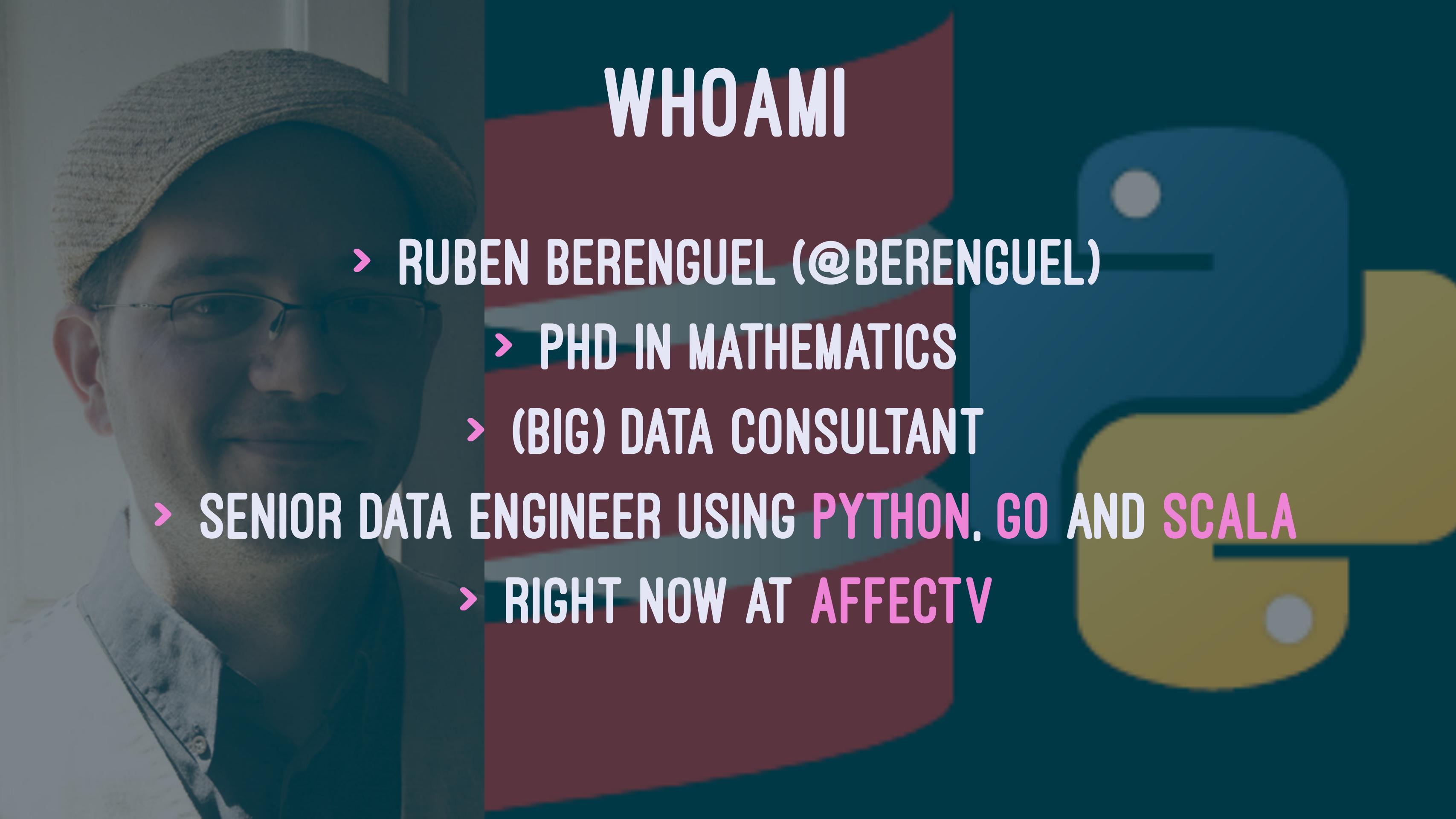


HOW DOES THAT PYSPARK  
THING WORK? AND WHY  
ARROW MAKES IT FASTER?

A close-up photograph of a man's face. He is wearing dark-rimmed glasses and a light-colored beanie. He appears to be looking down at a laptop screen, which is partially visible at the bottom of the frame. The background is blurred.

# 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)

$c_1$	$c_2$	$c_3$	$\dots$	$c_{42}$	$c_{43}$	$\dots$
$\vdots$	$\vdots$	$\vdots$				

SUM  
↓

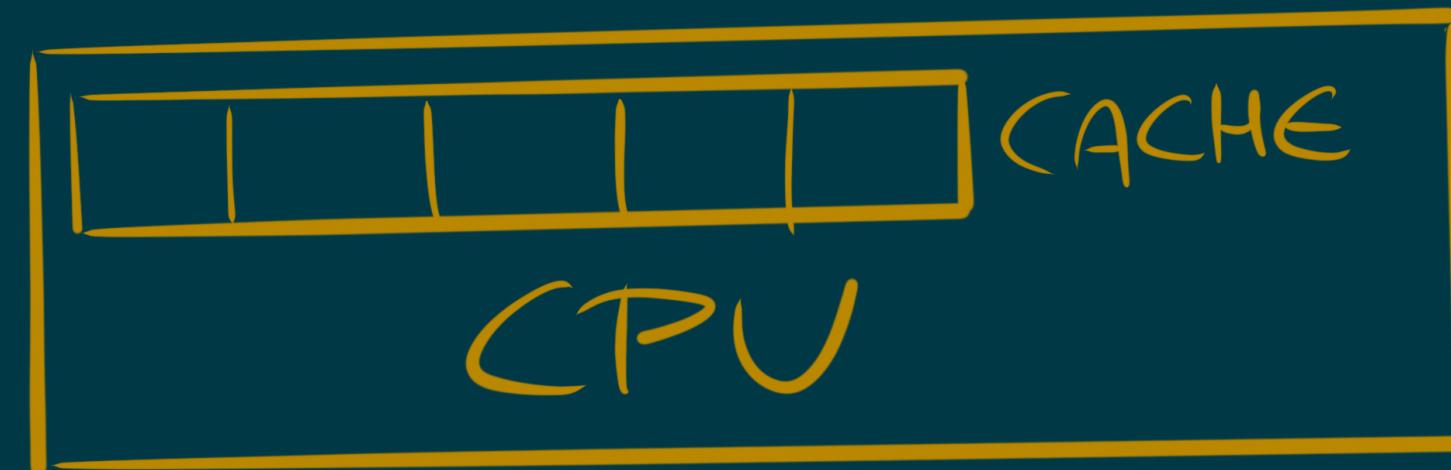
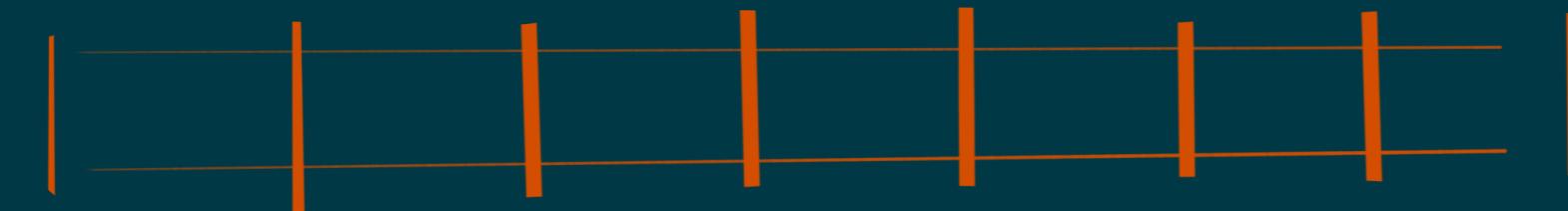
$c_1$	$c_2$	$c_3$	$\dots$	$c_{42}$	$c_{43}$	$\dots$
⋮	⋮	⋮				

