Extending Spark's Ingestion: Build Your Own Java Data Source

Jean Georges Perrin, @jgperrin, Oplo

#EUdev6

JGP

[@jgperrin](https://twitter.com/jgperrin)

Jean Georges Perrin

Chapel Hill, NC, USA

I🏗SW

[#Knowledge](#) =

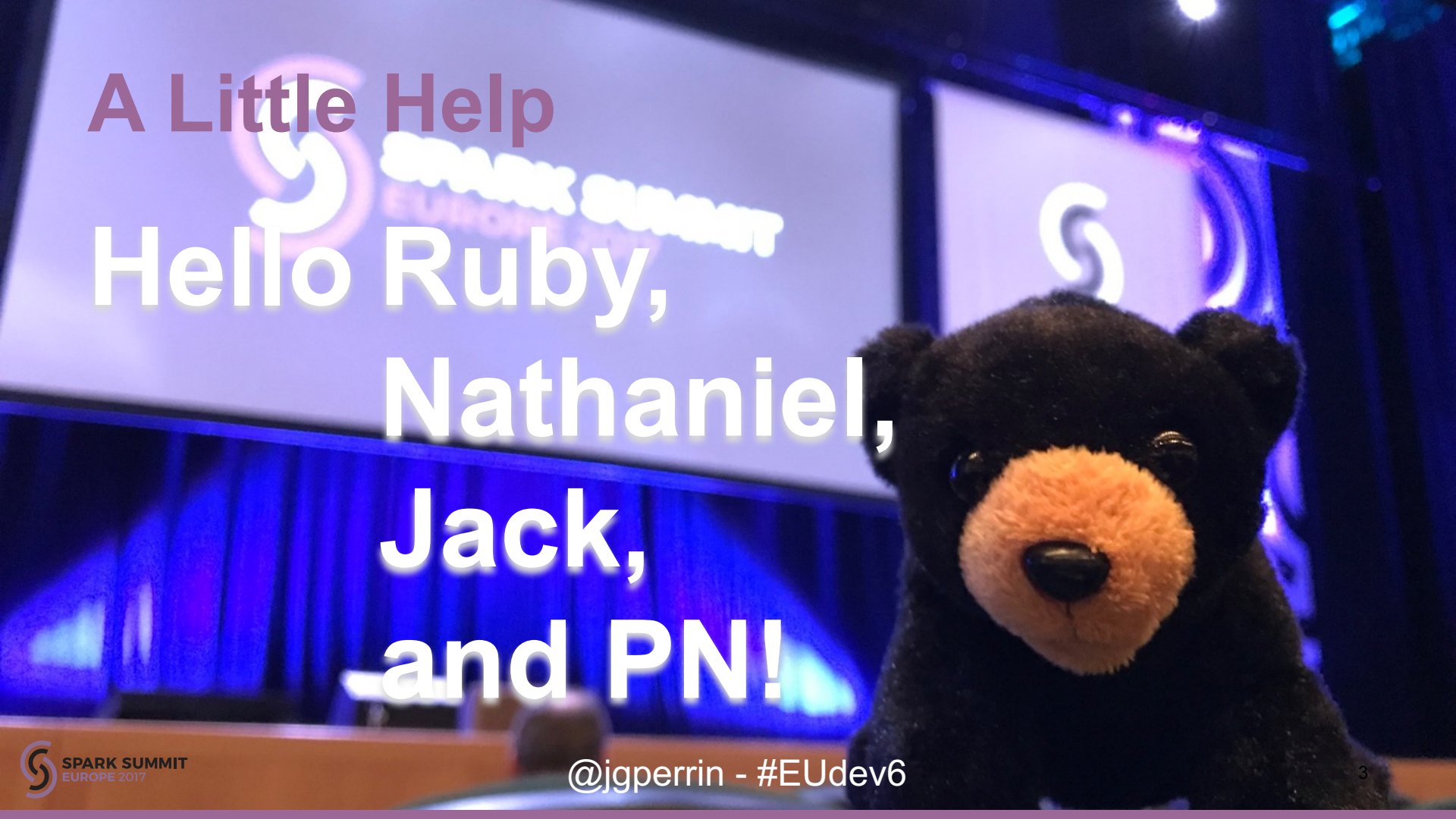
f ($\sum$ ([#SmallData](#), [#BigData](#)), [#DataScience](#))
& [#Software](#)

[#IBMChampion](#) x9 [#KeepLearning](#)

