Running Spark on a High-Performance Cluster using RDMA Networking and NVMe Flash

Patrick Stuedi, IBM Research

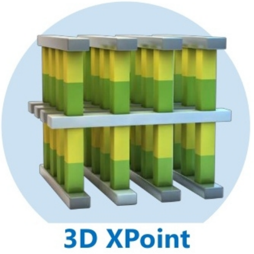
SPARK SUMMIT 2017

Hardware Trends

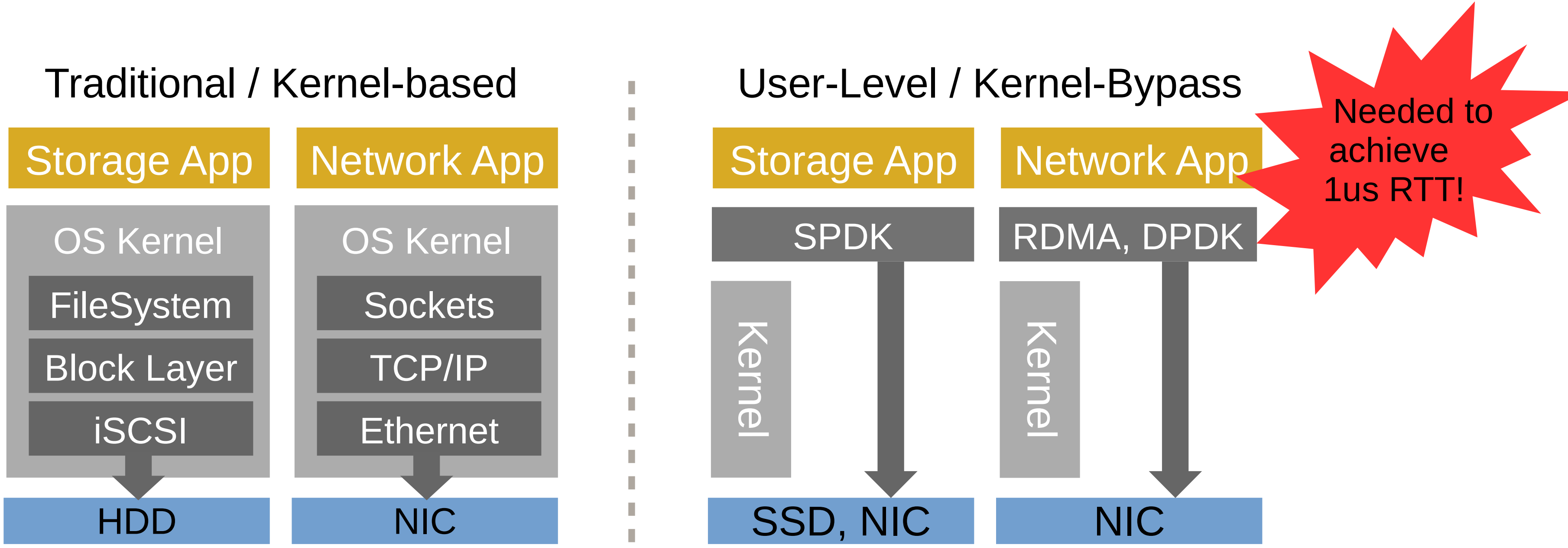
	2010	2017
		community target
Storage	100 MB/s 100ms	1000 MB/s 200us
Network	1Gbps 50us	10Gbps 20us
CPU	~3GHz	~3GHz

Hardware Trends

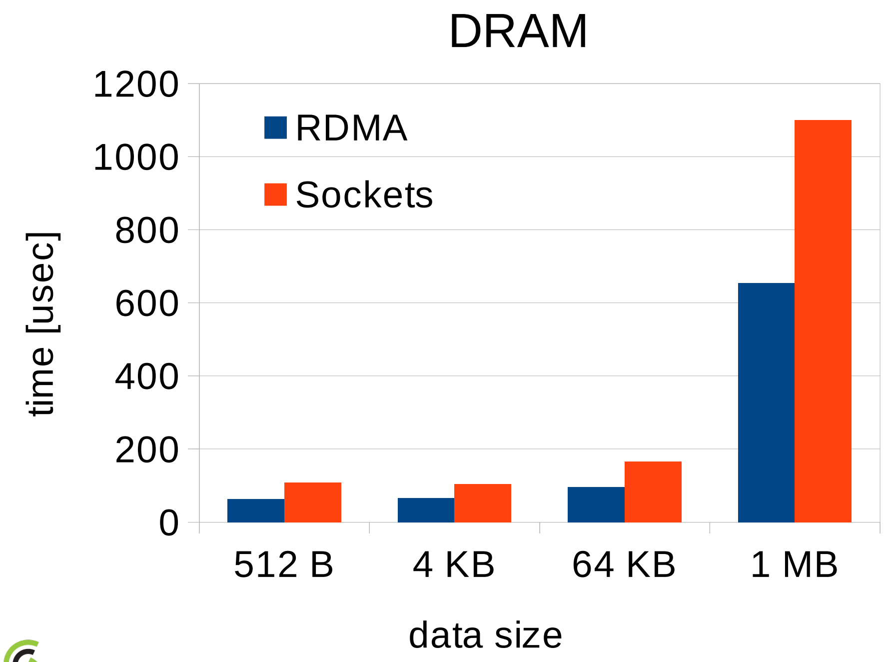
	2010	2017 (community target)	2017 (our target)
Storage	100 MB/s 100ms	1000 MB/s 200us	10 GB/s 50us
Network	1Gbps 50us	10Gbps 20us	100Gbps 1us
CPU	~3GHz	~3GHz	☹