ROW

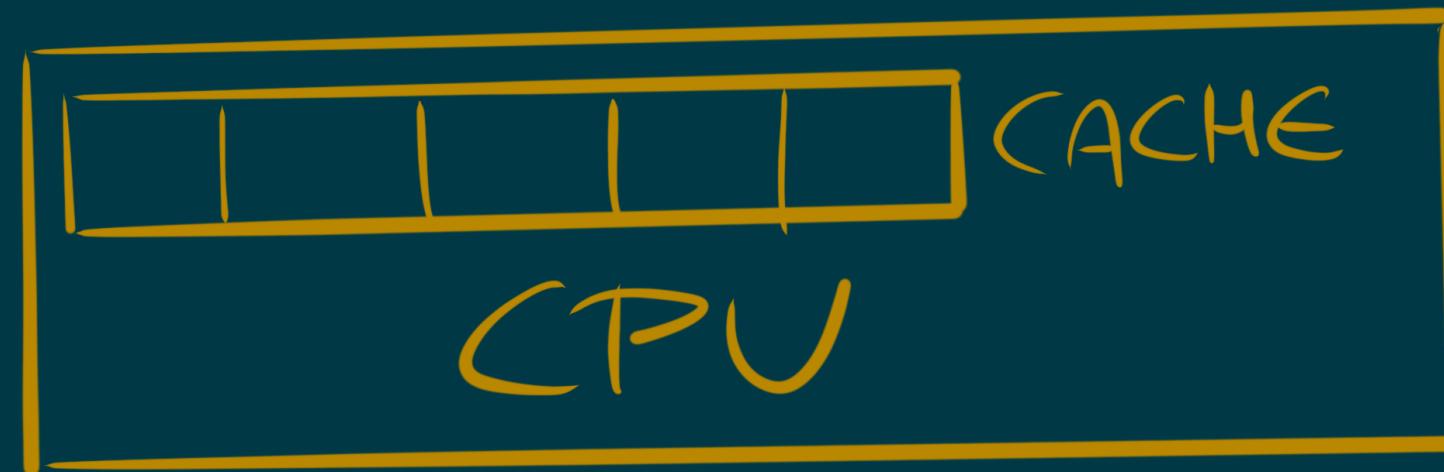


CPU

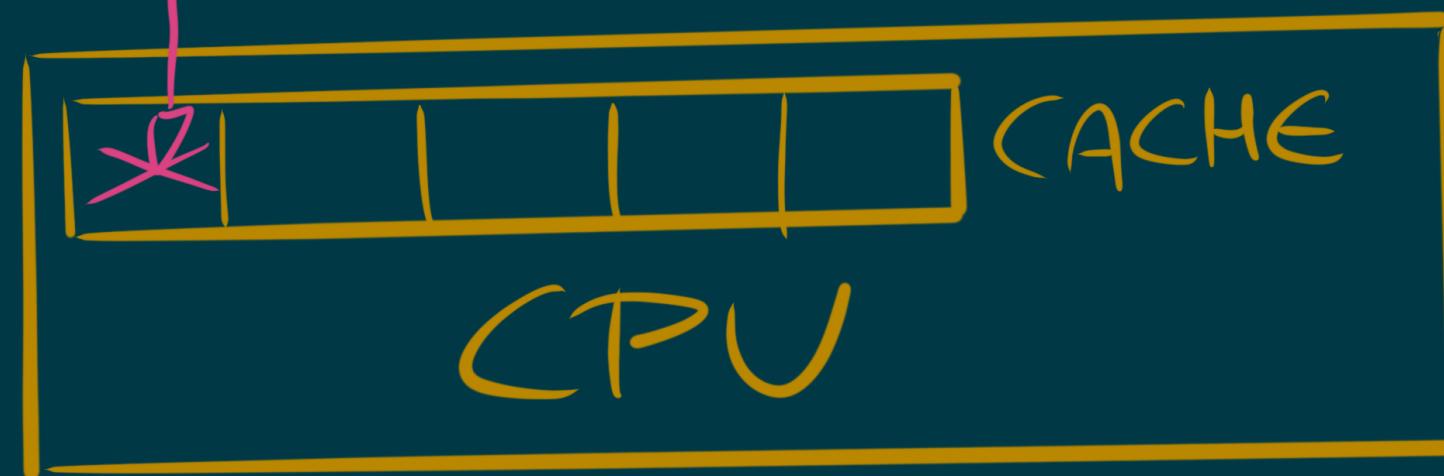
ROW



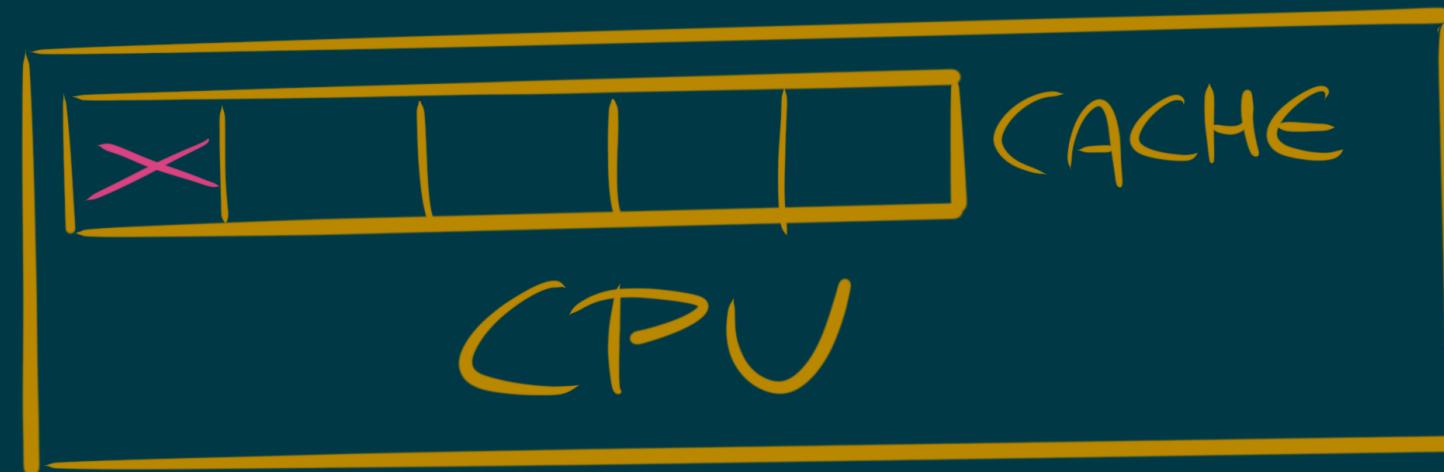
ROW



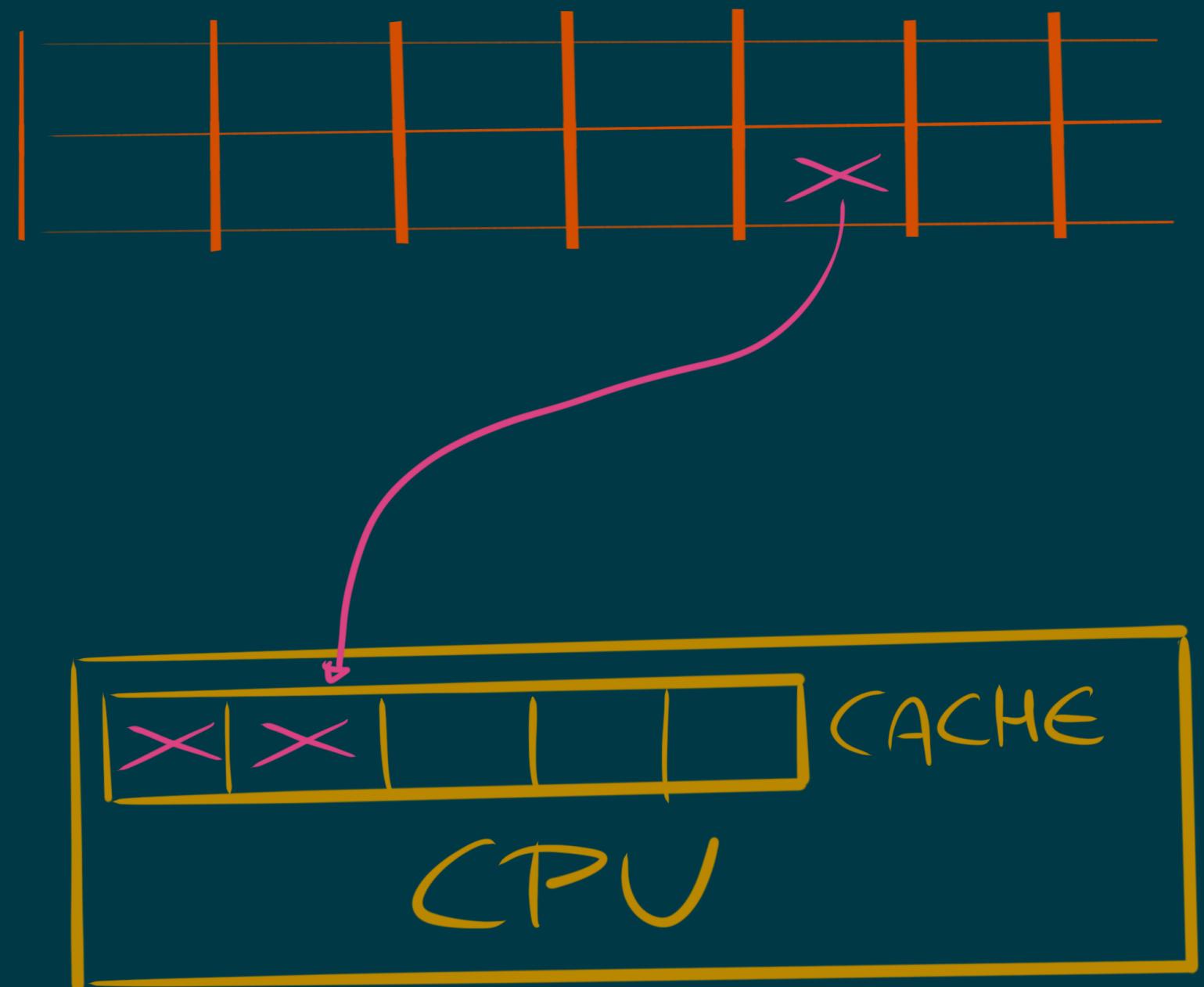
ROW

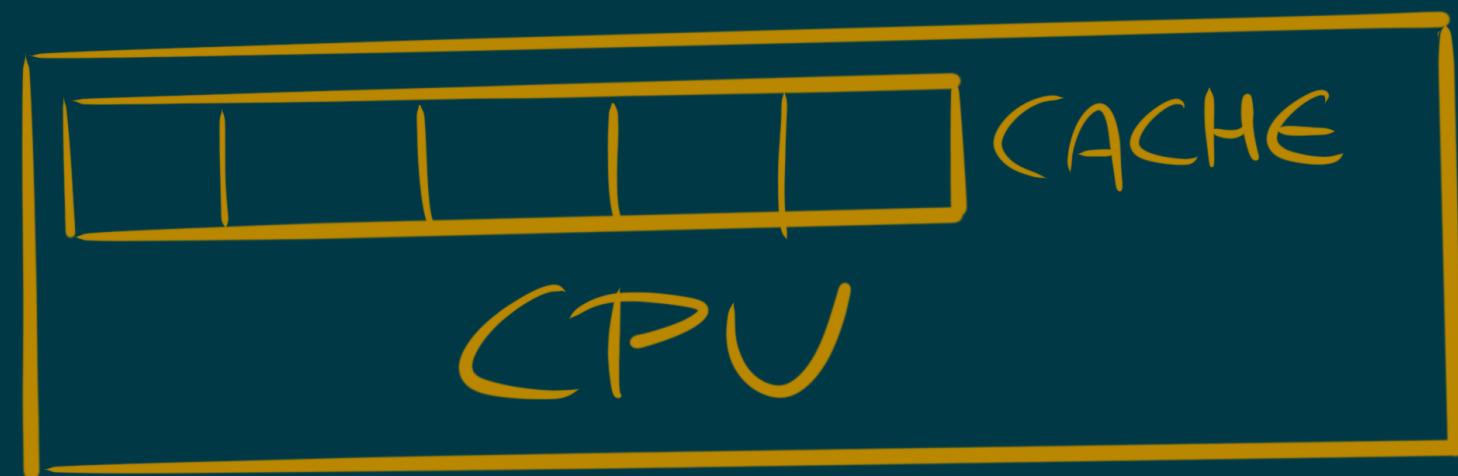


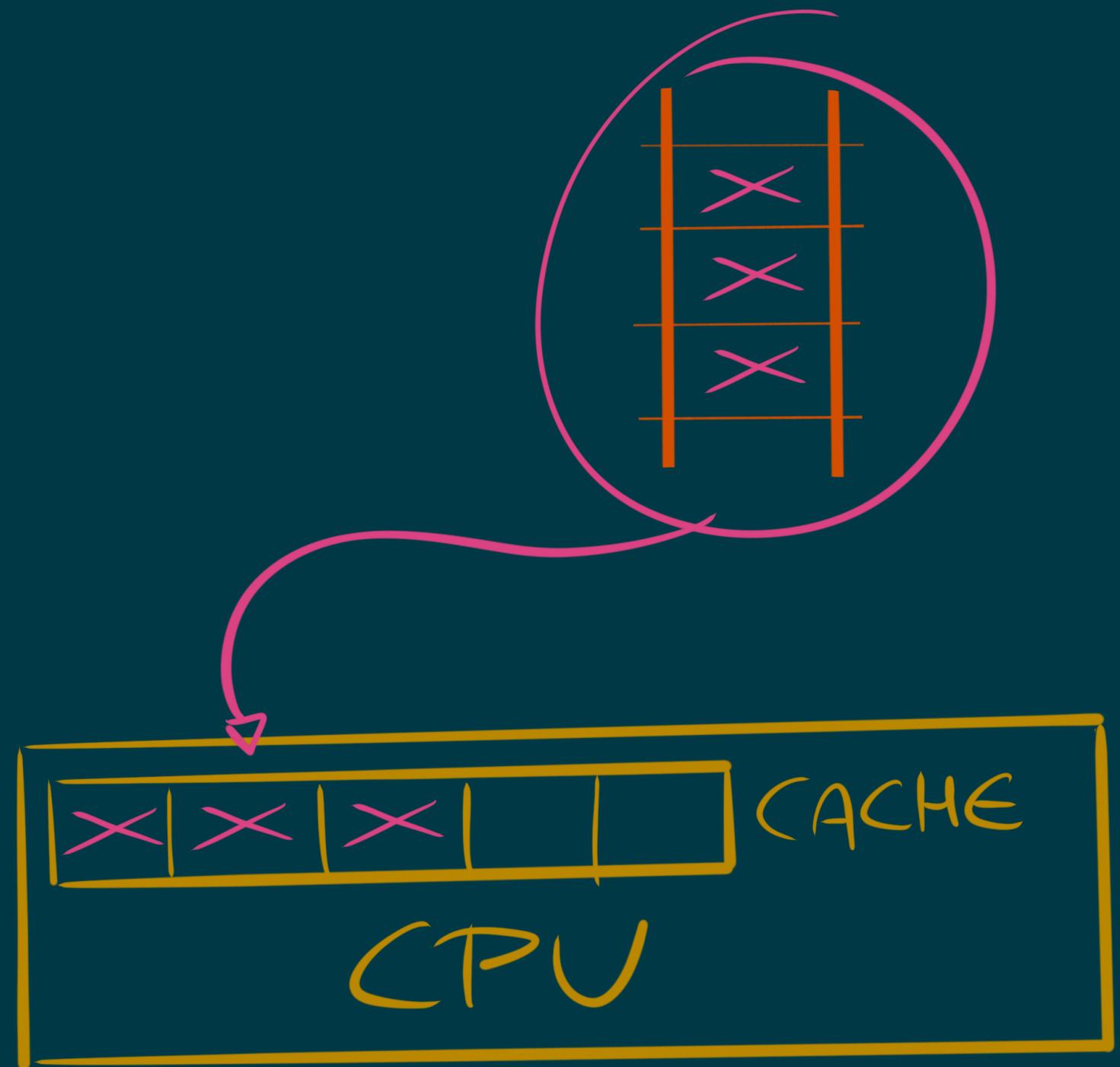
ROW



ROW







# WHAT IS ARROW?

# WHAT IS ARROW?

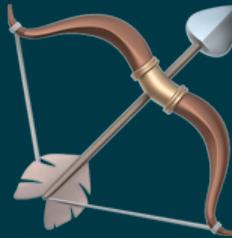
- > CROSS-LANGUAGE IN-MEMORY COLUMNAR FORMAT LIBRARY

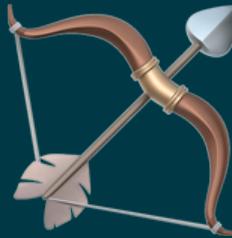
# WHAT IS ARROW?

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

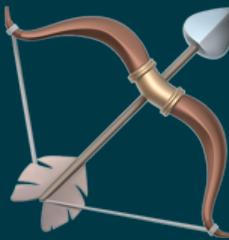
# WHAT IS ARROW?