@ http://jgp.net

oplo.io

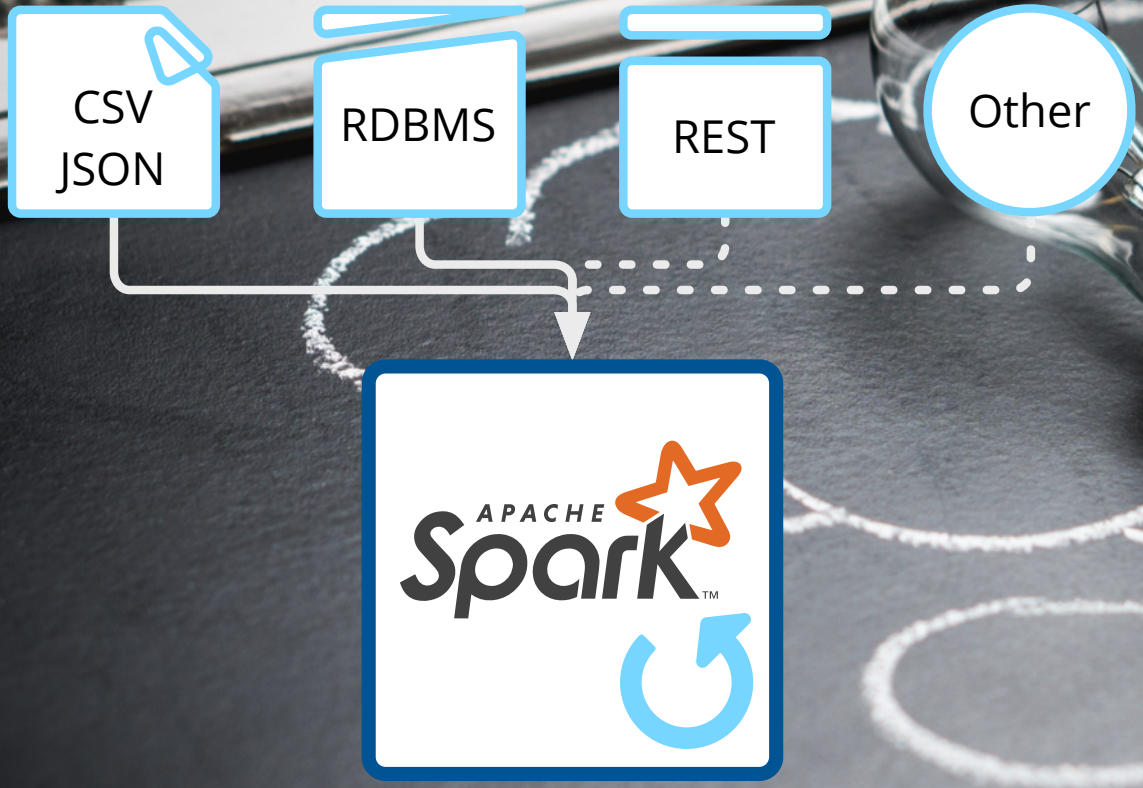
IBM CHAMPION


A Little Help
Hello Ruby,
Nathaniel,
Jack,
and PN!

And What About You?

- Who is a Java programmer?
- Who swears by Scala?
- Who has tried to build a **custom data source**?
- In Java?
- Who has succeeded?

