EvFS: User-level, Event-Driven File System for Non-Volatile Memory

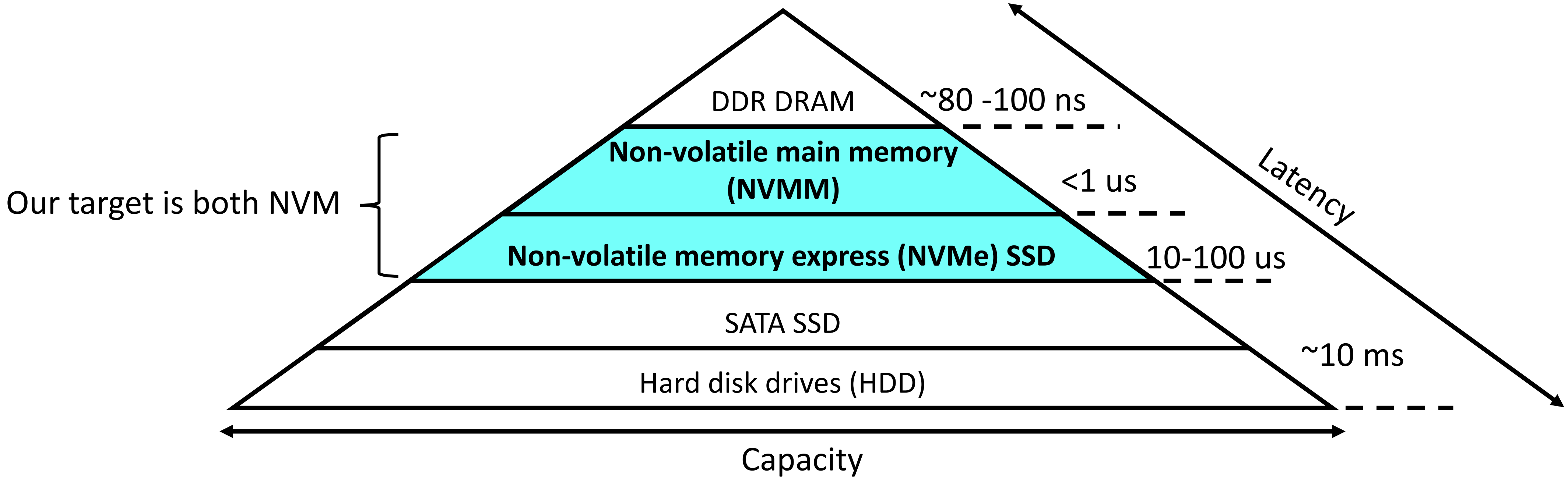
Takeshi Yoshimura, Tatsuhiro Chiba, Hiroshi Horii

IBM Research – Tokyo

HotStorage 2019

Non-volatile memory (NVM) is fast storage

- Enables low-latency data processing with persistency and high capacity
  - –Extremely lower latency (1 - 100 us) than SATA SSD and HDD (-10 ms)
  - –Higher capacity than DRAM
- Available as non-volatile main memory (NVMM) and NVM Express (NVMe)
  - –Apps can access both NVM types through file systems (FS) such as ext4

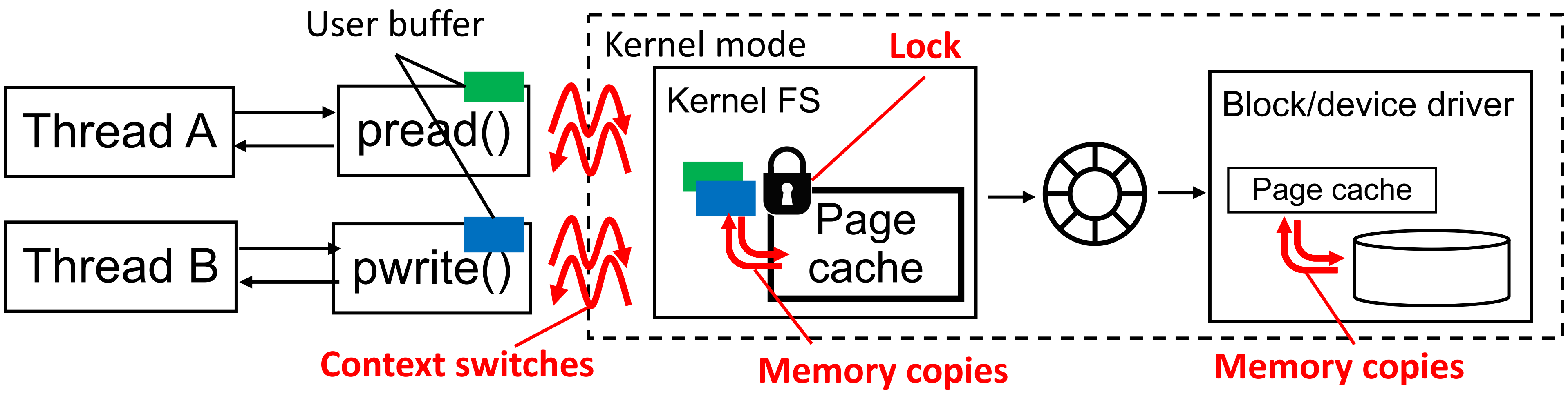


Modified figure of PMDK documentation (https://docs.pmem.io/getting-started-guide/introduction)

