SDC 18
SNIA EMEA

FEBRUARY 2018
TEL AVIV, ISRAEL

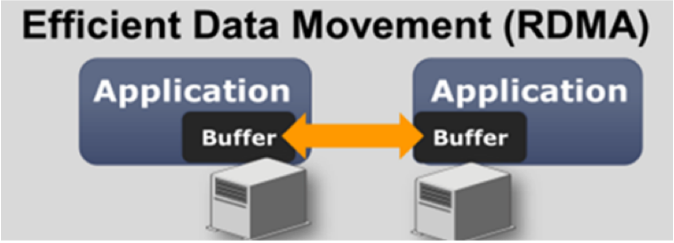
STORAGE DEVELOPER
CONFERENCE

Accelerating Storage with RDMA

Max Gurtovoy
Mellanox Technologies

What is RDMA?

- **Remote Direct Memory Access** - provides the ability to perform a direct memory access (DMA) from one computer into to another without involving either one's OS/CPU.
- Was created in 1999 (implementations: infiniband, RoCE, iWARP)
- Main characteristics:
  - High Bandwidth
  - Low latency
  - Zero copy (CPU offload) – Hardware based data transfers
  - Kernel bypass – Direct access to HW for user-level applications
  - QOS
  - Asynchronous transactions

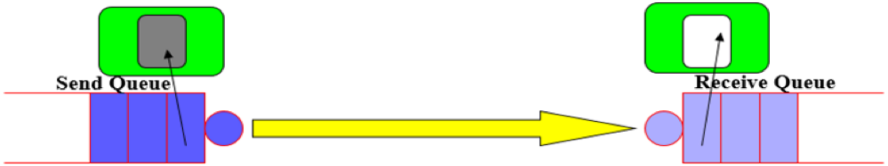


RDMA primitives

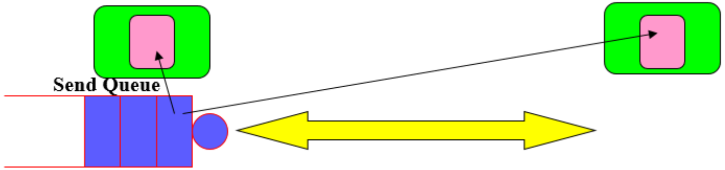
- QP (Queue-Pair) – send & recv queues, with various transport services, used for posting work requests to the HW:
  - RC (Reliable Connected) ~=TCP
  - UD (Unreliable Datagram) ~= UDP
  - UC (Unreliable Connected)
  - RD (Reliable Datagram) – defined by spec but no yet implemented
- CQ (Completion Queue) – used for reporting work requests completions to the host
- MR (Memory Region) – Describes a memory area, with the relevant permissions, accessible for RMDA from the device.
- PD (Protection Domain) – provides an association between QPs/MRs/MWs for enabling and controlling HCA access to host memory.
- Programming Model – Verbs

RDMA operations

- **Messaging:**
  - **RECV**: post a buffer for incoming data
  - **SEND**: send a buffer to a remote peer (who posted a RECV buffer for it in advance)

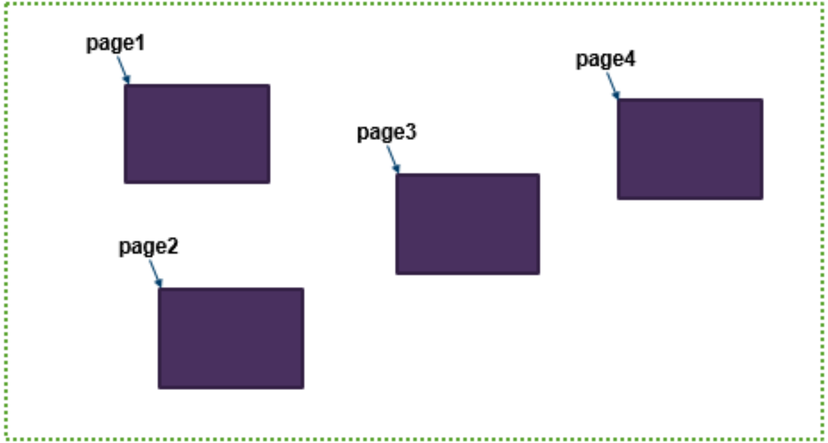


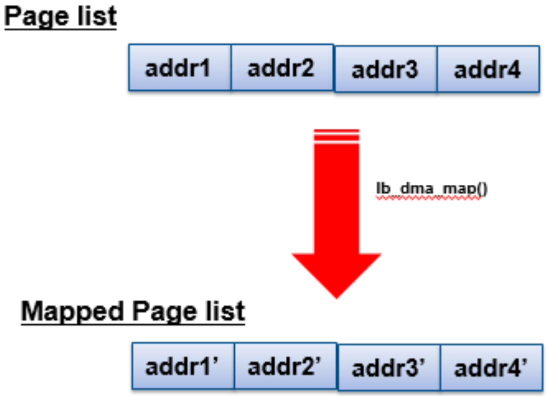
- **REG_MR**: memory registration for RDMA operations
- **One-sided:**
  - **RDMA_WRITE**: copy a local buffer (described by MR-L) to a remote buffer (MR-R)
  - **RDMA_READ**: copy a remote buffer (described by MR-R) to a local buffer (MR-L)



