UCI University of California, Irvine

Skyway: Connecting Managed Heaps in Distributed Big Data Systems

Khanh Nguyen, Lu Fang, Christian Navasca,
Harry Xu, Brian Demsky
University of California, Irvine

Shan Lu
University of Chicago

BIG DATA

BIG DATA

BIG DATA

cloudera® IMPALA

Hortonworks

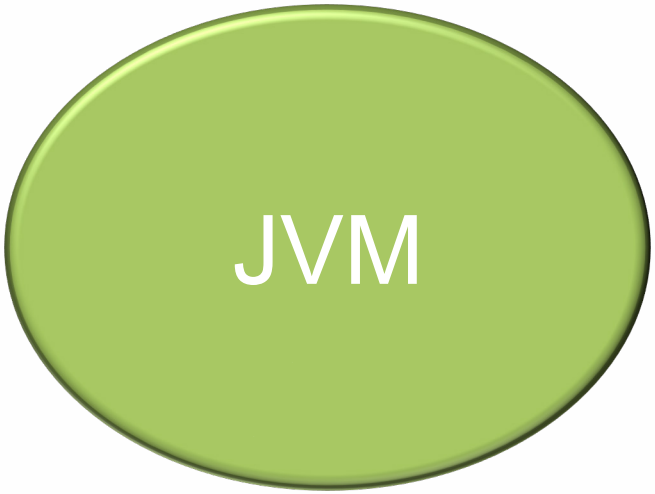
databricks®


Java™


Scala


Java™

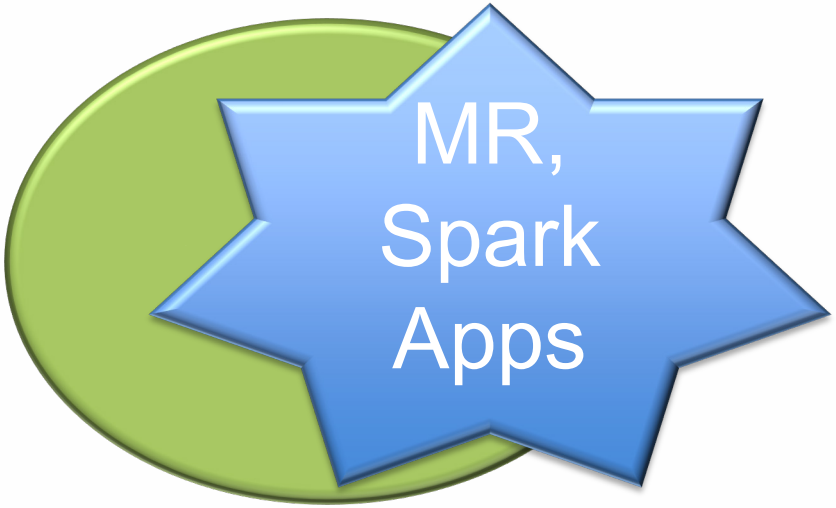

Scala

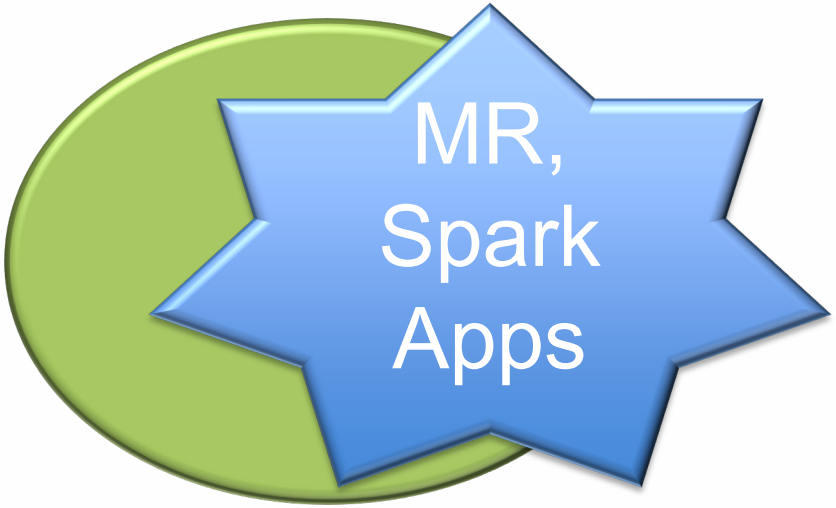

JVM


MR,
Spark
Apps


Java™


Scala


MR,
Spark
Apps



The managed runtime is costly

The managed runtime is costly

The managed runtime is costly

The managed runtime is costly

The managed runtime is less costly

outDataset

outDataset

serialization

```
OutputStream out =
  Shuffler.GetOutputStream(receiver_id);

for (Object o: outDataset) {
  out.writeObject(o);
}
```

outDataset

serialization

```
OutputStream out =
  Shuffler.GetOutputStream(receiver_id);

for (Object o: outDataset) {
  out.writeObject(o);
}
```

```
InputStream in =
  Shuffler.GetInputStream(sender_id);

while (in.hasData()) {
  Object o = in.readObject();
  inDataset.store(o)
}
```

deserialization

serialization

```
OutputStream out =
  Shuffler.GetOutputStream(receiver_id);

for (Object o: outDataset) {
  out.writeObject(o);
}
```

```
InputStream in =
  Shuffler.GetInputStream(sender_id);

while (in.hasData()) {
  Object o = in.readObject();
  inDataset.store(o)
}
```

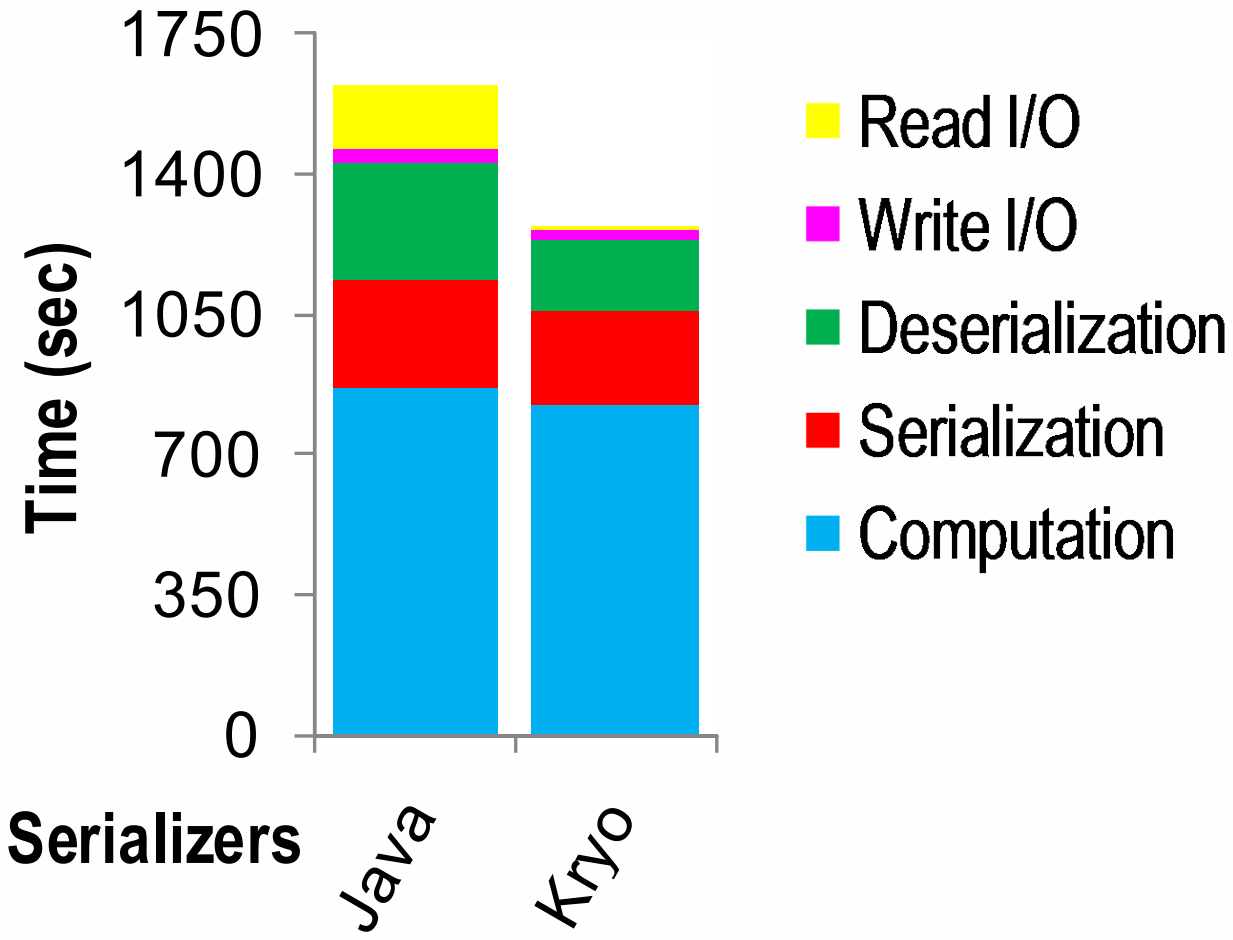
deserialization

inDataset

serialization

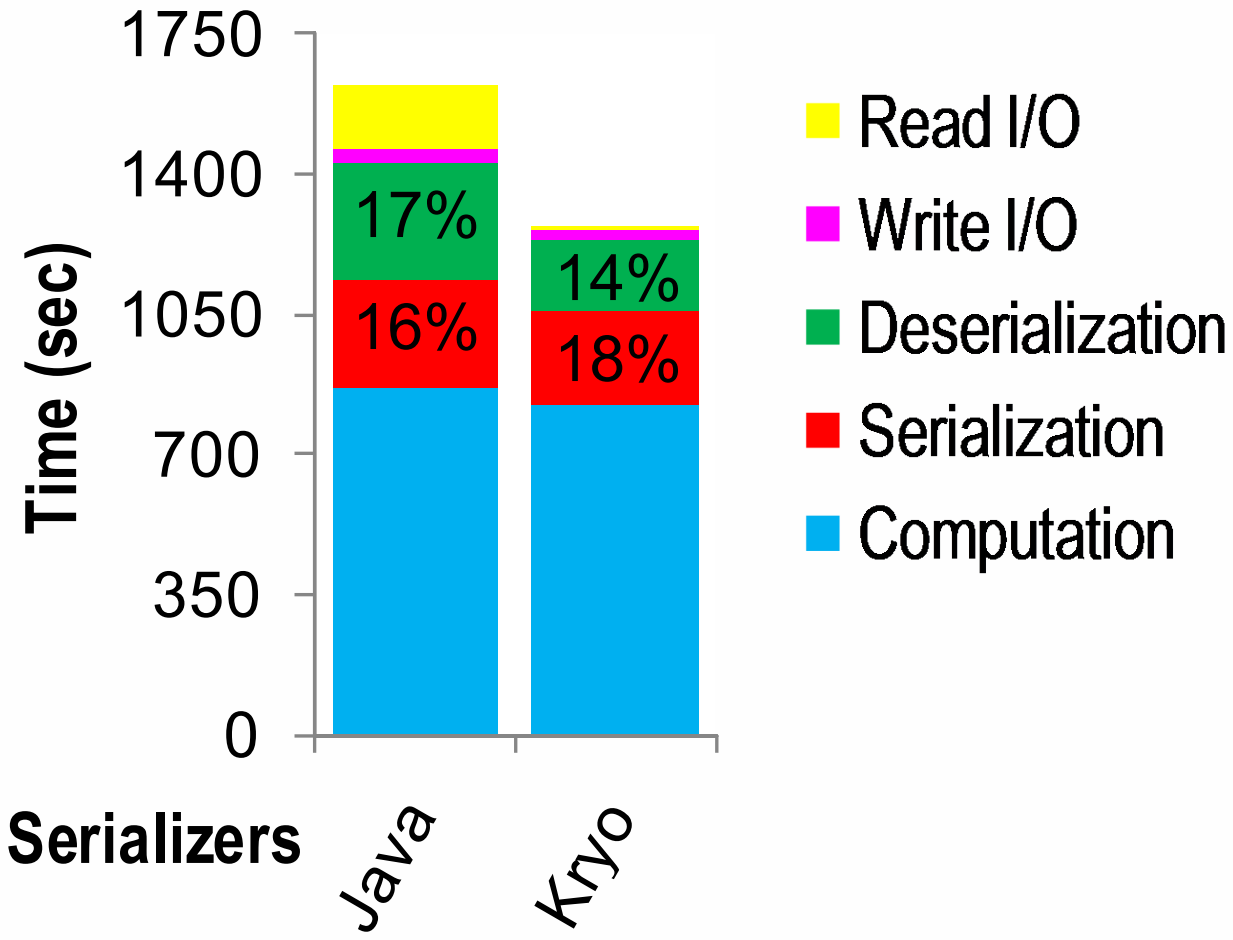
```
OutputStream out =
  Shuffler.GetOutputStream(receiver_id);

for (Object o: outDataset) {
  out.writeObject(o);
}
```

Data transfer costs



TriangleCounting over LiveJournal on Spark 2.1.0 with 3 slaves

Data transfer costs



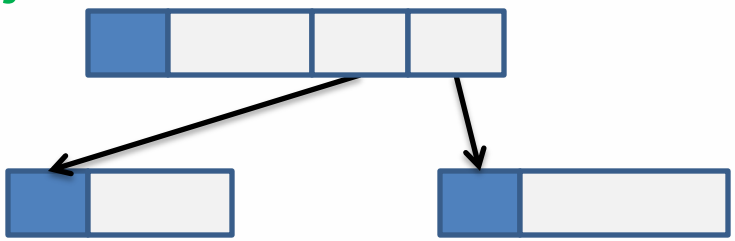
TriangleCounting over LiveJournal on Spark 2.1.0 with 3 slaves

