SMB3 Extensions for Low Latency

Tom Talpey

Microsoft

May 12, 2016

Problem Statement

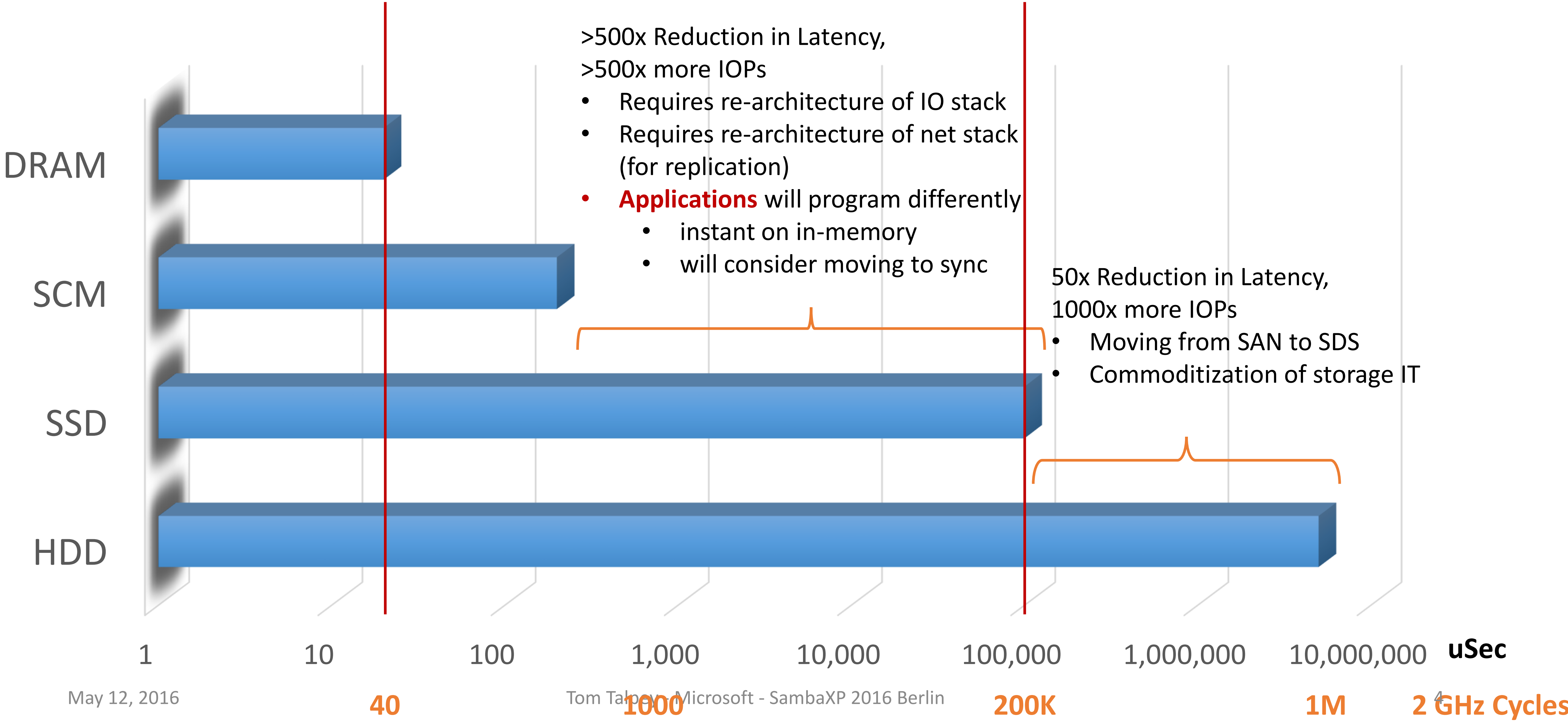
- "Storage Class Memory"
  - A new, disruptive class of storage
  - Nonvolatile medium with RAM-like performance
    - Low latency, high throughput, high capacity
  - Resides on memory bus
    - Byte addressable
  - Or also on PCIe bus
    - Block semantics
- New interface paradigms are rising to utilize it
  - Many based on time-honored methods (mapped files, etc)

Low Latency Storage

- 2000 – HDD latency – SAN arrays accelerated using memory
  - ~5000 usec latency
- 2010 – SSD latency –mere mortals can configure high perf storage
  - ~100 usec latency (50x improvement)
- 2016 – beginning of Storage Class Memory (SCM) revolution
  - <1 usec latency (local), <10 usec latency (remote) – (~100x improvement)
  - Volume deployment imminent (NVDIMM today)

5000x change over 15 years!

Storage Latencies and Storage API



Need for A New Programming Model

- Current programming model
  - Data records are created in volatile memory
    - Memory operations
  - Copied to HDD or SSD to make them persistent
    - I/O operations
- Opportunities provided by NVM devices
  - Software to skip the steps that copy data from memory to disks.
  - Software can take advantages of the unique capabilities of both persistent memory and flash NVM
- Need for a new programming model
  - Application writes persistent data directly to NVM which can be treated just like RAM
  - Mapped files, DAX, NVML
- Storage can follow this new model

