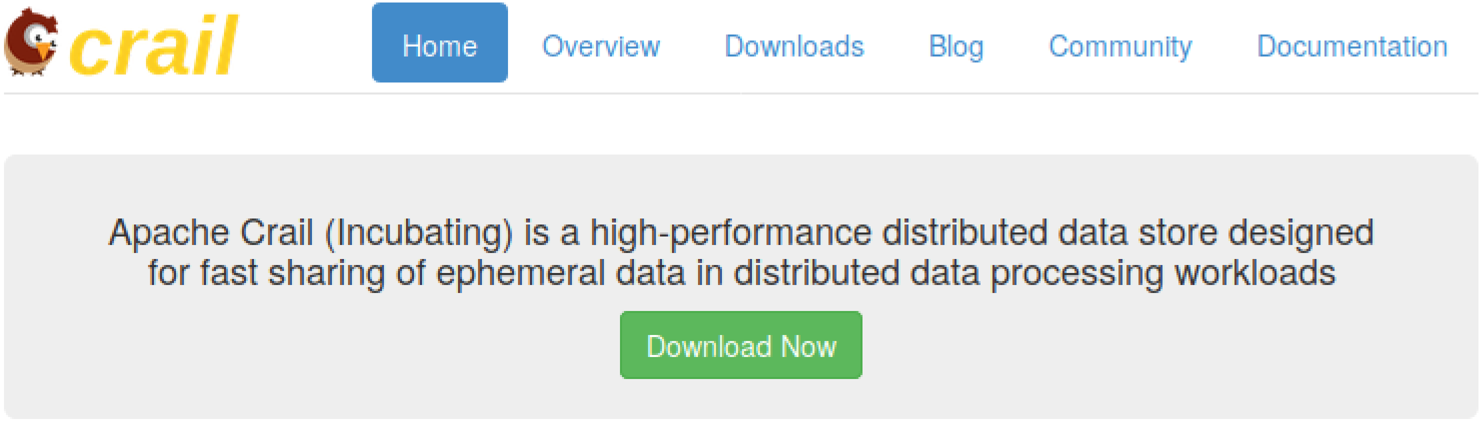
Data Processing at the Speed of 100 Gbps using Apache Crail

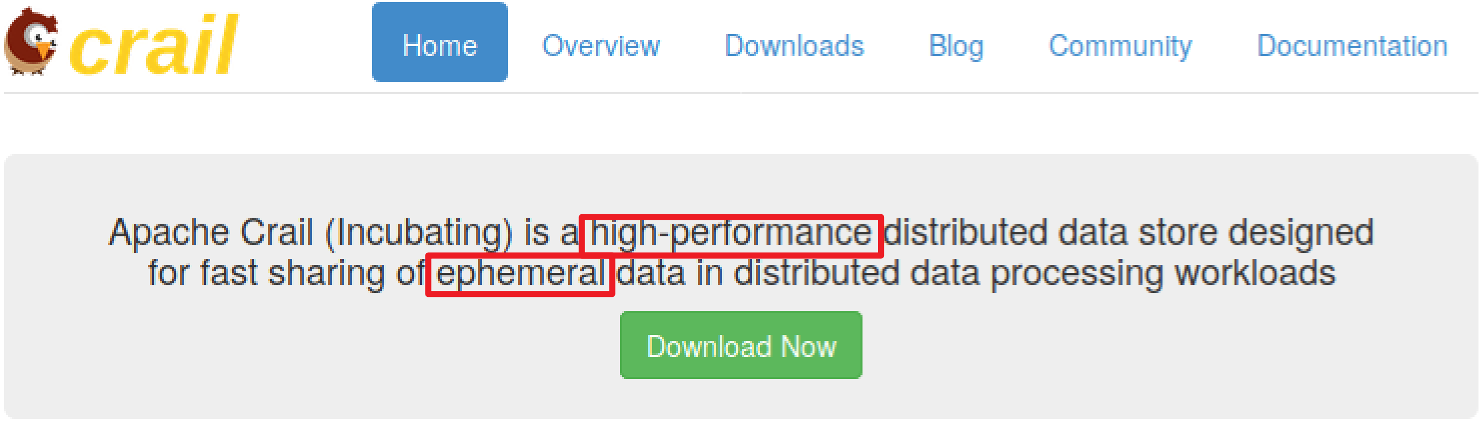
Patrick Stuedi
IBM Research

Apache Crail (crail.apache.org)



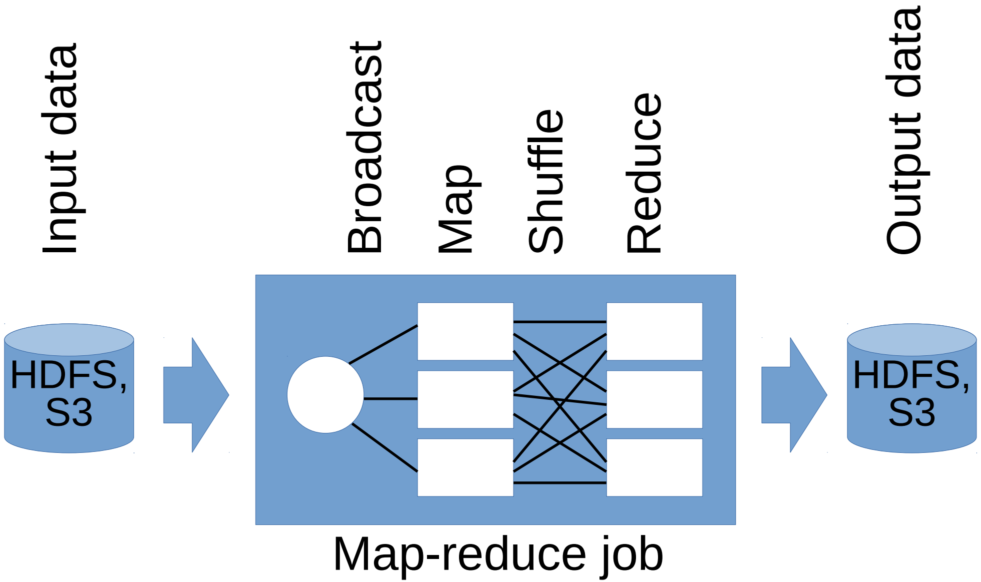
Apache Crail (Incubating) is a high-performance distributed data store designed for fast sharing of ephemeral data in distributed data processing workloads

Apache Crail (crail.apache.org)

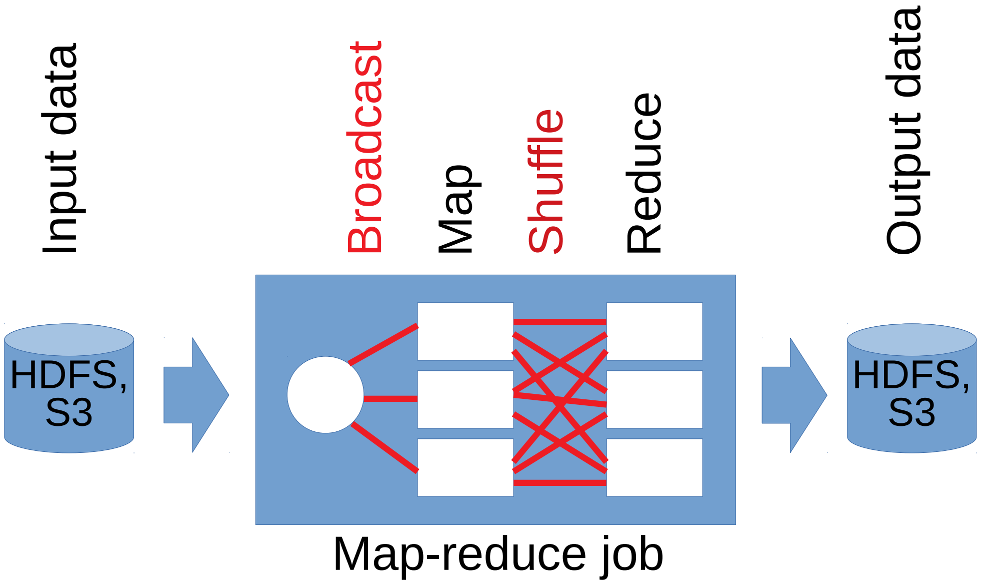


Apache Crail (Incubating) is a high-performance distributed data store designed for fast sharing of ephemeral data in distributed data processing workloads

