Big Data Analysis - Part III: Practical Machine Learning with MLlib and GraphX

Zhao Zhang
Data Mining and Statistics Group
Texas Advanced Computing Center
zzhang@tacc.utexas.edu
May 4, 2017

TACC

Overview

- Machine learning concept
- MLlib
- GraphX

Machine Learning Concept

- Supervised Learning
- Unsupervised Learning
- Others

Supervised Learning

- Supervised learning is the machine learning task of inferring a function from labeled training data. — Wiki
  - Linear Regression
  - Classification
    - Logistic Regression
    - Support Vector Machine (SVM)

Unsupervised Learning

- Unsupervised machine learning is the machine learning task of inferring a function to describe hidden structure from "unlabeled" data. — Wiki
  - A lower dimension representation (e.g., Principle Component Analysis)
  - A sparse representation (e.g., K-Means, Mixture Models)
  - An independent representation (e.g., PCA)

Goals

- Understand the basics of machine learning algorithms
- Use MLlib and GraphX to train models with real datasets

TACC

Dataset

- A small dataset of San Francisco and New York City real estate data
- 491 records in total
- 7 columns (in_sf, beds, bath, price, year_built, sqft, elevation)
- Scaled features

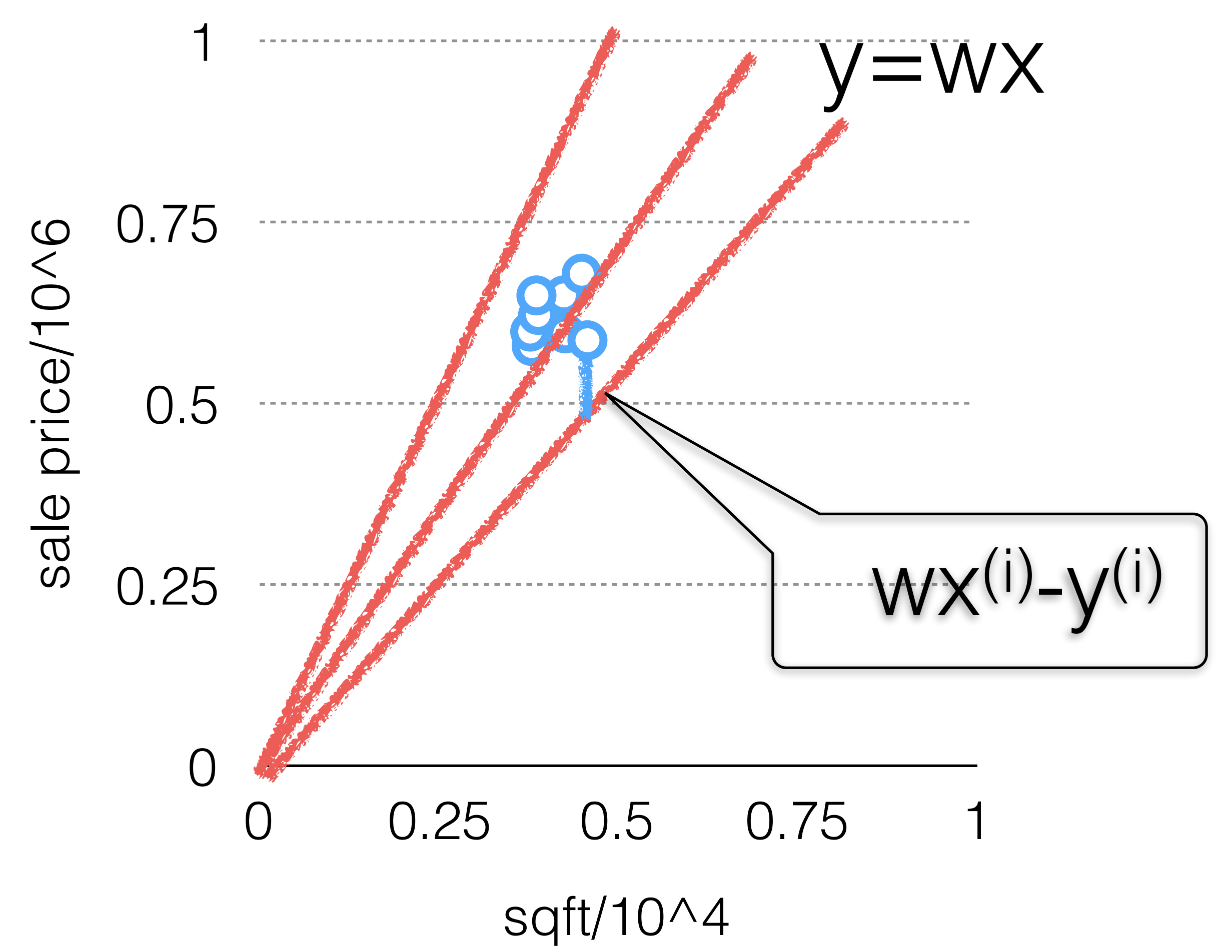
Keep in Mind

- The algorithm and code examples are for training purposes
- They do not necessarily reflect the performance of the algorithm

Linear Regression

- Predicting the house price using sqft
- Train a function y=wx to minimize $\sum(wx^{(i)}-y^{(i)})^2$

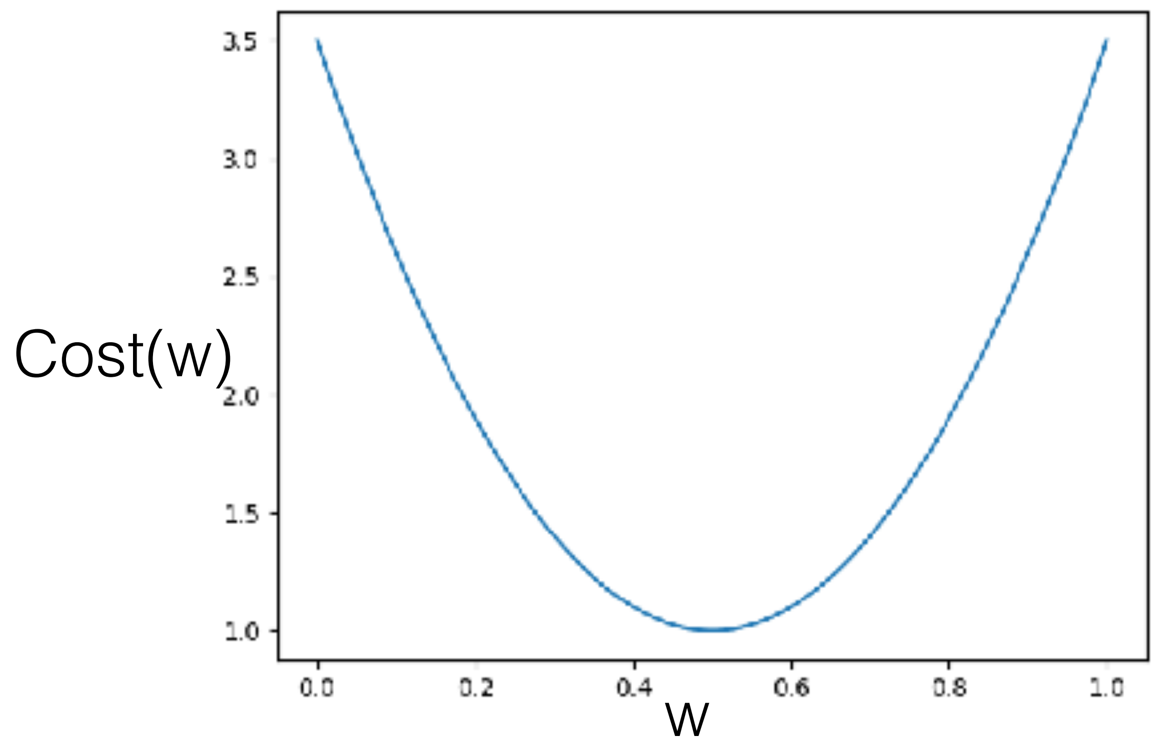
ID	$sqft/10^4$	$price/10^6$
1	0.3801	0.58
2	0.4271	0.5975
3	0.4580	0.588
4	0.3780	0.6
5	0.3890	0.623
6	0.4250	0.65
7	0.4500	0.68
8	0.3867	0.65
9	0.3815	?



Gradient Descent

- $\text{Cost}(w) = \sum(wx^{(i)}-y^{(i)})^2$

  $= \mathbf{w^2}\sum x^{(i)2} - 2\mathbf{w}y^{(i)}\sum x^{(i)}+\sum y^{(i)2}$



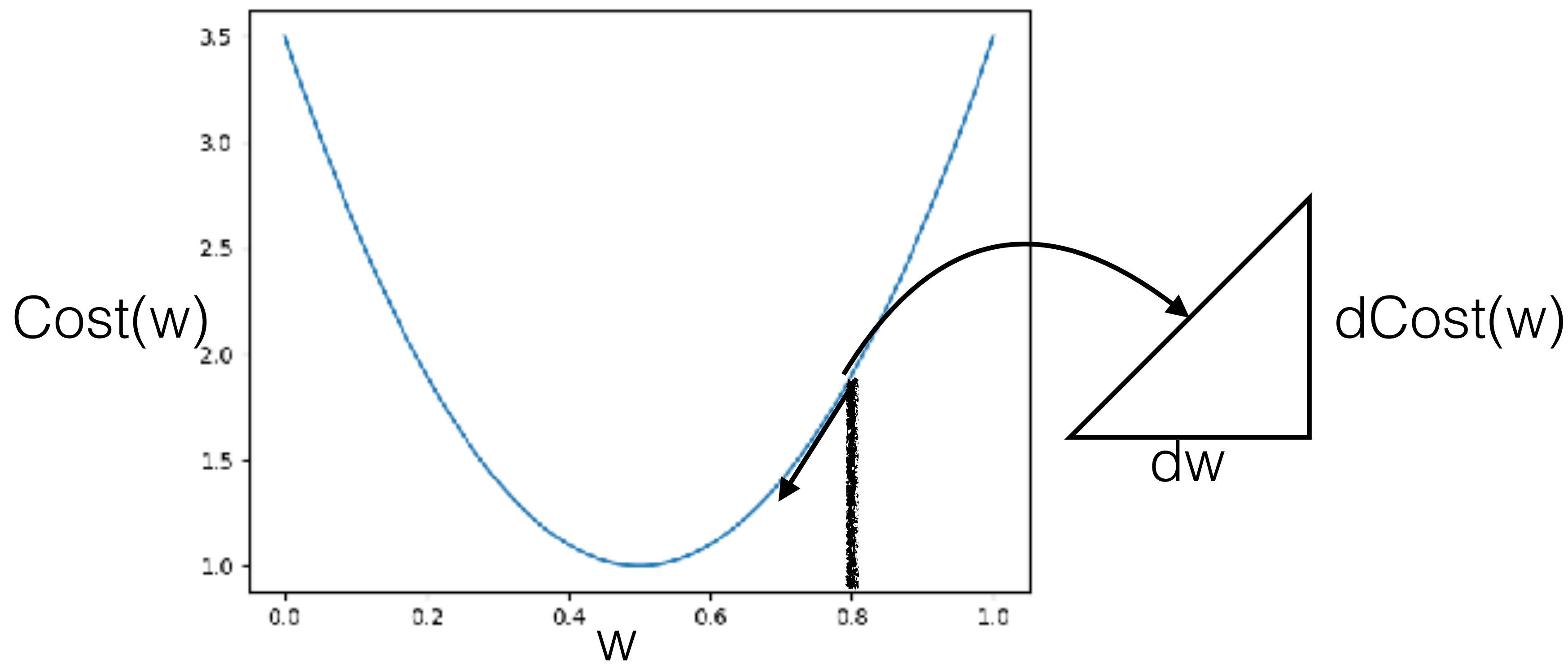
- $d\text{Cost}(w)/dw = 2w\sum x^{(i)2} - 2y^{(i)}\sum x^{(i)}$
- Cost(w) gets its minimum when $d\text{Cost}(w)/dw = 0$

TACC

Gradient Descent

- $Cost(w) = \sum(wx^{(i)}-y^{(i)})^2$

  $= \mathbf{w^2}\sum x^{(i)2} - 2\mathbf{w}y^{(i)}\sum x^{(i)}+\sum y^{(i)2}$



- Randomly start at w=0.8, calculate the derivative dCost(w)/dw at w=0.8
- Update $w = w-\alpha dCost(w)/dw$ until dCost(w)/dw converges to 0

Using MLlib

```
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LinearRegressionWithSGD

val lines = sc.textFile("/tmp/data/scaled-sf-ny-housing-train.csv")
val data = lines.map(l => {
  val w = l.split(",")
  LabeledPoint(w(3).toDouble, Vectors.dense(w(5).toDouble))
})
val model = LinearRegressionWithSGD.train(data,100)

model.weights

val trainError = lines.map(l => {
  val w = l.split(",")
  model.predict(Vectors.dense(w(5).toDouble))-w(3).toDouble
})
val mseTrain = trainError.map(x=>x*x).reduce(_+_)/400
> mseTrain: Double = 0.06472201882476669

val tlines = sc.textFile("/tmp/data/scaled-sf-ny-housing-test.csv")
val testError = tlines.map(l => {
  val w = l.split(",")
  model.predict(Vectors.dense(w(5).toDouble))-w(3).toDouble
})
val mseTest = testError.map(x=>x*x).reduce(_+_)/92
> mseTest: Double = 0.05897075938607083
```

TACC

Cost Function

- Regularization
  - $Cost(w) = \sum(wx^{(i)}-y^{(i)})^2 + \lambda/2 * w^2$
- Other options
  - Maximum Likelihood
  - KL divergence
  - cross-entropy

Linear Regression

- Extend the single-variant solution to the multi-variant solution
  - x is a vector of features, $x \in R^N$
  - w is a vector of weights, $w \in R^N$
  - Pre-requisite: linear algebra, multi-variant calculus

TACC

Using MLlib

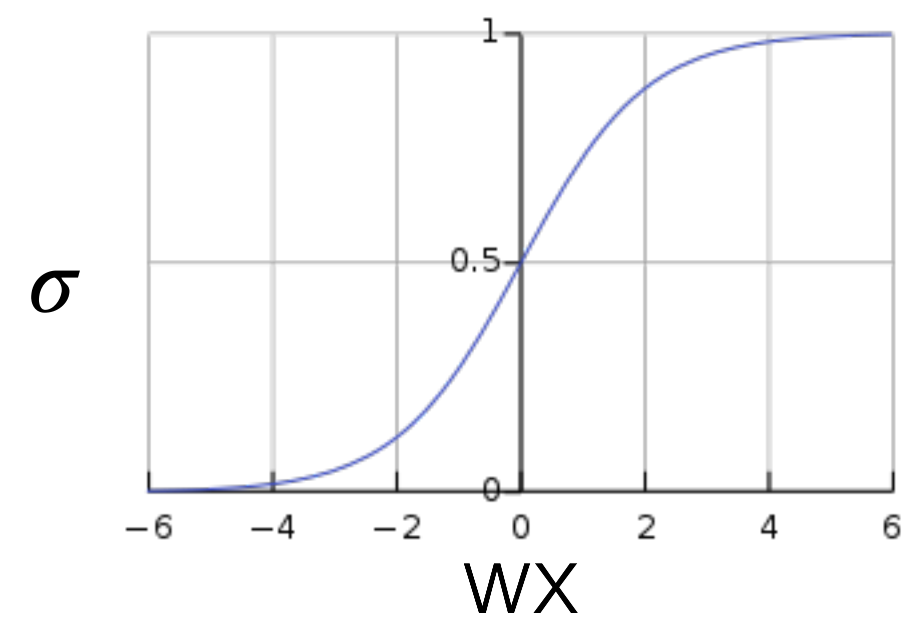
- Predict the house price using sqft, year_built, beds
  - We describe each house with a 3-element vector, e.g., $house^{(1)}$=(0.0769, 0.642, 0.1)
  - $price^{(1)}$ = (0.0798)

Using MLlib

```
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LinearRegressionWithSGD

val lines = sc.textFile("/tmp/data/scaled-sf-ny-housing-train.csv")
val data = lines.map(l => {
  val w = l.split(",")
  LabeledPoint(w(3).toDouble, Vectors.dense(w(5).toDouble, w(4).toDouble, w(1).toDouble))
})
val model = LinearRegressionWithSGD.train(data,100)

model.weights

val trainError = lines.map(l => {
  val w = l.split(",")
  model.predict(Vectors.dense(w(5).toDouble, w(4).toDouble, w(1).toDouble))-w(3).toDouble
})
val mseTrain = trainError.map(x=>x*x).reduce(_+_)/400
> mseTrain: Double = 0.06222798683227797

val tlines = sc.textFile("/tmp/data/scaled-sf-ny-housing-test.csv")
val testError = tlines.map(l => {
  val w = l.split(",")
  model.predict(Vectors.dense(w(5).toDouble, w(4).toDouble, w(1).toDouble))-w(3).toDouble
})
val mseTest = testError.map(x=>x*x).reduce(_+_)/92
> mseTest: Double = 0.05444971758384607
```

Classification

- We train a classifier to tell if a house is in San Francisco or the New York city
- Intuition — linear regression
  - One class with f(wx) > threshold
  - and the other class with f(wx) < threshold
- Using an outer sigmoid function on wx
  - $\sigma(wx) = 1/1+e^{-wx}$



Using Logistic Regression

```
import org.apache.spark.mllib.classification.{LogisticRegressionModel, LogisticRegressionWithLBFGS}
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.regression.LabeledPoint

val lines = sc.textFile("/tmp/data/scaled-sf-ny-housing-train.csv")
val data = lines.map(l => {
  val w = l.split(",")
  LabeledPoint(w(0).toDouble, Vectors.dense(w(5).toDouble, w(4).toDouble, w(1).toDouble))
})
val model = new LogisticRegressionWithLBFGS().setNumClasses(2).run(data)

model.weights

val trainPrediction = lines.map(l => {
  val w = l.split(",")
  (model.predict(Vectors.dense(w(5).toDouble, w(4).toDouble, w(1).toDouble)), w(0).toDouble)
})

val metrics = new MulticlassMetrics(trainPrediction)
metrics.precision
> res12: Double = 0.6575

val tlines = sc.textFile("/tmp/data/scaled-sf-ny-housing-test.csv")
val testPrediction = tlines.map(l => {
  val w = l.split(",")
  (model.predict(Vectors.dense(w(5).toDouble, w(4).toDouble, w(1).toDouble)), w(0).toDouble)
})

val metrics = new MulticlassMetrics(testPrediction)
metrics.precision
> res12: Double = 0.6956521739130435
```

Using SVM

```
import org.apache.spark.mllib.classification.{SVMModel, SVMWithSGD}
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.regression.LabeledPoint

val lines = sc.textFile("/tmp/data/scaled-sf-ny-housing-train.csv")
val data = lines.map(l => {
  val w = l.split(",")
  LabeledPoint(w(0).toDouble, Vectors.dense(w(5).toDouble, w(4).toDouble, w(1).toDouble))
})
val model = SVMWithSGD.train(data, 1000)

model.weights

val trainPrediction = lines.map(l => {
  val w = l.split(",")
  (model.predict(Vectors.dense(w(5).toDouble, w(4).toDouble, w(1).toDouble)), w(0).toDouble)
})

val metrics = new MulticlassMetrics(trainPrediction)
metrics.precision
> res12: Double = 0.525

val tlines = sc.textFile("/tmp/data/scaled-sf-ny-housing-test.csv")
val testPrediction = tlines.map(l => {
  val w = l.split(",")
  (model.predict(Vectors.dense(w(5).toDouble, w(4).toDouble, w(1).toDouble)), w(0).toDouble)
})

val metrics = new MulticlassMetrics(testPrediction)
metrics.precision
> res12: Double = 0.6304347826086957
```

