Module 5: Apache Spark

Shortcomings of MapReduce

Learning objectives

- List the main bottlenecks of MapReduce
- Explain how Apache Spark solves them

Shortcomings of MapReduce

Force your pipeline into Map and Reduce steps

Other workflows? i.e. join, filter, map-reduce-map

Shortcomings of MapReduce

Read from disk for each MapReduce job

Iterative algorithms? i.e. machine learning

Shortcomings of MapReduce

Only native JAVA programming interface

Other languages?
Interactivity?

Solution?

- New framework: same features of MapReduce and more
- Capable of reusing Hadoop ecosystem, e.g. HDFS, YARN…
- Born at UC Berkeley

Solutions by Spark

Other workflows? i.e. join, filter, map-reduce-map

~20 highly efficient distributed operations, any combination of them

Solutions by Spark

Iterative algorithms? i.e. machine learning

in-memory caching of data, specified by the user

Solutions by Spark

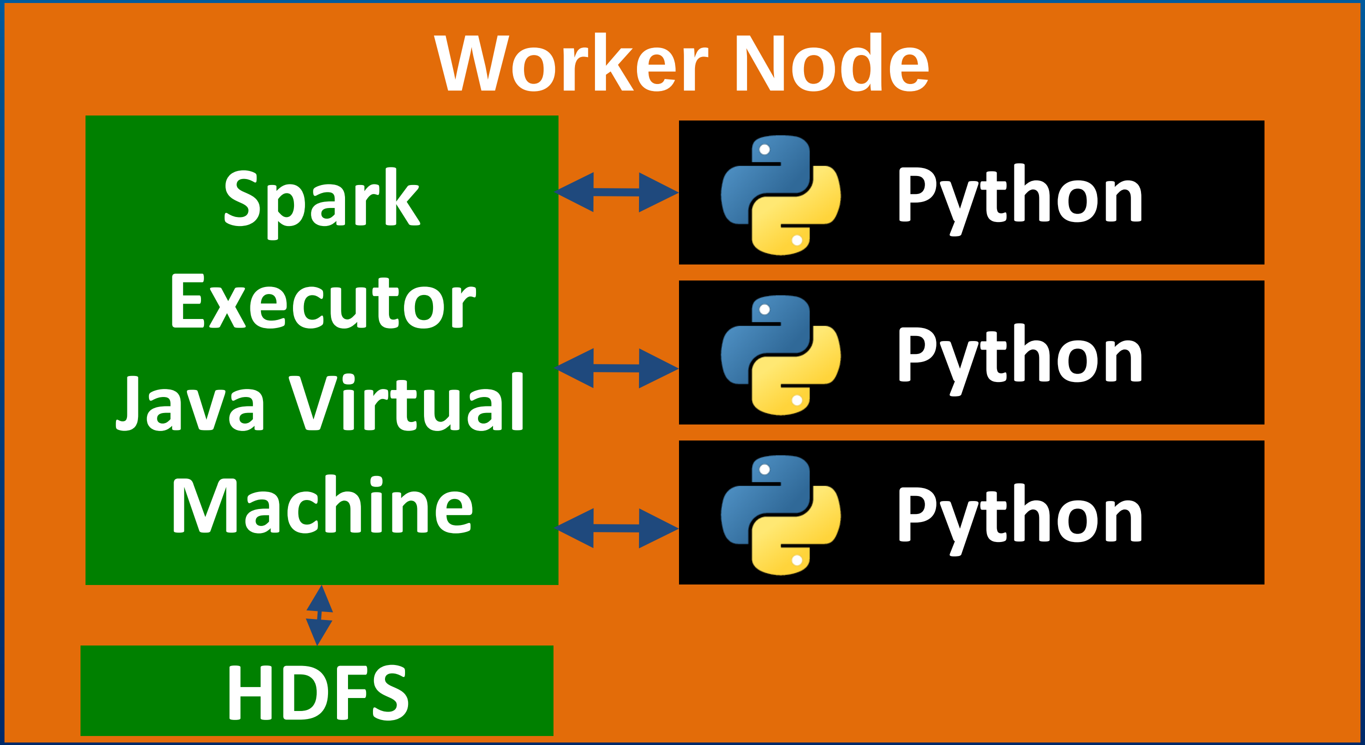
Interactivity? Other languages?

Native Python, Scala (, R) interface. Interactive shells.

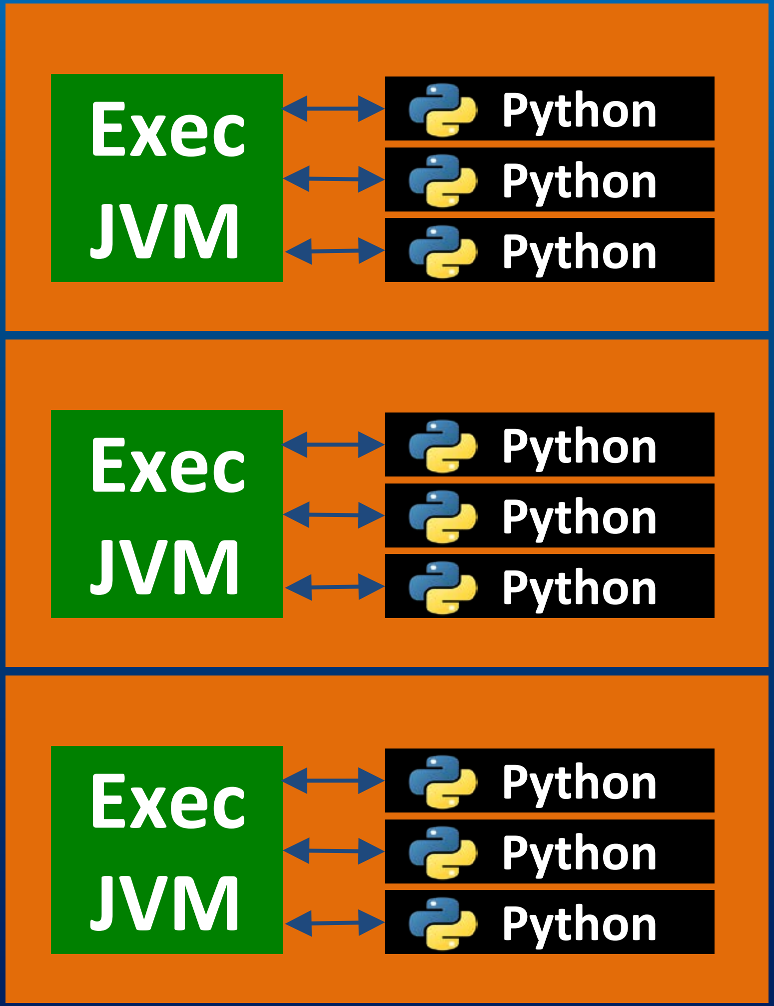
100TB Sorting competition

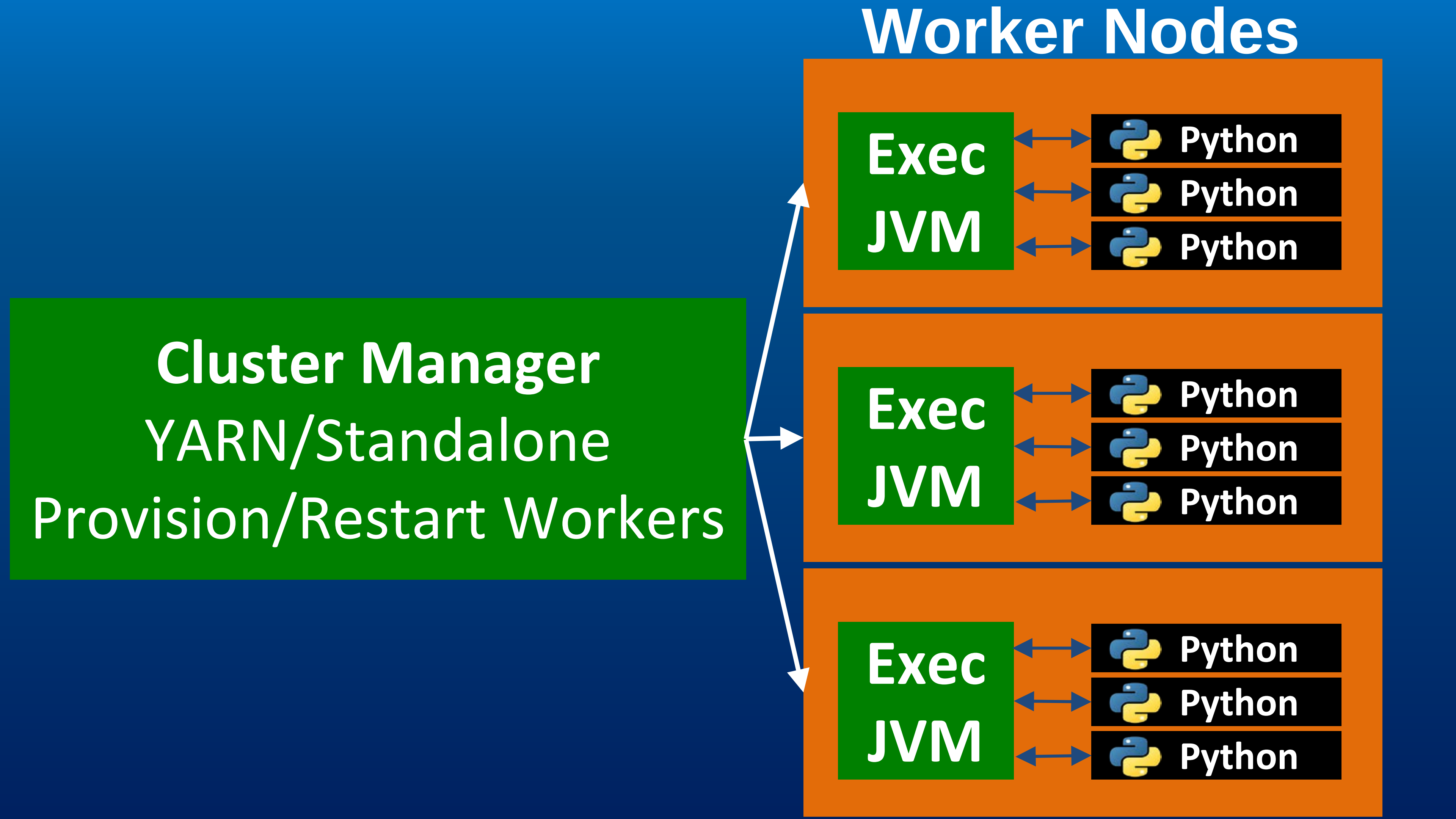
	Hadoop MR Record	**Spark Record**	**Spark 1 PB**
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**	**22.5 GB/min**

