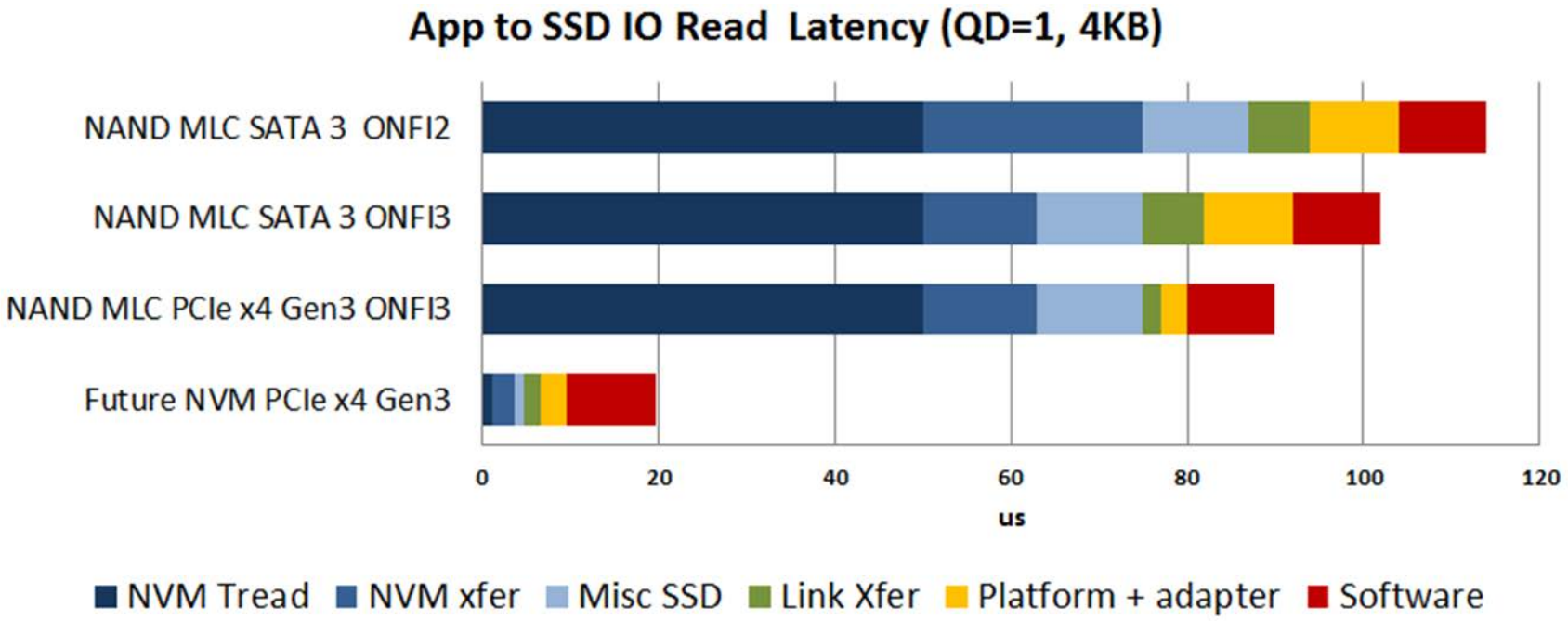
The SNIA NVM Programming TWG and the NVM Revolution

Walt Hubis
Vice Chair, Solid States Storage Initiative
Hubis Technical Associates
April 7, 2015

