

Scala and the JVM for Big Data: Lessons from Spark

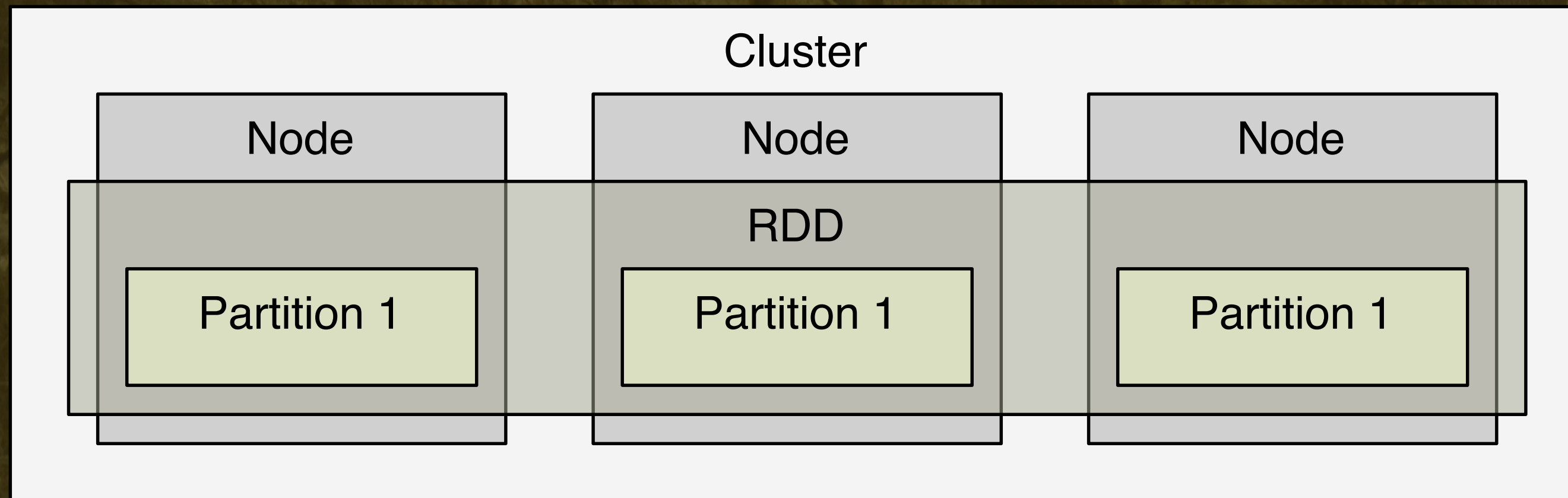
polyglotprogramming.com/talks
dean.wampler@lightbend.com
[@deanwampler](https://twitter.com/deanwampler)

Spark



A photograph of a forest with tall, thin trees and a path. In the foreground, two people are walking away from the camera on the path. The image is dimmed and serves as a background for the text.

A Distributed Computing Engine on the JVM



Resilient Distributed Datasets

Productivity?

Very concise, elegant, functional APIs.

- Scala, Java
- Python, R

- ... and SQL!

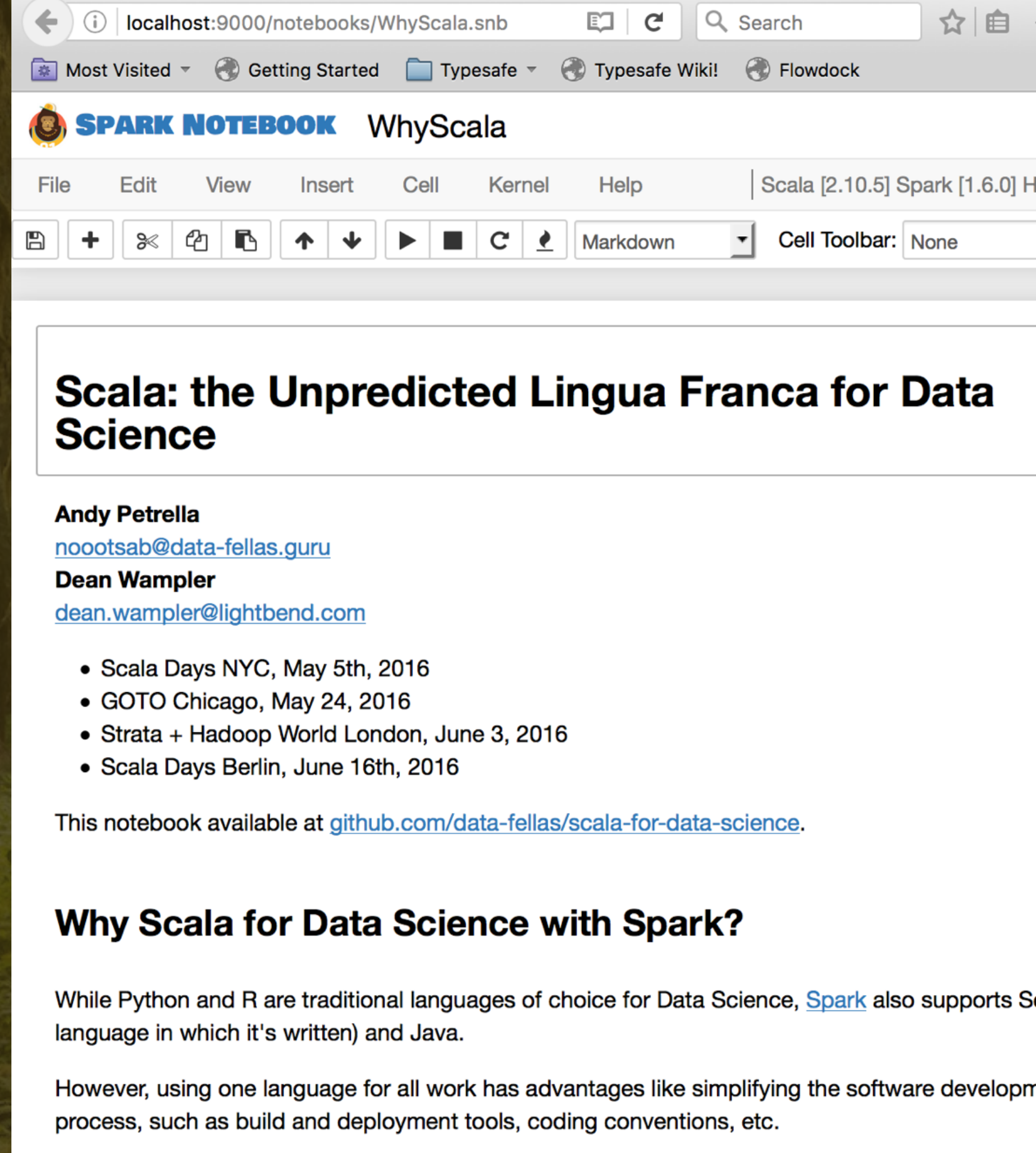
Productivity?

Interactive shell (REPL)

- Scala, Python, R, and SQL

Notebooks

- Jupyter
- Spark Notebook
- Zeppelin
- Beaker
- Databricks



The screenshot shows a web browser window displaying a Spark Notebook. The browser's address bar shows the URL `localhost:9000/notebooks/WhyScala.snb`. The notebook interface includes a menu bar with options like File, Edit, View, Insert, Cell, Kernel, and Help. Below the menu is a toolbar with icons for saving, adding, deleting, and running cells. The main content area displays the title "Scala: the Unpredicted Lingua Franca for Data Science" by Andy Petrella and Dean Wampler. A list of events where the notebook was presented is shown, including Scala Days NYC, GOTO Chicago, Strata + Hadoop World London, and Scala Days Berlin. The text also mentions that the notebook is available on GitHub and discusses why Scala is chosen for data science with Spark.

localhost:9000/notebooks/WhyScala.snb

Most Visited Getting Started Typesafe Typesafe Wiki! Flowdock

SPARK NOTEBOOK WhyScala

File Edit View Insert Cell Kernel Help | Scala [2.10.5] Spark [1.6.0] H

Markdown Cell Toolbar: None

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.





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



Inverted Index

inverse index

block

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

tracle!!

```
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))14
```

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

```
val sparkContext = new
SparkContext(master, "Hadoop provides...")
```

```
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))
}
```

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

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

RDD[(String,String),Int]: ((Hadoop,.../hadoop),20)


```
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 => seq.groupByKey().mapValues(_.sum)
}.saveAsTextFile("/path/to/output")
```

RDD[(String,Iterable((String,Int))]: (Hadoop,seq(.../hadoop,20),...))

