

BigData – Spark – Processing

Spark cluster mode distribution & Tuning

Course Esilv 2024
arnaud.nauwynck@gmail.com

this document:

[https://github.com/Arnaud-Nauwynck/presentations/pres-bigdata/
Spark-ClusterMode](https://github.com/Arnaud-Nauwynck/presentations/pres-bigdata/Spark-ClusterMode)

Outline

Reminder Spark-driver / spark-executors

Cluster Tuning Params

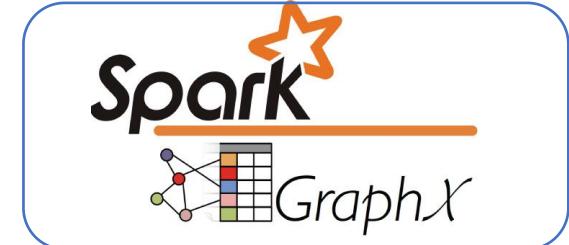
deployMode (local, spark://, yarn, k8s)

Others launchers (java embedded, livy, jupyter, pyspark ..)

Reminder

Spark-Core + ...

Structured Data



Azure Data Lake Storage Gen2

DataSource Connectors
(Hadoop API)



Modules



Languages Support

Cluster Manager

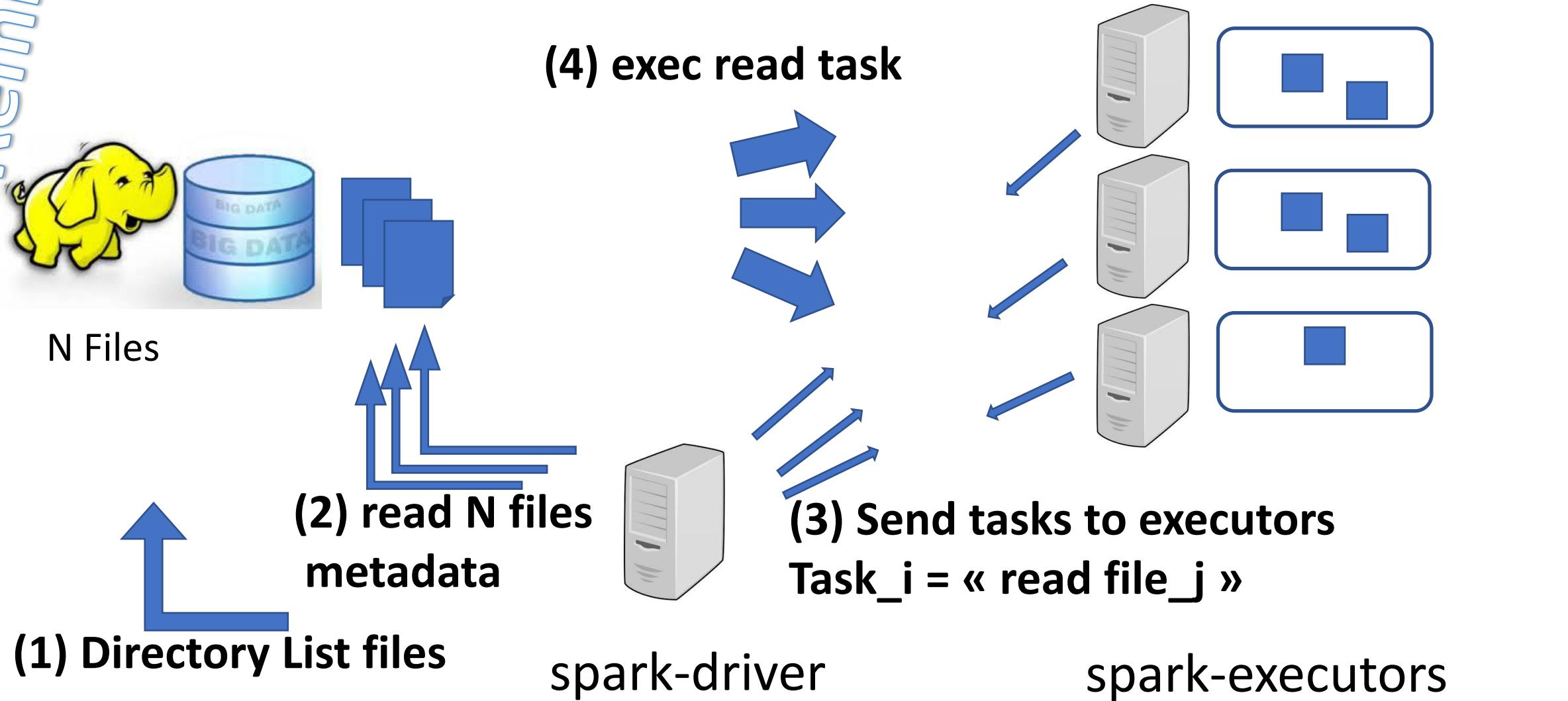


1 Spark application

= 1 Spark-driver + N spark-executors

Reminder

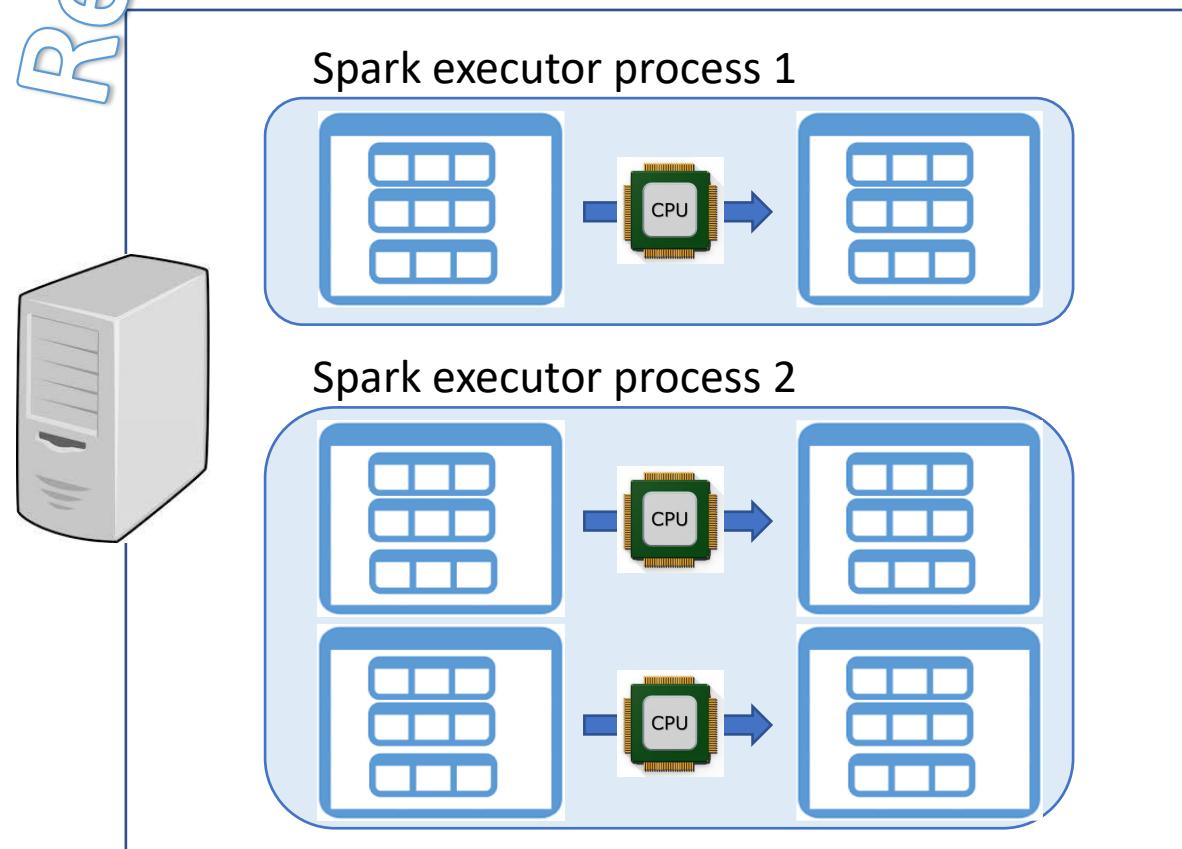
Read N Files – assign Tasks to Executors



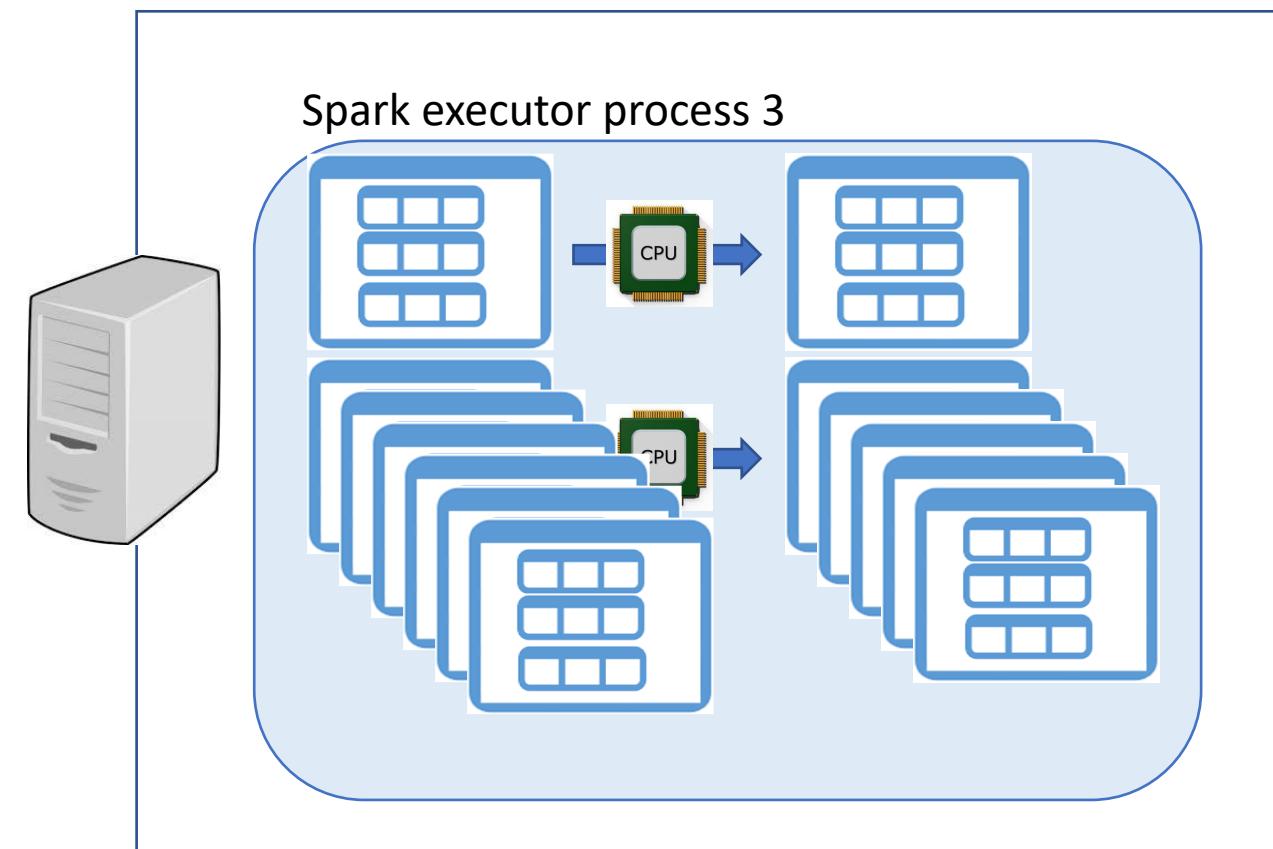
Reminder

Parallelizable per partition / available Cpu
1 or several partitions per Executor
..1 or several Executors

Worker node 1



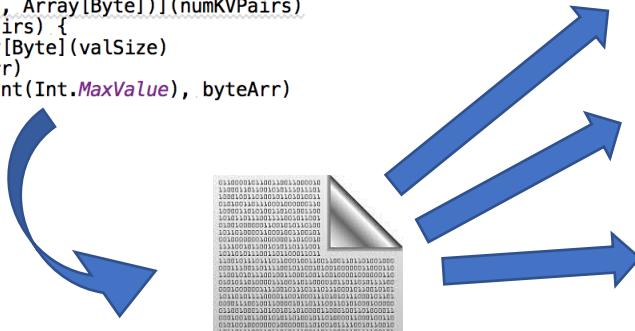
Worker node 2



Reminder

Program
= Dataset instructions

```
36 val sc = new SparkContext(args(0), "GroupBy Test",  
37   System.getenv("SPARK_HOME"), SparkContext.jarOfClass(this.getClass))  
38  
39 val pairs1 = sc.parallelize(0 until numMappers, numMappers).flatMap { p =>  
40   val rangen = new Random  
41   var arr1 = new Array[(Int, Array[Byte])](numKVPairs)  
42   for (i <- 0 until numKVPairs) {  
43     val byteArr = new Array[Byte](valSize)  
44     rangen.nextBytes(byteArr)  
45     arr1(i) = (rangen.nextInt(Int.MaxValue), byteArr)  
46   }  
47   arr1
```

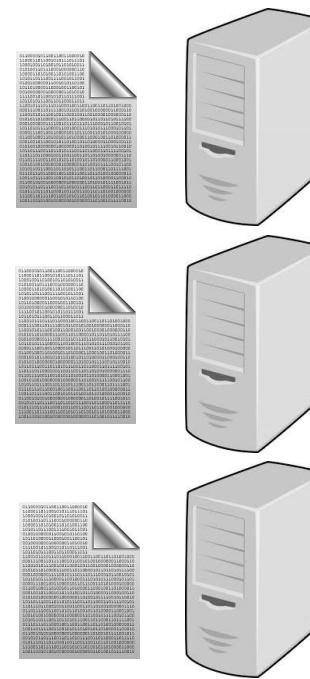


(1) Generate java code

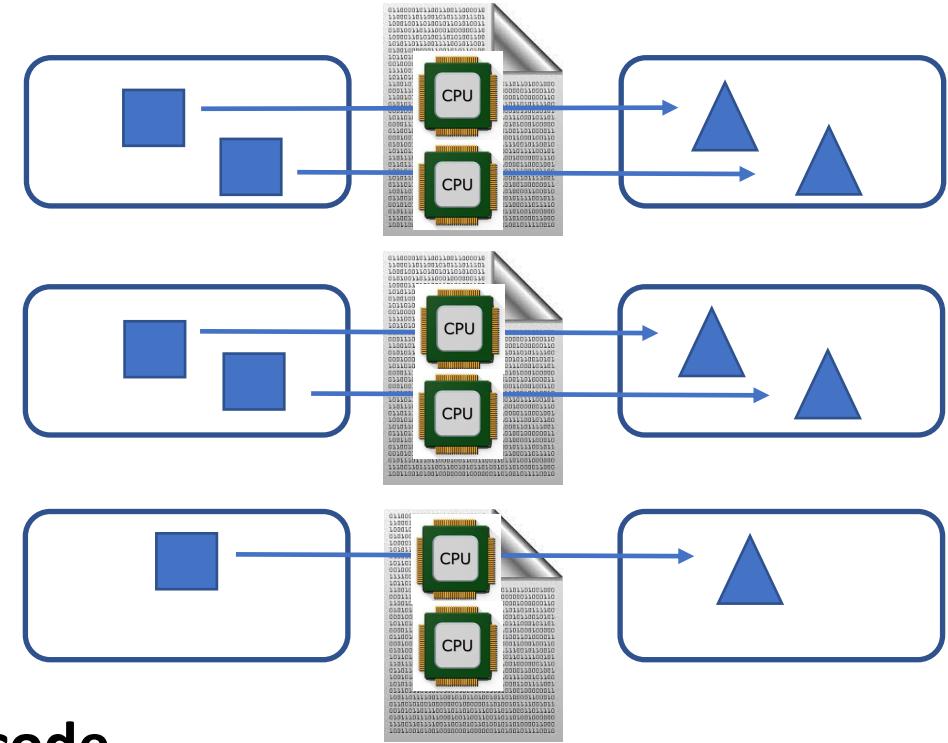
(RDD Spark sub-class « WholeStageCodeGen\$i »)

(2) Compile Bytecode

WholeStageCodeGen



(3) Send task + bytecode
to spark-executors



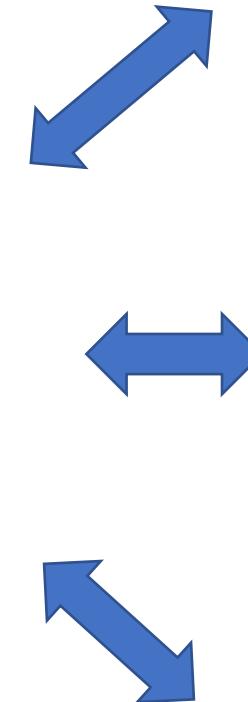
(4) Execute tasks

Reminder

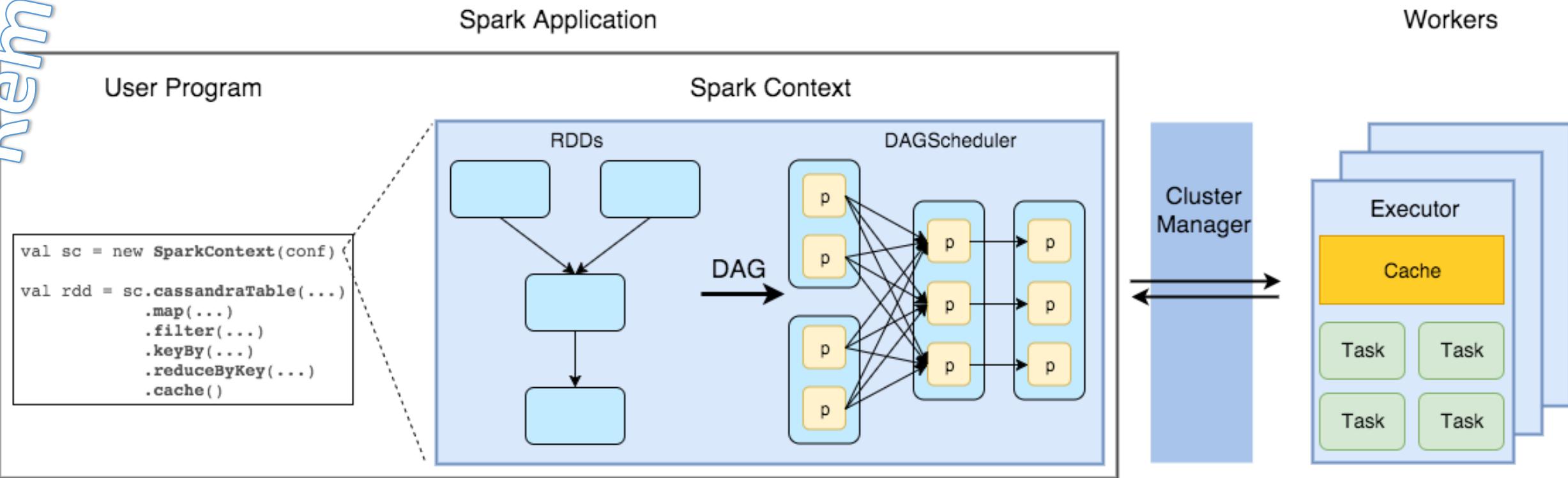
Cluster Manager Extensions: TaskScheduler API



```
abstract class TaskScheduler {  
    ..start(),stop(),  
  
    submitTasks, cancelTasks,  
  
    notify Host-Executor-Task changes  
}
```



Reminder



Reminder

CoarseGrainScheduler ... CoarseGrain Executer

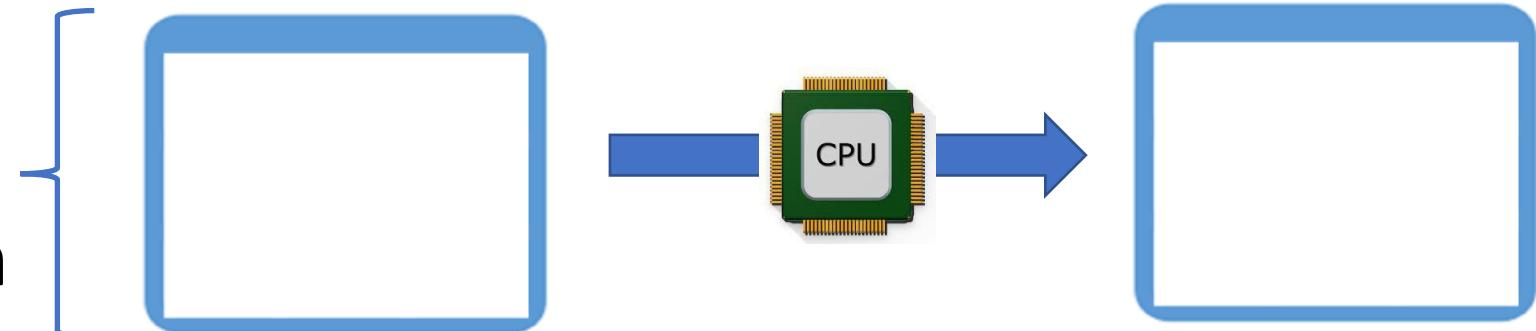
Spark-driver
Implements Fault Tolerance+Distribution



Spark-executor
... internally called « CoarseGrainExecutor » main loop



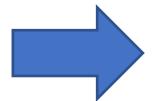
CoarseGrain
partition
= unit or recomputation



Outline



Reminder Spark-driver / spark-executors



Cluster Tuning Params

deployMode (local, spark://, yarn, k8s)

Others launchers (java embedded, livy, jupyter, pyspark ..)