My Problem…

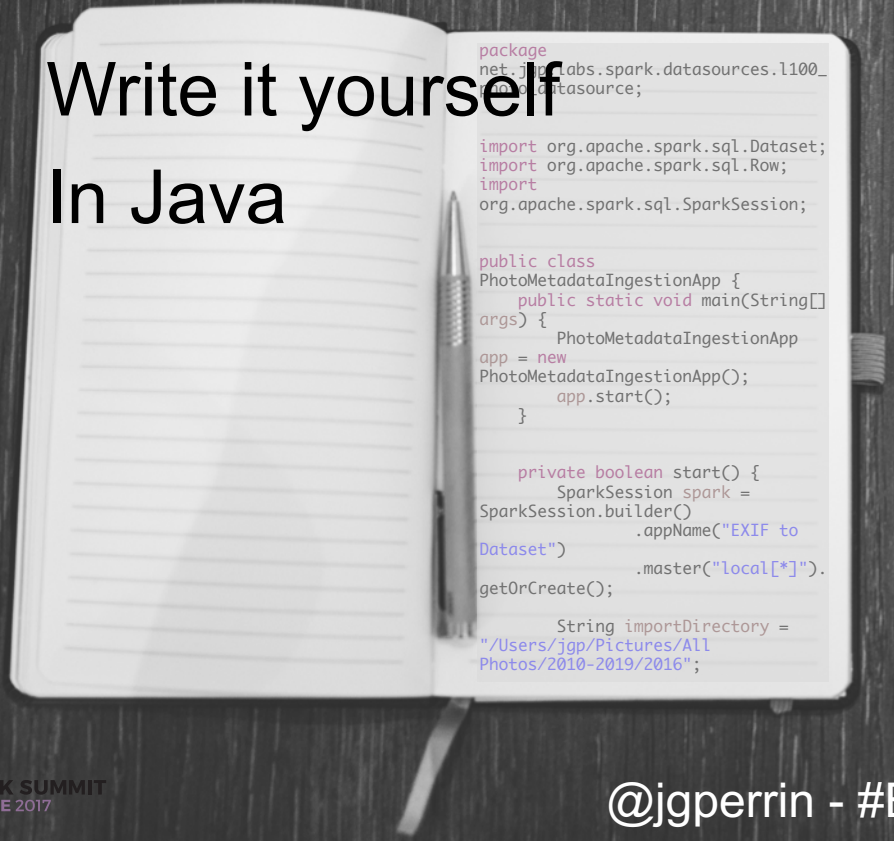


Solution #1

- Look in the Spark Packages library
- https://spark-packages.org/
- 48 data sources listed (some dups)
- Some with Source Code

Solution #2

- Write it yourself
- In Java



```
package
net.jgp.books.spark.datasources.l100_
exif.datasource;

import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import
org.apache.spark.sql.SparkSession;


public class
PhotoMetadataIngestionApp {
    public static void main(String[]
args) {
        PhotoMetadataIngestionApp
app = new
PhotoMetadataIngestionApp();
        app.start();
    }


    private boolean start() {
        SparkSession spark =
SparkSession.builder()
                .appName("EXIF to
Dataset")
                .master("local[*]").
getOrCreate();

        String importDirectory =
"/Users/jgp/Pictures/All
Photos/2010-2019/2016";
```

What is ingestion anyway?

- Loading
  - CSV, JSON, RDBMS…
  - Basically any data
- Getting a Dataframe (`Dataset<Row>`) with your data

Our lab

Import some of the metadata of our photos and start analysis…

Code

```
Dataset<Row> df = spark.read()
        .format("exif")
        .option("recursive", "true")
        .option("limit", "80000")
        .option("extensions", "jpg,jpeg")
        .load(importDirectory);
```

Spark Session

Short name for your data source

Options

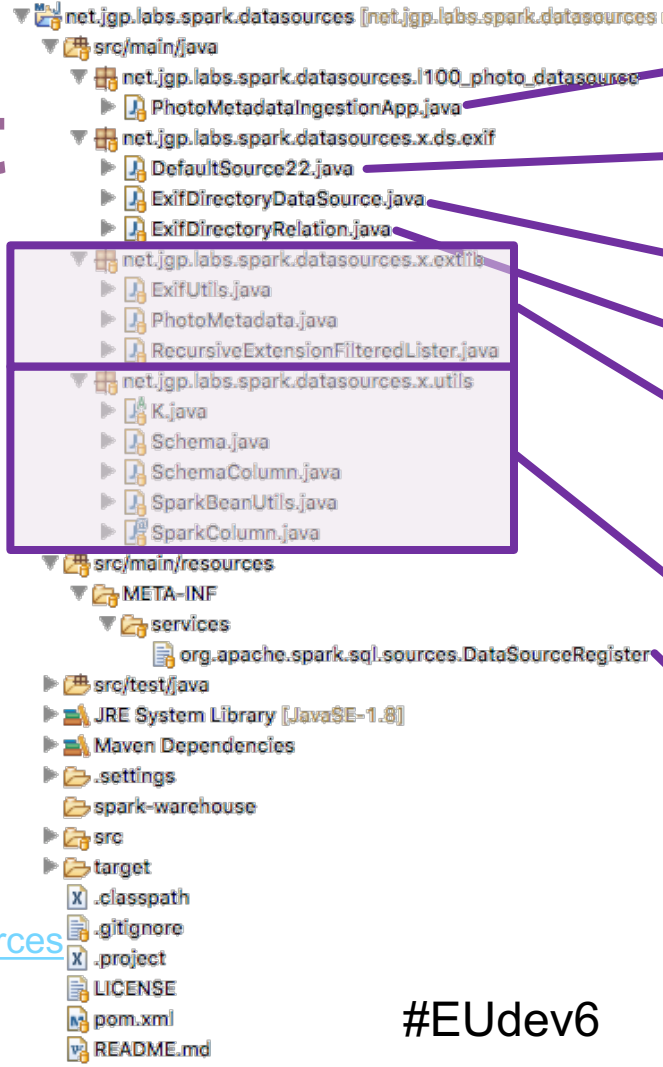
Where to start

/jgperrin/net.jgp.labs.spark.datasources

Short name

- Optional: can specify a full class name
- An alias to a class name
- Needs to be registered
- Not recommended during development phase

Project



The app

Data source's short name

Data source

Relation

An existing library you can embed/link to

Utilities

Information for registration of the data source

/jgperrin
/net.jgp.labs.spark.datasources

Library code

- A library you already use, you developed or acquired
- No need for anything special

