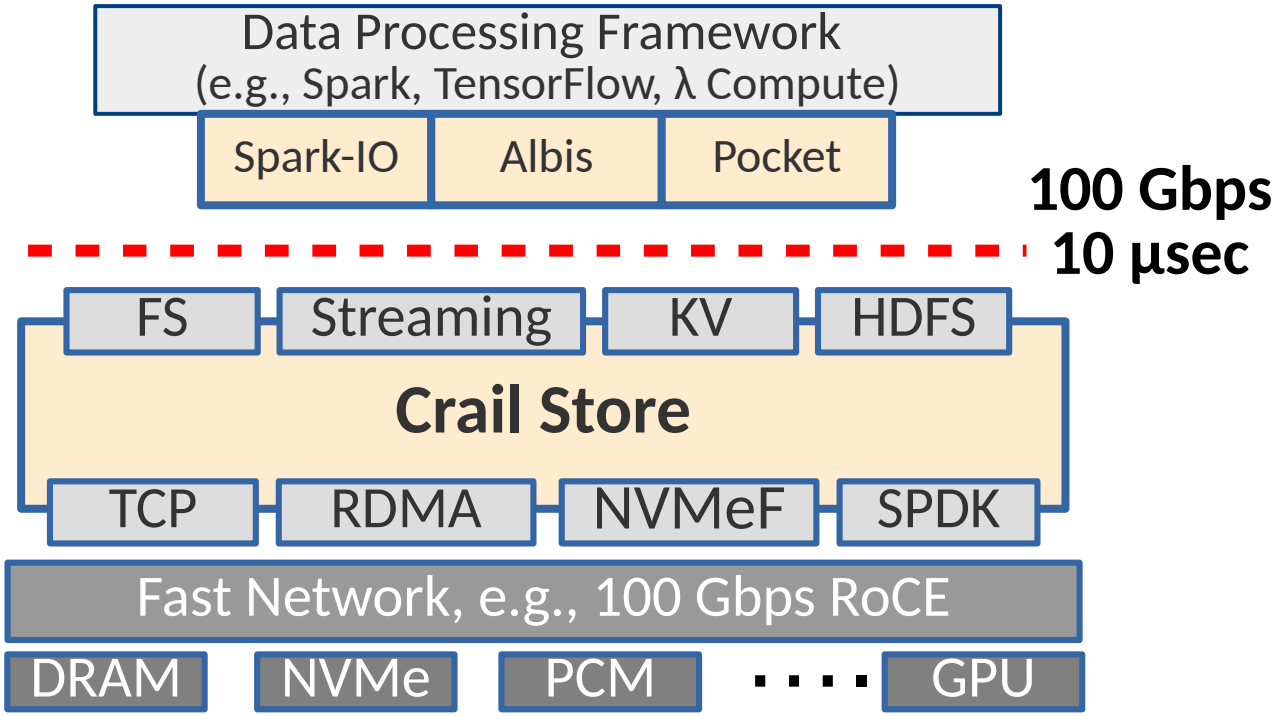
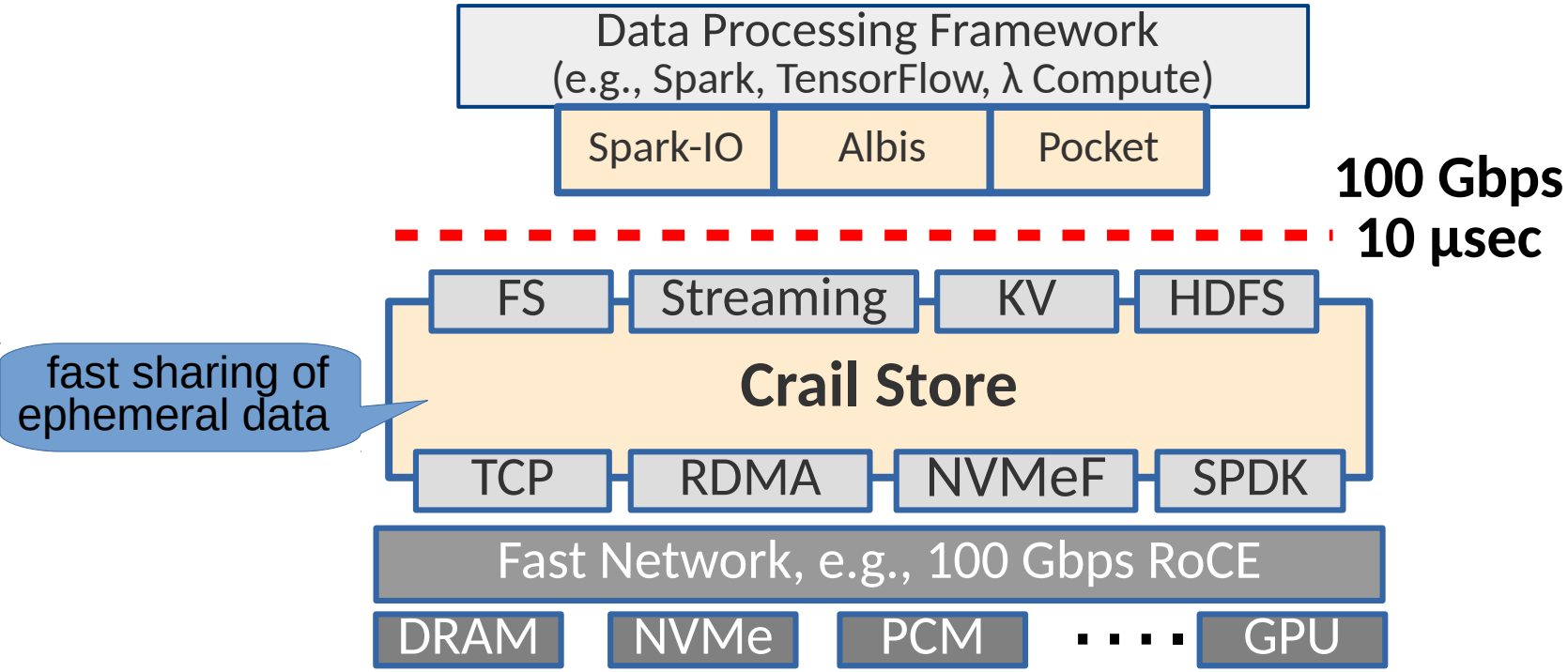
Data Processing at the Speed of 100 Gbps using Apache Crail