Need for A New Model



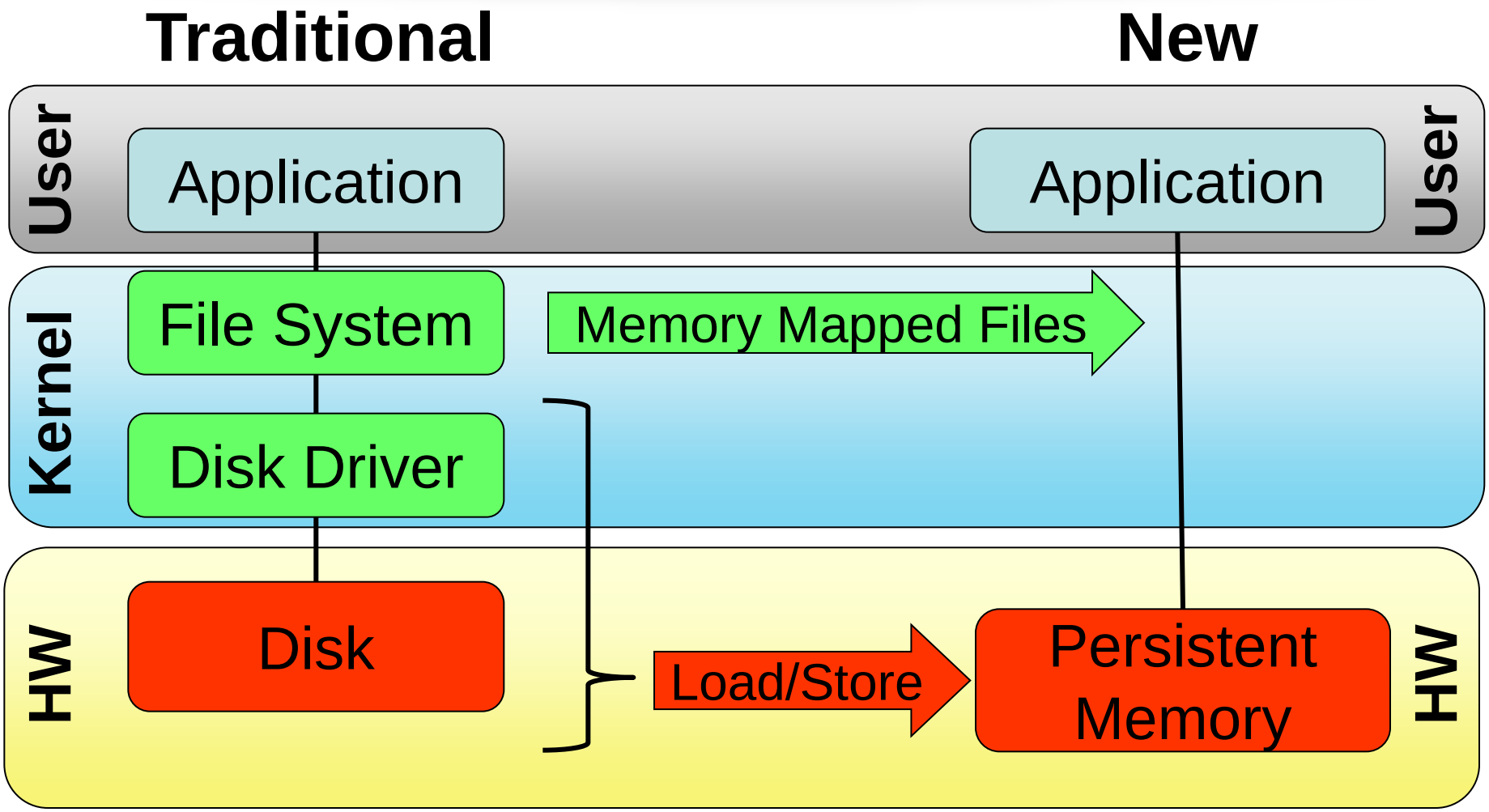
- With Next Generation NVM, the NVM is no longer the bottleneck
  - Need optimized platform storage interconnect
  - Need optimized software storage access methods

Application Access to Non-Volatile Memory

- Disk-like non-volatile memory
  - Appear as disk drives to applications
  - Accessed using disk stack
- Memory-like non-volatile memory
  - Appear as memory to applications
  - Applications store variables directly in RAM
  - No IO or even DMA is required

Memory-like non volatile memory is a type of persistent memory

Eliminate File System Latency with Memory Mapped Files



SNIA NVM Programming Model

- Version 1.1 approved by SNIA in March 2015
  - Downloadable by anyone
  - Version 1.0 approved by SNIA in December 2013
- 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 for incremental innovation (e.g. Linux DAX extensions)
- 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
- Access hints
- NVM-oriented operations