Memory registration

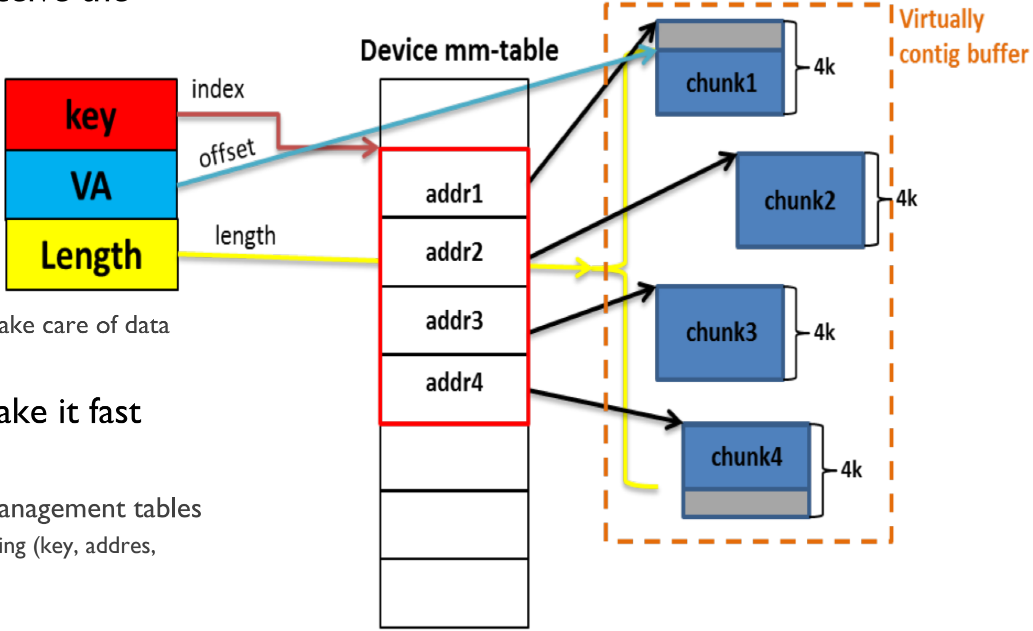
- So why we need to register memory ?
  - Avoid data corruption
  - Protect from unauthorized access
  - **Map the addresses to DMA language (PCI space)**



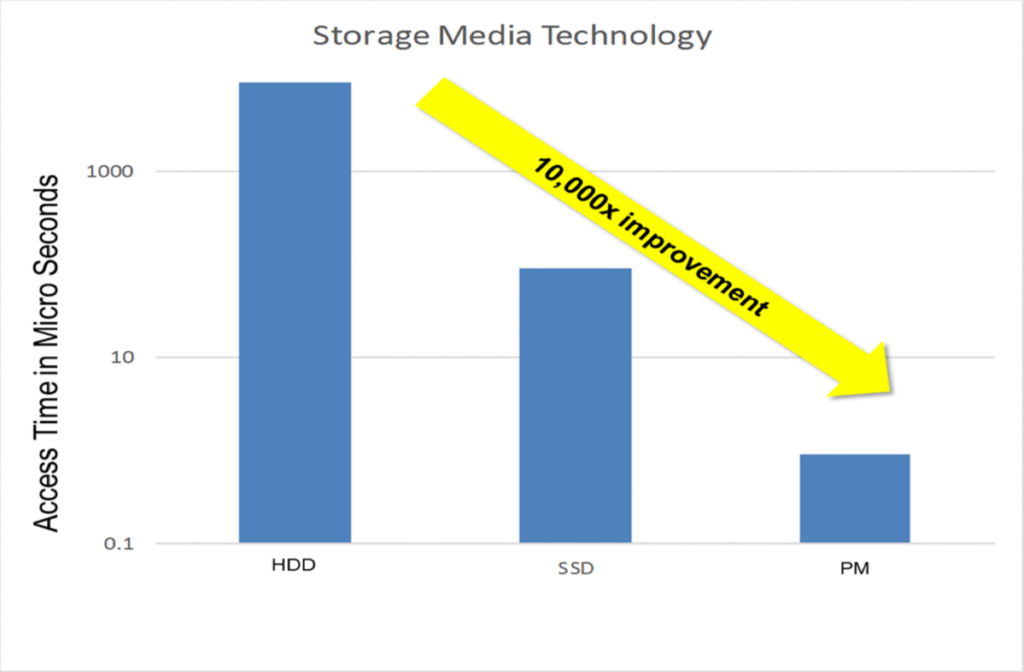


Use Fast Memory Registration

- Memory registration is a heavy operation (allocations, pinning, translation, FW commands …)
- In the kernel (iSER/SRP/NVMe-oF…) we always receive the buffer from the user.
  - User allocate a buffer
  - User open a file (block device or file system)
  - User call syscall read/write(buffer)
  - → the ULP sees this as a bio or as an sg list.
    - Pinning the buffer was done by the block layer (no need to take care of data corruption)
- One should use a special work request (WR) to make it fast
  - Use pre-allocated MR
  - Only DMA map the SG list and update the HW memory management tables
    - Using ib_sge object that represents a virtually contiguous buffer using (key, addres, length) tuple

