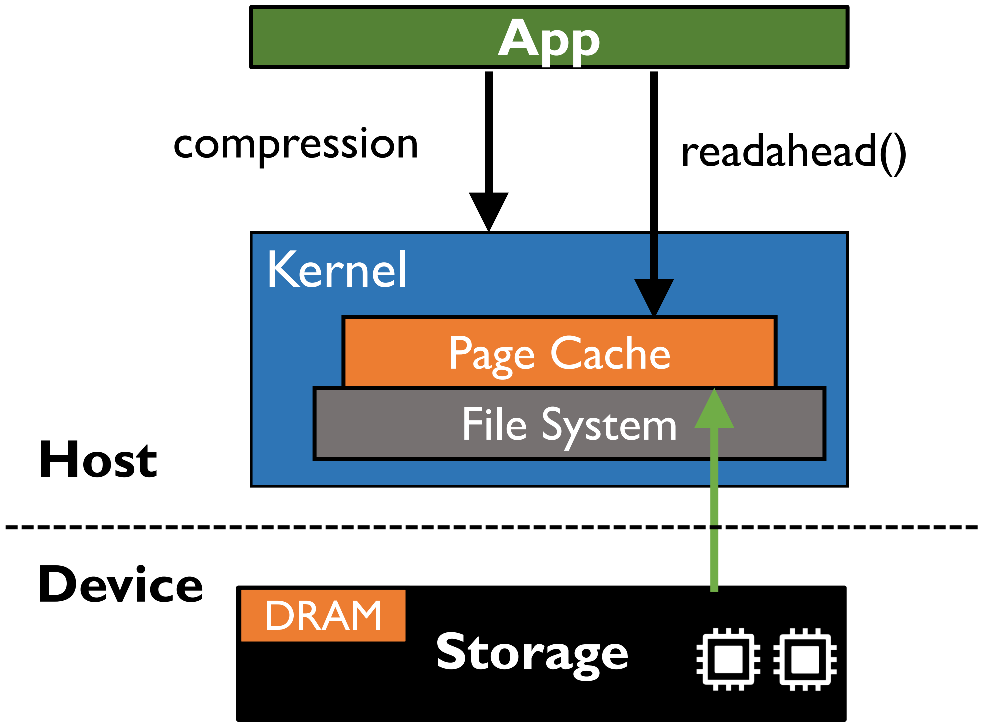


Computational Storage

Caching and Prefetching Designs

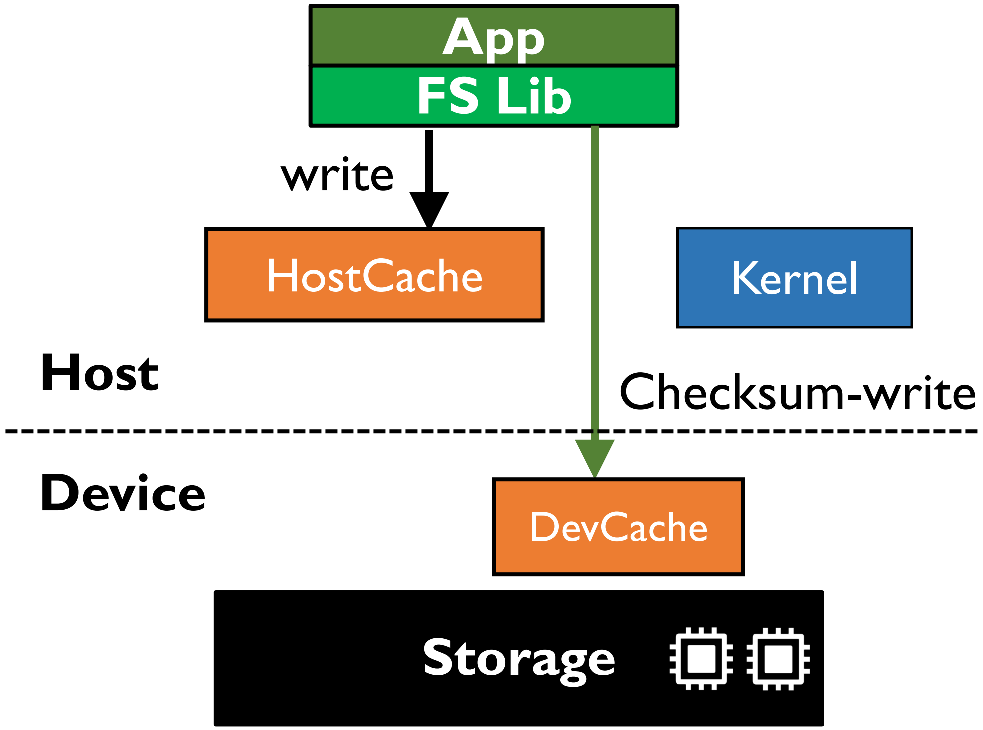
Host-level Caching

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



Near-storage Caching

(OmniCache [FAST ‘24])

