APACHE SPARK

Making Interactive Big Data Applications Fast AND Easy

Holden Karau (with thanks to Pat!)

Spark Overview

Goal: easily work with large scale data in terms of transformations on distributed data

- Traditional distributed computing platforms scale well but have limited APIs (map/reduce)
- Spark lets us tackle problems too big for a single machine
- Spark has an expressive data focused API which makes writing large scale programs easy

Scala vs Java API vs Python

Spark was originally written in Scala, which allows concise function syntax and interactive use

Java API added for standalone applications

Python API added more recently along with an interactive shell.

This course: mostly Scala, some translations shown to Java & Python

Outline

Introduction to Scala & functional programming

Spark Concepts

Spark API Tour

Stand alone application

A picture of a cat

Introduction to Scala

What is Scala?

Functions in Scala

Operating on collections in Scala

About Scala

High-level language for the JVM

- Object oriented + functional programming

Statically typed

- Comparable in speed to Java*
- Type inference saves us from having to write explicit types most of the time

Interoperates with Java

- Can use any Java class (inherit from, etc.)
- Can be called from Java code

Best way to Learn Scala

Interactive scala shell (just type scala)

Supports importing libraries, tab completing, and all of the constructs in the language

http://www.scala-lang.org/

Quick Tour of Scala

Declaring variables:

```
var x: Int = 7
var x = 7 // type inferred
val y = “hi” // read-only
```

Functions:

```
def square(x: Int): Int = x*x
def square(x: Int): Int = {
  x*x
}
def announce(text: String) =
{
  println(text)
}
```

Java equivalent:

```
int x = 7;

final String y = “hi”;
```

Java equivalent:

```
int square(int x) {
  return x*x;
}

void announce(String text) {
  System.out.println(text);
}
```

Scala functions (closures)

```
(x: Int) => x + 2 // full version
```

Scala functions (closures)

```
(x: Int) => x + 2 // full version

x => x + 2 // type inferred
```

Scala functions (closures)

```
(x: Int) => x + 2 // full version

x => x + 2 // type inferred

_ + 2 // placeholder syntax (each argument must be used
exactly once)
```

Scala functions (closures)

```
(x: Int) => x + 2 // full version

x => x + 2 // type inferred

_ + 2 // placeholder syntax (each argument must be used
exactly once)

x => { // body is a block of code
    val numberToAdd = 2
    x + numberToAdd
}
```

Scala functions (closures)

```
(x: Int) => x + 2 // full version

x => x + 2 // type inferred

_ + 2 // placeholder syntax (each argument must be used
exactly once)

x => { // body is a block of code
    val numberToAdd = 2
    x + numberToAdd
}

// Regular functions
def addTwo(x: Int): Int = x + 2
```

Quick Tour of Scala Part 2

(electric boogaloo)

Processing collections with functional programming

```
val lst = List(1, 2, 3)
list.foreach(x => println(x)) // prints 1, 2, 3
list.foreach(println)         // same

list.map(x => x + 2)    // returns a new List(3, 4, 5)
list.map(_ + 2)         // same

list.filter(x => x % 2 == 1)// returns a new List(1, 3)
list.filter(_ % 2 == 1)     // same

list.reduce((x, y) => x + y) // => 6
list.reduce(_ + _)           // same
```

All of these leave the list unchanged as it is immutable.

Functional methods on collections

There are a lot of methods on Scala collections, just **google Scala Seq** or http://www.scala-lang.org/api/2.10.4/index.html#scala.collection.Seq

Method on Seq[T]	Explanation
map(f: T => U): Seq[U]	Each element is result of f
flatMap(f: T => Seq[U]): Seq[U]	One to many map
filter(f: T => Boolean): Seq[T]	Keep elements passing f
exists(f: T => Boolean): Boolean	True if one element passes f
forall(f: T => Boolean): Boolean	True if all elements pass
reduce(f: (T, T) => T): T	Merge elements using f
groupBy(f: T => K): Map[K, List[T]]	Group elements by f
sortBy(f: T => K): Seq[T]	Sort elements
…..	

Cat picture from http://galato901.deviantart.com/art/Cat-on-Work-Break-173043455

Spark

Resilient Distributed Data Sets (the core building block)
Log Mining example
Fault Recovery

Spark

Write programs in terms of **transformations** on **distributed datasets**

Resilient Distributed Datasets

- Immutable, partitioned collections of objects spread across a cluster, stored in RAM or on Disk
- Built through lazy parallel transformations
- Automatically rebuilt on failure

Operations

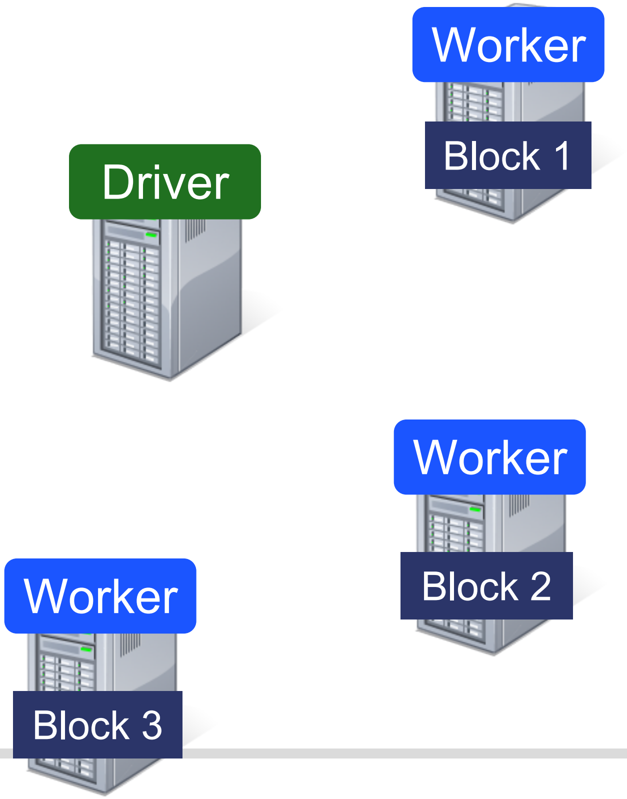
- Transformations (e.g. map, filter, groupBy)
- Actions (e.g. count, collect, save)

RDDs: Distributed

Cat picture by Adam Jones from Kelowna, BC, Canada

RDDs: Distributed

- Data does not have to fit on a single machine
- Data is separated into partitions
  - If we need we can operate on our data partition at a time



