Kathmandu Apache Spark Meetup

Jump Start with Apache® Spark™ 2.x on Databricks

https://www.meetup.com/kathmandu-apache-spark

https://github.com/ganeshchand/kathmandu-apache-spark-meetup-introduction-to-spark-2.0

Ganesh Chand
Kathmandu Apache Spark Meetup, Kathmandu, Nepal 12/23/2016

databricks™

$ whoami

- Data & Analytics engineer @ Databricks
- Previously Developer @ Guidewire Software
- Past engineering roles at:
  - Accenture, SunGard, Hitachi Software
- ganesh@databricks.com
- https://www.linkedin.com/in/chandganesh
- @geechand

Agenda for the next 2.5 hours

Hour 1.5

- About KASM
- Get to know Databricks
- Overview of Spark Fundamentals & Architecture
- What's New in Spark 2.0
- Unified APIs: SparkSessions, SQL, DataFrames, Datasets…
- Workshop Notebook 1
- Break

Hour 1

- Introduction to DataFrames, DataSets and Spark SQL
- Workshop Notebook 2
- Social hours

Kathmandu Apache Spark Meetup

A community of data practitioners and enthusiasts

Area of Interests:

- Functional programming (**scala**)
- Data Engineering (**Spark, Scala**)
- Data Science / Machine Learning (**Spark, Python, R**)
- Data Visualization (**D3.js**)
- Distributed Data Store (**Hadoop, S3, Redshift, Cassandra**)
- Cloud Platform (**Databricks, EMR**)
- Emerging tools & technologies (**Tenserflow, Apache Flink**)

https://www.meetup.com/kathmandu-apache-spark

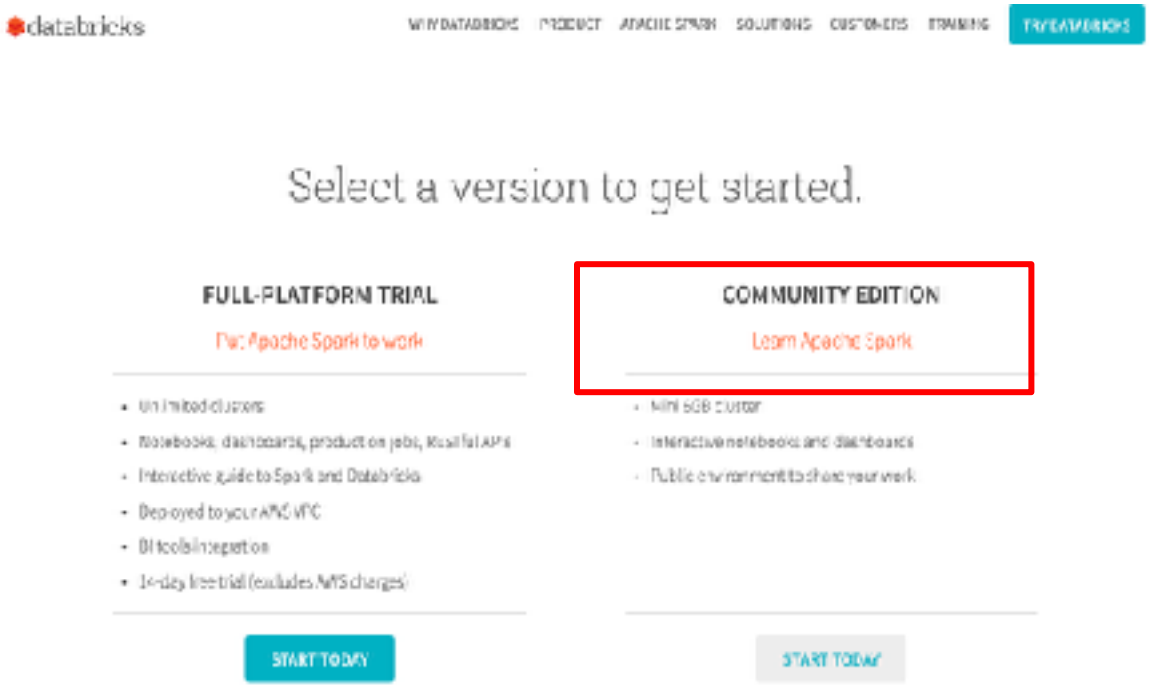
Kathmandu Apache Spark Meetup

2015

- Introduction to Big Data and Apache Spark
- Live coding
- 55+ attendees
- Sponsored by RoyalePi
- Guest Speaker from France - Ludwine Probst