Data transfer

Sender

Receiver

Object

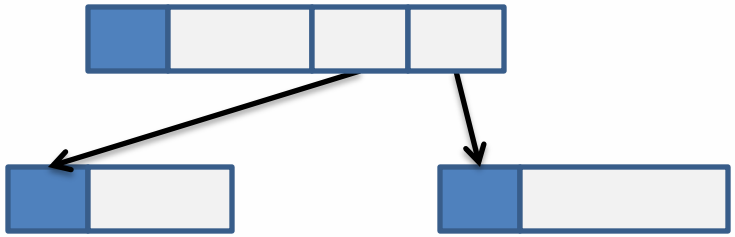


Data transfer

Sender

Receiver

Object



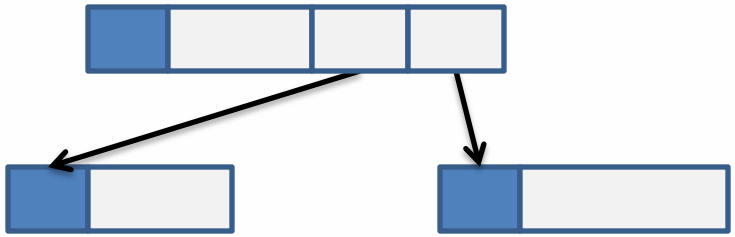
Serialization

Data transfer

Sender

Receiver

Object



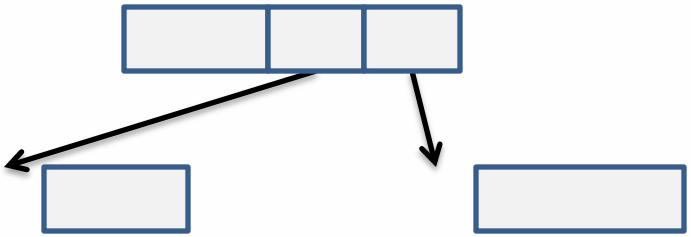

```
Reflection.
getField
```

Data transfer

Sender

Receiver

Object



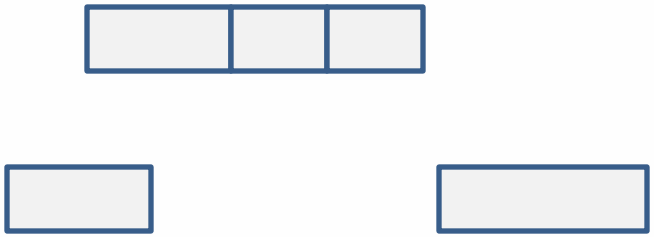

```
Reflection.
getField
```

Data transfer

Sender

Receiver

Object



Serialization

```
Reflection.
getField
```

Data transfer

Sender

Receiver

Object

Serialization

`Reflection.`
`getField`

Binary

Data transfer

Sender

Receiver

Object

Serialization

`Reflection.`
`getField`

Binary

Network

Data transfer

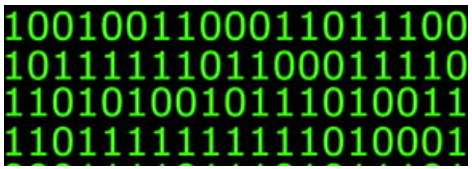
Sender

Receiver

Object

Binary

Network



Data transfer

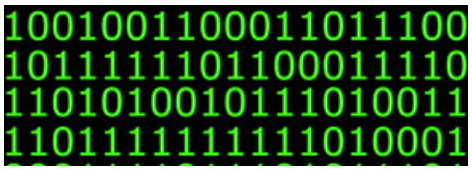
Sender

Receiver

Object

Deserialization

Binary



Data transfer

Sender

Receiver

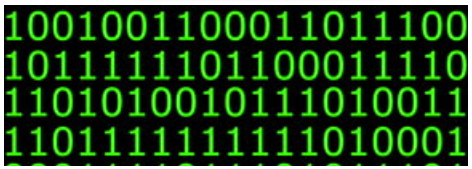
Object

Reflection.
allocate

Reflection.
setField

Deserialization

Binary



Data transfer

Sender

Receiver

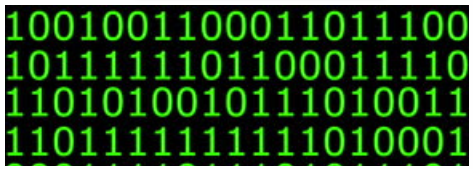
Object

Reflection.
allocate

Reflection.
setField

Deserialization

Binary

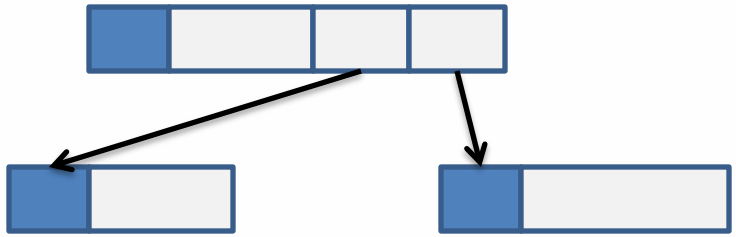


Data transfer

Sender

Receiver

Object

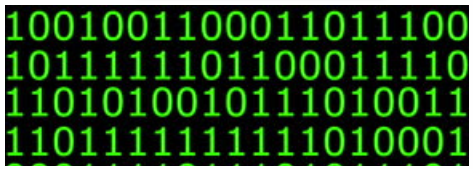


`Reflection.allocate`

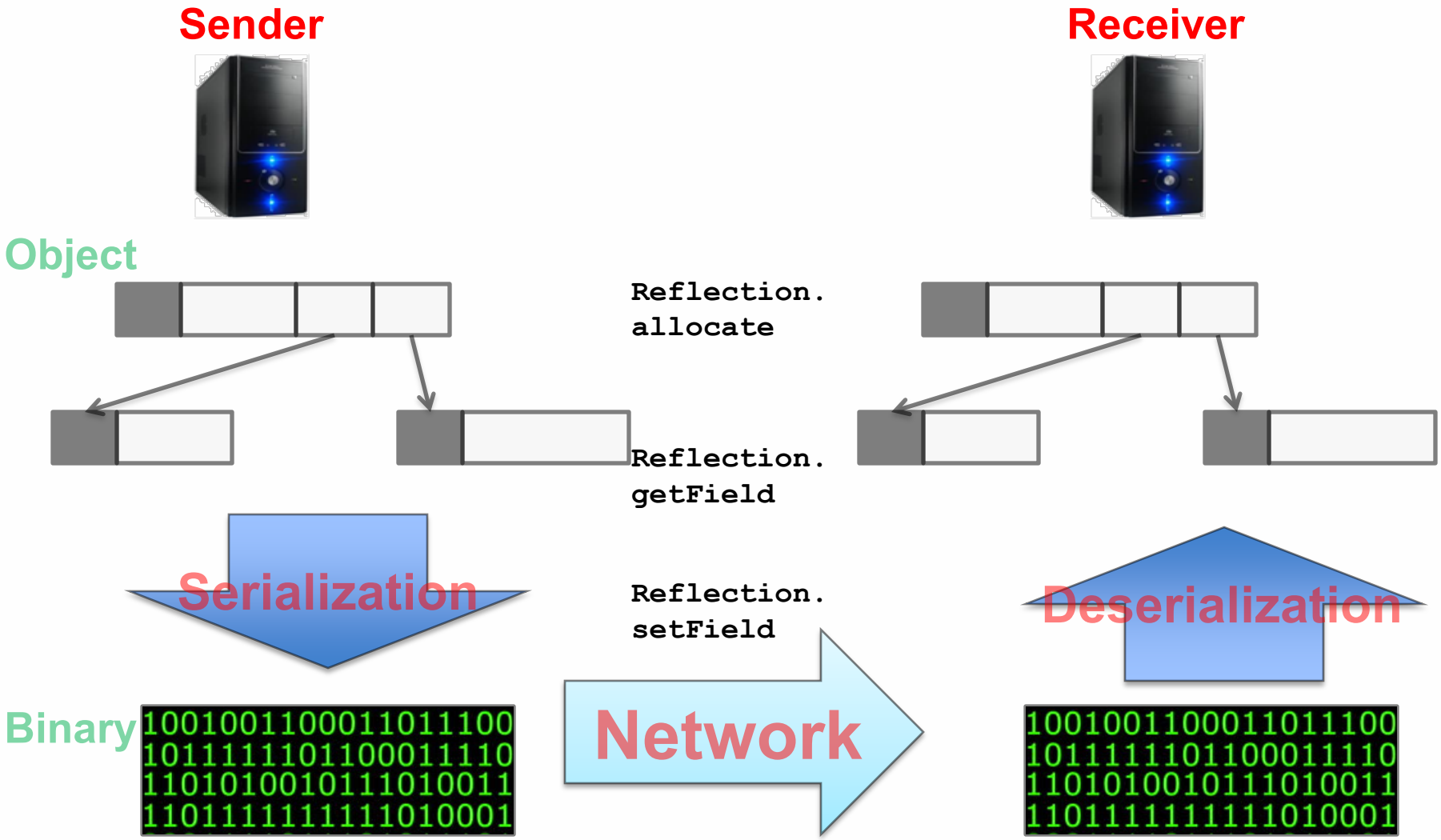
`Reflection.setField`

Deserialization

Binary



Data transfer



Data transfer

Sender

Receiver

WANTED:
a system-level solution


an analogy


an analogy


an analogy


an analogy


an analogy


an analogy

Our solution

Our solution

Sender

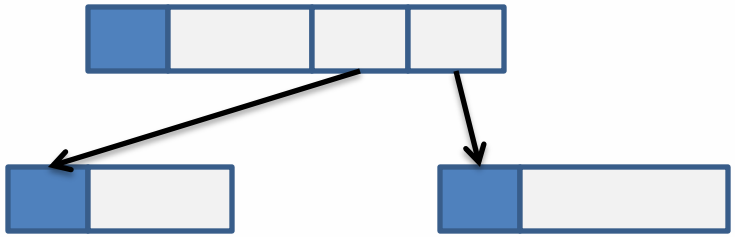
Receiver

Our solution

Sender

Receiver

Object



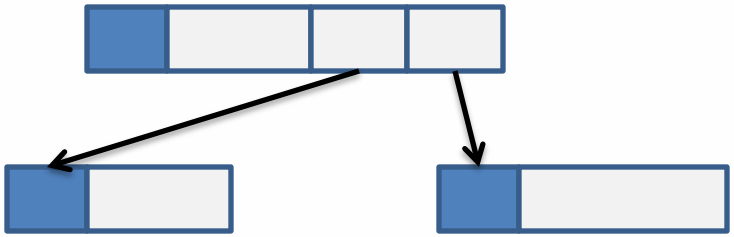
Our solution

Sender

Receiver

Skyway

Object



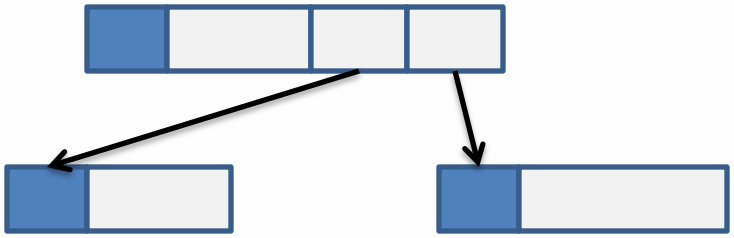
Our solution

Sender

Receiver

Skyway

Object



```
Reflection.      Reflection.      Reflection.
allocate         getField         setField
```

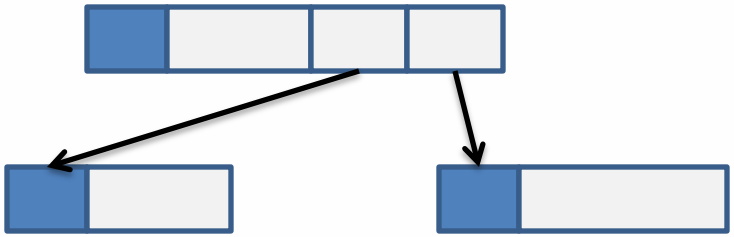
Our solution

Sender

Receiver

Skyway

Object



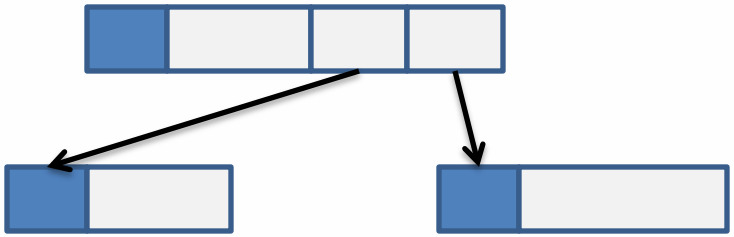
Our solution

Sender

Receiver

Skyway

Object



~~Reflection.allocate~~

~~Reflection.getField~~

~~Reflection.setField~~

Overlapping computation and data transfer

Skyway Overview

Skyway Overview

- Implemented in OpenJDK 8
  - Modified the class loader, the object/heap layout, the Parallel Scavenge GC
- Efficiently handle data transfer:
  - Outperforms 90 serializers
  - Improves Spark by 36% (Java) - 16% (Kryo)
  - Improves Flink by 19%

Challenges

Challenges

1. Type representation

Challenges

1. Type representation
   - Automated global type numbering

Challenges

1. Type representation
   - Automated global type numbering
2. Pointer representation

Challenges

1. Type representation
   - Automated global type numbering
2. Pointer representation
   - Use relative offsets

Challenges

1. Type representation
   - Automated global type numbering
2. Pointer representation
   - Use relative offsets
3. Local JVM adaptation

Challenges

1. Type representation
   - Automated global type numbering
2. Pointer representation
   - Use relative offsets
3. Local JVM adaptation
   - Visible for garbage collection

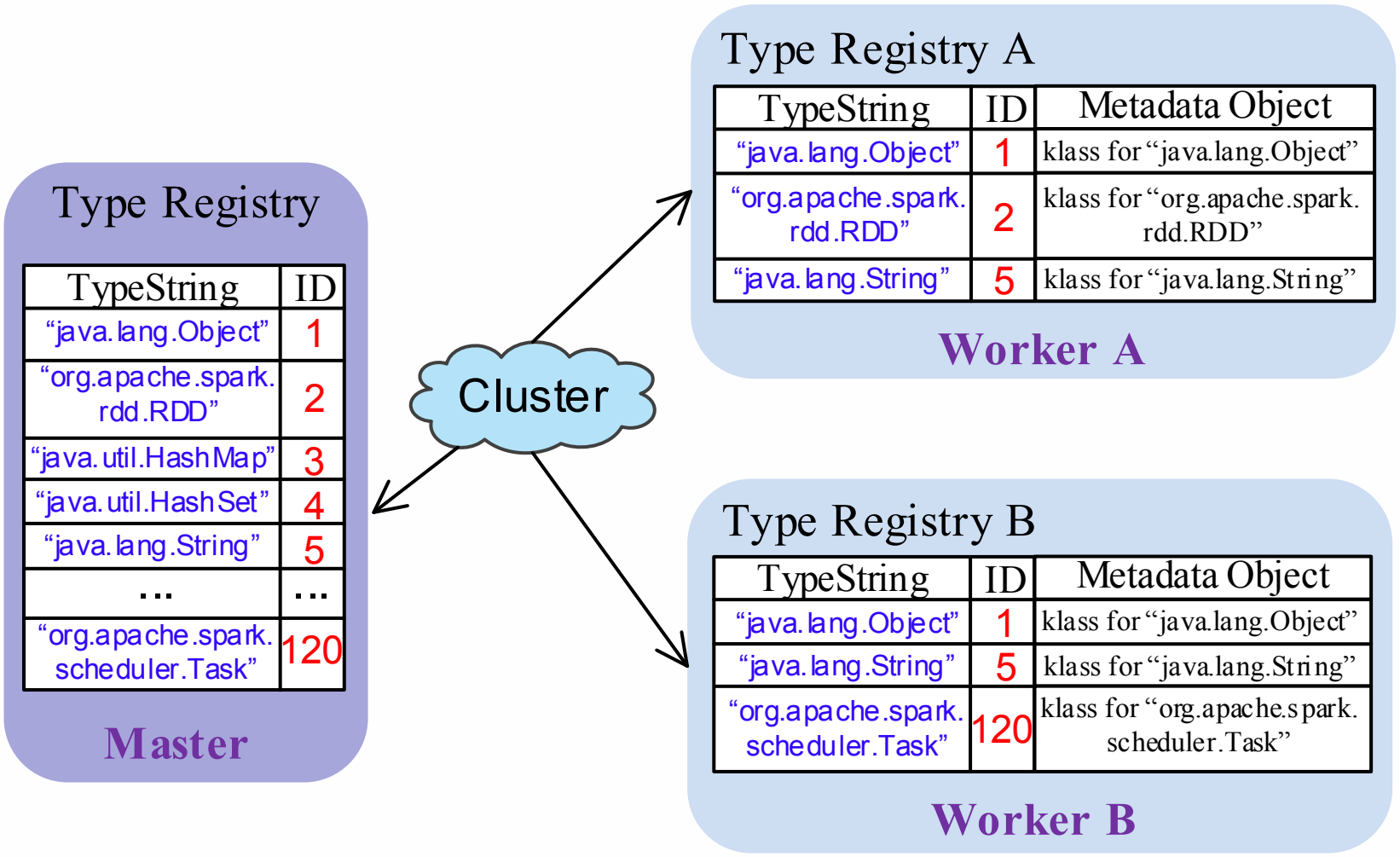
Challenges

1. Type representation
   - Automated global type numbering
2. Pointer representation
   - Use relative offsets
3. Local JVM adaptation
   - Visible for garbage collection
4. Work pipelining

Challenges

1. Type representation
   - Automated global type numbering
2. Pointer representation
   - Use relative offsets
3. Local JVM adaptation
   - Visible for garbage collection
4. Work pipelining
   - Buffering

