Non-volatile DIMMs: Datapath and Management models

Paul von Behren / Intel Corporation

Introduction to NVDIMM hardware

Hardware supporting persistent memory (pmem)

Yesterday: battery backed RAM

Today: NVDIMMs with DRAM and flash

- On power down, RAM copied to flash; on power up, copy back to RAM

Emerging NVDIMMs: Phase Change Memory, Memristor, many others

- Offer ~ 1000x speed-up over NAND, closer to DRAM

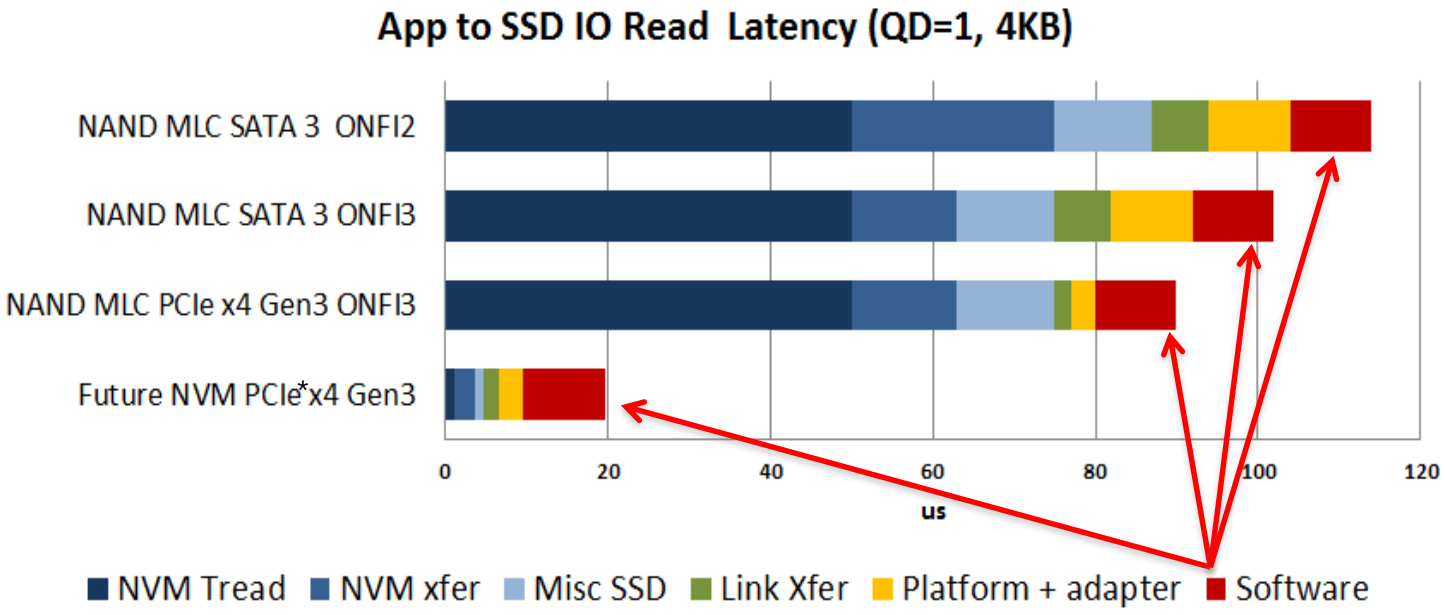
Characteristics as seen by software

- Load/Store (memory instructions) accessible
- Would reasonably stall CPU for a load instruction
- No need for paging (at least not by the OS)

Software access to pmem

Could treat pmem like disks/SSDs

- Existing software works, faster than with flash
- But we still have block stack latency (Intel SSD study)



With Next Generation NVM, hardw

New programming model for byte-addressable persistent memory (pmem)

SNIA goals for datapath programming

Define a programming model for direct access to pmem

- No kernel code in data path, use existing load/store instructions

Use a general approach for different types of pmem hardware

- Built into OSes
- Use existing OS solutions where appropriate
  - E.g., use existing file permissions rather than invent something new