Data Source Code

```
public class ExifDirectoryDataSource implements RelationProvider {

    @Override
    public BaseRelation createRelation(
            SQLContext sqlContext,
            Map<String, String> params) {

        java.util.Map<String, String> javaMap =
            mapAsJavaMapConverter(params).asJava();

        ExifDirectoryRelation br = new ExifDirectoryRelation();
        br.setSqlContext(sqlContext);

        ...

        br.setPhotoLister(photoLister);
        return br;
    }
}
```

Provides a relation

Needed by Spark

Needs to be passed to the relation

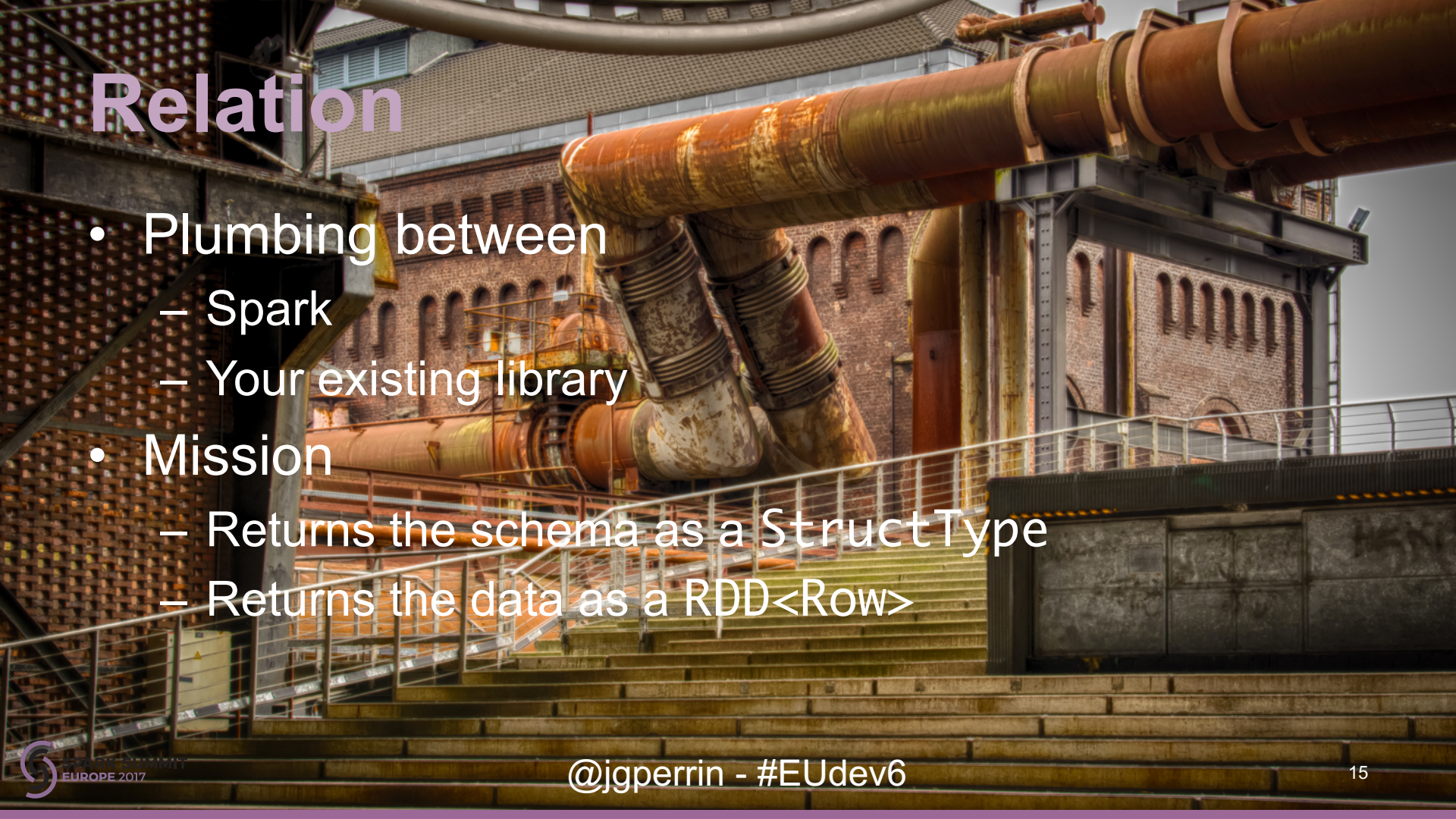
Scala 2 Java conversion (sorry!) – All the options

