

# Why Did We Create Kotlin API for Apache Spark

Pasha Finkelshteyn, JetBrains

# Pasha Finkelshteyn

Developer  for Big Data @ JetBrains

@asm0di0

# How did I fall in love with Big Data?

- 7 years ago Hadoop looked like a magic to Java enterprise developer
- Started to look for data-related projects
- Started to read about DBs
- Moved from Team Lead to Data Engineer



# First thing Data Engineer learns?



# Example

```
1 spark
2   .read
3   .csv(path)
4   .as[MyDTO]
5   .map(dto => AnotherDTO(dto.x, dto.y))
6   .filter(_.n != null)
7   .write
8   .json(path)
```

# Example

```
1 spark
2   .read
3   .csv(path)
4   .as[MyDTO]
5   .map(dto => AnotherDTO(dto.x, dto.y))
6   .filter(_.n != null)
7   .write
8   .json(path)
```

# Example

```
1 spark
2   .read
3   .csv(path)
4   .as[MyDTO]
5   .map(dto => AnotherDTO(dto.x, dto.y))
6   .filter(_.n != null)
7   .write
8   .json(path)
```

# Example

```
1 spark
2   .read
3   .csv(path)
4   .as[MyDTO]
5   .map(dto => AnotherDTO(dto.x, dto.y))
6   .filter(_.n != null)
7   .write
8   .json(path)
```

# Example

```
1 spark
2   .read
3   .csv(path)
4   .as[MyDTO]
5   .map(dto => AnotherDTO(dto.x, dto.y))
6   .filter(_.n != null)
7   .write
8   .json(path)
```

# What did I have

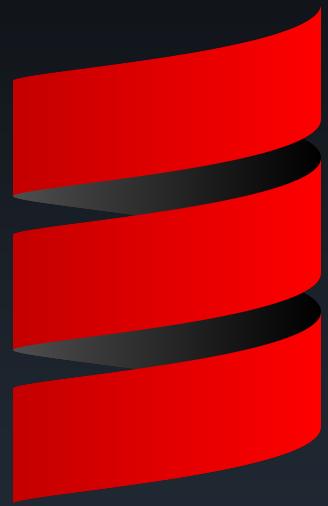
- Java
- Kotlin
- Scala
- Groovy
- bash, XML, YAML 😊

Sparse experience with other languages

# What did I have

- Core Java
- GC
- Lots of debug experience
- Distributed systems
- Architecture

# Supported languages





The most popular language in data engineering

```
1 from pyspark.sql import DataFrame, functions as F
2 from pyspark.sql.types import *
3
4 data_type = StructType([
5     StructField("pk", LongType(), False),
6     StructField("aa", StringType(), False)
7 ])
8
9 transformed_data = (
10    sample_df
11    .withColumn("json_object", F.get_json_object(F.col("json_value"), "$.data"))
12    .withColumn("parsed_struct", F.from_json(F.col("json_object"), data_type))
13    .selectExpr("parsed_struct.*")
14 )
```