TACC

Classification

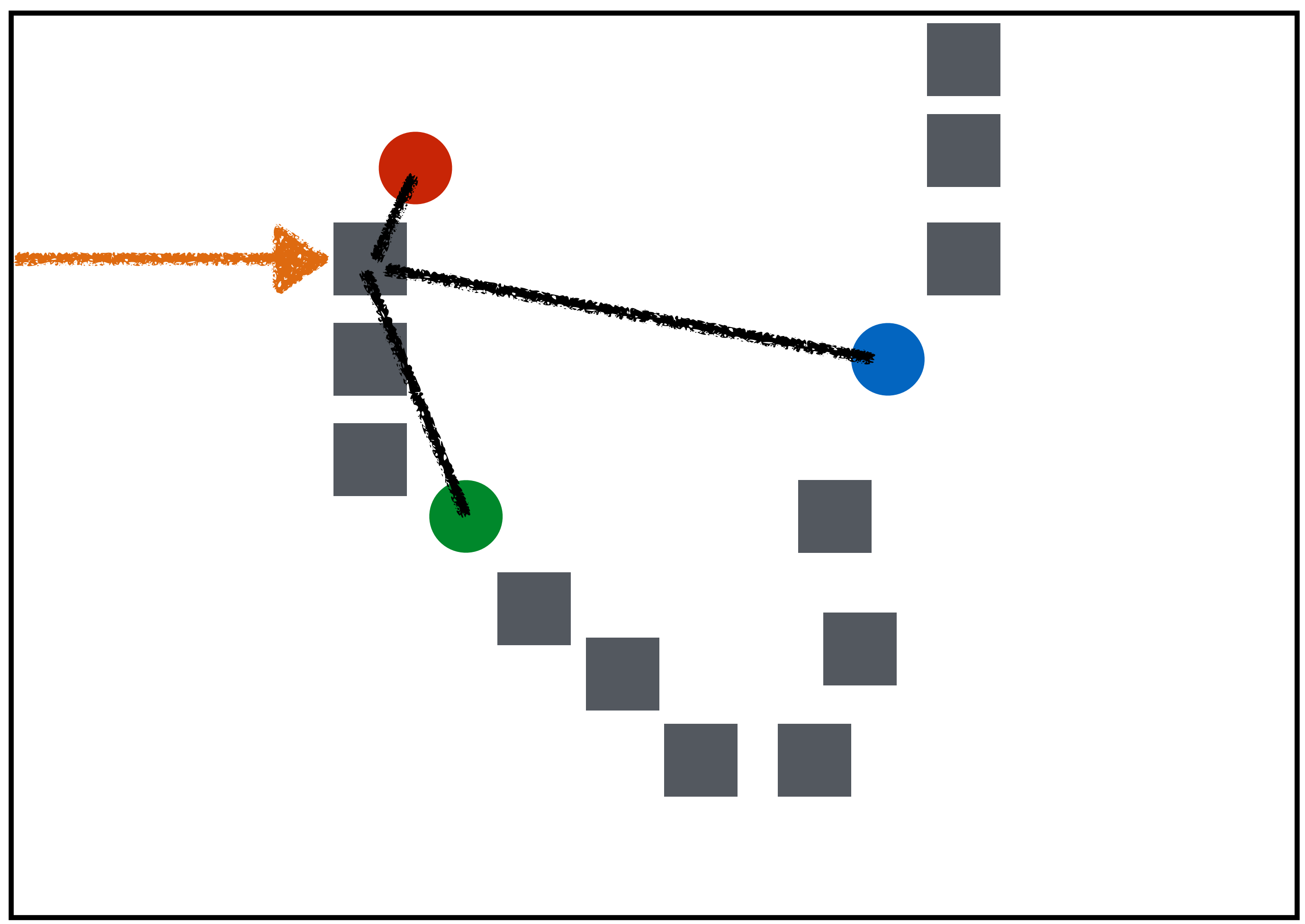
- Result interpretation
  - Random precision = 50%
  - Both linear classifier and Support Vector Machine are limited
  - Support Vector Machine is good for non-linear classification
- Multi-class problem
  - Use multiple classifier with maximum likelihood

Supervised Learning

- Linear regression
- Logistic regression
- How to use MLlib to train the model

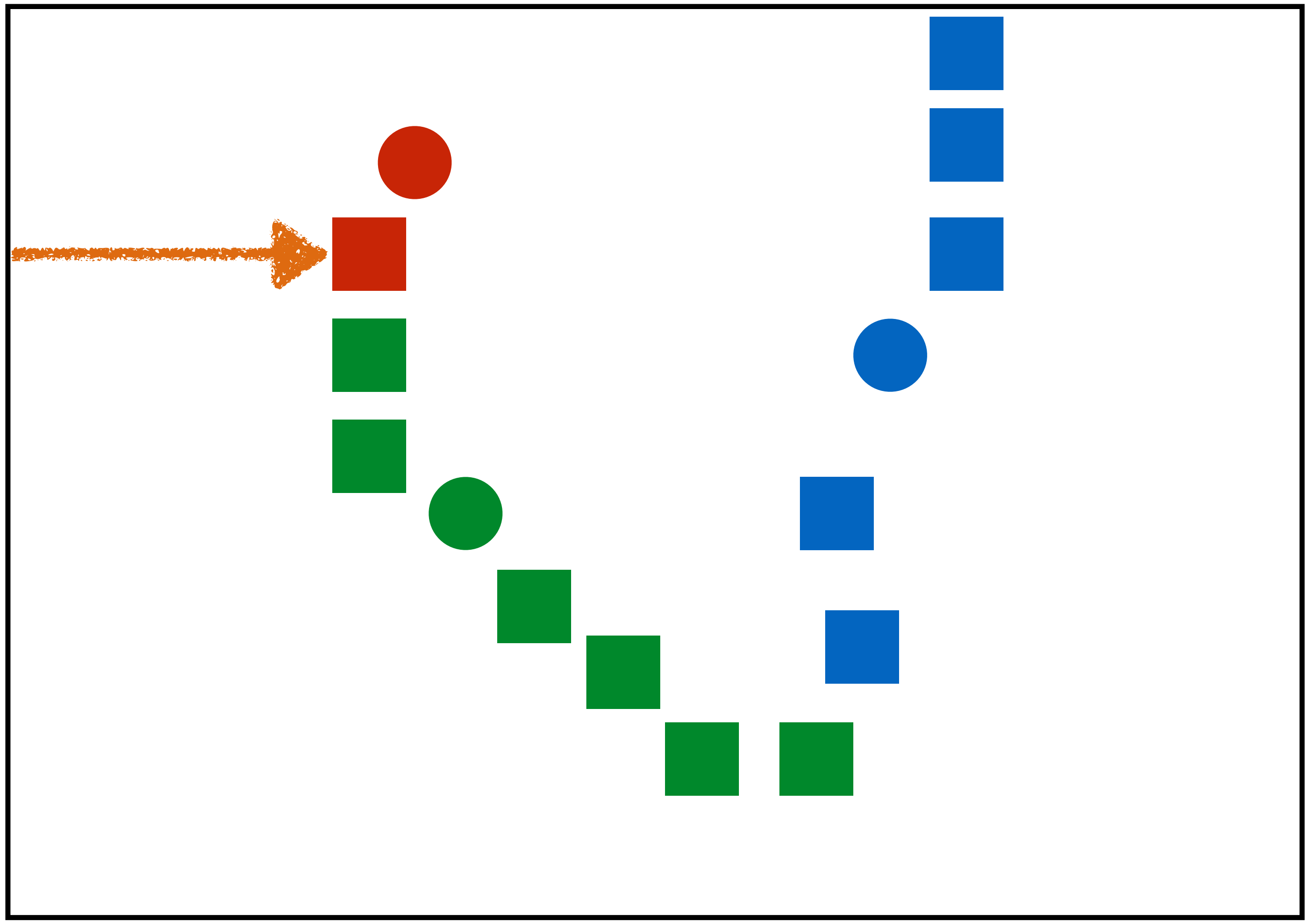
Unsupervised Learning

- k-means clustering partitions n observations into k clusters in which each observation belongs to the cluster with the nearest mean. — Wiki
- In practice, we use an efficient heuristic algorithm



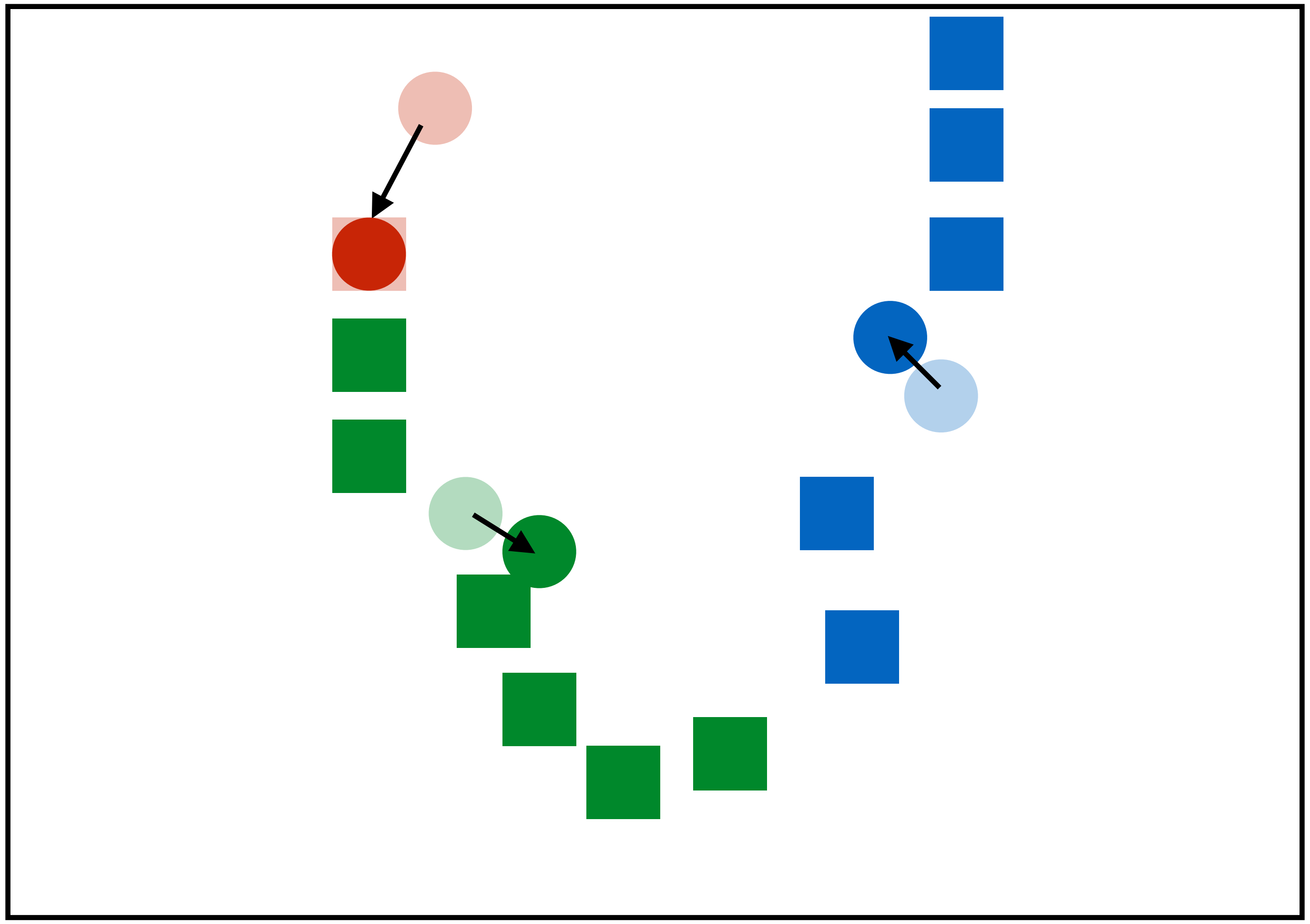
k-means Clustering

- k-means clustering partitions n observations into k clusters in which each observation belongs to the cluster with the nearest mean. — Wiki
- In practice, we use an efficient heuristic algorithm



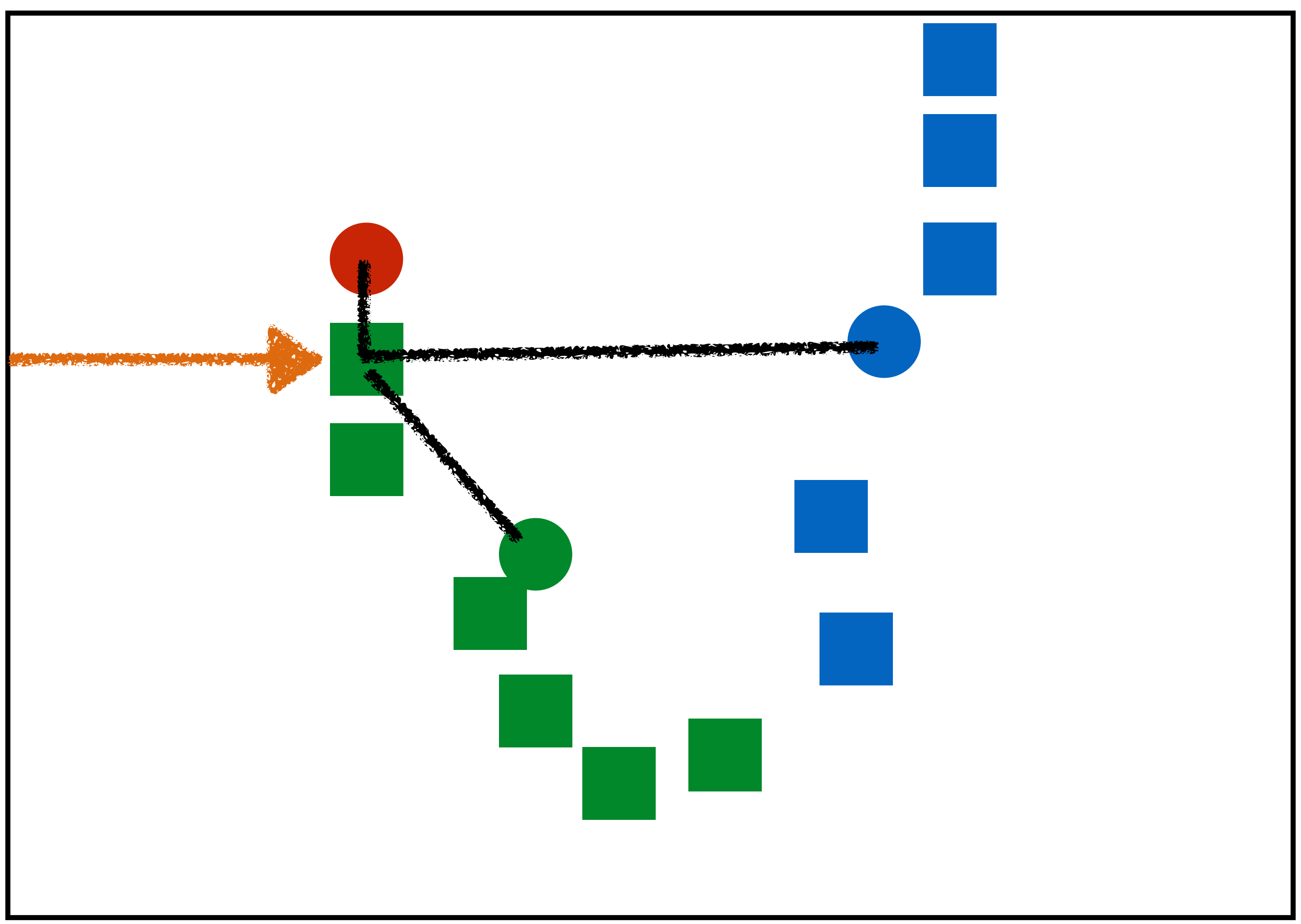
k-means Clustering

- k-means clustering partitions n observations into k clusters in which each observation belongs to the cluster with the nearest mean. — Wiki
- In practice, we use an efficient heuristic algorithm



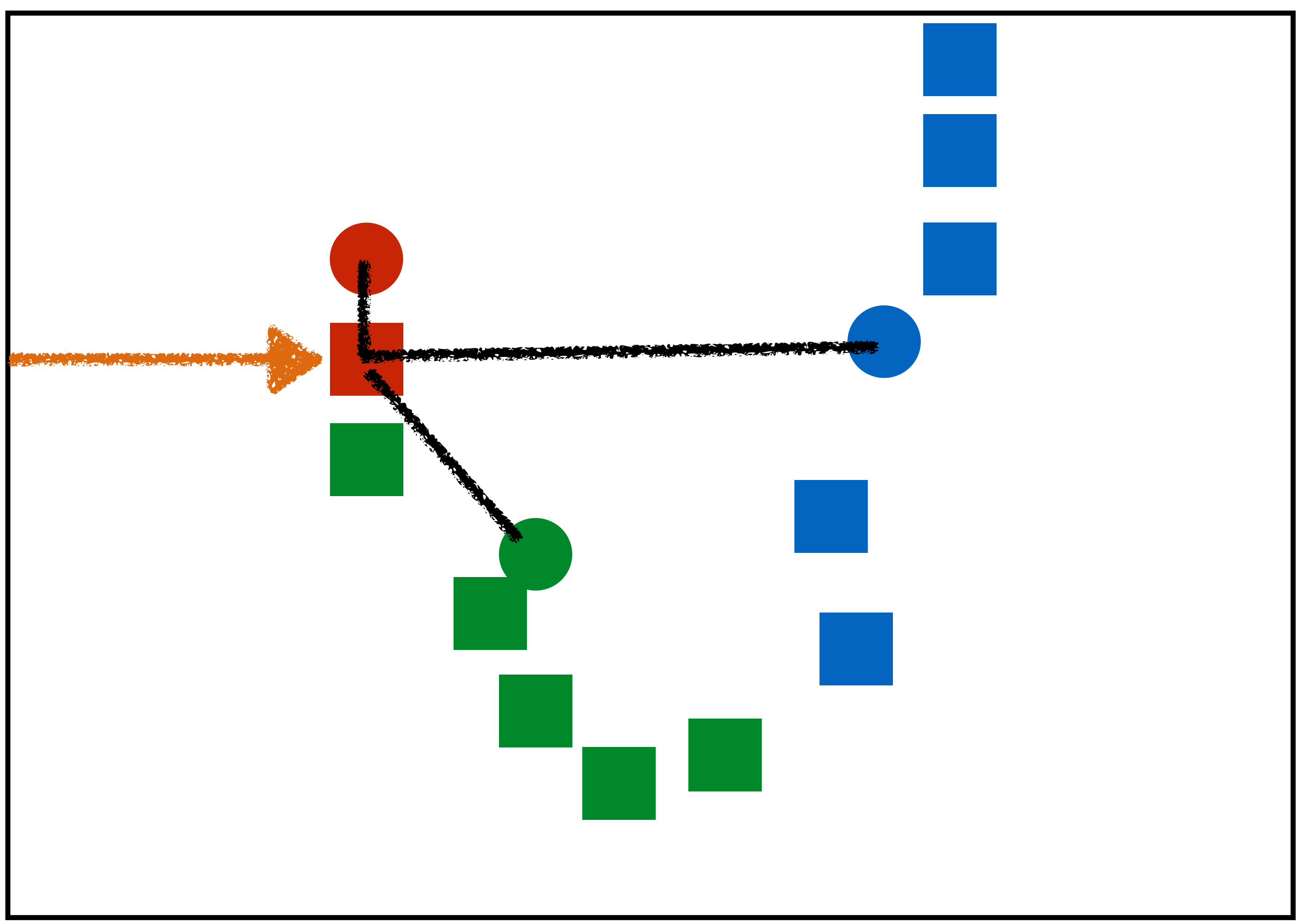
k-means Clustering

- k-means clustering partitions n observations into k clusters in which each observation belongs to the cluster with the nearest mean. — Wiki
- In practice, we use an efficient heuristic algorithm



