

Running **Spark** On a High- Performance Cluster Using **RDMA** and **NVMe Flash**

Patrick Stuedi
IBM Research

Diversity

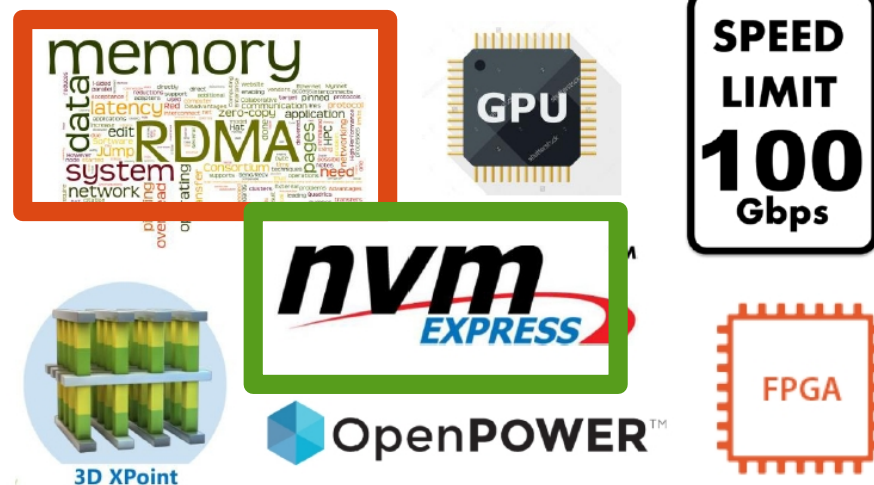
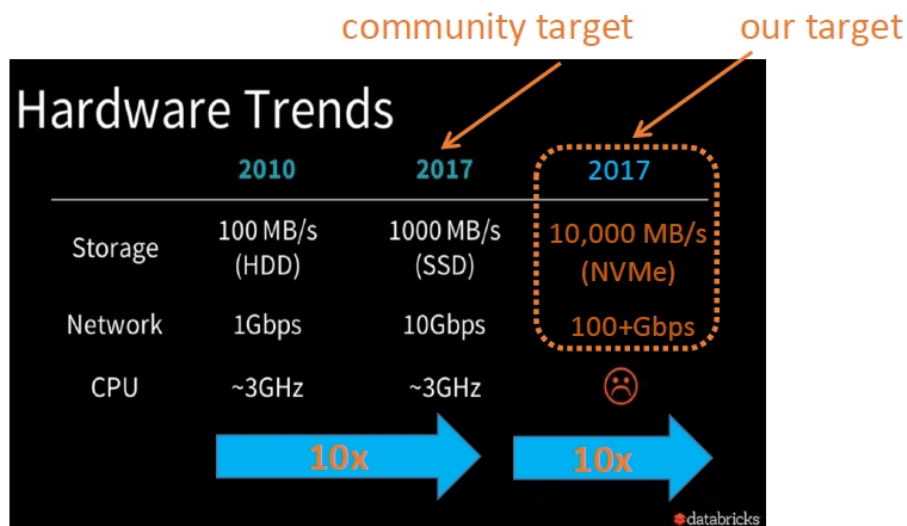


- 2

I/O Hardware Trends

Speed

Diversity



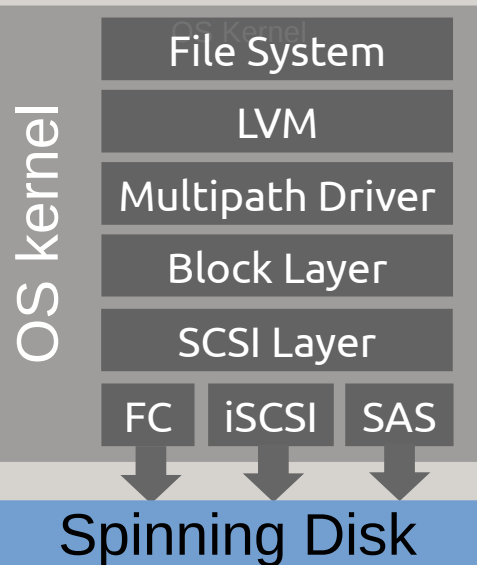
- **Network** interconnects have evolved from
 - Gbps bandwidth to 100Gbps
 - 100us delay to 1us delay
- **Storage** technology has evolved
 - Factor 100x-1000x