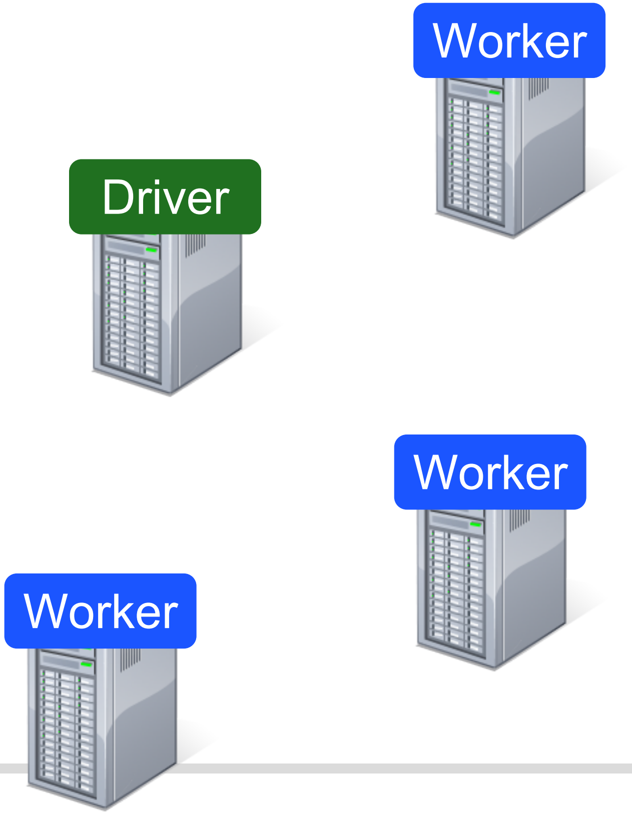
Example: Log Mining

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

“CAT” picture by Shaun Greiner

Example: Log Mining

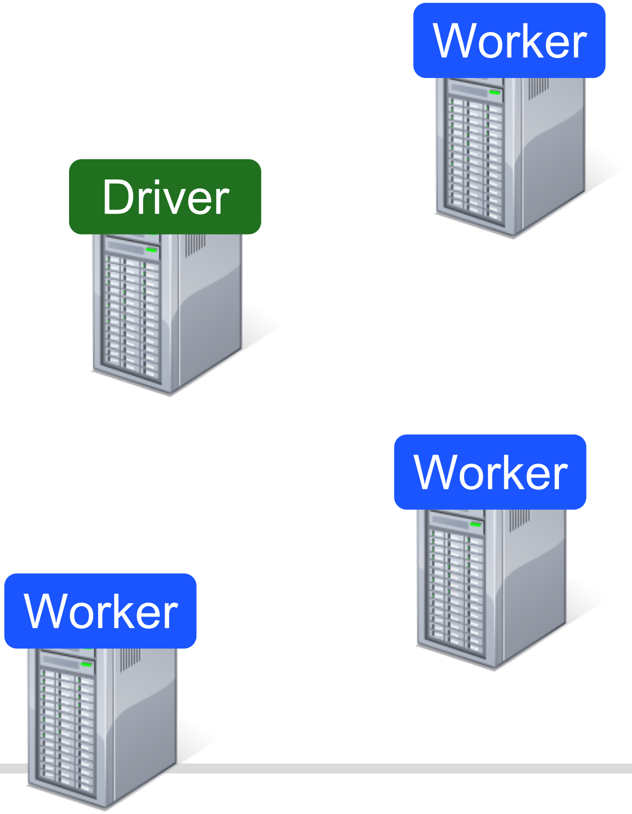
Load error messages from a log into memory, then interactively search for various patterns



Example: Log Mining

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

```
val lines = spark.textFile("hdfs://...")
```

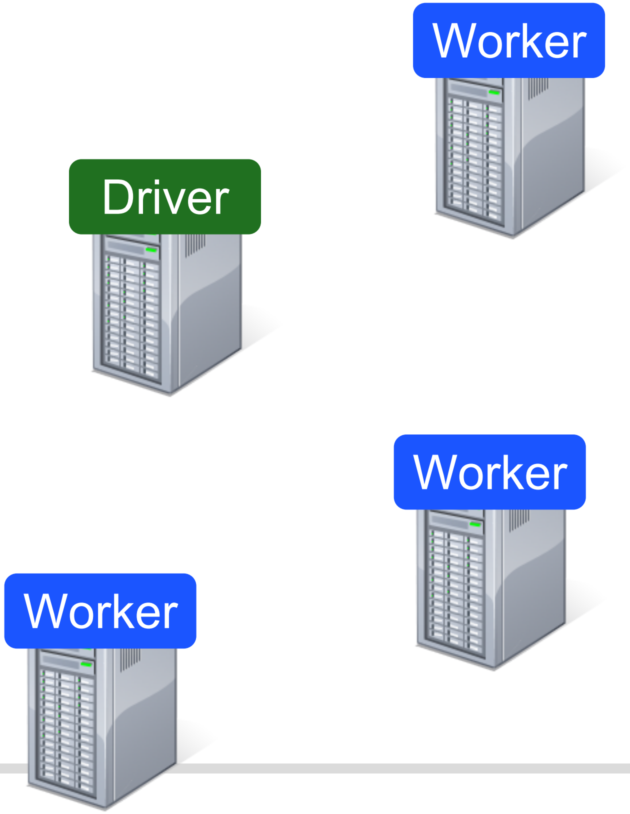


Example: Log Mining

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

Base RDD

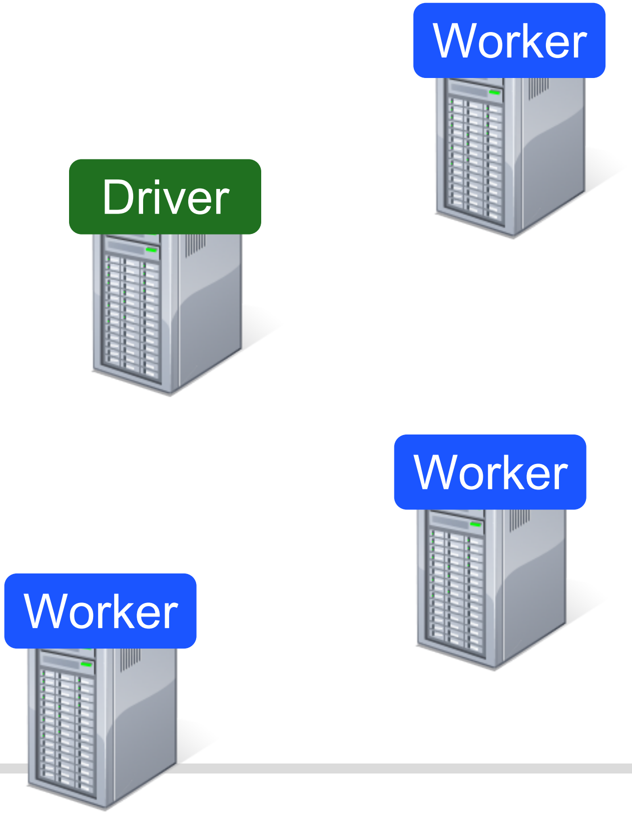
```
val lines = spark.textFile("hdfs://...")
```



Example: Log Mining

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

```
val lines = spark.textFile(“hdfs://...”)
val errors = lines.filter(_.startsWith(“ERROR”))
```

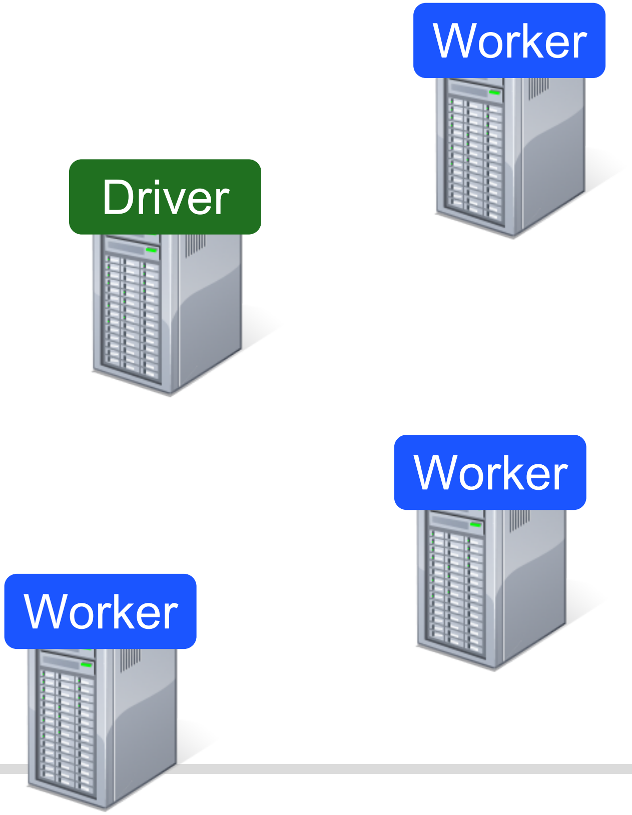


Example: Log Mining

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

Transformed RDD

```
val lines = spark.textFile(“hdfs://...”)
val errors = lines.filter(_.startsWith(“ERROR”))
```