Type registries

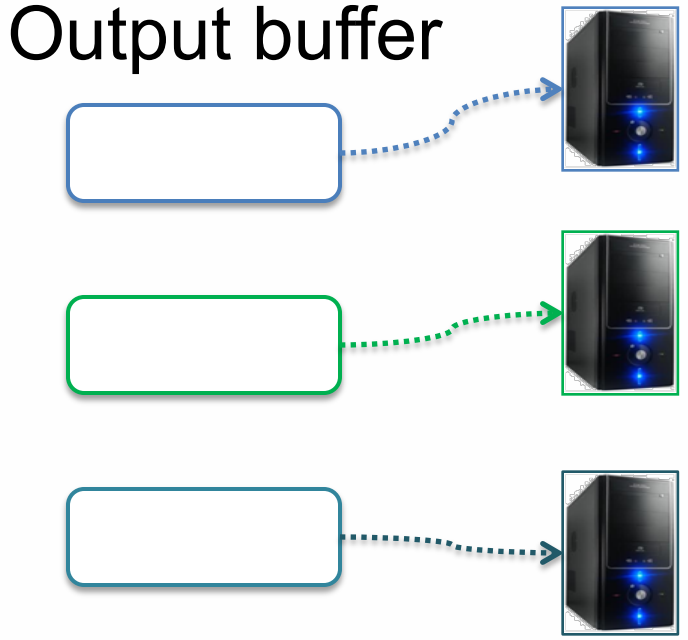


Output & Input buffer

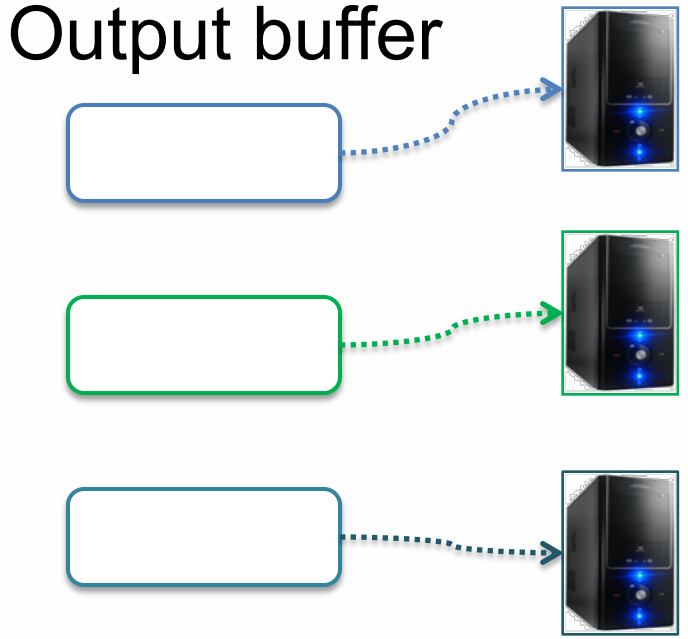
Output & Input buffer

Output buffer

Output & Input buffer

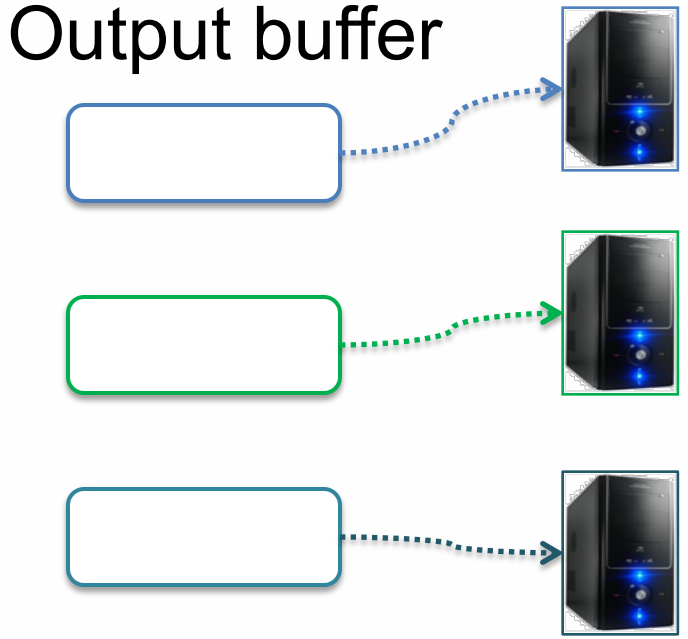


Output & Input buffer



- Segregated by **receivers**
- **One** for each receiver

Output & Input buffer

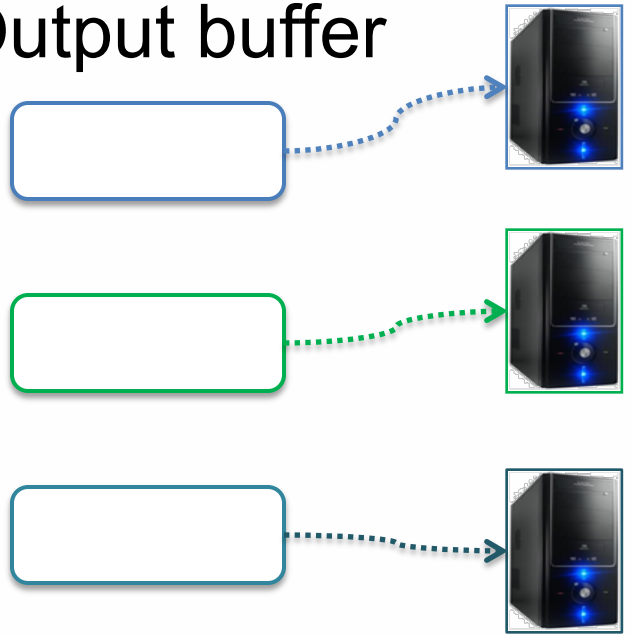


- Segregated by **receivers**
- **One** for each receiver
- In **native, off-the-heap** memory

Output & Input buffer

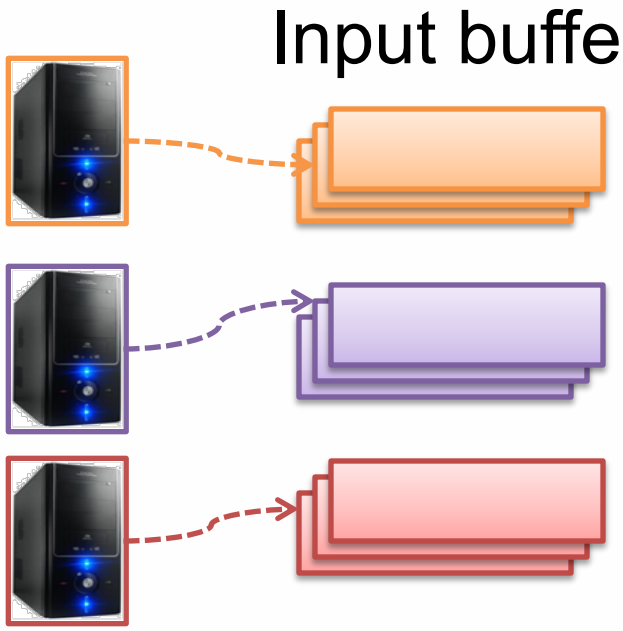
Input buffer

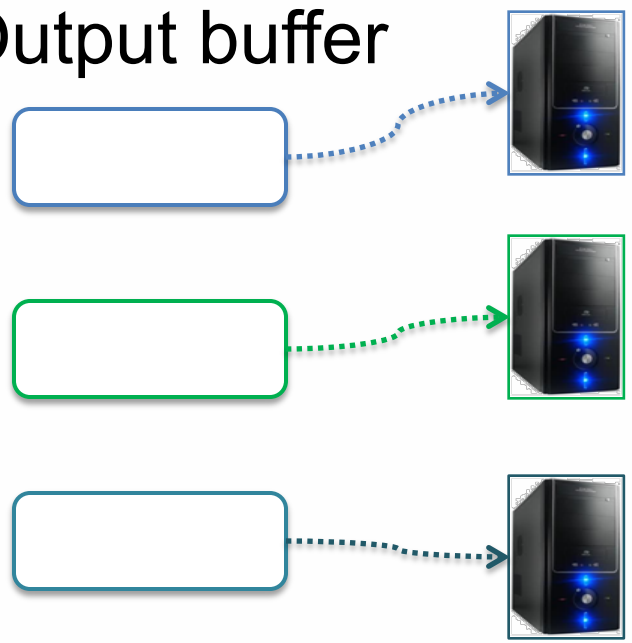
Output buffer



- Segregated by **receivers**
- **One** for each receiver
- In **native, off-the-heap** memory

Output & Input buffer





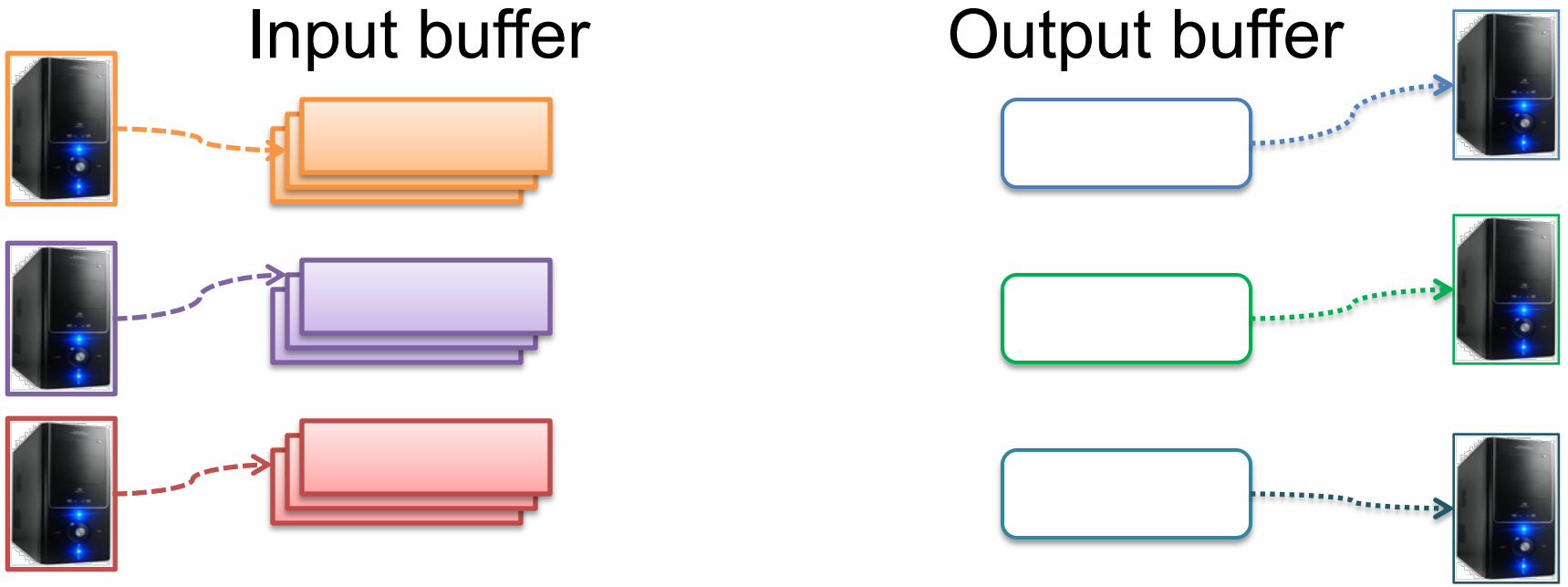
- Segregated by **receivers**
- **One** for each receiver
- In **native, off-the-heap** memory

Output & Input buffer

- Segregated by **senders**
- **Multiple** for each sender

- Segregated by **receivers**
- **One** for each receiver
- In **native, off-the-heap** memory

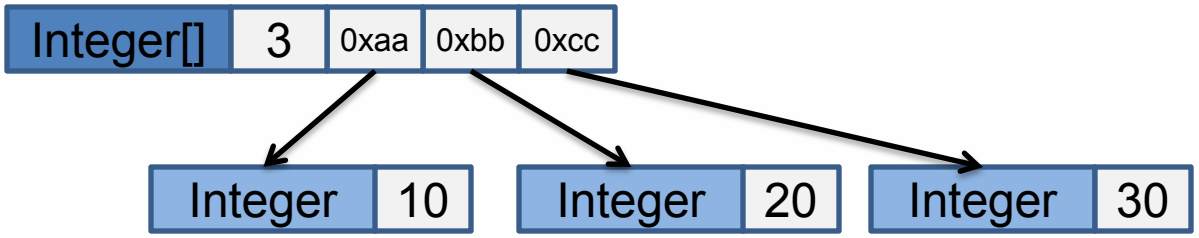
Output & Input buffer



- Segregated by **senders**
- **Multiple** for each sender
- In managed **heap**

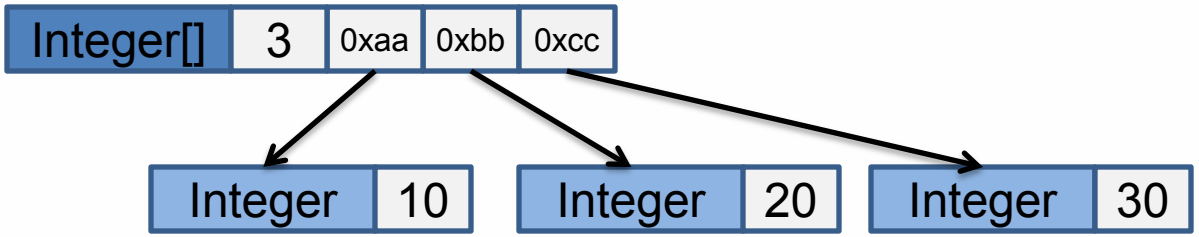
- Segregated by **receivers**
- **One** for each receiver
- In **native, off-the-heap** memory