I/O API Trends

Traditional

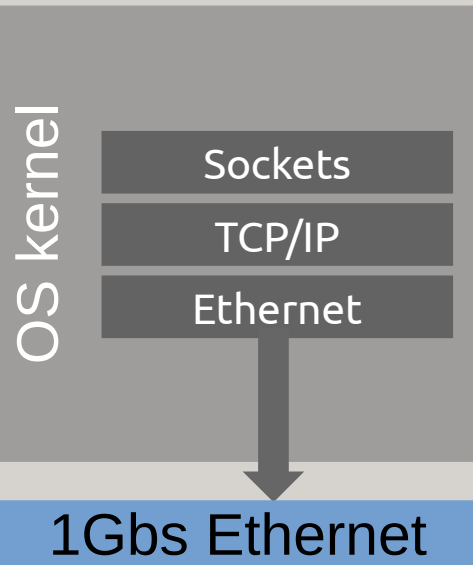
Storage

Runtime Application



Network

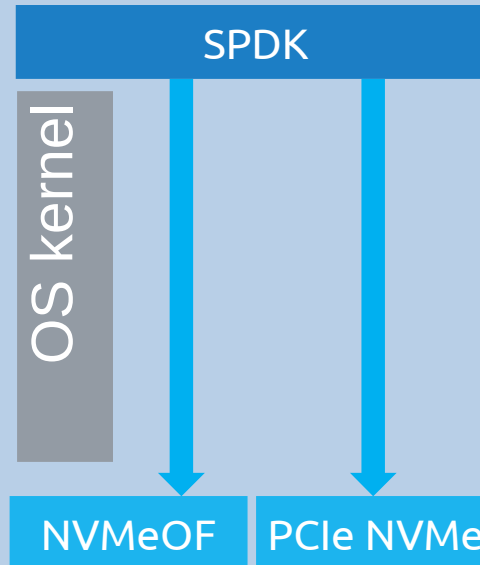
Runtime Application



Modern

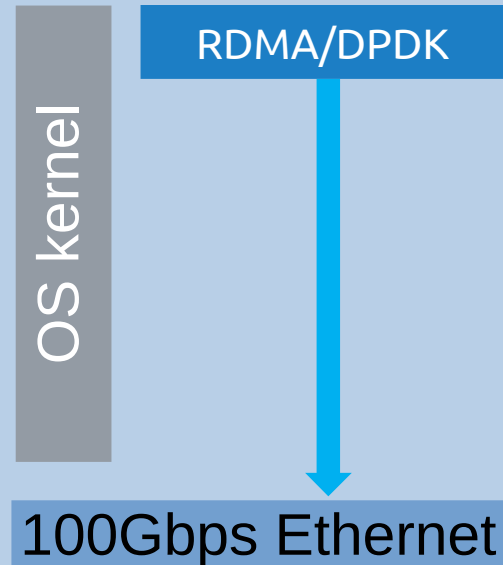
Storage

Runtime Application



Network

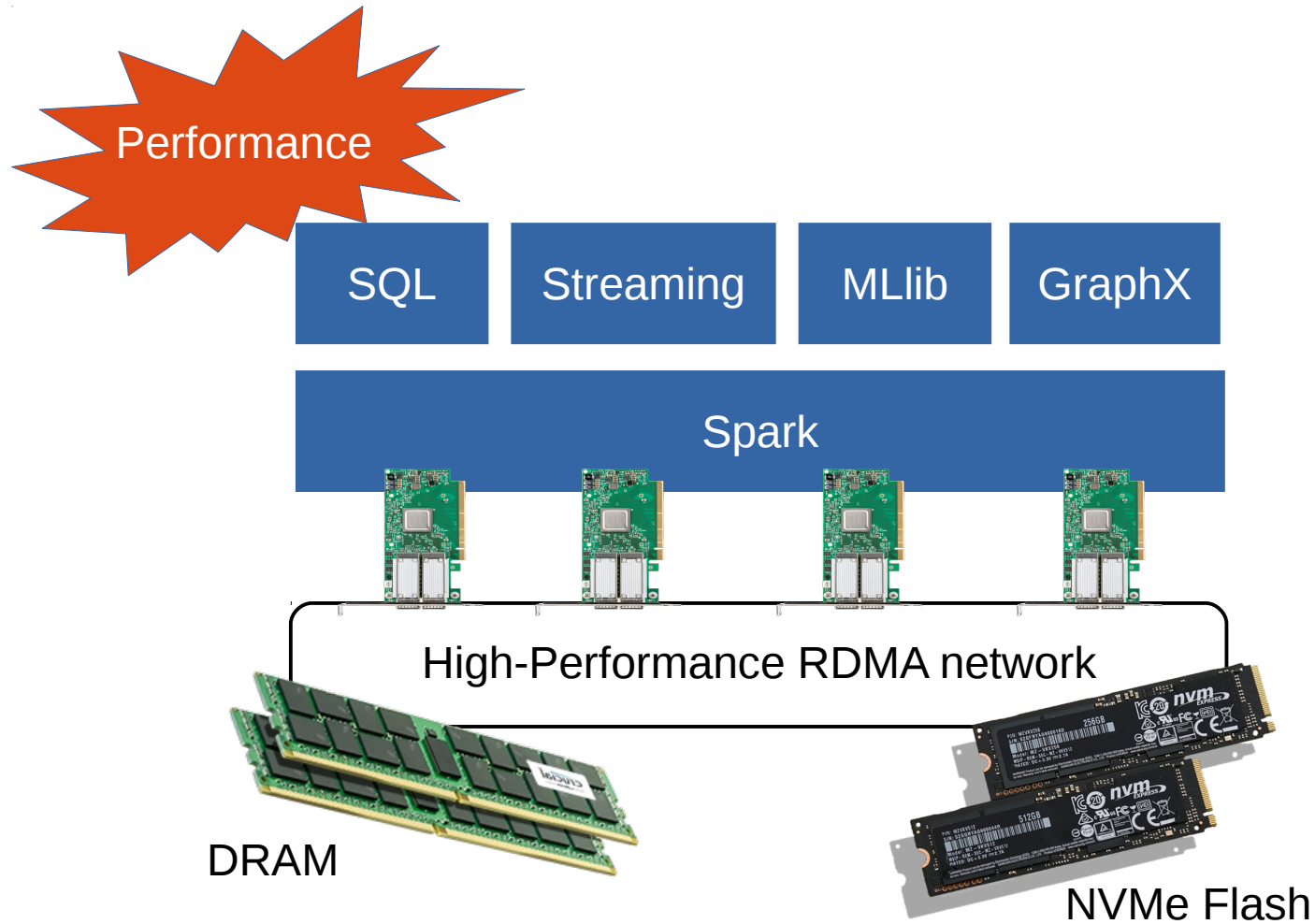
Runtime Application



Modern APIs for Networking and Storage offer asynchronous non-blocking user-level access to hardware

RDMA Example

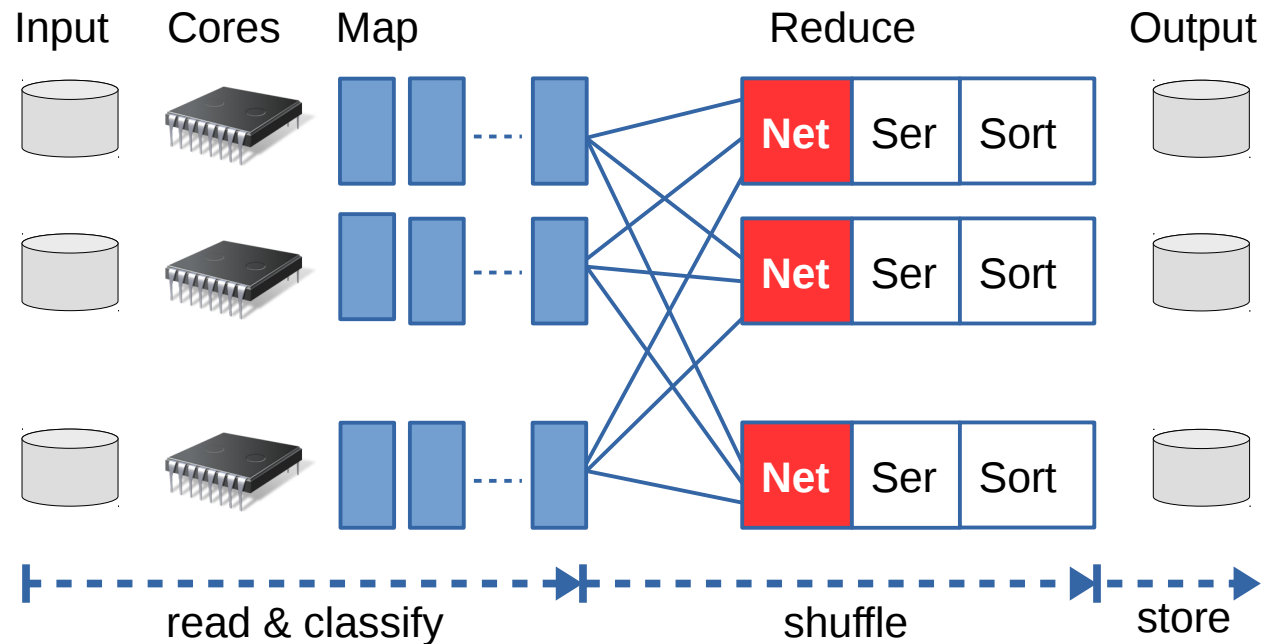
Let's Use it!



Case Study: Sorting in Spark

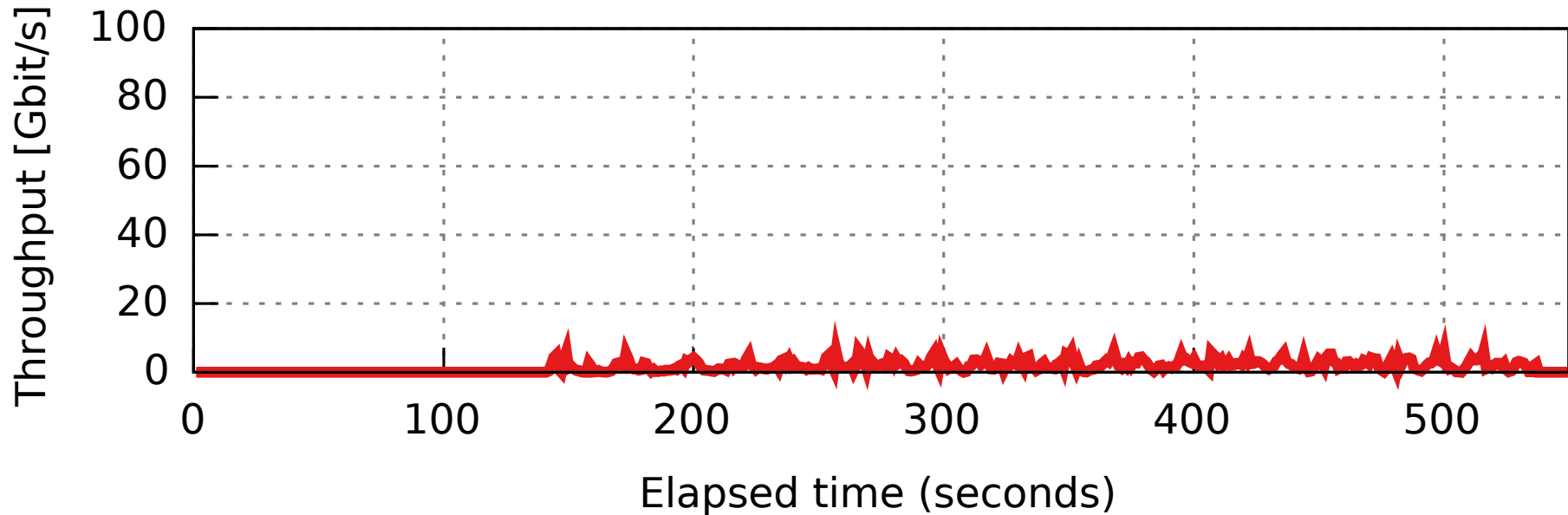
- Workload
 - Sort 12.8 TB of data on a 128 node cluster
- Cluster Hardware
 - DRAM: 512GB DDR4
 - Storage: 4x 1.2TB NVMe SSD
 - Network: 100GbE Mellanox RDMA
- Software
 - Ubuntu 16.04 with Linux kernel
 - Spark 2.0.0

