arnaud.nauwynck@gmail.com

Presentation BigData part 5: Spark – RDD - SQL

This document:
https://github.com/Arnaud-Nauwynck/presentations/blob/main/pres-bigdata/BigData-5-spark-rdd-sql.pdf

Spark (Recent) History & Ancestors

MapReduce @ Google

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

MPI

1995 Message Passing Interface

APACHE Spark™

2010 Spark paper

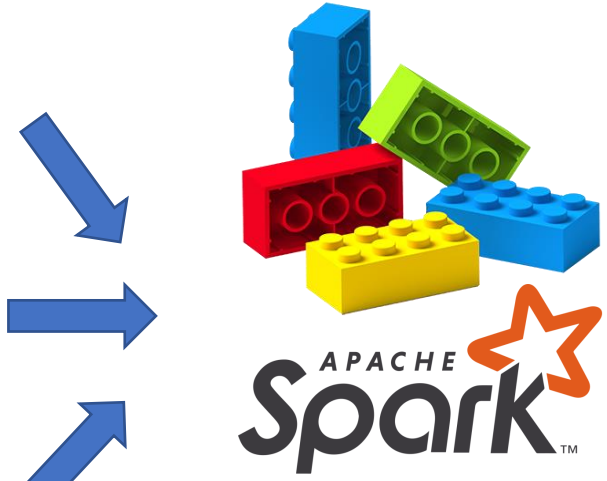
2013 Apache top-level

??

2015
Kubernetes

2020
Spark on K8s

Simple => Many Specific Systems => Unified

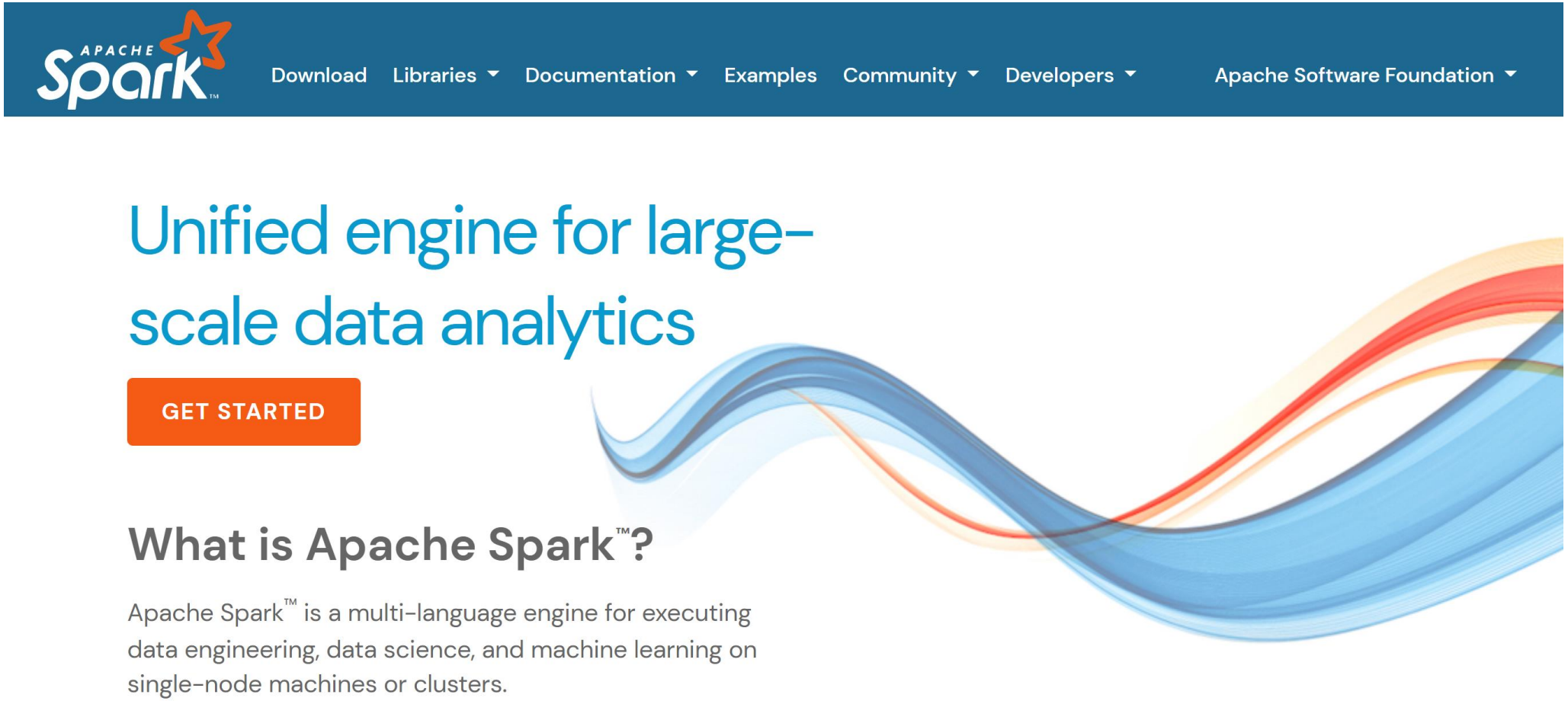


« **Simple** » ecosystem
(verbose inneficient & complex java code)

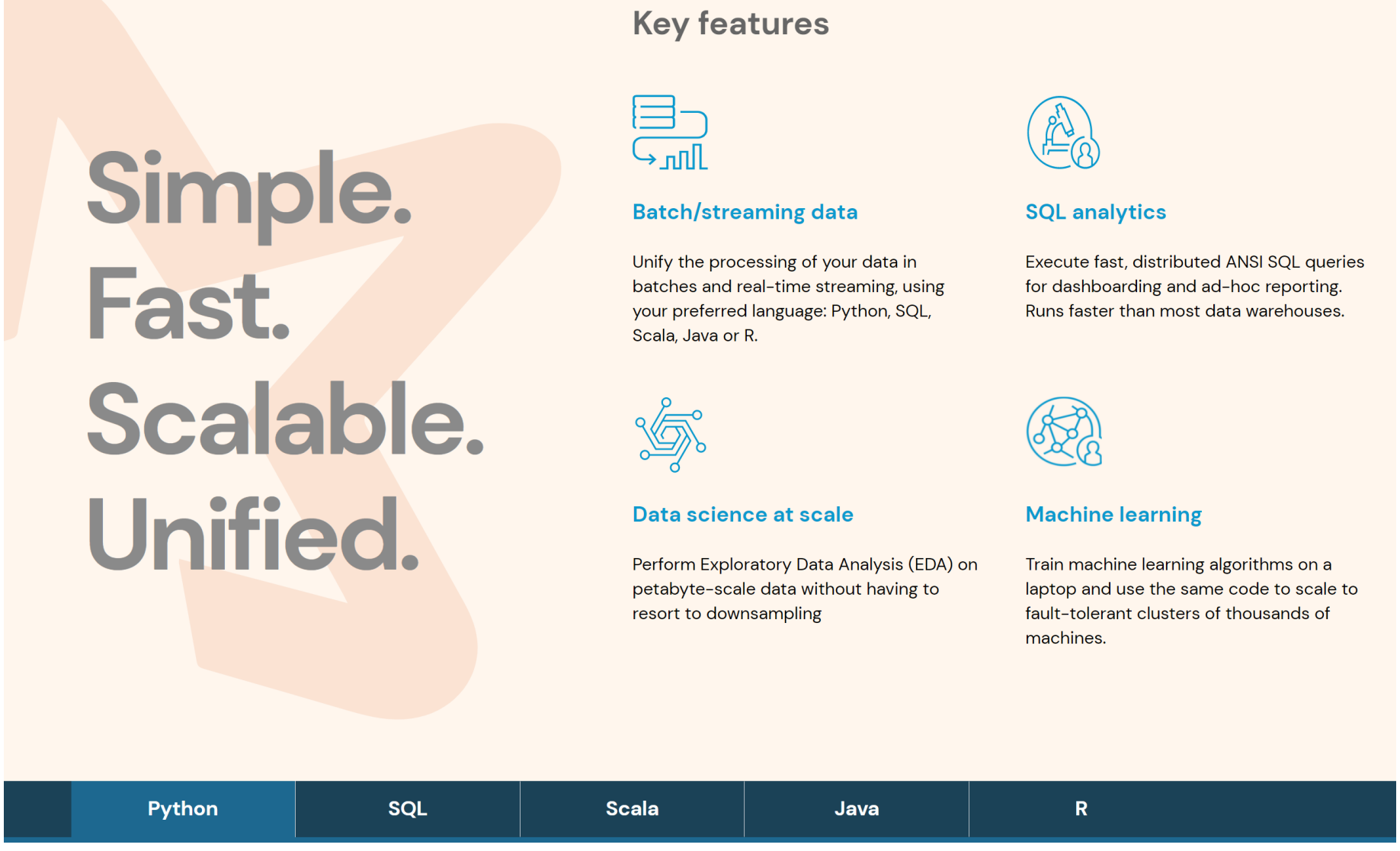
« **Bazard** » ecosystem
(**Too MANY TOO** SPECIFIC redundant, complexes)

“Unified” ecosystem
Simple
+ extensible modules

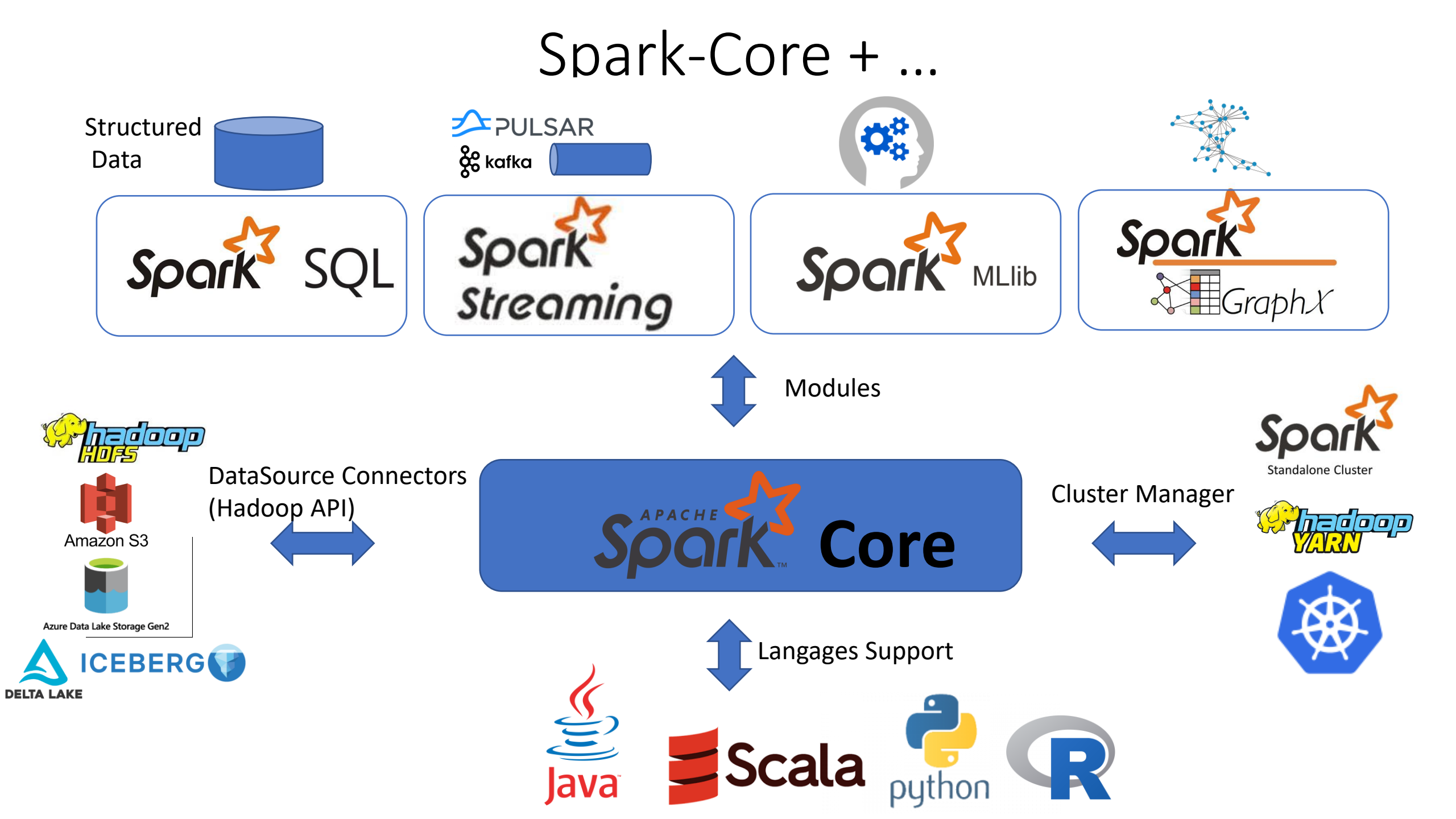
Spark = « **Unified Engine** »



Multi Purposes – Multi Langages

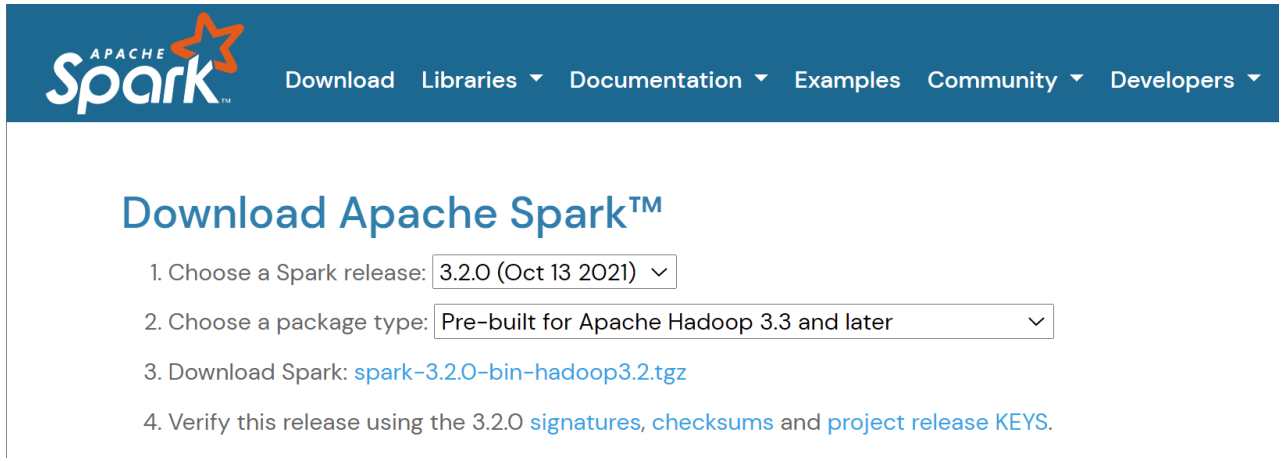


Spark-Core + ...