Example: Serialization



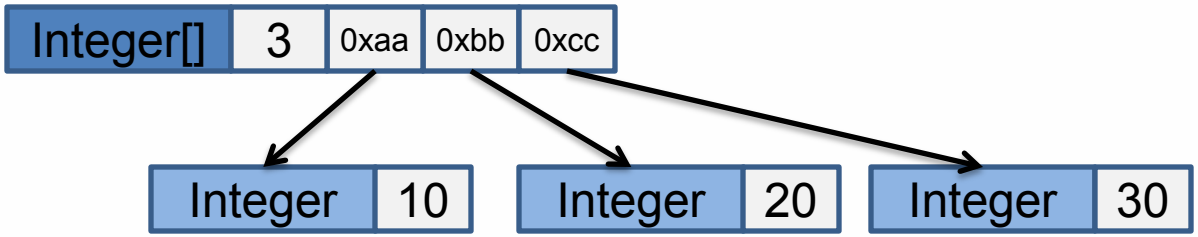
Example: Serialization

`writeObject()`



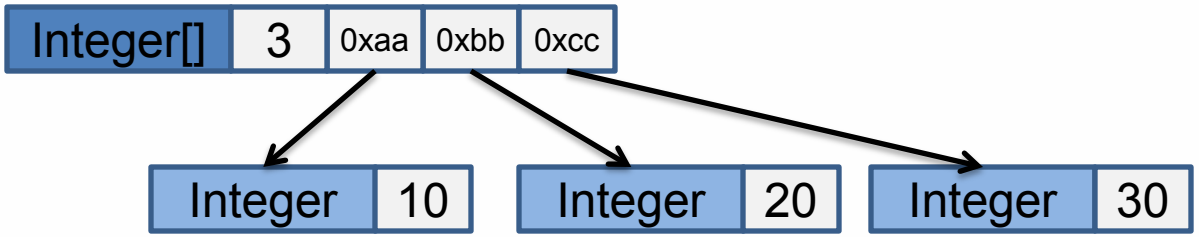
Example: Serialization

`writeObject()`



Example: Serialization

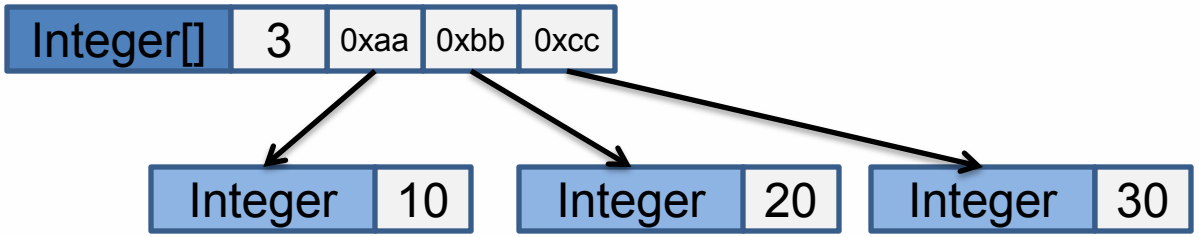
`writeObject()`



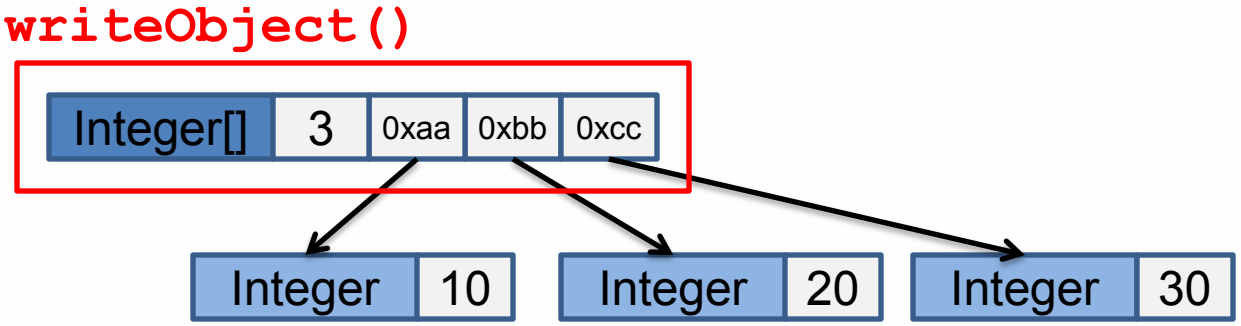
Output buffer
in native memory

Example: Serialization

`writeObject()`

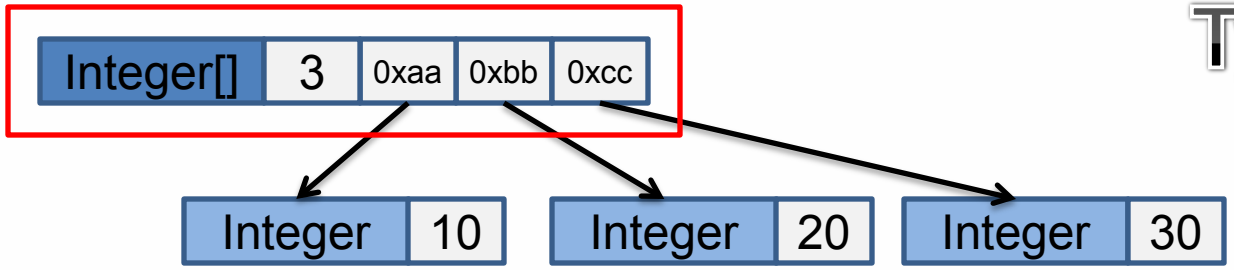


Example: Serialization



Example: Serialization

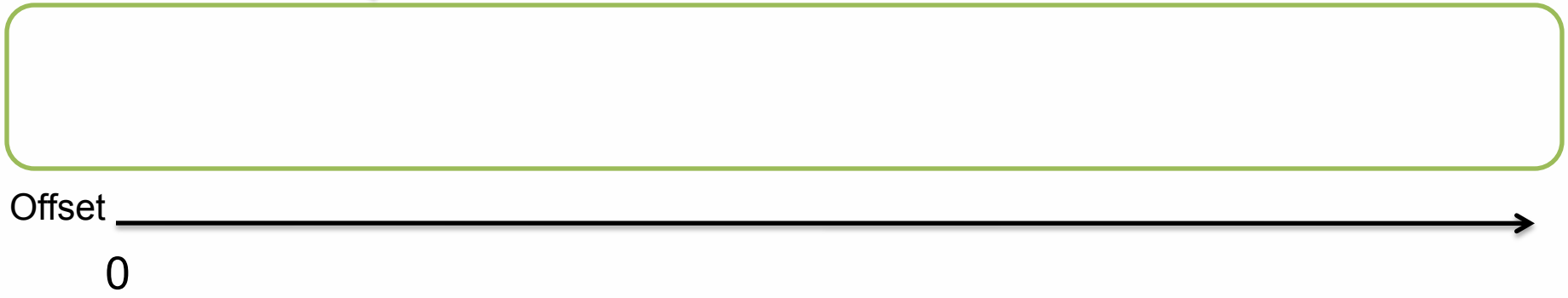
`writeObject()`



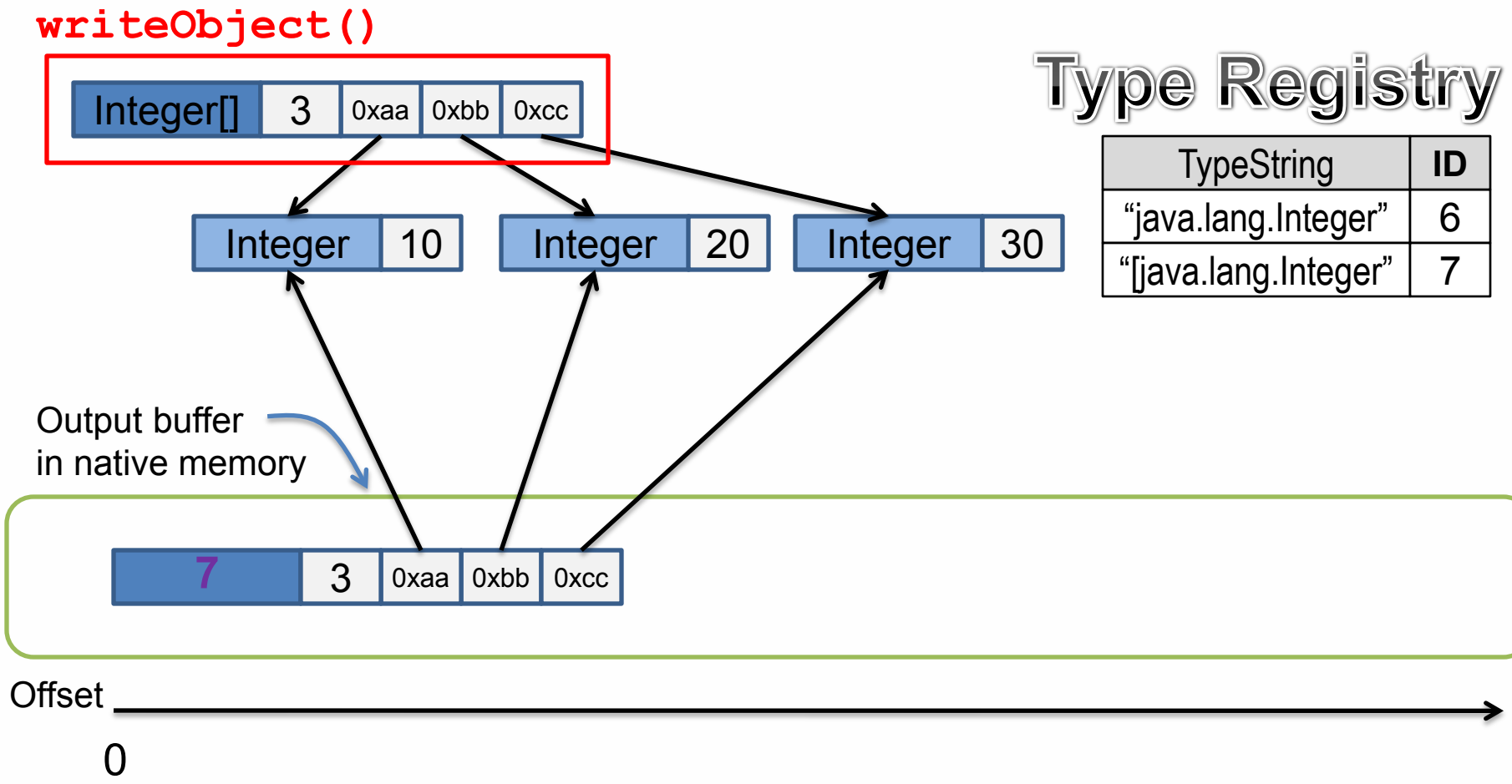
Type Registry

TypeString	ID
“java.lang.Integer”	6
“[java.lang.Integer”	7

Output buffer
in native memory

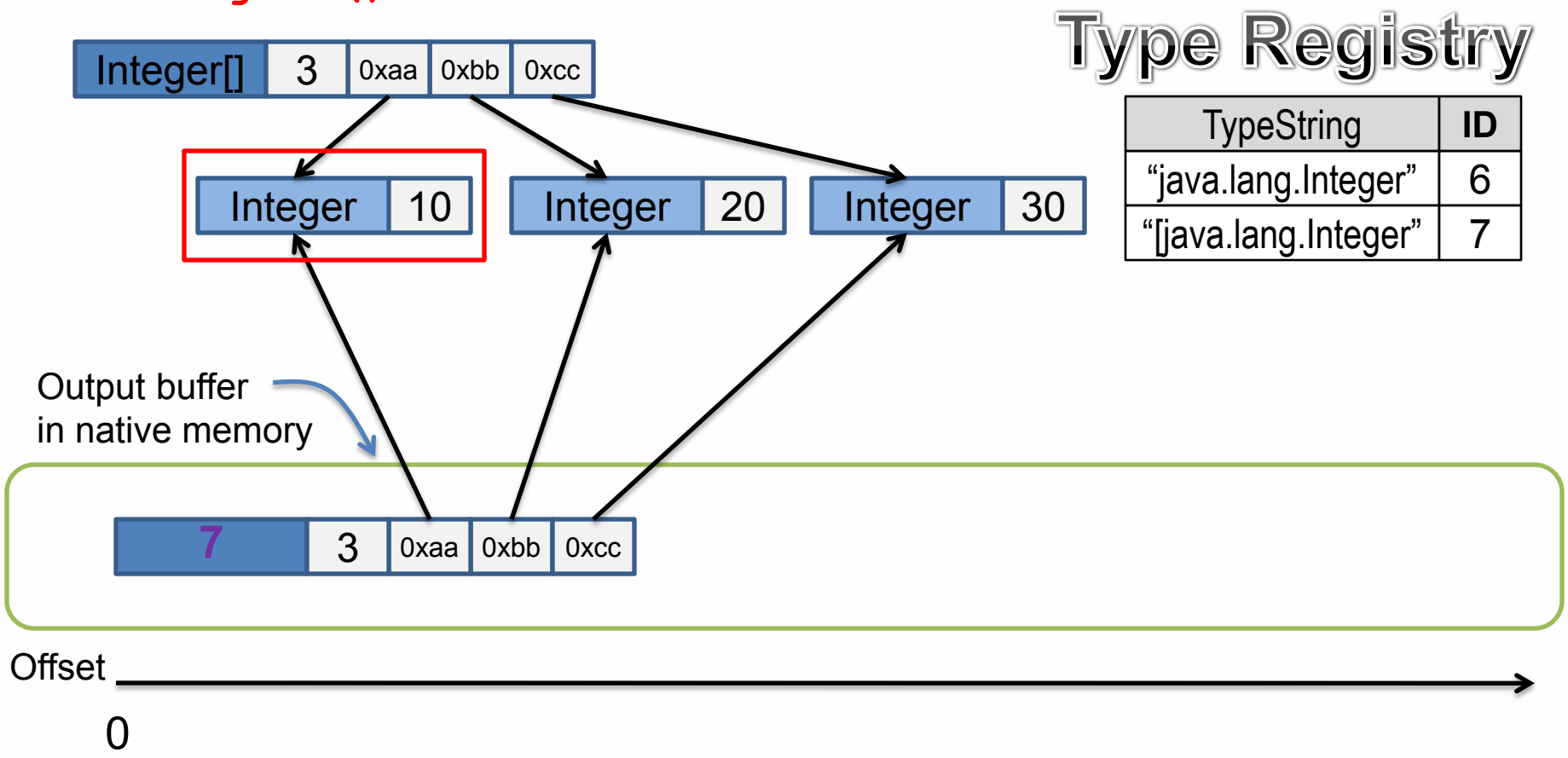


Example: Serialization



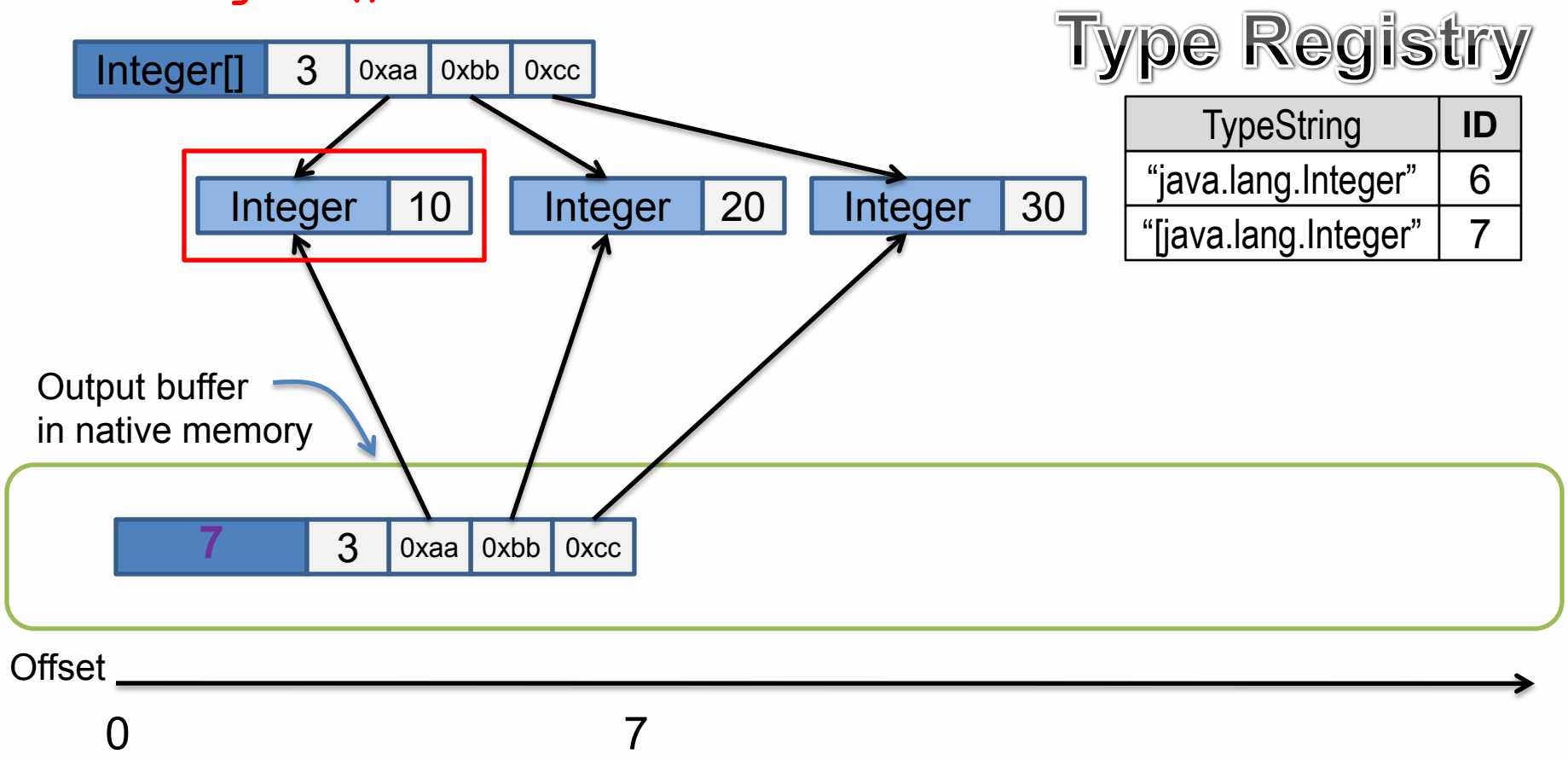
Example: Serialization

`writeObject()`



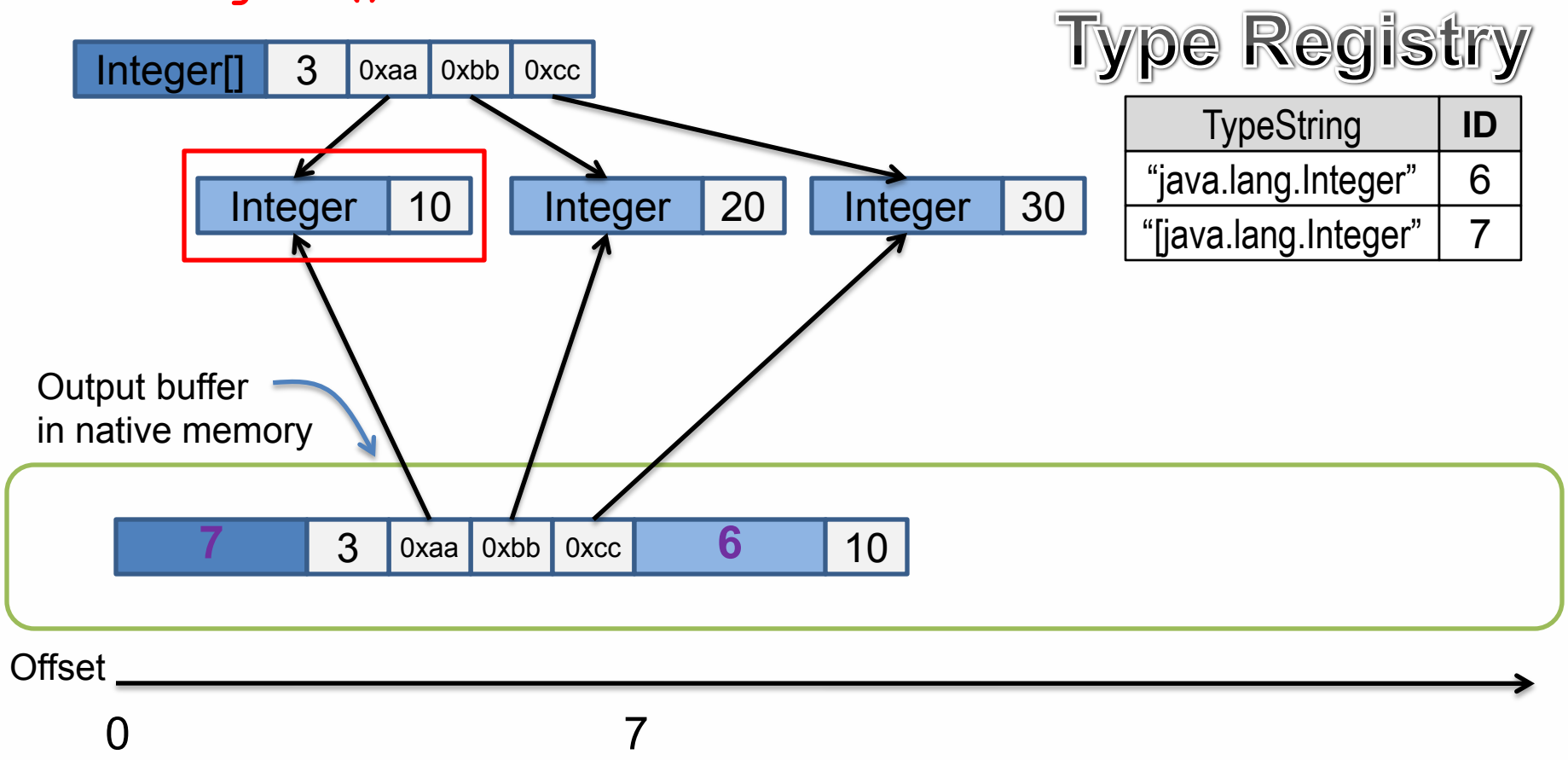
Example: Serialization

`writeObject()`



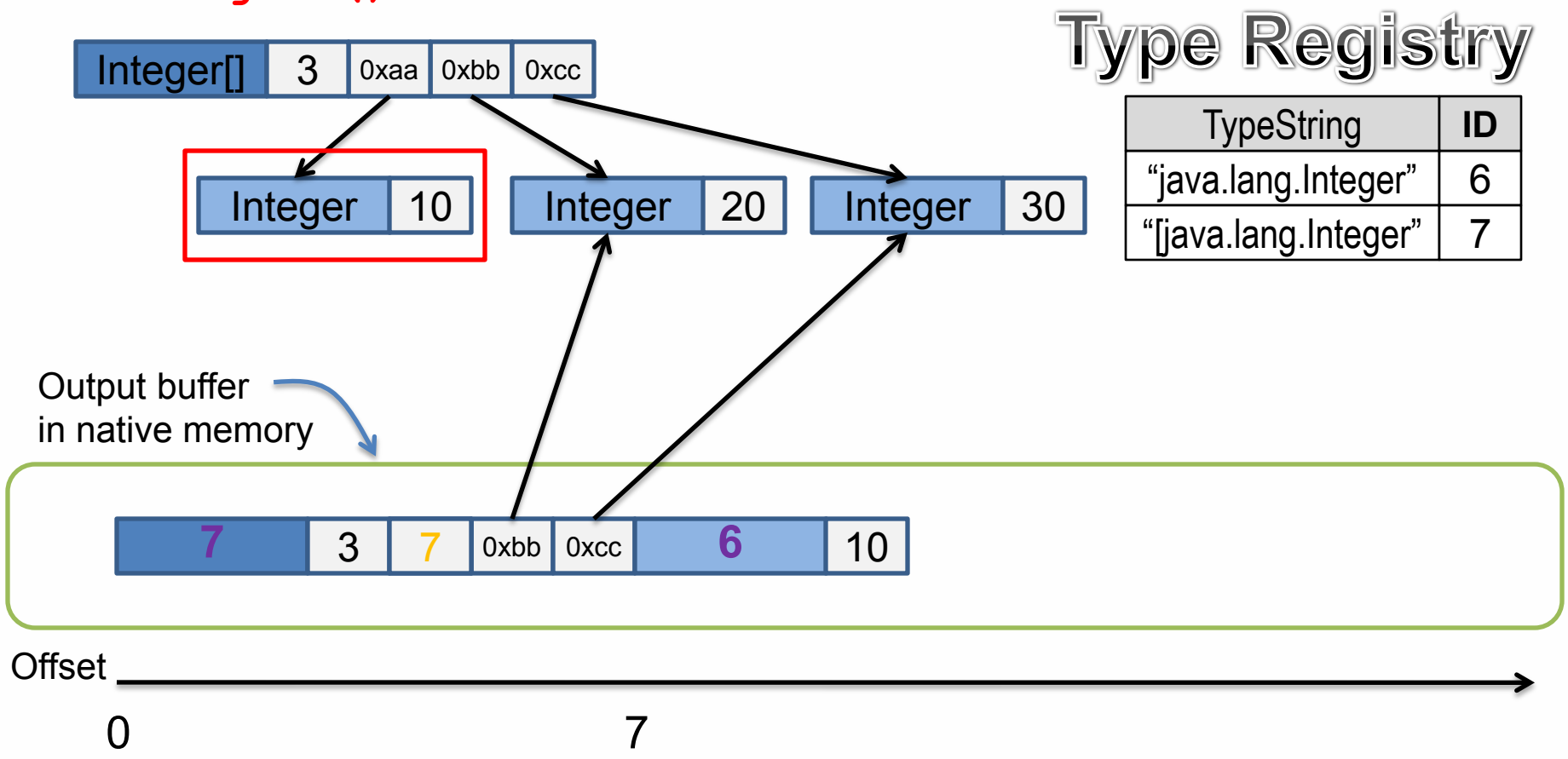
Example: Serialization

`writeObject()`



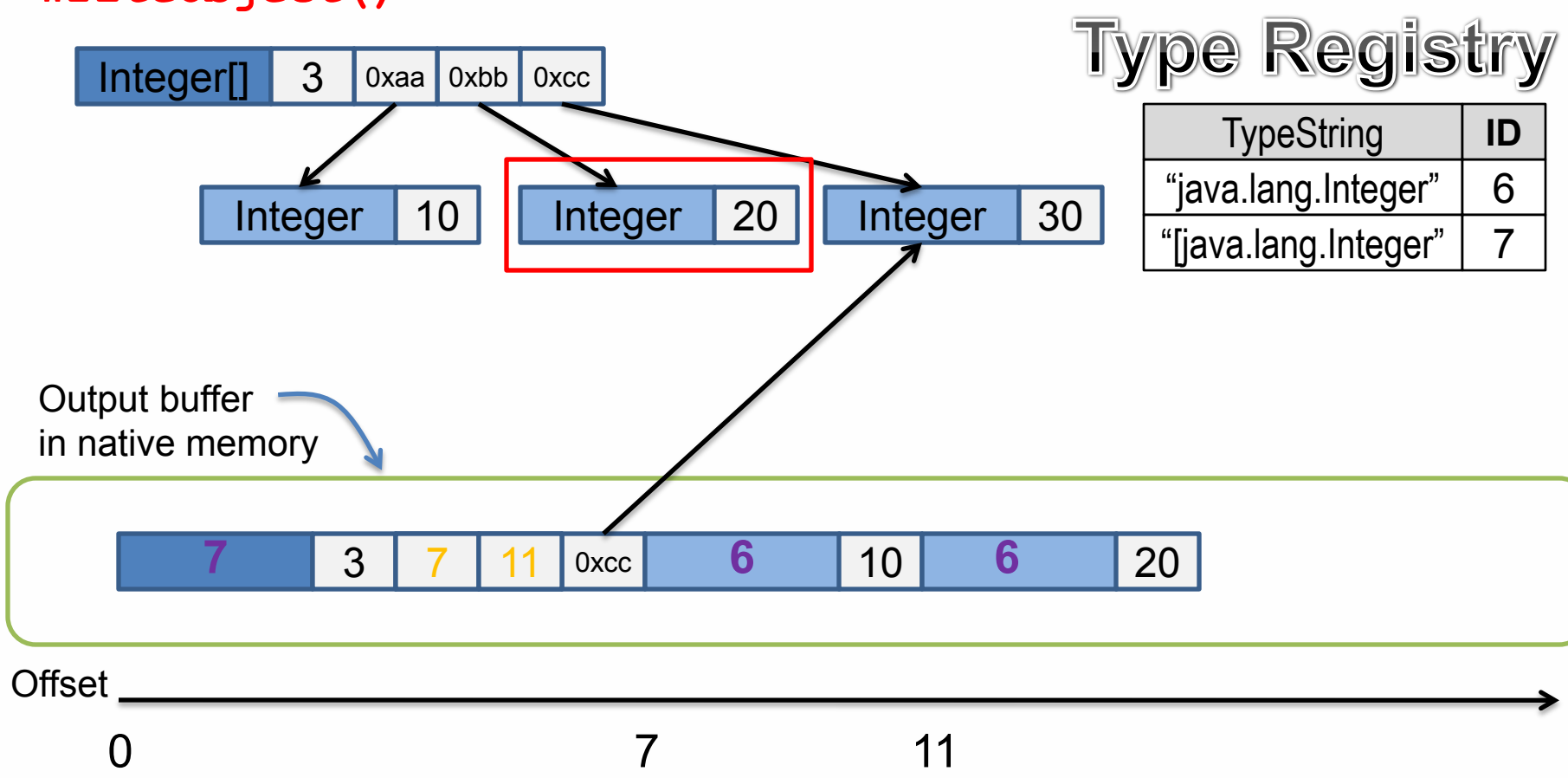
Example: Serialization

`writeObject()`



