Development with Apache Spark: Scala vs. Pyspark

BigData

BigData

- ...

BigData

- ...
- Apache Spark

BigData

- ...
- Apache Spark:
  - R
  - Java
  - Scala
  - Python

BigData

- ...
- Apache Spark:
  - R
  - Java
  - **Scala**
  - **Python**

kolodziejj.info
@unit03

Warm-up

```
import org.apache.spark.sql.types.{DoubleType,
IntegerType, StructField, StructType}
import org.apache.spark.sql.{DataFrame, Row,
SparkSession}
import scala.collection.JavaConversions._
```

```scala
import org.apache.spark.sql.types.{DoubleType,
IntegerType, StructField, StructType}
import org.apache.spark.sql.{DataFrame, Row,
SparkSession}
import scala.collection.JavaConversions._

object WarmUp {
  def getSparkSession: SparkSession = {
    SparkSession
      .builder
      .master("local[*]")
      .appName("local-spark")
      .getOrCreate()
  }
}
```

```scala
import org.apache.spark.sql.types.{DoubleType,
IntegerType, StructField, StructType}
import org.apache.spark.sql.{DataFrame, Row,
SparkSession}
import scala.collection.JavaConversions._

object WarmUp {
  def getSparkSession: SparkSession = {
    SparkSession
      .builder
      .master("local[*]")
      .appName("local-spark")
      .getOrCreate()
  }
}
```

```python
from pyspark.sql import (
    DataFrame,
    Row,
    SparkSession,
)
from pyspark.sql.types import (
     DoubleType,
     IntegerType,
     StructField,
     StructType,
)


def get_spark_session() -> SparkSession:
    return (
        SparkSession
        .builder
        .master("local[*]")
        .appName("local-spark")
        .getOrCreate()
    )
```

```
...

object WarmUp {
  ...
  def createData(
    sparkSession: SparkSession
  ): DataFrame = {
    val schema = StructType(
      Array(
        StructField("a", IntegerType),
        StructField("b", DoubleType),
      ),
    )

    sparkSession.createDataFrame(
      Seq(
        Row(2, 3.0),
        Row(1, 2.5),
        Row(2, 5.2),
      ),
      schema,
    )
  }
}
```

```
...

object WarmUp {
  ...
  def createData(
    sparkSession: SparkSession
  ): DataFrame = {
    val schema = StructType(
      Array(
        StructField("a", IntegerType),
        StructField("b", DoubleType),
      ),
    )

    sparkSession.createDataFrame(
      Seq(
        Row(2, 3.0),
        Row(1, 2.5),
        Row(2, 5.2),
      ),
      schema,
    )
  }
}
```

```
...

object WarmUp {
  ...
  def createData(
    sparkSession: SparkSession
  ): DataFrame = {
    val schema = StructType(
      Array(
        StructField("a", IntegerType),
        StructField("b", DoubleType),
      ),
    )

    sparkSession.createDataFrame(
      Seq(
        Row(2, 3.0),
        Row(1, 2.5),
        Row(2, 5.2),
      ),
      schema,
    )
  }
}
```

```
...

object WarmUp {
  ...
  def createData(
    sparkSession: SparkSession
  ): DataFrame = {
    val schema = StructType(
      Array(
        StructField("a", IntegerType),
        StructField("b", DoubleType),
      ),
    )

    sparkSession.createDataFrame(
      Seq(
        Row(2, 3.0),
        Row(1, 2.5),
        Row(2, 5.2),
      ),
      schema,
    )
  }
}
```

```scala
...

object WarmUp {
  ...
  def createData(
    sparkSession: SparkSession
  ): DataFrame = {
    val schema = StructType(
      Array(
        StructField("a", IntegerType),
        StructField("b", DoubleType),
      ),
    )

    sparkSession.createDataFrame(
      Seq(
        Row(2, 3.0),
        Row(1, 2.5),
        Row(2, 5.2),
      ),
      schema,
    )
  }
}
```

```python
...

def create_data(
    spark: SparkSession,
) -> DataFrame:
    schema = StructType(
        [
            StructField("a", IntegerType()),
            StructField("b", DoubleType()),
        ],
    )

    return spark.createDataFrame(
        [
            Row(a=2, b=3.),
            Row(a=1, b=2.5),
            Row(a=2, b=5.2),
        ],
        schema,
    )
```

```
...

object WarmUp {
  ...
  def main(args: Array[String]): Unit = {
    val spark: SparkSession = getSparkSession

    val df: DataFrame = createData(spark)

    df.printSchema()
    df.show()

    spark.stop()
  }
}
```

```scala
...

object WarmUp {
  ...
  def main(args: Array[String]): Unit = {
    val spark: SparkSession = getSparkSession

    val df: DataFrame = createData(spark)

    df.printSchema()
    df.show()

    spark.stop()
  }
}
```

```python
...

def main() -> None:
    spark = get_spark_session()

    df = create_data(spark)

    df.printSchema()
    df.show()

    spark.stop()


if __name__ == "__main__":
    main()
```

```
echo ‘name := "Foo"

version := "1.0"

scalaVersion := "2.12.10"

libraryDependencies += "org.apache.spark" %%
"spark-sql" % "3.1.2"
libraryDependencies += "org.scalatest" %%
"scalatest" % "3.2.9" % "test"
‘ > build.sbt
```

```
echo ‘name := "Foo"

version := "1.0"

scalaVersion := "2.12.10"

libraryDependencies += "org.apache.spark" %%
"spark-sql" % "3.1.2"
libraryDependencies += "org.scalatest" %%
"scalatest" % "3.2.9" % "test"
‘ > build.sbt

sbt package
```

```
echo ‘name := "Foo"

version := "1.0"

scalaVersion := "2.12.10"

libraryDependencies += "org.apache.spark" %%
"spark-sql" % "3.1.2"
libraryDependencies += "org.scalatest" %%
"scalatest" % "3.2.9" % "test"
‘ > build.sbt

sbt package
spark-submit \
  --class "WarmUp" \
  target/scala-2.12/foo_2.12-1.0.jar
```

