

Scala and the JVM for Big Data: Lessons from Spark

polyglotprogramming.com/talks
dean.wampler@lightbend.com
[@deanwampler](https://twitter.com/deanwampler)

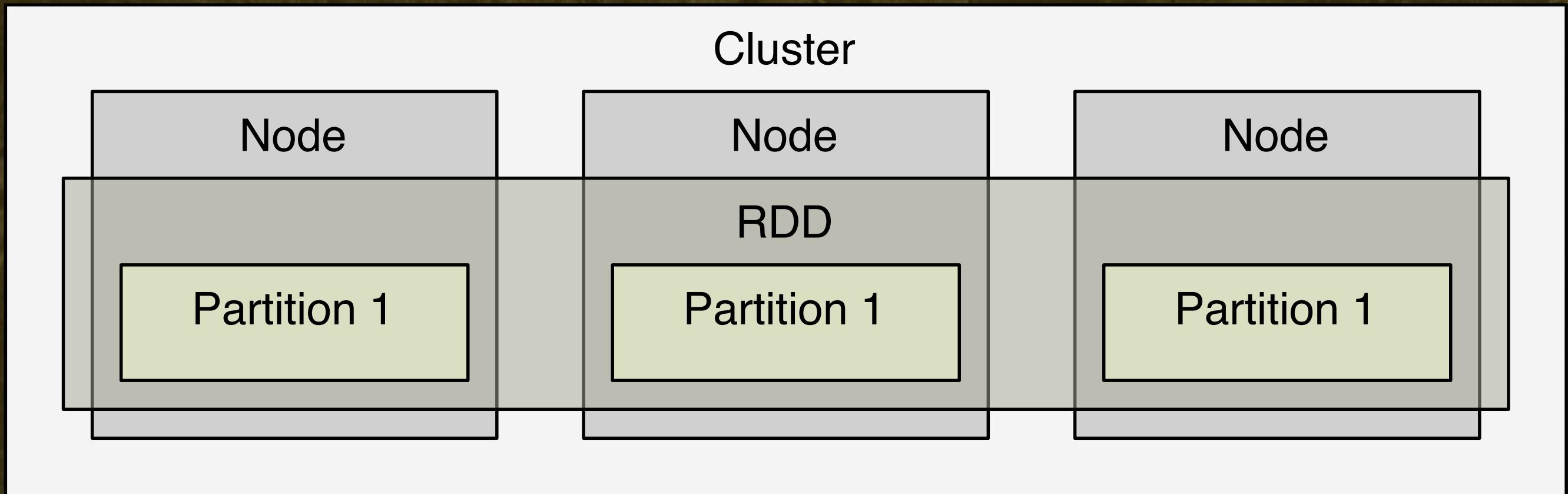


Lightbend

A photograph of a lush, green forest. In the foreground, two hikers wearing hats and backpacks walk away from the camera on a path covered in green ferns and moss. The forest is filled with tall, mossy trees, their trunks and branches heavily draped in vibrant green moss. Sunlight filters through the canopy, creating dappled light and shadow on the forest floor.

spark

A Distributed Computing Engine on the JVM



Resilient Distributed Datasets

Productivity?

Very concise, elegant, functional APIs.

- Scala, Java
- Python, R
- ... and SQL!

Productivity?

Interactive shell (REPL)

- Scala, Python, R, and SQL

Notebooks

- Jupyter
- Spark Notebook
- Zeppelin
- Beaker
- Databricks

The screenshot shows a web-based notebook interface for Spark. At the top, there's a header with a logo, the title "SPARK NOTEBOOK WhyScala", and a navigation bar with links like "File", "Edit", "View", "Insert", "Cell", "Kernel", and "Help". Below the header is a toolbar with various icons for file operations, cell execution, and navigation. The main content area contains a section titled "Scala: the Unpredicted Lingua Franca for Data Science" by Andy Petrella and Dean Wampler, with their contact information and a list of events where the notebook was presented. A note at the bottom indicates it's available on GitHub. The background of the slide features a photograph of two people walking through a dense forest.

Scala: the Unpredicted Lingua Franca for Data Science

Andy Petrella
nootsab@data-fellas.guru

Dean Wampler
dean.wampler@lightbend.com

- Scala Days NYC, May 5th, 2016
- GOTO Chicago, May 24, 2016
- Strata + Hadoop World London, June 3, 2016
- Scala Days Berlin, June 16th, 2016

This notebook available at github.com/data-fellas/scala-for-data-science.

Why Scala for Data Science with Spark?

While Python and R are traditional languages of choice for Data Science, [Spark](#) also supports Scala (the language in which it's written) and Java.

However, using one language for all work has advantages like simplifying the software development process, such as build and deployment tools, coding conventions, etc.



A photograph of a forest floor covered in a thick layer of green moss and various small, leafy plants. The scene is dense and textured, with sunlight filtering through the trees above.

Example: Inverted Index

Web Crawl

wikipedia.org/hadoop

Hadoop provides
MapReduce and HDFS

...

wikipedia.org/hbase

HBase stores data in HDFS

index

block

...

wikipedia.org/hadoop

...

...

Hadoop provides...

block

...

wikipedia.org/hbase

...

...

HBase stores...

...

Compute Inverted Index

index

block

| | |
|----------------------|--------------------|
| ... | ... |
| wikipedia.org/hadoop | Hadoop provides... |
| ... | ... |

block

| | |
|---------------------|-----------------|
| ... | ... |
| wikipedia.org/hbase | HBase stores... |
| ... | ... |

block

| | |
|--------------------|-----------------|
| ... | ... |
| wikipedia.org/hive | Hive queries... |
| ... | ... |

inverse index

block

| | |
|--------|---|
| ... | ... |
| hadoop | (.../hadoop,1) |
| hbase | (.../hbase,1),(.../hive,1) |
| hdfs | (.../hadoop,1),(.../hbase,1),(.../hive,1) |
| hive | (.../hive,1) |
| ... | ... |

block

| | |
|-----|-----|
| ... | ... |
| ... | ... |

block

| | |
|-----|-----|
| ... | ... |
| ... | ... |

block

| | |
|-----|-----|
| ... | ... |
| ... | ... |

Miracle!!

Inverted Index

inverse index

block

| | |
|--------|---|
| ... | ... |
| hadoop | (.../hadoop,1) |
| hbase | (.../hbase,1),(.../hive,1) |
| hdfs | (.../hadoop,1),(.../hbase,1),(.../hive,1) |
| hive | (.../hive,1) |
| ... | ... |

oracle!!

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._

val sparkContext = new SparkContext(master, "Inv. Index")
sparkContext.textFile("/path/to/input").
map { line =>
  val array = line.split(",", 2)
  (array(0), array(1)) // (id, content)
}.flatMap {
  case (id, content) =>
    toWords(content).map(word => ((word,id),1)) // toWords not shown
}.reduceByKey(_ + _).
map {
  case ((word,id),n) => (word,(id,n))
}.groupByKey.
mapValues {
  seq => sortByCount(seq) // Sort the value seq by count, desc.
}.saveAsTextFile("/path/to/output")
```

```
import org.apache.spark.SparkContext  
import org.apache.spark.SparkContext._
```

```
val sparkContext = new  
  SparkContext(master, "Inv. Index")  
sparkContext.textFile("/path/to/input").  
map { line =>  
  val array = line.split(",", 2)  
  (array(0), array(1))  
}.flatMap {  
  case (id, contents) =>  
    toWords(contents).map(w => ((w, id), 1))  
}
```

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._

val sparkContext: SparkContext = new SparkContext("local[2]", "WordCount")
sparkContext.textFile("/path/to/input").  

map { line =>  

    val array = line.split(" ", 2)  

    (array(0), array(1))  

}.flatMap {  

    case (id, contents) => new RDD[(String, String)](sc)  

    toWords(contents).map(w => ((w, id), 1))  

}
```

```
var array = line.split( , , 2)
(array(0), array(1))
}.flatMap {
  case (id, contents) =>
    toWords(contents).map(w => ((w,id),1))
}.reduceByKey(_ + _).
map {
  case ((word, id), count) => (word, (id, count))
}.groupByKey.
mapValues {
  seq => sortByCount(seq)
}.saveAsTextFile("/path/to/output")
```

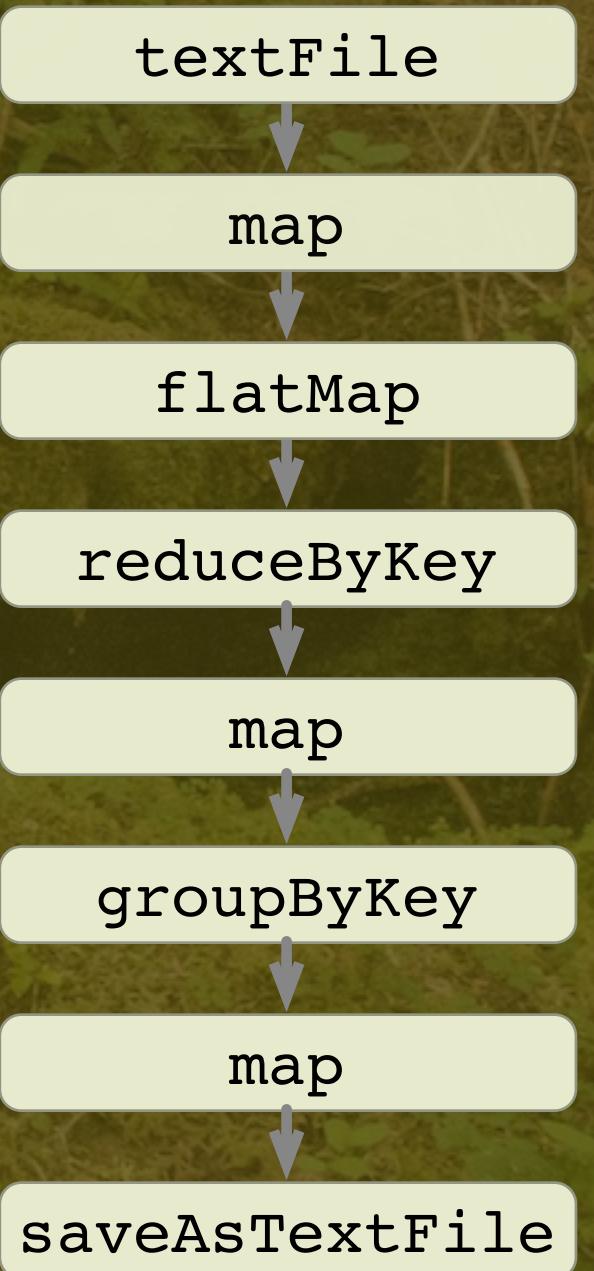
```
val array = iLine.split( , , 2)
(array(0), array(1))
}.flatMap {
  case (id, contents) =>
    toWords(contents).map(w => ((w, id), 1))
}.reduceByKey(_ + _).
map {
  case ((word, id), n) => (word, (id, n))
}.groupByKey.
mapValues {
  seq => RDD[(String,Iterable((String,Int)))]: (Hadoop,seq(.../hadoop,20),...)
}.saveAsTextFile("/path/to/output")
```

```
val array = line.split( , , 2)
(array(0), array(1))
}.flatMap {
  case (id, contents) =>
    toWords(contents).map(w => ((w, id), 1))
}.reduceByKey(_ + _).
map {
  case ((word, id), n) => (word, (id, n))
}.groupByKey.
mapValues {
  seq => sortByCount(seq)
}.saveAsTextFile("/path/to/output")
```

Productivity?