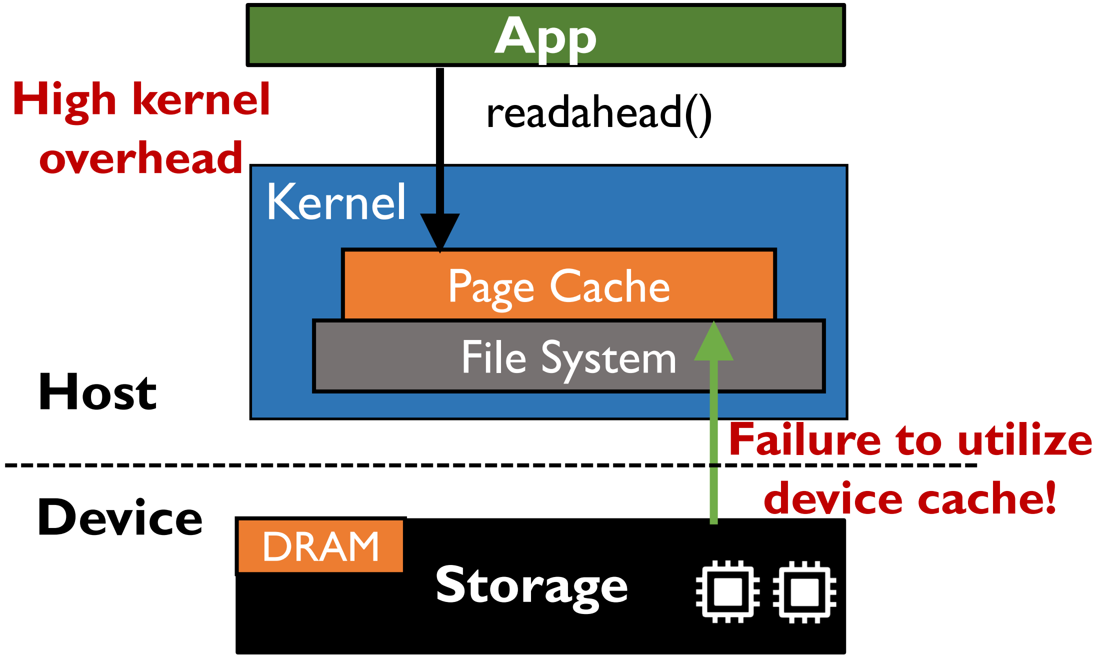


Outline

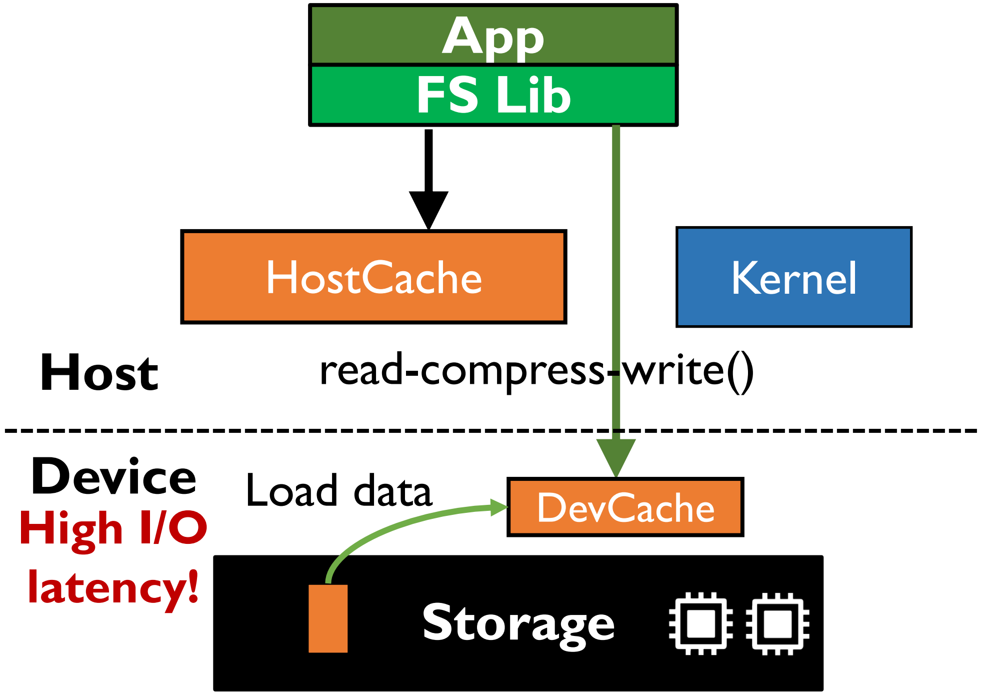
- Background
- Motivation
- Design
- Evaluation
- Conclusion

Lack of Prefetching for Device Cache

- Host-level caching supports system call like *readahead* by only prefetching data to host cache
