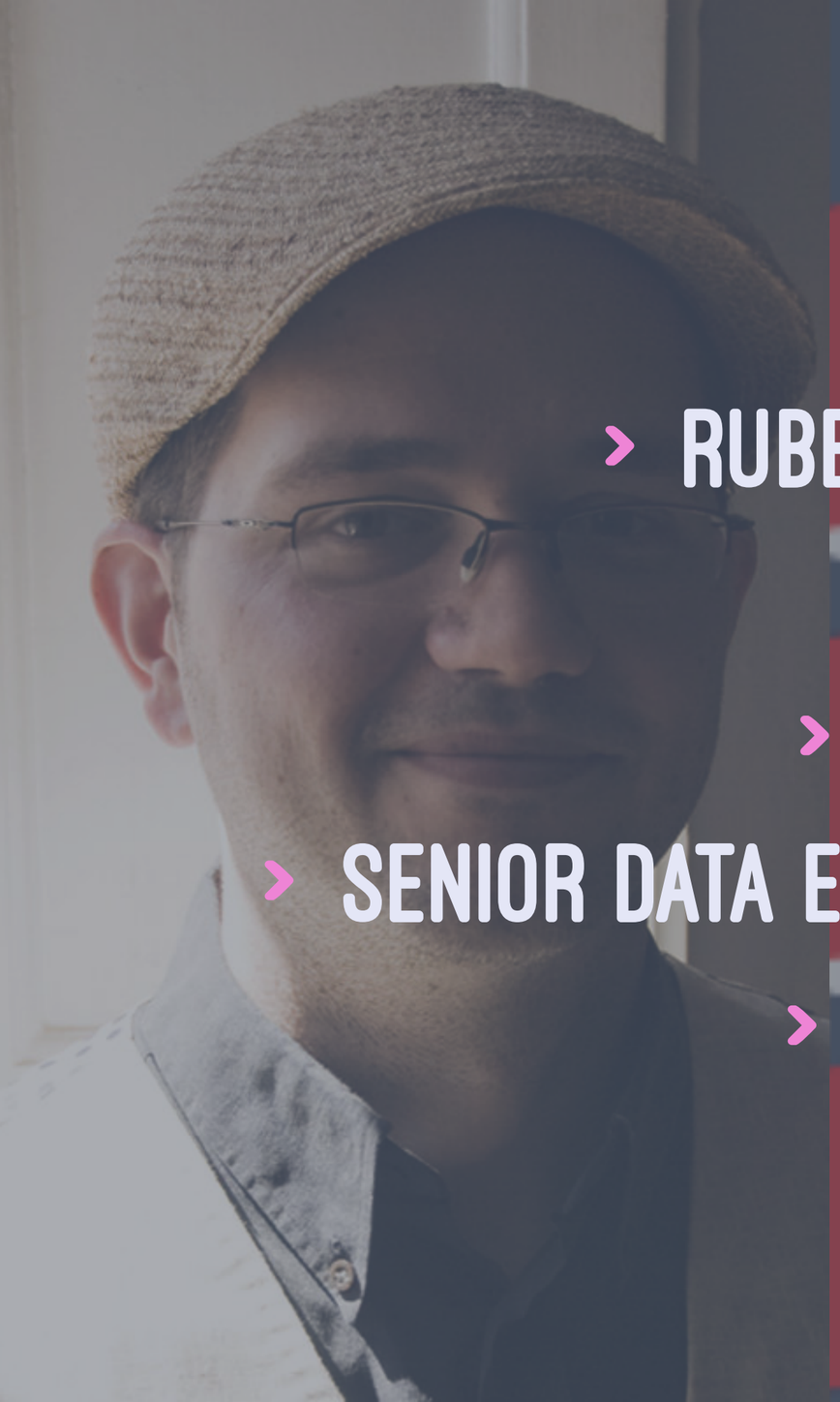
HOW DOES THAT PYSPARK THING WORK? AND WHY ARROW MAKES IT FASTER?



WHOAMI

- RUBEN BERENGUEL (@BERENGUEL)
- PHD IN MATHEMATICS
- (BIG) DATA CONSULTANT
- SENIOR DATA ENGINEER USING PYTHON, GO AND SCALA
- RIGHT NOW AT AFFECTV

WHAT IS PANDAS?

WHAT IS PANDAS?

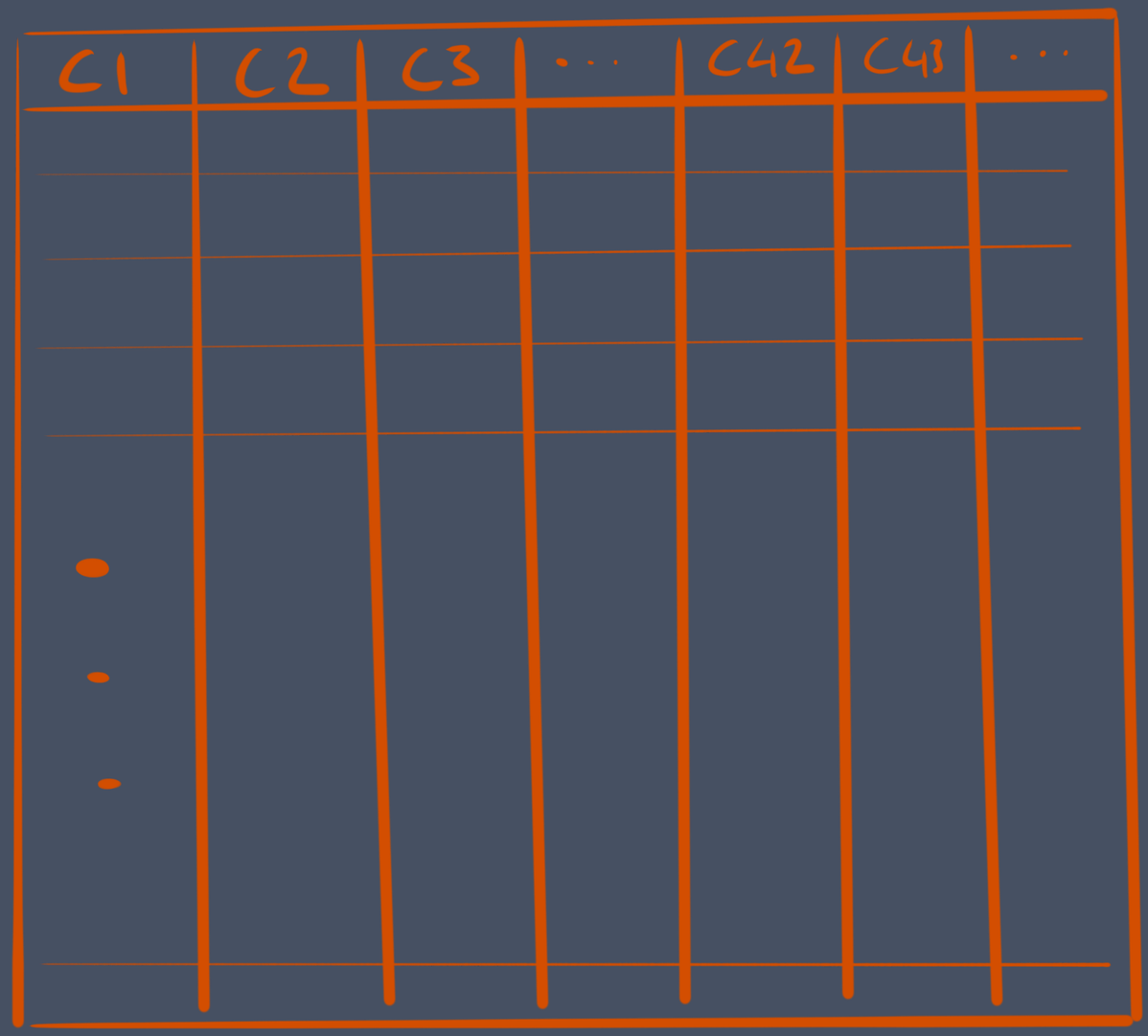
- PYTHON DATA ANALYSIS LIBRARY

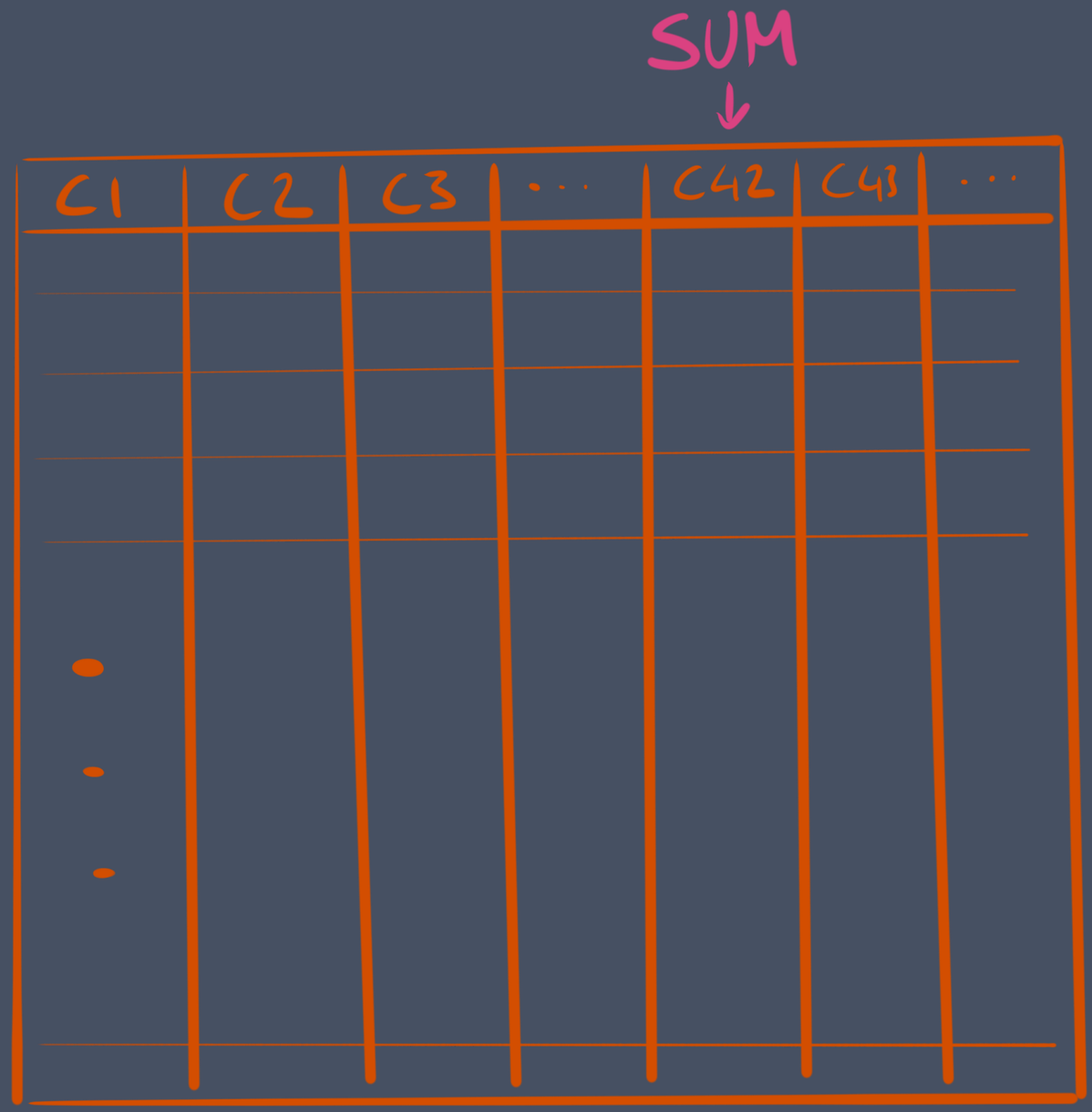
WHAT IS PANDAS?

- PYTHON DATA ANALYSIS LIBRARY
- USED EVERYWHERE DATA AND PYTHON APPEAR IN JOB OFFERS

WHAT IS PANDAS?

- PYTHON DATA ANALYSIS LIBRARY
- USED EVERYWHERE DATA AND PYTHON APPEAR IN JOB OFFERS
- EFFICIENT (IS COLUMNAR AND HAS A C AND CYTHON BACKEND)

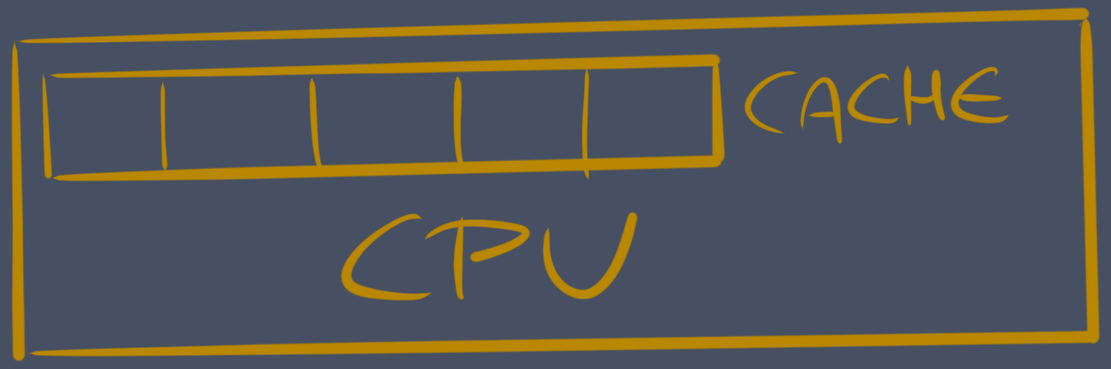

C1
C2
C3
...
C42
C43
...


SUM
C1
C2
C3
...
C42
C43
...