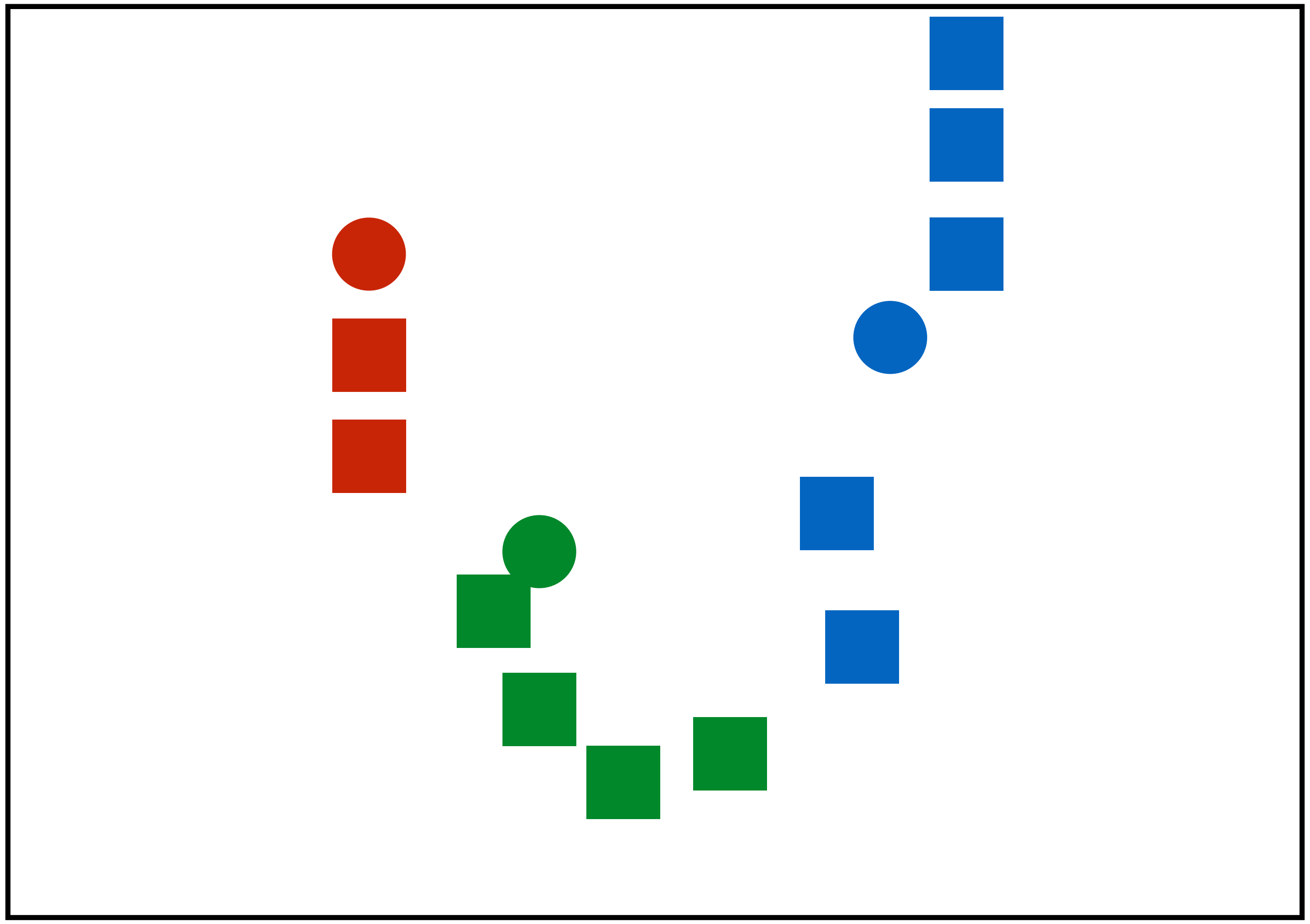
k-means Clustering

- k-means clustering partitions n observations into k clusters in which each observation belongs to the cluster with the nearest mean. — Wiki
- In practice, we use an efficient heuristic algorithm



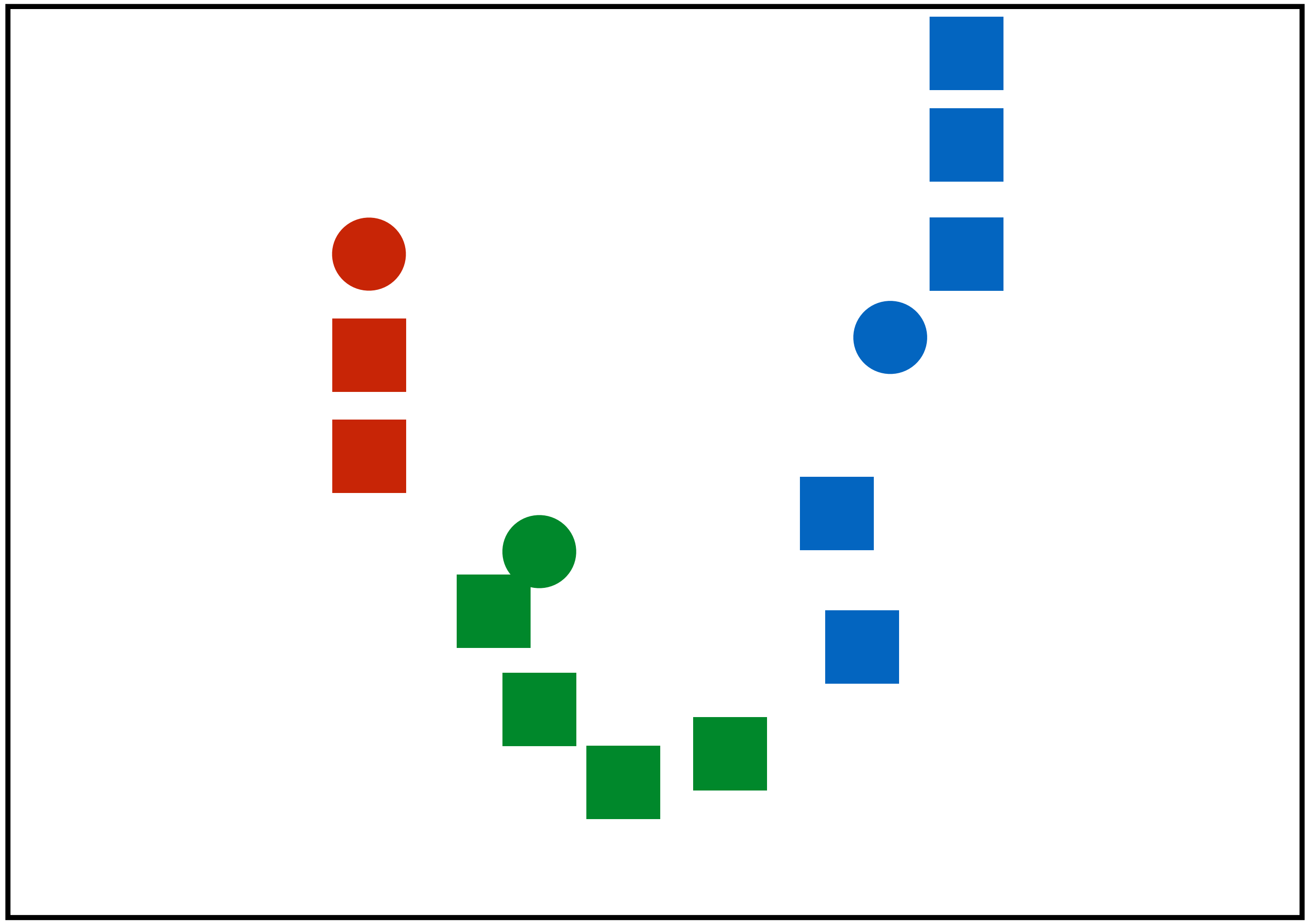
k-means Clustering

- k-means clustering partitions n observations into k clusters in which each observation belongs to the cluster with the nearest mean. — Wiki
- In practice, we use an efficient heuristic algorithm



k-means Clustering

- Each iteration partitions the n observations into clusters with nearer mean than the previous iteration



KMeans with MLlib

```
import org.apache.spark.mllib.linalg._
import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}

val lines = sc.textFile("/Users/zzhang/Works/training2016/data/scaled-sf-ny-housing-train.csv")

val data = lines.map(l => {
  val w = l.split(",")
  Vectors.dense(w(1).toDouble, w(2).toDouble, w(3).toDouble, w(4).toDouble, w(5).toDouble,
w(6).toDouble)
})

val clusters = KMeans.train(data, 2, 100)

val pred = lines.map(l => {
  val w = l.split(",")
  val v = Vectors.dense(w(1).toDouble, w(2).toDouble, w(3).toDouble, w(4).toDouble,
w(5).toDouble, w(6).toDouble)
  math.pow(cluster.predict(v) - w(0).toInt, 2)
})

res = pred.reduce(_+_)
>res: Int = 177
```

Other MLlib Functionalities

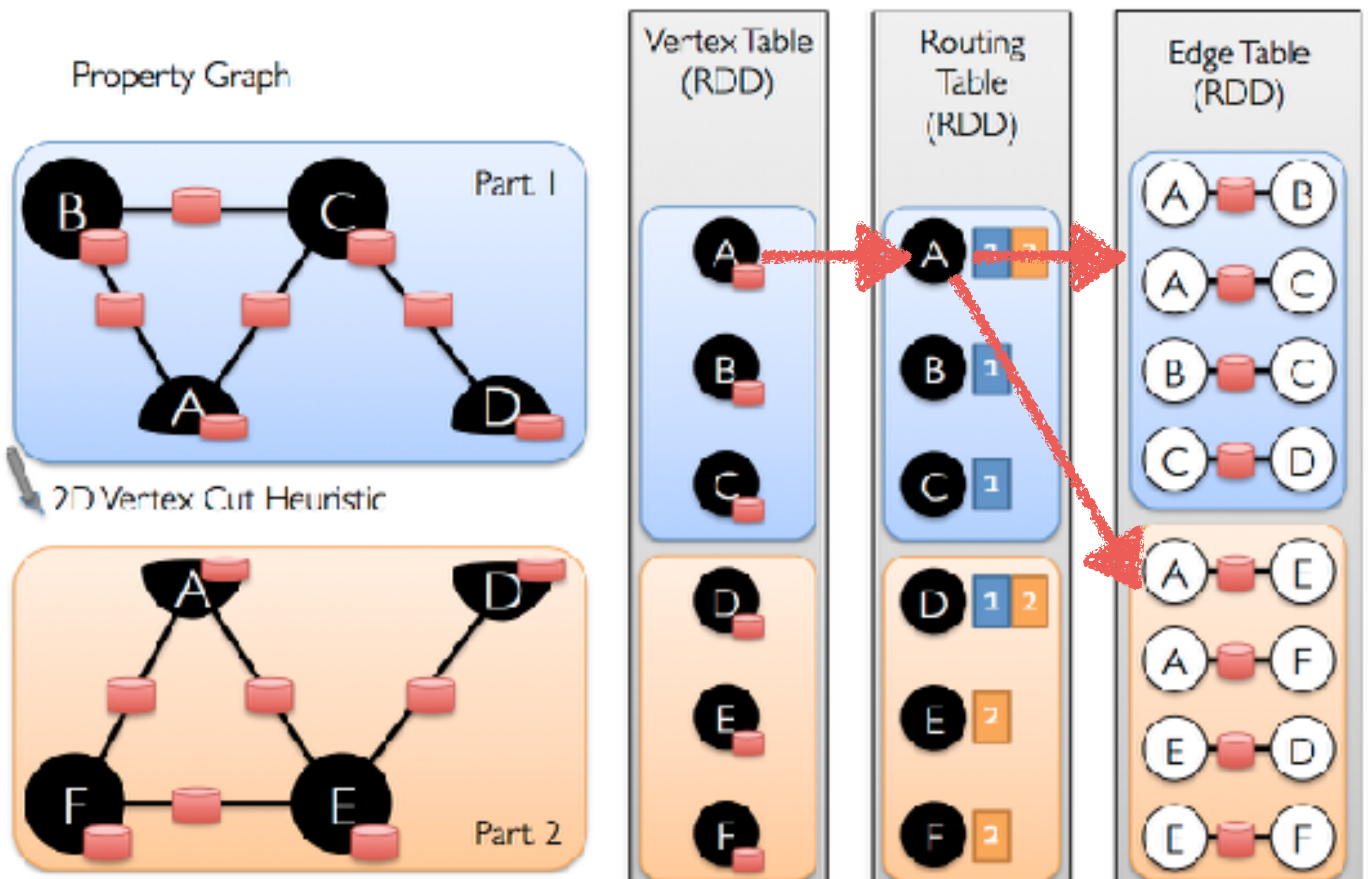
- Classification
  - org.apache.spark.mllib.classification.SVMWithSGD
  - org.apache.spark.mllib.classification.LogisticRegressionWithLBFGS
- Regression
  - org.apache.spark.mllib.regression.LinearRegressionWithSGD
  - org.apache.spark.mllib.regression.RidgeRegressionWithSGD
  - org.apache.spark.mllib.regression.LassoWithSGD
- Collaborative filtering
  - org.apache.spark.mllib.recommendation.ALS
- Clustering
  - org.apache.spark.mllib.clustering.KMeans
  - org.apache.spark.mllib.clustering.GaussianMixture
- Dimensionality Reduction
  - org.apache.spark.mllib.linalg.Matrix.computeSVD
  - org.apache.spark.mllib.linalg.Matrix.computePrincipalComponents
- Many Others

Graph Processing

- Google first released its PageRank algorithms in 1997, which formulates the Web as a directed graph and ranks the web pages
- Social networks (e.g., Facebook, Linkedin) formulates the social networks as directed graphs the do community detection and friends recommendation
- Inspired by Hadoop, there are numerous graph processing frameworks implemented in the past decade: Pregel, Giraph, GraphLab (acquired by Apple), GraphX, …

GraphX

- GraphX abstracts a graph with an RDD of vertices and an RDD of edges
- A graph is split by vertex cut



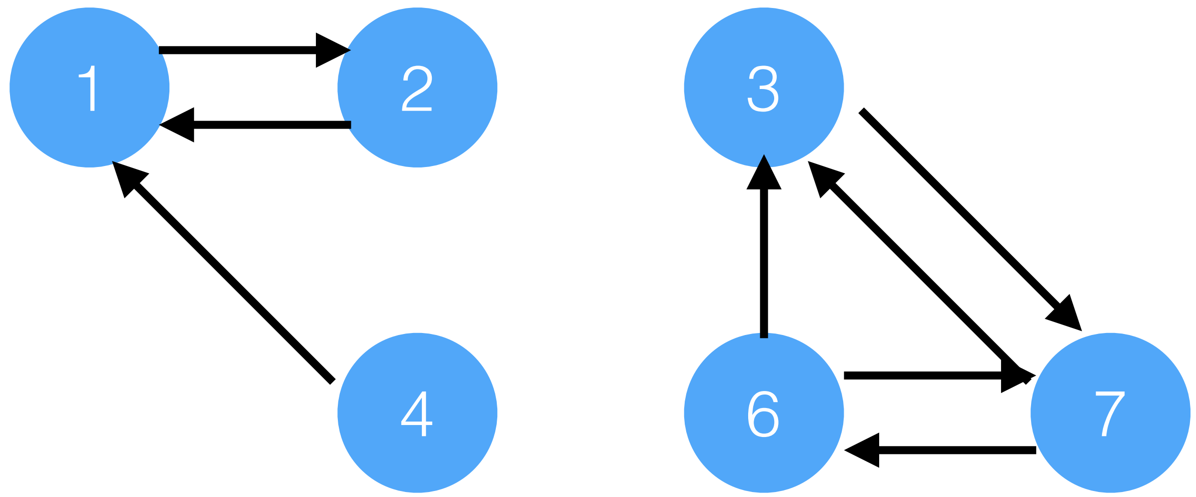
Courtesy image from http://spark.apache.org/docs/latest/graphx-programming-guide.html

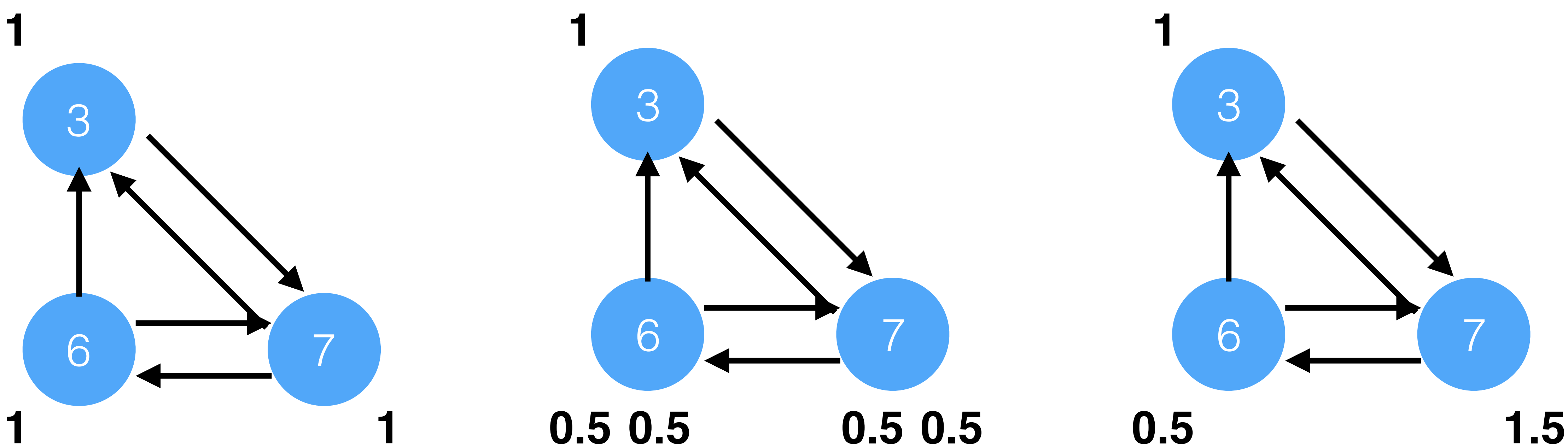
TACC

GraphX

- High level Algorithm
  - PageRank
  - Connected components
  - Triangle counting
  - Shortest path

PageRank





score = 0.15 +0.85*pagerank

Using GraphX

```
import org.apache.spark.graphx._

import org.apache.spark.graphx.util.GraphGenerators

val graph = GraphLoader.edgeListFile(sc, "/tmp/spark-training/data/followers.txt")

val ranks = graph.pageRank(0.0001).vertices

ranks.sortBy(_._2, false).collect
```

```
res10: Array[(org.apache.spark.graphx.VertexId, Double)] =
Array((1,1.4588814096664682), (2,1.390049198216498),
(7,1.2973176314422592), (3,0.9993442038507723),
(6,0.7013599933629602), (4,0.15))
```

