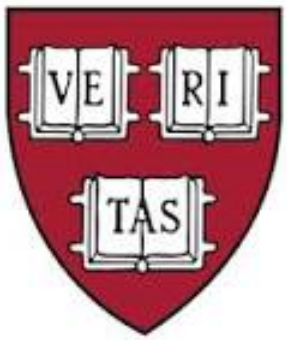


Section 11– Rahul Joglekar

CSCI-E63, Section 11, 04-16-2016

Learning Objectives …

1. Spark Refrersher
2. Spark Debugging – Local and Remote
3. Spark – Do's and Donts
   - WordCount - The Good, The bad , The Ugly
4. Machine Learning – Lay of the land
5. SparkML – Refresher
6. Logistic Regression
7. Code Review

References

Machine Learning with Spark by Nick Pentreath, PAKT Publishing, 2015

Advanced Analytics with Spark, by Sandy Ryza et al., O'Reilly, 2015

Learning Spark , Holden Karau et al,. O'Reilly, 2015

http://www.slideshare.net/hkarau

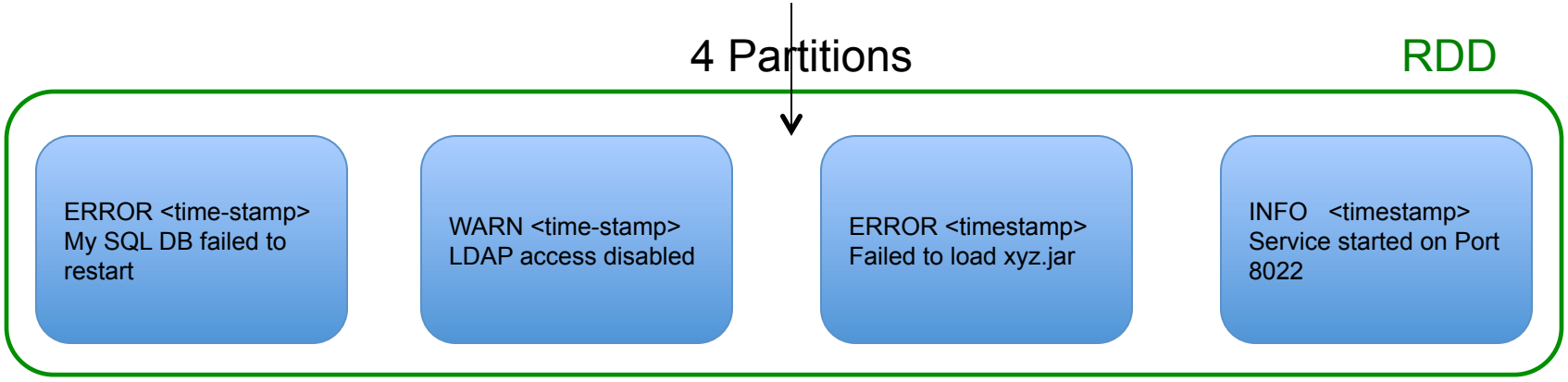
Spark Refresher

File → RDD

Example 1

Example 2

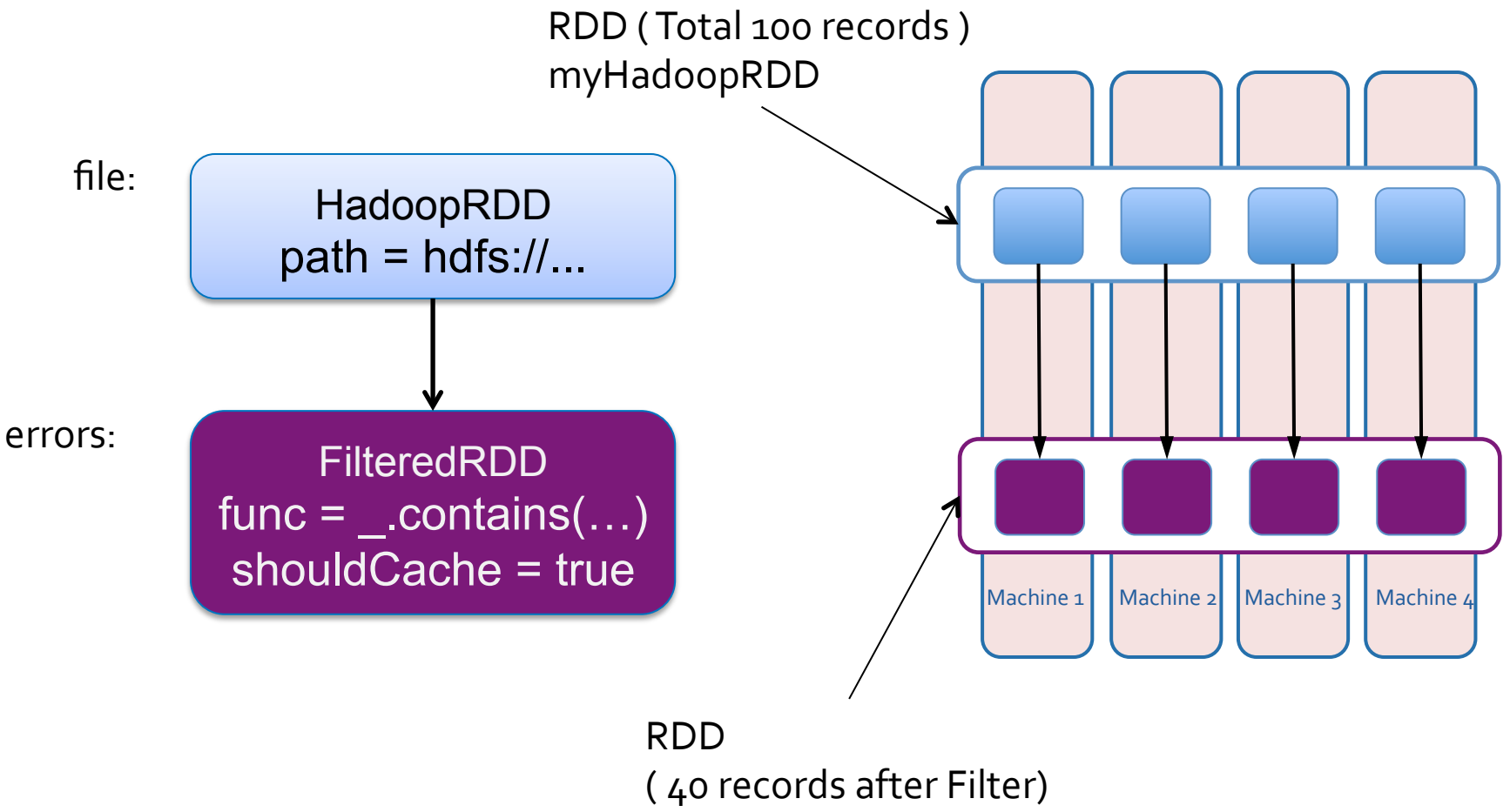
ERROR <time-stamp> My SQL DB failed to restart
WARN <time-stamp> LDAP access disabled
ERROR <timestamp> Failed to load xyz.jar
INFO <timestamp> Service started on Port 8022



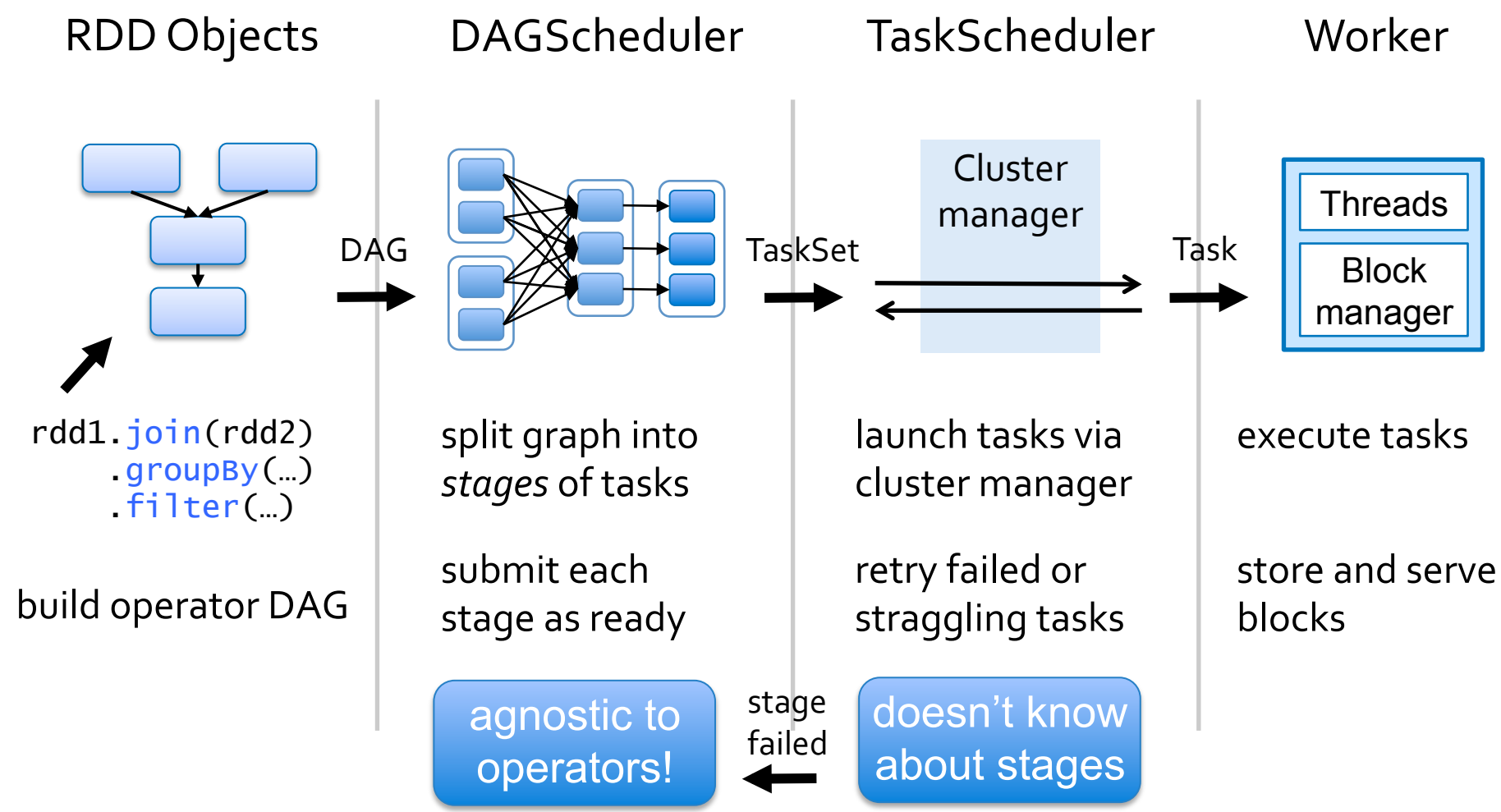
RDD Dataset and Partition View

Dataset-level view:

Partition-level view:



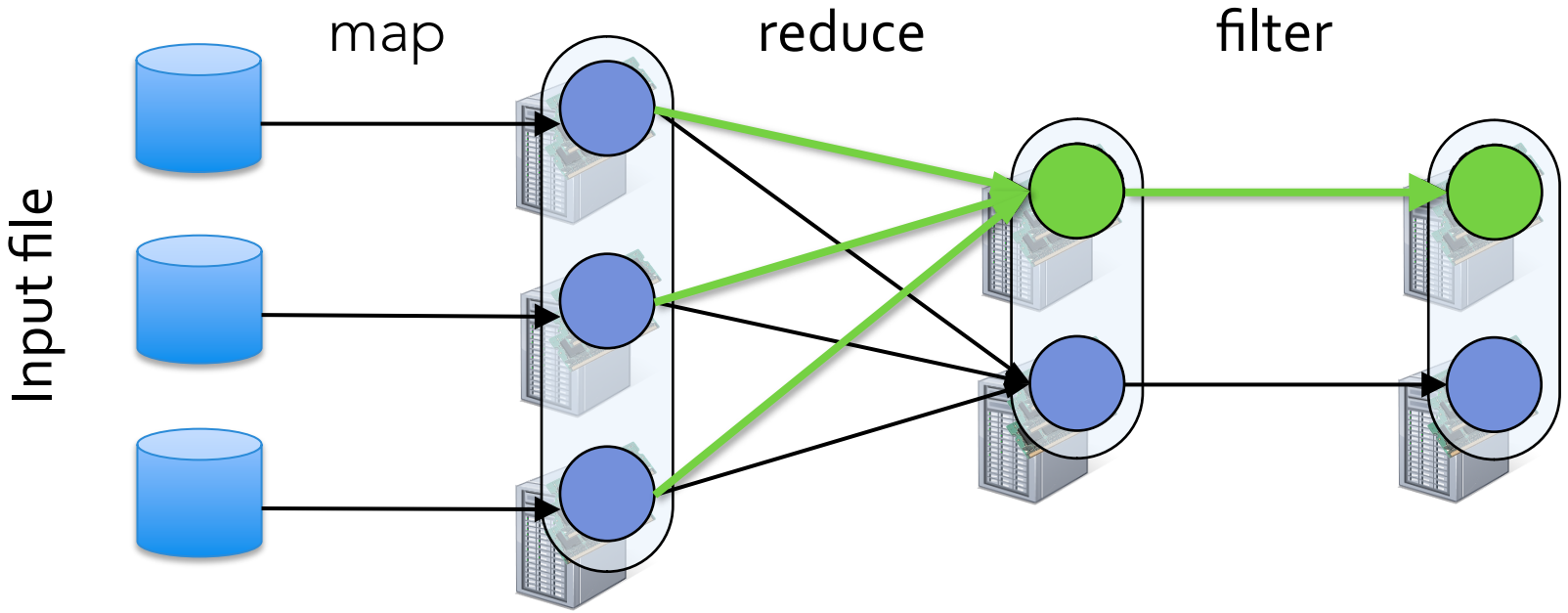
Job Scheduling Process



Fault Tolerance

RDDs track *lineage* info to rebuild lost data

- ```
  file.map(lambda rec: (rec.type, 1))
      .reduceByKey(lambda x, y: x + y)
      .filter(lambda (type, count): count > 10)
  ```

