OPENFABRICS ALLIANCE

14th ANNUAL WORKSHOP 2018

PERSISTENT MEMORY PROGRAMMING

THE REMOTE ACCESS PERSPECTIVE

Tom Talpey, Architect

Microsoft

April 10, 2018

Microsoft

SNIA. SSSI | SOLID STATE STORAGE

OUTLINE

- **SNIA NVMP Programming Model**
  - PMEM Remote Access considerations
- **New RDMA Semantics**
  - Extensions in support of PMEM at minimum latency
- **Windows PMEM Support**
  - Support for PMEM, and future possibilities

OPENFABRICS ALLIANCE

SNIA NVMP PROGRAMMING MODEL

PERSISTENT MEMORY VOLUME AND FILE

Includes Block, File, Volume and Persistent Memory (PM) File

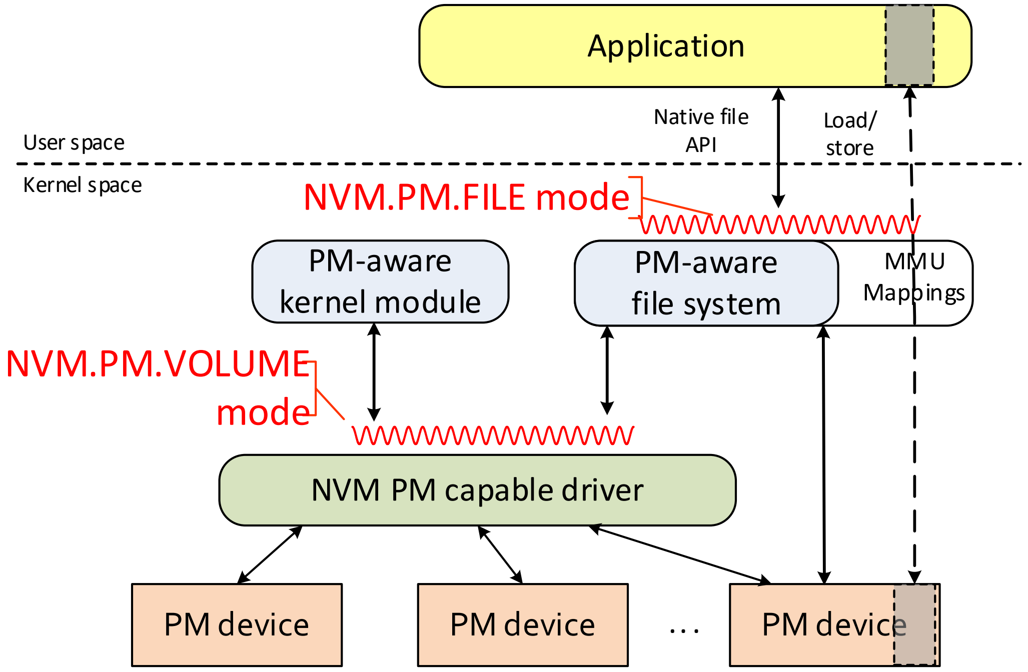
Use with memory-like NVM

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



Memory Mapping in NVM.PM.FILE mode enables direct access to persistent memory using CPU instructions