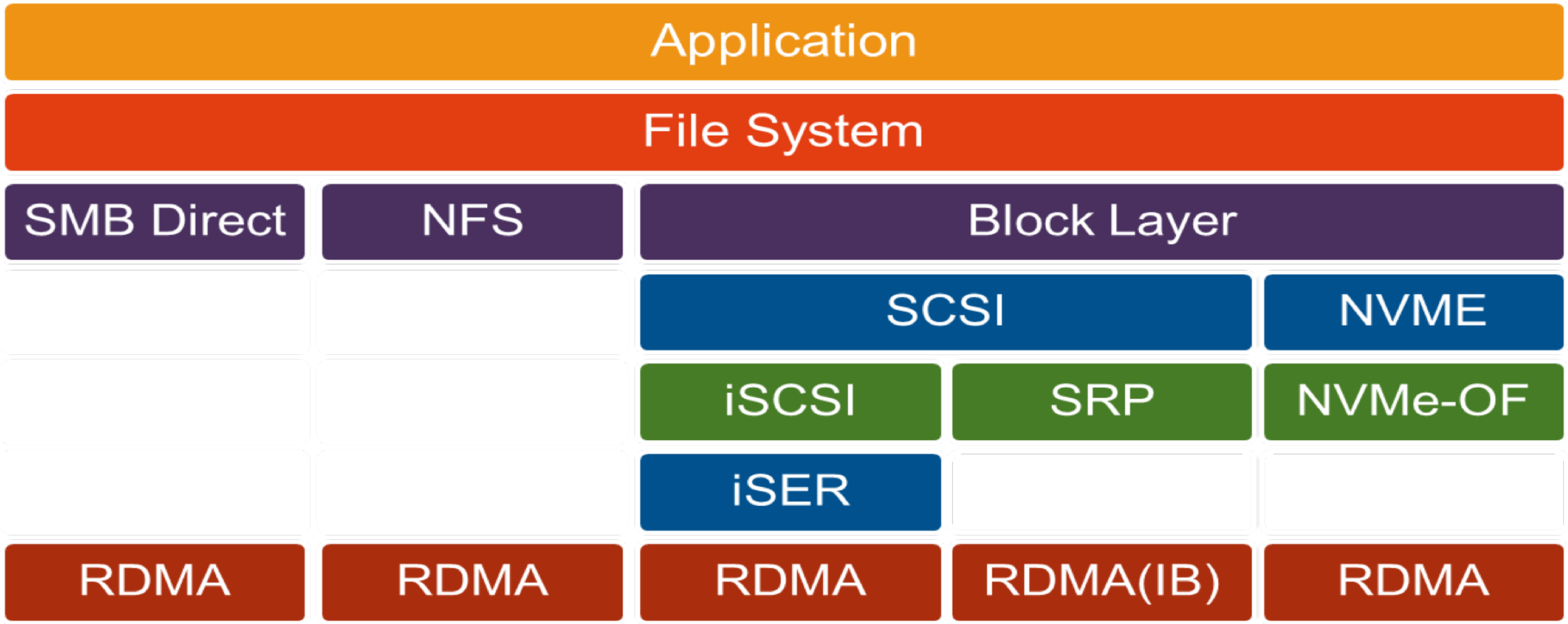


Why Should We Care About RDMA ?



Because Faster Storage Needs a Faster Network (not only in HPC) !!!

