Spark Programming Basics (1)

Lecture 3
October 25th, 2017

Jae W. Lee (jaewlee@snu.ac.kr)
Computer Science and Engineering
Seoul National University

***Slide credits**: Pat McDonough (Databricks), Prof. Anthony Joseph (BerkeleyX CS100.1x, CS105x), Holden Karau et al. (Learning Spark)*

Outline

- **Overview**
- Resilient Distributed Datasets (RDDs)
- Spark Transformations and Actions
- Spark Programming Model
- Spark Key-Value RDDs
- Shuffles
- Job Execution

What is Spark?

- **Fast and expressive cluster computing system compatible with Apache Hadoop**
- **Efficient**
  - General execution graphs
  - In-memory storage
- **Usable**
  - Rich APIs in Java, Scala, Python
  - Interactive shell

Key Concepts

- **Write programs in terms of transformations on distributed datasets**
- **Resilient Distributed Datasets (RDDs)**
  - Collections of objects spread across a cluster, stored in RAM or on Disk
  - Built through parallel transformations
  - Automatically rebuilt on failure
- **Operations**
  - Transformations (e.g. map, filter, groupBy)
  - Actions (e.g. count, collect, save)

Language Support

Python

```
lines = sc.textFile(...)
lines.filter(lambda s: “ERROR” in s).count()
```

Scala

```
val lines = sc.textFile(...)
lines.filter(x => x.contains(“ERROR”)).count()
```

Java

```
JavaRDD<String> lines = sc.textFile(...);
lines.filter(new Function<String, Boolean>() {
  Boolean call(String s) {
    return s.contains(“error”);
  }
}).count();
```

Standalone Programs

- Python, Scala, &Java

Interactive Shells

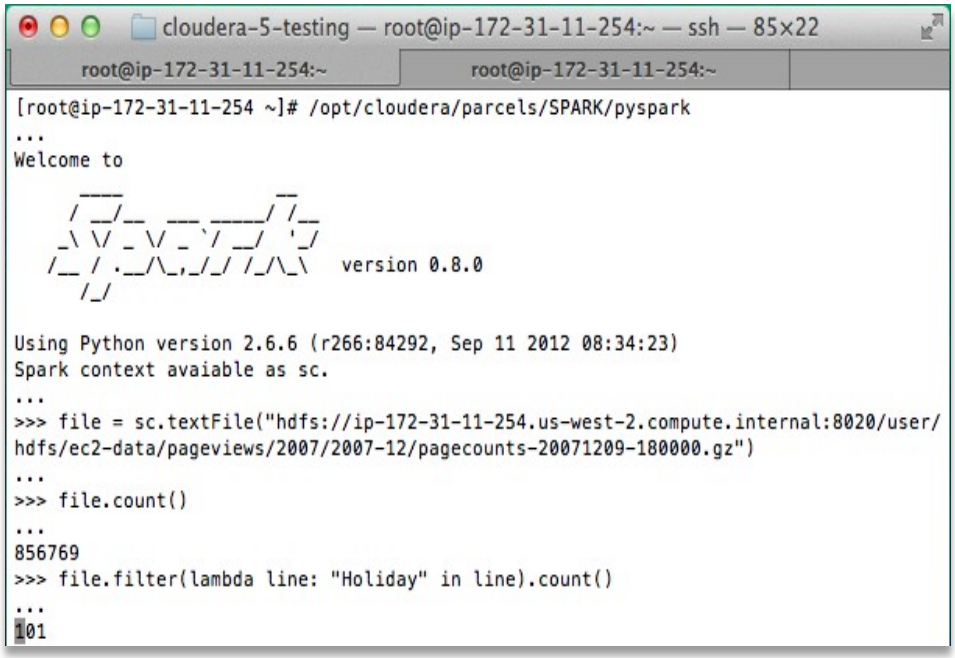
- Python & Scala

Performance

- Java & Scala are faster due to static typing
- …but Python is often fine

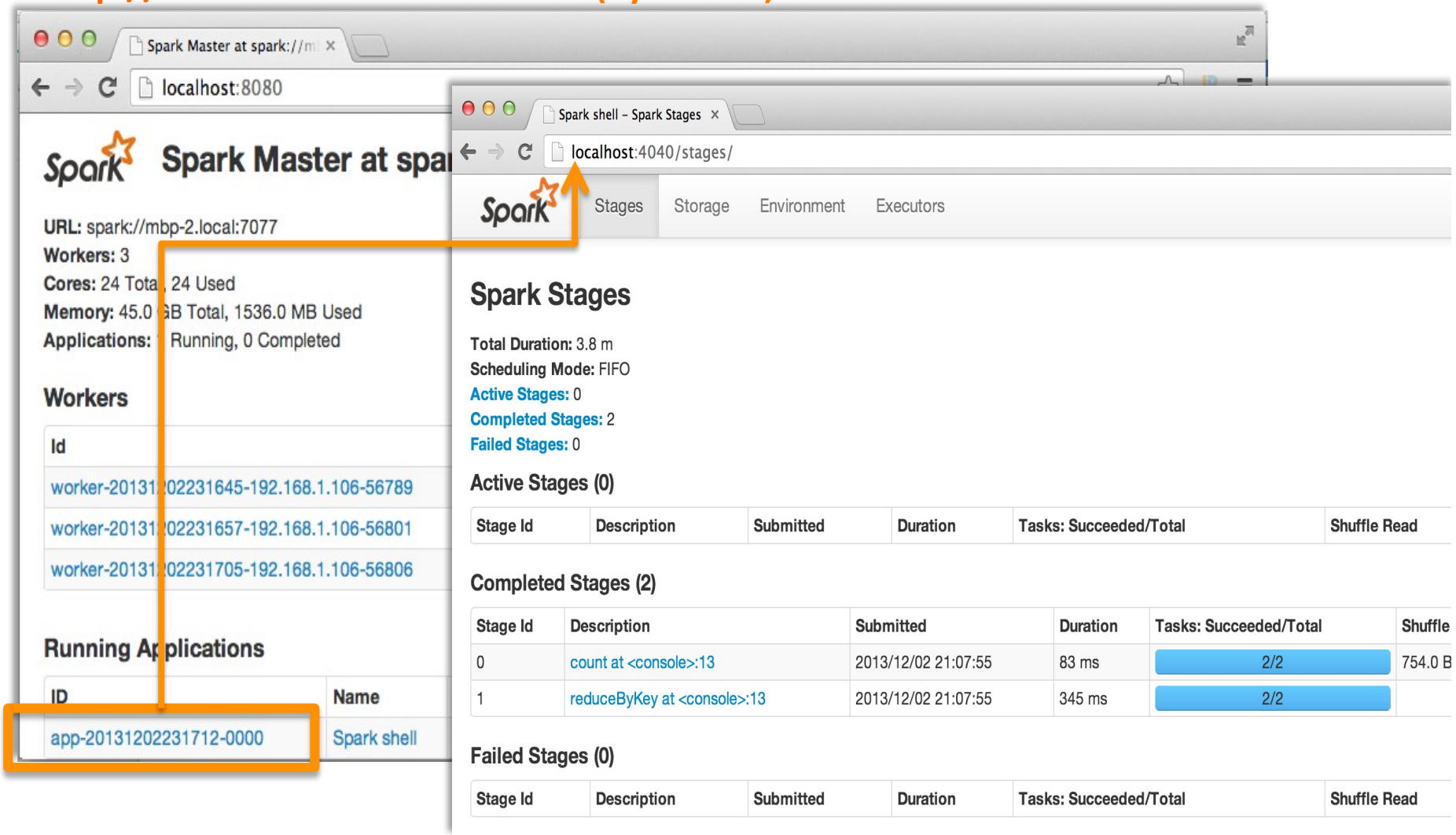
Interactive Shell

- The fastest way to learn Spark
- Available in Python and Scala
- Runs as an application on an existing Spark cluster…
- OR can run locally



Administrative GUIs

http://<Standalone Master>:8080 (by default)



Python Spark (pySpark)

- **We are using the Python programming interface to Spark (pySpark)**
- **pySpark provides an easy-to-use programming abstraction and parallel runtime:**
  - "Here's an operation, run it on all of the data"
- **RDDs are the key concept**

Using the shell

- **Launching: At $SPARK_HOME directory**

  `./bin/pyspark`

```
$ ./bin/pyspark --master local[4]
Python 2.7.6 (default, Oct 26 2016, 20:30:19)
[GCC 4.8.4] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
17/10/23 13:29:40 WARN NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.1.0
      /_/

Using Python version 2.7.6 (default, Oct 26 2016 20:30:19)
SparkSession available as 'spark'.
>>>
```

