



Spark Core - RDD

Spark SQL - DataSet

arnaud.nauwynck@gmail.com

This document:

[https://github.com/Arnaud-Nauwynck/presentations/
blob/main/pres-bigdata/8-spark-core-rdd-sql-dataset.pdf](https://github.com/Arnaud-Nauwynck/presentations/blob/main/pres-bigdata/8-spark-core-rdd-sql-dataset.pdf)

Spark (Recent) History & Ancestors

MapReduce @ 

2002 @Google 2004 Google
Paper published

2014 Google
No more used of MapReduce



2006 @Yahoo
Hadoop implementation
2008 Apache Open-Source

2012 Yarn (v2)

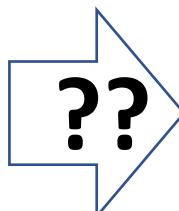
2021 MapReduce bashing
... HDFS & Hadoop also
HortonWork bought by Cloudera
HDInsight @Azure...very bad choice
.. To be abandonned



1995 Message Passing Interface



2010 Spark paper 2013 Apache top-level



2015 Kubernetes 2020
 Spark on K8s

Simple => Many Specific Systems => Unified



« Simple » ecosystem
(verbose inefficient &
complex java code)

« **Bazard** » ecosystem
(Too MANY TOO SPECIFIC
redundant, complexes)

“Unified” ecosystem
Simple
+ extensible modules

Spark = « Unified Engine »

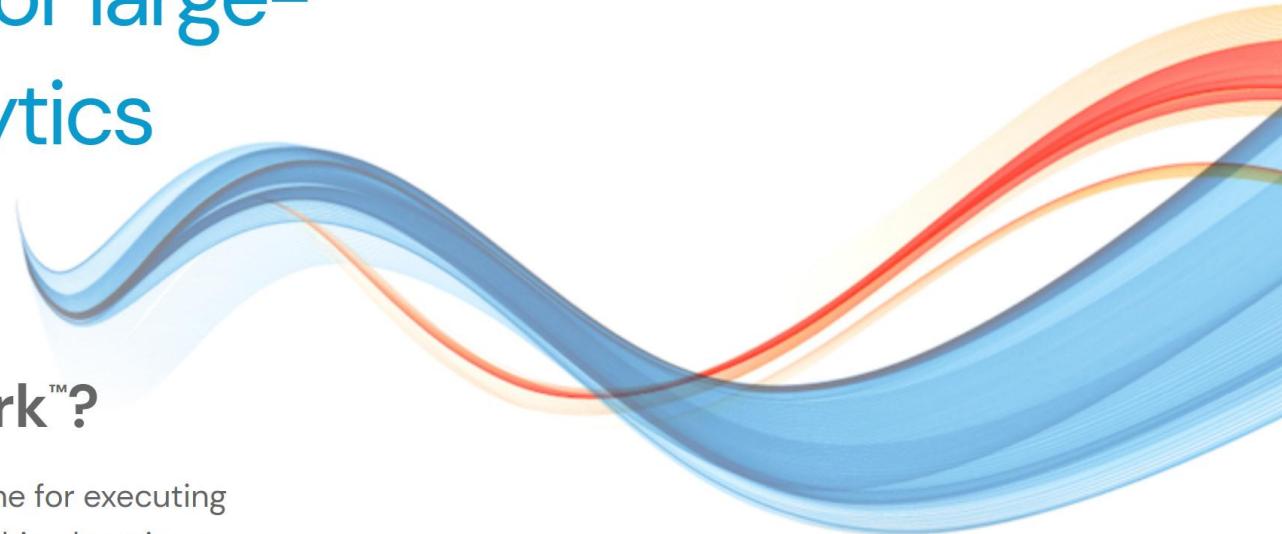


Unified engine for large-scale data analytics

[GET STARTED](#)

What is Apache Spark™?

Apache Spark™ is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters.



Multi Purposes – Multi Languages

**Simple.
Fast.
Scalable.
Unified.**

Key features



Batch/streaming data

Unify the processing of your data in batches and real-time streaming, using your preferred language: Python, SQL, Scala, Java or R.



SQL analytics

Execute fast, distributed ANSI SQL queries for dashboarding and ad-hoc reporting. Runs faster than most data warehouses.



Data science at scale

Perform Exploratory Data Analysis (EDA) on petabyte-scale data without having to resort to downsampling



Machine learning

Train machine learning algorithms on a laptop and use the same code to scale to fault-tolerant clusters of thousands of machines.

Python

SQL

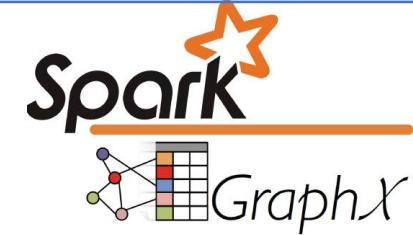
Scala

Java

R

Spark-Core + ...

Structured
Data



Modules



DataSource Connectors
(Hadoop API)



Cluster Manager



Languages Support



Getting Started

1 / Download

 [Download](#) [Libraries ▾](#) [Documentation ▾](#) [Examples](#) [Community ▾](#) [Developers ▾](#)

Download Apache Spark™

1. Choose a Spark release: **3.2.0 (Oct 13 2021)**
 2. Choose a package type: **Pre-built for Apache Hadoop 3.3 and later**
 3. Download Spark: [spark-3.2.0-bin-hadoop3.2.tgz](#)
 4. Verify this release using the 3.2.0 [signatures](#), [checksums](#) and [project release KEYS](#).

2/ Unzip + add to PATH

3/ launch

C:> bin\spark-shell

(or spark-submit, or spark-sql)

```
C:\Users\arnaud>spark-shell
Welcome to

$$\begin{array}{c} / \quad \backslash \quad / \quad \backslash \\ \backslash \quad \backslash \quad - \quad \backslash \quad / \quad / \quad / \quad \backslash \quad / \quad \backslash \\ / \quad \backslash \quad . \quad / \quad \backslash \quad , \quad / \quad / \quad / \quad \backslash \quad \backslash \quad / \quad / \end{array}$$
 version 3.5.0
```

```
Using Scala version 2.13.8 (OpenJDK 64-Bit Server VM, Java 20.0.1)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context Web UI available at http://DesktopArnaud:4040
Spark context available as 'sc' (master = local[*], app id = local-1734184851521).
Spark session available as 'spark'.

scala>
```

```
          scala>
scala> println(« Hello spark »);
```

launch spark-shell from terminal

C:\Users\arnaud>spark-shell

Welcome to

```
Using Scala version 2.13.8 (OpenJDK 64-Bit Server VM, Java 20.0.1)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context Web UI available at http://DesktopArnaud:4040
Spark context available as 'sc' (master = local[*], app id = local-1734184851521).
Spark session available as 'spark'.
```

```
scala> println("Hello spark")
```

Hello spark

scala>

inside spark
type scala code

Spark-shell> SCALA code

:paste

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

for(i <- 0 to 5) {
    println(s"Hello ${i}")
}
```

Ctrl-D

```
// Exiting paste mode, now interpreting.

Hello 0
Hello 1
Hello 2
Hello 3
Hello 4
Hello 5

scala>
```

spark-shell> help

```
scala> :help
```

All commands can be abbreviated, e.g., :he instead of :help.

:help [command]	print this summary or command-specific help
:completions <string>	output completions for the given string
:imports [name name ...]	show import history, identifying sources of names
:implicits [-v]	show the implicits in scope
:javap <path class>	disassemble a file or class name
:line <id> <line>	place line(s) at the end of history
:load <path>	interpret lines in a file
:paste [-raw] [path]	enter paste mode or paste a file
:power	enable power user mode
:quit	exit the REPL
:replay [options]	reset the REPL and replay all previous commands
:require <path>	add a jar to the classpath
:reset [options]	reset the REPL to its initial state, forgetting all session entries
:save <path>	save replayable session to a file
:sh <command line>	run a shell command (result is implicitly => List[String])
:settings <options>	update compiler options, if possible; see reset
:silent	disable/enable automatic printing of results
:type [-v] <expr>	display the type of an expression without evaluating it
:kind [-v] <type>	display the kind of a type. see also :help kind
:warnings	show the suppressed warnings from the most recent line which had any

Useful default key bindings:

TAB code completion

CTRL-ALT-T show type at cursor, hit again to show code with types/implicits inferred.

```
scala>
```

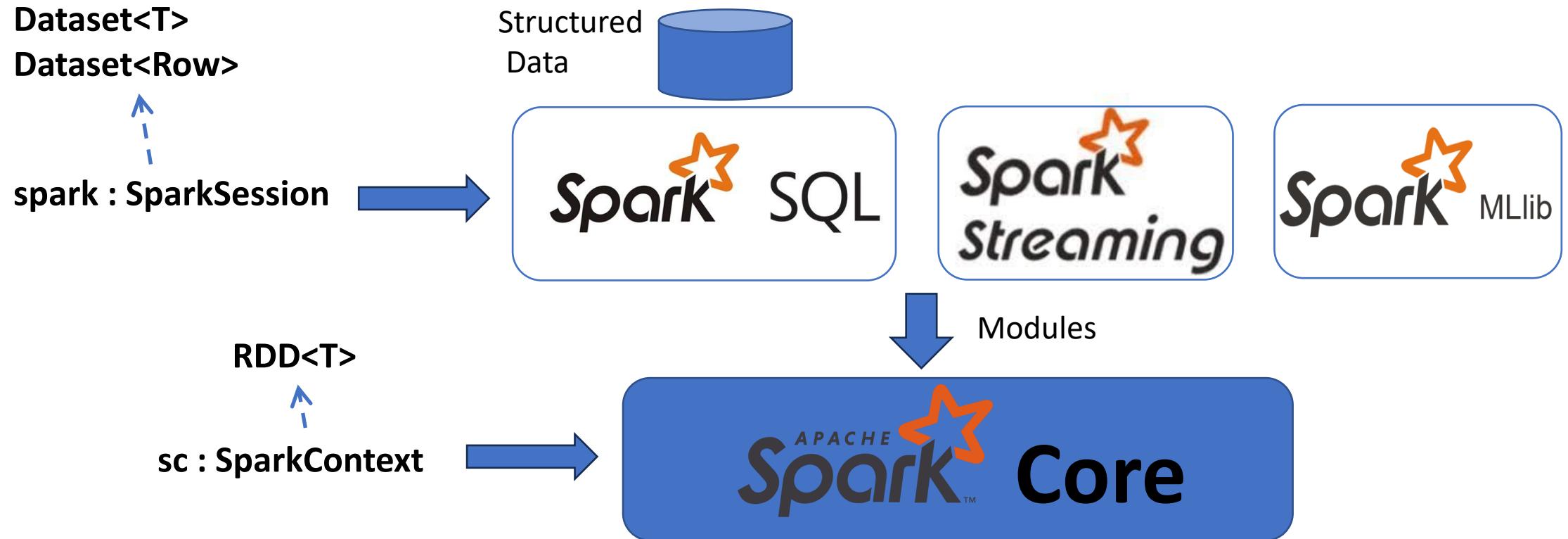
2 built-in Variables Available : 'sc', 'spark'

Spark context Web UI available at <http://DesktopArnáud:4040>

Spark context available as 'sc' (master = local[*], app id = local-1734183373641).

Spark session available as 'spark'

sc: SparkContext (in spark-core)
spark : SparkSession (in spark-sql)



```
scala> sc
val res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@5eb39c06

scala> spark
val res1: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@4cecbf3e
```

« Words Count » ... Hello world of BigData

loremIpsum.txt

 Lorem Ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
 Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
 Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
 Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

```
scala> val ds=spark.read.textFile("c:/data/loremIpsum.txt")
ds: org.apache.spark.sql.Dataset[String] = [value: string]
```

```
scala> ds.count // count lines
res1: Long = 4
```

Dataset.show

default show(20 /*line*/, true /*truncate*/)

```
scala> val ds = spark.read.textFile("c:/data/loremIpsum.txt")
val ds: org.apache.spark.sql.Dataset[String] = [value: string]
```

```
scala> ds.show()
+-----+
|      value|
+-----+
|Lorem Ipsum dolor...|
|Ut enim ad minim ...|
|Duis aute irure d...|
|Excepteur sint oc...|
+-----+
```

```
scala> ds.show(2, false)
+-----+
|value
+-----+
|Lorem Ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
|Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
+-----+
only showing top 2 rows
```

```
scala>
```

Words Count : flatMap(...).count

```
scala> val wordDs = ds.flatMap(line =>
  line.replaceAll("[,:.!?]", " ")
    .replaceAll(" ", " ")
    .split(" "))
)
```

```
words: .. Dataset[String] = [value: string]
```

```
scala> wordDs.count
res3: Long = 69
```

```
scala> wordDs.show(10, false)
```

```
scala> wordDs.count()
val res5: Long = 69

scala> wordDs.show(10)
+-----+
|      value|
+-----+
|      Lorem|
|      Ipsum|
|      dolor|
|      sit|
|      amet|
|consectetur|
|      adipiscing|
|      elit|
|      sed|
|      do|
+-----+
only showing top 10 rows
```

Same word Count, using Spark-core RDD ?

```
scala> val lineRdd = sc.textFile("c:/data/loremIpsum.txt")
val lineRdd: org.apache.spark.rdd.RDD[String] = c:/data/loremIpsum.txt MapPartitionsRDD[26] at textFile at <console>:1

scala> lineRdd.show()
^
      error: value show is not a member of org.apache.spark.rdd.RDD[String]

scala> lineRdd.foreach(println(_))
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
Lorem Ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

scala>
```

no "RDD.show()" method !!

not easy to print, even "RDD[String]"

wordCount using RDD

!!! you might never use RDD directly !!

