DOWN THE RABBIT HOLE OF

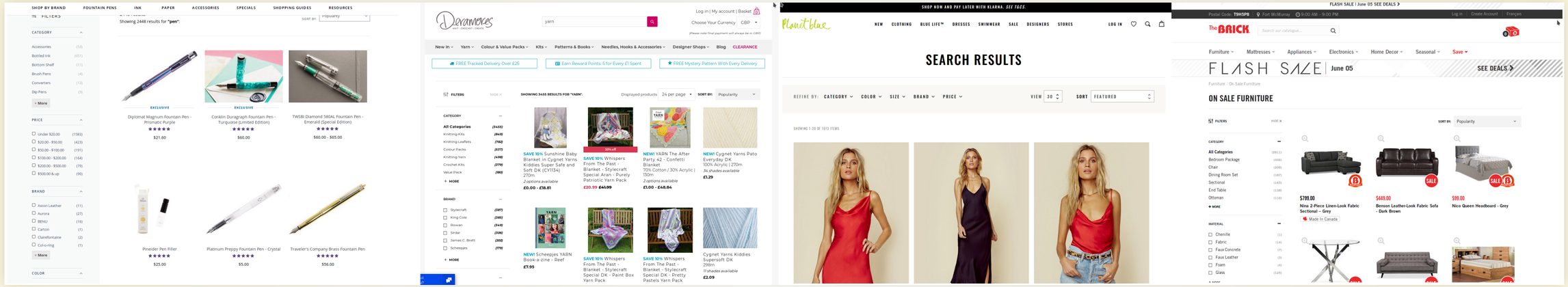
FLINK STATE SERIALIZATION

FOR BETTER PERFORMANCE AND SMALLER CHECKPOINT SIZE

Grebennikov Roman / findify.io / @public_void_grv / grv@dfdx.me

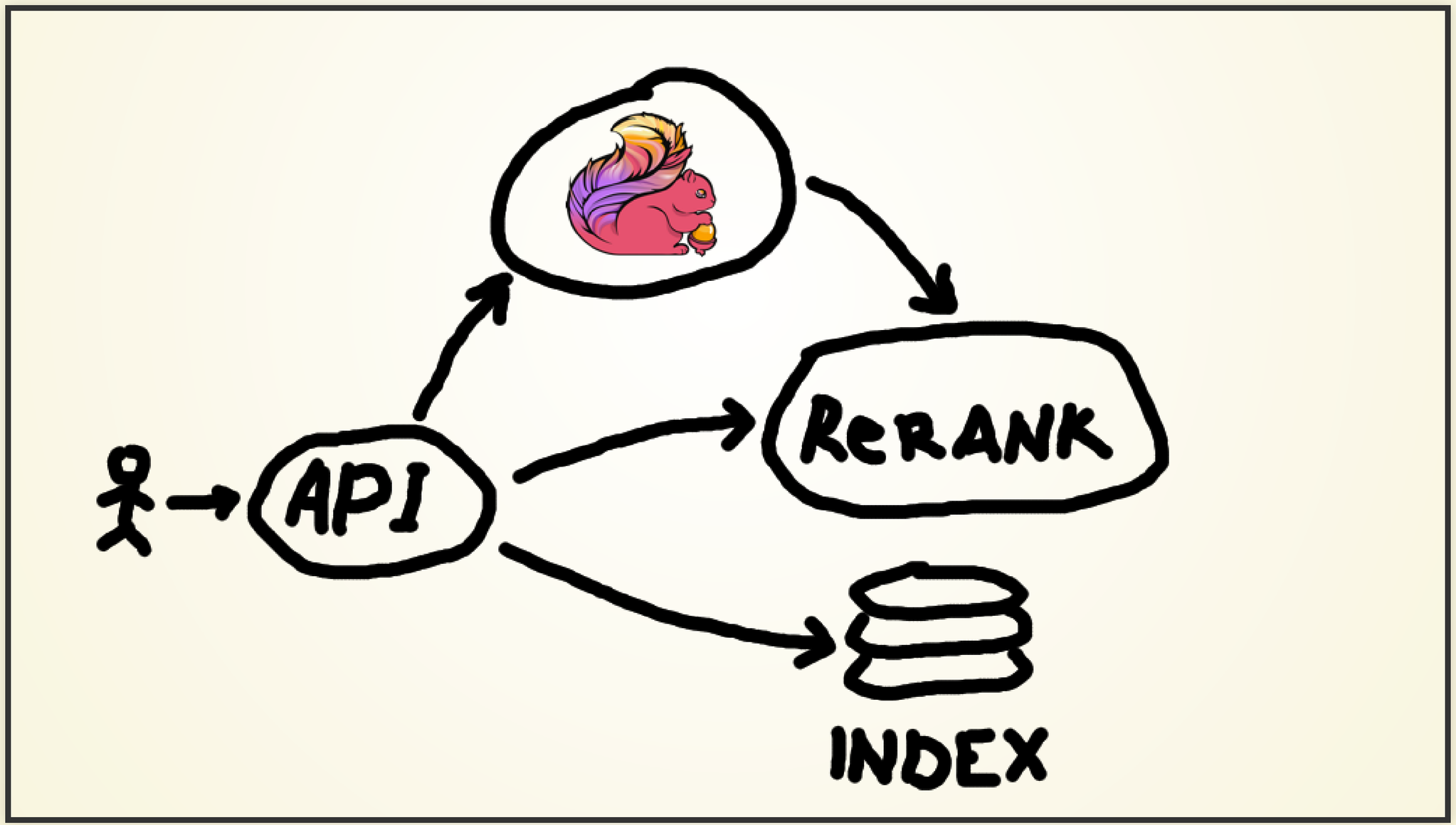
Flink Forward Berlin 2019

ABOUT FINDIFY



- white-label eCommerce SaaS search
- 1500 stores, 20M products
- 50M customers per month