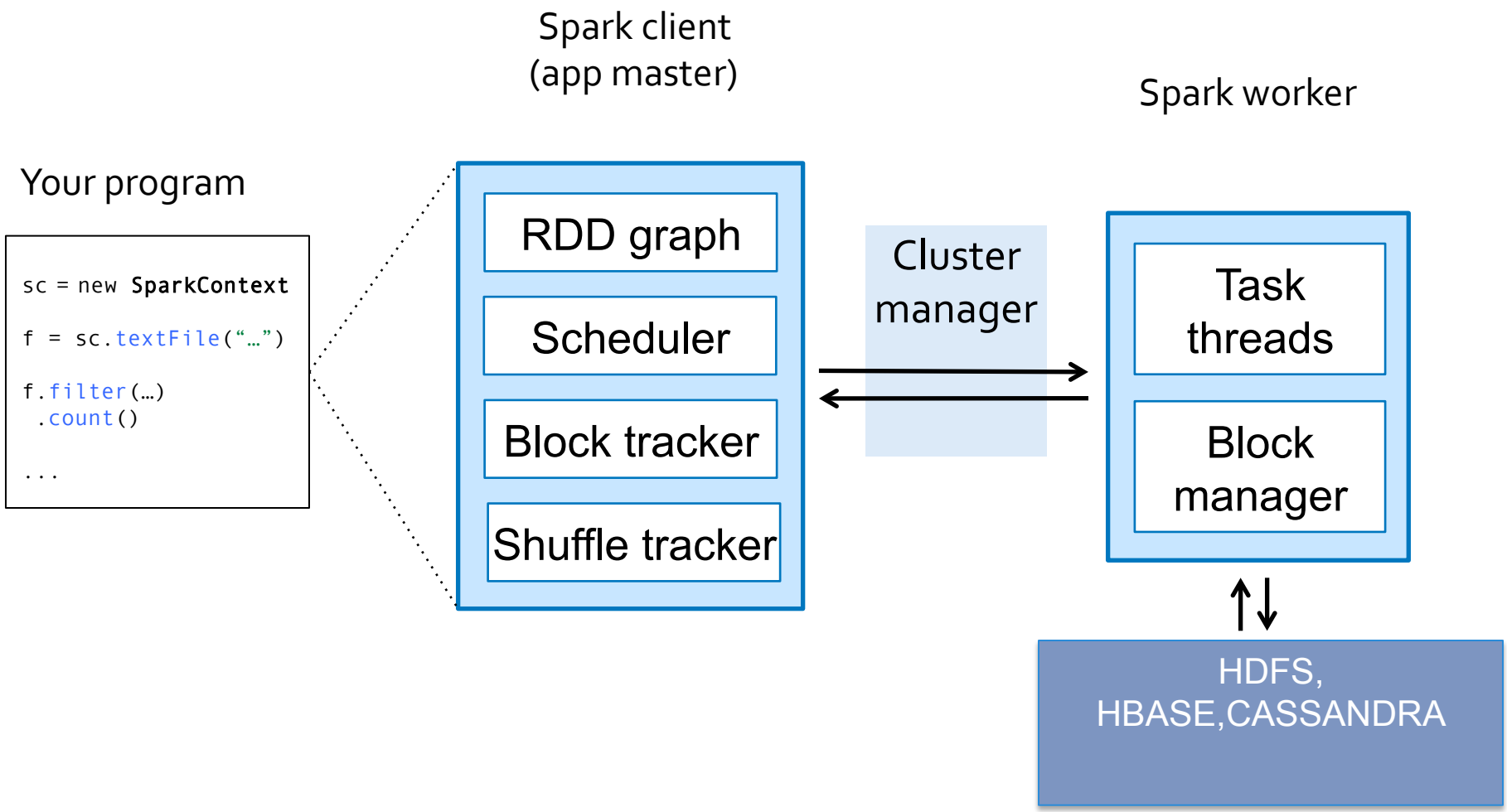

  ```

Execution on the Driver vs. Workers

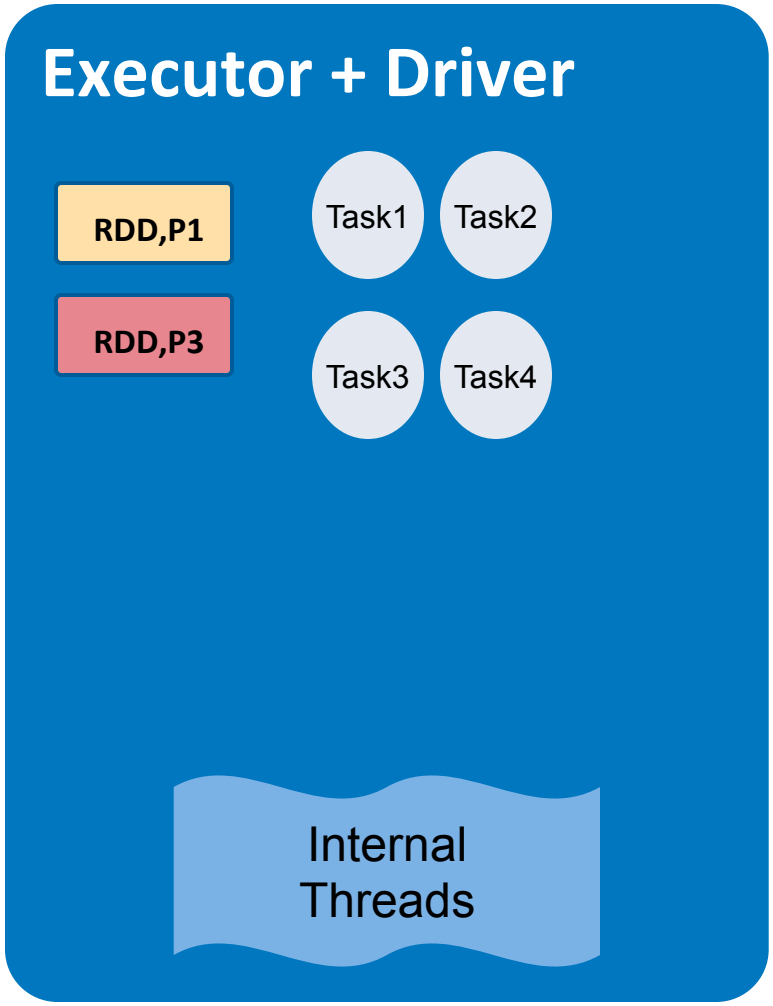
1. The **main program** are executed on the **Spark Driver.**
2. **Transformations** are executed on the **Spark Workers.**
3. **Actions** *may* transfer data from the **Workers** to the **Driver.**

```
1   wcoutput = sparkContext
        .textFile("hdfs://…")
2       .flatMap(lambda line: line.split())
        .map(lambda word: (word, 1))
        .reduceByKey(lambda a, b: a + b)
3       .collect()
    print wcoutput …
```

A Typical Spark Application



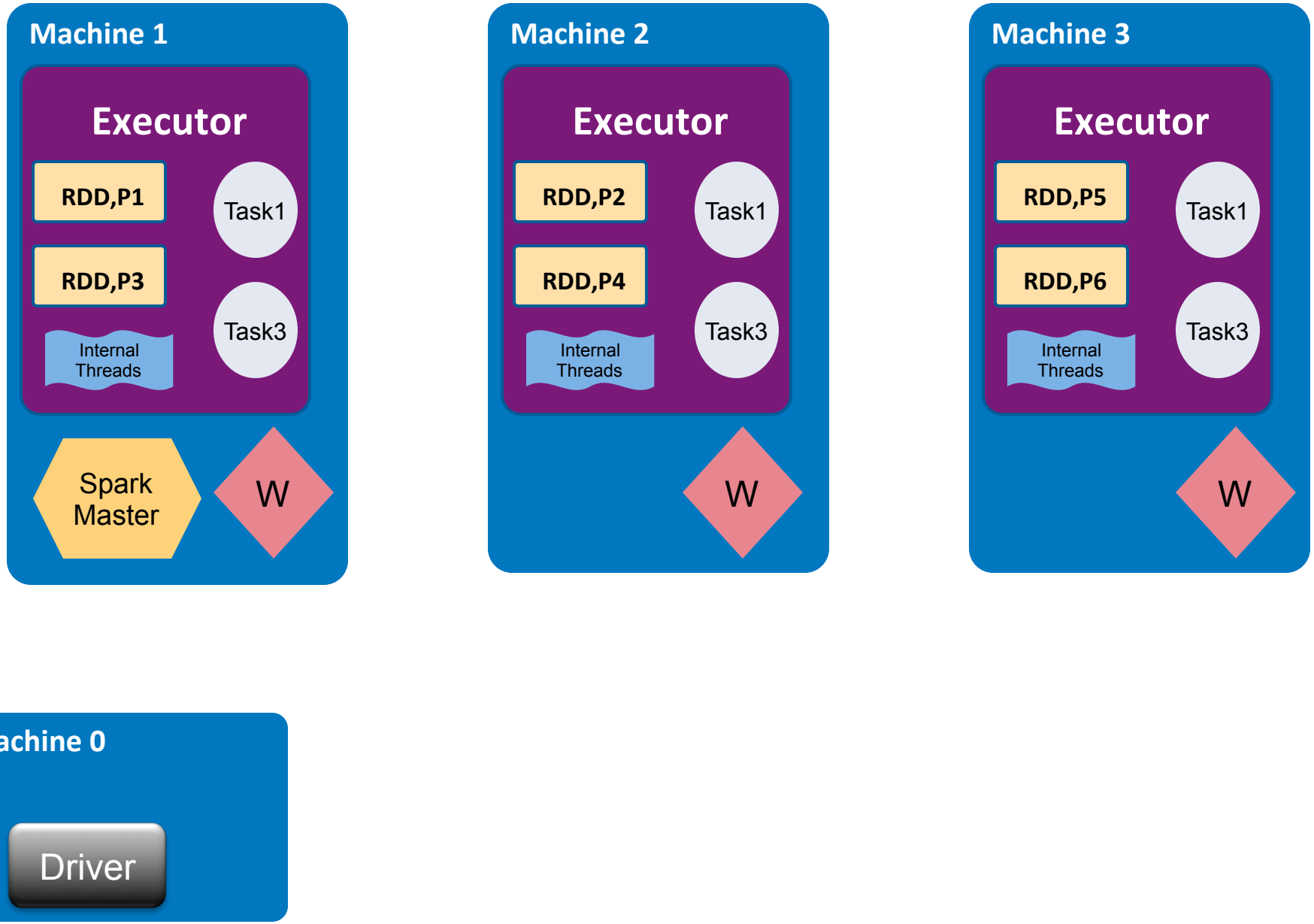
Local Mode



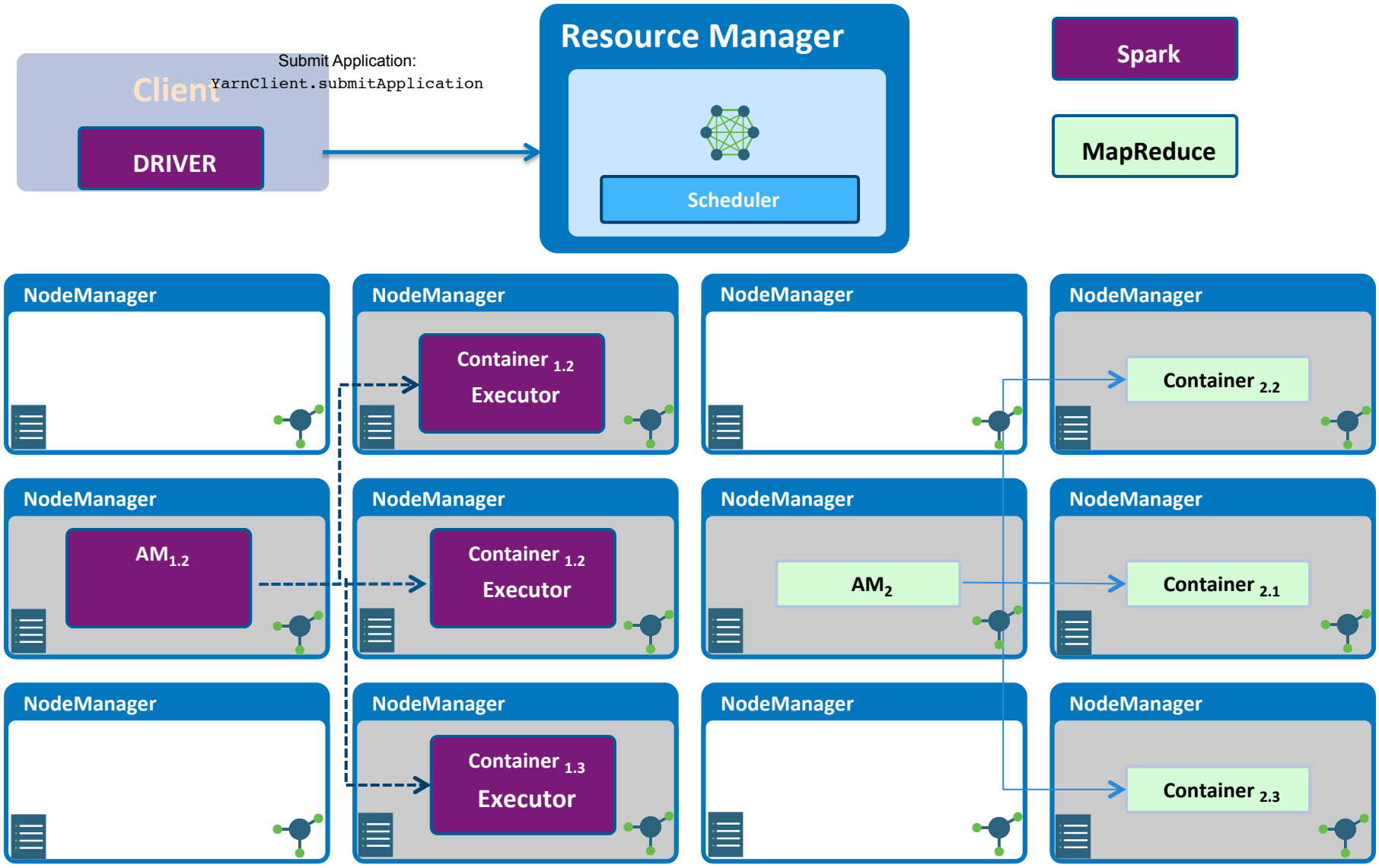
Worker Machine

-local
-local[N]
-local[*]

Standalone mode – Spark Cluster



Spark - Yarn Architecture – Client Mode

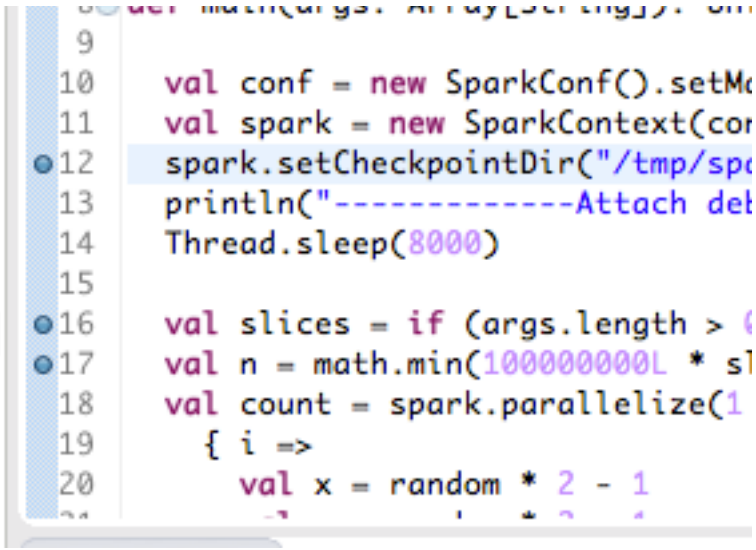


Debugging – Local and Remote

Debugging Spark – Locally in an IDE

Local Debugging

- Run in local mode (i.e. Spark master “local”) and
- debug with your favorite debugger
  - IntelliJ
  - Eclipse
  - println
- With a sample dataset



Eclipse – Local Debug

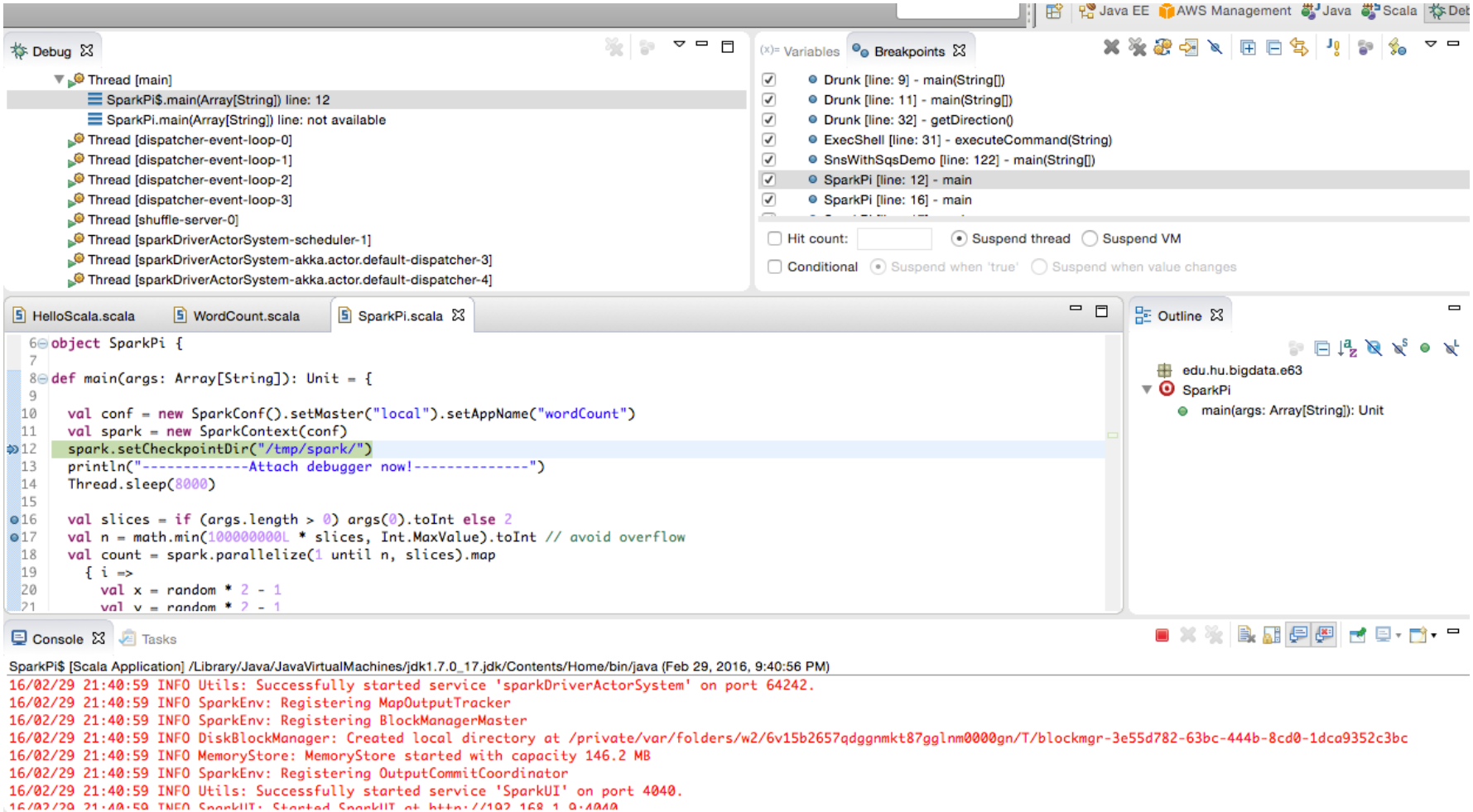
1. Setup break points.

2. Within Eclipse RUN → DEBUG AS → Scala Application

3. Switch perspective to “Debug” and the debuggers stops on the break points (screen shot below)

Debugging Spark – Locally in an IDE

Once the debugger is you will see the cursor stops at the break point



Debug Spark Applications – Remote/Cluster Apps

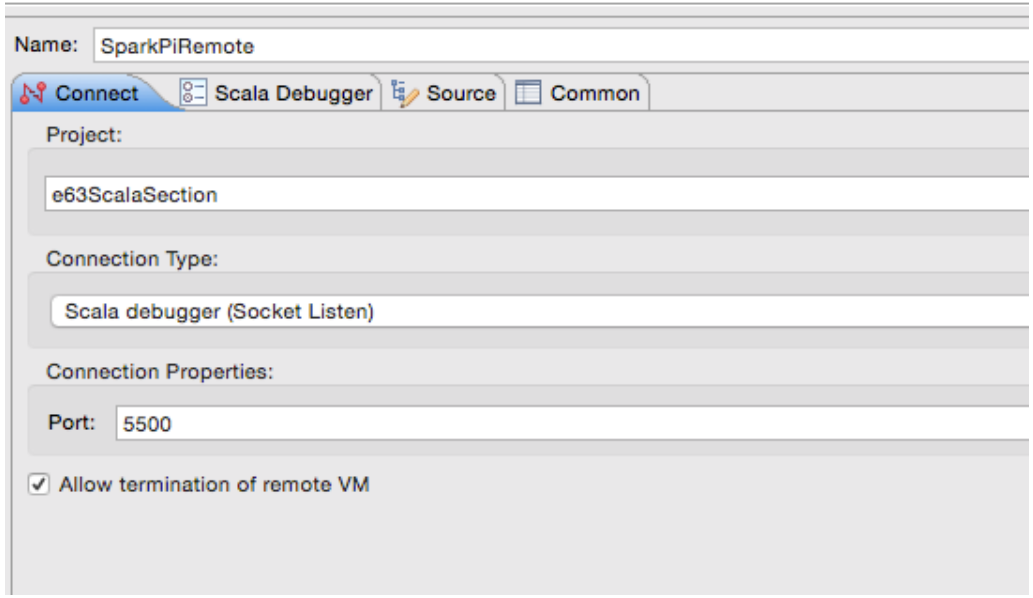
1. Remote Debugging - Assume the application Sparkpi is deployed to Cluster and later found to have has some errors that need debugging. The idea with Remote debugging is you attach eclipse to a Job that you will launch in the cluster and step through the code .

   To do this follow the steps below -
   - Setup a breakpoint in your code (Line #10 below)
   - Assume that SparkPi.jar has already been created and ready to run .

