SDC

STORAGE DEVELOPER CONFERENCE

SNIA SANTA CLARA, 2017

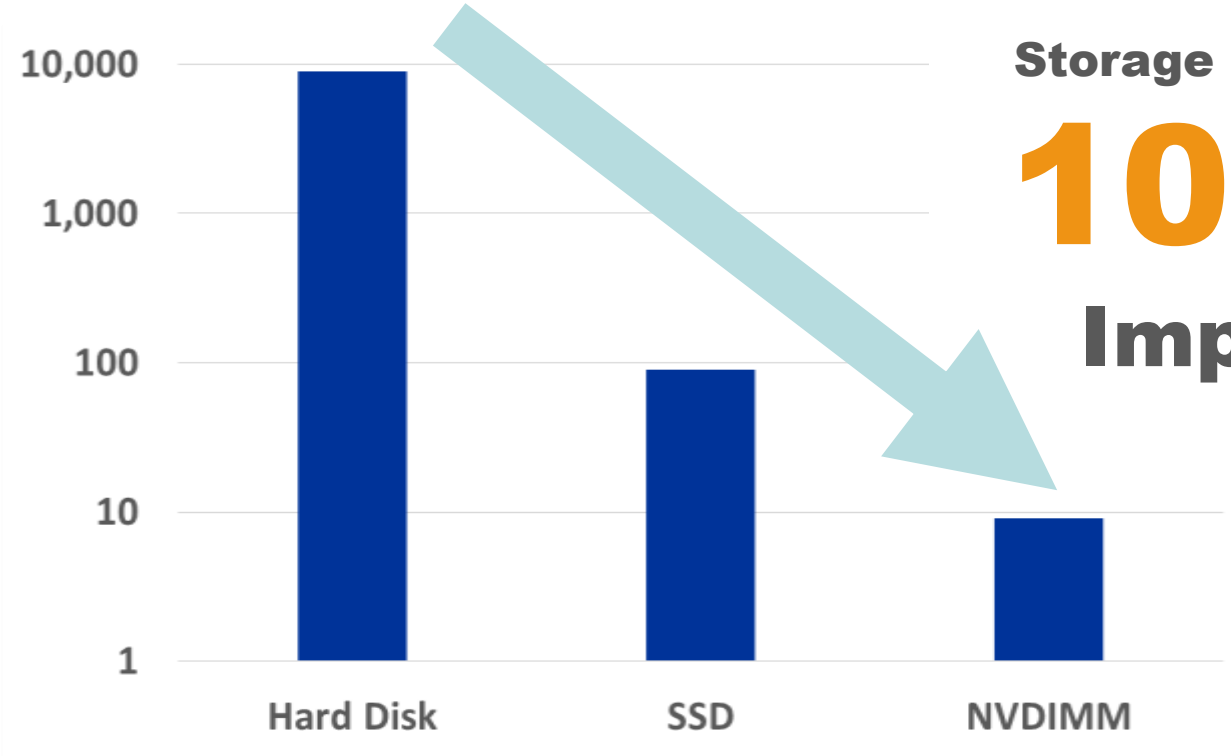
Ethernet Storage Fabrics

Using RDMA with Fast NVMe-oF Storage to Reduce Latency and Improve Efficiency

Kevin Deierling & Idan Burstein

Mellanox Technologies

Storage Media Technology



Storage Media Access Time

10,000X

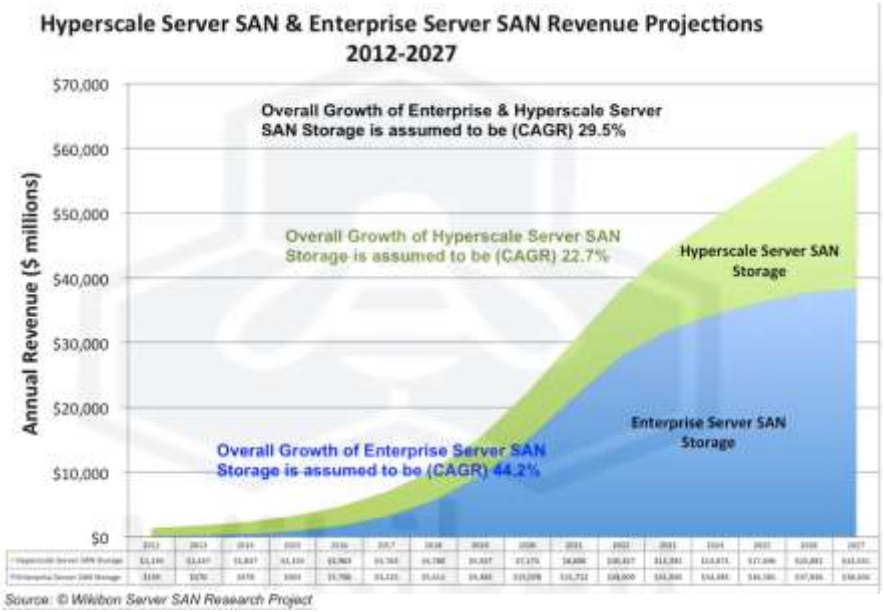
Improvement

Ethernet Storage Fabric

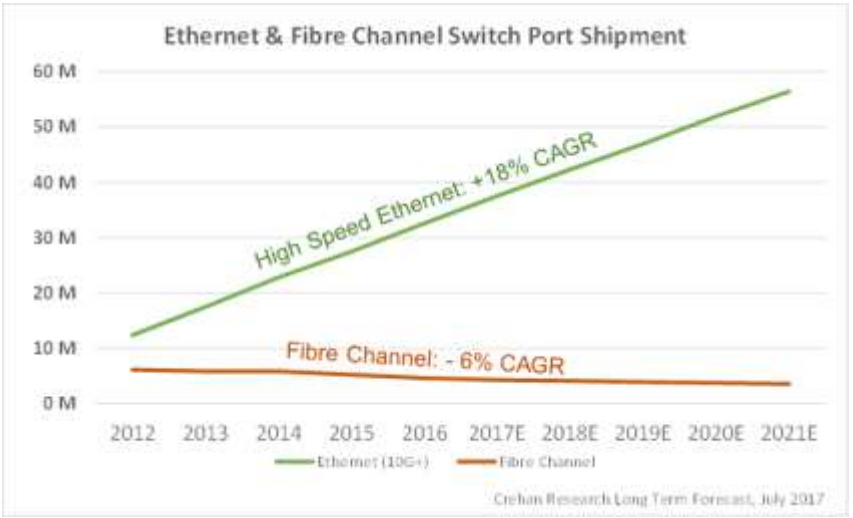
Everything a Traditional SAN Offers but …
Faster, Smarter, & Less Expensive