Kernel FS is a huge overhead for fast storage

- The major overheads are reported in [Peter '14], [Volos '14], etc.
  - User-kernel context switches
  - Locks
  - Memory copies around page cache
  - Other complex FS features

- In our experience, ext4 spent >5 us for *in-memory* 64B write()*
  - No fsync and persistent writes, but 500 % time for NVM latency



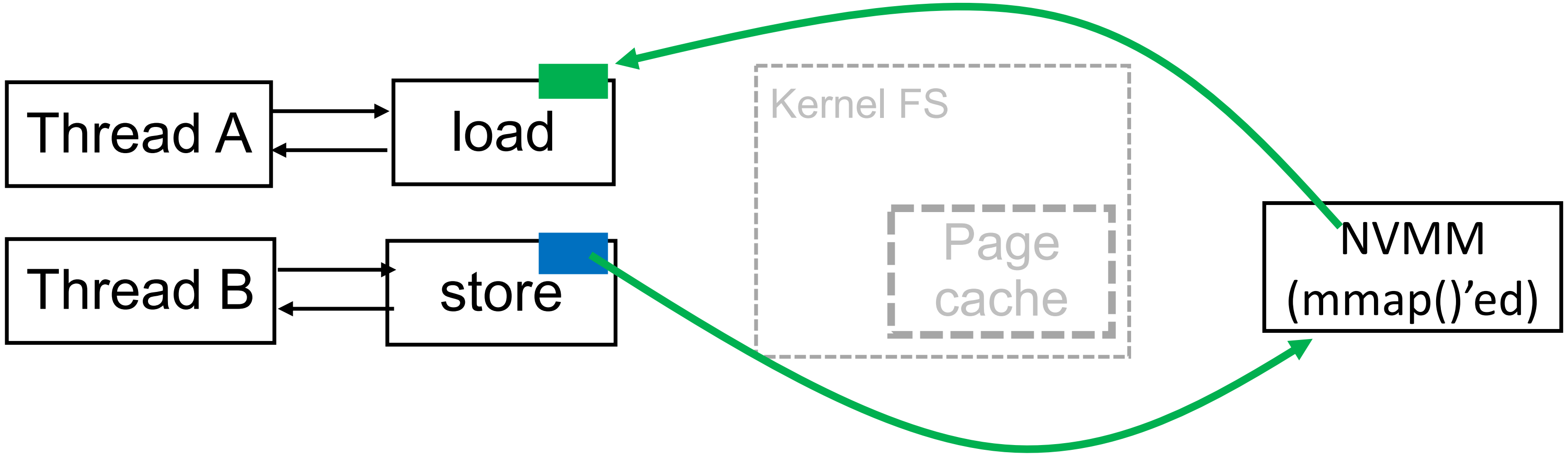
* experiments on
IBM Power System AC922
1.1TB RAM
160 logical Power9 cores
PCIe3 x8 6.4 TB NVMe
https://www.ibm.com/support/knowledgecenter/8335-GTH/p9hcd/fcec5e.htm
Ubuntu 18.04LTS, Linux 4.17
SPDK 19.0, DPDK 18.02
ext4: disabled journaling and readahead

Existing approach: Direct-access (DAX) FS

- Enables direct mapping of NVM to userspace
  - Linux ext4-DAX, PMFS [Dulloor ’14], Aerie [Volos ’14], SPDK* BlobFS

*Storage performance development kit (https://spdk.io/)

- Simplifies FS architecture
  - e.g., remove page cache to avoid redundant memory copies
- Provides POSIX APIs and DAX interfaces (e.g., mmap, get/put) to apps