Anatomy of Sorting in Spark



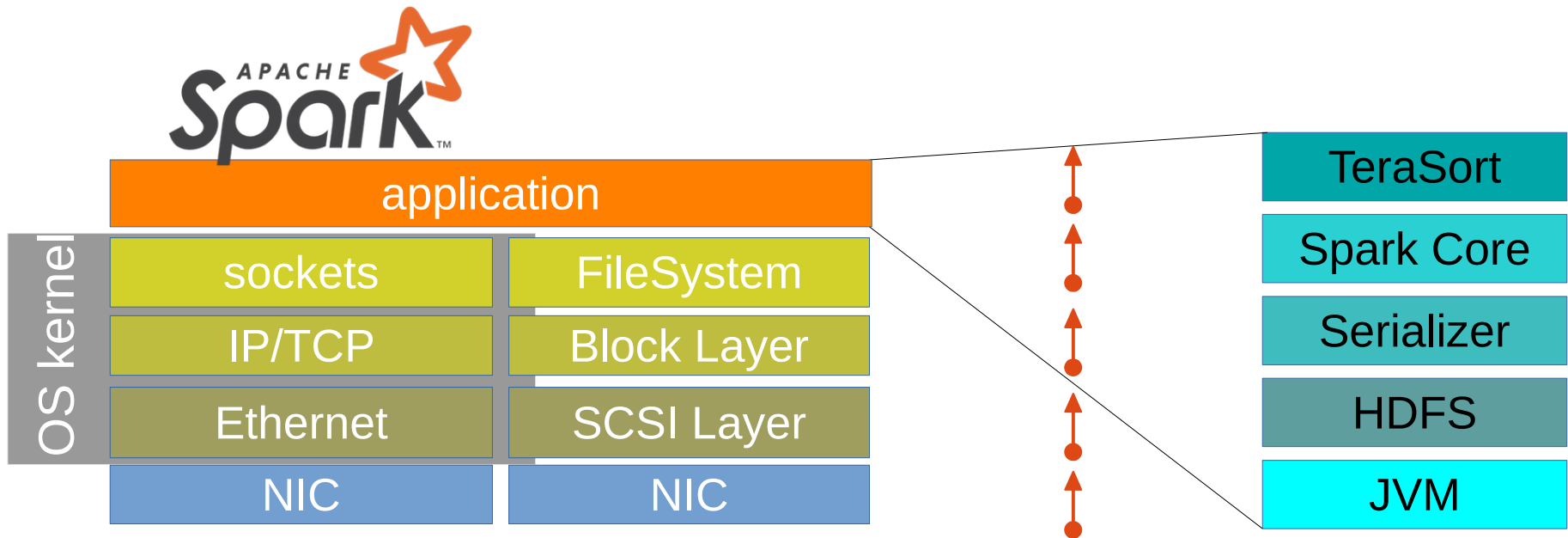
- Map task classify data into local files (typically absorbed by buffer cache)
- Reduce task fetch remote files over the network
- Sorting requires the entire data set to be shuffled over the network

How is the Network Used?



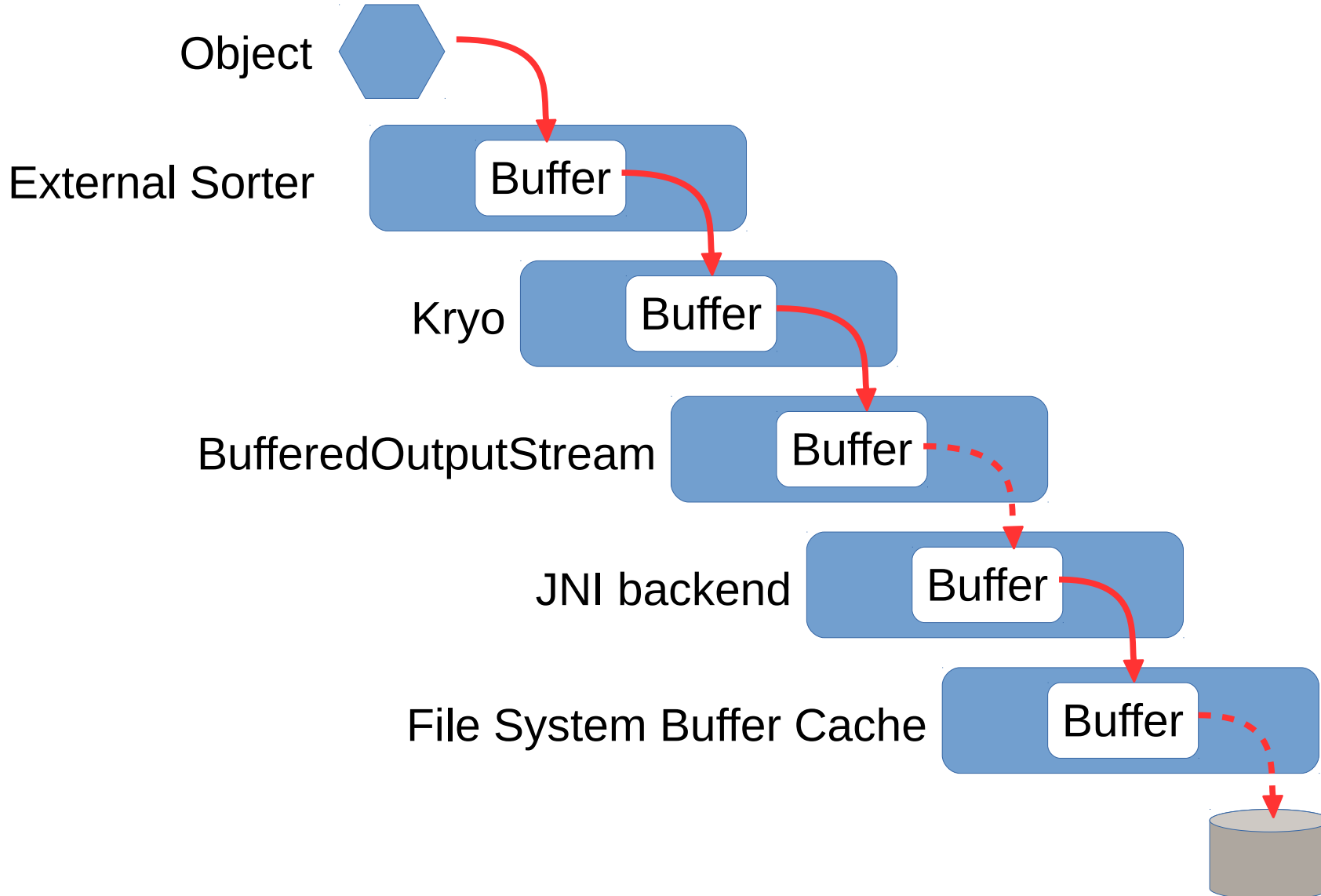
- Only 5-10 Gpbs of the network is being used