Emerging NVM Technologies

- Performance
- Performance
- Performance, cost

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

Conventional Block and File Modes

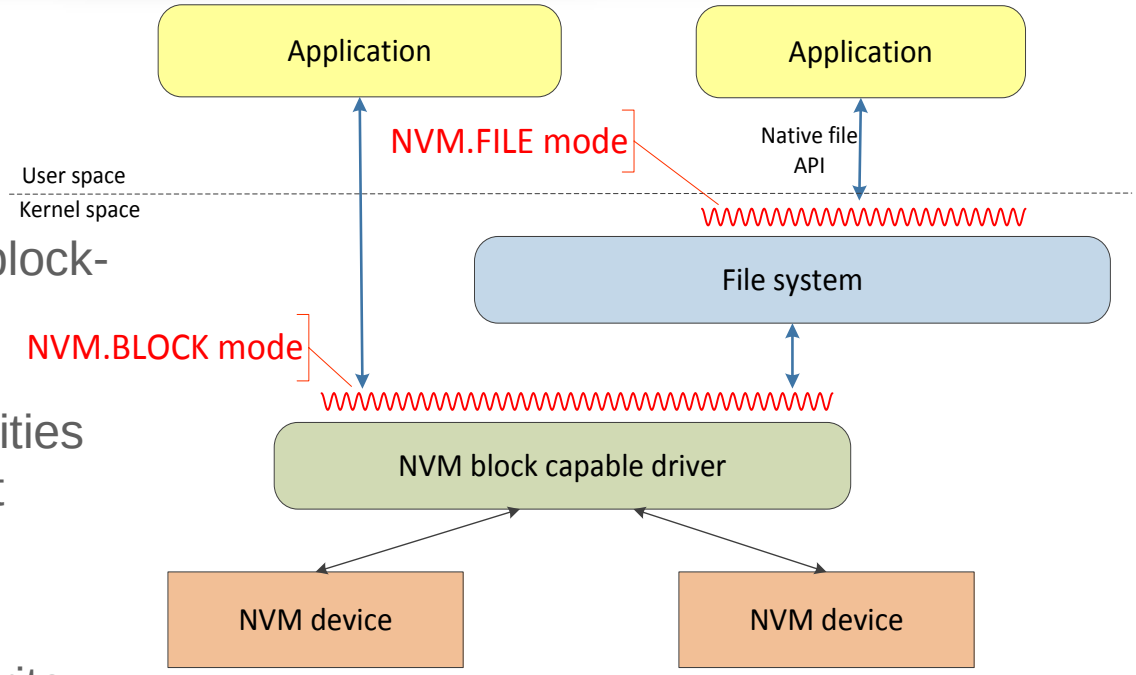
Use with disk-like NVM

NVM.BLOCK Mode

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

NVM.FILE Mode

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