Tuning Spark to use cluster power ?

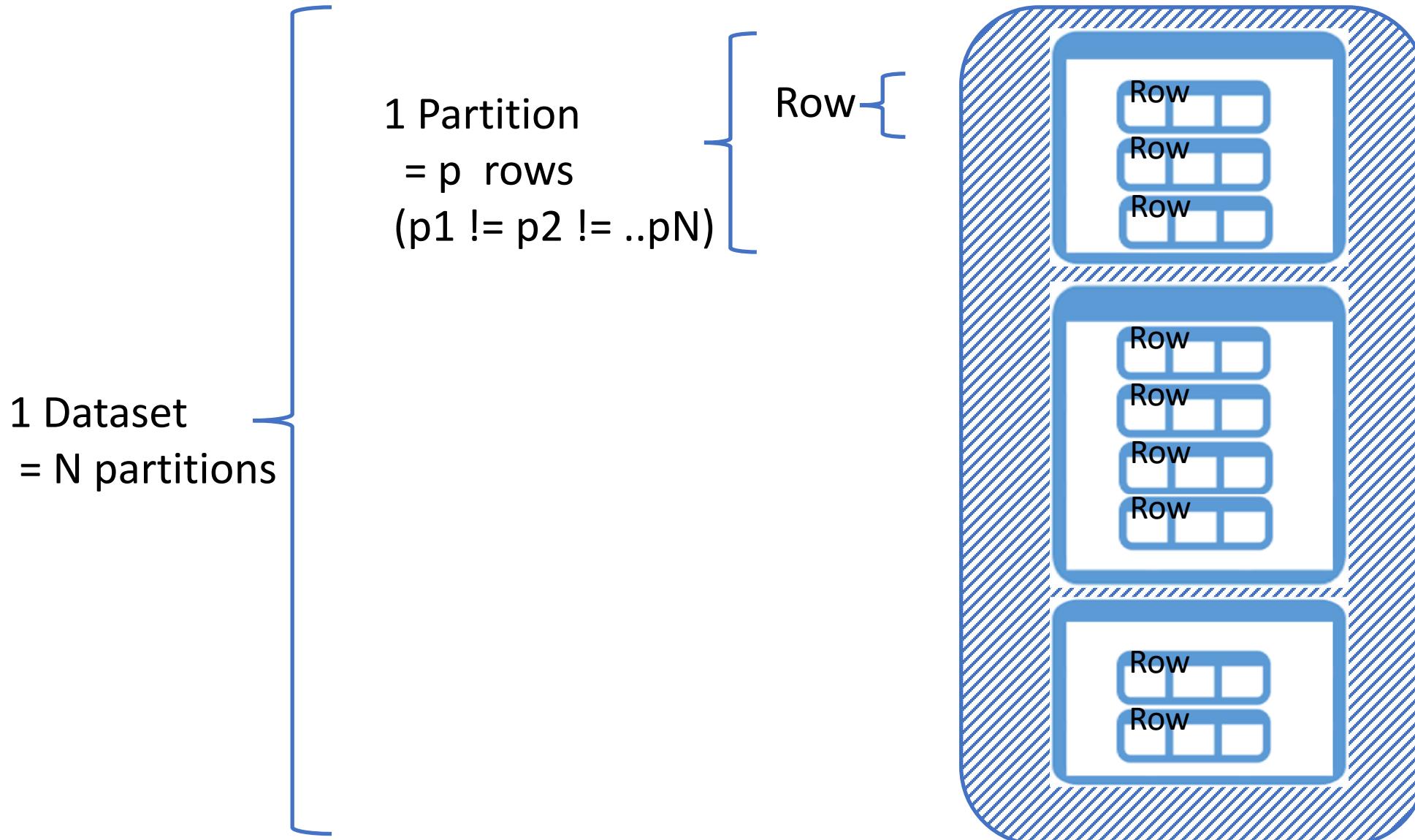
Distribution / Partitions / Parallelism



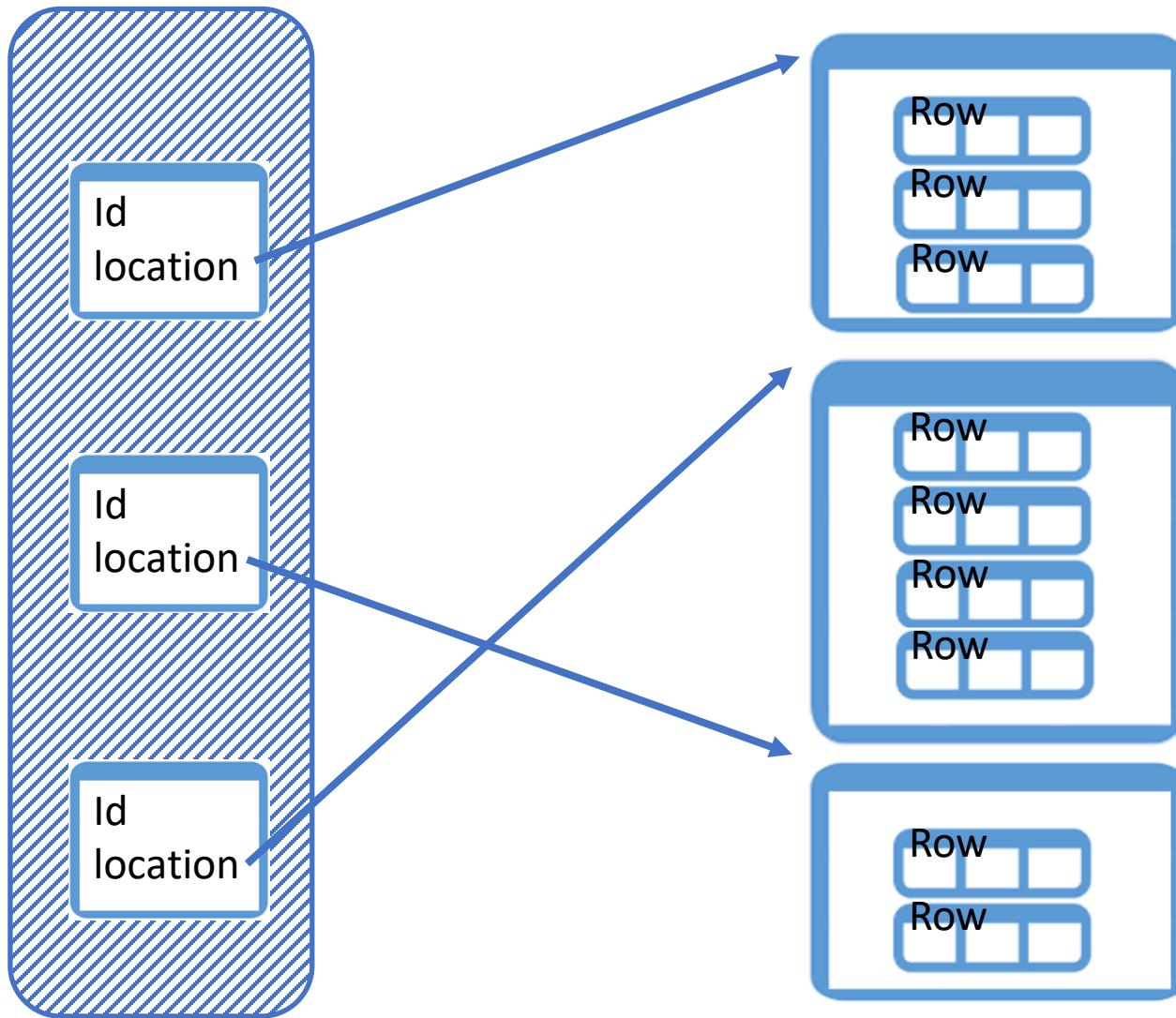
Memory needs for Dataset

driver != executer

DataSet = Partitions metadata / data



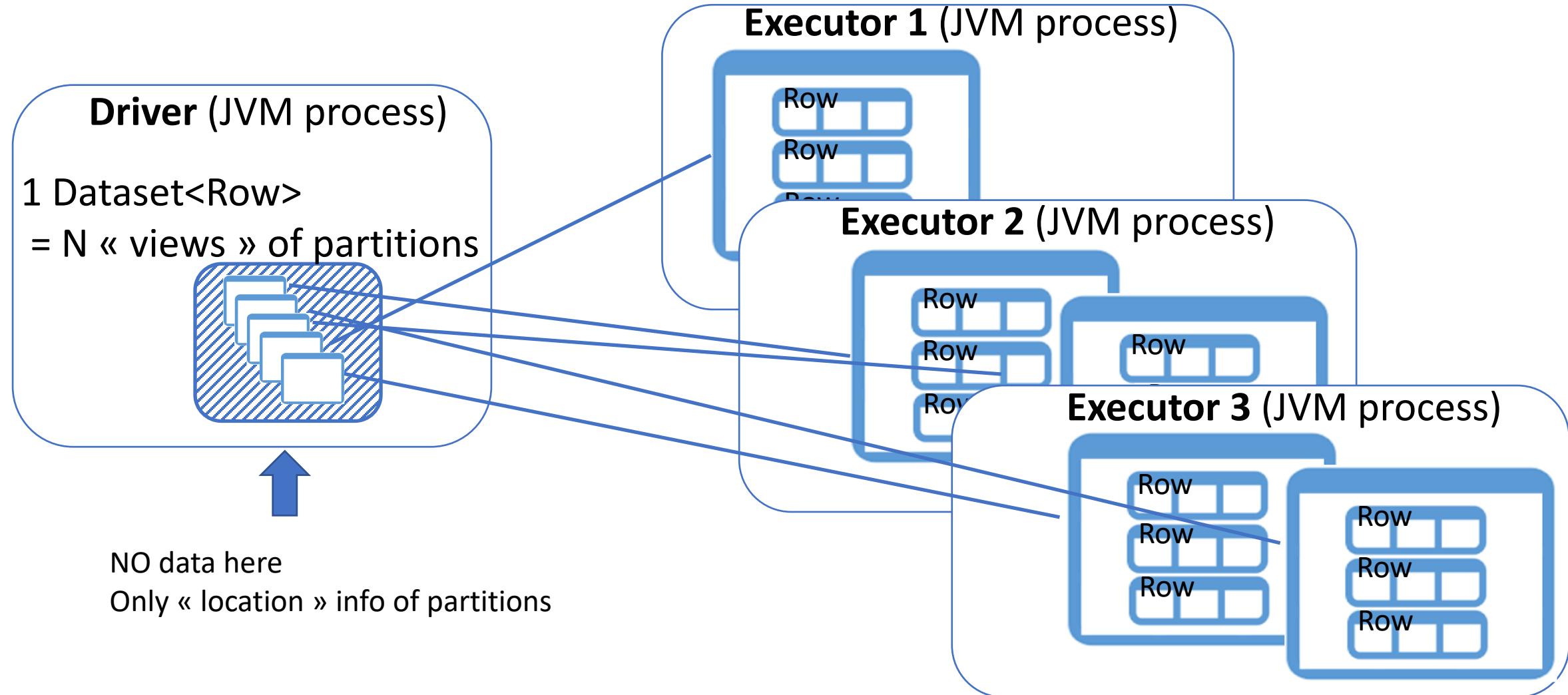
On Driver = metadata / on Executor = Data



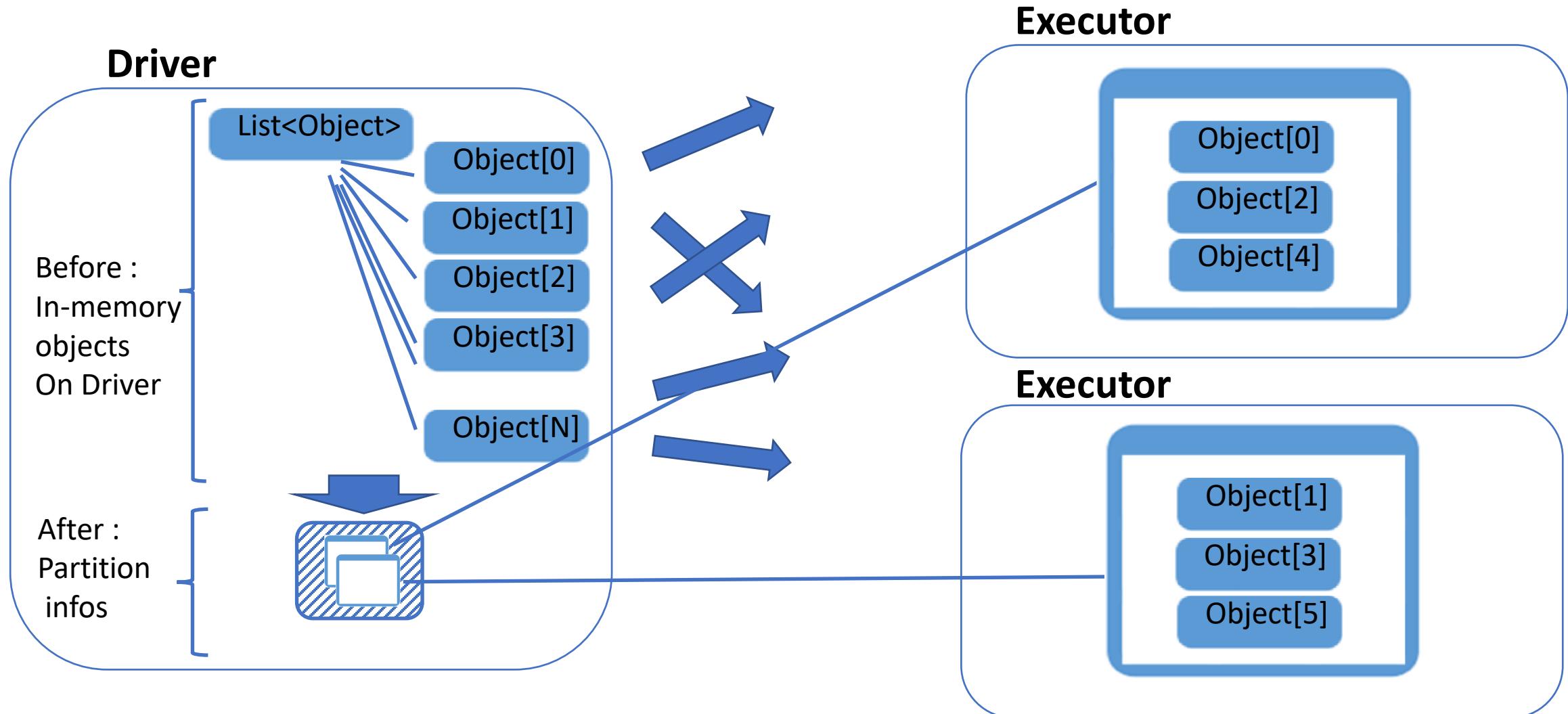
1 Dataset on spark-driver

N x Partitions (blocks) on spark-executors

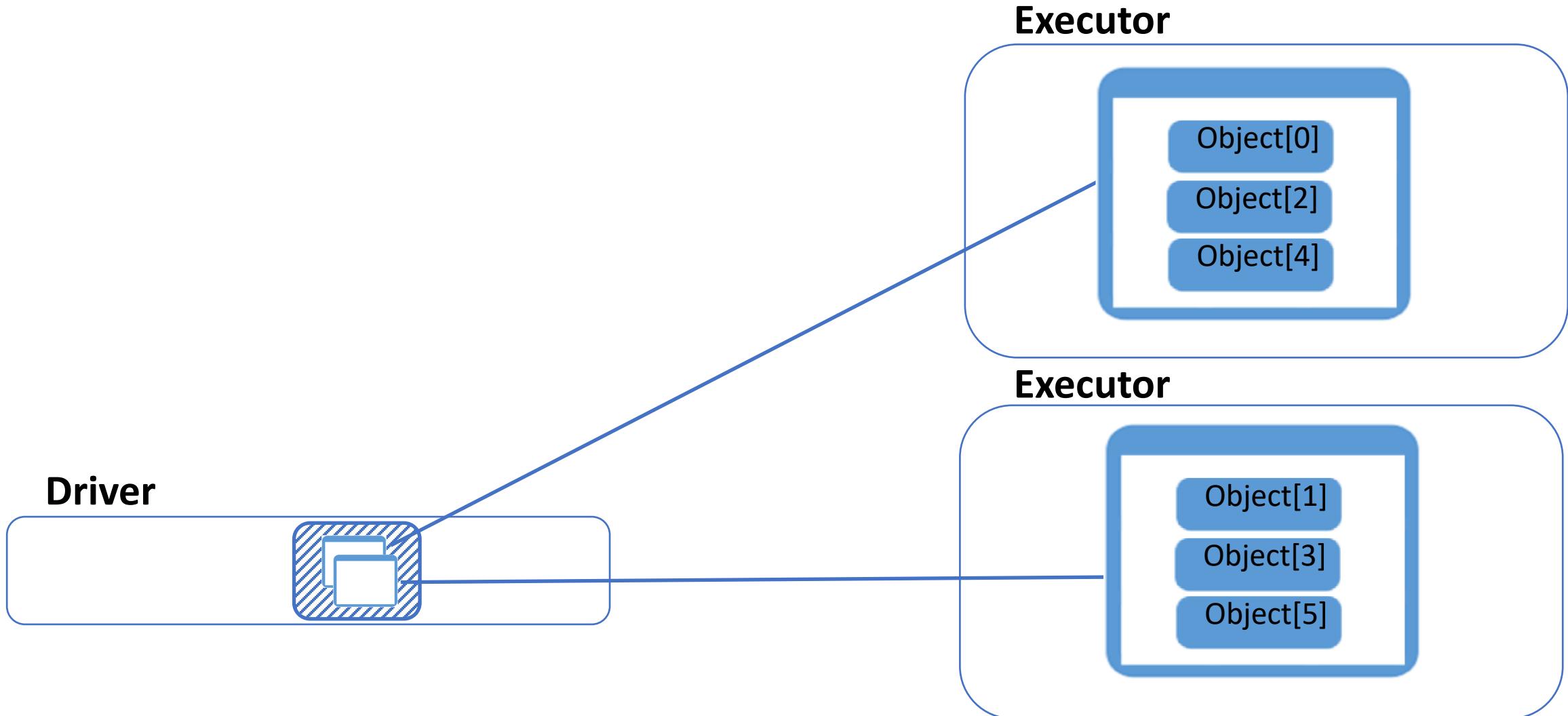
DataSet ... Partitions Distributed on Executors



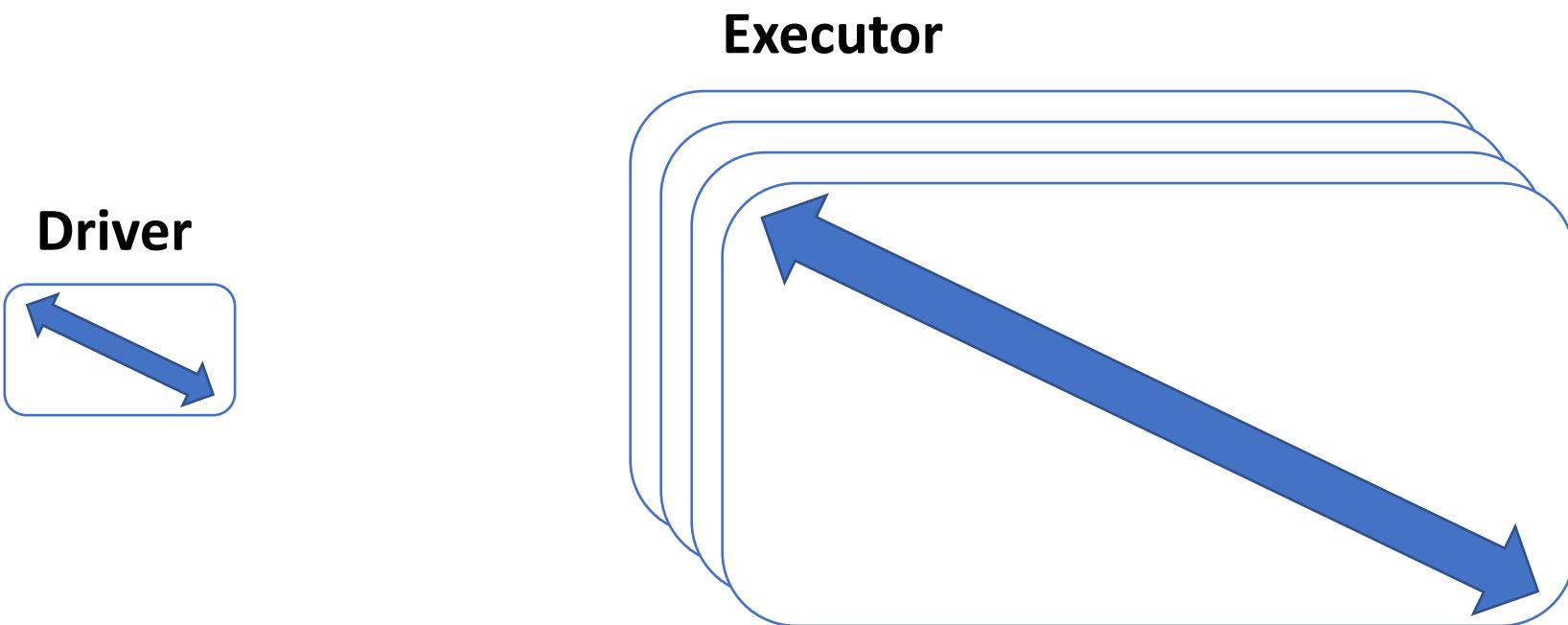
```
dataset = Spark.createDataset( inmemoryList )  
... serialize (send) data to Executors
```



Memory needs on Driver << Executors



Spark Memory Arguments



--driver-memory 1g

spark.driver.memory=1g

--executor-memory 50g

spark.executor.memory=50g

Spark Memory ...

