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

Jules S. Damji
Apache Spark Community Evangelist

Spark Saturday Meetup Workshop

databricks

I have used Apache Spark Before...

I know the difference between DataFrame and RDDs…

Spark Community Evangelist & Developer Advocate @ Databricks

Developer Advocate @ Hortonworks

Software engineering @: Sun Microsystems, Netscape, @Home, VeriSign, Scalix, Centrify, LoudCloud/Opsware, ProQuest

https://www.linkedin.com/in/dmatrix
@2twitme

> Evangelism isn't a job title, it's a way of life. – Guy Kawasaki

Agenda for the day

Morning

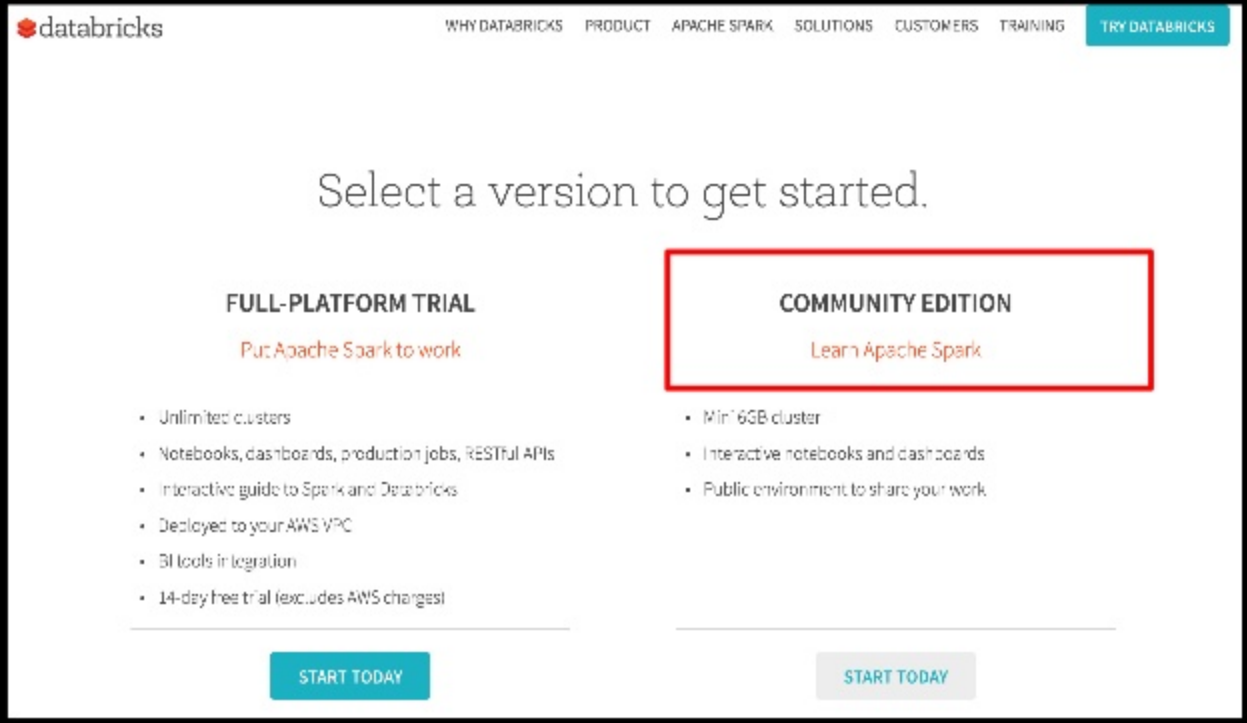
- Get to know Databricks
- Overview of Spark Fundamentals & Architecture
- What's New in Spark 2.x
- Break
- Unified APIs: SparkSessions, SQL, DataFrames, Datasets…
- Workshop Notebook 1
- Lunch

Afternoon

- Introduction to DataFrames, Datasets and Spark SQL
- Workshop Notebook 2
- Break
- Introduction to Structured Streaming Concepts
- Workshop Notebook 3
- Go Home…

Get to know Databricks

1. Get http://databricks.com/try-databricks
2. https://github.com/dmatrix/spark-saturday
3. [OR] Import Notebook: http://dbricks.co/ss_wkshp0



databricks

Why Apache Spark?