Persistent Memory Modes

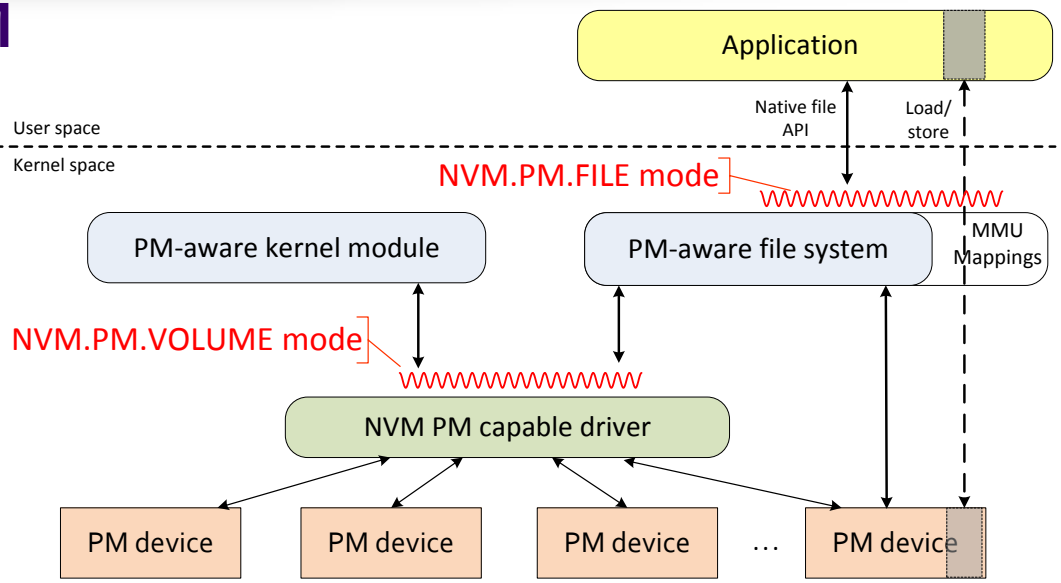
Use with memory-like NVM

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

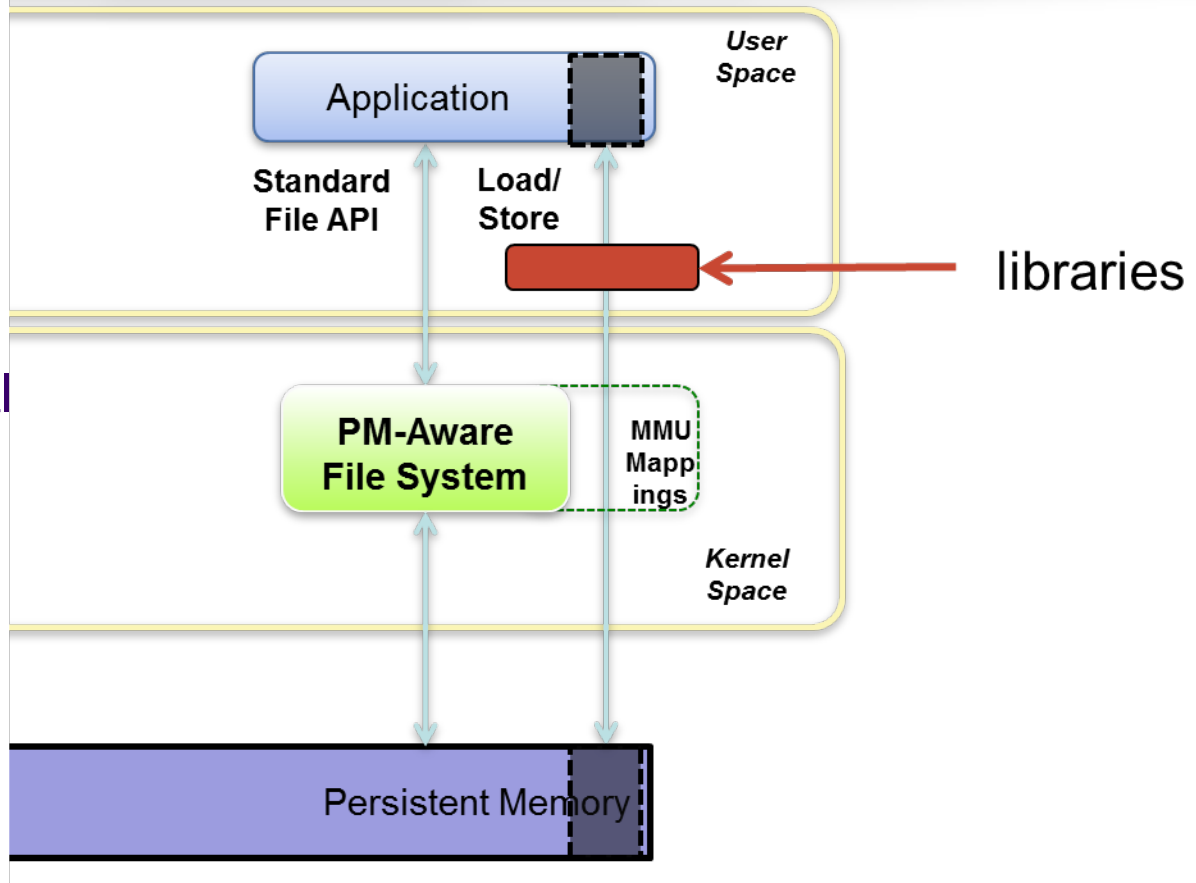
NVM.PM.FILE Mode

- 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
- Refine API with additional libraries that evolve into language extensions
- Add compatible functionality to PM file systems