What is the Problem

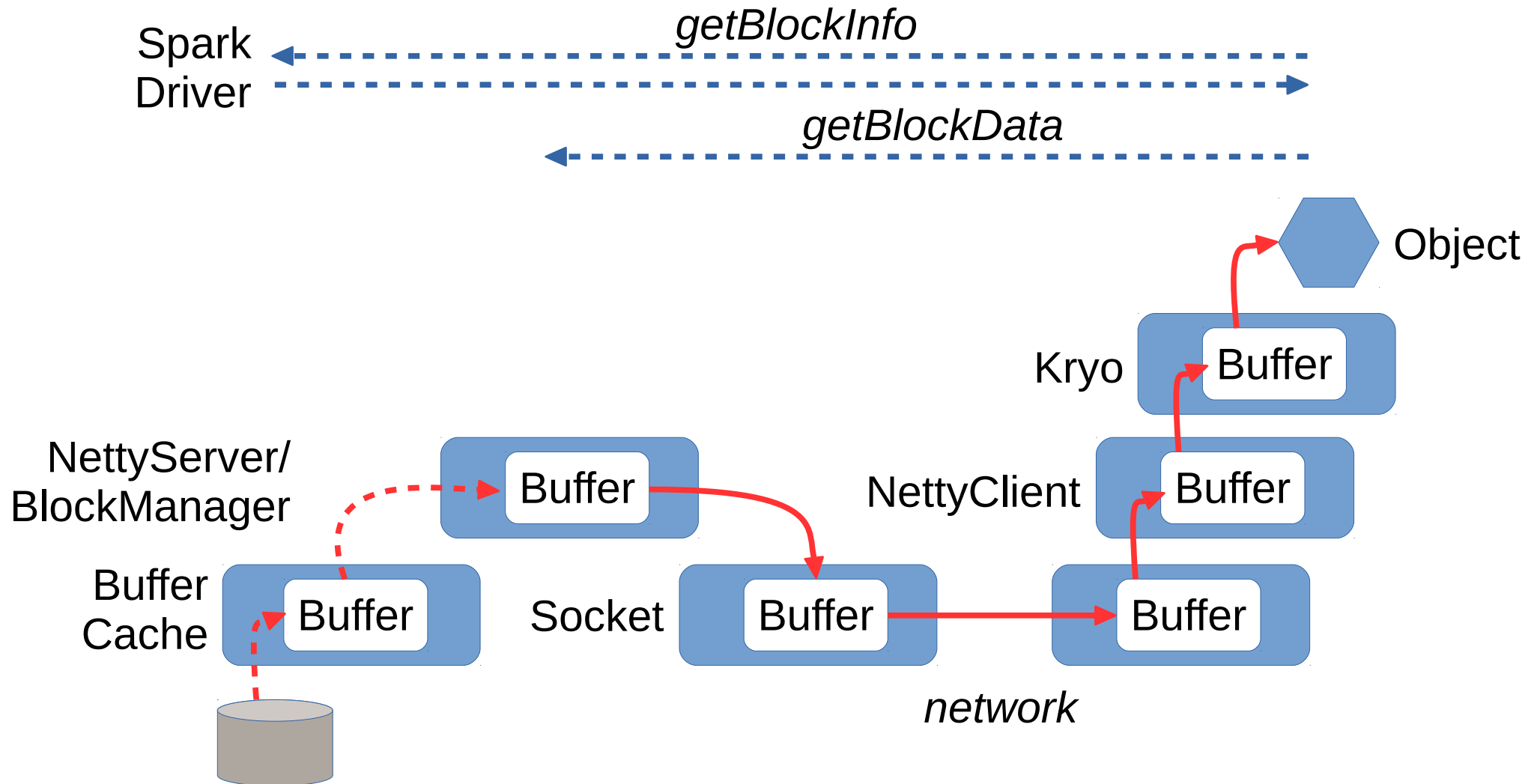


- Application use the legacy APIs
- Applications themselves are heavily layered!
- Overhead during local file system writing
- Overhead during network processing
 - Data copies, context switches, cache pollution, etc

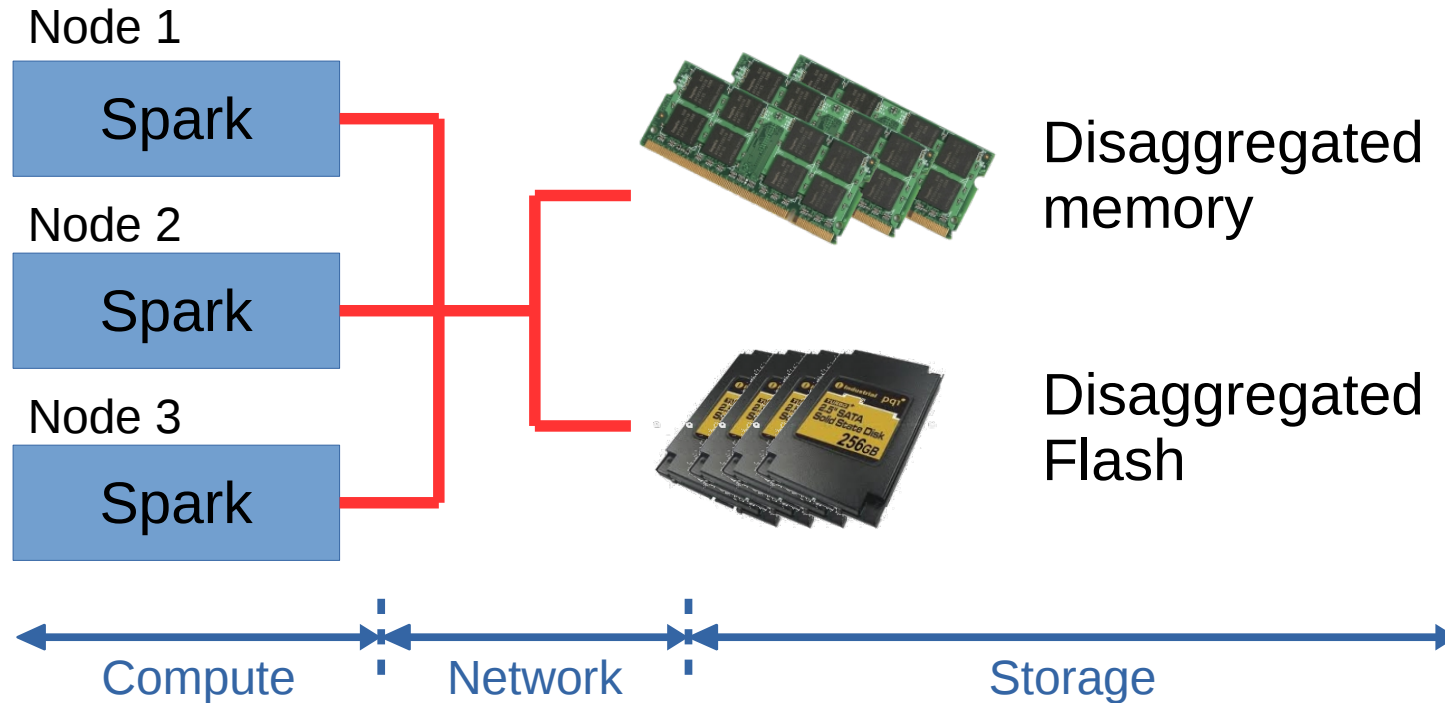
Example: Shuffle Writer (map)



Example: Shuffle Reader (reduce)



Other Challenges

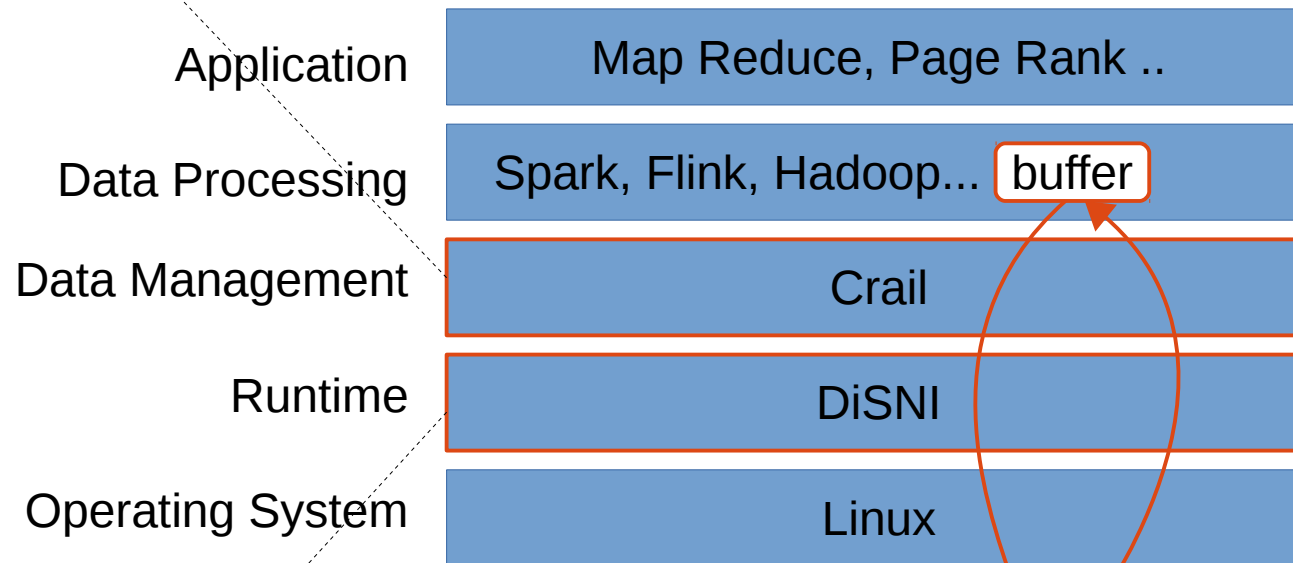


- Can't easily exploit new I/O properties, e.g., local \approx remote
 - Difficult to disaggregate memory and storage
 - Difficult to leverage storage hierarchies