Limitations of existing DAX FS

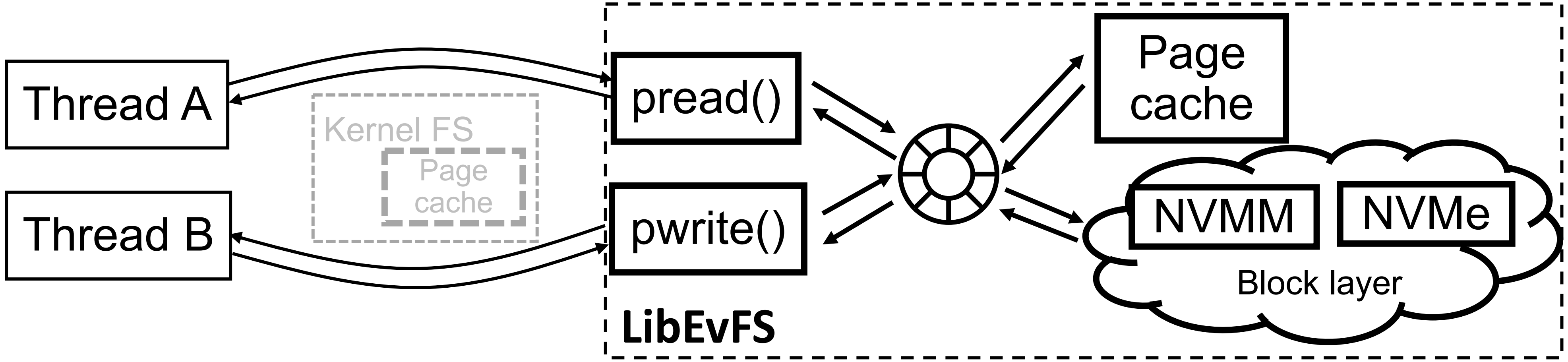
- DAX interfaces are non-portable
  - Many apps depend on POSIX file I/O, e.g., pread()
  - Apps need difficult device management such as cache flushes
- POSIX file I/O is suboptimal
  - Page cache removal can slowdown apps due to high write latency of NVM [Ou ’14]
  - DAX FS running in the kernel requires context switches for POSIX file I/O
  - BlobFS requires locks for page cache despite its limitation of access patterns

Direct-access FS	DAX interface	Running mode	Page cache
Linux ext4-DAX	mmap	Kernel	No
PMFS [Dulloor ’14]	mmap	Kernel	No*
Aerie [Volos ’14]	put/get	User	No
SPDK BlobFS	No	User	No random accesses

*HiNFS [Ou ’16] introduced Page cache in PMFS

Our proposal: EvFS

- Optimizes POSIX file I/O for general Linux apps on NVM
  - Least user-kernel context switches with full user-level storage stack
  - Lock-free page cache with event-driven architecture
  - Dynamic link library exposing POSIX APIs
- Provides direct I/O as a DAX interface
  - Enable apps to selectively bypass page cache for file I/O
- Built on top of SPDK block layer that supports both NVMM and NVMe
  - Can be extended to RAID, logical volumes, and other extended storage features



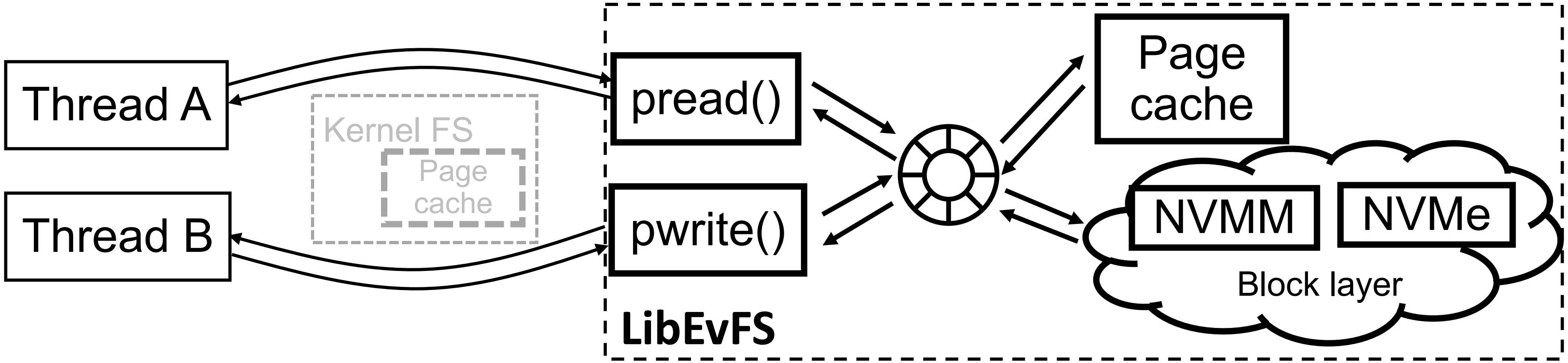
Contributions

- Show early design and implementation of user-level, event-driven FS for NVM
  - Not completed implementing all POSIX semantics yet
  - Not implemented journaling yet
- Report preliminary microbenchmark results with FIO and NVMe
  - Other benchmarks and NVMM evaluation are future work

Direct-access FS	DAX interface	Running mode	Page cache
Linux ext4-DAX	mmap	Kernel	No
PMFS [Dulloor '14]	mmap	Kernel	No
Aerie [Volos '14]	put/get	User	No
SPDK BlobFS	No	User	No random accesses
EvFS	Direct I/O	User	Yes

