OPENFABRICS ALLIANCE

15th ANNUAL WORKSHOP 2019

RDMA PERSISTENT MEMORY EXTENSIONS

Tom Talpey

Microsoft

[March 20, 2019]

Microsoft

OUTLINE

- **A “top-down” view from application programming model to protocol**

- **SNIA NVMP Programming Model, Remote Access for High Availability**
- **RDMA requirements and extensions**
- **Remote PM workload detailed examples**
- **Next steps**

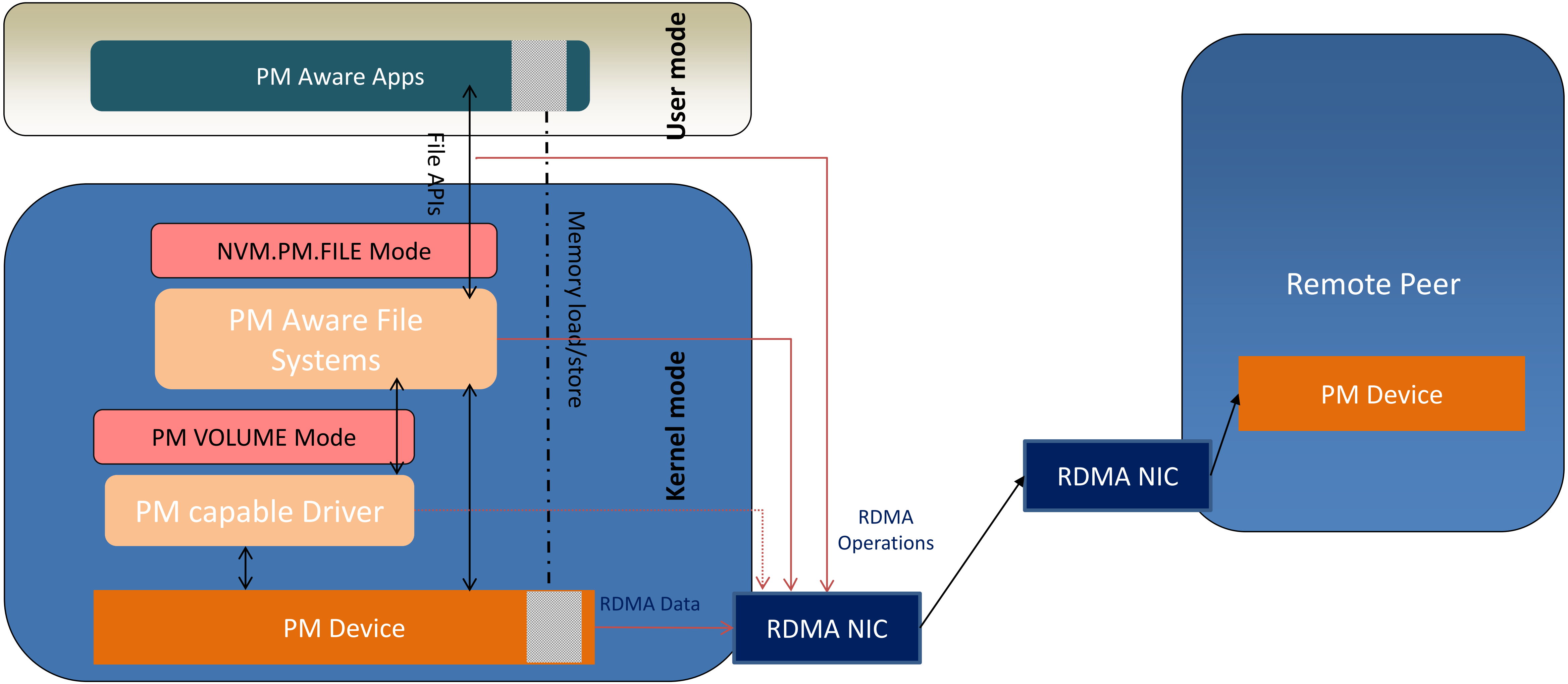
SNIA NVM PROGRAMMING MODEL

- **Version 1.2 current**
  - https://www.snia.org/sites/default/files/technical_work/final/NVMProgrammingModel_v1.2.pdf
- **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