The Crail Project: Performance View

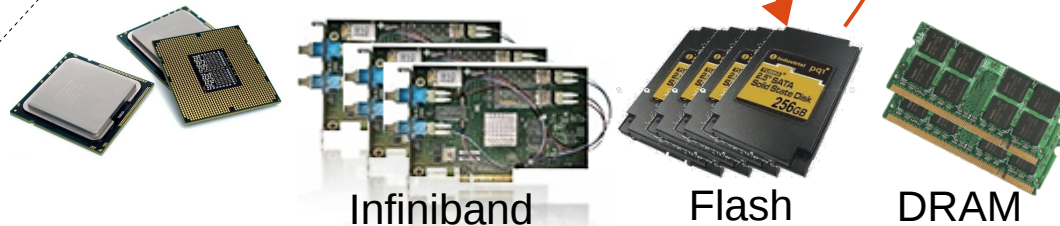
Fast distributed
storage for
temporary data

data is exchanged directly between Spark and
I/O devices (network & storage)



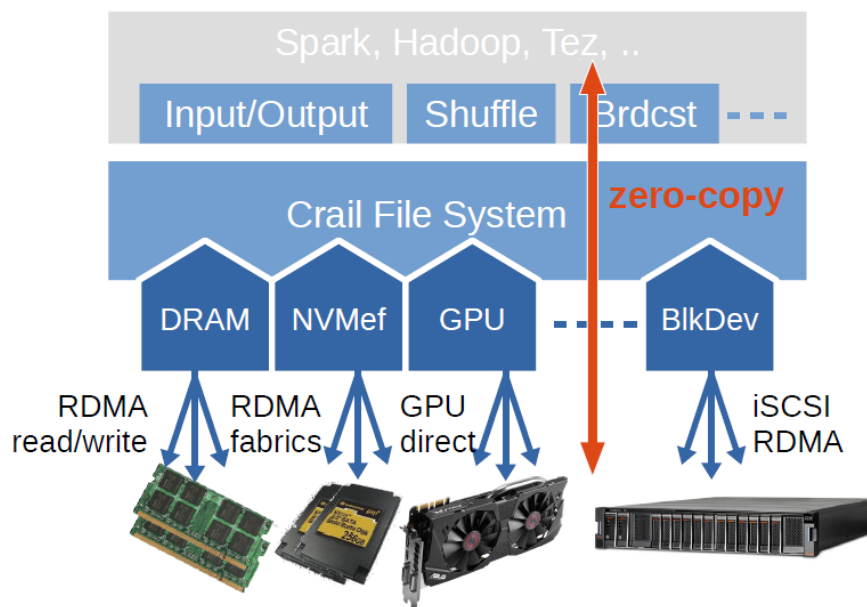
No data copies,
no context switches,
no cache pollution,
very thin callstack

Fast Hardware



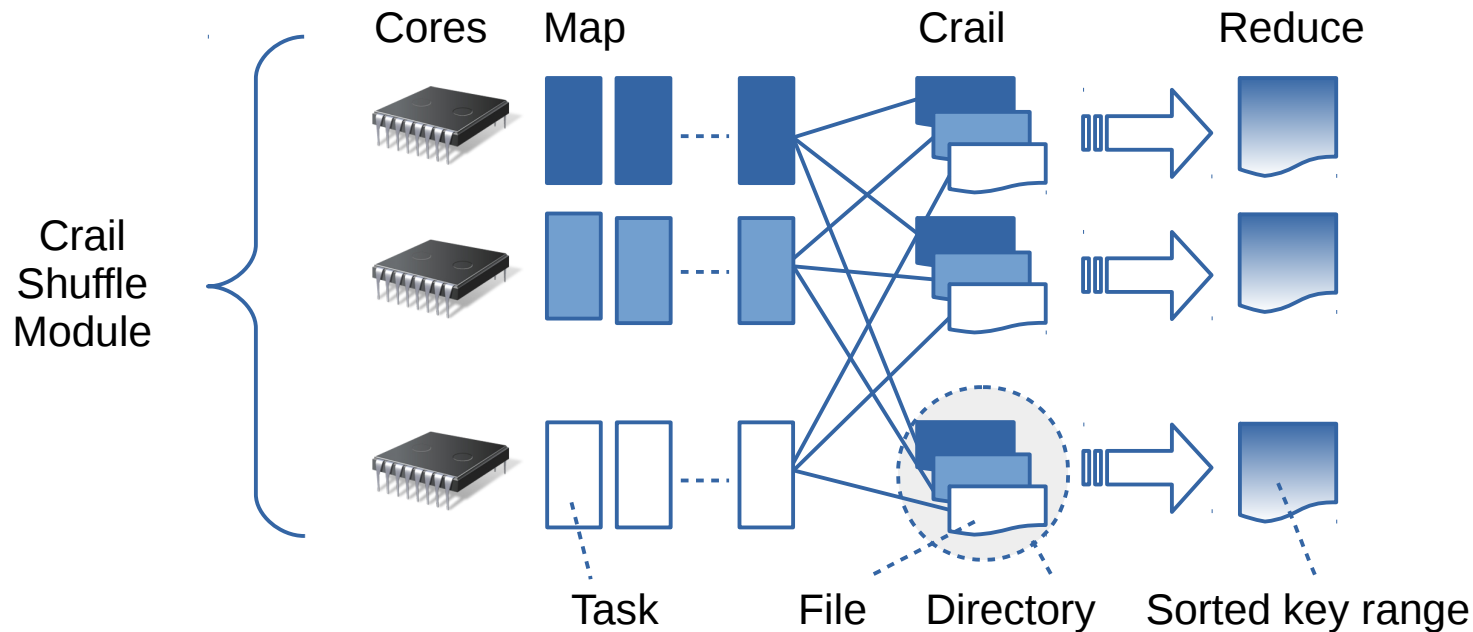
JVM I/O
bypass

The Crail Project: Diversity View



- Contains three types of components:
 - Crail Modules: implement higher-level I/O operations
 - Crail File System: backbone for any data movement
 - Crail storage backends: storage/network specific binding
- Crail's main features
 - Designed for high-performance networking and storage hardware
 - Designed explicitly for user-level storage and networking APIs
 - Designed to be completely pluggable (modules) and extendible (storage)

Crail Plugin Modules



Shuffle

- Spark specific plugin
- Maps key ranges to Crail dir's
- Selects storage affinity in order of best performance

Broadcast

- Spark specific plugin
- Stores broadcast variables as Crail files

HDFS Adaptor

- Generic plugin
- Exports HDFS API

Evaluation – Terasort

128 nodes OpenPOWER cluster

- 2 x IBM POWER8 10-core @ 2.9 GHz
- DRAM: 512GB DDR4
- 4 x 1.2 TB NVMe SSD
- 100GbE Mellanox ConnectX-4 EN (RoCE)
- Ubuntu 16.04 (kernel 4.4.0-31)
- Spark 2.0.2

