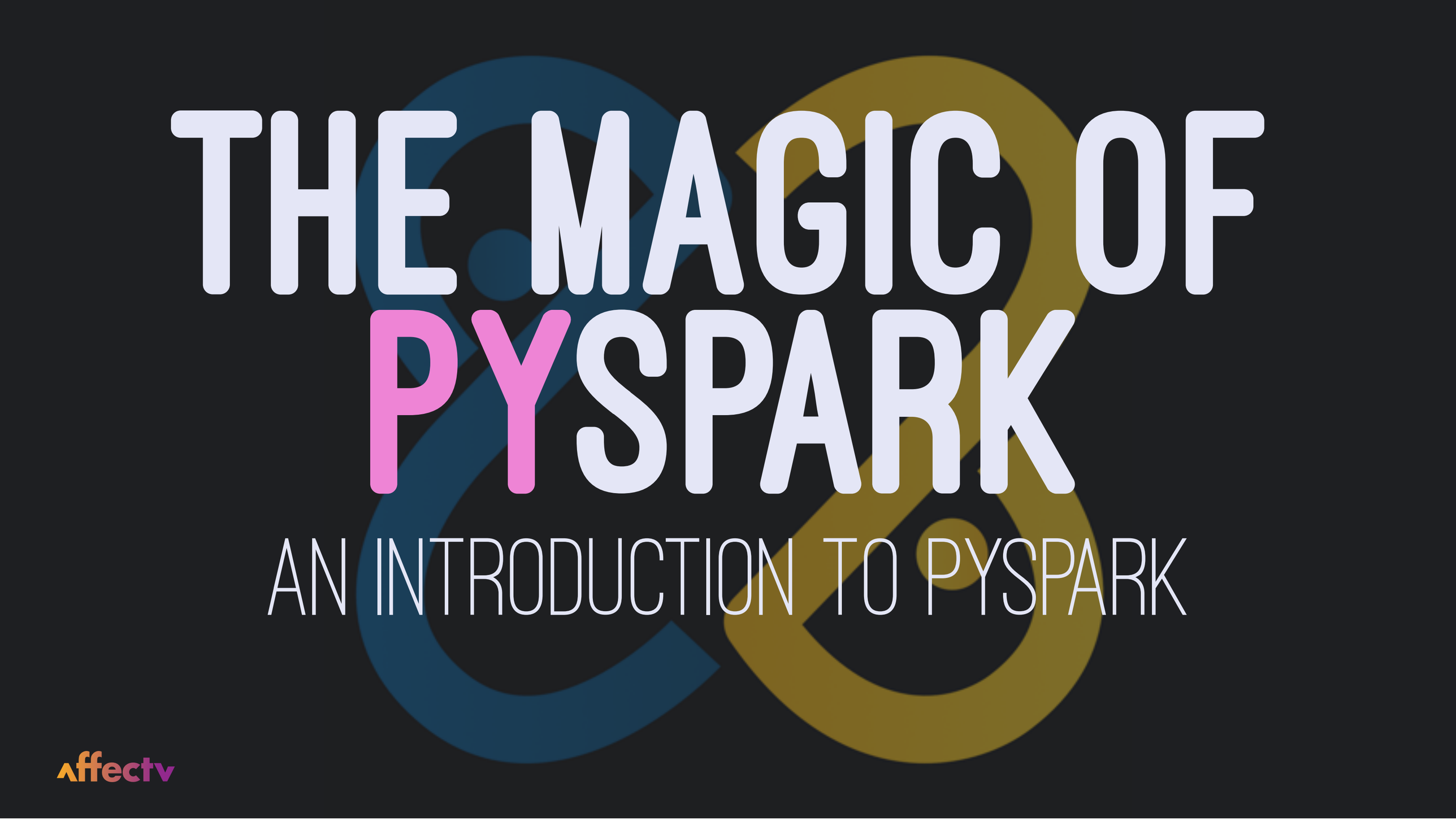
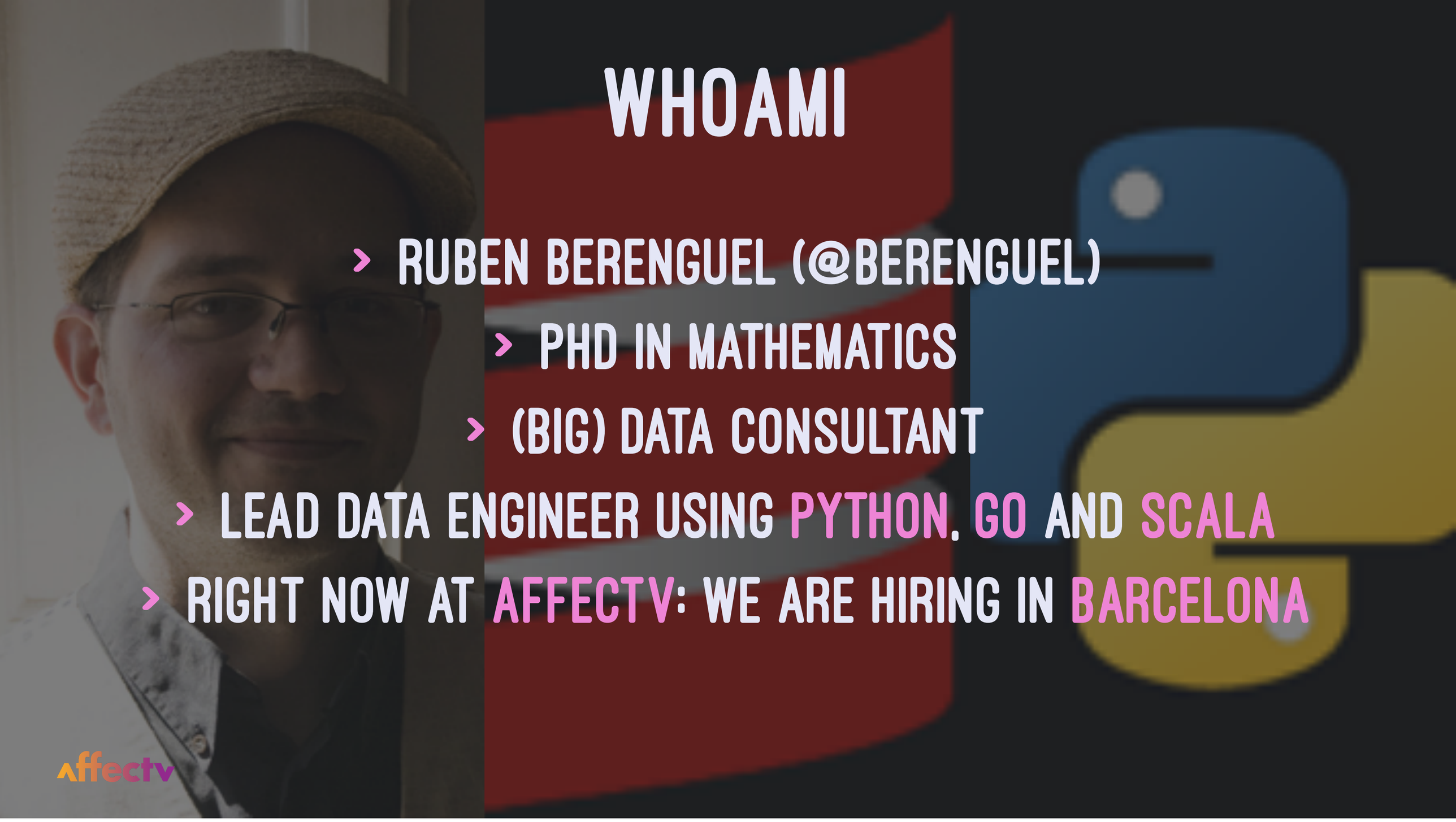


I will explain a bit what is Spark and how it works and then how PySpark works. Then, workshop time

affectv

Test footnote, hello there



WHOAMI

- RUBEN BERENGUEL (@BERENGUEL)
- PHD IN MATHEMATICS
- (BIG) DATA CONSULTANT
- LEAD DATA ENGINEER USING PYTHON, GO AND SCALA
- RIGHT NOW AT AFFECTV: WE ARE HIRING IN BARCELONA

WHAT IS SPARK?

