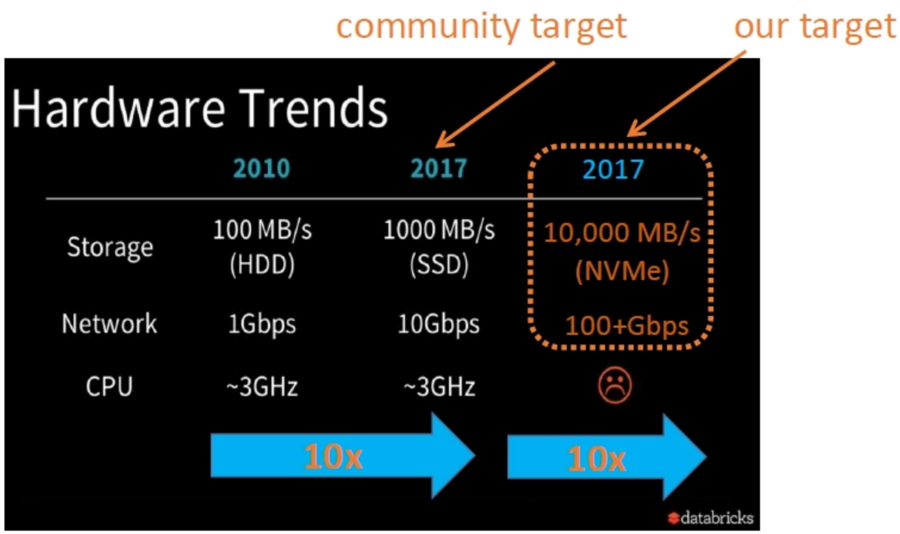
# Running Spark On a High-Performance Cluster Using RDMA and NVMe Flash

Patrick Stuedi
IBM Research

# I/O Hardware Trends

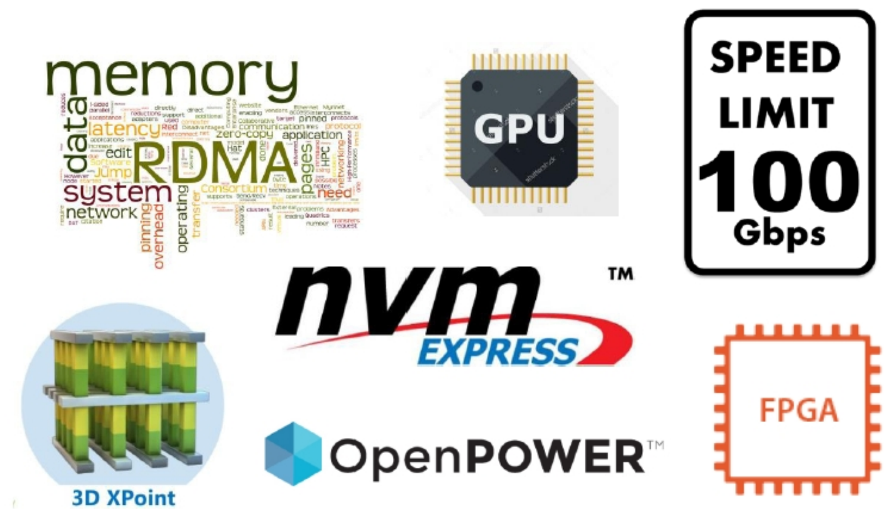
Speed




| | 2010 | 2017 | 2017 |
|---|---|---|---|
| Storage | 100 MB/s (HDD) | 1000 MB/s (SSD) | 10,000 MB/s (NVMe) |
| Network | 1Gbps | 10Gbps | 100+Gbps |
| CPU | ~3GHz | ~3GHz | ☹ |
| | 10x | | 10x |

databricks

Diversity

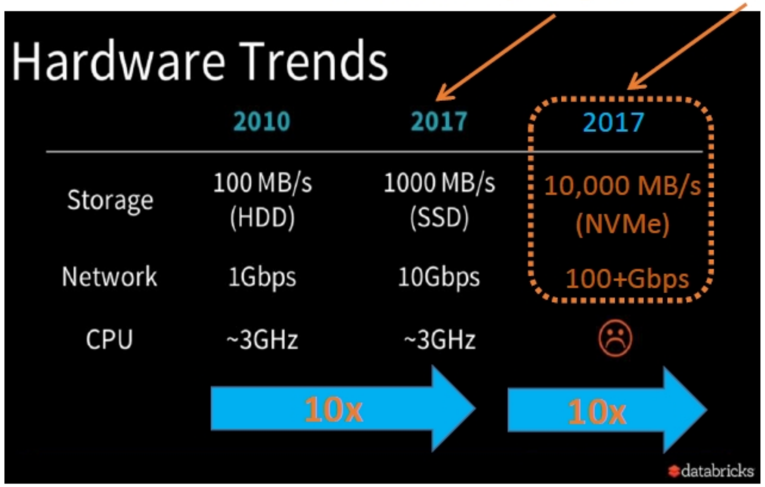



- Network interconnects have evolved from
  - Gbps bandwidth to 100Gbps
  - 100us delay to 1us delay
- Storage technology has evolved
  - Factor 100x-1000x

# I/O Hardware Trends

Speed

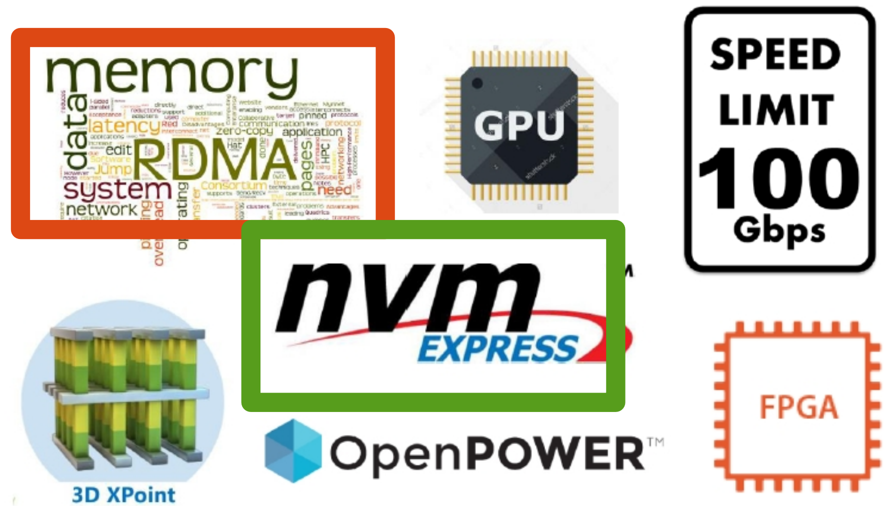
community target    our target


Hardware Trends

| | 2010 | 2017 | 2017 |
|---|---|---|---|
| Storage | 100 MB/s (HDD) | 1000 MB/s (SSD) | 10,000 MB/s (NVMe) |
| Network | 1Gbps | 10Gbps | 100+Gbps |
| CPU | ~3GHz | ~3GHz | ☹ |
| | 10x | | 10x |