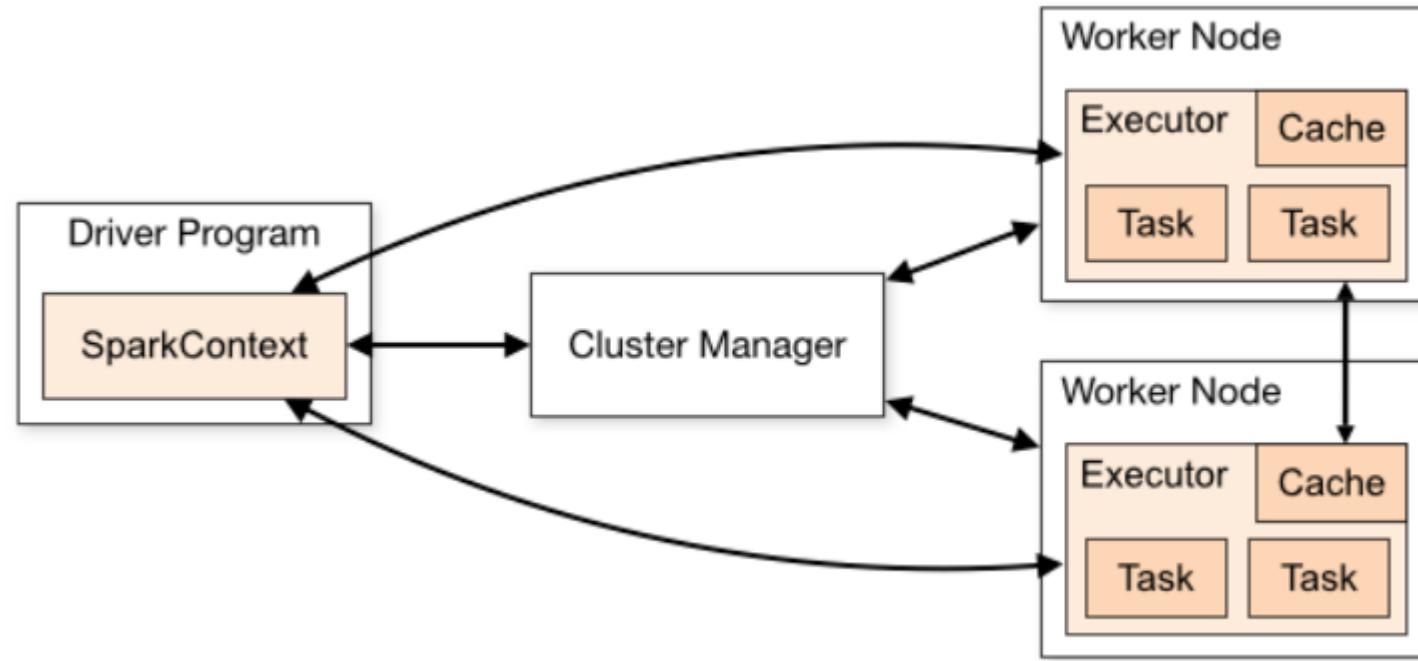
to {driver|executor} java –Xms .. –Xmx ..

spark.driver.memory	1g	<p>Amount of memory to use for the driver process, i.e. where SparkContext is initialized, in the same format as JVM memory strings with a size unit suffix ("k", "m", "g" or "t") (e.g. 512m, 2g).</p> <p><i>Note:</i> In client mode, this config must not be set through the <code>sparkConf</code> directly in your application, because the driver JVM has already started at that point. Instead, please set this through the <code>--driver-memory</code> command line option or in your default properties file.</p>
spark.executor.memory	1g	<p>Amount of memory to use per executor process, in the same format as JVM memory strings with a size unit suffix ("k", "m", "g" or "t") (e.g. 512m, 2g).</p>

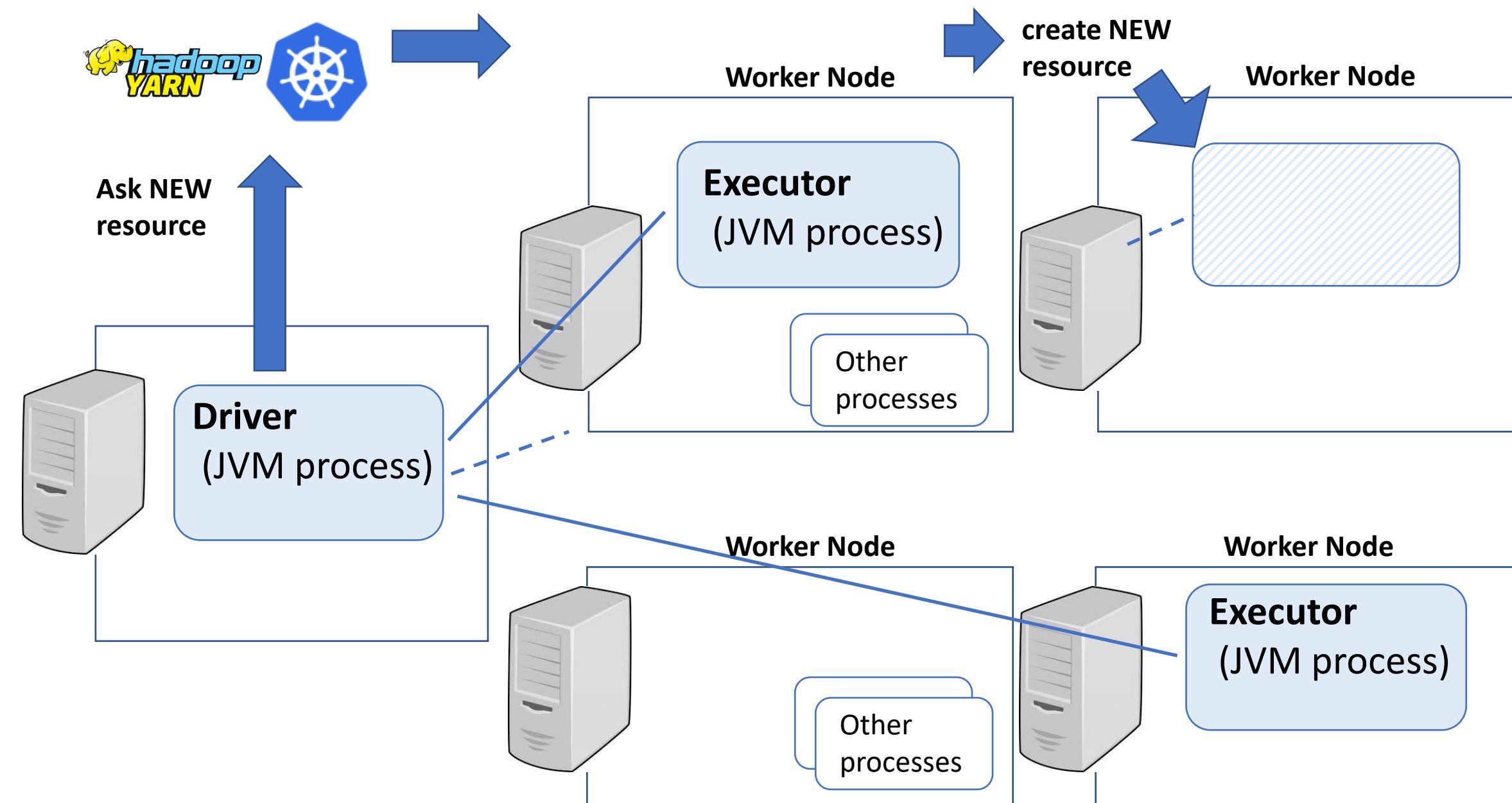
Use Cluster Resources :

Distributing N executors

Cluster Manager (SPI)



Executors .. JVM processes launched on Cluster



Driver ... Launch spark-executors

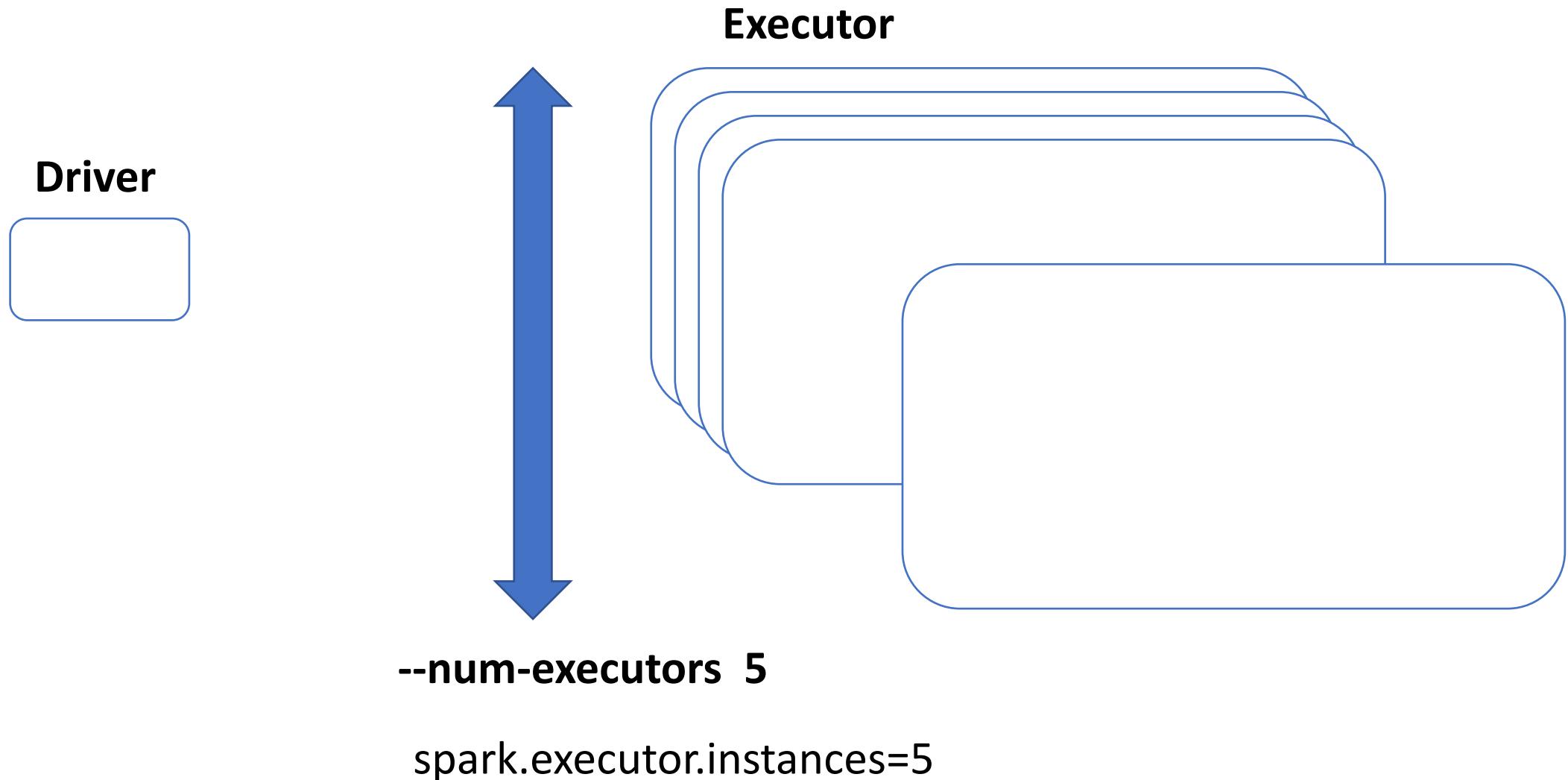
`${JAVA_HOME}/bin/java`

JVM args {
-server -Xmx30720m
-Djava.io.tmpdir=/yarn/local/usercache/<user>/appcache/<application_id>/<container_id>/tmp
-Dspark.history.ui.port=18080
-Dspark.driver.port=33029
-Dspark.yarn.app.container.log.dir /yarn/log/<application_id>/<container_id>/

Main class org.apache.spark.executor.**CoarseGrainedExecutorBackend**

main(
String[] args) {
--driver-url spark://CoarseGrainedScheduler@<driverhost>:33029
--executor-id 1
--hostname <executorhost>
--cores 5
--app-id <application_id>
--user-class-path file:/yarn/local/usercache/<user>/appcache/<application_id>/<container_id>/_app__.jar

Spark Cluster arguments



spark.executor.instances

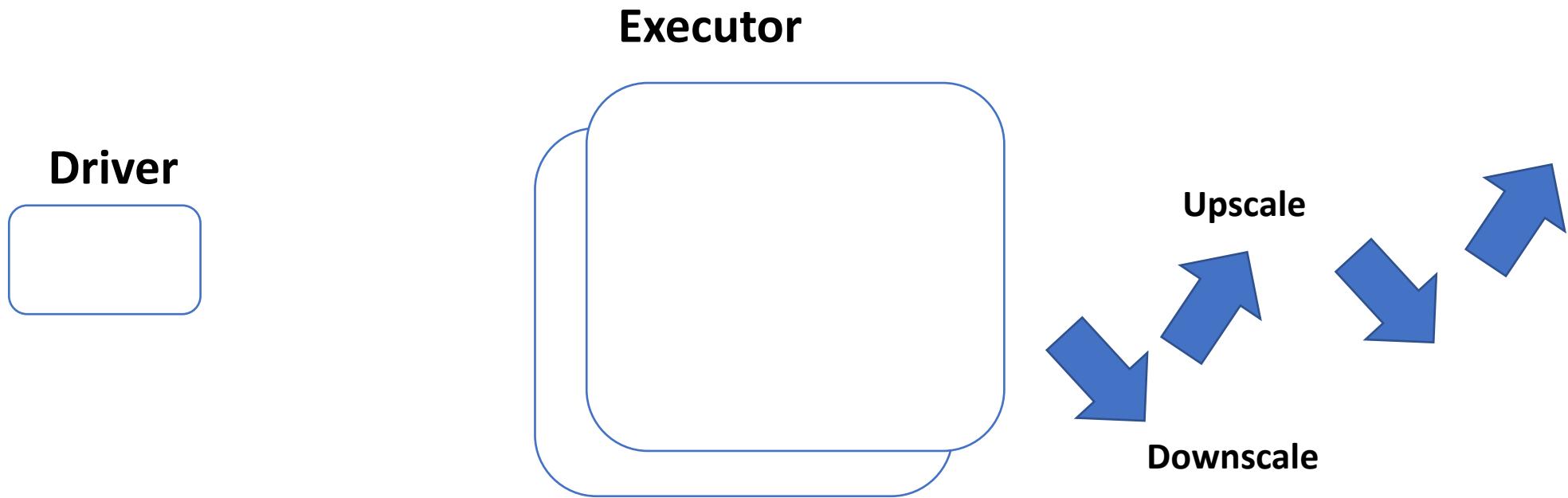
.. for Static allocation

spark.executor.instances	2	The number of executors for static allocation. With spark.dynamicAllocation.enabled, the initial set of executors will be at least this large.
--------------------------	---	--



.. See also Dynamic Allocation

Dynamic Allocation



--spark.dynamic.allocation.enabled true

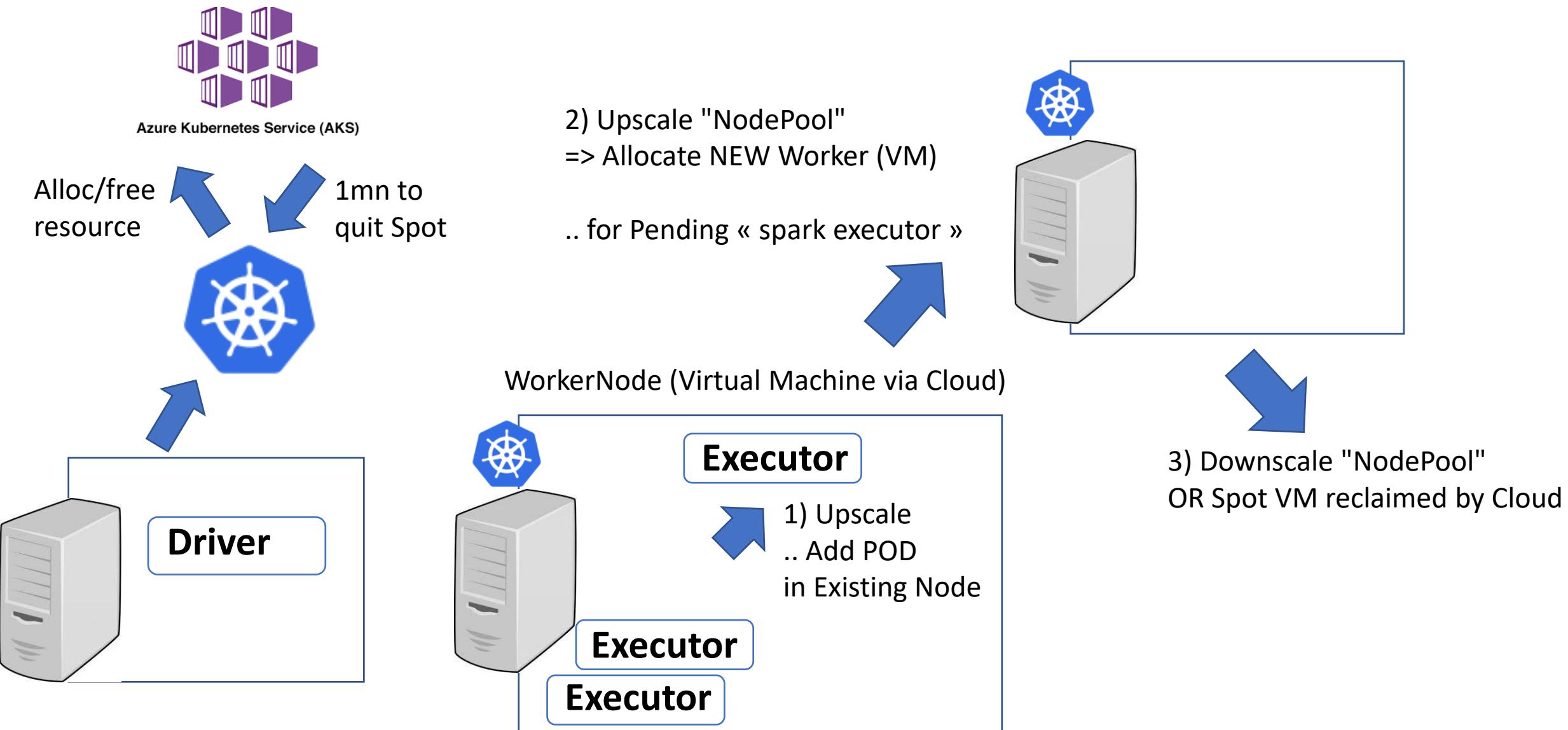
--spark.dynamic.allocation.{min | max | initialExecutors | executorIdleTimeout | ... }

Dynamic Allocation arguments

Dynamic Allocation

Property Name	Default	Meaning	Since Version
spark.dynamicAllocation.enabled	false	<p>Whether to use dynamic resource allocation, which scales the number of executors registered with this application up and down based on the workload. For more detail, see the description here.</p> <p>This requires spark.shuffle.service.enabled or spark.dynamicAllocation.shuffleTracking.enabled to be set. The following configurations are also relevant: spark.dynamicAllocation.minExecutors, spark.dynamicAllocation.maxExecutors, and spark.dynamicAllocation.initialExecutors</p> <p>spark.dynamicAllocation.executorAllocationRatio</p>	1.2.0
spark.dynamicAllocation.executorIdleTimeout	60s	If dynamic allocation is enabled and an executor has been idle for more than this duration, the executor will be removed. For more detail, see this description .	1.2.0
spark.dynamicAllocation.cachedExecutorIdleTimeout	infinity	If dynamic allocation is enabled and an executor which has cached data blocks has been idle for more than this duration, the executor will be removed. For more details, see this description .	1.4.0
spark.dynamicAllocation.initialExecutors	spark.dynamicAllocation.minExecutors	<p>Initial number of executors to run if dynamic allocation is enabled.</p> <p>If `--num-executors` (or `spark.executor.instances`) is set and larger than this value, it will be used as the initial number of executors.</p>	1.3.0
spark.dynamicAllocation.maxExecutors	infinity	Upper bound for the number of executors if dynamic allocation is enabled.	1.2.0
spark.dynamicAllocation.minExecutors	0	Lower bound for the number of executors if dynamic allocation is enabled.	1.2.0
spark.dynamicAllocation.executorAllocationRatio	1	By default, the dynamic allocation will request enough executors to maximize the parallelism according to the number of tasks to process. While this minimizes the latency of the job, with small tasks this setting can waste a lot of resources due to executor allocation overhead, as some executor might not even do any work. This setting allows to set a ratio that will be used to reduce the number of executors w.r.t. full parallelism. Defaults to 1.0 to give maximum parallelism. 0.5 will divide the target number of executors by 2. The target number of	2.4.0

Dynamic Spark + Dynamic K8s Cloud Cluster



Spark dynamicAllocation <=> K8s NodePool autoScaling

=> more (system) VMs from Cloud Provider

=> when full: NodePool "TriggerScaleUp" (VM provisionning may take ~3mn)

=> need more k8s resources

allocating new (applicative) "executor"

releasing (applicative) "executor"

=> use less k8s resources

=> when empty (0 running pod on Node), VM could be evicted from NodePool
(example: configure with 15mn delay)

=> less (system) VMs from Cloud Provider

K8s NodePool, Afinity, Toleration, Spot VM ..