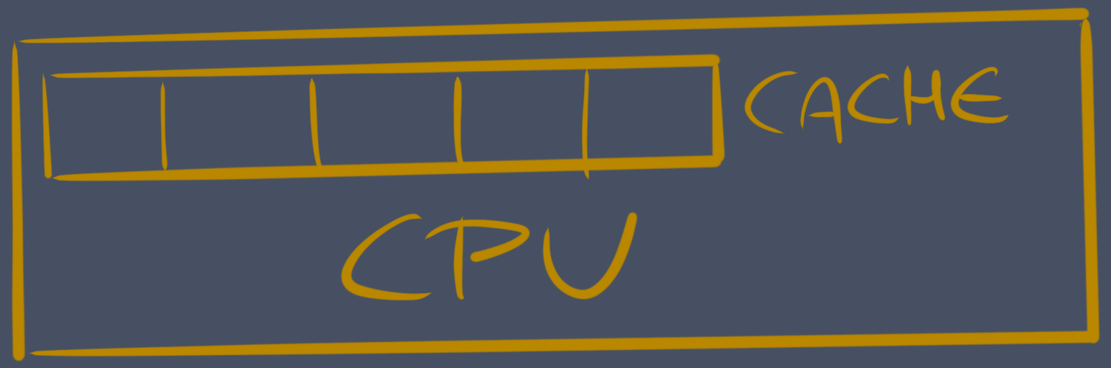

ROW

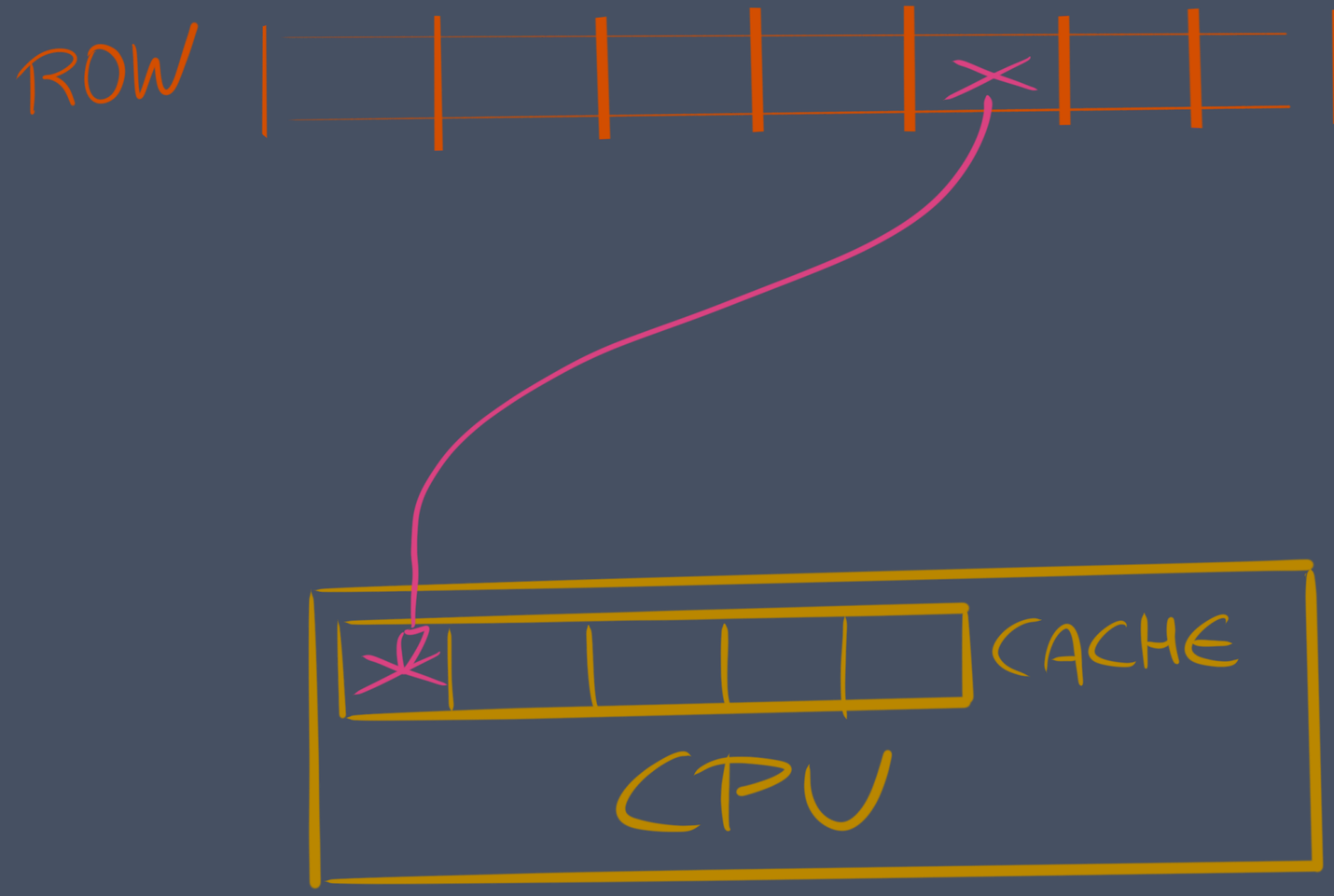

CPU

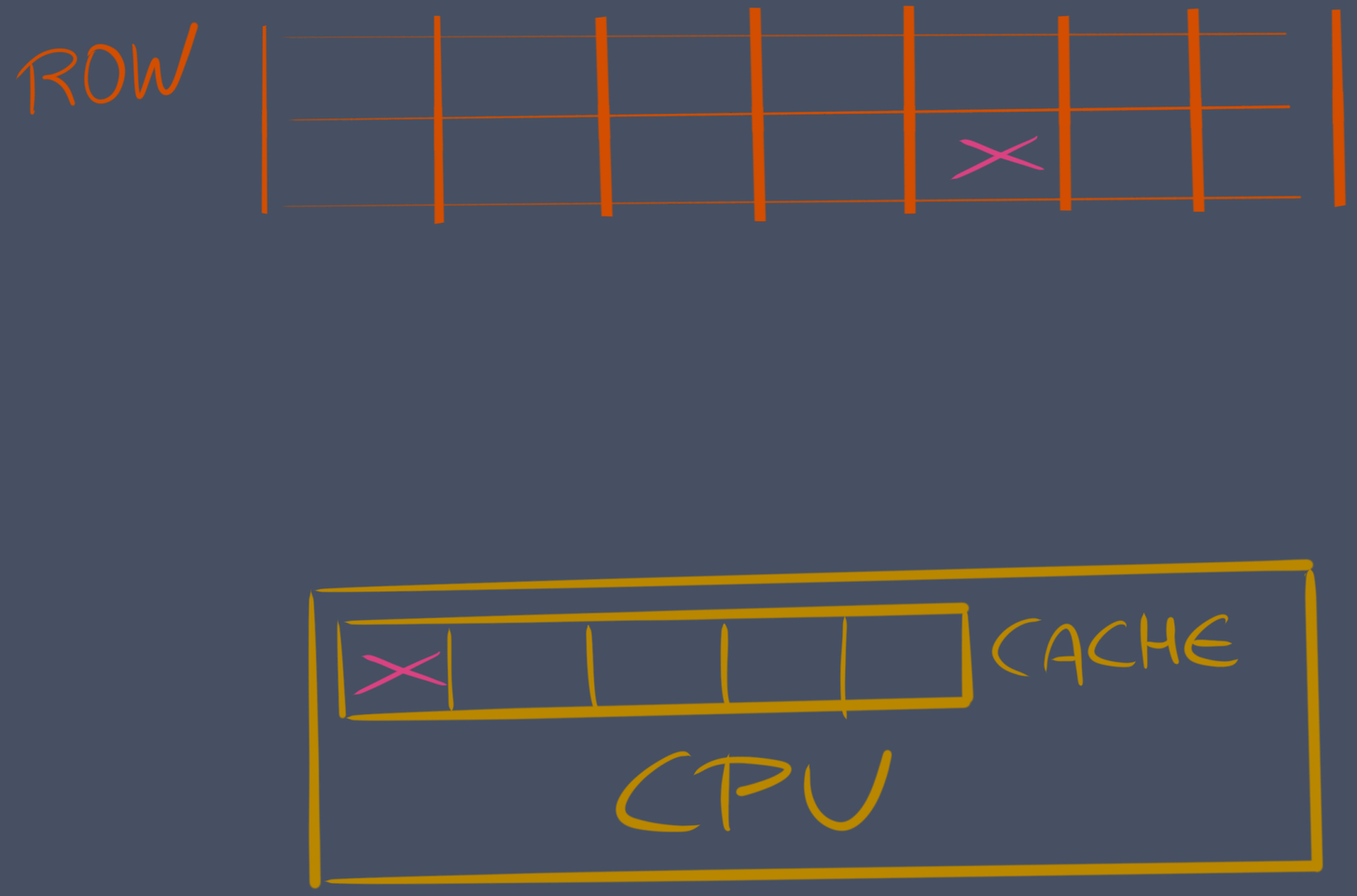

ROW

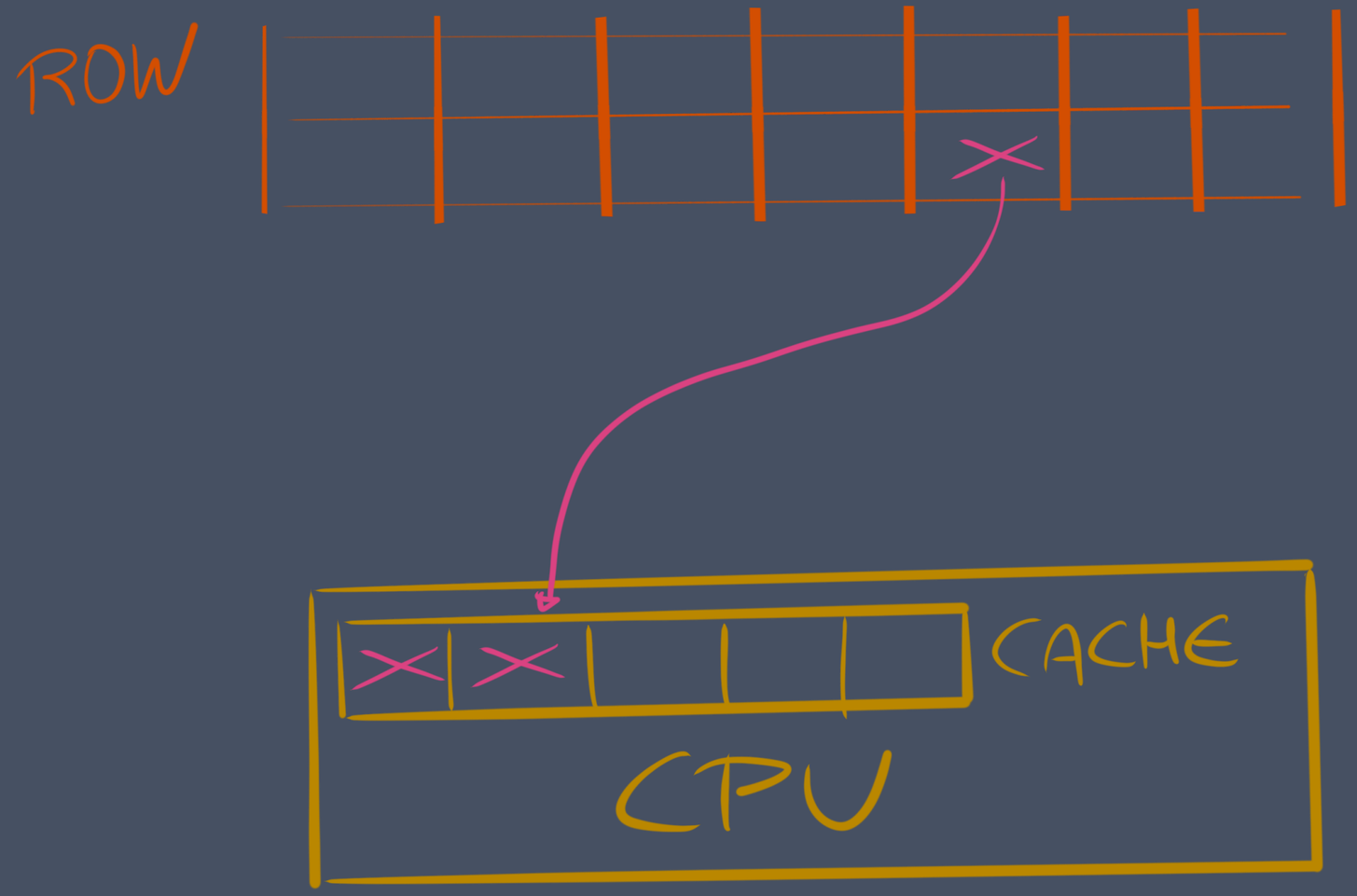

CACHE
CPU

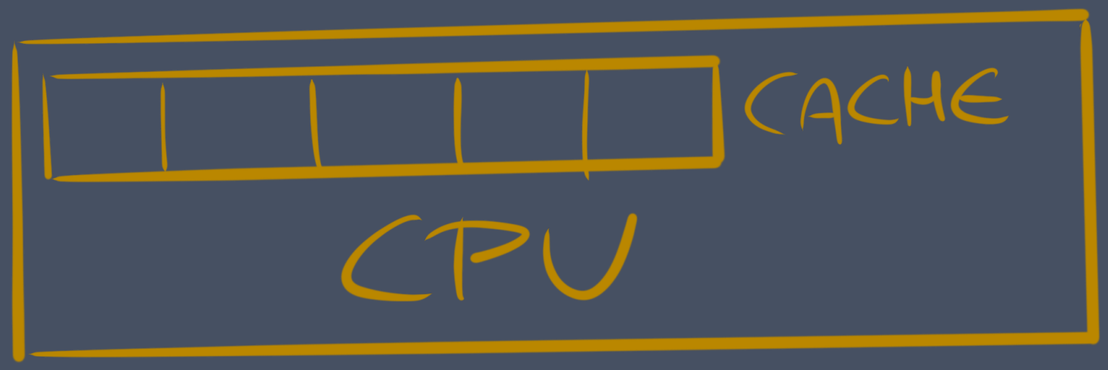

ROW

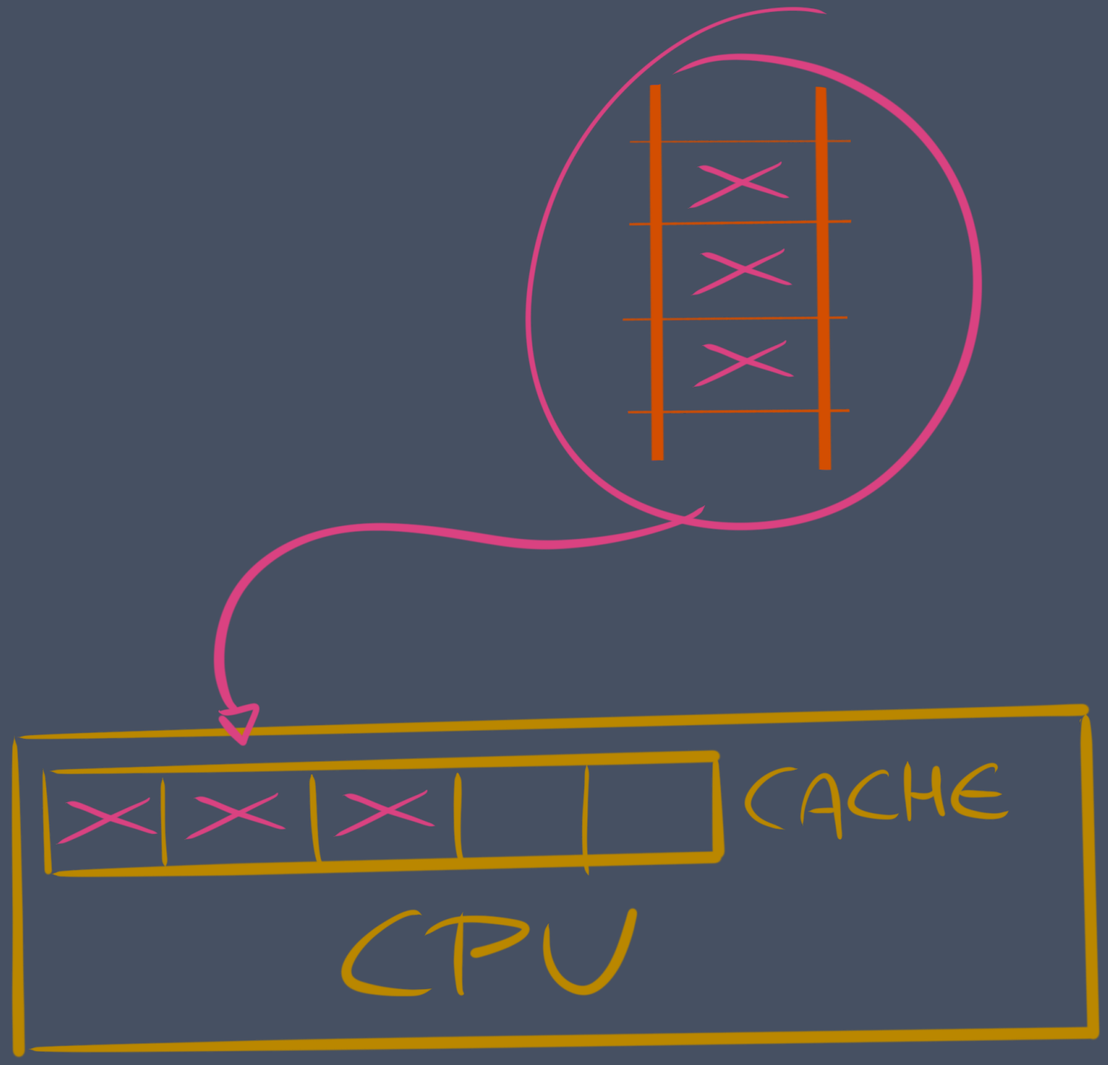

CACHE
CPU


ROW
CACHE
CPU


ROW
CACHE
CPU


ROW
CACHE
CPU


CACHE
CPU


CACHE
CPU

WHAT IS ARROW?

WHAT IS ARROW?

- CROSS-LANGUAGE IN-MEMORY COLUMNAR FORMAT LIBRARY

WHAT IS ARROW?

- CROSS-LANGUAGE IN-MEMORY COLUMNAR FORMAT LIBRARY
  - OPTIMISED FOR EFFICIENCY ACROSS LANGUAGES

WHAT IS ARROW?

- CROSS-LANGUAGE IN-MEMORY COLUMNAR FORMAT LIBRARY
  - OPTIMISED FOR EFFICIENCY ACROSS LANGUAGES
    - INTEGRATES SEAMLESSLY WITH PANDAS

- ARROW USES `RecordBatches`

- ARROW USES `RecordBatches`
- PANDAS USES BLOCKS HANDLED BY A `BlockManager`

- ARROW USES `RecordBatches`
- PANDAS USES BLOCKS HANDLED BY A `BlockManager`
- YOU CAN CONVERT AN ARROW `Table` INTO A PANDAS `DataFrame` EASILY

WHAT IS SPARK?

WHAT IS SPARK?

- **DISTRIBUTED COMPUTATION** FRAMEWORK

WHAT IS SPARK?

- DISTRIBUTED COMPUTATION FRAMEWORK
- OPEN SOURCE

WHAT IS SPARK?

- DISTRIBUTED COMPUTATION FRAMEWORK
  - OPEN SOURCE
  - EASY TO USE

WHAT IS SPARK?

- DISTRIBUTED COMPUTATION FRAMEWORK
  - OPEN SOURCE
  - EASY TO USE
- SCALES HORIZONTALLY AND VERTICALLY

HOW DOES SPARK WORK?

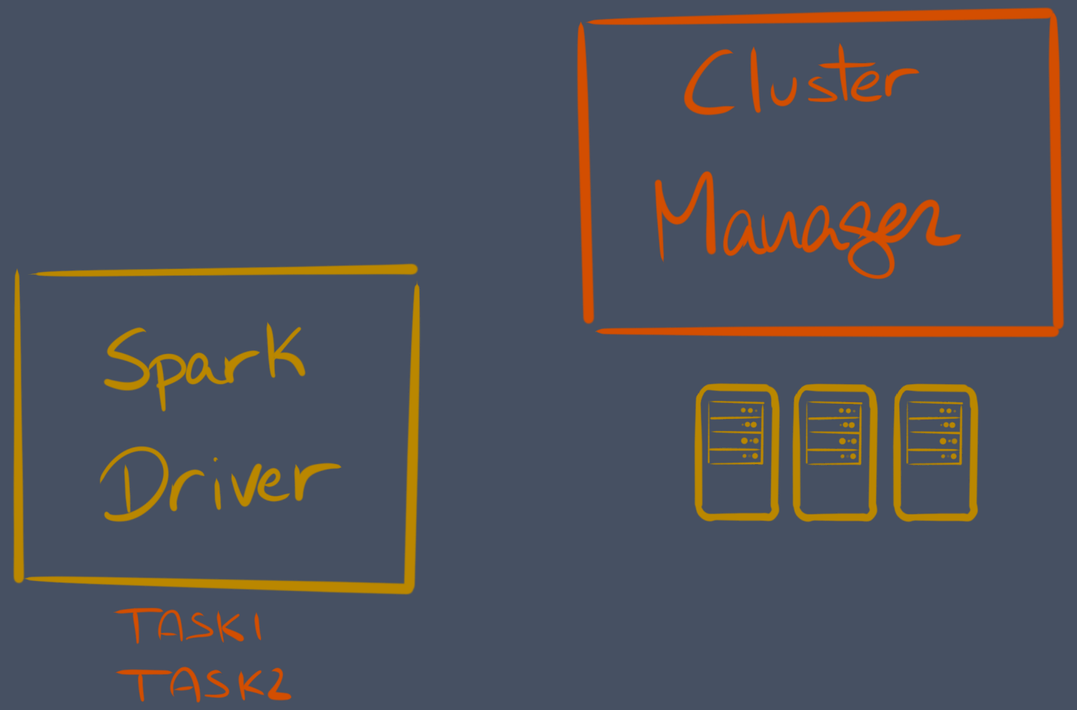
SPARK USUALLY SITS ON TOP OF A CLUSTER MANAGER

AND A
DISTRIBUTED
STORAGE

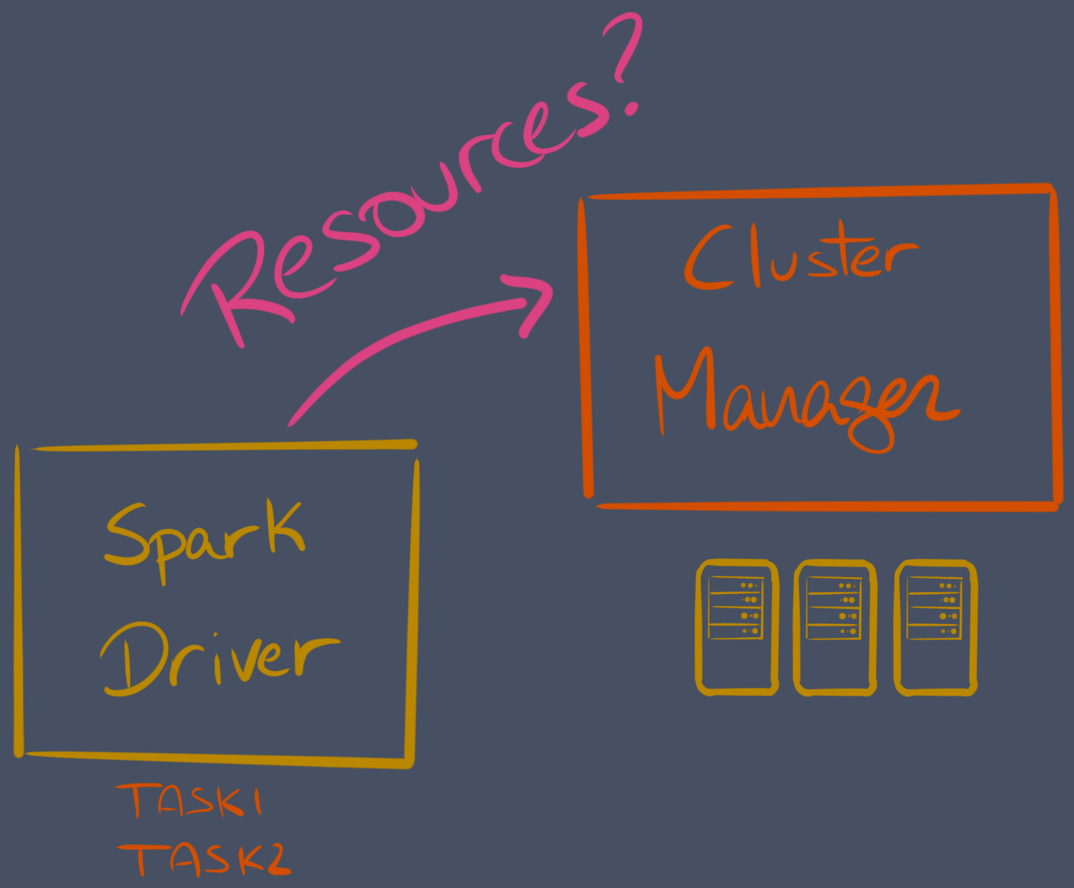
A SPARK PROGRAM
RUNS IN THE DRIVER

THE DRIVER REQUESTS RESOURCES FROM THE CLUSTER MANAGER TO RUN TASKS

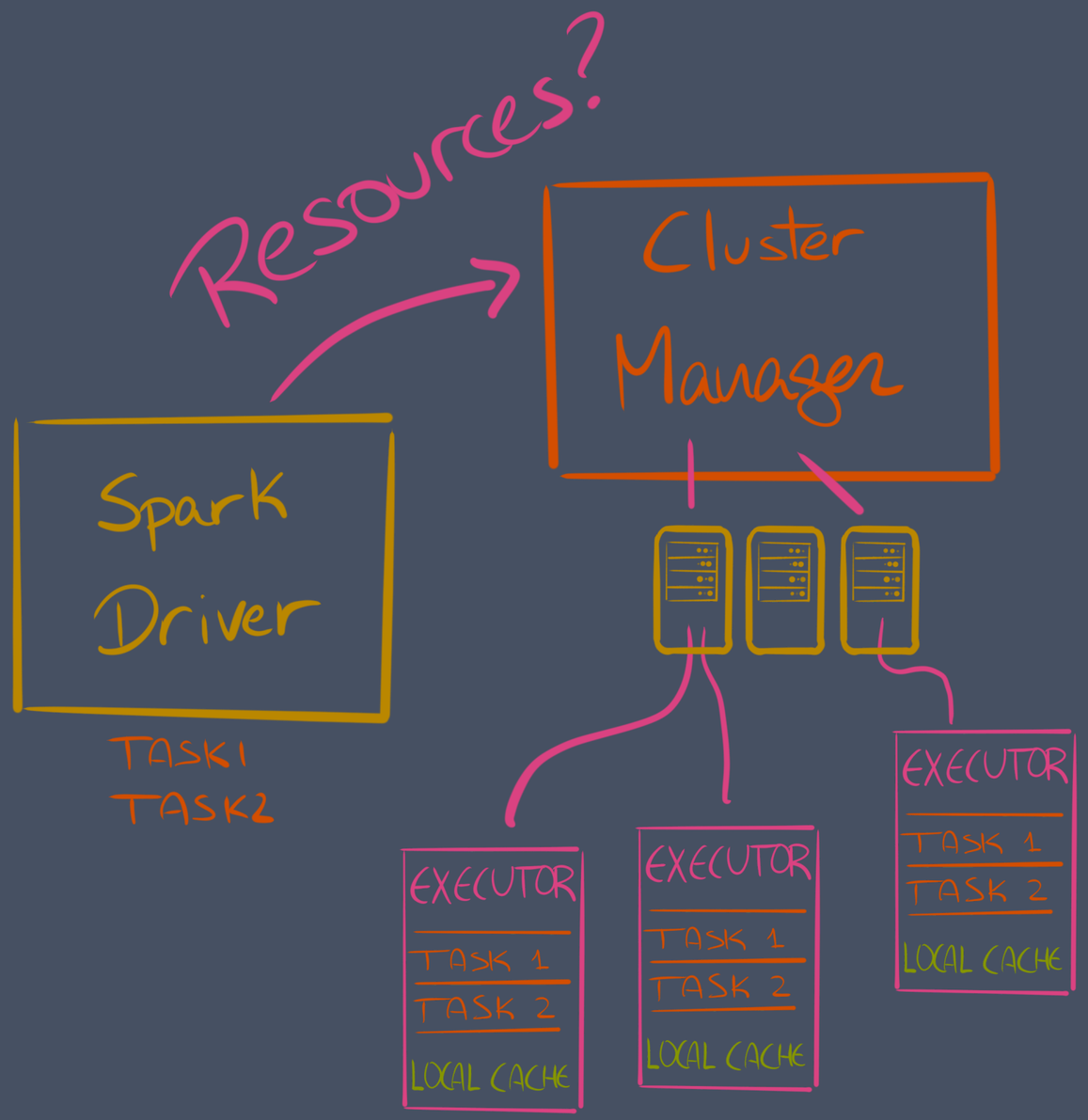
THE DRIVER REQUESTS RESOURCES FROM THE CLUSTER MANAGER TO RUN TASKS



