SNIA NVM Programming Model

Paul von Behren, Intel Corporation

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

- SNIA NVM Programming Model
- Recent advances in Non-Volatile Memory (NVM) technologies blur the line between storage and memory, creating a disruptive change to the way software is written. The new *SNIA NVM Programming Model* describes behavior provided by operating systems enabling applications, file systems, and other software to take advantage of new NVM capabilities. This tutorial describes how the programming model addresses NVM extensions for existing devices (such as SSDs) and persistent memory (such as NVDIMMs). It also describes the differences between software written for block storage (SSDs, disks) and persistent memory and the potential benefits of adapting software for persistent memory.

Overview

- Why is NVM programming important?
- What is NVM (relative to the programming model)
- The programming modes covered in the model
  - Modes for devices providing block storage behavior
  - Modes for devices providing memory behavior
- Wrap-up
  - Key takeaways for the data center

Why is NVM Programming Important?

- From the perspective of file system and applications, most NVM hardware emulates hard disks
  - With some NVM specific behavior
- But applications may benefit from SW extensions tailored to NVM hardware
  - For example, ATA TRIM command to enhance SSD endurance
    - Similar SCSI and NVM-Express commands
  - Requests from users/developers to enhance SW stack to enable access to NVM extensions
  - Emerging persistent memory may benefit from new programming approach

SNIA NVM Programming TWG

- Members:
  - Calypso Systems, Cisco, Contour Asset Management, Dell, EMC, FalconStor, Fujitsu, Fusion-io, HP, HGST, Hitachi, Huawei, IBM, IDT, Inphi, Intel, Intuitive Cognition Consulting, LSI, Marvell, Micron, Microsoft, NEC, NetApp, OCZ, Oracle, PMC-Sierra, Qlogic, Red Hat, Samsung, SanDisk, Seagate, Sony, Symantec, Tata Consultancy Services, Toshiba, Viking, Virident, VMware
- Charter:
  - Develop specifications for new software programming models as NVM becomes a standard feature of platforms
- Status: SNIA *NVM Programming Model* specification nearing publication (draft available)

What is NVM?

- As used in the SNIA *NVM Programming Model*, NVM is
  - Any type of non-volatile memory
  - Includes disk form factor SSDs
    - Using SATA, SCSI, vendor-specific, or commands
  - Includes NVM PCIe cards
    - Using NVM-Express, SCSI-Express, SATA Express, vendor-specific, or other commands
  - Also includes emerging persistent memory (PM) hardware
    - Such as NVDIMMs