DATA FLOW



STATEFUL AGGREGATIONS

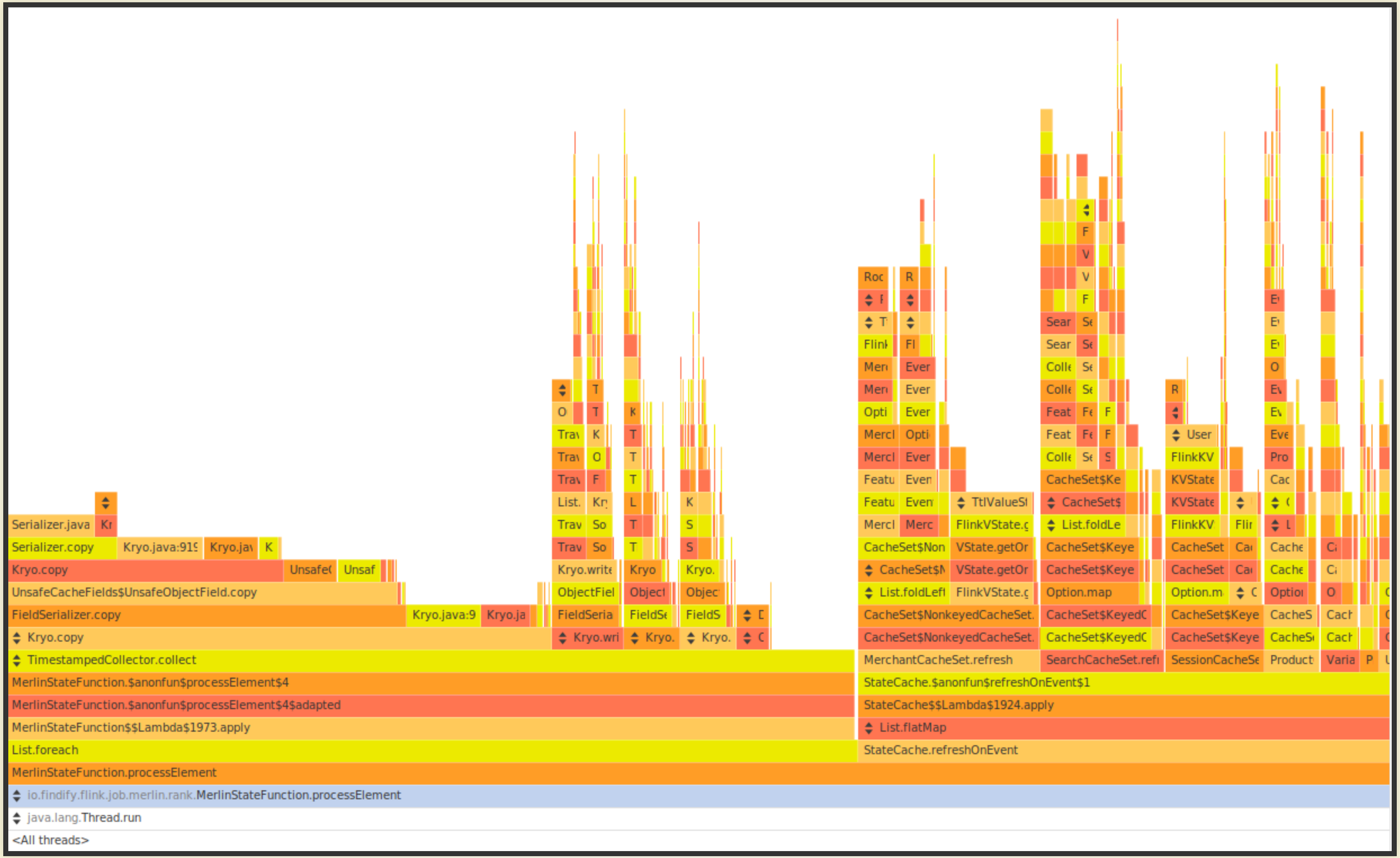
- scoped to merchant, query, product or visitor
- time windows of 1/2/4/7/14/30/60 days
- number of events
- q50/q90/q95 estimations

