Programming and Usage Models for Non-volatile Memory

Priya Sehgal
MTS, Advanced Technology Group, NetApp

SDC India
26th May 2016

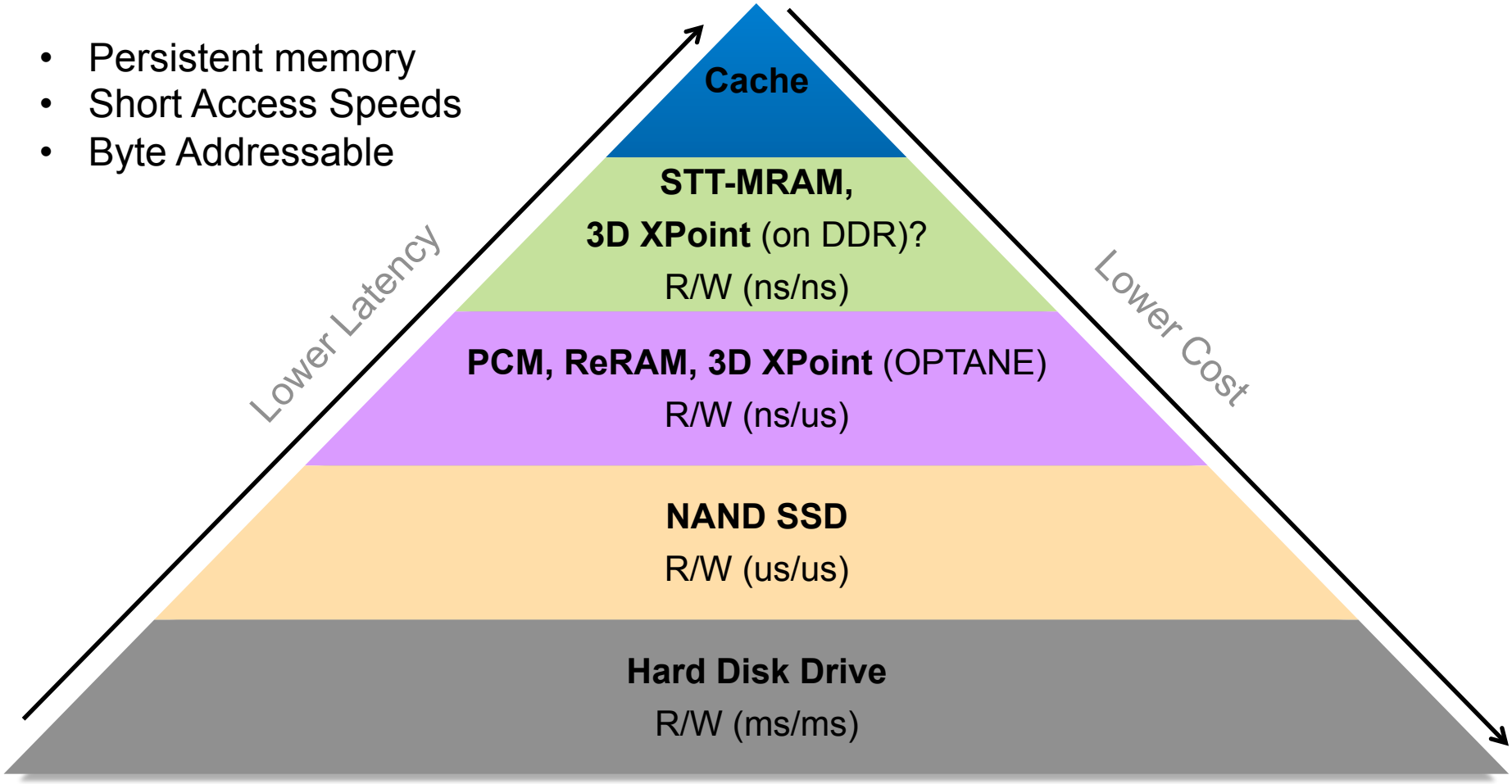
NetApp

Agenda

- Overview
- SNIA NVM Programming Model and Ecosystem
- File System Changes for NVM
  - Existing – Linux (ext4)
  - New – PMFS, BPFS
- NVM Library – libpmem example

Non-Volatile Memory Technology

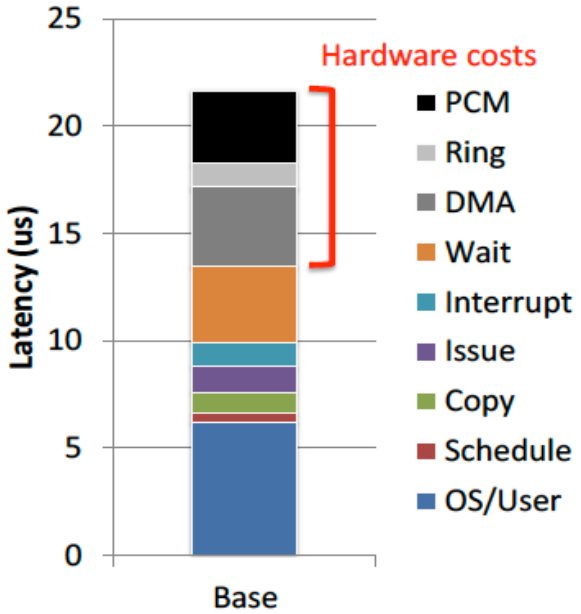
- Persistent memory
- Short Access Speeds
- Byte Addressable



