BigData – Spark – Processing

Spark cluster mode distribution & Tuning

Arnaud Nauwynck – Nov 2022

Outline

Reminder Spark-driver / spark-executers

Cluster Tuning Params

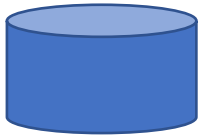
deployMode (local, spark://, yarn, k8s)

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

Spark-Core + ...

Reminder

Structured Data



Spark SQL

Spark Streaming

Spark MLlib

Spark GraphX

Modules

Amazon S3

Azure Data Lake Storage Gen2

DataSource Connectors (Hadoop API)

Spark Core

Cluster Manager

Langages Support

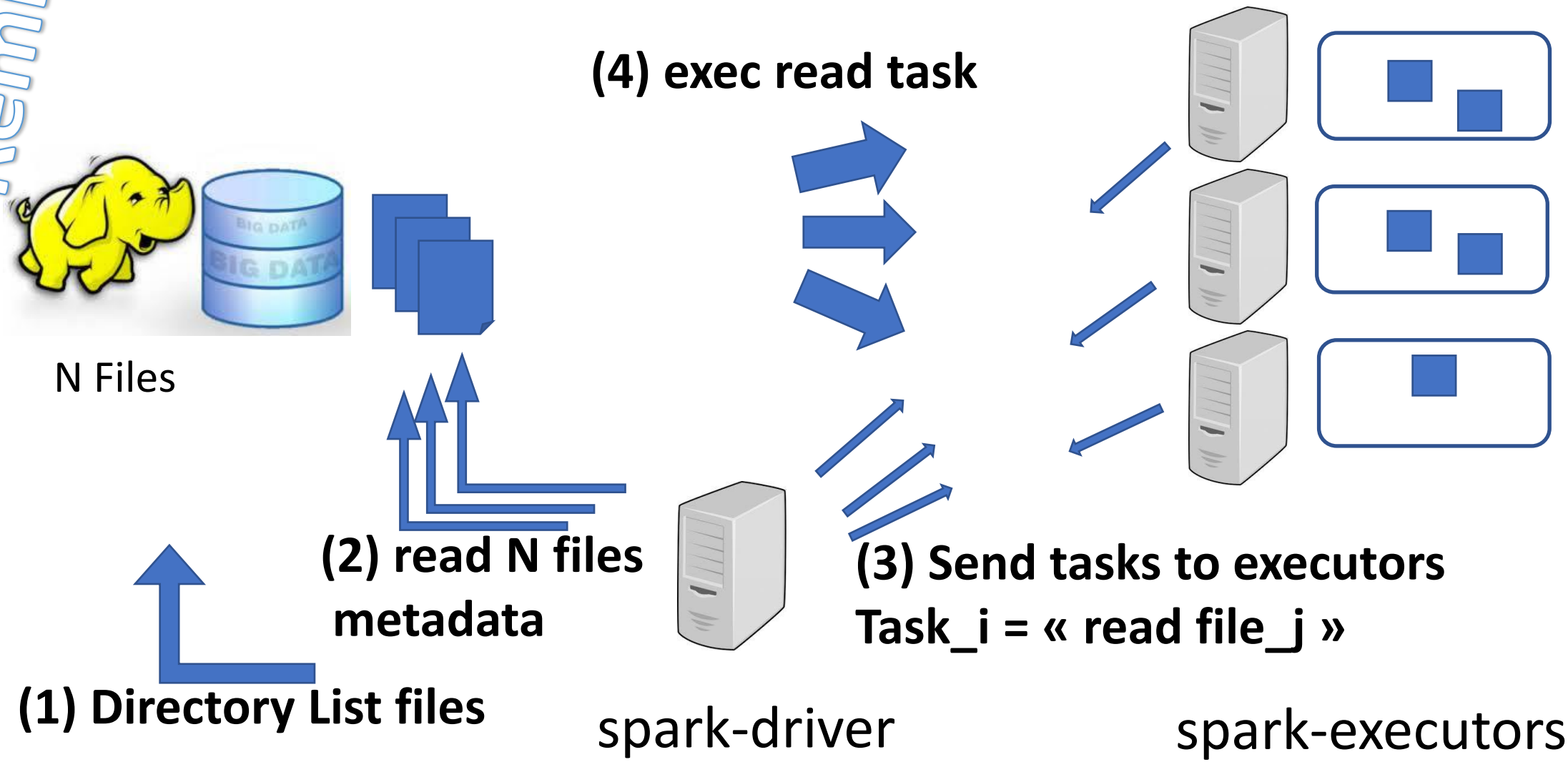
Reminder

1 Spark application

= 1 Spark-driver + N spark-executers

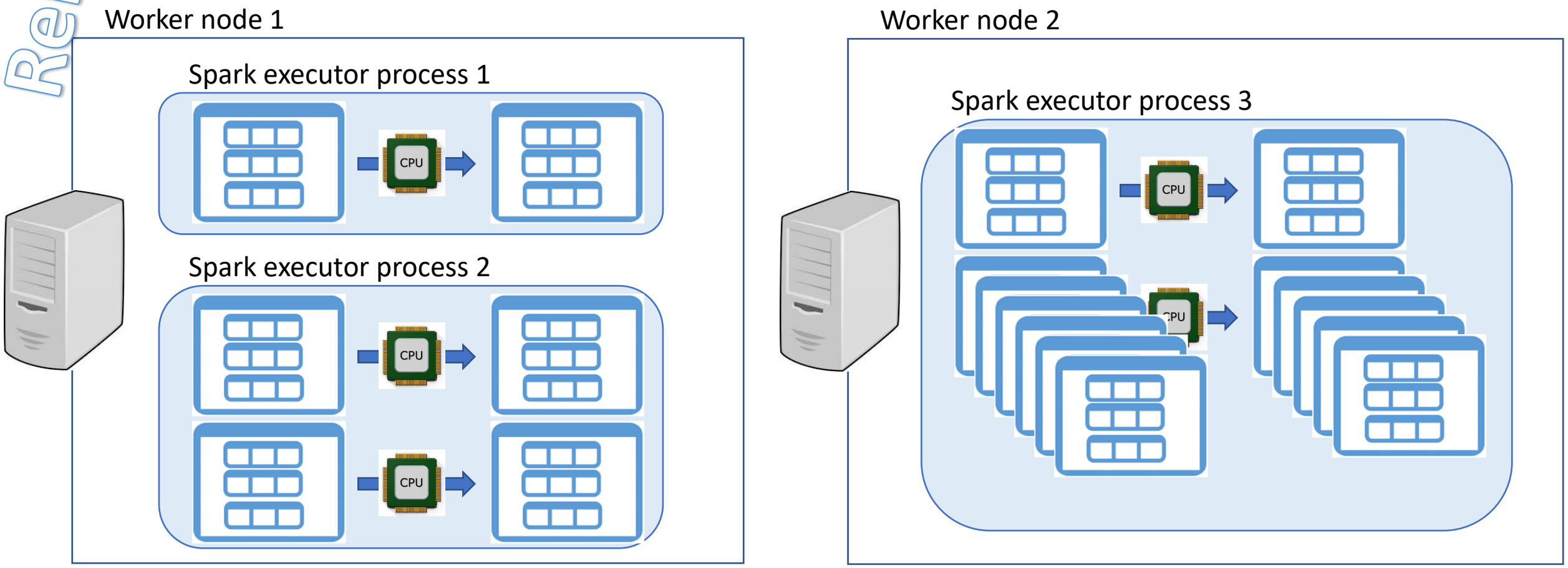
Reminder

Read N Files – assign Tasks to Executors



Reminder

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

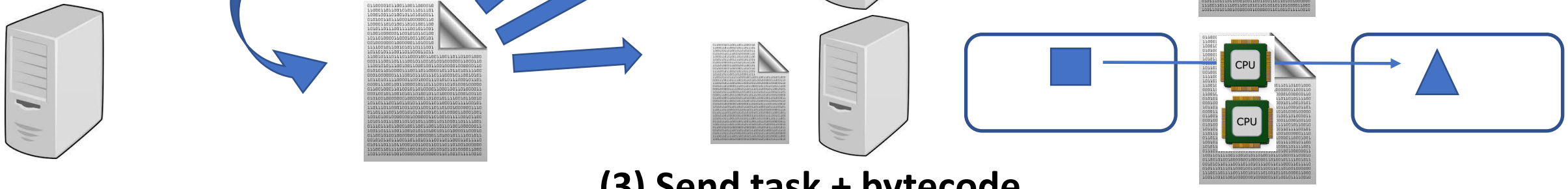


WholeStageCodeGen

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

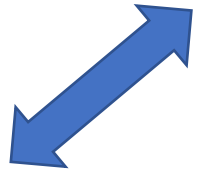
(3) Send task + bytecode to spark-executors

(4) Execute tasks

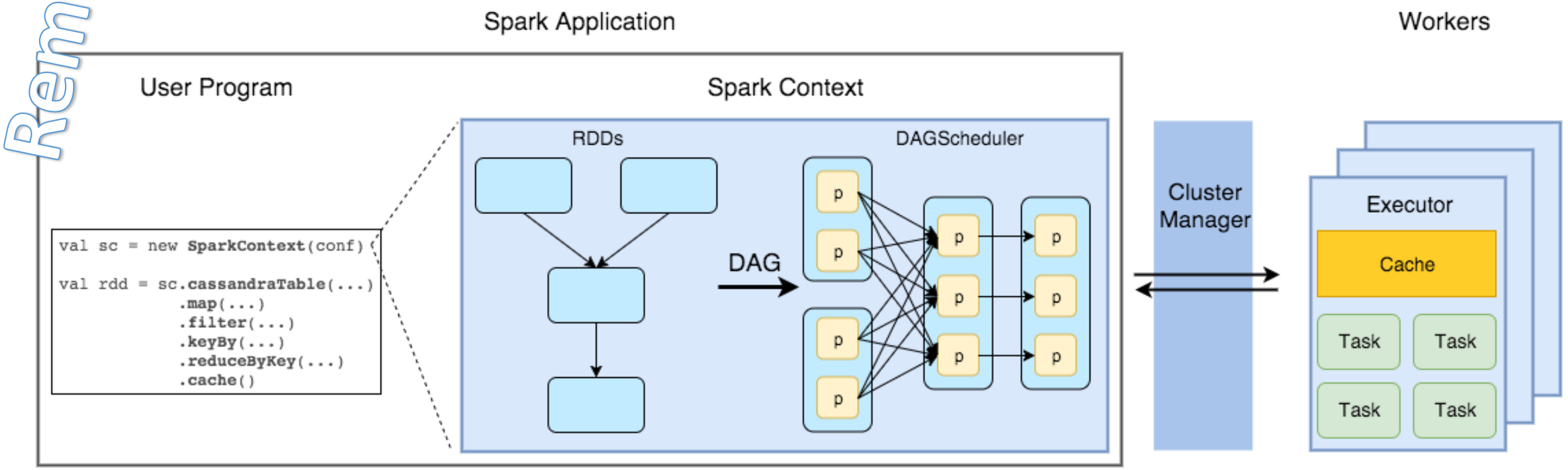
Reminder

Cluster Manager Extensions: TaskScheduler API


```
abstract class TaskScheduler {
  ..start(),stop(),

  submitTasks,cancelTasks,

  notify Host-Executor-Task changes
}
```



Reminder


Spark Application
Workers
User Program
Spark Context
RDDs
DAGScheduler
DAG
val sc = new SparkContext(conf)
val rdd = sc.cassandraTable(...)
.map(...)
.filter(...)
.keyBy(...)
.reduceByKey(...)
.cache()
p
Cluster Manager
Executor
Cache
Task
Task
Task
Task

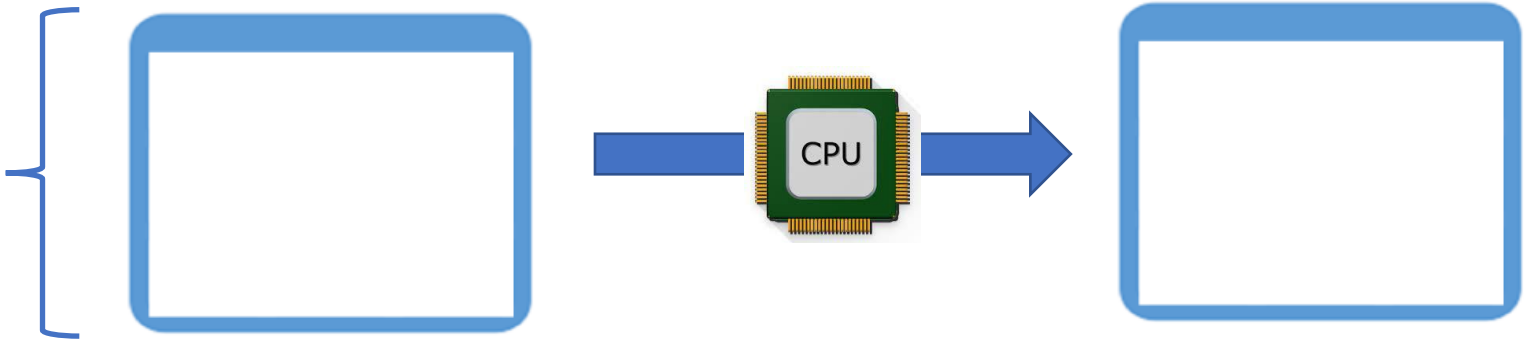
Reminder

CoarseGrainScheduler … CoarseGrain Executer

Spark-driver
Implements Fault Tolerance+Distribution

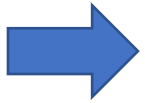
Spark-executor
… internally called « CoarseGrainExecutor » main loop