```
scala> val wordRdd = lineRdd.flatMap(line =>
|   line.replaceAll("[,:!?]", " ")
|   .replaceAll(" ", " ")
|   .split(" ")
| )
```

```
val wordRdd: org.apache.spark.rdd.RDD[String] =
MapPartitionsRDD[27] at flatMap at <console>:1
```

```
scala> wordRdd.count()
val res12: Long = 69
```

```
scala> val wordRdd = lineRdd.flatMap(line =>
|   line.replaceAll("[,:!?]", " ")
|   .replaceAll(" ", " ")
|   .split(" ")
| )
val wordRdd: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[27] at flatMap at <console>:1

scala> wordRdd.count()
val res12: Long = 69

scala> |
```

RDD vs Dataset look Similar ? But look closely

```
scala> ds
val res14: org.apache.spark.sql.Dataset[String] = [value: string]
```

```
scala> val wordDs = ds.flatMap(line =>
|   line.replaceAll("[,;.:!?]", " ")
|   .replaceAll(" ", " ")
|   .split(" ")
| )
val wordDs: org.apache.spark.sql.Dataset[String] = [value: string]
```



using spark-sql Dataset,
transformed dataset is still a "Dataset"

```
scala> lineRdd
val res13: org.apache.spark.rdd.RDD[String] = c:/data/loremIpsum.txt MapPartitionsRDD[26] at textFile at <console>:1
```

```
scala> val wordRdd = lineRdd.flatMap(line =>
|   line.replaceAll("[,;.:!?]", " ")
|   .replaceAll(" ", " ")
|   .split(" ")
| )
val wordRdd: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[27] at flatMap at <console>:1
```



using spark-core RDD,
transformed RDD is an instance of sub-class "MapPartitionsRDD"

Dataset vs RDD

Dataset and RDD are both lazy evaluated

BUT

RDD eval => just call the overriden method of sub-class (parent class is abstract)

Dataset action execution => need to analyze query, transform logical to physical
=> optimize => generate code => ... then finally call corresponding low-level RDD

RDD is low-level, Dataset is high-level using RDD internally

Words Count ... local[*] Debug in Java IDE

The screenshot shows a Java IDE interface with the following components:

- Top Bar:** Shows tabs for "Debug", "Project...", "Type ...", "JUnit", and several file icons.
- Left Sidebar:** Displays a tree view of threads, with the top item being "Thread [main] (Suspended breakpoint at SparkWordsCountAppMain.main(String[] args))".
- Code Editor:** The main window displays the code for `SparkWordsCountAppMain.java`. A breakpoint is set on line 54, which contains the log statement `Log.info("finished");`. The code implements a word count application using SparkSession and Dataset API.
- Variables View:** Located on the right, it lists variables with their values:

Name	Value
args	String[0] (id=38)
sparkConf	SparkConf (id=40)
spark	SparkSession (id=41)
loremIpsum	Dataset<T> (id=42)
lineCount	4
wordDs	Dataset<T> (id=43)
wordCount	69
- Console View:** At the bottom, the console output shows the results of the word count operation:

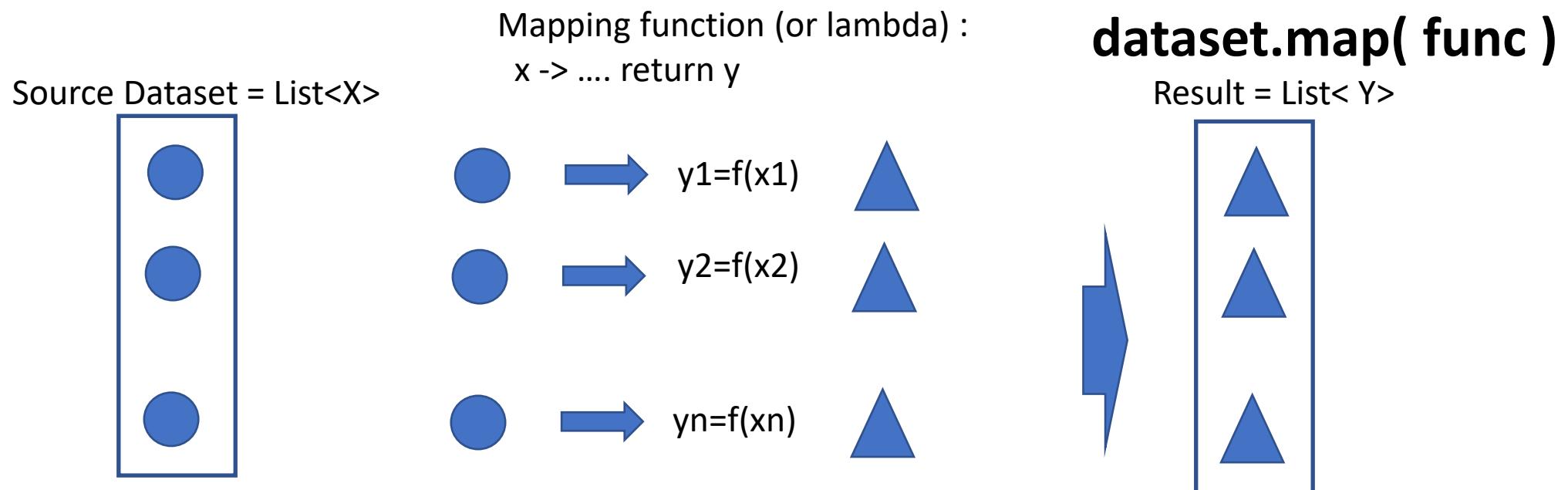
```
SparkWordsCountAppMain [Java Application]
+-----+
| tempor |
| incididunt |
| ut |
| labore |
| et |
| dolore |
| magna |
| aliqua. |
| Ut |
+-----+
only showing top 20 rows
```

11:06:20 INFO fr.an.tests.testspark.SparkWordsCountAppMain: words count:69

spark-submit --class <<MainClass>> <<JarFile>>

```
c:\arn\devPerso\test-snippets\test-spark (master -> origin)
λ spark-submit --class fr.an.tests.testspark.SparkWordsCountAppMain target/tests-spark-0.0.1-SNAPSHOT.jar
22/01/06 11:14:11 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
22/01/06 11:14:11 INFO SparkContext: Running Spark version 3.1.1
22/01/06 11:14:22 INFO DAGScheduler: Submitting ResultStage 5 (MapPartitionsRDD[28] at count at SparkWordsCountAppMain.java:51), which has no missing parents
22/01/06 11:14:22 INFO MemoryStore: Block broadcast_8 stored as values in memory (estimated size 10.1 KiB, free 413.6 MiB)
22/01/06 11:14:22 INFO MemoryStore: Block broadcast_8_piece0 stored as bytes in memory (estimated size 5.0 KiB, free 413.6 MiB)
22/01/06 11:14:22 INFO BlockManagerInfo: Added broadcast_8_piece0 in memory on DESKTOP-2EGCC8R:64427 (size: 5.0 KiB, free: 413.9 MiB)
22/01/06 11:14:22 INFO SparkContext: Created broadcast 8 from broadcast at DAGScheduler.scala:1383
22/01/06 11:14:22 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 5 (MapPartitionsRDD[28] at count at SparkWordsCountAppMain.java:51) (first 15 tasks are for partitions Vector(0))
22/01/06 11:14:22 INFO TaskSchedulerImpl: Adding task set 5.0 with 1 tasks resource profile 0
22/01/06 11:14:22 INFO TaskSetManager: Starting task 0.0 in stage 5.0 (TID 5) (DESKTOP-2EGCC8R, executor driver, partition 0, NODE_LOCAL, 4453 bytes) taskResourceAssignments Map()
22/01/06 11:14:22 INFO Executor: Running task 0.0 in stage 5.0 (TID 5)
22/01/06 11:14:22 INFO ShuffleBlockFetcherIterator: Getting 1 (60.0 B) non-empty blocks including 1 (60.0 B) local and 0 (0.0 B) host-local and 0 (0.0 B) remote blocks
22/01/06 11:14:22 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 4 ms
22/01/06 11:14:22 INFO Executor: Finished task 0.0 in stage 5.0 (TID 5). 2605 bytes result sent to driver
22/01/06 11:14:22 INFO TaskSetManager: Finished task 0.0 in stage 5.0 (TID 5) in 22 ms on DESKTOP-2EGCC8R (executor driver) (1/1)
22/01/06 11:14:22 INFO TaskSchedulerImpl: Removed TaskSet 5.0, whose tasks have all completed, from pool
22/01/06 11:14:22 INFO DAGScheduler: ResultStage 5 (count at SparkWordsCountAppMain.java:51) finished in 0,038 s
22/01/06 11:14:22 INFO DAGScheduler: Job 3 is finished. Cancelling potential speculative or zombie tasks for this job
22/01/06 11:14:22 INFO TaskSchedulerImpl: Killing all running tasks in stage 5: Stage finished
22/01/06 11:14:22 INFO DAGScheduler: Job 3 finished: count at SparkWordsCountAppMain.java:51, took 0,093309 s
22/01/06 11:14:22 INFO SparkWordsCountAppMain: words count:69
22/01/06 11:14:22 INFO SparkWordsCountAppMain: finished
22/01/06 11:14:22 INFO SparkUI: Stopped Spark web UI at http://DESKTOP-2EGCC8R:4040
22/01/06 11:14:22 INFO BlockManagerInfo: Removed broadcast_7_piece0 on DESKTOP-2EGCC8R:64427 in memory (size: 10.5 KiB, free: 413.9 MiB)
22/01/06 11:14:22 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
```

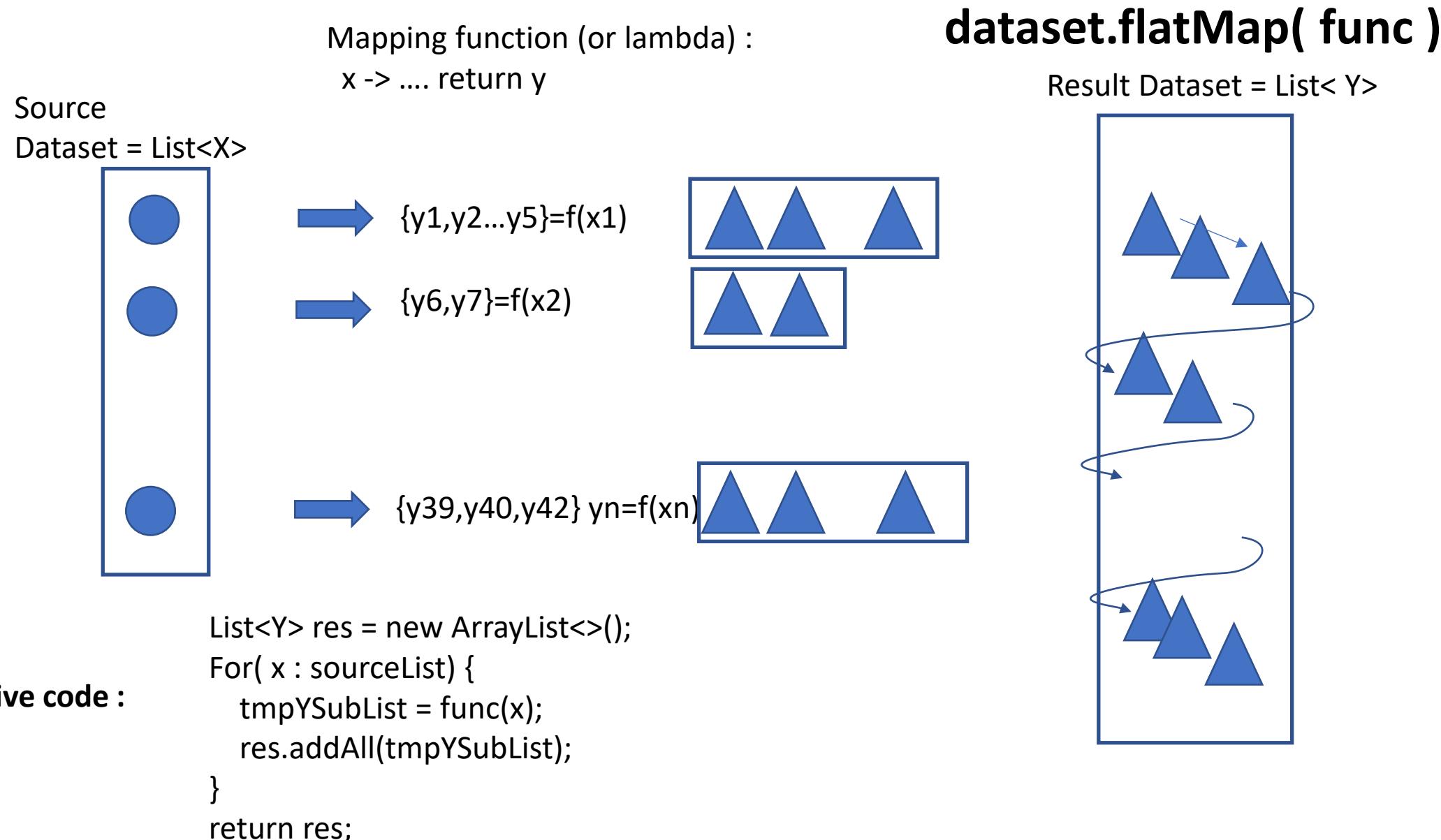
```
resultDs = dataset.map( x -> { ... return y; })
```



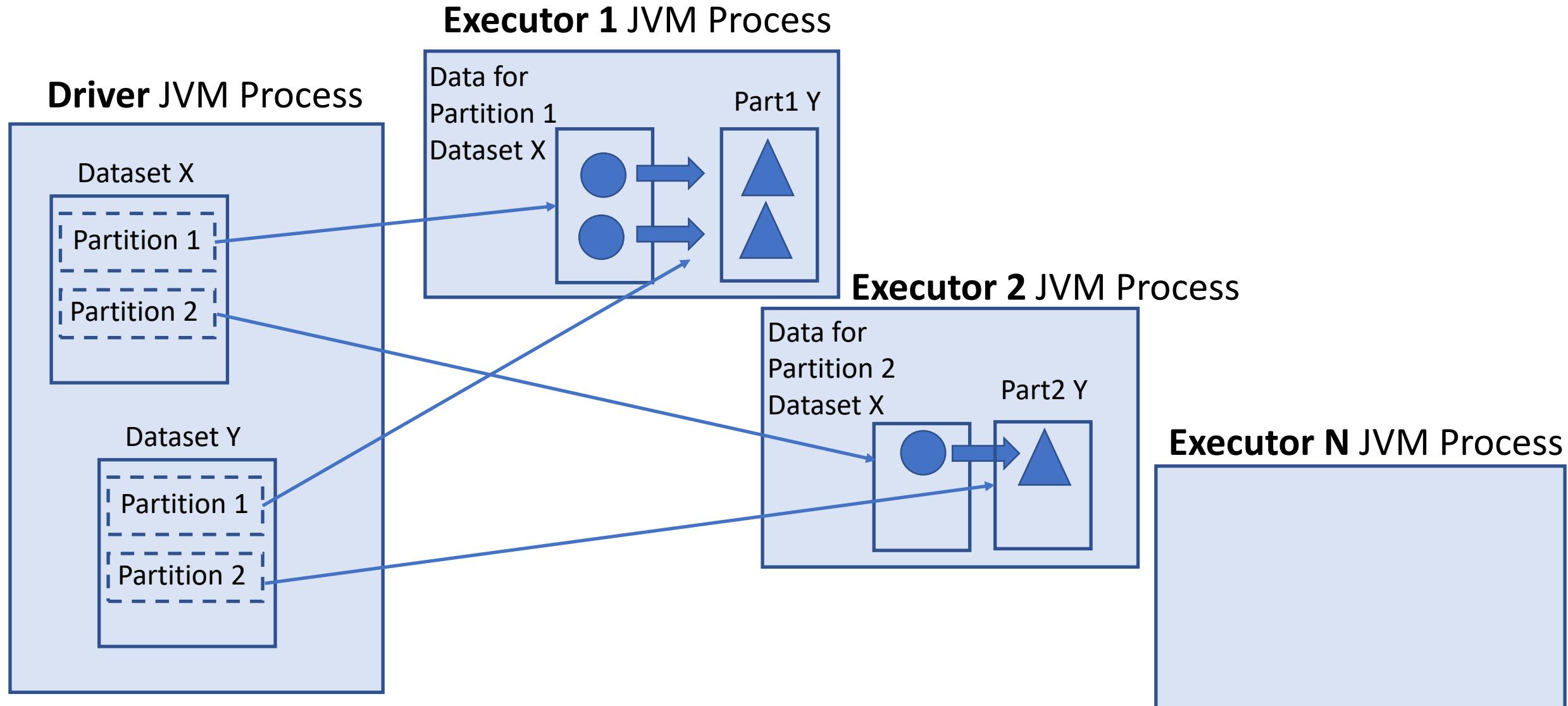
« Equivalent » imperative code :