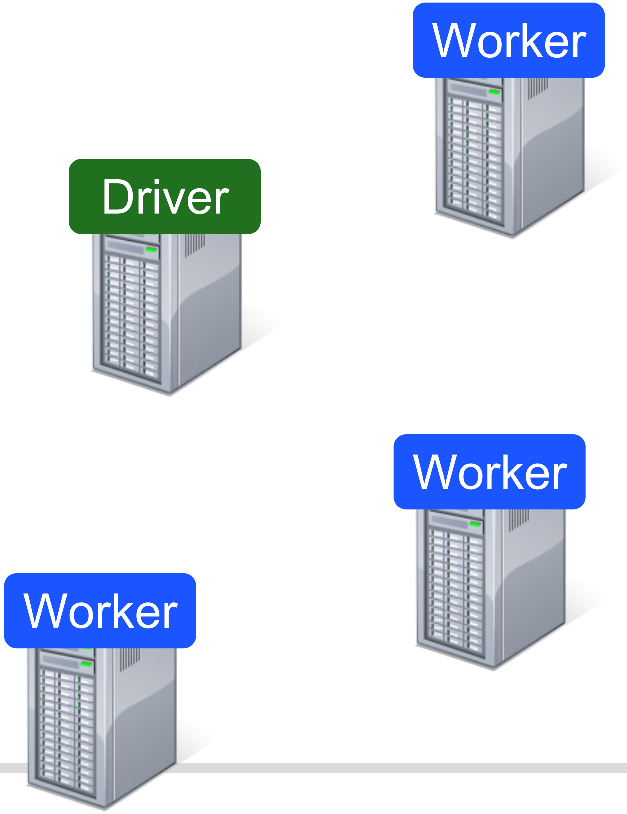
Example: Log Mining

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

```
val lines = spark.textFile(“hdfs://...”)
val errors = lines.filter(_.startsWith(“ERROR”))
val messages = errors.map(_.split(‘\t’)(2))



messages.filter(_.contains(“mysql”)).count()
```



Example: Log Mining

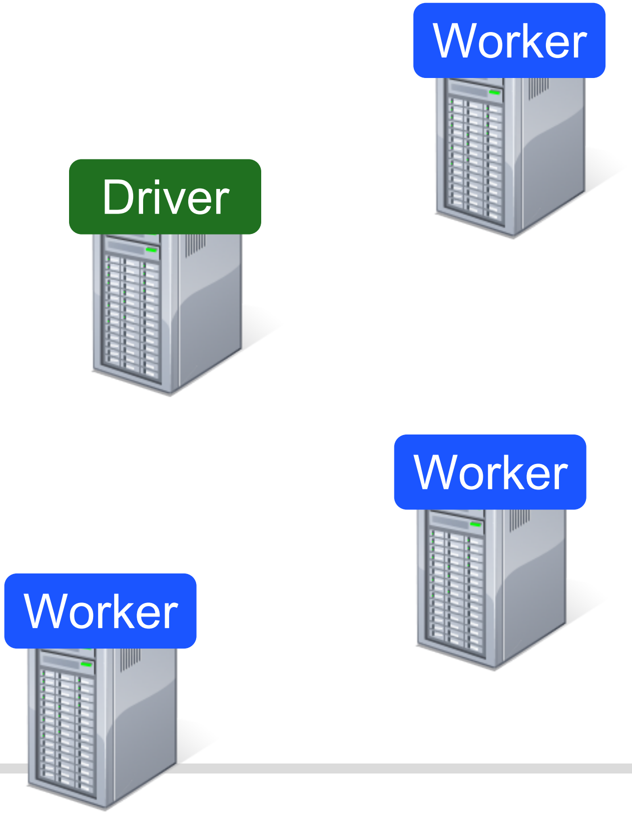
Load error messages from a log into memory, then interactively search for various patterns

```
val lines = spark.textFile(“hdfs://...”)
val errors = lines.filter(_.startsWith(“ERROR”))
val messages = errors.map(_.split(‘\t’)(2))
messages.cache()



messages.filter(_.contains(“mysql”)).count()
```

Cache the RDD



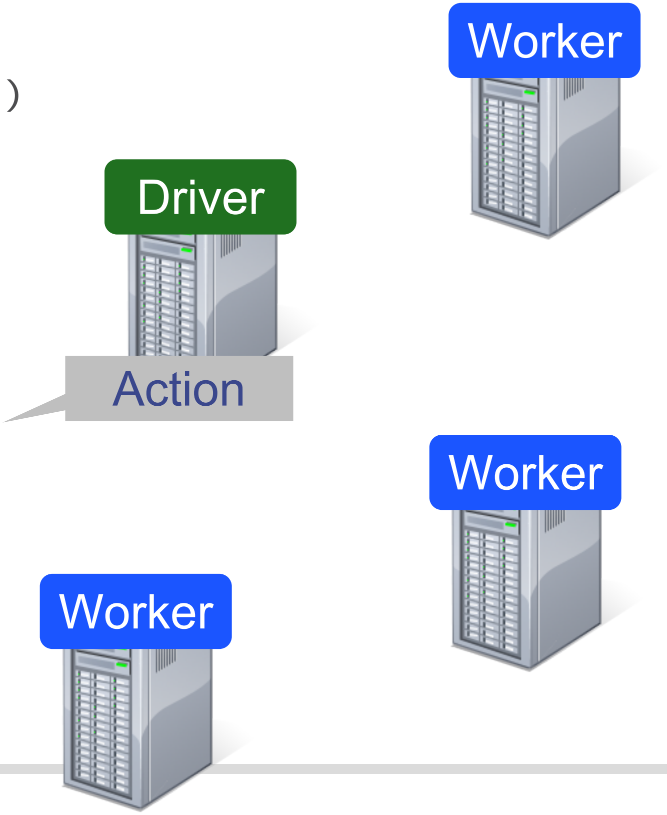
DATABRICKS

Example: Log Mining

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

```
val lines = spark.textFile(“hdfs://...”)
val errors = lines.filter(_.startsWith(“ERROR”))
val messages = errors.map(_.split(‘\t’)(2))
messages.cache()


messages.filter(_.contains(“mysql”)).count()
```

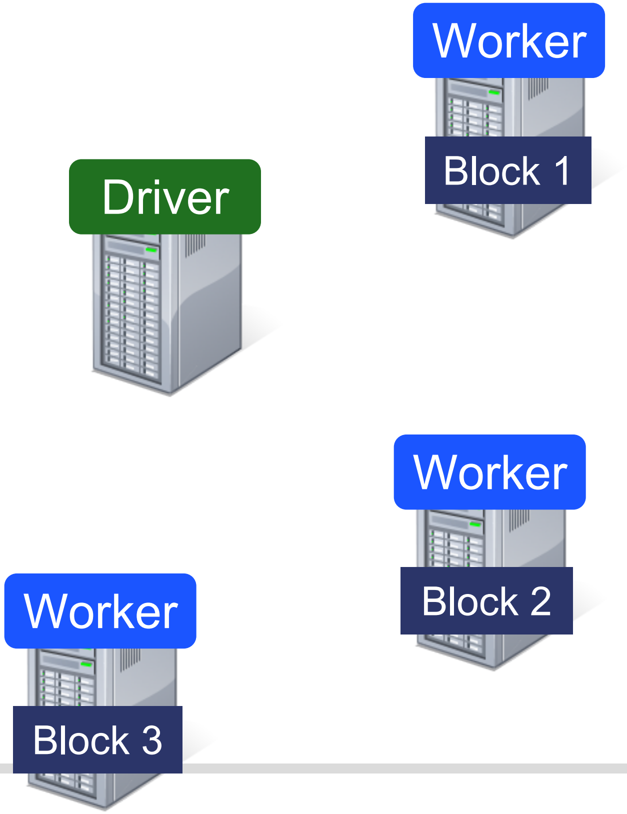


Example: Log Mining

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

```
val lines = spark.textFile(“hdfs://...”)
val errors = lines.filter(_.startsWith(“ERROR”))
val messages = errors.map(_.split(‘\t’)(2))
messages.cache()


messages.filter(_.contains(“mysql”)).count()
```