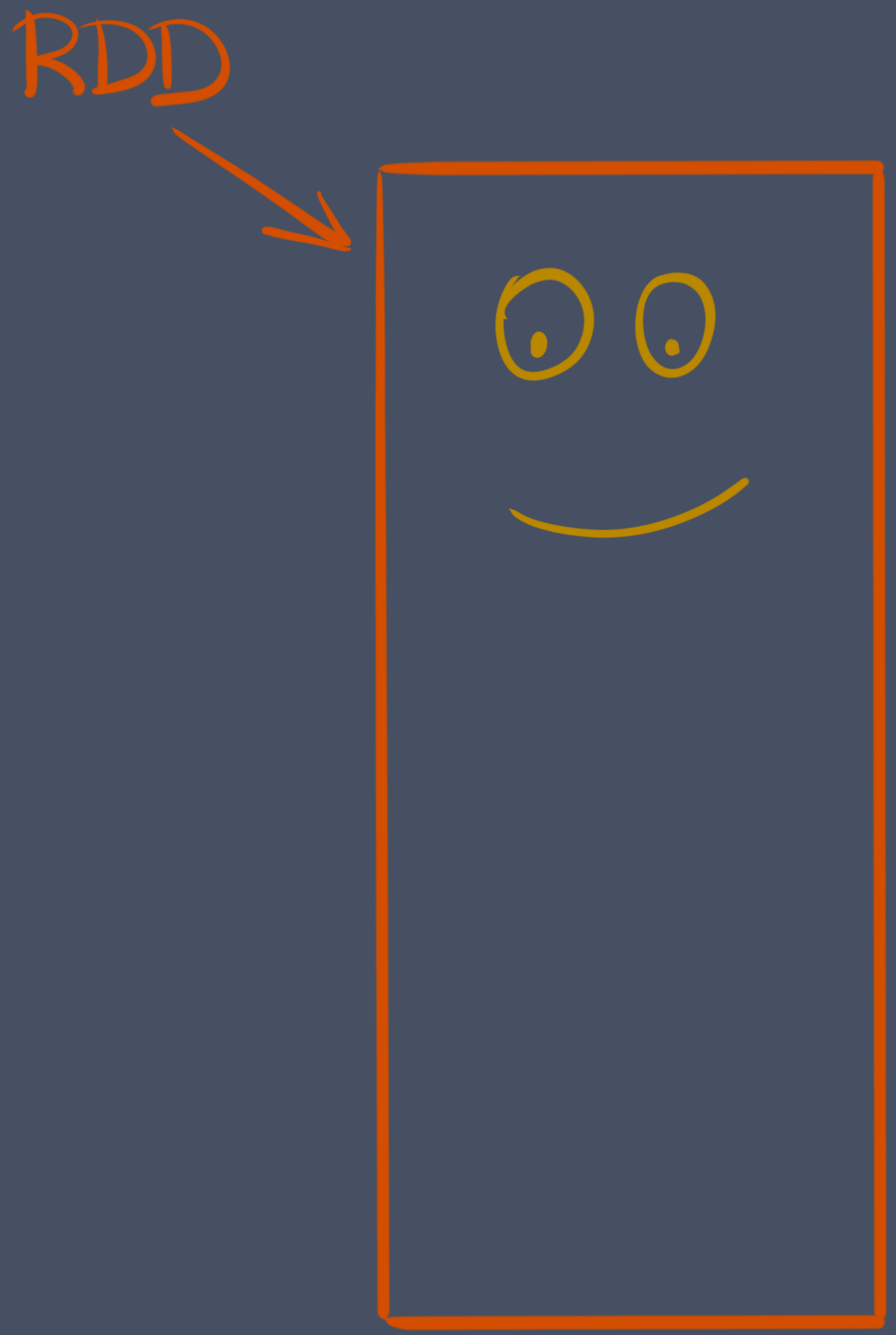
THE DRIVER REQUESTS RESOURCES FROM THE CLUSTER MANAGER TO RUN TASKS

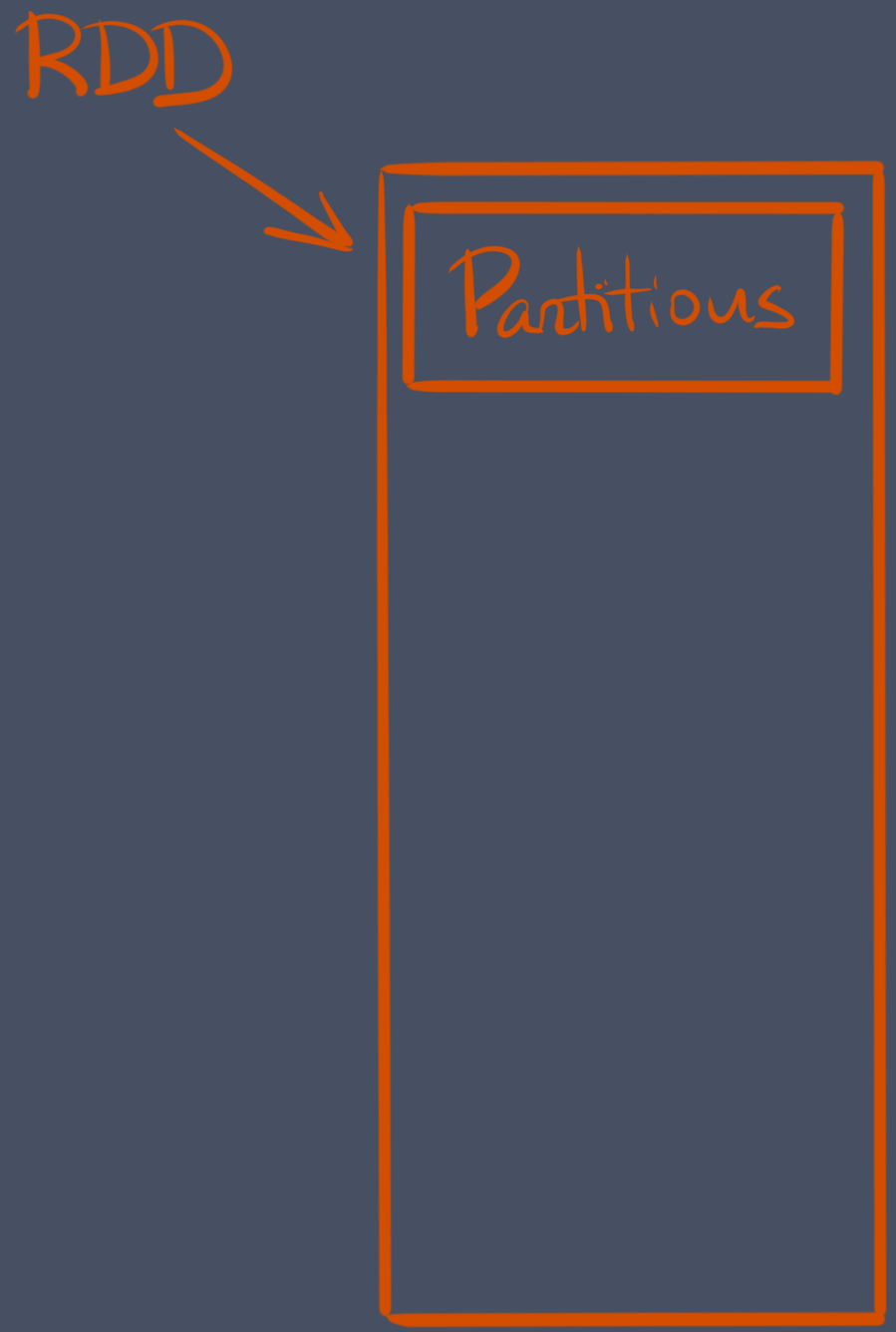


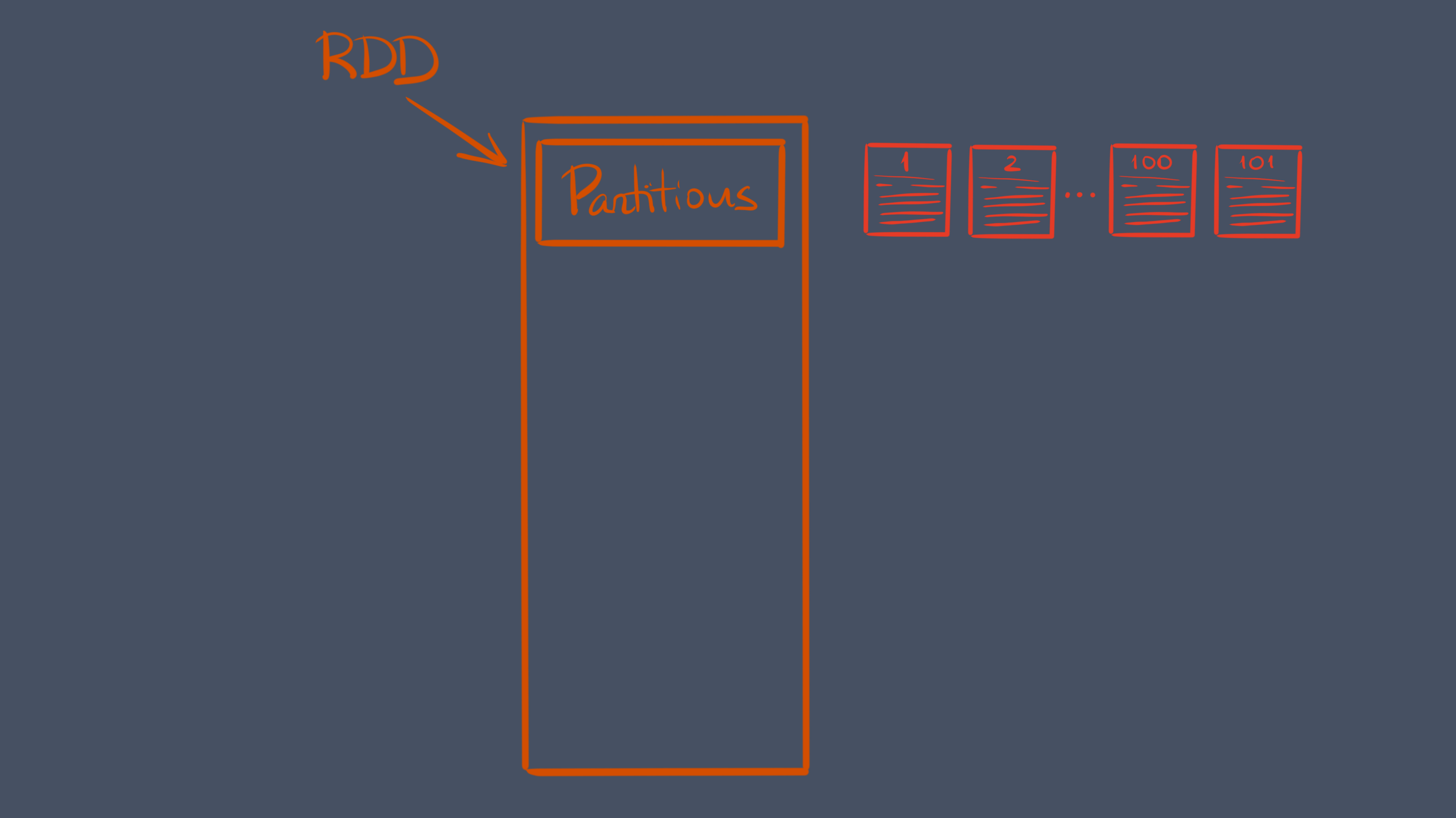
THE DRIVER REQUESTS RESOURCES FROM THE CLUSTER MANAGER TO RUN TASKS

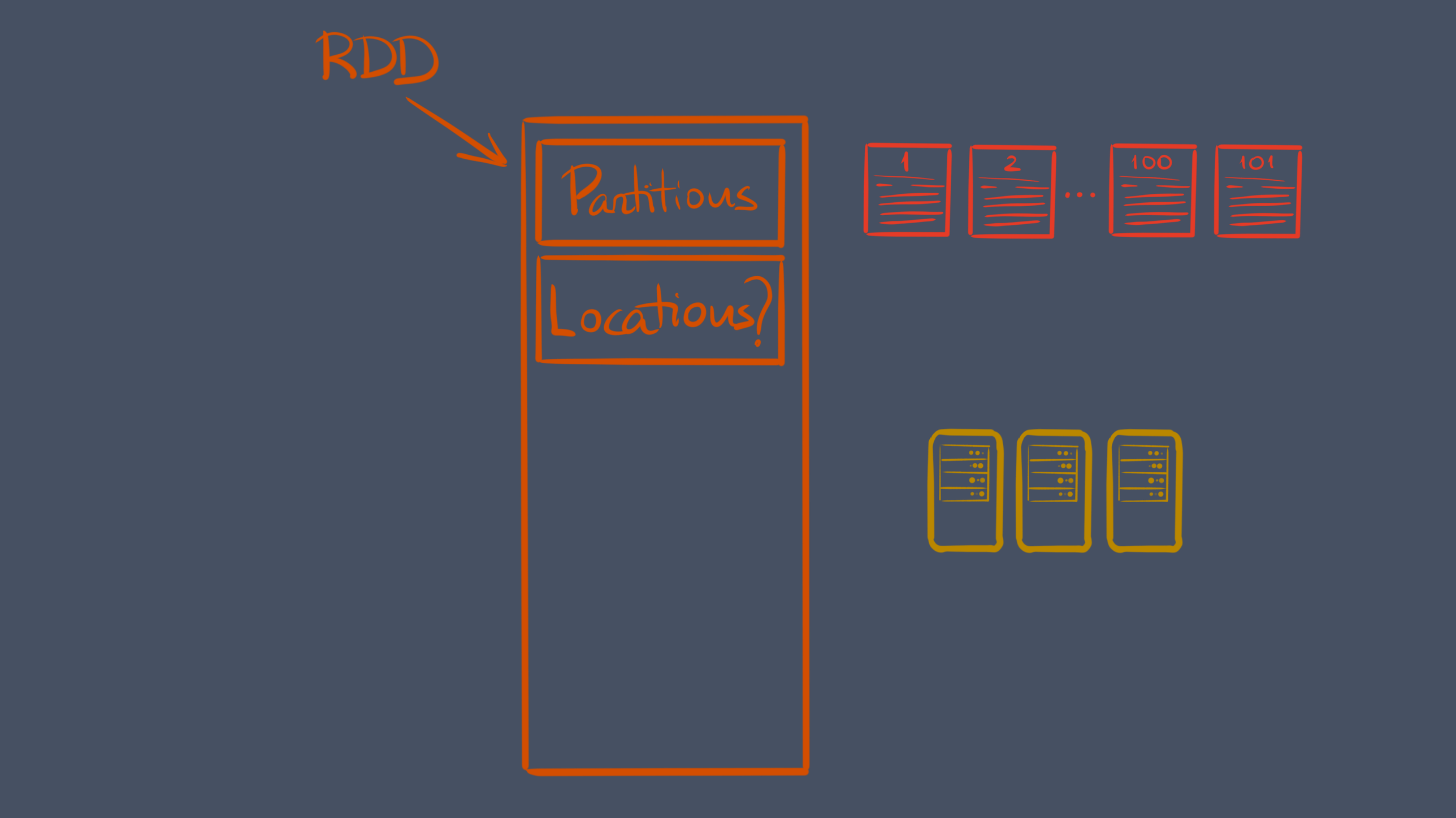


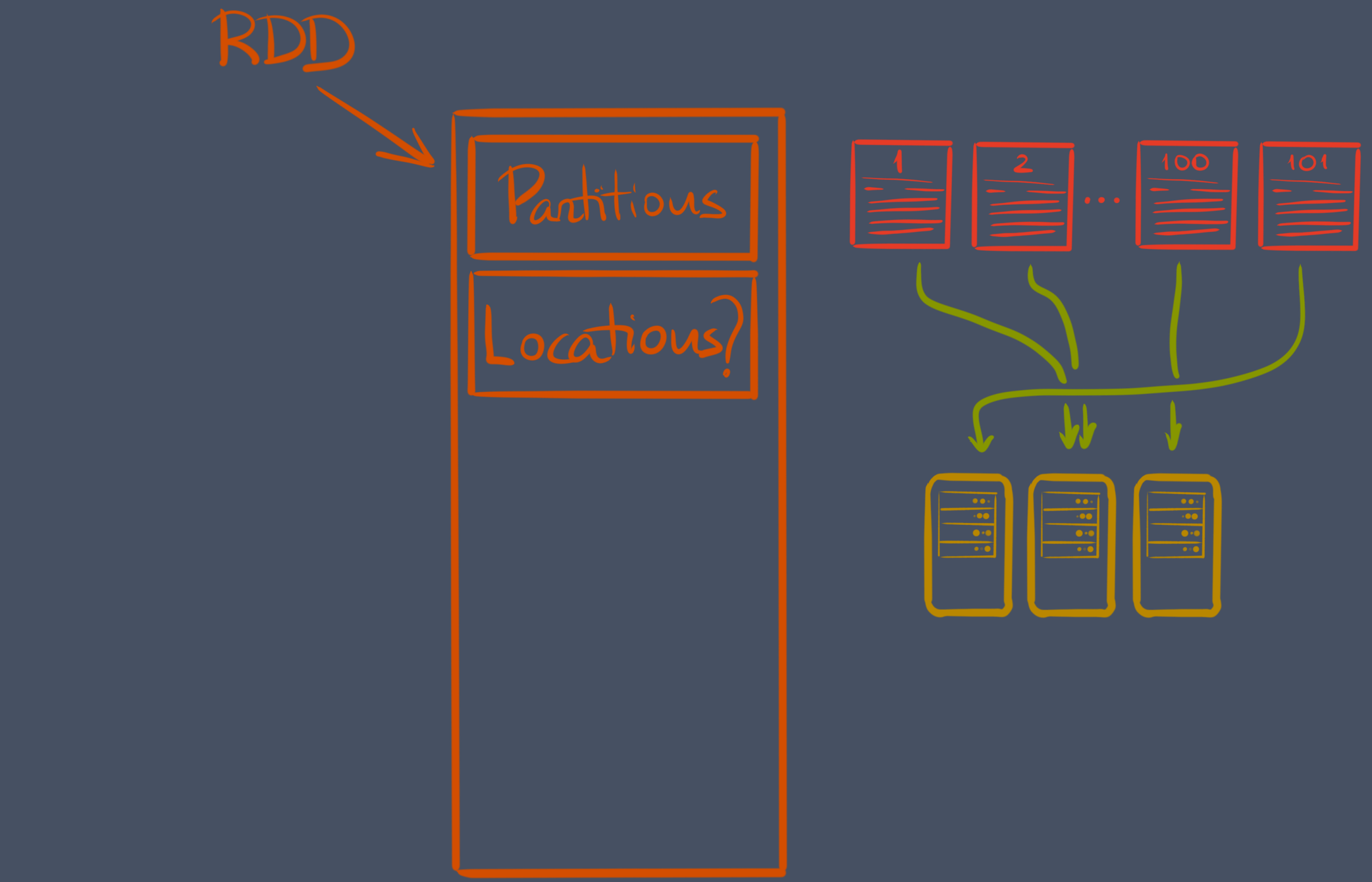
THE MAIN BUILDING BLOCK IS THE RDD: RESILIENT DISTRIBUTED DATASET