Key design of EvFS

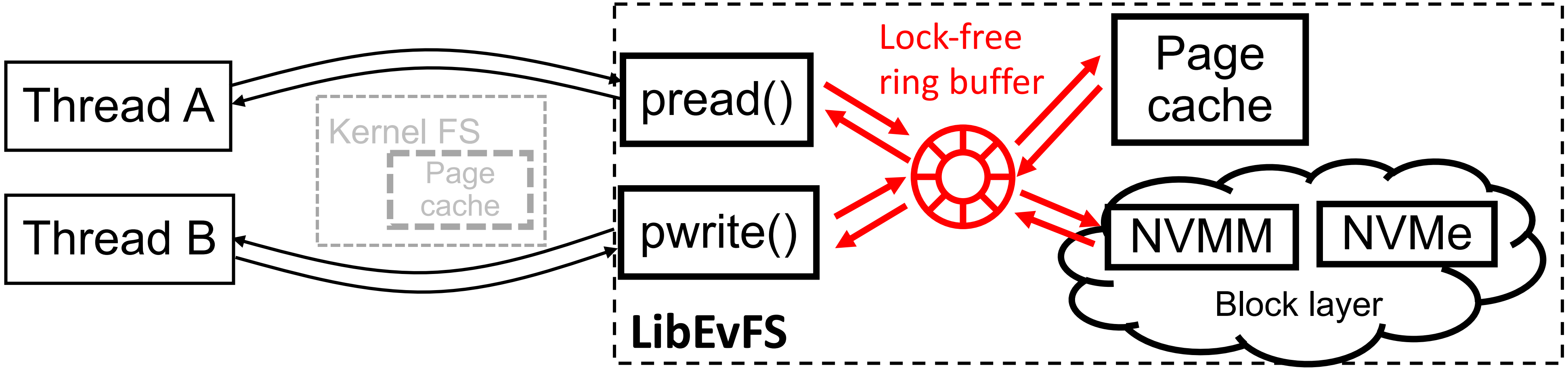
- Event-driven architecture
- A dynamic link library exposing POSIX APIs
- User-level storage stack



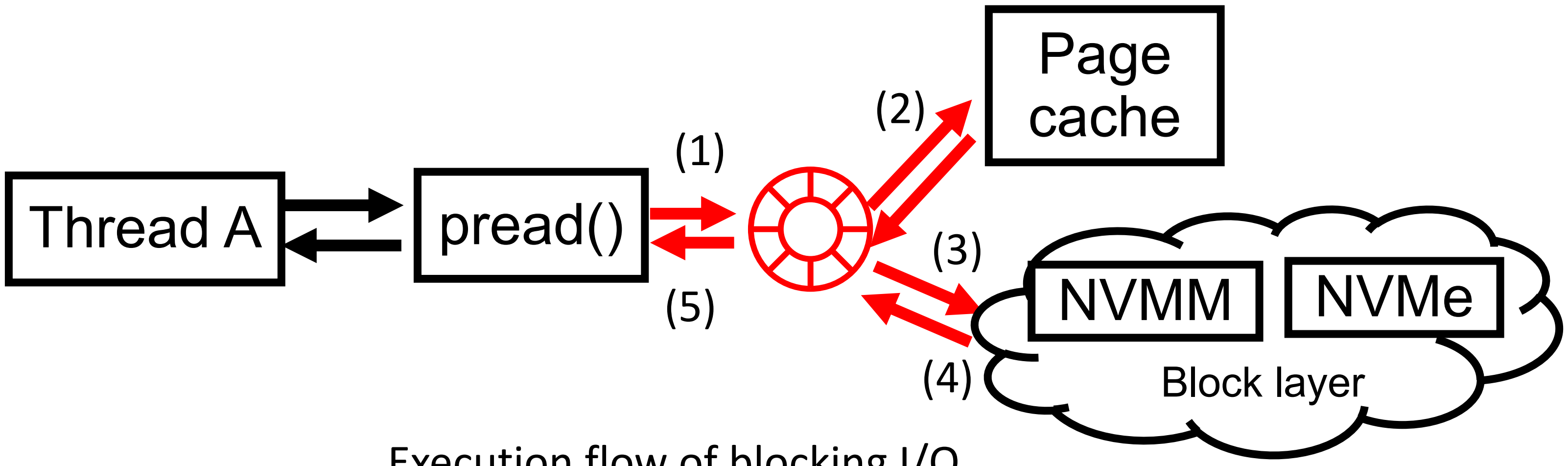
Event-driven architecture

- Execute all FS operations including page cache as asynchronous events
  - Create lock-free ring buffers to manage event descriptors
  - Run poller threads that atomically execute events, i.e., without locks
    - Eventually convert events into low-level requests to NVM
    - Execute I/O polling and notify its completion through callbacks

- Minimize the latency of POSIX file I/O
  - For blocking I/O, FS can reduce locks and coalesce I/O
  - For non-blocking I/O, apps can return immediately after submitting an event



