MAPR

An Overview of Apache Spark:

Spark Core, Spark DataFrames, Machine Learning, Streaming

Carol McDonald

Find my presentation and other related resources here:

http://events.mapr.com/JacksonvilleJava

(you can find this link in the event's page at meetup.com)

Today's Presentation

Free On-Demand Training

Whiteboard & demo videos

Free eBooks

Free Hadoop Sandbox

And more…

MAPR®

Agenda

- MapReduce Refresher
- What is Spark?
- The Difference with Spark
- Examples and Resources

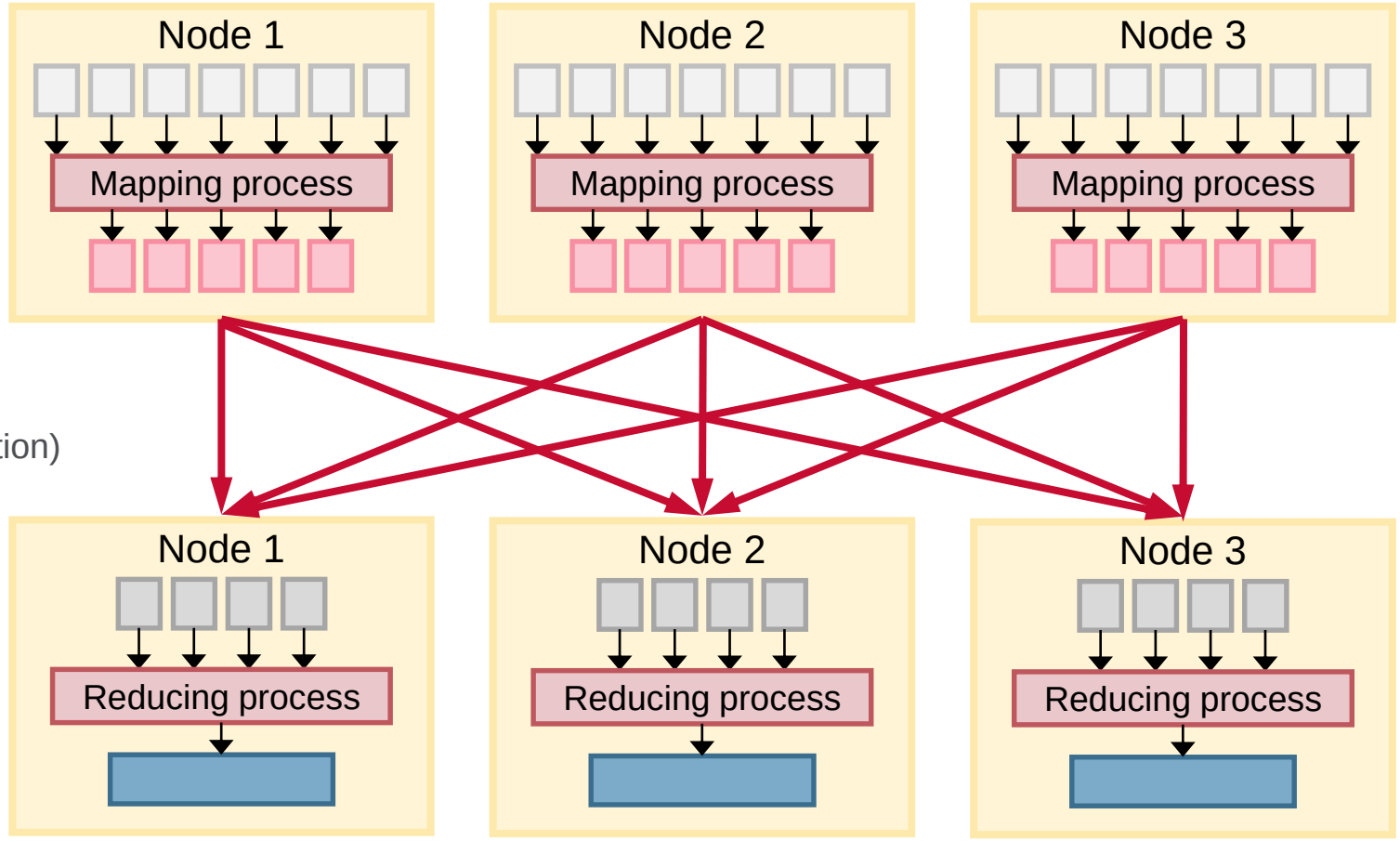
MapReduce Refresher

The Hadoop Strategy

Distribute data
(share nothing)

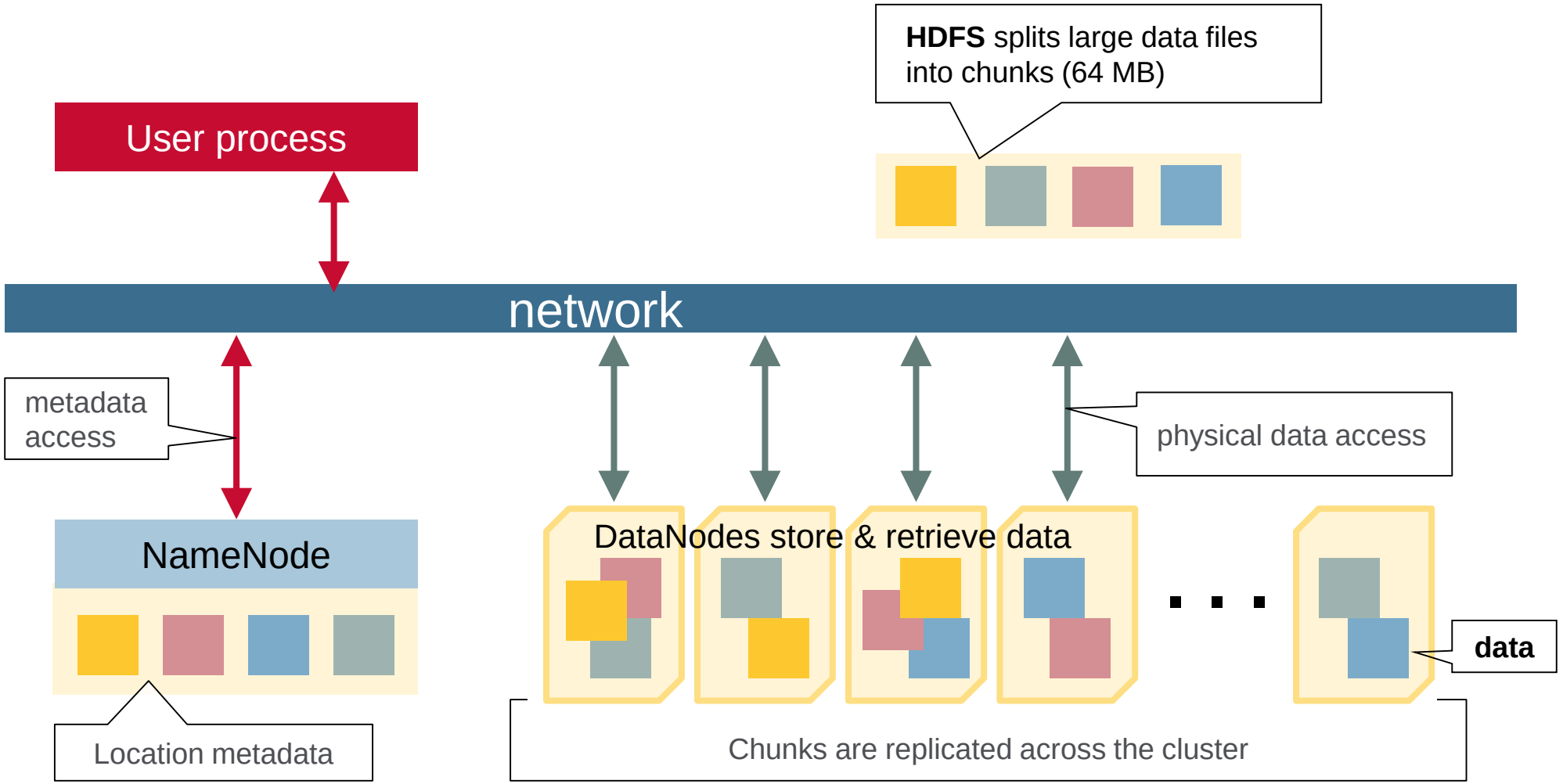
Distribute computation
(parallelization without synchronization)

Tolerate failures
(no single point of failure)

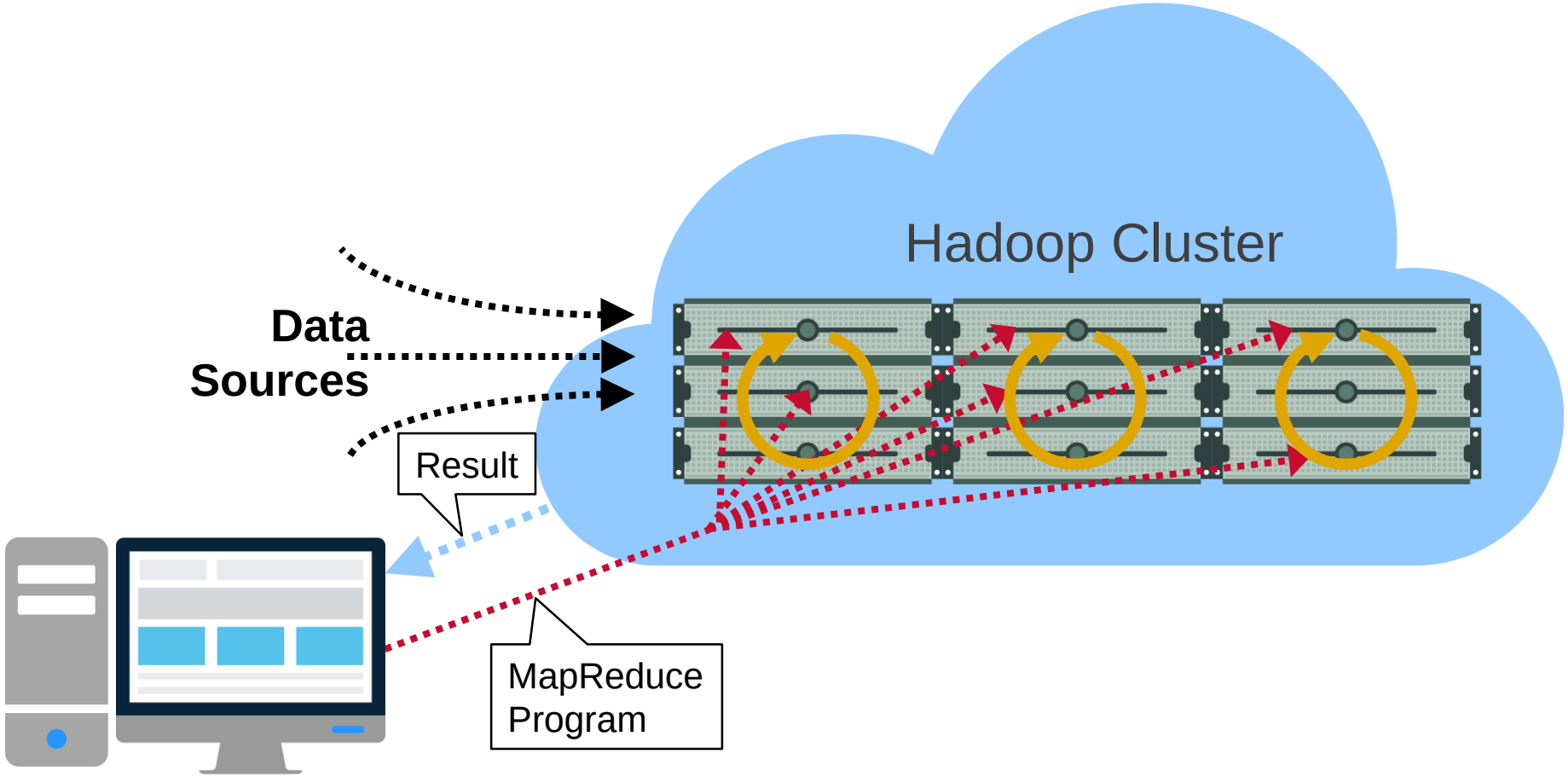


http://developer.yahoo.com/hadoop/tutorial/module4.html

Distribute Data: HDFS

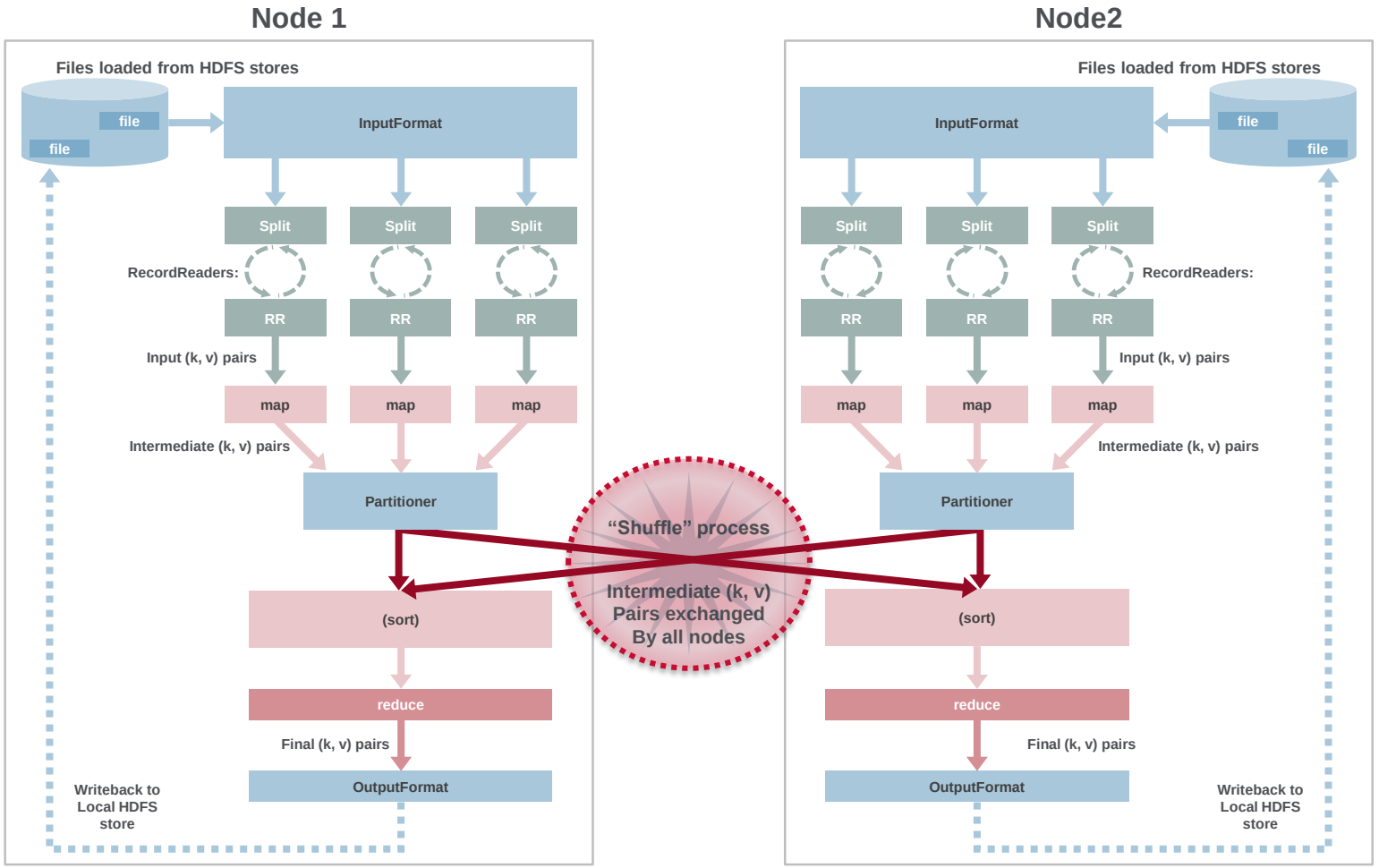


Distribute Computation



MapReduce Execution and Data Flow

- Map
  - Loading data and **defining** a key values
- Shuffle
  - Sorts,collects key-based data
- Reduce
  - Receives **(key, list)**'s to process and output



MapReduce Example: Word Count

"The time has come," the Walrus said,
"To talk of many things:
Of shoes—**and** ships—**and** sealing-wax

the, 1
time, 1
has, 1
come, 1
…
and, 1
…
and, 1
…

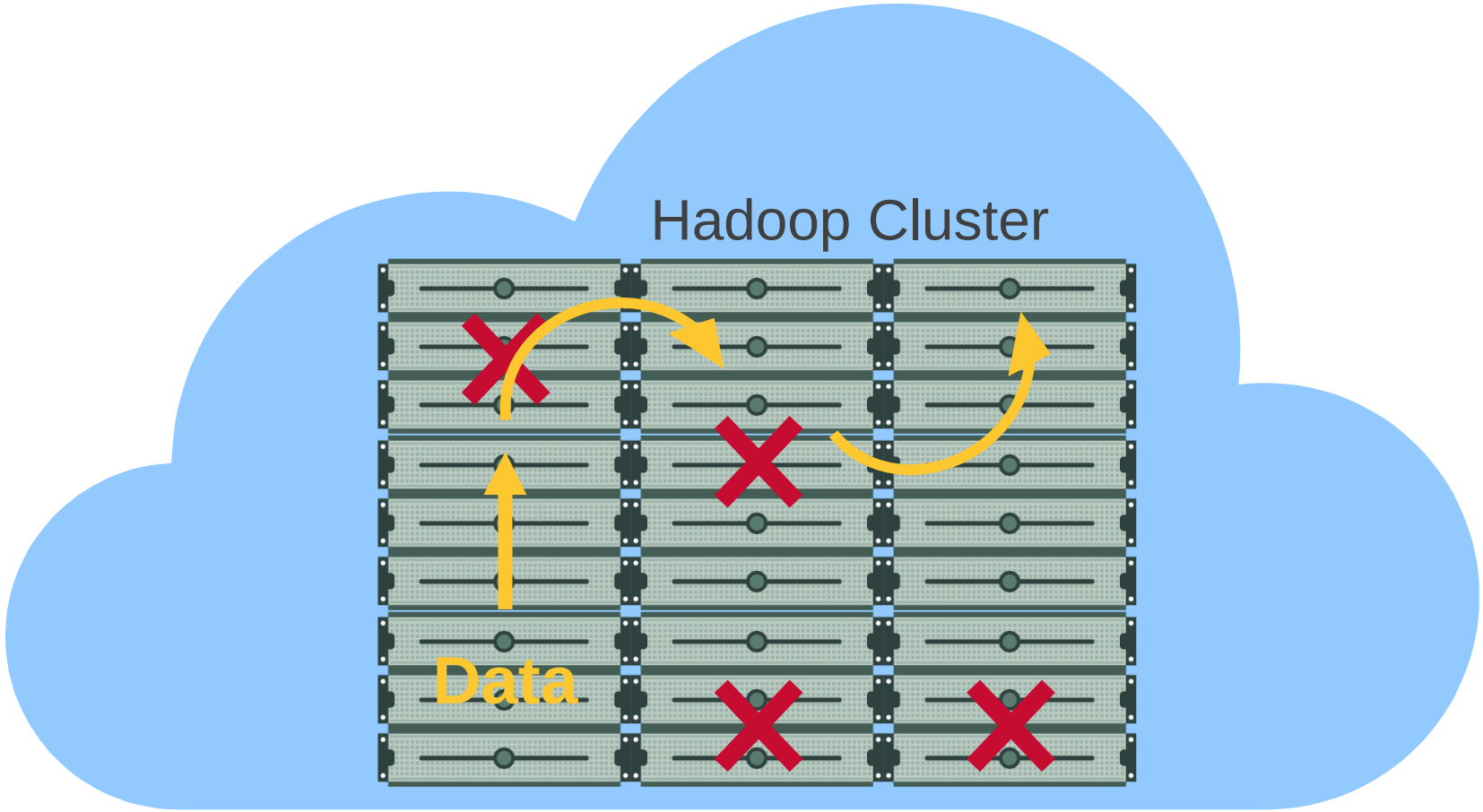
and, [1, **1**, 1]
come, [1,1,1]
has, [1,1]
the, [1,1,1]
time, [1,1,1,1]
…

and, 3
come, 3
has, 2
the, 3
time, 4
…

Input **Map** **Shuffle and Sort** **Reduce** **Output**

Reduce

Tolerate Failures



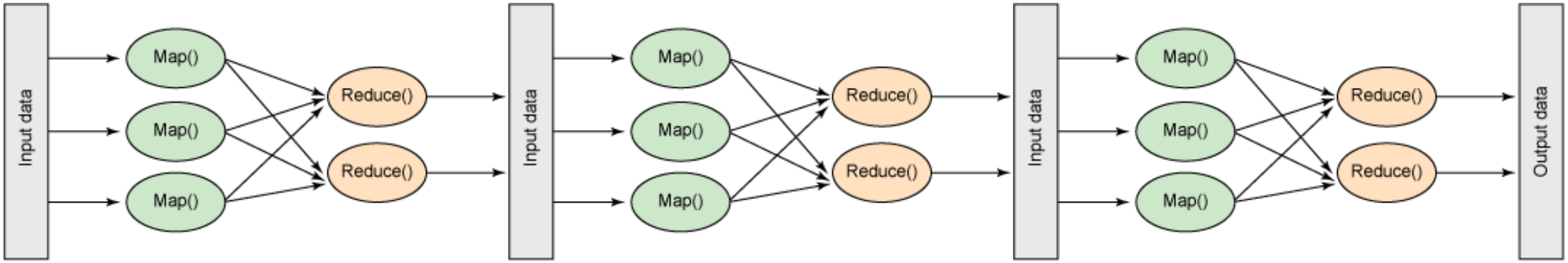
Failures are expected & managed gracefully

DataNode fails -> name node will locate replica

MapReduce task fails -> job tracker will schedule another one

MapReduce Processing Model

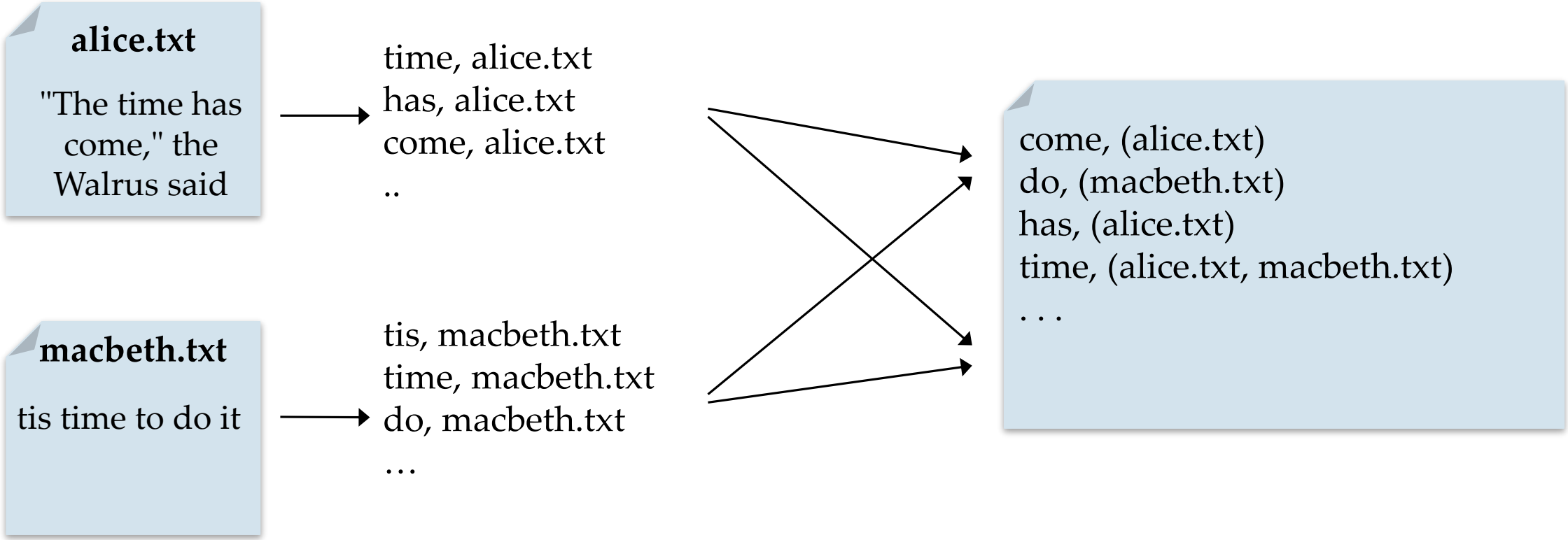
- For complex work, chain jobs together
  - Use a higher level language or DSL that does this for you