Variety of RDMA Storage Protocols



Application

File System

SMB Direct | NFS | Block Layer

SCSI | NVME

iSCSI | SRP | NVMe-OF

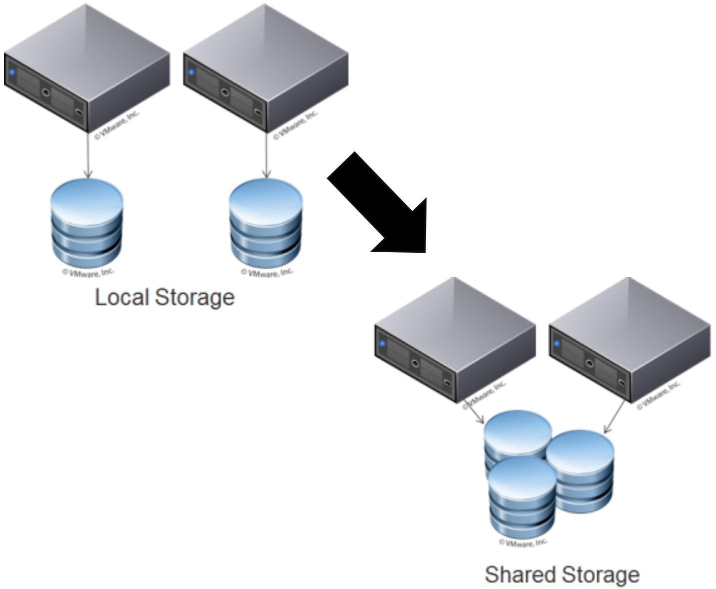
iSER

RDMA | RDMA | RDMA | RDMA(IB) | RDMA

Similar Exchange Model

Protocol Deep Dive – NVMe/NVMe-oF

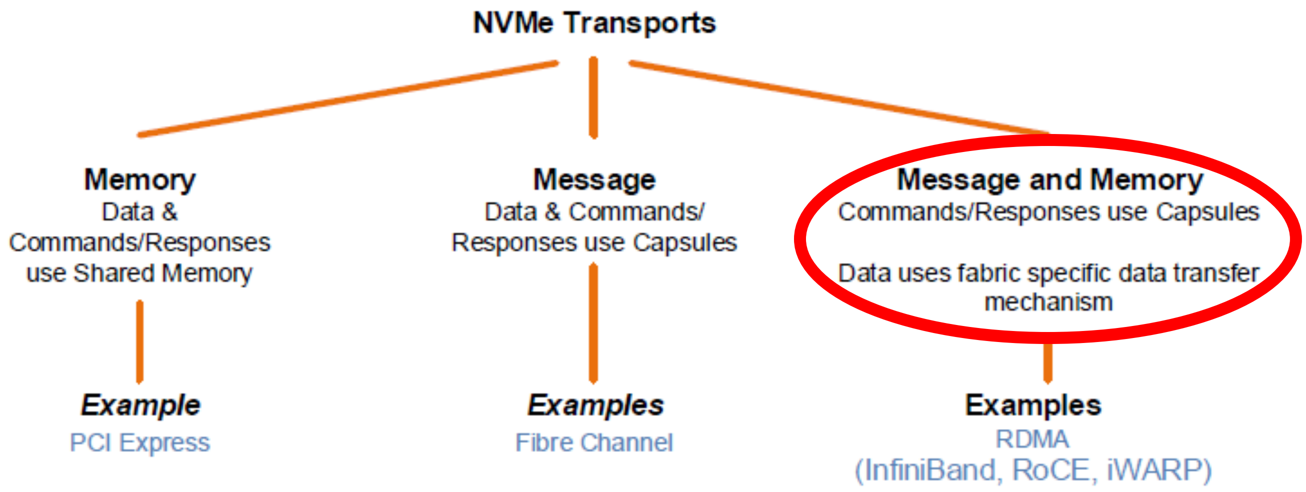
- Share NVMe SSDs with multiple servers
  - Better utilization, capacity, rack space, power
  - Scalability
  - management
- NVMe over Fabrics standard
  - Version 1.0 completed in June 2016
  - High performance access to remote SSD (not only SSD)
  - RDMA protocol is part of the standard (e.g. keyed SGLs)
    - Also FC and TCP (in progress)



9

NVMe-oF Exchange Model

Figure 1: Taxonomy of Transports



NVMe Transports

Memory
Data & Commands/Responses use Shared Memory

Example
PCI Express

Message
Data & Commands/ Responses use Capsules

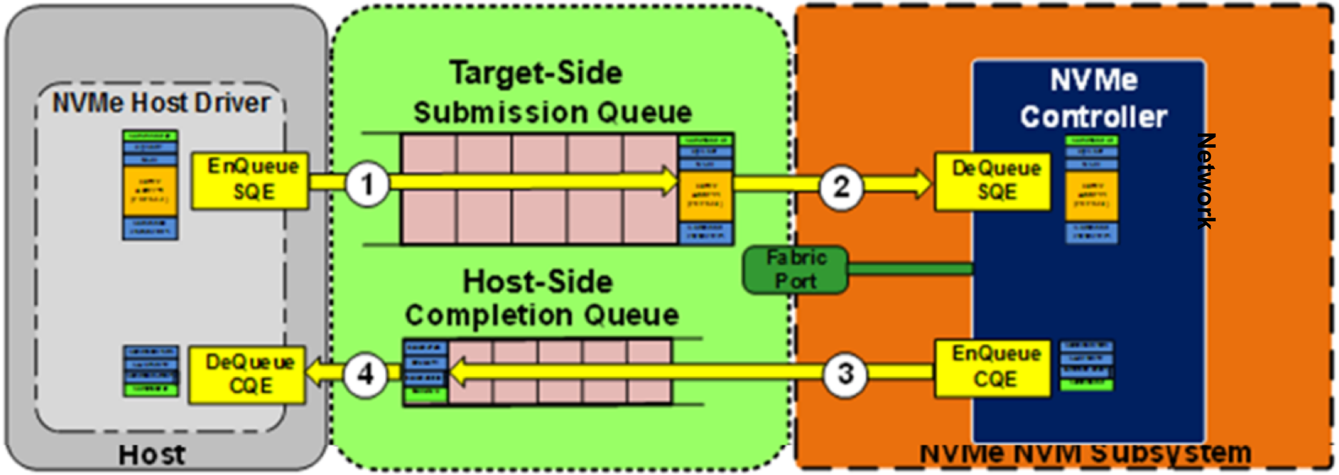
Examples
Fibre Channel

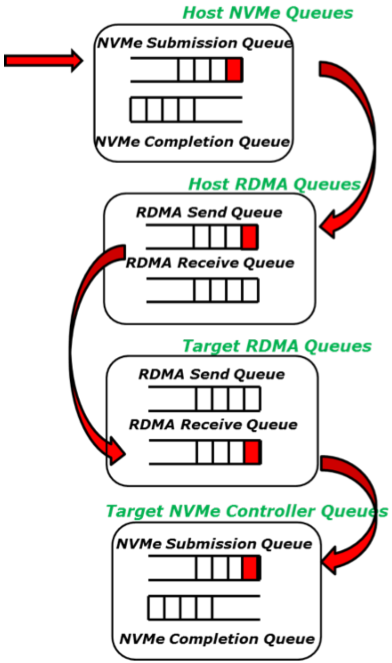
Message and Memory
Commands/Responses use Capsules

Data uses fabric specific data transfer mechanism

Examples
RDMA (InfiniBand, RoCE, iWARP)