NetApp

Implications to Software

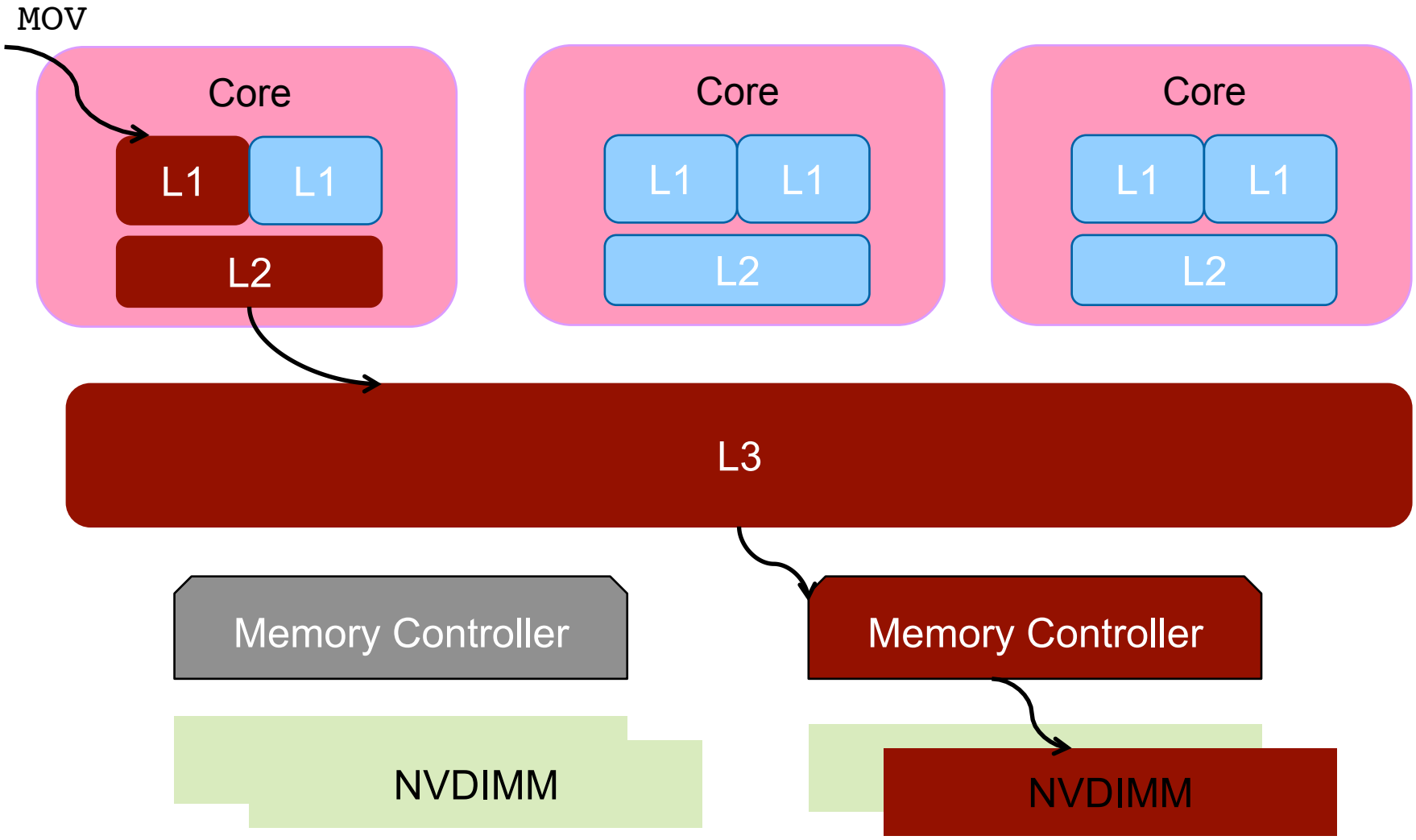
- Persistence: Volatility a Virtue!
  - Application or OS panics because of an illegal/wrong persistent memory address
  - Ensure durability
  - Ensure ordering
- Fast Access Speed
  - Software Stack overhead
- Byte Addressable
  - Block-oriented softwares
  - Can leverage Load/Store



[Caulfield, MICRO '10]