CoarseGrain
partition
= unit or recomputation



Outline

Reminder Spark-driver / spark-executers



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

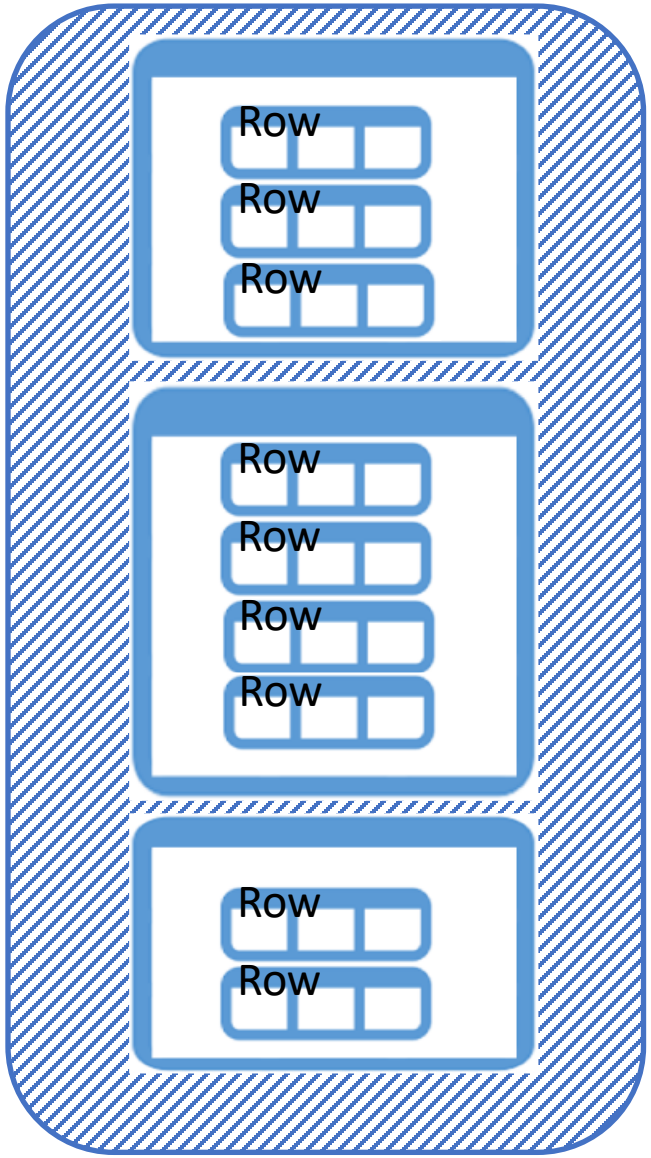
driver != executer

DataSet = Partitions metadata / data

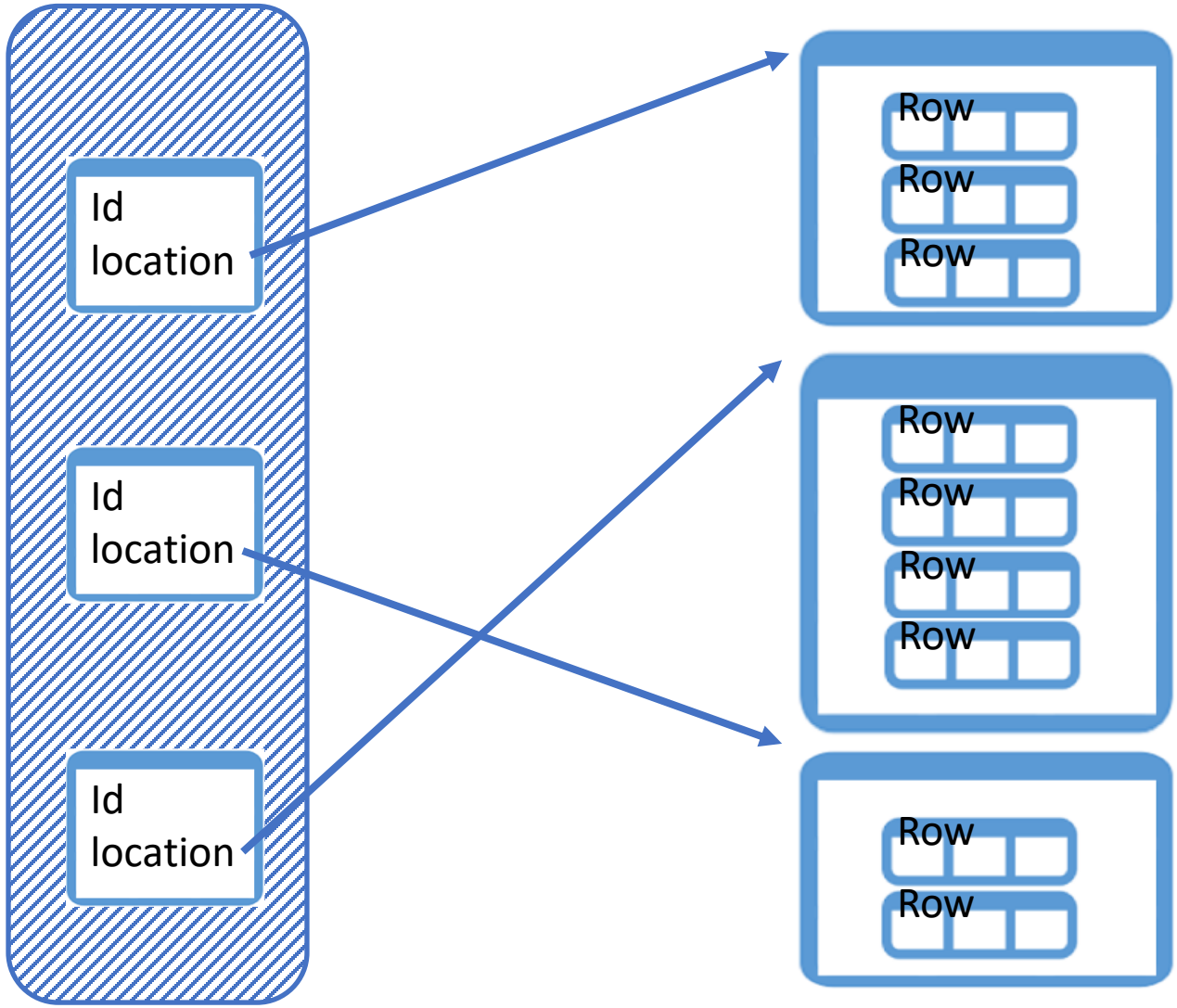
1 Dataset
= N partitions

1 Partition
= p rows
(p1 != p2 != ..pN)

Row



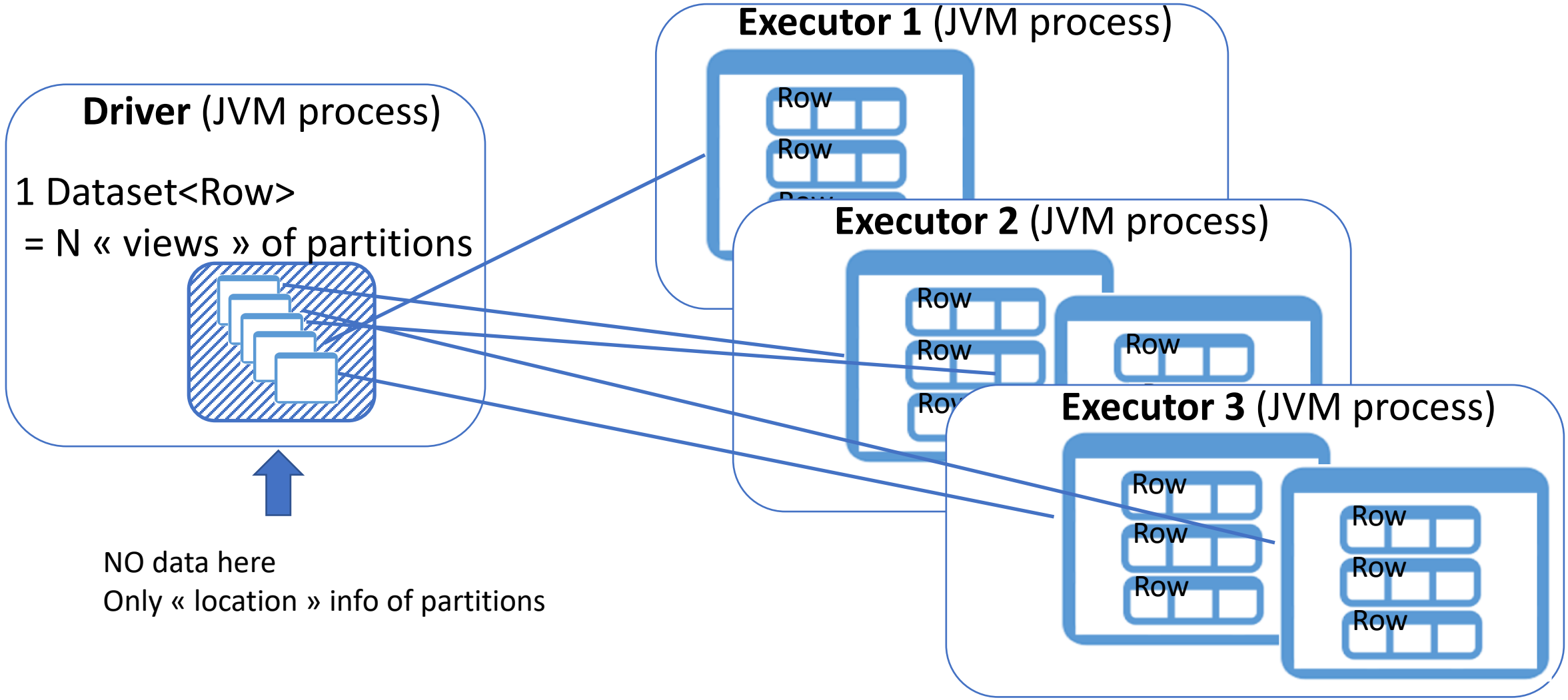
On Driver = metadata / on Executor = Datas



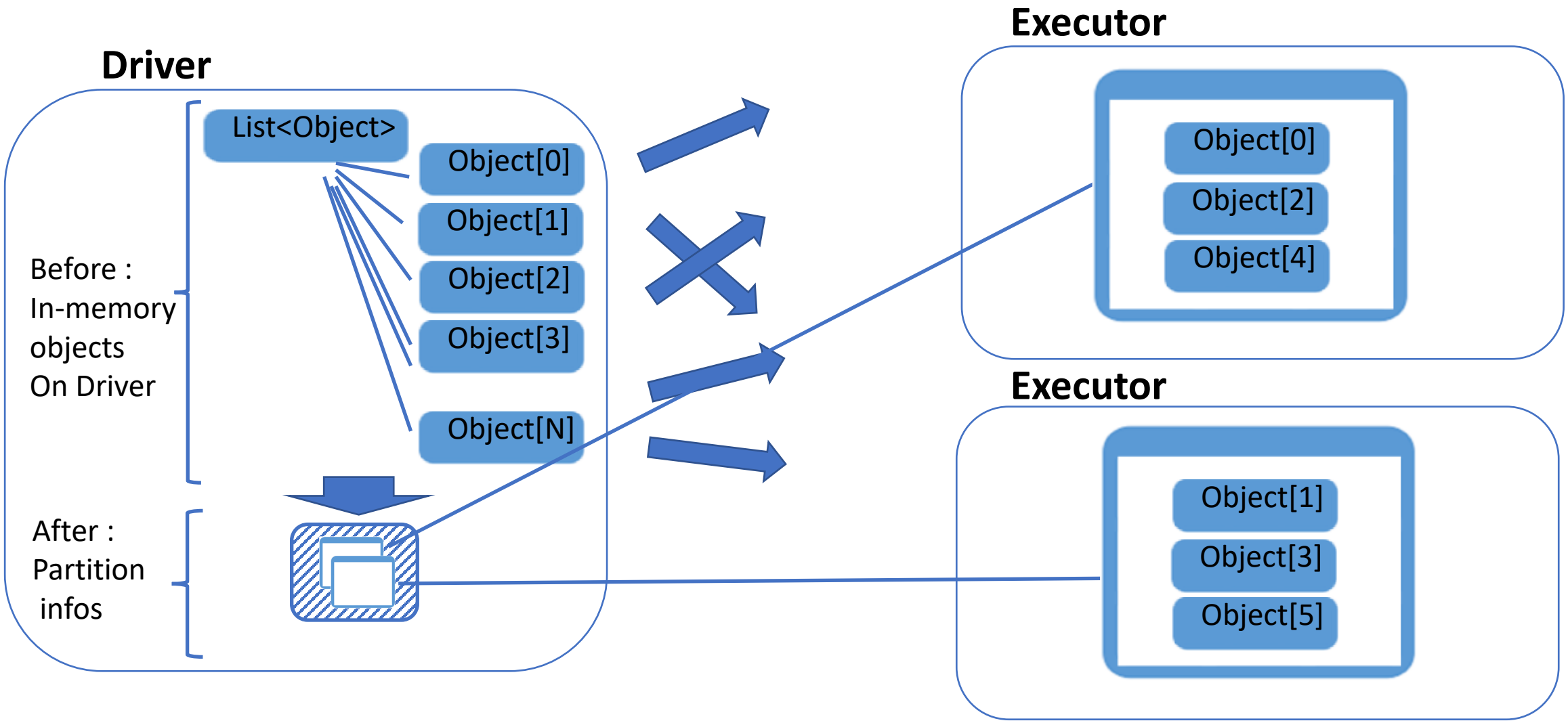
1 Dataset on spark-driver

N x Partitions (blocks) on spark-executors

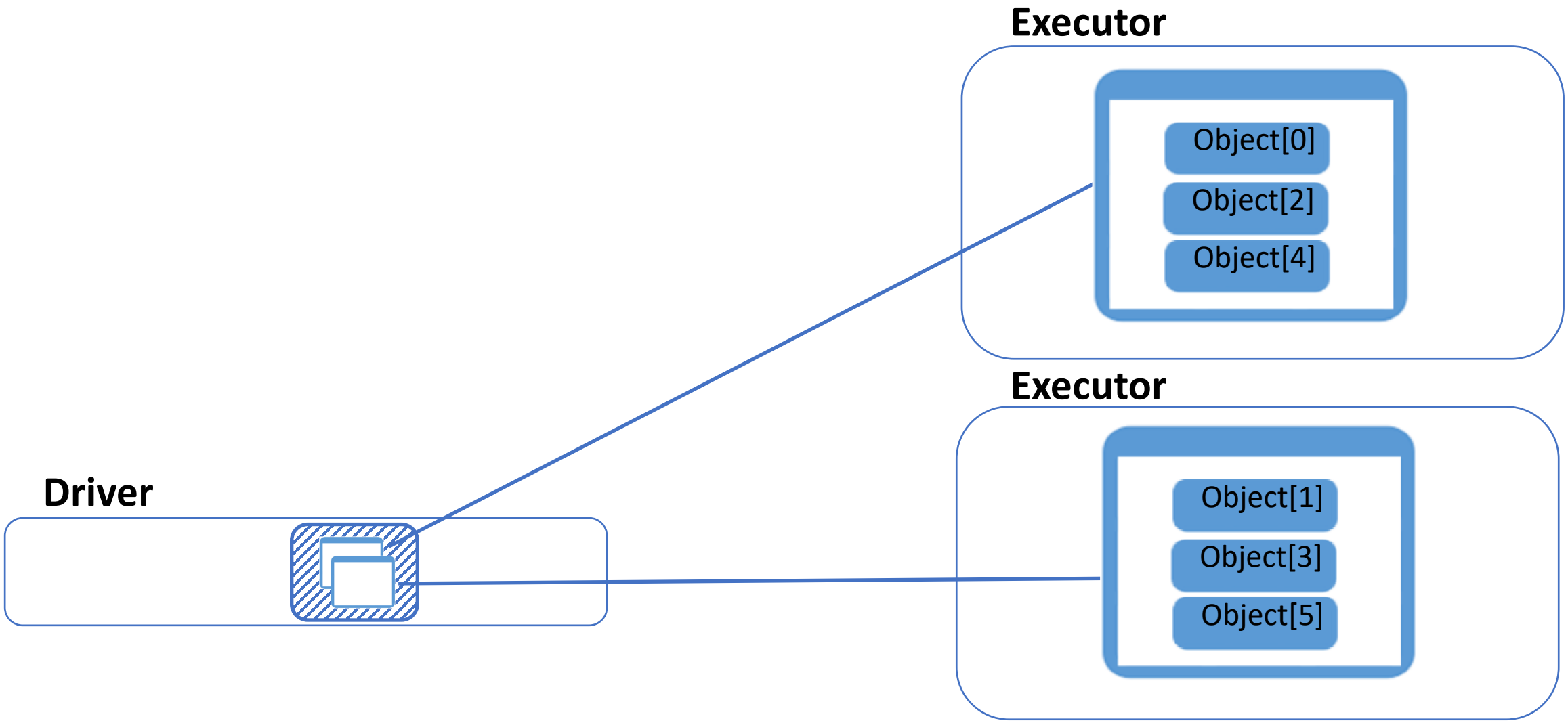
DataSet … Partitions Distributed on Executors



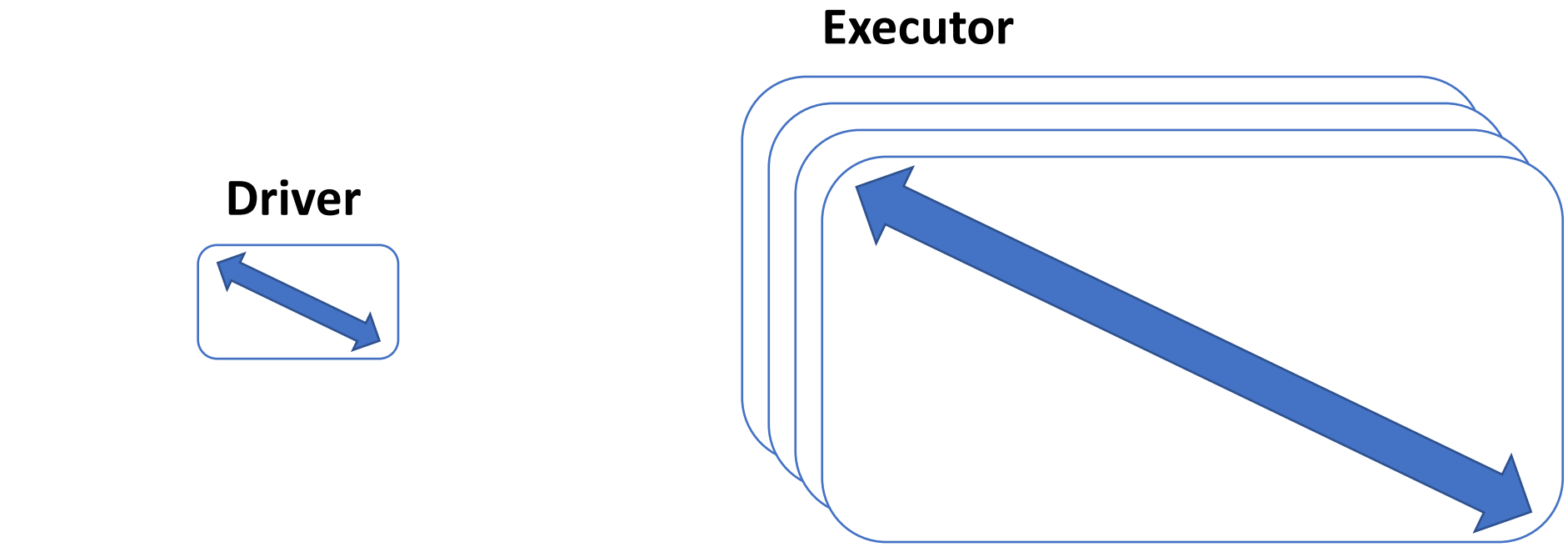
dataset = Spark.createDataset(inmemoryList)
… serialize (send) data to Executors



Memory needs on Driver << Executors



Spark Memory Arguments



--driver-memory 1g

spark.driver.memory=1g

--executor-memory 50g

spark.executor.memory=50g

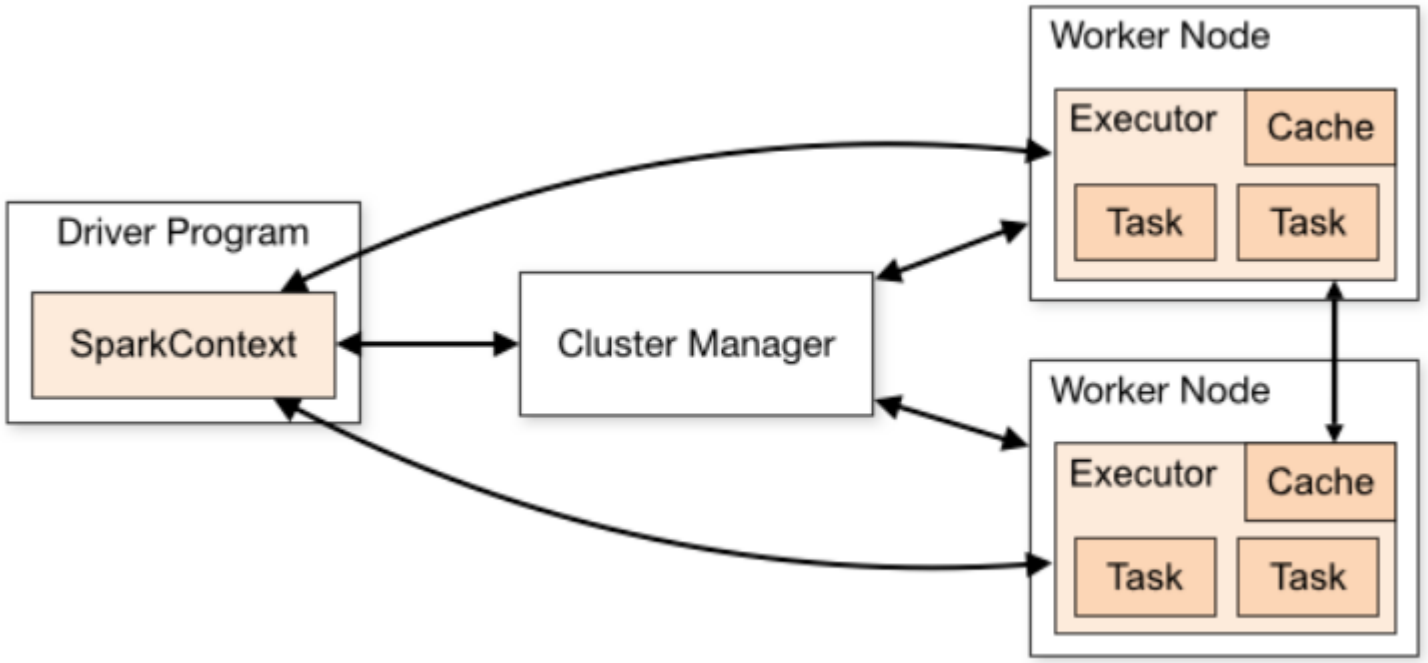
Spark Memory …
to {driver|executor} java –Xms .. –Xmx ..

`spark.driver.memory`	1g	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`). *Note:* In client mode, this config must not be set through the `SparkConf` directly in your application, because the driver JVM has already started at that point. Instead, please set this through the `--driver-memory` command line option or in your default properties file.
`spark.executor.memory`	1g	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`).

