Spark Training 101

Melbourne

Australia

April 2015 @ Telstra

Ned Shawa

Mark Moloney

Tim Findley

Kon

Agenda

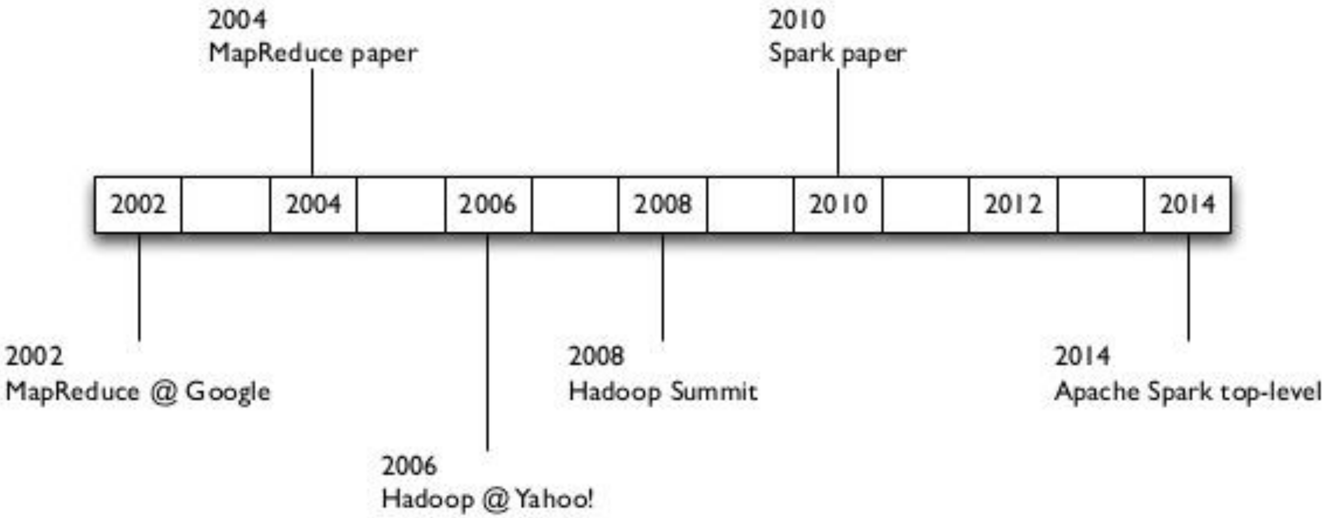
- Spark 101 (Ned) 6:00 – 7:00
- Introduction to Scala (Mark) 7:00-7:30
- Eclipse and Spark (Tim) 7:30 – 8:00
- IntelliJ and Spark (Kon) 8:00- 8:30

Demo Prep Work

- USB will rotate across all of you for copying the USB folder
- You will run spark from your USB folder
- If you prefer Linux and you are running windows we have a vmware and a virtual box appliance that you can run
- You will have a copy of the slides in PDF in the USB folder for following commands

Basics

A Brief History: *Functional Programming for Big Data*



The need for Spark

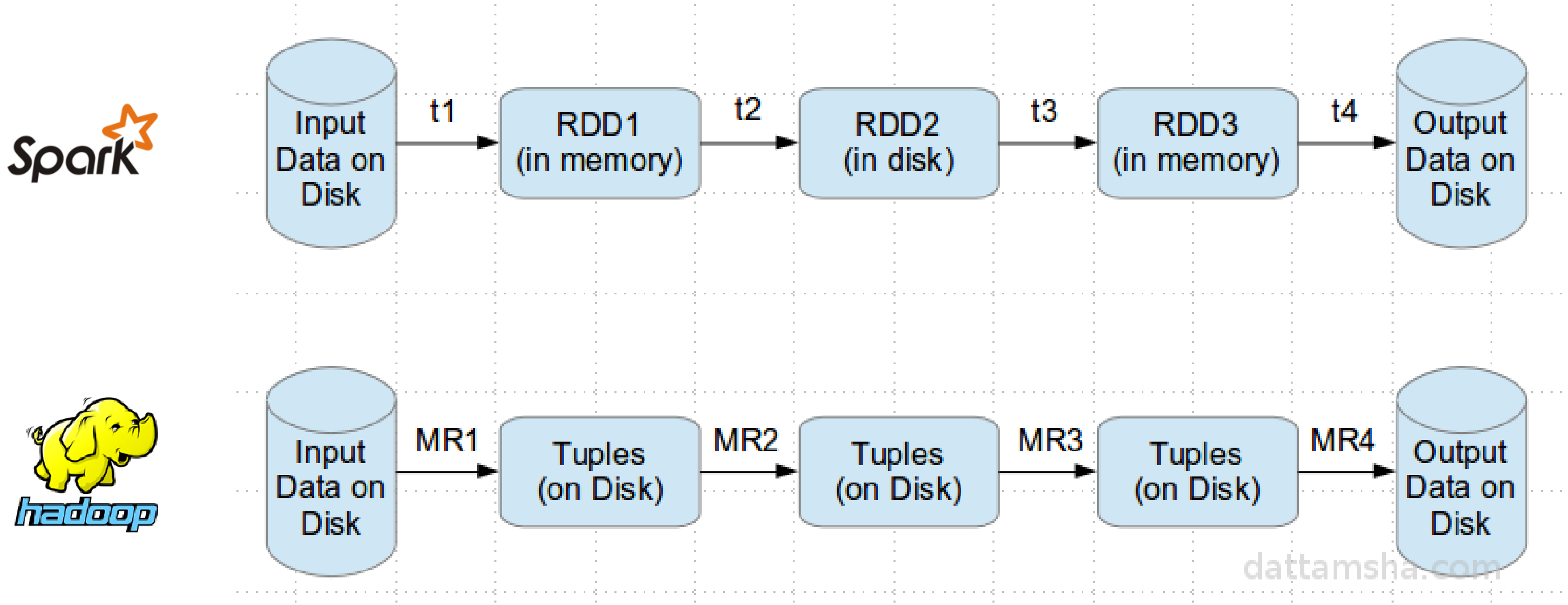
- Data movement across disks is expensive
- Apps and data reload results in re-fetching the dataset everytime
- Caching is limited
- Different data access layers are required for each usecase:
  - SQL(Hive,Hawq,Impala,..etc)
  - Graph
  - MR
  - Machine Learning