```
List<Y> res = new ArrayList<>();  
For( x : sourceList) {  
    y = func(x);  
    res.add(y);  
}  
return res;
```

`resultDs = dataset.flatMap(x -> {... return list<y>; })`

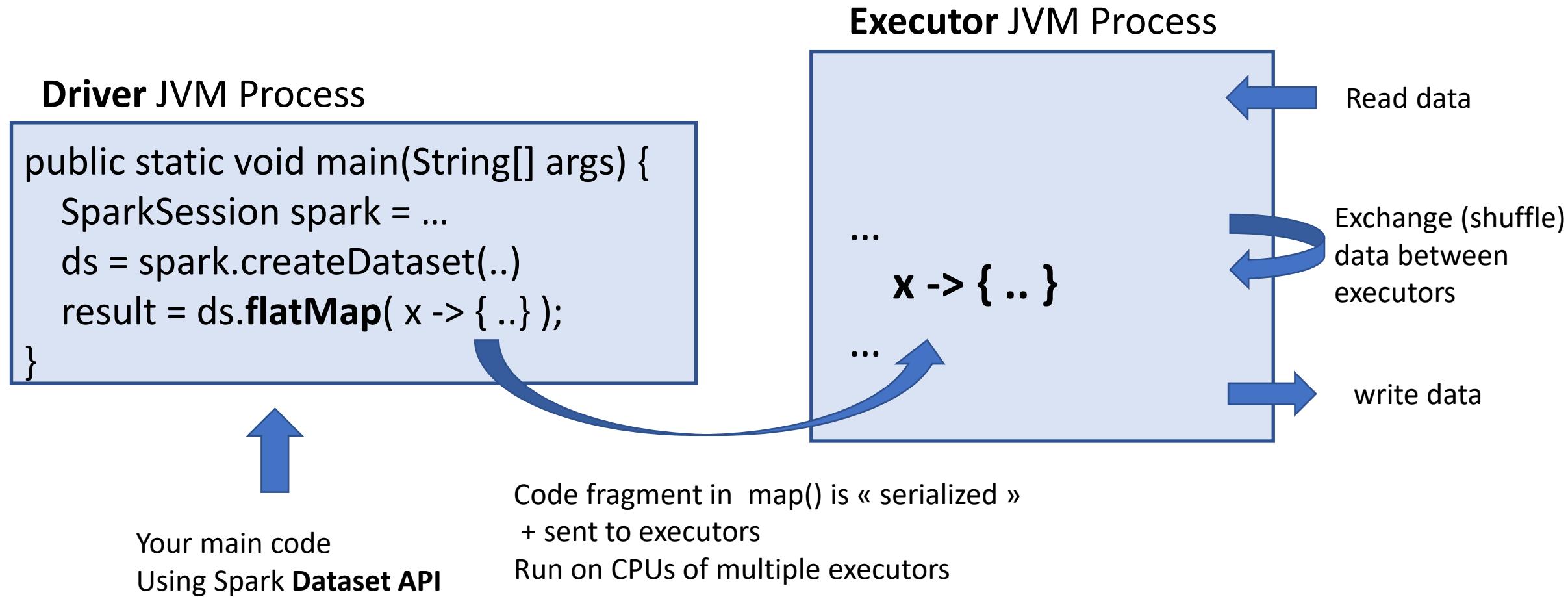


Dataset: « logical View » of « partitions » on Driver ... « data » allocated in-memory on Executors



Driver : Drives the main() program

Executors : Execute the functions on data



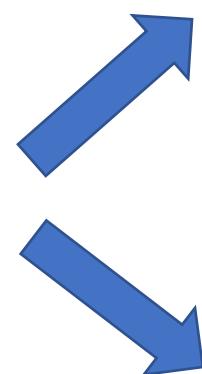
See the difference ?
Driver Api / Executor Engine
Code Logic / Data+Cpu Internal



Rdd.parallelize(..) vs Dataset.createDataset(..) ?

```
scala> val data = Array(1, 2, 3, 4, 5)
data: Array[Int] = Array(1, 2, 3, 4, 5)
```

Low-level API
using « **RDD** »



```
scala> val distData = sc.parallelize(data)
distData: org.apache.spark.rdd.RDD[Int] = ParallelCollec

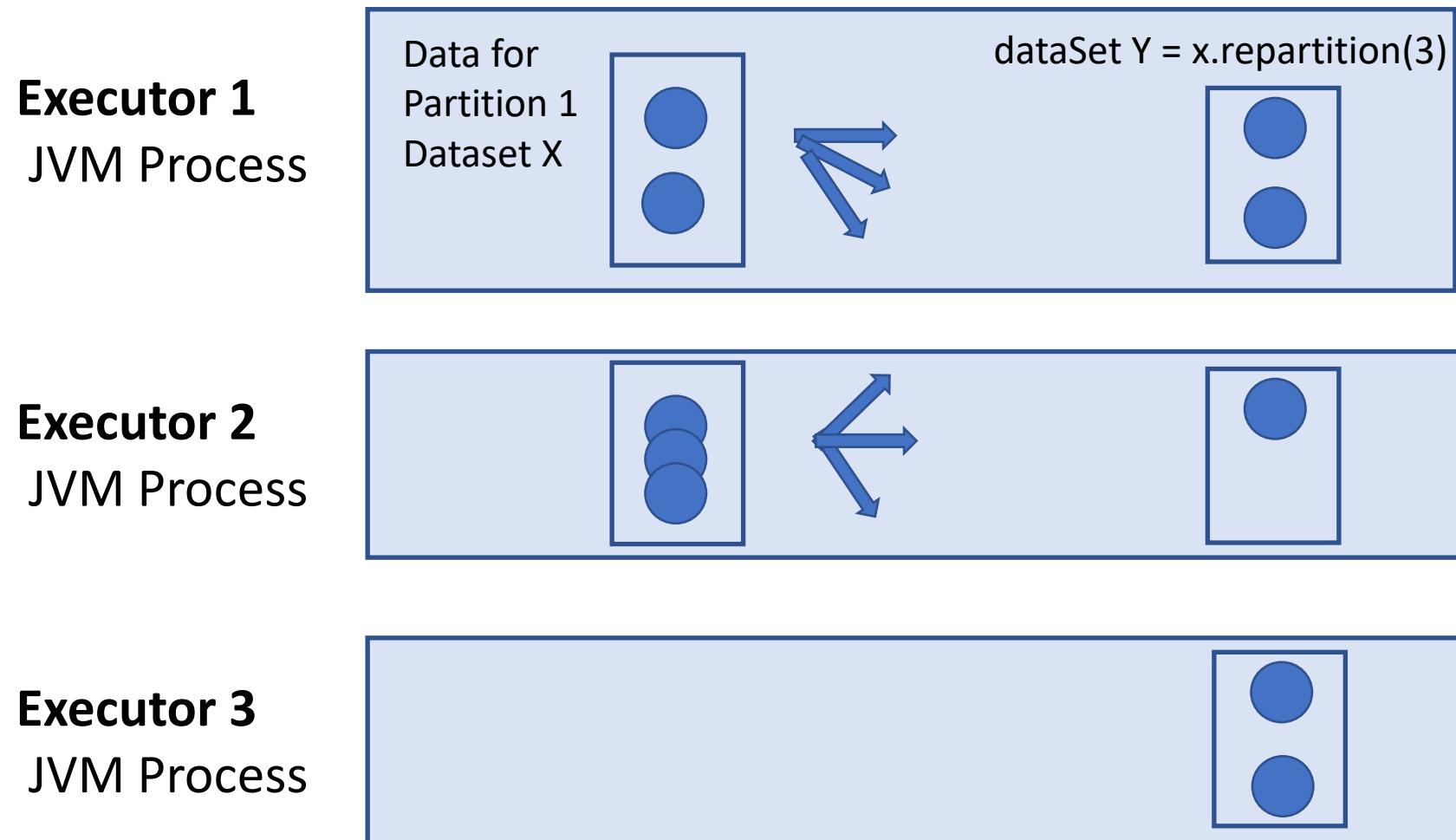
scala> distData.reduce((a,b) => a+b)
res2: Int = 15
```

High-level API
using « **DataSet** »

```
scala> val ds = spark.createDataset(data)
ds: org.apache.spark.sql.Dataset[Int] = [value: int]

scala> ds.reduce((a, b) => a+b)
res4: Int = 15
```

Exchange / Shuffle / Repartition / Coalesce / Distribute Wide transformation / Reduce



Wide vs Narrow Transformation

Wide Transformation (Exchange) between

dataSet Y = x.repartition(3).shuffle(..) .reduce(..) .sortBy()

Executor 1



Executor 2



Executor 3

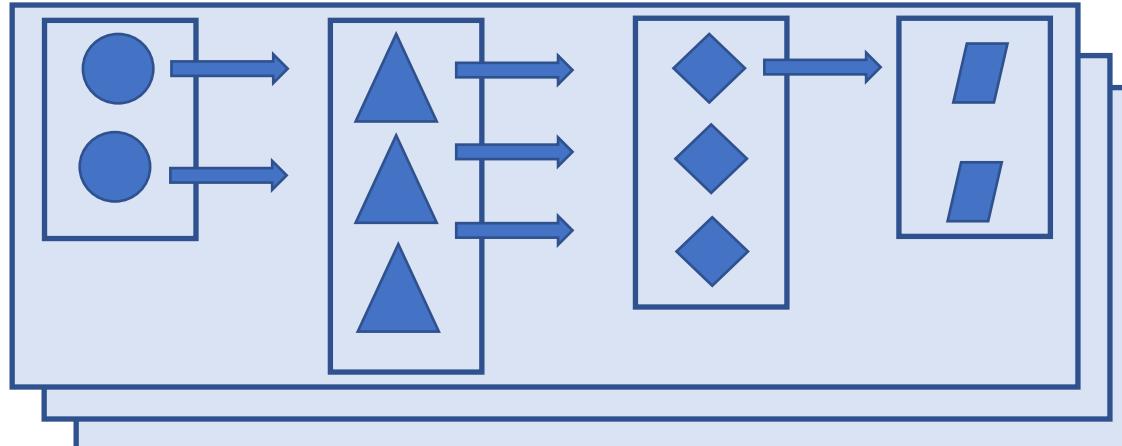


Network exchange
Between executors
... serialization/deserialization of byte data

Narrow Transformation

dataSet Y = x.map(func1) .flatMap(func2).filter(pred) .first(10) .sortWithinPartition ()

Partition Xi
+ corresp. Yi
on Executor 1/2/3



No data exchange
only in-memory pointer,
within same thread/process
... computation changes
but same logical partitionning

Things get complex...

« repartition() .flatMap() .explain() »

```
scala> loremIpsum.flatMap(line => line.replaceAll("[,:.!?]", " ").replaceAll(" ", " ").split(" ")).explain
== Physical Plan ==
*(1) SerializeFromObject [... AS value#116]
+- MapPartitions org.apache.spark.sql.Dataset$$Lambda$3735/14922651@1f3e32c, obj#115: java.lang.String
  +- DeserializeToObject value#12.toString, obj#114: java.lang.String
    +- FileScan text [value#12] Batched: false, DataFilters: [], Format: Text,
        Location: InMemoryFileIndex[file:/c:/data/loremIpsum.txt], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<value#116>
```