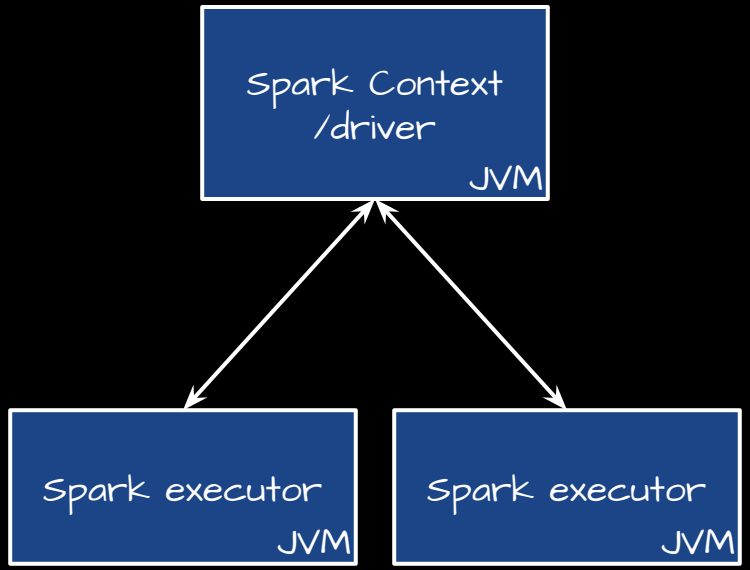
```
echo ‘name := "Foo"

version := "1.0"

scalaVersion := "2.12.10"

libraryDependencies += "org.apache.spark" %%
"spark-sql" % "3.1.2"
libraryDependencies += "org.scalatest" %%
"scalatest" % "3.2.9" % "test"
‘ > build.sbt

sbt package
spark-submit \
  --class "WarmUp" \
  target/scala-2.12/foo_2.12-1.0.jar
```