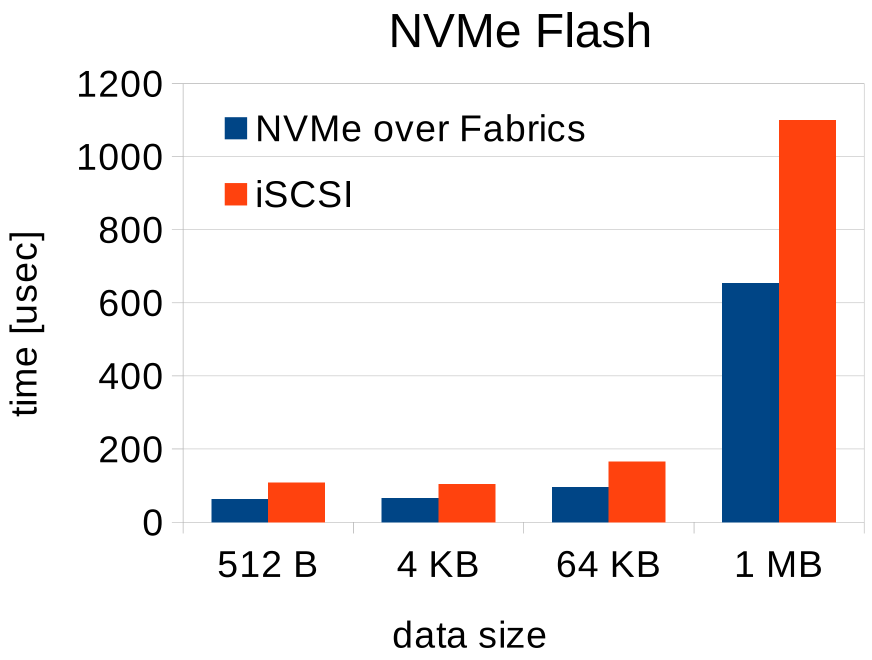


User-Level APIs

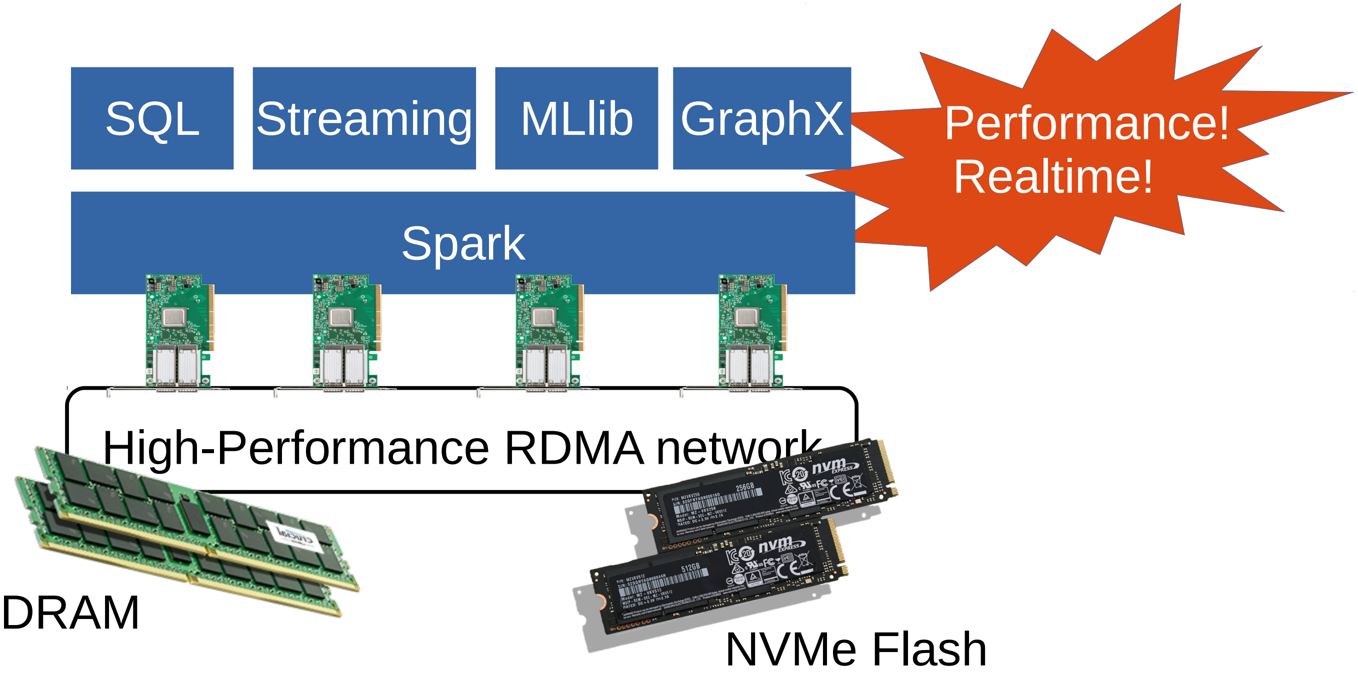


Remote Data Access

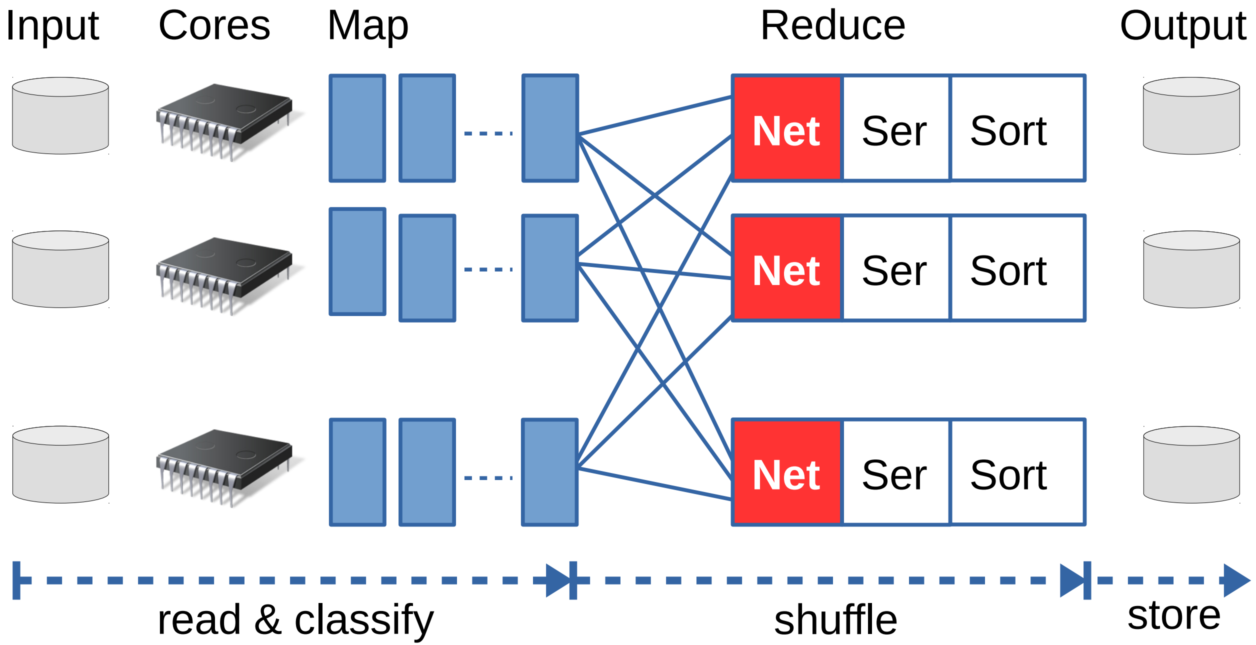




Let's Use it!


SQL
Streaming
MLlib
GraphX
Performance!
Realtime!
Spark
High-Performance RDMA network
DRAM
NVMe Flash