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

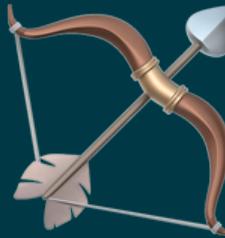




> ARROW USES RecordBatches



- > 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**



Cluster Manager

AND A  
**DISTRIBUTED  
STORAGE**



Cluster Manager

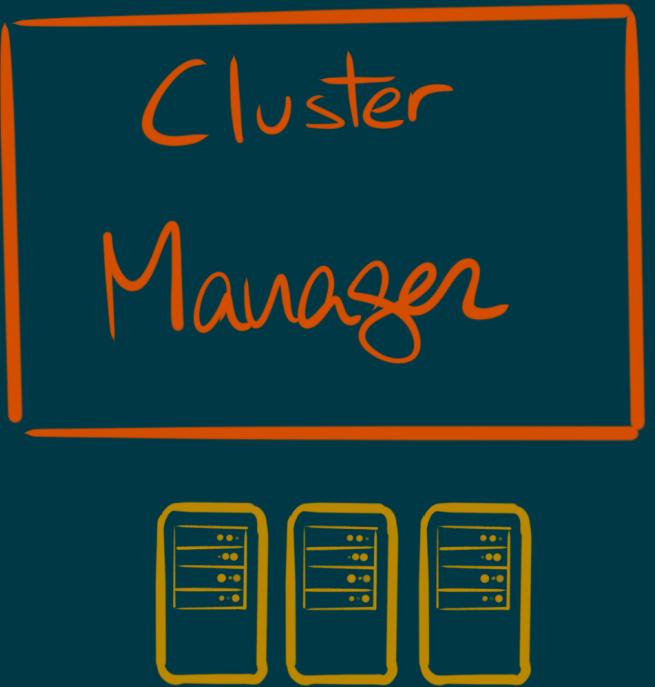
Distributed Storage



A SPARK PROGRAM  
RUNS IN THE DRIVER

THE DRIVER REQUESTS  
RESOURCES FROM THE  
CLUSTER MANAGER TO  
RUN TASKS

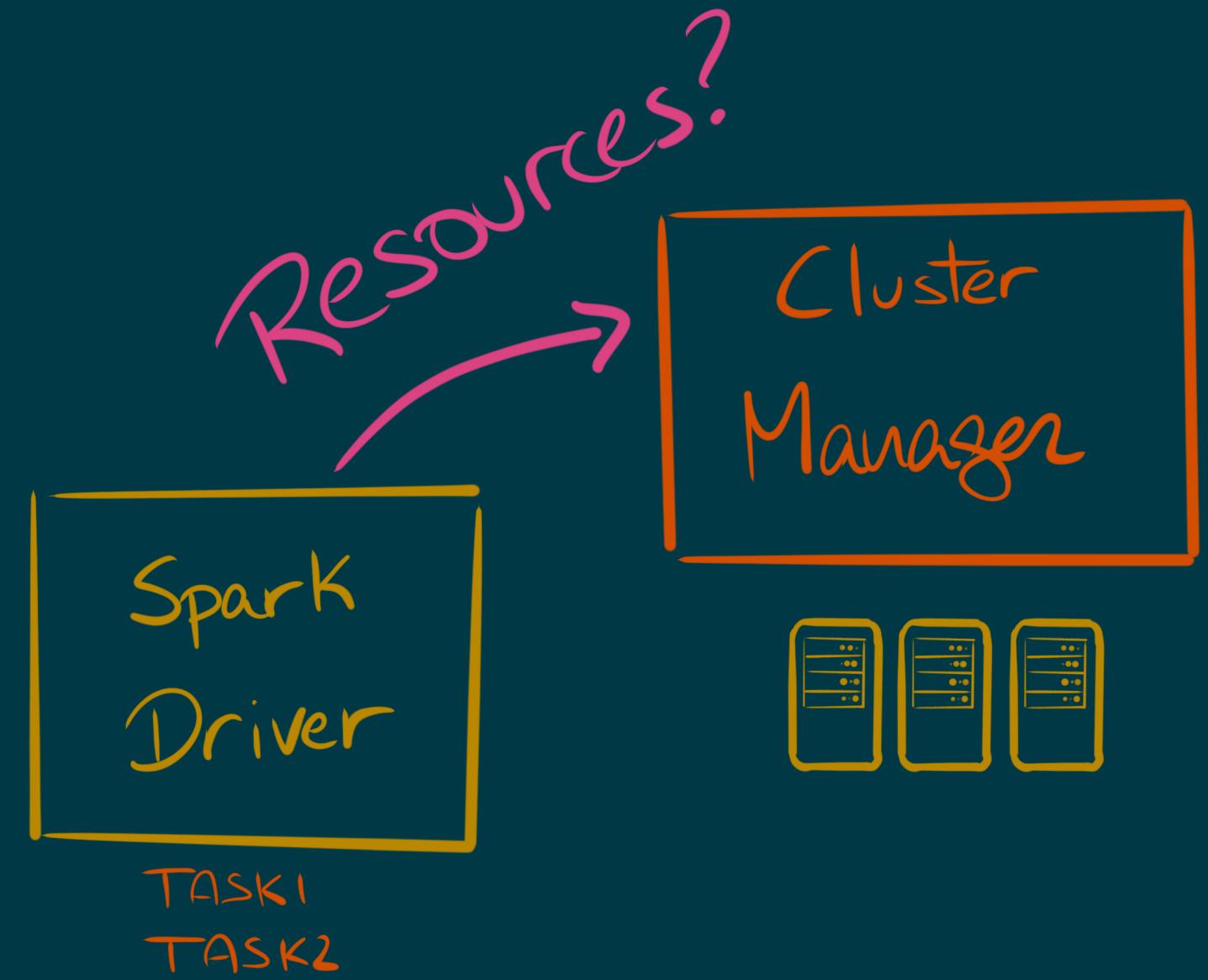
Spark  
Driver



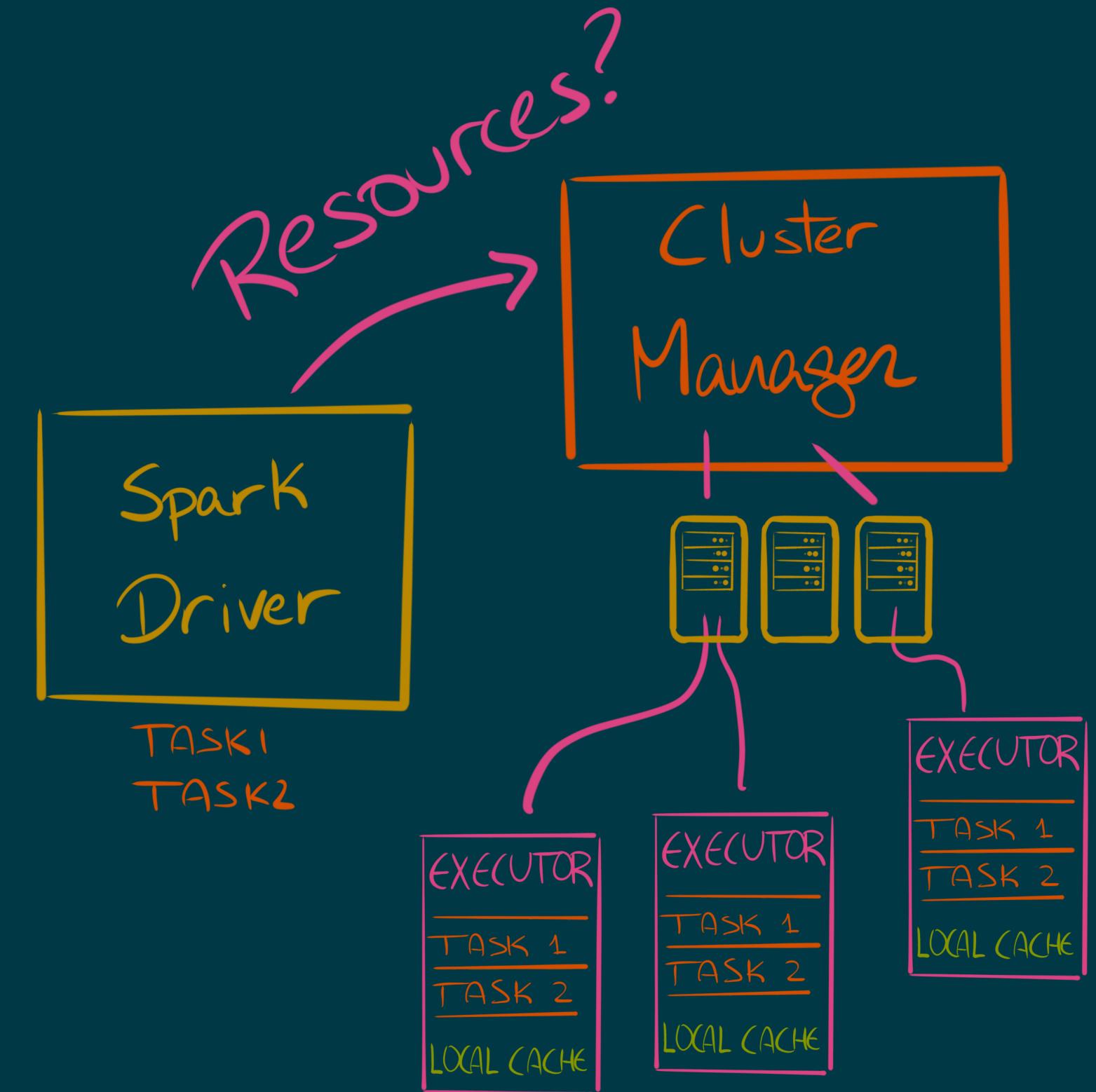
# THE DRIVER REQUESTS RESOURCES FROM THE CLUSTER MANAGER TO RUN TASKS



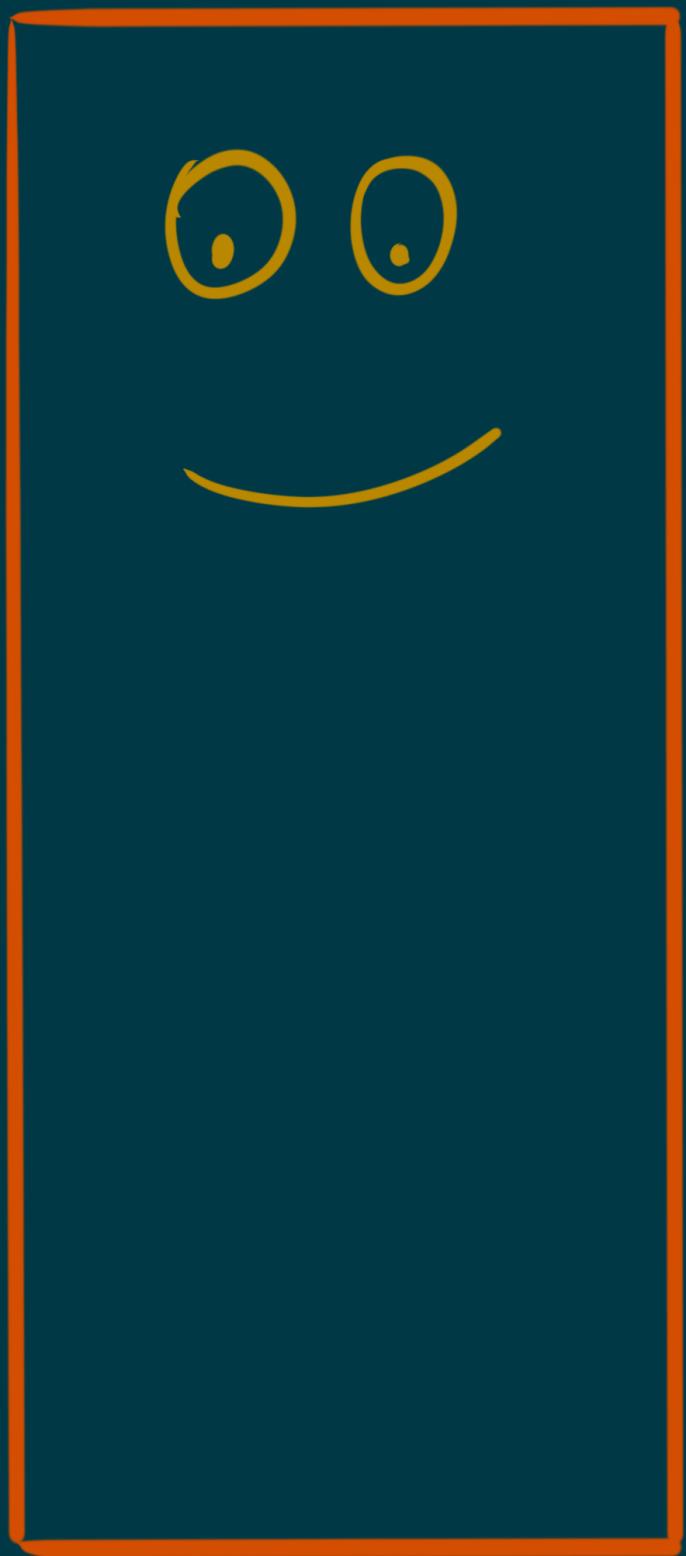
THE DRIVER REQUESTS  
RESOURCES FROM THE  
CLUSTER MANAGER TO  
RUN TASKS



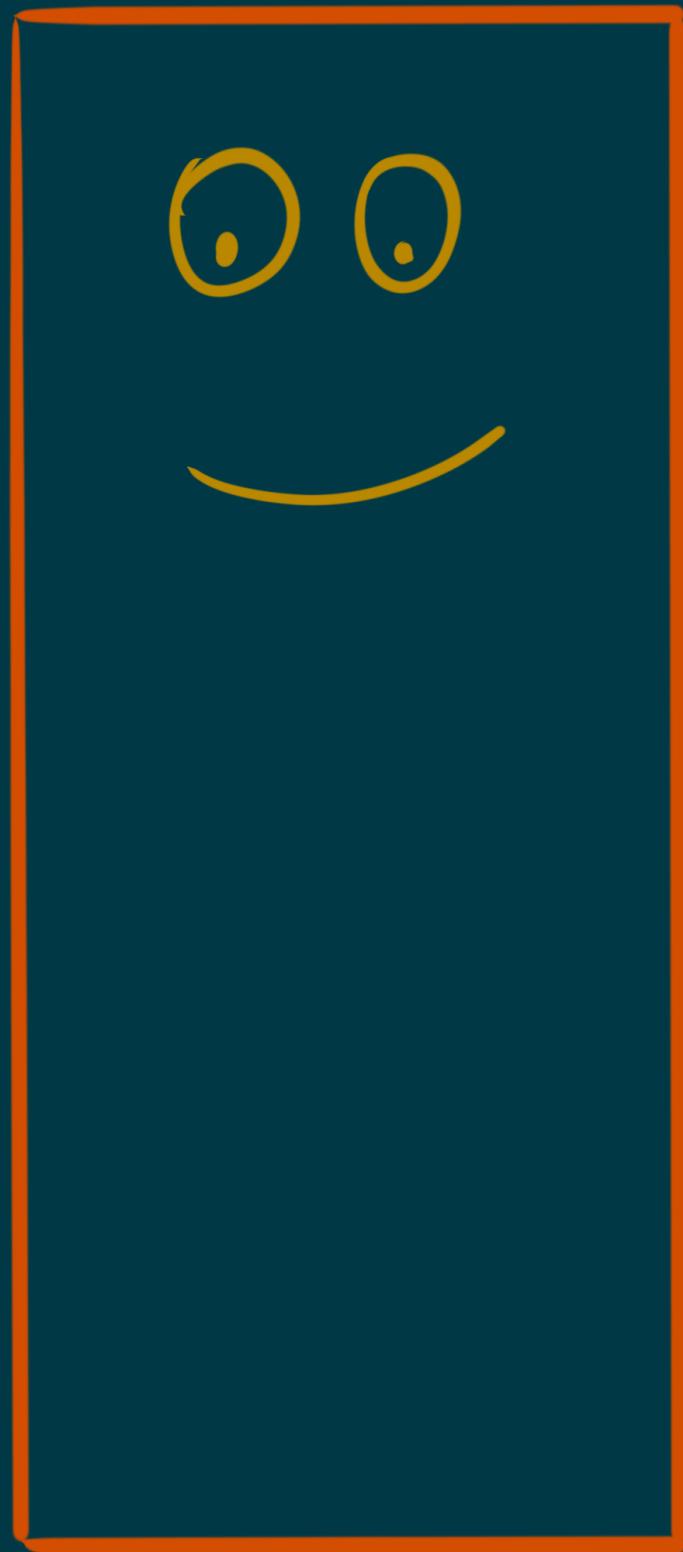
THE DRIVER REQUESTS  
RESOURCES FROM THE  
CLUSTER MANAGER TO  
RUN TASKS



THE MAIN BUILDING BLOCK  
IS THE RDD:  
RESILIENT DISTRIBUTED  
DATASET



RDD



RDD



RDD



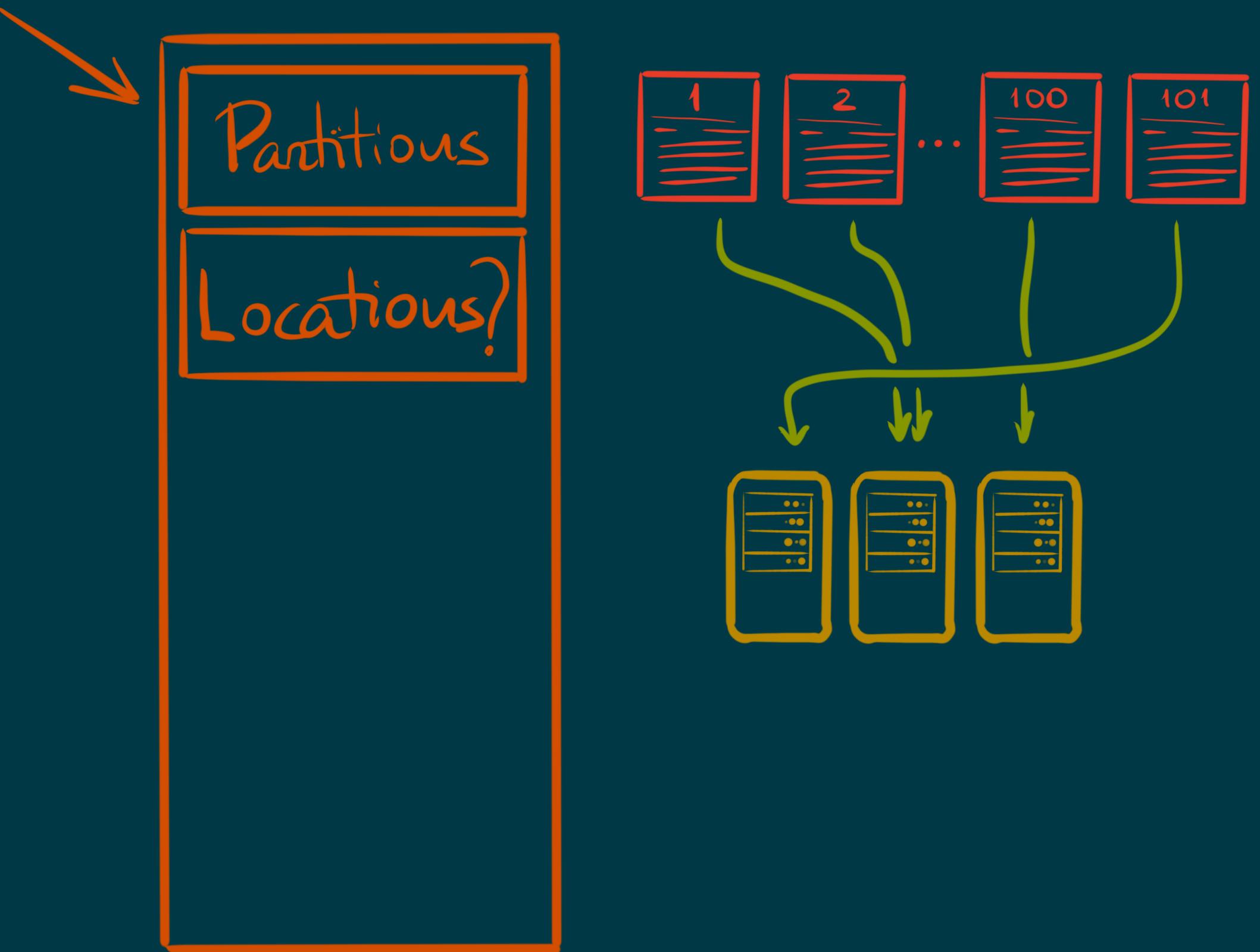
Partitions



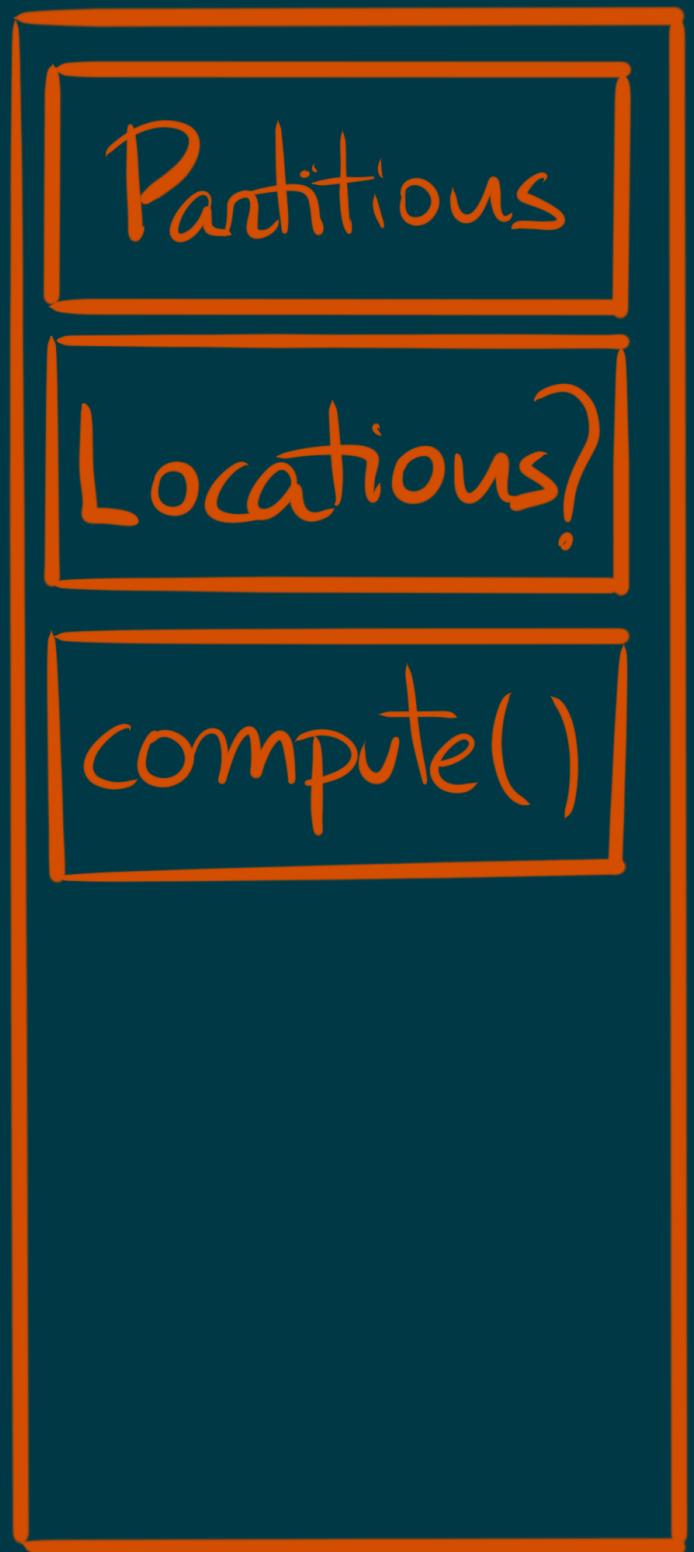
RDD



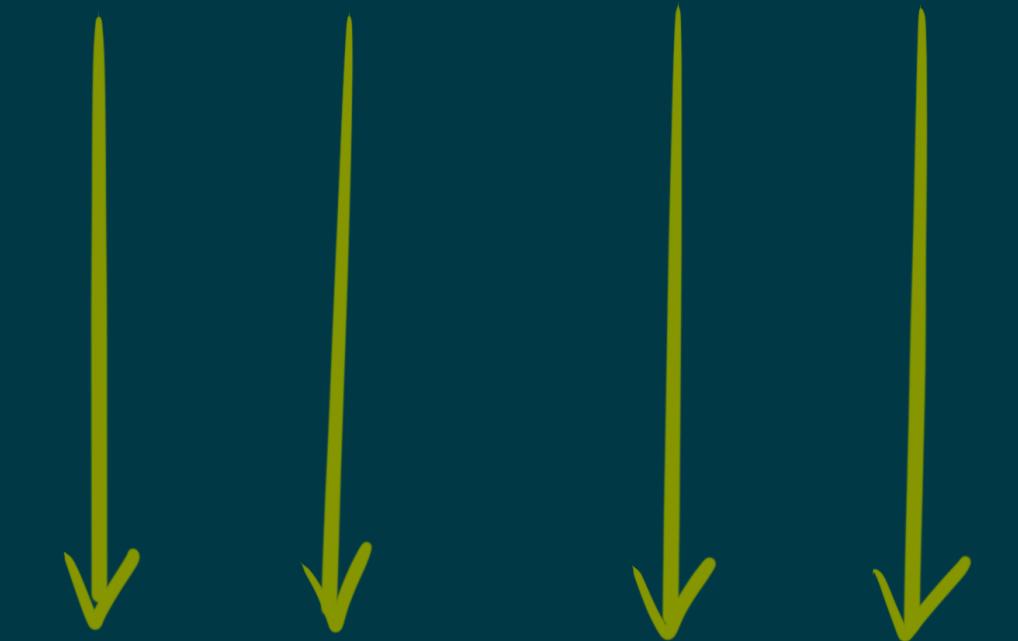
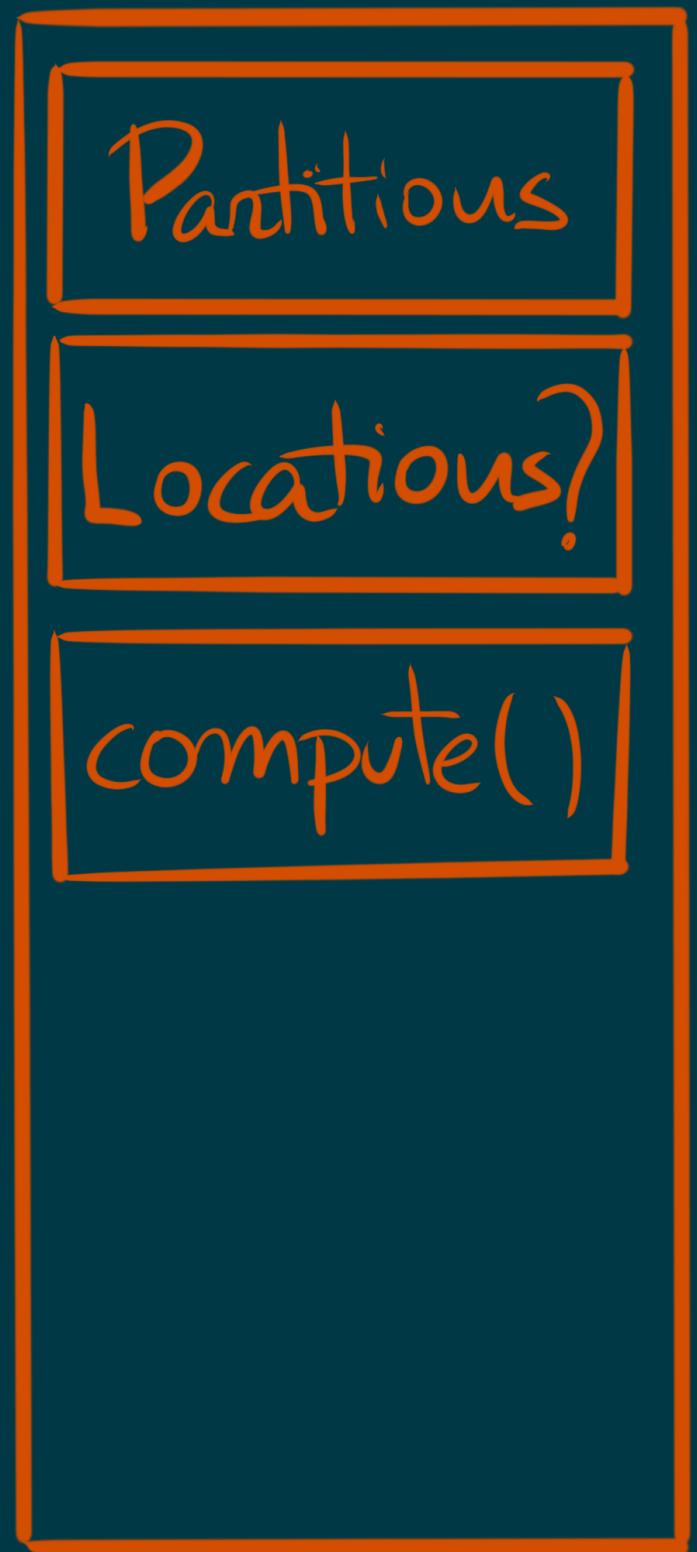
RDD



