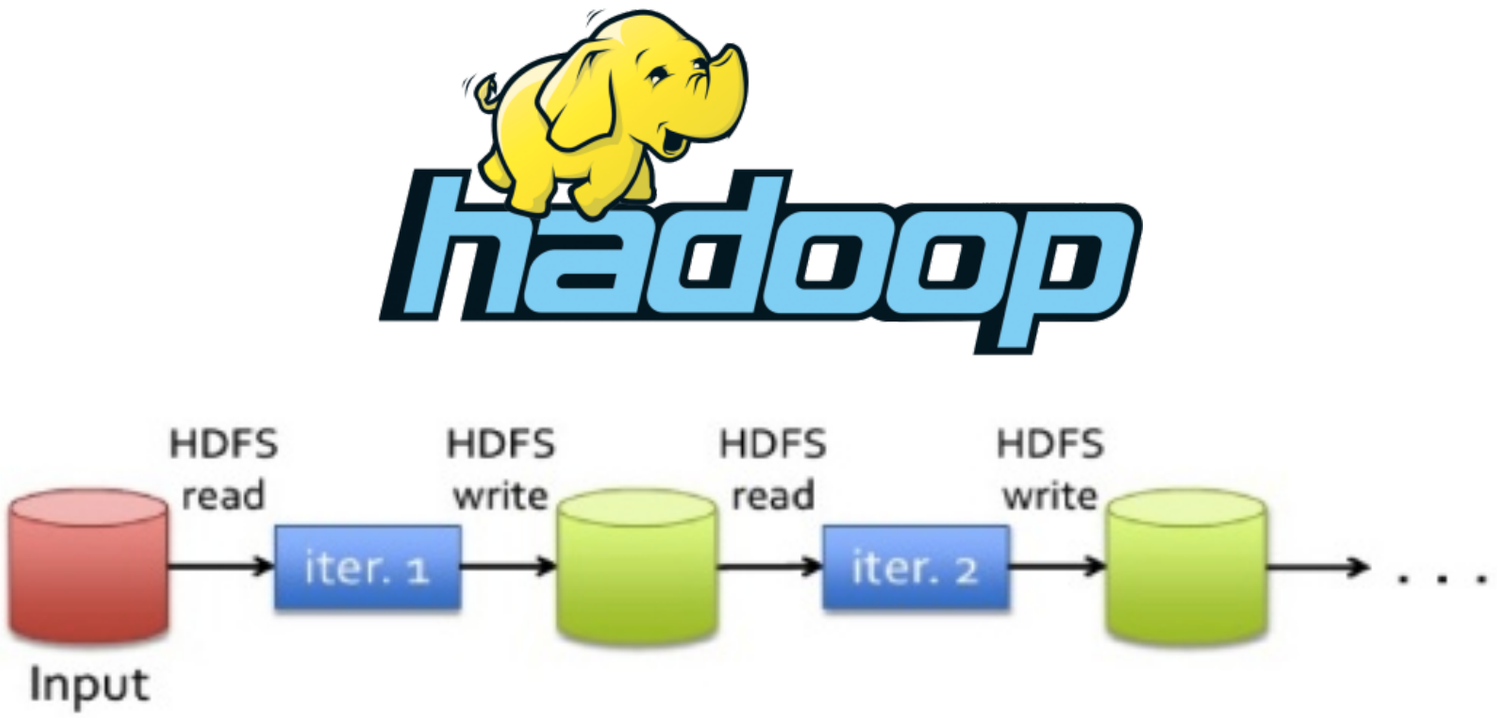
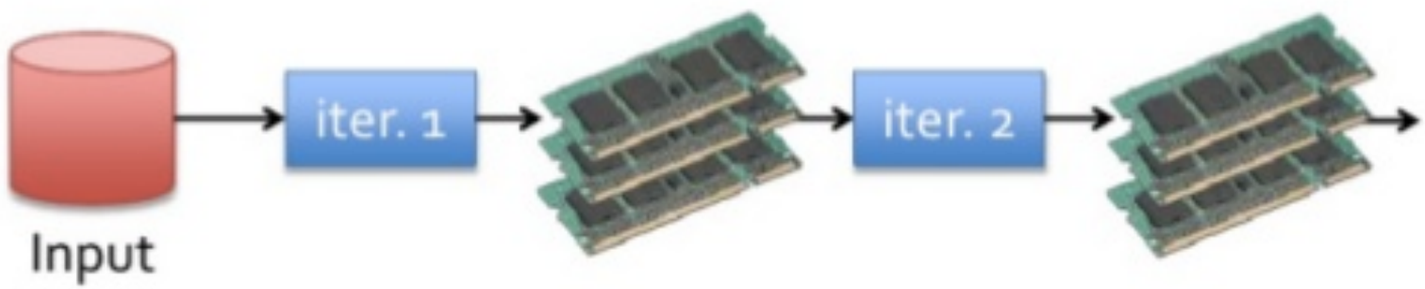
- Resilient Distributed Datasets (RDD)
- Transformations and Actions
- Building the Directed Acyclic Graph (DAG)
- Job, Stages, and Tasks
- Ways to Run Spark
- Spark DataFrames and SparkSQL
- Machine Learning on Spark
- Spark Tuning





THOMSON REUTERS

Resilient Distributed Dataset (RDD)

Dataset

```
1420589581, CEC023AA, 40712
1436660661, 175AB332, 60813
1469828518, 407BA6EF, 70179
…
…
…
```

5GB

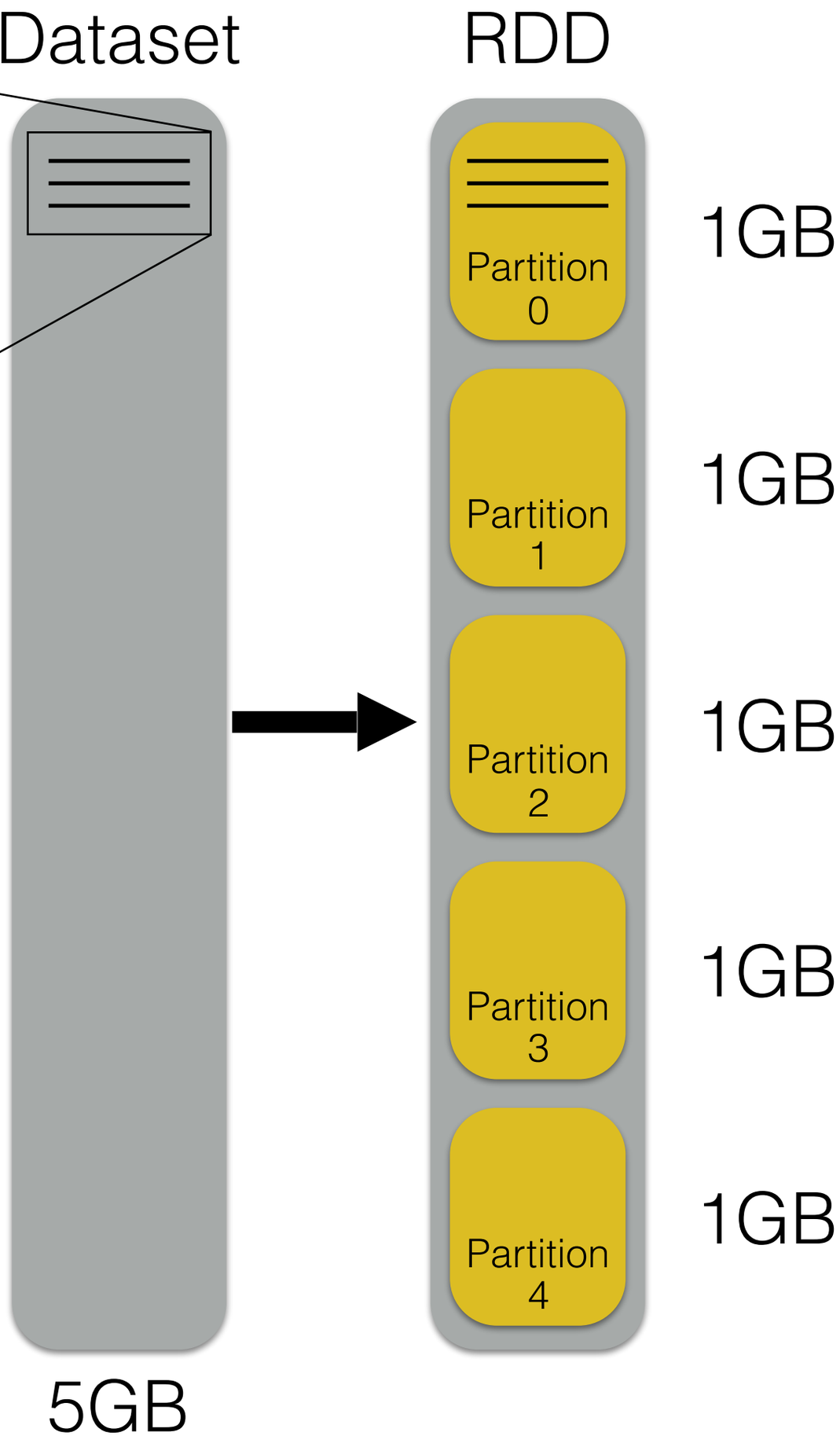
Dataset

```
1420589581, CEC023AA, 40712
1436660661, 175AB332, 60813
1469828518, 407BA6EF, 70179
…
…
…
```

5GB

```
1420589581, CEC023AA, 40712
1436660661, 175AB332, 60813
1469828518, 407BA6EF, 70179
…
…
…
```

Resilient Distributed Dataset



RDD

1GB

1GB

1GB

1GB

1GB

0

1

2

3

4

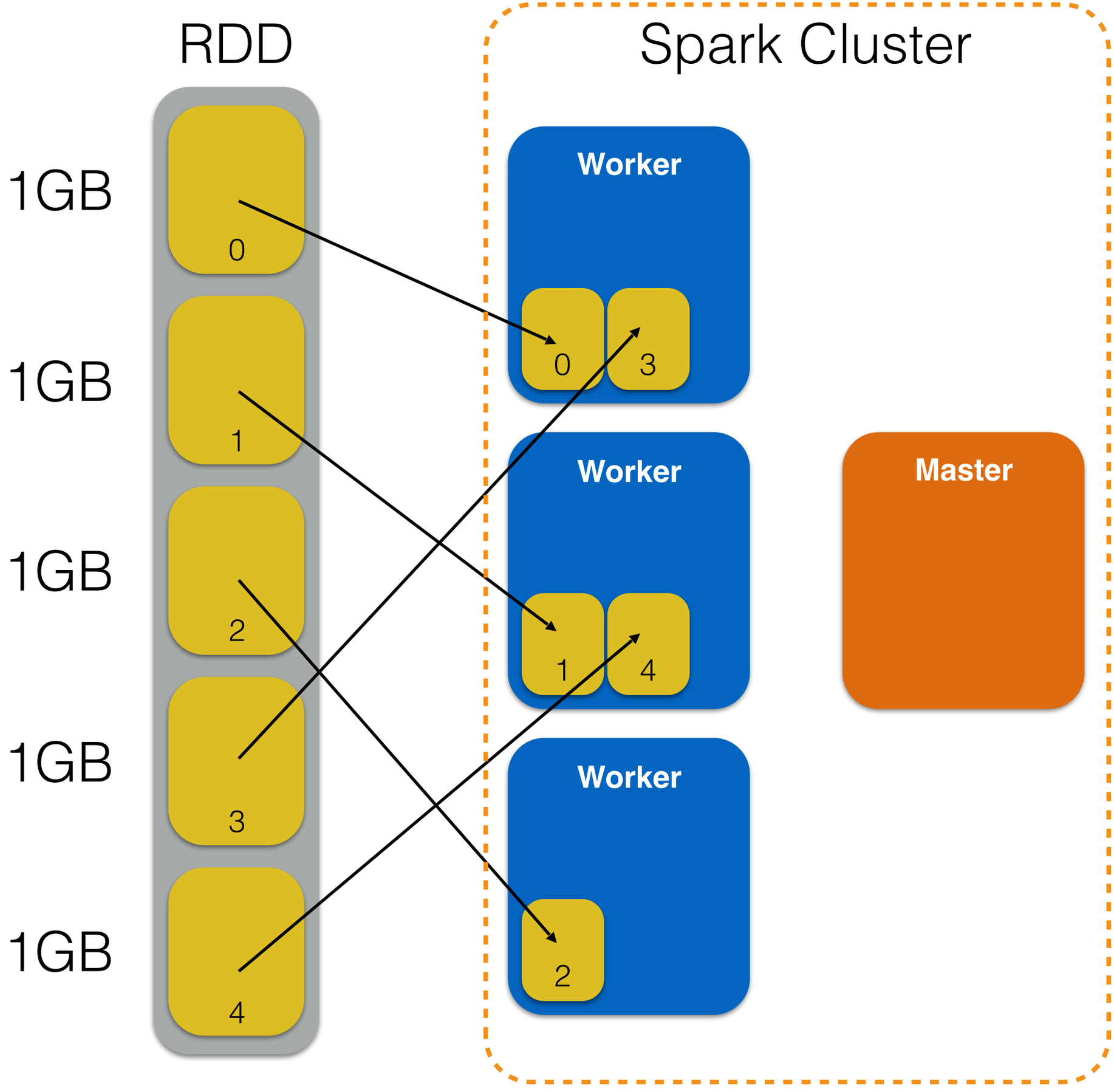
Spark Cluster

Worker

Worker

Worker

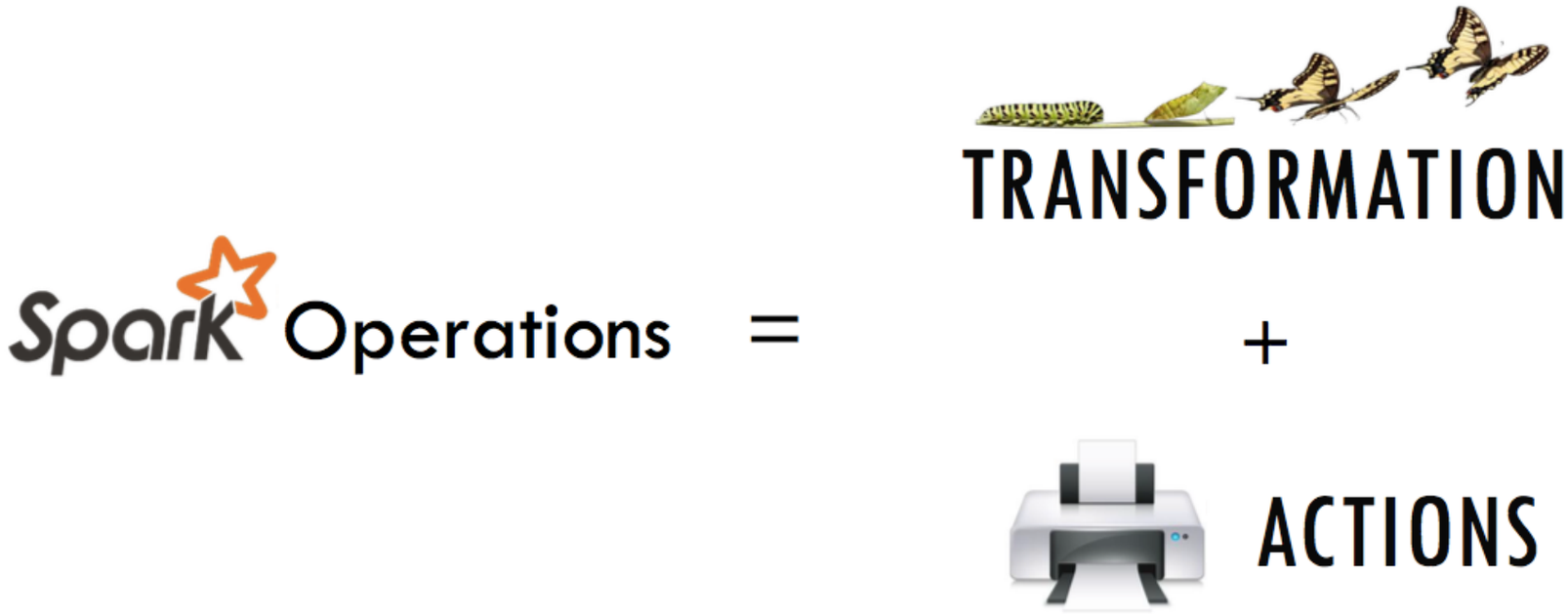
Master


RDD
Spark Cluster
1GB
1GB
1GB
1GB
1GB
0
1
2
3
4
Worker
0
3
Worker
1
4
Worker
2
Master

Transformations and Actions

Transformations and Actions

- Transformations are performed on RDDs
- Transformations are used to build a Directed Acyclic Graph (DAG)
- Actions trigger data to flow through the DAG



http://training.databricks.com/visualapi.pdf

Transformations and Actions

.map
.filter
.groupByKey
.reduceByKey

.sortByKey
.flatMap
.repartition

.collect
.take
.first

.count
.takeSample
.reduce

http://training.databricks.com/visualapi.pdf

Transformations and Actions

.map
.filter
.groupByKey
.reduceByKey

.sortByKey
.flatMap
.repartition

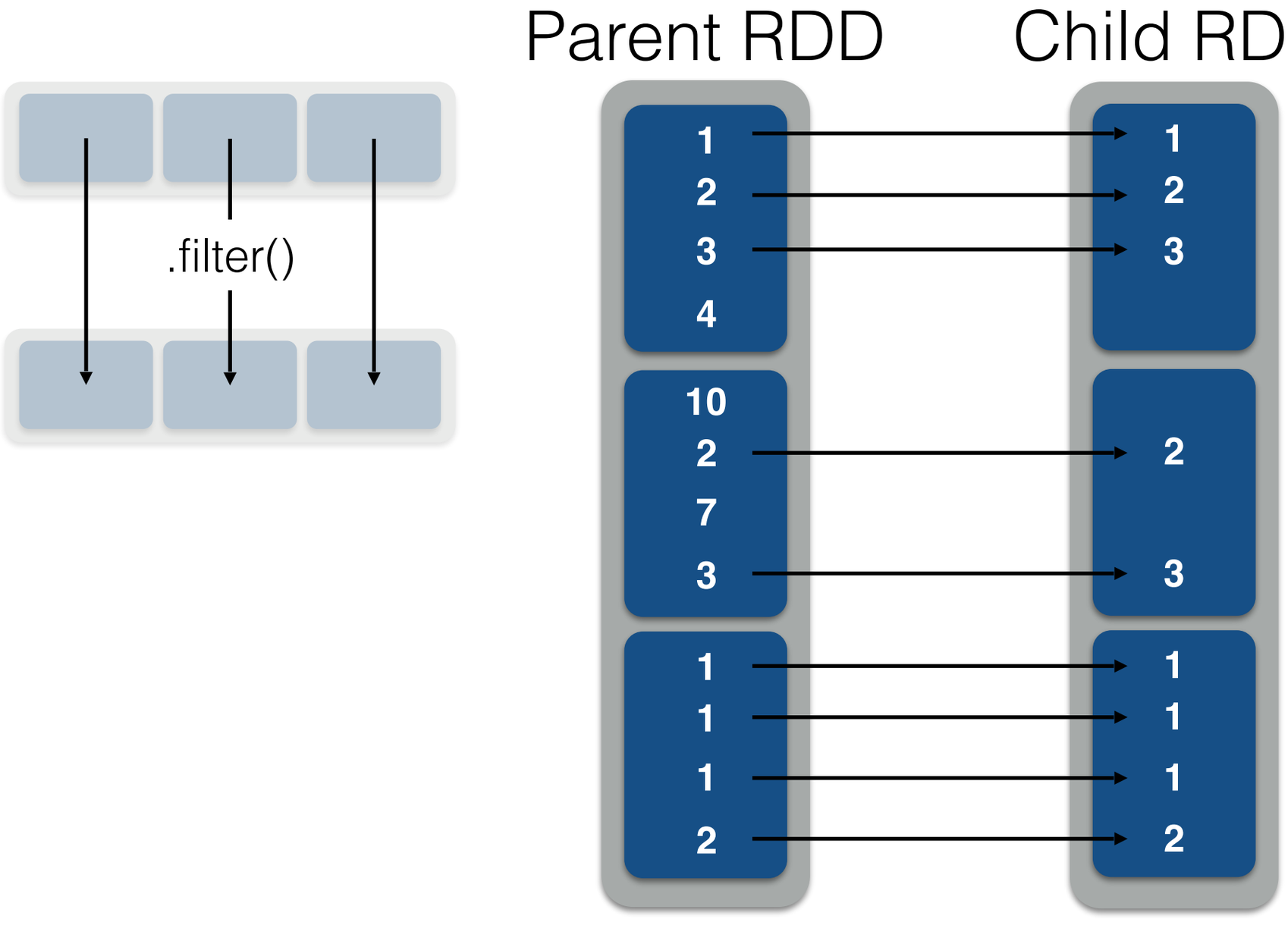
.collect
.take
.first

.count
.takeSample
.reduce

http://training.databricks.com/visualapi.pdf

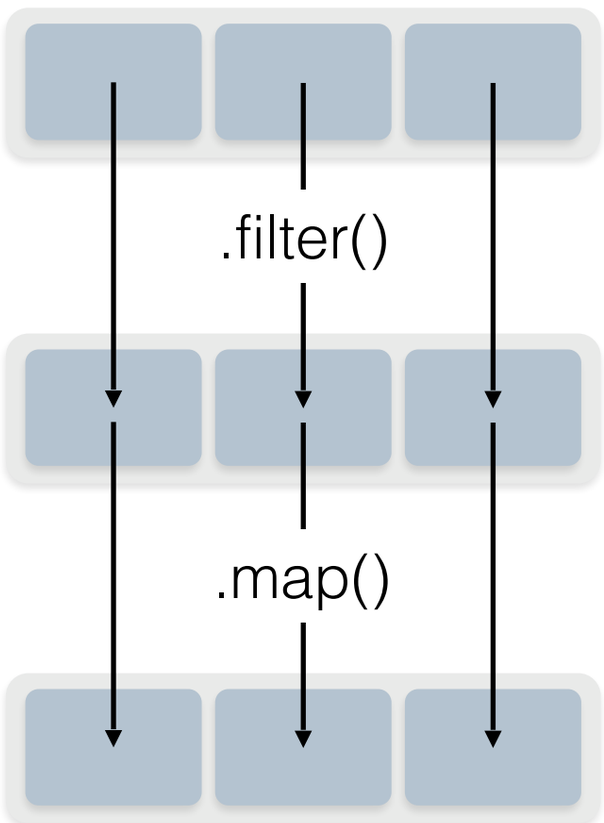
Basic Transformation Examples

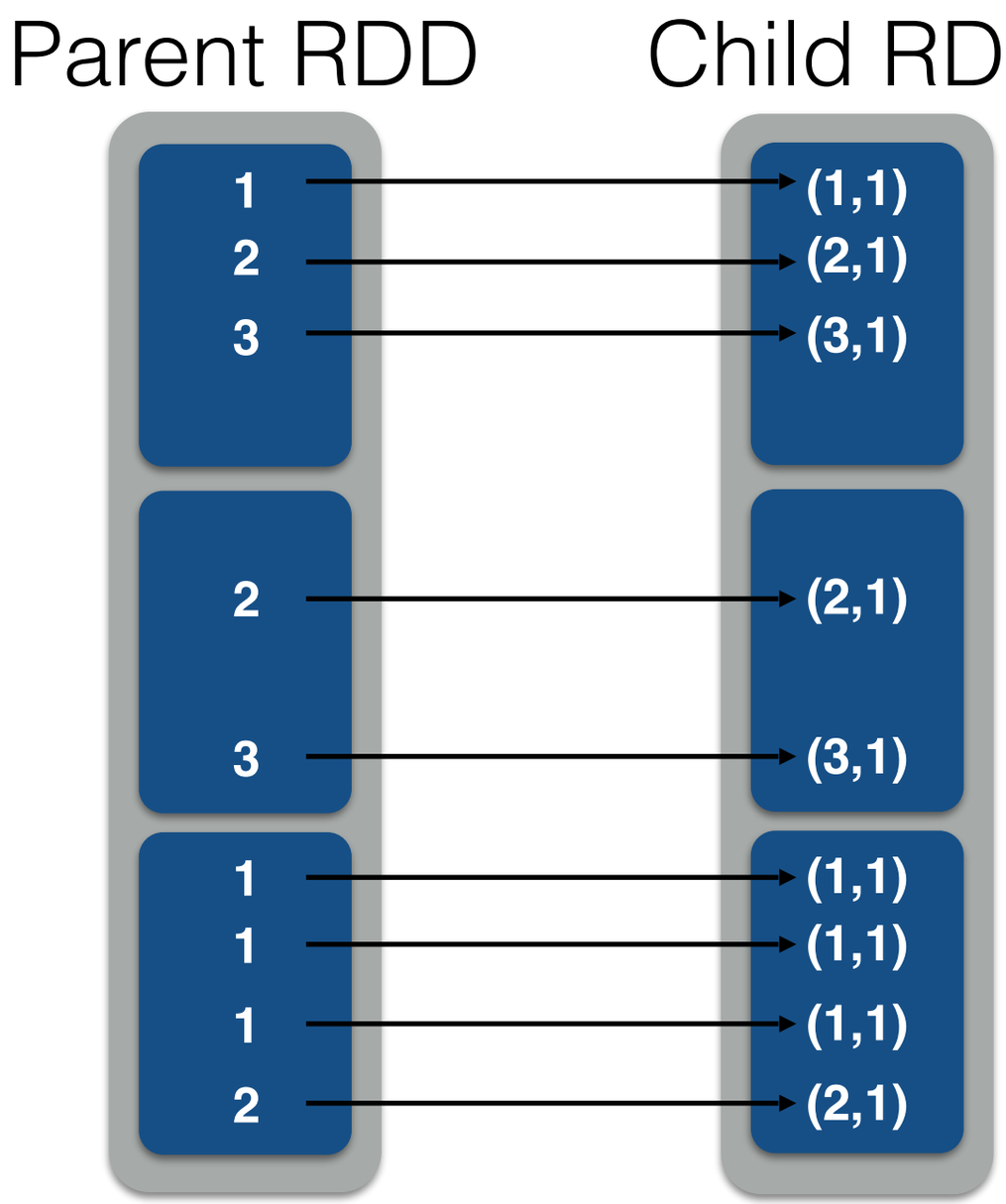
.filter(lambda r: r < 4)



