The NVM Revolution

Paul von Behren/Intel

SNIA Legal Notice

- The material contained in this tutorial is copyrighted by the SNIA unless otherwise noted.
- Member companies and individual members may use this material in presentations and literature under the following conditions:
  - Any slide or slides used must be reproduced in their entirety without modification
  - The SNIA must be acknowledged as the source of any material used in the body of any document containing material from these presentations.
- This presentation is a project of the SNIA Education Committee.
- Neither the author nor the presenter is an attorney and nothing in this presentation is intended to be, or should be construed as legal advice or an opinion of counsel. If you need legal advice or a legal opinion please contact your attorney.
- The information presented herein represents the author's personal opinion and current understanding of the relevant issues involved. The author, the presenter, and the SNIA do not assume any responsibility or liability for damages arising out of any reliance on or use of this information.

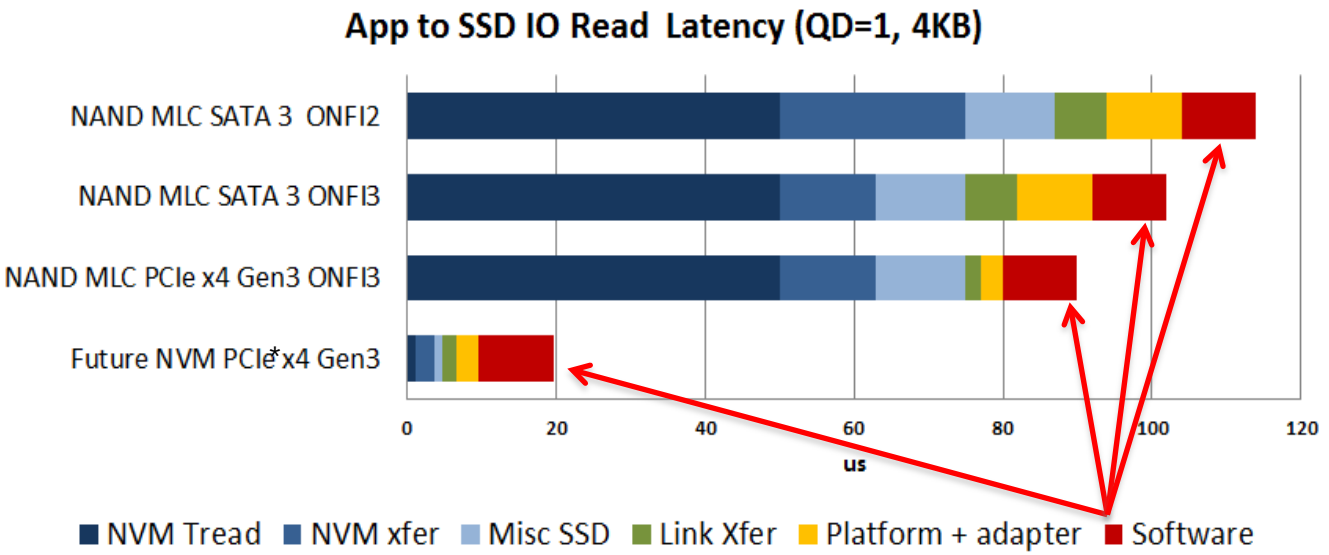
NO WARRANTIES, EXPRESS OR IMPLIED. USE AT YOUR OWN RISK.

Abstract

The NVM Revolution

- This presentation provides an introduction to the current activities leading to software architectures and methodologies for new Non-Volatile Memory (NVM) technologies, including the activities of the SNIA NVM Programming TWG (Technical Working Group). This session includes a review and discussion of the impacts of the SNIA *NVM Programming Model* (NPM). We will preview the current work on new technologies, including remote access, high availability, clustering, atomic transactions, error management, and current methodologies for dealing with NVM.