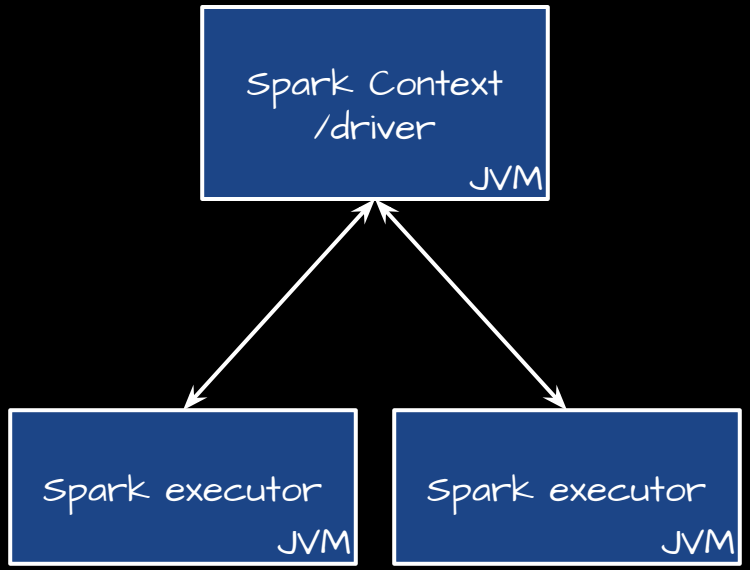
```
pip install pyspark>=3
python src/warm_up.py
```

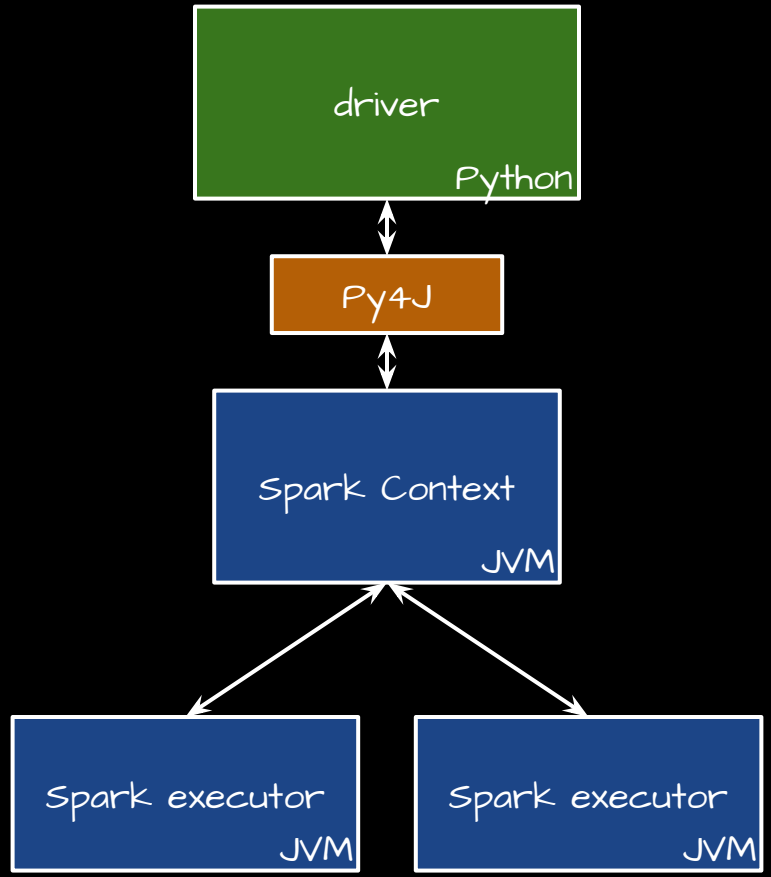
```
[...sth about compilation...]
[...some warnings...]
[...some info...]
root
 |-- a: integer (nullable = true)
 |-- b: double (nullable = true)

[...some more info...]
+---+---+
|  a|  b|
+---+---+
|  2|3.0|
|  1|2.5|
|  1|5.2|
+---+---+
```

```
[...sth about compilation...]
[...some warnings...]
[...some info...]
root
 |-- a: integer (nullable = true)
 |-- b: double (nullable = true)

[...some more info...]
+---+---+
|  a|  b|
+---+---+
|  2|3.0|
|  1|2.5|
|  1|5.2|
+---+---+
```

```
[...some warnings...]
root
 |-- a: integer (nullable = true)
 |-- b: double (nullable = true)

+---+---+
|  a|  b|
+---+---+
|  2|3.0|
|  1|2.5|
|  2|5.2|
+---+---+
```