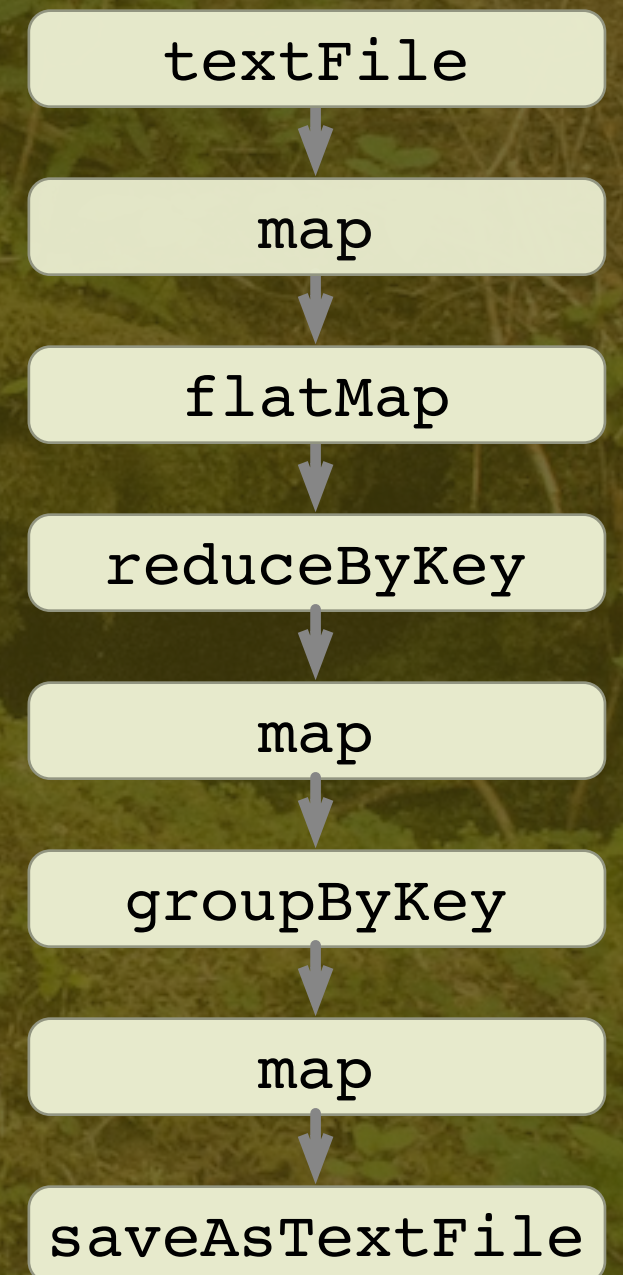
```
val array = line.split( , , 2)
(array(0), array(1))
}.flatMap {
  case (id, contents) =>
    toWords(contents).map(w => ((w, id), 1))
}.reduceByKey(_ + _)
map {
  case ((word, id), n) => (word, (id, n))
}.groupByKey
mapValues {
  seq => sortByCount(seq)
}.saveAsTextFile("/path/to/output")
```

RDD[(String,Iterable((String,Int))]: (Hadoop,seq(.../hadoop,20),...))

Productivity?

Intuitive API:

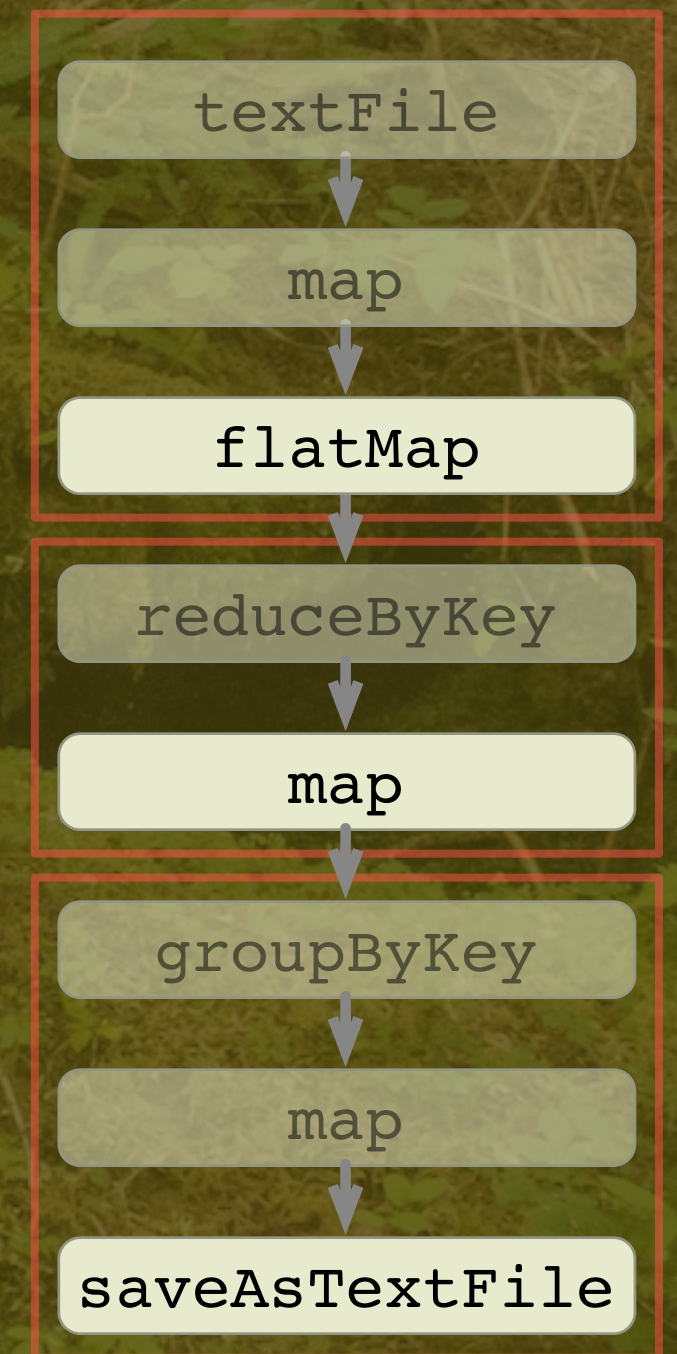
- Dataflow of steps.
- Inspired by Scala collections and functional programming.



Performance?

Lazy API:

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





Higher-Level APIs

A scenic mountain landscape featuring a calm lake in the middle ground, surrounded by dense evergreen forests and rocky terrain. In the foreground, a hiker with a blue backpack is walking on a rocky trail. The background shows majestic mountains under a blue sky with scattered clouds. Large white text is overlaid on the left side of the image.

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.