Case Study: Sorting in Spark

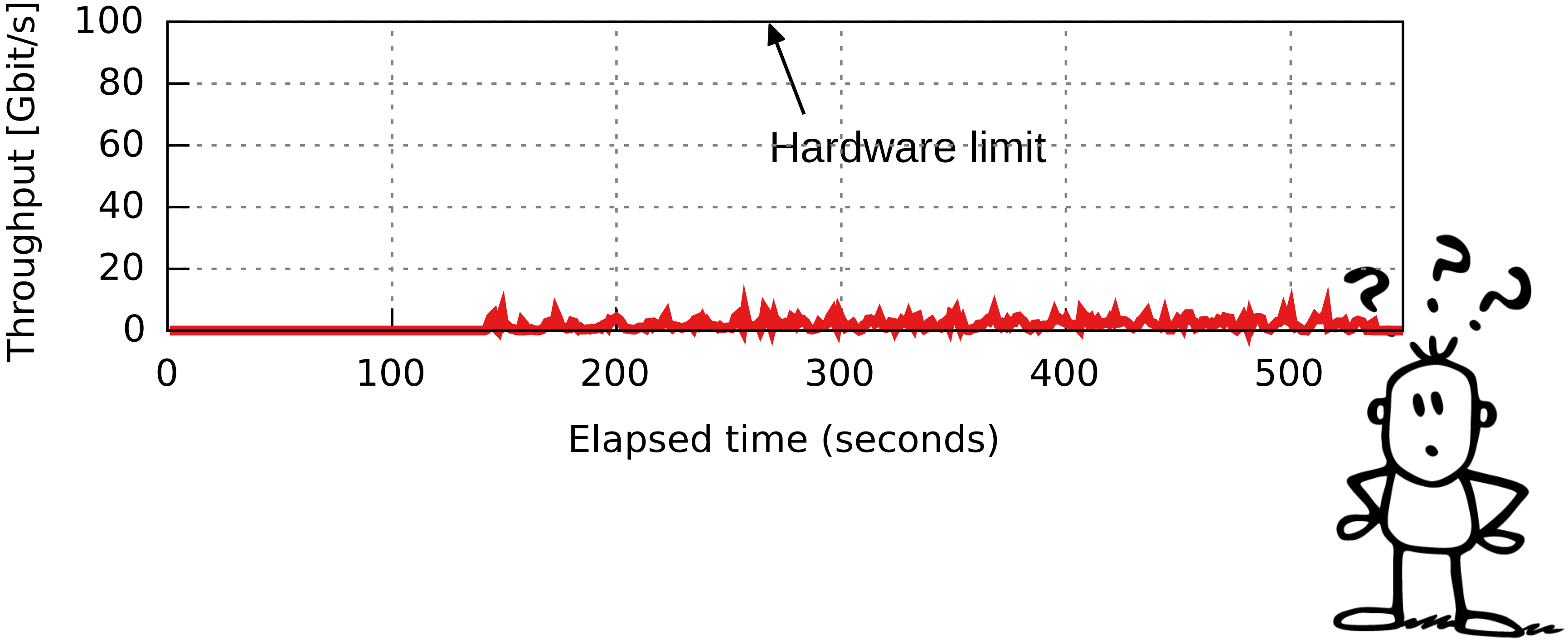


Experiment Setup

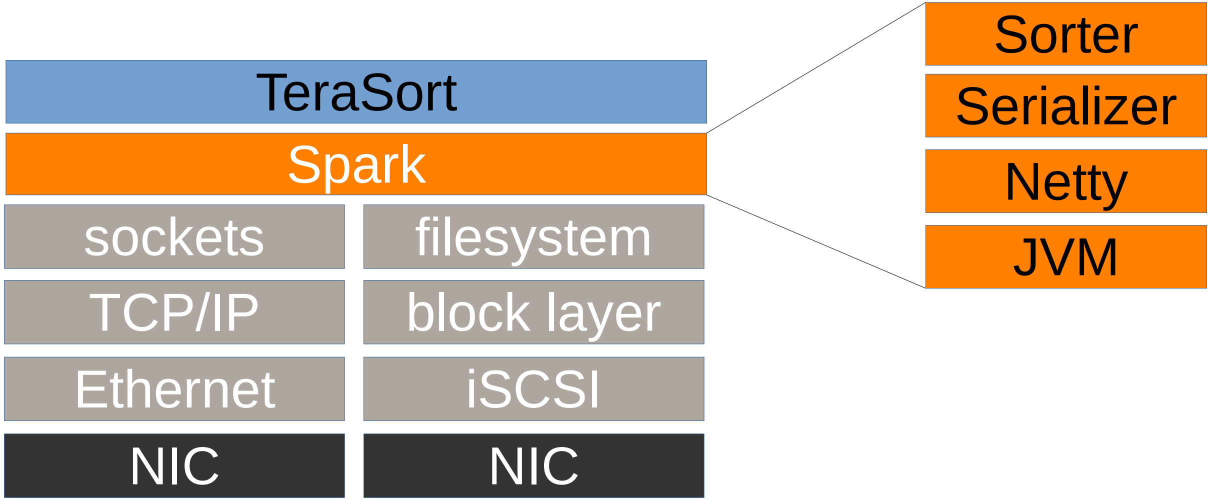
- Total data size: 12.8 TB
- Cluster size: 128 nodes
- Cluster hardware:
  - DRAM: 512 GB DDR 4
  - Storage: 4x 1.2 TB NVMe SSD
  - Network: 100GbE Mellanox RDMA

Flash bandwidth per node matches network bandwidth

How is the Network Used?

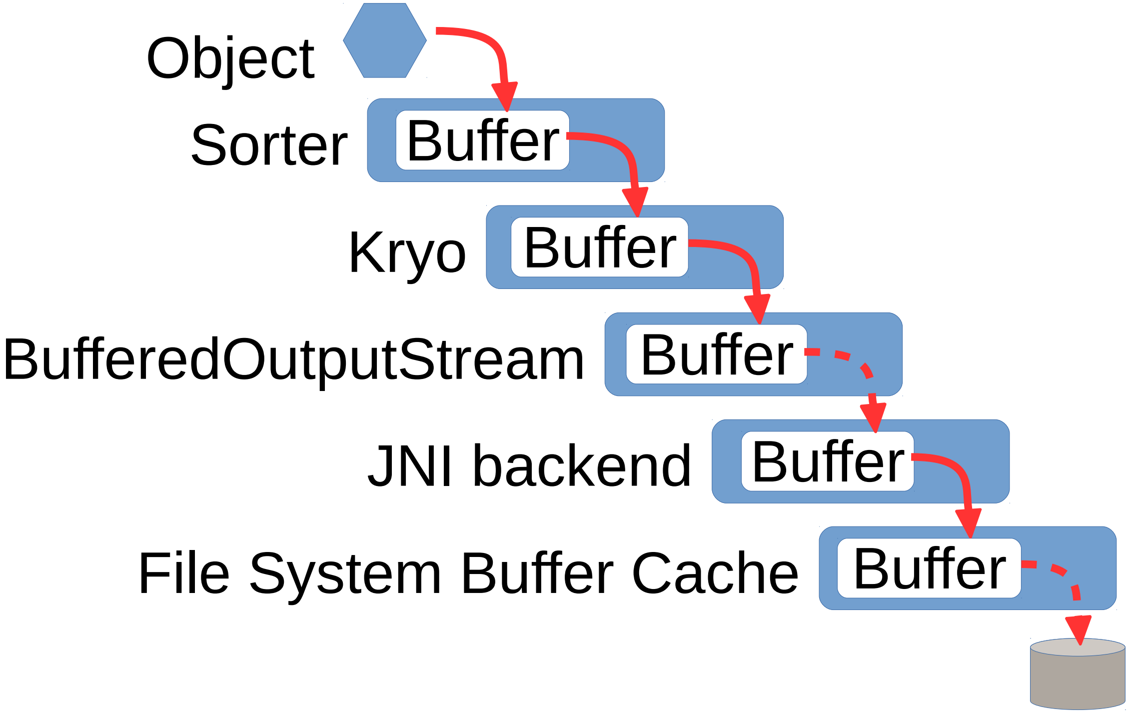


What is the Problem?

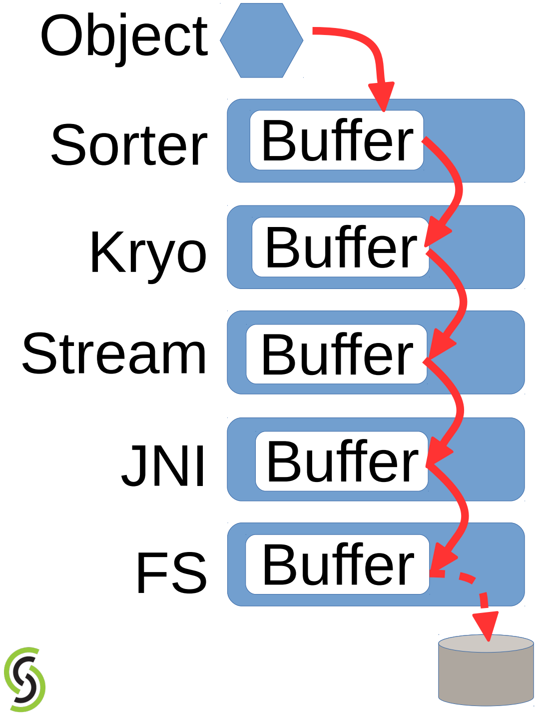


- Spark uses legacy networking and storage APIs: no kernel-bypass
- Spark itself introduces additional I/O layers: Netty, serializer, sorter, etc.