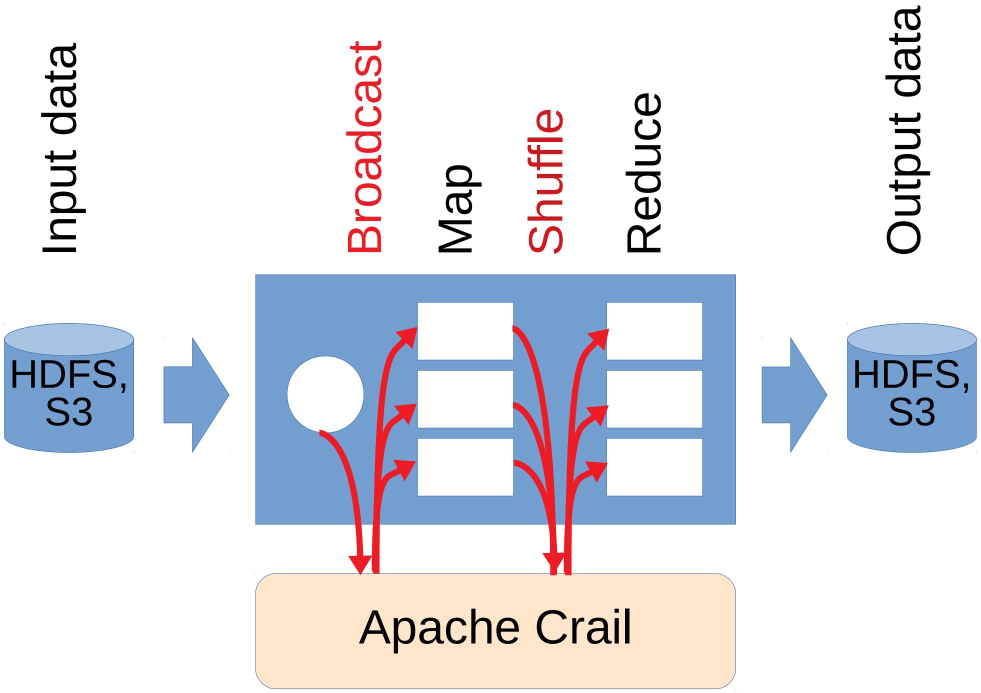
Ephemeral Data



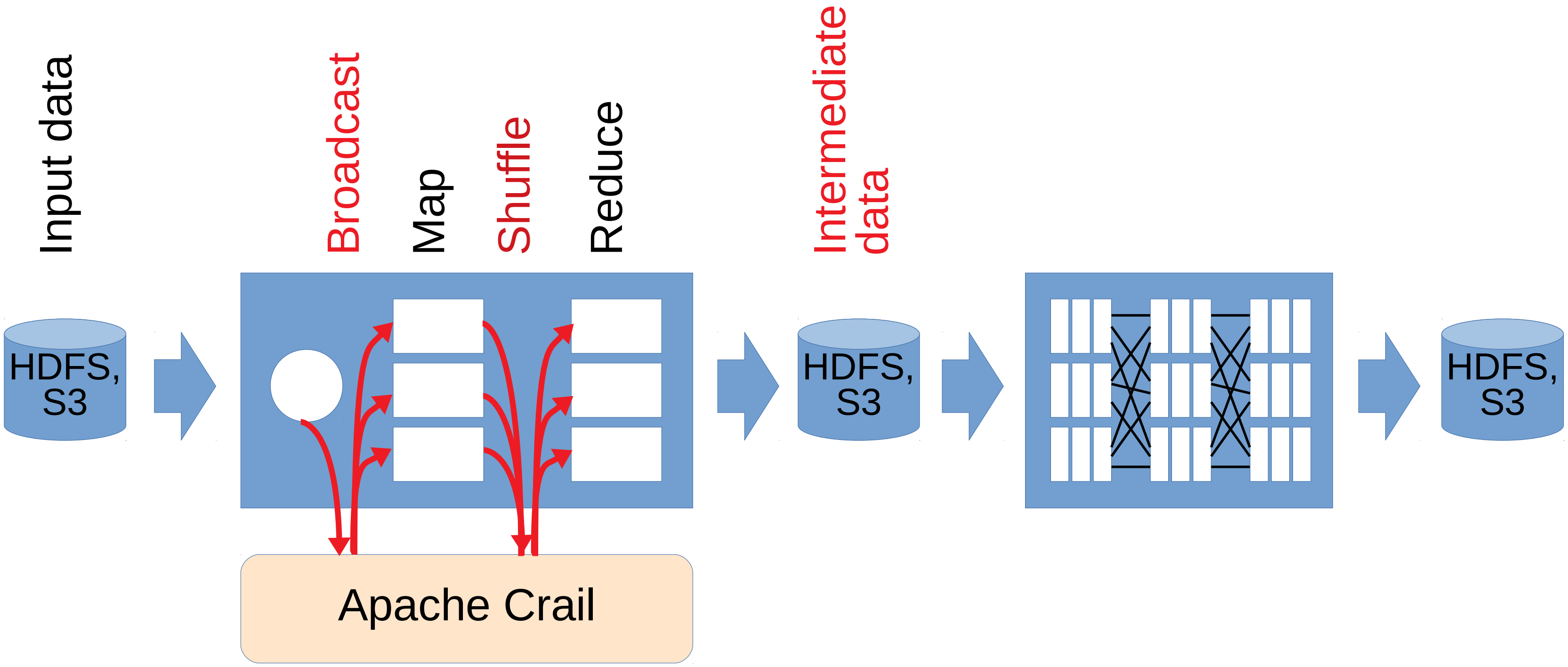
Ephemeral Data



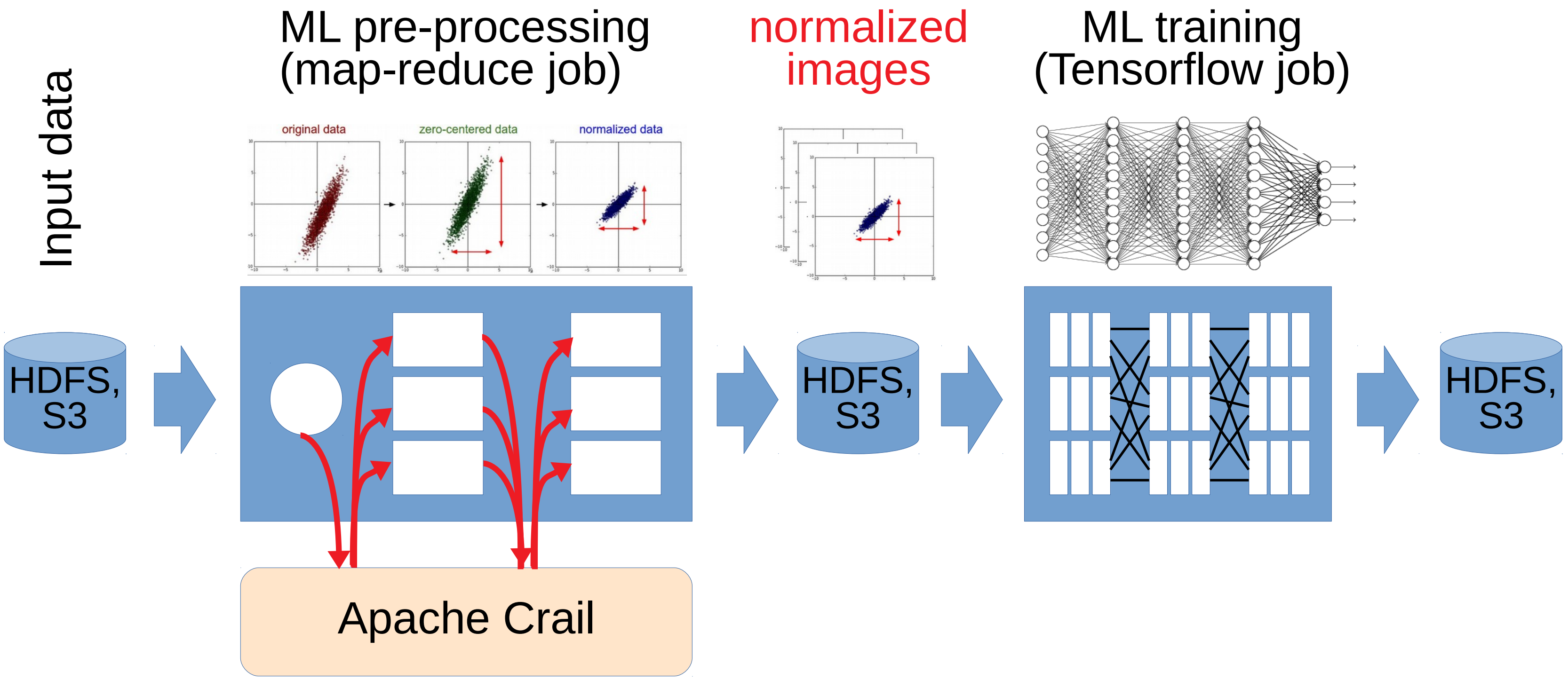
Ephemeral Data



Ephemeral Data



Ephemeral Data



Input data

ML pre-processing (map-reduce job)

normalized images

ML training (Tensorflow job)

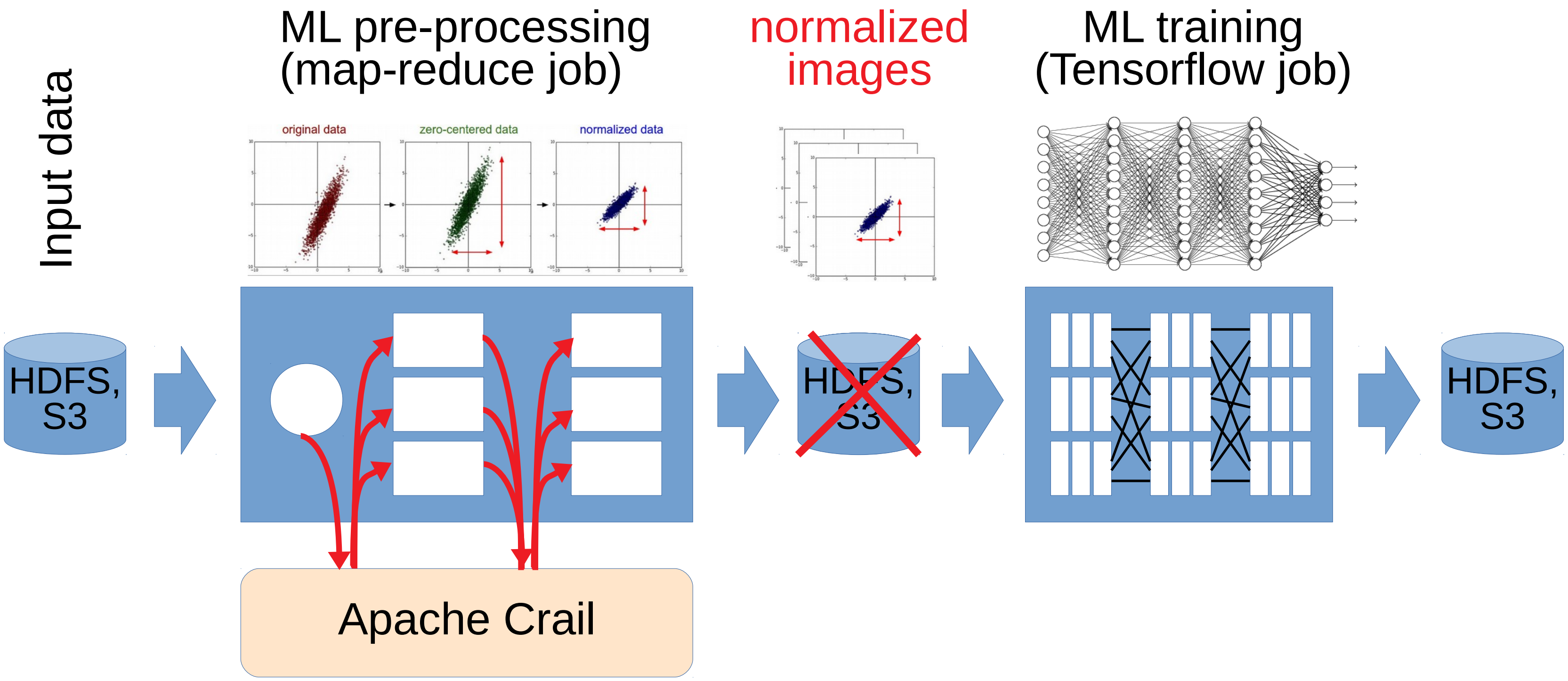
HDFS, S3

HDFS, S3

HDFS, S3

Apache Crail

Ephemeral Data



Input data

ML pre-processing
(map-reduce job)

normalized
images

ML training
(Tensorflow job)