Solution

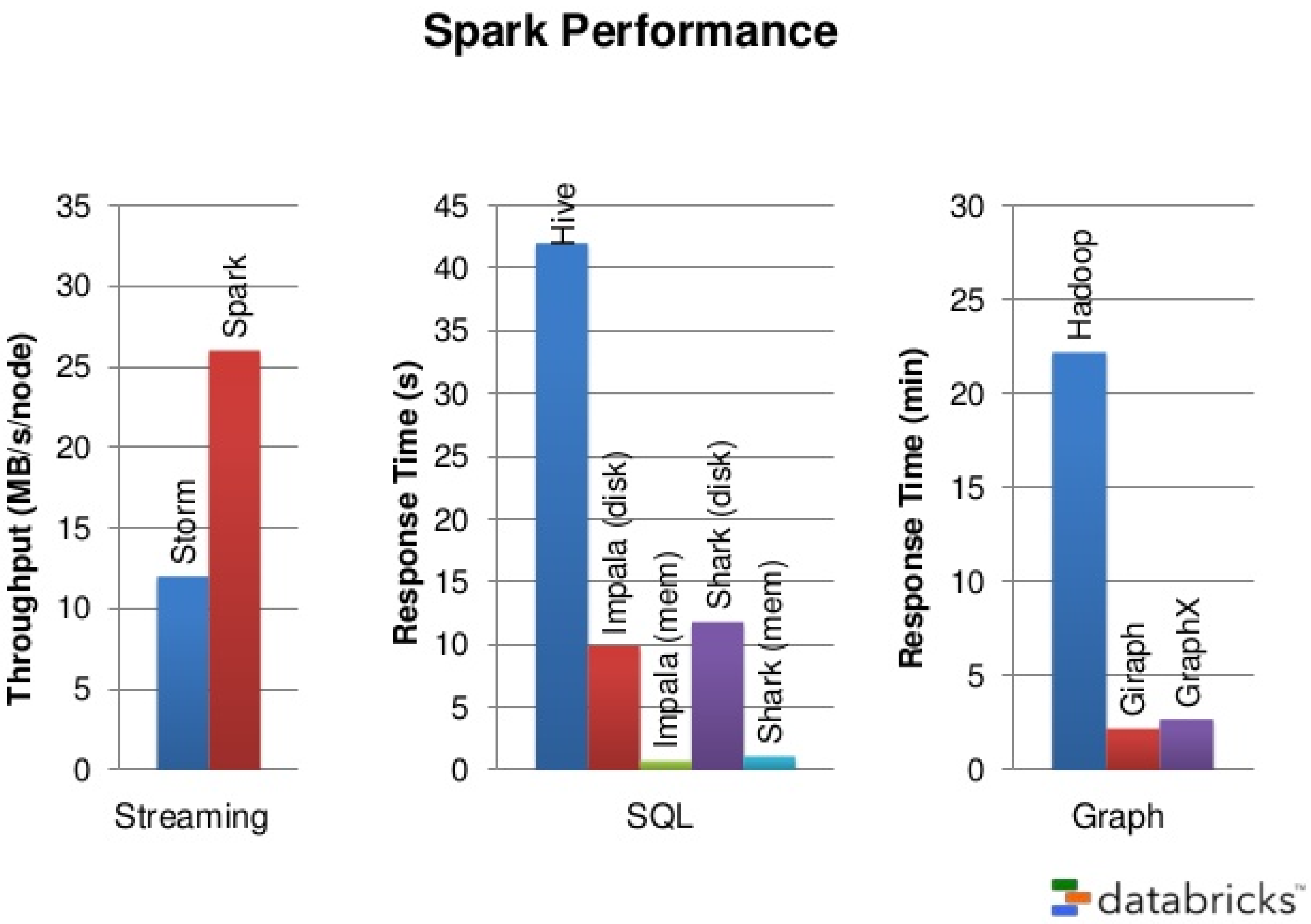
Resilient Distributed Datasets (RDDs)

Allow Apps to keep working sets in memory for efficient reuse Retain the attractive properties of MapReduce » Fault tolerance, data locality, scalability and Support a wide range of applications

And this means?