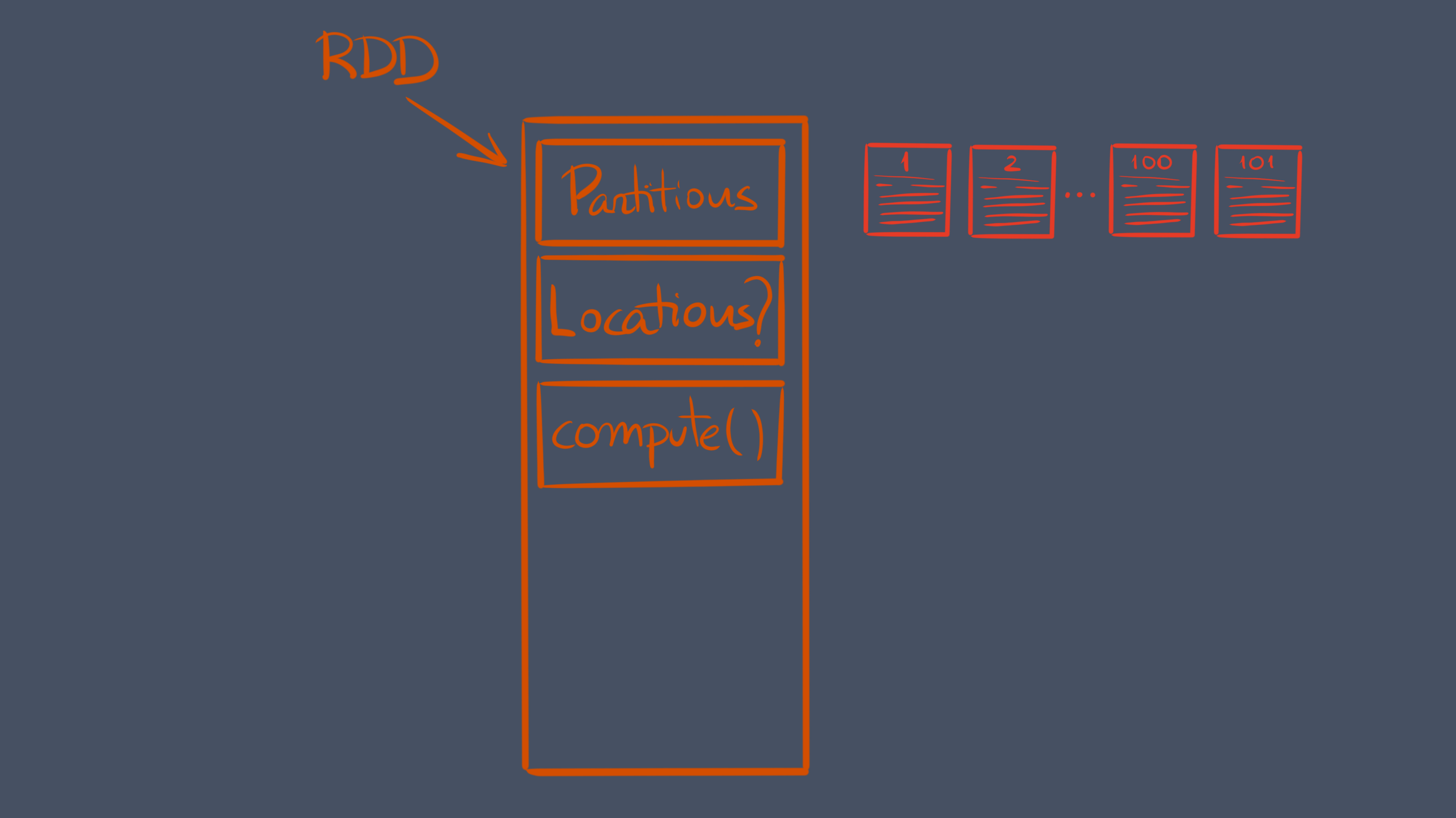

RDD

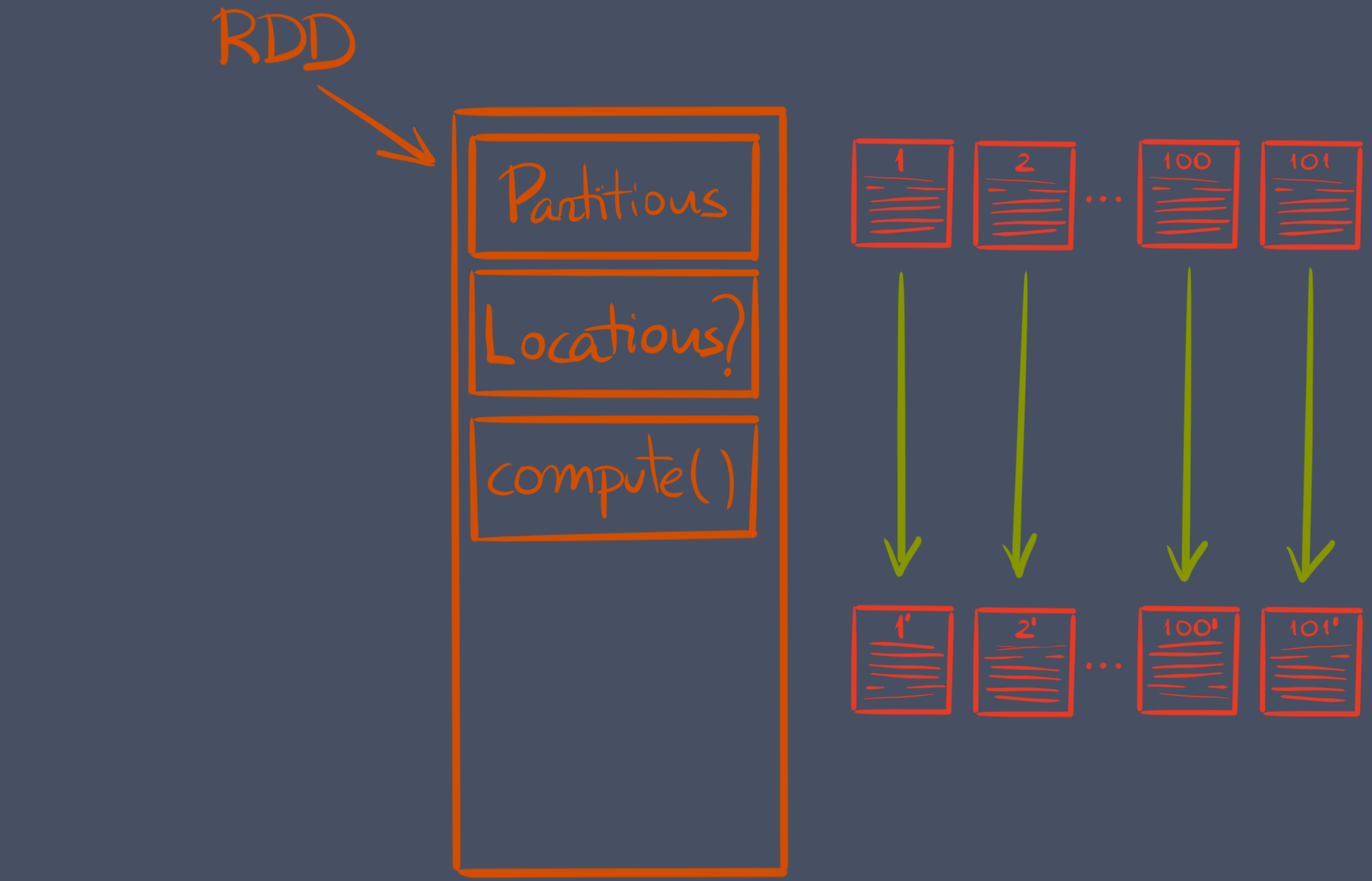

RDD
Partitions

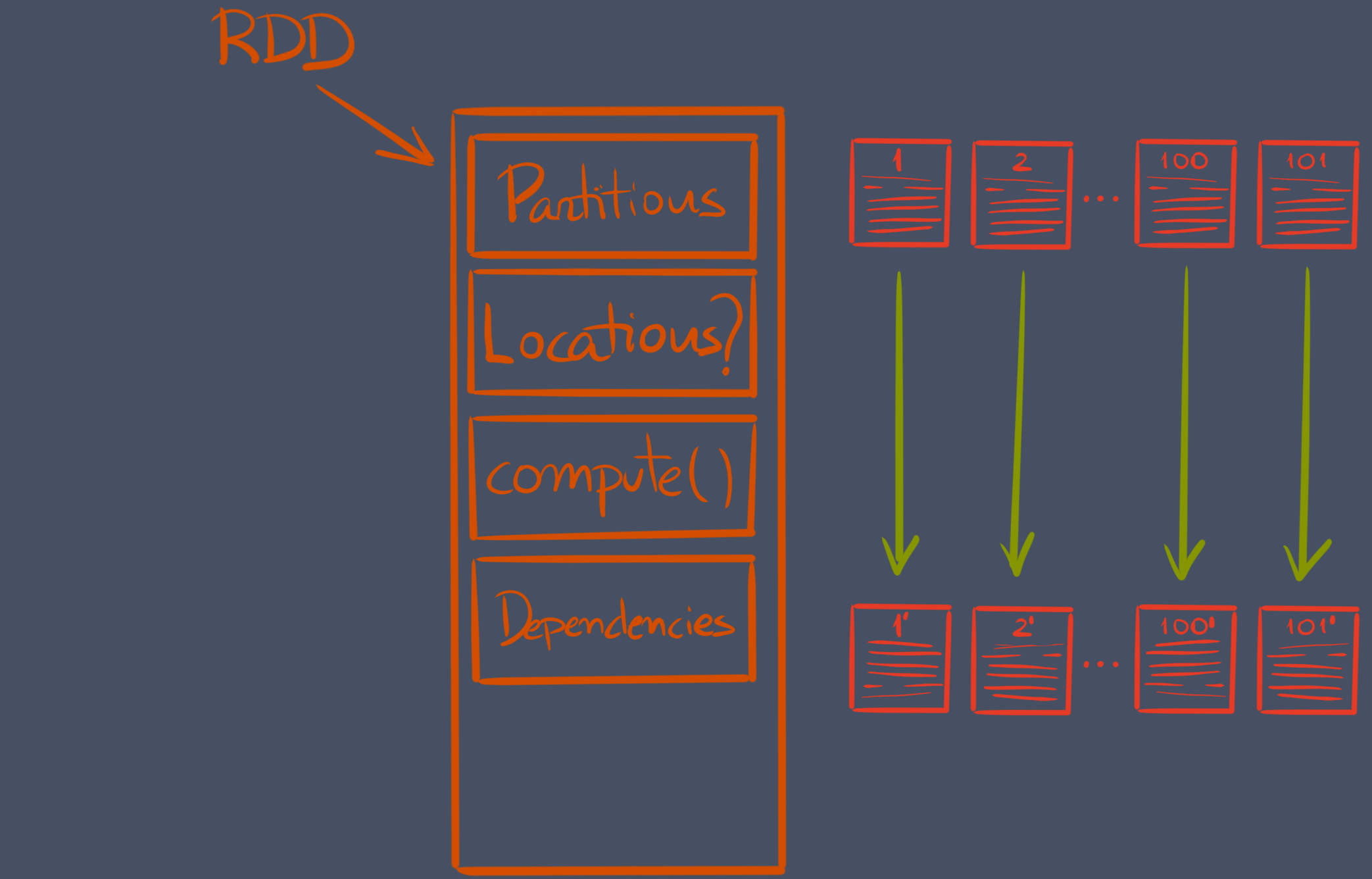

RDD
Partitions
1
2
100
101

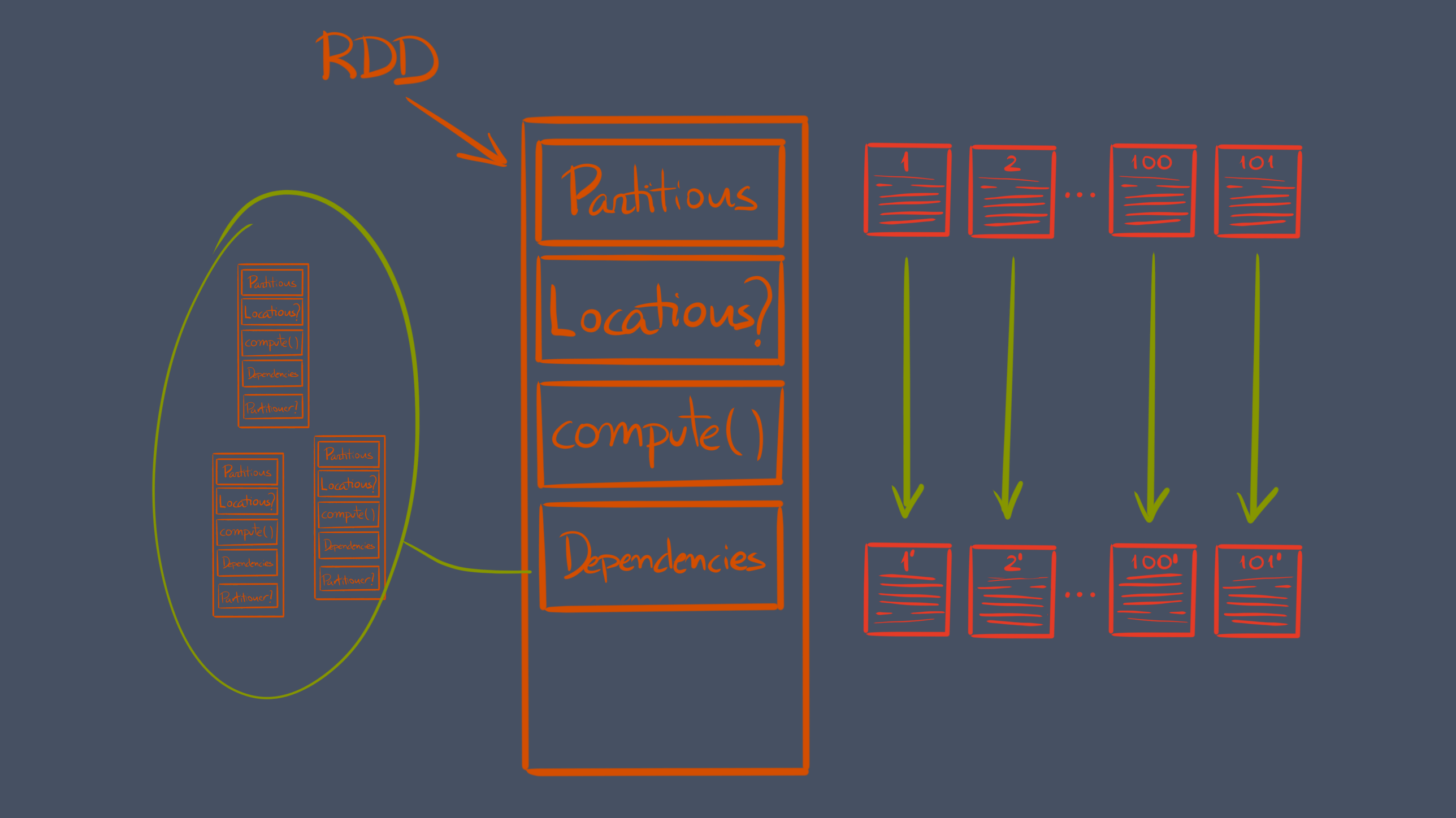

RDD
Partitions
Locations?
1
2
100
101

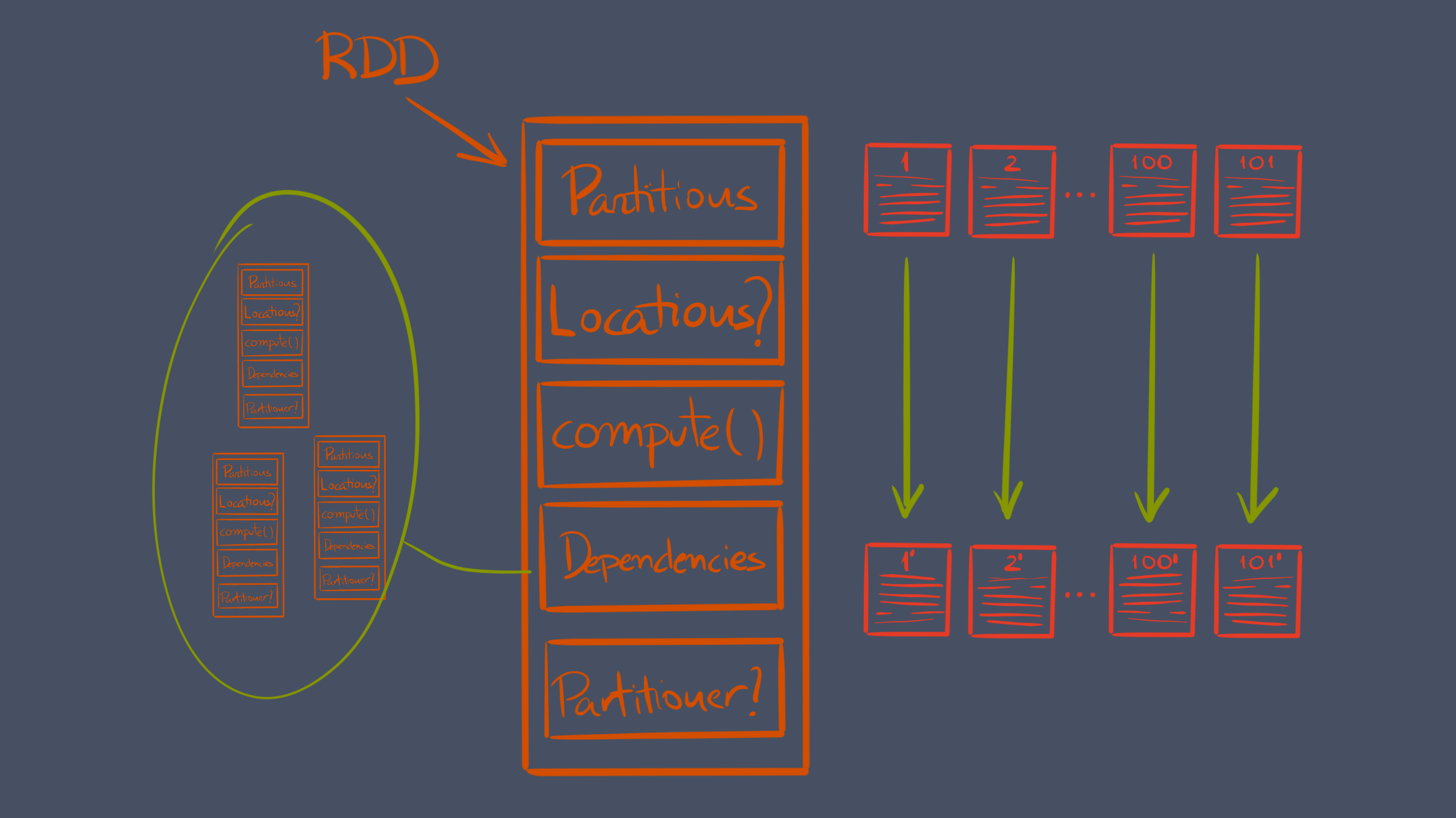

RDD
Partitions
Locations?
1
2
100
101


RDD
Partitions
Locations?
compute()
1
2
100
101


RDD
Partitions
Locations?
compute()
1
2
100
101
1'
2'
100'
101'


RDD
Partitions
Locations?
compute()
Dependencies
1
2
100
101
1'
2'
100'
101'


RDD
Partitions
Locations?
compute()
Dependencies
Partitioner?
1
2
100
101
1'
2'
100'
101'


RDD
Partitions
Locations?
compute()
Dependencies
Partitioner?
1
2
100
101
1'
2'
100'
101'

PYSPARK

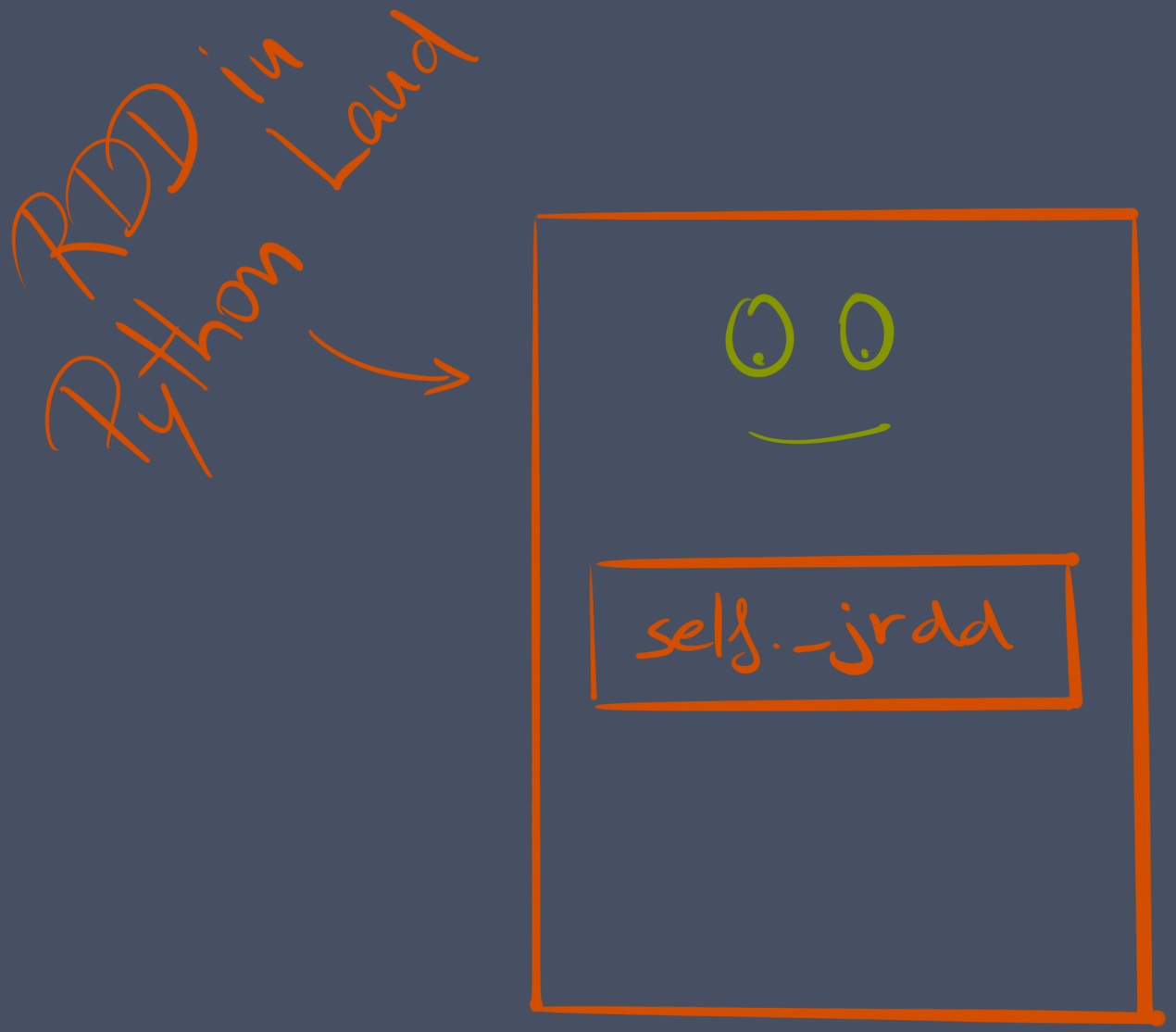
PYSPARK OFFERS A PYTHON API TO THE SCALA CORE OF SPARK

IT USES THE PY4J BRIDGE

```
# Connect to the gateway
gateway = JavaGateway(
    gateway_parameters=GatewayParameters(
        port=gateway_port,
        auth_token=gateway_secret,
        auto_convert=True))

# Import the classes used by PySpark
java_import(gateway.jvm, "org.apache.spark.SparkConf")
java_import(gateway.jvm, "org.apache.spark.api.java.*")
java_import(gateway.jvm, "org.apache.spark.api.python.*")
.
.
.
return gateway
```

