OPENFABRICS ALLIANCE

14th ANNUAL WORKSHOP 2018

NVMf based Integration of Non-volatile Memory in a Distributed System - Lessons learned

Jonas Pfefferle, Bernard Metzler, Patrick Stuedi,
Animesh Trivedi and Adrian Schuepbach

IBM Zurich Research

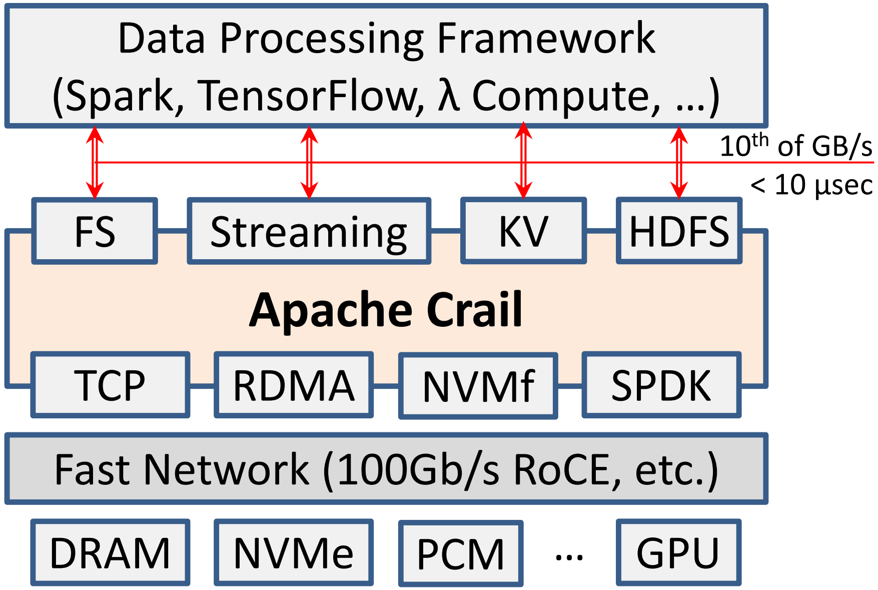
April 2018

Outline

- Target Distributed System: Apache Crail
- NVMf
  - Basic operation
  - Available implementations
- The Crail NVMf Tier
  - Integration with Crail
  - Implementation details
  - A new NVMf host library
- Measurements
  - Microbenchmarks
  - TeraSort
  - TPC-DS workload
- Summary & Outlook

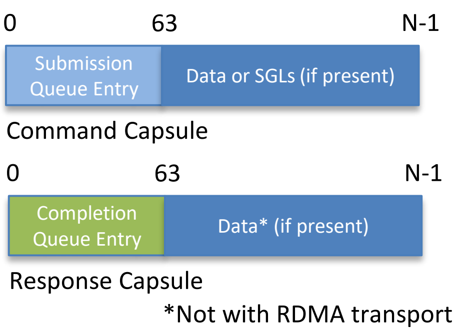
The Target System: Apache Crail (Incubating)

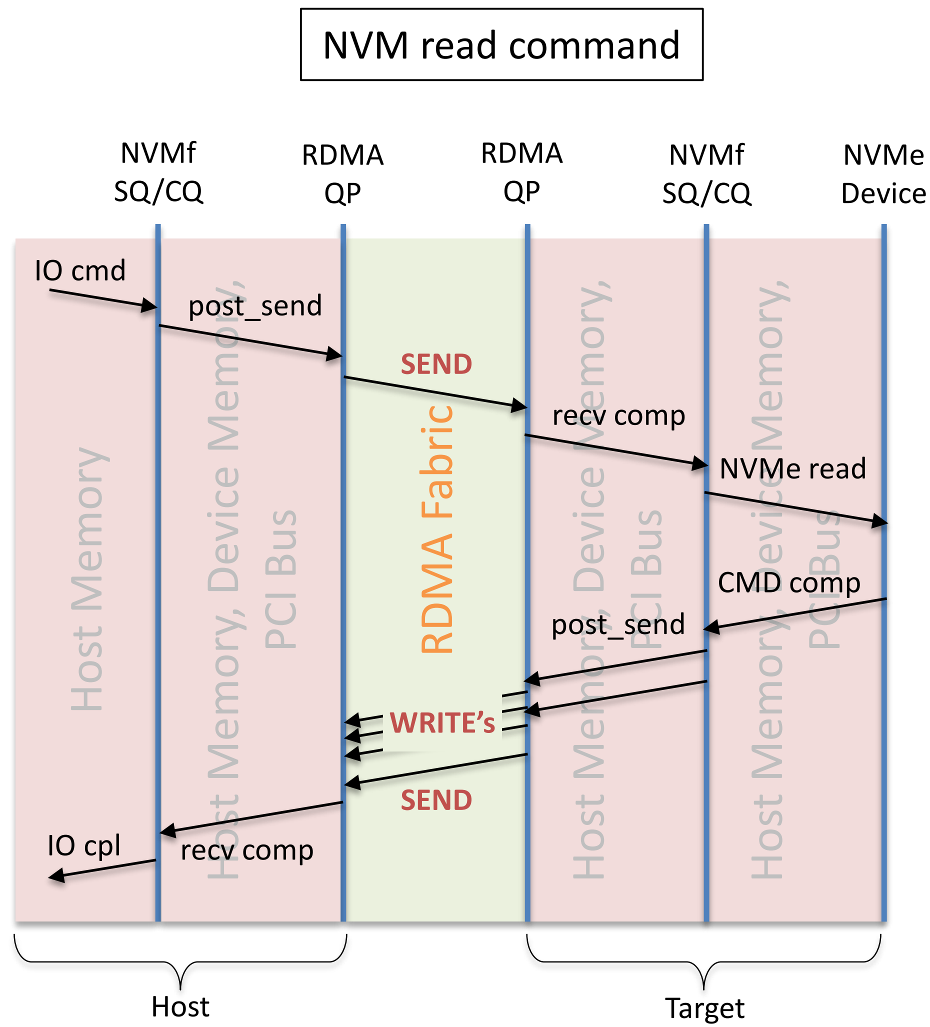
- Targets I/O acceleration for Big Data frameworks due to **user level** and **asynchronous I/O**
- Implements efficient distributed store for ephemeral data at native hardware speeds
- Unifies isolated, incompatible efforts to integrate high performance I/O with BigData processing frameworks
- Enables dramatically faster job completion
- Applicable to Spark, Flink, TensorFlow, Caffee, …
- Flexibly makes best use of available I/O infrastructure
- Apache Incubator Project (since 11/17)



NVMf in a Nutshell

- **Extends low latency NVMe interface over distance**
  - Retains BW and latency (< 5us additional delay)
  - Allows to retain user level I/O capability
  - Message based NVMe operations
  - Host/Target model
- **Defined transports: RDMA (IB, RoCE, iWarp), or FC**
  - Mapping of I/O Submission + Completion Queue to RDMA QP model
- **Fabrics + Admin Commands**
  - Discover, Connect, Authenticate, …
- **I/O Commands**
  - SEND/RECV to transfer Command/Response Capsules (+ optionally data)
  - READ/WRITE to transfer data