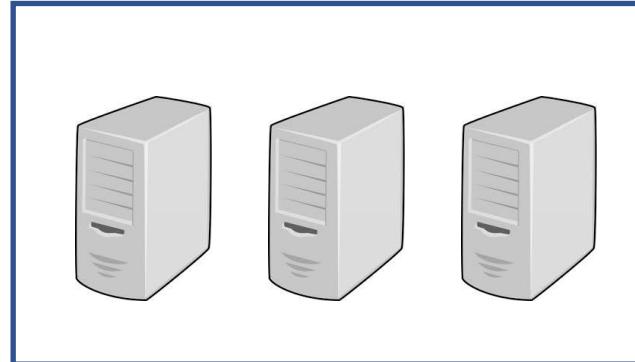


Azure Kubernetes Service (AKS)



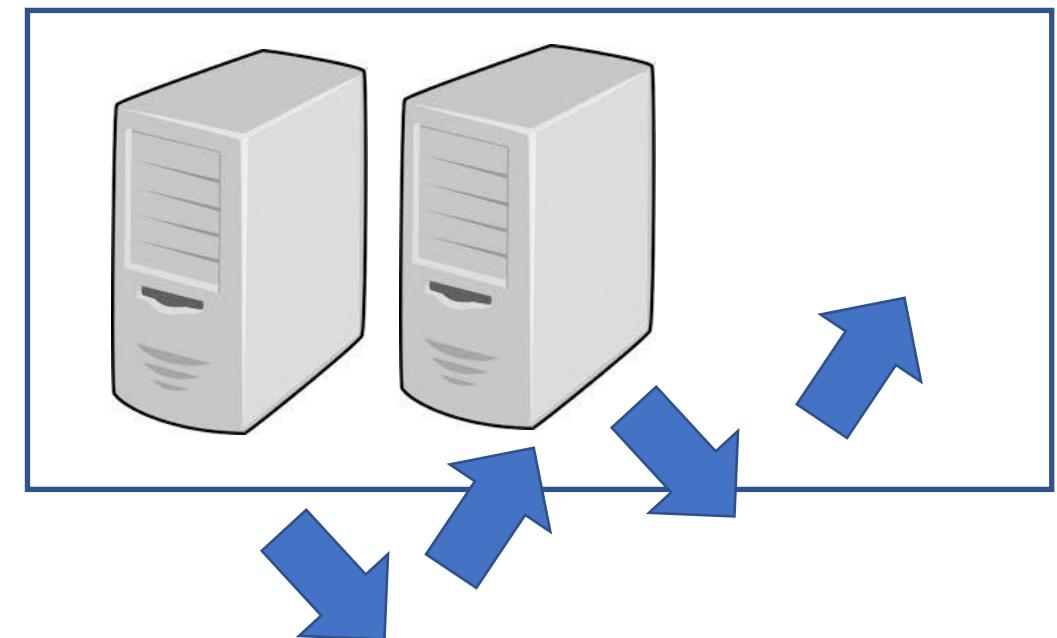
Node Pool 1

Small VMs ... reserved
=> for long running services, spark Drivers



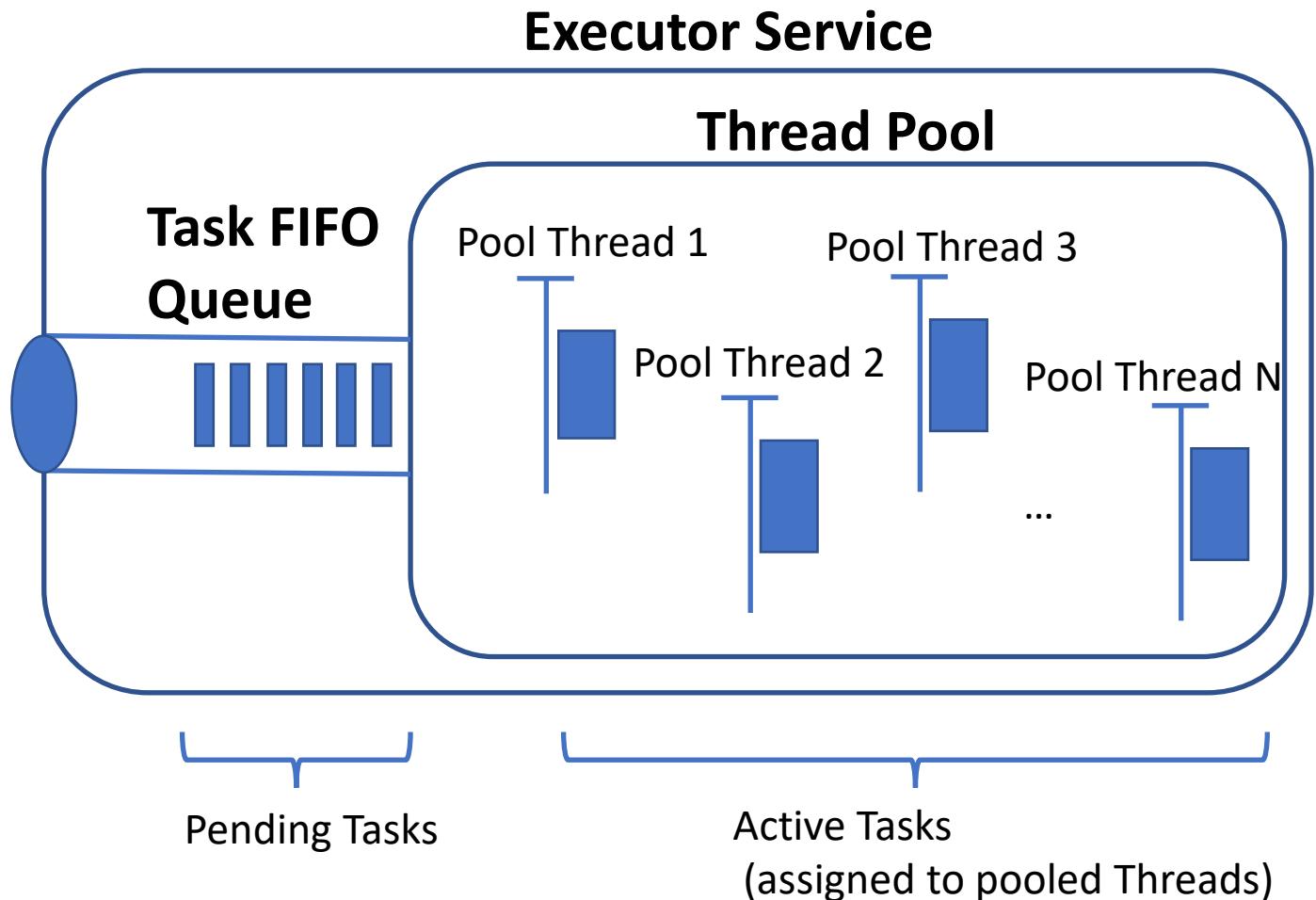
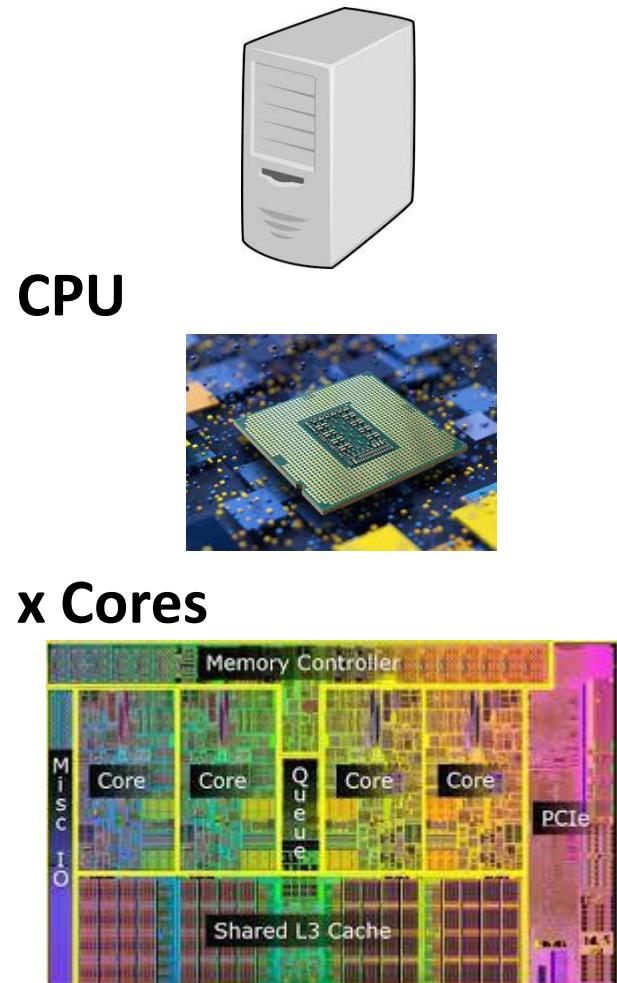
Node Pool 2

Big VMs ... « SPOT »
(cheaper, may be reclaimed by cloud)
=> for spark Executors

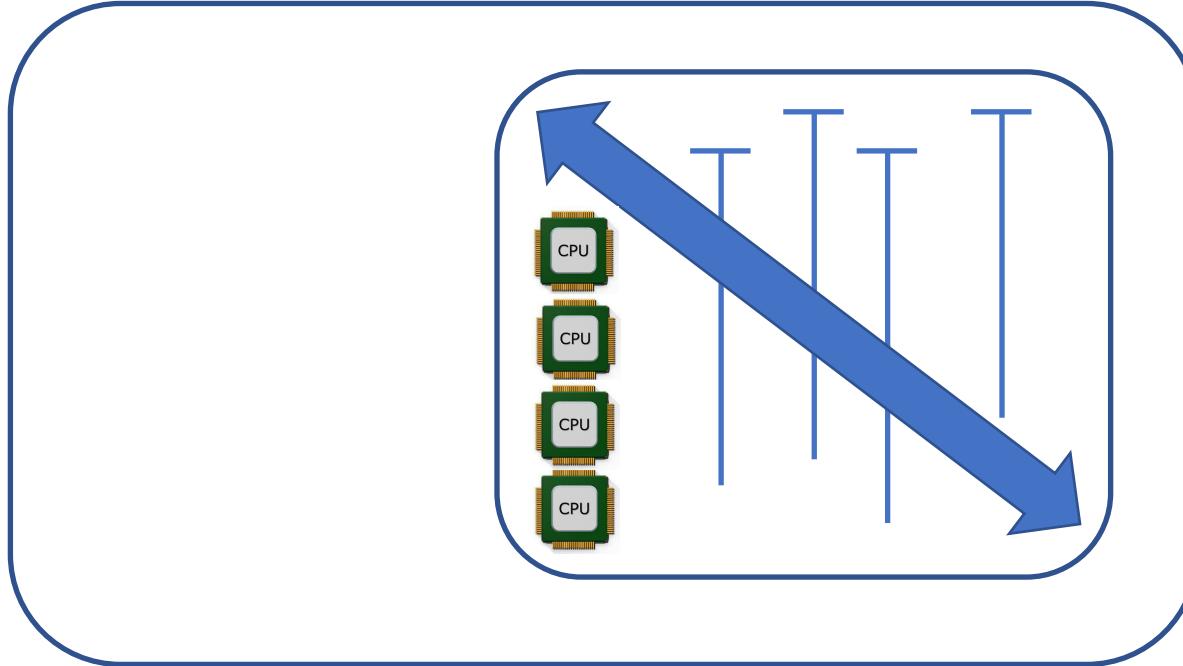


Use CPU Resource: 100% or sharing ?

Cpu Cores <-> Threads <-> max Active Tasks



Executor Cores argument



`spark.executor.cores`

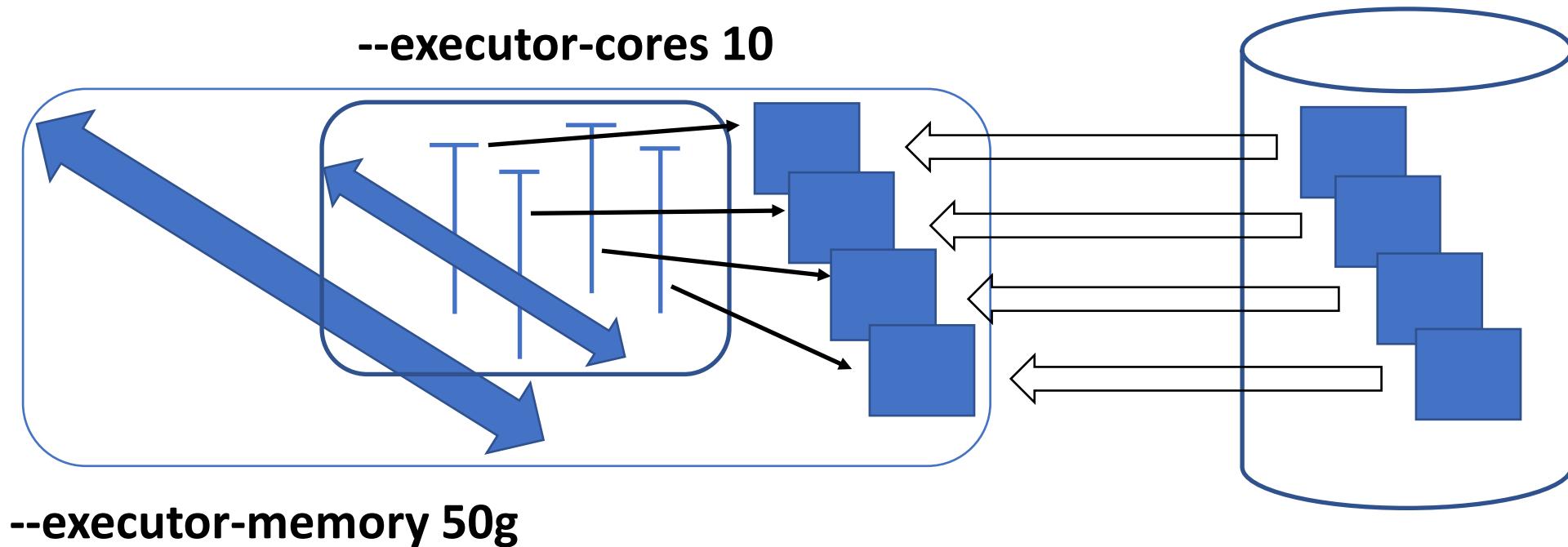
1 in YARN mode, all the available cores on the worker in standalone and Mesos coarse-grained modes.

The number of cores to use on each executor. In standalone and Mesos coarse-grained modes, for more detail, see [this description](#).

--executor-cores N

spark.executor.cores=N

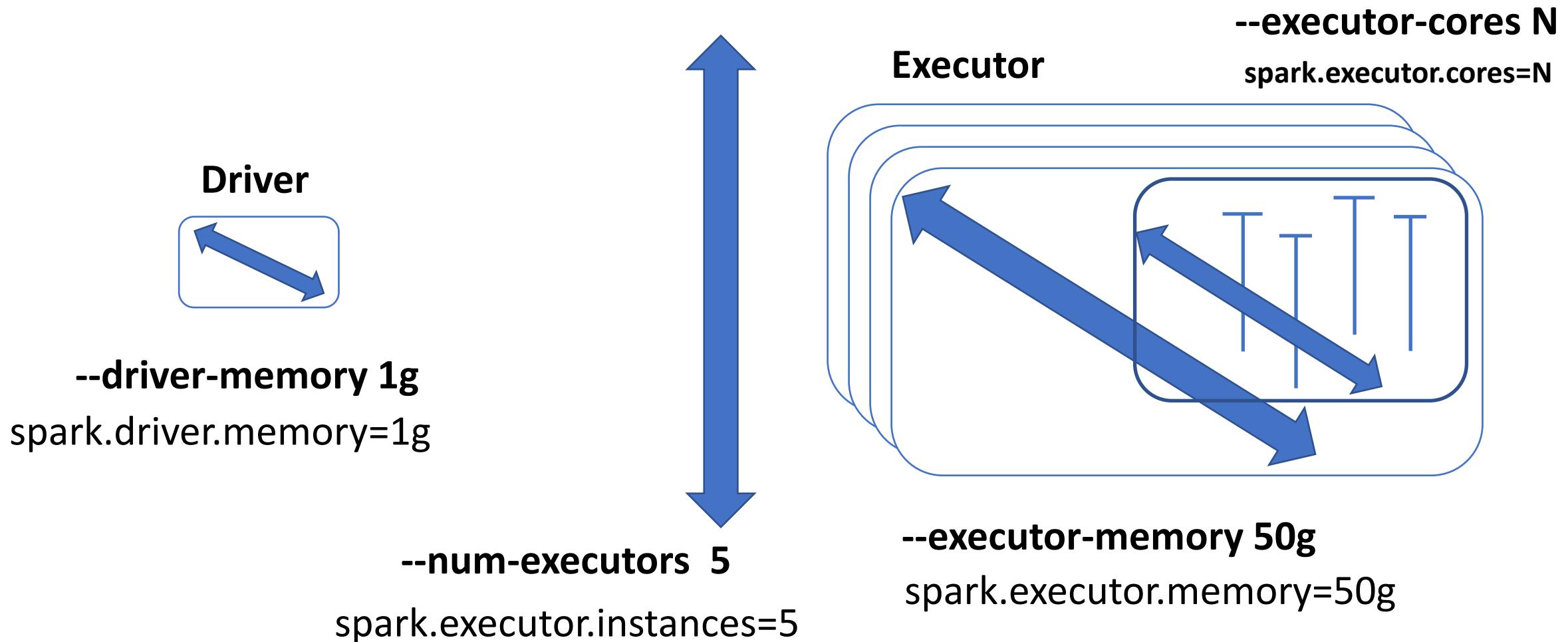
Relationship executor .cores <-> .memory



1 Active Core => 1 Task => read 1 File Block => 1 Dataset Partition

=> ... maybe 1 x 256 Mo (compressed) ... 1 x Go ?

Summary main arguments



Outline



Reminder Spark-driver / spark-executors



Cluster Tuning Params



deployMode (local, spark://, yarn, k8s)

Others launchers (java embedded, livy, jupyter, pyspark ..)

```
$ spark-submit --master ${mode} ...
```

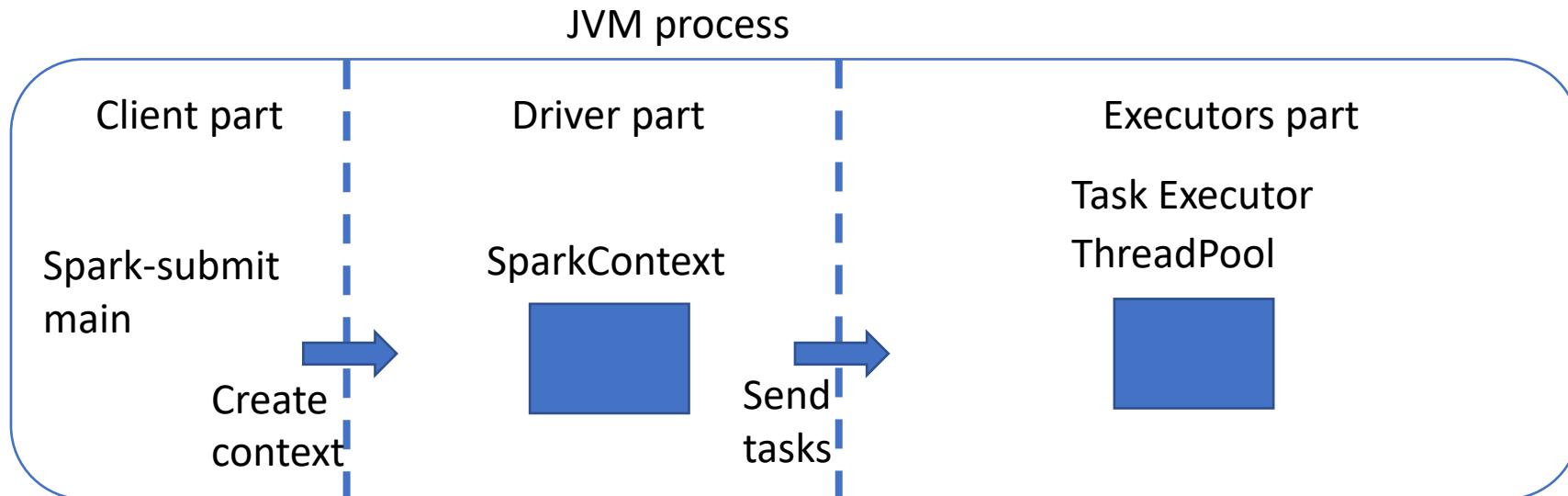
`\${mode}`: one of

- **local[N]**
- **mesos (deprecated)**
- **spark://**
- **yarn**
- **k8s**

--master local[*]

```
# Run application locally on 8 cores
./bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master local[8] \
/path/to/examples.jar \
100
```