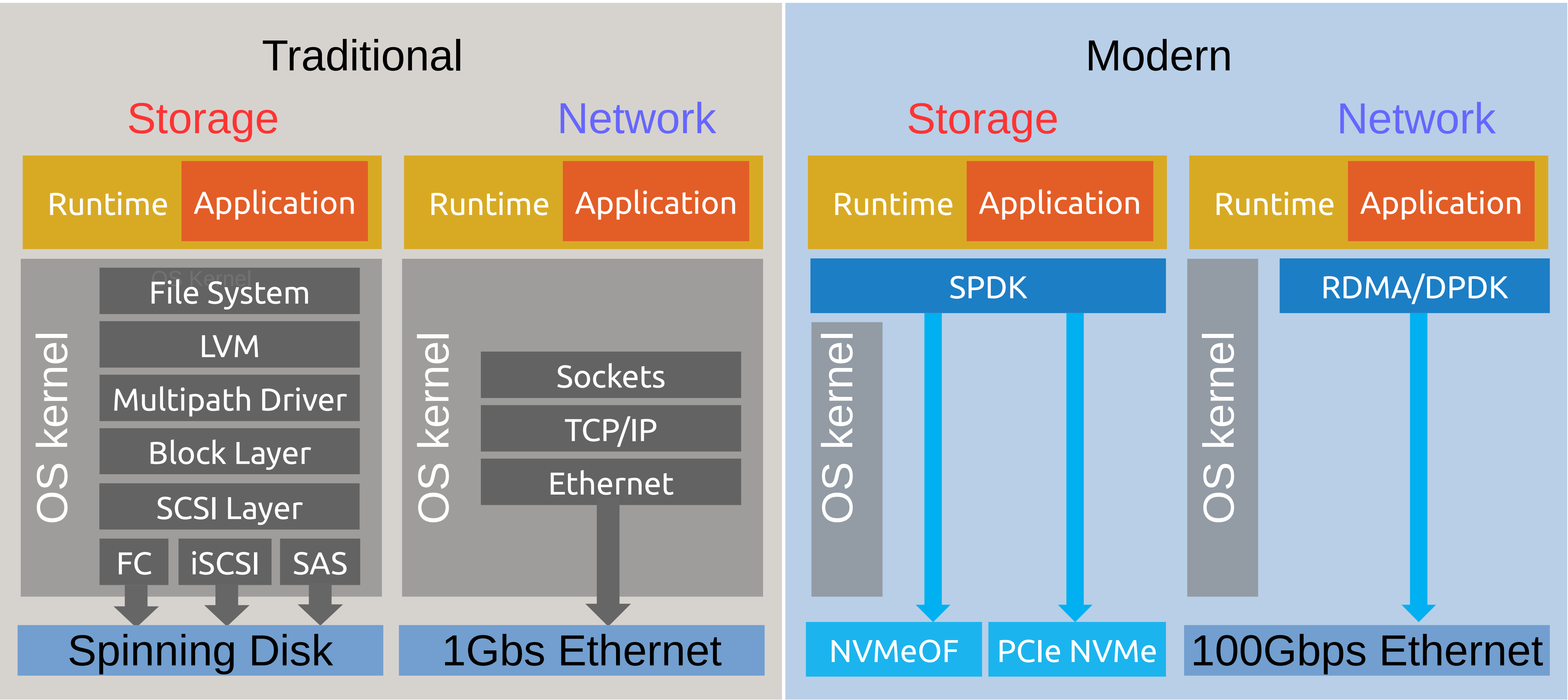
databricks

Diversity




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

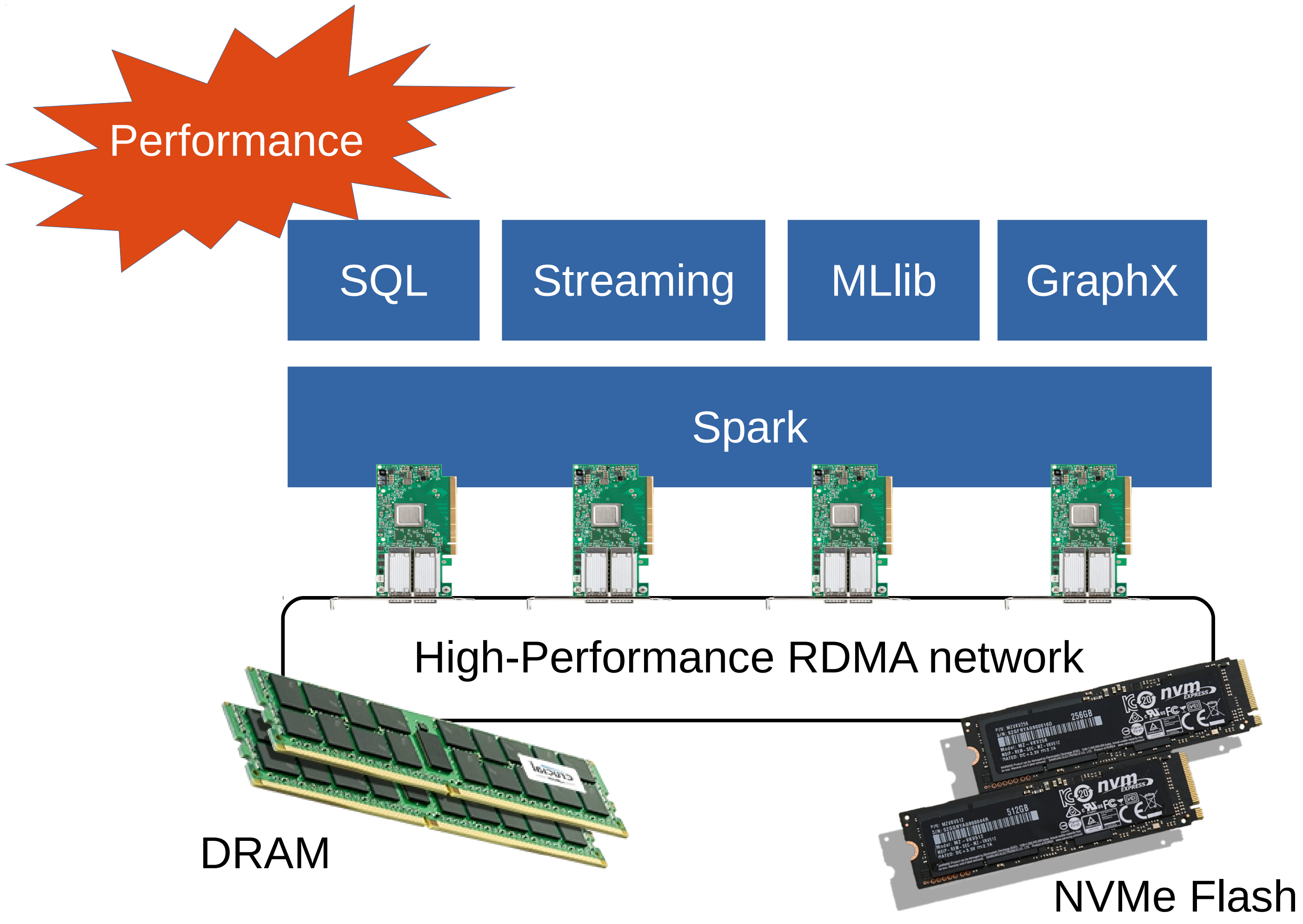
# I/O API Trends




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

# RDMA Example

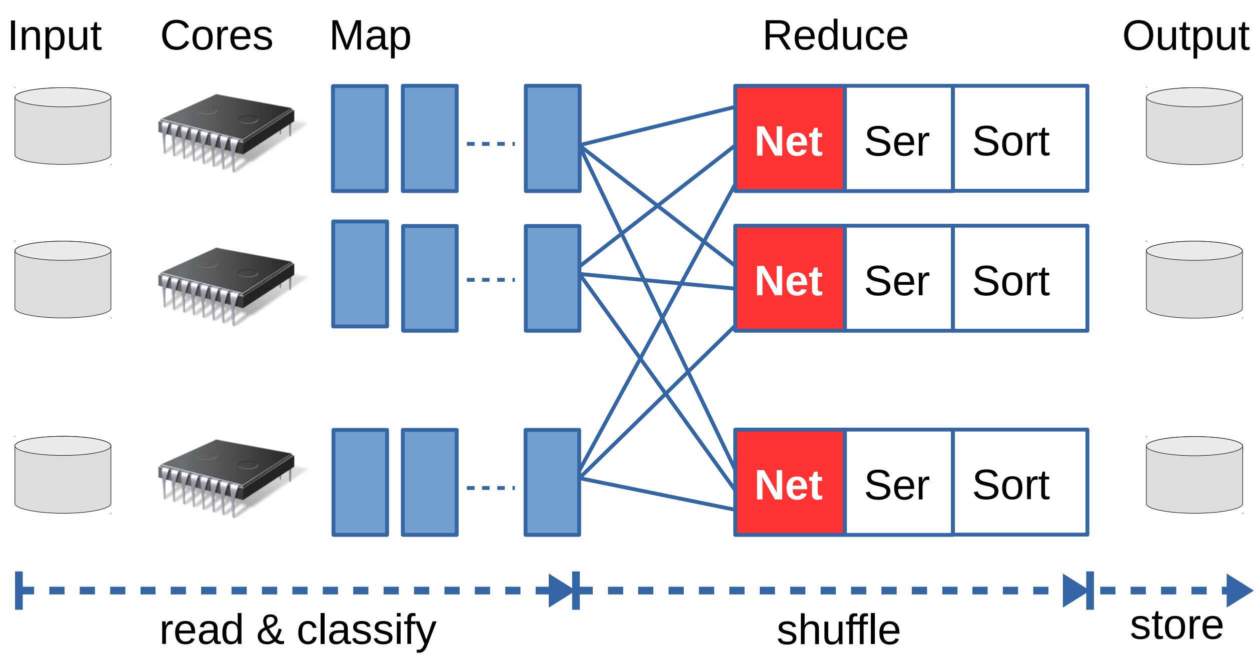
# Let's Use it!