Specify behavior, not a specific API

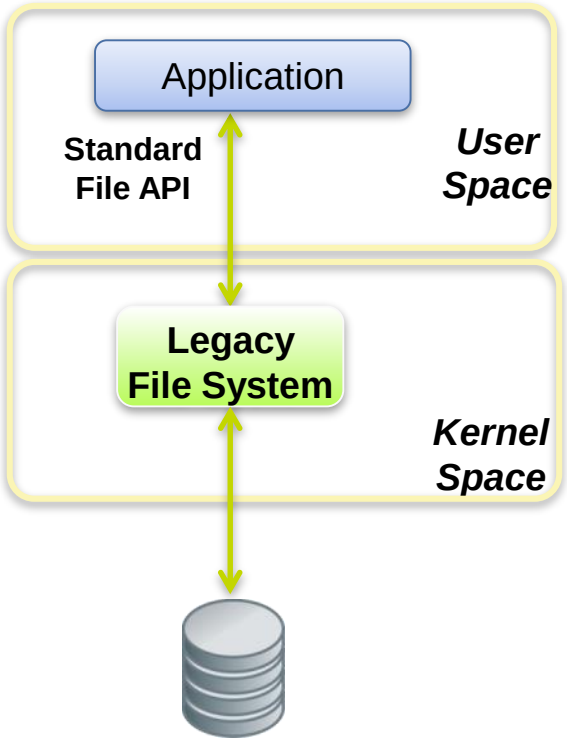
- Allow OS developers to implement APIs appropriate to the OS

Support application developer goals for power-fail safe atomicity

Legacy memory mapped files

Background: memory mapped files backed by block devices

- POSIX mmap(), Windows MapViewOfFile()
- Disk file mapped to virtual memory
- Paged to memory when referenced (by load/store instructions
- msync() flushes dirty pages to disk



Persistent memory programming model

With pmem, no paging between persistence and volatile memory

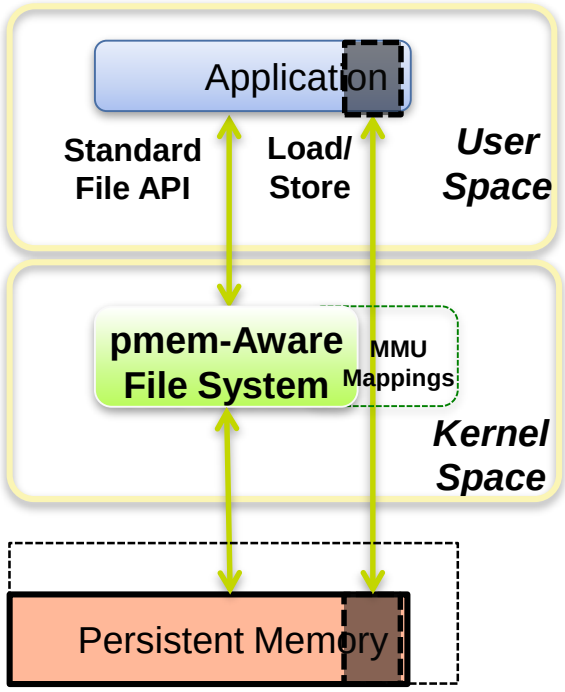
Memory map command causes pmem file to be mapped to app's virtual memory

Sync command flushes CPU cache

Load/Store commands directly access pmem

Standard read/write APIs still work

- Perhaps less performant



How this programming model meets goals

- pmem-aware file system allows pmem to be used by multiple applications, permissions controlled by admin
- Applications have direct access to pmem using load/store instructions
- Applications have control over CPU cache
  - As with block storage, applications (esp. databases) want to control cache – flush frequently for high durability, less frequently for low latency
- Works with existing (C/C++) compilers
  - Potential for friendlier programming model with language extensions, but we don't want to require language and compiler extensions
- Works with existing hardware
  - Tracking emerging pmem HW features, but don't require them

Status of programming model

Included in SNIA *NVM Programming Model* specification

- http://www.snia.org/tech_activities/standards/curr_standards/npm

SNIA NVM Programming Technical Workgroup

- Updating and extending the model
- Researching impact on transaction managers and distributed access to pmem

Supported by Linux kernel 4.2

- New –dax mount option for pmem-aware behavior in ext4 filesystem

User-space libraries building on the pmem programming model

- NVML (pmem.io/nvml) libraries supporting transactions and other usage models
- NVM Direct (https://github.com/oracle/NVM-Direct)

NVDIMM Management model

Impact of NVDIMMs on management tasks

Today's volatile memory

- Discovered by OS and treated by SW as single pool

NVDIMMs introduce "flavors" of memory

- NVDIMMs support persistent byte-addressable memory or block access (i.e., emulate disks/SSDs)
- Memory controllers may allow
  - Configuration of a subset as byte-addressable persistent memory, block, or as volatile
  - QoS-driven provisioning (such as DIMM mirroring)

