



# **Persistent Memory, NVM Programming Model, and NVDIMMs**



# Contents



- Introductions
- Persistent Memory Overview
- NVM Programming Model
- NVDIMM

# Speakers



## **Rob Peglar, SNIA Board Member/Senior Vice President and CTO, Symbolic IO**

Rob is a seasoned technology executive with 39 years of data storage, network and compute-related experience, is a published author and is active on many industry boards, providing insight and guidance. He brings a vast knowledge of strategy and industry trends to Symbolic IO. Rob is on the Board of Directors for the Storage Networking Industry Association (SNIA) and an advisor for the Flash Memory Summit. His role at Symbolic IO includes working with the management team to help drive the future product portfolio, executive-level forecasting, and customer/partner interaction from early-stage negotiations through implementation and deployment.



## **Alan Bumgarner, SNIA Technical Council Advisor/Strategic Planning/DCG Storage & Memory, Intel Corporation**

Alan began his career at Intel Corporation in Folsom, CA more than 20 years ago. Since then his roles included front line technical support, remote server management of multiple Intel datacenters, product/channel/technical marketing, field sales, and strategic product planning. These roles took him from California, to New Jersey, Texas, and Oregon, and finally back to Folsom, CA where he currently holds the position of Senior Strategic Planner for Storage Software in Intel's Datacenter Group. He earned a Bachelor of Science in Business Administration Systems from the University of Phoenix.



## **Tom Talpey, SNIA Technical Council Member/Networked Storage Architect, Microsoft**

Tom is currently an Architect in the File Server Team at Microsoft. His responsibilities include SMB3, SMB Direct (SMB3 over RDMA), and all the protocols and technologies that support the SMB ecosystem. Tom has worked for many years in the areas of network filesystems, network transports and RDMA, and contributed deeply to NFSv3, NFSv4, NFSv4.1, DAFS and SMB3, all of which he layered on RDMA. He recently has been working on applying these to remote access of Persistent Memory, to enable new classes of low latency storage deployable across diverse networks.

# What We Do

## Educate

developers and users on  
Persistent Memory and Solid State Storage



## Develop and Promote

models and architectures

## Collaborate

with leading industry  
alliances and organizations





## Technology Videos on YouTube

[www.youtube.com/user/SNIAVideo](http://www.youtube.com/user/SNIAVideo)



# Resources On Persistent Memory



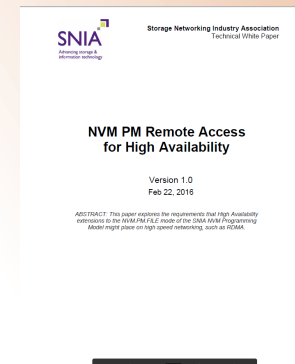
## SNIA White Papers

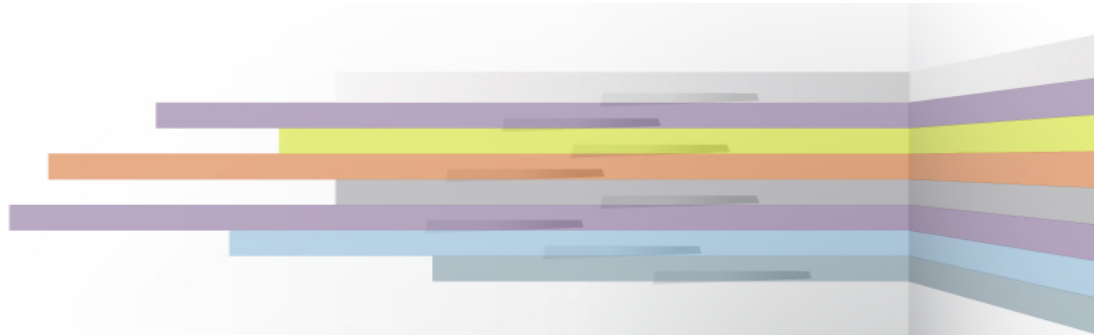
[www.snia.org/PM](http://www.snia.org/PM)

[www.snia.org/PM](http://www.snia.org/PM)

## Summits and Webcasts on Demand

[www.snia.org/webcasts](http://www.snia.org/webcasts)