```
8  def main(args: Array[String]): Unit = {
9
10   val conf = new SparkConf().setMaster("local").setAppName("wordCount")
11   val spark = new SparkContext(conf)
12   spark.setCheckpointDir("/tmp/spark/")
13   println("-------------Attach debugger now!--------------")
14   Thread.sleep(8000)
15
16   val slices = if (args.length > 0) args(0).toInt else 2
17   val n = math.min(10000000000L * slices, Int.MaxValue).toInt // avoid overflow
18   val count = spark.parallelize(1 until n, slices).map
19     { i =>
20       val x = random * 2 - 1
21       val y = random * 2 - 1
22       if (x*x + y*y < 1) 1
23         else 0
24     ].reduce( + )
```

Debug Spark Applications – Remote/Cluster Apps

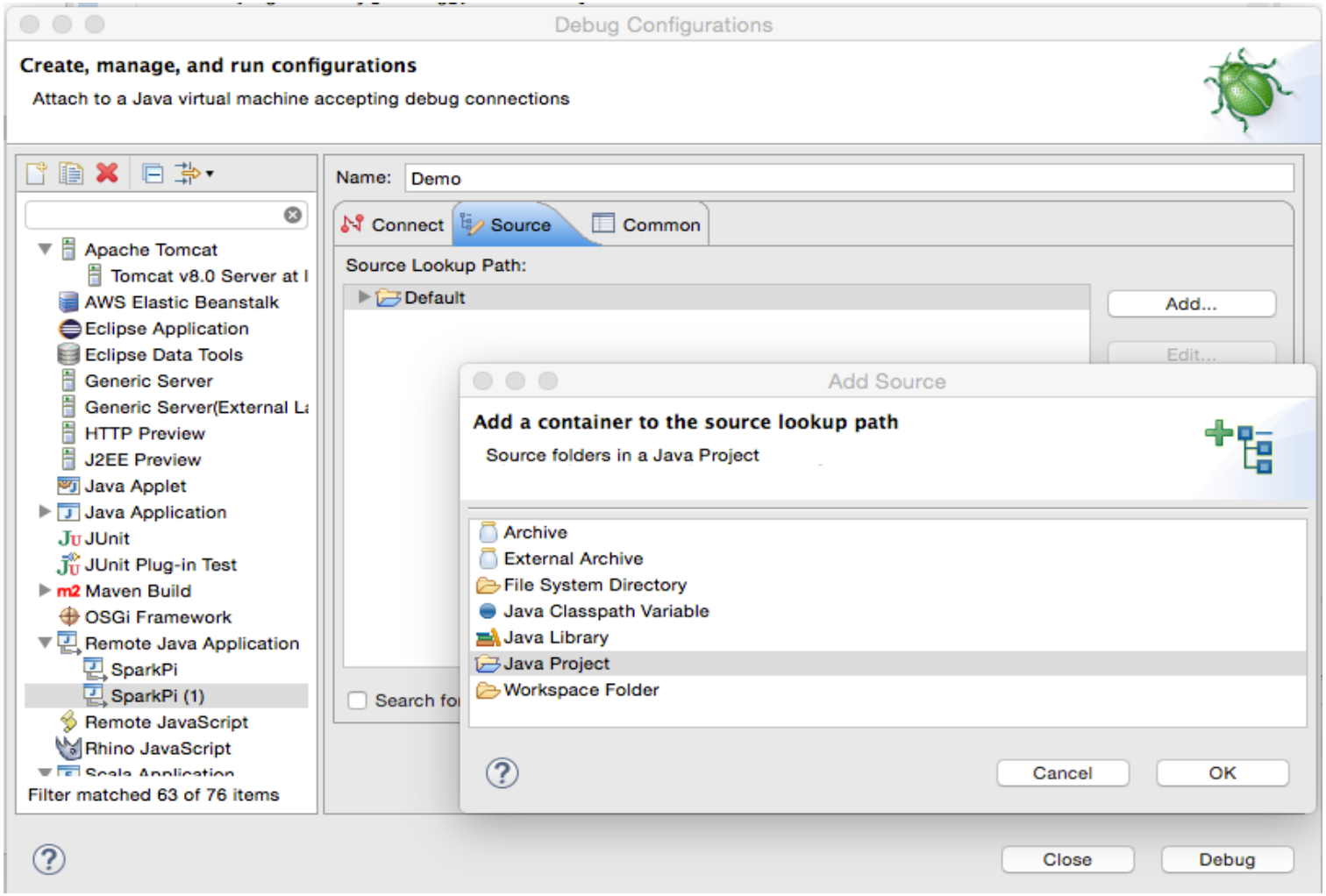
2. We will start Eclipse Debug in the Listen Mode ,
   - go to Eclipse → RUN -> DebugConfig
   - Next select “Remote Java Application” → New (+ button on top)
   - Connection type Socket Listen
   - Specify Port
   - Starts Eclipse in the listen mode , so eclipse accepts connections to it.
   - And we make a connection from the remote JVM in to Eclipse (next slides)



Debug Spark Applications – Remote/Cluster Apps

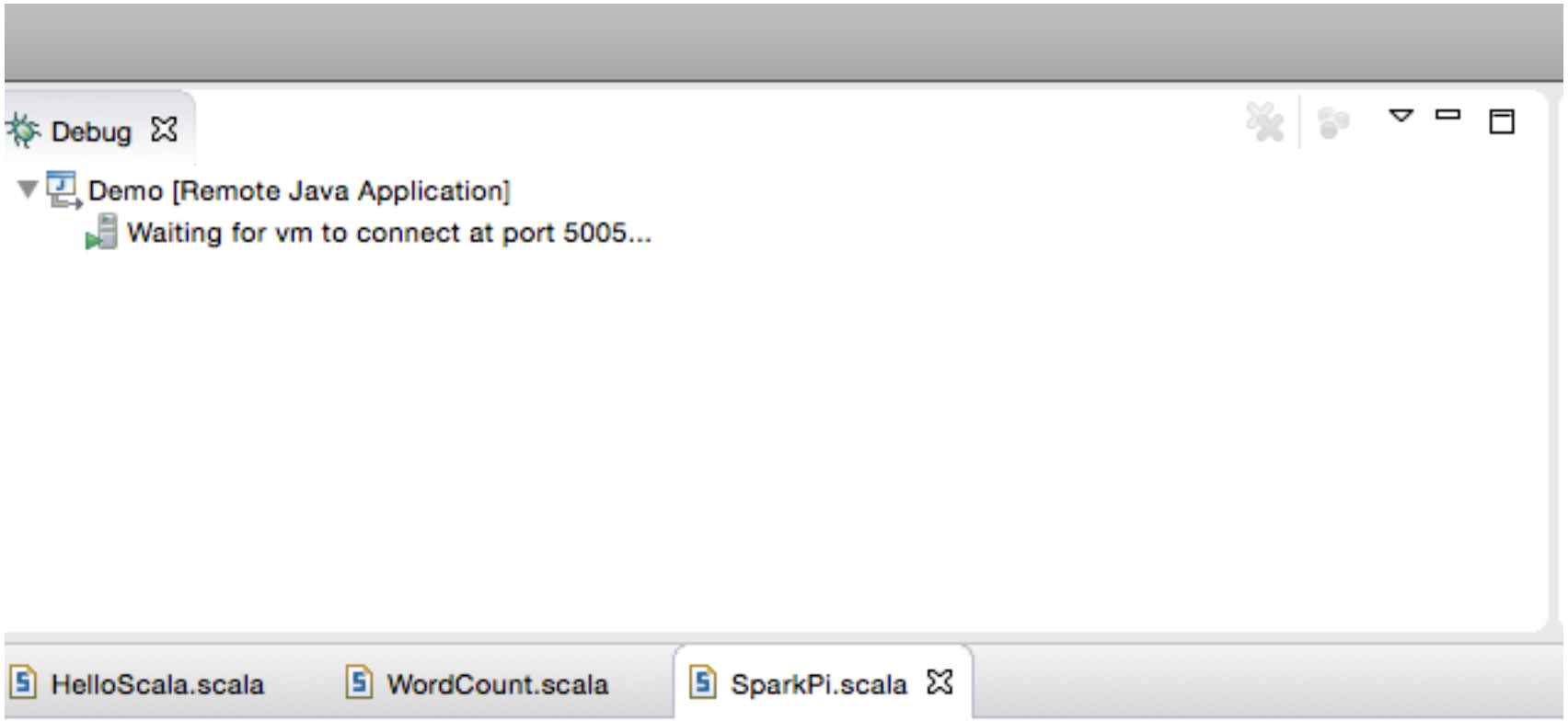
3. Next get right source added , Source Add → Java Project → <Select your project>

4. Next hit Debug



Debug Spark Applications – Remote/Cluster Apps

5. You will see that the Eclipse debugger is listening for the application to make a connection on port 5005



Debug Spark Applications – Remote/Cluster Apps

6. Launch the application

