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

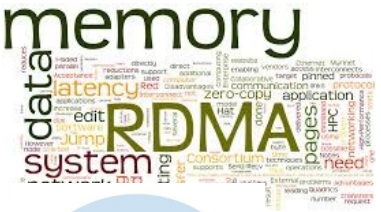
Patrick Stuedi, IBM Research

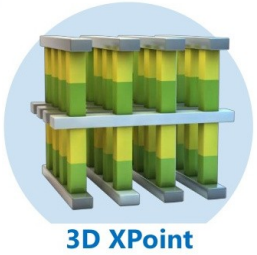
Hardware Trends

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

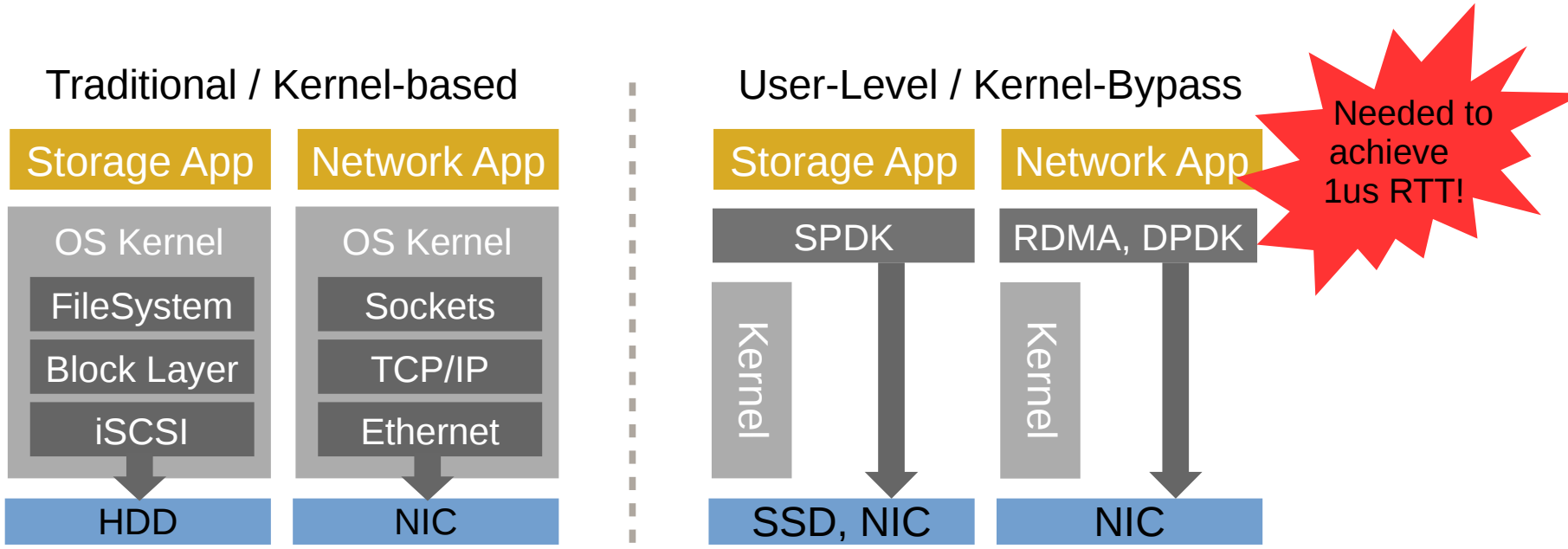
Hardware Trends

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

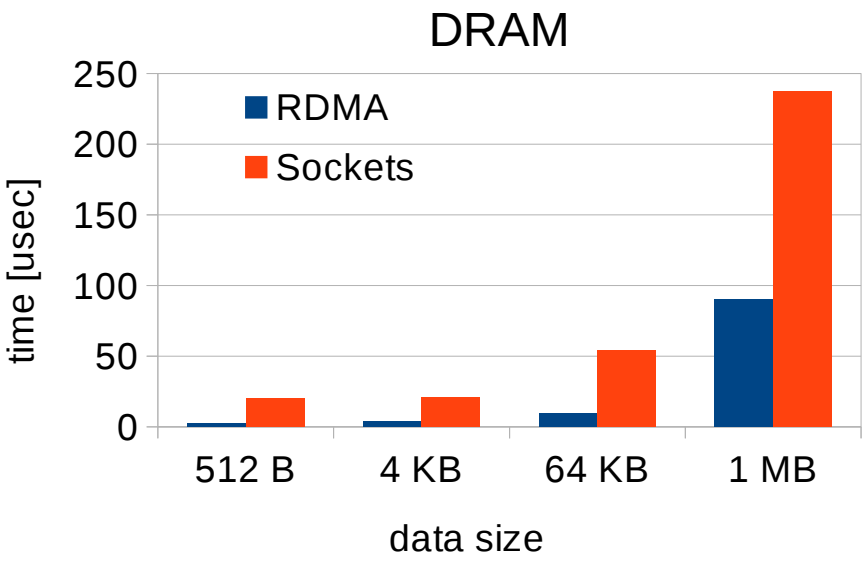