```
scala> loremIpsum.repartition(3).flatMap(line => line.replaceAll("[,:.!?]", " ").replaceAll(" ", " ").split(" ")).explain
== Physical Plan ==
*(1) SerializeFromObject [staticinvoke(... ) AS value#112]
+- MapPartitions org.apache.spark.sql.Dataset$$Lambda$3735/14922651@18e4389, obj#111: java.lang.String
  +- DeserializeToObject value#12.toString, obj#110: java.lang.String
    +- Exchange RoundRobinPartitioning(3), REPARTITION_WITH_NUM, [id=#235]
      +- FileScan text [value#12] Batched: false, DataFilters: [], Format: Text,
          Location: InMemoryFileIndex[file:/c:/data/loremIpsum.txt], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<value#112>
```

Confusing Questions at first Glance...

- 1/ RDD vs DataSet vs DataFrame ?
- 2/ meaning of Sql / Job / Task / Staging / Action ?
- 3/ Driver vs Executor ... where is executed my code ?
- 4/ Batch and Streaming api ?
- 5/ SQL or Code ? Functional API in java-scala-python ?
- 6/ Spark-shell / spark-sql / spark-submit / spark-thrift server / spark-history server ?
- 7/ Master = yarn/standone/k8s + mode = Client vs Cluster vs ...
- 8/ use Spark-ui / Console / logs?
- 9/ Performance diagnostic?
 DAG ? Metrics ? Shuffle ? SpillToDisk ? ScrewedPartition?
- 10/ Optimize or add more resources ?

Lets Explore..

What's visible from Spark-ui

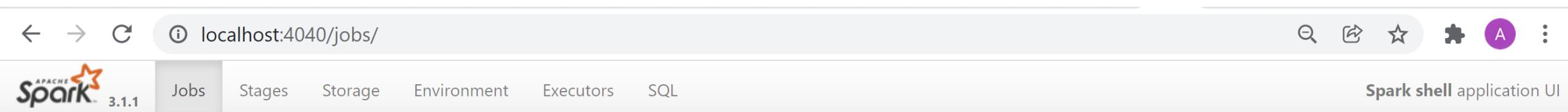
The concepts behinds RDD

The architecture of deployment

How to program/package/deploy java code

Spark-ui

http://localhost:4040



spark-ui [1/6]: Jobs

APACHE  3.1.1

Jobs Stages Storage Environment Executors SQL

Spark shell application UI

Spark Jobs [\(?\)](#)

User: arnaud
Total Uptime: 1,5 h
Scheduling Mode: FIFO
Completed Jobs: 11

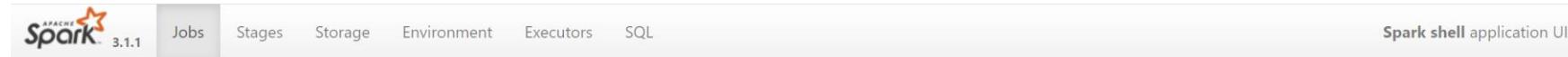
▶ Event Timeline

▼ Completed Jobs (11)

Page: 1 Pages. Jump to . Show items in a page.

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
10	show at <console>:26 show at <console>:26	2021/12/31 19:58:07	19 ms	1/1	<div style="width: 100%;">1/1</div>
9	count at <console>:26 count at <console>:26	2021/12/31 19:57:51	66 ms	2/2	<div style="width: 100%;">2/2</div>
8	show at <pastie>:26 show at <pastie>:26	2021/12/31 19:55:51	26 ms	1/1	<div style="width: 100%;">1/1</div>
7	show at <console>:26 show at <console>:26	2021/12/31 19:54:28	15 ms	1/1	<div style="width: 100%;">1/1</div>
6	show at <console>:26	2021/12/31 19:53:59	16 ms	1/1	<div style="width: 100%;">1/1</div>

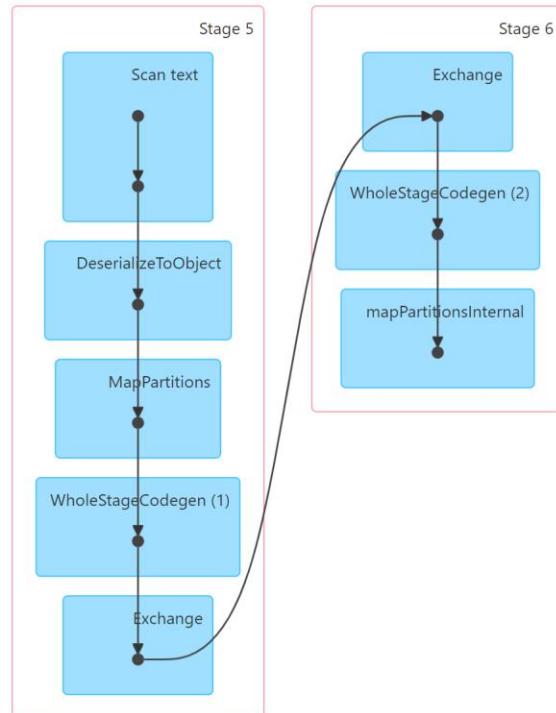
Spark-ui [1/6]: Job details DAG



Details for Job 4

Status: SUCCEEDED
Submitted: 2022/01/02 18:55:23
Duration: 0,2 s
Associated SQL Query: 2
Completed Stages: 2

- ▶ Event Timeline
- ▼ DAG Visualization



▼ Completed Stages (2)

Page: 1

1 Pages. Jump to 1 . Show 100 items in a page. Go

Spark-ui [1/6] detail Job timeline + statistics

APACHE  3.1.1

Jobs Stages Storage Environment Executors SQL

Spark shell application UI

Spark Jobs (?)

User: arnaud
Total Uptime: 27 min
Scheduling Mode: FIFO
Completed Jobs: 5

Event Timeline Enable zooming

Executors
Added (Blue)
Removed (Red)

Jobs
Succeeded (Blue)
Failed (Red)
Running (Green)

18:37 18:38 18:39 18:40 18:41 18:42 18:43 18:44 18:45 18:46 18:47 18:48 18:49 18:50 18:51 18:52 18:53 18:54 18:55
Sun 2 January

Completed Jobs (5)

Page: 1 Pages. Jump to Show items in a page. Go

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
4	count at <console>:26 count at <console>:26	2022/01/02 18:55:23	0,2 s	2/2	2/2
3	count at <console>:26 count at <console>:26	2022/01/02 18:54:35	0,4 s	2/2	2/2
2	reduce at <console>:26 reduce at <console>:26	2022/01/02 18:42:07	0,1 s	1/1	5/5
1	reduce at <console>:26 reduce at <console>:26	2022/01/02 18:39:46	19 ms	1/1	8/8
0	reduce at <console>:26 reduce at <console>:26	2022/01/02 18:39:22	1,0 s	1/1	8/8

Page: 1 Pages. Jump to Show items in a page. Go

Spark-ui [2/6]: detail Job > Stage > Task

Apache Spark 3.1.1

Jobs Stages Storage Environment Executors SQL

Spark shell application UI

Details for Stage 12 (Attempt 0)

Resource Profile Id: 0
Total Time Across All Tasks: 4 ms
Locality Level Summary: Process local: 1
Input Size / Records: 450.0 B / 1
Associated Job Ids: 10

▶ DAG Visualization
▶ Show Additional Metrics
▶ Event Timeline

Summary Metrics for 1 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	4.0 ms	4.0 ms	4.0 ms	4.0 ms	4.0 ms
GC Time	0.0 ms	0.0 ms	0.0 ms	0.0 ms	0.0 ms
Input Size / Records	450 B / 1	450 B / 1	450 B / 1	450 B / 1	450 B / 1

Showing 1 to 3 of 3 entries

▶ Aggregated Metrics by Executor

Tasks (1)

Show 20 entries

Search:

Index	Task ID	Attempt	Status	Locality level	Executor ID	Host	Logs	Launch Time	Duration	GC Time	Input Size / Records	Errors
0	12	0	SUCCESS	PROCESS_LOCAL	driver	DESKTOP-2EGCC8R		2021-12-31 19:58:07	4.0 ms		450 B / 1	

Showing 1 to 1 of 1 entries

Previous 1 Next

Spark-ui [3/6] Storage (cache, persist)

The screenshot shows the Apache Spark 3.1.1 UI interface. The top navigation bar includes links for Jobs, Stages, Storage (which is highlighted in grey), Environment, Executors, and SQL. To the right, it says "Spark shell application UI". The main content area is titled "Storage" and contains a table under the "RDDs" section. The table has columns for ID, RDD Name, Storage Level, Cached Partitions, Fraction Cached, Size in Memory, and Size on Disk. Two rows are listed:

ID	RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
28	*(1) SerializeFromObject [staticinvoke(class org.apache.spark.unsafe.types.UTF8String, StringType, fromString, input[0, java.lang.String, true], true, false) AS value#30] +- MapPartitions org.apache.spark.sql.Dataset\$\$Lambda\$4450/17435982@68a3d4, obj#29: java.lang.String +- DeserializeToObject value#21.toString, obj#28: java.lang.String +- FileScan text [value#21] Batched: false, DataFilters: [], Format: Text, Location: InMemoryFileIndex[file:/c:/data/lorem ipsum.txt], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<value:string>	Disk Memory Deserialized 1x Replicated	1	100%	952.0 B	0.0 B
40	FileScan text [value#21] Batched: false, DataFilters: [], Format: Text, Location: InMemoryFileIndex[file:/c:/data/lorem ipsum.txt], PartitionFilters: [], PushedFilters: [], ReadSchema: struct<value:string>	Disk Memory Deserialized 1x Replicated	1	100%	960.0 B	0.0 B

Spark-ui [4/6] Environment: jvm+spark+hadoop+..

The screenshot shows the Apache Spark 3.1.1 UI interface. At the top, there is a navigation bar with tabs: Jobs, Stages, Storage, Environment (which is highlighted in grey), Executors, and SQL. To the right of the tabs, it says "Spark shell application UI". On the left side, there is a sidebar with a "Environment" section containing links: Runtime Information, Spark Properties, Resource Profiles, Hadoop Properties, System Properties, and Classpath Entries. The main content area is titled "Environment" and contains a table titled "Spark Properties" with the following data:

▼ Spark Properties

Name	Value
spark.app.id	local-1640975191762
spark.app.name	Spark shell
spark.app.startTime	1640975189843
spark.driver.host	DESKTOP-2EGCC8R
spark.driver.port	57122
spark.executor.id	driver
spark.home	C:\apps\hadoop\spark-3.1.1
spark.jars	
spark.master	local[*]
spark.repl.class.outputDir	C:\Users\arnaud\AppData\Local\Temp\spark-169a28ae-4479-4585-ab47-1e354d2d1347\repl-ff2cda14-38ec-4b1e-9c63-c4f02e84bf14
spark.repl.class.uri	spark://DESKTOP-2EGCC8R:57122/classes
spark.scheduler.mode	FIFO
spark.sql.catalogImplementation	hive
spark.submit.deployMode	client
spark.submit.pyFiles	
spark.ui.showConsoleProgress	true

Spark-ui [5/6]: Executors / Driver

APACHE  3.1.1 Jobs Stages Storage Environment Executors SQL Spark shell application UI

Executors

[▶ Show Additional Metrics](#)

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(1)	0	0.0 B / 413.9 MiB	0.0 B	8	0	0	13	13	0.8 s (18.0 ms)	4.6 KiB	118 B	118 B	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(1)	0	0.0 B / 413.9 MiB	0.0 B	8	0	0	13	13	0.8 s (18.0 ms)	4.6 KiB	118 B	118 B	0

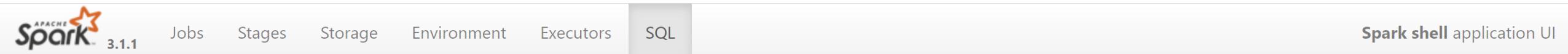
Executors

Show entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Thread Dump
driver	DESKTOP-2EGCC8R:57178	Active	0	0.0 B / 413.9 MiB	0.0 B	8	0	0	13	13	0.8 s (18.0 ms)	4.6 KiB	118 B	118 B	Thread Dump

Showing 1 to 1 of 1 entries

Spark-ui [6/6] SQL



SQL

Completed Queries: 3

▼ Completed Queries (3)

Page:

1 Pages. Jump to . Show items in a page.

ID ▾	Description	Submitted	Duration	Job IDs
2	count at <console>:26 <small>+details</small>	2022/01/02 18:55:23	0,2 s	[4]
1	count at <console>:26 <small>+details</small>	2022/01/02 18:54:35	0,7 s	[3]
0	reduce at <console>:26 <small>+details</small>	2022/01/02 18:42:06	0,2 s	[2]

Page:

1 Pages. Jump to . Show items in a page.

Spark-ui [6/6] Sql detail for query



Details for Query 2

Submitted Time: 2022/01/02 18:55:23

Duration: 0,2 s

Succeeded Jobs: 4

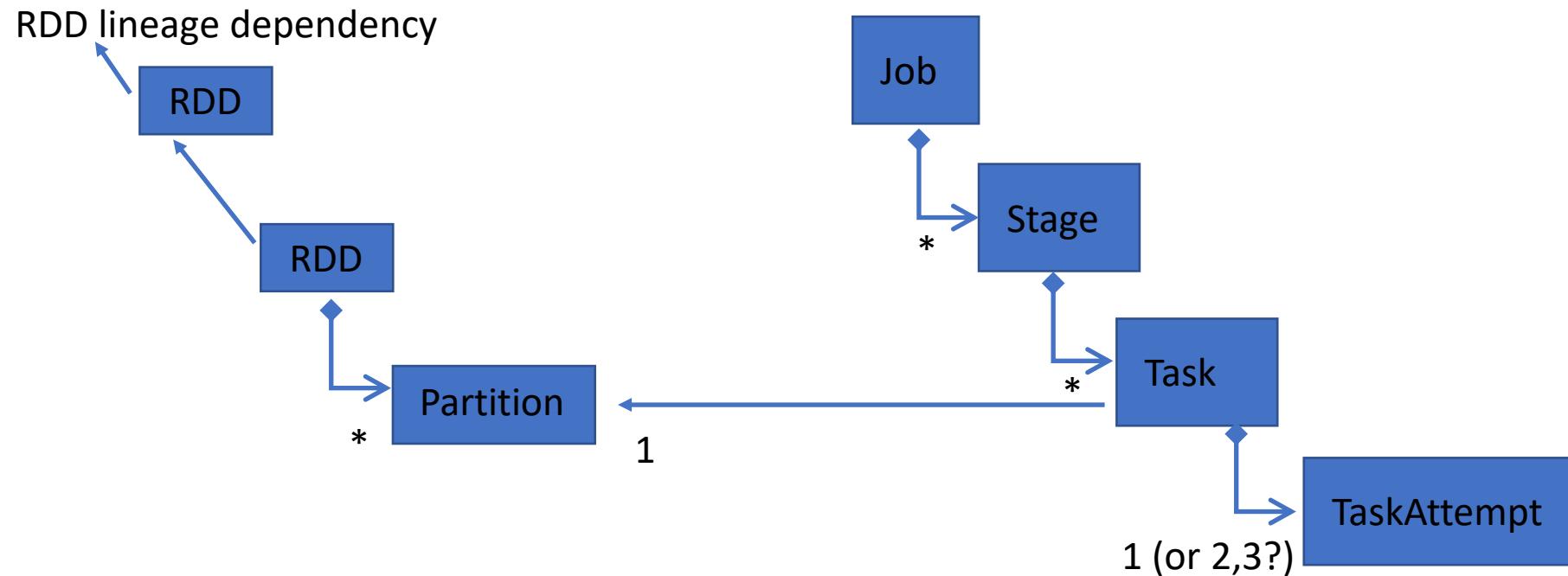
Show the Stage ID and Task ID that corresponds to the max metric



Spark-core Internal Concepts...

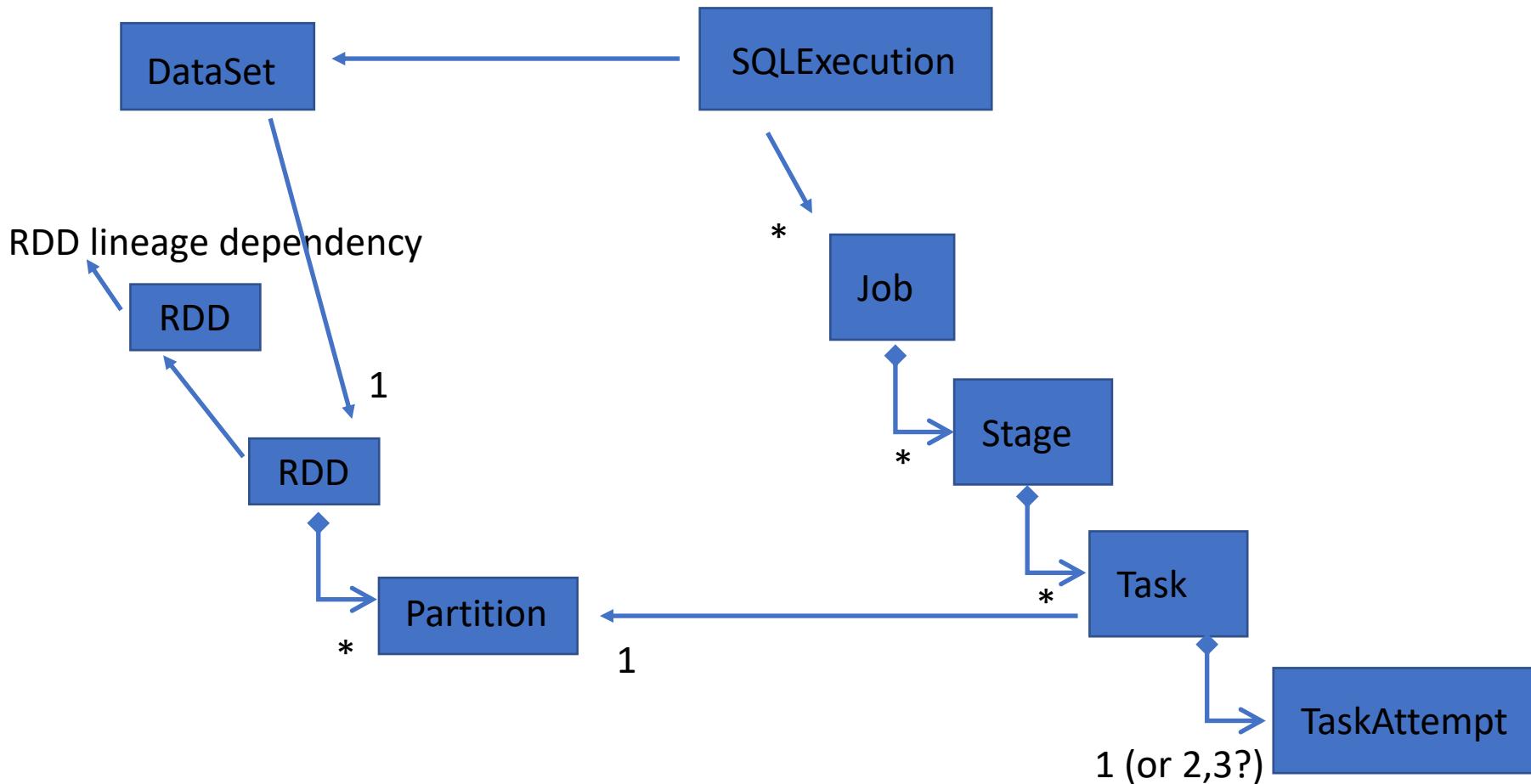
RDD, Job, Stage, Task

using only low-level spark-core "RDD.eval()
=> trigger 1 Job, with 1 or several Stages

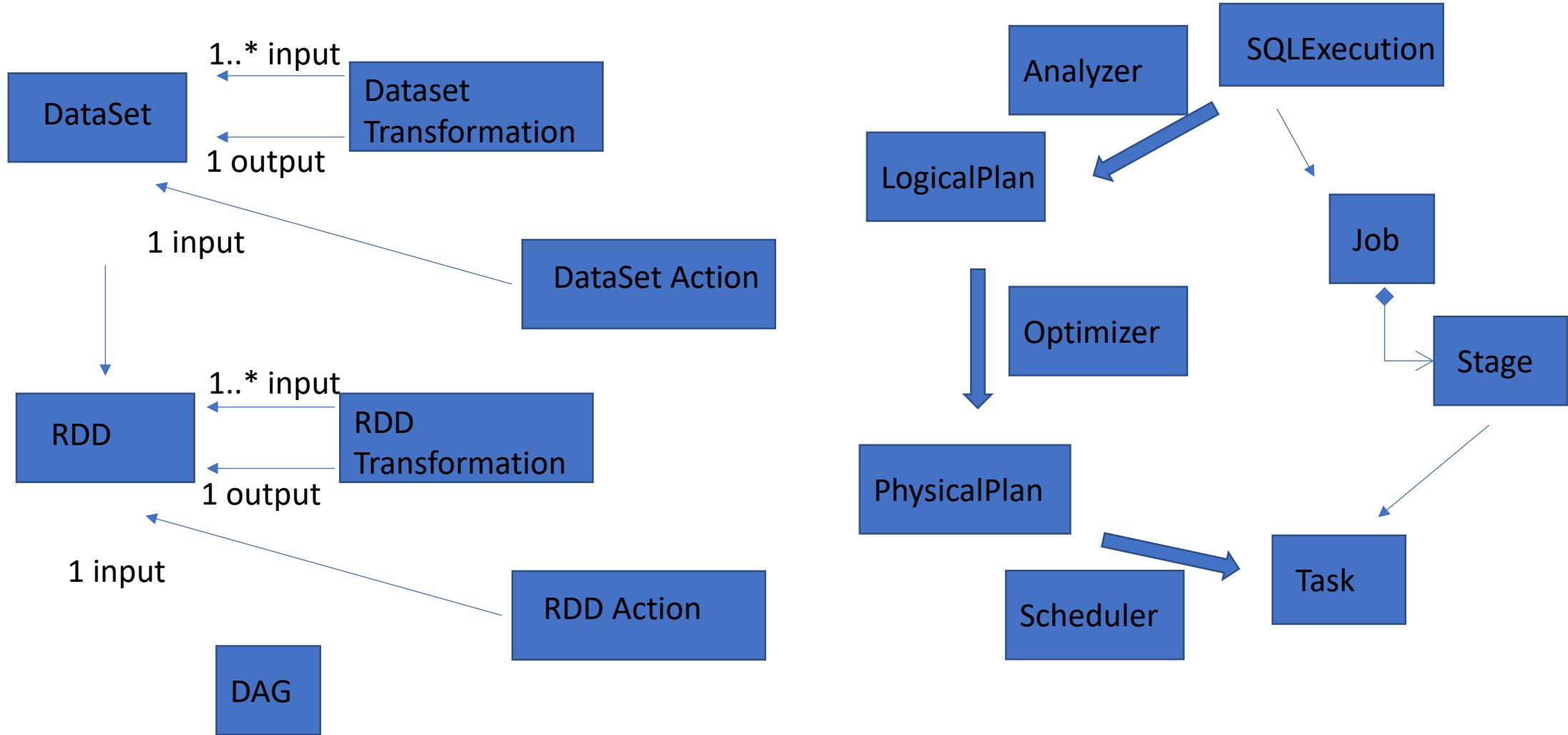


Spark-sql Internal Concepts... SQLExecution, Job, Stage, Task

using high-level spark-sql "Dataset.<action>()
=> trigger 1 or Several Jobs, with 1 or several Stages



Spark Internal Concepts... SQL, Job, Stage, Task

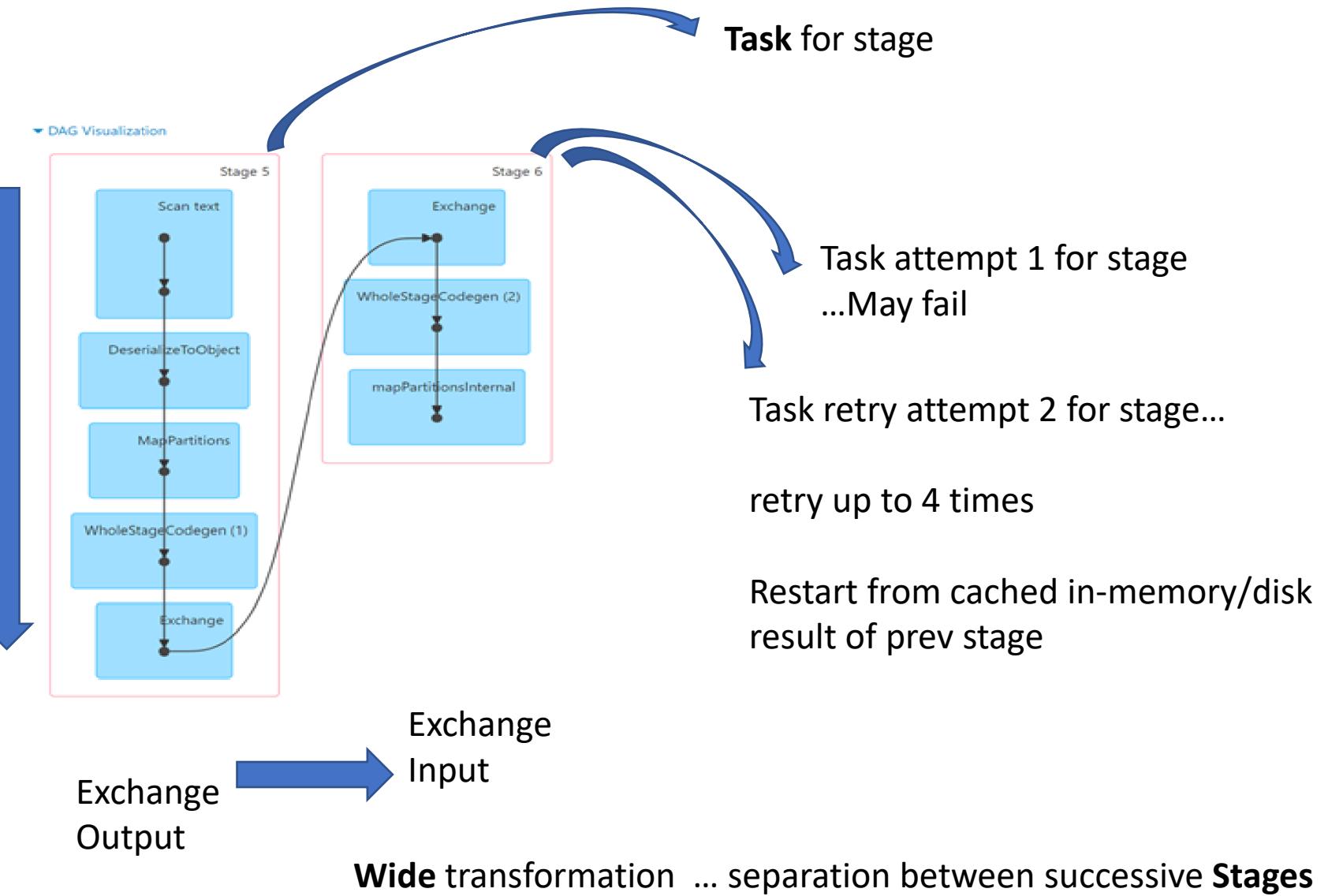


Job-Stage-Task in DAG

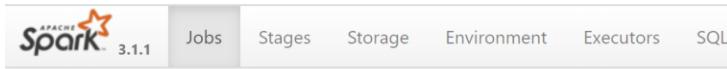
1 stage =
Successive **Narrow** transformations

Bytecode generated
... executed within same
Thread/Process

Job =
Full DAG of Stages
+ tasks (retry)
scheduling



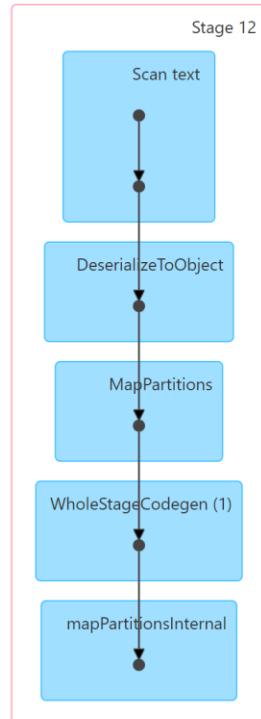
Word Count DAGS simple -vs- repartition()



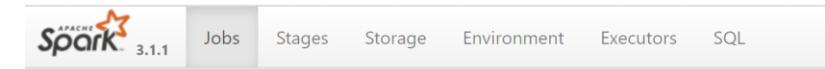
Details for Job 10

Status: SUCCEEDED
Submitted: 2021/12/31 19:58:07
Duration: 19 ms
Associated SQL Query: 10
Completed Stages: 1

- ▶ Event Timeline
- ▼ DAG Visualization



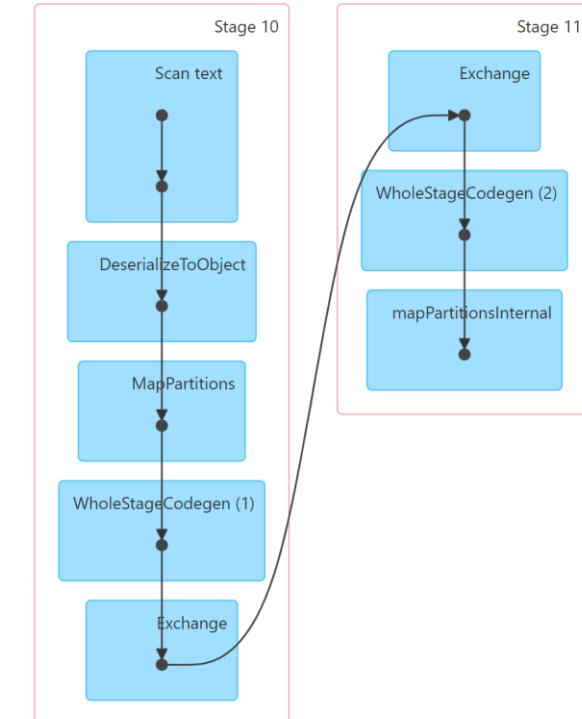
▼ Completed Stages (1)



Details for Job 9

Status: SUCCEEDED
Submitted: 2021/12/31 19:57:51
Duration: 66 ms
Associated SQL Query: 9
Completed Stages: 2

- ▶ Event Timeline
- ▼ DAG Visualization



▼ Completed Stages (2)

Dataset, RDD, DAG ..

R.D.D.

Resilient = can resist failure

.... can be recomputed on demand

Distributed = split in partitions, distributed over executors

Dataset = ... RDD result ends up to be a

`ImmutableList<ImmutableData>`

... from a function definition + inputs

`List<Data> computeFunc() { ... }`

abstract class org.apache.spark.rdd.RDD