Need for A New Model



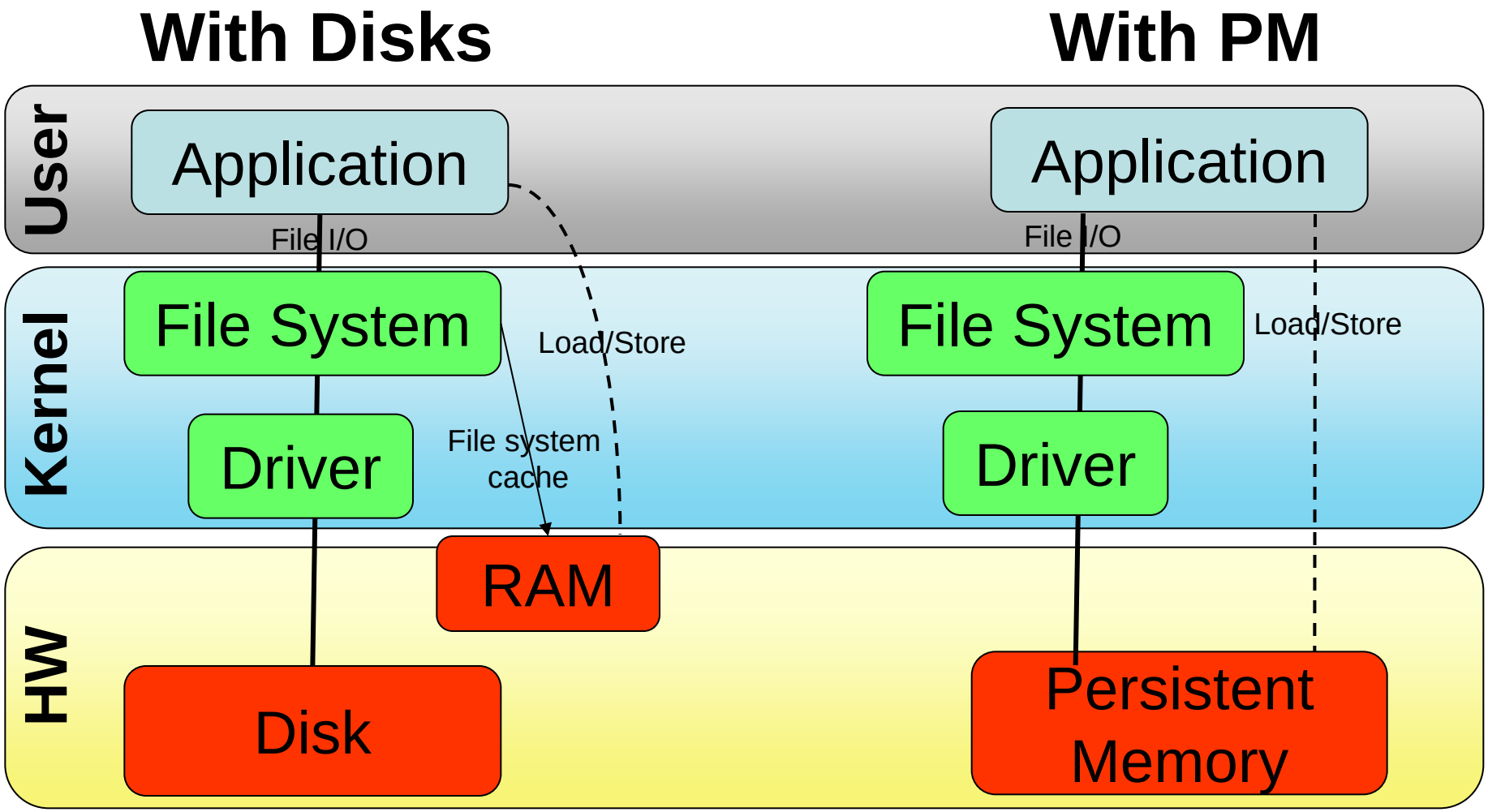
- With Next Generation NVM, hardware is no longer the bottleneck
  - Need optimized platform storage interconnect
  - Need optimized software storage access methods

NVM Programming Model specification organization

- Disk-like non-volatile memory
  - Appears as disk drives to applications
  - Accessed as traditional array of blocks
- Memory-like non-volatile memory
  - Appears as memory to applications
  - Applications store data directly in byte-addressable memory
  - No IO or even DMA is required

"Persistent memory" refers to Memory-like non-volatile memory

Memory Mapped File Programming Model



SNIA *NVM Programming Model*

- Version 1.1 approved by SNIA in March 2015
  - http://www.snia.org/tech_activities/standards/curr_standards/npm
- Expose new block and file features to applications
  - Atomicity capability and granularity
  - Thin provisioning management
- Use of memory mapped files for persistent memory
  - Existing abstraction that can act as a bridge
  - Limits the scope of application re-invention
  - Open source implementations available
- Programming Model, not API
  - Described in terms of attributes, actions and use cases
  - Implementations map actions and attributes to API's

The Four Modes

Block Mode Innovation

- Atomics
- NVM-oriented operations

Emerging NVM Technologies

- Failure-atomicity
- Error handling

	Traditional / Flash	Persistent Memory
User View	NVM.FILE mode	NVM.PM.FILE mode
Kernel Protected	NVM.BLOCK mode	NVM.PM.VOLUME mode
Media Type	Disk Drive	Persistent Memory
NVDIMM	Disk-Like	Memory-Like

