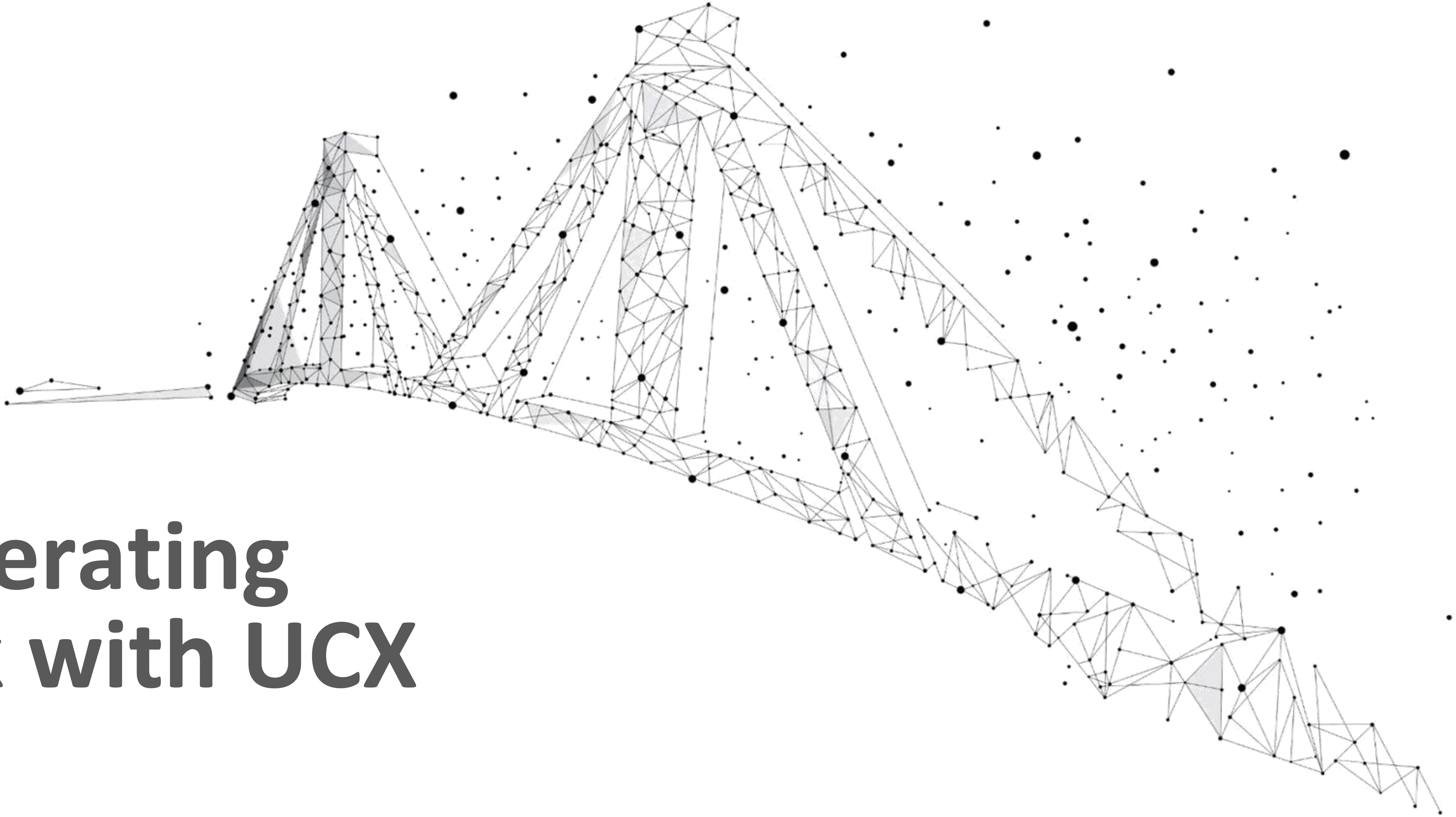


Accelerating Spark with UCX

Dec 2019

Unified Communication X (UCX) - high performance communication layer library (1/2)