Get to know Databricks

- Get Databricks community edition http://databricks.com/try-databricks

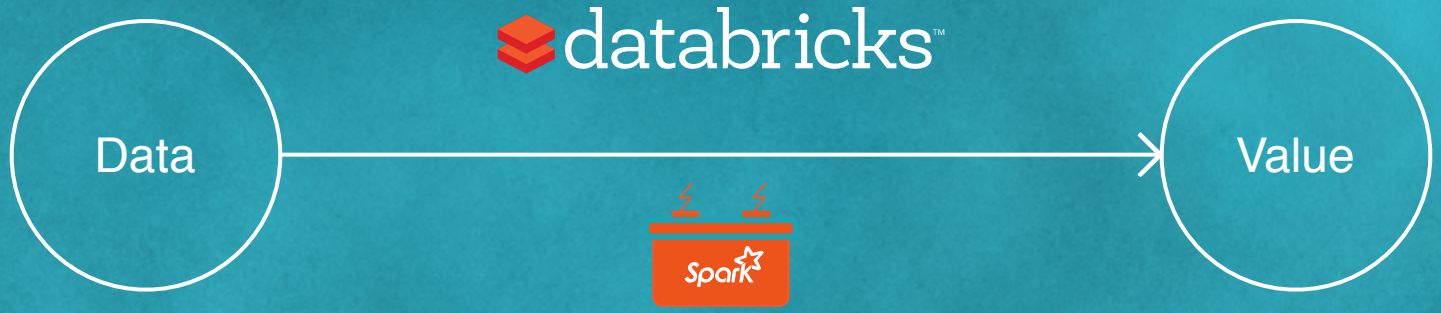


We are Databricks, the company behind Apache Spark

Founded by the creators of Apache Spark in 2013

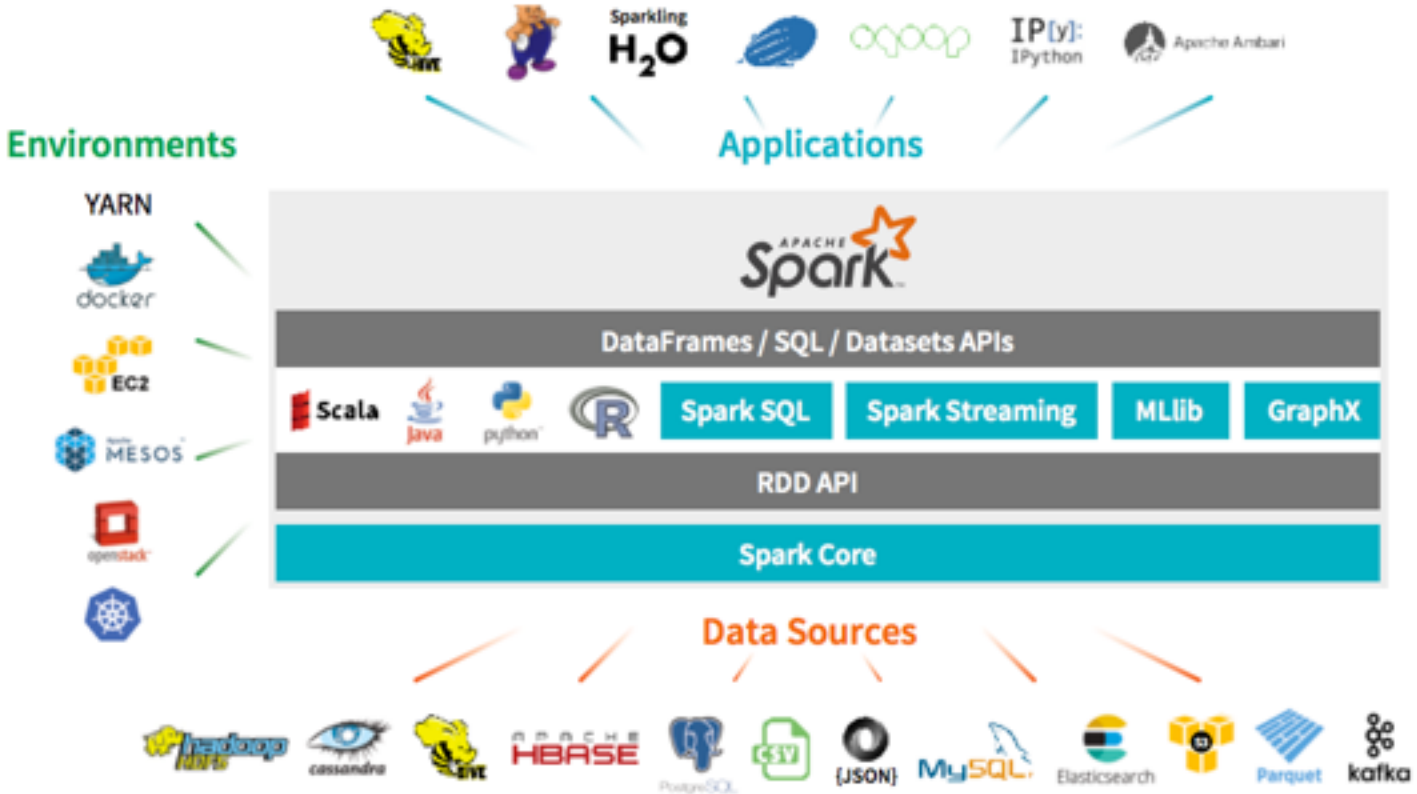
75%

Share of Spark code contributed by Databricks in 2014

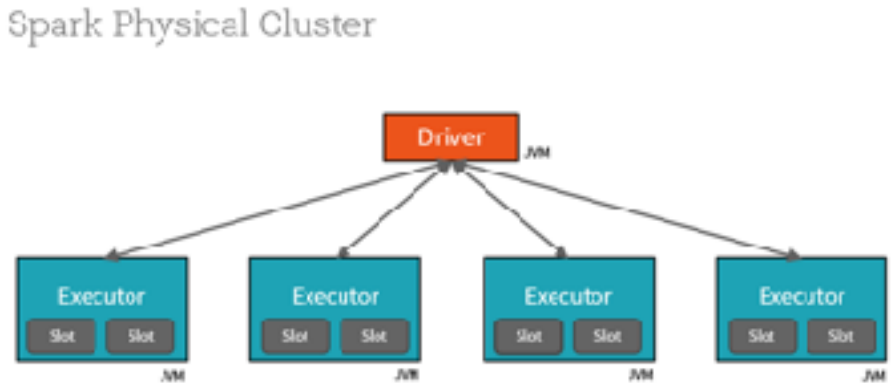


Created Databricks on top of Spark to make big data simple.

Unified engine across diverse workloads & environments



Apache Spark Architecture

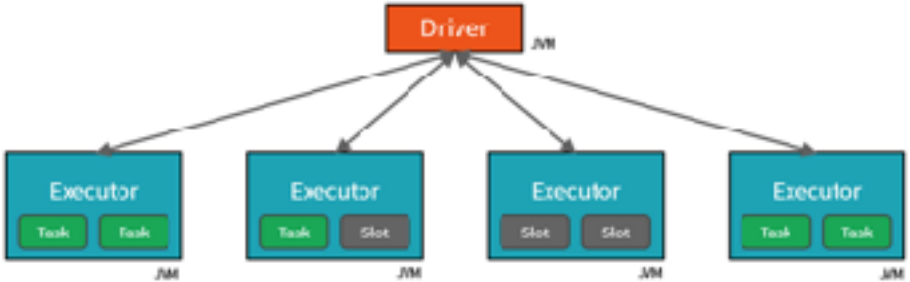


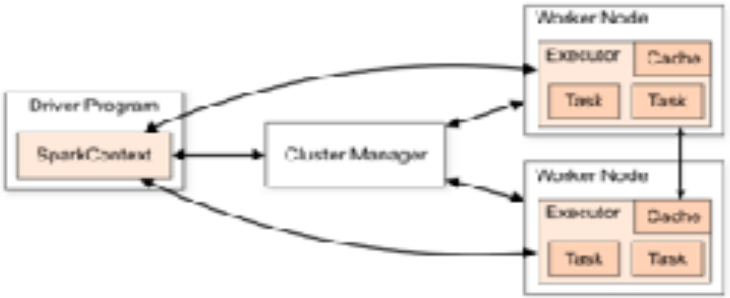
Deployments Modes

- Local
- Standalone
- YARN
- Mesos

Apache Spark Architecture

An Anatomy of an Application





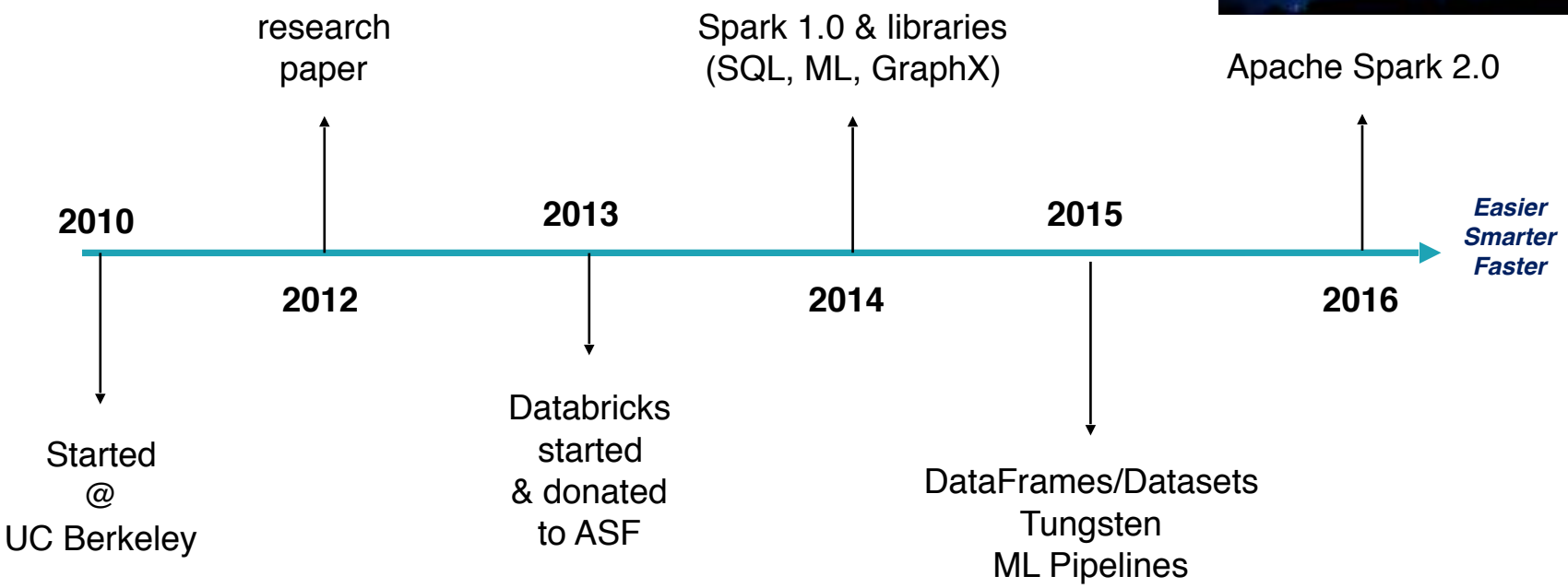
Spark Application

- Jobs
- Stages
- Tasks

databricks

How did we Get Here..?
Where we Going..?