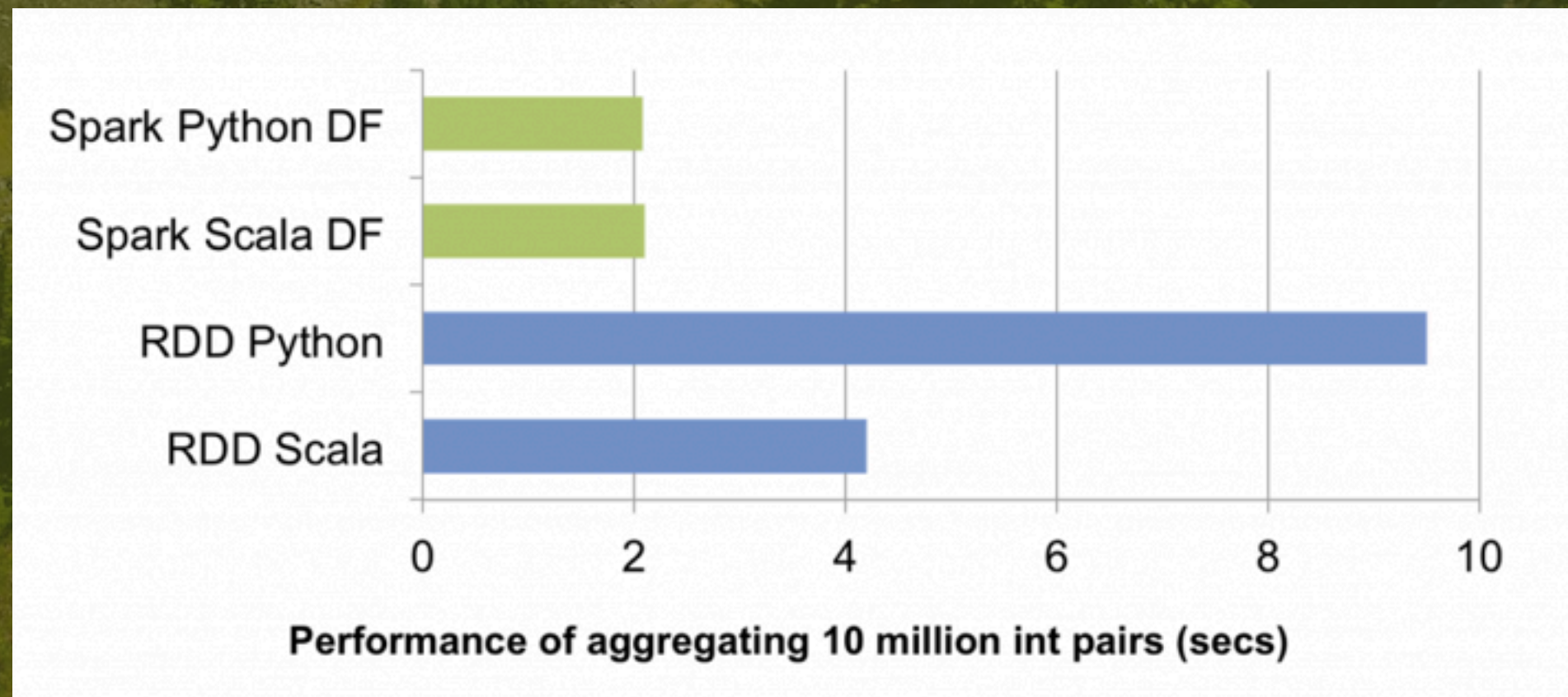
```
val planes_for_flights2 =
  flights.join(planes,
    flights("tailNum") ===
    planes("tailNum")).limit(100)
```

```
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!



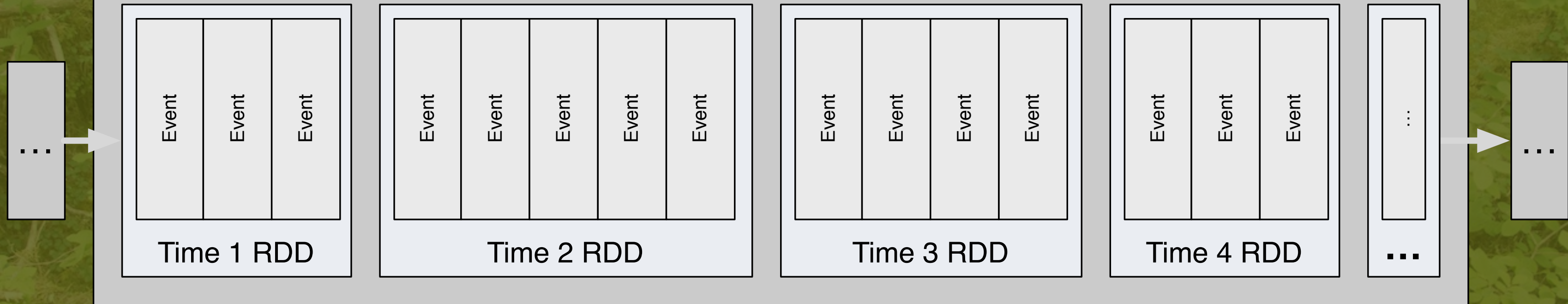
```
def join(right: Dataset[_], joinExprs: Column): DataFrame = {
def groupBy(cols: Column*): RelationalGroupedDataset = {
def orderBy(sortExprs: Column*): Dataset[T] = {
def select(cols: Column*): Dataset[...] = {
def where(condition: Column): Dataset[T] = {
def limit(n: Int): Dataset[T] = {
def intersect(other: 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 = {
```





Structured Streaming

DStream (discretized stream)



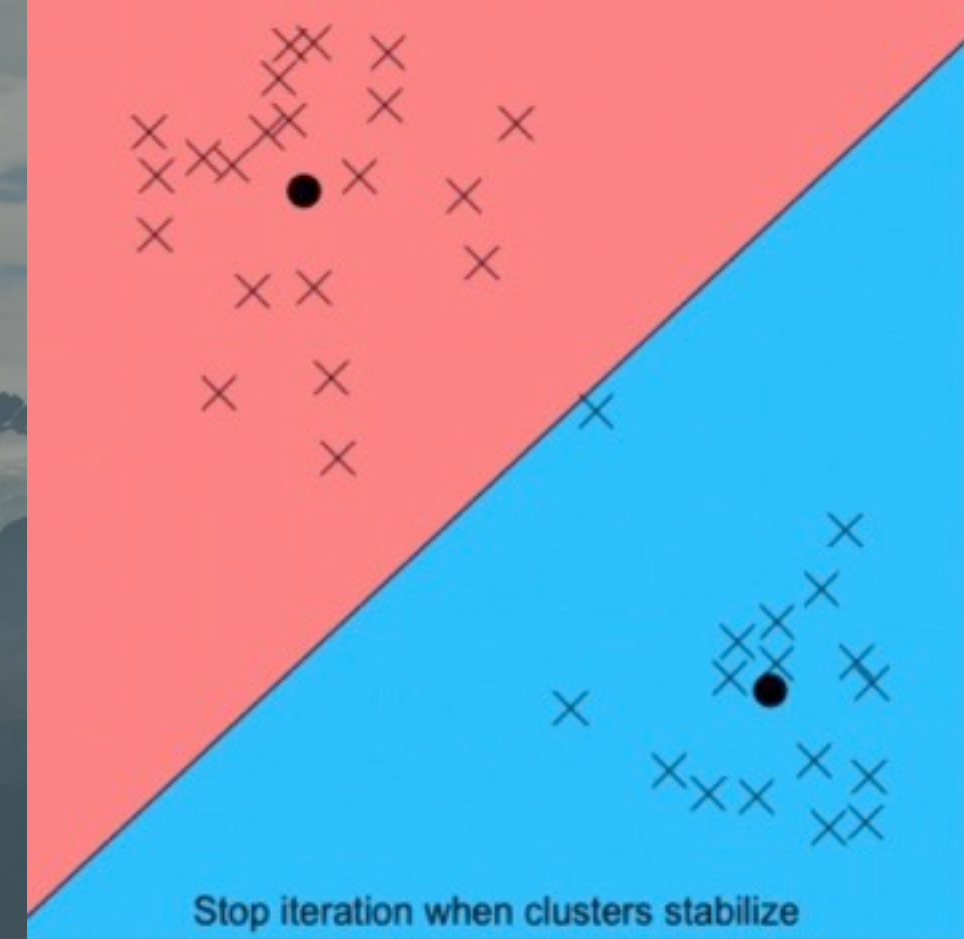
Window of 3 RDD Batches #1

Window of 3 RDD Batches #2



ML/Mlib

K-Means

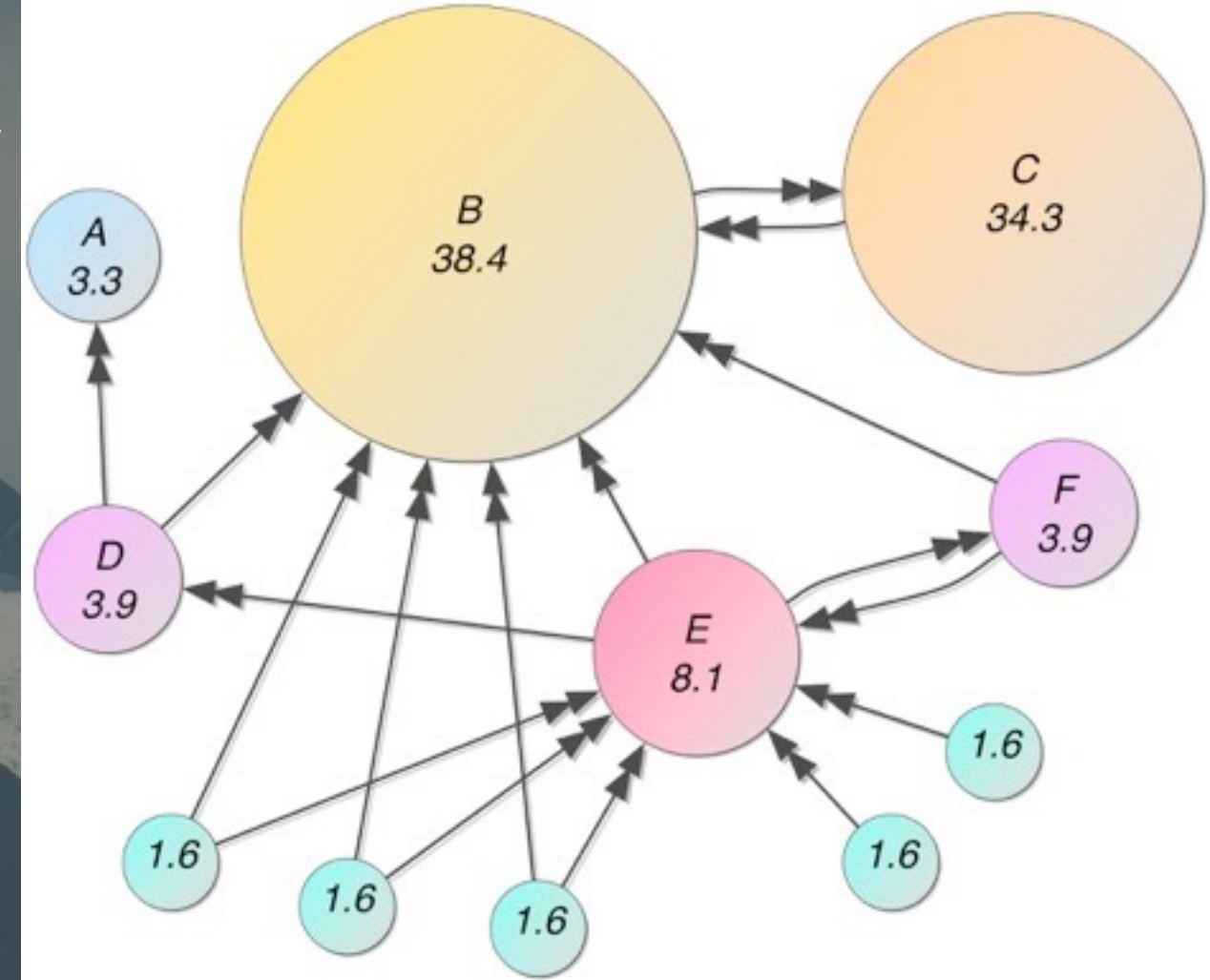


- Machine Learning requires:
 - Iterative training of models.
 - Good linear algebra perf.



GraphX

PageRank



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

A wide-angle photograph of a beach at low tide. The sky is filled with heavy, grey clouds. The ocean is visible in the distance with white-capped waves. A lone figure stands in the shallow water, looking out. The foreground shows a sandy beach with scattered seaweed and a long, thick green rope lying on the sand.

Foundation:

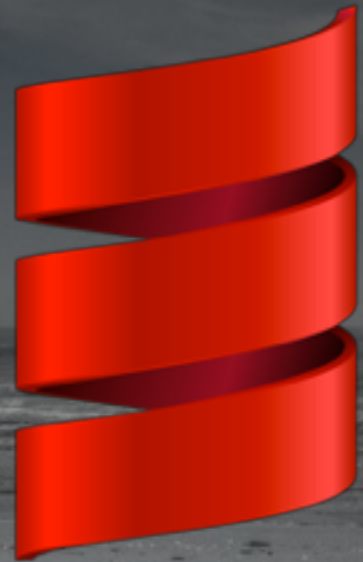
The JVM

A person stands on a wide, sandy beach, looking out at the ocean under a cloudy sky. The beach is littered with seaweed and a green rope lies on the sand in the foreground. The overall mood is contemplative and somewhat somber.

20 Years of DevOps

Lots of Java Devs

Tools and Libraries



Akka
Breeze
Algebird
Spire & Cats
Axle



...

Big Data Ecosystem



Spark



Scalding



Apache Kafka
A high-throughput distributed messaging system.



Hadoop



Cassandra



STORM



H₂O.ai



The Apache Software Foundation
<http://www.apache.org/>



Apache Solr



samza

A scenic landscape featuring a calm lake in the foreground, a dense forest of evergreen trees on the right side, and misty, forested mountains in the background under a cloudy sky. The overall tone is cool and atmospheric.

But it's

not perfect....



Richer data libs. in Python & R

Garbage Collection

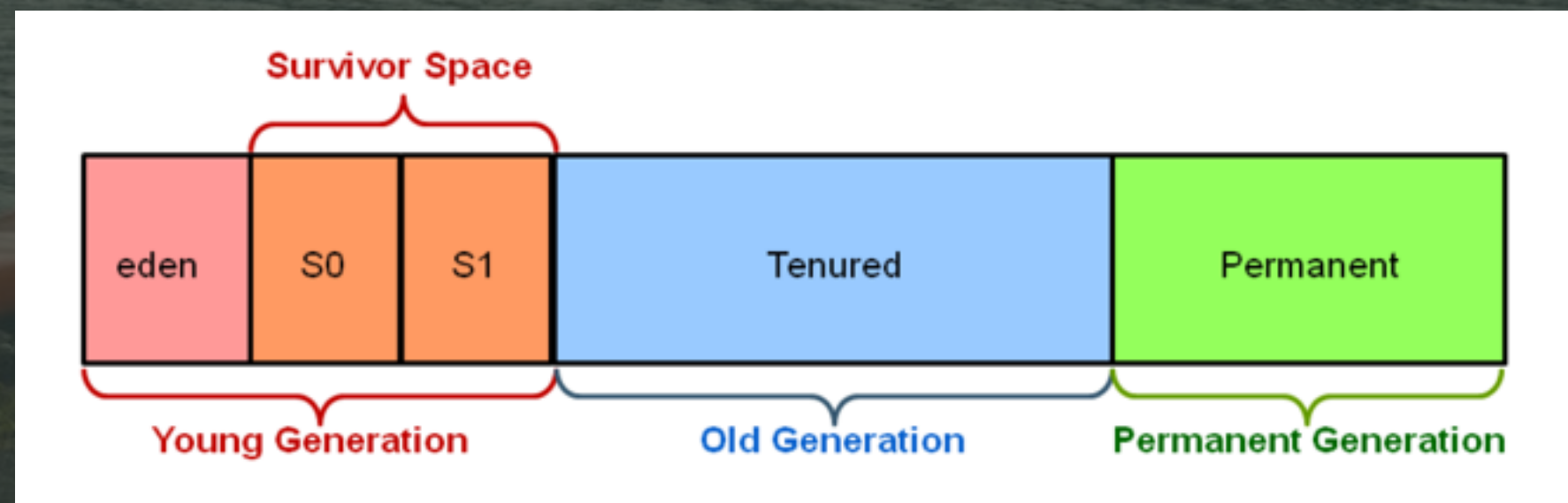


GC Challenges

- 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:InitiatingHeapOccupancyPercent=... -XX:ConcGCThread=...`

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]
```