Big Data Systems of Yesterday…

MapReduce/Hadoop

Pregel
Giraph
Dremel
Mahout
Storm
Impala
Drill
…

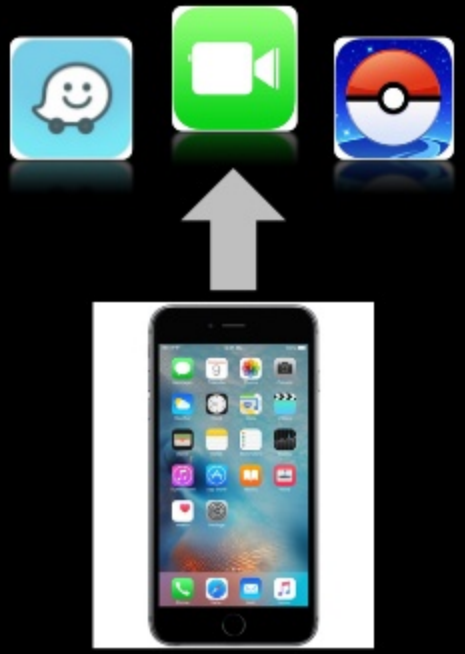
General batch processing

Specialized systems for new workloads

Hard to *combine* in pipelines

An Analogy ....