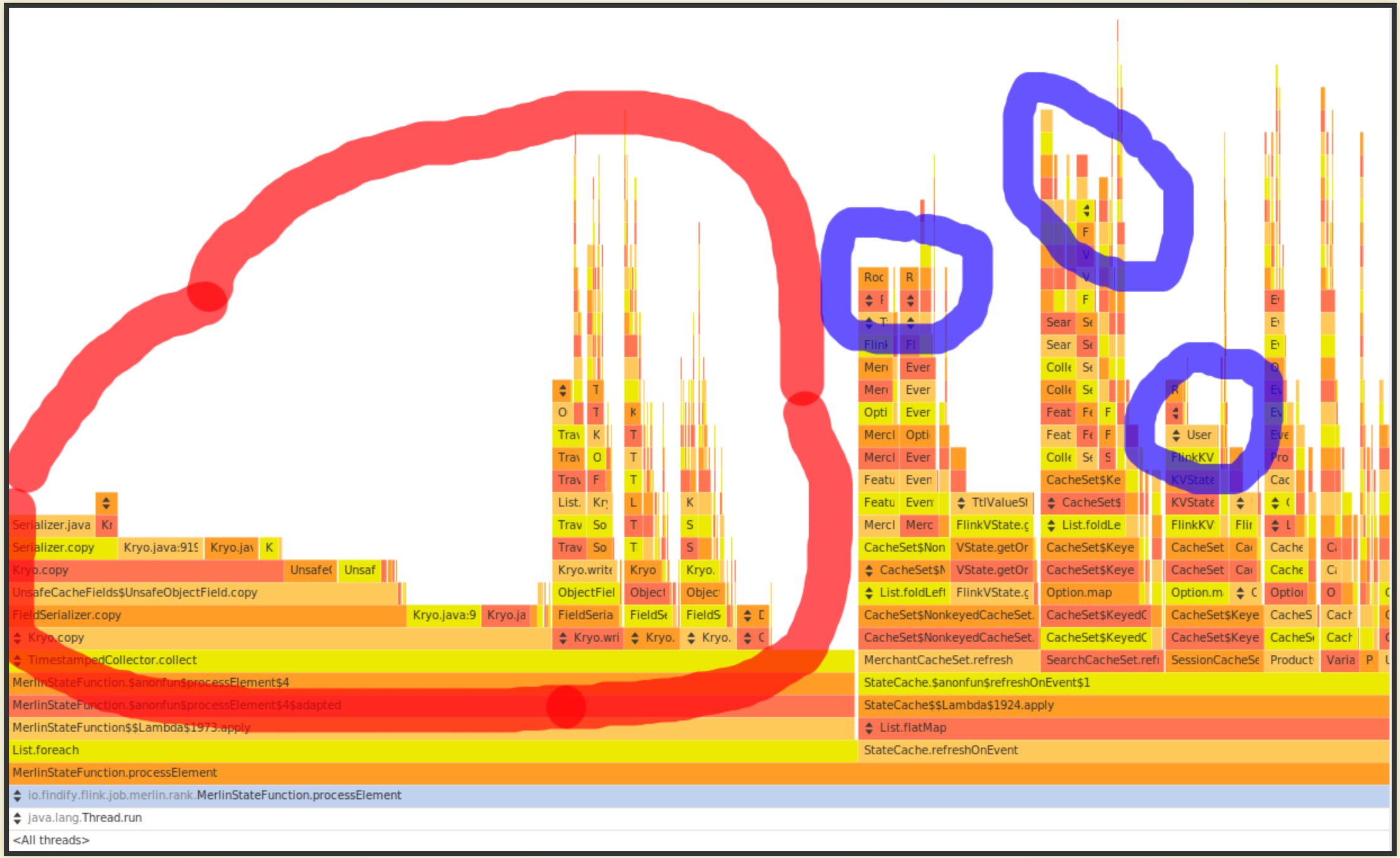
how to introduce a new aggregation?

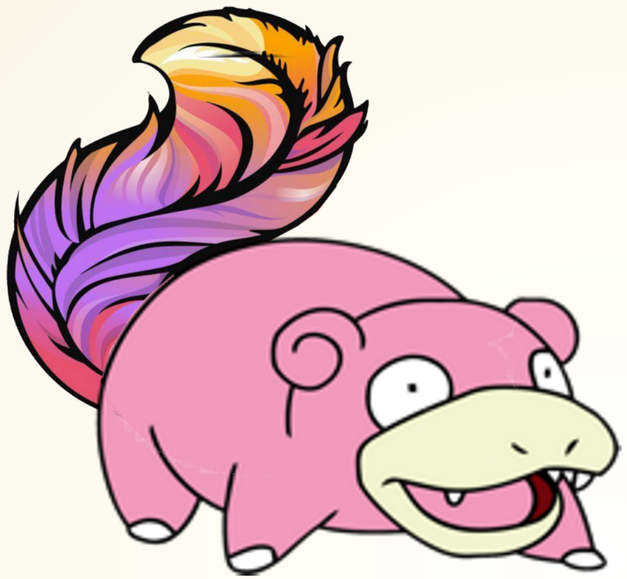
(RE-)BOOTSTRAPPING

- re-running a job on all the historical data
- ~10-50TB of data
- Too slow / too expensive

FLAME GRAPHS


Serializer.java
Serializer.copy
Kryo.copy
UnsafeCacheFields$UnsafeObjectField.copy
FieldSerializer.copy
Kryo.copy
TimestampedCollector.collect
MerlinStateFunction.$anonfun$processElement$4
MerlinStateFunction.$anonfun$processElement4adapted
MerlinStateFunction$$Lambda$1973.apply
List.foreach
MerlinStateFunction.processElement
io.findify.flink.job.merlin.rank.MerlinStateFunction.processElement
java.lang.Thread.run
<All threads>
Kryo.java:919
Kryo.java:9
CacheSet$NonkeyedCacheSet.
CacheSet$KeyedC
MerchantCacheSet.refresh
SearchCacheSet.refi
StateCache.$anonfun$refreshOnEvent$1
StateCache$$Lambda$1924.apply
List.flatMap
StateCache.refreshOnEvent
TtlValueSt
FlinkVState.g
VState.getOr
List.foldLe
Option.map
CacheSet$Keye
CacheSet$Ke
CacheSet$
FlinkKV
KVState
User
Pro
Cac
Cache
Option.m
CacheS
Cach
Produc
Varia
SessionCacheSe


Serializer.java
Serializer.copy
Kryo.copy
UnsafeCacheFields$UnsafeObjectField.copy
FieldSerializer.copy
Kryo.copy
TimestampedCollector.collect
MerlinStateFunction.$anonfun$processElement$4
MerlinStateFunction.$anonfun$processElement4adapted
MerlinStateFunction$$Lambda$1973.apply
List.foreach
MerlinStateFunction.processElement
io.findify.flink.job.merlin.rank.MerlinStateFunction.processElement
java.lang.Thread.run
<All threads>
Kryo.java:915
Kryo.java:9
Unsafe
Unsaf
Kryo.write
ObjectFiel
FieldSeria
Kryo.wri
Kryo.
FieldSe
Objec
FieldS
CacheSet$NonkeyedCacheSet.
CacheSet$NonkeyedCacheSet.
MerchantCacheSet.refresh
StateCache.$anonfun$refreshOnEvent$1
StateCache$$Lambda$1924.apply
List.flatMap
StateCache.refreshOnEvent
TtlValueSt
FlinkVState.g
VState.getOr
VState.getOr
FlinkVState.g
CacheSet$Keye
CacheSet$KeyedC
CacheSet$KeyedC
Option.map
SearchCacheSet.refi
CacheSet$Ke
CacheSet$
List.foldLe
CacheSet$Keye
CacheSet$Keye
SessionCacheSe
FlinkKV
KVState
KVState
KVState
CacheSet
CacheSet
Option.m
CacheS
CacheSe
Product
User
Pro
Varia