“second”

“first”

“third”

“zeroth”

Arrays

val person: Person

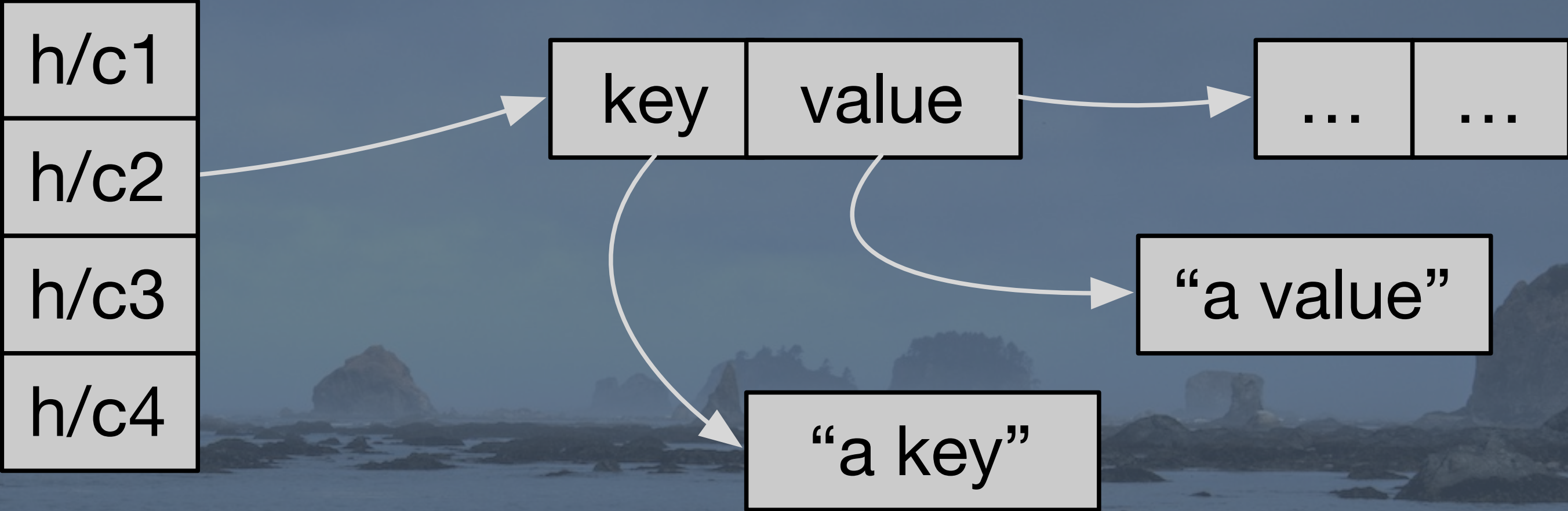
name: String	
age: Int	29
addr: Address	

“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?