Patrick Stuedi
IBM Research

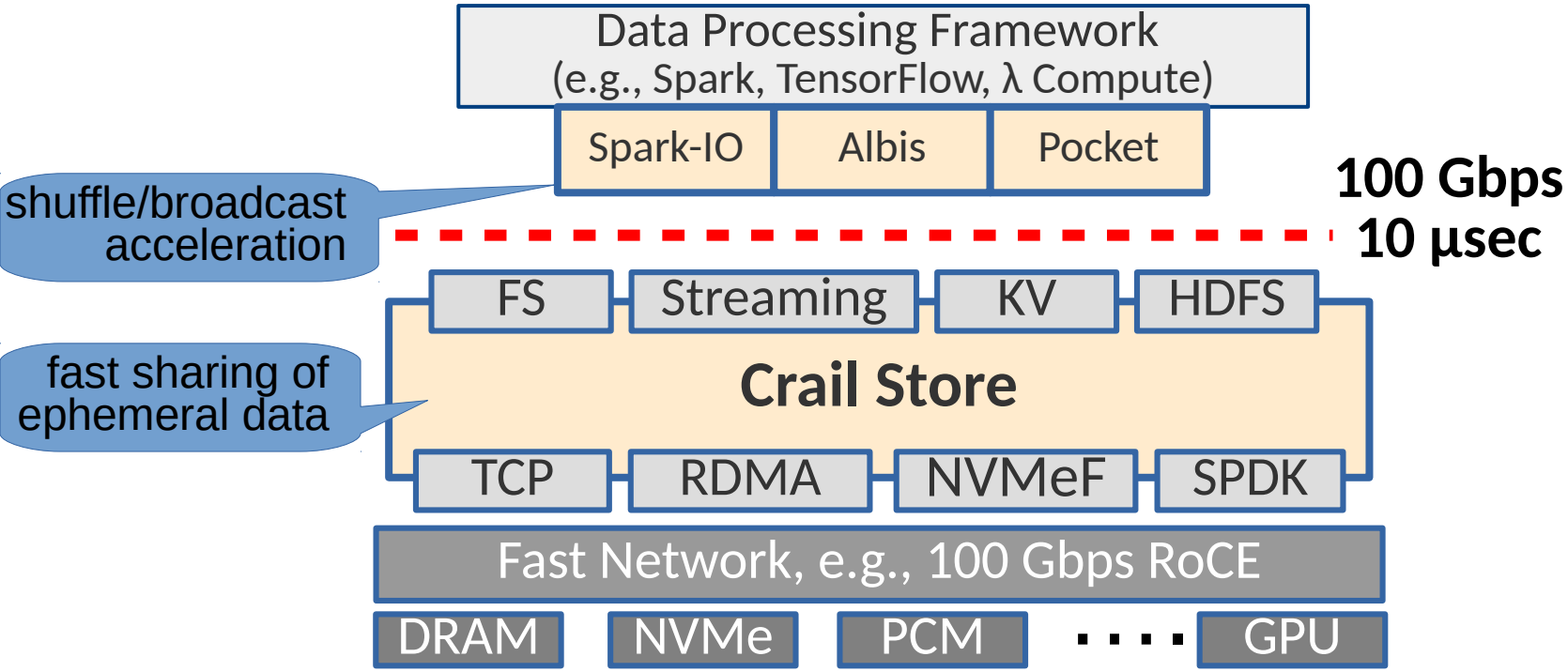
The CRAIL Project: Overview



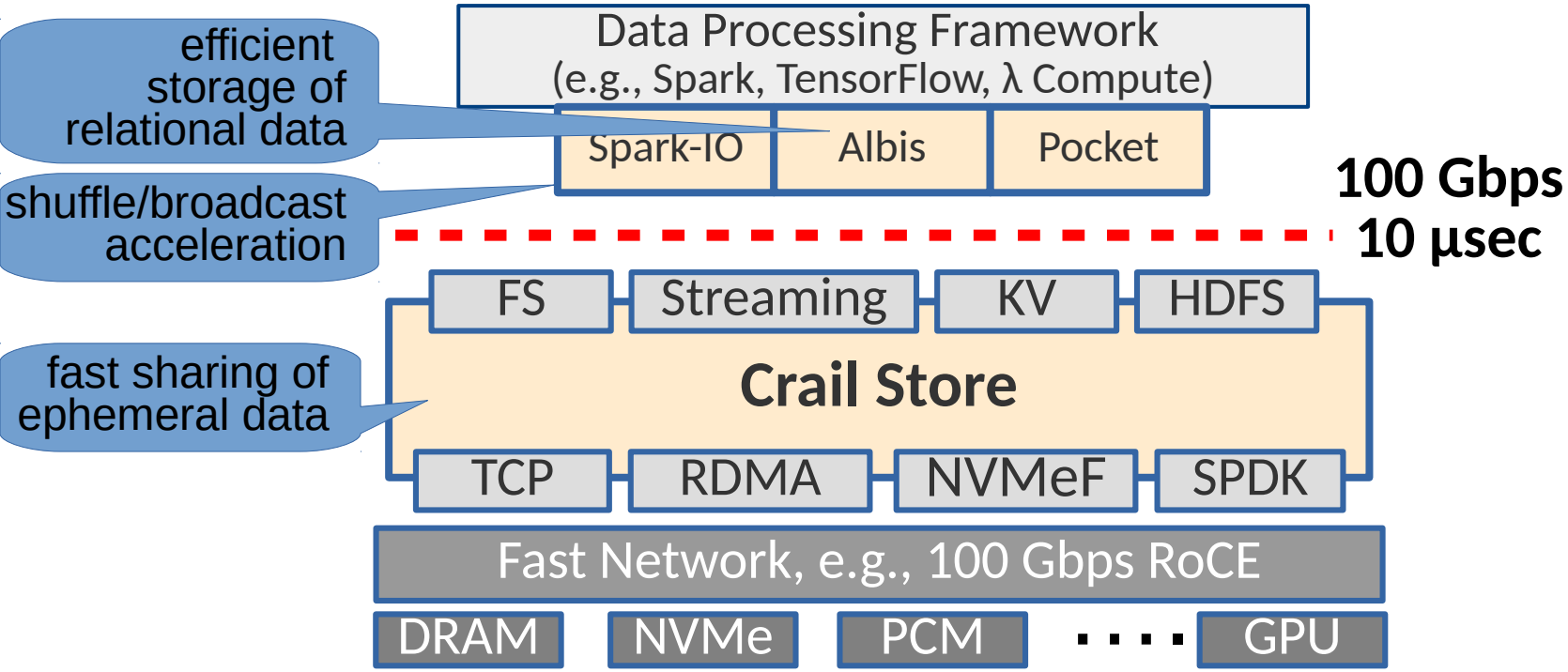
The CRAIL Project: Overview



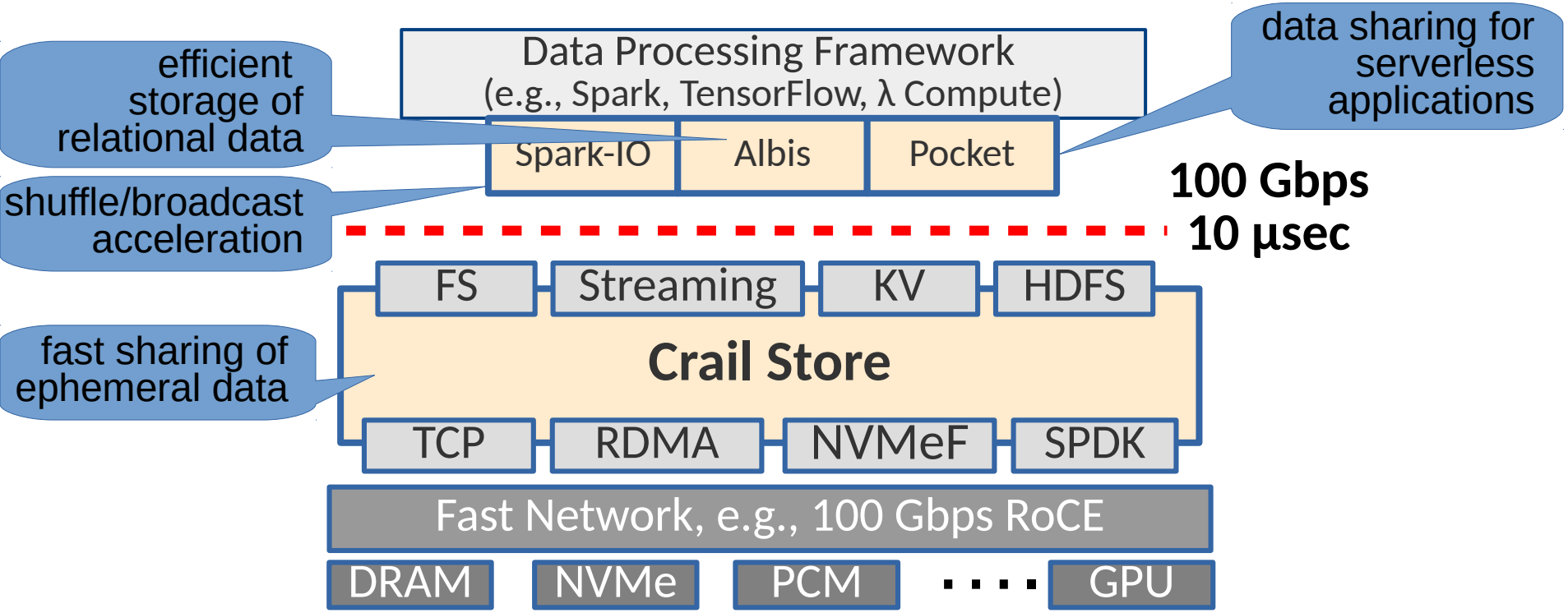
The CRAIL Project: Overview



The CRAIL Project: Overview



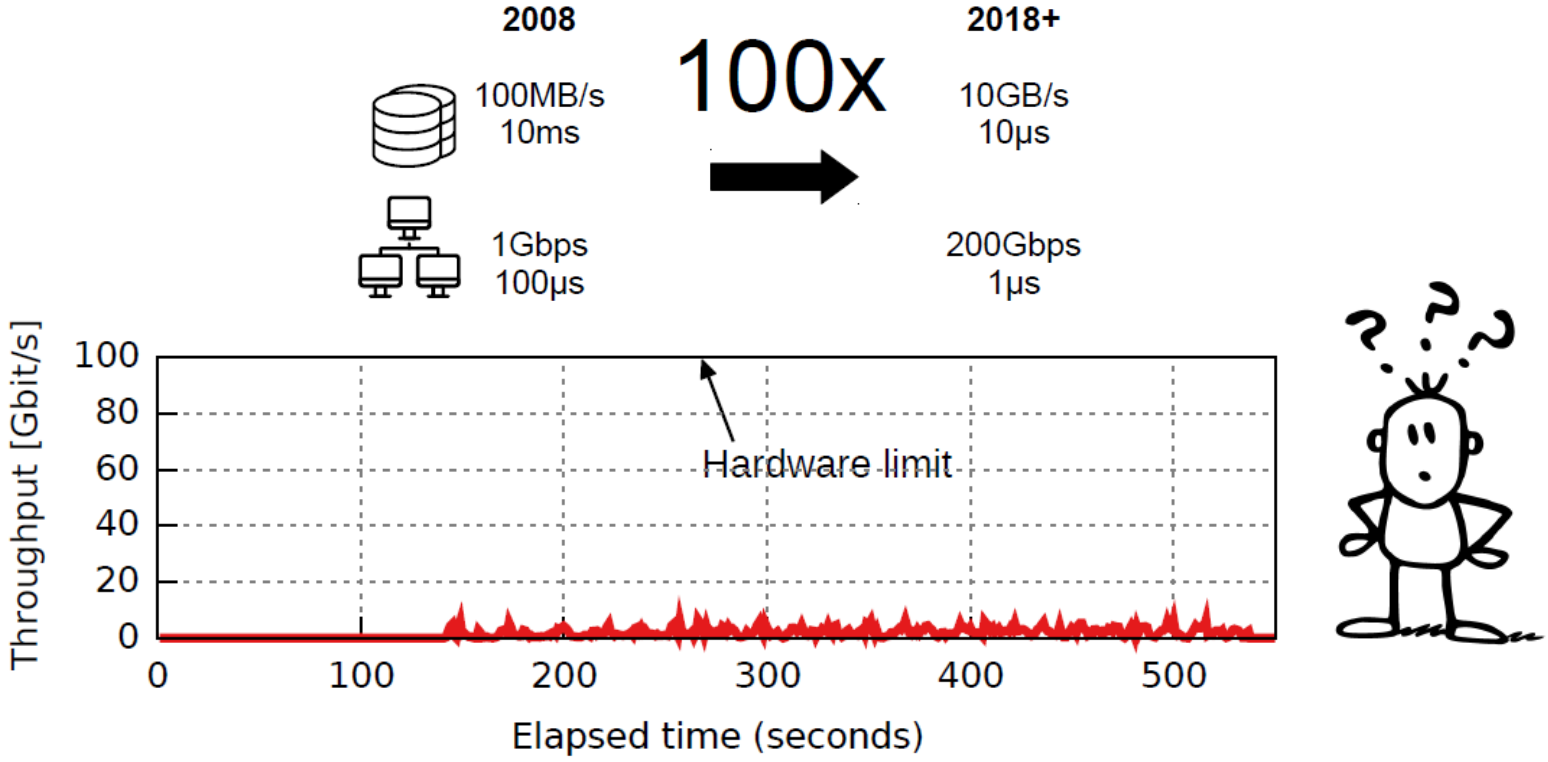
The CRAIL Project: Overview



Outline

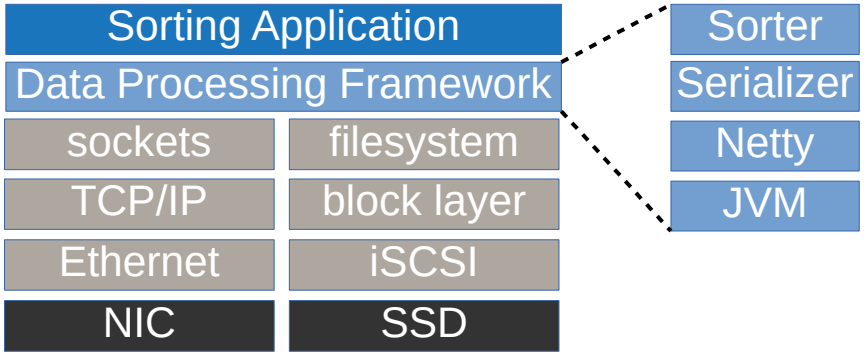
- Why CRAIL
- Crail Store
- Workload specific I/O Processing
  - File Format, shuffle engine, serverless
- Use Cases:
  - Disaggregation
  - Workloads: SQL, Machine Learning

#1 Performance Challenge (1)