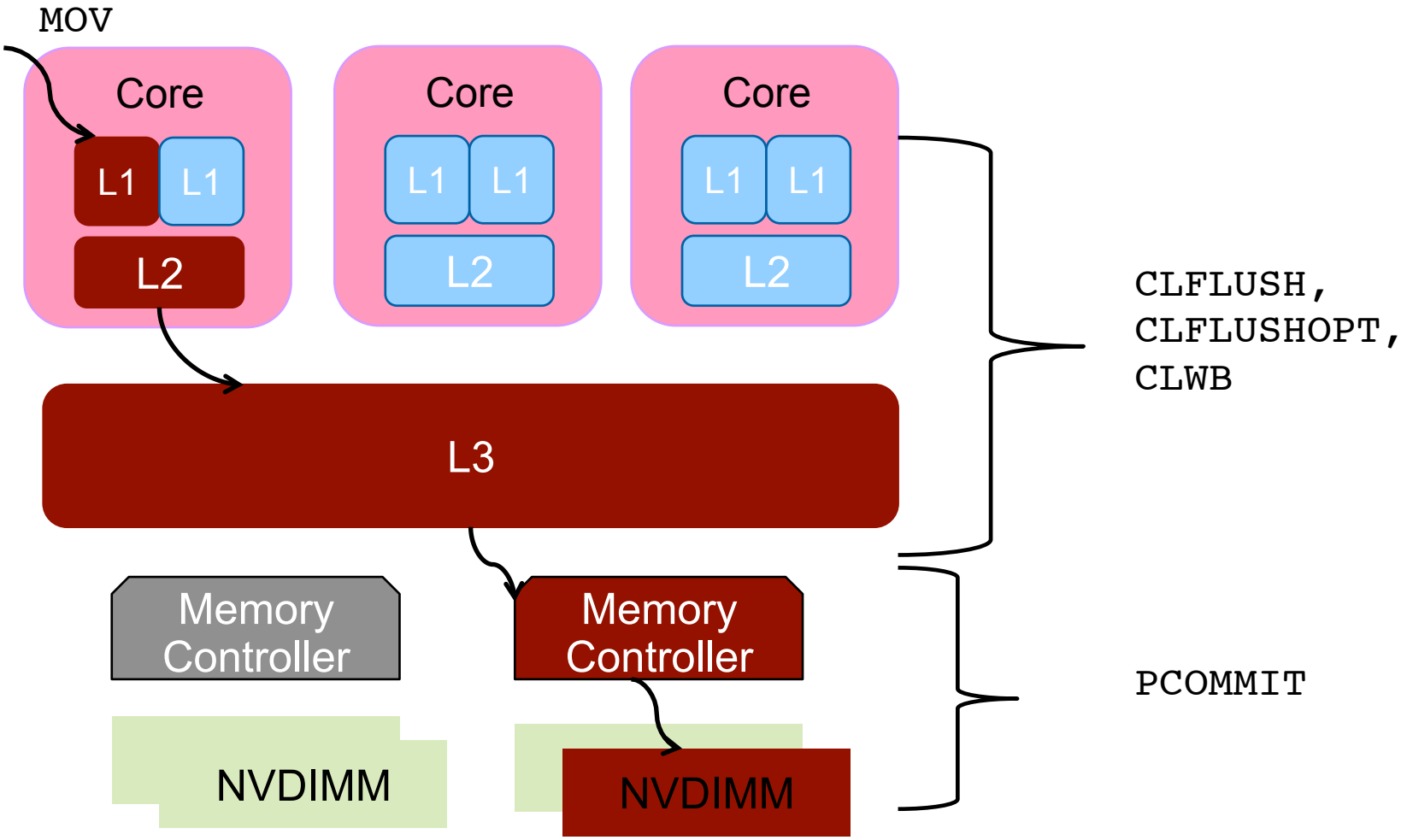
NetApp

Roadblocks to Persistence



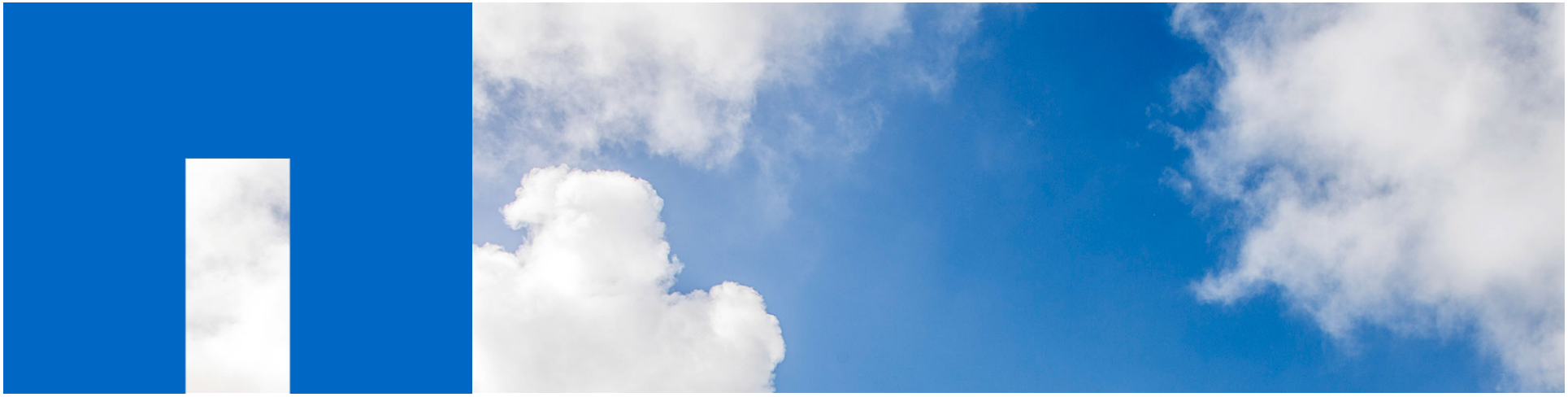
NetApp

Instruction Level Support



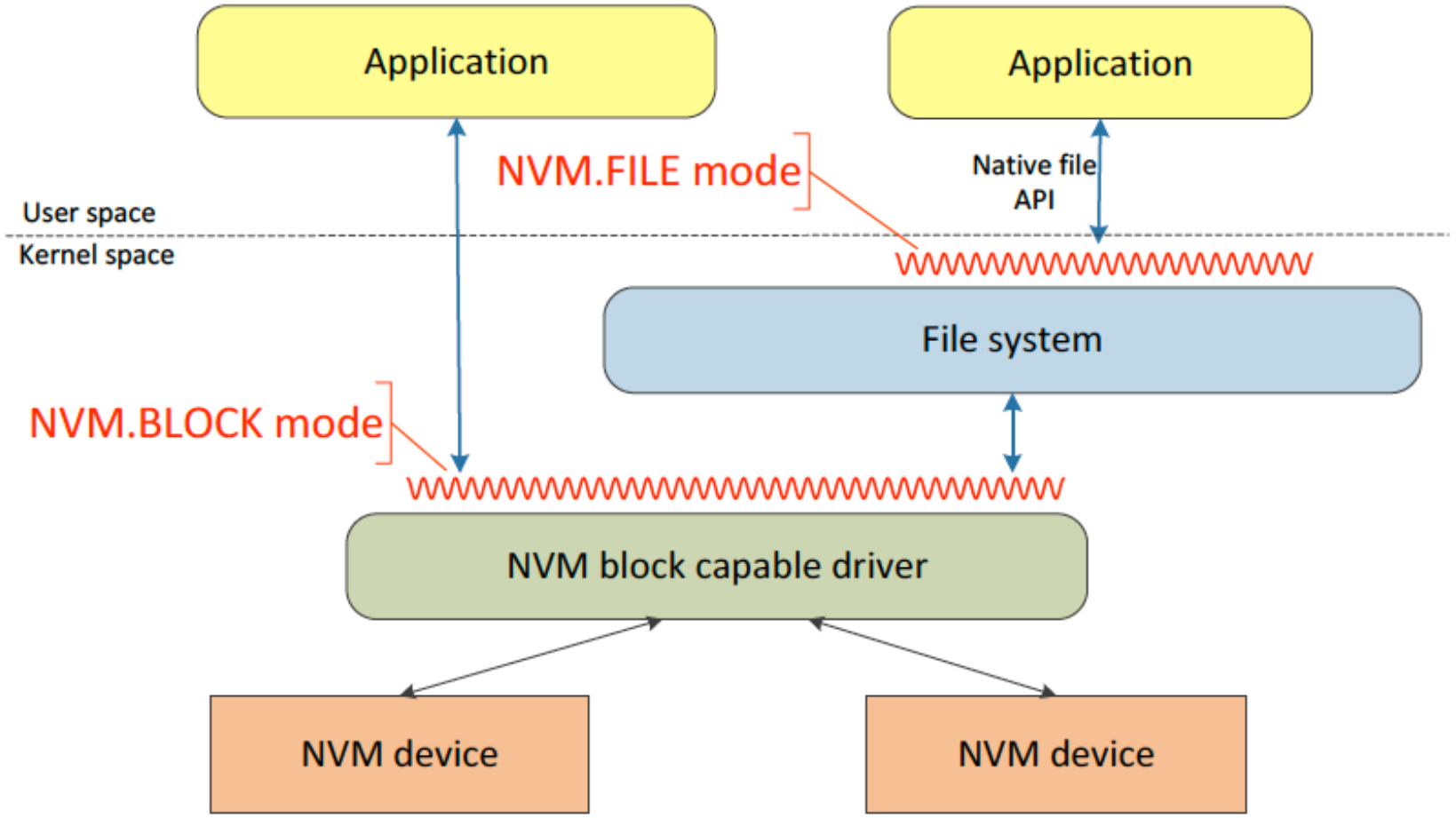
https://software.intel.com/sites/default/files/managed/b4/3a/319433-024.pdf