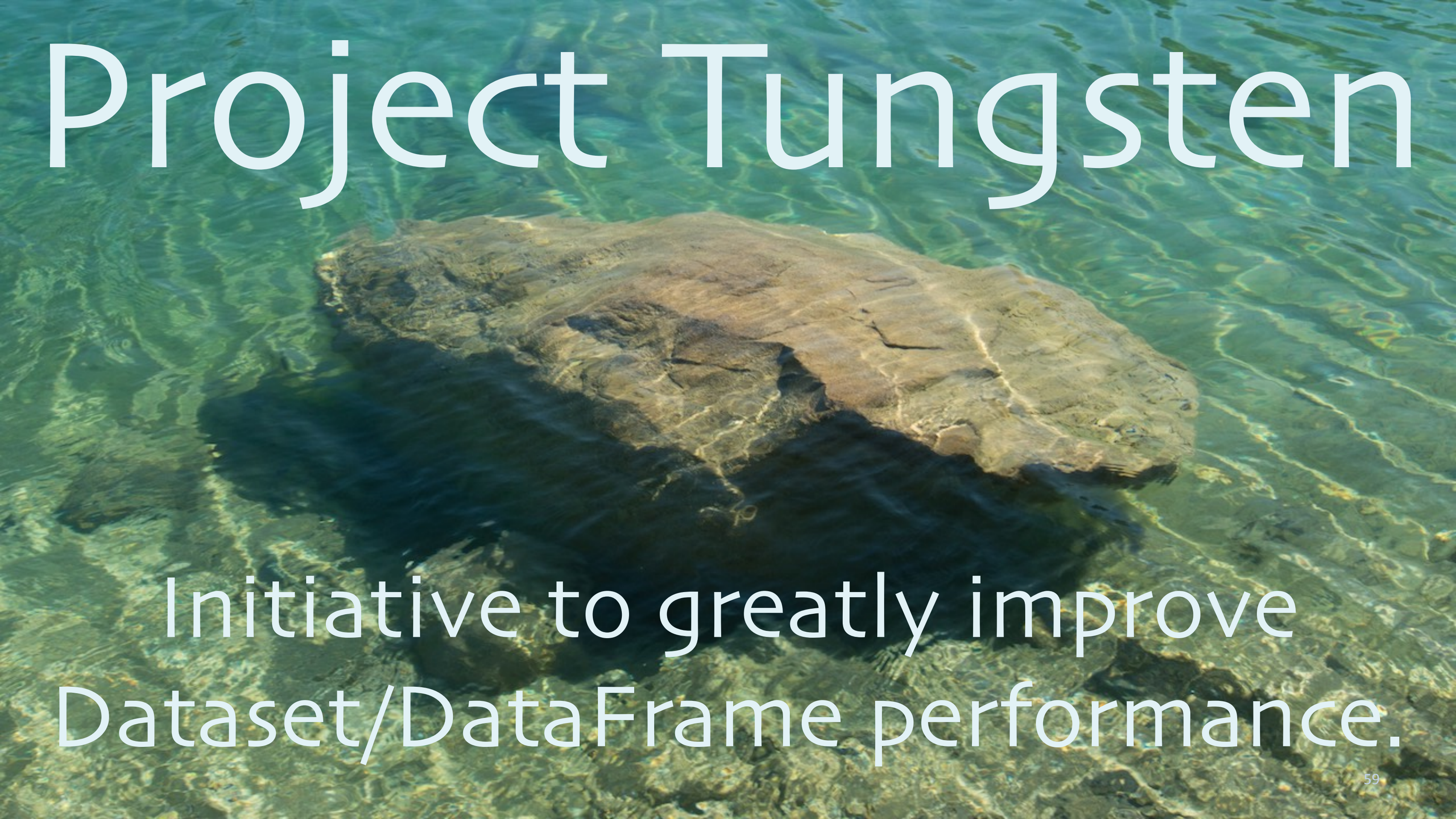
- But more CPU use today:
 - More Serialization.
 - More Compression.
 - More Hashing (joins, group-bys).

Improving Performance

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

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

Project Tungsten

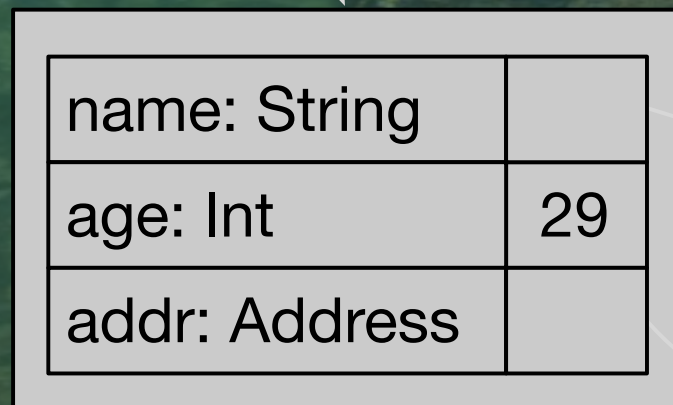
A large, flat, brownish rock is partially submerged in clear, shallow water. The water is a vibrant greenish-blue color, and the rock's shadow is cast onto the bottom of the water. The background shows more of the water and some rocks.

Initiative to greatly improve
Dataset/DataFrame performance.

Goals

Reduce References

val person: Person



val myArray: Array[String]



"Buck Trends"

"second"

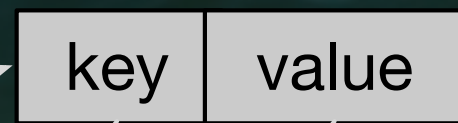
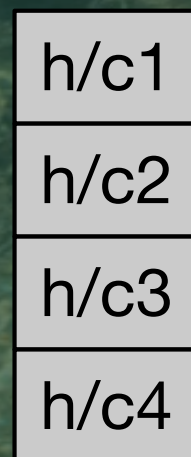
"first"

"third"

"zeroth"