- Near-storage caching does not support prefetching for device cache



Host-level Caching



Near-storage Caching

Challenges to Prefetch Data in Device

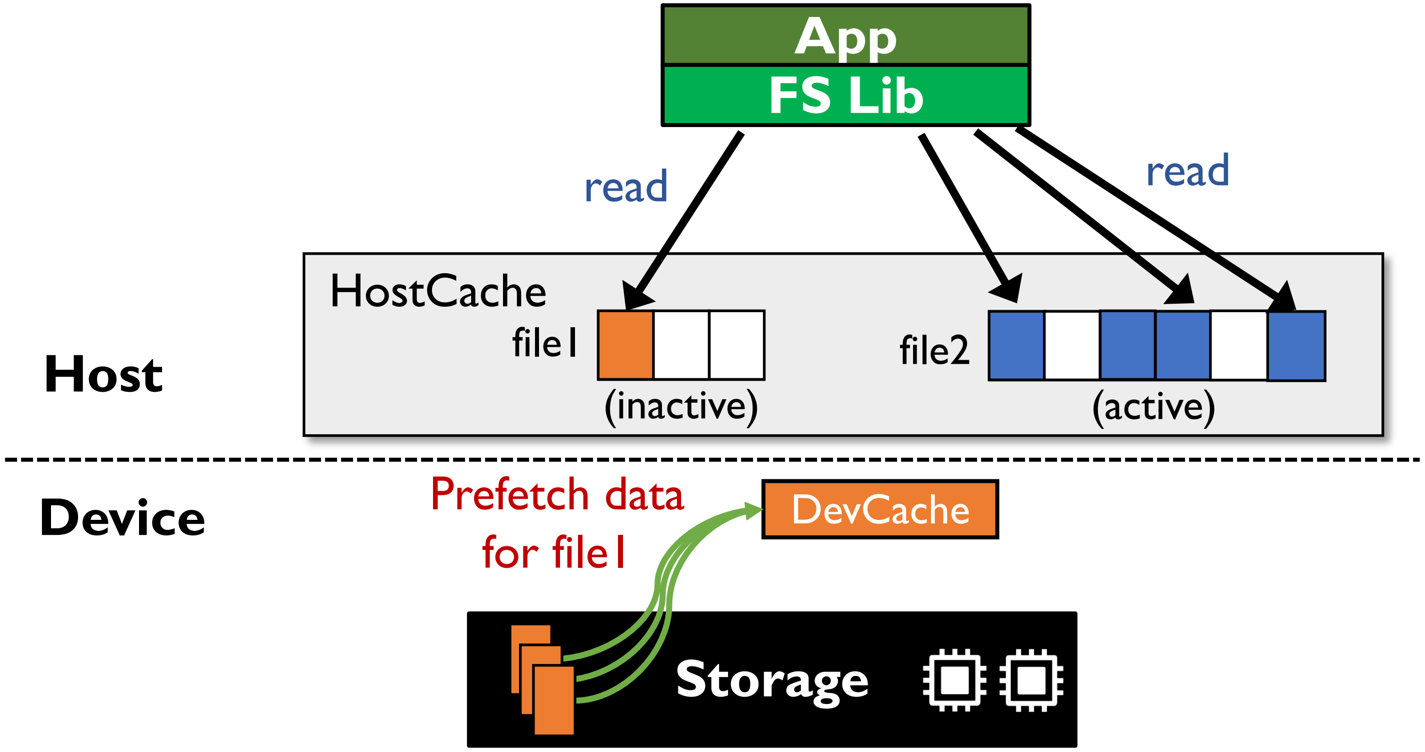
- Device is unaware of the files currently in use by the host (**lack of context**)