Unified API

Applications driven, simple, extendable, HW-agnostic

Focus on performance

Fast, scalable, highly optimized low latency high bandwidth messaging framework

Production quality

Multi-tier testing, used by top Mellanox customers in production

Open source

Collaboration between industry, laboratories, and academia

Innovation

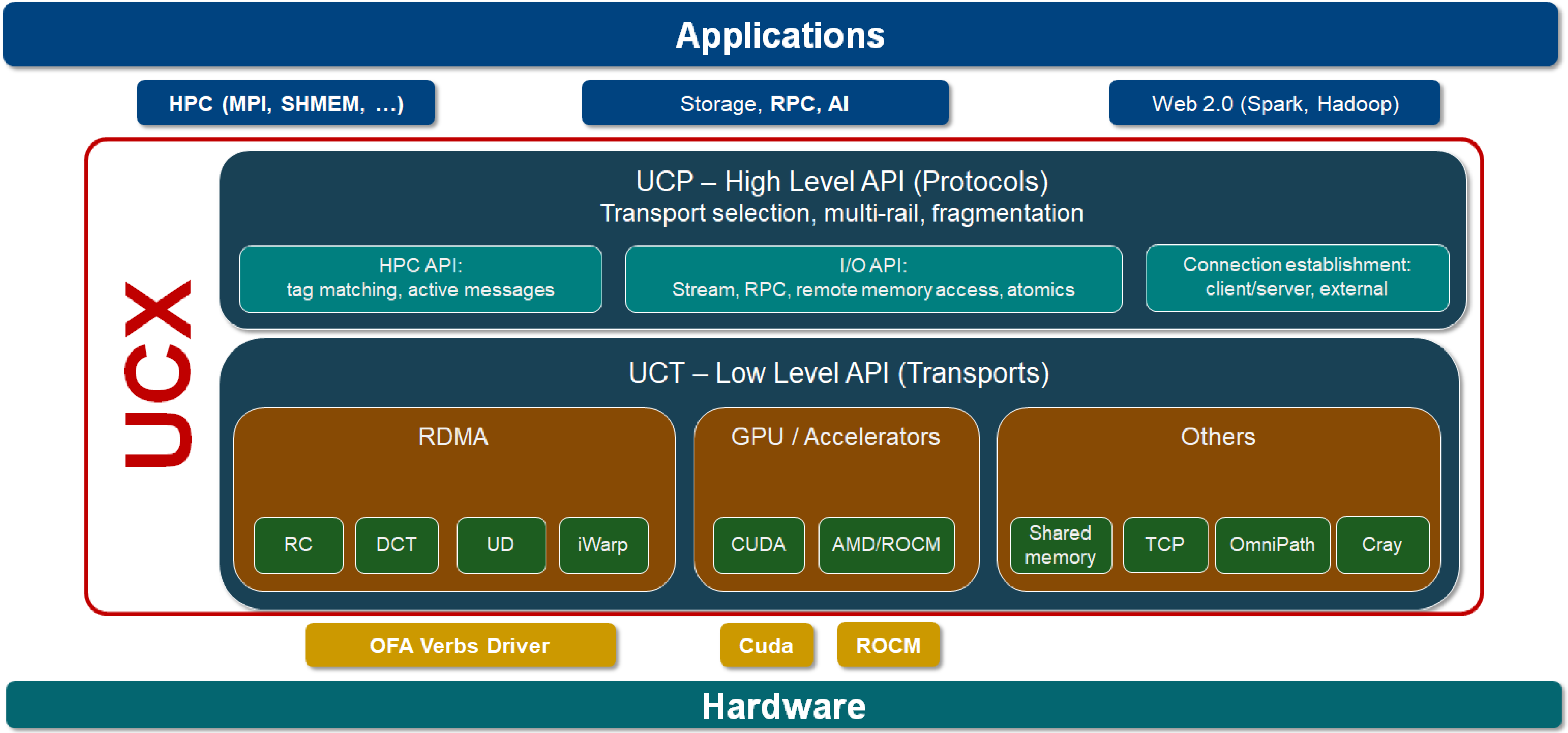
Concepts and ideas from research in academia and industry

Multi arch/transports

RoCE, InfiniBand, Cray, TCP, shared memory, GPUs, x86, ARM, POWER

Co-design of Network APIs

Unified Communication X (UCX) - high performance communication layer library (2/2)



JUCX – java bindings for UCX

- Transport abstraction - implemented on top of UCP layer
  - Can run over different types of transports (Shared memory, Infiniband/RoCE, Cuda,…)
- Ease of use API wrapper over high level UCP layer
- Supported operations: non blocking send/recv/put/get

JUCX API example

1. Instantiate ucp context:

```
UcpConetxt context = new UcpContext(new UcpParams().requestRmaFeature());
```

2. Instantiate ucp worker:

```
UcpWorker worker = context.newWorker(new UcpWorkerParams());
```

3. Instantiate ucp endpoint:

```
EndpointParams epp   = new UcpEndpointParams().setSocketAddress(InetSocketAddress("1.2.3.4:1234")
UcpEndpoint endpoint = worker.newEndpoint(epp);
```

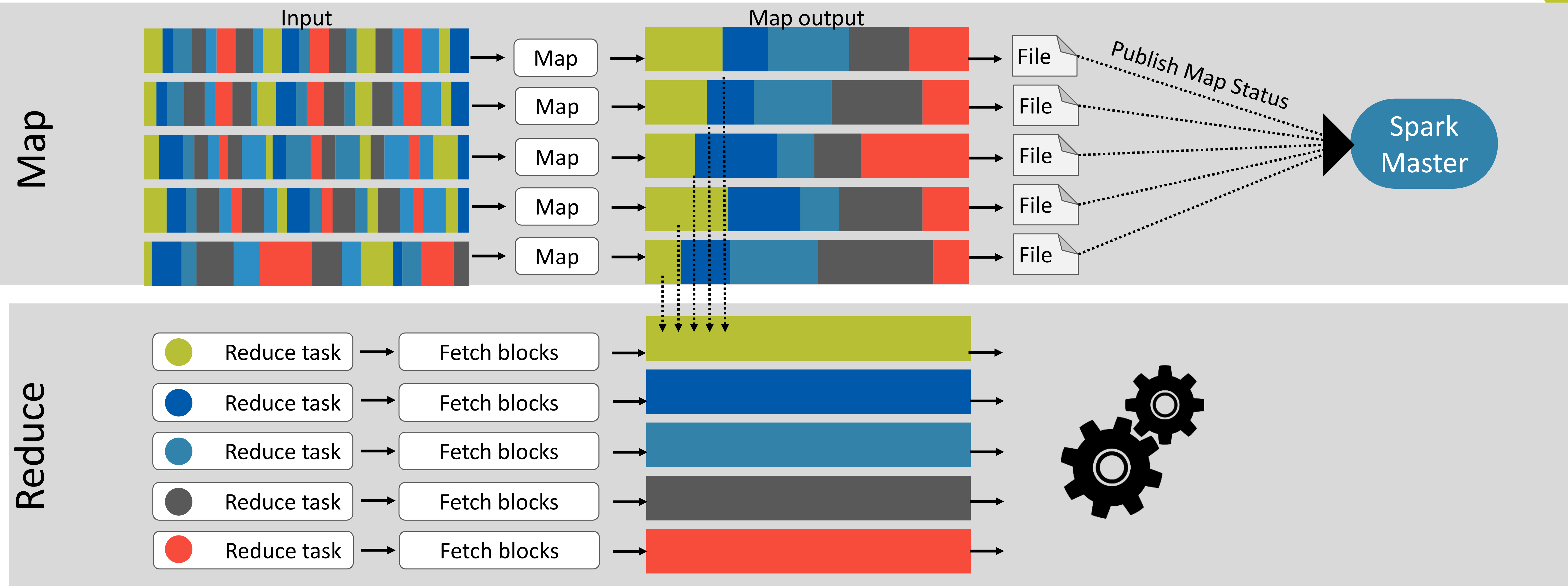
4. Perform get/put/send/recv operation on endpoint:

```
UcxRequest request = endpoint.getNonBlocking(remoteAddress, remoteKey, localBuffer);
```