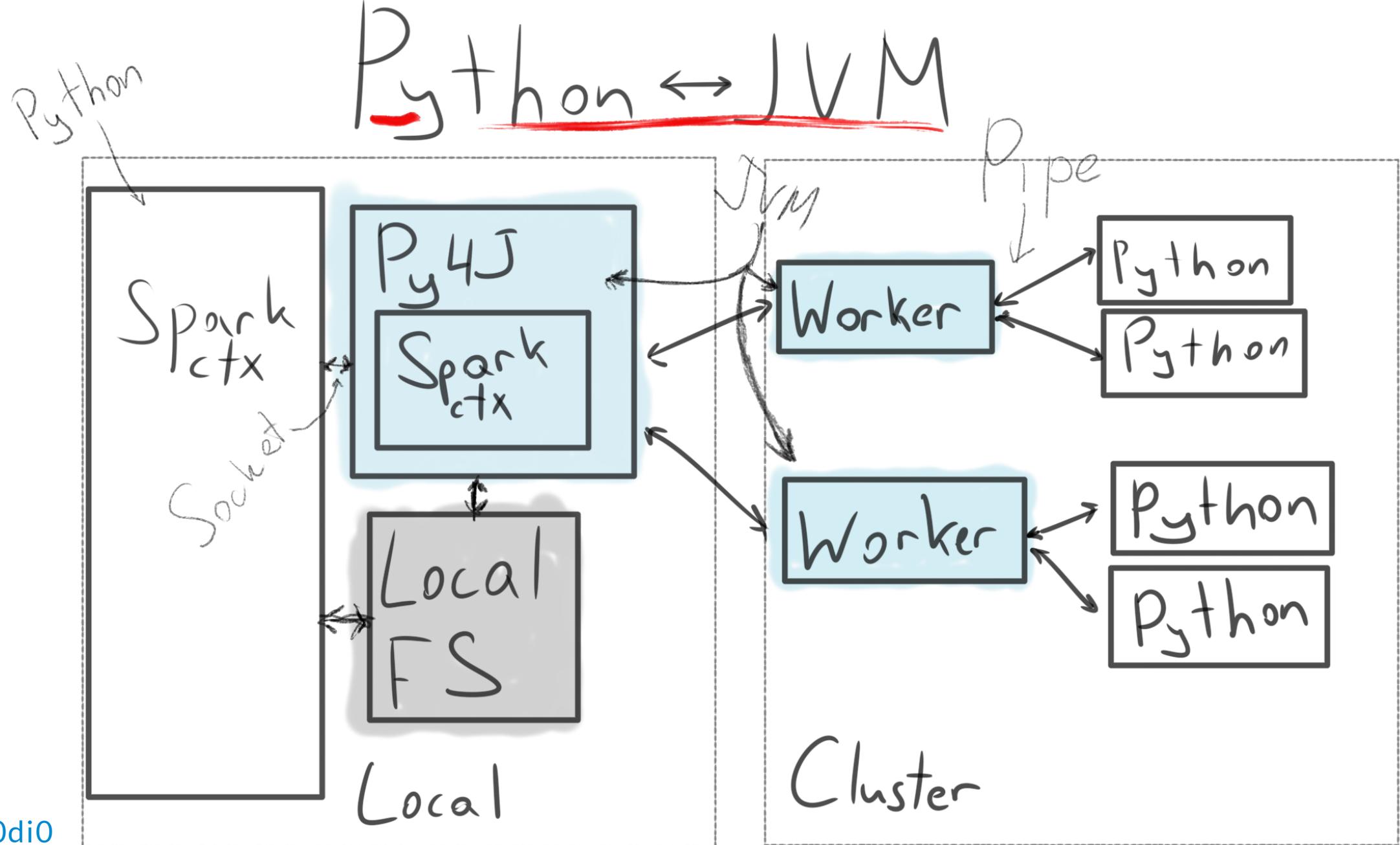
```
1 from pyspark.sql import DataFrame, functions as F
2 from pyspark.sql.types import *
3
4 data_type = StructType([
5     StructField("pk", LongType(), False),
6     StructField("aa", StringType(), False)
7 ])
8
9 transformed_data = (
10    sample_df
11    .withColumn("json_object", F.get_json_object(F.col("json_value"), "$.data"))
12    .withColumn("parsed_struct", F.from_json(F.col("json_object"), data_type))
13    .selectExpr("parsed_struct.*")
14 )
```

```
1 from pyspark.sql import DataFrame, functions as F
2 from pyspark.sql.types import *
3
4 data_type = StructType([
5     StructField("pk", LongType(), False),
6     StructField("aa", StringType(), False)
7 ])
8
9 transformed_data = (
10    sample_df
11    .withColumn("json_object", F.get_json_object(F.col("json_value"), "$.data"))
12    .withColumn("parsed_struct", F.from_json(F.col("json_object"), data_type))
13    .selectExpr("parsed_struct.*")
14 )
```

```
1 from pyspark.sql import DataFrame, functions as F
2 from pyspark.sql.types import *
3
4 data_type = StructType([
5     StructField("pk", LongType(), False),
6     StructField("aa", StringType(), False)
7 ])
8
9 transformed_data = (
10    sample_df
11    .withColumn("json_object", F.get_json_object(F.col("json_value"), "$.data"))
12    .withColumn("parsed_struct", F.from_json(F.col("json_object"), data_type))
13    .selectExpr("parsed_struct.*")
14 )
```