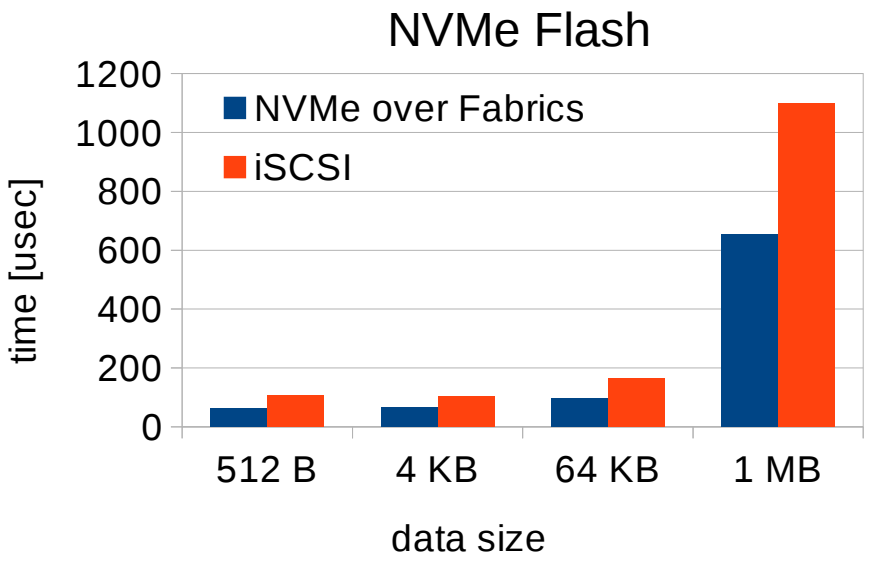


User-Level APIs



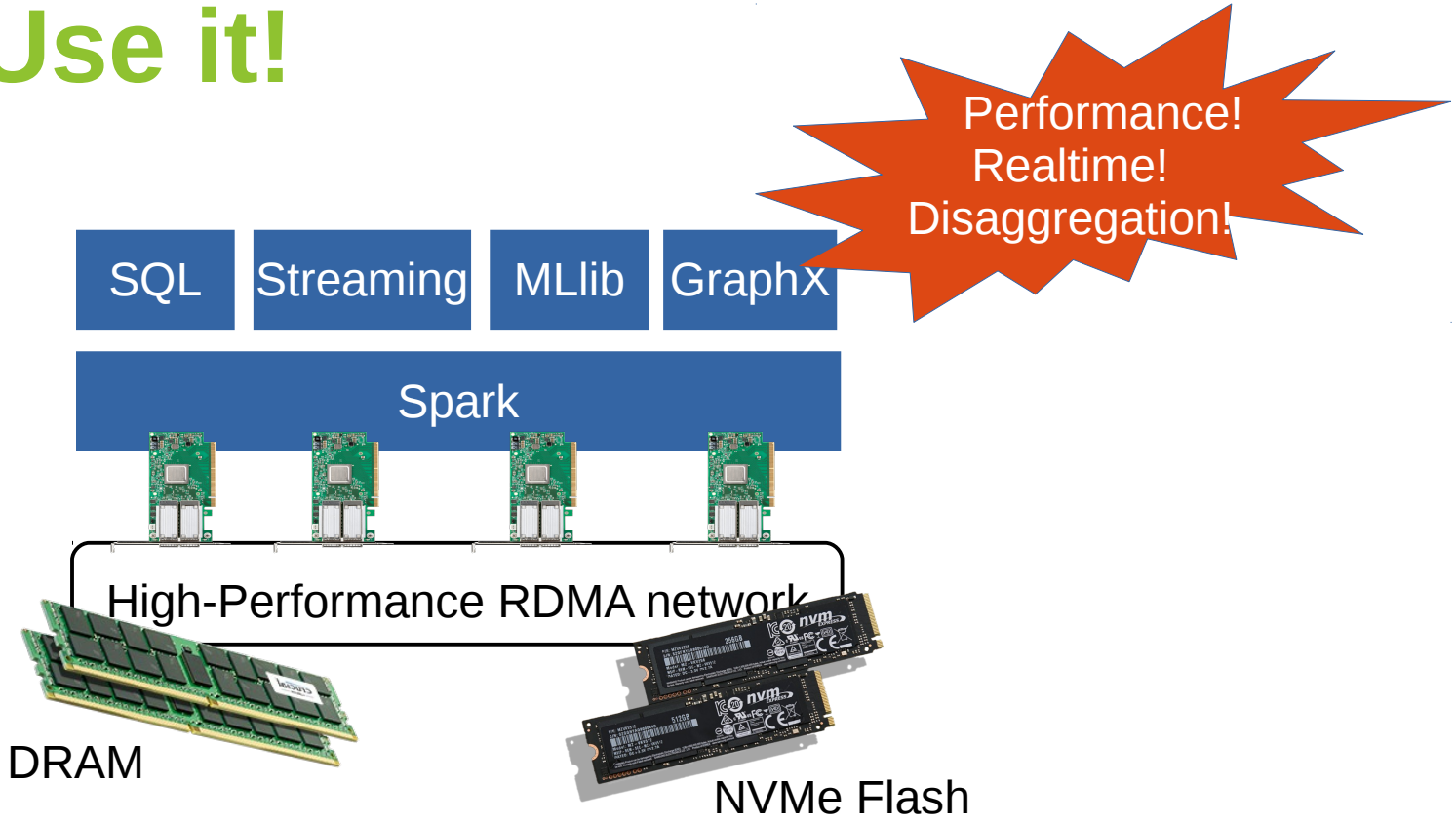
Remote Data Access



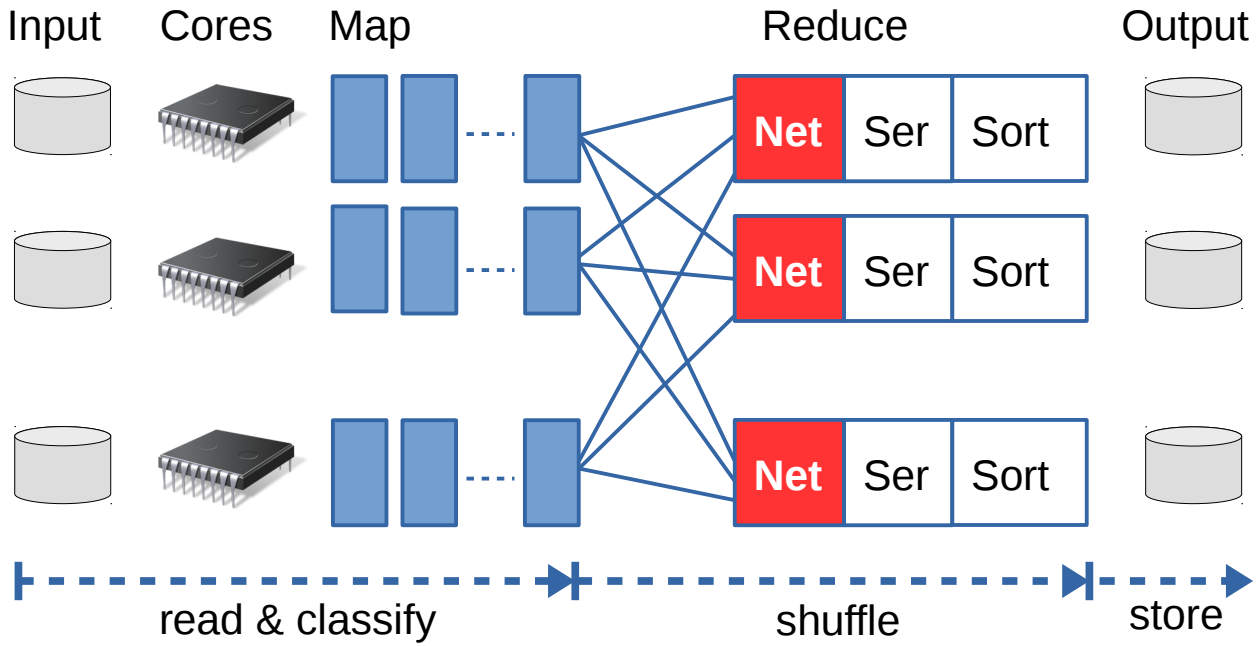


User-level APIs offer 2-10x better performance than traditional kernel-based APIs

Let's Use it!

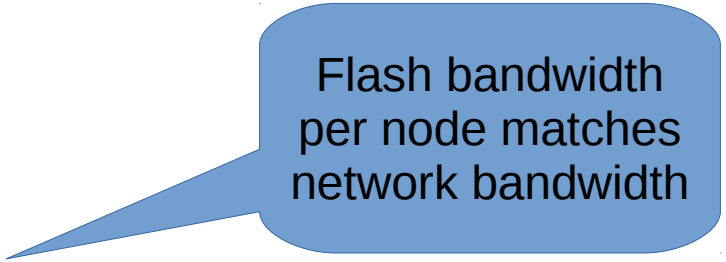

Performance!
Realtime!
Disaggregation!
SQL
Streaming
MLlib
GraphX
Spark
High-Performance RDMA network
DRAM
NVMe Flash