SNIA NVMP REMOTE ACCESS FOR HA

- **History**
  - Original Remote Access for High Availability white paper published 2016
  - Enhanced Remote Access white paper draft V1.1 in public review February 2019
    - http://www.snia.org/publicreview
- **NVM Programming Model Specification 1.3 in development**
  - Updating specification to reflect learning from implementations
  - Incorporate learning from remote access white paper
    - Asynchronous Flush
    - Remote persistence ordering, error handling
- **Remote Access Collaboration with Open Fabrics Alliance OFIWG**
  - OFIWG reviewing and commenting on PM Remote Access for HA V1.1
  - Expand remote access use case enumeration

PERSISTENT MEMORY (PM) MODES, +REMOTE



PM REMOTE ACCESS FOR HA

- **NVMP TWG-developed interface for remote PM**
- **Maximize alignment with local PM interface**
- **Take remote environment into account**
  - Including RDMA semantics and restrictions
- **Analyze the error cases**
  - As always, “the hard part”

- **Directly mappable to RDMA (with extensions):**
  - In NVMP 1.2:
    - OPTIMIZED_FLUSH
    - OPTIMIZED_FLUSH_AND_VERIFY
  - Under discussion (NVMP 1.3):
    - ASYNC_FLUSH (initiates flushing)
    - ASYNC_DRAIN (waits for flush completion, persist fence)
    - Ordering (write-after-flush)
- **Other NVM PM methods remotable via upper layer(s)**

ASYNCHRONOUS FLUSH

- **Optimized Flush semantics**
  - Flush both “pushes” Writes and subsequently performs actual Flush
  - Synchronous - always waits for completion of Flush on each region
- **Problem: RDMA latencies significantly larger than local**
  - Writes, Flush must traverse the network! (as must the Flush response)
  - This magnifies the above impacts of Optimized Flush
- **Solution: “Async Flush”**
  - Separate the two phases of Optimized Flush:
    - ASYNC_FLUSH (push writes to destination, and don’t wait)
    - ASYNC_DRAIN (invokes barrier and wait for writes to reach persistence)
    - Introduces “Ordering Nexus” to formally describe the Flush-Drain barrier fencing
  - Allows overlap, and parallel application processing (efficient middleware implementation)
  - Makes best use of network by “pushing early”
    - A.k.a “Giddy-up”
  - Lowers the latency of eventual Flush
    - Less data remaining to flush: less wait latency
  - Error conditions require careful analysis
    - Subject of NVMP TWG current work

PERSISTENCE VS VISIBILITY

- **Proposed two distinct “flush semantics” (previously one)**
  - Persistence (~current semantic)
  - Visibility, a.k.a. Global Observability (new semantic)
- **Emerging devices support these separately**
  - Visibility does not necessarily imply persistence (volatile cache in front of persistence)
  - Persistence does not necessarily imply visibility (multi-socket or multi-port architectures)
- **Applications desire to control both separately**
  - For efficiency with proper correctness
  - Promptly ensure data is persistent, later make data visible (storage)
  - Promptly ensure data is visible, later make persistent (shared memory)
  - But even if requesting both, Persistence and Visibility are not reached atomically!
    - ➢Don’t try this with Compare and Swap to PM (even locally)
- **Considering exposing this distinction in Programming Model**
  - “Flush type” modifier
  - And also RDMA protocol

UNDER DEVELOPMENT IN SNIA NVMP TWG

- **Scope of flush**
  - Conceptual “store barrier” or “order nexus”
  - Streams of stores, which are later flushed to ensure persistence
  - Flush hints (including remote DEEP_FLUSH)
  - Modeling these in programming interface, with an eye toward protocol
  - Understanding, and guiding, platform and protocol implementation
- **“Consumers of visibility” vs “Consumers of persistence”**
  - Failure semantic for consumers of persistence
- **Assurance of persistence integrity (OPTIMIZED_FLUSH_AND_VERIFY++)**
  - Explicit integrity semantic, as opposed to current Best-effort

REMOTE PERSISTENT MEMORY

REMOTE PM WORKLOADS

- **High Availability (HA)**
  - Resilience, recovery, “RAID-like” properties
  - Replication
  - Scaleout
- **Transactions**
  - Atomicity (failure atomicity)
- **Networked Shared Memory**
  - Including Pub/Sub model
- **And others!**

- **Desire to maintain:**
  - Ultra-low latency (~ +1 RTT w/o pipeline bubbles, i.e. single-digit microsecond total)
  - Programming model compatibility
    - Ideally, transparency!

