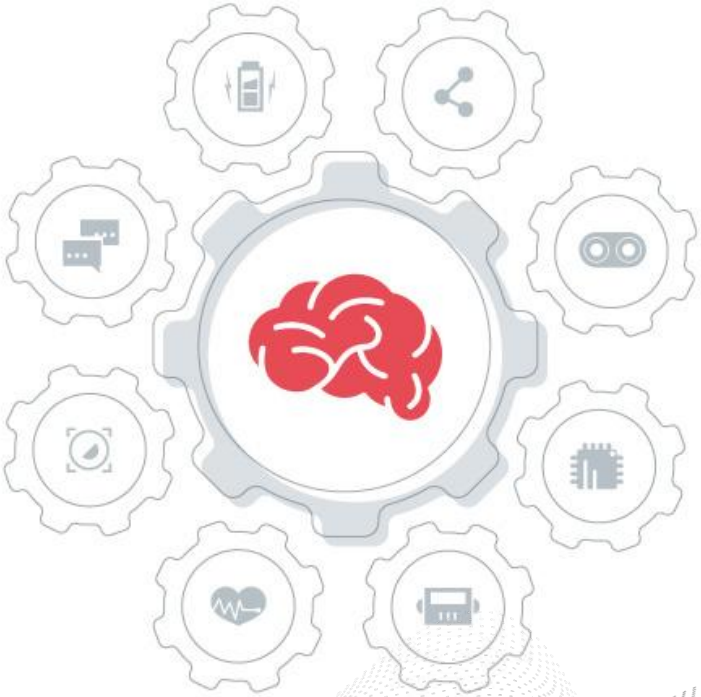
Accionlabs

An Introduction to Apache Spark and Scala

Chetan Khatri
chetan.khatri@accionlabs.com



WHO AM I ?

Lead - Data Science, Technology Evangelist @ Accion labs India Pvt. Ltd.
Committer @ Apache Spark, Apache HBase, Elixir Lang.
Co-Authored University Curriculum @ University of Kachchh, India.

Data Engineering @: Nazara Games, Eccella Corporation.
M.Sc. - Computer Science from University of Kachchh, India.

Past and Future Conference Talks!

FOSSASIA Singapore - March, 2018
HKOSCon Hong Kong - June, 2018
ZuriHac Switzerland - June, 2018
HBaseConAsia Beijing, China - Aug, 2018

Upcoming:
Scala.IO - Lyon, France - Oct, 2018

Agenda

- Apache Spark and Scala
- Resilient Distributed Datasets (RDDs)
- DataFrames and Datasets
- Spark Operations
- Data Platform Components
- Re-engineering Data processing platform
- Rethink - Fast Data Architecture
- Parallelism & Concurrency at Spark

What is Apache Spark ?

- Apache Spark is a fast and general-purpose cluster computing system / Unified Engine for massive data processing.
- It provides high level API for Scala, Java, Python and R and optimized engine that supports general execution graphs.

Structured Data / SQL - Spark SQL	Graph Processing - GraphX
Machine Learning - MLlib	Streaming - Spark Streaming, Structured Streaming

What is Scala ?

- Scala is a modern multi-paradigm programming language designed to express common programming patterns in a concise, elegant, and type-safe way.

- Scala is object-oriented
- Scala is functional
- Strongly typed, Type Inference
- Higher Order Functions
- Lazy Computation

Data Structures in Apache Spark ?

- RDD
- DataFrame
- DataSet

What are RDDs ?

1. Distributed Data Abstraction

Logical Model Across Distributed Storage on Cluster

HDFS, S3

2. Resilient & Immutable

RDD -> T -> RDD -> T -> RDD

T = Transformation

3. Compile-time Type Safe / Strongly type inference

Integer RDD

String or Text RDD

Double or Binary RDD

4. Lazy evaluation

RDD - T - RDD - T - RDD - T - RDD - A - RDD

T = Transformation
A = Action

Apache Spark Operations

Operations

Transformation

Action

Essential Spark Operations

	General	Math / Statistical	Set Theory / Relational	Data Structure / I/O
TRANSFORMATIONS	map gilter flatMap mapPartitions mapPartitionsWithIndex groupBy sortBy	sample randomSplit	union intersection subtract distinct cartesian zip	keyBy zipWithIndex zipWithUniqueID zipPartitions coalesce repartition repartitionAndSortWithinPartitions pipe
ACTIONS	reduce collect aggregate fold first take forEach top treeAggregate treeReduce forEachPartition collectAsMap	count takeSample max min sum histogram mean variance stdev sampleVariance countApprox countApproxDistinct	takeOrdered	saveAsTextFile saveAsSequenceFile saveAsObjectFile saveAsHadoopDataset saveAsHadoopFile saveAsNewAPIHadoopDataset saveAsNewAPIHadoopFile

When to use RDDs ?