MapReduce Design Patterns

- Summarization
  - Inverted index, counting
- Filtering
  - Top ten, distinct
- Aggregation
- Data Organziation
  - partitioning
- Join
  - Join data sets
- Metapattern
  - Job chaining

Inverted Index Example



MapReduce: The Good

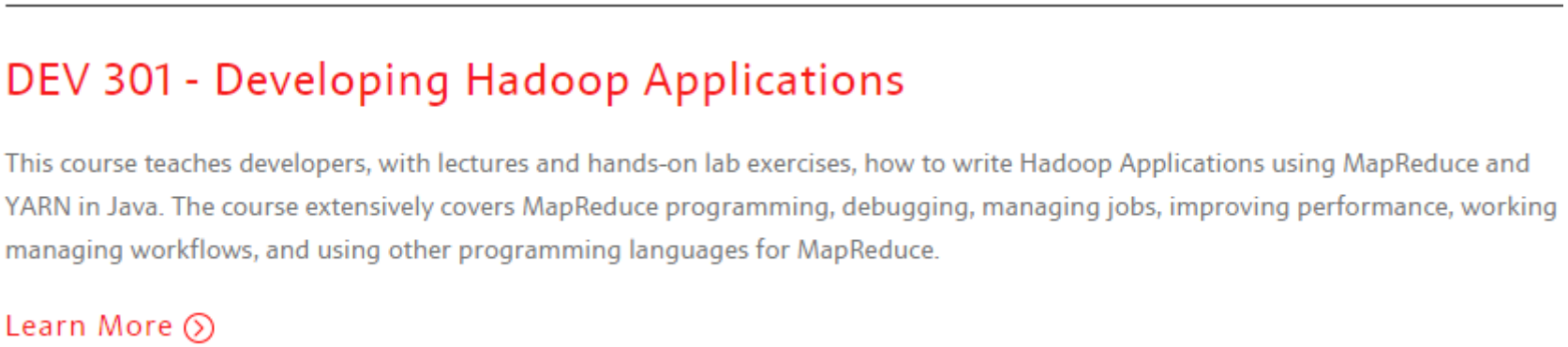
- Built in fault tolerance
- Optimized IO path
- Scalable
- Developer focuses on Map/Reduce, not infrastructure
- simple? API

MapReduce: The Bad

- Optimized for disk IO
  - Doesn’t leverage memory well
  - **Iterative** algorithms go through disk IO path again and again
- Primitive API
  - simple abstraction
  - Key/Value in/out

Free Hadoop MapReduce On Demand Training

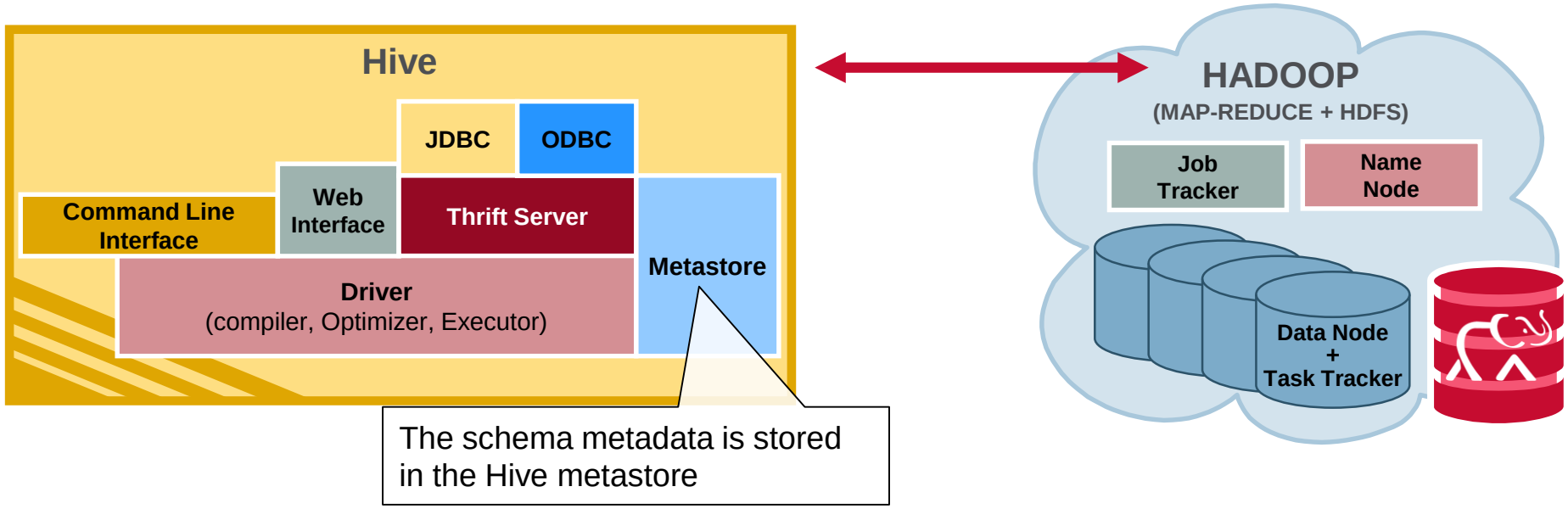
- https://www.mapr.com/services/mapr-academy/big-data-hadoop-online-training



What is Hive?

- Data Warehouse on top of Hadoop
  - analytical querying of data
    - **without programming**
- SQL like execution for Hadoop
  - SQL evaluates to **MapReduce** code
    - Submits jobs to your cluster

Hive Metastore



Hive Table definition

key string	price bigint	vol bigint

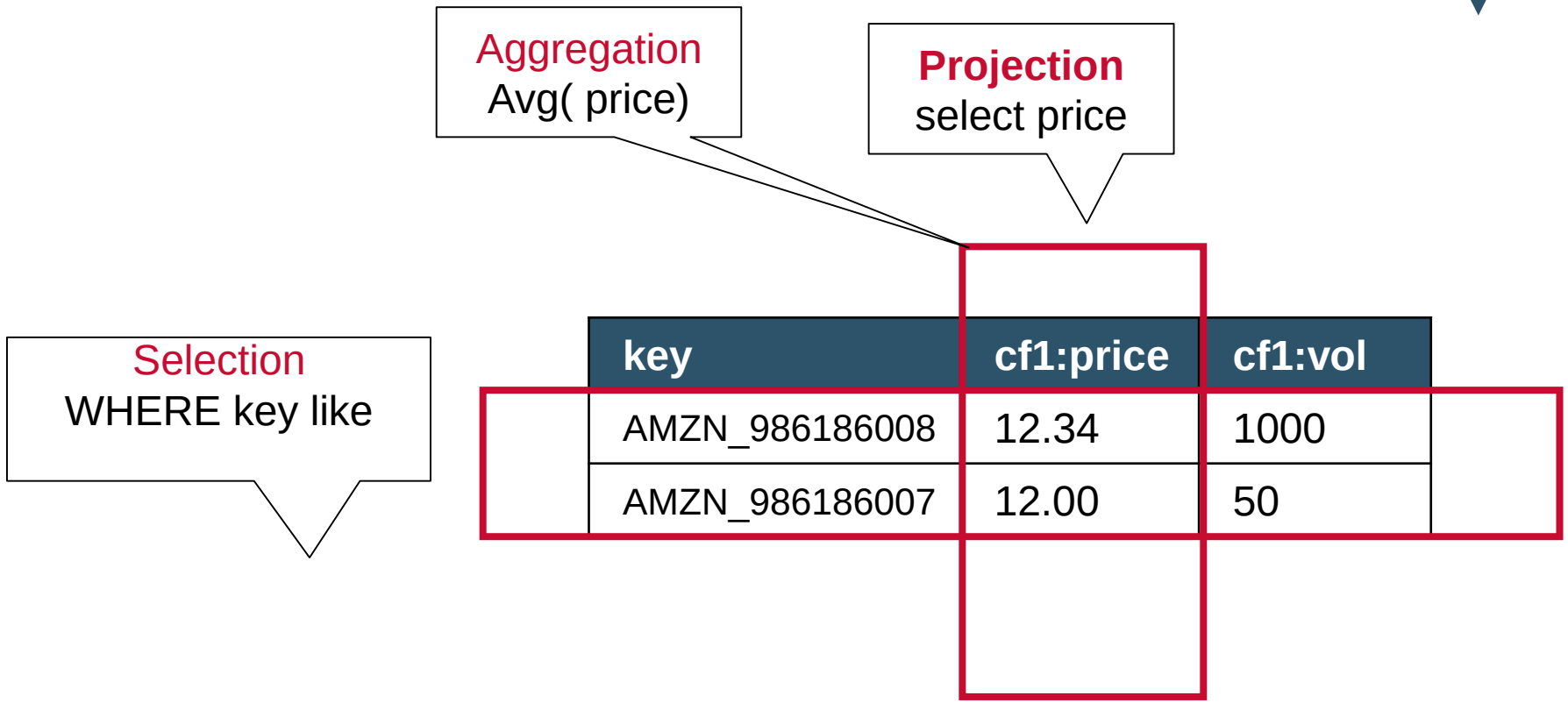
HBase trades_tall Table

key	cf1:price	cf1:vol
AMZN_986186008	12.34	1000
AMZN_986186007	12.00	50

Hive HBase – External Table

SQL evaluates to MapReduce code

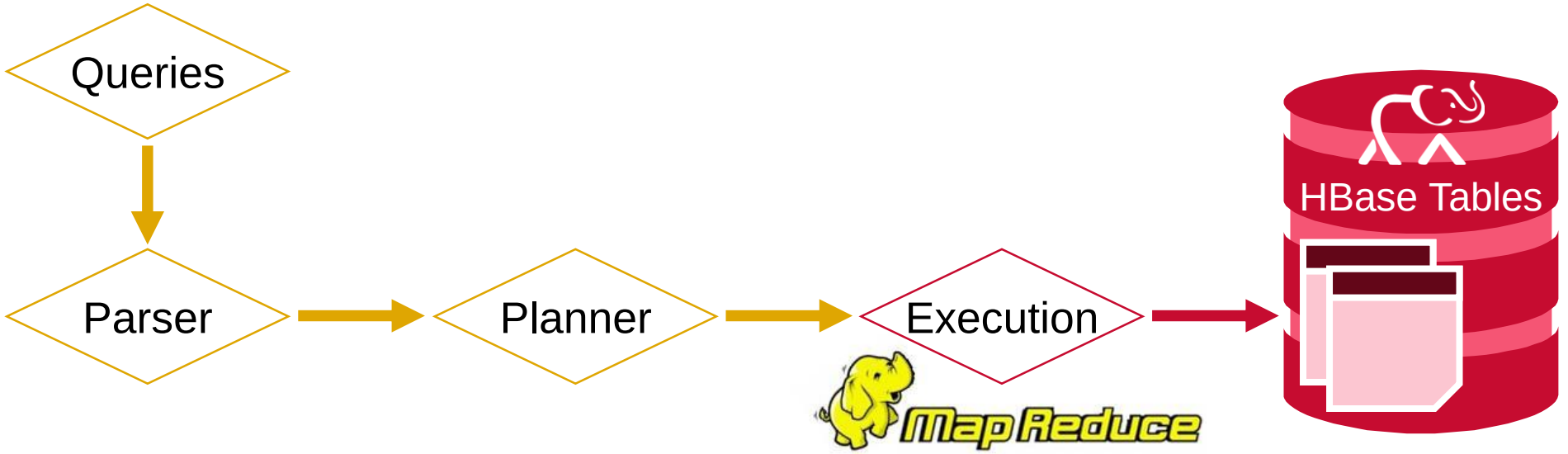
```
SELECT AVG(price) FROM trades WHERE key LIKE "AMZN" ;
```



Hive HBase – Hive Query

SQL evaluates to MapReduce code

```
SELECT AVG(price) FROM trades WHERE key LIKE "GOOG" ;
```

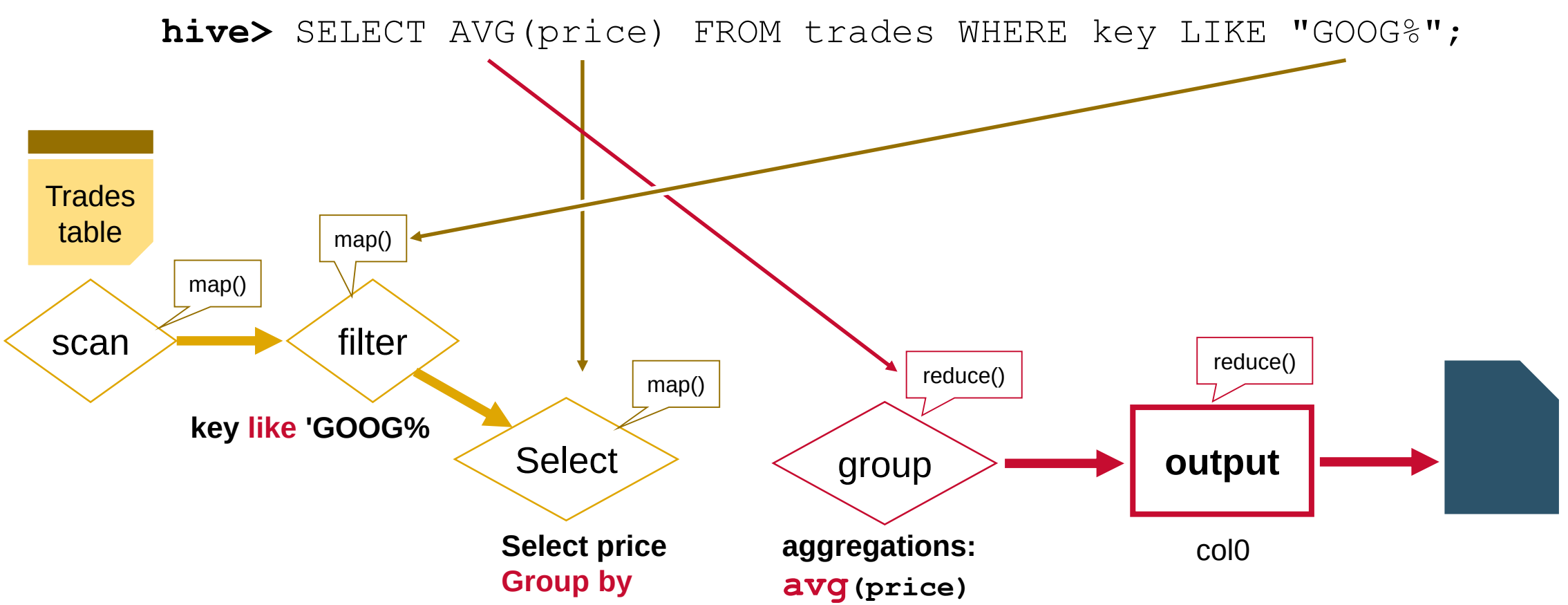


Hive Query Plan

- `EXPLAIN SELECT AVG(price) FROM trades WHERE key LIKE "GOOG%";`

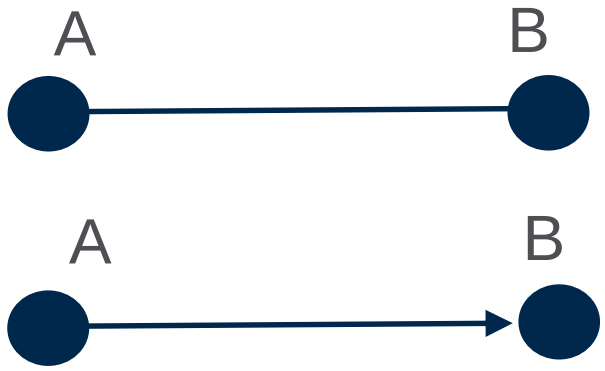
STAGE PLANS:
Stage: Stage-1
Map Reduce
Map Operator Tree:
TableScan
Filter Operator
predicate: (key like 'GOOG%') (type: boolean)
Select Operator
Group By Operator
Reduce Operator Tree:
Group By Operator
Select Operator
File Output Operator

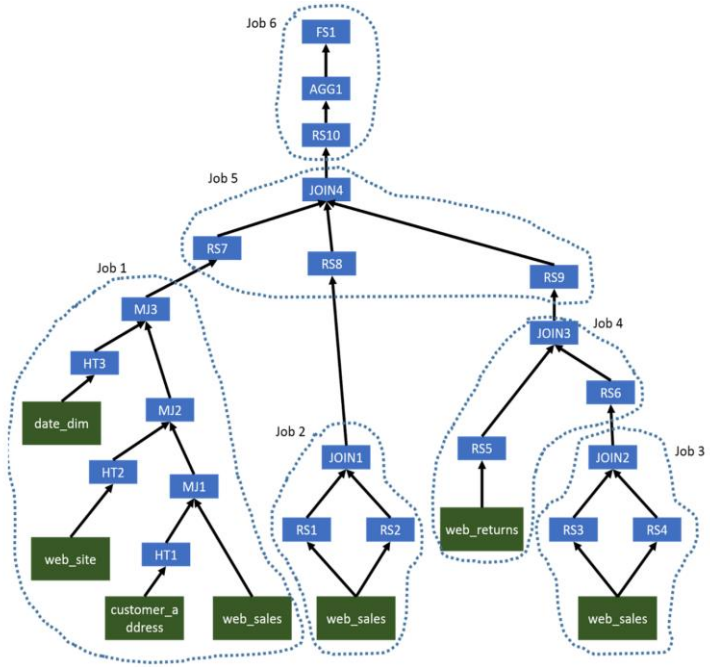
Hive Query Plan – (2)



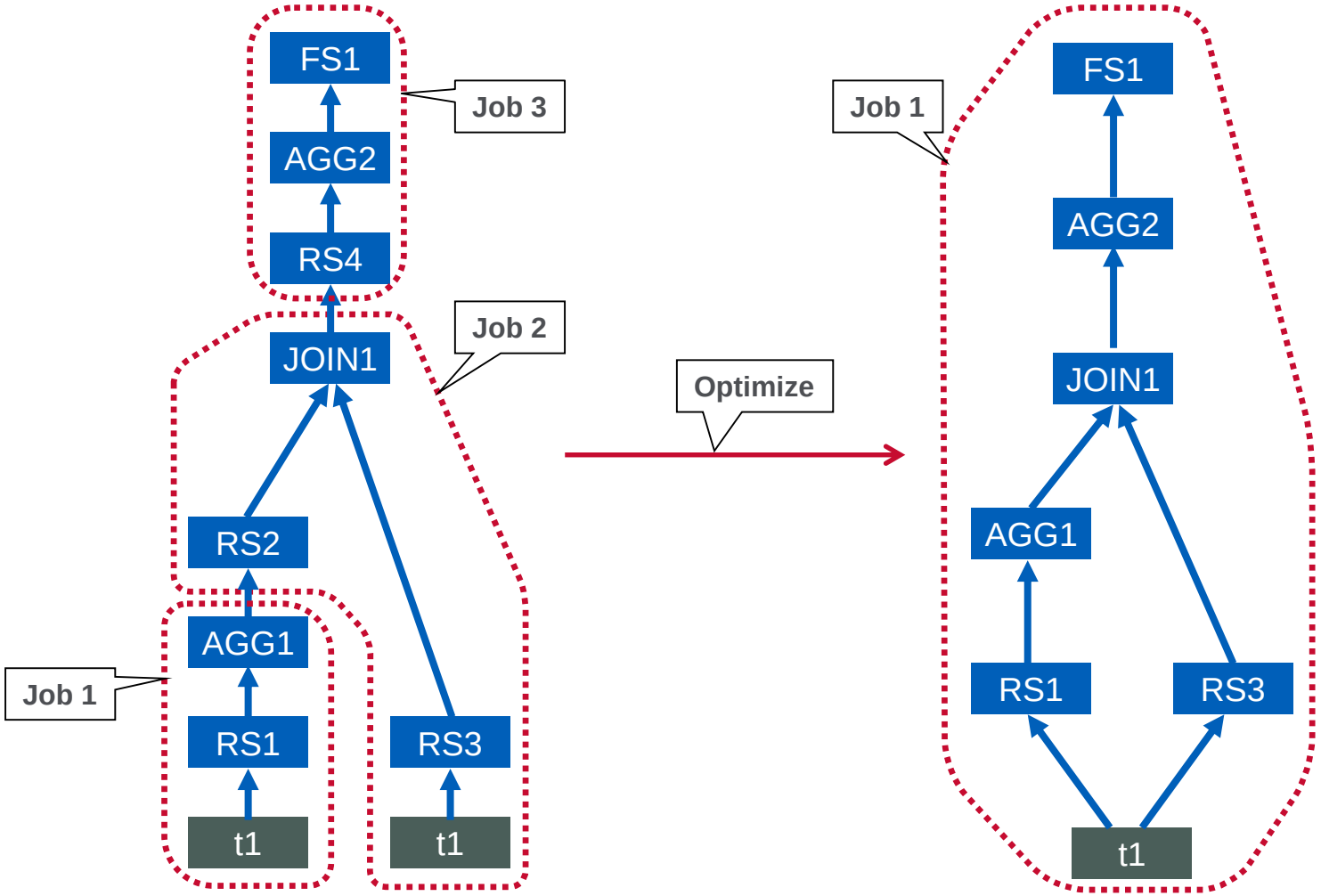
What is a Directed Acylic Graph (DAG) ?

- Graph
  - **vertices** (points) and **edges** (lines)
- Directed
  - Only in a single direction
- Acyclic
  - No looping



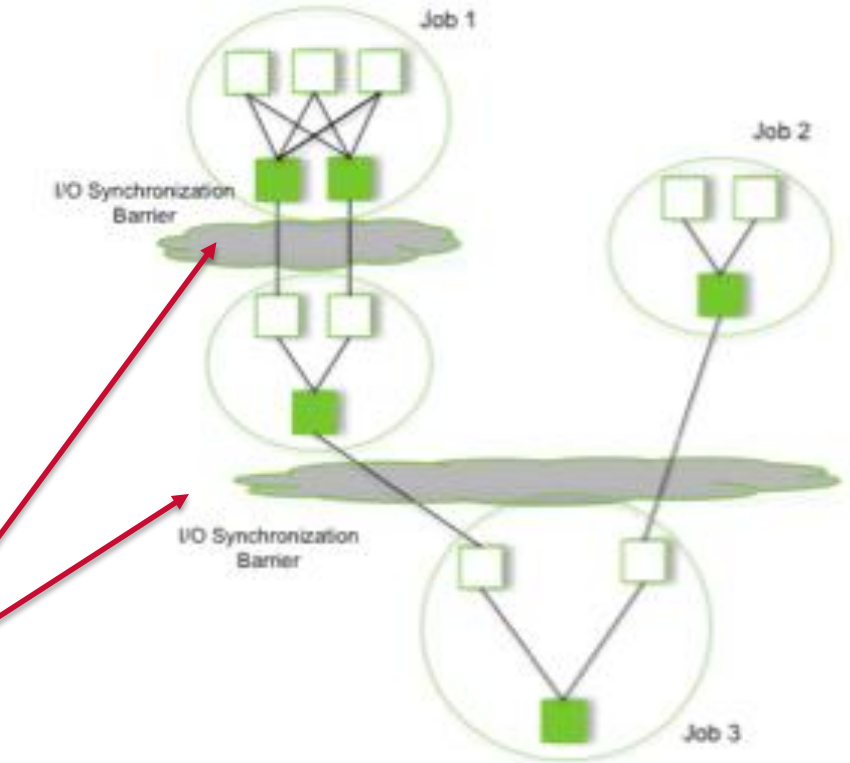


Hive Query Plan Map Reduce Execution

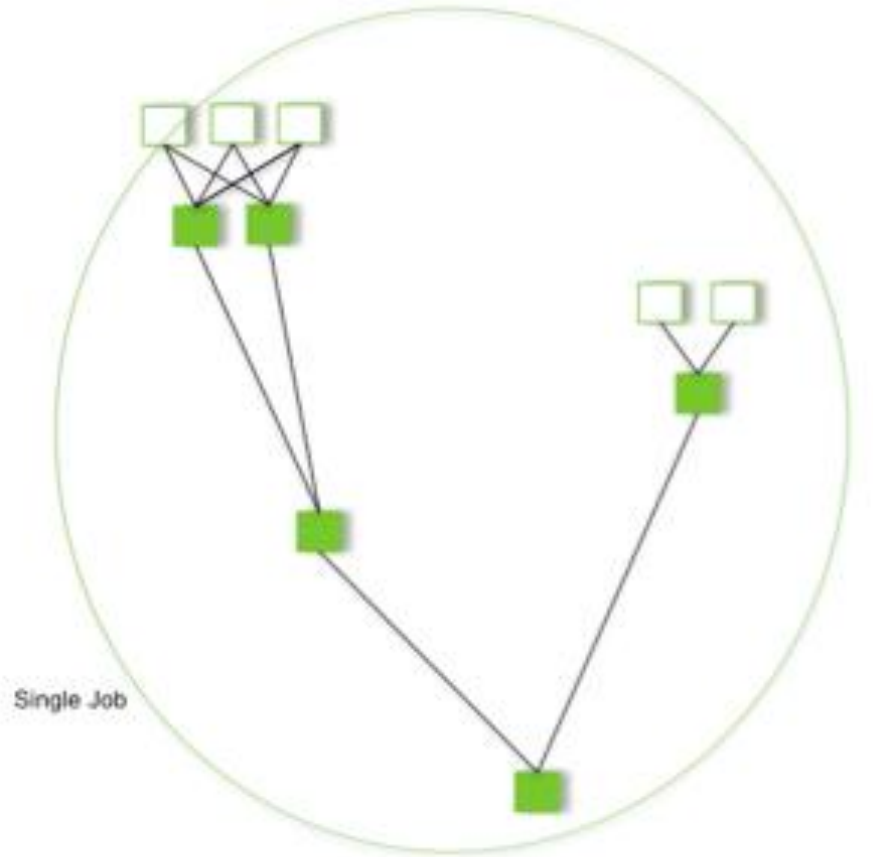


Iteration is slow because it writes/reads data to disk

Slow !