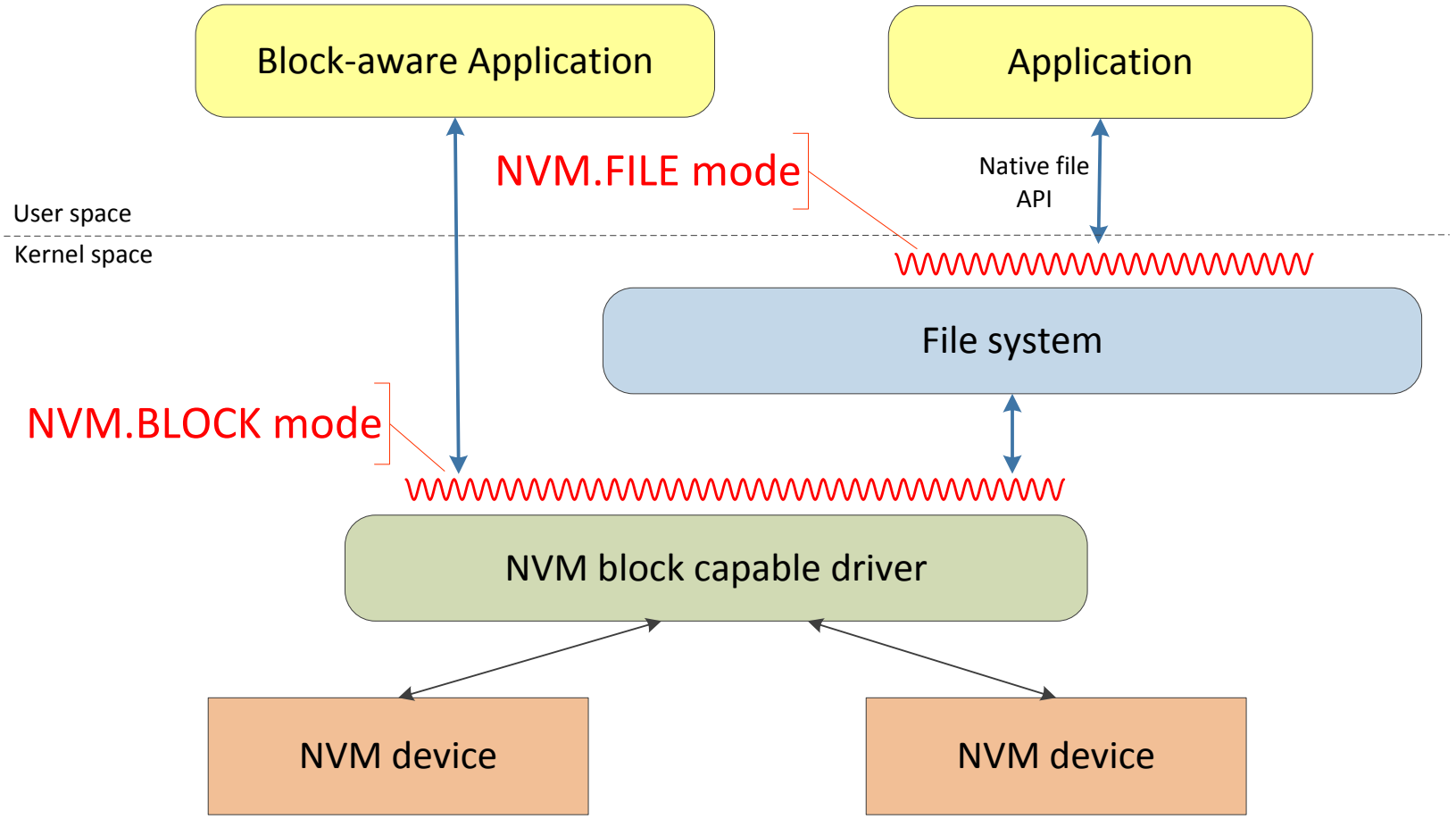
NVM Programming Modes

- *NVM Programming Model* spec addresses four modes
  - Applications using file systems optimized for block storage (HDDs/SSDs)
  - File systems (and advanced applications) optimized for block storage
  - Applications using file systems optimized for persistent memory
  - File systems optimized for persistent memory
- *NVM Programming Model* addresses NVM extensions to existing block storage software stack
- PM programming model is new, but builds on legacy file systems
- More modes may be added later (for new types of HW)

Block storage modes

- NVM.FILE mode - primary targets:
  - Applications striving to optimize I/O for NVM storage
  - Mission-critical applications needing to assure data on disk is consistent even when write operations are interrupted by power failures
- NVM.BLOCK mode – primary targets:
  - File systems
  - Other operating system (OS) components (such as hibernation) that directly use storage
  - Applications that use block commands directly (without a file system)
    - For example, some backup and database programs

NVM.BLOCK/FILE Modes



NVM.FILE Enhancements

- Additional device granularities (such as block sizes)
  - SSDs may have different physical and logical block sizes
    - And have commands allowing software to determine these sizes
  - There are additional granularities of interest to software
    - Does the device implementation assure that write operations smaller than a given size will not be "torn" due to a power failure?
    - Does the device have a checksum that applies to multi-byte granules? If so, it may be impossible to read any byte in the checksum granule, if one byte is unreadable.
    - These granules may not be the same as logical or physical block size
    - Software can use information about these granularities to place data in a way that survives certain types of device failures
  - *NVM Programming Model* specifies behavior for surfacing this information to OS components and applications

NVM.FILE Enhancements: Atomic Write

- Atomic writes are an emerging NVM device feature that assures data is consistent when a write operation is interrupted (for example, power failure)
- Applications may write data twice to achieve the same assurance for devices that don't support atomic write
  - On restart, the application uses the two copies to recover
- In other words, atomic writes achieve a high level of data integrity with fewer write commands
  - Fewer writes means higher performance
- Some NVM devices have advanced operation supporting atomic write to non-sequential block addresses

NVM.BLOCK Enhancements

- Atomic Write Behavior and Granularities
  - Similar to what's defined for NVM.FILE
- Scar command
  - Provides a way for one software component to mark blocks so that other software components see an error when they attempt to read the blocks.
  - The error condition is removed when the blocks are written

NVM.BLOCK Enhancements: Discard

- SSDs often have strategies to help increase endurance and performance by organizing the way blocks are written
  - *Wear leveling* is more effective when a higher percentage of blocks are known to be unused by the OS
  - Discard (AKA Trim) commands provide a way for software to inform the SSD that blocks are no longer in use
- First generation of discard commands met objectives for wear leveling
  - But the results of subsequent reads were not predictable
  - a read of a discarded block might return zeros, old data, a device-specific pattern