Hash Map

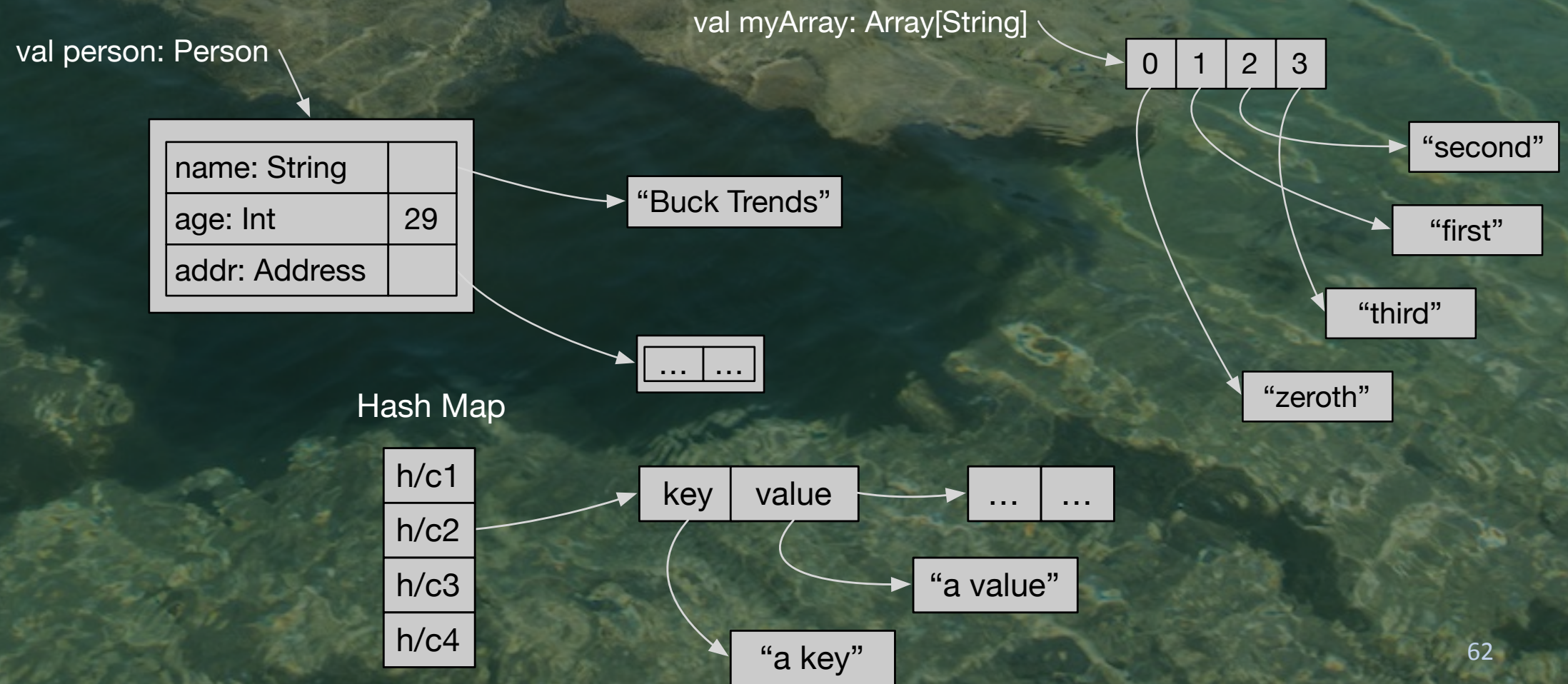


"a value"

"a key"

Reduce References

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



Less Expression Overhead

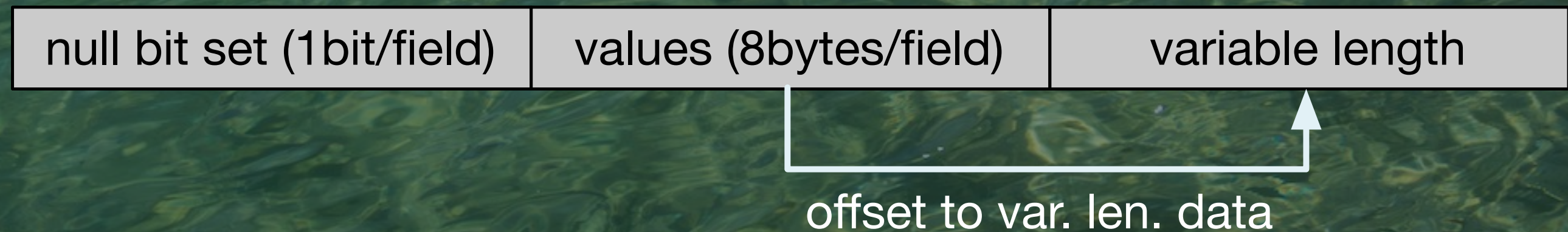
```
sql("SELECT a + b FROM table")
```

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

Implementation

Object Encoding

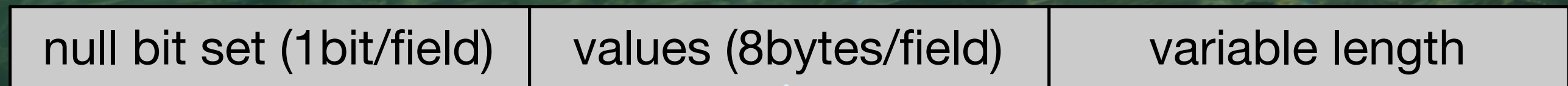
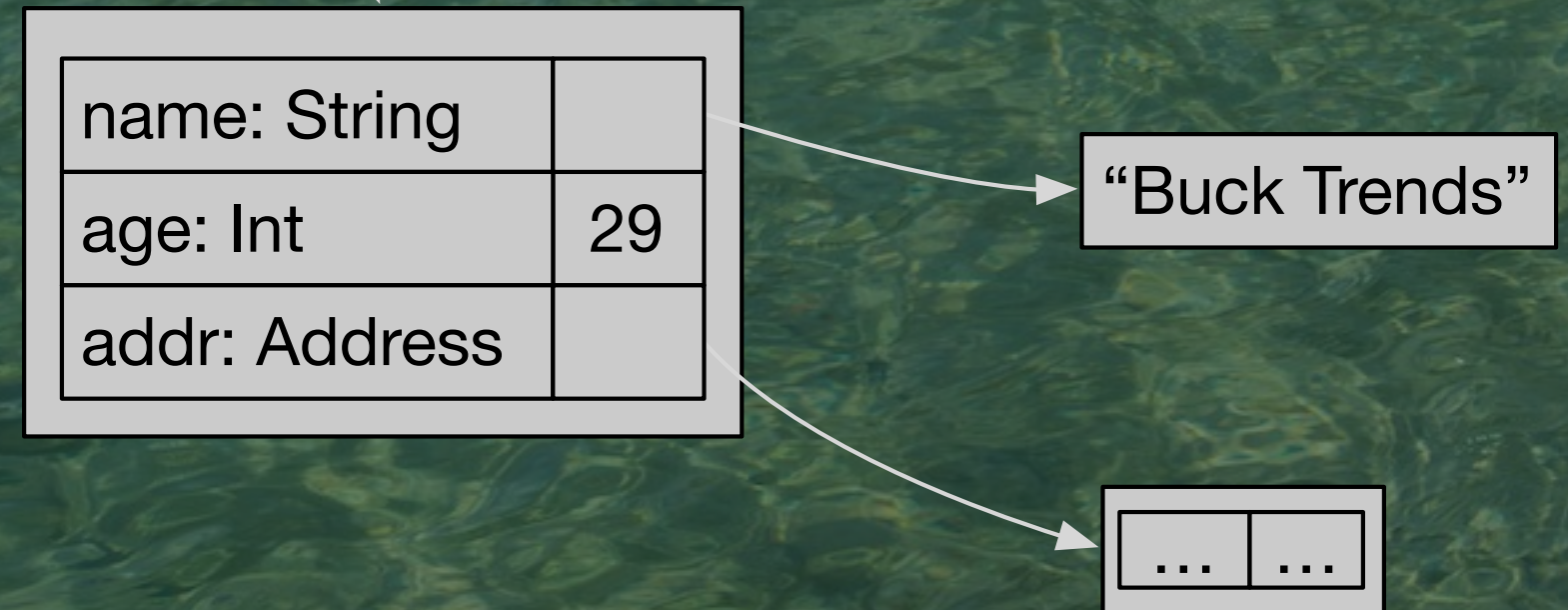
New CompactRow type:



- Compute hashCode and equals on raw bytes.