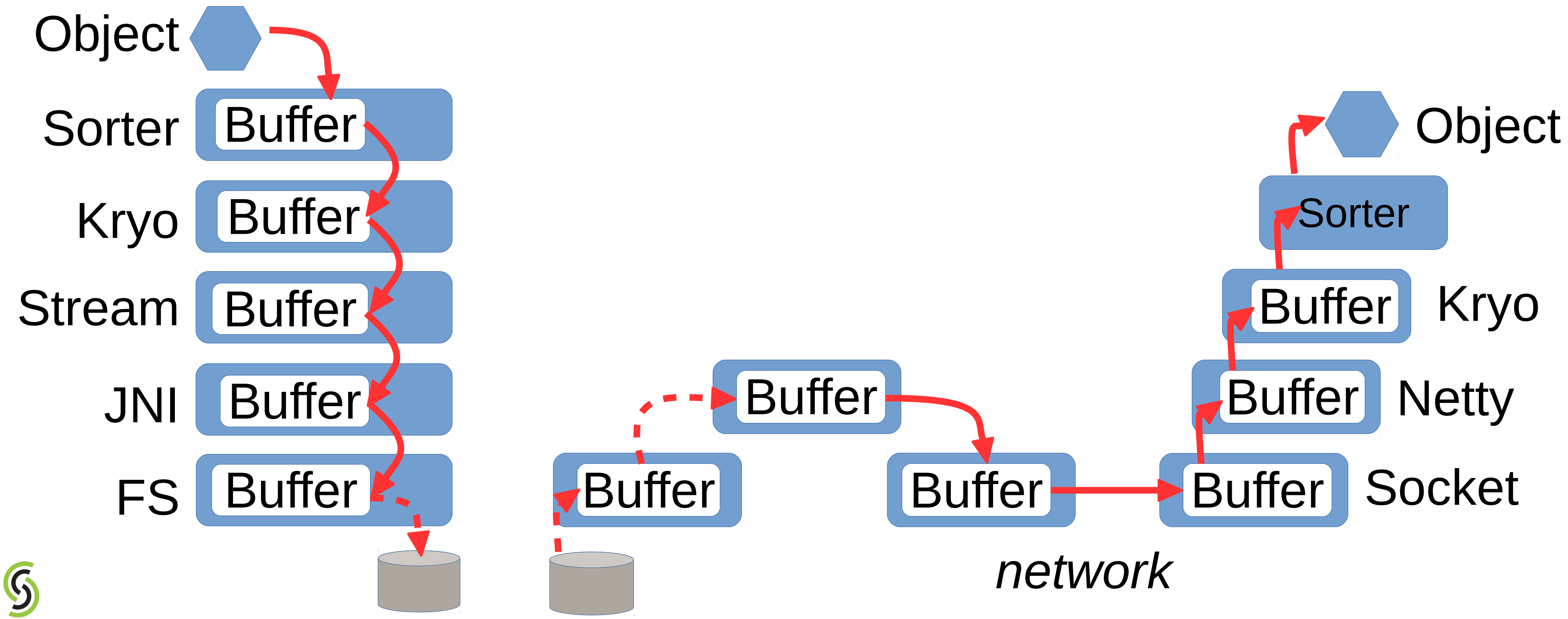
Example: Shuffle (Map)



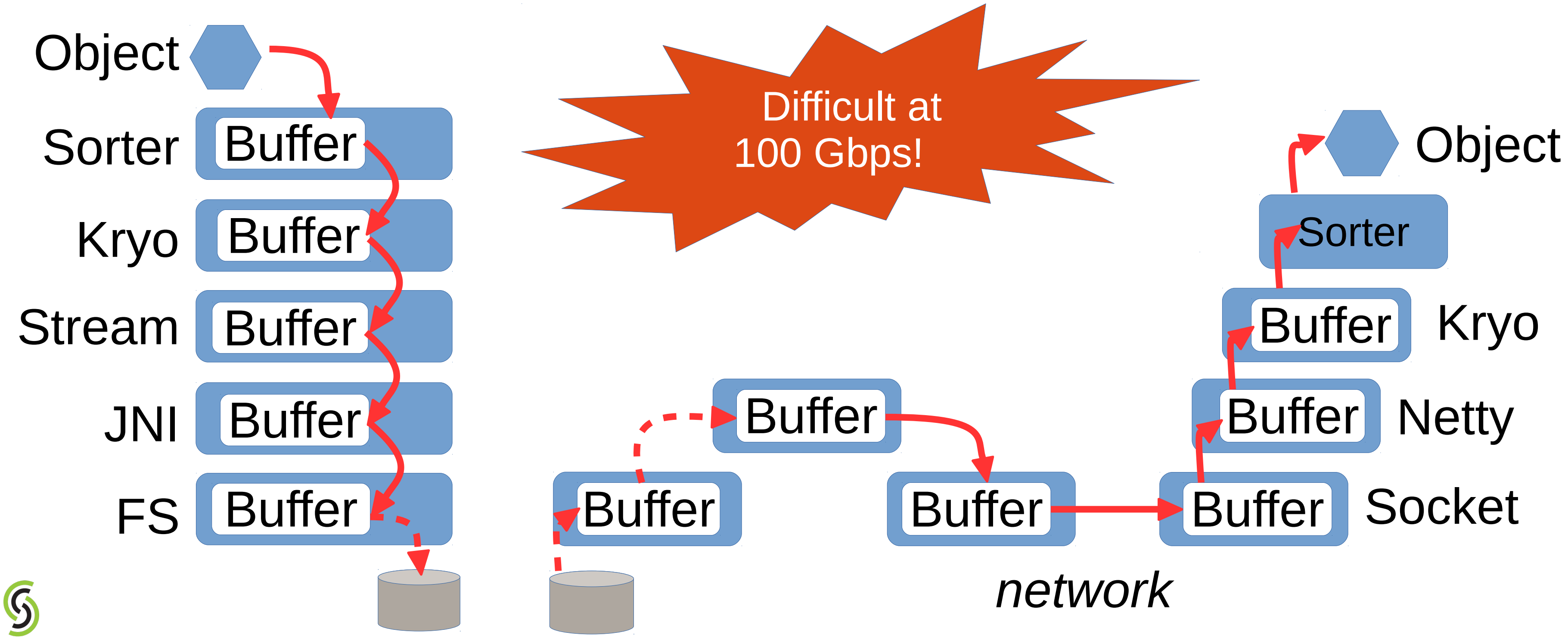
Example: Shuffle (Map)



Example: Shuffle (Map+Reduce)



Example: Shuffle (Map+Reduce)