Structured Data
PULSAR
kafka
Spark SQL
Spark Streaming
Spark MLlib
Spark GraphX
Modules
hadoop HDFS
Amazon S3
Azure Data Lake Storage Gen2
ICEBERG
DELTA LAKE
DataSource Connectors (Hadoop API)
APACHE Spark Core
Cluster Manager
Spark Standalone Cluster
hadoop YARN
Langages Support
Java
Scala
python
R

Getting Started

1/ Download



2/ Unzip + add to PATH

3/ launch

C:> bin\spark-shell

(or spark-submit, or spark-sql)

```
C:\apps\cmder
λ spark-shell
Spark context available as 'sc' (master = local[*], app id = local-16
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.1.1
      /_/

Using Scala version 2.12.10 (OpenJDK Client VM, Java 1.8.0_282)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

spark-shell> println(« Hello World »);

```
C:\apps\cmder
λ spark-shell
21/12/31 19:21:03 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform...
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://DESKTOP-2EGCC8R:4040
Spark context available as 'sc' (master = local[*], app id = local-1640974873670).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.1.1
      /_/

Using Scala version 2.12.10 (OpenJDK Client VM, Java 1.8.0_282)
Type in expressions to have them evaluated.
Type :help for more information.

scala> print("Hello World")
Hello World
scala>
```

spark-shell> help

```
scala> :help
All commands can be abbreviated, e.g., :he instead of :help.
:completions <string>    output completions for the given string
:edit <id>|<line>        edit history
:help [command]          print this summary or command-specific help
:history [num]           show the history (optional num is commands to show)
:h? <string>             search the history
: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 interpreter
: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

scala>
```

Spark-shell> SCALA code

:paste

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

for(i <- 0 to 10) {
 println(s"hello ${i}")
}
```

Ctrl-D

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

hello 0
hello 1
hello 2
hello 3
hello 4
hello 5
hello 6
hello 7
hello 8
hello 9
hello 10

scala>
```

« 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
```

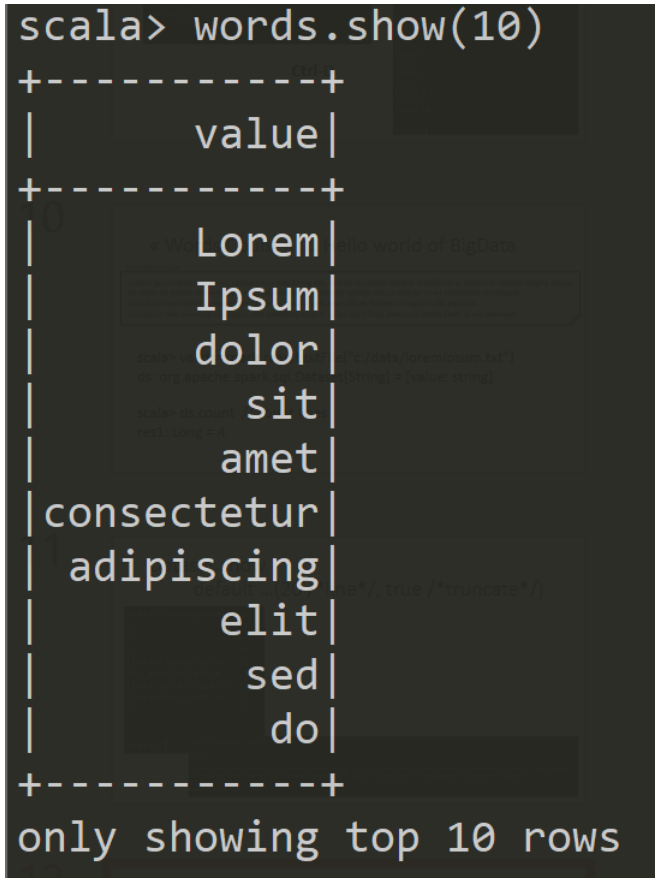
Dataset.show
default show(20 /*line*/, true /*truncate*/)

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


scala>
```

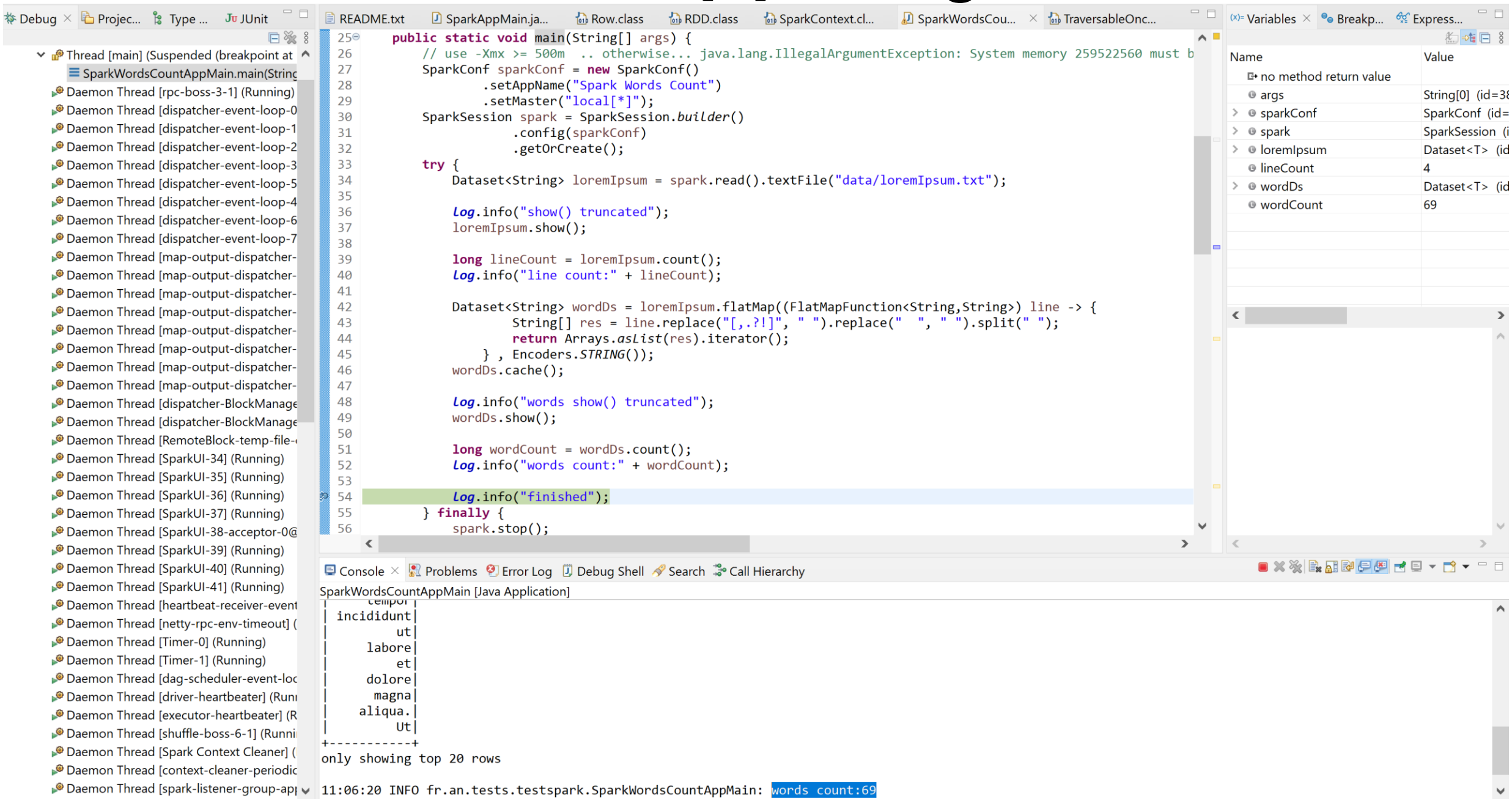
```
scala> loremIpsum.show(2, false)
+-----------------------------------------------------------------------------------------------------------------------------------------+
|value                                                                                                                                    |
+-----------------------------------------------------------------------------------------------------------------------------------------+
|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
```

Words Count : flatMap(…).count

```
scala> val words = loremIpsum.flatMap(line =>
        line.replaceAll("[,;.:!?]", " ")
                .replaceAll("  ", " ")
                .split(" ")
        )
words: .. Dataset[String] = [value: string]
scala> words.show(10, false)

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



Words Count … local[*] Debug in Java IDE



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

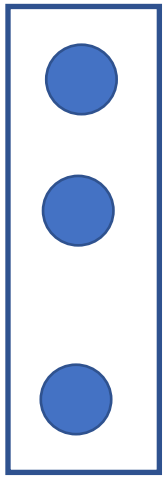
```
c:\arn\devPerso\test-snippets\test-spark (master -> origin)
λ spark-submit --class fr.an.tests.testspark.SparkWordsCountAppMain target/tests-spark-0.0.1-SNAPSHOT.jar
22/01/06 11:14:11 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes wh
ere 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
```

… skipped 1000 lines LOG …

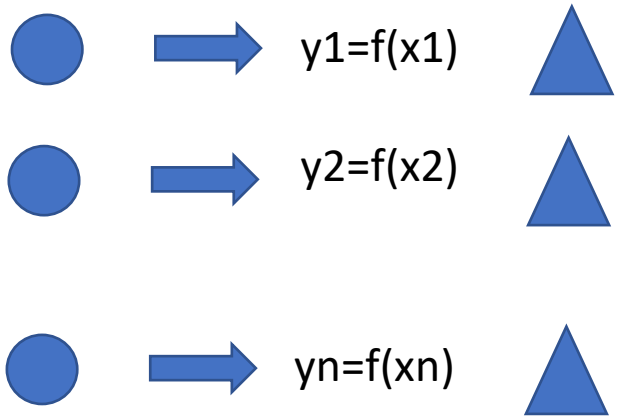
```
22/01/06 11:14:22 INFO DAGScheduler: Submitting ResultStage 5 (MapPartitionsRDD[28] at count at SparkWordsCountAppMain.java:51),
 which has no missing parents
22/01/06 11:14:22 INFO MemoryStore: Block broadcast_8 stored as values in memory (estimated size 10.1 KiB, free 413.6 MiB)
22/01/06 11:14:22 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 SparkWordsC
ountAppMain.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, NO
DE_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: 41
3.9 MiB)
22/01/06 11:14:22 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
```

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

Source Dataset = List<X>

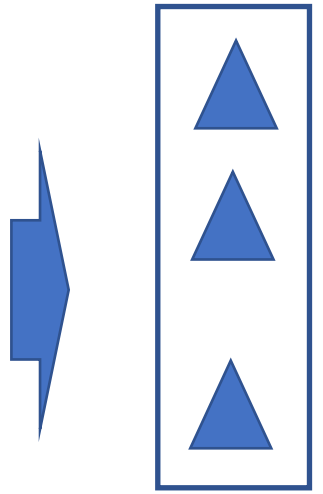


Mapping function (or lambda) :
x -> …. return y



dataset.map(func)

Result = List< Y>



« Equivalent » imperative code :

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

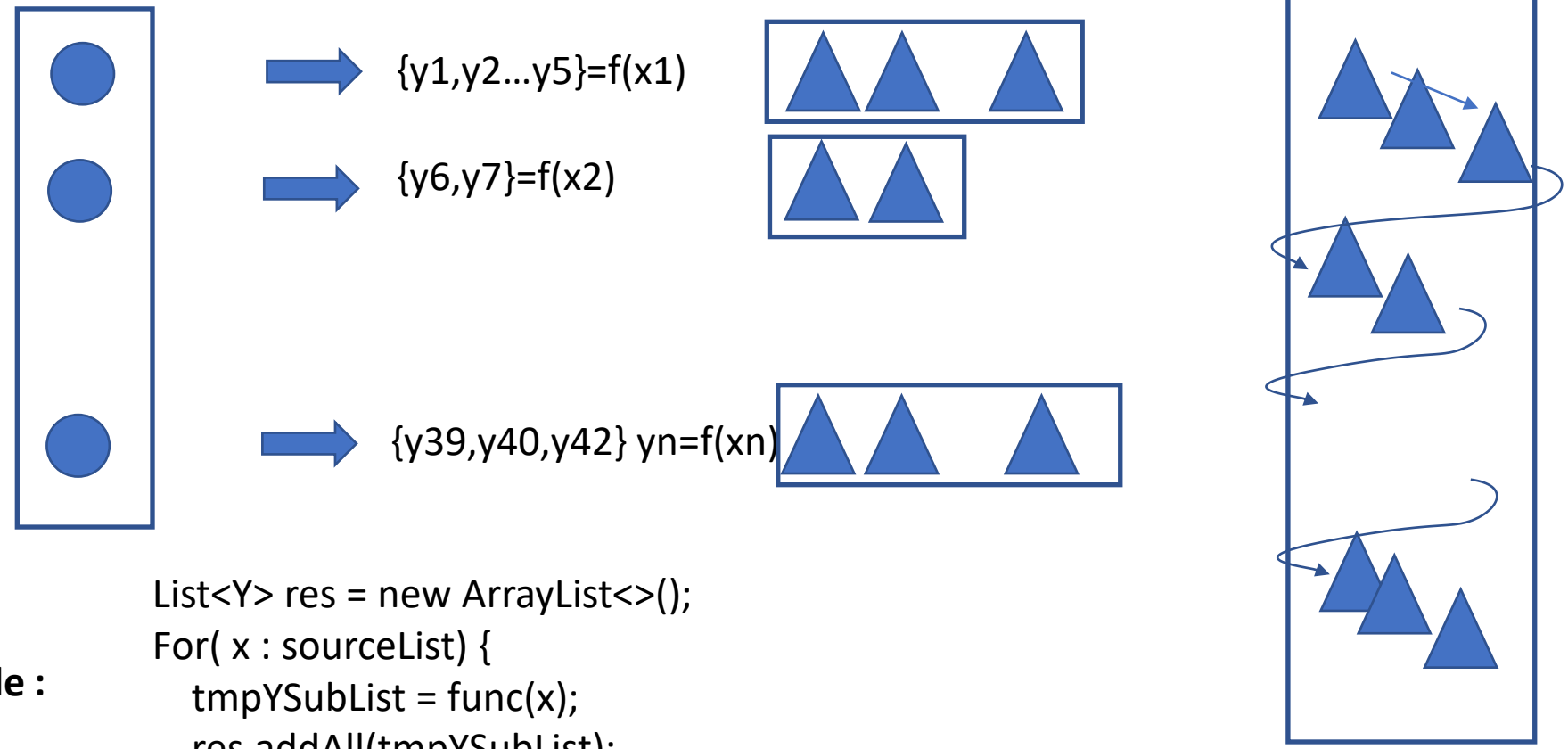
resultDs = dataset.flatMap(x -> {... return list<y>; })

Mapping function (or lambda) :
x -> …. return y

dataset.flatMap(func)

Source
Dataset = List<X>

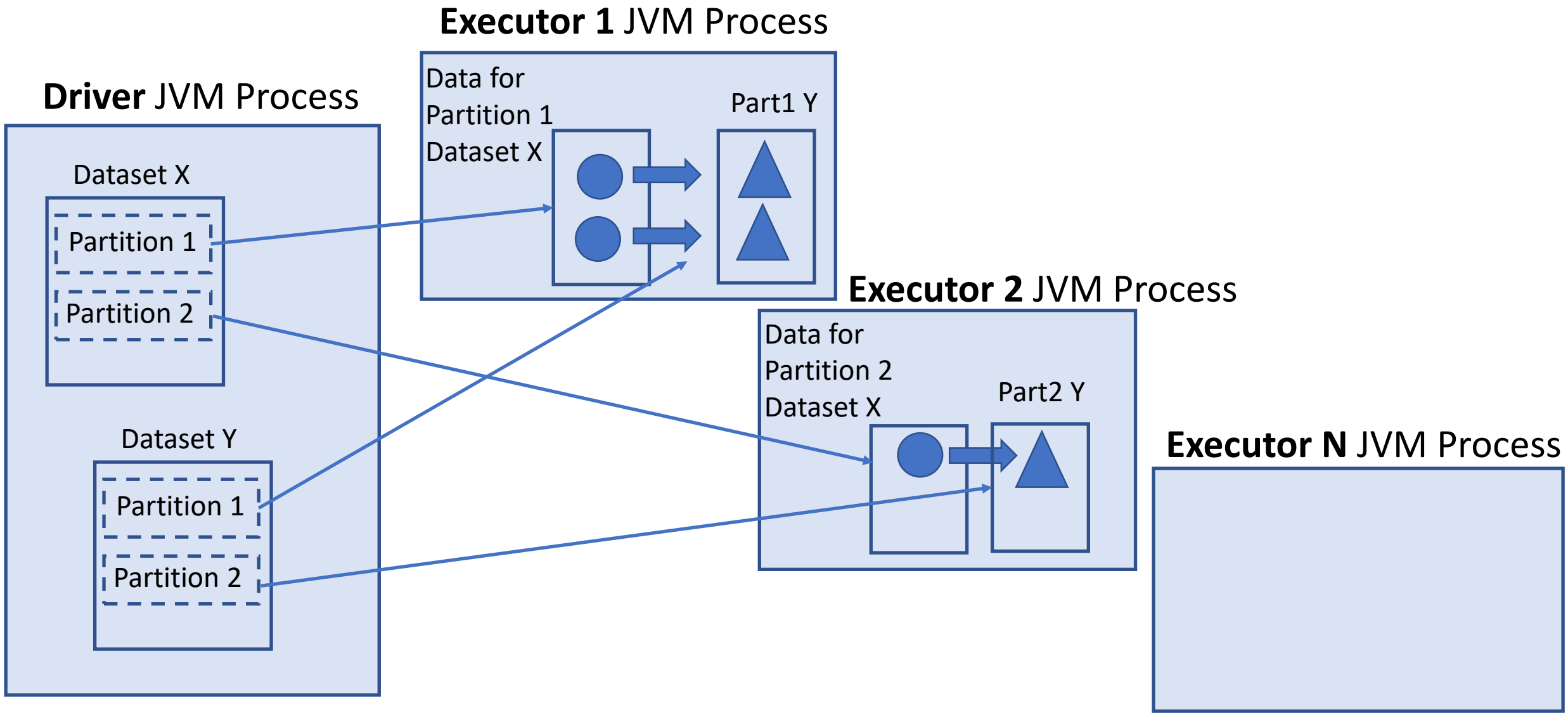
Result Dataset = List< Y>



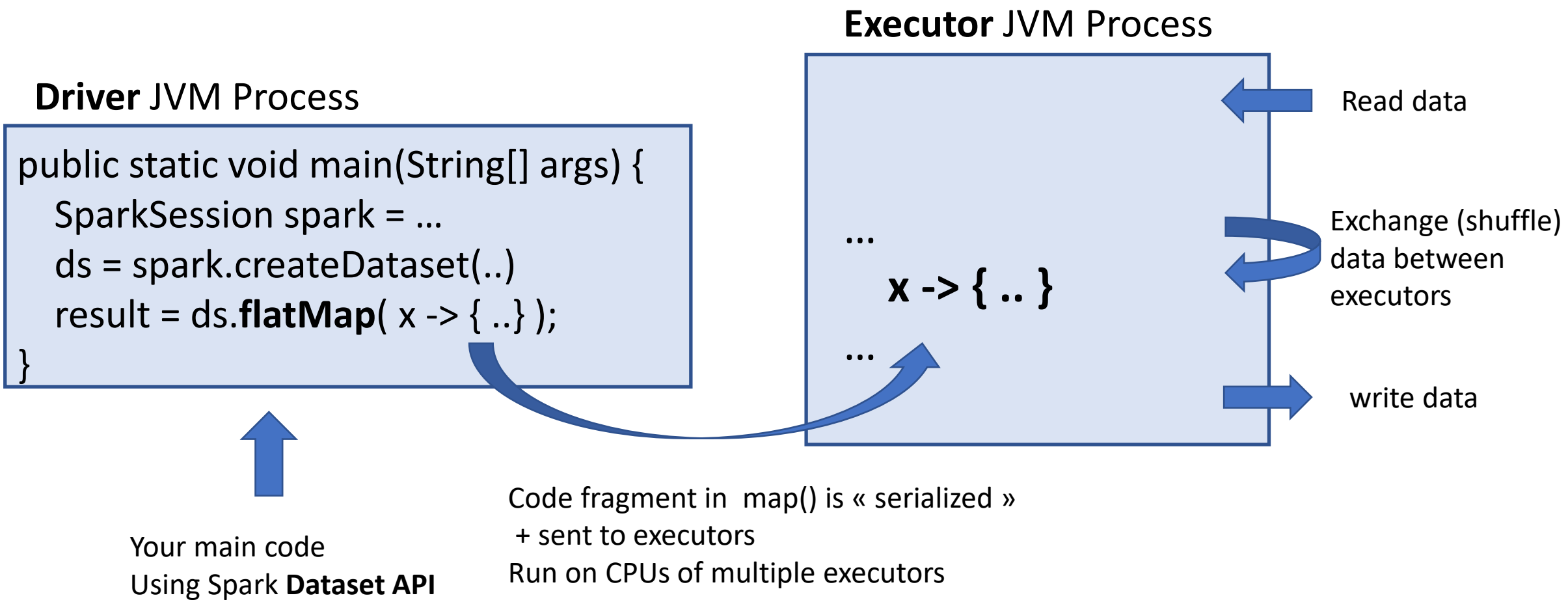
« Equivalent » imperative code :