TACC

Other GraphX Functionalities

- Connected Components
  - org.apache.spark.graphx.lib.connectedComponents
- Triangle Counting
  - org.apache.spark.graphx.lib.triangleCount
- Shortest Paths
  - org.apache.spark.graphx.lib.Shortestpaths
- Many others

Big Data Analysis - Part III: Spark Internals and Configurations

Zhao Zhang
Data Mining and Statistics Group
zzhang@tacc.utexas.edu
Texas Advanced Computing Center
May 4, 2017

These slides include talks given by Jey Kottalam of AMPLab at LBNL 2015 and Reynold Xin and Aaron Davidson of Databricks at Spark Summit 2014

Goals

- Understand Spark's architecture and components
- Understand the logic when an application is submitted
- Understand how the application is logically partitioned
- Understand how Spark manages memory

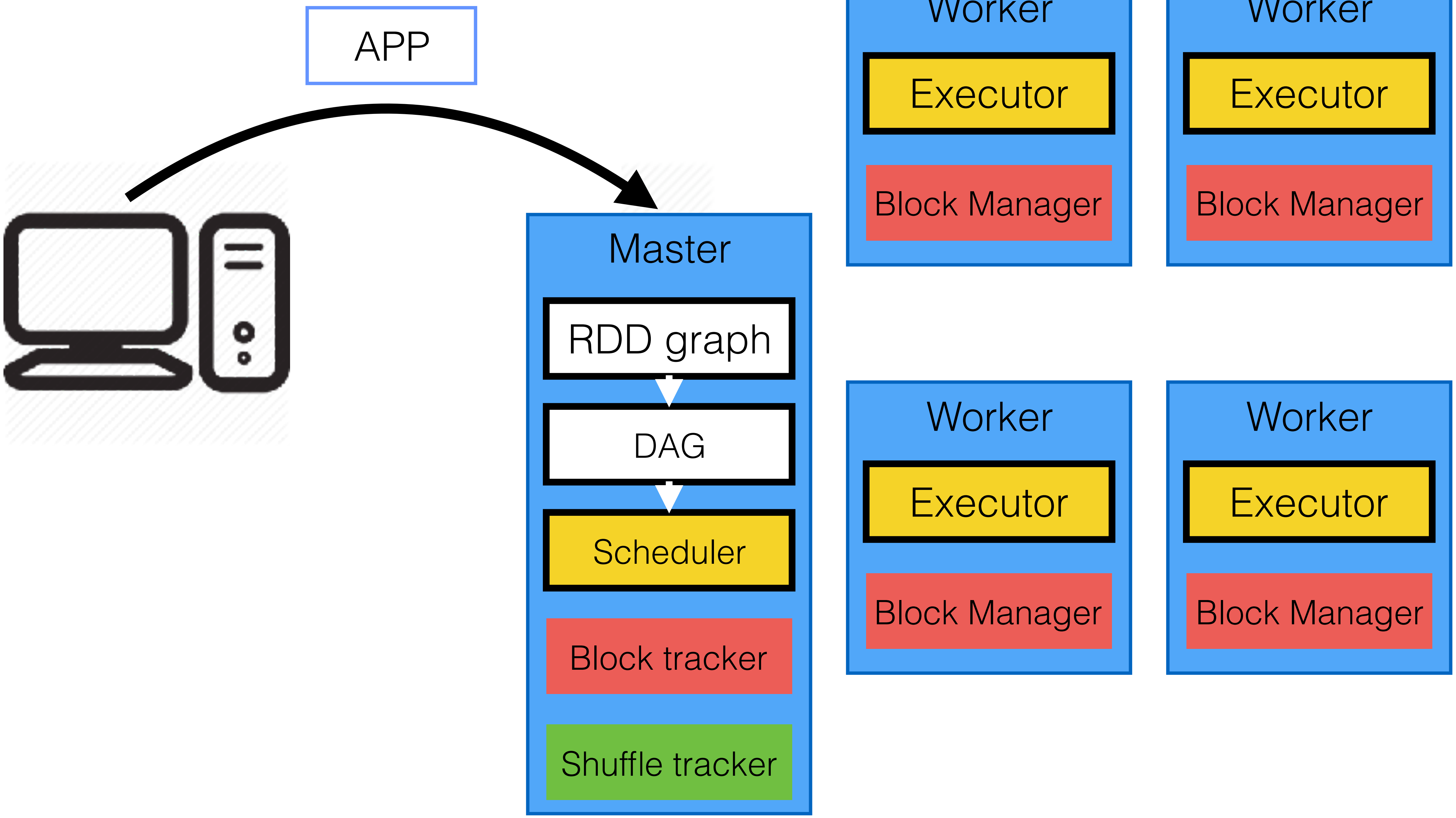
Outline

- Task Management
- Memory Management

Outline

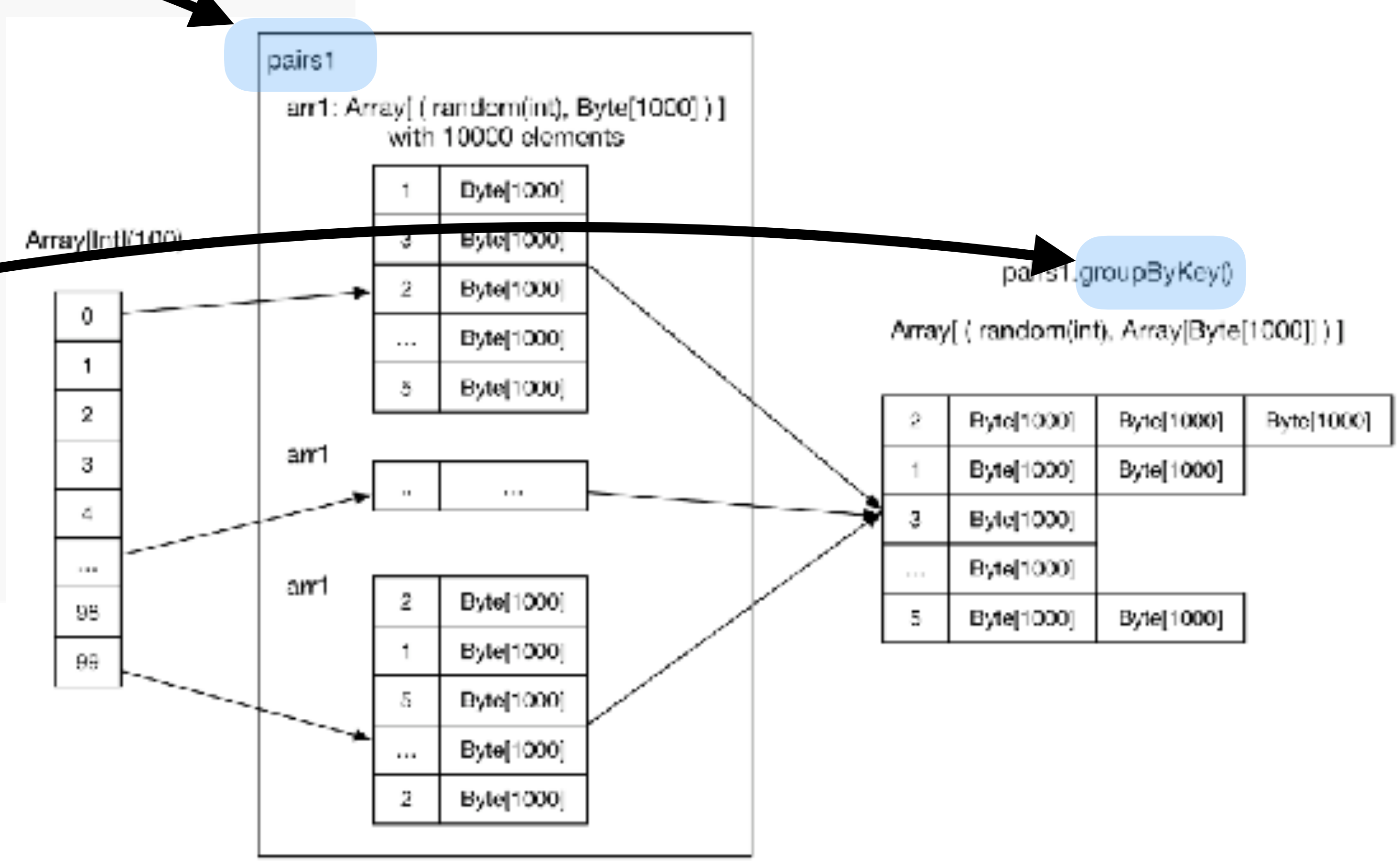
- Task Management
- Memory Management

Spark Architecture

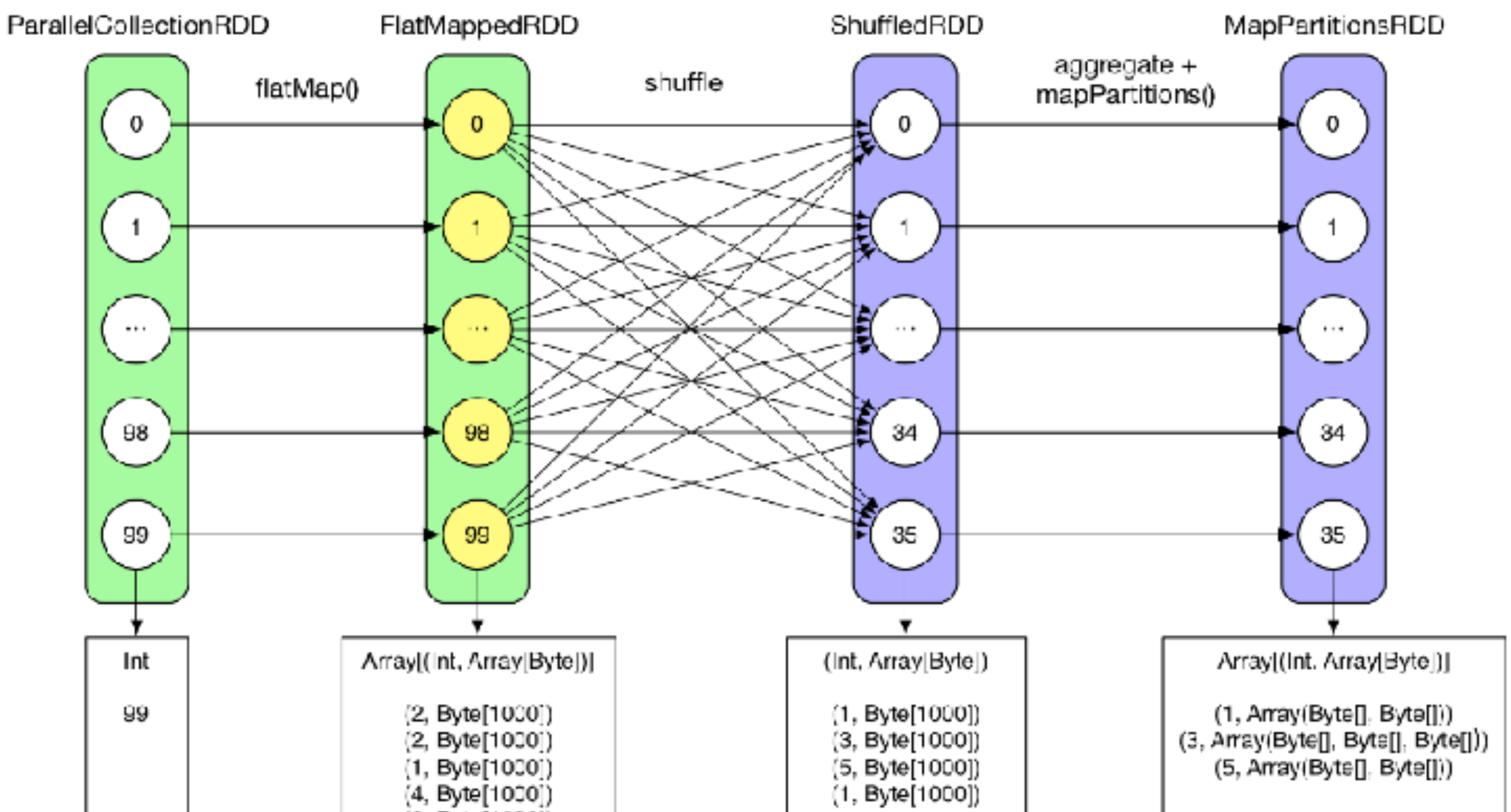


GroupByTest Example

```
object GroupByTest {
  def main(args: Array[String]) {
    val sparkConf = new SparkConf().setAppName("GroupBy Test")
    var numMappers = 100
    var numKVPairs = 10000
    var valSize = 1000
    var numReducers = 36

    val sc = new SparkContext(sparkConf)

    val pairs1 = sc.parallelize(0 until numMappers, numMappers).flatMap { p =>
      val ranGen = new Random
      var arr1 = new Array[(Int, Array[Byte])](numKVPairs)
      for (i <- 0 until numKVPairs) {
        val byteArr = new Array[Byte](valSize)
        ranGen.nextBytes(byteArr)
        arr1(i) = (ranGen.nextInt(Int.MaxValue), byteArr)
      }
      arr1
    }.cache
    // Enforce that everything has been calculated and in cache
    pairs1.count

    println(pairs1.groupByKey(numReducers).count)

    sc.stop()
  }
}
```



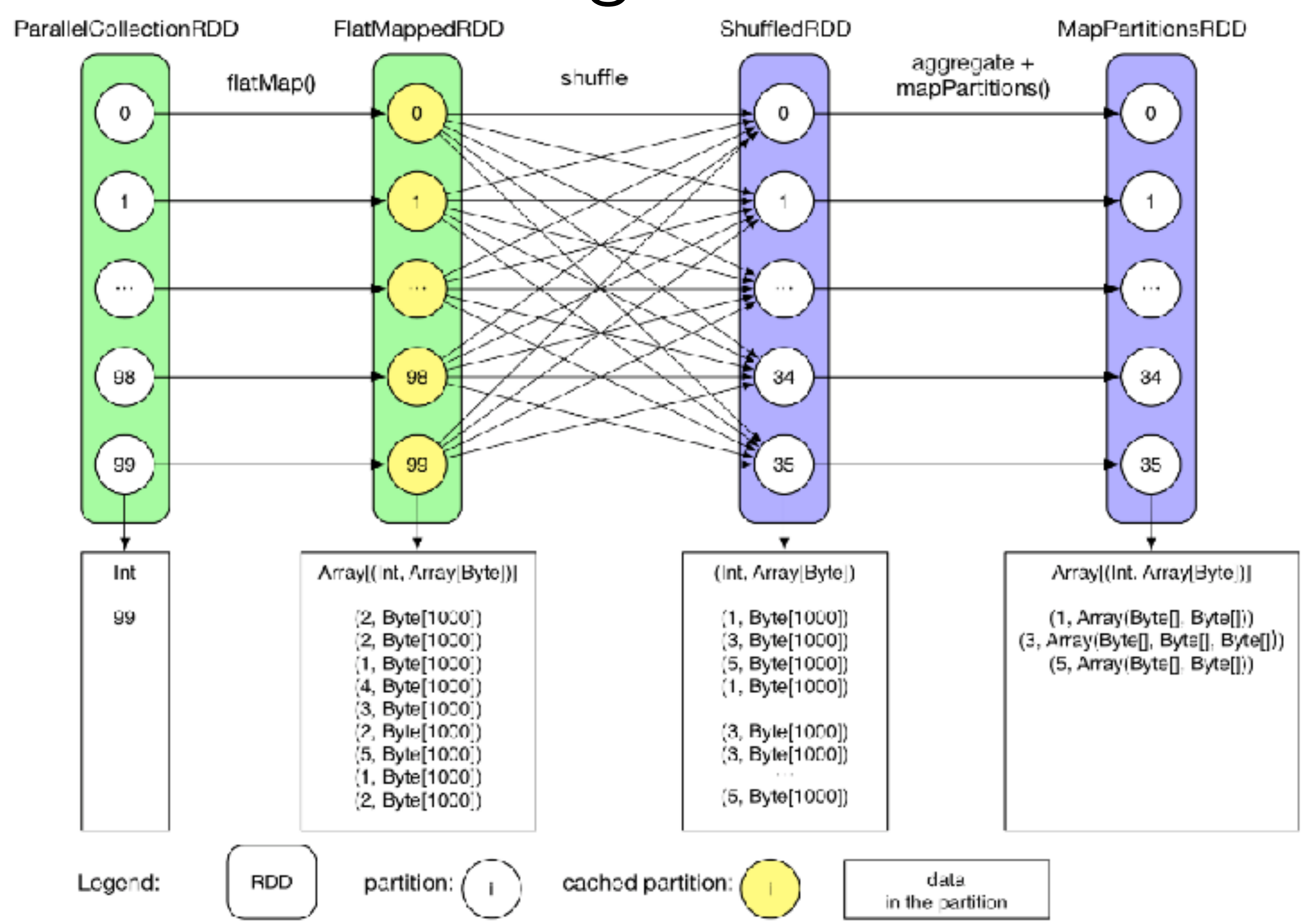
RDD Graph



MapPartitionsRDD[3] at groupByKey at GroupByTest.scala:51 (36 partitions)
ShuffledRDD[2] at groupByKey at GroupByTest.scala:51 (36 partitions)
FlatMappedRDD[1] at flatMap at GroupByTest.scala:38 (100 partitions)
ParallelCollectionRDD[0] at parallelize at GroupByTest.scala:38 (100 partitions)

DAG Generation

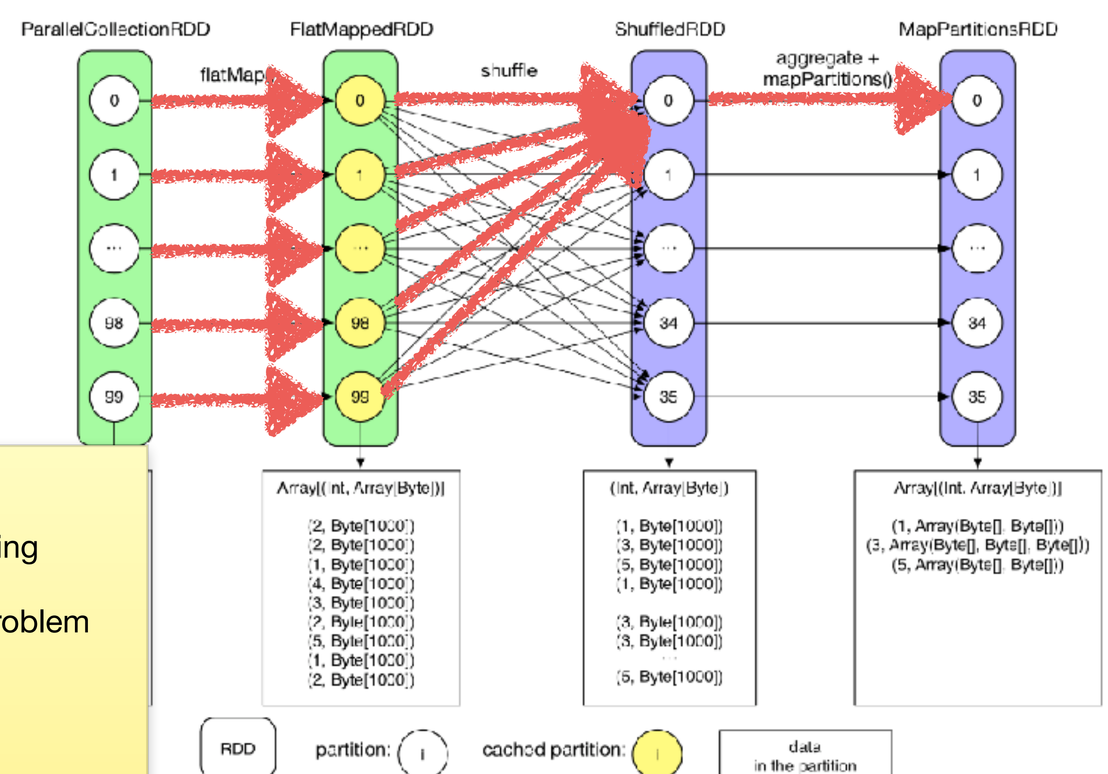
- One task per arrow?
- 100 map+ 100x36 shuffle + 36 reduce tasks
- Intermediate result storage