- You care about control of dataset and knows how data looks like, you care about low level API.
- Don’t care about lot’s of lambda functions than DSL.
- Don’t care about Schema or Structure of Data.
- Don’t care about optimization, performance & inefficiencies!
- Very slow for non-JVM languages like Python, R.
- Don’t care about Inadvertent inefficiencies.

Word Count!

```
import org.apache.spark.sql.SparkSession
object WordCount {
 def main(args: Array[String]) {
   val inputFile = args(0)
   val outputFile = args(1)
   val spark = SparkSession.builder().appName("Hello World").getOrCreate()
   // Create a Scala Spark Context.
   val sc = spark.sparkContext
   // Load our input data.
   val input =  sc.textFile(inputFile)
   // Split up into words.
   val words = input.flatMap(line => line.split(" "))
   // Transform into word and count.
   val counts = words.map(word => (word, 1)).reduceByKey{case (x, y) => x + y}
   // Save the word count back out to a text file, causing evaluation.
   counts.saveAsTextFile(outputFile)
 }
}
```

Inadvertent inefficiencies in RDDs

```
parsedRDD.filter { case (project, sprint, numStories) => project == "finance" }.
 map { case (_, sprint, numStories) => (sprint, numStories) }.
 reduceByKey(_ + _).
 filter { case (sprint, _) => !isSpecialSprint(sprint) }.
 take(100).foreach { case (project, stories) => println(s"project: $stories") }
```

Structured in Spark

DataFrames

Datasets

Why Dataset ?

- Strongly Typing
- Ability to use powerful lambda functions.
- Spark SQL's optimized execution engine (catalyst, tungsten)
- Can be constructed from JVM objects & manipulated using Functional transformations (map, filter, flatMap etc)
- A DataFrame is a Dataset organized into named columns

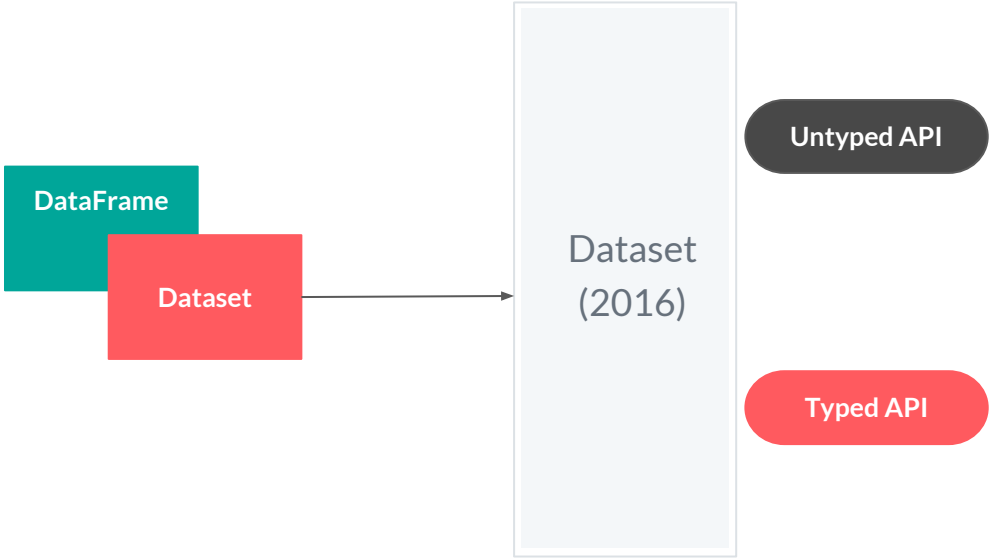
  DataFrame is simply a type alias of Dataset[Row]