Current Work

- Remote access white paper
  - Disaggregated memory
  - RDMA direct to NVM
  - High availability, clustering, capacity expansion use cases
  - *NVM PM Remote Access for High Availability* V02R3 (Draft)
- Atomic transactional behavior white paper
  - Add atomicity and recovery to programming model
  - Not addressed by current sync semantics
  - *Atomics Transactions Whitepaper* V4b (Draft)
- Open source contributions
  - Linux PMFS at https://github.com/linux-pmfs
  - Linux Pmem Examples: https://github.com/pmem/linux-examples

NVM Remote Access

NVM Programming Remote Access

- **Use case:**
  - RDMA copy from local to remote persistent memory
    - High availability memory mapped files
    - Built on NVM.PM.FILE from version 1 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

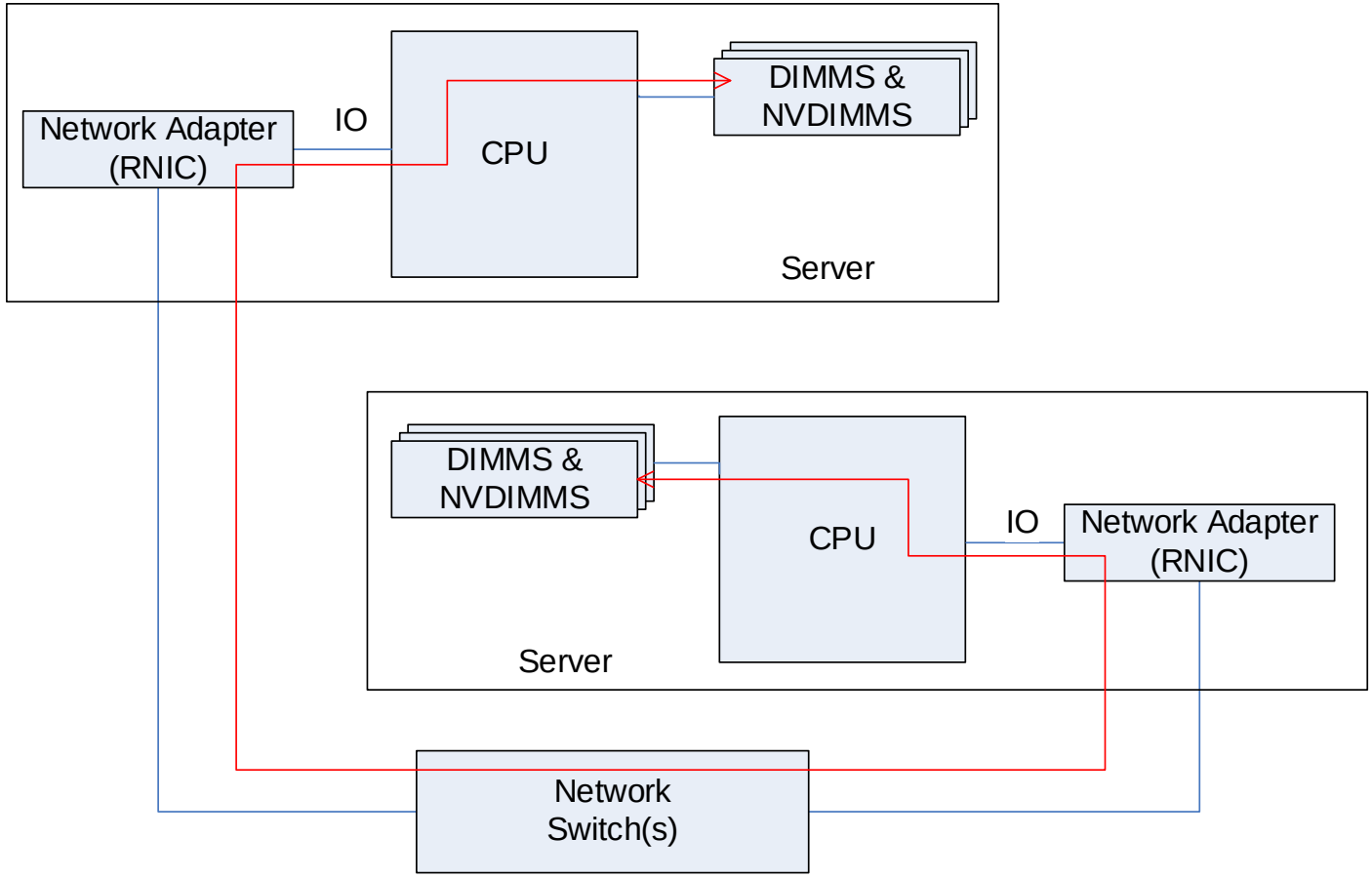
- Consistency Points
  - All data items recovered must have correct values relative to each other from the application point of view
  - Software uses hardware to insure that a failure or restart results in a consistent state
- Crash Consistency
  - Applications must be prepared to recover from *any* 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