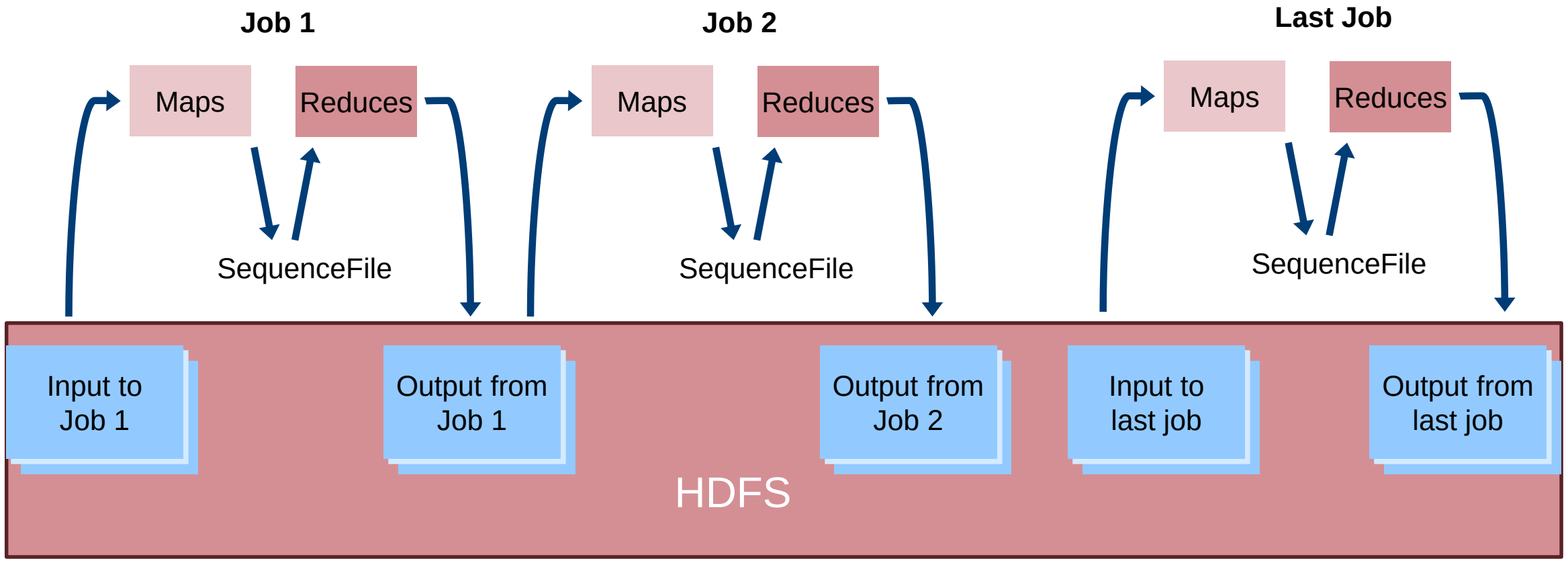
Pig/Hive - MR



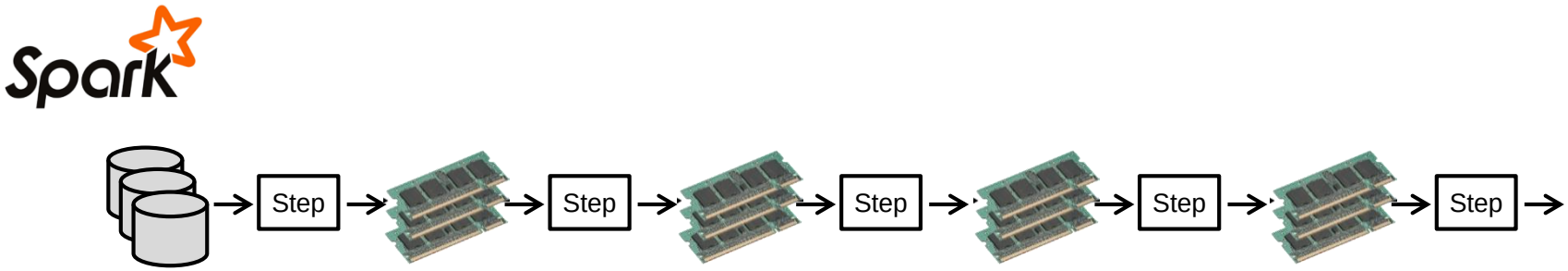
Pig/Hive - Tez

Typical MapReduce Workflows

Iteration is slow because it writes/reads data to disk



Iterations



In-memory Caching

- Data Partitions read from RAM instead of disk

Free HBase On Demand Training

(includes Hive and MapReduce with HBase)

- https://www.mapr.com/services/mapr-academy/big-data-hadoop-online-training

DEV 320 - HBase Data Model and Architecture

This course is intended for data analysts, data architects and application developers. DEV 320 provides you with a tho
understanding of the HBase data model and architecture, which is required before going on to designing HBase sche
applications.

Learn More

DEV 325 - HBase Schema Design

Targeted towards data analysts, data architects and application developers, the goal of this course is to enable you to
schemas based on design guidelines. You will learn about the various elements of schema design and how to design
offers an in-depth look at designing row keys, avoiding hot-spotting and designing column families. It discusses how
to an HBase model. You will learn the differences between tall tables and wide tables. Concepts are conveyed throug
of scenarios.

Learn More

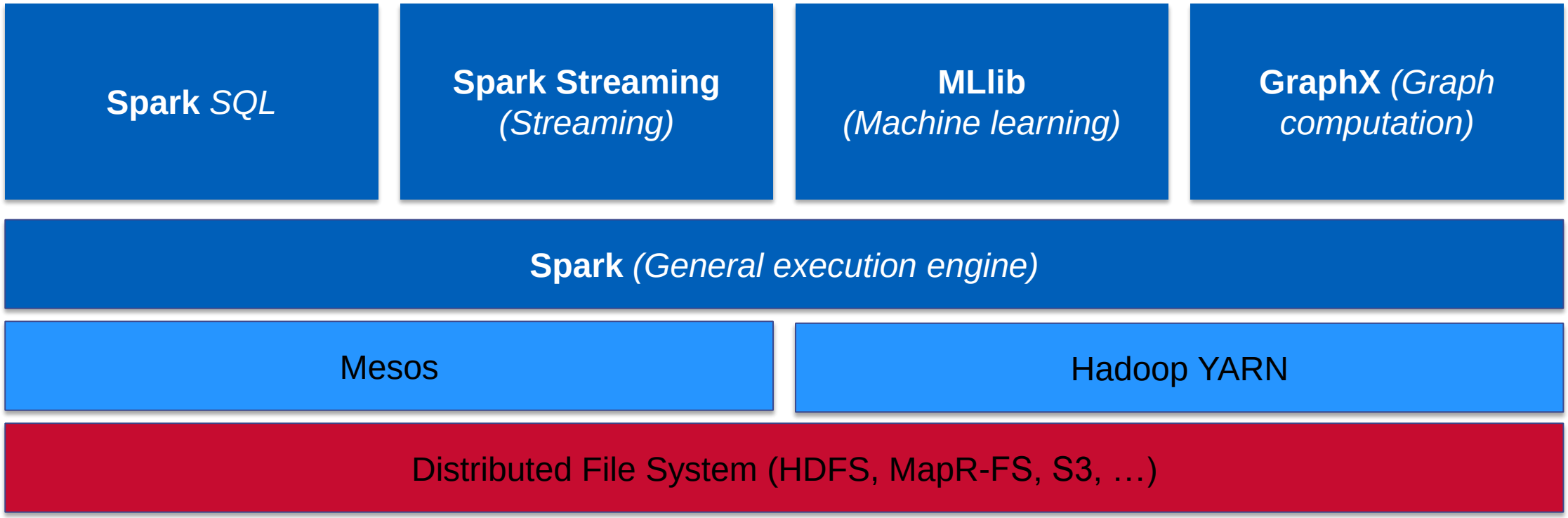
DEV 330 - Developing HBase Applications: Basics

NEW!

Targeted towards data architects and application developers who have experience with Java, the goal of this course i
programs using Hadoop as a distributed NoSQL datastore.

Apache Spark

Unified Platform



Why Apache Spark?

- Distributed Parallel Cluster computing
  - Scalable , Fault tolerant
- Programming Framework
  - APIs in Java, Scala, Python
  - Less code than map reduce
- Fast
  - Runs computations in memory
  - Tasks are threads

Spark Use Cases

- **Iterative** Algorithms on **large amounts** of data

- Some Example Use Cases:
  - Anomaly detection
  - Classification
  - Predictions
  - Recommendations

Why Iterative Algorithms

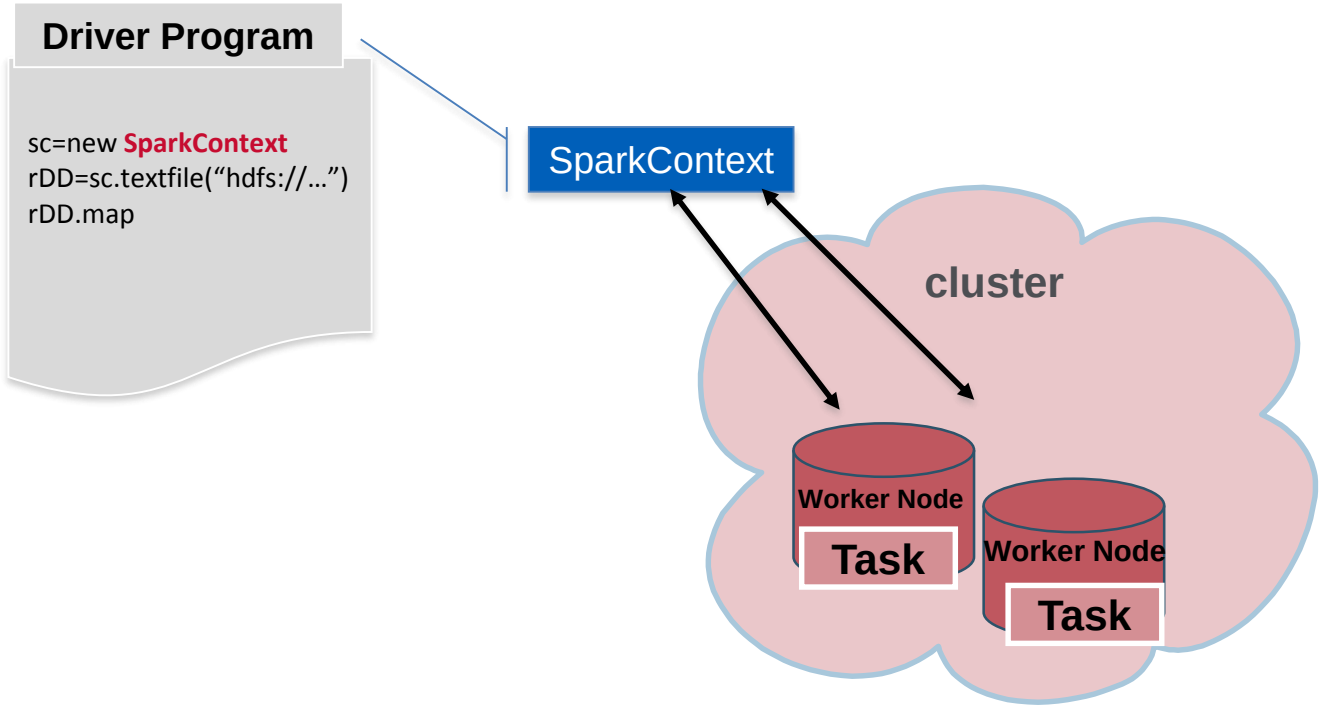
- Algorithms that need **iterations**
  - Clustering (K-Means, Canopy, …)
  - Gradient descent (e.g., Logistic Regression, Matrix Factorization)
  - Graph Algorithms (e.g., PageRank, Line-Rank, components, paths, reachability, centrality,)
  - **Alternating Least Squares ALS**
  - Graph communities / dense sub-components
  - Inference (believe propagation)
  - ...

Data Sources

- Local Files
- S3
- Hadoop Distributed Filesystem
  - any Hadoop InputFormat
- HBase
- other NoSQL data stores

How Spark Works

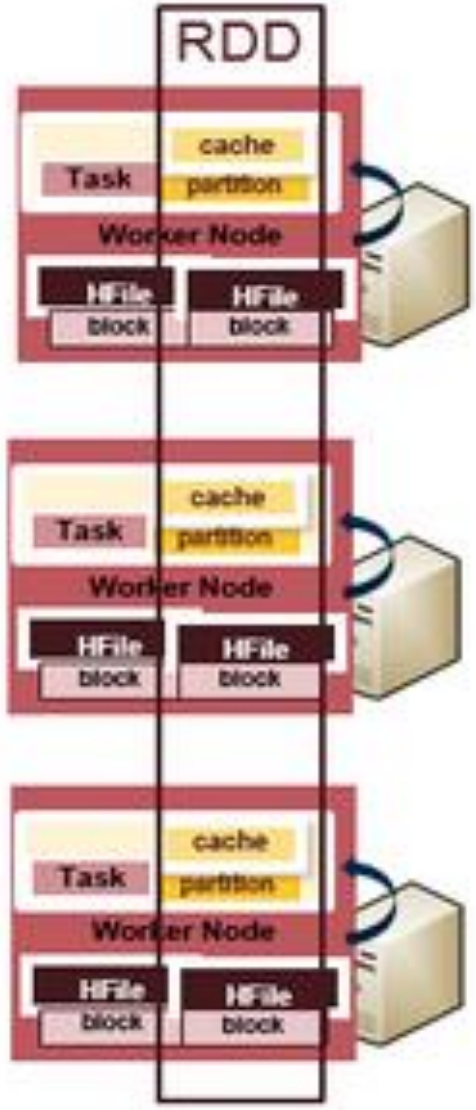
Spark Programming Model



Resilient Distributed Datasets (RDD)

Spark revolves around RDDs

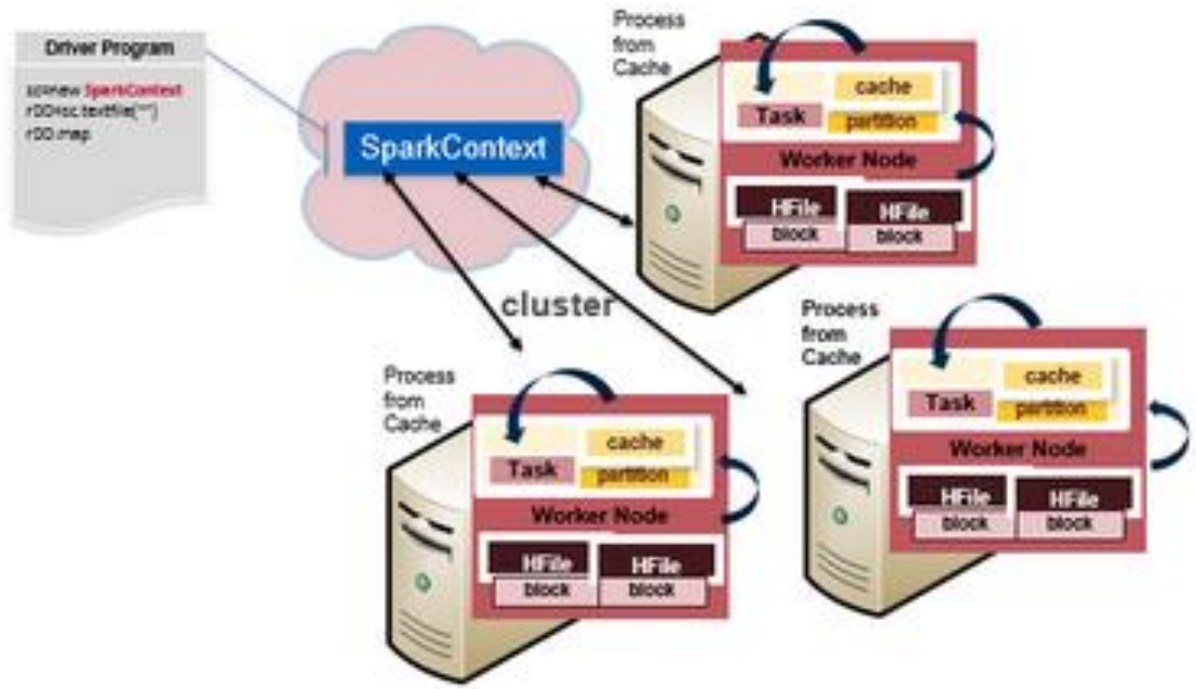
- read only collection of elements



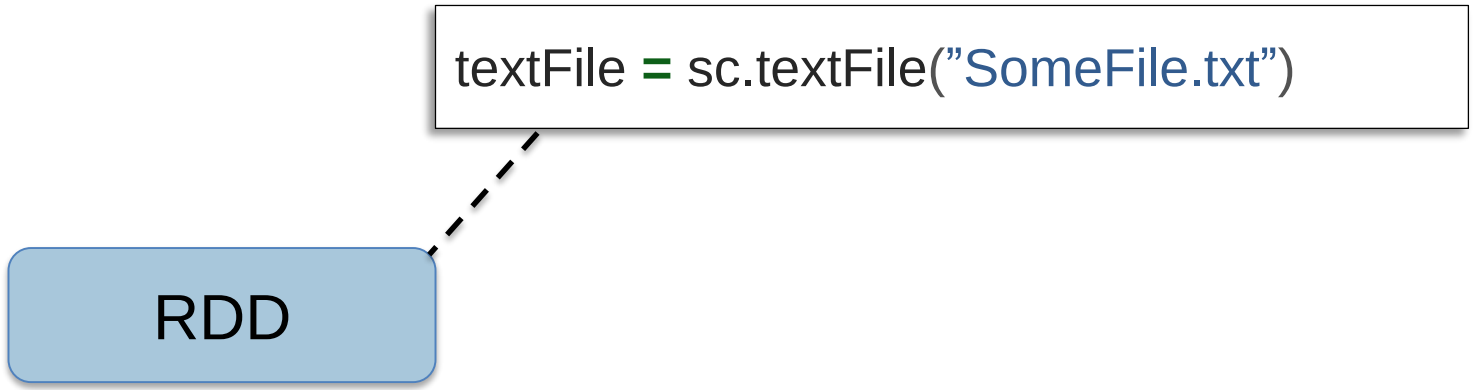
Resilient Distributed Datasets (RDD)

Spark revolves around RDDs

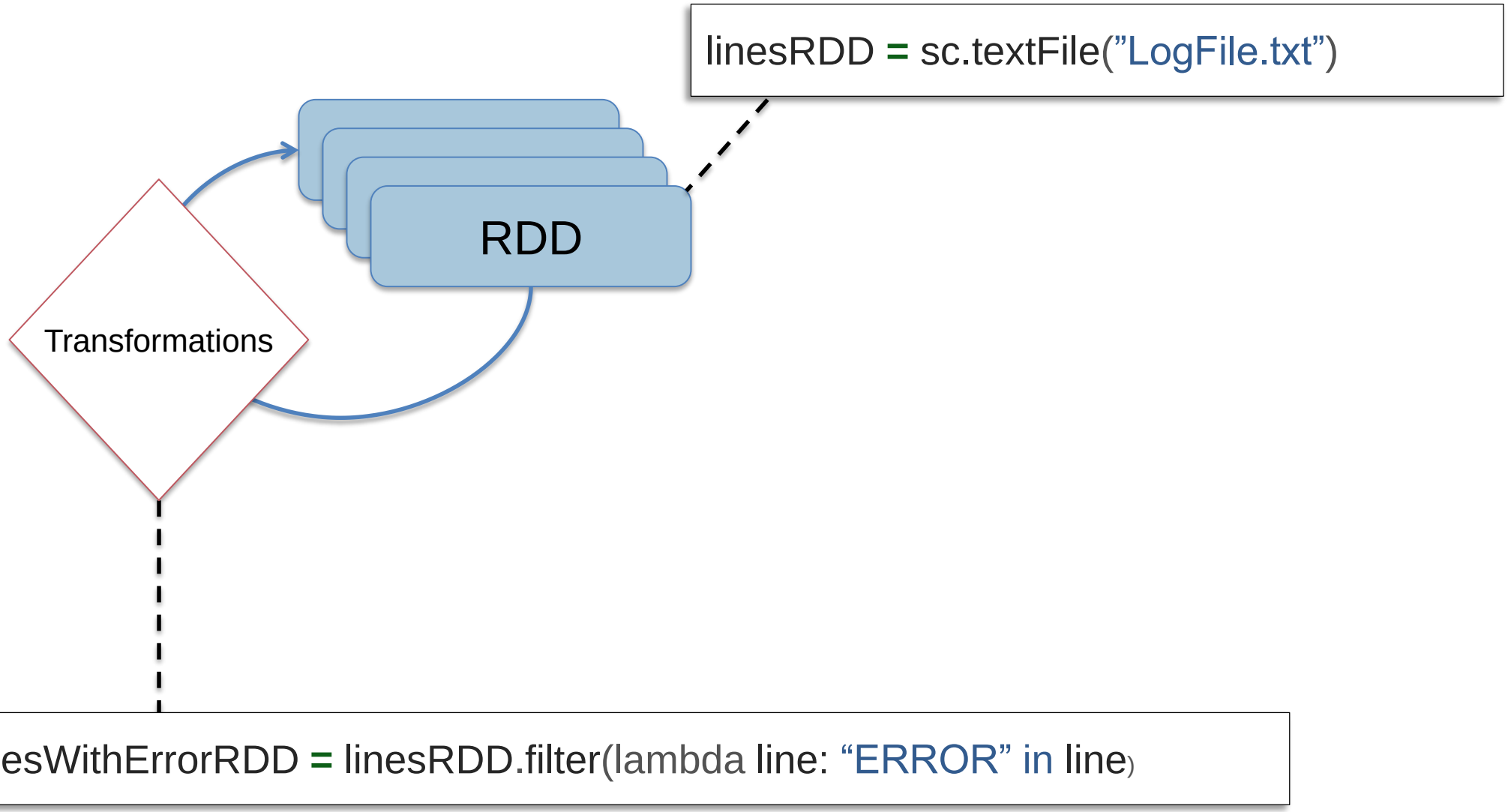
- read only **collection** of elements
- operated on in **parallel**
- **Cached** in memory
  - Or on disk
- Fault tolerant