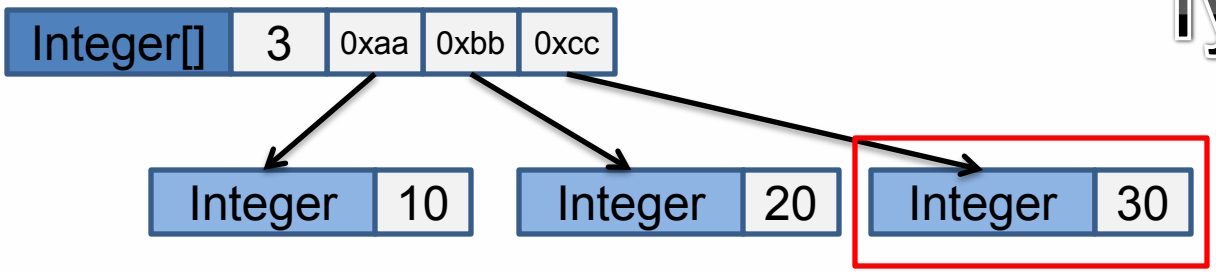
Example: Serialization

`writeObject()`



Example: Serialization

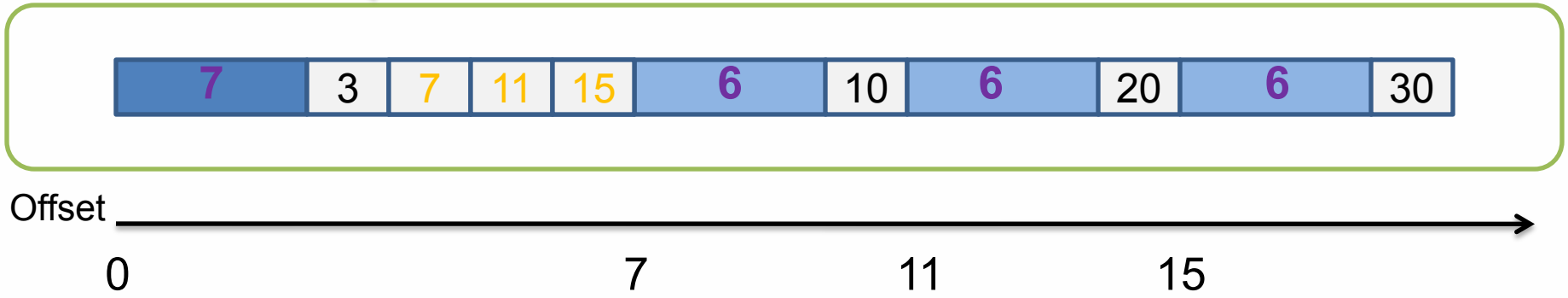
`writeObject()`



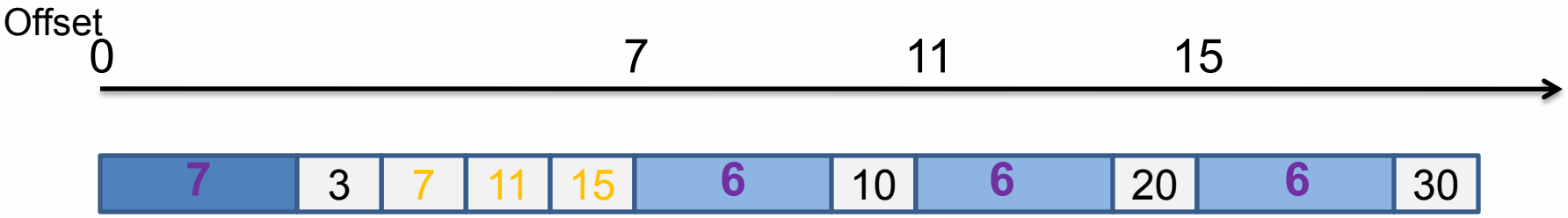
Type Registry

TypeString	ID
“java.lang.Integer”	6
“[java.lang.Integer”	7

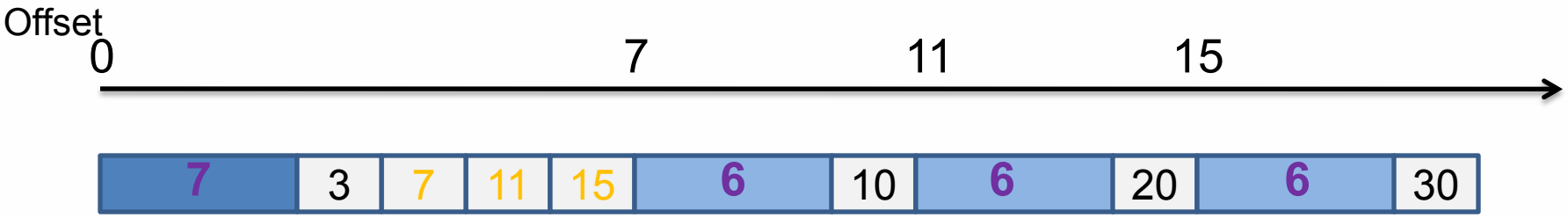
Output buffer
in native memory



Example: Deserialization



Example: Deserialization




```
readObject()
```

Example: Deserialization

`readObject()`

Input buffer
in heap

Example: Deserialization

`readObject()`

Input buffer
in heap

Type Registry

ID	MetadataObject
6	java.lang.Integer
7	java.lang.Integer[]

Example: Deserialization

`readObject()`

Input buffer in heap

Type Registry

ID	MetadataObject
6	java.lang.Integer
7	java.lang.Integer[]

Example: Deserialization

`readObject()`

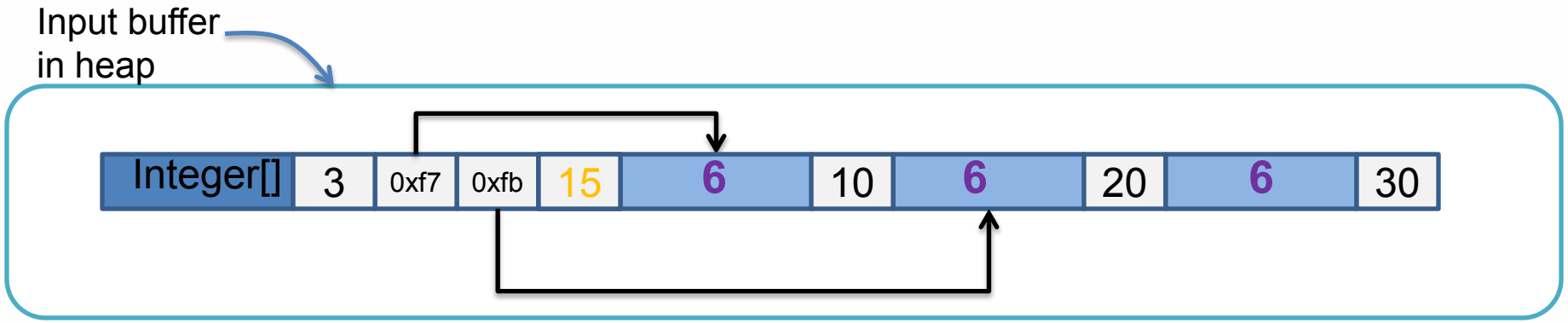
Input buffer in heap

Type Registry

ID	MetadataObject
6	java.lang.Integer
7	java.lang.Integer[]

Example: Deserialization

`readObject()`

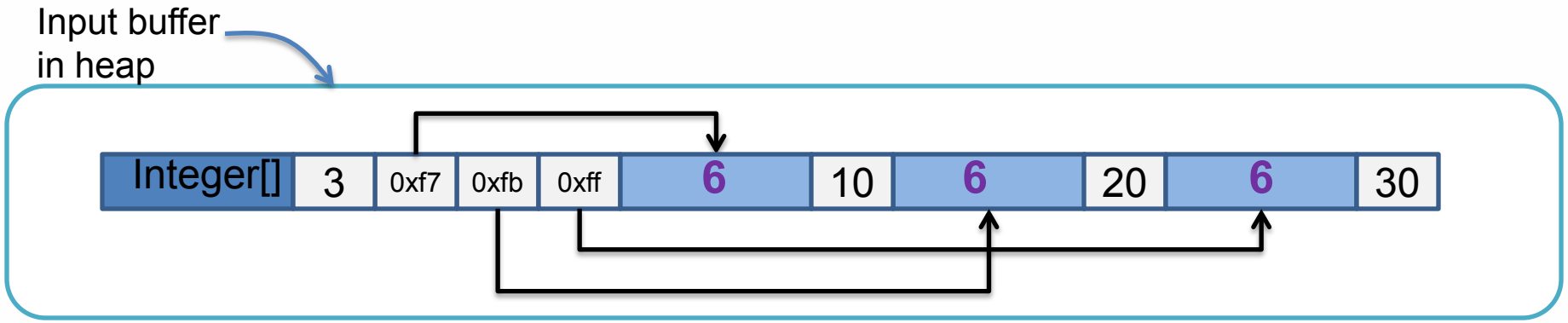


Type Registry

ID	MetadataObject
6	java.lang.Integer
7	java.lang.Integer[]