NVM.BLOCK Enhancements: Discard

- Standards (and devices) are being updated to address the read behavior
  - And defining new commands to allow software to determine which discard variants are supported
- *NVM Programming Model* defines ways for software to
  - determine what types of discard commands a device supports
  - issue discard commands
  - determine whether a block has been discarded

Persistent memory characteristics

- For this presentation, persistent memory (PM) is:
  - Not tablet-like memory for entire system
  - Not flash (as commonly used today)
  - Not Block oriented (like HDDs/SSDs)
  - Is a new type of hardware:
    - Think "battery-backed DIMMs"
    - But may appear in other formats – such as PCIe cards
- PM does not require apps to copy memory to/from disk
  - But PM does benefit from a different approach to using memory

Integrating PM with software

- Software can't use PM without modification
  - Current practice: applications allocate memory by size
    - Memory contents zero or undefined; applications must initialize
    - Memory allocated from pool; may not get same memory across a restart
- Three approaches to enable SW use of PM
  1. Hide PM from legacy applications, make it available for specific applications through non-standard APIs
     - Great for first generation SW; explore PM capabilities, limit issues
     - SW apps typically wait for OS integration to use new hardware
  2. Make the PM appear to be block devices ("virtual SSDs")
     - Use NVM.BLOCK/NVM.FILE modes; existing apps work without modification
     - Doesn't allow apps to take advantage of all PM features
  3. *NVM Programming Model* defines a new programming mode
     - Enable optimal use of PM

PM modes overview

- Based on existing memory-mapped file behavior
- Rather than applications allocating ranges of anonymous memory, PM volumes have names
- A PM-aware application maps a PM Volume to application virtual memory addresses
- The application directly works with these addresses in the same way it would use volatile memory
  - But PM content is non-volatile
- Depending on the OS and file-system, applications may be able to load/store data directly to PM (with no additional OS/FS overhead)