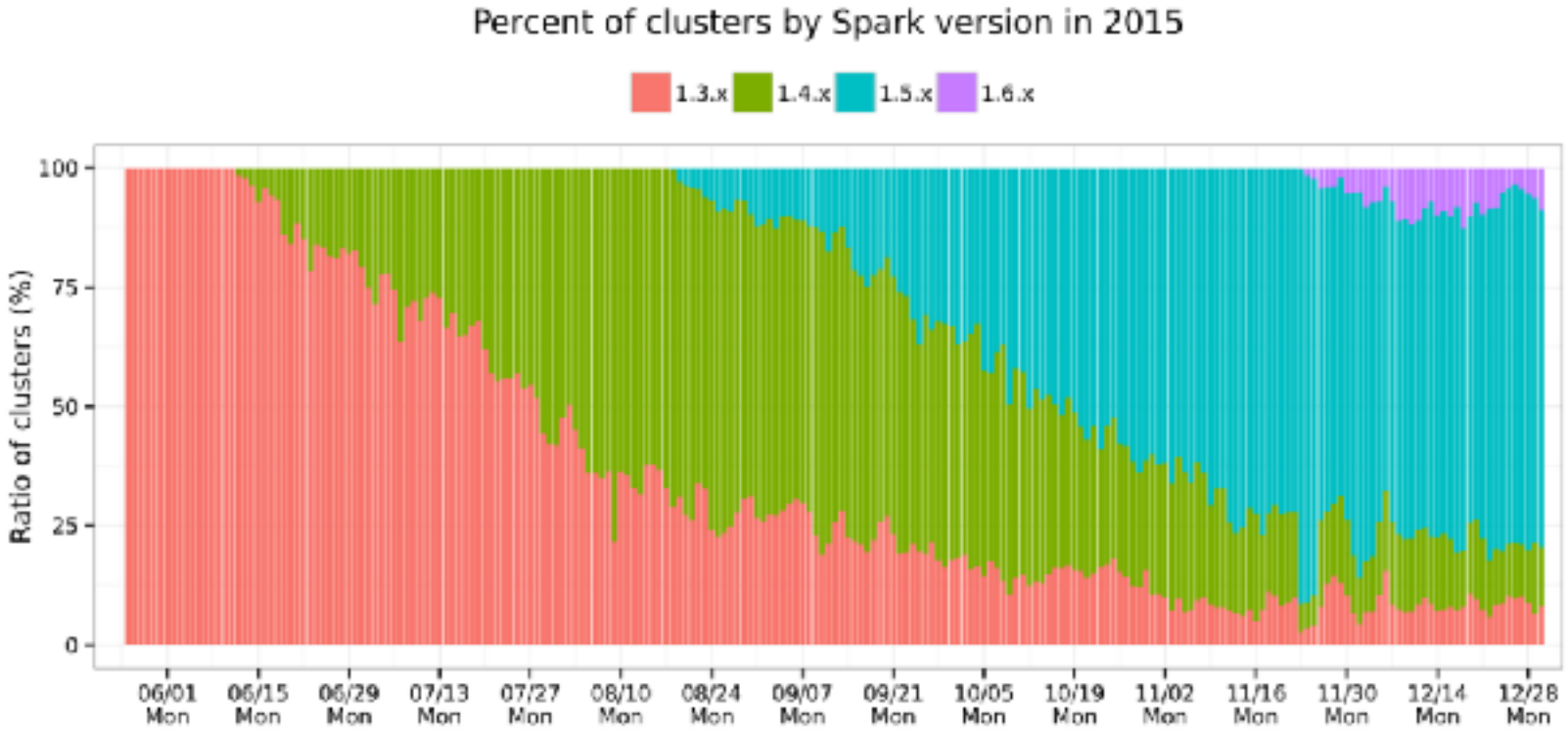
A Brief History



Apache Spark 2.0

- Steps to Bigger & Better Things….

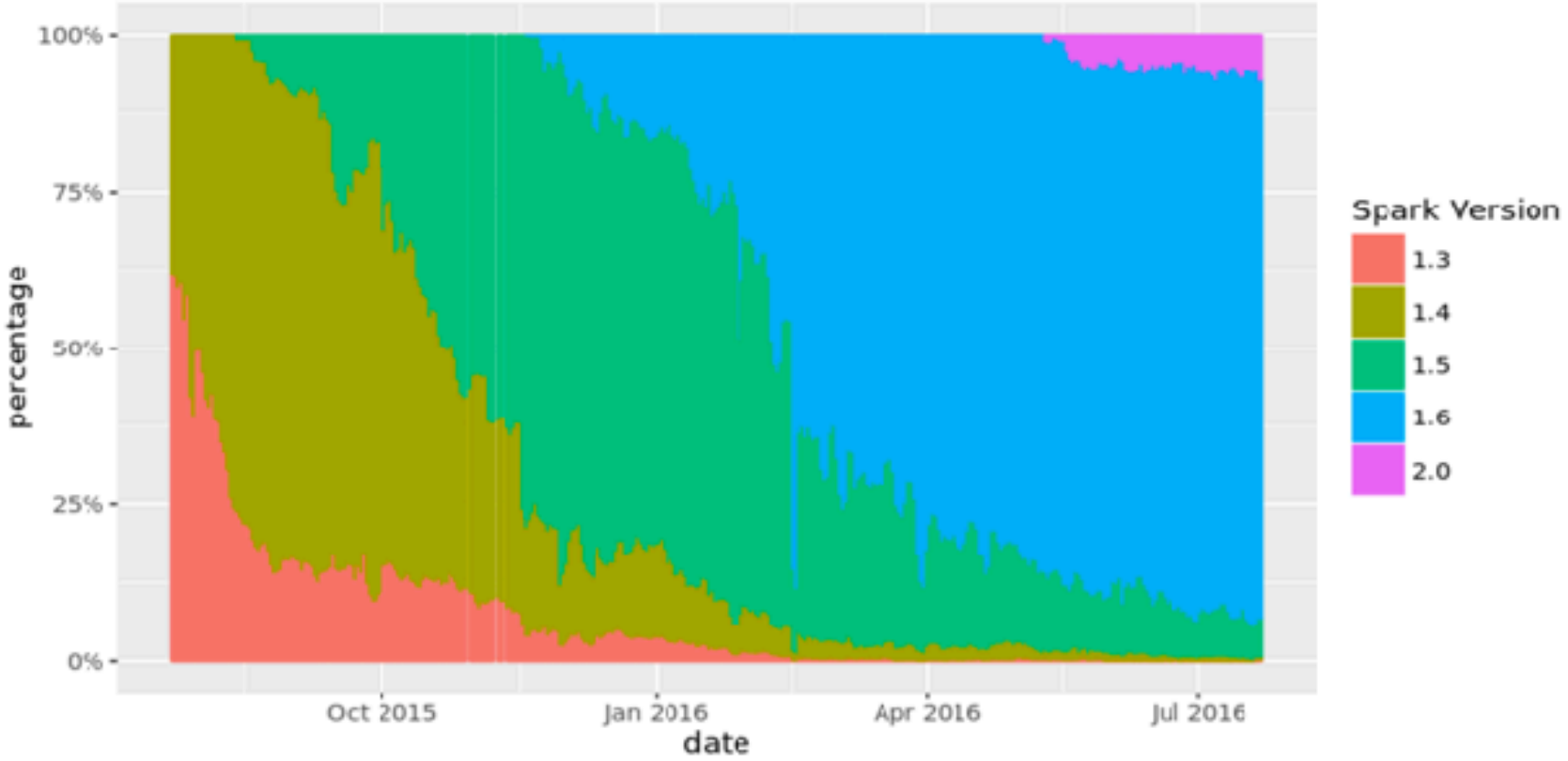
Builds on all we learned in past 2 years



Reynold Xin @rxin · Jan 14

The most interesting thing in Spark 2015 Year in Review post is the version dist over time. databricks.com/blog/2016/01/0...