DAG Generation

- One task per final partition?
- The first task is too big
- Need a smart algorithm to cache intermediate data

☑ Data is not computed until needed

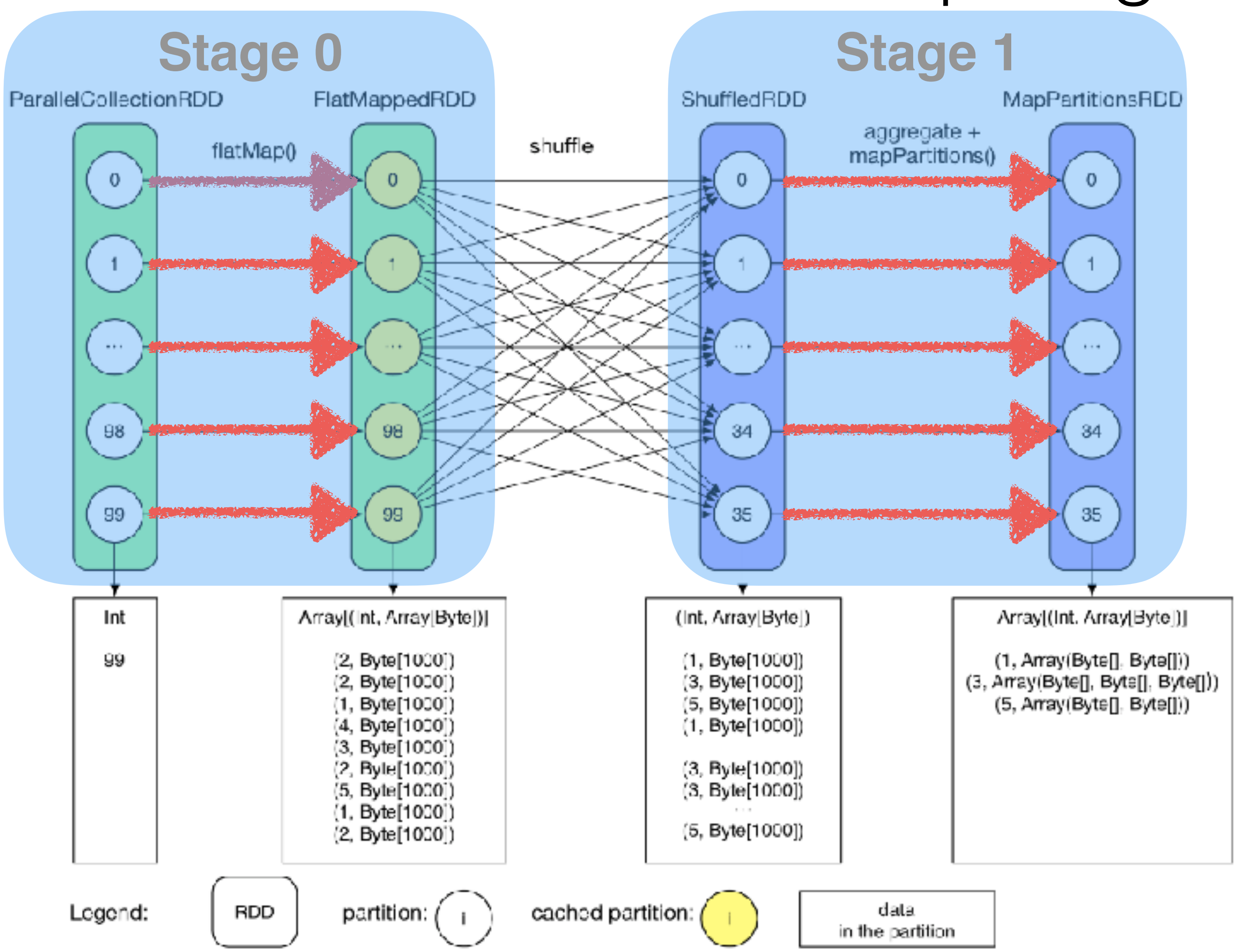


too big means poor load balancing

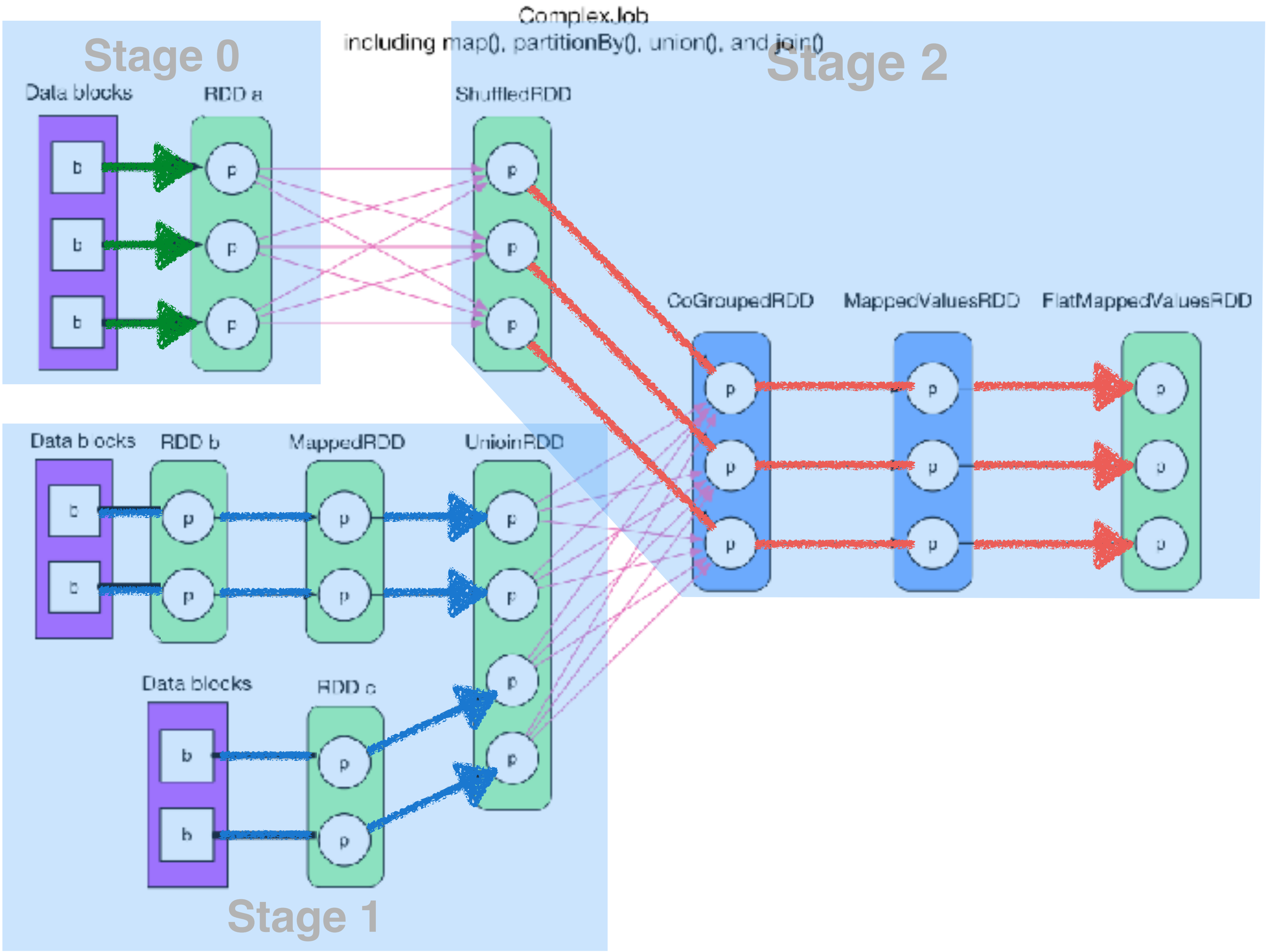
it is the shuffle that brings the problem

DAG Generation

- One task per final partition until a shuffle?
- What if there are consecutive map stages?



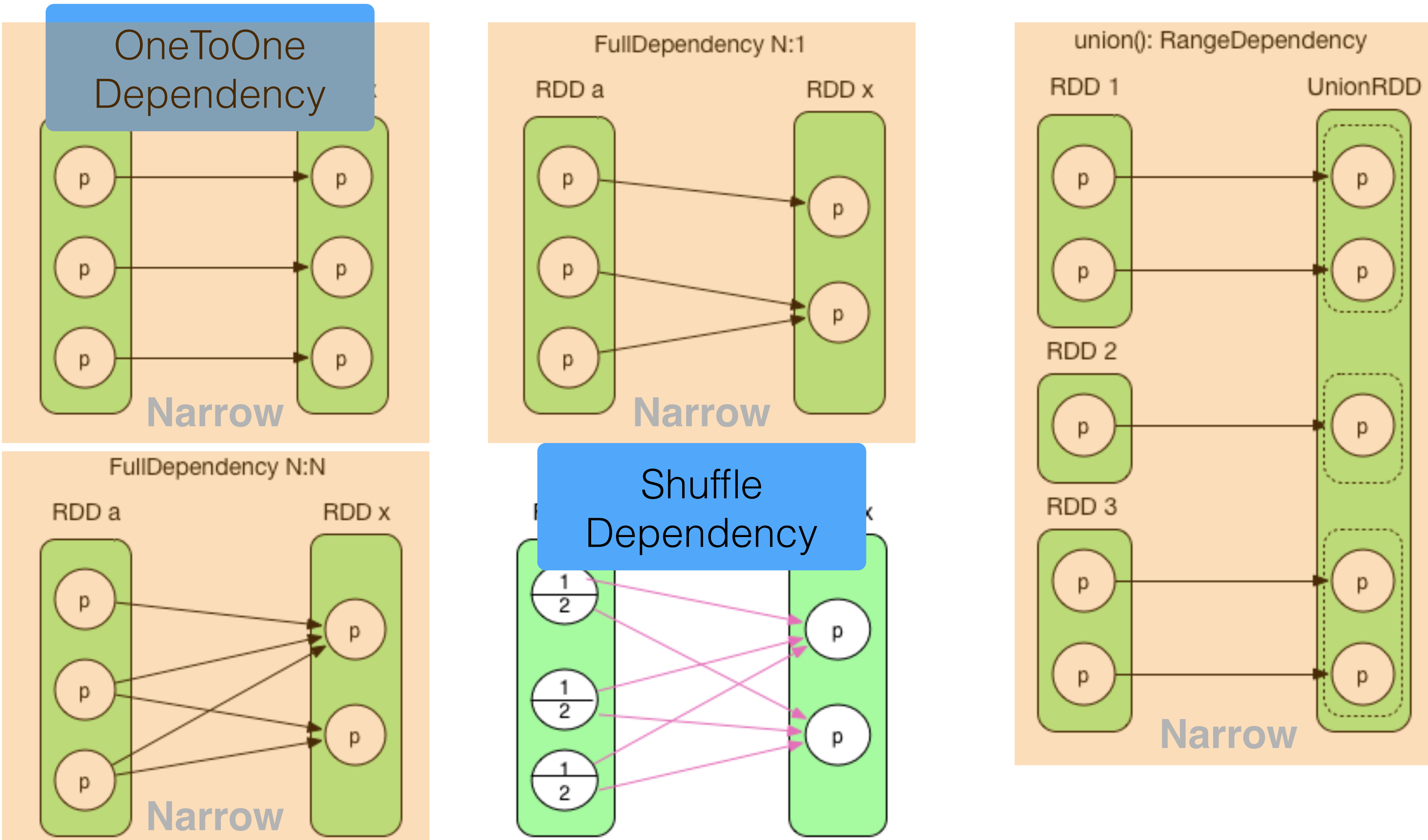
DAG Generation



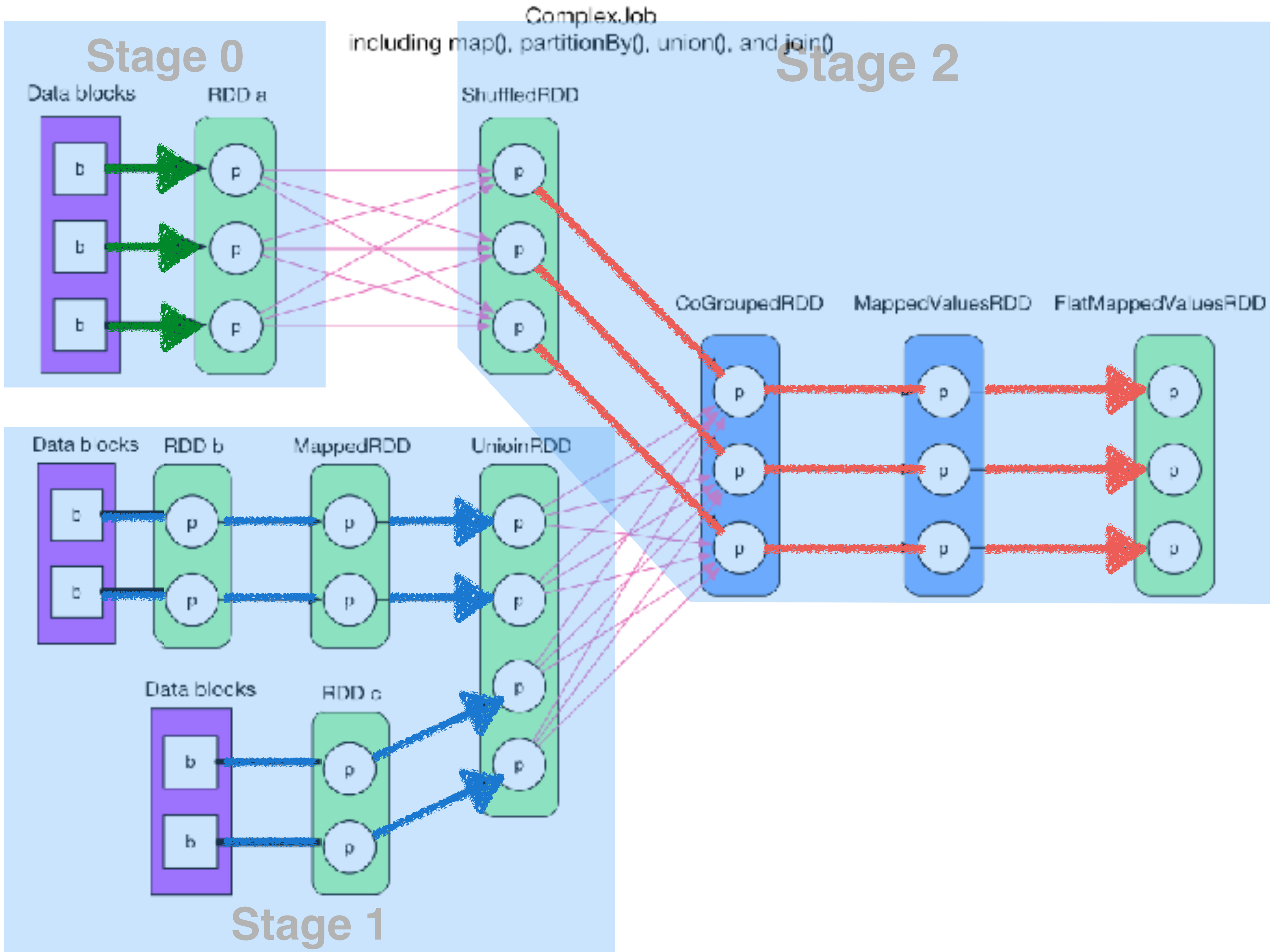
RDD Dependency

- How many parent RDDs are there for the current RDD?
- How many partitions are there in the current RDD?
- How does each partition in the current RDD depend on the partitions in parent RDDs?

RDD Dependency

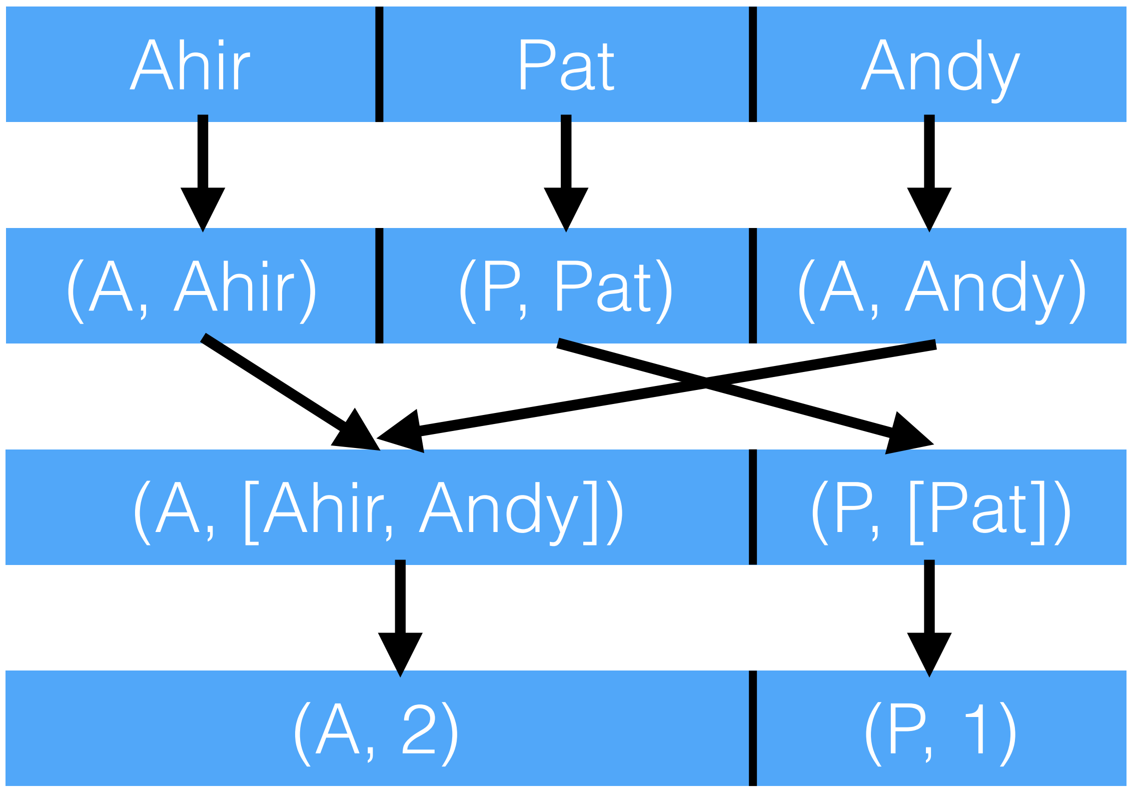


DAG Generation Revisited



Sample Execution of a Spark Job

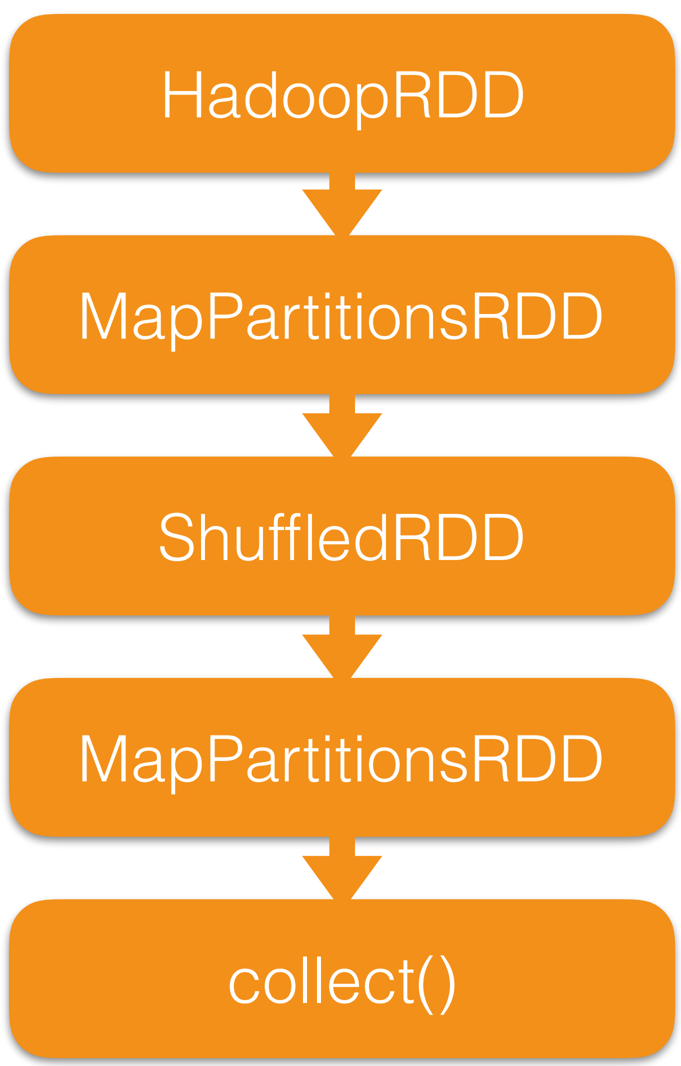
```
1. val lines = sc.textFile("hdfs://names")

2. val kvp = lines.map(name => (name(0), name))

3. val groups = kvp.groupByKey()

4. val res = groups.mapValues(names => names.toSet.size)

5. res.collect
```

