SDC 18
September 24-27, 2018
Santa Clara, CA
www.storagedeveloper.org

Update on the SNIA Persistent Memory Programming Model in Theory and Practice

Andy Rudoff
Intel

Agenda

- Why create the NVM Programming TWG?
- What the NVM Programming Model means to most people
- Details on actual implementations, specific to:
  - Intel
  - Linux
  - Windows
  - Virtualization
- Areas unspecified by NVMP
- Future work

What Motivated Us to Create the TWG

- Concerning direction in ecosystem
  - Products were emerging with private APIs
  - ISVs forced to choose to ”lock in” to a product
  - Lots of mis-information, conflicting APIs

- But my reasons (Intel) were…

intel OPTANE™ DC PERSISTENT MEMORY

intel XEON PLATINUM inside™

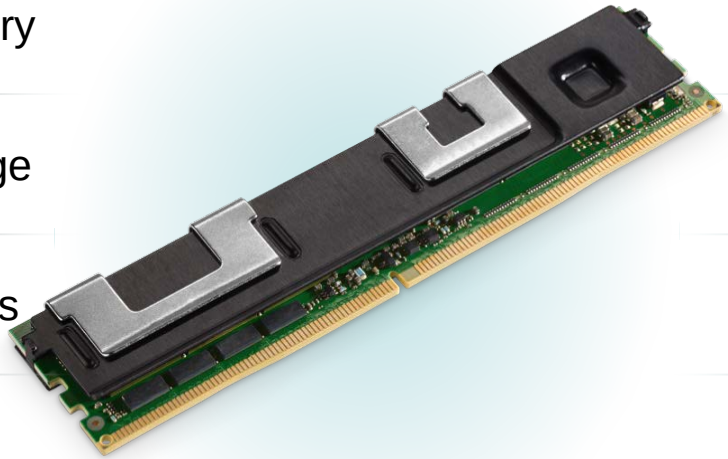
Big and Affordable Memory

128, 256, 512GB

High Performance Storage

DDR4 Pin Compatible

Direct Load/Store Access



Hardware Encryption

Native Persistence

High Reliability

What Everyone Should Know About Persistent Memory

- There are **many** ways to use it without modifying your program or even knowing it is installed in the system
- **Some** applications will want direct access to it
  - Best way to fully leverage what it can do

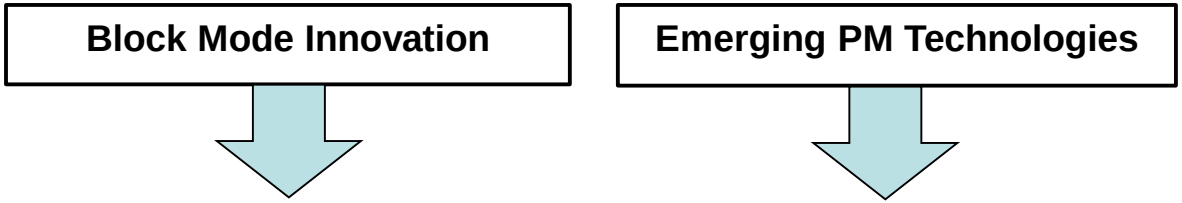
- The programming model is for those apps
  - Libraries like PMDK build on it too

The SNIA Programming Model

- High order bits
  - Model, not API
  - In-kernel and User space

- To be honest…
  - My goal of using memory-mapped files for pmem was achieved with spec version 1.0 and the rest has been value on top of that