Basic Transformation Examples

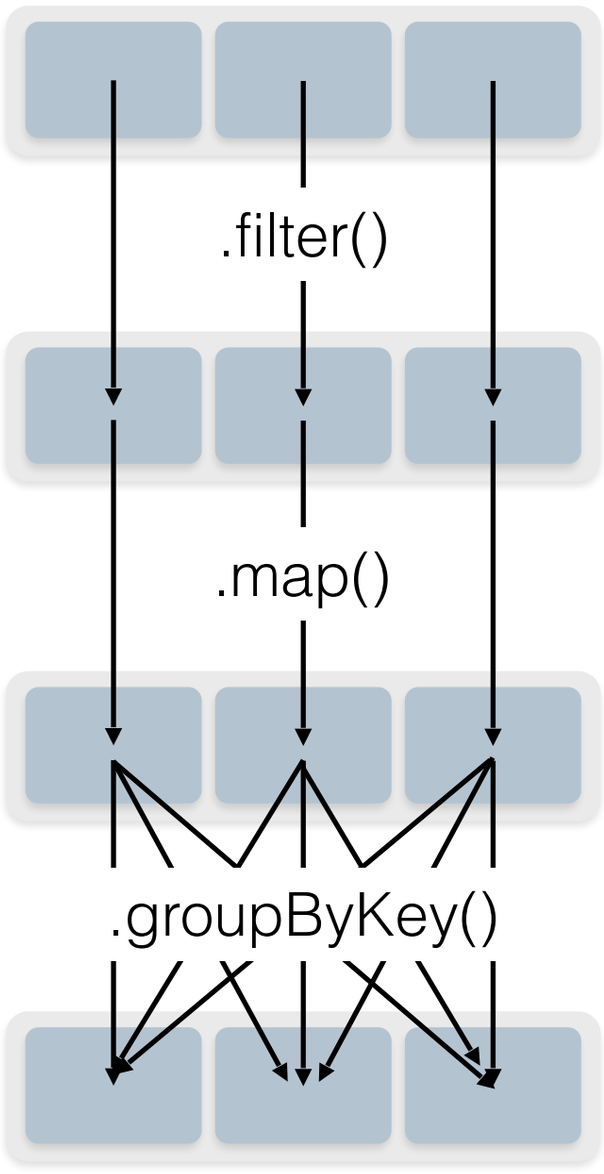
.map(lambda r: (r, 1))

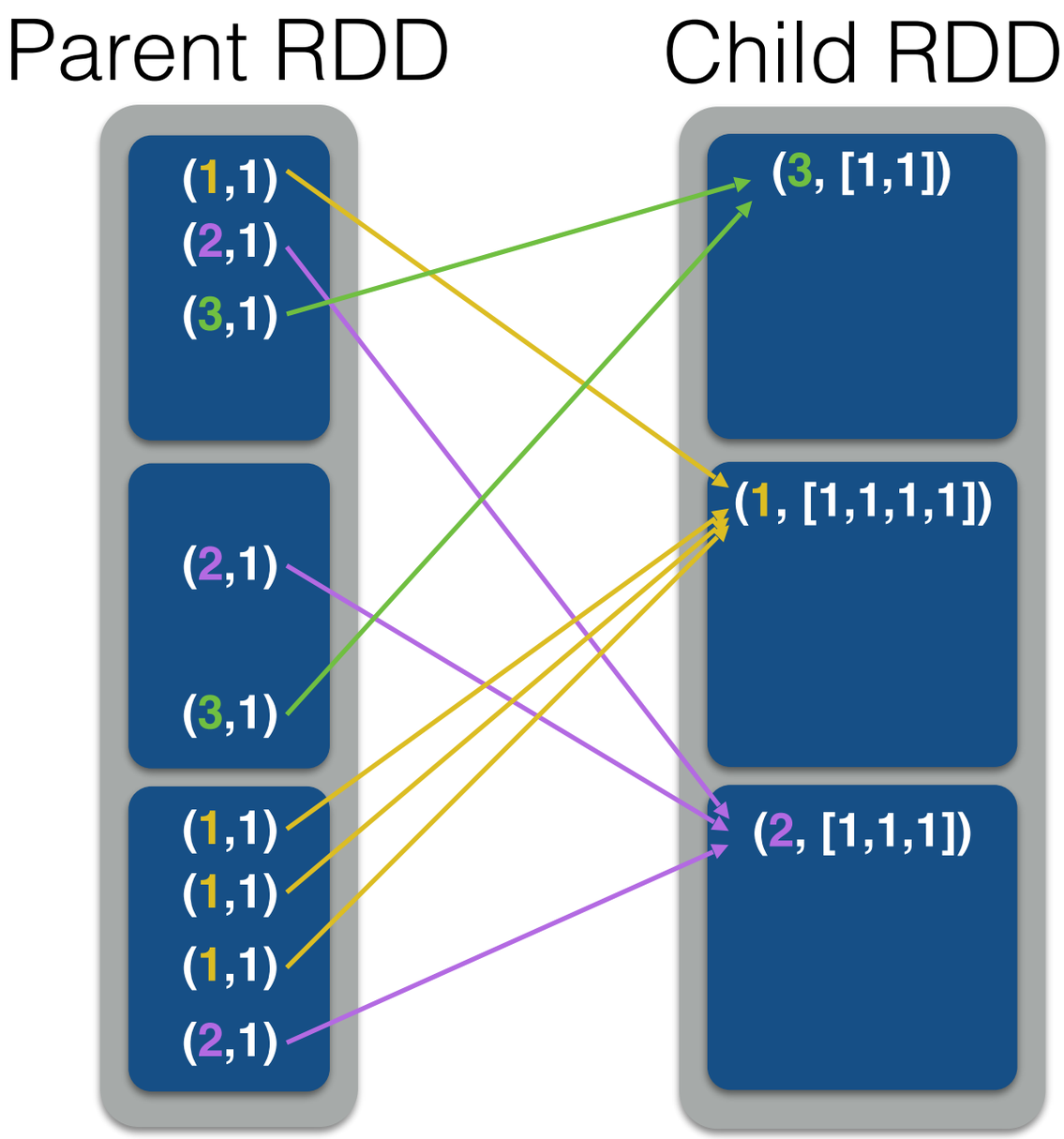




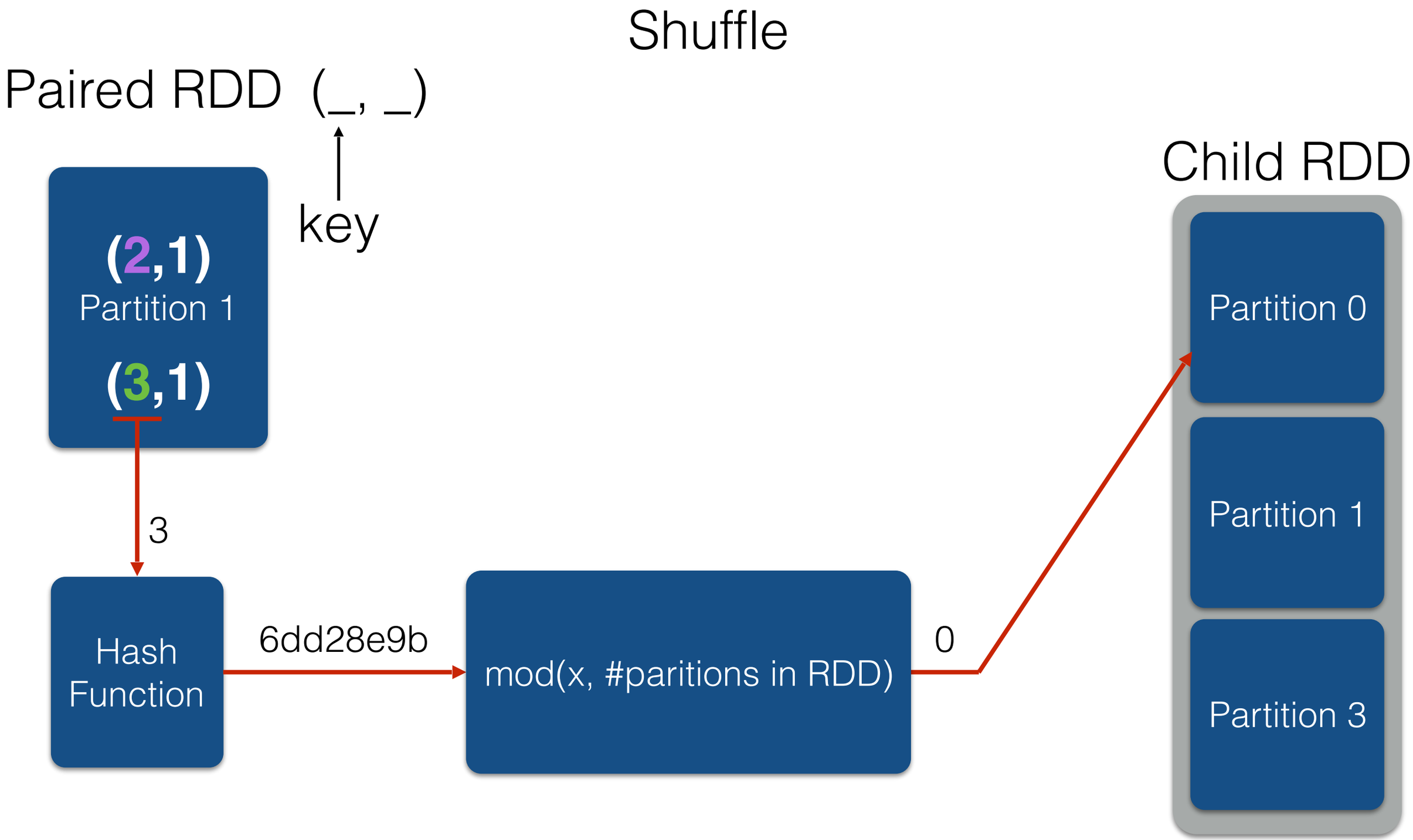
Basic Transformation Examples

.groupByKey()





Basic Transformation Examples

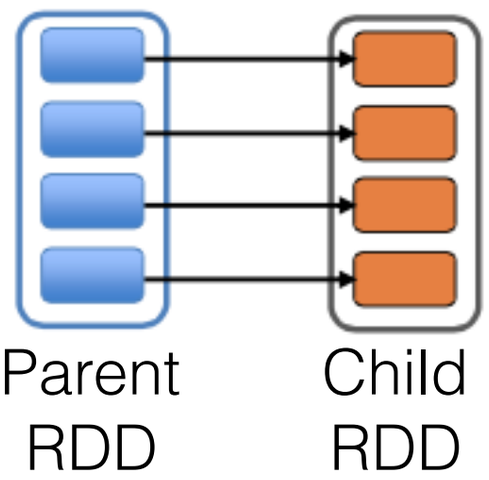


Narrow vs Wide Transformations

vs

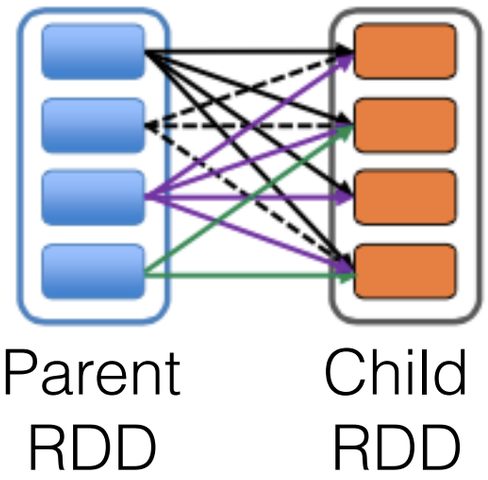
narrow

each partition of the parent RDD is used by at most one partition of the child RDD



wide

multiple child RDD partitions may depend on a single parent RDD partition

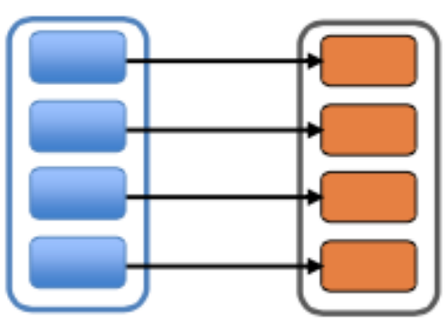


http://training.databricks.com/visualapi.pdf

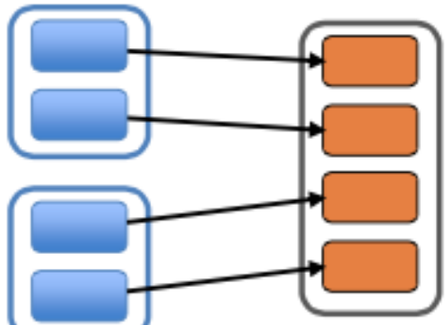
Narrow vs Wide Transformations

narrow

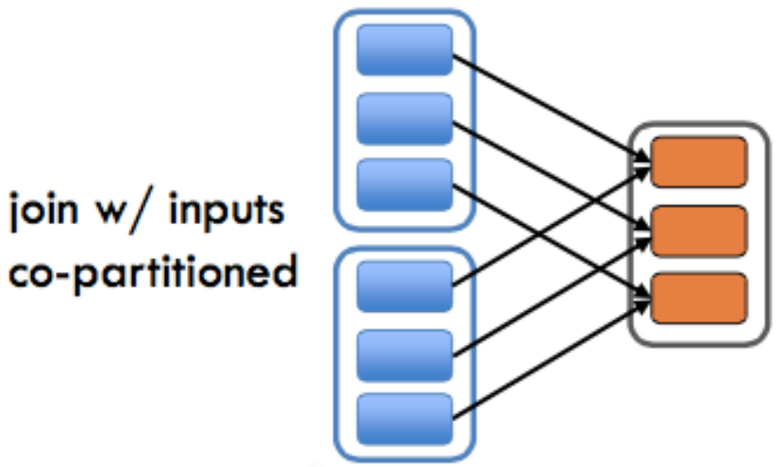
each partition of the parent RDD is used by at most one partition of the child RDD



map, filter

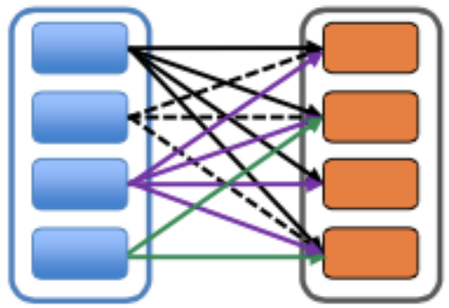


union

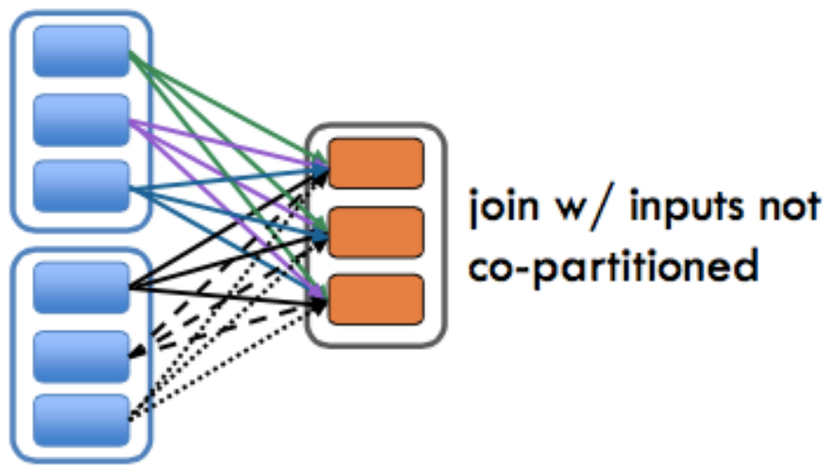


wide

multiple child RDD partitions may depend on a single parent RDD partition



groupByKey



http://training.databricks.com/visualapi.pdf

Transformations and Actions

.map
.filter
.groupByKey
.reduceByKey

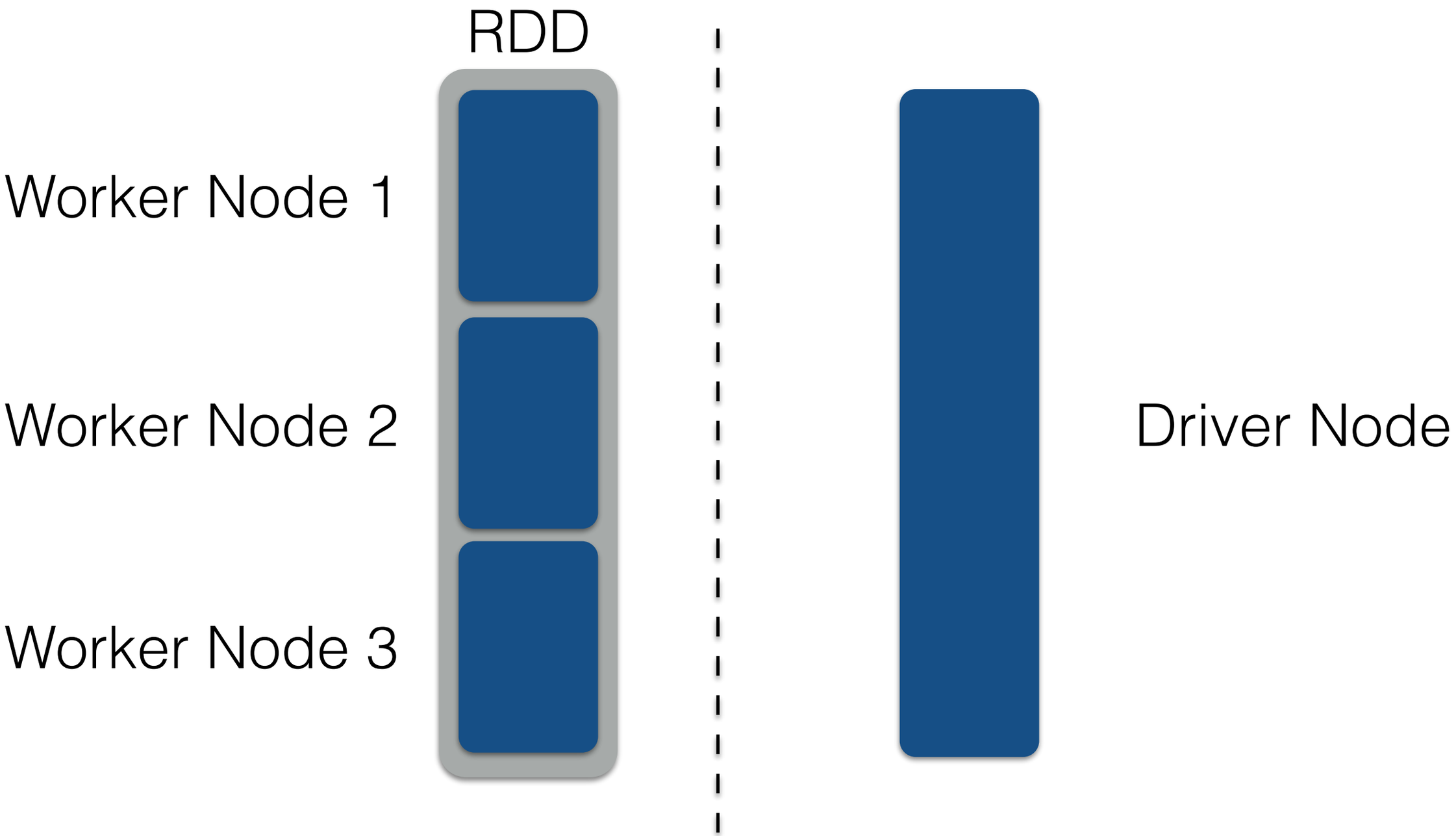
.sortByKey
.flatMap
.repartition

.collect
.take
.first

.count
.takeSample
.reduce

http://training.databricks.com/visualapi.pdf

.collect()



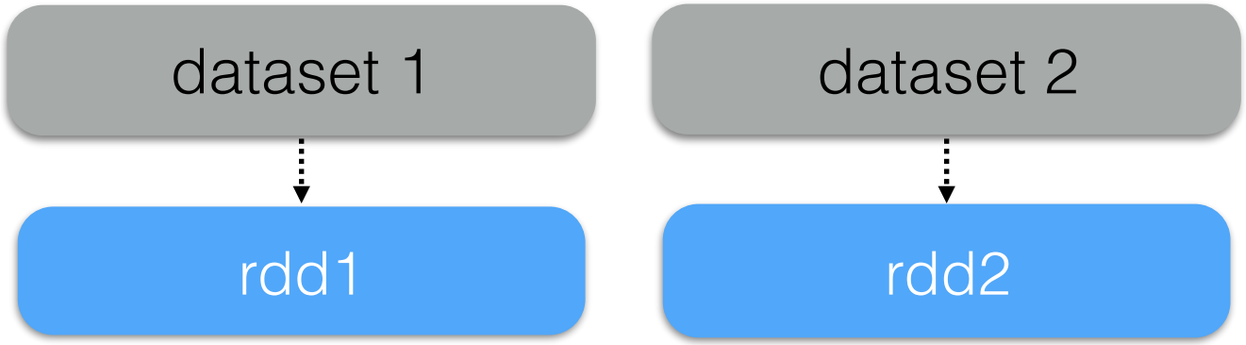
All partitions must fit into memory on the driver node

Building the Directed Acyclic Graph (DAG)