PERFORMANCE	INTELLIGENCE	EFFICIENCY
• Highest Bandwidth • Lowest latency • RDMA and storage offloads • Native NVMe-oF Acceleration	• Integrated & Automated Provisioning • Hardware-enforced Security & Isolation • Monitoring, Management, & Visualization • Storage-aware QoS	• Just Works Out of the Box • Flexibility: Block, File, Object, HCI • Converged: Storage, VM, Containers • Affordable: SAN without the $$

Networked Storage Growth is Ethernet



Source: © Wikibon Server SAN Research Project

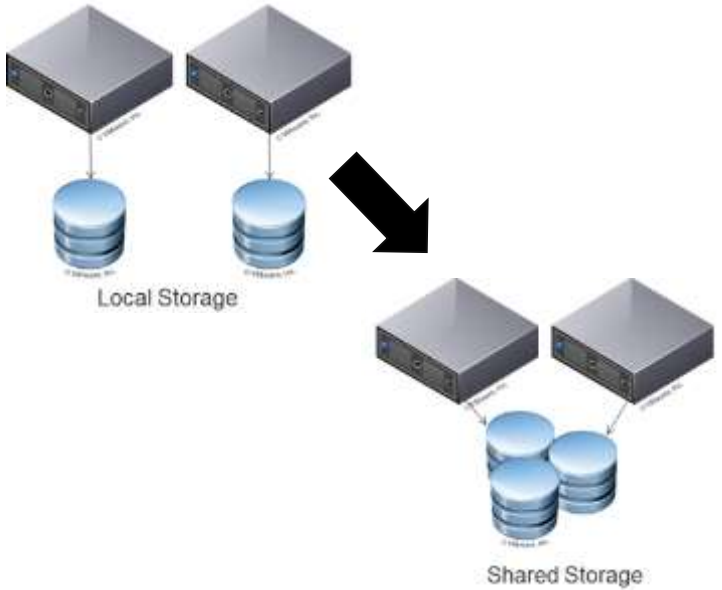


- Server SAN is the New Storage Network
  - *Because there is No Fibre Channel in the Cloud*

- Ethernet growing very rapidly driven by:
  - Cloud & HyperConverged Infrastructure
  - NVMe Over Fabrics
  - Software Defined Storage

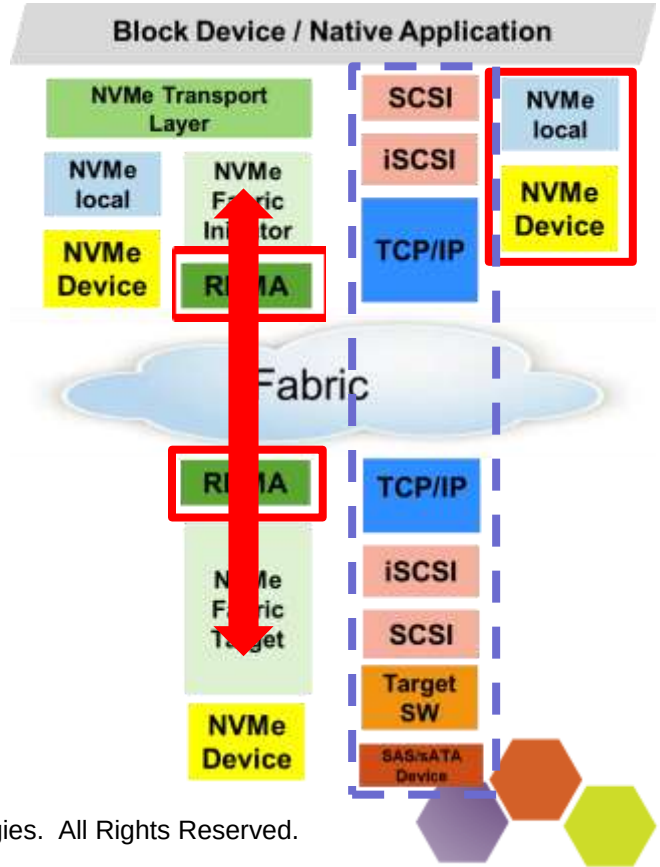
Extending NVMe Over Fabrics (NVMe-oF)

- NVMe SSDs shared by multiple servers
  - Better utilization, capacity, rack space, power
  - Scalability, management, fault isolation
- NVMe-oF industry standard
  - Version 1.0 completed in June 2016
- RDMA protocol is part of the standard
  - NVMe-oF version 1.0 includes a Transport binding specification for RDMA
  - Ethernet(RoCE) and InfiniBand