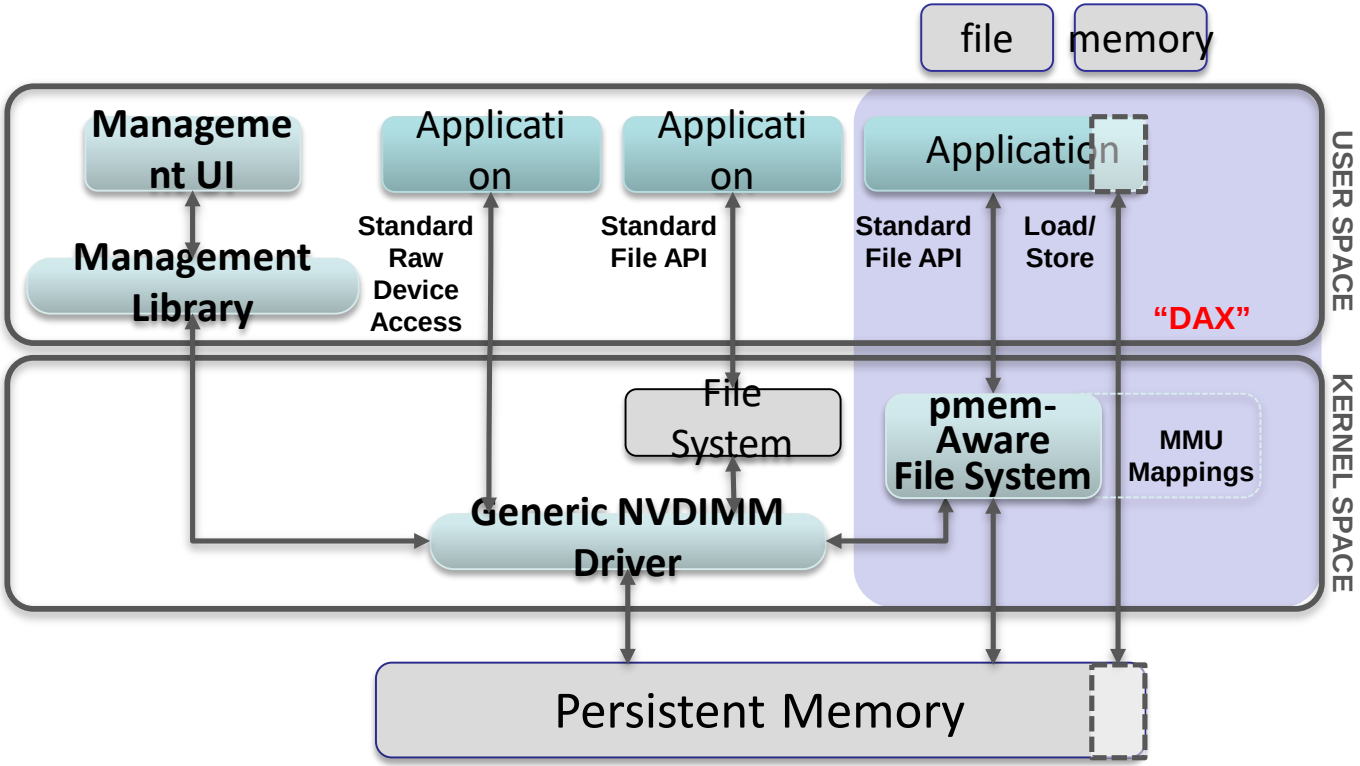
Doug's FMS Slides on the Four Modes



	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 (1.2) of the specification is available at
https://www.snia.org/sites/default/files/technical_work/final/NVMProgrammingModel_v1.2.pdf

