COMP9313: Big Data Management

Lecturer: Xubo Wang

Course web site: http://www.cse.unsw.edu.au/~cs9313/

Chapter 5.2: Spark IV

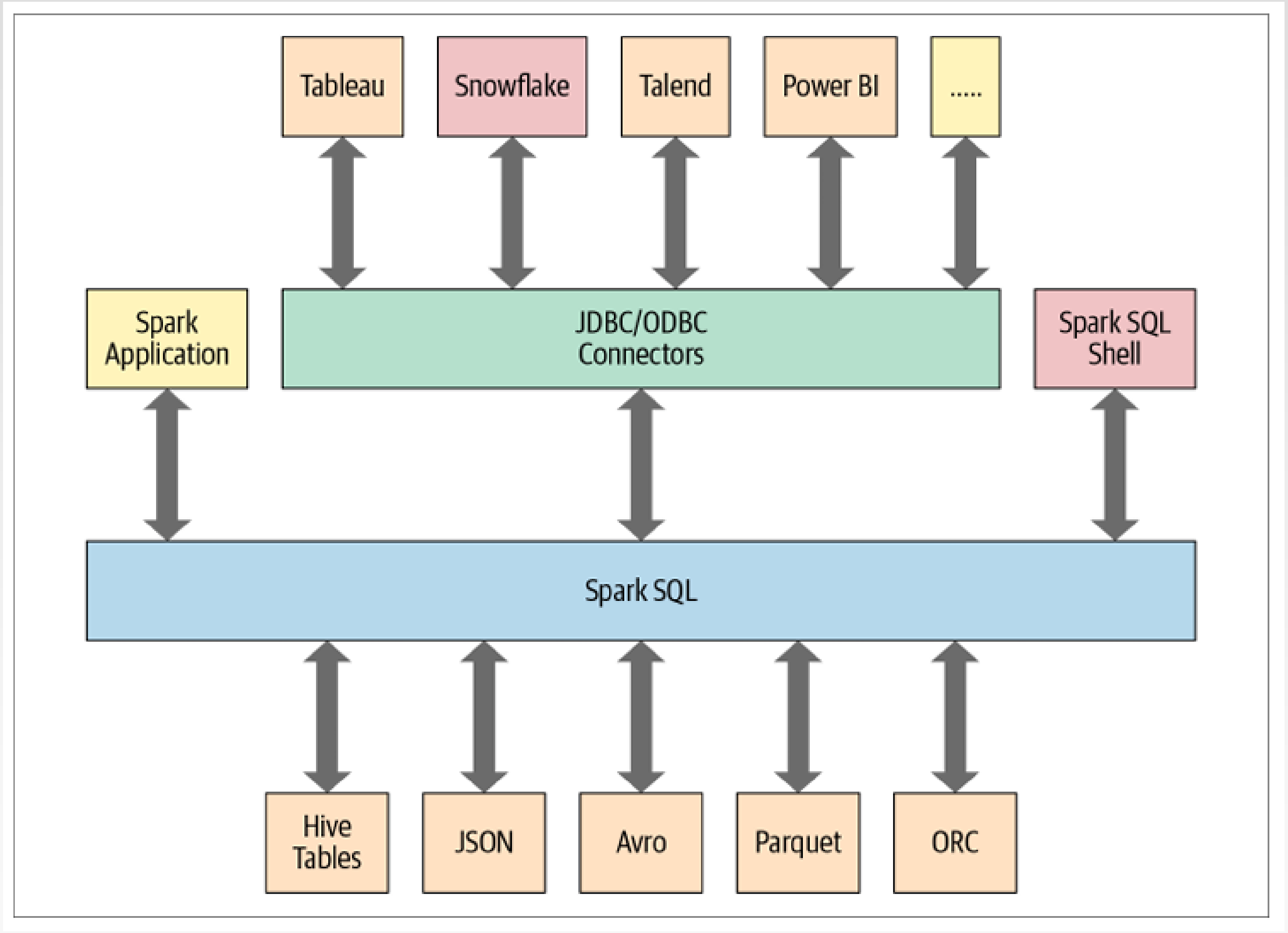
Part 1: Spark SQL

Spark SQL Overview

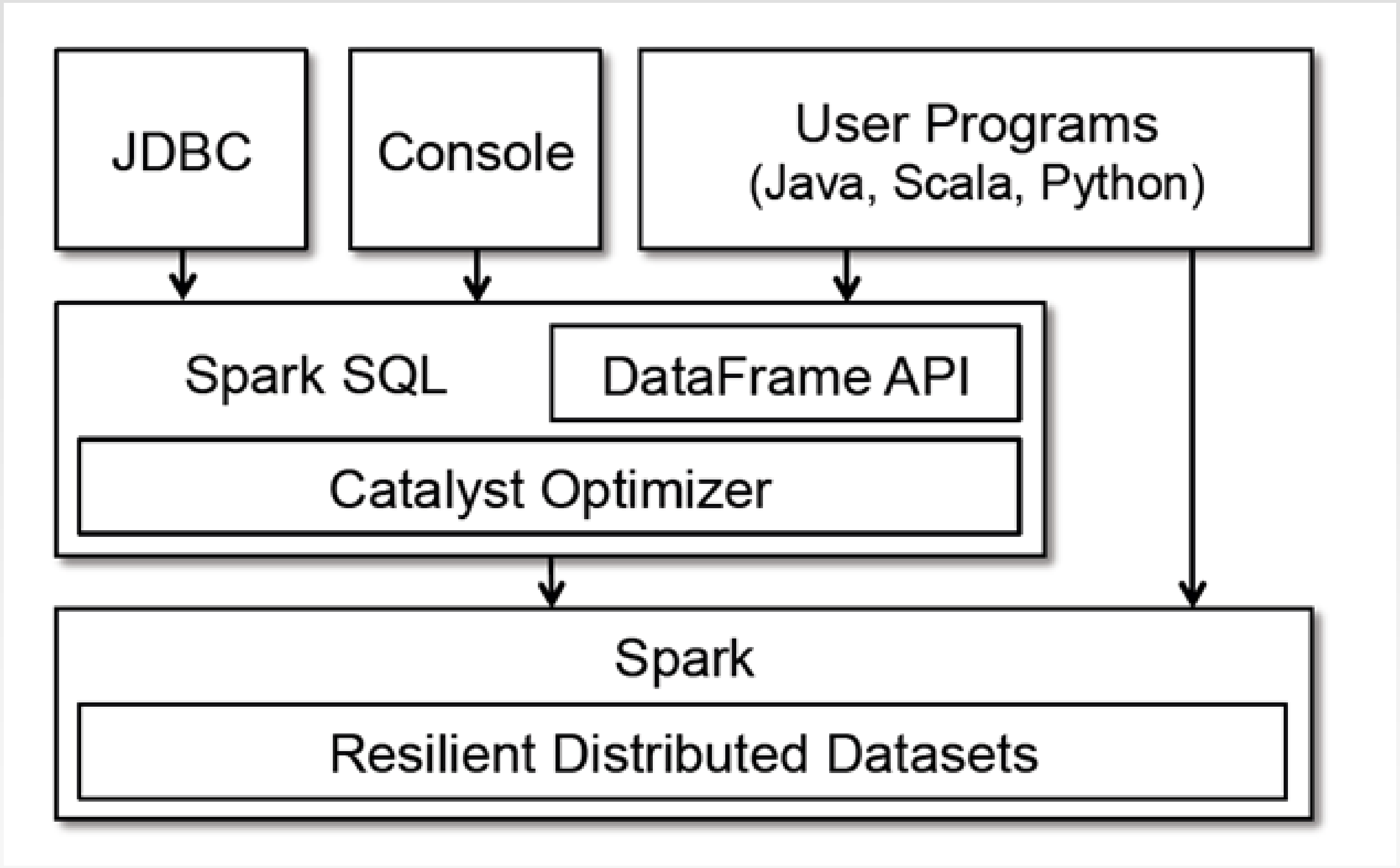
- Part of the core distribution since Spark 1.0, Transform RDDs using SQL in early versions (April 2014)
- Tightly integrated way to work with structured data (tables with rows/columns)
- Data source integration: Hive, Parquet, JSON, and more