Conventional Block and File Modes

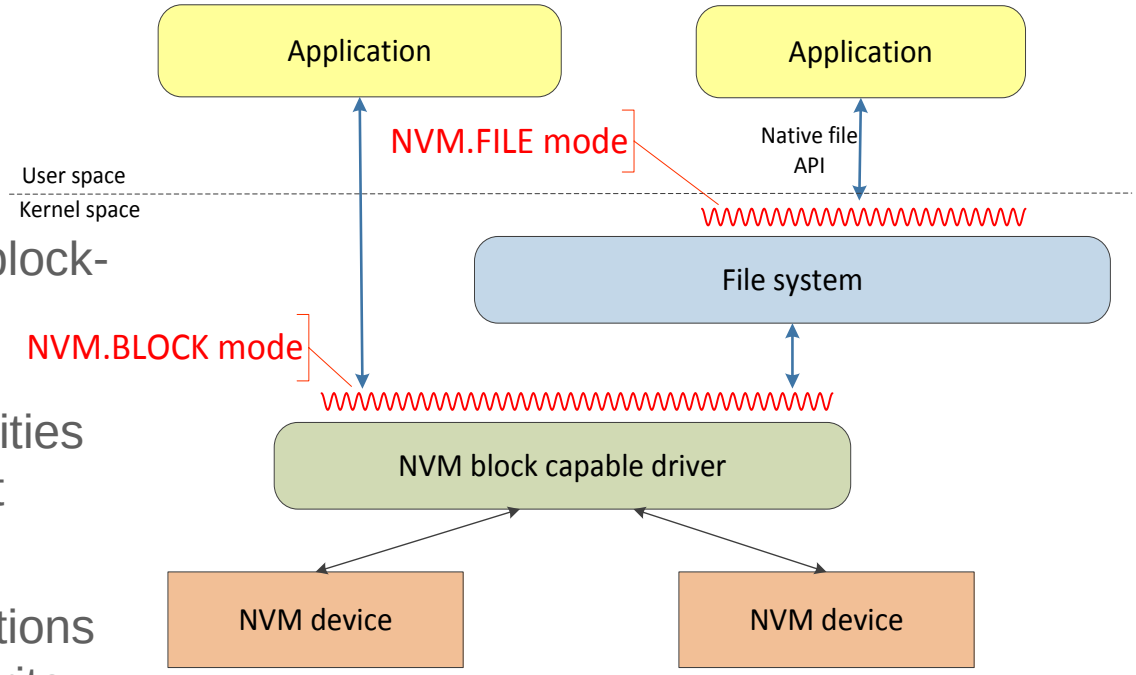
Use with disk-like NVM

NVM.BLOCK Mode

- Targeted for file systems and block-aware applications
- Atomic block writes
- Length and alignment granularities
- Thin provisioning management

NVM.FILE Mode

- Targeted for file based applications
- Discovery and use of atomic write features
- Discovery of granularities



Persistent Memory Modes

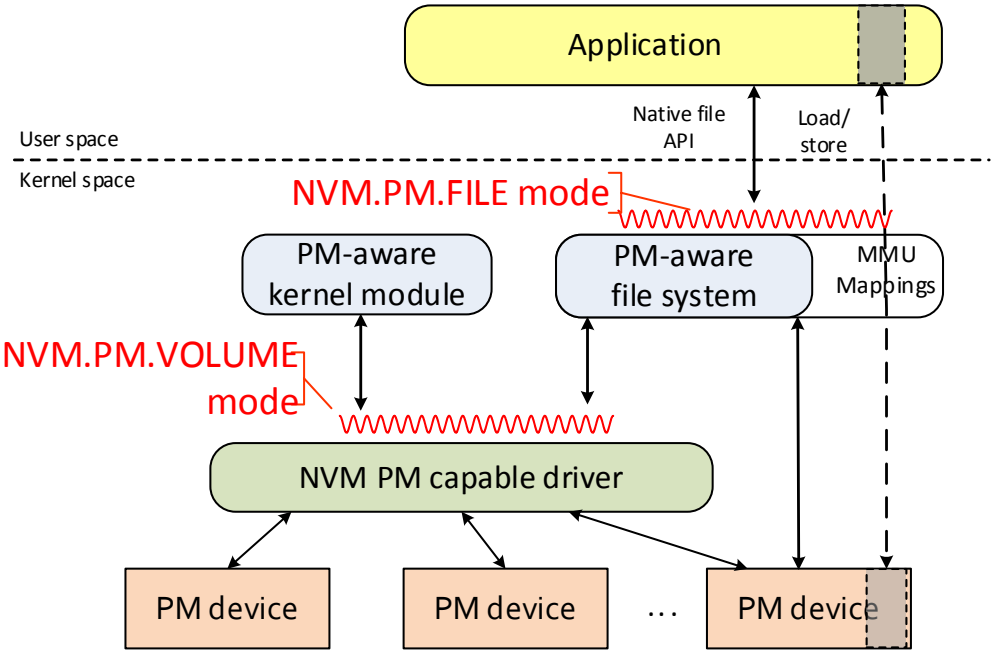
Use with memory-like NVM

NVM.PM.VOLUME Mode

- Software abstraction to OS components for Persistent Memory (PM) hardware
- List of physical address ranges for each PM volume
- Thin provisioning management