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

HOW DOES SPARK WORK?

affectv

SPARK USUALLY RUNS ON TOP OF A CLUSTER MANAGER

affectv

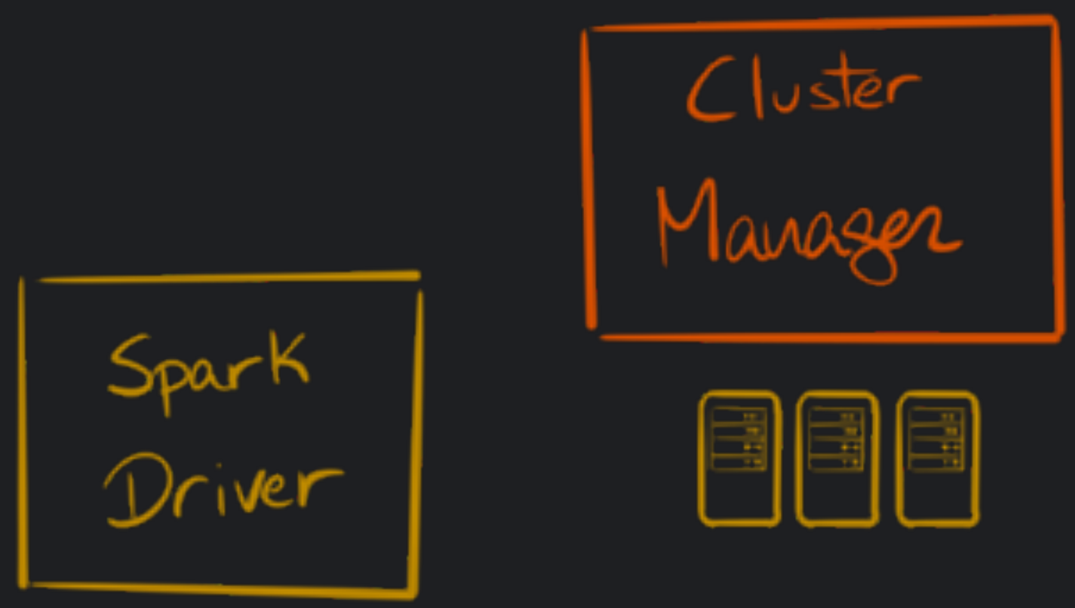
This can be standalone, YARN, Mesos or in the bleeding edge, Kubernetes (using the Kubernetes scheduler)

AND A DISTRIBUTED STORAGE

affectv

A SPARK PROGRAM
RUNS IN THE DRIVER

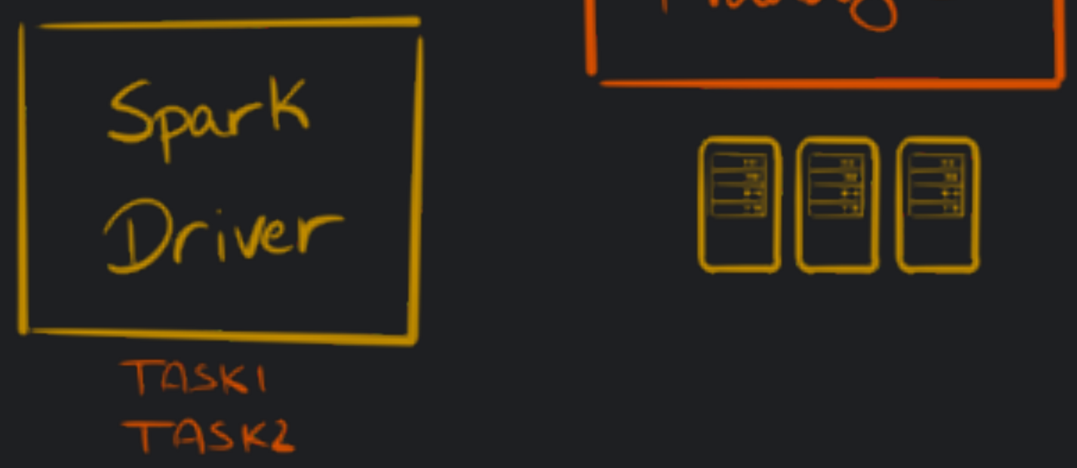
THE DRIVER REQUESTS RESOURCES FROM THE CLUSTER MANAGER TO RUN TASKS



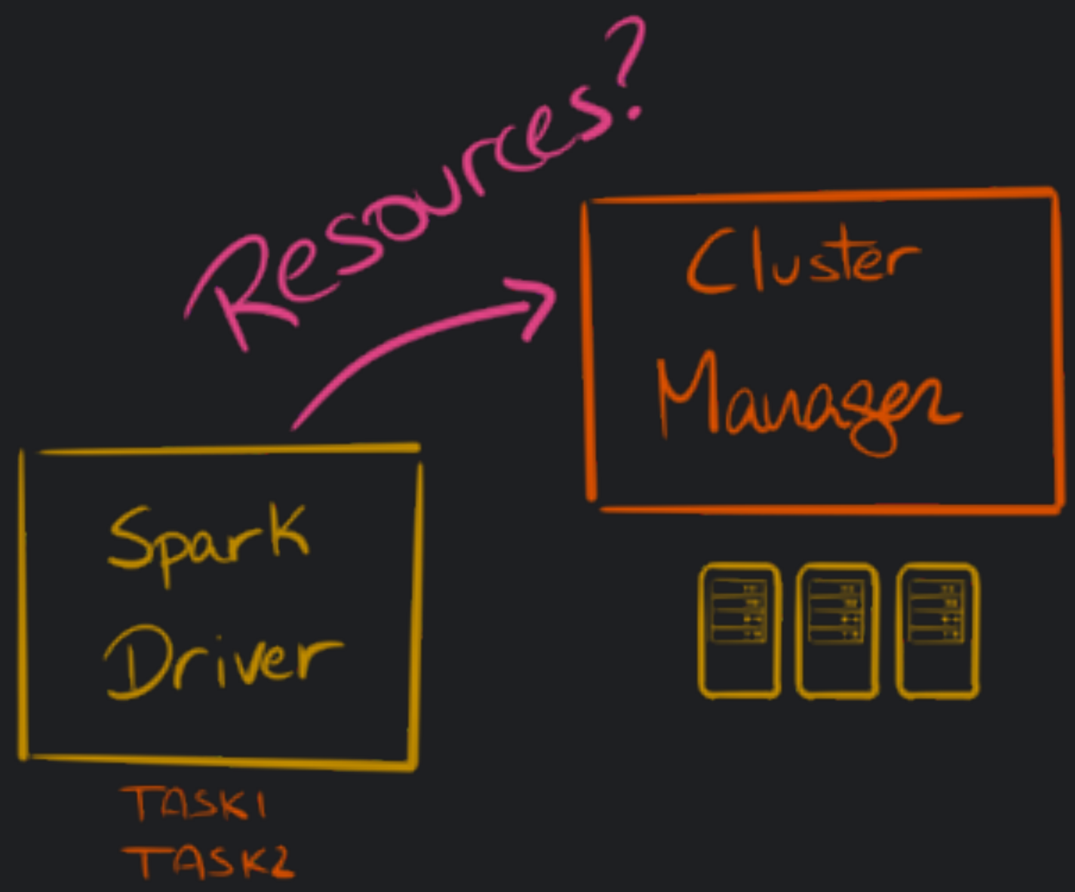
affectv

We usually don't need to worry about what the executors do (unless they blow up)