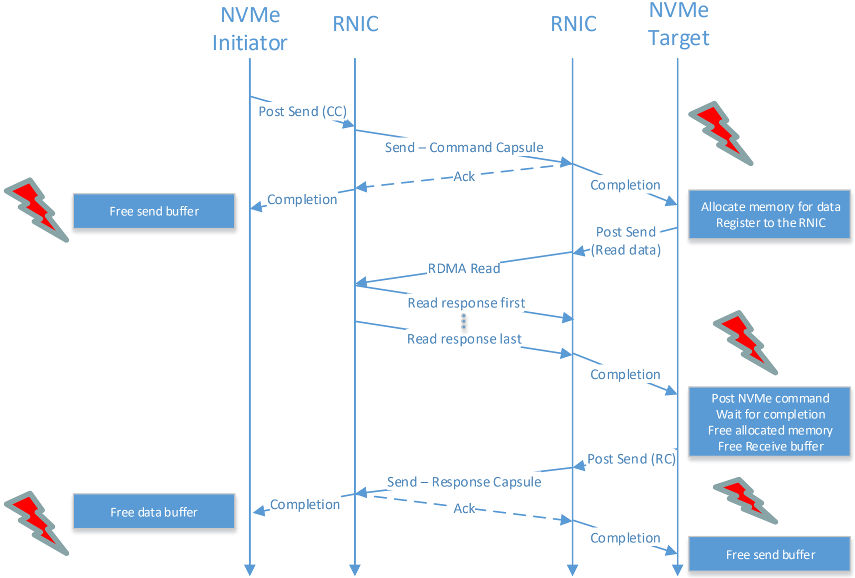
NVMe and NVMe-oF/RDMA Fit Together Well





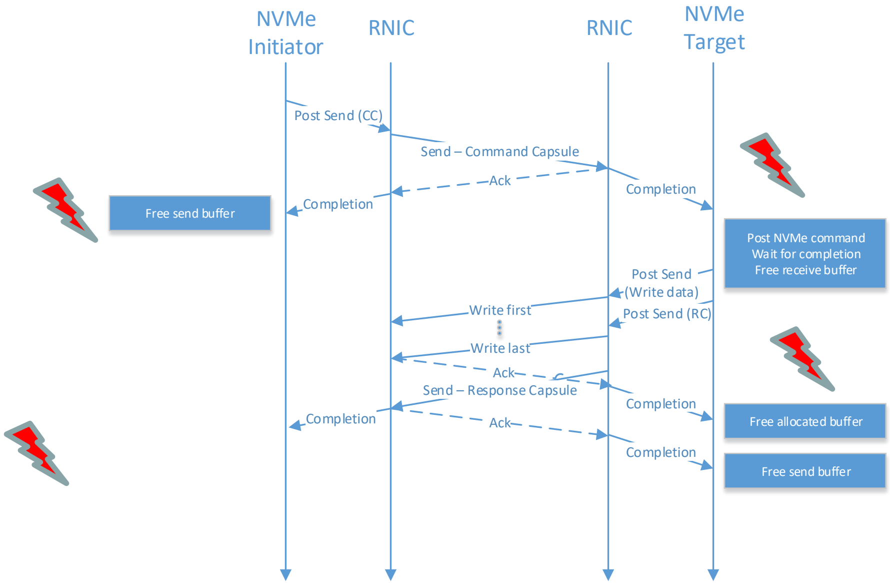
Example: NVMe-oF Protocol (Write)

- Host
  - Register Memory (get MR)
  - Post SEND carrying Command Capsule (CC) that contains SQE (Submission Queue Entry) and keyed SGL.
- 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 Response Capsule (RC)
    - Free memory
  - Upon SEND Completion
    - Free CC and completion resources



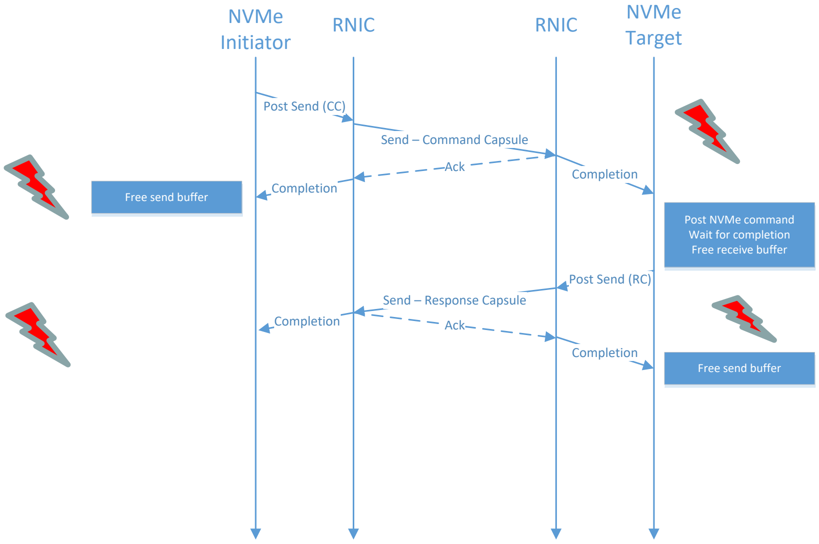
Example: NVMe-oF Protocol (Read)

- Host
  - Register memory (get MR)
  - Post SEND carrying Command Capsule (CC) that contains SQE (Submission Queue Entry) and keyed SGL.
- 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-oF Response Capsule (RC)
  - Upon SEND Completion
    - Free memory
    - Free CC and completion resources



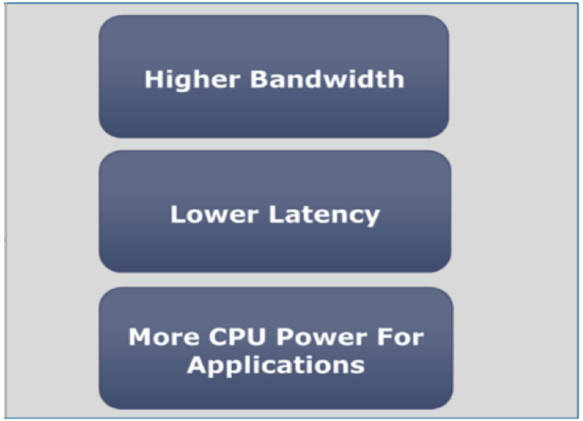
Example: NVMe-oF Protocol (Write IN-Capsule)

- Host
  - Post SEND carrying Command Capsule (CC) that contains SQE (Submission Queue Entry) and data.
  - Useful for small IO (Currently up to 4k)
- Subsystem
  - Upon RCV Completion
    - Allocate Memory for Data
  - Upon SSD completion
    - Send NVMe-oF Response Capsule (RC)
    - Free memory
  - Upon SEND Completion
    - Free RC and completion resources



Challenges ?!