- **Modes**
  - `--master <master URL>` specifies which master SparkContext connects to
  - Examples

```
./bin/pyspark --master local     # local, 1 worker thread
./bin/pyspark --master local[2]  # local, 2 worker threads
./bin/pyspark --master spark://host:port # Spark standalone
cluster (port = 7077 by default)
```

Using the shell

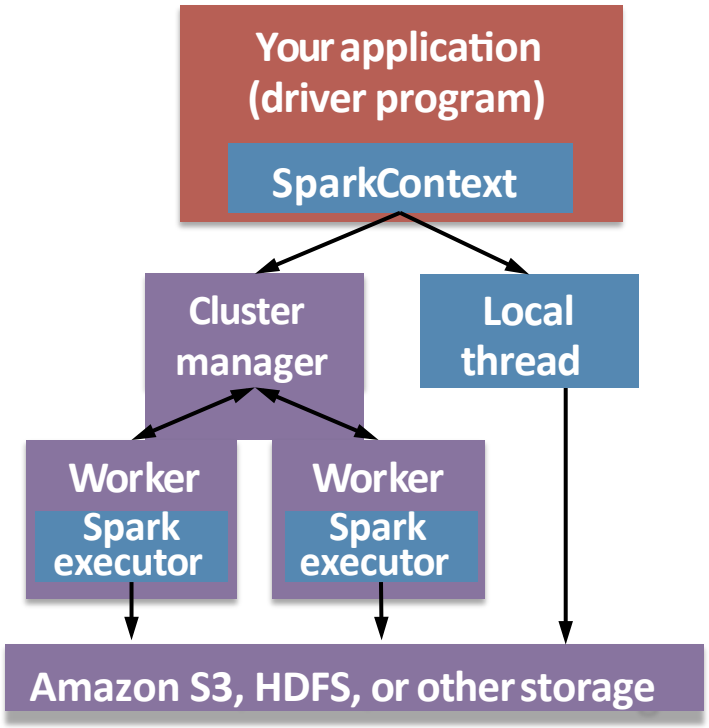
- **Launch the PySpark shell in IPython**
  - Enhanced Python interpreter

```
$ PYSPARK_DRIVER_PYTHON=ipython ./bin/pyspark
```

- **Use Jupyter notebook**
  - Previously known as IPython notebook

```
$ PYSPARK_DRIVER_PYTHON=jupyter \
  PYSPARK_DRIVER_PYTHON_OPTS=notebook ./bin/pyspark
```

Spark Driver and Workers



- **A Spark program is two programs:**
  - A driver program and a workers program
- **Worker programs run on cluster nodes or in local threads**
- **RDDs are distributed across workers**

Spark Context

- **A Spark program first creates a `SparkContext` object**
  - Main entry point to Spark functionality
  - Tells Spark how and where to access a cluster
  - pySpark shell automatically creates the `sc` variable
  - iPython and programs must use a constructor to create a new SparkContext
- **Use `SparkContext` to create RDDs**

In the labs, we create the SparkContext for you

Spark Essentials: Master

- The `master` parameter for a `SparkContext` determines which type and size of cluster to use.

Master Parameter	Description
`local`	run Spark locally with one worker thread (no parallelism)
`local[K]`	run Spark locally with K worker threads (ideally set to number of cores)
`spark://HOST:PORT`	connect to a Spark standalone cluster; PORT depends on config (7077 by default)
`mesos://HOST:PORT`	connect to a Mesos cluster; PORT depends on config (5050 by default)
`yarn`	connect to a YARN cluster; cluster location to be found based on HADOOP_CONF_DIR or YARN_CONF_DIR variable.

In the labs, we set the master parameter for you

Source: https://spark.apache.org/docs/latest/submitting-applications.html#master-urls

Outline

- Overview
- **Resilient Distributed Datasets (RDDs)**
- Spark Transformations and Actions
- Spark Programming Model
- Spark Key-Value RDDs
- Shuffles
- Job Execution

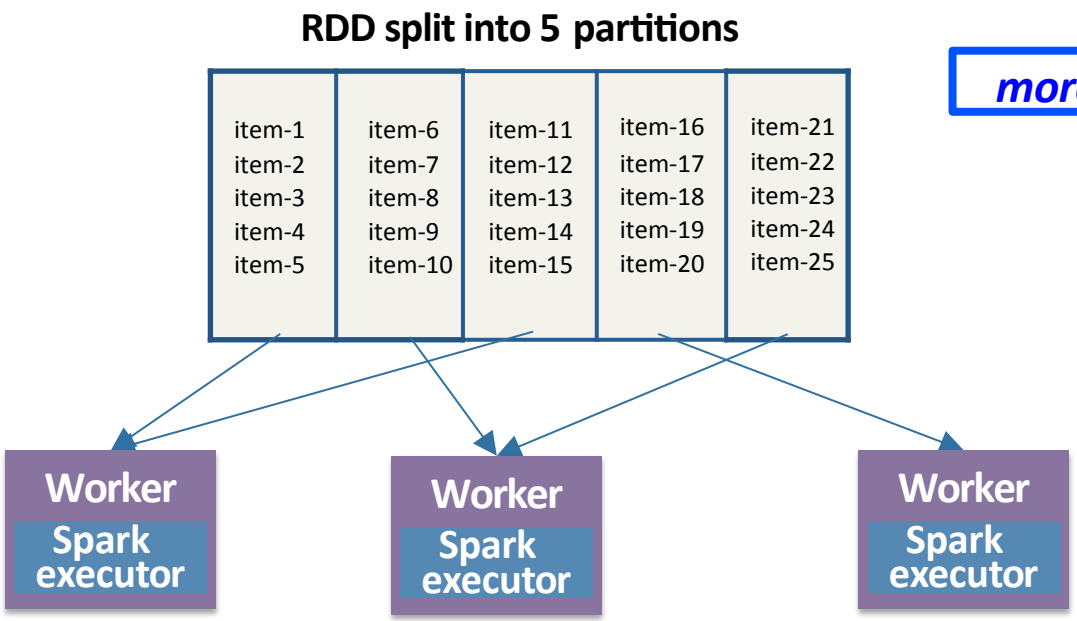
Resilient Distributed Datasets (RDDs)

- **The primary abstraction in Spark**
  - Immutable once constructed
  - Track lineage information to efficiently recompute lost data
  - Enable operations on collection of elements in parallel
- **You construct RDDs**
  - by *parallelizing* existing Python collections (lists)
  - by *transforming* an existing RDDs
  - from *files* in HDFS or any other storage system

Resilient Distributed Datasets (RDDs)

- **Programmer specifies number of partitions for an RDD**

(Default value used if unspecified)



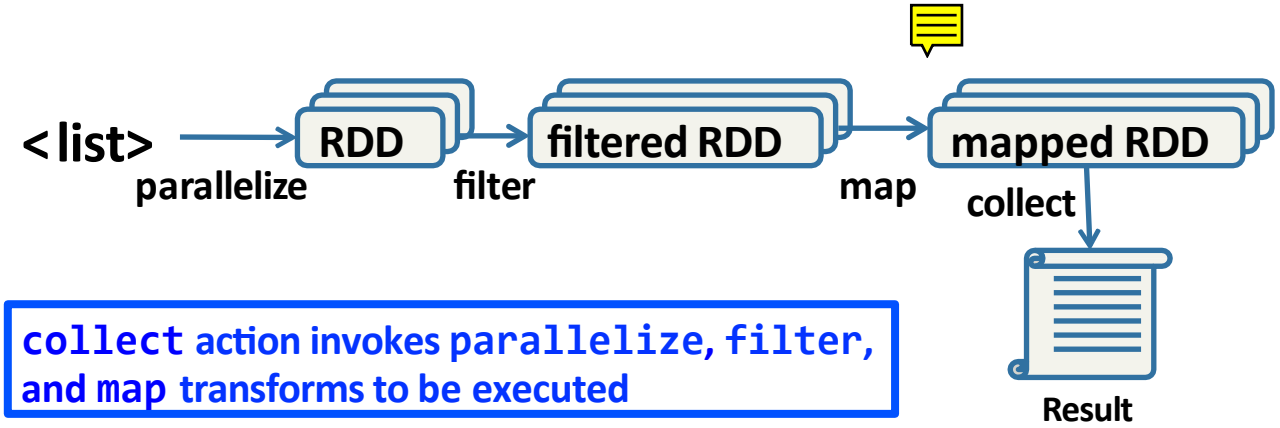
more partitions = more parallelism

Resilient Distributed Datasets (RDDs)