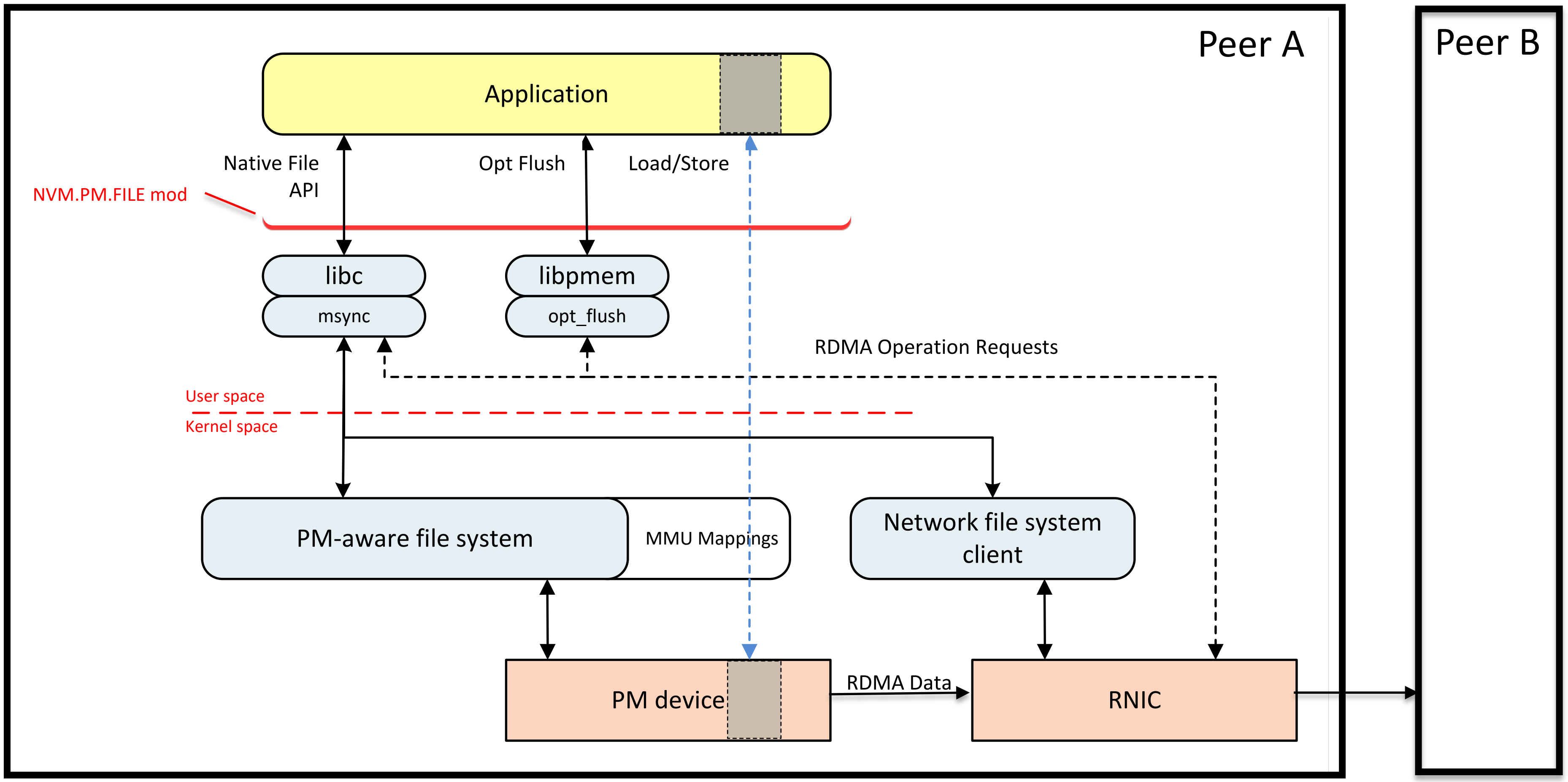
MORE ON MAP AND SYNC

Sync does not guarantee order

- **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
  - Async Flush / Drain – Optimized flush with later completion wait
- **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

REMOTE ACCESS FOR HA SOFTWARE MODEL

RDMA for HA During msync or opt_flush

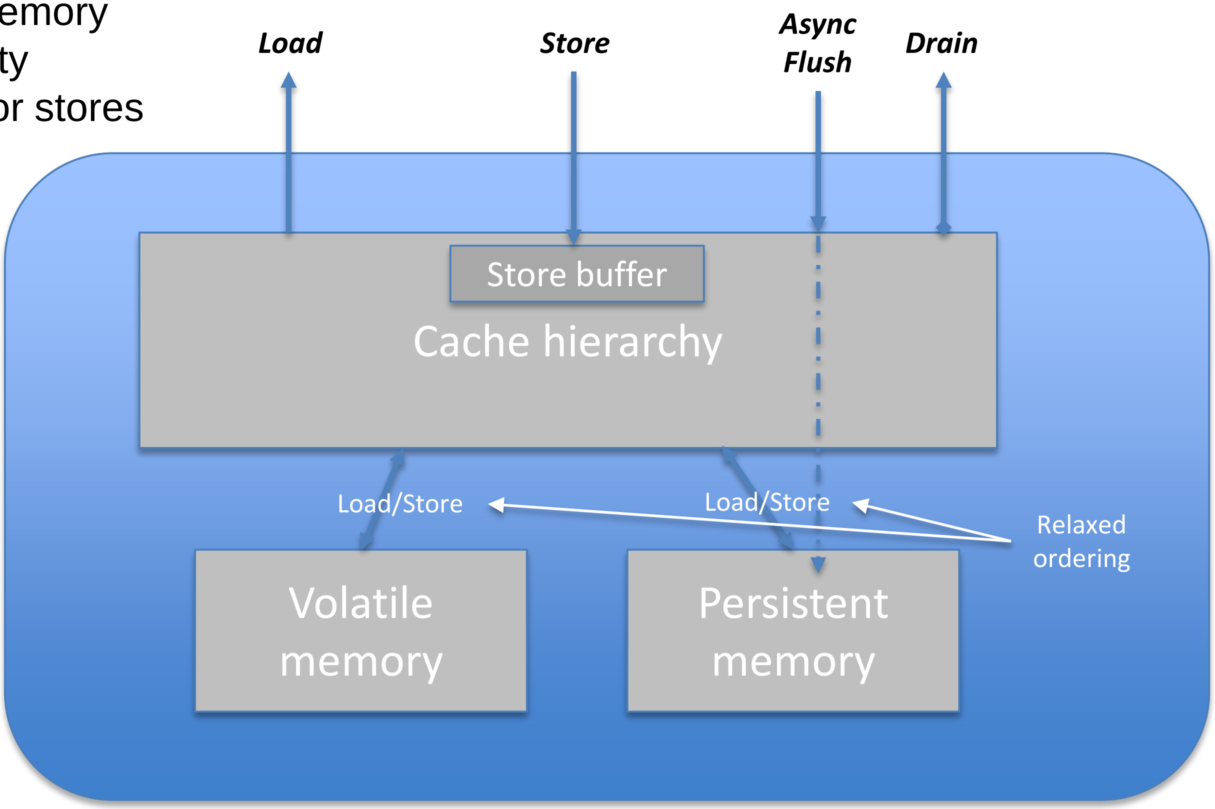


ASYNCHRONOUS FLUSH

- **In process of definition in SNIA NVMP TWG**
- **AsyncFlush() is OptimizedFlush() which does not wait for drain**
  - Local X86 analogue: clflush/clflushopt
- **Subsequent Drain() call required**
  - Local X86 analogue: sfence
- **Anticipated use by overlapped (parallel) programming**
- **Especially: Remote programming**

ASYNCHRONOUS FLUSH SEMANTICS

- **Async Flush defines *ordering of durability*, and nothing more**
- **Consistency (visibility) vs Persistency (durability)**
- **Relaxed order "memory API"**
  - Load/Store to visibility
    - Relaxed order to memory
  - Flush/Drain to durability
    - Ordered only to prior stores