```
$ export SPARK_JAVA_OPTS=-agentlib:jdwp=transport=dt_socket,suspend=y,server=n
address=localhost:5005
```

Make sure “server= n” when eclipse runs in Listen mode

```
$ spark-submit --class edu.hu.bigdata.e63.SparkPi --master local[1] ./ScalaPi-Debug.jar
```

7. Application connects to Eclipse in a debug mode and halts on the break point .

Eclipse Attach mode

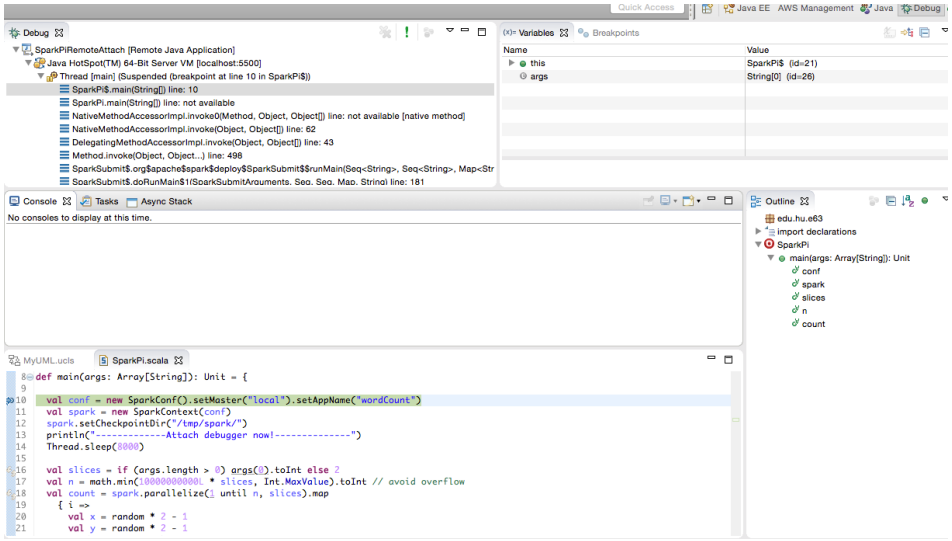
Yet another Other is to attach eclipse to attach eclipse to a running process

$ export SPARK_JAVA_OPTS=-agentlib:jdwp=transport=dt_socket,**suspend=y**,server=y address=localhost:5005

Make sure “server= y” when eclipse runs in Attach mode , here eclipse attaches to whatever port you tell it to attach to.

$ spark-submit --class edu.hu.bigdata.e63.SparkPi --master local[1] ./ScalaPi-Debug.jar

If you Do not set Suspend=y then the application cannot attach



Spark – Do’s and Donts

Shuffle

RDD's and moving Data across network

Often when working with distributed data, it's useful to organize data into key-value pairs. In Spark, these are PairRDDs.

Useful because: Pair RDDs allow you to act on each key in parallel or regroup data across the network.

Spark provides powerful extension methods for RDDs containing pairs (e.g., RDD[(K, V)]).

Some of the most important extension methods are:

```
def groupByKey(): RDD[(K, Iterable[V])]
def reduceByKey(func: (V, V) => V): RDD[(K, V)]
def join[W](other: RDD[(K, W)]): RDD[(K, (V, W))]
```

Depending on the operation, data in an RDD may have to be shuffled among worker nodes, using worker-worker communication. And this may cause inefficiencies

Understanding the Shuffle in Spark

Shuffle is the common cause of inefficiency in Spark and Shuffle is inevitable, we just need to think how do you minimize the adverse impact.

Word Count revisited ☹ - Scala

```
    val sc = new SparkContext(conf)
    // Load our input data.
    val inputRDD = sc.textFile(inputFile)
    // Split up into words.
    val words = inputRDD.flatMap(_.split(" "))
    // Transform into word and count. _+_
    val counts = words.map(word => (word,
1)).reduceByKey { case (x, y) => x + y }
```

When does Shuffle happen ?

Same Worker Node

Triggers a shuffle of data

```
sparkContext.textFile("hdfs://…")
            .flatMap(_.split(" "))
            .map(word => (word, 1))

            .[reduceByKey or groupByKey]
```

Shuffle with a groupByKey

Consider the word count code below which uses “groupByKey”

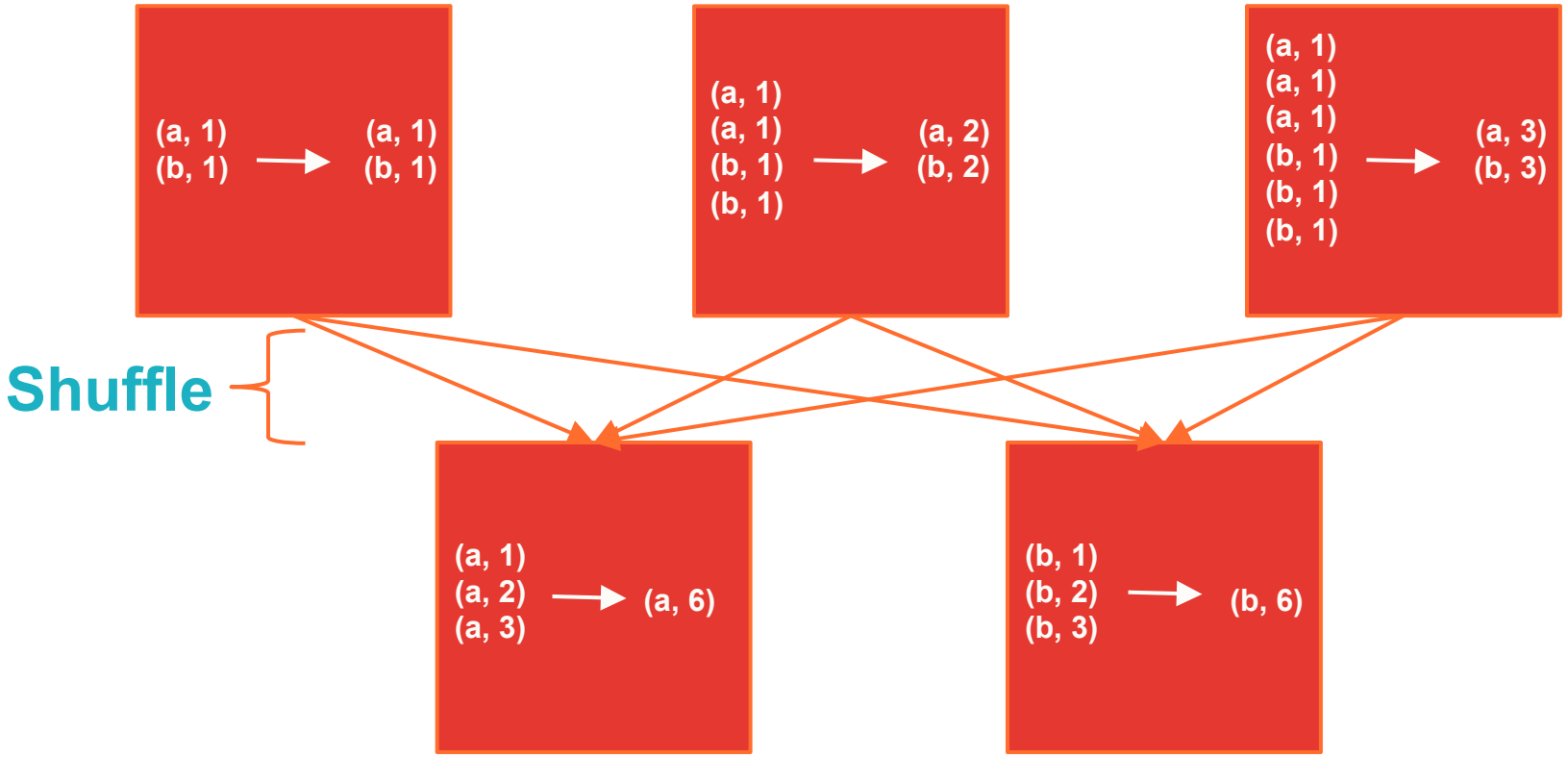
```
    val timerstart = System.currentTimeMillis()
    val conf = new SparkConf().setMaster("local[*]").setAppName("wordCount")
    // Create a Scala Spark Context.
    val sc = new SparkContext(conf)
    // Load our input data.
    val mypartitions=4
    val inputRDD = sc.textFile(inputFile,mypartitions )
    val sze=inputRDD.partitions.size


    //Split up into words - Used groupByKey
    val words = inputRDD
                .flatMap(_.split(" "))
                .map(word=> (word, 1))
                .groupByKey()
                .map {case (w, counts) => (w, counts.sum)}
    words.saveAsTextFile(outputFile)
    val elapsedtime = System.currentTimeMillis() - timerstart
    println ("Size of InputRDD : " +  sze  + "partitions" )
    println("Execution Time Using GroupbyKey : " + elapsedtime/1000.0 + " sec" )
  }
}
```

Problems Tasks Console Debug Properties Terminal

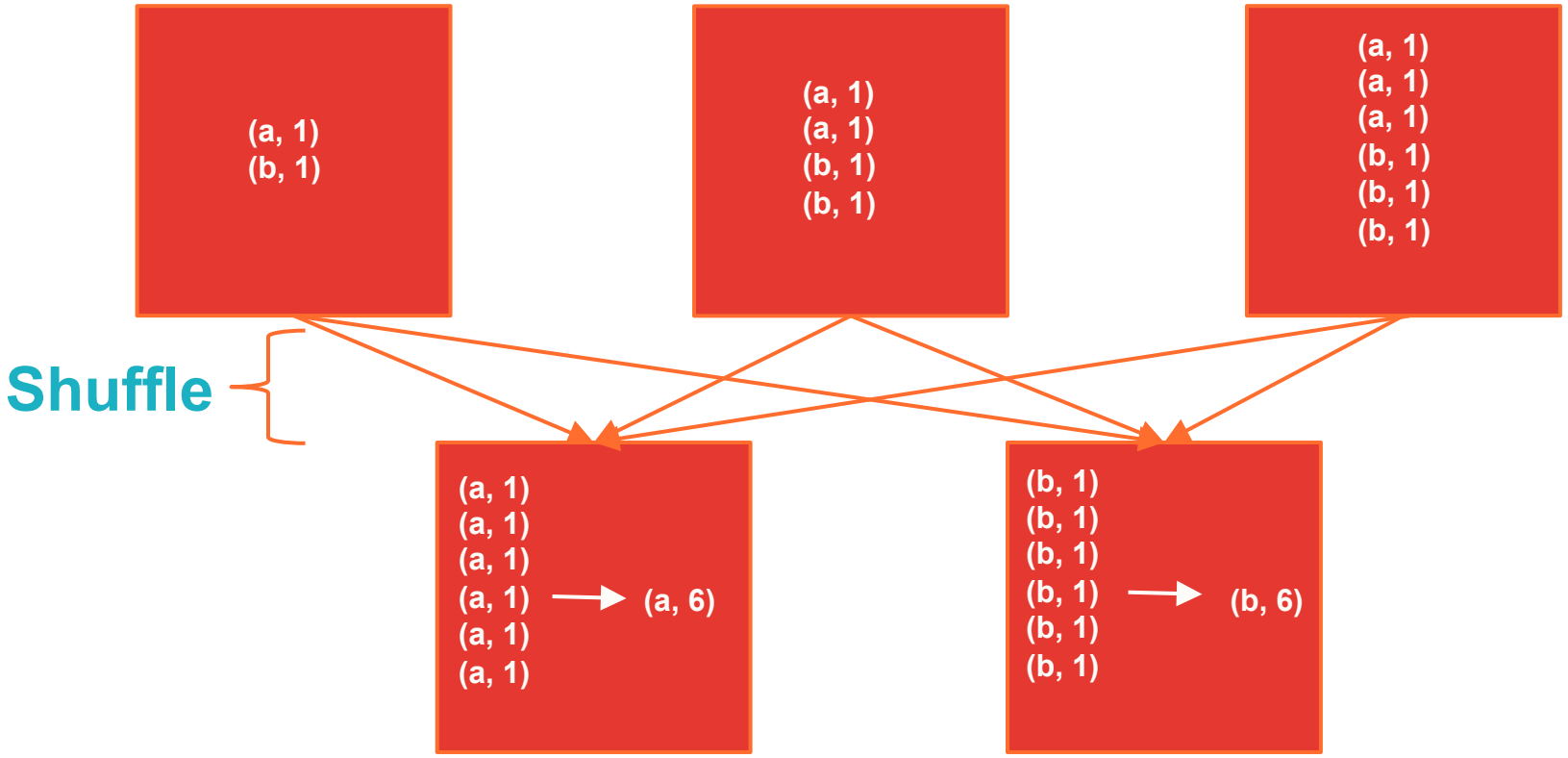
<terminated>SparkPiRemoteAttach [Remote Java Application]

ReduceByKey: Shuffle Step



With ReduceByKey, data is combined so each partition outputs at most one value for each key to send over the network.

groupByKey: Shuffle Step



With GroupByKey, all the data is wastefully sent over the network and collected on the reduce workers.

Prefer ReduceByKey over GroupByKey

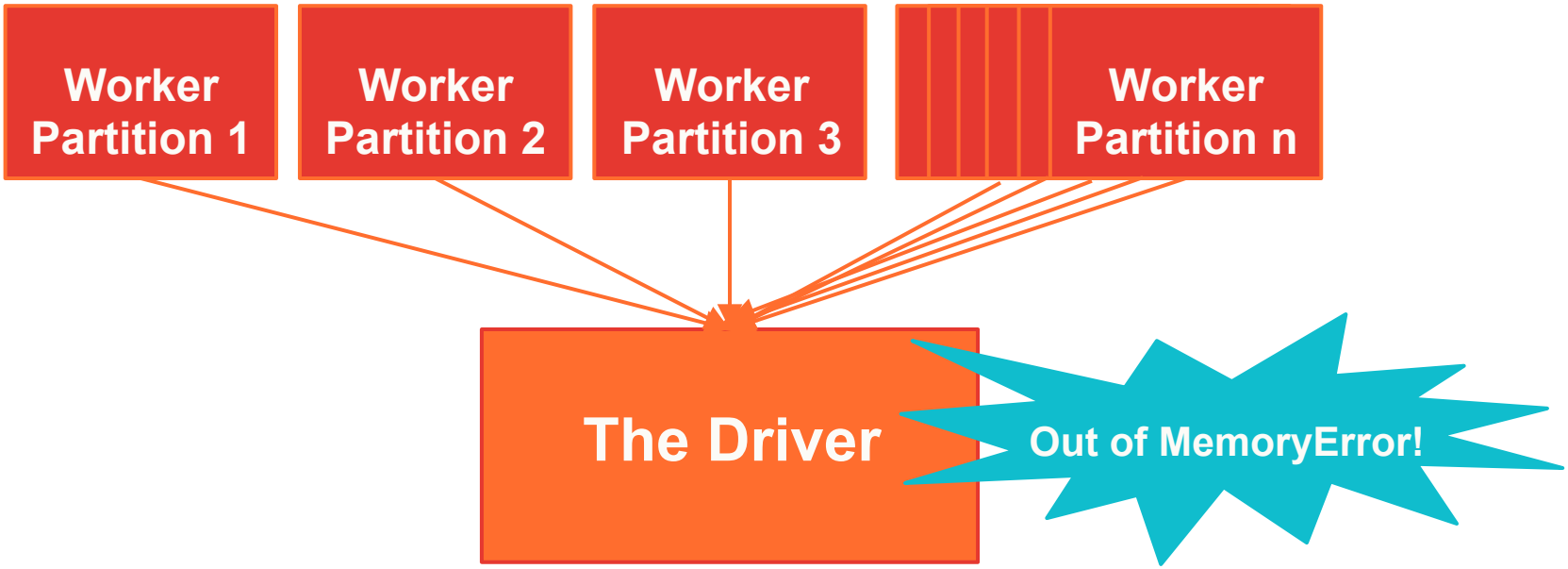
Caveat: Not all problems that can be solved by *groupByKey* can be calculated with *reduceByKey*. ReduceByKey requires combining all your values into another value with the exact same type.

In any case
reduceByKey, aggregateByKey, **foldByKey,** and **combineByKey**, preferred over **groupByKey**

Collect

What happens when calling collect()

collect() sends all the partitions to the single driver



collect() on a large RDD can trigger a OOM error

Don’t call collect() on a large RDD

Be cautious with all actions that may return ***unbounded output***.

Option 1: Choose actions that return a bounded output per partition, such as **count()** or **take(N)** .

Option 2: Choose actions that outputs directly from the workers such as **saveAsTextFile().**

Joining RDD's

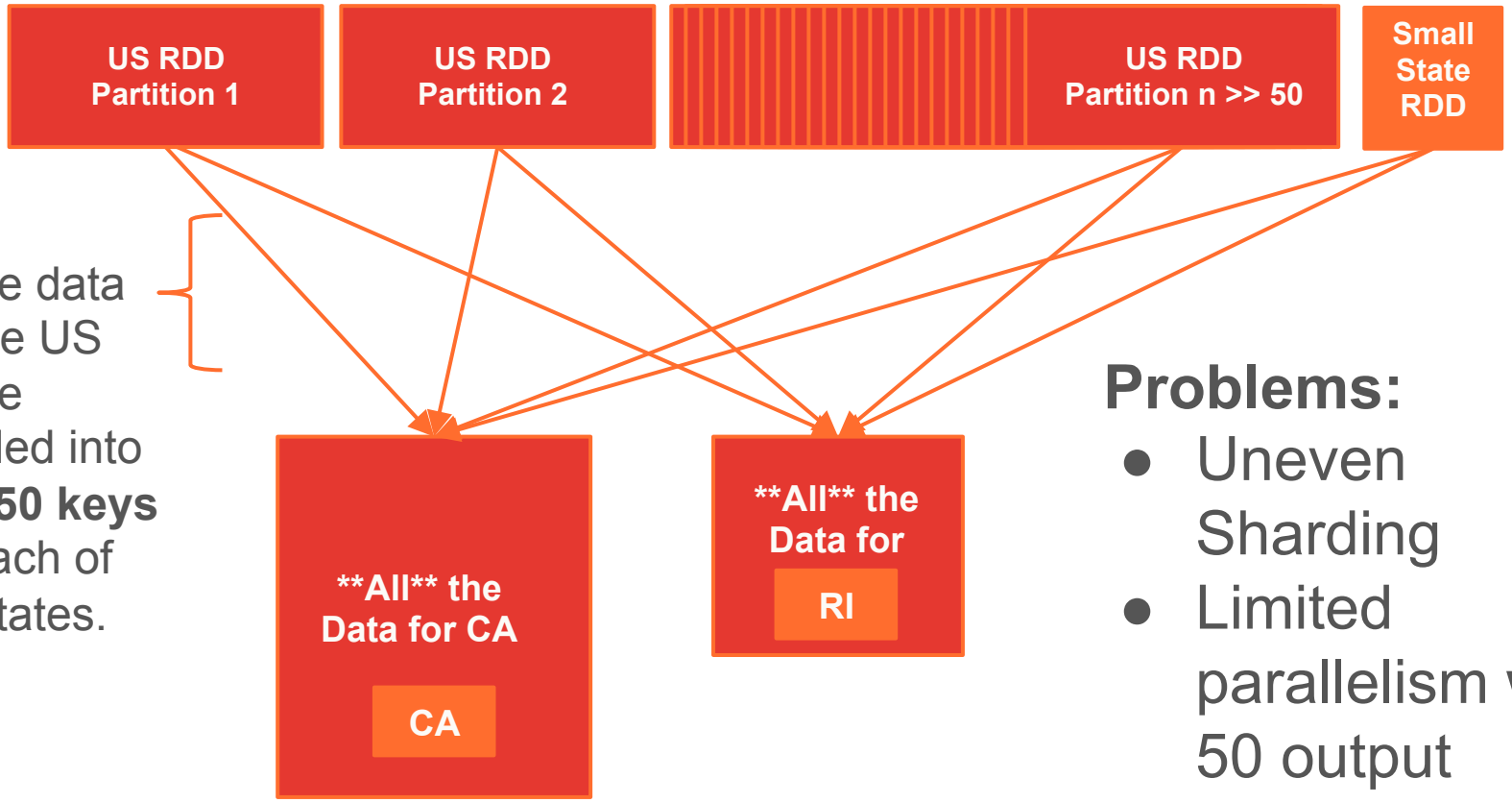
Join a Large Table with a Small Table