THE DRIVER REQUESTS RESOURCES FROM THE CLUSTER MANAGER TO RUN TASKS

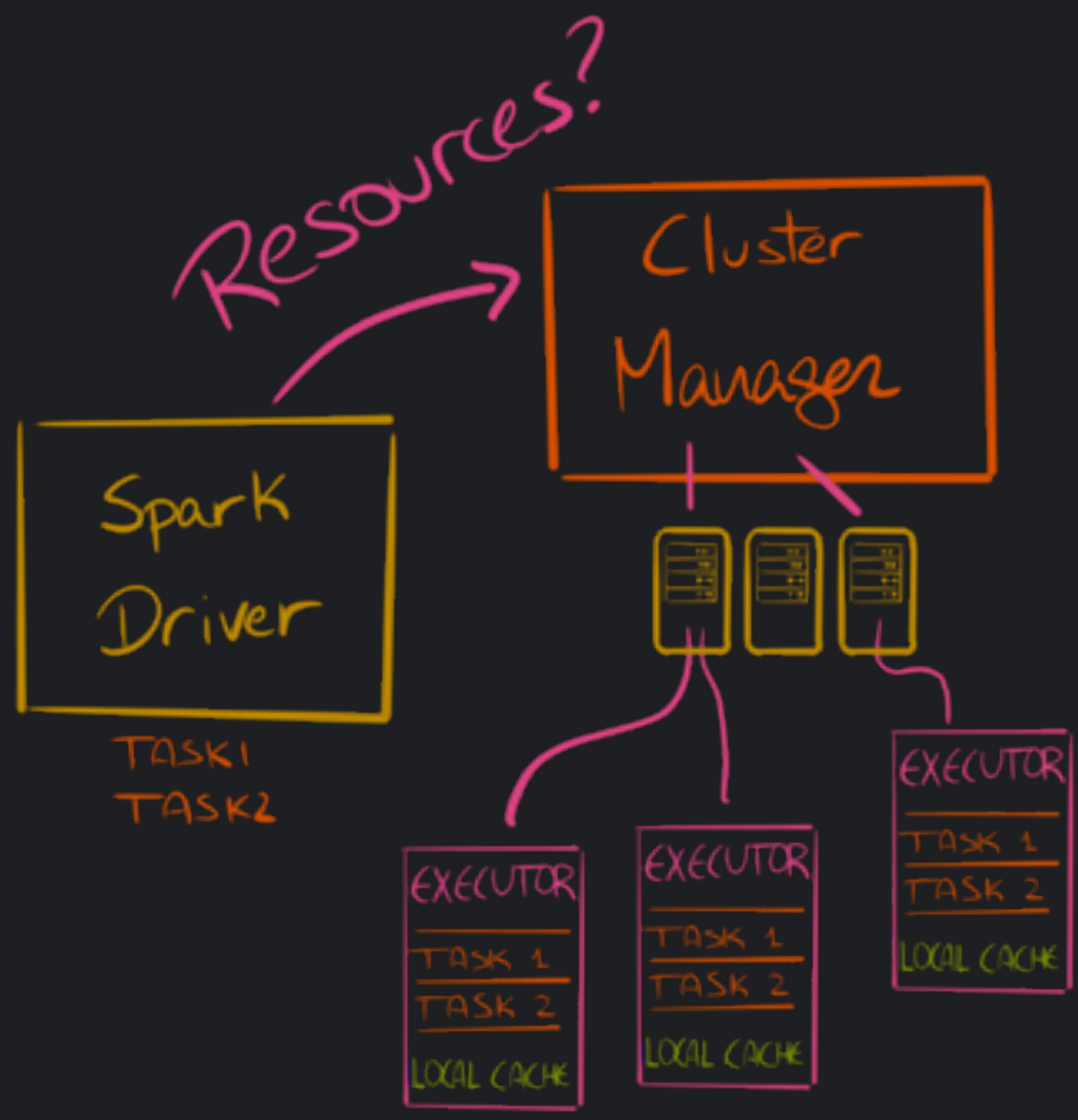


THE DRIVER REQUESTS RESOURCES FROM THE CLUSTER MANAGER TO RUN TASKS



affectv

THE DRIVER REQUESTS RESOURCES FROM THE CLUSTER MANAGER TO RUN TASKS



affectv

This is very nice, but what is the magic that lets us compute things on several machines, and is machine-failure safe?

THE MAIN BUILDING BLOCK IS THE RDD: RESILIENT DISTRIBUTED DATASET

affectv

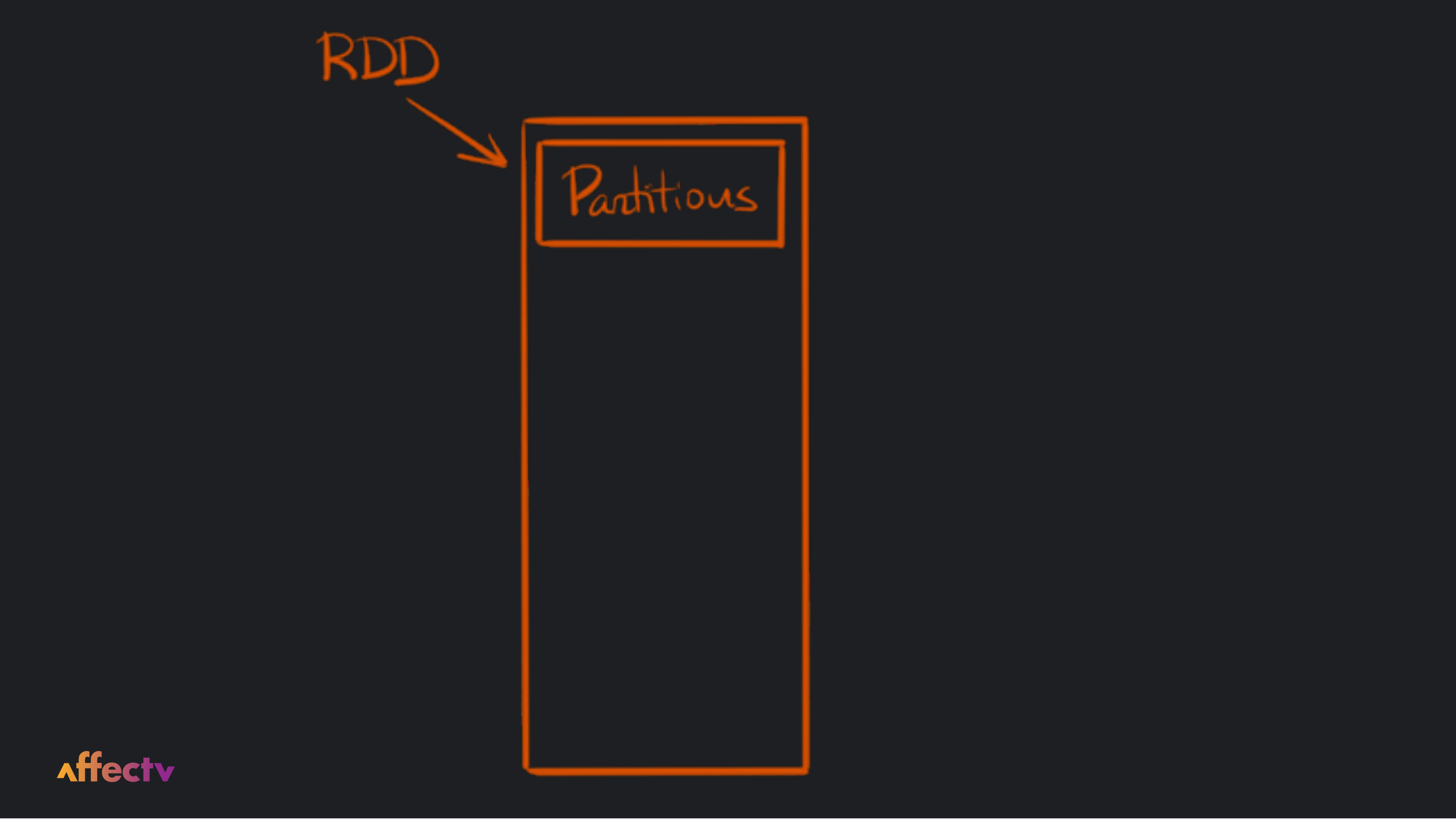
RDDs are defined in the Scala core. In Python and R we end up interacting with the underlying JVM objects