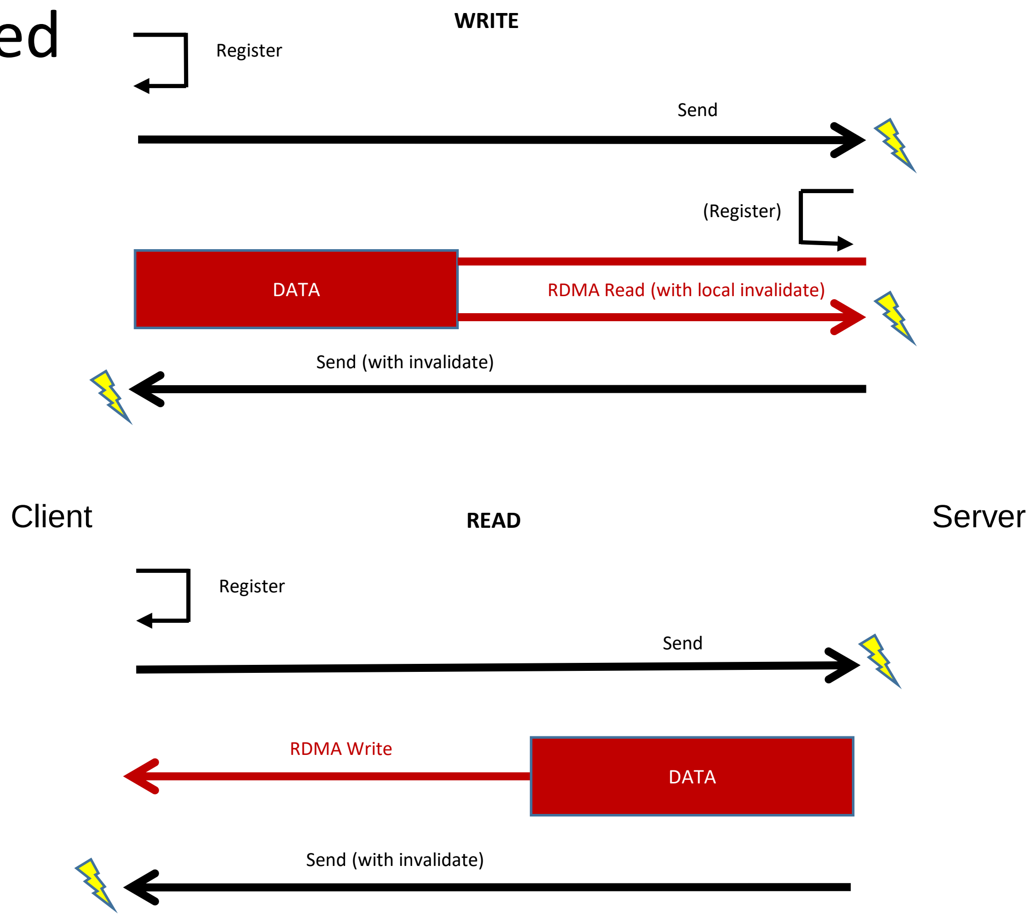
Local Filesystems and Local APIs

- DAX
  - Direct Access Filesystem
  - Windows and Linux (very) similar
- NVML
  - NVM Programming Library
  - Open source, included in Linux, future included in Windows
- Specialized interfaces
  - Databases
  - Transactional libraries
  - Language extensions (!)
  - etc

Push Mode

RDMA Transfers – Storage Protocols Today

- Direct placement model (simplified and optimized)
  - Client advertises RDMA region in scatter/gather list
  - Server performs all RDMA
    - More secure: client does not access server's memory
    - More scalable: server does not preallocate to client
    - Faster: for parallel (typical) storage workloads
  - SMB3 uses for READ and WRITE
    - Server ensures durability
    - NFS/RDMA, iSER similar
- Interrupts and CPU on both sides

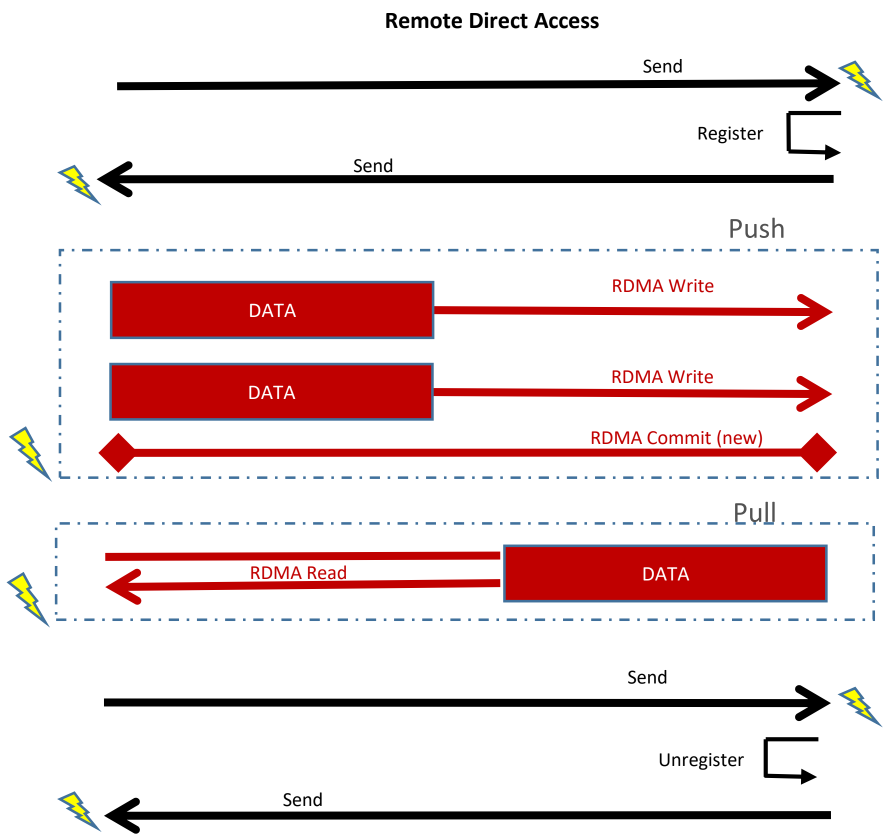


Latencies

- Undesirable latency contributions
  - Interrupts, work requests
    - Server request processing
    - Server-side RDMA handling
  - CPU processing time
    - Request processing
  - I/O stack processing and buffer management
    - To “traditional” storage subsystems
  - Data copies
- Can we reduce or remove all of the above to PM?