5. Progress request until it's completed:

```
while(!request.isCompleted()) {
   worker.progress();
}
```

Spark's Shuffle Basics



The Cost of Shuffling

- Shuffling is very expensive in terms of CPU, RAM, disk and network Ios
- Spark users try to avoid shuffles as much as they can
- Speedy shuffles can relieve developers of such concerns, and simplify applications

SparkUCX Shuffle Plugin

https://github.com/openucx/sparkucx

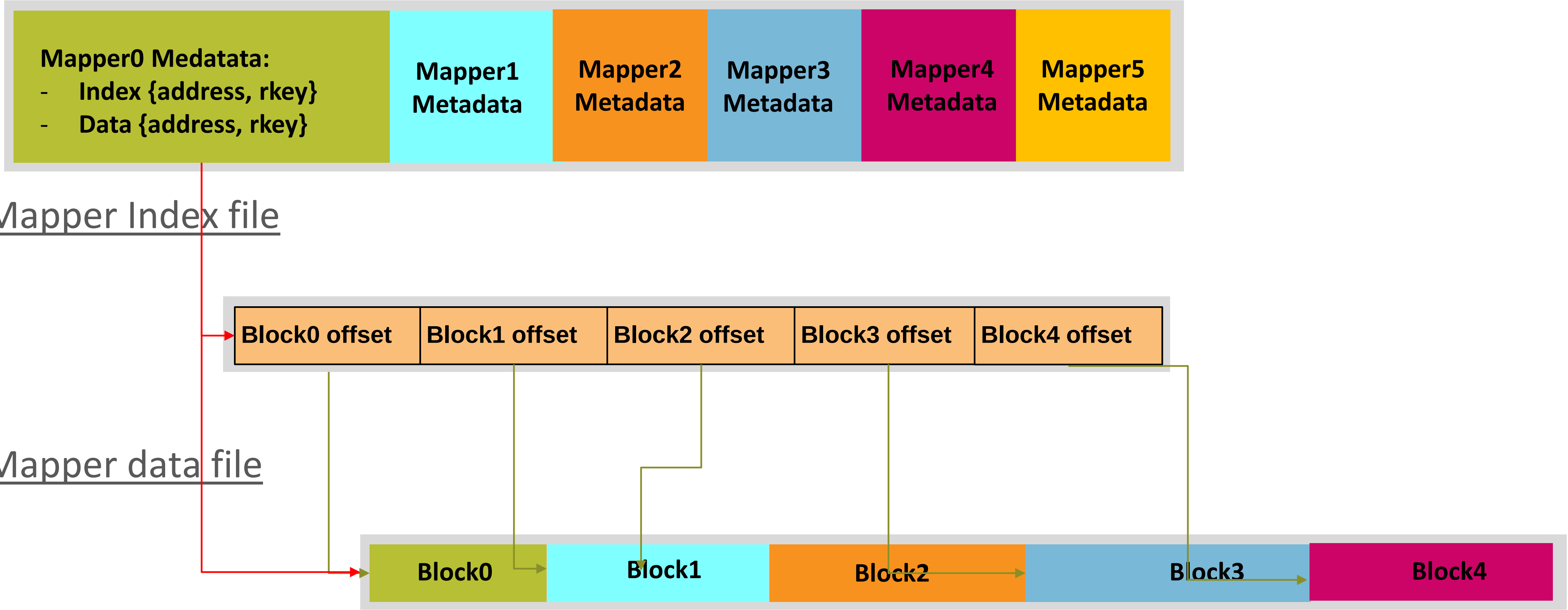
ShuffleManager Plugin

- Spark allows for external implementations of ShuffleManagers to be plugged in
  - Configurable per-job using: “spark.shuffle.manager”
- Interface allows proprietary implementations of Shuffle Writers and Readers, and essentially defers the entire Shuffle process to the new component