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

# Case Study: Sorting in Spark

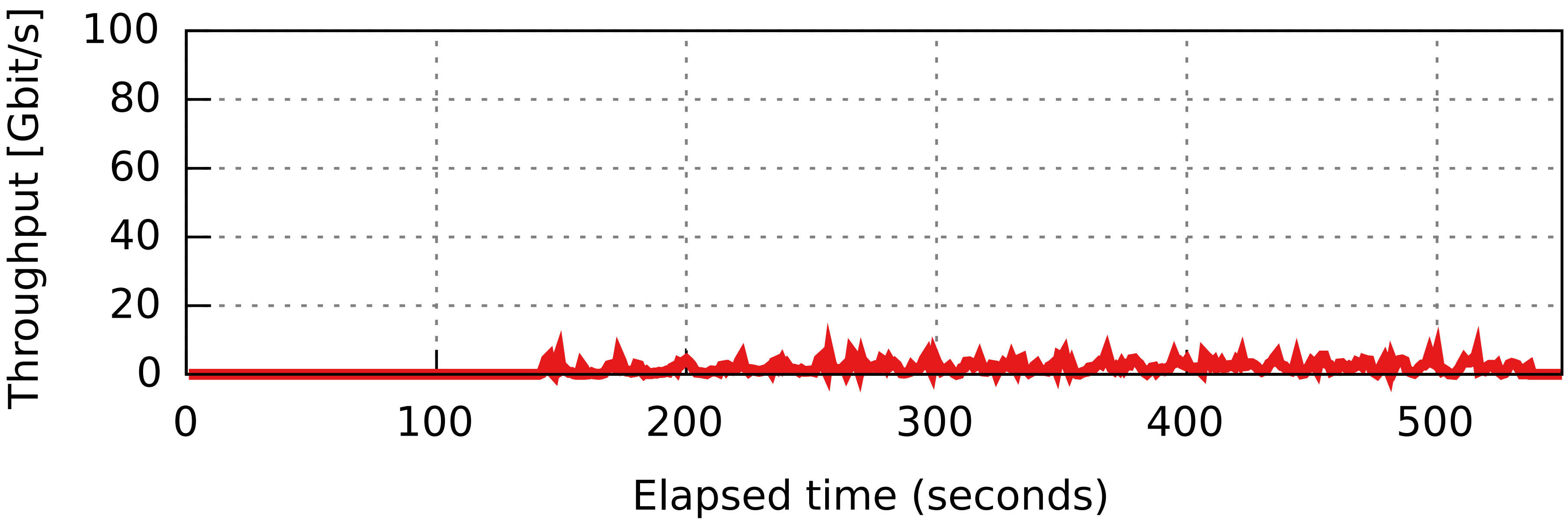
- Workload
  - Sort 12.8 TB of data on a 128 node cluster
- Cluster Hardware
  - DRAM: 512GB DDR4
  - Storage: 4x 1.2TB NVMe SSD
  - Network: 100GbE Mellanox RDMA
- Software
  - Ubuntu 16.04 with Linux kernel
  - Spark 2.0.0

# Anatomy of Sorting in Spark



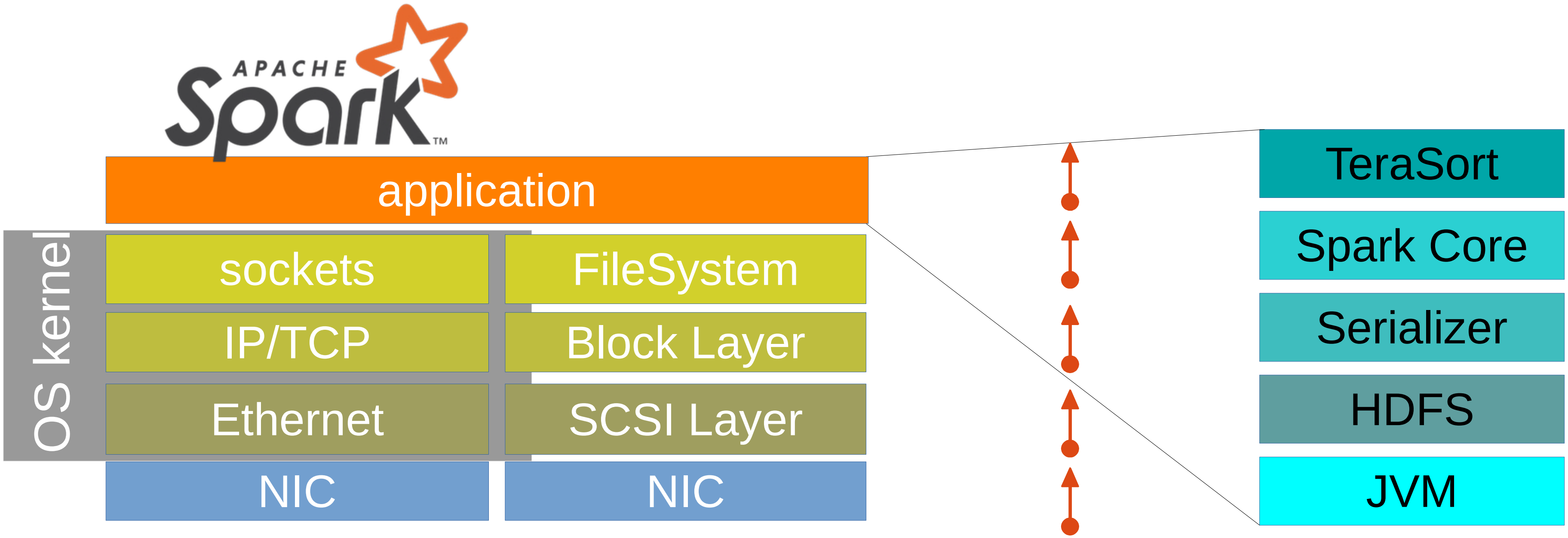

- Map task classify data into local files (typically absorbed by buffer cache)
- Reduce task fetch remote files over the network
- Sorting requires the entire data set to be shuffled over the network