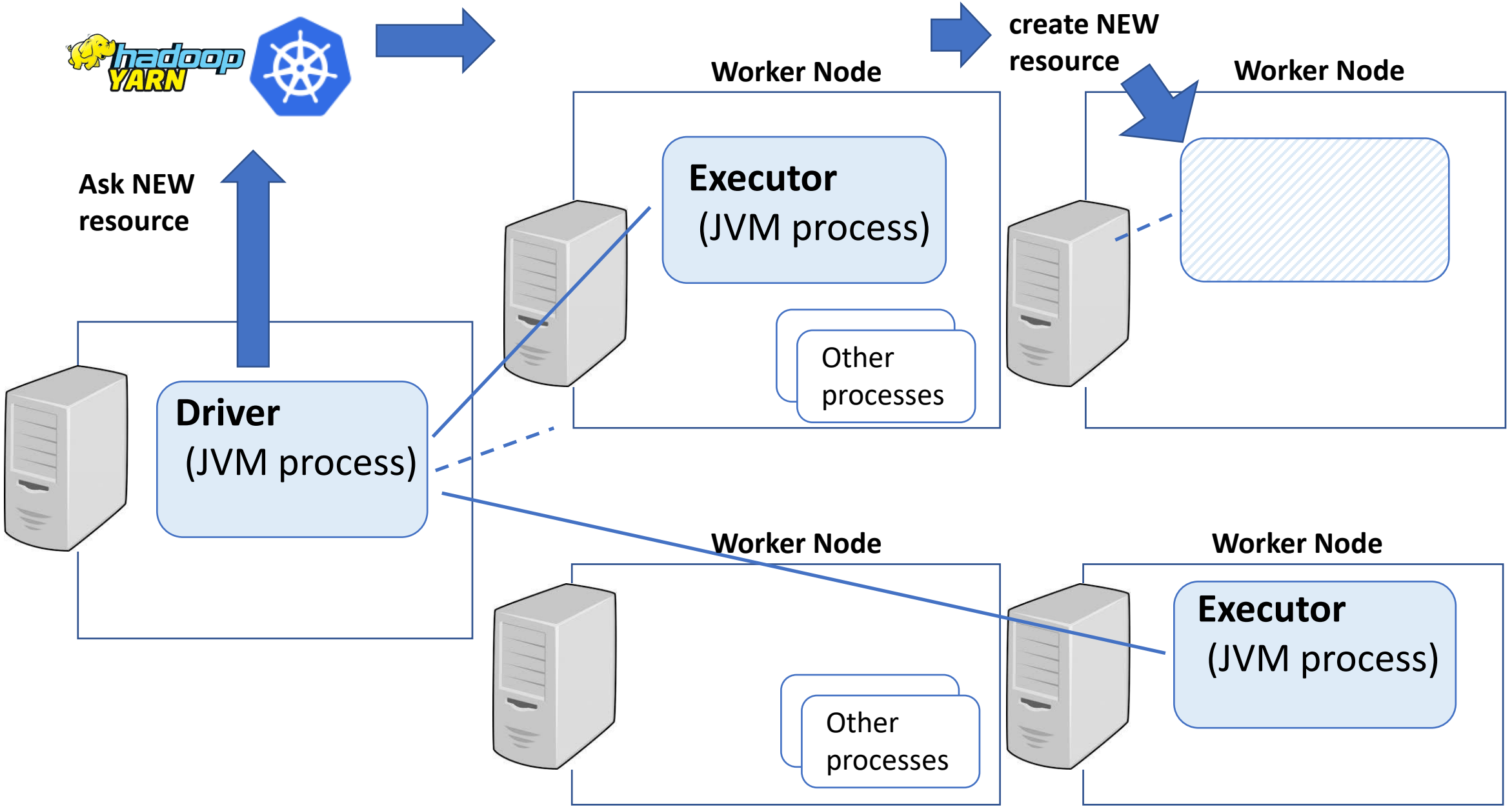
Use Cluster Resources :

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>/

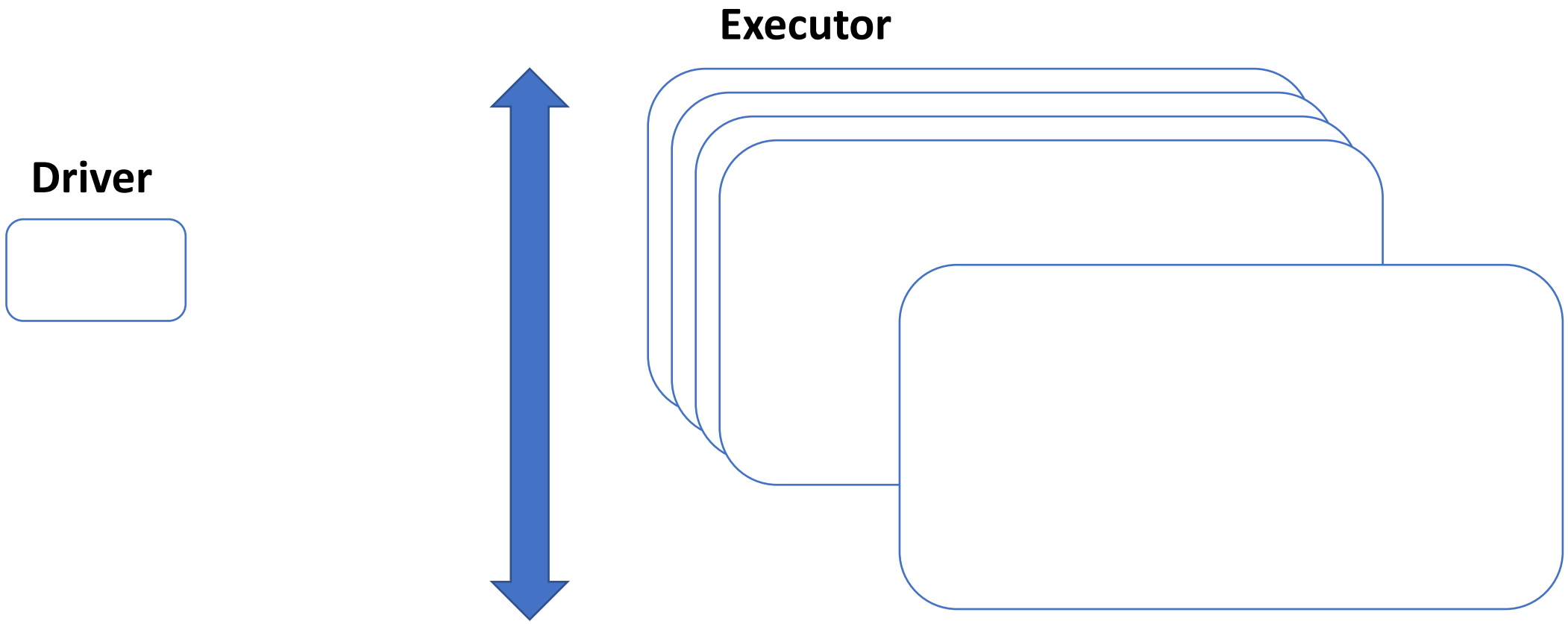
Main class

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



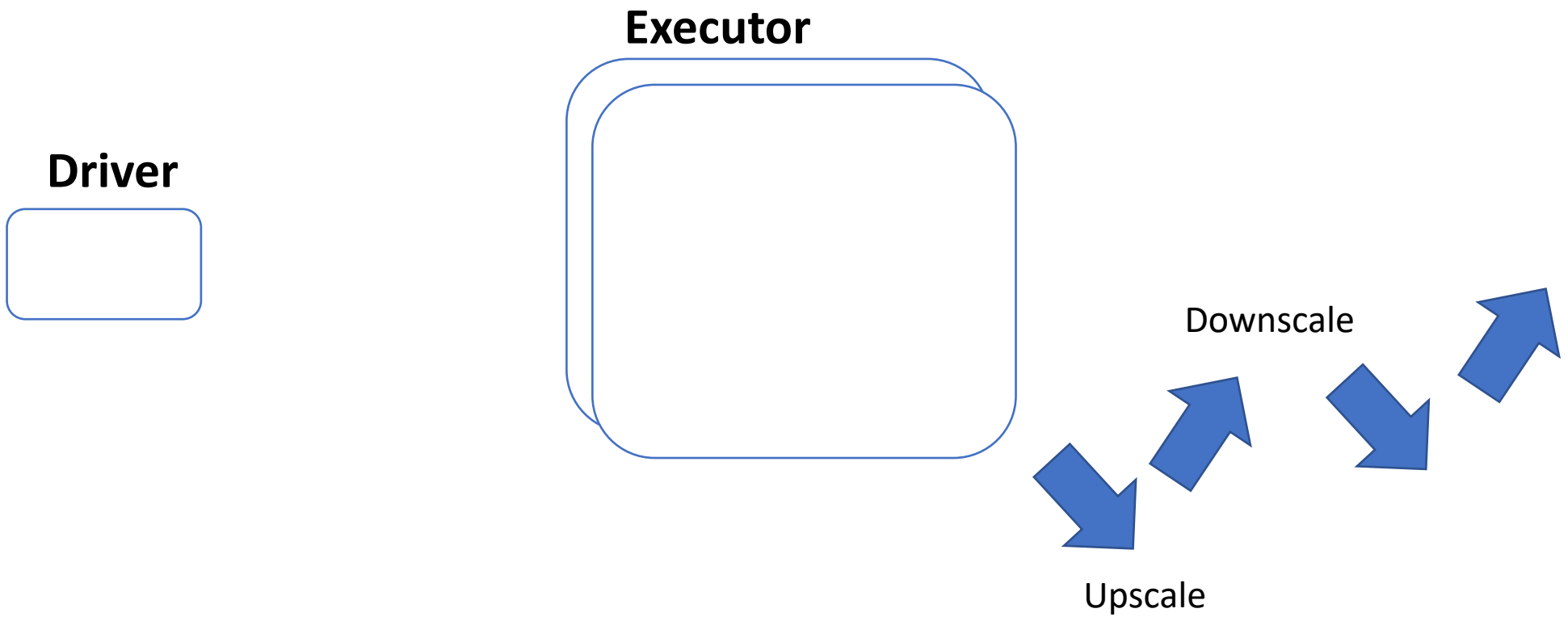
spark.executor.instances=5

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

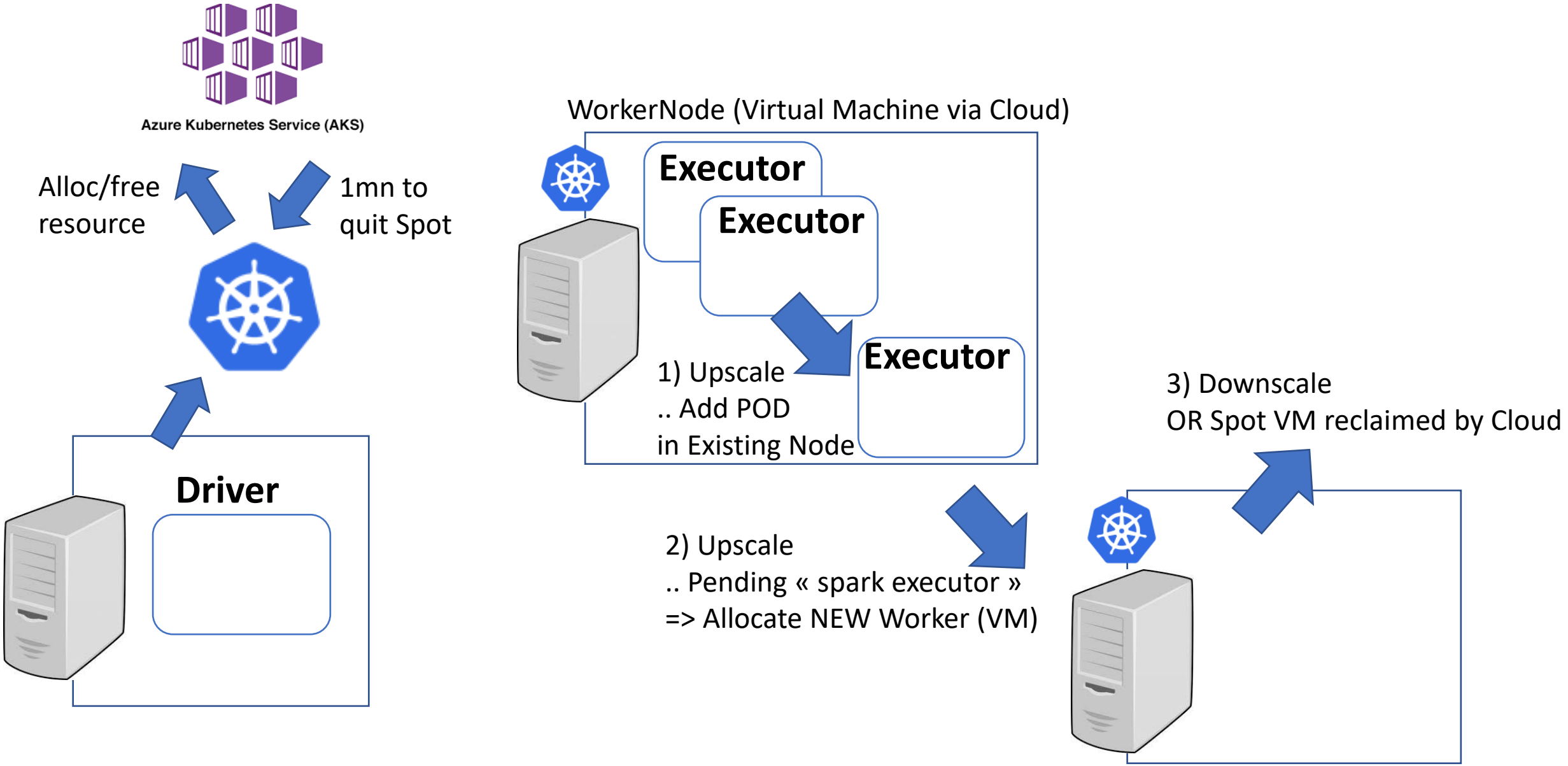
--spark.dynamic.allocation.{min | max | initialExecutors | executorIdleTimeout | … }

Dynamic Allocation arguments

Dynamic Allocation

Property Name	Default	Meaning	Since Version
`spark.dynamicAllocation.enabled`	false	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. 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` `spark.dynamicAllocation.executorAllocationRatio`	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`	Initial number of executors to run if dynamic allocation is enabled. If `--num-executors` (or `spark.executor.instances`) is set and larger than this value, it will be used as the initial number of executors.	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

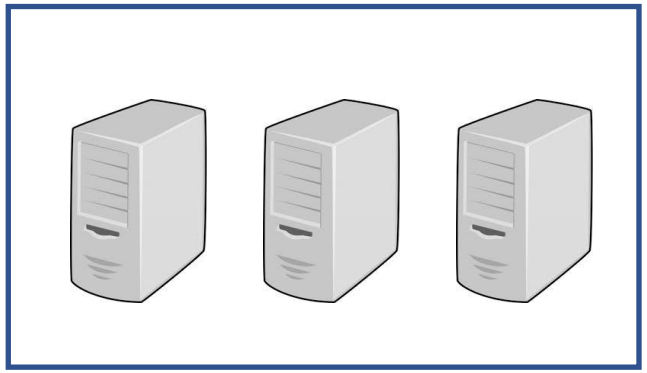
Dynamic Spark + Dynamic K8s Cloud Cluster



K8s NodePool, Afinity, Toleration, Spot VM ..

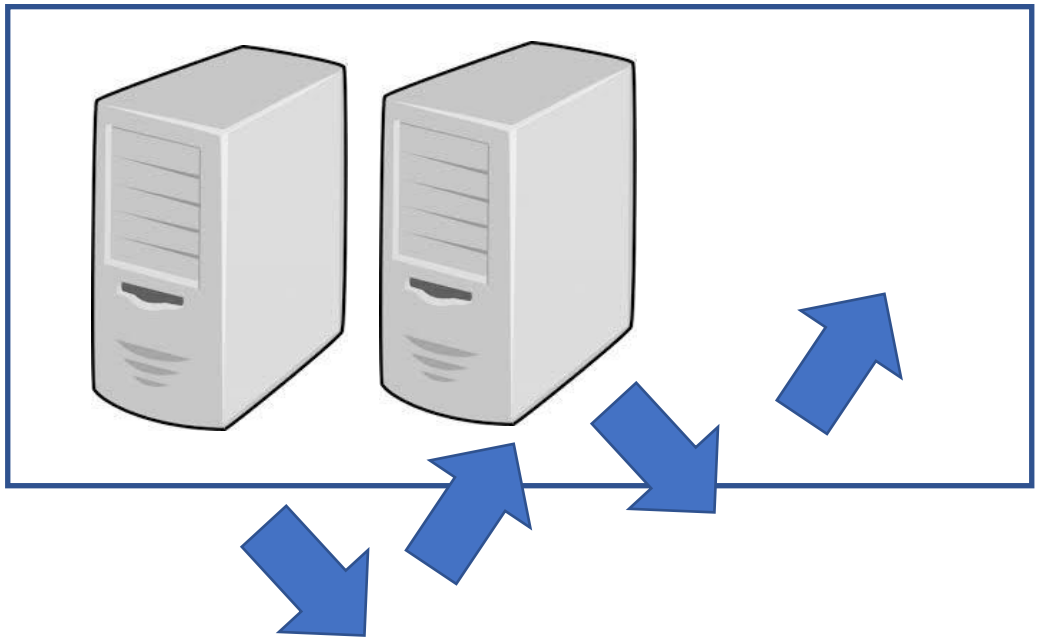
Node Pool 1

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



Node Pool 2

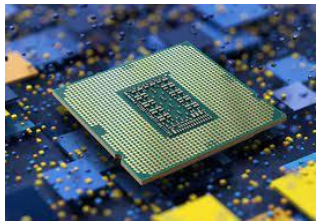
Big VMs … « SPOT »
(cheaper, may be reclaimed by cloud)
=> for spark Executors



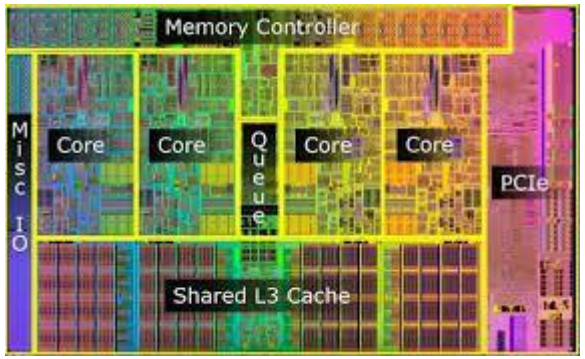
Use CPU Resource: 100% or sharing ?