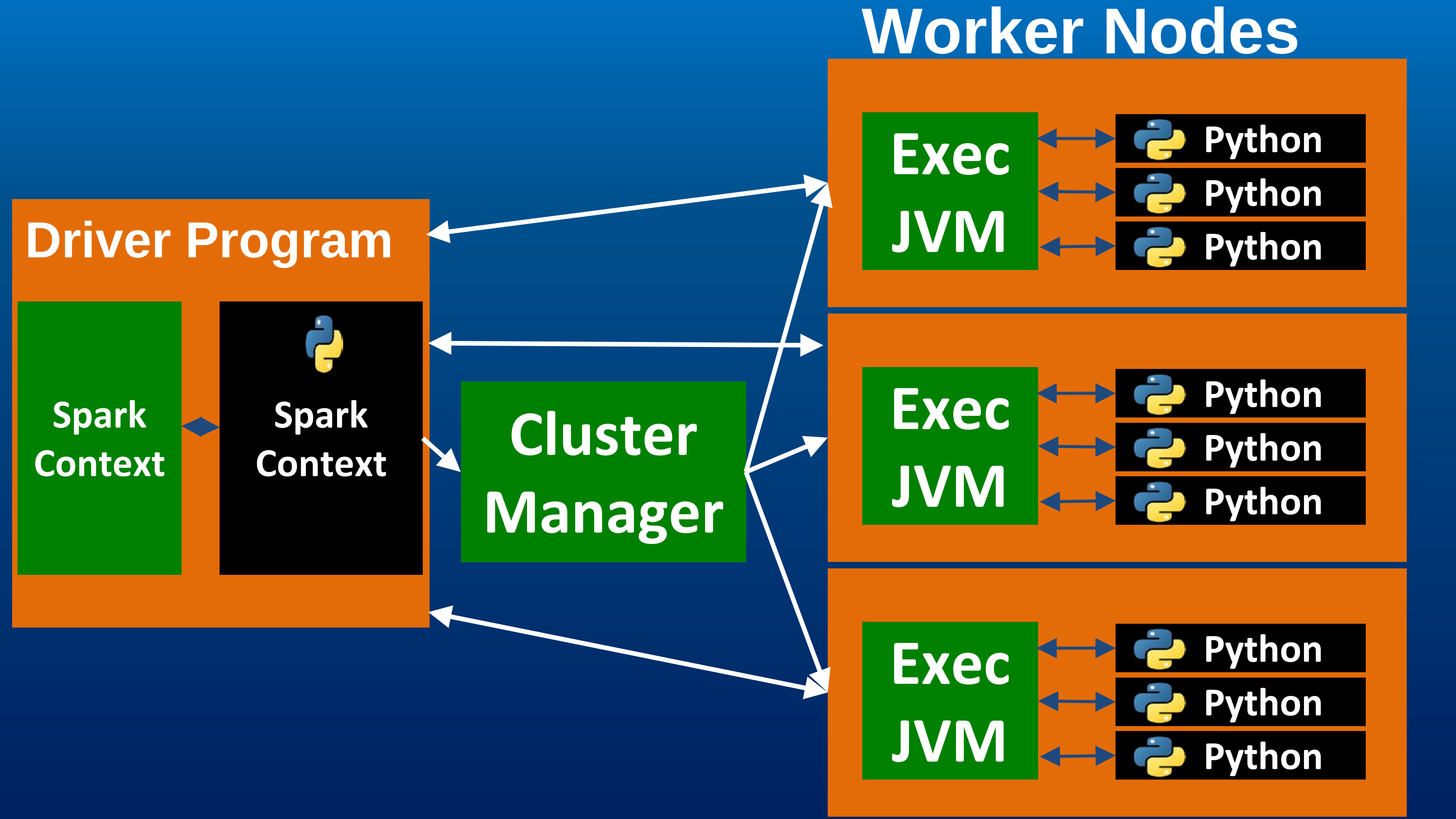
Architecture of Spark


Worker Node
Spark Executor Java Virtual Machine
Python
Python
Python
HDFS

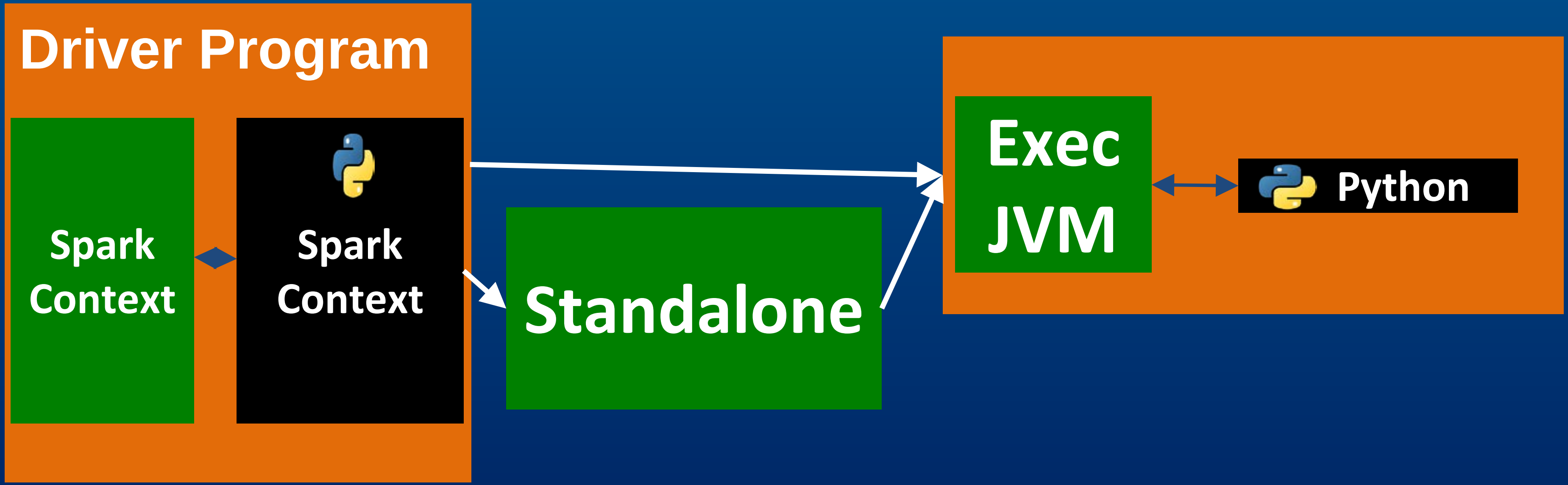
Worker Nodes

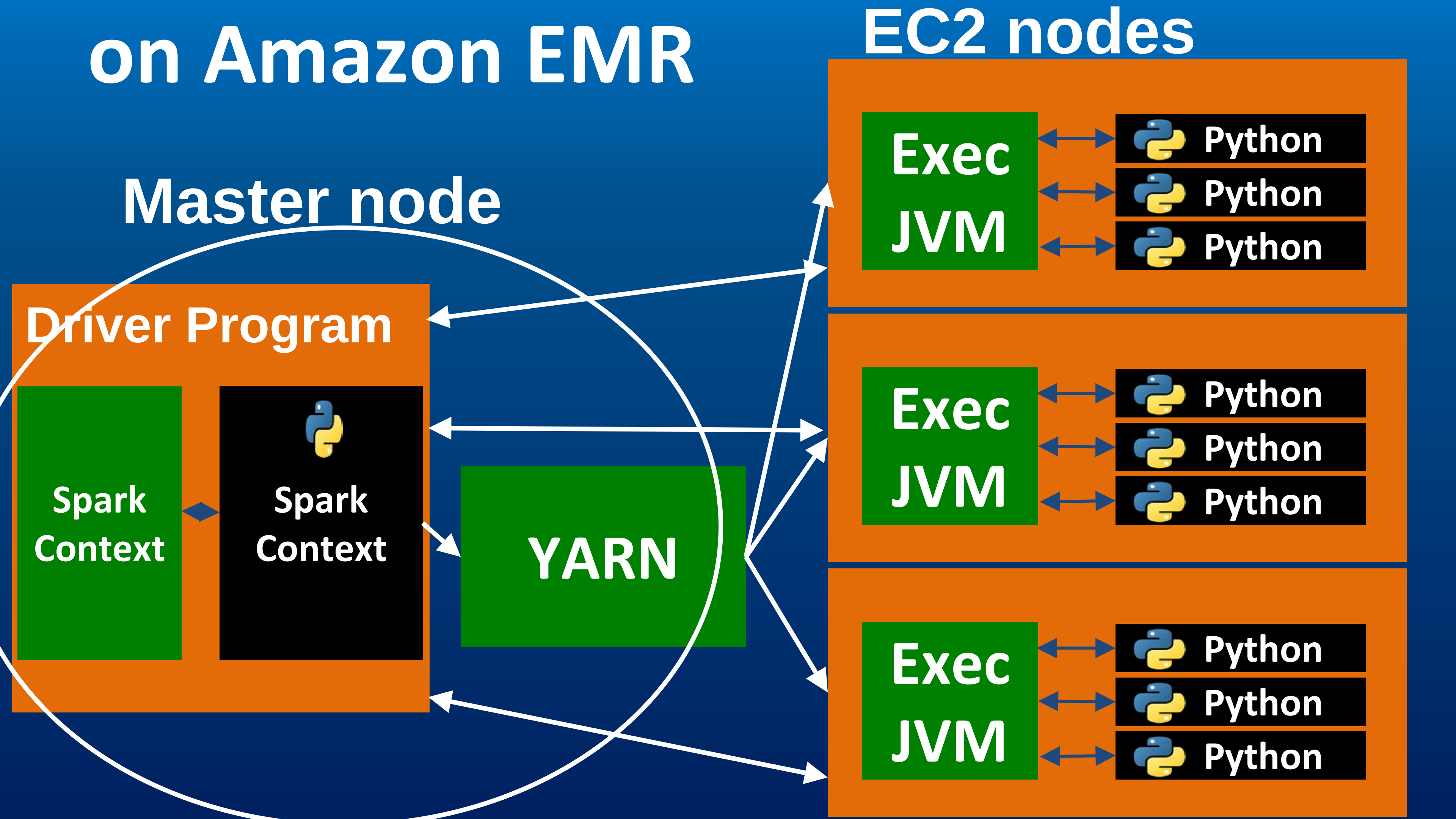



Worker Nodes
Cluster Manager
YARN/Standalone
Provision/Restart Workers
Exec
JVM
Python
Python
Python
Exec
JVM
Python
Python
Python
Exec
JVM
Python
Python
Python


Worker Nodes
Driver Program
Spark Context
Spark Context
Cluster Manager
Exec JVM
Python
Python
Python
Exec JVM
Python
Python
Python
Exec JVM
Python
Python
Python

on Cloudera VM




on Amazon EMR
EC2 nodes
Master node
Driver Program
Spark Context
Spark Context
YARN
Exec JVM
Python
Python
Python
Exec JVM
Python
Python
Python
Exec JVM
Python
Python
Python

Resilient Distributed Datasets

Resilient Distributed Dataset

Dataset

Data storage created from: HDFS, S3, HBase, JSON, text, Local hierarchy of folders

Or created transforming another RDD

Resilient Distributed Dataset

Distributed

Distributed across the cluster of machines

Divided in partitions, atomic chunks of data

Resilient Distributed Dataset

Resilient

Recover from errors, e.g. node failure, slow processes

Track history of each partition, re-run

RDD in PySpark

From the PySpark console:

```
integer_RDD = sc.parallelize(range(10), 3)
```

Check partitions

Gather all data on the driver:

```
integer_RDD.collect()
```

```
Out: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Check partitions

Maintain splitting in partitions:

```
integer_RDD.glom().collect()
```

```
Out: [[0, 1, 2], [3, 4, 5], [6, 7, 8, 9]]
```

Read text into Spark

from local filesystem:

```
text_RDD =
sc.textFile("file:///home/cloudera/testfile1")
```

from HDFS:

```
text_RDD =
sc.textFile("/user/cloudera/input/testfile1")
text_RDD.take(1) #outputs the first line
```

Wordcount in Spark: map

```
def split_words(line):
    return line.split()

def create_pair(word):
    return (word, 1)

pairs_RDD=text_RDD.flatMap(split_words).map(create_pair)
```

```
pairs_RDD.collect()
Out[]: [(u'A', 1),
 (u'long', 1),
 (u'time', 1),
 (u'ago', 1),
 (u'in', 1),
 (u'a', 1),
 (u'galaxy', 1),
 (u'far', 1),
 (u'far', 1),
 (u'away', 1)]
```

Wordcount in Spark: reduce

```
def sum_counts(a, b):
    return a + b

wordcounts_RDD = pairs_RDD.reduceByKey(sum_counts)

wordcounts_RDD.collect()
```

```
Out[]:
[(u'A', 1),
 (u'ago', 1),
 (u'far', 2),
 (u'away', 1),
 (u'in', 1),
 (u'long', 1),
 (u'a', 1),
 (u'time', 1),
 (u'galaxy', 1)]
```

Transformations

Transformations

- RDD are immutable
- Never modify RDD in place
- Transform RDD to another RDD
- Lazy

Create RDD

from local filesystem:

```
text_RDD =
sc.textFile("file:///home/cloudera/testfile1")
```

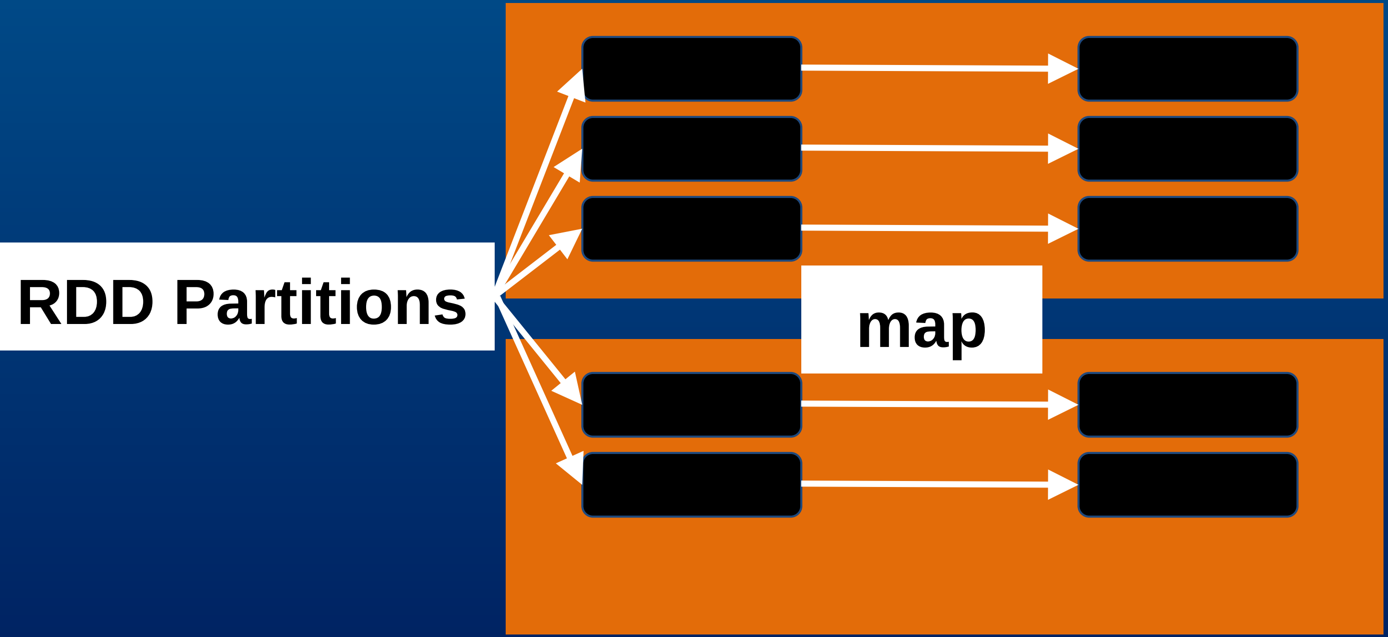
Apply a transformation: map

`map` : apply function to each element of RDD

```
def lower(line):
    return line.lower()
lower_text_RDD = text_RDD.map(lower)
```

map

`map` : apply function to each element of RDD



Other transformations

- flatMap(func) - map then flatten output
- filter(func) - keep only elements where func is true
- sample(withReplacement, fraction, seed) - get a random data fraction
- coalesce(numPartitions) - merge partitions to reduce them to numPartitions

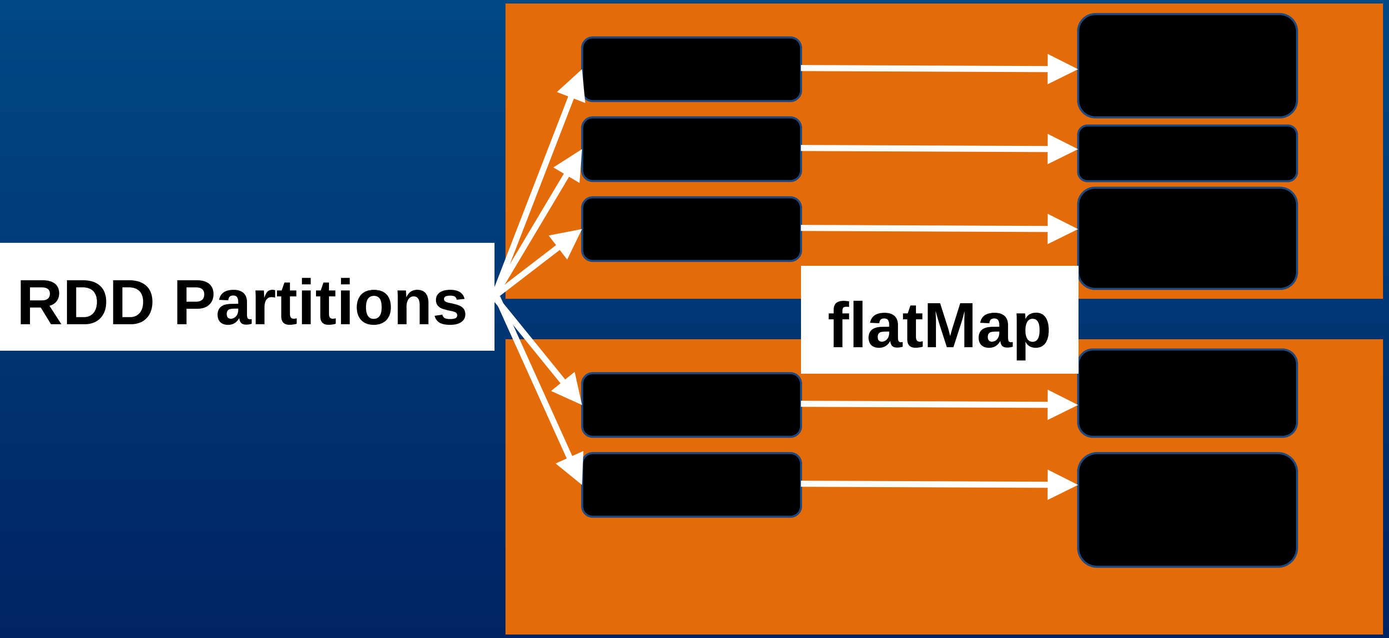
flatMap