Example: Deserialization

`readObject()`

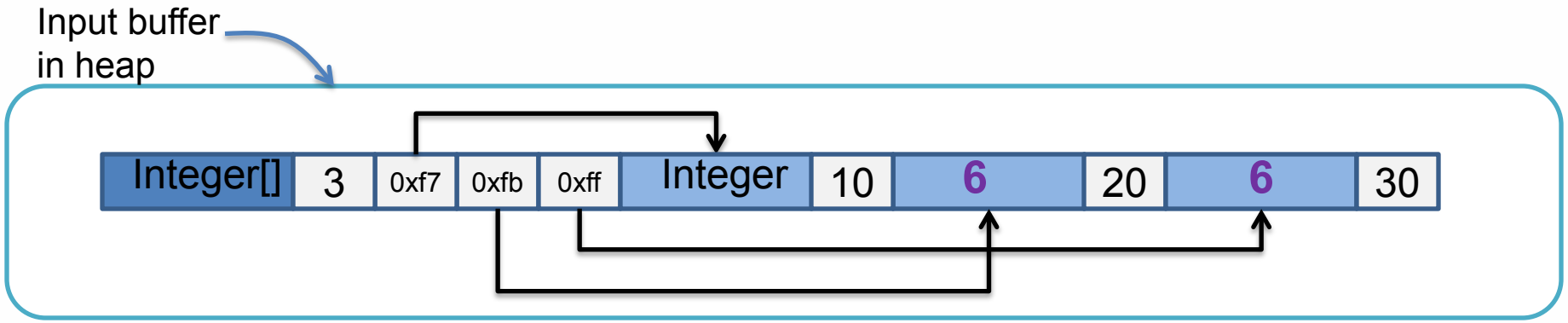


Type Registry

ID	MetadataObject
6	java.lang.Integer
7	java.lang.Integer[]

Example: Deserialization

`readObject()`

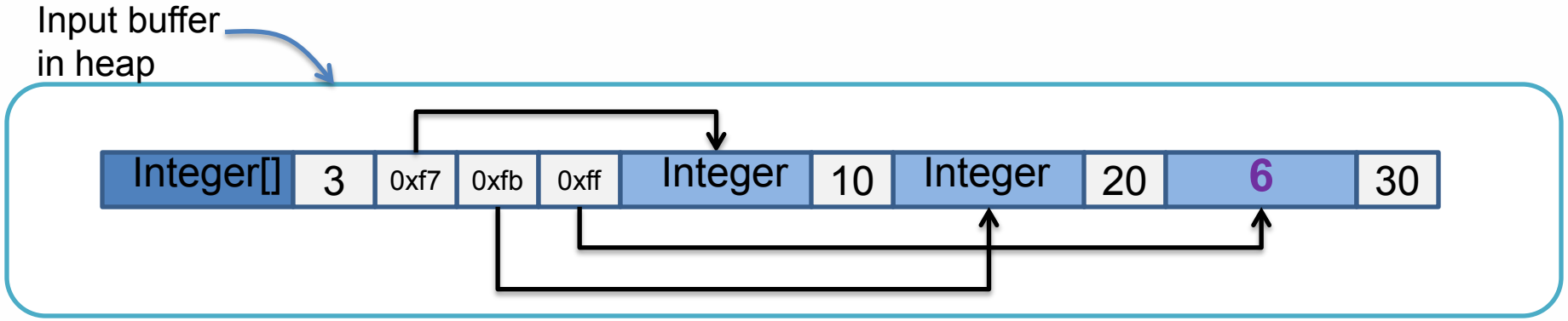


Type Registry

ID	MetadataObject
6	java.lang.Integer
7	java.lang.Integer[]

Example: Deserialization

`readObject()`

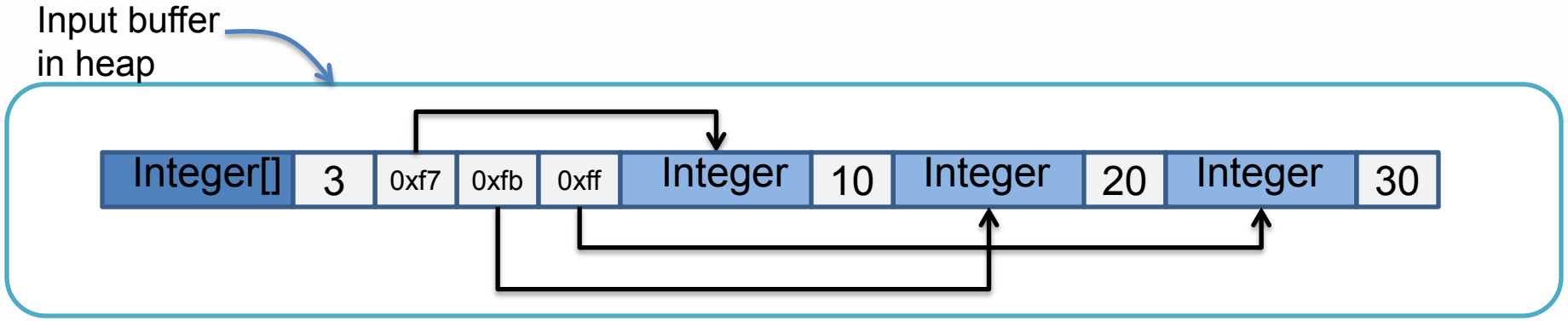


Type Registry

ID	MetadataObject
6	java.lang.Integer
7	java.lang.Integer[]

Example: Deserialization

`readObject()`



Type Registry

ID	MetadataObject
6	java.lang.Integer
7	java.lang.Integer[]

In the paper

- Cyclic references
- Shared objects
- Support for threads
- Interaction with GC
- Integrating Skyway in real systems

Evaluations - Microbenchmark

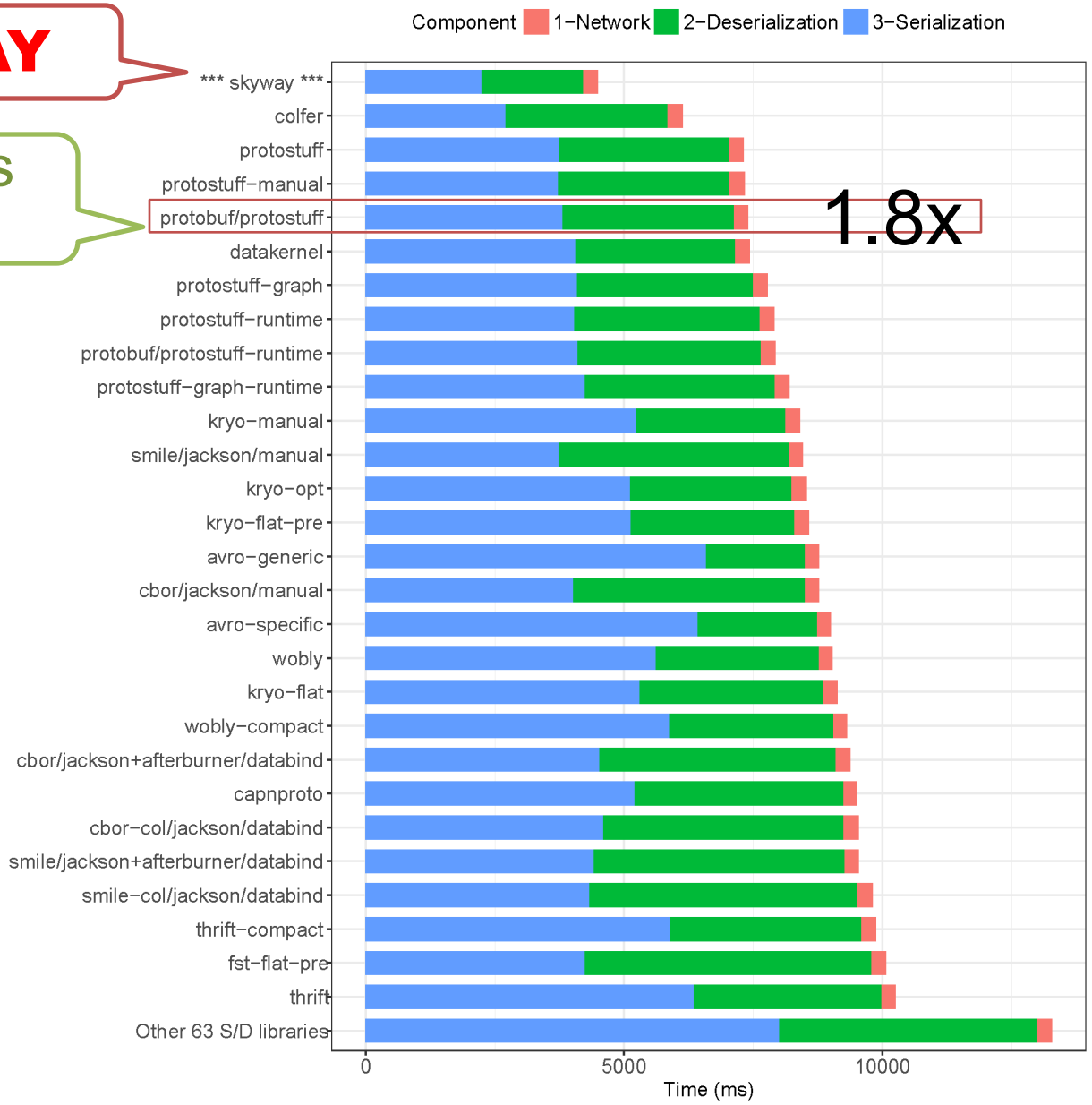
- Java Serializer Benchmark Set
  - Extensive performance evaluation with existing **90** serializers

SKYWAY


Component
1−Network
2−Deserialization
3−Serialization
*** skyway ***
colfer
protostuff
protostuff−manual
protobuf/protostuff
datakernel
protostuff−graph
protostuff−runtime
protobuf/protostuff−runtime
protostuff−graph−runtime
kryo−manual
smile/jackson/manual
kryo−opt
kryo−flat−pre
avro−generic
cbor/jackson/manual
avro−specific
wobly
kryo−flat
wobly−compact
cbor/jackson+afterburner/databind
capnproto
cbor−col/jackson/databind
smile/jackson+afterburner/databind
smile−col/jackson/databind
thrift−compact
fst−flat−pre
thrift
Other 63 S/D libraries
0
5000
10000
Time (ms)

SKYWAY

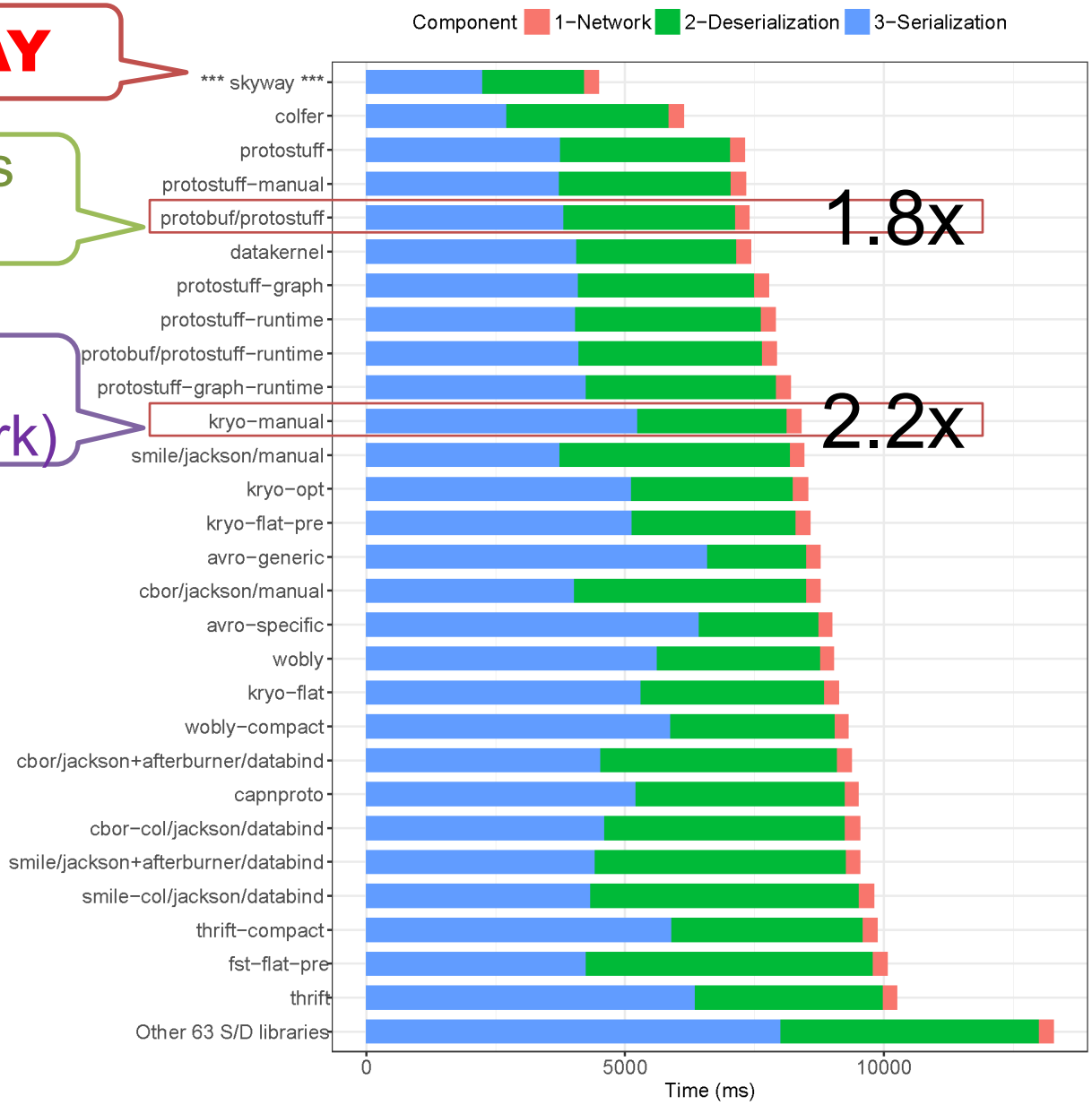
GOOGLE's Protobuf



SKYWAY

GOOGLE's Protobuf

Kryo (rec. by Spark)