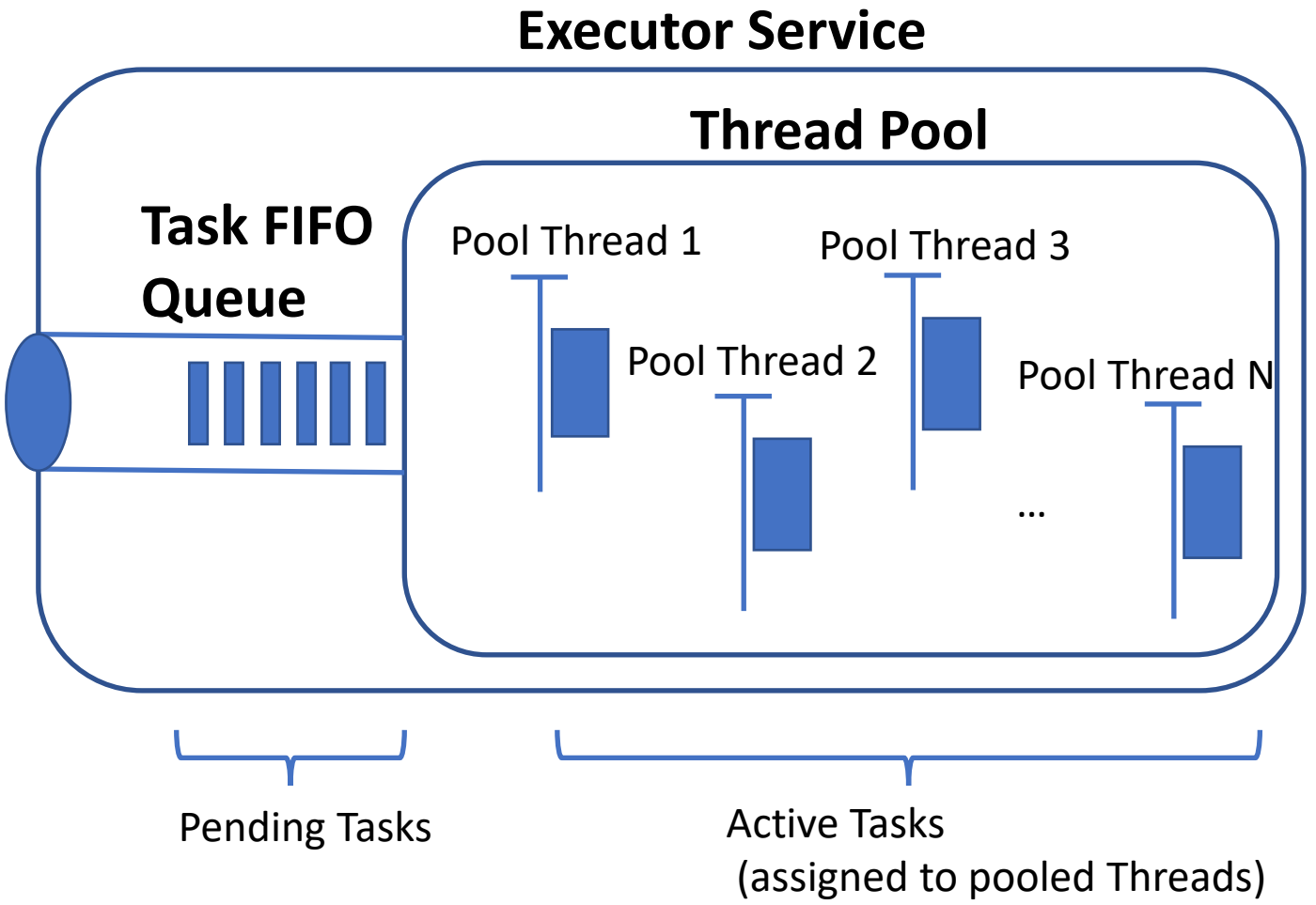
Cpu Cores <-> Threads <-> max Active Tasks

CPU

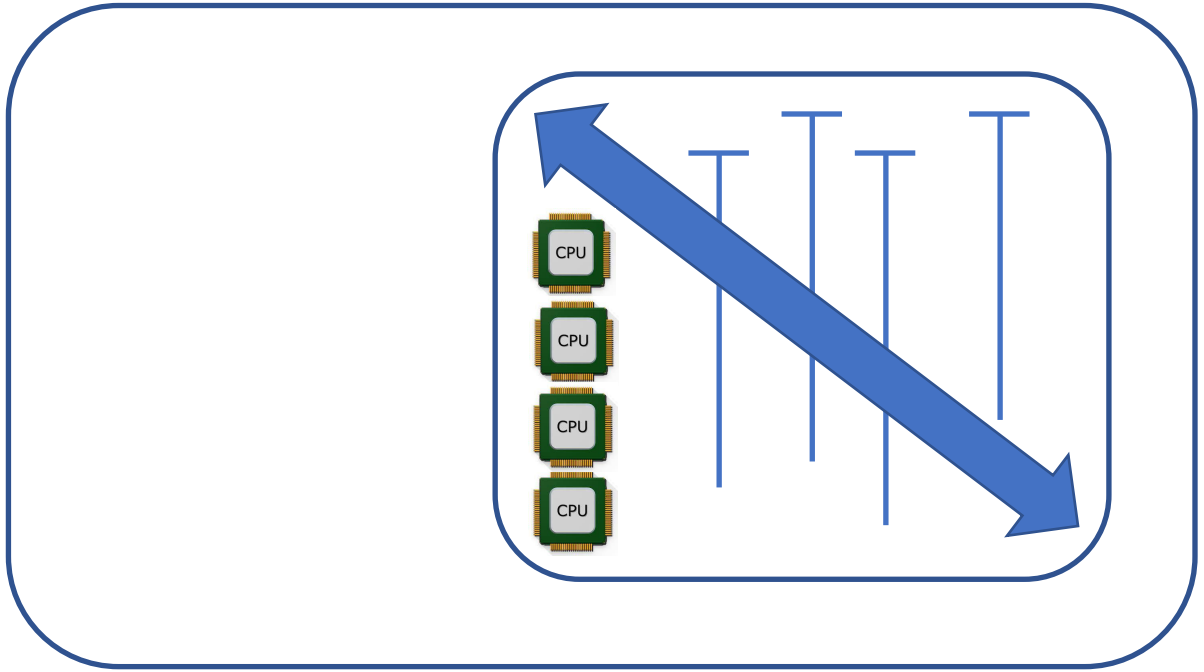


x Cores





Executor Cores argument

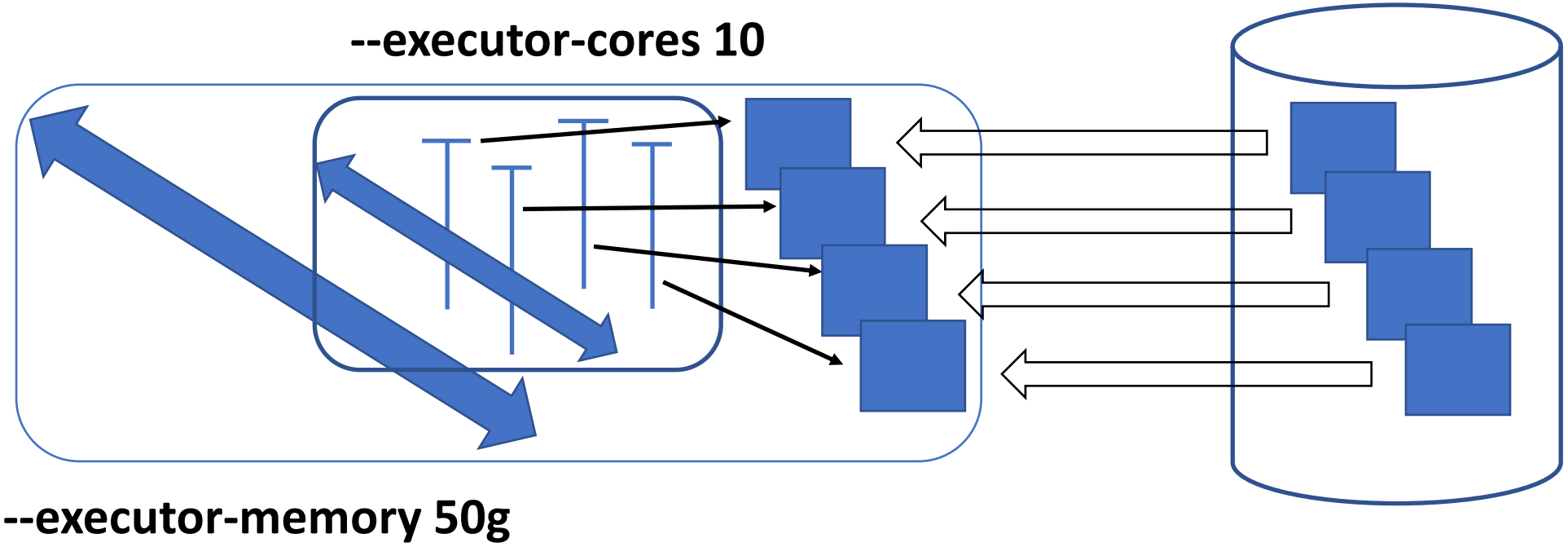


`spark.executor.cores`	1 in YARN mode, all the available cores on the worker in standalone and Mesos coarse-grained modes.	The number of cores to use on each executor. In standalone and Mesos coarse-grained modes, for more detail, see this description.

--executor-cores N

spark.executor.cores=N

Relationship executor .cores <-> .memory

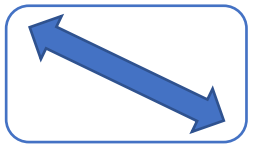


1 Active Core => 1 Task => read 1 File Block => 1 Dataset Partition

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

Summary main arguments

Driver



--driver-memory 1g

spark.driver.memory=1g

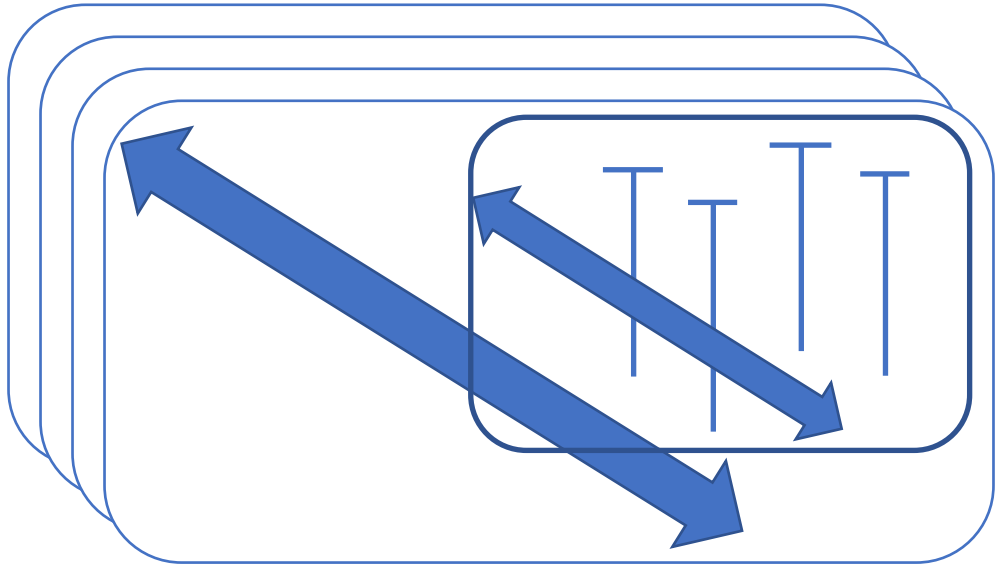
--num-executors 5

spark.executor.instances=5

Executor

--executor-cores N

spark.executor.cores=N



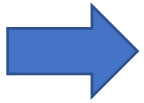
--executor-memory 50g

spark.executor.memory=50g

Outline

Reminder Spark-driver / spark-executers

Cluster Tuning Params



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

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

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

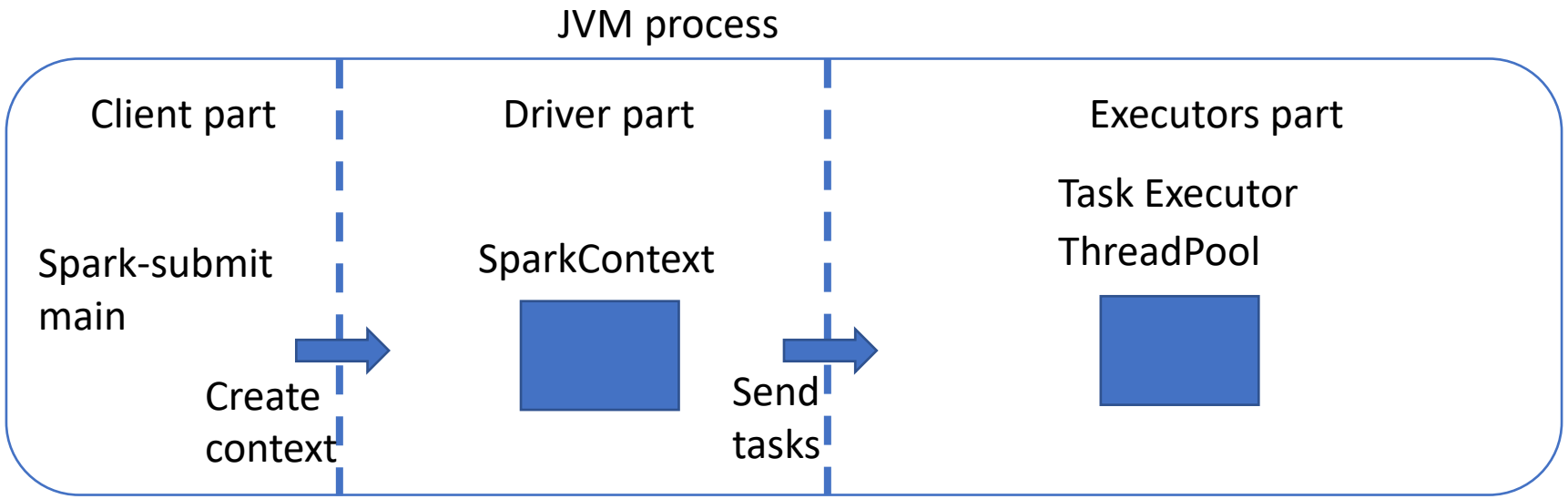
${mode}: one of

- **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
```

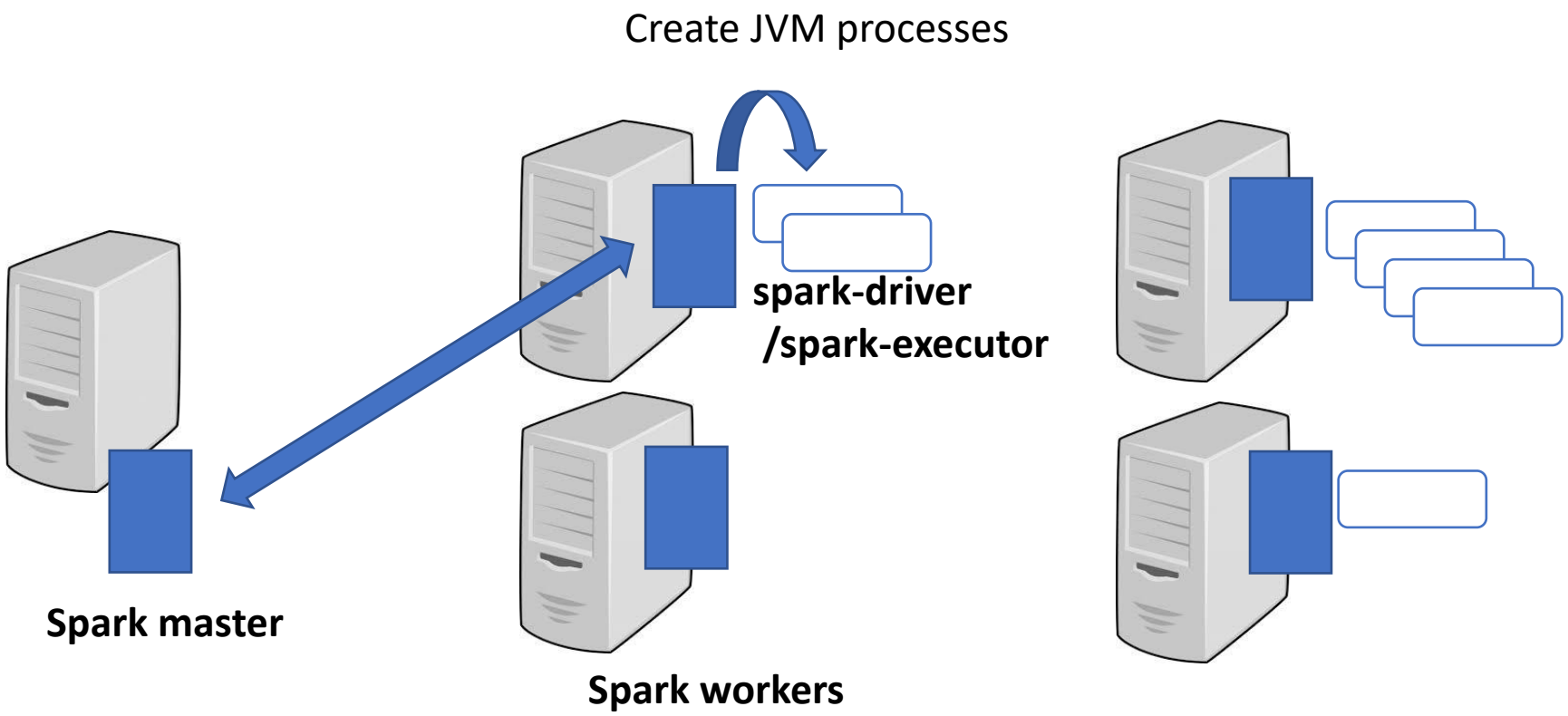
Local mode: 1 JVM
client = driver = executors



--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
```