Case Study: Sorting in Spark

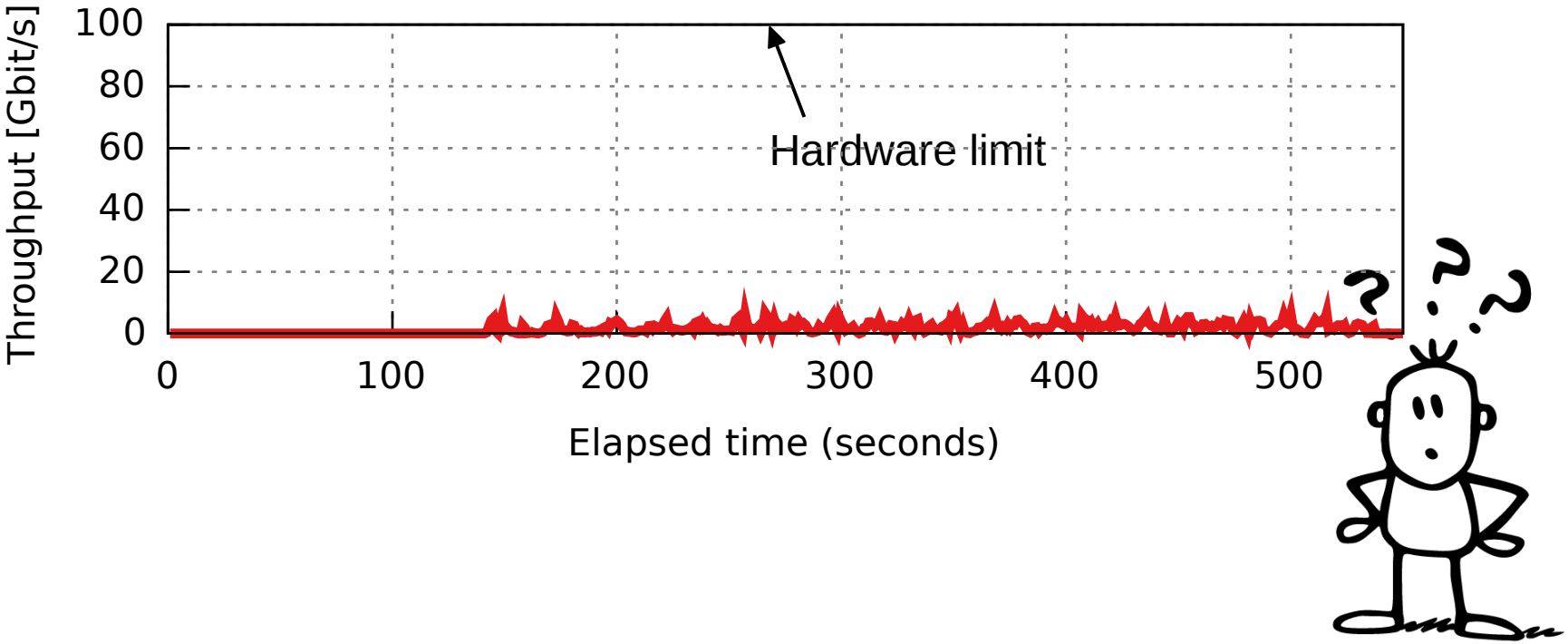


Experiment Setup

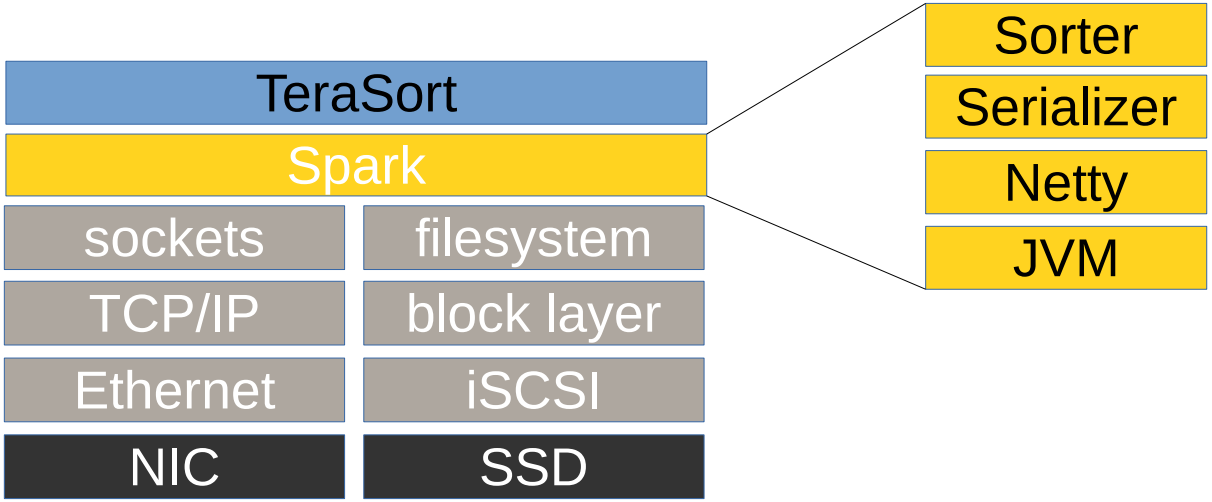
- Total data size: 12.8 TB
- Cluster size: 128 nodes
- Cluster hardware:
  - DRAM: 512 GB DDR 4
  - Storage: 4x 1.2 TB NVMe SSD
  - Network: 100GbE Mellanox RDMA



How is the Network Used?

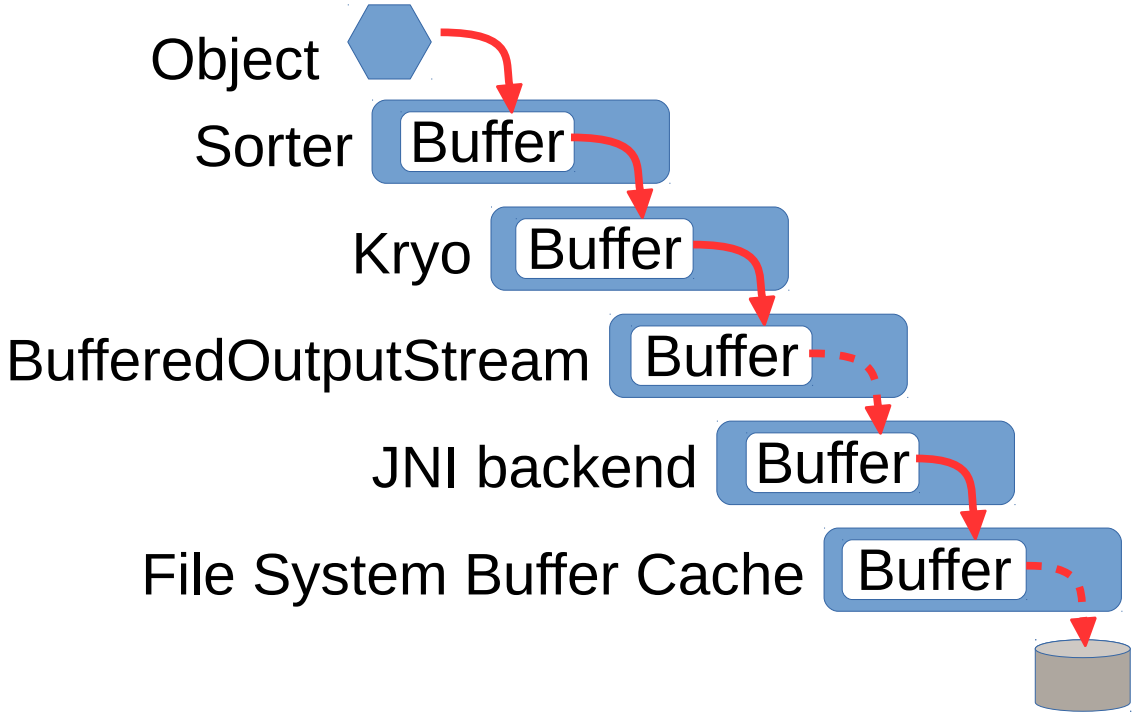


What is the Problem?

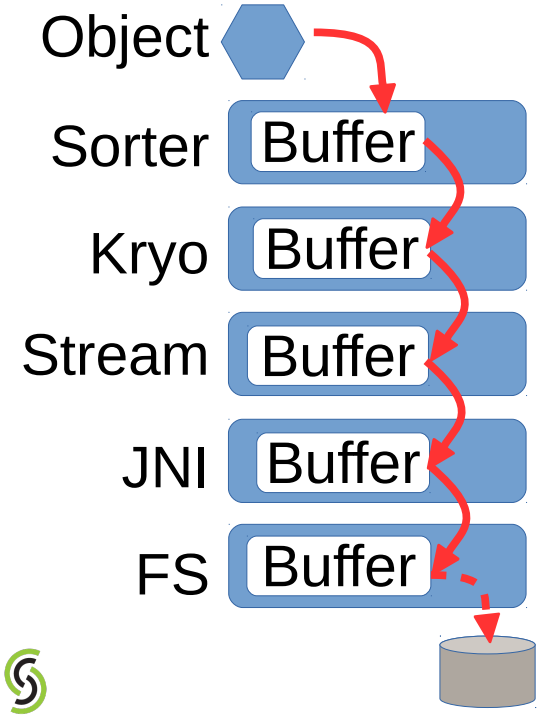


- Spark uses legacy networking and storage APIs: no kernel-bypass
- Spark itself introduces additional I/O layers: Netty, serializer, sorter, etc.