Structured APIs in Apache Spark

	SQL	DataFrames	Datasets
Syntax Errors	Runtime	Compile Time	Compile Time
Analysis Errors	Runtime	Runtime	Compile Time

Analysis errors are caught before a job runs on cluster

Unification of APIs in Apache Spark 2.0



- DataFrame = Dataset [Row]
  Alias
- Dataset [T]

DataFrame API Code

```
//  convert   RDD -> DF with column names
val parsedDF = parsedRDD.toDF("project", "sprint", "numStories")
//filter,   groupBy,  sum, and  then agg()
parsedDF.filter($"project" === "finance").
 groupBy($"sprint").
 agg(sum($"numStories").as("count")).
 limit(100).
 show(100)
```

project	sprint	numStories
finance	3	20
finance	4	22

DataFrame -> SQL View -> SQL Query

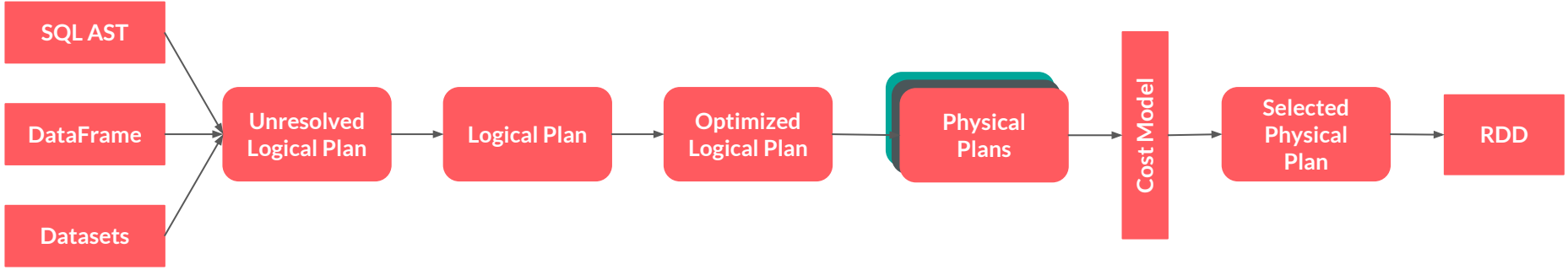
```
parsedDF.createOrReplaceTempView("audits")
val results = spark.sql(
 """SELECT  sprint,  sum(numStories)
 AS  count  FROM audits  WHERE  project  =  'finance' GROUP  BY sprint
 LIMIT 100""")
results.show(100)
```

project	sprint	numStories
finance	3	20
finance	4	22

Why Structure APIs ?

```
// DataFrame
data.groupBy("dept").avg("age")
// SQL
select dept, avg(age) from data group by 1
// RDD
data.map { case (dept, age) => dept -> (age, 1) }
 .reduceByKey { case ((a1, c1), (a2, c2)) => (a1 + a2, c1 + c2) }
 .map { case (dept, (age, c)) => dept -> age / c }
```