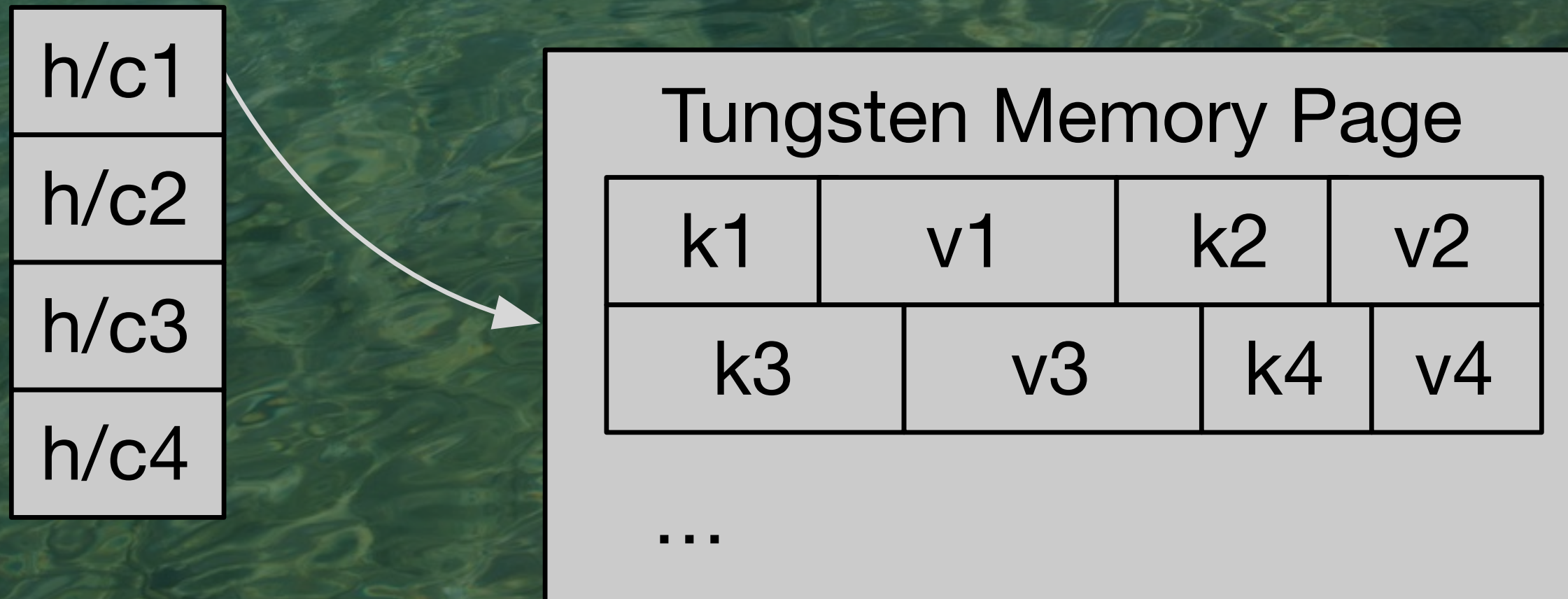
- Compare:

val person: Person



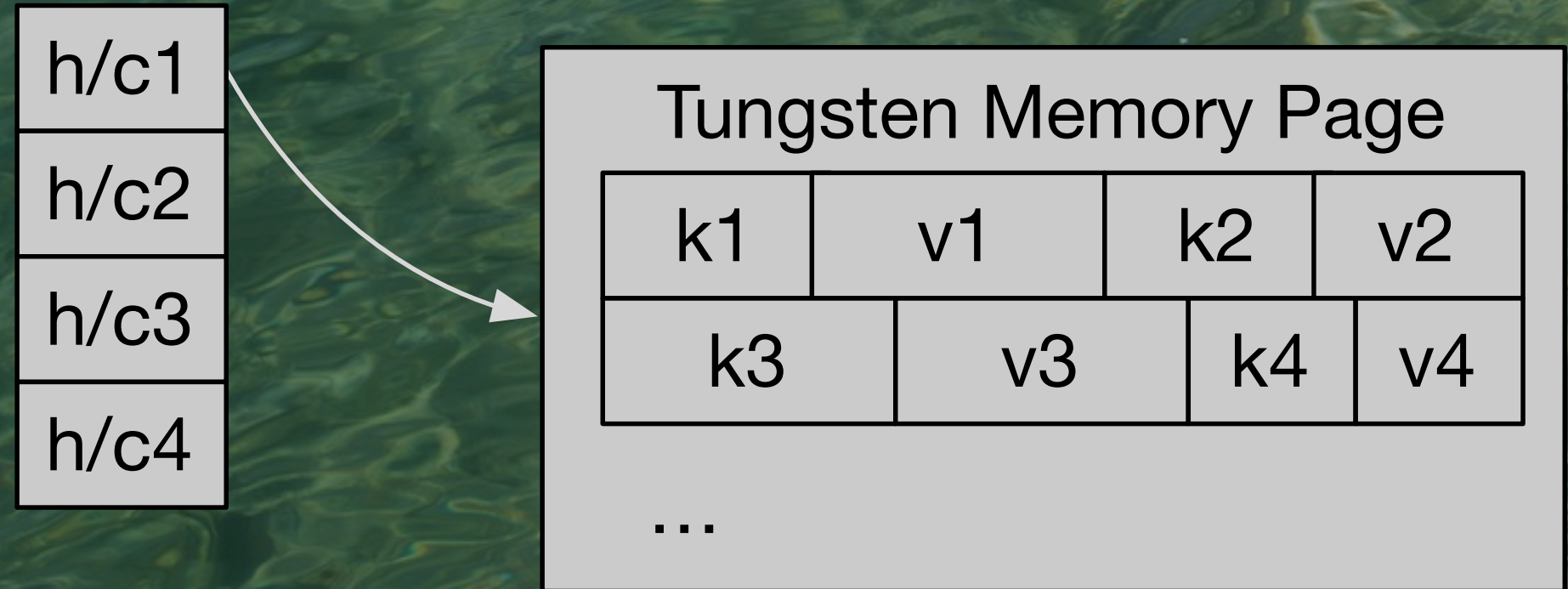
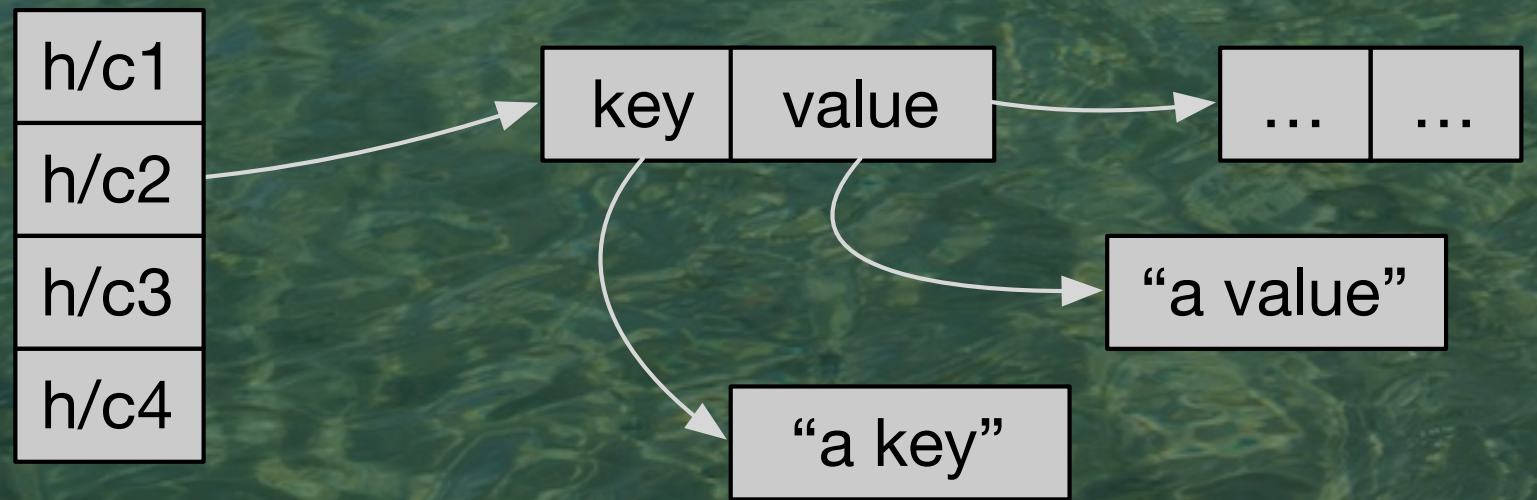
offset to var. len. data

- 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.)



Long operations
aren't atomic

According to the
JVM spec

No Unsigned Types

What's
 $\text{factorial}(-1)$?