The relation to be exploited

/jgperrin
/net.jgp.labs.spark.datasources

Relation

- Plumbing between
  - Spark
  - Your existing library
- Mission
  - Returns the schema as a `StructType`
  - Returns the data as a RDD<Row>



Relation

```
public class ExifDirectoryRelation extends BaseRelation
        implements Serializable, TableScan {
    private static final long serialVersionUID = 4598175080399877334L;

    @Override
    public RDD<Row> buildScan() {
...
        return rowRDD.rdd();
    }

    @Override
    public StructType schema() {
...
        return schema.getSparkSchema();
    }


    @Override
    public SQLContext sqlContext() {
        return this.sqlContext;
    }
...
}
```

TableScan is the key, other more specialized Scan available

The data…

The schema: may/will be called first

SQL Context

/jgperrin
/net.jgp.labs.spark.datasources

Relation – Schema

```
@Override
public StructType schema() {
    if (schema == null) {
        schema =
            SparkBeanUtils.getSchemaFromBean(PhotoMetadata.class);
    }
    return schema.getSparkSchema();
}
```

A utility function that introspect a Java bean and turn it into a ”Super” schema, which contains the required `StructType` for Spark

/jgperrin
/net.jgp.labs.spark.datasources

Relation – Data

```
@Override
public RDD<Row> buildScan() {
    schema();

    List<PhotoMetadata> table = collectData();

    JavaSparkContext sparkContext = new JavaSparkContext(sqlContext.sparkContext());
    JavaRDD<Row> rowRDD = sparkContext.parallelize(table)
            .map(photo -> SparkBeanUtils.getRowFromBean(schema, photo));

    return rowRDD.rdd();
}

private List<PhotoMetadata> collectData() {
    List<File> photosToProcess = this.photoLister.getFiles();
    List<PhotoMetadata> list = new ArrayList<PhotoMetadata>();
    PhotoMetadata photo;

    for (File photoToProcess : photosToProcess) {
        photo = ExifUtils.processFromFilename(photoToProcess.getAbsolutePath());
        list.add(photo);
    }
    return list;
}
```

Collect the data

Creates the RDD by parallelizing the list of photos

Scans the files, extract EXIF information: the interface to your library…

/jgperrin
/net.jgp.labs.spark.datasources

Application

```
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SparkSession;

public class PhotoMetadataIngestionApp {
    public static void main(String[] args) {
        PhotoMetadataIngestionApp app = new PhotoMetadataIngestionApp();
        app.start();
    }

    private boolean start() {
        SparkSession spark = SparkSession.builder()
            .appName("EXIF to Dataset").master("local[*]").getOrCreate();

        Dataset<Row> df = spark.read()
            .format("exif")
            .option("recursive", "true")
            .option("limit", "80000")
            .option("extensions", "jpg,jpeg")
            .load("/Users/jgp/Pictures");

        df = df
            .filter(df.col("GeoX").isNotNull())
            .filter(df.col("GeoZ").notEqual("NaN"))
            .orderBy(df.col("GeoZ").desc());

        System.out.println("I have imported " + df.count() + " photos.");
        df.printSchema();
        df.show(5);
        return true;
    }
}
```

Normal imports, no reference to our data source

Local mode

Classic read

Standard dataframe API: getting my ”highest” photos!

Standard output mechanism

/jgperrin
/net.jgp.labs.spark.datasources

Output

```
+--------------------+-------+---------+--------+--------------------+--------------------+--------------------+--------------------+--------------------+----------+--------------------+----------+----------+------+-----+
|                Name|   Size|Extension|MimeType|           Directory|    FileCreationDate| FileLastAccessDate| FileLastModifiedDate|            Filename|      GeoY|                Date|      GeoX|      GeoZ|Height|Width|
+--------------------+-------+---------+--------+--------------------+--------------------+--------------------+--------------------+--------------------+----------+--------------------+----------+----------+------+-----+
|        IMG_0177.JPG|1129296|      JPG|  image/jpeg|/Users/jgp/Pictur...|2015-04-14 08:14:42|2017-10-24 11:20:31| 2015-04-14 08:14:42|/Users/jgp/Pictur...| -111.17341|2015-04-14 15:14:42| 36.38115|  9314.743|  1936| 2592|
|        IMG_0176.JPG|1206265|      JPG|  image/jpeg|/Users/jgp/Pictur...|2015-04-14 09:14:36|2017-10-24 11:20:31| 2015-04-14 09:14:36|/Users/jgp/Pictur...| -111.17341|2015-04-14 16:14:36| 36.38115|  9314.743|  1936| 2592|
|        IMG_1202.JPG|1799552|      JPG|  image/jpeg|/Users/jgp/Pictur...|2015-05-05 17:52:15|2017-10-24 11:20:28| 2015-05-05 17:52:15|/Users/jgp/Pictur...| -98.471405|2015-05-05 15:52:15|29.528196|  9128.197|  2448| 3264|
|        IMG_1203.JPG|2412908|      JPG|  image/jpeg|/Users/jgp/Pictur...|2015-05-05 18:05:25|2017-10-24 11:20:28| 2015-05-05 18:05:25|/Users/jgp/Pictur...|   -98.4719|2015-05-05 16:05:25|29.526403|  9128.197|  2448| 3264|
|        IMG_1204.JPG|2261133|      JPG|  image/jpeg|/Users/jgp/Pictur...|2015-05-05 15:13:25|2017-10-24 11:20:29| 2015-05-05 15:13:25|/Users/jgp/Pictur...|  -98.47248|2015-05-05 16:13:25|29.526756|  9128.197|  2448| 3264|
+--------------------+-------+---------+--------+--------------------+--------------------+--------------------+--------------------+--------------------+----------+--------------------+----------+----------+------+-----+
only showing top 5 rows
```