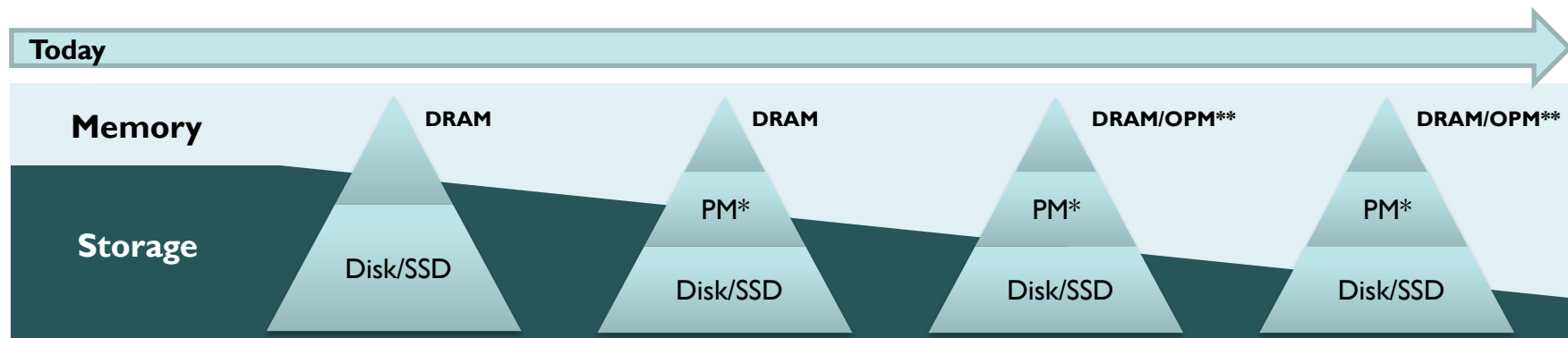
# Persistent Memory



# Memory & Storage Convergence



- Volatile and non-volatile technologies are continuing to converge



\*PM = Persistent Memory

\*\*OPM = On-Package Memory

## New and Emerging Memory Technologies

HMC

3DXPoint™  
Memory

Low Latency  
NAND

HBM

MRAM

Managed  
DRAM

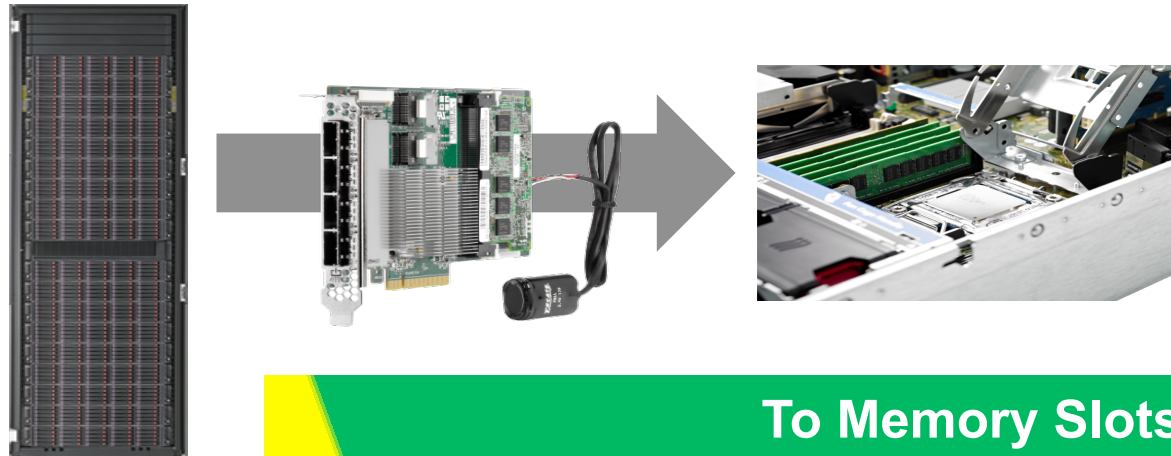
RRAM

PCM

# Persistent Memory (PM) Vision



## Persistent Memory Brings Storage



***Fast***  
Like Memory

**Persistent**  
Like Storage

- For system acceleration
- For real-time data capture, analysis and intelligent response



# Persistent Memory (PM)

is a type of Non-Volatile Memory (NVM)



## ➤ Disk-like non-volatile memory

- ◆ Persistent RAM disk
- ◆ Appears as disk drives to applications
- ◆ Accessed as traditional array of blocks

## ➤ Memory-like non-volatile memory (PM)

- ◆ Appears as memory to applications
- ◆ Applications store data directly in byte-addressable memory
- ◆ No IO or even DMA is required

# Persistent Memory (PM) Characteristics



- Byte addressable from programmer's point of view
- Provides Load/Store access
- Has Memory-like performance
- Supports DMA including RDMA
- Not Prone to unexpected latencies associated with demand paging or page caching
- Think Power Protected RAM



# **NVM Programming Model – Writing Applications for Persistent Memory**

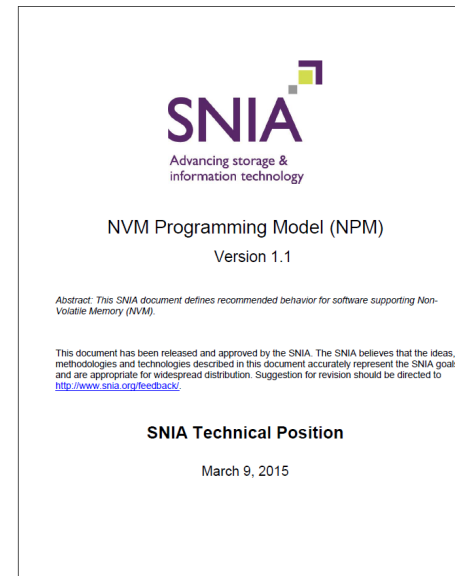


# Role of the NVM Programming Model



## ➤ Rally the industry around a view of Persistent Memory that is:

- ◆ Application centric
- ◆ Vendor neutral
- ◆ Achievable today
- ◆ Beyond storage
  - › Applications
  - › Memory
  - › Networking
  - › Processors



# NVM Programming Model TWG - Mission



## ➤ Accelerate the availability of software that enables Persistent Memory hardware.

- ◆ Hardware includes SSD's and PM
- ◆ Software spans applications and OS's

## ➤ Create the NVM Programming Model

- ◆ Describes application visible behaviors
- ◆ Allows API's to align with OS's
- ◆ Exposes opportunities in networks and processors

# SNIA NVM Programming Model

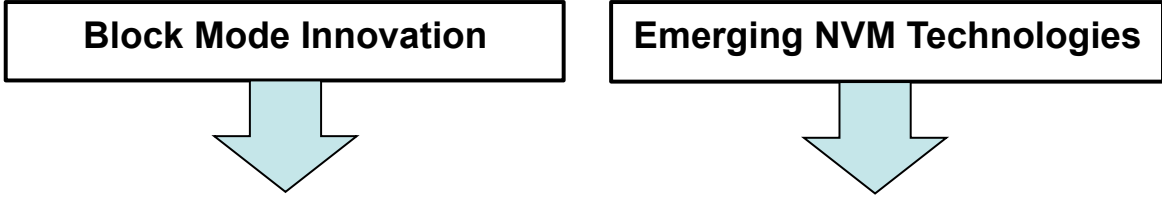


- Version 1.1 approved by SNIA in March 2015
  - ◆ [http://www.snia.org/tech\\_activities/standards/curr\\_standards/npm](http://www.snia.org/tech_activities/standards/curr_standards/npm)
- Expose new block and file features to applications
  - ◆ Atomicity capability and granularity
  - ◆ Thin provisioning 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 NVM Programming Model Has 4 Modes