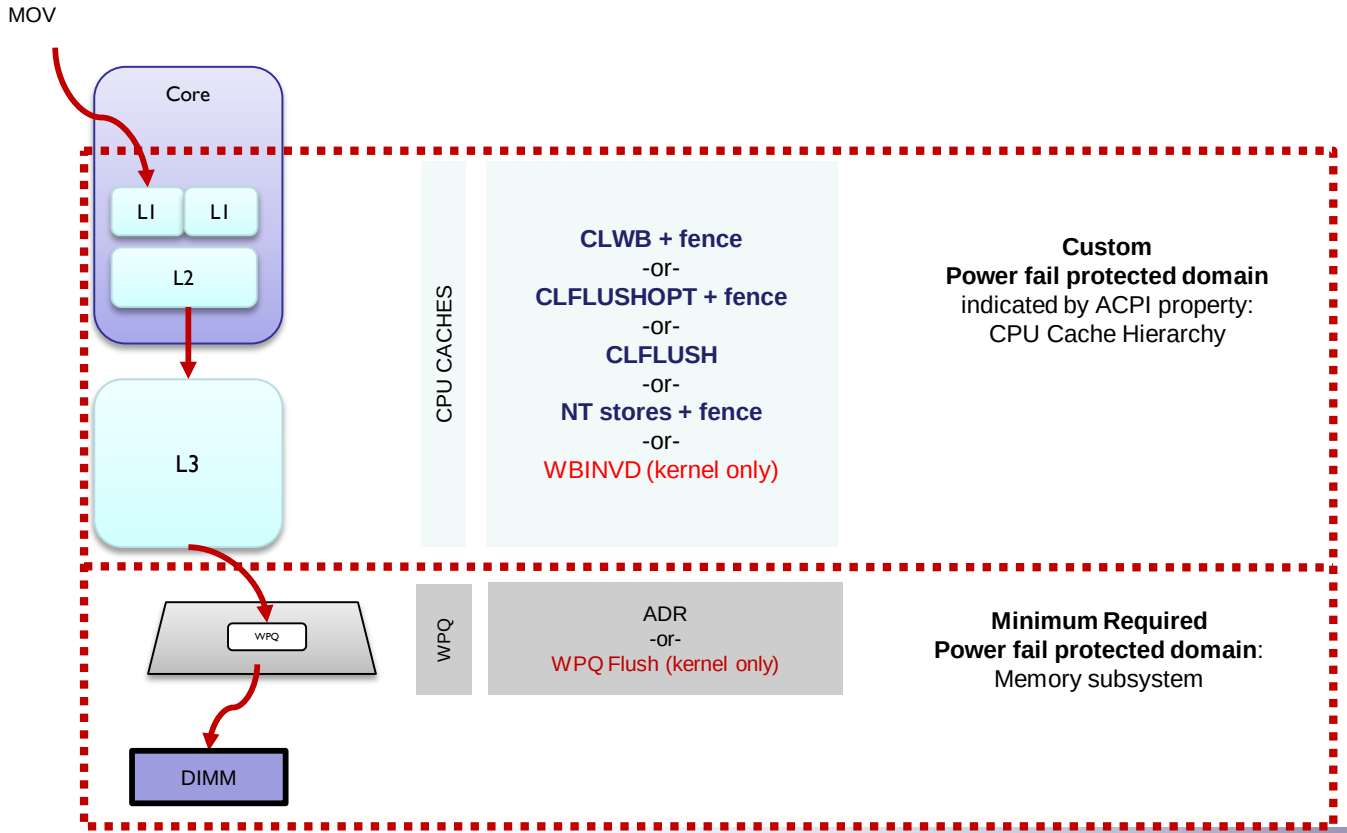
My Summary of the Programming Model



Intel-Specific Implementation Details

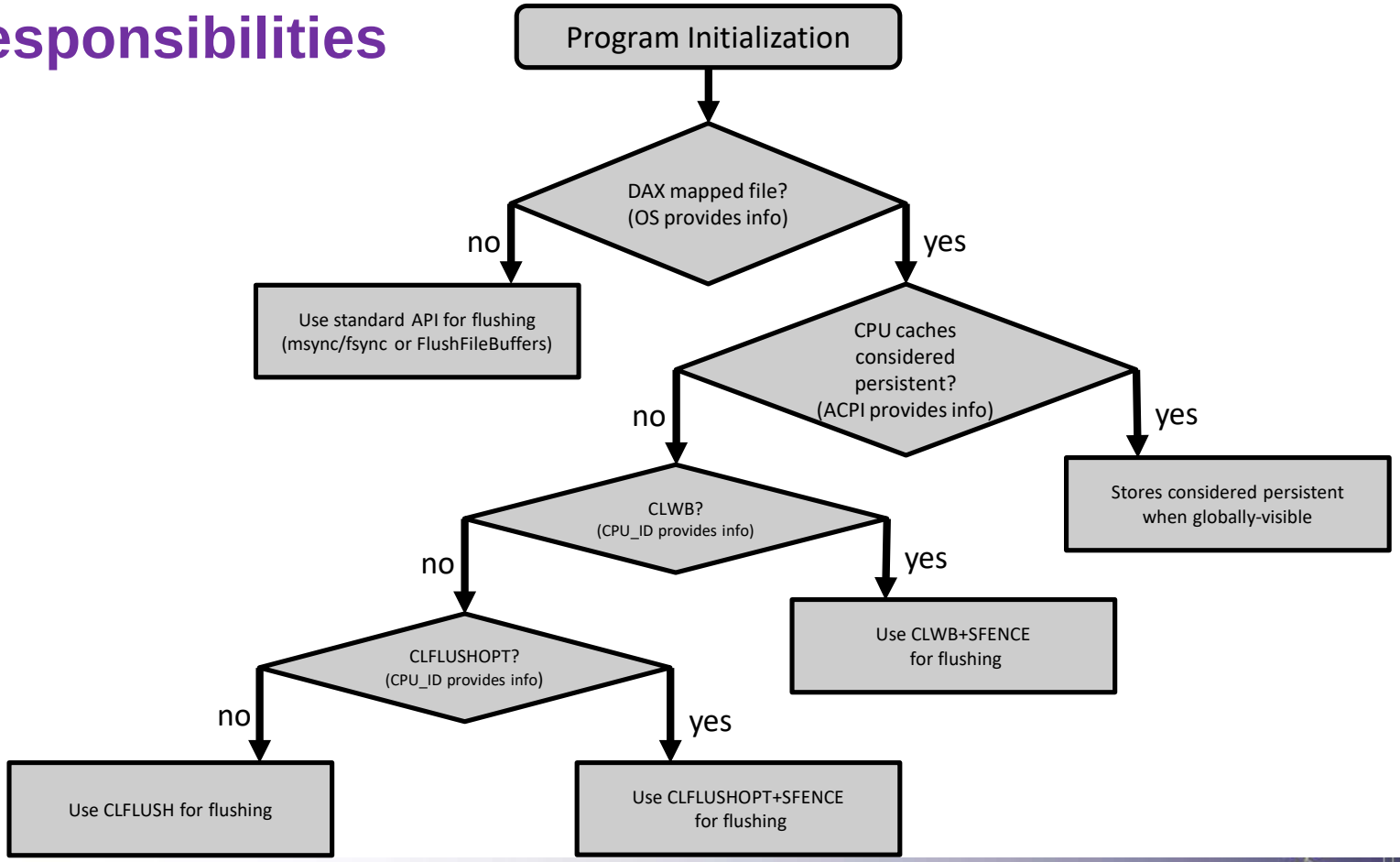
- Communicating with the OS
  - ACPI 6.0+
    - NFIT
    - SMART
  - E820 table
  - HMAT
  - UEFI
    - BTT
- DSMs for communicating with NVDIMMs
  - Not a standard
- CPU Cache Flush, ~~PCOMMIT~~, ADR, eADR, Deep Flush