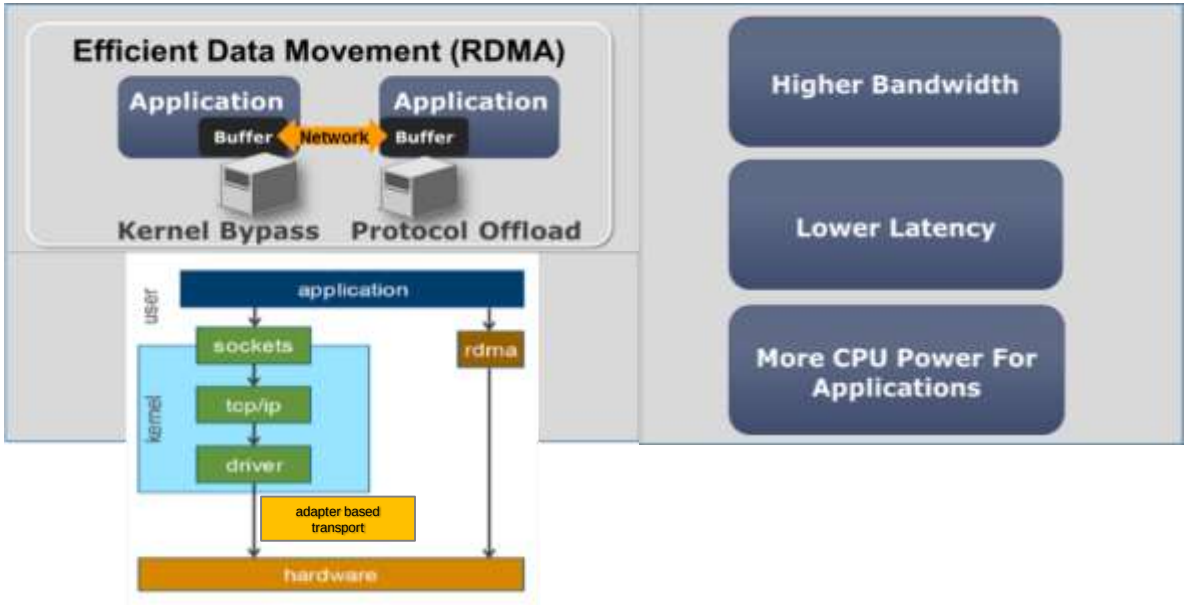


How Does NVMe-oF Maintain Performance?

- Extends NVMe efficiency over a fabric
  - NVMe commands and data structures transferred end to end
- RDMA is key to performance
  - Reduces latency
  - Increased throughput
  - Eliminates TCP/IP overhead