Block Mode Innovation

Emerging NVM Technologies



	IO	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

The current version (1.1) of the specification is available at  
[http://www.snia.org/tech\\_activities/standards/curr\\_standards/npm](http://www.snia.org/tech_activities/standards/curr_standards/npm)

# Programming Model Modes



- **Block and File modes use IO**
  - ◆ Data is read or written using RAM buffers
  - ◆ Software controls how to wait (context switch or poll)
  - ◆ Status is explicitly checked by software
- **Volume and PM modes enable Load/Store**
  - ◆ Data is loaded into or stored from processor registers
  - ◆ Processor makes software wait for data during instruction
  - ◆ No status checking – errors generate exceptions



# File and Block Mode Extensions

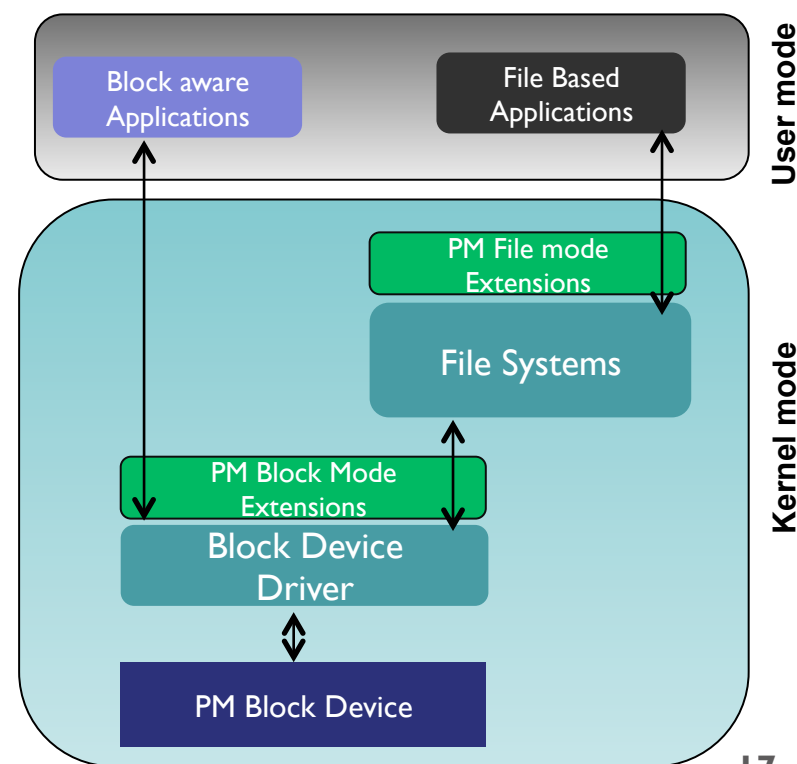


## ➤ NVM.BLOCK Mode

- ◆ 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 (PM) Modes

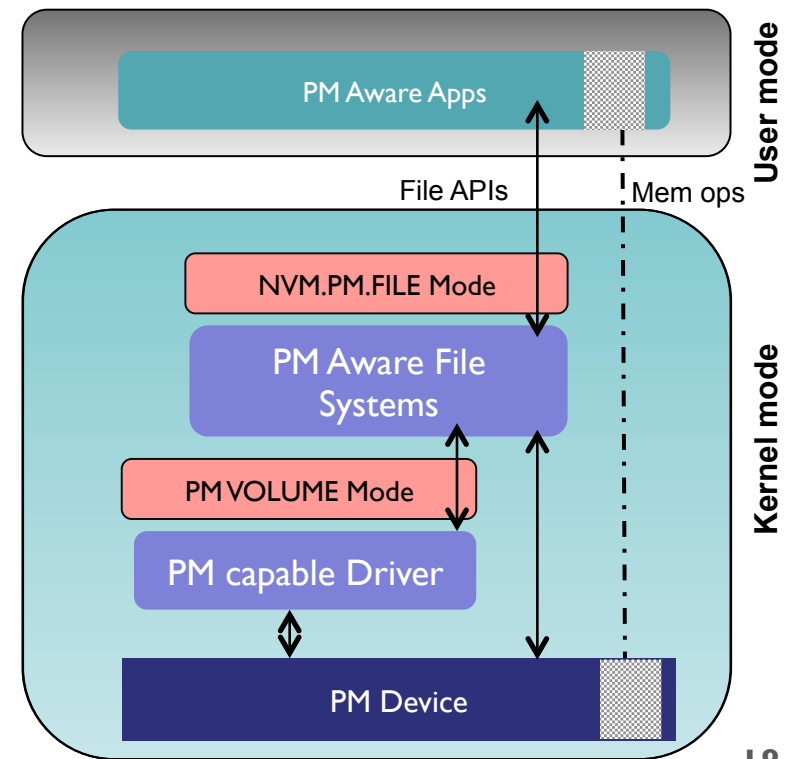


## ➤ NVM.PM.VOLUME Mode

- ◆ Software abstraction for persistent memory hardware
- ◆ Address ranges
- ◆ Thin provisioning management

## ➤ NVM.PM.FILE Mode

- ◆ Application behavior for accessing PM
- ◆ Mapping PM files to application address space
- ◆ Syncing PM files



# Map and Sync



## ◆ Map

- ◆ Associates memory addresses with open file
- ◆ Caller may request specific address

## ◆ Sync

- ◆ Flush CPU cache for indicated range
- ◆ Additional Sync types
- ◆ Optimized Flush – multiple ranges from user space
- ◆ Optimized Flush and Verify – Optimized flush with read back from media

## ◆ Warning! Sync does not guarantee order

- ◆ Parts of CPU cache may be flushed out of order
- ◆ This may occur before the sync action is taken by the application
- ◆ Sync only guarantees that all data in the indicated range has been flushed some time before the sync completes

# PM Pointers



How can one persistent memory mapped data structure refer to another?