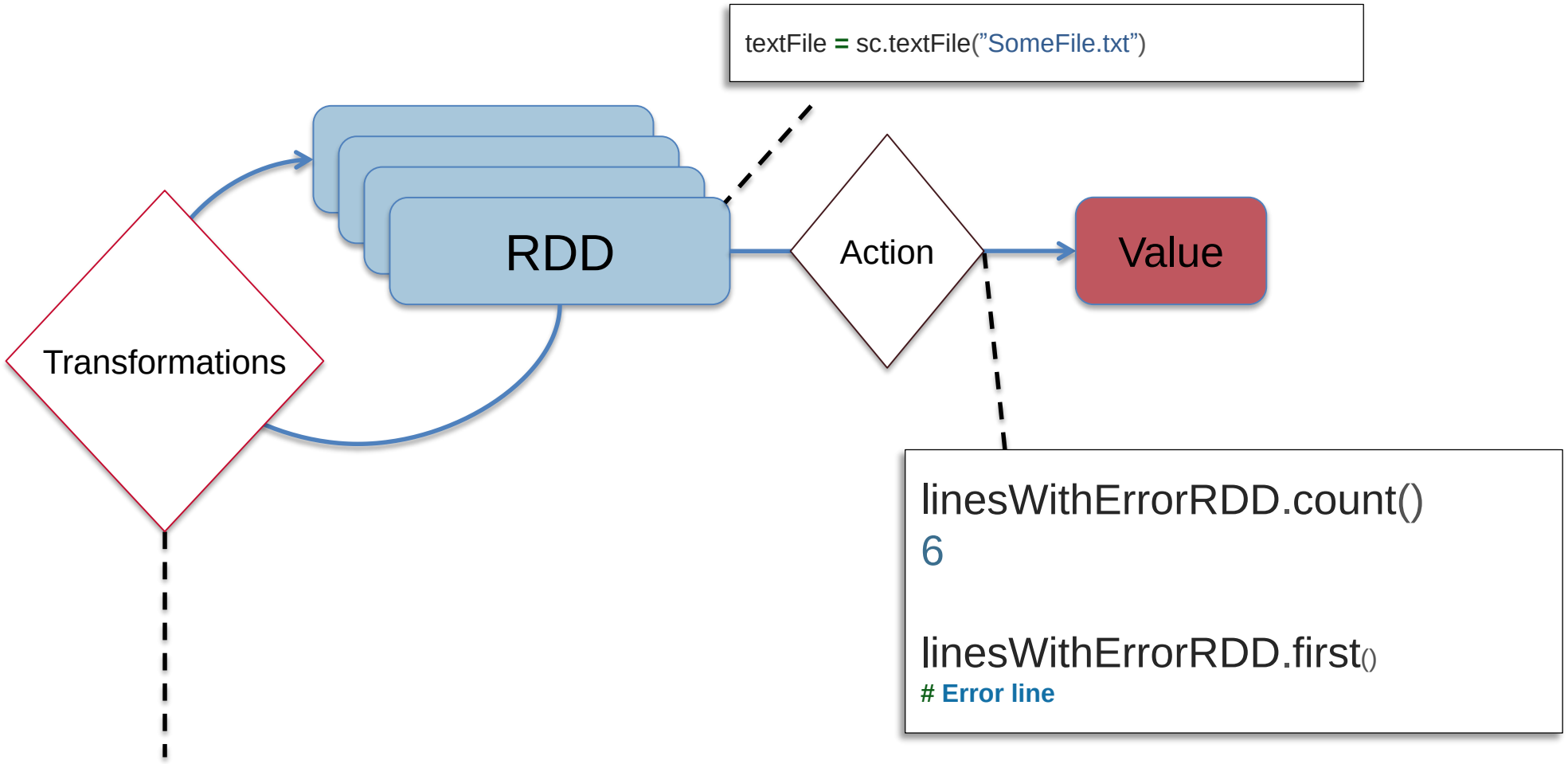
Working With RDDs



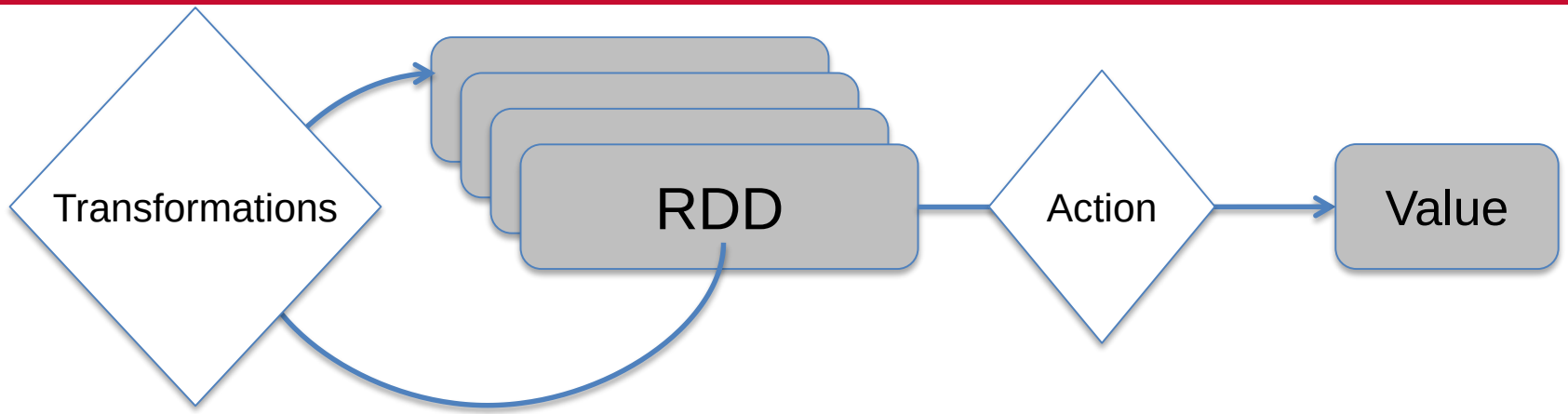
Working With RDDs



Working With RDDs



RDD Transformations and Actions



Transformations
(define a new RDD)

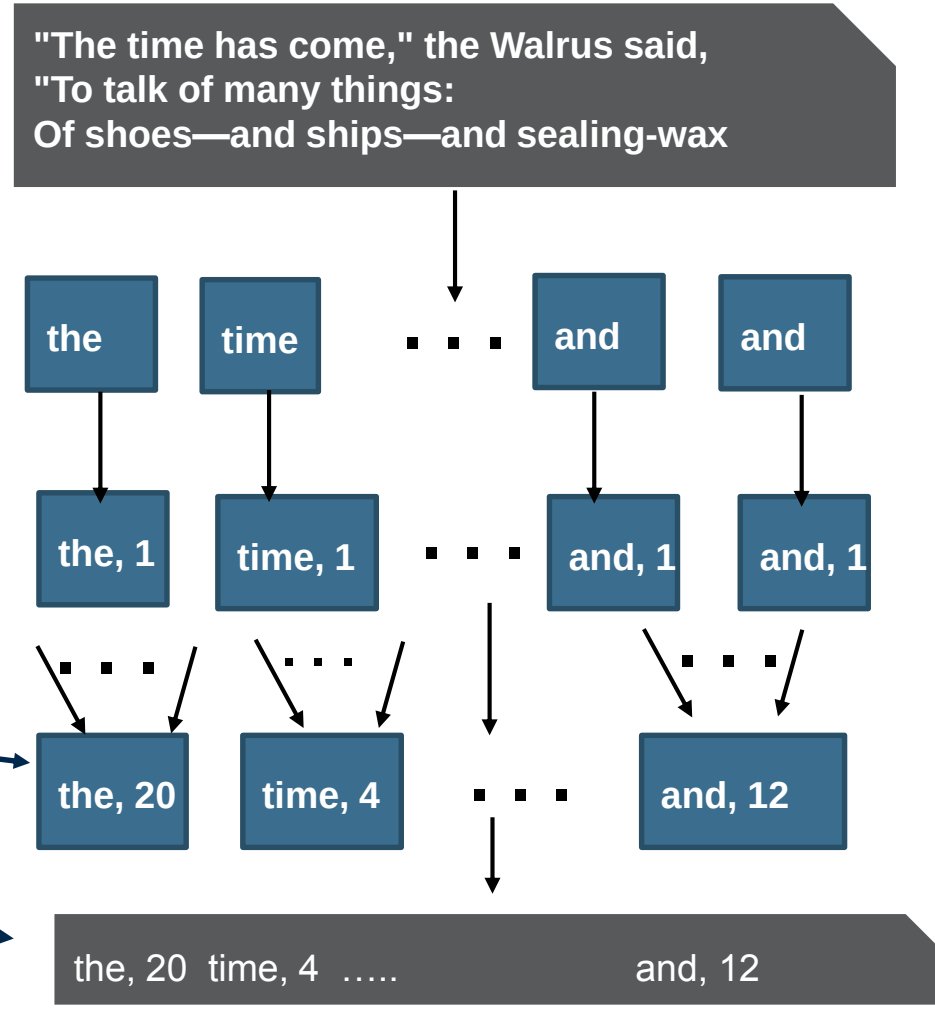
map
filter
sample
union
groupByKey
reduceByKey
join
cache
…

Actions
(return a value)

reduce
collect
count
save
lookupKey
…

Example Spark Word Count in Scala

```
// Load our input data.
val input =  sc.textFile(inputFile)

// Split it up into words.
val words = input.flatMap(line => line.split(" "))

// Transform into pairs and count.
val counts = words
      .map(word => (word, 1))
      .reduceByKey{case (x, y) => x + y}

// Save the word count back out to a text file,
counts.saveAsTextFile(outputFile)
```

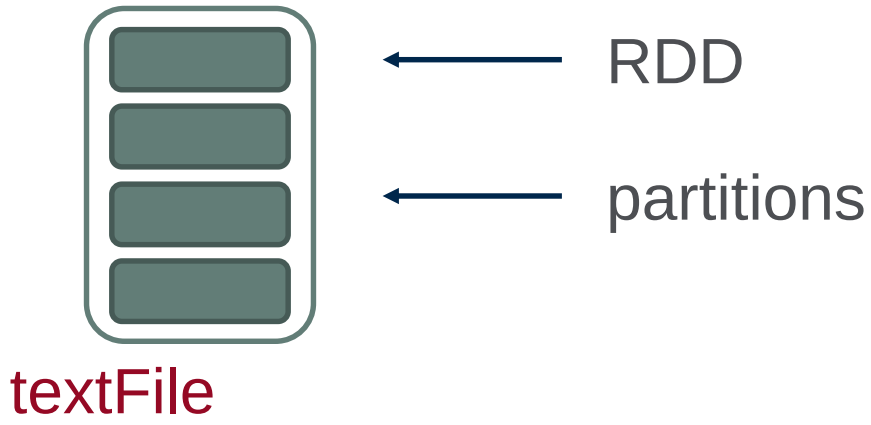


Example Spark Word Count in Scala

```
// Load  input data.
val input =  sc.textFile(inputFile)
```

HadoopRDD

MapPartitionsRDD



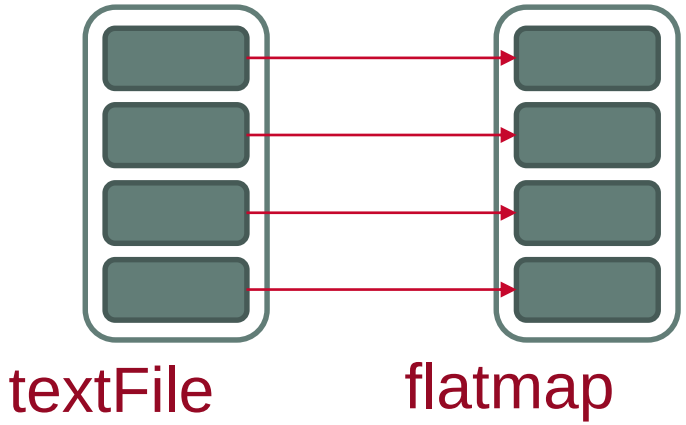
Example Spark Word Count in Scala

```
// Load our input data.
val input =  sc.textFile(inputFile)
// Split it up into words.
val words = input.flatMap(line => line.split(" "))
```

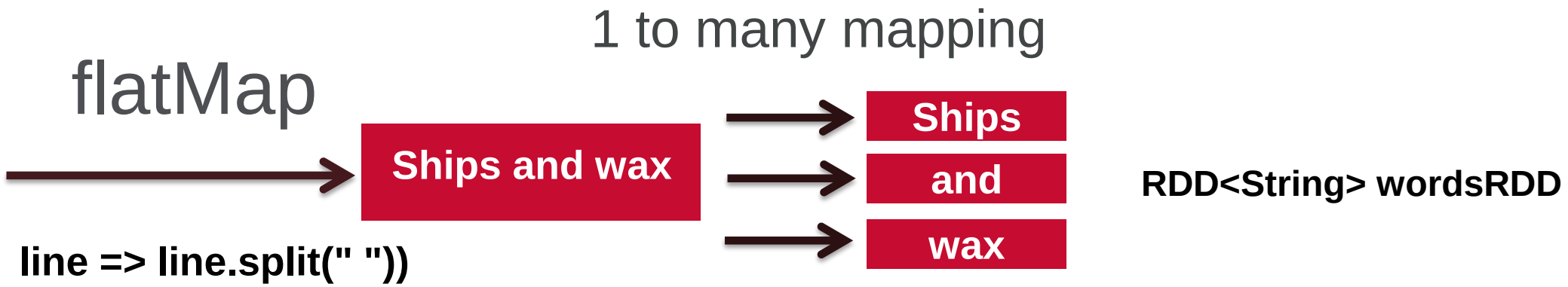
HadoopRDD

MapPartitionsRDD

MapPartitionsRDD



FlatMap


1 to many mapping
flatMap
Ships and wax
Ships
and
wax
line => line.split(" "))
RDD<String> wordsRDD

Example Spark Word Count in Scala

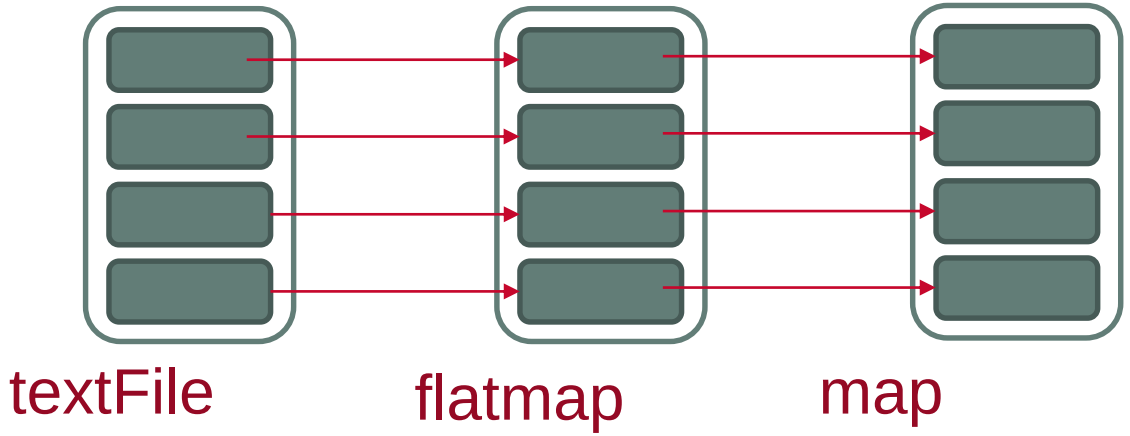
```
val input =  sc.textFile(inputFile)
val words = input.flatMap(line => line.split(" "))
// Transform into pairs
val counts = words.map(word => (word, 1))
```

HadoopRDD

MapPartitionsRDD

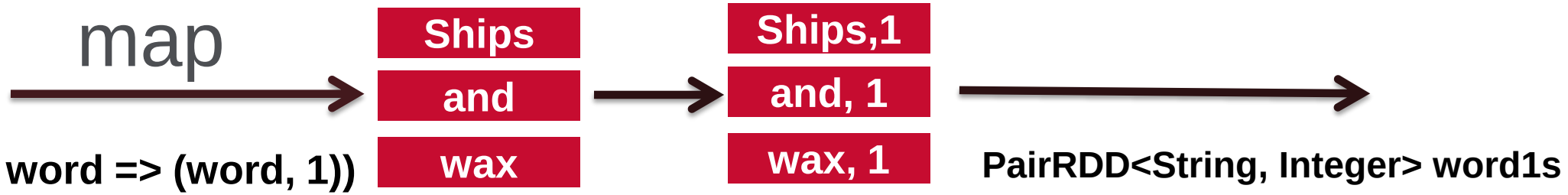
MapPartitionsRDD

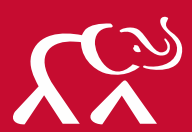
MapPartitionsRDD



Map

1 to 1 mapping





Example Spark Word Count in Scala

```
val input =  sc.textFile(inputFile)
val words = input.flatMap(line => line.split(" "))
val counts = words
      .map(word => (word, 1))
      .reduceByKey{case (x, y) => x + y}
```

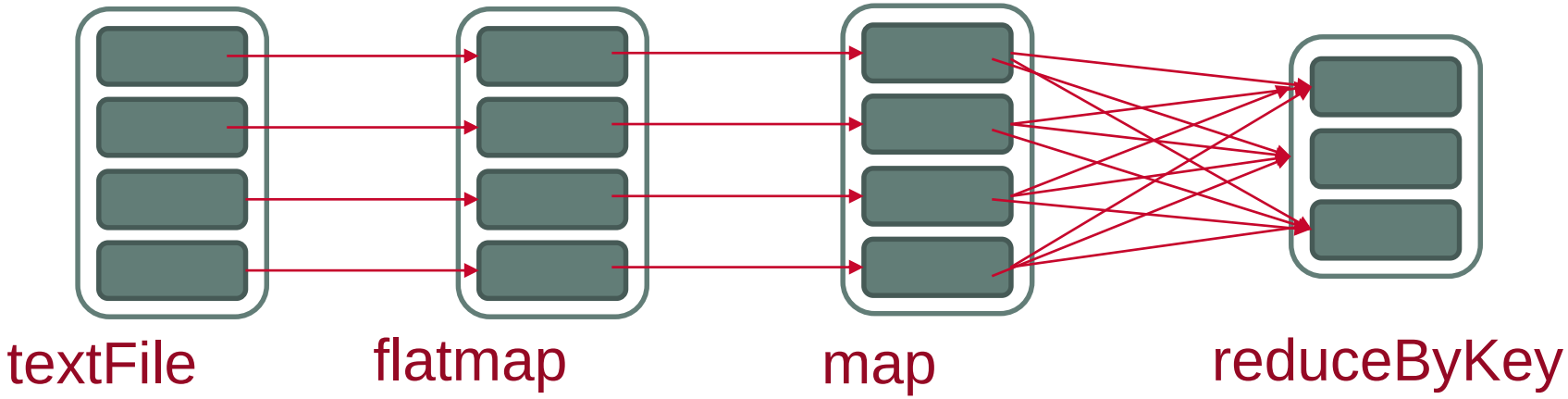
HadoopRDD

MapPartitionsRDD

MapPartitionsRDD

MapPartitionsRDD

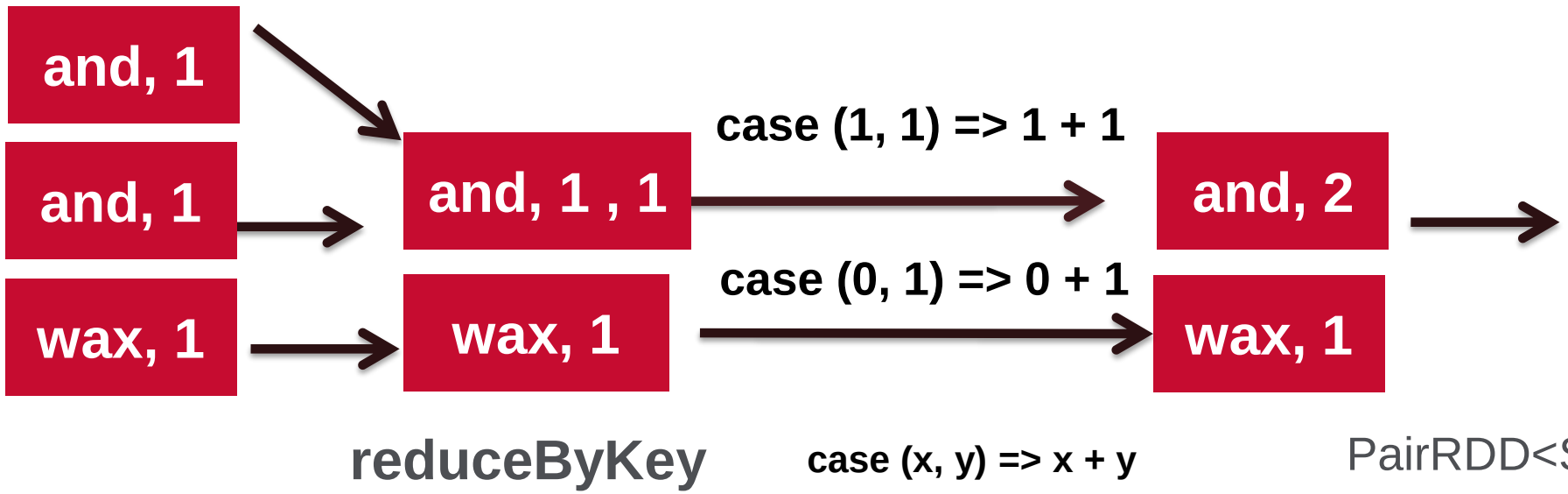
ShuffledRDD



reduceByKey

PairRDD Function :

reduceByKey(func: (value, value) ⇒ value): RDD[(key, value)



Example Spark Word Count in Scala

```
val input =  sc.textFile(inputFile)
val words = input.flatMap(line => line.split(" "))
val counts = words
        .map(word => (word, 1))
        .reduceByKey{case (x, y) => x + y}
val countArray = counts.collect()
```

HadoopRDD

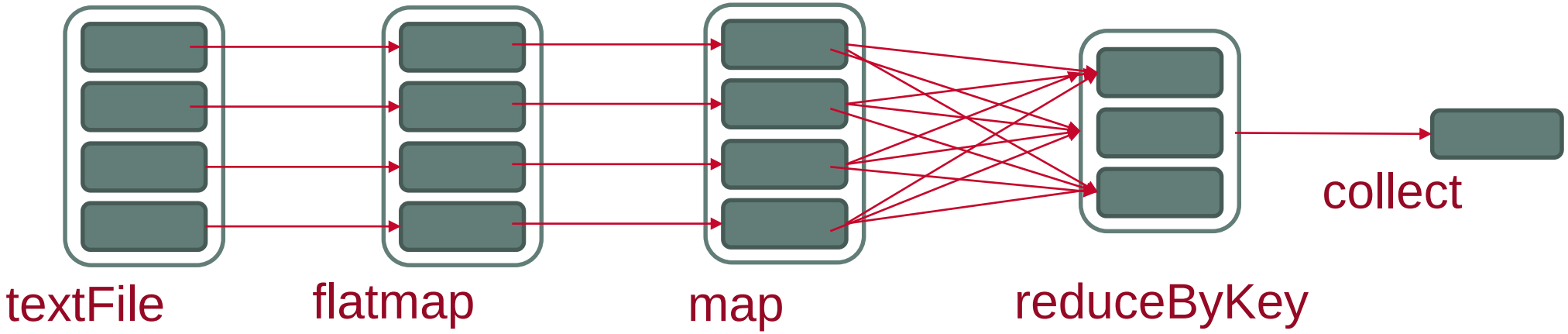
MapPartitionsRDD

MapPartitionsRDD

MapPartitionsRDD

ShuffledRDD

Array



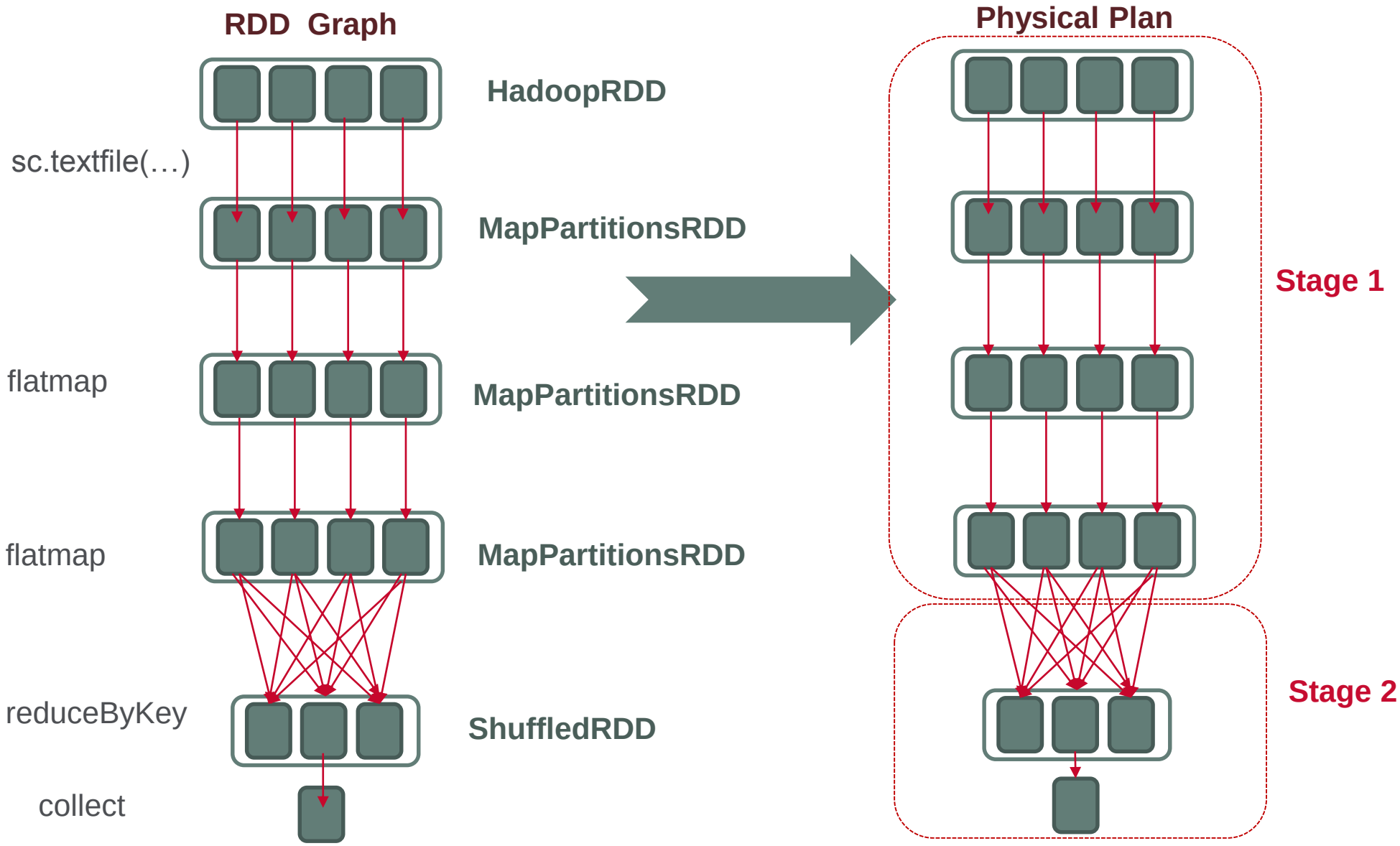
Demo Interactive Shell

- Iterative Development
  - Cache those RDDs
  - Open the shell and ask questions
    - We have all wished we could do this with MapReduce
  - Compile / save your code for scheduled jobs later
- Scala – spark-shell
- Python – pyspark