Spark Context
/driver
JVM
Spark executor
JVM
Spark executor
JVM


Spark Context
/driver
JVM
Spark executor
JVM
Spark executor
JVM


driver
Python
Py4J
Spark Context
JVM
Spark executor
JVM
Spark executor
JVM

Typing

Typing

- strong

Typing

- strong
- static

Typing

- strong
- static
- => many errors detected during compilation phase

Typing

- strong
- static
- => many errors detected during compilation phase

- strong

Typing

- strong
- static
- => many errors detected during compilation phase

- strong
- dynamic

Typing

- strong
- static
- => many errors detected during compilation phase

- strong
- dynamic
- => we need more tests and more static analysis

Typing

- strong
- static
- => many errors detected during compilation phase

- strong
- dynamic
- => we need more tests and more static analysis
- optional type hints
  - personally: recommended!!

Spark APIs

- RDDs (Resilient Distributed Datasets)

Spark APIs

- RDDs (Resilient Distributed Datasets)
- DataFrames

Spark APIs

- RDDs (Resilient Distributed Datasets)
- DataFrames
- Datasets (statically typed)

Spark APIs

- RDDs (Resilient Distributed Datasets)
- DataFrames
- Datasets (statically typed)
  - only in Scala

Spark APIs

- RDDs (Resilient Distributed Datasets)
- DataFrames
- Datasets (statically typed)
  - only in Scala
- Spark SQL

Spark APIs

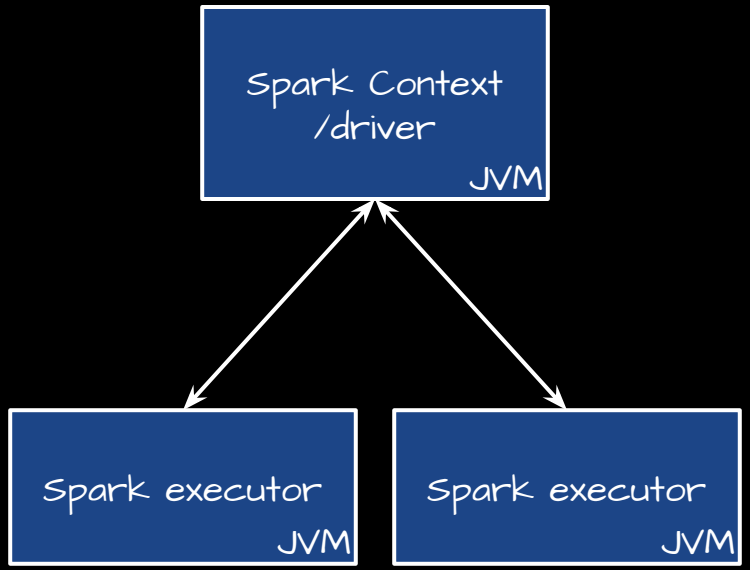
- RDDs (Resilient Distributed Datasets)
- DataFrames
- Datasets (statically typed)
  - only in Scala
- Spark SQL
- UDFs (User-Defined Functions)