- SparkUCX utilizes this interface to introduce RDMA in the Shuffle process

SparkUCX memory layout object model

- Driver global metadata buffer
- Mapper Index file
- Mapper data file



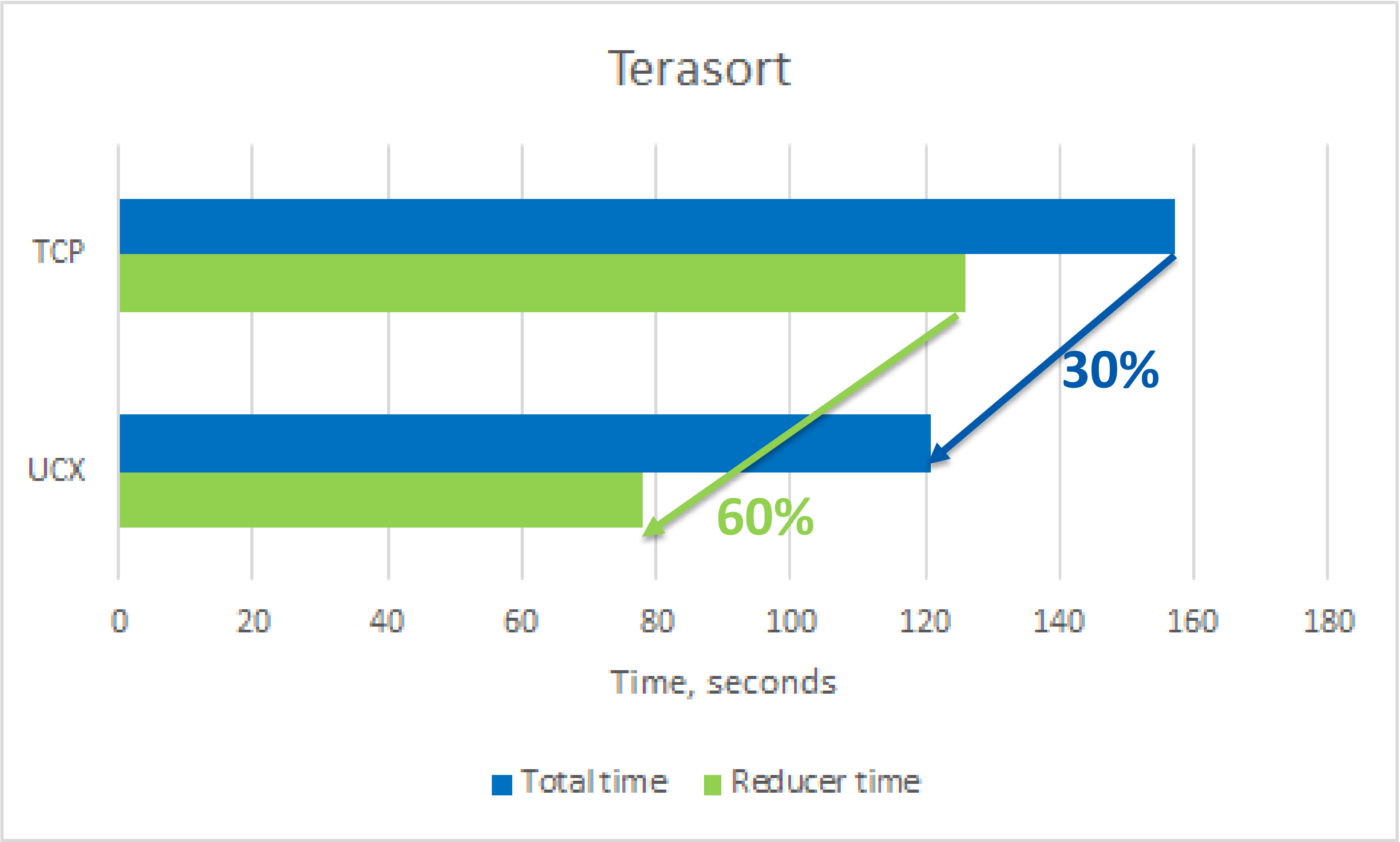
SparkUCX operation flow

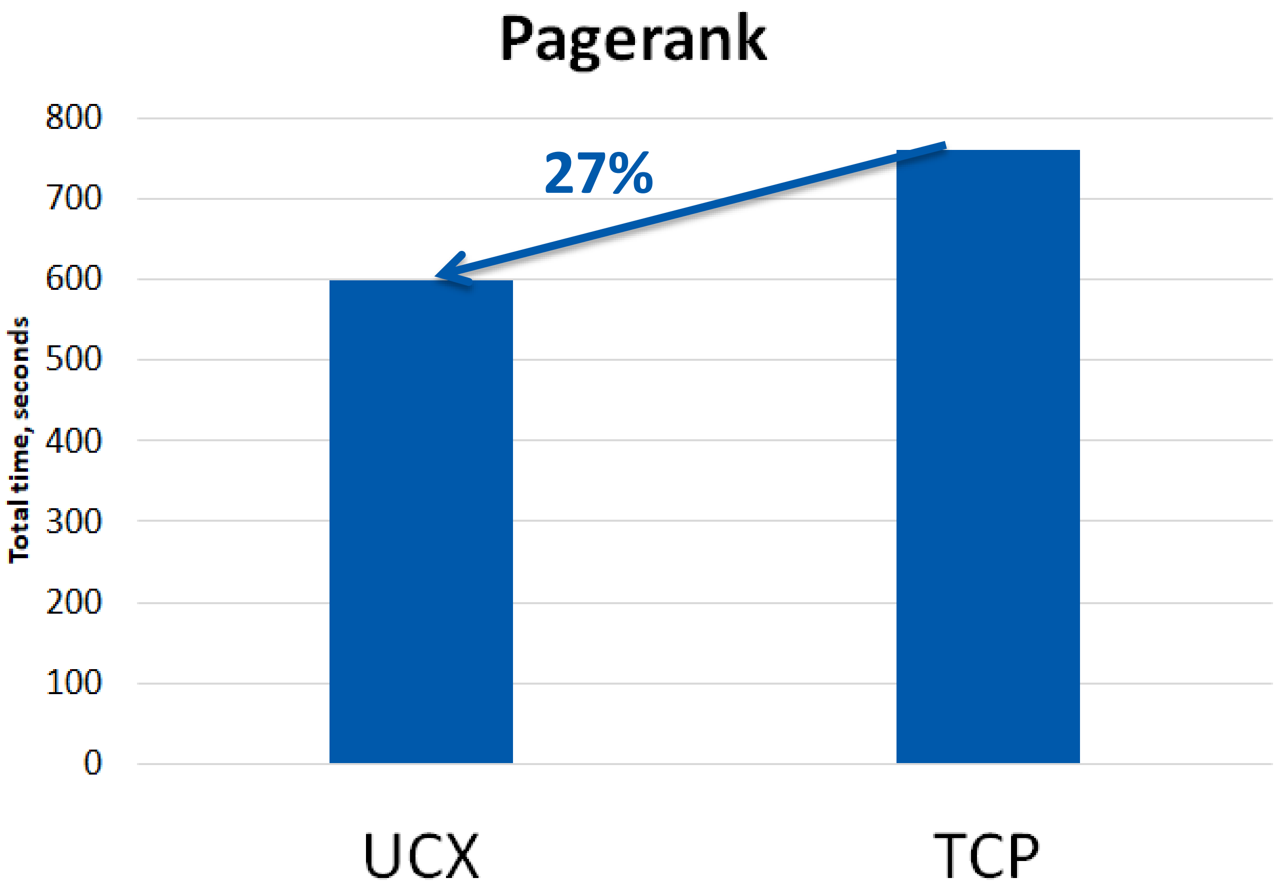
- **Initialization**:
  Spark driver allocates global metadata buffer per shuffle stage, to hold addresses and memory keys of data and index files on mappers.
- **Mapper phase**:
  - mmap() and register index and data files
  - Publish {address, rkey} to driver metadata buffer (*ucp_put*).
- **Reduce phase**:
  - Fetch metadata from driver (*ucp_get*)
  - For each block:
    - Fetch offset in data file, from index file (*ucp_get*).
    - Fetch block contents from data file (*ucp_get*).