RDMA: More Efficient Networking

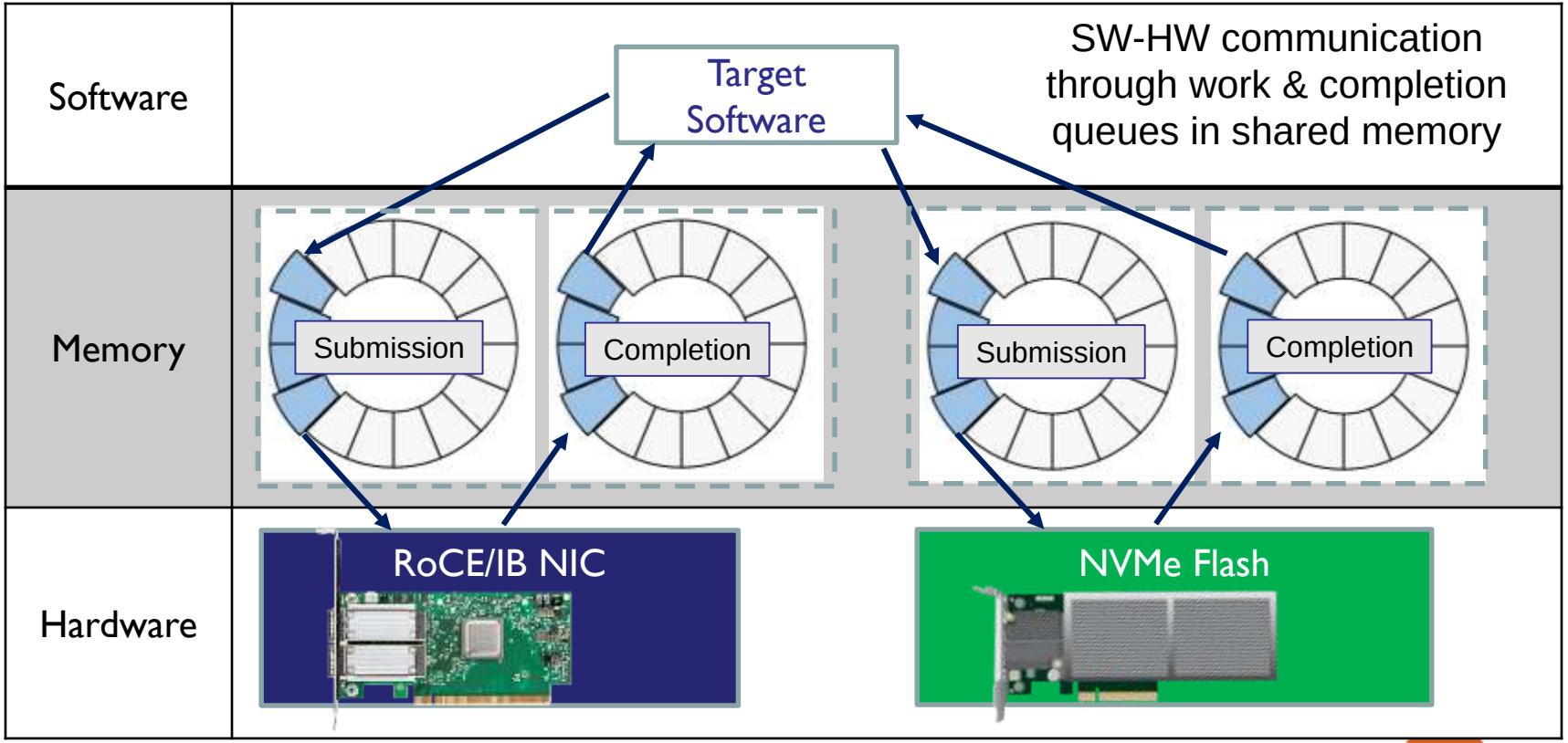


RDMA Performs Four Critical Functions in Hardware

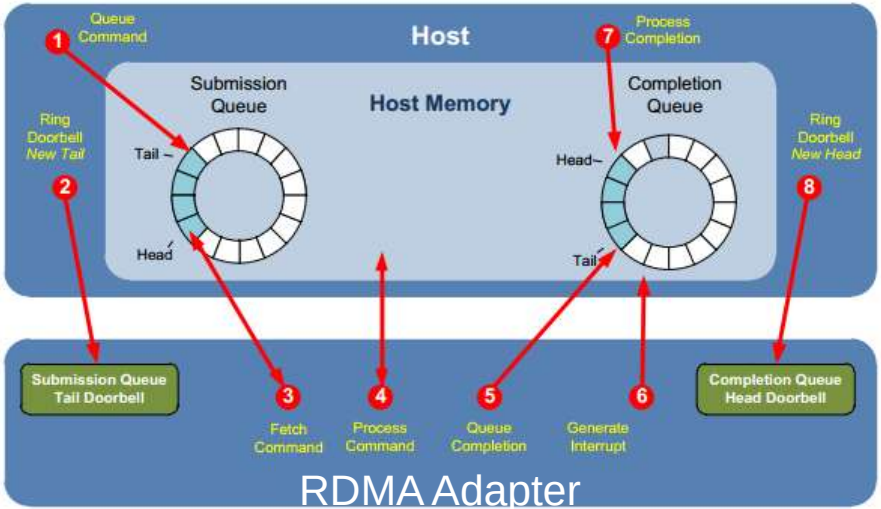
1. Reliable Data Transport
2. App-level user space I/O
   - AKA: Kernel Bypass
3. Address Translation
4. Memory Protection

- CPU not consumed moving data - Free to run apps!

RDMA is Natural Extension for NVMe



Memory Queue Based Data Transfer Flow



Command Submission

1. Host writes command to submission queue
2. Host writes updated submission queue tail pointer to doorbell

Command Processing

3. Controller fetches command
4. Controller processes command

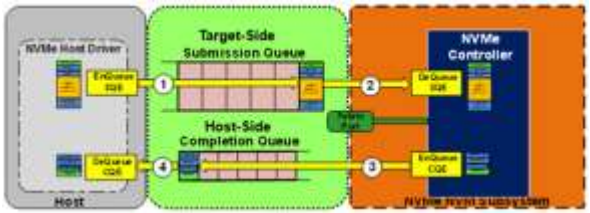
Command Completion

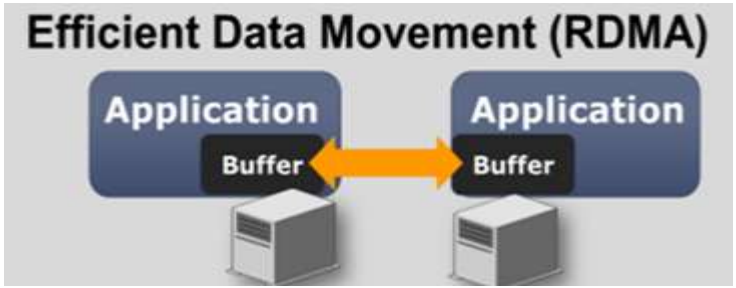
5. Controller writes completion to completion queue
6. Controller generates MSI-X interrupt
7. Host processes completion
8. Host writes updated completion queue head pointer to doorbell

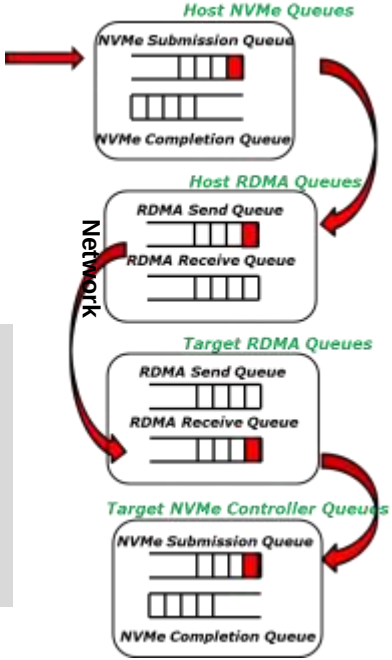
RDMA

- Transport built on simple primitives deployed for 15 years in the industry
  - **Queue Pair (QP)** – RDMA communication end point: send, receive and completion queues
  - **Connect** for establishing connection mutually
  - RDMA **Registration** of memory region (REG_MR) for enabling virtual network access to memory
  - **SEND** and **RCV** for reliable two-sided messaging
  - RDMA **READ** and RDMA **WRITE** for reliable one-sided memory to memory transmission