NVDIMMs (and other advances) add management complexity

- NVDIMM Pool topology (configured as volatile, persistent byte addressable, persistent block addressable, or unused)
  - Allow admin to allocate pmem capacity to pmem-aware applications
- Ability to configure pool topology (allocate from unused to …)
  - QoS provisioning
- Allocation of memory volumes from pools
- Health reporting
- Asynchronous notifications
- Firmware updates
- Physical Topology (NUMA affinity to CPU sockets)

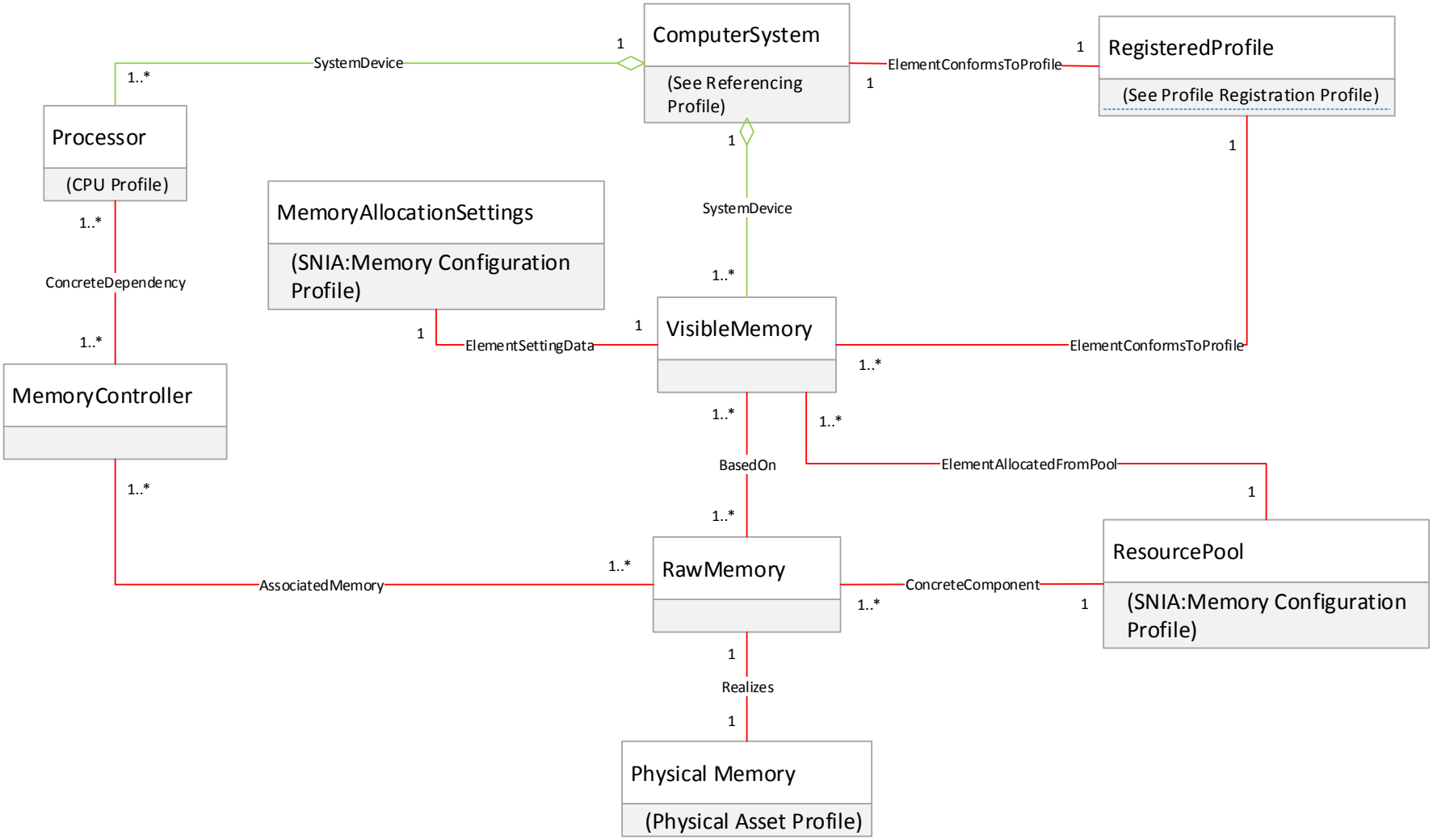
Emerging NVDIMM management standards

CIM profiles from DMTF and SNIA

- DMTF System Memory (DSP1026) existing profile limited to volatile memory
  - Concern in DMTF that extending this profile could break existing clients
- DMTF Multi-type System Memory (DSP1071) (in progress)
  - reports memory regions and associated QoS attributes,
  - plus affinity and visibility of regions to processors
- SNIA SMI-S 1.7 (in progress) includes two memory configuration profiles
  - Memory Configuration Profile
    - Assign NVDIMM capacity to volatile pool or to persistent pools
  - Persistent Memory Configuration Profile
    - Carve persistent pools into namespaces (which SW treats as volumes)
  - Final draft available “soon” (http://www.snia.org/tech_activities/publicreview)