Intuitive API:

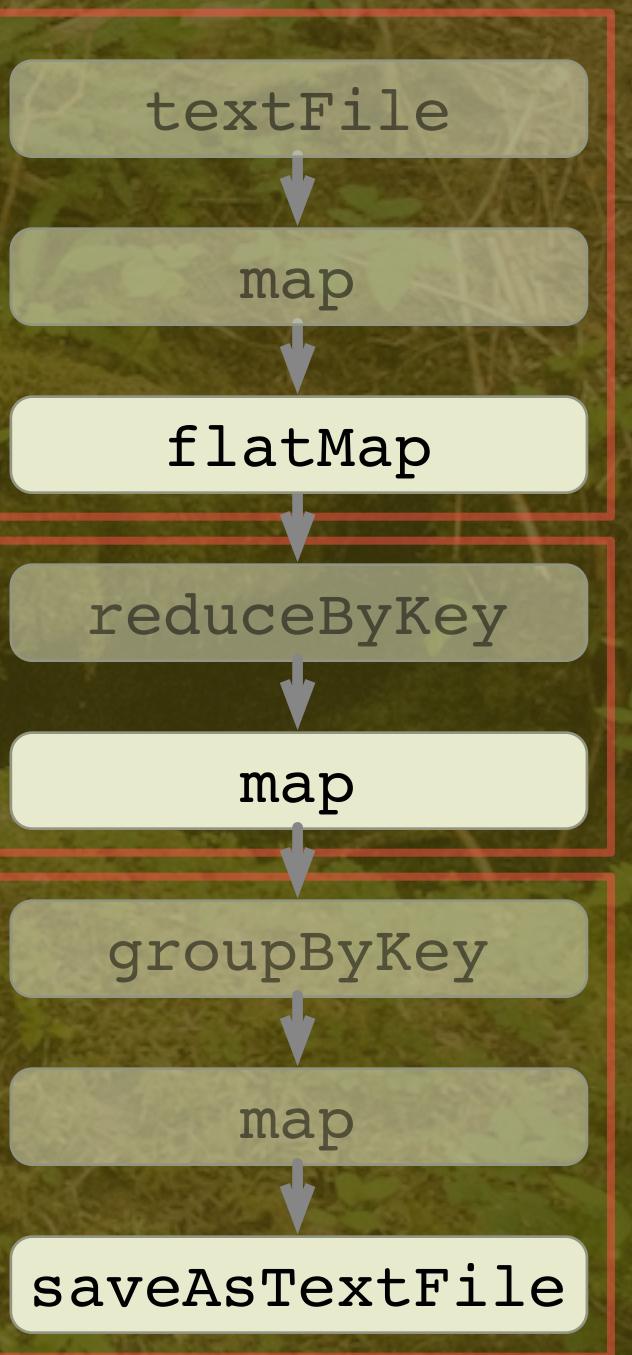
- Dataflow of steps.
- Inspired by Scala collections and functional programming.



Performance?

Lazy API:

- Combines steps into “stages”.
- Cache intermediate data in memory.





Higher-Level APIs

A photograph of a dense forest with tall, thin trees and two hikers walking away from the camera on a path.

A scenic mountain landscape featuring a clear blue lake nestled among green forests and rocky terrain. In the foreground, a hiker in a blue shirt and backpack walks along a stone path through a lush green field. The background shows a range of mountains under a bright blue sky with scattered clouds.

SQL:
Datasets/
DataFrames

Example

```
import org.apache.spark.SparkSession
val spark = SparkSession.builder()
    .master("local")
    .appName("Queries")
    .getOrCreate()

val flights =
  spark.read.parquet(".../flights")
val planes =
  spark.read.parquet(".../planes")
flights.createOrReplaceTempView("flights")
planes. createOrReplaceTempView("planes")
flights.cache(); planes.cache()

val planes_for_flights1 = sqlContext.sql("""
  SELECT * FROM flights f
  JOIN planes p ON f.tailNum = p.tailNum LIMIT 100""")
val planes_for_flights2 =
  flights.join(planes,
    flights("tailNum") ===
    planes ("tailNum")).limit(100)
```

```
import org.apache.spark.SparkSession  
val spark = SparkSession.builder()  
  .master("local")  
  .appName("Queries")  
  .getOrCreate()
```

```
val flights =  
  spark.read.parquet(".../flights")  
val planes =  
  spark.read.parquet(".../planes")  
flights.createOrReplaceTempView("flights")  
planes.createOrReplaceTempView("planes")  
flights.cache(); planes.cache()
```

```
import org.apache.spark.SparkSession  
val spark = SparkSession.builder()  
  .master("local")  
  .appName("Queries")  
  .getOrCreate()
```

```
val flights =  
  spark.read.parquet(".../flights")  
val planes =  
  spark.read.parquet(".../planes")  
flights.createOrReplaceTempView("flights")  
planes.createOrReplaceTempView("planes")  
flights.cache(); planes.cache()
```

```
planes.createOrReplaceTempView("planes")
flights.cache(); planes.cache()
```

```
val planes_for_flights1 = sqlContext.sql("""
    SELECT * FROM flights f
    JOIN planes p ON f.tailNum = p.tailNum
LIMIT 100""")
```

Returns another
Dataset.

```
val planes_for_flights2 =
  flights.join(planes,
    flights("tailNum") ===
    planes ("tailNum")).limit(100)
```

```
planes.createOrReplaceTempView("planes")
flights.cache(); planes.cache()
```

```
val planes_for_flights1 = sqlContext.sql("""
    SELECT * FROM flights f
    JOIN planes p ON f.tailNum = p.tailNum
LIMIT 100""")
```

Returns another
Dataset.

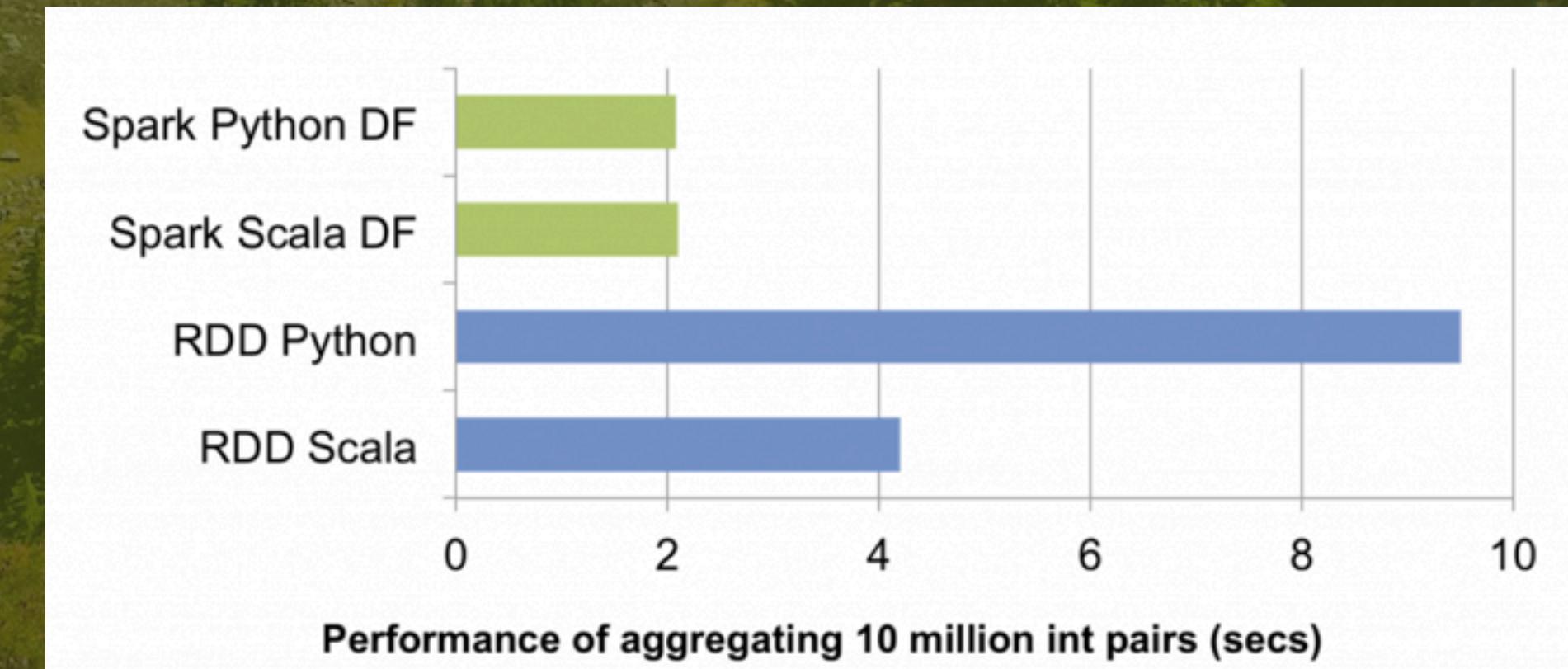
```
val planes_for_flights2 =
  flights.join(planes,
    flights("tailNum") ===
    planes ("tailNum")).limit(100)
```

```
val planes_for_flights2 =  
  flights.join(planes,  
    flights("tailNum") ===  
    planes ("tailNum")).limit(100)
```

Not an “arbitrary”
anonymous function, but a
“Column” instance.

Performance

The Dataset API has the same performance for all languages:
Scala, Java,
Python, R,
and SQL!



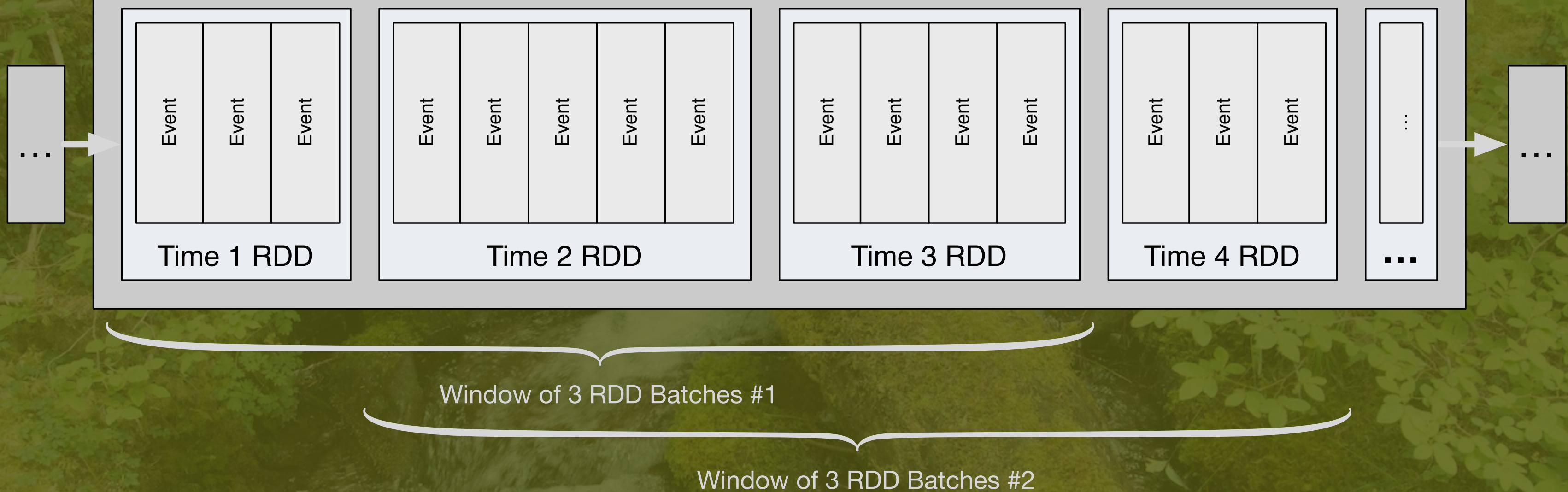
```
def join(right: Dataset[_], joinExprs: Column): DataFrame = {  
  def groupBy(cols: Column*): RelationalGroupedDataset = {  
    def orderBy(sortExprs: Column*): Dataset[T] = {  
      def select(cols: Column*): Dataset[...] = {  
        def where(condition: Column): Dataset[T] = {  
          def limit(n: Int): Dataset[T] = {  
            def intersect(other: Dataset[T]): Dataset[T] = {  
              def sample(withReplacement: Boolean, fraction, seed) = {  
                def drop(col: Column): DataFrame = {  
                  def map[U](f: T => U): Dataset[U] = {  
                    def flatMap[U](f: T => Traversable[U]): Dataset[U] = {  
                      def foreach(f: T => Unit): Unit = {  
                        def take(n: Int): Array[Row] = {  
                          def count(): Long = {  
                            def distinct(): Dataset[T] = {  
                              def agg(exprs: Map[String, String]): DataFrame = {
```