Example execution flow



Execution flow of blocking I/O

(1)pread() called by apps enqueues file I/O and sleeps

(2)Page cache parses file I/O and submit a block I/O event

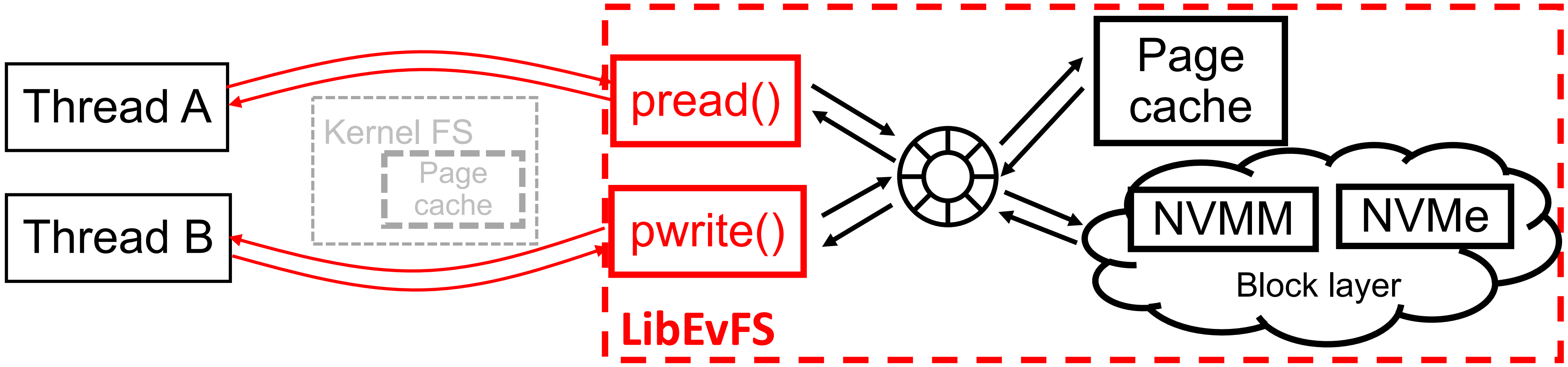
(3)Block layer parses and submits the I/O to NVM and executes I/O polling

(4)If I/O is completed, the block layer calls the callback for page cache

(5)The callback notifies the I/O completion to the sleeping context

Dynamic link library exposing POSIX APIs

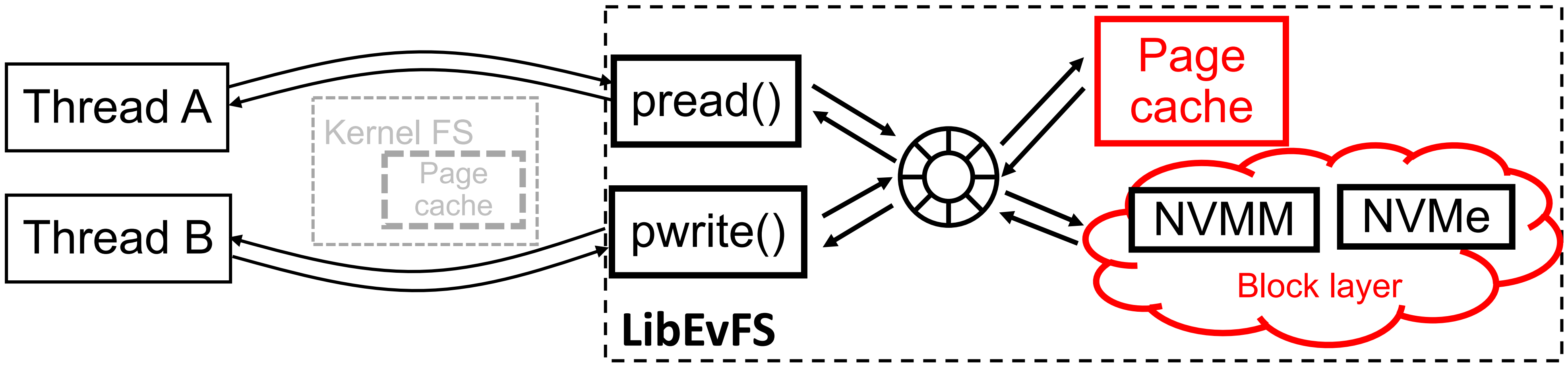
- EvFS exposes POSIX functions (e.g., pread) with its dynamic link library
  - Apps have to load libEvFS before LIBC and define device configs and mounted path
- The POSIX functions invoke EvFS for file I/O under the mounted path
  - Non-file I/O or accesses outside of the mounted path are redirected to LIBC
  - The EvFS library creates a private mount point for an app
- Hook thread-creation APIs in LIBC to minimize the latency
  - Create per-thread I/O channel and memory pool
    - Avoid thread contentions and system calls for memory allocations for event descriptors



User-level storage stack

- EvFS is built on top of SPDK Blobstore to manage NVM data
  - Regard BLOB, a management unit of NVM data in Blobstore, as inode as done by BlobFS
  - Emulate a directory structure with special BLOBs that have pointers to other BLOBs
  - Support user-level block drivers of SPDK NVMe and PMDK NVMM
    - Can also run with various advanced block drivers (e.g., RAID) in SPDK

- EvFS introduces Linux-like page cache at userspace
  - Cache NVM data in device page-granularity with offset as a key
  - Allow bypassing page cache with O_DIRECT in open() flags