9 19



Apache Spark Usage over Time by Version

Major Features in Apache Spark 2.0

Unifying Datasets and DataFrames & SparkSessions

Easier

Tungsten Phase 2 speedups of 5-10x & Catalyst Optimizer

Faster

Structured Streaming real-time engine on SQL / DataFrames

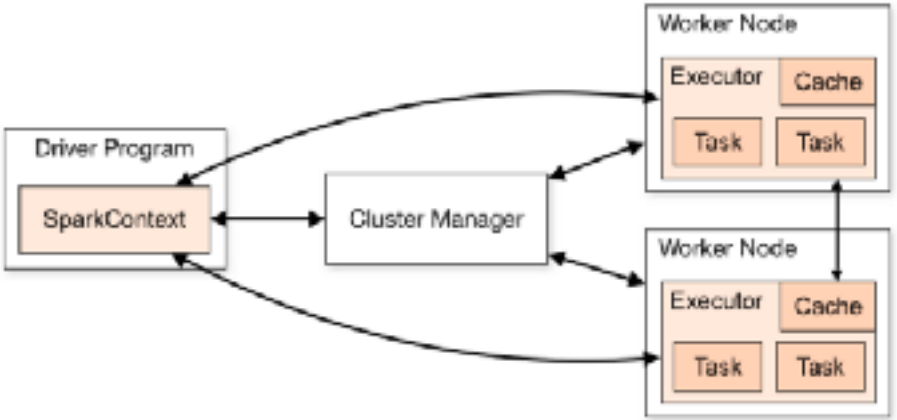
Smarter

Unified API Foundation for the Future: Spark Sessions, Dataset, DataFrame, MLlib, Structured Streaming…

SparkSession – A Unified entry point to Spark

- SparkSession is the “SparkContext” for Dataset/ DataFrame
  - Entry point for reading data and writing data
  - Working with metadata
  - Setting Spark Configuration
  - Driver uses for Cluster resource management

SparkSession vs SparkContext



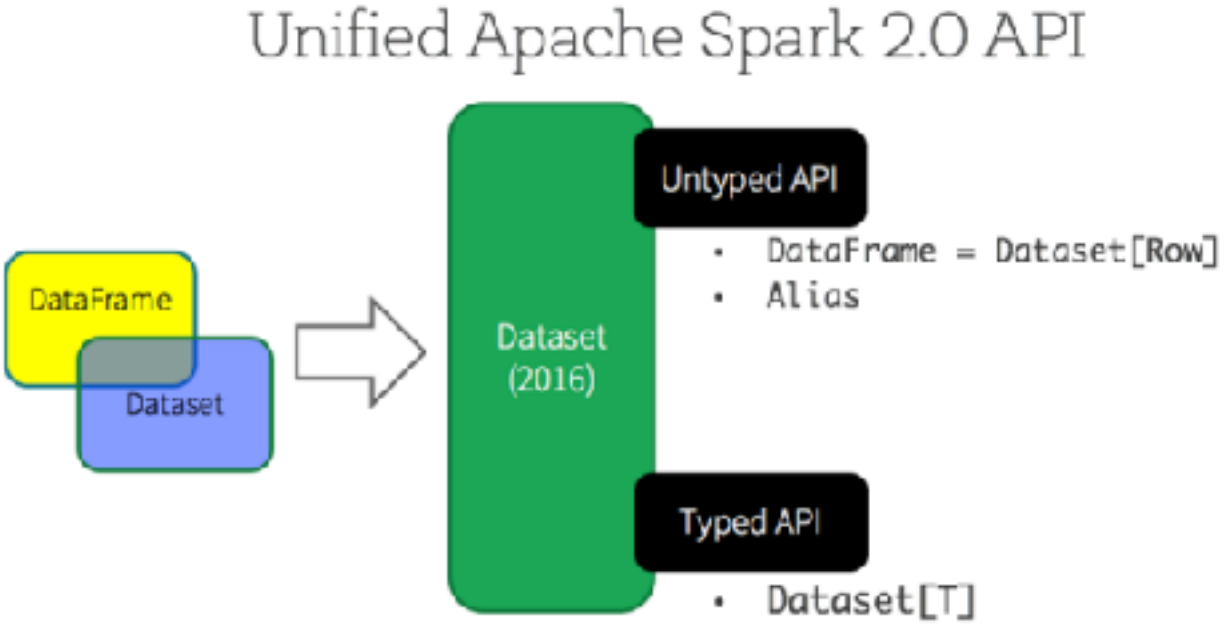
SparkSessions Subsumes

- SparkContext
- SQLContext
- HiveContext
- StreamingContext
- SparkConf

```
val warehouseLocation = "file:${system:user.dir}/spark-warehouse"

val spark = SparkSession
  .builder()
  .appName("SparkSessionZipsExample")
  .config("spark.sql.warehouse.dir", warehouseLocation)
  .enableHiveSupport()
  .getOrCreate()
```

Datasets and DataFrames



Long Term

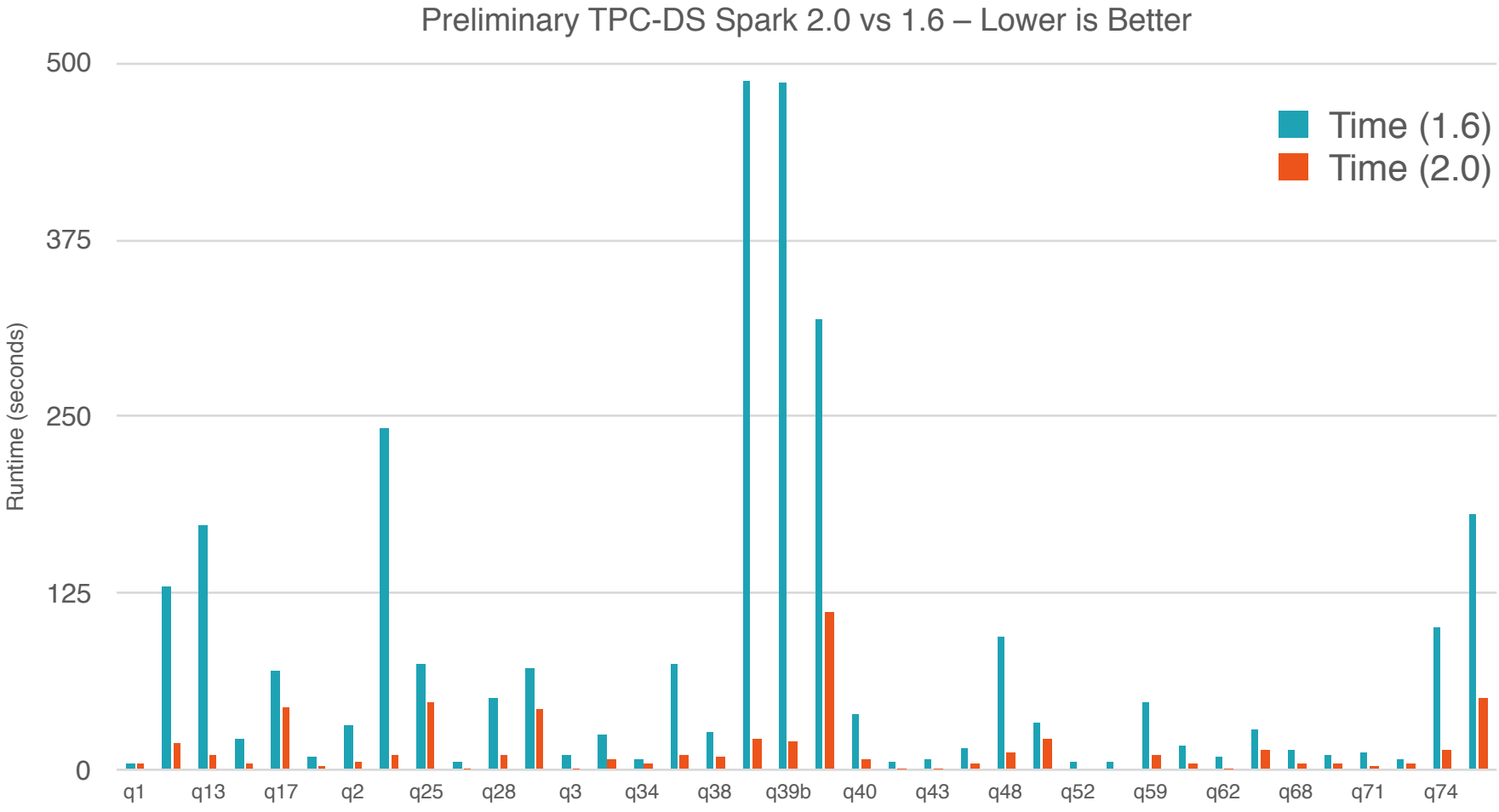
- RDD will remain the low-level API in Spark
  - For control and certain type-safety in Java/Scala
- Datasets & DataFrames give richer semantics and optimizations
  - For semi-structured data and DSL like operations
  - New libraries will increasingly use these as interchange format
  - Examples: Structured Streaming, MLlib, GraphFrames
    - A Tale of Three APIs: RDDs, DataFrames and Datasets