- Use its virtual address as a pointer
  - ◆ Assumes it will get the same address every time it is memory mapped
  - ◆ Requires special virtual address space management
- Use an offset from a relocatable base
  - ◆ Base could be the start of the memory mapped file
  - ◆ Pointer includes namespace reference

# Failure Atomicity



## ➤ Current processor + memory systems

- ◆ Guarantee inter-process consistency (SMP)
- ◆ But only provide limited atomicity with respect to failure
  - System reset/restart/crash
  - Power Failure
  - Memory Failure

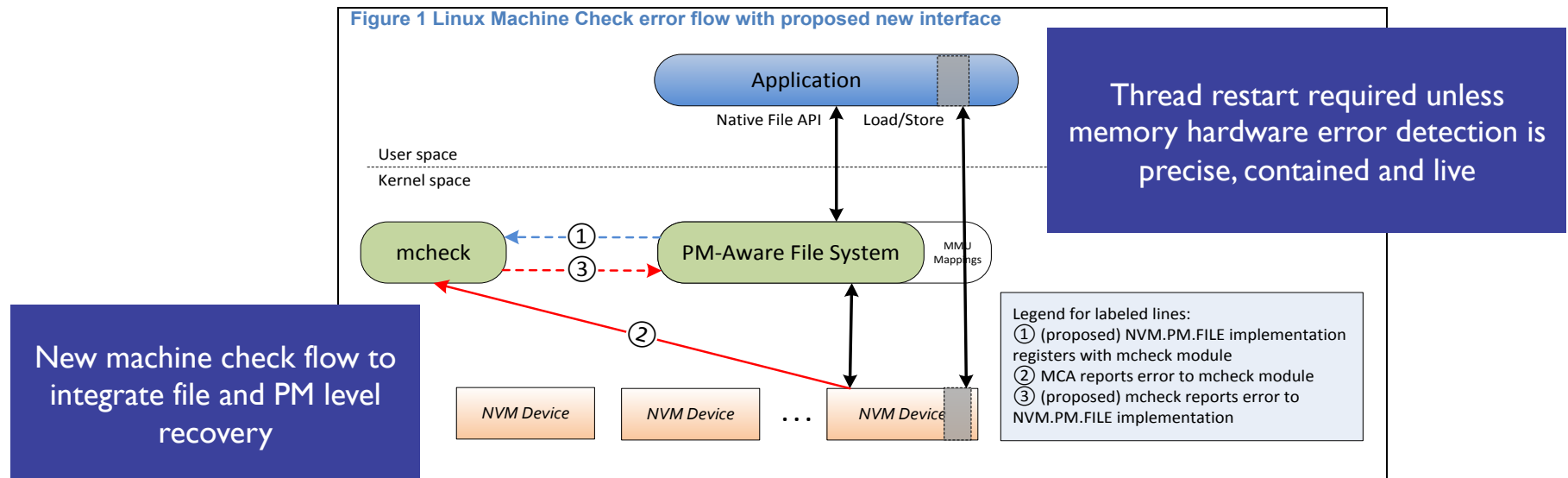
## ➤ Failure atomicity is processor architecture specific

- ◆ Processors provide failure atomicity of aligned fundamental data types
- ◆ Fundamental data types include pointers and integers
- ◆ PM programs use these to create larger atomic updates or transactions
- ◆ Fallback is an additional checksum or CRC

# Error Handling – Exceptions Instead of Status



Figure 1 Linux Machine Check error flow with proposed new interface



- Contained: exact memory location(s) are identified
- Precise: instruction execution can be resumed (RTI)
- Live: reported without restart

- Application gets exception if file level recovery fails or backtracking is needed