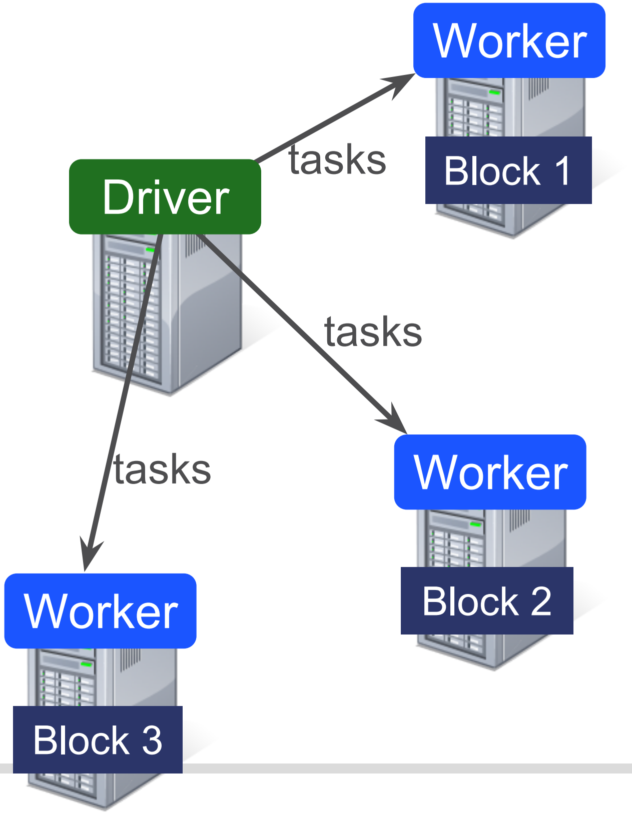


Example: Log Mining

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

```
val lines = spark.textFile(“hdfs://...”)
val errors = lines.filter(_.startsWith(“ERROR”))
val messages = errors.map(_.split(‘\t’)(2))
messages.cache()


messages.filter(_.contains(“mysql”)).count()
```

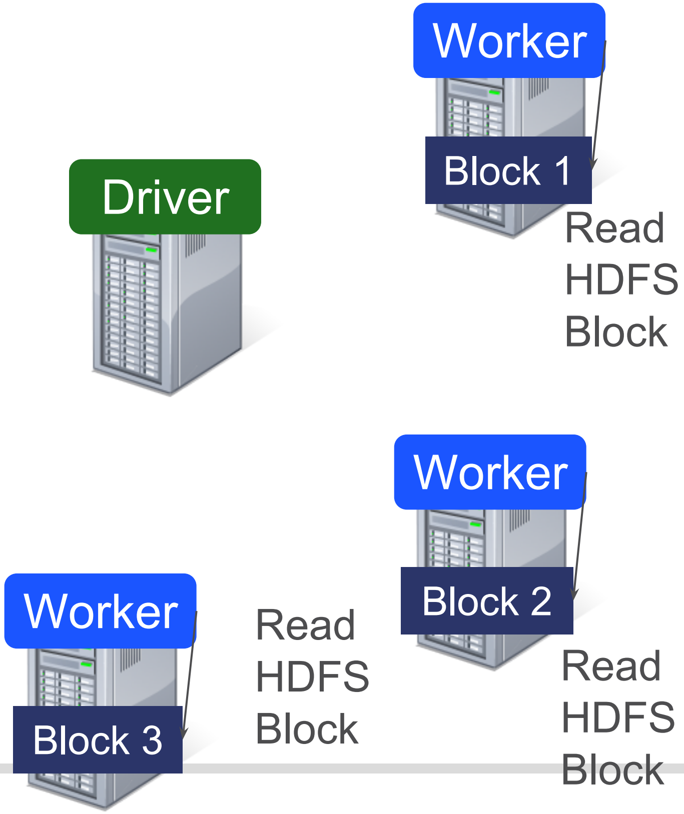


Example: Log Mining

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

```
val lines = spark.textFile(“hdfs://...”)
val errors = lines.filter(_.startsWith(“ERROR”))
val messages = errors.map(_.split(‘\t’)(2))
messages.cache()


messages.filter(_.contains(“mysql”)).count()
```

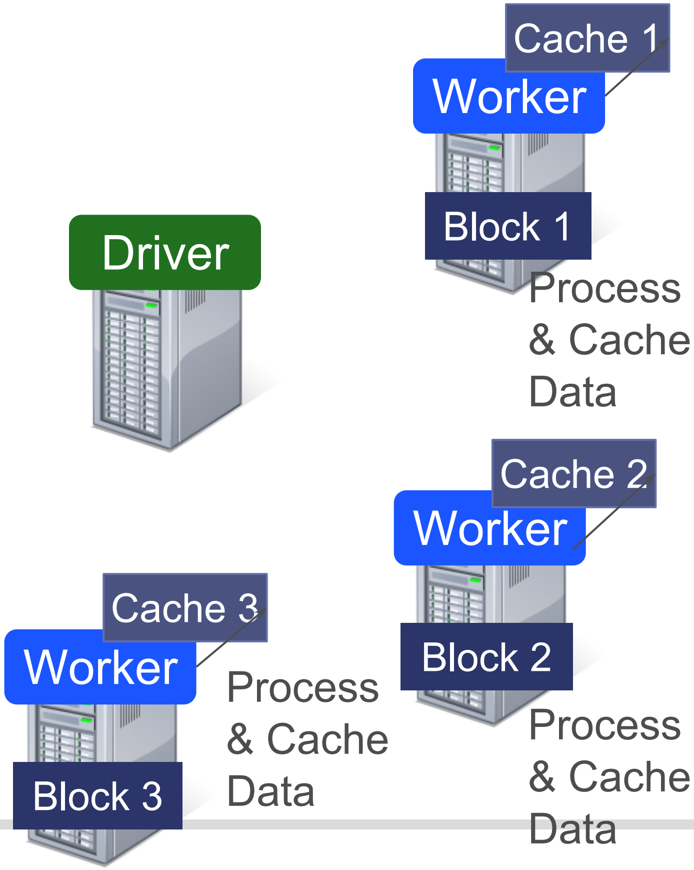


Example: Log Mining

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

```
val lines = spark.textFile("hdfs://...")
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split('\t')(2))
messages.cache()


messages.filter(_.contains("mysql")).count()
```