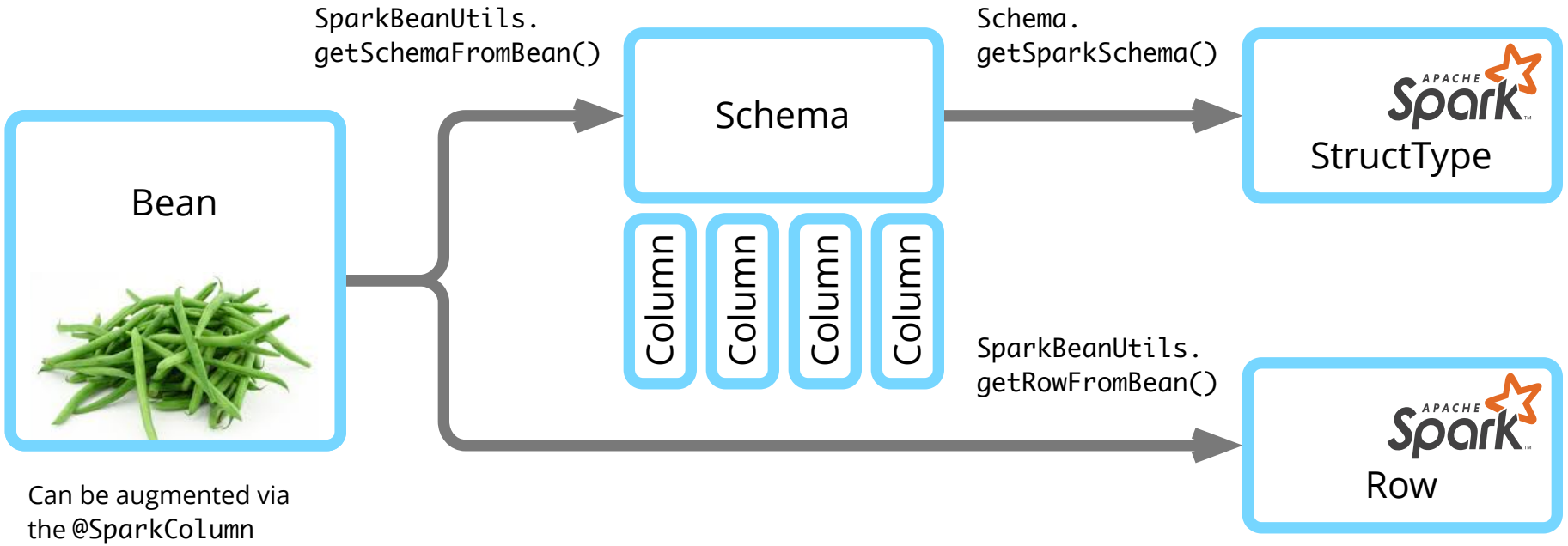
```
+--------------------+-------+----------+--------------------+---------+----------+
|                Name|   Size|      GeoY|                Date|     GeoX|      GeoZ|
+--------------------+-------+----------+--------------------+---------+----------+
|        IMG_0177.JPG|1129296| -111.17341|2015-04-14 15:14:42| 36.38115|  9314.743|
|        IMG_0176.JPG|1206265| -111.17341|2015-04-14 16:14:36| 36.38115|  9314.743|
|        IMG_1202.JPG|1799552| -98.471405|2015-05-05 15:52:15|29.528196|  9128.197|
|        IMG_1203.JPG|2412908|   -98.4719|2015-05-05 16:05:25|29.526403|  9128.197|
|        IMG_1204.JPG|2261133|  -98.47248|2015-05-05 16:13:25|29.526756|  9128.197|
+--------------------+-------+----------+--------------------+---------+----------+
only showing top 5 rows
```