# Recent NVMP TWG Work in Progress

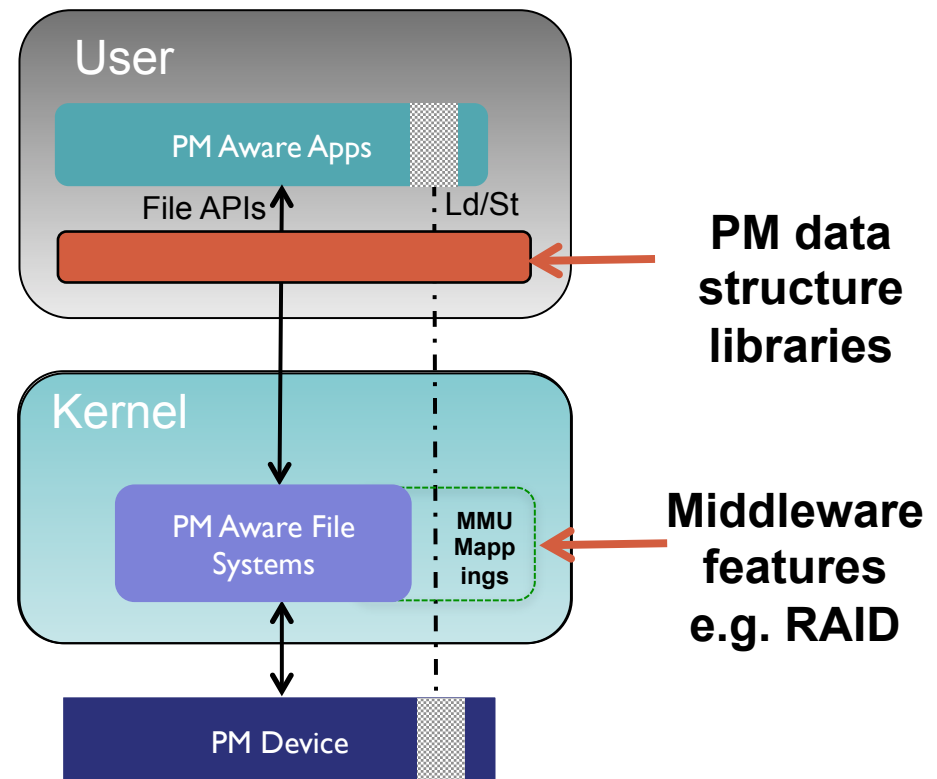


## ➤ Atomicity White Paper in Final Review

Transactional PM Libraries

## ➤ Remote Access for HA White Paper Published

High Availability PM -  
Remote Optimized Flush



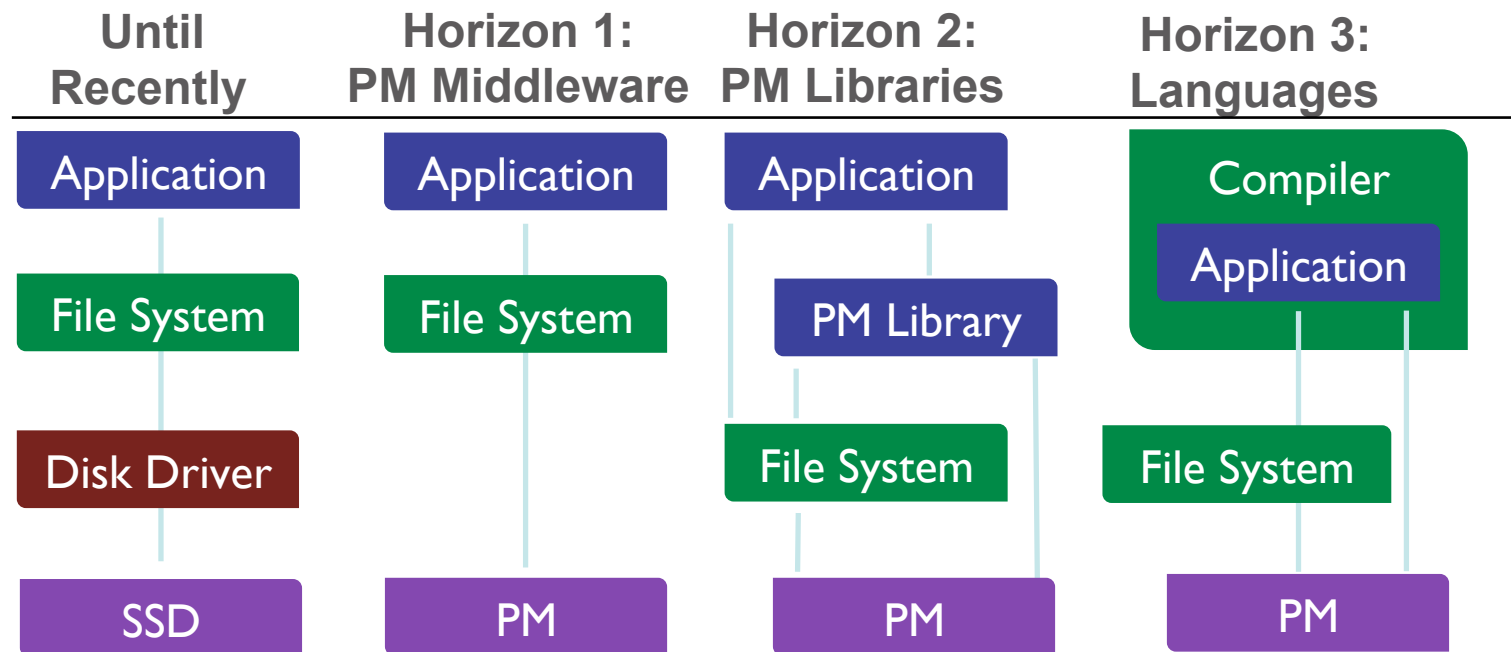
# Summary



- The NVM Programming Model is aligning the industry (<http://pmem.io/>)
  - ◆ Gaining common terminology
  - ◆ Not forcing specific APIs
  - ◆ <http://snia.org/forums/sssi/nvmp>
- What are we doing with it?
  - ◆ PM models expose it
    - › Linux PMFS at <https://github.com/linux-pmfs>
  - ◆ New PM models build on existing ones
    - › Linux Pmem Examples: <https://github.com/pmem/linux-examples>
    - › New TWG work items
- Emerging technologies will drive increasing work in this area as cost comes down



# Application Horizons





**NVDIMM**



# NVDIMM Special Interest Group (SIG)

## Overview

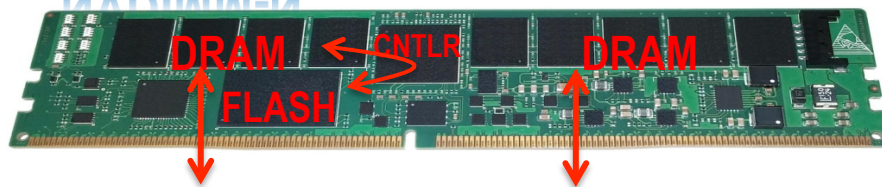


- Member companies collaborate to:
  - ◆ Provide education on how system vendors can design in NVDIMMs
  - ◆ Communicate existing industry standards, and areas for vendor differentiation
  - ◆ Help technology and solution vendors whose products integrate NVDIMMs to communicate their benefits and value to the greater market
- SIG develops vendor-agnostic user perspective case studies, best practices and vertical industry requirements
- SIG evangelizes their activities at conferences, tradeshow, and by webcasts and presentations