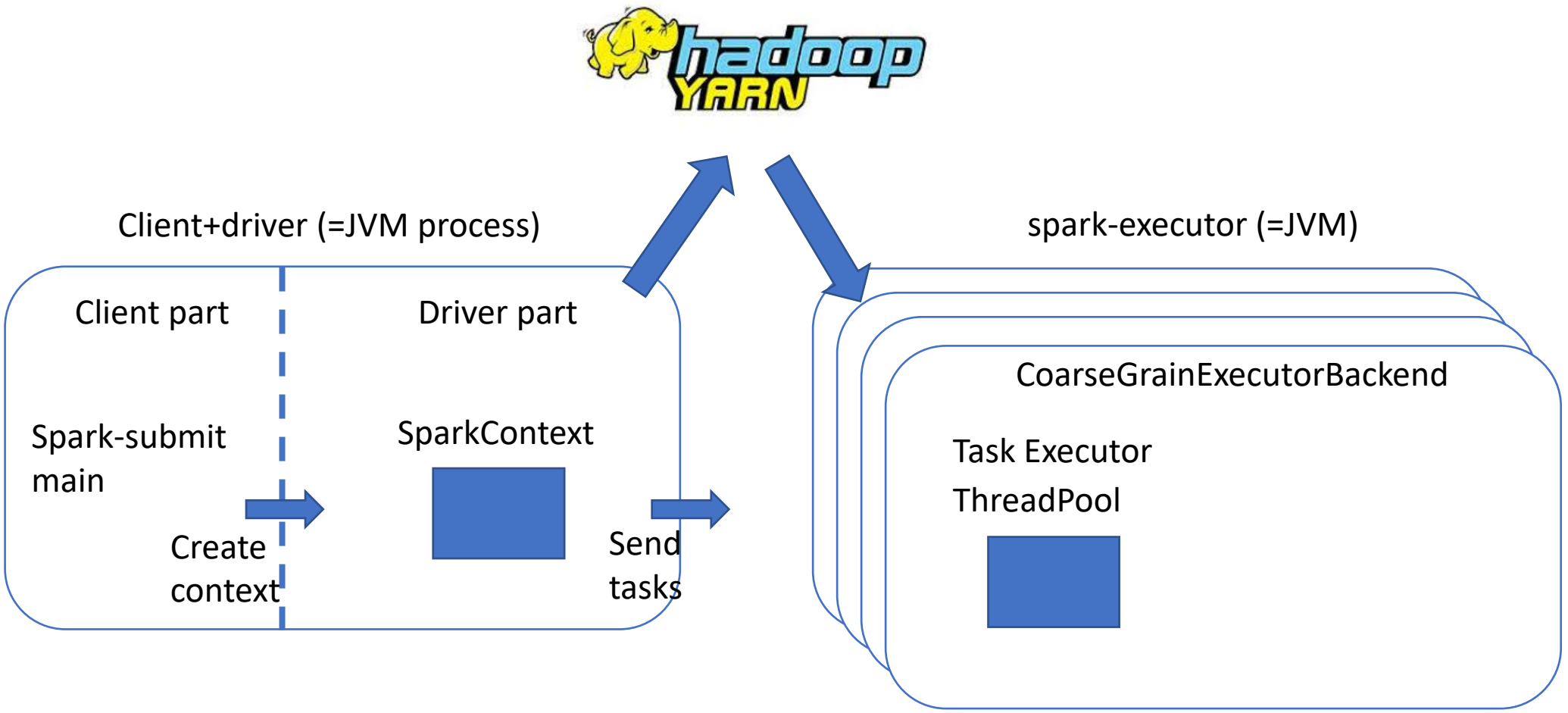
Spark Standalone Cluster



--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

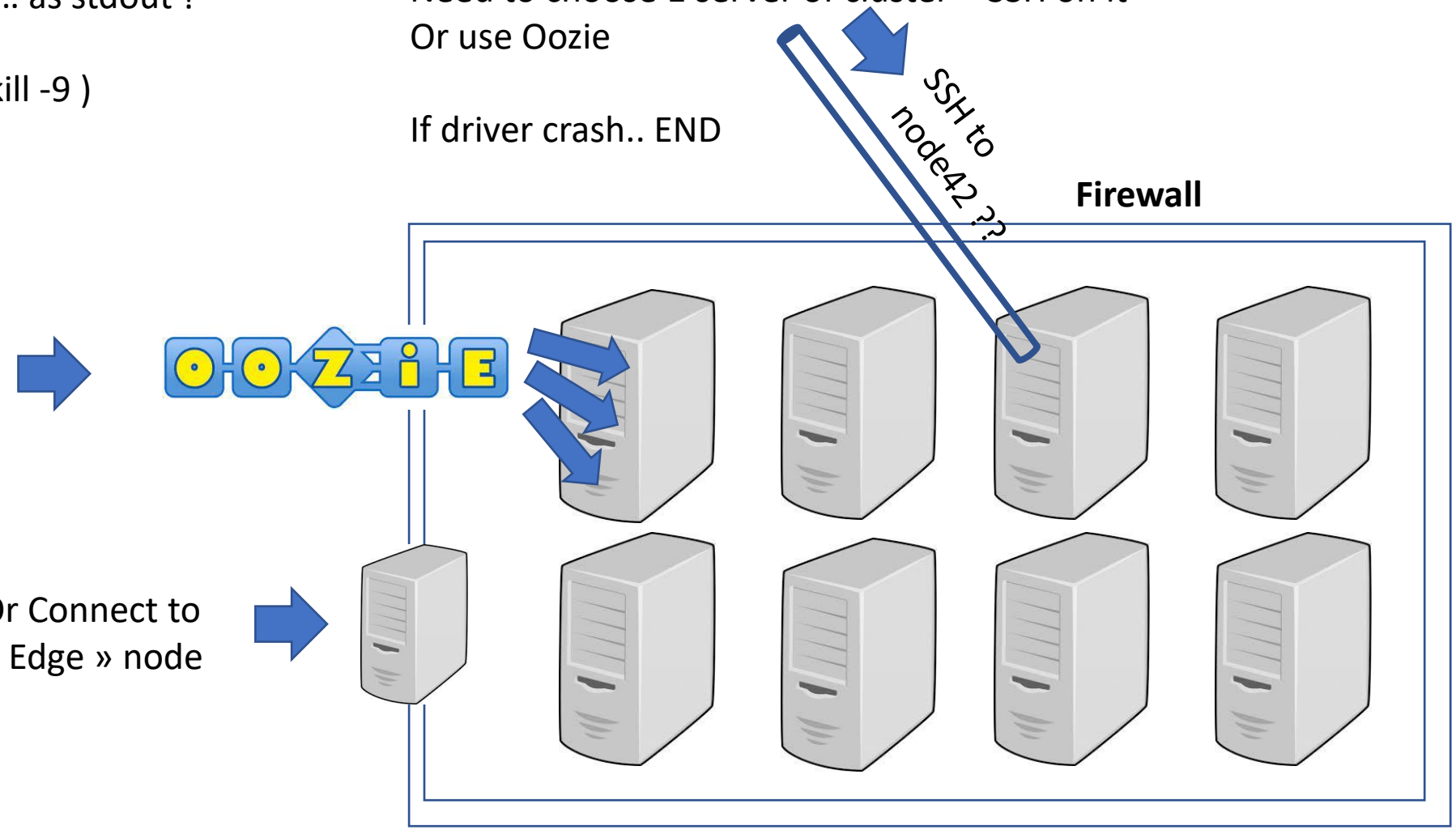
Easy to see driver logs… as stdout !

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

Client = Driver … within firewall

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

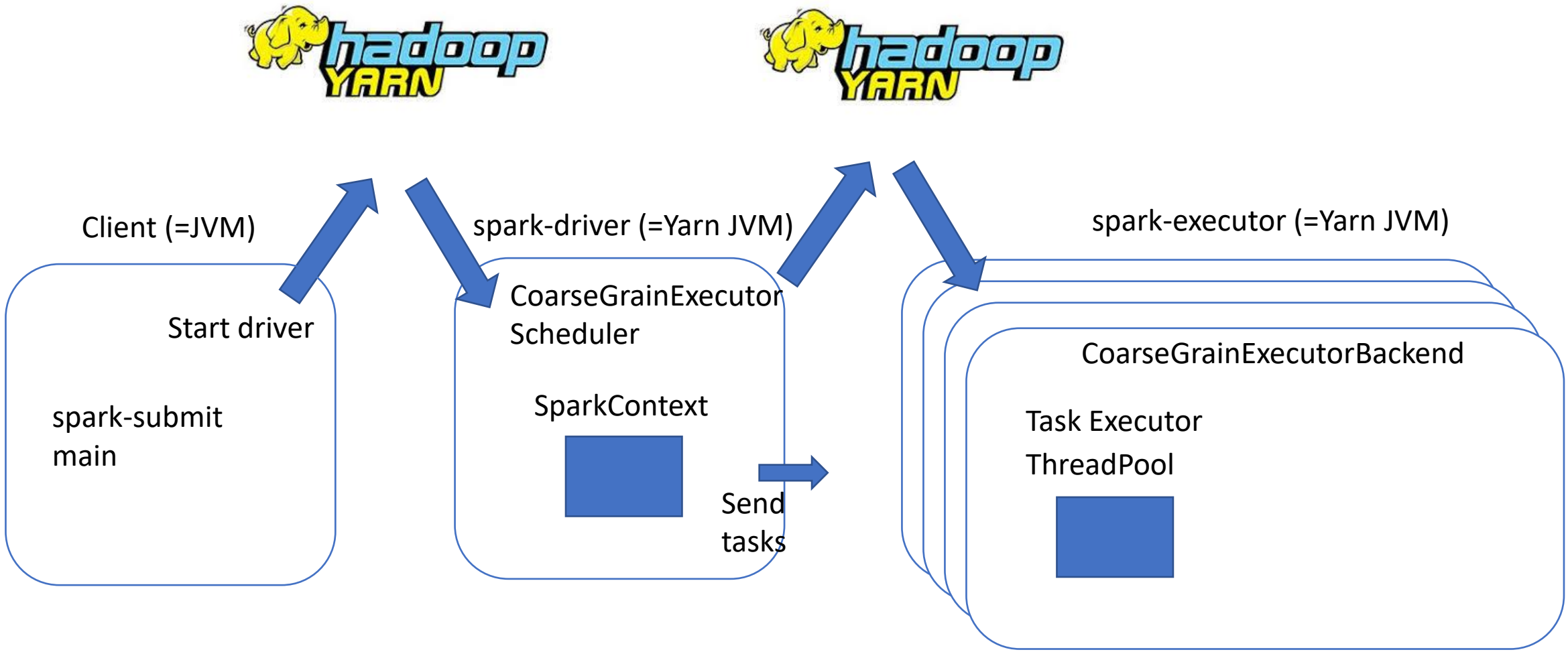
If driver crash.. END



--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

Client is ligtweight
Many Clients can run on « Edge » Node

If Driver crashes… client will relaunch driver!

More difficult to see logs (Yarn Logs!!)

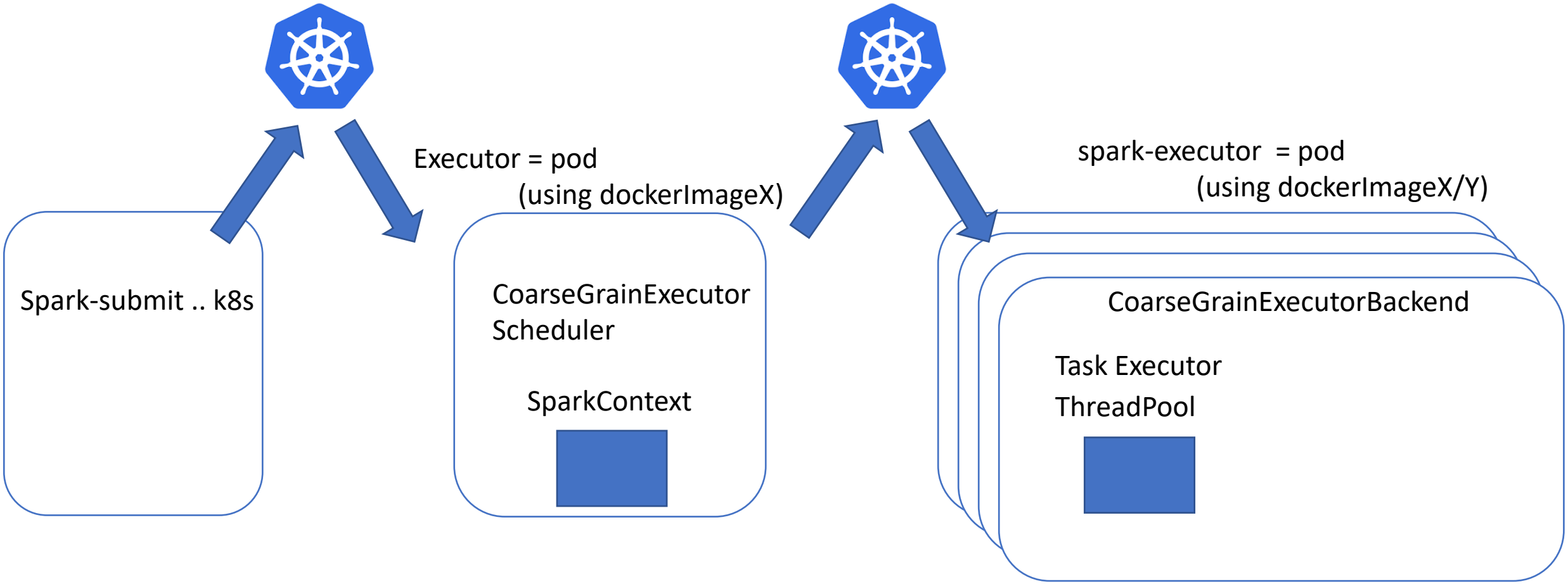
More difficult to kill : yarn app –kill ${applicationId}

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

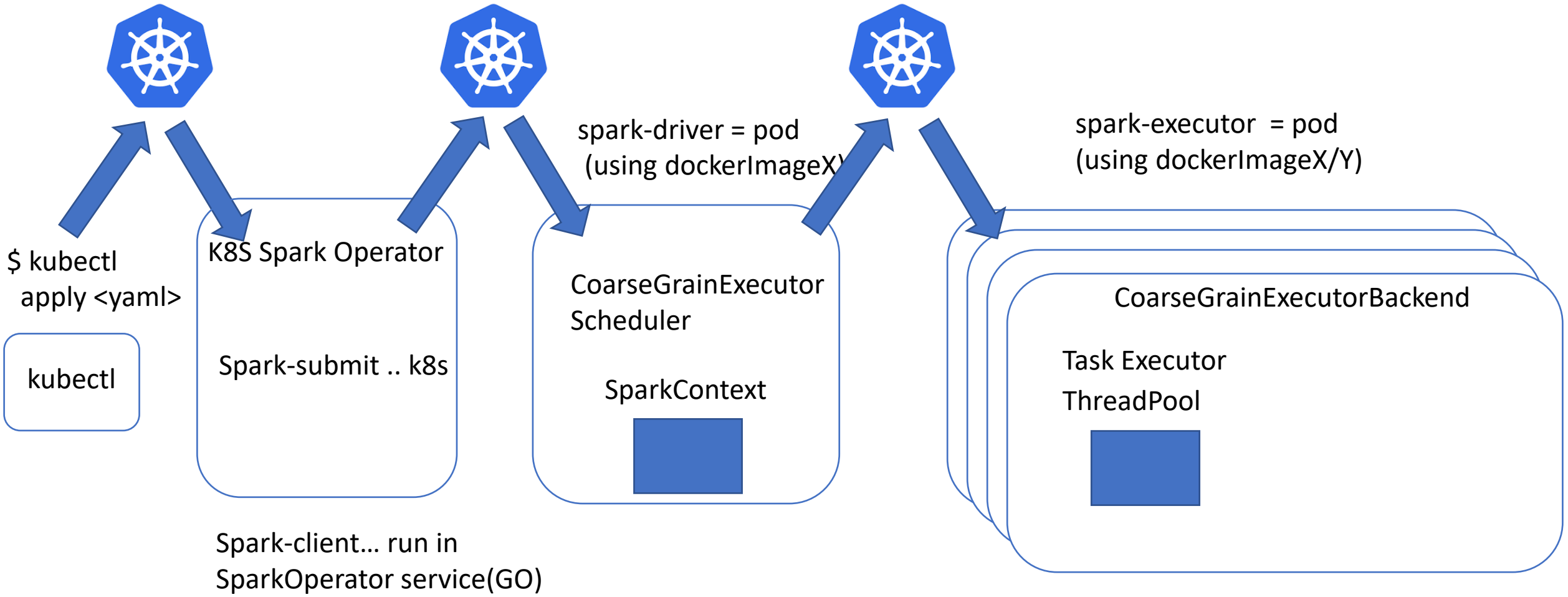
Spark on Kubernetes … Docker Images/Jars