A photograph of a lush green forest. In the foreground, there's a rocky path or stream bed covered in moss and small plants. Several large, mossy tree trunks stand prominently, their bark textured and dark. Sunlight filters through the canopy of leaves above, creating bright highlights and deep shadows.

Structured Streaming

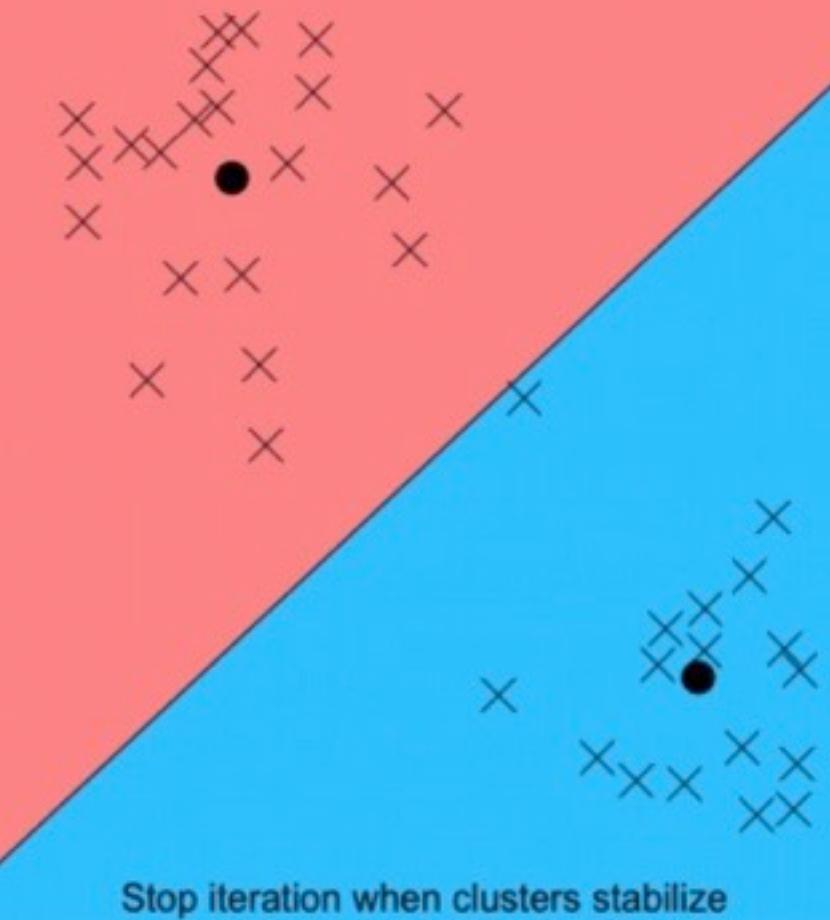
DStream (discretized stream)



A wide-angle photograph of a mountain range during sunset or sunrise. The mountains are silhouetted against a bright sky with scattered clouds. In the foreground, a grassy hillside is dotted with several tall evergreen trees. The overall atmosphere is serene and natural.

ML/MLib

K-Means

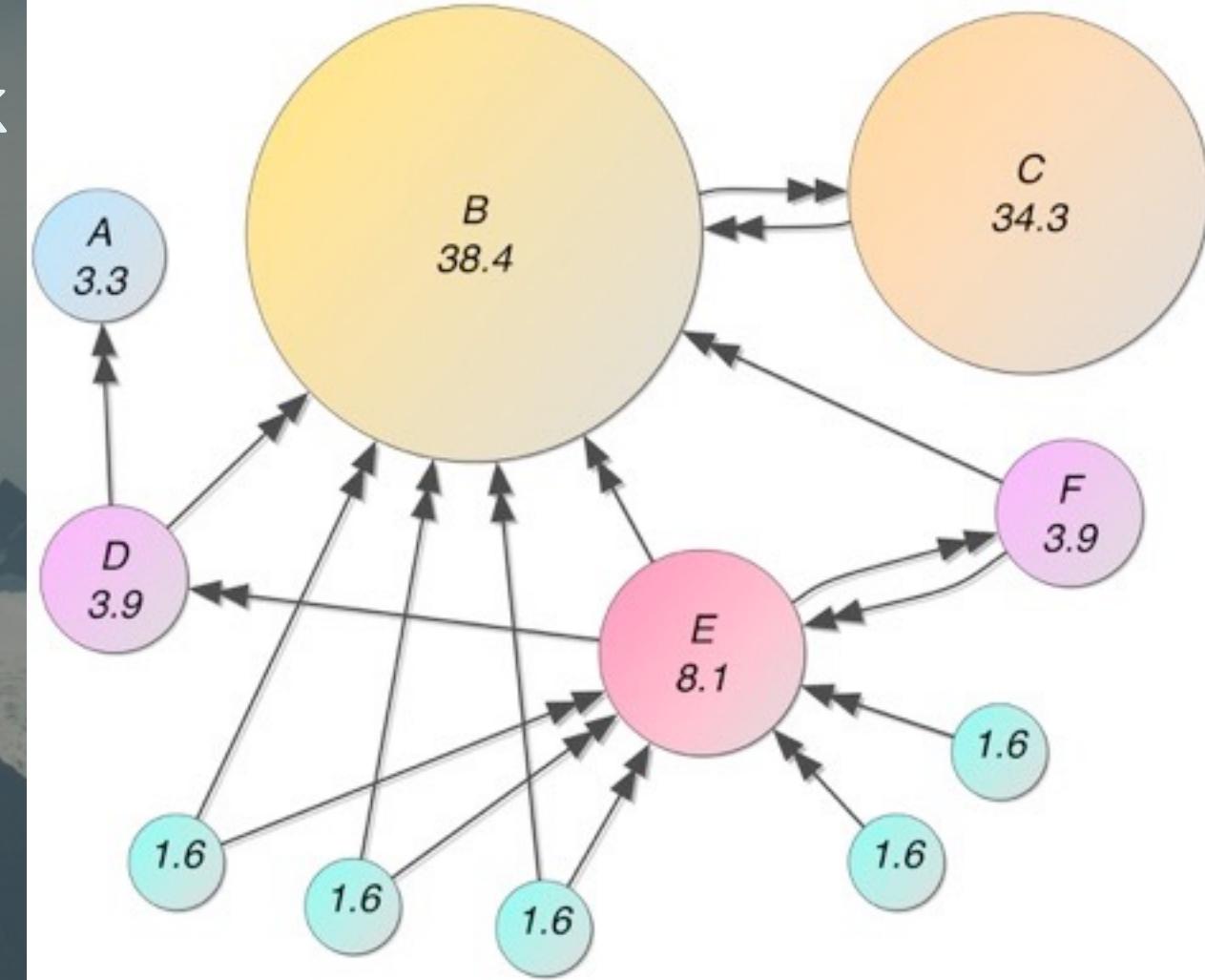


- Machine Learning requires:
 - Iterative training of models.
 - Good linear algebra perf.



GraphX

PageRank



- Graph algorithms require:
 - Incremental traversal.
 - Efficient edge and node reps.

Foundation:

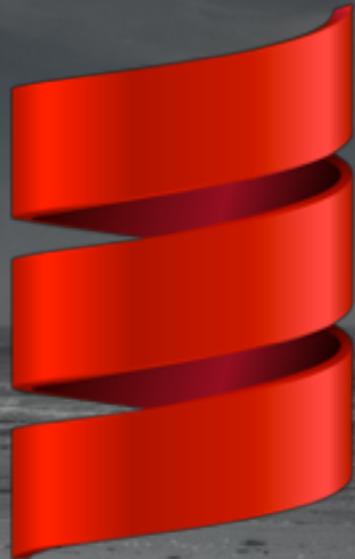
The JVM

20 Years of DevOps

A black and white photograph of a beach at low tide. A lone figure is walking away from the camera towards the horizon. In the foreground, a green rope lies across the sand. The sky is filled with dramatic, layered clouds.

Lots of Java Devs

Tools and Libraries



Akka

Breeze

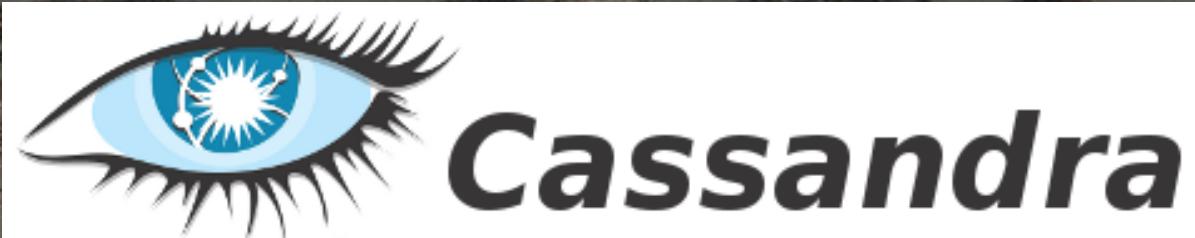
Algebird

Spire & Cats

Axle

...

Big Data Ecosystem



A dark, moody landscape featuring a large body of water in the foreground, a dense forested mountain on the right, and a range of mountains in the background partially obscured by low-hanging clouds.

But it's
not perfect....

A close-up photograph of a large pile of dark brown, textured kelp or seaweed floating in clear, shallow water. The seaweed is tangled and piled high, with some bright yellowish-orange coloration from smaller organisms like anemones or barnacles. The water is calm, reflecting the light.