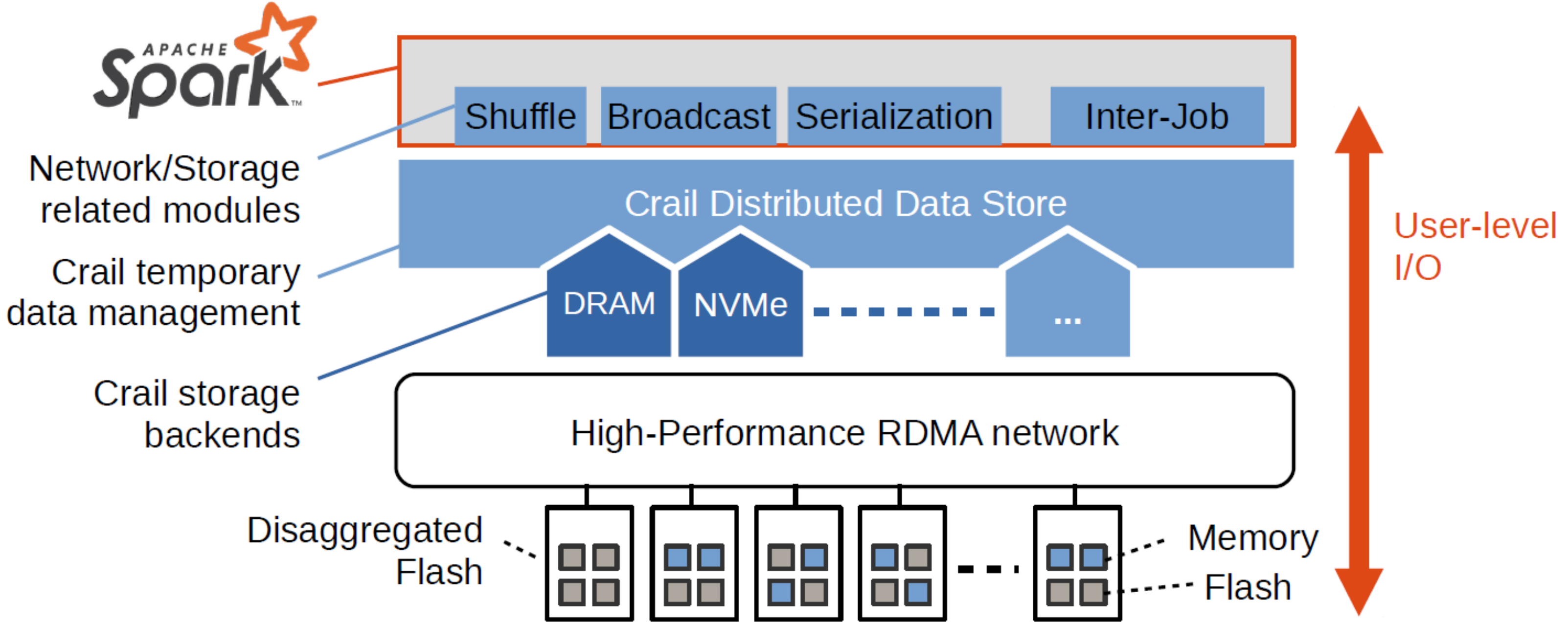


SPARK SUMMIT 2017

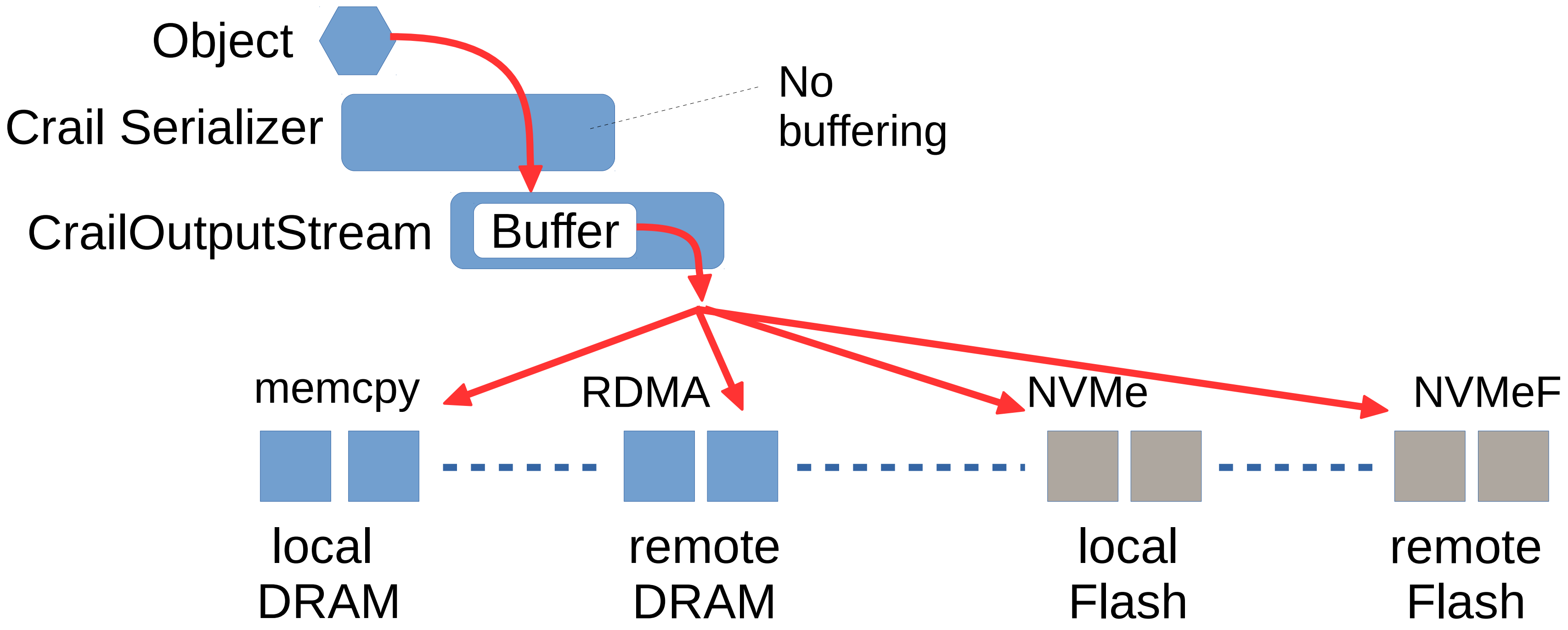
How can we fix this...

- Not just for shuffle
  - Also for broadcast, RDD transport, inter-job sharing, etc.
- Not just for RDMA and NVMe hardware
  - But for any possible future high-performance I/O hardware
- Not just for co-located compute/storage
  - Also for resource disaggregation, heterogeneous resource distribution, etc.
- Not just improve things
  - Make it perform at the hardware limit