- Managing smaller near-storage (DRAM) caches critical by supporting timely eviction

- Applying kernel prefetching to the device is infeasible due to high software overheads (e.g., system call, traversing the per-file xarray)

Device is Unaware of Host File Access

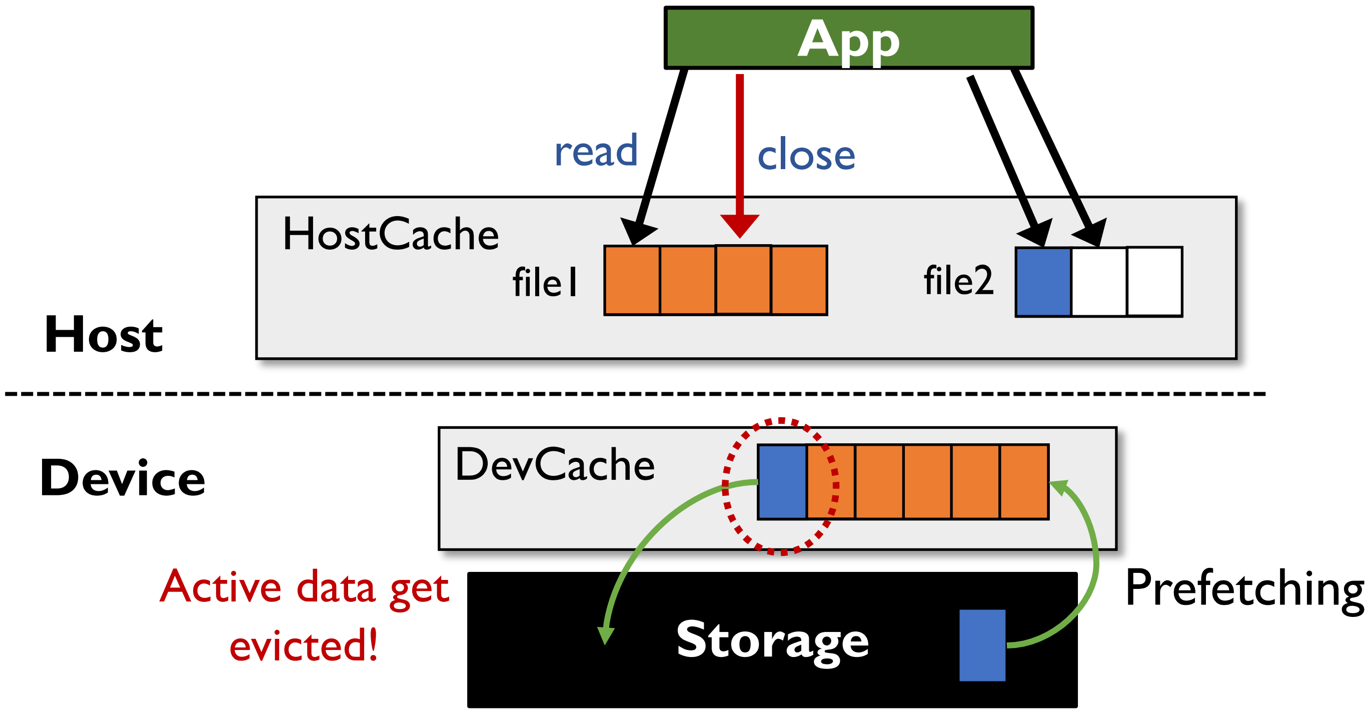
- Device lacks a context on current active file or object being accessed
    - Prefetching on inactive files