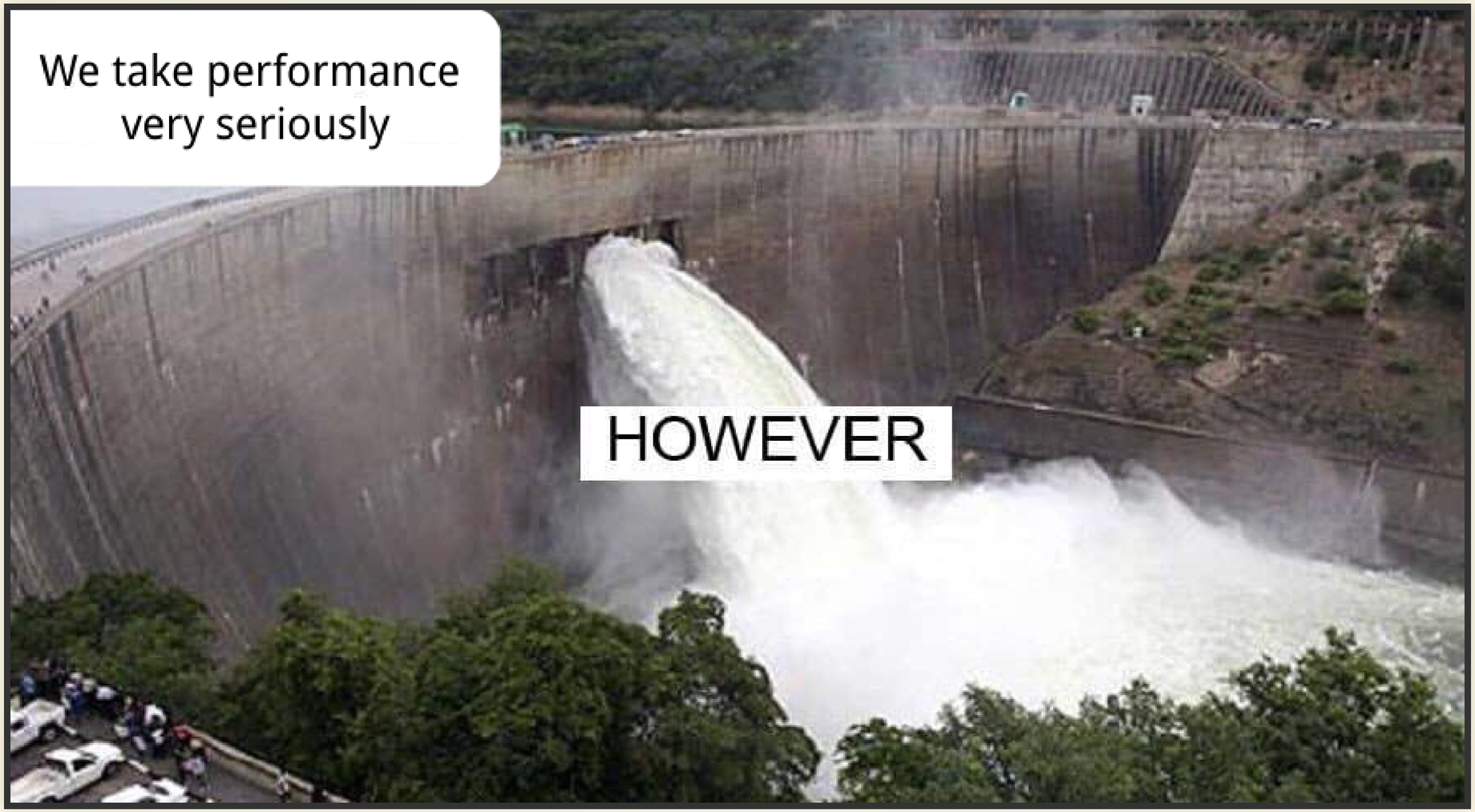
Flink

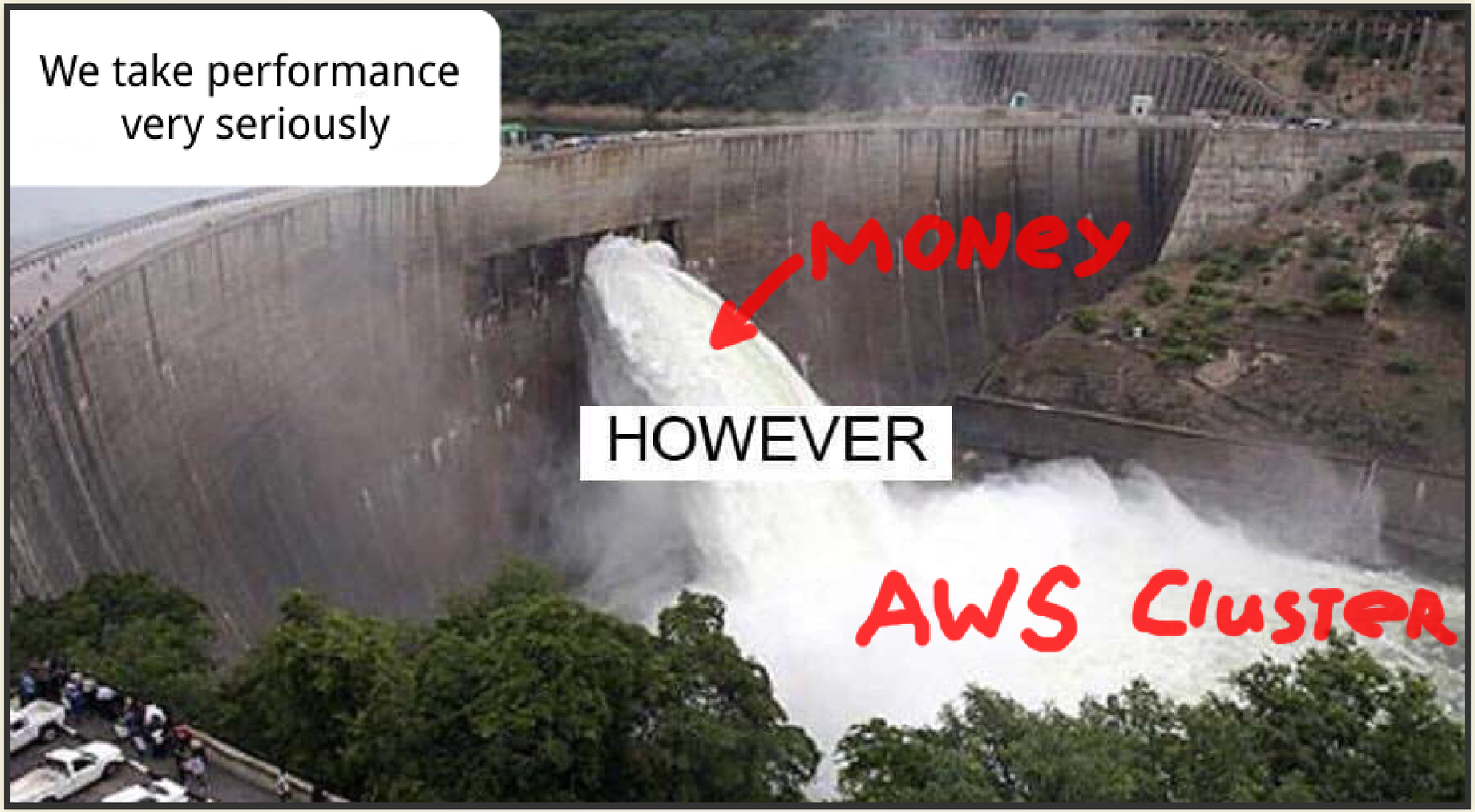
- 60% - ctx.collect()
- 30% - state.get() / state.put()
- 10% - business logic

CHECKING LOGS

```
[main] INFO org.apache.flink.api.java.typeutils.TypeExtractor -
       Class class com.foo.Bar cannot be used as a POJO type because
       not all fields are valid POJO fields, and must be processed as GenericType
```

- known type - predefined serializer
- POJO type - POJO/CaseClass serializer
- other type - Kryo fallback


We take performance very seriously
HOWEVER


We take performance very seriously
MONEY
HOWEVER
AWS CLUSTER

IS KRYO EXPENSIVE?

```
case class SimpleCase(a: String)
class SimpleClass(val a: String)

@Benchmark
def serializeCaseClass = serialize(SimpleCase("hello"), caseSerializer)

@Benchmark
def serializeClass = serialize(new SimpleClass("hello"), kryoSerializer)

@Benchmark
def serializeCustom = serialize(new SimpleClass("hello"), customKryoSerializer)
```

```
[info] Benchmark                                Mode  Cnt    Score   Error  Units
[info] SerdeBenchmark.serializeCaseClass        avgt   30   46.445 ± 1.861  ns/op