Other notable API improvements

- DataFrame-based ML pipeline API becoming the main MLlib API
- ML model & pipeline persistence with almost complete coverage
  - In all programming languages: Scala, Java, Python, R
- Improved R support
  - (Parallelizable) User-defined functions in R
  - Generalized Linear Models (GLMs), Naïve Bayes, Survival Regression, K-Means

Towards SQL 2003

- Today, Spark can run all 99 TPC-DS queries!
- New standard compliant parser (with good error messages!)
- Subqueries (correlated & uncorrelated)
- Approximate aggregate stats
  - https://databricks.com/blog/2016/07/26/introducing-apache-spark-2-0.html
  - https://databricks.com/blog/2016/06/17/sql-subqueries-in-apache-spark-2-0.html


Preliminary TPC-DS Spark 2.0 vs 1.6 – Lower is Better
Runtime (seconds)
500
375
250
125
0
Time (1.6)
Time (2.0)
q1
q13
q17
q2
q25
q28
q3
q34
q38
q39b
q40
q43
q48
q52
q59
q62
q68
q71
q74

How exactly do I learn and build Spark Apps?

- **Spark REPL** - ideal for learning and quick prototyping
- IDE Tool (**IntelliJ**) - ideal for library and end-to-end app development
- Interactive tools - ideal for exploring datasets and data storytelling.
  - **Databricks Notebook**
  - Zeppelin
  - Spark-Notebook
  - Jupyter-Scala

Workshop: Notebook on SparkSession

- Import Notebook into your Spark 2.0 Cluster
  - http://dbricks.co/sswksh1
  - http://docs.databricks.com
  - http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.SparkSession
- Familiarize your self with Databricks Notebook environment
- Work through each cell
  - CNTR + <return> / Shift + Return
- Try challenges
- Break…