SPARK SUMMIT EUROPE 2017

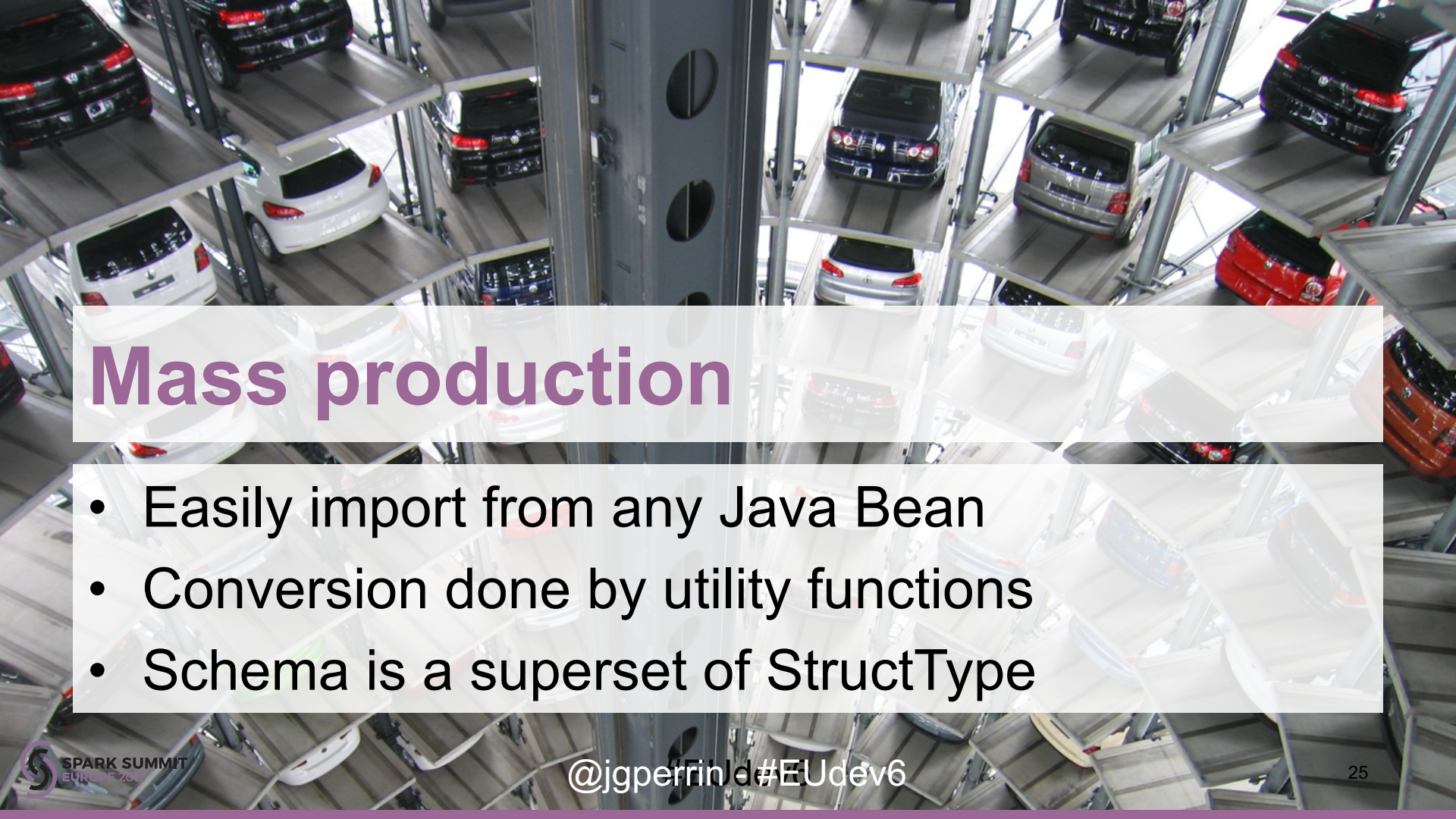
SPARK SUMMIT EUROPE 2017

A Little Extra

Schema, Beans, and Annotations



Can be augmented via the `@SparkColumn` annotation



Mass production

- Easily import from any Java Bean
- Conversion done by utility functions
- Schema is a superset of StructType

Conclusion

- **Reuse** the Bean to schema and Bean to data in your project
- Building custom data source to REST server or non standard format is **easy**
- **No need for a pricey conversion** to CSV or JSON
- There is **always** a solution in Java
- *On you:* check for parallelism, optimization, extend schema (order of columns)

Going Further

- Check out the code (fork/like) https://github.com/jgperrin/net.jgp.labs.spark.datasources
- Follow me @jgperrin
- Watch for my Java + Spark book, coming out soon!
- (If you come in the RTP area in NC, USA, come for a Spark meet-up and let's have a drink!)