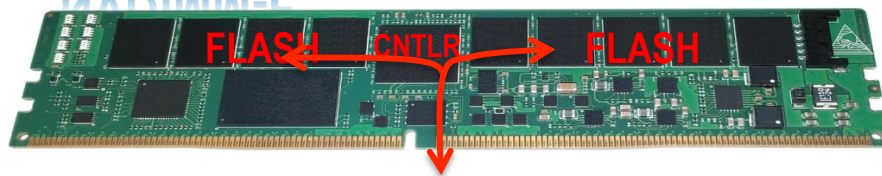
# NVDIMM Types



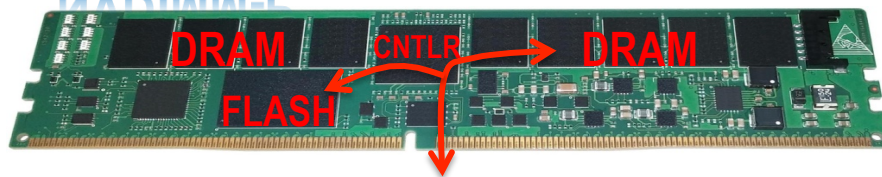
## NVDIMM-N



## NVDIMM-F

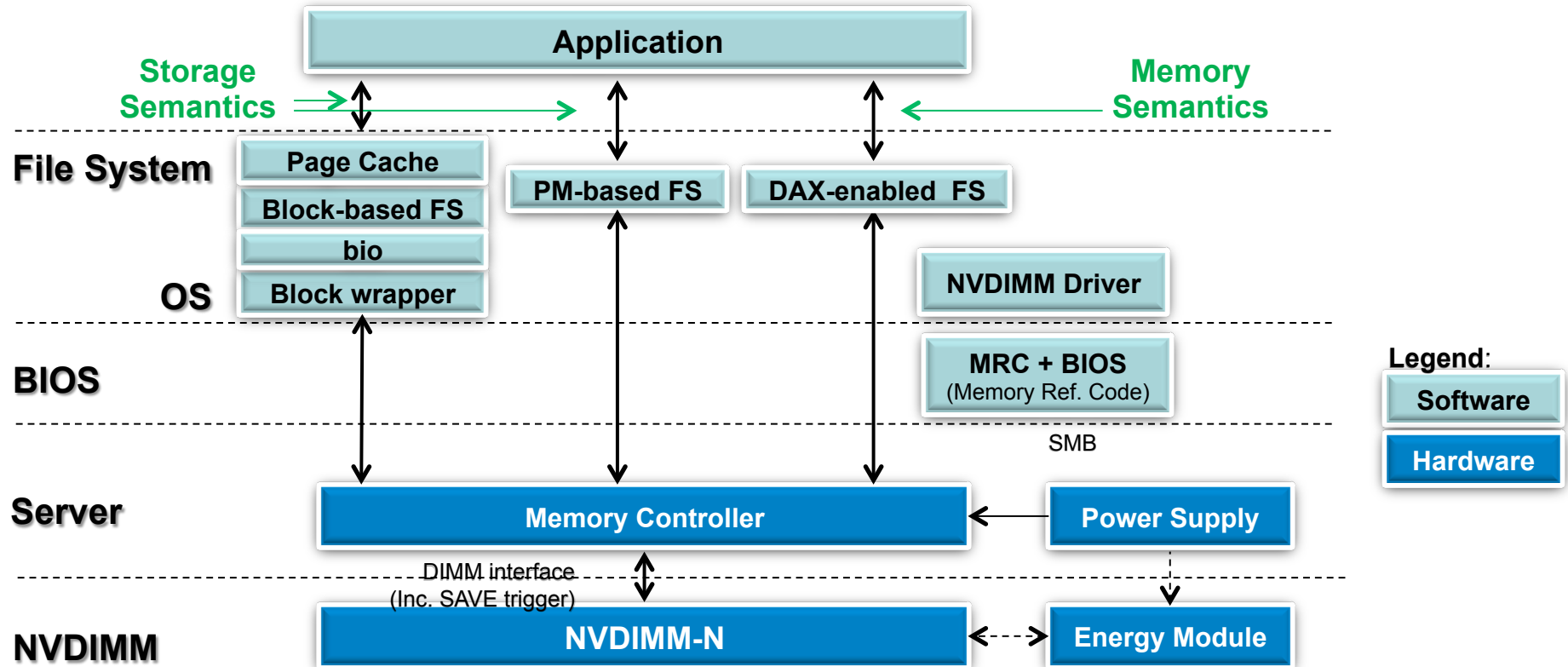


## NVDIMM-P



- ◆ Host has direct access to DRAM
  - ◆ CNTLR moves DRAM data to Flash on power fail
  - ◆ Requires backup power (typically 10's of seconds)
  - ◆ CNTLR restores DRAM data from Flash on next boot
  - ◆ Communication through SMBus (JEDEC std)
- 
- ◆ Host accesses Flash through controller
  - ◆ Block-access to Flash, similar to an SSD
  - ◆ Enables NAND capacity in the memory channel (even volatile operation)
  - ◆ Communication through SMBus (JEDEC std TBD)
- 
- ◆ Combination of -N and -F
  - ◆ Host accesses memory through controller
  - ◆ Definition still under discussion
  - ◆ Sideband signaling for transaction ID bus
  - ◆ Extended addressing for large linear addresses

# NVDIMM Cookbook



# Application Access to NVDIMMs



- Disk-like NVDIMMs (Type F or P)
  - ◆ Appear as disk drives to applications
  - ◆ Accessed using disk stack
- Memory-like NVDIMMs (Type N or P)
  - ◆ Appear as memory to applications
  - ◆ Applications store variables directly in RAM
  - ◆ No IO or even DMA is required
- Memory-like NVDIMMs are a type of persistent memory
- **NVDIMMs are available today!**