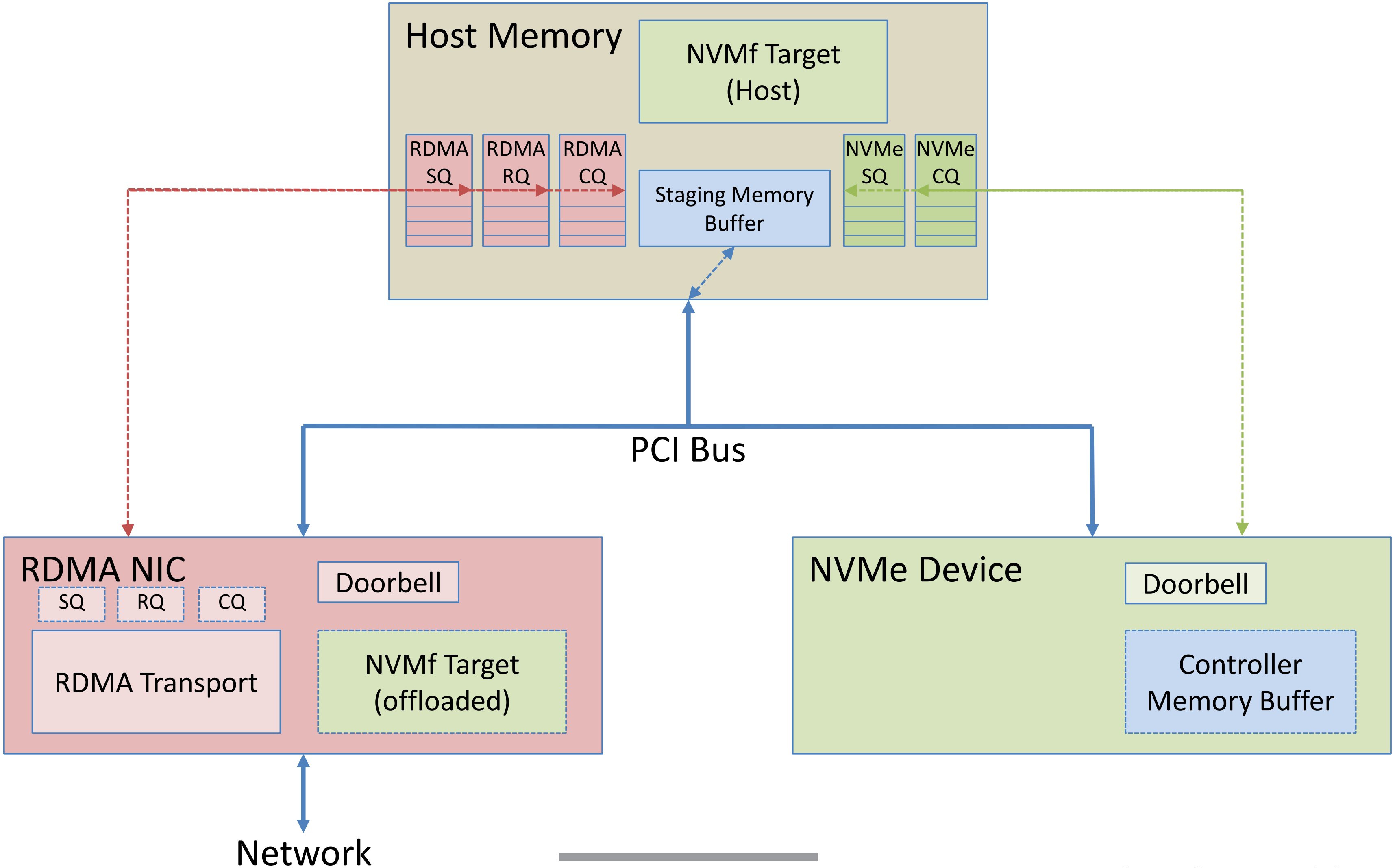


NVMf Implementations available

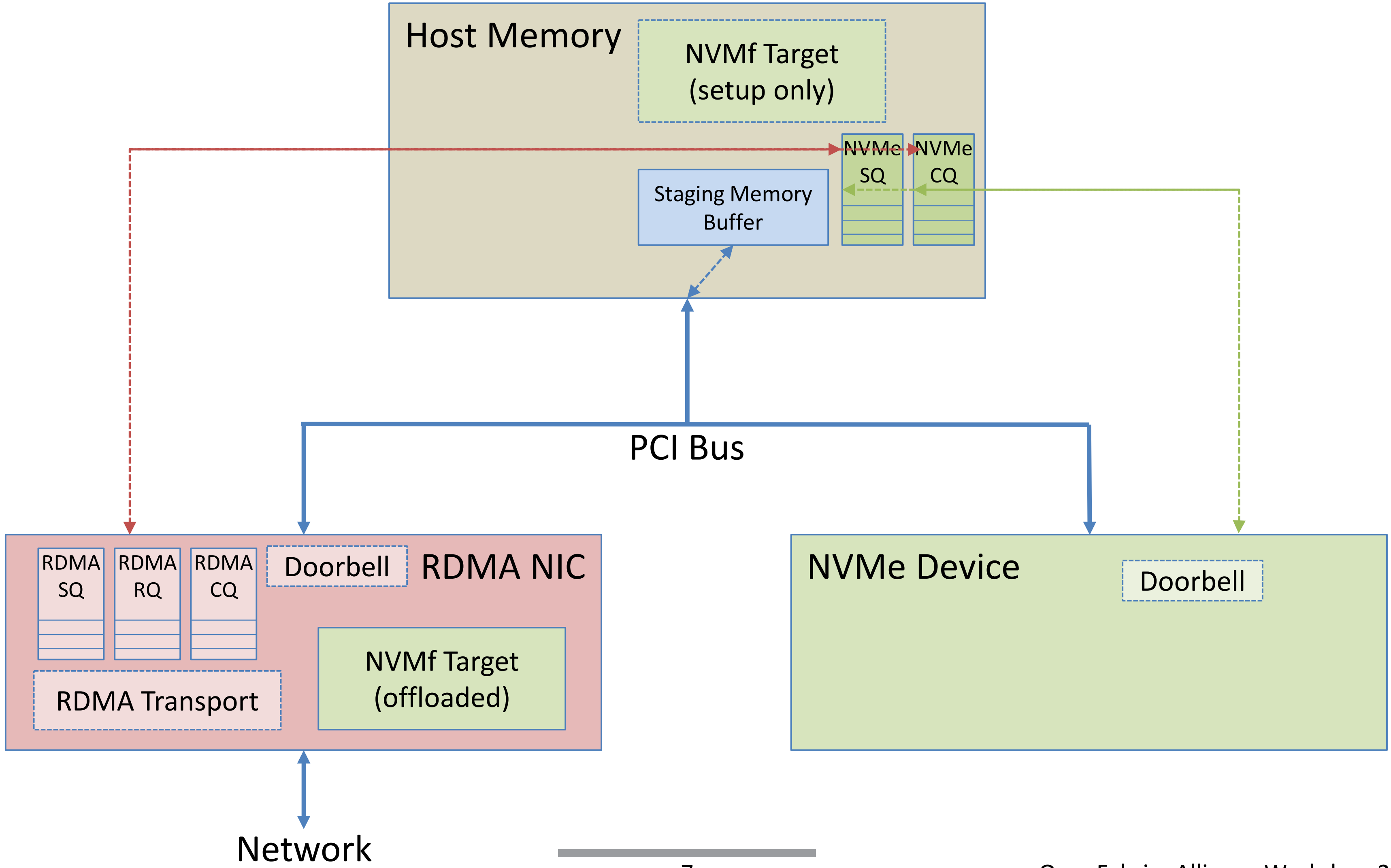
	Host	Target
User Level	SPDK nvme jNVMf*	SPDK nvmf_tgt
OS Kernel (Block Layer)	(Linux nvme-rdma)	Linux nvmet-rdma
Hardware Offload		Mellanox CX5 nvmet-rdma