Don't forget to rate

Go raibh maith agaibh.

Jean Georges "JGP" Perrin

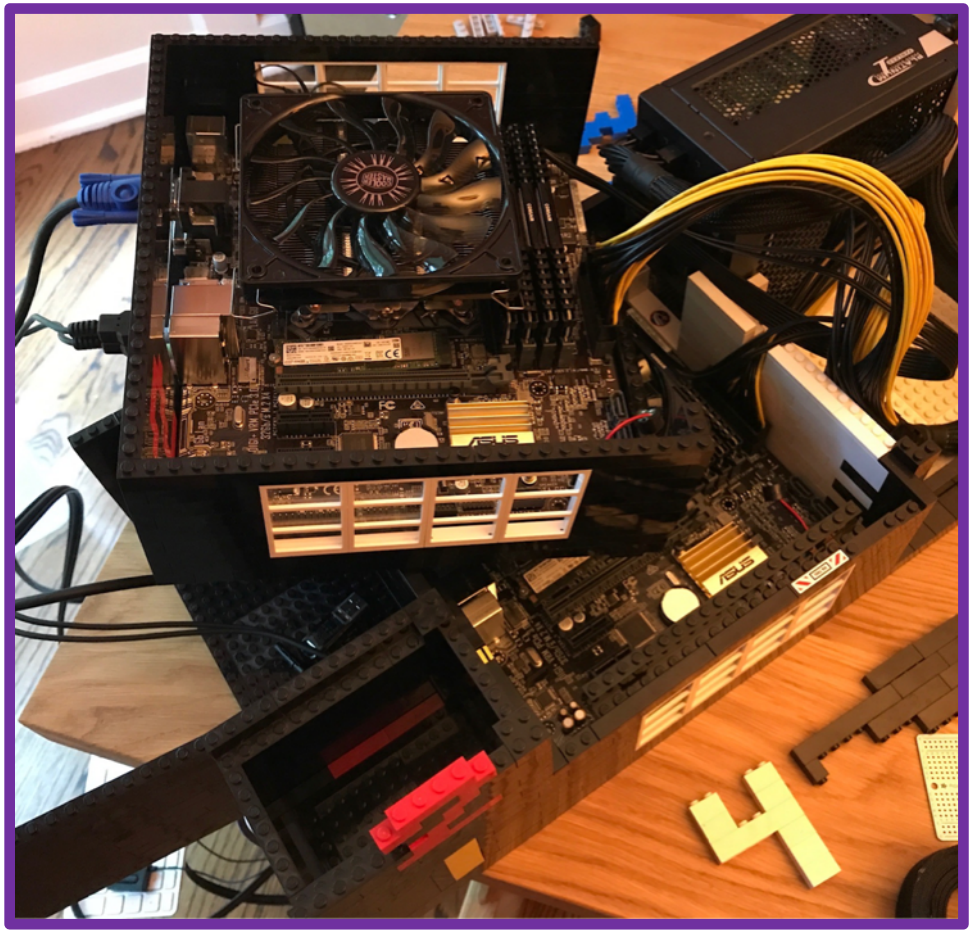
@jgperrin

SPARK SUMMIT EUROPE 2017

oplo.io

IBM CHAMPION

Backup Slides & Other Resources



Discover CLEGO @ http://bit.ly/spark-clego

More Reading & Resources

- My blog: http://jgp.net
- This code on GitHub: https://github.com/jgperrin/net.jgp.labs.spark.datasources
- Java Code on GitHub: https://github.com/jgperrin/net.jgp.labs.spark

/jgperrin/net.jgp.labs.spark.datasources

Photo & Other Credits

- JGP @ Oplo, Problem, Solution 1&2, Relation, Beans: Pexels
- Lab, Hoover Dam, Clego: © Jean Georges Perrin
- Rijsttafel: © Nathaniel Perrin

Abstract

EXTENDING APACHE SPARK'S INGESTION: BUILDING YOUR OWN JAVA DATA SOURCE

By Jean Georges Perrin (@jgperrin, Oplo)

Apache Spark is a wonderful platform for running your analytics jobs. It has great ingestion features from CSV, Hive, JDBC, etc. however, you may have your own data sources or formats you want to use. Your solution could be to convert your data in a CSV or JSON file and then ask Spark to do ingest it through its built-in tools. However, for enhanced performance, we will explore the way to build a data source, in Java, to extend Spark's ingestion capabilities. We will first understand how Spark works for ingestion, then walk through the development of this data source plug-in.

Targeted audience: Software and data engineers who need to expand Spark's ingestion capability.

Key takeaways:

- Requirements, needs & architecture – 15%.
- Build the required tool set in Java – 85%.

Session hashtag: #EUdev6