New applications



Apache Spark Philosophy

Unified engine for complete data applications

High-level user-friendly APIs

Unified engine across diverse workloads & environments

CONTRIBUTED ARTICLES

Apache Spark: A Unified Engine for Big Data Processing

By Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, Ion Stoica

Communications of the ACM, Vol. 59 No. 11, Pages 56-65

10.1145/2934664

Comments

SIGN IN for Full Access

User Name

About Databricks

TEAM

Started Spark project (now Apache Spark) at UC Berkeley in 2009

MISSION

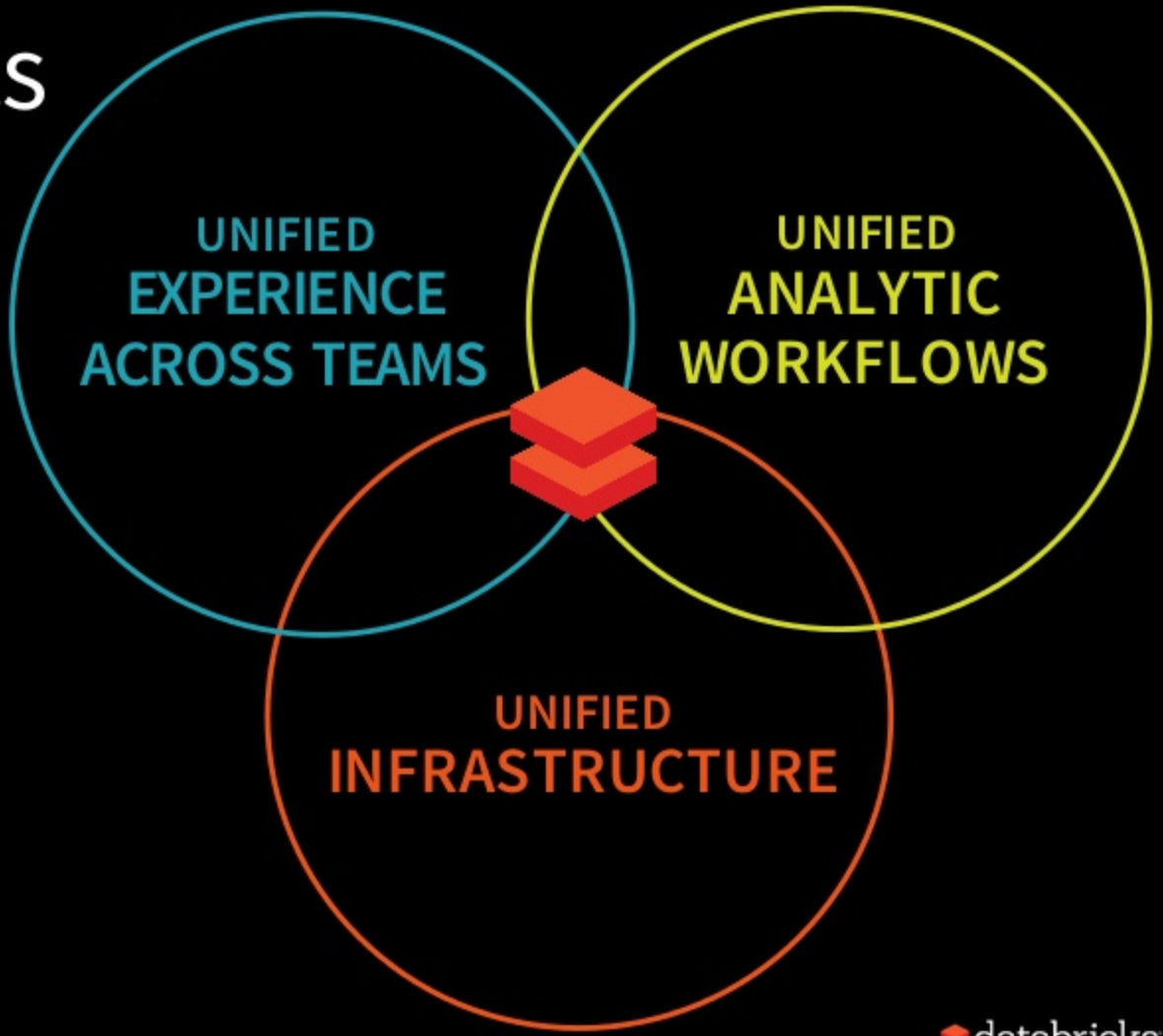
Making Big Data Simple

PRODUCT

Unified Analytics Platform

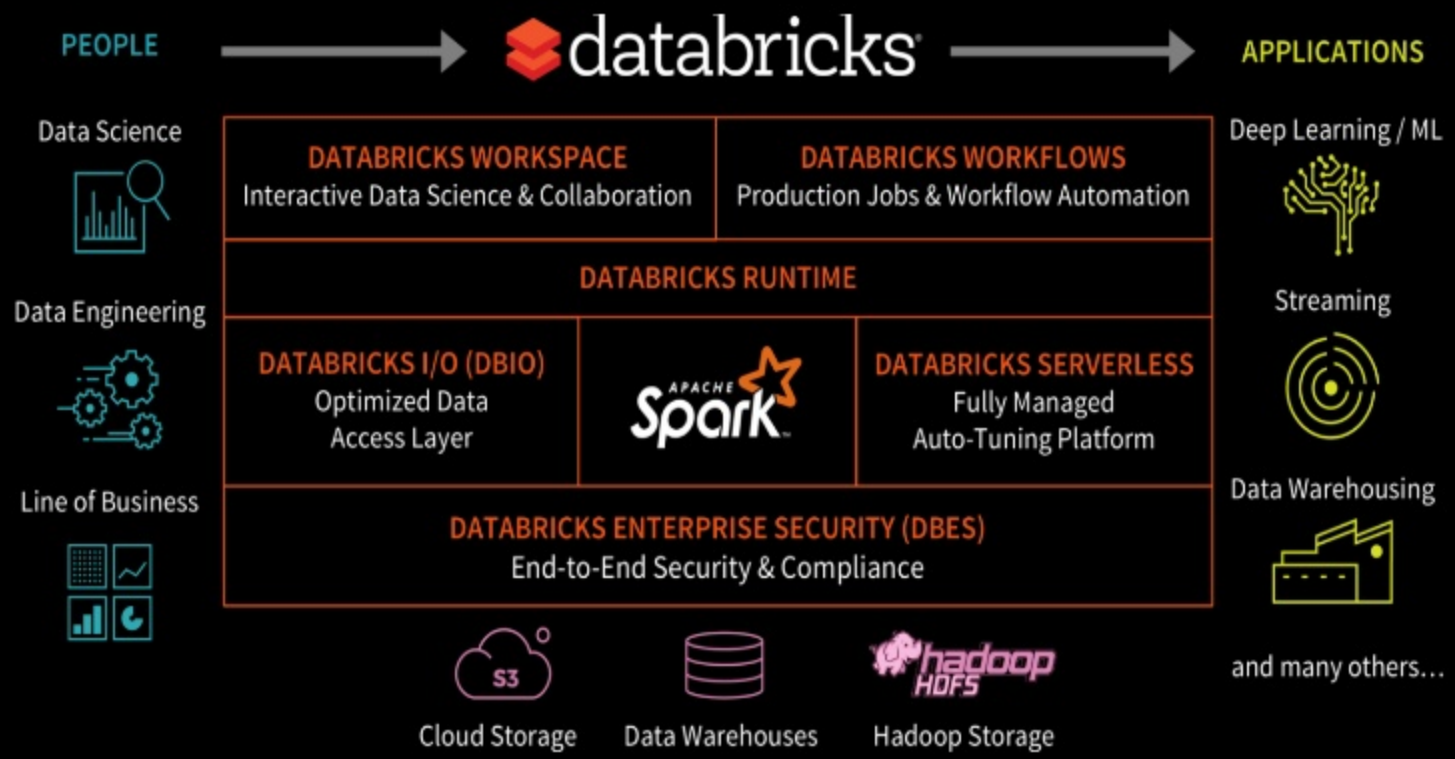
Unified Analytics Platform

Accelerate innovation by unifying data science, engineering and business.