# NVDIMM-N Applications

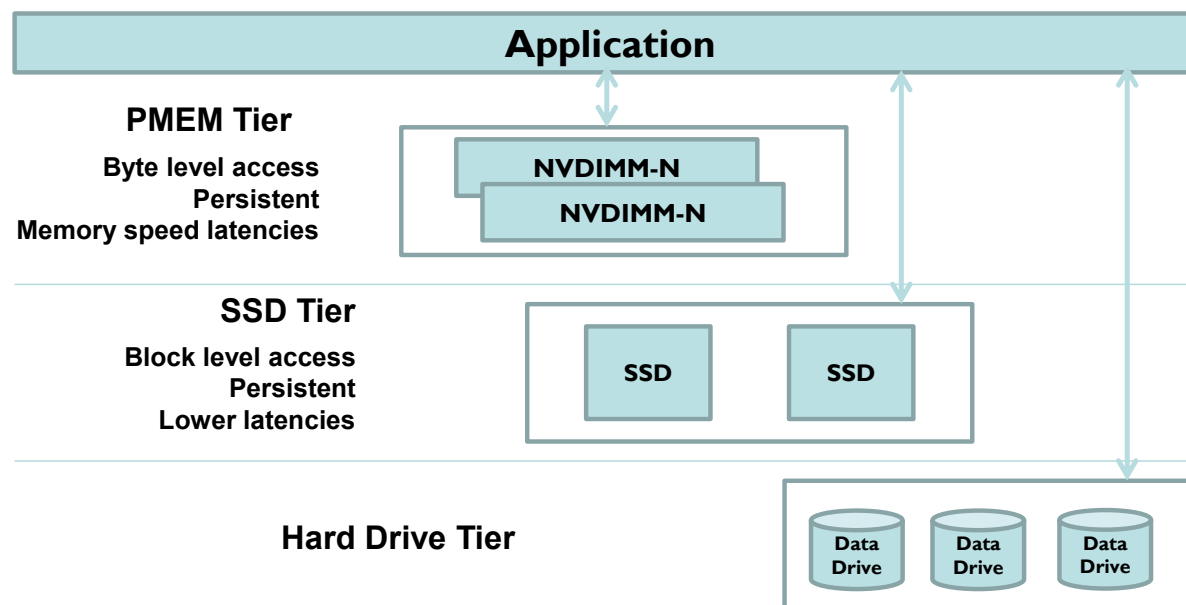


- In-Memory Database: Journaling, reduced recovery time, Ex-large tables
- Traditional Database: Log acceleration by write combining and caching
- Enterprise Storage: Tiering, caching, write buffering and meta data storage
- Virtualization: Higher VM consolidation with greater memory density
- High-Performance Computing: Check point acceleration and/or elimination



# NVDIMM Use Case

## Application Persistent Data Tier





# Linux Kernel 4.4+ NVDIMM-N OS Support



- ❖ Linux 4.2 + subsystems added support of NVDIMMs. Mostly stable from 4.4
- ❖ NVDIMM modules presented as device links: `/dev/pmem0`, `/dev/pmem1`
- ❖ QEMO support (experimental)
- ❖ XFS-DAX and EXT4-DAX available

## DAX

File system extensions to bypass the page cache and block layer to memory map persistent memory, from a PMEM block device, directly into a process address space.

## BTT (Block,Atomic)

Block Translation Table: Persistent memory is byte addressable. Existing software may have an expectation that the power-fail-atomicity of writes is at least one sector, 512 bytes. The BTT is an indirection table with atomic update semantics to front a PMEM/BLK block device driver and present arbitrary atomic sector sizes.

## PMEM

A system-physical-address range where writes are persistent. A block device composed of PMEM is capable of DAX. A PMEM address range may span an interleave of several DIMMs.

## BLK

A set of one or more programmable memory mapped apertures provided by a DIMM to access its media. This indirection precludes the performance benefit of interleaving, but enables DIMM-bounded failure modes.

# Windows NVDIMM-N OS Support



- Windows Server 2016 supports DDR4 NVDIMM-N
- Block Mode
  - ◆ No code change, fast I/O device (4K sectors)
  - ◆ Still have software overhead of I/O path
- Direct Access
  - ◆ Achieve full performance potential of NVDIMM using memory-mapped files on Direct Access volumes (NTFS-DAX)
  - ◆ No I/O, no queueing, no async reads/writes
- More info on Windows NVDIMM-N support:
  - ◆ <https://channel9.msdn.com/events/build/2016/p466>
  - ◆ <https://channel9.msdn.com/events/build/2016/p470>

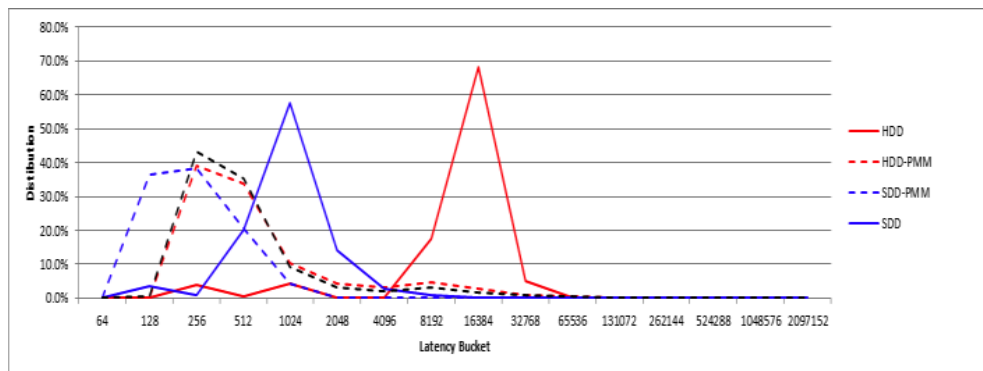
4K Random Write	Thread Count	IOPS	Latency (us)
NVDIMM-N (block)	1	187,302	5.01
NVDIMM-N (DAX)	1	1,667,788	0.52

# Application Benefits – Windows Example



## ◆ Tail of Log in SQL 2016

- ◆ Writes updates to SQL log through persistent memory first
- ◆ Uses memory instructions to issue log updates to persistent memory directly
- ◆ Utilizes memory-mapped files on NTFS Direct Access (DAX) volume