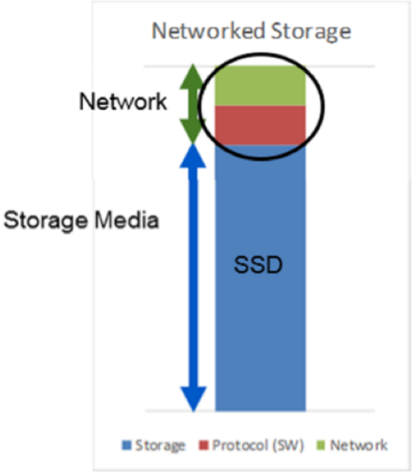
- Performance
  - Same as DAS
- Reduce memory foot print
  - Share resources
- Scale
  - Data is growing
  - We must have a ultra fast network
- Save $$$
  - Build systems with cheaper CPU/HW
- Save CPU cycles
  - Offload data path by HW
- High availability
  - multipathing



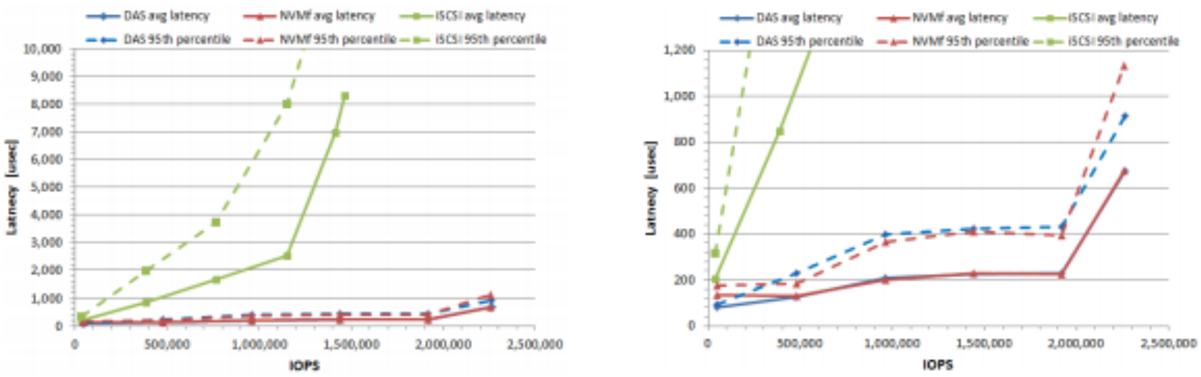
Higher Bandwidth

Lower Latency

More CPU Power For Applications



NVMe-oF/RDMA has Great Performance !



(a) Full view

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

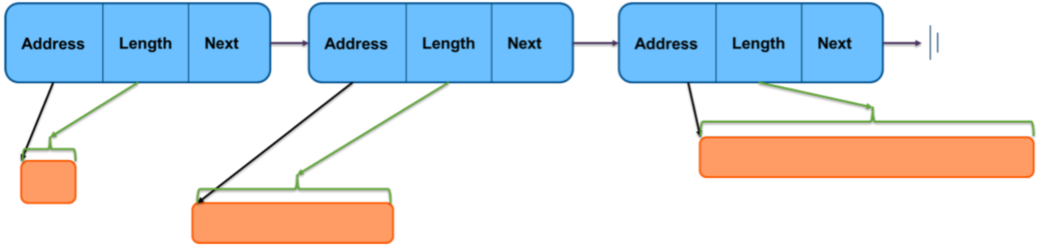
Figure 6: Average and tail latencies of DAS, NVMf, and iSCSI for different request loads. Since iSCSI saturates early and exhibits significantly higher latencies, we show both the full range view (a), and a zoom-in on DAS and NVMf (b).

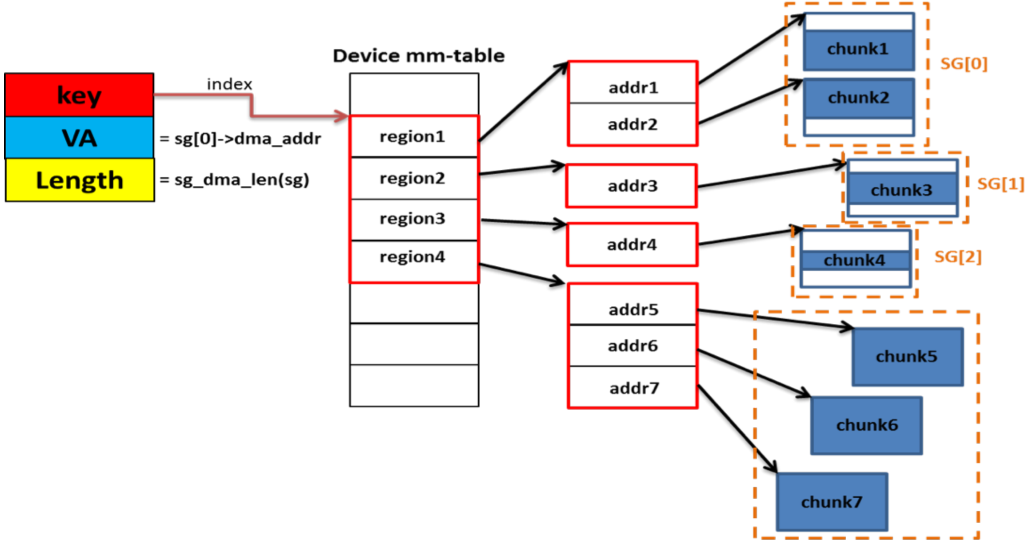
Can we do better ?