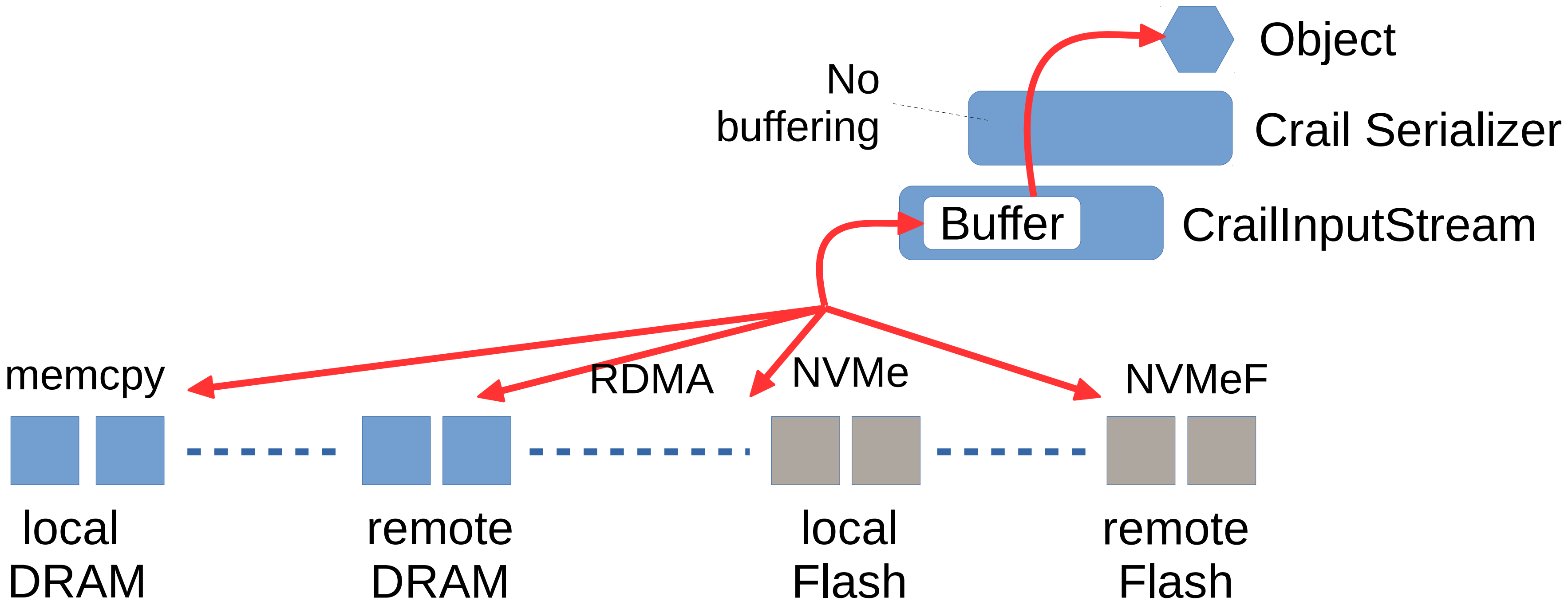
The CRAIL Approach



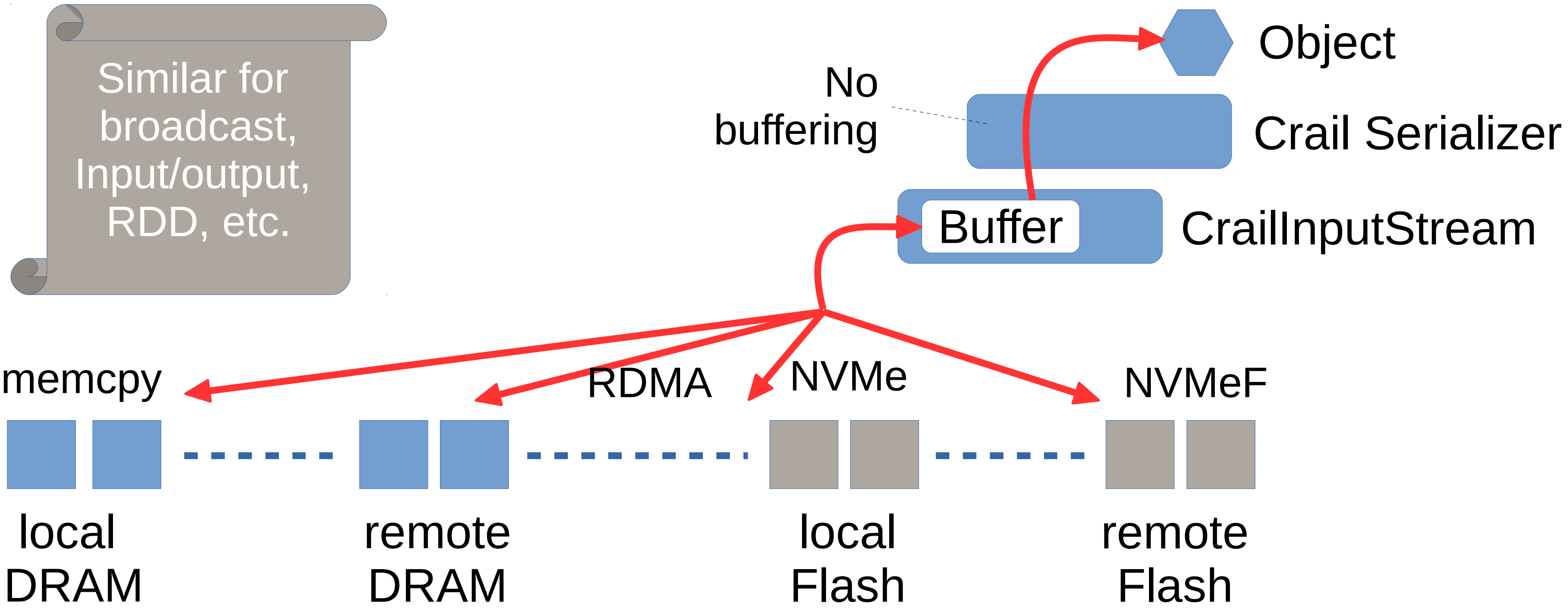
Example: Crail Shuffle (Map)



Example: Crail Shuffle (Reduce)



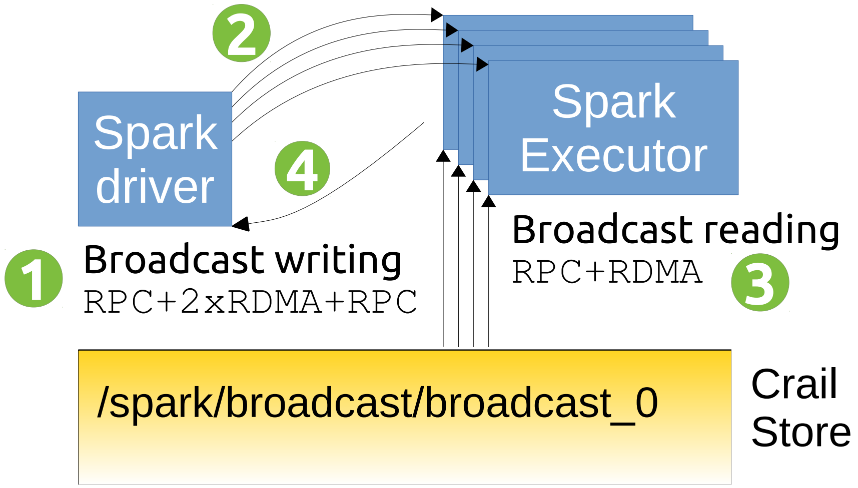
Example: Crail Shuffle (Reduce)

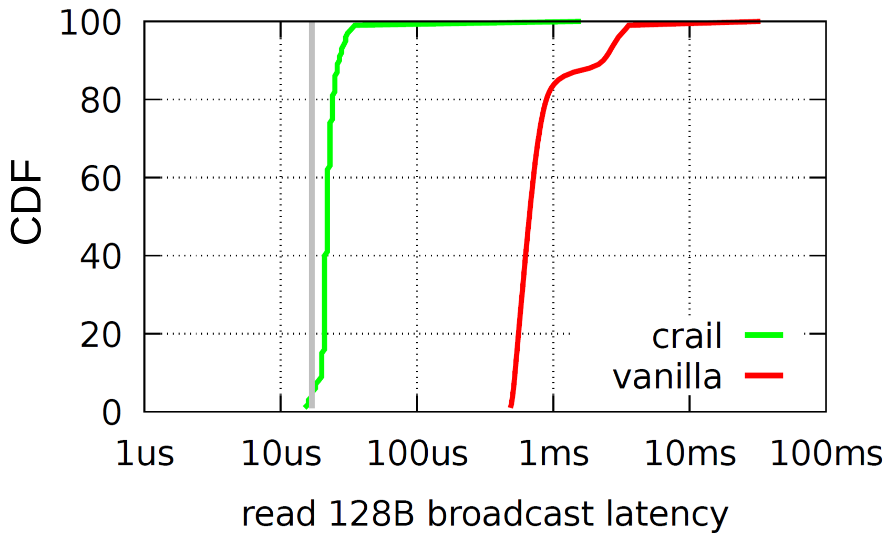


Performance: Configuration

- Experiments
  - Performance: Broadcast, GroupBy, Sorting, SQL (memory)
  - Disaggregation, tiering: Sorting (memory/flash)
- Cluster size: 8 nodes, except TeraSort: 128 nodes
- Cluster hardware:
  - DRAM: 512 GB DDR 4
  - Storage: 4x 1.2 TB NVMe SSD
  - Network: 100GbE Mellanox RDMA