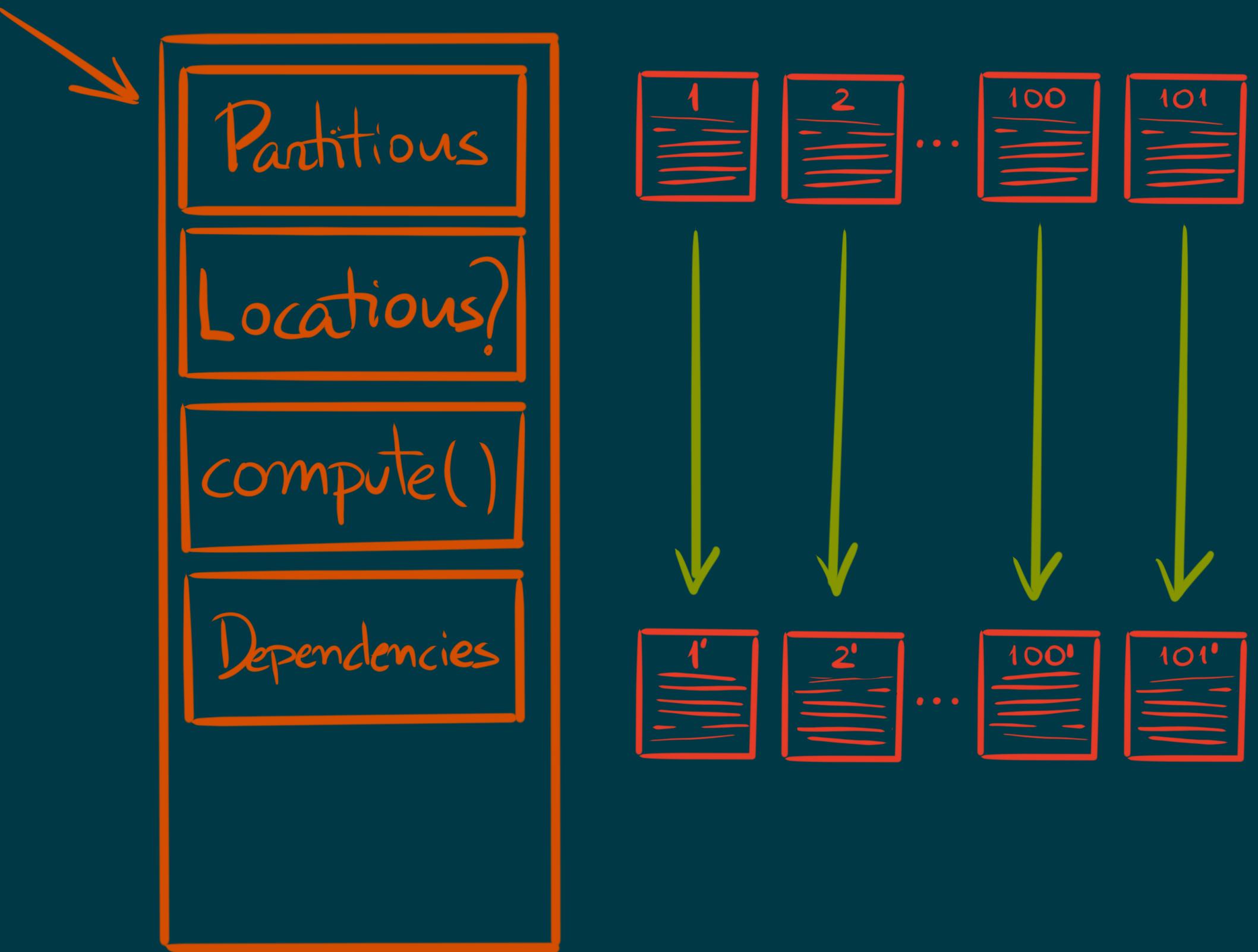
RDD



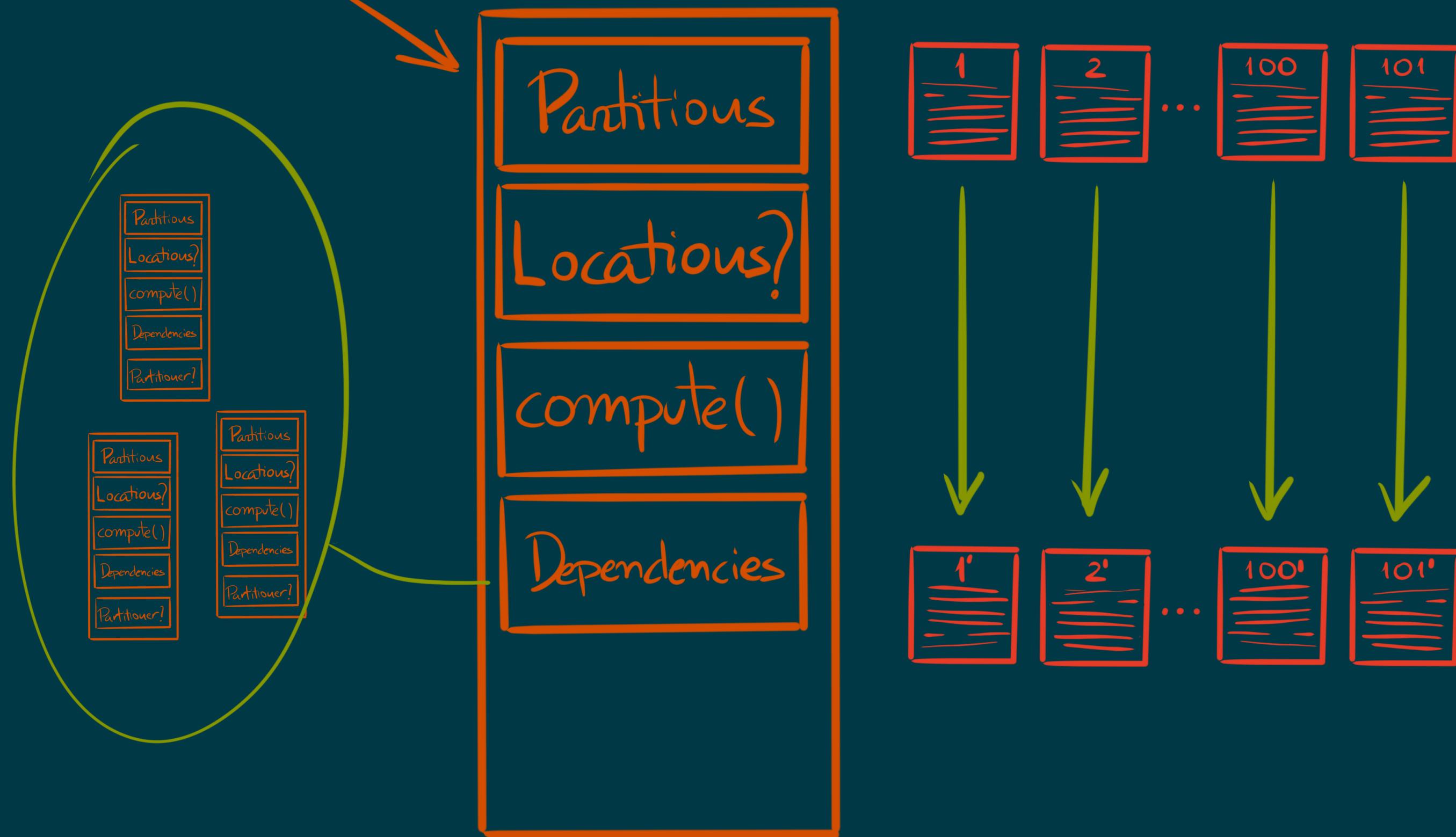
RDD



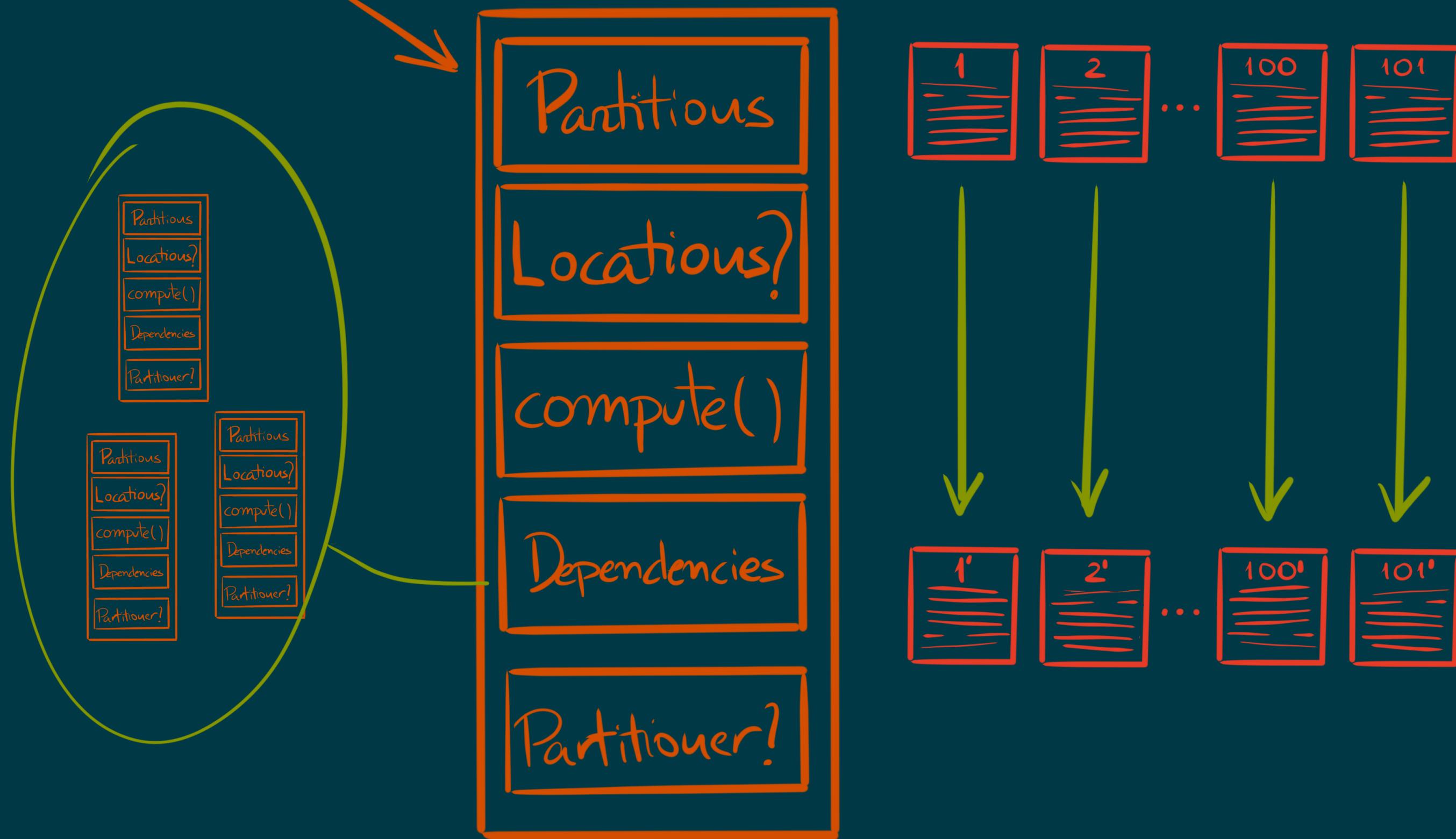
RDD



# RDD



# RDD



# 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



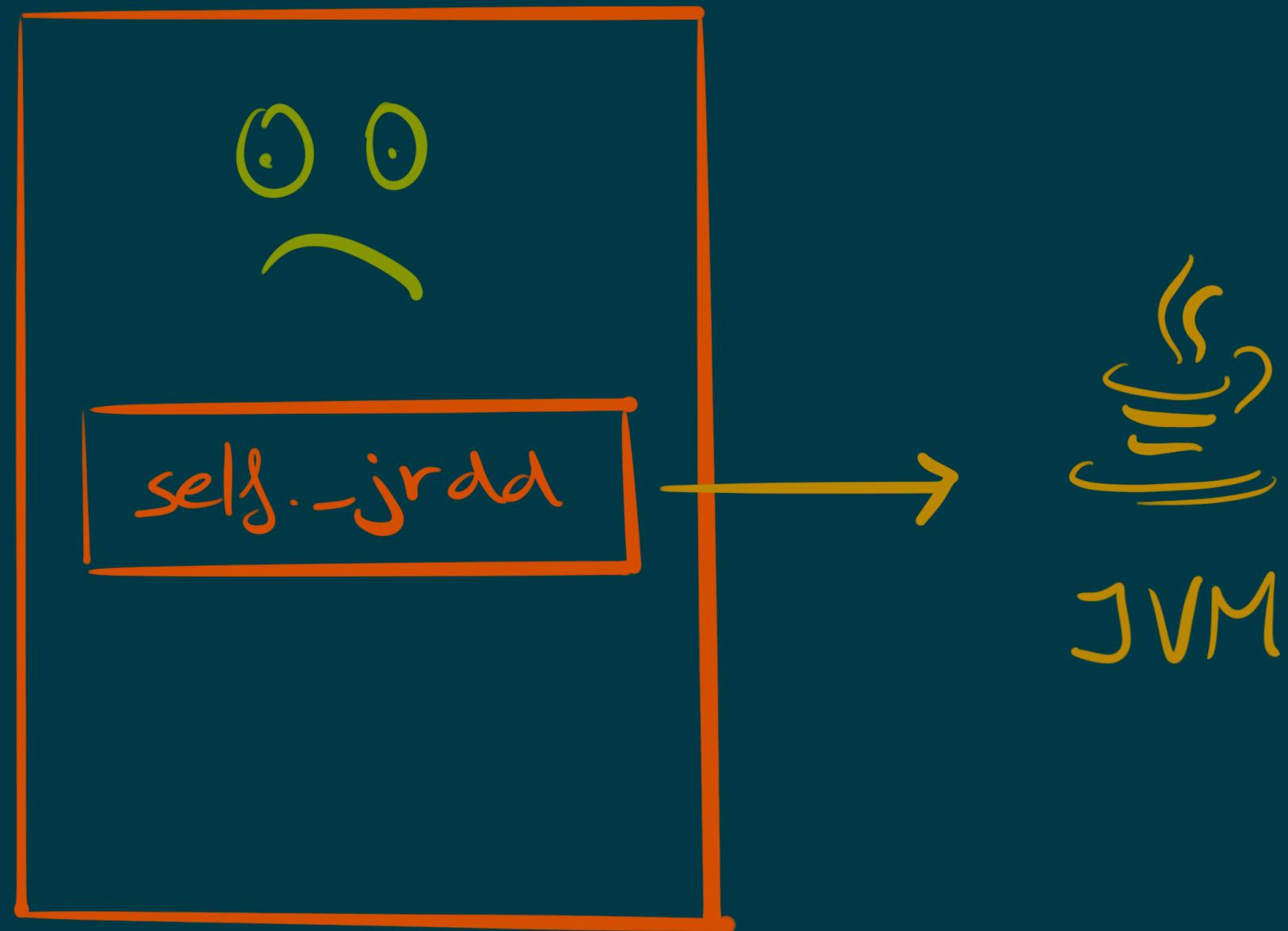
M  
is for  
murder

RDD in  
Python Land



j  
is for  
java

RDD in  
Python Land



RDD in  
Python Land



Py4J bridge  
JVM

THE MAIN ENTRYPONTS  
ARE RDD AND  
PipelinedRDD(RDD)