RDD in Python Land

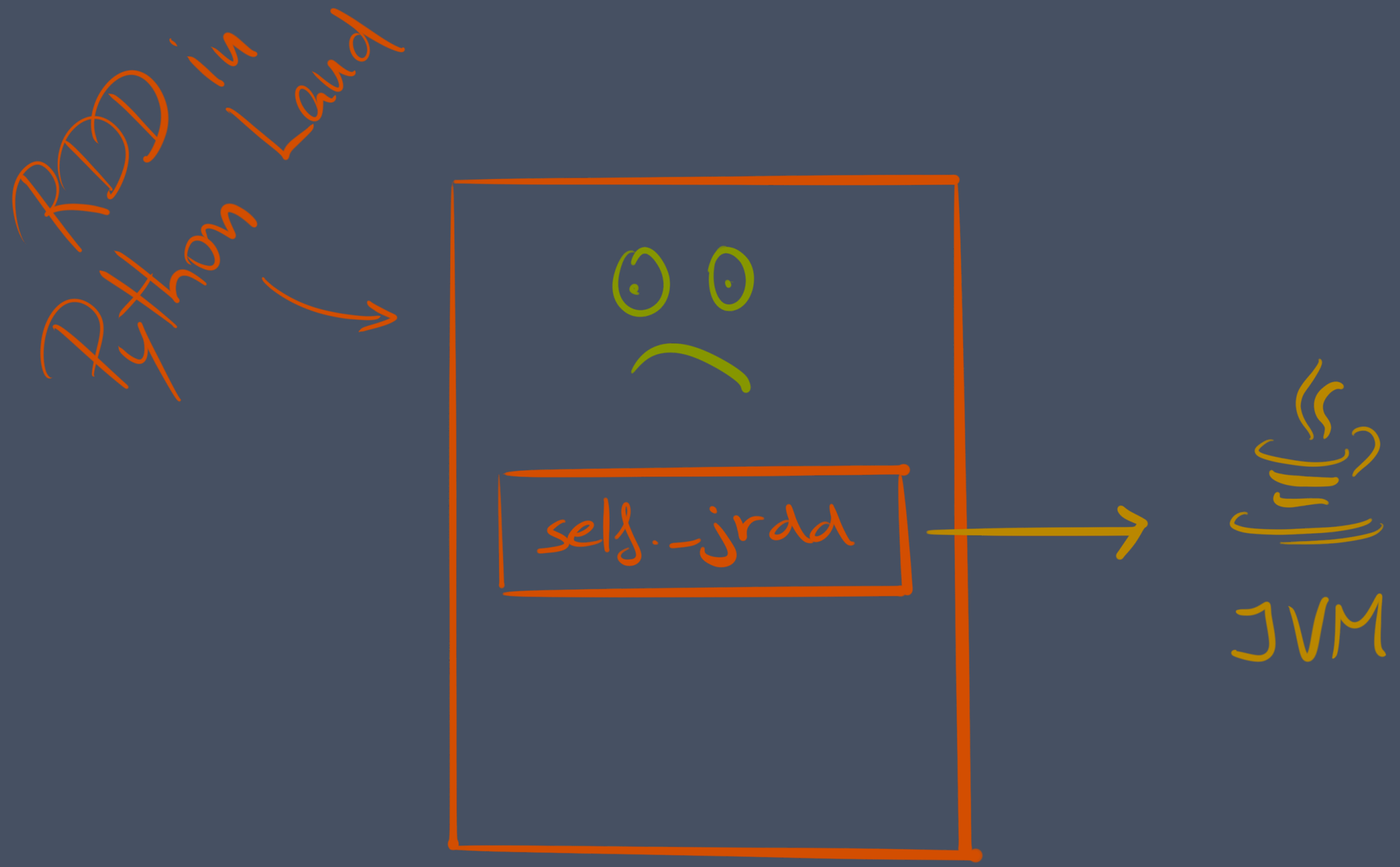

RDD in Python Land
self._jrdd

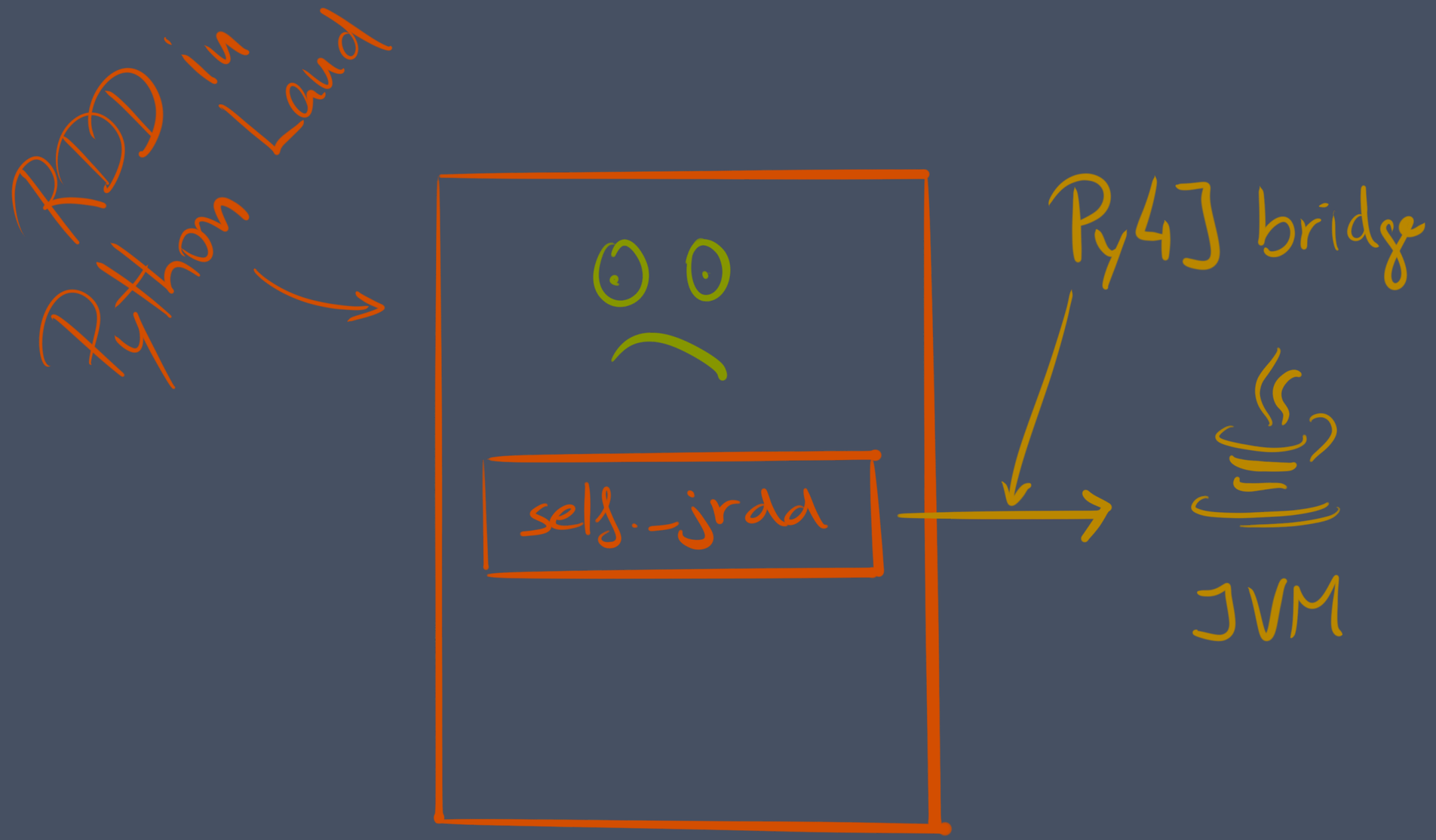
RDD in Python Land

M is for murder

RDD in Python Land

J is for Java


RDD in Python Land
self._jrdd
JVM

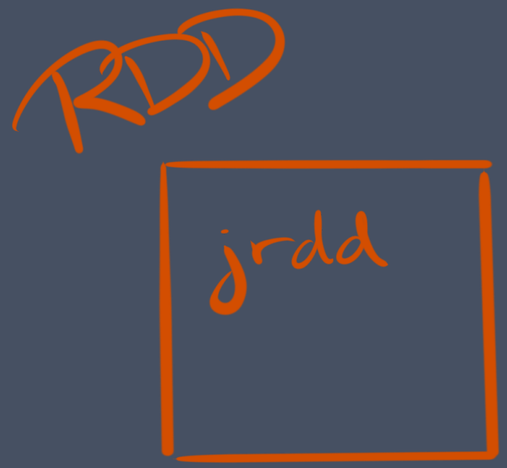

RDD in Python Land
self._jrdd
Py4J bridge
JVM

THE MAIN ENTRYPOINTS ARE RDD AND PipelinedRDD(RDD)

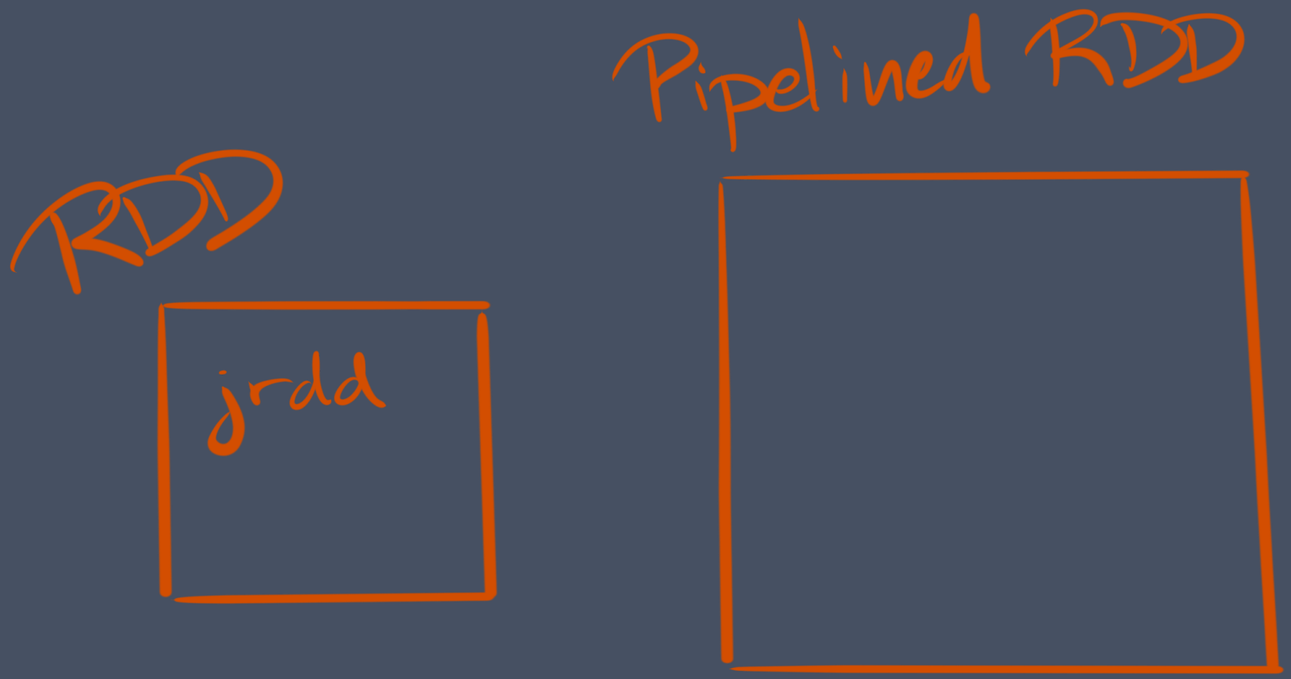
PipelinedRDD

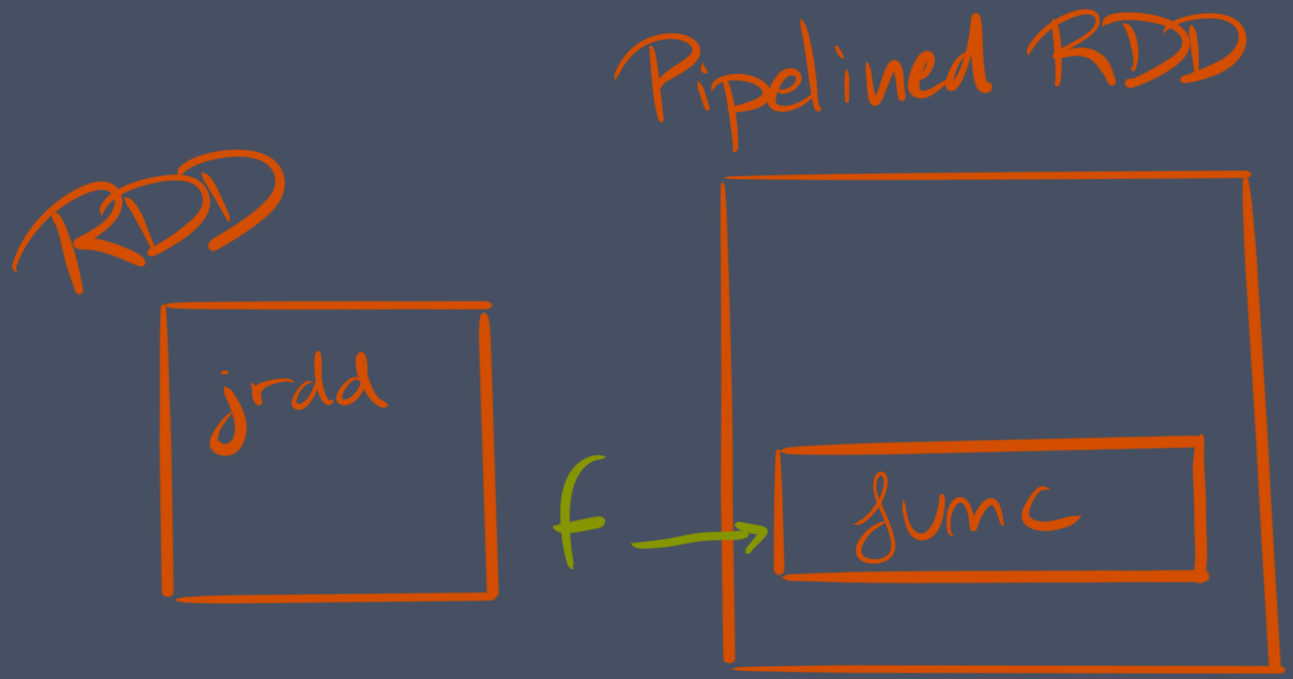
BUILDS IN THE JVM A

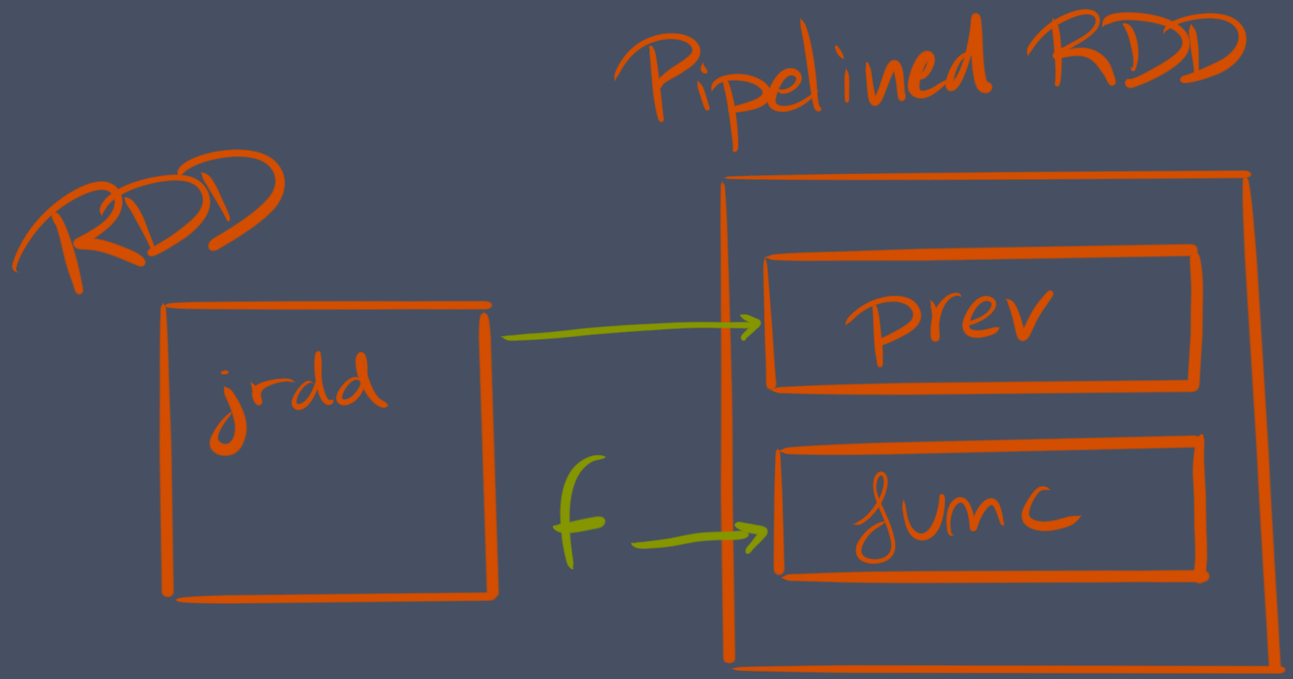
PythonRDD

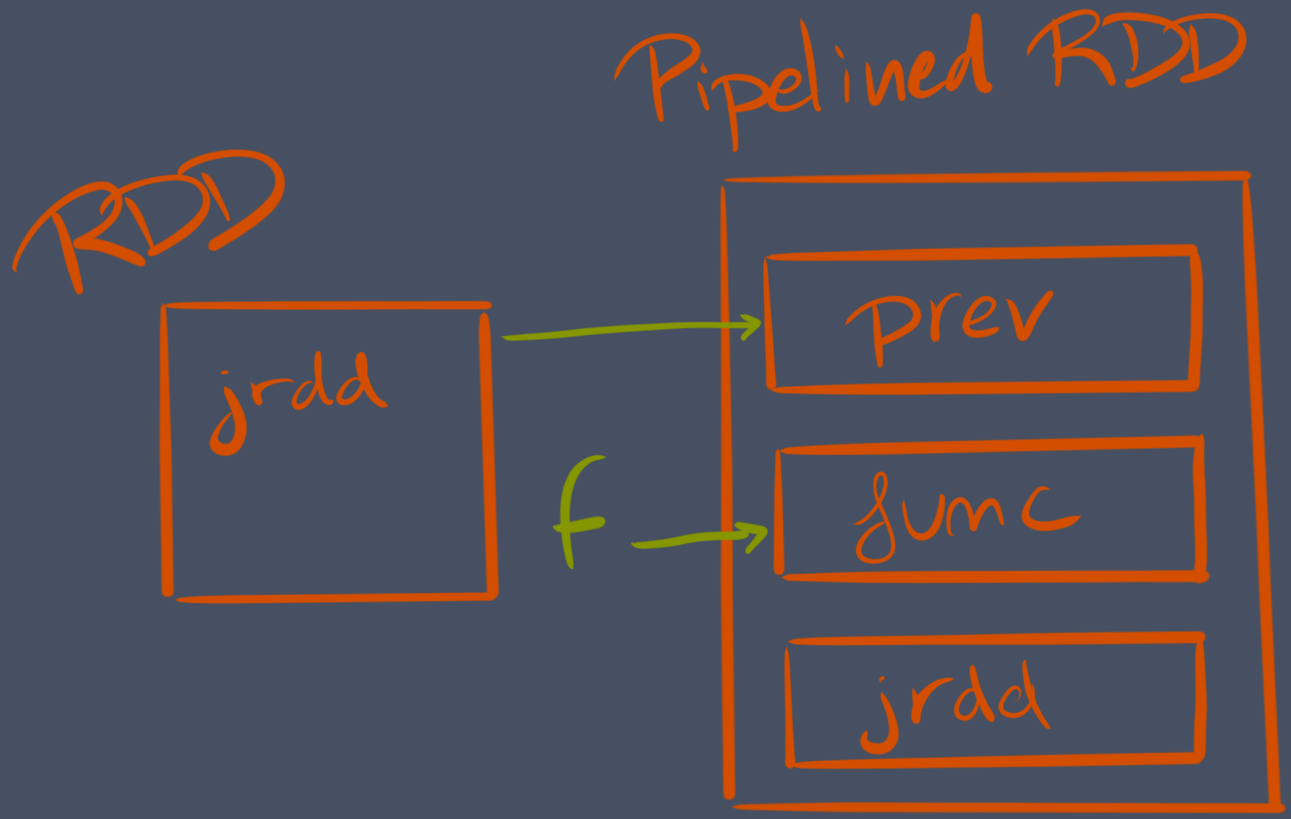

RDD
jrdd

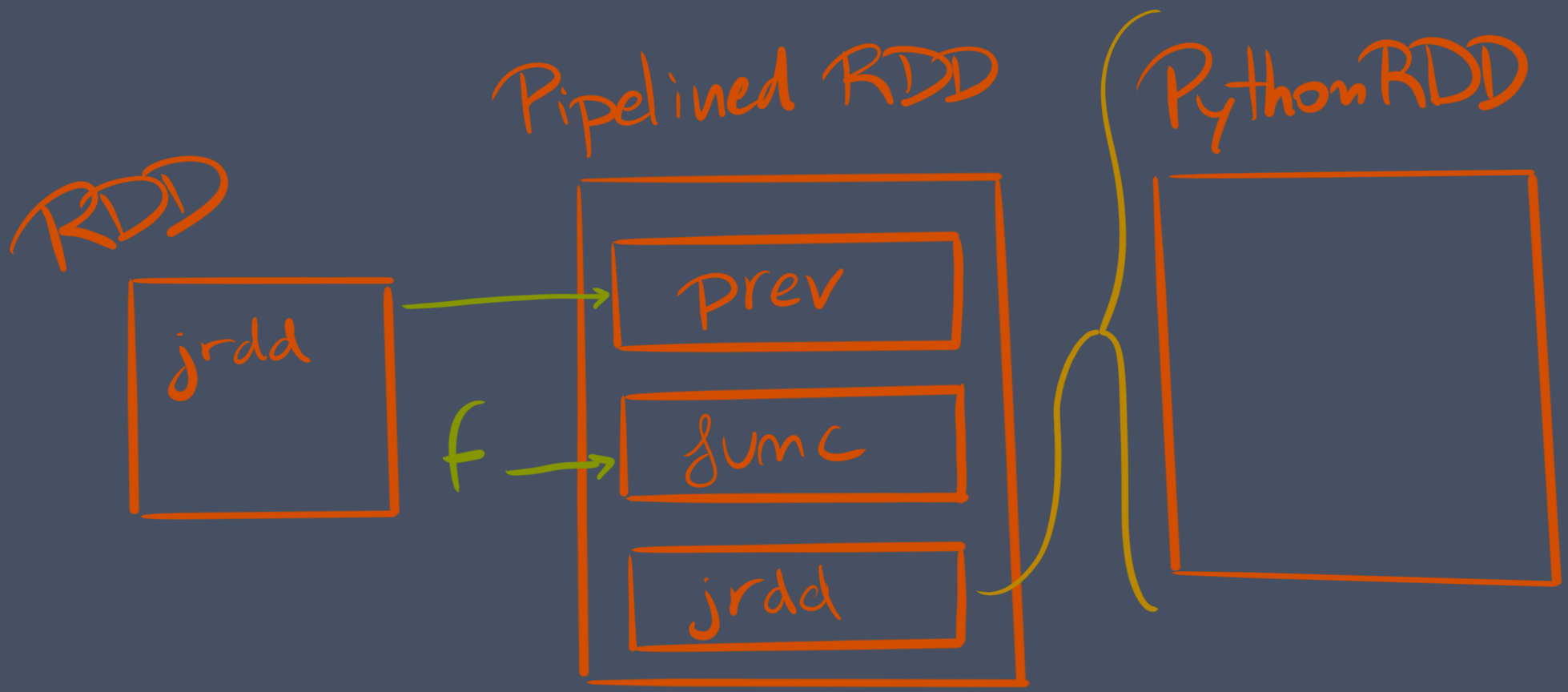

RDD
jrdd
MAP(f)