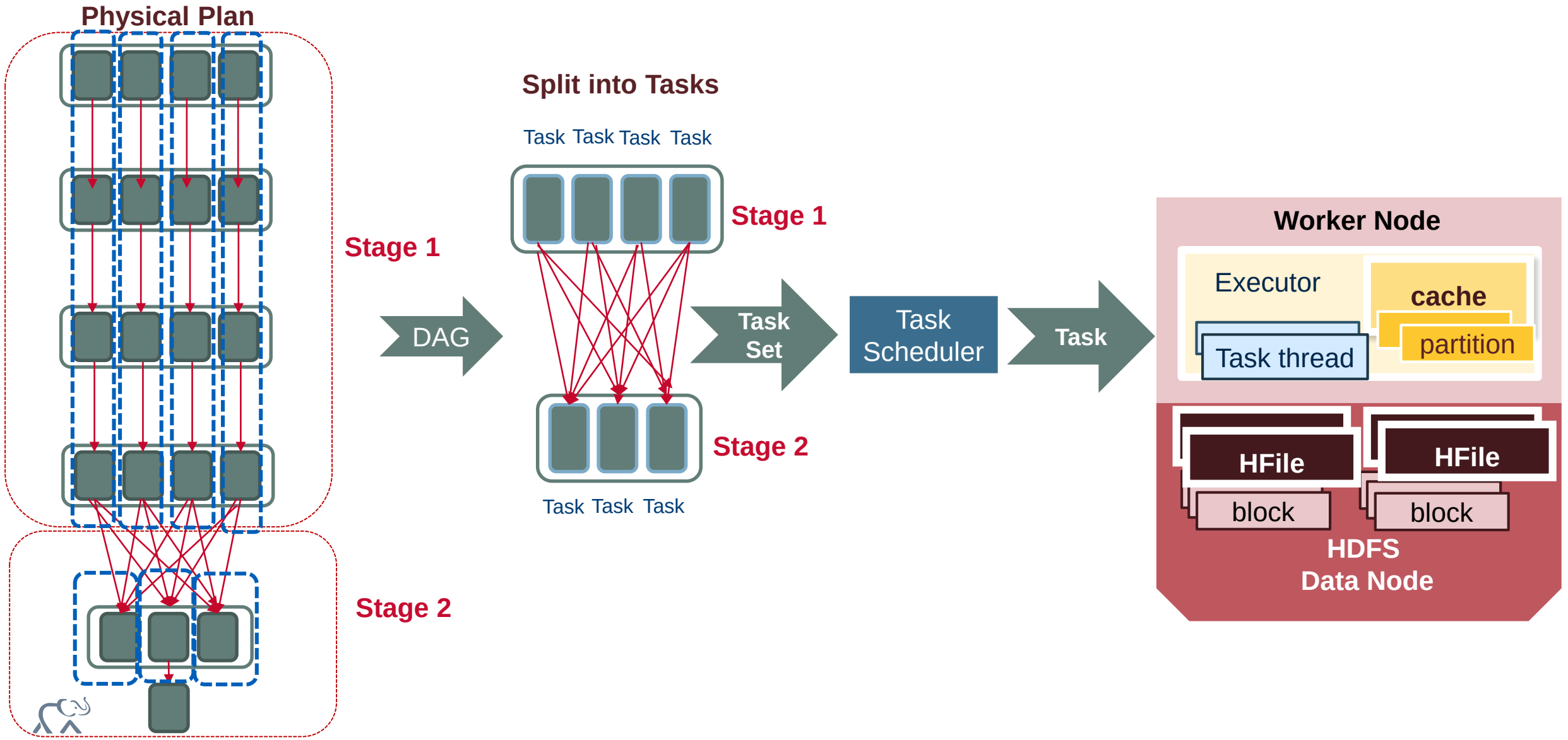
```
mbo@mbo-ubuntu-vbox:~/mbo/spark$ MASTER=spark://localhost:7077 ./spark-shell
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 0.9.0-SNAPSHOT
      /_/

Using Scala version 2.9.3 (Java HotSpot(TM) Client VM, Java 1.6.0_45)
Initializing interpreter...
Creating SparkContext...
Spark context available as sc.
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Components Of Execution

Spark RDD DAG -> Physical Execution plan



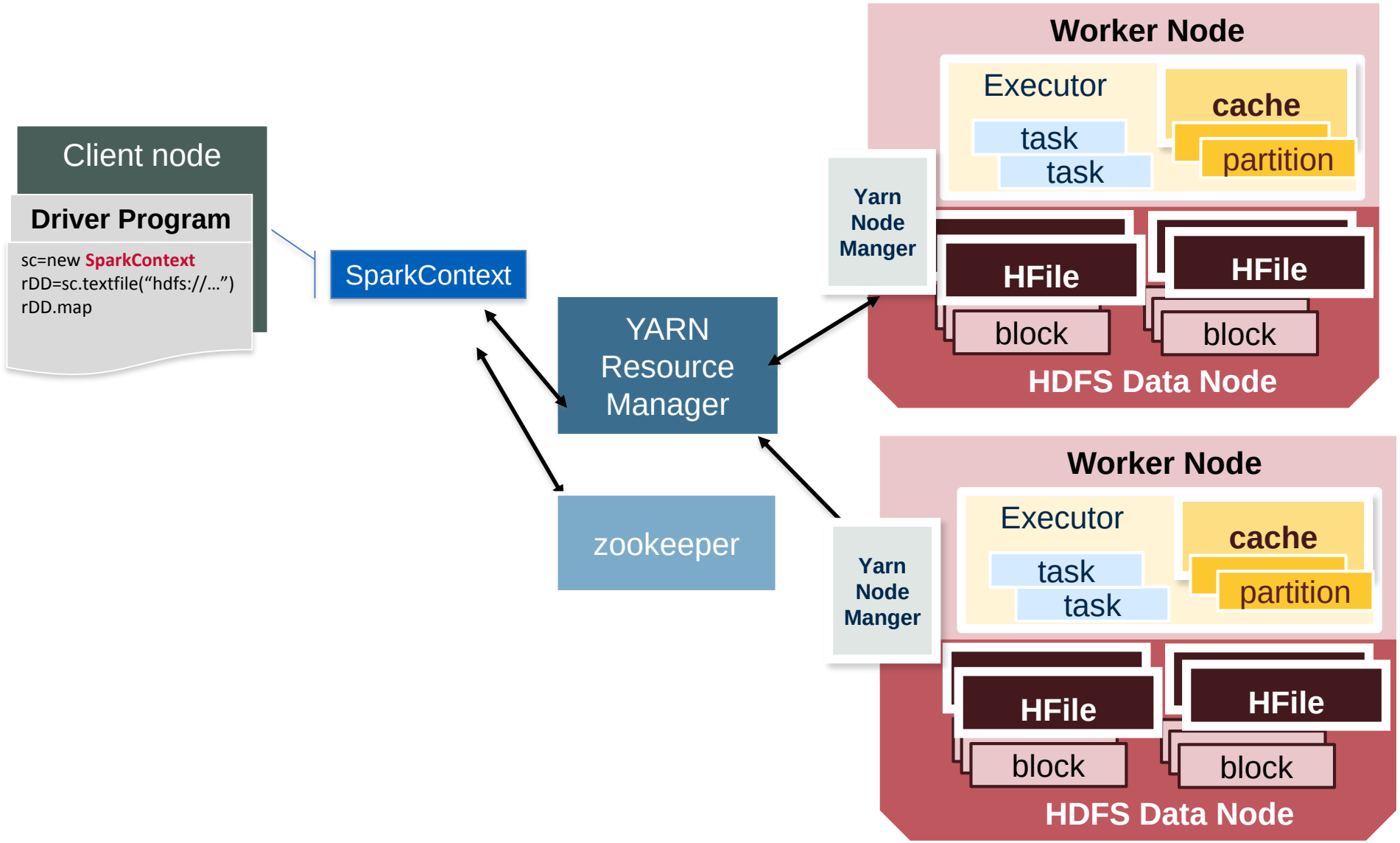
Physical Execution plan -> Stages and Tasks



Summary of Components

- Task ： unit of execution
- Stage: **Group** of Tasks
  - Base on **partitions** of RDD
  - Tasks run in **parallel**
- DAG ： Logical Graph of RDD operations
- RDD ： **Parallel dataset** with partitions

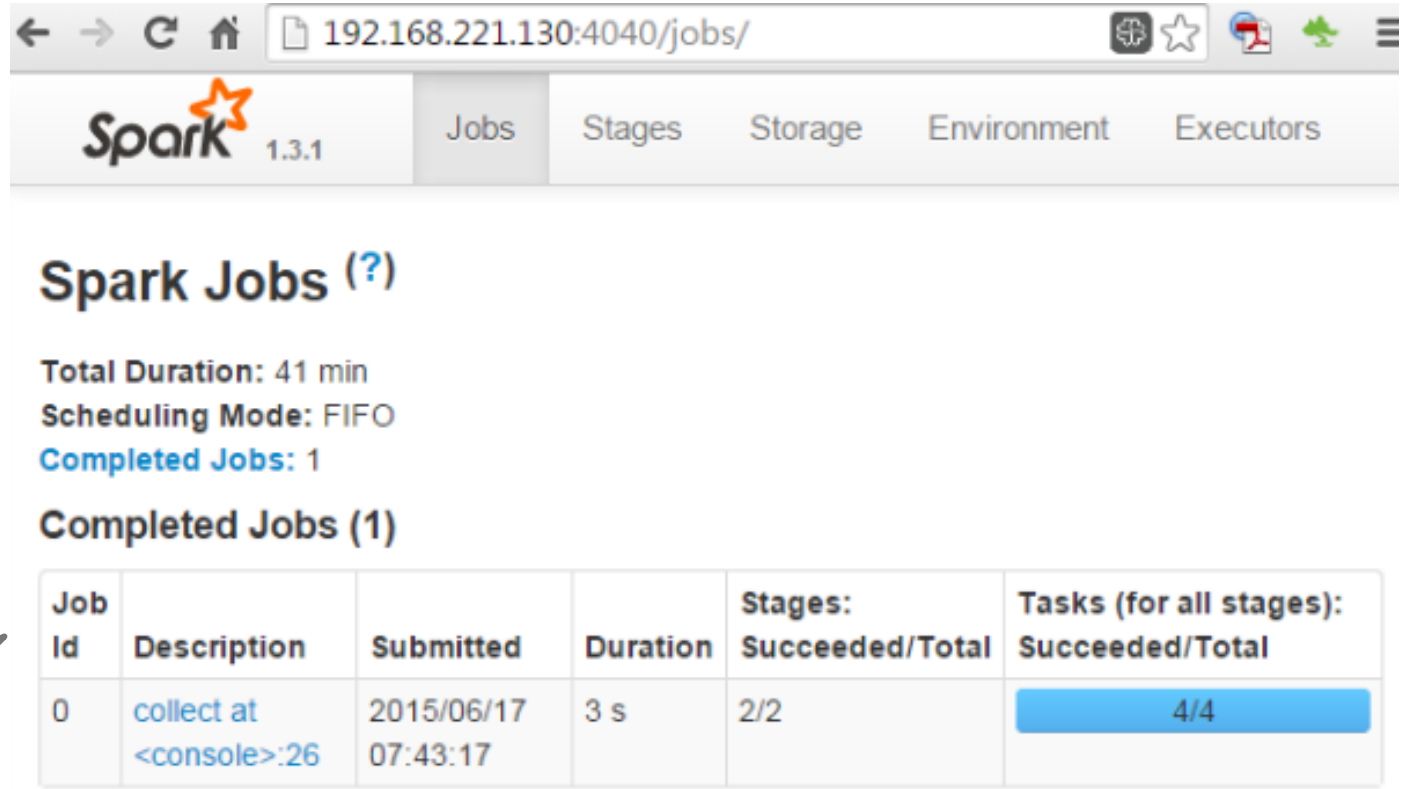
How Spark Application runs on a Hadoop cluster



Deploying Spark – Cluster Manager Types

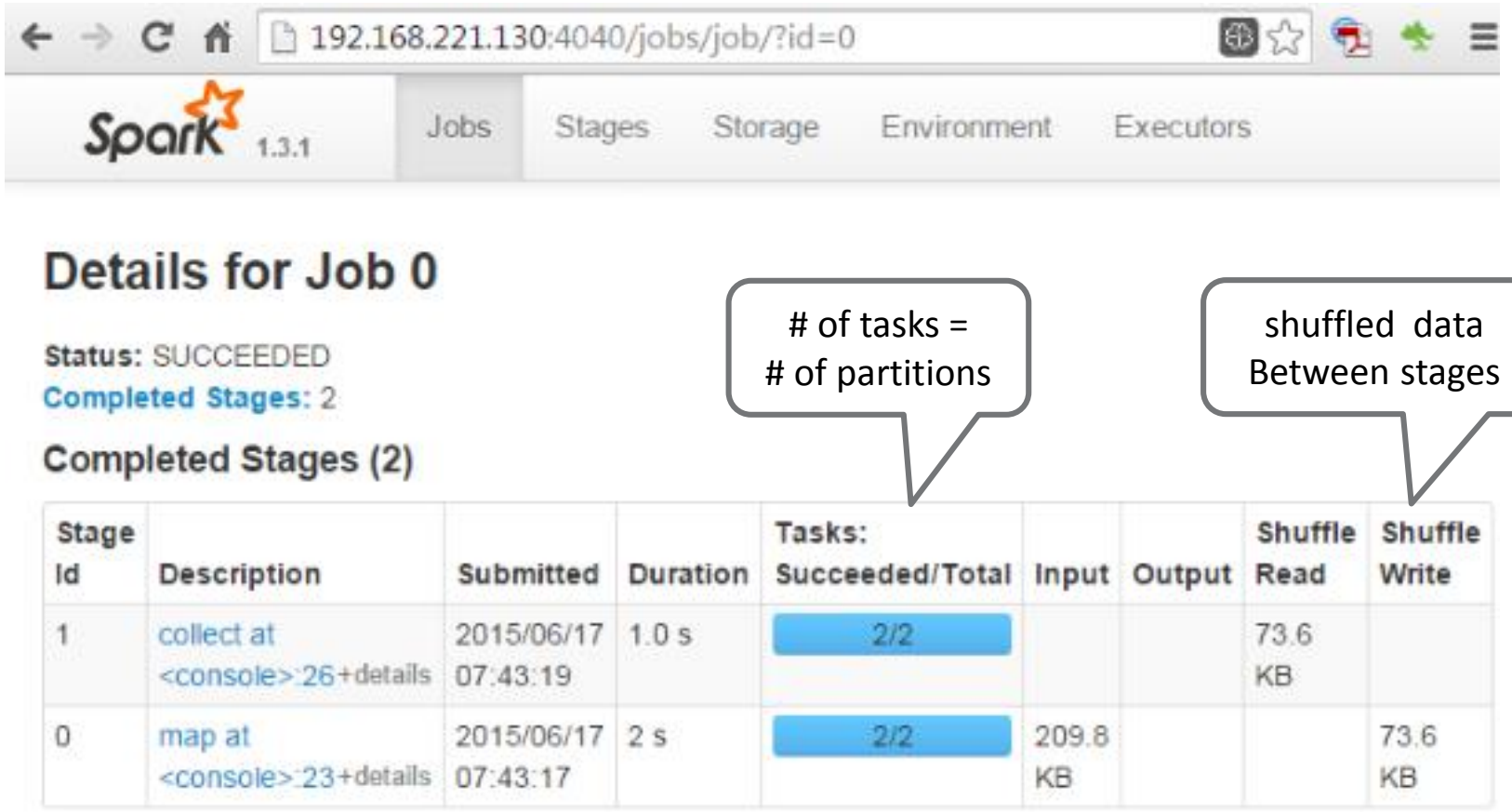
- Standalone mode
- Mesos
- YARN
- EC2
- GCE

Spark Web UI: Jobs

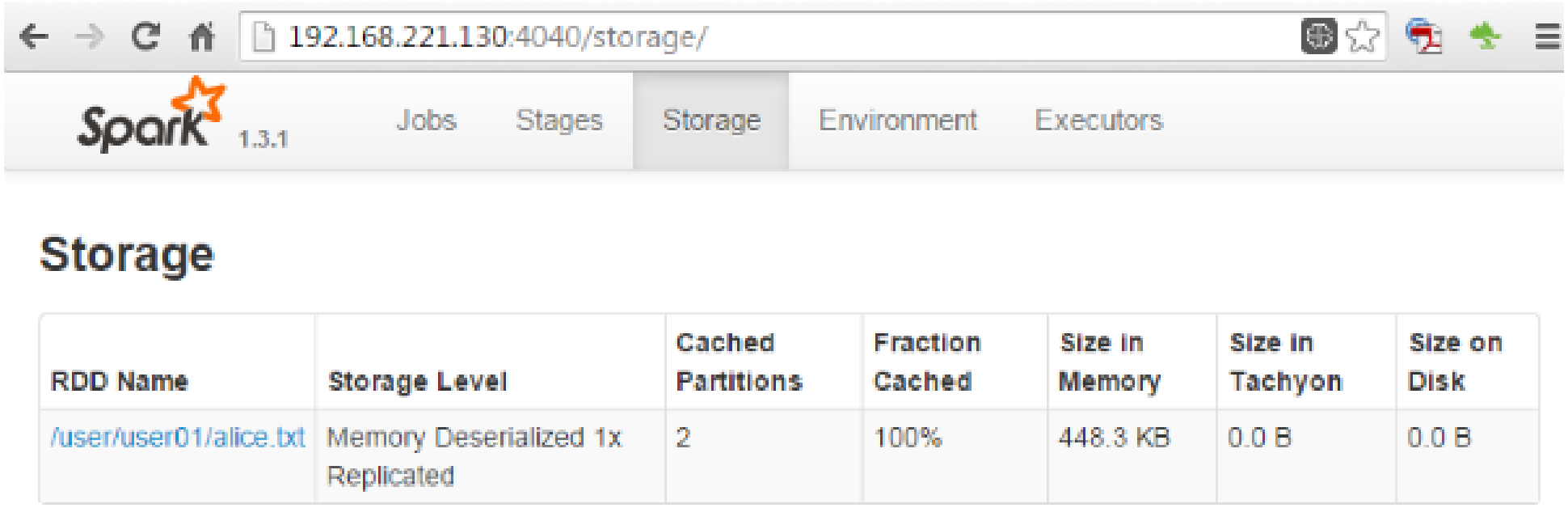


Job description
Is last action

Spark Web UI: Stages



Spark Web UI: Storage



RDD

Spark Web UI: Executors

192.168.221.130:4040/executors/

Spark 1.3.1 Jobs Stages Storage Environment Executors

Executors (1)

Memory: 448.3 KB Used (246.0 MB Total)
Disk: 0.0 B Used

Executor ID	Address	RDD Blocks	Memory Used	Disk Used	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time	Input	Shuffle Read	Shuffle Write	Thread Dump
<driver>	localhost:56501	2	448.3 KB / 246.0 MB	0.0 B	0	0	4	4	4.6 s	209.8 KB	0.0 B	73.6 KB	Thread Dump

Thread call stack

MapR Tutorial: Getting Started with Spark on MapR Sandbox

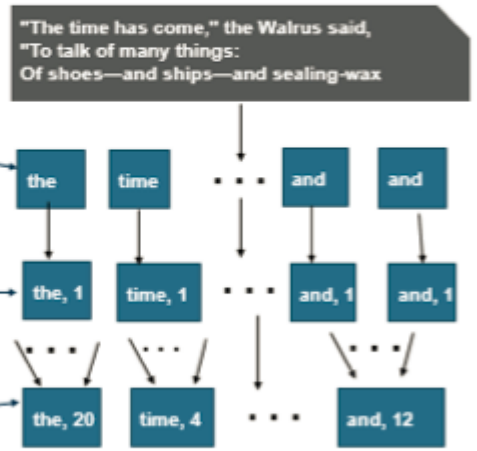
- https://www.mapr.com/products/mapr-sandbox-hadoop/tutorials/spark-tutorial

Next, we will look at how to write, compile, and run a Standalone Spark word count application

step through the following word count application in Java.

Example Word Count App in Java

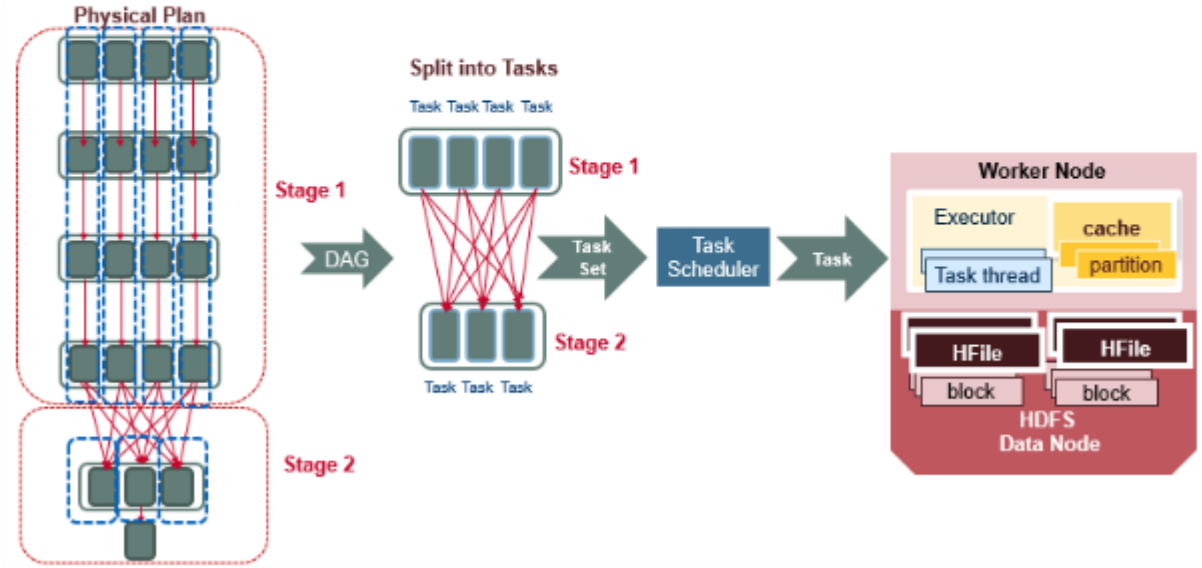
```
JavaRDD<String> input = sc.textFile(inputFile);
// Split each line into words
JavaRDD<String> words = input.flatMap(
  new FlatMapFunction<String, String>() {
    public Iterable<String> call(String x) {
      return Arrays.asList(x.split(" "));
  }});
// Turn the words into (word, 1) pairs
JavaPairRDD<String, Integer> word1s== words.mapToPair(
  new PairFunction<String, String, Integer>(){
    public Tuple2<String, Integer> call(String x){
      return new Tuple2(x, 1);
  }});
// reduce add the pairs by key to produce counts
JavaPairRDD<String, Integer> counts =word1s.reduceByKey(
  new Function2<Integer, Integer, Integer>(){
    public Integer call(Integer x, Integer y){
      return x + y;
  }});
```



MapR Blog: Getting Started with the Spark Web UI

- https://www.mapr.com/blog/getting-started-spark-web-ui

The scheduler splits the RDD graph into stages, based on the transformations. The narrow transformations (transformations without data movement) will be grouped (pipe-lined) together into a single stage. This physical plan has two stages, with everything before ShuffledRDD in the first stage.



Example: Log Mining

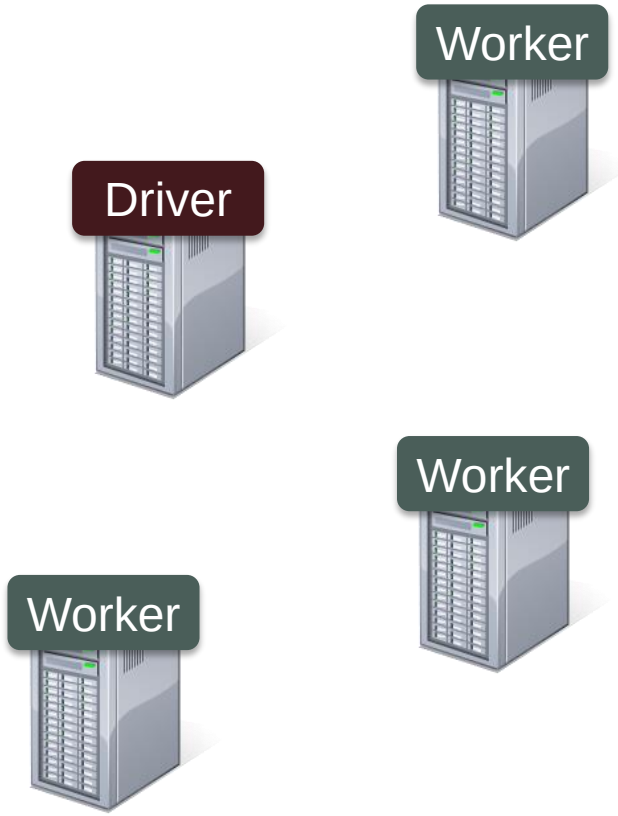
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

Based on slides from Pat McDonough at DATABRICKS

Example: Log Mining

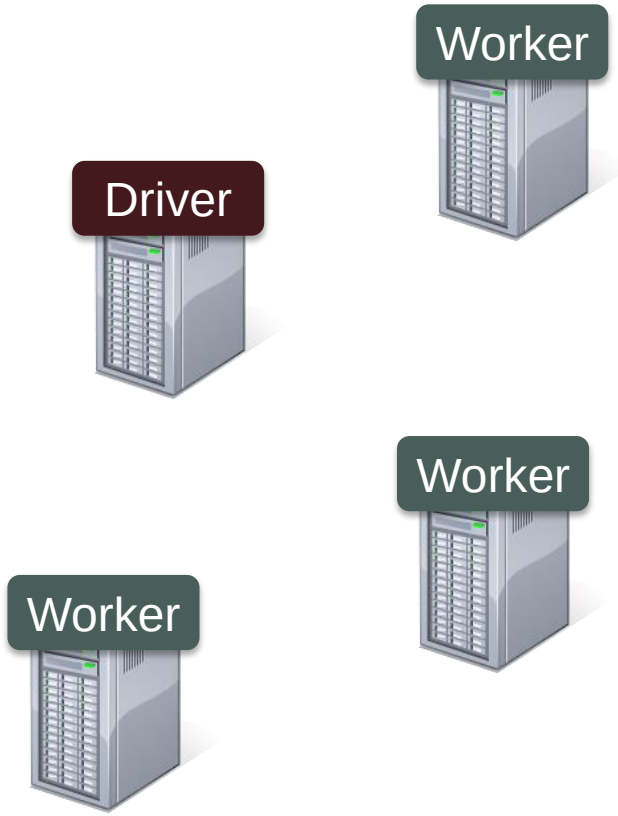
Load error messages from a log into memory, then interactively search for various patterns



Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
```