LOCAL IMPLEMENTATION

- Load => load
- Store => store
- Async flush => pmem_flush
  - x86-64: clflush to region(s) which are target of flush
- Drain => pmem_drain
  - x86-64: sfence to wait for store pipeline(s) to complete

REMOTE IMPLEMENTATION

- Load => load
- Store => store
- **Async flush => RDMA Write**
  - RDMA Write from region(s) which are target of flush
    - Note: page-fault model (at time of load/store) far too expensive and imprecise
  - Efficiently leverages relaxed ordering semantic to defer RDMA Write
    - NVMP Interface facilitates this by providing region list
    - Async method allows application to invoke as overlapped “giddy-up”
- **Drain => RDMA Flush**
  - RNIC RDMA Flush to all queuepairs written-to by the RDMA Write(s)
  - Note: region or segment scope under discussion in future
  - See following subsection for RDMA Flush definition

OPENFABRICS
ALLIANCE

NEW RDMA SEMANTICS

RDMA PROTOCOLS

- **Need a <u>remote</u> guarantee of <u>Durability</u>**
  - In support of OptimizedFlush()/ AsyncFlush()
- **RDMA Write alone is not sufficient for this semantic**
  - Guarantees remote <u>visibility</u>, but not final memory residency (durability)
    - Similar to CPU cache => memory semantic
  - Does not guarantee ordered delivery w.r.t. other operations (“post” only)
- **An extension is desired**
  - Proposed “RDMA Commit”, a.k.a. “RDMA Flush”
- **Executes as non-posted operation**
  - Ordered, Flow controlled, acknowledged (like RDMA Read or Atomic)
  - Initiator requests specific byte ranges to be made durable
  - Responder acknowledges only when durability complete
  - Strong consensus on these basics

RDMA FLUSH (CONCEPT)

- **New wire operation, and new verb**
- **Implementable in iWARP and IB/RoCE**
- **Initiating RNIC provides region, other commit parameters**
  - Under control of local API at client/initiator
- **Receiving RNIC queues operation to proceed in-order**
  - Like non-posted RDMA Read or Atomic processing
  - Subject to flow control and ordering
- **RNIC pushes pending writes to targeted region(s)**
  - Alternatively, NIC may simply opt to push all writes
- **RNIC performs any necessary PM commit**
  - Possibly interrupting CPU in current architectures
  - Future (highly desirable to avoid latency) perform via PCIe
- **RNIC responds when durability is assured**