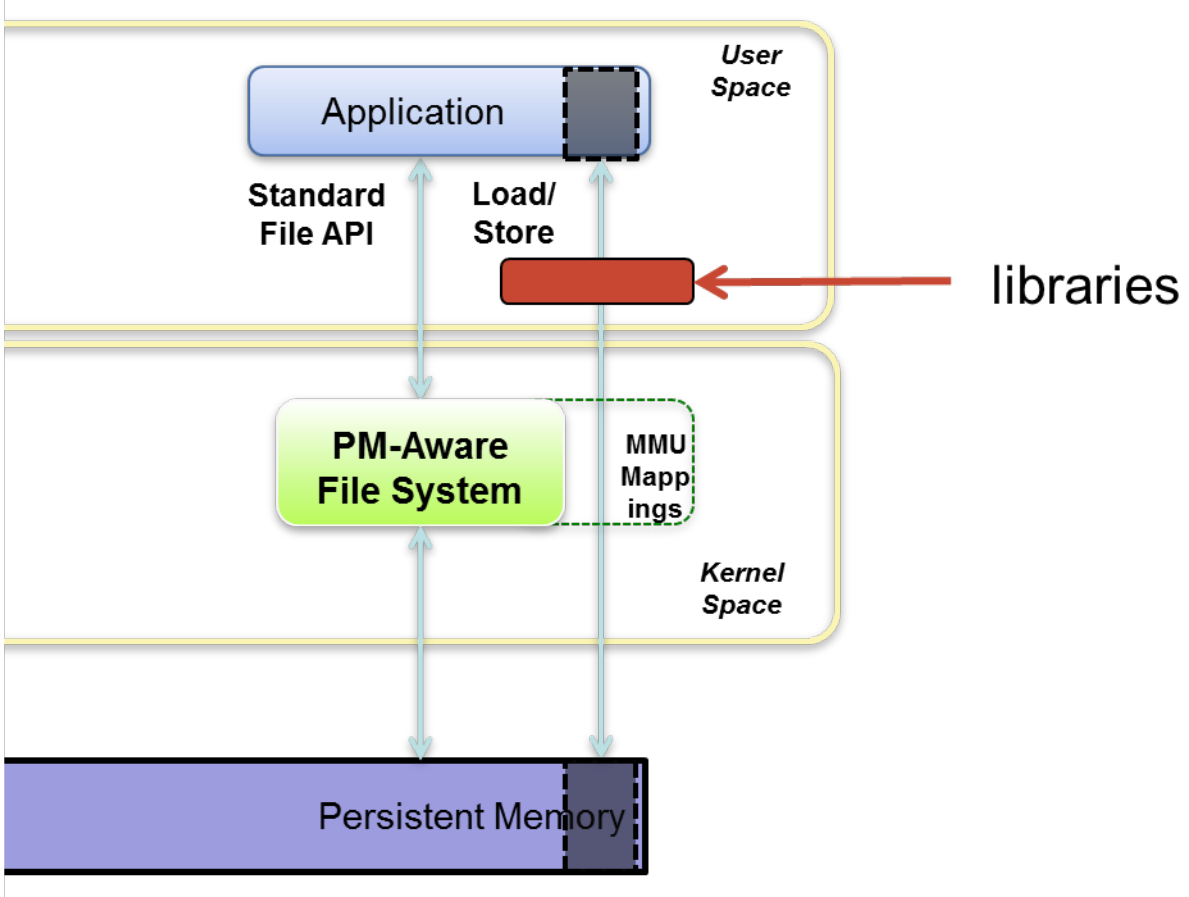
NVM.PM.FILE Mode

- Describes the behavior for applications accessing persistent memory Discovery and use of atomic write features
- Mapping PM files (or subsets of files) to virtual memory addresses
- Syncing portions of PM files to the persistence domain



Building on the Basic PM Model

- NVM.PM.FILE programming model "surfaces" PM to application
- Considering refinements to model with additional libraries
- Some may evolve into language extensions



NVM Programming TWG Current Work

- *NVM PM Remote Access for High Availability* white paper
  - Disaggregated memory
  - RDMA direct to NVM
  - High availability, clustering, capacity expansion use cases
- *Persistent Memory Atomics and Transactions* white paper
  - Considerations for software implementing atomicity and transactions with PM
  - Builds on the SNIA programming model
- Programming model improvements related to error handling
- PM security issues & hardware encryption

- More information about the NVM Programming TWG
  - http://www.snia.org/forums/sssi/nvmp