- Yes we can !!
- Currently WIP in Linux
  - Interrupt/completion moderation (AKA coalescing):
    - A technique in which events would normally trigger a HW interrupt are held back, either until a certain amount of work is pending, or a timeout timer triggers
  - Register non contiguous buffer using indirect MR
    - The user can provide an iovec where each entry has its own length
    - We can’t assume user buffers consists of full pages
    - We don’t want the block layer to use bounce buffers – save CPU cycles
    - Use HW that supports indirection in MM table

ConnectX-4 (and above) devices supports indirection

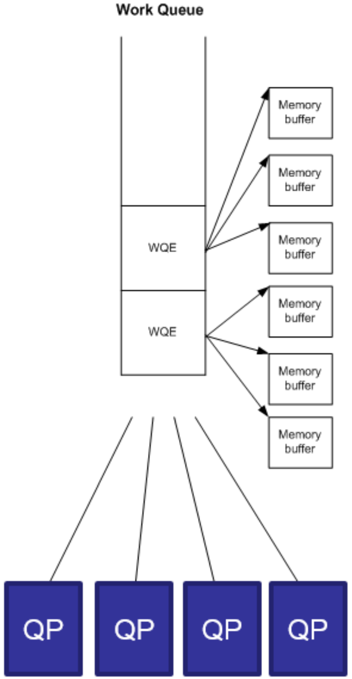
- Implemented in iSER
- SRP/NVMe-oF patches submitted
  - Use IB_MR_TYPE_SG_GAPS
- **Please Try it !!**





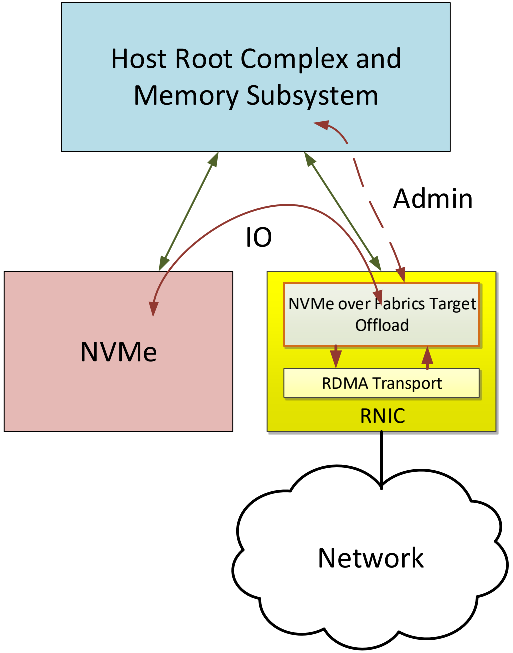
Reducing Memory foot print by using SRQs

- SRQ stands for Shared Receive Queue
- QPs/Connections are cheap, Receive buffers are not !
- Solution: Share receive buffering resources between QPs
  - According to the parallelism required by the application
  - Locality of completions
  - scalability
- NVMe-oF implementation today uses 1 SRQ per HCA
  - Lock contention in the data path
  - No parallelism
  - Better to use SRQ per core or per completion vector (MSI-X)
- **We have submitted patches to fix performance in Linux – please try!**