```
/**  
 * A Resilient Distributed Dataset (RDD), the basic abstraction in Spark. Represents an immutable,  
 * partitioned collection of elements that can be operated on in parallel. This class contains the  
 * basic operations available on all RDDs, such as `map`, `filter`, and `persist`. In addition,  
 * [[org.apache.spark.rdd.PairRDDFunctions]] contains operations available only on RDDs of key-value  
 * pairs, such as `groupByKey` and `join`;  
 * [[org.apache.spark.rdd.DoubleRDDFunctions]] contains operations available only on RDDs of  
 * Doubles; and  
 * [[org.apache.spark.rdd.SequenceFileRDDFunctions]] contains operations available on RDDs that  
 * can be saved as SequenceFiles.  
 * All operations are automatically available on any RDD of the right type (e.g. RDD[(Int, Int)])  
 * through implicit.  
 *  
 * Internally, each RDD is characterized by five main properties:  
 *  
 * - A list of partitions  
 * - A function for computing each split  
 * - A list of dependencies on other RDDs  
 * - Optionally, a Partitioner for key-value RDDs (e.g. to say that the RDD is hash-partitioned)  
 * - Optionally, a list of preferred locations to compute each split on (e.g. block locations for  
 * an HDFS file)  
 *  
 * All of the scheduling and execution in Spark is done based on these methods, allowing each RDD  
 * to implement its own way of computing itself. Indeed, users can implement custom RDDs (e.g. for  
 * reading data from a new storage system) by overriding these functions. Please refer to the  
 * <a href="http://people.csail.mit.edu/matei/papers/2012/nsdi_spark.pdf">Spark paper</a>  
 * for more details on RDD internals.  
 */  
abstract class RDD[T: ClassTag](  
    @transient private var _sc: SparkContext,  
    @transient private var deps: Seq[Dependency[_]]  
) extends Serializable with Logging {  
  
    // ======  
    // Methods that should be implemented by subclasses of RDD  
    // ======  
  