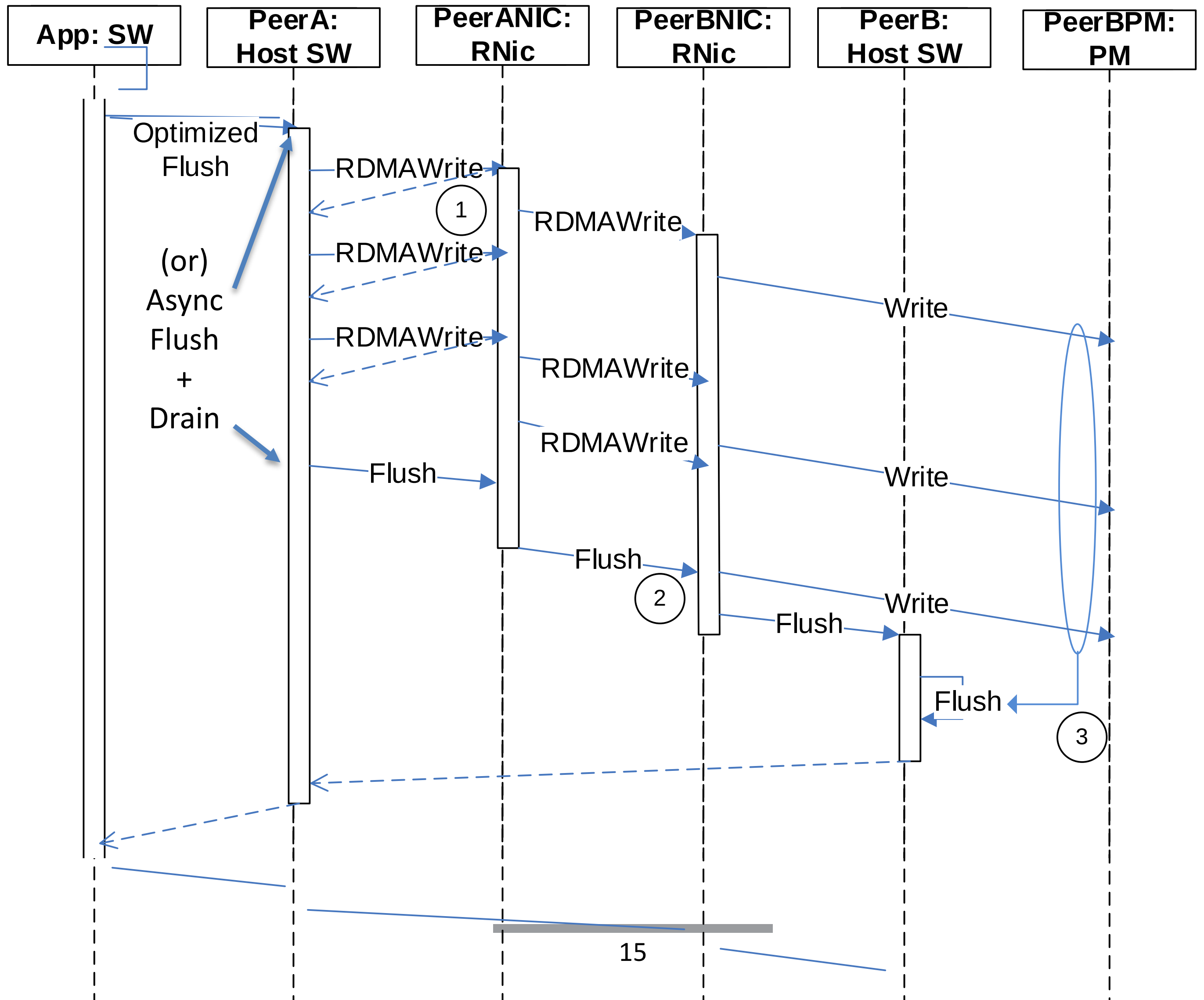
STANDARDS DISCUSSION

- **Broad IBTA agreement on RDMA Flush approach**
- **Discussion continues on semantic details**
  - Per-segment, per-region or per-connection?
  - Implications on the API (SNIA NVMP TWG)
    - Important to converge
- **Ongoing discussion on “non-posted write”**
  - Provides write-after-flush semantic
  - Useful for efficient log write, transactions, etc
    - While avoiding pipeline bubbles

- **PCI support for Flush from RDMA adapter**
  - To Memory, CPU, PCI Root, PM device, PCIe device, …
  - Avoids CPU interaction
  - Supports strong data persistency model
- **Possibly achievable without full-blown PCIe extensions**
  - “Hints”, magic platform register(s), etc
  - Accelerate adoption with platform-specific support

REMOTE ACCESS FOR HA LADDER DIAGRAM

Remote Optimized Flush



EXAMPLE: LOG WRITER (FILESYSTEM)

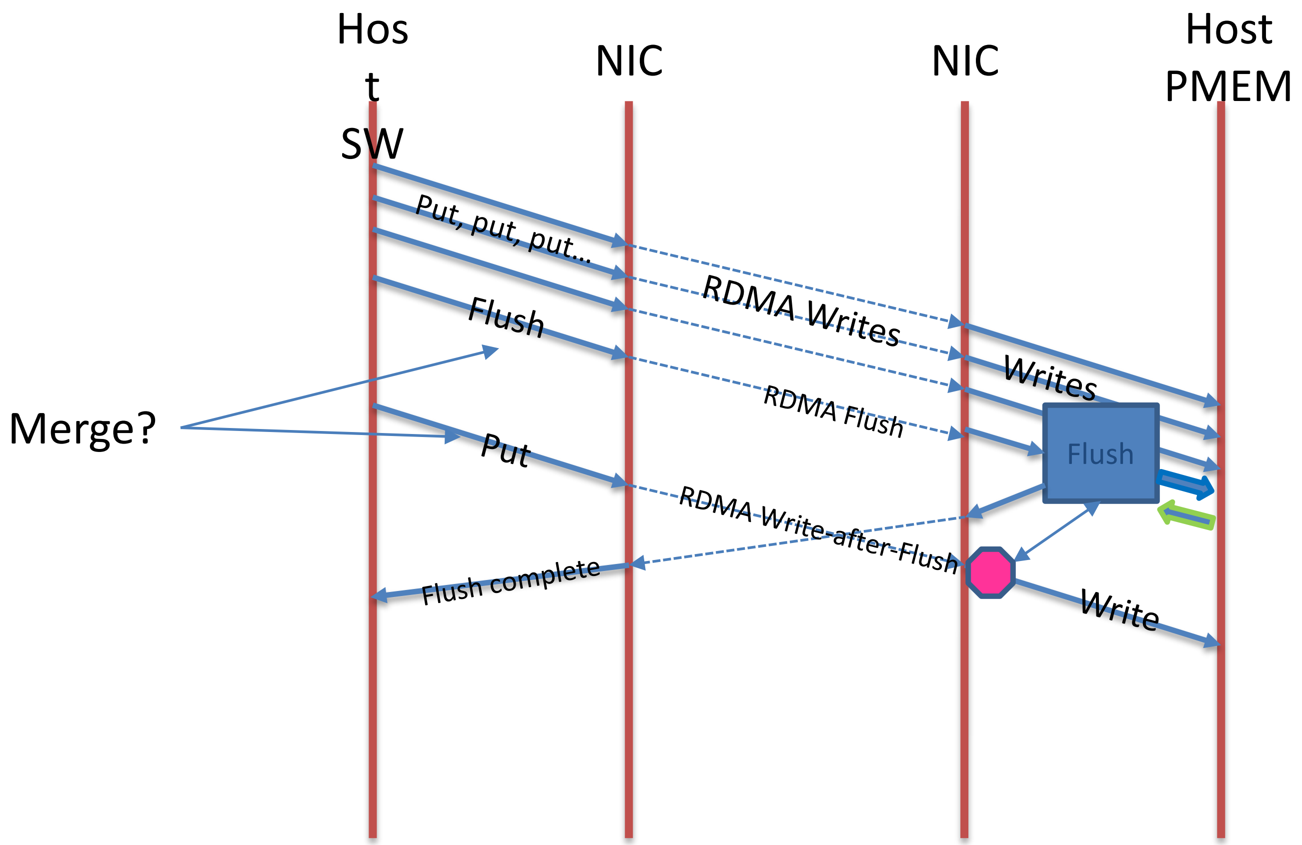
- **For (ever)**

  { Write log record, Commit }, { Write log pointer (, Commit) }

  - Latency is critical
  - Log pointer cannot be placed until log record is successfully made durable
    - Log pointer is the validity indicator for the log record
    - Transaction model
  - Log records are eventually retired, buffer is circular
- **Protocol implications:**
  - Must wait for first commit (and possibly the second)
  - Wait introduces a pipeline bubble – very bad for throughput and overall latency
  - Desire an ordering between Commit and second Write to avoid this
- **Proposed solution: “Non-posted write”**
  - Special Write which executes “like a Read”
    - Ordered with other non-posted operations (esp. Flush)
    - Specific size and alignment restrictions
  - Being discussed in IBTA
- **Not yet expressed in SNIA NVMP interface**