Intel: How the Hardware Works



Not shown:
MCA
ADR Failure Detection

App Responsibilities



Linux: Exposing Persistent Memory to Apps

- DAX mechanism added (replacing old XIP mechanism)
  - Allowed drivers & file systems to provide direct access
  - ext4 & XFS support upstream

Linux: A Few Surprises That Came Up

- Using general-purpose filesystems
  - pmfs work derailed early on
- Requiring MAP_SYNC
  - Took a long time to arrive at this solution
- Attitude on per-mount DAX versus per-file
- Emerging support for RDMA
- Device DAX

Linux: Device DAX

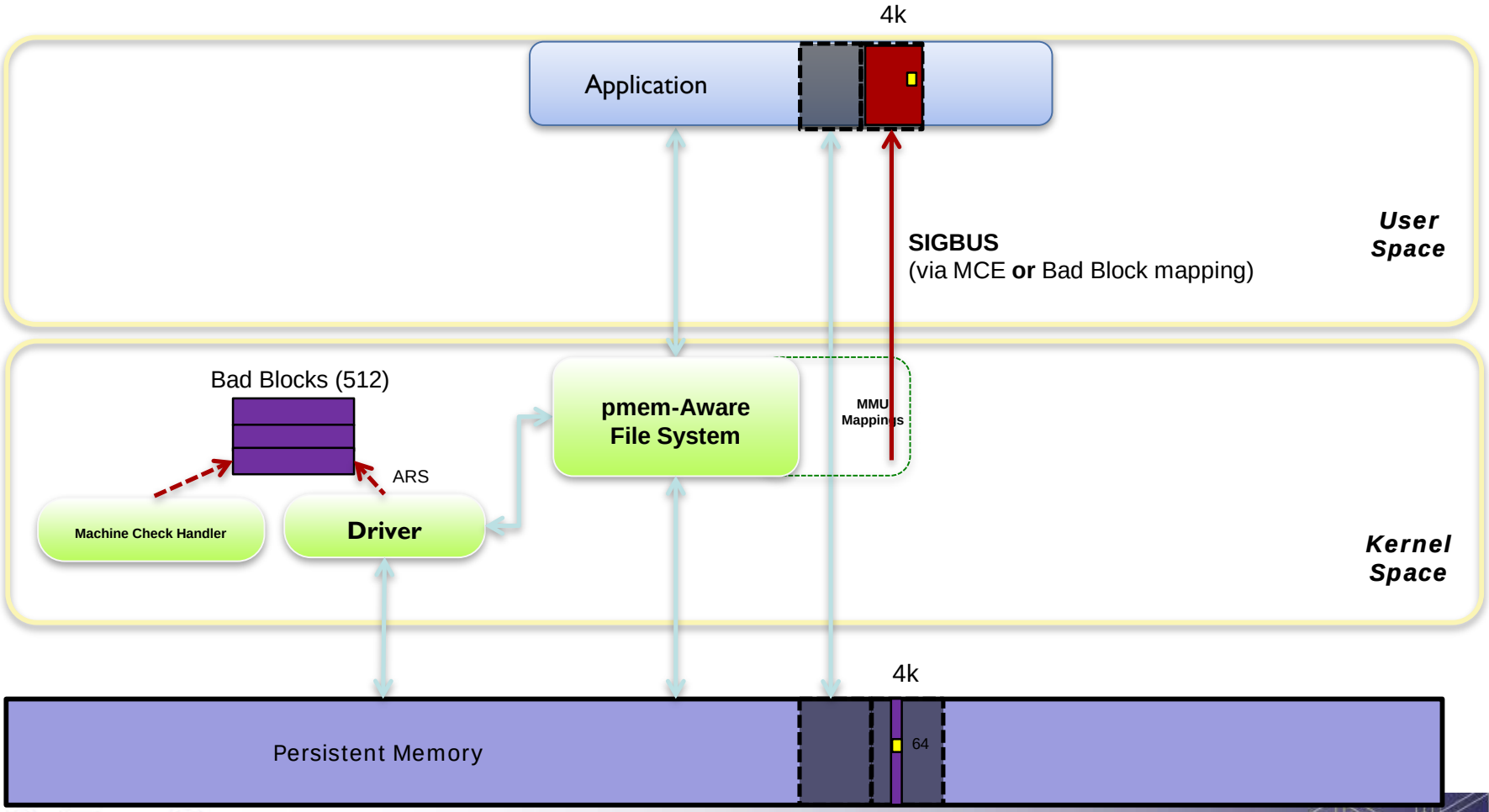
- Doesn't follow the SNIA programming model
- Surprising behavior for app writers:
  - read(2)/write(2)/msync(2) don't work!
  - stat(2) doesn't tell you the size
    - Can't back it up using off-the-shelf tools
    - PMDK library hides as much of this as possible
- Only solution we have for
  - RDMA w/long-lived registrations until WIP finishes
  - Avoiding some minor FS annoyances (for now)
    - ZFOD allocations, app control of large pages

Linux: Deep Flush

- Implemented in driver
- Exposed via sysfs (for Device DAX users)
- FS journal writes use it
- msync()/fsync() invokes it
- Unsafe shutdown detection
  - Left to user space to handle (for now)
  - Some last minute tweaks (permission issues)

Linux: Uncorrectable Error Handling

- Tracked in OS, driver will clear them on block zero
- “mcsafe” version of memcpy
  - not used by any upstream FS yet
  - NOVA did some excellent work in this area
- Apps can discover, catch SIGBUS
  - punch hole/delete file to clear
  - Clearing poison & writing new data NOT atomic
    - PMDK provides one solution to this
- Complication: not safe to clear poison with ISA (yet)



Windows: Implementation

- DAX mechanism added
  - NTFS so far
- User Space flush always safe
  - No need for something like MAP_SYNC
- No equivalent of Device DAX
- No exposure of Deep Flush to user space
- Emerging support for uncorrectables

Virtualization

- Several products announced to the public:
  - Support for pmem programming model in a guest VM
  - Expected to use virtual NFIT table
  - VMware (vSphere 6.7), Hyper-V (announced intentions), KVM (upstream)
- Executive summary:
  - No need for applications to be aware they are in a VM when using pmem
  - Kudos to all these products for making this happen!
- Model (and other specs) silent of hard problems, like:
  - How to detect eADR while allowing live migration
  - Flushing large ranges
  - Handling poison/machine checks in guests

Where We Could've Done Better

- Spec has some inconsistencies
  - Example: arguments to operations like Flush, Clear Error
- Spec has some areas that aren't covered at all
  - Whether CPU caches even need flushing
    - Key to building anything on top of the NVMP programming model
  - How to discover
    - Performance
    - Granularity of operations
    - Known lost data/Incomplete flush on failure
      - (bad block list, unsafe shutdown count, etc.)

Future Work for TWG

- Continue to evolve spec
  - Bring it up to date
  - Add section on current practices
- Continue to evolve rpmem, RAS, transactions
- Continue education on persistent memory