PipelinedRDD  
BUILDS IN THE JVM A  
PythonRDD





RDD

j RDD

RDD

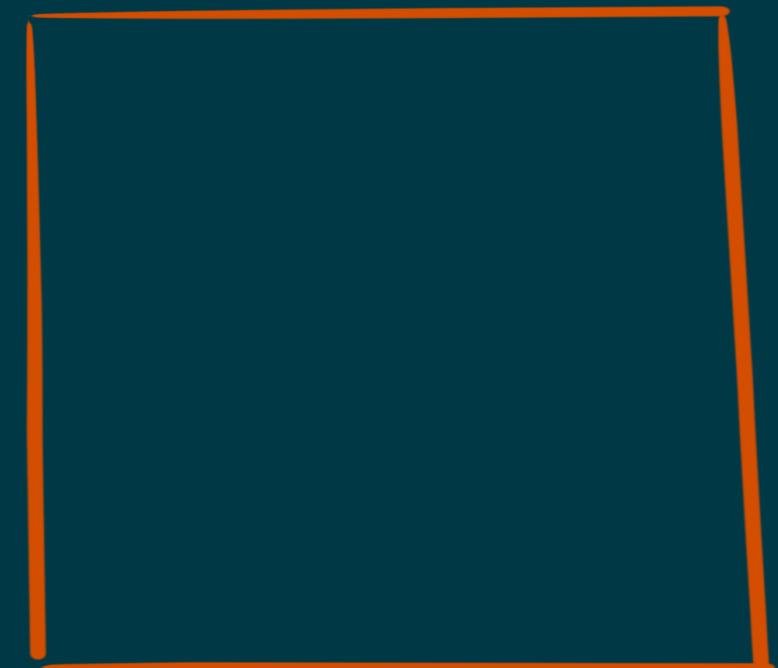
jRDD

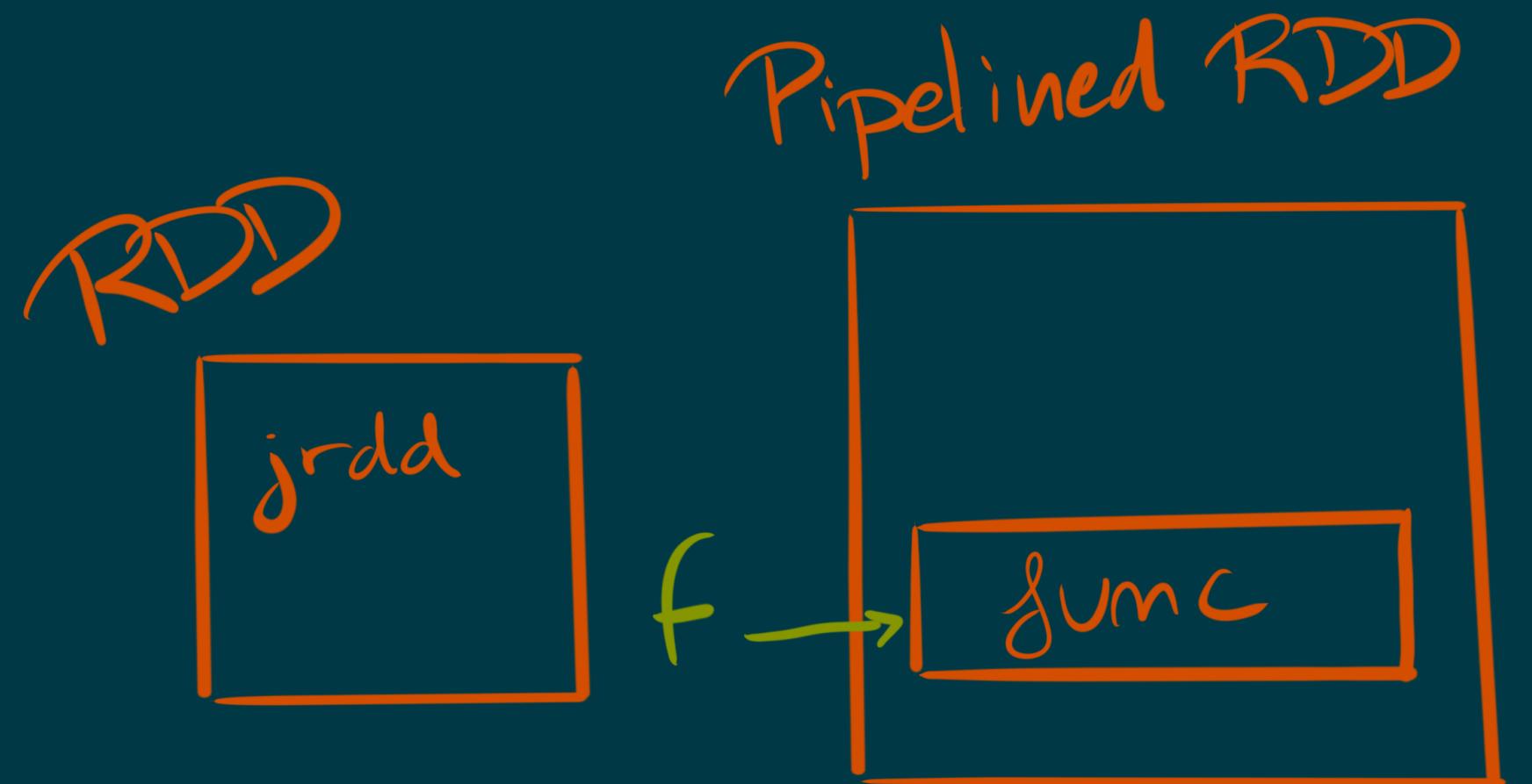
MAP(f)