# How is the Network Used?



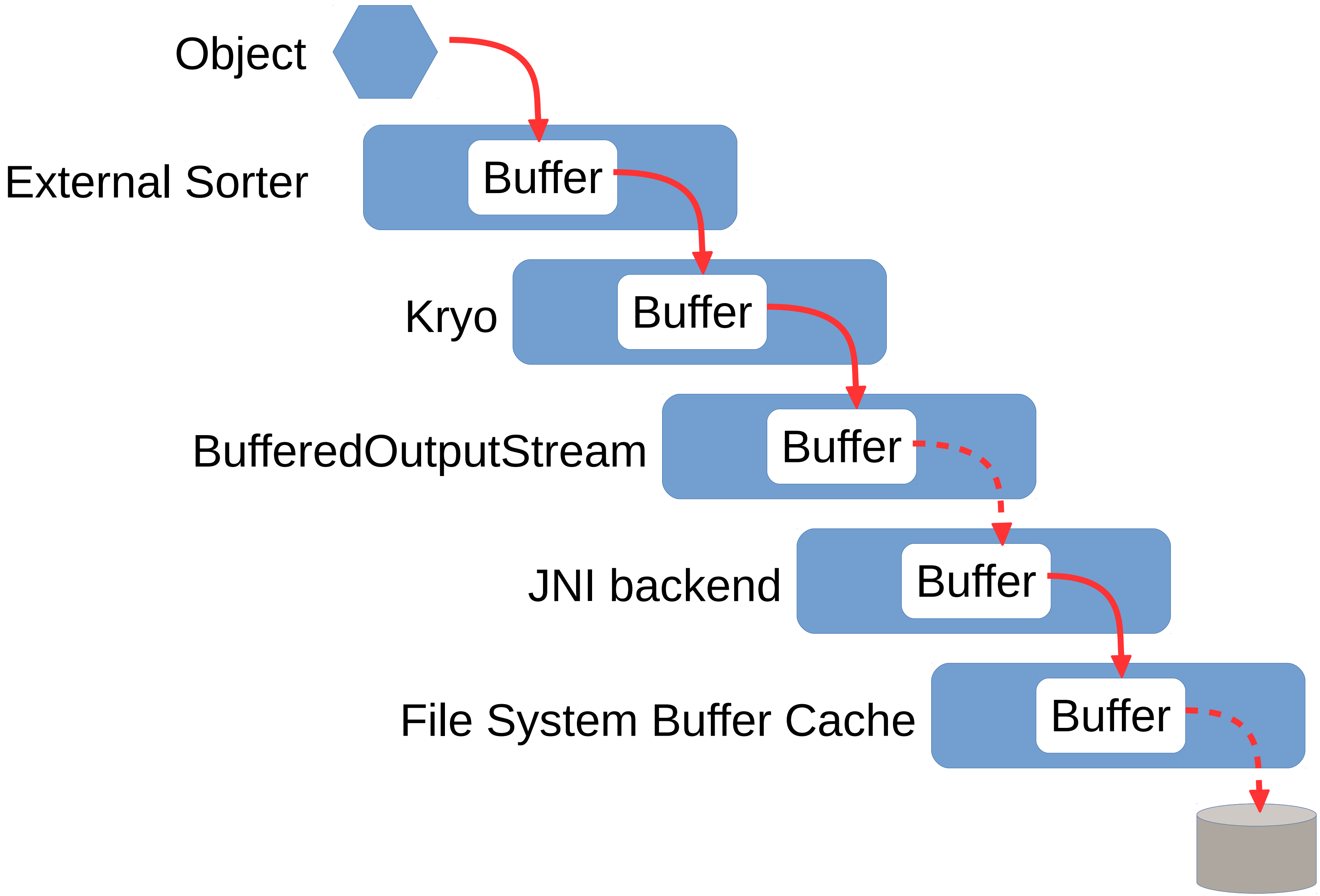

- Only 5-10 Gpbs of the network is being used

# What is the Problem

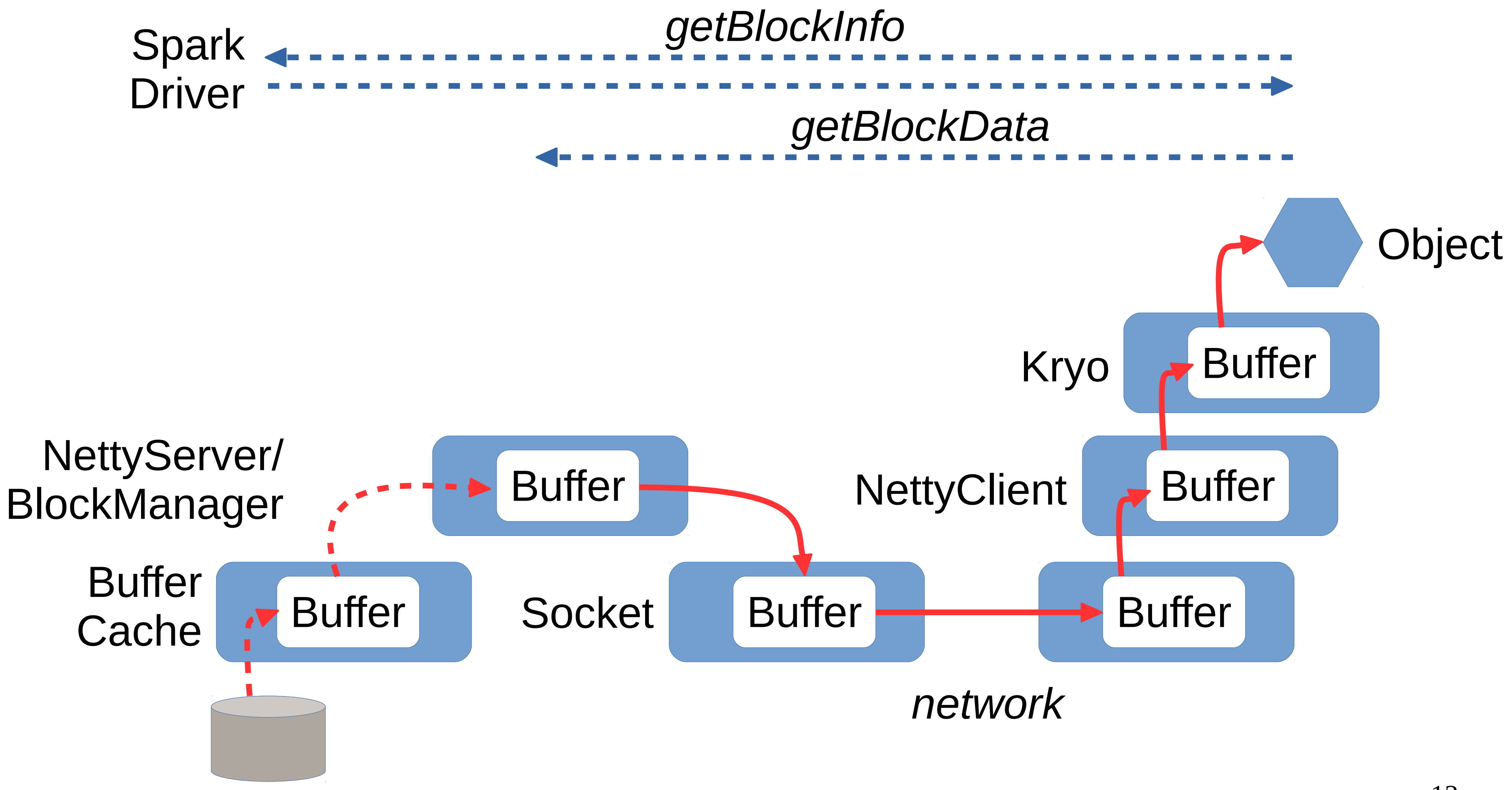



- Application use the legacy APIs
- Applications themselves are heavily layered!
- Overhead during local file system writing
- Overhead during network processing
  - Data copies, context switches, cache pollution, etc