DataFrames/Datasets & Spark SQL & Catalyst Optimizer

The not so secret truth…

SQL

is not about SQL

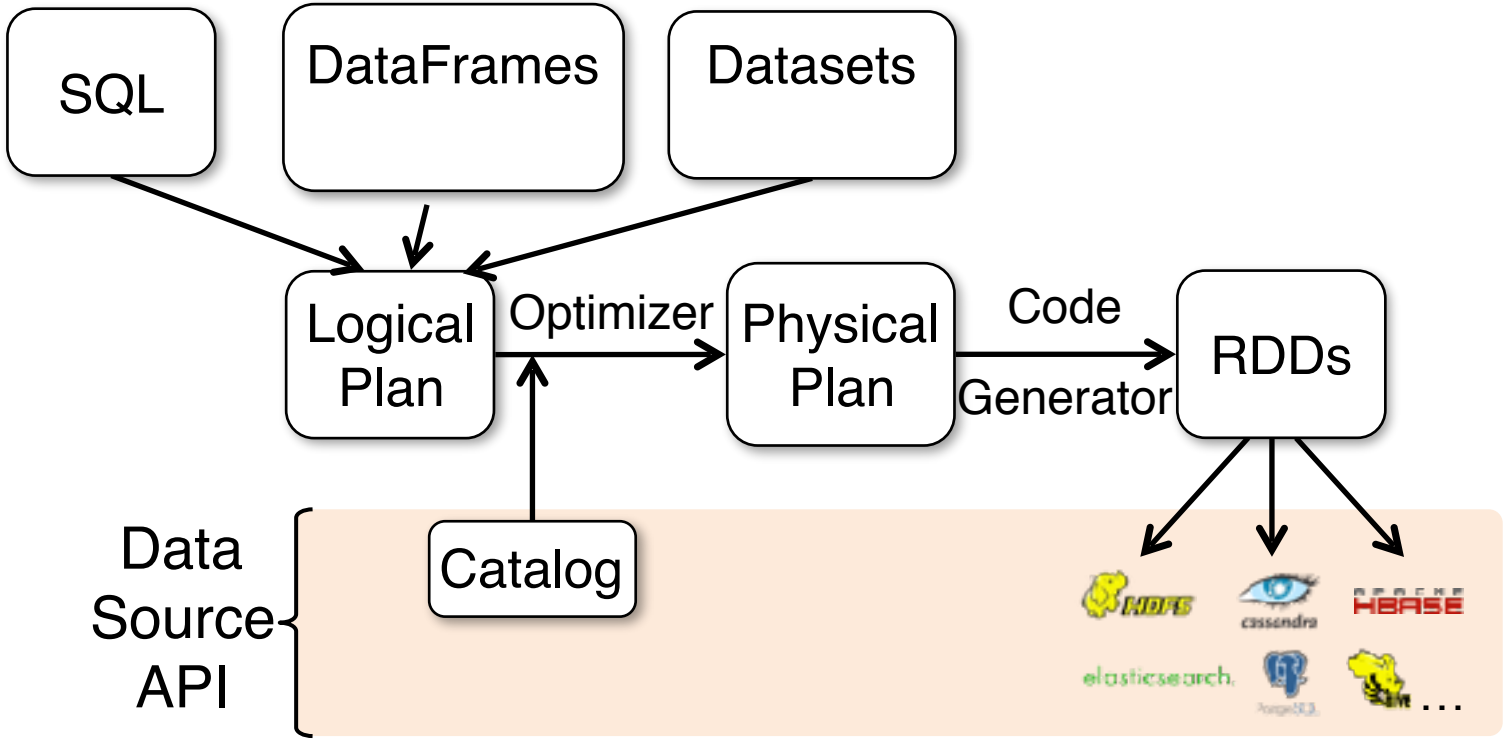
is about more than SQL

Spark SQL: The whole story

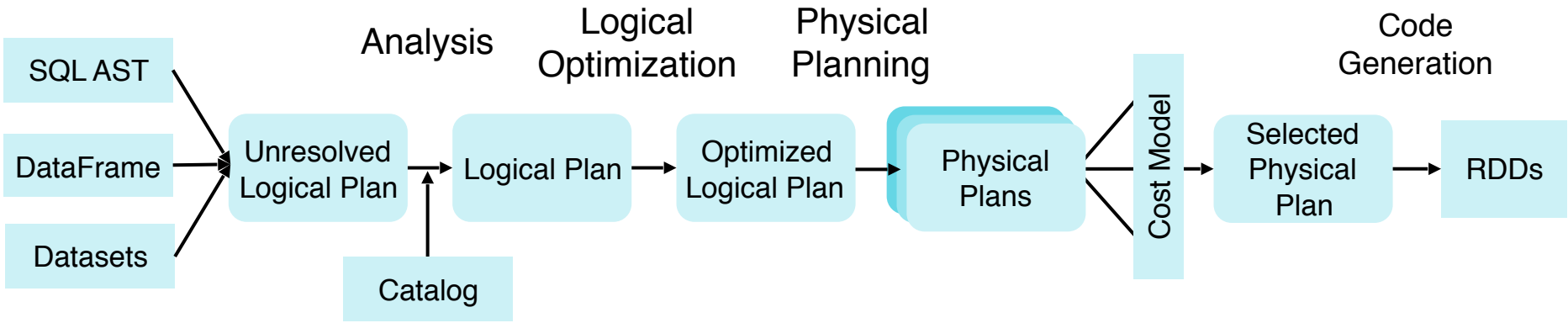
Is About Creating and Running Spark Programs Faster:

- Write less code
- Read less data
- Let the optimizer do the hard work

Spark SQL Architecture



Using Catalyst in Spark SQL



Analysis: analyzing a logical plan to resolve references

Logical Optimization: logical plan optimization

Physical Planning: Physical planning

Code Generation: Compile parts of the query to Java bytecode

Catalyst Optimizations

Logical Optimizations

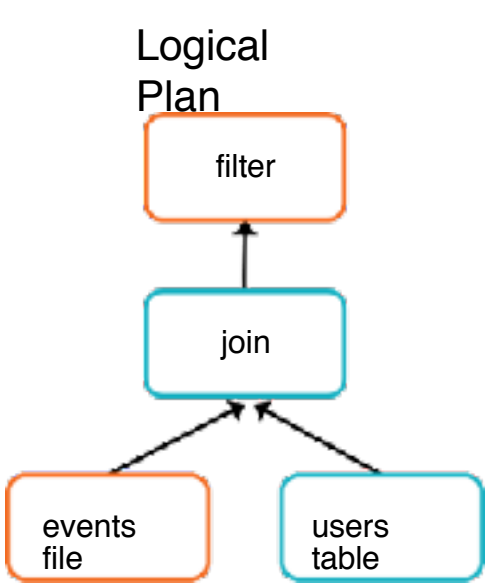
- Push filter predicates down to data source, so irrelevant data can be skipped
- **Parquet:** skip entire blocks, turn comparisons on strings into cheaper integer comparisons via dictionary encoding
- **RDBMS:** reduce amount of data traffic by pushing predicates down

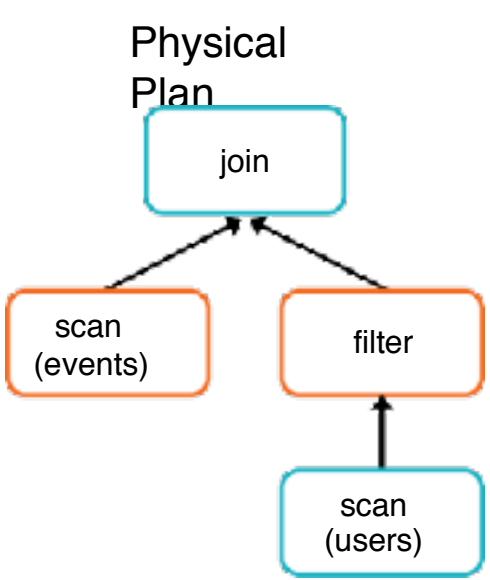
Create Physical Plan & generate JVM bytecode

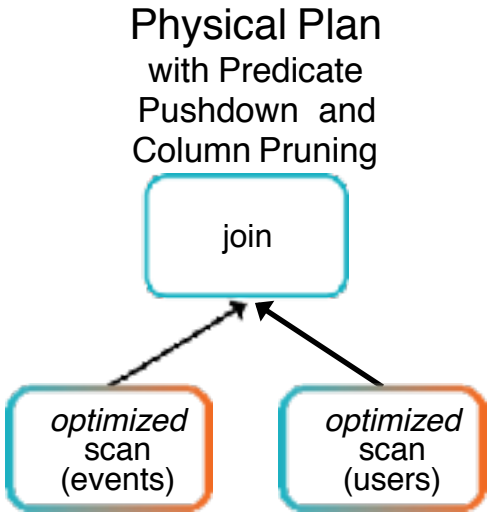
- Catalyst compiles operations into physical plans for execution and generates JVM bytecode
- Intelligently choose between broadcast joins and shuffle joins to reduce network traffic
- **Lower level optimizations:** eliminate expensive object allocations and reduce virtual function calls

```
def add_demographics(events):
    u = sqlCtx.table("users")                    # Load partitioned Hive table
    events \
      .join(u, events.user_id == u.user_id) \    # Join on user_id
      .withColumn("city", zipToCity(df.zip))     # Run udf to add city column

events = add_demographics(sqlCtx.load("/data/events", "parquet"))
training_data = events.where(events.city == "New York").select(events.timestamp).collect()
```