- Two types of operations: ***transformations*** and ***actions***
- Transformations are lazy (*not computed immediately*)
- Transformed RDD is computed when action runs on it
- Persist (cache) RDDs in memory or disk

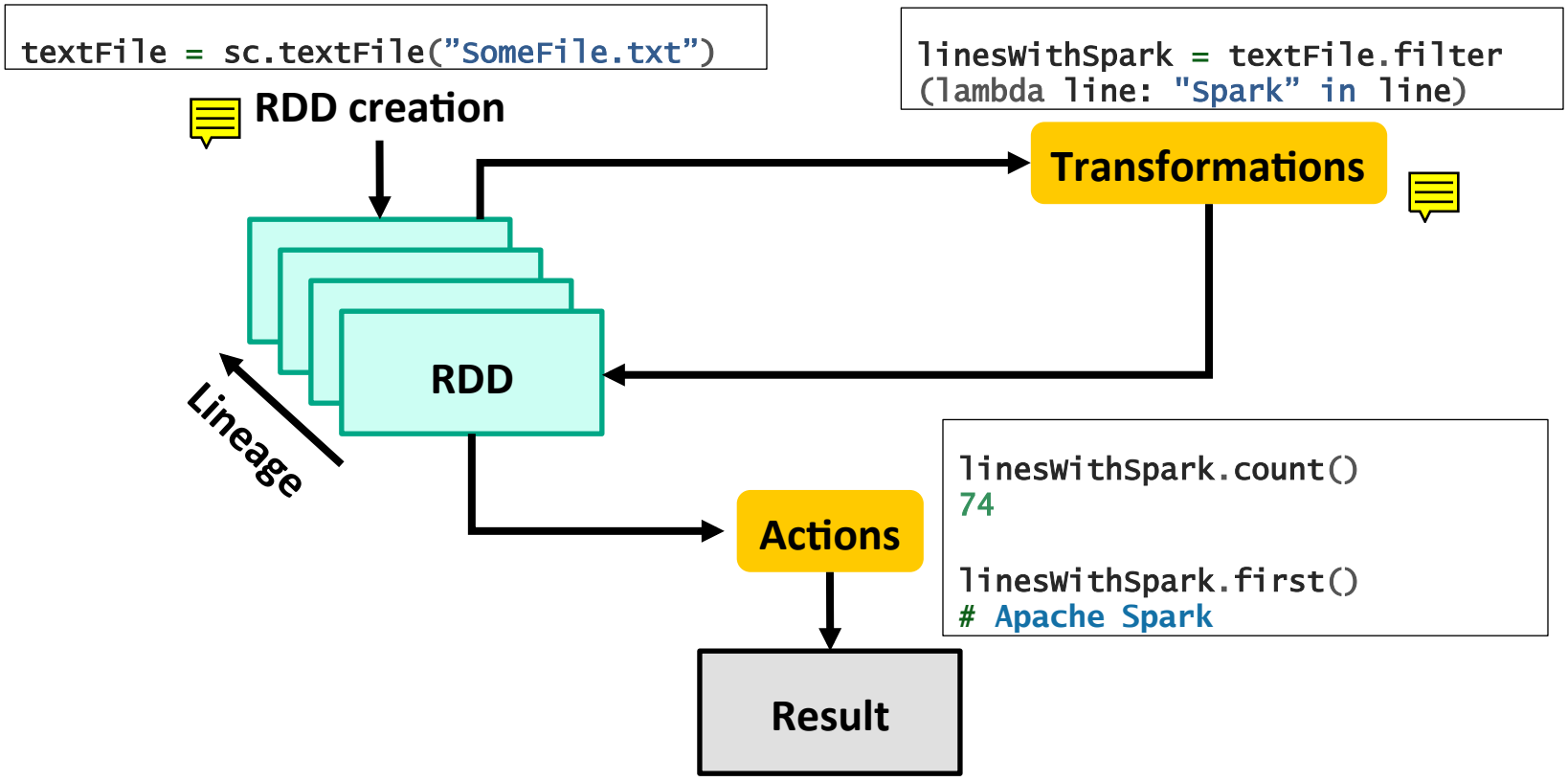
Working with RDDs

- Create an RDD from a data source:  <list>
- Apply transformations to an RDD: map, filter, ...
- Apply actions to an RDD: collect, count, ...



Working with RDDs

- Another view



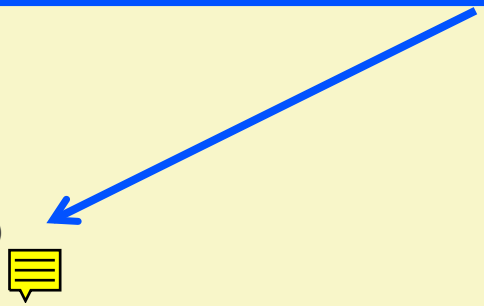
* image from http://data-flair.training/blogs/apache-spark-lazy-evaluation/

Creating an RDD

- Create RDDs from Python collections (lists)

```
>>> data = [1, 2, 3, 4, 5]
>>> data
[1, 2, 3, 4, 5]

>>> rDD = sc.parallelize(data, 4)
>>> rDD
ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:229
```

No computation occurs with `sc.parallelize()`
- Spark only records how to create the RDD with four partitions



Creating an RDD

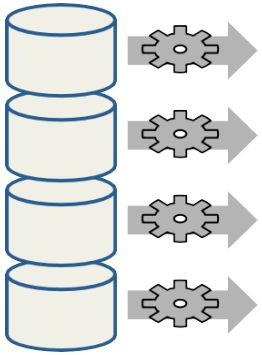
- **From external datasets**
  - Spark can create distributed datasets from any storage source supported by Hadoop, including your local file system, HDFS, Cassandra, HBase, Amazon S3, etc.
  - Spark supports text files, SequenceFiles, and any other Hadoop InputFormat.
  - Text file RDDs can be created using `SparkContext`'s `textFile` method (read as a collection of lines).

```
>>> distFile = sc.textFile("README.md", 4)  # 4 is minimum # of partitions

>>> distFile
README.md MapPartitionsRDD[1] at textFile at NativeMethodAccessorImpl.java: 0

>>> distfile.first()
u'# Apache Spark'
```

Creating an RDD

- Creating an RDD from a file

```
distFile = sc.textFile("...", 4)
```



- RDD distributed in 4 partitions
- Elements are lines of input
- *Lazy evaluation* means no execution happens now

Creating an RDD

- **More examples**

```
# Turn a Python collection into an RDD
> sc.parallelize([1, 2, 3])

# Load text file from local FS, HDFS, or S3
> sc.textFile("file.txt")
> sc.textFile("directory/*.txt")
> sc.textFile("hdfs://namenode:9000/path/file")

# Use existing Hadoop InputFormat (Java/Scala only)
> sc.hadoopFile(keyClass, valClass, inputFmt, conf)
```

Outline

- Overview
- Resilient Distributed Datasets (RDDs)
- **Spark Transformations and Actions**
- Spark Programming Model
- Spark Key-Value RDDs
- Shuffles
- Job Execution

Spark Transformations

- **Create new datasets from an existing one**
- **Use *lazy evaluation*: results not computed right away – instead Spark remembers set of transformations applied to base dataset**
  - Spark optimizes the required calculations
  - Spark recovers from failures and slow workers
- **Think of this as a recipe for creating result**

Spark Transformations

- **Some examples**

Transformation	Description
`map(`*`func`*`)`	return a new distributed dataset formed by passing each element of the source through a function *func*
`filter(`*`func`*`)`	return a new dataset formed by selecting those elements of the source on which *func* returns true
`distinct([`*`numTasks`*`]))`	return a new dataset that contains the distinct elements of the source dataset
`flatMap(`*`func`*`)`	similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item)

Spark Transformations

- **Review: Python `lambda` functions**
  - Small anonymous functions (not bound to a name)

```
lambda a, b: a + b
```

    - returns the sum of its two arguments

  - Can use lambda functions wherever function objects are required
  - Restricted to a single expression

Spark Transformations

- Using lambda functions

```
>>> rdd = sc.parallelize([1, 2, 3, 4])
>>> rdd.map(lambda x: x * 2)
RDD: [1, 2, 3, 4] → [2, 4, 6, 8]

>>> rdd.filter(lambda x: x % 2 == 0)
RDD: [1, 2, 3, 4] → [2, 4]

>>> rdd2 = sc.parallelize([1, 4, 2, 2, 3])
>>> rdd2.distinct()
RDD: [1, 4, 2, 2, 3] → [1, 4, 2, 3]
```