Building the DAG

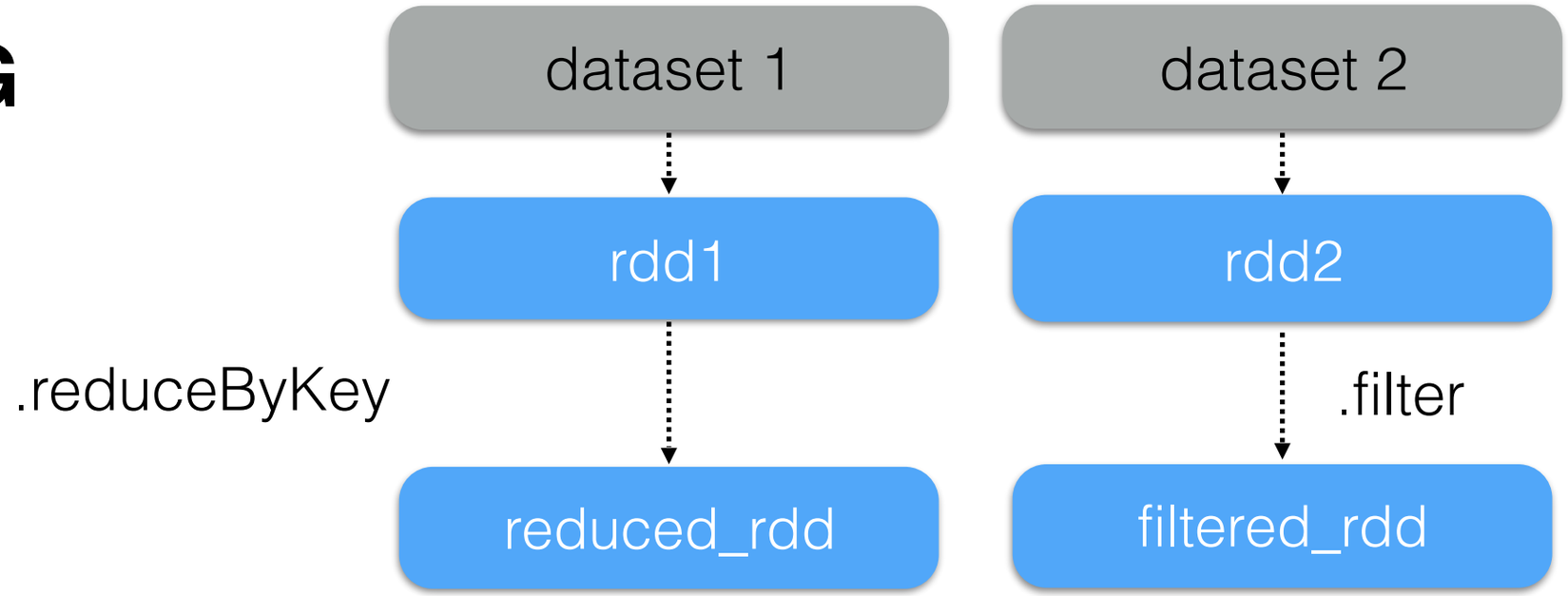
```
rdd1 = sc.textFile(“dataset1.txt”)
rdd2 = sc.textFile(“dataset2.txt”)

reduced_rdd = rdd1.reduceByKey(…)
filtered_rdd = rdd2.filter(…)

mapped_filtered_rdd = filtered_rdd.map(…)

joined_rdd = reduced_rdd.join(mapped_filtered_rdd)

mapped_joined_rdd = joined_rdd.map(…)

result = mapped_joined_rdd.collect()
```

Building the DAG




```
rdd1 = sc.textFile(“dataset1.txt”)
rdd2 = sc.textFile(“dataset2.txt”)

reduced_rdd = rdd1.reduceByKey(…)
filtered_rdd = rdd2.filter(…)

mapped_filtered_rdd = filtered_rdd.map(…)

joined_rdd = reduced_rdd.join(mapped_filtered_rdd)

mapped_joined_rdd = joined_rdd.map(…)

result = mapped_joined_rdd.collect()
```

Building the DAG



rdd1 = sc.textFile(“dataset1.txt”)
rdd2 = sc.textFile(“dataset2.txt”)

```
reduced_rdd = rdd1.reduceByKey(…)
filtered_rdd = rdd2.filter(…)
```

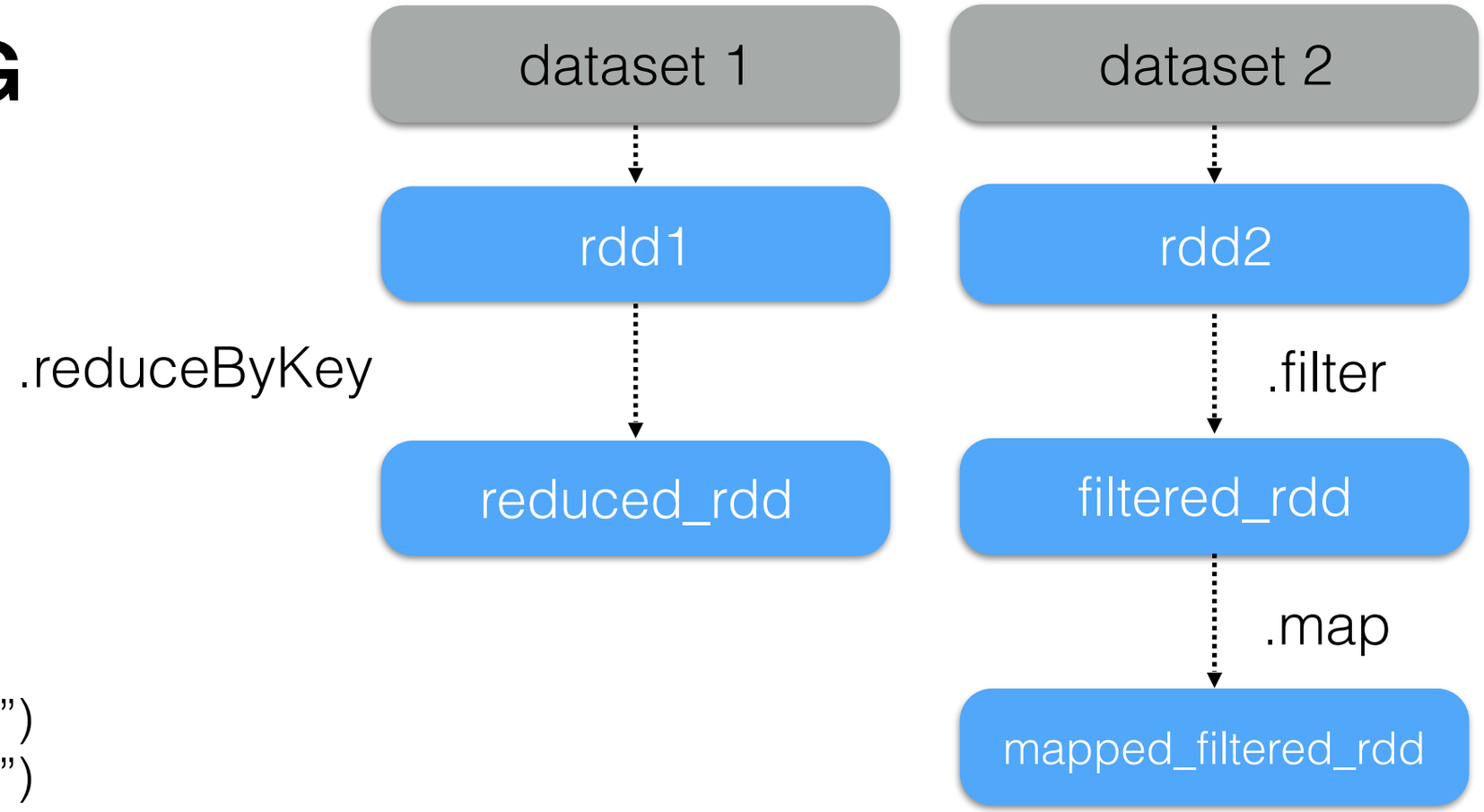
mapped_filtered_rdd = filtered_rdd.**map**(…)

joined_rdd = reduced_rdd.**join**(mapped_filtered_rdd)

mapped_joined_rdd = joined_rdd.**map**(…)

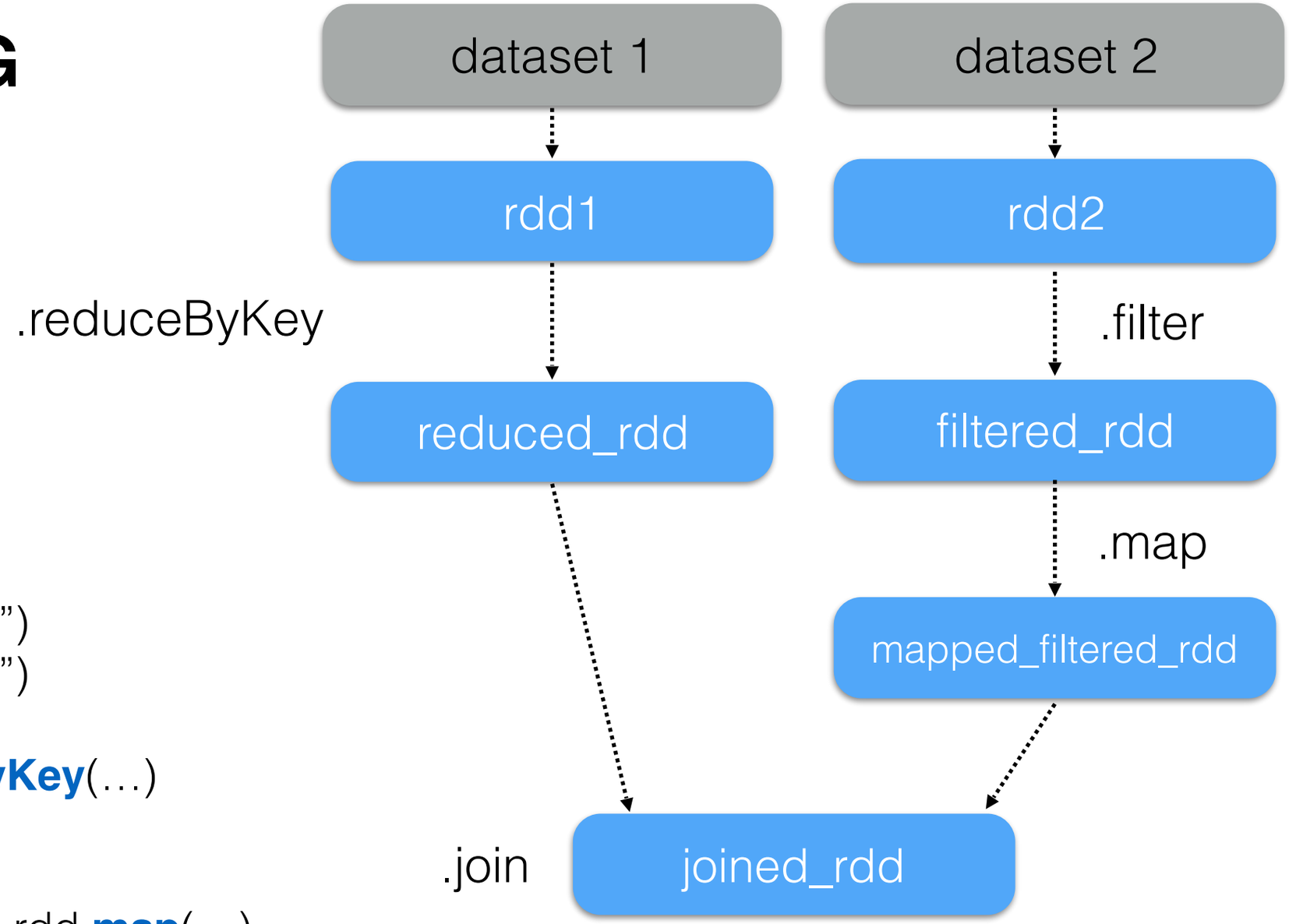
result = mapped_joined_rdd.**collect**()

Building the DAG




```
rdd1 = sc.textFile(“dataset1.txt”)
rdd2 = sc.textFile(“dataset2.txt”)

reduced_rdd = rdd1.reduceByKey(…)
filtered_rdd = rdd2.filter(…)

mapped_filtered_rdd = filtered_rdd.map(…)

joined_rdd = reduced_rdd.join(mapped_filtered_rdd)

mapped_joined_rdd = joined_rdd.map(…)

result = mapped_joined_rdd.collect()
```

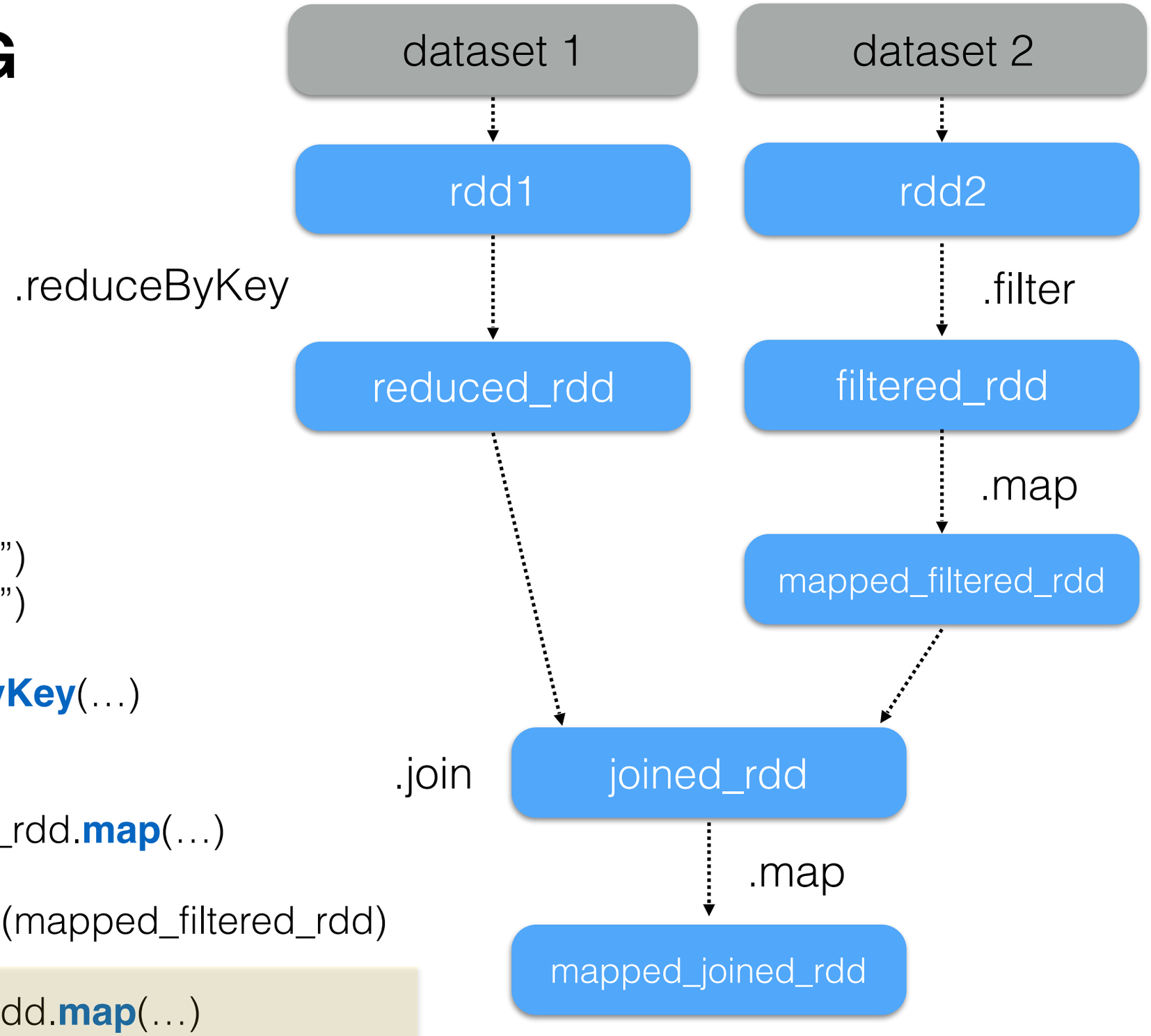
Building the DAG




```
rdd1 = sc.textFile(“dataset1.txt”)
rdd2 = sc.textFile(“dataset2.txt”)

reduced_rdd = rdd1.reduceByKey(…)
filtered_rdd = rdd2.filter(…)

mapped_filtered_rdd = filtered_rdd.map(…)

joined_rdd = reduced_rdd.join(mapped_filtered_rdd)

mapped_joined_rdd = joined_rdd.map(…)

result = mapped_joined_rdd.collect()
```

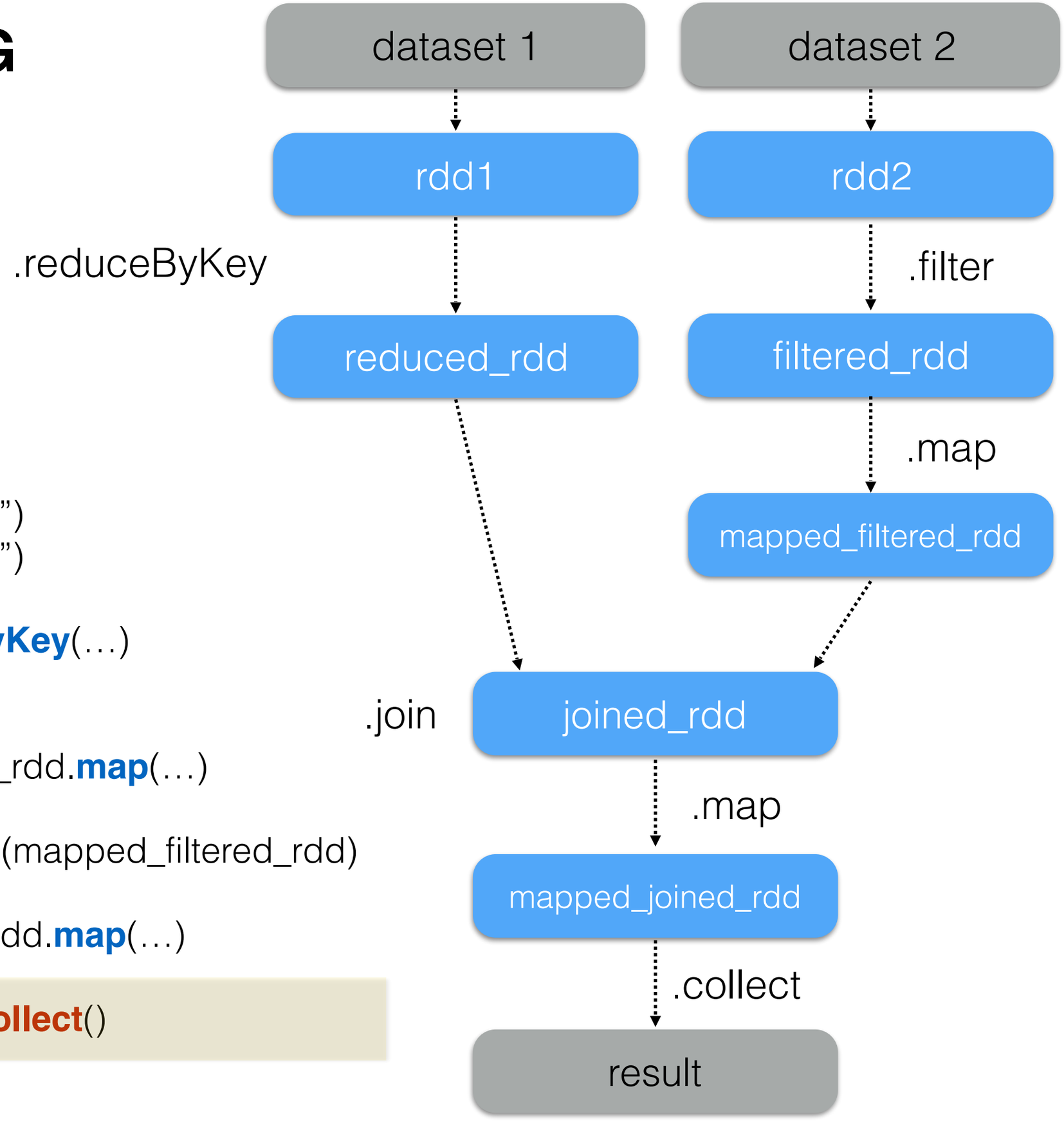
Building the DAG

```
rdd1 = sc.textFile(“dataset1.txt”)
rdd2 = sc.textFile(“dataset2.txt”)

reduced_rdd = rdd1.reduceByKey(…)
filtered_rdd = rdd2.filter(…)

mapped_filtered_rdd = filtered_rdd.map(…)

joined_rdd = reduced_rdd.join(mapped_filtered_rdd)

mapped_joined_rdd = joined_rdd.map(…)

result = mapped_joined_rdd.collect()
```



Building the DAG

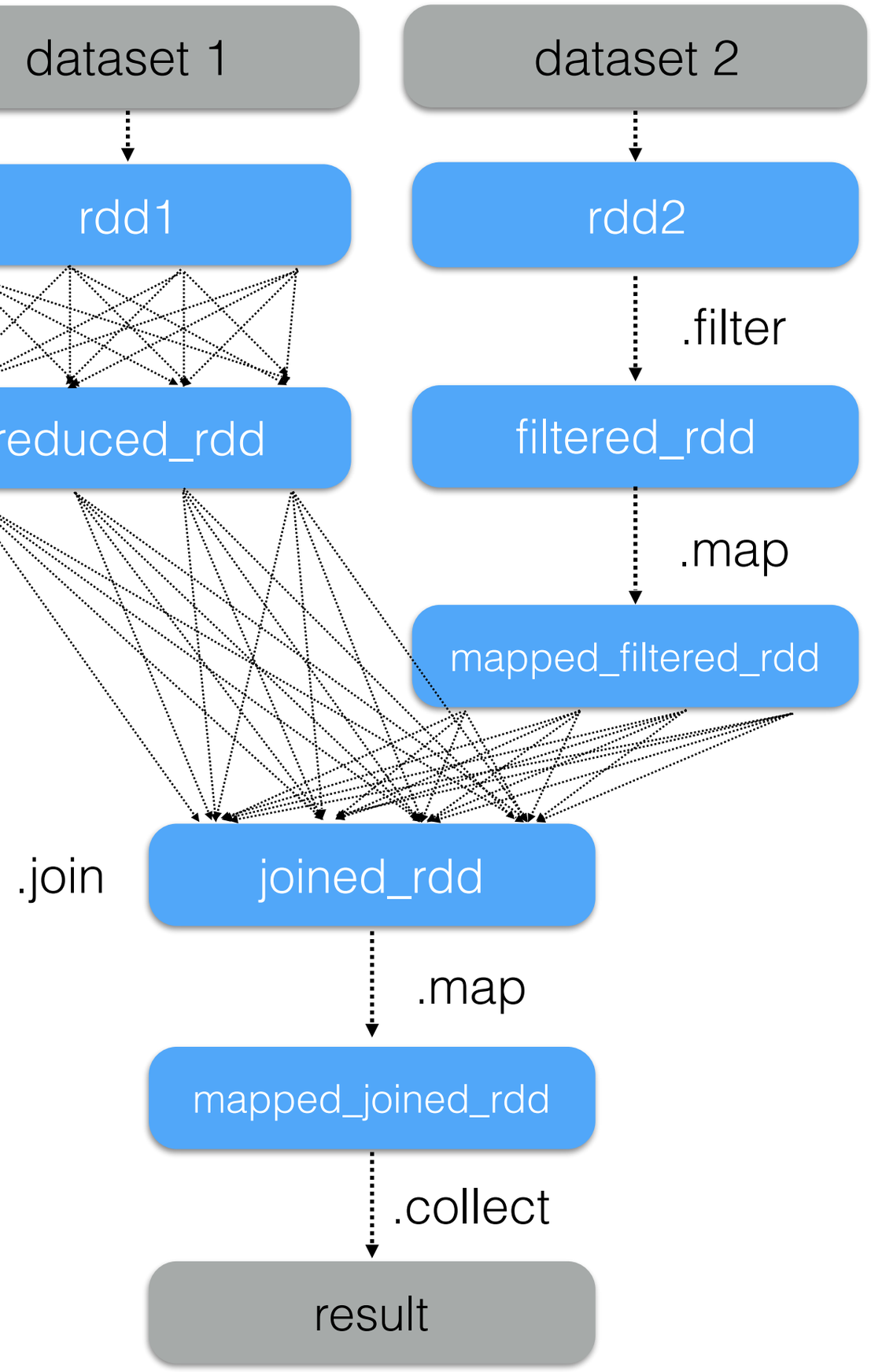
```
rdd1 = sc.textFile(“dataset1.txt”)
rdd2 = sc.textFile(“dataset2.txt”)

reduced_rdd = rdd1.reduceByKey(…)
filtered_rdd = rdd2.filter(…)

mapped_filtered_rdd = filtered_rdd.map(…)

joined_rdd = reduced_rdd.join(mapped_filtered_rdd)

mapped_joined_rdd = joined_rdd.map(…)

result = mapped_joined_rdd.collect()
```