[info] SerdeBenchmark.serializeClass            avgt   30  178.688 ± 4.571  ns/op
[info] SerdeBenchmark.serializeCustom           avgt   30  147.989 ± 4.005  ns/op
```

RICH CLASSES

```
case class RichCase(user: String,
                    item: String,
                    ts: Long,
                    events: Array[Int],
                    flag: Boolean,
                    nested: SimpleCase)
```

```
[info] Benchmark                                Mode  Cnt    Score   Error  Units
[info] SerdeBenchmark.serializeRichCase         avgt   50  202.480 ± 6.384  ns/op
[info] SerdeBenchmark.serializeRichClass        avgt   50  497.562 ± 8.758  ns/op
```

ALGEBRAIC DATA TYPES IN SCALA

```
sealed trait Event
case class Pageview(item: String) extends Event
case class Purchase(item: String, price: BigDecimal) extends Event
```

BUT WHY?

```
env
   .fromElements(1,2,3)
   .map(_ * 2)
   .map(_.toString)
```

BUT WHY?

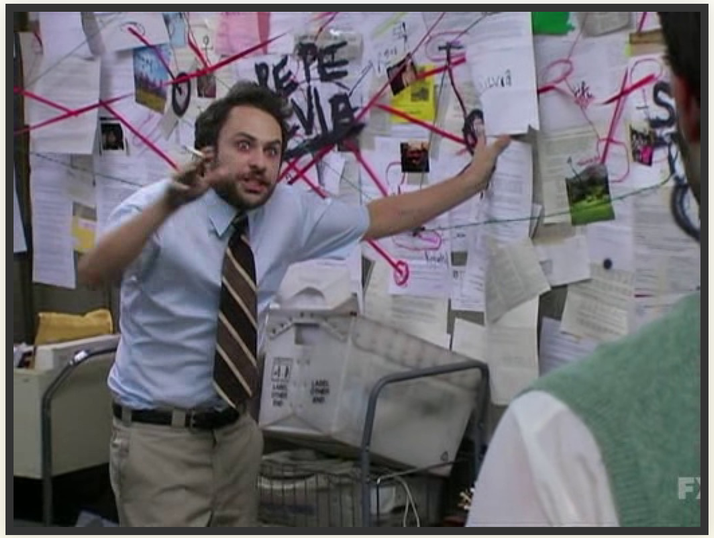
```
env
   .fromElements(1,2,3)
   .map(_ * 2)
   .map(_.toString) // <-- needs to know how to serialize this type

// defined as
def map[R: TypeInformation](fun: T => R): DataStream[R]
```