NVM Remote Access

NVM Programming Remote Access

- **Use case: mirroring data to remote PM**
  - RDMA read and write access to remote persistent memory
    - High availability memory mapped files
    - Built on NVM.PM.FILE from NVM Programming Model
- **Requirements:**
  - Assurance of remote durability
  - Efficient byte range access (e.g., scatter-gather RDMA)
  - Efficient large transfers
  - Atomicity of fundamental data types
  - Resource recovery and hardware fencing after failure

NVM Memory Access Hardware Taxonomy

- **Various topologies examined**
  - Local persistent memory
    - PM in the same servers as the accessing processor
  - Disaggregated persistent memory
    - PM is not contained within the server
    - Accessed at memory speed
  - Network Persistent Memory
    - PM accessed through a high speed network
  - Virtual shared persistent memory
    - Emulating cache coherent shared memory across networked memory using software

Recoverability

- High Durability
  - Data will not be lost regardless of failures
  - May be limited due to implementation of system
- High Availability
  - Data will remain accessible to hosts regardless of failures
  - May be limited due to implementation of system

The distinction between high durability and high availability makes it clear that high availability requires networked access to persistent memory. The network plays an important "fault isolation" role for high availability.

Consistency and Recovery

- Consistency Points
  - All data items recovered must have correct values relative to each other from the application point of view
  - Software uses hardware and software techniques to guarantee that a failure or restart results in a consistent state
- Crash Consistency
  - Applications must be prepared to recover from *any* state of the writes that were in flight when a failure occurs
  - Recovery from a crash consistent image is the same as a cold restart after a system crash

Crash consistency is a complex approach to recovery from an application standpoint. It forces considerable overhead to precisely communicate every sync action to networked persistent memory.

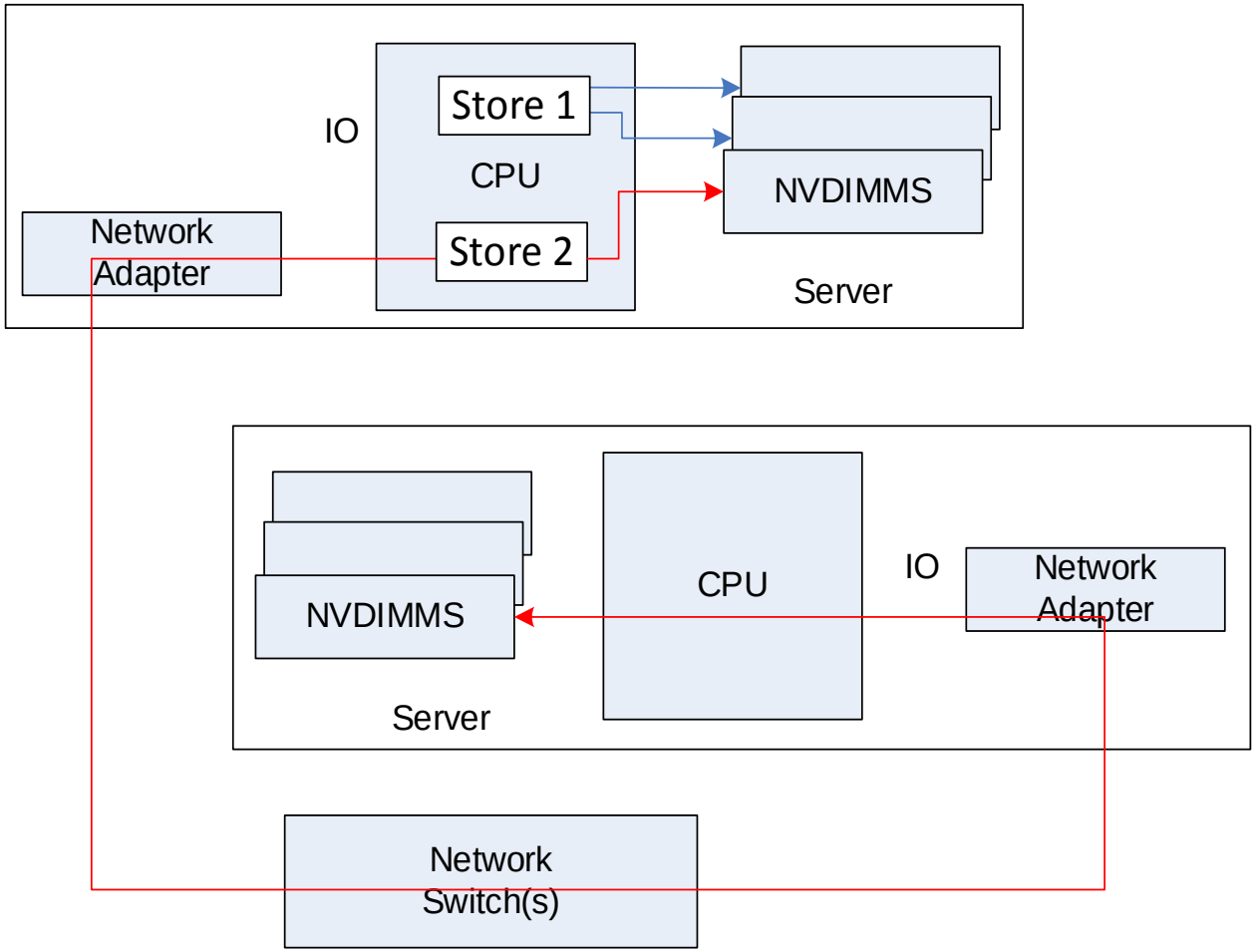
NVM Programming Model Approach to Recoverability