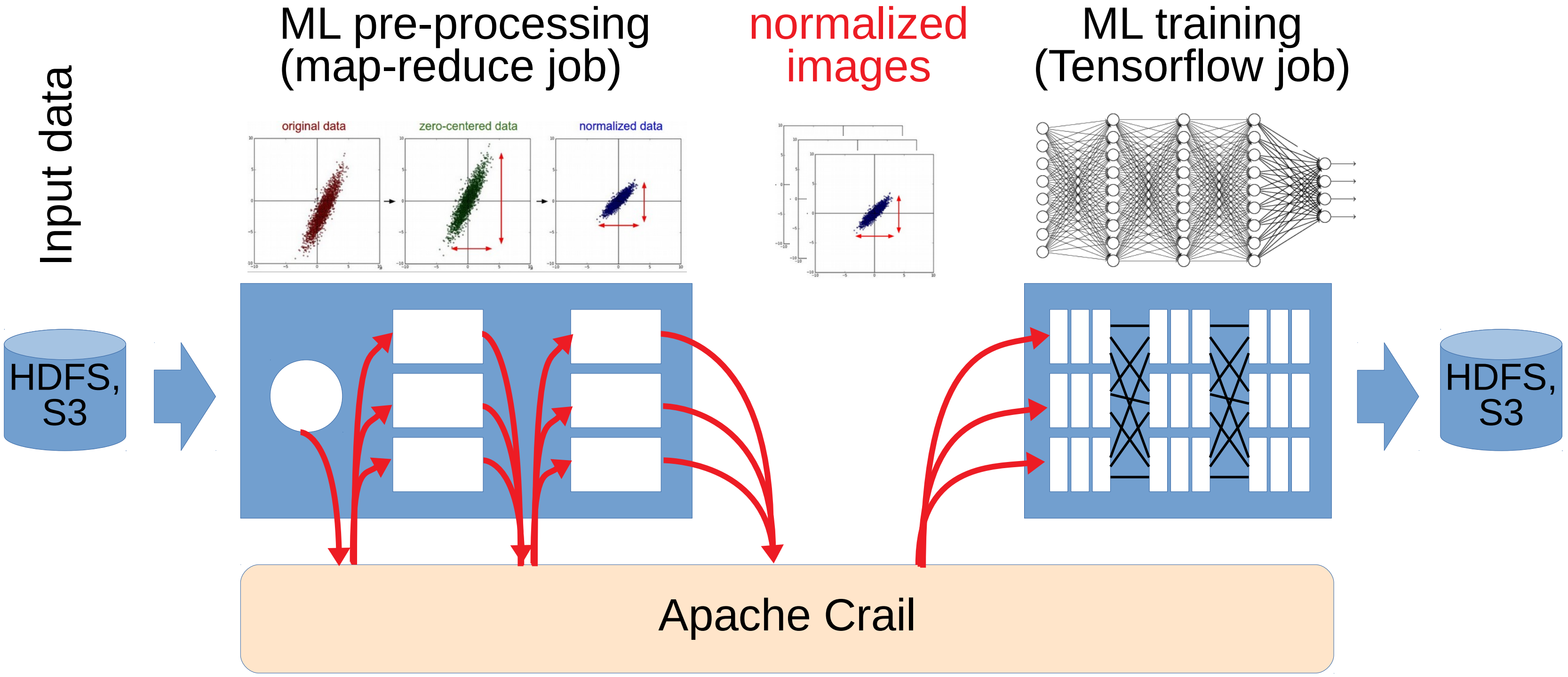
HDFS,
S3

HDFS,
S3

HDFS,
S3

Apache Crail

Ephemeral Data



Input data

ML pre-processing (map-reduce job)

normalized images

ML training (Tensorflow job)