WRITES, FLUSH AND WRITE-AFTER-FLUSH

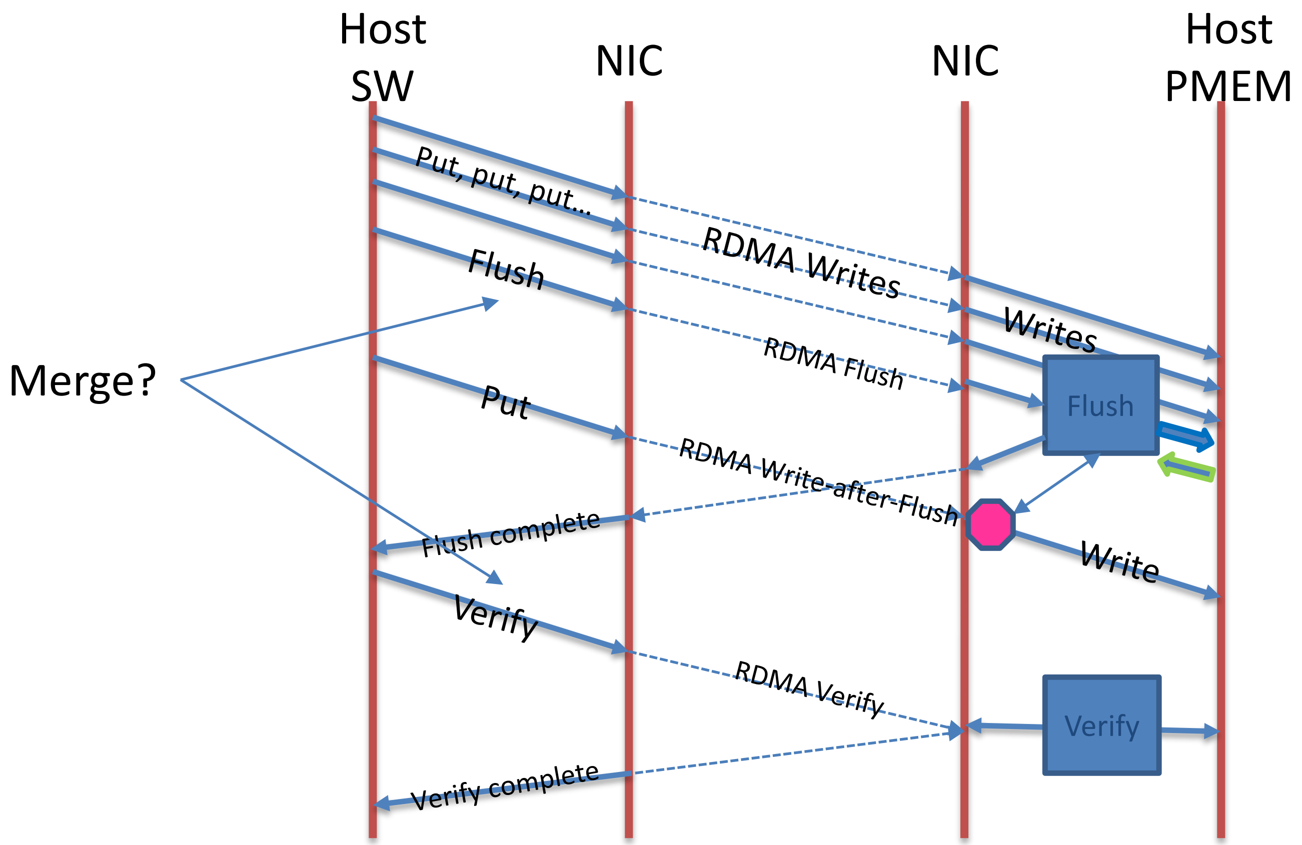
"Non-posted Write"



RDMA “VERIFY”

- **After a RDMA Write + RDMA Flush, how does the initiator know the data is intact?**
  - Or in case of failure, which data is **not** intact?
  - Reading data, signaling remote CPU inefficient and impractical
- **Add integrity hashes to a new operation**
  - Or, possibly, piggybacked on Flush (which would mean only one protocol change)
  - Note, not unlike SCSI T10 DIF
- **Hashing implemented in RNIC or Library “implementation”**
  - Algorithm(s) to be negotiated by upper layers
  - Which could be in
    - Platform, e.g. storage device itself
    - RNIC hardware/firmware, e.g. RNIC performs readback/integrity computation
    - Other hardware on target platform, e.g. chipset, memory controller
    - Software, e.g. target CPU
- **Roughly mapped to SNIA NVMP TWG OptimizedFlushAndVerify()**