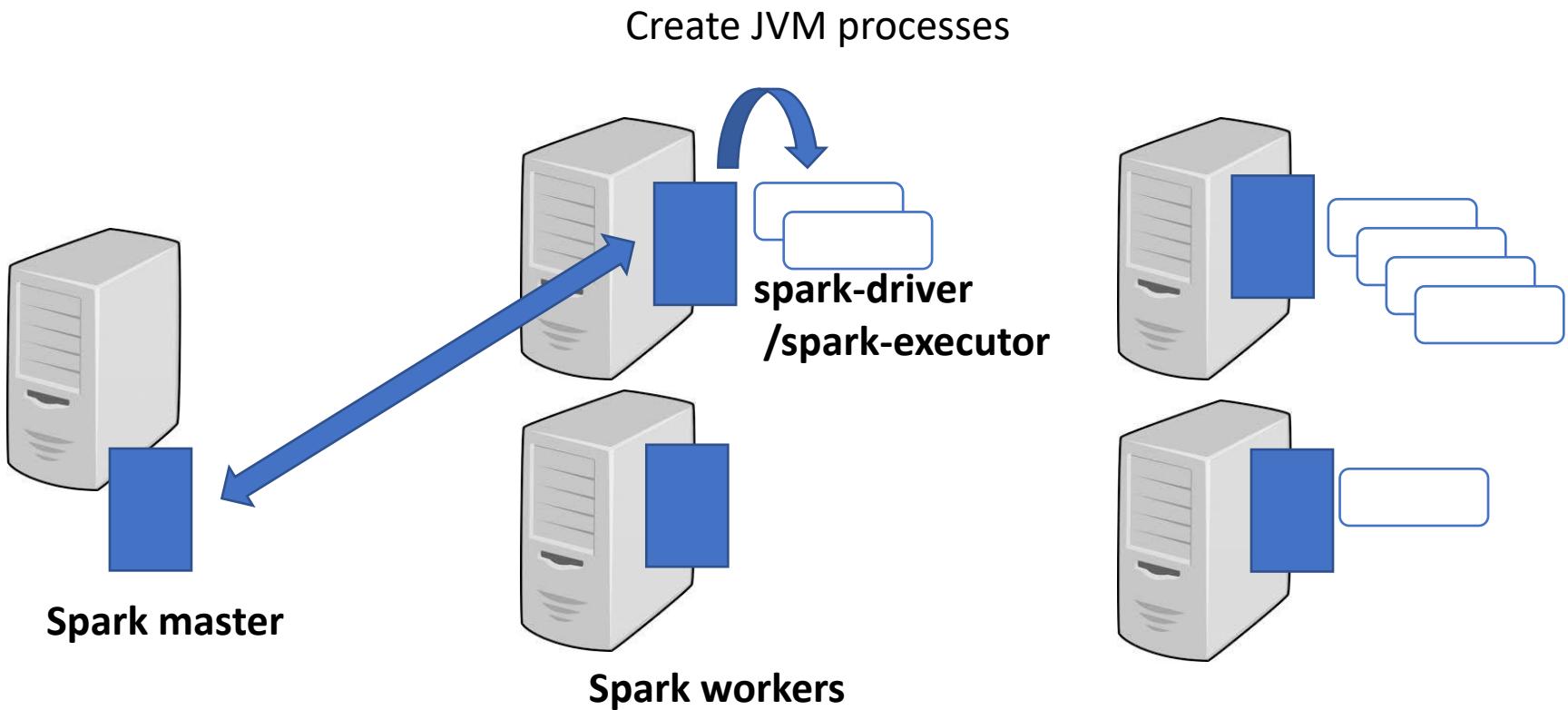
Local mode: 1 JVM client = driver = executors



--master spark://<sparkServer>:<port>

```
# Run on a Spark standalone cluster in client deploy mode
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://207.184.161.138:7077 \
  --executor-memory 20G \
  --total-executor-cores 100 \
  /path/to/examples.jar \
  1000
```

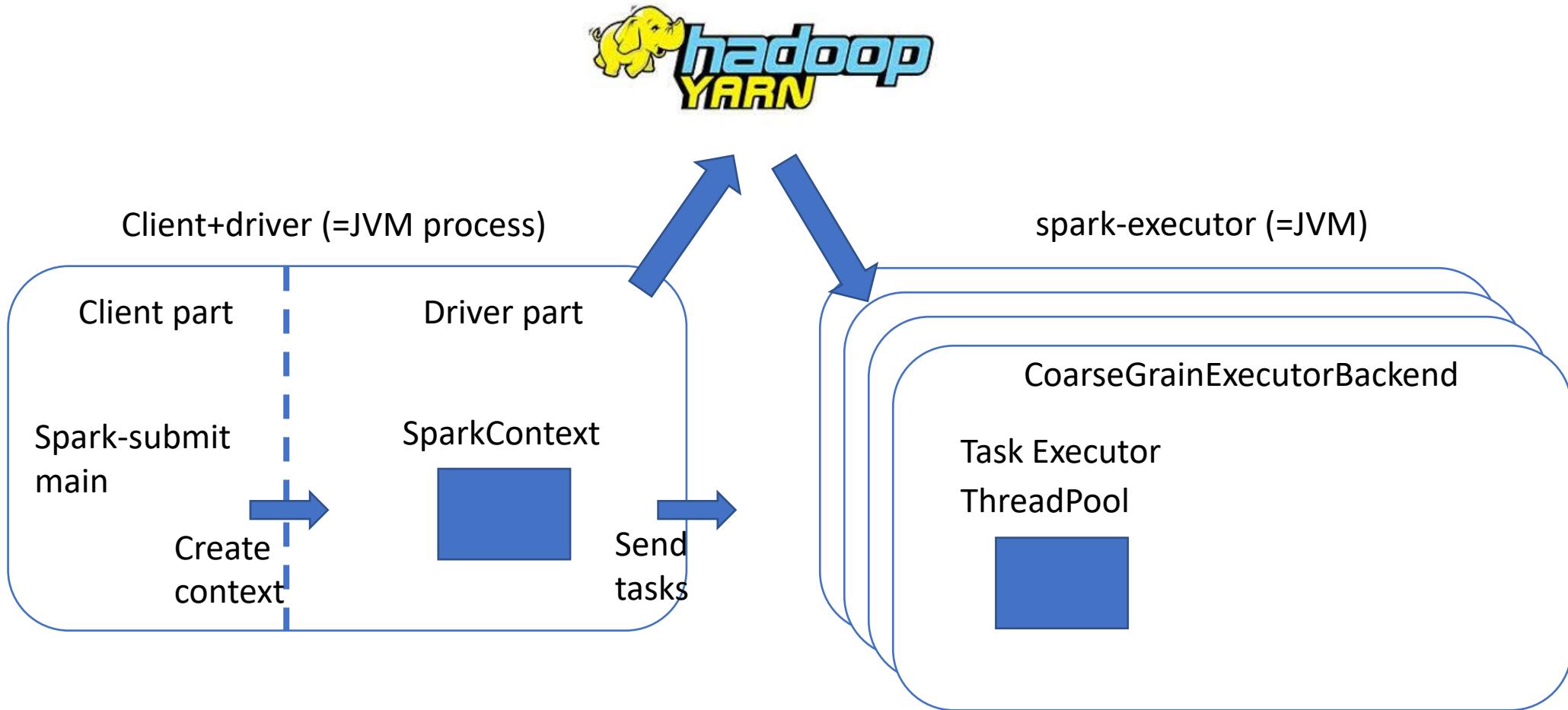
Spark Standalone Cluster



--master yarn --deploy-mode client

```
# Run on a YARN cluster in client deploy mode
export HADOOP_CONF_DIR=XXX
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master yarn \
  --deploy-mode client \
  --executor-memory 20G \
  --num-executors 50 \
  /path/to/examples.jar \
  1000
```

Yarn Client Mode



Pros/Cons

Easy to start: spark-submit.sh

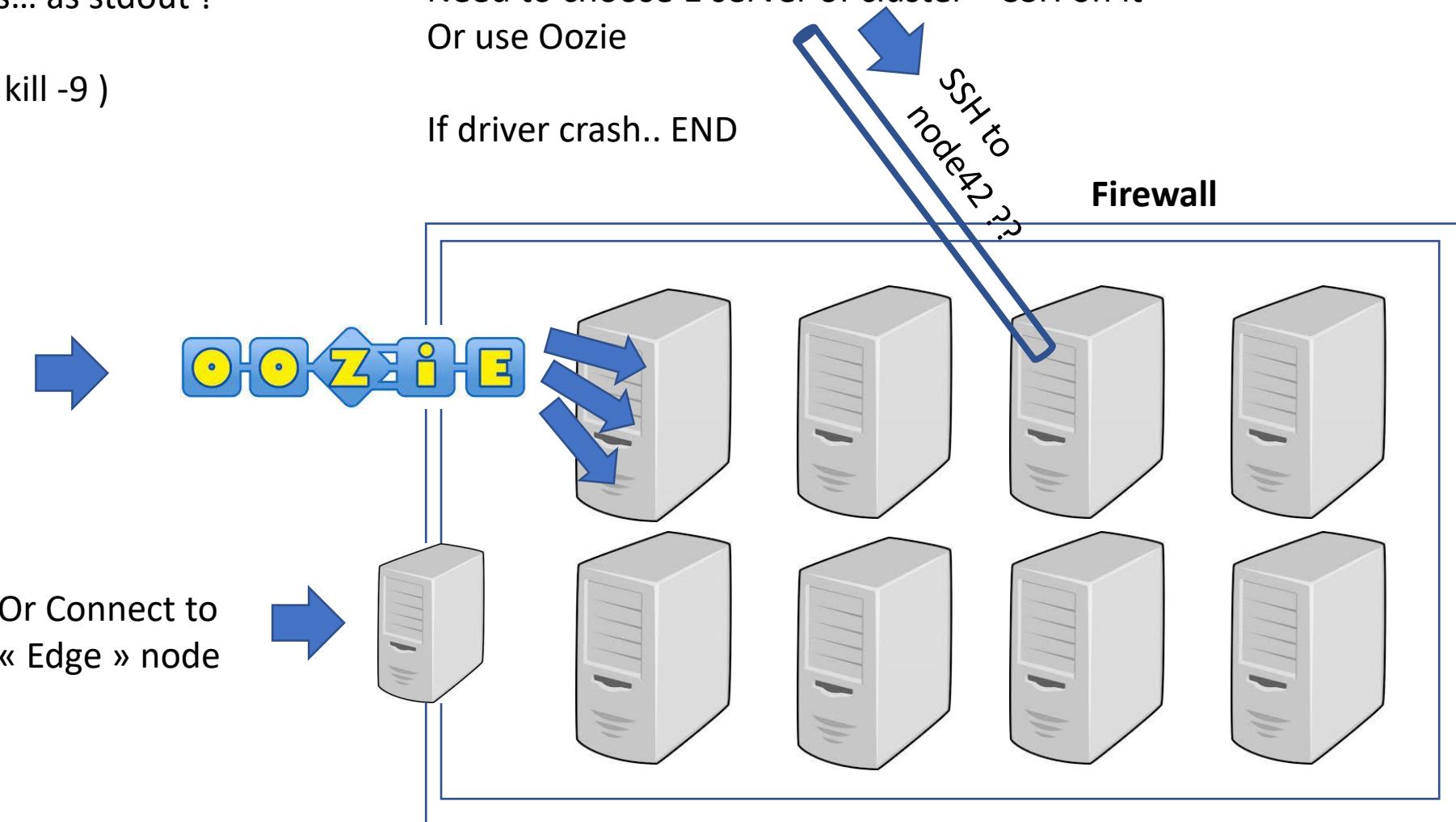
Easy to see driver logs... as stdout !

Easy to kill (Ctrl+C / kill -9)

Client = Driver ... within firewall

Need to choose 1 server of cluster + SSH on it
Or use Oozie

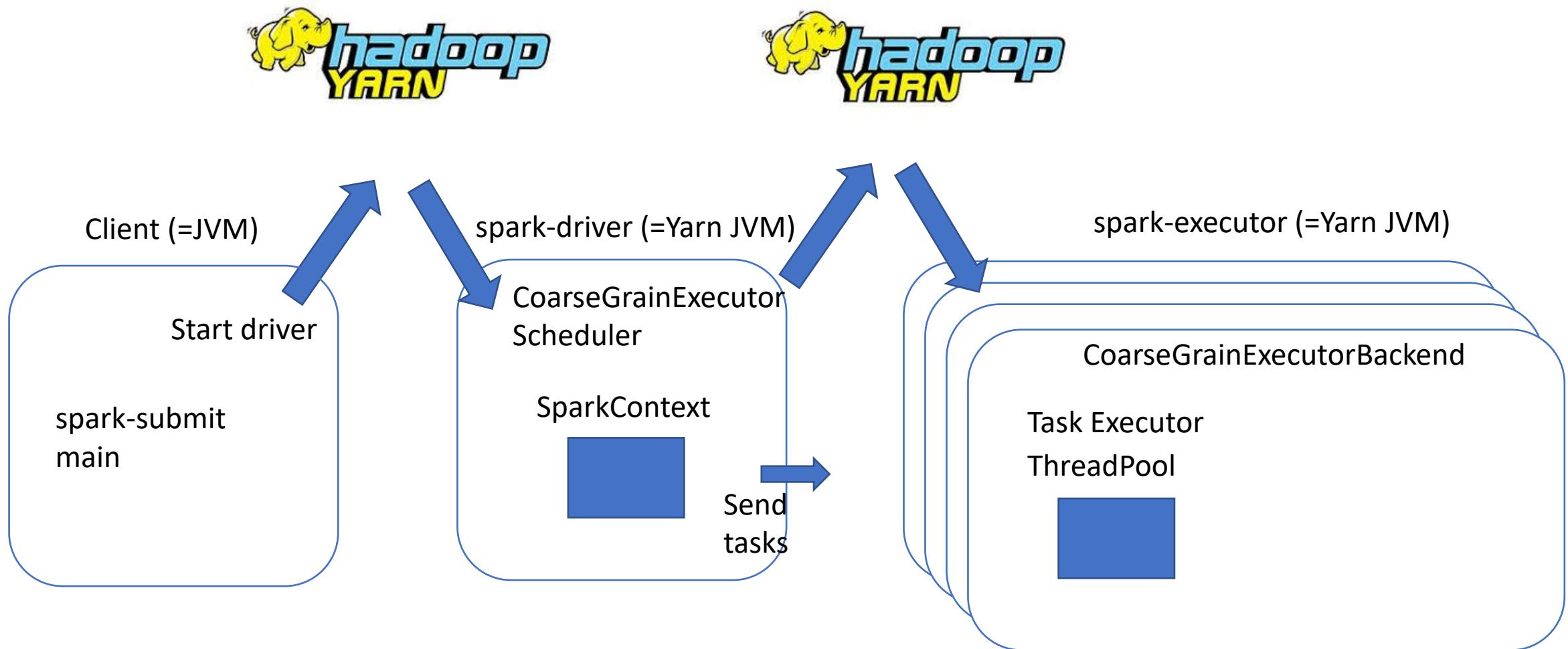
If driver crash.. END



--master yarn --deploy-mode cluster

```
# Run on a YARN cluster in cluster deploy mode
export HADOOP_CONF_DIR=XXX
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master yarn \
  --deploy-mode cluster \
  --executor-memory 20G \
  --num-executors 50 \
  /path/to/examples.jar \
  1000
```

Yarn Cluster Mode



Pros-Cons Yarn « Cluster » vs « Client » mode

Client can run outside firewall / cluster

More difficult to see logs (Yarn Logs!!)

Client is lightweight

More difficult to kill : yarn app –kill \${applicationId}

Many Clients can run on « Edge » Node

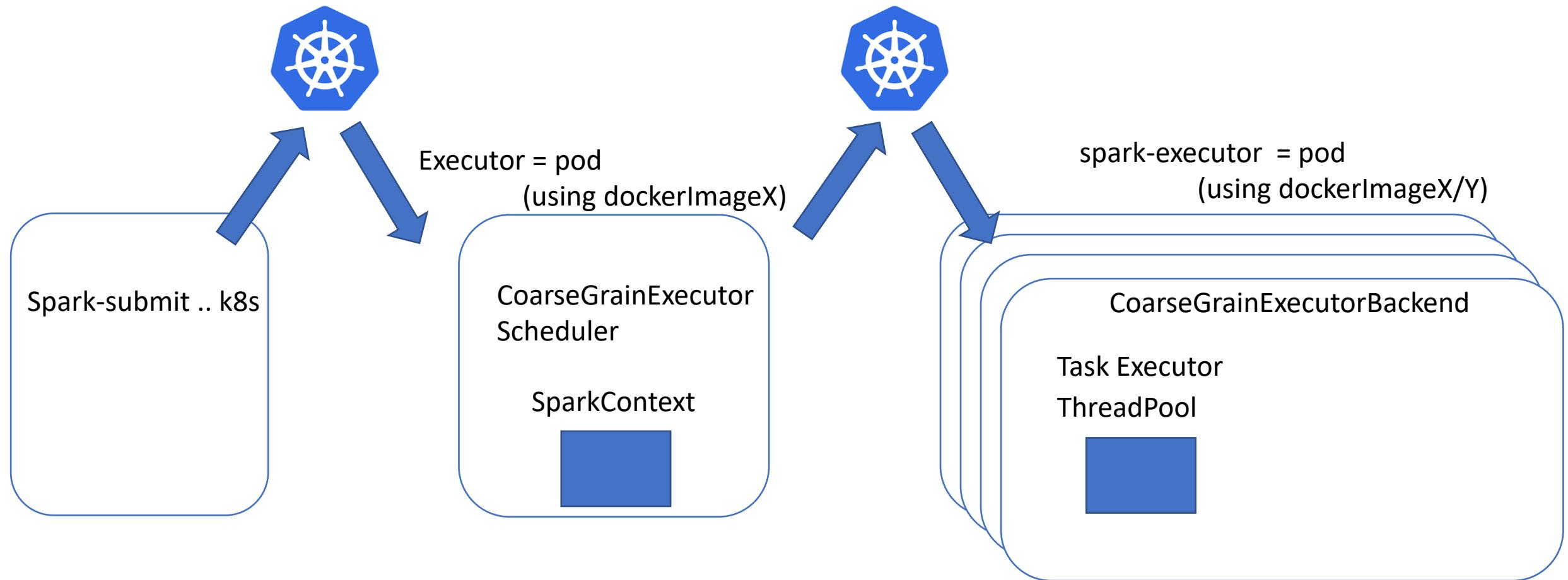
Kill client => does not kill driver !!
(Zombie yarn driver ?)

If Driver crashes... client will relaunch driver!

--k8s://<k8sApiServer>:443 (basic without K8s operator)

```
# Run on a Kubernetes cluster in cluster deploy mode
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master k8s://xx.yy.zz.ww:443 \
  --deploy-mode cluster \
  --executor-memory 20G \
  --num-executors 50 \
  http://path/to/examples.jar \
  1000
```

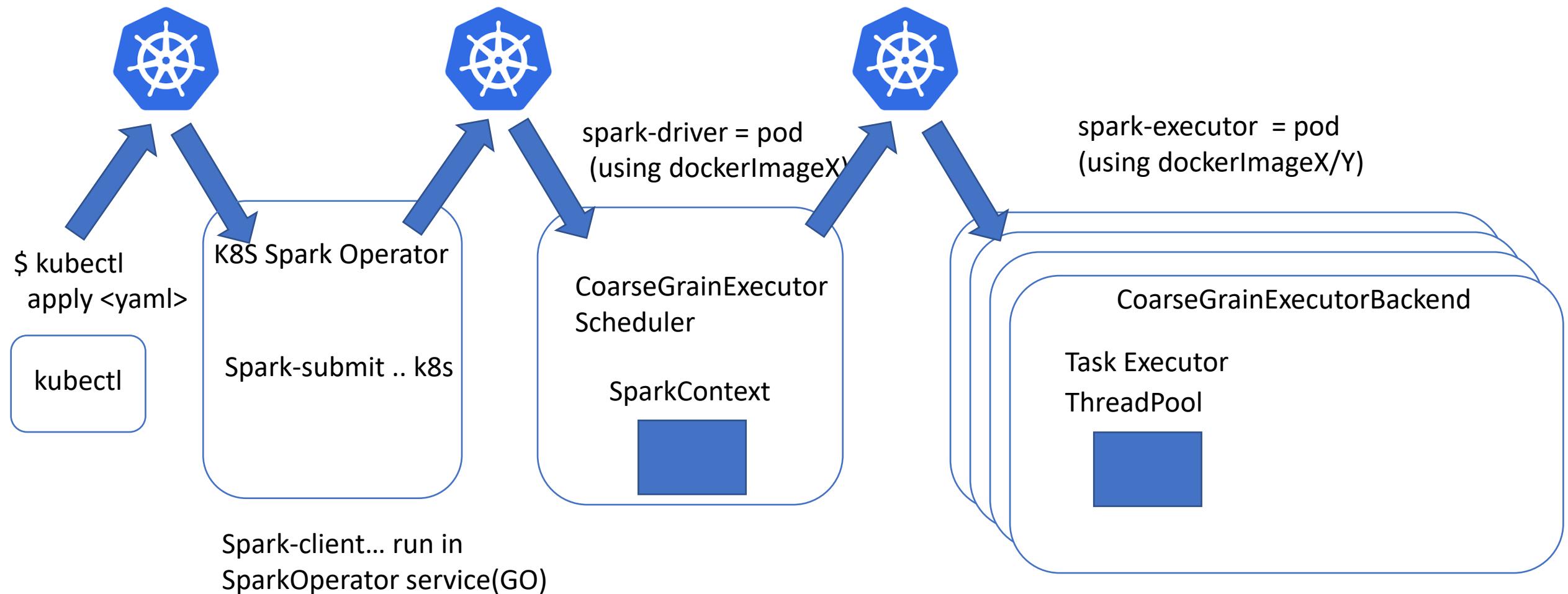
Spark on Kubernetes ... Docker Images/Jars



Spark on K8s with SparkOperator mode

```
$ kubectl apply -f examples/spark-pi.yaml
```

Spark on K8S, using K8s SparkOperator



spark-on-k8s-operator : K8s CRD from Google

The screenshot shows the GitHub repository page for `GoogleCloudPlatform/spark-on-k8s-operator`. The repository is public, has 82 watchers, 1.1k forks, and 2.2k stars. It contains 366 issues, 46 pull requests, and 46 actions. The `Code` tab is selected, showing the master branch with 5 branches and 63 tags. The commit history lists several updates, including one from ImpSy and multiple commits from the .github/workflows directory. The repository is described as a Kubernetes operator for managing Apache Spark applications on Kubernetes. It uses labels for kubernetes, spark, apache-spark, kubernetes-operator, kubernetes-controller, kubernetes-crd, and google-cloud-dataproc.