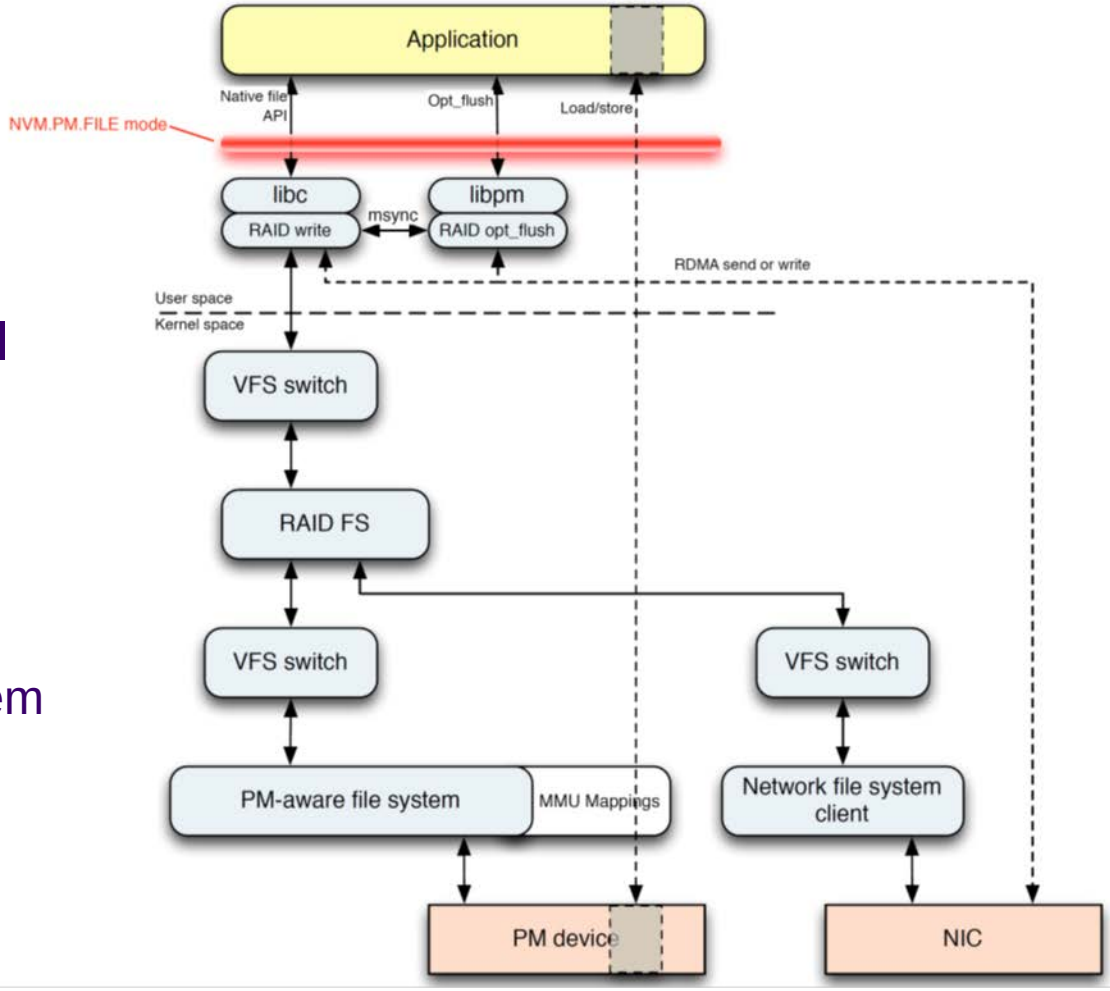
- Atomicity and Atomic Granularity
  - Addresses crash consistency issues
- Optimized NVM Flush
  - Addresses consistency point issues
- Data and Access Recovery Scenarios
  - In-line recovery
  - Backtracking recovery
  - Local application restart
  - Application failover