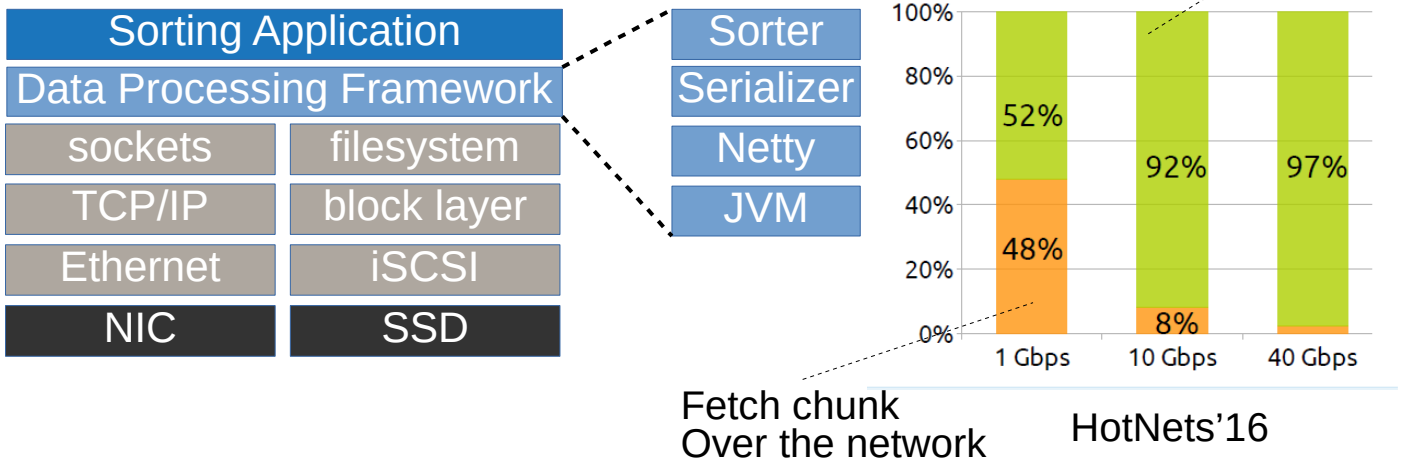
#1 Performance Challenge (2)

	1 Gbps	HDD	100 Gbps	Flash
Bandwidth	117 MB/s	140 MB/s	12.5 GB/s	3.1 GB/s
cycle/unit	38,400	10,957	360	495



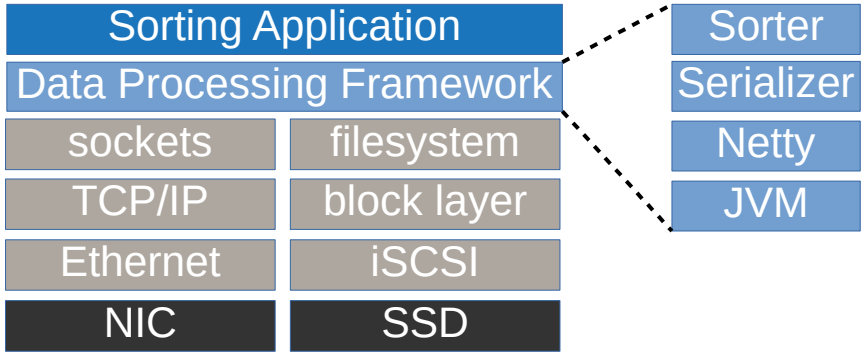
#1 Performance Challenge (2)

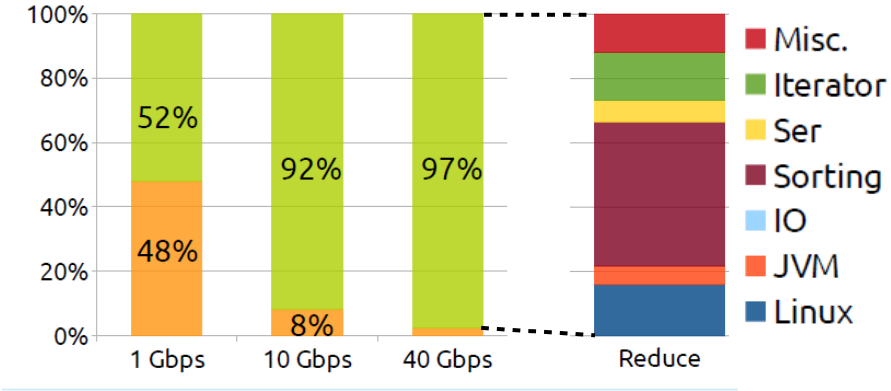
	1 Gbps	HDD	100 Gbps	Flash
Bandwidth	117 MB/s	140 MB/s	12.5 GB/s	3.1 GB/s
cycle/unit	38,400	10,957	360	495



#1 Performance Challenge (2)

	1 Gbps	HDD	100 Gbps	Flash
Bandwidth	117 MB/s	140 MB/s	12.5 GB/s	3.1 GB/s
cycle/unit	38,400	10,957	360	495

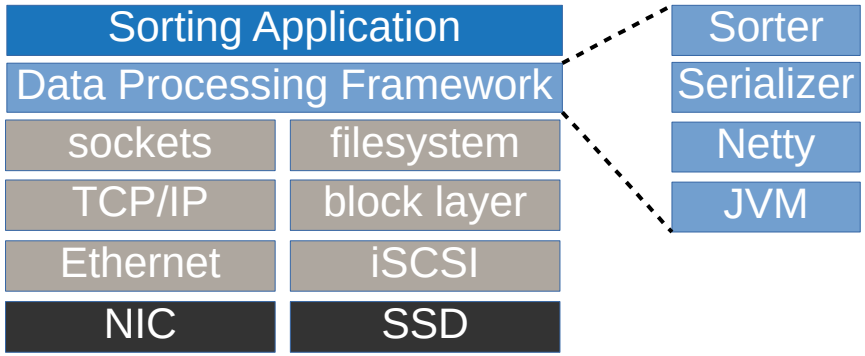


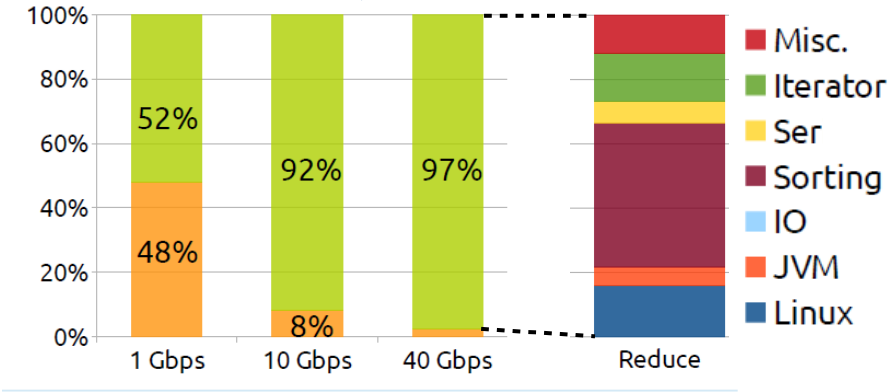


#1 Performance Challenge (2)

	1 Gbps	HDD	100 Gbps	Flash
Bandwidth	117 MB/s	140 MB/s	12.5 GB/s	3.1 GB/s
cycle/unit	38,400	10,957	360	495

software overhead are spread over the entire stack





#2 Diversity

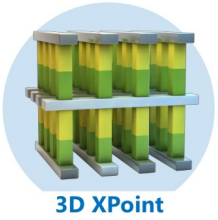
APACHE OpenWhisk™

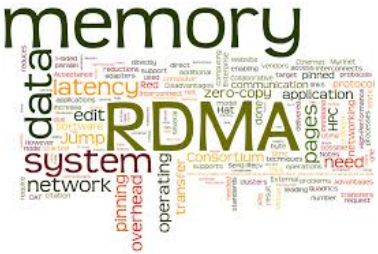
NVMe OVER FABRICS POWERED BY RoCE

SPDK

DPDK
DATA PLANE DEVELOPMENT KIT

RDMA Verbs

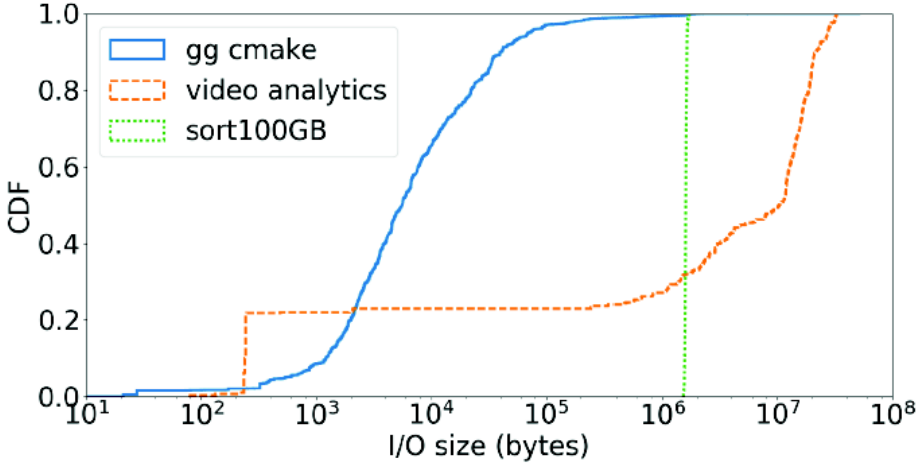




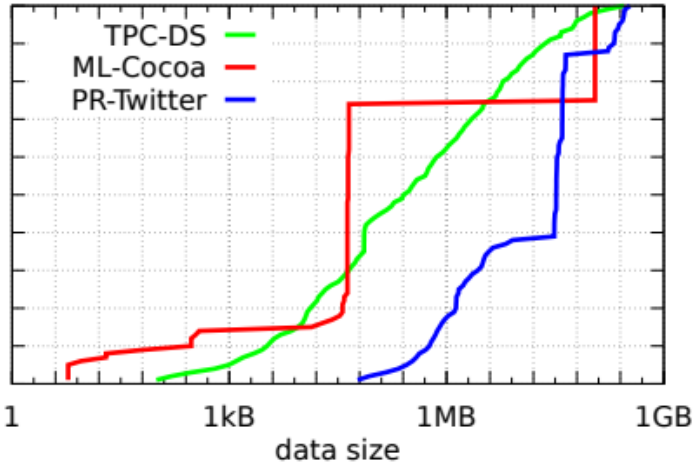
Diverse hardware technologies / complex programming APIs / many frameworks

#3 Ephemeral Data

serverless (AWS lambda) applications



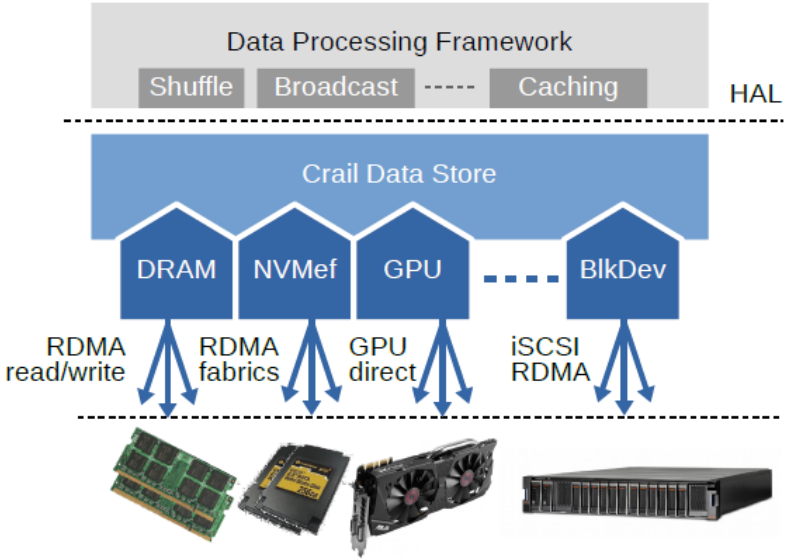
Spark applications



Ephemeral data has unique properties (e.g., wide range of I/O size)