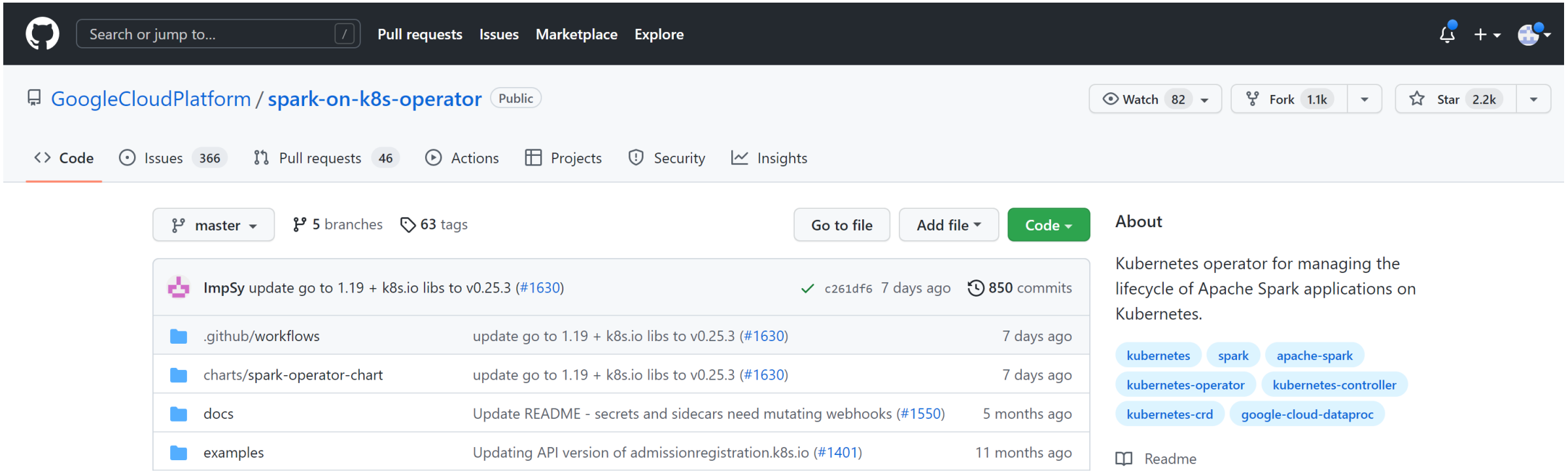
Spark on K8s with SparkOperator mode

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

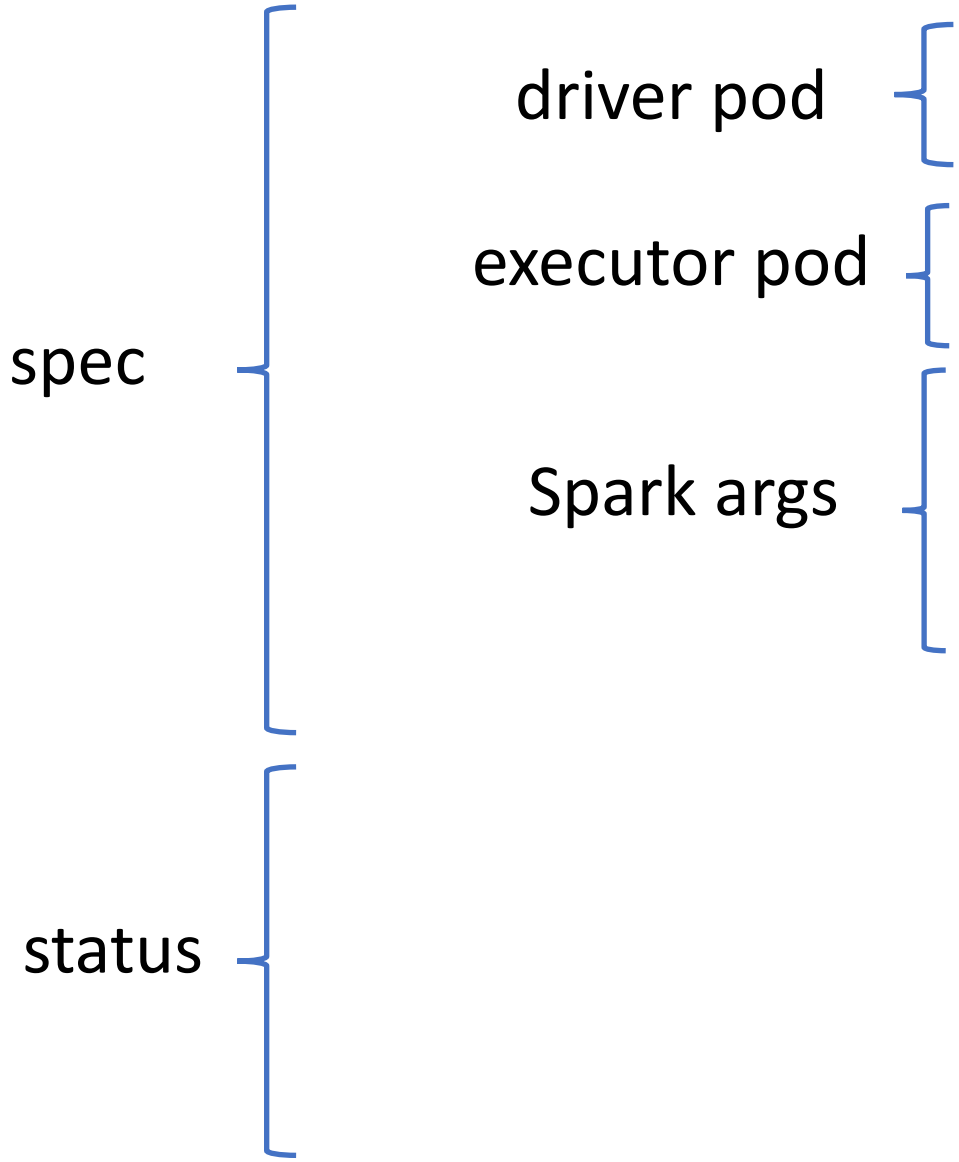
Spark on K8S, using K8s SparkOperator



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



$ kubectl get sparkapplications .. -o=yaml



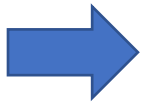

```yaml
apiVersion: sparkoperator.k8s.io/v1beta2
kind: SparkApplication
metadata:
  ...
spec:
  deps: {}
  driver:
    coreLimit: 1200m
    cores: 1
    labels:
      version: 2.3.0
    memory: 512m
    serviceAccount: spark
  executor:
    cores: 1
    instances: 1
    labels:
      version: 2.3.0
    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-executers

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 »

```java
import org.apache.spark.sql.Dataset;
import org.apache.spark.sql.Encoders;
import org.apache.spark.sql.SparkSession;

public class SimpleMain {

    public static void main(String[] args) {
        System.out.println("ensure HADOOP_HOME env var is set, (and Winutils on windows)");
        System.out.println("HADOOP_HOME: " + System.getenv("HADOOP_HOME"));

        System.out.println("start embedded SparkContext - SparkSession");
        SparkSession spark = SparkSession.builder()
                .master("local[*]")
                .appName("myapp")
                .getOrCreate();

        List<String> ls = Arrays.asList("Hello spark");
        Dataset<String> ds = spark.createDataset(ls, Encoders.STRING());
        ds.show();

        System.out.println("finish SparkContext");
        spark.close();
    }
}
```

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)

1

activeContext

0..1

```
class SparkContext {
…
}
```

new RDD + transforms

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

1

```
class SparkSession {
…
}
```

getOrCreate()

new Dataset<> + transforms

JavaSparkContext, JavaRDD, JavaFunction ?

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

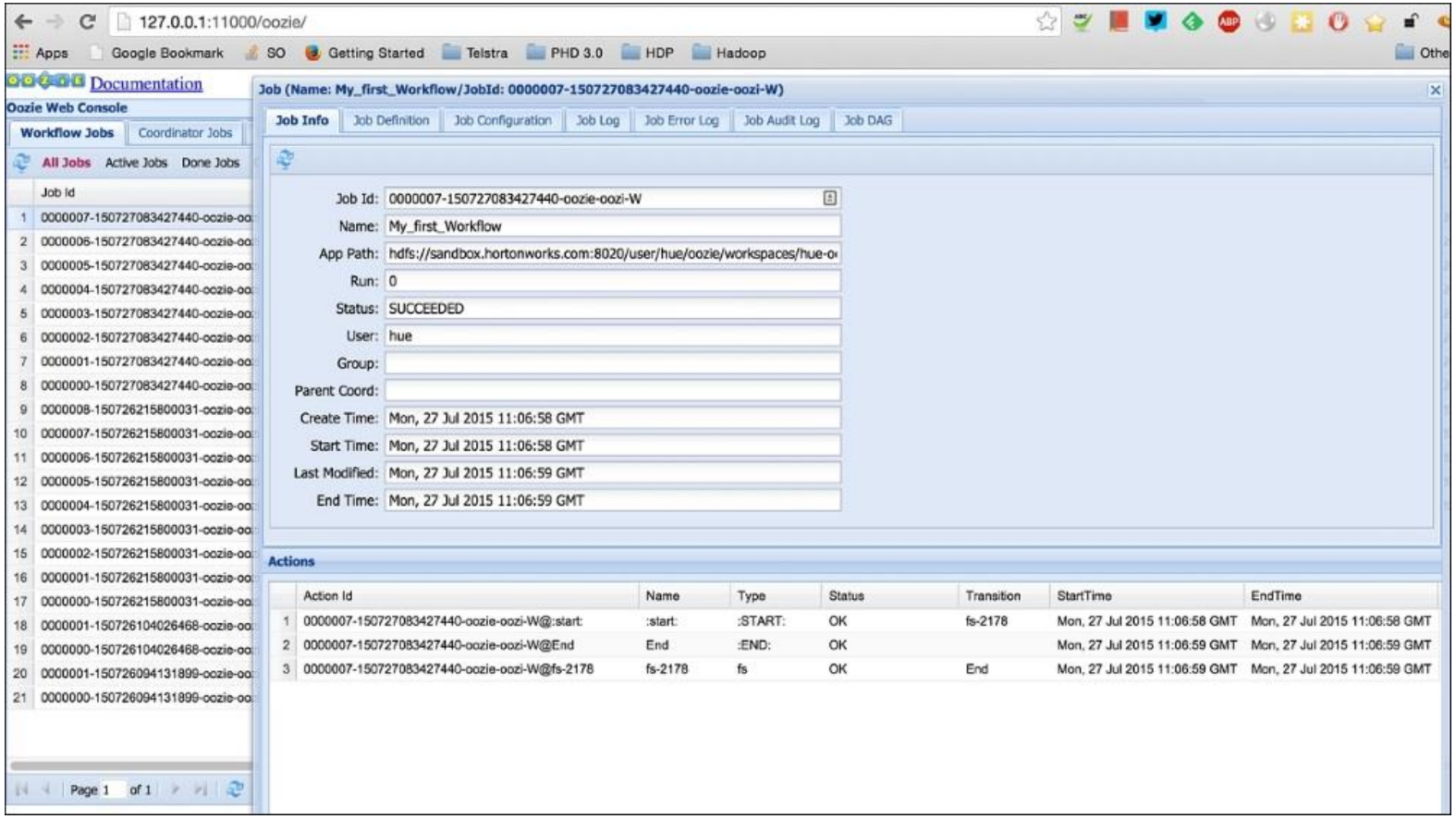
some specific 

helper classes have been added, for Java developpers

some deprecated, ex: SQLContext

Oozie … http Api wrapper for Yarn + UI

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



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

$ **curl** –X POST /oozie/job –d @config.properties

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>

Oozie Spark Action Extension

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.

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>
```

LIVY

https://livy.apache.org/

LIVY Apache Livy

Get Started | Documentation | Community | Apache

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

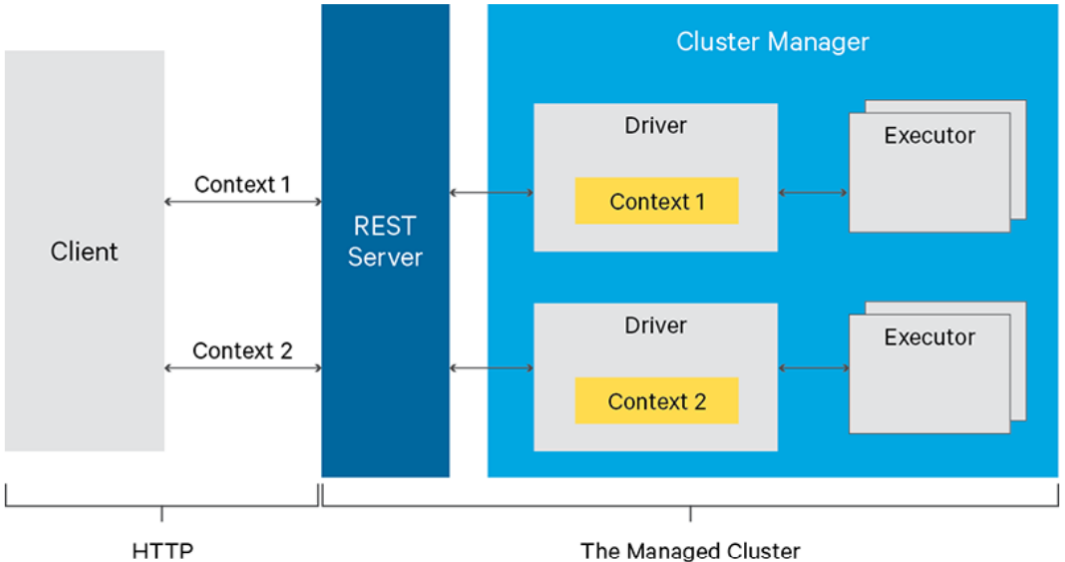
Don't worry, no changes to existing programs are needed to use Livy. Just build Livy with Maven, deploy the configuration file to your Spark cluster, and you're off! Check out Get Started to get going.

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

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

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

${id}

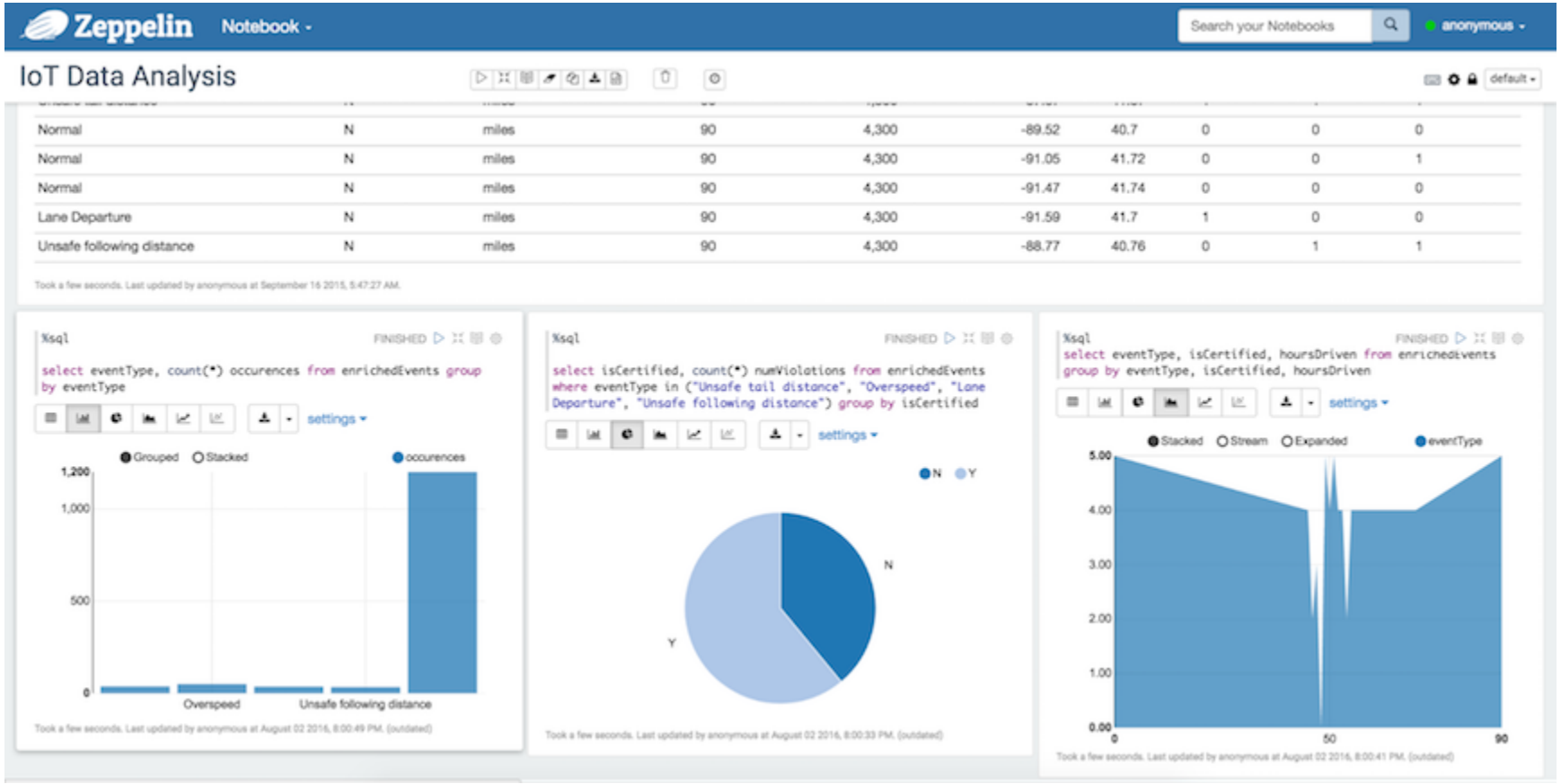
$ **statResp = $(curl –x POST http://<livyhost>:8998/sessions/ 2 /statements -d statement-req.json)**

statement-req.json

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

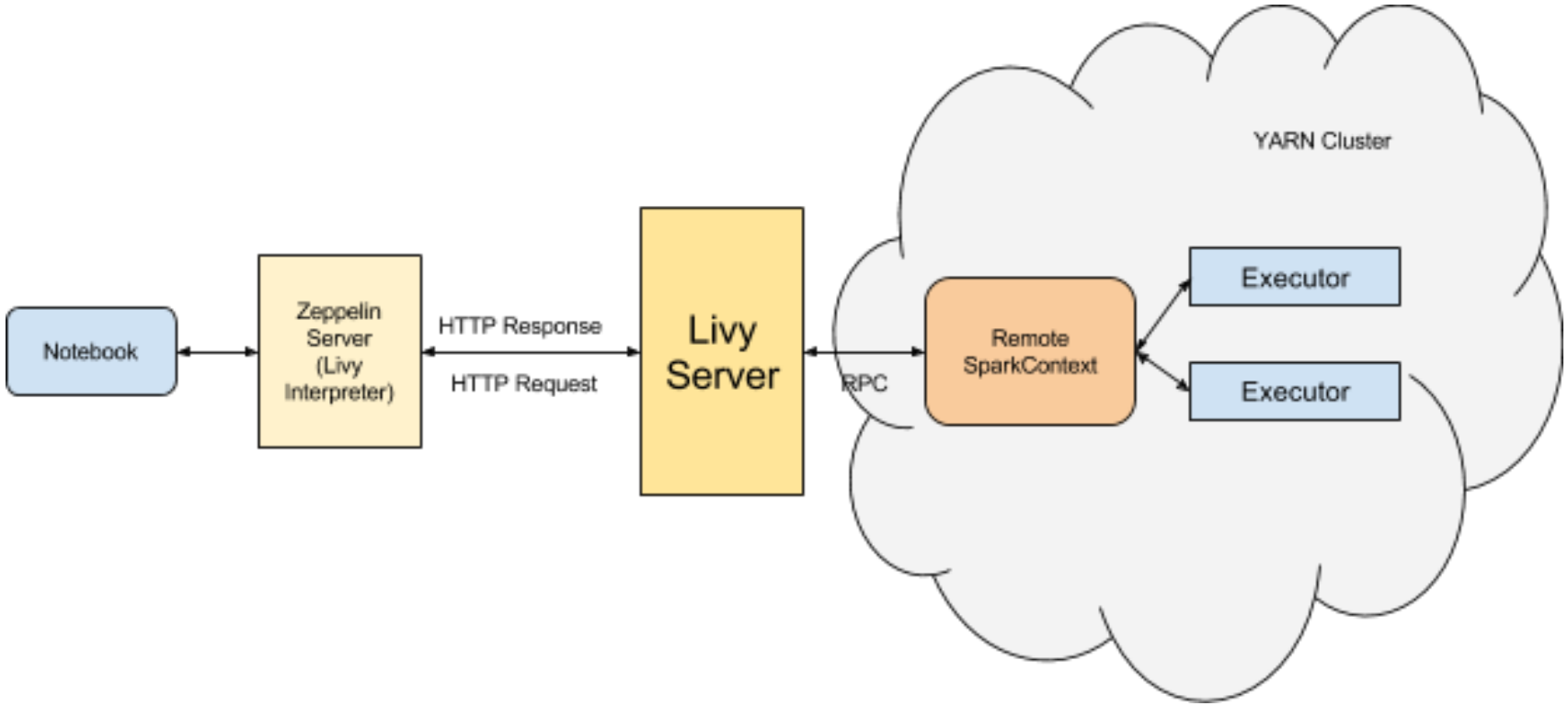
Apache Zeppelin

Zeppelin



Apache Zeppelin

Zeppelin -> Livy Interactive Session



Config: livy.spark.*

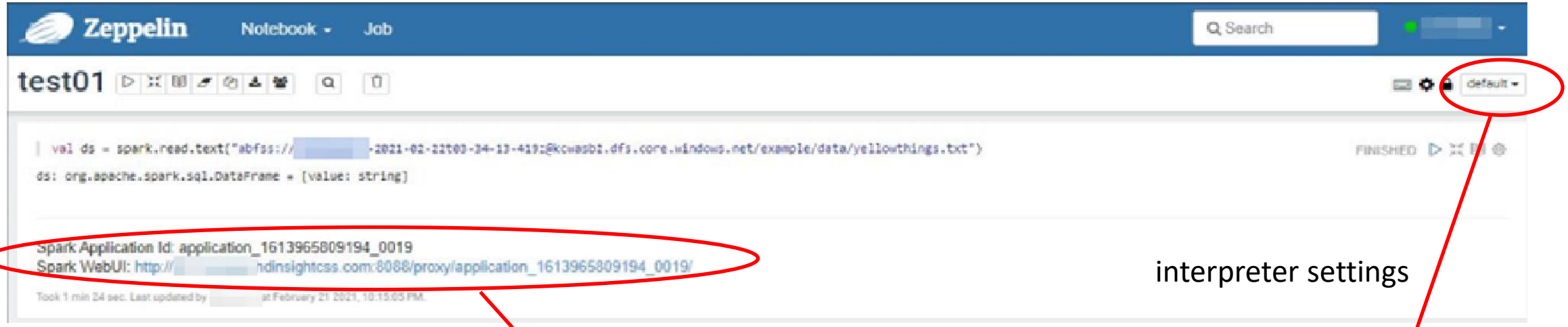
Zeppelin 0.10.1 Quick Start Usage Setup Interpreter More