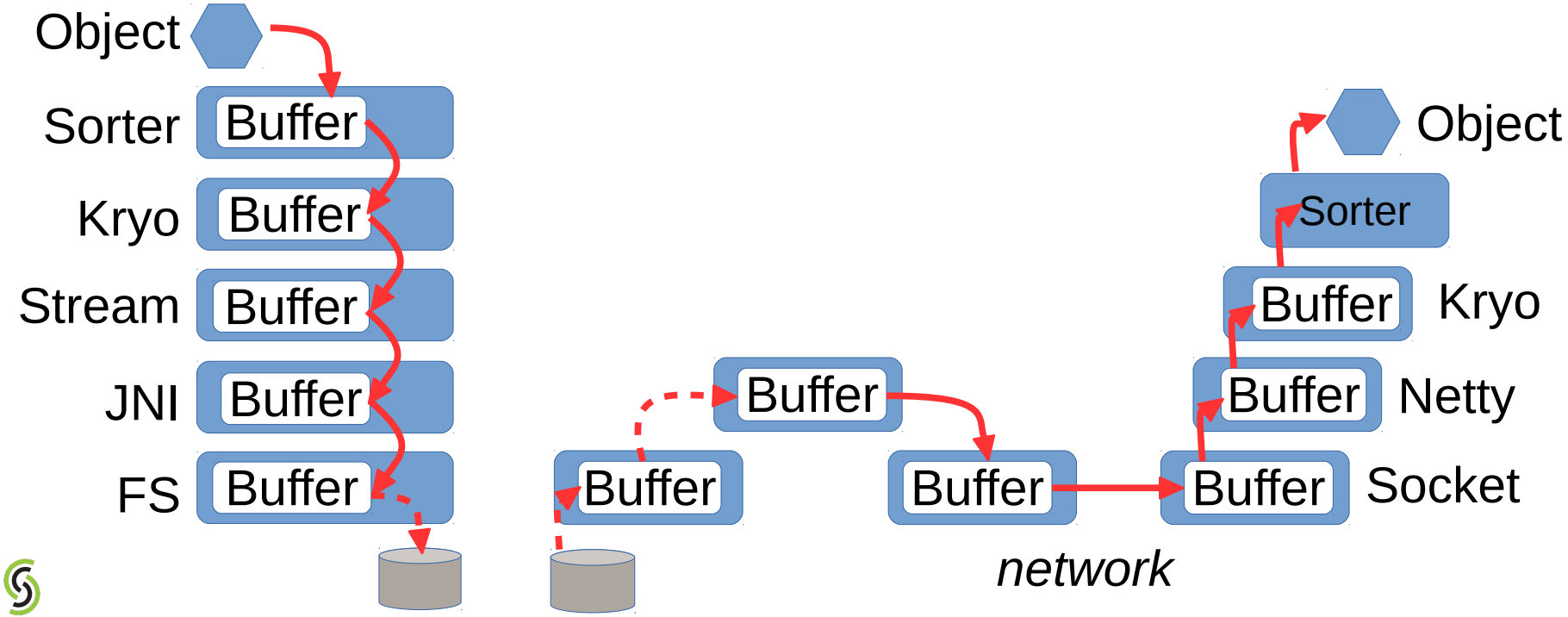
Example: Shuffle (Map)



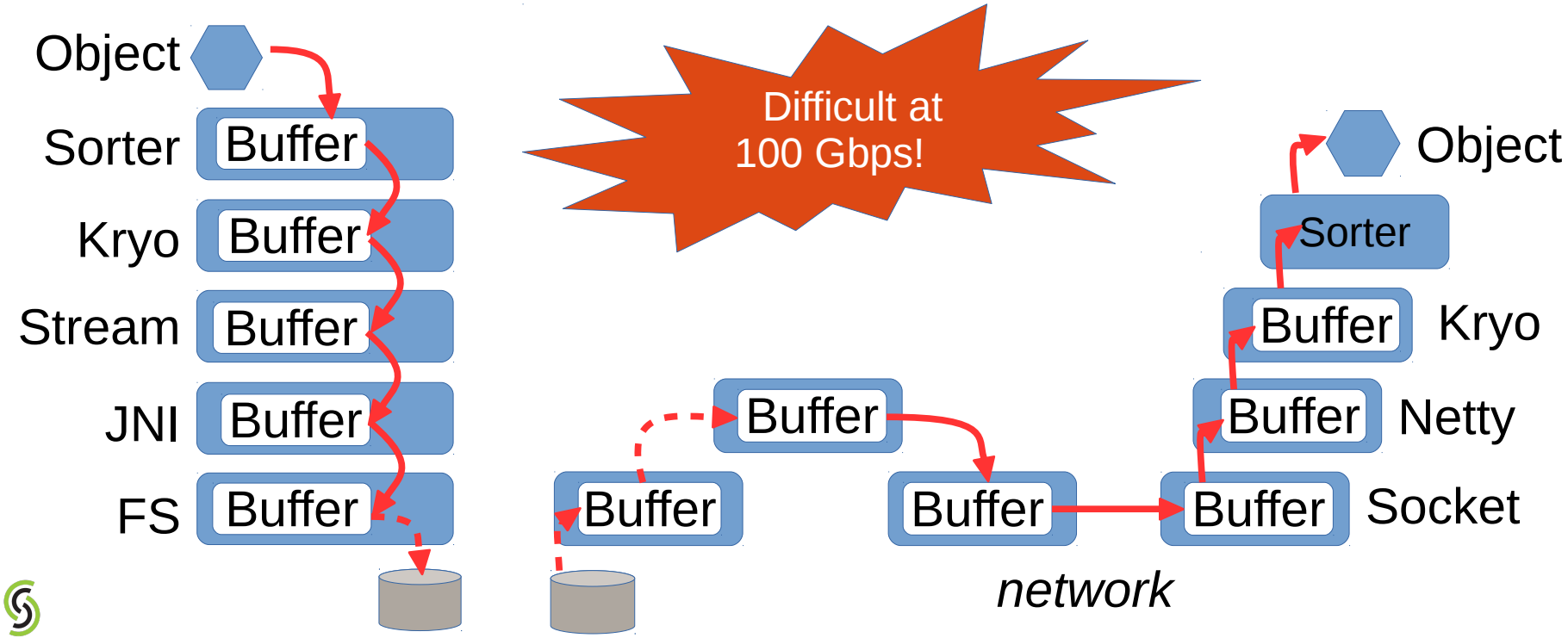
Example: Shuffle (Map)



Example: Shuffle (Map+Reduce)



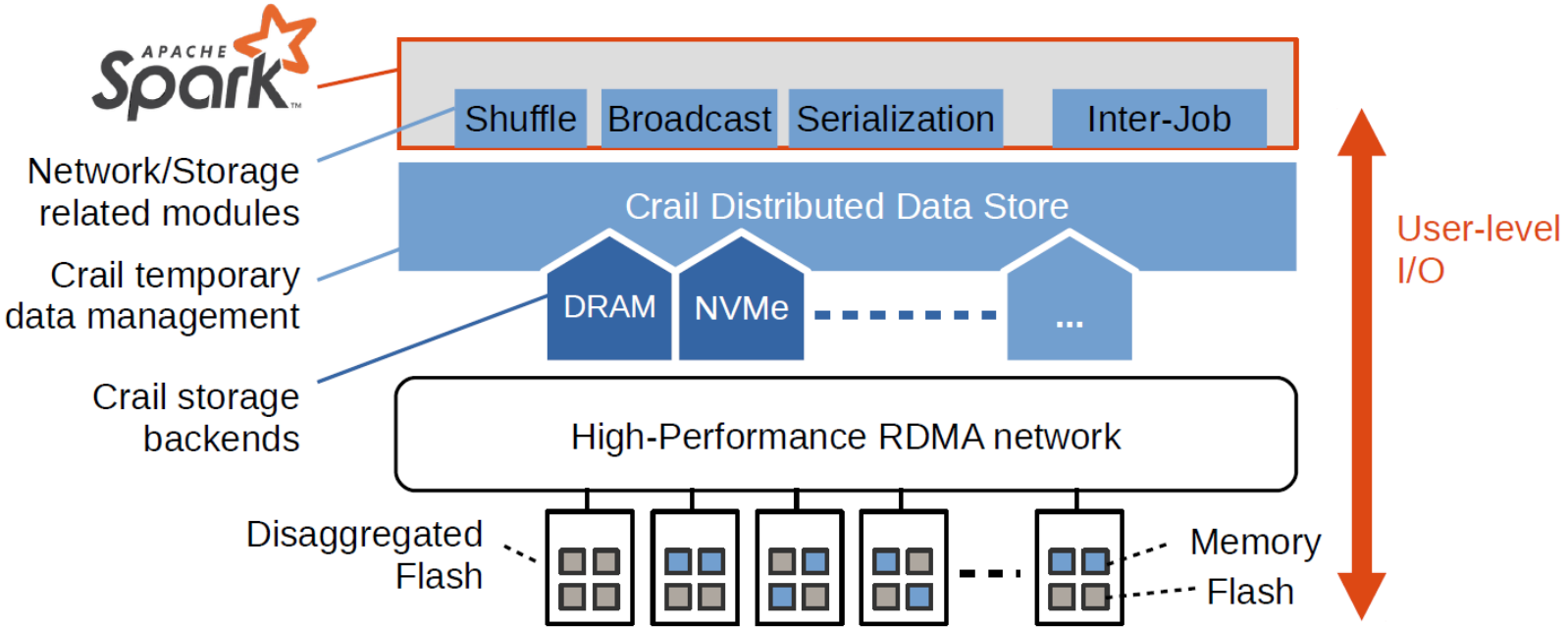
Example: Shuffle (Map+Reduce)