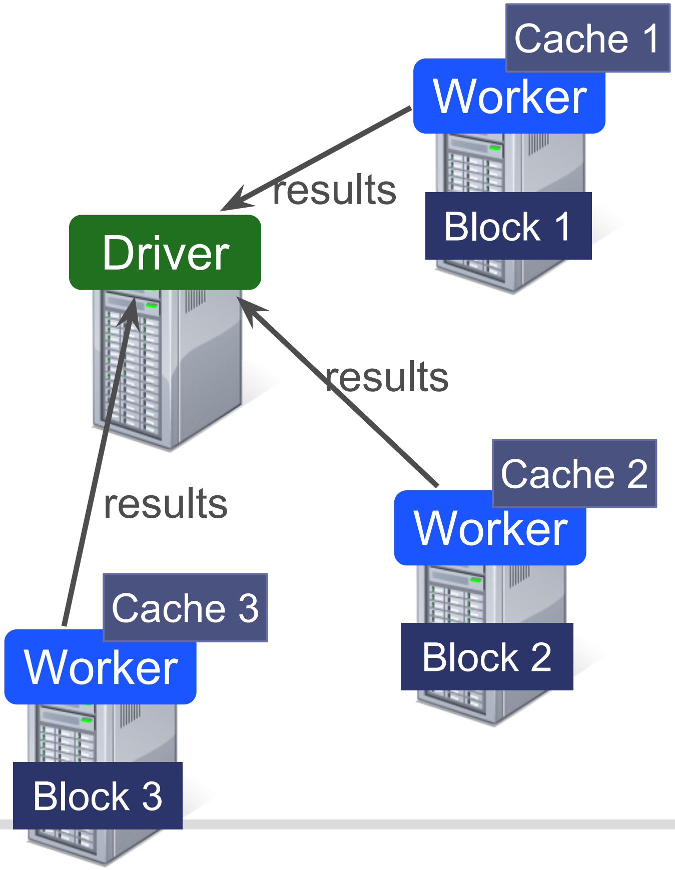


Example: Log Mining

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

```
val lines = spark.textFile(“hdfs://...”)
val errors = lines.filter(_.startsWith(“ERROR”))
val messages = errors.map(_.split(‘\t’)(2))
messages.cache()


messages.filter(_.contains(“mysql”)).count()
```

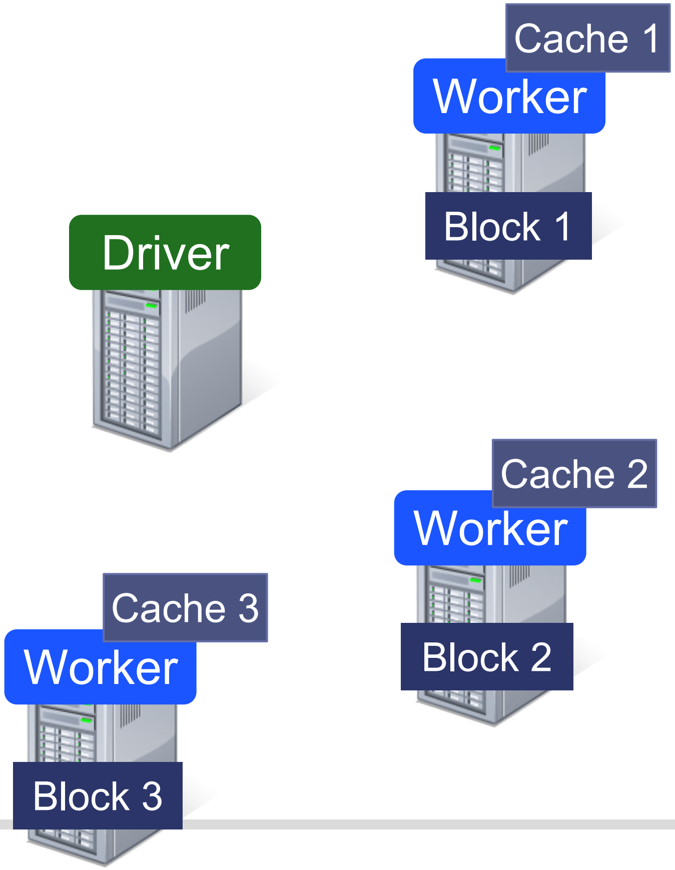


Example: Log Mining

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

```
val lines = spark.textFile(“hdfs://...”)
val errors = lines.filter(_.startsWith(“ERROR”))
val messages = errors.map(_.split(‘\t’)(2))
messages.cache()


messages.filter(_.contains(“mysql”)).count()
messages.filter(_.contains(“php”)).count()
```

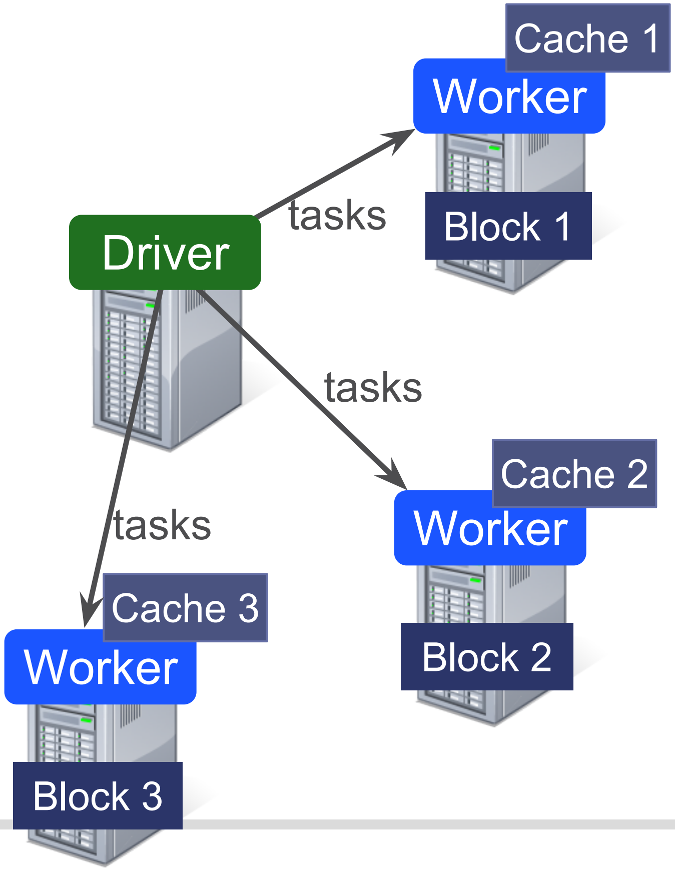


Example: Log Mining

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

```
val lines = spark.textFile("hdfs://...")
val errors = lines.filter(_.startsWith("ERROR"))
val messages = errors.map(_.split('\t')(2))
messages.cache()
```

```
messages.filter(_.contains("mysql")).count()
messages.filter(_.contains("php")).count()
```

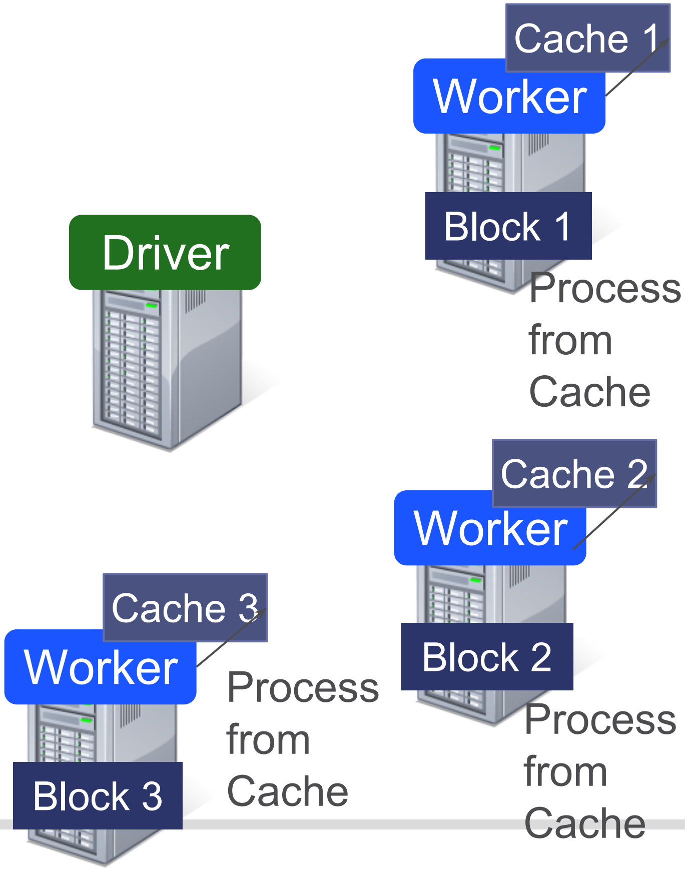


Example: Log Mining

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

```
val lines = spark.textFile(“hdfs://...”)
val errors = lines.filter(_.startsWith(“ERROR”))
val messages = errors.map(_.split(‘\t’)(2))
messages.cache()


messages.filter(_.contains(“mysql”)).count()
messages.filter(_.contains(“php”)).count()
```