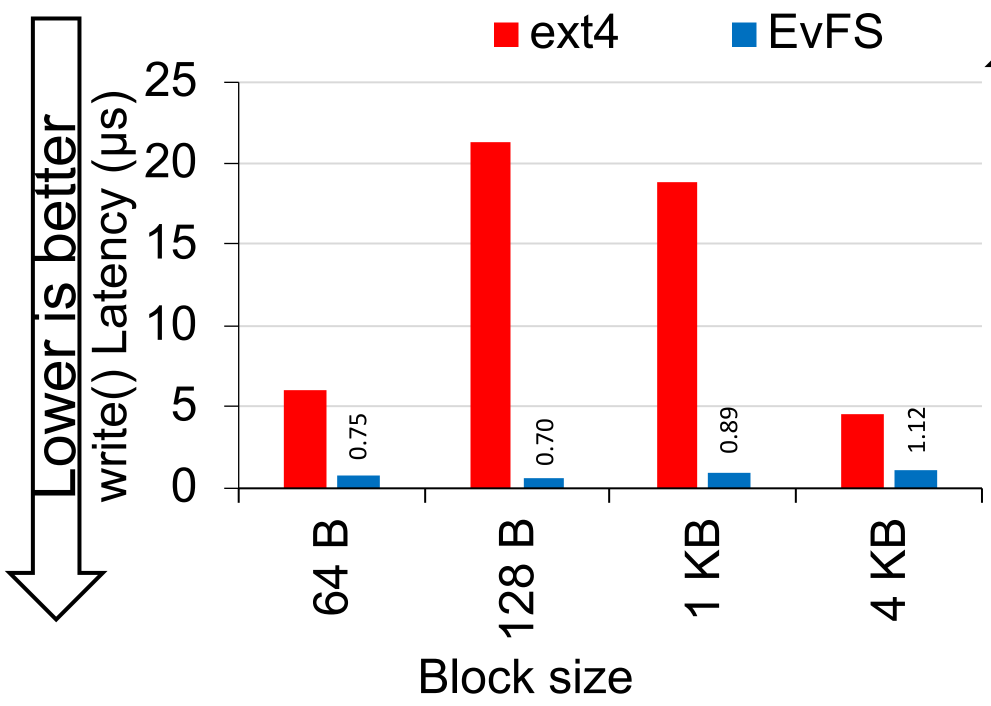
Preliminary evaluation

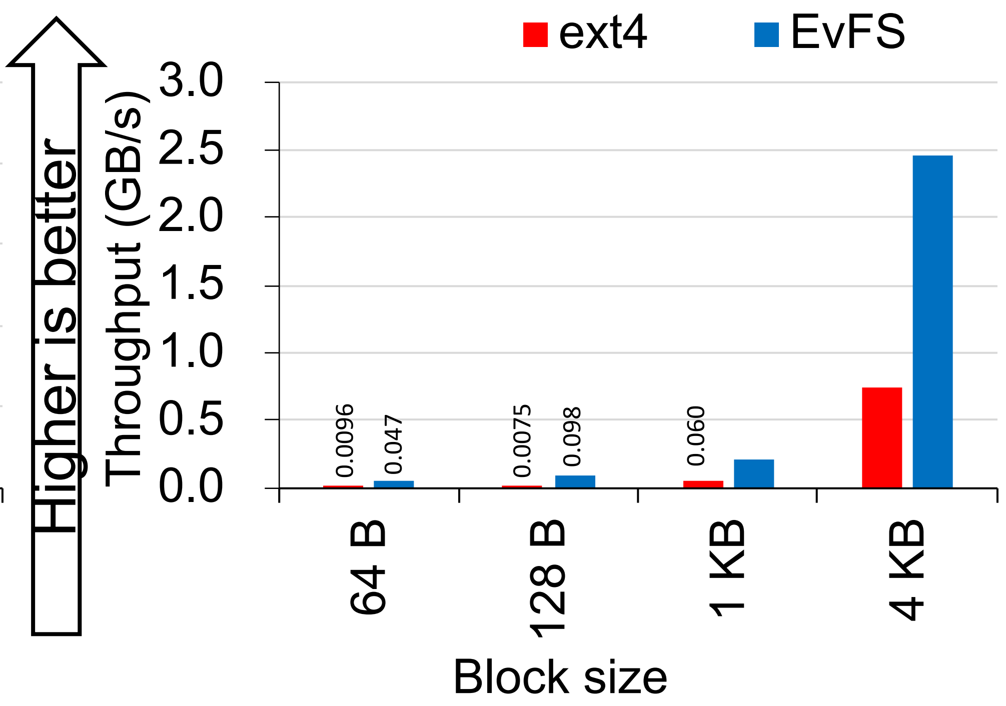
- Compare EvFS and ext4 performance with FIO
  - Evaluate random access latency and throughput with a single thread
  - Measure non-blocking writes and blocking reads/writes with/without direct I/O
  - Disable the journaling of ext4 and readahead
  - Suppose that we have enough memory