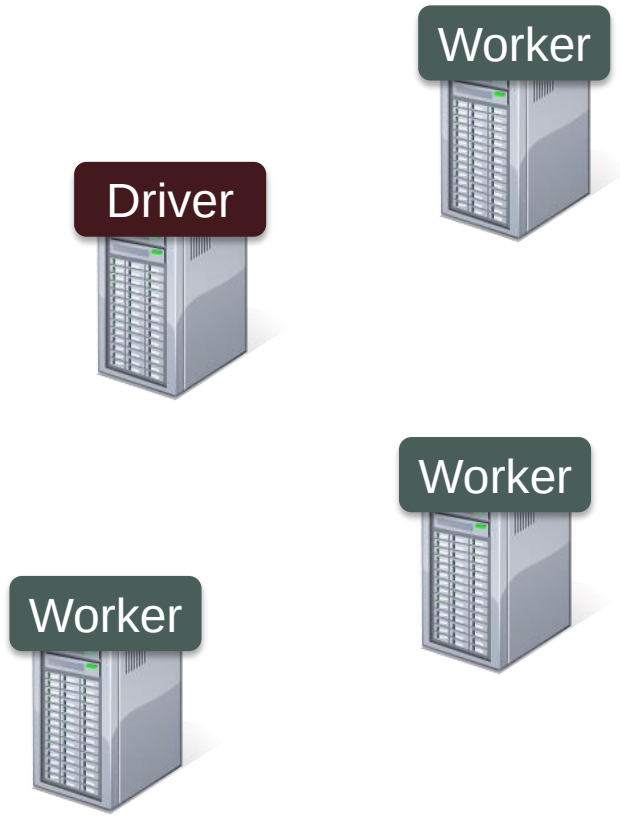


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

Base RDD

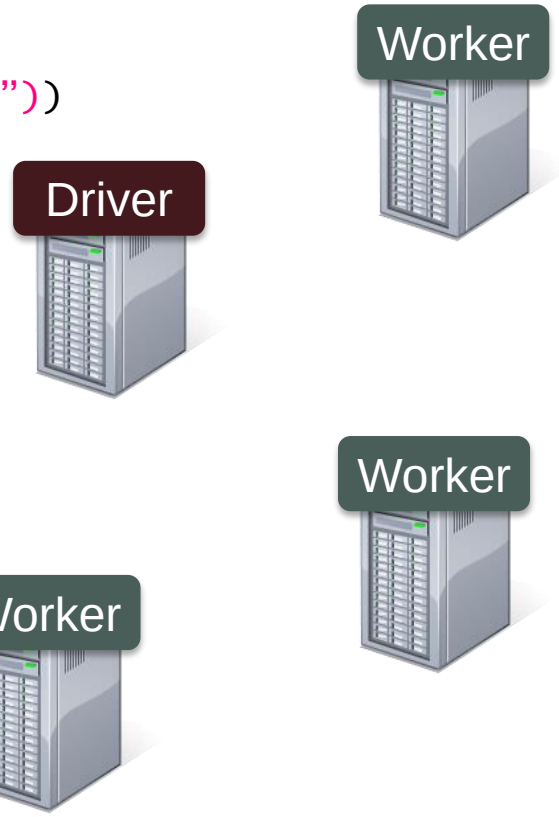
```
lines = spark.textFile(“hdfs://...”)
```



Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
```

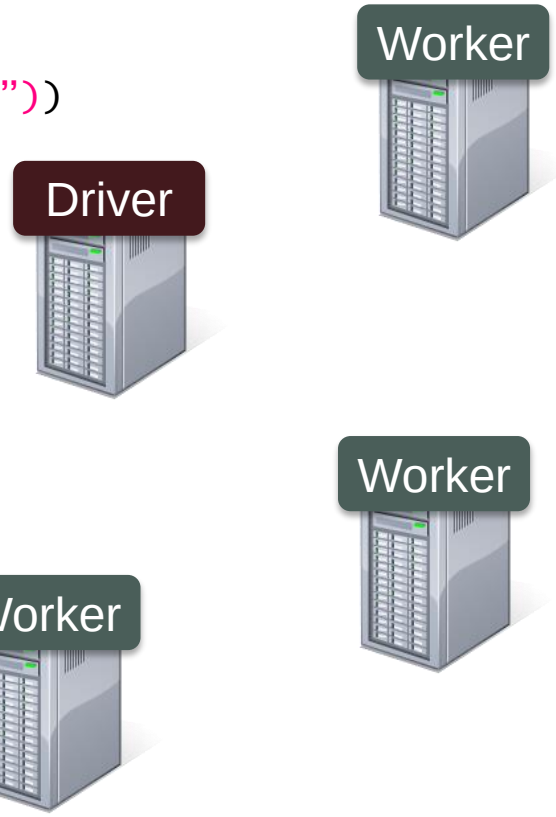


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

Transformed RDD

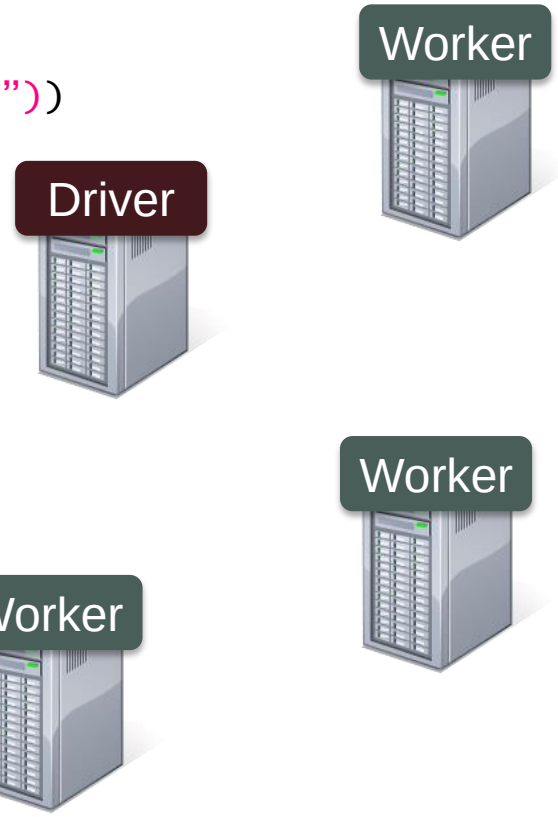
```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
```



Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

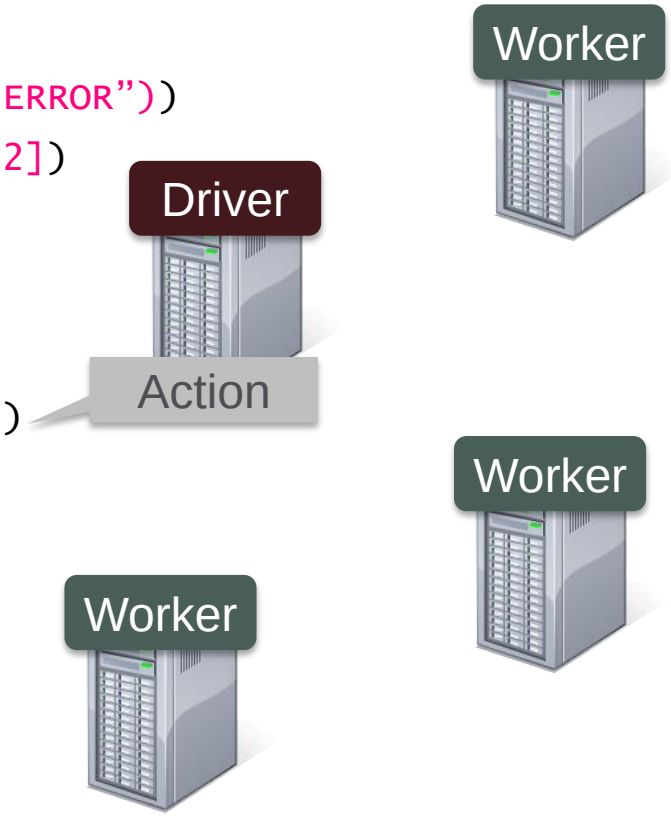
```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()


messages.filter(lambda s: “mysql” in s).count()
```



Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

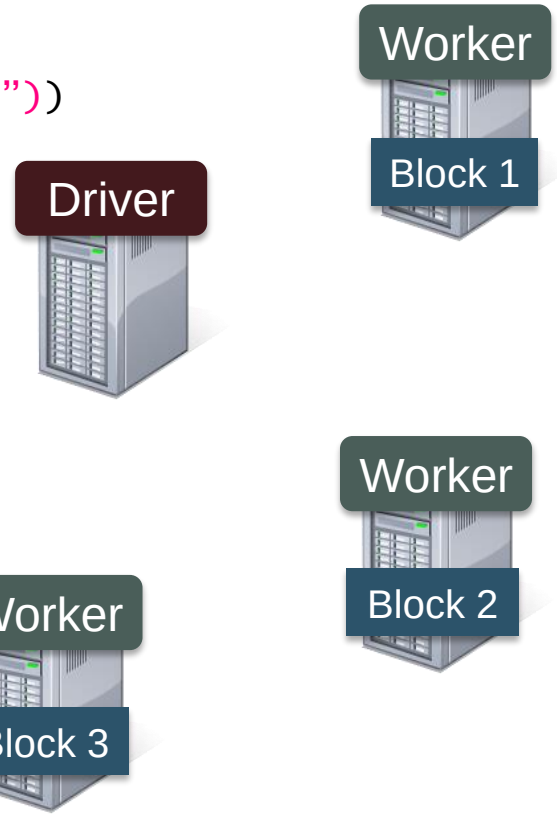
```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()


messages.filter(lambda s: “mysql” in s).count()
```



Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()


messages.filter(lambda s: “mysql” in s).count()
```

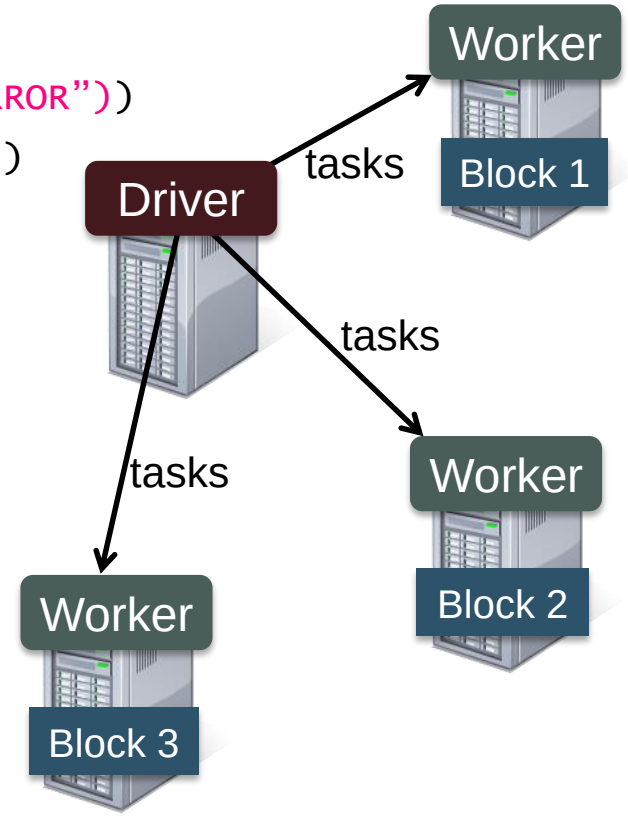


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()
```

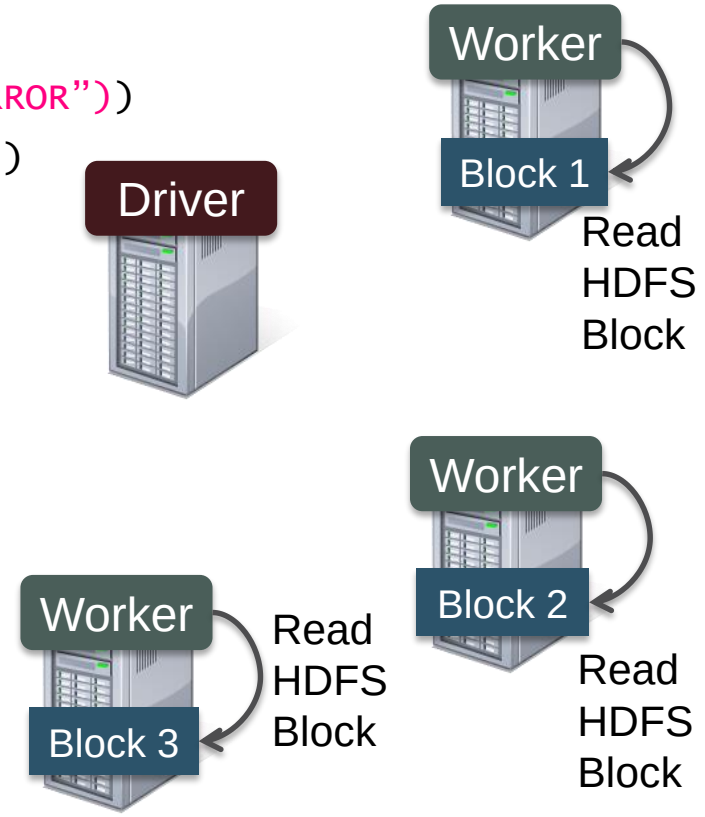
```
messages.filter(lambda s: “mysql” in s).count()
```



Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

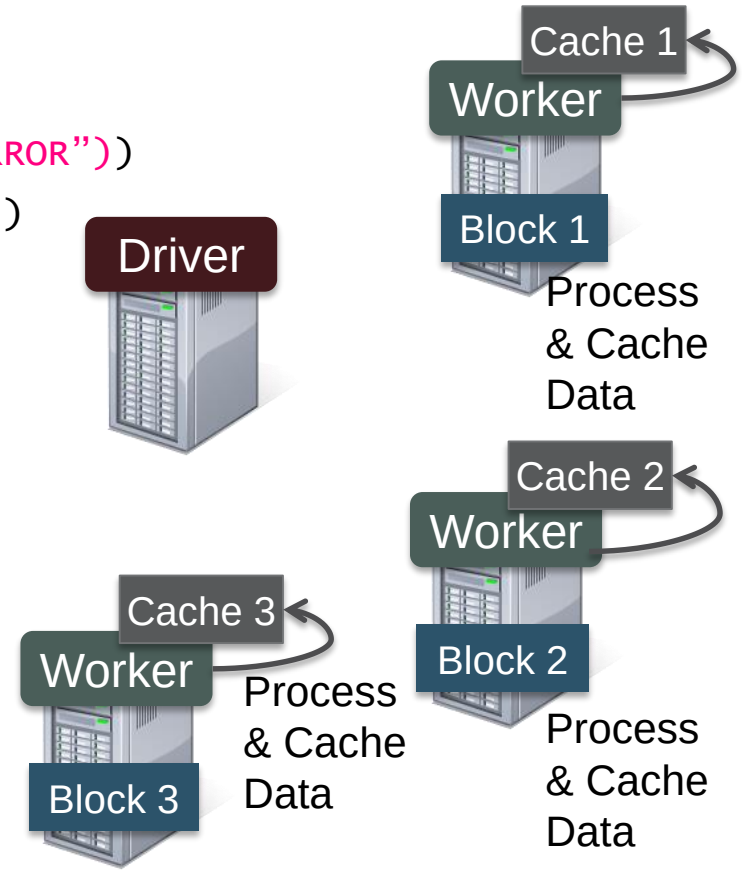
```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()


messages.filter(lambda s: “mysql” in s).count()
```



Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()


messages.filter(lambda s: “mysql” in s).count()
```

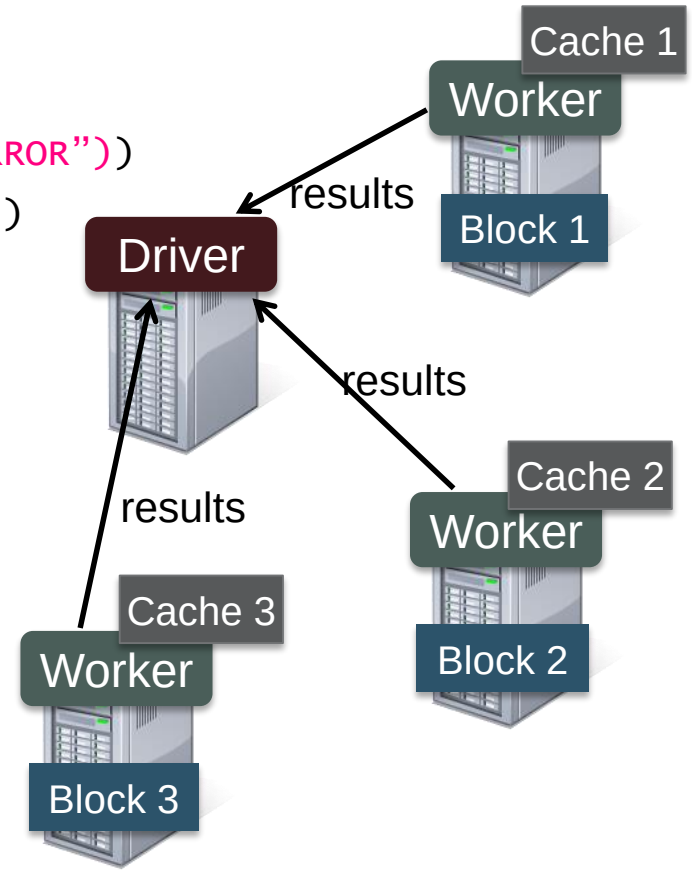


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()
```

```
messages.filter(lambda s: “mysql” in s).count()
```

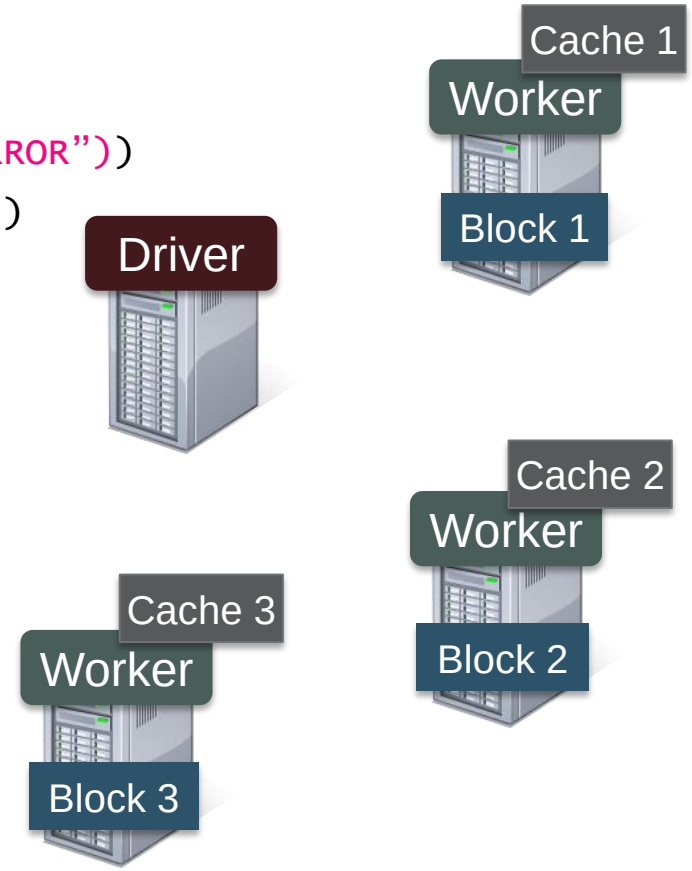


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()
```

```
messages.filter(lambda s: “mysql” in s).count()
messages.filter(lambda s: “php” in s).count()
```