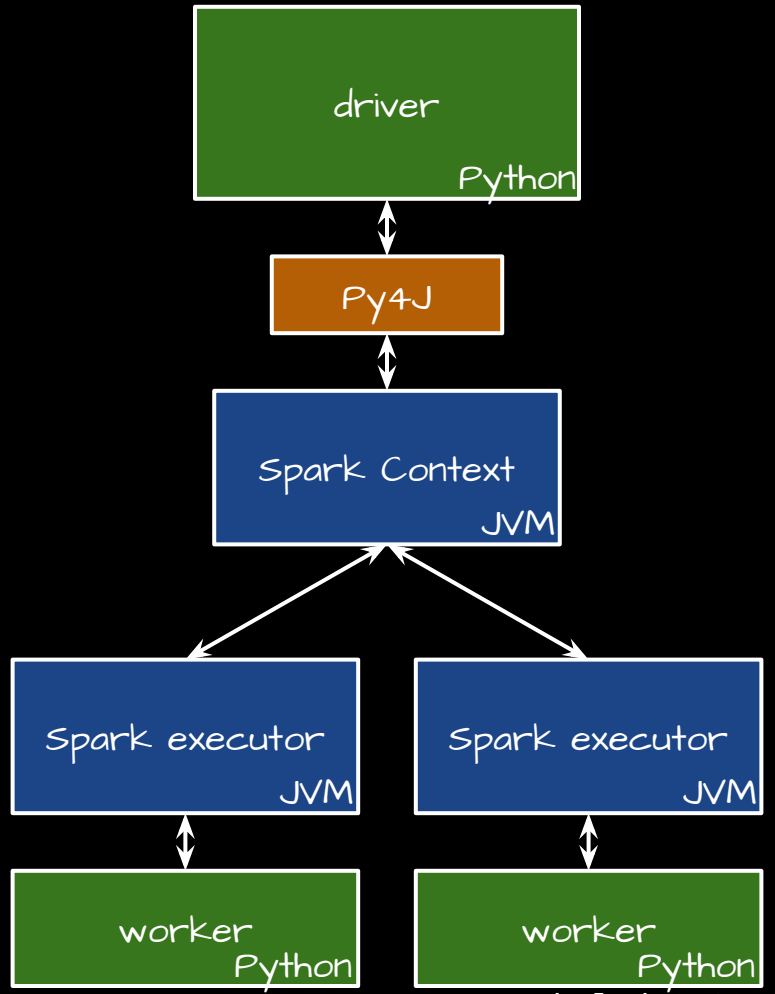
Spark APIs

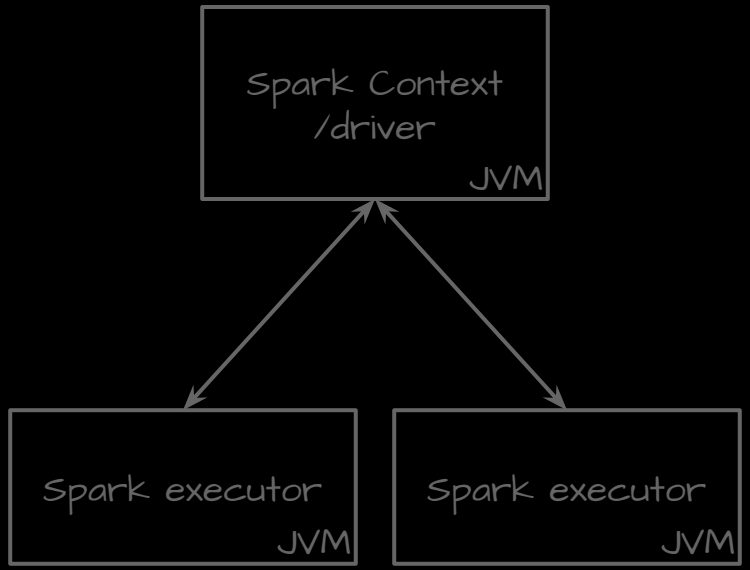
- RDDs (Resilient Distributed Datasets)
- DataFrames
- Datasets (statically typed)
  - only in Scala
- Spark SQL
- UDFs (User-Defined Functions)
  - Scala: within JVM processes