- Environment: IBM Power System AC922
  - 2 sockets x 20 cores x 4 SMT (POWER9 3.8 GHz), 1 TB RAM
  - Ubuntu 18.04 LTS, Linux 4.17
  - NVMe: PCIe3 x8 6.4 TB https://www.ibm.com/support/knowledgecenter/8335-GTH/p9hcd/fcec5e.htm

Result 1/3: Non-blocking writes

- EvFS reached ~0.7 us at 64 and 128 B writes
  - –ext4 showed 5 - 20 us
- EvFS showed up to 2.5 GB/s with a single thread
  - –Both EvFS and ext4 write only page cache
  - –Minimized latency by context switch elimination and event-driven architecture

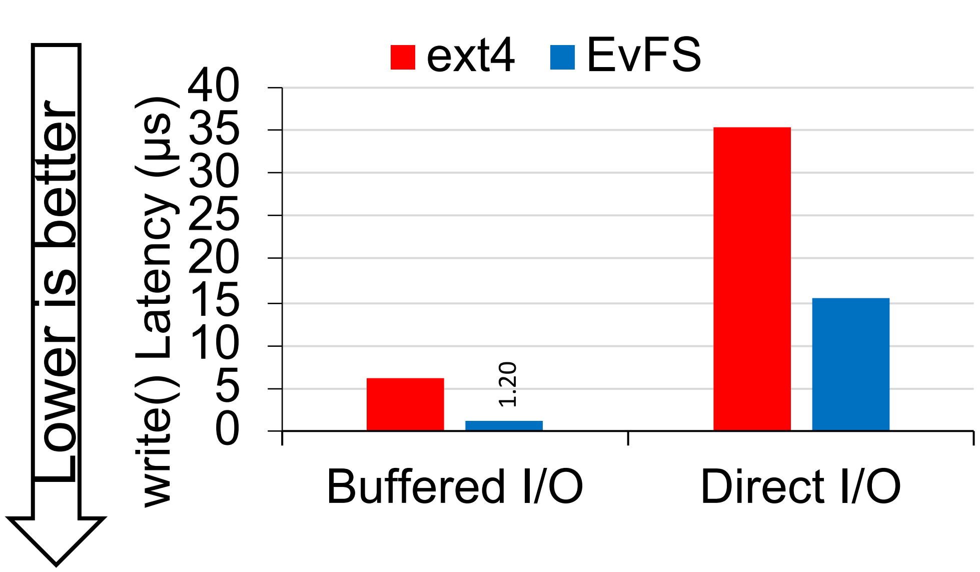


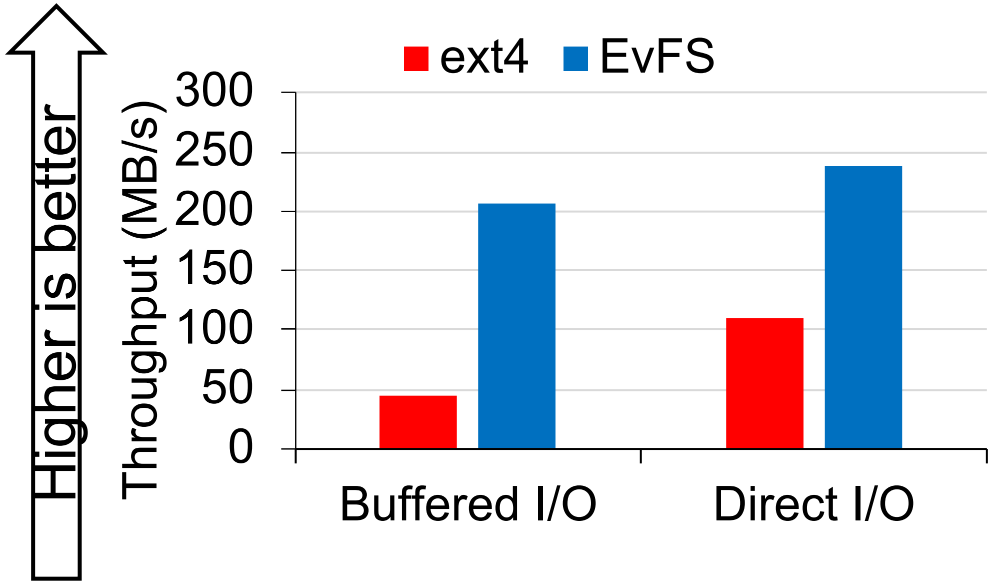


IBM Power System AC922
1.1TB RAM
160 logical Power9 cores
PCIe3 x8 6.4 TB NVMe
https://www.ibm.com/support/knowledgecenter/8335-GTH/p9hcd/fcec5e.htm
Ubuntu 18.04LTS, Linux 4.17
SPDK 19.0, DPDK 18.02
ext4: disabled journaling and readahead

Result 2/3: Blocking writes

- EvFS reduced the latency of direct I/O by 20 us
- Direct I/O showed better throughput than buffered I/O
  - Buffered I/O is measured by a pair of write() and fsync()
  - Direct I/O can accelerate apps with self-managed cache