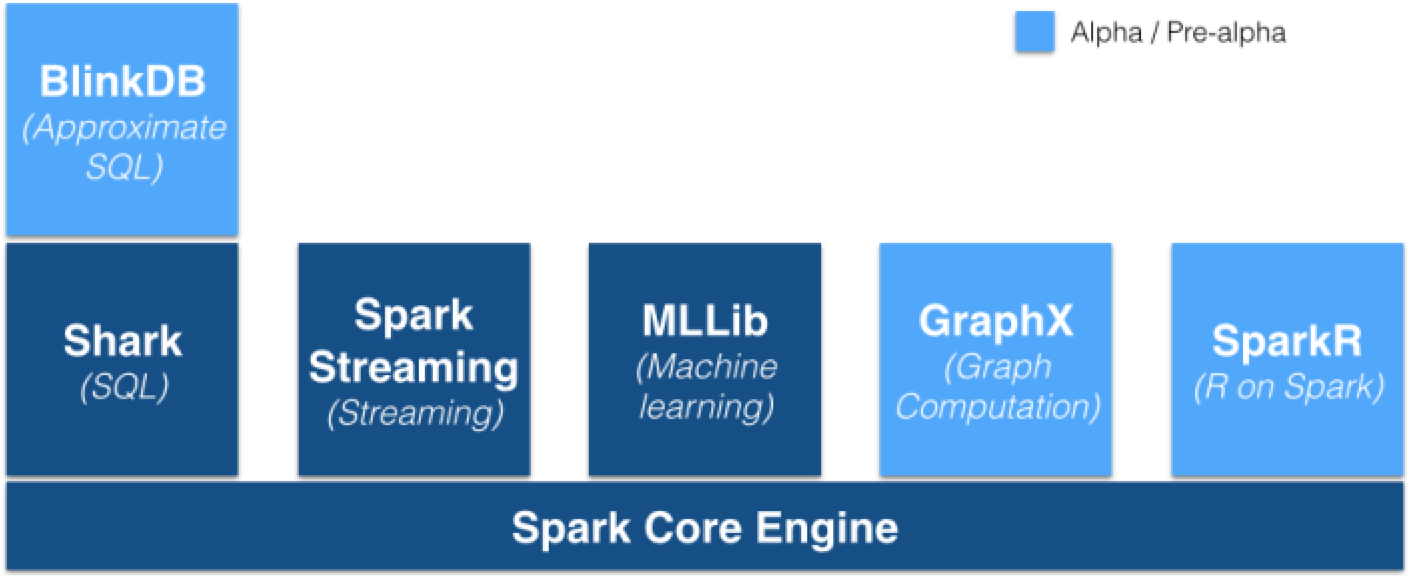
And also means



Spark Programming Model

- Key idea: *resilient distributed datasets (RDDs)*
  - Distributed collections of objects that can be cached in memory across cluster nodes
  - Manipulated through various parallel operators
  - Automatically rebuilt on failure
- Interface
  - Clean language-integrated API in Scala
  - Can be used *interactively* from Scala console

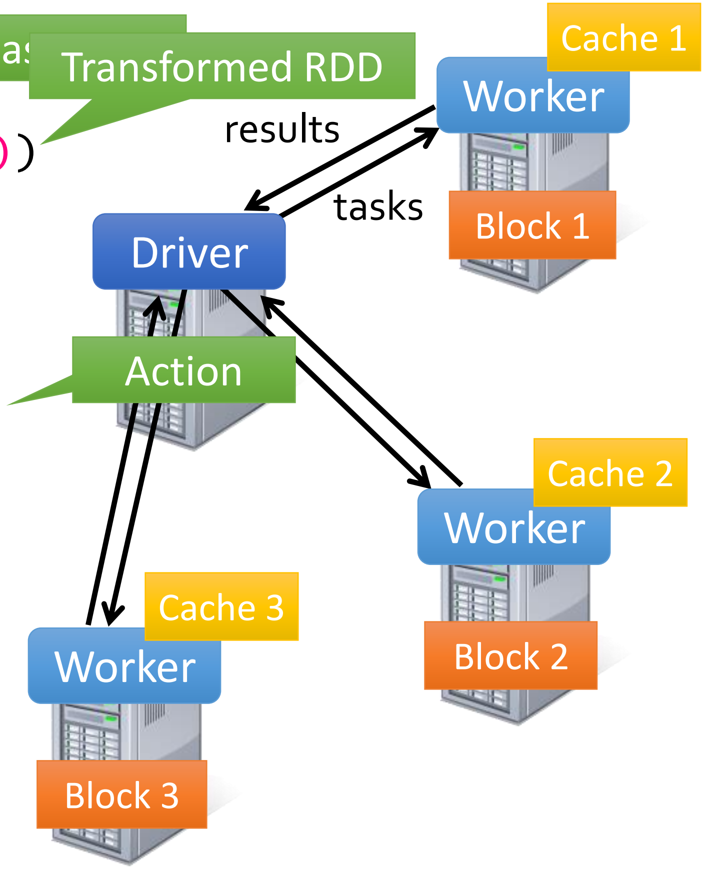
Spark Components (Updated)



Example: Log Mining

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

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

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```



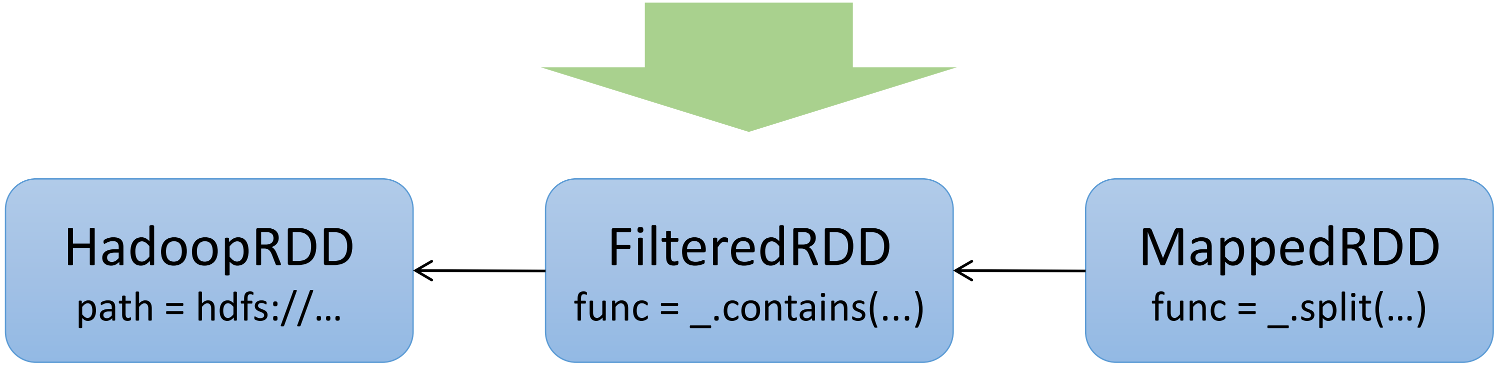
Result: scaled to 1 TB data in 5-7 sec
(vs 170 sec for on-disk data)

Fault Tolerance

RDDs track the series of transformations used to build them (their *lineage*) to recompute lost data

E.g:

```
messages = textFile(...).filter(_.contains("error"))
                        .map(_.split(‘\t’)(2))
```