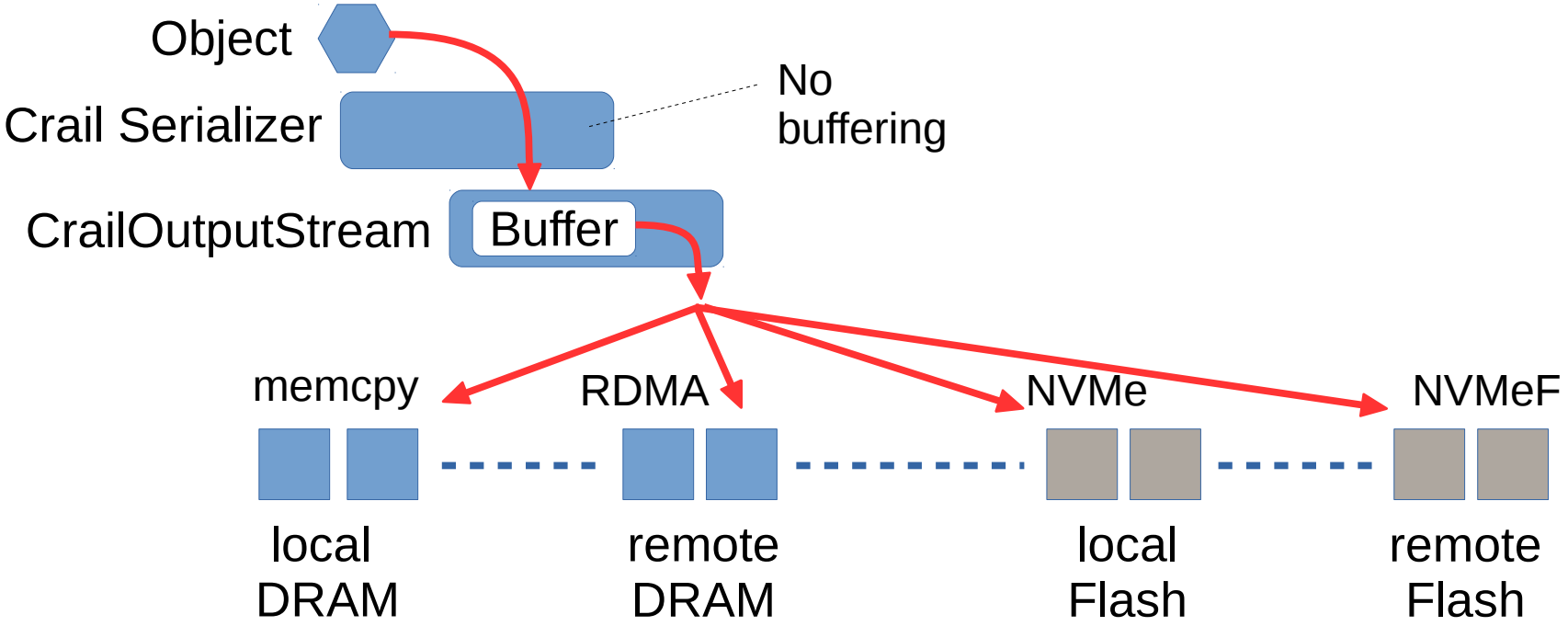
How can we fix this...