affectv

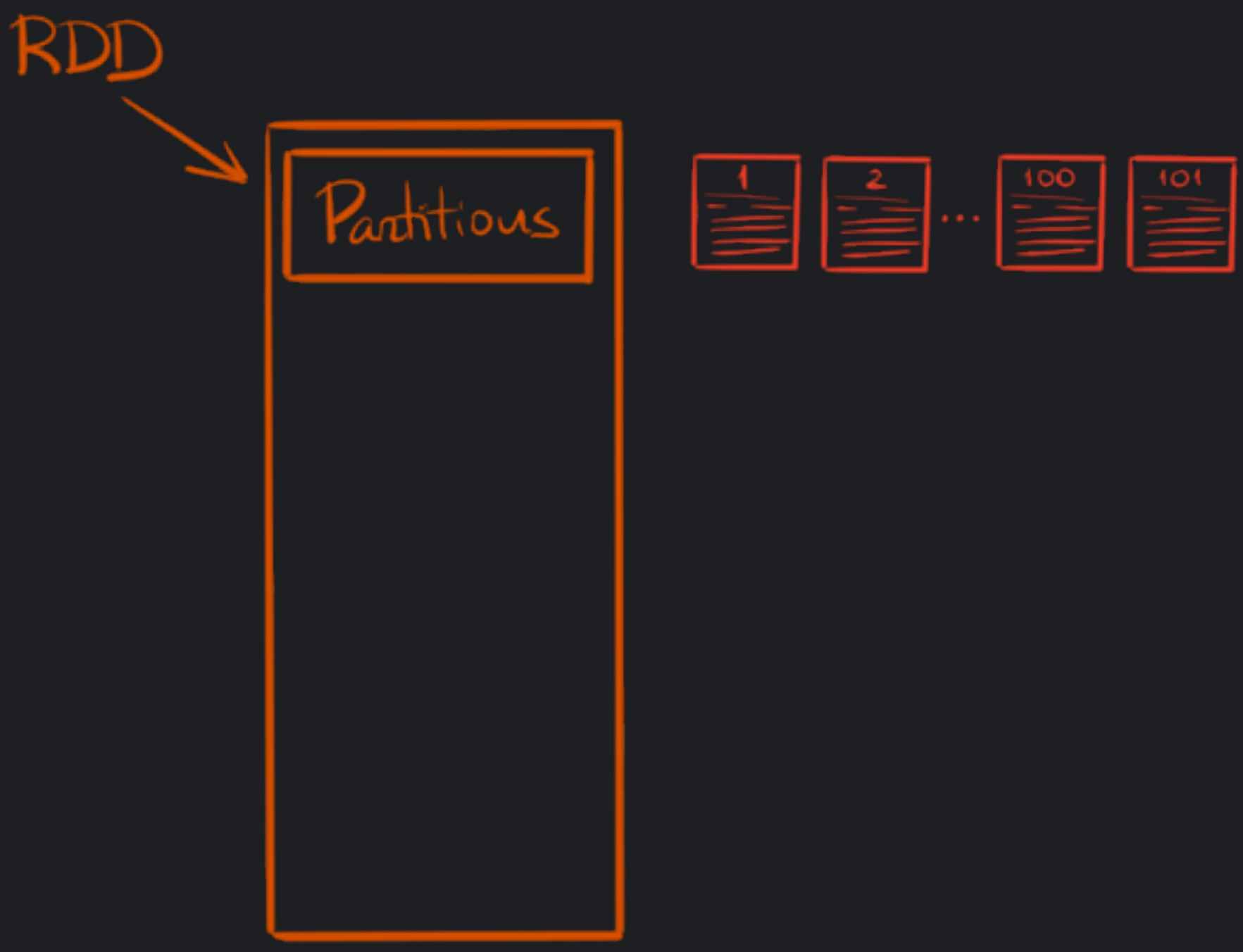
Happy RDD

affectv

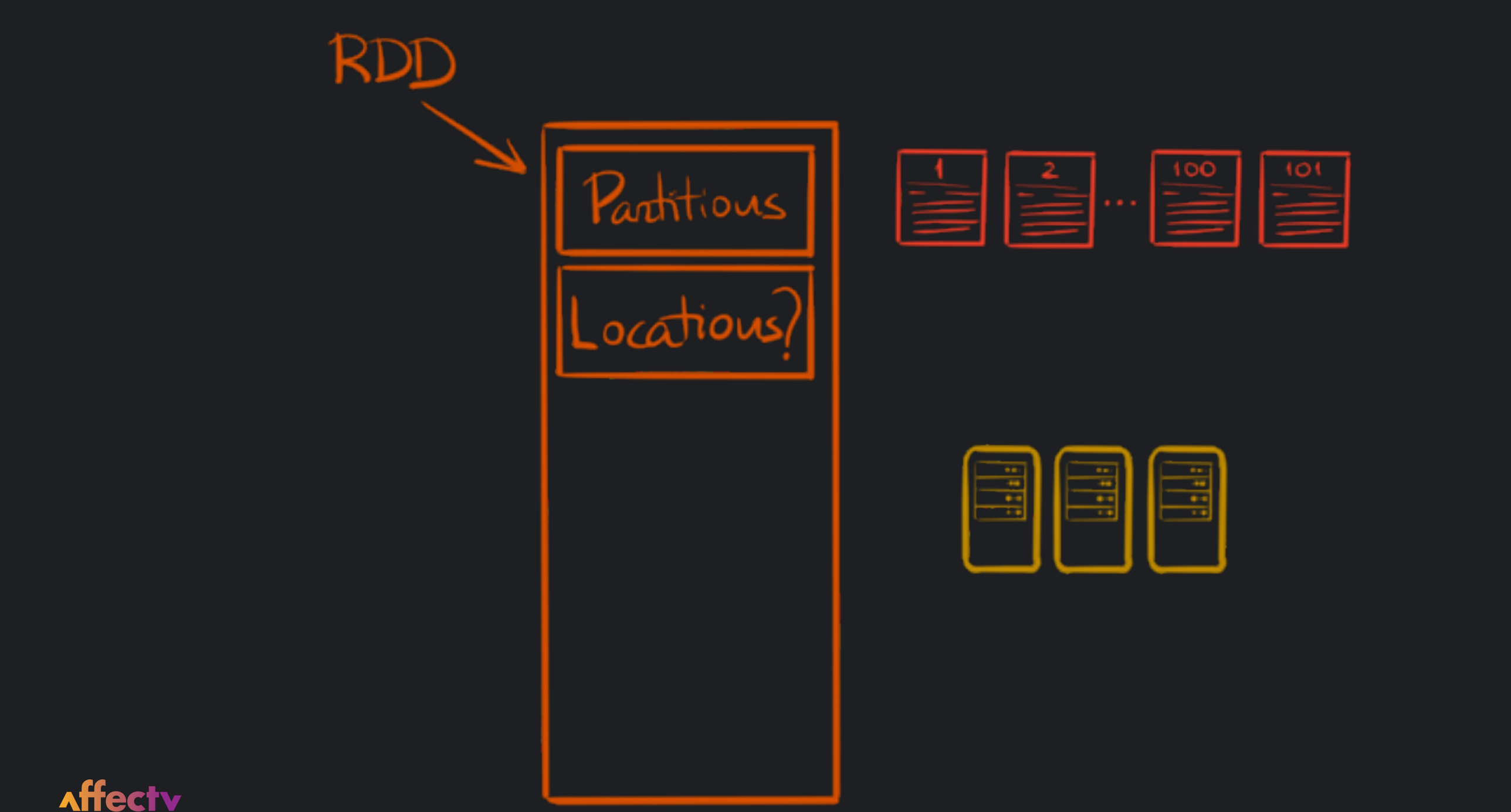
Happy RDD. A RDD needs 5 items to be considered complete (2 of them optional)



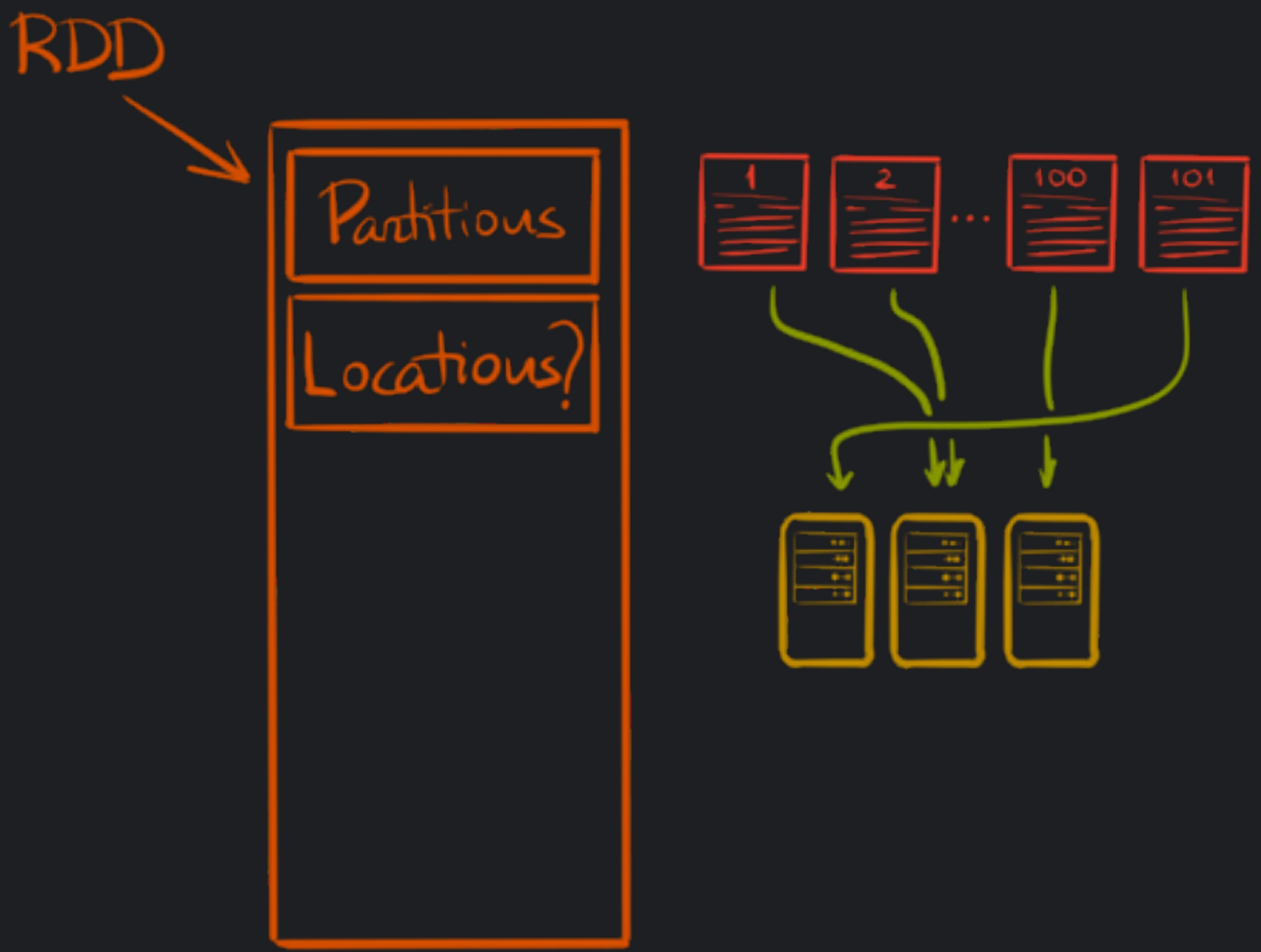
Partitions define how the data is *partitioned* across machines