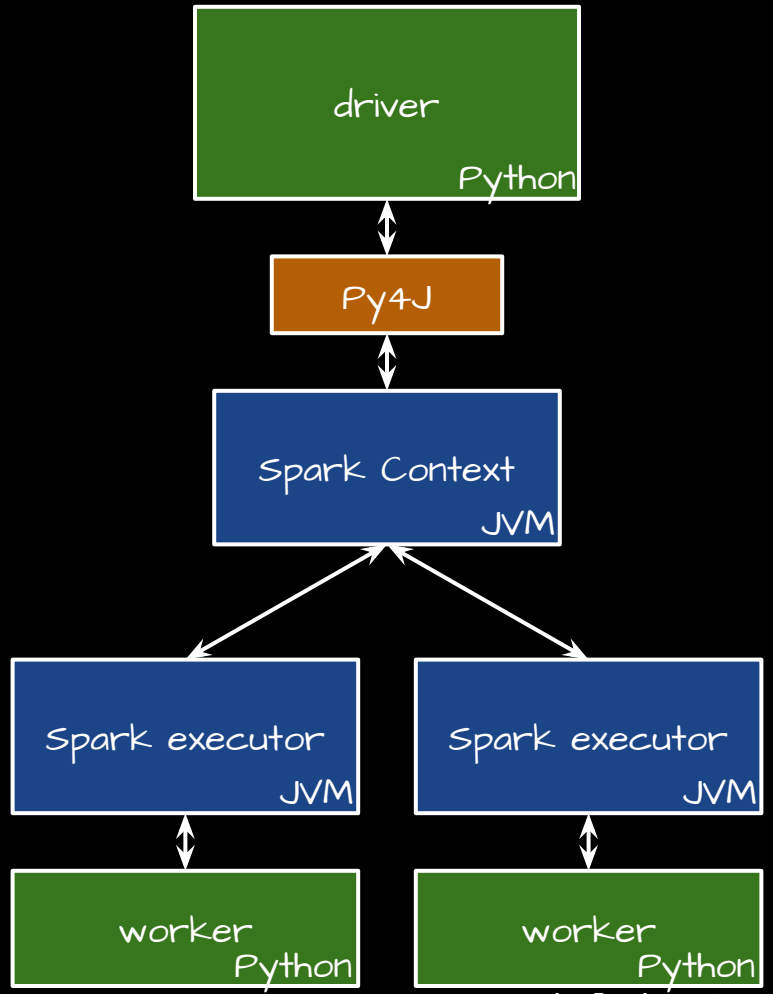
Spark APIs

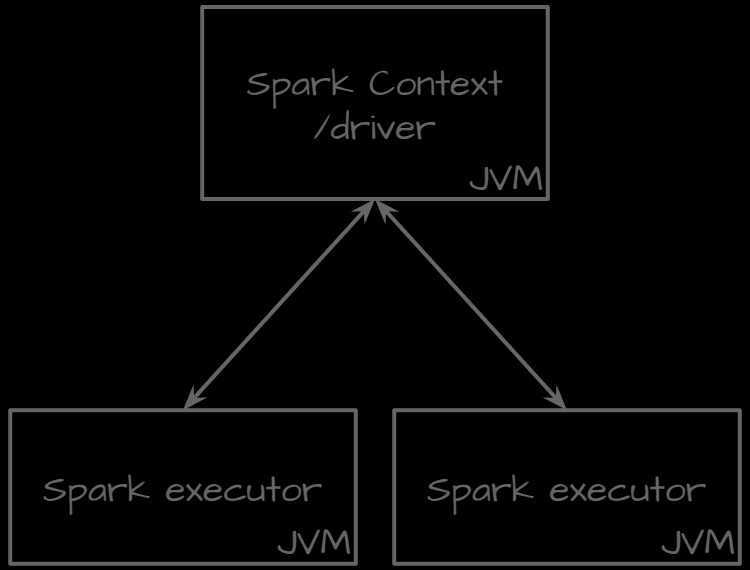
- RDDs (Resilient Distributed Datasets)
- DataFrames
- Datasets (statically typed)
  - only in Scala
- Spark SQL
- UDFs (User-Defined Functions)
  - Scala: within JVM processes
  - PySpark...