- Spark SQL is **not** about SQL.
  - Aims to Create and Run Spark Programs Faster

Spark SQL connectors and data sources



Spark Programming Interface


JDBC
Console
User Programs (Java, Scala, Python)
Spark SQL
DataFrame API
Catalyst Optimizer
Spark
Resilient Distributed Datasets

Starting Point: SparkSession

- The entry point into all functionality in Spark is the SparkSession class
  - Python

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.master("local").appName("Spark SQL
basic example").getOrCreate()
```

  - SparkSession since Spark 2.0 provides built-in support for Hive features including the ability to write queries using HiveQL, access to Hive UDFs, and the ability to read data from Hive tables

Creating DataFrames from JSON

- With a SparkSession, applications can create DataFrames based on the content of a JSON file:

```
df = spark.read.json("examples/src/main/resources/people.json")

// Displays the content of the DataFrame to stdout

df.show()
// +----+-------+
// | age|   name|
// +----+-------+
// |null|Michael|
// |  30|   Andy|
// |  19| Justin|
// +----+-------+
```

Running SQL Queries Programmatically

- The *sql* function on a SparkSession enables applications to run SQL queries programmatically and returns the result as a DataFrame.

```
// Register the DataFrame as a SQL temporary view
df.createOrReplaceTempView("people")

sqlDF = spark.sql("SELECT * FROM people")
sqlDF.show()
// +----+-------+
// | age|   name|
// +----+-------+
// |null|Michael|
// |  30|   Andy|
// |  19| Justin|
// +----+-------+
```

Global Temporary View

- Temporary views in Spark SQL are session-scoped and will disappear if the *session* that creates it terminates

- Global temporary view: a temporary view that is shared among all sessions and keep alive until the Spark *application* terminates

- Global temporary view is tied to a system preserved database global_temp, and we must use the qualified name to refer it, e.g. SELECT * FROM global_temp.view1

Global Temporary View Example

```
// Register the DataFrame as a global temporary view
df.createGlobalTempView("people")

// Global temporary view is tied to a system preserved database `global_temp`
spark.sql("SELECT * FROM global_temp.people").show()
// +----+-------+
// | age|   name|
// +----+-------+
// |null|Michael|
// |  30|   Andy|
// |  19| Justin|
// +----+-------+

// Global temporary view is cross-session
spark.newSession().sql("SELECT * FROM global_temp.people").show()
// +----+-------+
// | age|   name|
// +----+-------+
// |null|Michael|
// |  30|   Andy|
// |  19| Justin|
// +----+-------+
```

Find full example code at
https://github.com/apache/spark/blob/master/examples/src/main/scala/org/apache/spark/examples/sql/SparkSQLExample.scala

Spark SQL Built-in Functions

- Spark SQL provides several built-in standard functions org.apache.spark.sql.functions to work with DataFrame/Dataset and SQL queries. All these Spark SQL Functions return org.apache.spark.sql.Column type.
  - String Functions
  - Date & Time Functions
  - Collection Functions
  - Math Functions
  - Aggregate Functions
  - Window Functions
  - You can check the examples of these functions at: https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/functions.html
  - In order to use these SQL Standard Functions, you need to import below packing into your application.