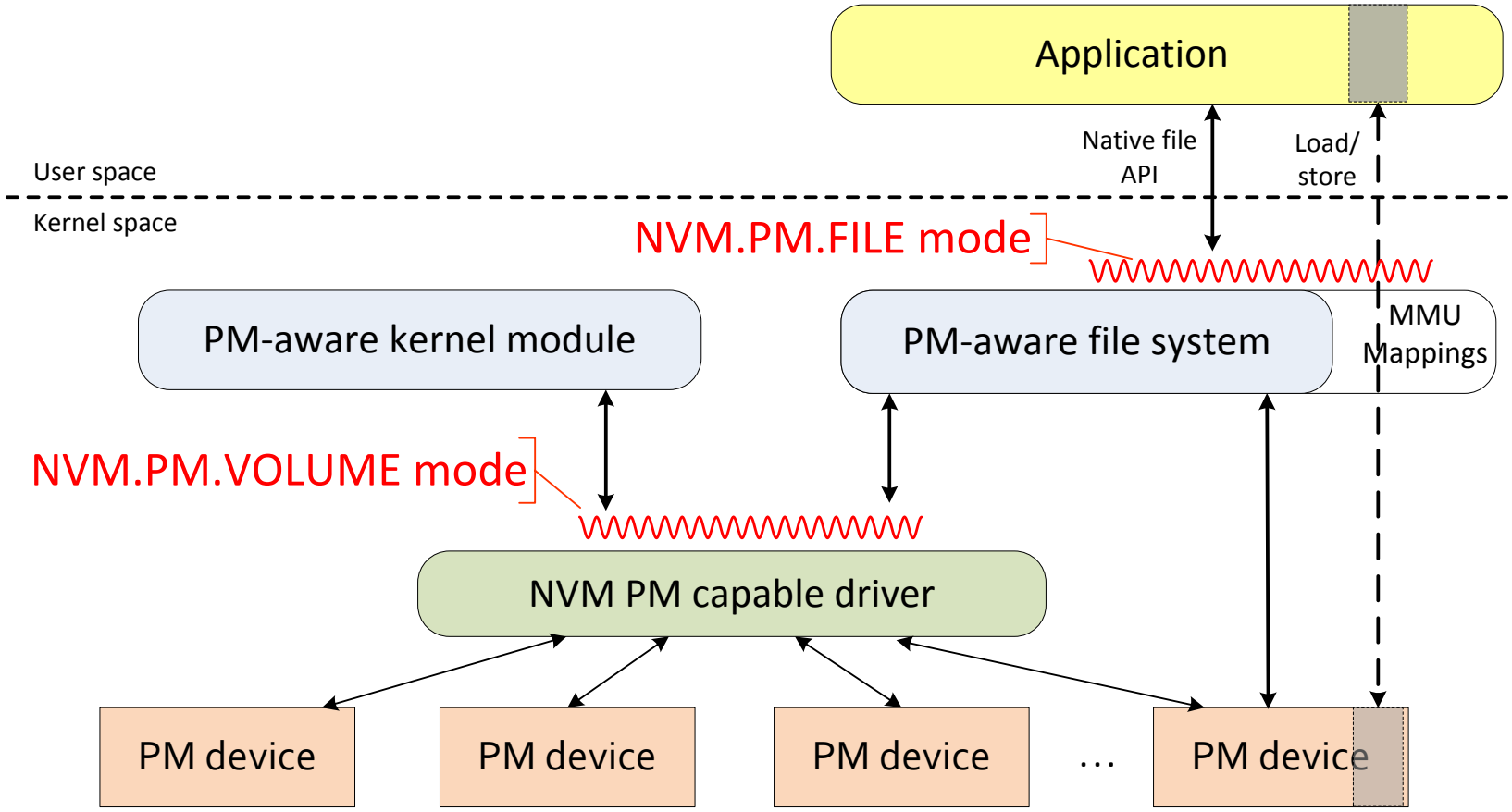
Impact of this programming mode

- What types of applications benefit from this mode?
  - In-memory databases
    - No need to wait for memory to be re-populated from disk at startup
  - Applications using small data objects
    - For example: cloud software object stores, key/value or NoSQL databases
  - Persistent caches
    - Caches get "smarter" over time – learn what data benefits most from residing in cache
    - With volatile memory, cache SW must re-learn after each restart
- Some applications may not see performance benefits from a PM-aware programming mode
  - "Virtual SSD" mode may be best way to utilize PM

Persistent Memory Modes

- NVM.PM.FILE mode - primary targets:
  - Enhanced application support related to persistent memory
  - Enable optimize hardware performance
  - Assure saved data is consistent even when operations are interrupted by power failures
- NVM.PM.VOLUME Mode
  - PM-aware file systems
  - Other PM-aware OS components

Persistent Memory Modes



NVM.PM.FILE features

- Based on existing memory-mapped file behavior
- Use existing file system approach to naming and permissions for PM
- Load/store access (existing operations for memory access)
- SYNC operations
  - Assure data is flushed to hardware
- GET_ERROR_INFO
  - Generic interface to details of PM errors

NVM.PM Volume

- GET_RANGESET
  - Allows OS component (like a file system) to map volume-relative addresses to physical memory addresses
- SYNC actions
  - Hardware-independent interface to flush address ranges
  - Supports smaller byte ranges than POSIX msync
- Granularities
  - Similar to NVM.BLOCK – allow OS components to place data to prevent single PM error from impacting two copies of same data
- Discard
  - Similar to NVM.BLOCK – allow OS components to tell PM controller that a range of bytes is no longer in use

Key takeaways for data centers

- At this time, PM hardware is not common in servers
  - But PM expected to be more common in the next couple years
- PM "virtual SSDs" expected to outperform conventional SSDs
  - Allowing existing software to work as-is
  - But this approach still limited by a SW stack optimized for HDDs
- The "killer apps" for PM include
  - Apps that copy a lot of data from disk to memory at startup
  - Cache components that gets more efficient over time
  - SW that reads/writes small chunks of data

Key takeaways for data centers

- Also research application support for block NVM features
  - Such as atomic write and smart use of granularities
- Enabling application use of an NVM device feature may require updates to drivers, OS, and file systems
- Ask vendors about support for SNIA *NVM Programming Model*
  - Programming Model helping drive broad range of PM-aware SW
  - Support for model should help applications work across multiple NVM devices

More information

- SNIA portal for NVM Programming:
  - Up-to-date information about the *NVM Programming Model*, related materials and software
  - Links to approved versions will be added to this portal
  - http://snia.org/forums/sssi/nvmp
- Draft specification may be downloaded here:
  - http://snia.org/sites/default/files/NVMProgrammingModel_v1r5DRAFT.pdf
- Questions? Comments?
  - nvmptwg-info@snia.org

Attribution & Feedback

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

Authorship History

Paul von Behren, September 3rd, 2013

Updates:

Name/Date
Name/Date
Name/Date

Additional Contributors

Andy Rudoff

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