IMPORT ORG.APACHE.FLINK.API.SCALA._

- a magical macro function there
- generates the **TypeInformation[T]** for you

MACRO IS NOT A MAGIC

- compiler plugin
- inspects AST of your code
- for each implicit **TypeInformation**[T] request
- creates a **TypeInformation**[T] typeclass instance



```scala
def mkTypeInfo[T: c.WeakTypeTag](desc: UDTDescriptor): c.Expr[TypeInformation[T]] = desc match {

  case f: FactoryTypeDescriptor ⇒ mkTypeInfoFromFactory(f)

  case cc@CaseClassDescriptor(_, tpe, _, _, _) ⇒
    mkCaseClassTypeInfo(cc)(c.WeakTypeTag(tpe).asInstanceOf[c.WeakTypeTag[Product]])
      .asInstanceOf[c.Expr[TypeInformation[T]]]

  case tp: TypeParameterDescriptor ⇒ mkTypeParameter(tp)

  case p : PrimitiveDescriptor ⇒ mkPrimitiveTypeInfo(p.tpe)
  case p : BoxedPrimitiveDescriptor ⇒ mkPrimitiveTypeInfo(p.tpe)

  case n: NothingDescriptor ⇒
    reify { new ScalaNothingTypeInfo().asInstanceOf[TypeInformation[T]] }

  case u: UnitDescriptor ⇒ reify { new UnitTypeInfo().asInstanceOf[TypeInformation[T]] }

  case e: EitherDescriptor ⇒ mkEitherTypeInfo(e)

  case e: EnumValueDescriptor ⇒ mkEnumValueTypeInfo(e)

  case tr: TryDescriptor ⇒ mkTryTypeInfo(tr)

  case o: OptionDescriptor ⇒ mkOptionTypeInfo(o)

  case a : ArrayDescriptor ⇒ mkArrayTypeInfo(a)

  case l : TraversableDescriptor ⇒ mkTraversableTypeInfo(l)

  case v : ValueDescriptor ⇒
    mkValueTypeInfo(v)(c.WeakTypeTag(v.tpe).asInstanceOf[c.WeakTypeTag[Value]])
      .asInstanceOf[c.Expr[TypeInformation[T]]]

  case pojo: PojoDescriptor ⇒ mkPojo(pojo)

  case javaTuple: JavaTupleDescriptor ⇒ mkJavaTuple(javaTuple)

  case d ⇒ mkGenericTypeInfo(d)
}
```

ADT AND PERFORMANCE

```
sealed trait ADT
case class Foo(a: String) extends ADT
case class Bar(a: Int) extends ADT
```

```
[info] Benchmark                                  Mode  Cnt    Score   Error  Units
[info] SerdeBenchmark.serializeCaseClass          avgt   30   46.445 ± 1.861  ns/op
[info] SerdeBenchmark.serializeClass              avgt   30  178.688 ± 4.571  ns/op
[info] SerdeBenchmark.serializeADT                avgt   30  282.714 ± 5.060  ns/op
```

LOOK INSIDE!

```
00000000 57 01 01 82  61                                          W...a
```

```
00000000 01 00 6D 65  2E 64 66 64  78 2E 66 6C  .me.dfdx.fl
0000000C 69 6E 6B 73  74 61 74 65  2E 53 65 72  inkstate.Ser
00000018 64 65 42 65  6E 63 68 6D  61 72 6B 24  deBenchmark$
00000024 46 6F EF 01  01 82 61                   Fo....a
```

5 vs 43 bytes

ADT AND KRYO

```
sealed trait ADT
case class Foo(a: String) extends ADT
case class Bar(a: Int) extends ADT
```

- Kryo has no idea about concrete type
- Encodes the full runtime class name
- But it's a **sealed trait**, so scala compiler knows!

BRING YOUR OWN TYPEINFORMATION

- Make default TypeInformation for **Foo** and **Bar**
- Create custom **TypeInformation[ADT]**
- A lot of boilerplate code!

NESTED ADT

```
sealed trait ADT
case class Foo(a: String) extends ADT
case class Bar(a: Int) extends ADT
case class Batch(items: List[ADT])
```

NESTED ADT

```
sealed trait ADT
case class Foo(a: String) extends ADT
case class Bar(a: Int) extends ADT
case class Batch(items: List[ADT])

implicit val adtTypeInfo: TypeInformation[ADT] = ??? // <-- our own impl
implicit val batchTypeInfo = createTypeInformation[Batch]
```

```scala
def mkTypeInfo[T: c.WeakTypeTag](desc: UDTDescriptor): c.Expr[TypeInformation[T]] = desc match {

  case f: FactoryTypeDescriptor ⇒ mkTypeInfoFromFactory(f)

  case cc@CaseClassDescriptor(_, tpe, _, _, _) ⇒
    mkCaseClassTypeInfo(cc)(c.WeakTypeTag(tpe).asInstanceOf[c.WeakTypeTag[Product]])
      .asInstanceOf[c.Expr[TypeInformation[T]]]

  case tp: TypeParameterDescriptor ⇒ mkTypeParameter(tp)

  case p : PrimitiveDescriptor ⇒ mkPrimitiveTypeInfo(p.tpe)
  case p : BoxedPrimitiveDescriptor ⇒ mkPrimitiveTypeInfo(p.tpe)

  case n: NothingDescriptor ⇒
    reify { new ScalaNothingTypeInfo().asInstanceOf[TypeInformation[T]] }

  case u: UnitDescriptor ⇒ reify { new UnitTypeInfo().asInstanceOf[TypeInformation[T]] }

  case e: EitherDescriptor ⇒ mkEitherTypeInfo(e)

  case e: EnumValueDescriptor ⇒ mkEnumValueTypeInfo(e)

  case tr: TryDescriptor ⇒ mkTryTypeInfo(tr)

  case o: OptionDescriptor ⇒ mkOptionTypeInfo(o)

  case a : ArrayDescriptor ⇒ mkArrayTypeInfo(a)

  case l : TraversableDescriptor ⇒ mkTraversableTypeInfo(l)

  case v : ValueDescriptor ⇒
    mkValueTypeInfo(v)(c.WeakTypeTag(v.tpe).asInstanceOf[c.WeakTypeTag[Value]])
      .asInstanceOf[c.Expr[TypeInformation[T]]]

  case pojo: PojoDescriptor ⇒ mkPojo(pojo)

  case javaTuple: JavaTupleDescriptor ⇒ mkJavaTuple(javaTuple)

  case d ⇒ mkGenericTypeInfo(d)
}
```