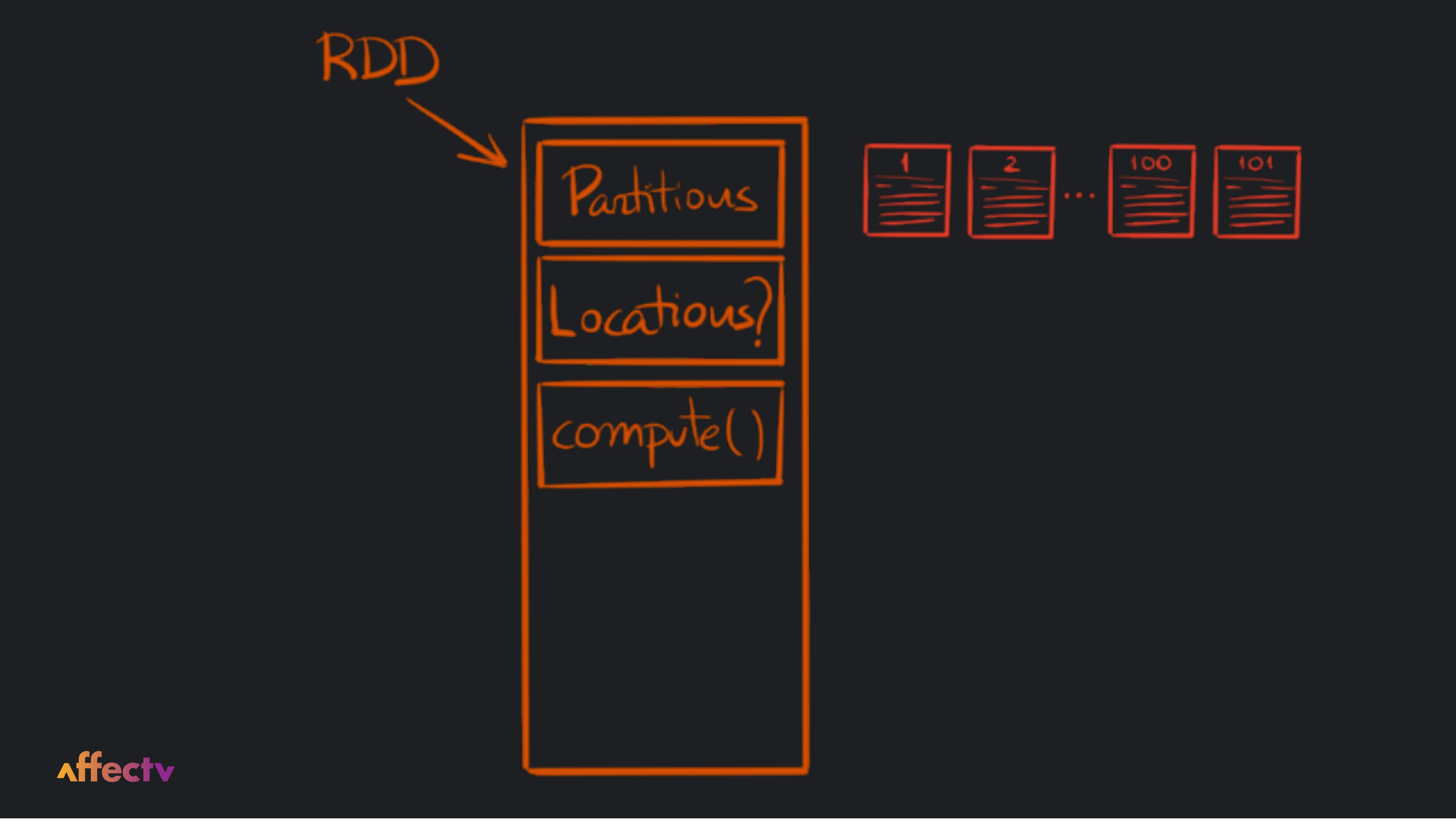
affectv



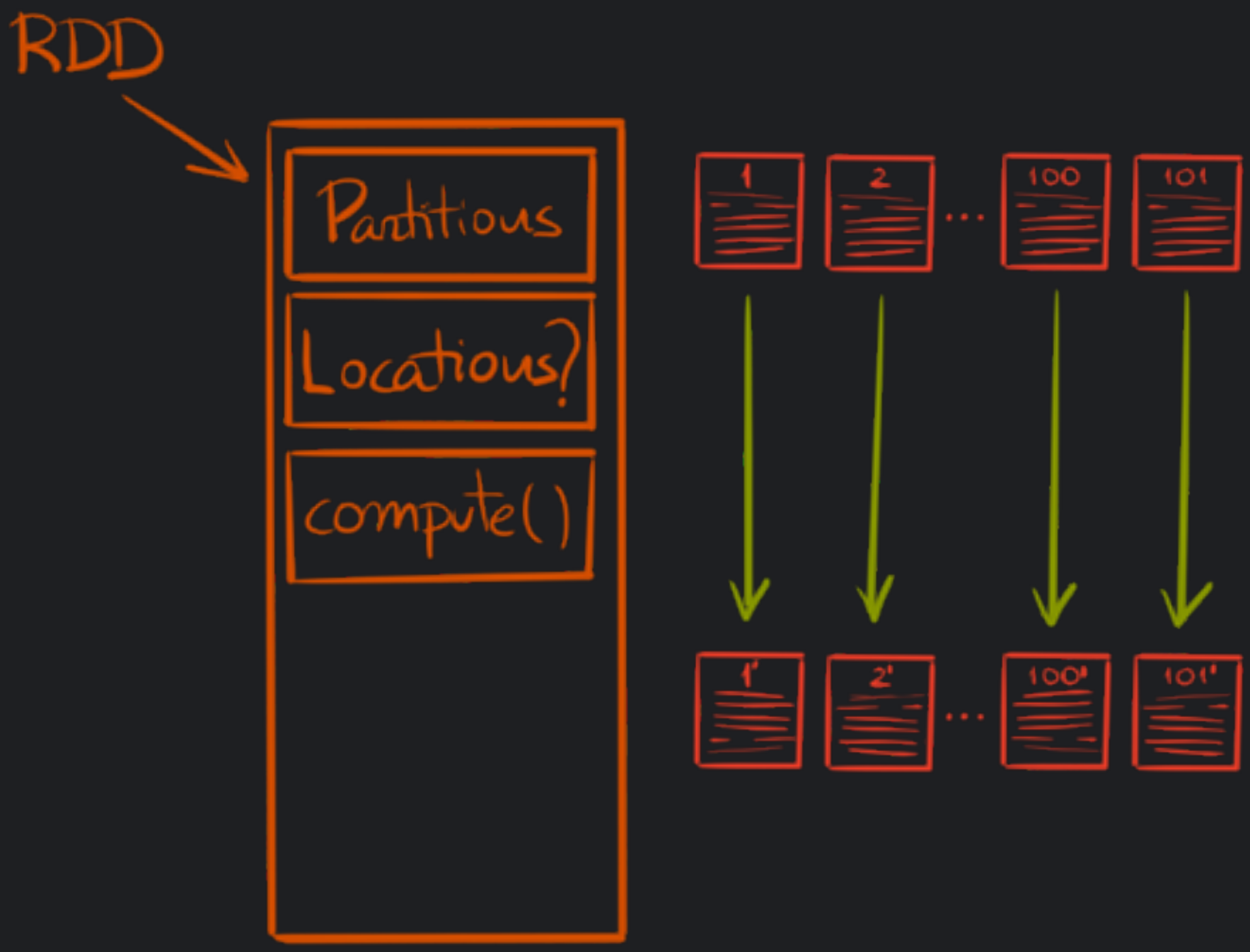
Locations (`preferredLocations`) are optional, and allow for fine grained execution to avoid shuffling data across the cluster