WRITE, FLUSH AND VERIFY

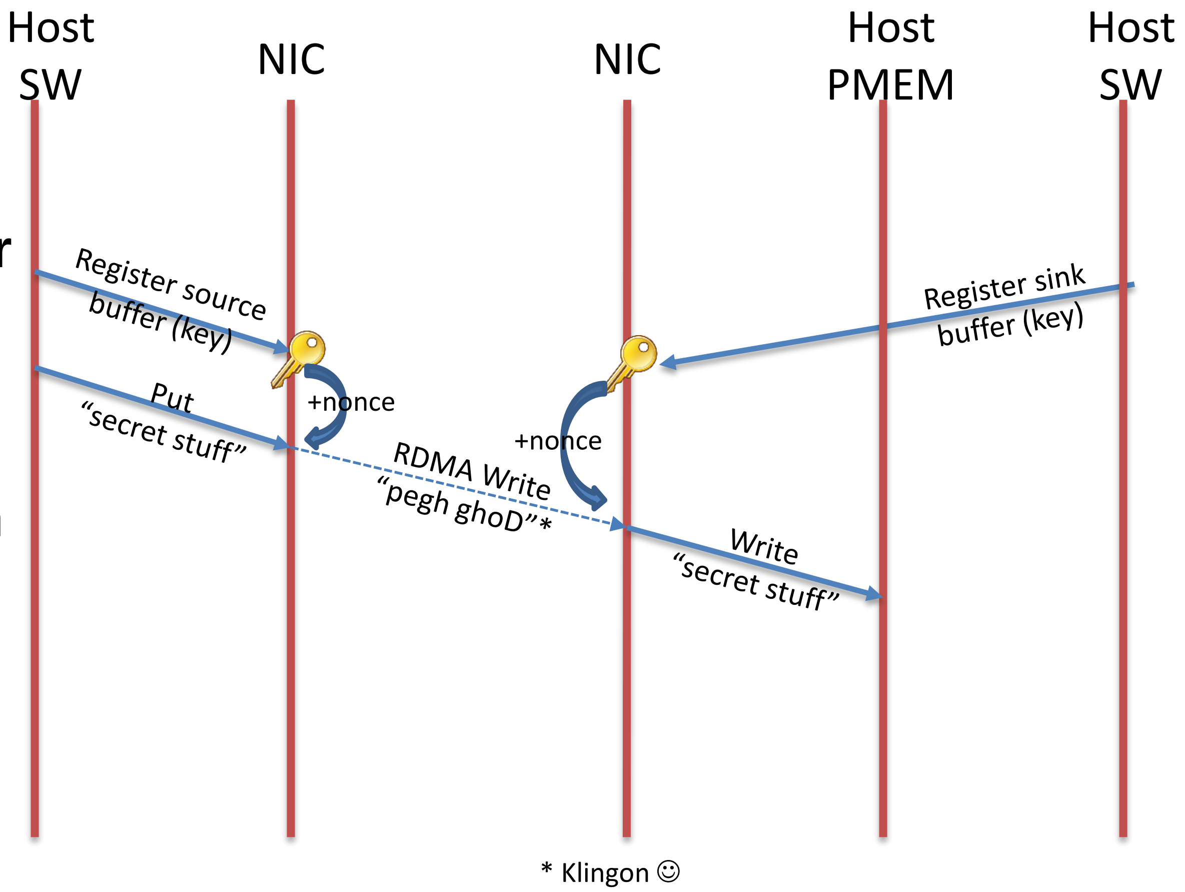


PRIVACY

- **Upper layers protect their send/receive messages today**
- **But RDMA direct transfers are not protected**
  - No RDMA encryption standard
- **Desire to protect <u>User Data </u>with <u>User Key</u>**
  - Not global, machine or connection key!
  - Rules out IPsec, TLS, DTLS
- **Why not just use the on-disk crypto?**
  - Typically a block cipher, requiring block not byte access
  - No integrity – requires double computation
- **Authenticated stream cipher (e.g. AES-CCM/GCM as used by SMB3)**
  - Provides wire privacy *and* integrity, efficiently
    - (not at-rest – still need RDMA Verify)
  - Arbitrary number of bytes per transfer
  - Shares cipher and keying with upper layer
  - But, how to plumb key into RDMA NIC message processing?
- **Enhance RDMA Memory Regions**
  - Which does not require RDMA protocol change!

RDMA WRITE ENCRYPTION

- **Extend MR verb and NIC TPT to include key**
  - Handle = Register(PD, buffer, mode, **key**)
- **Keys held by upper layer, user policy, and passed down to NIC**
- **NIC uses key when reading or writing each region**



CIPHER HOUSEKEEPING

- **Authenticated ciphers typically employ nonces (e.g. AES-GCM)**
  - Same {key,nonce} pair used at each end to encrypt/decrypt each message
  - Never reuse nonce for different payload!
    - Upper layer must coordinate nonce usage with RDMA layer
    - RDMA layer must consider when retrying
  - NIC may derive nonce sequence from RDMA connection
    - E.g. from RDMA msn. Not from the MR!
  - Alternatively, prepend/append to data buffer
  - Upper layer consumes nonce space, too
- **Re-keying necessary when nonce space exhausted!**
  - Nonces are large (SMB3 employs 11 bytes for GCM), but require careful management and sharing with ULP
  - Key management is upper layer responsibility, as it should be

RDMA QOS

- **Upper layers have no trouble saturating even 100Gb networks with RDMA today**
- **Memory can sink writes at least this fast**

➢ **Networks will rapidly congest**
  - Rate control, fairness and QoS are required in the RDMA NIC

- **Simplistically, bandwidth limiting**
- **More sophisticated approach desirable**
  - Classification/end-to-end QoS techniques
    - SDC2014 presentation (“resources” slide)
  - Software-Defined Network techniques
    - Generic Flow Table-based policy
- **Existing support in many Enterprise-class NICs**

OPENFABRICS ALLIANCE

WINDOWS PMEM SUPPORT

WINDOWS PERSISTENT MEMORY

- **Persistent Memory is supported in Windows**
  - PMEM support is **foundational** in Windows going forward
- **Support for JEDEC-defined NVDIMM-N devices available from**
  - Windows Server 2016
  - Windows 10 (Anniversary Update – Fall 2016)
- **PMDK is fully supported by Windows**
  - Code and binaries available on Github (Resources slide)

WINDOWS PMEM ACCESS

DAX mode (at right)