Richer data IIBs.
in Python & R

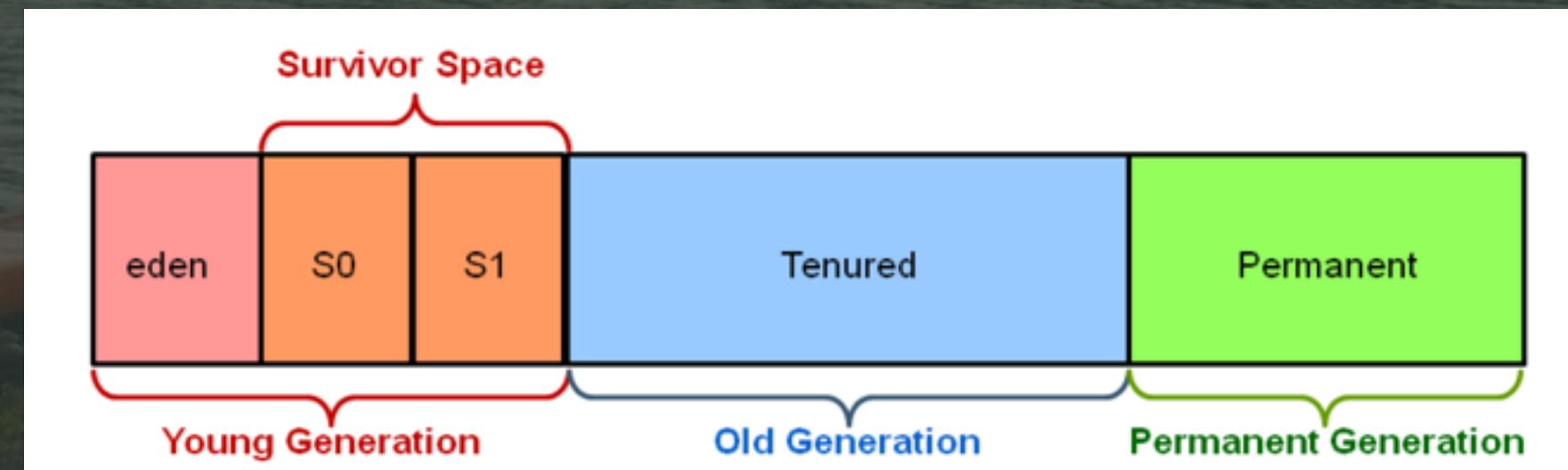
Garbage Collection

GC Challenges

- Typical Spark heaps: 10s-100s GB.
- Uncommon for “generic”, non-data services.

GC Challenges

- Too many cached RDDs leads to huge old generation garbage.
- Billions of objects => long GC pauses.



Tuning GC

- Best for Spark:
 - -XX:UseG1GC -XX:-ResizePLAB -
Xms... -Xmx... -
XX:InitiatingHeapOccupancyPerce
nt=... -XX:ConcGCThread=...

[databricks.com/blog/2015/05/28/tuning-java-garbage-collection-for-spark-
applications.html](https://databricks.com/blog/2015/05/28/tuning-java-garbage-collection-for-spark-applications.html)

JVM Object Model

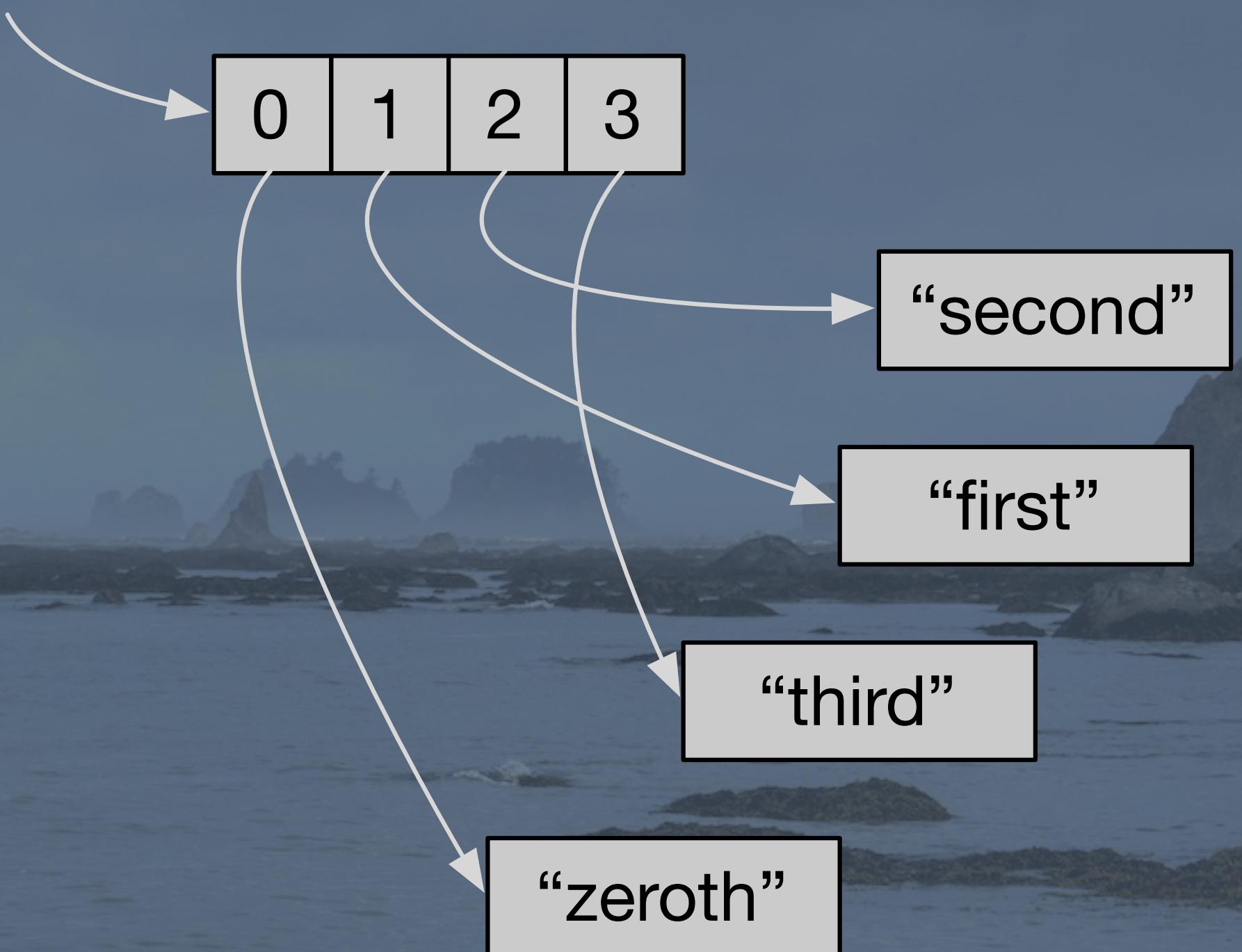


Java Objects?

- “abcd”: 4 bytes for raw UTF8, right?
- 48 bytes for the Java object:
 - 12 byte header.
 - 8 bytes for hash code.
 - 20 bytes for array overhead.
 - 8 bytes for UTF16 chars.

val myArray: Array[String]

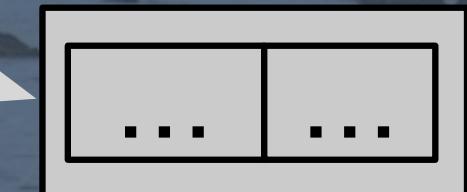
Arrays



val person: Person

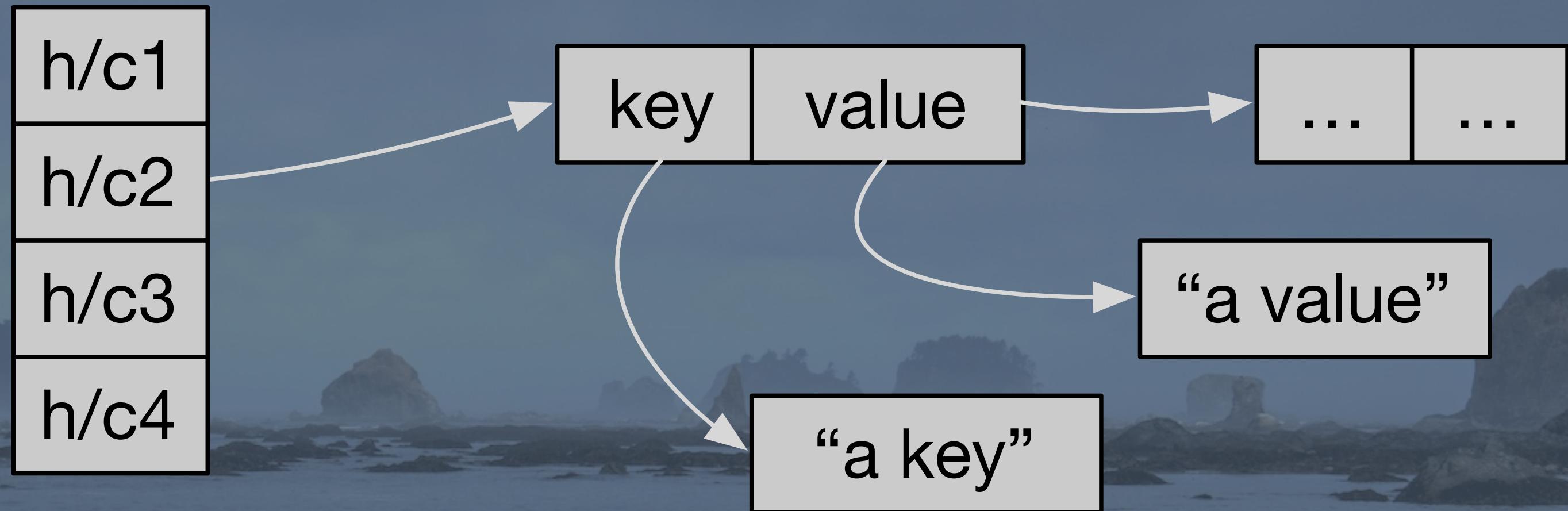
| | |
|---------------|----|
| name: String | |
| age: Int | 29 |
| addr: Address | |

“Buck Trends”



Class Instances

Hash Map



Hash Maps

Improving Performance

Why obsess about this?

Spark jobs are CPU bound:

- Improve network I/O? ~2% better.
- Improve disk I/O? ~20% better.

What changed?

- Faster HW (compared to ~2000)
 - 10Gbs networks
 - SSDs.

What changed?

- Smarter use of I/O
 - Pruning unneeded data sooner.
 - Caching more effectively.
 - Efficient formats, like Parquet.

What changed?

- But more CPU use today:
 - More Serialization.
 - More Compression.
 - More Hashing (joins, group-bys).