*https://github.com/zrlio/jNVMf

Landscape of NVMf Implementations (Target)

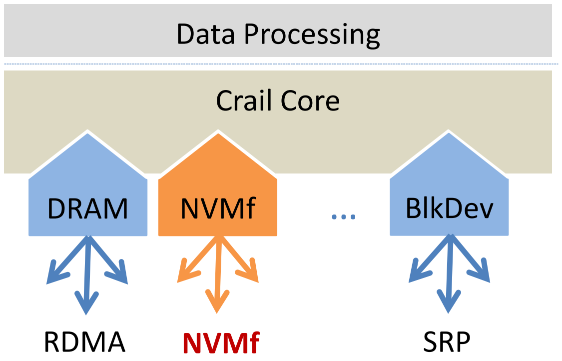


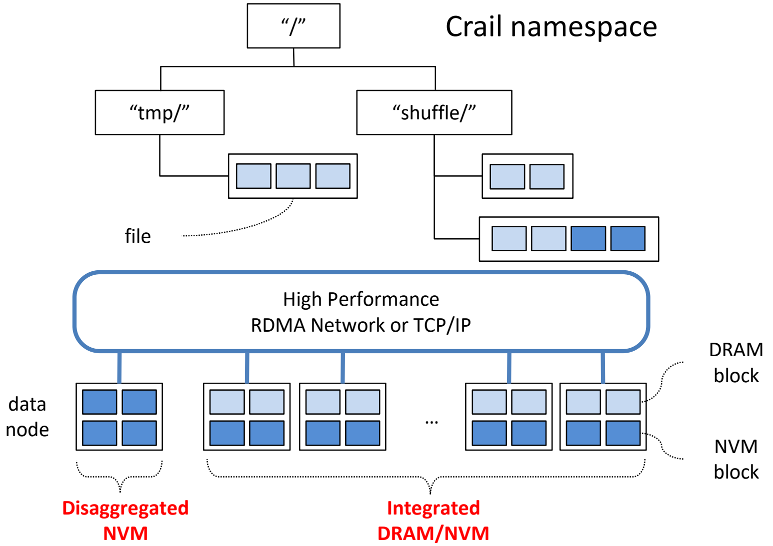
Mellanox Target Offloading (w/o CMB)

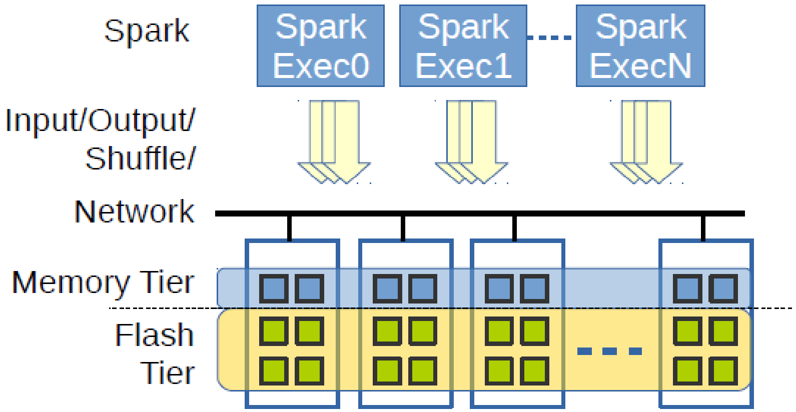


Crail: The NVMf Tier

- **Another fast storage tier in Crail**
  - I/O to distributed NVMe at its native speed
  - Uses NMVf for remote NVM access
  - User level I/O at client side
- **Flexibility**
  - Storage disaggregation or integration
  - Spill over from DRAM, or dedicated tier
- **NVMf integration**
  - Client library
  - Data node implementation





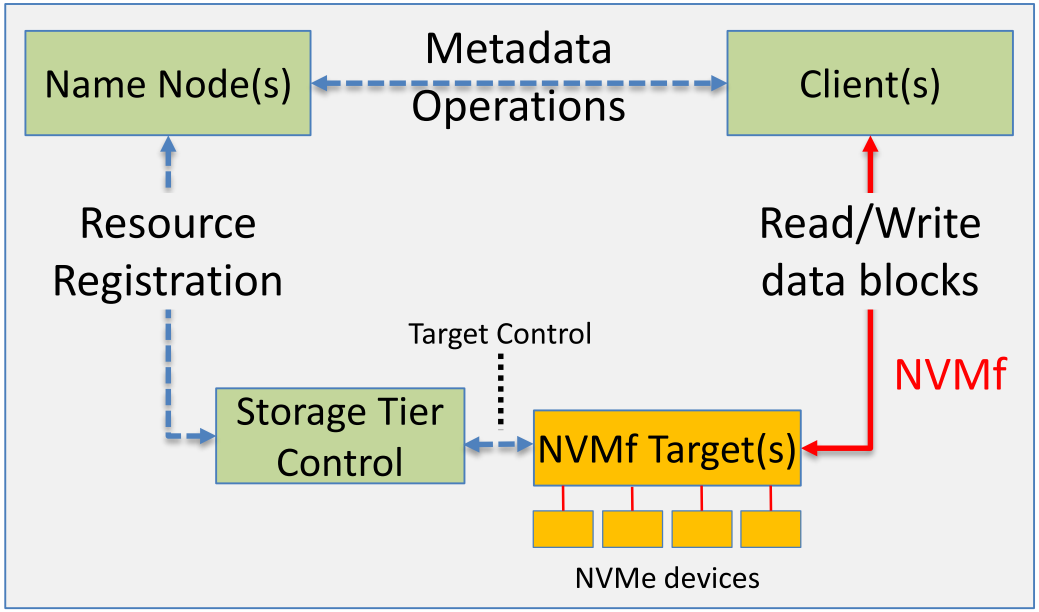


Putting Things together – NVMf in Crail

- **Crail Storage Tier Control**
  - Connects to any NVMf Target code
    - SPDK, Linux kernel, Offload
  - Registers targets NVMe blocks with NameNode(s)
  - Not in the block read/write NVMf fast path client <-> target