Benchmarking eco-system

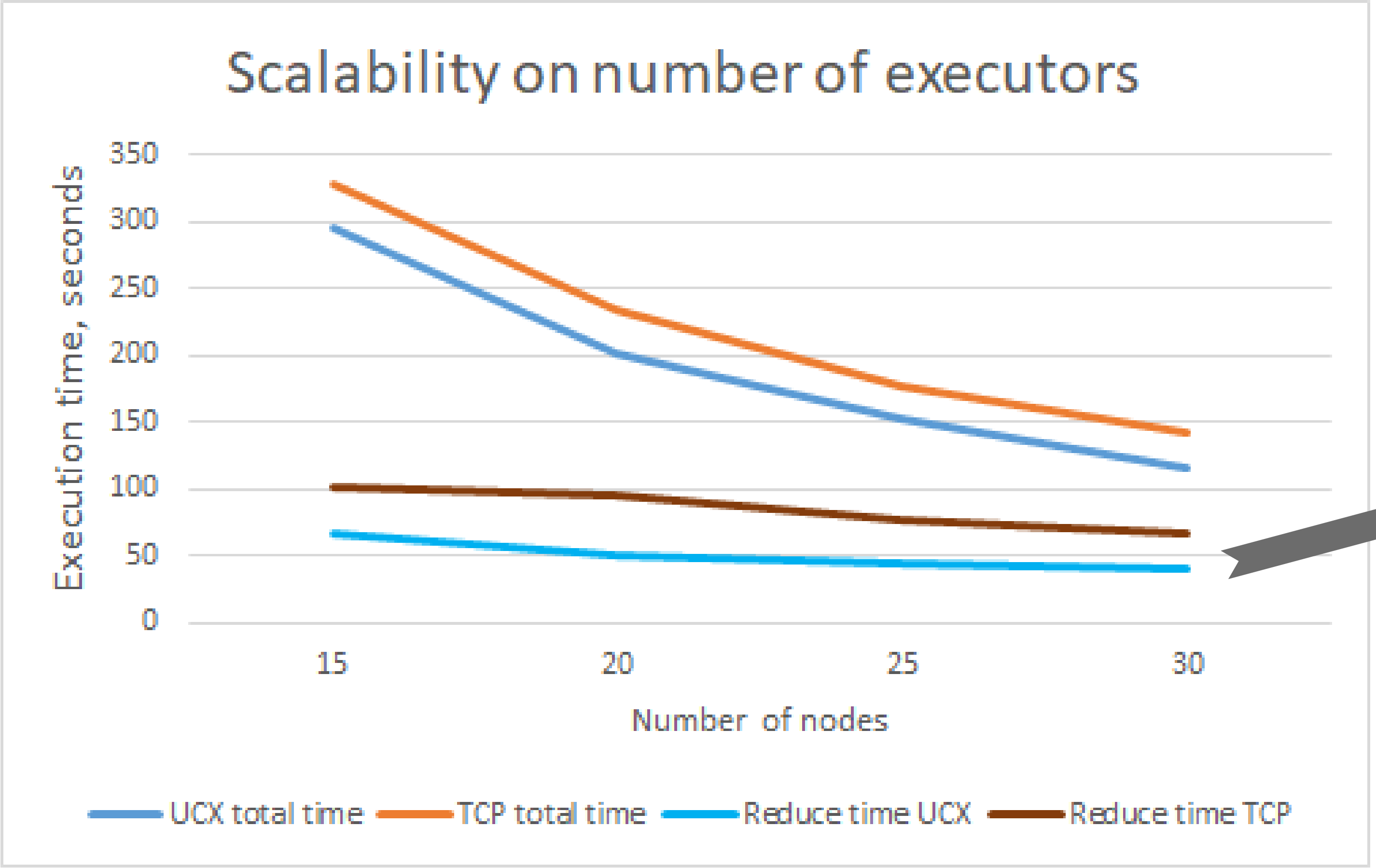
- Benchmarks: Terasort + Pagerank
  - https://github.com/zrlio/crail-spark-terasort
  - https://github.com/Intel-bigdata/HiBench
- Terasort:
  - 1.2 TB input, 10K mappers, 15k reducers
- Pagerank:
  - Bigdata Hibench workload (600 Gb), 5K mappers, 15K reducers
- 15 nodes: Broadwell @ 2.60GHz, 250GB RAM, 500GB HDD
- ConnectX-5: Infiniband: 100G EDR. TCP device: **IPoIB 100G**
- Red Hat Enterprise Linux Server release 7.5 (Maipo) (kernel: 3.10.0-862.el7.x86_64)
- MLNX_OFED_LINUX-4.6-1.0.1.1.
- Spark-2.4.3, Hadoop-2.9.2, UCX v1.7.0