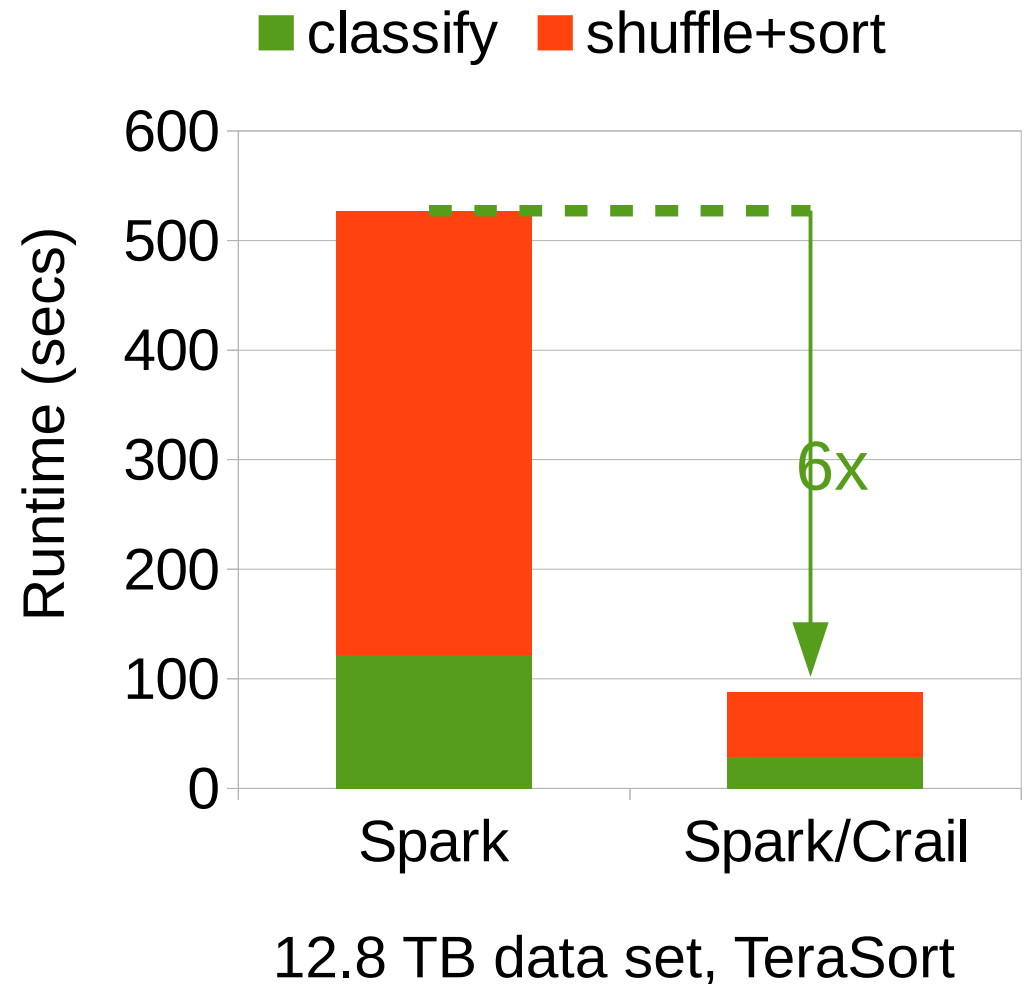
Evaluation – Terasort

128 nodes OpenPOWER cluster

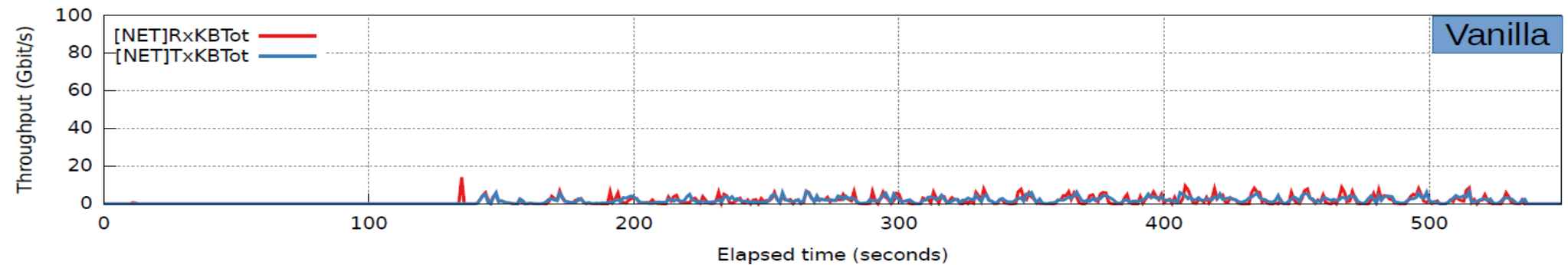
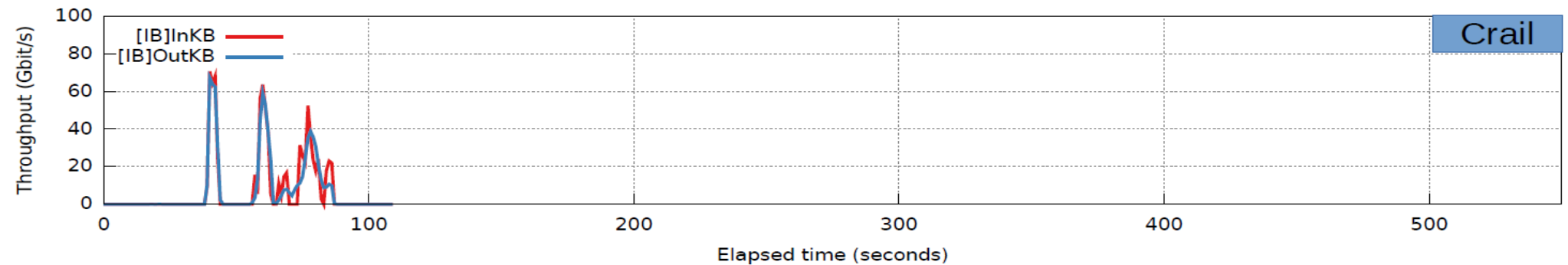
- 2 x IBM POWER8 10-core @ 2.9 GHz
- DRAM: 512GB DDR4
- 4 x 1.2 TB NVMe SSD
- 100GbE Mellanox ConnectX-4 EN (RoCE)
- Ubuntu 16.04 (kernel 4.4.0-31)
- Spark 2.0.2

Performance gain: 6x

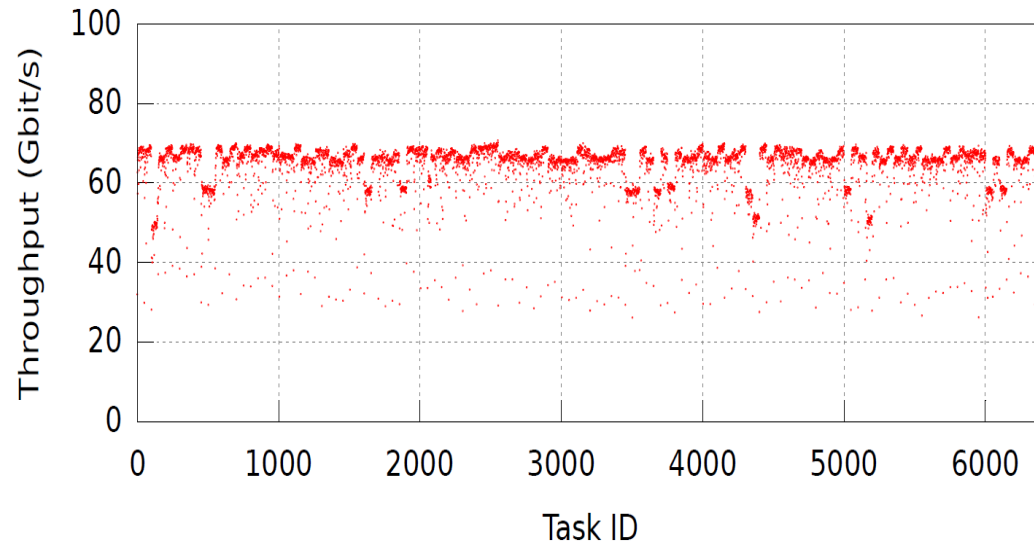
- Most gain from reduce phase:
 - Crail shuffler much faster than Spark build-in
 - Dramatically reduced CPU involvement
 - Dramatically improved network usage
- Map phase: all activity local
 - Still faster than vanilla Spark



Evaluation – Network IO



- Vanilla Spark runs on 100GbE
- Spark/Crail runs on 100Gb RoCE/RDMA
- Vanilla Spark peaks at ~10Gb/s
- Spark/Crail shuffle delivers ~70Gb/s