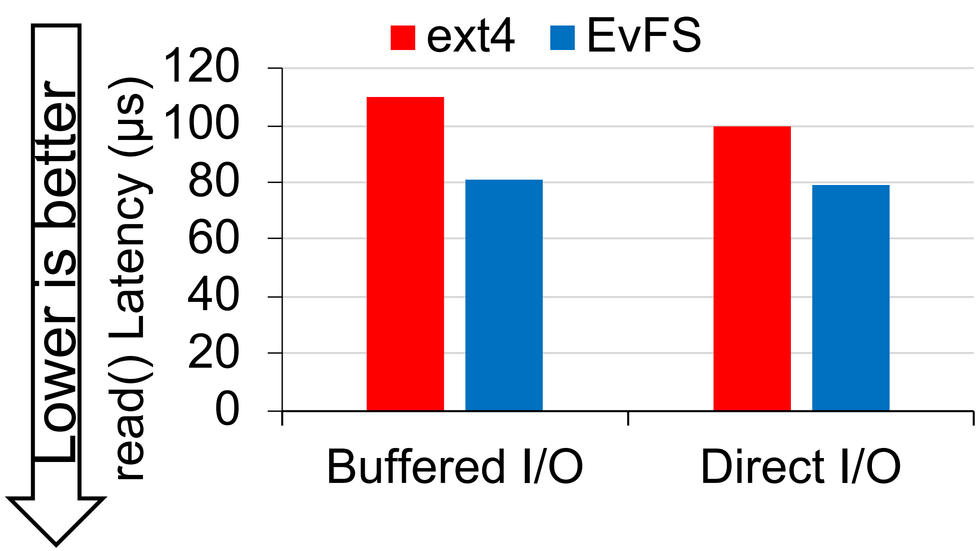


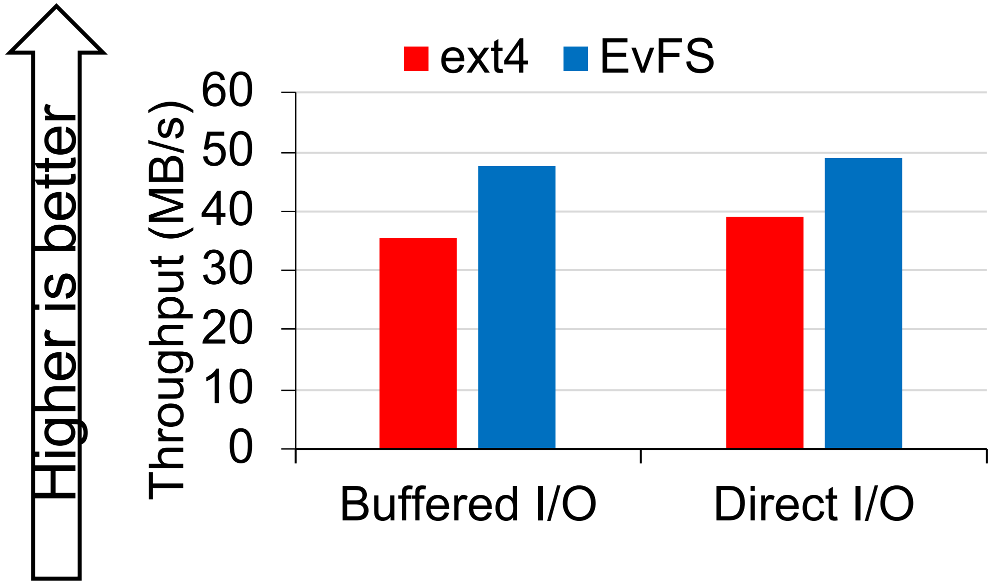


IBM Power System AC922
1.1TB RAM
160 logical Power9 cores
PCIe3 x8 6.4 TB NVMe
https://www.ibm.com/support/knowledgecenter/8335-GTH/p9hcd/fcec5e.htm
Ubuntu 18.04LTS, Linux 4.17
SPDK 19.0, DPDK 18.02
ext4: disabled journaling and readahead
Blocksize: 4 KB

Result 3/3: Blocking reads

- EvFS reduced latency for both buffered and direct I/O by 20 us





IBM Power System AC922
1.1TB RAM
160 logical Power9 cores
PCIe3 x8 6.4 TB NVMe
https://www.ibm.com/support/knowledgecenter/8335-GTH/p9hcd/fcec5e.htm
Ubuntu 18.04LTS, Linux 4.17
SPDK 19.0, DPDK 18.02
ext4: disabled journaling and readahead
Blocksize: 4 KB

Summary

- Showed early design and implementation of EvFS for NVM
  - EvFS minimizes the latency of file I/O with full user-level storage stack, event-driven architecture, and direct I/O
  - FIO showed 700 ns latency for non-blocking writes
  - EvFS reduced the latency for blocking I/O by 20 usec

- Future work:
  - Implementation of missing POSIX semantics, journaling, etc.
  - Evaluation with NVMM and other benchmarks