# Python

- F is made to distinct Spark built-ins from self-made.  
Who have ever created their own function over column?
- API is untyped
- Everything is string-based
- UDF s are SLOW
- Custom type support is complex



# Python

```
checkResult = VerificationSuite(spark) \
    .onData(df) \
    .addCheck(
        check.hasSize(lambda x: x >= 3) \
        .hasMin("b", lambda x: x == 0) \
        .isComplete("c") \
        .isUnique("a") \
        .isContainedIn("a", ["foo", "bar", "baz"]) \
        .isNonNegative("b")) \
    .run()
```

scala

# The best\* official API

- Typed and untyped APIs
- Awesome smart encoders
- Spark is written in Scala:
  - Best interop possible
- Huge ecosystem

\* by my own rating among official APIs

# The hard parts

```
implicit val z1 = 2
def addTo(n: Int) = {
    def add(x: Int)(y: Int)(implicit z: Int) = x + y + z
    add(n) _
}
val addTo1 = addTo(1)
addTo1(2)
```

# The hard parts

```
trait HashSet[+T] {  
    def add[U >: T](item: U)  
}
```

# The hard parts

```
trait Zulimba[+T] {  
    def gogoThere[U >: T](next: U)  
}
```

# The hard parts

- Hard to learn for people without JVM knowledge
- Hard to read code somebody else wrote
- Easy to abuse language features
  - operator overloading
  - implicit
  - traits

**Scala's type system is awesome and  
powerful!**

# **Scala's type system is awesome and powerful!**

Yes, but even Spark doesn't utilize it's full power

# Scala's type system is awesome and powerful!

Yes, but even Spark doesn't utilize it's full power

For the greater good

# Objects in Spark

```
1 val newInstance = NewInstance(  
2     cls, arguments, ObjectType(cls), propagateNull = false)  
3 expressions.If(  
4     IsNull(path),  
5     expressions.Literal.create(null, ObjectType(cls)),  
6     newInstance  
7 )
```

# Objects in Spark

```
1 val newInstance = NewInstance(  
2     cls, arguments, ObjectType(cls), propagateNull = false)  
3 expressions.If(  
4     IsNull(path),  
5     expressions.Literal.create(null, ObjectType(cls)),  
6     newInstance  
7 )
```

# Objects in Spark

```
1 val newInstance = NewInstance(  
2     cls, arguments, ObjectType(cls), propagateNull = false)  
3 expressions.If(  
4     IsNull(path),  
5     expressions.Literal.create(null, ObjectType(cls)),  
6     newInstance  
7 )
```

# Objects in Spark

```
1 val newInstance = NewInstance(  
2     cls, arguments, ObjectType(cls), propagateNull = false)  
3 expressions.If(  
4     IsNull(path),  
5     expressions.Literal.create(null, ObjectType(cls)),  
6     newInstance  
7 )
```

# Objects in Spark

```
1 val newInstance = NewInstance(  
2     cls, arguments, ObjectType(cls), propagateNull = false)  
3 expressions.If(  
4     IsNull(path),  
5     expressions.Literal.create(null, ObjectType(cls)),  
6     newInstance  
7 )
```

# Objects in Spark

```
1 val newInstance = NewInstance(  
2     cls, arguments, ObjectType(cls), propagateNull = false)  
3 expressions.If(  
4     IsNull(path),  
5     expressions.Literal.create(null, ObjectType(cls)),  
6     newInstance  
7 )
```