```
from pyspark.sql.functions import *
```

WordCount using Spark SQL

- Create temporary view

```
fileDF = spark.read.text("file:///home/comp9313/inputText")

from pyspark.sql.functions import *
wordsDF = fileDF.select(explode(split(fileDF.value, ' ')).alias("word"))

wordsDF.createOrReplaceTempView("wc")
```

- Count words using SQL

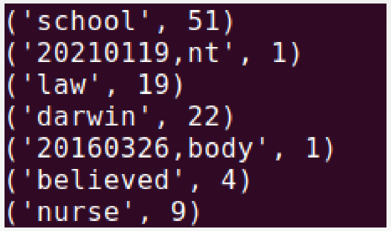
```
wc_df = spark.sql("SELECT word, COUNT(*) as count FROM wc GROUP BY
word ORDER BY count DESC")
wc_df.show()
```

PySpark Standalone Code (RDD)

```
from pyspark import SparkContext, SparkConf
import sys

class WordCount:
    def run(self, inputPath, outputPath):
        conf = SparkConf().setAppName("word count").setMaster("local[3]")
        sc = SparkContext(conf=conf)

        fileRDD = sc.textFile(inputPath)
        wordsRDD = fileRDD.flatMap(lambda line: line.lower().split())
        pairsRDD = wordsRDD.map(lambda word: (word, 1))
        countRDD = pairsRDD.reduceByKey(lambda a, b: a+b)

        countRDD.saveAsTextFile(outputPath)
        sc.stop()

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Wrong inputs")
        sys.exit(-1)
    WordCount().run(sys.argv[1], sys.argv[2])
```

PySpark Standalone Code (RDD)

- The input and output could be on HDFS or on your local file system
  - We receive them from the command line
    - spark-submit wordcount.py file:///home/comp9313/pg100.txt file:///home/comp9313/outuput
    - spark-submit wordcount.py hdfs:///localhost:9000/user/comp9313/input hdfs:///localhost:9000/user/comp9313/output
- We can use the RDD textFile() operation to read the data into an RDD
- We can use RDD saveAsTextFile() operation to write the data to disk
  - The result contains parentheses by default



```
('school', 51)
('20210119,nt', 1)
('law', 19)
('darwin', 22)
('20160326,body', 1)
('believed', 4)
('nurse', 9)
```

  - You can format the output and then save to file
- Remember to release the resources by sc.stop() finally

PySpark Standalone Code (DataFrame)

```
from pyspark.sql.session import SparkSession
from pyspark.sql.functions import *
import sys

class WordCount:
    def run(self, inputPath, outputPath):
        spark = SparkSession.builder.master("local").appName("word
count").getOrCreate()
        fileDF = spark.read.text(inputPath)
        wordsDF = fileDF.selectExpr("explode(split(value, ' ')) as
word").withColumn("word", lower(col("word")))

        countDF = wordsDF.groupBy("word").count()
        countDF.write.format("csv").save(outputPath)
        #resDF = countDF.select(concat(col("word"), lit(","), col("count")))
        #resDF.write.text(outputPath)
        spark.stop()

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Wrong inputs")
        sys.exit(-1)
    WordCount().run(sys.argv[1], sys.argv[2])
```

PySpark Standalone Code (DataFrame)

- We first need to create a SparkSession object spark
  - You can utilize the SparkSession.builder to configure your task: spark = SparkSession.builder.master("local").appName("word count").getOrCreate()

- The input and output could be on HDFS or on your local file system

- Remember to release the resources by spark.stop() finally

PySpark Standalone Code (DataFrame)

- We can use the pyspark.sql.DataFrameReader.text() operation to read the text data into a DataFrame
  - pyspark.sql.DataFrameReader.csv()
  - pyspark.sql.DataFrameReader.json()
  - https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/io.html

- We can use pyspark.sql.DataFrameWriter.text() to write a DataFrame **with a single column of string type** to a file
- We can use pyspark.sql.DataFrameWriter.csv() or pyspark.sql.DataFrameWriter.format("csv").save() to store the data as a csv file
  - You can also use other formats such as json
  - You can also use pyspark.sql.DataFrameWriter.format("text").save(), but it also requires a DataFrame **with a single column of string type**

select() vs selectExpr()

- Both functions are used to select columns from a DataFrame.
  - select(): You can select the single or multiple columns of the DataFrame by passing the column names to the select() function