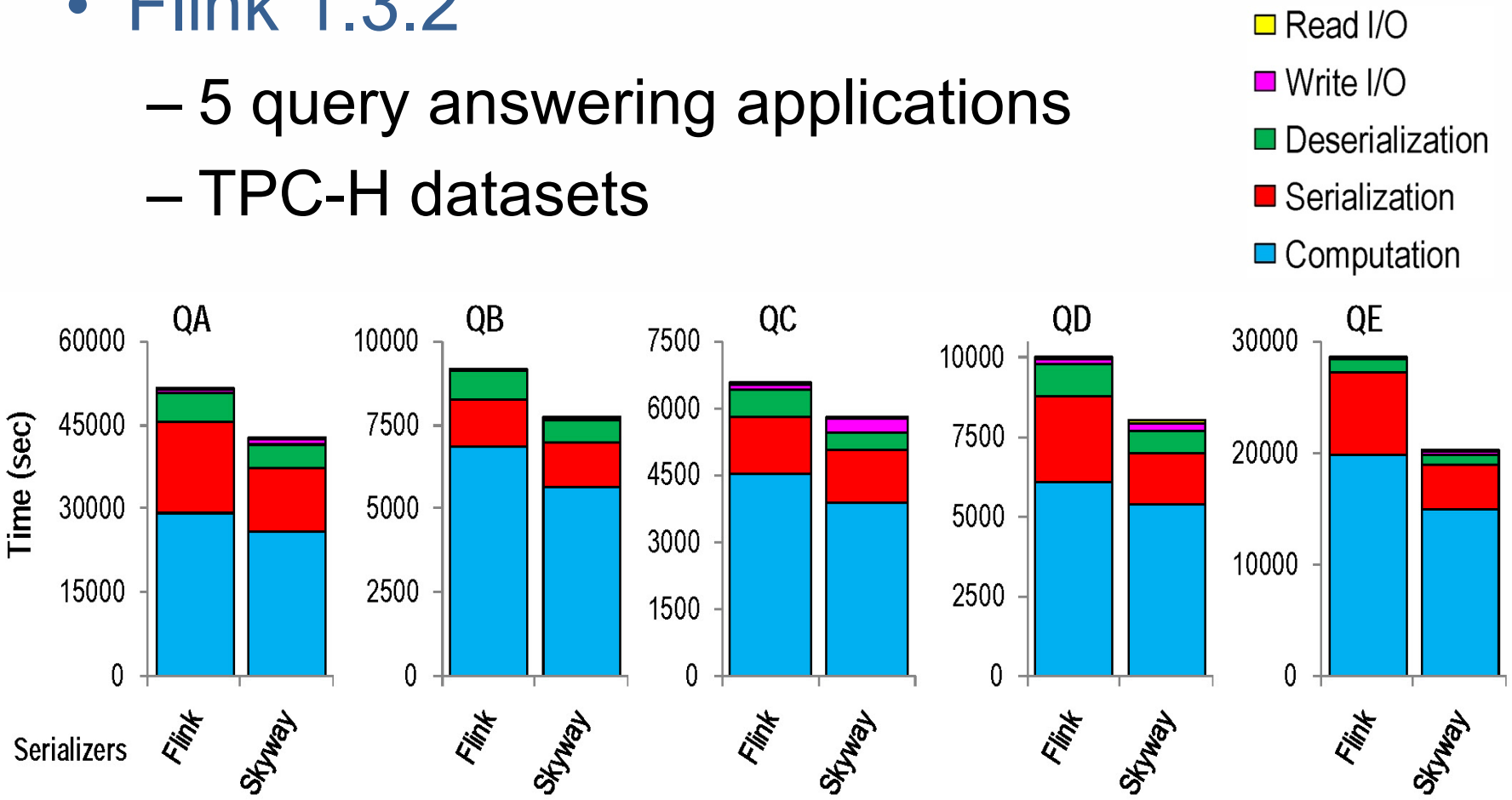


Evaluations – Real Systems

- Flink 1.3.2
  - 5 query answering applications
  - TPC-H datasets

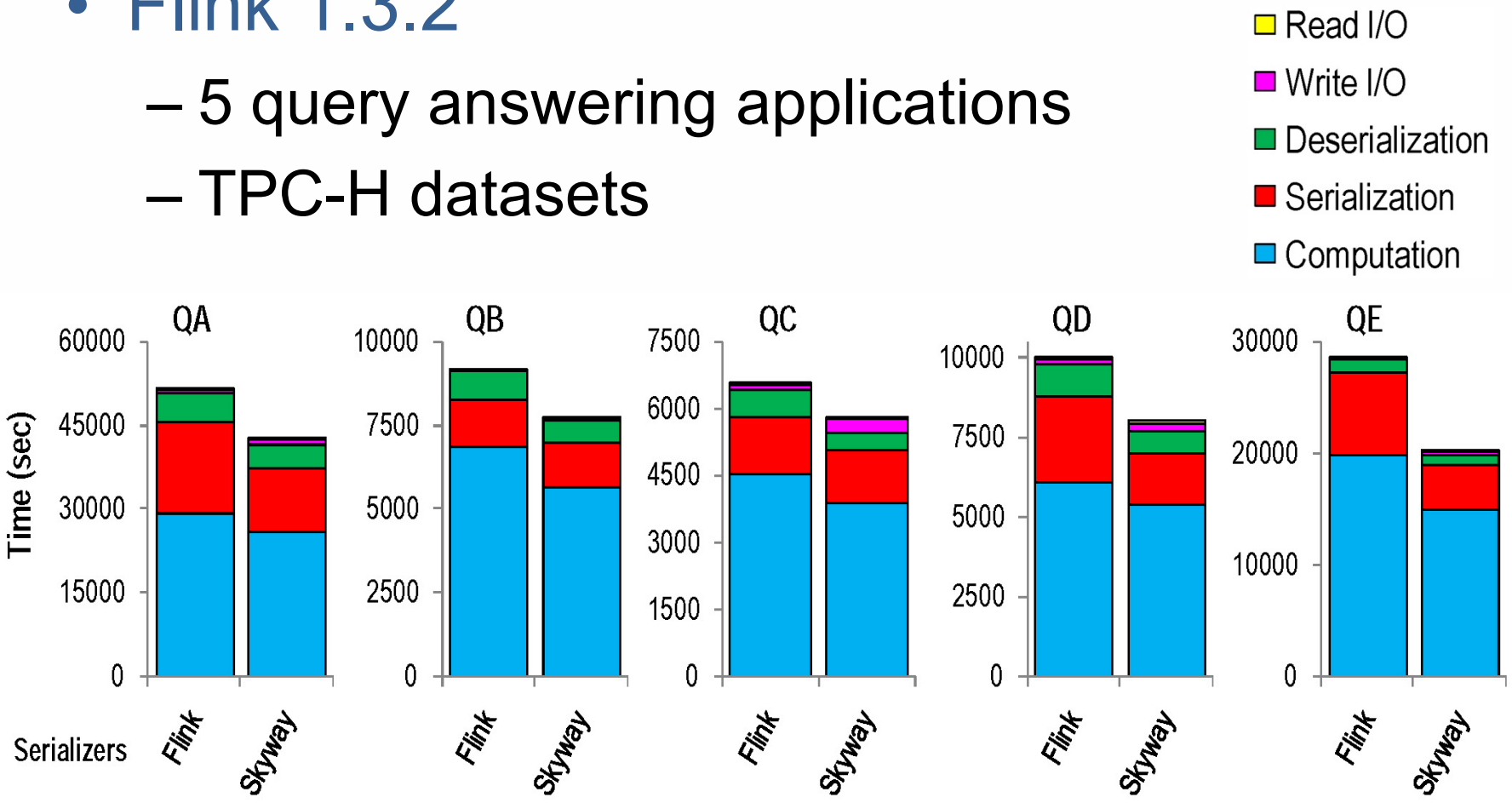
Evaluations – Real Systems

- Flink 1.3.2
  - 5 query answering applications
  - TPC-H datasets



Evaluations – Real Systems

- Flink 1.3.2
  - 5 query answering applications
  - TPC-H datasets

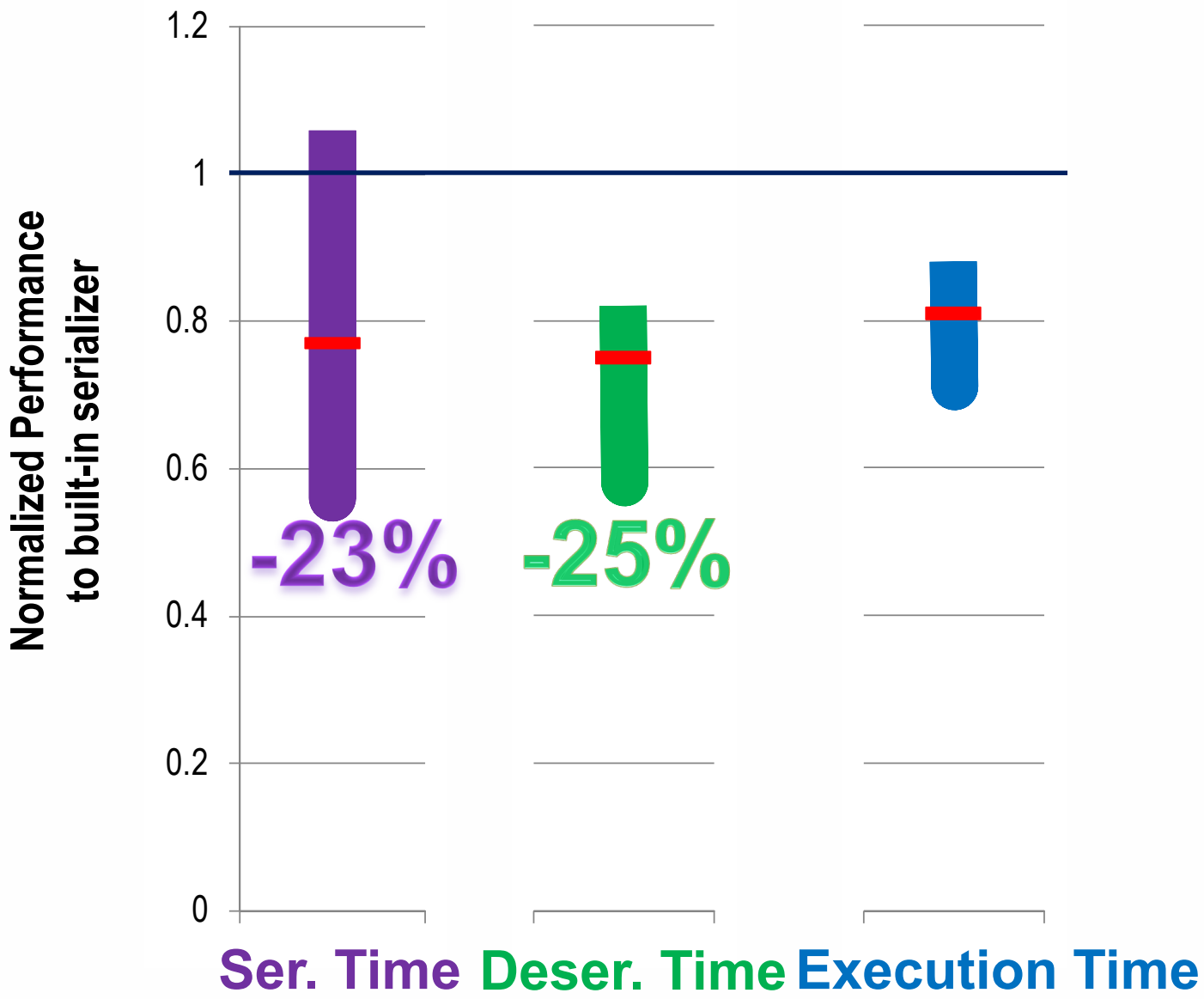


On average, reduces end-to-end time by **19%**

Improvement Summary: Flink

Improvement Summary: Flink

Improvement Summary: Flink



Improvement Summary: Flink

Evaluations – Real Systems

- Spark 2.1.0
  - 4 applications: **WordCount**, **PageRank**, **ConnectedComponents**, and **TriangleCounting**
  - 4 datasets:

    LiveJournal, **Orkut**, **UK-2005**, and **Twitter**

Evaluations – Real Systems

- Spark 2.1.0
  - 4 applications: **WordCount**, **PageRank**, **ConnectedComponents**, and **TriangleCounting**
  - 4 datasets:

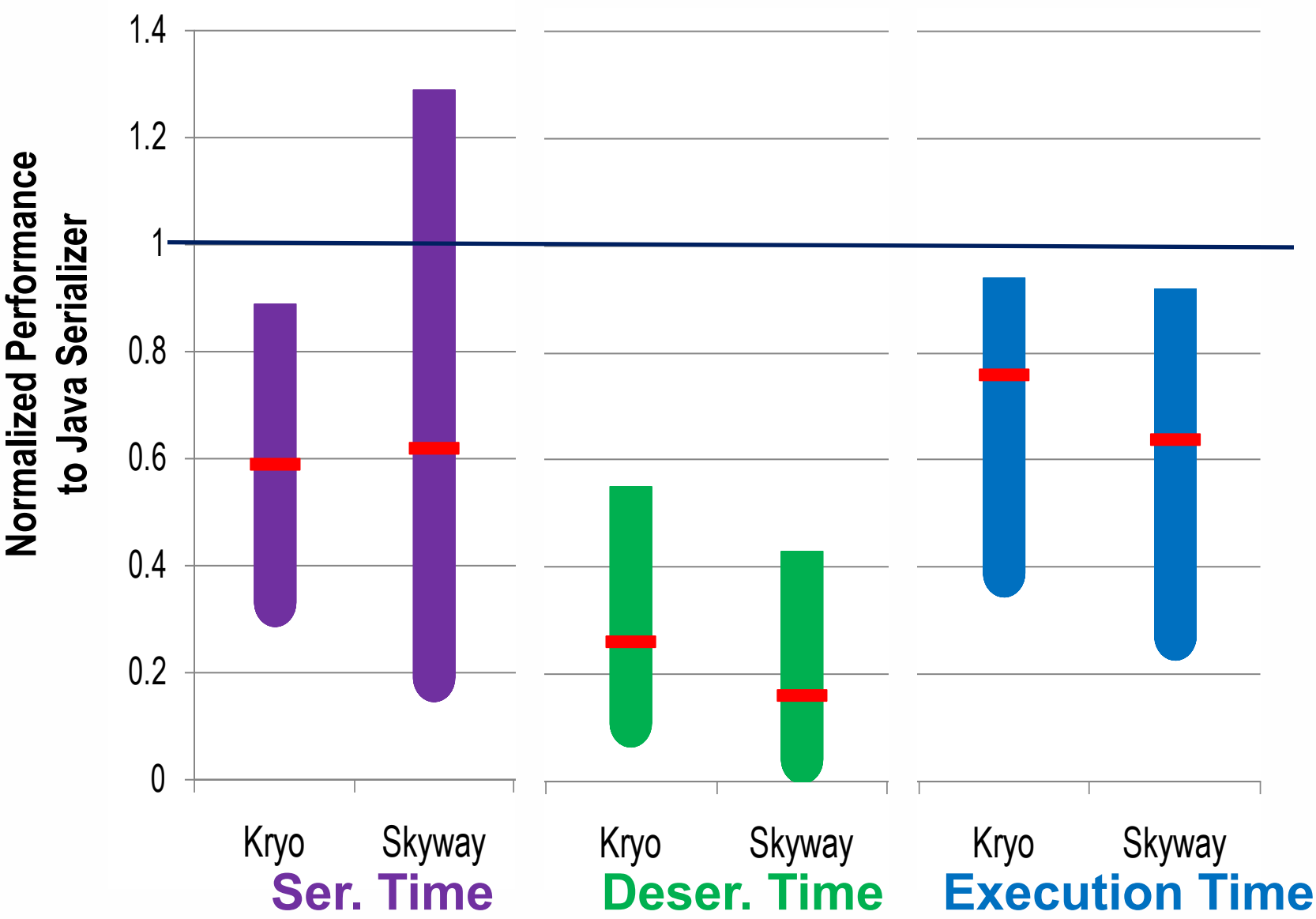
    LiveJournal, **Orkut**, **UK-2005**, and **Twitter**

On average, reduces end-to-end time

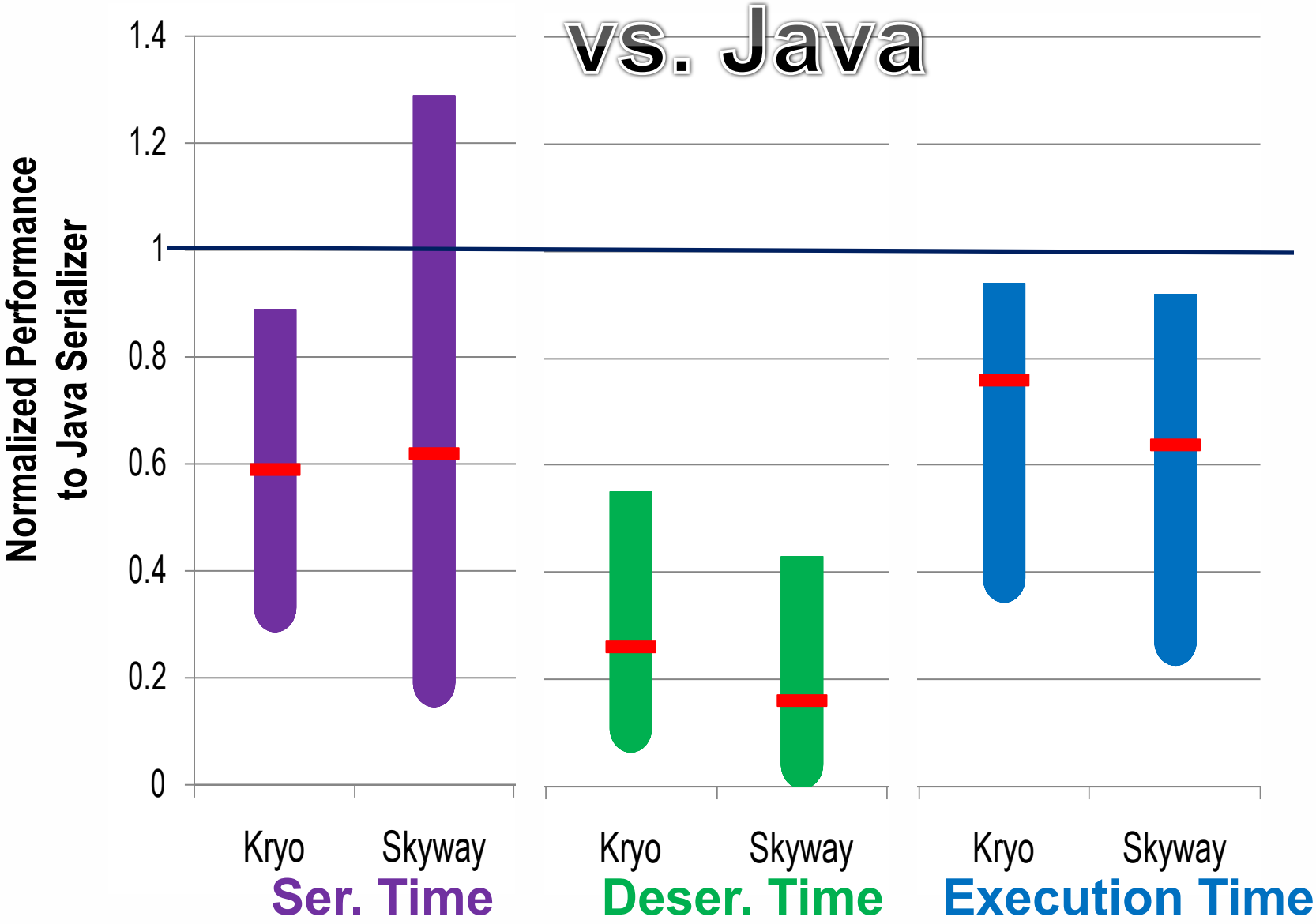
by **16%** (w.r.t. Kryo)

by **36%** (w.r.t. Java serializer)