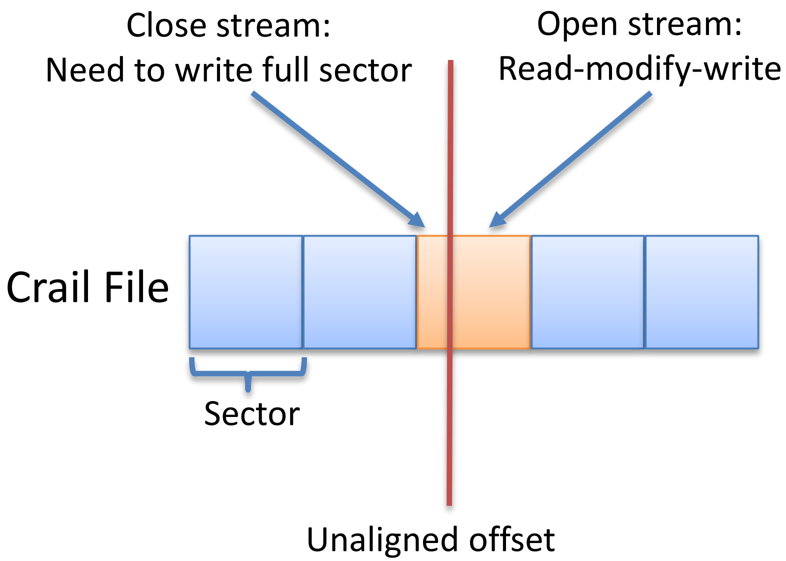
- **Crail Name Node**
  - Maintain storage resources
    - Get blocks registered
    - Give out blocks and back
    - May write logs

- **Crail NVMf Client**
  - Deploys NVMf Host code
    - SPDK host library, or
    - jNVMf host library
  - Connects with NVMf Target(s)
  - Gains NVMe block access info via NameNode
  - Allows for application sub-block read/write access
    - Must implement RMW



Crail NVMf Tier Design Discussion

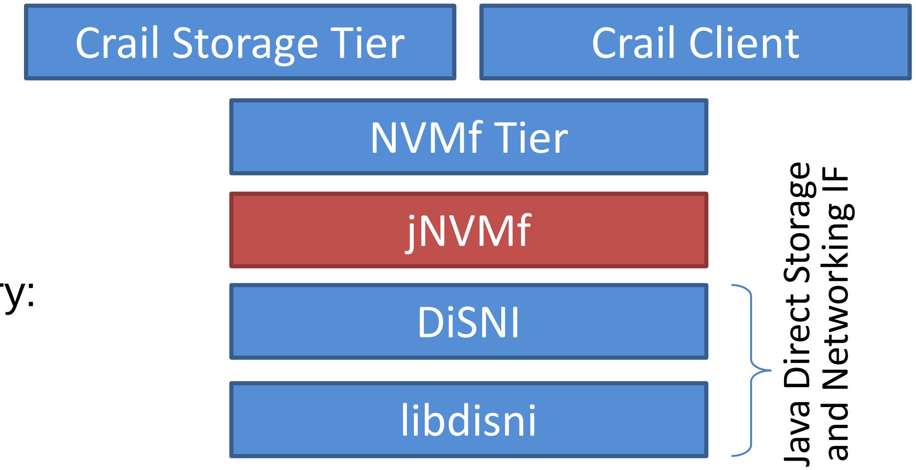
- NVMf tier must follow semantics of all Crail storage tiers:
  - Provide bytegranular access to data store, but
  - Must cope with block only media access
  - RMW implementation
    - NVMf does not support byte offset access for RDMA transport (no bit bucket type)
    - Becomes client activity above NVMf host
    - Adds another full network round trip



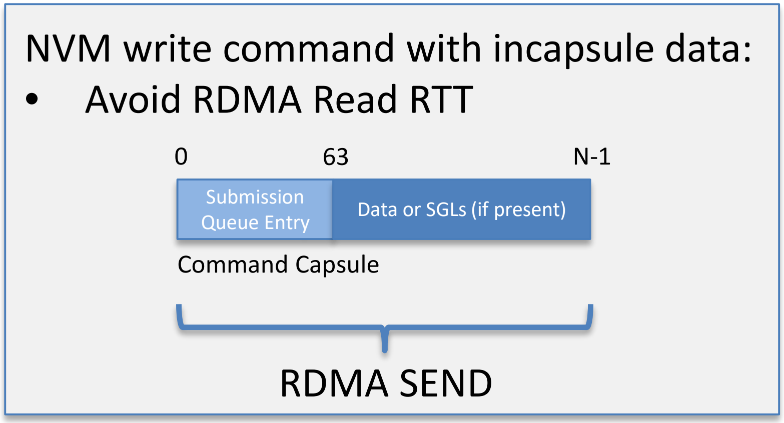
- Try to avoid unaligned accesses
  - ✓ Append only
  - ✓ Use buffered streams which internally aligns to block and buffer sizes
  - But cannot avoid unaligned access when closing and reopening buffered stream

jNVMf: Yet another NVMf Host Library?

- **Dependencies and Stability**
  - SPDK → DPDK
  - Memory management
  - Bugs
  - DPDK not meant to be used as a library:
    - “owns” application
    - No shared library build



- **Clean slate approach: jNVMf**
  - Simpler memory management
  - Incapsule data support (write accel.)
  - Dataset management support
  - RDMA inline SEND support (read accel.)
  - Reduced JNI overhead
  - Easier to integrate with Crail



NVMe Devices we deployed

	NVMe Version	Read Latency (4K)	Write Latency (4K)	Read IOPS (4K)	Write IOPS (4K)	Read Throughput	Write Throughput
Samsung 960Pro	1.2	53.3us	7.8us	483K	386K	3186MB/s	1980MB/s
Intel 3D XPoint 900p	1.2	6.3us	7.2us	580K	557K	2586MB/s	2195MB/s

Many advanced NVMe features not supported:

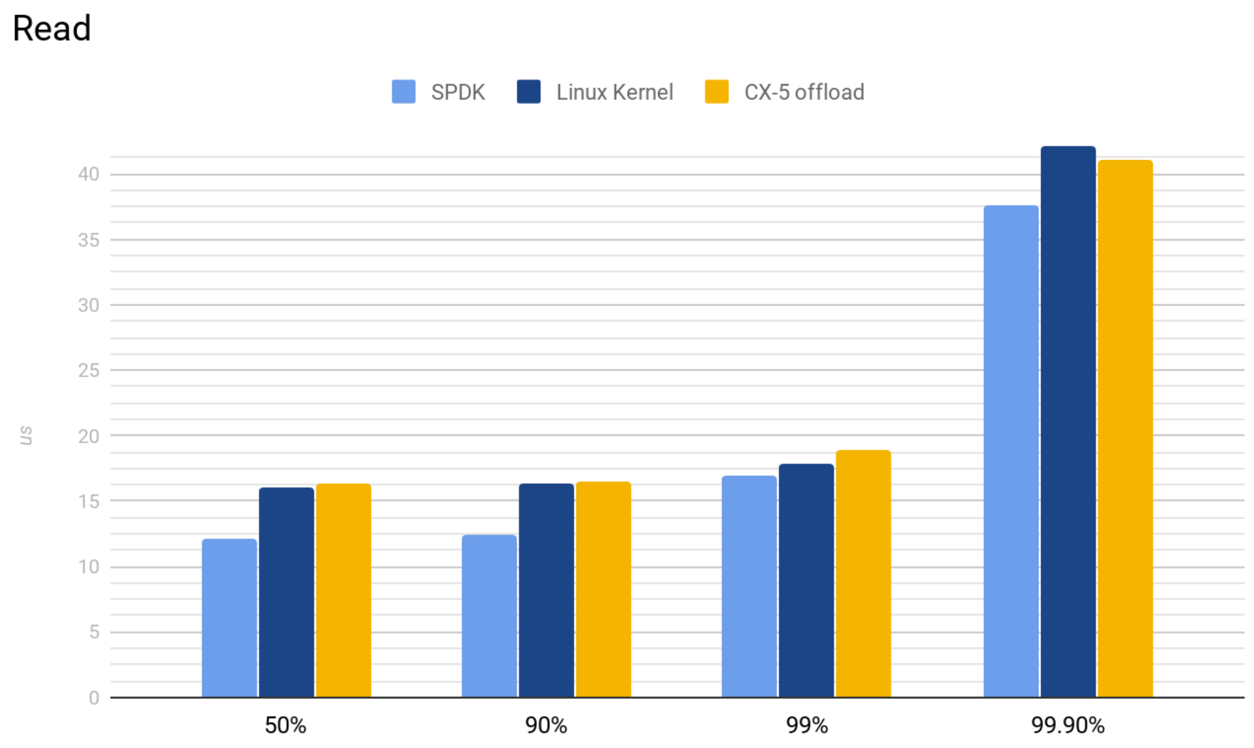
- Scatter-gather list
- Arbitration mechanisms
- Namespace management (or only 1 namespace is supported)
- Controller memory buffer

NVMf Latency 4k (Read)

- Intel 900p device
- SPDK host

- SPDK target fastest
- Linux kernel and CX-5 offload similar delay
- Zero CPU load for CX-5 target (CPU load not shown)

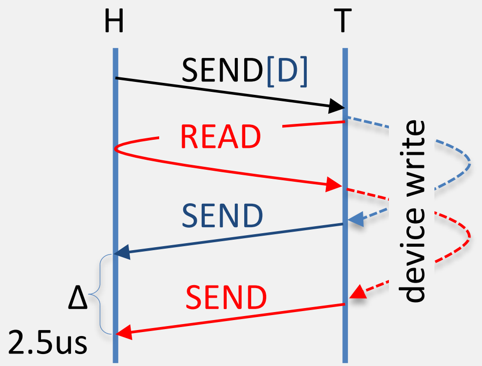
Comparing NVMf targets: SPDK, Linux Kernel, CX-5 offload



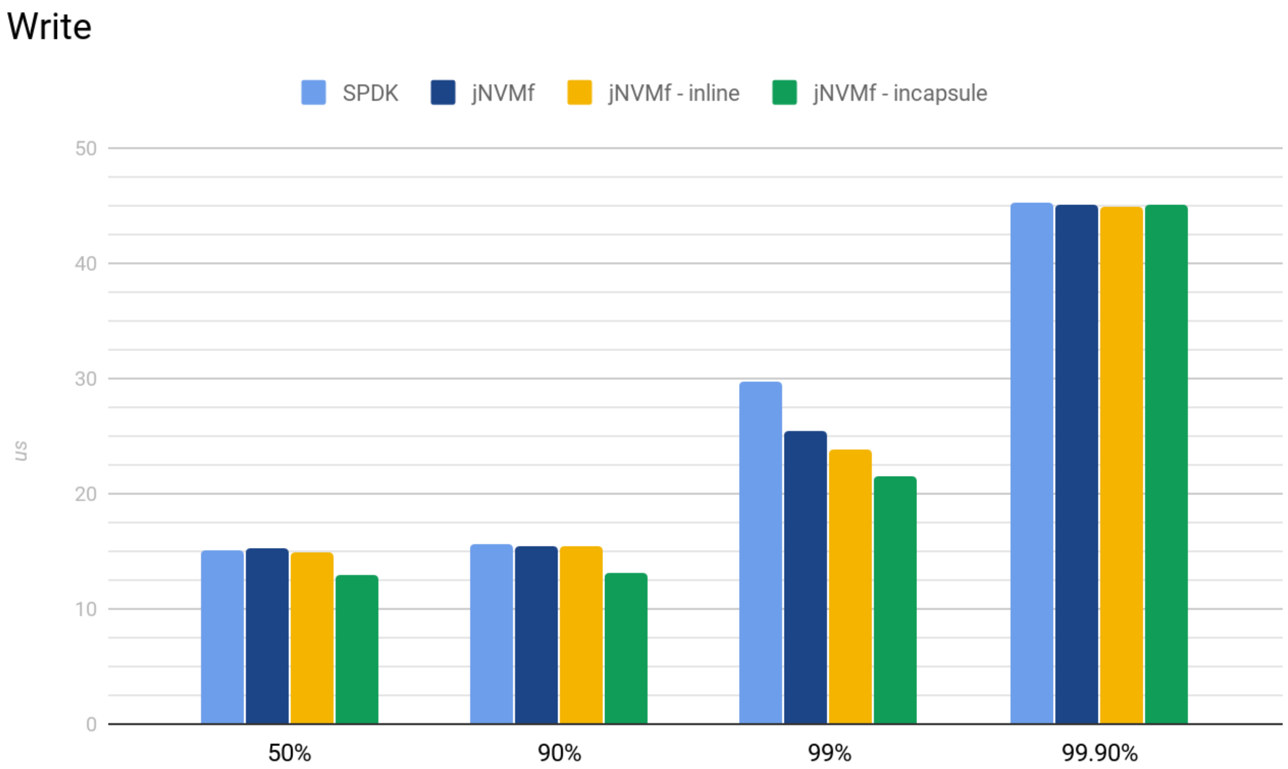
SPDK vs jNVMf Latency 4k (Write)

- SPDK NVMf target
- Intel 900p device

- SPDK client and jNVMf show similar performance
- No 'Java penalty'
- Enabling RDMA inline Send gives marginal advantage
- Enabling Incapsule data saves 1 RTT