Determine Stages

Look for
Shuffle Boundaries

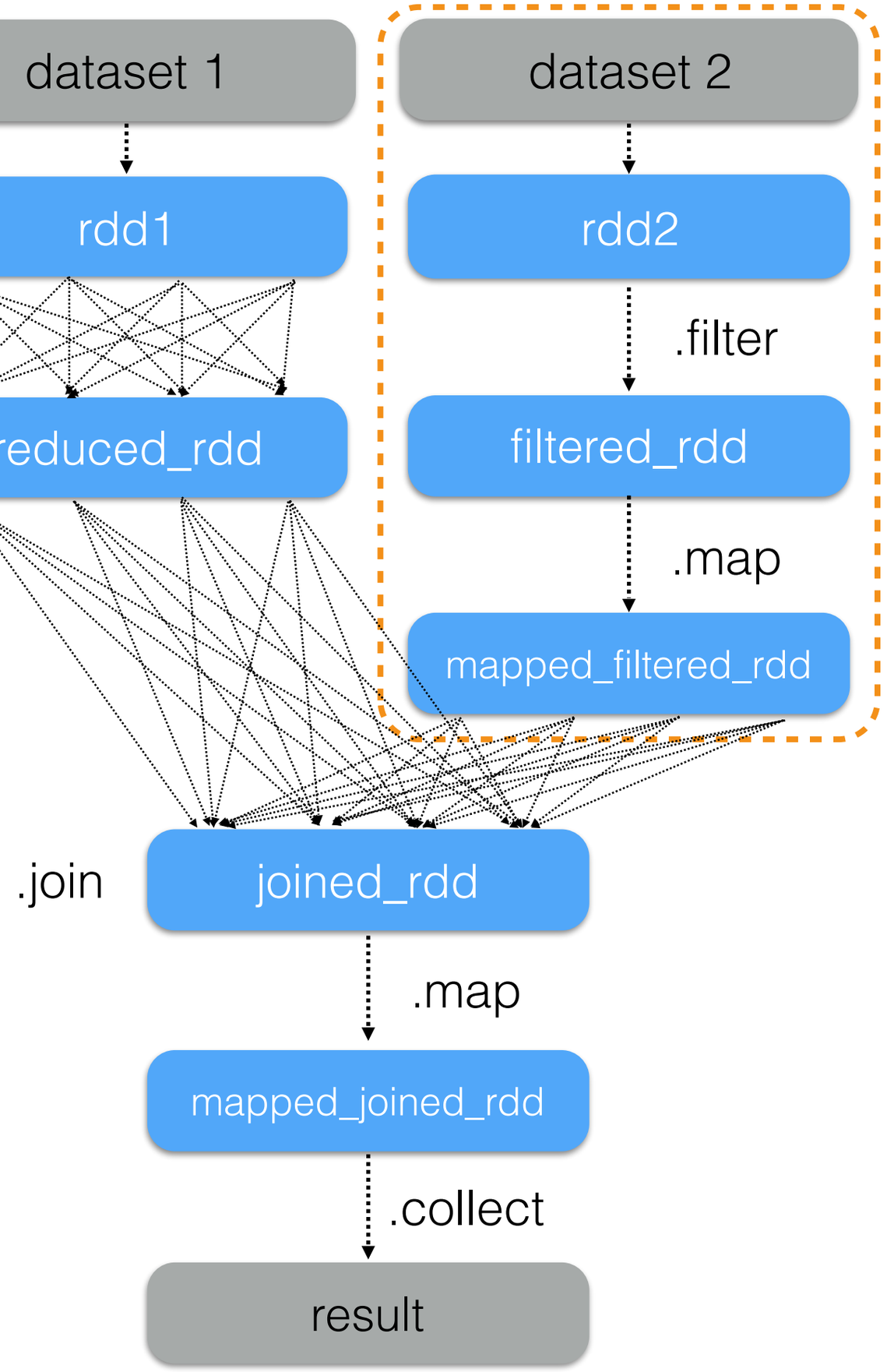
.reduceByKey



Determine Stages

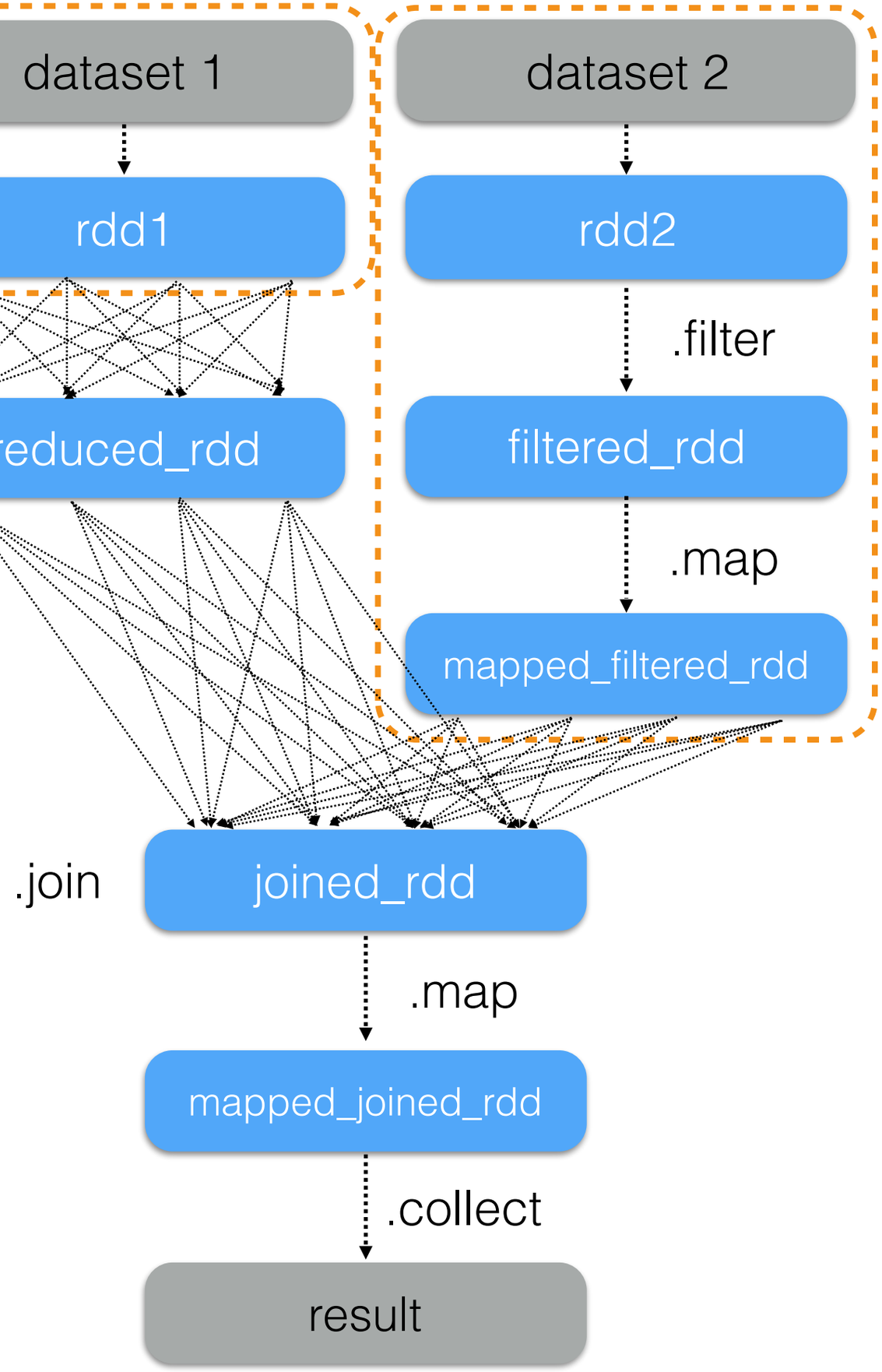
Look for
Shuffle Boundaries

.reduceByKey



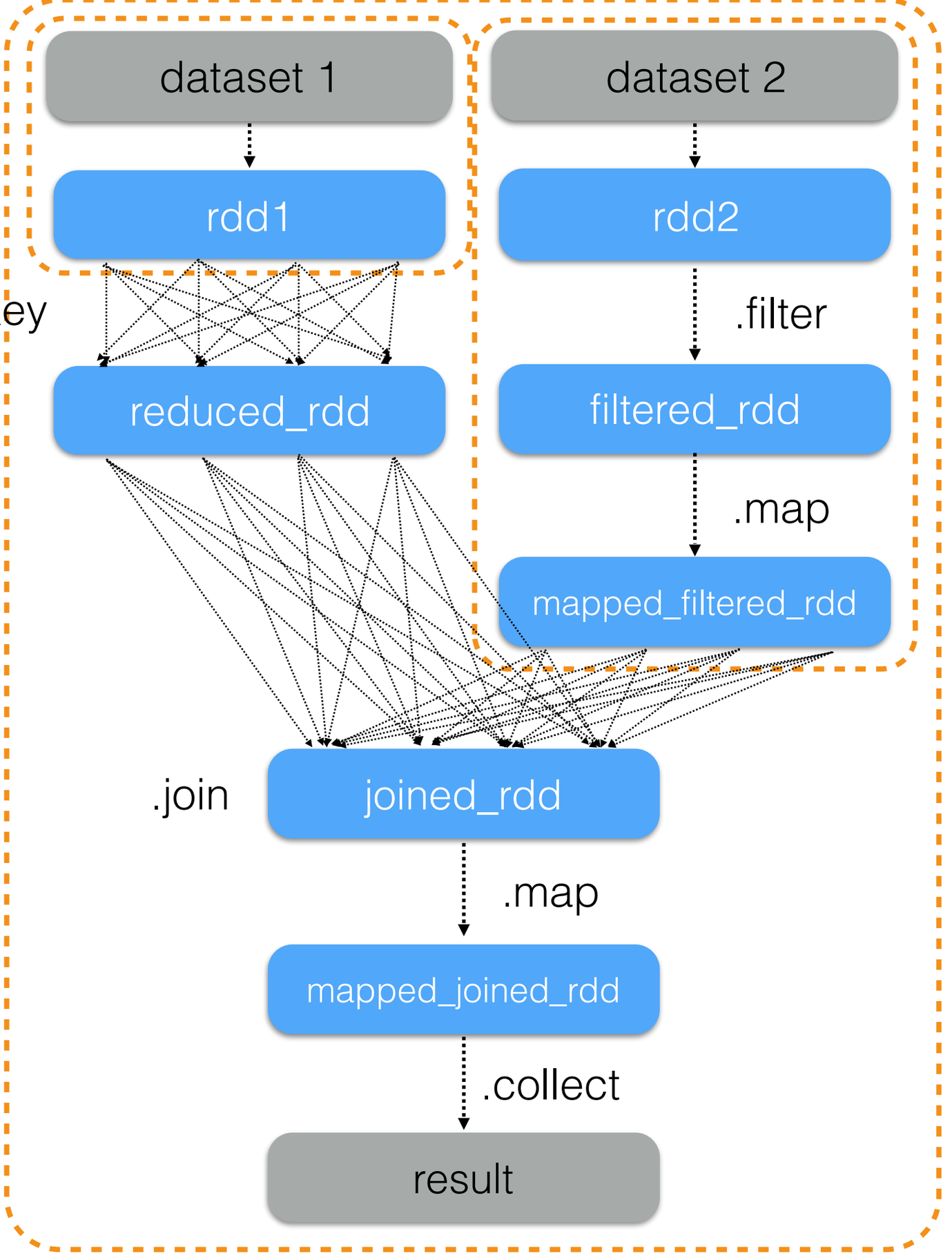
Determine Stages

Look for
Shuffle Boundaries

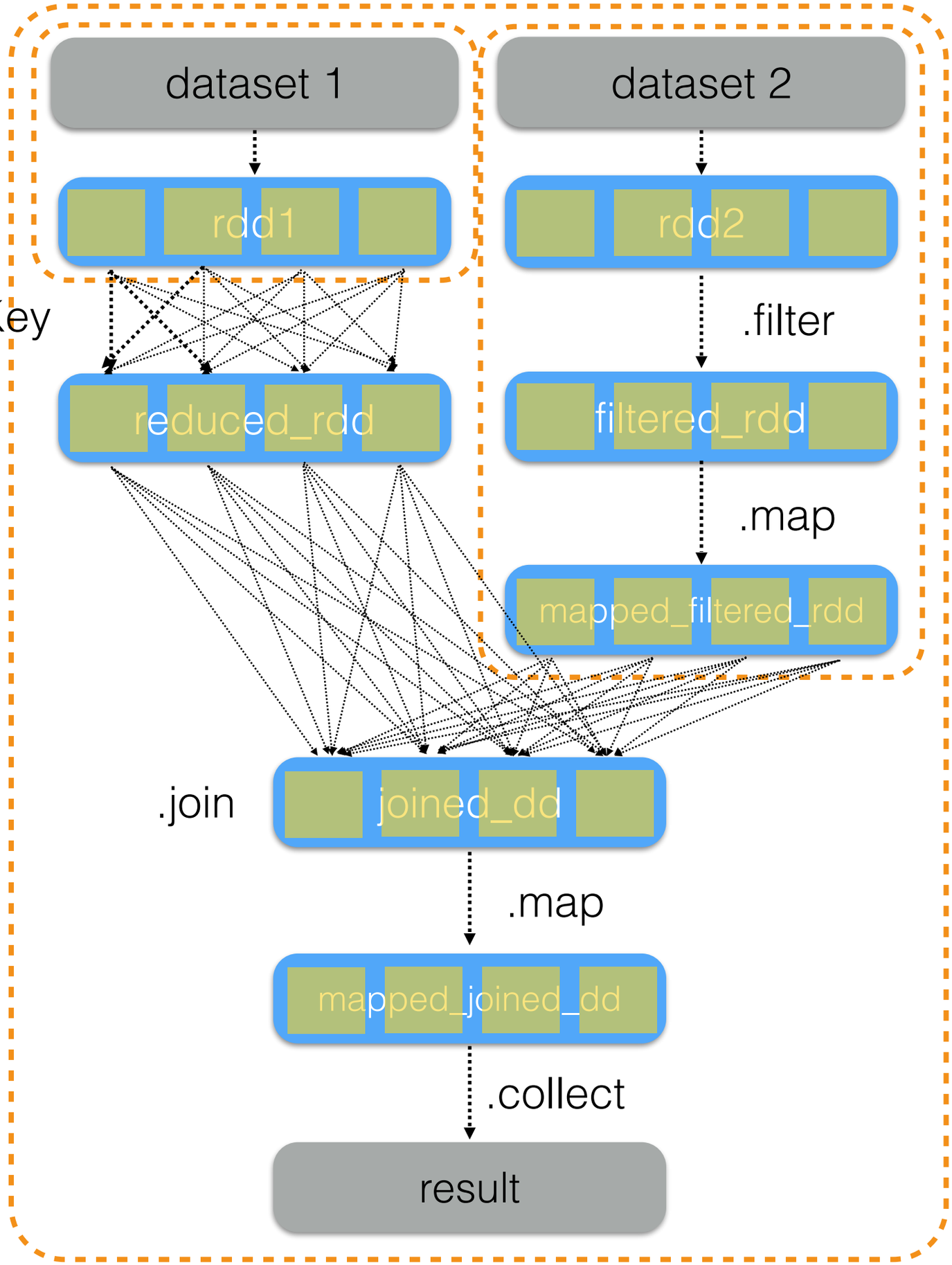


Determine Stages

Look for
Shuffle Boundaries



Run Data Through DAG

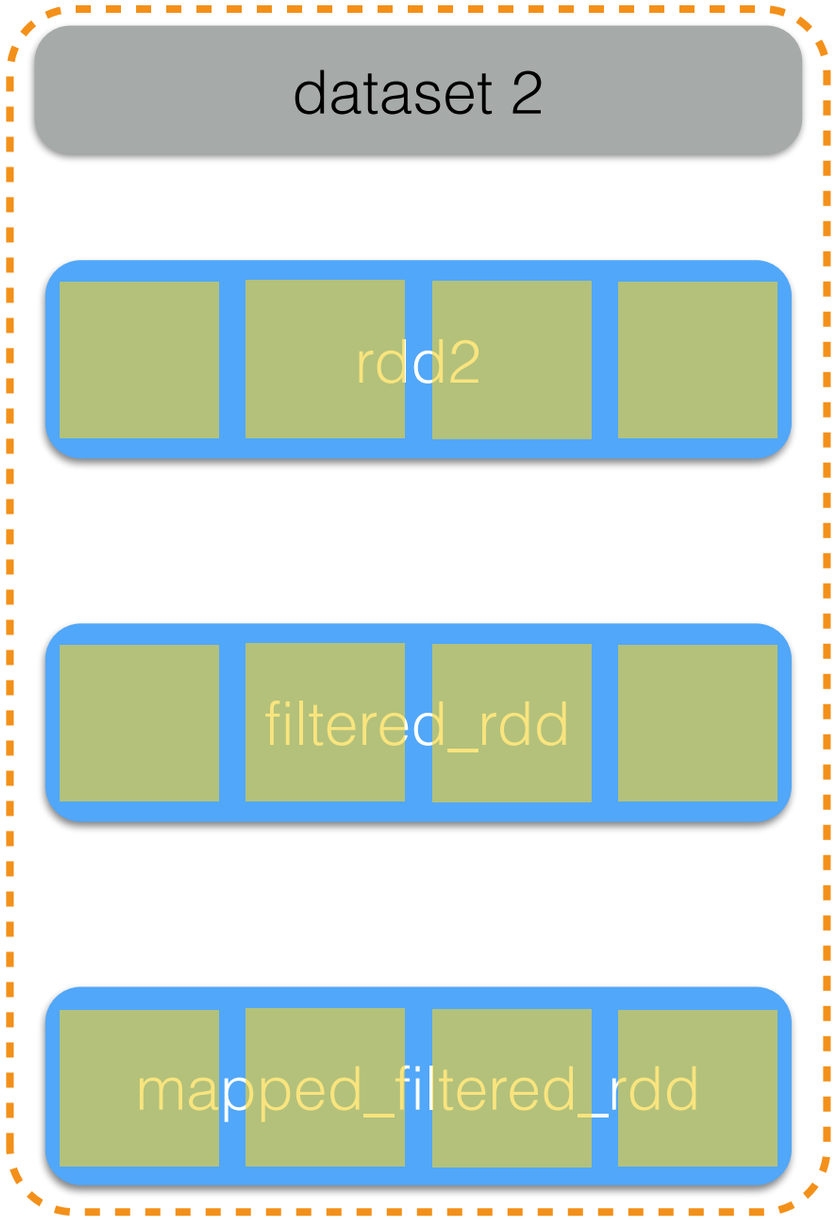


Data flow Through a Stage

Stage 0

.filter

.map

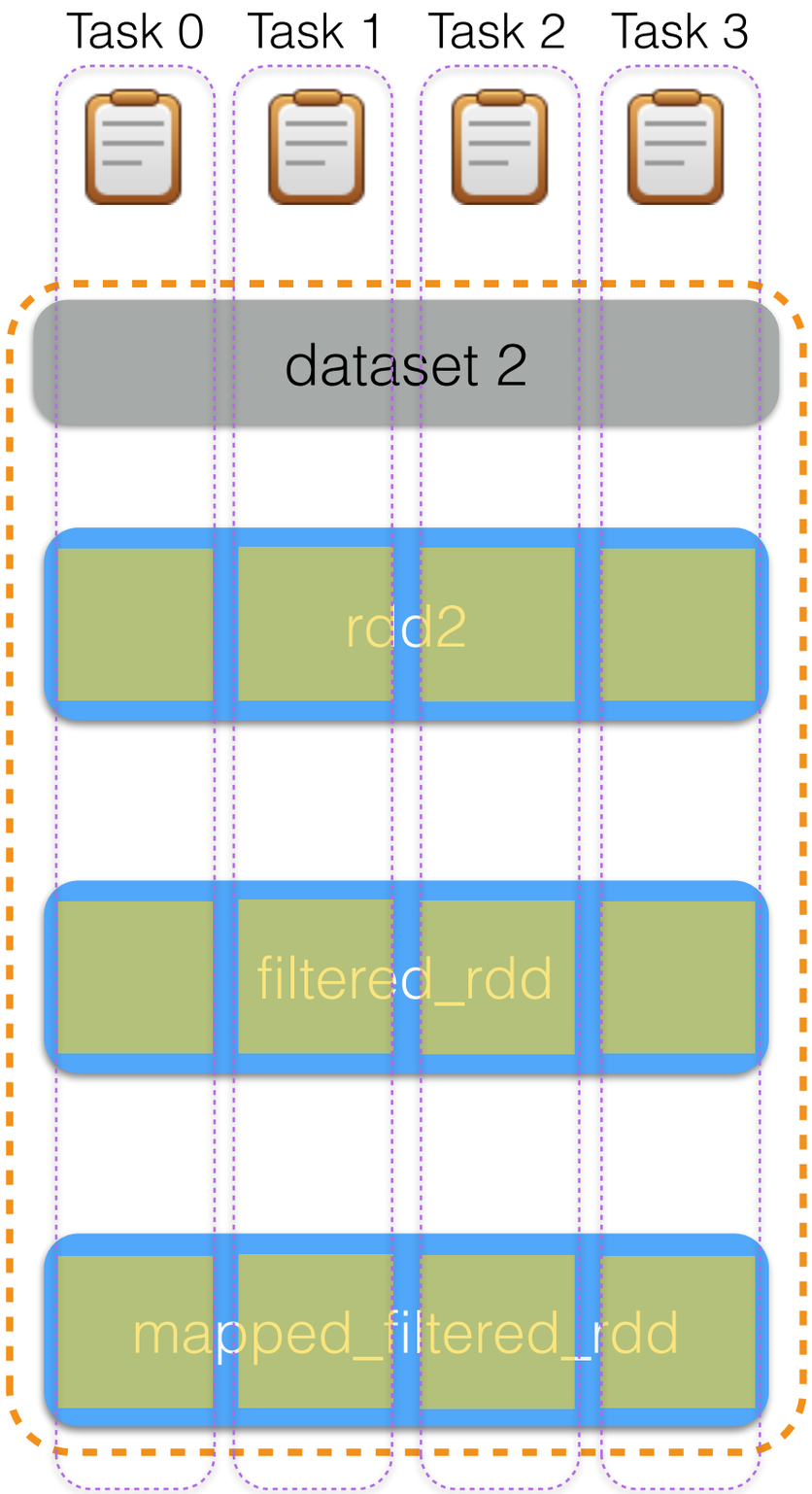


Data flow Through a Stage

Stage 0

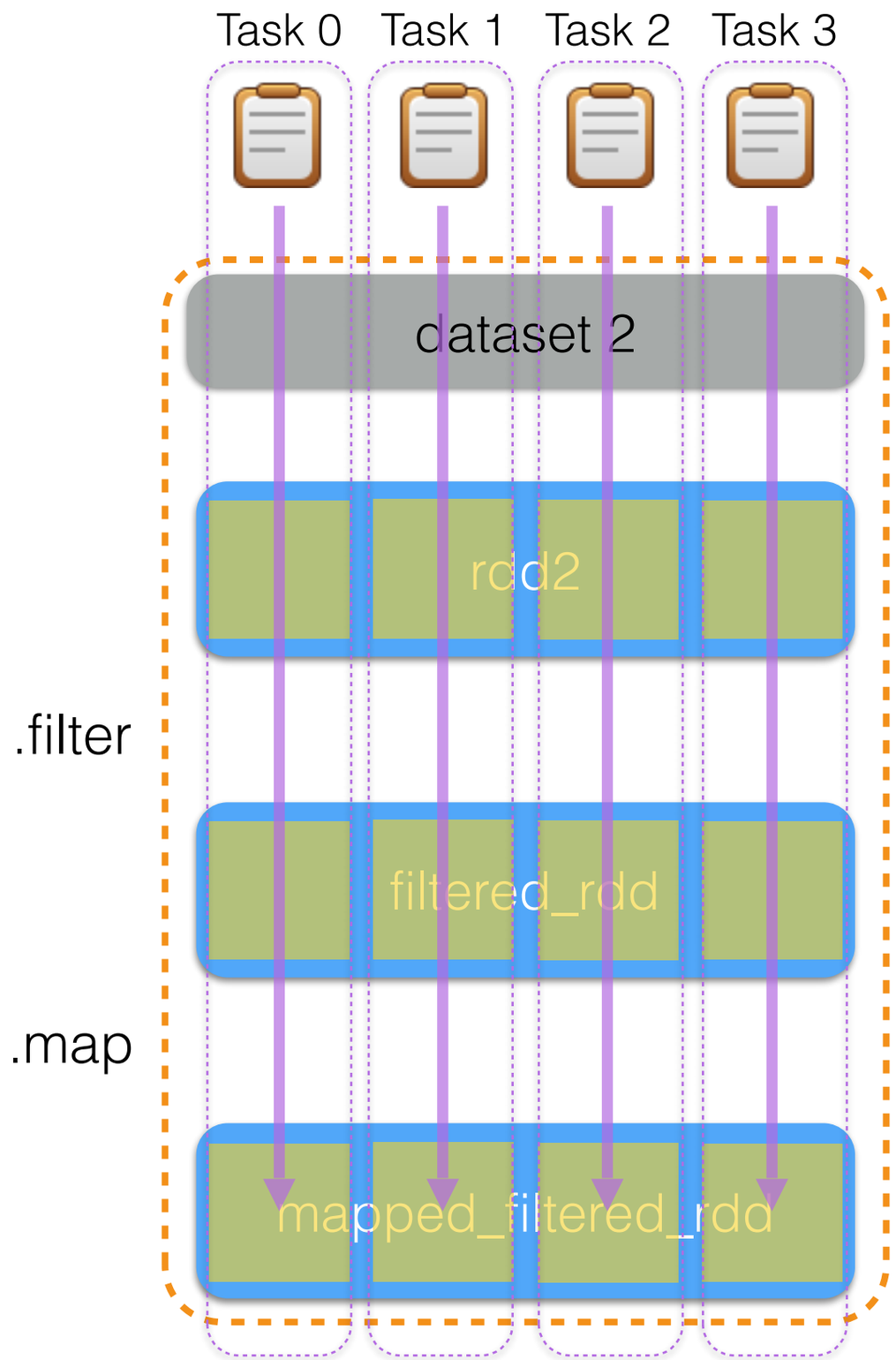
.filter

.map



Data flow Through a Stage

Stage 0



Jobs, Stages and Tasks

Jobs, Stages, Tasks

- Jobs are triggered by Actions
- Jobs are composed of one or more Stages
- Stages are composed of one or more Tasks