NetApp



NVM Prog. Ecosystem

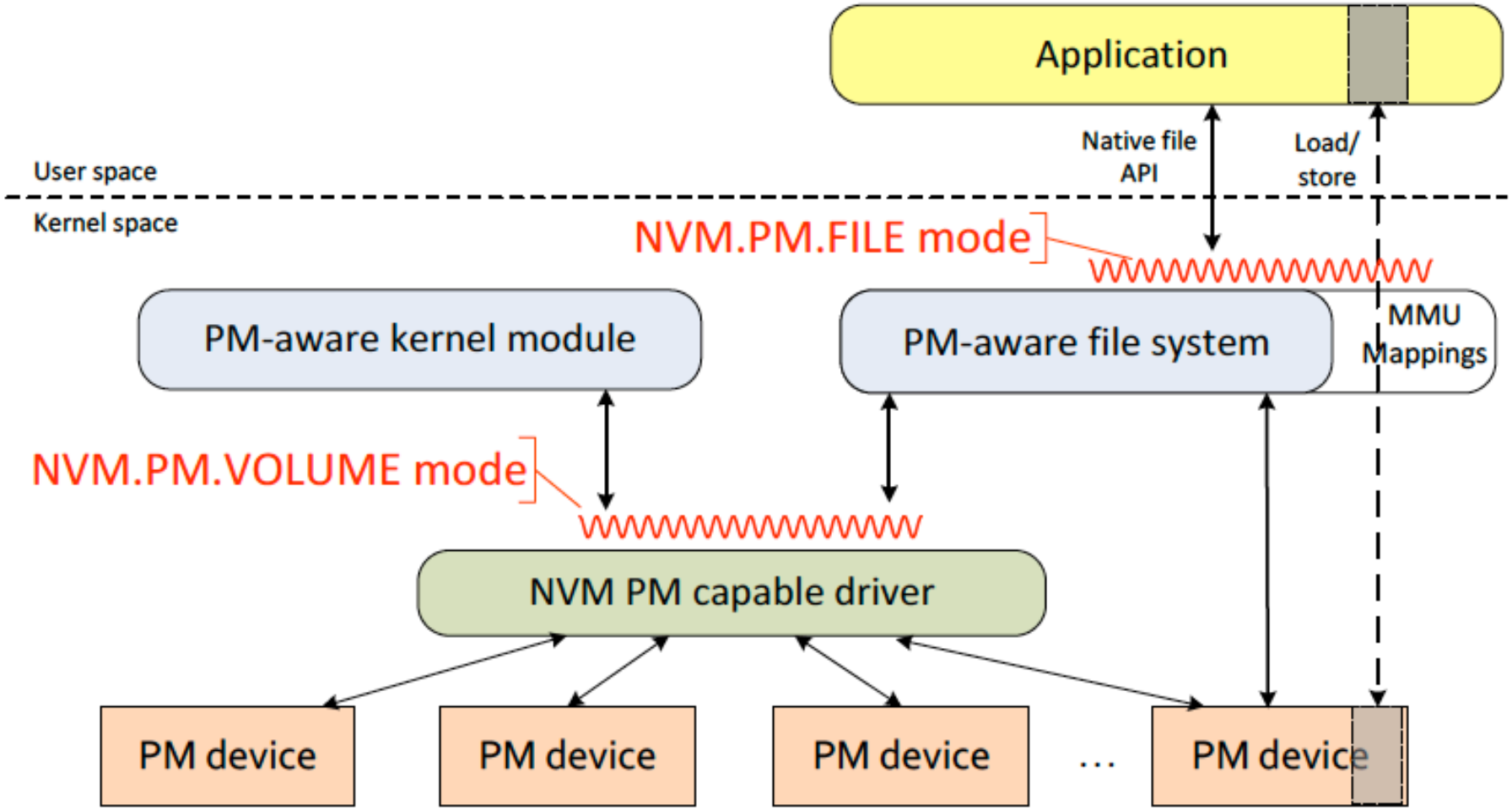
SNIA NVM Programming Model



www.snia.org/sites/default/files/NVMProgrammingModel_v1r10DRAFT_0.pdf

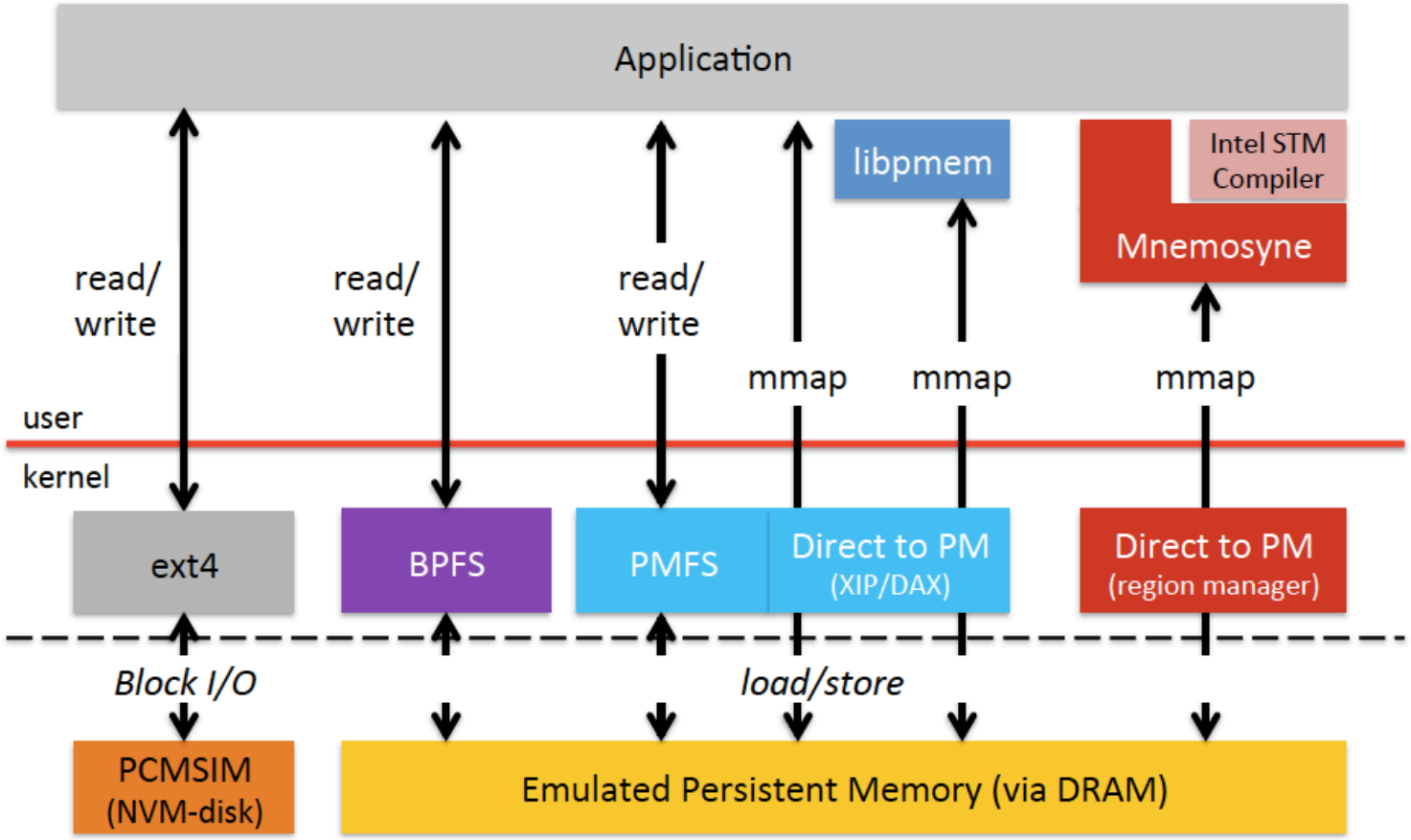
NetApp

SNIA NVM Programming Model

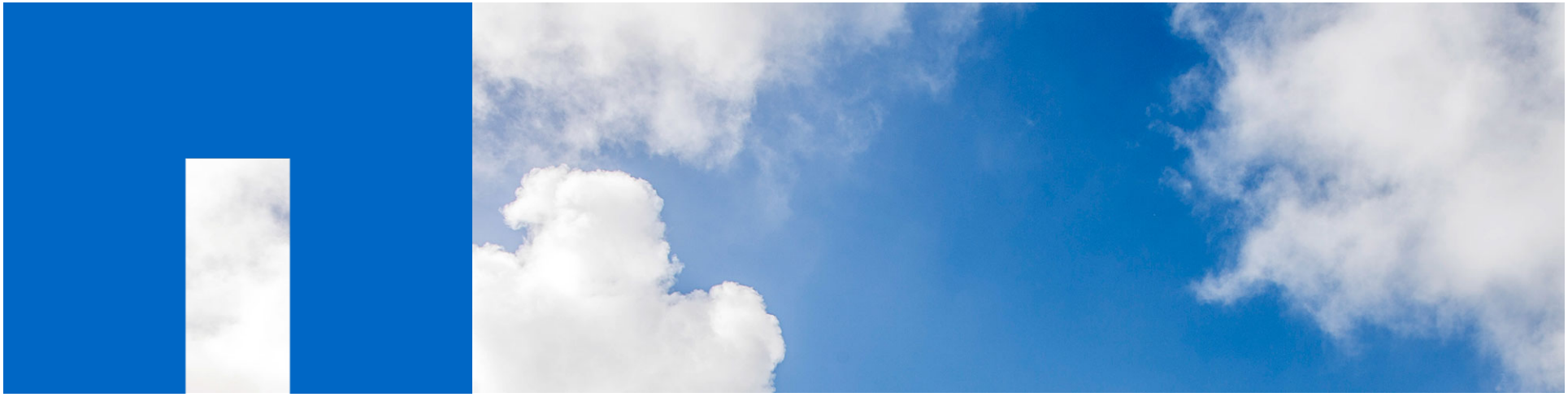


NetApp

Ecosystem Overview



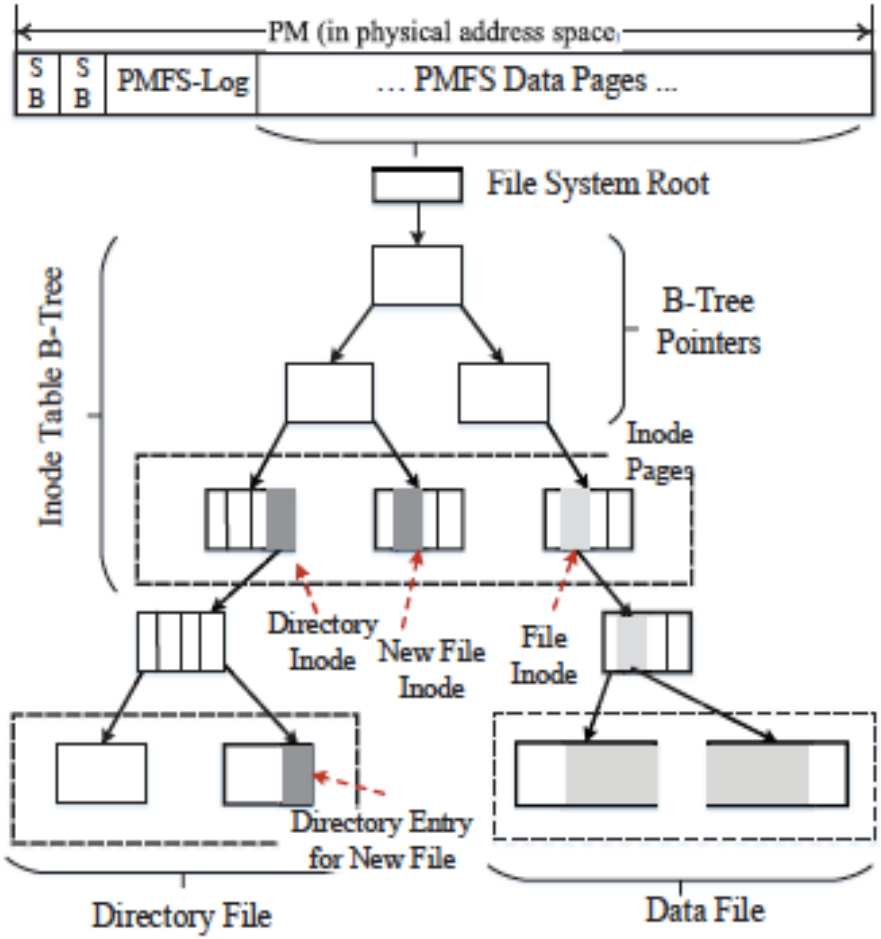
NetApp



File System Changes

Intel's PMFS

- Design Goals