HDFS, S3

Apache Crail

HDFS, S3

Why/when to use Crail

HDD

10 Gb/s Ethernet

Why/when to use Crail

No Crail needed

HDD

10 Gb/s Ethernet

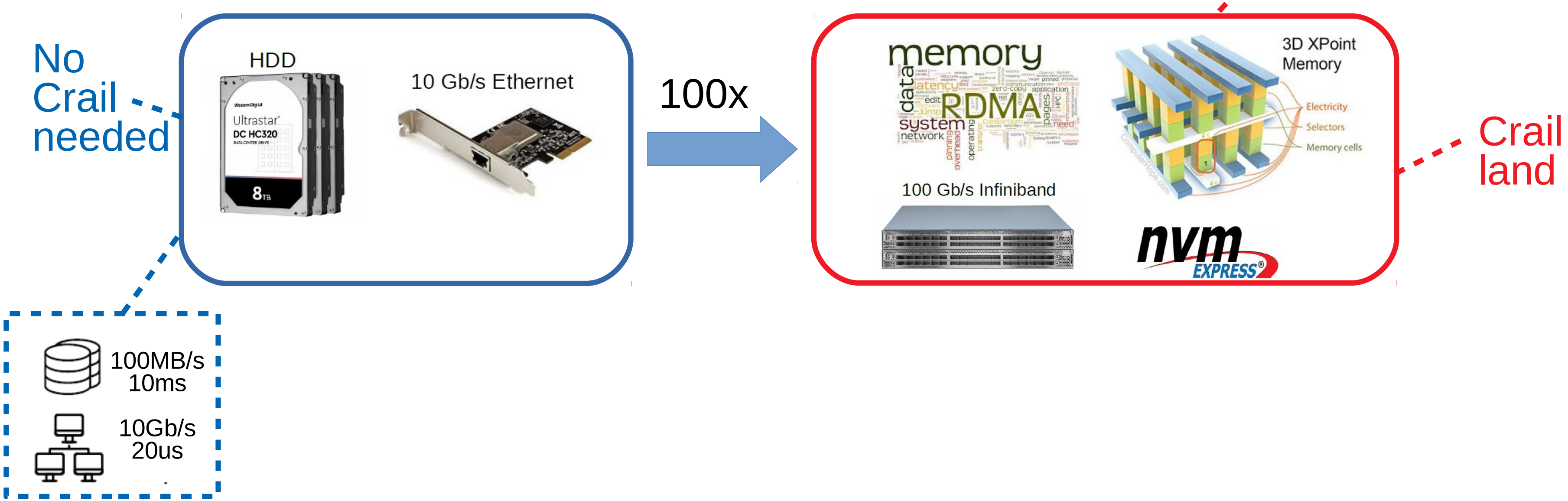
100MB/s
10ms

10Gb/s
20us

Why/when to use Crail

10GB/s
10us

200Gb/s
1us



No Crail needed

HDD

10 Gb/s Ethernet

100x

100 Gb/s Infiniband

Crail land

100MB/s
10ms

10Gb/s
20us

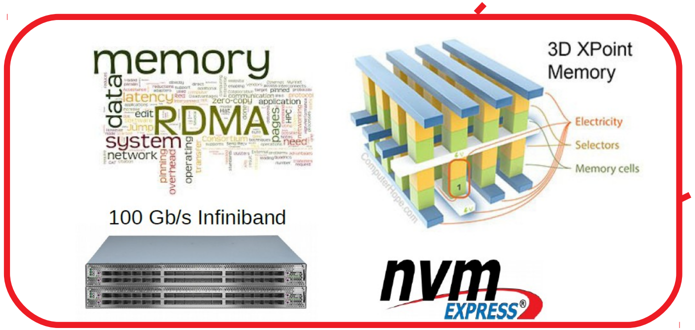
Why/when to use Crail

10GB/s
10us

200Gb/s
1us

No
Crail
needed

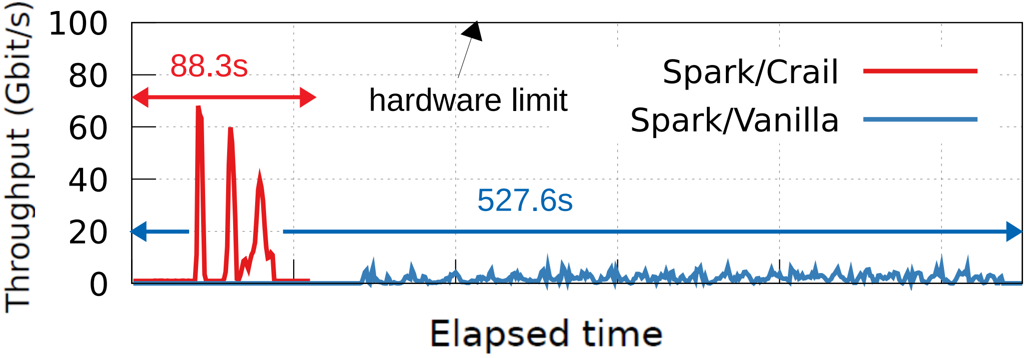
100x



Crail
land

100MB/s
10ms

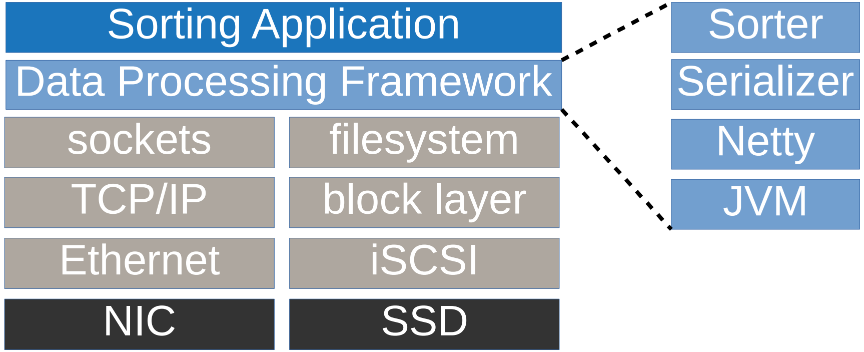
10Gb/s
20us



Terasort
12.8 TB data
128 nodes

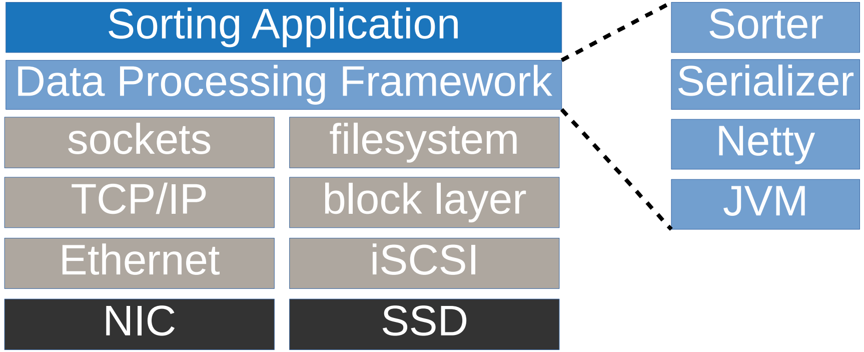
Performance Challenge

	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

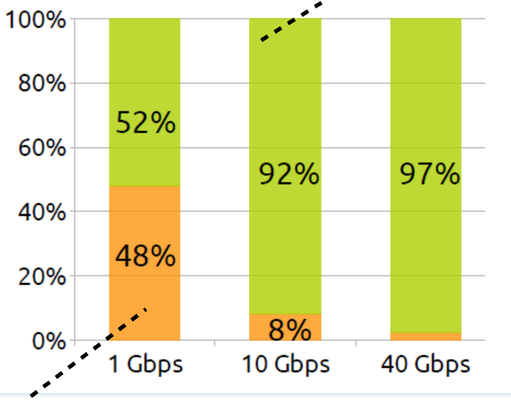


Performance Challenge

	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



Process chunk
In reduce task

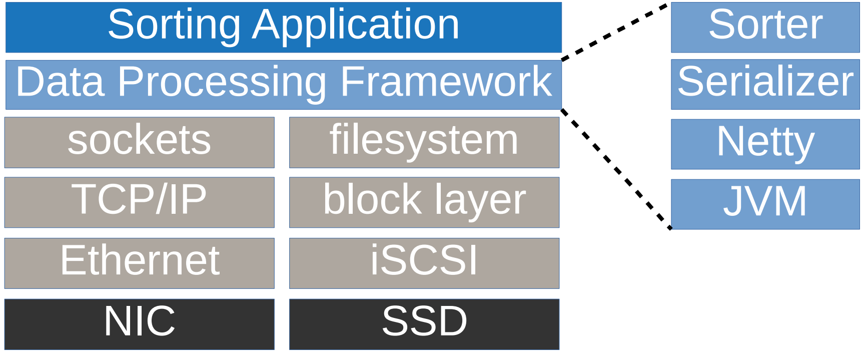


Fetch chunk
Over the network

HotNets'16

Performance Challenge

	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



Sorting Application

Data Processing Framework

sockets | filesystem

TCP/IP | block layer

Ethernet | iSCSI

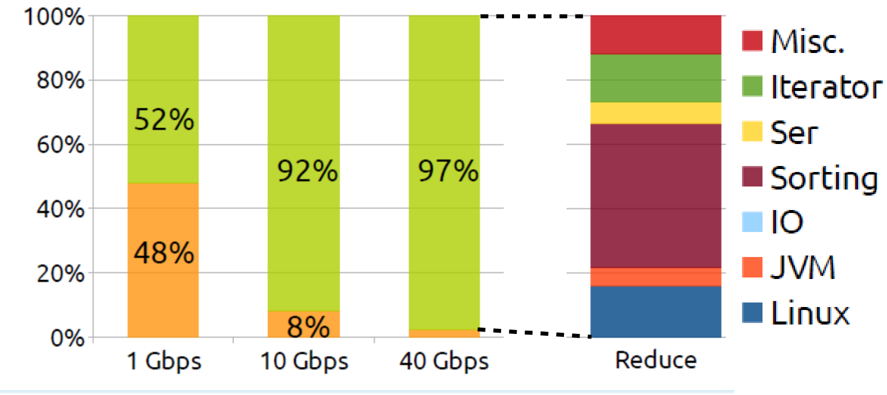
NIC | SSD

Sorter

Serializer

Netty

JVM

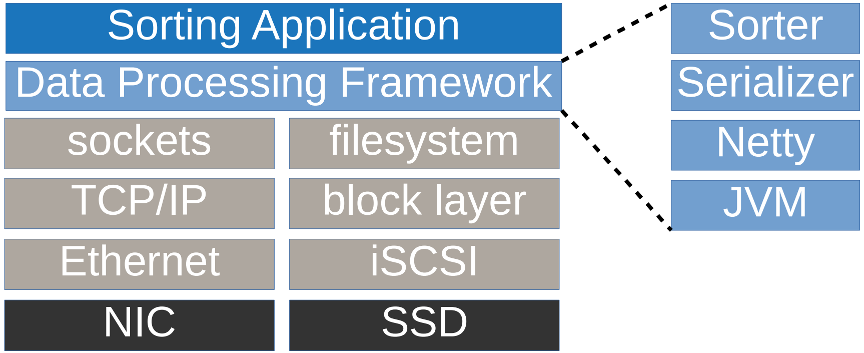


HotNets’16

Performance Challenge

	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



Sorting Application

Data Processing Framework

sockets | filesystem

TCP/IP | block layer

Ethernet | iSCSI

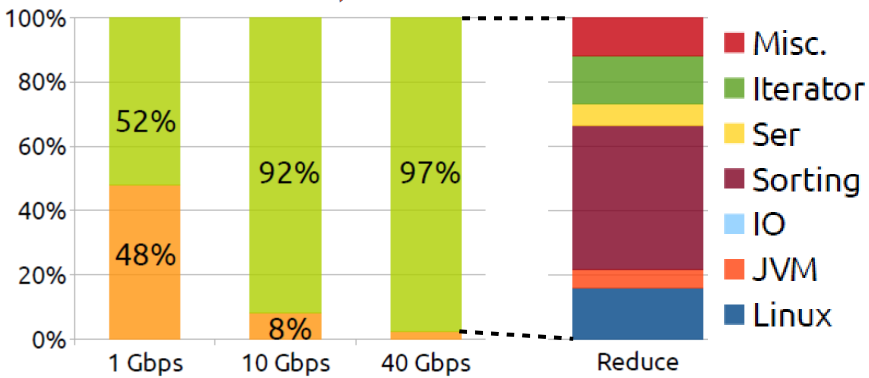
NIC | SSD

Sorter

Serializer

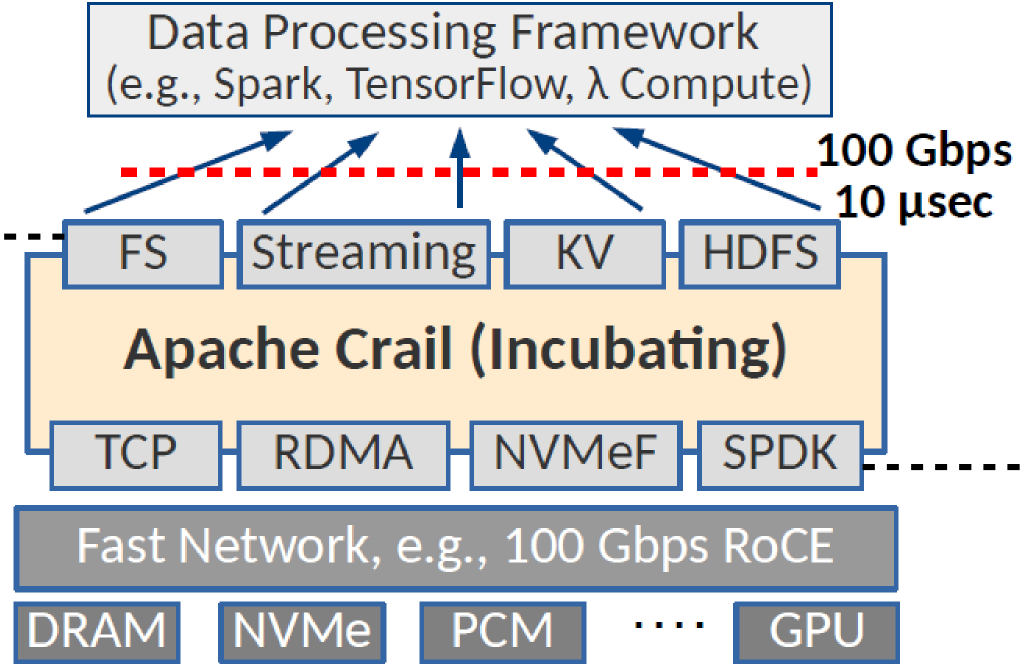
Netty

JVM



HotNets’16

Crail Overview



Data Processing Framework
(e.g., Spark, TensorFlow, λ Compute)

100 Gbps
10 µsec

FS | Streaming | KV | HDFS

Apache Crail (Incubating)

TCP | RDMA | NVMeF | SPDK

Fast Network, e.g., 100 Gbps RoCE

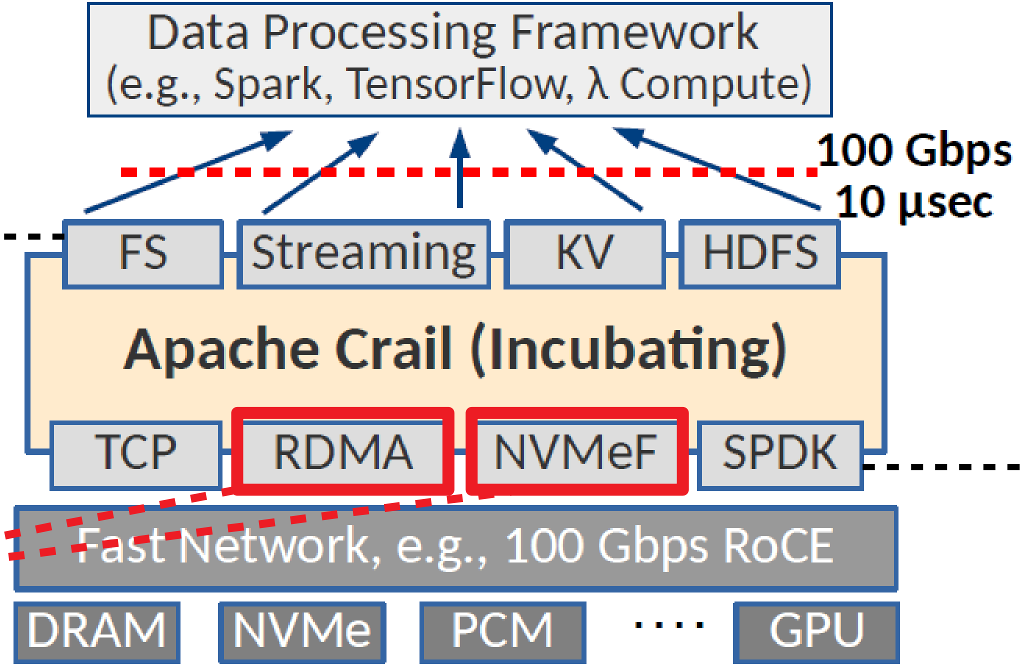
DRAM | NVMe | PCM | .... | GPU

Multiple interfaces

Multiple storage backends
(pluggable, open interface)