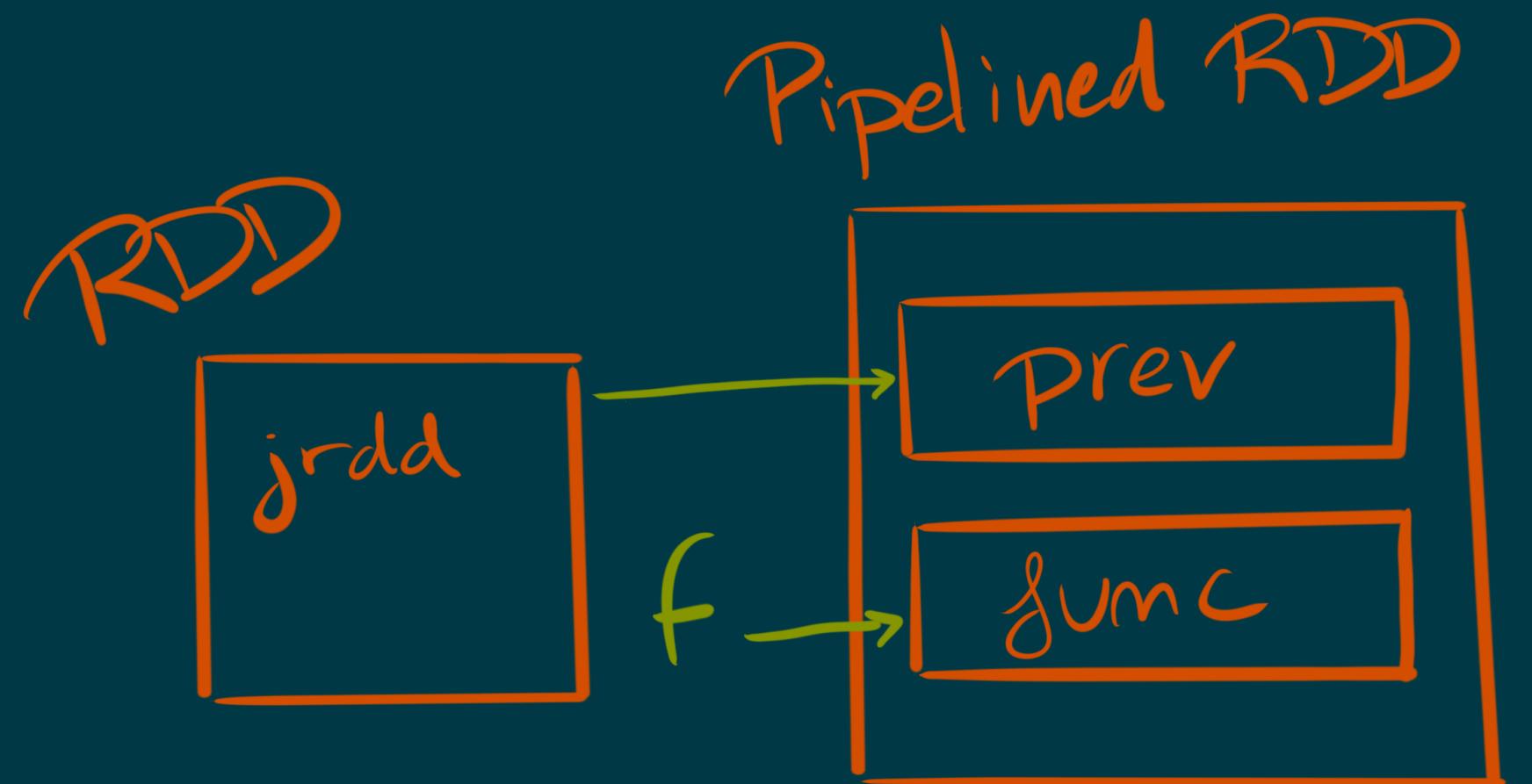
RDD

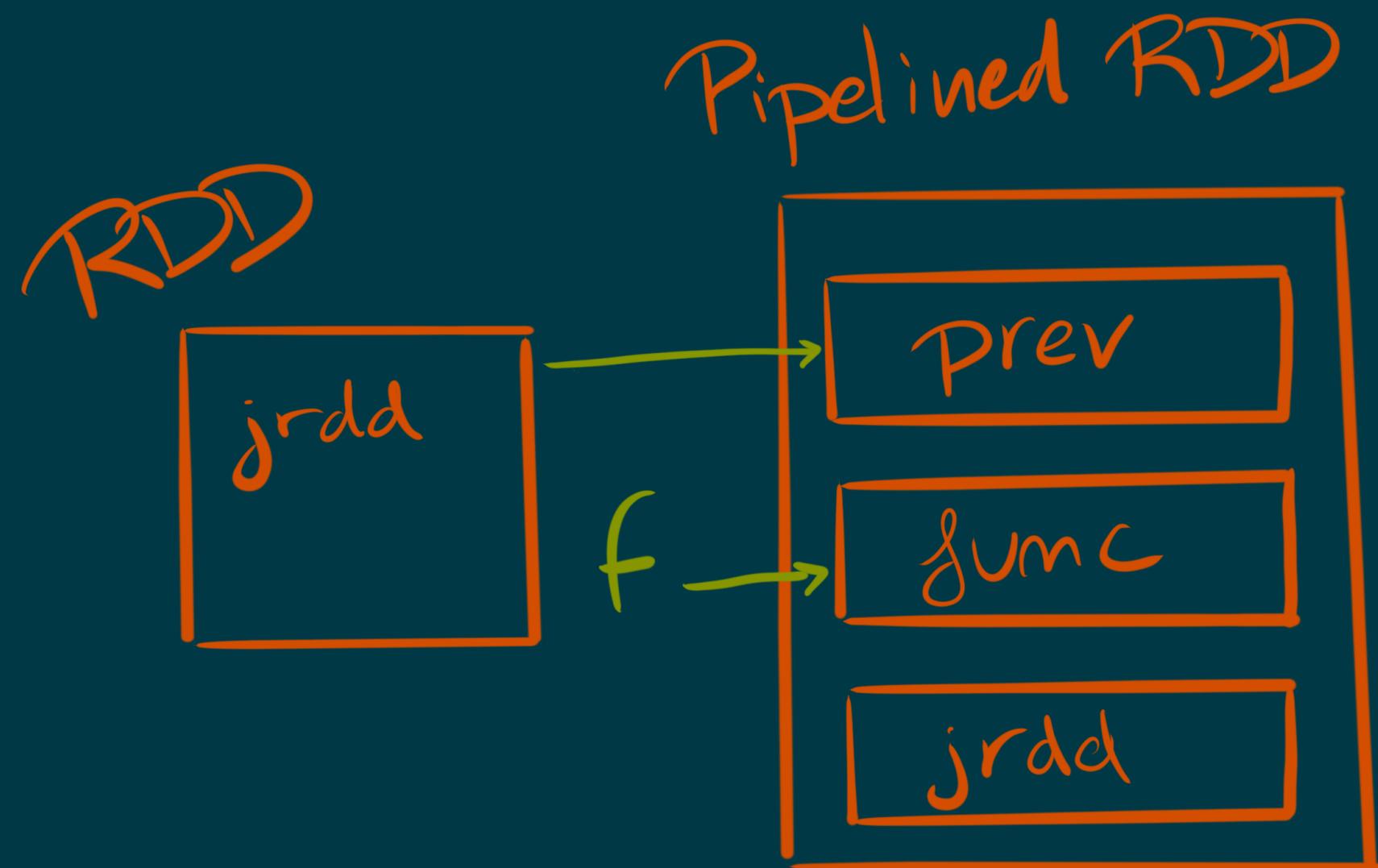


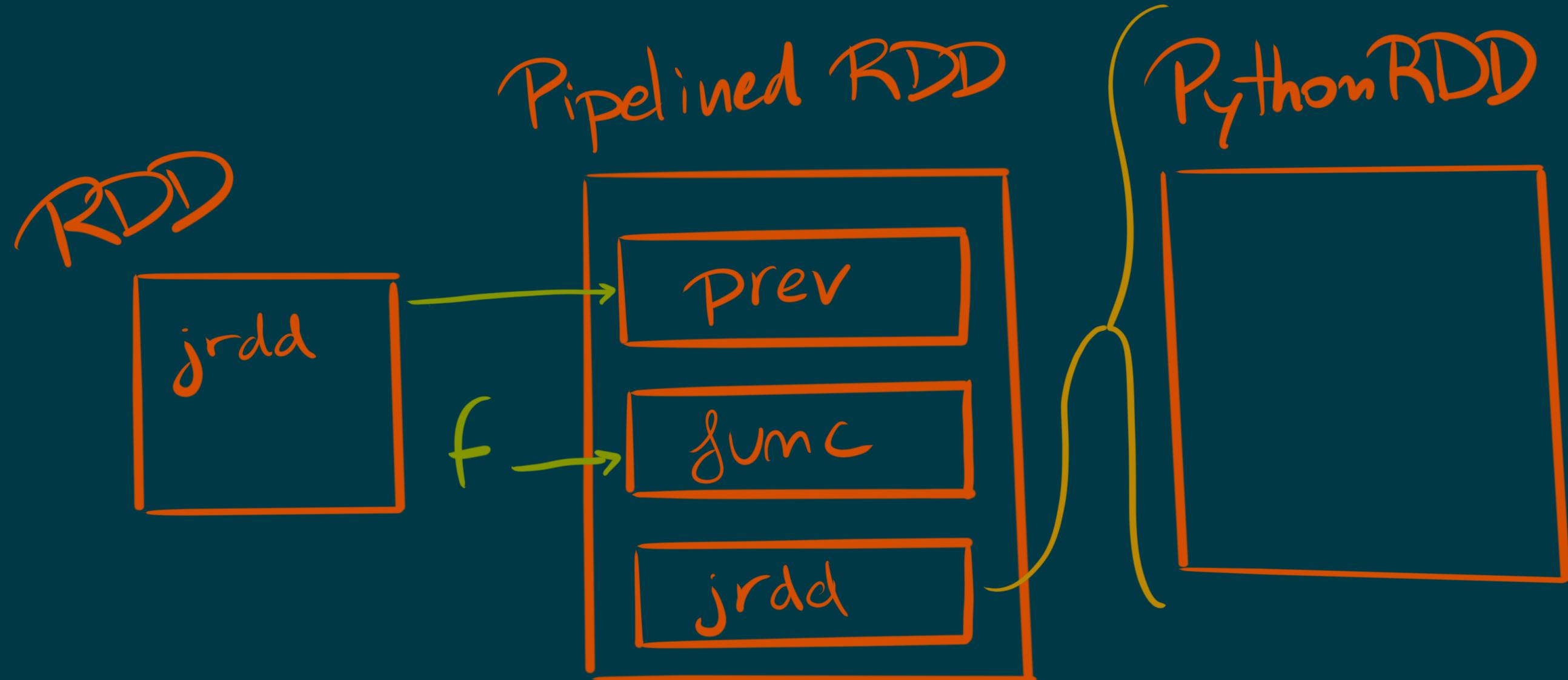
Pipelined RDD

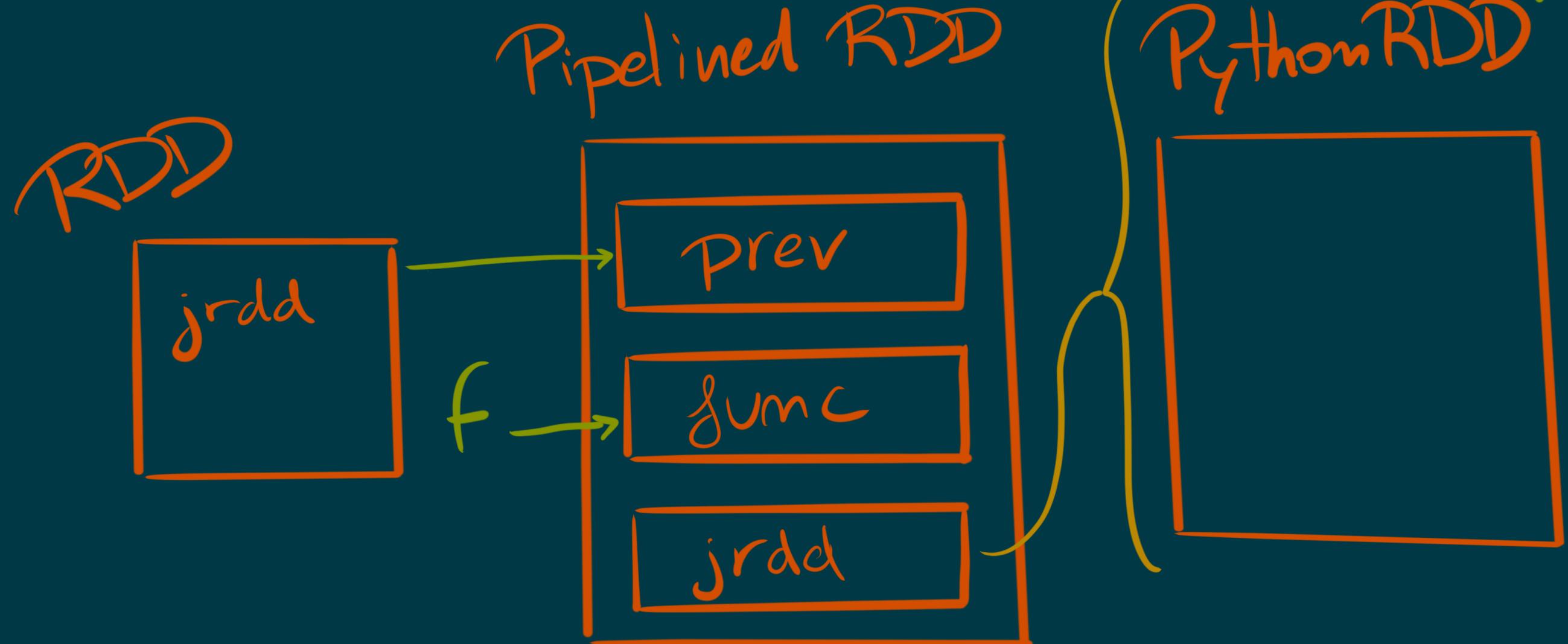


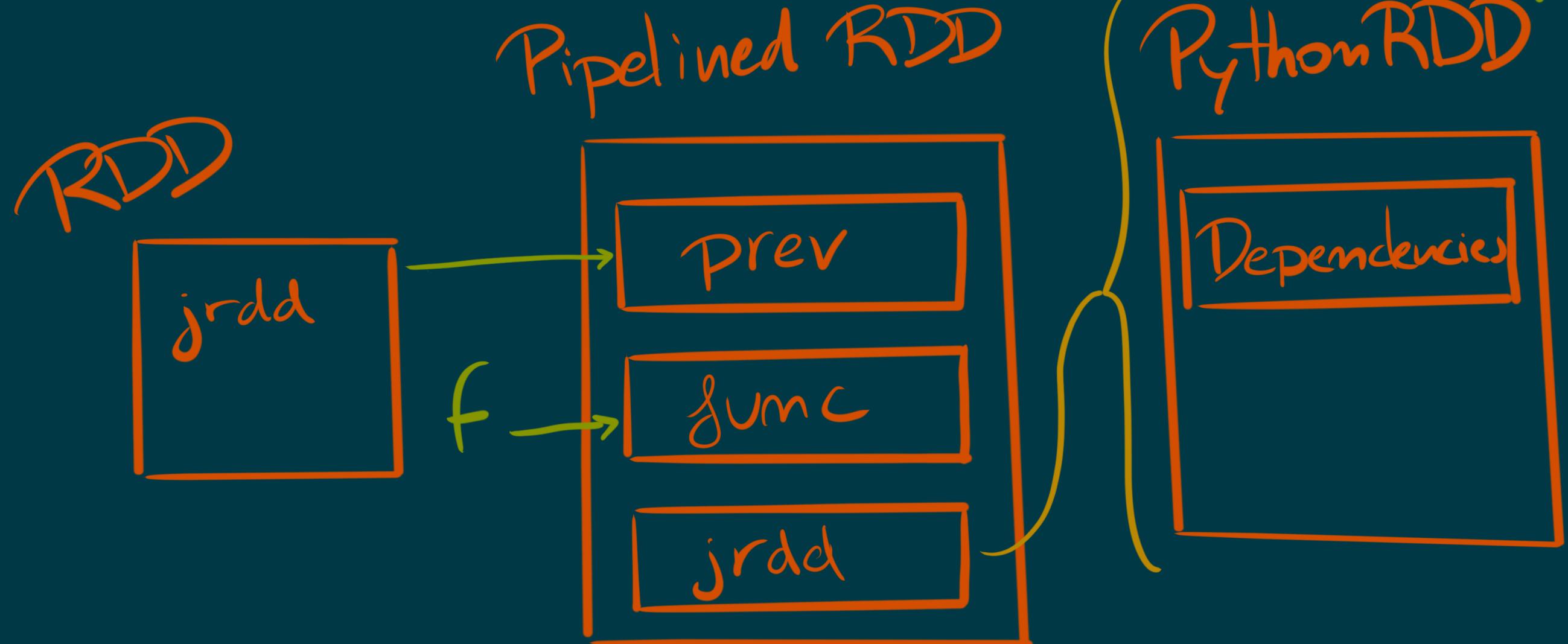






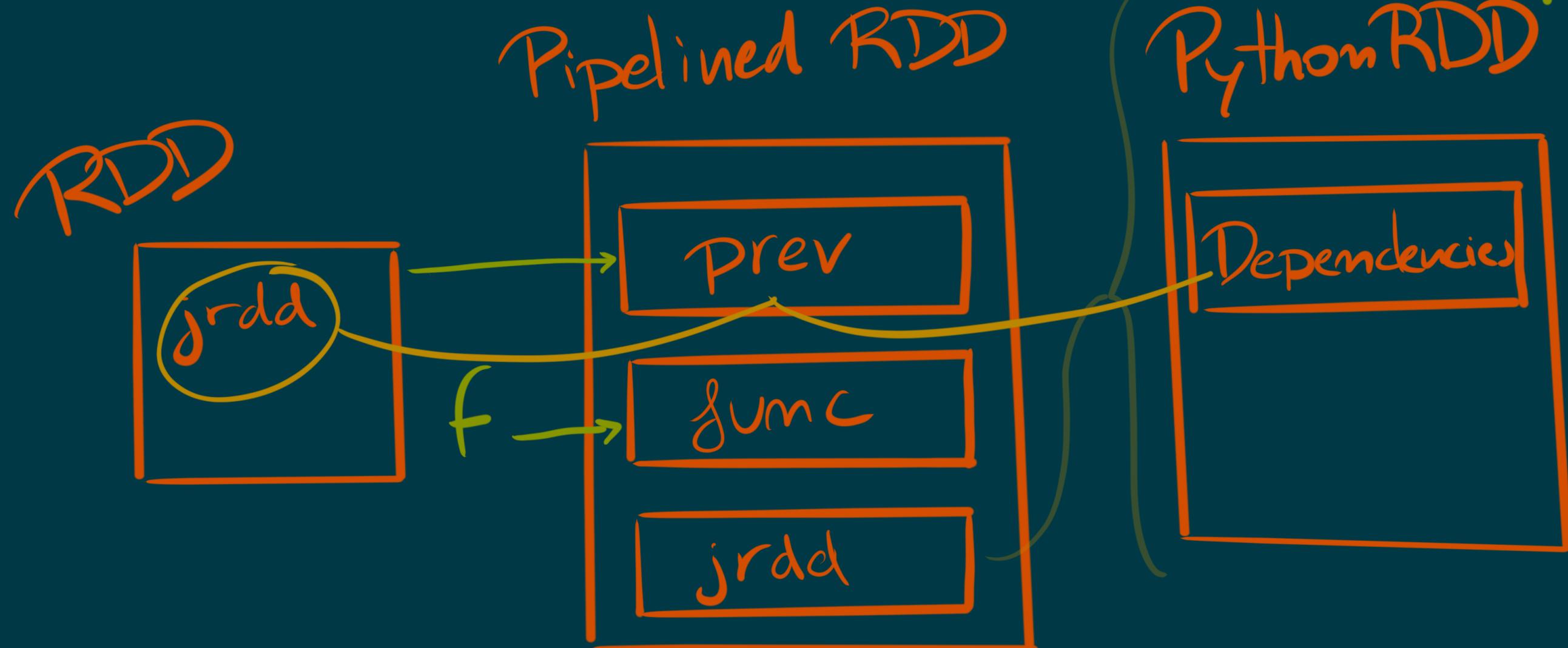






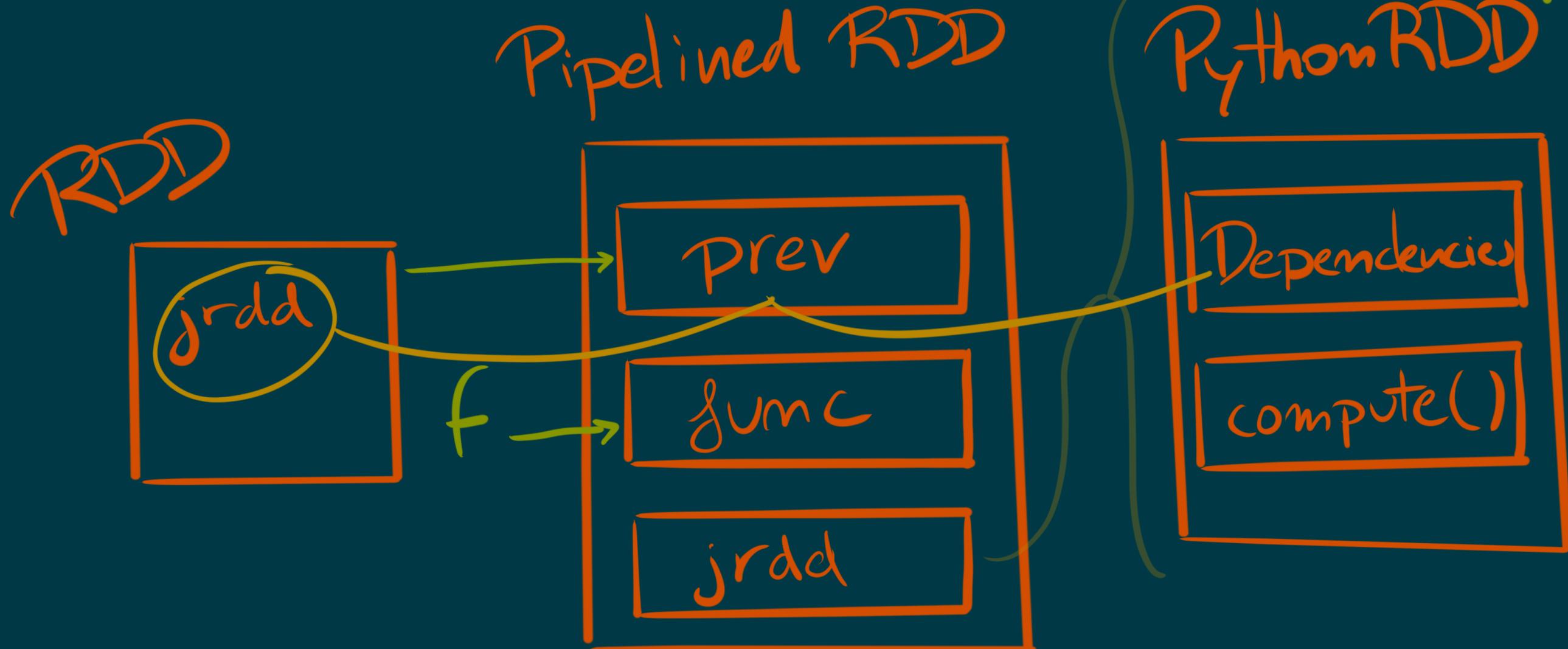
class in Scala

PythonRDD



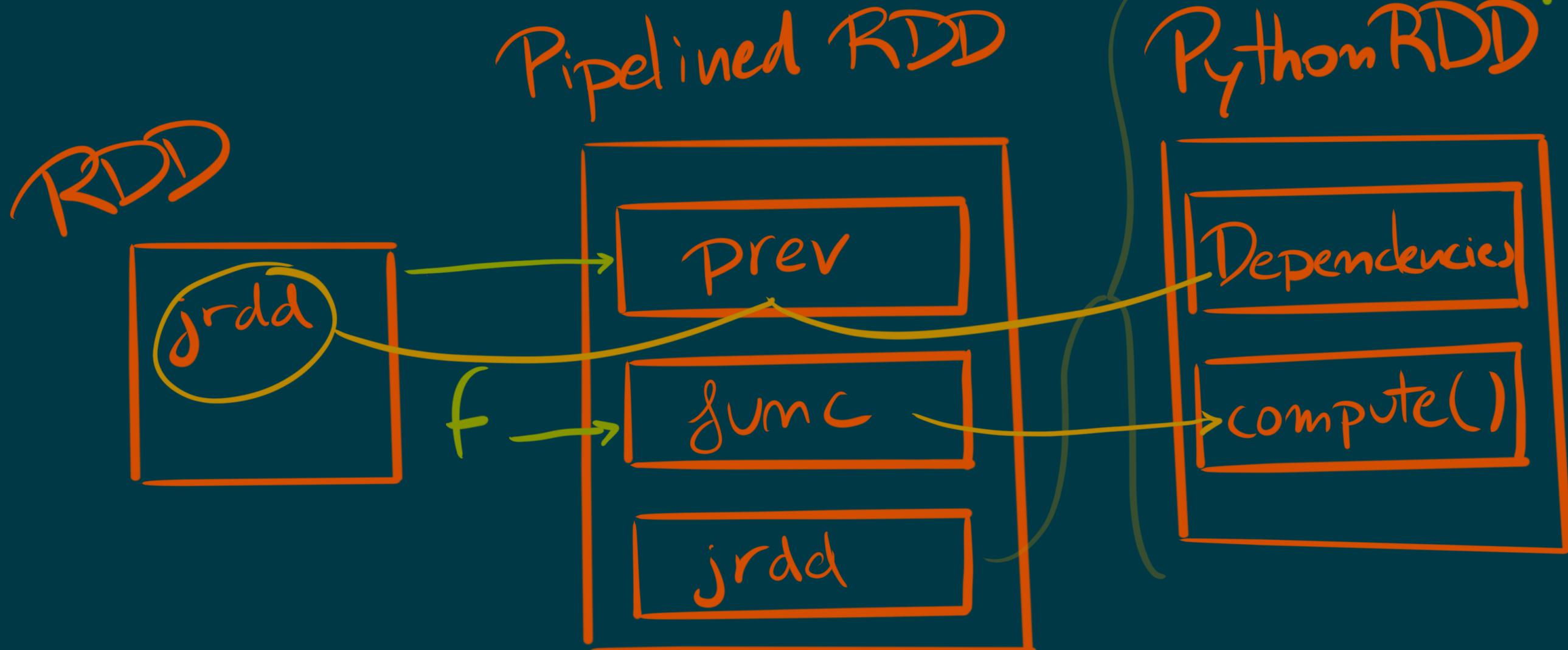
class in Scala

PythonRDD



class in Scala

PythonRDD





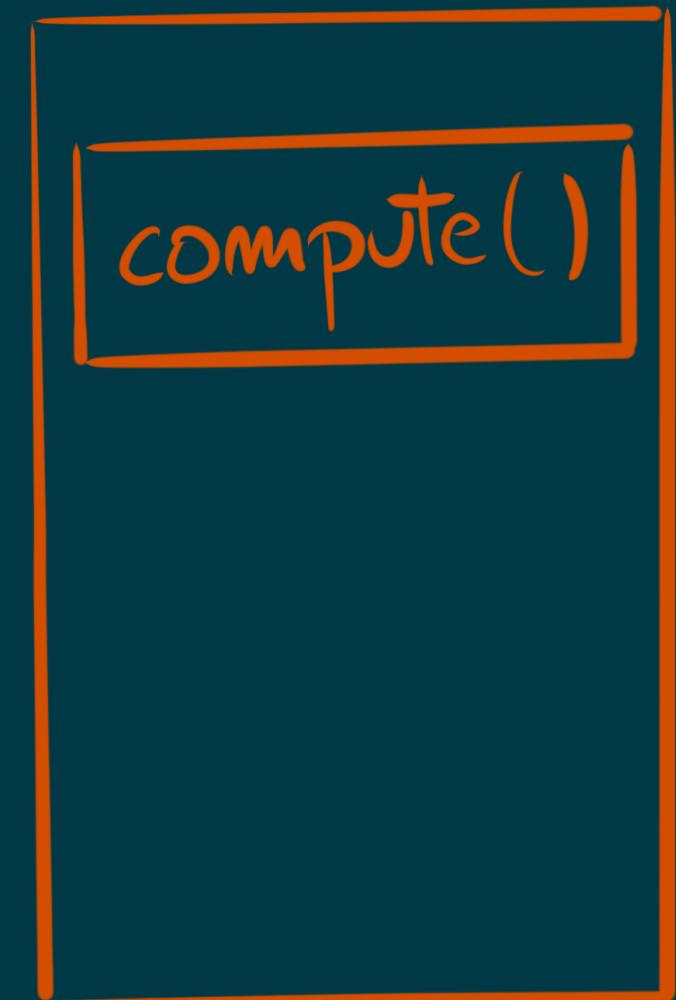
THE MAGIC IS  
IN  
compute

compute  
IS RUN ON EACH  
**EXECUTOR** AND STARTS  
A PYTHON **WORKER** VIA  
PythonRunner