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

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)
```

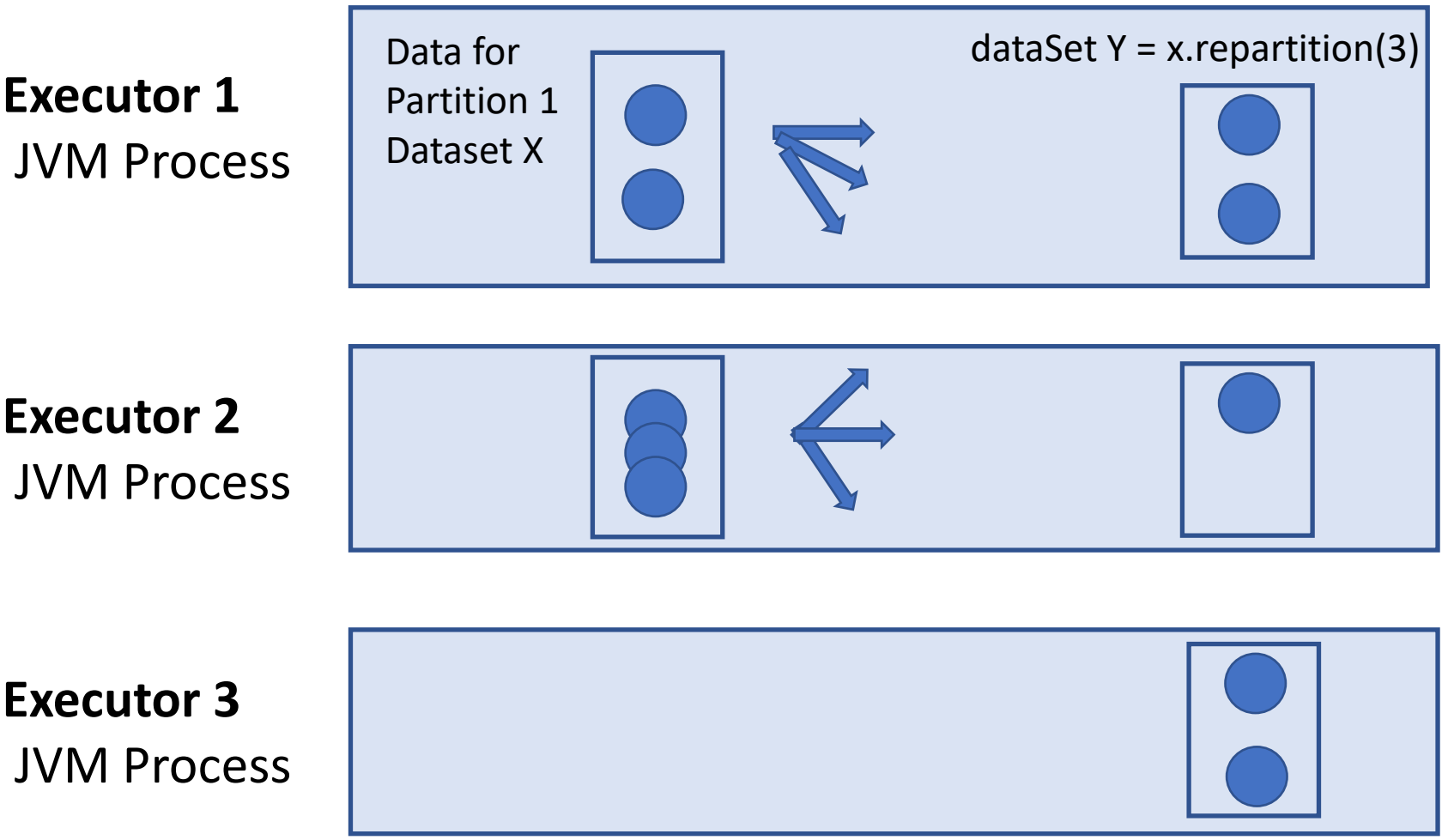
Low-level API
using « **RDD** »

```
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

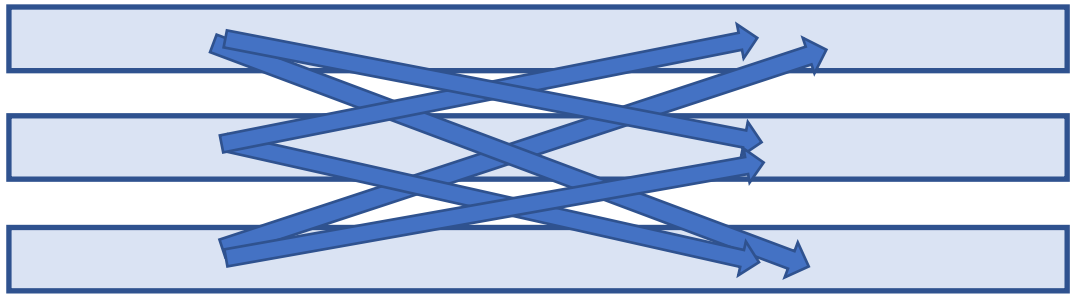


Wide vs Narrow Transformation

Wide Transformation (Exchange) between

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

Executor 1
Executor 2
Executor 3

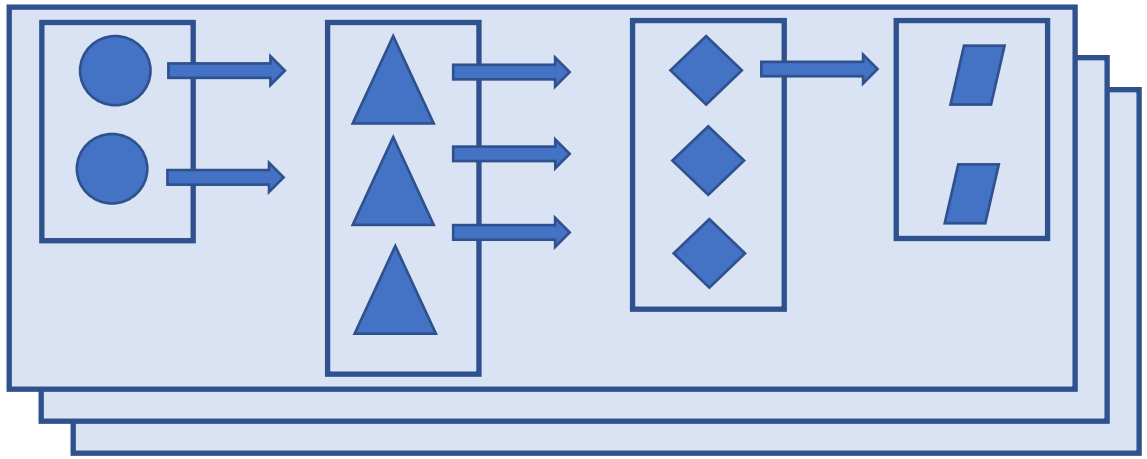


Network exchange
Bewteen executors
… serialization/deserialization of byte data

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
```

```
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<valu
```

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

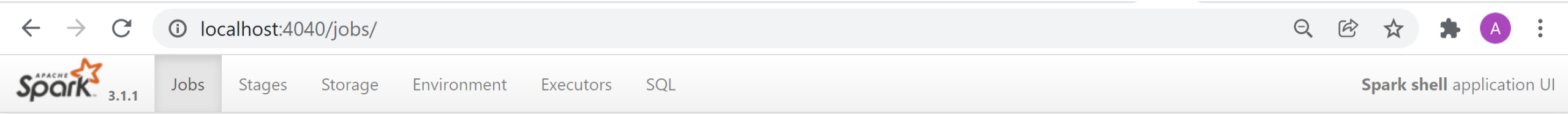
The architecture of deployment

How to program/package/deploy java code

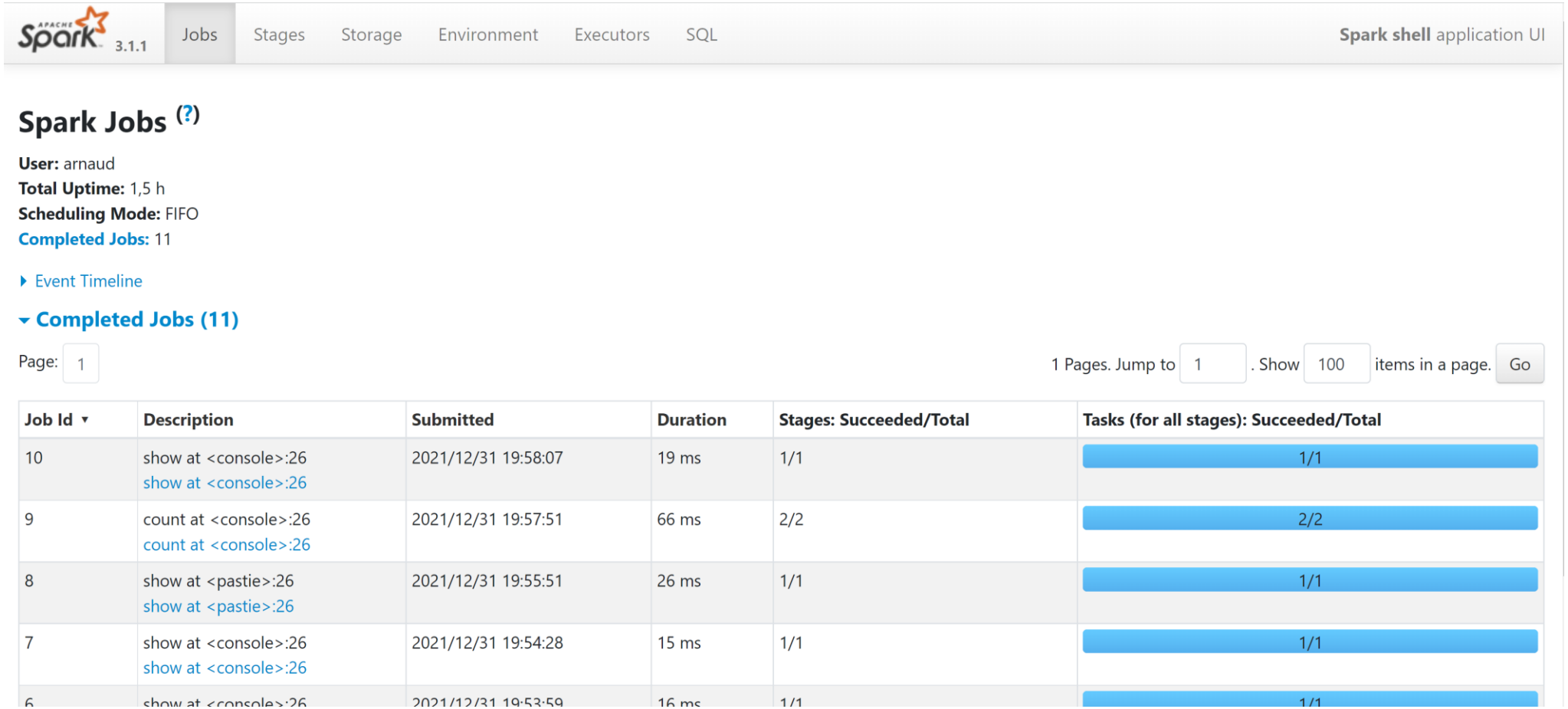
Spark-ui