# Everything is expression

- NewInstance
- ObjectType
- If
- IsNull
- Literal

Type safety is not silver bullet

# java

# Java is verbose

```
public class Employee {  
    private String name;  
    private String id;  
    private double sal;  
    public String getName() { }  
    public void setName(String name) { }  
    public String getId() { }  
    public void setId(String id) { }  
    public double getSal() { }  
    public void setSal(double sal) { }  
    public int hashCode() { }  
    public boolean equals() { }  
}
```

```
Encoder<Person> personEncoder = Encoders.bean(Person.class);
Dataset<Person> javaBeanDS = spark.createDataset(
    Collections.singletonList(person),
    personEncoder
);
// --- and then ---
Encoder<Integer> integerEncoder = Encoders.INT();
Dataset<Integer> primitiveDS = spark.createDataset(
    Arrays.asList(1, 2, 3),
    integerEncoder);
Dataset<Integer> transformedDS = primitiveDS.map(
    (MapFunction<Integer, Integer>) value → value + 1,
    integerEncoder);
```

# Java is verbose

R

# R is hard

```
1 mtcars_tbl ← copy_to(sc, mtcars)
2
3 partitions ← mtcars_tbl %>%
4   filter(hp ≥ 100) %>%
5   mutate(cyl8 = cyl == 8) %>%
6   sdf_partition(training = 0.5, test = 0.5, seed = 1099)
```

# R is hard

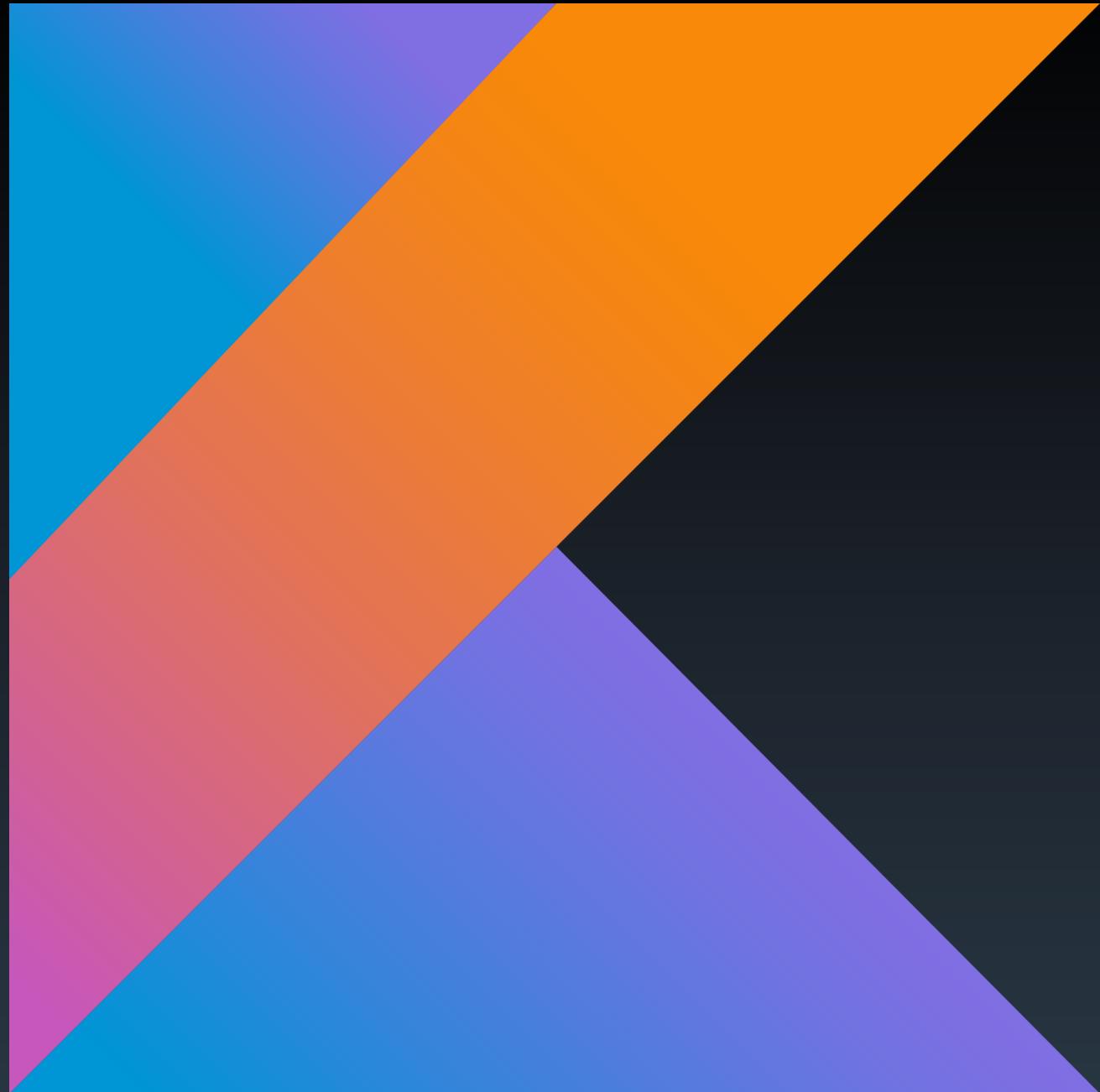
```
1 mtcars_tbl ← copy_to(sc, mtcars)
2
3 partitions ← mtcars_tbl %>%
4   filter(hp ≥ 100) %>%
5   mutate(cyl8 = cyl == 8) %>%
6   sdf_partition(training = 0.5, test = 0.5, seed = 1099)
```