    /**  
     * :: DeveloperApi ::  
     * Implemented by subclasses to compute a given partition.  
     */  
    @DeveloperApi  
    def compute(split: Partition, context: TaskContext): Iterator[T]
```

RDD

- conf() : SparkConf
 - ^Acompute(Partition, TaskContext) : Iterator<T>
 - ^AgetPartitions() : Partition[]
 - getDependencies() : Seq<Dependency<?>>
 - getPreferredLocations(Partition) : Seq<String>
 - partitioner() : Option<Partitioner>
 - sparkContext() : SparkContext
 - id() : int
 - name() : String
-
- ^Fdependencies() : Seq<Dependency<?>>
 - ^Fpartitions() : Partition[]
 - ^FgetNumPartitions() : int
 - ^FpreferredLocations(Partition) : Seq<String>
 - ^Fiterator(Partition, TaskContext) : Iterator<T>
 - getNarrowAncestors() : Seq<RDD<?>>
 - computeOrReadCheckpoint(Partition, TaskContext)
 - getOrCompute(Partition, TaskContext) : Iterator<T>

- isEmpty() : boolean
 - collectPartitions() : Object[]
 - checkpoint() : void
 - localCheckpoint() : RDD<T>
 - barrier() : RDDBarrier<T>
 - doCheckpoint() : void
 - markCheckpointed() : void
 - clearDependencies() : void
 - toJavaRDD() : JavaRDD<T>
 - isBarrier() : boolean
 - isCheckpointed() : boolean
 - isCheckpointedAndMaterialized() : boolean
 - isLocallyCheckpointed() : boolean
 - isReliablyCheckpointed() : boolean
 - getCheckpointFile() : Option<String>
 - creationSite() : CallSite
 - scope() : Option<RDDOperationScope>
 - getCreationSite() : String
 - elementClassTag() : ClassTag<T>
 - checkpointData() : Option<RDDCheckpointData<T>>
 - checkpointData\$_eq(Option<RDDCheckpointData<T>>) :
 - firstParent(ClassTag<U>) <U> : RDD<U>
 - parent(int, ClassTag<U>) <U> : RDD<U>
 - context() : SparkContext
 - retag(Class<T>) : RDD<T>
 - retag(ClassTag<T>) : RDD<T>
-
- persist() : RDD<T>
 - cache() : RDD<T>
 - persist(StorageLevel) : RDD<T>
 - unpersist(boolean) : RDD<T>
 - unpersist\$default\$1() : boolean
 - getStorageLevel() : StorageLevel

- # RDD operators...
- withScope(Function0<U>) <U> : U
 - map(Function1<T, U>, ClassTag<U>) <U> : RDD<U>
 - flatMap(Function1<T, TraversableOnce<U>>, ClassTag<U>) <U> : RDD<U>
 - filter(Function1<T, Object>) : RDD<T>
 - distinct(int, Ordering<T>) : RDD<T>
 - distinct\$default\$2(int) : Ordering<T>
 - repartition(int, Ordering<T>) : RDD<T>
 - repartition\$default\$2(int) : Ordering<T>
 - coalesce(int, boolean, Option<PartitionCoalescer>, Orderer) : RDD<T>
 - coalesce\$default\$2() : boolean
 - coalesce\$default\$3() : Option<PartitionCoalescer>
 - coalesce\$default\$4(int, boolean, Option<PartitionCoalescer>) : RDD<T>
 - sample(boolean, double, long) : RDD<T>
 - sample\$default\$3() : long
 - randomSplit(double[], long) : RDD<T>[]
 - randomSplit\$default\$2() : long
 - randomSampleWithRange(double, double, long) : RDD<T>
 - takeSample(boolean, int, long) : Object
 - union(RDD<T>) : RDD<T>
 - \$plus\$plus(RDD<T>) : RDD<T>
 - sortBy(Function1<T, K>, boolean, int, Ordering<K>, ClassTag<K>) : RDD<T>
 - sortBy\$default\$2() <K> : boolean
 - sortBy\$default\$3() <K> : int
 - intersection(RDD<T>) : RDD<T>
 - intersection(RDD<T>, Partitioner, Ordering<T>) : RDD<T>
 - intersection(RDD<T>, int) : RDD<T>
 - intersection\$default\$3(RDD<T>, Partitioner) : Ordering<T>
 - cartesian(RDD<U>, ClassTag<U>) <U> : RDD<Tuple2<T, U>>
 - groupBy(Function1<T, K>, ClassTag<K>) <K> : RDD<T>
 - groupBy(Function1<T, K>, int, ClassTag<K>) <K> : RDD<T>
 - groupBy(Function1<T, K>, Partitioner, ClassTag<K>, Ordering<K>) : RDD<T>
 - groupBy\$default\$4(Function1<T, K>, Partitioner) <K> : RDD<T>
 - pipe(String) : RDD<String>
 - pipe(String, Map<String, String>) : RDD<String>
 - pipe(Seq<String>, Map<String, String>, Function1<Func<String>, Map<String, String>>) : RDD<String>
 - mapPartitions\$default\$2() <U> : boolean
 - mapPartitionsWithIndexInternal(Function2<Object, Iterator<U>>, Function2<Object, Iterator<U>>) <U> : RDD<U>
 - mapPartitionsInternal(Function1<Iterator<T>, Iterator<U>>) <U> : boolean
 - mapPartitionsWithIndex(Function2<Object, Iterator<T>>, Function2<Object, Iterator<T>>) <U> : RDD<U>
 - mapPartitionsWithIndexInternal\$default\$2() <U> : boolean
 - mapPartitionsWithIndexInternal\$default\$3() <U> : boolean
 - mapPartitionsWithIndex\$default\$2() <U> : boolean
 - zip(RDD<U>, ClassTag<U>) <U> : RDD<Tuple2<T, U>>
 - zipPartitions(RDD, boolean, Function2<Iterator<T>, Iterator>) : RDD
 - zipPartitions(RDD, Function2<Iterator<T>, Iterator>) : RDD
 - zipPartitions(RDD, RDD<C>, boolean, Function3<Iterator<T>, Iterator<C>>) : RDD
 - zipPartitions(RDD, RDD<C>, Function3<Iterator<T>, Iterator<C>>) : RDD
 - zipPartitions(RDD, RDD<C>, RDD<D>, boolean, Function4<Iterator<T>, Iterator<C>, Iterator<D>>) : RDD
 - zipPartitions(RDD, RDD<C>, RDD<D>, Function4<Iterator<T>, Iterator<C>, Iterator<D>>) : RDD
 - foreach(Function1<T, BoxedUnit>) : void
 - foreachPartition(Function1<Iterator<T>, BoxedUnit>) : void
 - toLocalIterator() : Iterator<T>
 - collect(PartialFunction<T, U>, ClassTag<U>) <U> : RDD<U>
 - subtract(RDD<T>) : RDD<T>
 - subtract(RDD<T>, int) : RDD<T>
 - subtract(RDD<T>, Partitioner, Ordering<T>) : RDD<T>
 - subtract\$default\$3(RDD<T>, Partitioner) : Ordering<T>
 - reduce(Function2<T, T, T>) : T
 - treeReduce(Function2<T, T, T>, int) : T
 - treeReduce\$default\$2() : int
 - fold(T, Function2<T, T, T>) : T
 - aggregate(U, Function2<U, T, U>, Function2<U, U, U>, ClassTag<U>) <U> : RDD<U>
 - treeAggregate(U, Function2<U, T, U>, Function2<U, U, U>, ClassTag<U>) <U> : RDD<U>
 - treeAggregate\$default\$4(U) <U> : int
 - countApprox(long, double) : PartialResult<BoundedDouble>
 - countApprox\$default\$2() : double
 - countByValue(Ordering<T>) : Map<T, Object>
 - countByValue\$default\$1() : Ordering<T>
 - countByValueApprox(long, double, Ordering<T>) : PartialResult<BoundedDouble>
 - countByValueApprox\$default\$2() : double
 - countByValueApprox\$default\$3(long, double) : Ordering<T>
 - countApproxDistinct(int, int) : long
 - countApproxDistinct(double) : long
 - countApproxDistinct\$default\$1() : double
 - take(int) : Object
 - takeSample\$default\$3() : long
 - top(int, Ordering<T>) : Object
 - takeOrdered(int, Ordering<T>) : Object
 - max(Ordering<T>) : T
 - min(Ordering<T>) : T
 - saveAsTextFile(String) : void
 - saveAsTextFile(String, Class<? extends CompressionCodec>) : void
 - saveAsObjectFile(String) : void
 - keyBy(Function1<T, K>) <K> : RDD<Tuple2<K, T>>
 - distinct() : RDD<T>
 - glom() : RDD<Object>
 - collect() : Object
 - count() : long
 - zipWithIndex() : RDD<Tuple2<T, Object>>
 - zipWithUniqueId() : RDD<Tuple2<T, Object>>
 - first() : T

Sub-class Hierarchy extending RDD

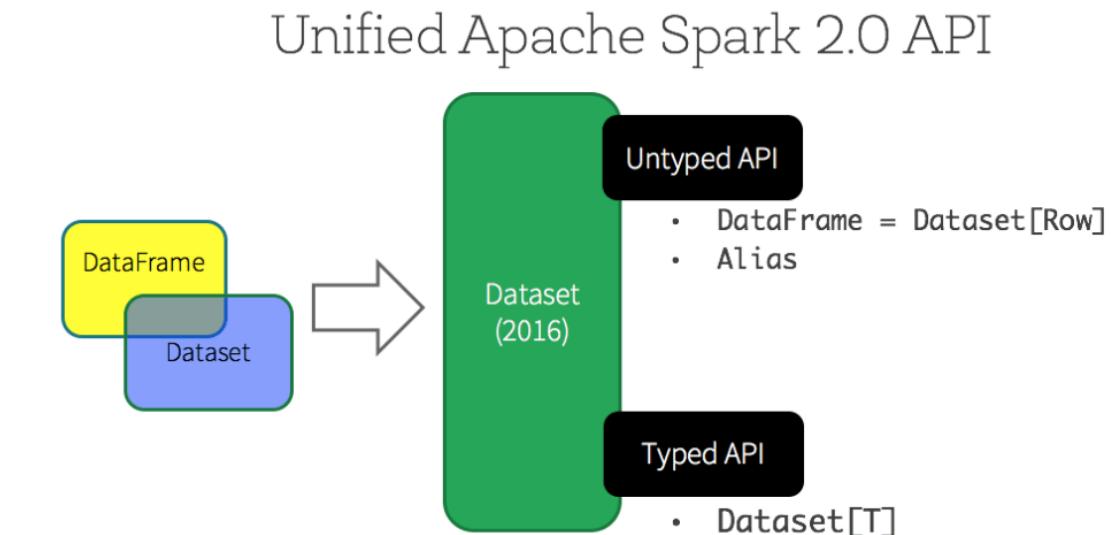


Deprecated DataFrame api ... now DataSet<Row>

DataFrame is deprecated since 2016

Spark still have 2 usage « kind »

- untyped (sql) ... DataSet<Row> = like Jdbc ResultSet
- Typed ... like ArrayList<YourClass>



interface org.apache.spark.sql.Row

Ctor + schema +
SCALA « deconstructor »

```
Row
  empty() : Row
  merge(Seq<Row>) : Row
  fromTuple(Product) : Row
  fromSeq(Seq<Object>) : Row
  unapplySeq(Row) : Some<Seq<Object>>
  size() : int
  length() : int
  schema() : StructType
  apply(int) : Object
```

```
/*
 * This method can be used to extract fields from a [[Row]] object in a pattern match.
 * {{{
 * import org.apache.spark.sql._
 *
 * val pairs = sql("SELECT key, value FROM src").rdd.map {
 *   case Row(key: Int, value: String) =>
 *     key -> value
 * }
 * }}}
 */
def unapplySeq(row: Row): Some[Seq[Any]] = Some(row.toSeq)
```

String + json

- [toString\(\): String](#)
- [copy\(\): Row](#)
- [anyNull\(\): boolean](#)
- [equals\(Object\): boolean](#)
- [hashCode\(\): int](#)
- [toSeq\(\): Seq<Object>](#)
- [mkString\(\): String](#)
- [mkString\(String\): String](#)
- [mkString\(String, String, String\): String](#)
- [getAnyValAs\(int\) <T>: T](#)
- [json\(\): String](#)
- [prettyJson\(\): String](#)
- [jsonValue\(\): JValue](#)

Typed field getters

- [get\(int\): Object](#)
- [isNullAt\(int\): boolean](#)
- [getBoolean\(int\): boolean](#)
- [getByte\(int\): byte](#)
- [getShort\(int\): short](#)
- [getInt\(int\): int](#)
- [getLong\(int\): long](#)
- [getFloat\(int\): float](#)
- [getDouble\(int\): double](#)
- [getString\(int\): String](#)
- [getDecimal\(int\): BigDecimal](#)
- [getDate\(int\): Date](#)
- [getLocalDate\(int\): LocalDate](#)
- [getTimestamp\(int\): Timestamp](#)
- [getInstant\(int\): Instant](#)
- [getSeq\(int\) <T>: Seq<T>](#)
- [getList\(int\) <T>: List<T>](#)
- [getMap\(int\) <K, V>: Map<K, V>](#)
- [getJavaMap\(int\) <K, V>: Map<K, V>](#)
- [getStruct\(int\): Row](#)
- [getAs\(int\) <T>: T](#)
- [getAs\(String\) <T>: T](#)
- [fieldIndex\(String\): int](#)
- [getValuesMap\(Seq<String>\) <T>: Map<String](#)

Class « Row » methods API ... you should not care ?

With SQL built-in functions

... you normally do NOT need any « map(row => { ... row.someMethod(..) }) »

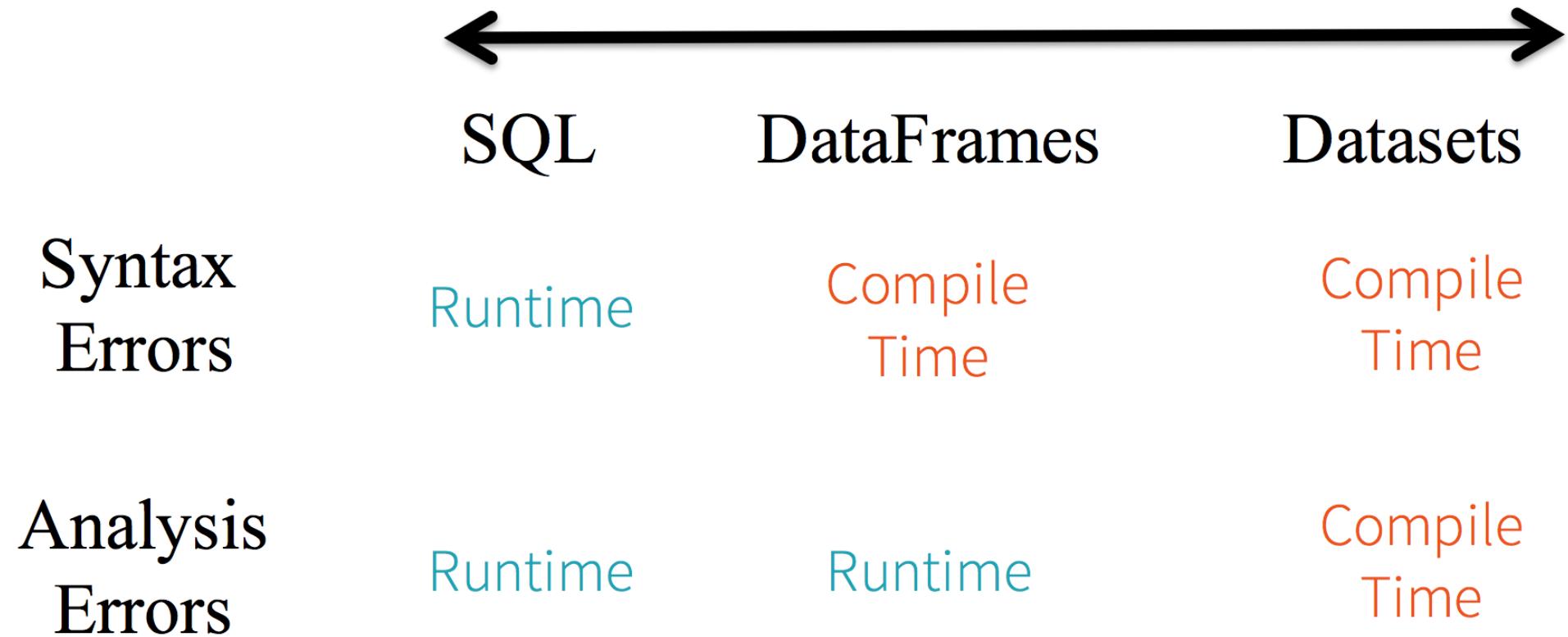
Map code fragments are advanced usages, running on « Executors »

... You focus on « Driver » code

List<Row> ls = dataset.collect();

... probably a bad idea for in-memory fitting data on driver side

SQL -vs- Compiled Custom Class



SQL ... « where » vs filter(lambda predicate)

```
spark.sql("")  
  select p.id  
  from User  
  where p.firstName = 'arnaud'  
")
```