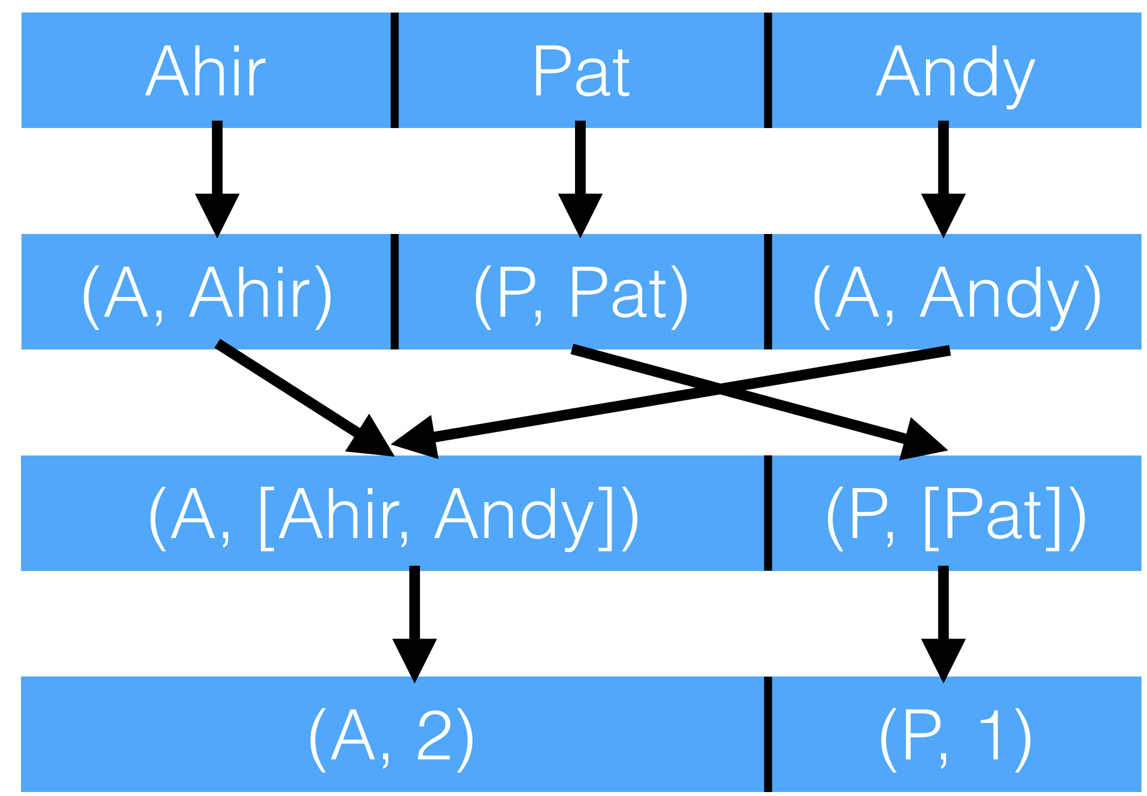


Create RDDs

`val lines = sc.textFile("hdfs://names")`	HadoopRDD
`val kvp = lines.map(name => (name(0), name))`	MapPartitionsRDD
`val groups = kvp.groupByKey()`	ShuffledRDD
`val res = groups.mapvalues(names => names.toSet.size)`	MapPartitionsRDD
`res.collect`	collect()

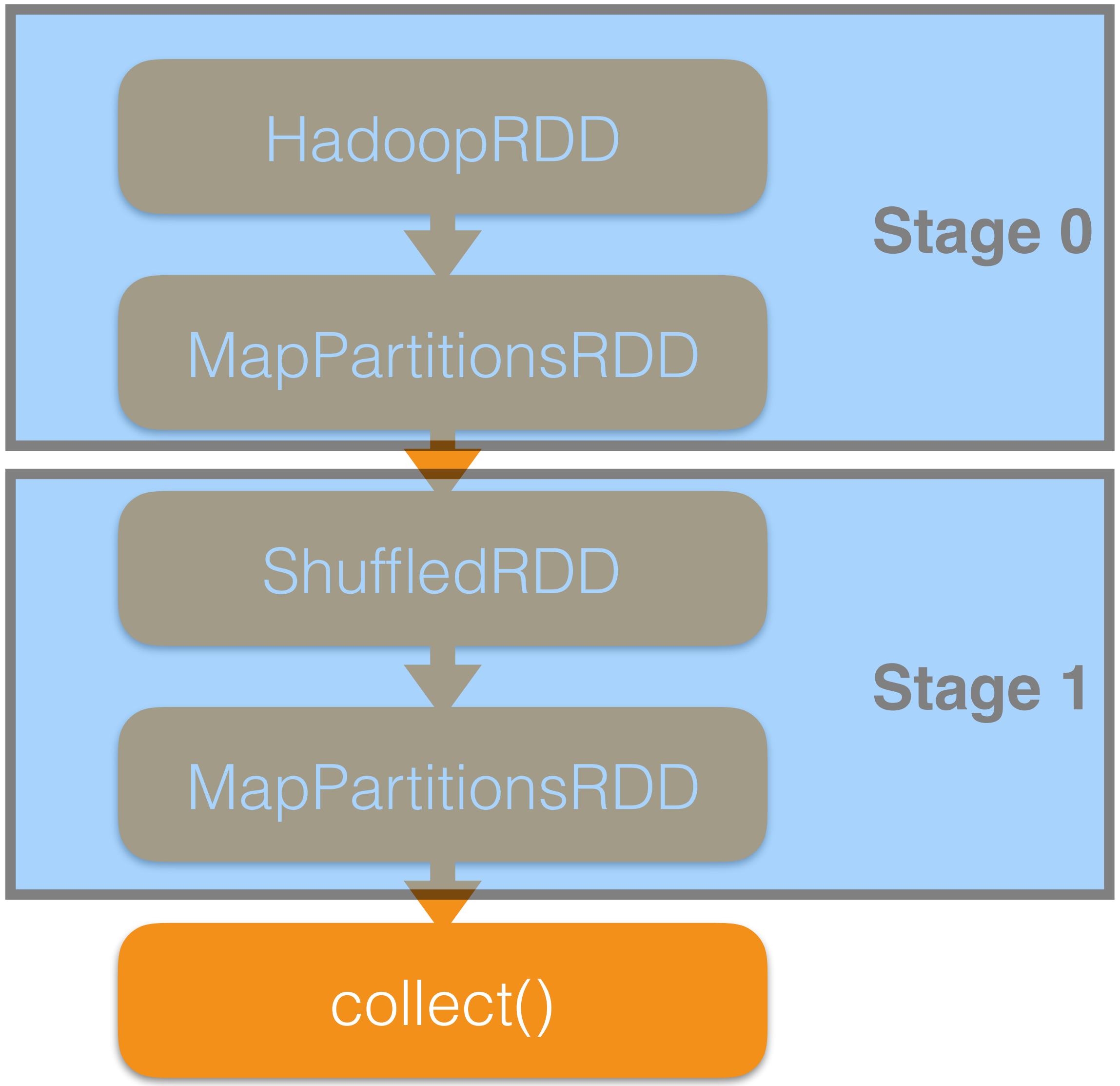
Create Execution Plan

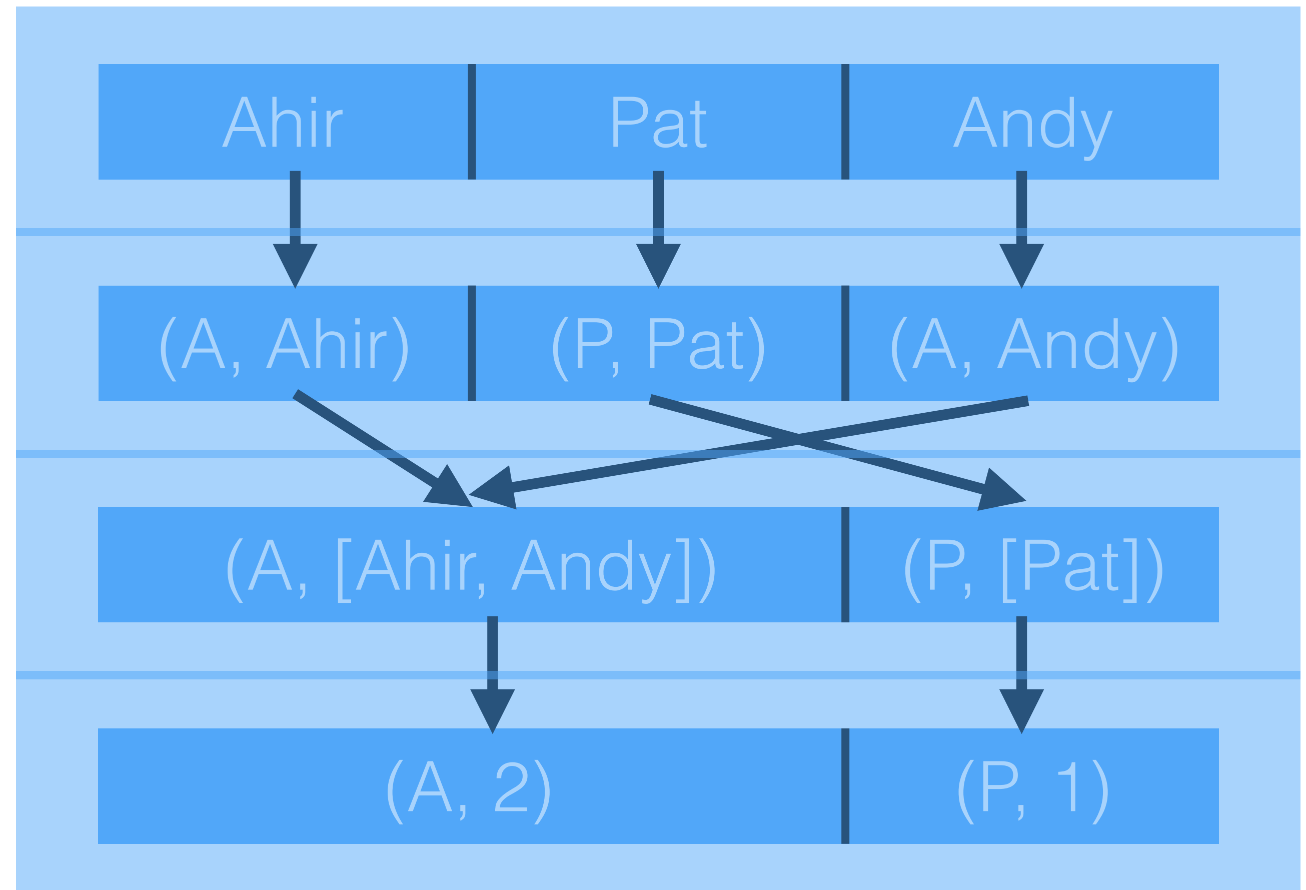




res0 = [(A, 2), (P, 1)]

Create Execution Plan





res0 = [(A, 2), (P, 1)]

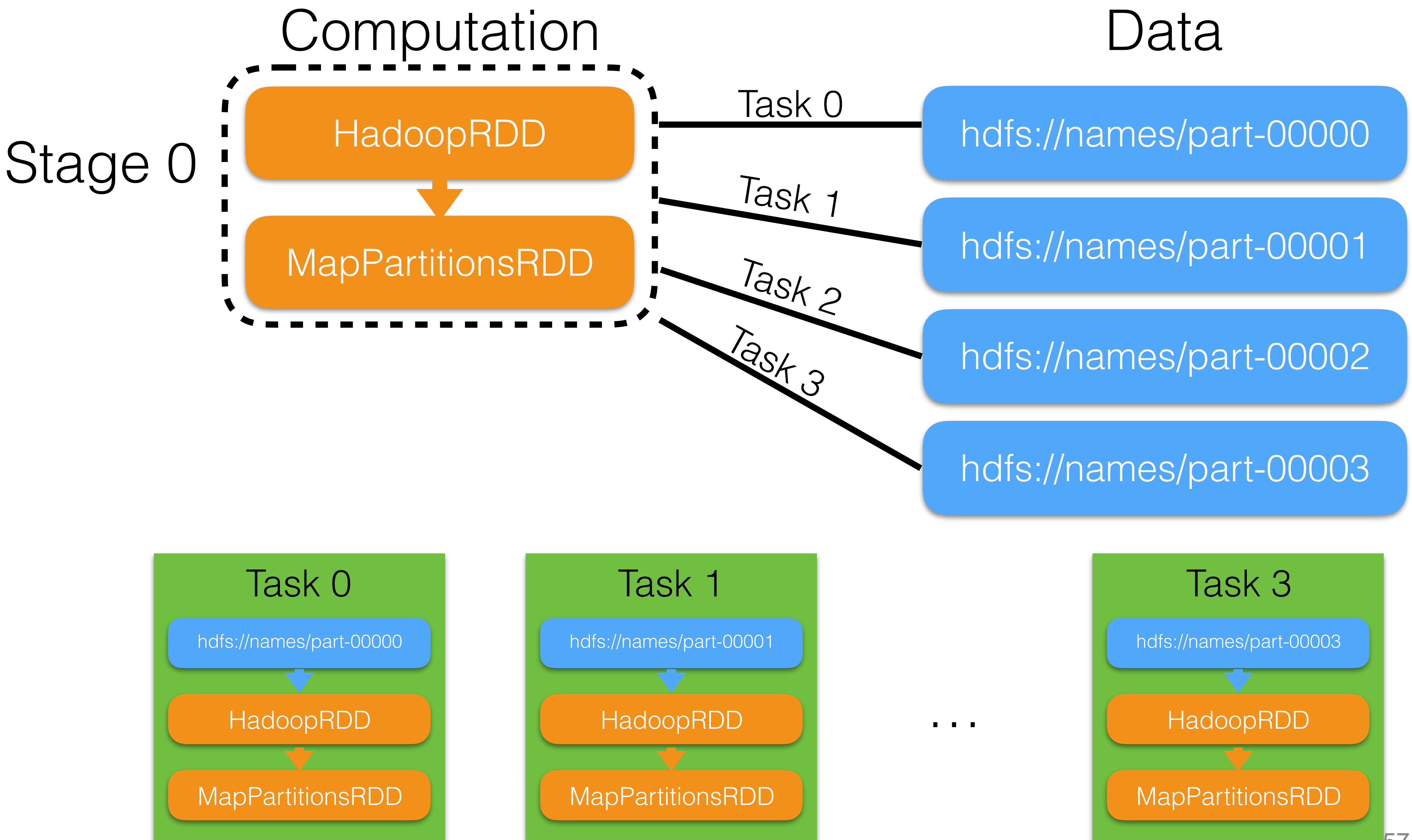
Log Interpretation

17/05/03 17:19:33 INFO SparkContext: Starting job: collect at <console>:45
17/05/03 17:19:33 INFO DAGScheduler: Got job 321 (collect at <console>:45) with 2 output partitions
17/05/03 17:19:33 INFO DAGScheduler: Final stage: **ResultStage 340** (collect at <console>:45)
17/05/03 17:19:33 INFO DAGScheduler: Parents of final stage: List(**ShuffleMapStage 339**)
17/05/03 17:19:33 INFO DAGScheduler: Missing parents: List(ShuffleMapStage 339)
17/05/03 17:19:33 INFO DAGScheduler: Submitting ShuffleMapStage 339 (MapPartitionsRDD[658] at map at <console>:38), which has no missing parents
17/05/03 17:19:33 INFO DAGScheduler: Submitting 2 missing tasks from ShuffleMapStage 339 (MapPartitionsRDD[658] at map at <console>:38)
17/05/03 17:19:33 INFO TaskSchedulerImpl: Adding task set 339.0 with 2 tasks
…
17/05/03 17:19:33 INFO TaskSetManager: **Finished task 0.0 in stage 339.0** (TID 688) in 4 ms on localhost (1/2)
17/05/03 17:19:33 INFO TaskSetManager: **Finished task 1.0 in stage 339.0** (TID 689) in 4 ms on localhost (2/2)
17/05/03 17:19:33 INFO DAGScheduler: **ShuffleMapStage 339** (map at <console>:38) **finished** in 0.004 s
…
17/05/03 17:19:33 INFO DAGScheduler: waiting: Set(**ResultStage 340**)
17/05/03 17:19:33 INFO DAGScheduler: Submitting ResultStage 340 (MapPartitionsRDD[660] at mapValues at <console>:42), which has no missing parents
17/05/03 17:19:33 INFO DAGScheduler: Submitting 2 missing tasks from ResultStage 340 (MapPartitionsRDD[660] at mapValues at <console>:42)
17/05/03 17:19:33 INFO TaskSchedulerImpl: Adding task set 340.0 with 2 tasks
…
17/05/03 17:19:33 INFO TaskSetManager: **Finished task 1.0 in stage 340.0** (TID 691) in 2 ms on localhost (1/2)
17/05/03 17:19:33 INFO TaskSetManager: **Finished task 0.0 in stage 340.0** (TID 690) in 2 ms on localhost (2/2)
17/05/03 17:19:33 INFO DAGScheduler: **ResultStage 340** (collect at <console>:45) **finished** in 0.002 s
17/05/03 17:19:33 INFO DAGScheduler: **Job 321 finished**: collect at <console>:45, took 0.009950 s
res14: Array[(Char, Int)] = Array((P,1), (A,2))