Comparing NVMf host libraries: SPDK, jNVMf

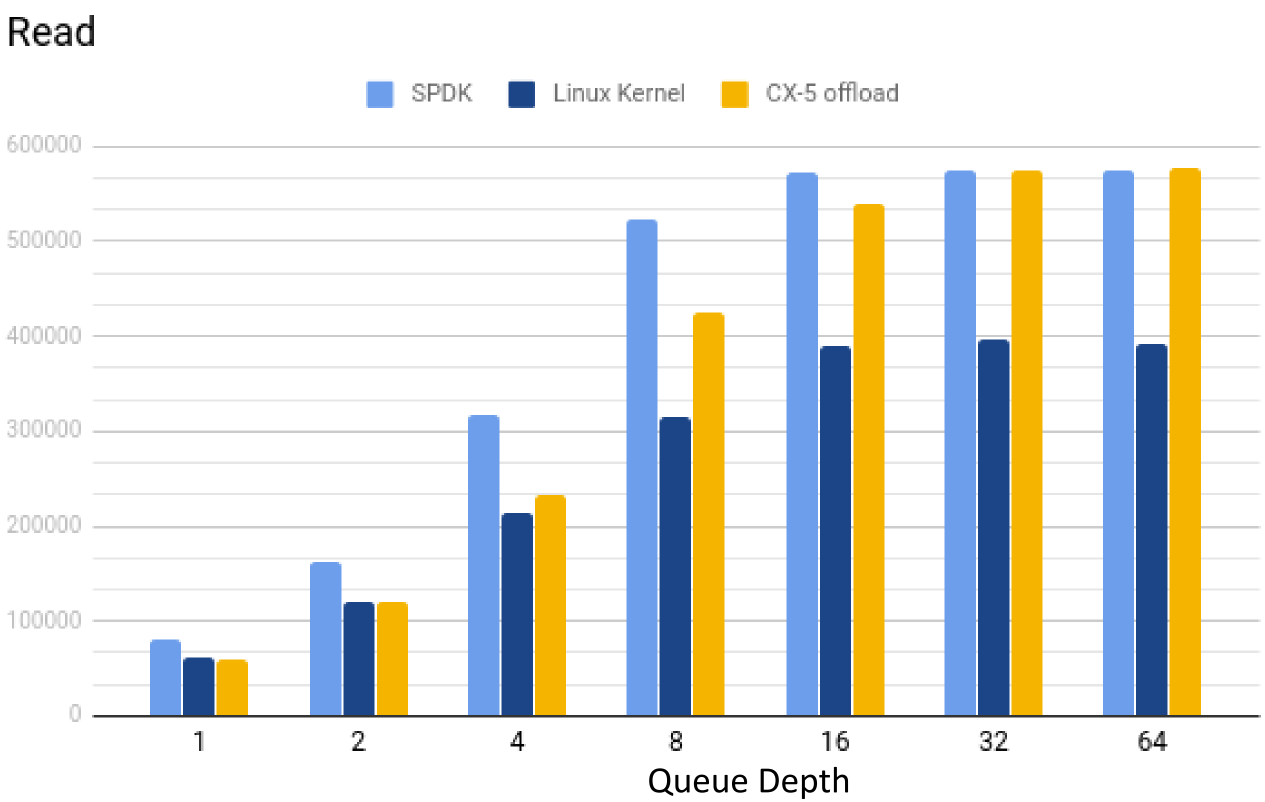


NVMf IOPS 4k (Read)

- Intel 900p device
- SPDK client

- CX-5 offload and SPDK target saturate device
- Kernel targets flattens out at ~400K IOPs
- Zero CPU load for CX-5 target (CPU load not shown)

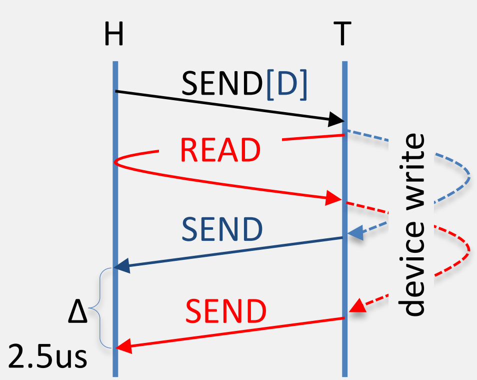
Comparing NVMf targets: SPDK, Linux Kernel, CX-5 offload



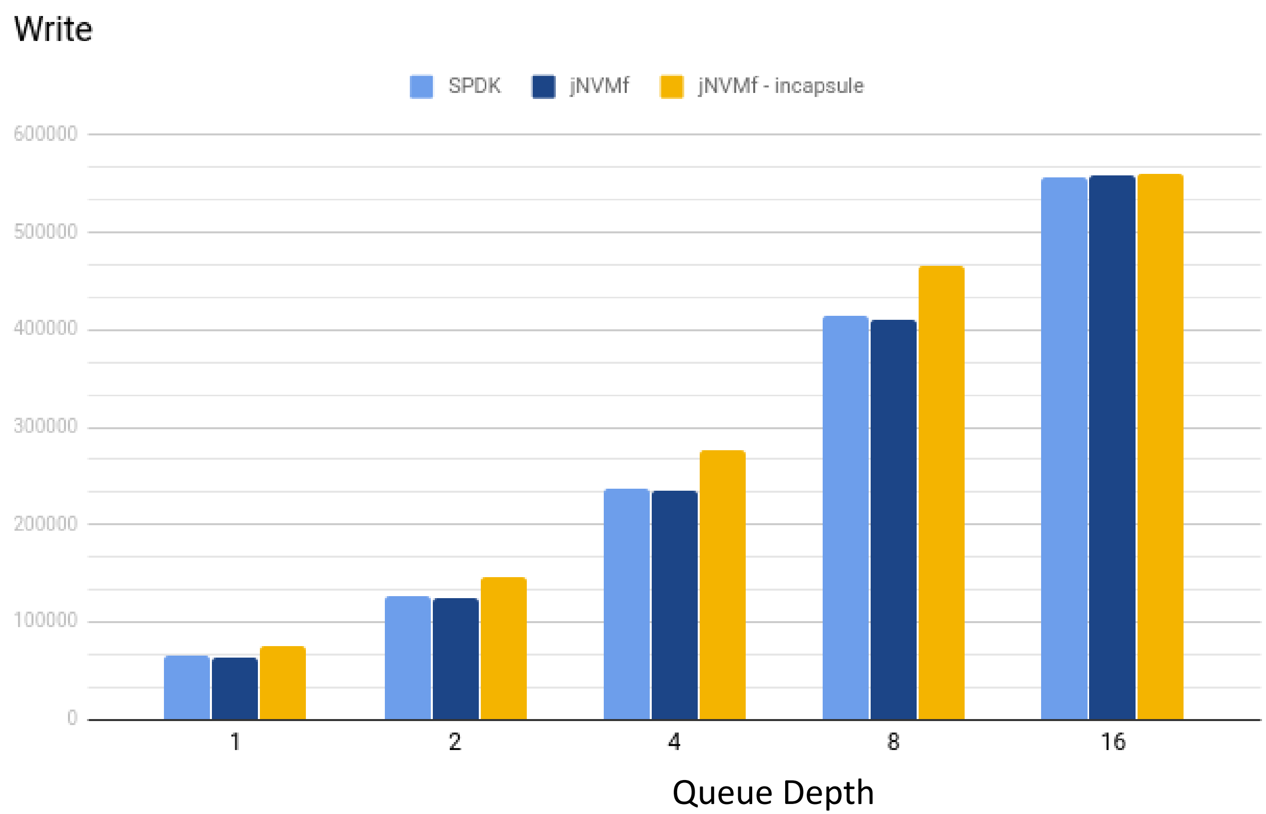
SPDK vs jNVMf IOPS 4k (Write)