- Atomicity and Atomic Granularity
  - Addresses crash consistency issues
- Optimized NVM Flush
  - Addresses consistency point issues
- Data and Access Recovery Scenarios
  - In-line recovery
  - Backtracking recovery
  - Local application restart
  - Application failover

Remote Access Hardware

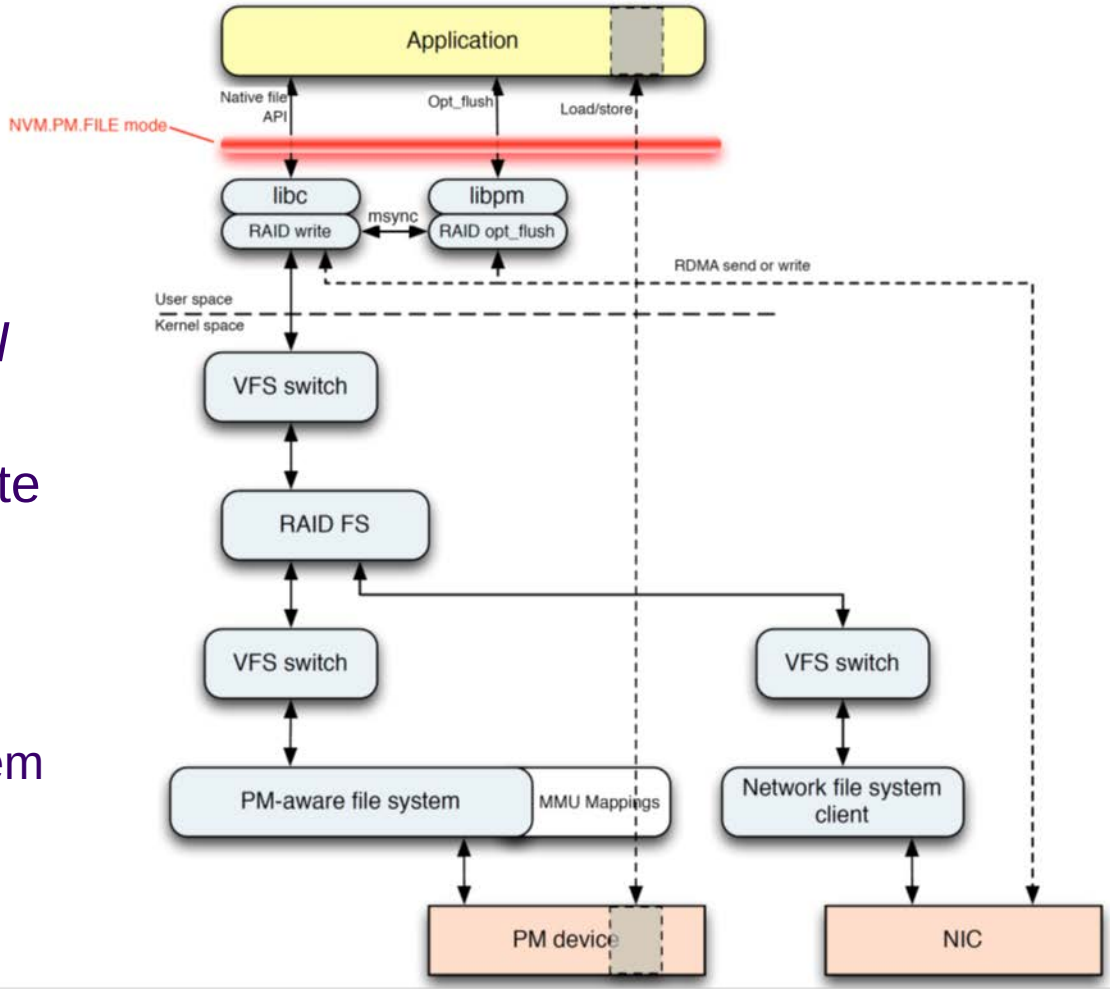
For *high availability,* local data redundancy across NVDIMMs will suffice. (blue lines)

To additionally provide *high availability*, remote data redundancy across servers or external storage nodes is required. (red lines)