Columns: Predicate pushdown

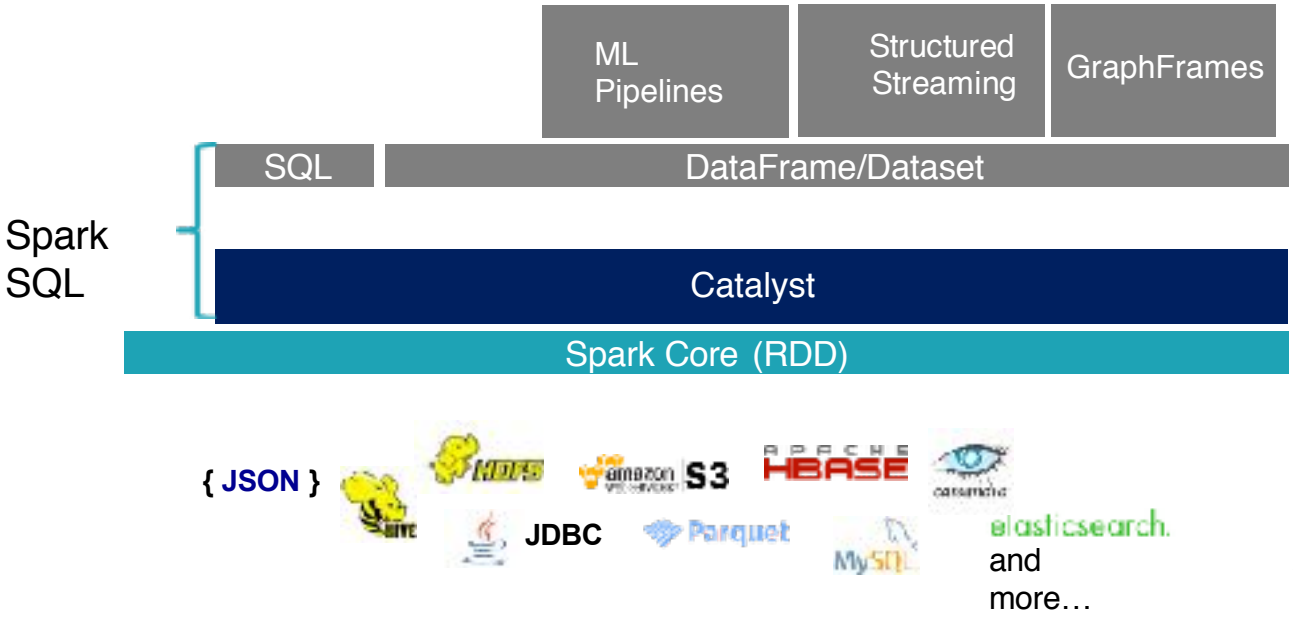
You Write

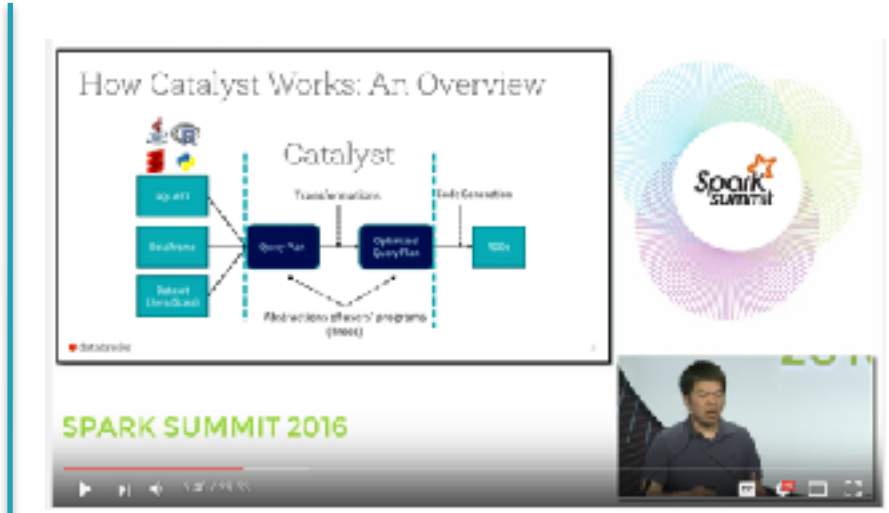
```
spark.read
  .format("jdbc")
  .option("url", "jdbc:postgresql:dbserver")
  .option("dbtable", "people")
  .load()
  .where($"name" === "michael")
```

Spark Translates For Postgres

```
SELECT * FROM people WHERE name = 'michael'
```

Foundational Spark 2.0 Components





http://people.csail.mit.edu/matei/papers/2015/sigmod_spark_sql.pdf

Dataset Spark 2.0 APIs

Background: What is in an RDD?

- Dependencies
- Partitions (with optional locality info)
- Compute function: Partition => Iterator[T]

Opaque Computation & Opaque Data

Structured APIs In Spark

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

Analysis errors are reported before a distributed job starts

Dataset API in Spark 2.0

- Typed interface over DataFrames / Tungsten

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

• val dataframe = spark.read.json(“people.json”)
• val ds: Dataset[Person] = dataframe.as[Person]

• ds.filter(p => p.name.startsWith(“M”))
    .groupBy(“name”)
    .avg(“age”)
```

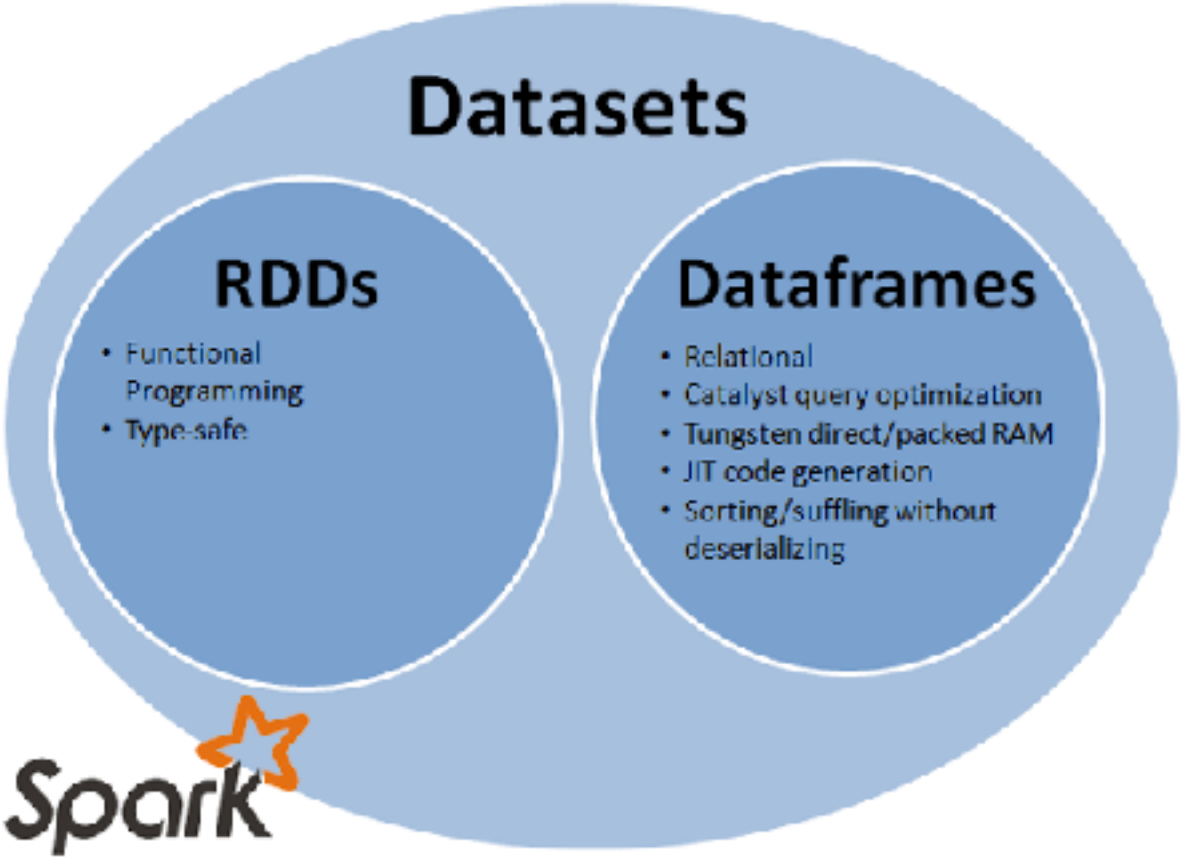
Datasets API

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

```
val df = spark.read.json("people.json")

// Convert data to domain objects.
case class Person(name: String, age: Int)
val ds: Dataset[Person] = df.as[Person]
ds.filter(_.age > 30)