Configuration

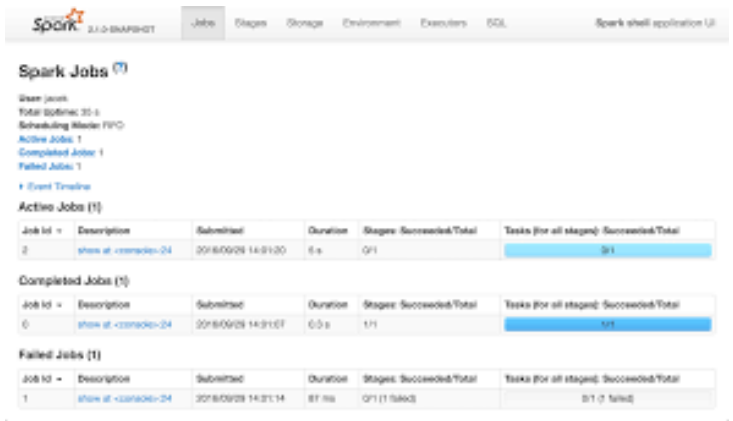
We added some common configurations for spark, and you can set any configuration you want. You can find all Spark configurations in here. And instead of starting property with `spark.` it should be replaced with `livy.spark.`. Example: `spark.driver.memory` to `livy.spark.driver.memory`

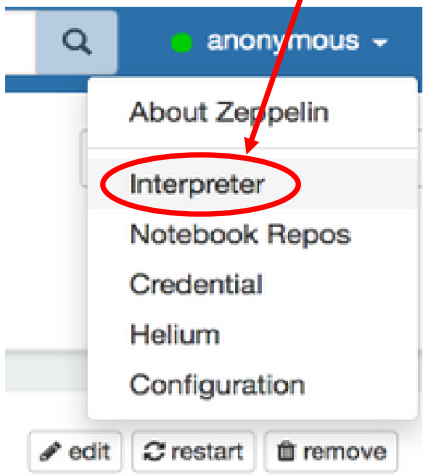
Property	Default	Description
zeppelin.livy.url	http://localhost:8998	URL where livy server is running
zeppelin.livy.spark.sql.maxResult	1000	Max number of Spark SQL result to display.
zeppelin.livy.spark.sql.field.truncate	true	Whether to truncate field values longer than 20 characters or not
zeppelin.livy.session.create_timeout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

Link Zeppelin -> SparkUI Application

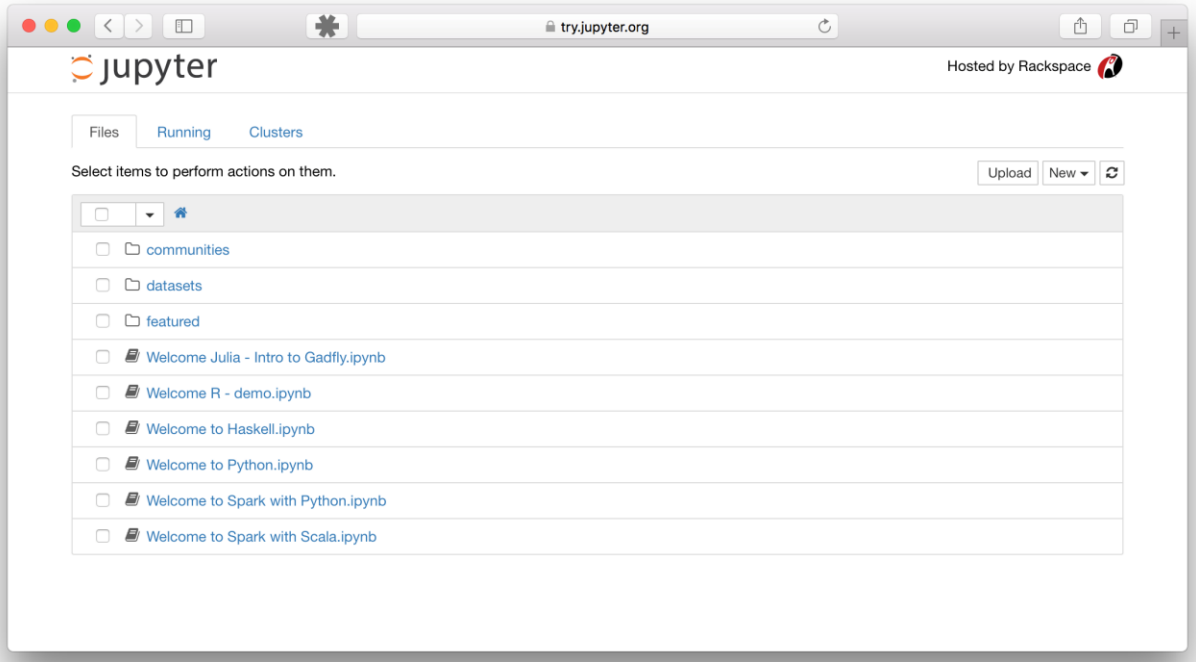


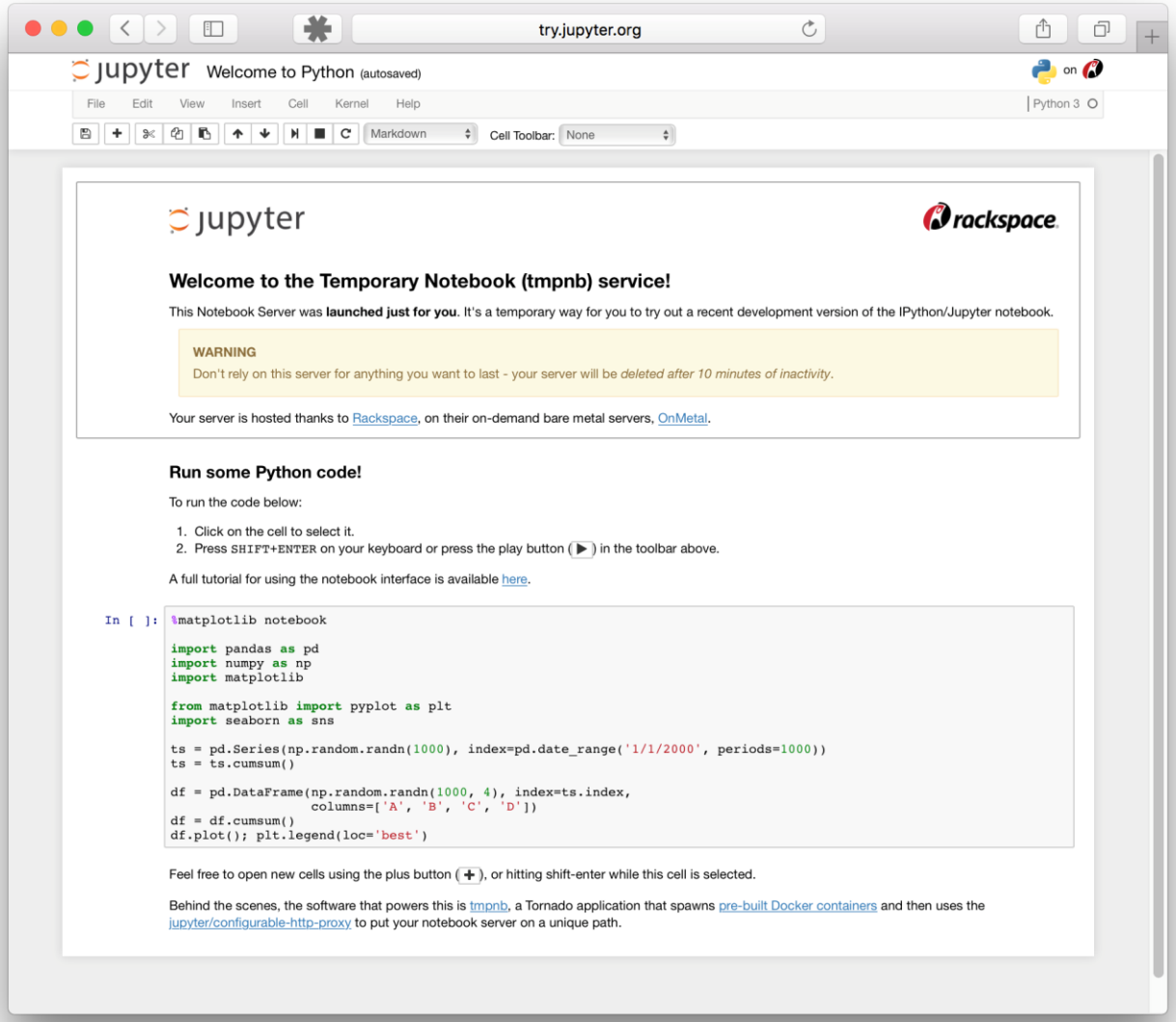
interpreter settings



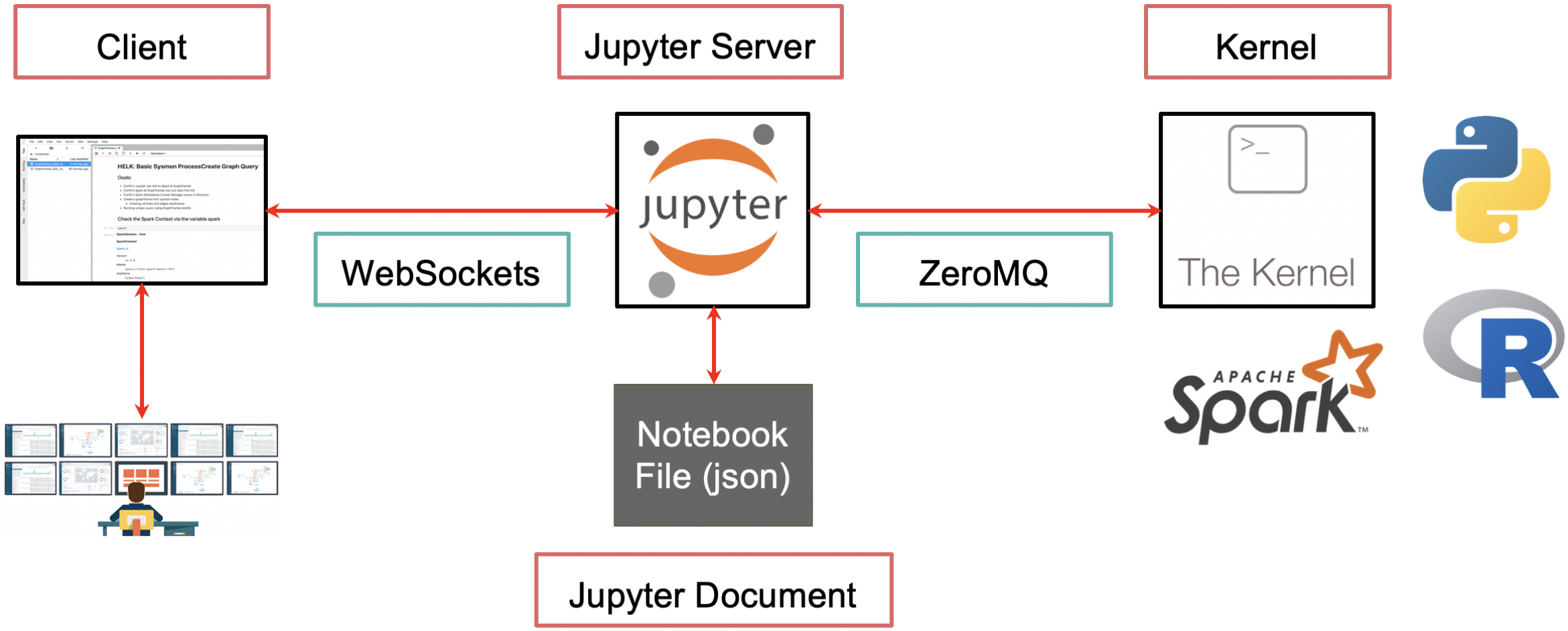


Jupyter Notebook





Jupyter – Kernel(s) architecture



Jupyter + python kernel + pyspark

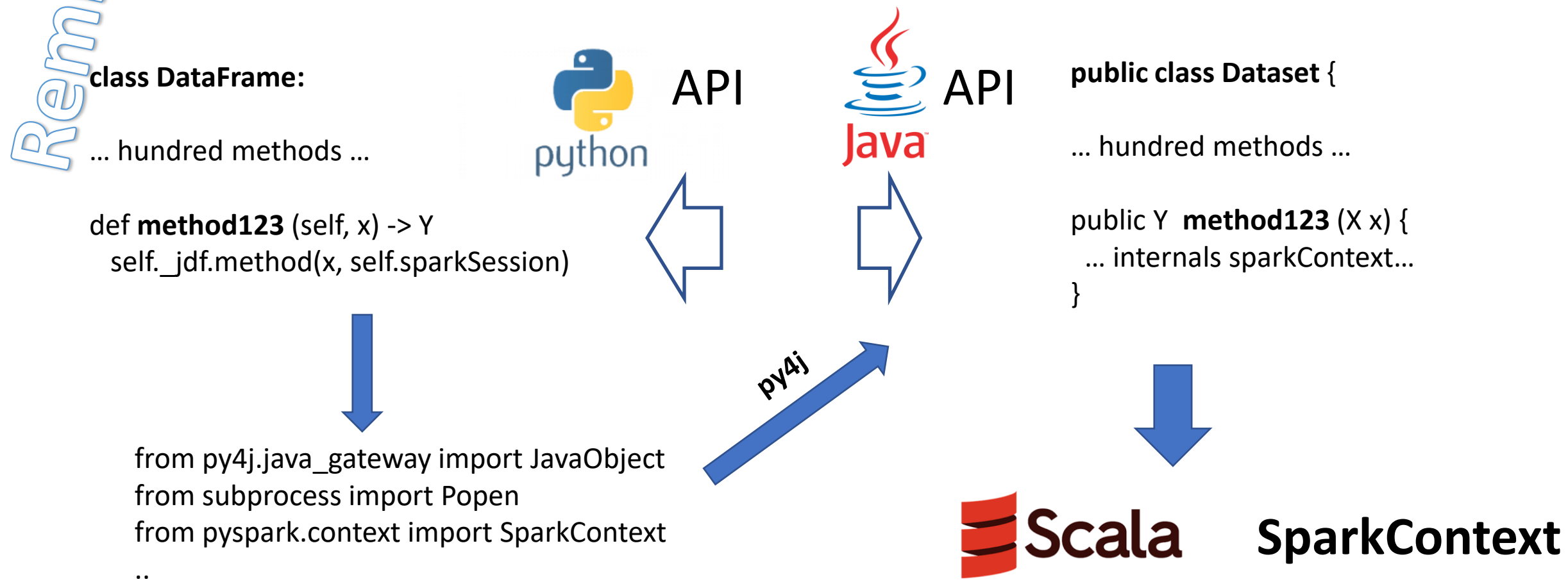
In [11]:
```
from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

sqlContext = SQLContext(sc)
df  = sqlContext.sql("select 1 as A")
df.show()
```

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

PySpark

Reminder

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

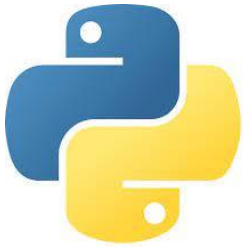


Python + pyspark (GLUE package)
… = spark JVM proces + socket

$ pyspark

SparkContext (spark-driver)

python.exe



Popen(‘java.exe’, …)

socket


```
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, JavaObject, 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
```