Catalyst in Spark



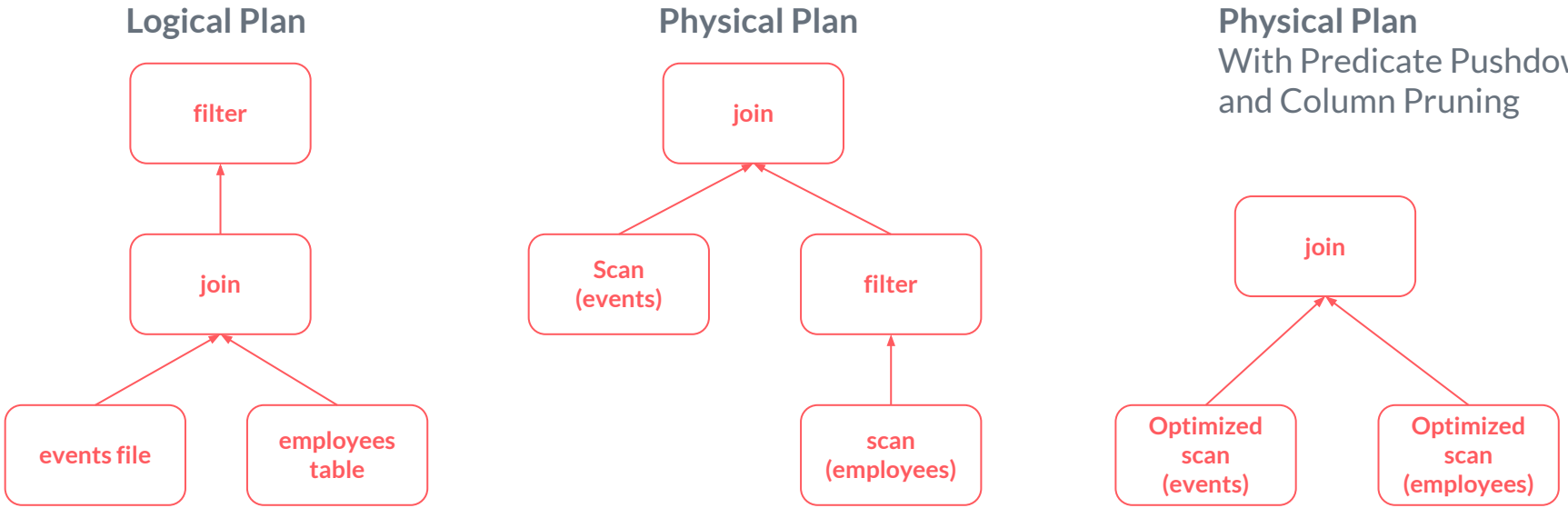
Dataset API in Spark 2.x

```scala
val employeesDF = spark.read.json("employees.json")

// Convert data to domain objects.
case class Employee(name: String, age: Int)

val employeesDS: Dataset[Employee] = employeesDF.as[Employee]
val filterDS = employeesDS.filter(p => p.age > 3)
```

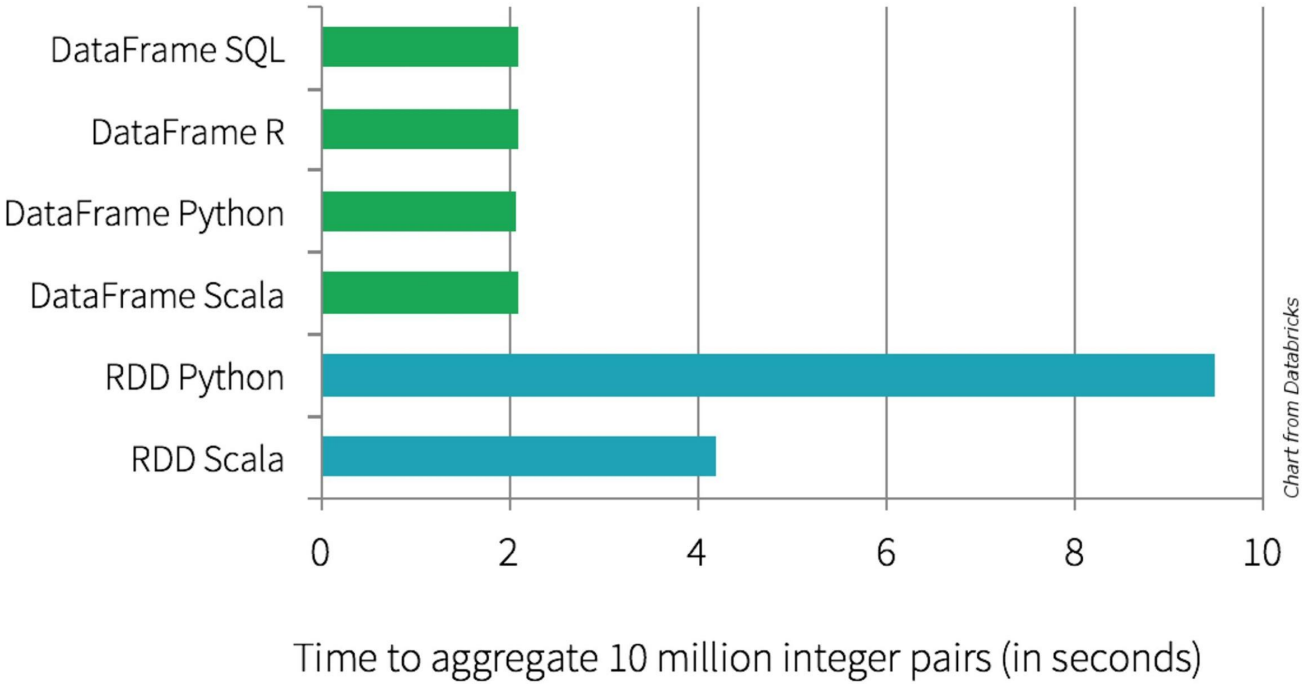
Type-safe: operate on domain objects with compiled lambda functions.