Jobs, Stages, Tasks

- Jobs are triggered by Actions
- Jobs are composed of one or more Stages
- Stages are composed of one or more Tasks

Job

ACTIONS

.**collect**()

.**take**()

.**first**()

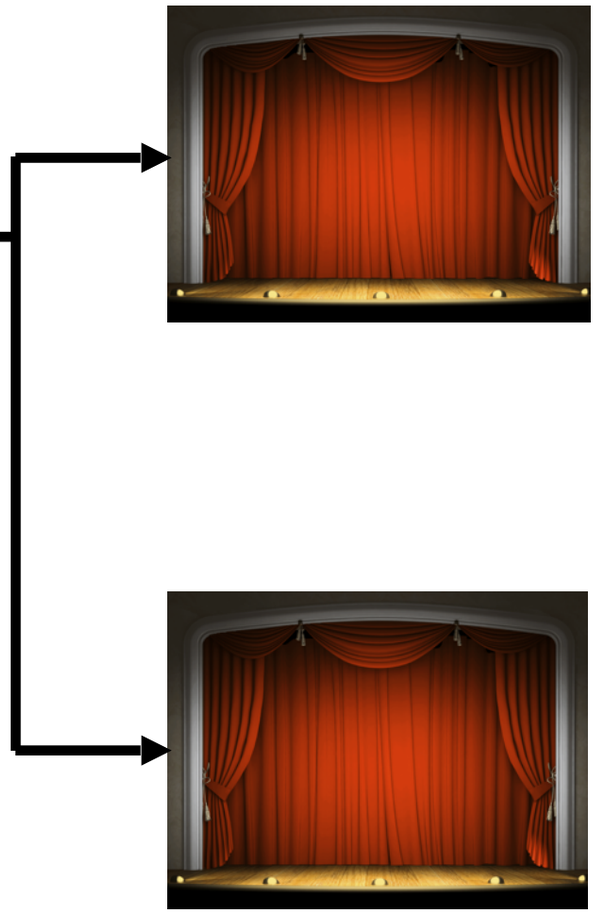
.**takeSample**()

…

Jobs, Stages, Tasks

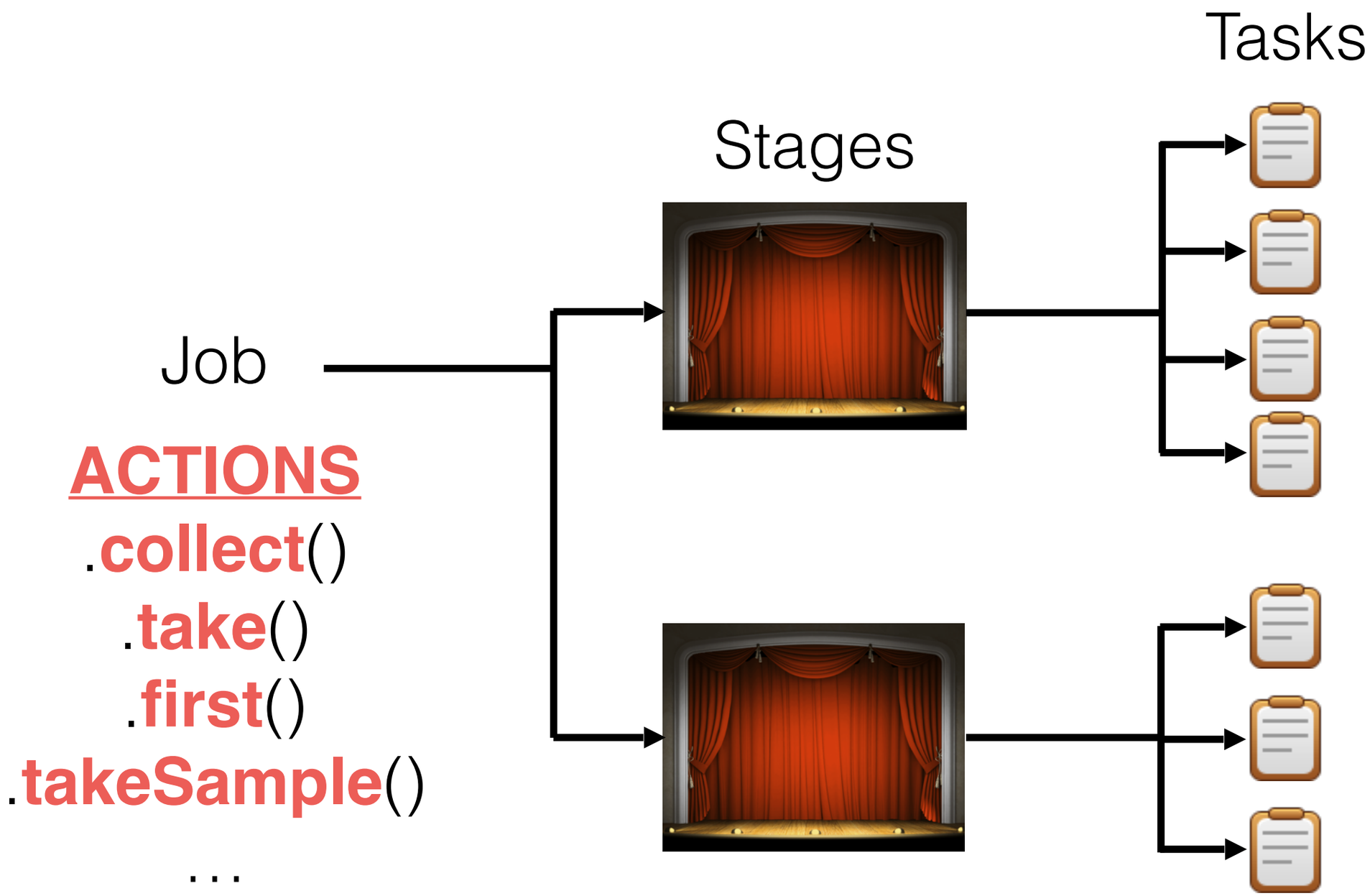
- Jobs are triggered by Actions
- Jobs are composed of one or more Stages
- Stages are composed of one or more Tasks

Stages



Jobs, Stages, Tasks

- Jobs are triggered by Actions
- Jobs are composed of one or more Stages
- Stages are composed of one or more Tasks



Ways to Run Spark

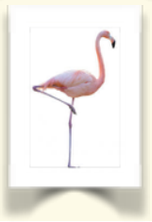
Local

Standalone

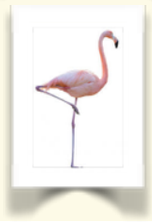
YARN

Mesos

Ways to Run Spark

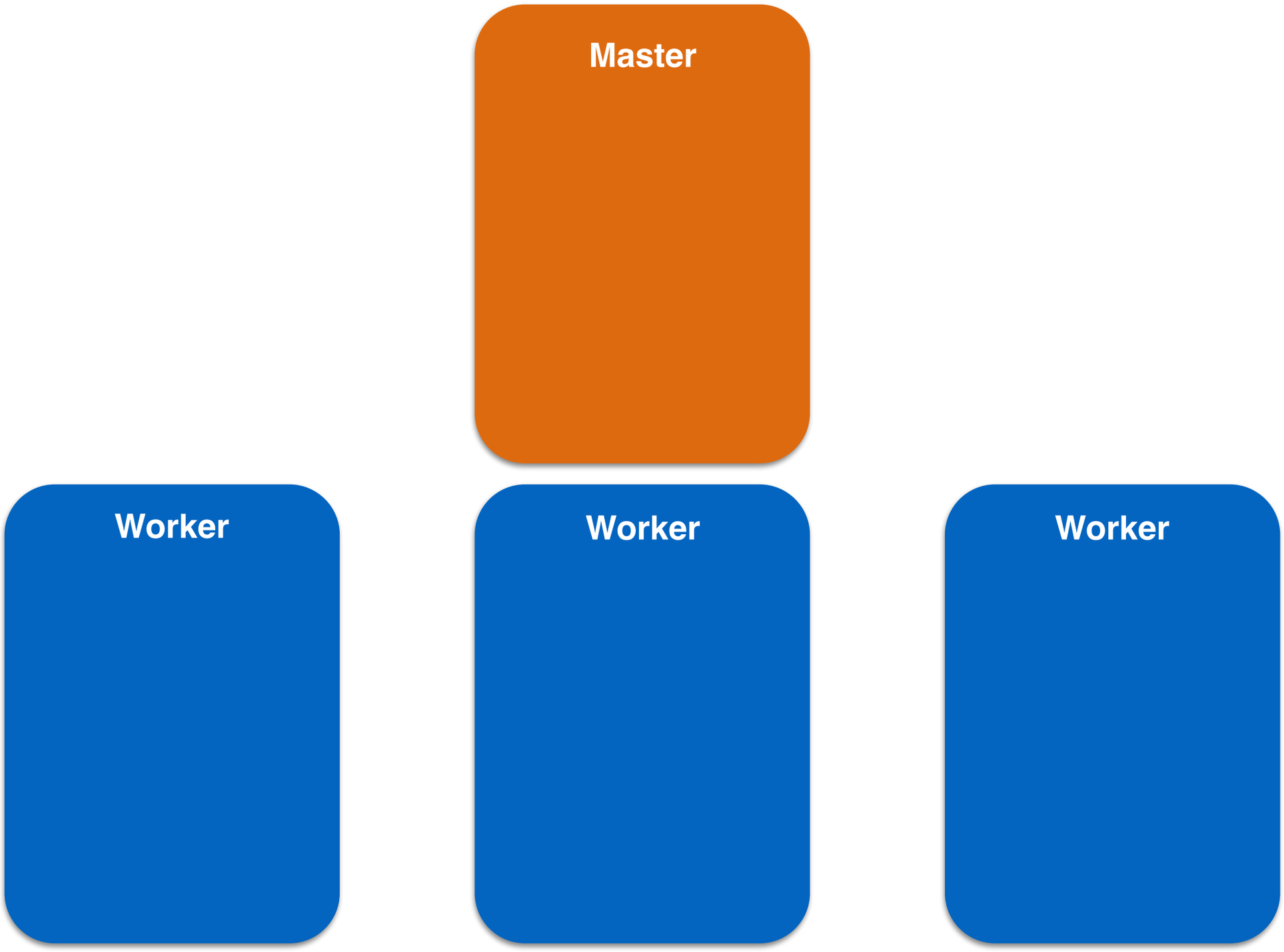
Static Resource Management		
	Local	
	Standalone	
	YARN	Dynamic Resource Management
	Mesos	

Ways to Run Spark

Static Resource Management		
	Local	
	Standalone	
	YARN	
	Mesos	

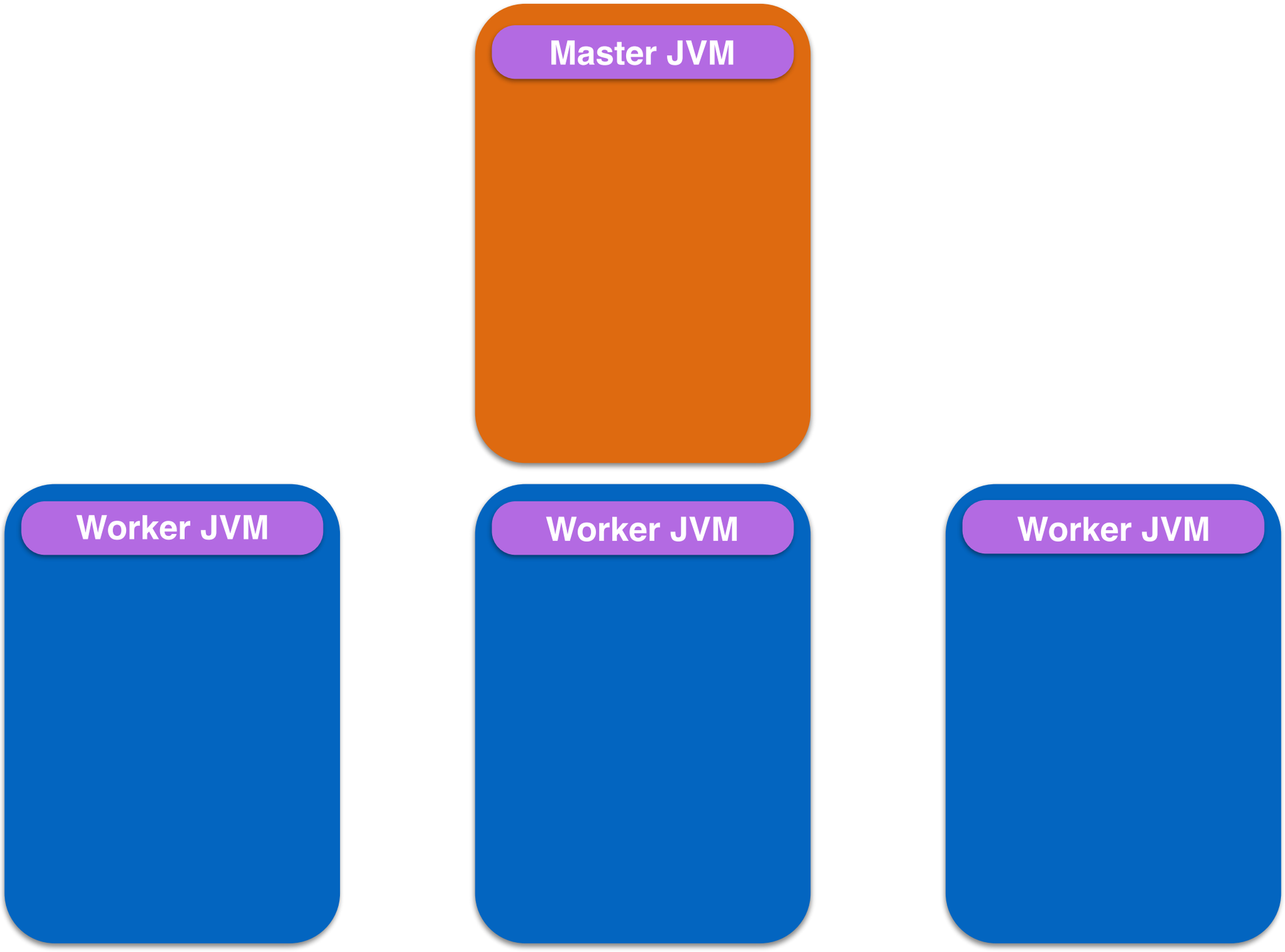
Dynamic Resource Management

Standalone Mode



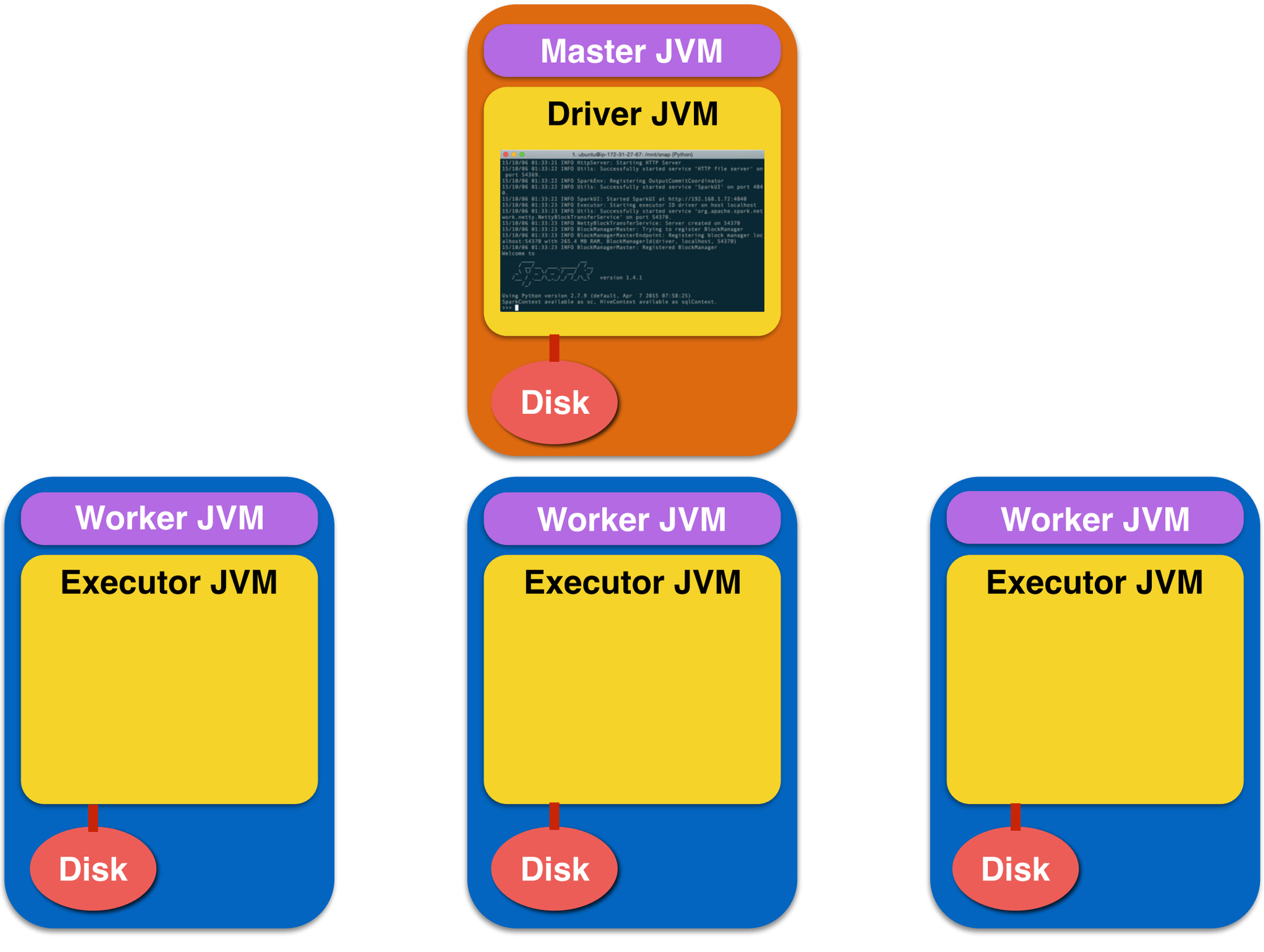
Standalone Mode

$SPARK_HOME/sbin/**start-all.sh**



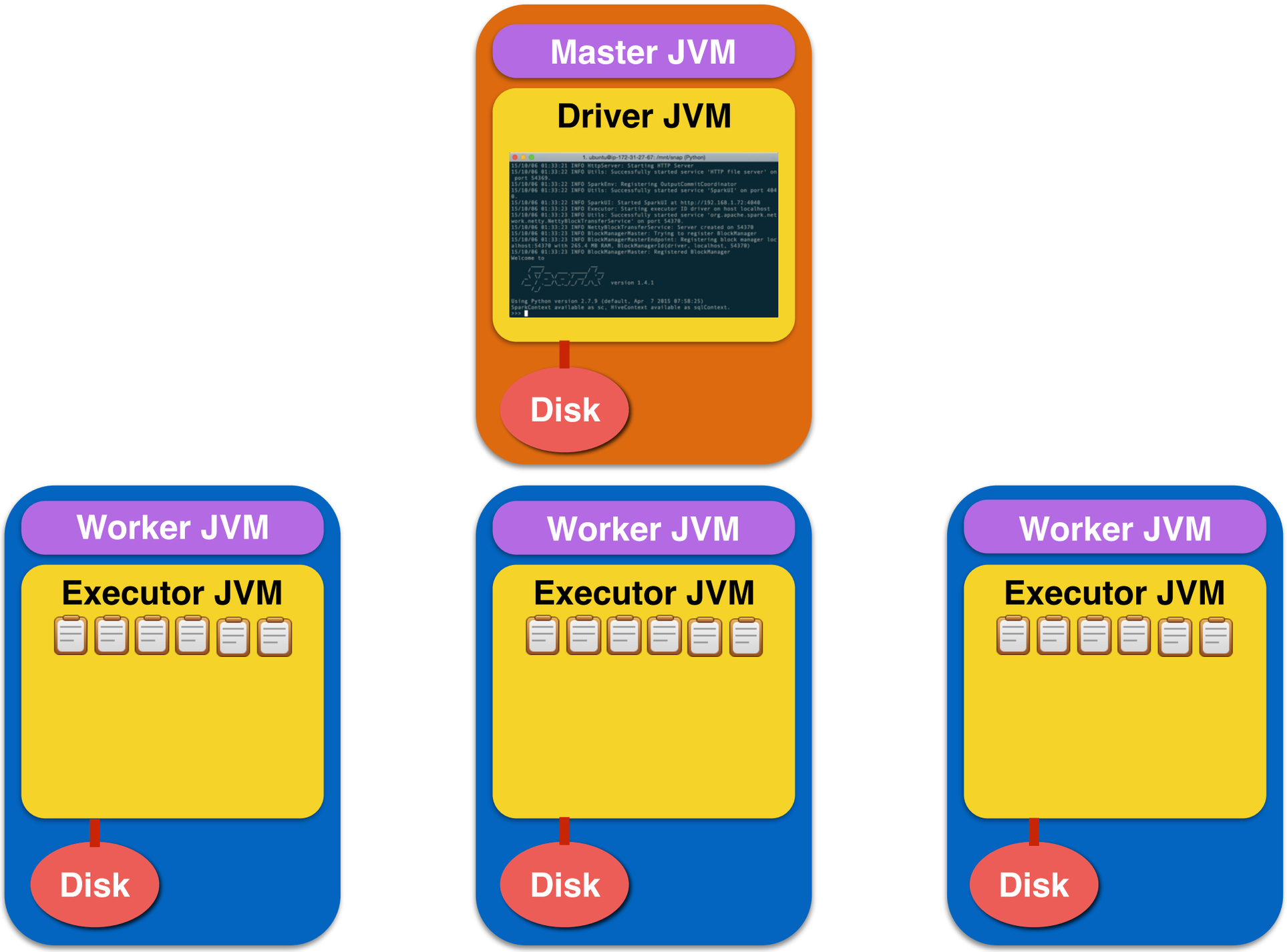
Standalone Mode

pyspark --master spark://master_hostname:7077
spark-shell --master spark://master_hostname:7077