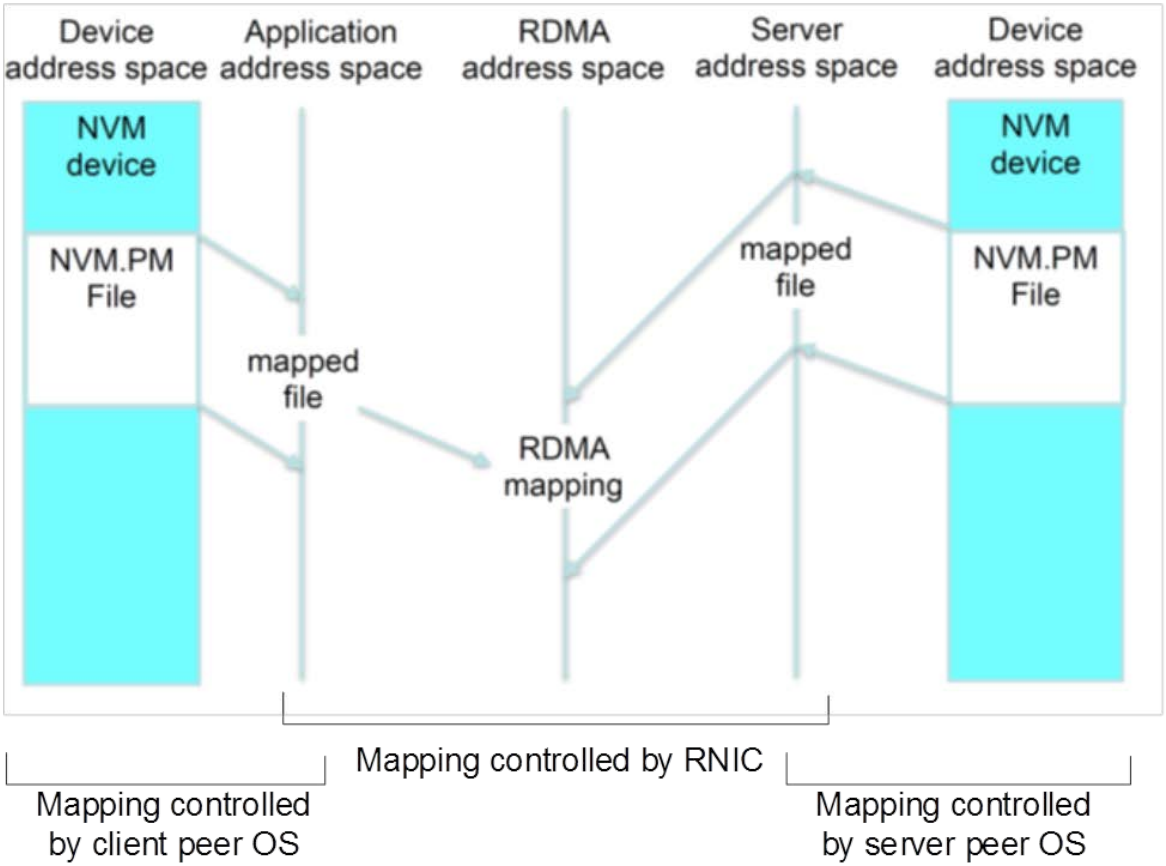
Software Context Example

- Standard file API
- *NVM Programming Model* optimized flush
- Uses replications to remote system to achieve HA
  - local file system
  - remote file system
    - via network file system client and NIC



RDMA for HA

- Examine durability, performance, and address space issues.
- Only the “Device” address spaces must match
  - Sufficient to allow restoration and failover
  - Orchestrated by peer file/operating systems



NVM RDMA Security

- Address security issues for NVM RDMA
  - Data at rest
  - Data in flight
  - Authentication
  - Authorization
- Threat models
- Transport security
- RDMA security model
  - In the context of NVM RDMA
  - Performance issues with existing models