Sorting Comparison

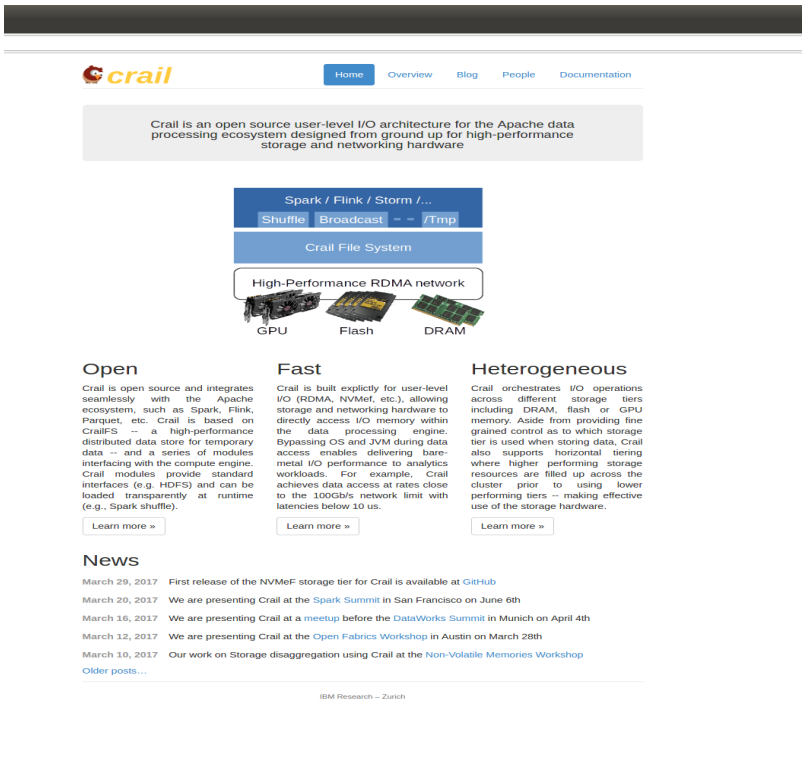
| | Spark + Crail | Spark 2.0.2 | Winner 2014 | Winner 2016 |
|---------------------|------------------|----------------|----------------|----------------|
| Size TB | 12.8 | | 100 | |
| Time sec | 98 | 527 | 1406 | 98.6 |
| Cores | 2560 | | 6592 | 10240 |
| Nodes | 128 | | 206 | 512 |
| NW Gb/s | 100 | | 10 | 100 |
| Rate TB/min | 7.8 | 1.4 | 4.27 | 44.78 |
| Rate/core GB/min | 3.13 | 0.58 | 0.66 | 4.4 |

- Spark/Crail CPU efficiency is close to 2016 sorting benchmark winner: **3.13 vs. 4.4 GB/min/core**
- 2016 winner runs native C code!

Crail is Open Source!

www.crail.io

<https://github.com/zrlio>



The screenshot shows the Crail website. At the top is a navigation bar with links: Home, Overview, Blog, People, and Documentation. Below the navigation bar is a hero section with the text: "Crail is an open source user-level I/O architecture for the Apache data processing ecosystem designed from ground up for high-performance storage and networking hardware". Below this is a diagram showing the architecture: "Spark / Flink / Storm / ..." at the top, followed by "Shuffle Broadcast ... /Tmp", then "Crail File System", and finally "High-Performance RDMA network" which connects to "GPU", "Flash", and "DRAM". Below the diagram are three columns: "Open", "Fast", and "Heterogeneous", each with a brief description of Crail's capabilities and a "Learn more »" button. At the bottom is a "News" section with a list of recent updates and a "Older posts..." link.

Crail is an open source user-level I/O architecture for the Apache data processing ecosystem designed from ground up for high-performance storage and networking hardware

Spark / Flink / Storm / ...
Shuffle Broadcast ... /Tmp
Crail File System
High-Performance RDMA network
GPU Flash DRAM

Open

Crail is open source and integrates seamlessly with the Apache ecosystem, such as Spark, Flink, Parquet, etc. Crail is based on CrailFS -- a high-performance distributed data store for temporary data -- and a series of modules interfacing with the compute engine. Crail modules provide standard interfaces (e.g. HDFS) and can be loaded transparently at runtime (e.g., Spark shuffle).

[Learn more »](#)

Fast

Crail is built explicitly for user-level I/O (RDMA, NVMe, etc.), allowing storage and networking hardware to directly access I/O memory within the data processing engine. Bypassing OS and JVM during data access enables delivering bare-metal I/O performance to analytics workloads. For example, Crail achieves data access at rates close to the 100Gb/s network limit with latencies below 10 us.

[Learn more »](#)

Heterogeneous

Crail orchestrates I/O operations across different storage tiers including DRAM, flash or GPU memory. Aside from providing fine-grained control as to which storage tier is used when storing data, Crail also supports horizontal tiering where higher performing storage resources are filled up across the cluster prior to using lower performing tiers -- making effective use of the storage hardware.

[Learn more »](#)

News

March 29, 2017 First release of the NVMeF storage tier for Crail is available at [GitHub](#)

March 20, 2017 We are presenting Crail at the [Spark Summit](#) in San Francisco on June 6th

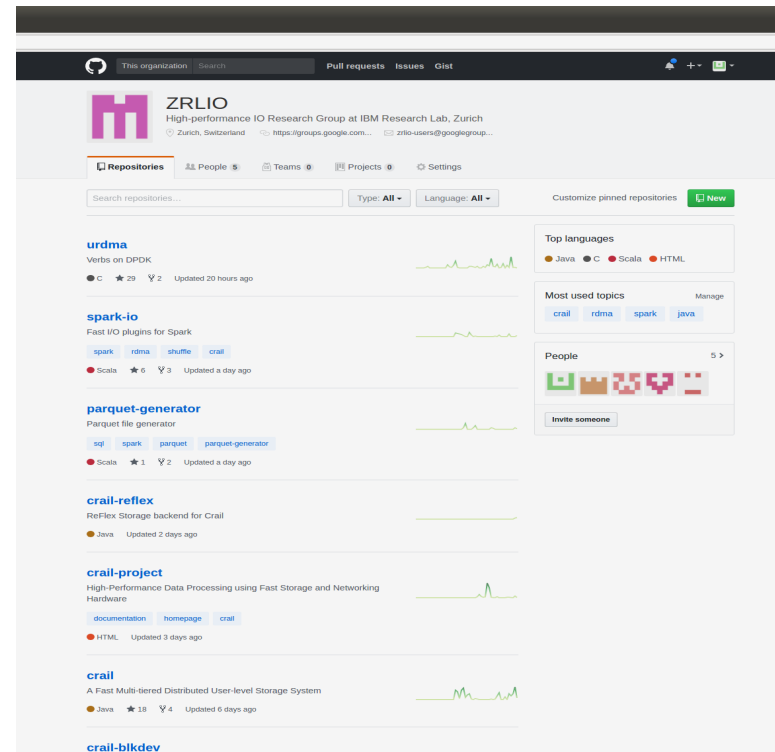
March 16, 2017 We are presenting Crail at a [meetup](#) before the [DataWorks Summit](#) in Munich on April 4th

March 12, 2017 We are presenting Crail at the [Open Fabrics Workshop](#) in Austin on March 28th

March 10, 2017 Our work on Storage disaggregation using Crail at the [Non-Volatile Memories Workshop](#)

[Older posts...](#)

IBM Research - Zurich



The screenshot shows the Zrlio GitHub organization page. At the top is a navigation bar with links: This organization, Search, Pull requests, Issues, and Gist. Below the navigation bar is the Zrlio organization profile, which includes the name "ZRLIO", a description "High-performance IO Research Group at IBM Research Lab, Zurich", and location "Zurich, Switzerland". Below the profile is a list of repositories. The repositories listed are: "urdma" (Verbs on DPDK), "spark-io" (Fast I/O plugins for Spark), "parquet-generator" (Parquet file generator), "crail-reflex" (ReFlex Storage backend for Crail), "crail-project" (High-Performance Data Processing using Fast Storage and Networking Hardware), "crail" (A Fast Multi-tiered Distributed User-level Storage System), and "crail-blkdev". Each repository has a star count, a fork count, and a last updated date. On the right side of the repository list are two sidebars: "Top languages" showing Java, C, Scala, and HTML, and "Most used topics" showing crail, rdma, spark, and java. Below these is a "People" section with a list of team members and an "Invite someone" button.

This organization Search Pull requests Issues Gist

ZRLIO
High-performance IO Research Group at IBM Research Lab, Zurich
Zurich, Switzerland <https://groups.google.com...> zrlio-users@googlegroup...