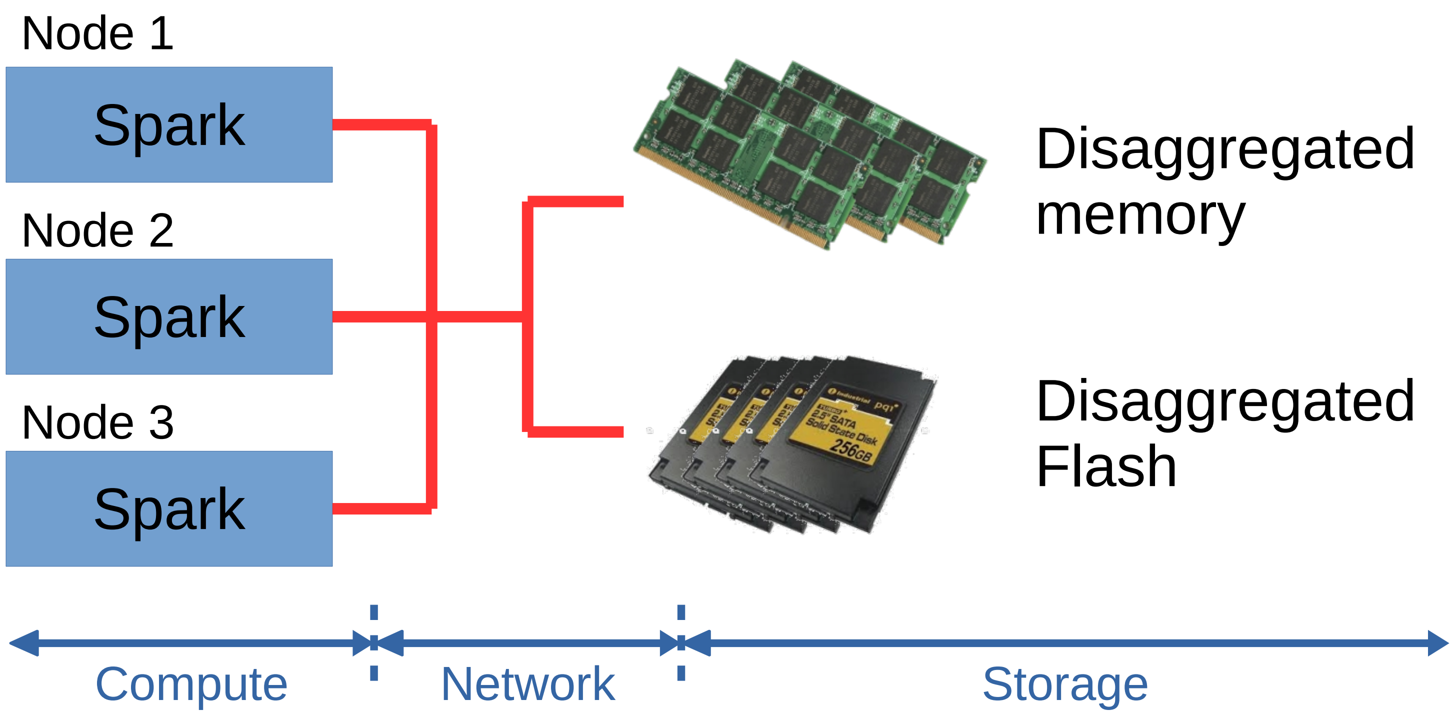
# Example: Shuffle Writer (map)



# Example: Shuffle Reader (reduce)



# Other Challenges

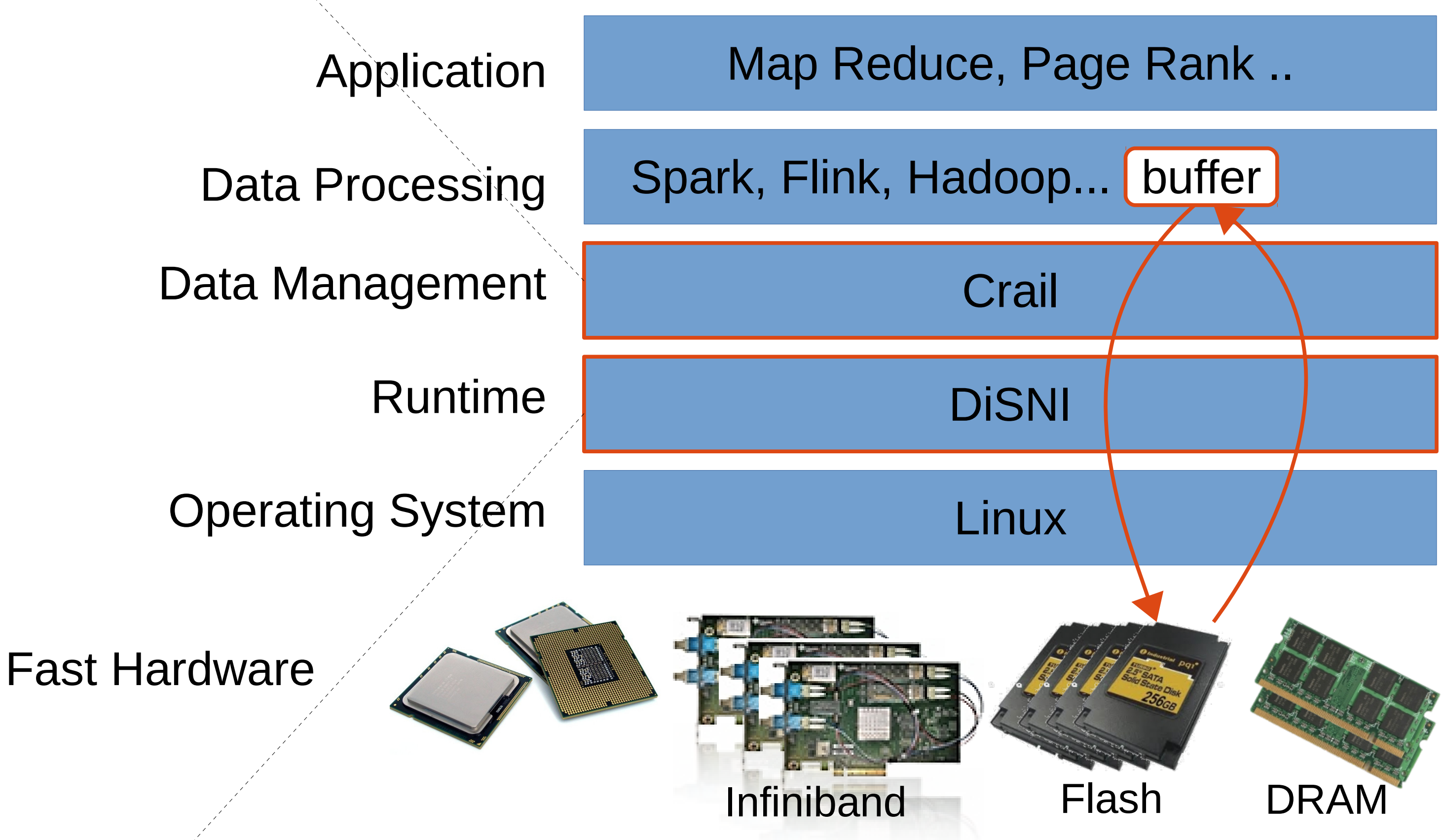



- Can't easily exploit new I/O properties, e.g., local ~= remote
  - Difficult to disaggregate memory and storage
  - Difficult to leverage storage hierarchies

# The Crail Project: Performance View

Fast distributed storage for temporary data

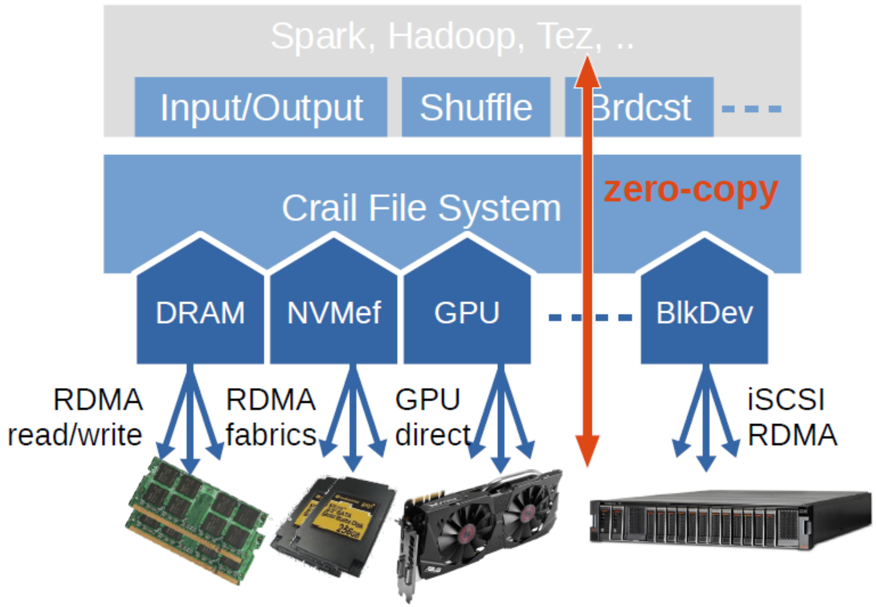
data is exchanged directly between Spark and I/O devices (network & storage)