databricks

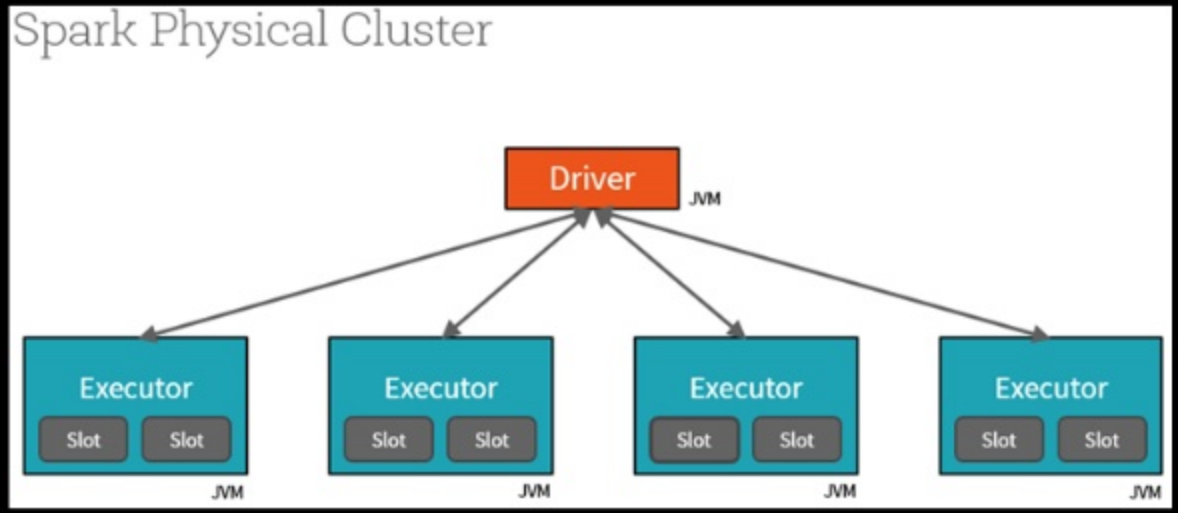
The Unified Analytics Platform



Apache Spark Architecture

databricks

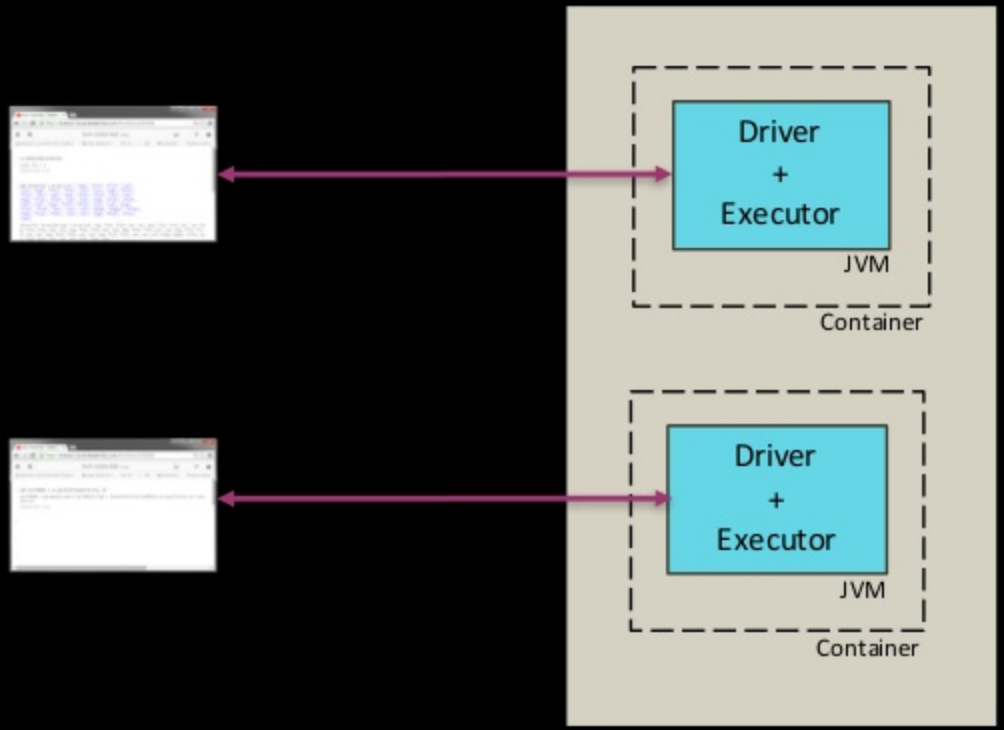
Apache Spark Architecture



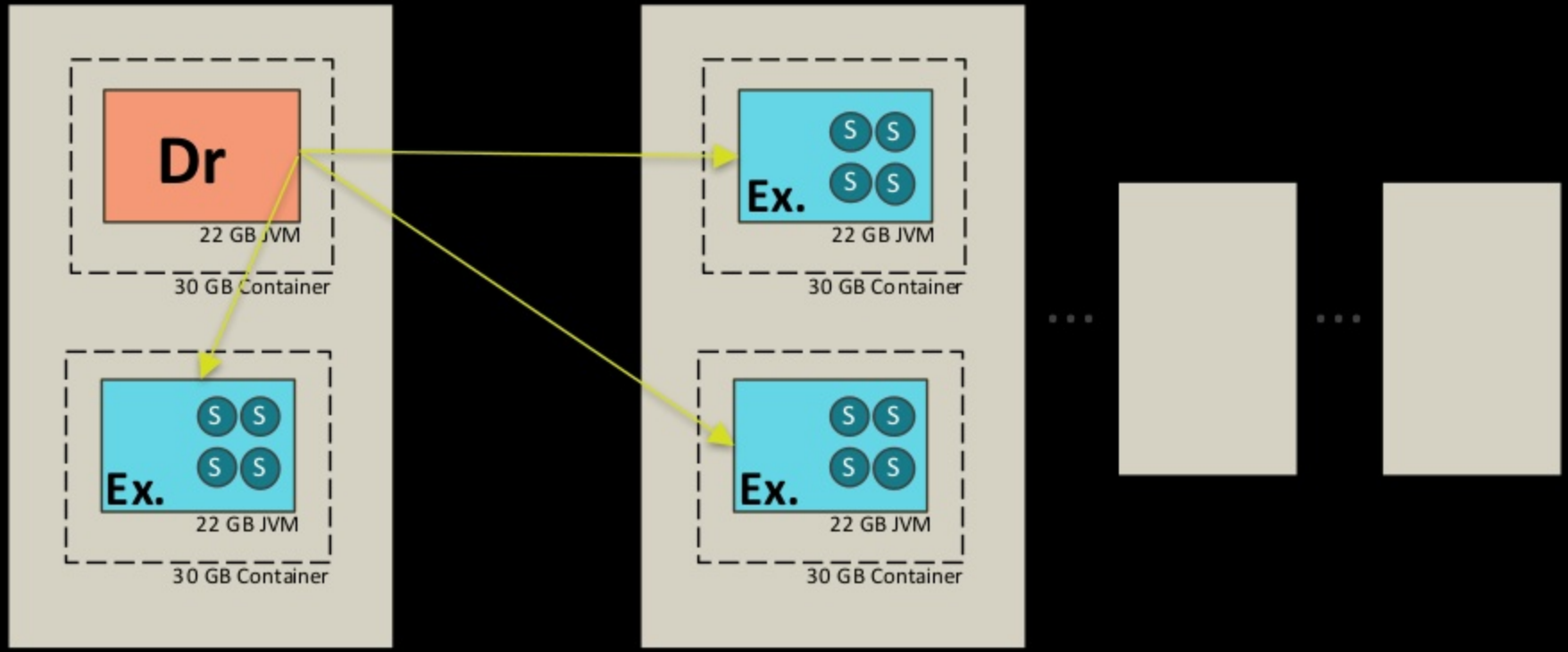
Deployments Modes

- Local
- Standalone
- YARN
- Mesos

Local Mode in Databricks



Standalone Mode



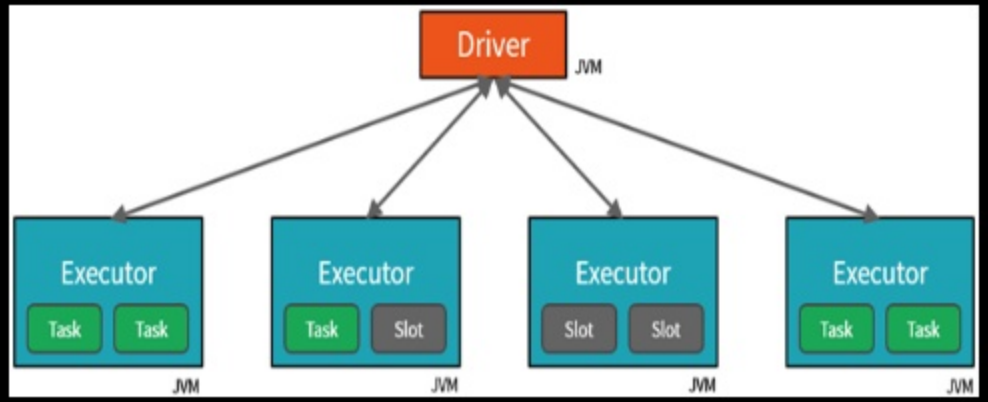
Spark Deployment Modes

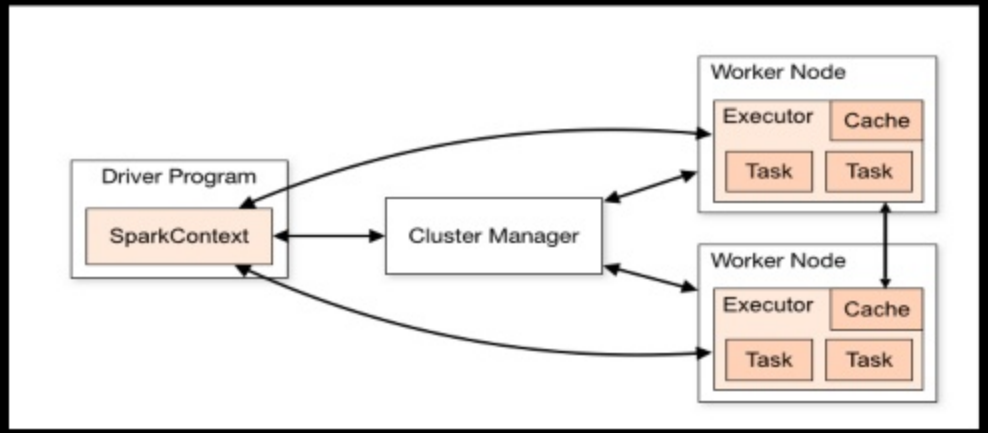
MODE	DRIVER	WORKER	EXECUTOR	MASTER
Local	Runs on a single JVM	Runs on the same JVM as the driver	Runs on the same JVM as the driver	Runs on a single host
Standalone	Can run on any node in the cluster	Runs on its own JVM on each node	Each worker in the cluster will launch its own JVM	Can be allocated arbitrarily where the master is started
Yarn (client)	On a client, not part of the cluster	YARN NodeManager	YARN's NodeManager's Container	YARN's Resource Manager works with YARN's Application Master to allocate the containers on NodeManagers for Executors.
YARN (cluster)	Runs within the YARN's Application Master	Same as YARN client mode	Same as YARN client mode	Same as YARN client mode
Mesos (client)	Runs on a client machine, not part of Mesos cluster	Runs on Mesos Slave	Container within Mesos Slave	Mesos' master
Mesos (cluster)	Runs within one of Mesos' master	Same as client mode	Same as client mode	Mesos' master

Table 1. Cheat Sheet Depicting Deployment Modes And Where Each Spark Component Runs