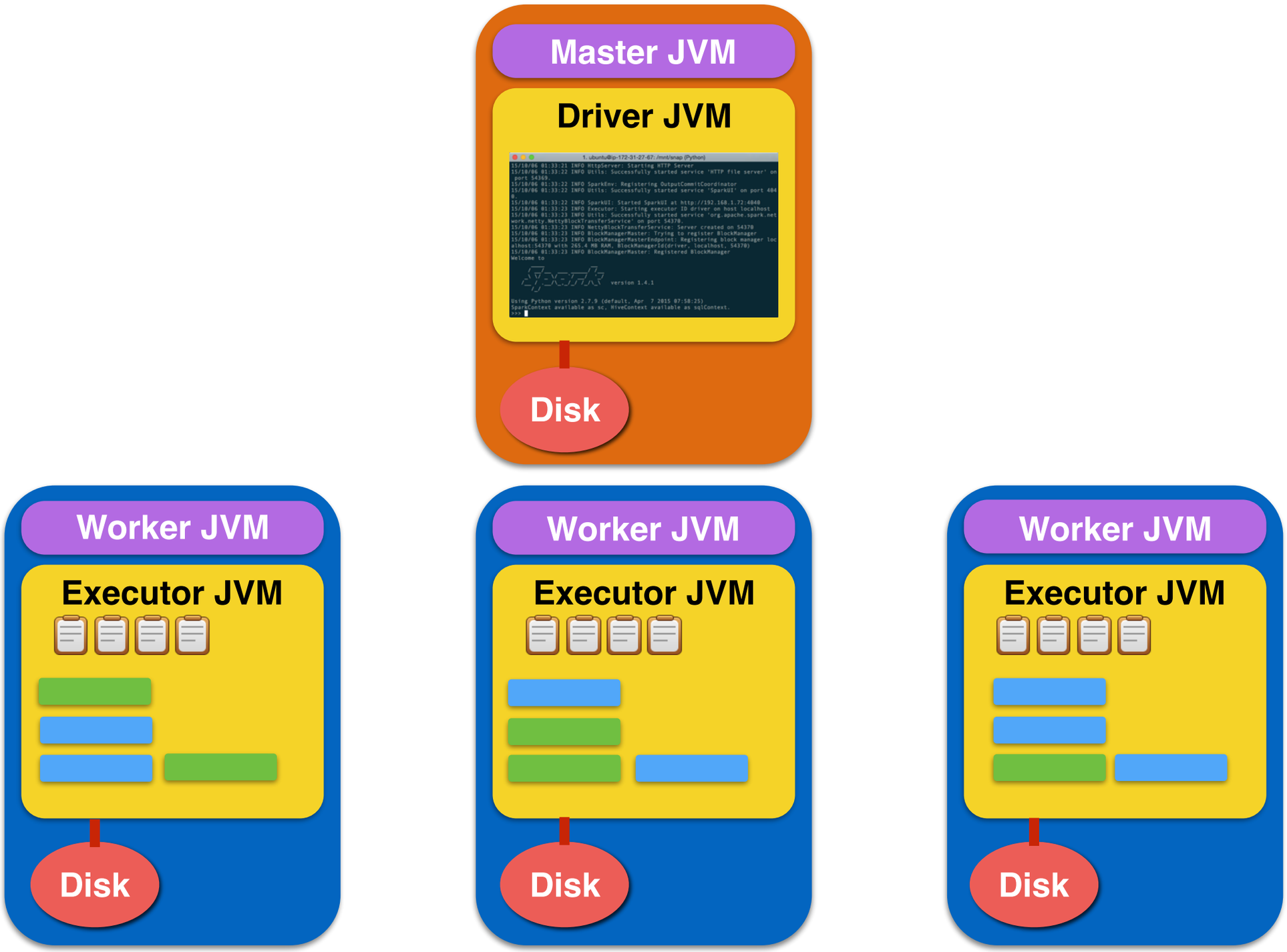
Standalone Mode

$SPARK_HOME/conf/**spark-env.sh**
export SPARK_WORKER_CORES=6



Standalone Mode

Parallelism with multiple partitions in RDD



Spark DataFrames and Spark SQL

DataFrames and Spark SQL

RDD

```
(1420589581, CEC023AA, 40712)
(1436660661, 175AB332, 60813)
(1469828518, 407BA6EF, 70179)
…
```

DataFrame

```
Row(timestamp=1420589581, uid=CEC023AA, zip=40712)
Row(timestamp=1436660661, uid=175AB332, zip=60813)
Row(timestamp=1469828518, uid=407BA6EF, zip=70179)
…
```

DataFrames and Spark SQL

DataFrame

df

```
Row(timestamp=1420589581, uid=CEC023AA, zip=40712)
Row(timestamp=1436660661, uid=175AB332, zip=60813)
Row(timestamp=1469828518, uid=407BA6EF, zip=70179)
…
```

```
zip_grouped_DF = df.select("zip","uid")\
                   .distinct()\
                   .groupBy(“zip")\
                   .count()\
                   .withColumnRenamed("count", “unique_users")\
                   .orderBy(“unique_users")
```

DataFrames and Spark SQL

DataFrame

df

```
Row(timestamp=1420589581, uid=CEC023AA, zip=40712)
Row(timestamp=1436660661, uid=175AB332, zip=60813)
Row(timestamp=1469828518, uid=407BA6EF, zip=70179)
…
```

```
df.registerTempTable("my_table")

zip_grouped_DF = sqlContext.sql("""
     SELECT zip, COUNT(DISTINCT uid) as unique_users
     FROM my_table
     GROUP BY zip
     ORDER BY unique_users
     """)
```

Machine Learning on Spark

Differences between MLLib and ML Pipeline

MLLib

Inputs/Outputs are RDDs

Pros: Many available APIs for distributed ML processing

Cons: Must massage RDDs to fit specifically the input requirements of APIs

ML Pipeline

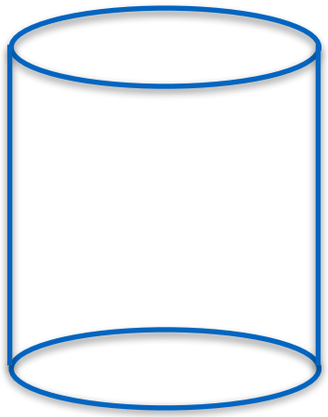
Inputs/Outputs are DataFrames

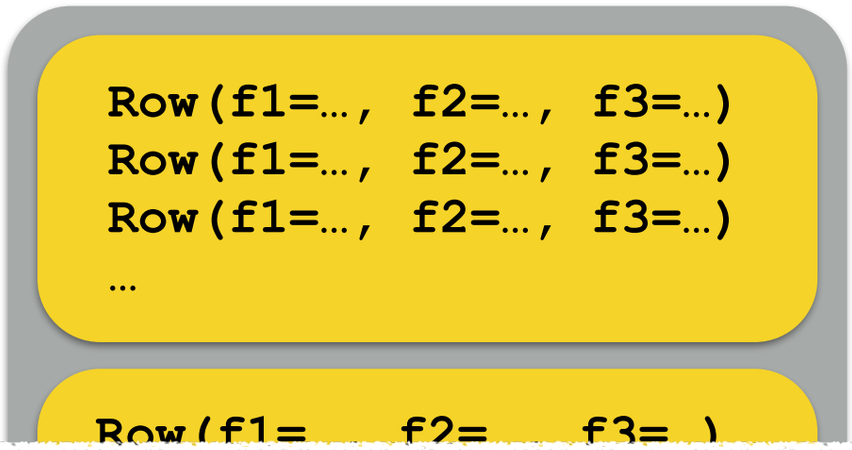
Pros: High level API to help with fast iterations on ML data processing

Cons: Can be less flexible in some cases

ML Pipeline Concepts

- **DataFrame**
- Transformer
- Estimator
- Pipeline

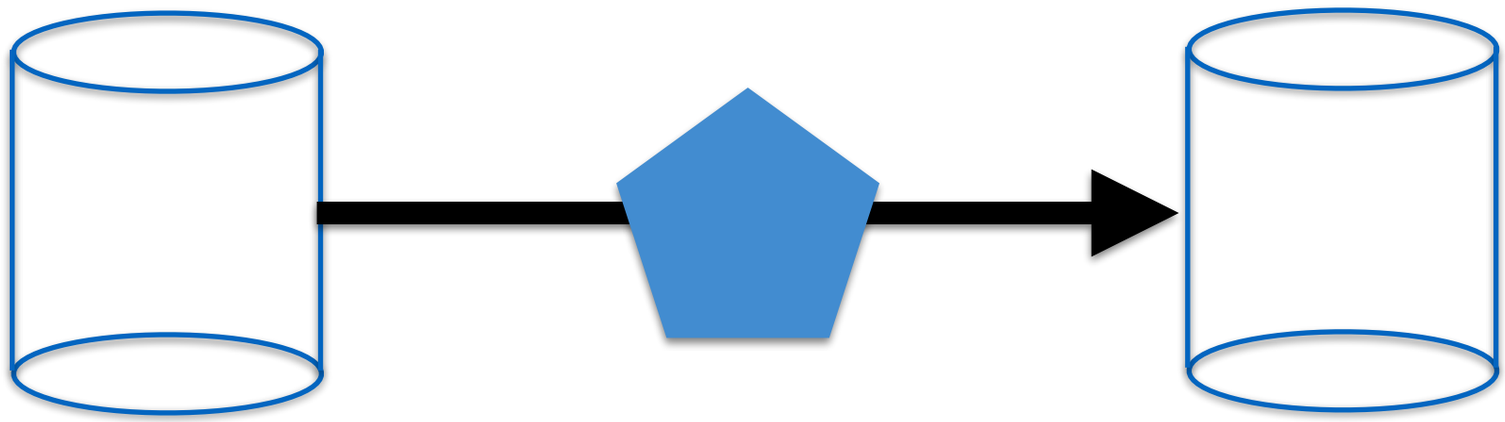




ML Pipeline Concepts

- DataFrame
- **Transformer**
- Estimator
- Pipeline

Transformer

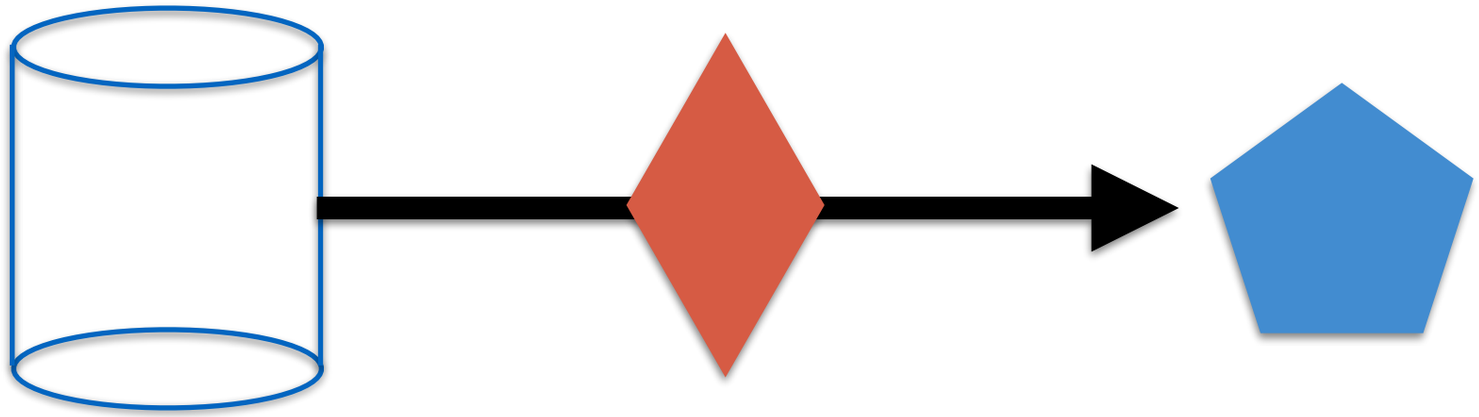


Tokenizer
Binarizer
Word2Vec
…

ML Pipeline Concepts

- DataFrame
- Transformer
- **Estimator**
- Pipeline

Estimator **Transformer**



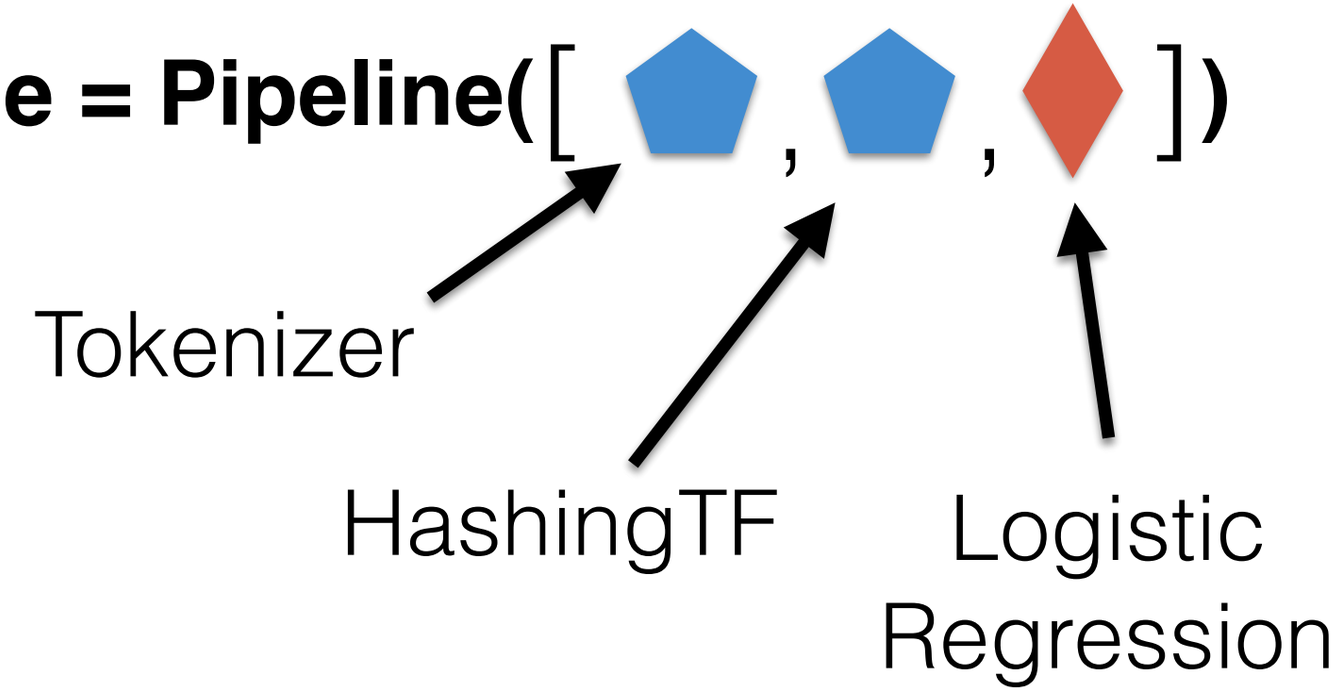
LogisticRegression
LinearRegression
RandomForestRegressor
…

ML Pipeline Concepts

- DataFrame
- Transformer
- Estimator
- **Pipeline**

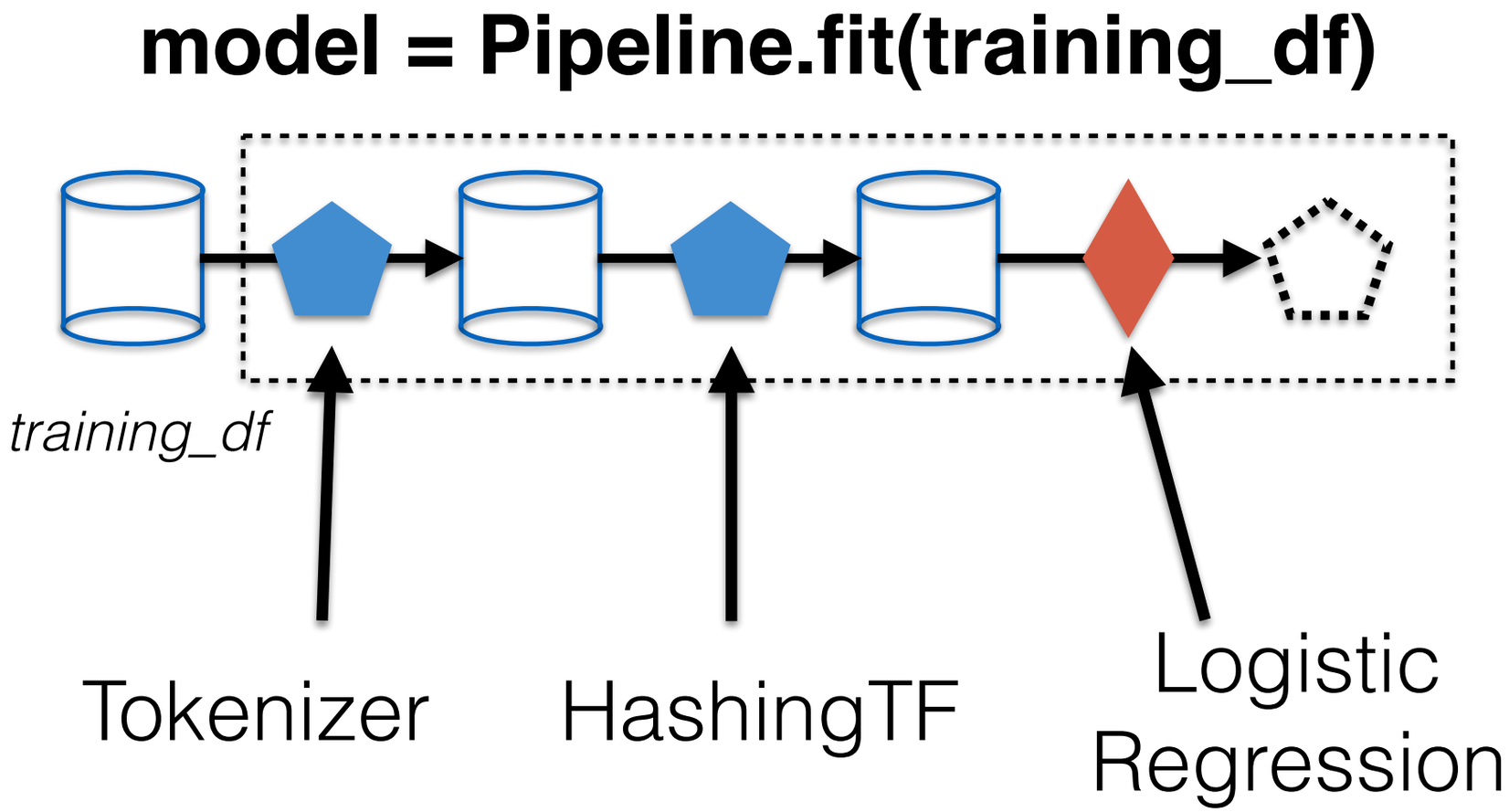
Pipeline is an Estimator 

pipeline = Pipeline([, ,])