```
df.select("firstname","lastname").show()
df.select(df.firstname,df.lastname).show()
df.select(df["firstname"],df["lastname"]).show()

#By using col() function
from pyspark.sql.functions import col
df.select(col("firstname"),col("lastname")).show()

#By using column index
df.select(df.columns[2:4]).show(3)
```

- selectExpr() projects a set of SQL expressions and returns a new DataFrame. The difference is that it takes a set of SQL expressions in a string to execute.

select() vs selectExpr()

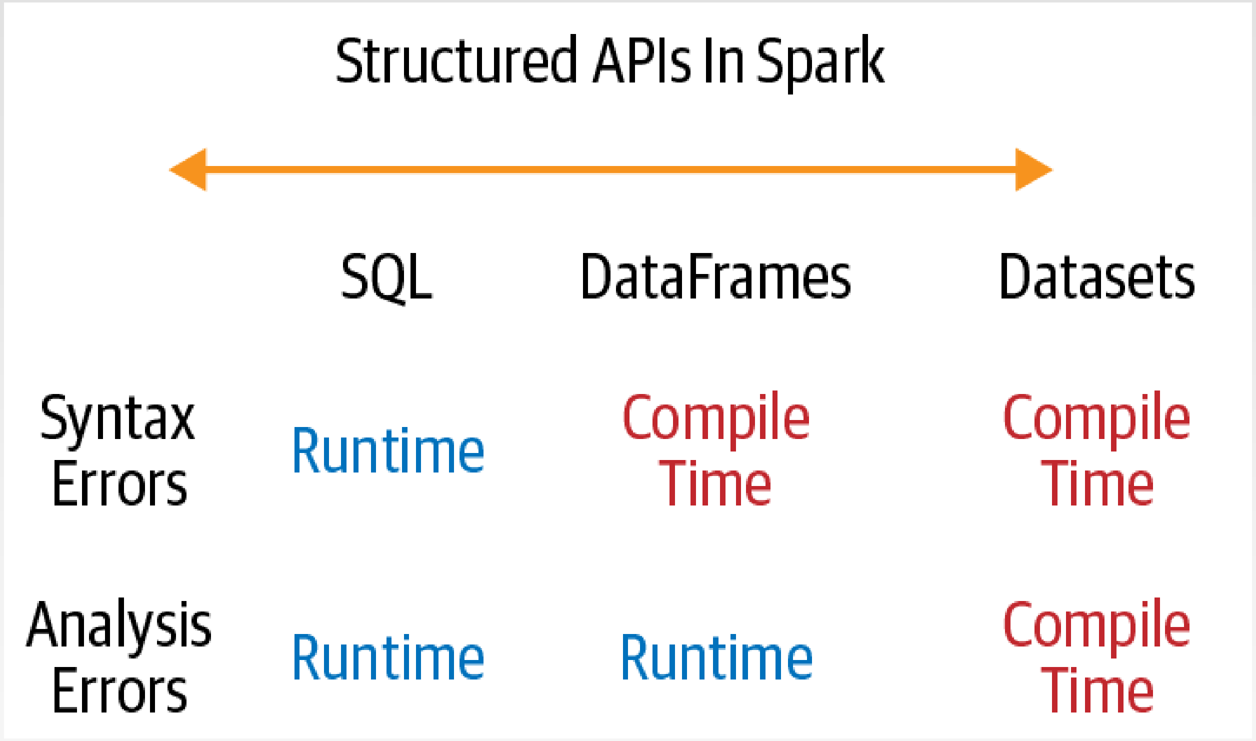
- Both functions are used to select columns from a DataFrame.
  - select(): You can select the single or multiple columns of the DataFrame by passing the column names to the select() function