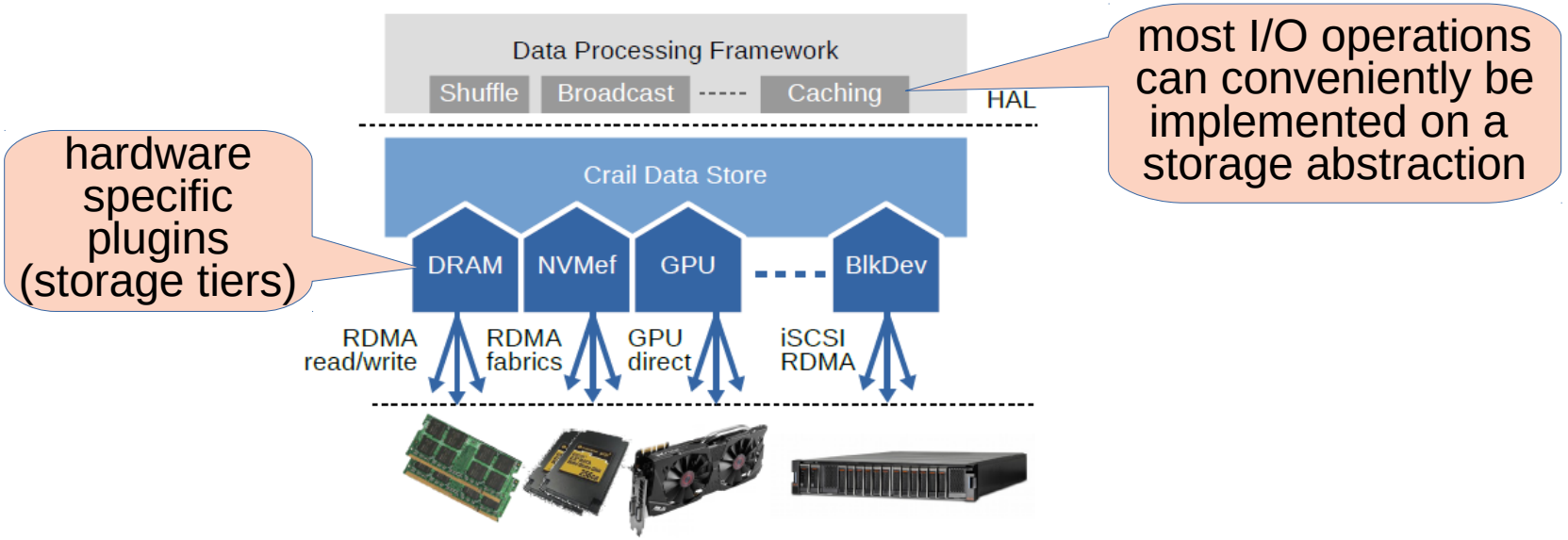
CRAIL Approach (1)

Abstract hardware via high-level storage interface



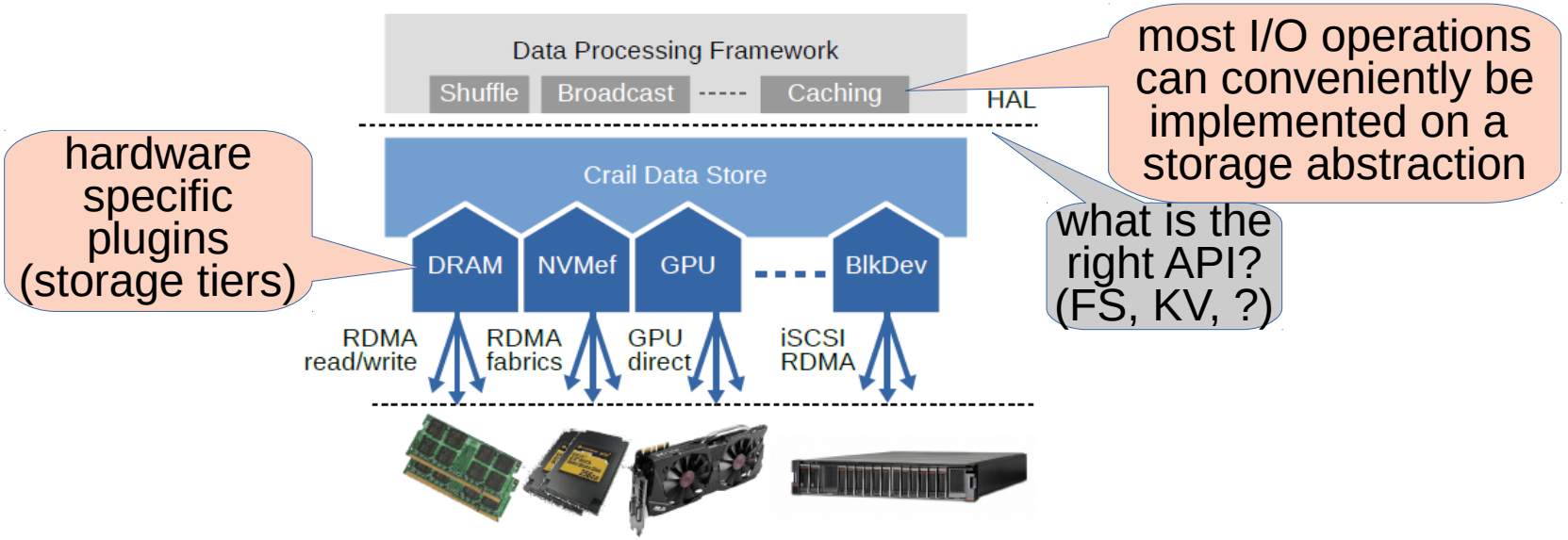
CRAIL Approach (1)

Abstract hardware via high-level storage interface



CRAIL Approach (1)

Abstract hardware via high-level storage interface



CRAIL Approach (2)

Filesystem-like interface:

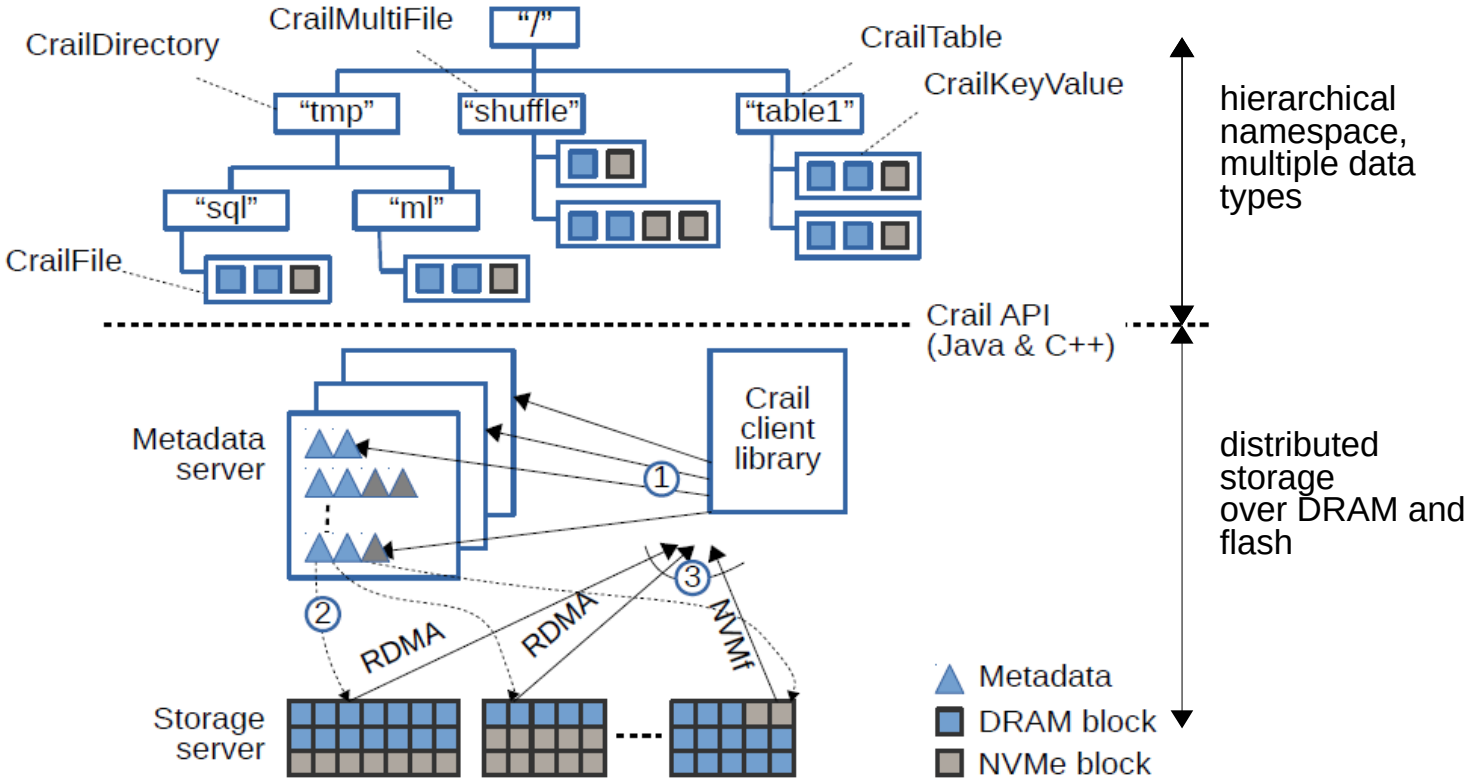
- Hierarchical namespace
  - Helps to organize data (shuffle, tmp, etc) for different jobs
- Separate data from metadata plane
  - Reading/writing involves block metatdata lookup
  - Cheap on a low-latency network (few usecs)
  - Flexible: data objects can be of arbitrary size
- Specific data types
  - KeyValue files: last create wins
  - Shuffle files: efficient reading of multiple files in a directory
- Let applications control the details
  - Data placement policy: which storage node or storage tier to use

CRAIL Approach (3)