# R is hard

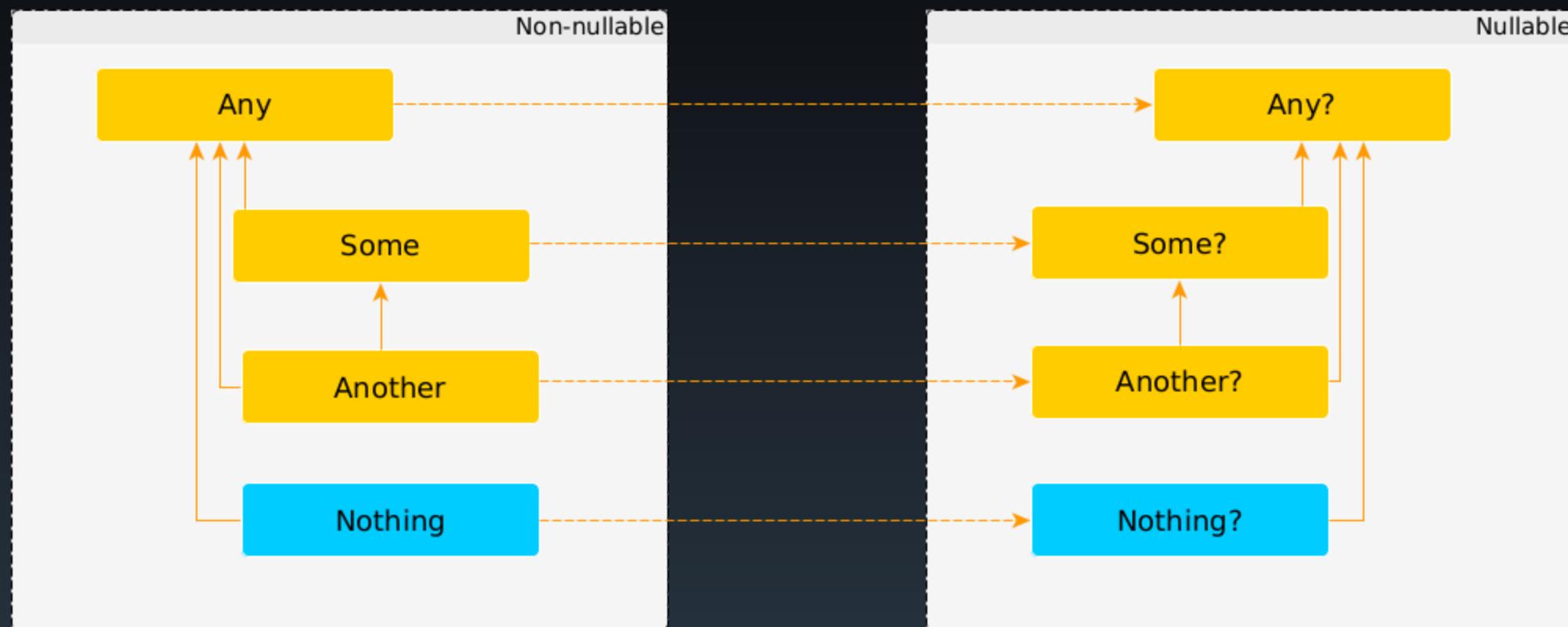
```
1 mtcars_tbl ← copy_to(sc, mtcars)
2
3 partitions ← mtcars_tbl %>%
4   filter(hp ≥ 100) %>%
5   mutate(cyl8 = cyl == 8) %>%
6   sdf_partition(training = 0.5, test = 0.5, seed = 1099)
```