- 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

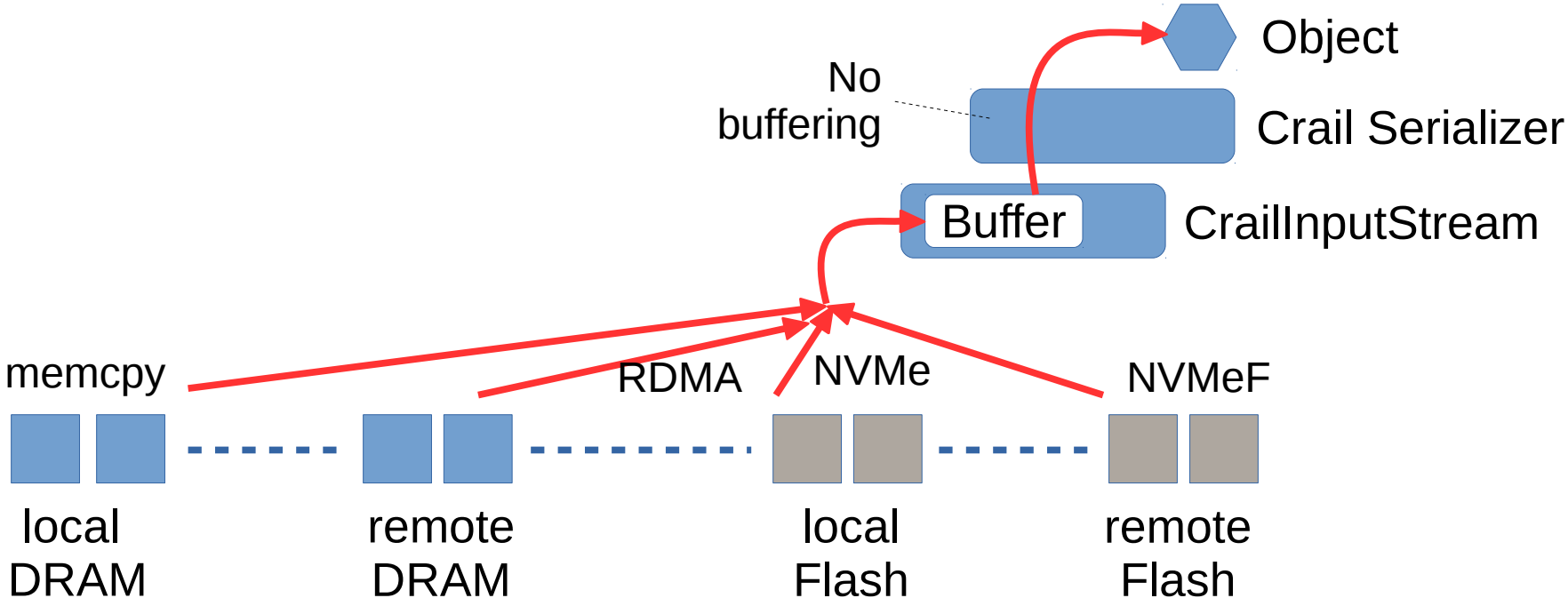
The CRAIL Approach



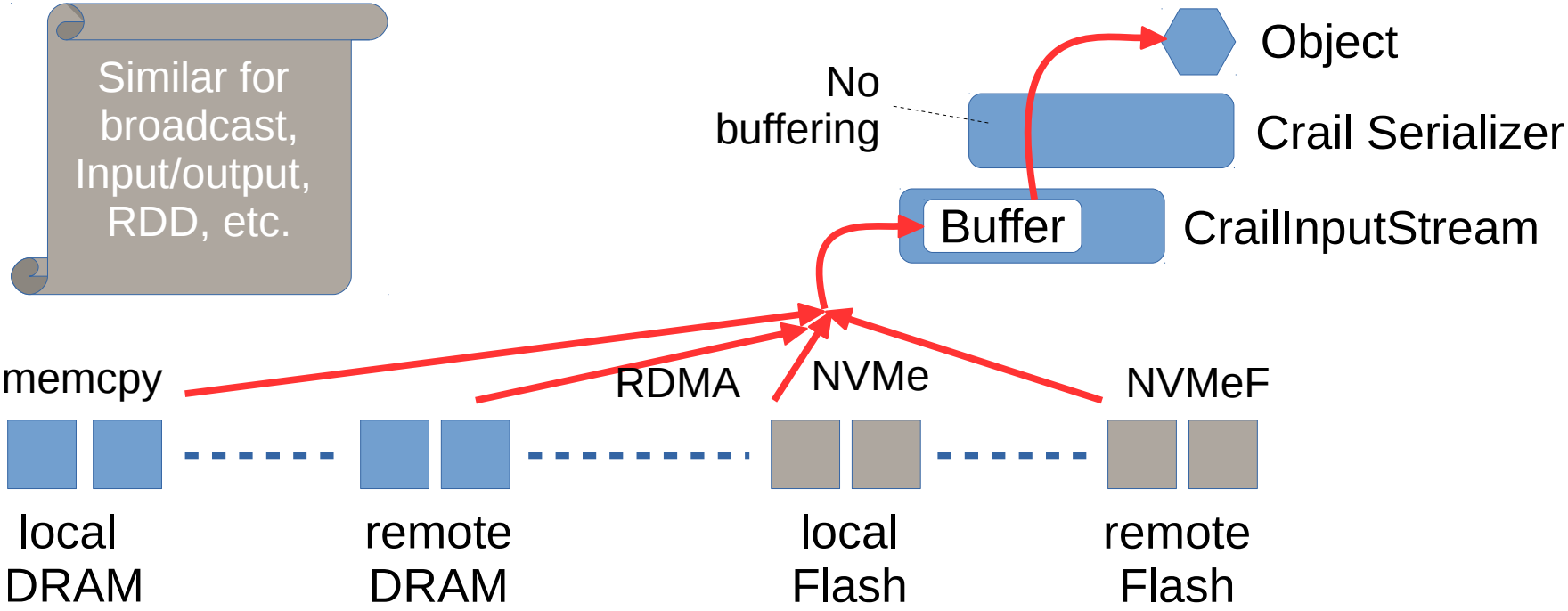
Example: Crail Shuffle (Map)



Example: Crail Shuffle (Reduce)



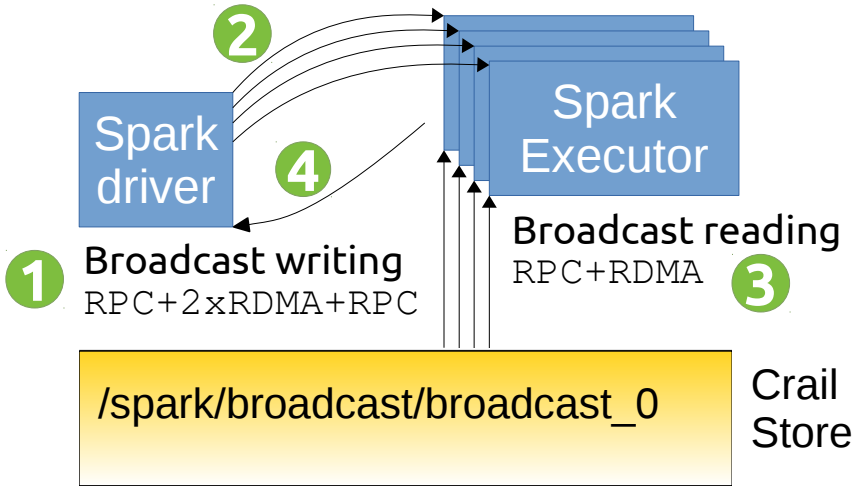
Example: Crail Shuffle (Reduce)

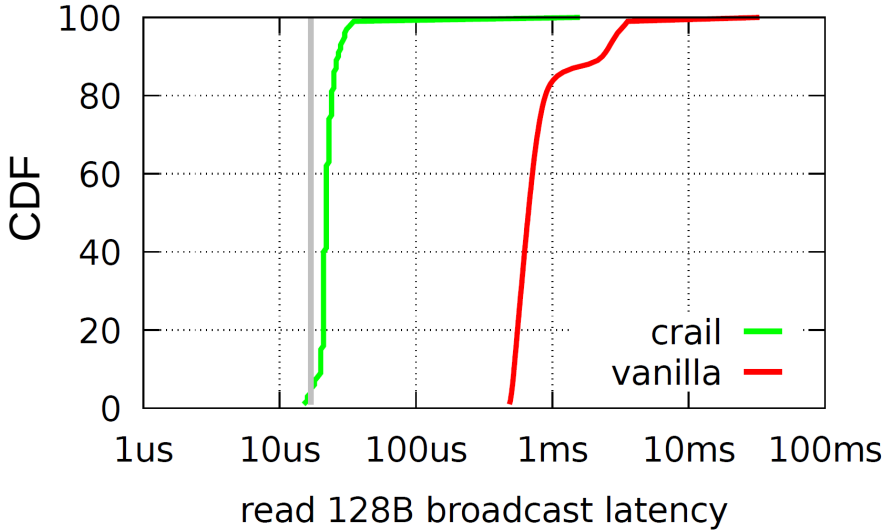


Performance: Configuration

- Experiments
  - Performance: Broadcast, GroupBy, Sorting, SQL (memory)
  - Disaggregation, tiering (memory/flash)
- Cluster size: 8 nodes, except Sorting: 128 nodes
- Cluster hardware:
  - DRAM: 512 GB DDR 4
  - Storage: 4x 1.2 TB NVMe SSD
  - Network: 100GbE Mellanox RDMA
- Software
  - Spark 2.1.0
  - “hadoop.tmp.dir”: RamFS for microbenchmarks, flash SSD for workloads