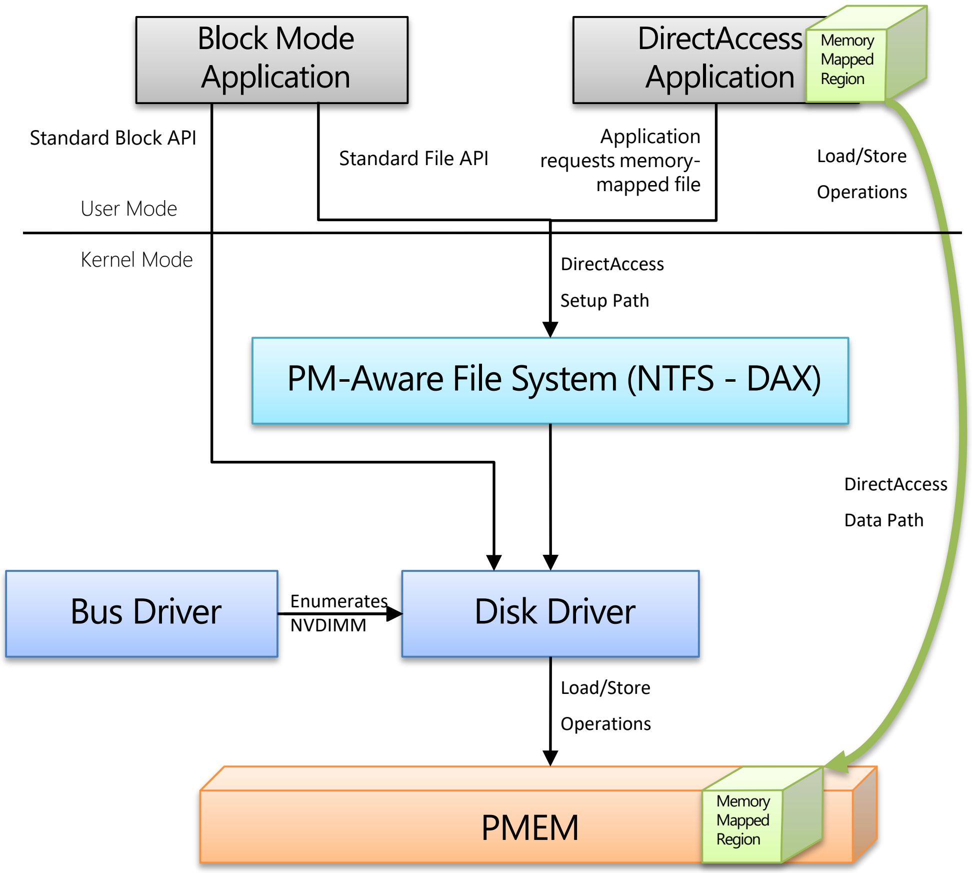
- Direct Access app obtains direct access to PMEM via existing memory-mapping semantics
- Updates directly modify PMEM, Storage Stack not involved (Green data path)
- True device performance (no software overhead)
- Byte-Addressable
- “NVM.PM.FILE”
- Available since Windows 10 Anniversary Update / Server 2016

Block mode (at left)

- App continues to use traditional kernel-mediated APIs (File and Block)
- Full compatibility with existing apps
- Improved performance via underlying hardware
- “NVM.PM.VOLUME”
- Available since Windows 10 Anniversary Update / Server 2016

PMEM-Aware (later slide)

- “PMEM-aware” open coded
- Native “Rtl” API (Windows 10 Spring 2017 and future Server)
- PMDK open source



WINDOWS NATIVE PMEM APIS

- **Available from both user and kernel modes**
- **Flush to durability in optimal fashion for each hardware architecture**
- **Supported in Windows 10 1703 (Spring 2017) and Windows Server (future)**

- **RtlGetNonVolatileToken**
  https://msdn.microsoft.com/en-us/library/windows/hardware/mt800701.aspx
- **RtlWriteNonVolatileMemory**
  https://msdn.microsoft.com/en-us/library/windows/hardware/mt800702.aspx
- **RtlFlushNonVolatileMemory/RtlFlushNonVolatileMemoryRanges**
  https://msdn.microsoft.com/en-us/library/windows/hardware/mt800698.aspx
  https://msdn.microsoft.com/en-us/library/windows/hardware/mt800699.aspx
- **RtlDrainNonVolatileFlush**
  https://msdn.microsoft.com/en-us/library/windows/hardware/mt800697.aspx
- **RtlFreeNonVolatileToken**
  https://msdn.microsoft.com/en-us/library/windows/hardware/mt800700.aspx

USING DAX IN WINDOWS

DAX Volume Creation

→`Command prompt: Format n: /dax /q`

→`Powershell: Format-Volume -DriveLetter n -IsDAX $true`

DAX Volume Identification

Is it a DAX volume?

→**call** `GetVolumeInformation("C:\", ...)`

→**check** `lpFileSystemFlags` **for** `FILE_DAX_VOLUME (0x20000000)`

Is the file on a DAX volume?