RDMA Push Mode (Schematic)

- Enhanced direct placement model
  - Client requests server resource of file, memory region, etc
    - MAP_REMOTE_REGION(offset, length, mode r/w)
  - Server pins/registers/advertises RDMA handle for region
  - Client performs all RDMA
    - RDMA Write to region
    - RDMA Read from region (“Pull mode”)
    - No requests of server (no server CPU/interrupt)
      - Achieves near-wire latencies
  - Client remotely commits to PM (new RDMA operation!)
    - Ideally, no server CPU interaction
    - RDMA NIC optionally signals server CPU
    - Operation completes at client only when remote durability is guaranteed
- Client periodically updates server via master protocol
  - E.g. file change, timestamps, other metadata
- Server can call back to client
  - To recall, revoke, manage resources, etc
- Client signals server (closes) when done



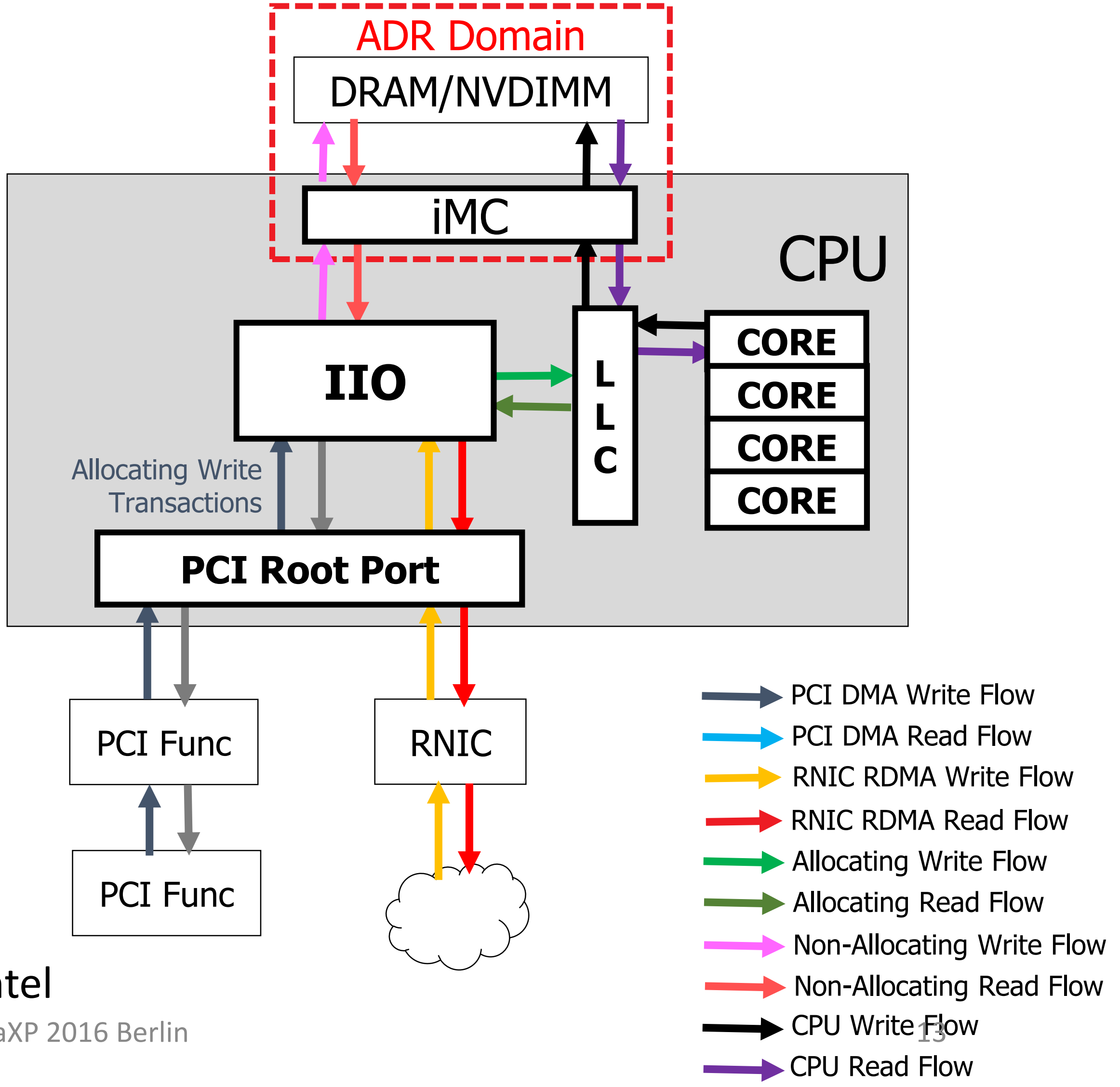
Push Mode Implications

- Historically, RDMA storage protocols avoided push mode
- For good reasons:
  - Non-exposure of server memory
  - Resource conservation
  - Performance (perhaps surprisingly)
    - Server scheduling of data with I/O
    - Write congestion control – server-mediated data pull
- Today:
  - Server memory can be well-protected with little performance compromise
  - Resources are scalable
  - However, congestion issue remains
    - Upper storage layer crediting
    - Hardware (RDMA NIC) flow control
    - QoS infrastructure
      - Existing Microsoft/MSR innovation to the rescue?