```
Dataset<User> ds = ... load(.. Encoder.bean(User.class) );  
ds  
  .filter( x -> x.getFirstName().equals("arnaud") )  
  .map( x -> x.getId() );
```

Concise vs Maintanable ...

Java : verbose langage with parenthesis ...

+ additionnal JavaBean getter/setter naming conventions verbosity ...

Spark use « Encoder.beans() » to enforce this

But works well + safe at compile time + easily readable & maintainable

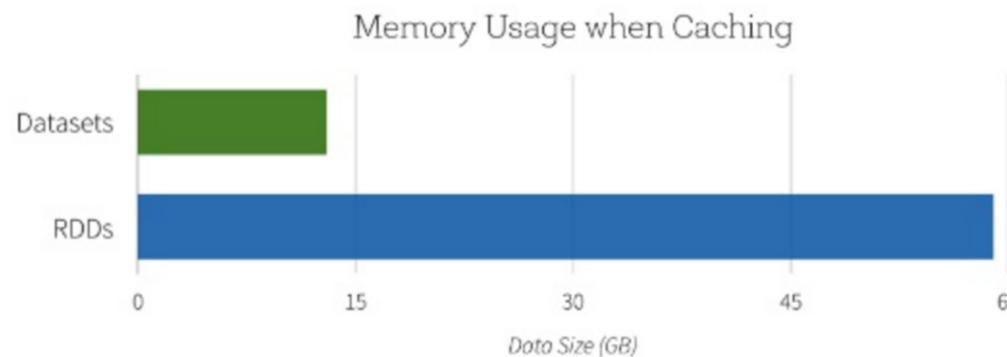
Did you ever try to refactor > 20 000 lines of legacy SQL ?

Did you ever read > 20 000 lines of complex/idiomatic Scala code ?

Spark Tungsten ...

Internal memory encoder for « Row »

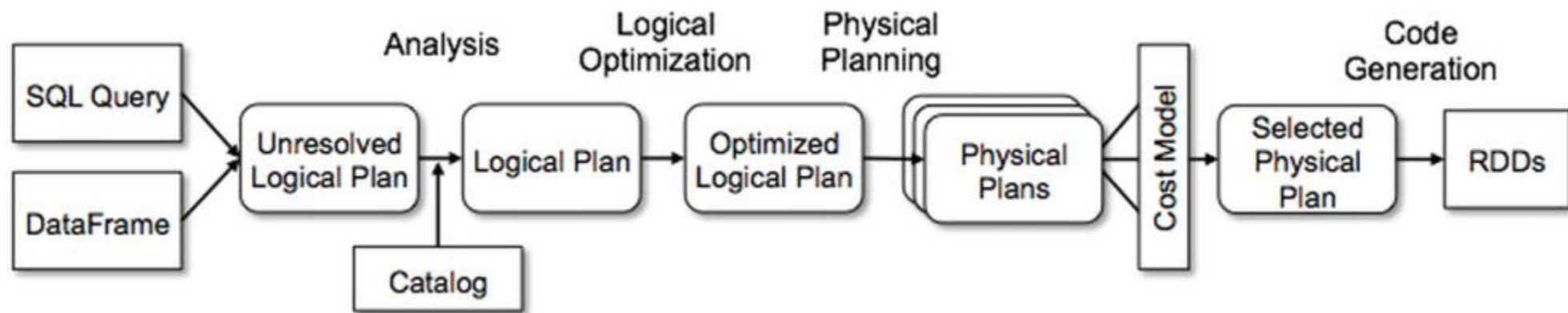
Space Efficiency



Second, since **Spark as a compiler** understands your Dataset type JVM object, it maps your type-specific JVM object to **Tungsten's internal memory representation** using **Encoders**. As a result, **Tungsten** Encoders can efficiently serialize/deserialize JVM objects as well as generate compact bytecode that can execute at superior speeds.

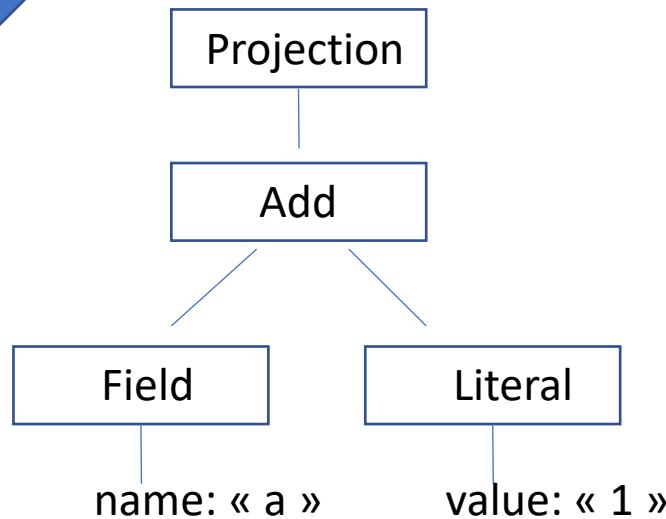
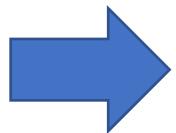
- ◦ ◦ **OffHeap vs Managed JVM Memory**
⇒ May handle Huge DATA
with « small » JVM GC memory
- ◦ ◦ **Efficient network (shuffling) serialization**
- ◦ ◦ **Efficient SpillToDisk (swapping)**

SQL / DataSet -> AST -> Catalyst -> Code + RDD



Compiler chain: Parser -> AST -> ...

Sql> select (a+1) from ...



```
110  case class Add(left: Expression, right: Expression) extends BinaryArithmetic {
111    override def symbol: String = "+"
112
113    lazy val numeric = dataType match {
114      case n: NumericType => n.numeric.asInstanceOf[Numeric[Any]]
115      case other => sys.error(s"Type $other does not support numeric operations")
116    }
117
118    override def eval(input: Row): Any = {
119      val evalE1 = left.eval(input)
120      if(evalE1 == null) {
121        null
122      } else {
123        val evalE2 = right.eval(input)
124        if (evalE2 == null) {
125          null
126        } else {
127          numeric.plus(evalE1, evalE2)
128        }
129      }
130    }
131  }
```

Catalyst AST -> CodeGenerator

apache / spark Public Watch 2.1k Fork 25k Star 31.7k

<> Code Pull requests 241 Actions Projects Security Insights

fedbf7074 spark / sql / catalyst / src / main / scala / org / apache / spark / sql / catalyst / expressions / codegen / **CodeGenerator.scala** Go to file ...

```
33 /**
34  * A base class for generators of byte code to perform expression evaluation. Includes a set of
35  * helpers for referring to Catalyst types and building trees that perform evaluation of individual
36  * expressions.
37 */
38 abstract class CodeGenerator[InType <: AnyRef, OutType <: AnyRef] extends Logging {
363     case Add(e1, e2) => (e1, e2) evaluate { case (eval1, eval2) => q"$eval1 + $eval2" }
364     case Subtract(e1, e2) => (e1, e2) evaluate { case (eval1, eval2) => q"$eval1 - $eval2" }
365     case Multiply(e1, e2) => (e1, e2) evaluate { case (eval1, eval2) => q"$eval1 * $eval2" }
640 }
641 }
```

Architecture Components

Client-Cluster ... Driver & Executors

program/library/servers

Cluster Client -> Driver (-> SparkContext) -> Executor

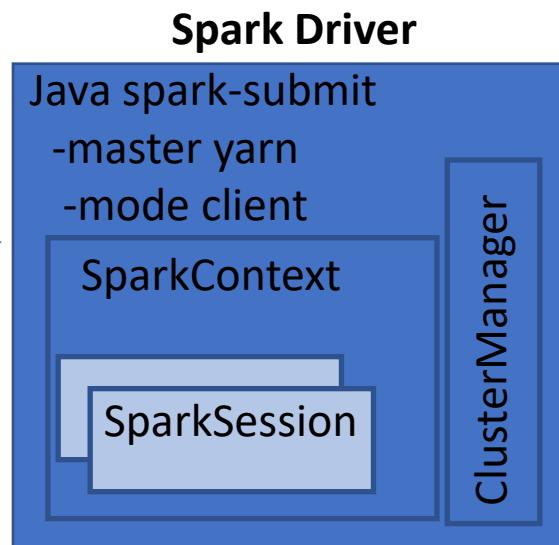
Case 1/ Launch directly java.. main(String[] args) {
 SparkContext.getOrCreate()..
}

Case 2/ spark-submit –mode **client**

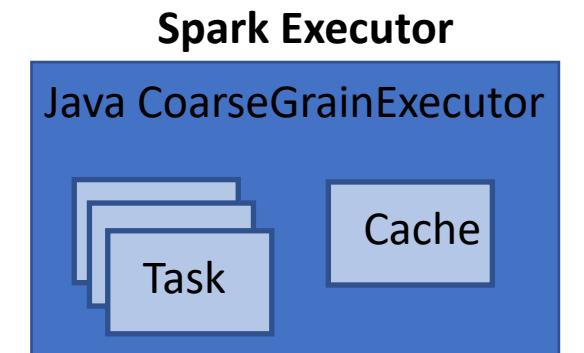
Case 3/
Yarn launch spark –mode **cluster**
+ relaunch if failed

Java spark-submit
-master yarn
-mode cluster

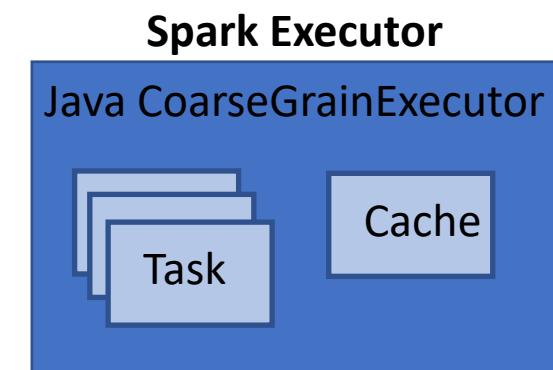
OS



OS WorkerNode
(+ YarnNodeManager)



OS WorkerNode
(+ YarnNodeManager)



OS WorkerNode
(+ YarnNodeManager)

Develop, Package, Deploy

pom.xml

```
<properties>
    <spark.version>3.5.3</spark.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-core_2.13</artifactId>
        <version>${spark.version}</version>
    </dependency>

    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-sql_2.13</artifactId>
        <version>${spark.version}</version>
    </dependency>
</dependencies>
```

Strange version suffix in artifactId
... the scala runtime version

SparkConf / (Java)SparkContext / SparkSession

```
SparkConf sparkConf = new SparkConf()
    .setAppName(appName)
    .setMaster(master)
    ;

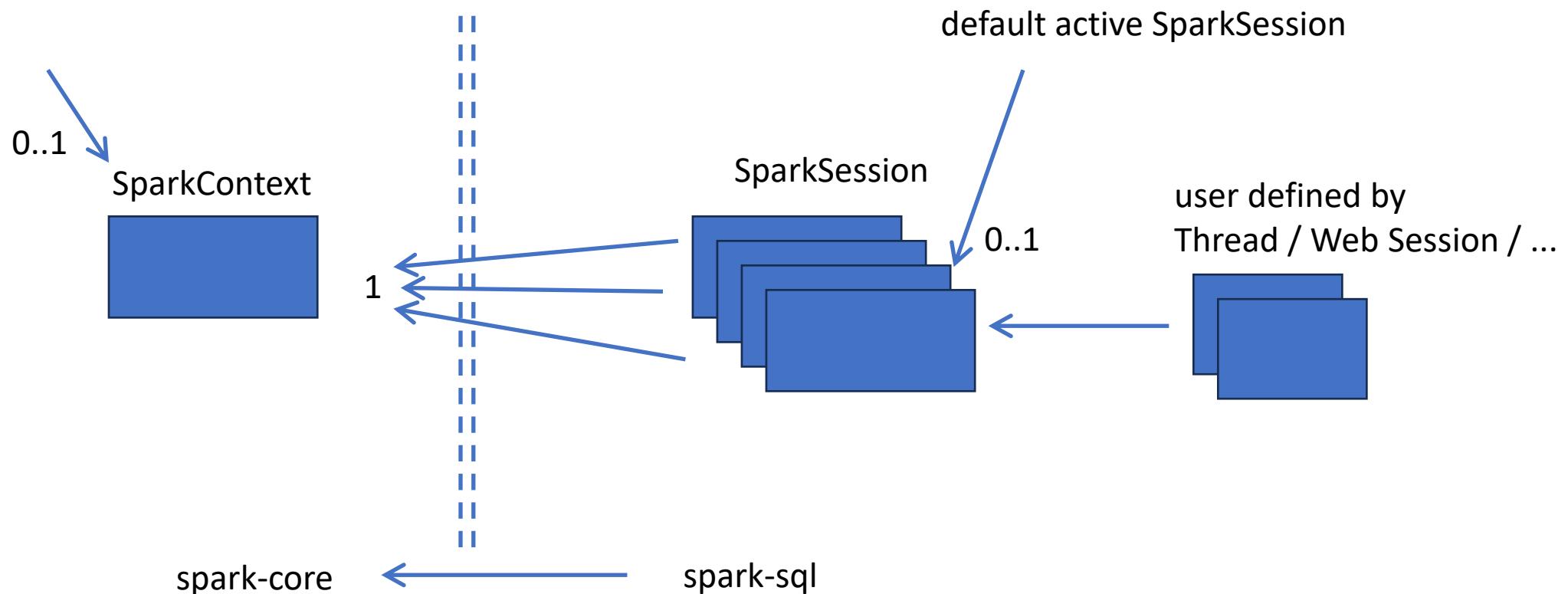
this.spark = SparkSession.builder()
    .config(sparkConf)
    .getOrCreate();

try {
    this.sc = spark.sparkContext();
    this.jsc = JavaSparkContext.fromSparkContext(sc);
    this.hadoopConf = jsc.hadoopConfiguration();
    this.hadoopFs = FileSystem.get(hadoopConf);
```

1 (singleton) SparkContext

1...N SparkSession

1 Active SparkSession



Spark-Java API with Lambda (ambiguous method, need cast)

```
FlatMapFunction<String, String> lineToWordIterorFunc = line -> {
    String[] res = line.replace("[,.?!]", " ").replace(" ", " ").split(" ");
    return Arrays.asList(res).iterator();
};
Dataset<String> wordDs = loremIpsum.flatMap(lineToWordIterorFunc, Encoders.STRING());
```

Java API ... need « Encoder » (implicit in Scala)

```
loremIpsum.flatMap(line -> line.replace("[,.?!]", " ").replace(" ", " ").split(" "));
```

The method flatMap(Function1<String, TraversableOnce<U>, Encoder<U>) in the type Dataset<String> is not applicable for the arguments ((<no type> line) -> {})



Press 'F2' for focus

Missing second param in java ..

```
/**  
 * (Scala-specific)  
 * Returns a new Dataset by first applying a function to all elements of this Dataset,  
 * and then flattening the results.  
 *  
 * @group typedrel  
 * @since 1.6.0  
 */  
def flatMap[U : Encoder](func: T => TraversableOnce[U]): Dataset[U] =  
  mapPartitions(_.flatMap(func))
```

Java API ... not convenient vs Spark-Scala API

```
loremIpsum.flatMap(line -> line.replace("[,.?!]", " ").replace(" ", " ").split(" ")),  
    Encoders.STRING());
```

Type mismatch: cannot convert from String[] to TraversableOnce<String>
Press 'F2' for focus

```
loremIpsum.flatMap(  
    line -> {  
        String res[] = line.replace("[,.?!]", " ").replace(" ", " ").split(" ");  
        return Arrays.asList(res).iterator();  
    },  
    Encoders.STRING());
```

Type mismatch: cannot convert from Iterator<String> to TraversableOnce<String>
2 quick fixes available:
 [Add cast to 'TraversableOnce<String>'](#)
 [Change method return type to 'Iterator<String>'](#)
Press 'F2' for focus

using scala.Function1 + inner class ... NotSerializableException !

The screenshot shows a code editor with Scala code and a terminal window below it.

```
45
46     scala.Function1<String, TraversableOnce<String>> lineToWordScalaFunc =
47         new AbstractFunction1<String, TraversableOnce<String>>() {
48             @Override
49             public TraversableOnce<String> apply(String line) {
50                 String[] res = line.replace("[,.?!]", " ").replace(" ", " ").split(" ");
51                 return JavaConverters.iteratorAsScalaIterableConverter(JavaConverters.iteratorAsScalaIterableConverter(Arrays.asList(res))).asScala();
52             }
53         };
54     Dataset<String> wordScalaDs = loremIpsum.flatMap(lineToWordScalaFunc, Encoders.STRING());
55     wordScalaDs.show();
56 }
```

Console × Problems Error Log Debug Shell Search Call Hierarchy

<terminated> SparkWordsCountAppMain [Java Application] C:\apps\jdk\jdk-8\bin\javaw.exe (6 janv. 2022 à 10:15:13 – 10:15:28)

```
10:15:22 INFO fr.an.tests.testspark.SparkWordsCountAppMain: line count:4
Exception in thread "main" org.apache.spark.SparkException: Job aborted due to stage failure: Task not serializable: java.io.NotSerializableException
Serialization stack:
 - object not serializable (class: fr.an.tests.testspark.SparkWordsCountAppMain$1, value: <function1>)
 - element of array (index: 0)
 - array (class [Ljava.lang.Object;, size 1)
 - field (class: java.lang.invoke.SerializedLambda, name: capturedArgs, type: class [Ljava.lang.Object;)
 - object (class java.lang.invoke.SerializedLambda, SerializedLambda[capturingClass=class org.apache.spark.sql.Dataset, functionalInterface=true])
 - writeReplace data (class: java.lang.invoke.SerializedLambda)
 - object (class org.apache.spark.sql.Dataset$$Lambda$2247/2630833, org.apache.spark.sql.Dataset$$Lambda$2247/2630833@18b07ae)
 - field (class: org.apache.spark.sql.execution.MapPartitionsExec, name: func, type: interface scala.Function1)
 - object (class org.apache.spark.sql.execution.MapPartitionsExec, MapPartitions org.apache.spark.sql.Dataset$$Lambda$2247/2630833@18b07ae,
```

Isolated top-level class implements Serializable

```
private static class LineToWordFunc extends AbstractFunction1<String, TraversableOnce<String>> implements Serializable {  
    private static final long serialVersionUID = 1L;  
    @Override  
    public TraversableOnce<String> apply(String line) {  
        String[] res = line.replace("[,.?!]", " ").replace(" ", " ").split(" ");  
        return JavaConverters.iteratorAsScalaIterableConverter(JavaConverters.asScalaIteratorConverter(Arrays.asList(res))).asScala();  
    }  
}  
  
private static void testSparkScalaApi(Dataset<String> loremIpsum) {  
    scala.Function1<String, TraversableOnce<String>> lineToWordScalaFunc = new LineToWordFunc(); // serializable sub-class  
    Dataset<String> wordScalaDs = loremIpsum.flatMap(lineToWordScalaFunc, Encoders.STRING());  
    wordScalaDs.show();  
}
```

Local[*] Debug from IDE

The screenshot shows a Java application named "SparkWordsCountAppMain" running in debug mode. The code is located in the file "Dataset.class". A breakpoint is set at line 46, which is part of the main method. The stack trace on the left shows numerous daemon threads and one main thread. The code itself is a word count application using Spark's Dataset API.