Crail Overview

Multiple interfaces

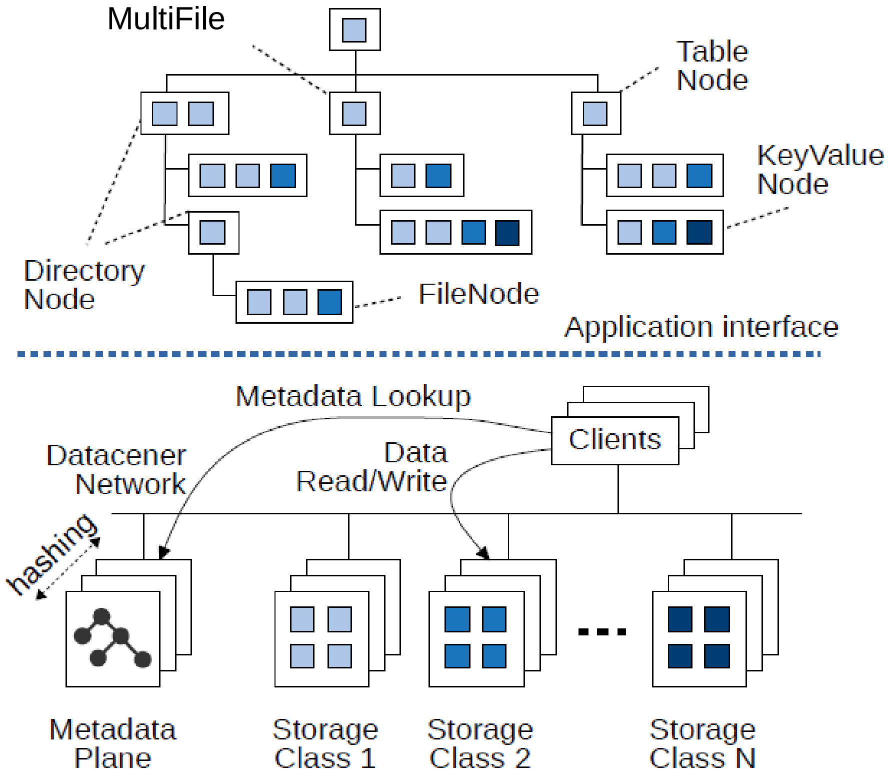


Data Processing Framework
(e.g., Spark, TensorFlow, λ Compute)
100 Gbps
10 µsec
FS
Streaming
KV
HDFS
Apache Crail (Incubating)
TCP
RDMA
NVMeF
SPDK
Fast Network, e.g., 100 Gbps RoCE
DRAM
NVMe
PCM
....
GPU

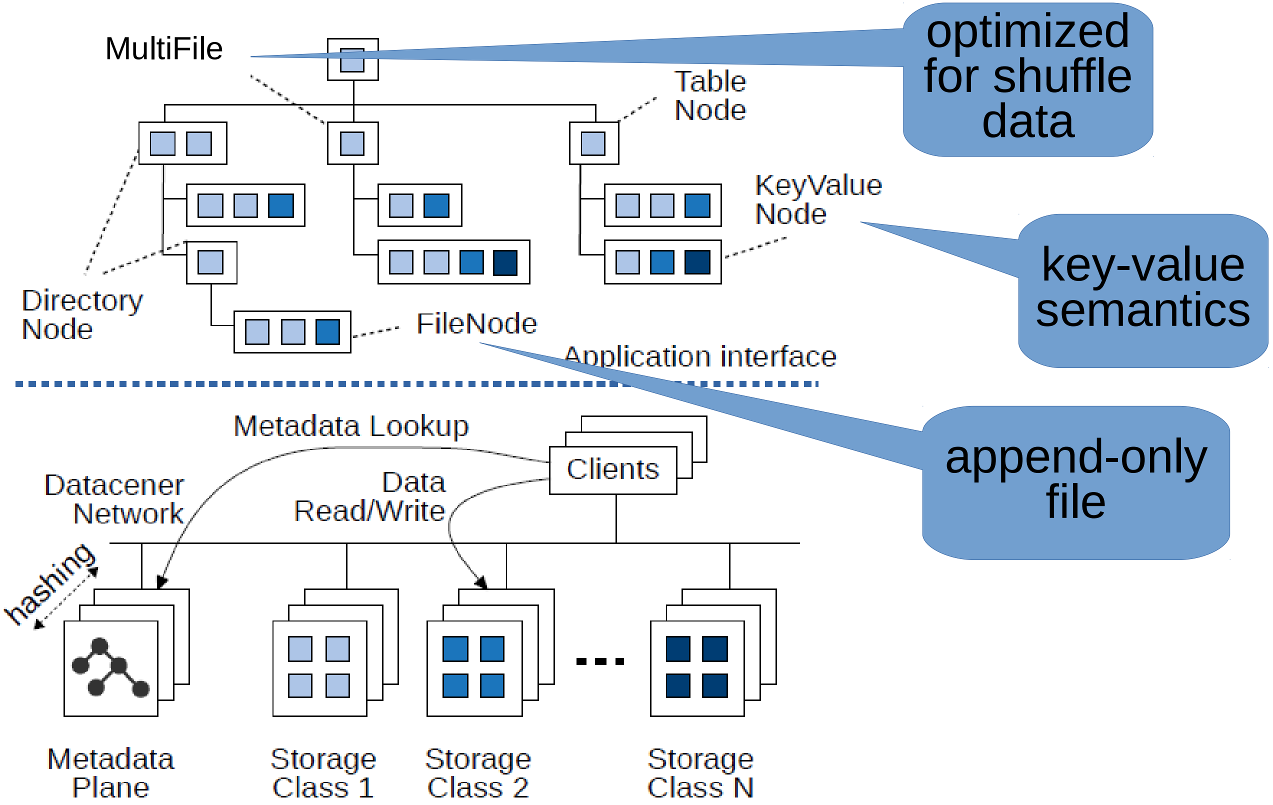
Multiple storage backends (pluggable, open interface)

primary high-performance storage backends

Crail Architecture & API



Crail Architecture & API

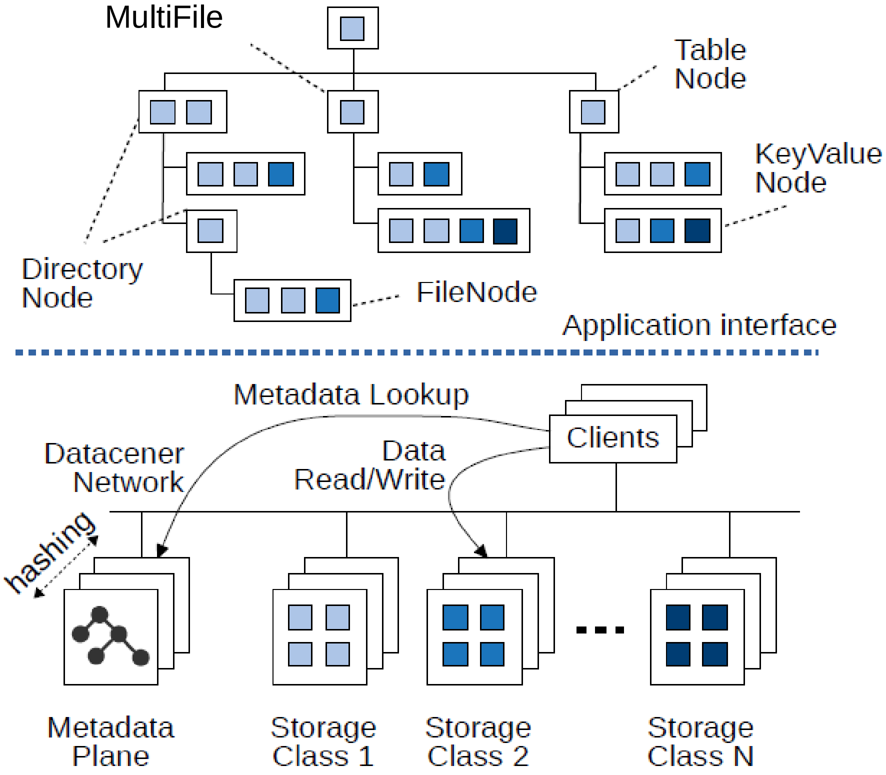


optimized for shuffle data

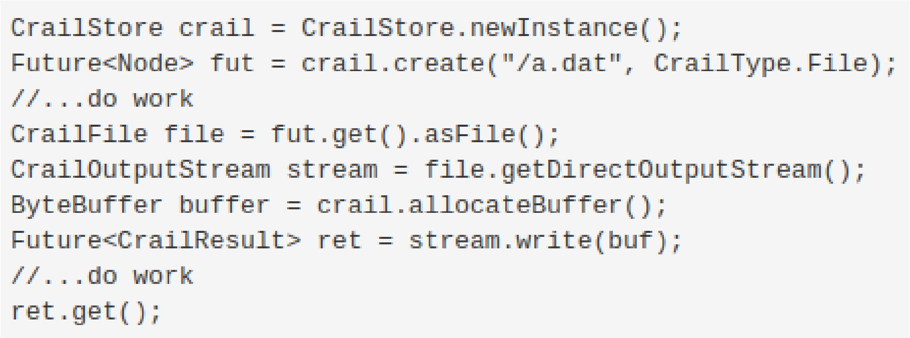
key-value semantics

append-only file

Crail Architecture & API

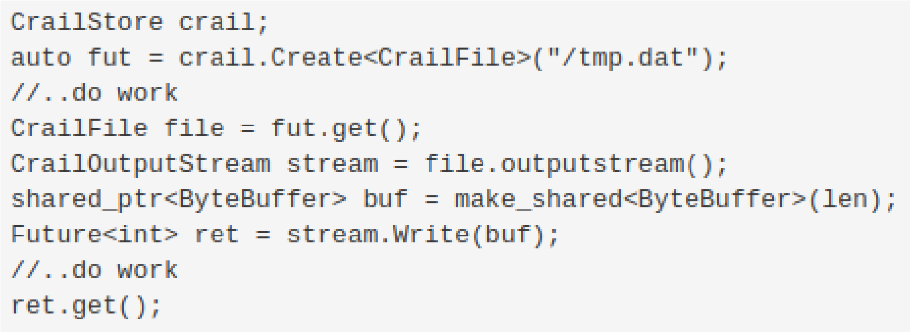


Java:



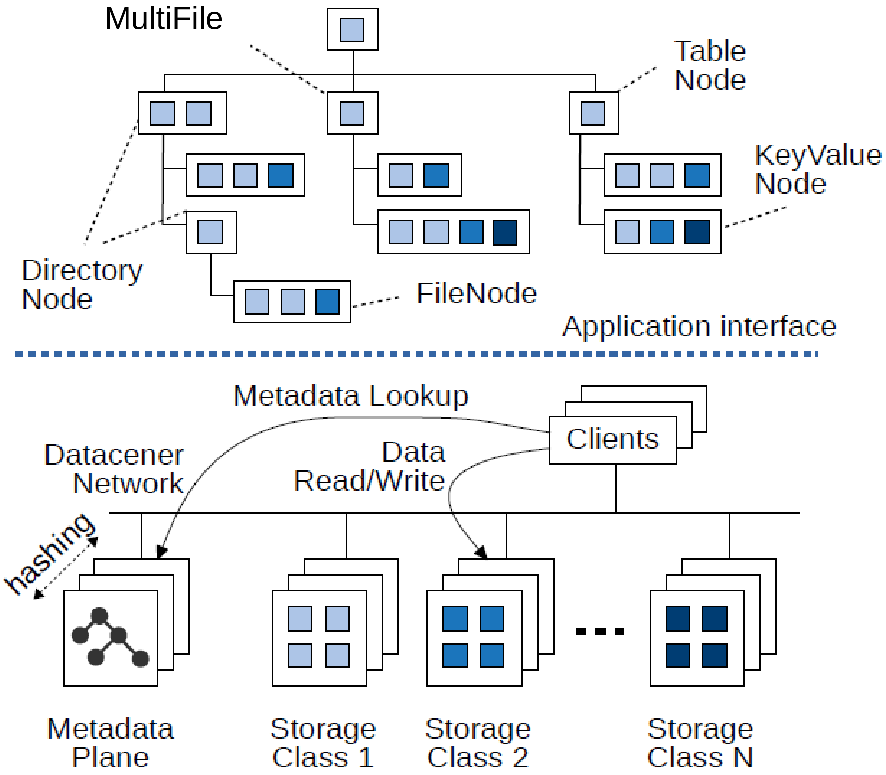
```
CrailStore crail = CrailStore.newInstance();
Future<Node> fut = crail.create("/a.dat", CrailType.File);
//...do work
CrailFile file = fut.get().asFile();
CrailOutputStream stream = file.getDirectOutputStream();
ByteBuffer buffer = crail.allocateBuffer();
Future<CrailResult> ret = stream.write(buf);
//...do work
ret.get();
```

C++:



```
CrailStore crail;
auto fut = crail.Create<CrailFile>("/tmp.dat");
//..do work
CrailFile file = fut.get();
CrailOutputStream stream = file.outputstream();
shared_ptr<ByteBuffer> buf = make_shared<ByteBuffer>(len);
Future<int> ret = stream.Write(buf);
//..do work
ret.get();
```

Crail Architecture & API



Java:

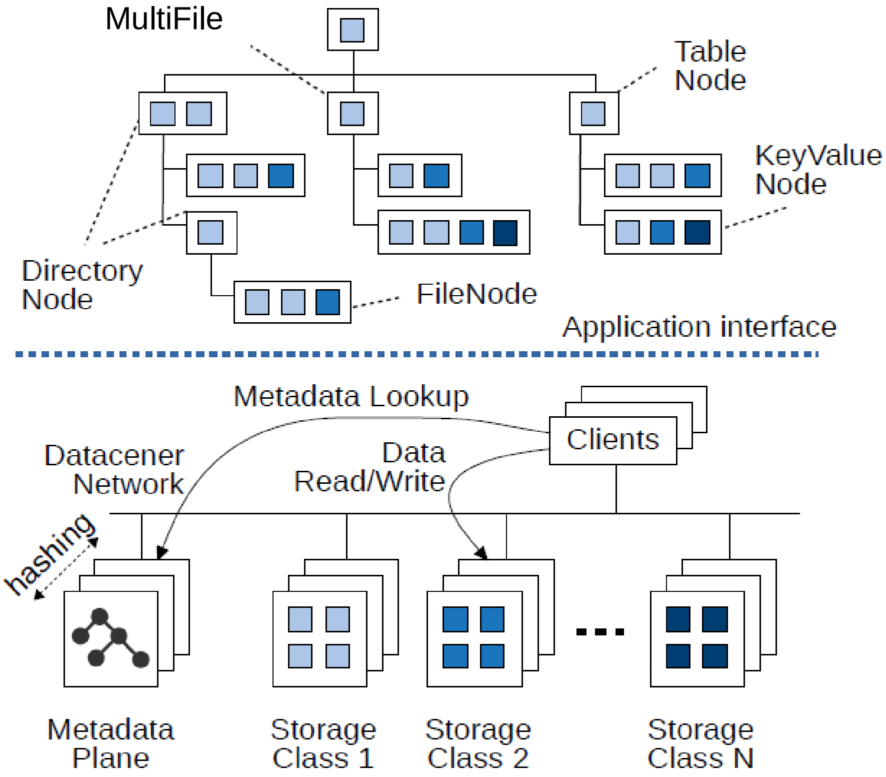
```
CrailStore crail = CrailStore.newInstance();
Future<Node> fut = crail.create("/a.dat", CrailType.File);
//...do work
CrailFile file = fut.get().asFile();
CrailOutputStream stream = file.getDirectOutputStream();
ByteBuffer buffer = crail.allocateBuffer();
Future<CrailResult> ret = stream.write(buf);
//...do work
ret.get();
```

Node type

C++:

```
CrailStore crail;
auto fut = crail.Create<CrailFile>("/tmp.dat");
//..do work
CrailFile file = fut.get();
CrailOutputStream stream = file.outputstream();
shared_ptr<ByteBuffer> buf = make_shared<ByteBuffer>(len);
Future<int> ret = stream.Write(buf);
//..do work
ret.get();
```

Crail Architecture & API



Java:

```
CrailStore crail = CrailStore.newInstance();
Future<Node> fut = crail.create("/a.dat", CrailType.File);
//...do work
CrailFile file = fut.get().asFile();
CrailOutputStream stream = file.getDirectOutputStream();
ByteBuffer buffer = crail.allocateBuffer();
Future<CrailResult> ret = stream.write(buf);
//...do work
ret.get();
```

non-blocking & asynchronous

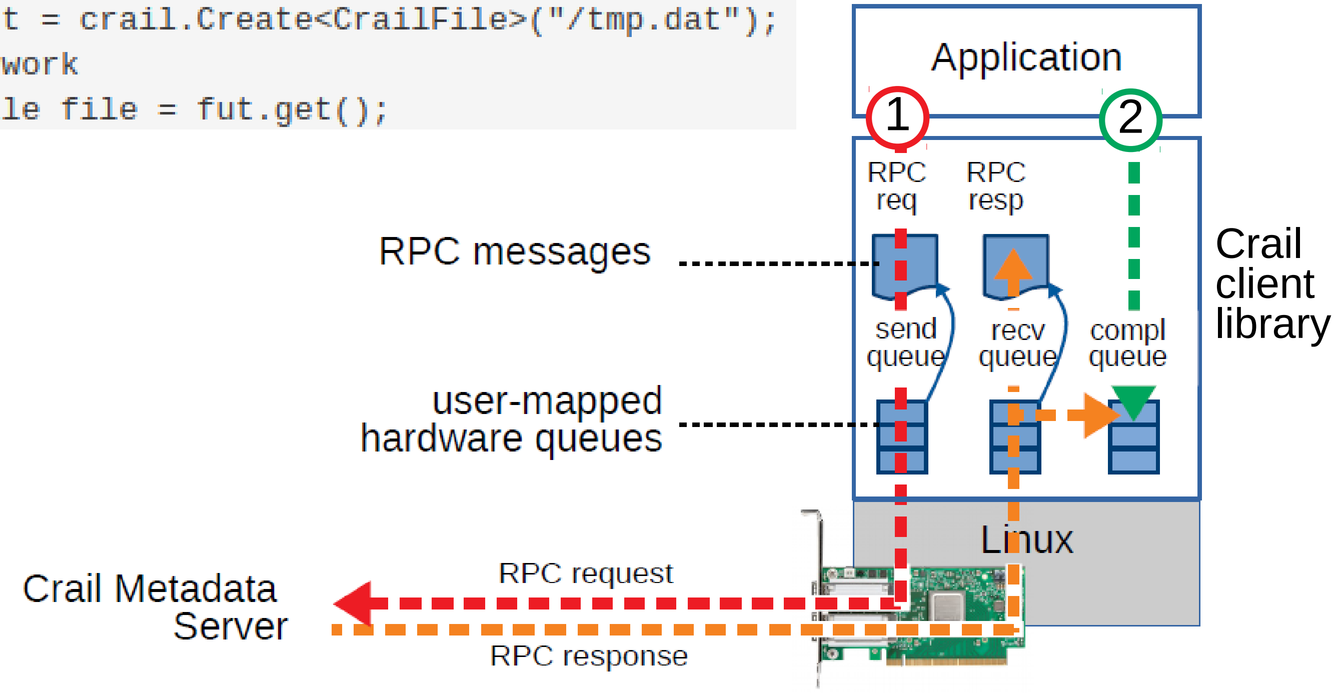
C++:

```
CrailStore crail;
auto fut = crail.Create<CrailFile>("/tmp.dat");
//..do work
CrailFile file = fut.get();
CrailOutputStream stream = file.outputstream();
shared_ptr<ByteBuffer> buf = make_shared<ByteBuffer>(len);
Future<int> ret = stream.Write(buf);
//..do work
ret.get();
```

Where does the performance come from?

User-Level I/O: Metadata

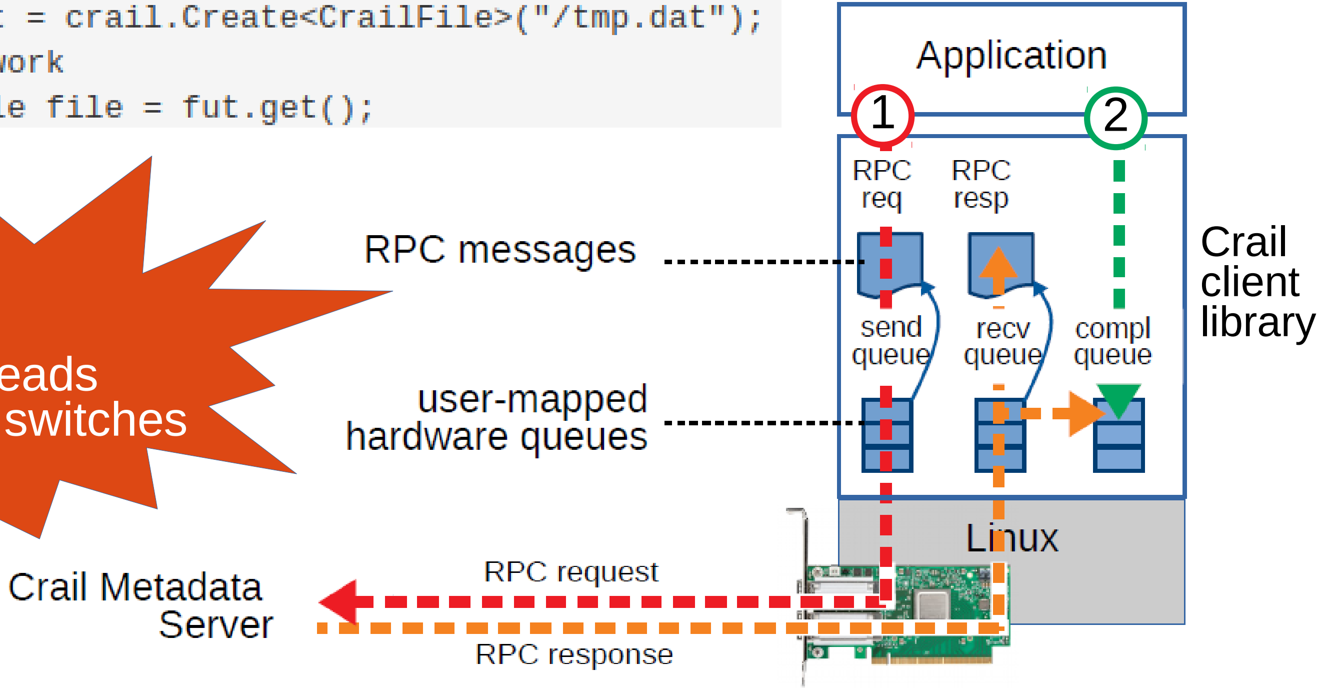
```
1  auto fut = crail.Create<CrailFile>("/tmp.dat");
   //..do work
2  CrailFile file = fut.get();
```