  - Efficient access to PM by apps
  - Optimize for byte addressability
  - Protect from stray writes
- Contributions
  - Remove block layer and page cache - Enable direct access to PM via XIP (DAX)
  - Supports large pages
  - Atomic in-place update to metadata
  - Provides crash consistency
    - Fine grained undo logging
  - Leverage *write-protect* in kernel



http://dl.acm.org/citation.cfm?id=2592814

NetApp

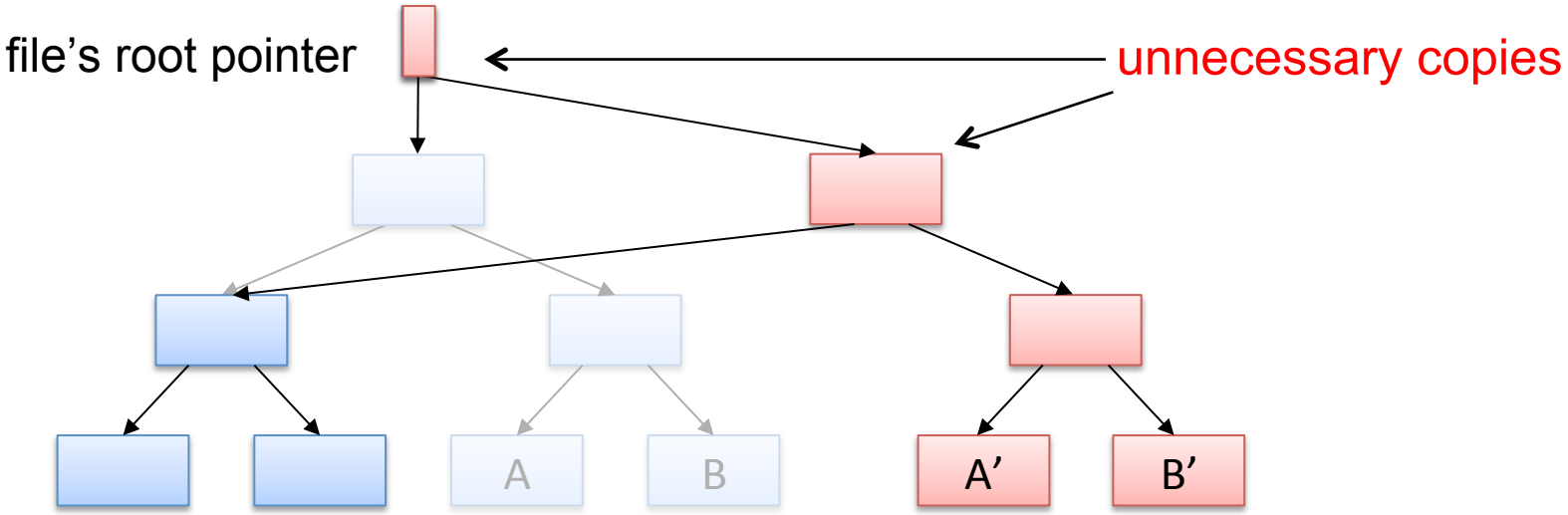
Microsoft's BPFS

- Ensures file system consistency guarantees
- Use copy-on-write up to root of file system
- Problem:
  - Any change requires bubbling to the FS root
  - Small writes result in large copy overhead
- Solution:
  - Uses Short-Circuit Shadow Paging
    - Adopt in-place and/or atomic update when possible
    - Uses byte-addressability and atomic 64b writes