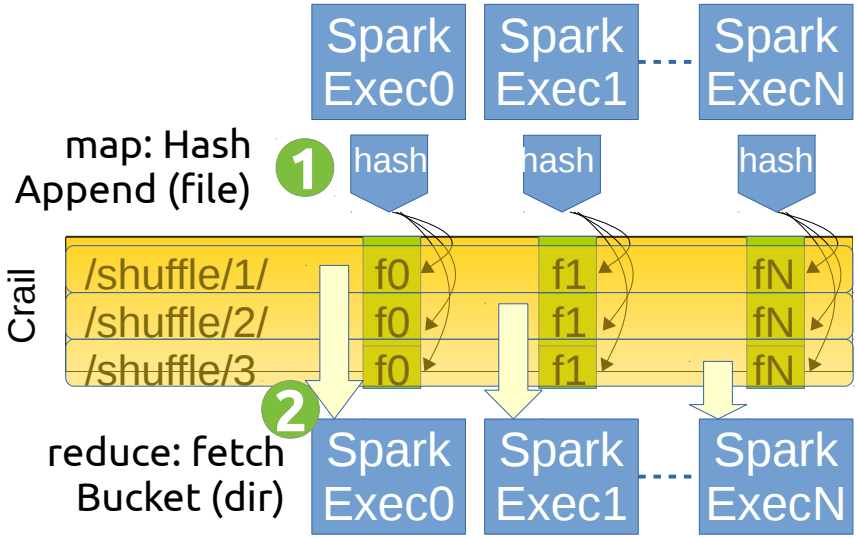
Spark Broadcast

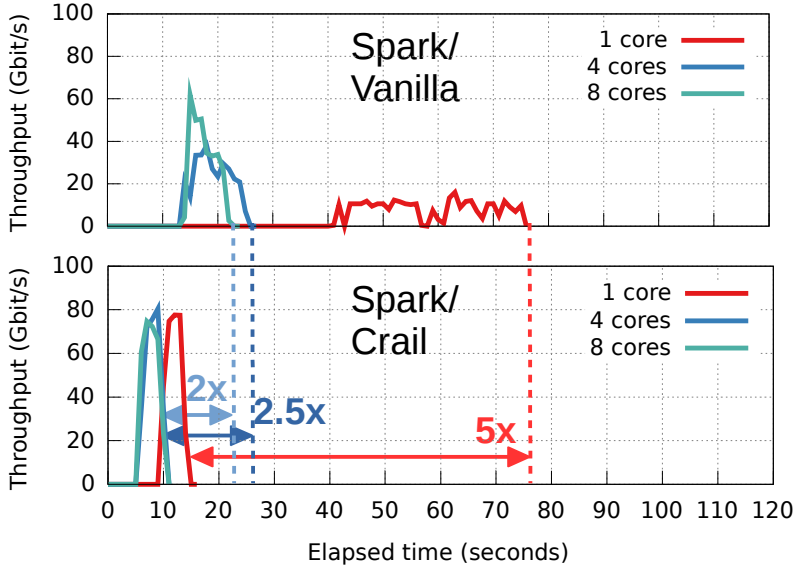





```
val bcVar = sc.Broadcast(new Array[Byte](128))
sc.parallelize(1 to tasks, tasks).map(_ => {
  bcVar.value.length
}).count
```

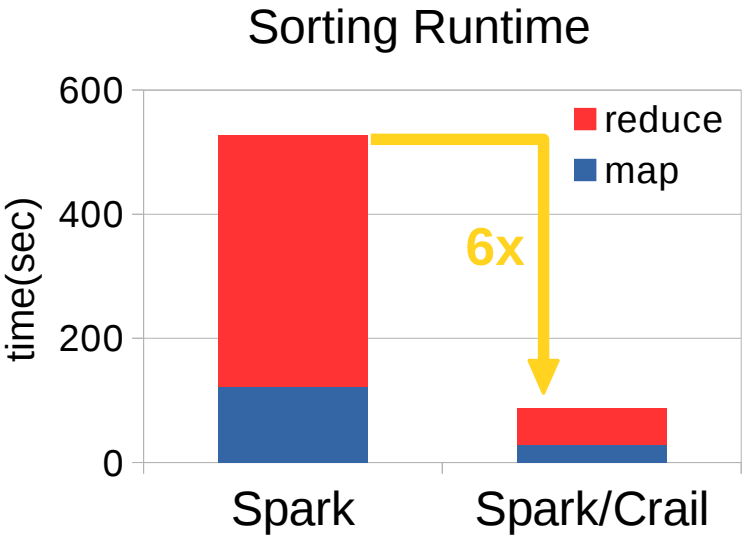
Spark GroupBy (80M keys, 4K)

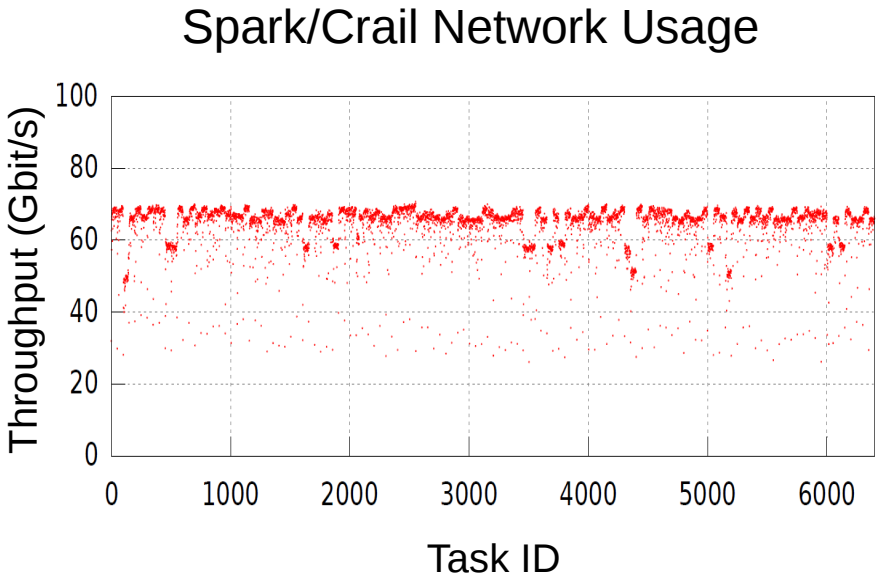





```
val pairs = sc.parallelize(1 to tasks, tasks).flatmap(_ => {
  var values = new array[(Long,Array[Byte])](numKeys)
  values = initValues(values)
}).cache().groupByKey().count()
```

Sorting 12.8 TB on 128 nodes





How fast is this?

Spark

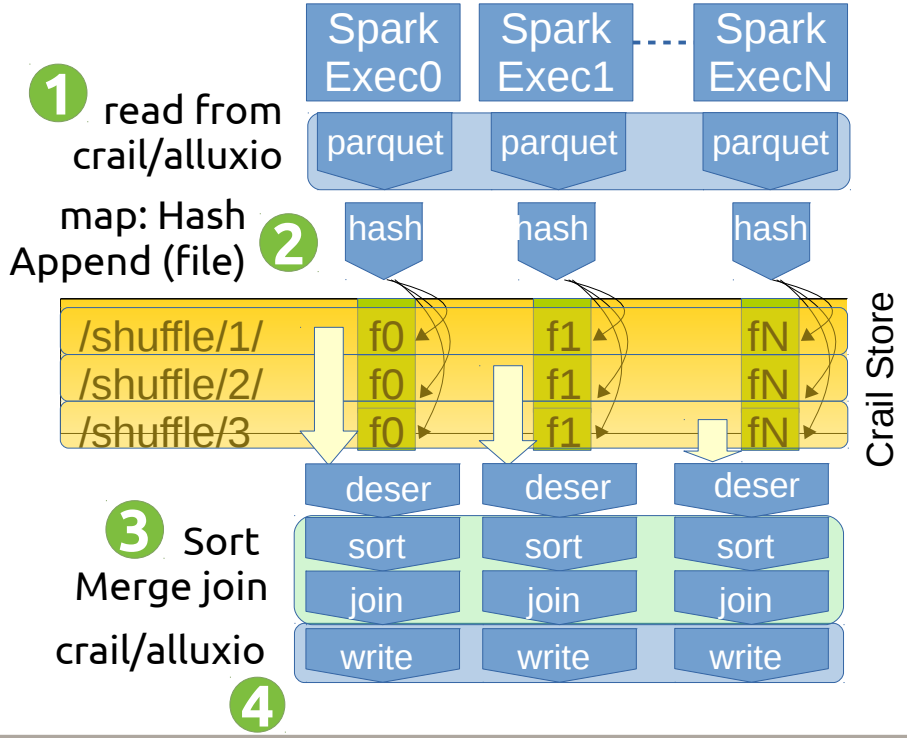
Native C distributed sorting benchmark

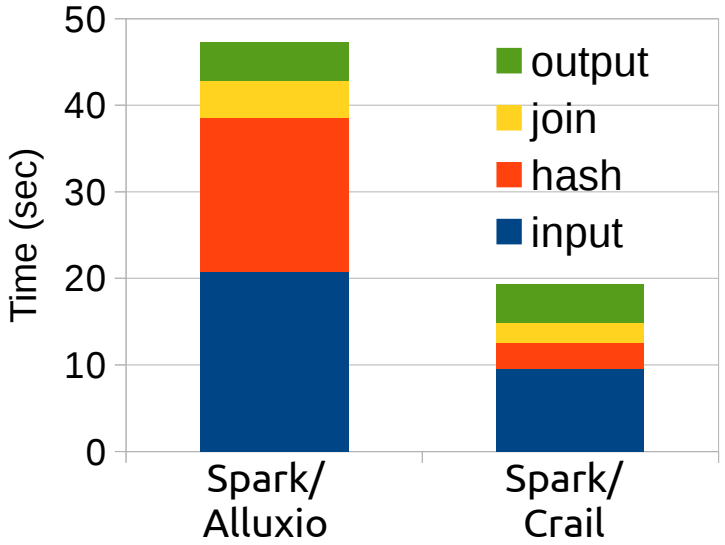
www.sortingbenchmark.org

	Spark/Crail	Winner 2014	Winner 2016
Size (TB)	12.8	100	100
Time (sec)	98	1406	134
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