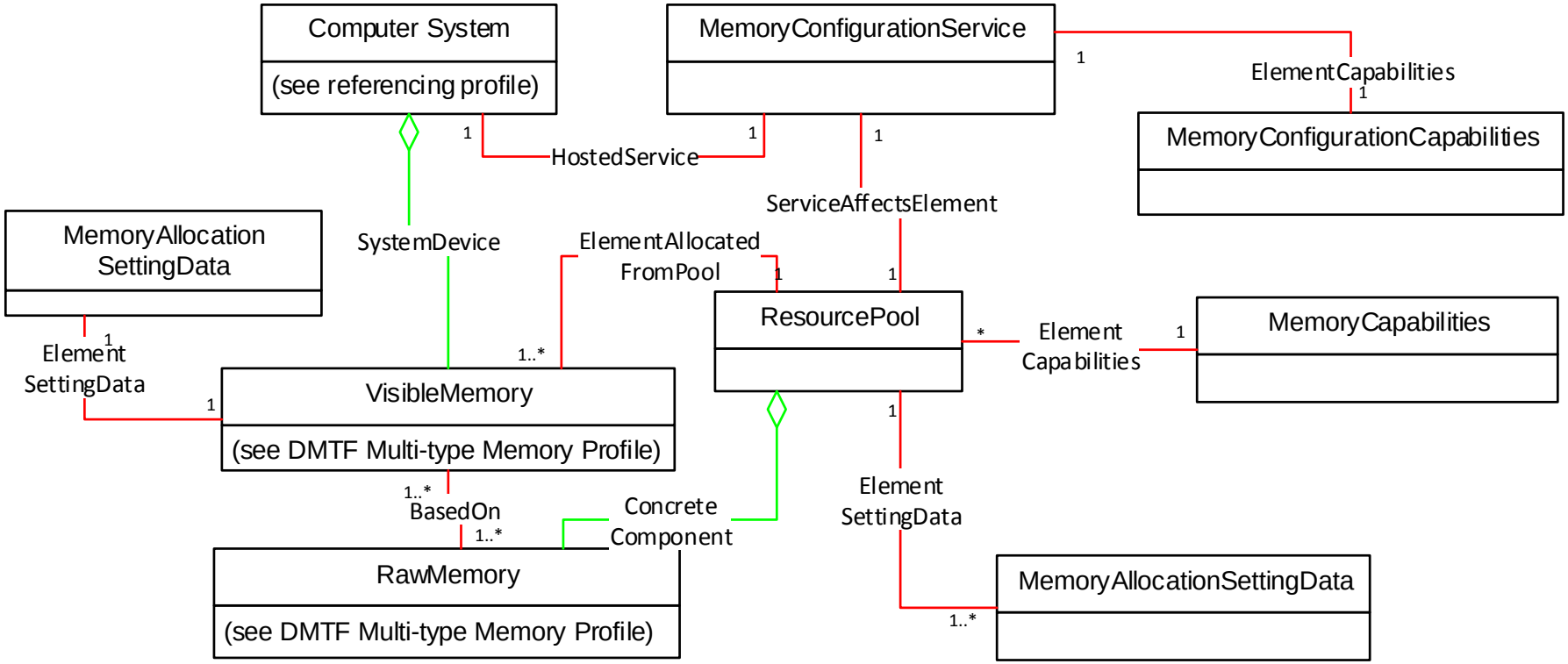
DMTF Multi-Type Memory profile model

See DSP1071 for details



Configure Memory

See SMI-S 1.7 Memory Configuration profile

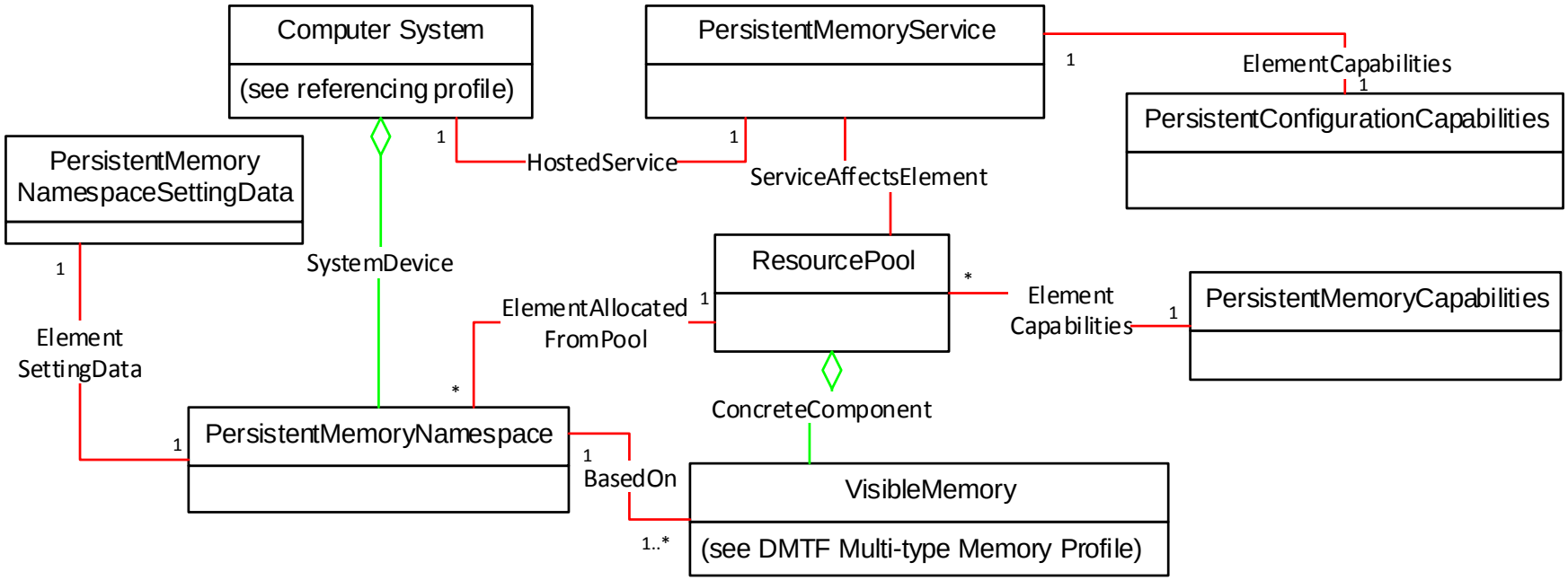


Primary Use Case: allocate visible memory from a pool (of raw memory)

- MemoryConfigurationService iAllocateFromPool() s used to request an allocation.
- The end result of the allocation is the new CIM_VisibleMemory and CIM_MemoryAllocationSettingData instances.
- The CIM_VisibleMemory instance is associated to the CIM_ResourcePool by an ElementAllocatedFromPool association.

Configure Persistent Memory

See SMI-S 1.7 Persistent Memory Configuration profile



Primary Use Case: Create a persistent namespace

- Enumerate instances of MemoryResource, find a pool that has sufficient remaining capacity and who's associated CIM_PersistentMemoryCapabilities indicates support for the desired quality of service.
- Follow the CIM_ServiceAffectsElement association to the PersistentMemoryService instance that handles the selected pool.
- Utilize the AllocateFromPool() extrinsic, specify pool, desired size, quality of service parameters.

Takeaways

- NVDIMM are available now
  - and will be more and more common in the future
  - will offer more configuration options
- Operating system support for NVDIMMs is emerging
- Applications are being rewritten to utilize NVDIMMs where available
- Platform vendors are planning NVDIMM configuration tools, health and fault monitoring, topology reporting
- Some NVDIMM technologies expected to support dynamic configuration
  - E.g., automation/orchestration that can change balance of volatile to persistent memory in response to workload changes

paul.von.behren@intel.com