- Intel 900p device
- SPDK target

- Both SPDK and jNVMf host saturate device already at QD 16
- jNVMf incapsule data better for smaller QD



Comparing NVMf host libraries: SPDK, jNVMf



TeraSort and TPC-DS Benchmark Setup

- Setup
  - 100Gbs RoCE: Mellanox ConnectX-5, SN2700
  - 8 machines, 16 cores Intel E5-2690 @2.9Ghz
  - 256GB DRAM, 128GB of it given to Spark
- NVMe devices used
  - Samsung 960Pro 1TB
- NVMf Targets deployed
  - SPDK 18.01
  - Linux Kernel 4.13
- Experiments
  - TPC-DS
  - TeraSort

Vanilla Spark setup:

- All data **always** kept in **DRAM**
- Input/Output to /tmpfs HDFS mount
- All intermediate operations in DRAM
- No disk spill

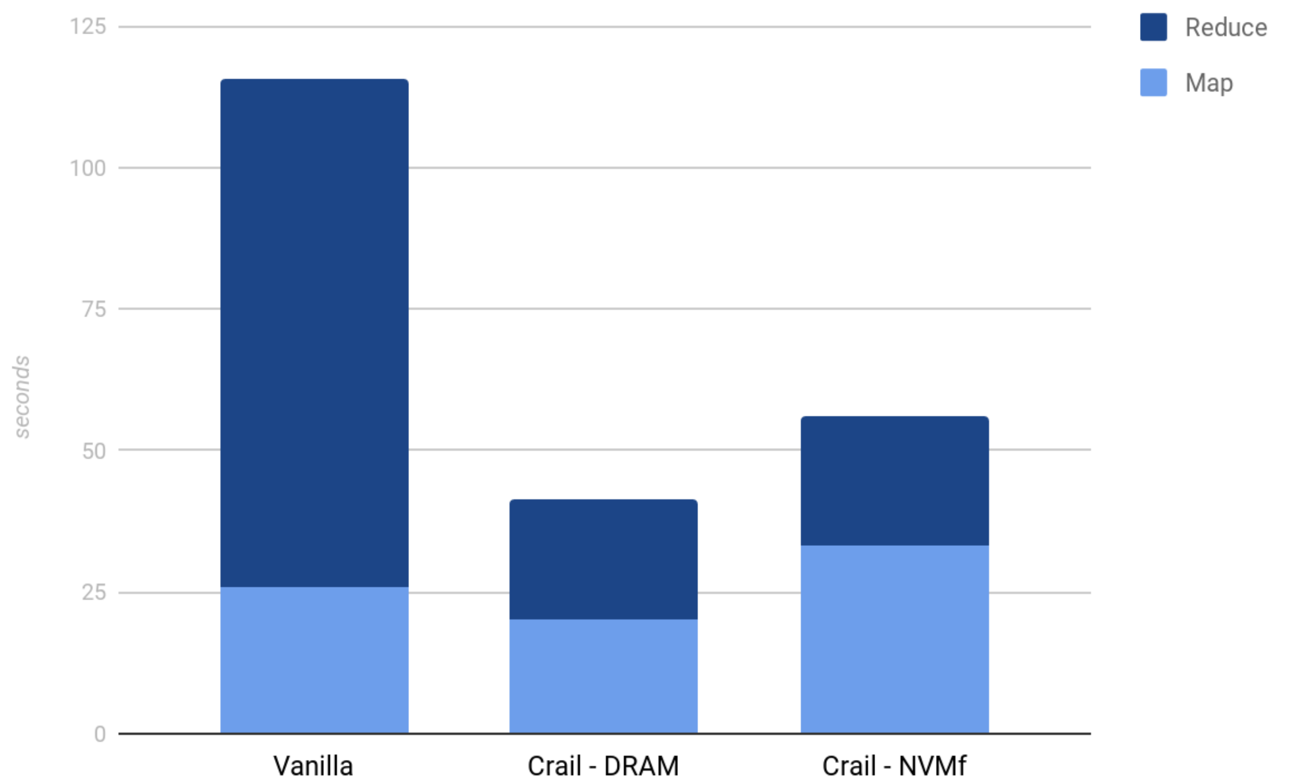
Crail NVMf setup:

- **All data** read and written within **NVMf**
- No shuffle etc. in DRAM

Spark TeraSort Benchmark (Spark@DRAM vs Crail@NVMf)

- Samsung 960Pro
- jNVMf client
- SPDK target
- Sorting 200GB
- **Spark with input/output and map/reduce in DRAM**
- **Crail input/output + map/reduce in NVMf**