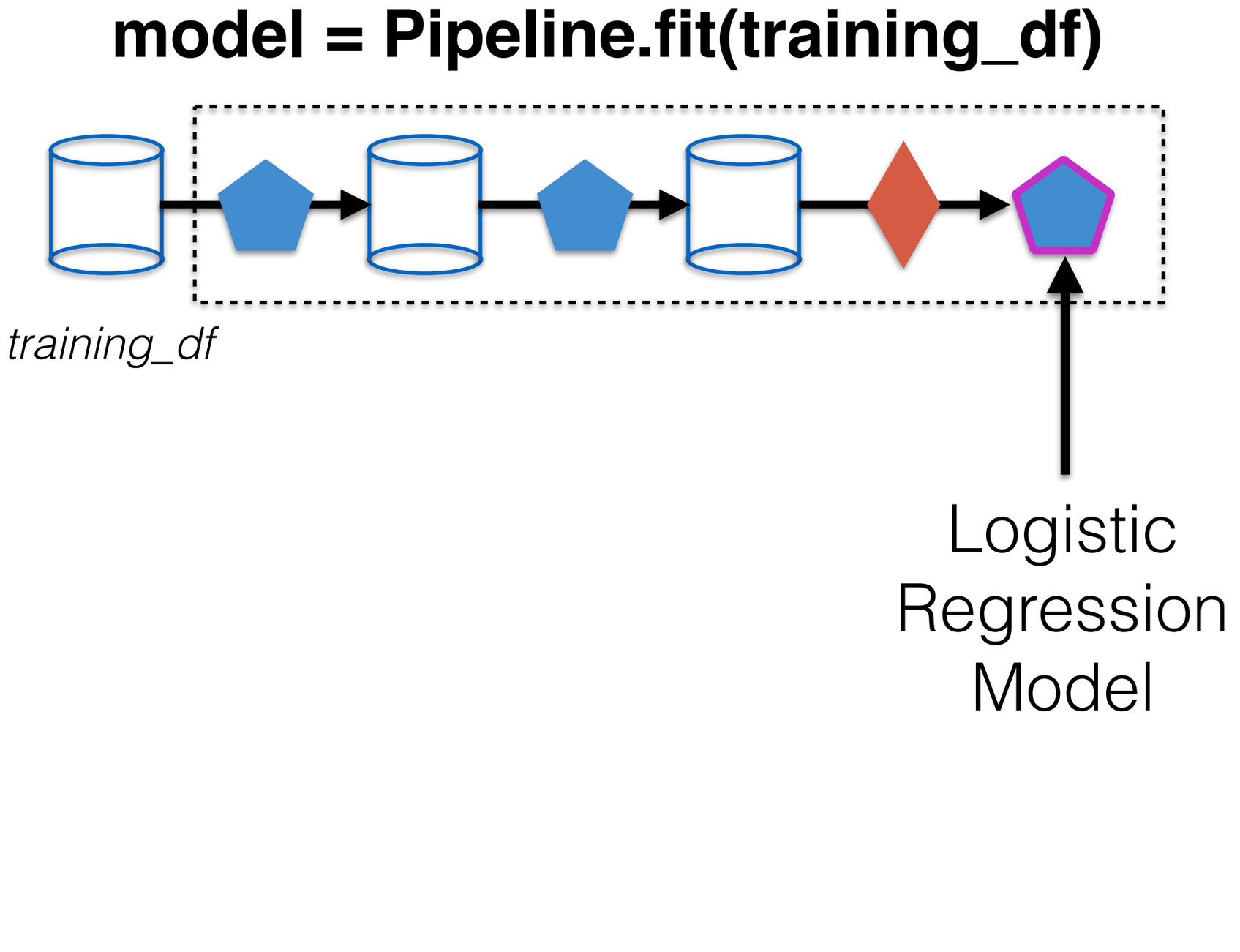
ML Pipeline Concepts

- DataFrame
- Transformer
- Estimator
- **Pipeline**



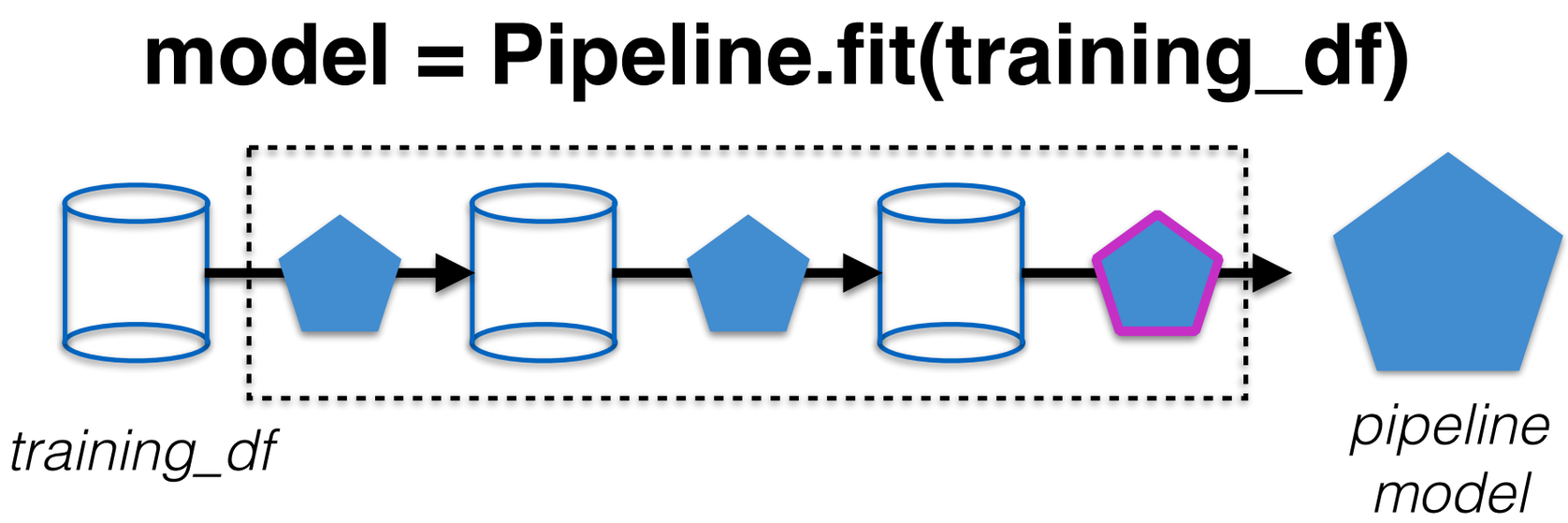
ML Pipeline Concepts

- DataFrame
- Transformer
- Estimator
- **Pipeline**



ML Pipeline Concepts

- DataFrame
- Transformer
- Estimator
- **Pipeline**

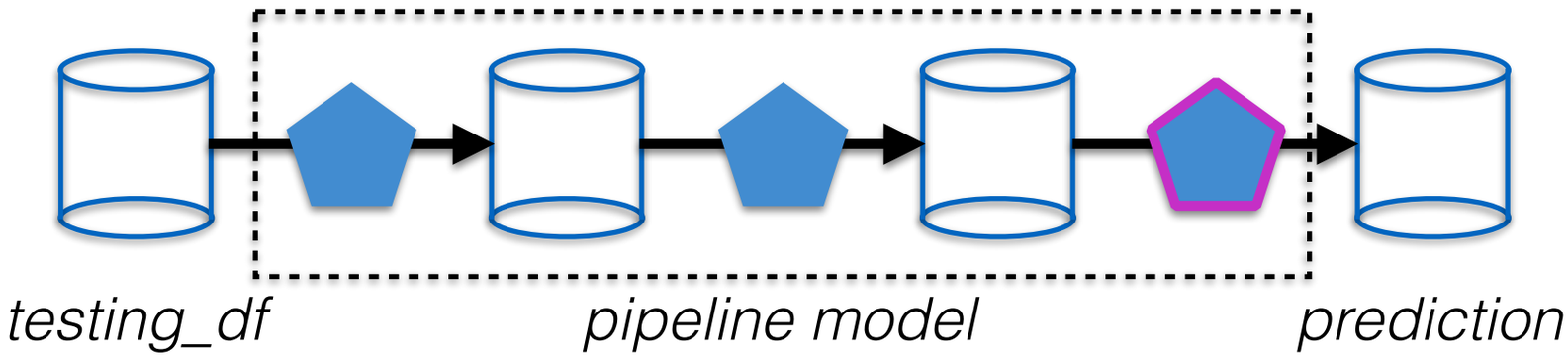


The output of the pipeline fit is a model which is a Transformer 

ML Pipeline Concepts

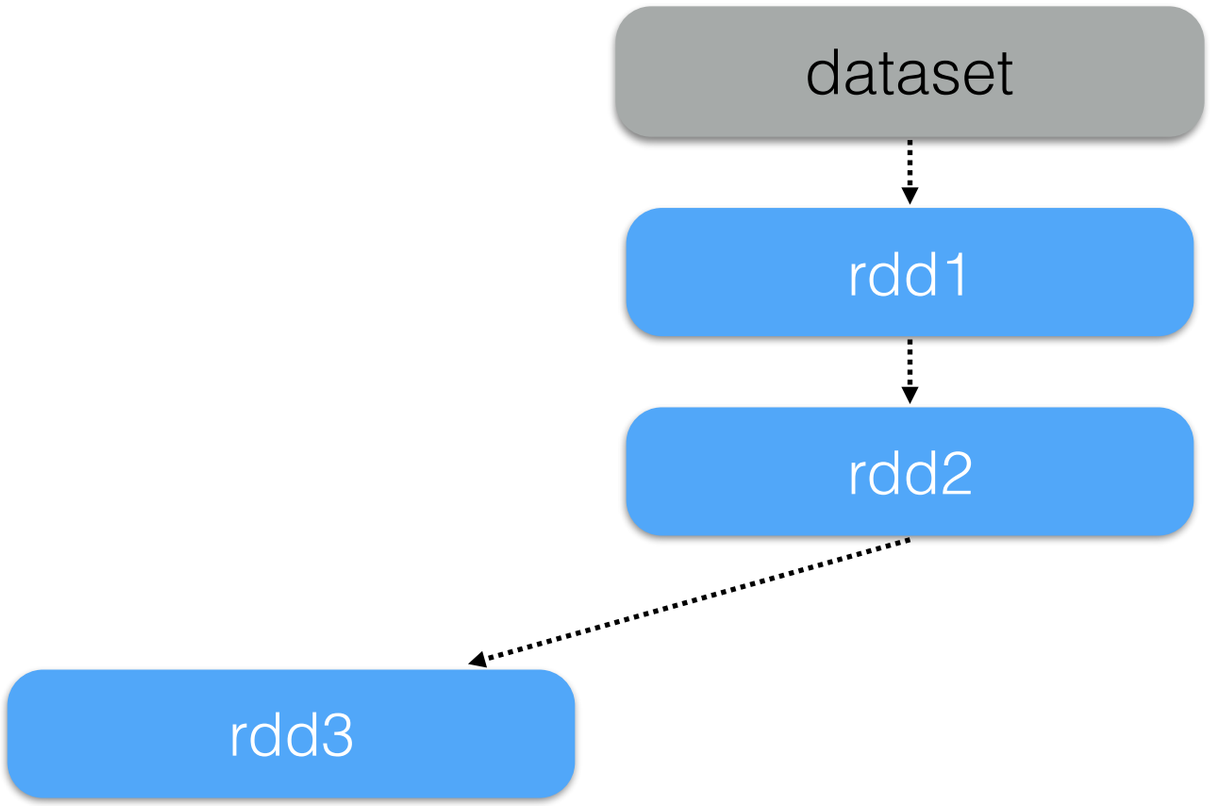
- DataFrame
- Transformer
- Estimator
- **Pipeline**

prediction = model.transform(testing_df)

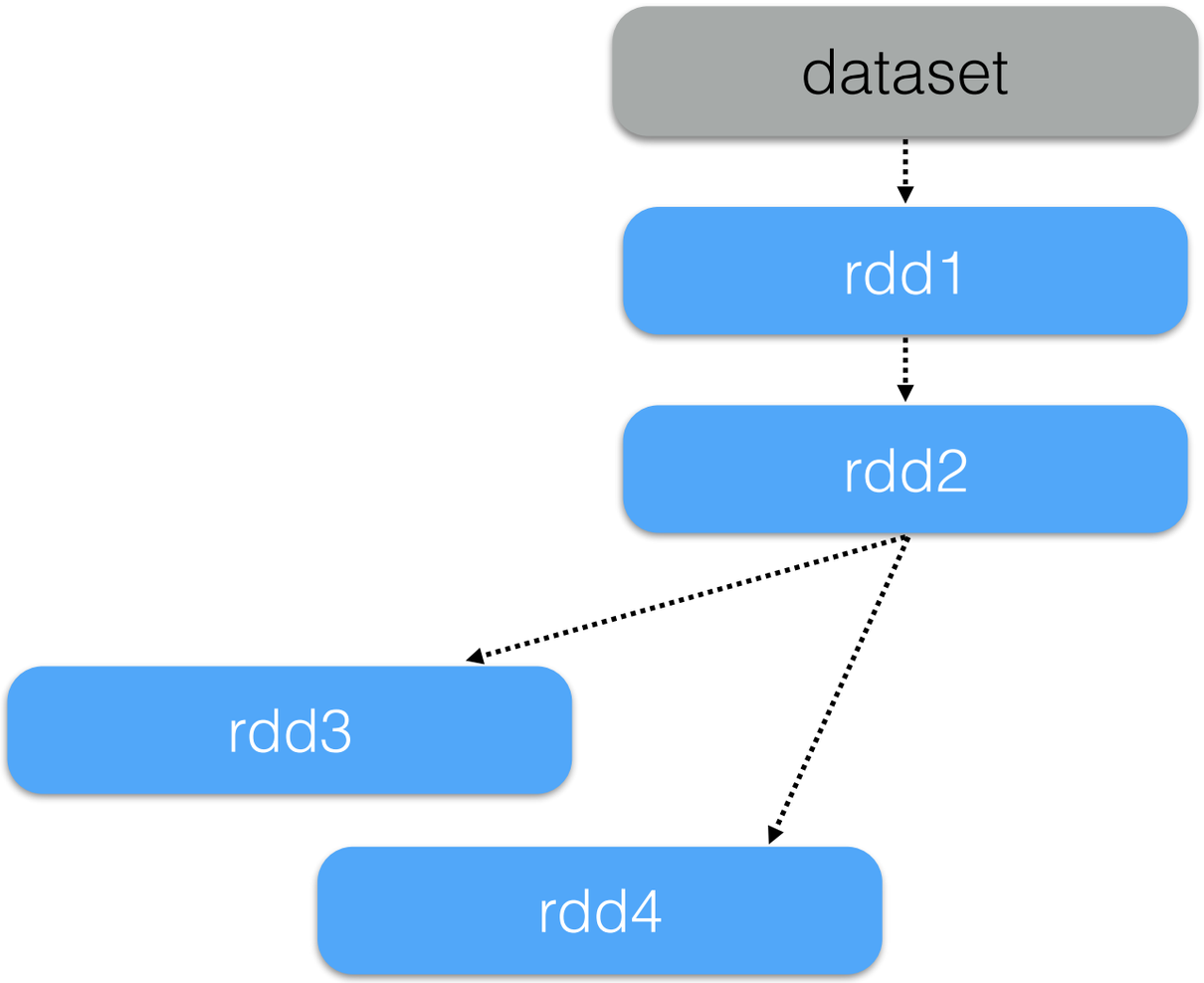


Spark Tuning

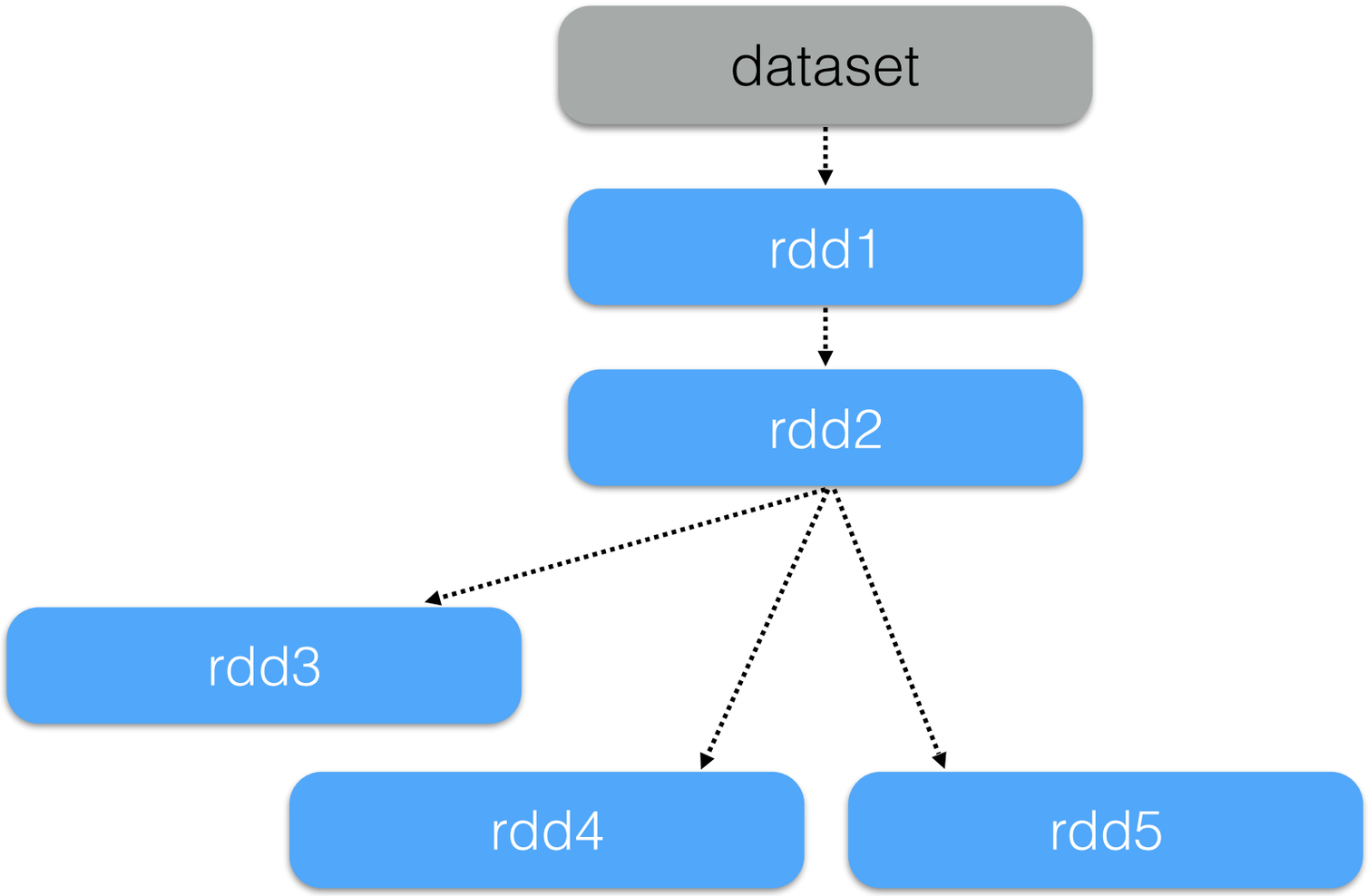
Persistence and Caching



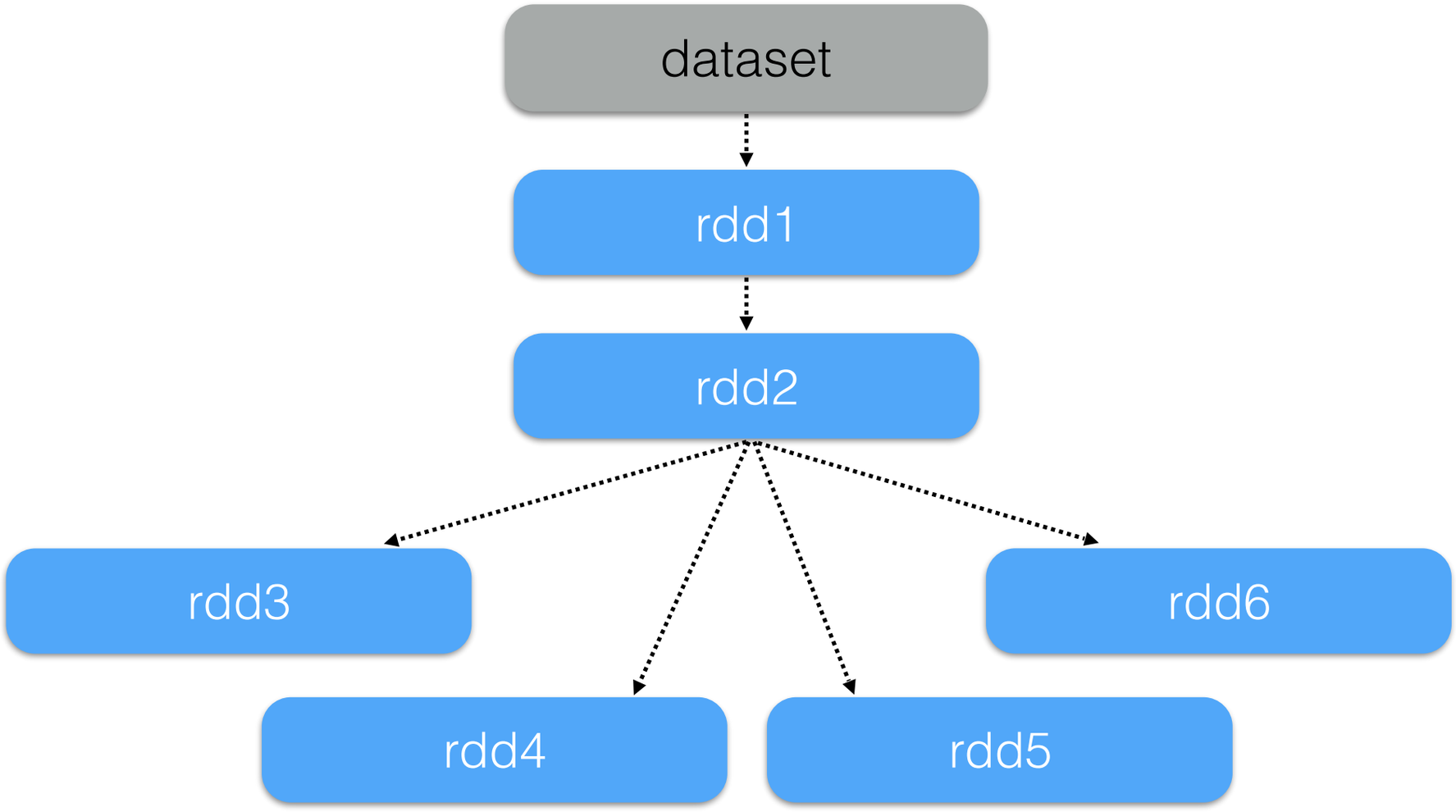
Persistence and Caching



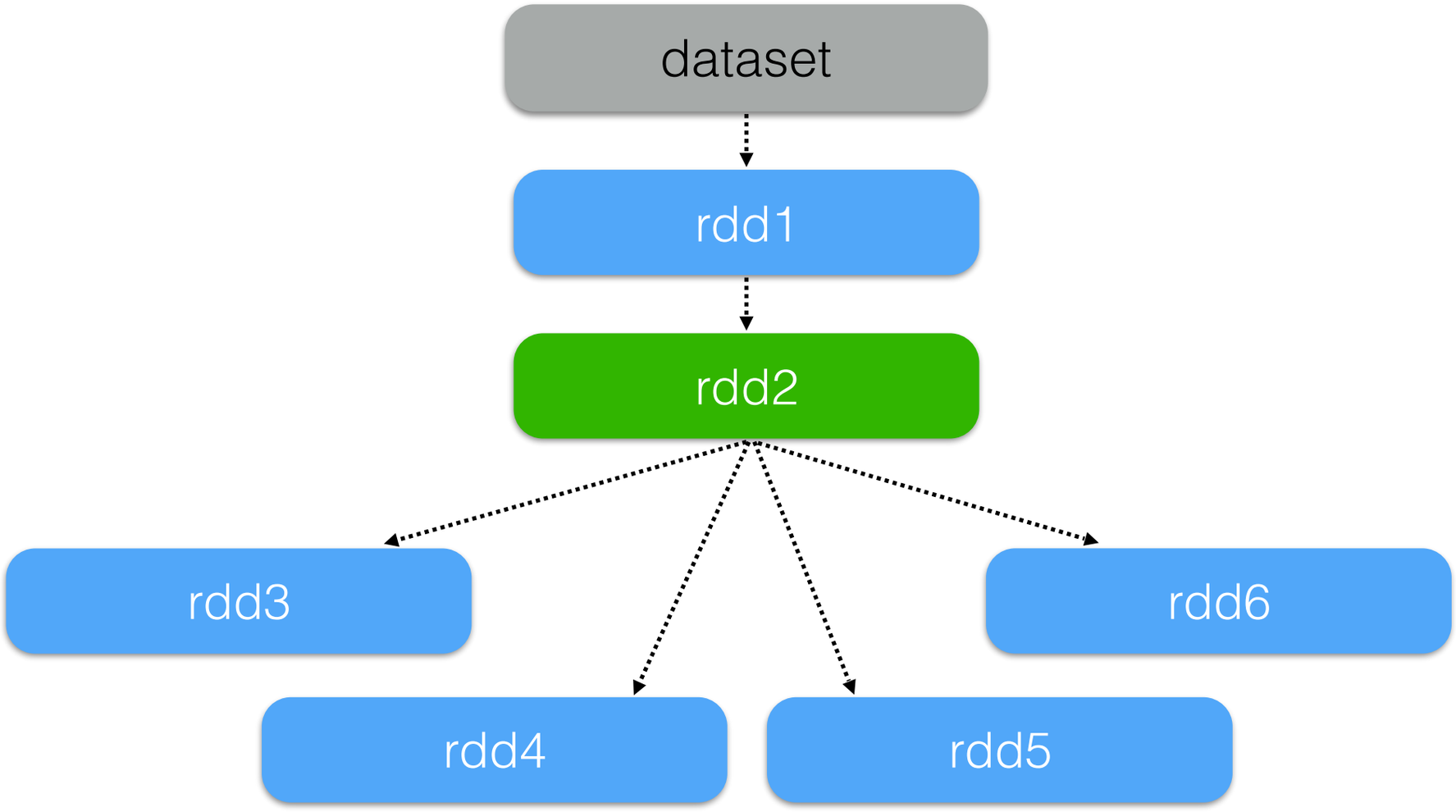
Persistence and Caching



Persistence and Caching



Persistence and Caching



Persistence and Caching

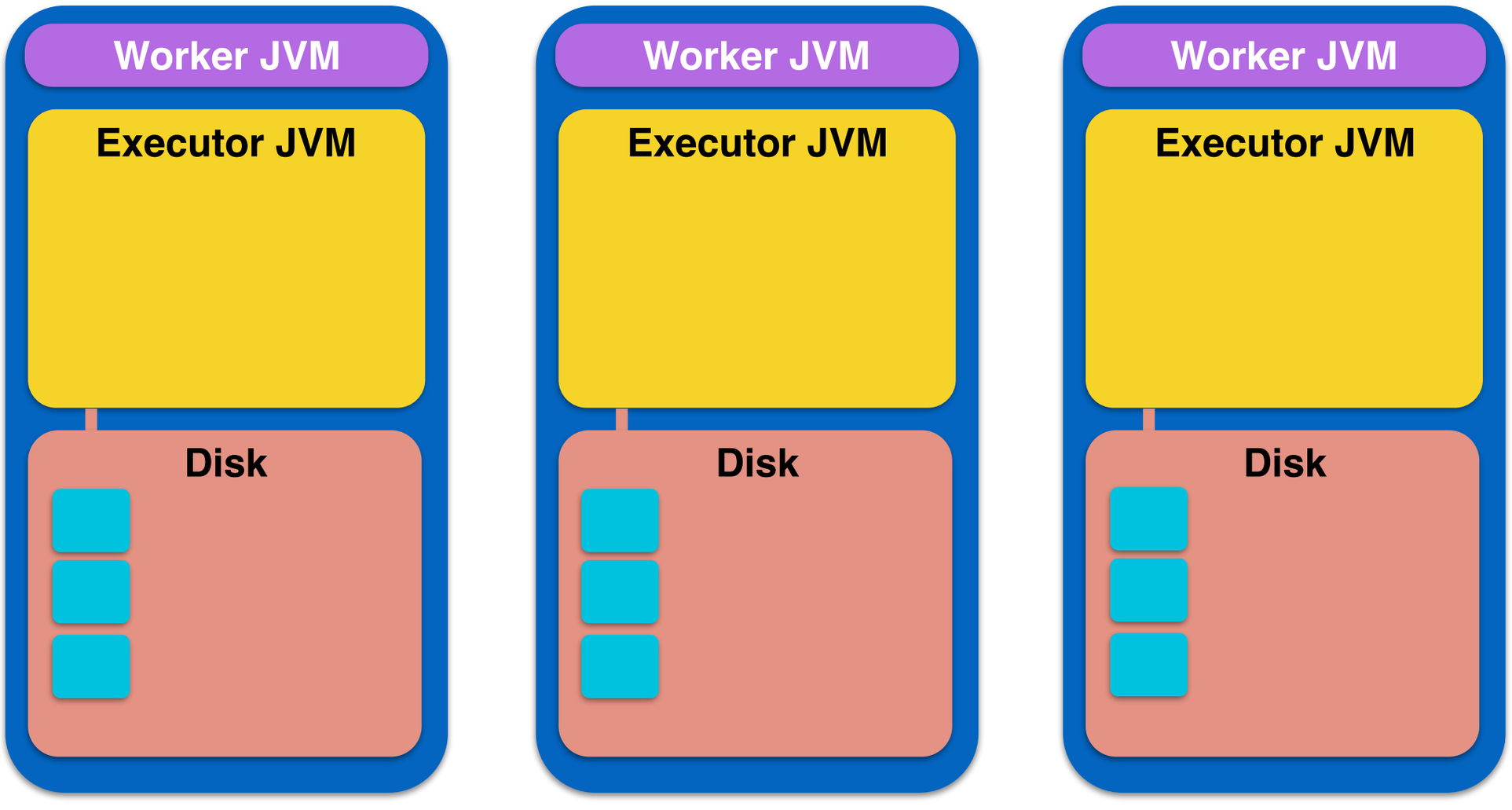
General Options for Persisting

```
       disk only - Always Serialized
memory and disk - Spill to disk when memory is full Ser/De
    memory only - Same as calling .cache() Ser/De
       off heap - Tachyon Storage System
```