Function literals (green) are closures automatically passed to workers

Spark Transformations

- Using lambda functions

```
>>> rdd = sc.parallelize([1, 2, 3])
>>> rdd.map(lambda x: [x, x+5])
RDD: [1, 2, 3] → [[1, 6], [2, 7], [3, 8]]

>>> rdd.flatMap(lambda x: [x, x+5])
RDD: [1, 2, 3] → [1, 6, 2, 7, 3, 8]
```

Function literals (green) are closures automatically passed to workers

Spark Transformations

- More lambda example

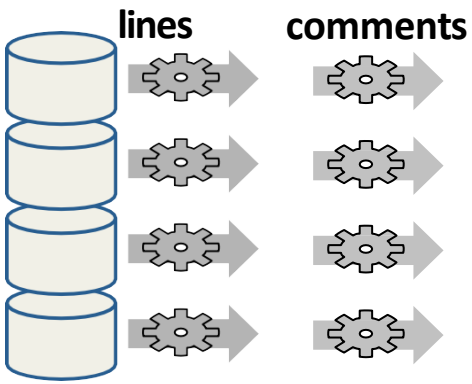
```
> nums = sc.parallelize([1, 2, 3])

# Pass each element through a function
> squares = nums.map(lambda x: x*x)   // {1, 4, 9}

# Keep elements passing a predicate
> even = squares.filter(lambda x: x%2 == 0) // {4}

# Map each element to zero or more others
>  nums.flatMap(lambda x: => range(x)) # => {0, 0, 1, 0, 1, 2}
```

Range object (sequence of numbers 0, 1, …, x-1)

Spark Transformations

- Using a top-level function

```
lines = sc.textFile("...", 4)

comments = lines.filter(isComment)
```

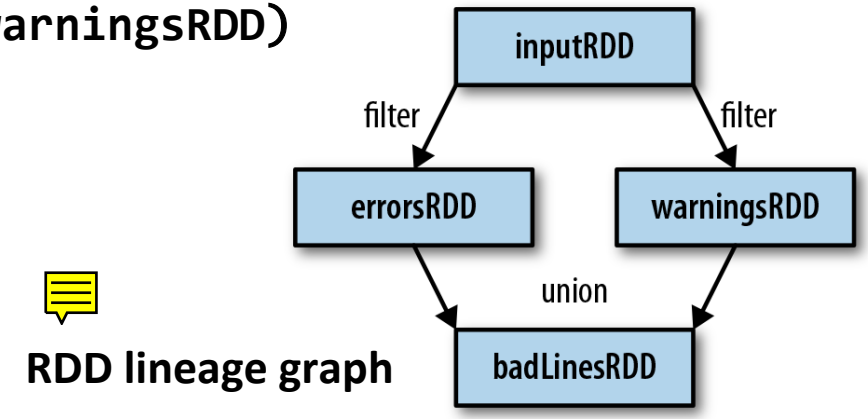


Lazy evaluation means nothing executes – Spark saves recipe for transforming source

Spark Transformations

- Processing a log file

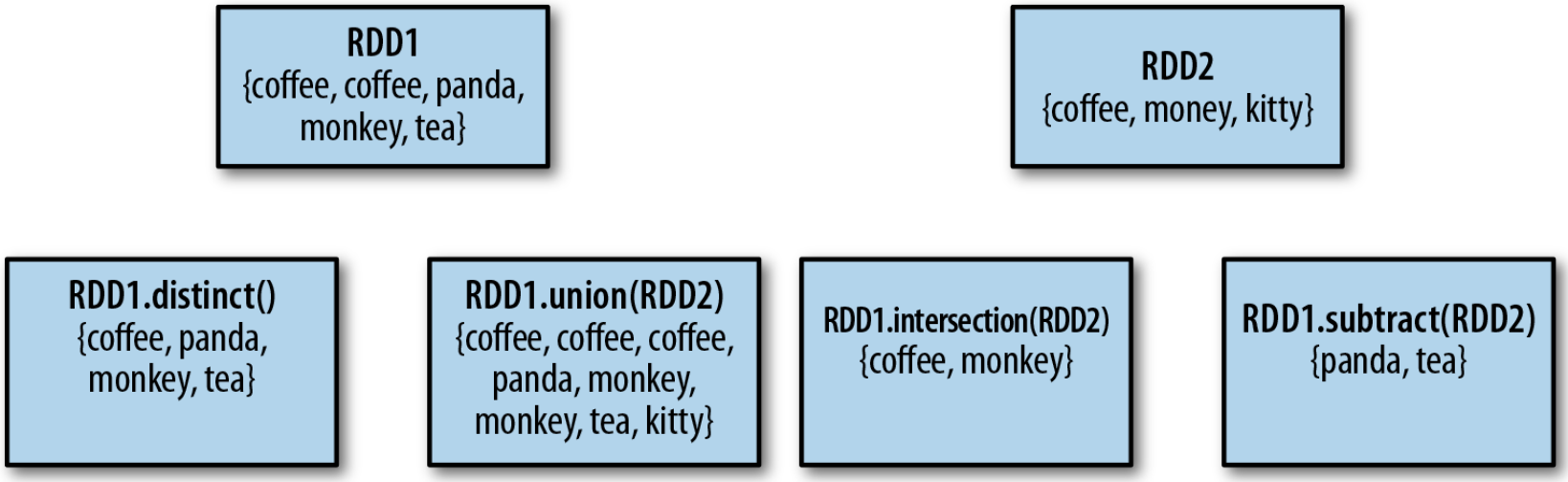
```
> inputRDD = sc.textFile("log.txt")

# RDD for "error" lines
> errorsRDD = inputRDD.filter(lambda x: "error" in x)

# RDD for "warning" lines
> warningsRDD = inputRDD.filter(lambda x: "warning" in x)

# Union them into badlinesRDD
> badlinesRDD = errorsRDD.union(warningsRDD)
```



RDD lineage graph

Spark Transformations

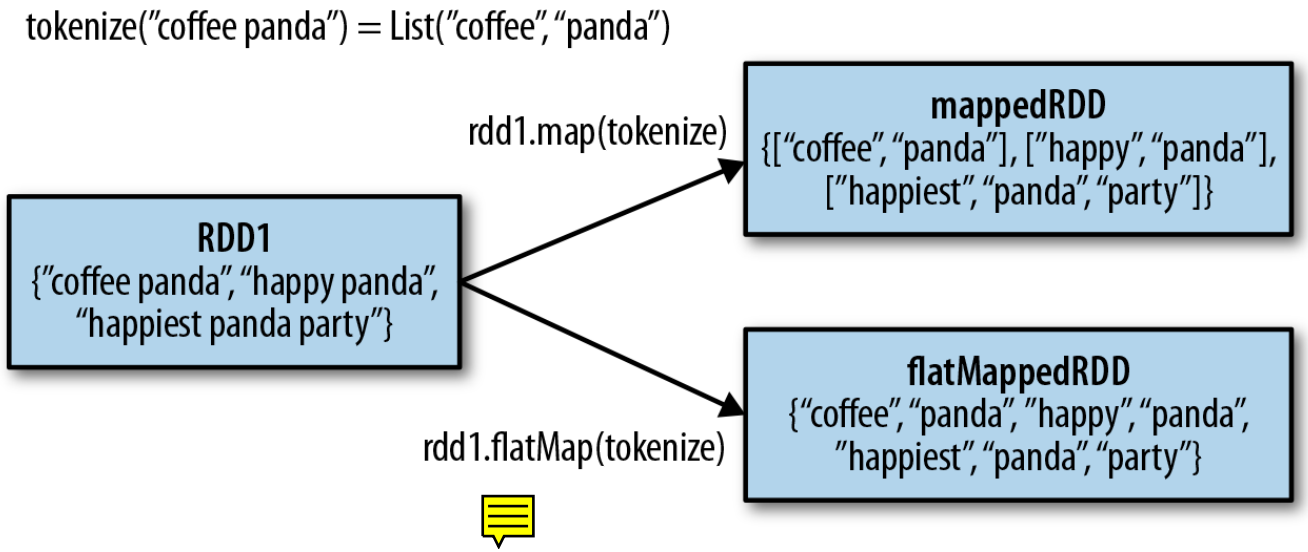
- **Four set operations on RDD1 and RDD2**
  - `distinct`, `union`, `intersection`, `subtract`