```
def split_words(line):
    return line.split()

words_RDD = 
text_RDD.flatMap(split_words)
words_RDD.collect()
```

```
Out[]:
[u'A',
 u'long',
 u'time',
 u'ago',
 u'in',
 u'a',
 u'galaxy',
 u'far',
 u'far',
 u'away']
```

flatMap

flatMap : map then flatten output



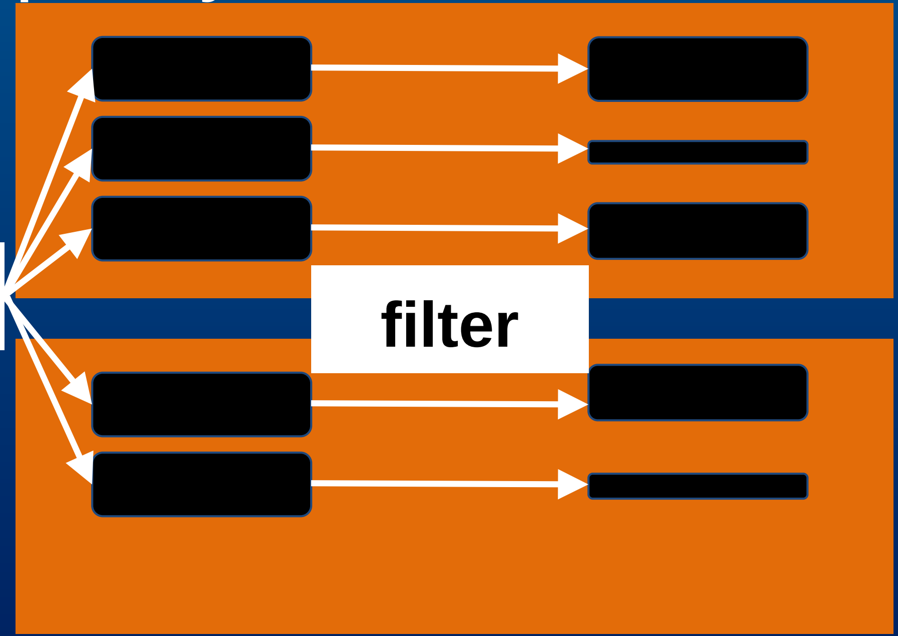
filter

```
def starts_with_a(word):
    return word.lower().startswith("a")
words_RDD.filter(starts_with_a).collect()
```

Out[]: [u'A', u'ago', u'a', u'away']

filter

`filter` : keep only elements where func is true

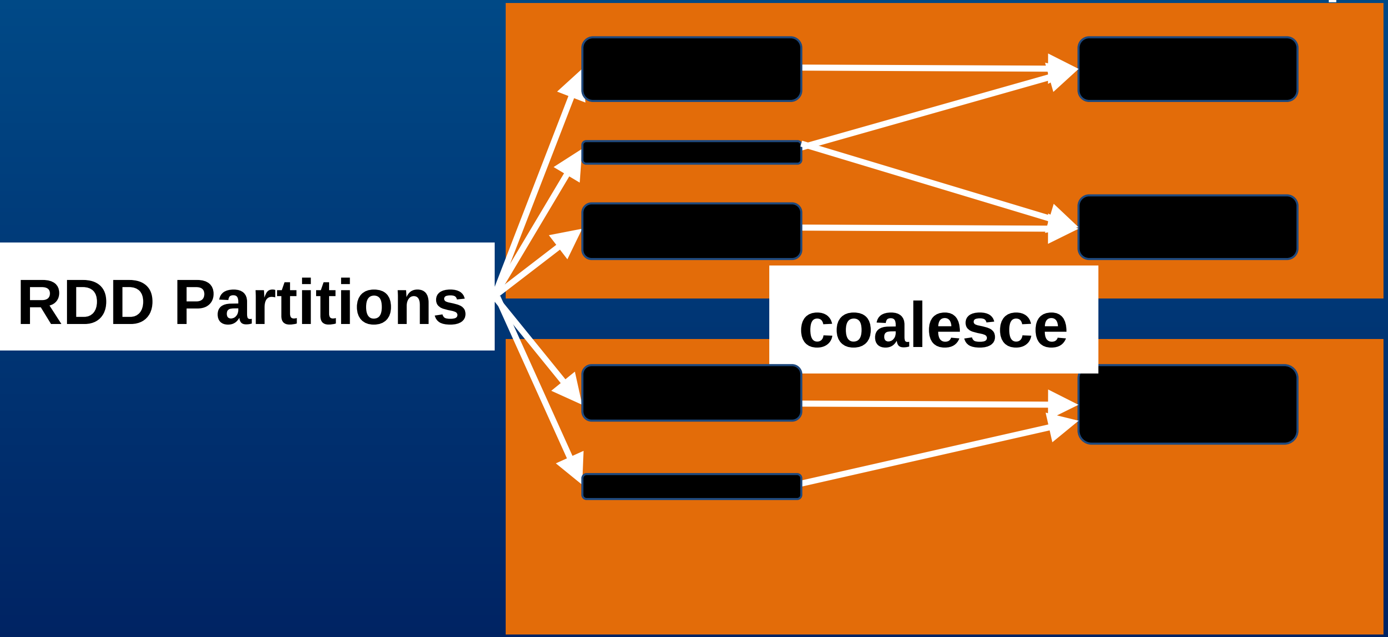


coalesce

```
sc.parallelize(range(10), 4).glom().collect()

Out[]: [[0, 1], [2, 3], [4, 5], [6, 7, 8, 9]]

sc.parallelize(range(10), 4).coalesce(2).glom().collect()

Out[]: [[0, 1, 2, 3], [4, 5, 6, 7, 8, 9]]
```

coalesce

coalesce : reduce the number of partitions



Wide Transformations

Transformations of (K,V) pairs

```
def create_pair(word):
    return (word, 1)
```

```
pairs_RDD=text_RDD.flatMap(split_words).map(create_pair)
```

```
pairs_RDD.collect()
```

```
Out[]: [(u'A', 1),
 (u'long', 1),
 (u'time', 1),
 (u'ago', 1),
 (u'in', 1),
 (u'a', 1),
 (u'galaxy', 1),
 (u'far', 1),
 (u'far', 1),
 (u'away', 1)]
```

groupByKey

groupByKey : (K, V) pairs => (K, iterable of all V)

(A, 1)

(B, 8)

(A, 2)

(A, 5)

(A, [1, 2, 5])

(B, [8])

```
pairs_RDD.groupByKey().collect()
```

```
Out[]: [(u'A', <pyspark.resultiterable.ResultIterable at XXX>),
 (u'ago', <pyspark.resultiterable.ResultIterable at XXX>),
 (u'far', <pyspark.resultiterable.ResultIterable at XXX>),
 (u'away', <pyspark.resultiterable.ResultIterable at XXX>),
 (u'in', <pyspark.resultiterable.ResultIterable at XXX>),
 (u'long', <pyspark.resultiterable.ResultIterable at XXX>),
 (u'a', <pyspark.resultiterable.ResultIterable at XXX>),<
<MORE output>
```