Example: Logistic Regression

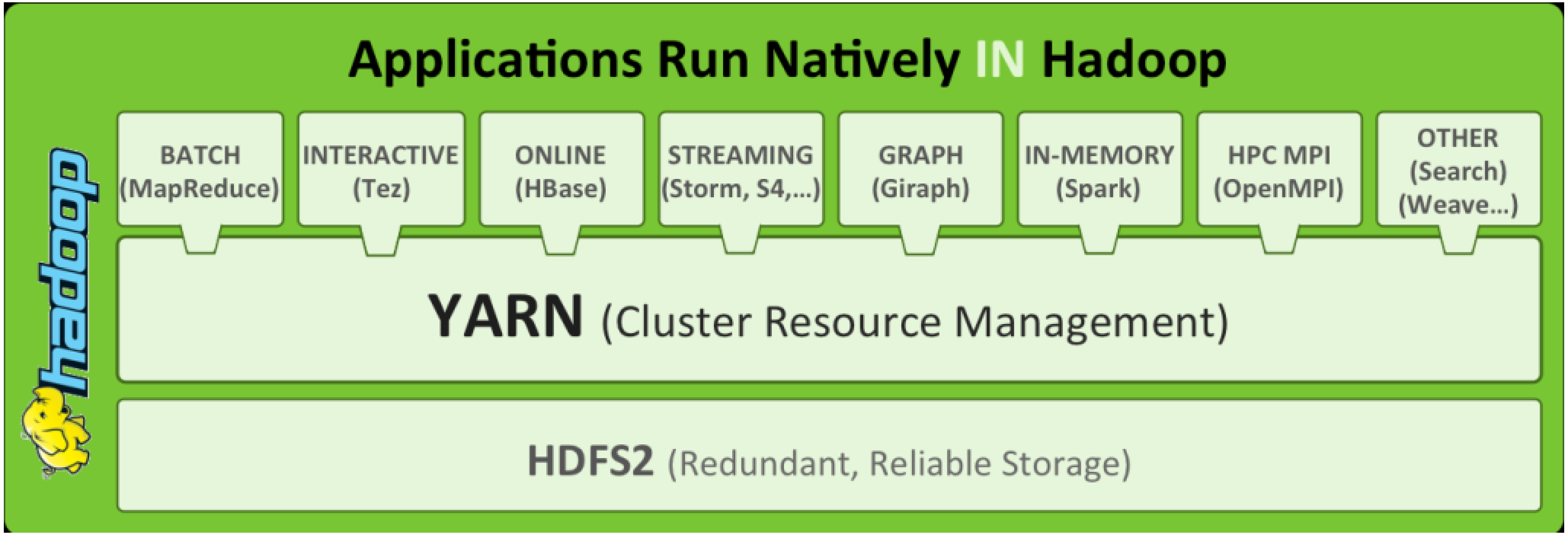
```
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
```

Load data in memory once

Initial parameter vector

Repeated MapReduce steps to do gradient descent

YARN



Applications Run Natively IN Hadoop
BATCH
(MapReduce)
INTERACTIVE
(Tez)
ONLINE
(HBase)
STREAMING
(Storm, S4,...)
GRAPH
(Giraph)
IN-MEMORY
(Spark)
HPC MPI
(OpenMPI)
OTHER
(Search)
(Weave...)
YARN (Cluster Resource Management)
HDFS2 (Redundant, Reliable Storage)
hadoop

Spark Deployment

- Standalone (Mac/Linux/Windows…etc)
- Hadoop / Yarn (Pivotal,Hortonworks..etc)
- Mesos (Cluster Computing)

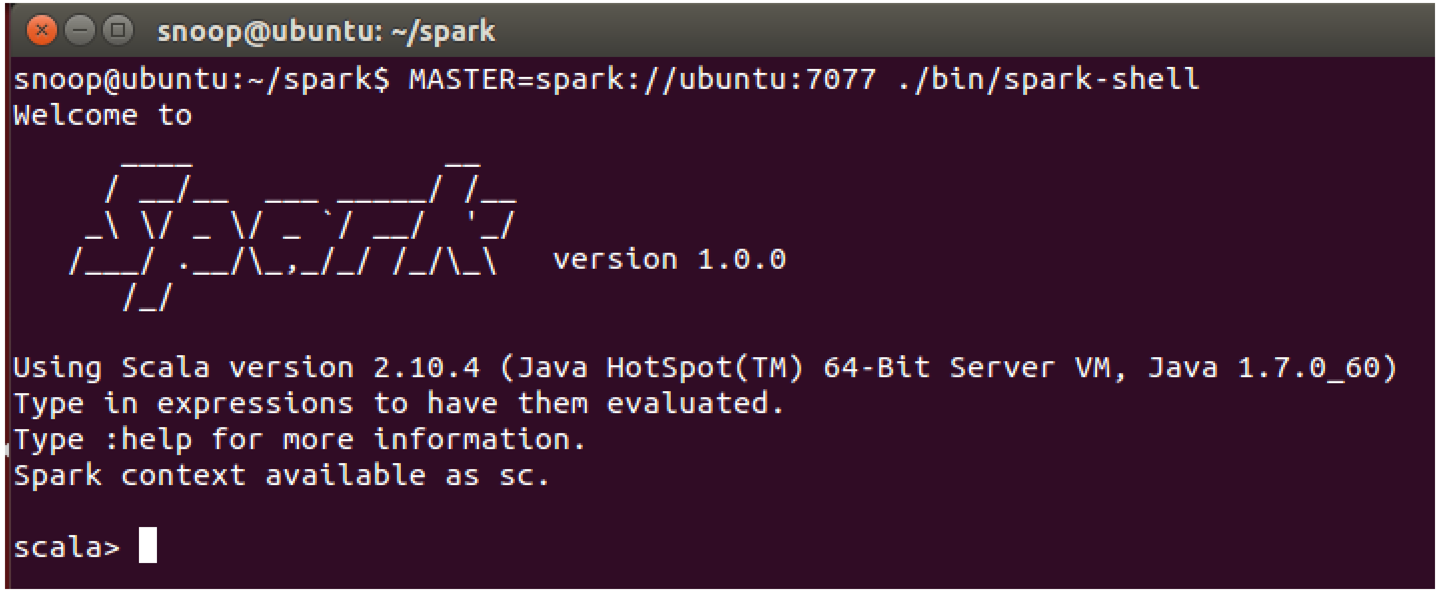
Building Spark

- Maven, Gradle Or SBT
- Options depends on target:
  - Hive Thrift Server
  - Hadoop Version
  - …etc