Spark Transformations

- Splitting lines into words by using `flatMap`

```
> lines = sc.parallelize(["hello world", "hi"])

# Split!
> words = lines.flatMap(lambda line: line.split(" "))

# returns "hello"
> words.first()
```



Passing Functions to Spark

- **Spark's API relies heavily on passing functions in the driver program to run on the cluster.**
- **There are three recommended ways to do this.**
  - Lambda expression
    - For simple functions that can be written as an expression
    - Lambdas do not support multi-statement functions or statements that do not return a value.
  - Top-level functions in a module
  - Local def's inside the function calling into Spark

```
word = rdd.filter(lambda s: "error" in s)

def containsError(s):
    return "error" in s
word = rdd.filter(containsError)
```


Source: http://spark.apache.org/docs/latest/rdd-programming-guide.html#passing-functions-to-spark

Passing Functions to Spark

- **Caveat!**
  - When you pass a function that is the member of an object, or contains references to fields in an object, Spark sends the *entire object* to worker nodes and causes a lot of waste!
    (instead of the small bit of information you need...)

```
# Passing a function with field references
# (NOT OK)
class SearchFunctions(object):
  def __init__(self, query):
      self.query = query
  def isMatch(self, s):
      return self.query in s
  def getMatchesFunctionReference(self, rdd):
      # Problem: references all of "self" in "self.isMatch"
      return rdd.filter(self.isMatch)
  def getMatchesMemberReference(self, rdd):
      # Problem: references all of "self" in "self.query"
      return rdd.filter(lambda x: self.query in x)
```

```
# Function passing without field references
# (OK)
class WordFunctions(object):
  ...
  def getMatchesNoReference(self, rdd):
      # Safe: extract only the field we need into a local variable
      query = self.query
      return rdd.filter(lambda x: query in x)
```

Passing Functions to Spark

- **Closure**
  - To execute jobs, Spark breaks up the processing of RDD operations into **tasks**, each of which is executed by an **executor**.
  - Prior to execution, Spark computes the task's **closure**.
    - Variables and methods which must be visible for the executor to perform its computations on the RDD
  - This closure is serialized and sent to each executor.
    - The variables within the closure (e.g., `increment_counter`) sent to each executor are now copies and thus, when `counter` is referenced within the `foreach` function, it's no longer the counter on the driver node.

```
counter = 0
rdd = sc.parallelize(data)

# Wrong: Don't do this!!
def increment_counter(x):
    global counter
    counter += x
rdd.foreach(increment_counter)

print("Counter value: ", counter)
```

Passing Functions to Spark

- **Using local variables**
  - Any external variables you use in a closure will automatically be shipped to the cluster:

```
> query = sys.stdin.readline()
> pages.filter(lambda x: query in x).count()
```

  - Some caveats:
    - Each task gets a new copy (updates aren't sent back)
    - Variable must be Serializable / Pickle-able
    - Don't use fields of an outer object (ships all of it!)

Spark Actions

- Cause Spark to execute recipe to transform source
- Mechanism for getting results out of Spark

Spark Actions

- **Some example actions**

Action	Description
`reduce(`*`func`*`)`	aggregate dataset's elements using function *func*. *func* takes two arguments and returns one, and is commutative and associative so that it can be computed correctly in parallel
`take(`*`n`*`)`	return an array with the first *n* elements
`collect()`	return all the elements as an array WARNING: make sure will fit in driver program
`takeOrdered(`*`n, key=func`*`)`	return *n* elements ordered in ascending order or as specified by the optional key function
`saveastextfile(`*`path`*`)`	save this RDD as a text file to *path*, using string representations of elements

Spark Actions

- Getting data out of RDDs

```
>>> rdd = sc.parallelize([1, 2, 3])
>>> rdd.reduce(lambda a, b: a * b)
Value: 6

>>> rdd.take(2)
Value: [1,2] # as list

>>> rdd.collect()
Value: [1,2,3] # as list
```

Spark Actions

- Getting data out of RDDs

```
>>> rdd = sc.parallelize([5,3,1,2])
>>> rdd.takeOrdered(3, lambda s: -1 * s)
Value: [5,3,2] # as list
```

Spark Actions

- More examples

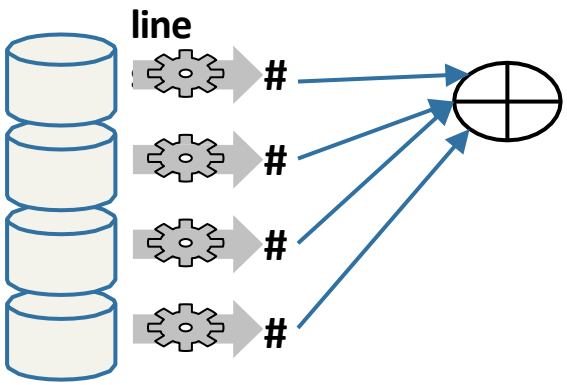
```
> nums = sc.parallelize([1, 2, 3])

# Retrieve RDD contents as a local collection
> nums.collect() # => [1, 2, 3]

# Return first k elements
> nums.take(2)   # => [1, 2]

# Count number of elements
> nums.count()   # => 3

# Merge elements with an associative function
> nums.reduce(lambda x, y: x + y)   # => 6

# Write elements to a text file
> nums.saveAsTextFile("hdfs://file.txt")   # => 6
```

Outline

- Overview
- Resilient Distributed Datasets (RDDs)
- Spark Transformations and Actions
- **Spark Programming Model**
- Spark Key-Value RDDs
- Shuffles
- Job Execution

Spark Programming Model

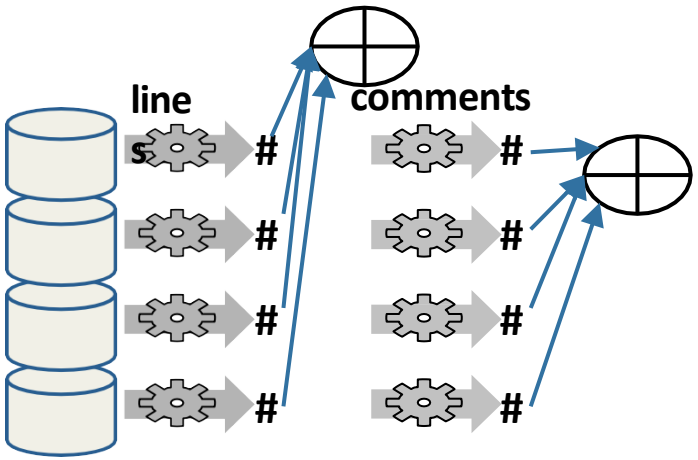
```
lines = sc.textFile("...", 4)

print lines.count()
```



`count()` causes Spark to:
- read data
- sum within partitions
- combine sums in driver

Spark Programming Model

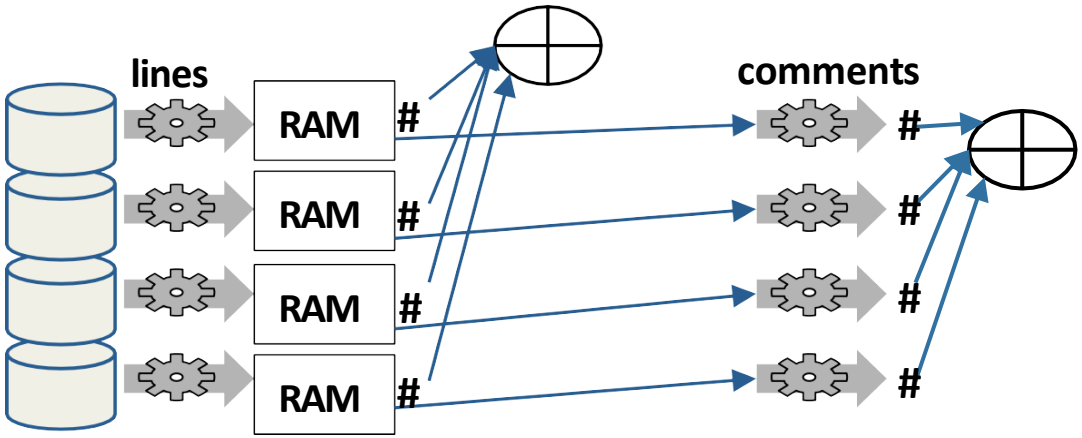
```
lines = sc.textFile("...", 4)
comments = lines.filter(isComment)
print lines.count(), comments.count()
```



Spark recomputes lines:
- read data (again)
- sum within partitions
- combine sums in driver