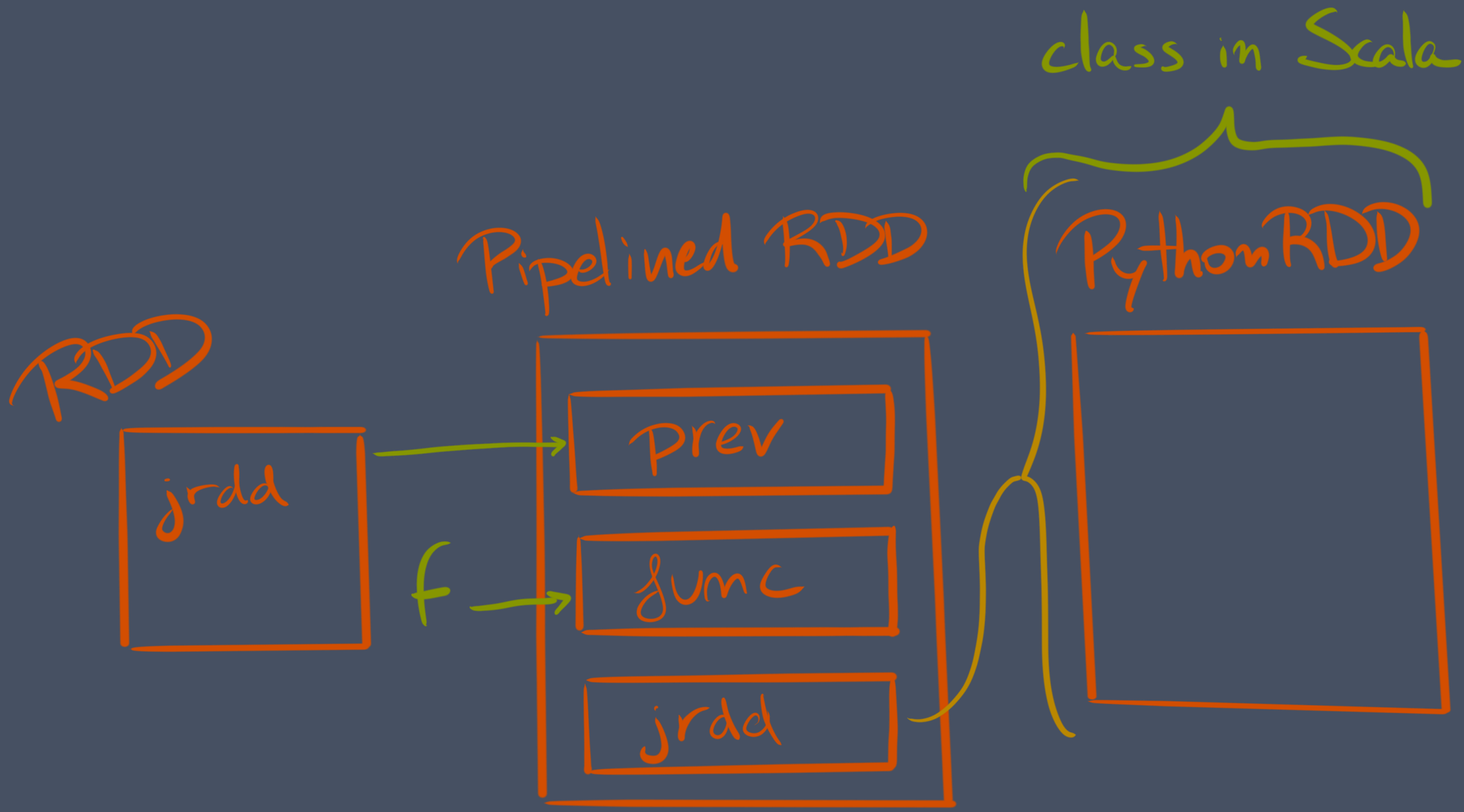

Pipelined RDD
RDD
jrdd

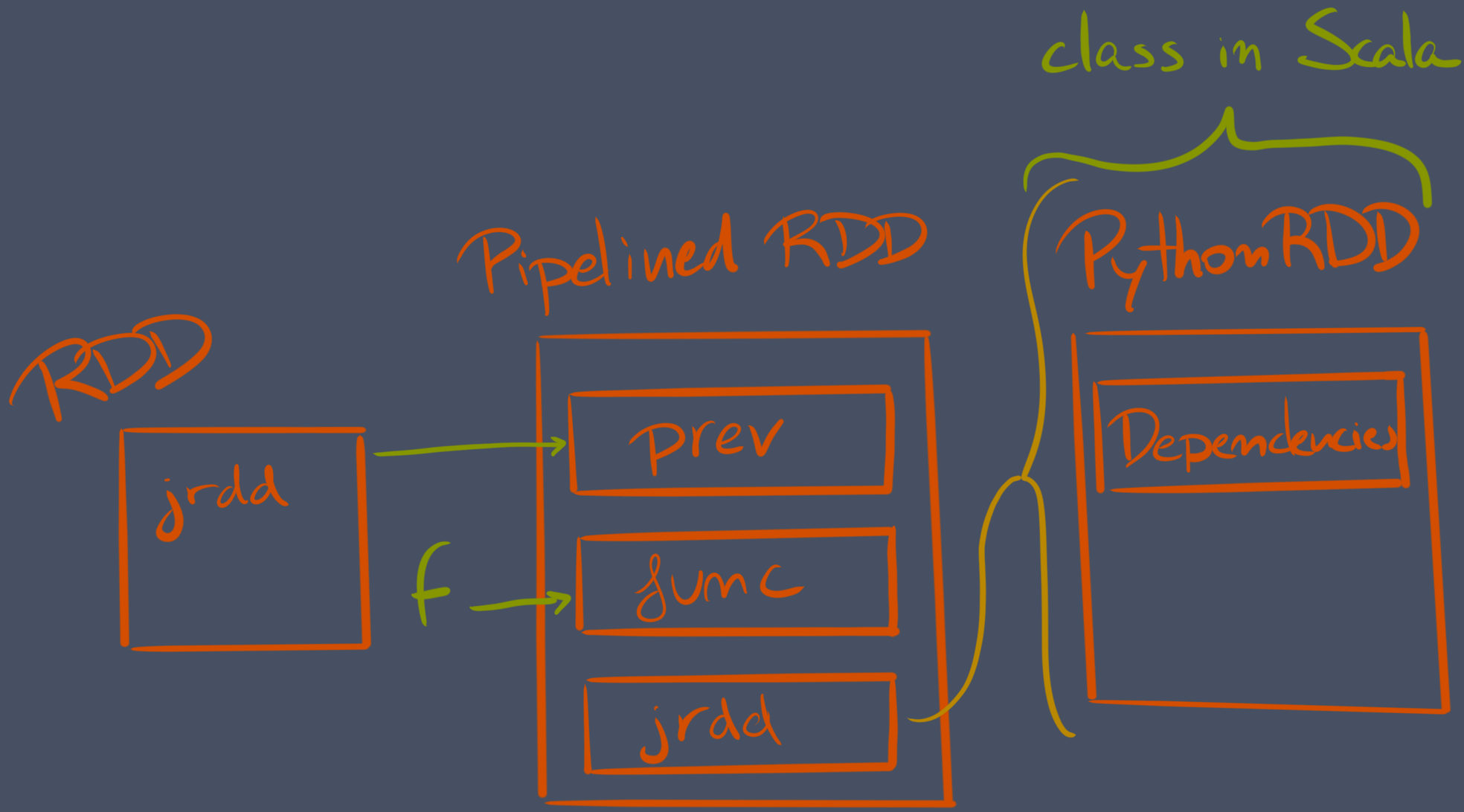

Pipelined RDD
RDD
jrdd
f
jumc

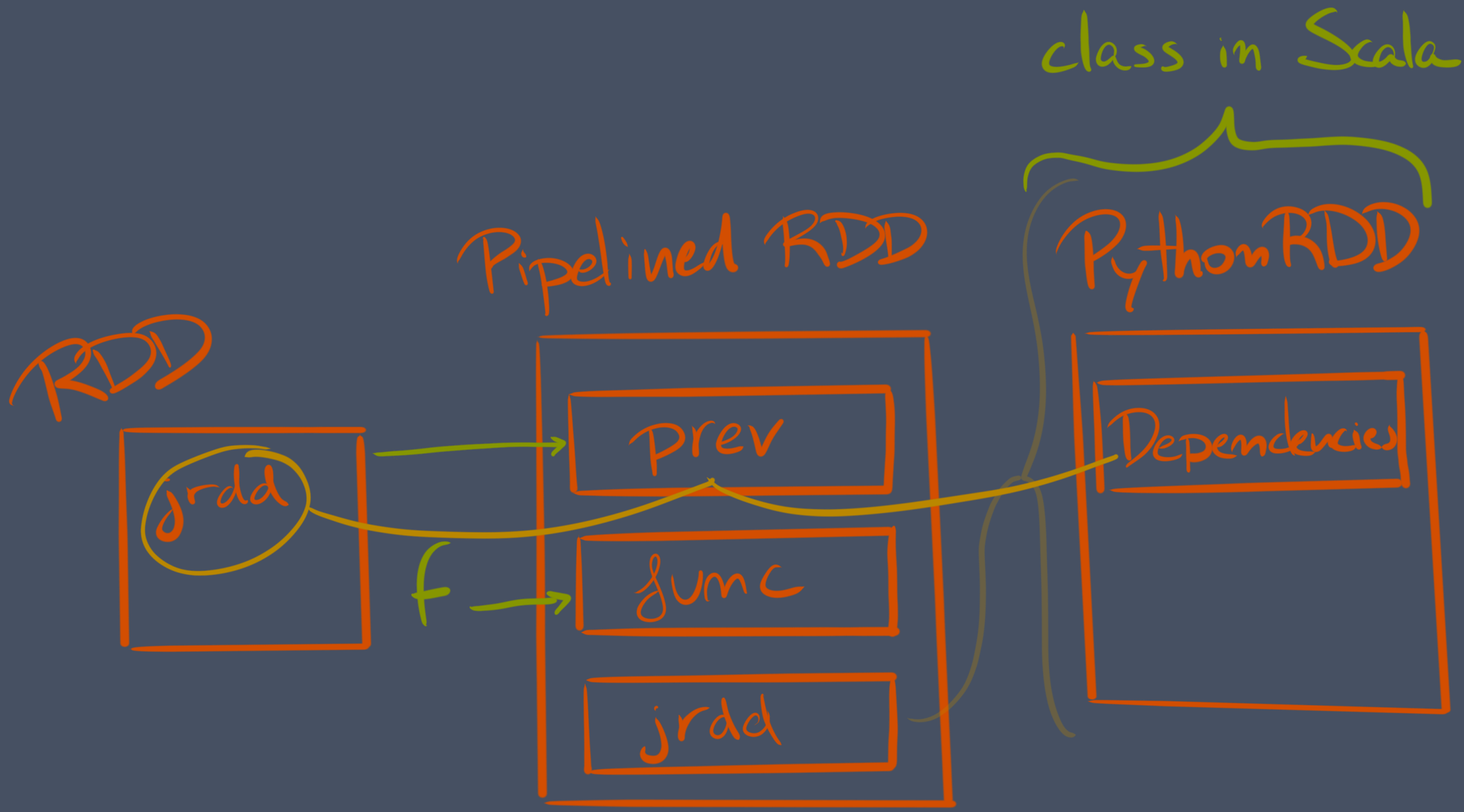

Pipelined RDD
RDD
jrdd
prev
f
func

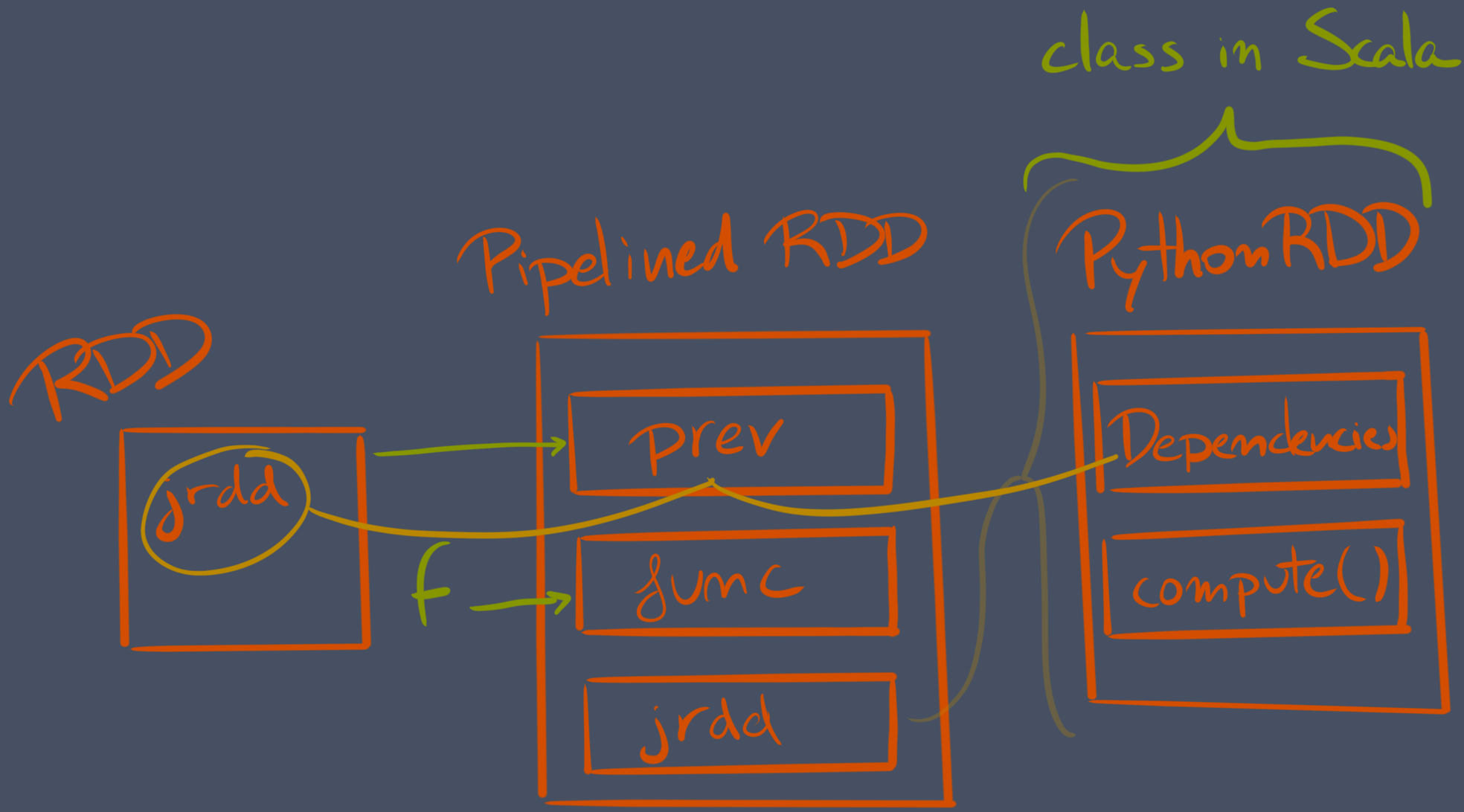

Pipelined RDD
RDD
jrdd
f
prev
func
jrdd

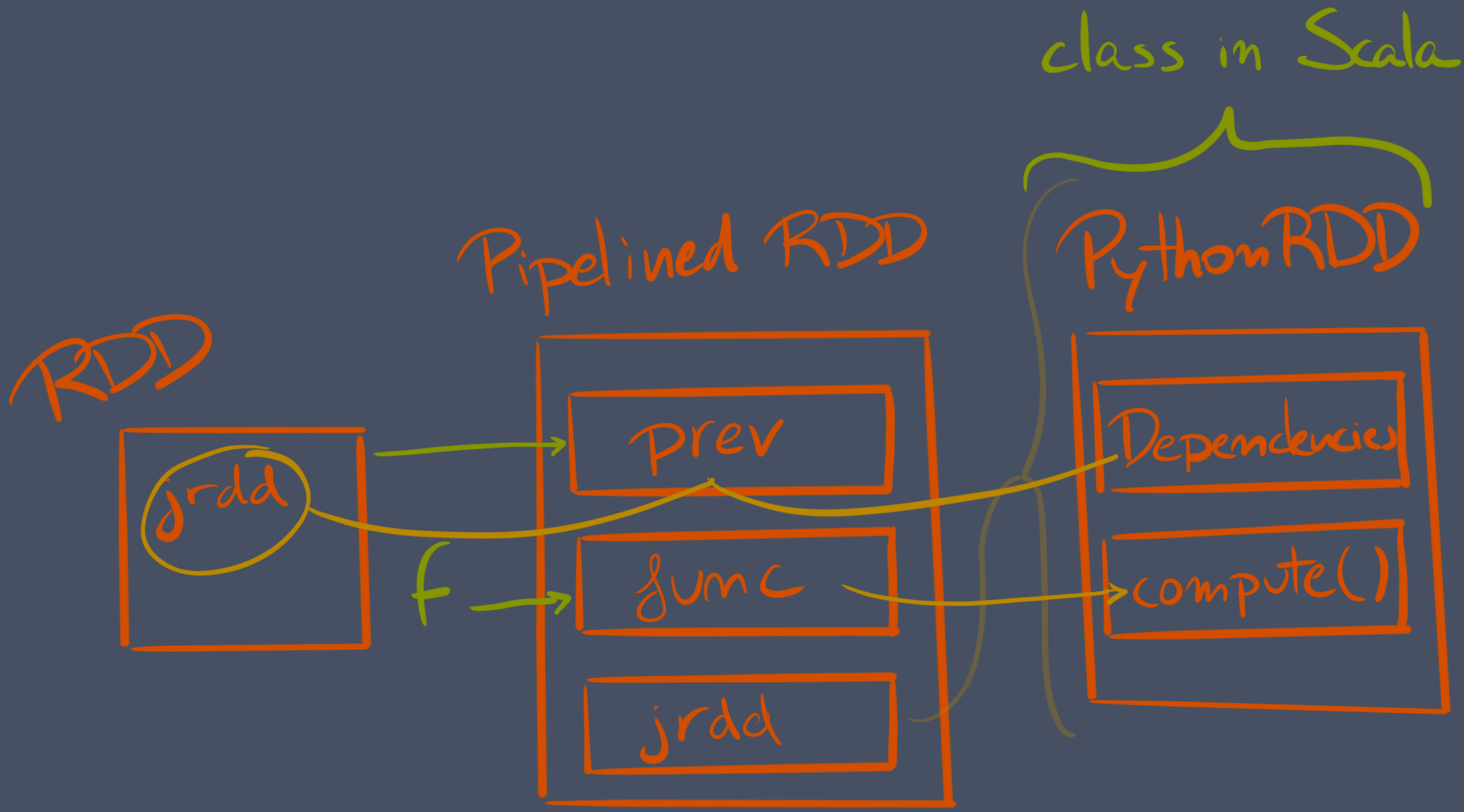

RDD
jrdd
Pipelined RDD
prev
f
func
jrdd
PythonRDD


class in Scala
PythonRDD
Pipelined RDD
RDD
jrdd
prev
f
func
jrdd


class in Scala
RDD
jrdd
Pipelined RDD
prev
f
func
jrdd
PythonRDD
Dependencies


class in Scala
Pipelined RDD
PythonRDD
RDD
prev
Dependencies
jrdd
f
func
jrdd


class in Scala
Pipelined RDD
PythonRDD
RDD
jrdd
prev
Dependencies
f
func
compute()
jrdd