```
for k,v in pairs_RDD.groupByKey().collect():
    print "Key:", k, ",Values:", list(v)
```

Out[]: Key: A , Values: [1]

Key: ago , Values: [1]

Key: far , Values: [1, 1]

Key: away , Values: [1]

Key: in , Values: [1]

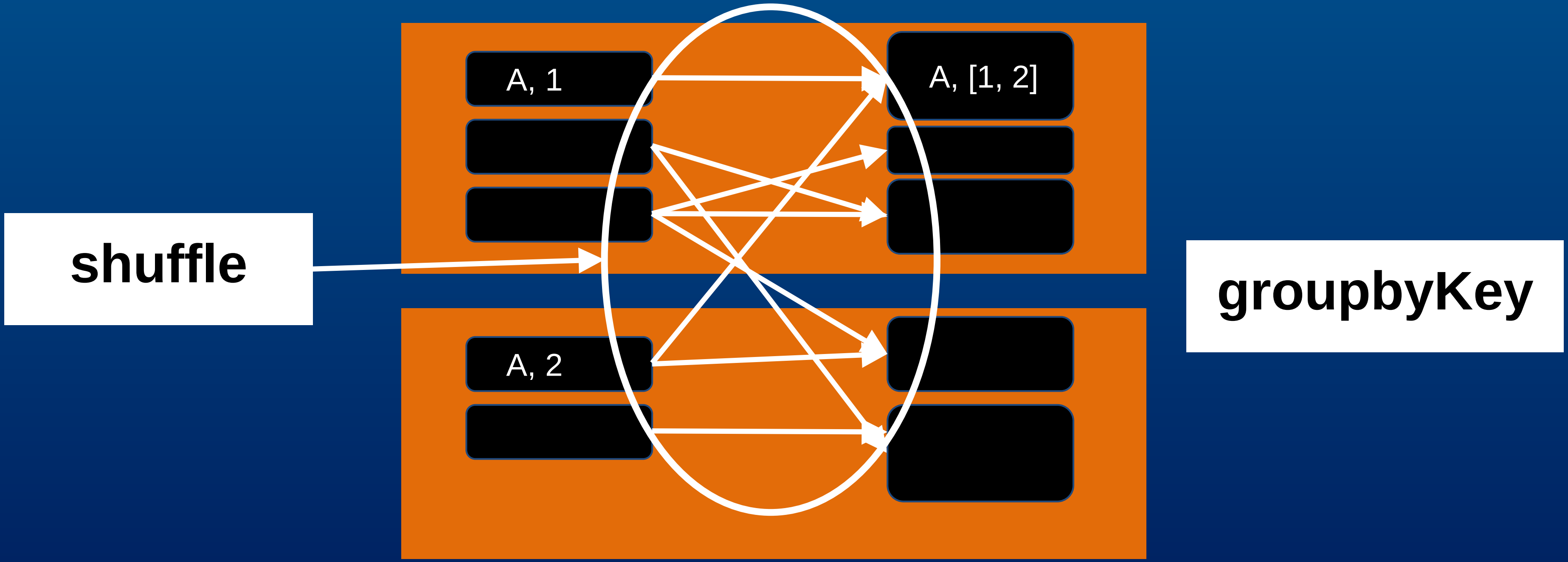
Key: long , Values: [1]

Key: a , Values: [1]

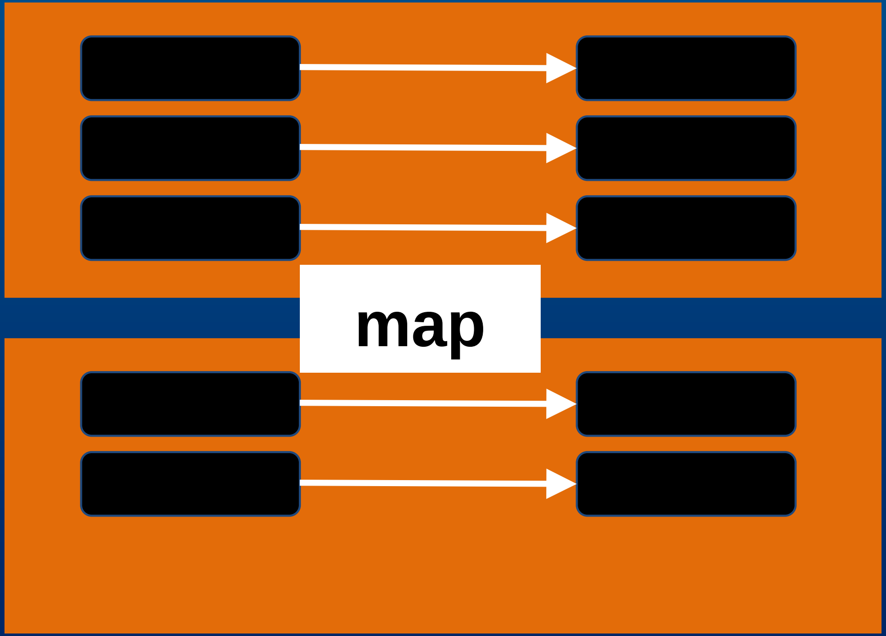
<MORE output>

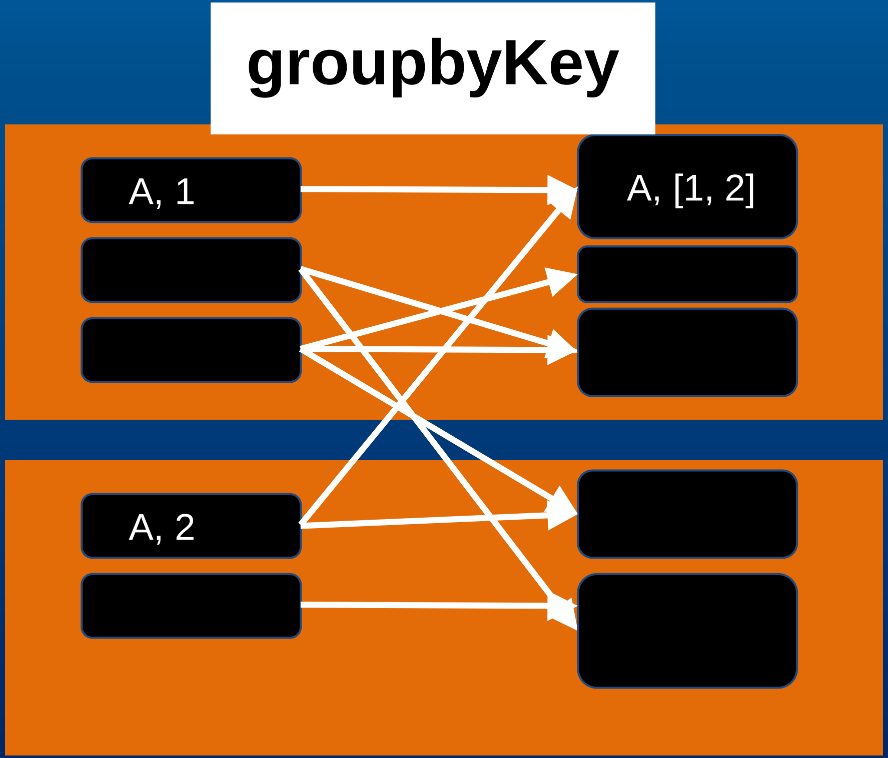
groupByKey

groupByKey : (K, V) pairs => (K, iterable of all V)



Narrow vs Wide





Wide transformations

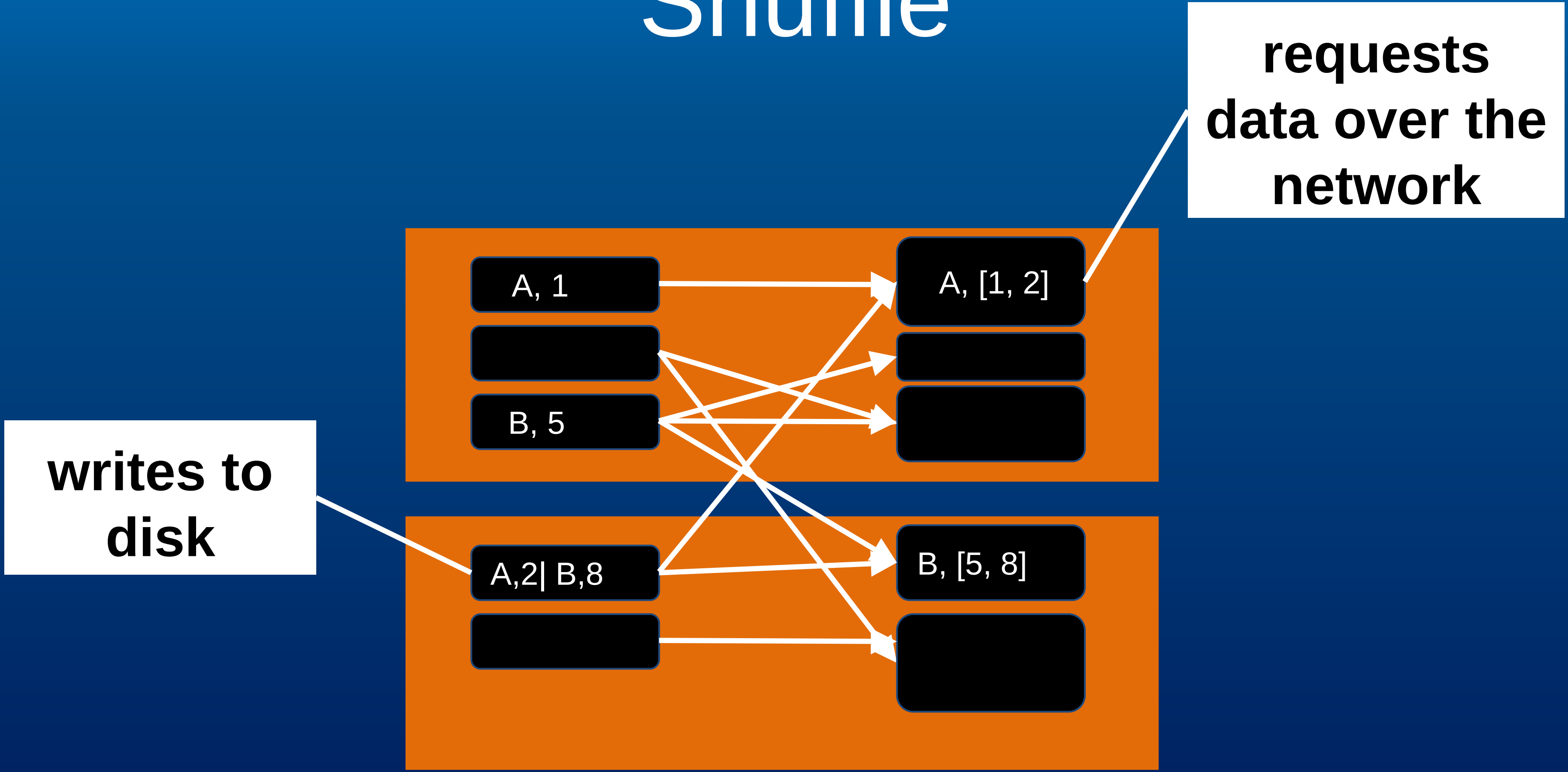
- `groupByKey` : (K, V) pairs => (K, iterable of all V)
- `reduceByKey(func)` : (K, V) pairs => (K, result of reduction by func on all V)
- `repartition(numPartitions)` : similar to coalesce, shuffles all data to increase or decrease number of partitions to numPartitions

Shuffle

Shuffle

- Global redistribution of data
- High impact on performance

Shuffle



requests data over the network
A, 1
B, 5
A, [1, 2]
writes to disk
A,2| B,8
B, [5, 8]

Know shuffle, avoid it

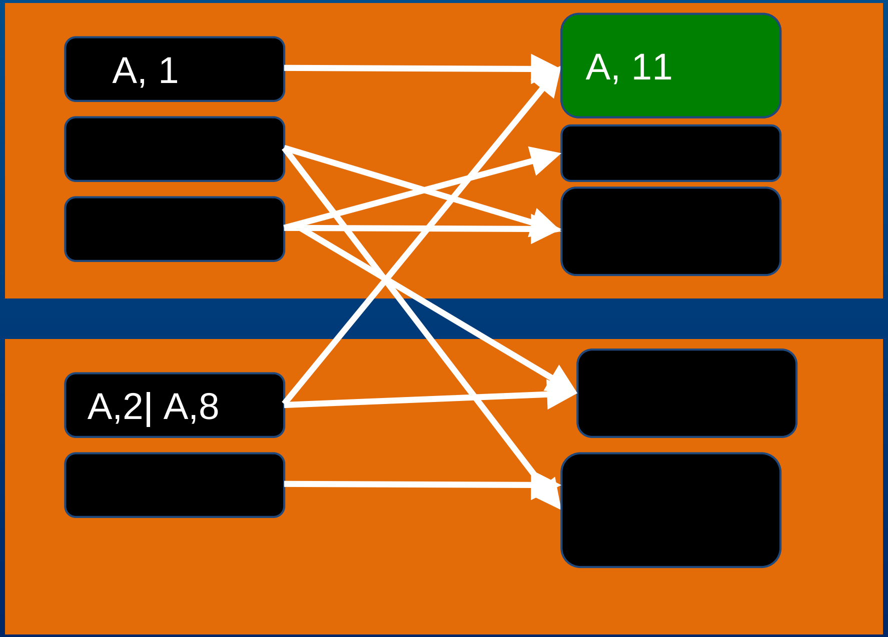
- Which operations cause it?
- Is it necessary?

Really need groupByKey?

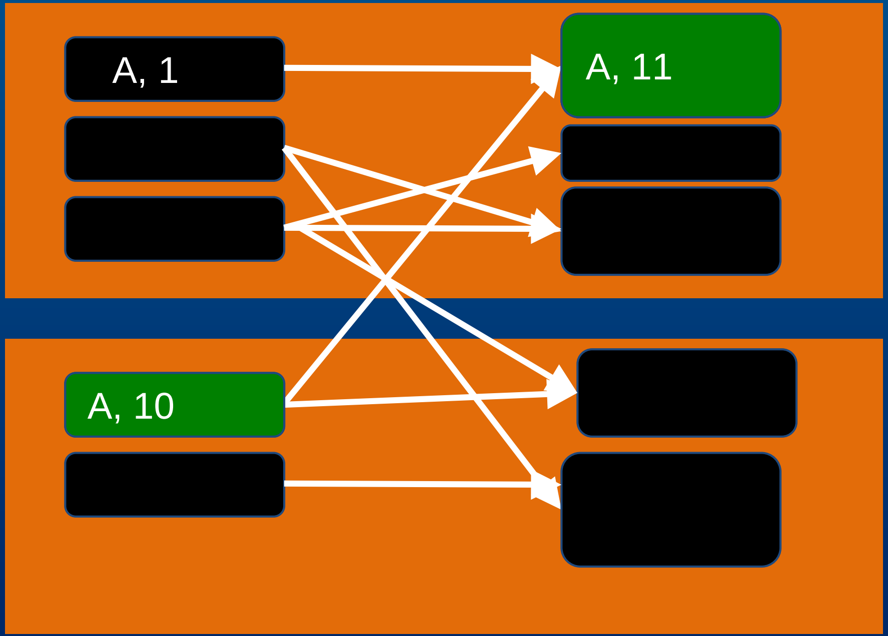
groupByKey: (K, V) pairs => (K, iterable of all V)

if you plan to call reduce later in the pipeline,
use reduceByKey instead.

groupByKey + reduce

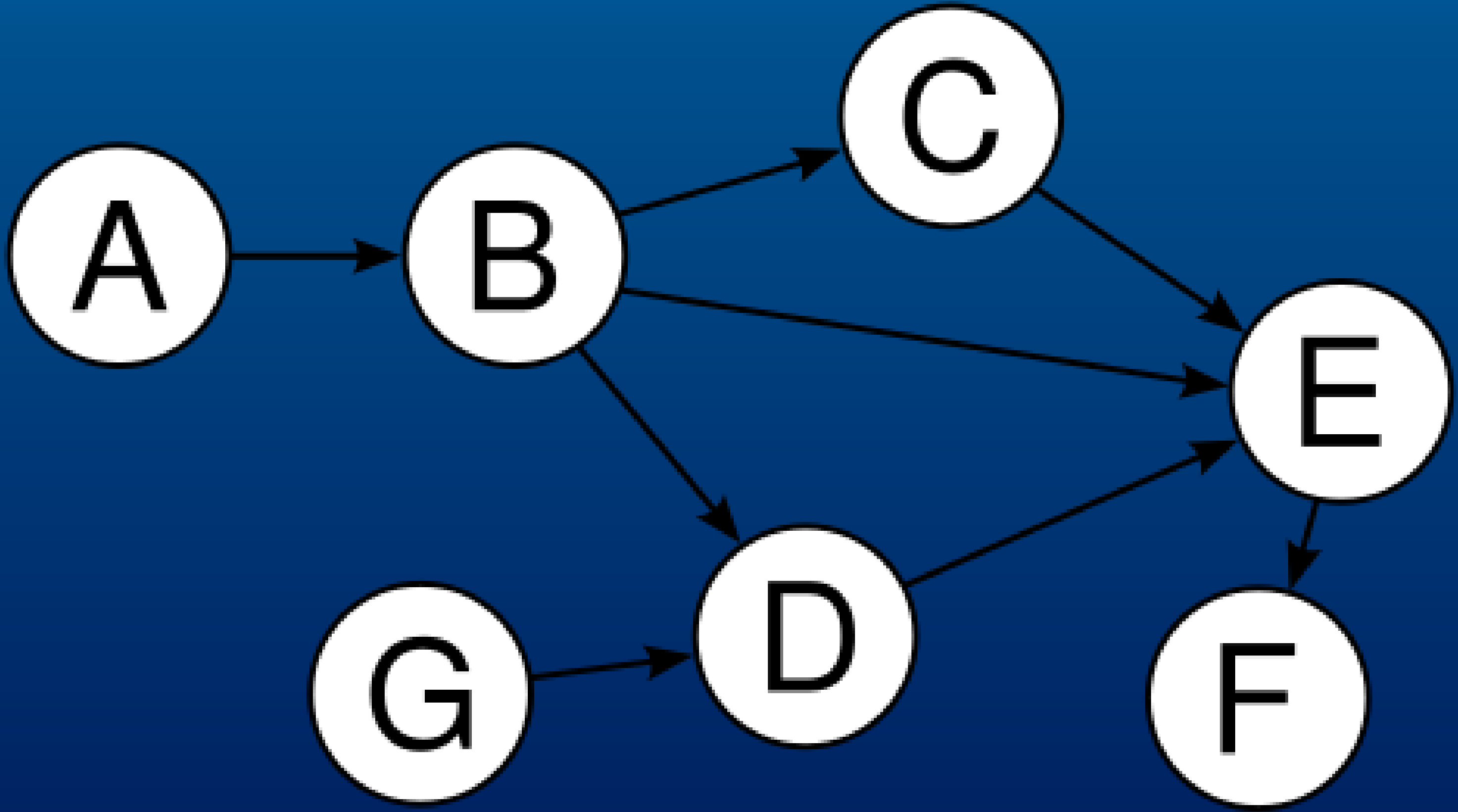


reduceByKey



Directed Acyclic Graph Scheduler

Directed Acyclic Graphs



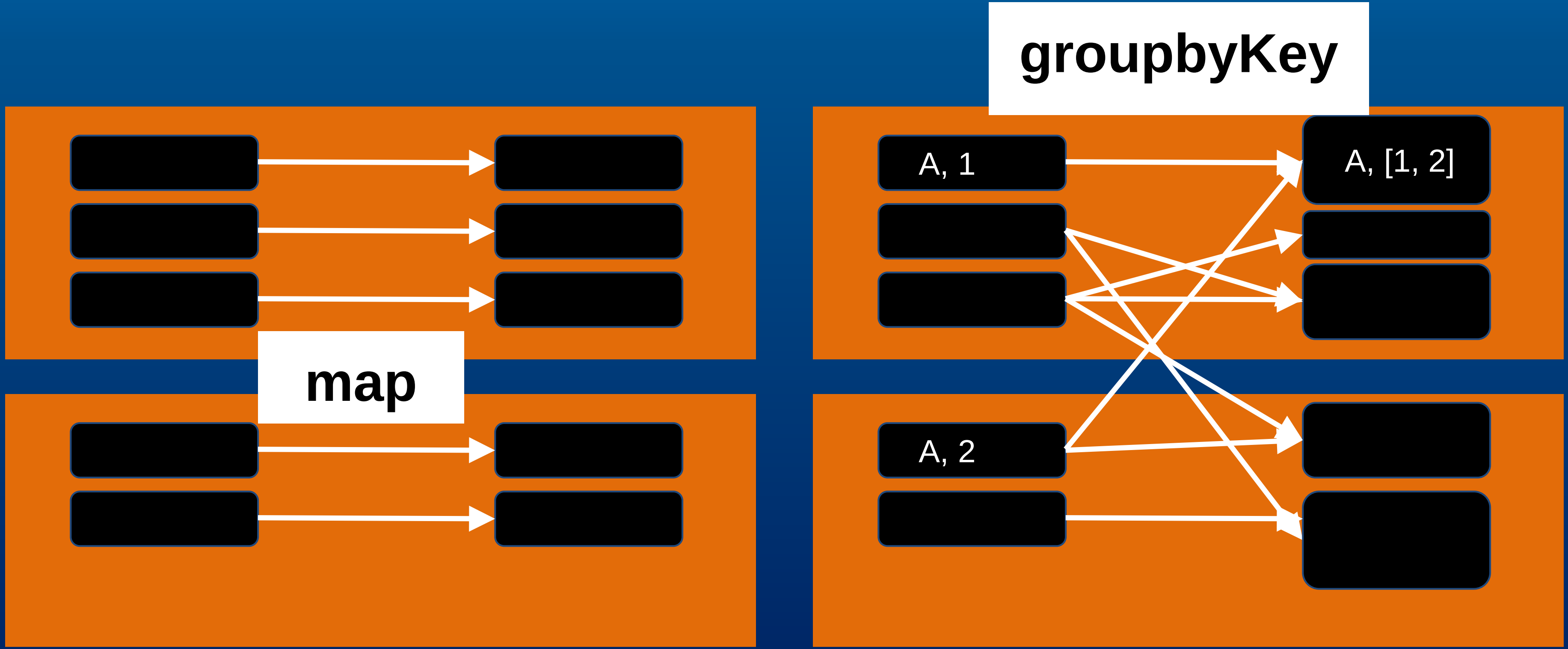
Directed Acyclic Graphs

Track **dependencies**!
(also known as lineage or provenance)

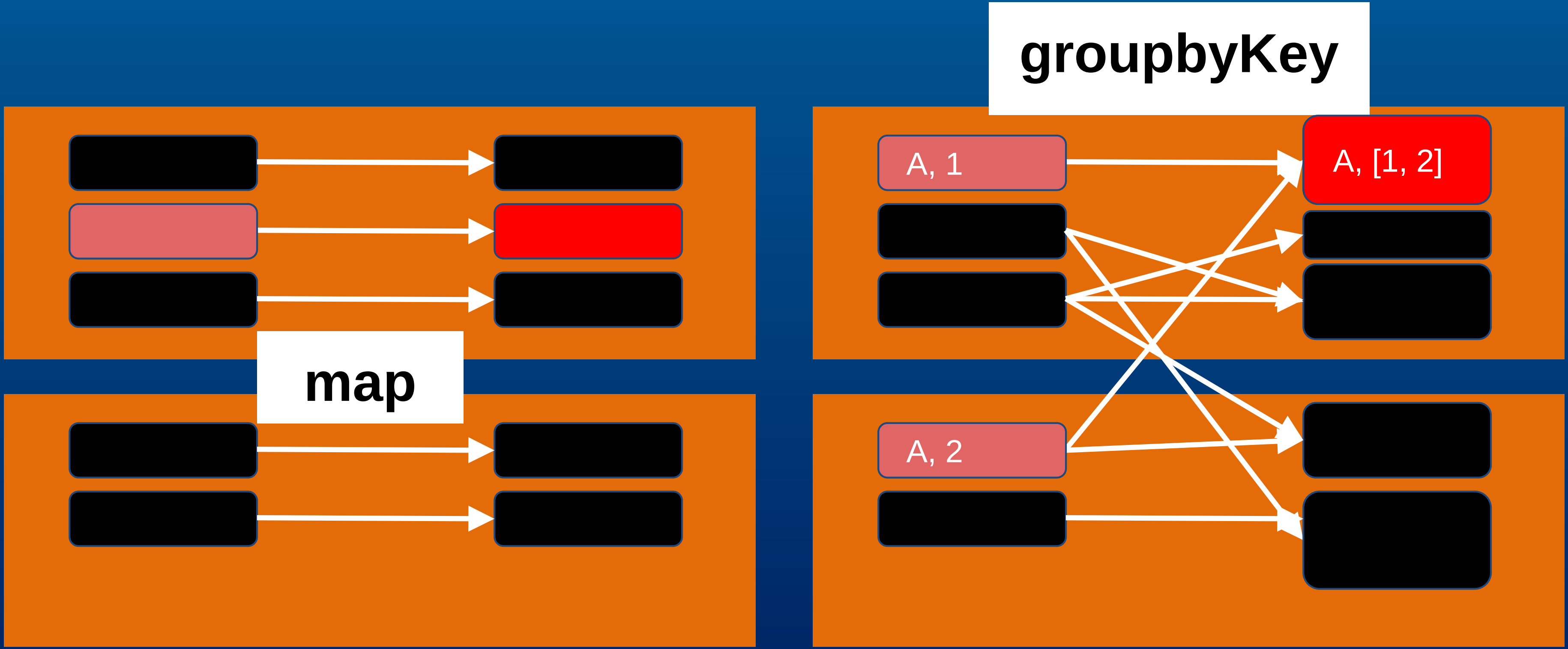
DAG in Spark

- nodes are RDDs
- arrows are Transformations

Narrow vs Wide



Narrow vs Wide



Transformations of (K,V) pairs