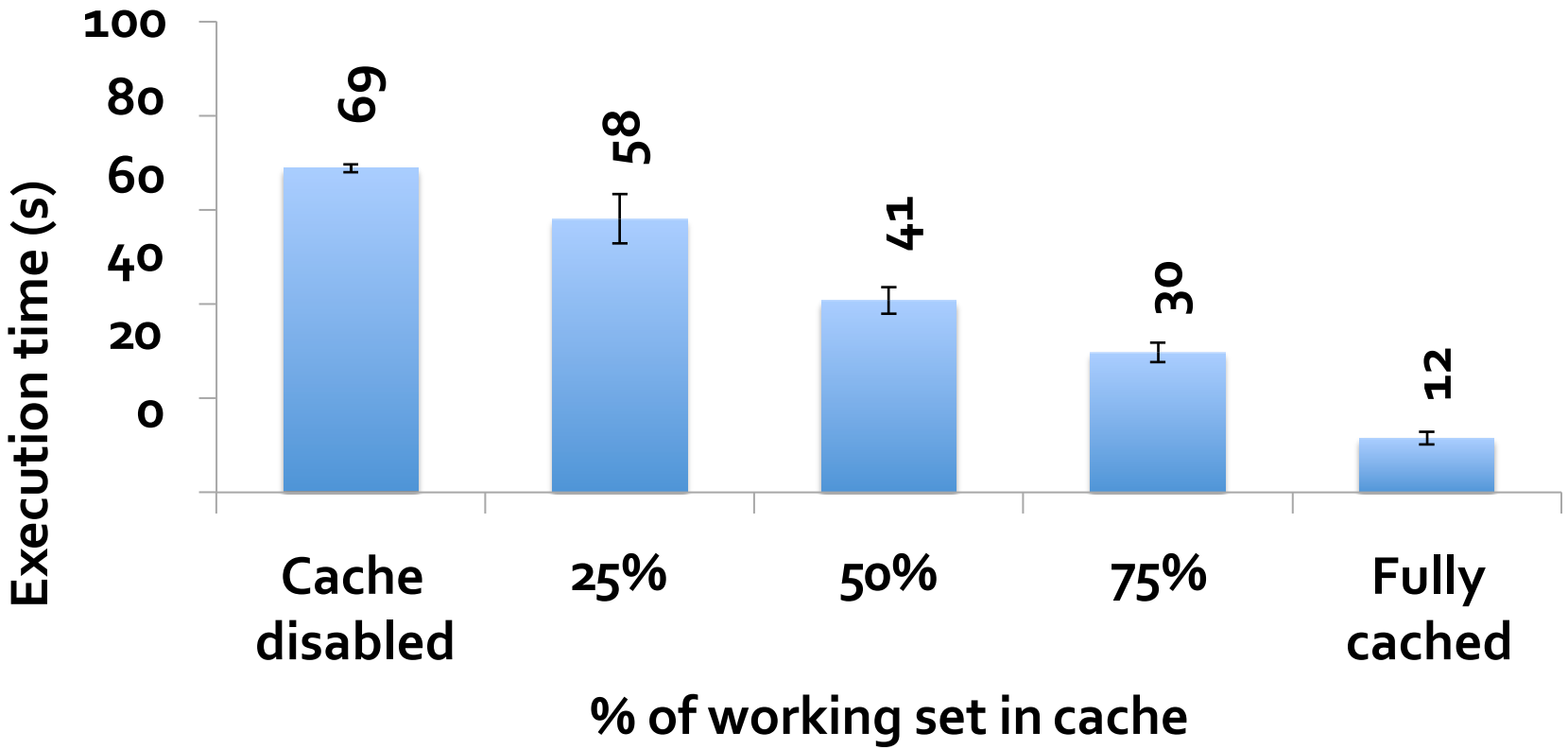
Caching RDDs

```
lines = sc.textFile("...", 4)
lines.cache() # save, don't recompute!
comments = lines.filter(isComment)
print lines.count(),comments.count()
```



Caching RDDs

- Scaling down execution time



Spark Program Lifecycle

1. Create RDDs from external data or parallelize a collection in your driver program
2. Lazily transform them into new RDDs
3. `cache()` some RDDs for reuse
4. Perform actions to execute parallel computation and produce results

Example: Log Mining

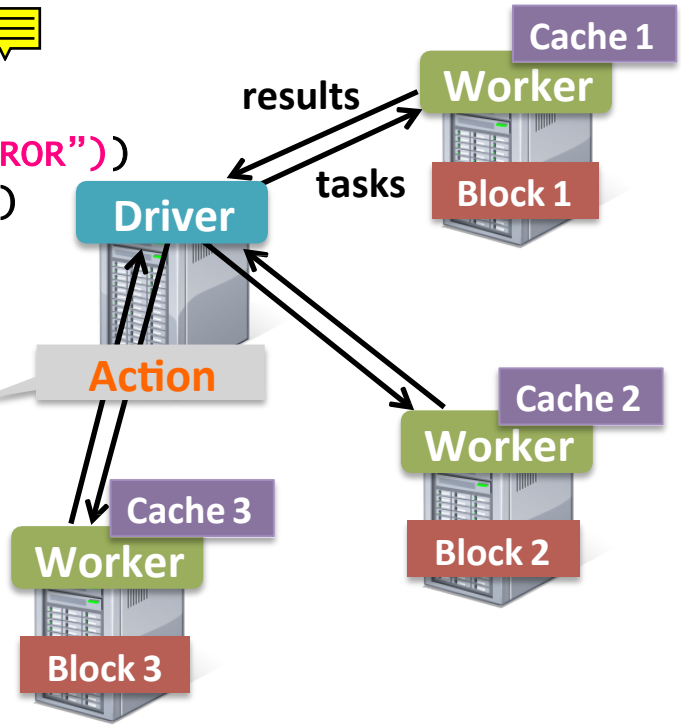
- Load error messages from a log into memory, then interactively search for various patterns

BaseRDD

TransformedRDD

```
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()
. . .
```

Action



Full-–text search of Wikipedia
- 60GB on 20 EC2 machine
- 0.5 sec vs. 20s for on-disk

Outline

- Overview
- Resilient Distributed Datasets (RDDs)
- Spark Transformations and Actions
- Spark Programming Model
- **Spark Key-Value RDDs (Pair RDDs)**
- Shuffles
- Job Execution

Spark Key-Value RDDs

- **Similar to MapReduce, Spark supports Key-Value pairs**
- **Spark provides special operations on RDDs containing key/value pairs.**
  - These RDDs are called pair RDDs
- **Each element of a pair RDD is a pair tuple.**

```
>>> rdd = sc.parallelize([(1, 2), (3, 4)])
RDD: [(1, 2), (3, 4)]
```

Working with Key-Value Pairs

- Spark's "distributed reduce" transformations operate on RDDs of key-value pairs

Python:

```
pair = (a, b)
       pair[0] # => a
       pair[1] # => b
```

Scala:

```
val pair = (a, b)
       pair._1 // => a
       pair._2 // => b
```

Java:

```
Tuple2 pair = new Tuple2(a, b);
       pair._1 // => a
       pair._2 // => b
```

Key-Value Transformations

- **Some Key-Value transformations**

Key-Value Transformation	Description
`reduceByKey(`*`func`*`)`	return a new distributed dataset of (K,V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V,V) → V
`sortByKey()`	return a new dataset (K,V) pairs sorted by keys in ascending order
`groupByKey()`	return a new dataset of (K, Iterable<V>) pairs

Key-Value Transformations

- **Examples**

```
>>> rdd = sc.parallelize([(1,2), (3,4), (3,6)])
>>> rdd.reduceByKey(lambda a, b: a + b)
RDD: [(1,2), (3,4), (3,6)] → [(1,2), (3,10)]

>>> rdd2 = sc.parallelize([(1,'a'), (2,'c'), (1,'b')])
>>> rdd2.sortByKey()
RDD: [(1,'a'), (2,'c'), (1,'b')] →
                [(1,'a'), (1,'b'), (2,'c')]
```

Key-Value Transformations

- groupByKey example

```
>>> rdd2 = sc.parallelize([(1,'a'), (2,'c'), (1,'b')])
>>> rdd2.groupByKey()
RDD: [(1,'a'), (1,'b'), (2,'c')] →
                [(1,['a','b']), (2,['c'])]
```

Be careful using groupByKey() as it can cause a lot of data movement across the network and create large Iterables at workers