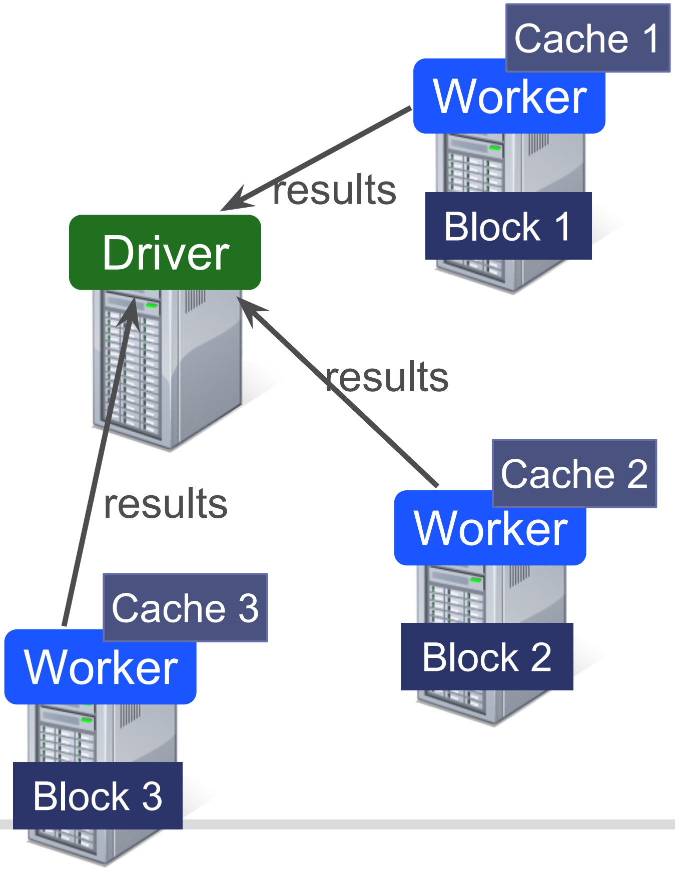
Example: Log Mining

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

```
val lines = spark.textFile(“hdfs://...”)
val errors = lines.filter(_.startsWith(“ERROR”))
val messages = errors.map(_.split(‘\t’)(2))
messages.cache()



messages.filter(_.contains(“mysql”)).count()
messages.filter(_.contains(“php”)).count()
```

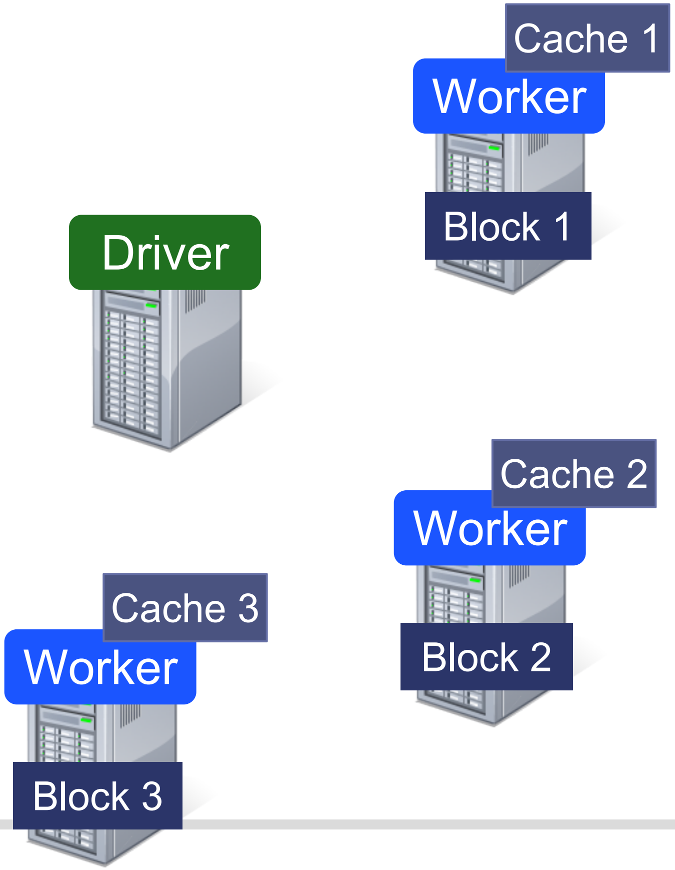


Example: Log Mining

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

```
val lines = spark.textFile(“hdfs://...”)
val errors = lines.filter(_.startsWith(“ERROR”))
val messages = errors.map(_.split(‘\t’)(2))
messages.cache()


messages.filter(_.contains(“mysql”)).count()
messages.filter(_.contains(“php”)).count()
```



Cache your data ➔ Faster Results

1 TB of log data data

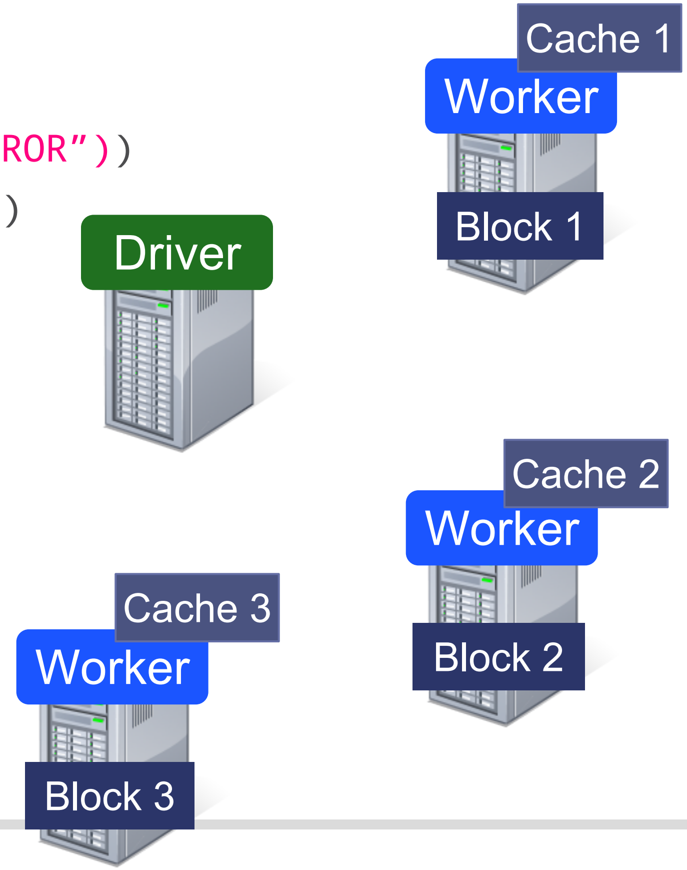
- 5-7 sec from cache vs. 170s for on-disk