```
def create_pair(word):
    return (word, 1)
```

```
pairs_RDD=text_RDD.flatMap(split_words).map(create_pair)
```

```
for k,v in pairs_RDD.groupByKey().collect():
    print "Key:", k, ",Values:", list(v)
```

Out[]: Key: A , Values: [1]

Key: ago , Values: [1]

Key: far , Values: [1, 1]

Key: away , Values: [1]

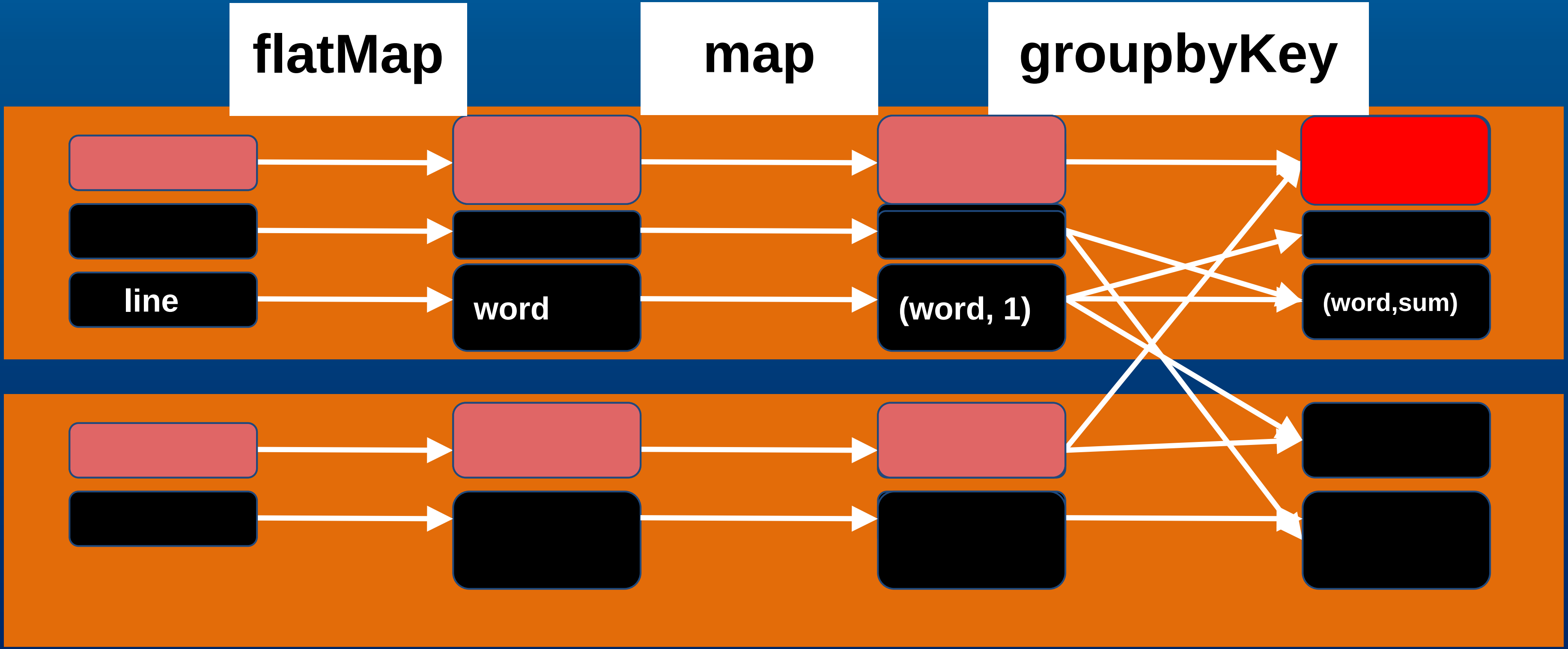
Key: in , Values: [1]

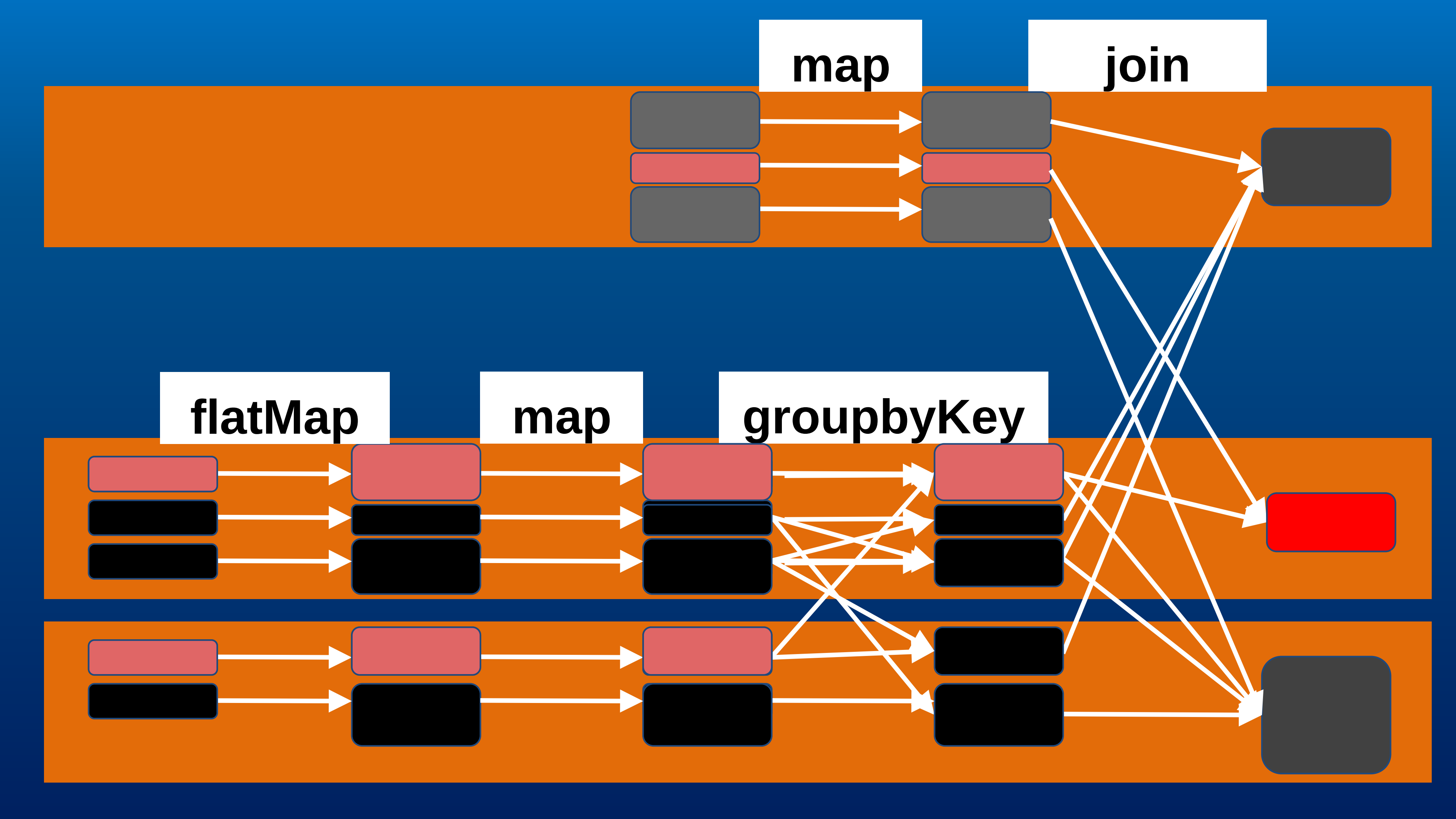
Key: long , Values: [1]

Key: a , Values: [1]

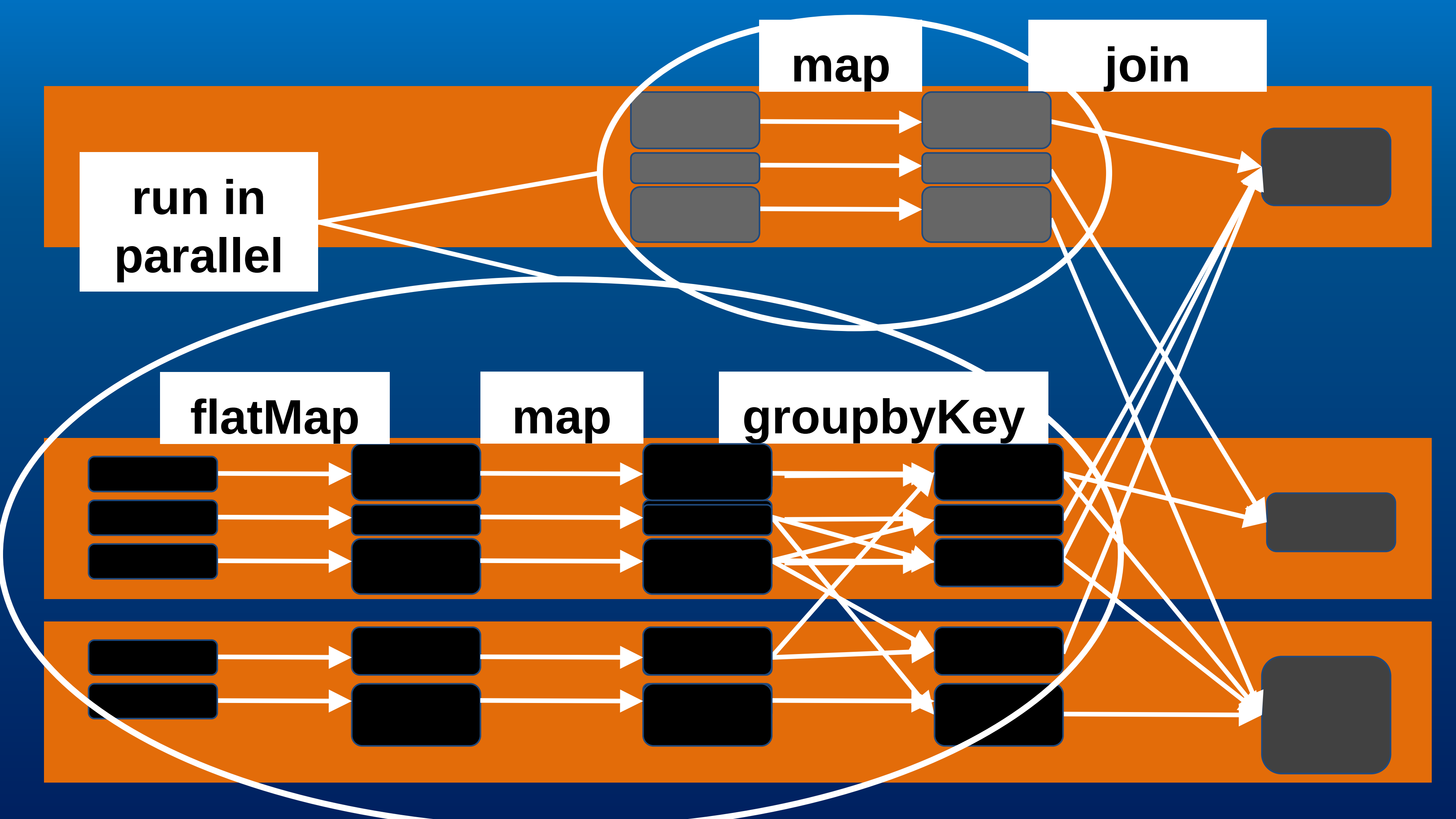
<MORE output>

Spark DAG of transformations





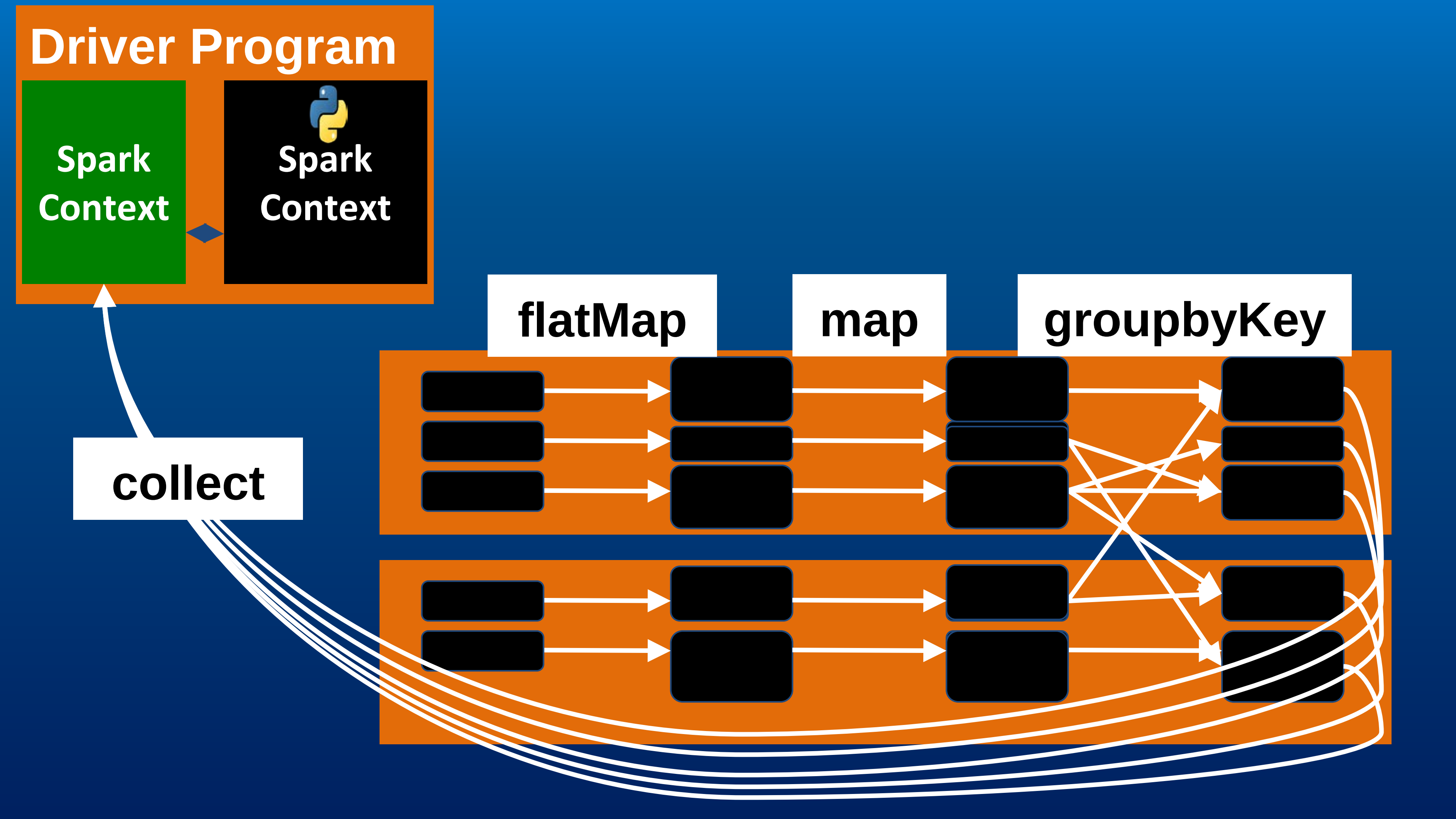
map
join
flatMap
map
groupbyKey


map
join
run in
parallel
flatMap
map
groupbyKey

Actions

What is an action

- Final stage of workflow
- Triggers execution of the DAG
- Returns results to the Driver or writes to HDFS


Driver Program
Spark Context
Spark Context
flatMap
map
groupbyKey
collect

Actions

- collect() - copy all elements to the driver
- take(n) - copy first n elements
- reduce(func) - aggregate elements with func (takes 2 elements, returns 1)
- saveAsTextFile(filename) - save to local file or HDFS

Caching

Caching

- By default each job re-processes from HDFS
- Mark RDD with `.cache()`
- Lazy

When?

- Generally not the input data
- Do validation and cleaning
- Cache for iterative algorithm

How?

- Memory (most common)
- Disk (rare)
- Both (for heavy calculations)

Speedup

- Easily 10x or even 100x depending on application
- Caching is gradual
- Fault tolerant

Wordcount with caching

from HDFS:

```
text_RDD =
sc.textFile("/user/cloudera/input/testfile1")
```

```python
def split_words(line):
    return line.split()

def create_pair(word):
    return (word, 1)

pairs_RDD=text_RDD.flatMap(split_words).map(create_pair)
pairs_RDD.cache()
```

```
def sum_counts(a, b):
    return a + b

wordcounts_RDD = pairs_RDD.reduceByKey(sum_counts)

```

First job:

```
wordcounts_RDD.collect()
```

Second job:

```
pairs_RDD.take(1)
```

Broadcast variables

Broadcast variables

- Large variable used in all nodes
- Transfer just once per Executor
- Efficient peer-to-peer transfer

Broadcast variable example

For example large configuration dictionary or lookup table:

```
config = sc.broadcast({"order":3, "filter":True})
config.value
```

Accumulators

Accumulator

- Common pattern of accumulating to a variable across the cluster
- Write-only on nodes

Accumulator example

```
accum = sc.accumulator(0)
def test_accum(x):
    accum.add(x)

sc.parallelize([1, 2, 3, 4]).foreach(test_accum)

accum.value
Out[]: 10
```

Big Data Analytics in Apache Spark

Big Data Analytics with Spark

- Spark Dataframes to work with tabular data
- Data cleaning, summary, statistics
- Spark Dataframes with SQL and Hive

Open PySpark

```
PYSPARK_DRIVER_PYTHON=ipython pyspark
```

Introduction to Spark Dataframes

Types of RDD: text

from local filesystem:

```
text_RDD =
sc.textFile("file:///home/cloudera/testfile1")
```

```
text_RDD.collect()
Out[]: [u'A long time ago in a galaxy far far away']
```

Types of RDD: key-value pairs

```
def split_words(line):
    return line.split()

def create_pair(word):
    return (word, 1)

pairs_RDD=text_RDD.flatMap(split_words
).map(create_pair)
```

```
pairs_RDD.collect()
Out[]: [(u'A', 1),
 (u'long', 1),
 (u'time', 1),
 (u'ago', 1),
 (u'in', 1),
 (u'a', 1),
 (u'galaxy', 1),
 (u'far', 1),
 (u'far', 1),
 (u'away', 1)]
```

Tabular dataset

Most real-world datasets have
records (rows)

each with
multiple values (columns)

Tweets

user	text	datetime	favorites	retweets
andreazonca	"spark is cool"	"2015-10-1 9:04"	5	3

Reviews

business	text	datetime	starts	user
Pan Bon	"great pizza!"	"2015-10-1 9:04"	5	andreazonca

Logs

http_code	ip	datetime	user_agent
200	127.0.0.1	"2015-10-1 9:04"	Firefox

Tabular datasets

```
students = sc.parallelize([
[100, "Alice", 8.5, "Computer Science"],
[101, "Bob", 7.1, "Engineering"],
[102, "Carl", 6.2, "Engineering"]
])
```

Mean of a column

```
def extract_grade(row):
    return row[2]

students.map(extract_grade).mean()
Out[]: 17.26666
```

Group by column

```
def extract_degree_grade(row):
    return (row[3], row[2])

degree_grade_RDD =
students.map(extract_degree_grade)
degree_grade_RDD.collect()
```

Group by column

Intermediate RDD:

```
degree_grade_RDD.collect()
Out[]:
[('Computer Science', 8.5),
 ('Engineering', 7.0999999999999996),
 ('Engineering', 6.2000000000000002)]
```

Group by column

Reduce by key to get the final result:

```
degree_grade_RDD.reduceByKey(max).collect()
```

```
Out[]:
[('Engineering', 7.0999999999999996),
 ('Computer Science', 8.5)]
```

Introducing Spark Dataframes

User friendly interface

Under-the-hood optimization for table-like datasets

```
students_df = sqlCtx.createDataFrame(students,
    ["id", "name", "grade", "degree"])

students_df.printSchema()
root
 |-- id: long (nullable = true)
 |-- name: string (nullable = true)
 |-- grade: double (nullable = true)
 |-- degree: string (nullable = true)
```

sqlCtx.createDataFrame?

Create a DataFrame from an RDD of tuple/list, list or pandas.DataFrame.

`schema` could be :class:`StructType` or a list of column names.

When `schema` is a list of column names, the type of each column
will be inferred from `rdd`.

When `schema` is None, it will try to infer the column name and type
from `rdd`, which should be an RDD of :class:`Row`, or namedtuple,
or dict.

If referring needed, `samplingRatio` is used to determined how many
rows will be used to do referring. The first row will be used if
`samplingRatio` is None.

:param data: an RDD of Row/tuple/list/dict, list, or pandas.DataFrame
:param schema: a StructType or list of names of columns
:param samplingRatio: the sample ratio of rows used for inferring
:return: a DataFrame

```
>>> l = [('Alice', 1)]
>>> sqlCtx.createDataFrame(l).collect()
[Row(_1=u'Alice', _2=1)]
>>> sqlCtx.createDataFrame(l, ['name', 'age']).collect()
[Row(name=u'Alice', age=1)]
```

Mean of a column

```
students_df.agg({"grade": "mean"}).collect()
```

```
Out[]: [Row(AVG(grade#30)=7.266666666666666)]
```

Find all available operations:

[redacted]

Group by column

```
students_df.groupBy("degree").max("grade").collect()
Out[]:
[Row(degree=u'Computer Science',
MAX(grade#30)=8.5),
 Row(degree=u'Engineering',
MAX(grade#30)=7.09999999999)]
```

Pretty print with show

```
students_df.groupBy("degree").max("grade").show()
```

degree	MAX(grade#30)
Computer Science	8.5
Engineering	7.1

Final remarks on Dataframes

- special kind of RDD
- transformations/actions/DAG work the same way
- automatic optimization to Java bytecode
- Python as fast as Scala/Java

Create Spark Dataframes

Specify a Schema

```
students_df = sqlCtx.createDataFrame(students,
    ["id", "name", "grade", "degree"])
```

```python
from pyspark.sql.types import *
schema = StructType([
StructField("id", LongType(), True),
StructField("name", StringType(), True),
StructField("grade", DoubleType(), True),
StructField("degree", StringType(), True) ])

students_df = sqlCtx.createDataFrame(students, schema)
```

```
students_df.printSchema()

root
 |-- id: long (nullable = true)
 |-- name: string (nullable = true)
 |-- grade: double (nullable = true)
 |-- degree: string (nullable = true)
```

Load a JSON file

```
students_json = [
'{"id":100, "name":"Alice", "grade":8.5,
"degree":"Computer Science"}',
'{"id":101, "name":"Bob", "grade":7.1,
"degree":"Engineering"}']
with open("students.json", "w") as f:
    f.write("\n".join(students_json))
```

Dump JSON file conent

```
!cat students.json

{"id":100, "name":"Alice", "grade":8.5, "degree":"Computer Science"}
{"id":101, "name":"Bob", "grade":7.1, "degree":"Engineering"}
```

Create Dataframe with jsonFile

```
sqlCtx.jsonFile("file:///home/cloudera/students.json").show()

degree              grade  id    name
Computer Science    8.5    100   Alice
Engineering         7.1    101   Bob
```

Load Dataframe from CSV

- Not included in Spark
- Load from spark-packages.org

spark-packages.org

Spark Packages

Feedback Register a package Login Find a package

A community index of packages for Apache Spark.

140 package

All (140) Core (5) Data Sources (22) Machine Learning (35) Streaming (21) Graph (5) PySpark (2) Applications (5) Deployment (8) Examples (8) Tools (13)

spark-avro

Integration utilities for using Spark with Apache Avro data

from: @databricks / owner: @pwendell / Latest release: 2.0.1-s_2.10 (2015-09-08) / Apache-2.0 / (8)

4 sql 3 input 3 avro

spark-redshift

Spark and Redshift integration

from: @databricks / owner: @pwendell / Latest release: 0.5.2 (2015-10-23) / Apache-2.0 / (3)

1 input 1 sql 1 redshift

kafka-spark-consumer

Receiver Based Low Level Kafka-Spark Consumer with builtin Back-Pressure Controller

@dibbhatt / Latest release: 1.0.5 (2015-10-08) / Apache-2.0 / (5)

3 streaming 2 kafka

thunder

Large-scale neural data analysis with Spark

@freeman-lab / Latest release: 0.4.1 (2014-11-27) / Apache-2.0 / (4)

spark-packages.org/package/databricks/spark-csv

Spark Packages

Feedback Register a package Login Find a pa

spark-csv (homepage)

Spark SQL CSV data source

from: @databricks / owner: @falaki / ★★★★★ (7)

This packages adds a new CSV data source to Spark SQL. CSV files can be read as Schema RDD and registered as table. Supports quotes and headers.

Tags

1 sql | 1 SparkSQL | 1 DataSource | 1 csv

How to [+]

Include this package in your Spark Applications using:

spark-shell, pyspark, or spark-submit

```
> $SPARK_HOME/bin/spark-shell --packages com.databricks:spark-csv_2.11:1.2.0
```

Releases

Version: 1.2.0-s_2.11 (82344b | zip | jar) / Date: 2015-08-07 / License: Apache-2.0 / Scala version: 2.11

Spark Scala/Java API compatibility: 1.2.0 - 43% , 1.3.0 - 77% , 1.4.0 - 100%

Version: 1.2.0-s_2.10 (82344b | zip | jar) / Date: 2015-08-07 / License: Apache-2.0 / Scala version: 2.10

Spark Scala/Java API compatibility: 1.0.0 - 11% , 1.1.0 - 38% , 1.2.0 - 43% , 1.3.0 - 77% , 1.4.0 - 100%

Version: 1.0.3 (464a3e | zip | jar) / Date: 2015-04-04 / License: Apache-2.0 / Scala version: 2.11

Spark Scala/Java API compatibility: 1.2.0 - 43% , 1.3.0 - 100%

Restart PySpark

```
PYSPARK_DRIVER_PYTHON=ipython pyspark --packages com.databricks:spark-csv_2.10:1.2.0
```

Automatically download and include new packages and dependencies

Load sample yelp csv

```
yelp_df = sqlCtx.load(
source="com.databricks.spark.csv",
header = 'true',
inferSchema = 'true',
path =
'file:///usr/lib/hue/apps/search/examples/collections/solr_co
nfigs_yelp_demo/index_data.csv')
```

```
yelp_df.printSchema()
root
 |-- business_id: string (nullable = true)
 |-- cool: integer (nullable = true)
 |-- date: string (nullable = true)
 |-- funny: integer (nullable = true)
 |-- id: string (nullable = true)
 |-- stars: integer (nullable = true)
 |-- text: string (nullable = true)
 |-- type: string (nullable = true)
 |-- useful: integer (nullable = true)
 |-- user_id: string (nullable = true)
 |-- name: string (nullable = true)
 |-- full_address: string (nullable = true)
 |-- latitude: double (nullable = true)
 |-- longitude: double (nullable = true)
 |-- neighborhoods: string (nullable = true)
 |-- open: string (nullable = true)
 |-- review_count: integer (nullable = true)
 |-- state: string (nullable = true)
```