Consistency and Durability - Platform

RDMA with byte-addressable PM – Intel HW Architecture - Background

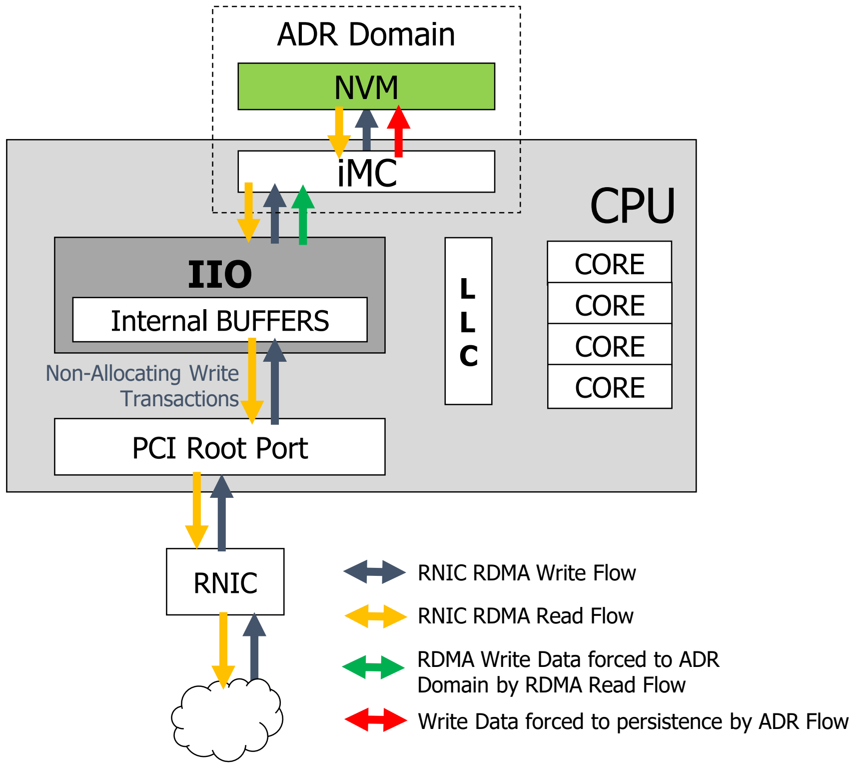
- **ADR – Asynchronous DRAM Refresh**
  - Allows DRAM contents to be saved to NVDIMM on power loss
  - Requires special hardware with PS or supercap support
  - ADR Domain – All data inside of the domain is protected by ADR and will make it to NVM before power dies. The integrated memory controller (iMC) is currently inside of the ADR Domain.
  - HW does not guarantee the order that cache lines are written to NVM during an ADR event
- **IIO – Integrated IO Controller**
  - Controls IO flow between PCIe devices and Main Memory
  - “Allocating write transactions”
    - PCI Root Port will utilize write buffers backed by LLC core cache when the target write buffer has WB attribute
    - Data buffers naturally aged out of cache to main memory
  - “Non-Allocating write transactions”
    - PCI Root Port Write transactions utilize buffers not backed by cache
    - Forces write data to move to the iMC without cache delay
  - Various Enable/Disable methods, non-default
- **DDIO – Data Direct IO**
  - Allows Bus Mastering PCI & RDMA IO to move data directly in/out of LLC Core Caches
  - Allocating Write transactions will utilize DDIO

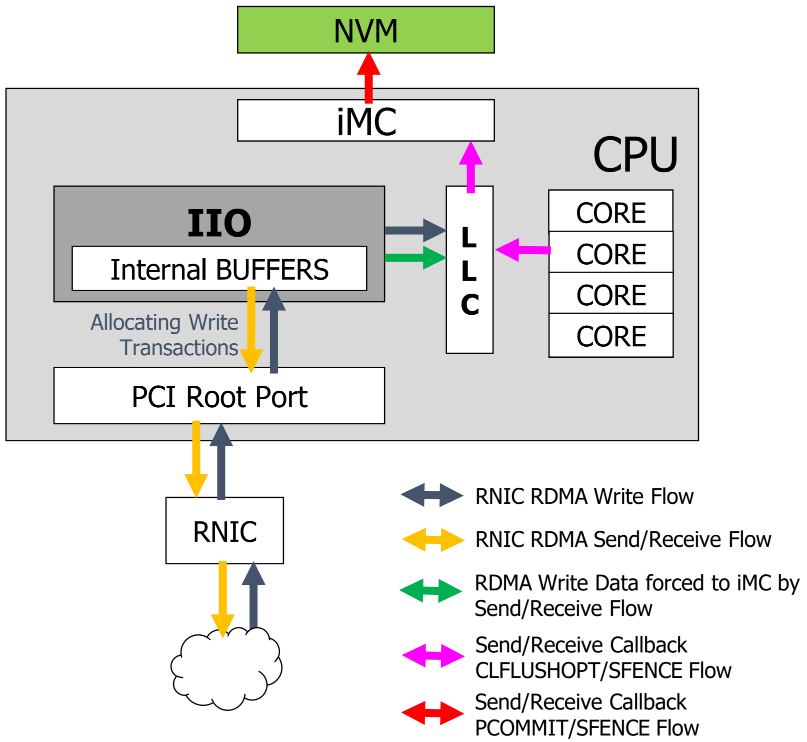


Credit: Intel

Durability Workarounds

- Alternatives proposed – also see SDC 2015 Intel presentation
- Significant performance (latency) implications, however!