RDMA FLUSH

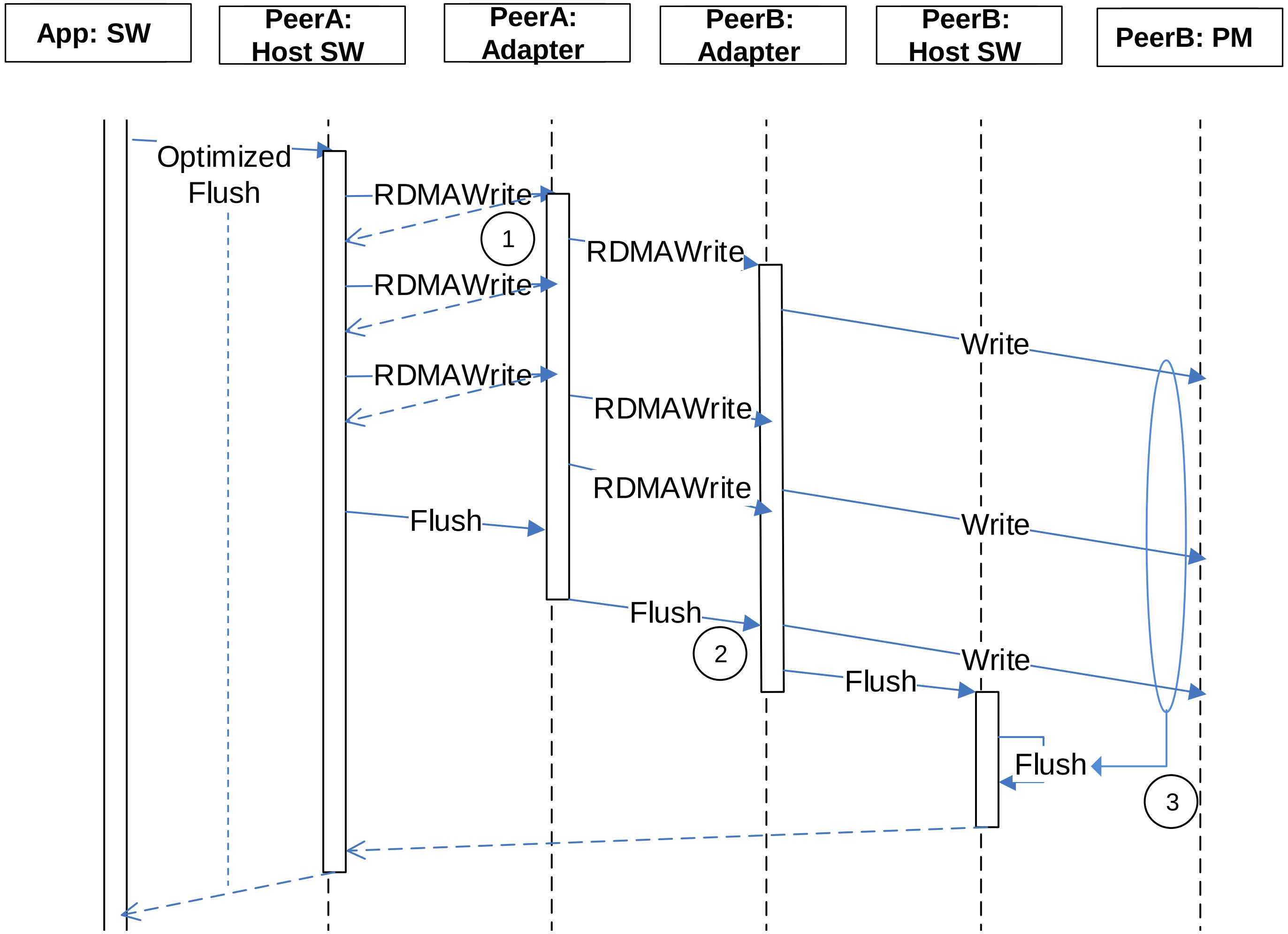
- **New RDMA transport operation**
- **Existing RDMA memory operations remain unchanged**
- **Flush executes like RDMA Read**
  - Ordered, Flow controlled, acknowledged
    - All prior RDMA writes on QP guaranteed to have “pushed” prior to executing Flush
    - IB “non-posted”, iWARP “queued”
  - Requestor specifies byte ranges to be made durable
    - Memory Region range-based {region handle, offset, length}
      - Responder response guarantees specified range is persisted
      - Responder may flush additional bytes based on implementation
  - Single Flush acts upon many prior Writes
  - Responder acknowledges only when persistence complete
    - Connection breaks if error occurs