MEET FLINK-SHAPELESS

https://github.com/joroKr21/flink-shapeless

*Flink-Shapeless replaces the default macro based implicit provider for **TypeInformation[T]** in Apache Flink's Scala API with automatic type class derivation based on Shapeless.*

- Dead and abandoned
- Shapeless and large case classes = 

MEET FLINK-ADT

https://github.com/findify/flink-adt

- nicely plays with Scala: case objects, nested types
- same performance as Flink
- can handle case classes with 100+ fields

demo

SMALL BENCHMARK

```
sealed trait ADT
case class Foo(a: String) extends ADT
case class Bar(a: Int) extends ADT
```

```
[info] Benchmark                          Mode  Cnt    Score   Error  Units
[info] ADTBenchmark.measureKryoADT        avgt   30  197.463 ± 3.912  ns/op
[info] ADTBenchmark.measureADT            avgt   30   38.621 ± 0.268  ns/op
[info] ADTBenchmark.measureFoo            avgt   30   39.103 ± 0.775  ns/op
```

CUSTOM SERIALIZERS

tradeoff between state size and throughput

- less compact, but faster String serializer
- var-length encoding of arrays of primitives

A TALE OF STRINGS

- identifiers of products, users, etc.
- 30% of all serialization wall time
- String - most popular type*

* - among field types in all Findify data models

DEFAULT STRING SERIALIZER

```
(cs != null) {
    // the length we write is offset by one, because a length
    int lenToWrite = cs.length()+1;
    if (lenToWrite < 0) {
        throw new IllegalArgumentException("CharSequence is t
    }

    // write the length, variable-length encoded
    while (lenToWrite >= HIGH_BIT) {
        out.write( i: lenToWrite | HIGH_BIT);
        lenToWrite >>>= 7;
    }
    out.write(lenToWrite);

    // write the char data, variable length encoded
    for (int i = 0; i < cs.length(); i++) {
        int c = cs.charAt(i);

        while (c >= HIGH_BIT) {
            out.write( i: c | HIGH_BIT);
            c >>>= 7;
        }
        out.write(c);
    }
```

SIMPLER STRINGSERIALIZER

```
override def deserialize(source: DataInputView): String = {
  source.readUTF()
}
override def serialize(record: String, target: DataOutputView): Unit = {
  target.writeUTF(record)
}
```

STRING SERIALIZATION

```
val string = "helloworld"
```

```
Benchmark                                                Mode  Cnt     Score   Error  Units
StringSerializerBenchmark.measureDefault @ jdk8          avgt   10   100.417 ± 1.452  ns/op
StringSerializerBenchmark.measureDefault @ jdk11         avgt   10    85.629 ± 1.418  ns/op

StringSerializerBenchmark.measureSimple @ jdk8           avgt   10    45.996 ± 0.607  ns/op
StringSerializerBenchmark.measureSimple @ jdk11          avgt   10    56.086 ± 0.984  ns/op
```

1.5x-2x faster!

STRING SERIALIZERS

```
(cs ≠ null) {
 // the length we write is offset by one, because a length
 int lenToWrite = cs.length()+1;
 if (lenToWrite < 0) {
     throw new IllegalArgumentException("CharSequence is t
 }

 // write the length, variable-length encoded
 while (lenToWrite ≥ HIGH_BIT) {
     out.write( i: lenToWrite | HIGH_BIT);
     lenToWrite >>>= 7;
 }
 out.write(lenToWrite);

 // write the char data, variable length encoded
 for (int i = 0; i < cs.length(); i++) {
     int c = cs.charAt(i);

     while (c ≥ HIGH_BIT) {
         out.write( i: c | HIGH_BIT);
         c >>>= 7;
     }
     out.write(c);
 }
```

```
int i;
for(i = 0; i < strlen; ++i) {
    c = str.charAt(i);
    if (c < 1 || c > 127) {
        break;
    }

    bytearr[var8++] = (byte)c;
}

for(; i < strlen; ++i) {
    c = str.charAt(i);
    if (c ≥ 1 && c ≤ 127) {
        bytearr[var8++] = (byte)c;
    } else if (c > 2047) {
        bytearr[var8++] = (byte)(224 | c >> 12 & 15);
        bytearr[var8++] = (byte)(128 | c >> 6 & 63);
        bytearr[var8++] = (byte)(128 | c >> 0 & 63);
    } else {
        bytearr[var8++] = (byte)(192 | c >> 6 & 31);
        bytearr[var8++] = (byte)(128 | c >> 0 & 63);
    }
}
```