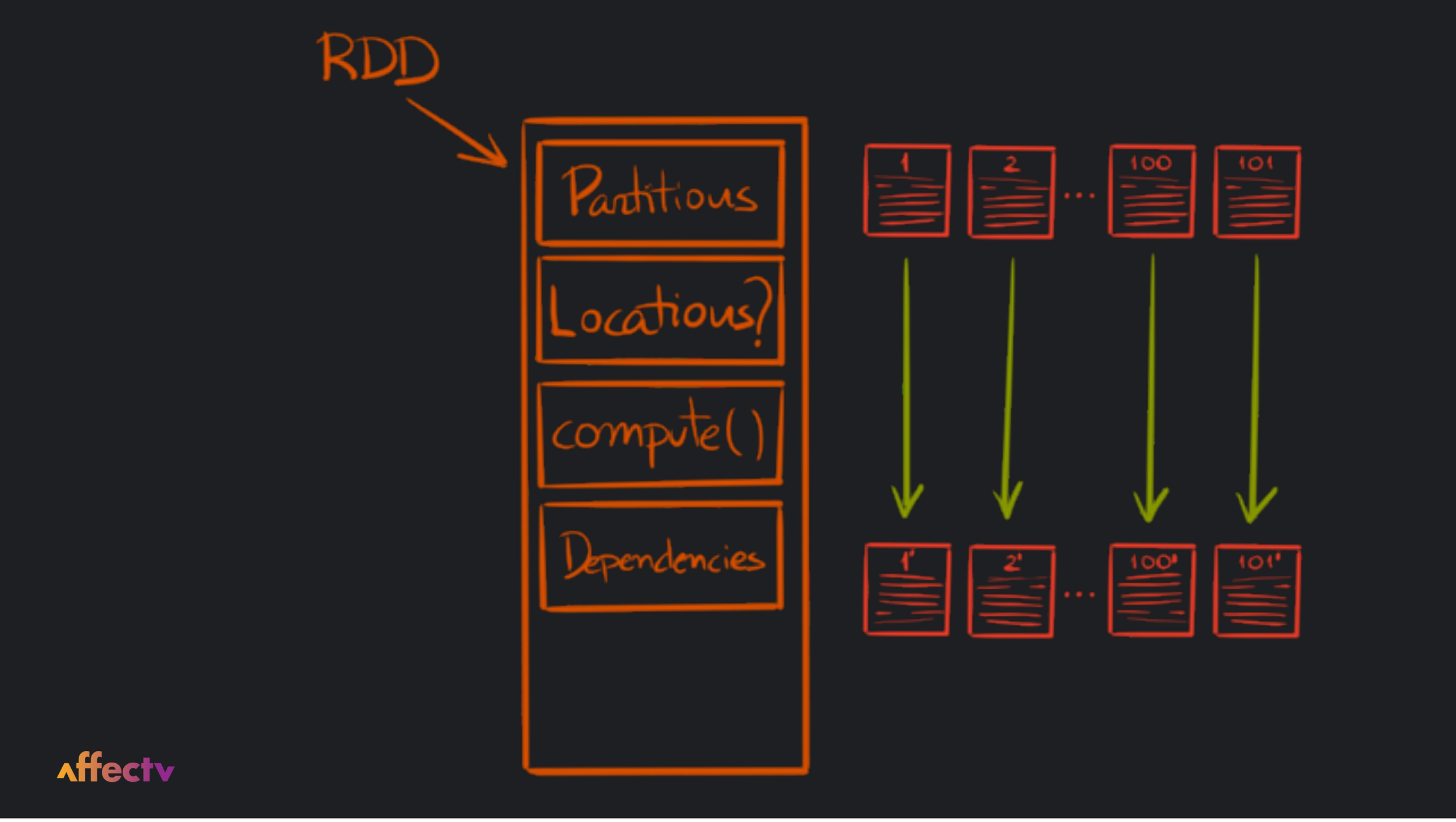
affectv



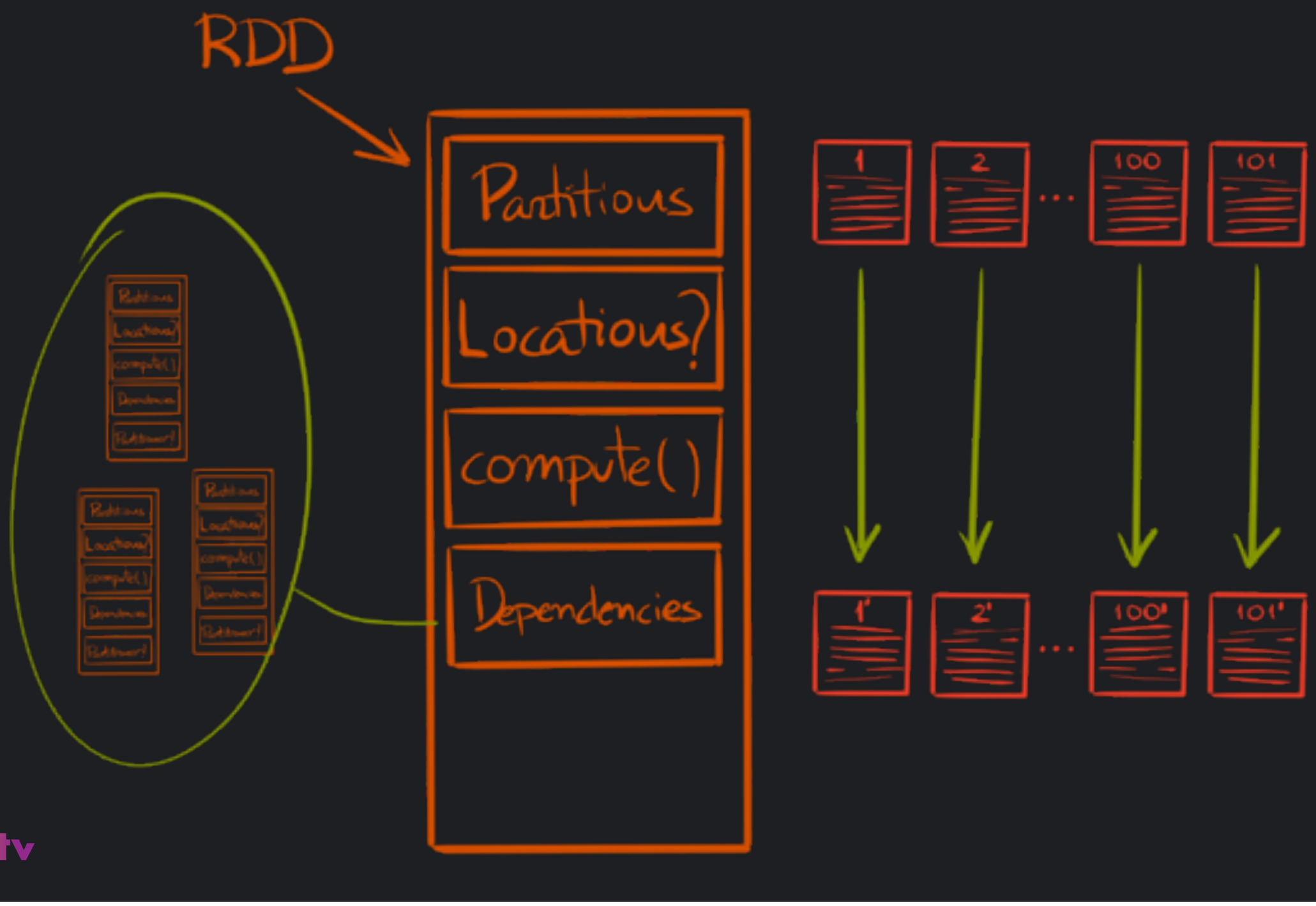
Compute is evaluated in the executors, each executor has access to its assigned partitions. The result of compute is a new RDD, with different data