http://research.microsoft.com/pubs/81175/BPFS.pdf

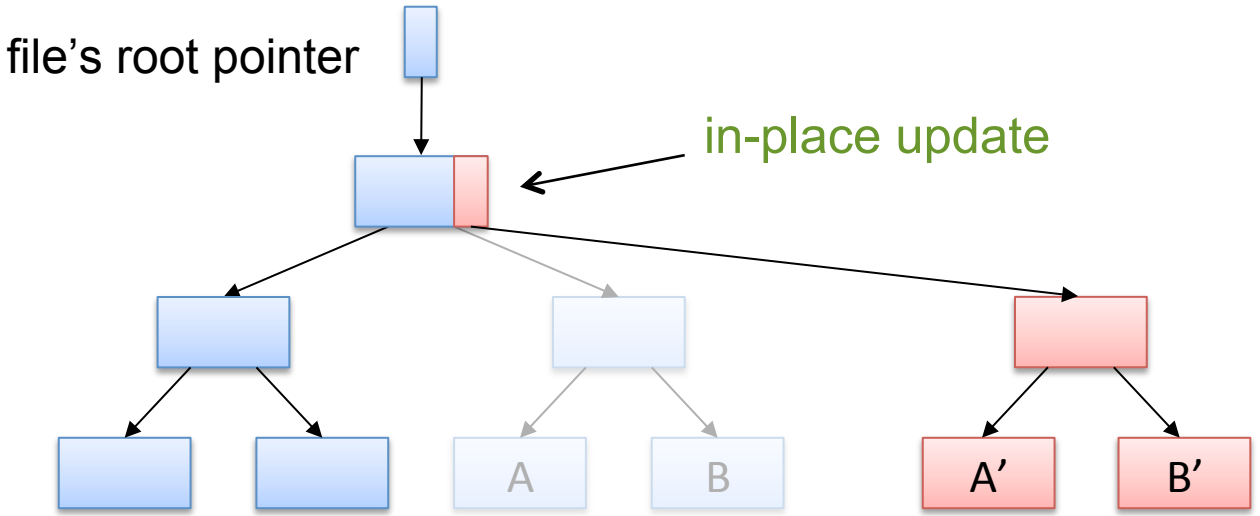
Problem with CoW (Shadow Paging)

Any small change results in large copy overhead



Short-Circuit Shadow Paging

- In-place update when possible – save on copies
- Appends committed by updating the file size