Repositories All People Teams Projects Settings

Search repositories... Type: All Language: All Customize pinned repositories [New](#)

urdma
Verbs on DPDK
C 29 2 Updated 20 hours ago

spark-io
Fast I/O plugins for Spark
spark rdma shuffle crail
Scala 6 3 Updated a day ago

parquet-generator
Parquet file generator
sql spark parquet parquet-generator
Scala 1 2 Updated a day ago

crail-reflex
ReFlex Storage backend for Crail
Java Updated 2 days ago

crail-project
High-Performance Data Processing using Fast Storage and Networking Hardware
documentation homepage crail
HTML Updated 3 days ago

crail
A Fast Multi-tiered Distributed User-level Storage System
Java 18 4 Updated 6 days ago

crail-blkdev

Top languages
Java C Scala HTML

Most used topics
crail rdma spark java Manage

People 5
[Invite someone](#)

Related Work

Three classes of related work:

- New Data Processing Systems for High-Performance Network & Storage Hardware
 - FARM, RamCloud, HERD, etc
 - Fast, but mostly academic, proprietary interfaces
- Updates/patches to existing Systems
 - Ohio Spark/Hadoop Distro
 - Slow because no radical changes possible: fetrofitting RDMA/Flash integration into existing file/socket based I/O stacks
- Memory/Flash caches/stores
 - Example: Tacyon
 - Slow because not designed for high-performance hardware

Conclusion

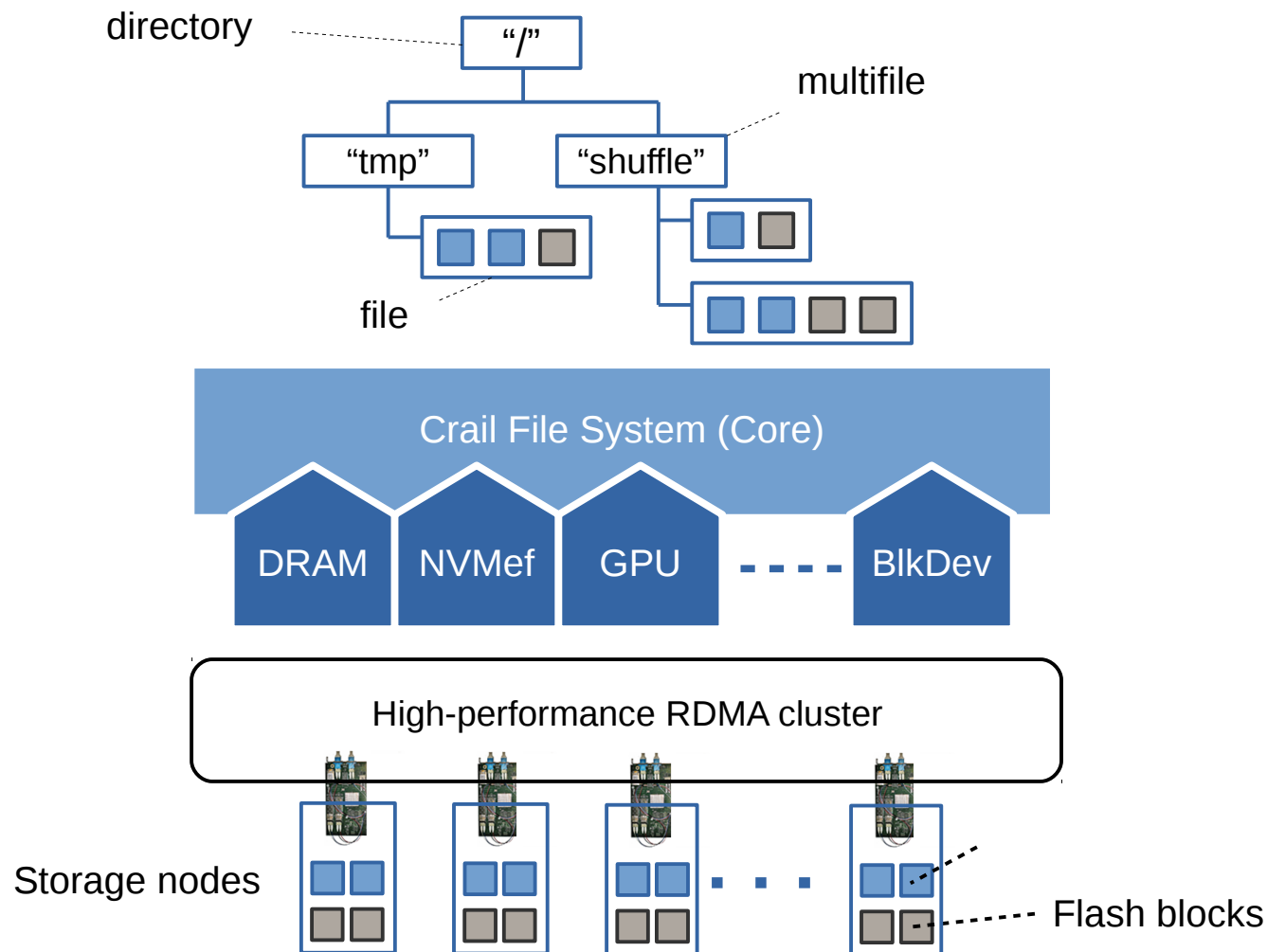
Today's open source analytics stacks:

- Existing analytics stacks designed for yesterday's commodity hardware
- Performance on high-end hardware inhibited by heavy-layered stack architecture

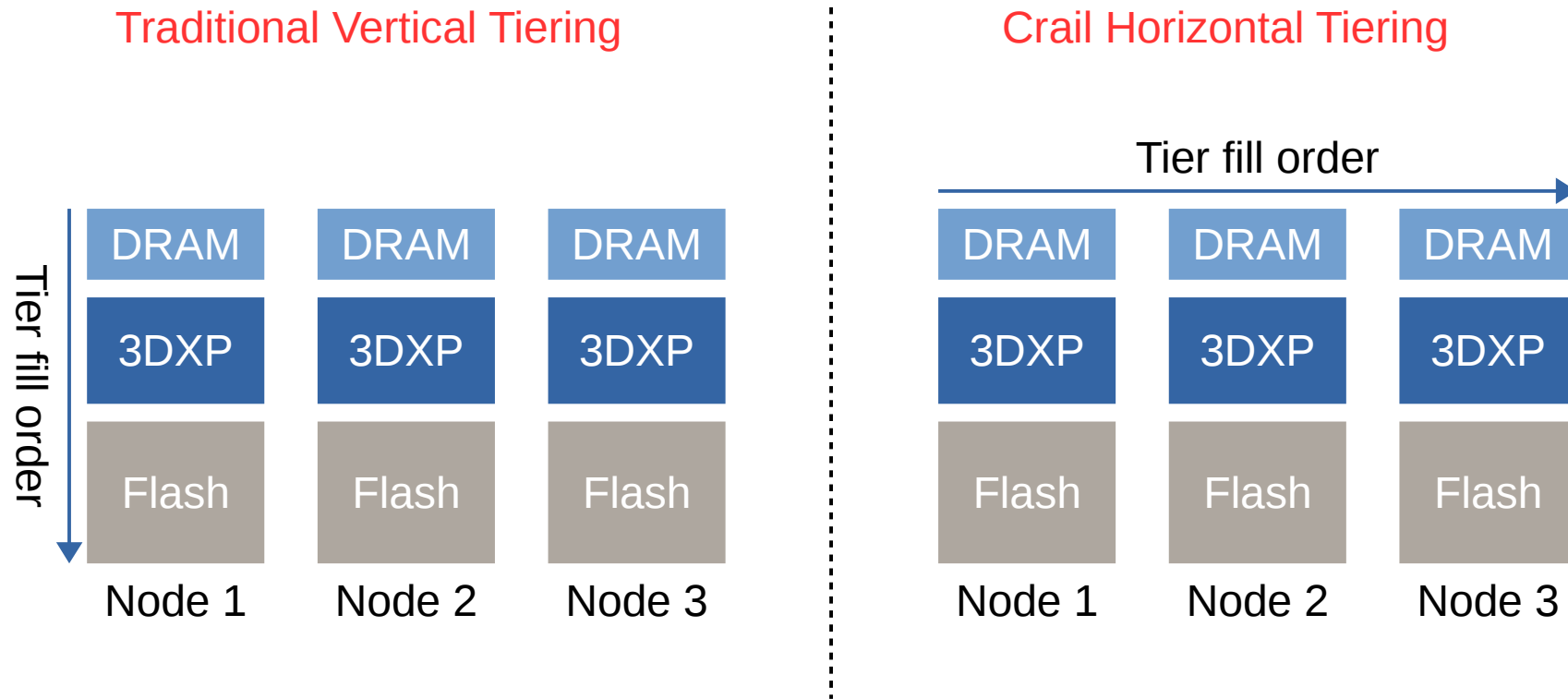
The Crail Approach:

- Radical re-design of I/O (network & storage) for analytics by exploiting modern hardware
 - RDMA, NVMe & NVMe over fabrics
- Enable high-performance disaggregated storage for analytics
- Extend Spark operation to take advantage of Crail
- Crail is open source: www.crail.io

The Crail Store



Crail Storage Tiering



With horizontal tiering, higher-performing tiers are filled up across the cluster prior to using lower performing tiers