Improving Performance

To improve performance, we need to focus on the CPU, the:

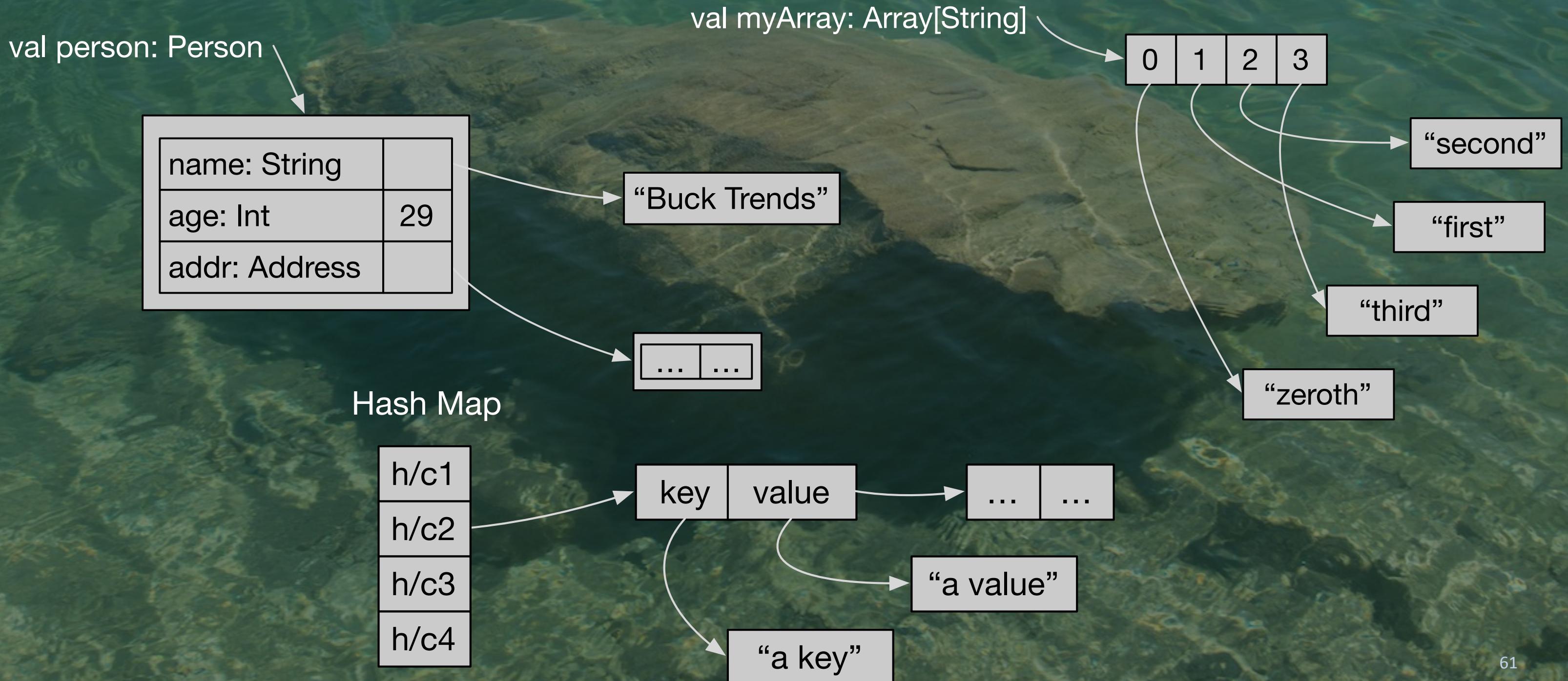
- Better algorithms, sure.
- And optimize use of memory.

Project Tungsten

Initiative to greatly improve
Dataset/DataFrame performance.

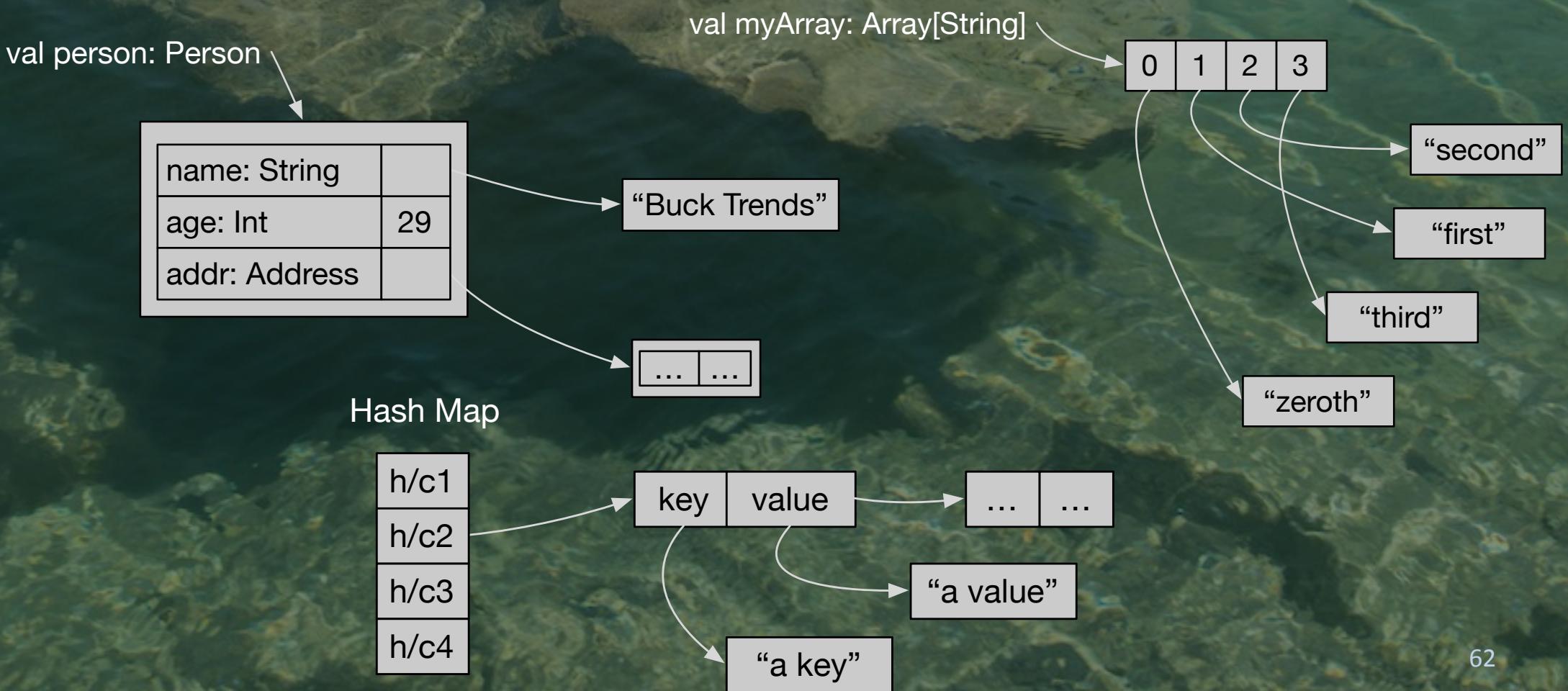
Goals

Reduce References



Reduce References

- Fewer, bigger objects to GC.
- Fewer cache misses



Less Expression Overhead

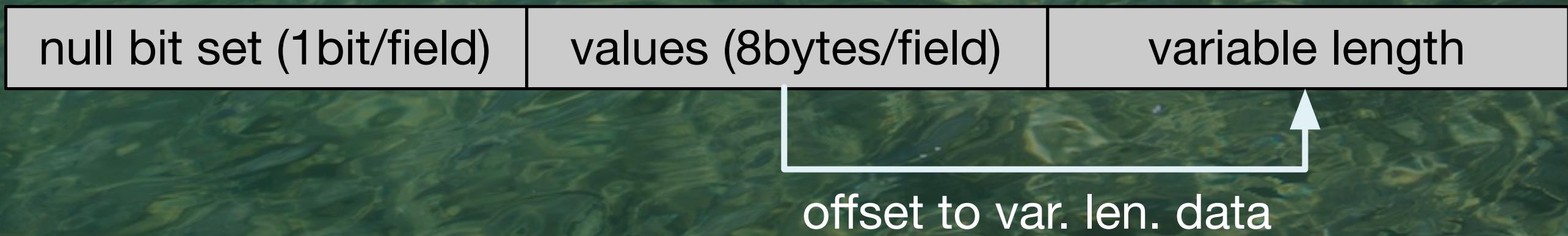
```
sql("SELECT a + b FROM table")
```

- Evaluating expressions billions of times:
 - Virtual function calls.
 - Boxing/unboxing.
 - Branching (if statements, etc.)

Implementation

Object Encoding

New CompactRow type:



- Compute hashCode and equals on raw bytes.

• Compare:

val person: Person

| | |
|---------------|----|
| name: String | |
| age: Int | 29 |
| addr: Address | |

“Buck Trends”



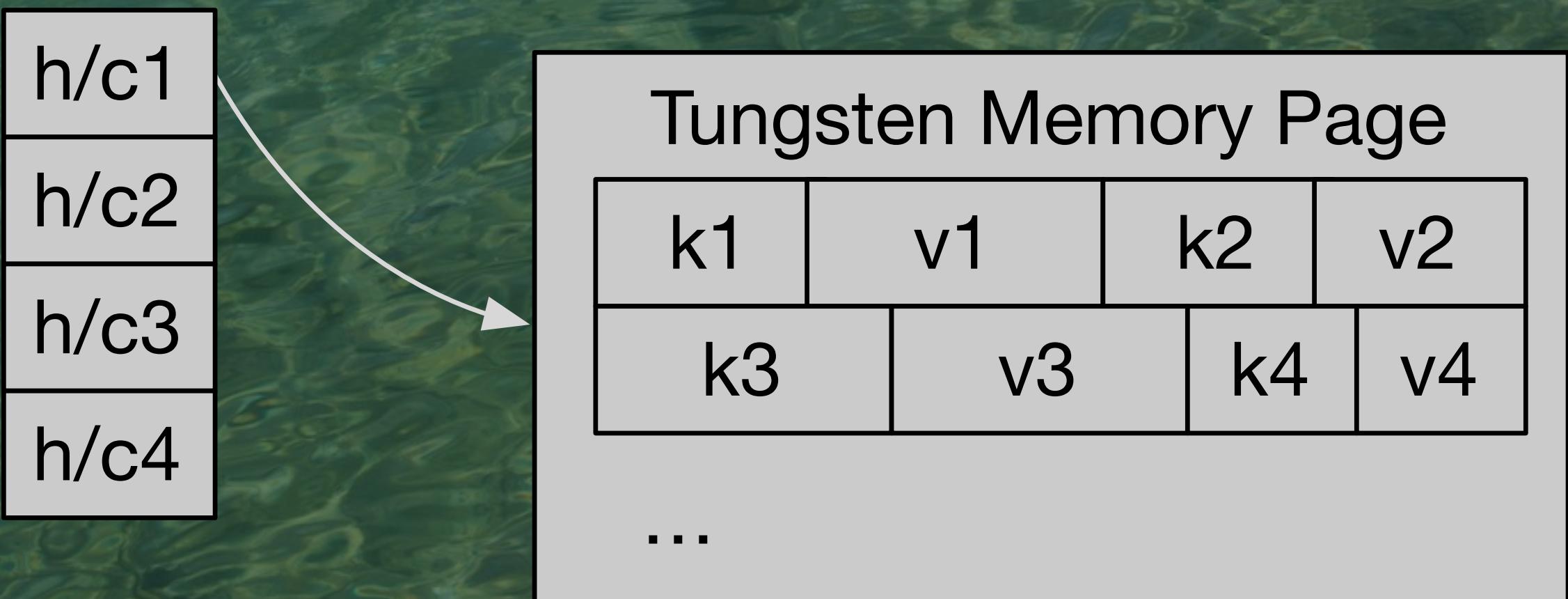
null bit set (1bit/field)

values (8bytes/field)