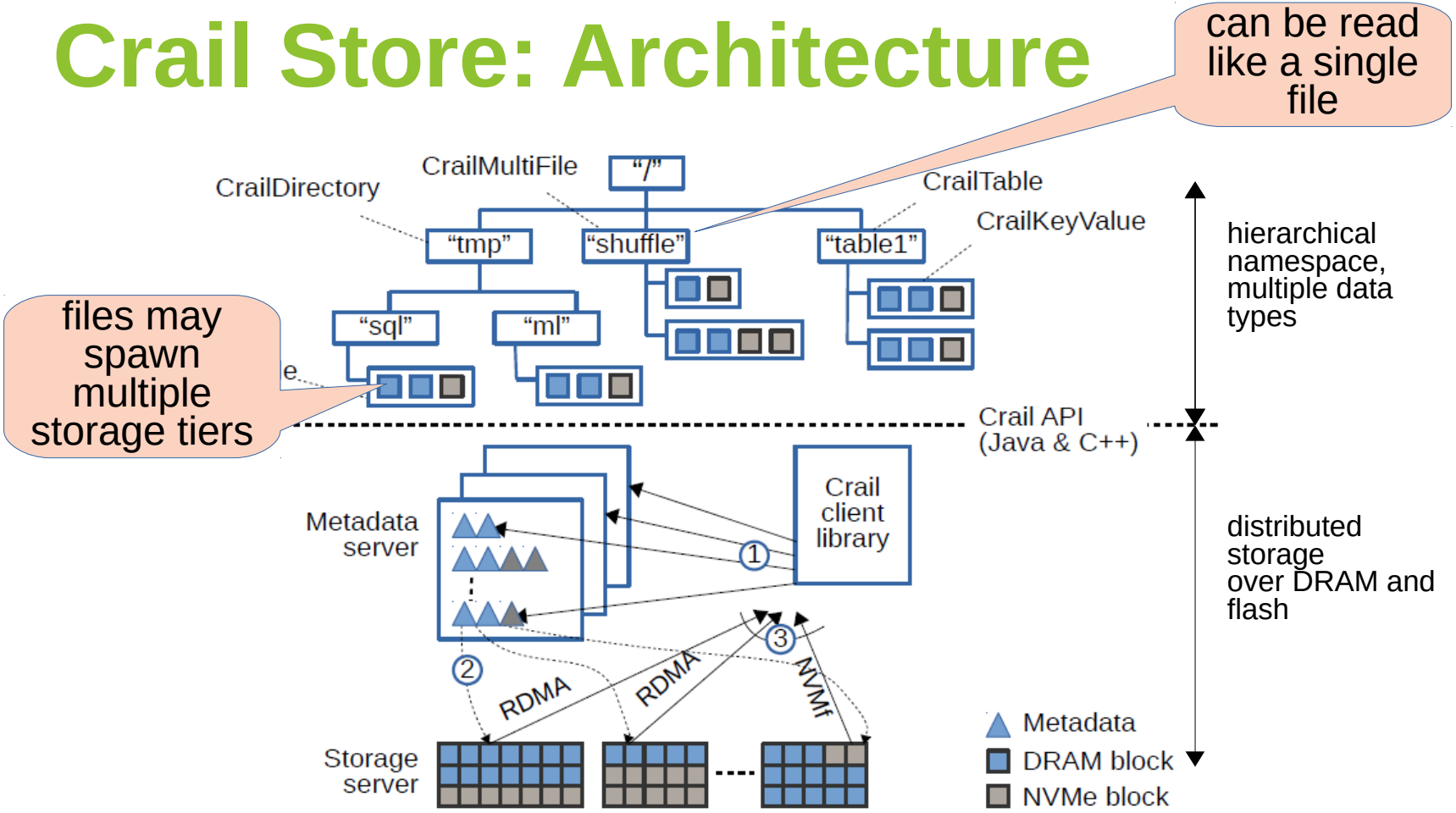
Careful software design:

- Leverage user-level APIs
  - RDMA, NFMf, DPDK, SPDK
- Seperate data from control operations
  - Memory allocation, string parsing, etc.
- Efficient non-blocking operations
  - Avoid army of threads, let the hardware do the work
- Leverage byte-address storage
  - Transmit no more data than what is read/written

Crail Store: Architecture



Crail Store: Architecture



Crail Store: Deployment Modes

compute/storage
co-located

compute/storage
disaggregated

flash storage
disaggregation

Metadata server

Flash storage server

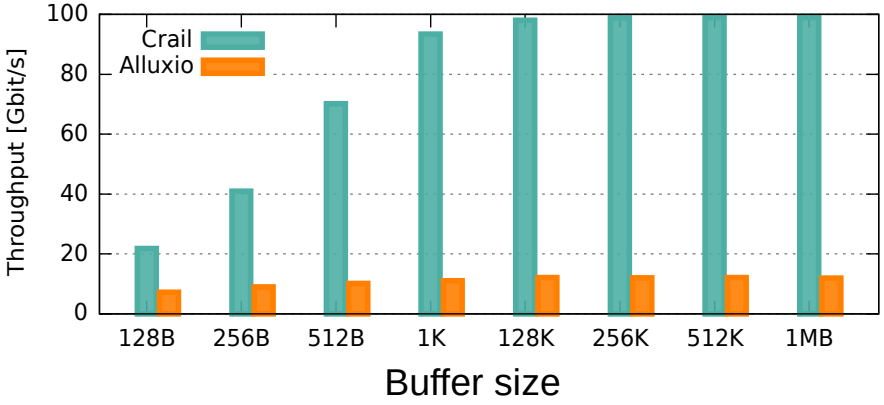
DRAM storage server

Application compute

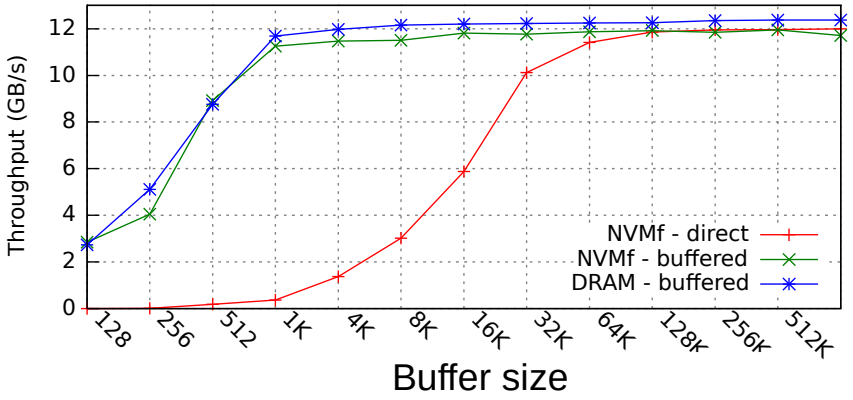
Crail Store: Read Throughput

Performance of a single client running on one core only

DRAM

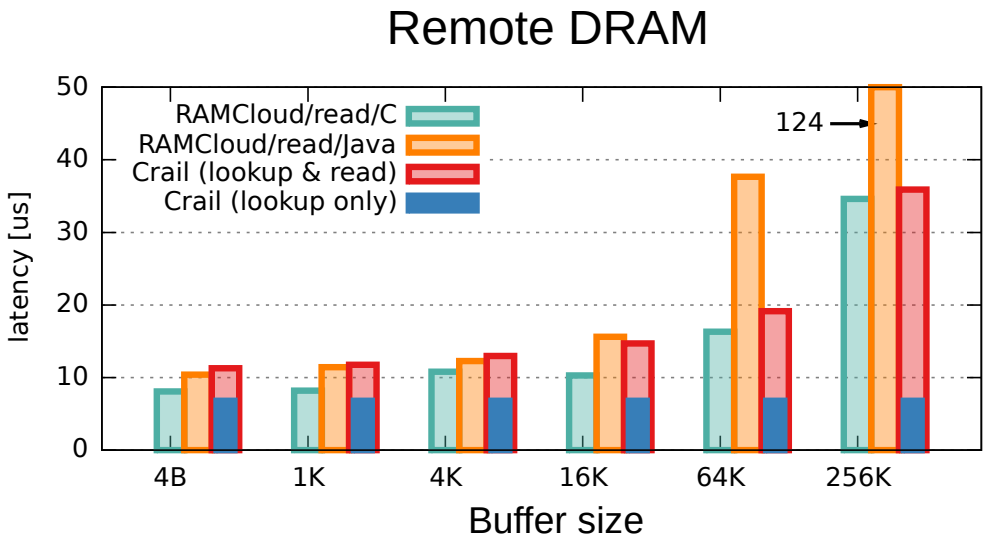


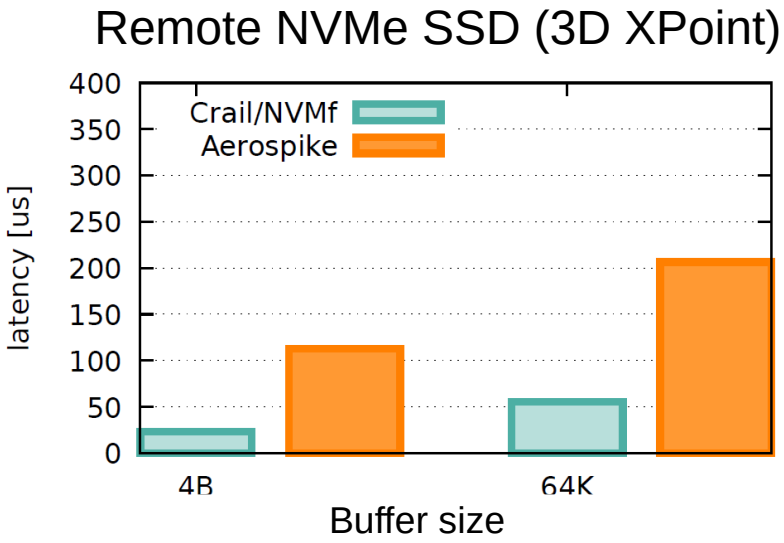
NVMf



Crail reaches line speed at for an I/O size of 1K

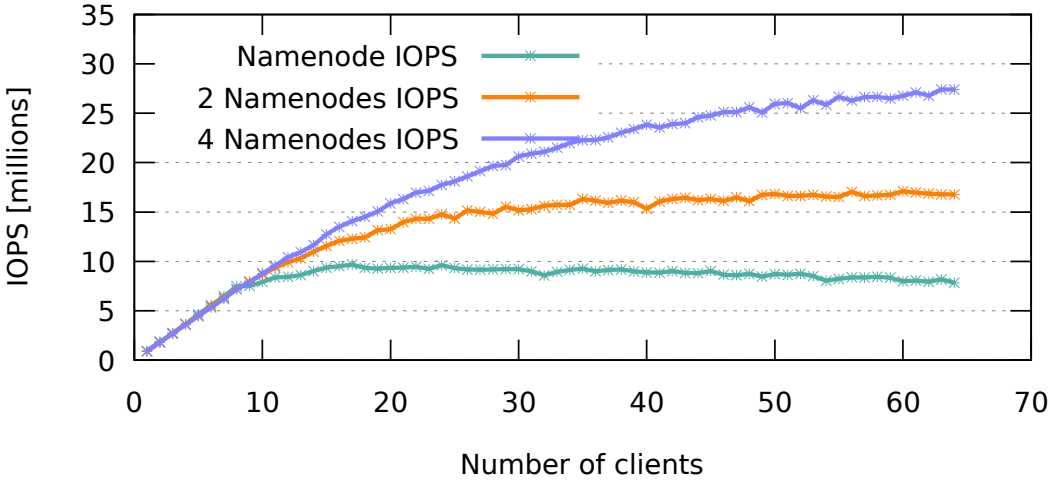
Crail Store: Read Latency

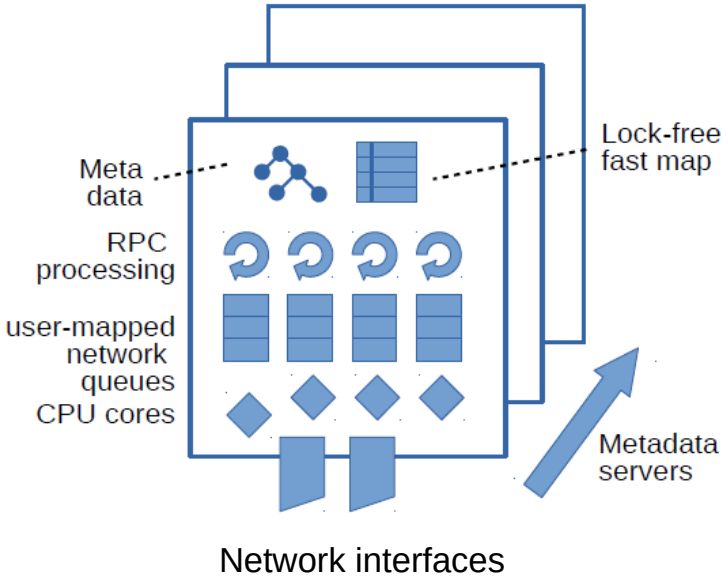




Crail remote read latencies (DRAM and NVM) are very close to the hardware latencies