// Compute histogram of age by name.
val hist = ds.groupBy(_.name).mapGroups {
  case (name, people: Iter[Person]) =>
    val buckets = new Array[Int](10)
    people.map(_.age).foreach { a =>
      buckets(a / 10) += 1
    }
    (name, buckets)
}
```



Source: michaelmalak

Project Tungsten II

Project Tungsten

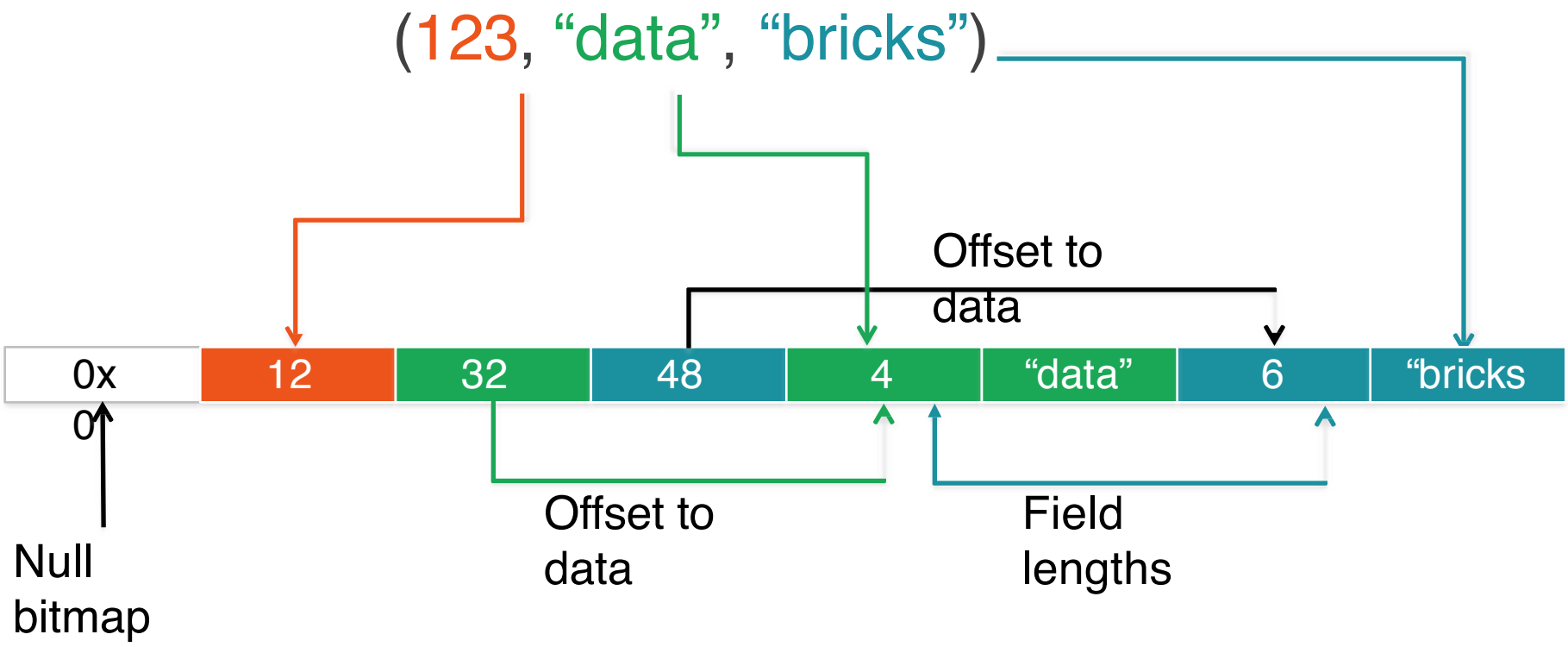
- Substantially speed up execution by optimizing CPU efficiency, via: SPARK-12795

(1) Runtime code generation

(2) Exploiting cache locality

(3) Off-heap memory management

Tungsten's Compact Row Format

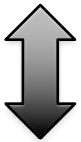


Encoders

Encoders translate between domain objects and Spark's internal format

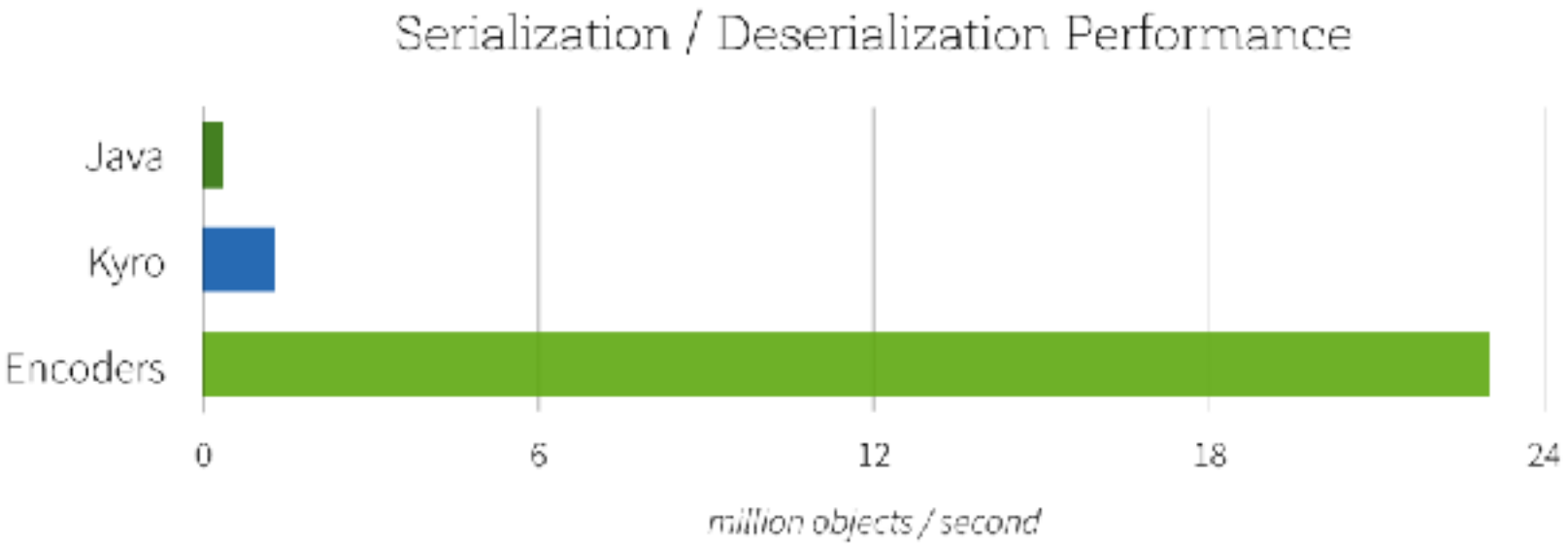
JVM Object

MyClass(123, “data”, “bricks”)



Internal Representation

Datasets: Lightning-fast Serialization with Encoders



Performance of Core Primitives

	cost per row (single thread)	
primitive	**Spark 1.6**	**Spark 2.0**
filter	15 ns	1.1 ns
sum w/o group	14 ns	0.9 ns
sum w/ group	79 ns	10.7 ns
hash join	115 ns	4.0 ns
sort (8 bit entropy)	620 ns	5.3 ns
sort (64 bit entropy)	620 ns	40 ns
sort-merge join	750 ns	700 ns

Intel Haswell i7 4960HQ 2.6GHz, HotSpot 1.8.0_60-b27, Mac OS X 10.11

Project Tungsten: Bringing Apache Spark Closer to Bare Metal

by Reynold Xin and Josh Rosen

In a previous blog post, we looked back and surveyed performance improvements made to Apache Spark in the past year. In this post, we look forward and share with you the next chapter, which we are calling Project Tungsten. 2014 witnessed Spark setting the world record in large-scale sorting and saw major improvements across the entire engine from Python to SQL to machine learning. Performance optimization is a never ending process.

Project Tungsten will be the largest change to Spark's execution engine since the project's inception. It focuses on substantially improving the efficiency of memory and CPU for Spark applications, to push performance closer to the limits of modern hardware. This effort includes three initiatives:

1. *Memory Management and Binary Processing*: leveraging application semantics to manage memory explicitly and eliminate the overhead of JVM object model and garbage collection
2. Cache-aware computation: algorithms and data structures to exploit memory hierarchy
3. Code generation: using code generation to exploit modern compilers and CPUs

SPARK SUMMIT 2016

Workshop: Notebook on DataFrames/ Datasets & Spark SQL

- Import Notebook into your Spark 2.0 Cluster
  - http://dbricks.co/sswksh2A
  - http://dbricks.co/sswksh2
  - https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.Dataset
- Work through each Notebook cell
- Try challenges
- Break..

Resources

- docs.databricks.com
- Spark Programming Guide
- Structured Streaming Programming Guide
- Databricks Engineering Blogs
- sparkhub.databricks.com
- https://spark-packages.org/

Do you have any questions
for my prepared answers?