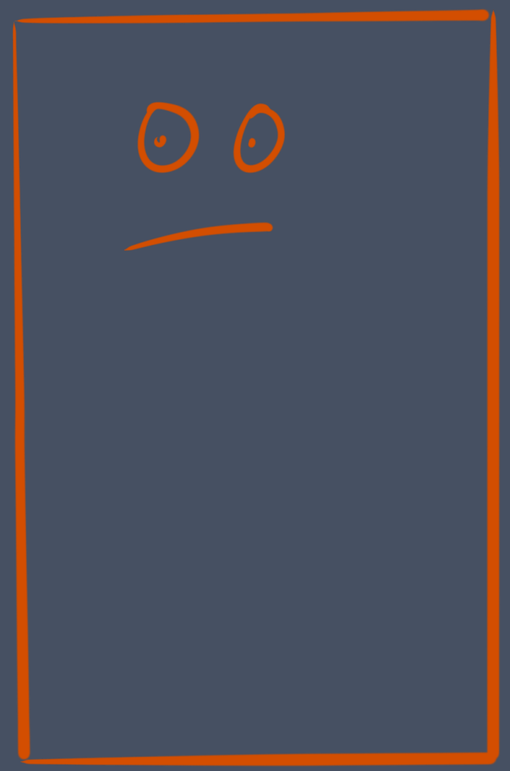

class in Scala
Pipelined RDD
PythonRDD
RDD
jrdd
prev
f
func
jrdd
Dependencies
compute()

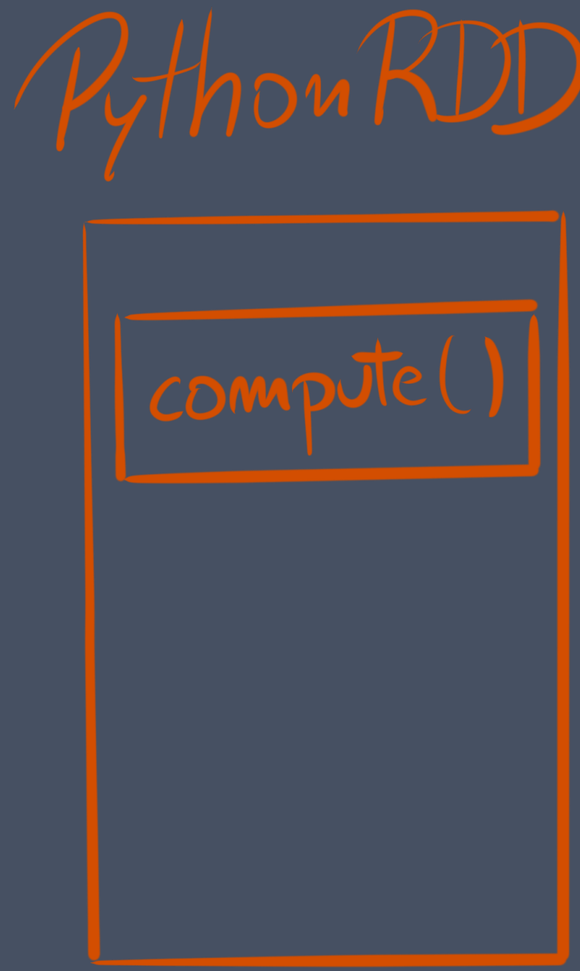
THE MAGIC IS IN compute

`compute`

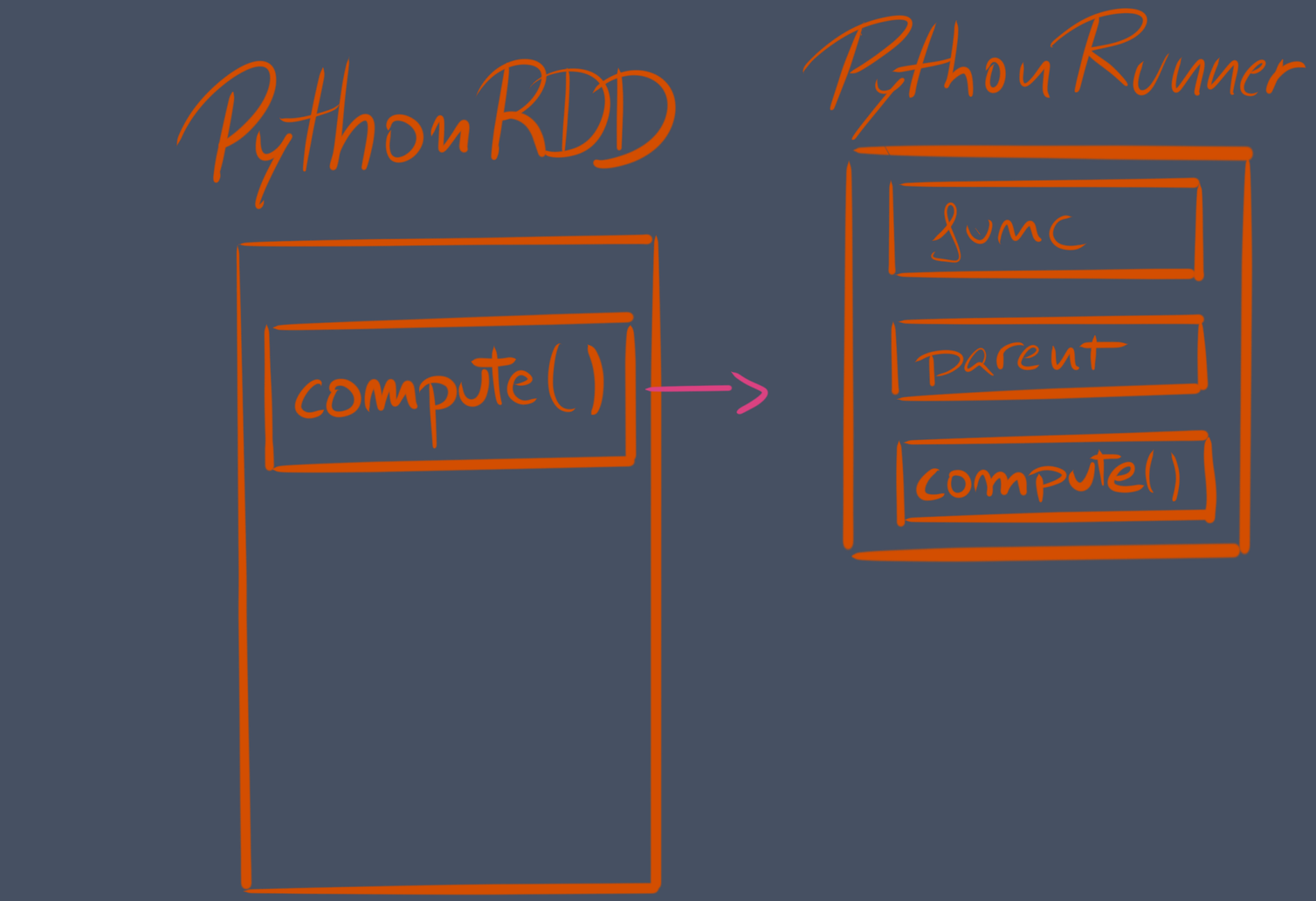
IS RUN ON EACH EXECUTOR AND STARTS A PYTHON WORKER VIA

`PythonRunner`

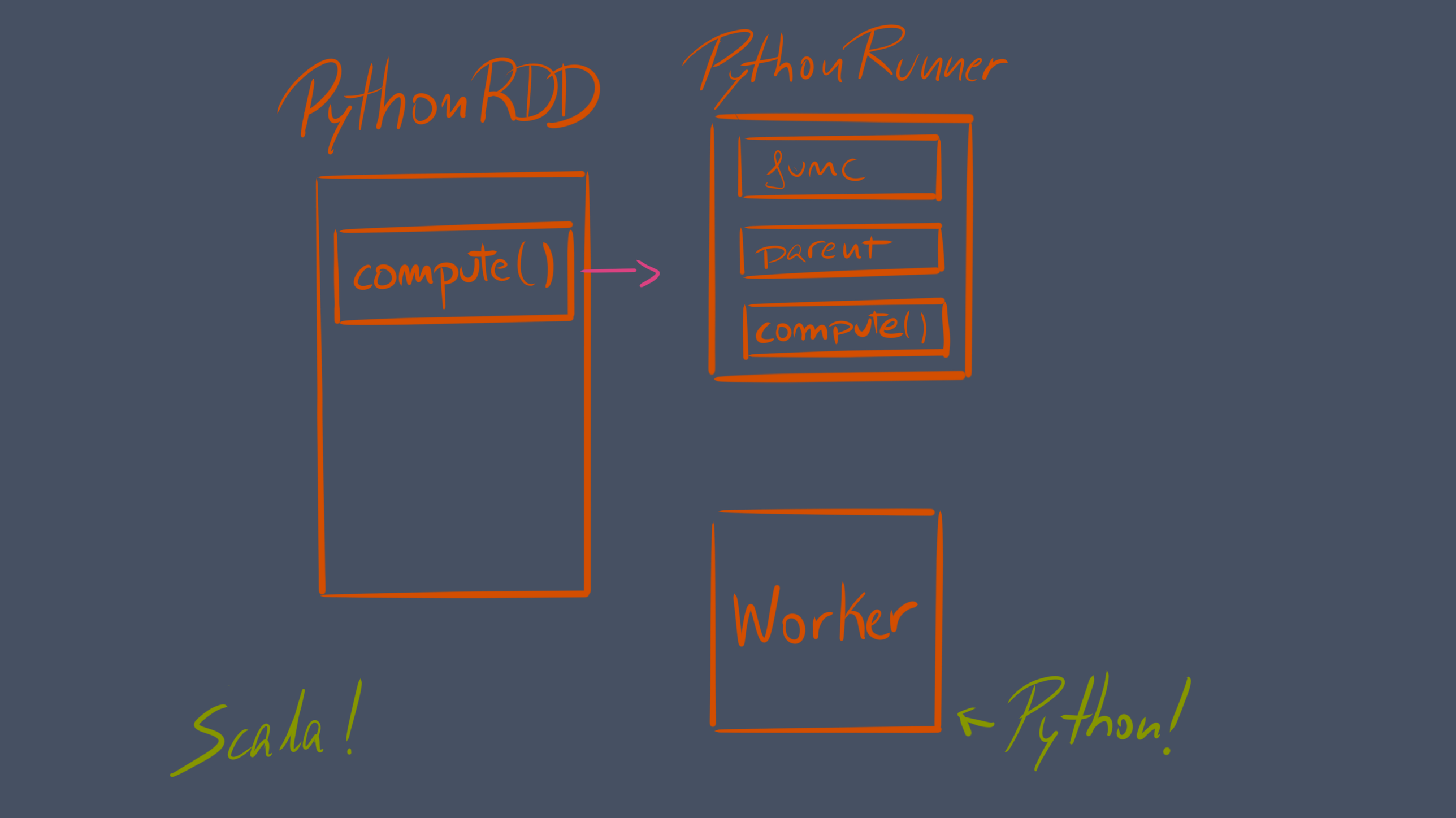


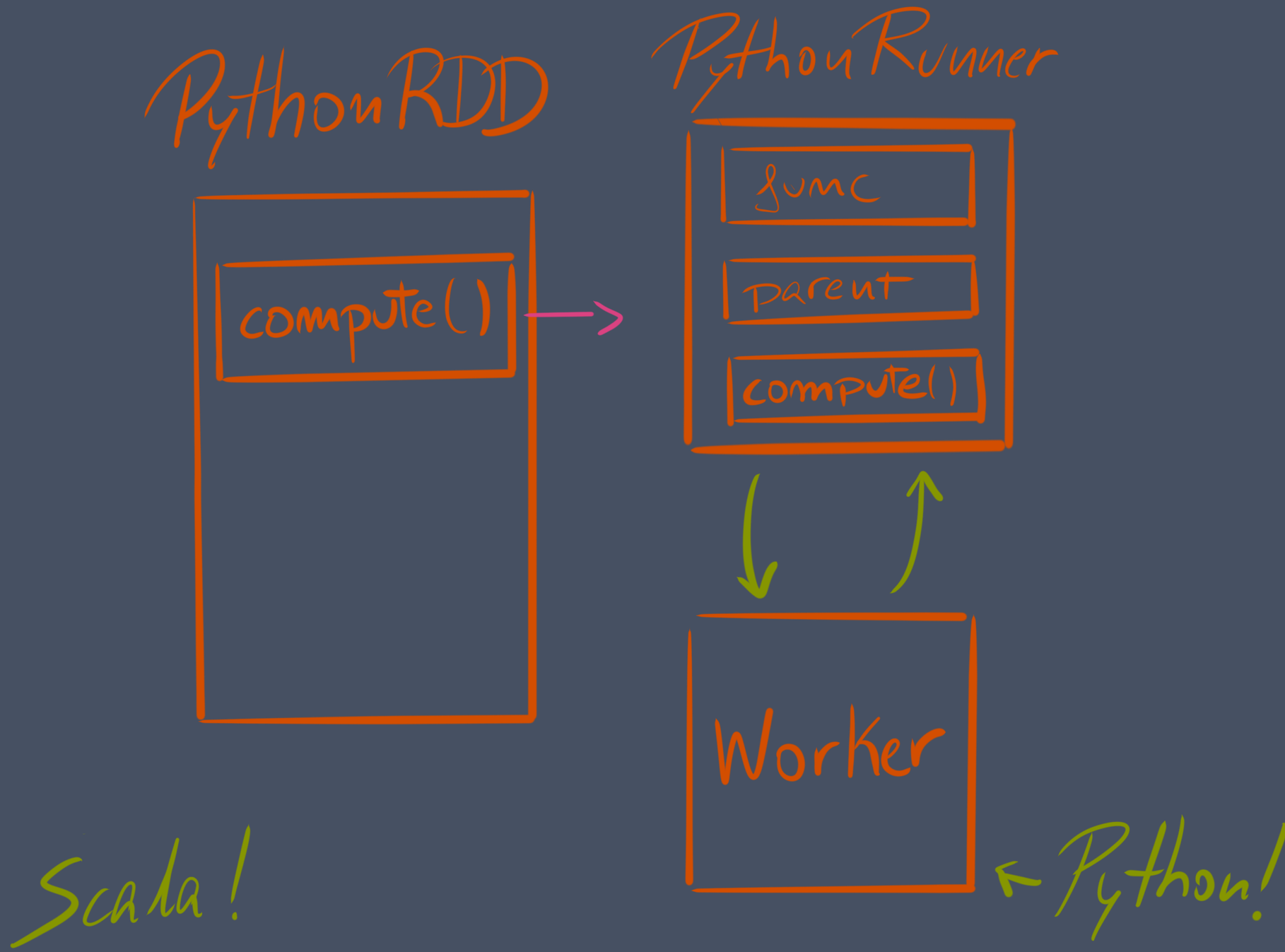


Scala!



Scala!


PythonRDD
compute()
PythonRunner
func
parent
compute()
Worker
Scala!
Python!


PythonRDD
compute()
PythonRunner
func
parent
compute()
Worker
Scala!
Python!

WORKERS ACT AS STANDALONE PROCESSORS OF STREAMS OF DATA

WORKERS ACT AS STANDALONE PROCESSORS OF STREAMS OF DATA

- CONNECTS BACK TO THE JVM THAT STARTED IT

WORKERS ACT AS STANDALONE PROCESSORS OF STREAMS OF DATA

- CONNECTS BACK TO THE JVM THAT STARTED IT
- LOAD INCLUDED PYTHON LIBRARIES

WORKERS ACT AS STANDALONE PROCESSORS OF STREAMS OF DATA

- CONNECTS BACK TO THE JVM THAT STARTED IT
- LOAD INCLUDED PYTHON LIBRARIES
- DESERIALIZES THE PICKLED FUNCTION COMING FROM THE STREAM

WORKERS ACT AS STANDALONE PROCESSORS OF STREAMS OF DATA

- CONNECTS BACK TO THE JVM THAT STARTED IT
- LOAD INCLUDED PYTHON LIBRARIES
- DESERIALIZES THE PICKLED FUNCTION COMING FROM THE STREAM
- APPLIES THE FUNCTION TO THE DATA COMING FROM THE STREAM

WORKERS ACT AS STANDALONE PROCESSORS OF STREAMS OF DATA

- CONNECTS BACK TO THE JVM THAT STARTED IT
- LOAD INCLUDED PYTHON LIBRARIES
- DESERIALIZES THE PICKLED FUNCTION COMING FROM THE STREAM
- APPLIES THE FUNCTION TO THE DATA COMING FROM THE STREAM
- SENDS THE OUTPUT BACK

BUT... WASN'T SPARK MAGICALLY OPTIMISING EVERYTHING?

YES. FOR SPARK

DataFrame

SPARK WILL GENERATE A PLAN

(A DIRECTED ACYCLIC GRAPH)