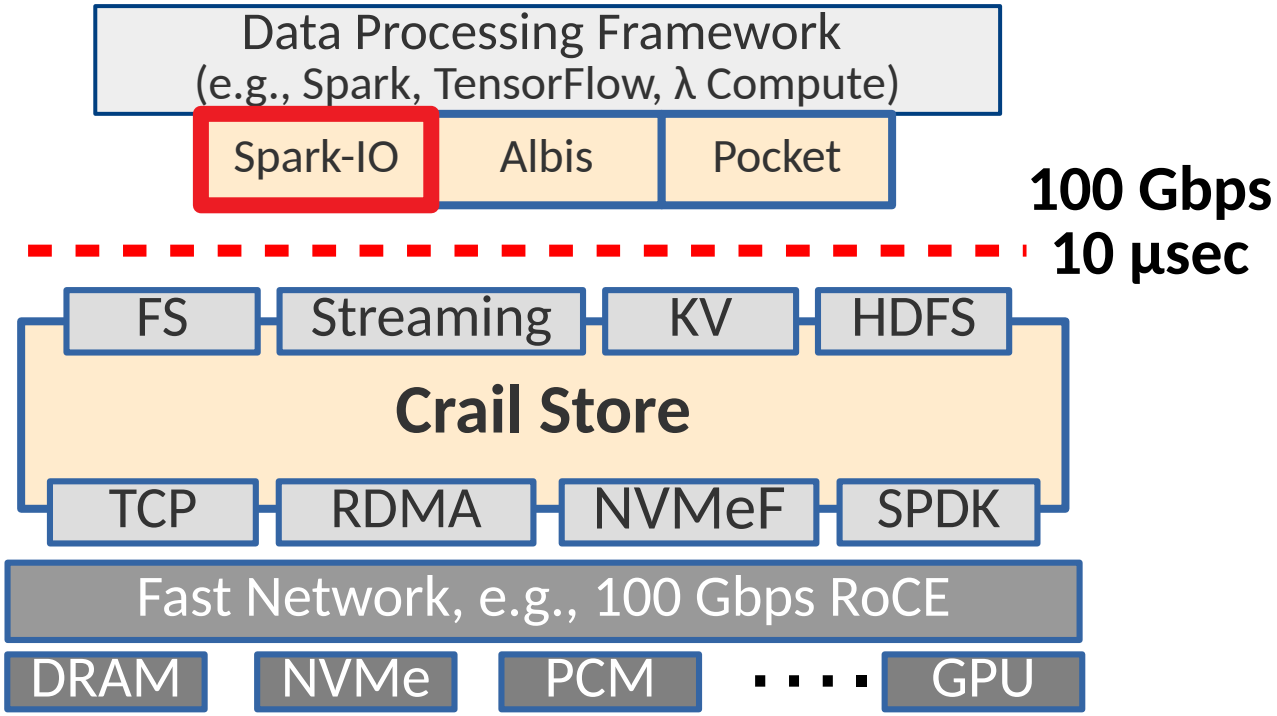
Metadata server scalability



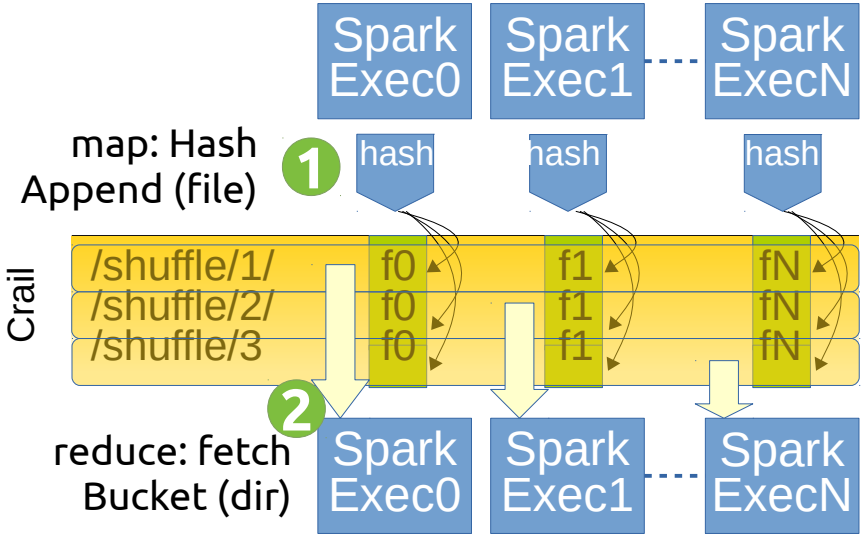


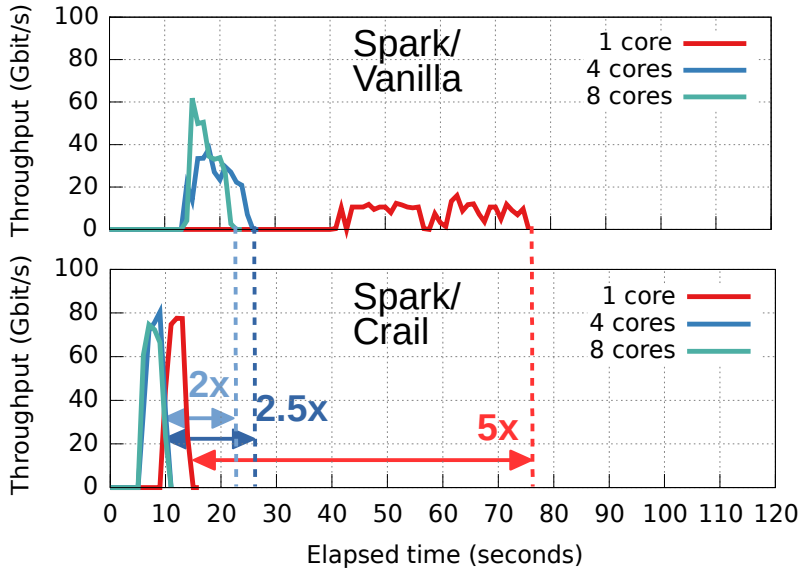
A single metadata server can process 10M Crail lookup ops/sec

Running Workloads: MapReduce



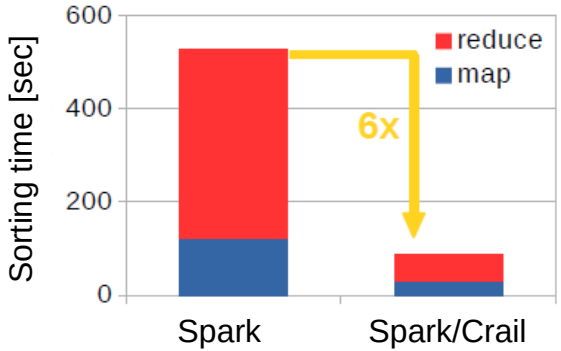
Spark GroupBy (80M keys, 4K)

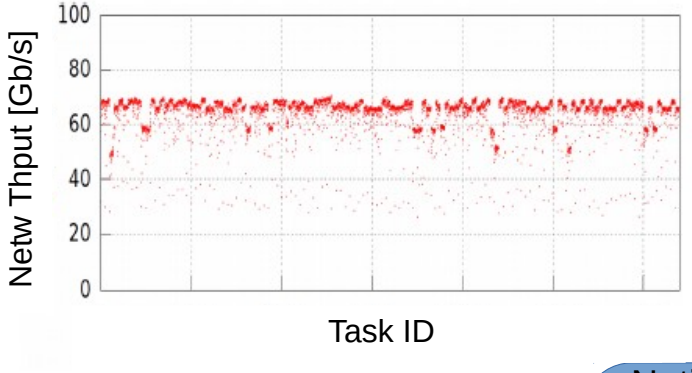





```
val pairs = sc.parallelize(1 to tasks, tasks).flatmap(_ => {
  var values = new array[(Long,Array[Byte])](numKeys)
  values = initValues(values)
}).cache().groupByKey().count()
```

Sorting 12.8 TB on 128 nodes





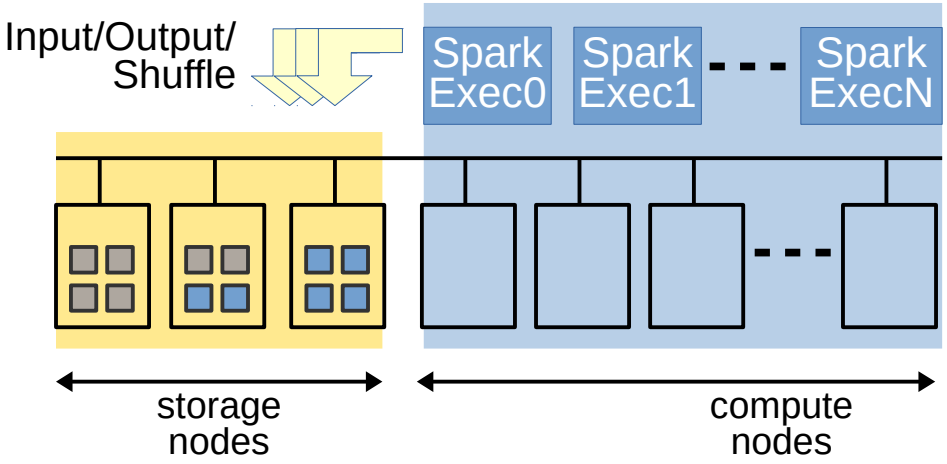
www.sortingbenchmark.org

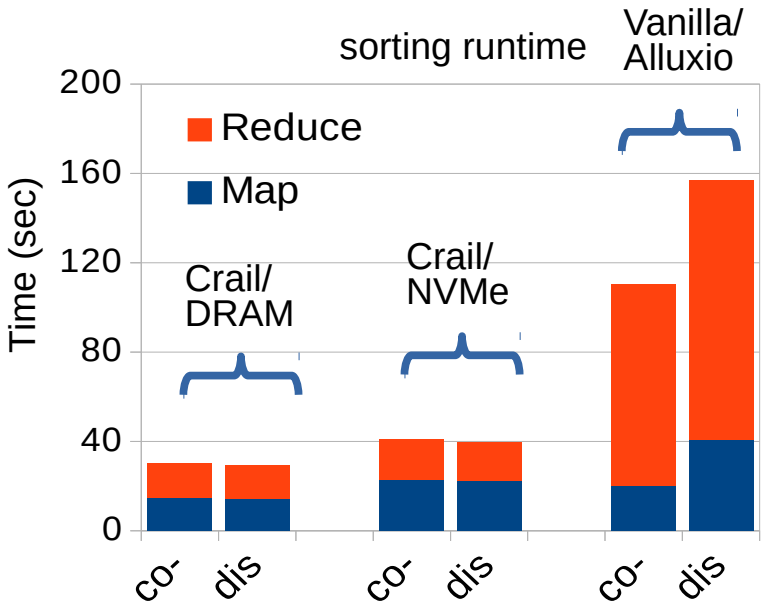
	Spark/Crail	Winner 2014	Winner 2016
Size (TB)	12.8	100	100
Time (sec)	98	1406	134
Total cores	2560	6592	10240
Network HW (Gbit/s)	100	10	100
Rate/core (GB/min)	3.13	0.66	4.4