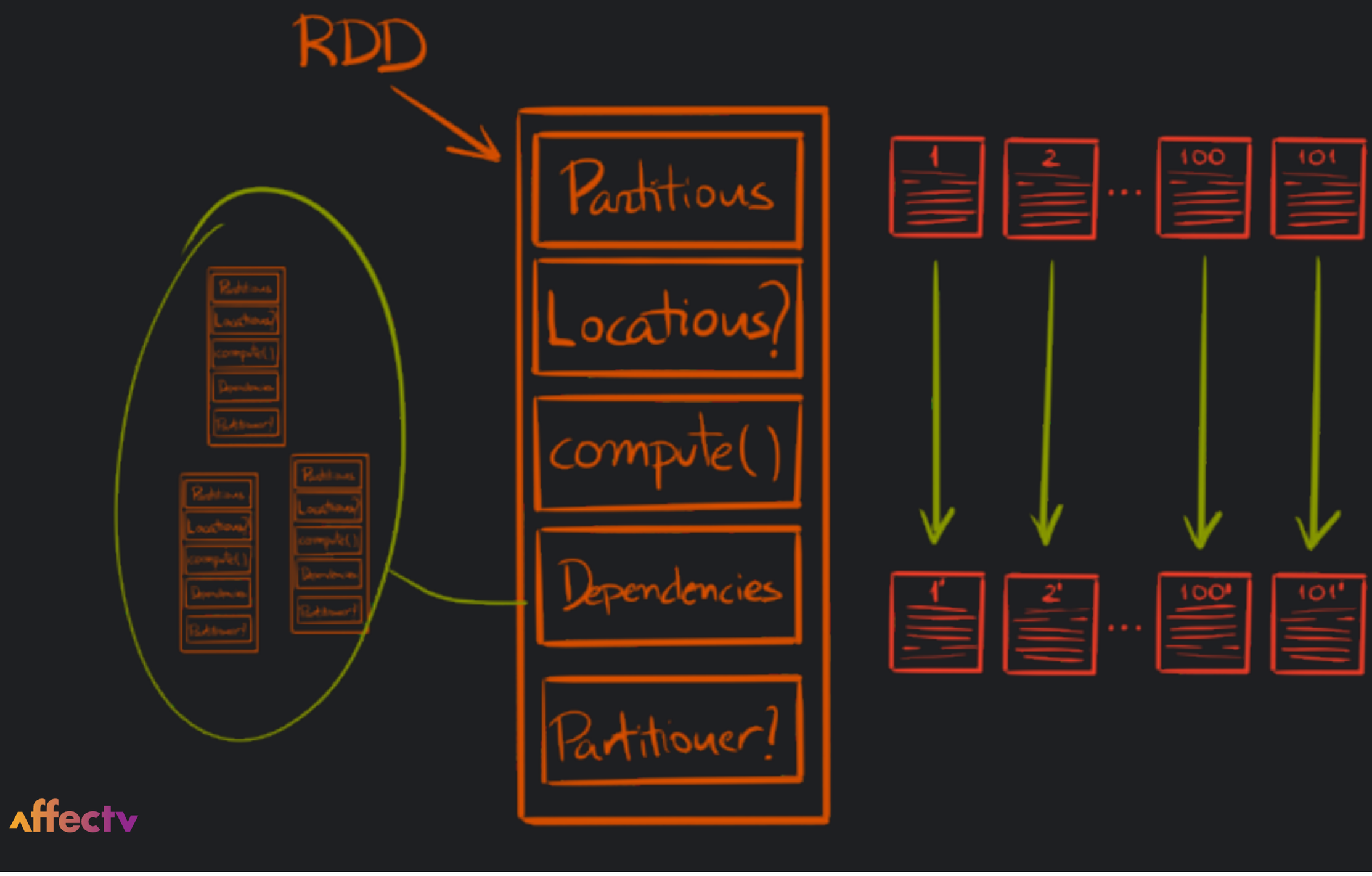
affectv



Dependencies are also called *lineage*. Each **RDD** (and by extension, each partition) has a specific way to be computed. If for some reason one is lost
(say, a machine dies), **Spark** knows how to recompute *only* that


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

affectv



The partitioner needs to be a 1-1 function from keys/whatever to the data, to allow recomputing. It is as well optional (Spark will default to something sensible then)

TWO KIND OF OPERATIONS:

TRANSFORMATIONS

&

ACTIONS

affectv

TRANSFORMATIONS: RESHAPE THE DATA. THEY ARE PART OF STAGES

- FILTER
- MAP
- JOIN

affectv

The split between stage and stage happens when we *shuffle* data.

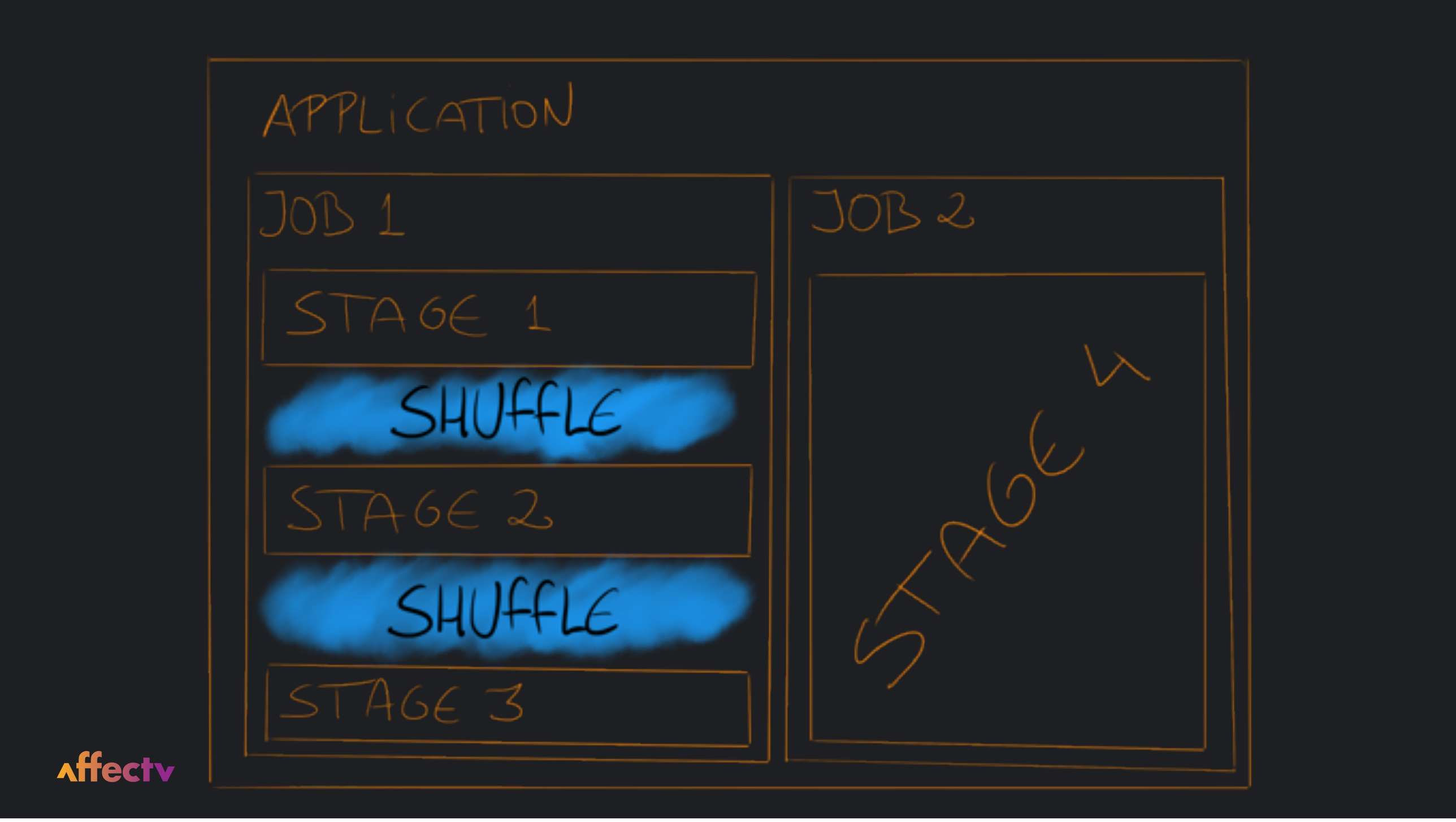
Shuffling is the move of data from executor to executor, required by many operations. If we could avoid shuffling, distributed computing would be easier