Search or jump to... / Pull requests Issues Marketplace Explore

GoogleCloudPlatform / `spark-on-k8s-operator` Public Watch 82 Fork 1.1k Star 2.2k

Code Issues 366 Pull requests 46 Actions Projects Security Insights

master 5 branches 63 tags Go to file Add file Code

ImpSy update go to 1.19 + k8s.io libs to v0.25.3 (#1630) ✓ c261df6 7 days ago 850 commits

.github/workflows update go to 1.19 + k8s.io libs to v0.25.3 (#1630) 7 days ago

charts/spark-operator-chart update go to 1.19 + k8s.io libs to v0.25.3 (#1630) 7 days ago

docs Update README - secrets and sidecars need mutating webhooks (#1550) 5 months ago

examples Updating API version of admissionregistration.k8s.io (#1401) 11 months ago

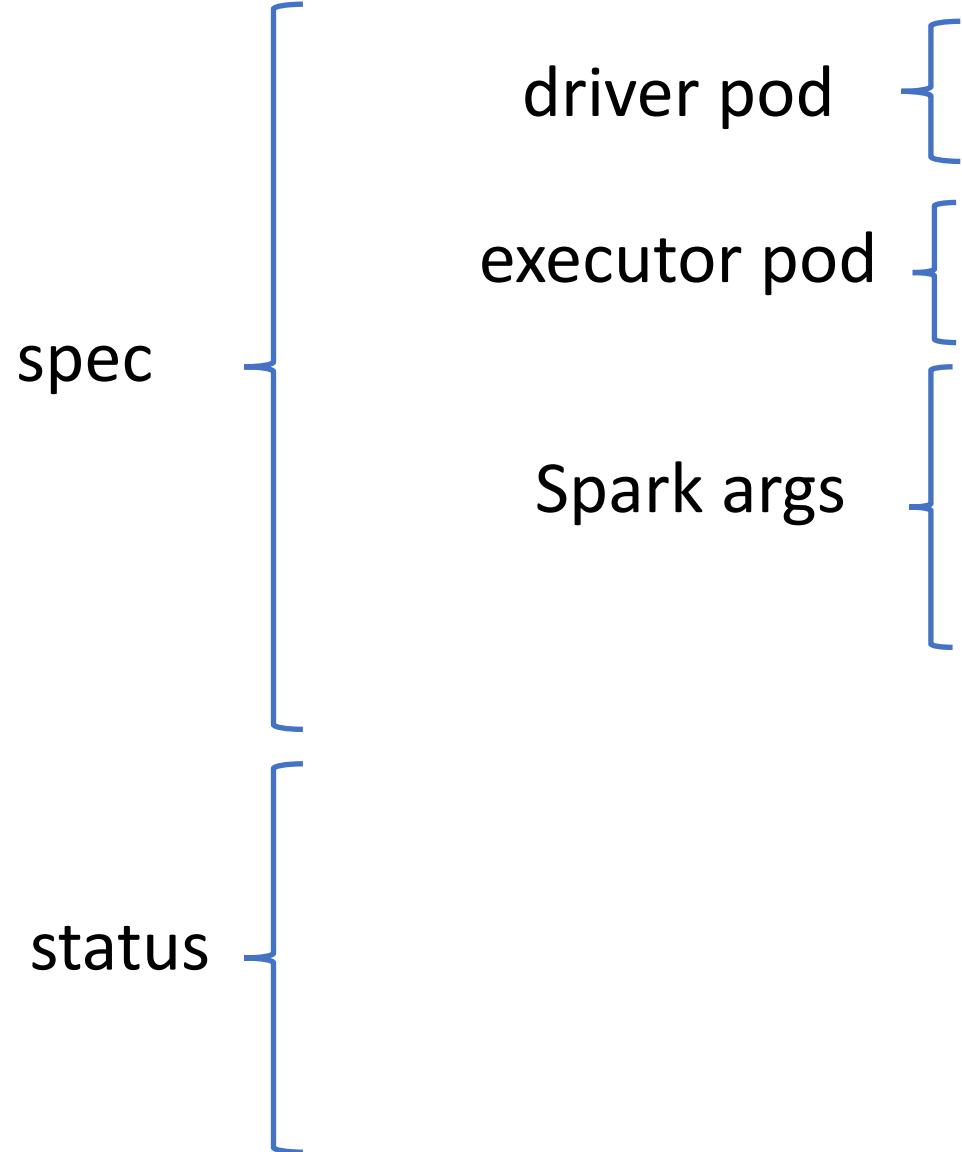
About

Kubernetes operator for managing the lifecycle of Apache Spark applications on Kubernetes.

kubernetes spark apache-spark
kubernetes-operator kubernetes-controller
kubernetes-crd google-cloud-dataproc

Readme

```
$ kubectl get  
sparkapplications .. -o=yaml
```



```
apiVersion: sparkoperator.k8s.io/v1beta2  
kind: SparkApplication  
metadata:  
    ...  
spec:  
    deps: {}  
    driver:  
        coreLimit: 1200m  
        cores: 1  
        labels:  
            version: 2.3.0  
            memory: 512m  
            serviceAccount: spark  
        executor:  
            cores: 1  
            instances: 1  
            labels:  
                version: 2.3.0  
                memory: 512m  
            image: gcr.io/ynli-k8s/spark:v3.1.1  
            mainApplicationFile: local:///opt/spark/examples/jars/spark-examples_2.12-3.1.1.jar  
            mainClass: org.apache.spark.examples.SparkPi  
            mode: cluster  
            restartPolicy:  
                type: OnFailure  
                onFailureRetries: 3  
                onFailureRetryInterval: 10  
                onSubmissionFailureRetries: 5  
                onSubmissionFailureRetryInterval: 20  
            type: Scala  
    status:  
        sparkApplicationId: spark-5f4ba921c85ff3f1cb04bef324f9154c9  
        applicationState:  
            state: COMPLETED  
        completionTime: 2018-02-20T23:33:55Z  
        driverInfo:  
            podName: spark-pi-83ba921c85ff3f1cb04bef324f9154c9-driver  
            webUIAddress: 35.192.234.248:31064  
            webUIPort: 31064  
            webUIServiceName: spark-pi-2402118027-ui-svc  
            webUIIngressName: spark-pi-ui-ingress  
            webUIIngressAddress: spark-pi.ingress.cluster.com  
        executorState:  
            spark-pi-83ba921c85ff3f1cb04bef324f9154c9-exec-1: COMPLETED  
    LastSubmissionAttemptTime: 2018-02-20T23:32:27Z
```

Outline



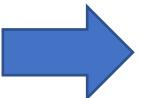
Reminder Spark-driver / spark-executors



Cluster Tuning Params

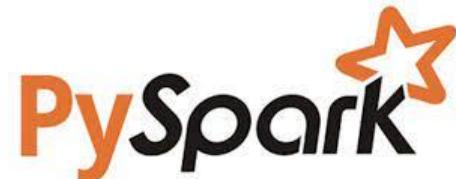


deployMode (local, spark://, yarn, k8s)



Others launchers (java embedded, livy, jupyter, pyspark ..)

More ways to start spark





Plain old java main(String[] args)

Spark « Hello World »

```
6 import org.apache.spark.sql.Dataset;
7 import org.apache.spark.sql.Encoders;
8 import org.apache.spark.sql.SparkSession;
9
10 public class SimpleMain {
11
12     public static void main(String[] args) {
13         System.out.println("ensure HADOOP_HOME env var is set, (and Winutils on windows)");
14         System.out.println("HADOOP_HOME: " + System.getenv("HADOOP_HOME"));
15
16         System.out.println("start embedded SparkContext - SparkSession");
17         SparkSession spark = SparkSession.builder()
18             .master("local[*]")
19             .appName("myapp")
20             .getOrCreate();
21
22         List<String> ls = Arrays.asList("Hello spark");
23         Dataset<String> ds = spark.createDataset(ls, Encoders.STRING());
24         ds.show();
25
26         System.out.println("finish SparkContext");
27         spark.close();
28     }
29 }
```

The screenshot shows a Java application running in an IDE. The code is a simple "Hello World" application using Apache Spark's Dataset API. It prints environment variables, creates a SparkSession, and shows a dataset containing the string "Hello spark". The execution output in the console window shows the expected results.

```
Console × Problems Progress Error Log Debug Shell Search Call Hierarchy Search
<terminated> SimpleSparkMain [Java Application] C:\apps\jdk\jdk-8\bin\javaw.exe (19 nov. 2022, 20:20:06 – 20:20:14) [pid: 18140]
ensure HADOOP_HOME env var is set, (and Winutils on windows)
HADOOP_HOME: C:\apps\hadoop\spark-3.3.0-hadoop-3.3
start embedded SparkContext - SparkSession
+-----+
|      value|
+-----+
|Hello spark|
+-----+
finish SparkContext
```



Spark maven pom.xml <dependency>

```
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-core_${scala.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.apache.spark</groupId>
    <artifactId>spark-sql_${scala.version}</artifactId>
    <version>${spark.version}</version>
    <scope>provided</scope>
</dependency>
```

Use <scope>provided</scope>
when deployed in HADOOP cluster



SparkContext / SparkSession ?

SparkContext = in module spark-core
singleton, instanciable only on Driver, multi-thread safe

SparkSession = in module spark-sql
also multi-thread safe (~ SparkContext + SQL UDFs, temporaryViews..)

When creating a SparkSession, it internally check/create SparkContext

It is possible to have multiple sessions
(for having different SQL UDF, temporaryView, .. per user session)



SparkConf – SparkContext - SparkSession

```
class SparkConf {  
... key=value  
+ hadoopConf  
}
```

package org.apache.spark (module spark-core)

```
class SparkContext {  
...  
}
```

new RDD + transforms

1

activeContext

0..1

package org.apache.spark.sql (module spark-sql)

```
class SparkSession {  
...  
}
```

getOrCreate()

new Dataset<> + transforms



JavaSparkContext, JavaRDD, JavaFunction ?

Because Spark has a cleaner API in Scala
(less verbose, using implicits + operator overload),

some specific helper classes have been added, for Java developpers

some deprecated, ex: SQLContext



Oozie ... http Api wrapper for Yarn + UI

```
$ curl http://<knox>/oozie/
```

The screenshot shows a web browser window for the Oozie Web Console at the URL `127.0.0.1:11000/oozie/`. The title bar indicates the page is titled "Job (Name: My_first_Workflow/JobId: 0000007-150727083427440-oozie-oozi-W)". The main content area is divided into two sections: "Job Info" and "Actions".

Job Info:

- Job Id: 0000007-150727083427440-oozie-oozi-W
- Name: My_first_Workflow
- App Path: hdfs://sandbox.hortonworks.com:8020/user/hue/oozie/worksheets/hue-oozie-oozi-W
- Run: 0
- Status: SUCCEEDED
- User: hue
- Group:
- Parent Coord:
- Create Time: Mon, 27 Jul 2015 11:06:58 GMT
- Start Time: Mon, 27 Jul 2015 11:06:58 GMT
- Last Modified: Mon, 27 Jul 2015 11:06:59 GMT
- End Time: Mon, 27 Jul 2015 11:06:59 GMT

Actions:

Action Id	Name	Type	Status	Transition	StartTime	EndTime
1 0000007-150727083427440-oozie-oozi-W@:start	:start	:START:	OK	fs-2178	Mon, 27 Jul 2015 11:06:58 GMT	Mon, 27 Jul 2015 11:06:58 GMT
2 0000007-150727083427440-oozie-oozi-W@End	End	:END:	OK		Mon, 27 Jul 2015 11:06:59 GMT	Mon, 27 Jul 2015 11:06:59 GMT
3 0000007-150727083427440-oozie-oozi-W@fs-2178	fs-2178	fs	OK	End	Mon, 27 Jul 2015 11:06:59 GMT	Mon, 27 Jul 2015 11:06:59 GMT



Oozie <shell> -> bash spark-submit.sh

```
$ curl -X POST /oozie/job -d @config.properties
```

A diagram illustrating the submission process. On the left, a terminal command is shown: '\$ curl -X POST /oozie/job -d @config.properties'. A blue arrow points down to a 'config.properties' file, which contains the line 'oozie.wfapplication.path=hdfs://app/workflow.xml'. Another blue arrow points up from this file towards a 'workflow.xml' document on the right.

config.properties

```
oozie.wfapplication.path=hdfs://app/workflow.xml
```

workflow.xml

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:1.0">
  ...
  <action name="[NODE-NAME]">
    <shell xmlns="uri:oozie:shell-action:1.0">
      <resource-manager>[RESOURCE-MANAGER]</resource-manager>
      <name-node>[NAME-NODE]</name-node>
      <exec>spark-submit.sh</exec>
      <argument>[ARG-VALUE]</argument>
      <env-var>[VAR1=VALUE1]</env-var>
      ...
      <env-var>[VARN=VALUEN]</env-var>
      <file>[FILE-PATH]</file>
      ...
    </shell>
    <ok to="[NODE-NAME]"/>
    <error to="[NODE-NAME]"/>
  </action>
  ...
</workflow-app>
```



Oozie Workflow.xml - <spark>

Syntax:

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:1.0">
  ...
  <action name="[NODE-NAME]">
    <spark xmlns="uri:oozie:spark-action:1.0">
      <resource-manager>[RESOURCE-MANAGER]</resource-manager>
      <name-node>[NAME-NODE]</name-node>
      <prepare>
        <delete path="[PATH]"/>
        ...
        <mkdir path="[PATH]"/>
        ...
      </prepare>
      <job-xml>[SPARK SETTINGS FILE]</job-xml>
      <configuration>
        <property>
          <name>[PROPERTY-NAME]</name>
          <value>[PROPERTY-VALUE]</value>
        </property>
        ...
      </configuration>
      <master>[SPARK MASTER URL]</master>
      <mode>[SPARK MODE]</mode>
      <name>[SPARK JOB NAME]</name>
      <class>[SPARK MAIN CLASS]</class>
      <jar>[SPARK DEPENDENCIES JAR / PYTHON FILE]</jar>
      <spark-opts>[SPARK-OPTIONS]</spark-opts>
      <arg>[ARG-VALUE]</arg>
      ...
      <arg>[ARG-VALUE]</arg>
      ...
    </spark>
    <ok to="[NODE-NAME]"/>
    <error to="[NODE-NAME]"/>
  </action>
  ...
</workflow-app>
```

Oozie Spark Action Extension

- Spark Action
 - Spark Action Logging
 - Spark on YARN
 - PySpark with Spark Action
 - Using Symlink in <jar>
- Appendix, Spark XML-Schema
 - AE.A Appendix A, Spark XML-Schema
 - Spark Action Schema Version 1.0
 - Spark Action Schema Version 0.2
 - Spark Action Schema Version 0.1

Spark Action

The `spark` action runs a Spark job.

The workflow job will wait until the Spark job completes before continuing to the next action.

*Submit Jobs from Anywhere*

Livy enables programmatic, fault-tolerant, multi-tenant submission of Spark jobs from web/mobile apps (no Spark client needed). So, multiple users can interact with your Spark cluster concurrently and reliably.

Use Interactive Scala or Python

Livy speaks either Scala or Python, so clients can communicate with your Spark cluster via either language remotely. Also, batch job submissions can be done in Scala, Java, or Python.

No Code Changes Needed

Don't worry, no changes to existing programs are needed to use Livy. Just build Livy with Maven, deploy the configuration file to your Spark cluster, and you're off! Check out [Get Started](#) to get going.

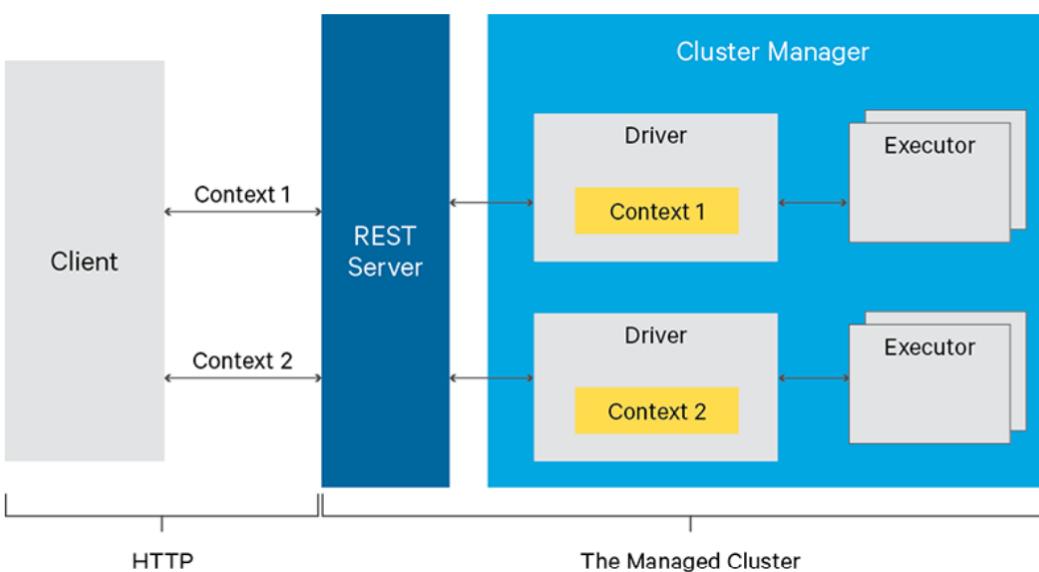
<https://livy.apache.org/>

What is Apache Livy?

Apache Livy is a service that enables easy interaction with a Spark cluster over a REST interface. It enables easy submission of Spark jobs or snippets of Spark code, synchronous or asynchronous result retrieval, as well as Spark Context management, all via a simple REST interface or an RPC client library. Apache Livy also simplifies the interaction between Spark and application servers, thus enabling the use of Spark for interactive web/mobile applications. Additional features include:

- Have long running Spark Contexts that can be used for multiple Spark jobs, by multiple clients
- Share cached RDDs or Dataframes across multiple jobs and clients
- Multiple Spark Contexts can be managed simultaneously, and the Spark Contexts run on the cluster (YARN/Mesos) instead of the Livy Server, for good fault tolerance and concurrency
- Jobs can be submitted as precompiled jars, snippets of code or via java/scala client API
- Ensure security via secure authenticated communication

To learn more, [watch this tech session video](#) from Spark Summit West 2016.





Livy ... submit via http Statefull batch / session

POST /batches

GET /batches

GET /batches/{batchId}

GET /batches/{batchId}/state

GET /batches/{batchId}/log

DELETE /batches/{batchId}

POST /sessions

GET /sessions

GET /sessions/{sessionId}

GET /sessions/{sessionId}/state

GET /sessions/{sessionId}/log

POST /sessions/{sessionId}/completion

DELETE /sessions/{sessionId}

POST /sessions/{sessionId}/statements

POST /sessions/{sessionId}/statements/{statementId}/cancel

GET /sessions/{sessionId}/statements

GET /sessions/{sessionId}/statements/{statementId}



Livy Batch Sample

```
spark-submit \  
  --master yarn --deploy-mode cluster \  
  --executor-memory 20G \  
  --class org.apache.spark.examples.SparkPi \  
  /path/to/examples.jar \  
  1000
```



```
$ curl -x POST http://<livyhost>:8998/batches -d @req.json
```

req.json:

```
{  
  "executorMemory": "20g",  
  "className": "org.apache.spark.examples.SparkPi",  
  "file": "/path/to/examples.jar",  
  "args": [1000]  
}
```



Livy (Interactive) Session - Statement

```
$ response = $( curl -x POST http://<livyhost>:8998/sessions -d session1.json )
```

session1.json:

```
{  
  "kind": "spark",  
  "executorMemory": "20g",  
  "files": [ "/path/to/examples.jar" ]  
}
```

response:

```
{  
  "id" : 2 ,  
  "appId" : "application_123_456",  
  "owner" : "user", "kind": "spark",  
  "log": [ « log line1 », « line2 » ],  
  "state": "started",  
  "appInfo": { ... }  
}
```

id=\$(.. jq '.id')

\${id}

```
$ statResp = $( curl -x POST http://<livyhost>:8998/sessions/2/statements  
  -d statement-req.json )
```

statement-req.json

```
{  
  "mode": "raw",  
  "raw": "{  
    \"code\": \" val ds = spark.sql(\" SELECT * from table1 \") \"  
  }"  
}
```



Apache Zeppelin

Zeppelin

Zeppelin Notebook - Search your Notebooks anonymous default

IoT Data Analysis

Normal	N	miles	90	4,300	-89.52	40.7	0	0	0	0	0
Normal	N	miles	90	4,300	-91.05	41.72	0	0	0	1	
Normal	N	miles	90	4,300	-91.47	41.74	0	0	0	0	
Lane Departure	N	miles	90	4,300	-91.59	41.7	1	0	0	0	
Unsafe following distance	N	miles	90	4,300	-88.77	40.76	0	1	1	1	

Took a few seconds. Last updated by anonymous at September 16 2015, 9:47:27 AM.