Credit: Intel

RDMA Durability – Protocol Extension

“Doing it right” - RDMA protocols

- Need a remote guarantee of Durability
- RDMA Write alone is not sufficient for this semantic
  - Completion at sender does not mean data was placed
    - NOT that it was even sent on the wire, much less received
    - Some RNICs give stronger guarantees, but never that data was stored remotely
  - Processing at receiver means only that data was accepted
    - NOT that it was sent on the bus
    - Segments can be reordered, by the wire or the bus
    - Only an RDMA completion at receiver guarantees placement
      - And placement != commit/durable
  - No Commit operation
- Certain platform-specific guarantees can be made
  - But the remote client cannot know them
  - E.g. RDMA Read-after-RDMA Write (which won’t generally work)

RDMA protocol extension

- Two “obvious” possibilities
  - RDMA Write with placement acknowledgement
    - Advantage: simple API – set a “push bit”
    - Disadvantage: significantly changes RDMA Write semantic, data path (flow control, buffering, completion). Requires creating a “Write Ack”.
    - Requires significant changes to RDMA Write hardware design
      - And also to initiator work request model (flow controlled RDMA Writes would block the send work queue)
    - **Undesirable**
  - RDMA “Commit”
    - New operation, flow controlled/acknowledged like RDMA Read or Atomic
    - Disadvantage: new operation
    - Advantage: simple API – “flush”, operates on one or more regions (allows batching), preserves existing RDMA Write semantic (minimizing RNIC implementation change)
    - **Desirable**

RDMA Commit (concept)

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

Other RDMA Commit Semantics

- Desirable to include other semantics with Commit:
  - Atomically-placed data-after-commit
    - E.g. “log pointer update”
  - Immediate data
    - E.g. to signal upper layer
  - An entire message
    - For more complex signaling
    - Can be ordinary send/receive, only with new specific ordering requirements
  - Additional processing, e.g. integrity check
  - These may be best implemented in ordered following operations
- Decisions will be workload-dependent
  - Small log-write scenario will always commit
  - Bulk data movement will permit batching

Platform-specific Extensions

- PCI extension to support Commit
  - Allow NIC to provide durability directly and efficiently
  - To Memory, CPU, PCI Root, PM device, PCIe device, …
  - Avoids CPU interaction
  - Supports strong data consistency model
- Performs equivalent of:
  - CLFLUSHOPT (region list)
  - PCOMMIT
- Or if NIC is on memory bus or within CPU complex…
  - Other possibilities exist
  - Platform-specific implementations, on platform-local basis
- Standard extensions are most desirable

Latencies (expectations)

- Single-digit microsecond remote Write+Commit
  - Push mode minimal write latencies <<10us (2-3us + data wire time)
  - Commit time is NIC-managed and platform+payload dependent
  - Note, this is **order-of-magnitude improvement** over today's transfer mode
    - 30-50us as mentioned
- Remote Read also possible
  - Roughly same latency as write, but without commit
- No server interrupt
  - Zero server CPU overhead
  - Once RDMA and PCIe extensions in place
- Single client interrupt
  - Moderation and batching can reduce further when pipelining
- Deep parallelism with Multichannel and flow control management