# Thoughts on R

- There are ideas on deprecating SparkR
- There always will be performance gap between native (JVM) and other languages



# Null-aware type system



# Extension methods

```
fun Iterable<Int>.sum() = reduce { a, b -> a + b }
```

# Extension methods

```
fun Iterable<Int>.sum() = reduce { a, b -> a + b }
```

This  already exists in stdlib

# Join them together

Scala:

```
payment.join(customer, Seq("customerId"), "left_outer")
```

A bit later:

```
.map { _._2.id }
```

# NullPointerException



Because Java has unique exception for everything.

# We can do better

Kotlin:

```
payment.leftJoin(customer, col("customerId"))
```

A bit later

```
.map { it.second.id }
```

This  won't compile!

# We can do better

Kotlin:

```
payment.leftJoin(customer, col("customerId"))
```

A bit later

```
.map { it.second?.id }
```

This  will!

# DSL-building capabilities

```
1 fun html(init: HTML.() → Unit): HTML {  
2     val result = HTML()  
3     HTML.init() // or return HTML().apply { init () }  
4     return result  
5 }  
6 fun HTML.h1(text: String) = addElement("<h1>$text</h1>")  
7 html { h1("Example") }
```

# DSL-building capabilities

```
1 fun html(init: HTML.() → Unit): HTML {  
2     val result = HTML()  
3     HTML.init() // or return HTML().apply { init () }  
4     return result  
5 }  
6 fun HTML.h1(text: String) = addElement("<h1>$text</h1>")  
7 html { h1("Example") }
```

# DSL-building capabilities

```
1 fun html(init: HTML.() → Unit): HTML {  
2     val result = HTML()  
3     HTML.init() // or return HTML().apply { init () }  
4     return result  
5 }  
6 fun HTML.h1(text: String) = addElement("<h1>$text</h1>")  
7 html { h1("Example") }
```

# DSL-building capabilities

```
1 fun html(init: HTML.() → Unit): HTML {  
2     val result = HTML()  
3     HTML.init() // or return HTML().apply { init () }  
4     return result  
5 }  
6 fun HTML.h1(text: String) = addElement("<h1>$text</h1>")  
7 html { h1("Example") }
```

# DSL-building capabilities

```
1 fun html(init: HTML.() → Unit): HTML {  
2     val result = HTML()  
3     HTML.init() // or return HTML().apply { init () }  
4     return result  
5 }  
6 fun HTML.h1(text: String) = addElement("<h1>$text</h1>")  
7 html { h1("Example") }
```