```
join_rdd = sqlContext.sql("select *
  FROM people_in_the_us
  JOIN states
  ON people_in_the_us.state = states.name")

print join_rdd.toDebugString()
```

- ShuffledHashJoin?
- BroadcastHashJoin?

ShuffledHashJoin



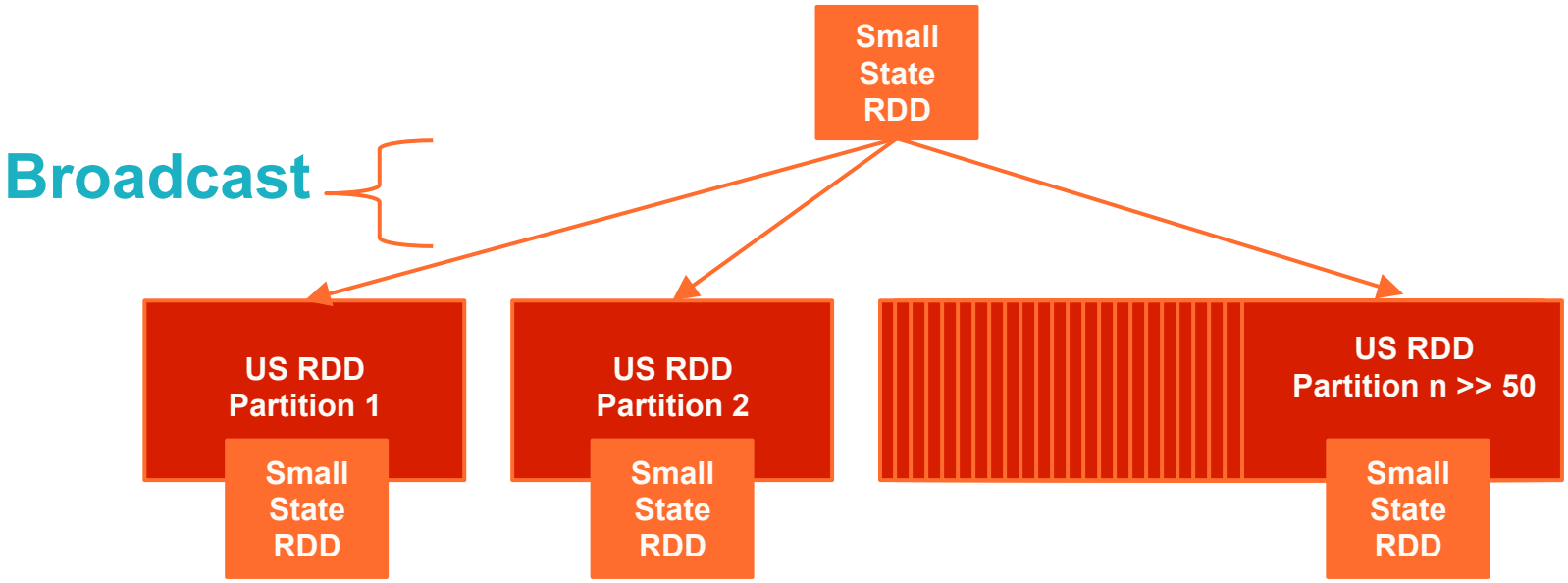
Problems:

- Uneven Sharding
- Limited parallelism w/ 50 output partitions

Even a larger Spark cluster will not solve these problems!

BroadcastHashJoin

Solution: Broadcast the Small RDD to all worker nodes.



Parallelism of the large RDD is maintained (n output partitions), and shuffle is not even needed.

How to Configure BroadcastHashJoin

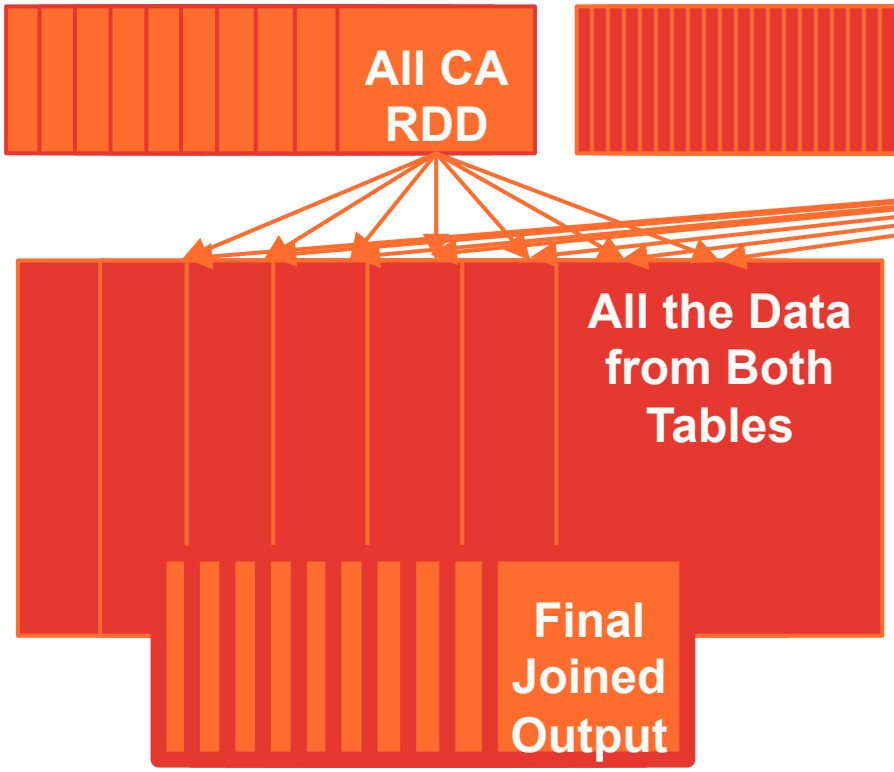
- **See the Spark SQL programming guide for your Spark version for how to configure.**
- **For Spark 1.6:**
  - Set spark.sql.autoBroadcastJoinThreshold.
  - sqlContext.sql("ANALYZE TABLE state_info COMPUTE STATISTICS noscan")
- **Use .toDebugString() or EXPLAIN to double check.**

Join a Medium Table with a Huge Table

```
join_rdd = sqlContext.sql("select *
  FROM people_in_california
  LEFT JOIN all_the_people_in_the_world
  ON people_in_california.id =
     all_the_people_in_the_world.id")
```

Final output keys = keys people_in_california, so this don't need a huge Spark cluster, right?

Left Join - Shuffle Step



Shuffles everything before dropping keys

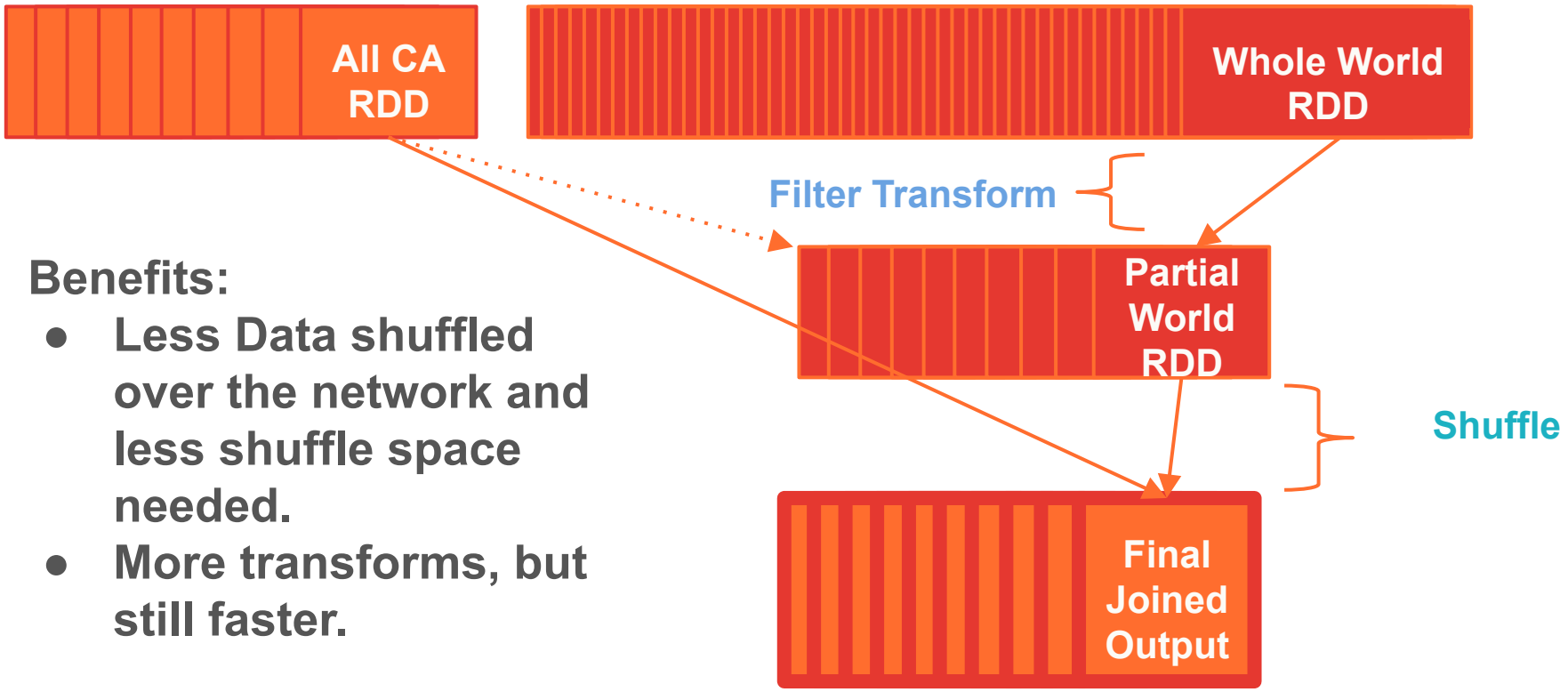
Not a Problem:

- Even Sharding
- Good Parallelism

The Size of the Spark Cluster to run this job is limited by the Large table rather than the Medium Sized Table.

What's a Better Solution?

Filter the WholeWorld RDD for only entries that match the CA ID



Benefits:

- Less Data shuffled over the network and less shuffle space needed.
- More transforms, but still faster.

What's the Tipping Point for Huge?

- There aren't always strict rules for optimizing.
- If you were only considering two small columns from the World RDD in Parquet format, the filtering step may not be worth it.

You should understand your data and it's unique properties in order to best optimize your Spark Job.

In Practice: Detecting Shuffle Problems

Tasks

Index	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	Scheduler Delay	Task Deserialization Time	GC Time	Result Serialization Time	Getting Result Time	Input	Write Time	Shuffle Write	Errors
0	23	0	SUCCESS	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:32	9 s	10 ms	3 ms	0.1 s	0 ms	0 ms	60.9 MB (hadoop)	12 ms	5.1 MB	
1	24	0	SUCCESS	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:32	9 s	10 ms	1 ms	0.1 s	0 ms	0 ms	56.4 MB (hadoop)	14 ms	4.7 MB	
3	26	0	SUCCESS	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:32	8 s	12 ms	1 ms	0.1 s	0 ms	0 ms	54.3 MB (hadoop)	15 ms	4.3 MB	
2	25	0	SUCCESS	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:32	9 s	11 ms	1 ms	0.1 s	0 ms	0 ms	54.5 MB (hadoop)	11 ms	4.4 MB	
4	27	0	SUCCESS	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:40	8 s	7 ms	1 ms	0.1 s	0 ms	0 ms	54.0 MB (hadoop)	11 ms	4.4 MB	
5	28	0	SUCCESS	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:40	8 s	8 ms	1 ms	0.1 s	0 ms	0 ms	53.7 MB (hadoop)	10 ms	4.3 MB	
6	29	0	SUCCESS	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:41	7 s	7 ms	1 ms	0.1 s	0 ms	0 ms	54.0 MB (hadoop)	10 ms	4.4 MB	
7	30	0	SUCCESS	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:41	8 s	7 ms	1 ms	0.1 s	0 ms	0 ms	54.4 MB (hadoop)	10 ms	4.3 MB	
8	31	0	RUNNING	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:47	6 s	0 ms	0 ms		0 ms	0 ms				
9	32	0	RUNNING	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:48	5 s	0 ms	0 ms		0 ms	0 ms				
10	33	0	RUNNING	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:48	5 s	0 ms	0 ms		0 ms	0 ms				
11	34	0	RUNNING	PROCESS_LOCAL	0 / ip-10-0-187-189.us-west-2.compute.internal	2015/02/03 22:03:49	4 s	0 ms	0 ms		0 ms	0 ms				

Check the Spark UI pages for task level detail about your Spark job.

Things to Look for:

- Tasks that take much longer to run than others.
- Speculative tasks that are launching.
- Shards that have a lot more input or shuffle output than others.

What About Parallelism?

Parallalism

Can be specified in a number of ways

–RDD partition number
- sc.textFile("input", minSplits = 10)
- sc.parallelize(1 to 10000, numSlices = 10)

–Mapper side parallelism
- Usually inherited from parent RDD(s)

–Reducer side parallelism
- rdd.reduceByKey(_ + _, numPartitions = 10)
- rdd.reduceByKey(partitioner = p, _ + _)

Zoom In and Zoom Out

–RDD.repartition(numPartitions: Int)
–RDD.coalesce(numPartitions: Int,shuffle: Boolean)

A Trap of Partitions

Consider this example