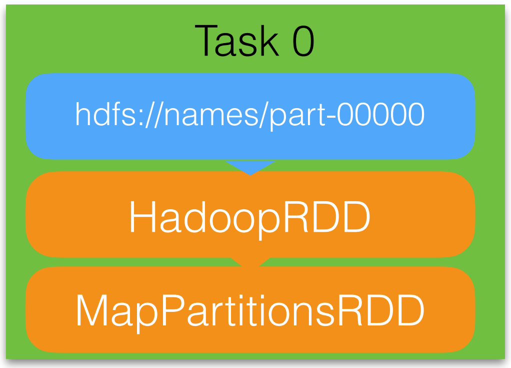
Schedule Tasks

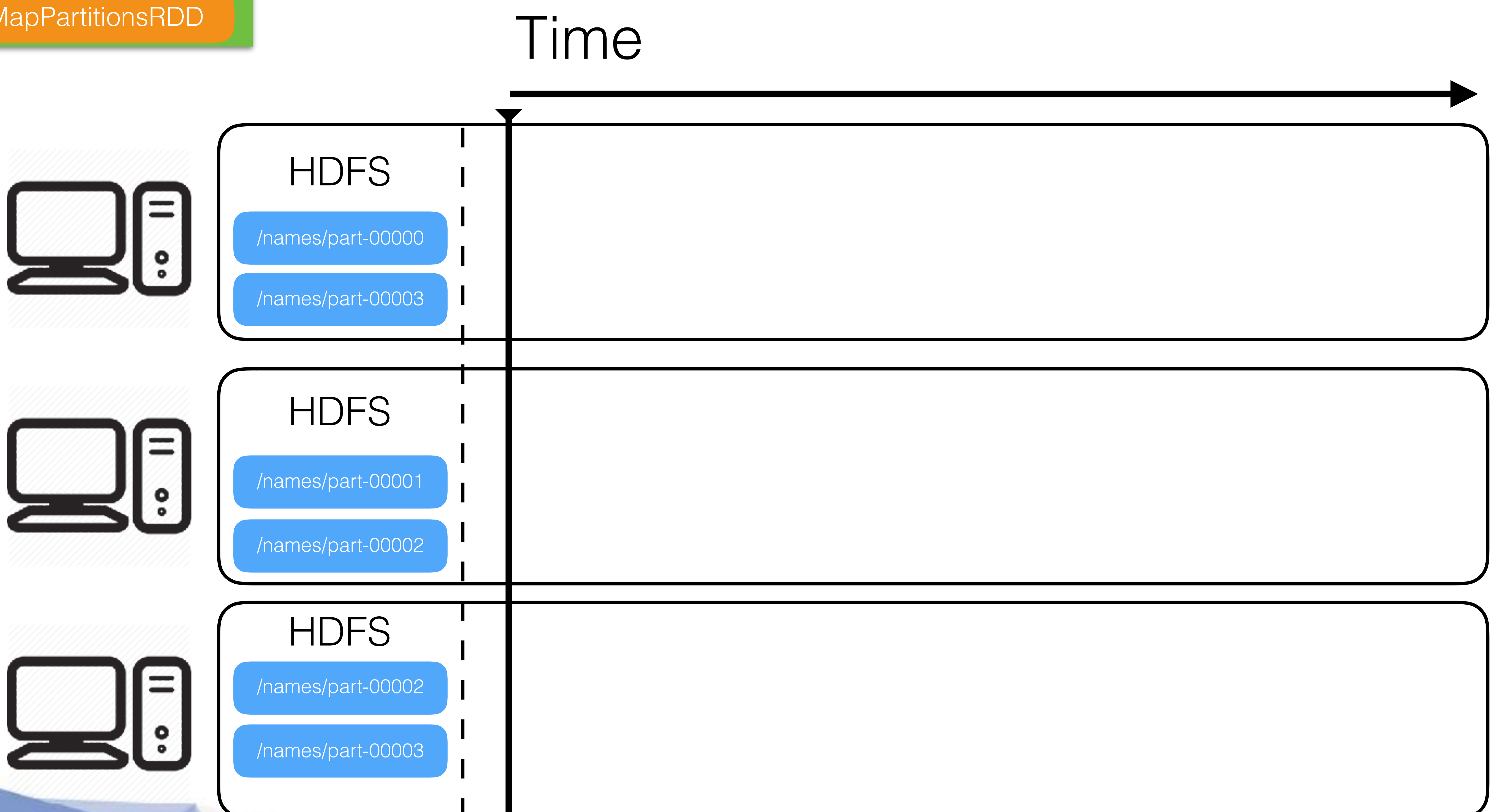
- Split each stage into tasks based on partitions
- A task is data + computation
- From the last stage, recursively find parent stages, then schedule the stage that all parent stages have been executed or there is no parent stage
- Execute all tasks within a stage before moving on to the next

Schedule Tasks

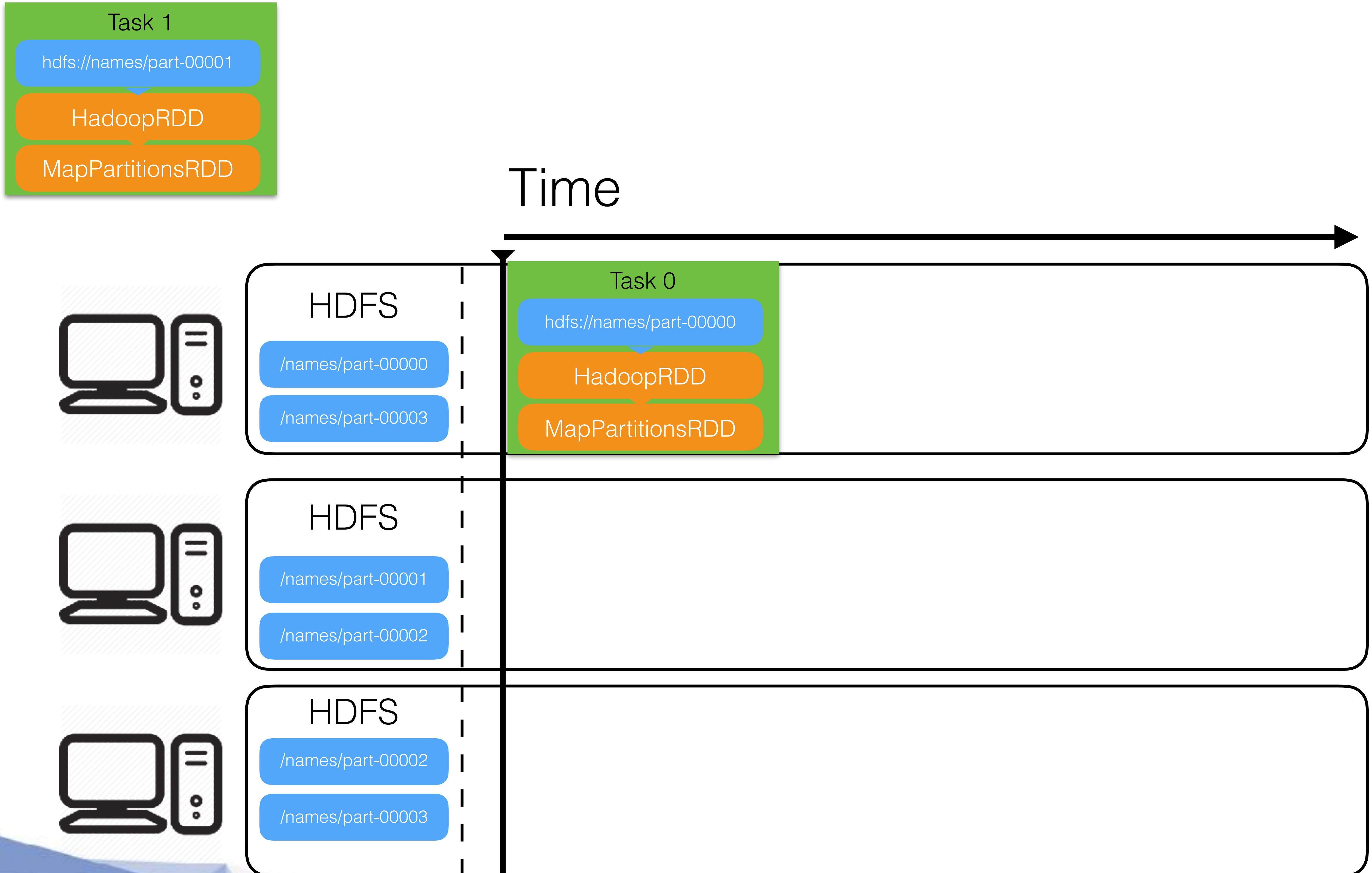


Schedule Tasks





Schedule Tasks



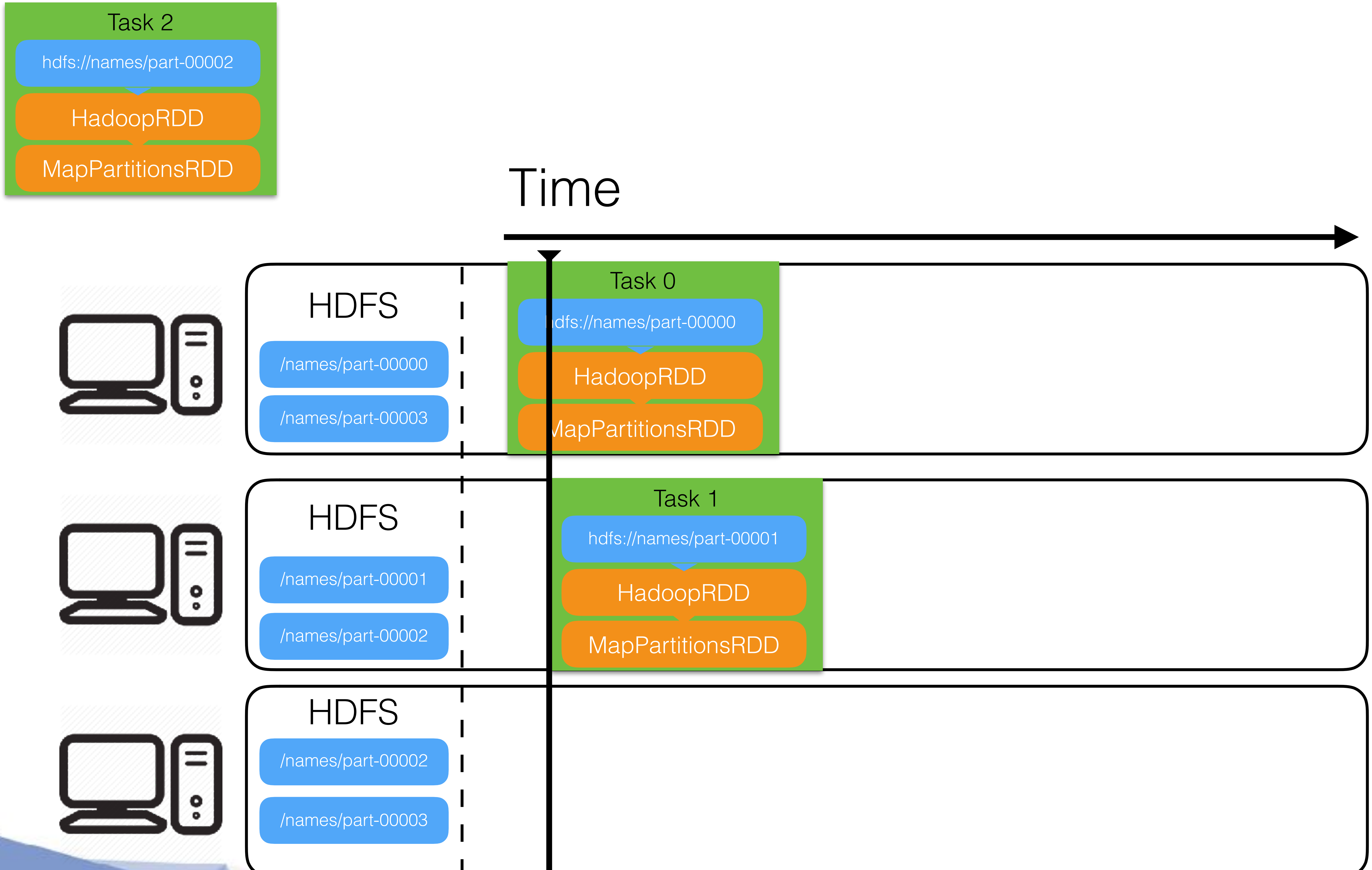
TACC

Delay Scheduling

- How long a task should waits for the node that has the data?
  - spark.locality.wait (default 3s)
  - spark.locality.wait.process
  - spark.locality.wait.node
  - spark.locality.wait.rack

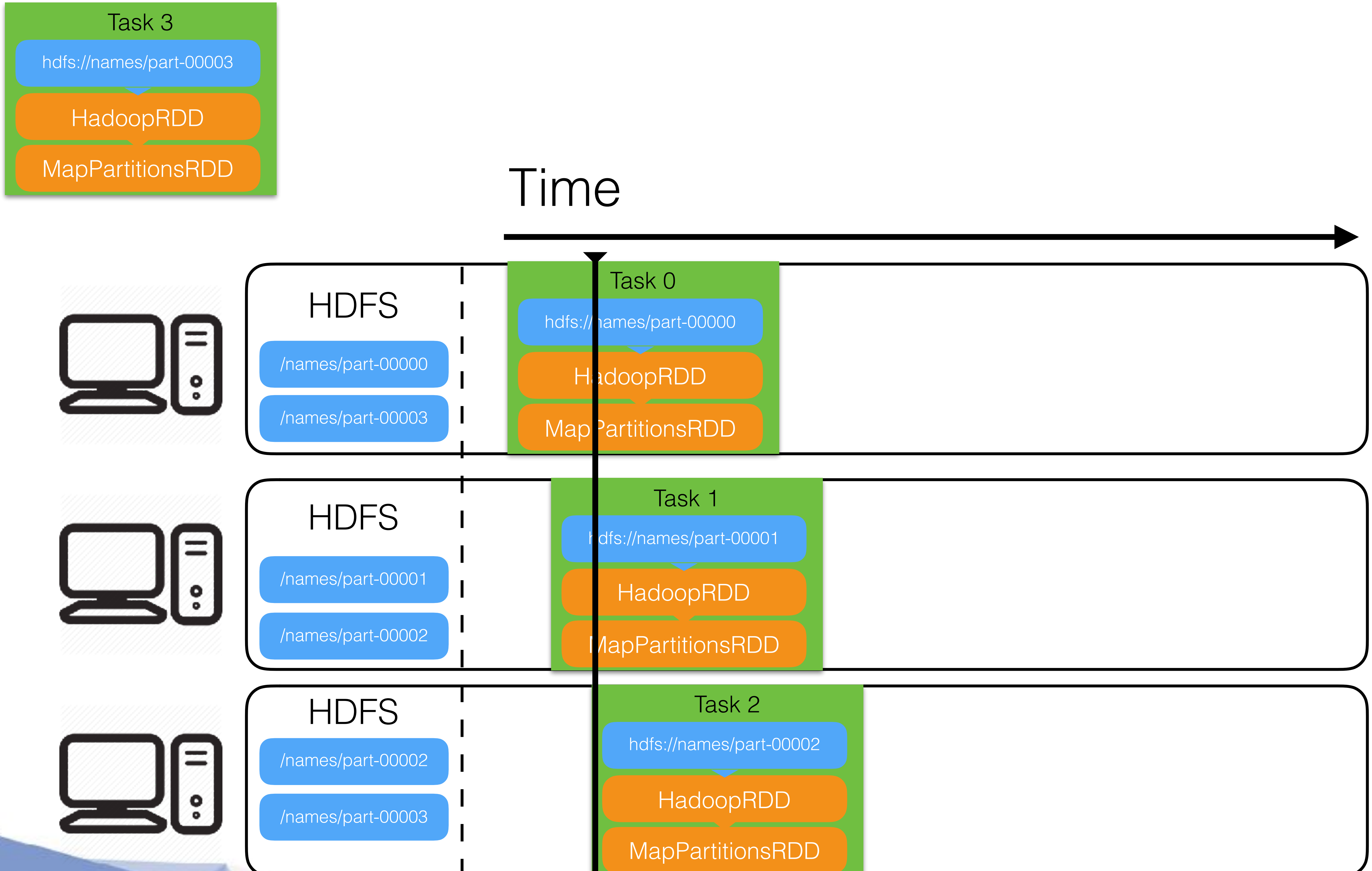
TACC

Schedule Tasks



TACC

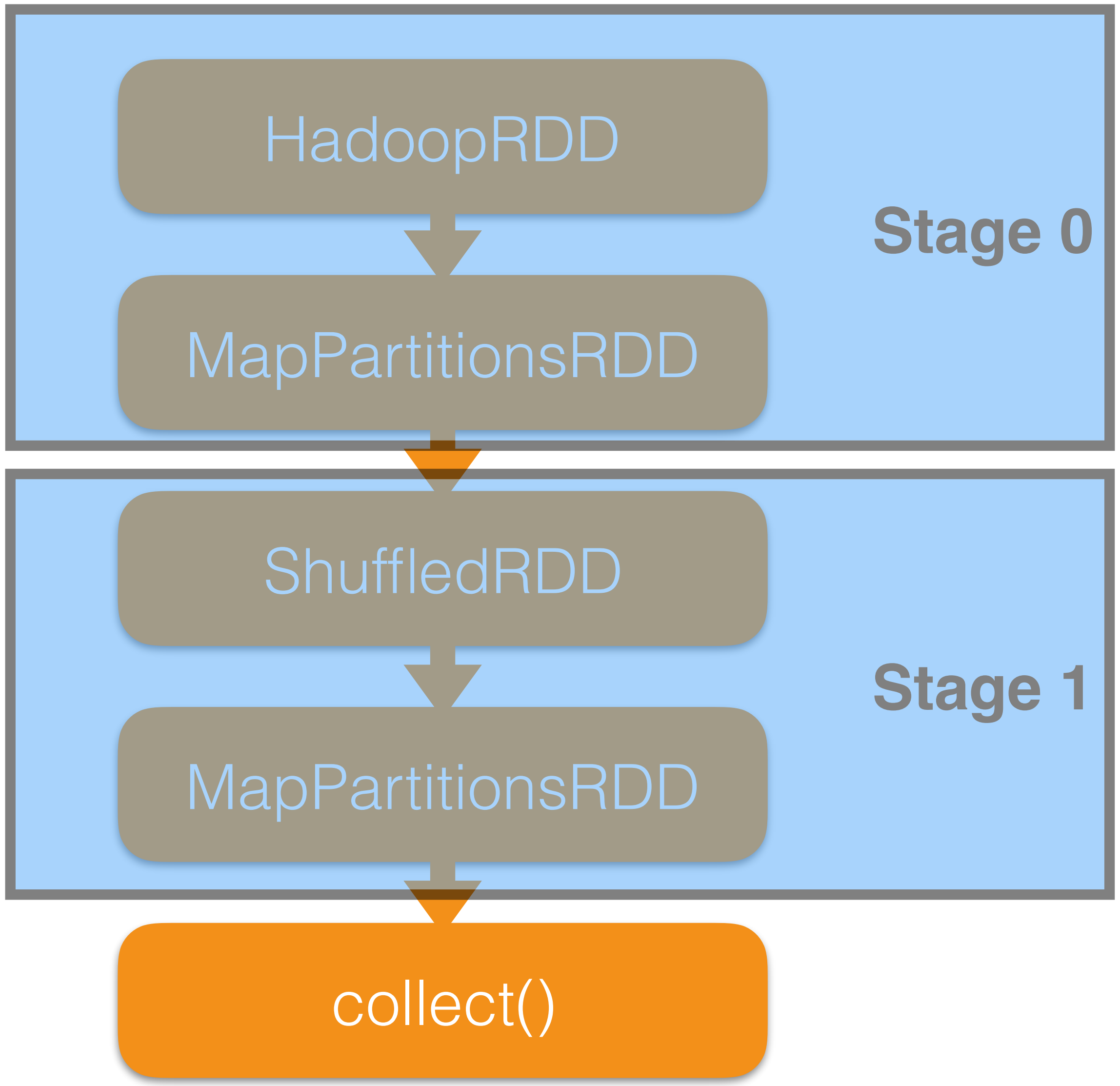
Schedule Tasks



Now Stage 1 Finishes

- Shuffle
- Stage 0 Execution
- Results

Done


HadoopRDD
MapPartitionsRDD
Stage 0
ShuffledRDD
MapPartitionsRDD
Stage 1
collect()