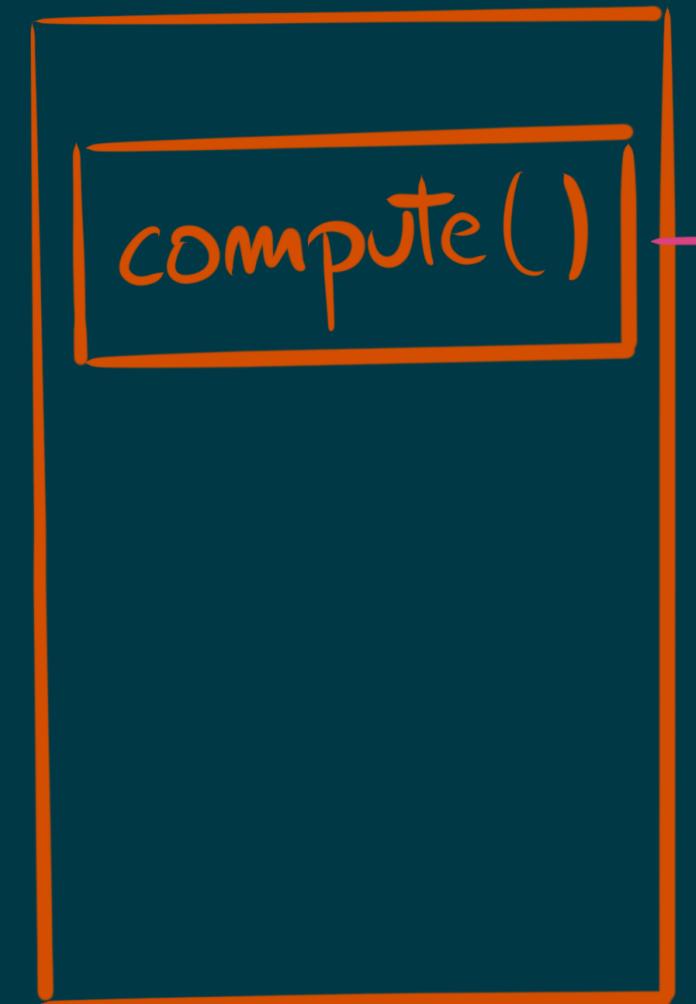


Python RDD

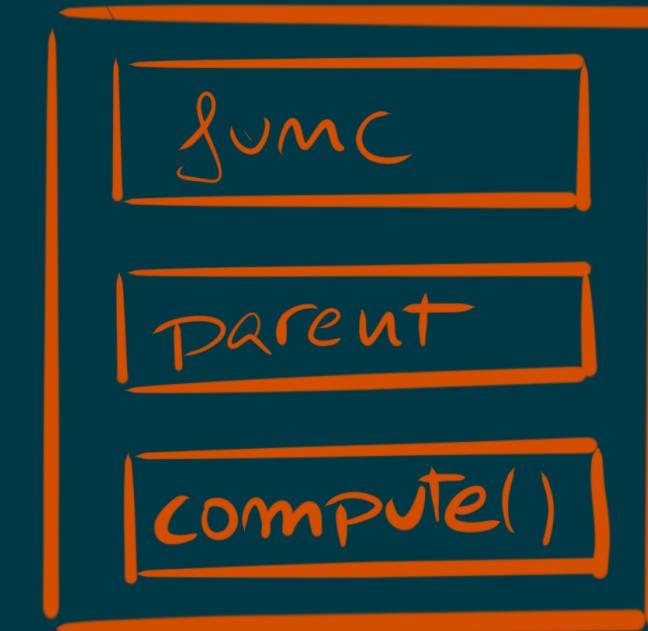


Scala!

PythonRDD

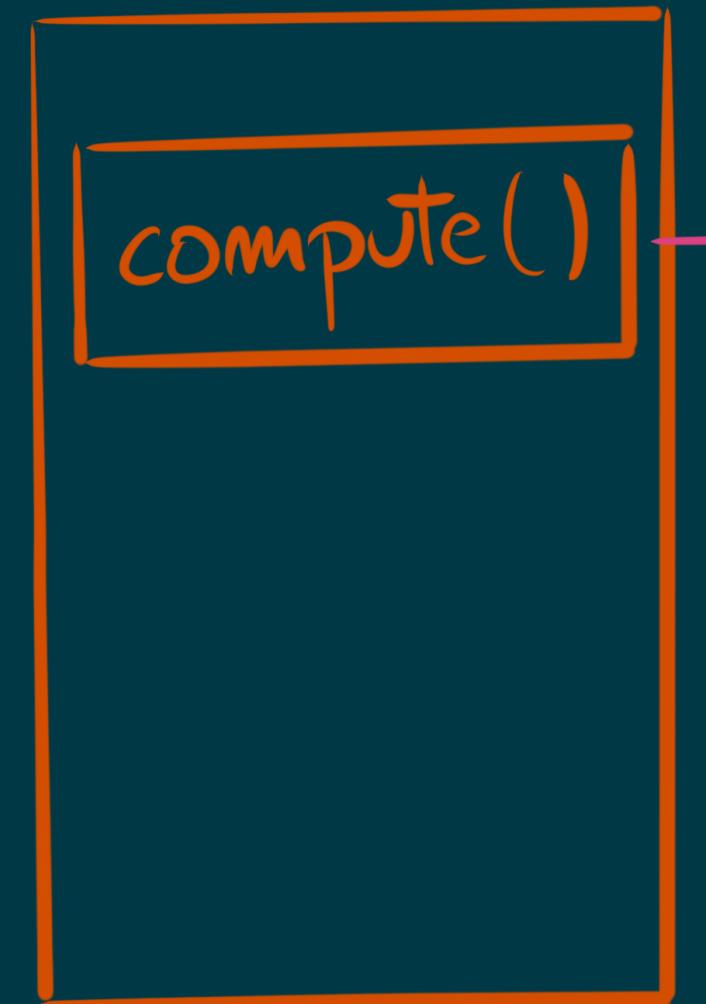


PythonRunner

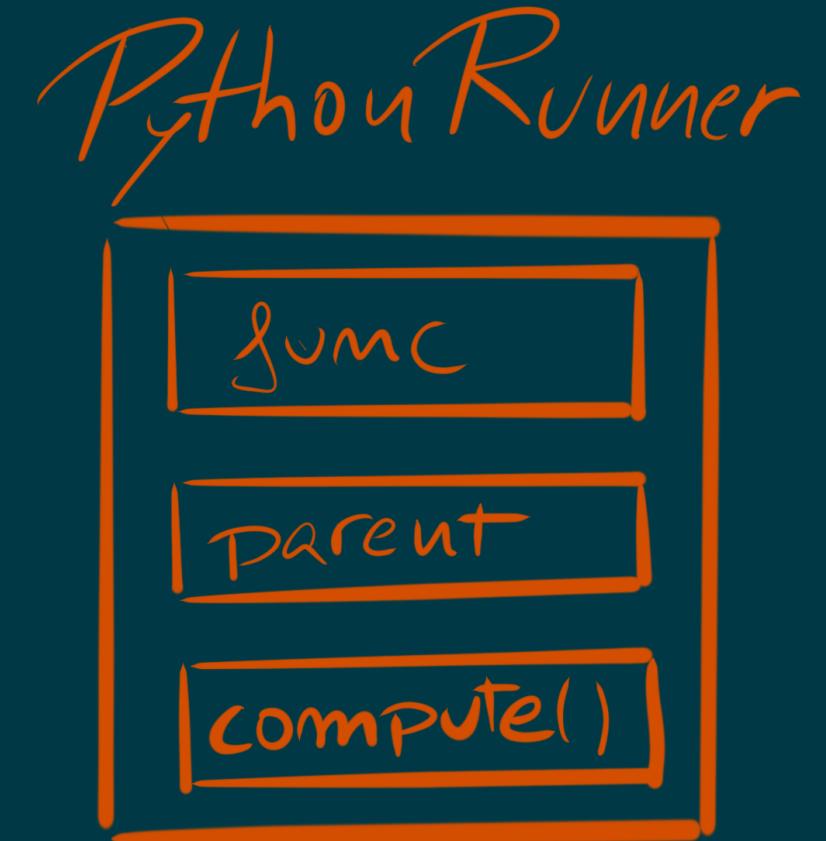


Scala!

PythonRDD



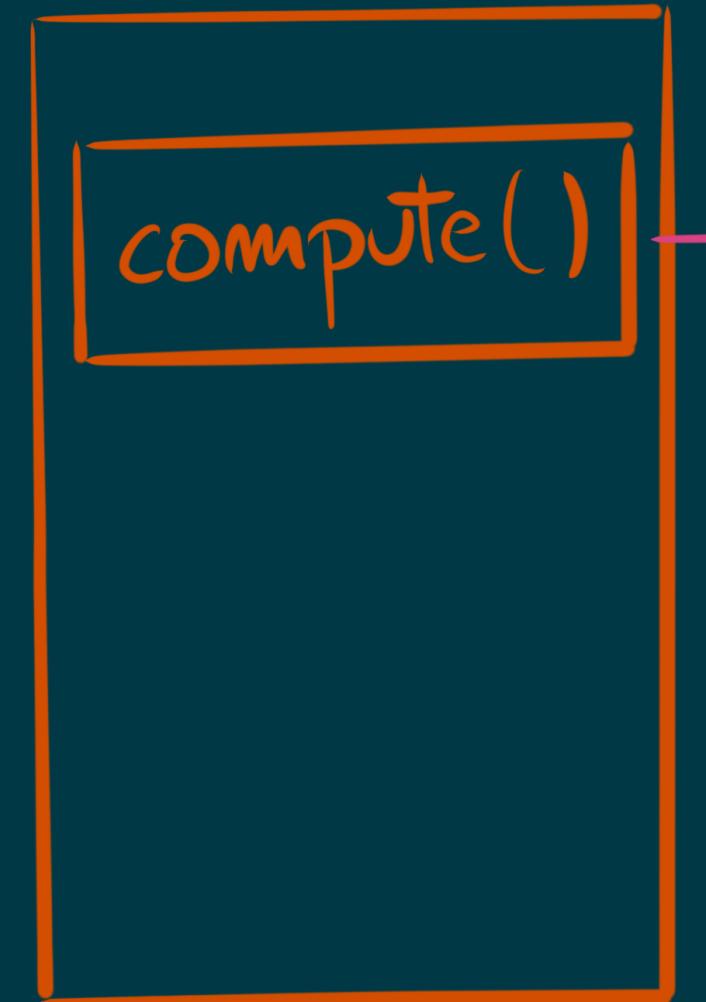
Scala!



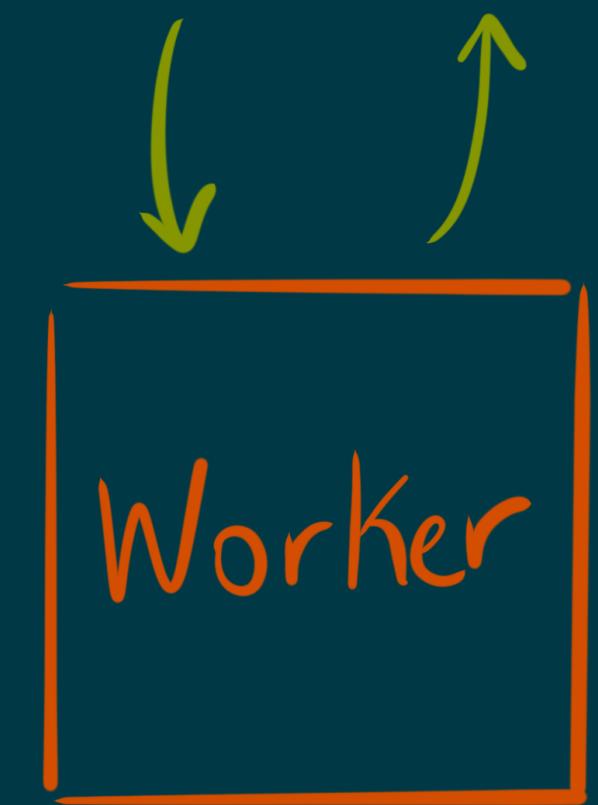
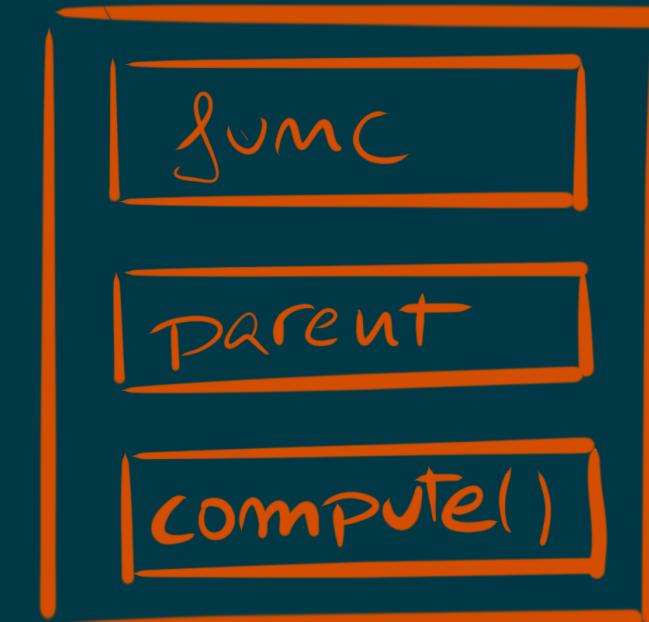
Worker

← Python!