- Example: ***mvn -Pyarn -Phadoop-2.4 -Dhadoop.version=2.4.0 -DskipTests clean package***
- Refer to https://spark.apache.org/docs/latest/building-spark.html

Running Spark

- Interactive Shell(/bin/spark-shell):
  - Python
  - Scala
- Built in Apps
  - Java
  - Scala
  - Python



Spark UI

Spark Master at spark://127.0.0.1:7077

URL: spark://127.0.0.1:7077
Workers: 1
Cores: 6 Total, 6 Used
Memory: 8.5 GB Total, 512.0 MB Used
Applications: 1 Running, 1 Completed
Drivers: 0 Running, 0 Completed

1

Workers

2

Id	Address	State	Cores	Memory
worker-20141001184107-127.0.0.1-55410	127.0.0.1:55410	ALIVE	6 (6 Used)	8.5 GB (512.0 MB Used)

Running Applications

3

ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20141001184908-0000	Spark shell	6	512.0 MB	2014/10/01 18:49:08	russellspitzer	RUNNING	52 min

Completed Applications

4

ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20141001184917-0001	Spark shell	0	512.0 MB	2014/10/01 18:49:17	russellspitzer	FINISHED	52 min

Spark Context (SC)

- Represents the connection to the cluster
- One SC is only allowed per JVM
- Allows you to interact and create RDDs

```
Val file = sc.textFile("some dataset")....
```

Exploring Spark

Loading dataset

```
val water_data= sc.textFile("/usb/data/vic_water.csv")
```

Basic operations

```
water_data.count()
water_data.first()
water_data.last()....
```

Exploring Spark

Filter

```
val coburg = water_data.filter(line=>line.contains("Coburg"))
coburg.collect()
```

Caching

```
water_data.persist()
water_data.unpersist()
```

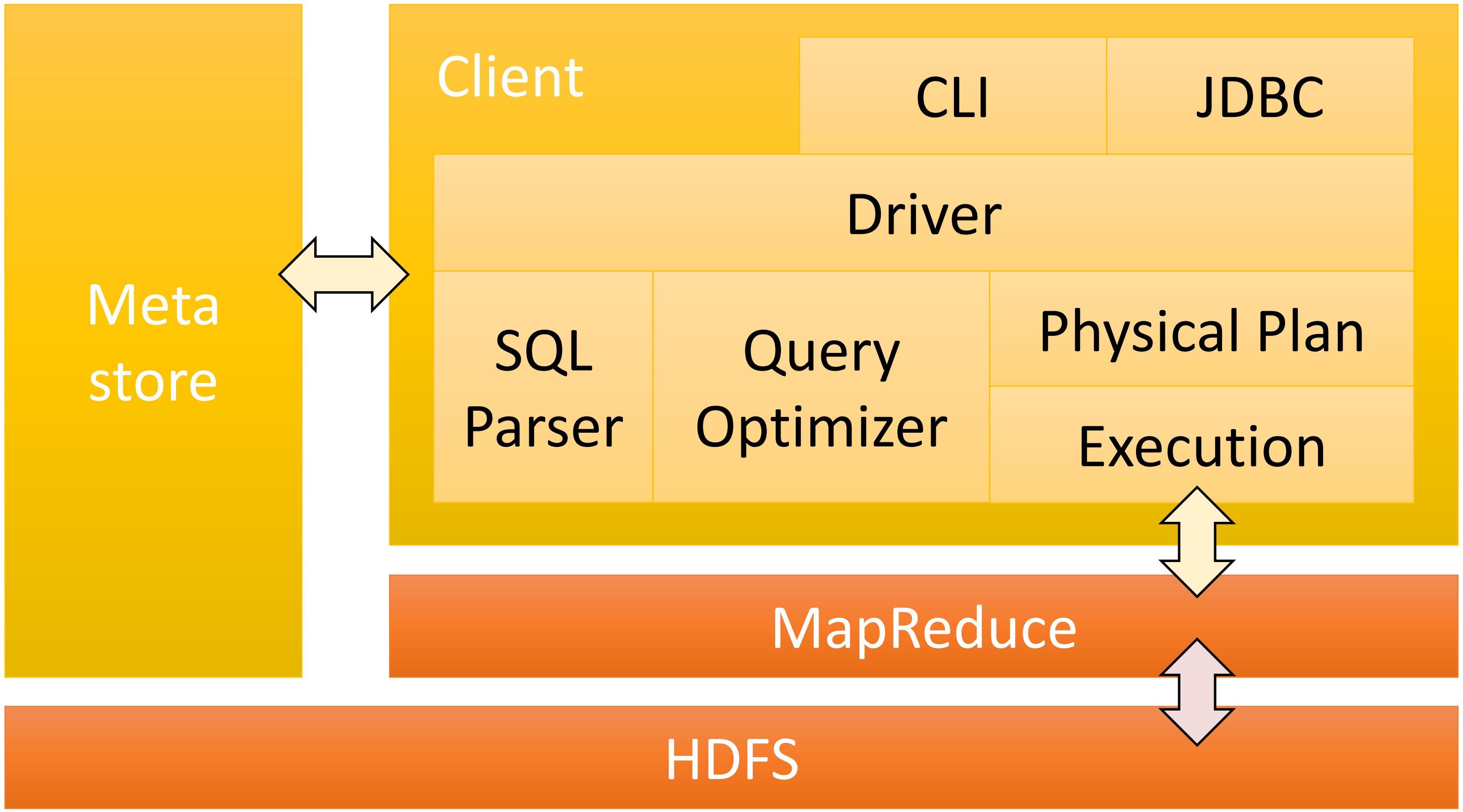
Spark SQL

- Subset of HiveQL
- Uses SchemaRDD
- Direct integration with Hive
- Many sources (JDBC,Text,JSON,Parquet…etc)