Native C distributed sorting benchmark

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

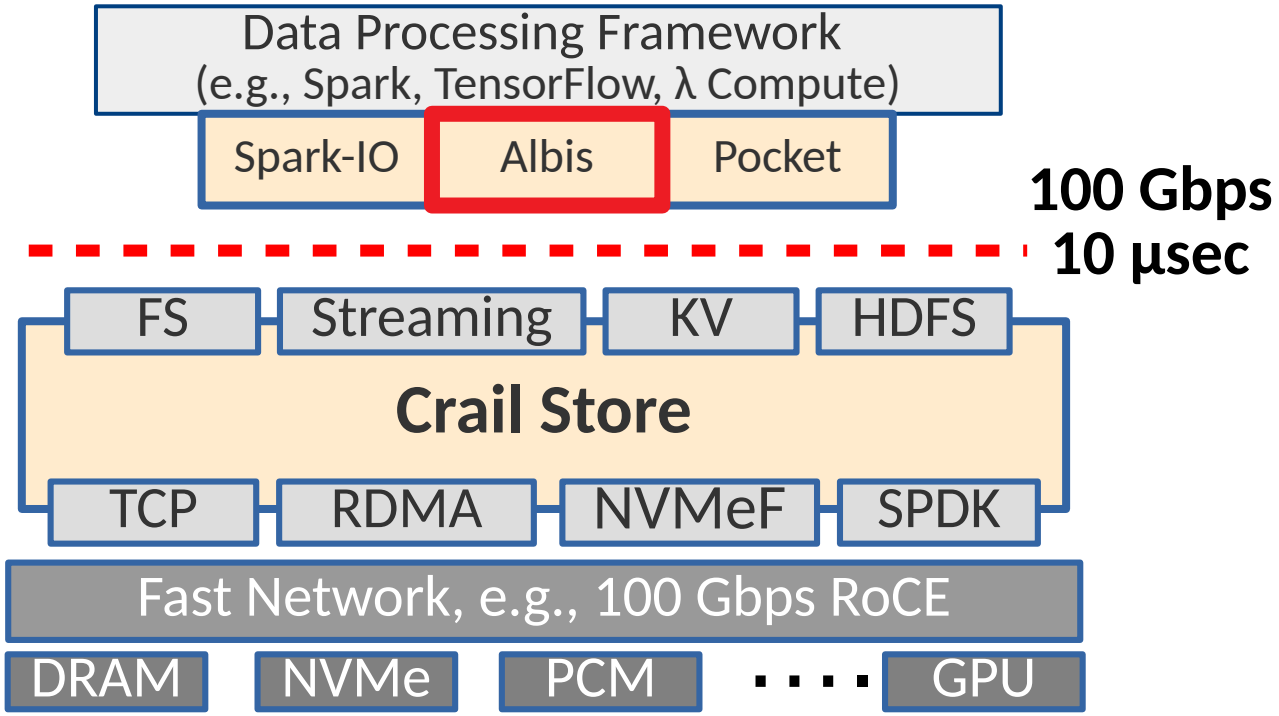
DRAM & Flash Disaggregation



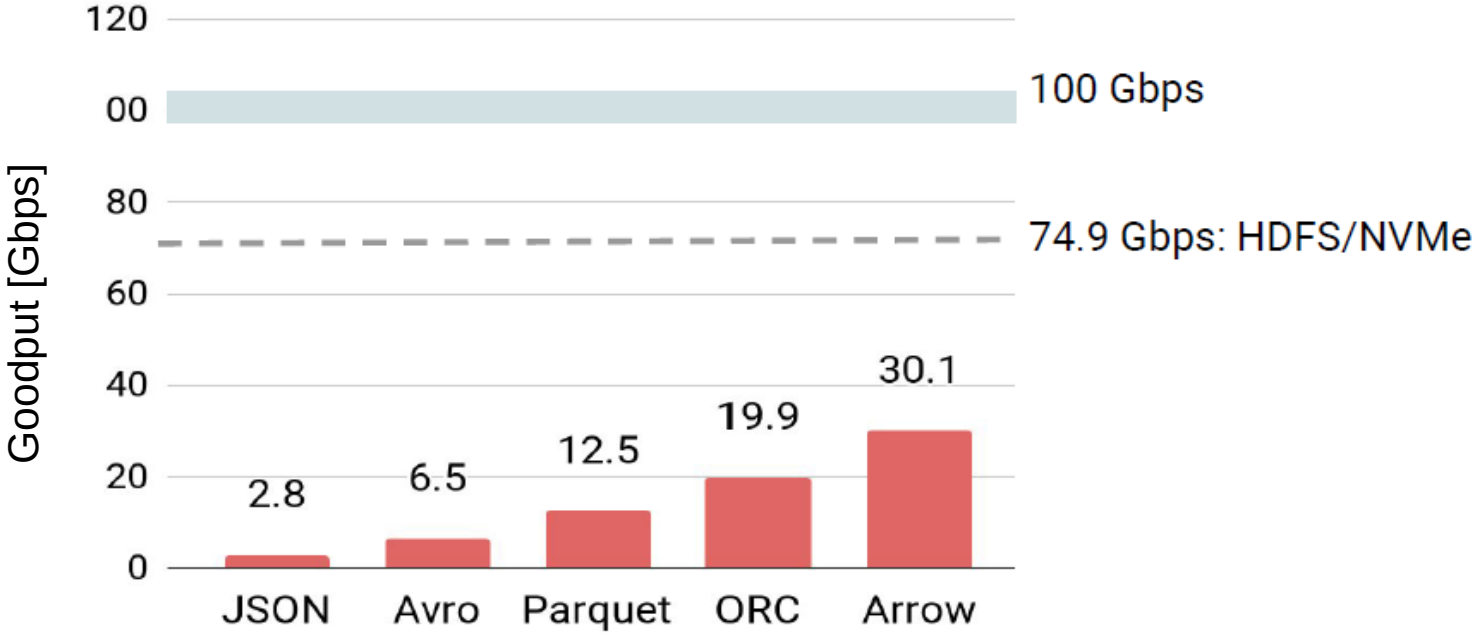


Using Crail, a Spark 200GB sorting workload can be run with memory and flash disaggregated at no extra cost

Running Workloads: SQL



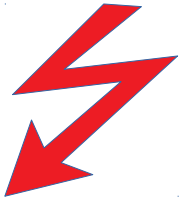
Reading Relational Data



None of the common file formats delivers a performance close to the hardware speed

Revisiting Design Principles

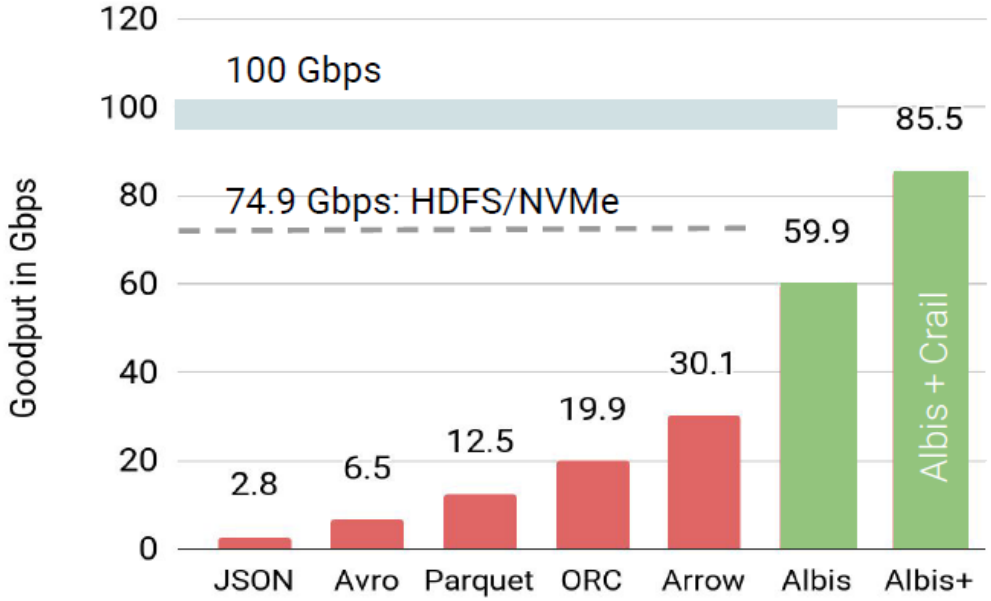
- Traditional Assumption: CPU is fast, I/O is slow
  - Use compression, encoding, etc.
  - Pack data and metadata together
  - Avoid metadata lookups



Mismatch in case of fast I/O hardware!