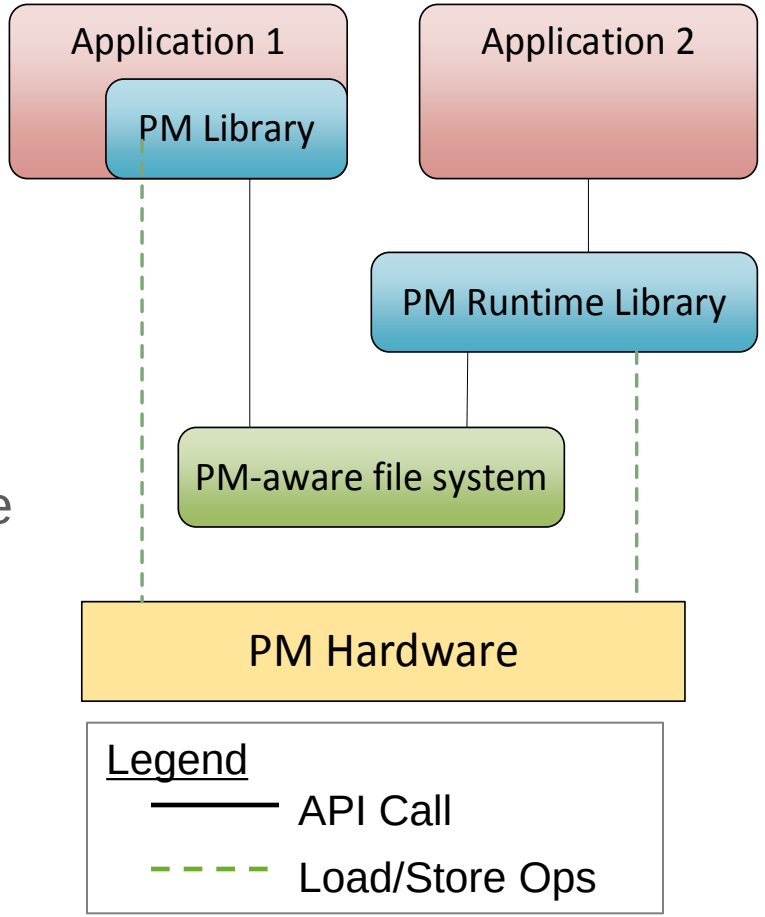
Persistent Memory Atomics and Transactions

Work-In-Progress White Paper

- Discusses PM-specific issues for software implementing atomic operations or transaction managers
  - Atomic operations provide atomicity as part of a specific task
    - For example, append to a log – and assure no partial updates
  - Transaction managers allow caller to specify a list of data ranges that are updated atomically
- Examples of PM-specific issues
  - Volatile memory is implicitly freed on a program failure; software needs to assure persistent memory doesn't "leak"
  - For optimal performance, software should consider hardware granularities – like machine architecture cache line size
  - Utilize hardware atomic operations, but software techniques needed for larger, disjoint memory ranges

Relationship with the SNIA *NVM Programming Model*

- Application 1
  - Links with a PM-aware library
  - Uses a standard compiler
- Application 2
  - Uses a PM-aware compiler
  - Compiled code invokes a run-time library



Goals for PM-aware libraries

- Provide for failure-atomic store operations to PM
  - Including the event of system or power failure
- Provide for failure-atomic operations for large address ranges
  - And groups of ranges
- Work with standard compilers
  - Standard language support for PM is a goal, but we want to consider approach that can be used now
- Work with existing CPUs
  - Atomicity and transactions rely on existing or near term processor capabilities
- Ability to avoid unnecessary/redundant instrumentation
  - Allow applications to selectively control flushing, avoid overhead of atomicity and transactions where not needed

White paper topics

- *Persistent Memory Atomics and Transactions* paper topics include
  - Examine current methodologies
    - Transactions and atomic operations optimized for PM
    - Software Transactional Memory
    - Avoiding latencies with “compiler hides concurrency” approaches
    - Relationship to C11 and C11++
    - PM-aware data structures
  - Consider PM aspects of use cases
    - Append to a file atomically
    - Transactional update to multiple records in key/value database

Related Open Source Implementations

- Linux DAX Extensions - PM-aware file system (NVM.PM.FILE)
  - Support ext4 on NV-DIMMs
  - http://lwn.net/Articles/588218/
  - DAX changes accepted in Linux kernel 4.0
  - Support for NVDIMM detection from BIOS (e820) in kernel 4.2
- PM transactional libraries
  - NVML: http://pmem.io/nvml/
  - NVM-Direct: https://github.com/oracle/NVM-Direct

The SNIA NVM Programming TWG

- TWG goals:
  - Accelerating availability of software enabling NVM
  - Specifications and white papers
- To Join the TWG
  - 1) To get a SNIA login, go here: https://members.snia.org/kmembership_info/person_signup
  - 2) To join the TWG, go here: https://members.snia.org/apps/org/workgroup/nvmptwg/ click on the “join group” link (under Documents) and fill out the form.
- Weekly calls, bi-monthly face-to-face meetings
  - No additional cost to SNIA members

Related Tutorials

NVDIMM Cookbook – A Guide to NVDIMM Integration

Attribution & Feedback

The SNIA Education Committee thanks the following Individuals for their contributions to this Tutorial.

Authorship History

Walt Hubis, Hubis Technical Associates, Vice Chair, SNIA SSSI & Slide Author, DSI Presenter

Mark Carlson, Toshiba

Paul von Behren, Intel, Chair SNIA NVM Programming TWG

Additional Contributors

Doug Voigt, HP, Chair SNIA NVM Programming TWG

Jim Ryan, Intel

Steve Byan, NetApp

*Please send any questions or comments regarding this SNIA Tutorial to **tracktutorials@snia.org***