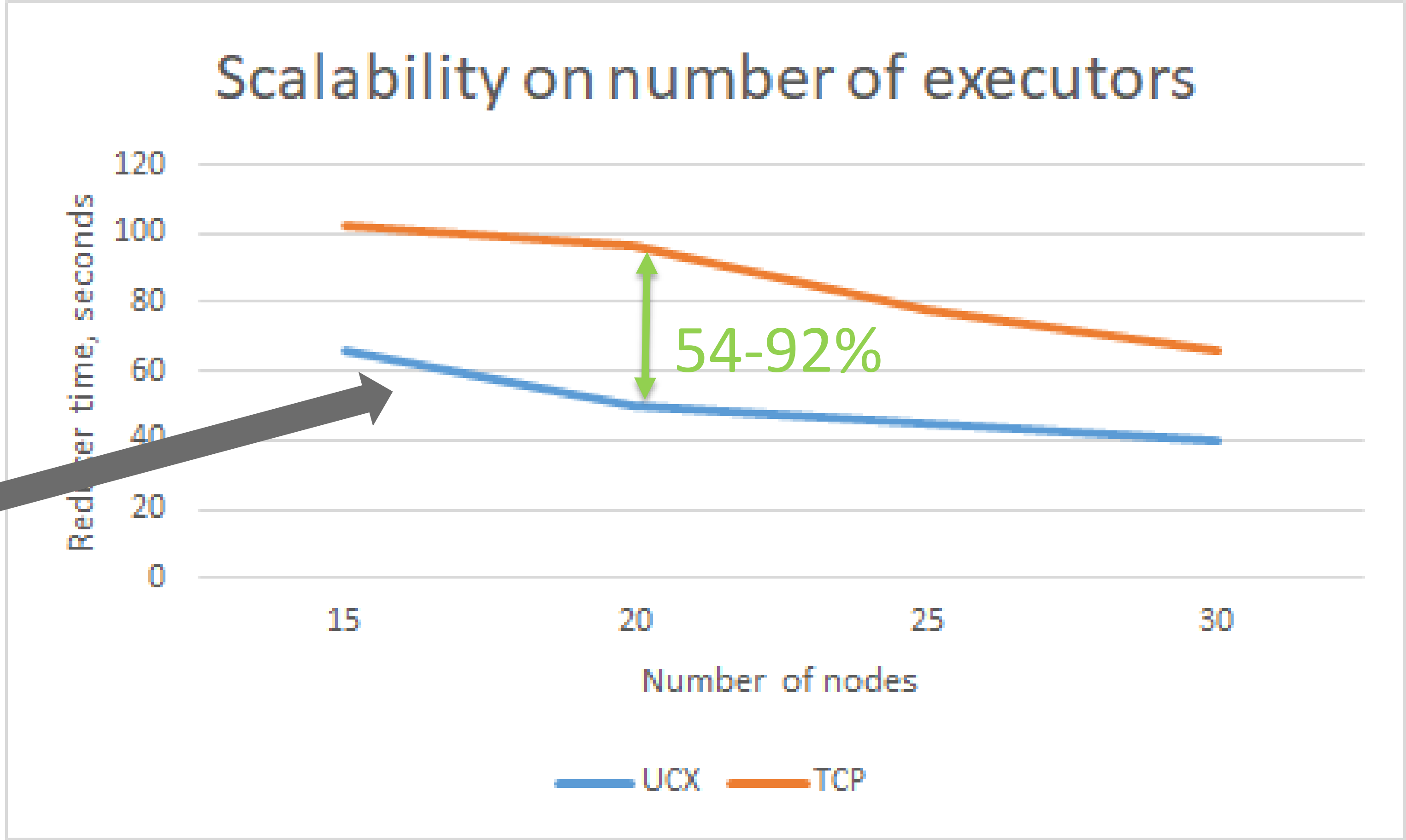
TCP vs UCX performance (1/3)