```
sc.textFile("input", minSplits = 2)
            .map {     line =>
                       val Array(key, value) = line.split(",")
                       key.toInt -> value.toInt
                   }
            .reduceByKey(_ + _)
            .saveAsText("output")
```

- Run in local mode
- 60+GB input file

What happens ??

A Trap of Partitions

Consider this example

```
sc.textFile("input", minSplits = 2)
            .map {     line =>
                       val Array(key, value) = line.split(",")
                       key.toInt -> value.toInt
                  }
            .reduceByKey(_ + _)
            .saveAsText("output")
```

- Run in local mode
- 60+GB input file

Actually, this may generate 4,000,000 temporary files…WHY ???

A Trap of Partitions

Relook at our example

```
sc.textFile("input", minSplits = 2)
```

- 1 partition per HDFS split
- Similar to mapper task number in Hadoop MapReduce, the minSplitsparameter is only a hint forsplit number
- Actual split number is also controlled by some other properties like minSplitSize and default FS block size
- In this case, the final split size equals to local FS block size, which is 32MB by default, and 60GB / 32MB ≈2K
- ReduceByKey generates $2K^2$ shuffle outputs = 4M

A Trap of Partitions – So what should I do ?

Consider this example

```
sc.textFile("input", minSplits = 2).coalesce(2)
            .map {     line =>
                       val Array(key, value) = line.split(",")
                       key.toInt -> value.toInt
                  }
            .reduceByKey(_ + _)
            .saveAsText("output")
```

Use RDD.coalesce() to control partition number precisely.

Machine Learning

What is ML


CUSTOMERS WHO BOUGHT THIS ITEM:
ALSO BOUGHT:
piccolo
demotivateur.fr

ML Application

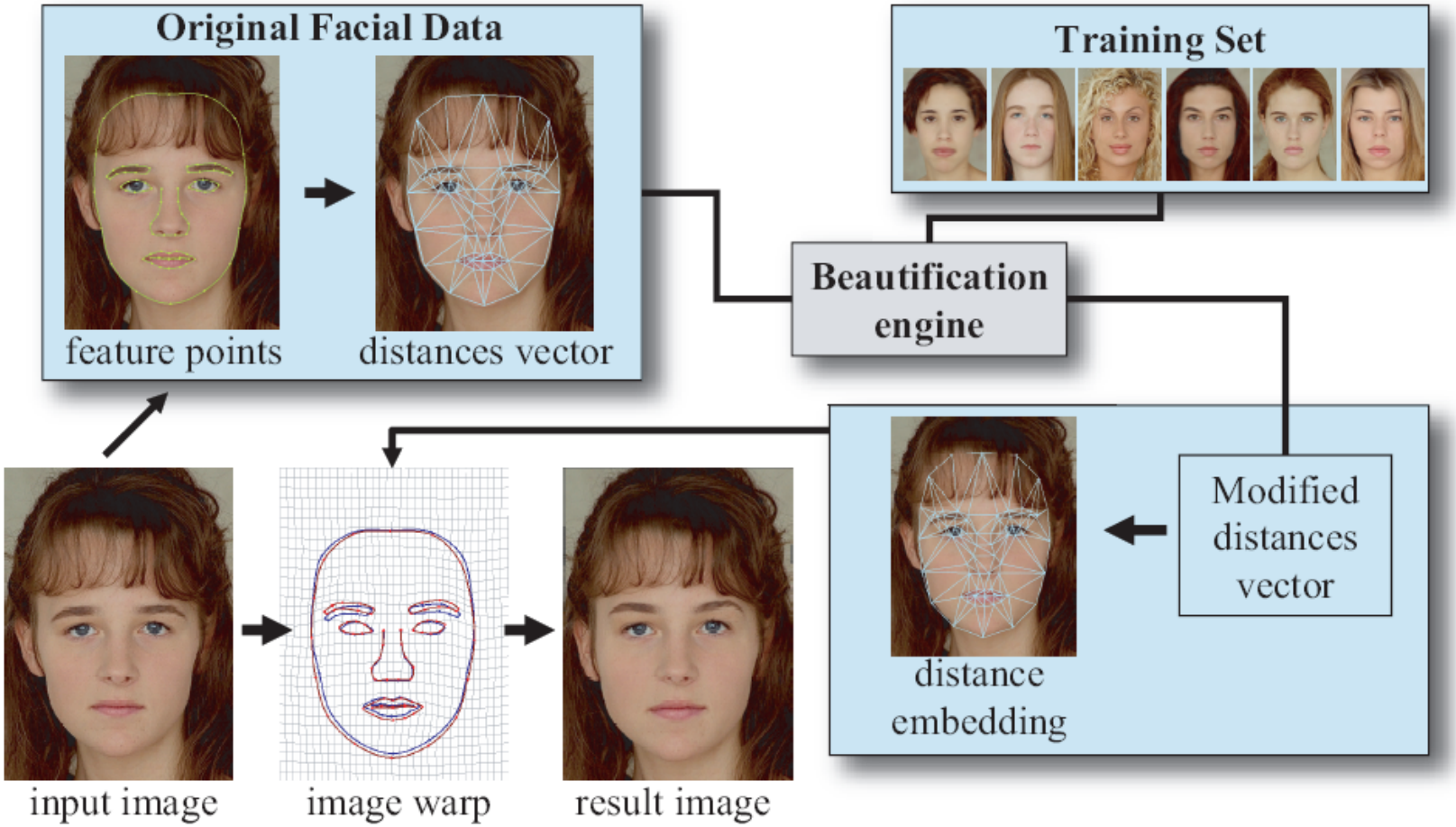
From left to right: original face, KNN-beautified with K = 3, KNN-beautified with optimal K, SVR-beautified.

http://www.leyvand.com/beautification2008/attractiveness2008.pdf

And Fiona

http://www.leyvand.com/beautification2008/attractiveness2008.pdf

Face Beautification



http://www.leyvand.com/beautification2008/attractiveness2008.pdf

Why ML ?

The Far Future—Coming Soon

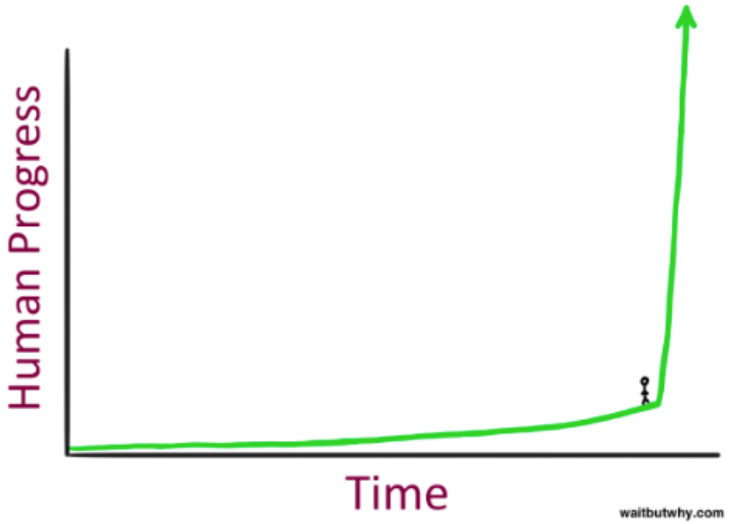
We are on the edge of change comparable to the rise of human life on Earth. – Vernor Vinge (Prof, SDSU)

AI is fast becoming a reality

- ANI (Narrow)
- AGI (General)
- ASI (Super Intelligence)

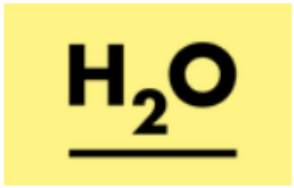
We are currently in ANI stage and to go to AGI and ASI

- More compute power
- Better algorithms and system