Key-Value Transformations

- **More key-value operations**

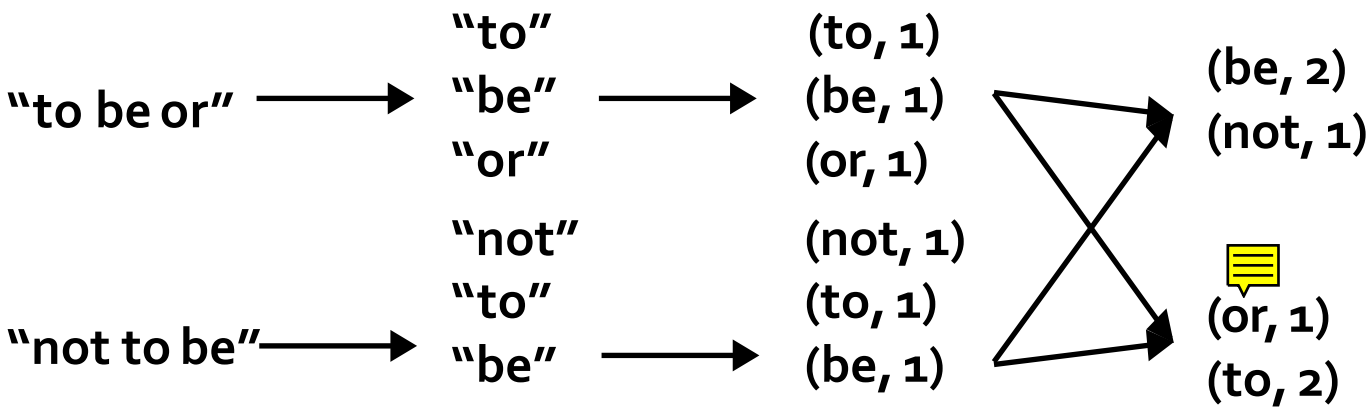
```
> pets = sc.parallelize([("cat", 1), ("dog", 2), ("cat", 3)])

# Reduction example
> pets.reduceByKey(lambda x, y: x + y) # => {(cat, 3), (dog, 1)}

# groupByKey
> pets.groupByKey()    # => {(cat, [1, 2]), (dog, [1])}

# Count number of elements
> pets.sortByKey()     # => {(cat, 1), (cat, 2), (dog, 1)}
```

  - reduceByKey also automatically implements combiners on the map side

Key-Value Transformations

- **Example: Word Count**

```
lines = sc.textFile("hamlet.txt")
counts = lines.flatMap(lambda line: line.split(" "))
               .map(lambda word: (word, 1))
               .reduceByKey(lambda x, y: x + y)
```



Key-Value Transformations

- Other key-value operations

```
visits = sc.parallelize([  (“index.html”, “1.2.3.4”),
                           (“about.html”, “3.4.5.6”),
                           (“index.html”, “1.3.3.1”)  ])

pageNames = sc.parallelize([ (“index.html”, “Home”),
                             (“about.html”, “About”) ])

visits.join(pageNames)
# ("index.html", ( "1.2.3.4", "Home" ))
# ("index.html", ( "1.3.3.1", "Home" ))
# ("about.html", ( "3.4.5.6", "About" ))

visits.cogroup(pageNames)
# ("index.html", ( ["1.2.3.4", "1.3.3.1"], ["Home"] ))
# ("about.html", ( ["3.4.5.6"], ["About"] ))
```

Key-Value Transformations

- **Setting the Level of Parallelism**
  - All the pair RDD operations take an optional second parameter for the number of tasks

```
> words.reduceByKey(lambda x, y: x + y, 5)
> words.groupByKey(5)
> visits.join(pageViews, 5)
```

Outline

- Overview
- Resilient Distributed Datasets (RDDs)
- Spark Transformations and Actions
- Spark Programming Model
- Spark Key-Value RDDs (Pair RDDs)
- **Shuffles**
- Job Execution

Shuffle Operations

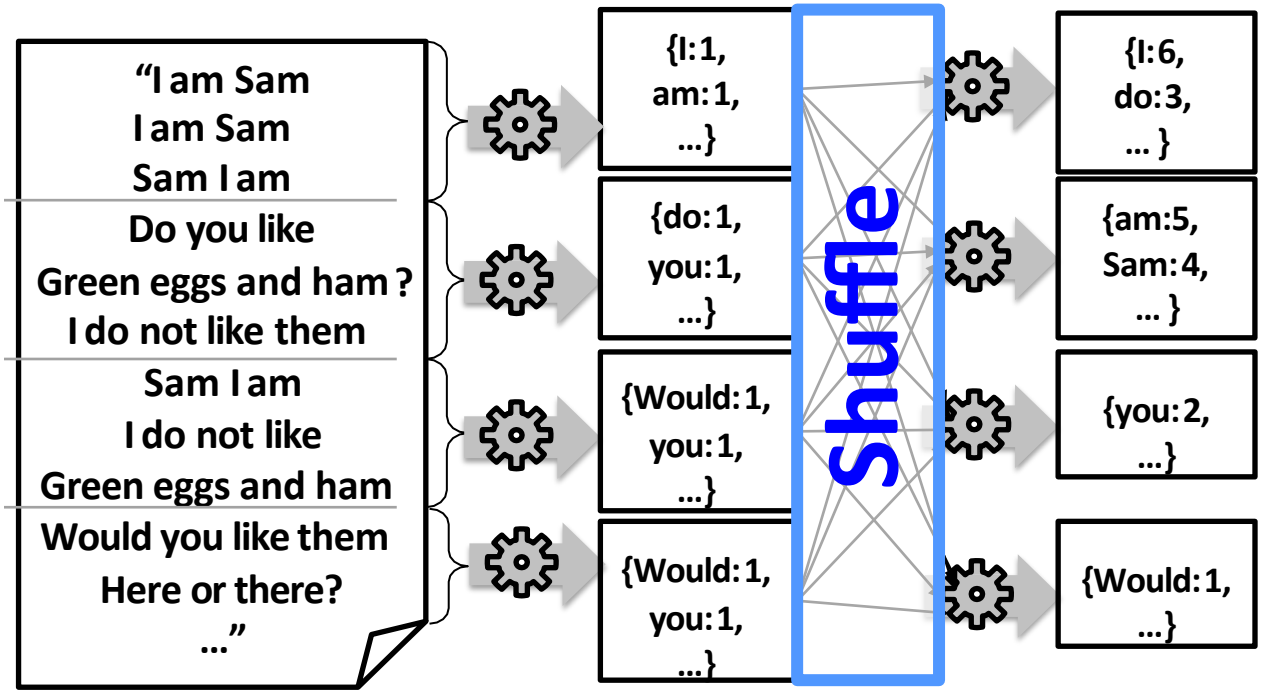
- **Certain operations within Spark trigger an event known as the shuffle.**
  - The shuffle is Spark's mechanism for re-distributing data so that it's grouped differently across partitions.

- **Operations that cause a shuffle include:**
  - Repartition operations: `repartition`, `coalesce`
  - *ByKey operations: `groupByKey`, `reduceByKey`
  - Join operations: `cogroup`, `join`

Shuffle Operations

- **Example: reduceByKey**

```
>>> rdd = sc.parallelize([("a", 1), ("b", 1), ("a", 1)])
>>> sorted(rdd.groupByKey().mapValues(len).collect())
[('a', 2), ('b', 1)]
>>> sorted(rdd.groupByKey().mapValues(list).collect())
[('a', [1, 1]), ('b', [1])]
```



Shuffle Operations

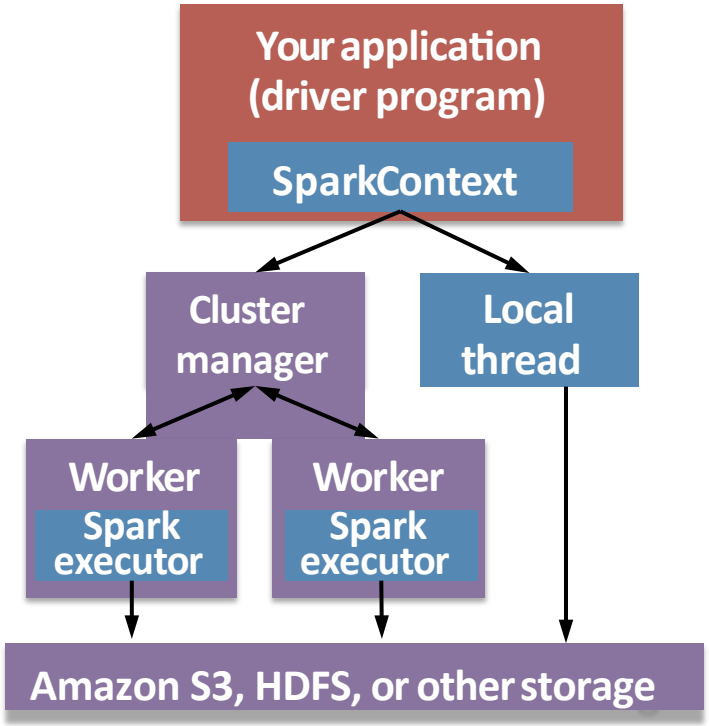
- **Performance impact**
  - Expensive operation
    - Involving disk I/O, data serialization, and network I/O
  - Can consume significant amounts of heap memory by employing in-memory data structures to organize records before or after transferring them
    - When data does not fit in memory Spark will spill these tables to disk, incurring the additional overhead of disk I/O and increased garbage collection.
  - Also generates a large number of intermediate files on disk
    - These files are preserved until the corresponding RDDs are no longer used and are garbage collected to avoid re-creation of shuffle files if the lineabe is re-computed.
  - Shuffle behavior can be tuned by adjusting a variety of config. parameters
    - See "Spark Configuration Guide"