REMOTE FLUSH (BASIC SEMANTIC)

- **Application view**
- **Optimized Flush invokes library**
  - Library initiates RDMA Write(s) to RPM
  - Library initiates Remote Flush
    - Ordered after prior Writes
    - And blocks for Write+Flush completion
    - Returns (only) when Flush is complete

- **Tricky bits:**

1. **RDMA Writes complete at requestor before stores to PM at responder**
2. **Remote Flush arrives before Writes are executed at responder**
3. **Remote Flush must wait at responder until all Writes are safely in PM**

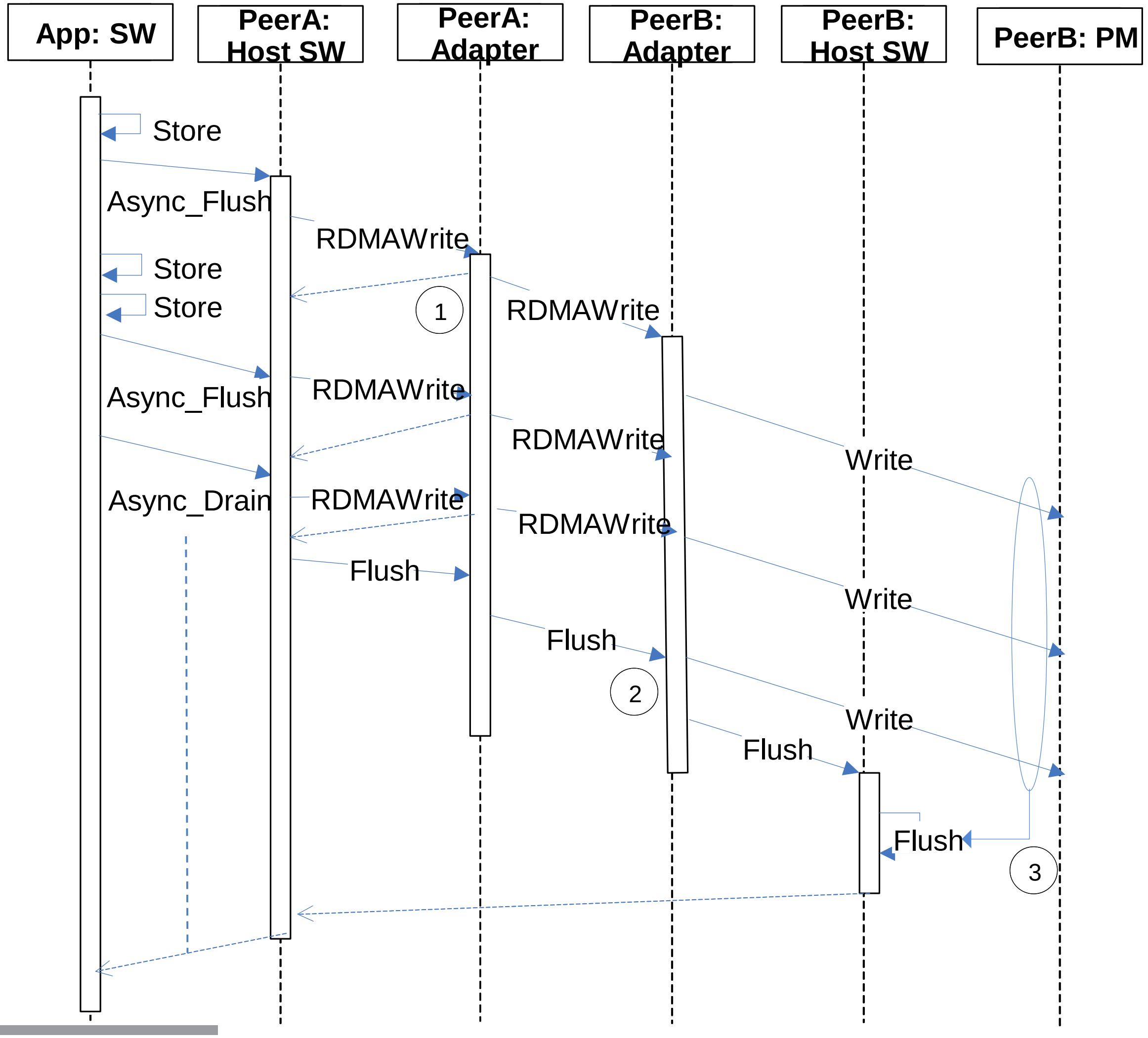


ASYNC FLUSH (ENHANCED SEMANTIC)

- **Application overlapped processing**
- **Async_Flush invokes library**
  - Library initiates RDMA Write(s) to RPM
  - Pipelined - does not wait, immediately returns
- **Additional application processing…**
- **Async_Flush initiates more RDMA Write(s)**
  - Pipelined - does not wait
- **Async_Drain initiates Remote Flush**
  - Library queues RDMA Flush after all prior RDMA Writes
  - Async_Drain completes only after all Writes Flush to PM
    - Note: application may also continue during this processing

- **Tricky bits (1,2,3):**
  - Same as in prior example!
  - But note subtlety:
    - Application **Flush** -> RDMA **Write**
    - Application **Drain** -> RDMA **Flush**

- **RDMA protocol:**
  - ***Same as in prior example!***
  - “Ordering Nexus” is simply the Queue Pair