```
df.select("firstname","lastname").show()
df.select(df.firstname,df.lastname).show()
df.select(df["firstname"],df["lastname"]).show()

#By using col() function
from pyspark.sql.functions import col
df.select(col("firstname"),col("lastname")).show()

#By using column index
df.select(df.columns[2:4]).show(3)
```

-
  ```
  >>>df = spark.createDataFrame([
      (2, "Alice"), (5, "Bob")], schema=["age", "name"])
  >>>df.selectExpr("age * 2", "abs(age)").show()
  +---------+--------+
  |(age * 2)|abs(age)|
  +---------+--------+
  |        4|       2|
  |       10|       5|
  +---------+--------+
  ```

Error Detection of Structured APIs

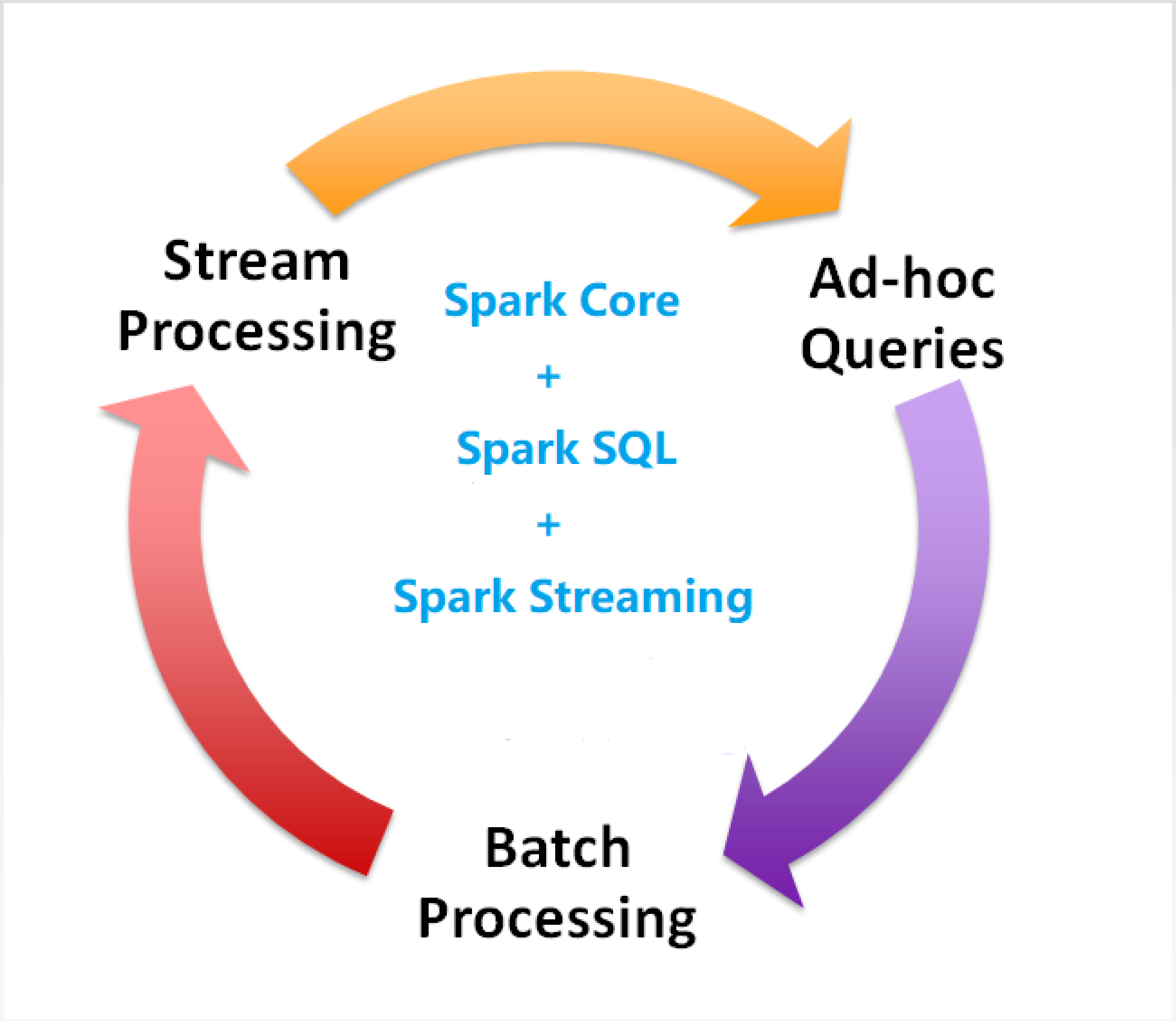
- If you want errors caught during compilation rather than at runtime, choose the appropriate API



Structured APIs In Spark

	SQL	DataFrames	Datasets
Syntax Errors	Runtime	Compile Time	Compile Time
Analysis Errors	Runtime	Runtime	Compile Time

Vision - one stack to rule them all



Part 2: Spark Programming Practice

Practice

- Problem 1: Given the data in format of key-value pairs <Int, Int>, find the maximum value for each key across all values associated with that key.