Write python Object
to pickle format

read java Object
from pickle format

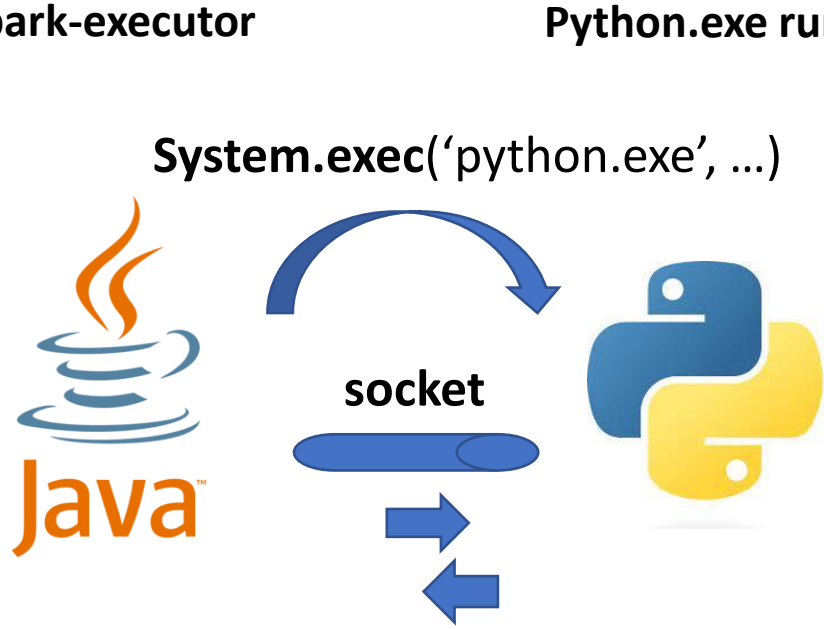
Py4J https://www.py4j.org/

Python UDF function

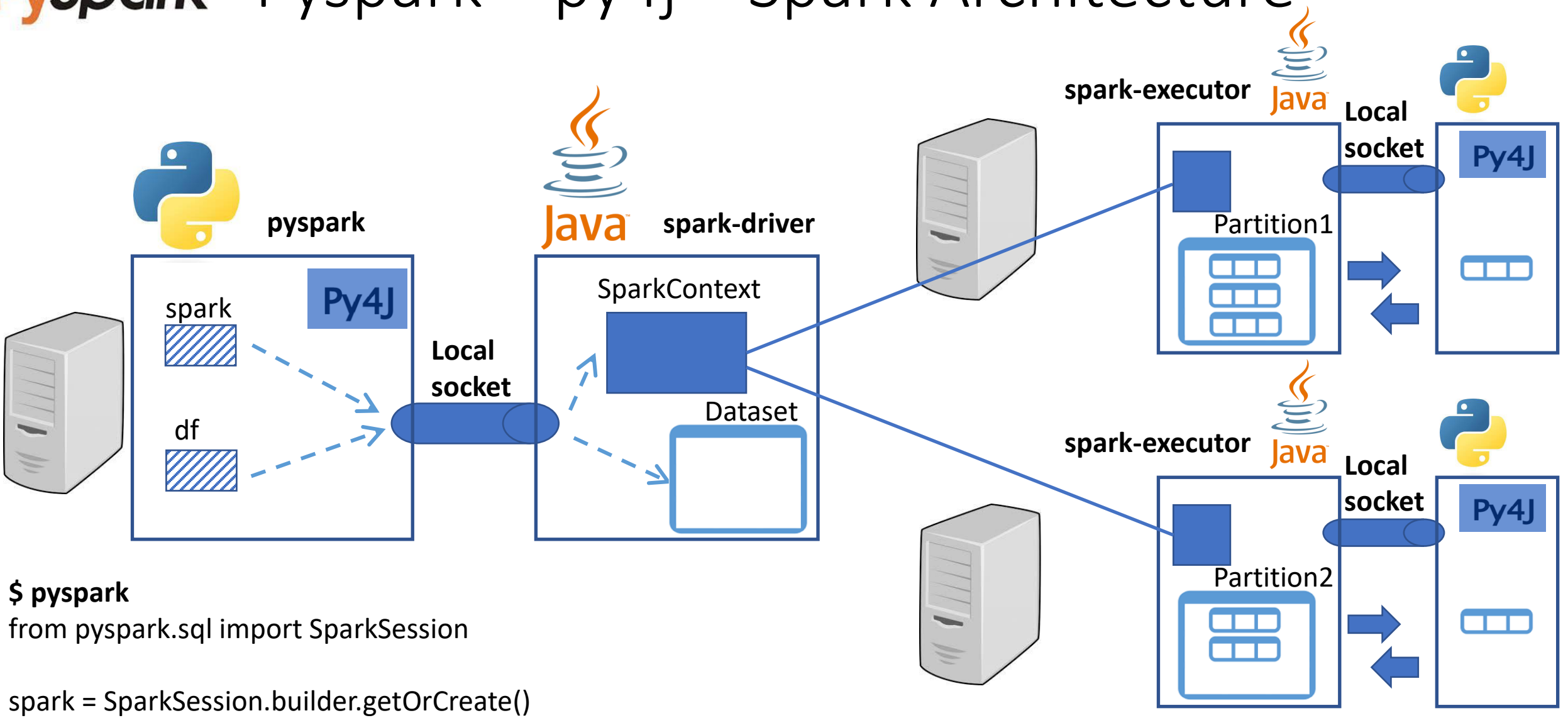
PySpark supports various UDFs and APIs to allow users to execute Python native functions. See also the latest Pandas UDFs and Pandas Function APIs. For instance, the example below allows users to directly use the APIs in a pandas Series within Python native function.

[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()
```

```
+------------------+
|pandas_plus_one(a)|
+------------------+
|                 2|
|                 3|
|                 4|
+------------------+
```



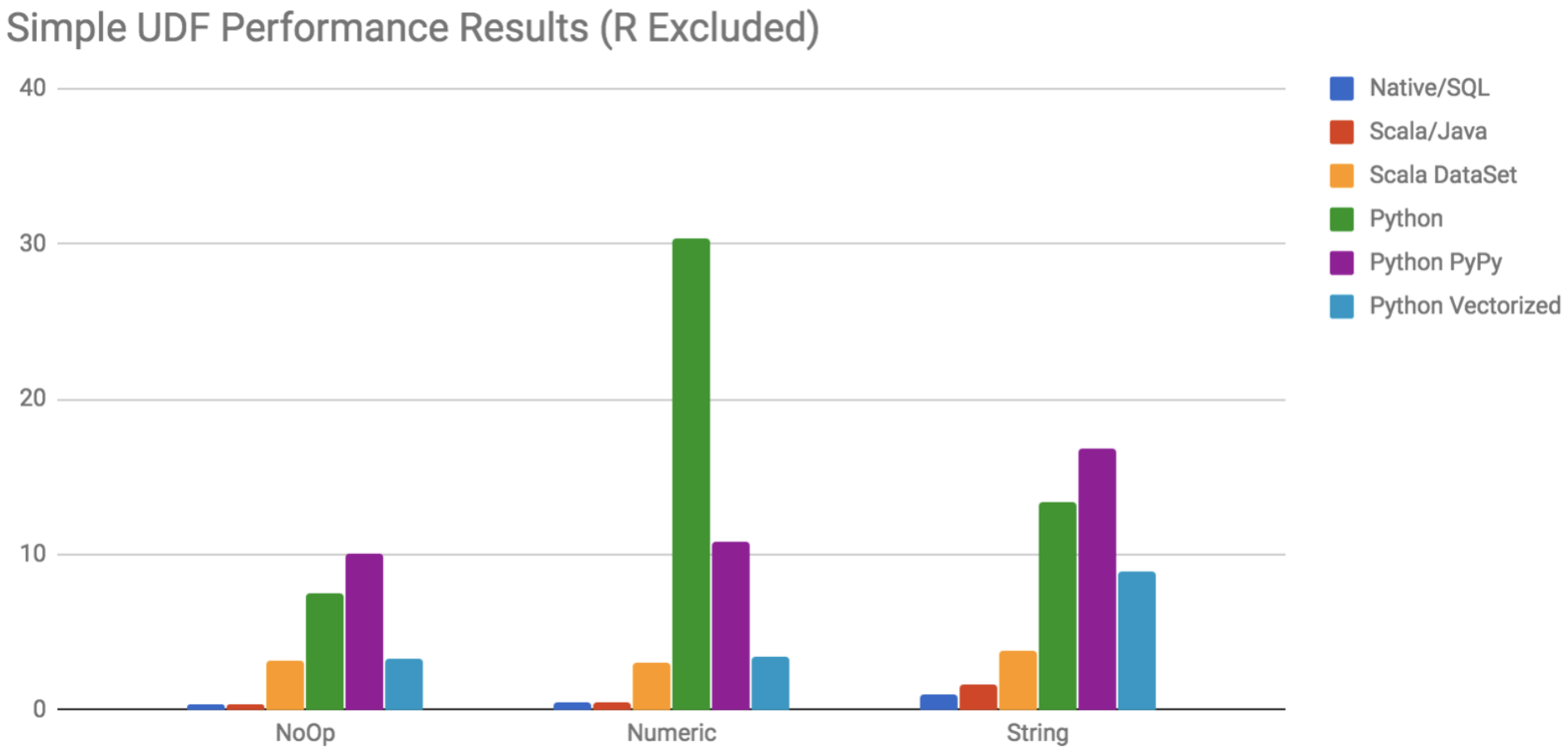
Pyspark – py4j – Spark Architecture



$ pyspark

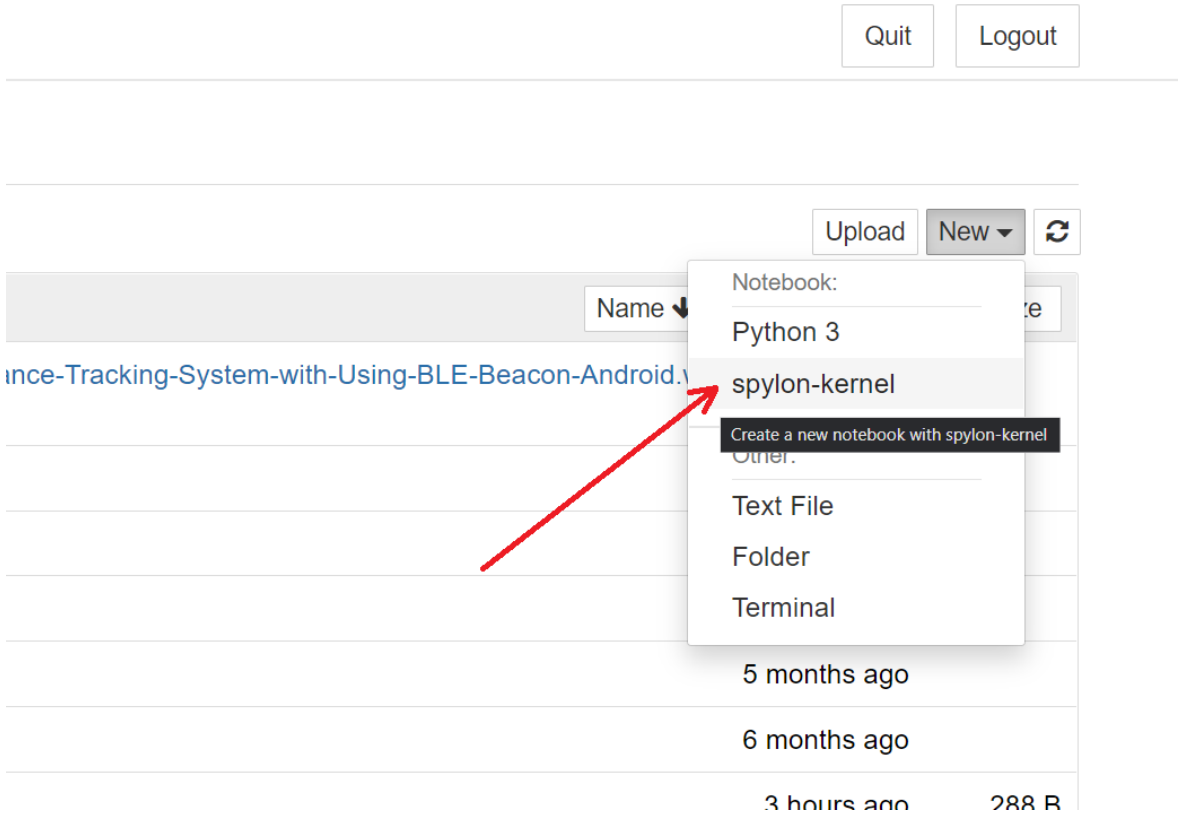
```
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()
df = spark.select(« SELECT * TABLE table1 »)
```

Python + Pickle = Performances Pourries

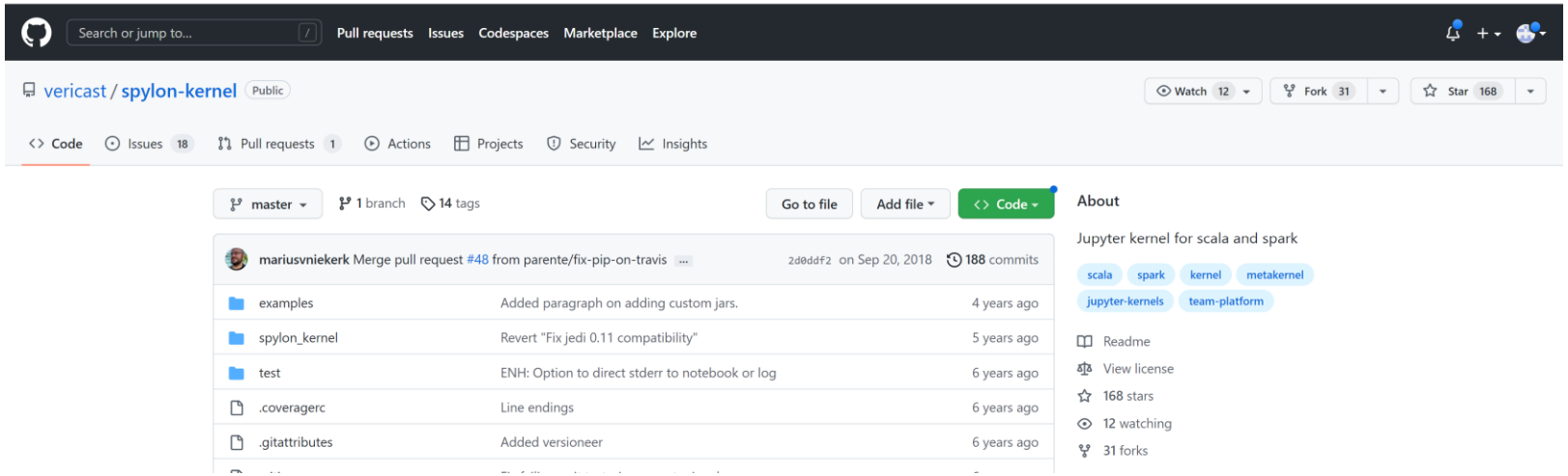


Jupyter + spylon (spark scala kernel)

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



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()
```

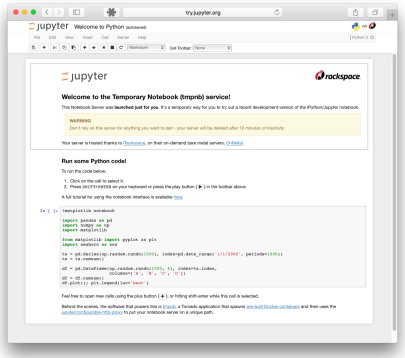
```
%%scala
val x = 8
x
```

Using it as a Library

Finally, you can use spylon-kernel as a Python library. Do
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()
```

Jupyter + spylon architecture



%%scala

Text
html

jupyter

%%scala

spylon-kernel

socket

spark-driver

SparkContext

Dataset

spark-executor

Partition1

spark-executor

Partition1

Enough for Today !

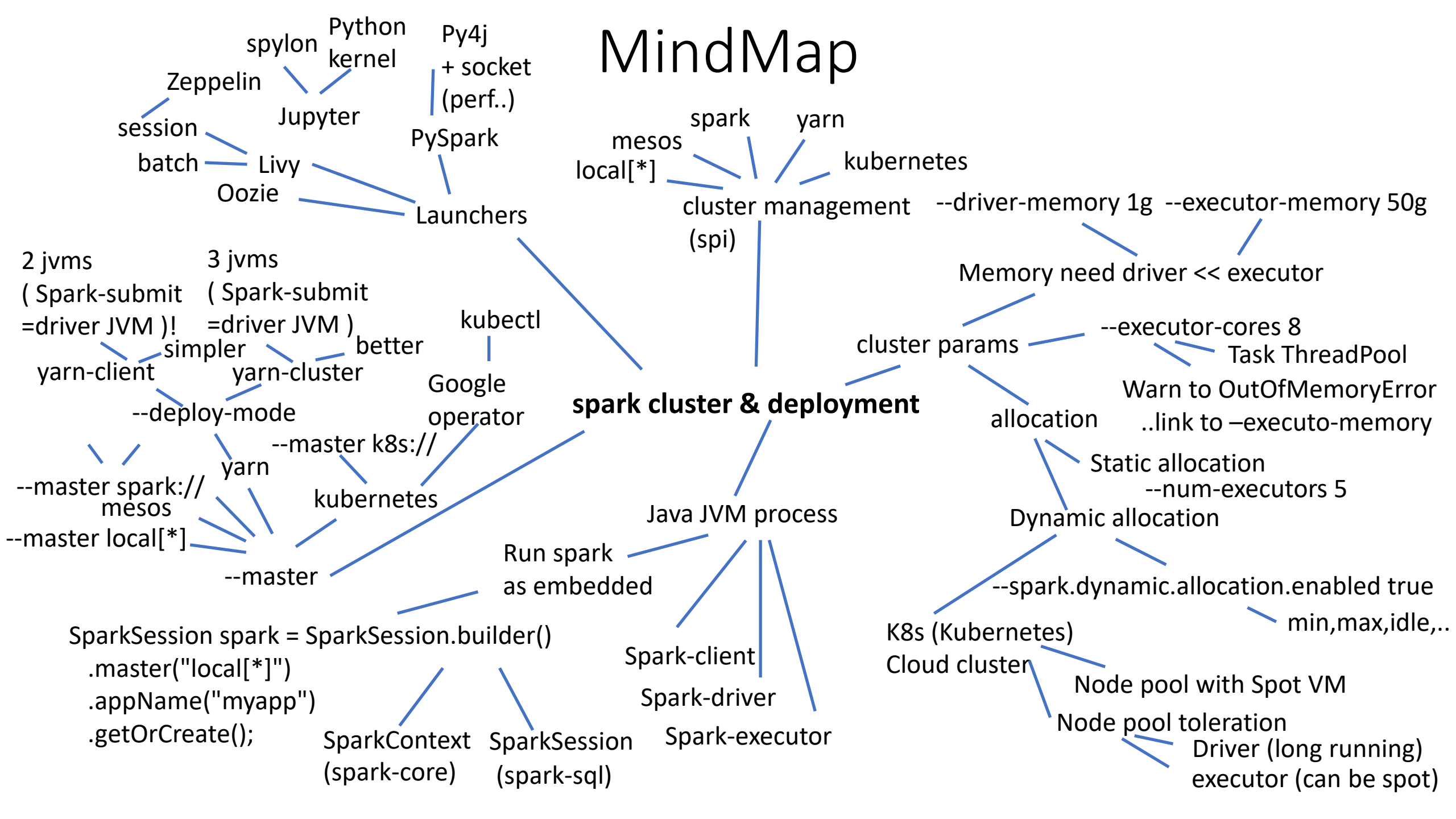
… but more to come

Questions are Welcome

Take Away

What Did you learn ?

MindMap


spylon
Python kernel
Py4j + socket (perf..)
Zeppelin
Jupyter
PySpark
session
batch
Livy
Oozie
Launchers
spark
yarn
mesos
local[*]
kubernetes
cluster management (spi)
--driver-memory 1g
--executor-memory 50g
Memory need driver << executor
2 jvms (Spark-submit =driver JVM)!
3 jvms (Spark-submit =driver JVM)
simpler
better
yarn-client
yarn-cluster
kubectl
Google operator
--deploy-mode
--master k8s://
yarn
--master spark://
mesos
--master local[*]
kubernetes
--master
spark cluster & deployment
cluster params
--executor-cores 8
Task ThreadPool
Warn to OutOfMemoryError
..link to –executo-memory
allocation
Static allocation
--num-executors 5
Dynamic allocation
--spark.dynamic.allocation.enabled true
min,max,idle,..
Java JVM process
Run spark as embedded
SparkSession spark = SparkSession.builder()
.master("local[*]")
.appName("myapp")
.getOrCreate();
SparkContext (spark-core)
SparkSession (spark-sql)
Spark-client
Spark-driver
Spark-executor
K8s (Kubernetes) Cloud cluster
Node pool with Spot VM
Node pool toleration
Driver (long running)
executor (can be spot)

Next Steps

Lesson 1 : introduction to BigData / Hadoop cluster / Spark

Lesson 2 : this document : spark-core / spark-sql

Lesson 3 : cluster management, tuning args, deploy-mode

Lesson 4 : Advanced Spark: UI, parquet PPD, java api, spark-streaming, ..

Questions ?

arnaud.nauwynck@gmail.com