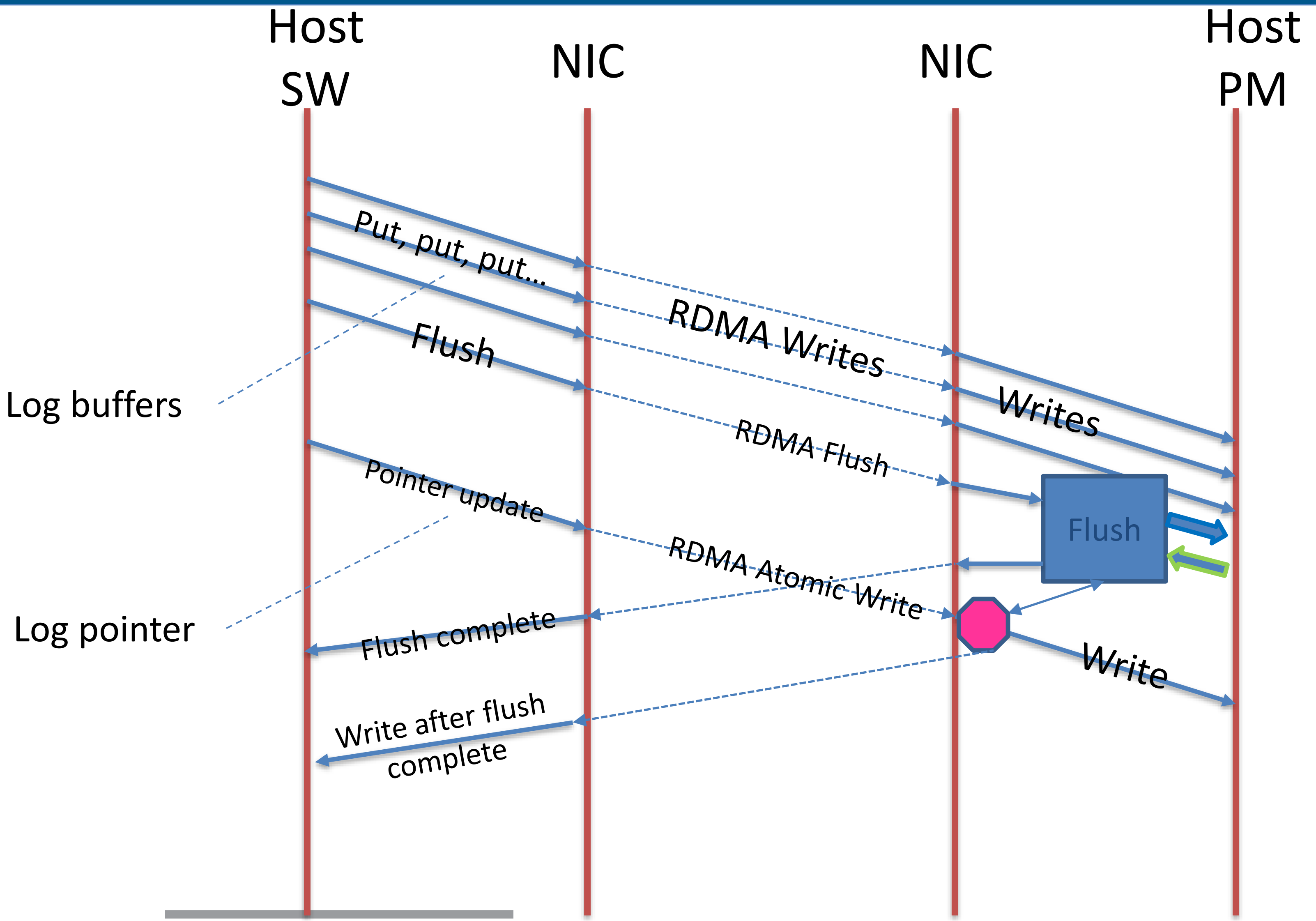
ADDITIONAL DESIRED SEMANTICS

- **Transactional write**
  - Atomically place 8-byte sized, 8-byte aligned data
  - With ordering guarantee to eliminate pipeline bubble(s)
- **Integrity**
  - Compute-the-hash
  - In support of Optimized Flush and Verify
  - Enhanced flush types (Deep Flush)
- **Security**
  - Encrypt on wire / at rest
  - Possible without protocol extension

- **RDMA support for these under discussion**
  - SNIA NVM Programming TWG
  - IBTA
  - IETF