NVMe and NVMeoF Fit Together Well

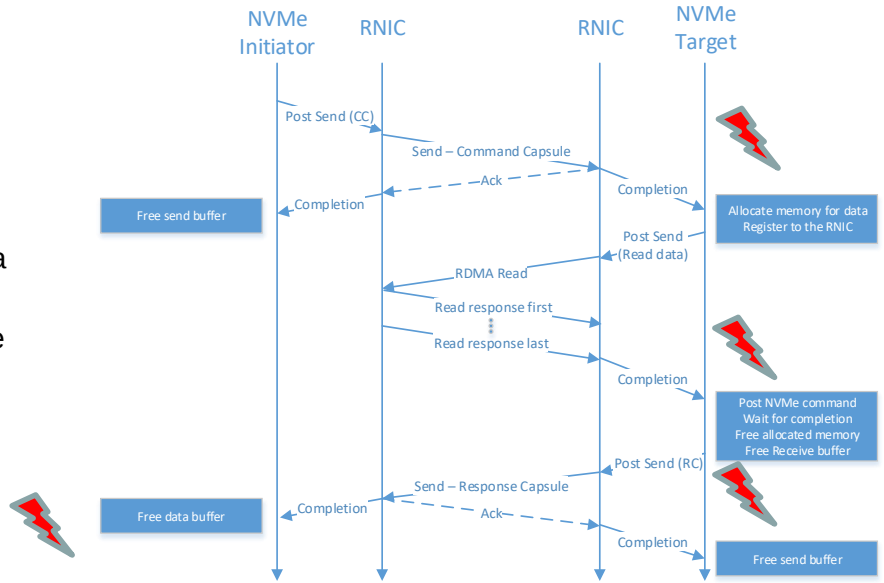






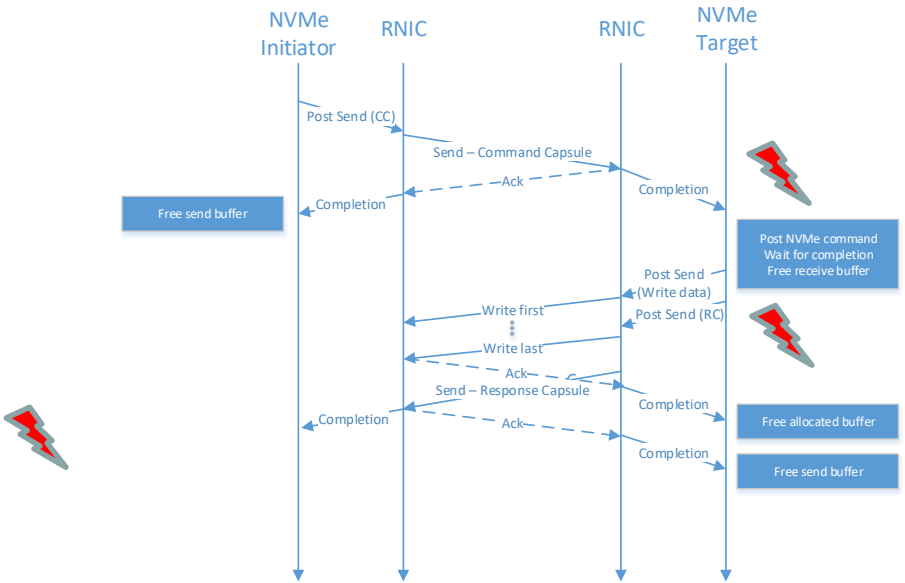
NVMe-OF IO WRITE

- Host
  - Post SEND carrying Command Capsule (CC)
- Subsystem
  - Upon RCV Completion
    - Allocate Memory for Data
    - Post RDMA READ to fetch data
  - Upon READ Completion
    - Post command to backing store
  - Upon SSD completion
    - Send NVMe-OF RC
    - Free memory
  - Upon SEND Completion
    - Free CC and completion resources



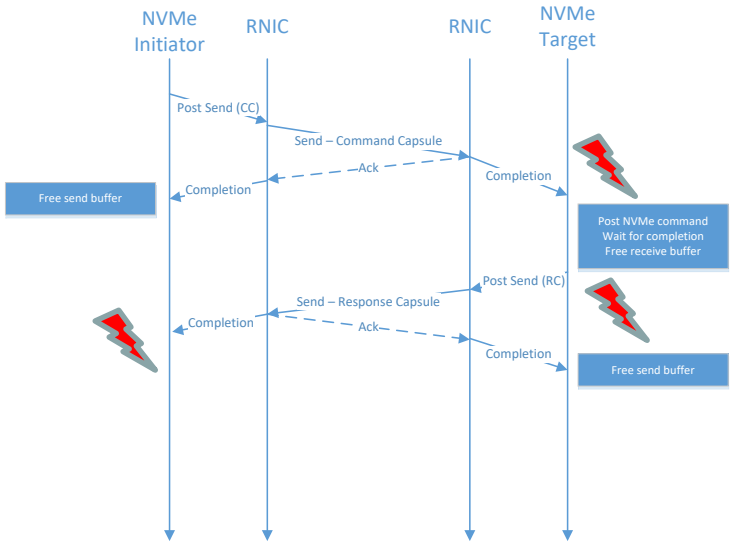
NVMe-OF IO READ

- Host
  - Post SEND carrying Command Capsule (CC)
- Subsystem
  - Upon RCV Completion
    - Allocate Memory for Data
    - Post command to backing store
  - Upon SSD completion
    - Post RDMA Write to write data back to host
    - Send NVMe RC
  - Upon SEND Completion
    - Free memory
    - Free CC and completion resources

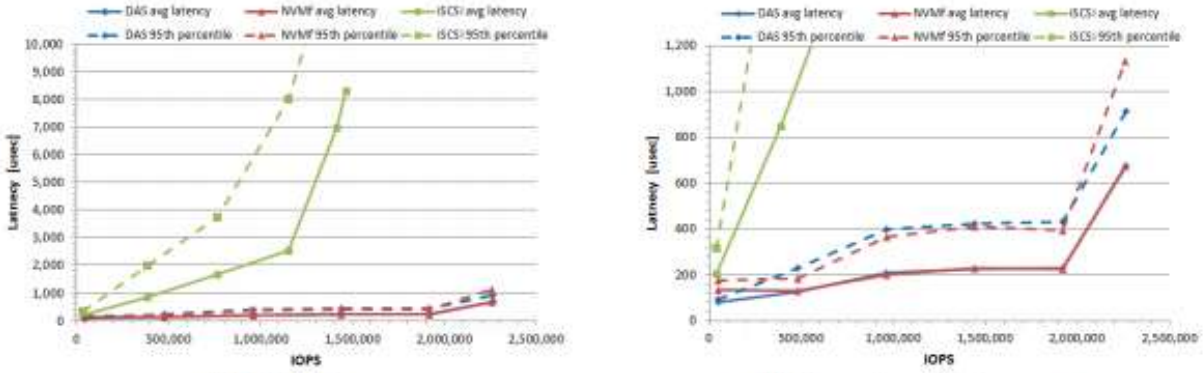


NVMe-OF IO WRITE IN-Capsule

- Host
  - Post SEND carrying Command Capsule (CC)
- Subsystem
  - Upon RCV Completion
    - Allocate Memory for Data
  - Upon SSD completion
    - Send NVMe-OF RC
    - Free memory
  - Upon SEND Completion
    - Free CC and completion resources



NVMf is Great!