Outline

- Overview
- Resilient Distributed Datasets (RDDs)
- Spark Transformations and Actions
- Spark Programming Model
- Spark Key-Value RDDs (Pair RDDs)
- Shuffles
- **Job Execution**

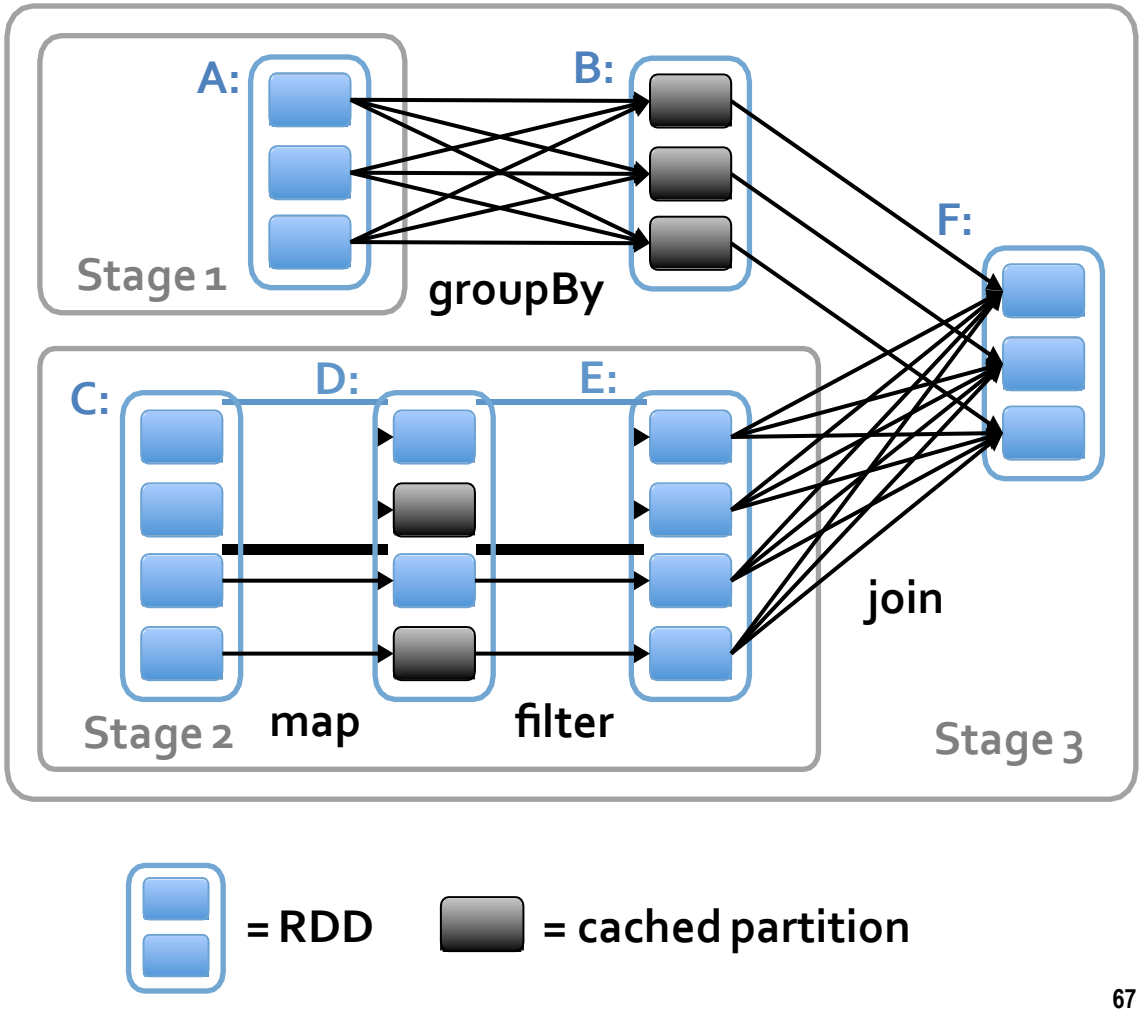
Job Execution

- **Software Components**
  - Spark runs as a library in your program (1 instance per app)
  - Runs tasks locally or on cluster
    - Mesos, YARN or standalone mode
  - Accesses storage systems via Hadoop InputFormat API
    - Can use HBase, HDFS, S3, …



Job Execution

- **Task Scheduler**
  - General task graphs
  - Automatically pipelines functions
  - Data locality aware
  - Partitioning aware to avoid shuffles



Job Execution

- **Task Scheduler: Advanced Features**
  - Controllable partitioning
    - Speed up joins against a dataset
  - Controllable storage formats
    - Keep data serialized for efficiency, replicate to multiple nodes, cache on disk
  - Shared variables: broadcasts, accumulators (in Lecture 4)
  - See online docs for details!

Job Execution

- **Local execution**
  - Just pass `local` or `local[k]` as master URL
  - Debug using local debuggers
    - For Java / Scala, just run your program in a debugger
    - For Python, use an attachable debugger (e.g. PyDev)
  - Great for development & unit tests

Job Execution

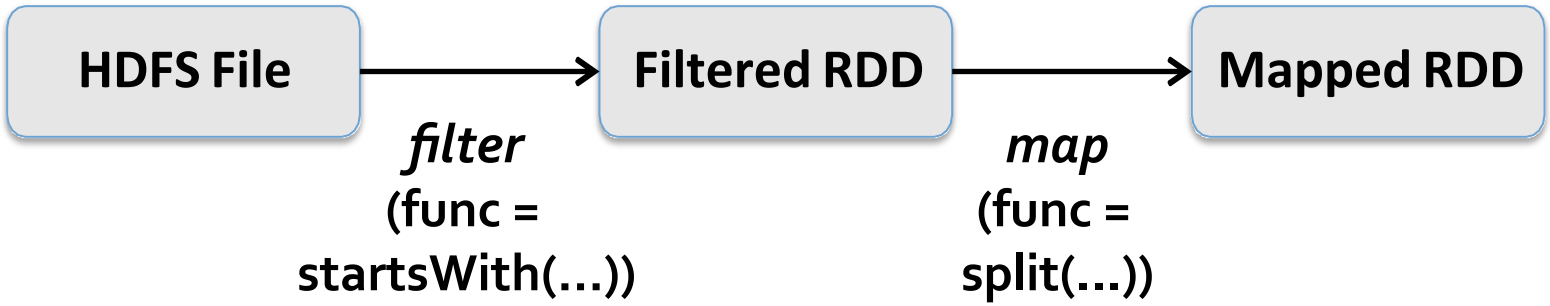
- **Cluster execution**
  - Several options for private clusters:
    - Standalone mode (similar to Hadoop's deploy scripts)
    - Mesos
    - Hadoop YARN
  - `spark-ec2`: Amazon EC2 cluster setup script
    - allows you to launch, manage and shut down Apache Spark clusters on Amazon EC2

```
./spark-ec2 -k keypair –i id_rsa.pem –s num_slaves \
[launch|stop|start|destroy] clusterName
```

Job Execution

- **Fault recovery**
  - RDDs track lineage information that can be used to efficiently recompute lost data

```
msgs = textFile.filter(lambda s: s.startsWith("ERROR"))
                .map(lambda s: s.split("\t")[2])
```



Spark References

- **Spark Programming Guide**
  - http://spark.apache.org/docs/latest/programming-guide.html
- **Spark Python API Docs**
  - http://spark.apache.org/docs/latest/api/python/index.html