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

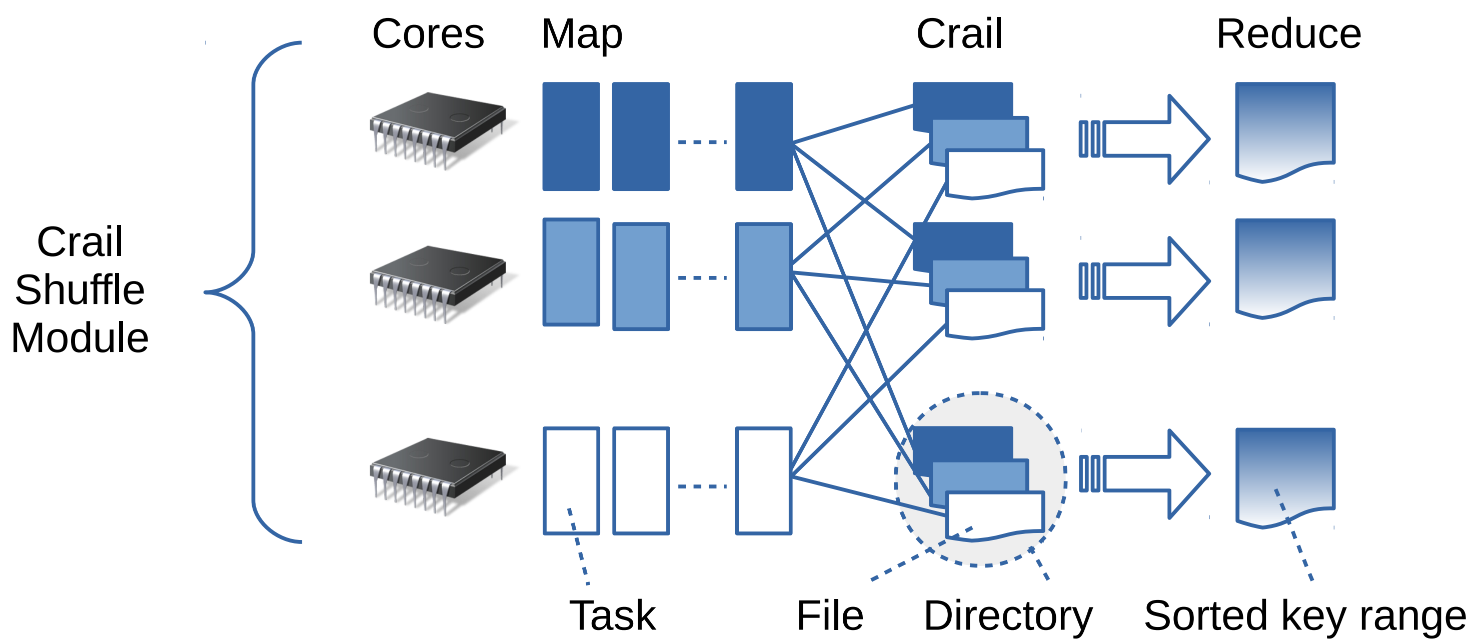
JVM I/O bypass

# 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

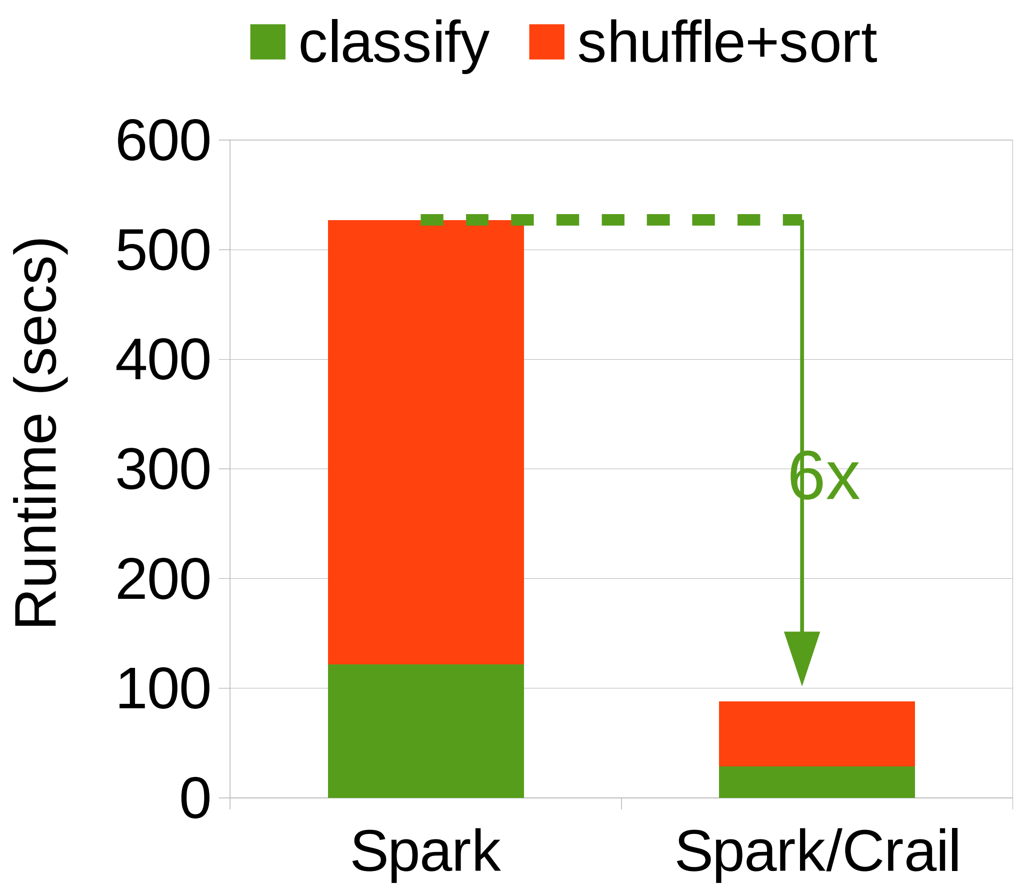
# Evaluation – Terasort

**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

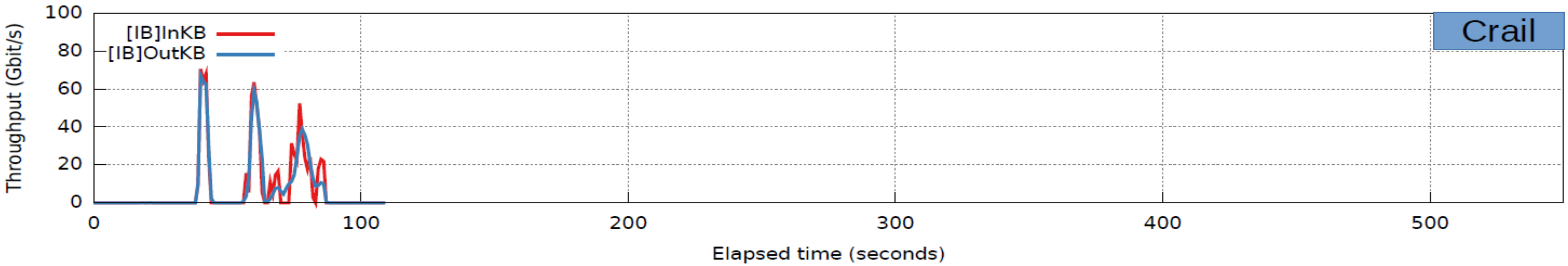
**Performance gain: 6x**

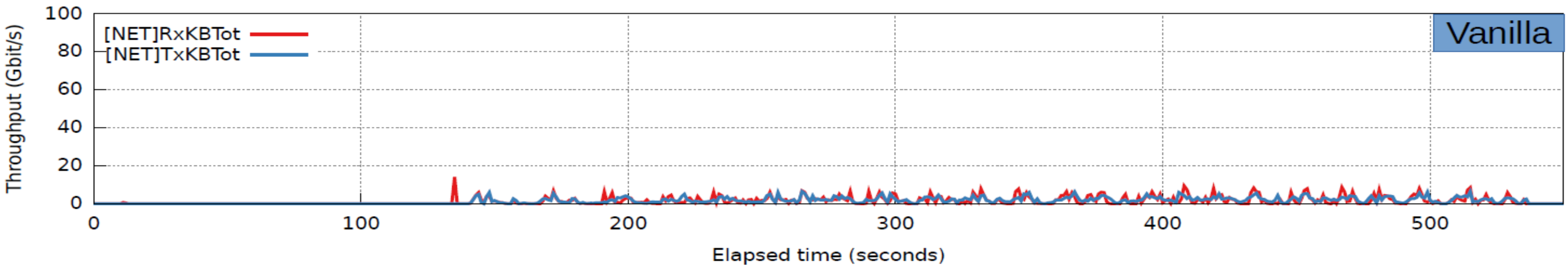
- Most gain from reduce phase:
  - Crail shuffler much faster than Spark build-in
  - Dramatically reduced CPU involvement
  - Dramatically improved network usage
- Map phase: all activity local
  - Still faster than vanilla Spark




12.8 TB data set, TeraSort