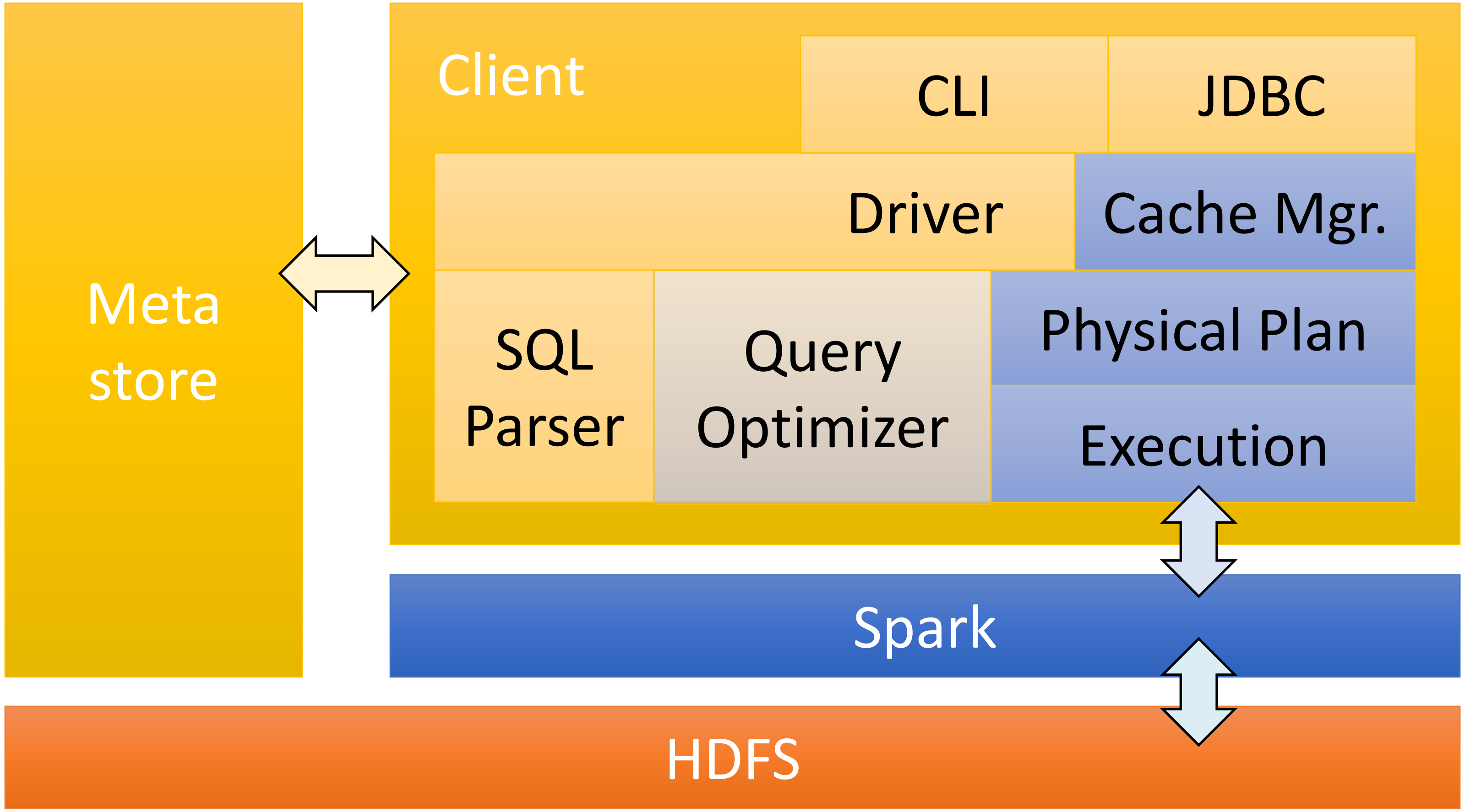
Motivation

- Hive is great, but Hadoop’s execution engine makes even the smallest queries take minutes
- Scala is good for programmers, but many data users only know SQL
- **Can we extend Hive to run on Spark?**

Hive Architecture



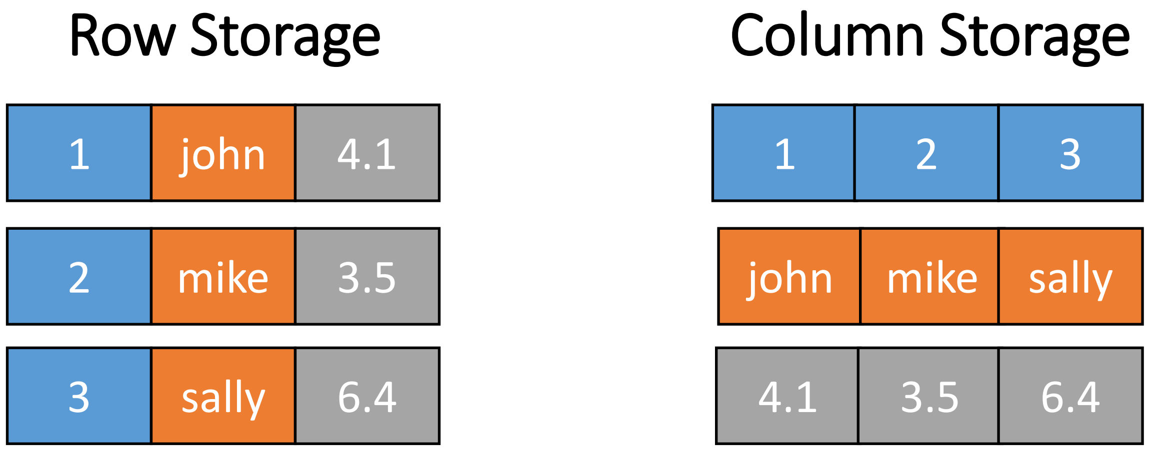
Spark –SQL Architecture



[Engle et al, SIGMOD 2012]