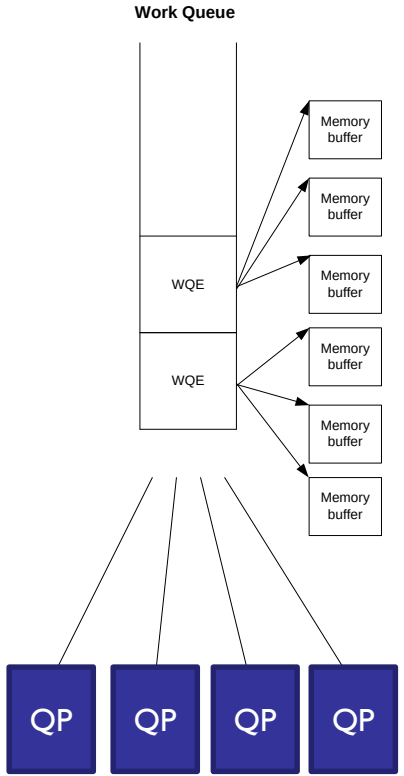
(a) Full view

(b) Zoom-in on 0-1ms range

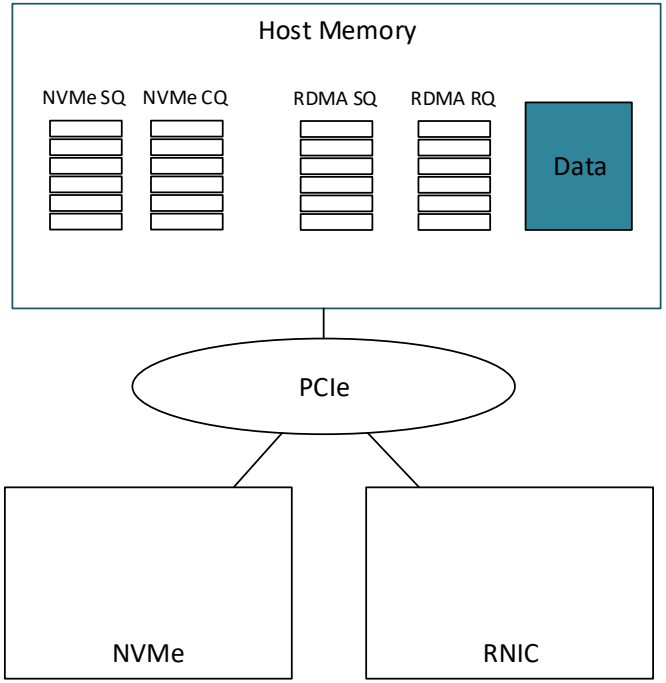
Figure 6: Average and tail latencies of DAS, NVMf, and iSCSI for different request loads. Since iSCSI saturates early and exhibits significantly higher latencies, we show both the full range view (a), and a zoom-in on DAS and NVMf (b).

Shared Receive Queue

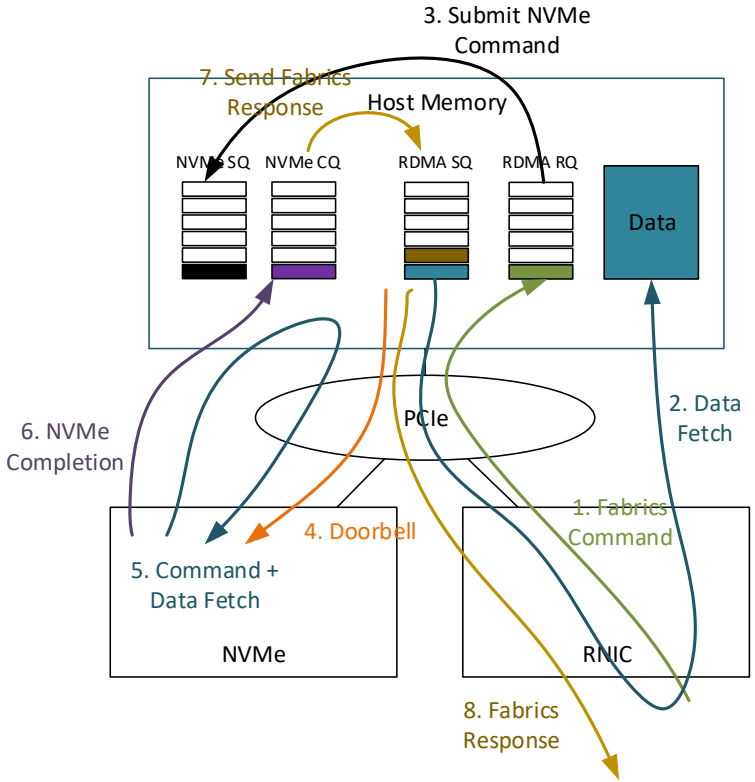
- Problem: memory foot print of a connection and the buffer associated with them
  - Locality, scalability, etc.
- Solution: Share receive buffering resources between QPs
  - According to the parallelism required by the application
- E2E credits is being managed by RNR NACK
  - TO associated with the application latency
- **We have submitted patches to fix performance in Linux – please try!**



NVMe-OF System Example



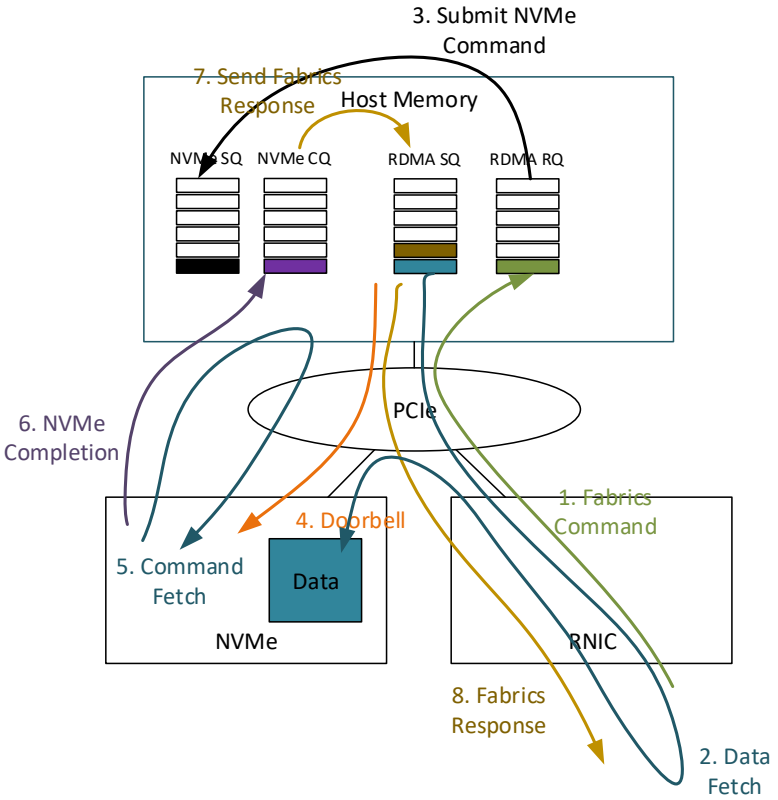
Target Data Path (NVMe WRITE)