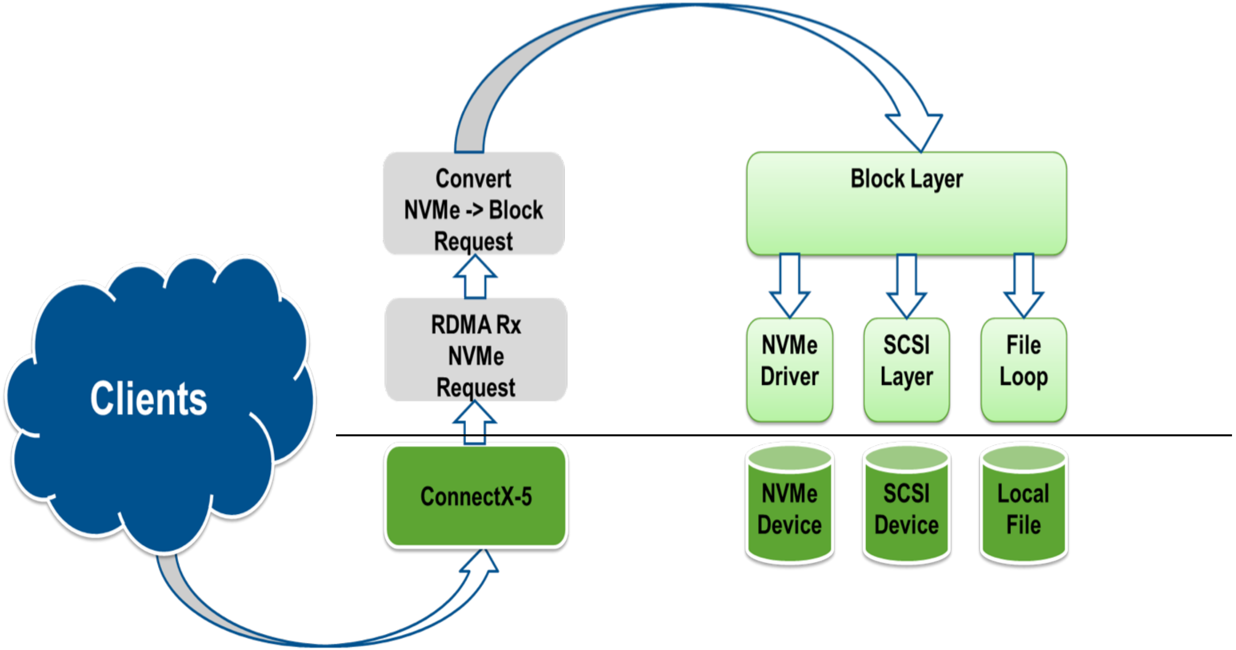


Save CPU by using NVMe-oF Target Offload

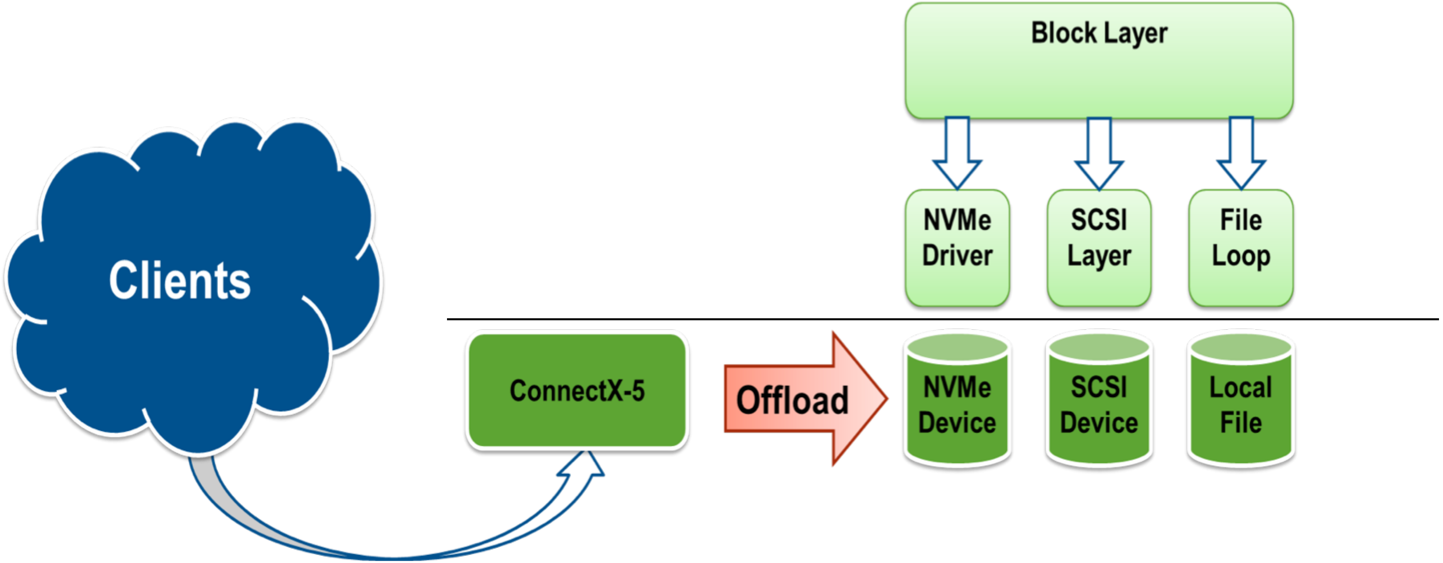
- NVMe-oF is built on top of RDMA
  - Transport communication in hardware
- NVMe-oF target offload enable the NVMe hosts to access the remote NVMe devices w/o **any** CPU processing
  - By offloading the entire NVMe-oF data path
  - Encap/Decap NVMe-oF <-> NVMe is done by the adapter with 0% CPU
    - CPU is available for other applications
- Easy configuration: “echo 1 > .../subsystems/<subsys>/attr_offload”
- Admin operations are maintained in software
- IOPs with 0% CPU (512B IO read)
  - Connectx-5 – **1.0-1.2 MIOPs**
  - Bluefield SoC – **7.5 MIOPs**
- Upstream submission – TBD
  - Currently available in MLNX_OFED package
  - Linux fork is available: https://github.com/Mellanox/NVMEoF-P2P/
- Save $$$ - NVMe-oF target systems can use cheaper CPUs


Host Root Complex and Memory Subsystem
Admin
IO
NVMe
NVMe over Fabrics Target Offload
RDMA Transport
RNIC
Network

NVMe-oF Target non-offload – data path



NVMe-oF Target offload – data path



RDMA Block based storage protocols in Linux

Feature	NVMe-oF	iSER	SRP
Fast memory registration	V	V	V
Indirect memory registration	WIP	V	WIP
SRQ	V		V
SRQ per core	WIP		
Remote Mkey invalidation	V	V	
Block MQ	V		V
RoCE support	V	V	WIP
User space tools	nvmecli/nvmetcli	iscsiadm/targetcli	srp_daemon/targetcli
High availability	dm-multipath/nvme-multipath	dm-multipath	dm-multipath
T10-PI		V	
User space open source target	SPDK	TGT	

Thanks !

maxg@mellanox.com