# DSL-building capabilities

```
1 fun html(init: HTML.() → Unit): HTML {  
2     val result = HTML()  
3     HTML.init() // or return HTML().apply { init () }  
4     return result  
5 }  
6 fun HTML.h1(text: String) = addElement("<h1>$text</h1>")  
7 html { h1("Example") }
```

# Correct caching

```
1 ds0f(1, 2, 3, 4, 5)
2   .map { it to (it + 2) }
3   .withCached {
4     // source dataset is cached here
5     showDS()
6   } // and unpersisted here
7   .map { c(it.first, it.second, (it.first + it.second) * 2) }
```

# Correct caching

```
1 ds0f(1, 2, 3, 4, 5)
2   .map { it to (it + 2) }
3   .withCached {
4     // source dataset is cached here
5     showDS()
6   } // and unpersisted here
7   .map { c(it.first, it.second, (it.first + it.second) * 2) }
```

# withSpark

```
1 withSpark(logLevel = SparkLogLevel.INFO,
2             props = mapOf("spark.sqlcodegen.wholeStage" to true) {
3     val first = dsOf(Left(1, "a"), Left(2, "b"))
4     val second = dsOf(Right(1, 100), Right(3, 300))
5     first
6         .leftJoin(second, first.col("id").eq(second.col("id")))
7         .map { c(it.first.id, it.first.name, it.second?.value) }
8         .show()
9 }
```

# withSpark

```
1 withSpark(logLevel = SparkLogLevel.INFO,
2             props = mapOf("spark.sqlcodegen.wholeStage" to true) {
3     val first = dsOf(Left(1, "a"), Left(2, "b"))
4     val second = dsOf(Right(1, 100), Right(3, 300))
5     first
6         .leftJoin(second, first.col("id").eq(second.col("id")))
7         .map { c(it.first.id, it.first.name, it.second?.value) }
8         .show()
9 }
```

# withSpark

```
1 withSpark(logLevel = SparkLogLevel.INFO,
2             props = mapOf("spark.sqlcodegen.wholeStage" to true) {
3     val first = dsOf(Left(1, "a"), Left(2, "b"))
4     val second = dsOf(Right(1, 100), Right(3, 300))
5     first
6         .leftJoin(second, first.col("id").eq(second.col("id")))
7         .map { c(it.first.id, it.first.name, it.second?.value) }
8         .show()
9 }
```

# withSpark

```
1 withSpark(logLevel = SparkLogLevel.INFO,
2             props = mapOf("spark.sqlcodegen.wholeStage" to true) {
3     val first = dsOf(Left(1, "a"), Left(2, "b"))
4     val second = dsOf(Right(1, 100), Right(3, 300))
5     first
6         .leftJoin(second, first.col("id").eq(second.col("id")))
7         .map { c(it.first.id, it.first.name, it.second?.value) }
8         .show()
9 }
```

# withSpark signature

```
inline fun withSpark(props: Map<String, Any> = emptyMap(),  
                     master: String = "local[*]",  
                     appName: String = "Kotlin Spark Sample",  
                     logLevel: SparkLogLevel = ERROR,  
                     func: KSparkSession.() -> Unit)
```

# withSpark signature

```
inline fun withSpark(builder: SparkSession.Builder,  
                    logLevel: SparkLogLevel = ERROR,  
                    func: KSparkSession.() -> Unit)
```

# Kotlin for Spark benefits

- Easy to read
- No encoders
- Null-safety
- Extension methods and DSL helpers
- Scala-like API
- Option to opt-out from any "non-native" experience
- Strong typing

# Things to improve

- ML support
- Better syntax for UDFs
- More audience
- More extensions
- Scala 3 (?)

[github.com/JetBrains/kotlin-spark-api](https://github.com/JetBrains/kotlin-spark-api)

Thank you! 

Pasha Finkelshteyn, JetBrains

 [asm0di0](https://twitter.com/asm0di0)