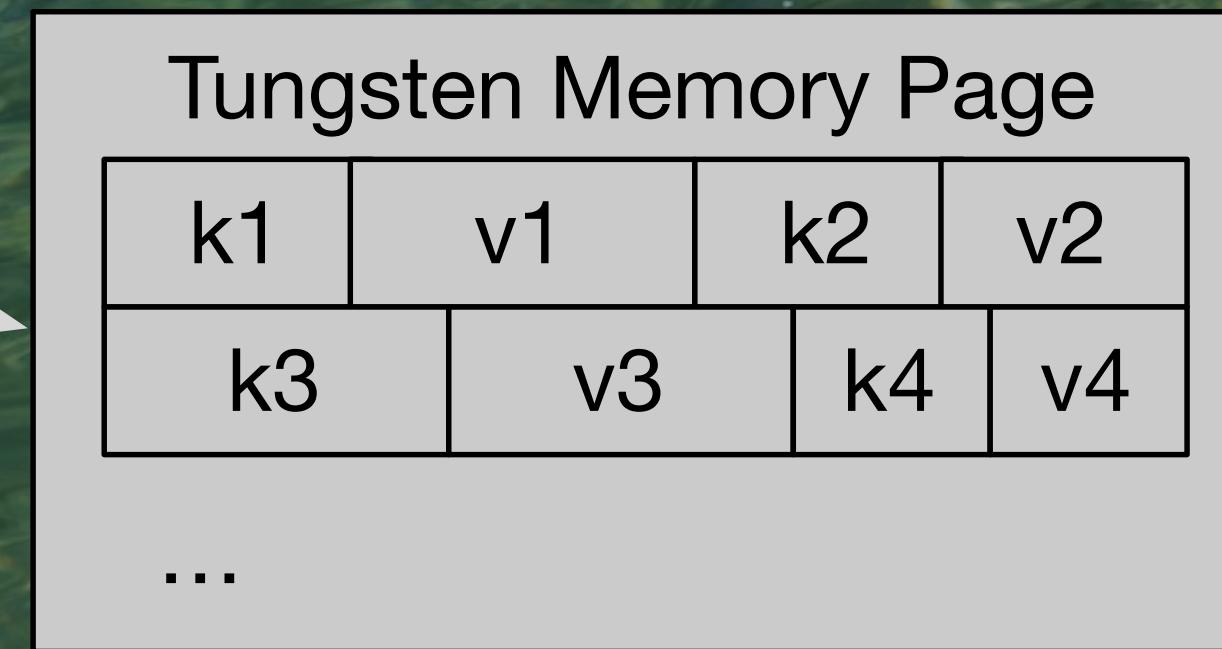
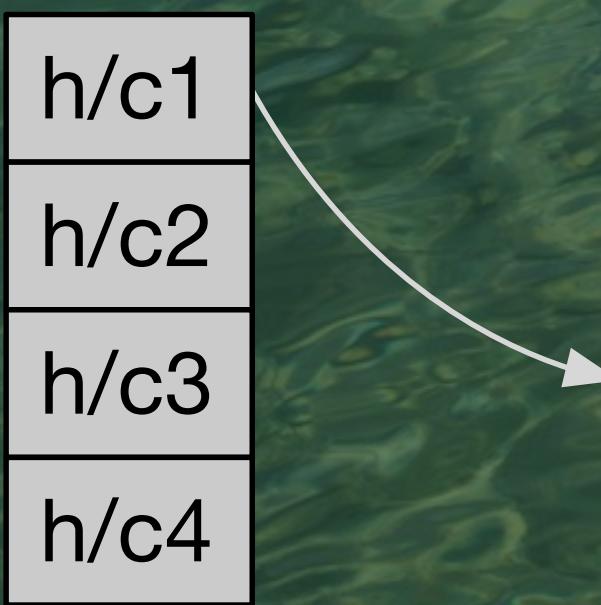
variable length

offset to var. len. data

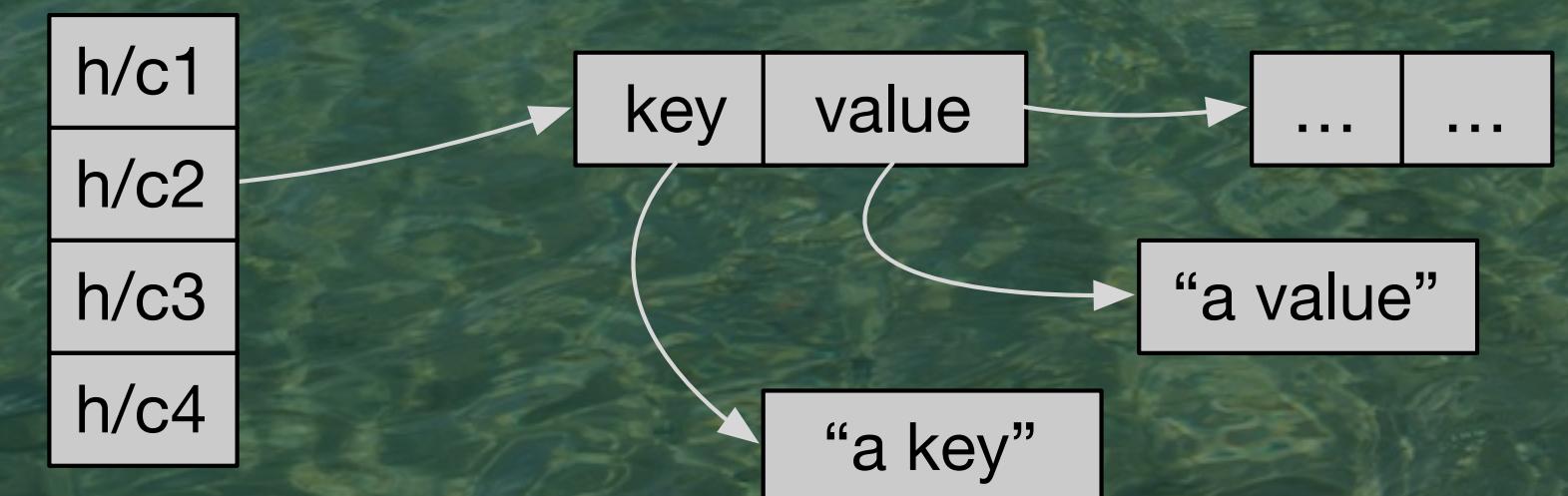
- BytesToBytesMap:



• Compare



Hash Map



Memory Management

- Some allocations off heap.
 - sun.misc.Unsafe.

Less Expression Overhead

```
sql("SELECT a + b FROM table")
```

- Solution:
 - Generate custom byte code.
 - Spark 1.X - for subexpressions.

Less Expression Overhead

```
sql("SELECT a + b FROM table")
```

- Solution:
 - Generate custom byte code.
 - Spark 1.X - for subexpressions.
 - Spark 2.0 - for whole queries.

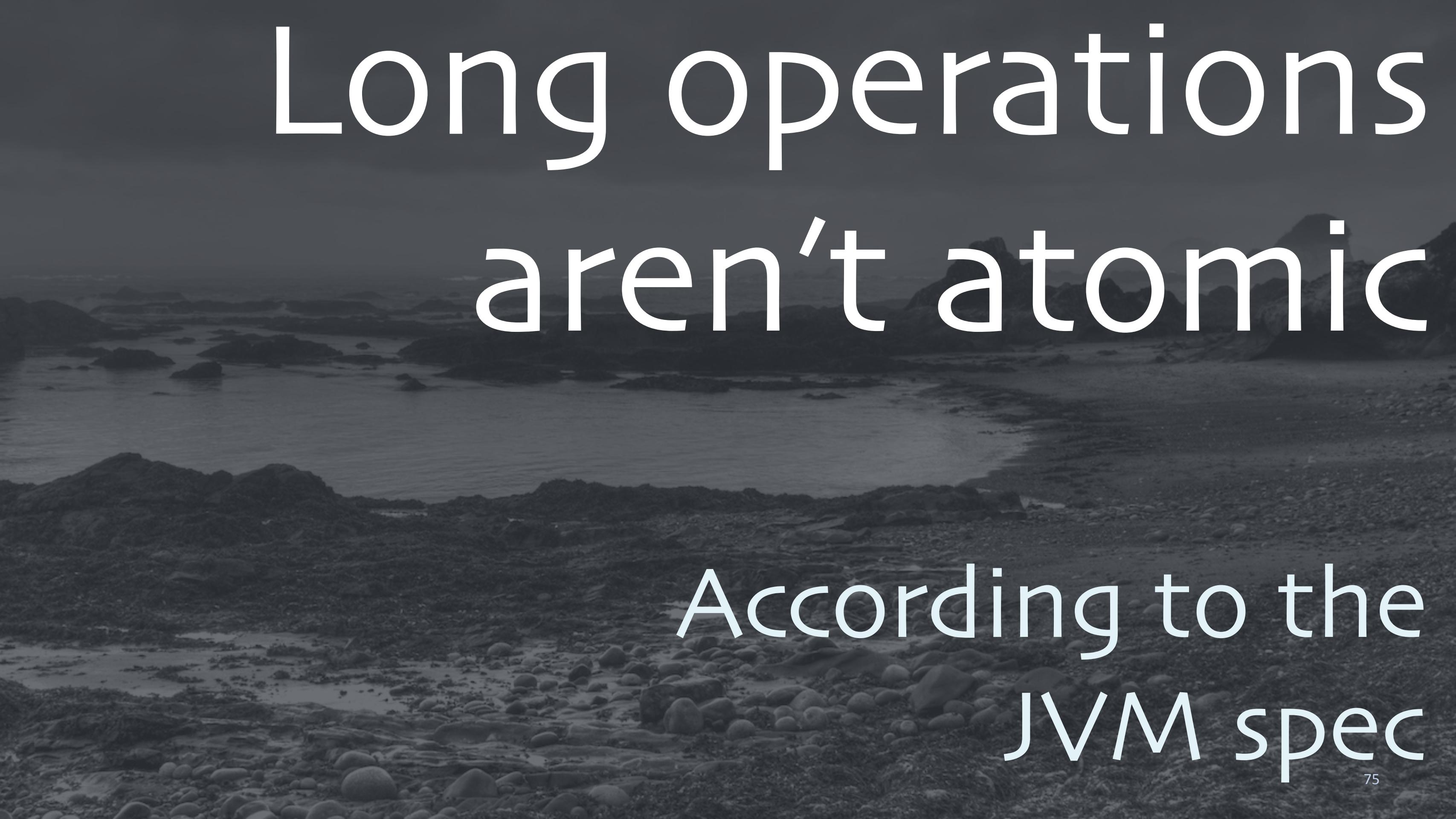


No Value Types

(Planned for Java 9 or 10)

```
case class Timestamp(epochMillis: Long) {  
  
  def toString: String = { ... }  
  
  def add(delta: TimeDelta): Timestamp = {  
    /* return new shifted time */  
  }  
  ...  
}
```

Don't allocate on the heap;
just push the primitive long
on the stack.
(scalac does this now.)

A black and white photograph of a coastal landscape. In the foreground, there's a rocky beach. Beyond it, a range of hills or mountains covered in low-lying vegetation. A large body of water, possibly a bay or a wide river, stretches across the middle ground, with more hills visible on the far shore under a hazy sky.

Long operations
aren't atomic

According to the
JVM spec

No Unsigned Types

What's
factorial(-1)?