→**call** `GetVolumeInformationByHandleW(hFile, ...)`

→**check** `lpFileSystemFlags` **for** `FILE_DAX_VOLUME (0x20000000)`

IMPLICIT FLUSH CODE EXAMPLE

Memory Mapping:

```
HANDLE hMapping = CreateFileMapping(hFile, NULL, PAGE_READWRITE, 0, 0, NULL);
LPVOID baseAddress = MapViewOfFile(hMapping, FILE_MAP_WRITE, 0, 0, size);
memcpy(baseAddress + writeOffset, dataBuffer, ioSize);
FlushViewOfFile(baseAddress, 0);
```

OR … use non-temporal instructions for NVDIMM-N devices:

```
HANDLE hMapping = CreateFileMapping(hFile, NULL, PAGE_READWRITE, 0, 0, NULL);
LPVOID baseAddress = MapViewOfFile(hMapping, FILE_MAP_WRITE, 0, 0, size);
RtlCopyMemoryNonTemporal(baseAddress + writeOffset, dataBuffer, ioSize );
```

EXPLICIT FLUSH CODE EXAMPLE

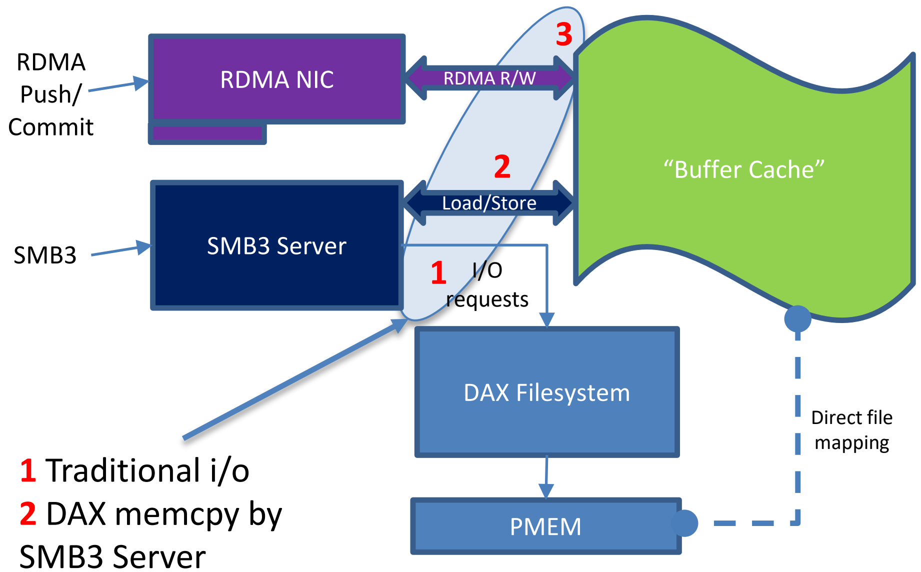
```
HANDLE hMapping = CreateFileMapping(hFile, NULL, PAGE_READWRITE, 0, 0, NULL);
LPVOID baseAddress = MapViewOfFile(hMapping, FILE_MAP_WRITE, 0, 0, size);
PVOID token = RtlGetNonVolatileToken(baseAddress, size, &token);
for (;;)
{
   *(int *)baseAddress = random();         // Write to PMEM
   RtlFlushNonVolatileMemory(token, baseAddress, sizeof(int), 0);   // flags=0
   baseAddress += sizeof(int);
}
RtlFreeNonVolatileToken(token);
CloseHandle(hMapping);
```

ASYNCHRONOUS FLUSH CODE EXAMPLE

```
HANDLE hMapping = CreateFileMapping(hFile, NULL, PAGE_READWRITE, 0, 0, NULL);
LPVOID baseAddress = MapViewOfFile(hMapping, FILE_MAP_WRITE, 0, 0, size);
PVOID token = RtlGetNonVolatileToken(baseAddress, size, &token);
for (;;)
{
   *baseAddress = random();           // Write to PMEM
   RtlFlushNonVolatileMemory(token, baseAddress, sizeof(int),
        FLUSH_NV_MEMORY_IN_FLAG_NO_DRAIN );
   baseAddress += sizeof(int);
   // do more stuff
   RtlDrainNonVolatileFlush(token);   // Wait for prior Flush
}
RtlFreeNonVolatileToken(token);
CloseHandle(hMapping);
```

EXAMPLE: GOING REMOTE – SMB3

- **SMB3 RDMA and "Push Mode"**
- **SMB3 Server may implement DAX-mode direct mapping (2)**
  - Reduced server overhead
- **RDMA NIC may implement Flush extension (3)**
  - Enables zero-copy *remote* read/write/flush to DAX file
  - Ultra-low latency and overhead
- **2, 3 can be enabled *before* RDMA extensions become available, with slight extra cost**



1 Traditional i/o
2 DAX memcpy by SMB3 Server
3 Push Mode direct from RDMA NIC

RESOURCES

- **SNIA NVM Programming TWG:**
  - http://www.snia.org/forums/sssi/nvmp
  - NVMP 1.2
    https://www.snia.org/sites/default/files/SDC/2017/presentations/Solid_State_Stor_NVM_PM_NVDIMM/Talpey_Tom_Microsoft%20_SNIA_NVM_Programming_Model_V%201.2_and_Beyond.pdf

- **SNIA Storage Developers:**
  - Remote Persistent Memory – With Nothing But Net
    https://www.snia.org/sites/default/files/SDC/2017/presentations/Solid_State_Stor_NVM_PM_NVDIMM/Talpey_Tom_RemotePersistentMemory.pdf
  - Low Latency Remote Storage – A Full-stack View
    https://www.snia.org/sites/default/files/SDC/2016/presentations/persistent_memory/Tom_Talpey_Low_Latency_Remote_Storage_A_Full-stack_View.pdf
  - Storage Quality of Service for Enterprise Workloads
    https://www.snia.org/sites/default/files/TomTalpey_Storage_Quality_Service.pdf

- **Windows PMEM Programming:**
  - https://msdn.microsoft.com/en-us/library/windows/hardware/ff553354.aspx
  - https://github.com/pmem/pmdk
  - https://github.com/pmem/pmdk/releases

- **Open Fabrics Workshop:**
  - Remote Persistent Memory Access - Workload Scenarios and RDMA Semantics
    https://www.openfabrics.org/images/eventpresos/2017presentations/405_RemotePM_TTalpey.pdf

OPENFABRICS ALLIANCE

14th ANNUAL WORKSHOP 2018

THANK YOU