```
pairs = sc.parallelize([(1, 2), (3, 4),… …])

resMax = pairs.reduceByKey(lambda x,y: x if x>y else y)

resMax.foreach(lambda x: print(x[0], x[1]))
```

Practice

❖ Problem 2: Given a pair RDD of type [(String, Int)], compute the per-key average

key	value
panda	0
pink	3
pirate	3
panda	1
pink	4

key	value
panda	0.5
pink	3.5
pirate	3

Practice

- Problem 2: Given a pair RDD of type [(String, Int)], compute the per-key average

key	value
panc	
pink	
pirat	
panda	1
pink	4

```
pair.mapValues(lambda x: (x,1))
      .reduceByKey(lambda x,y: (x[0]+y[0], x[1]+y[1]))
      .mapValues(lambda x: x[0]/x[1])
```

■

key	value
pirate	3

mapValues

mapValues

```
df = spark.createDataFrame(list, schema = "key string, value int")
df.groupBy("key").avg("value").show()
```

panda	(0, 1)
pink	(3, 1)
pirate	(3, 1)
panda	(1, 1)
pink	(4, 1)

reduceByKey $\longrightarrow$

key	value
panda	(1, 2)
pink	(7, 2)
pirate	(3, 1)

Practice (RDD)

- Problem 3: Given a collection of documents, compute the average length of words starting with each letter.

```
textFile = sc.textFile(inputFile)
words = textFile.flatMap(lambda line: line.split(" ")).map(lambda x: x.lower())

counts = words.filter(lambda x: len(x) >=1 and x[0]<='z' and x[0]>='a').map(lambda x:
(x[0], (len(x), 1)))

avgLen = counts.reduceByKey(lambda a, b: (a[0]+b[0], a[1]+b[1])).map(lambda x:
(x[0], x[1][0]/x[1][1]))

avgLen.foreach(lambda x: print(x[0], x[1]))
```

Practice (DataFrame)

- Problem 3: Given a collection of documents, compute the average length of words starting with each letter.

```
fileDF = spark.read.text(inputFile)

wordsDF = fileDF.select(explode(split(fileDF.value, " ")).alias("word")).withColumn("word",
lower(col("word")))

wordsDF = wordsDF.filter(length(col("word")) >=1).filter((col("word").substr(0,1)<= 'z') &
(col("word").substr(0,1)>='a'))

pairDF = wordsDF.select(wordsDF.word.substr(0, 1),length(wordsDF.word)).toDF("letter",
"length")

countsDF = pairDF.groupBy("letter").agg(count("letter").alias("totalCount"),
sum("length").alias("totalLength"))

avgDF = countsDF.withColumn("ratio",
countsDF.totalLength/countsDF.totalCount).select("letter", "ratio").orderBy("letter")

avgDF.write.format("csv").save(outputFolder)
```

Practice (Spark SQL)

- Problem 3: Given a collection of documents, compute the average length of words starting with each letter.