TO COMPUTE THE RESULT

AND THE PLAN WILL BE OPTIMISED USING CATALYST

DEPENDING ON THE FUNCTION, THE OPTIMISER WILL CHOOSE

PythonUDFRunner

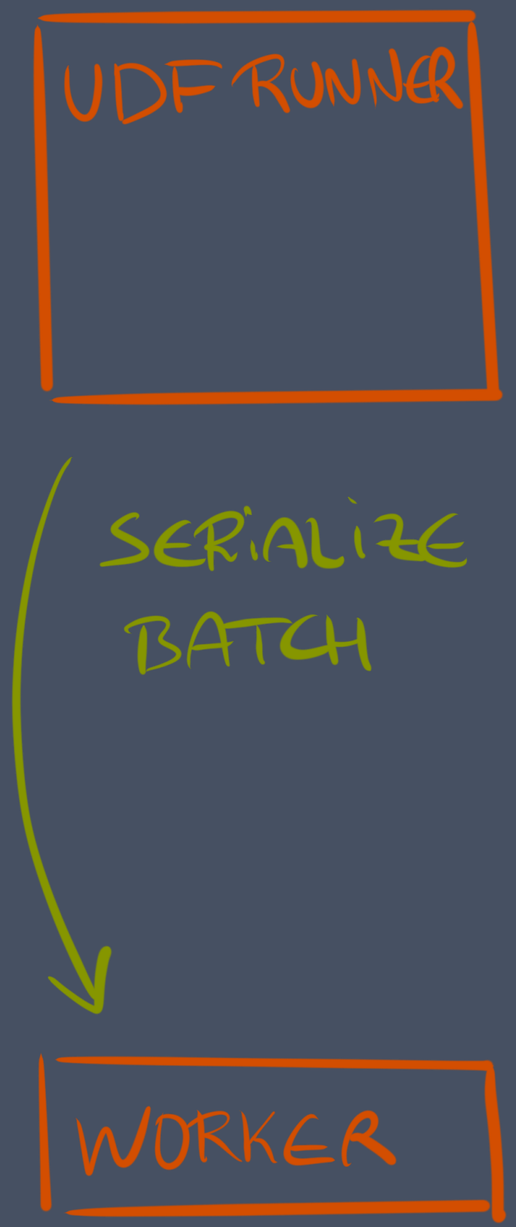
OR

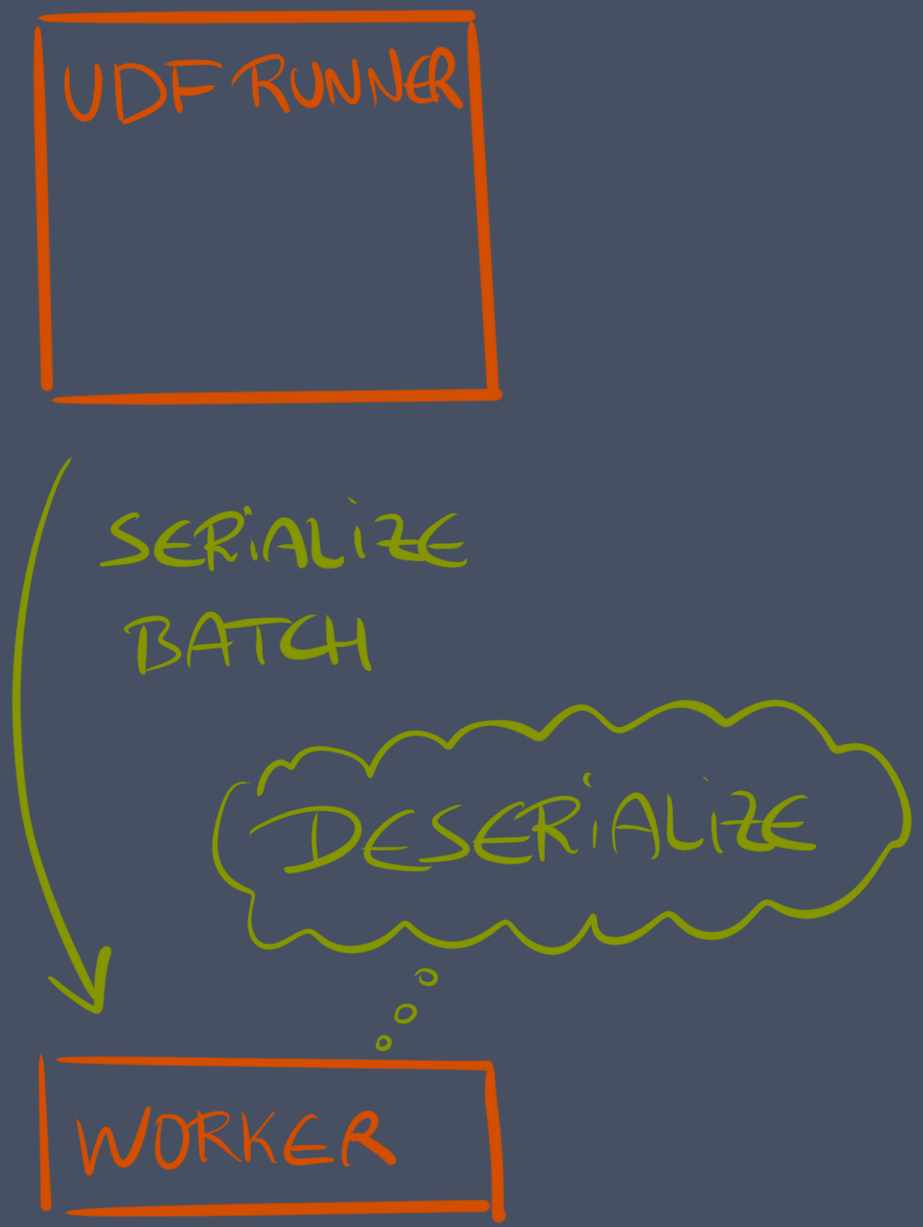
PythonArrowRunner

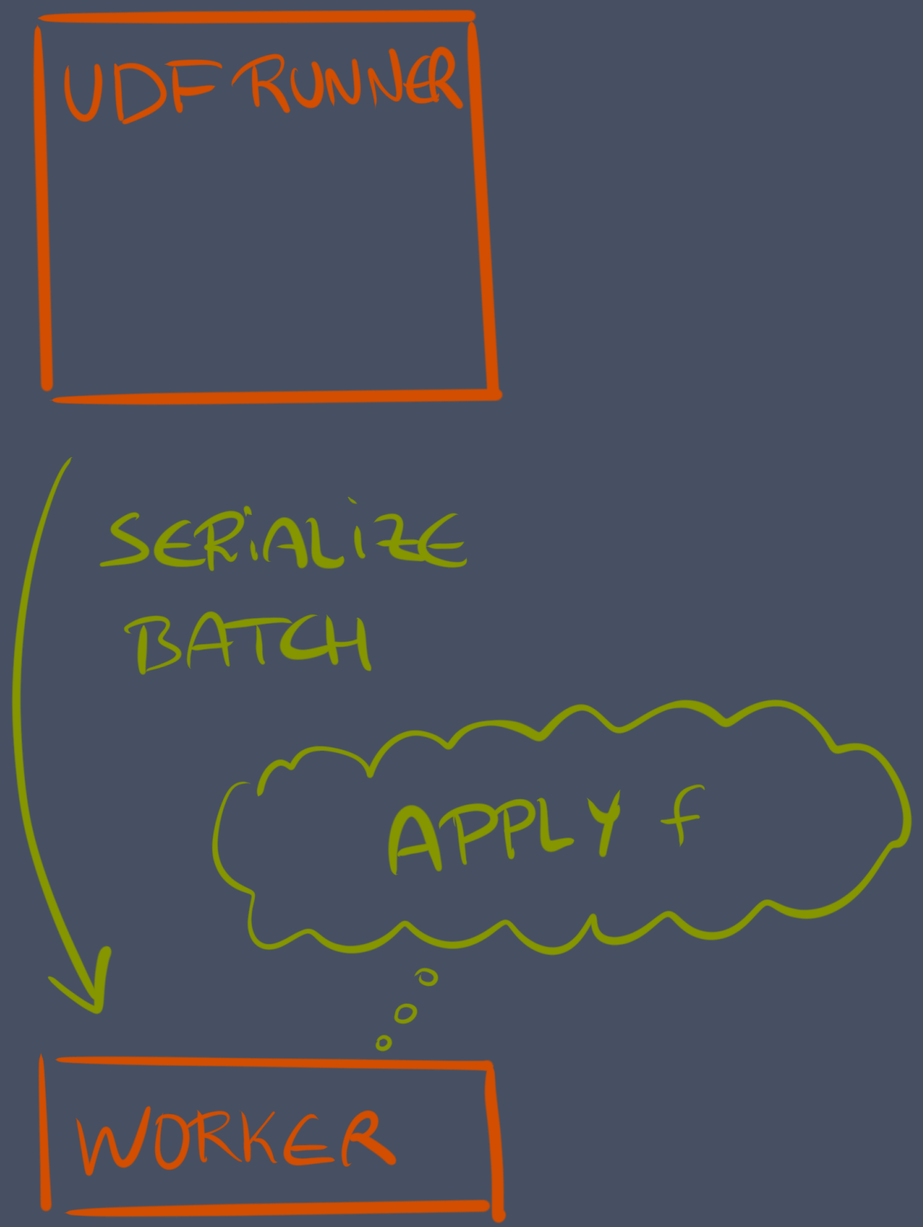
(BOTH EXTEND PythonRunner)

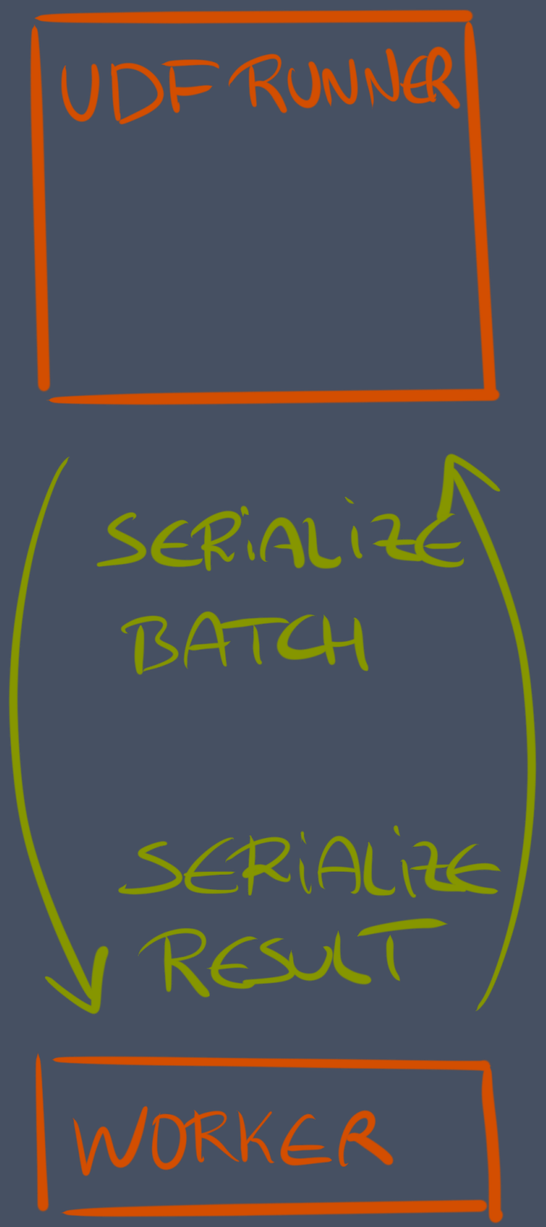

UDF RUNNER


WORKER

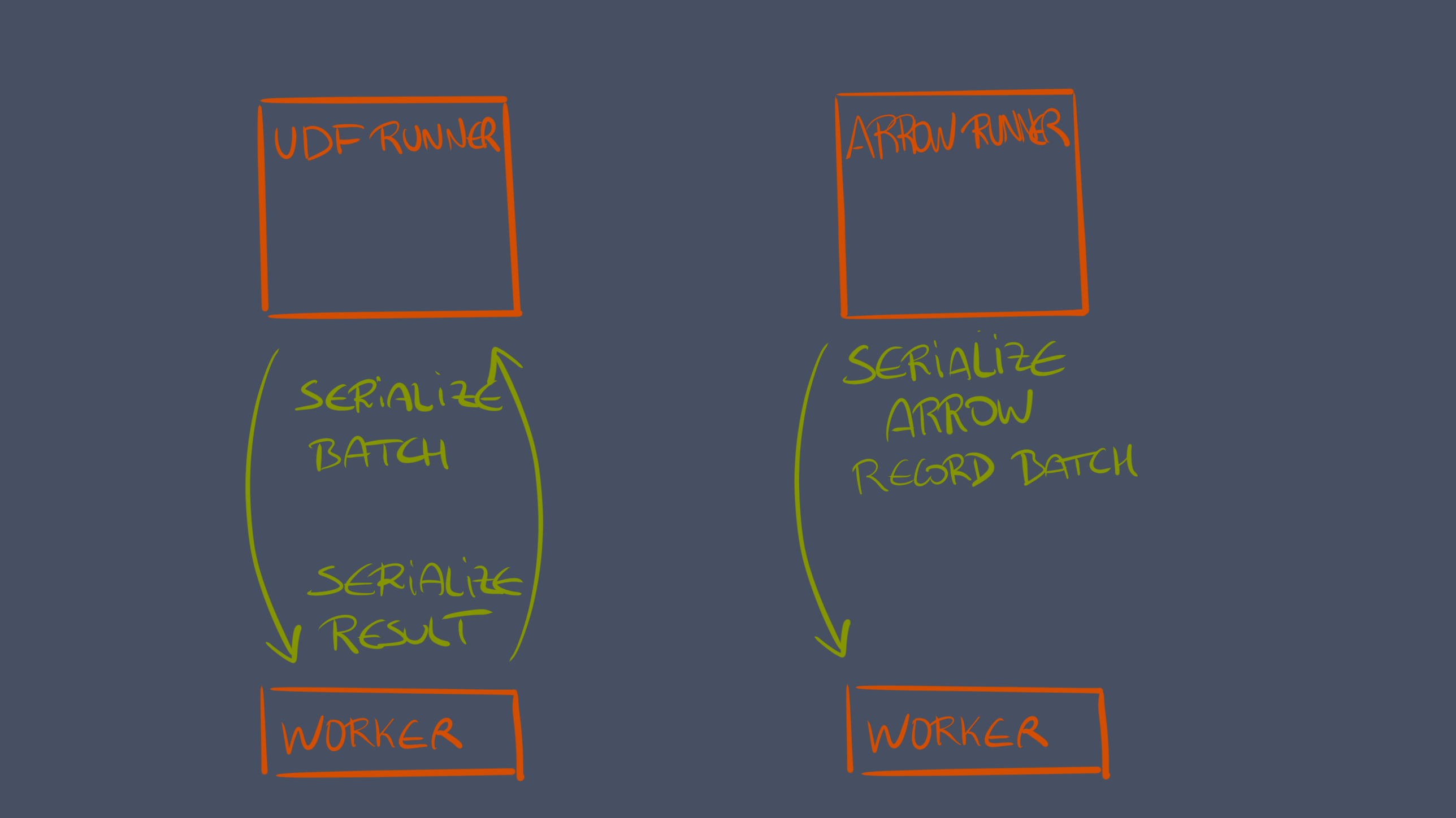

UDF RUNNER
SERIALIZE
BATCH
WORKER

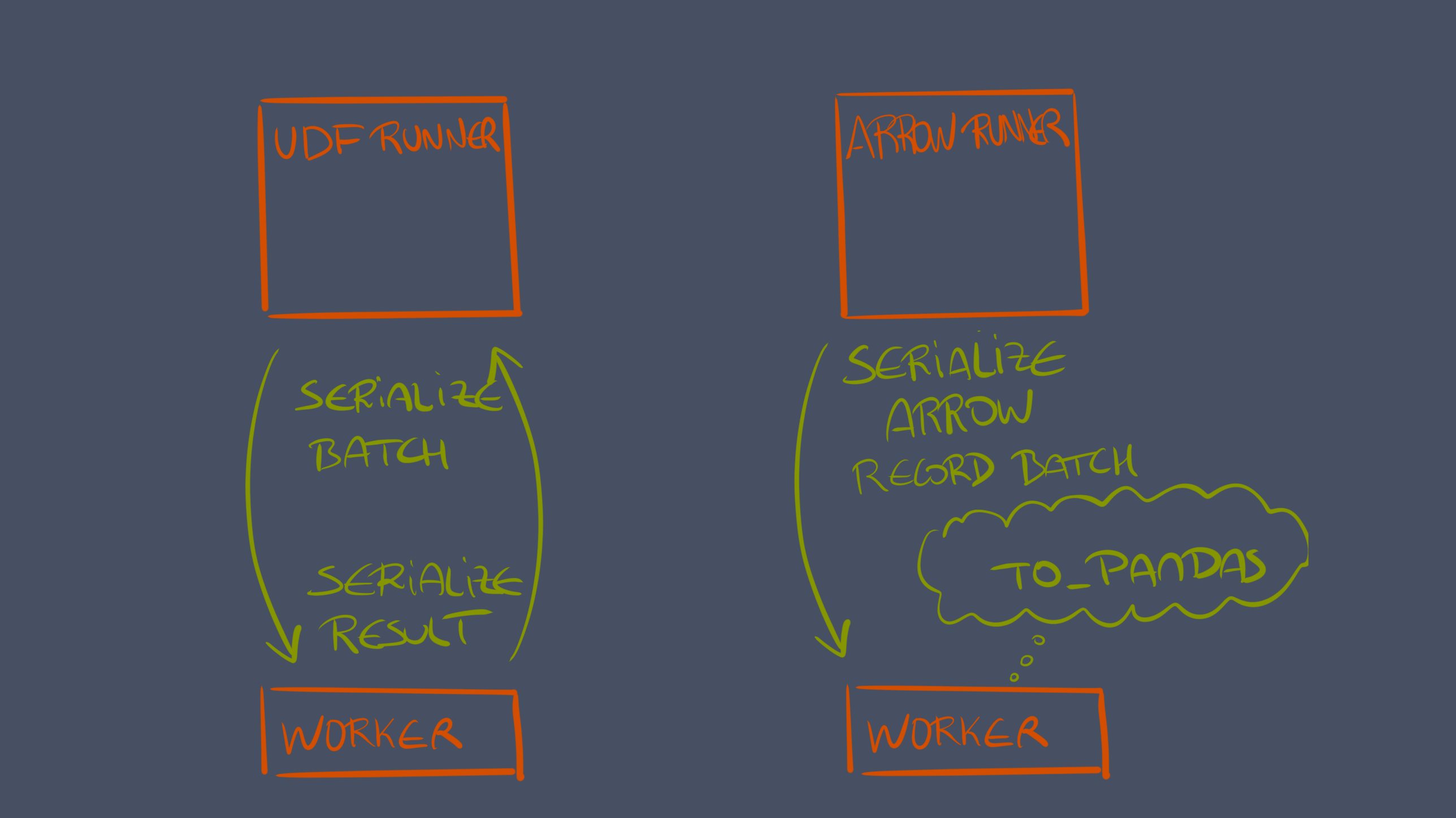

UDF RUNNER
SERIALIZE
BATCH
DESERIALIZE
WORKER

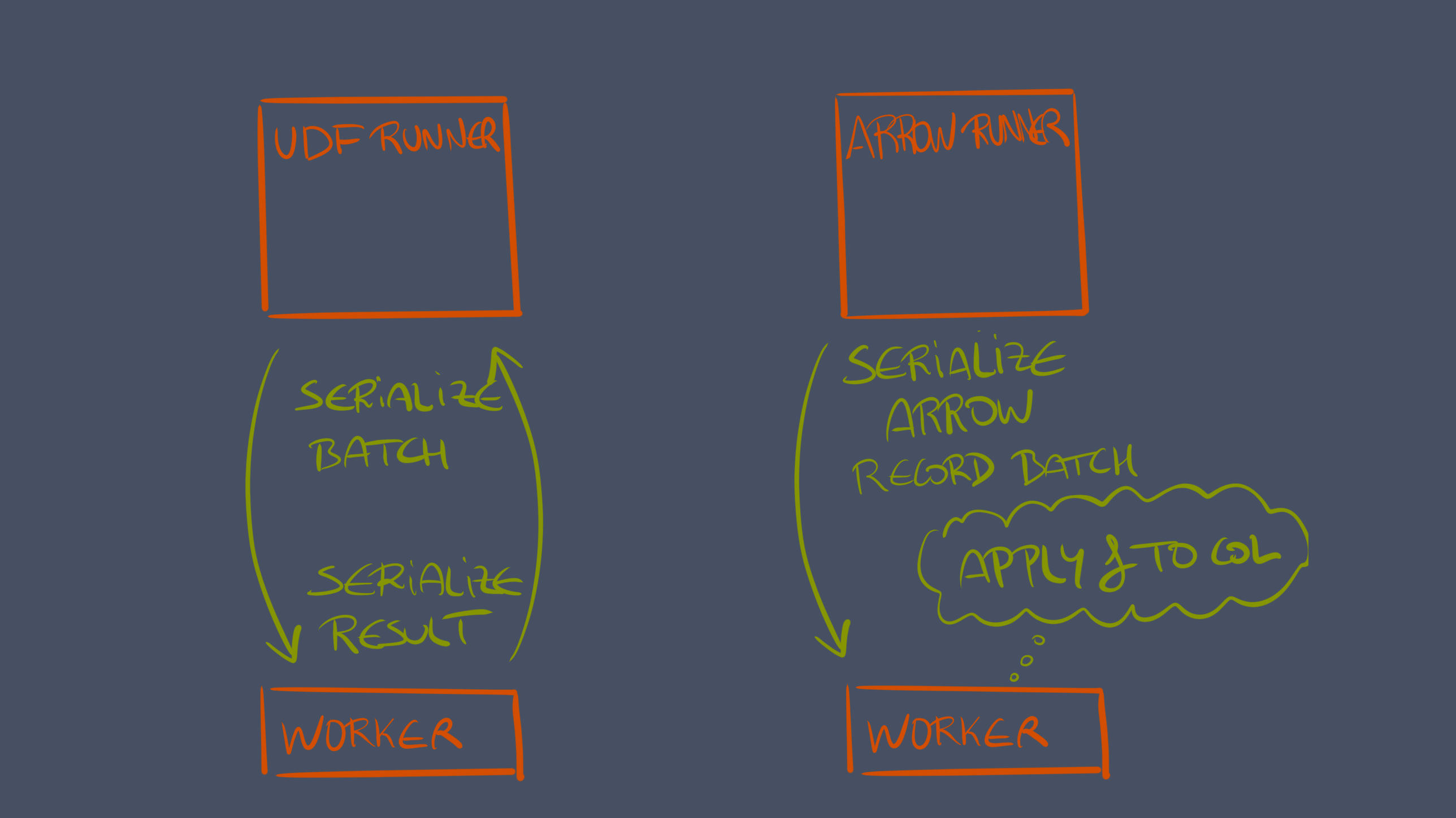

UDF RUNNER
SERIALIZE
BATCH
APPLY f
WORKER


UDF RUNNER
SERIALIZE BATCH
SERIALIZE RESULT
WORKER


UDF RUNNER
ARROW RUNNER
SERIALIZE BATCH
SERIALIZE RESULT
WORKER
WORKER


UDF RUNNER
SERIALIZE BATCH
SERIALIZE RESULT
WORKER
ARROW RUNNER
SERIALIZE ARROW RECORD BATCH
WORKER


UDF RUNNER
SERIALIZE BATCH
SERIALIZE RESULT
WORKER
ARROW RUNNER
SERIALIZE ARROW RECORD BATCH
TO_PANDAS
WORKER


UDF RUNNER
SERIALIZE BATCH
SERIALIZE RESULT
WORKER
ARROW RUNNER
SERIALIZE ARROW RECORD BATCH
APPLY f TO GOL
WORKER


UDF RUNNER
SERIALIZE BATCH
SERIALIZE RESULT
WORKER
ARROW RUNNER
SERIALIZE ARROW RECORD BATCH
SERIALIZE RESULT AS ARROW
WORKER

IF WE CAN DEFINE OUR FUNCTIONS USING PANDAS `Series` TRANSFORMATIONS WE CAN SPEED UP PYSPARK CODE FROM 3X TO 100X!

RESOURCES

- SPARK DOCUMENTATION
- HIGH PERFORMANCE SPARK BY HOLDEN KARAU
- MASTERING APACHE SPARK 2.3 BY JACEK LASKOWSKI
- SPARK'S GITHUB
- BECOME A CONTRIBUTOR

QUESTIONS?


THANKS!

FURTHER REFERENCES

ARROW

ARROW'S HOME
ARROW'S GITHUB
ARROW SPEED TESTS
ARROW TO PANDAS CONVERSION SPEED
STREAMING COLUMNAR DATA WITH APACHE ARROW
WHY PANDAS USERS SHOULD BE EXCITED BY APACHE ARROW
ARROW-PANDAS COMPATIBILITY LAYER CODE
ARROW TABLE CODE
PYARROW IN-MEMORY DATA MODEL

PANDAS

PANDAS' HOME
PANDAS' GITHUB
IDIOMATIC PANDAS GUIDE
PANDAS INTERNALS CODE
PANDAS INTERNALS DESIGN
DEMYSTIFYING PANDAS' INTERNALS (TALK BY MARC GARCIA)

SPARK/PYSPARK

PYSPARK SERIALIZERS CODE
FIRST STEPS TO USING ARROW (ONLY IN THE PYSPARK DRIVER)
SPEEDING UP PYSPARK WITH APACHE ARROW
ORIGINAL JIRA ISSUE: VECTORIZED UDFS IN SPARK
INITIAL DOC DRAFT
BLOG POST BY BRYAN CUTLER (LEADER FOR THE VEC UDFS PR)
INTRODUCING PANDAS UDF FOR PYSPARK
ORG.APACHE.SPARK.SQL.VECTORIZED

PY4J

PY4J'S HOME
PY4J'S GITHUB
REFLECTION ENGINE

EOF