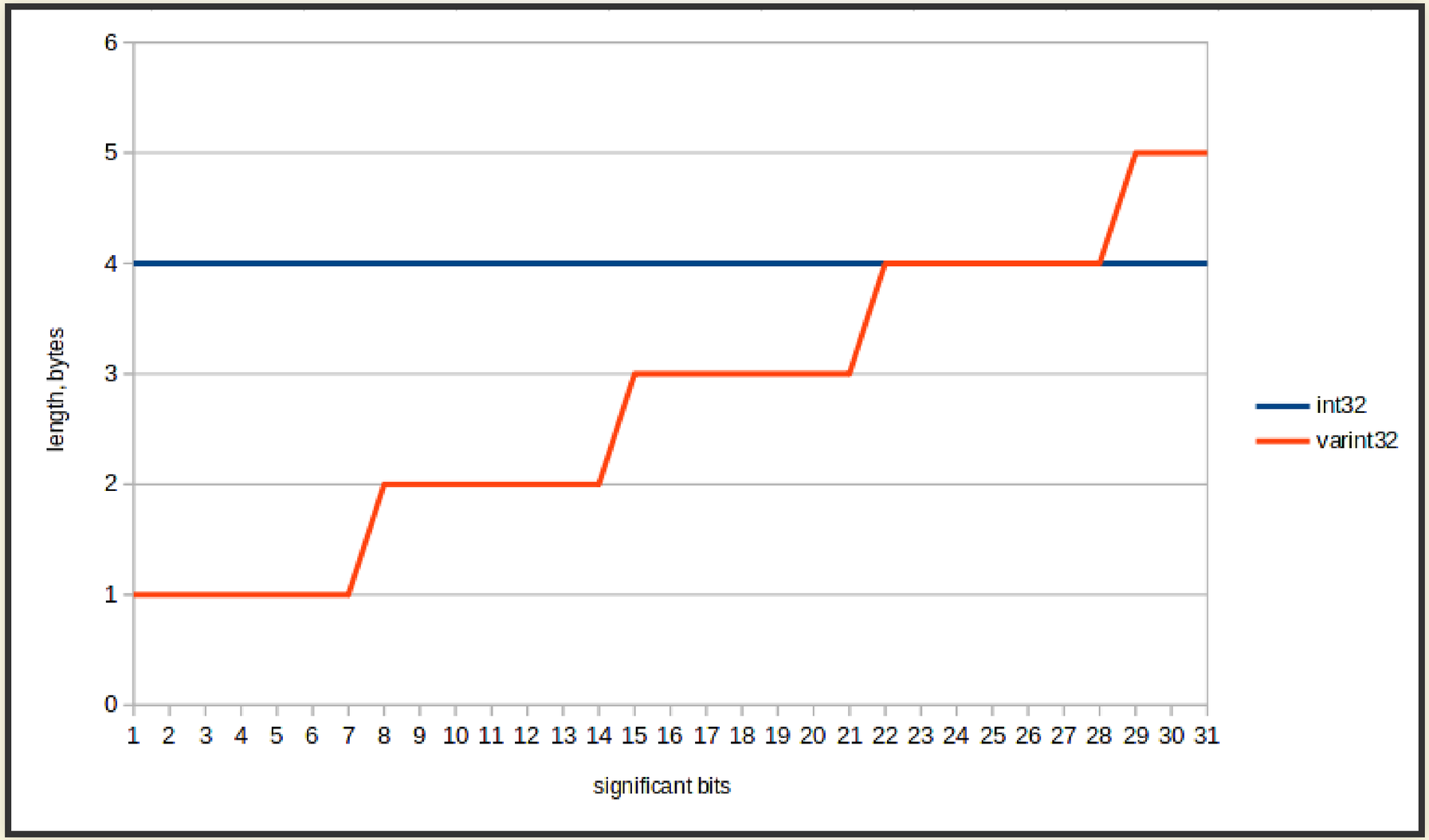
A TALE OF ARRAY[INT]

sliding window estimates of:

- counters -> per-day buckets
- percentiles -> count-min sketch

demo: varint

INTEGERS ARE NOT THE SAME



INTEGERS ARE NOT THE SAME

```
[info] Benchmark                                (BITS)  Mode  Cnt     Score    Error  Units
[info] VarLenBenchmark.measureDefault                7  avgt   50  1958.302 ± 15.122  ns/op
[info] VarLenBenchmark.measureDefault               14  avgt   50  1957.182 ± 10.889  ns/op
[info] VarLenBenchmark.measureDefault               21  avgt   50  1950.064 ± 11.593  ns/op
[info] VarLenBenchmark.measureDefault               28  avgt   50  1966.412 ± 12.555  ns/op
[info] VarLenBenchmark.measureDefault               31  avgt   50  1963.758 ± 12.369  ns/op

[info] VarLenBenchmark.measureVarInt                 7  avgt   50   501.154 ±  2.985  ns/op
[info] VarLenBenchmark.measureVarInt                14  avgt   50  1066.417 ± 25.846  ns/op
[info] VarLenBenchmark.measureVarInt                21  avgt   50  1581.119 ±  8.286  ns/op
[info] VarLenBenchmark.measureVarInt                28  avgt   50  2060.435 ± 11.029  ns/op
[info] VarLenBenchmark.measureVarInt                31  avgt   50  2562.302 ± 15.398  ns/op
```

CONDITIONAL WRITES!

```
public final void writeInt(int var1) throws IOException {
    this.out.write(var1 >>> 24 & 255);
    this.out.write(var1 >>> 16 & 255);
    this.out.write(var1 >>> 8 & 255);
    this.out.write(var1 >>> 0 & 255);
```

```
public static void putVarInt(int v, DataOutput sink) throws IOException {
    while (true) {
        int bits = v & 0x7f;
        v >>>= 7;
        if (v == 0) {
            sink.writeByte((byte) bits);
            return;
        }
        sink.writeByte((byte) (bits | 0x80));
    }
```

RESULTS

- 2.5x reduction in state size
- 4x increase in throughput

RESULTS

- 2.5x reduction in state size
- 4x increase in throughput

FUTURE

- FLINK-14346 - String serialization performance improvement
- Var-length encoding for primitives
- ADT support in flink-scala mkTypeInfo?

LINKS

- benchmarks: https://github.com/shuttie/flink-state-benchmarks
- library: https://github.com/findify/flink-adt


That's all Folks!