Near-storage Caching

Timely Cache Eviction

- Limited device DRAM size used for near-storage caching leads to frequent cache eviction of active files (and blocks)



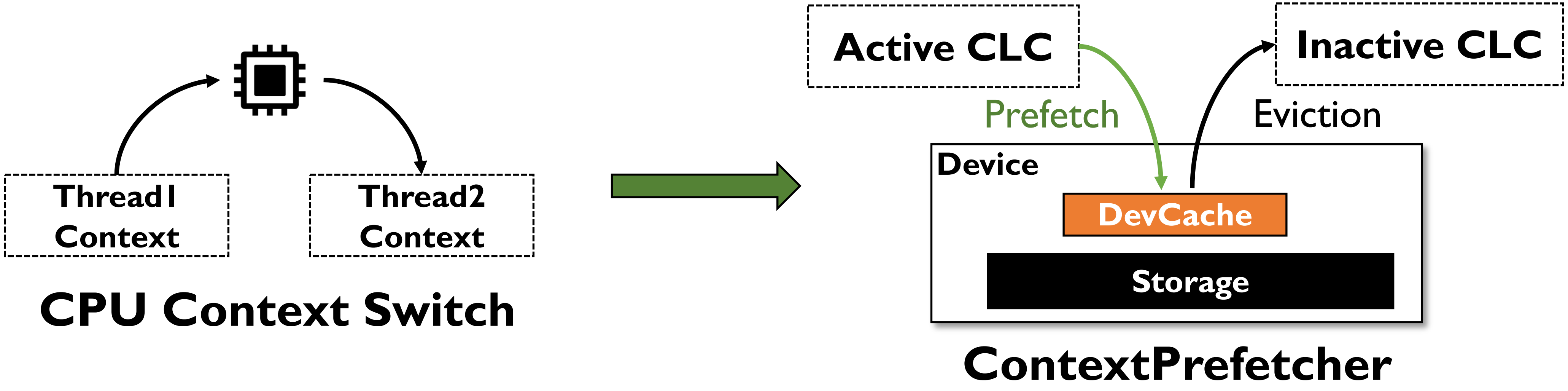
Near-storage Caching

Outline

- Background
- Motivation
- Design
- Evaluation
- Conclusion

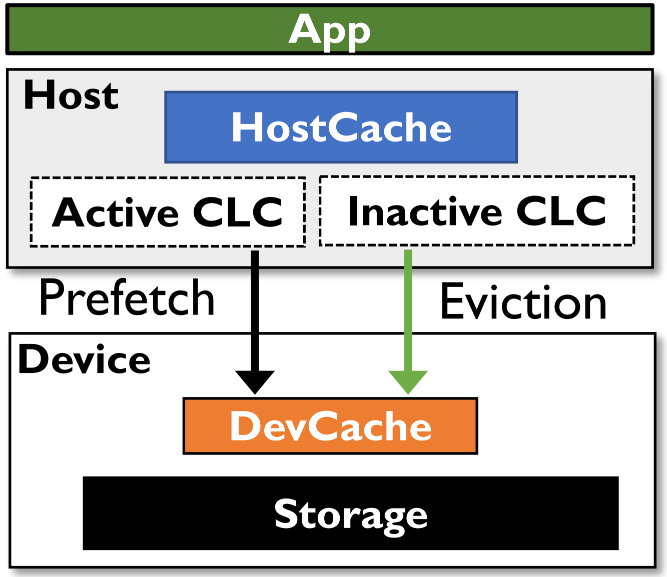
Our Solution: ContextPrefetcher

Prefetching based on **Cross-layered Context (CLC)**, a virtual entity that spans across the