http://www.ted.com/talks/jeremy_howard_the_wonderful_and_terrifying_implications_of_computers_that_can_learn?language=en
Reference: http://waitbutwhy.com/2015/01/artificial-intelligence-revolution-1.html

Tools in Land of ML



VOWPAL WABBIT

…….And many more

ML and Big Data .. Where is this headed

1. New use cases push towards faster execution platforms and real time predictions engines.

2. Traditional MapReduce on Hadoop is fading away, especially for Machine Learning

3. Apache Spark has become the top Player of the Scalable ML Platforms, thanks to its high level API and performances.

4. Rise of Machine Learning public APIs to easily integrate models into application and other data processing workflows.

Mahout

Moved from MapReduce towards modern and faster backends, namely “Spark” and “H20”

Now provide a fluent DSL that integrates with Scala and Spark

MLLib

Machine Learning library within Spark :

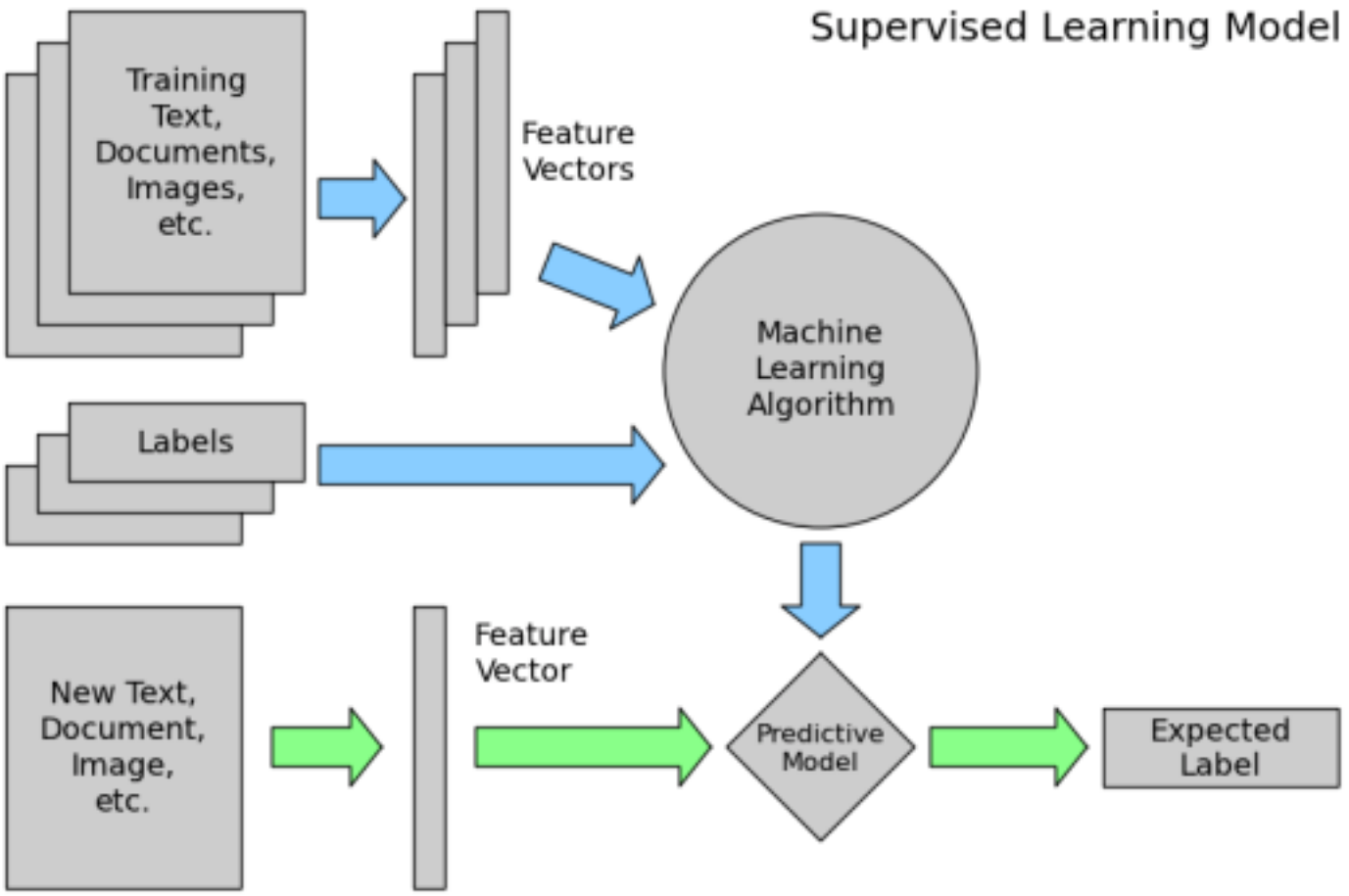
- Provides an integrated predictive and data analysis workflow
- Broad collections of algorithms and applications
- Integrates with the whole Spark Ecosystem

Three APIs in :

Lambda
(Java 8)

interactive shell (scala & python)

ML Workflow


Supervised Learning Model
Training Text, Documents, Images, etc.
Feature Vectors
Machine Learning Algorithm
Labels
New Text, Document, Image, etc.
Feature Vector
Predictive Model
Expected Label

H20.ai

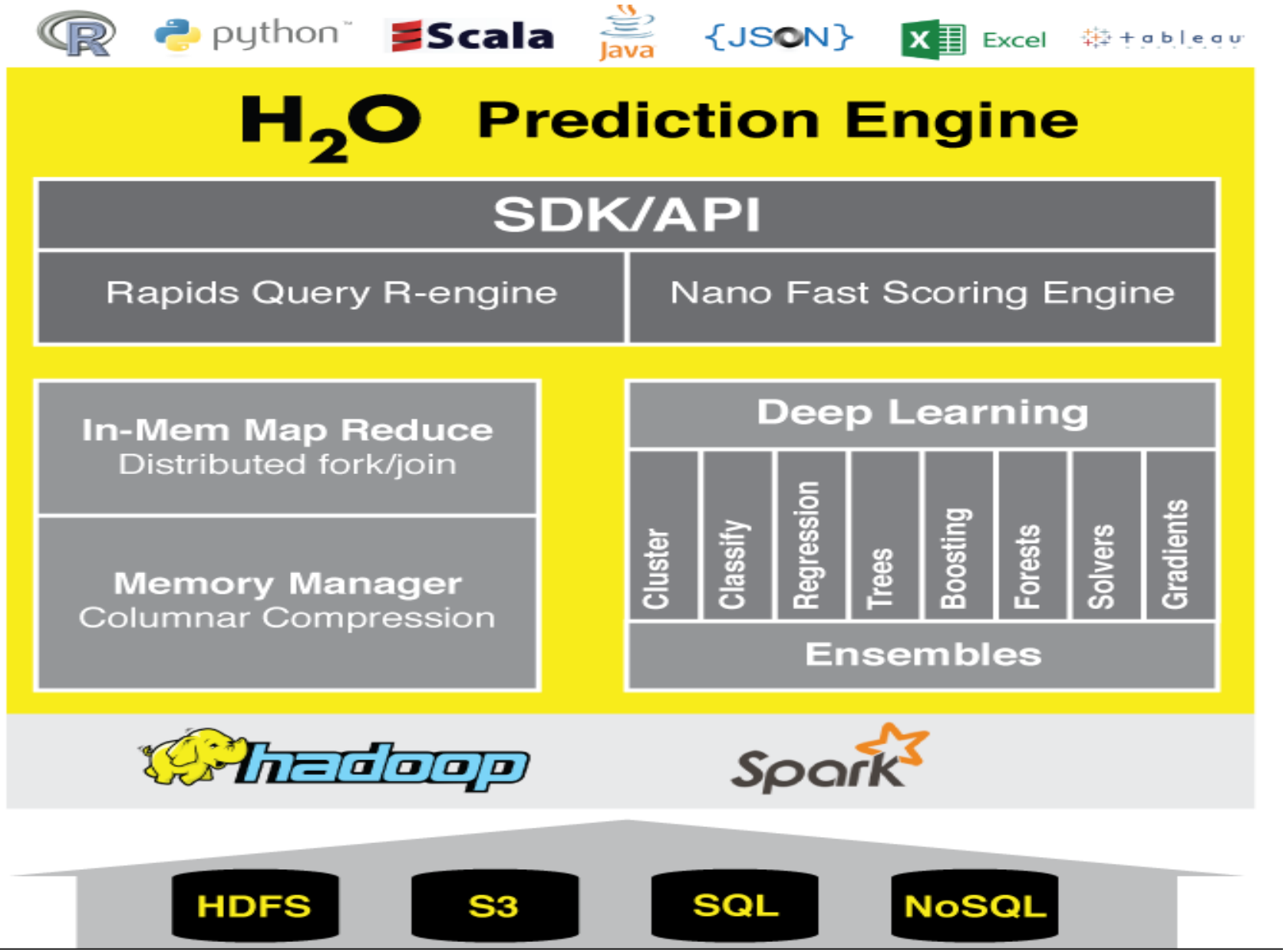
H_2O

- $H_2$0 is a fast (really fast), statistics, Machine Learning and maths engine on the JVM.

- Edited by 0xdata (commercial entity) and focus on bringing robust and highly performant machine learning algorithms to popular Big Data workloads.

- Has APIs in R, Java, Scala and Python and integrates to third parties tools like Tableau and Excel.

H20 Architecture



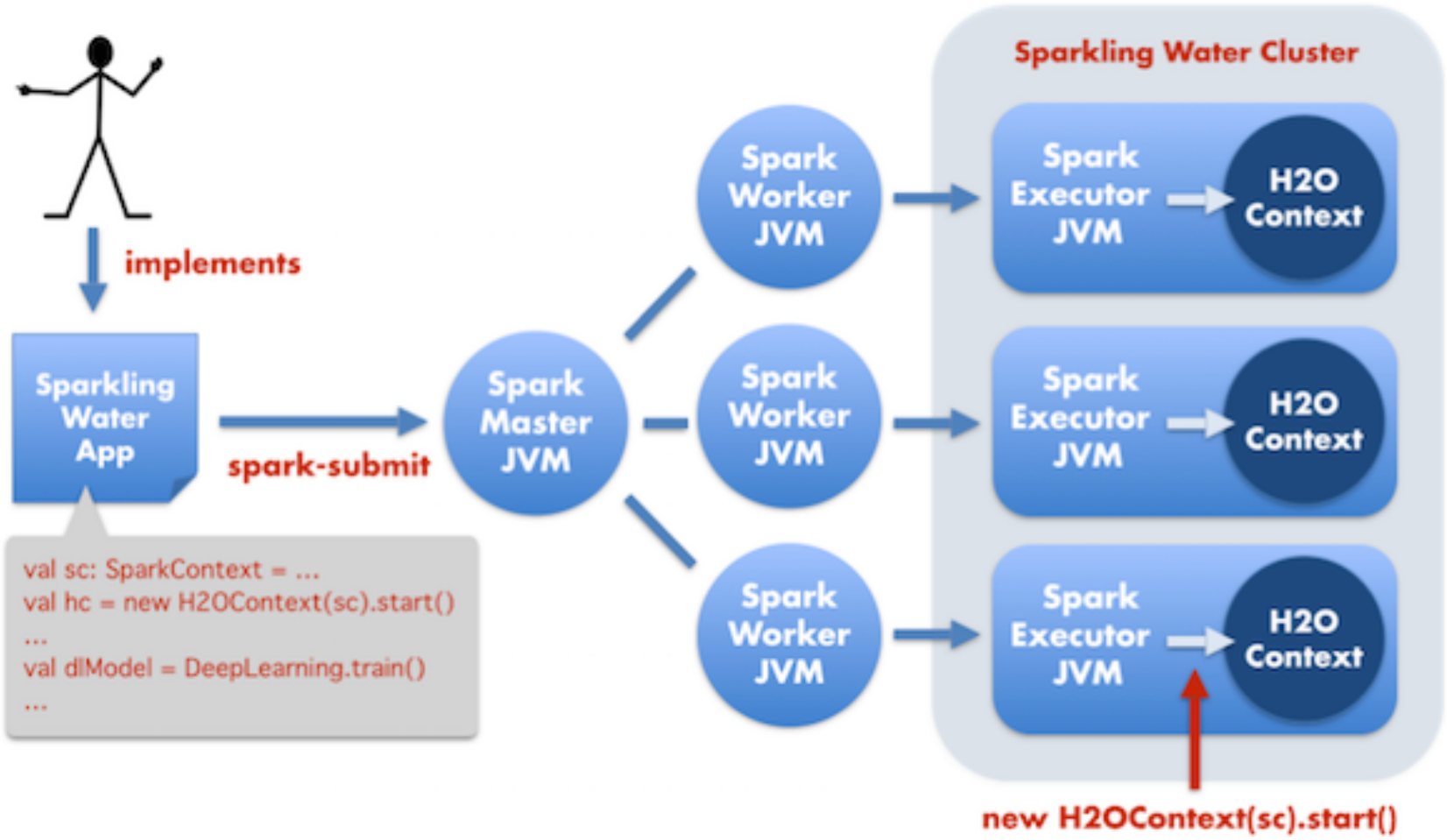
http://www.h2o.ai/product/

Sparkling Water

Transparent use of H2O data and algorithms with the Spark API.

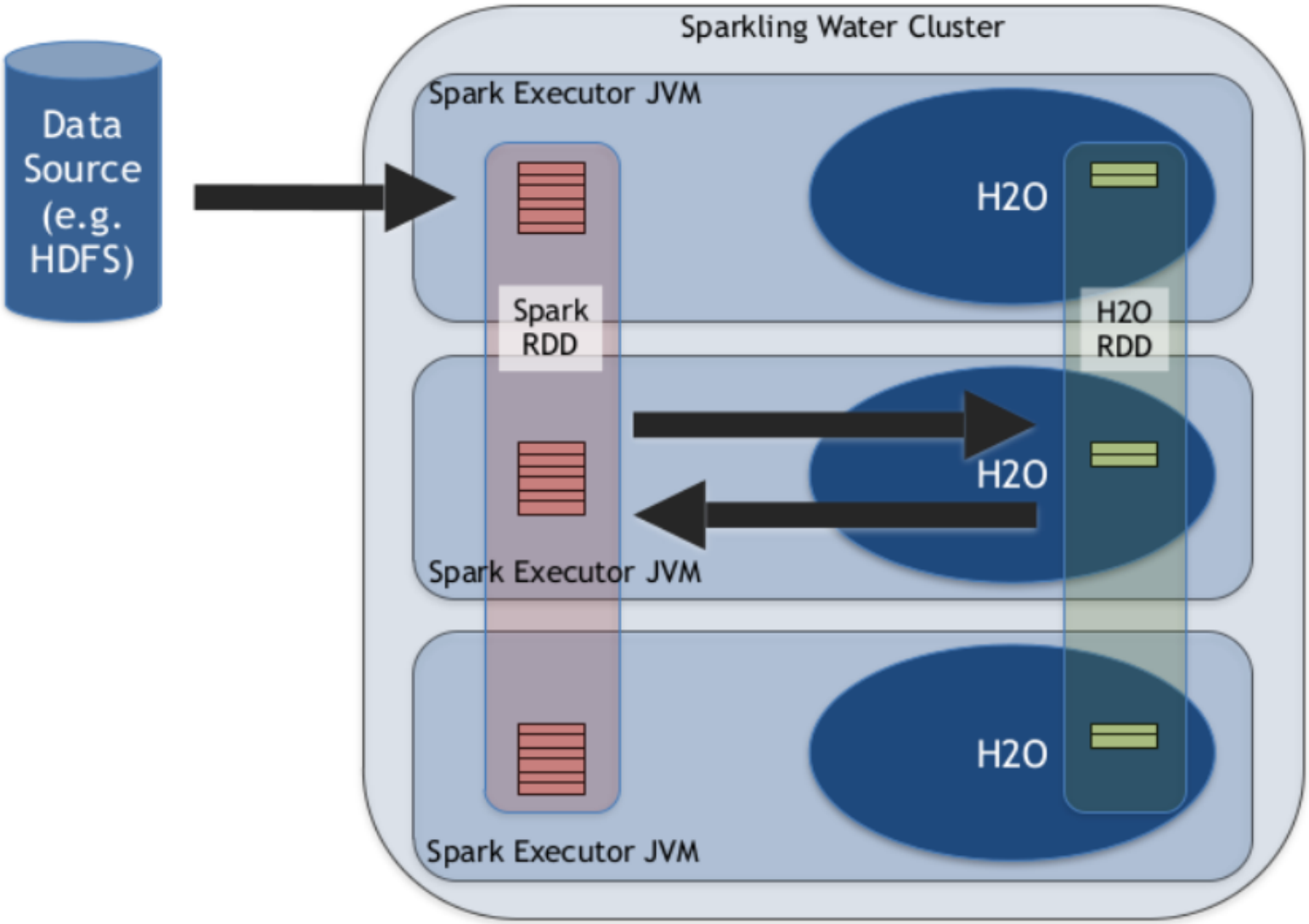
Provides a custom RDD : H2ORDD

High Level Architecture



http://blog.cloudera.com/blog/2015/10/how-to-build-a-machine-learning-app-using-sparkling-water-and-apache-spark/

H20RDD Vs {Other}RDD



http://www.slideshare.net/0xdata/2015-02-19stratatalk

Spark MLLib

- Data Types –

1. Local vector

```
import org.apache.spark.mllib.linalg.{Vector, Vectors}
>val dv: Vector = Vectors.dense(1.0, 0.0, 3.0)
>val sv1: Vector = Vectors.sparse(3, Array(0, 2), Array(1.0, 3.0))
```

2. Labeled point

```
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPointval
>val pos = LabeledPoint(1.0, Vectors.dense(1.0, 0.0, 3.0))
```

3. Local matrix