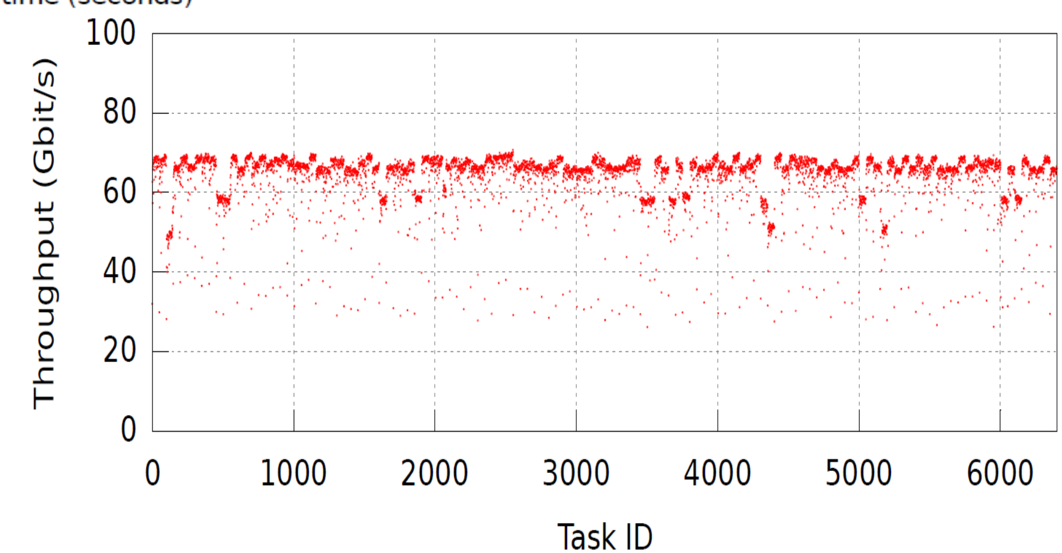
# Evaluation – Network IO




- 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

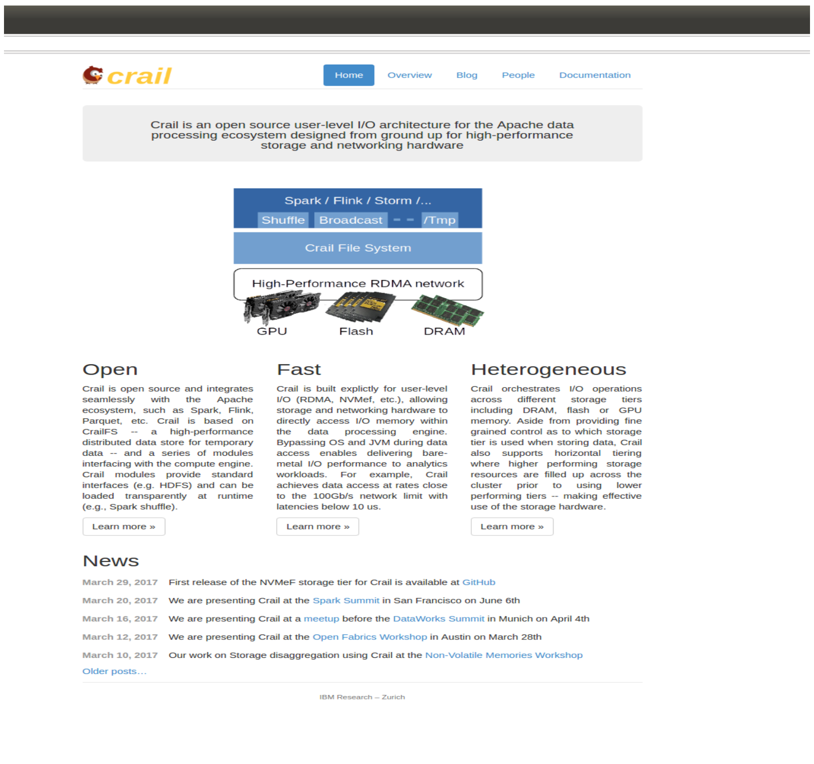
| | Spark + Crail | Spark 2.0.2 | Winner 2014 | Winner 2016 |
|---|---|---|---|---|
| Size TB | 12.8 | | 100 | |
| Time sec | 98 | 527 | 1406 | 98.6 |
| Cores | 2560 | | 6592 | 10240 |
| Nodes | 128 | | 206 | 512 |
| NW Gb/s | 100 | | 10 | 100 |
| Rate TB/min | 7.8 | 1.4 | 4.27 | 44.78 |
| Rate/core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

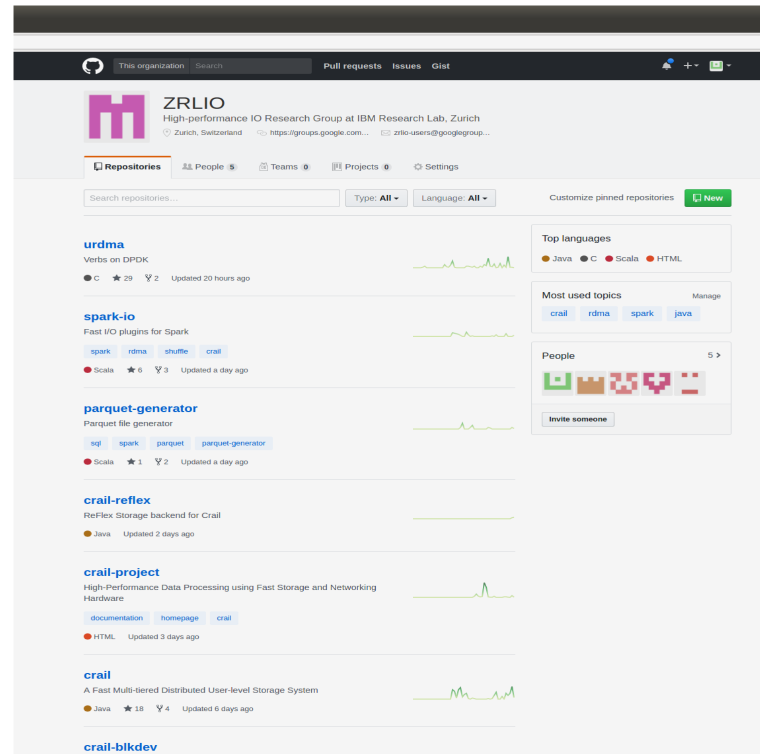
- Spark/Crail CPU efficiency is close to 2016 sorting benchmark winner: **3.13 vs. 4.4 GB/min/core**
- 2016 winner runs native C code!