host and the device and is used for managing **active and inactive data**



ContextPrefetcher Overview

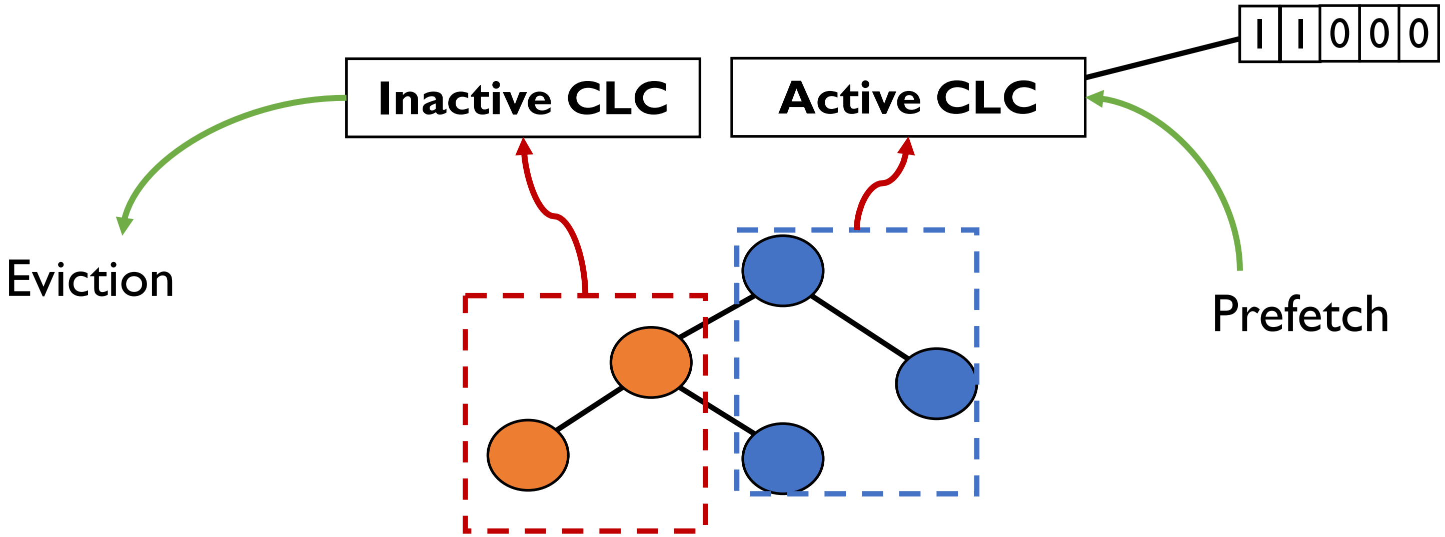
- A host-guided high-performant prefetching framework
- Prefetch based on Cross-layered Context (CLC)
  - A virtual entity that spans across the host and the device
- Use CLC to track active and inactive data such as files, objects or a range of blocks
- Supports timely eviction of data associated with inactive CLC