Efficient In-Memory Storage

- Simply caching Hive records as Java objects is inefficient due to high per-object overhead
- Instead, Spark-SQL employs column-oriented storage using **arrays of primitive types**



Efficient In-Memory Storage

- Simply caching Hive records as Java objects is inefficient due to high per-object overhead
- Instead, Spark-SQL employs column-oriented storage using **arrays of primitive types**

Row Storage

Column Storage

Benefit: similarly compact size to serialized data, but >5x faster to access

3	sally	6.4

4.1	3.5	6.4

Spark SQL (1.3)

Built-In

{ JSON }
JDBC
Parquet
HIVE
MySQL
PostgreSQL
HDFS
amazon web services S3
H2

External

AVRO
CSV
dBase
APACHE HBASE
elasticsearch.
cassandra
amazon web services Amazon Redshift

and more…

What are DataFrames?

- Distributed Collection of Data organized in Columns
- Equivalent to Tables in Databases or DataFrame in R/PYTHON
- Much richer optimization than any other implementation of DF
- Can be constructed from a wide variety of sources and APIs

Exploring Spark SQL

```
import sqlContext.implicits._
case class water (postcode: Int, suburb:String, cons_08: Int, cons_09: Int)

val water_table
=water_data.map(_.split("|")).map(t=>water(t(0).trim.toInt,t(1),t(2).trim.toInt,t(3).tri
m.toInt)).toDF()

water_table.registerTempTable("water")
```

Reading/Writing a DataFrame

```
val df = sqlContext.load("/usb/data/people.json", "json")
Val df =sqlContext.load("/usb/data/users.parquet", "parquet")
df.show()
df.printSchema()
df.select ("name").show()
df.select("name","favorite_color").show()
df.select("name").save("/usb/data/names.parquet", "parquet")
df.registerTempTable("df")
sqlContext.sql("select * from df").foreach(println)
```

A Brief Introduction to Scala
(and how it can help you build better data apps)

Mark Moloney
markmo@me.com

The moot

- Functional programming style works well with data-driven applications
- Scala supports a functional programming style without losing integration options and operational support of the Java ecosystem
- Knowing Scala == Spark power user

Some facts and history

- Scala is an **object-functional** language
- Object-oriented, C-style syntax (but with less boilerplate)
- Has features of FP languages (e.g. Haskell, Scheme)
- Statically typed but uses type inference (can sometimes appear dynamically typed)
- A key focus of the language has been on making development of concurrent systems easier.
- Scala code compiles to Java bytecode to leverage the Java Virtual Machine (JVM)
  - 20 years of tuning to perform comparably to C, plus a wealth of libraries, management tools, and acceptance by most ops groups of large companies
  - You can create Java classes and call Java methods directly from Scala

- It was created by Martin Odersky (who had previously worked on the Java compiler) and publically released in early 2004
- Commercial support for the language is provided by Typesafe, a company founded by Odersky

What is Functional Programming anyway?

- Imperative programming has been the norm:
  - modifying mutable variables (meaning data that can change after initialization)
  - using assignments
  - and control structures such as if-then-else, loops, etc.
- However, “Concurrency is the Dr. Evil to mutable state’s Austin Powers.”[1]
- If two different threads can change the same data at the same time, its difficult to guarantee that the execution will leave the data in a valid state

- In contrast, functional programming favours:
  - immutable values
  - functions that always return a value, and if given the same inputs, will always return the same value
  - recursion and “flow syntax”

1. Bruce Tate, Seven Languages in Seven Weeks

Basics

```
val a = "Can't touch this"
a = "We'll see"

var b = "Change me"
b = "OK"

def square(x:Int) = x * x

def sumOfSquares(x:Int, y:Int): Int = {
      val x2 = x * x
      val y2 = y * y
      x2 + y2
}

val numbers = List(1, 2, 3, 4)
numbers(2)
val uniqueNumbers = Set(1, 1, 2)
val hostPort = ("localhost", 80)
hostPort._1
hostPort._2
val fooMap = Map("foo" -> "bar", "fi" -> "baz")
val v = fooMap.get("foo")
v
v.get
fooMap.get("binkle")
fooMap.getOrElse("binkle", "boo")

val range = 0 until 10
```

Type Inference

Scala

```scala
val b = 1 < 2 // x is a Boolean

val word = "Hello" // word is a String
```

Java

```java
Boolean b = 1 < 2

String word = "Hello"
```

Functional Style

Scala

```scala
val numbers = List(1, 2, 3, 4)

numbers.map((x:Int) => x * 2)
numbers.map(_ * 2)

val result = numbers.map(_ * 2)
result.foreach((x: Int) => {
	println(x)
})
result.foreach(println)

numbers.map(_ * 2).filter(_ < 5)

result.take(1)

numbers.reduce((x: Int, y:Int) => x + y)
numbers.reduce(_ + _)
def plus(x:Int, y:Int) = x + y
numbers.reduce(plus)

numbers.foldLeft(0) { (m: Int, n: Int) => {
	println("m: " + m + " n: " + n)
	m + n
}

val nestedNumbers = List(List(1, 2), List(3, 4))
nestedNumbers.map(xs => xs.map(_ * 2))
nestedNumbers.flatMap(xs => xs.map(_ * 2))
```

Java

```java
int[] numbers = new int[] { 1, 2, 3, 4 };
//or
List<Integer> numbers = new ArrayList<Integer>() {{
	add(1);
	add(2);
	add(3);
	add(4);
}};

// map equivalent
int[] mapResult = new int[numbers.length];
for (int i = 0; i < numbers.length; i++) {
	mapResult [i] = numbers[i] * 2;
}

// reduce equivalent
int total = 0;
for (int i : numbers) {
	total += i;
}
```