Persistence and Caching

General Options for Persisting

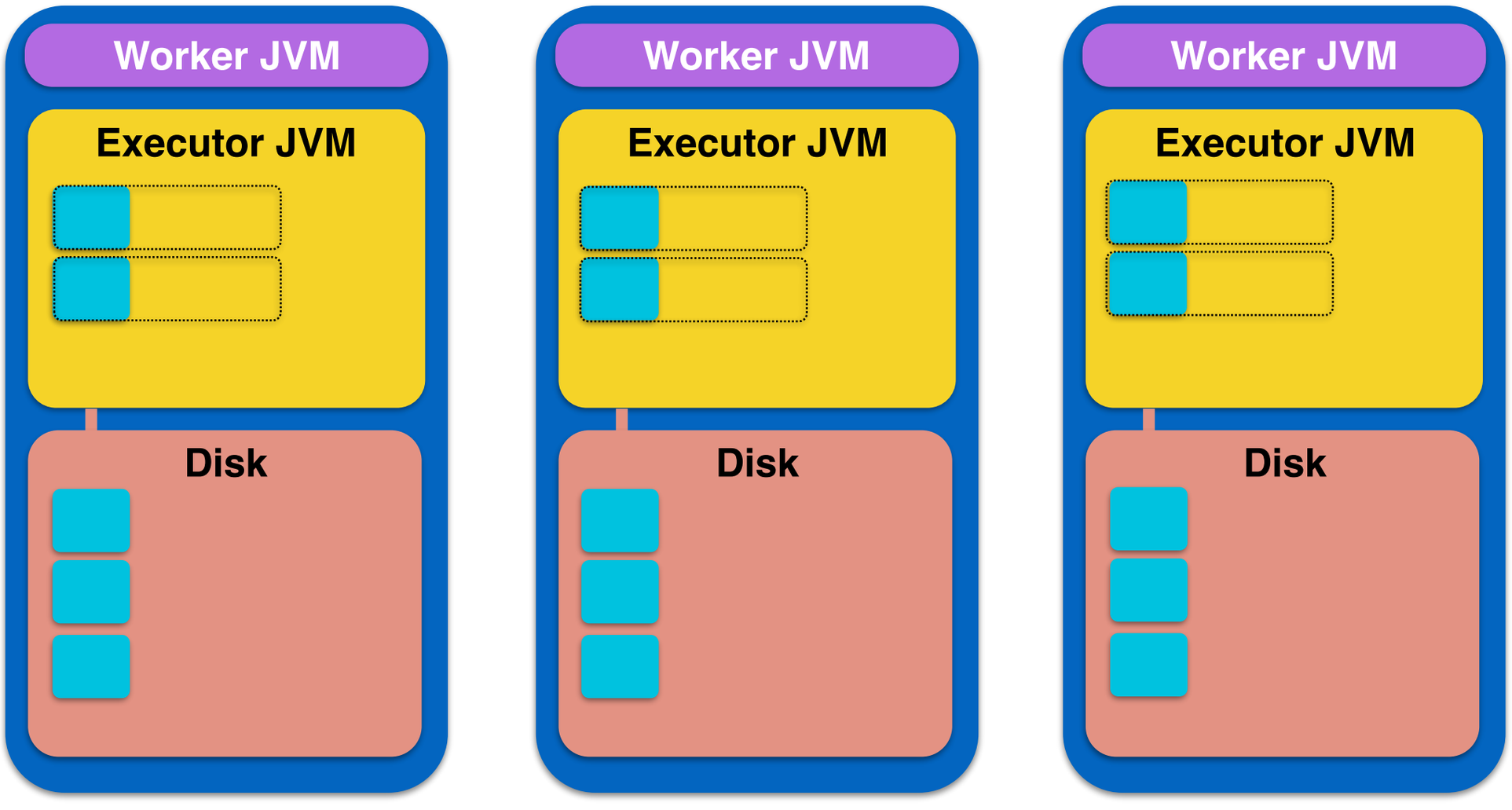
```
disk only - Always Serialized
memory and disk - Spill to disk when memory is full Ser/De
memory only - Same as calling .cache() Ser/De
off heap - Tachyon Storage System
```



Persistence and Caching

General Options for Persisting

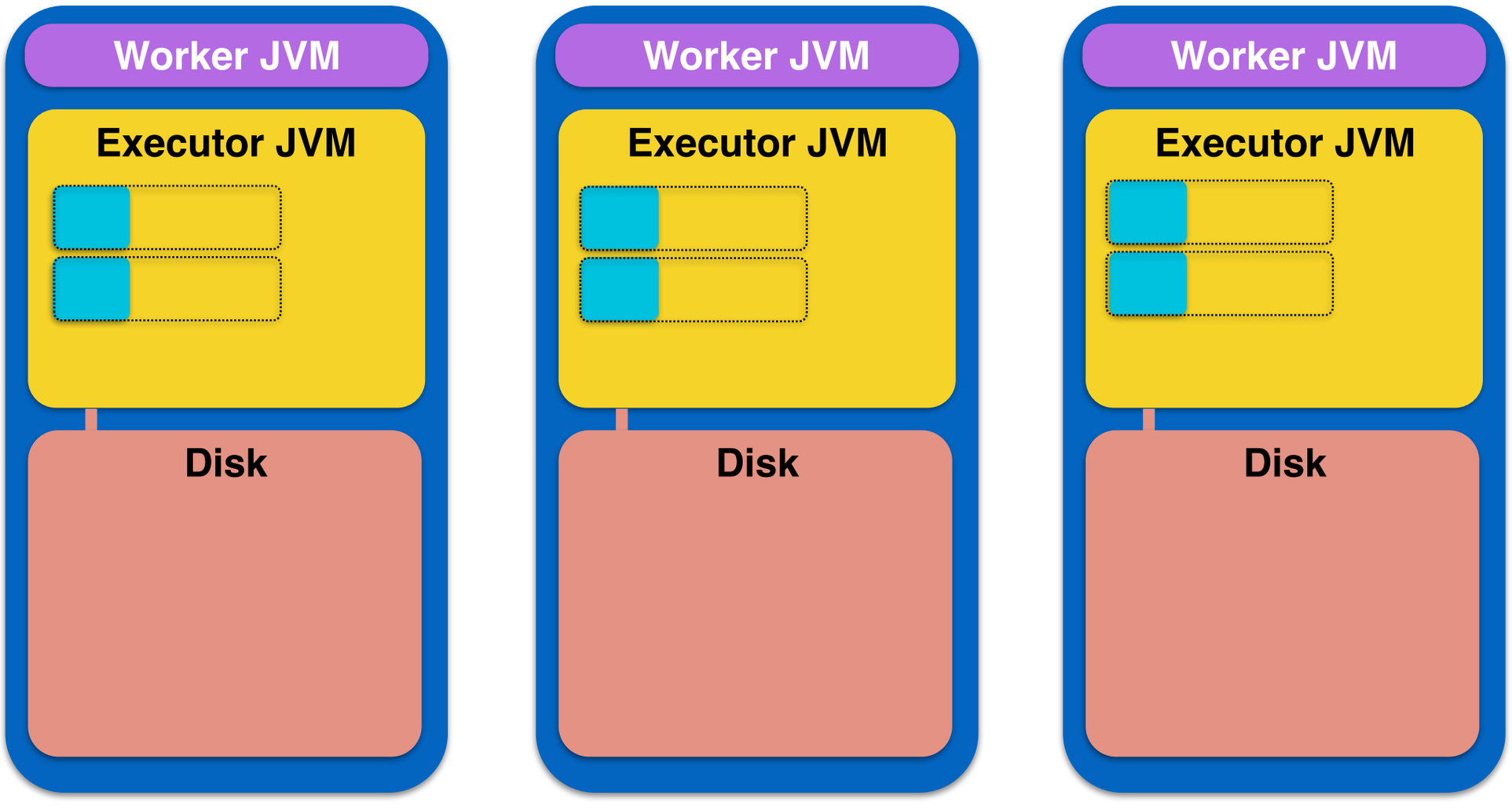
```
disk only - Always Serialized
memory and disk - Spill to disk when memory is full Ser/De
memory only - Same as calling .cache() Ser/De
off heap - Tachyon Storage System
```



Persistence and Caching

General Options for Persisting

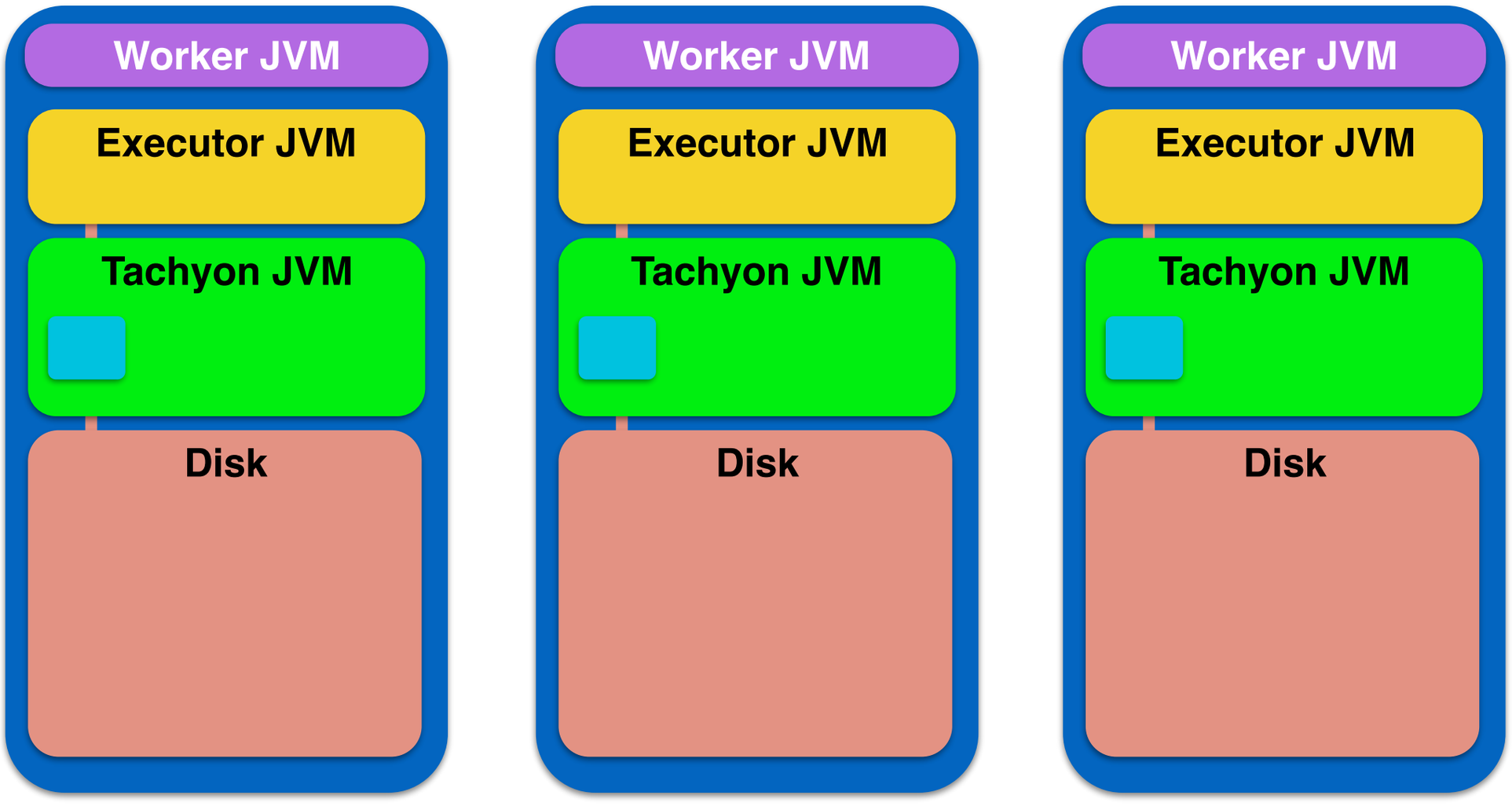
```
disk only - Always Serialized
memory and disk - Spill to disk when memory is full Ser/De
memory only - Same as calling .cache() Ser/De
off heap - Tachyon Storage System
```



Persistence and Caching

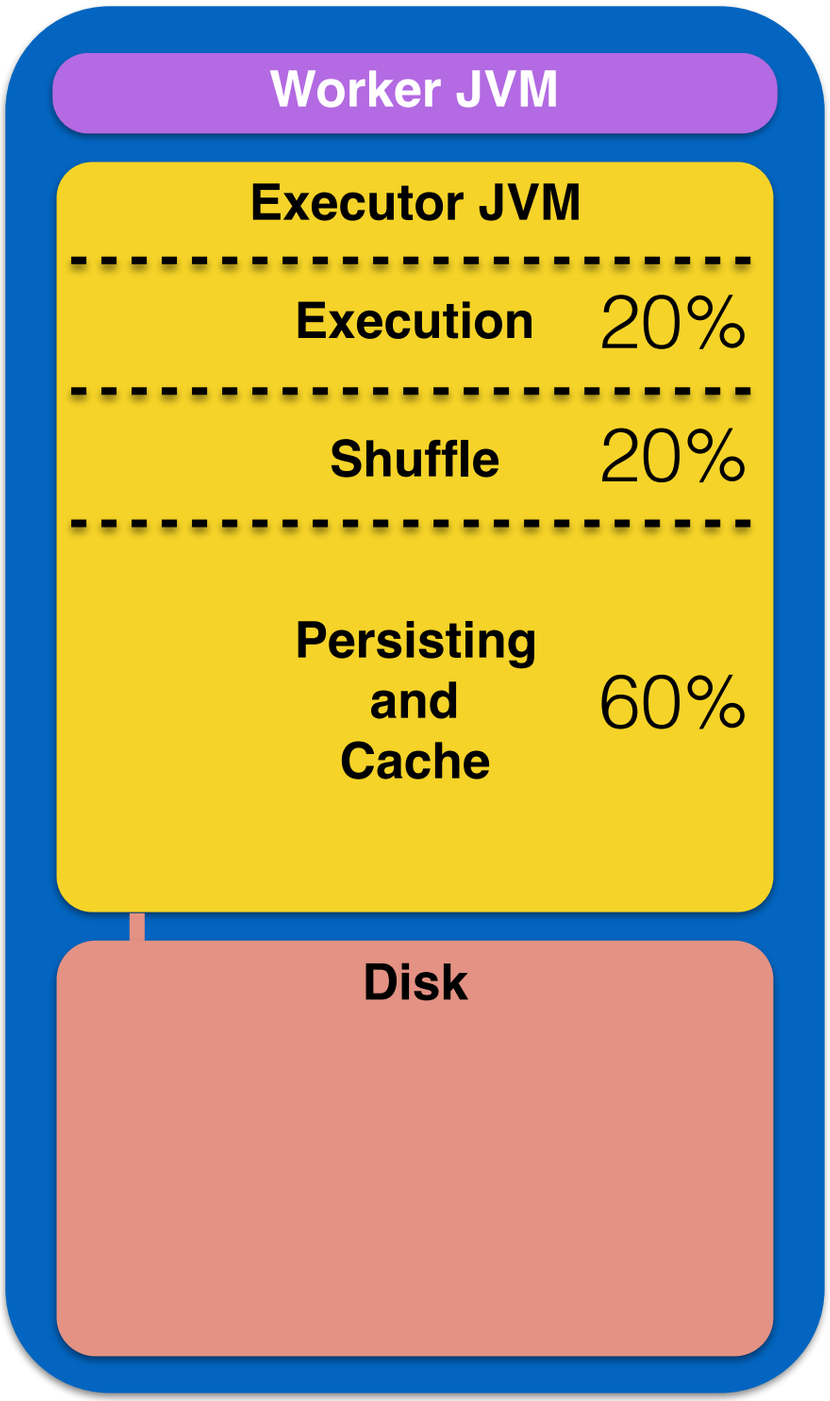
General Options for Persisting

```
disk only - Always Serialized
memory and disk - Spill to disk when memory is full Ser/De
memory only - Same as calling .cache() Ser/De
off heap - Tachyon Storage System
```



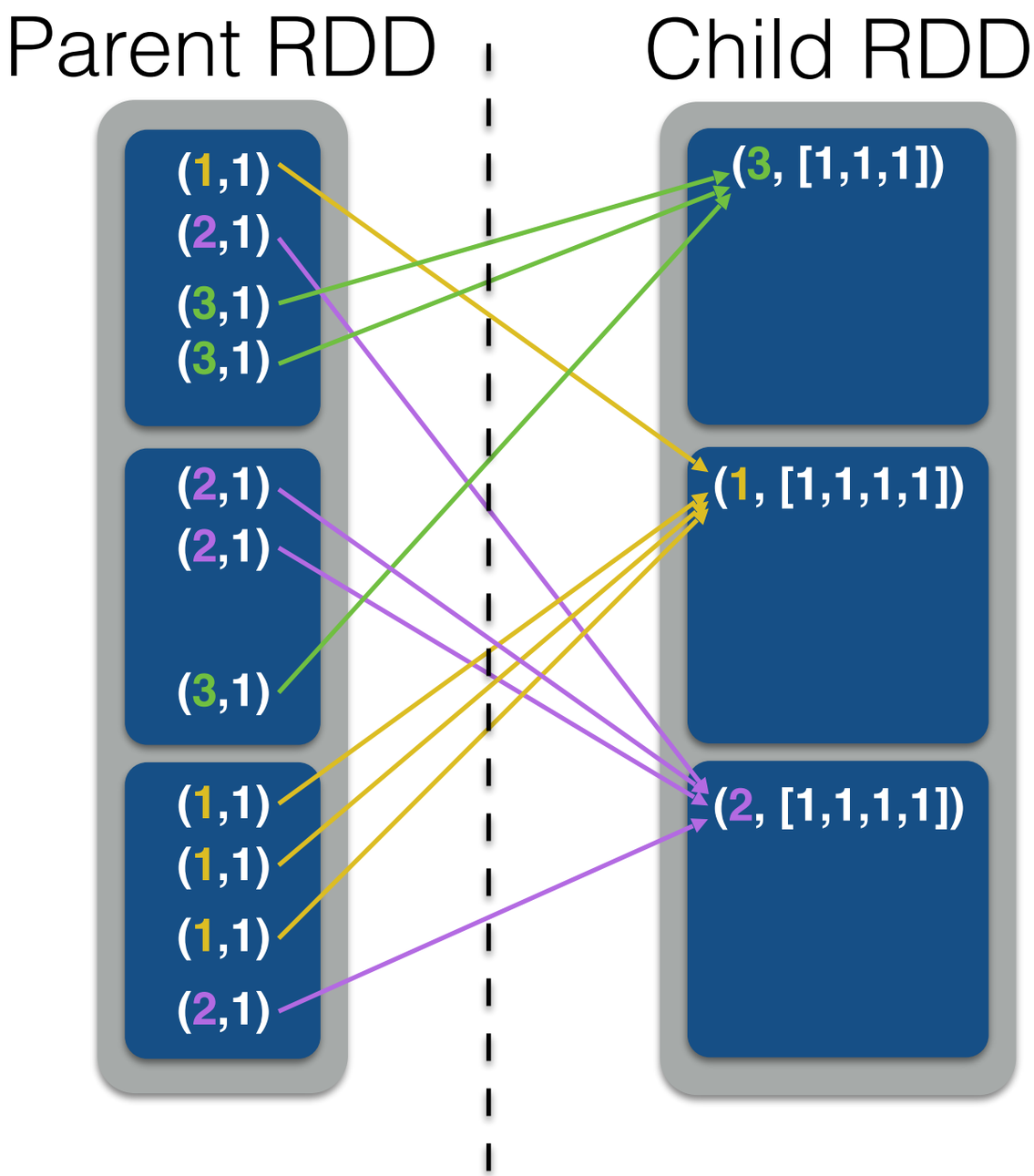
Persistence and Caching

Default Split of the Executor JVM



reduceByKey vs groupByKey

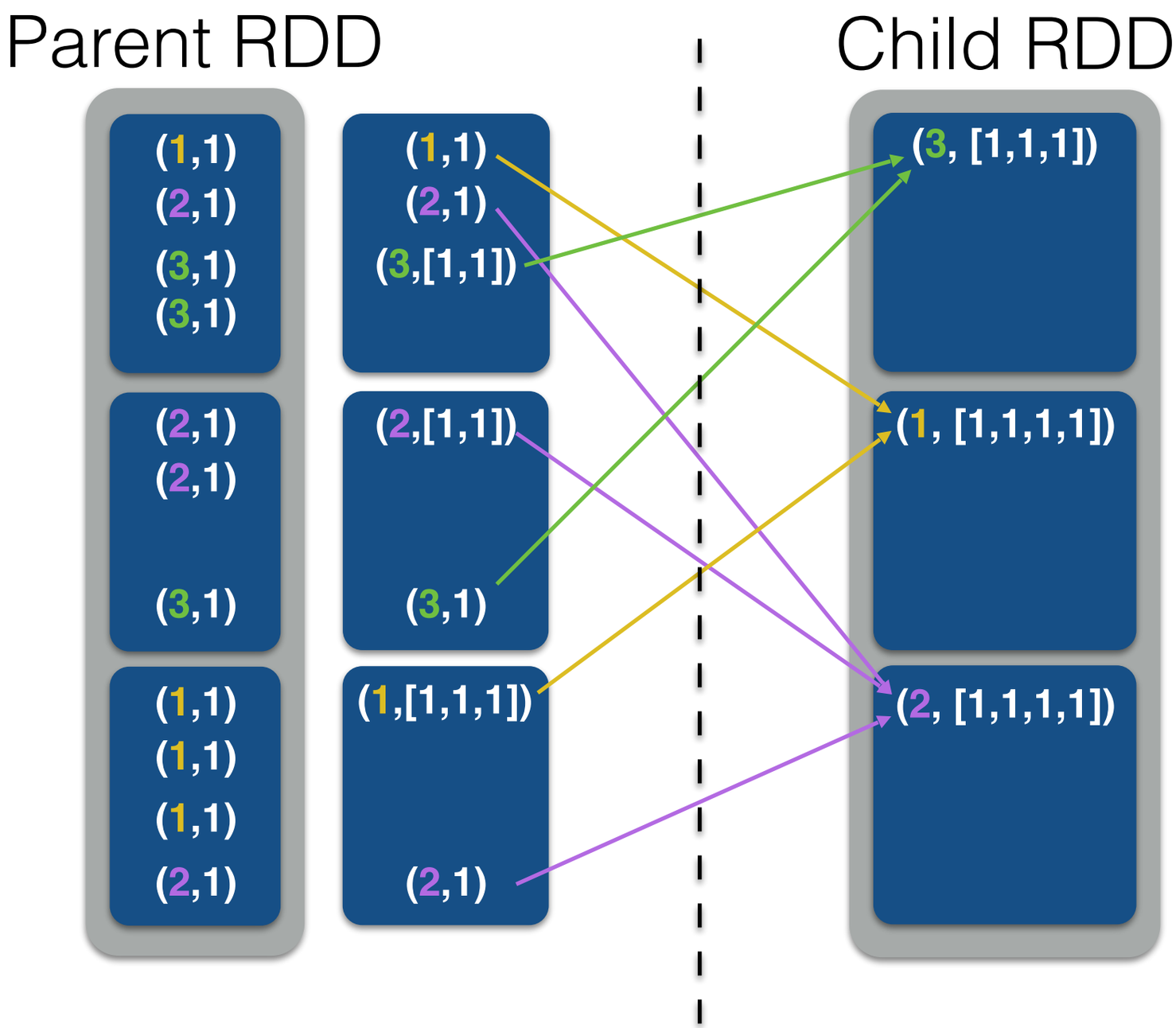
.groupByKey()



11 records shuffled across network

reduceByKey vs groupByKey

.reduceByKey()



7 records shuffled across network