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
```

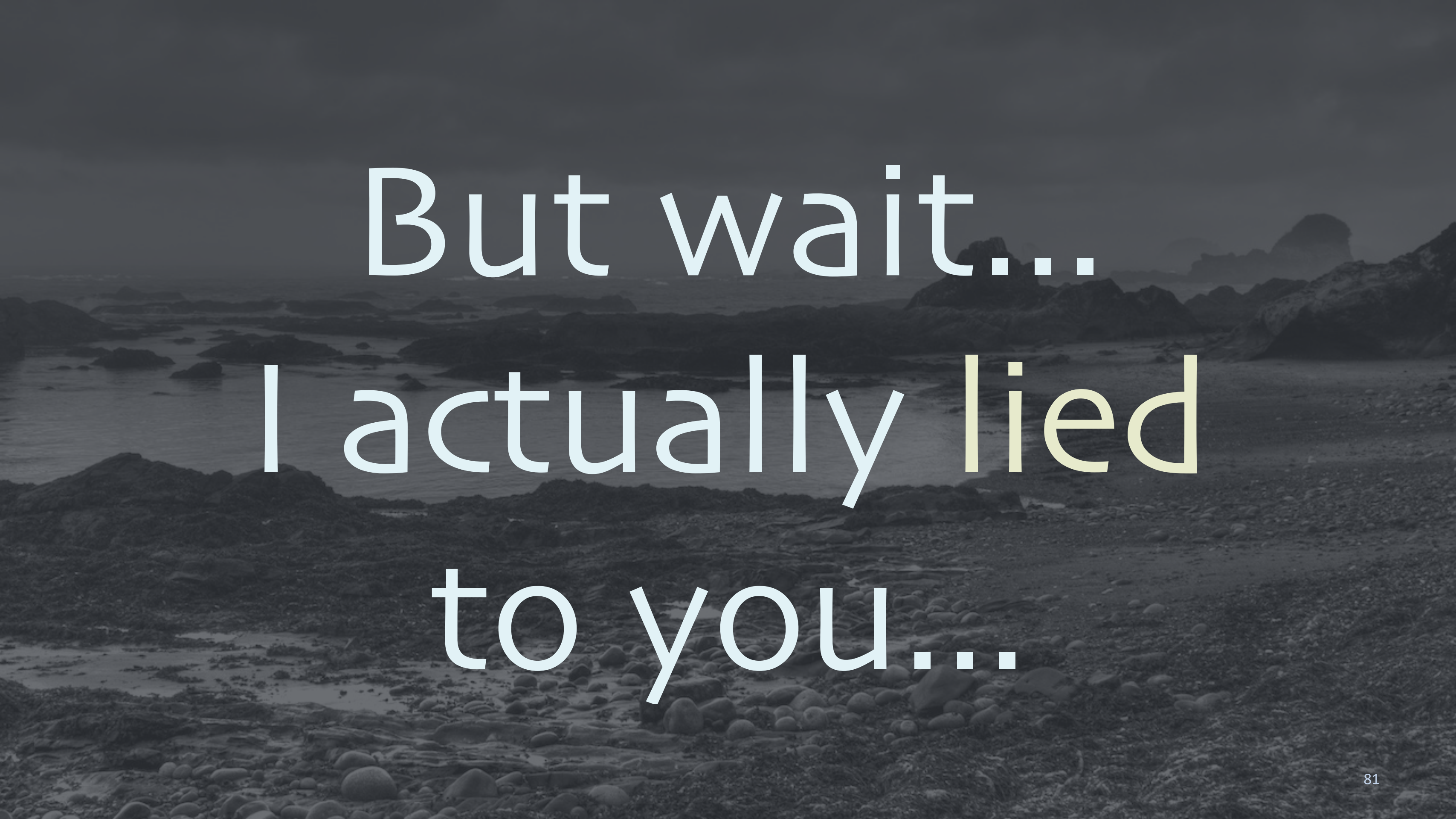
```
scala> sc.parallelize(0 until 100000).
  | map(i => b.value(i))
```

Boom!

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

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

```
...
```

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



Spark handles large
broadcast variables
by breaking them
into blocks.



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.JavaS
```

```
    .writeObject(...)
```

```
...
```

```
at ...spark.util.ClosureClean
```

```
...
```

```
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.ByteArrayOutputStream
```

```
...
```

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

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

```
...
```

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

```
...
```

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

Verify that it's
“clean” (serializable).
`i => 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.ClosureCleaner
```

```
...
```

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

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

```
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.JavaSerializer
```

```
    .writeObject(...)
```

```
...
```

```
at ...spark.util.ClosureCleaner
```

...which requires copying
an array...

What array???

$i \Rightarrow b.value(i)$

```
...
```

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

```
...
```


A person wearing a hat and a backpack is walking away from the camera on a wide, sandy beach. The beach is wet, and the person's reflection is clearly visible in the shallow water. The background shows a dense forest of trees under a hazy sky.

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)
```

```
  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).
```

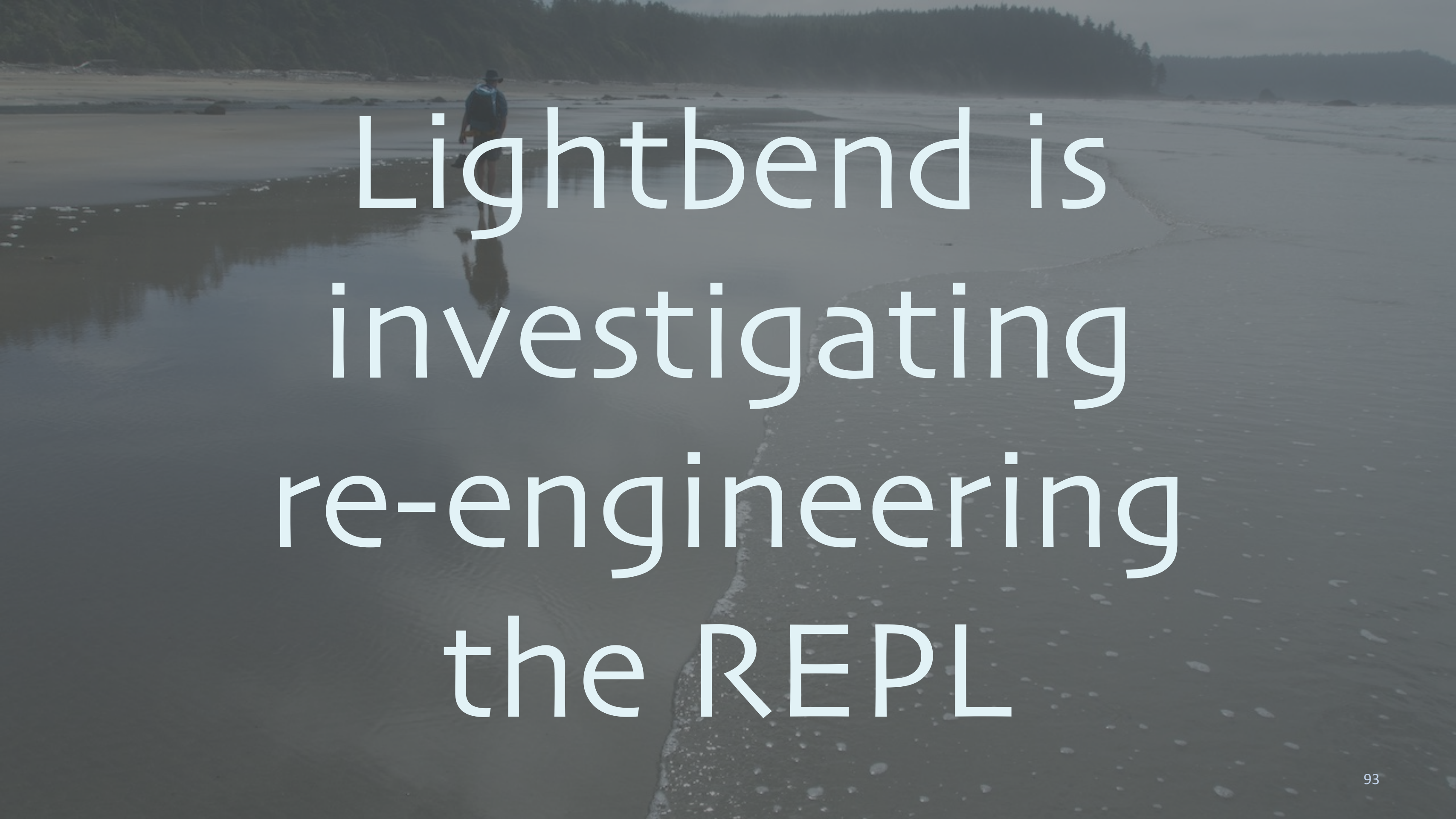
- Scala compiles:

... sucks in the whole object!

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

So, this closure over "b"...

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

A person wearing a hat and a backpack is walking on a wide, sandy beach at low tide. The water is shallow and reflects the person and the sky. The background shows a line of trees and a hazy sky. The text is overlaid on the image in a large, white, sans-serif font.

Lightbend is
investigating
re-engineering
the REPL

A person wearing a hat and a backpack is walking on a wide, sandy beach at low tide. The water is shallow and reflects the person and the surrounding landscape. The background shows a dense forest of trees on a hillside. The overall scene is serene and quiet.

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](https://twitter.com/deanwampler)

Questions?

Bonus Material

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