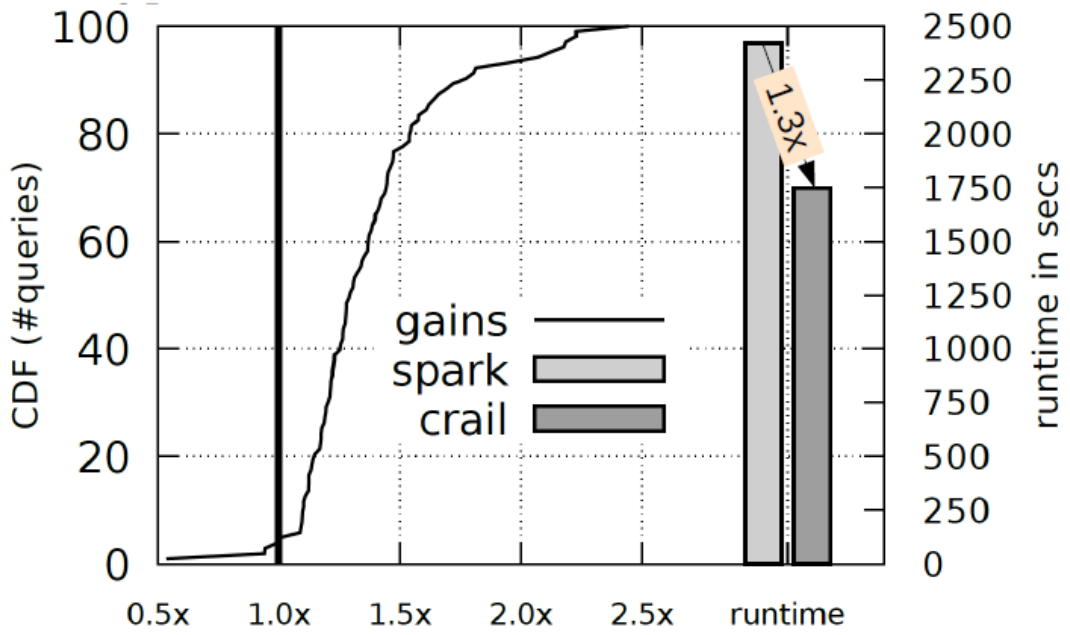
- Albis: new file format designed for fast I/O hardware
- Albis design principles
  - Avoid CPU pressure, i.e., no compression, encoding, etc.
  - Simple metadata management

Reading Relational with Albis



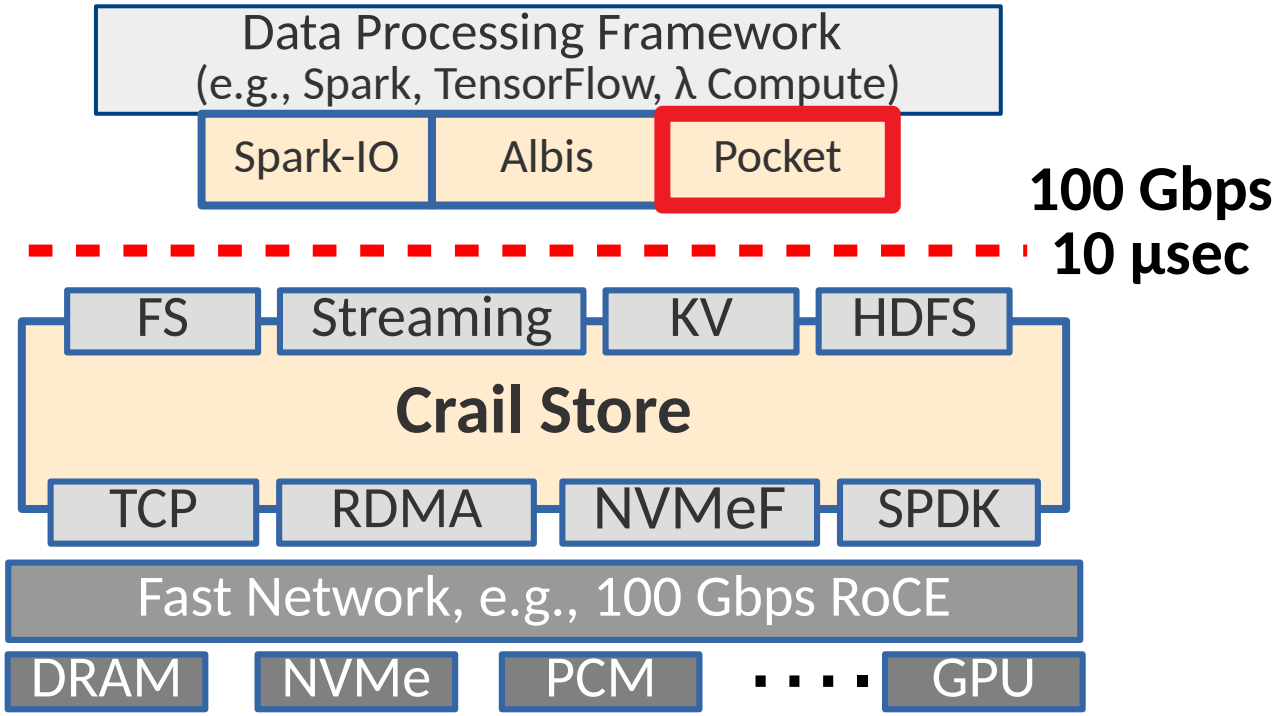
Albis/Crail delivers 2-30x performance improvements over other formats

TPC-DS using Albis/Crail



Albis/Crail delivers up to 2.5x performance gains

Running Workloads: Serverless

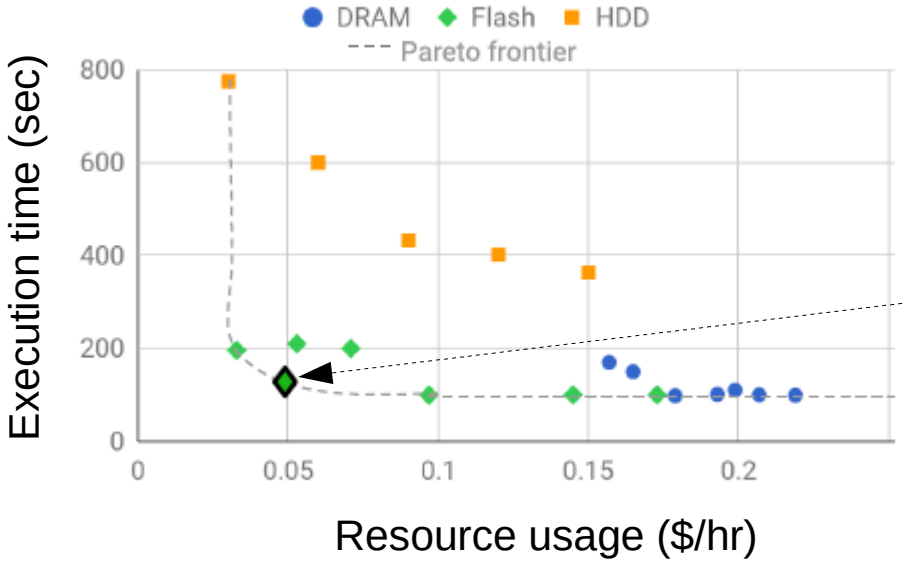


Serverless Computing

- Data sharing implemented using remote storage
  - Enables fast and fine-grained scaling
- Problem: existing storage platforms not suitable
  - Slow (e.g., S3)
  - No dynamic scaling (e.g. Redis)
  - Designed for either small or large data sets
- Can we use Crail? Not as is.
  - Most clouds don’t support RDMA, NVMf, etc.
  - Lacks automatic & elastic resource management

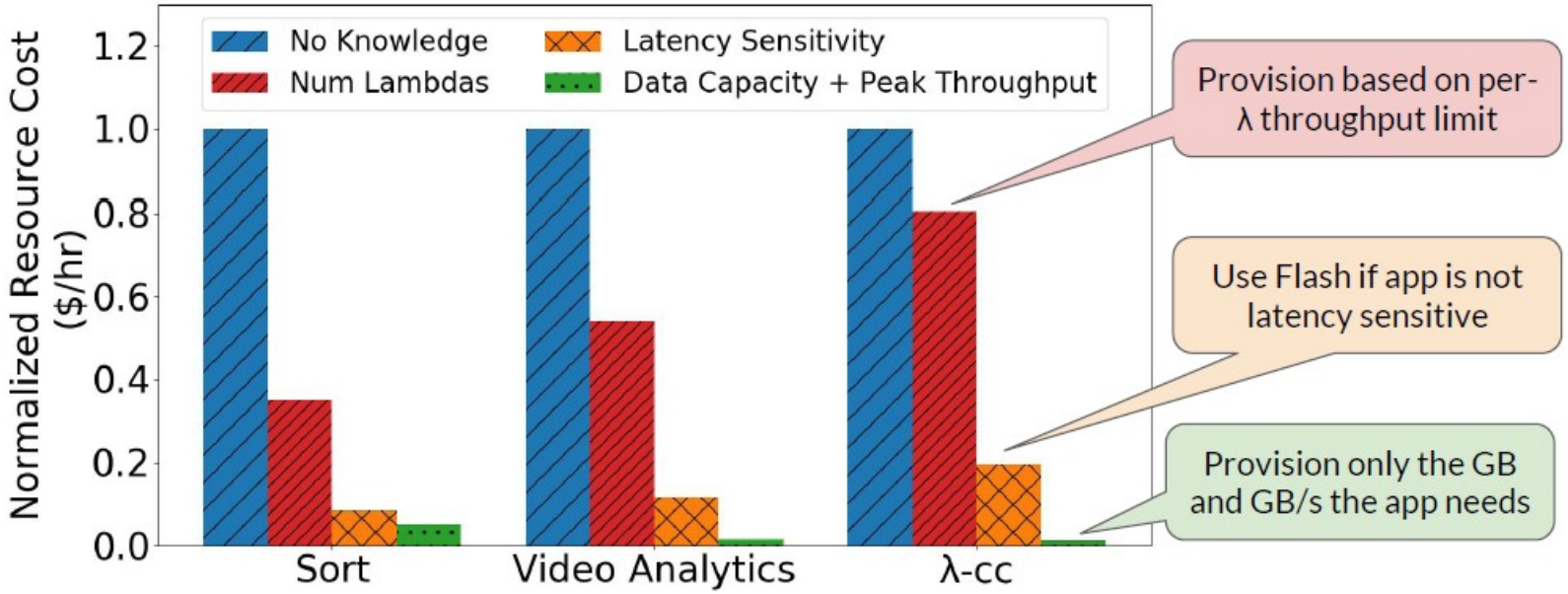
Pocket

- An elastic distributed data store for ephemeral data sharing in serverless analytics



Pocket dynamically rightsizes storage resources (nodes, media) in an attempt to find a spot with a good performance price ratio

Pocket: Resource Utilization



Pocket cost-effectively allocates resources based on user/framework hints

Conclusions

- Effectively using high-performance I/O hardware for data processing is challenging
- Crail is an attempt to re-think how data processing systems should interact with network and storage hardware
  - User-level I/O
  - Storage disaggregation
  - Memory/flash convergence
  - Elastic resource provisioning

References

- Crail: A High-Performance I/O Architecture for Distributed Data Processing, **IEEE Data Bulletin 2017**
- Albis: High-Performance File-format for Big Data, **Usenix ATC'18**
- Navigating Storage for Serverless Computing, **Usenix ATC'18**
- Pocket: Ephemeral Storage for Serverless Analytics, **OSDI'18** (to appear)
- Running Apache Spark on a High-Performance Cluster Using RDMA and NVMe Flash, **Spark Summit'17**
- Serverless Machine Learning using Crail, **Spark Summit'18**
- Apache Crail, http://crail.apache.org

Contributors

Animesh Trivedi, Jonas Pfefferle, Bernard Metzler, Adrian Schuepbach, Ana Klimovic, Yawen Wang, Michael Kaufmann, Yuval Degani, ...