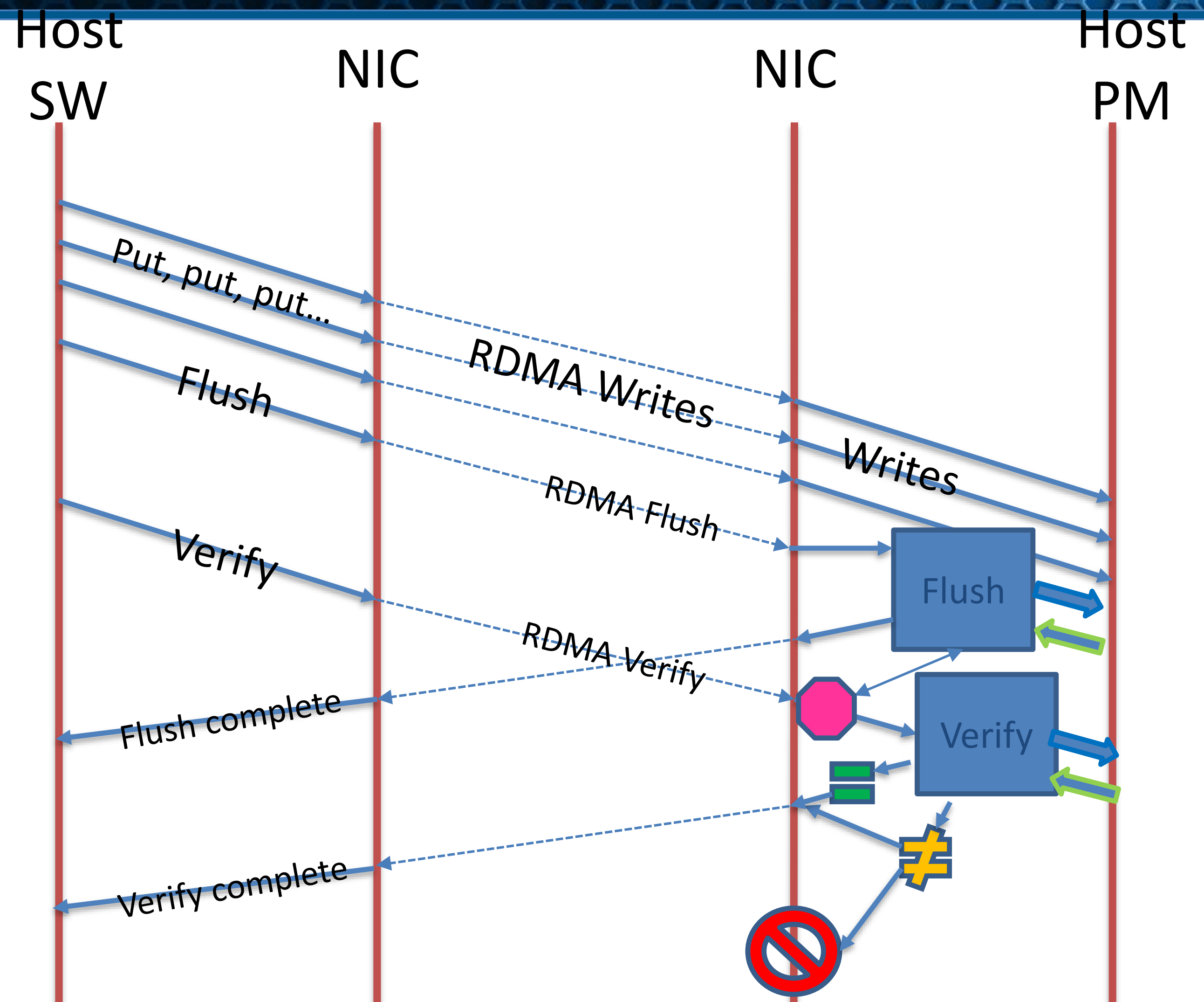
WRITE, FLUSH AND ATOMIC WRITE

- **RDMA Atomic Write**
  - Additional new non-posted/queued operation
  - Executes at responder only after successful prior non-posted operations (i.e. Flush)
  - Implementable at responder with or without PCIe Atomic support
- **Logwriter example shown**
- **Similarly able to support 2-phase commit**



WRITE, FLUSH AND VERIFY

- **RDMA “Verify”**
  - *Under discussion*
- **Computes and returns the hash of a region**
  - Non-posted/queued to execute at responder only after prior Flush etc
  - Must read the actual persistence domain, not the visibility domain!
  - Optional behavior to return the hash, or break connection on mismatch
- **In support of enhanced “Optimized Flush and Verify”**
- **Supports “paranoid log writer”**
  - Using break-on-mismatch to fence a following Atomic Write
  - Without requiring a pipeline bubble!
- **Also supports “scrub”**
  - Using return-the-hash



ROLE OF THE UPPER LAYER

- **Connection management**
- **Authentication**
  - Key derivation and provisioning
  - Nonce management
- **Authorization**
  - Granting and revoking of remote “push handles”
- **Assigning QoS policy**
- **And all the other things Upper Layers already do**

- **Think of RDMA and extensions as an “offload” for the PM-aware data handling**

STANDARDS EFFORTS

- **IETF**