```
Spark context Web UI available at http://DESKTOP-2EGCC8R:4040
Spark context available as 'sc' (master = local[*], app id = l
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.1.1
      /_/
```

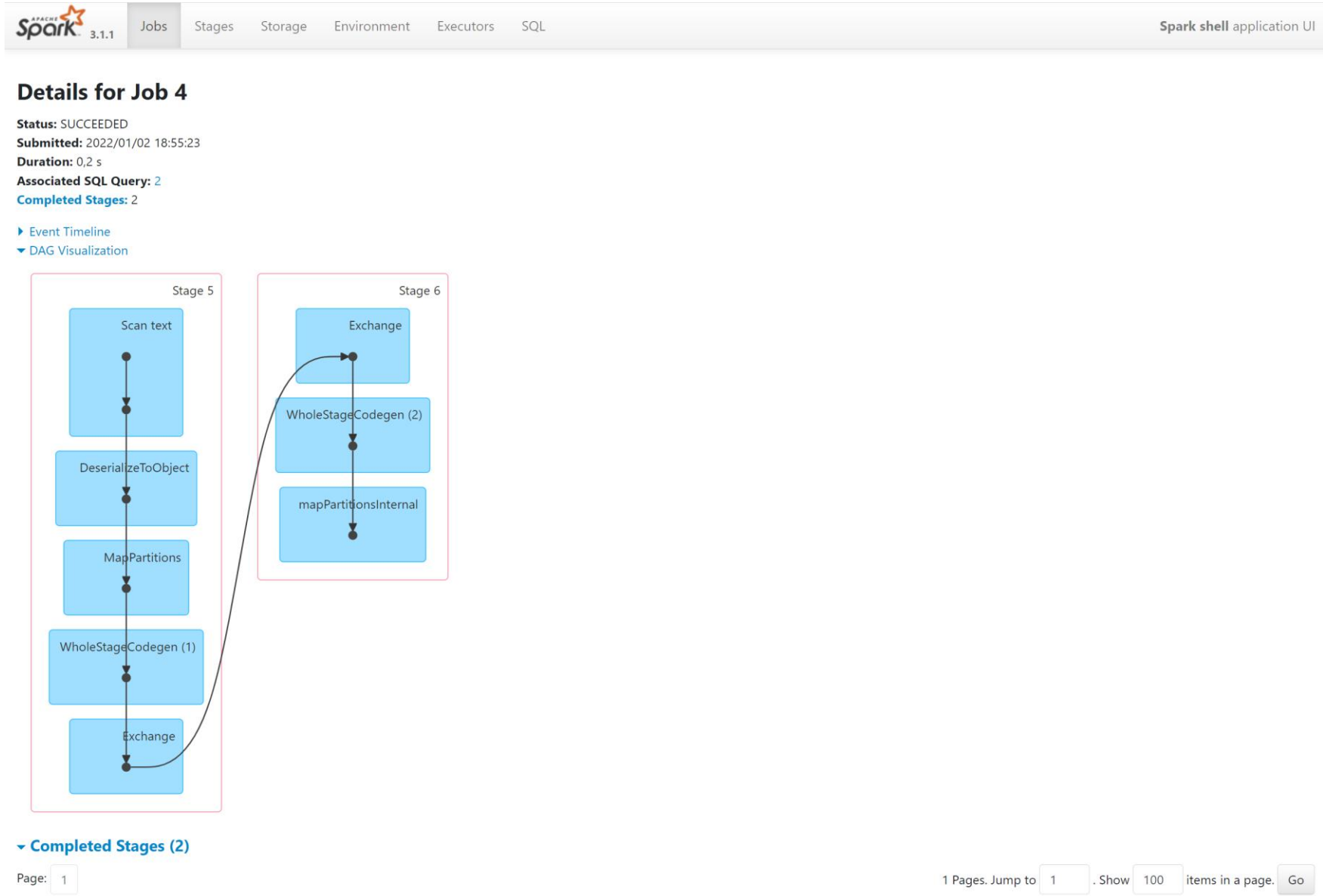
http://localhost:4040



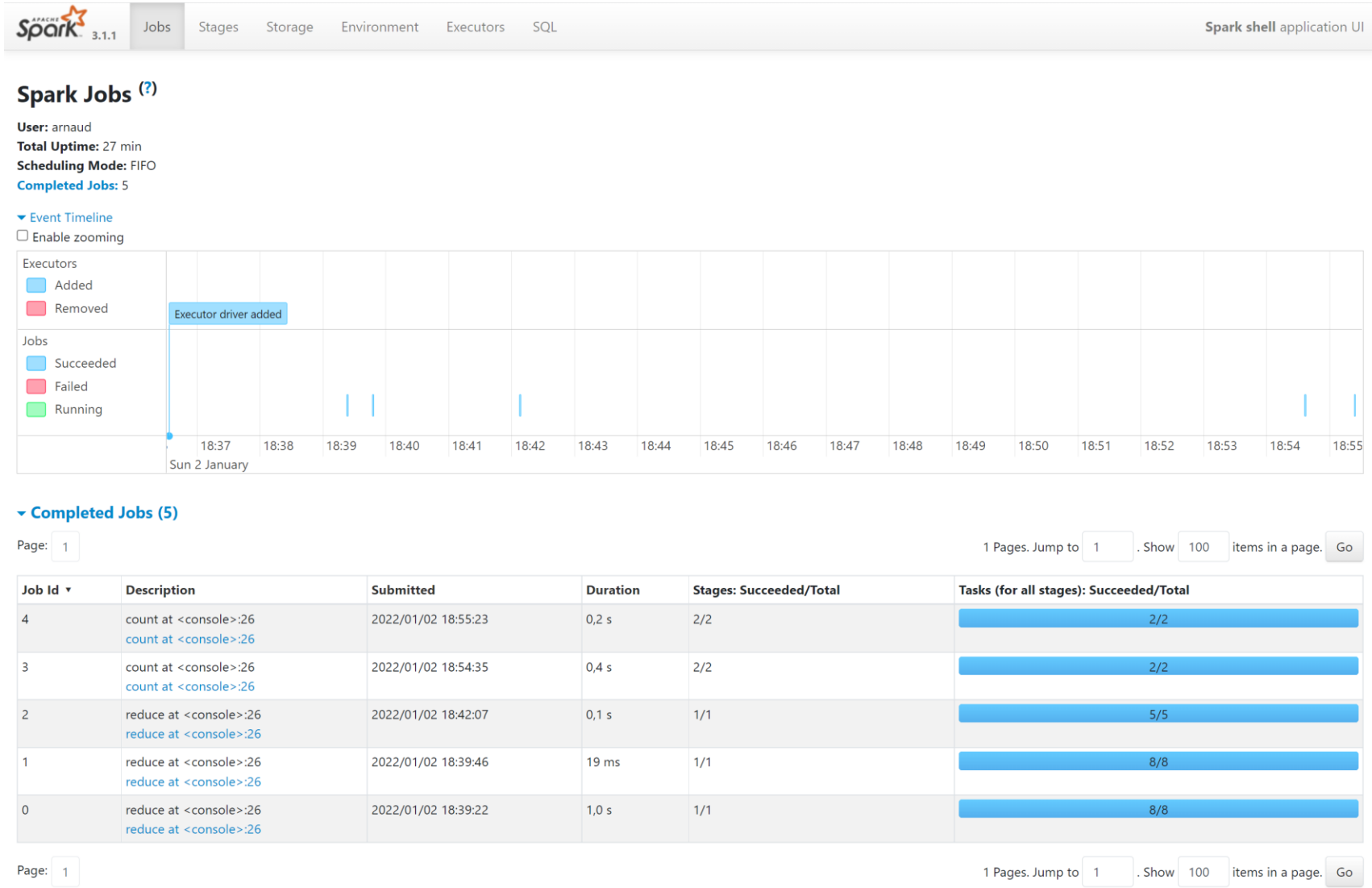
spark-ui [1/6]: Jobs



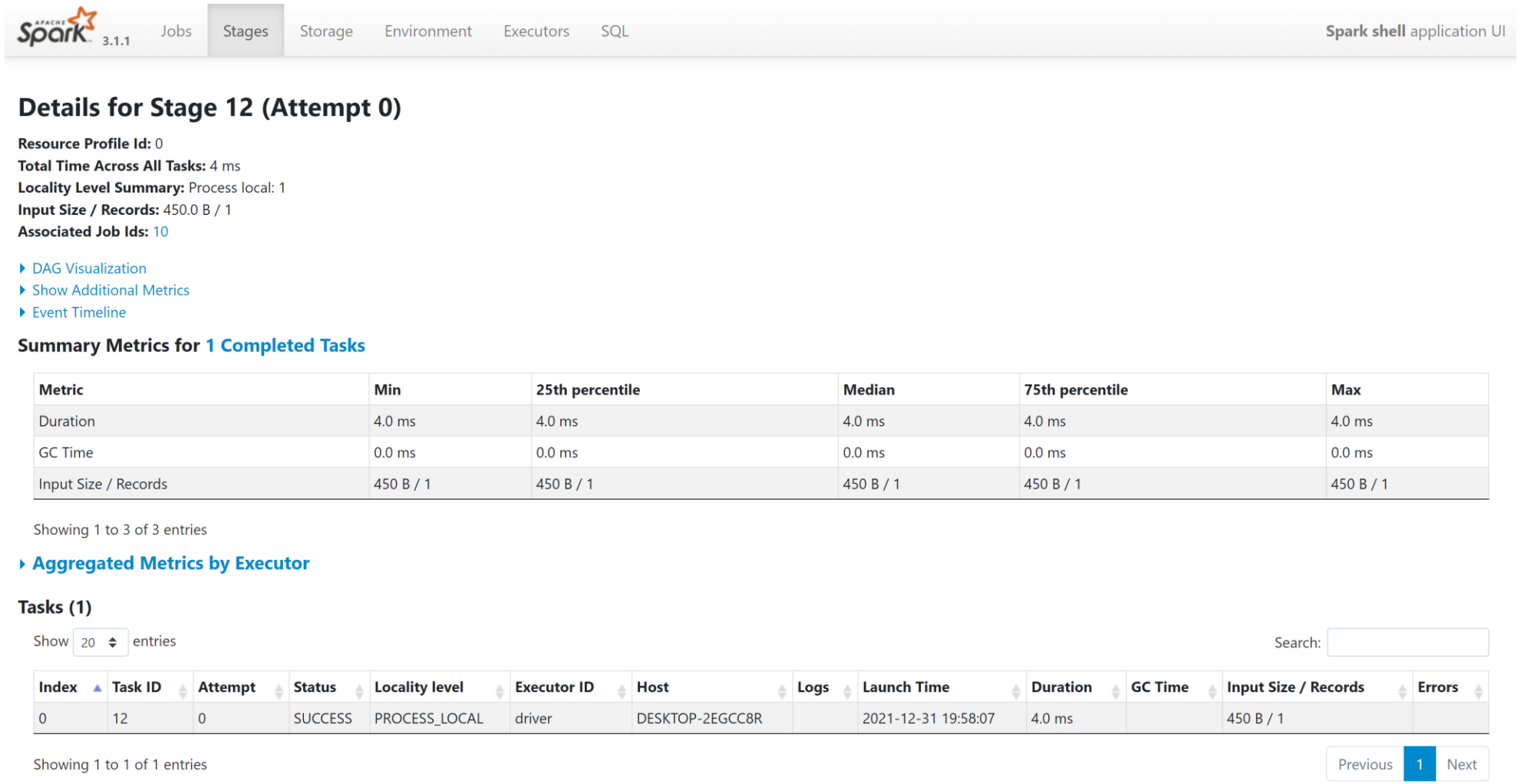
Spark-ui [1/6]: Job details DAG



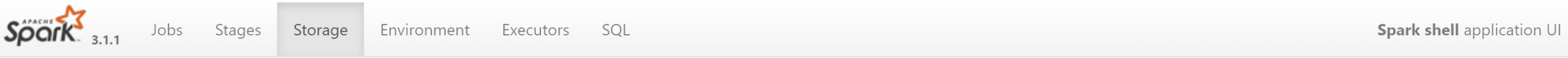
Spark-ui [1/6] detail Job timeline + statistics



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



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

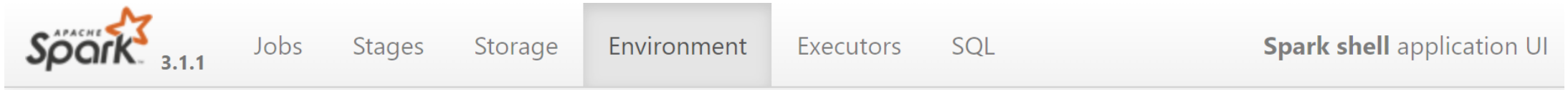


Storage

▾ RDDs

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/loremIpsum.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/loremIpsum.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+..



Environment

- Runtime Information
- Spark Properties
- Resource Profiles
- Hadoop Properties
- System Properties
- Classpath Entries

▾ 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

Spark 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

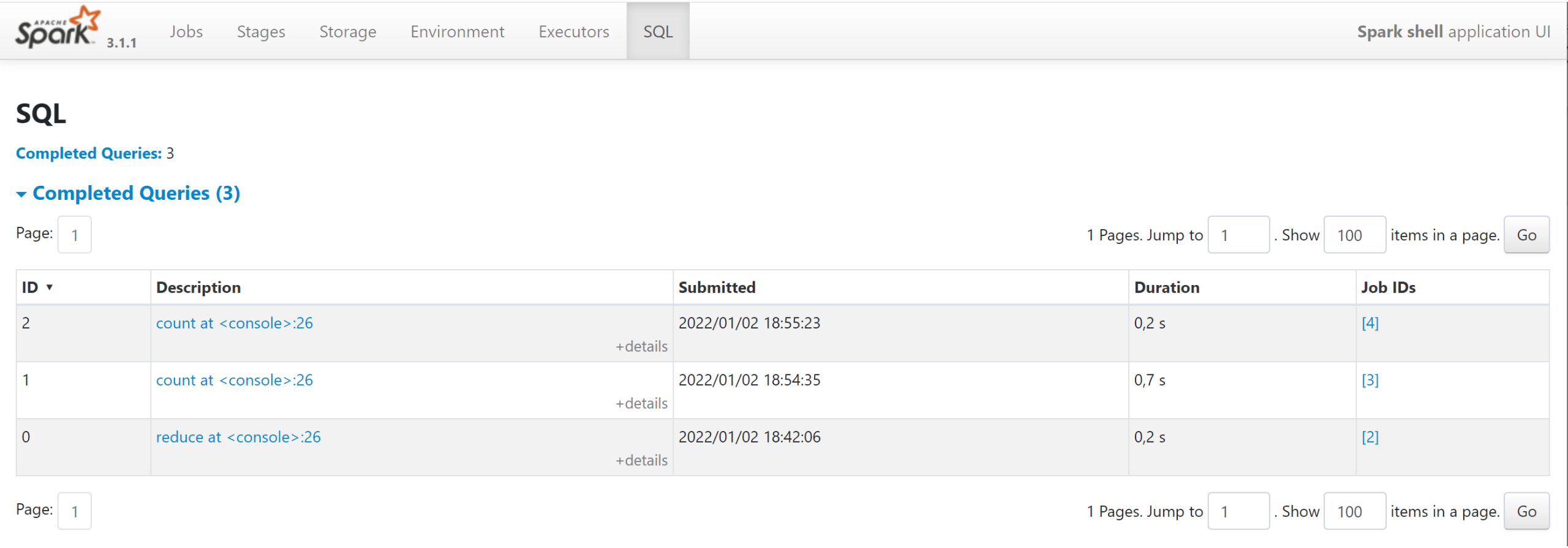
Executors

Show 20 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 Previous 1 Next

Spark-ui [6/6] SQL



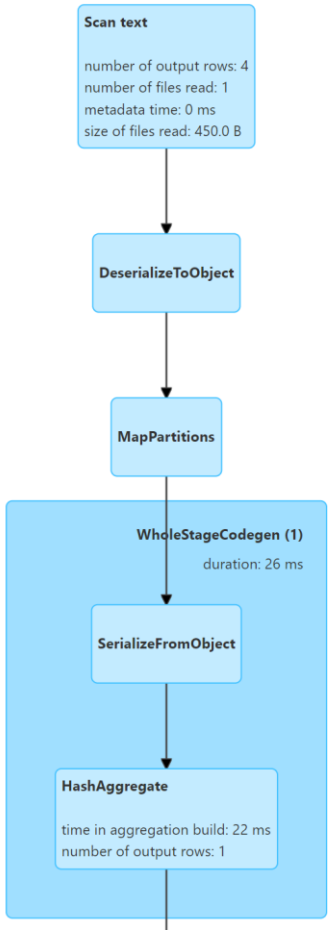
Spark-ui [6/6] Sql detail for query

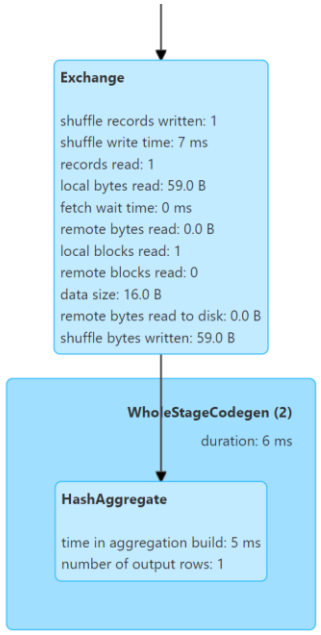
Spark 3.1.1 | Jobs | Stages | Storage | Environment | Executors | SQL

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



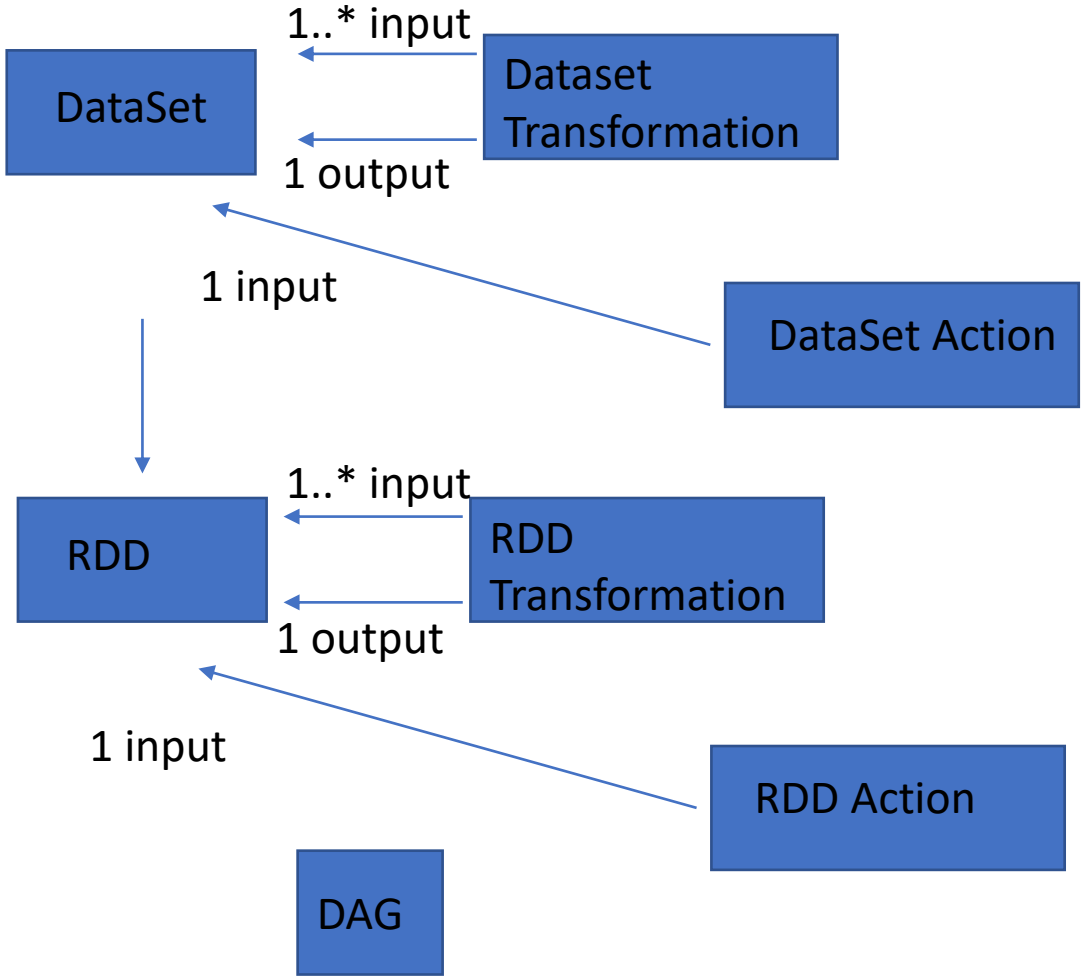


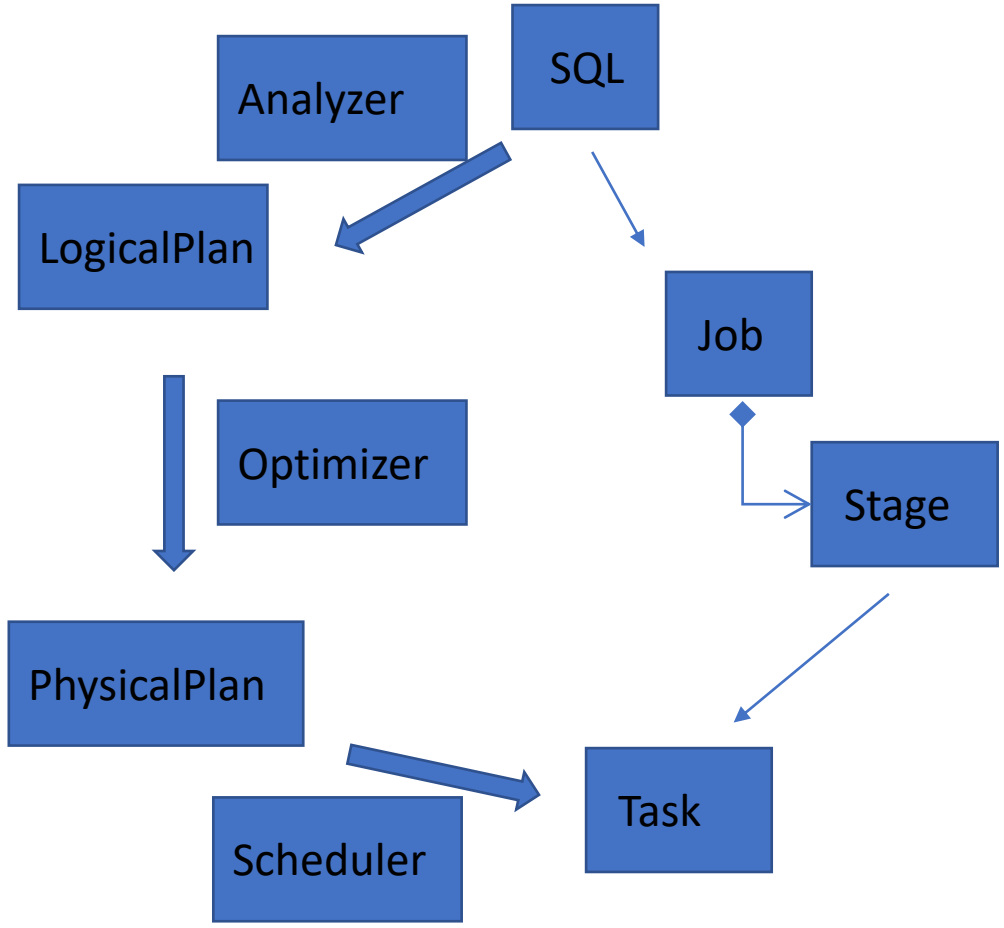
▾ Details