Xsql FINISHED settings

```
select eventType, count(*) occurrences from enrichedEvents group by eventType
```

Grouped Stacked occurrences

Took a few seconds. Last updated by anonymous at August 02 2016, 8:00:49 PM. (outdated)

Xsql FINISHED settings

```
select isCertified, count(*) numViolations from enrichedEvents where eventType in ("Unsafe tail distance", "Overspeed", "Lane Departure", "Unsafe following distance") group by isCertified
```

Grouped Stacked settings

Took a few seconds. Last updated by anonymous at August 02 2016, 8:00:33 PM. (outdated)

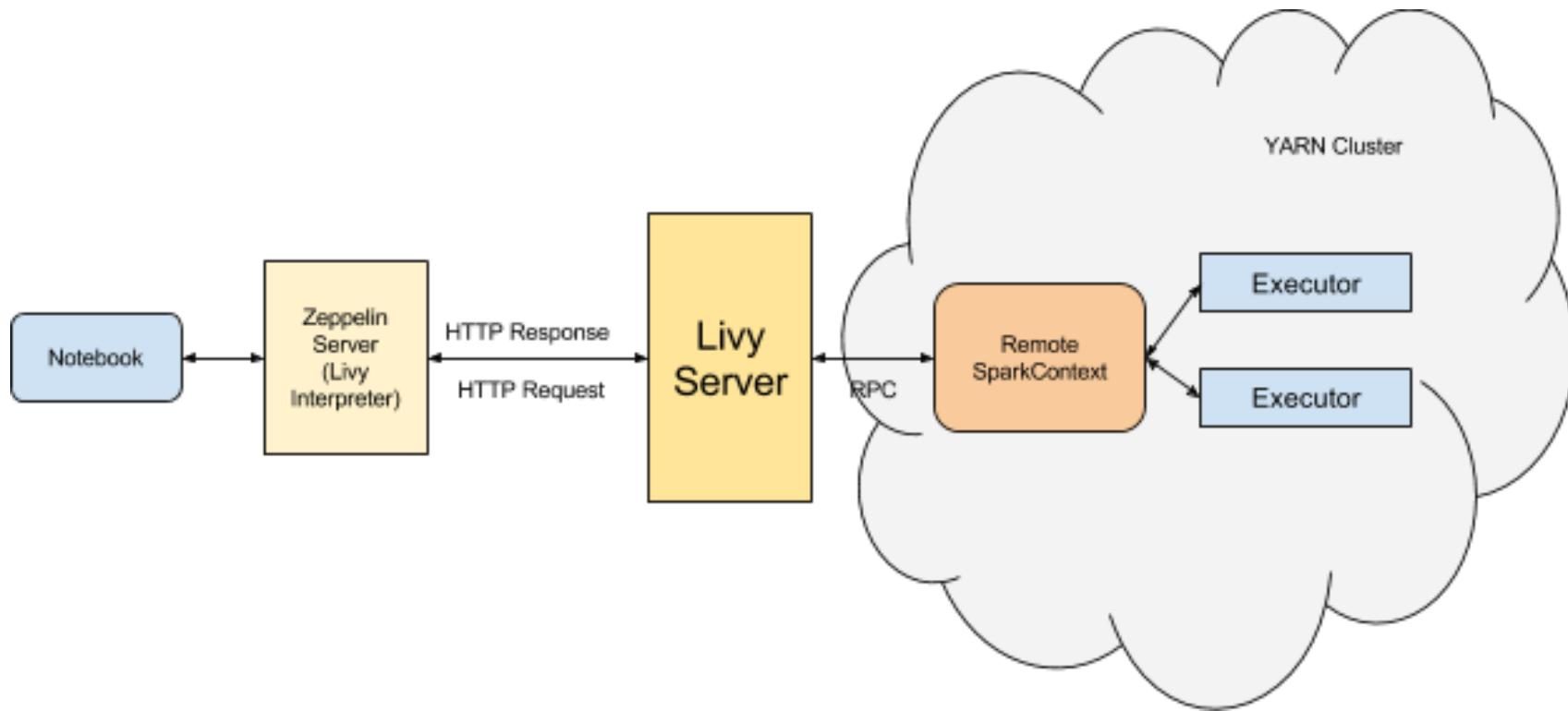
Xsql FINISHED settings

```
select eventType, isCertified, hoursDriven from enrichedEvents group by eventType, isCertified, hoursDriven
```

Stacked Stream Expanded eventType

Took a few seconds. Last updated by anonymous at August 02 2016, 8:00:41 PM. (outdated)

Zeppelin -> Livy Interactive Session





Configuration

We added some common configurations for spark, and you can set any configuration you want. You can find all Spark configurations in [here](#). And instead of starting property with `spark.` it should be replaced with `livy.spark..` Example: `spark.driver.memory` to `livy.spark.driver.memory`

Property	Default	Description
<code>zeppelin.livy.url</code>	<code>http://localhost:8998</code>	URL where livy server is running
<code>zeppelin.livy.spark.sql.maxResult</code>	1000	Max number of Spark SQL result to display.
<code>zeppelin.livy.spark.sql.field.truncate</code>	true	Whether to truncate field values longer than 20 characters or not
<code>zeppelin.livy.session.create_timeout</code>	120	Timeout in seconds for session creation
<code>zeppelin.livy.displayAppInfo</code>	true	Whether to display app info
<code>zeppelin.livy.pull_status.interval.millis</code>	1000	The interval for checking paragraph execution status
<code>livy.spark.driver.cores</code>		Driver cores. ex) 1, 2.
<code>livy.spark.driver.memory</code>		Driver memory. ex) 512m, 32g.
<code>livy.spark.executor.instances</code>		Executor instances. ex) 1, 4.
<code>livy.spark.executor.cores</code>		Num cores per executor. ex) 1, 4.
<code>livy.spark.executor.memory</code>		Executor memory per worker instance. ex) 512m, 32g.
<code>livy.spark.dynamicAllocation.enabled</code>		Use dynamic resource allocation. ex) True, False.
		Remove an executor which has cached data

Apache Zeppelin

Link Zeppelin -> SparkUI Application

The screenshot shows a Zeppelin notebook titled "test01". A red circle highlights the "Spark Application Id: application_1613965809194_0019" and "Spark WebUI: http://.../hdinsightcss.com:8080/proxy/application_1613965809194_0019/" text. Another red circle highlights the "default" interpreter dropdown in the top right. A red arrow points from the "Spark Application Id" text to the "Job" tab of the Spark UI.

```
| val ds = spark.read.text("abfss://...@kcosasbt1.dfs.core.windows.net/example/data/yellowthings.txt")  
ds: org.apache.spark.sql.DataFrame = [value: string]
```

Spark Application Id: application_1613965809194_0019
Spark WebUI: http://.../hdinsightcss.com:8080/proxy/application_1613965809194_0019/
Took 1 min 24 sec. Last updated by anonymous at February 21 2021, 10:15:05 PM.

Zeppelin Notebook Job Job Storage Environment Executors SQL Spark shell application UI

interpreter settings

The screenshot shows the "Jobs" tab of the Spark UI. It displays the "Active Jobs" section with one job listed and the "Completed Jobs" section with one job listed. A red arrow points from the "Spark Application Id" text in the Zeppelin screenshot to this "Jobs" tab.

Spark Jobs

Active Jobs (1)

Job ID	Description	Submitted	Duration	Stages: Successeds/Total	Tasks (for all stages): Successeds/Total
0	spark://<node>-24	2016/09/28 14:09:20	6 s	0/1	0/1

Completed Jobs (1)

Job ID	Description	Submitted	Duration	Stages: Successeds/Total	Tasks (for all stages): Successeds/Total
0	spark://<node>-24	2016/09/28 14:09:20	0 s	1/1	0/1

Failed Jobs (1)

Job ID	Description	Submitted	Duration	Stages: Successeds/Total	Tasks (for all stages): Successeds/Total
1	spark://<node>-24	2016/09/28 14:21:14	81 ms	0/1 (1 failed)	0/1 (1 failed)

The screenshot shows the Zeppelin header menu with the "anonymous" user dropdown open. A red circle highlights the "Interpreter" option in the dropdown menu. A red arrow points from the "Interpreter" text in the Zeppelin screenshot to this "Interpreter" menu item.

Q anonymous

About Zeppelin

Interpreter

Notebook Repos

Credential

Helium

Configuration

edit restart remove



Jupyter Notebook

A screenshot of the Jupyter file browser interface. The title bar says "try.jupyter.org". Below it, there are tabs for "Files", "Running", and "Clusters". A message says "Select items to perform actions on them." On the left, there's a sidebar with icons for "communities", "datasets", "featured", and several "Welcome" notebooks for various languages (Julia, R, Haskell, Python, Spark with Python, Scala). At the bottom right, there are "Upload" and "New" buttons.

A screenshot of a Jupyter notebook interface titled "try.jupyter.org". The title bar shows "try.jupyter.org" and "Python 3". The main content area has a "jupyter" logo and the text "Welcome to the Temporary Notebook (tmpnb) service!". It includes a "WARNING" box stating "Don't rely on this server for anything you want to last - your server will be deleted after 10 minutes of inactivity." It also mentions "Your server is hosted thanks to Rackspace, on their on-demand bare metal servers, OnMetal.". Below this, there's a section titled "Run some Python code!" with instructions: "To run the code below: 1. Click on the cell to select it. 2. Press SHIFT+ENTER on your keyboard or press the play button (▶) in the toolbar above." A "full tutorial for using the notebook interface is available [here](#)". A code cell titled "In []" contains the following Python code:

```
In [ ]: %matplotlib notebook
import pandas as pd
import numpy as np
import matplotlib

from matplotlib import pyplot as plt
import seaborn as sns

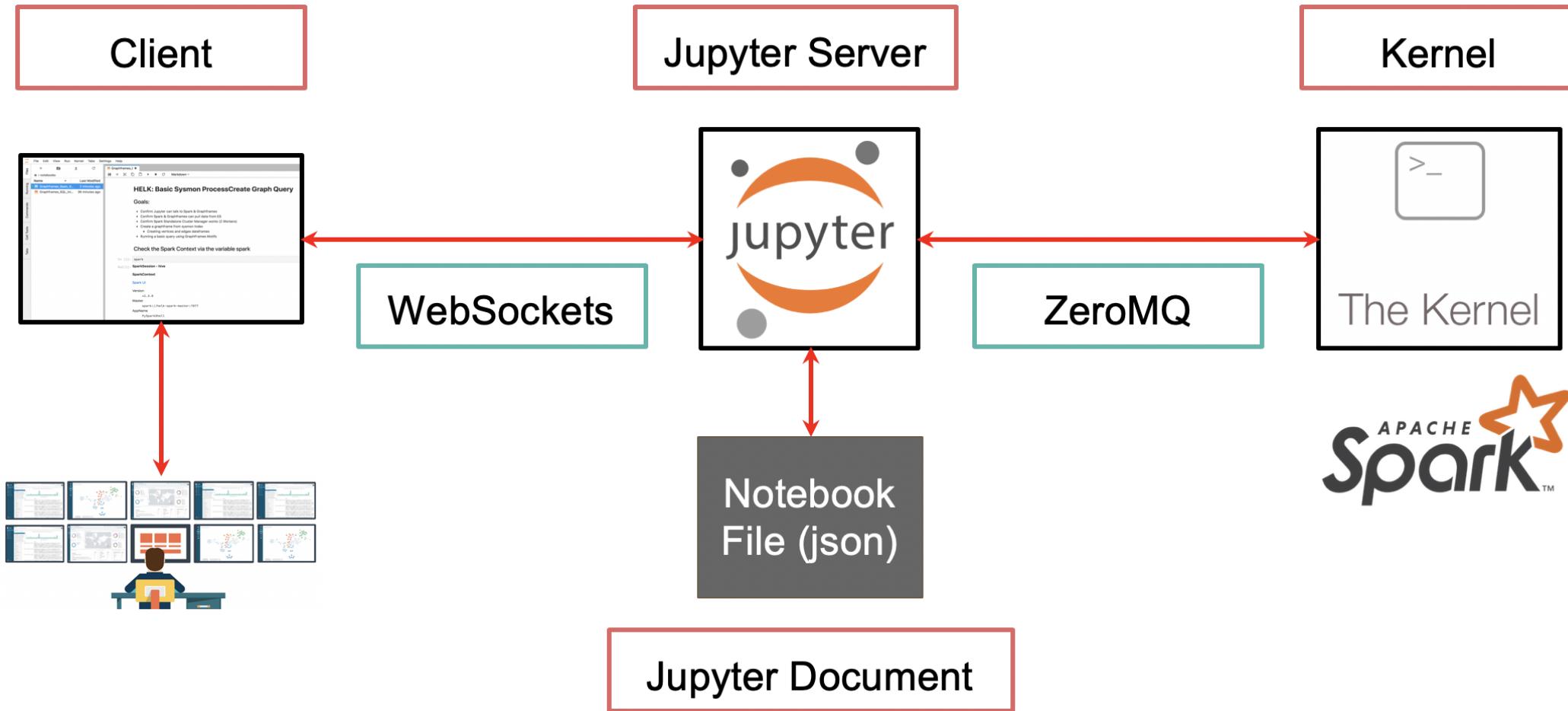
ts = pd.Series(np.random.randn(1000), index=pd.date_range('1/1/2000', periods=1000))
ts = ts.cumsum()

df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index,
                  columns=['A', 'B', 'C', 'D'])
df = df.cumsum()
df.plot(); plt.legend(loc='best')
```

At the bottom, there's a note: "Feel free to open new cells using the plus button (+), or hitting shift-enter while this cell is selected." and "Behind the scenes, the software that powers this is tmpnb, a Tornado application that spawns pre-built Docker containers and then uses the jupyter/configurable-http-proxy to put your notebook server on a unique path."



Jupyter – Kernel(s) architecture





Jupyter + python kernel + pyspark

```
In [11]: from pyspark.context import SparkContext  
from pyspark.sql.session import SparkSession  
from pyspark.sql.types import *  
from pyspark.sql.functions import *  
  
sqlContext = SQLContext(sc)  
df = sqlContext.sql("select 1 as A")  
df.show()
```

```
+---+  
| A|  
+---+  
| 1|  
+---+
```



More « Extensions » backport API to other Languages (Python, R)

class DataFrame:

... hundred methods ...

```
def method123 (self, x) -> Y  
    self._jdf.method(x, self.sparkSession)
```



```
from py4j.java_gateway import JavaObject  
from subprocess import Popen  
from pyspark.context import SparkContext  
  
..  
pid = Popen(« spark-submit » ...)  
Socket(.. )
```



API



API

public class Dataset {

... hundred methods ...

```
public Y method123 (X x) {  
    ... internals sparkContext...  
}
```



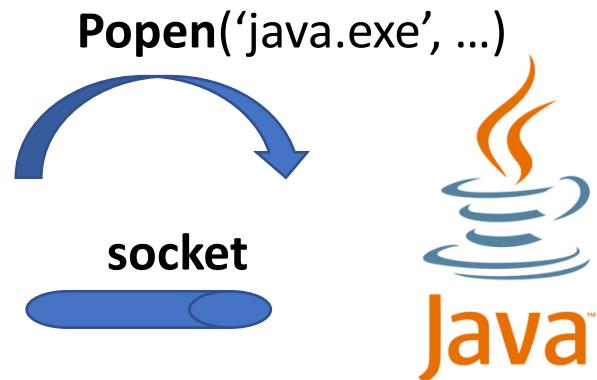
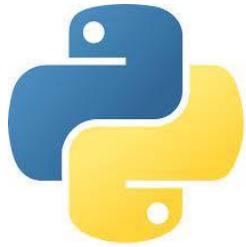
py4j



Python + pyspark (GLUE package) ... = spark JVM proces + socket

\$ pyspark

python.exe



Write python Object
to pickle format → read java Object
from pickle format

SparkContext (spark-driver)

```
23 import socket
24 import platform
25 import tempfile
26 import time
27 from subprocess import Popen, PIPE
28
29 from py4j.java_gateway import java_import, JavaGateway, GatewayParameters
30 from py4j.clientserver import ClientServer, JavaParameters, PythonParameters
31 from pyspark.find_spark_home import _find_spark_home
32 from pyspark.serializers import read_int, write_with_length, UTF8Deserializer
```



<https://www.py4j.org/>



Python UDF function

PySpark supports various UDFs and APIs to allow users to execute Python native functions. See also the latest [Pandas UDFs](#) and [Pandas Function APIs](#). For instance, the example below allows users to directly use the APIs in a [pandas Series](#) within Python native function.

```
[21]: import pandas as pd
from pyspark.sql.functions import pandas_udf

@pandas_udf('long')
def pandas_plus_one(series: pd.Series) -> pd.Series:
    # Simply plus one by using pandas Series.
    return series + 1

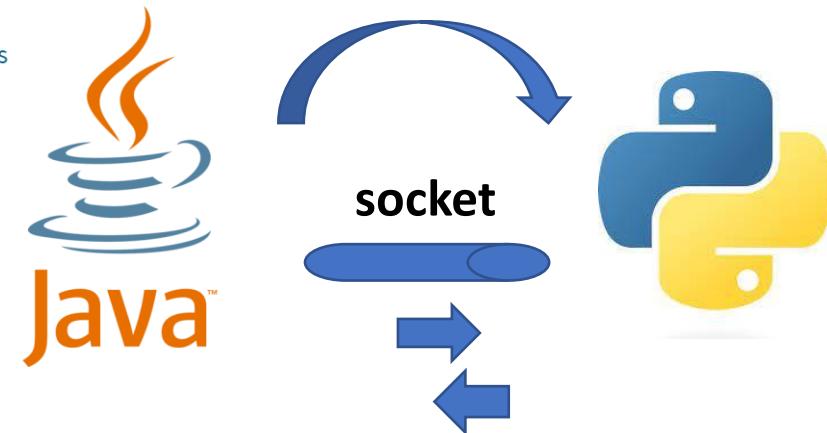
df.select(pandas_plus_one(df.a)).show()
```