ACTIONS: RETURN A RESULT. THEY DEFINE JOBS

- COUNT
- SHOW
- WRITE

affectv

Actions create a barrier between stages


APPLICATION
JOB 1
STAGE 1
SHUFFLE
STAGE 2
SHUFFLE
STAGE 3
JOB 2
STAGE 4
affectv

PYSPARK

affectv

I will use PySpark to refer to the Python API on top of Spark, and say Scala Spark or just Spark for the Scala API

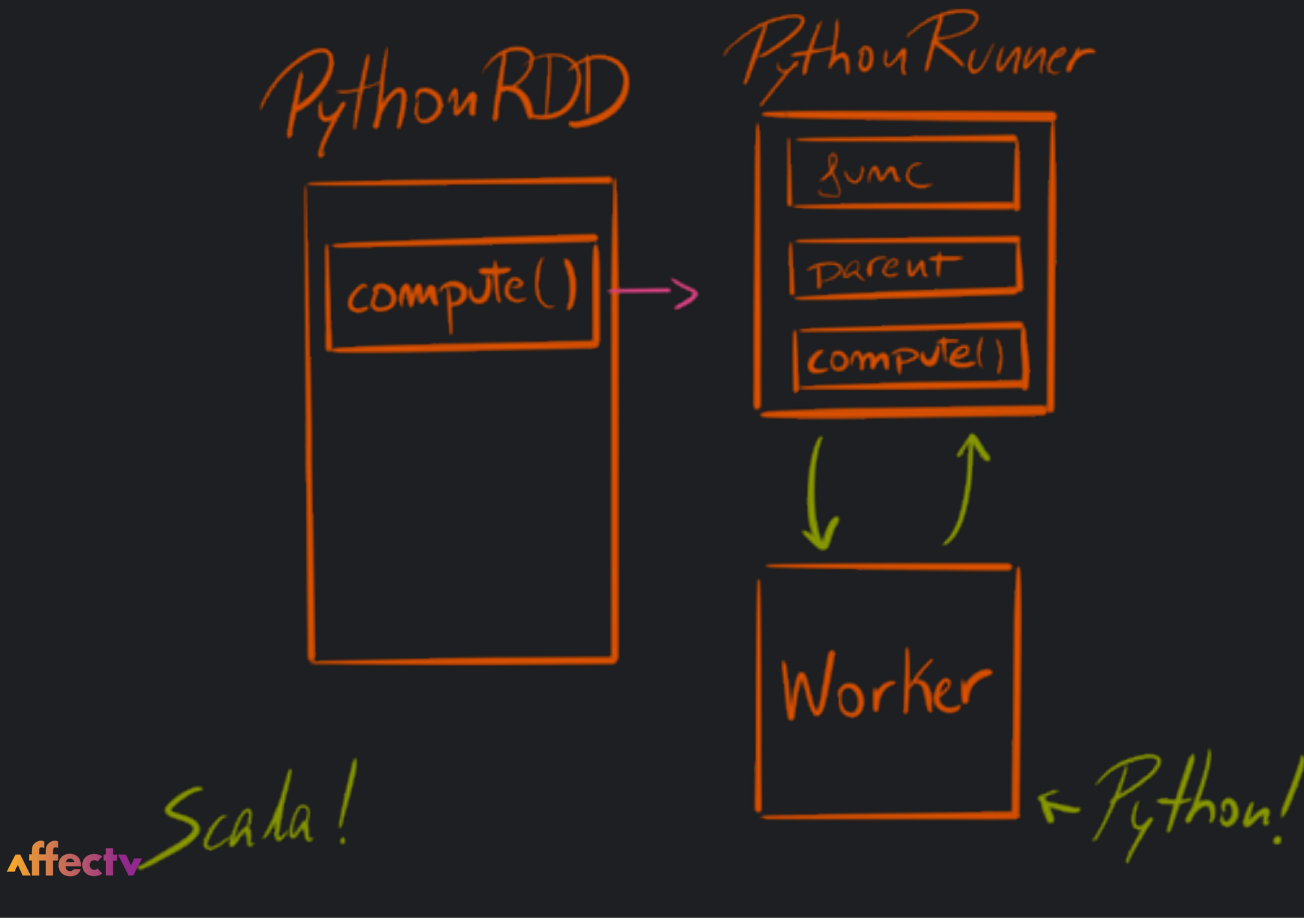
PYSPARK OFFERS A PYTHON API TO THE SCALA CORE OF SPARK

affectv

IT USES THE PY4J BRIDGE

affectv

Each object in PySpark comes bundled with a JVM gateway (started when the
Python driver starts). Python methods then act on the internal JVM object by passing serialised messages to the Py4J gateway


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

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

affectv

WHAT ABOUT DATAFRAMES?

affectv

Isn't Spark supposed to be pretty magic and plan and optimise stuff?

THE RDD API EXPECTS YOU TO HANDLE EVERYTHING.

BUT SPARK CAN MAGICALLY OPTIMISE EVERYTHING..

affectv

..IF YOU USE SPARK DataFrame

affectv

You can think of DataFrames as RDDs which actually refer to tables. They have column names, and may have types for each column

When you operate on DataFrames, plans are created magically. And actually it will generate a logical plan, an optimised logical plan, an execution plan...

AND THE PLAN WILL BE OPTIMISED USING CATALYST

affectv

The Catalyst optimiser. There's also a code generator in there (using Janino to compile Java code in real time, how that works and why is a matter for another presentation...) Catalyst prunes trees

PLEASE. PLEASE. USE SPARK ≥ 2.3 (OR. THE LATEST YOU CAN)

affectv

YOU SHOULD TRY TO USE DataFrame UNLESS THERE IS A REASON NOT TO (LIKE SOME MLLIB METHOD)

affectv

YOU SHOULD ALSO ENABLE ARROW OPTIMISATIONS TO SPEED UP THE DATA TRANSFER FROM THE JVM TO PYTHON

WE'LL SEE WHY AND HOW LATER DURING THE WORKSHOP

affectv

NEW PROJECT: KOALAS[1]

OFFERS A UNIFIED API BETWEEN PANDAS AND SPARK DATAFRAMES (AS MUCH AS POSSIBLE)

[1] HTTPS://GITHUB.COM/DASK/DASK

affectv

The project is here

PURE PYTHON ALTERNATIVE: DASK[1]

ANOTHER DISTRIBUTED FRAMEWORK

[1] HTTPS://GITHUB.COM/DASK/DASK

affectv

The project is here

SHOULD YOU USE SPARK?

affectv

YES IF

- THE DATA IS VERY LARGE (SIGNIFICANTLY LARGER THAN MEMORY)
- YOUR ORG ALREADY HAS SPARK CLUSTER OR CODEBASE
- THERE IS NO BETTER ALTERNATIVE

NO IF

- YOU WANT TO ADD SPARK TO YOUR CV
  - JAVA STACKTRACES SCARE YOU

affectv

RESOURCES

- SPARK DOCUMENTATION
- HIGH PERFORMANCE SPARK BY HOLDEN KARAU
- THE INTERNALS OF APACHE SPARK 2.4.2 BY JACEK LASKOWSKI
- SPARK'S GITHUB
- BECOME A CONTRIBUTOR

affectv


THANKS!
affectv

WORKSHOP TIME!

affectv

GET THE SLIDES AND NOTEBOOK WE WILL USE FROM MY GITHUB:

github.com/rberenguel/

THE REPOSITORY IS

pyspark_workshop

affectv

FURTHER REFERENCES

affectv

ARROW

ARROW'S HOME
ARROW'S GITHUB
ARROW SPEED BENCHMARKS
ARROW TO PANDAS CONVERSION BENCHMARKS
POST: STREAMING COLUMNAR DATA WITH APACHE ARROW
POST: WHY PANDAS USERS SHOULD BE EXCITED BY APACHE ARROW
CODE: ARROW-PANDAS COMPATIBILITY LAYER CODE
CODE: ARROW TABLE CODE
PYARROW IN-MEMORY DATA MODEL
BALLISTA: A POC DISTRIBUTED COMPUTE PLATFORM (RUST)
PYJAVA: POC ON JAVA/SCALA AND PYTHON DATA INTERCHANGE WITH ARROW

affectv

PANDAS

PANDAS' HOME
PANDAS' GITHUB
GUIDE: IDIOMATIC PANDAS
CODE: PANDAS INTERNALS
DESIGN: PANDAS INTERNALS
TALK: DEMYSTIFYING PANDAS' INTERNALS, BY MARC GARCIA
MEMORY LAYOUT OF MULTIDIMENSIONAL ARRAYS IN NUMPY

affectv

SPARK/PYSPARK

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

affectv

PY4J

PY4J'S HOME
PY4J'S GITHUB
CODE: REFLECTION ENGINE

affectv

EOF

affectv