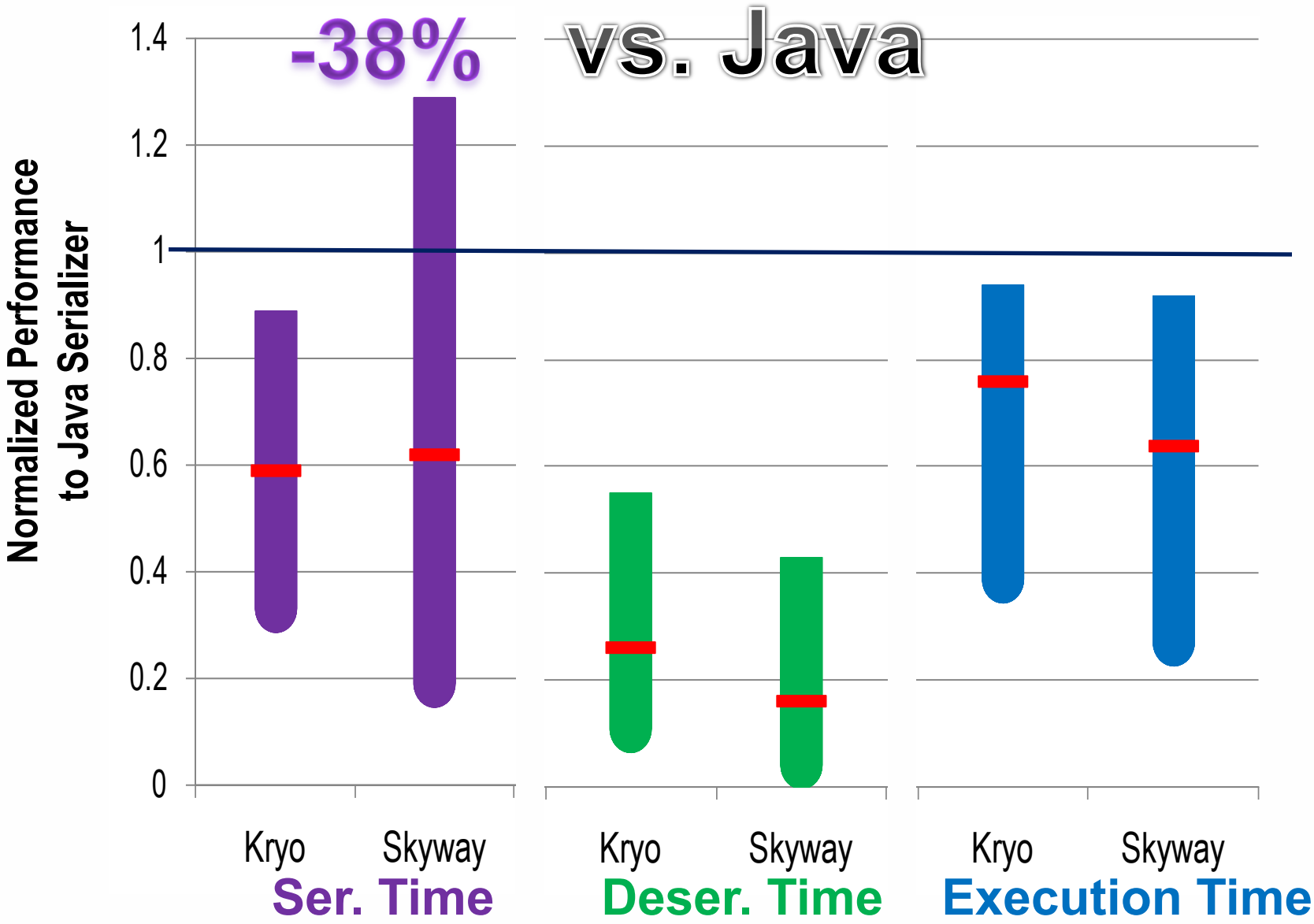
Improvement Summary: Spark



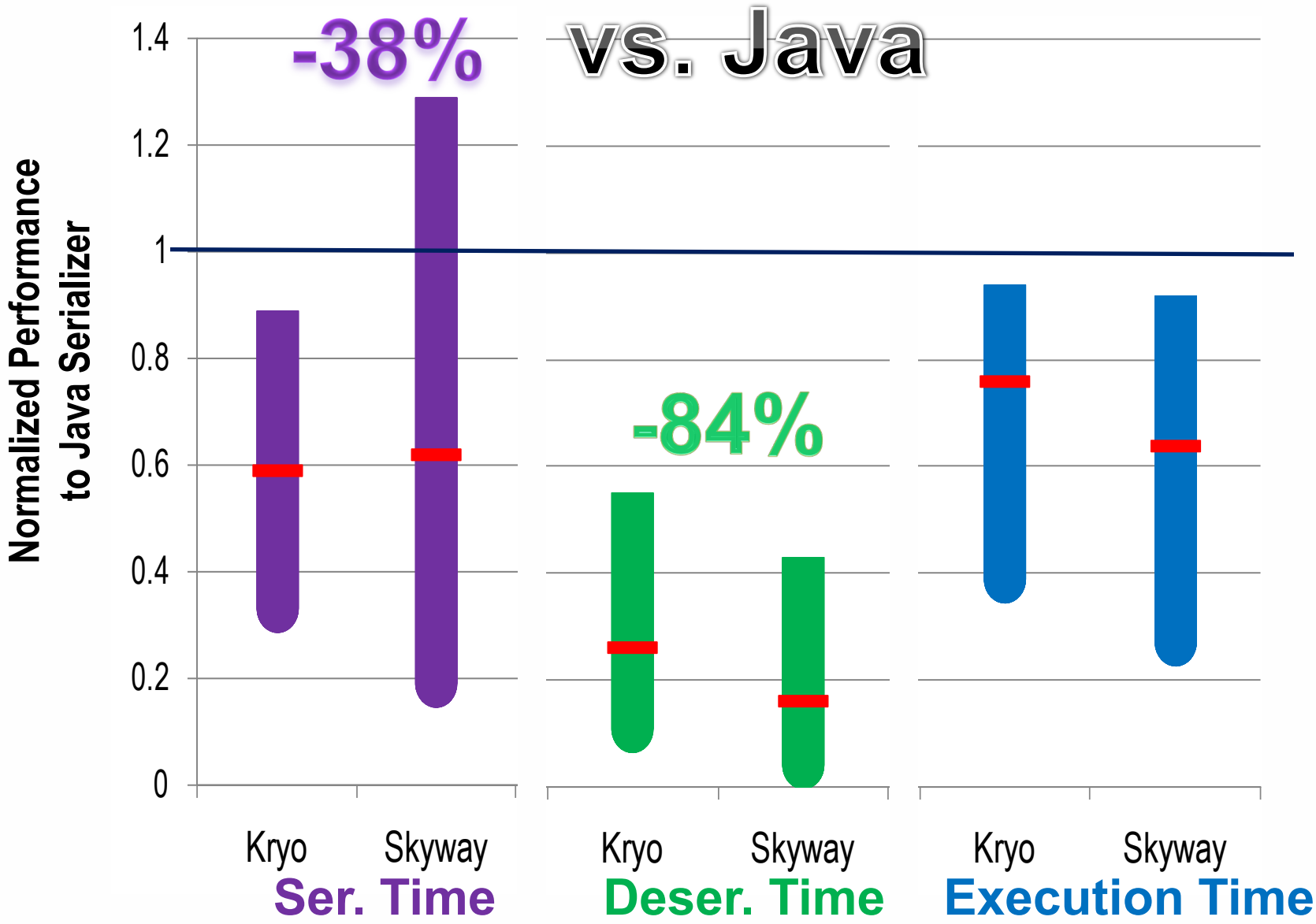
Improvement Summary: Spark vs. Java



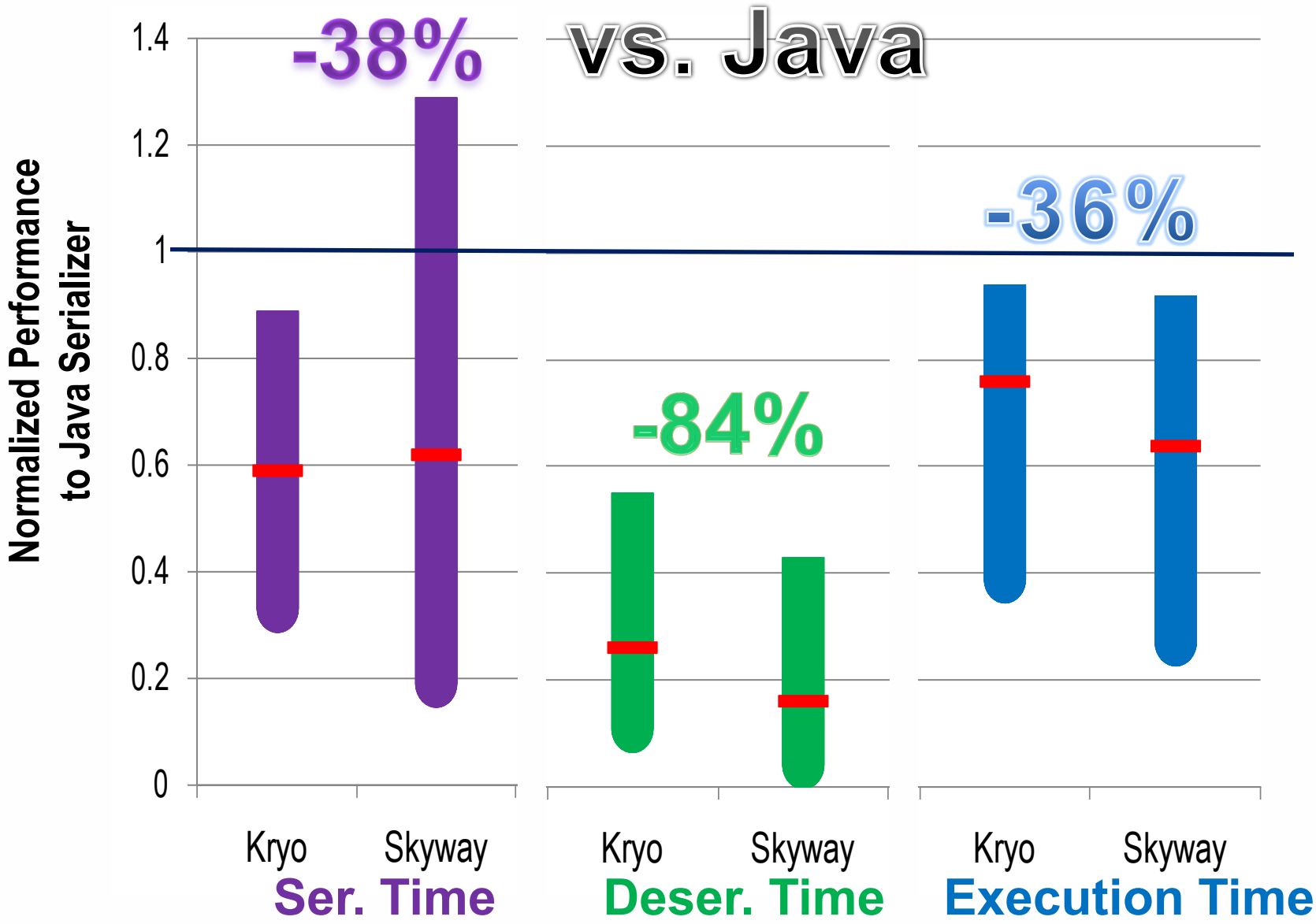
Improvement Summary: Spark



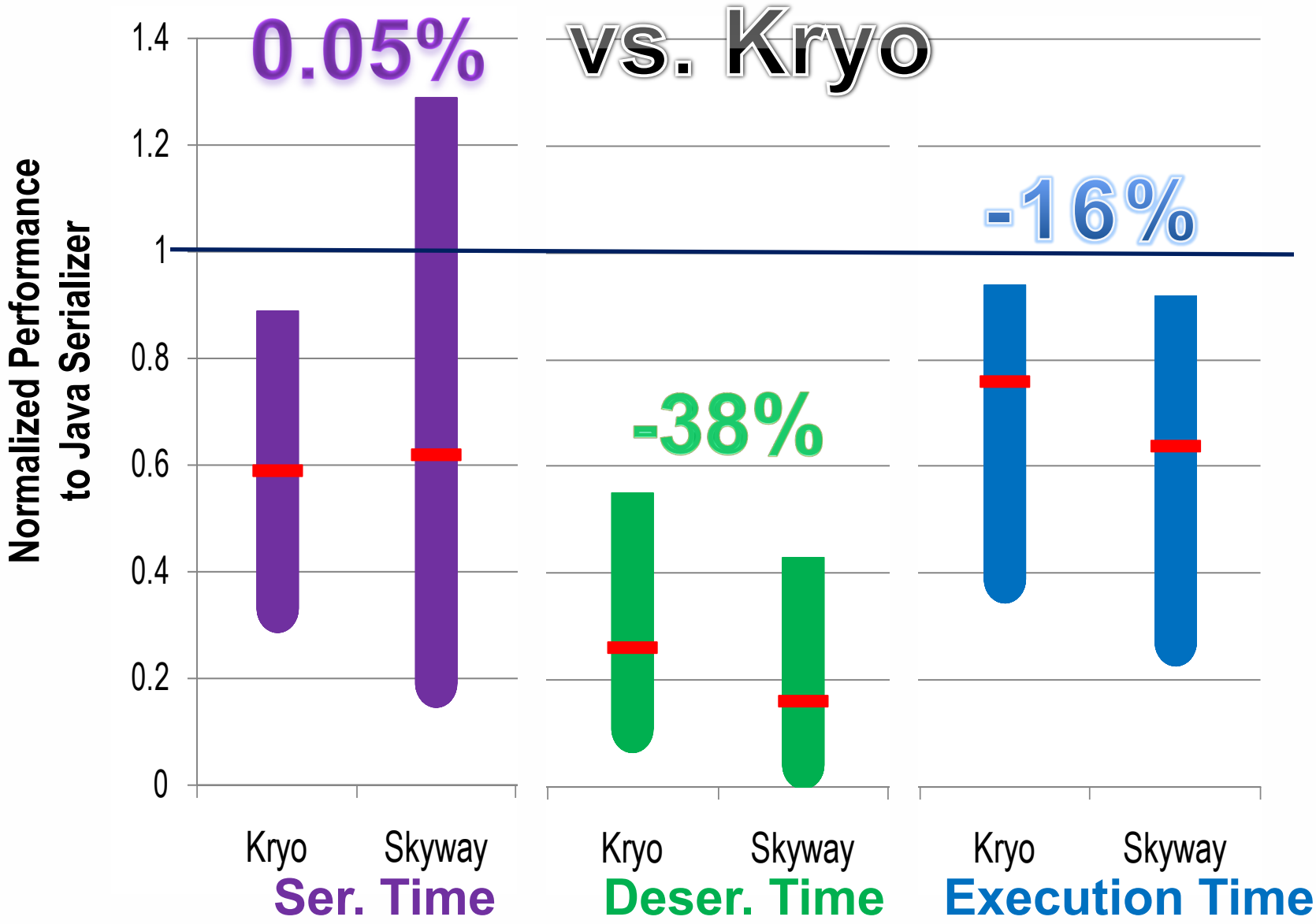
Improvement Summary: Spark



Improvement Summary: Spark



Improvement Summary: Spark



Conclusion

- *Goal:* Reduce data transfer costs in Big Data systems
- *Solution:* **Skyway**, the first JVM-based serializer
  - Efficiently transfer data
  - Easy to integrate

Thank You!