ContextPrefetcher

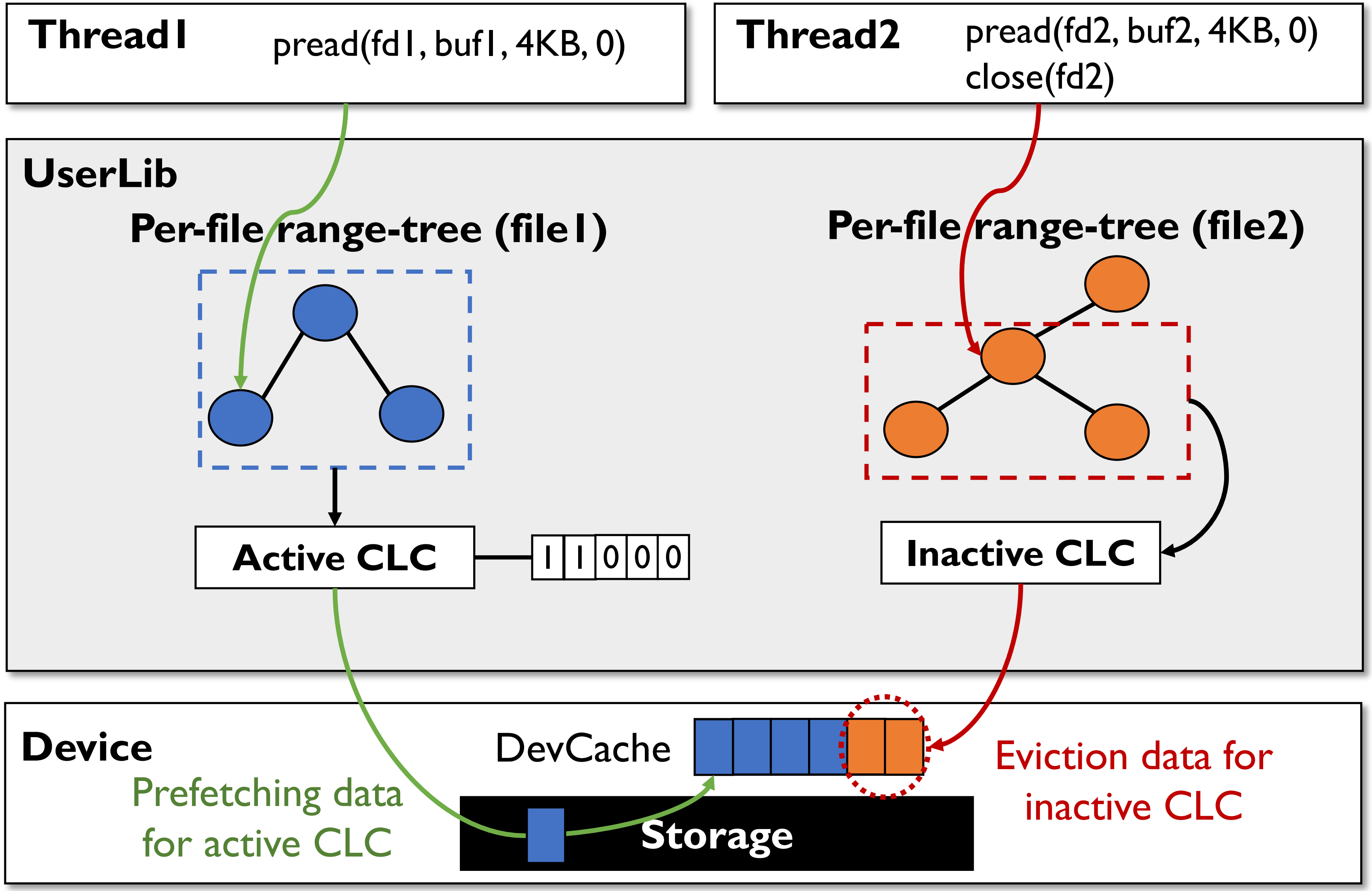
Cross-layered Context (CLC)

- Cross-layered Context is a virtual entity used for tracking active/inactive data
  - It could be files, objects (within object stores), or a range of blocks
  - Each node in CLC has a bitmap to indicate each page is prefetched or not



Per-file range-tree (file1)

ContextPrefetcher Example



Thread1 pread(fd1, buf1, 4KB, 0)