Arrays Indexed with Ints

Byte Arrays
limited to 2GB!

```
scala> val N = 1100*1000*1000  
N2: Int = 1100000000 // 1.1 billion
```

```
scala> val array = Array.fill[Short](N)(0)  
array: Array[Short] = Array(0, 0, ...)
```

```
scala> import  
org.apache.spark.util.SizeEstimator
```

```
scala> SizeEstimator.estimate(array)  
res3: Long = 2200000016 // 2.2GB
```

```
scala> val b = sc.broadcast(array)
...broadcast.Broadcast[Array[Short]] = ...
```

```
scala> SizeEstimator.estimate(b)
res0: Long = 2368
```

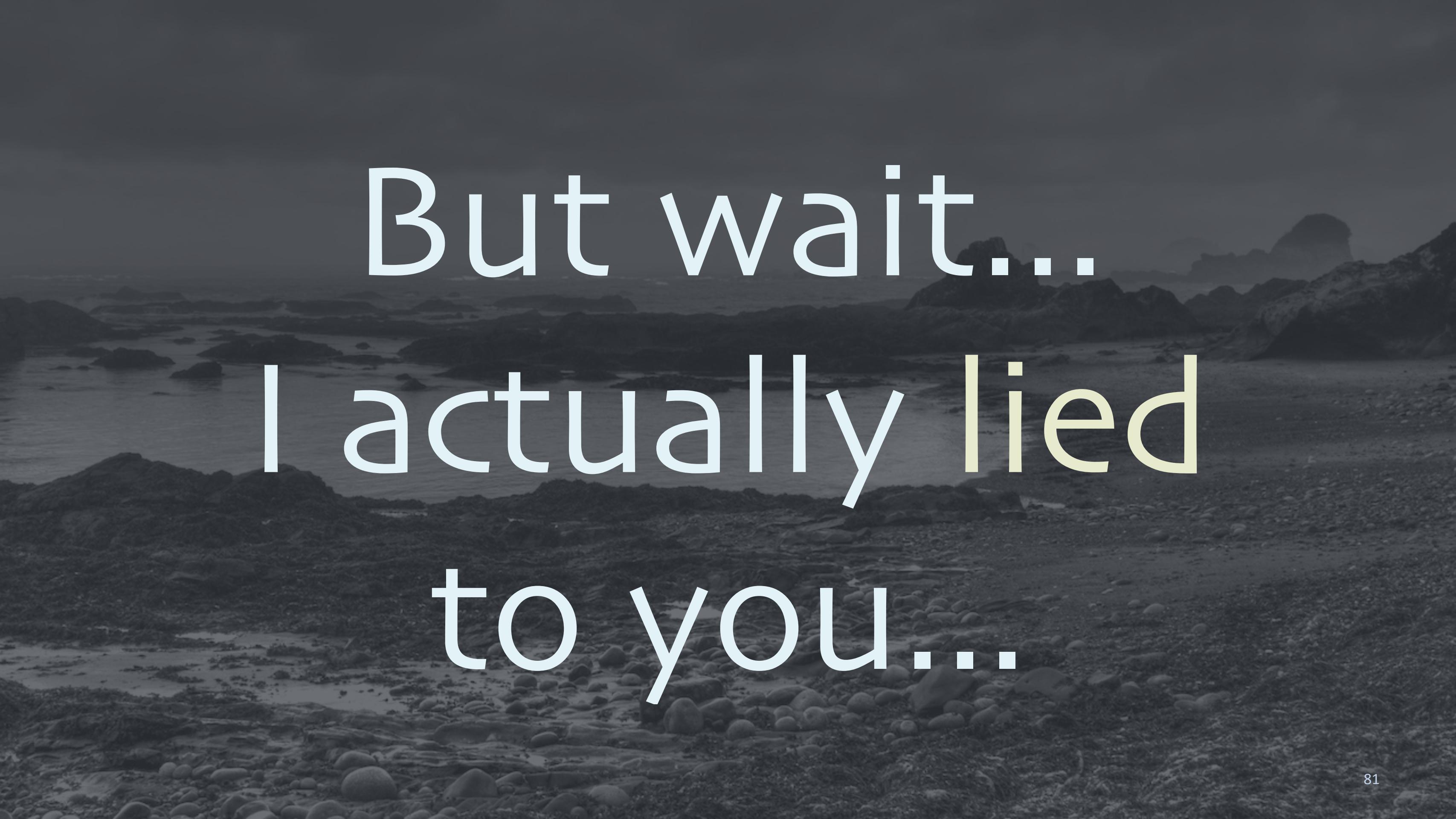
```
scala> sc.parallelize(0 until 100000).
| map(i => b.value(i))
```

```
scala> SizeEstimator.estimate(b)  
res0: Long = 2368
```

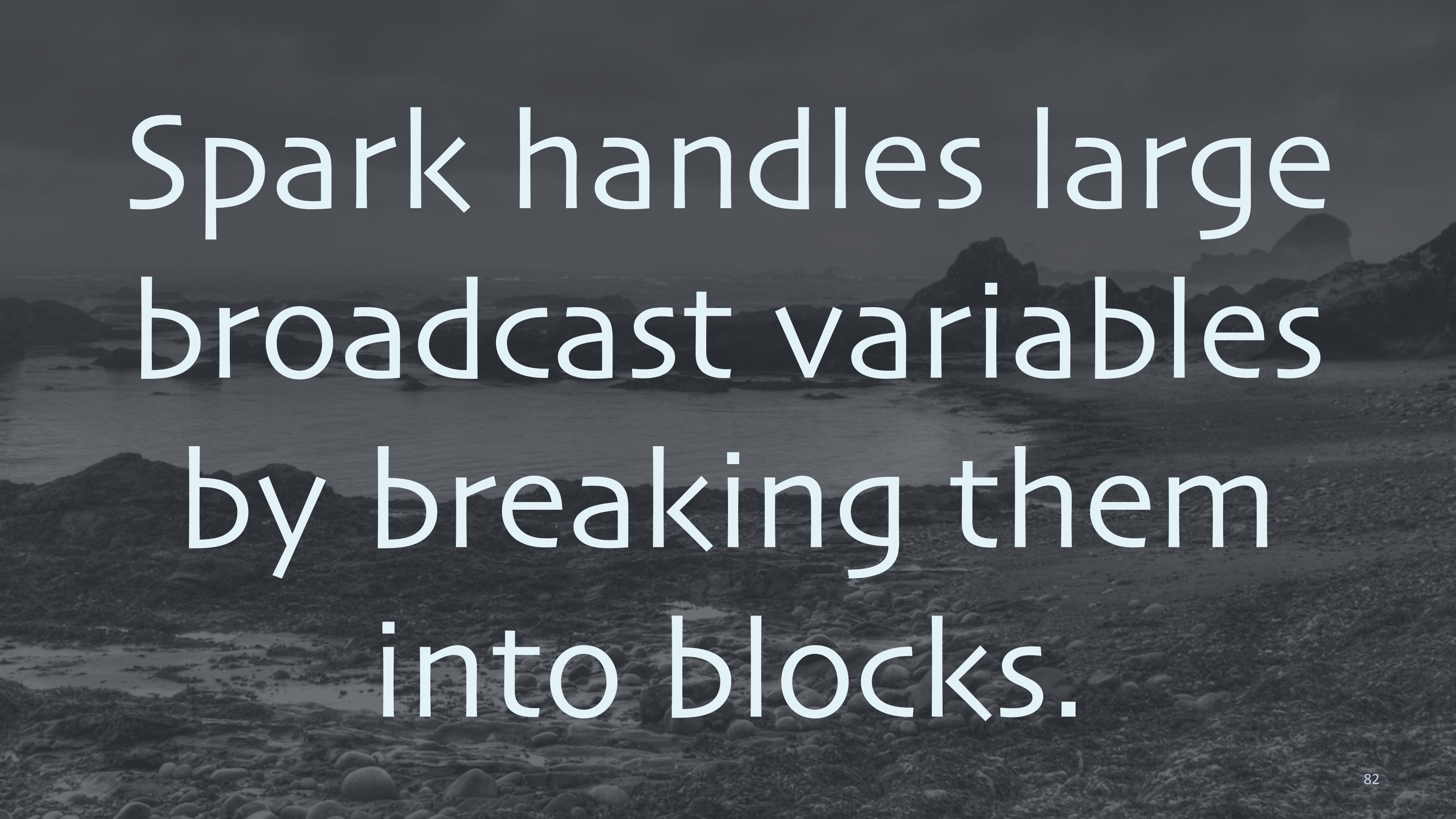
```
scala> sc.parallelize(0 until 100000).  
| map(i => b.value(i))
```

Boom!

```
java.lang.OutOfMemoryError:  
  Requested array size exceeds VM limit  
  
at java.util.Arrays.copyOf(...)  
...
```

A black and white photograph of a rugged coastline. In the foreground, there's a rocky beach with many smooth, rounded stones. Beyond the beach, the ocean extends to a distant horizon where numerous small, low-lying islands or rock formations are scattered across the water. The sky above is overcast and hazy.

But wait...
I actually lied
to you...

A black and white photograph of a coastal scene. In the foreground, there's a rocky, pebbled beach. The ocean waves are crashing onto the shore. In the background, there are several hills or mountains, some of which are partially obscured by mist or low-hanging clouds.

Spark handles large
broadcast variables
by breaking them
into blocks.

A photograph of a person walking along a wide, wet beach. The water is shallow and reflects the sky. In the background, a dense forest lines the shore. The foreground is dominated by the wet sand and the edge of the ocean waves.

Scala
REPL

java.lang.OutOfMemoryError:

 Requested array size exceeds VM limit

at java.util.Arrays.copyOf(...)

...

at java.io.ByteArrayOutputStream.write(...)

...

at java.io.ObjectOutputStream.writeObject(...)

at ...spark.serializer.JavaSerializationStream
 .writeObject(...)

...

at ...spark.util.ClosureCleaner\$.ensureSerializable(...)