Case Classes

Scala

```scala
case class Person(firstName: String, lastName: String)
```

```scala
val fred = Person("fred", "wilson")

println(fred.lastName)
> Wilson
```

And supports pattern matching!

```scala
def greetFred(randomPerson: Any) = {
   randomPerson match {
      case Person("fred", _) => println("G'day Fred")
      case _                 => println("Hello")
   }
}

greetFred(fred)
> G'day Fred
```

Java

```java
public class Person implements java.io.Serializable {

  private String firstName;
  private String lastName;

  public String getFirstName() {

    return firstName;
  }

  public void setFirstName(String firstName) {
    this.firstName = firstName;
  }

  public String getLastName() {
    return lastName;
  }

  public void setLastName(String lastName) {
    this.lastName = lastName;
  }

  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    Person person = (Person) o;

    if (firstName != null ? !firstName.equals(person.firstName) : person.firstName != null) return false;
    if (lastName != null ? !lastName.equals(person.lastName) : person.lastName != null) return false;

    return true;
  }

  @Override
  public int hashCode() {
    int result = firstName != null ? firstName.hashCode() : 0;
    result = 31 * result + (lastName != null ? lastName.hashCode() : 0);
    return result;
  }

  @Override
  protected Object clone() throws CloneNotSupportedException {
    return super.clone();
  }

  @Override
  public String toString() {
    return "Person{" +
        "firstName='" + firstName + "\'" +
        ", lastName='" + lastName + "\'" +
        "}";
  }
}
```

Pattern Matching

This example reads n-gram counts from a file:

```
9 WORDTAG O Test
11 WORDTAG O cysts
43 WORDTAG O splice
6 WORDTAG O extensively
1 WORDTAG I-GENE heterodimer

13796 2-GRAM * *
749 3-GRAM * * I-GENE
11320 3-GRAM I-GENE O O
9622 3-GRAM I-GENE I-GENE O
```

```
def extractCounts() {
  var counts: Source = null
  try {
    counts = Source.fromFile("/data/h1-p/gene.counts")
    counts.getLines().foreach(_ split "\\s+" match {
      case Array(k, "WORDTAG", tag, word)       => { wordTagCounts((tag, word)) = k.toInt; }
      case Array(k, "1-GRAM", tag)              => { unigramCounts(tag) = k.toInt }
      case Array(k, "2-GRAM", tag1, tag2)       => { bigramCounts((tag1, tag2)) = k.toInt }
      case Array(k, "3-GRAM", tag1, tag2, tag3) => { trigramCounts((tag1, tag2, tag3)) = k.toInt }
    })
  } finally {
    if (counts != null) counts.close()
  }
}
```

Extras if time

Working with non-tabular data

Scala

XML

```
val movies =
   <movies>
      <movie genre="action">Pirates of the Carribean</movie>
      <movie genre="fairytale">Edward Scissorhands</movie>
   </movies>

movies \ "movie"
(movies \ "movie")(0) \ "@genre"
```

JSON

```
import play.api.libs.json._

case class Person(name: String, age: Int)

val json: JsValue = Json.parse("""
{
   "name": "Billy Bob",
   "age": 42
}
""")

val name = json \ "name"

implicit val personReads: Reads[Person] = (
   (JsPath \ "name").read[String] and
   (JsPath \ "age").read[Int]
)(Person.apply _)

val personResult: JsResult[Person] = json.validate[Person]
val person = personResult.get
```

Isn't creating new values inefficient?

- The immutable data structures in Scala are designed to be efficient. For example:

  Consider an immutable singly-linked linked list:

  L = d → c → b → a

  ↑

  L can be stored with just a reference to the head of the list, the element d

  Suppose we want to create a new list L2 with element **e** added to the head of the list

  L2 = e → (pointer to d)

  However, suppose we want to create a new list L3 with the elements d → c → **j** → b → a

  We can’t mutate L since that would also change L2, therefore we need to copy elements d, c

  L3 = d → c → j → (pointer to b)

  Similar to how Git manages branches, commits up to a common ancestor are not copied

What is a Map Operation?

Using JavaScript as an example:

```
function square(a) {
    for (i = 0; i < a.length; i++) {
        a[i] = a[i]^2;
    }
}
```

Generalized Map Operation

```
function map(fn, a) {
        for (i = 0; i < a.length; i++) {
                a[i] = fn(a[i]);
        }
}
```

Invoked as:

```
map(function (x) { return x^2; }, a);
```

What is a Reduce Operation?

```
function sum(a) {
	var s = 0;
	for (i = 0; i < a.length; i++) {
		s += a[i];
	}
	return s;
}
```

Generalized Reduce Operation

```
function reduce(fn, a, init) {
	var s = init;
	for (i = 0; i < a.length; i++) {
		s = fn(s, a[i]);
	}
	return s;
}
```

Invoked as:

```
reduce(function (a, b) { return a + b; }, a, 0);
```

Machine Learning Application of Map-Reduce

- Batch gradient descent: $$\theta j := \theta j - \alpha \frac{1}{400} \boxed{\sum_{i=1}^{400}} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)})$$

Map

- Machine 1: $(x^{(1)}, y^{(1)}), \ldots, (x^{(100)}, y^{(100)})$
  $$temp_j^{(1)} = \sum_{i=1}^{100} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$
- Machine 2: $(x^{(101)}, y^{(101)}), \ldots, (x^{(200)}, y^{(200)})$
  $$temp_j^{(2)} = \sum_{i=101}^{200} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$
- Machine 3: $(x^{(201)}, y^{(201)}), \ldots, (x^{(300)}, y^{(300)})$
  $$temp_j^{(3)} = \sum_{i=201}^{300} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$
- Machine 4: $(x^{(301)}, y^{(301)}), \ldots, (x^{(400)}, y^{(400)})$
  $$temp_j^{(4)} = \sum_{i=301}^{400} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$$

Reduce

$$\theta_j := \theta_j - \alpha \frac{1}{400} (temp_j^{(1)} + temp_j^{(2)} + temp_j^{(3)} + temp_j^{(4)})$$