Controller Memory Buffer

- Internal memory of the NVMe devices exposed over the PCIe
- Few MB are enough to buffer the PCIe bandwidth for the latency of the NVMe device
  - Latency ~ 100-200usec, Bandwidth ~ 25-50 GbE → Capacity ~ 2.5MB
- Enabler for peer to peer communication of data and commands between RDMA capable NIC and NVMe SSD
- Optional from NVMe 1.2

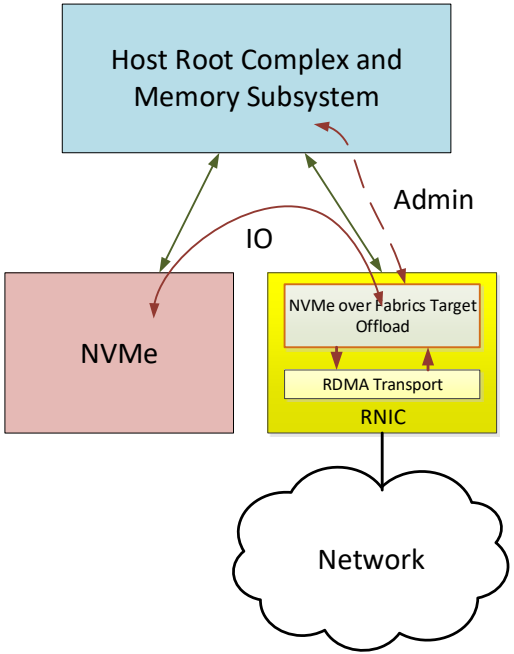
Target Data Path with CMB (NVMe WRITE)