Remote Access Hardware



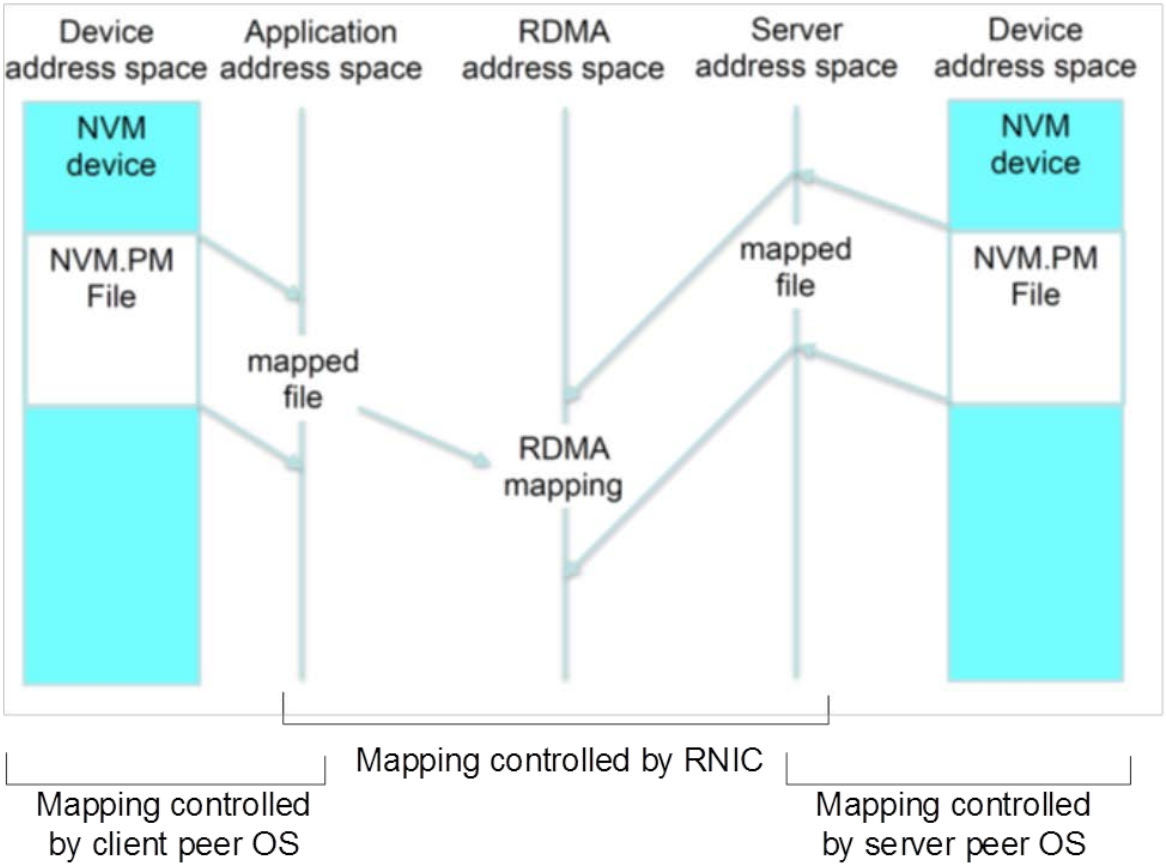
Software Context Example

- Standard file API
- NVM Programming Model optimized flush
- RAID software for 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