```
== Physical Plan ==
* HashAggregate (7)
+- Exchange (6)
   +- * HashAggregate (5)
      +- * SerializeFromObject (4)
         +- MapPartitions (3)
            +- DeserializeToObject (2)
               +- Scan text  (1)


(1) Scan text
Output [1]: [value#21]
Batched: false
Location: InMemoryFileIndex [file:/c:/data/loremIpsum.txt]
ReadSchema: struct<value:string>

(2) DeserializeToObject
Input [1]: [value#21]
Arguments: value#21.toString, obj#28: java.lang.String

(3) MapPartitions
Input [1]: [obj#28]
Arguments: org.apache.spark.sql.Dataset$$Lambda$4450/17435982@68a3d4, obj#29: java.lang.String
```

```
(4) SerializeFromObject [codegen id : 1]
Input [1]: [obj#29]

(5) HashAggregate [codegen id : 1]
Input: []
Keys: []
Functions [1]: [partial_count(1)]
Aggregate Attributes [1]: [count#35L]
Results [1]: [count#36L]

(6) Exchange
Input [1]: [count#36L]
Arguments: SinglePartition, ENSURE_REQUIREMENTS, [id=#55]

(7) HashAggregate [codegen id : 2]
Input [1]: [count#36L]
Keys: []
Functions [1]: [count(1)]
Aggregate Attributes [1]: [count(1)#32L]
Results [1]: [count(1)#32L AS count#33L]
```

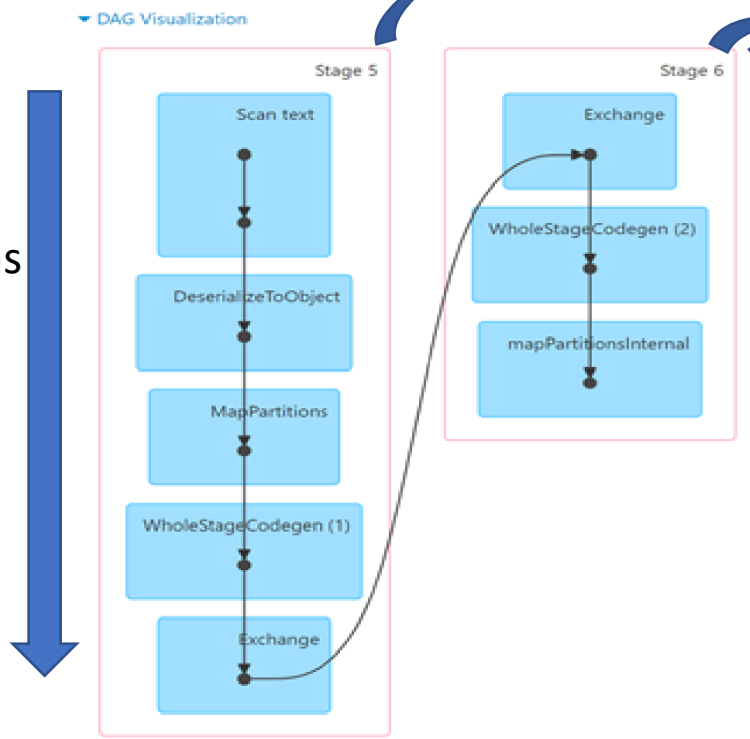
Spark Internal Concepts…
SQL, Job, Stage, Task





Job-Stage-Task in DAG

Task for stage



Task attempt 1 for stage
…May fail

Task retry attempt 2 for stage…

retry up to 4 times

Restart from cached in-memory/disk result of prev stage

1 stage =
Successive **Narrow** transformations

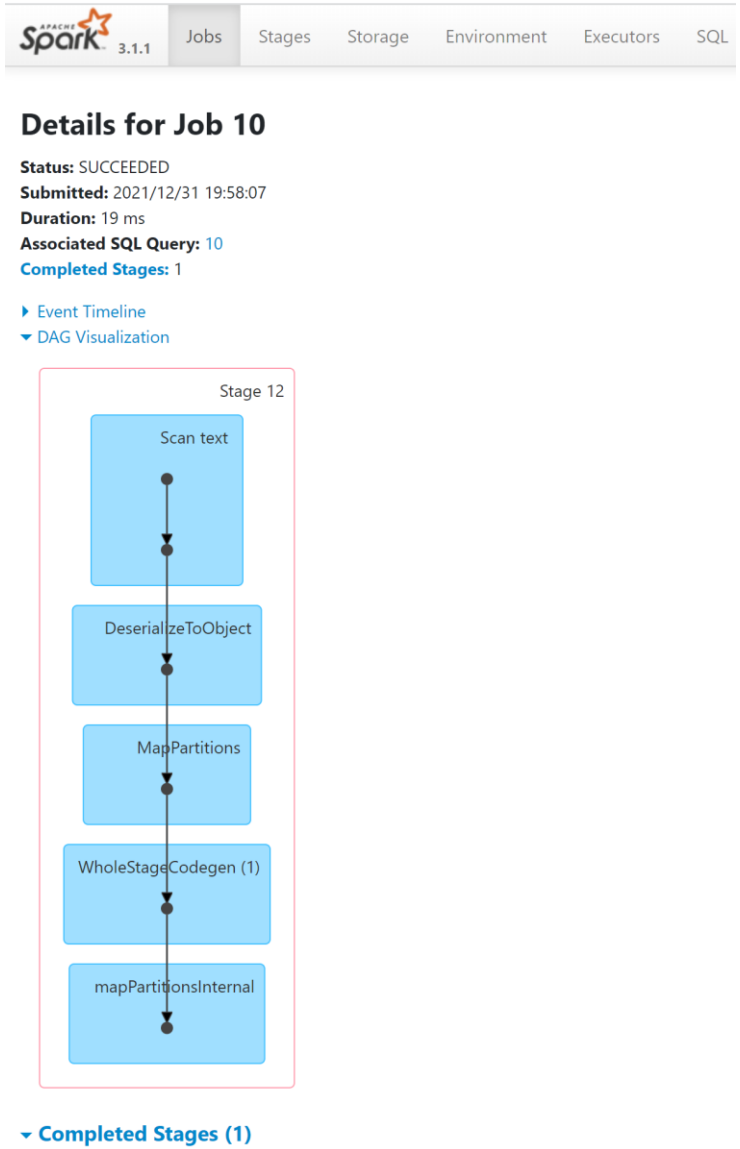
Bytecode generated
… executed within same Thread/Process

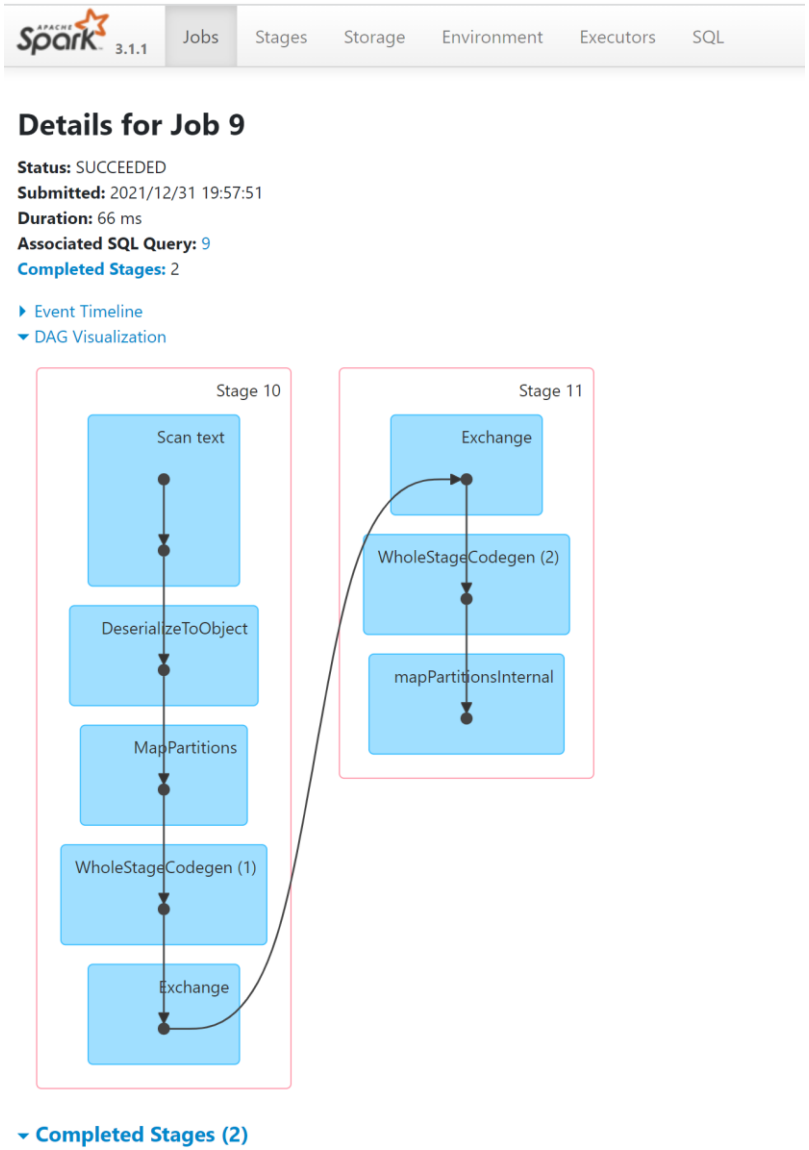
Job =
Full DAG of Stages
+ tasks (retry) scheduling

Exchange Output → Exchange Input

Wide transformation … separation between successive **Stages**

Word Count DAGS
simple -vs- repartition()





Dataset, RDD, DAG ..

R.D.D.

Resilient = can resist failure

…. can be recomputed on demand

Distributed = split in partitions, distributed over executors

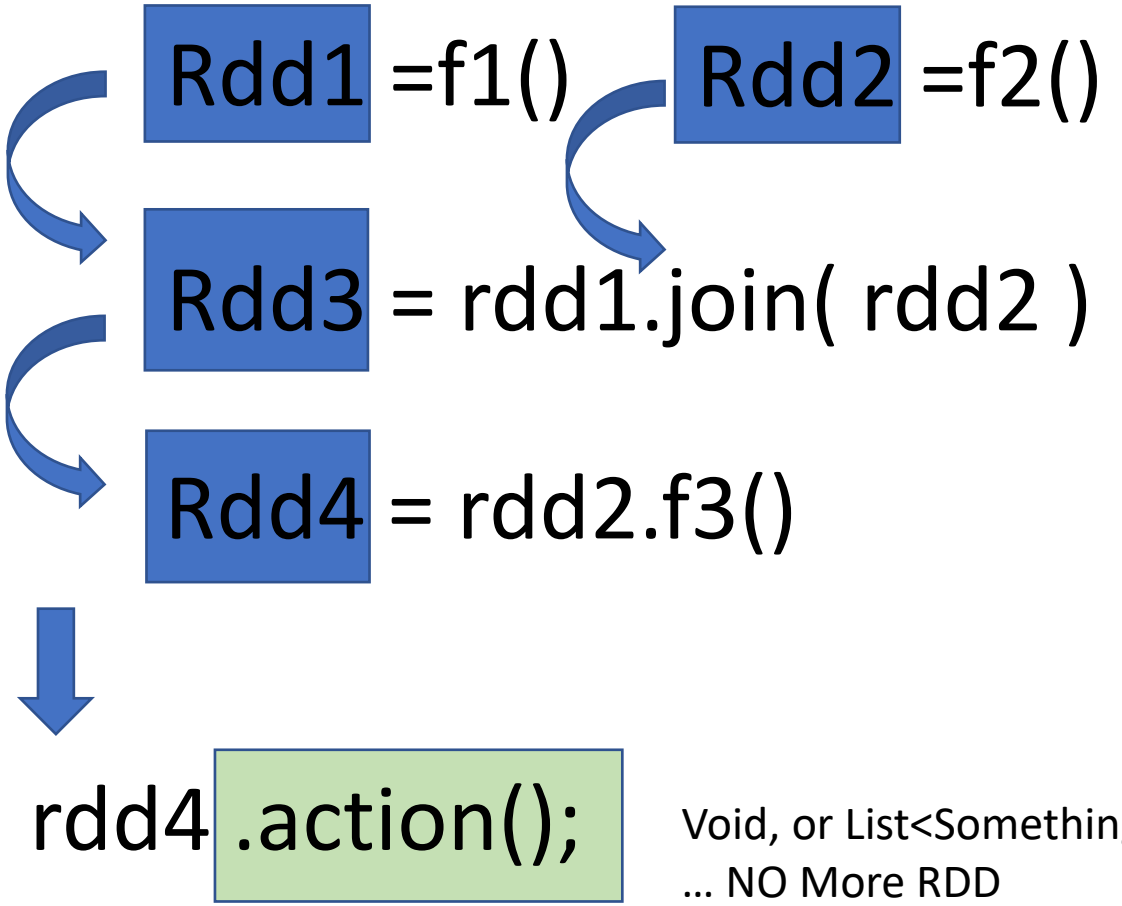
Dataset = … RDD result ends up to be a

ImmutableList<ImmutableData>

… from a function definition + inputs

List<Data> computeFunc() { … }

Compose RDD Functions



Void, or List<SomethingElse>
… NO More RDD

Signature:
(SparkSession, …) -> RDD

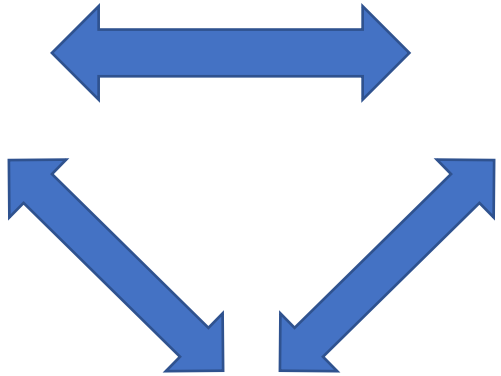
(RDD, …) -> RDD

(RDD, …) -> other

RDD Declarations / Expressions / DAG

RDD **Code Declaration**
(SSA: Single State Assignement**)**

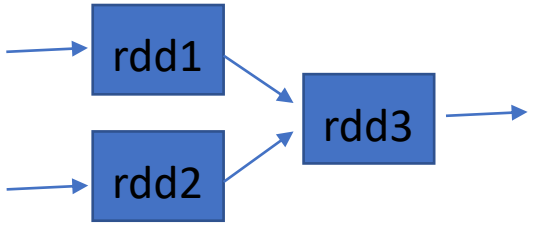
rdd1=..
rdd2=..
rdd3=..



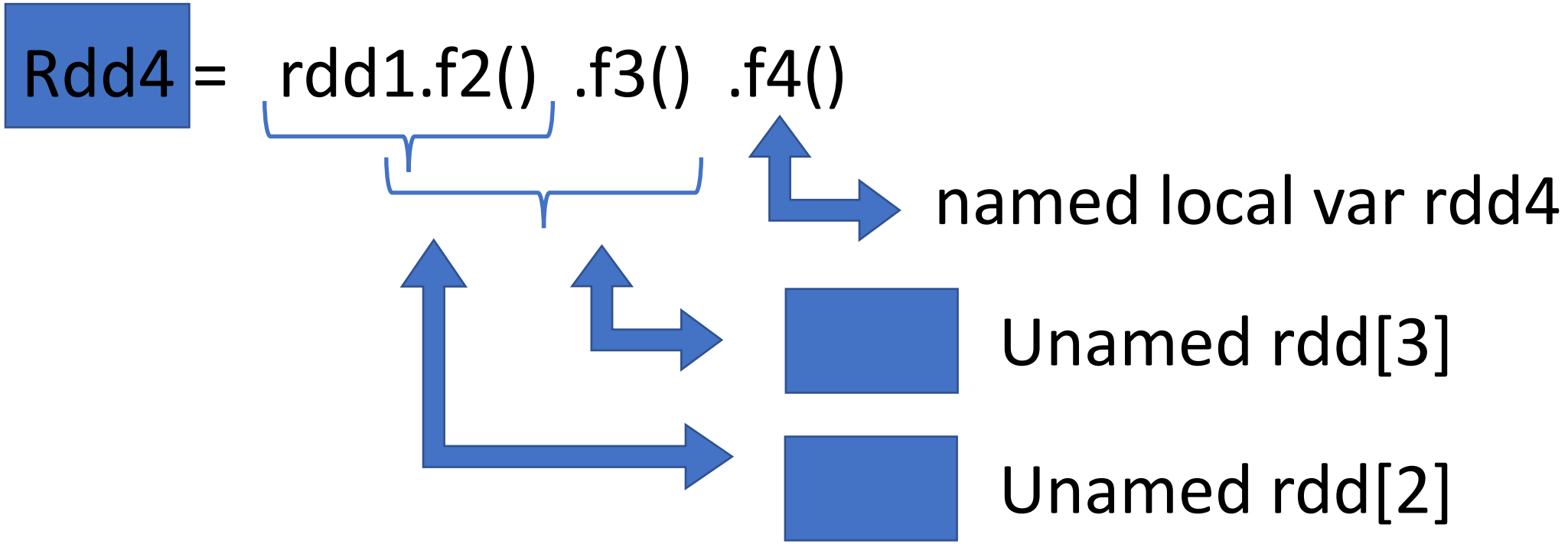
Algebraic Expressions

[f1() .join(f2())].f3()

DAG (Directed Acyclic Graph)



Intermediate (un-named) RDDs using Functionnal API



RDD: Lazy Transformations … then Action

Rdd1 =f1()

Defined, but lazy
=> nothing computed

Rdd2 = rdd1.f2()

=> nothing computed

Rdd3 = rdd2.f3()

=> nothing computed

rdd3 .action();

=> Compute all : f3(f2 (f1()))

Lazy … not necessarily efficient saving

Rdd1 =f1()

Rdd2 = rdd1.f2()

Computed TWICE

Rdd3 = rdd2.f3()

rdd3 .action3();

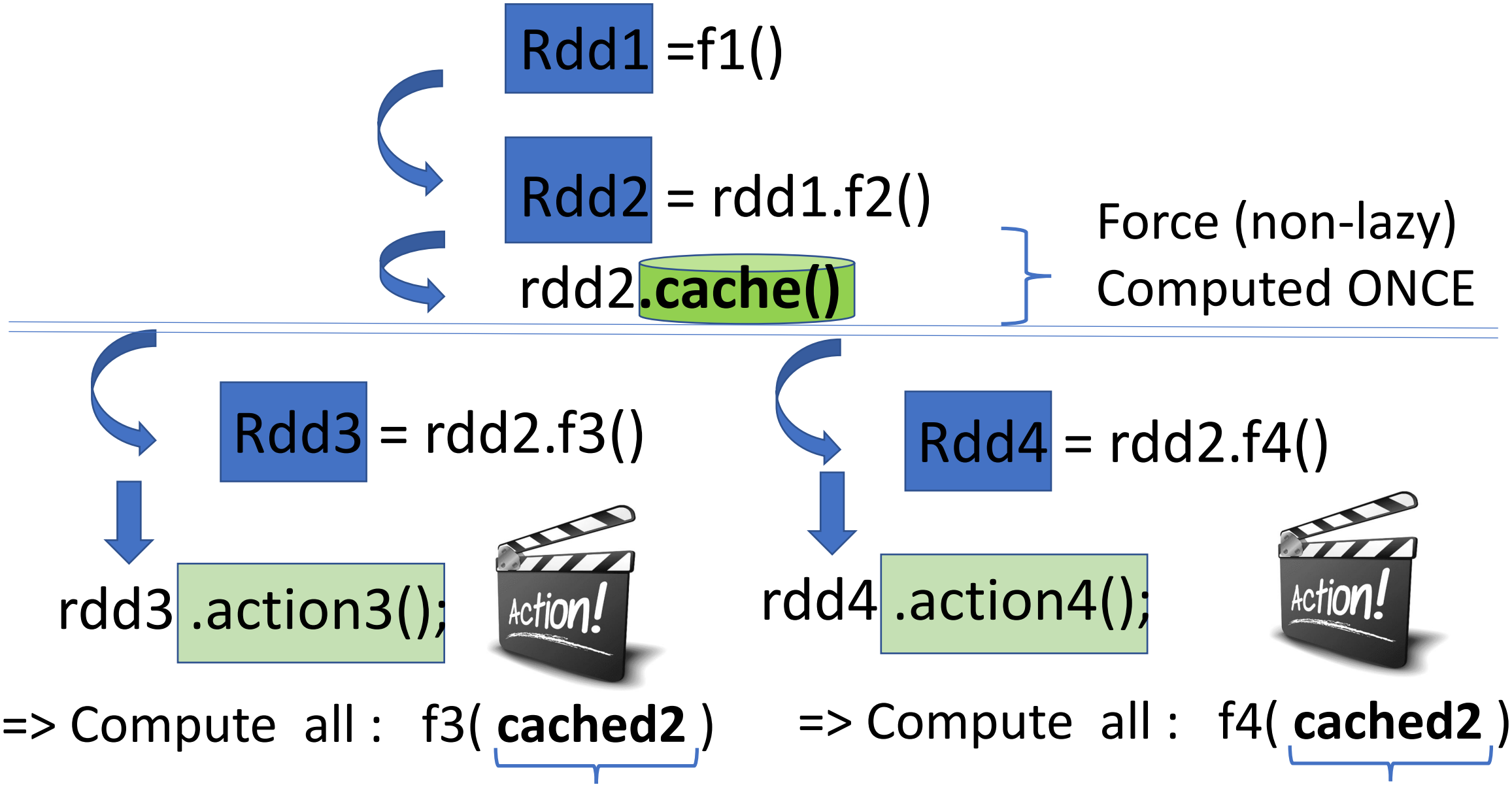
=> Compute all : f3(**f2 (f1())**)

Rdd4 = rdd2.f4()

rdd4 .action4();

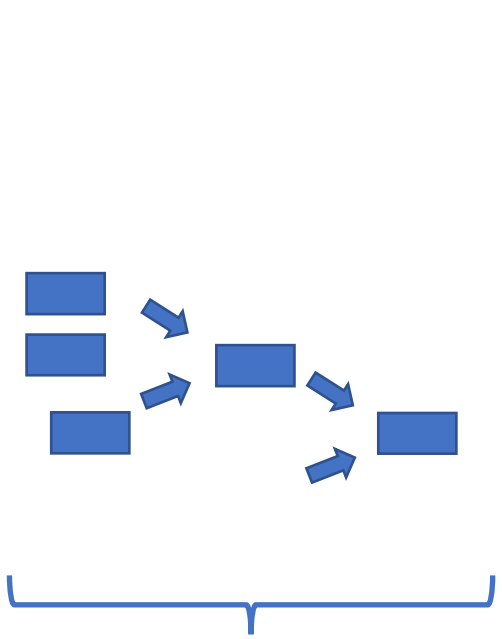
=> Compute all : f4(**f2 (f1())**)

cache() when: common intermediate RDD / safety checkpoint (no restart from scratch)



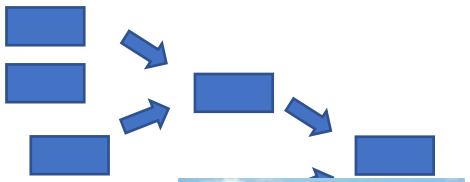
RDD.cache() or .persist() to avoid recomputing

result = Func1 (**cached**(rddFunc2 (rddFunc3 (… rddFunc N)))))

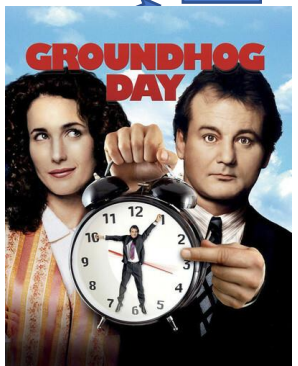


Compute once …

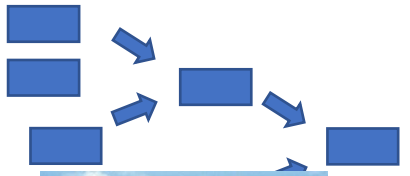
Cached Mem/Disk

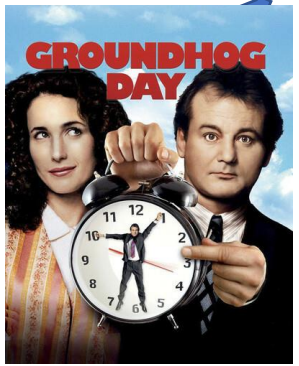


Re-evaluate if Fail

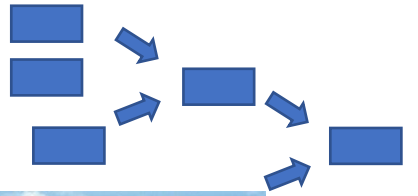


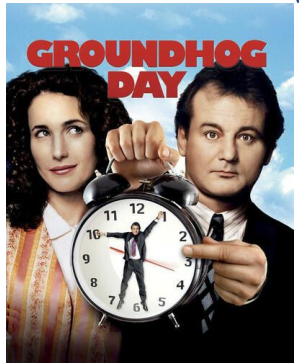
From scratch again?





…





Spark Essentials: *Persistence*

transformation	*description*
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER	Store RDD as serialized Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.
MEMORY_AND_DISK_SER	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.
MEMORY_ONLY_2, **MEMORY_AND_DISK_2,** etc	Same as the levels above, but replicate each partition on two cluster nodes.

abstract class org.apache.spark.rdd.RDD