PythonRDD



PythonRunner



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

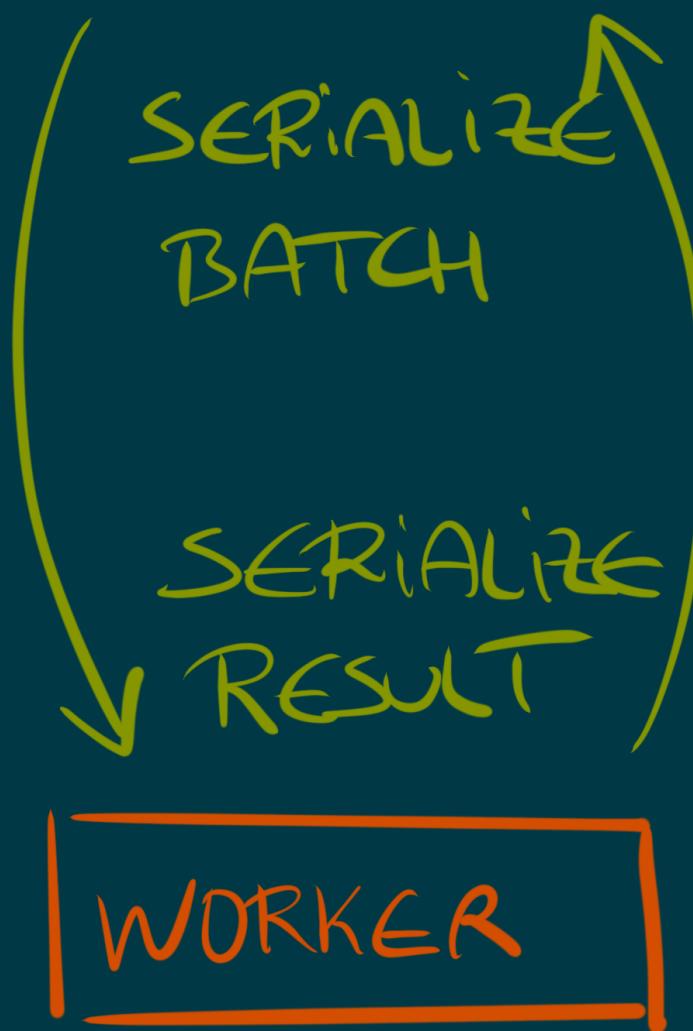
SERIALIZED  
BATCH

SERIALIZED  
RESULT

WORKER

UDF RUNNER

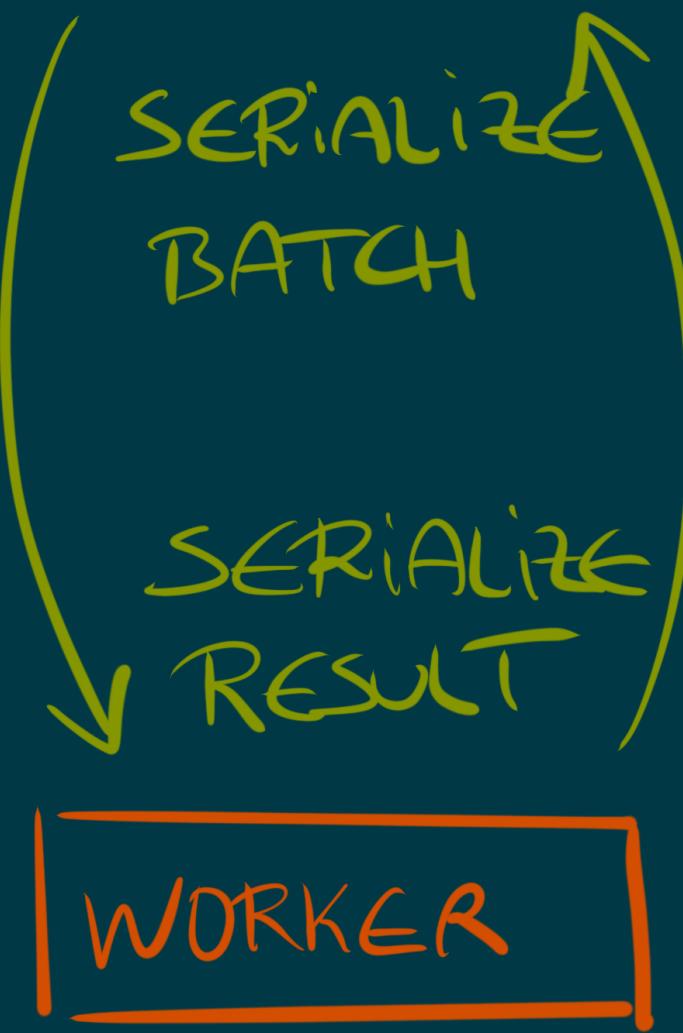
ARROW RUNNER



WORKER

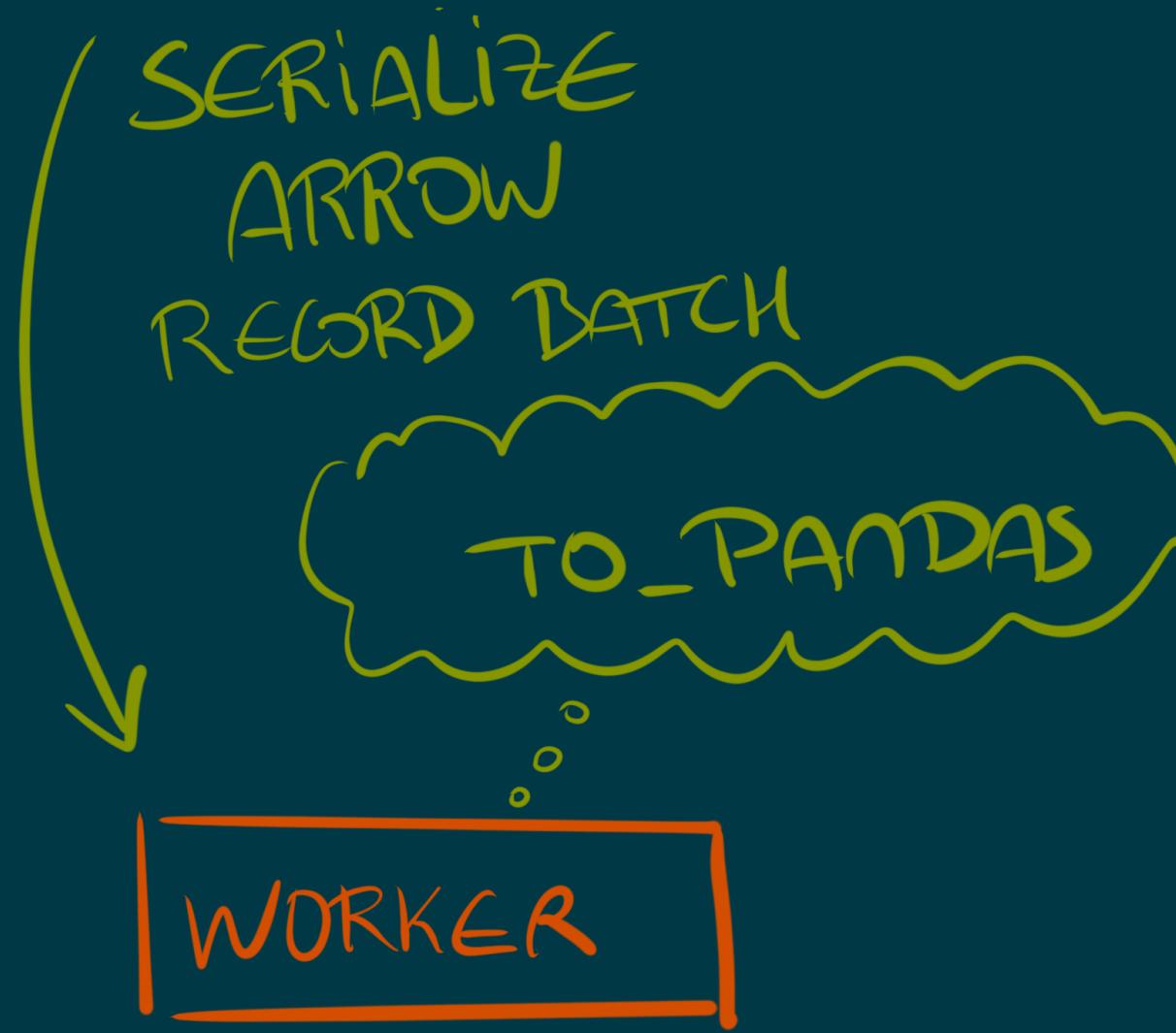
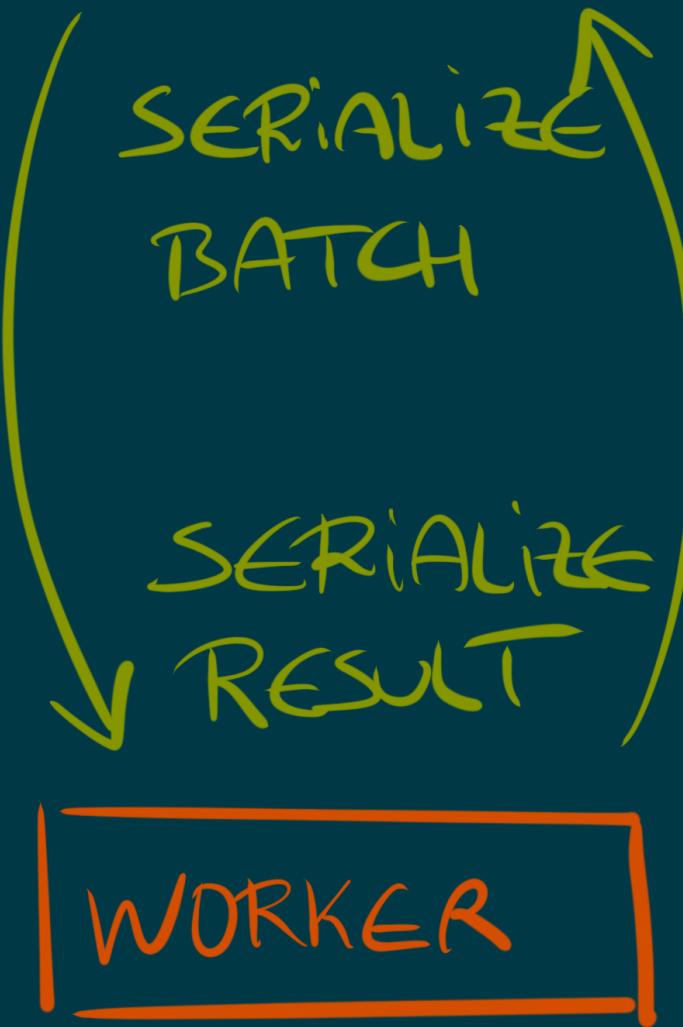
UDF RUNNER

ARROW RUNNER



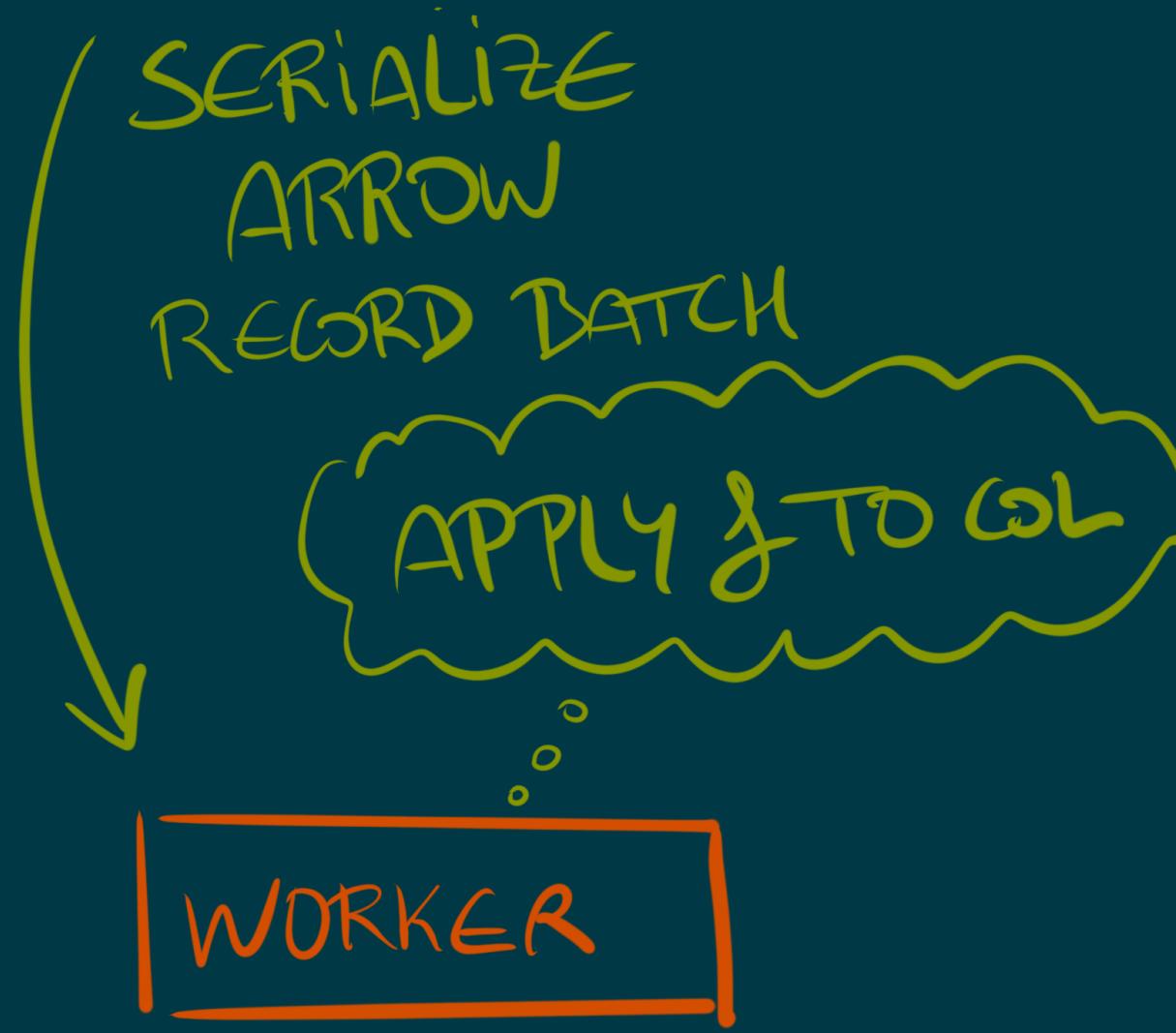
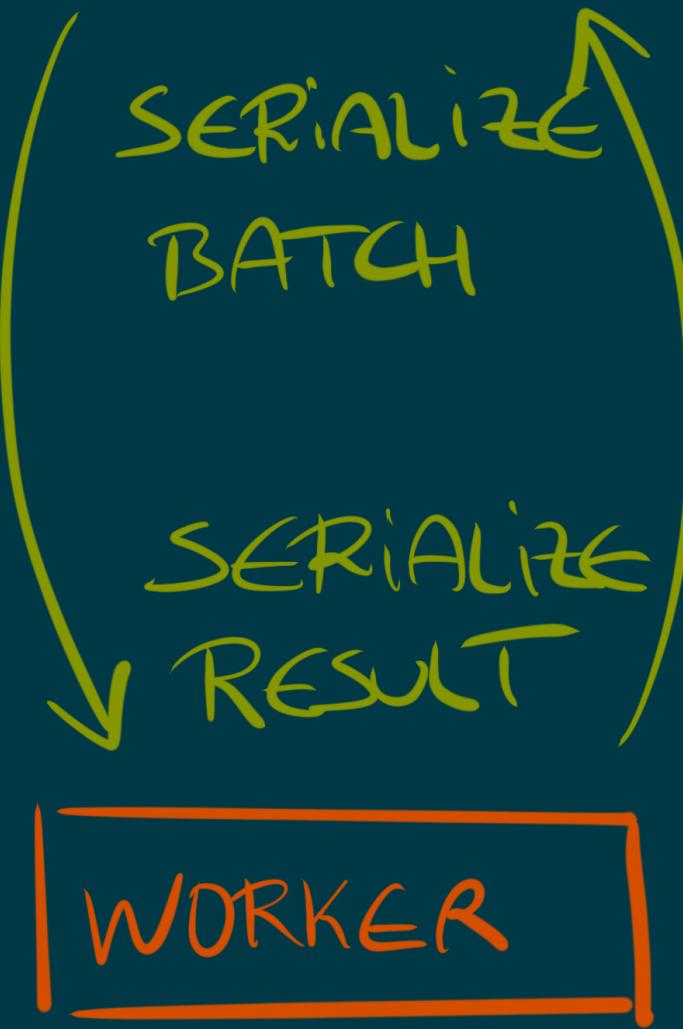
UDF RUNNER

ARROW RUNNER



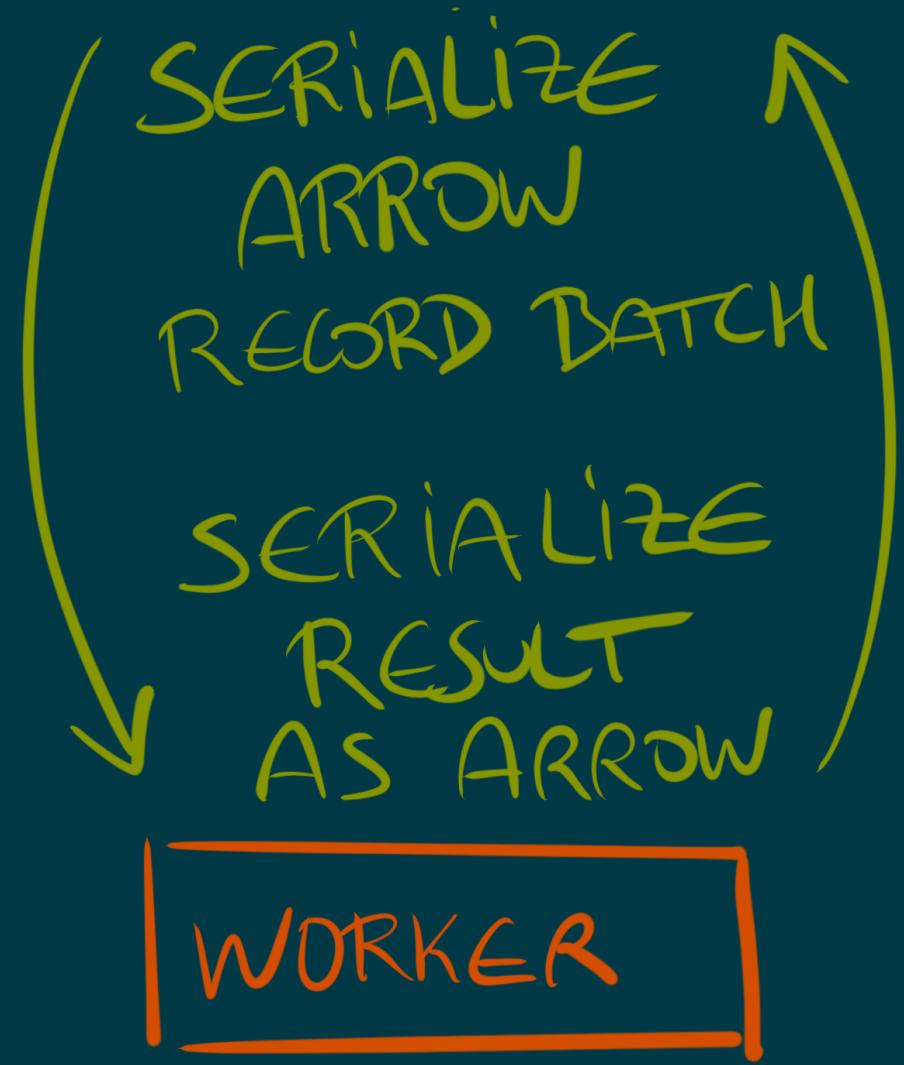
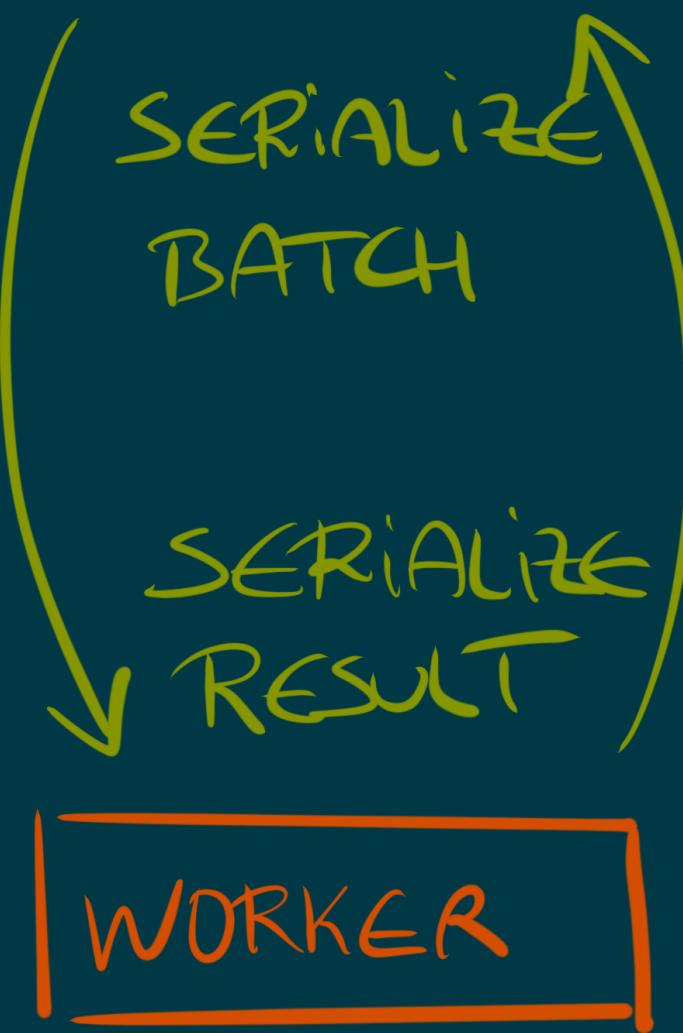
UDF RUNNER

ARROW RUNNER



UDF RUNNER

ARROW RUNNER



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](#)

# 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