```
yelp_df.count()
Out[]: 1000L
```

Analytics with Dataframes on a Yelp reviews dataset

Explore the Yelp dataset

```
yelp_df = sqlCtx.load(
source='com.databricks.spark.csv',
header = 'true',
inferSchema = 'true',
path =
'file:///usr/lib/hue/apps/search/examples/collections/solr_configs_yelp_demo/index_data.csv')
```

Reference a column

As attribute:

```
yelp_df.useful
        Out[]: Column<useful>
```

As key:

```
yelp_df["useful"]
        Out[]: Column<useful>
```

Filtering

```
yelp_df.filter(yelp_df.useful >= 1).count()

yelp_df.filter(yelp_df["useful"] >= 1).count()

yelp_df.filter("useful >= 1").count()

Out[]: 601L
```

select

```
yelp_df["useful"].agg({"useful":"max"}).collect()
Out[]: AttributeError: 'Column' object has no attribute 'agg'

yelp_df.select("useful")
Out[]: DataFrame[useful: int]

yelp_df.select("useful").agg({"useful":"max"}).collect()
Out[]: [Row(MAX(useful#267)=28)]
```

Create a modified DataFrame

Rescale the useful column from 0-28 to 0-100.

Create a 2 columns DataFrame

```
yelp_df.select("id", "useful").take(5)

[Row(id=u'fWKvX83p0-ka4JS3dc6E5A', useful=5),
 Row(id=u'IjZ33sJrzXqU-0X6U8NwyA', useful=0),
 Row(id=u'IESLBzqUCLdSzSqm0eCSxQ', useful=1),
 Row(id=u'G-WvGaISbqqaMHlNnByodA', useful=2),
 Row(id=u'1uJFq2r5QfJG_6ExMRCaGw', useful=0)]

```

Modify column

```
yelp_df.select("id", yelp_df.useful/28*100).show(5)
|
id                       ((useful / 28) * 100)
fWKvX83p0-ka4JS3d...  17.857142857142858
IjZ33sJrzXqU-0X6U...     0.0
IESLBzqUCLdSzSqm0... 3.5714285714285716
G-WvGaISbqqaMHlNn... 7.142857142857142
1uJFq2r5QfJG_6ExM...  0.0
|
```

Cast (truncate) to integer

```
yelp_df.select("id",
(yelp_df.useful/28*100).cast("int")).show(5)

id                    CAST(((useful / 28) * 100)), IntegerType)
fWKvX83p0-ka4JS3d... 17
IjZ33sJrzXqU-0X6U...     0
IESLBzqUCLdSzSqm0... 3
G-WvGaISbqqaMHINn... 7
1uJFq2r5QfJG_6ExM... 0
```

Save as new dataframe

```
useful_perc_data = yelp_df.select(
    "id",
    (yelp_df.useful/28*100).cast("int")
)
```

```
useful_perc_data.columns
```

```
Out[]: [u'id', u'CAST(((useful / 28) * 100), IntegerType)']
```

alias - rename a column

```
useful_perc_data = yelp_df.select(
    "id",
    (yelp_df.useful/28*100).cast("int").alias("useful_perc")
)
```

```
useful_perc_data.columns
```

```
Out[]: [u'id', u'useful_perc']
```

alias - rename a column

```
useful_perc_data = yelp_df.select(
    "id",
    (yelp_df.useful/28*100).cast("int").alias("useful_perc")
)
```

```
useful_perc_data.columns
```

```
Out[]: [u'id', u'useful_perc']
```

alias - rename also id

```
useful_perc_data = yelp_df.select(
    yelp_df["id"].alias("uid"),
    (yelp_df.useful/28*100).cast("int").alias("useful_perc")
)
```

```
useful_perc_data.columns
```

```
Out[]: [u'uid', u'useful_perc']
```

Ordering by column

Import functions for ascending/descending order:

```
from pyspark.sql.functions import asc, desc
```

order by usefulness

```
useful_perc_data = yelp_df.select(
    yelp_df["id"].alias("uid"),
    (yelp_df.useful/28*100).cast("int").alias("useful_perc")
).orderBy(desc("useful_perc"))
```

```
useful_perc_data.show(2)
uid                   useful_perc
RqwFPp_qPu-1h87pG... 100
YAXPKM-Hck6-mjF74... 82
```

Join inputs

id	useful_perc
9yKzy9PApe	17

id	review_count	state
9yKzy9PApe	6	"CA"

Join results

id	useful_perc	review_count
9yKzy9PApe	17	6

Join

```
useful_perc_data.join(
    yelp_df,
    yelp_df.id == useful_perc_data.uid,
    "inner"
)
```

Join - select

```
useful_perc_data.join(
    yelp_df,
    yelp_df.id == useful_perc_data.uid,
    "inner"
).select(useful_perc_data.uid, "useful_perc", "review_count")
```

Join - select - show

```
useful_perc_data.join(
    yelp_df,
    yelp_df.id == useful_perc_data.uid,
    "inner"
).select(useful_perc_data.uid, "useful_perc",
"review_count").show(5)
```

Output dataset

uid	useful_perc	review_count
WRBYytJAaJI1BTQG5...	71	362
GXj4PNAi095-q9ynP...	3	76
1sn0-eY_d1Dhr6Q2u...	0	9
MtFe-FuiOmo0vlo16...	0	7
EMYmuTlyeNBy5QB9P...	7	19

Cache in memory

```
useful_perc_data.join(
    yelp_df,
    yelp_df.id == useful_perc_data.uid,
    "inner"
).cache().select(useful_perc_data.uid, "useful_perc",
"review_count").show(5)
```

Run it again!

Analytics with Dataframes on HTTP server logs

Log analytics

Available in the Cloudera VM at:

`/usr/lib/hue/apps/search/examples/collections/solr_configs_log_analytics_demo/index_data.csv`

Log analytics

Check file contents on the terminal:

```
head
/usr/lib/hue/apps/search/examples/collections/solr_configs_log_analytics_demo/index_data.csv
```

Columns

code,protocol,request,app,user_agent_major,region_code,country_code,id,city,subapp,latitude,method,client_ip,user_agent_family,bytes,referer,country_name,extension,url,os_major,longitude,device_family,record,user_agent,time,os_family,country_code3

Start PySpark

Need to load spark-csv for CSV support:

```
PYSPARK_DRIVER_PYTHON=ipython pyspark --packages com.databricks:spark-csv_2.10:1.X.X
```

(Try to) read logs CSV

```
logs_df = sqlCtx.load(
source="com.databricks.spark.csv",
header = 'true',
inferSchema = 'true',
path =
'file:///usr/lib/hue/apps/search/examples/collections/solr_co
nfigs_log_analytics_demo/index_data.csv')

logs_df.count()
```

Parsing error

```
ERROR csv.CsvRelation$: Exception while parsing
line: ",Mozilla/4.0 (compatible; MSIE 7.0;
Windows NT 5.1; Trident/4.0; ....
```

Inspect the file with VIM

```
code,protocol,request,app,user_agent_major,region_code,country_code,id,city,subapp,latitude,method,client_ip,user_ag
record,user_agent,time,os_family,country_code3^M
200,HTTP/1.1,GET /metastore/table/default/sample_07 HTTP/1.1,metastore,,00,SG,8836e6ce-9a21-449f-a372-9e57641389b3,S
ore/table/default/sample_07,,103.85579999999999,Other,"demo.gethue.com:80 128.199.234.236 - - [04/May/2014:06:35:49
.0 (compatible; phpservermon/3.0.1; +http://www.phpservermonitor.org)""
",Mozilla/5.0 (compatible; phpservermon/3.0.1; +http://www.phpservermonitor.org),2014-05-04T06:35:49Z,Other,SGP^M
200,HTTP/1.1,GET /metastore/table/default/sample_07 HTTP/1.1,metastore,,00,SG,6ddf6e38-7b83-423c-8873-39842dca2dbb,S
ore/table/default/sample_07,,103.85579999999999,Other,"demo.gethue.com:80 128.199.234.236 - - [04/May/2014:06:35:50
.0 (compatible; phpservermon/3.0.1; +http://www.phpservermonitor.org)""
",Mozilla/5.0 (compatible; phpservermon/3.0.1; +http://www.phpservermonitor.org),2014-05-04T06:35:50Z,Other,SGP^M
200,HTTP/1.1,GET /search/?collection=10000001 HTTP/1.1,search,,00,SG,313bb28e-dd7c-4364-a11e-9ffb0db7b303,Singapore,
```

Access Hadoop configuration

Spark relies on Hadoop functionality for reading data.

```
sc._jsc.hadoopConfiguration()
```

Set input file delimiter

Spark relies on Hadoop functionality for reading data.

```
sc._jsc.hadoopConfiguration().set('textinputformat.record.delimiter', '\r\n')
```

Read logs CSV

```
logs_df = sqlCtx.load(
source="com.databricks.spark.csv",
header = 'true', inferSchema = 'true',
path =
'file:///usr/lib/hue/apps/search/examples/collections/solr_configs_log_analytics_demo/index_data.csv')

logs_df.count()
Out[]: 9410L
```

Display of logs DataFrame

```
code protocol request                    app       user_agent_major region_code country_code id
e extension url                        os_major longitude          device_family record               user_agent
200  HTTP/1.1 GET /metastore/ta... metastore null              00           SG           8836e6ce-9a21-449...
            /metastore/table/... null     103.85579999999999 Other          demo.gethue.com:8... Mozilla/5.0
200  HTTP/1.1 GET /metastore/ta... metastore null              00           SG           6ddf6e38-7b83-423...
            /metastore/table/... null     103.85579999999999 Other          demo.gethue.com:8... Mozilla/5.0
200  HTTP/1.1 GET /search/?coll... search    null              00           SG           313bb28e-dd7c-436...
            /search/?collecti... null     103.85579999999999 Other          demo.gethue.com:8... Mozilla/5.0
200  HTTP/1.1 GET /search/?coll... search    null              00           SG           ecb47c61-a9e4-4b5...
            /search/?collecti... null     103.85579999999999 Other          demo.gethue.com:8... Mozilla/5.0
200  HTTP/1.1 HEAD / HTTP/1.1               null              00           SG           affdb6b9-3657-4d1...
            /                    null     103.85579999999999 Other          demo.gethue.com:8... Mozilla/5.0
```

```
root
 |-- code: integer (nullable = true)
 |-- protocol: string (nullable = true)
 |-- request: string (nullable = true)
 |-- app: string (nullable = true)
 |-- user_agent_major: integer (nullable = true)
 |-- region_code: string (nullable = true)
 |-- country_code: string (nullable = true)
 |-- id: string (nullable = true)
 |-- city: string (nullable = true)
 |-- subapp: string (nullable = true)
 |-- latitude: double (nullable = true)
 |-- method: string (nullable = true)
 |-- client_ip: string (nullable = true)
 |-- user_agent_family: string (nullable = true)
 |-- bytes: integer (nullable = true)
 |-- referer: string (nullable = true)
 |-- country_name: string (nullable = true)
 |-- extension: string (nullable = true)
```

Count by HTTP code

Count the log events by HTTP code (i.e. how many 200 OK, 404 Not found...)

```
logs_df.groupBy("code").count().show()
```

code	count
500	2
301	71
302	1943
502	6
304	117
400	1
200	7235
401	10
404	11

```
from pyspark.sql.functions import asc, desc
logs_df.groupBy("code").count().orderBy(desc("count")).show()

code count
200  7235
302  1943
304  117
301  71
408  14
404  11
```

Compute average

```
logs_df.groupBy("code").avg("bytes").show()

code AVG(bytes#47)
500  4684.5
301  424.61971830985914
302  415.6510550694802
502  581.0
304  185.26495726495727
400  0.0
```

Mean, Min, Max by code

Compute in a single operation Mean, Min and Max by HTTP code

```
import pyspark.sql.functions as F

logs_df.groupBy("code").agg(
                    logs_df.code,
                    F.avg(logs_df.bytes),
                    F.min(logs_df.bytes),
                    F.max(logs_df.bytes)
).show()
```

Mean, Min, Max by code

code	AVG(bytes#47)	MIN(bytes#47)	MAX(bytes#47)
500	4684.5	422	8947
301	424.61971830985914	331	499
302	415.6510550694802	304	1034
502	581.0	581	581
304	185.26495726495727	157	204
400	0.0	0	0
200	41750.03759502419	0	9045352
401	12472.8	8318	28895
404	17872.454545454544	7197	23822
408	440.57142857142856	0	514

Completed DataFrames

- Completed analytics with DataFrames
- Next we'll focus on interoperability with SQL query language and Hive

Spark SQL

DataFrames and Databases

DataFrame & Database table: conceptually equivalent.

Spark makes them interoperable

DataFrames & SQL

- Query existing Spark DataFrames (however created) with SQL
  - Same functionality, different interface
  - Legacy SQL

Read data and persistency

- Load data from Hive / other databases
- Save tables to Hive

Explore the Yelp dataset

```
yelp_df = sqlCtx.load(
source='com.databricks.spark.csv',
header = 'true',
inferSchema = 'true',
path =
'file:///usr/lib/hue/apps/search/examples/collections/solr_configs_yelp_demo/index_data.csv')
```

Register as a SQL table

- DataFrame already has Schema
- Create a temporary table with:

```
yelp_df.registerTempTable("yelp")
```

Run SQL statements

```
filtered_yelp = sqlCtx.sql("SELECT * FROM yelp WHERE useful >= 1")
filtered_yelp

Out[]: DataFrame[business_id: string, cool: int, date: string, funny: int, id: string, stars: int, text: string, type: string, useful: int, user_id: string, name: string, full_address: string, latitude: double, longitude: double, neighborhoods: string, open: string, review_count: int, state: string]
```

Filtering

```
filtered_yelp.count()
Out[]: 601L
```

```
yelp_df.filter(yelp_df.useful >= 1).count()
Out[]: 601L
```

aggregation

```
sqlCtx.sql("SELECT MAX(useful) AS max_useful FROM yelp").collect()
Out[]: [Row(max_useful)=28)]


yelp_df.agg({"useful":"max"}).collect()
Out[]: [Row(MAX(useful#267)=28)]
```

Join - select - show

```
useful_perc_data.join(
    yelp_df,
    yelp_df.id == useful_perc_data.uid,
    "inner"
).select(useful_perc_data.uid, "useful_perc", "review_count")
```

Register as SQL table

```
useful_perc_data.registerTempTable("useful_perc_data")
```

join

```
sqlCtx.sql(
"""SELECT useful_perc_data.uid, useful_perc,
review_count
FROM useful_perc_data
INNER JOIN yelp
ON useful_perc_data.uid=yelp.id"""
)
```

Performance

- Either DataFrame calls or SQL
- Same under-the-hood optimizer (Catalyst)
- Creates DAG
- Parallel execution
- Creates bytecode

Spark and Hive

Spark and Hive

- copied hive-site.xml to Spark conf/
- Spark read / write to Hive

Hive table to DataFrame

- sqlCtx.sql has access to Hive tables
- Load data uploaded during the Hive class
- Result is a DataFrame