```scala
/**
 * 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 {
```

```scala
  // ========================================================================
  // 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
- A compute(Partition, TaskContext) : Iterator<T
- A getPartitions() : Partition[]
- getDependencies() : Seq<Dependency<?>
- getPreferredLocations(Partition) : Seq<Strin
- partitioner() : Option<Partitioner>
- sparkContext() : SparkContext
- id() : int
- name() : String

- F dependencies() : Seq<Dependency<?>>
- F partitions() : Partition[]
- F getNumPartitions() : int
- F preferredLocations(Partition) : Seq<String>
- F iterator(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>

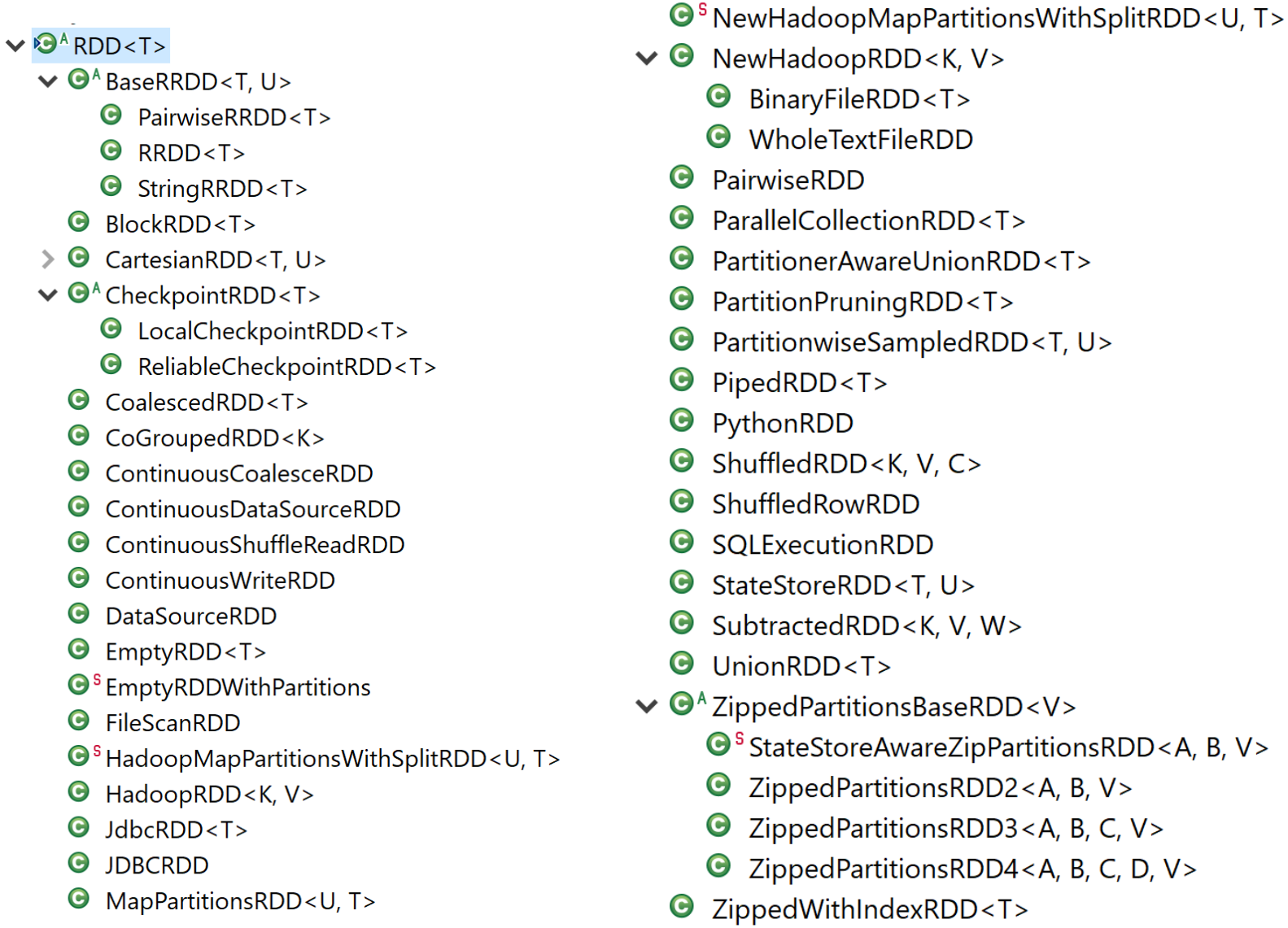
- persist() : RDD<T>
- cache() : RDD<T>
- persist(StorageLevel) : RDD<T>
- unpersist(boolean) : RDD<T>
- unpersist$default$1() : boolean
- getStorageLevel() : StorageLevel