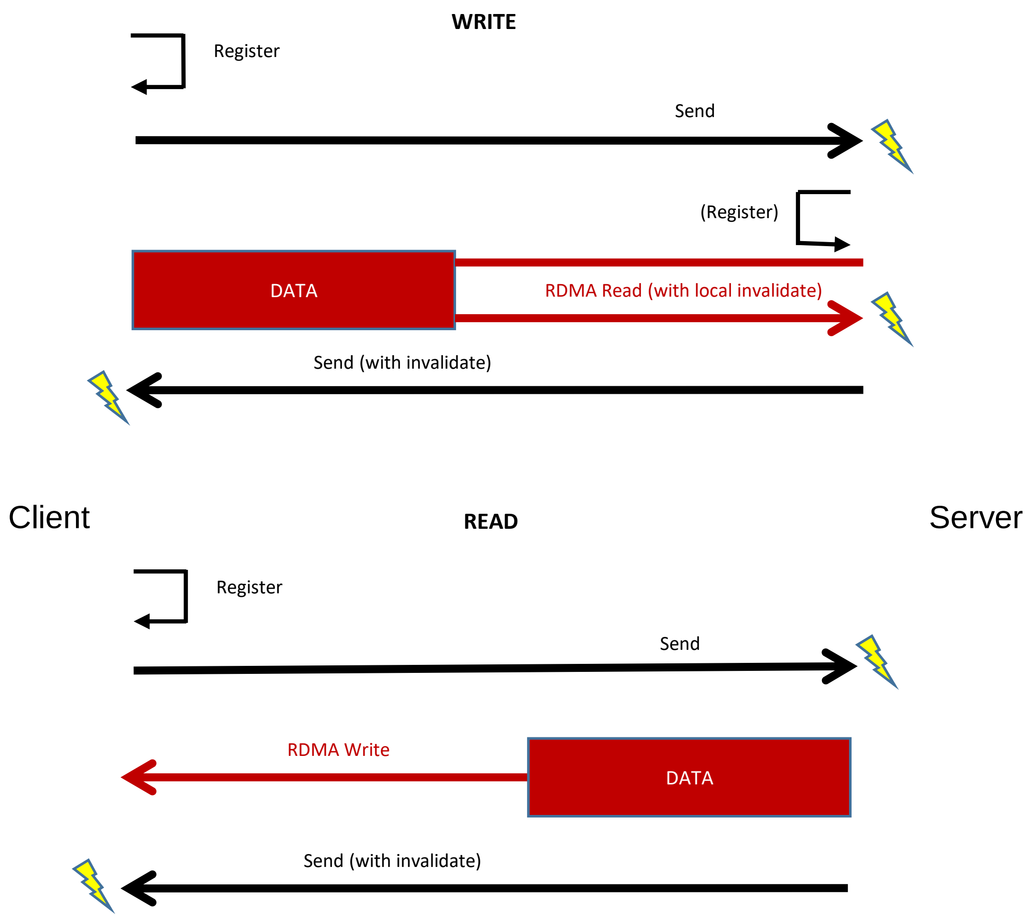
SMB3 Push Mode

SMB3 Low Latency - Architecture

- The following is only a possible approach!
  - Applies to any DAX-enabled OS e.g. Linux, Windows, et al.
  - Disclaimer: this is NOT a specification, and NOT a protocol
- Hope that Samba, and all SMB3 and SMB Direct implementations can adopt this to take advantage of PMEM-capable systems as they appear
- DAX:
  - http://www.snia.org/sites/default/files/NVM/2016/presentations/Neal%20Christiansen_SCM_in_Windows_NVM_Summit.pdf
  - http://www.snia.org/sites/default/files/NVM/2016/presentations/JeffMoyer_Persistent-Memory-in-Linux.pdf

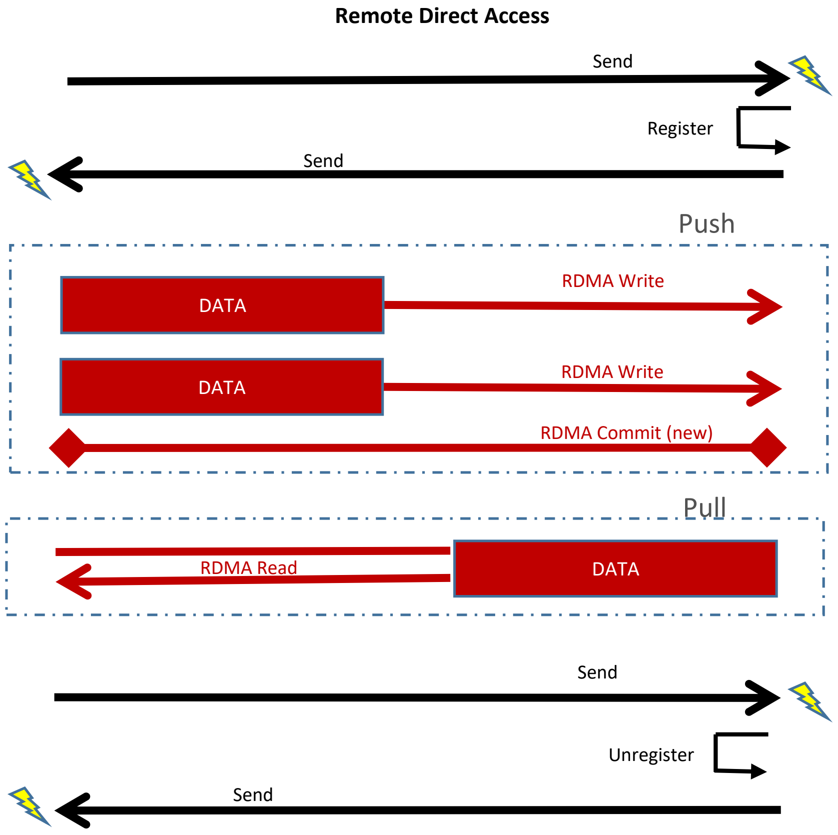
SMB3 Commit - Traditional Mode

- Basic steps:
  - Open file, obtain handle and lease
  - Write file with SMB2_WRITE (or read with SMB2_READ)
    - Buffered or writethrough
    - Server process performs load/store operations to PMEM file
  - Flush file with SMB2_FLUSH
    - Performs Commit when share is DAX-enabled
  - Handle possible recalls
  - Close file
- Advantages:
  - Traditional API, fully functional
- Disadvantages:
  - Higher latencies from roundtrips, message signaling, work requests



SMB3 Push Mode

- Basic steps:
  - Open DAX-enabled file
  - Obtain a lease
  - Request a push-mode registration
  - While (TRUE)
    - Push (or pull) data
    - Commit data to durability
  - Release registration
  - Drop lease
  - Close handle
- Details follow



SMB3 Push Mode - Open

- Opening file performs “usual” processing
  - Authorization, handle setup, etc
  - New SMB2 Create Context (perhaps) signals desired Push Mode
  - Sent in conjunction with requesting a lease
    - If DAX filesystem and lease, establishes mapped-file r/w interface to server
    - Returns the create context to indicate push mode is available
- Obtaining a lease performs “usual” processing **plus**
  - The lease provides a means to manage mapping
  - If mapping changes, or if later registration revokes, lease recall is performed
- DAX mode may require certain “traditional” client steps
  - Issued via SMB3
  - E.g. extending file to allocate blocks / establish mapping
- Otherwise, nothing unusual

SMB3 Push Mode – Registration

- Need to register file region(s) for RDMA Write from client
- Client issues new FSCTL
  - MAP_REMOTE_REGION(offset, length, mode r/w)
  - Server pins mapping, RDMA registers pages, returns RDMA region handle
- Client can request multiple regions
  - And multiple r/w modes
- Client can remotely write or read