```
customers_df = sqlCtx.sql("SELECT * FROM customers")
customers_df.show()
```

Printout of customers_df

customer_id	customer_fname	customer_lname	customer_email	customer_password	customer_street	customer_city	customer_state	customer_zipcode
1	Richard	Hernandez	XXXXXXXXX	XXXXXXXXX	6303 Heather Plaza	Brownsville	TX	78521
2	Mary	Barrett	XXXXXXXXX	XXXXXXXXX	9526 Noble Embers...	Littleton	CO	80126
3	Ann	Smith	XXXXXXXXX	XXXXXXXXX	3422 Blue Pioneer...	Caguas	PR	00725
4	Mary	Jones	XXXXXXXXX	XXXXXXXXX	8324 Little Common	San Marcos	CA	92069
5	Robert	Hudson	XXXXXXXXX	XXXXXXXXX	10 Crystal River ...	Caguas	PR	00725
6	Mary	Smith	XXXXXXXXX	XXXXXXXXX	3151 Sleepy Quail...	Passaic	NJ	07055
7	Melissa	Wilcox	XXXXXXXXX	XXXXXXXXX	9453 High Concession	Caguas	PR	00725
8	Megan	Smith	XXXXXXXXX	XXXXXXXXX	3047 Foggy Forest...	Lawrence	MA	01841
9	Mary	Perez	XXXXXXXXX	XXXXXXXXX	3616 Quaking Street	Caguas	PR	00725
10	Melissa	Smith	XXXXXXXXX	XXXXXXXXX	8598 Harvest Beac...	Stafford	VA	22554
11	Mary	Huffman	XXXXXXXXX	XXXXXXXXX	3169 Stony Woods	Caguas	PR	00725
12	Christopher	Smith	XXXXXXXXX	XXXXXXXXX	5594 Jagged Ember...	San Antonio	TX	78227
13	Mary	Baldwin	XXXXXXXXX	XXXXXXXXX	7922 Iron Oak Gar...	Caguas	PR	00725
14	Katherine	Smith	XXXXXXXXX	XXXXXXXXX	5666 Hazy Pony Sq...	Pico Rivera	CA	90660
15	Jane	Luna	XXXXXXXXX	XXXXXXXXX	673 Burning Glen	Fontana	CA	92336
16	Tiffany	Smith	XXXXXXXXX	XXXXXXXXX	6651 Iron Port	Caguas	PR	00725
17	Mary	Robinson	XXXXXXXXX	XXXXXXXXX	1325 Noble Pike	Taylor	MI	48180
18	Robert	Smith	XXXXXXXXX	XXXXXXXXX	2734 Hazy Butterf...	Martinez	CA	94553
19	Stephanie	Mitchell	XXXXXXXXX	XXXXXXXXX	3543 Red Treasure...	Caguas	PR	00725
20	Mary	Ellis	XXXXXXXXX	XXXXXXXXX	4703 Old Route ...	West New York	NJ	07093

```
customers_df.printSchema()
root
 |-- customer_id: integer (nullable = true)
 |-- customer_fname: string (nullable = true)
 |-- customer_lname: string (nullable = true)
 |-- customer_email: string (nullable = true)
 |-- customer_password: string (nullable = true)
 |-- customer_street: string (nullable = true)
 |-- customer_city: string (nullable = true)
 |-- customer_state: string (nullable = true)
 |-- customer_zipcode: string (nullable = true)
```

Run unmodified SQL queries

```
sqlCtx.sql("""select c.category_name,
count(order_item_quantity) as count from order_items oi
inner join products p on oi.order_item_product_id =
p.product_id inner join categories c on c.category_id =
p.product_category_id group by c.category_name
order by count desc
limit 10"""
).show()
```

Most popular categories

category_name	count
Cleats	24551
Men's Footwear	22246
Women's Apparel	21035
Indoor/Outdoor Games	19298
Fishing	17325
Water Sports	15540
Camping & Hiking	13729
Cardio Equipment	12487
Shop By Sport	10984
Electronics	3156

Run unmodified SQL queries

```
sqlCtx.sql("""select p.product_id, p.product_name, r.revenue
from products p inner join
(select oi.order_item_product_id, sum(cast(oi.order_item_subtotal as float)) as
revenue from order_items oi inner join orders o on oi.order_item_order_id =
o.order_id
where o.order_status <> 'CANCELED'
and o.order_status <> 'SUSPECTED_FRAUD'
group by order_item_product_id) r
on p.product_id = r.order_item_product_id
order by r.revenue desc limit 10"""
).show()
```

Top 10 products by revenue

product_id	product_name	revenue
1004	Field & Stream Sp...	6637668.282318115
365	Perfect Fitness P...	4233794.3682899475
957	Diamondback Women...	3946837.004547119
191	Nike Men's Free 5...	3507549.2067337036
502	Nike Men's Dri-FI...	3011600.0
1073	Pelican Sunstream...	2967851.6815185547
1014	O'Brien Men's Neo...	2765543.314743042
403	Nike Men's CJ Eli...	2763977.4868011475
627	Under Armour Girl...	1214896.220287323
565	adidas Youth Germ...	63490.0

Save DataFrames to Hive

`registerTempTable` only gives temporary SQL-like access to DataFrames

Store permanently to Hive with:

```
yelp_df.saveAsTable("yelp_reviews")
```

Check persistency

- Restart PySpark
- Run: `sqlCtx.sql("SELECT * FROM yelp").show()`
- Fails with "Table not found"

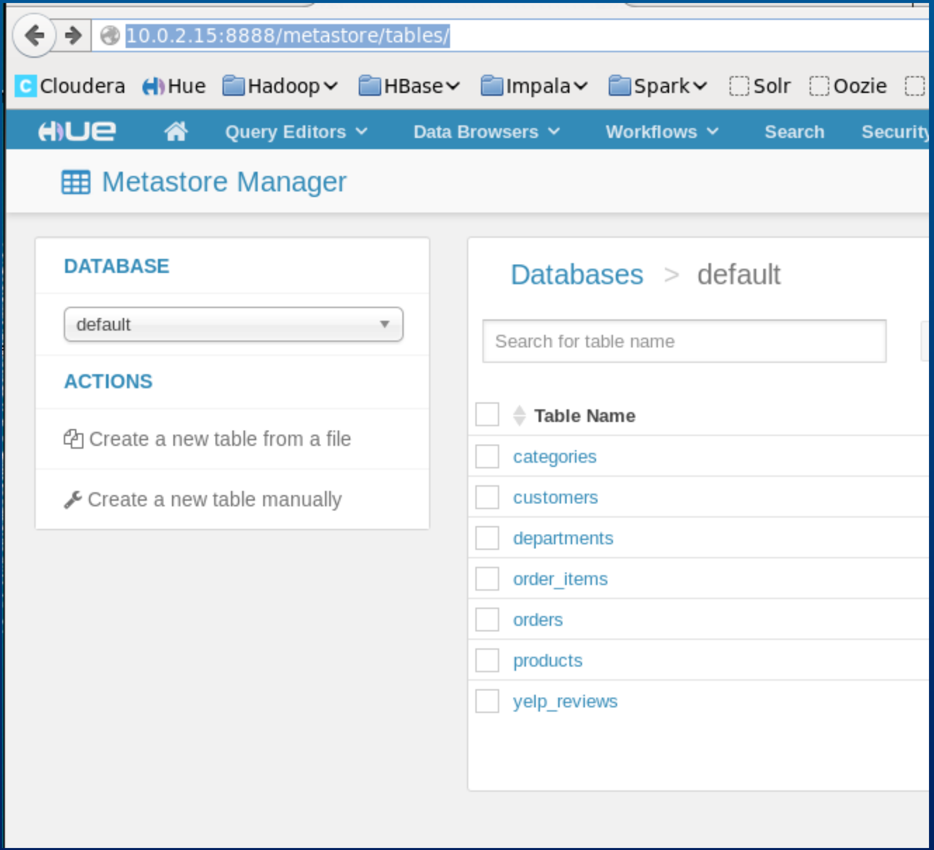
Check persistency

- Restart PySpark
- Run: `sqlCtx.sql("SELECT * FROM yelp_reviews").show()`
- Restores from Hive

Loaded Yelp DataFrame

```
business_id          cool date       funny id                   stars text                type     useful user_id              name                 full_address         latitude      longitude      neighbor
hoods open  review_count state
9yKzy9PApeiPPOUJE... 2    2011-01-26 0     fWKvX83p0-ka4JS3d... 4     My wife took me h... business 5      rLtl8ZkDX5vH5nAx9... Morning Glory Cafe   6106 S 32nd St Ph... 33.3907928467 -112.012504578 []
        True  116         AZ
ZRJwVLyzEJq1VAihD... 0    2011-07-27 0     IjZ33sJrzXqU-0X6U... 4     I have no idea wh... business 0      0a2KyEL0d3Yb1V6ai... Spinato's Pizzeria   4848 E Chandler B... 33.305606842  -111.978759766 []
        True  102         AZ
6oRAC4uyJCsJl1X0W... 0    2012-06-14 0     IESLBzqUCLdSzSqm0... 4     love the gyro pla... business 1      0hT2KtfLiobPvh6cD... Haji-Baba            1513 E  Apache Bl... 33.4143447876 -111.913032532 []
        True  265         AZ
_1QQZuf4zZOyFCvXc... 1    2010-05-27 0     G-WvGaISbqqaMHlNn... 4     Rosie, Dakota, an... business 2      uZetl9T0NcROGOyFf... Chaparral Dog Park   5401 N Hayden Rd ... 33.5229454041 -111.90788269  []
        True  88          AZ
6ozycU1RpktNG2-1B... 0    2012-01-05 0     1uJFq2r5QfJG_6ExM... 4     General Manager S... business 0      vYmM4KTsC8ZfQBg-j... Discount Tire        1357 S Power Road... 33.3910255432 -111.68447876  []
        True  5           AZ
-yxfBYGB6SEqszmxJ... 4    2007-12-13 1     m2CKSsepBCoRYWxiR... 3     Quiessence is, si... business 3      sqYN3lNgvPbPCTRsM... Quiessence Restau... 6106 S 32nd St Ph... 33.3907928467 -112.012504578 []
        True  109         AZ
zp713qNhx8d9KCJJn... 7    2010-02-12 4     riFQ3vxNpP4rWLk_C... 4     Drop what you're ... business 7      wFweIWhv2fREZV_dY... La Condesa Gourme... 1919 N 16th St Ph... 33.4691314697 -112.04750824  []
        True  307         AZ
hW0Ne_HTHEAgGF1rA... 0    2012-07-12 0     JL7GXJ9u4YMx7Rzs0... 3     Luckily, I didn't... business 1      1ieuYcKS7zeAv_U15... Phoenix Sky Harbo... 3400 E Sky Harbor... 33.4347496033 -112.006439209 []
        True  862         AZ
wNUea3IXZWD63bbOQ... 0    2012-08-17 0     XtnfnYmnJYi71yIuG... 3     Definitely come f... business 0      Vh_DlizgGhSqQh4qf... Stingray Sushi       2574 E Camelback ... 33.5096054077 -112.025741577 []
        True  163         AZ
nMHhuYan8e3cONo3P... 0    2010-08-11 0     jJAIXA46pU1swYyRC... 4     Nobuo shows his u... business 1      sUNkXg8-KFtCMQDV6... Nobuo At Teeter H... 622 E Adams St Ph... 33.4495391846 -112.065666199 []
        True  189         AZ
AsSCv0q_BWqIe3mX2... 1    2010-06-16 1     E11jzpKz9Kw5K7fuA... 4     The oldish man wh... business 3      -OMlS6yWkYjVldNhC... Cookiez On Mill      514 S Mill Ave St... 33.4248809814 -111.940200806 []
        False 74          AZ
e9nN4XxjdHj4qtKCO... 1    2011-10-21 0     3rPt0LxF7rgmEUrzn... 4     Wonderful Vietnam... business 1      C1rHp3dmepNea7Xio... Lee's Sandwiches     1901 W Warner Rd ... 33.3347129822 -111.874786377 []
        True  192         AZ
h53YuCiIDfEFSJCQp... 1    2010-01-11 0     cGnKNX3I9rthE0-TH... 4     They have a limit... business 2      UPtysDF6cUDUxq2KY... Jason's Deli         1065 E Baseline R... 33.3796195984 -111.809425354 []
        True  36          AZ
WGNIYMeXPyoWav1AP... 1    2011-12-23 0     FvEEw1_OsrYdvwLV5... 4     Good tattoo shop.... business 2      Xm8HXE1JHqscXe5BK... The Lady Luck Tat... 961 E Guadalupe R... 33.3637619019 -111.9272995   []
        True  25          AZ
yc5AH9H71xJidA_J2... 1    2010-05-20 0     pfUwBKYYmUXeiwrhD... 4     I'm 2 weeks new t... business 1      JOG-4G4e8ae3lx_sz... Rosie McCaffrey's    906 E Camelback R... 33.5095176697 -112.061569214 []
        True  151         AZ
Vb9FPCEL6Ly24PNxL... 0    2011-03-20 0     HvqmdqWcerVWO3Gs6... 4     Was it worth the ... business 2      ylWOj2y7TV2e3yYeW... Z Pizza              13637 N Tatum Blv... 33.6101531982 -111.976852417 []
        False 28          AZ
supigcPNO9IKo6ola... 3    2008-10-12 2     HXP_0Ul-FCmA4f-k9... 3     We went here on a... business 4      SBbftLzfYYKItOMFw... 1130 The Restaurant  455 N 3rd St Ste ... 33.452796936  -112.069320679 []
        True  86          AZ
O510Re68mOy9dU490... 0    2010-05-03 0     j4SIzrIy0WrmW4yr4... 3     okay this is the ... business 0      u1KWcbPMvXFEEYkZZ... Oakville Grocery     15015 N Scottsdal... 33.6246795654 -111.924377441 []
        False 39          AZ
b5cEoKR8iQliq-yT2... 5    2009-03-06 4     v0cTd3PNpYCkTyGKS... 4     I met a friend fo... business 6      UsULgP4bKA8RMzs8d... Carlsbad Tavern      3313 N Hayden Rd ... 33.4869194031 -111.908737183 []
        True  262         AZ
4JzzbSbK9wml0BJZW... 1    2011-11-17 1     a0lCu-j2Sk_kHQsZi... 2     They've gotten be... business 1      nDBly08j5URmrHQ2J... Frontier Airlines    Phoenix Sky Harbo... 33.4396476746 -112.026153564 []
        True  13          AZ
```

quickstart.cloudera:8888/metastore/tables



Conclusion

- Analytics with DataFrames, filtering, aggregation, joins, grouping
- How to add new packages to Spark and modify Hadoop configuration
- Operate on DataFrames with SQL
- Persist DataFrames as Hive tables