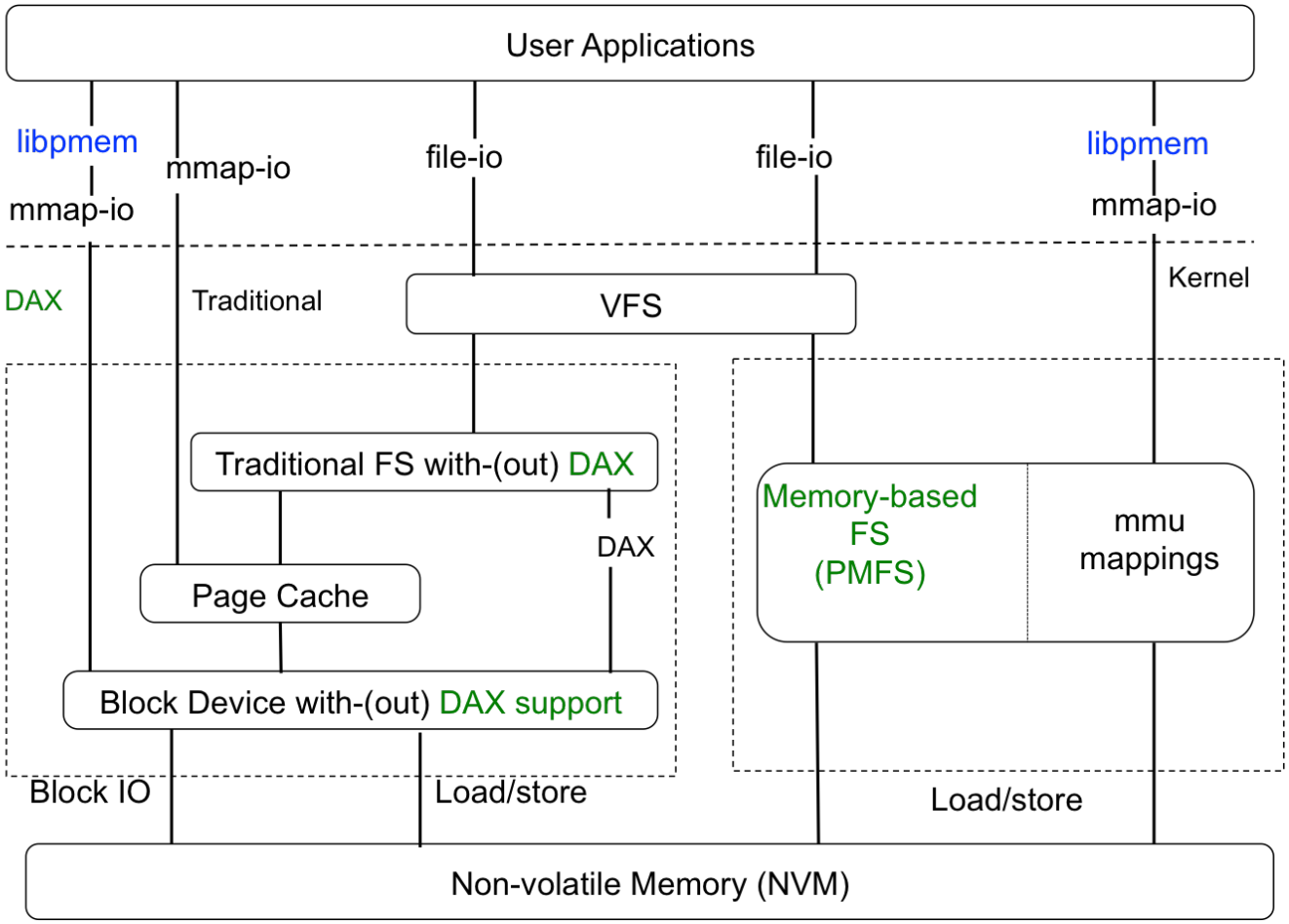


Linux File System Support

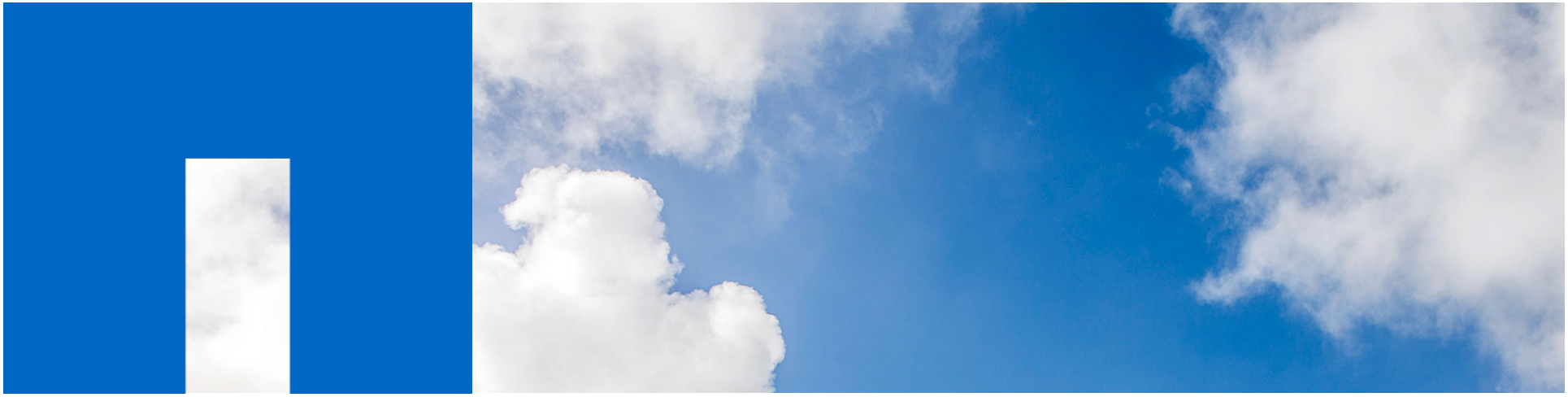
- Direct Access for Files (DAX)
  - Bypass page cache
  - Perform read/write directly to the storage device – Synchronous
- Support from block driver
  - `direct_access`
  - Examples: brd – RAM backed block device driver
- Support from File System
  - Direct_IO functions – `dax_do_io()`
  - Implementing mmap file operations for DAX files
    - Page fault functions
  - Examples: ext2, ext4

https://www.kernel.org/doc/Documentation/filesystems/dax.txt

Traditional vs. Optimized File Systems



NetApp



NVM Library

NVM Library

- library for using memory-mapped persistence, specifically for PM
- Supports SNIA NVM API
- Builds on Linux DAX
- Provides a collection of libraries:
  - libpmem: low-level persistent memory support – persistent memory instructions for flushing changes to PM.
  - libpmemlog: pmem-resident log file (uses libpmem)
  - libpmemobj: transactional object store (uses libpmem)
  - And many more

- Link: http://pmem.io/nvml/libpmem/

http://pmem.io/nvml/libpmem/

Example using libpmem

```
/* create a pmem file */
if ((fd = open("/pmem-fs/myfile", O_CREAT|O_RDWR, 0666)) < 0) {
        perror("open");
        exit(1);
}
/* allocate the pmem */
posix_fallocate(fd, 0, PMEM_LEN))

/* memory map it */
if ((pmemaddr = pmem_map(fd)) == NULL) {
        perror("pmem_map");
        exit(1);
}

/* store a string to the persistent memory */
strcpy(pmemaddr, "hello, persistent memory.");
/* flush above strcpy to persistence */
pmem_persist(pmemaddr, PMEM_LEN);

strcpy(pmemaddr, "hello again, persistent memory.");
pmem_flush(pmemaddr, PMEM_LEN);
pmem_drain();
```

Force changes to NVM

Flush processor caches

Wait for h/w buffers to drain

Summary

- Scratching the surface → more research going on areas such as remote memory access, security, etc.
- Existing software (without changes) on NVM would result in:
  - Sub-optimal performance
  - Consistency/Durability issues
- Usage Models
  - NVM Block Device w/ existing/modified file systems or direct access
  - NVM File Systems on NVM Driver
  - NVM Library (talks to NVM Filesystem) – libpmem
  - Persistent Memory Regions and Persistent heap
    - Manage using memory management unit
    - APIs: pmap, punmap, pmalloc, pfree
    - E.g., Mnemosyne, NV-Heaps

Thank you.

References

BPFS: http://research.microsoft.com/pubs/81175/BPFS.pdf

[Caulfield, MICRO '10]: Moneta: A High-performance Storage Array Architecture for Next-generation, Non-volatile Memories, Adrian M. Caulfield, Arup De, Joel Coburn, Todor I. Mollov, Rajesh K. Gupta, and Steven Swanson, MICRO 43

Intel Manual: https://software.intel.com/sites/default/files/managed/b4/3a/319433-024.pdf

[libpmem]: http://pmem.io/nvml/libpmem/

Linux DAX: https://www.kernel.org/doc/Documentation/filesystems/dax.txt

Mnemosyne: http://research.cs.wisc.edu/sonar/papers/mnemosyne-asplos2011.pdf

NV-Heaps: http://www.msr-waypoint.com/pubs/198372/Asplos2011_NVHeaps.pdf

PMFS: http://dl.acm.org/citation.cfm?id=2592814

SNIA NVM Programming TWG: www.**snia**.org/sites/default/files/**NVM**ProgrammingModel_v1r10**DRAFT**_0.pdf