```
fileDF = spark.read.text(inputFile)

fileDF.selectExpr("explode(split(value, ' ')) as word").createOrReplaceTempView("words")

spark.sql("select * from words where length(word)>=1 and substr(word, 0, 1)>='a' and
substr(word, 0, 1)<='z' ").createOrReplaceTempView("filteredwords")

spark.sql("select substr(word, 0, 1) as letter, length(word) as length from
filteredwords").createOrReplaceTempView("pair")

spark.sql("select letter, sum(length) as totalLength, count(*) as totalCount from pair group by
letter").createOrReplaceTempView("count")

avgDF = spark.sql("select letter, totalLength/totalCount as ratio from count order by letter")

avgDF.write.format("csv").save(outputFolder)
```

Pass a Function to RDD Operations

- For a given text file, find the longest word from each line.

```
from pyspark import SparkContext, SparkConf
import sys

def findlongest(termList):
    maxTerm = ""
    for t in termList:
        if len(t) > len(maxTerm):
            maxTerm = t
    return maxTerm

class WordCount:
    def run(self, inputPath, outputPath):
        sc = SparkContext("local", "longest")
        fileRDD = sc.textFile(inputPath)
        wordsRDD = fileRDD.map(lambda line: line.lower().split())
        longestRDD = wordsRDD.map(lambda termList: findlongest(termList))
        longestRDD.saveAsTextFile(outputPath)
        sc.stop()
if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Wrong inputs")
        sys.exit(-1)
    WordCount().run(sys.argv[1], sys.argv[2])
```

Define Your Own Function with UDF

- PySpark UDF (a.k.a User Defined Function) is the most useful feature of Spark SQL & DataFrame that is used to extend the PySpark build in capabilities.

- PySpark UDF's are similar to UDF on traditional databases. In PySpark, you create a function in a Python syntax and wrap it with PySpark SQL udf() or register it as udf and use it on DataFrame and SQL respectively.

- https://sparkbyexamples.com/pyspark/pyspark-udf-user-defined-function
- https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.functions.udf.html#pyspark.sql.functions.udf

Pass a Function to DataFrame Operations

- For a given text file, find the longest word from each line.

```
from pyspark.sql.session import SparkSession
from pyspark.sql.functions import *
import sys

def findlongest(termList):
    maxTerm = ""
    for t in termList:
        if len(t) > len(maxTerm):
            maxTerm = t
    return maxTerm

class WordCount:
    def run(self, inputPath, outputPath):
        spark = SparkSession.builder.master("local").appName(“longest").getOrCreate()
        fileDF = spark.read.text(inputPath)
        wordsDF = fileDF.select(split(fileDF.value, ' ').alias('termlist'))
        longestUDF = udf(lambda termList: findlongest(termList))
        resDF = wordsDF.withColumn('longest', longestUDF('termlist')).select('longest')
        resDF.write.text(outputPath)
        spark.stop()

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Wrong inputs")
        sys.exit(-1)
    WordCount().run(sys.argv[1], sys.argv[2])
```

References

- http://spark.apache.org/docs/latest/index.html
- Spark SQL guide: http://spark.apache.org/docs/latest/sql-programming-guide.html
- https://spark.apache.org/docs/3.3.0/api/scala/org/apache/spark/index.html
- https://spark.apache.org/docs/3.3.0/api/python/getting_started/index.html
- Learning Spark. 2nd edition
- Spark SQL Functions: https://sparkbyexamples.com/spark/spark-sql-functions/

End of Chapter 5.2