  - RDMA “Commit” (Flush) concept introduced as iWARP protocol extension
  - Published as individual Internet-Draft, IETF Feb 2016
  - https://tools.ietf.org/html/draft-talpey-rdma-commit-00
  - Significant updates being prepared for new publication

- **IBTA**
  - RDMA Flush discussions begin in IBTA LWG, Sep 2016
  - Intended to become a new Annex to InfiniBand/RoCE specification (not yet publicly available)
  - https://www.snia.org/sites/default/files/PM-Summit/2019/presentations/11-PMSummit19-Burstein-Making-RM-Persistent.pdf

- **The above specifications are in harmony on Flush semantics**
  - Applications need not be concerned with choice of transport (common Verbs)

- **PCIe semantics desirable**
  - PCI SIG reportedly considering Flush semantic
    - To enable platform-independent RNIC behaviors
  - PCI “Atomic Ops ECN” (August 2017)
    - May provide additional semantic guarantees for Atomic Write RDMA operation

RDMA PM EXTENSIONS NEXT STEPS

- **SNIA NVMP TWG specification work continues**
  - OFIWG feedback on semantics
- **IBTA, IETF RDMA Standards specification proceed**
- **OFIWG and RDMA software implementation**
  - In Open Source, commercial operating systems, etc
- **RDMA vendor implementation**
- **PCI SIG specification and broad PCIe implementation**

OPENFABRICS ALLIANCE

15th ANNUAL WORKSHOP 2019

THANK YOU

Tom Talpey

Microsoft

Microsoft