Example: DataFrame Optimization

```
employees.join(events, employees("id") === events("eid"))
         .filter(events("date") > "2015-01-01")
```

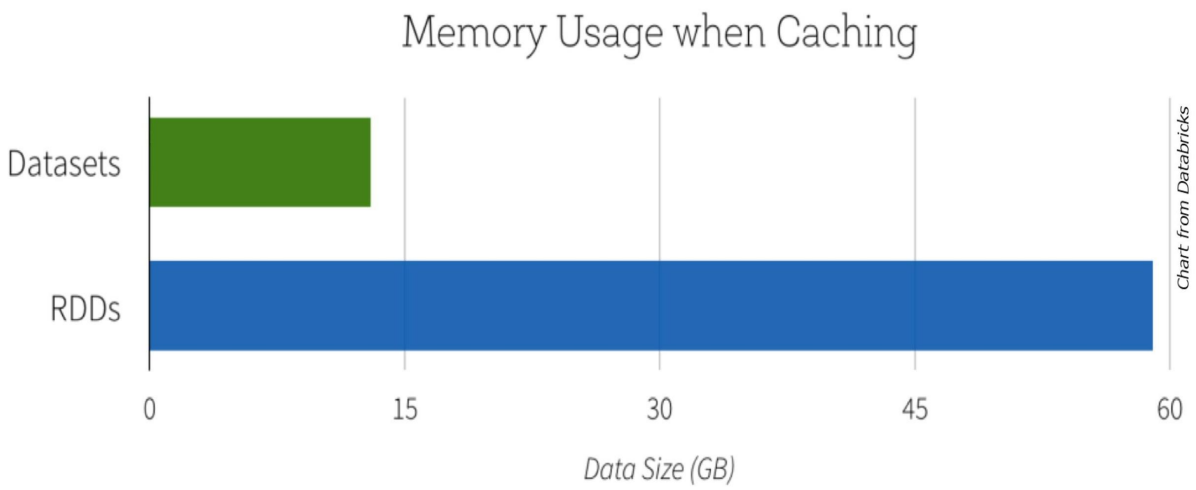


DataFrames are Faster than RDDs



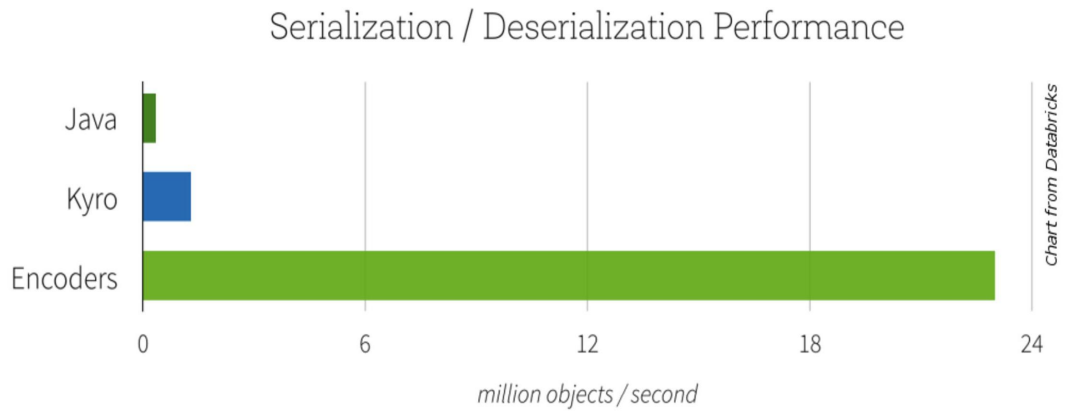
Source: Databricks

Datasets takes less Memory than RDDs



Source: Databricks

Datasets are faster



Enabler ?

Accionlabs

Open Source

Questions ?

Thank you for your Time!