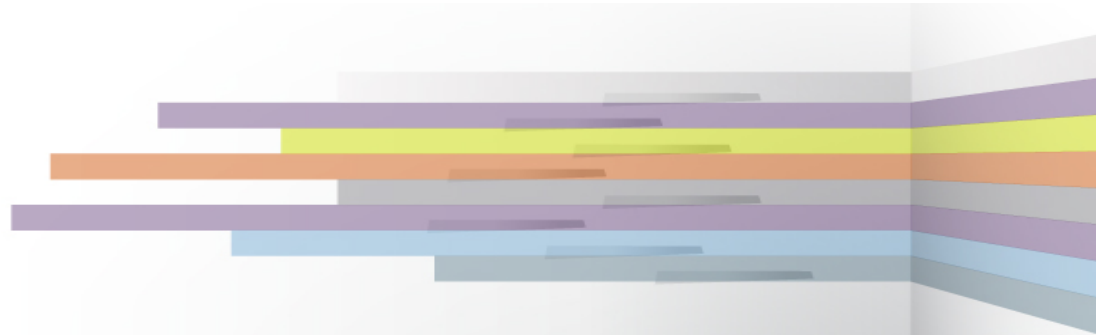
	HDD	HDD-PM	SDD-PM	SDD
64 us]	0.0%	0.0%	0.0%	0.0%
128 us]	0.0%	0.1%	<b>36.3%</b>	3.5%
256 us]	3.9%	<b>39.2%</b>	<b>38.3%</b>	0.9%
512 us]	0.4%	<b>34.0%</b>	<b>20.7%</b>	<b>20.1%</b>
1024 us]	4.4%	<b>10.4%</b>	4.5%	<b>57.6%</b>
2048 us]	0.0%	4.2%	0.1%	<b>14.2%</b>
4096 us]	0.1%	3.0%	0.0%	2.6%
8192 us]	<b>17.6%</b>	4.7%	0.0%	0.9%
16384 us]	<b>68.2%</b>	2.6%	0.0%	0.2%
32768 us]	<b>5.0%</b>	1.0%	0.0%	0.0%
65536 us]	0.3%	0.6%	0.0%	0.0%
131072 us]	0.1%	0.1%	0.0%	0.0%
262144 us]	0.0%	0.0%	0.0%	0.0%
524288 us]	0.0%	0.0%	0.0%	0.0%
1048576 us]	0.0%	0.0%	0.0%	0.0%
2097152 us]	0.0%	0.0%	0.0%	0.0%

Source: Microsoft

# Summary



- The NVM Programming Model is perfect for NVDIMMs
  - ◆ Block and File mode atomicity features for Type F
  - ◆ PM Mode memory mapped storage for Type N
  
- Use the NVM programming model with NVDIMMs
  - ◆ Enable a path forward for applications
  - ◆ Lead the way to innovation in NVM optimized software



**Thank You!**





## Backup Slides



# SNIA NVM Programming Model



- Developed to address the ongoing proliferation of new non-volatile memory functionality and new persistent memory technologies.
- The NVM Programming Model is necessary to enable an industry-wide community of persistent memory producers and consumers to move forward together through a number of significant storage and memory system architecture changes.
- The specification defines recommended behavior between various user space and operating system (OS) kernel components supporting persistent memory.
- The specification describes several operational modes of access. Each mode is described in terms of use cases, actions and attributes that inform user and kernel space components of functionality that is provided by a given compliant implementation.

# Persistent Memory Modes



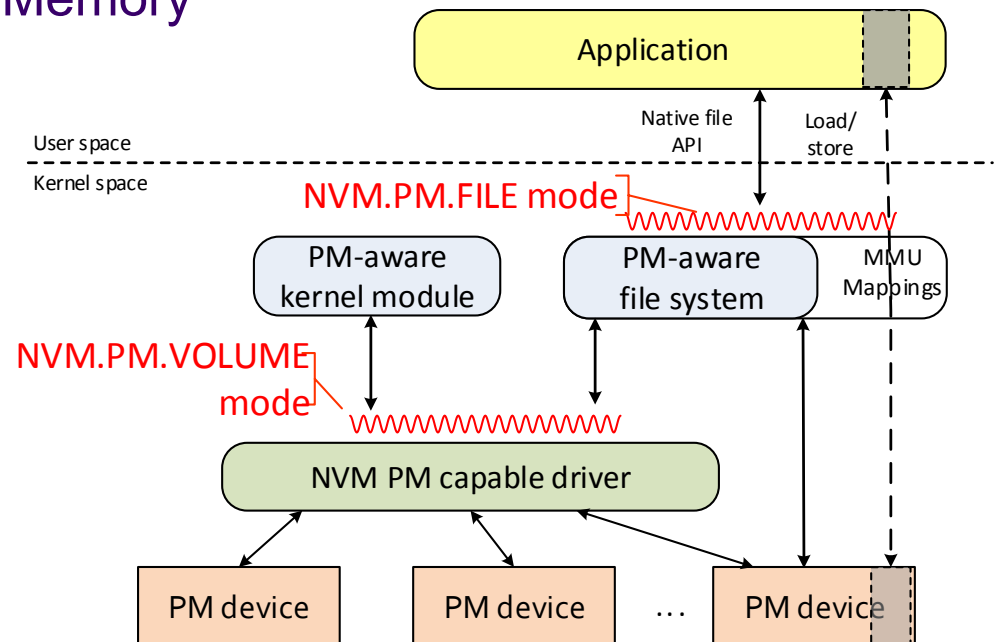
## ➤ Use with memory-like Persistent Memory

### ➤ 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





# Expected Usage of PM Modes



## ◆ Uses for NVM.PM.VOLUME

- ◆ Kernel modules
- ◆ PM aware file systems
- ◆ Storage stack components

## ◆ Uses for NVM.PM.File

- ◆ Applications
  - › Persistent datasets, directly addressable, no DRAM footprint
  - › Persistent caches (warm cache effect)
- ◆ Reconnect-able BLOBs of persistence  
(Binary Large Object – set of binary data stored as a single entity)
  - › Naming
  - › Permissions