SMB3 Push Mode – Operation

- Client performs remote writes and reads directly to and from file
  - Entirely within RDMA layers, no server processing at all!
- If remote RDMA Commit operation available:
  - Client commits writes via RDMA
- Otherwise:
  - Client commits vis SMB2_FLUSH
- Client may periodically update file metadata timestamps, etc
  - Using SMB2_IOCTL(FILE_BASIC_INFORMATION), etc
- Note – Push Mode cannot:
  - Add blocks / append file
  - Punch holes, etc
  - These must be done with traditional SMB3 operations

SMB3 Push Mode - Recall

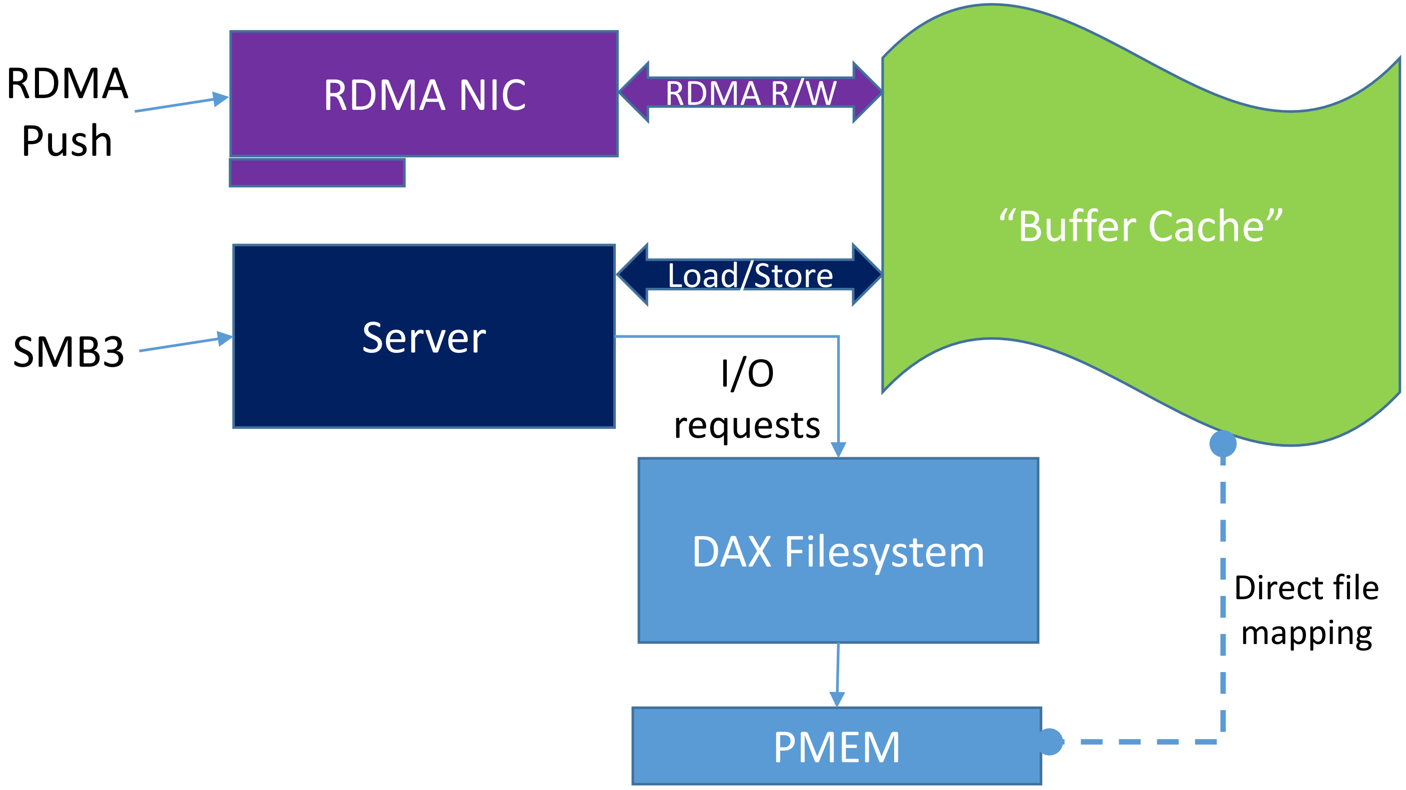
- Server must manage sharing and registration
- Client ownership of a lease covers both
- Recalled upon:
  - Sharing violation (as usual)
  - Mapping change (new DAX callback)
  - Registration change (RDMA resource behavior)
    - Managed by Server itself
- When recalled, client performs the usual:
  - Flush any dirty buffers (and possibly commit)
  - Return the lease
  - Possibly re-obtain a new lease and re-register for push mode
    - If not, fall back to “normal” SMB3 operations

SMB3 Push Mode – Done

- When done, “usual” processing
  - Return lease / close handle
  - Returning the lease and/or closing handle clears push mode registration
    - Or also… an explicit fsctl? TBD
- Push Mode registration is reset during handle recovery
  - Must be re-obtained after handle recovered (like leases)
  - New mapping, new RDMA handle
    - If issued early, RDMA operations would fail and kill connection

SMB3 Push Mode – Fun Fact #1

- All push mode handles are in "buffered" mode
- Server opens DAX files this way (both Linux and Windows)
  - Direct access create context is hint to do this
  - Server may do it anyway, when DAX is detected
    - Allows load/store for r/w
- Buffered mode enables direct mapping and direct RDMA reading/writing of PMEM-resident file
  - Direct mapping allows RDMA read/write, and RDMA Commit