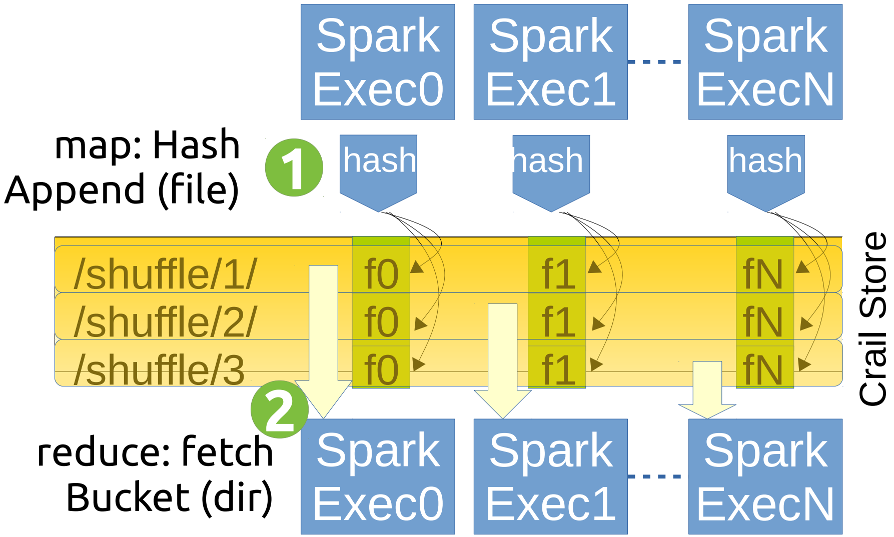
Spark Broadcast

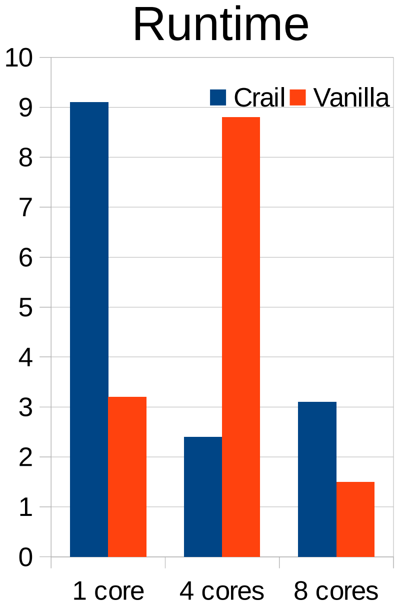


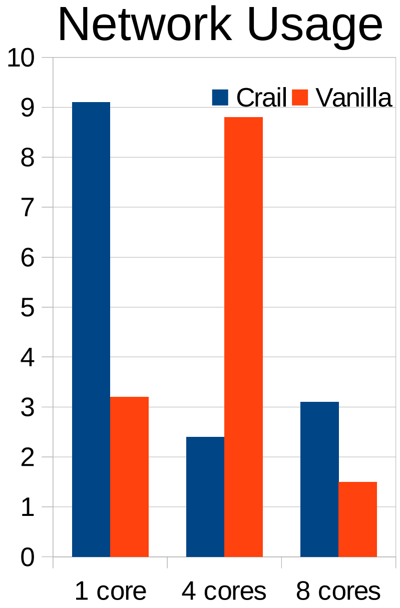



```
val bcVar = sc.Broadcast(new Array[Byte](128))
sc.parallelize(1 to tasks, tasks).map(_ => {
  bcVar.value.length
}).count
```

Spark GroupBy

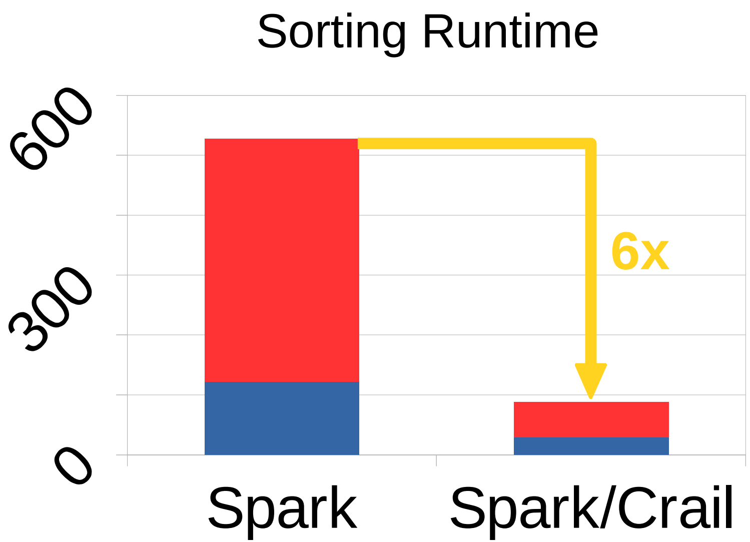


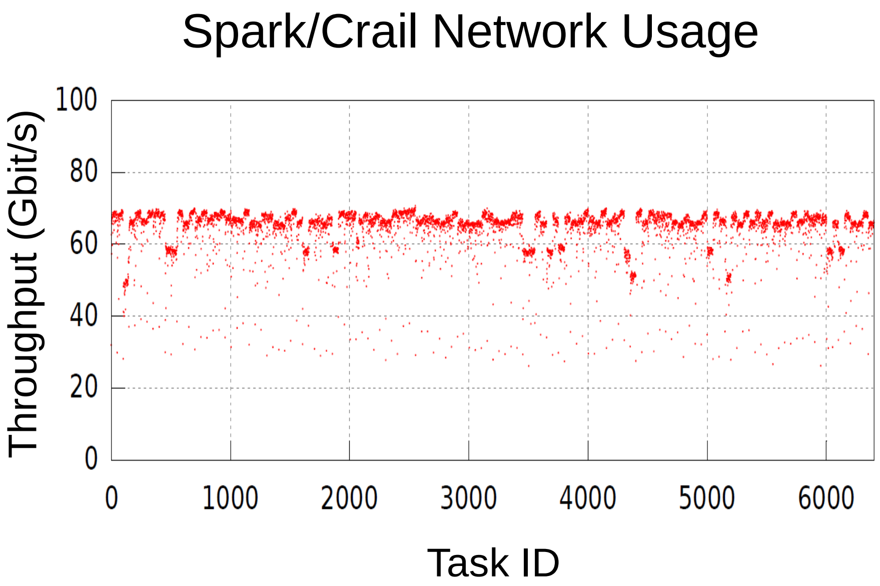





```
val pairs = sc.parallelize(1 to tasks, tasks).flatmap(_ => {
  var values = new array[(Long,Array[Byte])](numKeys)
  values = initValues(values)
}).cache().groupByKey().map(v => v._1).count()
```

Sorting 12.8 TB on 128 nodes





How fast is this?