...

at org.apache.spark.rdd.RDD.map(...)

```
java.lang.OutOfMemoryError:  
  Requested array size exceeds VM limit
```

```
at java.util.Arrays.copyOf(...)  
...  
at java.io.ByteArrayOutputStream.write(...)  
...  
at java.io.ObjectOutputStream  
at ...spark.serializer.JavaSe  
  .writeObject(...)  
...  
at ...spark.util.ClosureClear  
...  
at org.apache.spark.rdd.RDD.map(...)
```

Pass this closure to

RDD.map:

i => b.value(i)

```
java.lang.OutOfMemoryError:  
  Requested array size exceeds VM limit
```

```
at java.util.Arrays.copyOf( ... )
```

```
...  
at java.io.ByteArrayOutputSt:
```

```
...  
at java.io.ObjectOutputStream:
```

```
at ...spark.serializer.JavaSe  
.writeObject( ... )
```

```
...  
at ...spark.util.ClosureCleaner$.ensureSerializable( ... )
```

```
...  
at org.apache.spark.rdd.RDD.map( ... )
```

Verify that it's

“clean” (serializable).
 $i \Rightarrow b.value(i)$

```
java.lang.OutOfMemoryError:  
  Requested array size exceeds VM limit
```

```
at java.util.Arrays.copyOf(...)
```

```
...
```

```
at java.io.ByteArrayOutputStream.write(...)
```

```
...
```

```
at java.io.ObjectOutputStream.writeObject(...)
```

```
at ...spark.serializer.JavaSerializationStream  
  .writeObject(...)
```

```
...
```

```
at ...spark.util.ClosureClea
```

...which it does by
serializing to a byte array...

```
...
```

```
at org.apache.spark.rdd.RDD.i
```

```
java.lang.OutOfMemoryError:  
  Requested array size exceeds VM limit
```

```
at java.util.Arrays.copyOf( ... )
```

```
...  
at java.io.ByteArrayOutputStream:
```

```
...  
at java.io.ObjectOutputStream:
```

```
at ...spark.serializer.JavaSeri  
  .writeObject( ... )
```

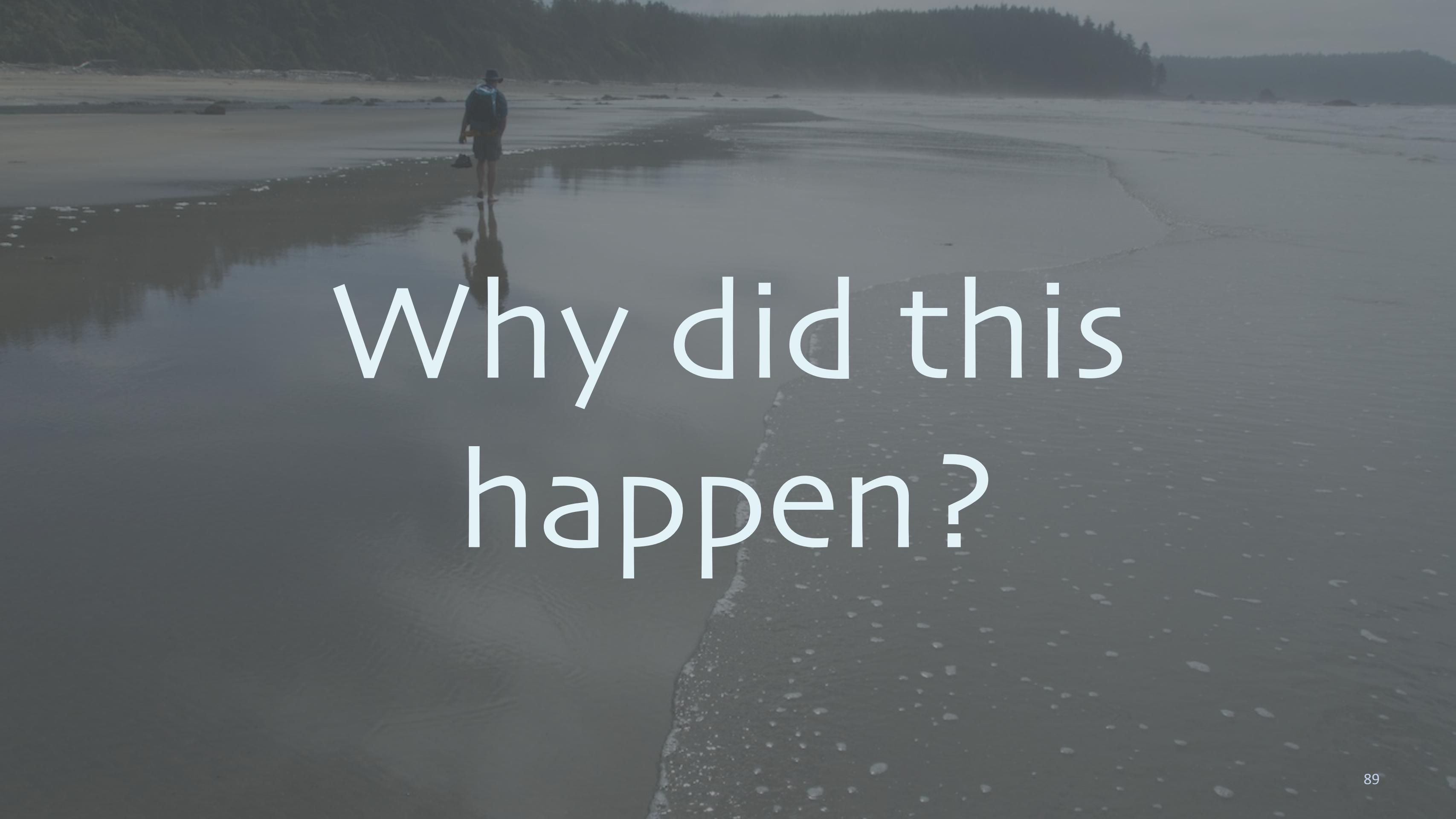
```
...  
at ...spark.util.ClosureCleaner.  
  i => b.value(i)
```

...which requires copying
an array...

What array???

```
scala> val array = Array.fill[Short](N)(0)
```

```
...
```

A photograph of a person walking along a wide, wet beach. The person is wearing a dark jacket, shorts, and a hat, and is carrying a backpack. They are walking away from the camera towards a dense forest. The beach is wet and reflective, with many small pools of water scattered across the sand. The ocean waves are visible in the background, crashing onto the shore. The overall atmosphere is peaceful and contemplative.

why did this
happen?

- You write:

```
scala> val array = Array.fill[Short](N)(0)
scala> val b = sc.broadcast(array)
scala> sc.parallelize(0 until 100000).
| map(i => b.value(i))
```



```
scala> val array = Array.fill[Short](N)(0)
scala> val b = sc.broadcast(array)
scala> sc.parallelize(0 until 100000).
           | map(i => b.value(i))
```

- Scala compiles:

```
class $iwC extends Serializable {
  val array = Array.fill[Short](N)(0)
  val b = sc.broadcast(array)
```

```
class $iwC extends Serializable {
  sc.parallelize(...).map(i => b.value(i))
}
```

```
scala> val array = Array.fill[Short](N)(0)
scala> val b = sc.broadcast(array)
scala> sc.parallelize(0 until 100000).
```

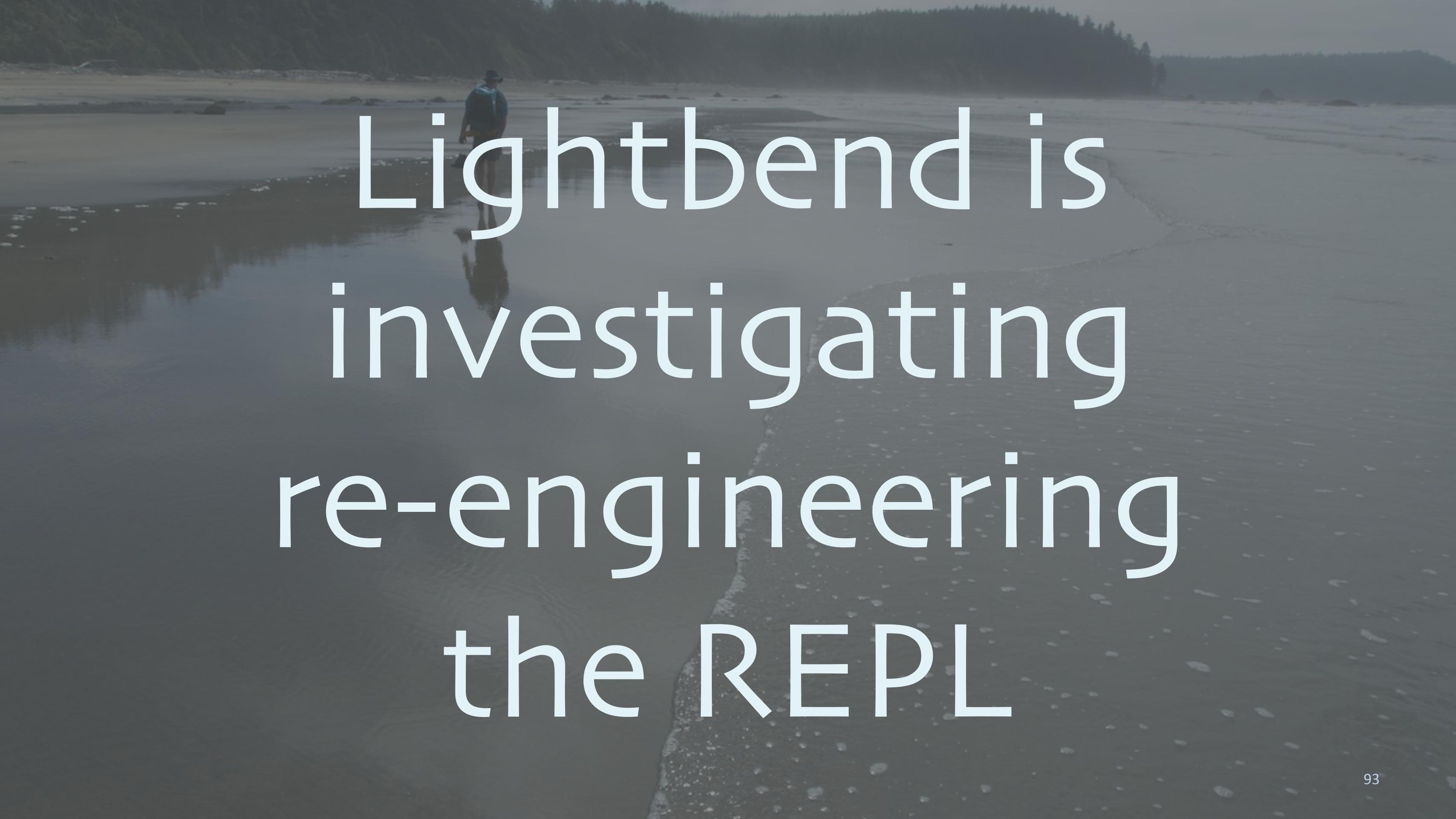
- Scala compiles:

... sucks in the whole object!

```
class $iwC extends Serializable {
  val array = Array.fill[Short](N)(0)
  val b = sc.broadcast(array)
```

```
class $iwC extends Serializable {
  val array = Array.fill[Short](N)(0)
  val b = sc.broadcast(array)
  sc.parallelize(...).map(i => b.value(i))}
```

So, this closure over “b”...

A black and white photograph of a person walking along a wide, shallow beach. The beach is covered in wet sand and small pools of water. In the background, a dense forest of tall evergreen trees lines the horizon under a clear sky.

Lightbend is
investigating
re-engineering
the REPL

A wide-angle photograph of a beach at low tide. A person wearing a hat and dark clothing walks away from the camera towards the horizon. The sand is wet and reflects the light. In the background, a dense forest of evergreen trees lines the shore. The sky is overcast.

Workarounds...

- Transient is often all you need:

```
scala> @transient val array =  
|   Array.fill[Short](N)(0)  
scala> ...
```

```
object Data { // Encapsulate in objects!
    val N = 1100*1000*1000
    val array = Array.fill[Short](N)(0)
    val getB = sc.broadcast(array)
}

object Work {
    def run(): Unit = {
        val b = Data.getB // local ref!
        val rdd = sc.parallelize(...).
            map(i => b.value(i)) // only needs b
        rdd.take(10).foreach(println)
    }
}
```

Why Scala?



See the longer version
of this talk at
polyglotprogramming.com/talks

polyglotprogramming.com/talks

lightbend.com/fast-data-platform

dean.wampler@lightbend.com

@deanwampler

Questions?

Bonus Material

You can find an extended version of this talk with more details at
polyglotprogramming.com/talks