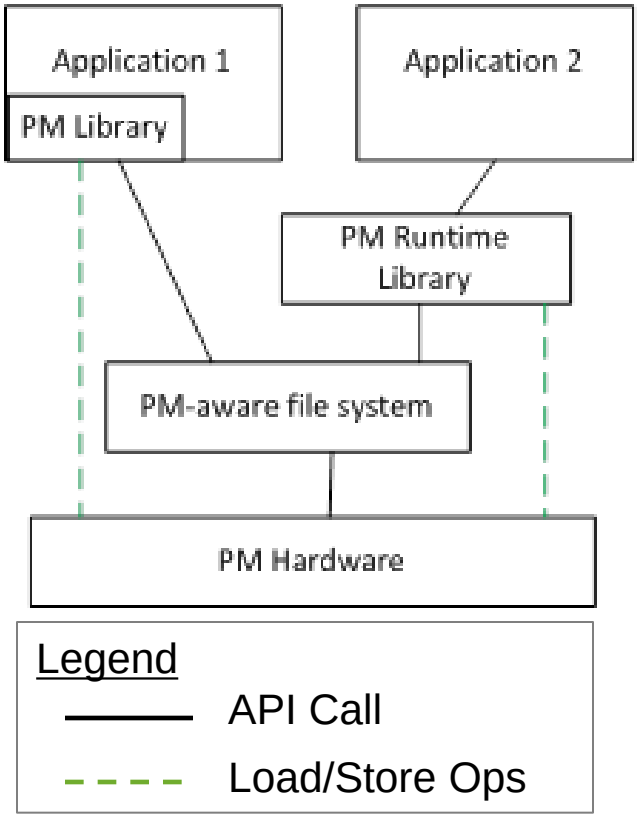
Atomic Transactions

Atomic Transactions

- Guidelines for atomic capabilities and transactions
- Use byte-addressable persistent memory (PM)
  - Like volatile memory (RAM)
  - Use processor load and store operations
- Retains contents across power loss
  - Like storage
- Provide atomic store operations
  - Assure data consistency
  - Known state in the event of store failure

Relationship with the SNIA NVM Programming Model

- Application 1
  - Links in a PM-aware library.
  - Uses a compiler without PM support.
- Application 2
  - Uses a PM-aware compiler
  - And run-time library.



NVM Programming Model Atomic Store Operations

- Provide for atomic store operations to PM
  - Including the event of system or power failure
- Provide operations for large address ranges
  - Or groups of ranges
- Provide atomic durability to PM
- Provide atomic transactions of arbitrary sizes
- No language extensions required
  - Does not preclude the use of C11 and associated memory model.
- No required hardware extensions
  - Atomicity and transactions rely on existing or near term processor capabilities.

Topics

- Use Cases
  - Append to a file atomically
- NVM Programming Model Atomics and Transactions
  - Form a transaction model for PM
  - Review academic and research efforts
- 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

Open Source

Open Source Contributions

- Linux Pmem Examples
  - http://pmem.io/nvml/libpmem/
  - https://github.com/pmem/linux-examples
- Using NVM in a C Application
  - 4:50 PM - 5:20 PM Today
- Processor Support for the NVM Programming Model
  - 4:20 – 4:50 PM Today
- Linux DAX Extensions
  - Support ext4 on NV-DIMMs
  - Implement PM-aware file system (NVM.PM.FILE)
  - http://lwn.net/Articles/588218/

Questions

Acknowledgements

- Thanks for the help!
  - Paul Von Behren, Intel, Chair SNIA NVM Programming TWG
  - Doug Voigt, HP, Chair SNIA NVM Programming TWG
  - Jim Ryan, Intel
  - Steve Byan, NetApp