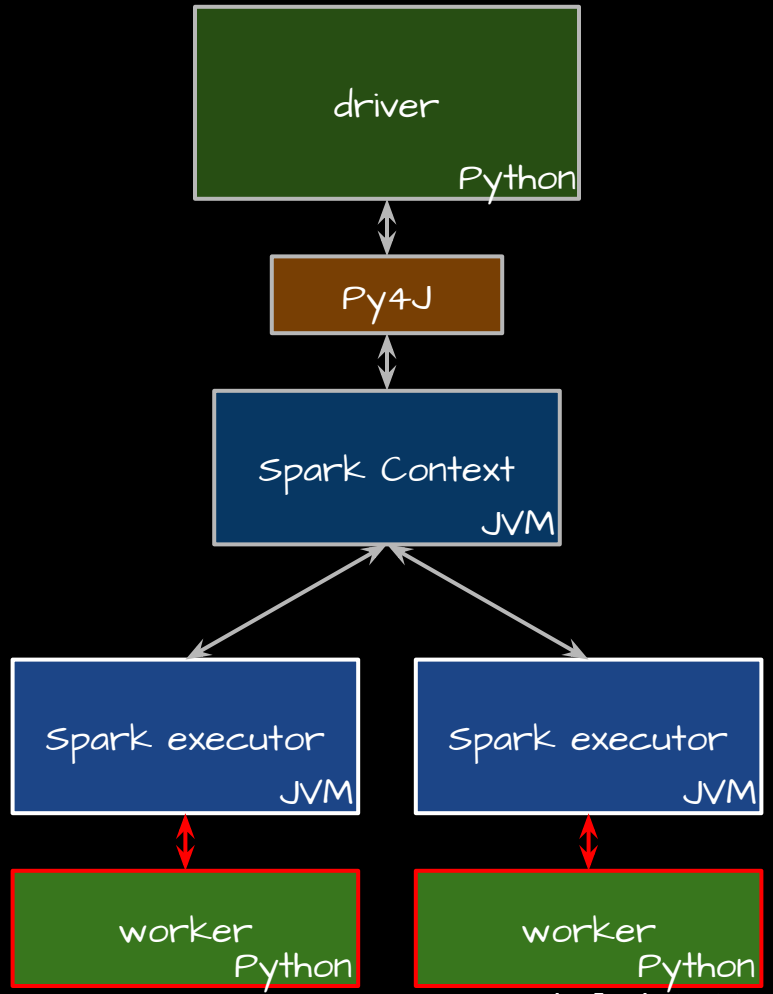

Spark Context
/driver
JVM
Spark executor
JVM
Spark executor
JVM


driver
Python
Py4J
Spark Context
JVM
Spark executor
JVM
Spark executor
JVM
worker
Python
worker
Python


Spark Context
/driver
JVM
Spark executor
JVM
Spark executor
JVM


driver
Python
Py4J
Spark Context
JVM
Spark executor
JVM
Spark executor
JVM
worker
Python
worker
Python


Spark Context
/driver
JVM
Spark executor
JVM
Spark executor
JVM


driver
Python
Py4J
Spark Context
JVM
Spark executor
JVM
Spark executor
JVM
worker
Python
worker
Python

Ecosystem

Ecosystem

- Spark (Scala):
  - Structured Streaming
  - MLlib
- PySpark:
  - Structured Streaming
  - MLlib

Ecosystem

- Spark (Scala):
  - Structured Streaming
  - MLlib
  - GraphX
- PySpark:
  - Structured Streaming
  - MLlib
  - ~~GraphX~~

Ecosystem

- Spark (Scala):
  - Structured Streaming
  - MLlib
  - GraphX
  - **Scala first**
- PySpark:
  - Structured Streaming
  - MLlib
  - ~~GraphX~~

Ecosystem

- Spark (Scala):
  - Structured Streaming
  - MLlib
  - GraphX
  - **Scala first**
- PySpark:
  - Structured Streaming
  - MLlib
  - ~~GraphX~~ (but graphframes)

Ecosystem

- Spark (Scala):
  - Structured Streaming
  - MLlib
  - GraphX
  - **Scala first**
- PySpark:
  - Structured Streaming
  - MLlib
  - ~~GraphX~~ (but graphframes)
- NumPy, Pandas, Koalas...

Ecosystem

- Spark (Scala):
  - Structured Streaming
  - MLlib
  - GraphX
  - **Scala first**
- PySpark:
  - Structured Streaming
  - MLlib
  - ~~GraphX~~ (but graphframes)
- NumPy, Pandas, Koalas...
- TensorFlow, ...
- ...