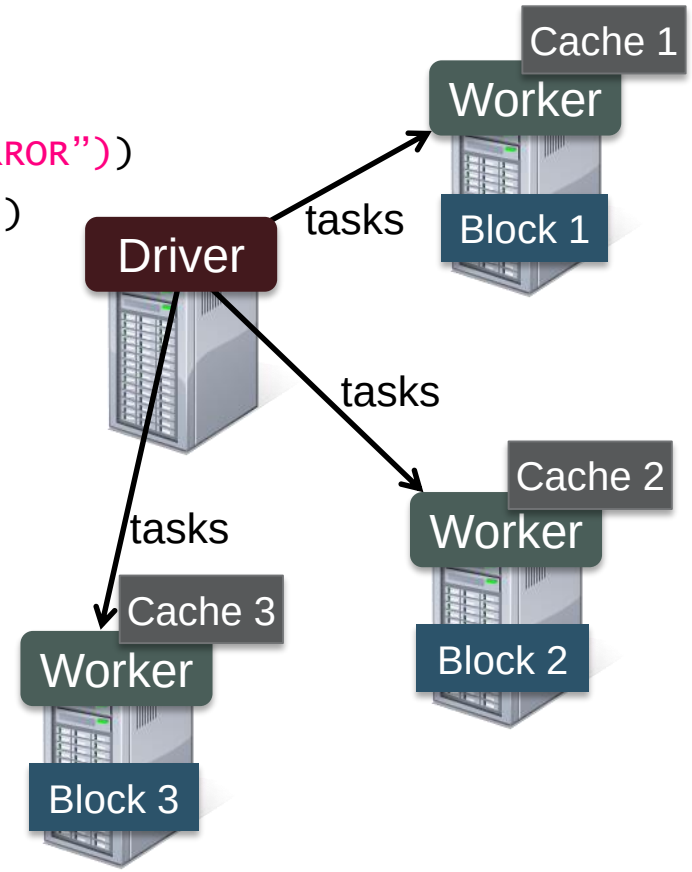


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()
```

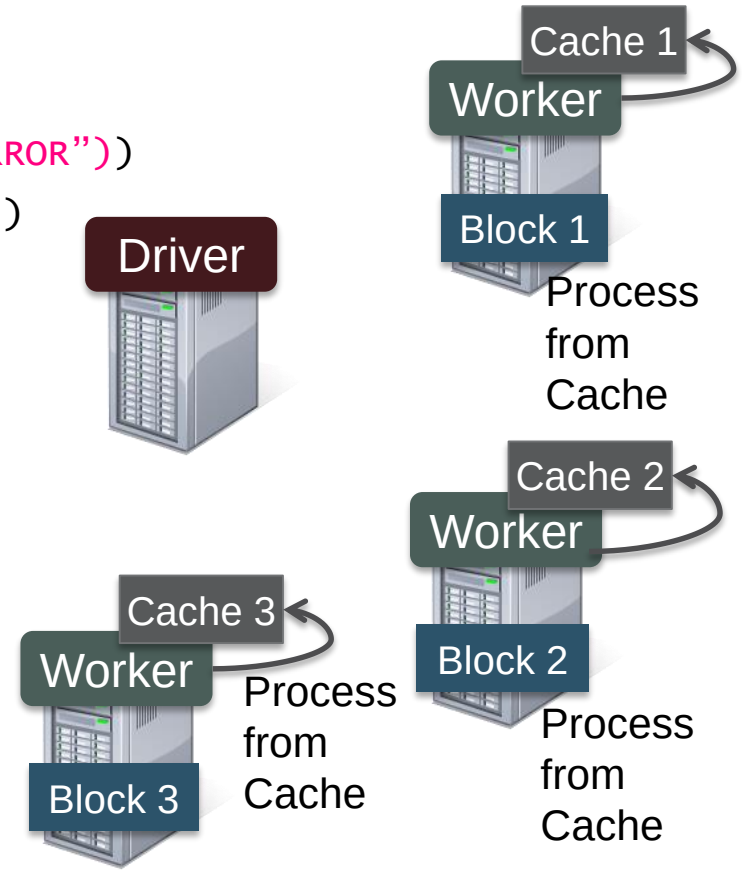
```
messages.filter(lambda s: “mysql” in s).count()
messages.filter(lambda s: “php” in s).count()
```



Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()


messages.filter(lambda s: “mysql” in s).count()
messages.filter(lambda s: “php” in s).count()
```

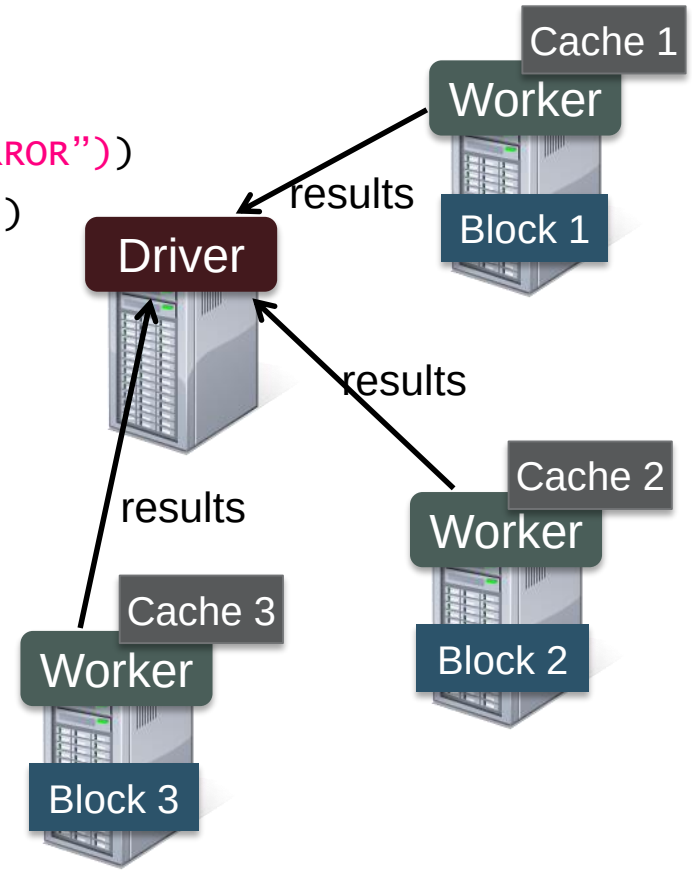


Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(“\t”)[2])
messages.cache()
```

```
messages.filter(lambda s: “mysql” in s).count()
messages.filter(lambda s: “php” in s).count()
```



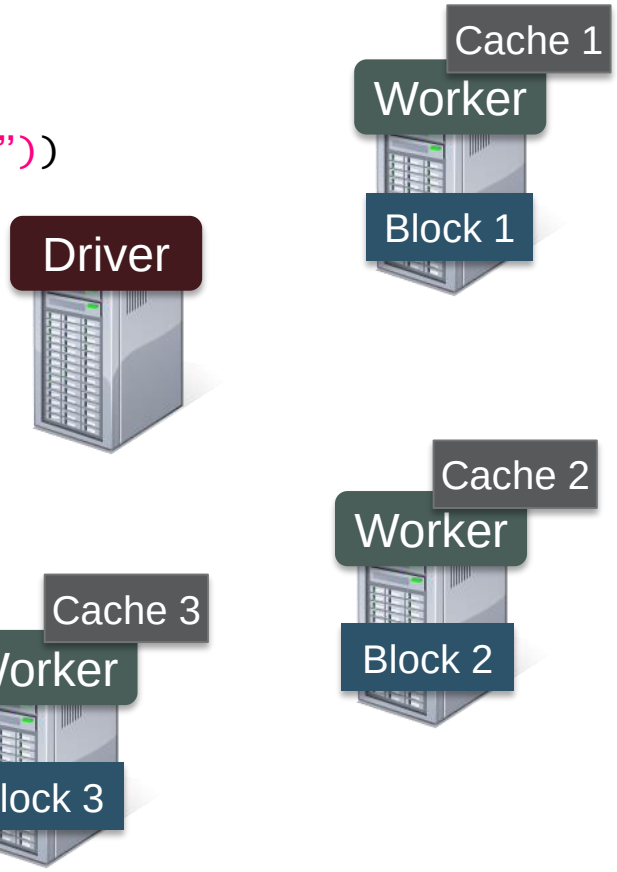
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split("\t")[2])
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()
messages.filter(lambda s: "php" in s).count()
```

Cache your data ➔ Faster Results



Dataframes

Scenario: Online Auction Data

AuctionID	Bid	Bid Time	Bidder	Bidder Rate	Open Bid	Price	Item	Days To Live
8213034705	95	2.927373	jake7870	0	95	117.5	xbox	3
8213034705	115	2.943484	davidbresler2	1	95	117.5	xbox	3
8213034705	100	2.951285	gladimacowgirl	58	95	117.5	xbox	3
8213034705	117.5	2.998947	daysrus	95	95	117.5	xbox	3

Creating DataFrames

```
// Define schema using a Case class
case class Auction(auctionid: String, bid: Float,
bidtime: Float, bidder: String, bidderrate: Int,
openbid: Float, finprice: Float, itemtype: String,
dtl: Int)
```

Creating DataFrames

```
// Create the RDD
val auctionRDD=sc.textFile("/user/user01/data/ebay.csv")
  .map(_.split(","))

// Map the data to the Auctions class
val auctions=auctionRDD.map(a=>Auction(a(0),
a(1).toFloat, a(2).toFloat, a(3), a(4).toInt,
a(5).toFloat, a(6).toFloat, a(7), a(8).toInt))
```

Creating DataFrames

```
// Convert RDD to DataFrame
val auctionsDF=auctions.toDF()

//To see the data
auctionsDF.show()
```

```
auctionid  bid   bidtime  bidder          bidderrate openbid price item daystolive
8213034705 95.0  2.927373 jake7870        0          95.0    117.5 xbox 3
8213034705 115.0 2.943484 davidbresler2   1          95.0    117.5 xbox 3 …
```

Inspecting Data: Examples

```
//To see the schema
auctionsDF.printSchema()
root
 |-- auctionid: string (nullable = true)
 |-- bid: float (nullable = false)
 |-- bidtime: float (nullable = false)
 |-- bidder: string (nullable = true)
 |-- bidderrate: integer (nullable = true)
 |-- openbid: float (nullable = false)
 |-- price: float (nullable = false)
 |-- item: string (nullable = true)
 |-- daystolive: integer (nullable = true)
```

Inspecting Data: Examples

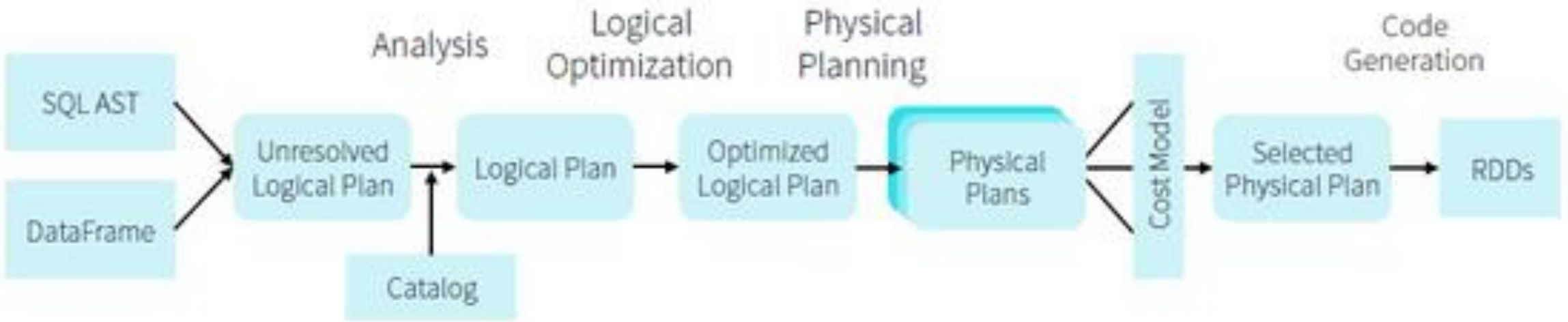
```
//Total number of bids
val totbids=auctionsDF.count()

//How many bids per item?
auction.groupBy("auctionid", "item").count.show
```

```
auctionid  item    count
3016429446 palm    10
8211851222 xbox    28
3014480955 palm    12
8214279576 xbox    4
3014844871 palm    18
3014012355 palm    35
1641457876 cartier 2
```

Creating DataFrames

```
// Register the DF as a table
auctionsDF.registerTempTable(“auctionsDF”)

val results =sqlContext.sql("SELECT auctionid, MAX(price)
as maxprice FROM auction  GROUP BY item,auctionid")
results.show()
auctionid  maxprice
3019326300 207.5
8213060420 120.0 . . .
```

The physical plan for DataFrames



DataFrame Excecution plan

```
// Print the physical plan to the console
auction.select("auctionid").distinct.explain()

== Physical Plan ==
Distinct false
Exchange (HashPartitioning [auctionid#0], 200)
 Distinct true
 Project [auctionid#0]
   PhysicalRDD
       [auctionid#0,bid#1,bidtime#2,bidder#3,
       bidderrate#4,openbid#5,price#6,item#7,daystolive#8],
       MapPartitionsRDD[11]  at mapPartitions at
       ExistingRDD.scala:37
```

MapR Blog: Using Apache Spark DataFrames for Processing of Tabular Data

- https://www.mapr.com/blog/using-apache-spark-dataframes-processing-tabular-data

Using Apache Spark DataFrames for Processing of Tabular Data

June 24, 2015
Carol McDonald

This post will help you get started using Apache Spark DataFrames with Scala on the MapR Sandbox. The new Spark DataFrames API is designed to make big data processing on tabular data easier. A Spark DataFrame is a distributed collection of data organized into named columns that provides operations to filter, group, or compute aggregates, and can be used with Spark SQL. DataFrames can be constructed from structured data files, existing RDDs, tables in Hive, or

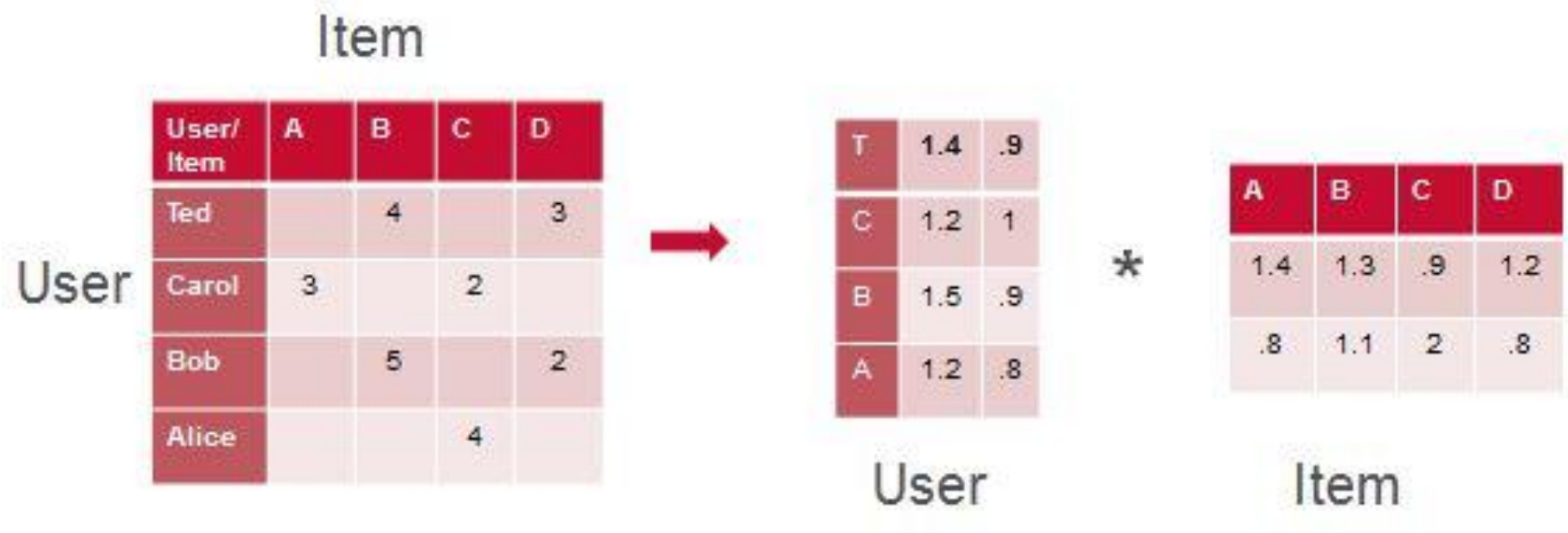
Machine Learning

Collaborative Filtering with Spark

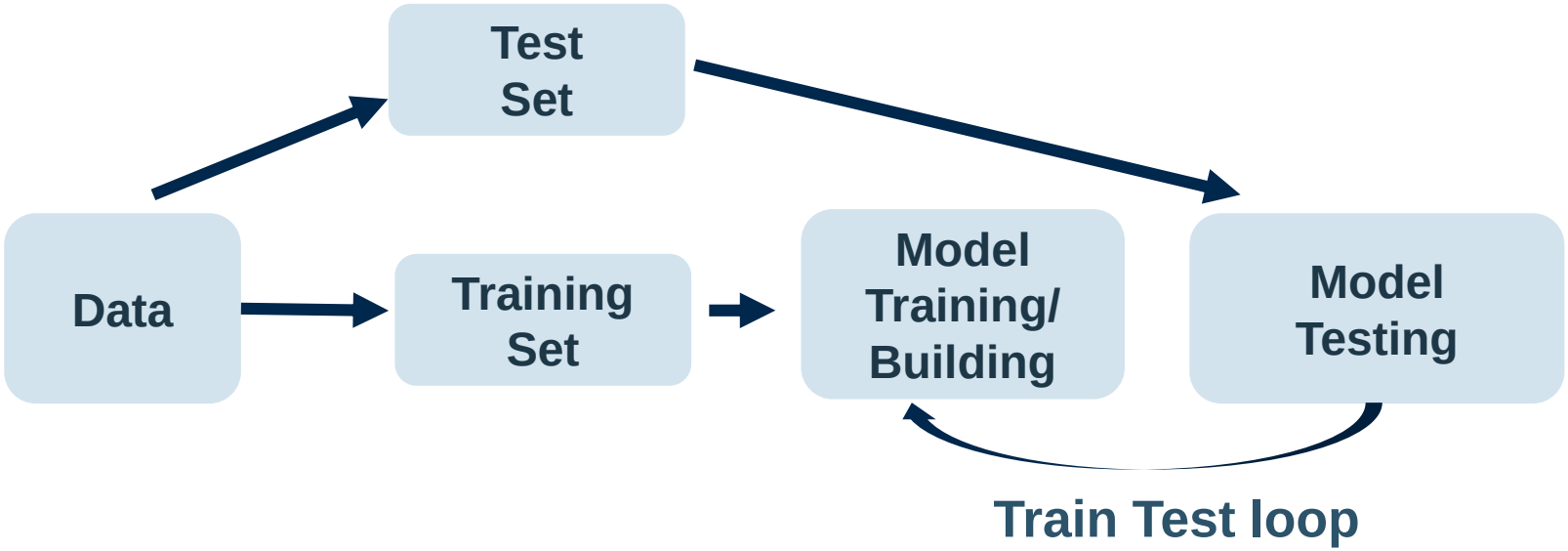
- Recommend Items
  - (filtering)
- Based on User preferences data
  - (collaborative)

Alternating Least Squares

- approximates sparse user item rating matrix
- as product of two dense matrices, User and Item factor matrices



ML Cross Validation Process



Users Data

```
[user01@maprdemo moviemed]$ head users.dat
1::F::1::10::48067
2::M::56::16::70072
3::M::25::15::55117
4::M::45::7::02460
5::M::25::20::55455
6::F::50::9::55117
7::M::35::1::06810
8::M::25::12::11413
9::M::25::17::61614
10::F::35::1::95370
[user01@maprdemo moviemed]$
```

Parse Input

```
case class User(userId: Int, gender: String, age: Int,
occupation: Int, zip: String)
// parse  User
def parseUser(str: String): User = {
      val fields = str.split("::")
      User(fields(0).toInt, fields(1).toString,
       fields(2).toInt,fields(3).toInt, fields(4).toString)
 }
// load the data into DataFrames
val usersDF = sc.textFile("./users.dat")
                .map(parseUser).toDF()
```

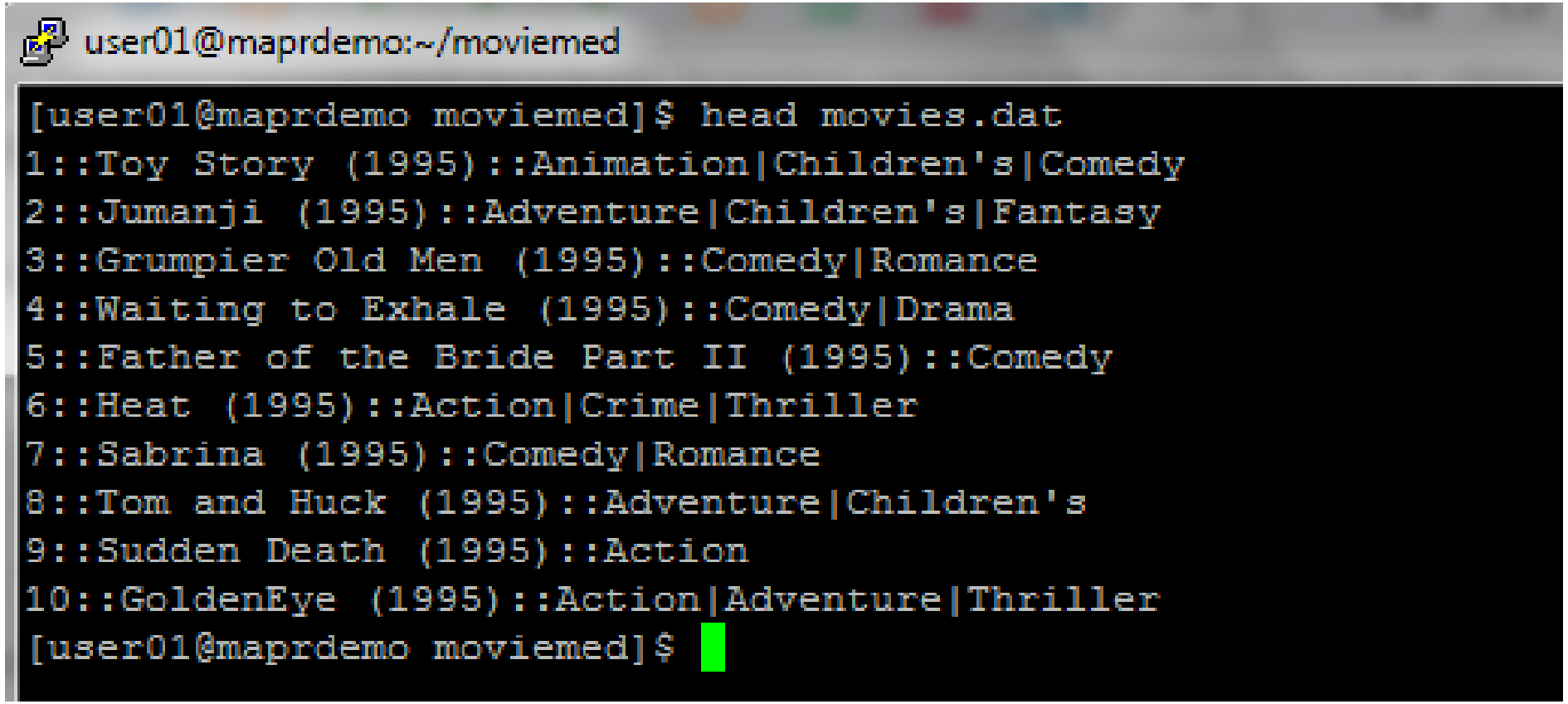
Movie Data

```
[user01@maprdemo moviemed]$ head movies.dat
1::Toy Story (1995)::Animation|Children's|Comedy
2::Jumanji (1995)::Adventure|Children's|Fantasy
3::Grumpier Old Men (1995)::Comedy|Romance
4::Waiting to Exhale (1995)::Comedy|Drama
5::Father of the Bride Part II (1995)::Comedy
6::Heat (1995)::Action|Crime|Thriller
7::Sabrina (1995)::Comedy|Romance
8::Tom and Huck (1995)::Adventure|Children's
9::Sudden Death (1995)::Action
10::GoldenEye (1995)::Action|Adventure|Thriller
```

Parse Input

```
case class Movie(movieId: Int, title: String, genres:
Seq[String])
// parse  Movie
def parseMovie(str: String): Movie = {
      val fields = str.split("::")
      Movie(fields(0).toInt, fields(1))
 }
// load the data into DataFrames
val moviesDF = sc.textFile("./movies.dat")
                 .map(parseMovie).toDF()
```

Movie Ratings Data



Parse Input

```
// parse input UserID::MovieID::Rating
def parseRating(str: String): Rating= {
      val fields = str.split("::")
      Rating(fields(0).toInt, fields(1).toInt,
     fields(2).toDouble)
}

// create an RDD of Ratings objects
val ratingsRDD = ratingText.map(parseRating).cache()
```

Create Model

```
// Randomly split ratings RDD into training data RDD (80%)
and test data RDD (20%)
val splits = ratingsRDD.randomSplit(Array(0.8, 0.2), 0L)

val trainingRatingsRDD = splits(0).cache()
val testRatingsRDD = splits(1).cache()

 // build a ALS user product matrix model with rank=20,
iterations=10
val model = (new
ALS().setRank(20).setIterations(10).run(trainingRatingsRDD))
```

Get predictions

```
// Get the top 4 movie predictions for user 4169
val topRecsForUser = model.recommendProducts(4169, 5)

// get predicted ratings to compare to test ratings
val predictionsForTestRDD = model.predict(testUserProductRDD)
```

Test Model

```
// prepare  predictions for comparison
val predictionsKeyedByUserProductRDD = predictionsForTestRDD.map{
  case Rating(user, product, rating) => ((user, product), rating)
}
// prepare  test for comparison
val testKeyedByUserProductRDD = testRatingsRDD.map{
  case Rating(user, product, rating) => ((user, product), rating)
}
//Join the  test with  predictions
val testAndPredictionsJoinedRDD =
testKeyedByUserProductRDD.join(predictionsKeyedByUserProductRDD)
```

Test Model

```
val falsePositives =(testAndPredictionsJoinedRDD.filter{
  case ((user, product), (ratingT, ratingP)) =>
       (ratingT <= 1 && ratingP >=4)
  })
falsePositives.take(2)
```

```
Array[((Int, Int), (Double, Double))] =
((3842,2858),(1.0,4.106488210964762)),
((6031,3194),(1.0,4.790778049100913))
```

Test Model Mean Absolute Error

```
//Evaluate the model using Mean Absolute Error (MAE) between
test and predictions
val meanAbsoluteError = testAndPredictionsJoinedRDD.map {
  case ((user, product), (testRating, predRating)) =>
    val err = (testRating - predRating)
    Math.abs(err)
}.mean()
meanAbsoluteError: Double = 0.7244940545944053
```

Soon to Come

- Spark On Demand Training
  - https://www.mapr.com/services/mapr-academy/
- Blogs and Tutorials:
  - Movie Recommendations with Collaborative Filtering
  - Spark Streaming

Machine Learning Blog

- https://www.mapr.com/blog/parallel-and-iterative-processing-machine-learning-recommendations-spark

Parallel and Iterative Processing for Machine Learning Recommendations with Spark

Streaming

What is Streaming?