- withScope(Function0<U>) <U> : U
- map(Function1<T, U>, ClassTag<U>) <U> : RDD<U>
- flatMap(Function1<T, TraversableOnce<U>>, ClassTag<
- 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$2(int) : Ordering<T>
- coalesce(int, boolean, Option<PartitionCoalescer>, Orde
- coalesce$default$2() : boolean
- coalesce$default$3() : Option<PartitionCoalescer>
- coalesce$default$4(int, boolean, Option<PartitionCoales
- sample(boolean, double, long) : RDD<T>
- sample$default$3() : long
- randomSplit(double[], long) : RDD<T>[]
- randomSplit$default$2() : long
- randomSampleWithRange(double, double, long) : RDD<
- takeSample(boolean, int, long) : Object
- union(RDD<T>) : RDD<T>
- $plus$plus(RDD<T>) : RDD<T>
- sortBy(Function1<T, K>, boolean, int, Ordering<K>, Clas
- 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$3(RDD<T>, Partitioner) : Ordering<
- cartesian(RDD<U>, ClassTag<U>) <U> : RDD<Tuple2<T
- groupBy(Function1<T, K>, ClassTag<K>) <K> : RDD<Tu
- groupBy(Function1<T, K>, int, ClassTag<K>) <K> : RDD<
- groupBy(Function1<T, K>, Partitioner, ClassTag<K>, Ord
- groupBy$default$4(Function1<T, K>, Partitioner) <K> :
- pipe(String) : RDD<String>
- pipe(String, Map<String, String>) : RDD<String>
- pipe(Seq<String>, Map<String, String>, Function1<Func
- mapPartitions$default$2() <U> : boolean
- mapPartitionsWithIndexInternal(Function2<Object, Iterator
- mapPartitionsInternal(Function1<Iterator<T>, Iterator<U>
- mapPartitionsInternal$default$2() <U> : boolean
- mapPartitionsWithIndex(Function2<Object, Iterator<T>, Ite
- mapPartitionsWithIndex(Function2<Object, Iterator<T>, Ite
- mapPartitionsWithIndexInternal$default$2() <U> : boolean
- mapPartitionsWithIndexInternal$default$3() <U> : boolean
- mapPartitionsWithIndex$default$2() <U> : boolean
- zip(RDD<U>, ClassTag<U>) <U> : RDD<Tuple2<T, U>>
- zipPartitions(RDD<B>, boolean, Function2<Iterator<T>, Ite
- zipPartitions(RDD<B>, Function2<Iterator<T>, Iterator<B>
- zipPartitions(RDD<B>, RDD<C>, boolean, Function3<Itera
- zipPartitions(RDD<B>, RDD<C>, Function3<Iterator<T>, It
- zipPartitions(RDD<B>, RDD<C>, RDD<D>, boolean, Funct
- zipPartitions(RDD<B>, RDD<C>, RDD<D>, Function4<Iter
- foreach(Function1<T, BoxedUnit>) : void
- foreachPartition(Function1<Iterator<T>, BoxedUnit>) : voi
- toLocalIterator() : Iterator<T>
- collect(PartialFunction<T, U>, ClassTag<U>) <U> : RDD<U
- subtract(RDD<T>) : RDD<T>
- subtract(RDD<T>, int) : RDD<T>
- subtract(RDD<T>, Partitioner, Ordering<T>) : RDD<T>
- subtract$default$3(RDD<T>, Partitioner) : Ordering<T>
- reduce(Function2<T, T, T>) : T
- treeReduce(Function2<T, T, T>, int) : T
- treeReduce$default$2() : int
- fold(T, Function2<T, T, T>) : T
- aggregate(U, Function2<U, T, U>, Function2<U, U, U>, Cla
- treeAggregate(U, Function2<U, T, U>, Function2<U, U, U>
- treeAggregate$default$4(U) <U> : int
- countApprox(long, double) : PartialResult<BoundedDouble
- countApprox$default$2() : double
- countByValue(Ordering<T>) : Map<T, Object>
- countByValue$default$1() : Ordering<T>
- countByValueApprox(long, double, Ordering<T>) : PartialR
- countByValueApprox$default$2() : double
- countByValueApprox$default$3(long, double) : Ordering<T
- countApproxDistinct(int, int) : long
- countApproxDistinct(double) : long
- countApproxDistinct$default$1() : double
- take(int) : Object

RDD operators…

- 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
- 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

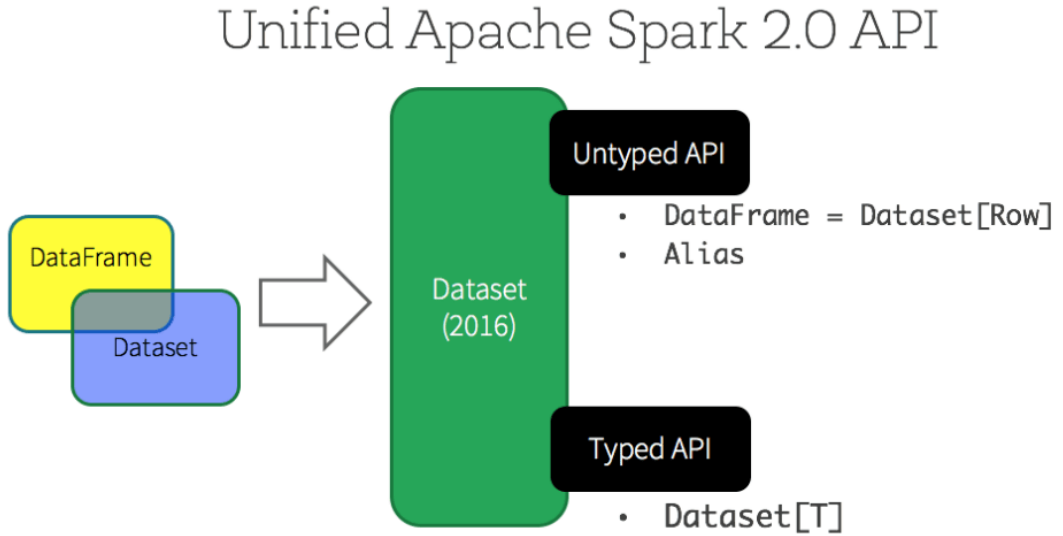


Deprecated DataFrame api … now DataSet<Row>

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

String + json

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

```
/**
 * 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)
```

Typed field getters

- A get(int) : Object
- D isNullAt(int) : boolean
- D getBoolean(int) : boolean
- D getByte(int) : byte
- D getShort(int) : short
- D getInt(int) : int
- D getLong(int) : long
- D getFloat(int) : float
- D getDouble(int) : double
- D getString(int) : String
- D getDecimal(int) : BigDecimal
- D getDate(int) : Date
- D getLocalDate(int) : LocalDate
- D getTimestamp(int) : Timestamp
- D getInstant(int) : Instant
- D getSeq(int) <T> : Seq<T>
- D getList(int) <T> : List<T>
- D getMap(int) <K, V> : Map<K, V>
- D getJavaMap(int) <K, V> : Map<K, V>
- D getStruct(int) : Row
- D getAs(int) <T> : T
- D getAs(String) <T> : T
- D fieldIndex(String) : int
- D 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	DataFrames	Datasets
Syntax Errors	Runtime	Compile Time	Compile Time
Analysis Errors	Runtime	Runtime	Compile Time

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

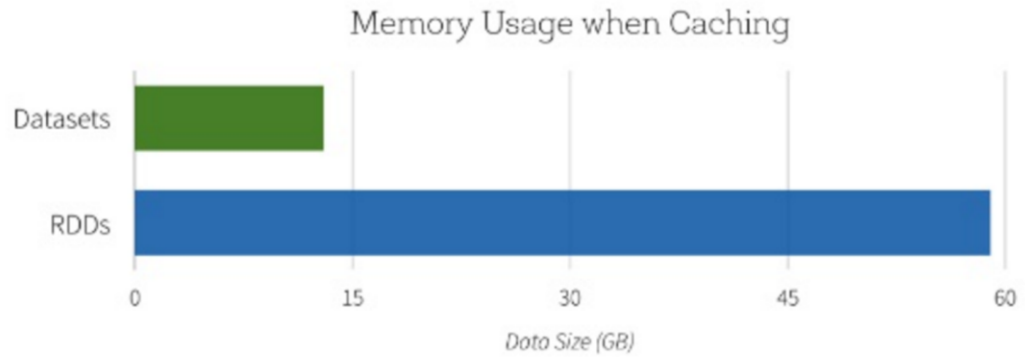
But works well + safe at compile time + easily readable & maintanable

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



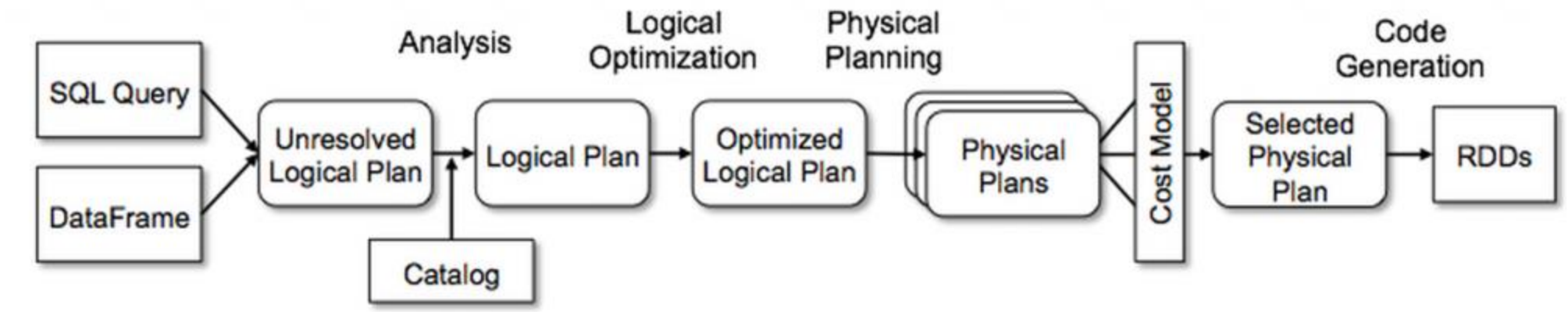
Second, since **Spark as a compiler** understands your Dataset type JVM object, it maps your type-specific JVM object to Tungsten's internal memory representation using **Encoders**. As a result, **Tungsten** Encoders can efficiently serialize/deserialize JVM objects as well as generate compact bytecode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

OffHeap vs Managed JVM Memory
⇒ May handle Huge DATA
with « small » JVM GC memory

Efficient network
(shuffling) serialization

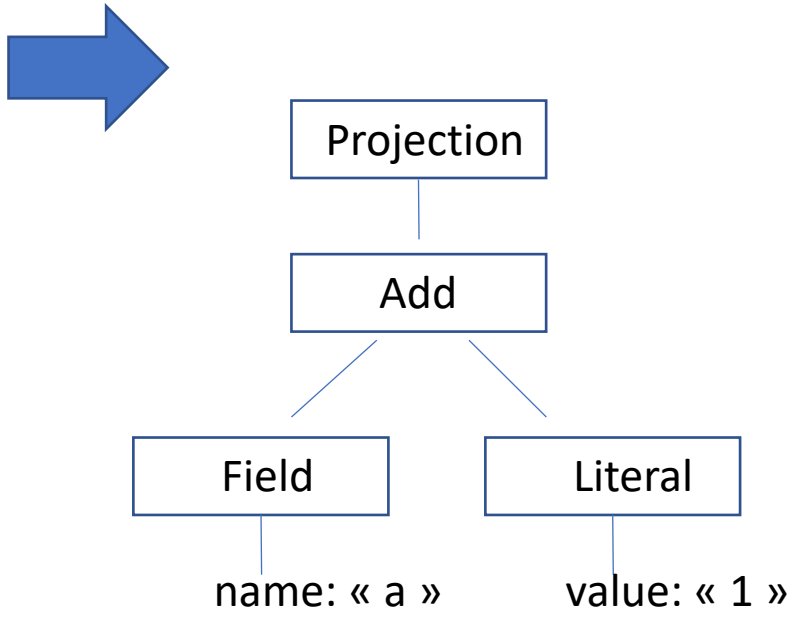
Efficient SpillToDisk
(swapping)

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



Compiler chain: Parser -> AST -> ..

Sql> select **(a+1)** from …




```scala
case class Add(left: Expression, right: Expression) extends BinaryArithmetic {
  override def symbol: String = "+"

  lazy val numeric = dataType match {
    case n: NumericType => n.numeric.asInstanceOf[Numeric[Any]]
    case other => sys.error(s"Type $other does not support numeric operations")
  }

  override def eval(input: Row): Any = {
    val evalE1 = left.eval(input)
    if(evalE1 == null) {
      null
    } else {
      val evalE2 = right.eval(input)
      if (evalE2 == null) {
        null
      } else {
        numeric.plus(evalE1, evalE2)
      }
    }
  }
}
```

Catalyst AST -> CodeGenerator

apache / spark Public

Watch 2.1k | Fork 25k | Star 31.7k

Code | Pull requests 241 | Actions | Projects | Security | Insights

fedbfc7074 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

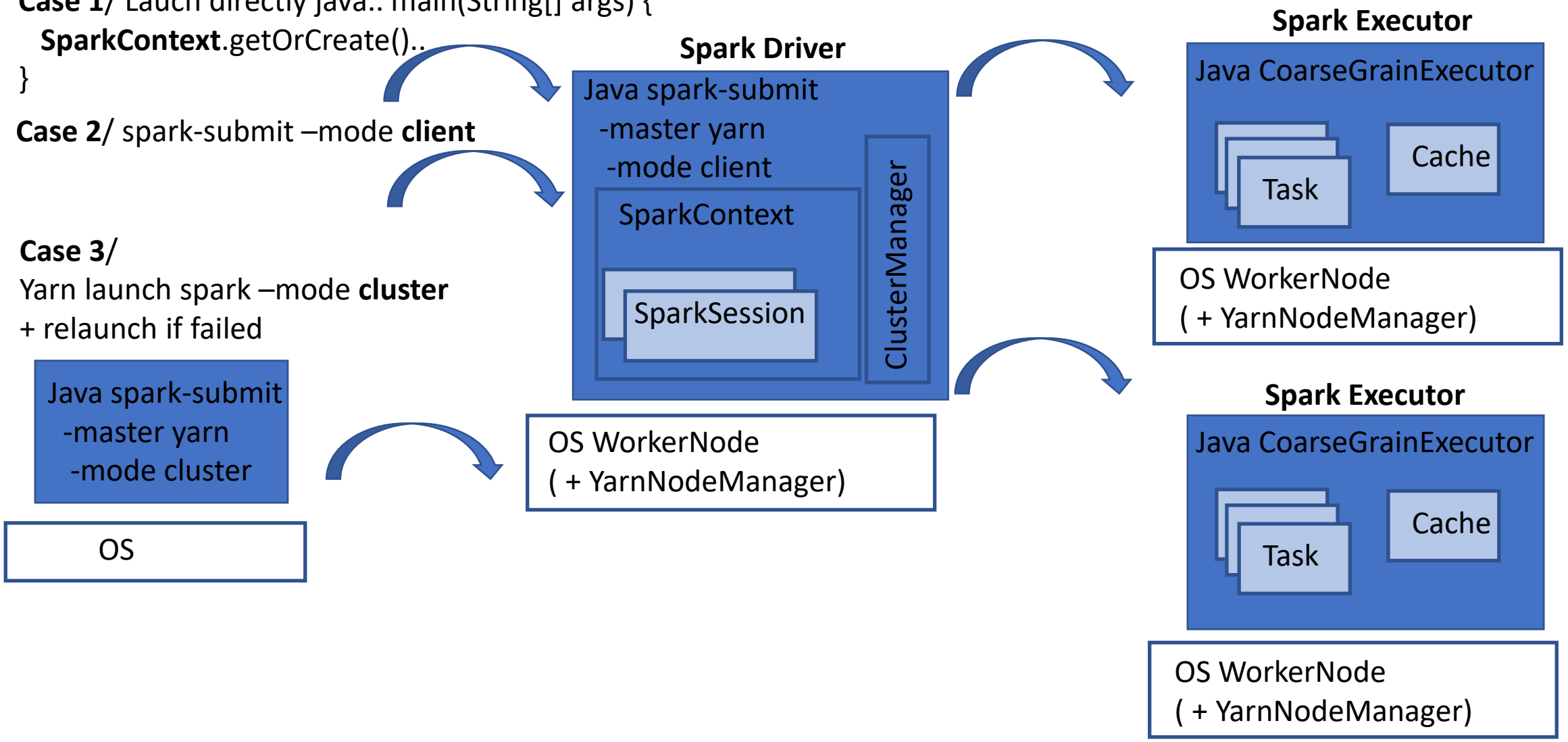
program/library/servers

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

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

Case 2/ spark-submit –mode **client**

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



Develop, Package, Deploy

pom.xml

```
<dependencies>

    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-core_2.12</artifactId>
    </dependency>

    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-sql_2.12</artifactId>
    </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);
```