# Crail is Open Source!

www.crail.io

https://github.com/zrlio



# 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

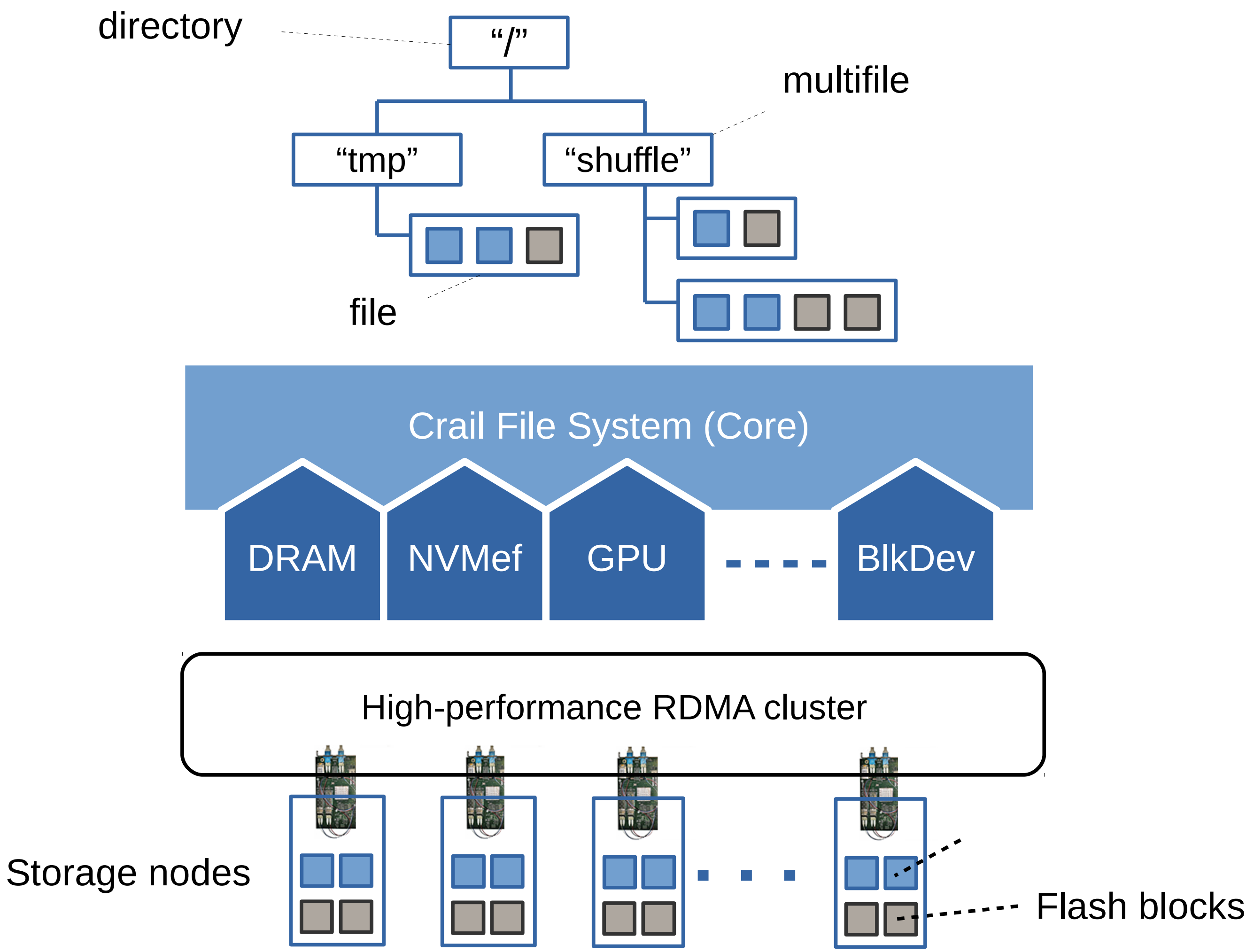
## Today's open source analytics stacks:

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

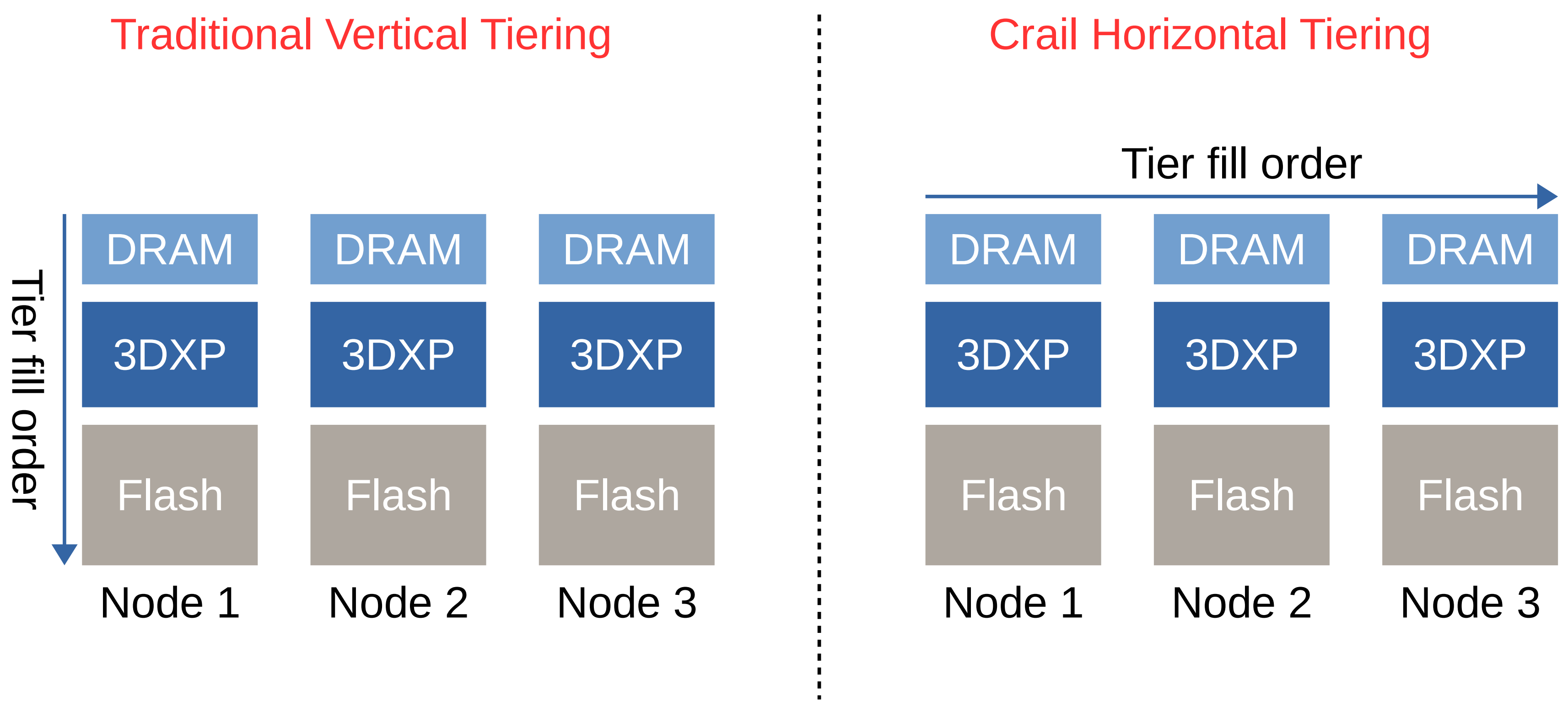
## The Crail Approach:

- 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