Ecosystem

- Spark (Scala):
  - Structured Streaming
  - MLlib
  - GraphX
  - **Scala first**
- Breeze, Spire, ...
- DeepLearning.scala, BigDL, H2O, ...
- ...

- PySpark:
  - Structured Streaming
  - MLlib
  - ~~GraphX~~ (but graphframes)
- NumPy, Pandas, Koalas...
- TensorFlow, ...
- ...

Ecosystem

- benefits in a given project

Ecosystem

- benefits *in a given project*
  - adjacent code

Ecosystem

- benefits *in a given project*
  - adjacent code
  - modularization

Ecosystem

- benefits *in a given project*
  - adjacent code
  - modularization
- team's competences and aspirations

Deployment

Deployment

- Dependencies in a JAR

Deployment

- Dependencies in a JAR
- Dependencies:
  - in a ZIP file
  - in an archived virtualenv
  - or installed on a worker node

Deployment

- Dependencies in a JAR

```
spark-submit \
  --master spark://host:port
  --class MyClass \
  path/to.jar
```

- Dependencies:
  - in a ZIP file
  - in an archived virtualenv
  - or installed on a worker node

```
spark-submit \
  --master spark://host:port
  --py-files ... \
  --archives ... \
  path/to.py
```

Maintenance

Maintenance

- team's competences (and aspirations)

Maintenance

- team's competences (and aspirations)
- collaboration with others

Maintenance

- team's competences (and aspirations)
- collaboration with others
- typing (and other language differences)

Maintenance

- team's competences (and aspirations)
- collaboration with others
- typing (and other language differences)
- compilation time
- tools:
  - testing
  - static analysis
  - profiling
  - ...

Popularity

Popularity

- JetBrains and StackOverflow developer surveys

Popularity

- JetBrains and StackOverflow developer surveys
- StackOverflow questions

[scala] [apache-spark]:

[python] [apache-spark]:
[pyspark]:

Popularity

- JetBrains and StackOverflow developer surveys
- StackOverflow questions

  `[scala] [apache-spark]`: **~23.6k**

  `[python] [apache-spark]`: **~9.6k**
  `[pyspark]`: **~35k**

Performance

Performance

- select and save

Scala:

PySpark:

Performance

- select and save

  Scala: **~1.7 min**

  PySpark: **~1.7 min**

Performance

- select and save

  Scala: **~1.7 min**

  PySpark: **~1.7 min**

- select, add a column in a UDF, save

  Scala:

  PySpark:

Performance

- select and save

  Scala: **~1.7 min** PySpark: **~1.7 min**

- select, add a column in a UDF, save

  Scala: **~2.5 min** PySpark:

Performance

- select and save

  Scala: **~1.7 min** PySpark: **~1.7 min**

- select, add a column in a UDF, save

  Scala: **~2.5 min** PySpark: **~4,1 min**

Performance

- select and save

  Scala: **~1.7 min** PySpark: **~1.7 min**

- select, add a column in a UDF, save

  Scala: **~2.5 min** PySpark: **~4,1 min**

- **be wary of such benchmarks :)**
  - how well do they represent your use-case?

Performance

- select and save

  Scala: **~1.7 min** PySpark: **~1.7 min**

- select, add a column in a UDF, save

  Scala: **~2.5 min** PySpark: **~4,1 min**

- be wary of such benchmarks :)
  - how well do they represent your use-case?
- your performance requirements
  - are they real - or only ambitional?

So... how to choose?

How to choose?

1. functional requirements
    - more focus on tools than the language

How to choose?

1. functional requirements
   - more focus on tools than the language
2. performance requirements
   - are these real - or only ambitional?

How to choose?

1. functional requirements
   - more focus on tools than the language
2. performance requirements
   - are these real - or only ambitional?
3. competences (and aspirations!) of maintainers
   - present and future ones

How to choose?

1. functional requirements
   - more focus on tools than the language
2. performance requirements
   - are these real - or only ambitional?
3. competences (and aspirations!) of maintainers
   - present and future ones
4. other non-functional requirements
   - testing, typing, compilation time etc.

Thank you! :)
Questions?

kolodziejj.info/talks/pyspark-scala

allegro Tech