SMB3 Push Mode – Fun Fact #2

- Push mode management requires new server plumbing
- Registration recalls may be needed for:
  - Sharing violations/file changes (as usual)
  - DAX filesystem events (file relocation, extension, etc)
  - RDMA resource constraints
  - Any other server matter
- Lease recall is mechanism for all

☞ Server “upcalls” may originate within and without filesystem

SMB3 Push Mode – Fun Fact #3

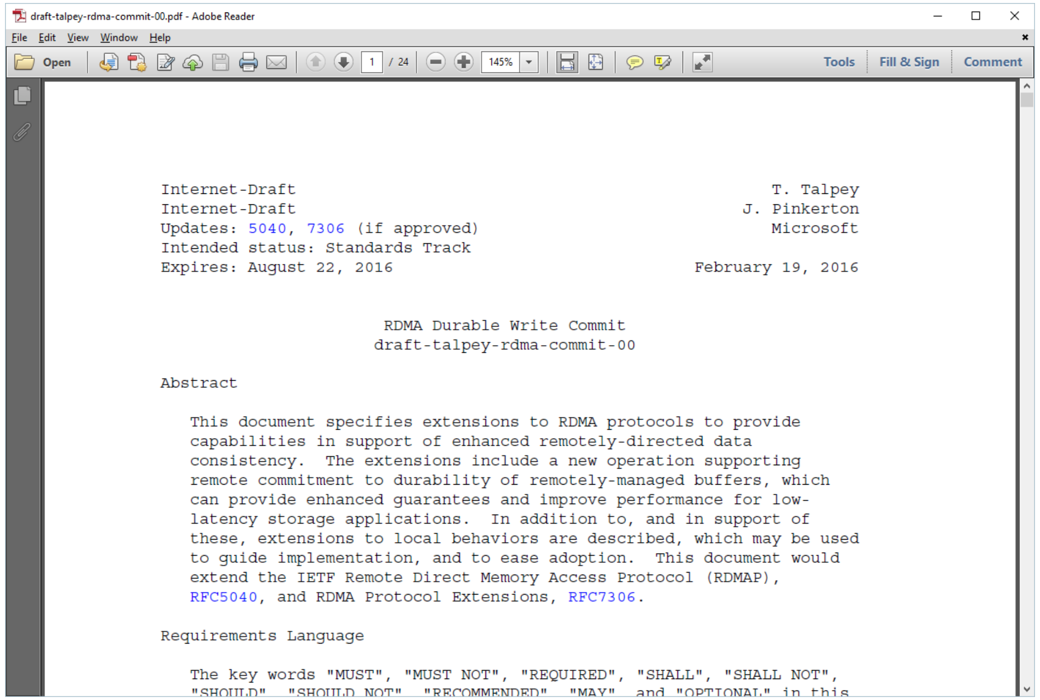
- Push Mode fan-in from many (1000's+) clients can congest interface
- And also congest the PMEM
- RDMA itself does not provide congestion control
- But, SMB3 does
  - With credits, and also Storage QoS
  - Existing client-side behavior can mitigate
- RNICs can also provide QoS control
- More thinking needed here

SMB3 Push Mode – Fun Fact #4

- Data-in-flight protection and data-at-rest integrity validation?
  - In SMB3, provided by signing and/or encryption, and the backend file store
  - Push mode transfers all data via RDMA, “below” SMB3
- RDMA protocols do not currently support signing or encryption
  - Specifications refer to IPsec or other lower-layer facility
  - Extensions to be discussed in RDMA standards areas
- Remote integrity also not available
  - Remote server CPU not at all involved in transfers
  - Extensions also being discussed for RDMA remote integrity validation

External RDMA Efforts

- Requirements and Protocol
  - For RDMA Commit operation
  - Also local PM behaviors
    - Memory registration
  - Independent of transport
    - Applies to iWARP, IB, RoCE
- IETF Working Group
  - STORM: RDMA (iWARP) and Storage (iSCSI)
  - Recently closed, but active for discussion
  - Another WG, or individual process TBD
- Also discussing in
  - IBTA (IB/RoCE) expected
  - SNIA NVM TWG
  - Open Fabrics DS/DA? etc.



draft-talpey-rdma-commit-00.pdf - Adobe Reader

```
Internet-Draft                                              T. Talpey
Internet-Draft                                           J. Pinkerton
Updates: 5040, 7306 (if approved)                           Microsoft
Intended status: Standards Track
Expires: August 22, 2016                            February 19, 2016

                     RDMA Durable Write Commit
                    draft-talpey-rdma-commit-00

Abstract

   This document specifies extensions to RDMA protocols to provide
   capabilities in support of enhanced remotely-directed data
   consistency.  The extensions include a new operation supporting
   remote commitment to durability of remotely-managed buffers, which
   can provide enhanced guarantees and improve performance for low-
   latency storage applications.  In addition to, and in support of
   these, extensions to local behaviors are described, which may be used
   to guide implementation, and to ease adoption.  This document would
   extend the IETF Remote Direct Memory Access Protocol (RDMAP),
   RFC5040, and RDMA Protocol Extensions, RFC7306.

Requirements Language

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
```

https://datatracker.ietf.org/doc/draft-talpey-rdma-commit/

Resources

- SNIA NVM Programming TWG:
  - http://www.snia.org/forums/sssi/nvmp
- SDC 2015:
  - http://www.snia.org/events/storage-developer/presentations15
- NVM Summit 2016:
  - http://www.snia.org/nvmsummit2016
- Open Fabrics Workshop:
  - https://openfabrics.org/index.php/2016-ofa-workshop-presentations.html