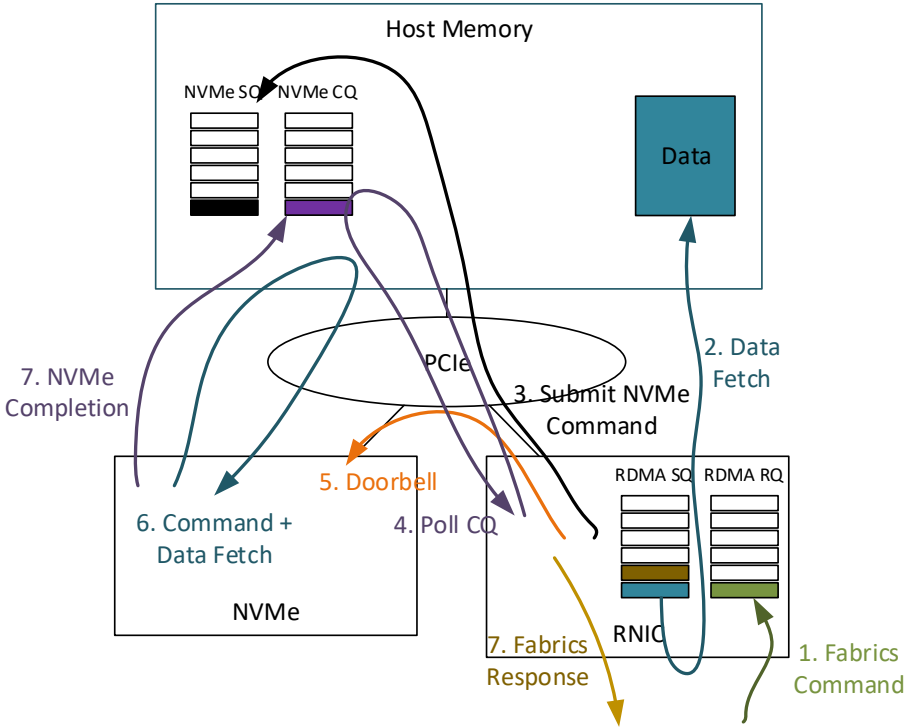
SW Arch for P2P NVMe and RDMA

NVMe Over Fabrics Target Offload

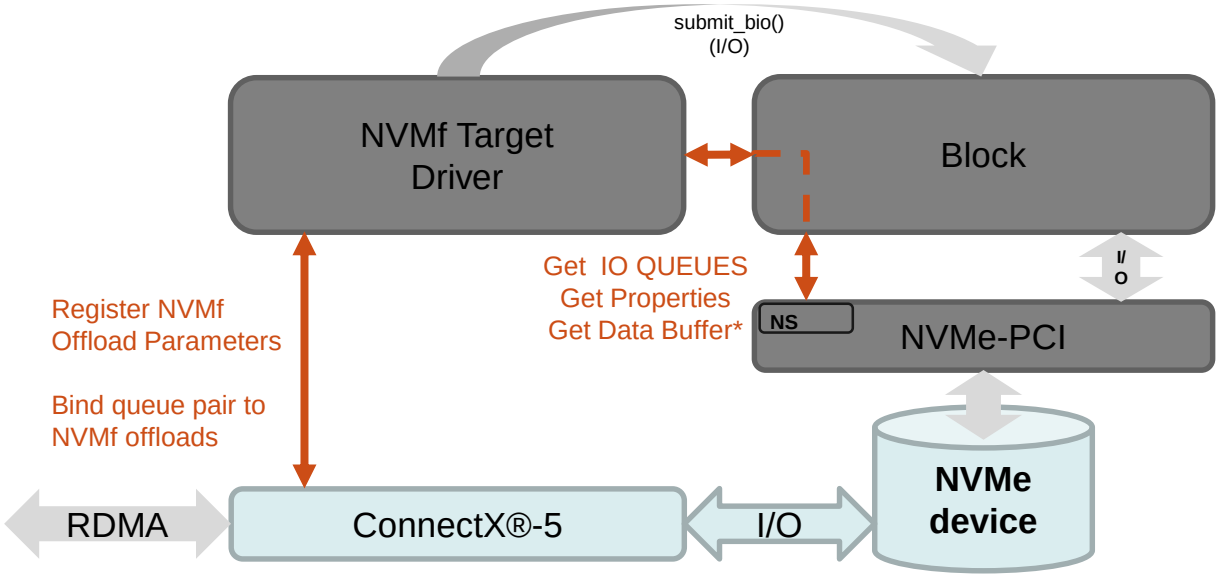
- NVMe Over Fabrics is built on top of RDMA
  - Transport communication in hardware
- NVMe over Fabrics target offload enable the NVMe hosts to access the remote NVMe devices w/o any CPU processing
  - By offloading the control part of the NVMf data path
  - Encapsulation/Decapsulation NVMf <-> NVMe is done by the adapter with 0% CPU
  - Resiliency – i.e. NVMe is exposed through Kernel Panic
  - OS Noise
- Admin operations are maintained in software



Target Data Path with NVMf Target Offload (NVMe WRITE)

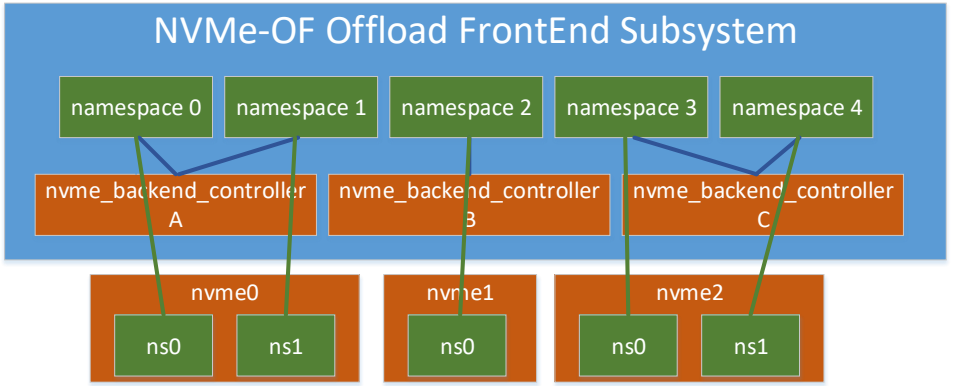


Software API for NVMf Target Offload

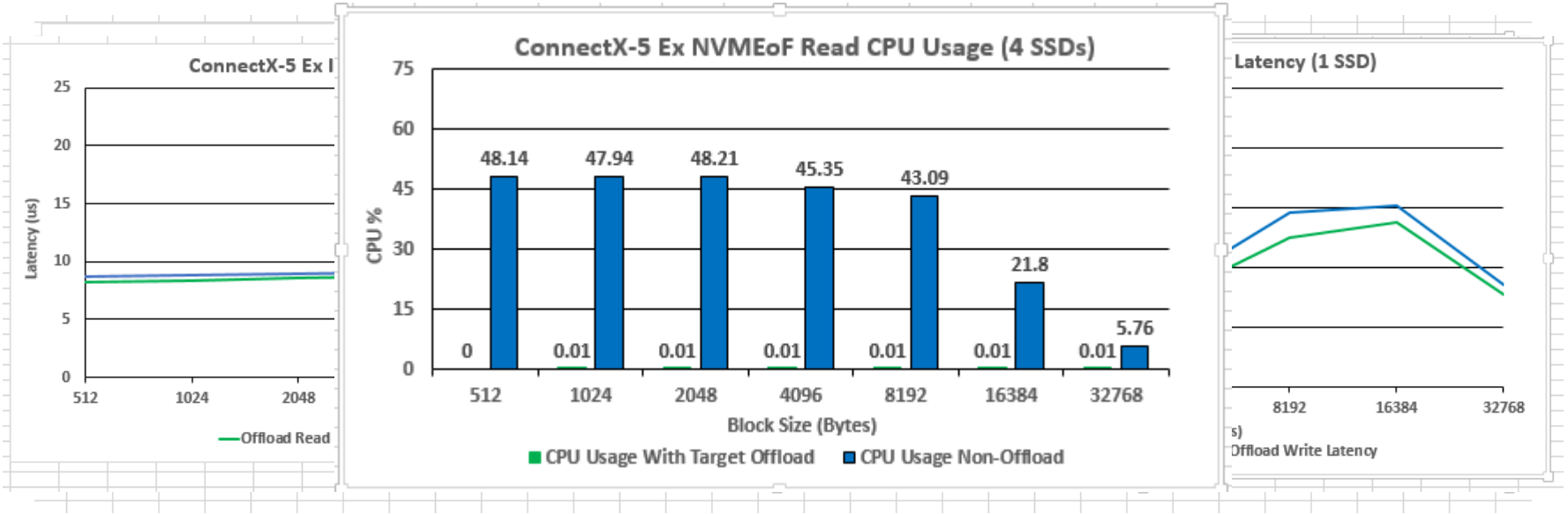


Namespace / Controller Virtualization

- Controller virtualization
  - Expose a single backend controller as multiple NVMf controllers
  - Multiplex the commands in the transport
  - To enable scalability of the amount of supported initiators
- Subsystem virtualization and namespace provisioning
  - Expose a single NVMf front-end subsystem for multiple NVMe subsystems
  - Provision namespaces to NVMf frontend controllers



Performance



Target Data Path with NVMf Target Offload and CMB (NVMe WRITE)