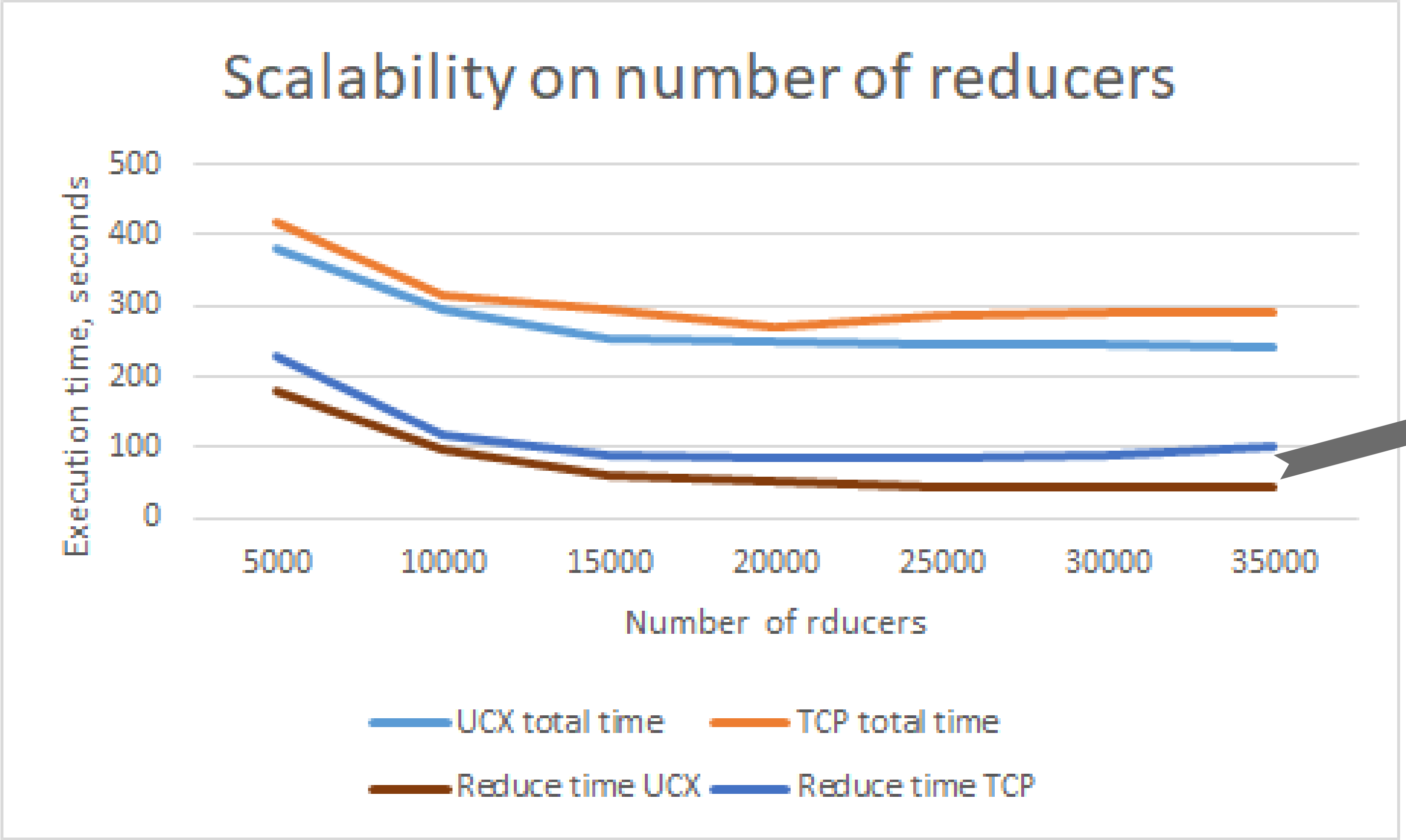


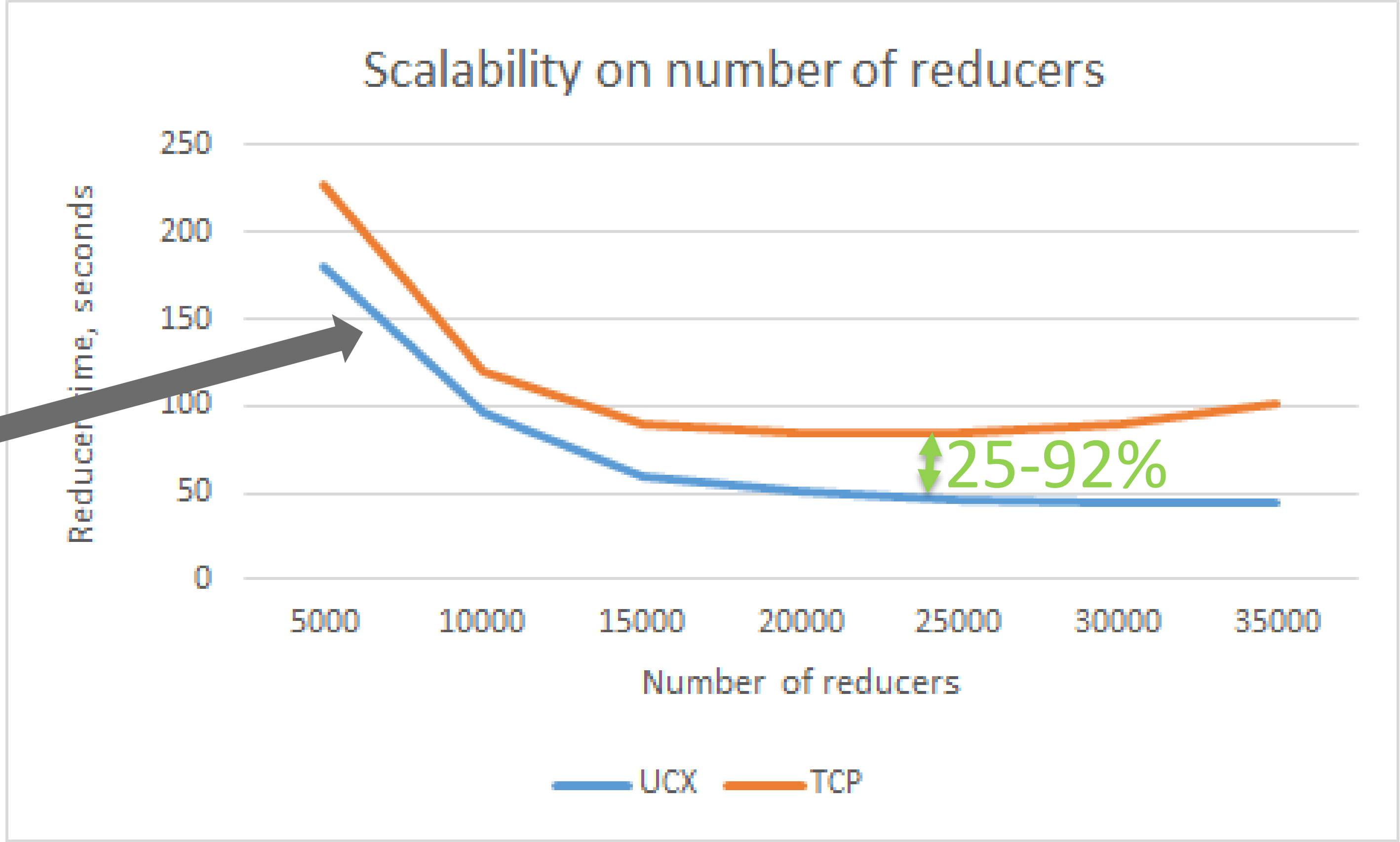
TCP vs UCX Terasort scalability (2/3)