```
>val dm: Matrix = Matrices.dense(3, 2, Array(1.0, 3.0, 5.0, 2.0, 4.0, 6.0))
```

4. Distributed matrix
   - RowMatrix
   - IndexedRowMatrix
   - CoordinateMatrix
   - BlockMatrix

Spark MLLib

- Basic Statistics

1. Summary statistics
2. Correlations
3. Stratified sampling
4. Hypothesis testing
5. Streaming Significance Testing
6. Random data generation
7. Kernel density estimation

- **Feature Extraction and Transformation**

1. TF-IDF
2. Word2Vec
3. StandardScaler
4. Normalizer
5. ChiSqSelector
6. ElementwiseProduct
7. PCA

MLLib

- **Dimensionality Reduction**

  1. SVD
  2. PCA

- Algorithms – Classification & Regression

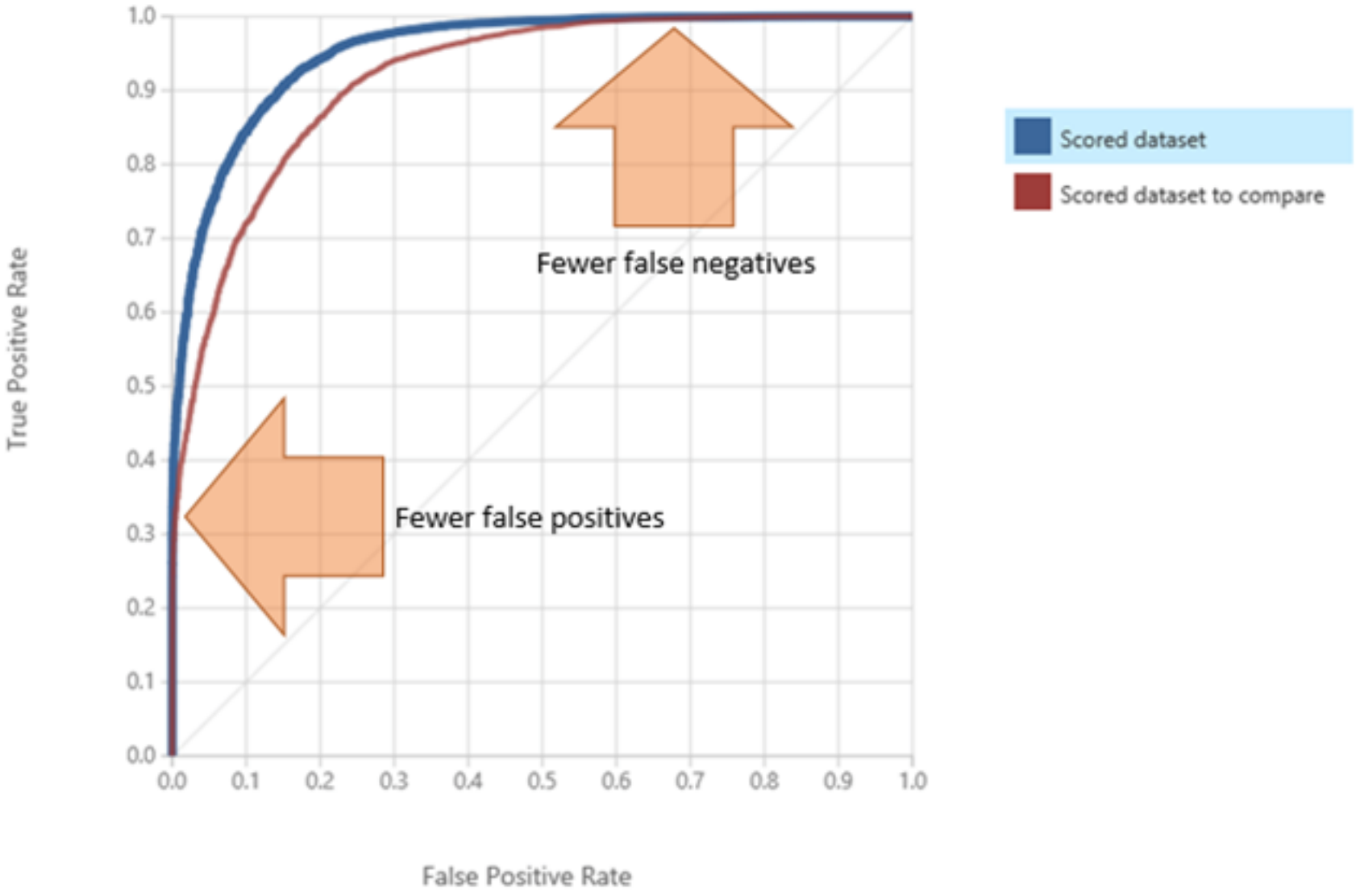
Problem Type	**Supported Methods**
Binary Classification	linear SVMs, logistic regression, decision trees, random forests, gradient-boosted trees, naive Bayes
Multiclass Classification	logistic regression, decision trees, random forests, naive Bayes
Regression	linear least squares, Lasso, ridge regression, decision trees, random forests, gradient-boosted trees, isotonic regression

MLLib Algorithms

- Algorithms – Collaborative Filtering – ALS

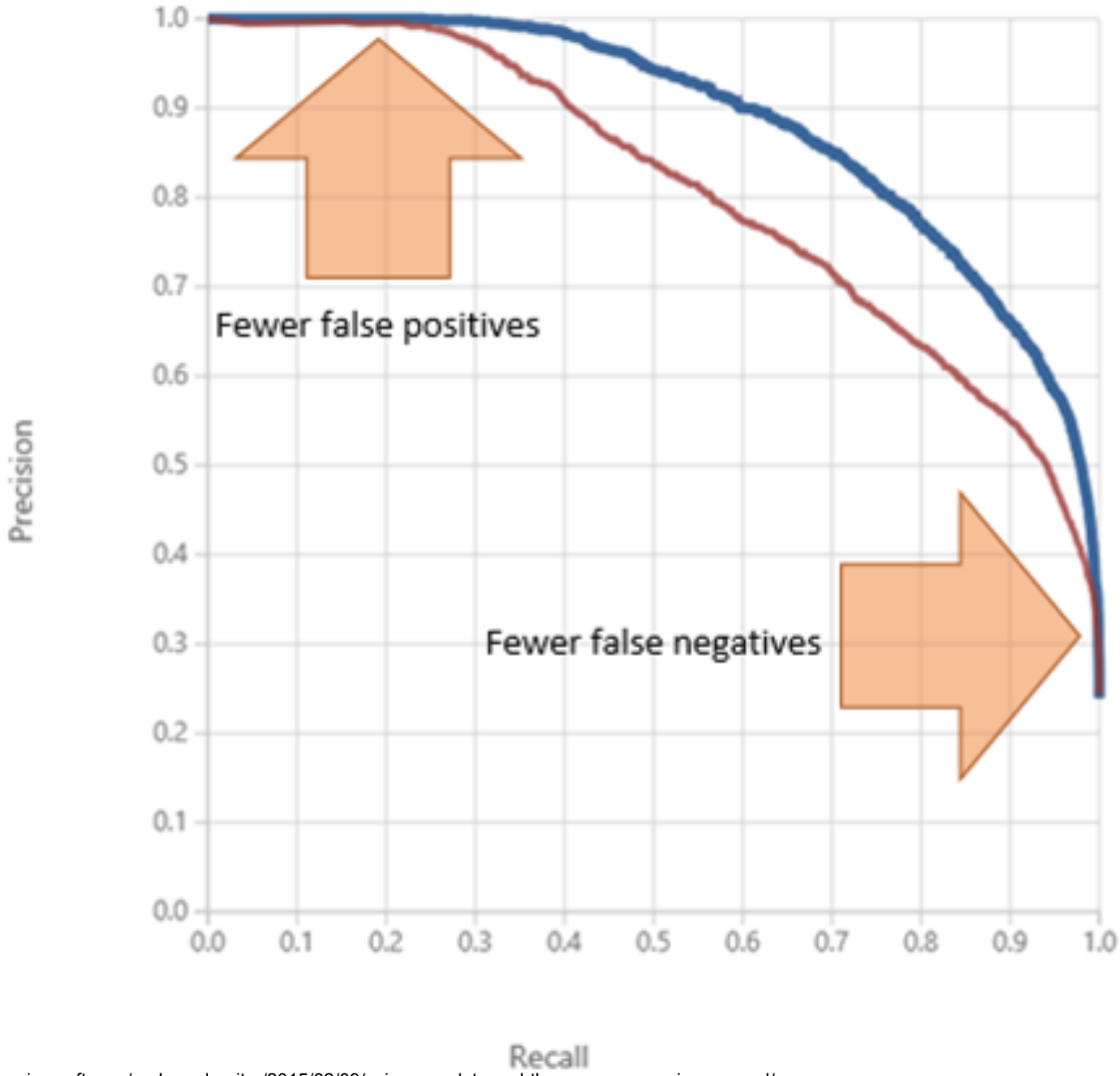
- Algorithms – Clustering
  - K-means
  - Gaussian mixture
  - Power iteration clustering (PIC)
  - Latent Dirichlet allocation (LDA)
  - Bisecting k-means
  - Streaming k-means

ROC



https://blogs.msdn.microsoft.com/andreasderuiter/2015/02/09/using-roc-plots-and-the-auc-measure-in-azure-ml/

Precision and Recall



https://blogs.msdn.microsoft.com/andreasderuiter/2015/02/09/using-roc-plots-and-the-auc-measure-in-azure-ml/

Lets Write a Classifier

SU – Kaggle Competition

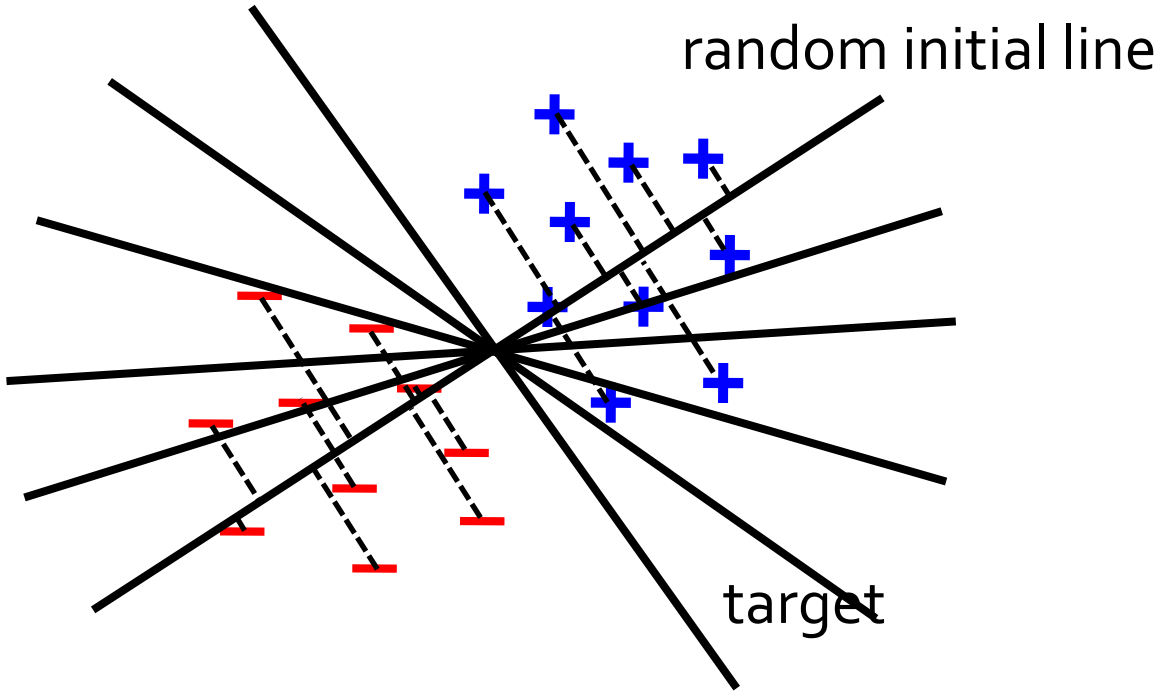
We will use the data from a Kaggle competition which classifies web pages as Ephemeral Vs Evergreen.

StumbleUpon is a user-curated web content discovery engine that recommends relevant, high quality pages and media to its users, based on their interests. While some pages we recommend, such as news articles or seasonal recipes, are only relevant for a short period of time, others maintain a timeless quality and can be recommended to users long after they are discovered.
In other words, pages can either be classified as "ephemeral" or "evergreen".

We have been given the Training and Test data along with the layout of the data , our Job is to write a Model that can do predictions going forward.

https://www.kaggle.com/c/stumbleupon

- **Goal: Write a model that will classify pages as Evergreen Vs Ephemeral OR find best line separating two sets of points**
- **Evergreen + Vs Ephemeral −**



Data

FieldName	Type	Description
url	string	Url of the webpage to be classified
urlid	integer	StumbleUpon's unique identifier for each url
boilerplate	json	Boilerplate text
alchemy_category	string	Alchemy category (per the publicly available Alchemy API found at www.alchemyapi.com)
alchemy_category_score	double	Alchemy category score (per the publicly available Alchemy API found at www.alchemyapi.com)
avglinksize	double	Average number of words in each link
commonLinkRatio_1	double	# of links sharing at least 1 word with 1 other links / # of links
commonLinkRatio_2	double	# of links sharing at least 1 word with 2 other links / # of links
commonLinkRatio_3	double	# of links sharing at least 1 word with 3 other links / # of links
commonLinkRatio_4	double	# of links sharing at least 1 word with 4 other links / # of links
compression_ratio	double	Compression achieved on this page via gzip (measure of redundancy)
embed_ratio	double	Count of number of <embed> usage
frameBased	integer (0 or 1)	A page is frame-based (1) if it has no body markup but have a frameset markup
frameTagRatio	double	Ratio of iframe markups over total number of markups
hasDomainLink	integer (0 or 1)	True (1) if it contains an <a> with an url with domain
html_ratio	double	Ratio of tags vs text in the page
image_ratio	double	Ratio of <img> tags vs text in the page
is_news	integer (0 or 1)	True (1) if StumbleUpon's news classifier determines that this webpage is news
lengthyLinkDomain	integer (0 or 1)	True (1) if at least 3 <a> 's text contains more than 30 alphanumeric characters
linkwordscore	double	Percentage of words on the page that are in hyperlink's text
news_front_page	integer (0 or 1)	True (1) if StumbleUpon's news classifier determines that this webpage is front-page news
non_markup_alphanum_characters	integer	Page's text's number of alphanumeric characters
numberOfLinks	integer	Number of <a> markups
numwords_in_url	double	Number of words in url
parametrizedLinkRatio	double	A link is parametrized if it's url contains parameters or has an attached onClick event
spelling_errors_ratio	double	Ratio of words not found in wiki (considered to be a spelling mistake)
label	integer (0 or 1)	User-determined label. Either evergreen (1) or non-evergreen (0); available for train.tsv only

Writing a ML Model

First ML Model – Refer Object InitialMLModels.scala

Approach –

1. Set spark context
2. Read in the training data , notice it's a tsv and not a csv , next Clean the data and create a RDD , remove headers, split on tabs , clean up quotes , convert negatives to "0"
3. Next we define the label which is col 26 , and we also start looking at features , for now we just consider numeric features col 5-25 .
4. We set various ML params like Iterations, thresholds etc etc
5. We create 5 models – **LogisticRegressionWithSGD, SVMWithSGD , NaiveBayes, DecisionTree , LogisticRegressionWithLBFGS**
6. Next we make a prediction on all our model with the same train dataset and manually calculate the accuracy for each model
7. Manual calculation is okay for first program but we can do better , use the metrics that come with the **BinaryClassificationMetrics class . Refer MLModelMetrics.scala**

Writing a ML Model

Improved version of ML Model – Refer Object ImprovedLR.scala

Approach –

1. 50% accuracy for LR is unacceptable . What can we do now ?
2. Let us get some metrics from out of our features to better understand the data.

```
val vectors = data.map(lp => lp.features)
val matrix = new RowMatrix(vectors)
val matrixSummary = matrix.computeColumnSummaryStatistics()
```

3. We noticed the variations so clearly the data is not scaled so let us scale the data with Mllib's standard scaler

```
val scaler = new StandardScaler(withMean = true, withStd = true).fit(vectors)
```

4. Also we should not be calculating the accuracy manually lets use PR and ROC

Writing a ML Model

All ML Models with metrics – Refer Object BetterLR.scala

Approach –

1. We were able to improve our model performance with scaled data ,
2. If we look at categories , we find we have 14 different categories and we could create a categorical vector and retrain.
3. So we create a categorical vector and add it to our feature vector to improve model performance .

Thank You