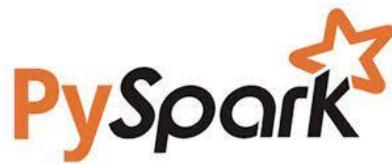
a	pandas_plus_one(a)
1	2
2	3
3	4

spark-executor

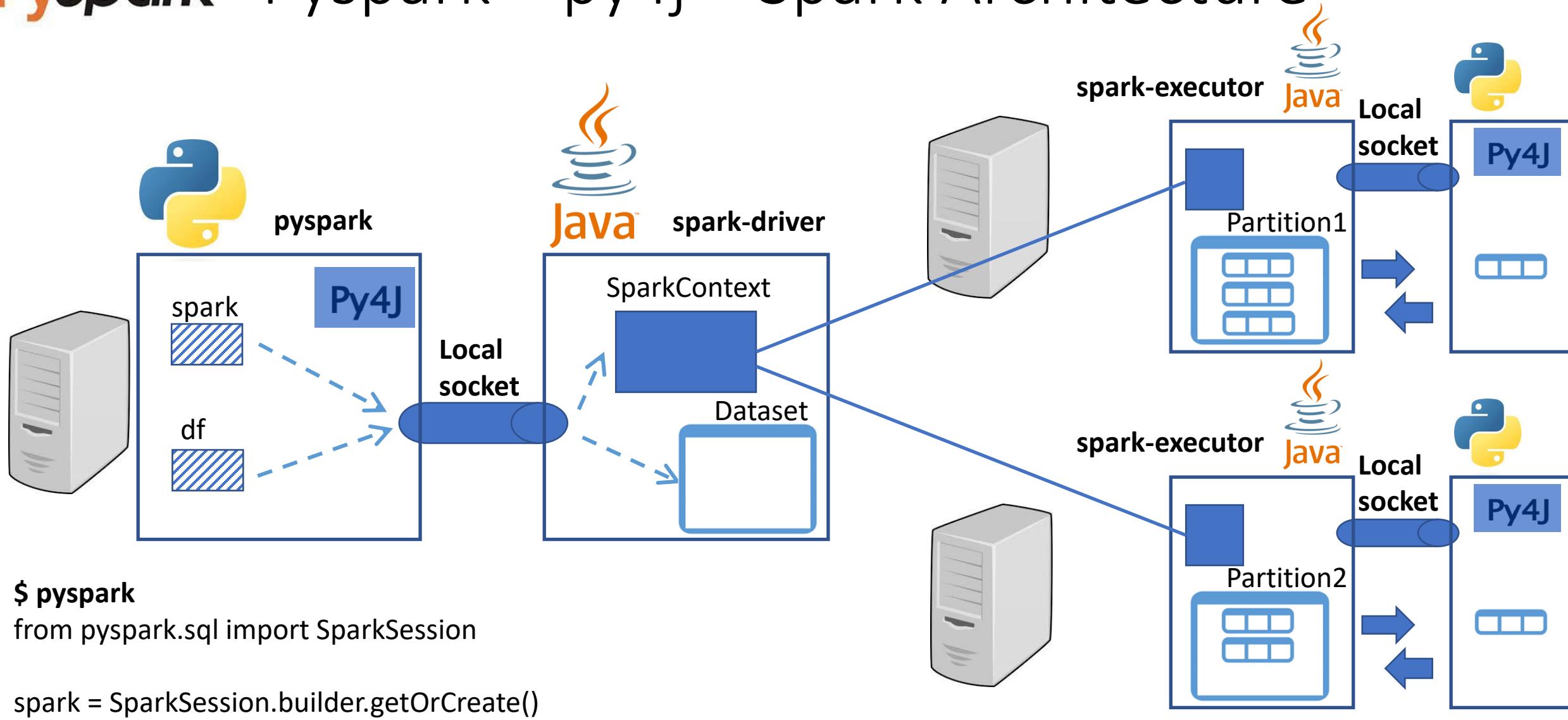
Python.exe runner

`System.exec('python.exe', ...)`



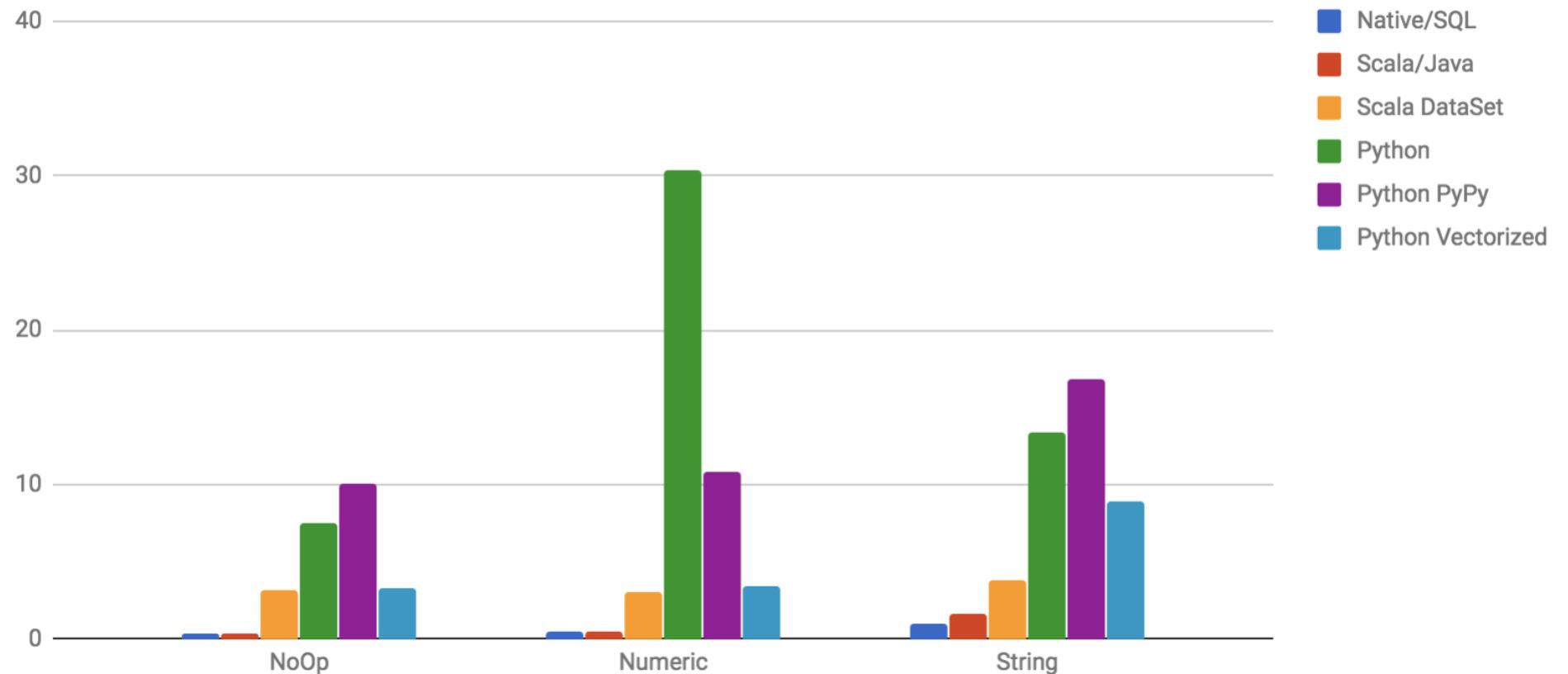


Pyspark – py4j – Spark Architecture



Python + Pickle = Performances Pourries

Simple UDF Performance Results (R Excluded)





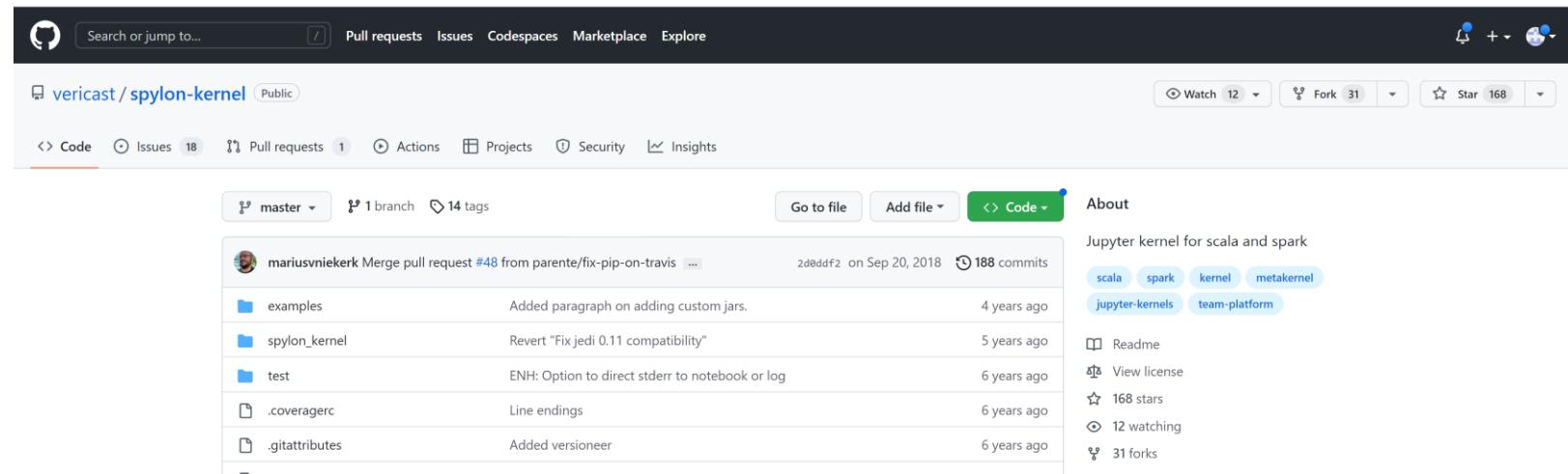
Jupyter
Scala

Jupyter + spylon (spark scala kernel)

```
pip install spylon-kernel
# or
conda install -c conda-forge spylon-kernel
```

The screenshot shows the Jupyter Notebook interface. At the top right, there are 'Quit' and 'Logout' buttons. Below them is a toolbar with 'Upload', 'New ▾', and a refresh icon. A red arrow points from the text in the previous block to the 'New ▾' button, which has a dropdown menu open. The dropdown menu is titled 'Notebook:' and contains options: 'Python 3' (which is currently selected), 'spylon-kernel' (highlighted with a black box), 'Create a new notebook with spylon-kernel', 'Text File', 'Folder', and 'Terminal'. To the right of the dropdown, there is some faint text: 'Name ▾', 'Recent Notebooks', and '5 months ago', '6 months ago', '3 hours ago', and '288 R'.

spylon : %%scala



Using it as an IPython Magic

You can also use spylon-kernel as a magic in an IPython notebook. Do this when you want to mix a little bit of Scala into your primarily Python notebook.

```
from spylon_kernel import register_ipython_magics  
register_ipython_magics()
```

```
%%scala  
val x = 8  
x
```

Using it as a Library

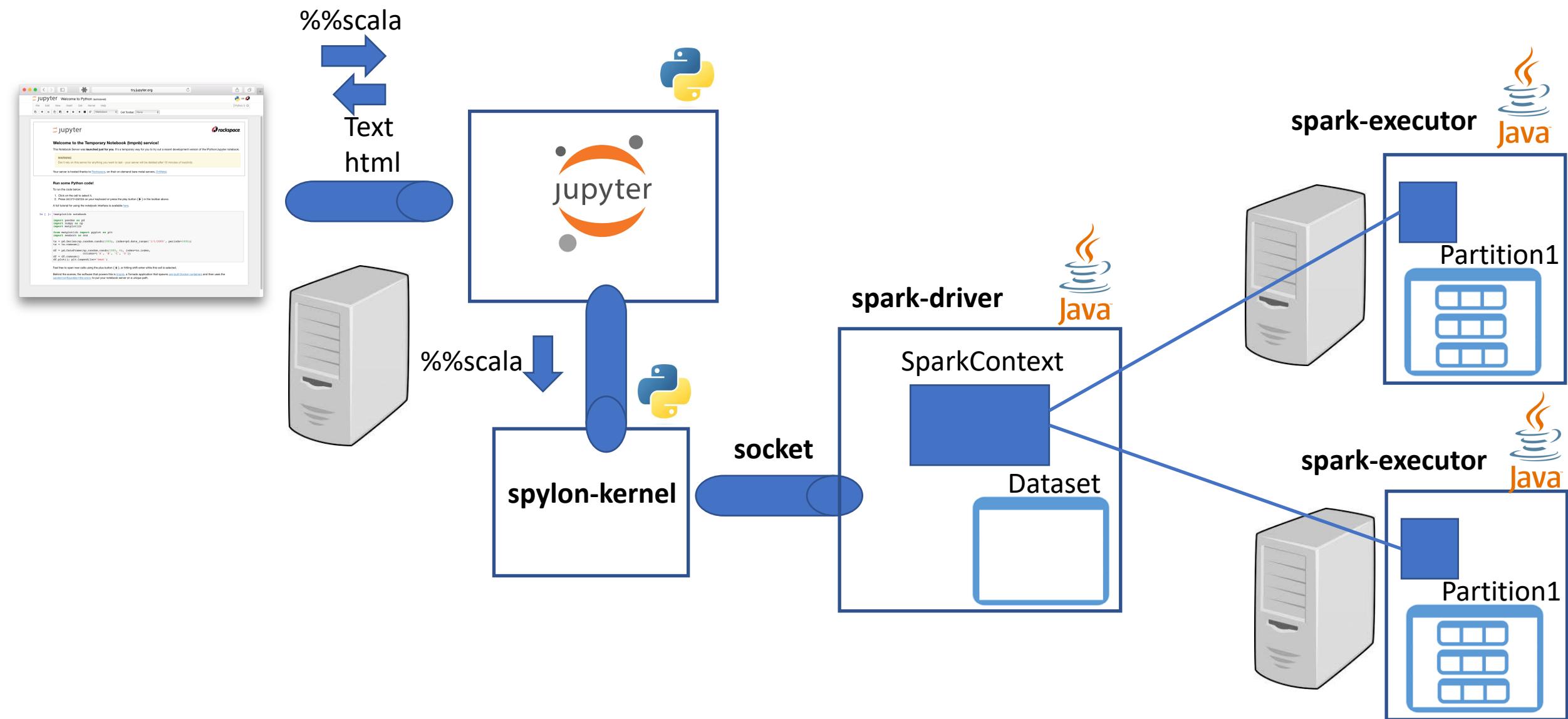
Finally, you can use spylon-kernel as a Python library. Do this in a Python script or shell.

```
from spylon_kernel import get_scala_interpreter  
  
interp = get_scala_interpreter()  
  
# Evaluate the result of a scala code block.  
interp.interpret("""  
    val x = 8  
    x  
""")  
  
interp.last_result()
```

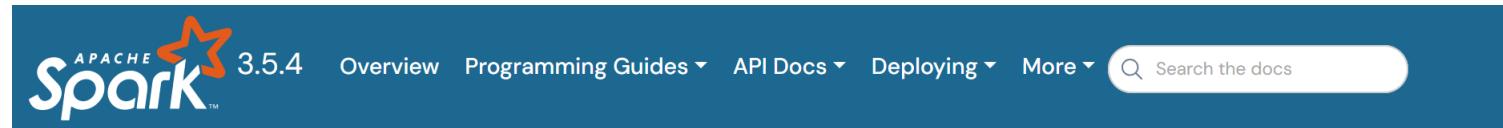


Scala

Jupyter + spylon architecture



Spark Connect ... new since version 3.4



Spark Connect Overview

Building client-side Spark applications

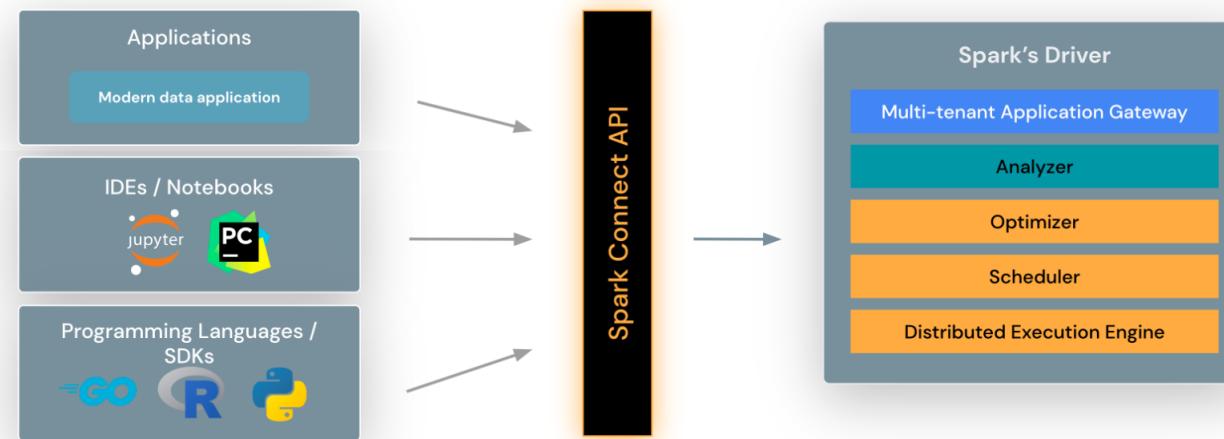
In Apache Spark 3.4, Spark Connect introduced a decoupled client-server architecture that allows remote connectivity to Spark clusters using the DataFrame API and unresolved logical plans as the protocol. The separation between client and server allows Spark and its open ecosystem to be leveraged from everywhere. It can be embedded in modern data applications, in IDEs, Notebooks and programming languages.

To get started, see [Quickstart: Spark Connect](#).

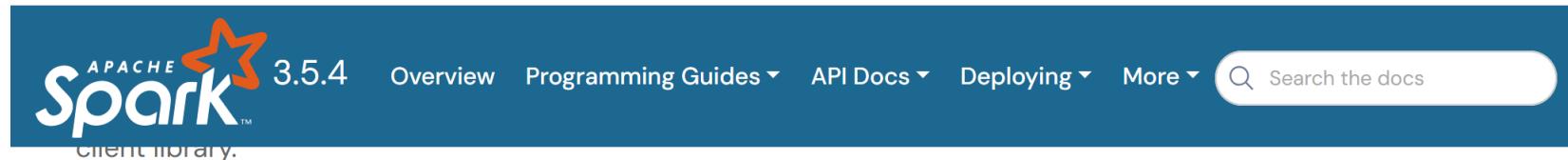
Spark Connect



Thin client, with the full power of Apache Spark



./sbin/start-connect-server.sh



The image shows the Apache Spark 3.5.4 documentation header. It features the Apache Spark logo with a red star icon and the text "APACHE Spark 3.5.4". To the right are navigation links: Overview, Programming Guides, API Docs, Deploying, More, and a search bar labeled "Search the docs".

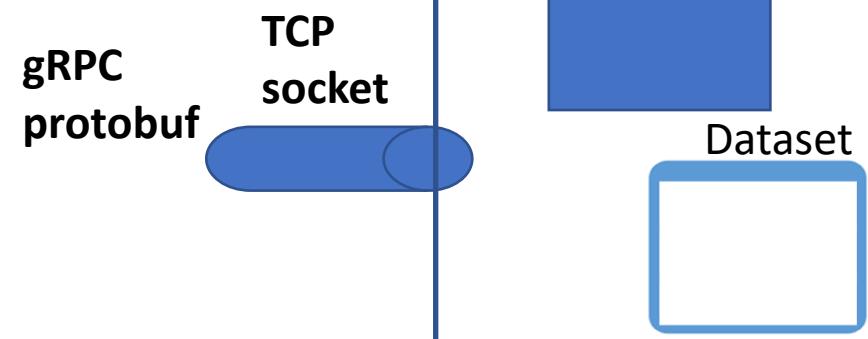
Download and start Spark server with Spark Connect

In a terminal window, go to the spark folder in the location where you extracted Spark before and run the `start-connect-server.sh` script to start Spark server with Spark Connect, like in this example:

```
./sbin/start-connect-server.sh --packages org.apache.spark:spark-connect_2.12:3.5.4
```

Note that we include a Spark Connect package (`spark-connect_2.12:3.5.4`) when starting Spark server. This is required to use Spark Connect. Make sure to use the same version of the package as the Spark version you downloaded previously. In this example, Spark 3.5.4 with Scala 2.12.

**spark-connect-server
(spark-driver + gRpc)** 



example: Spark Connect via python (no need JRE on client)

The screenshot shows a screenshot of the Apache Spark documentation website. The header includes the Apache Spark logo and the text "3.5.4 Overview Programming Guides ▾ API Docs ▾ Deploying ▾ More ▾ Search the docs". Below the header, the title "Use Spark Connect in standalone applications" is displayed, followed by tabs for "Python" and "Scala". A text block instructs users to install PySpark with pip and provides a code snippet for setup.py. Another text block shows an example of creating a Spark session with remote functionality.

Use Spark Connect in standalone applications

Python Scala

First, install PySpark with `pip install pyspark[connect]==3.5.0` or if building a packaged PySpark application/library, add it your `setup.py` file as:

```
install_requires=[  
    'pyspark[connect]==3.5.0'  
]
```

When writing your own code, include the `remote` function with a reference to your Spark server when you create a Spark session, as in this example:

```
from pyspark.sql import sparkSession  
spark = sparkSession.builder.remote("sc://localhost").getOrCreate()
```

Enough for Today !

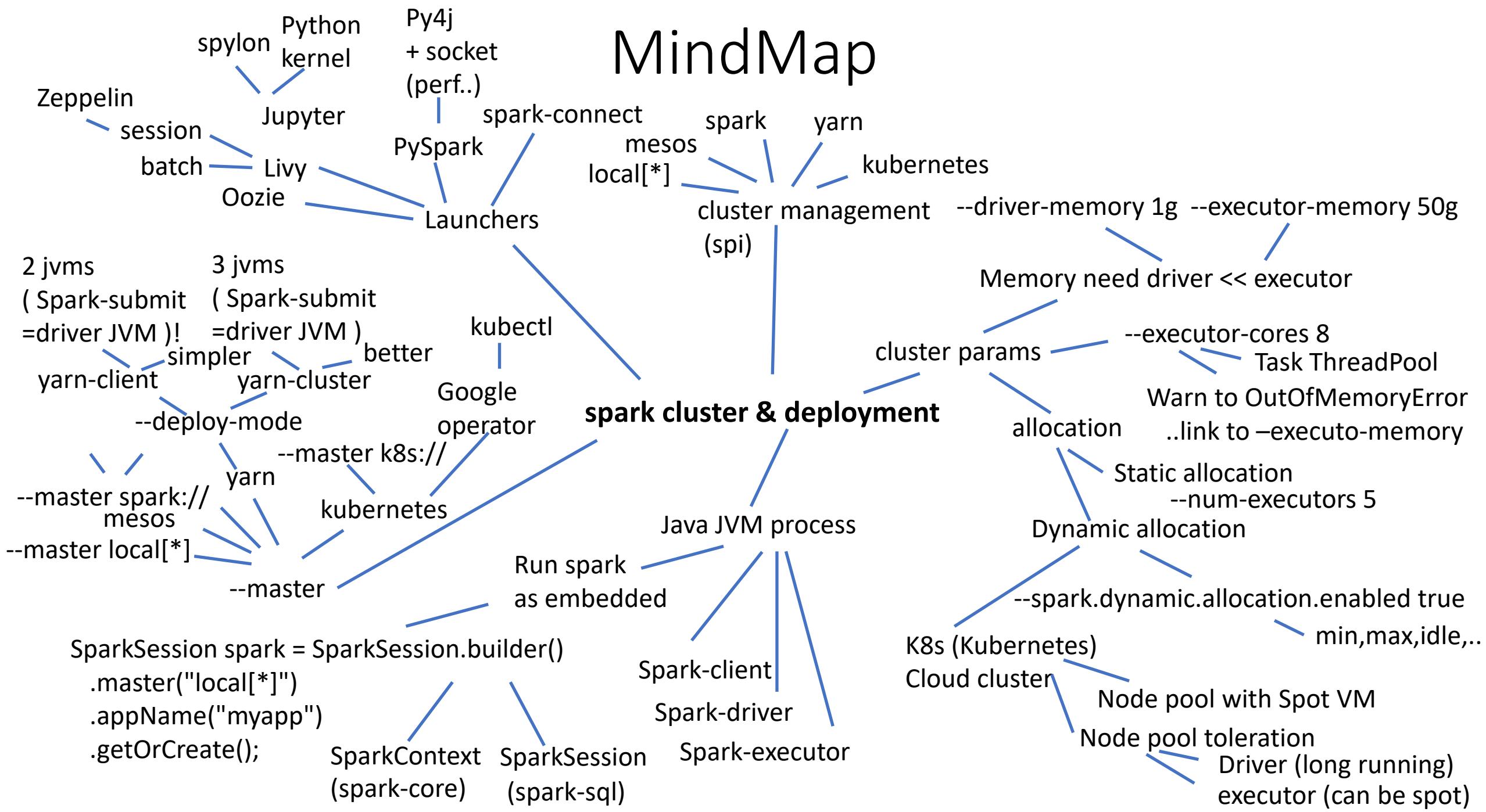
... but more to come

Questions are Welcome

Take Away

What Did you learn ?

MindMap



Questions ?

arnaud.nauwynck@gmail.com