Spark-Java API with Lambda (ambigous 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 !

```
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.iterableAsScalaIterableConverter(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

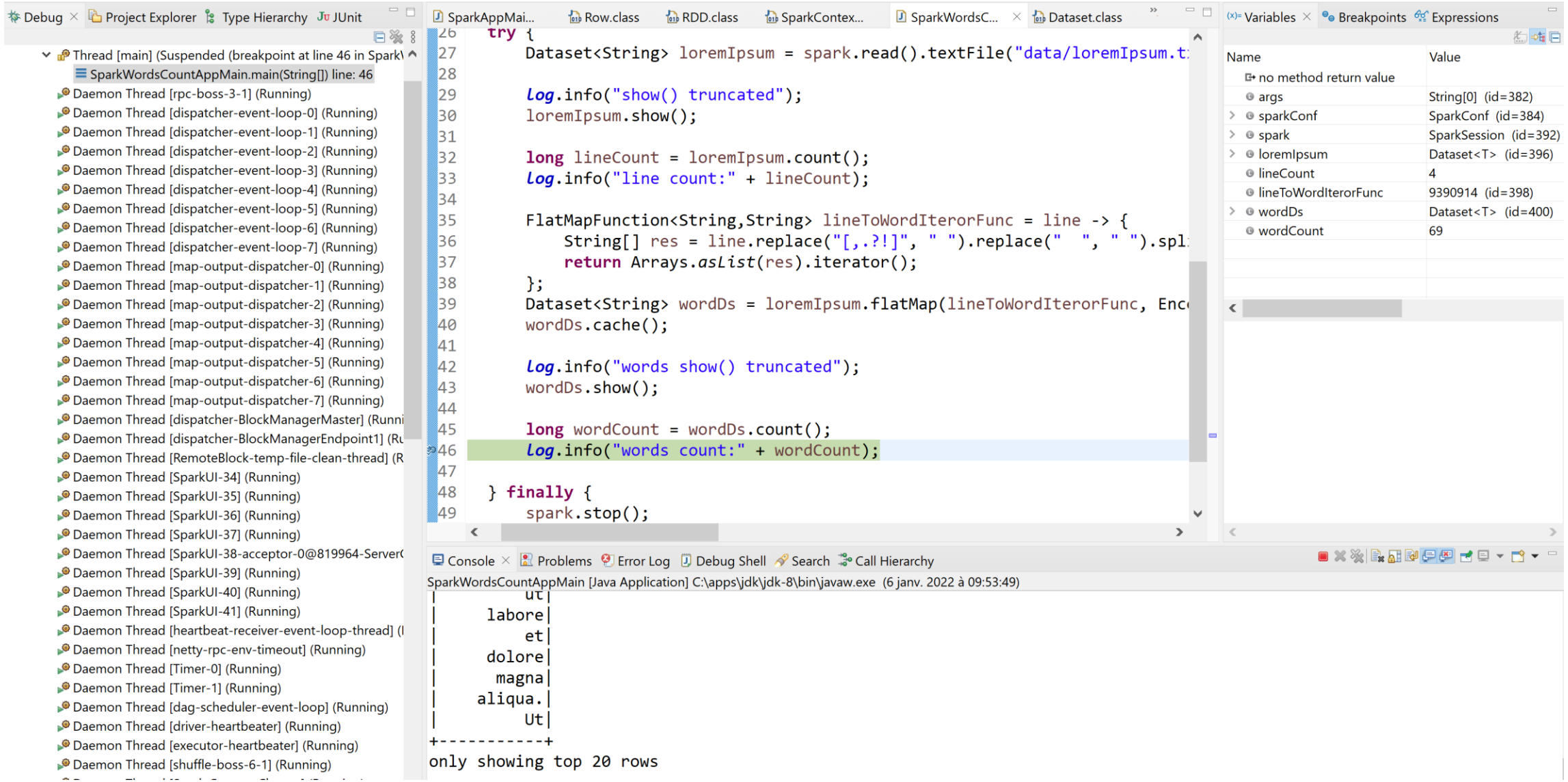
<terminated> SparkWordsCountAppMain [Java Application] C:\apps\jdk\jdk-8\bin\javaw.exe (6 janv. 2022 à 10:15:13 – 10:15:28)

```
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.NotSerializableExcept
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
        - 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,
```

Trick… isolated top-level Function1 class for 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.iterableAsScalaIterableConverter(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();
}
```

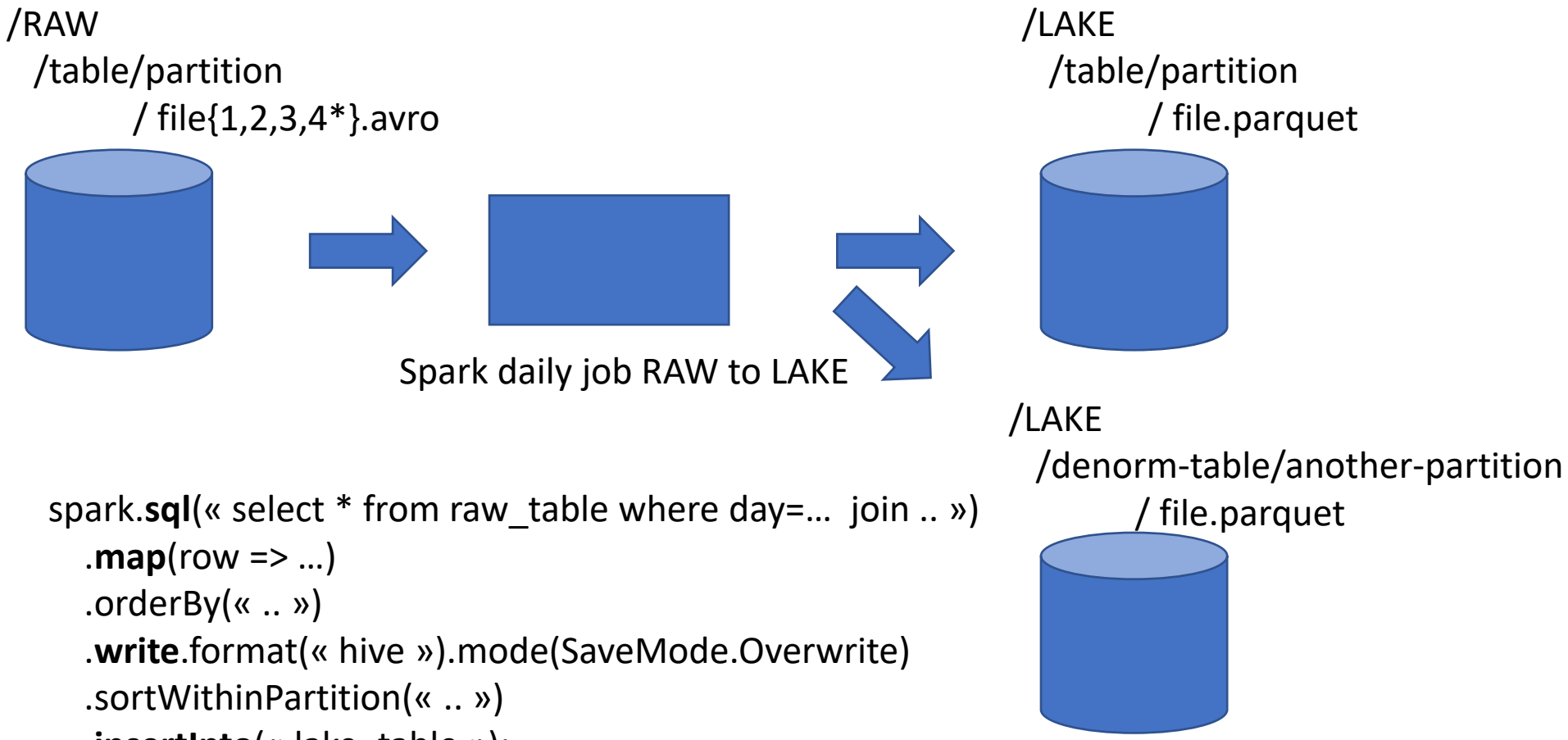
Local[*] Debug from IDE



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.cla
        // => 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");
}
```

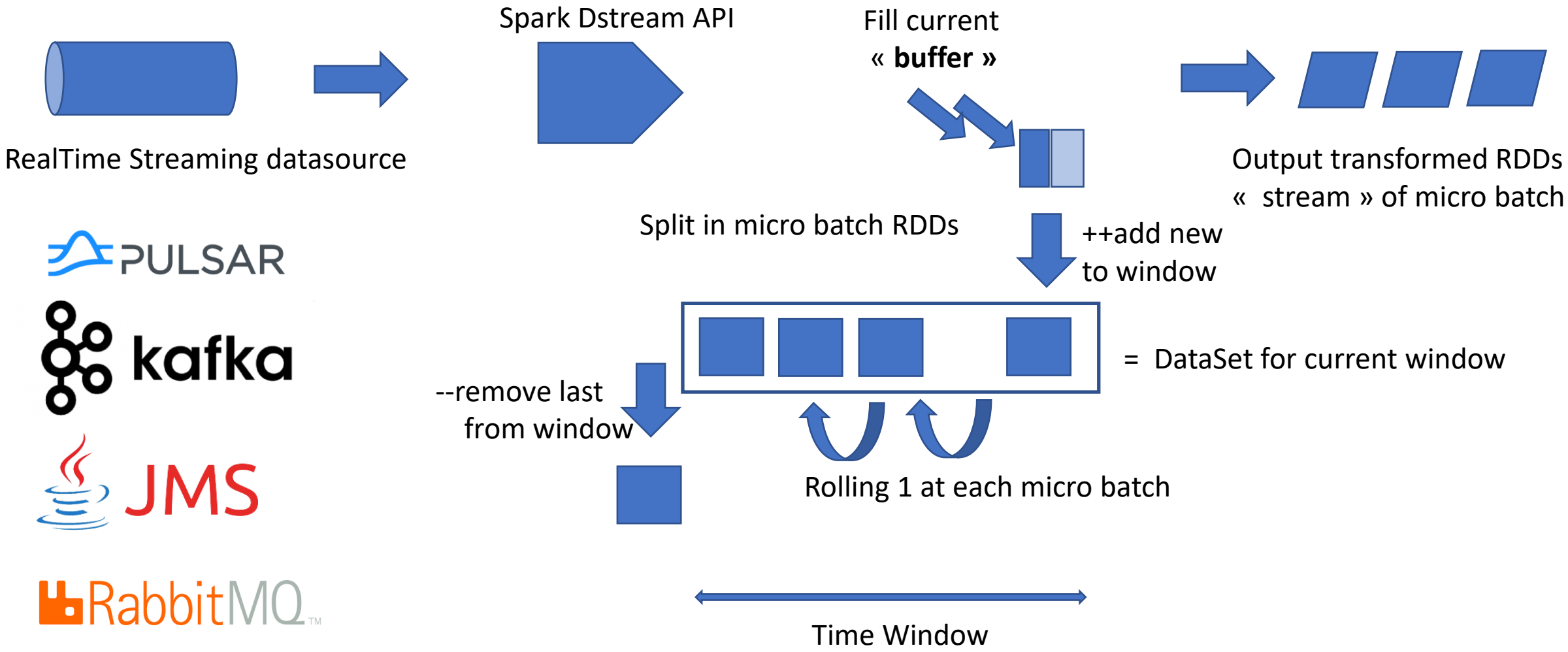
Typical Usage: process RAW to LAKE



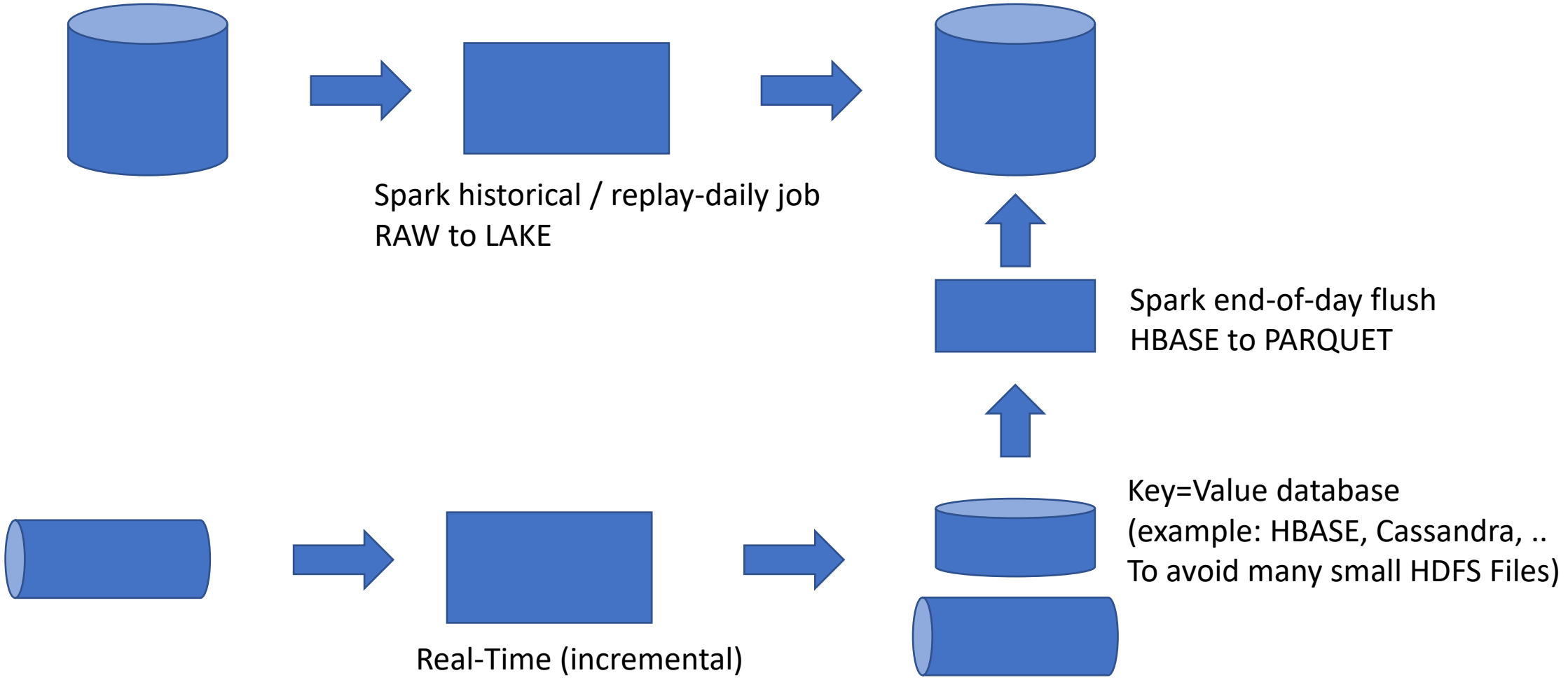

```
spark.sql(« select * from raw_table where day=…  join .. »)
   .map(row => …)
   .orderBy(« .. »)
   .write.format(« hive »).mode(SaveMode.Overwrite)
   .sortWithinPartition(« .. »)
   .insertInto(« lake_table »);
```

Spark Streaming : micro Batches

Spark DStream API … as + enriched DataSet API

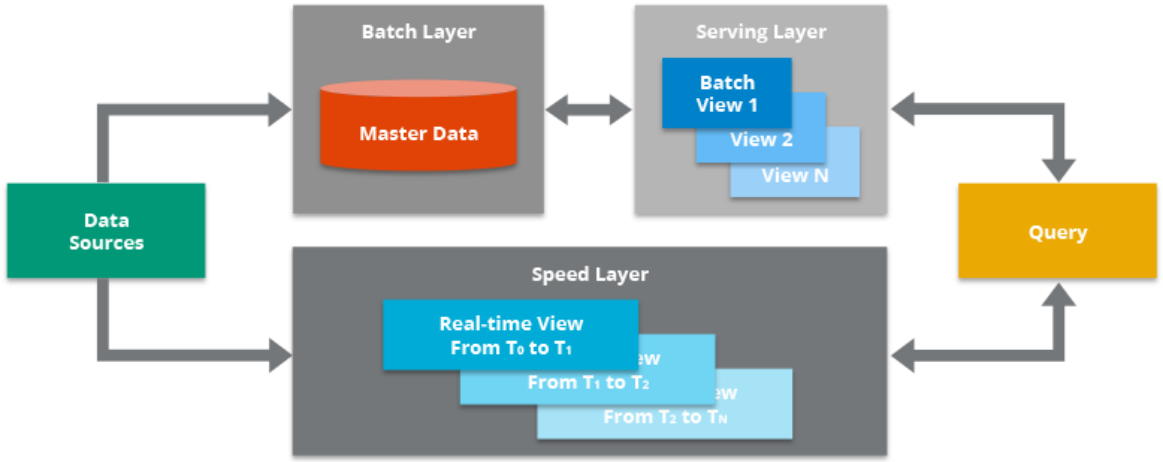


Typical Usage … Streaming vs Daily

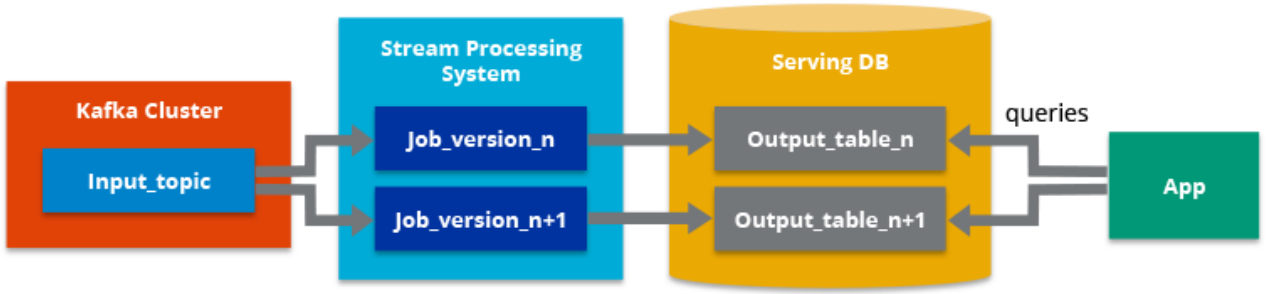


Typical Architectures.. Lambda vs Kappa

Lambda



Kappa



Conclusion & Questions