```
try {
    Dataset<String> loremIpsum = spark.read().textFile("data/loremIpsum.txt");
    Log.info("show() truncated");
    loremIpsum.show();

    long lineCount = loremIpsum.count();
    Log.info("line count:" + lineCount);

    FlatMapFunction<String, String> lineToWordIterorFunc = line -> {
        String[] res = line.replace("[,.?!]", " ").replace(" ", " ").split(" ");
        return Arrays.asList(res).iterator();
    };
    Dataset<String> wordDs = loremIpsum.flatMap(lineToWordIterorFunc, Encoders.defaultString());
    wordDs.cache();

    Log.info("words show() truncated");
    wordDs.show();

    long wordCount = wordDs.count();
    Log.info("words count:" + wordCount);
} finally {
    spark.stop();
}
```

The "Variables" window on the right displays the current state of variables:

Name	Value
no method return value	
args	String[0] (id=382)
sparkConf	SparkConf (id=384)
spark	SparkSession (id=392)
loremIpsum	Dataset<T> (id=396)
lineCount	4
lineToWordIterorFunc	9390914 (id=398)
wordDs	Dataset<T> (id=400)
wordCount	69

The "Console" tab at the bottom shows the output of the application, which includes the top 20 words from the lorem ipsum dataset.

```
ut
labore
et
dolore
magna
aliqua.
Ut
-----
only showing top 20 rows
```

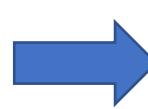
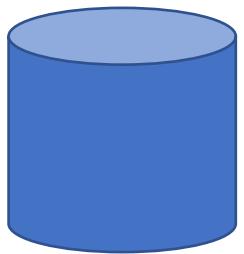
Advanced Sql / Dataset

```
private static void advancedSparkSqlCode(SparkSession spark) {
    Dataset<Row> ds = spark.sql(
        "SELECT latest.id, latest.col1, latest.col2, joined.col3"
        + " FROM ( SELECT *"
        + "     , row_number() OVER (PARTITION BY id ORDER BY timestamp DESC) as rankNum"
        + "     FROM some_hive_db.input_table"
        + "     WHERE rankNum=1"
        + " ) latest"
        + " LEFT JOIN department joined ON latest.deptno = joined.deptno"
    )
    // => convert Dataset<Row> -> Dataset<PojoBean>
    .as(Encoders.bean(PojoBean.class))
    // => compute enrich -> Dataset<PojoOutputBean>
    .map((MapFunction<PojoBean,PojoOutputBean>) (pojo -> computePojo2Output(pojo)), Encoders.bean(PojoOutputBean.class))
    // => convert -> Dataset<Row>
    .toDF();

    // optimize for read later
    ds // .repartition(1) .. equivalent to coalesce()
    // .orderBy("id")
    .repartition(2, ds.col("col1"))
    .sortWithinPartitions("id")
    // => write as PARQUET
    .write()
    .format("hive").mode(SaveMode.Overwrite)
    .insertInto("some_hive_db.output_table");
}
```

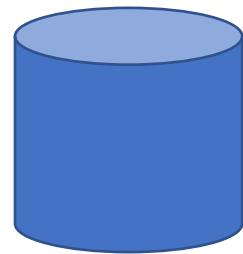
Typical Usage: process RAW to LAKE

/RAW
/table/partition
/ file{1,2,3,4*}.avro



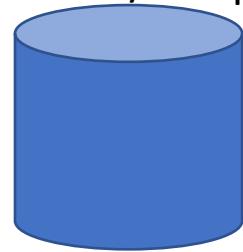
Spark daily job RAW to LAKE

/LAKE
/table/partition
/ file.parquet



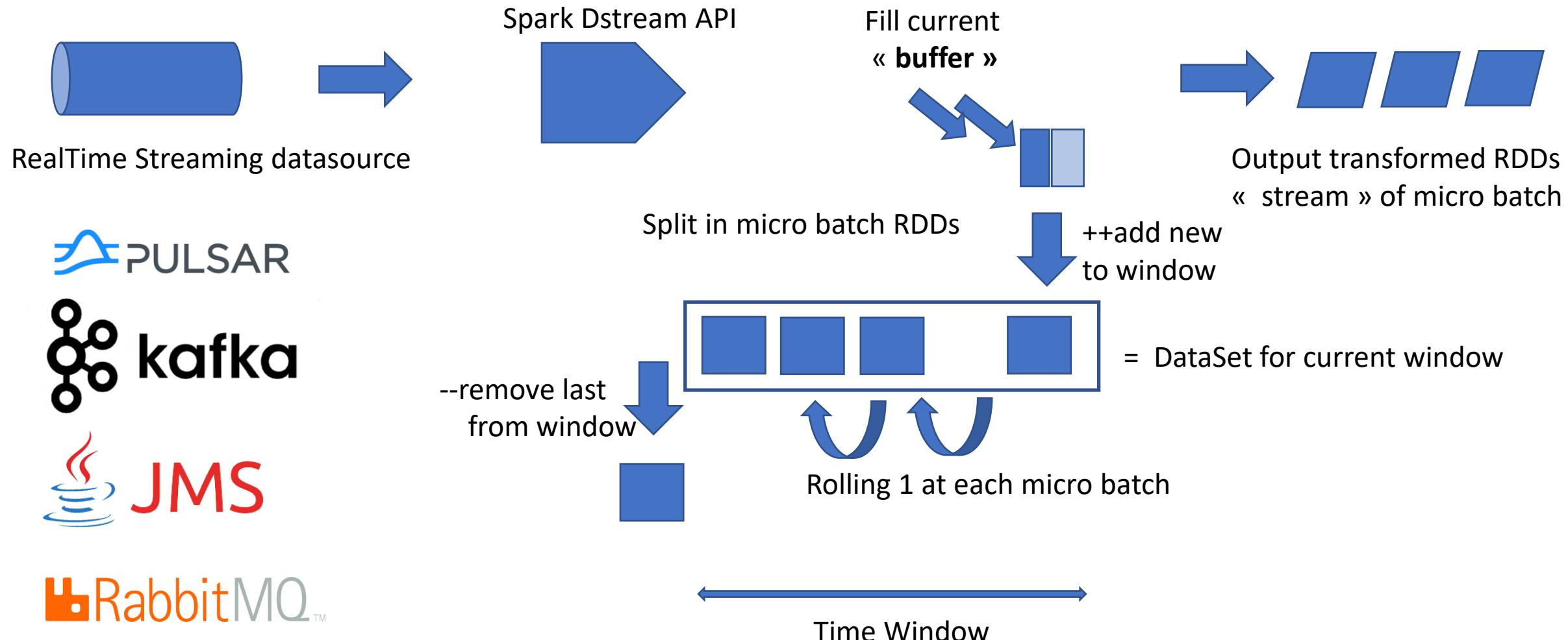
```
spark.sql(" select * from raw_table where day=... join .. ..")  
.map(row => ...)  
.orderBy(..)  
.write.format("hive").mode(SaveMode.Overwrite)  
.sortWithinPartition(..)  
.insertInto("lake_table");
```

/LAKE
/denorm-table/another-partition
/ file.parquet



Spark Streaming : micro Batches

Spark DStream API ... as + enriched DataSet API



DStream using similar to Dataset API

all the API in Dataset are similar on DStream

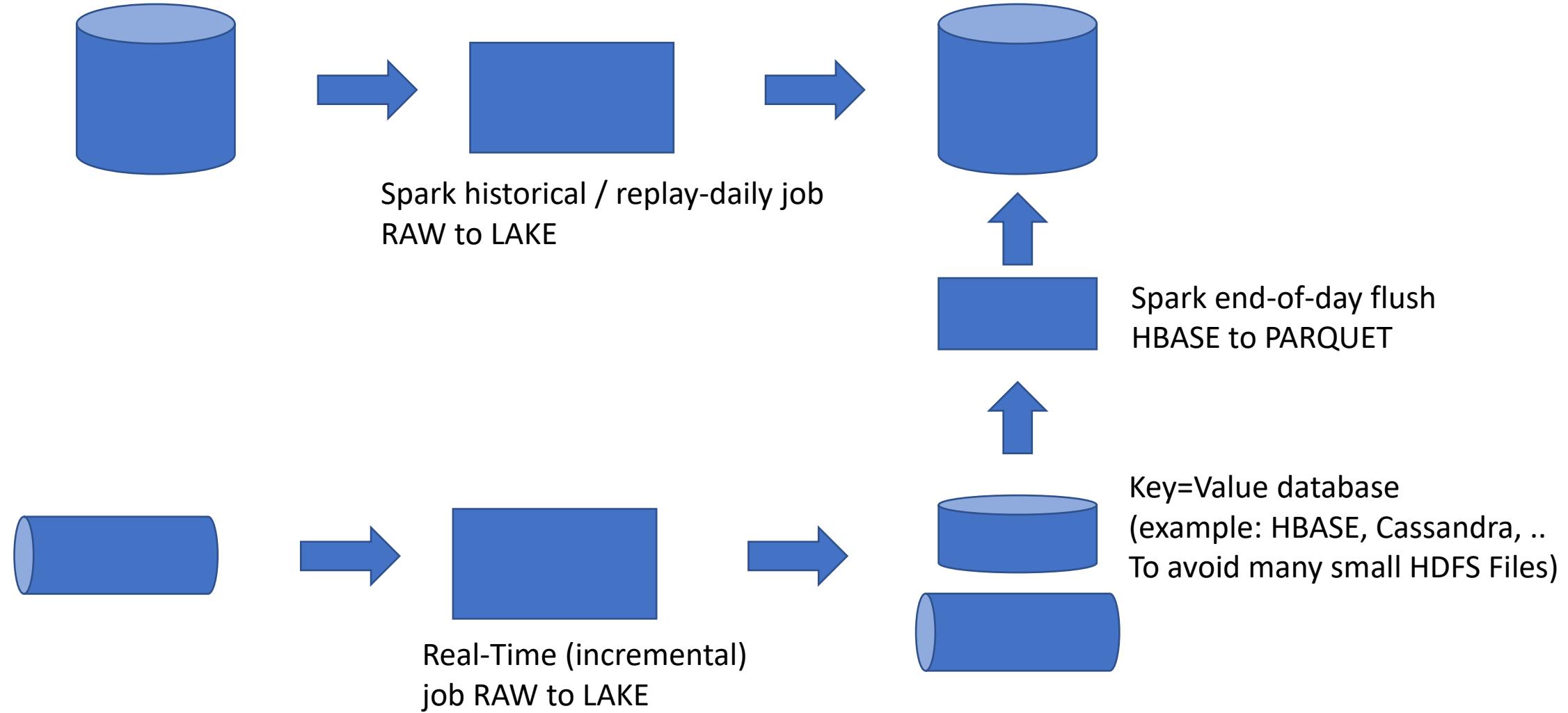
example:

```
DStream sourceDstream = ...
DStream transformedDstream = sourceDstream.filter(..).map(..).repartition(..);
```

is similar to

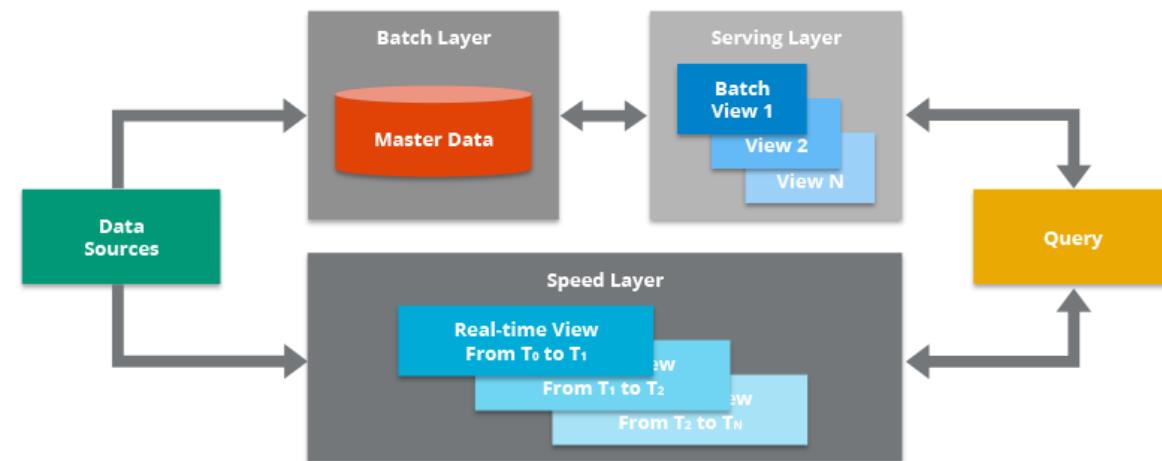
```
Dataset nextDs_5s = ... sourceDstream. <<next microbatch dataset>>
Dataset transformedDs = nextDs_5s.filter(..).map(..).repartition(..);
```

Typical Usage ... Streaming vs Daily

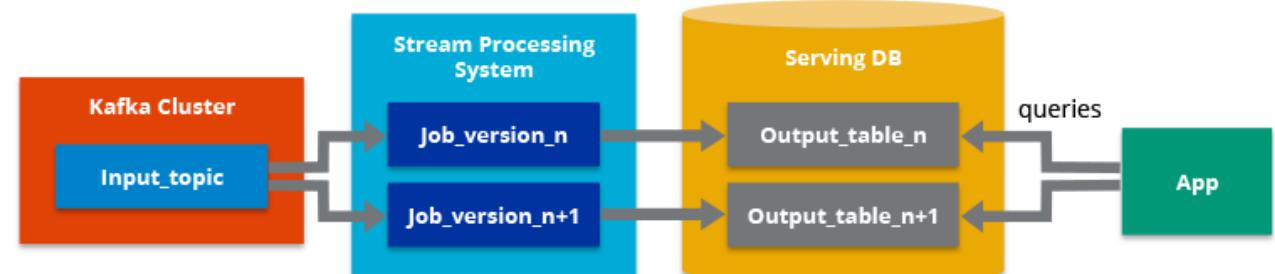


Typical Architectures.. Lambda vs Kappa

Lambda



Kappa



Conclusion & Questions