TCP vs UCX Terasort scalability (3/3)



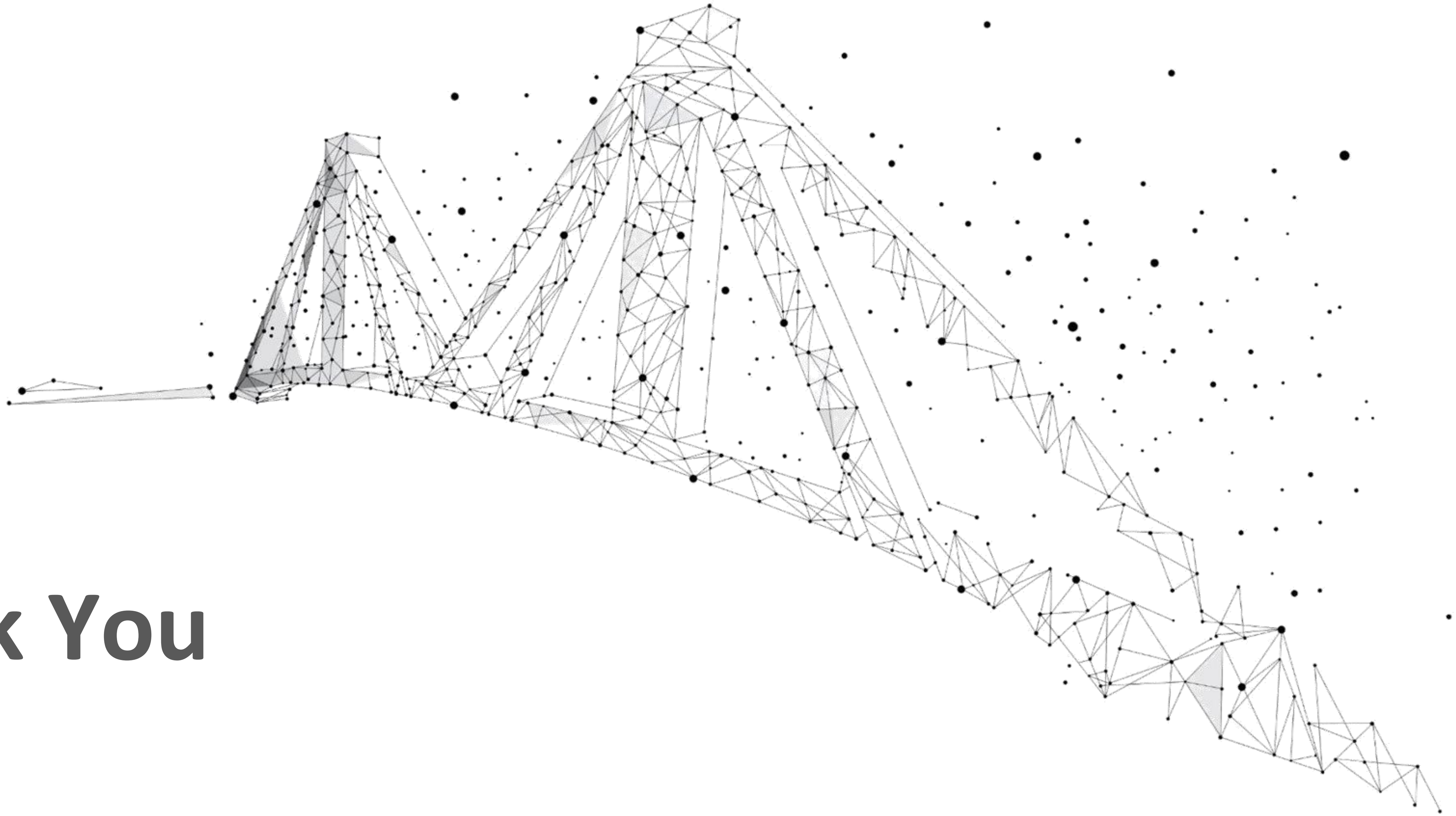


SparkRDMA vs. SparkUCX

SparkRDMA	SparkUCX
Based on abandoned IBM DiSNi verbs package	Based on UCX high-level API which has dedicated R&D and wide community. Production grade.
Supports IB/ROCE with RC only	Supports IB, ROCE with RC/DC/Shared memory, and TCP as fallback
Not scalable, CQ and progress thread per connection	Scalable: CQ per executor
Communications progress on dedicated thread which consumes CPU %	Communications are initiated from application threads and progressed asynchronously by hardware
RDMA protocols are implemented in Java	Based on standard UCX API and protocols hiding complexity of RDMA
Registering each data block with different key	Registering all data as single chunk
Showed improved vs. **worst** TCP numbers	Showed improved vs. **best** TCP numbers

Future work

- Optimizations on multiple benchmarks (TPC-DS, TPC-H, etc.)
- Support shuffle data larger then memory
- GPU memory support
- HDFS optimization with UCX



Thank You