User-Level I/O: Metadata

```
1  auto fut = crail.Create<CrailFile>("/tmp.dat");
   //..do work
2  CrailFile file = fut.get();
```

No threads
No context switches



RPC messages

user-mapped hardware queues

Crail Metadata Server

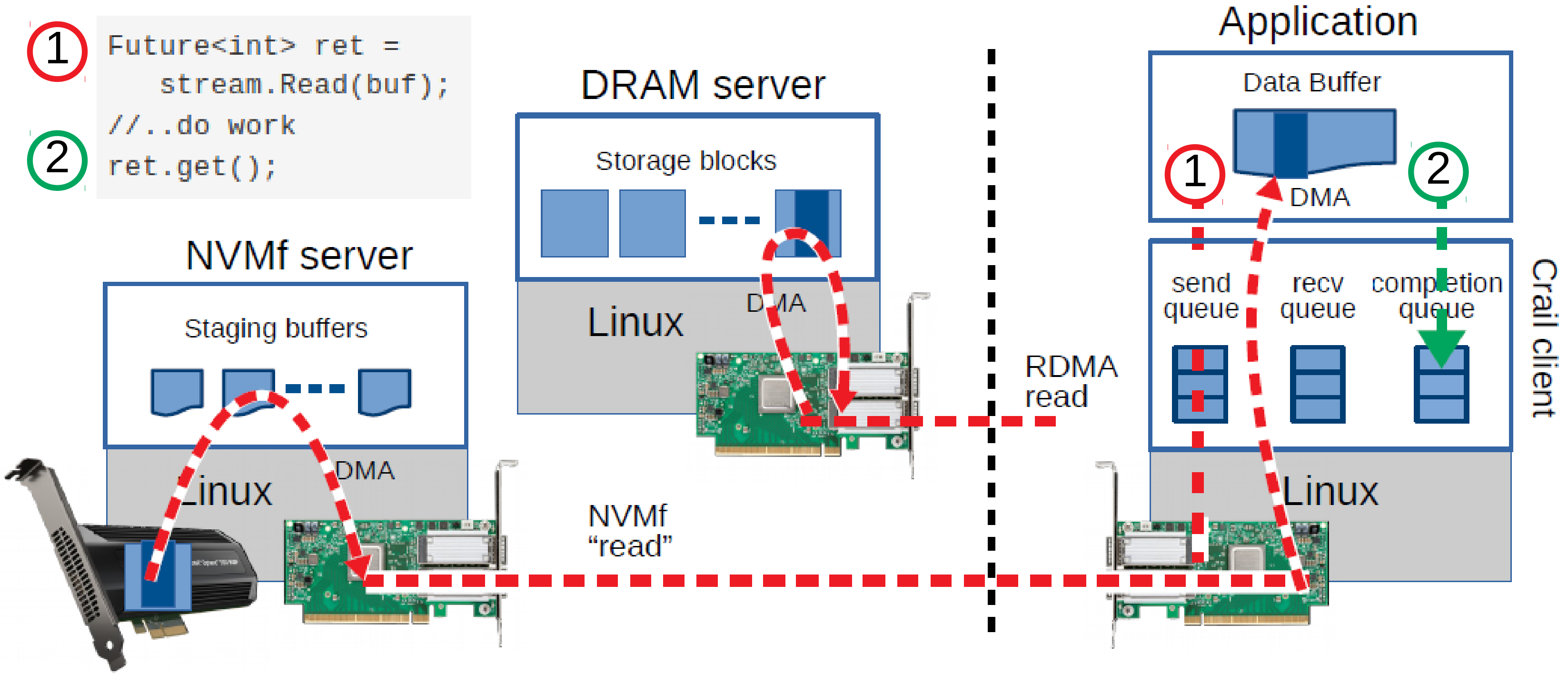
User-Level I/O: Data

1

```
Future<int> ret =
    stream.Read(buf);
//..do work
```

2

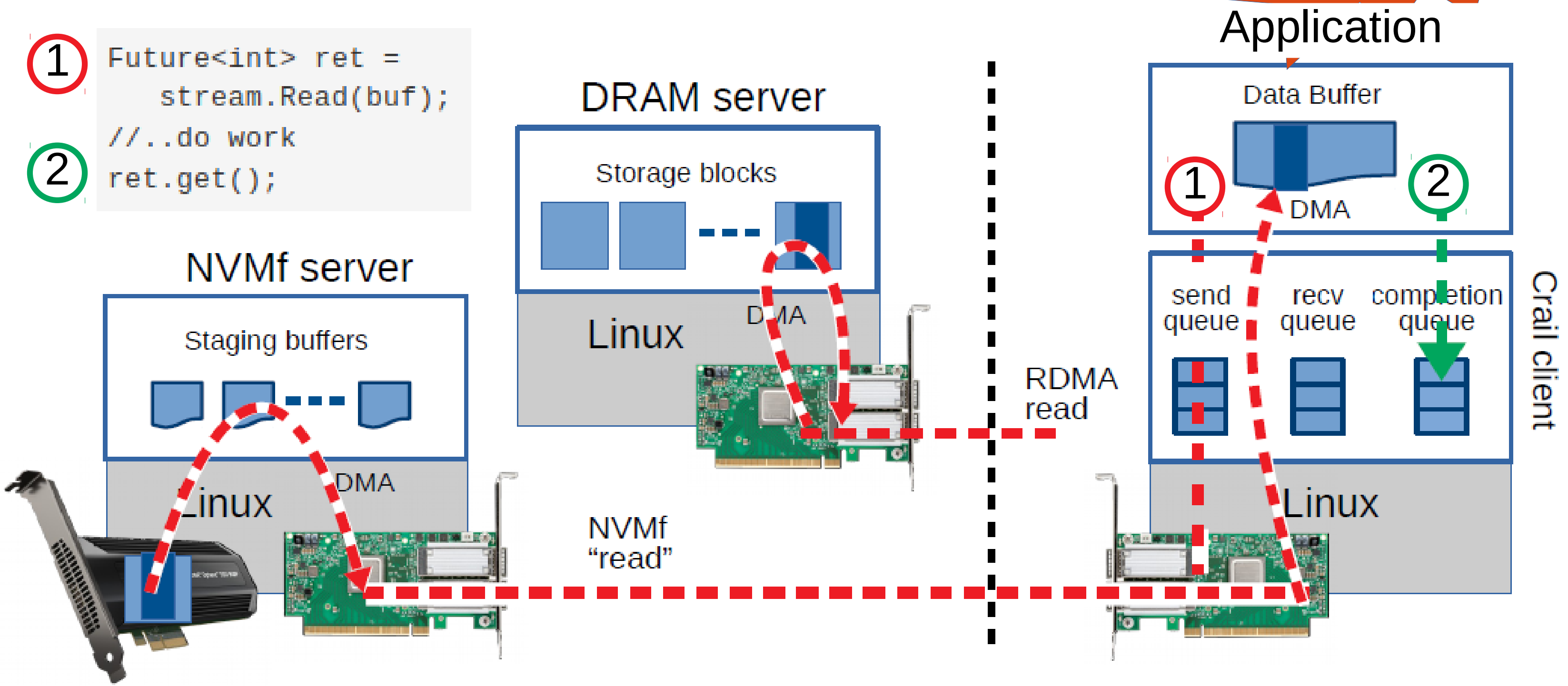
```
ret.get();
```



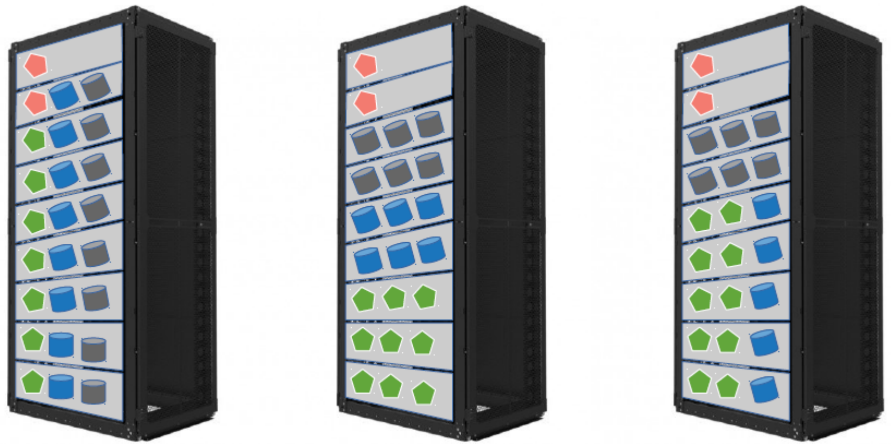
User-Level I/O: Data

zero-copy, transfer only data that is requested

```
Future<int> ret =
    stream.Read(buf);
//..do work
ret.get();
```