Example: Log Mining

Pretty much the same in Python

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


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



Cache your data ➔ Faster Results

1 TB of log data data

- 5-7 sec from cache vs. 170s for on-disk

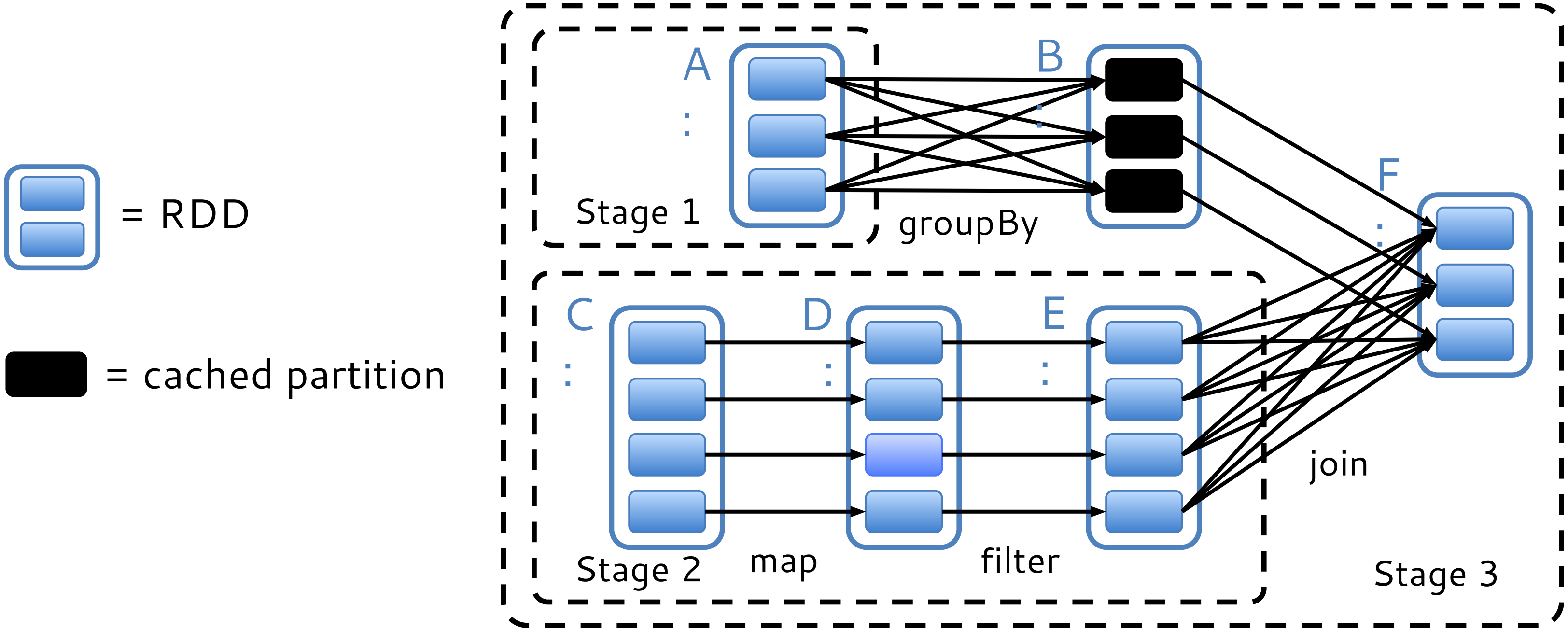
Fast: Using RAM, Operator Graphs

In-memory Caching

- Data Partitions read from RAM instead of disk

Operator Graphs

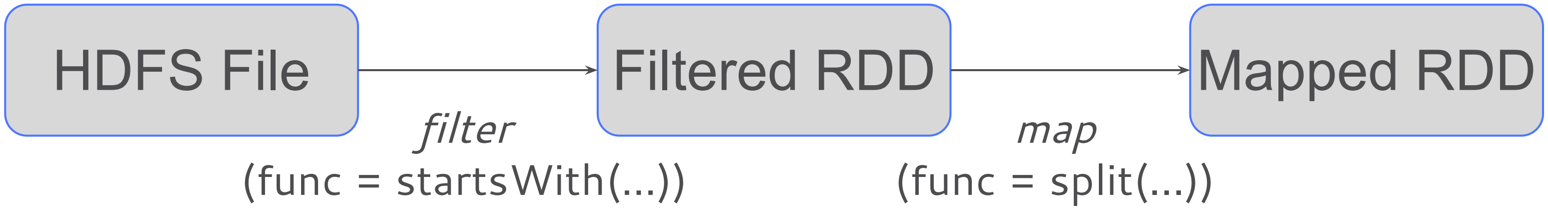
- Scheduling Optimizations
- Fault Tolerance



Fault Recovery

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

```
msgs = textFile.filter(_.contains(“ERROR”))
               .map(_.split(‘\t’)(2))
```



HDFS File

filter
(func = startsWith(…))

Filtered RDD

map
(func = split(...))

Mapped RDD

Tour of Spark operations

API for working with RDDs
Basic operations
Key, Value pairs

Easy: Expressive API

```
map
filter
groupBy
sort
union
join
leftOuterJoin
rightOuterJoin
```

```
reduce
count
fold
reduceByKey
groupByKey
cogroup
cross
zip
```

```
sample
take
first
partitionBy
mapWith
pipe
save     ...
```

More operations listed in online API docs at http://spark.apache.org/docs/latest/api/core/index.html#org.apache.spark.rdd.RDD

Creating RDDs

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

# Turn a Scala collection into an RDD
>sc.parallelize(List(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)
```

Basic Transformations (scala)

```
>val nums = sc.parallelize(List(1, 2, 3))

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

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

// Map each element to zero or more others
>nums.flatMap(x => 0.to(x))
//=> {0, 1, 0, 1, 2, 0, 1, 2, 3}
```

Less Basic Transformations (scala)

```
// Pass each partition through a function
>val squares = nums.mapPartition(x.map(x * x))   // {1
4, 9}
```

Set operations

- **this.union(rdd) -** Produce a new RDD with elements from both rdds (fast!)
- **this.intersect*(rdd) -** surprisingly slow
- **this.cartesian(rdd) -** Produce an RDD with the cartesian product from both RDDs (possibly not very fast)

Basic Actions (scala)

```
>val nums = sc.parallelize(List(1, 2, 3))

// Retrieve RDD contents as a local collection
>nums.collect() //=> List(1, 2, 3)

// Return first K elements
>nums.take(2)   //=> List(1, 2)

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

// Merge elements with an associative function
>nums.reduce{case (x, y) => x + y}  //=> 6

// Write elements to a text file
>nums.saveAsTextFile(“hdfs://file.txt”)
```

Basic Transformations (python)

```
>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}
```

Basic Actions (python)

```
>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”)
```

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
```

Some Key-Value Operations

```
>pets = sc.parallelize(
  List(("cat", 1), ("dog", 1), ("cat", 2)))
>pets.reduceByKey(_ + _)
                  //=> ((cat, 3), (dog, 1))
>pets.groupByKey() //=> {(cat, [1, 2]), (dog, [1])}
>pets.sortByKey()  //=> {(cat, 1), (cat, 2), (dog, 1)}
```

`reduceByKey` also automatically implements combiners on the map side

More PairRDD functions at http://spark.apache.org/docs/latest/api/core/index.html#org.apache.spark.rdd.PairRDDFunctions

Some Key-Value Operations

(python)

```
>pets = sc.parallelize(
  [(“cat”, 1), (“dog”, 1), (“cat”, 2)])
>pets.reduceByKey(lambda x, y: x + y)
                  # => {(cat, 3), (dog, 1)}
>pets.groupByKey() # => {(cat, [1, 2]), (dog, [1])}
>pets.sortByKey()  # => {(cat, 1), (cat, 2), (dog, 1)}
```

`reduceByKey` also automatically implements combiners on the map side

Other Key-Value Operations

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

>pageNames = sc.parallelize(List( (“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”, (Seq(“1.2.3.4”, “1.3.3.1”), Seq(“Home”)))
// (“about.html”, (Seq(“3.4.5.6”), Seq(“About”)))
```

Setting the Level of Parallelism

All the pair RDD operations take an optional second parameter for number of tasks

```
> words.reduceByKey(_ + _, 5)
> words.groupByKey(5)
> visits.join(pageViews, 5)
```

Can also set the `spark.default.parallelism` property

Using Local Variables

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

```
>  val query = “pandas”
>  pages.filter(_.contains(query))
        .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!)