Spark

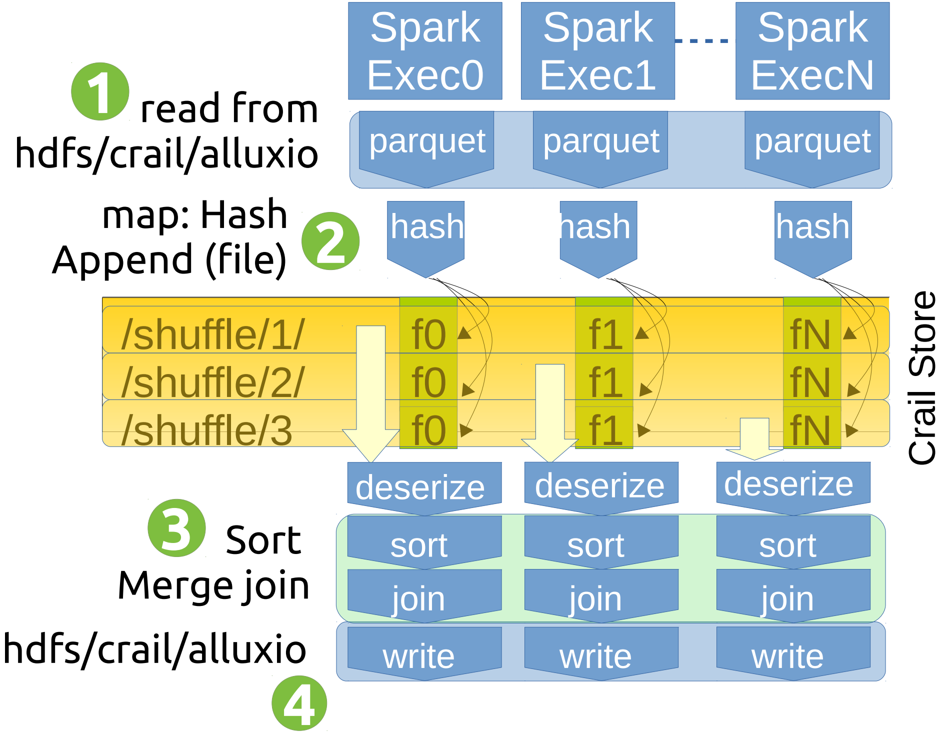
Native C distributed sorting benchmark

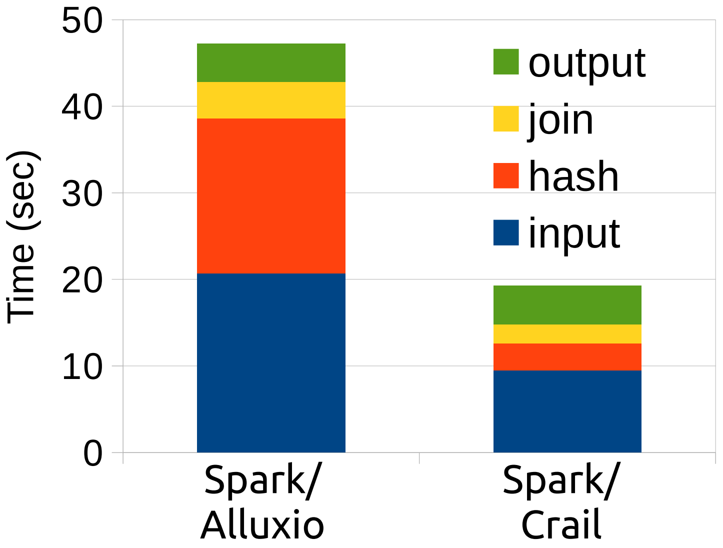
www.sortingbenchmark.org

	Spark/Crail	Winner 2014	Winner 2016
Size (TB)	12.8	100	100
Time (sec)	98	1406	98.6
Total cores	2560	6592	10240
Network HW (Gbit/s)	100	10	100
Rate/core (GB/min)	3.13	0.66	4.4

Sorting rate of Crail/Spark only 27% slower than rate of Winner 2016

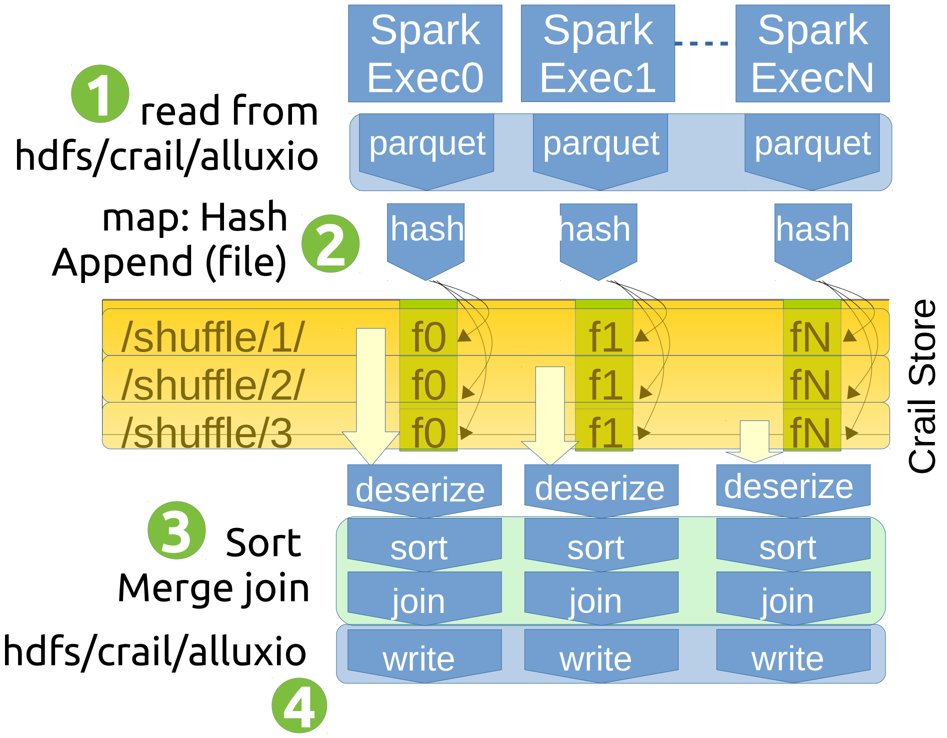
Spark SQL Join

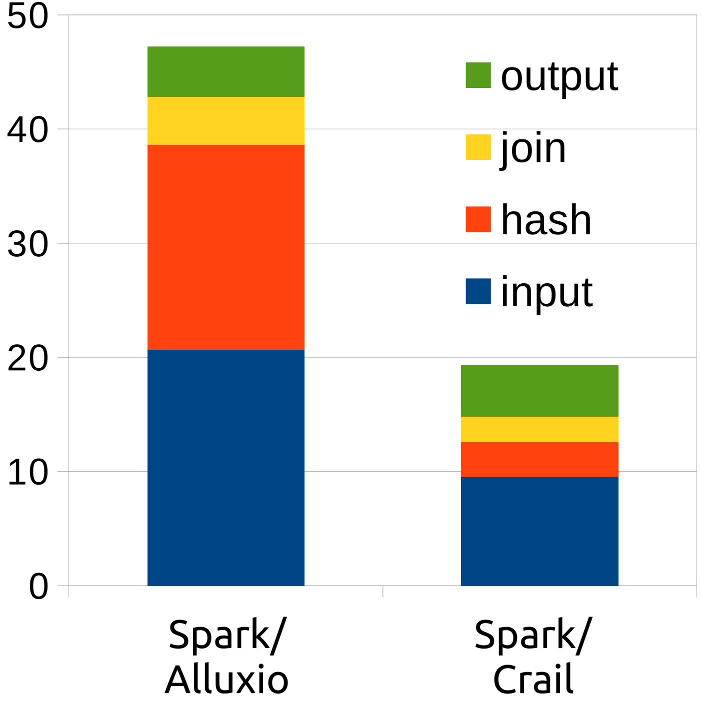





```
val ds1 = spSess.read.parquet("...")
val ds2 = spSess.read.parquet("...")
val resultDS = ds1.joinWith(ds1, ds1(key) === ds2(key))
resultDS.write.format("parquet").mode(SaveMode.Overwrite).save("...")
```

Spark SQL Join (Broadcast)





Conclusions

- Effectively using high-performance I/O hardware in Spark is challenging
- Crail is an attempt to re-think how Spark should interact with network and storage hardware
  - User-level I/O, storage disaggregation, memory/flash convergence

Crail for Spark is Open Source

- www.crail.io
- github.com/zrlio/spark-io
- github.com/zrlio/crail
- github.com/zrlio/parquetgenerator
- github.com/zrlio/crail-terasort

Thank You.

The CRAIL Approach