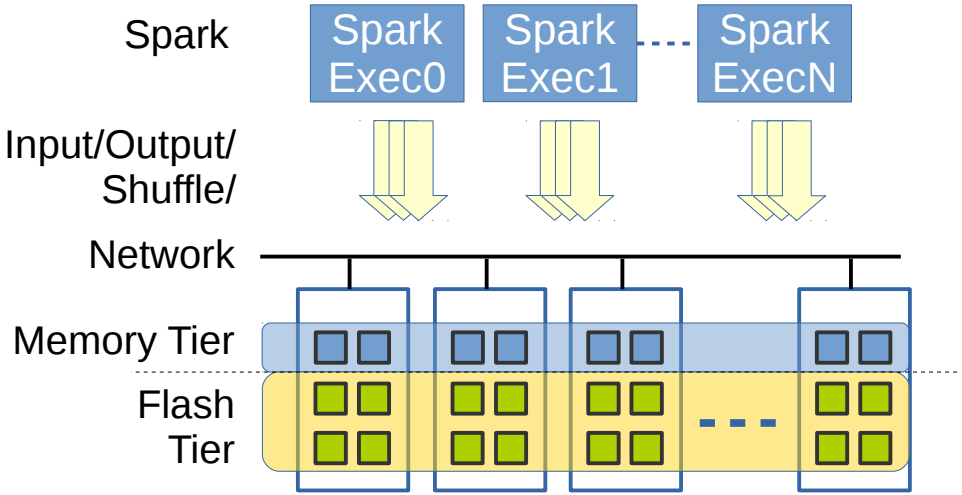
Spark SQL Join

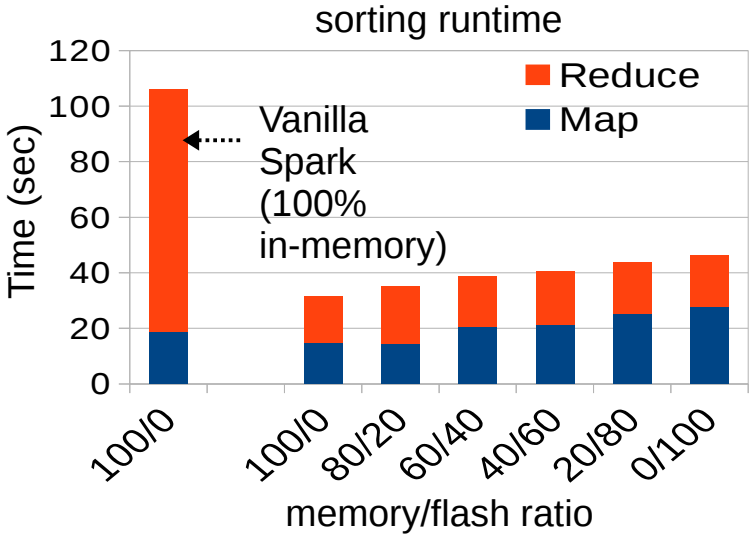





```
val ds1 = sparkSession.read.parquet("...") //64GB dataset
val ds2 = sparkSession.read.parquet("...") //64GB dataset
val resultDS = ds1.joinWith(ds2, ds1("key") === ds2("key")) //~100GB
resultDS.write.format("parquet").mode(SaveMode.Overwrite).save("...")
```

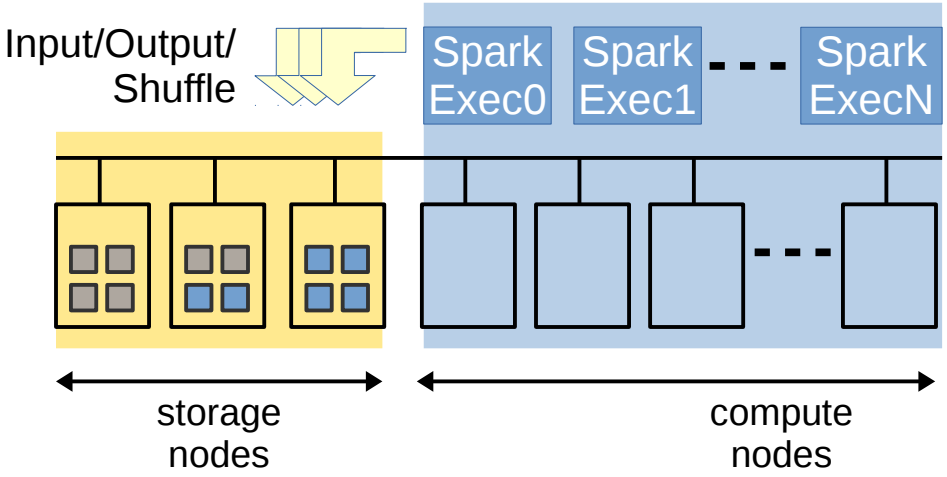
Storage Tiering: DRAM & NVMe

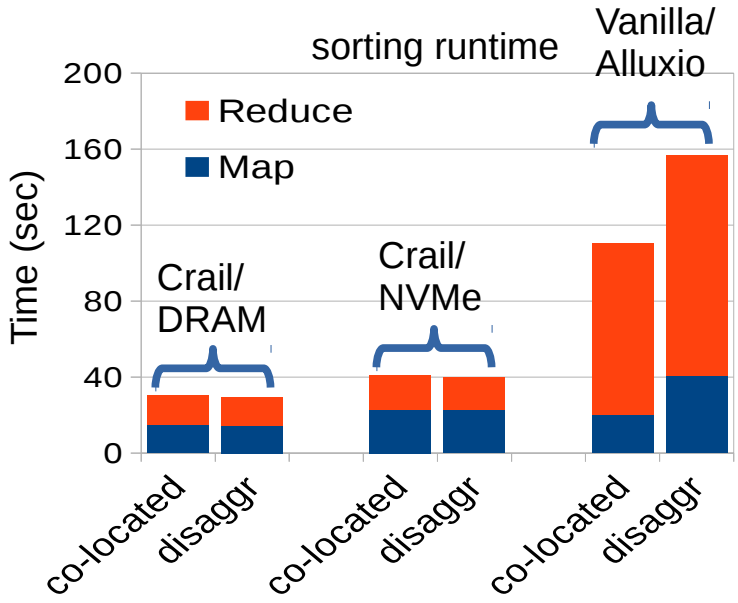




With Crail, replacing memory with NVMe flash
for storing input/output/shuffle reduces the 200GB sorting runtime by only 48%

DRAM & Flash Disaggregation





Using Crail, a Spark 200GB sorting workload can be run with memory and flash disaggregated at no extra cost

Conclusions

- Effectively using high-performance I/O hardware in Spark is challenging
- Crail is an attempt to re-think how data processing frameworks (not only Spark) should interact with network and storage hardware
  - User-level I/O, storage disaggregation, memory/flash convergence
- Spark's modular architecture allows Crail to be used seamlessly

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.

Contributors:
Animesh Trivedi, Jonas Pfefferle, Michael Kaufmann, Bernard Metzler, Radu Stoica, Ioannis Koltsidas, Nikolos Ioannou, Christoph Hagleitner, Peter Hofstee, Evangelos Eleftheriou, ...