Complete App (Scala)

```
import org.apache.spark._
import org.apache.spark.SparkContext._

object WordCount {
    def main(args: Array[String]) {
      val sc = new SparkContext(args(0), "BasicMap",
                                System.getenv("SPARK_HOME"))
      val input = sc.textFile(args(1))
      val counts = input.flatMap(_.split(" "))
                        .map((_, 1)).reduceByKey(_ + _)
      counts.saveAsTextFile(args(2))
    }
}
```

Getting Spark

Download: http://spark.apache.org/downloads.html

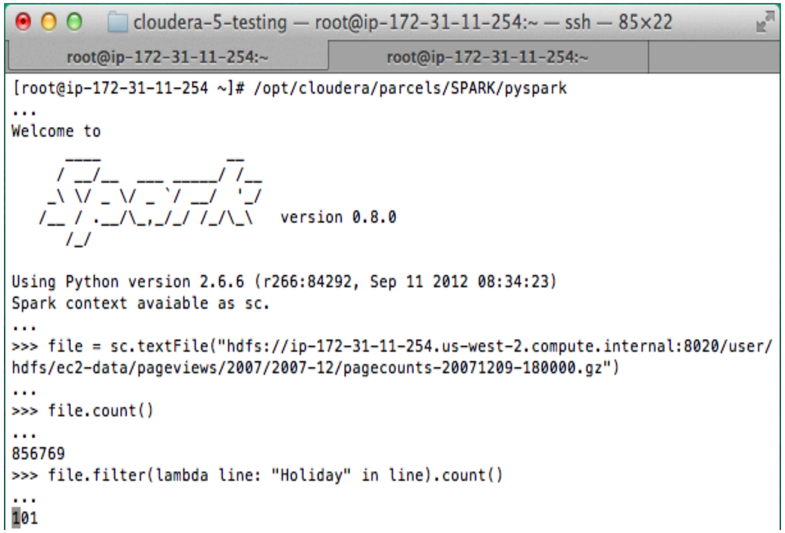
Link with Spark in your sbt/maven project:

groupId: org.apache.spark
artifactId: spark-core_2.10
version: 0.9.0-incubating

Using the Shell

Launching:

```
spark-shell
pyspark (IPYTHON=1)
```



```
[root@ip-172-31-11-254 ~]# /opt/cloudera/parcels/SPARK/pyspark
...
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 0.8.0
      /_/

Using Python version 2.6.6 (r266:84292, Sep 11 2012 08:34:23)
Spark context avaiable as sc.
...
>>> file = sc.textFile("hdfs://ip-172-31-11-254.us-west-2.compute.internal:8020/user/
hdfs/ec2-data/pageviews/2007/2007-12/pagecounts-20071209-180000.gz")
...
>>> file.count()
...
856769
>>> file.filter(lambda line: "Holiday" in line).count()
...
101
```

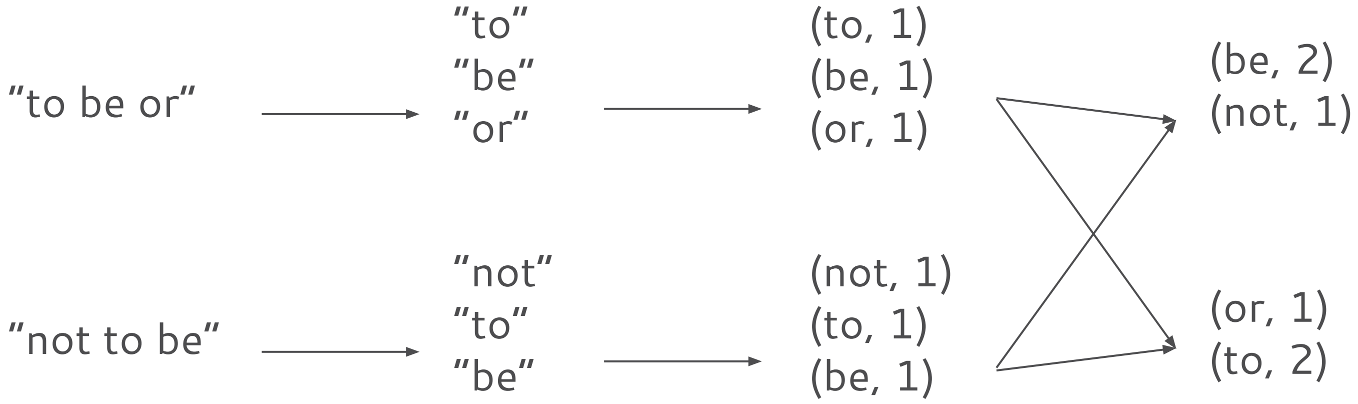
Modes:

```
MASTER=local     ./spark-shell    # local, 1 thread
MASTER=local[2] ./spark-shell    # local, 2 threads
MASTER=spark://host:port ./spark-shell  # cluster
```

Cat photo from https://www.flickr.com/photos/deerwooduk/579761138/in/photolist-4GCc4z-4GCbAV-6Ls27-34evHS-5UBnJv-TeqMG-4iNNn5-4w7s61-6GMLYS-6H5QWY-6aJLUT-tqfrf-6mJ1Lr-84kGX-6mJ1GB-vVqN6-dY8aj5-y3jK-7C7P8Z-azEtd/

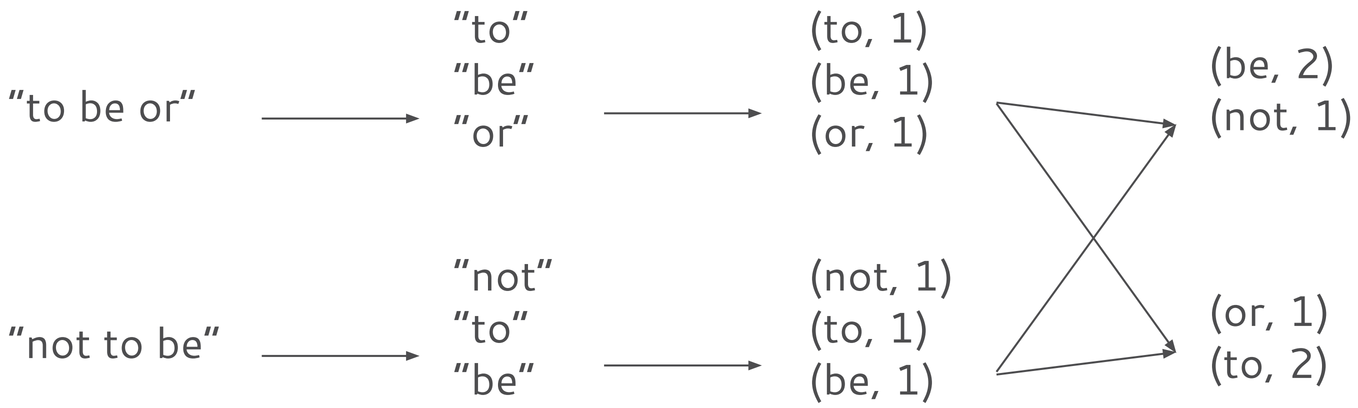
Example: Word Count

```
>val lines = sc.textFile(“hamlet.txt”)
>val counts = lines.flatMap(_.split(“ ”))
                   .map((_, 1))
                   .reduceByKey(_ + _)
```



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)
```



“to be or”

“to”
“be”
“or”

(to, 1)
(be, 1)
(or, 1)

“not to be”

“not”
“to”
“be”

(not, 1)
(to, 1)
(be, 1)

(be, 2)
(not, 1)

(or, 1)
(to, 2)