Cache

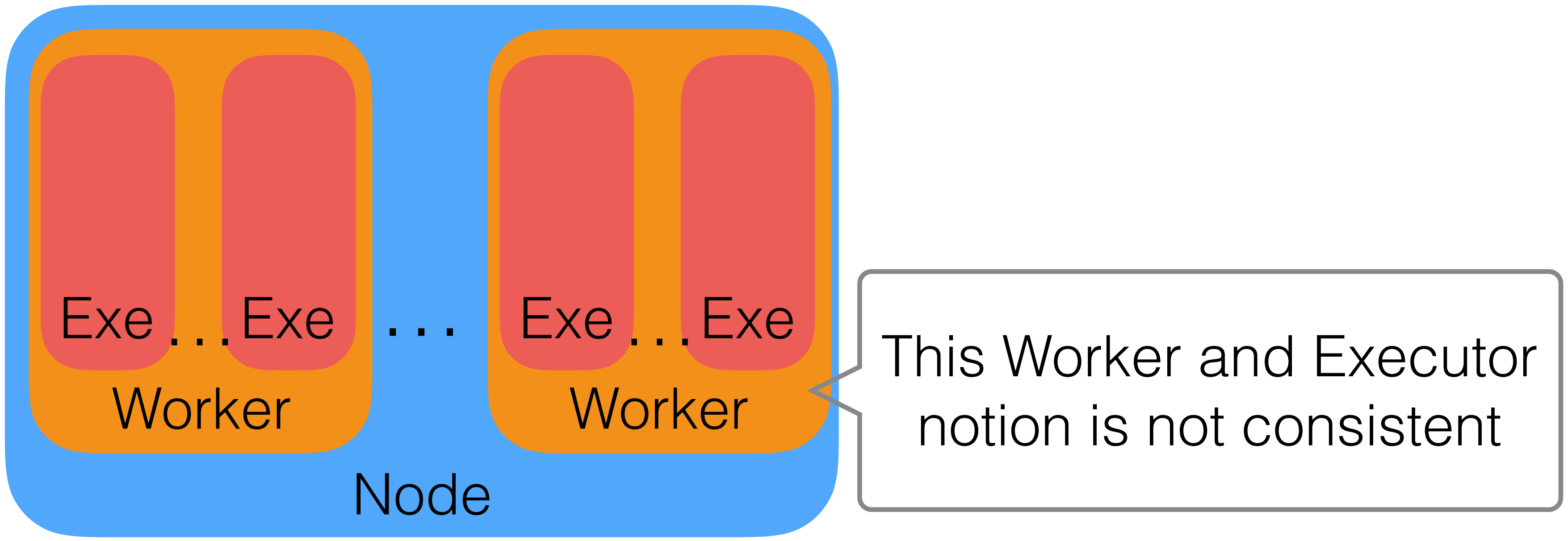
- Cache Option

```
val lines = sc.textFile("hdfs://names")
val kvp = lines.map(name => (name(0), name))
val groups = kvp.groupByKey()
val res = groups.mapvalues(names => names.toSet.size)
res.collect
```

HadoopRDD → MapPartitionsRDD → ShuffledRDD → MapPartitionsRDD → collect()

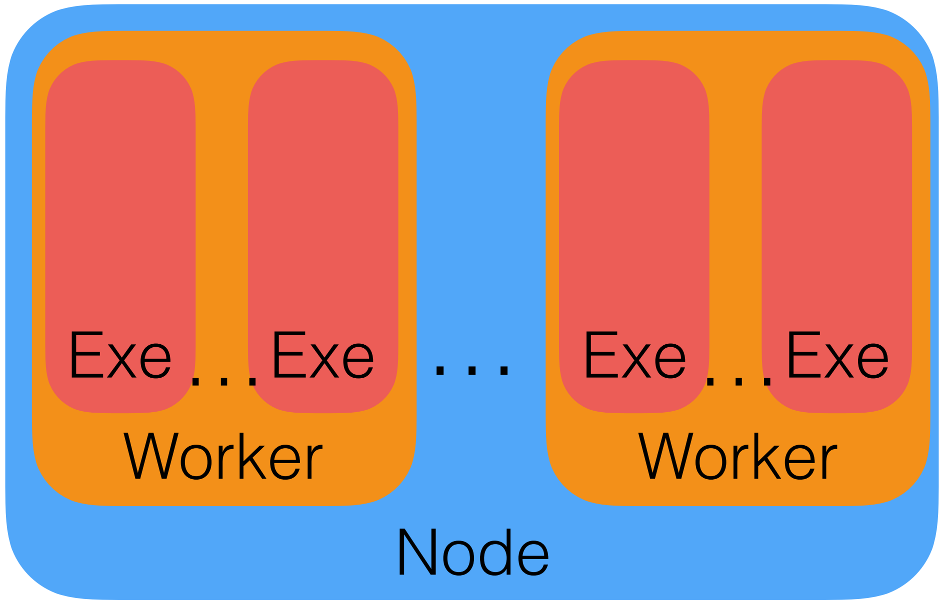
Executor Deployment

- Node — Worker — Executor



- Workers share the same physical node
- Executors in the same worker shared the same process

Executor Deployment



- Spark YARN mode, in conf/spark-env.sh
- SPARK_EXECUTOR_INSTANCES (default 2)
- SPARK_EXECUTOR_CORES (default 1)

Review

- How does Spark generate DAG?
- How does Spark partition a DAG into tasks?
- How do Spark tasks get schedule?
- How to set Spark executors

TACC

Outline

- Task Management
- Memory Management

Spark Memory Consumption

- Spark consumes more memory than you think
  - Each Java object has a roughly 16-byte "object header"
  - Java Strings have ~40 bytes of overhead
  - Common collection classes, such as HashMap and LinkedList, has a "wrapper object" for each entry
  - Collections of primitive types are stored as "boxed" objects such as Java.lang.Integer

TACC

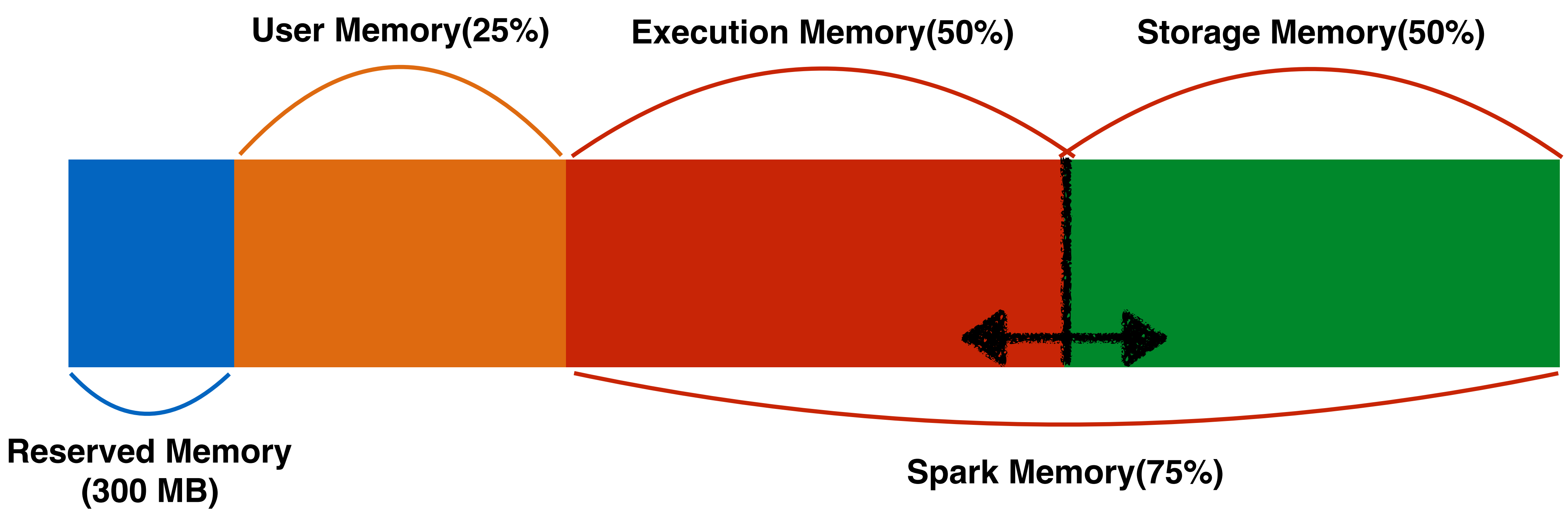
Spark Symptoms

- If Spark is running with insufficient memory
  - Spark is running slow due to garbage collection
  - Executor get lost due to Out-of-Memory (OOM) exception

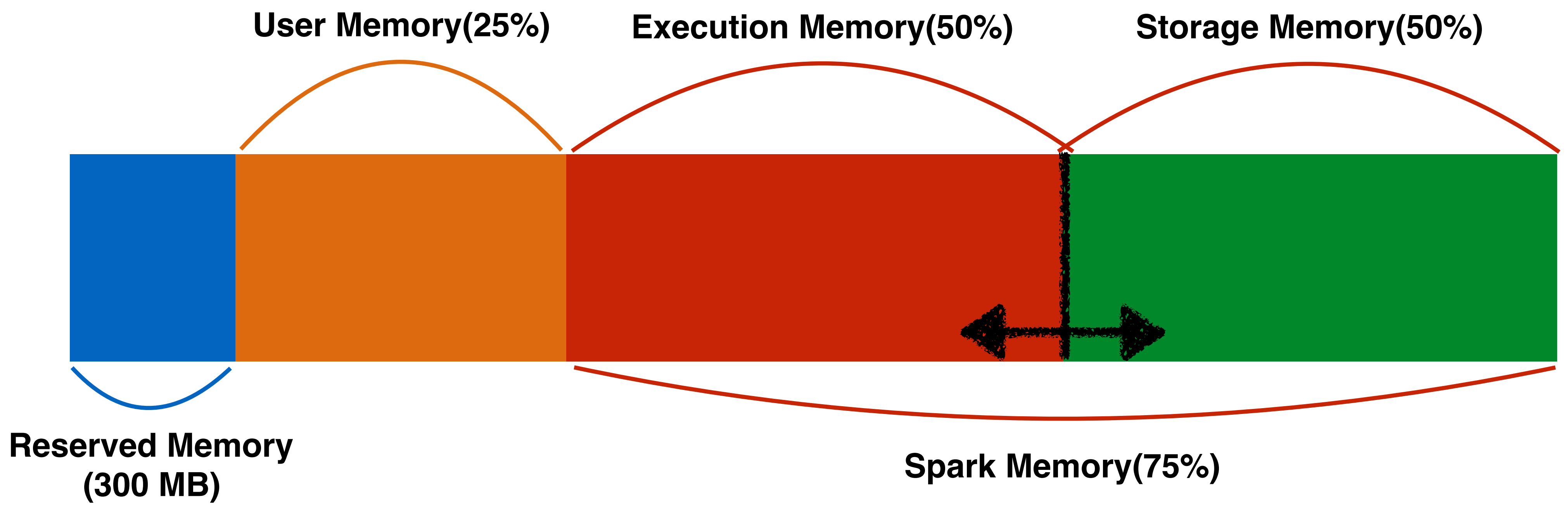
TACC

Memory Management Model

- Spark Memory Allocation

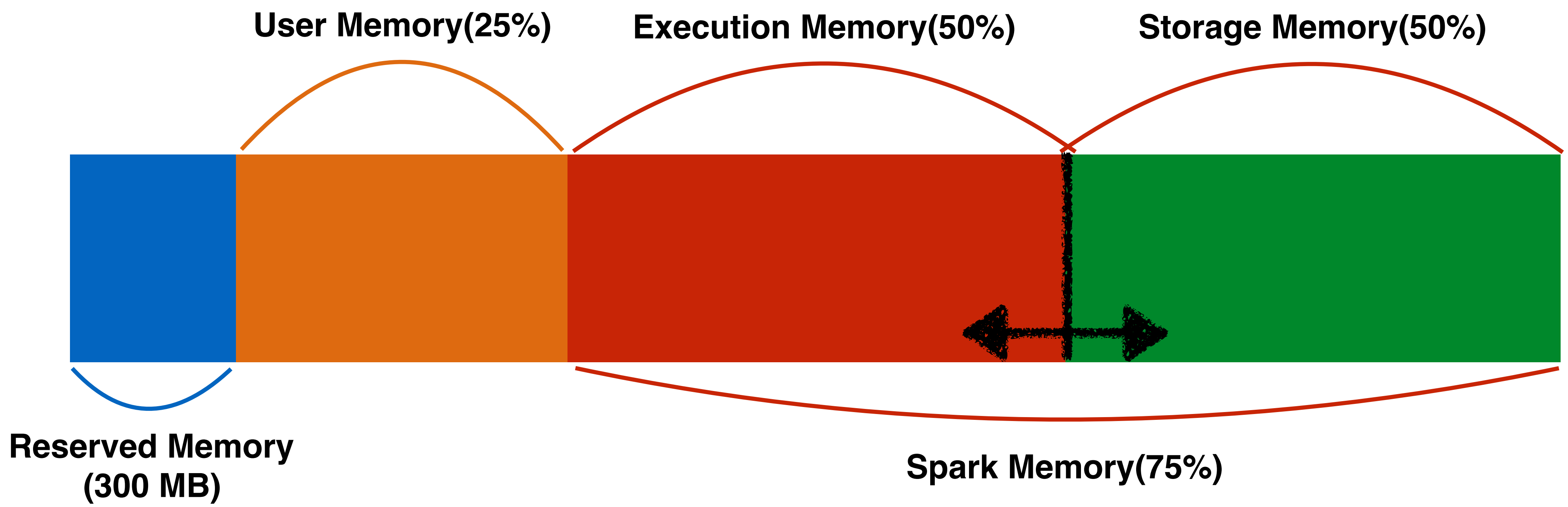


Memory Management Settings



- User Memory: users' data structures used in RDD transformations
- Execution Memory: for Spark's internal storage of objects, e.g., shuffle buffer on the mapper
- Storage Memory: for Spark's cached RDD, broadcast blocks, and serialized data unrolling

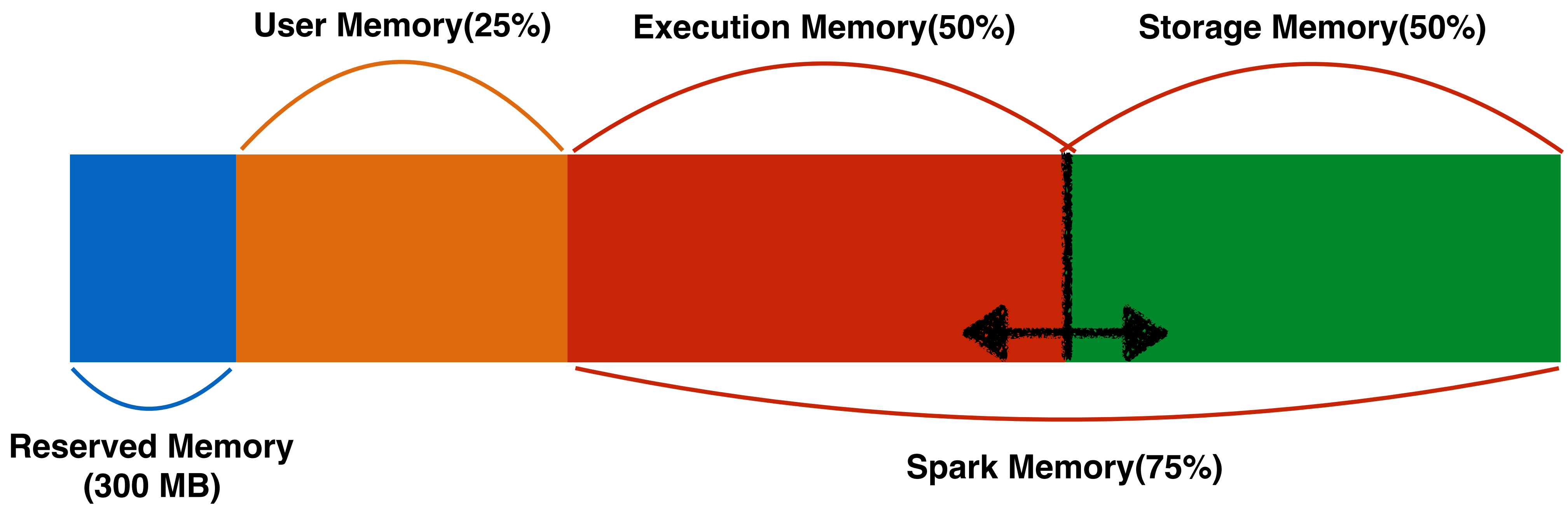
Memory Management Settings



- val rdd = sc.parallelize(List(1,2,3,4,5,6), 2)
- val temp = rdd.mapPartitions(x => {
-   **val m** = x.zipWithIndex.toMap
-   m.toIterator
- })
- **temp.cache**
- val res = temp.**reduceByKey(_+_)**
- res.count

TACC

Memory Management Settings



- In conf/spark-defaults.conf
- spark.testing.reservedMemory — DO NOT USE
- spark.memory.fraction (default 0.75)
- spark.memory.storageFraction (dynamic, default 0.5)

TACC

Solutions to Symptoms

- If Spark is running with insufficient memory
  - Spark is running slow due to garbage collection
  - Executor get lost due to Out-of-Memory (OOM) exception
- The usual solution on Spark standalone cluster
  - Allocate more memory for Spark executor by setting **"export SPARK_WORKER_MEMORY 96g"** in spark-env.sh
  - Increase parallelism
    - Try more reducers by setting "**spark.default.parallelism**" in spark-defaults.conf
    - Set the number of partitions of the largest parent RDD (e.g., **sc.textFile(path, 128)**),
    - Set the second argument of the reduce function (e.g., **PairRDDFunctions.reduceByKey(func, numPartitions)**)

TACC

Solutions to Symptoms

- The usual solution on YARN cluster
  - Allocate more memory for Spark worker by passing "--executor-memory 2g" to the spark-submit command line

```
spark-submit --class org.apache.spark.examples.SparkPi \
    --master yarn \
    --deploy-mode cluster \
    --driver-memory 4g \
    --executor-memory 2g \
    --executor-cores 1 \
    lib/spark-examples*.jar \
    10
```

TACC

Self-checklist

- How does Spark partition a DAG into stages of tasks?
- How does Spark schedule tasks on a cluster?
- How does Spark configure parallelism?
- How does Spark executor manages its memory?