Data Stream:

- Unbounded sequence of data arriving continuously

Stream processing:

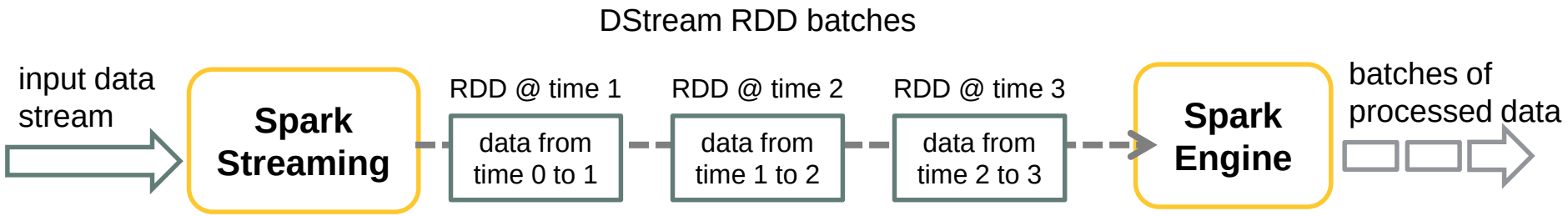
- Low latency processing, querying, and analyzing of real time streaming data

What is Spark Streaming?

- enables scalable, high-throughput, fault-tolerant stream processing of live data
- extension of the core Spark AP

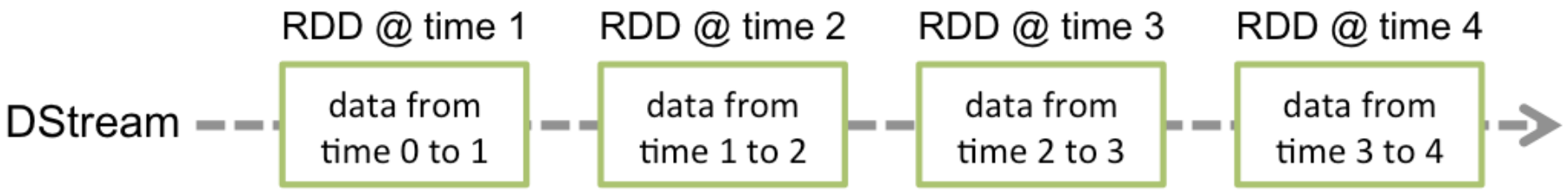
Streaming Architecture

- Divide data stream into batches of X seconds
- Process each batch of data as an RDDs using RDD transformations, which create new RDDs
- processed results are pushed out in batches

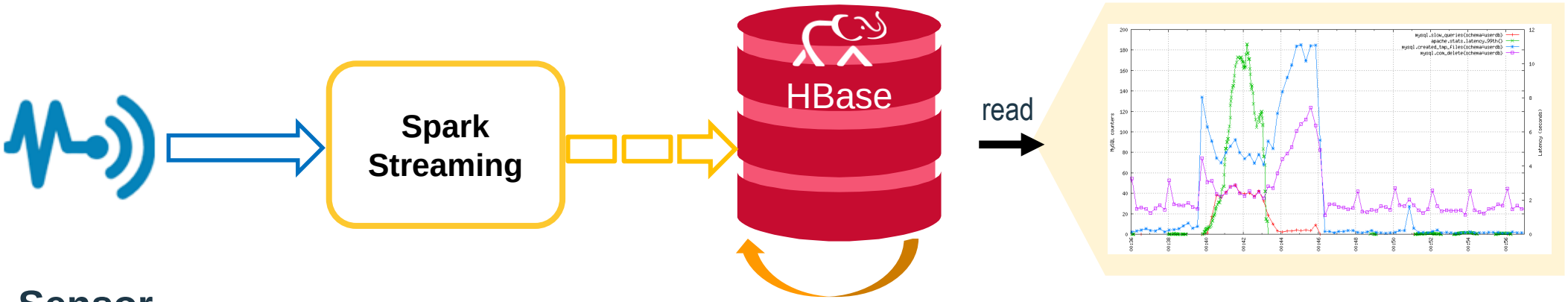


Spark Streaming Terminology

- Input Streams
  - Stream of raw data from input sources
- Discretized Streams (DStreams)
  - Stream from input sources or generated after transformation
  - Continuous series of RDDs



Use Case: Oil Rig Sensor, Time Series Data



Sensor
Time stamped data

Spark Processing

Data for
real-time monitoring

Function to convert Line of CSV data to Sensor Object

sensordata.csv

```
1  ResourceID,Date,Time,HZ,Displace,Flow,SedimentPPM,PressureLbs,ChlorinePPM
2  COHUTTA,3/10/14,1:01,10.27,1.73,881,1.56,85,1.94
3  COHUTTA,3/10/14,1:02,9.67,1.731,882,0.52,87,1.79
4  COHUTTA,3/10/14,1:03,10.47,1.732,882,1.7,92,0.66
```

```
case class Sensor(resid: String, date: String, time: String,
   hz: Double, disp: Double, flo: Double, sedPPM: Double,
   psi: Double, chlPPM: Double)

def parseSensor(str: String): Sensor = {
   val p = str.split(",")
   Sensor(p(0), p(1), p(2), p(3).toDouble, p(4).toDouble, p(5).toDouble,
       p(6).toDouble, p(7).toDouble, p(8).toDouble)
}
```

Schema

- All events stored, data CF could be set to expire data
- Filtered alerts put in alerts CF
- Daily summaries put in Stats CF

Row key	CF data			CF alerts		CF stats		
	hz	…	psi	psi	…	hz_avg	…	psi_min
COHUTTA_3/10/14_1:01	10.37		84	0				
COHUTTA_3/10/14						10		0

Configure to Write to HBase

```
// set JobConfiguration variables for writing to HBase
val jobConfig: JobConf = new JobConf(conf, this.getClass)
jobConfig.setOutputFormat(classOf[TableOutputFormat])
jobConfig.set(TableOutputFormat.OUTPUT_TABLE, tableName)
```

Function to convert Sensor data to HBase Put object (row)

```
//  Convert a row of sensor object data to an HBase put object
def convertToPut(sensor: Sensor): (ImmutableBytesWritable, Put) = {
    val dateTime = sensor.date + " " + sensor.time
    // create a composite row key: sensorid_date time
    val rowkey = sensor.resid + "_" + dateTime
    val put = new Put(Bytes.toBytes(rowkey))
    // add to column family data, column  data values to put object
    put.add(cfDataBytes, Bytes.toBytes("hz"), Bytes.toBytes(sensor.hz))
    put.add(cfDataBytes, Bytes.toBytes("disp"), Bytes.toBytes(sensor.disp))
    put.add(cfDataBytes, Bytes.toBytes("flo"), Bytes.toBytes(sensor.flo))
    put.add(cfDataBytes, Bytes.toBytes("sedPPM"), Bytes.toBytes(sensor.sedPPM))
    put.add(cfDataBytes, Bytes.toBytes("psi"), Bytes.toBytes(sensor.psi))
    put.add(cfDataBytes, Bytes.toBytes("chlPPM"), Bytes.toBytes(sensor.chlPPM))
    return (new ImmutableBytesWritable(Bytes.toBytes(rowkey)), put)
}
```

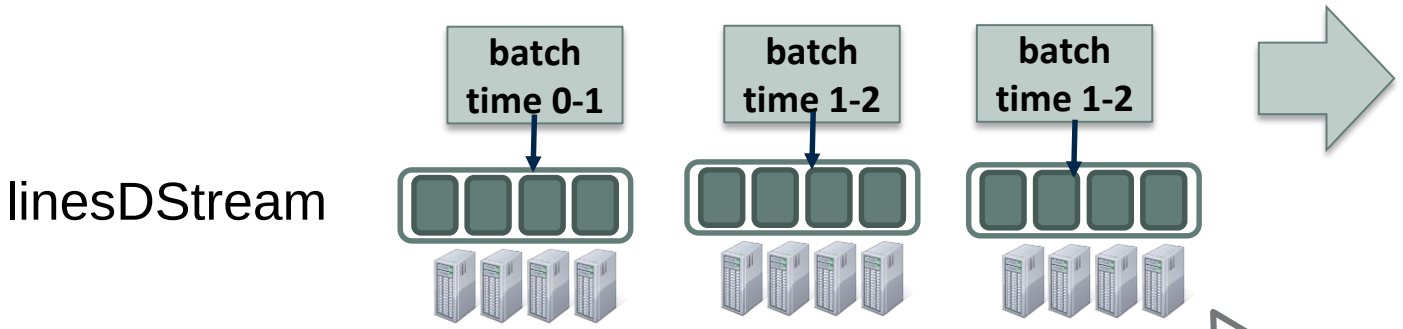
Row key	CF data		
	hz	**…**	**psi**
COHUTTA_3/10/14_1:01	**10.37**		**84**

Get a DStream

```
// create a StreamingContext, the main entry point for all streaming functionality
val ssc = new StreamingContext(sparkConf, Seconds(2))

// create a DStream that represents streaming data from a directory  source
val linesDStream = ssc.textFileStream("/user/user01/stream")
```

Process DStream

```
val linesDStream = ssc.textFileStream("/user/user01/stream")
```



DStream: a sequence of RDDs representing a stream of data

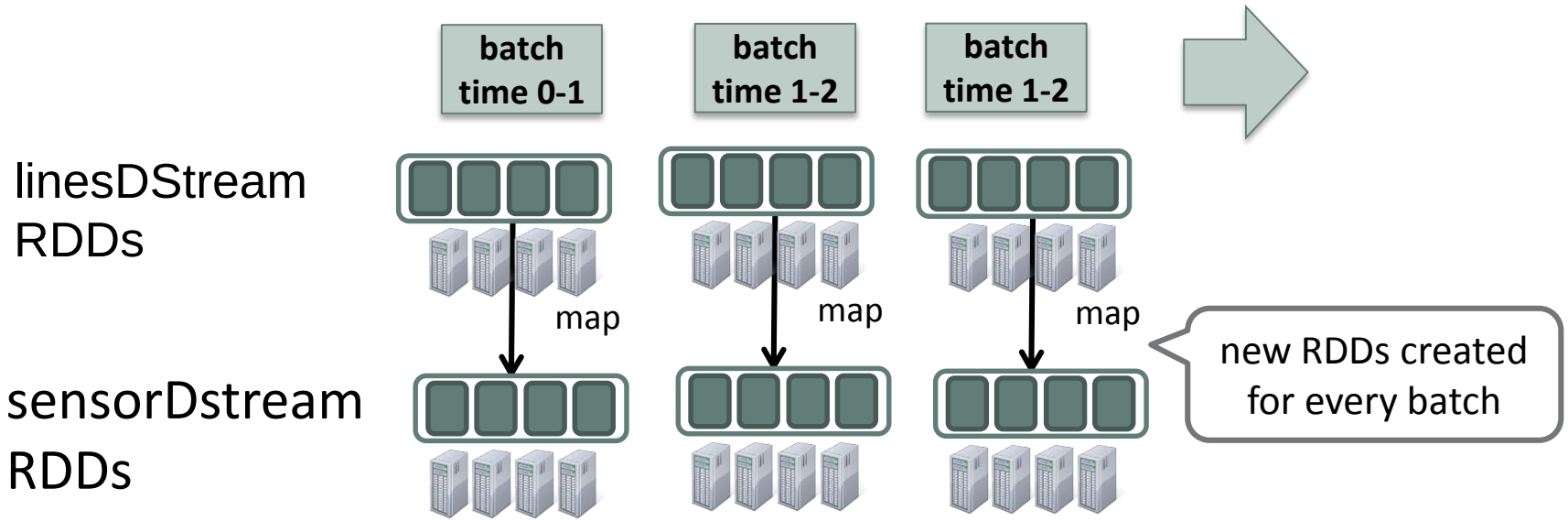
stored in memory as an RDD

Process DStream

```
// parse each line of data in linesDStream into sensor objects
val sensorDStream = linesDStream.map(Sensor.parseSensor)
```

Process DStream

```
val linesDStream = ssc.textFileStream("/user/user01/stream")
val sensorDStream = linesDStream.map(Sensor.parseSensor)
```

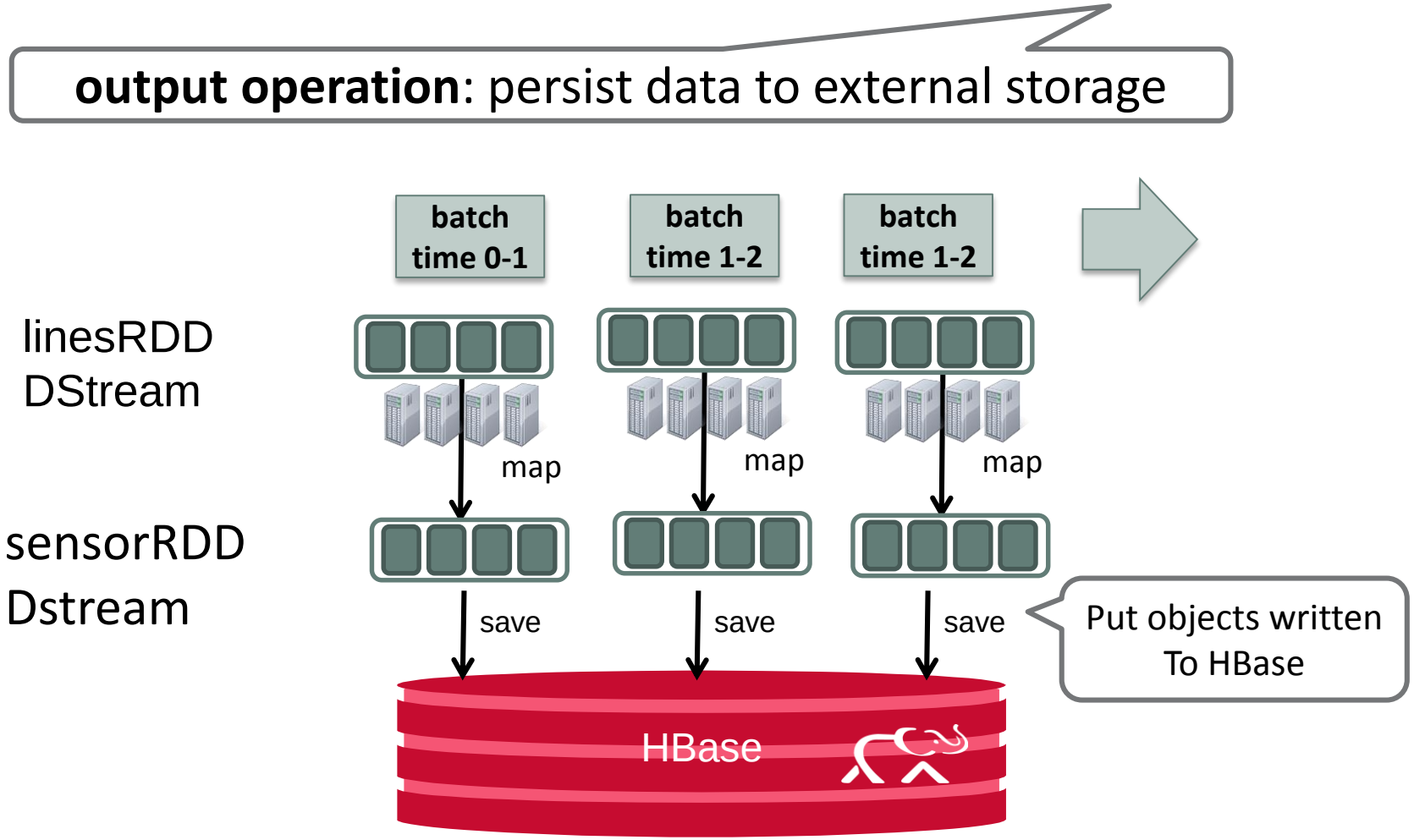


Process DStream

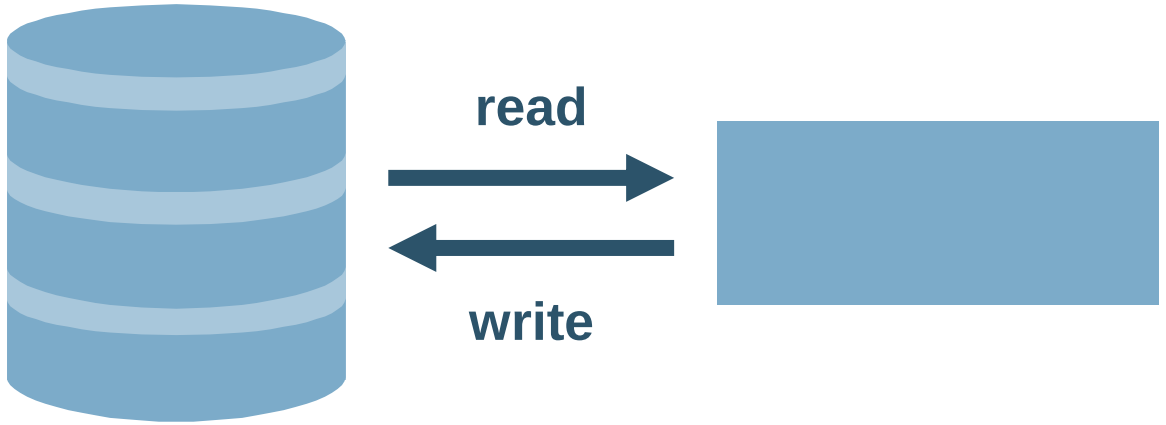
```
// for Each RDD parse into a sensor object filter
sensorDStream.foreachRDD { rdd =>
   // filter sensor data for low psi
   val alertRDD = sensorRDD.filter(sensor => sensor.psi < 5.0)

   // convert to put and write to HBase
   rdd.map(Sensor.convertToPut).saveAsHadoopDataset(jobConfig)

   // convert alert to put object write to HBase alerts
   rdd.map(Sensor.convertToPutAlert)
      .saveAsHadoopDataset(jobConfig)
}
```

Save to HBase

```
rdd.map(Sensor.convertToPut).saveAsHadoopDataset(jobConfig
```



Using HBase as a Source and Sink

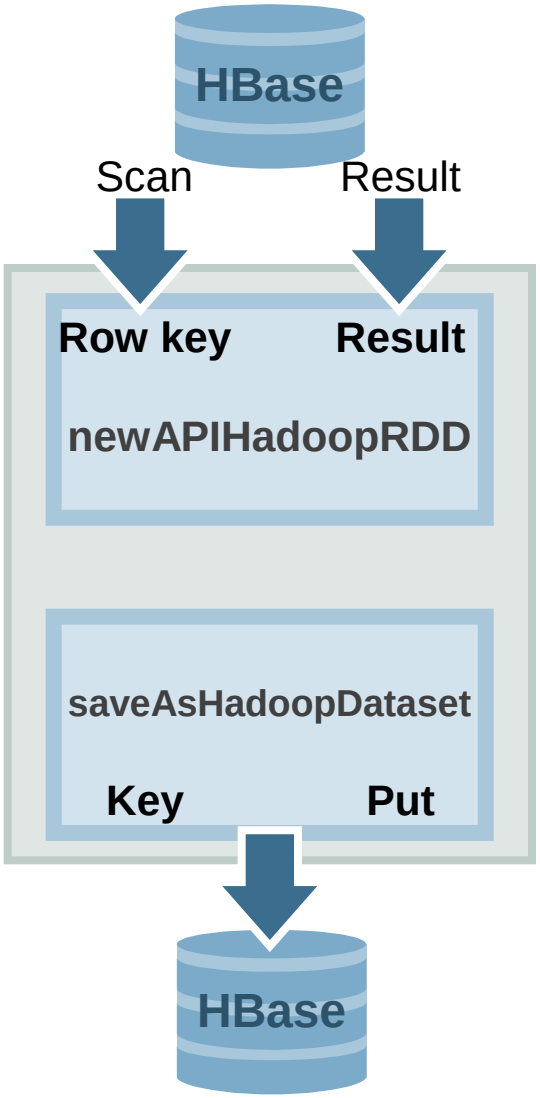


HBase database

Spark application

EXAMPLE: calculate and store summaries, Pre-Computed, Materialized View

HBase Read and Write




```
val hBaseRDD = sc.newAPIHadoopRDD(
      conf,classOf[TableInputFormat],
      classOf[org.apache.hadoop.hbase.io.ImmutableBytesWritable],
      classOf[org.apache.hadoop.hbase.client.Result])
```

```
keyStatsRDD.map { case (k, v) => convertToPut(k, v)
}.saveAsHadoopDataset(jobConfig)
```

Read HBase

```
// Load an RDD of (rowkey, Result) tuples from HBase table
val hBaseRDD = sc.newAPIHadoopRDD(conf, classOf[TableInputFormat],
   classOf[org.apache.hadoop.hbase.io.ImmutableBytesWritable],
   classOf[org.apache.hadoop.hbase.client.Result])
// get Result
val resultRDD = hBaseRDD.map(tuple => tuple._2)
// transform into an RDD of (RowKey, ColumnValue)s
val keyValueRDD = resultRDD.map(
       result => (Bytes.toString(result.getRow()).split(" ")(0),
       Bytes.toDouble(result.value)))

// group by rowkey , get statistics for column value
val keyStatsRDD = keyValueRDD.groupByKey().mapValues(list =>
     StatCounter(list))
```

Write HBase

```
// save to HBase table CF data
 val jobConfig: JobConf = new JobConf(conf, this.getClass)
    jobConfig.setOutputFormat(classOf[TableOutputFormat])
    jobConfig.set(TableOutputFormat.OUTPUT_TABLE, tableName)


    // convert rowkey, psi stats , write to hbase table stats column family
    keyStatsRDD.map {
        case (k, v) => convertToPut(k, v) })
        .saveAsHadoopDataset(jobConfig)
```

Examples and Resources

Find my presentation and other related resources here:

http://events.mapr.com/JacksonvilleJava

(you can find this link in the event's page at meetup.com)

Today's Presentation

Free On-Demand Training

Whiteboard & demo videos

Free eBooks

Free Hadoop Sandbox

And more…

MAPR®

Spark on MapR

- Certified Spark Distribution
- Fully supported and packaged by MapR in partnership with Databricks
  - mapr-spark package with Spark, Shark, Spark Streaming today
  - Spark-python, GraphX and MLLib soon
- YARN integration
  - Spark can then allocate resources from cluster when needed

References

- Spark web site: http://spark.apache.org/
- https://databricks.com/
- Spark on MapR:
  - http://www.mapr.com/products/apache-spark
- Spark SQL and DataFrame Guide
- Apache Spark vs. MapReduce – Whiteboard Walkthrough
- Learning Spark - O'Reilly Book
- Apache Spark

Q&A

Engage with us!

@mapr   maprtech

mapr-technologies   MapR

kbotzum@mapr.com   maprtech