Thread2 pread(fd2, buf2, 4KB, 0)
close(fd2)

UserLib

Per-file range-tree (file1)

Per-file range-tree (file2)

Active CLC

1	1	0	0	0

Inactive CLC

Device

DevCache

Storage

Prefetching data for active CLC

Eviction data for inactive CLC

ContextPrefetcher Challenges and Future Work

- Efficiently detect active and inactive context
  - Possible solution: ML-based detection

- Collaboratively prefetch the data of files across host and device
  - Possible solution: Parallel prefetching across devices or files

- Communication overhead between host and device for prefetching
  - Possible solution: Vector-based I/O

Outline

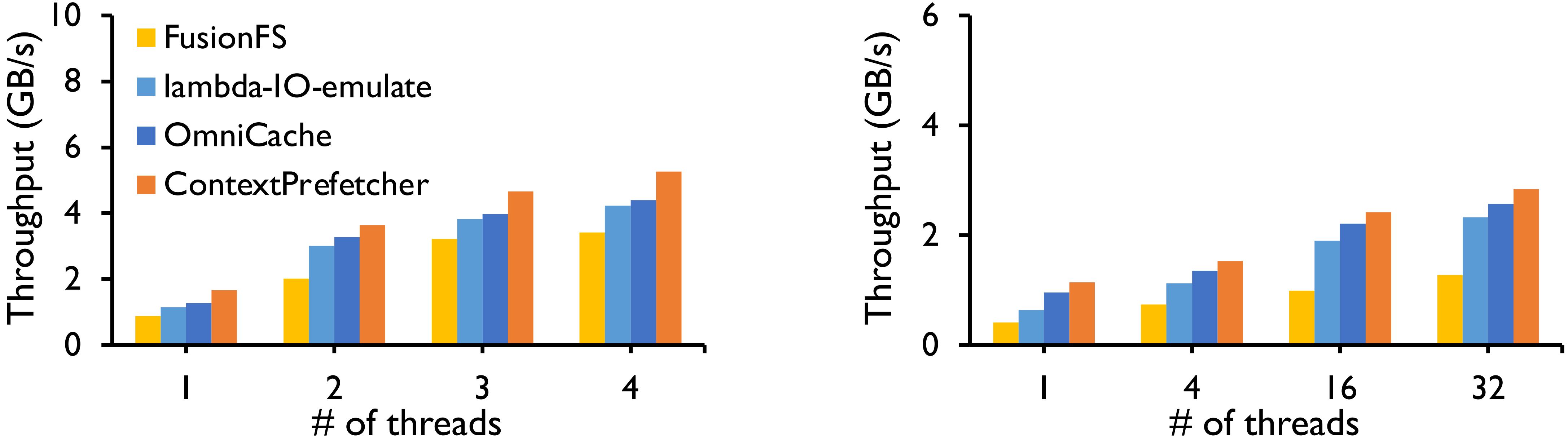
- Background
- Motivation
- Design
- Evaluation
- Conclusion

Preliminary Evaluation

- 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
  - FusionFS [FAST ‘22] (without caching and prefetching support)
  - Emulated λ-IO without FPGA but with host-level OS caching and prefetching [FAST ‘23] (near-storage design)
  - OmniCache [FAST ‘24] (unified caching design for near-storage accelerators)

Microbench

- Each thread concurrently opens multiple files, performs sequential/random read
  - Employ total 20GB cache size
  - OmniCache and ContextPrefetcher use 1GB DevCache



Sequential Read

Random Read

ContextPrefetcher improves I/O performance by efficient prefetching and eviction

Outline

- Background
- Motivation
- Design
- Evaluation
- Conclusion

Conclusion

- ContextPrefetcher: A novel context-aware prefetching approach for near-storage devices
  - Cross-layered Context (CLC), a virtual entity that spans across the host and the device

- ContextPrefetcher provides efficient prefetching and eviction based on CLC

- Achieves significant performance gains during our preliminary evaluation

Thanks! Q&A?

I’m on the job market!