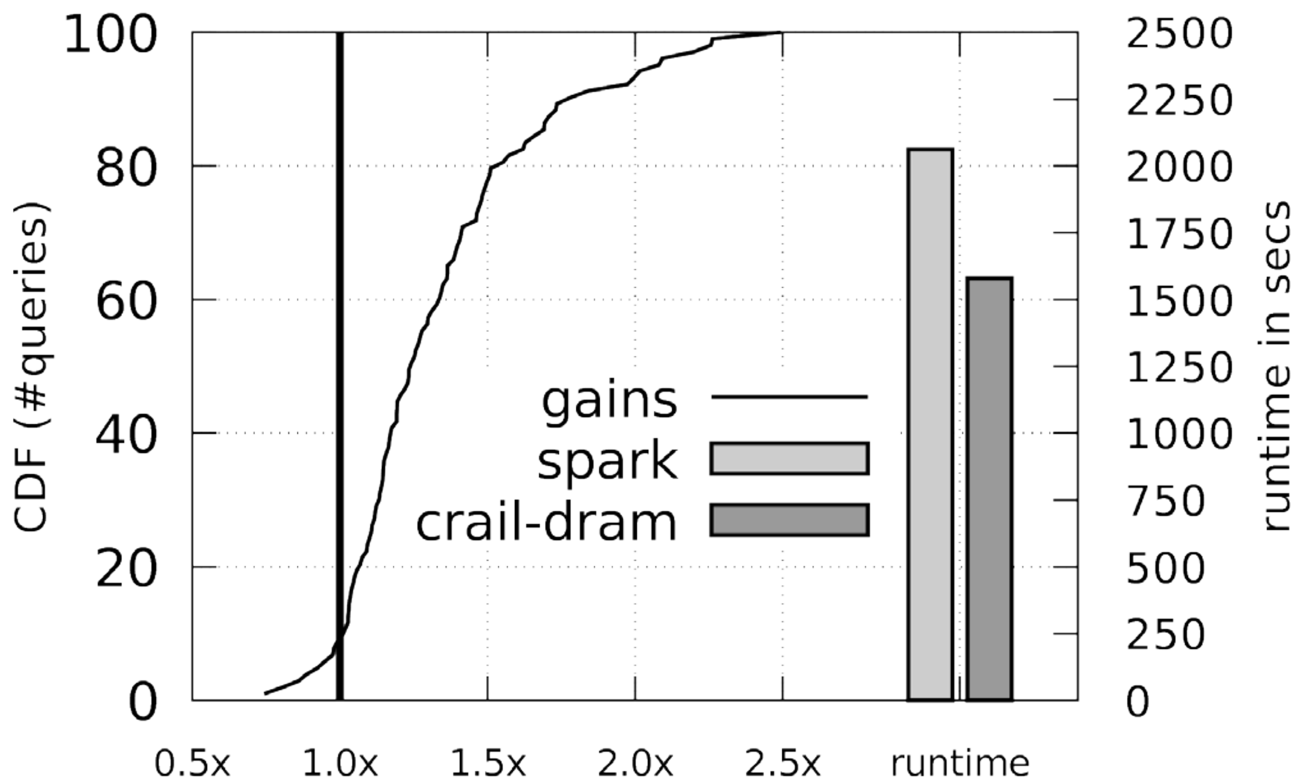
- Crail DRAM 3 times faster than vanilla Spark
- Crail 100% NVMf tier clearly outperforms vanilla Spark
- Reduce time same for Crail DRAM or NVMf
- Some penalty during Map phase (write into NVMf)



TPC-DS Query Performance (Spark@DRAM vs Crail@DRAM)

- **Spark with input/output + shuffle in DRAM**
- **Crail input/output + shuffle in DRAM**

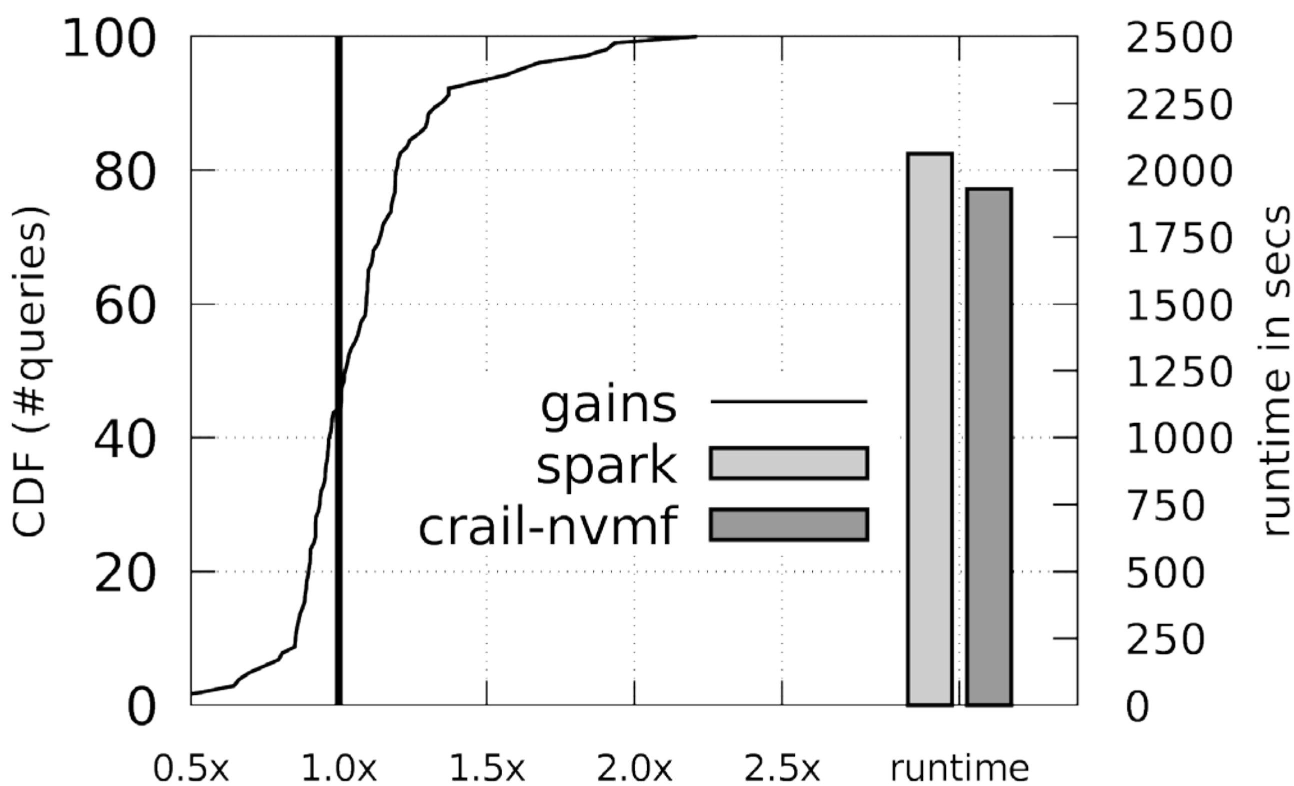
- Almost all queries are faster, up to 2.5x
- Crail makes more efficient use of available hardware resources



TPC-DS Query Performance (Spark@DRAM vs Crail@NVMf)

- Samsung 960Pro devices
- jNVMf client
- Kernel NVMf target
- **Spark with input/output + shuffle in DRAM**
- **Crail input/output + shuffle on NVMf**

- Half of the queries are faster
- Overall Crail with NVMf faster than Vanilla Spark in DRAM: save cost and speed up!



Summary & Outlook

- NVMf is the adequate extension of NVMe in a distributed system
  - Allows efficient management of ephemeral data which do not fit into DRAM
  - Using Crail + NVMf: lower cost and better performance
- NVMf supports clever device I/O management
  - In-Capsule data accelerates short NVM writes
  - Dataset management allows to pass hints to device - unfortunately not yet supported in all NVMf implementations
  - Adding Bit Bucket semantic to NVMf would help, even if target device is not capable of it
- We today mainly looked at DRAM replacement
  - NVM performance is one aspect – persistency is another
  - Working on NVMf tier data recovery feature (logging, replay)
- NVM tier access patterns need further investigation
  - At file level – data are written sequentially, but
  - Writing huge amounts of data in parallel – globally random access at device level
- Apache Crail including NVMf tier open source at https://github.com/apache/incubator-crail

OPENFABRICS ALLIANCE

14th ANNUAL WORKSHOP 2018

THANK YOU

Bernard Metzler

IBM Zurich Research