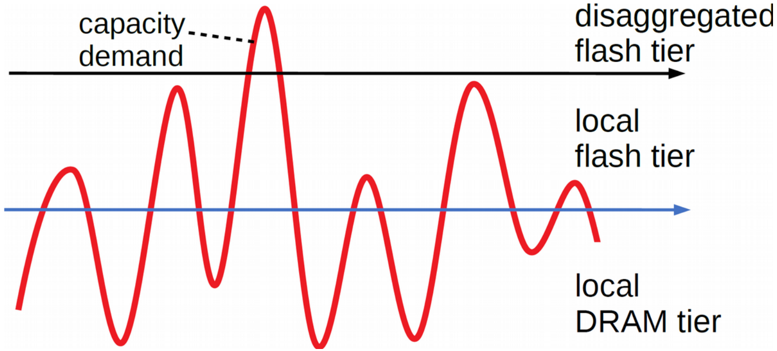
Crail Deployment Modes



compute/storage
co-located

storage
disaggregation

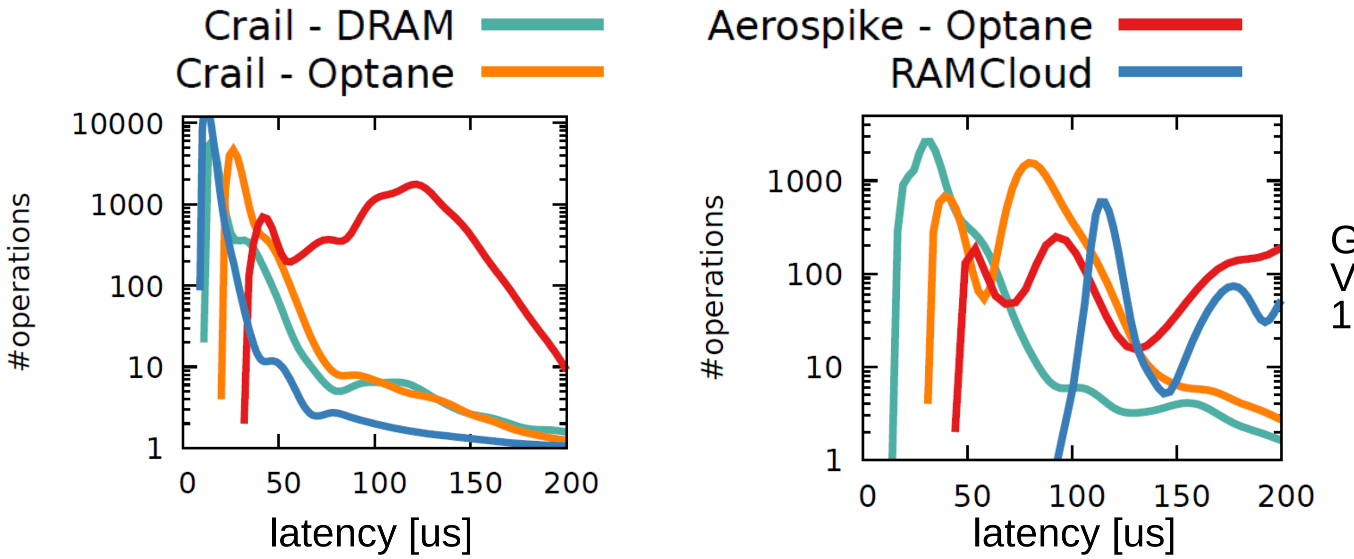
flash storage
disaggregation



- ⬟ Metadata server
- ⛁ Flash storage server
- ⛁ DRAM storage server
- ⬟ Application compute

YCSB KeyValue Workload

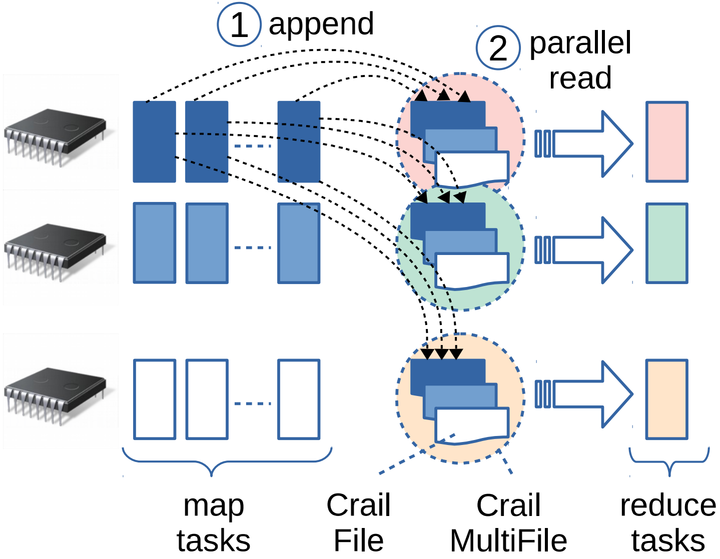
GET
Value size:
1KB

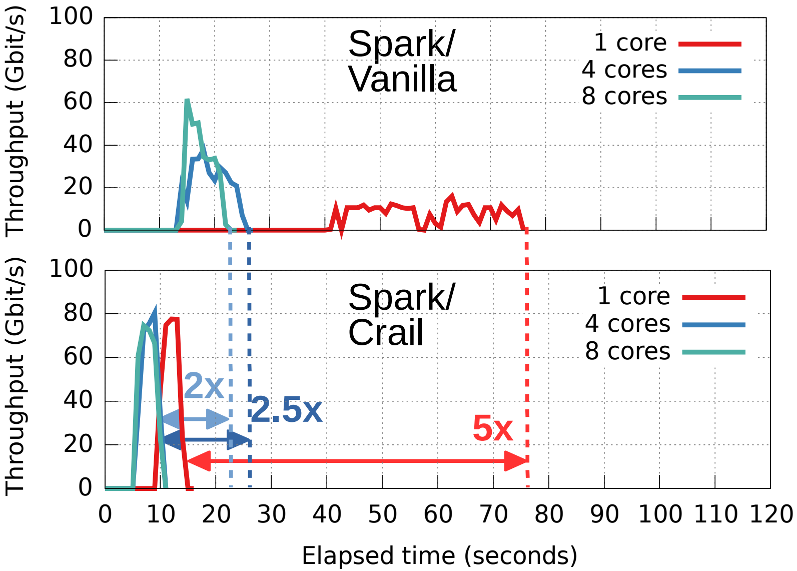


GET
Value size:
100KB

Crail offers Get latencies of ~12us and 30us for DRAM and NVM for 100 byte KV pairs
Crail offers Get latencies of ~30us and 40us for DRAM and NVM for 1000 byte KV pairs

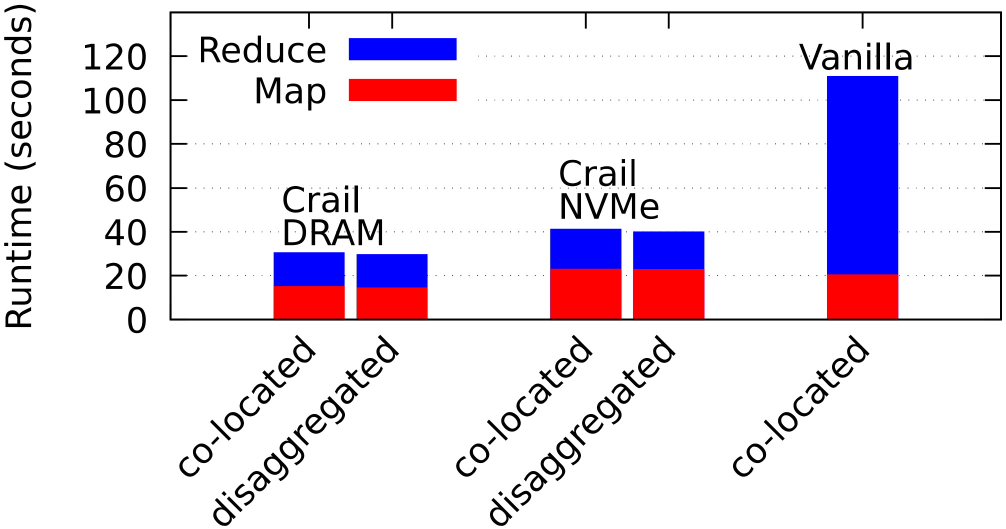
Spark GroupBy (80M keys, 4K)





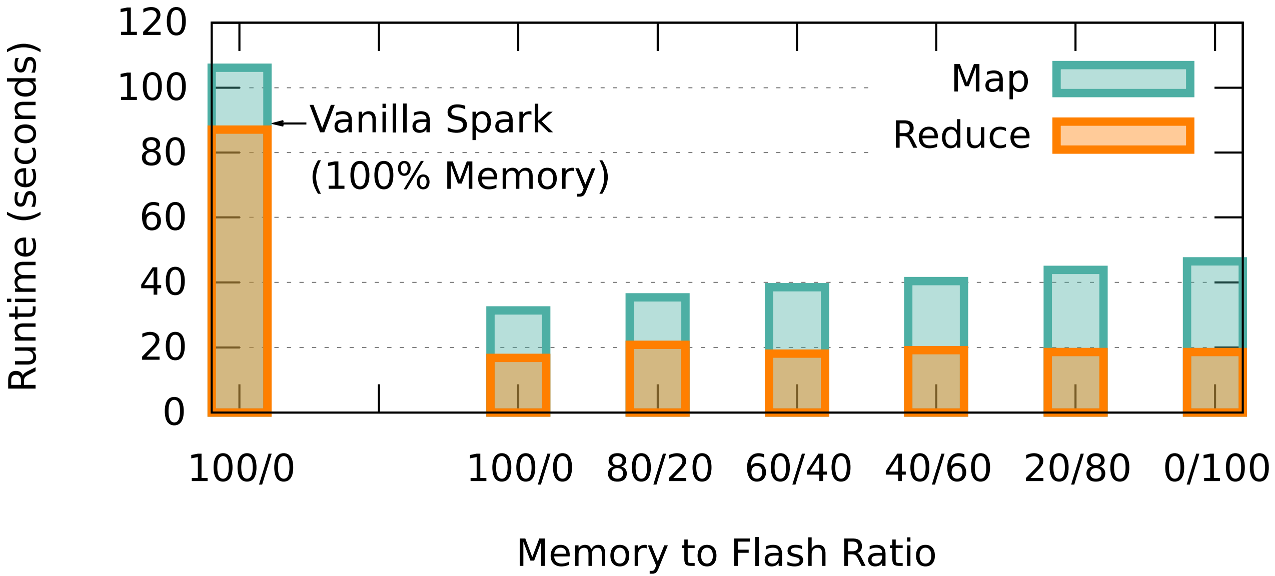
Spark shuffling via Crail on a single core is 2x faster
than vanilla Spark on 8 cores per executor (8 executors)

DRAM & Flash Disaggregation



Crail enables disaggregation of temporary data at no cost

DRAM/Flash Tiering



Using flash only increases the sorting time by around 48%

Conclusions

- Apache Crail: Fast distributed “tmp”
  - User-level I/O
  - Storage disaggregation
  - Memory/flash convergence
- Applications
  - Intra-job scratch space (shuffle, broadcast, etc.)
  - Multi-job pipelines
- Coming soon
  - Native Crail (C++)
  - Tensorflow-Crail