Apache Spark Architecture

An Anatomy of an Application

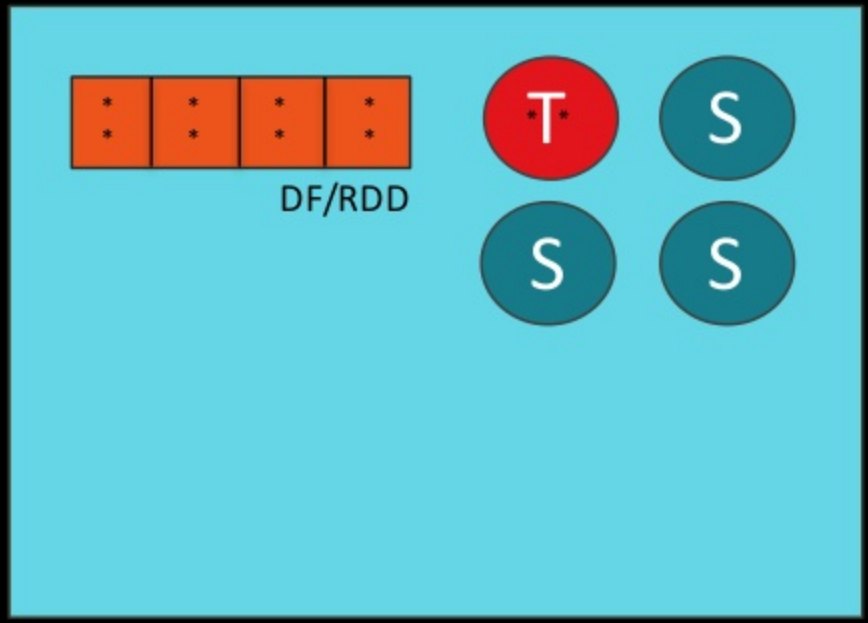




Spark Application

- Jobs
- Stages
- Tasks

A Spark Executor

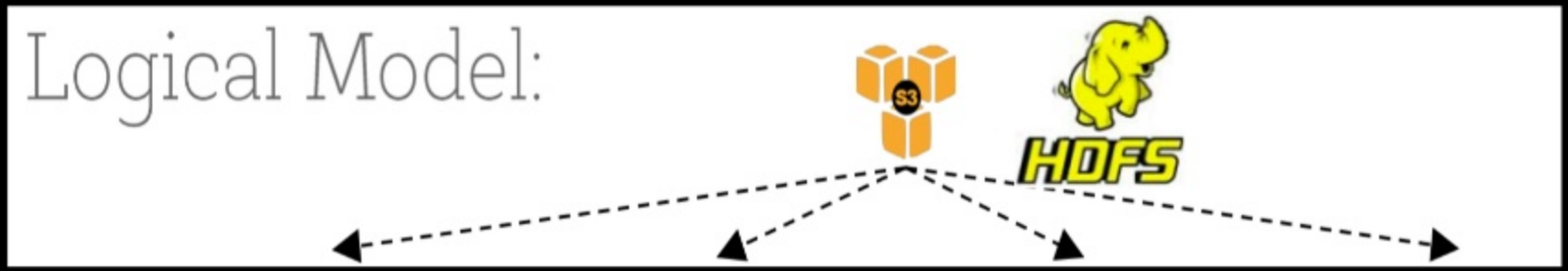


Resilient Distributed Dataset (RDD)

What are RDDs?

- … Distributed data abstraction
- … Resilient & Immutable
- … Lazy
- … *Compile Type-safe*
- … *Semi-structured or unstructured*

Logical Model:



ts	m	1304
ts	d	2237
ts	m	1600

ts	d	3901
ts	d	2491
ts	d	2288

ts	m	1172
ts	m	2137
ts	d	3176

ts	m	2538
ts	d	2837
ts	d	3400

TRANSFORMATIONS

Apache Spark Operations =

+

ACTIONS

Transformations *(lazy)*	Actions
`orderBy`	`show`
`filter`	`count`
`groupBy`	`take`
`select`	`collect`
`drop`	`save`
`join`	

Transformations contribute to a query plan,
but nothing is executed until an action is called

vs

distributed

occurs across the cluster

driver

result must fit in driver JVM

 = easy  = medium

Essential Core & Intermediate Spark Operations

TRANSFORMATIONS

General

- map
- filter
- flatMap
- mapPartitions
- mapPartitionsWithIndex
- groupBy
- sortBy

Math / Statistical

- sample
- randomSplit

Set Theory / Relational

- union
- intersection
- subtract
- distinct
- cartesian
- zip

Data Structure / I/O

- 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

= easy = medium

Essential Core & Intermediate Spark Operations

TRANSFORMATIONS

General
- flatMapValues
- groupByKey
- reduceByKey
- reduceByKeyLocally
- foldByKey
- aggregateByKey
- sortByKey
- combineByKey
- keys
- values

Math / Statistical
- sampleByKey

Set Theory / Relational
- cogroup (=groupWith)
- join
- subtractByKey
- fullOuterJoin
- leftOuterJoin
- rightOuterJoin

Data Structure / I/O
- partitionBy

ACTIONS

- countByKey
- countByValue
- countByValueApprox
- countApproxDistinctByKey
- countApproxDistinctByKey
- countByKeyApprox
- sampleByKeyExact

Types of RDDs:

HadoopRDD: core functionality for reading data in HDFS using older MR API

MapPartitionsRDD: a result of calling actions like `map`, `flatMap`, `filter`, `mapPartitions`, etc

PairRDD: holds key/value pairs, a result of `groupByKey` and `join` operations

PipedRDD: result of piping elements to a forked external process

ShuffledRDD: a result after shuffling (`repartition` or `coalesce`)

7 Steps to Mastering Apache Spark 2.0

Sep 2016 KDnuggets Silver Blog

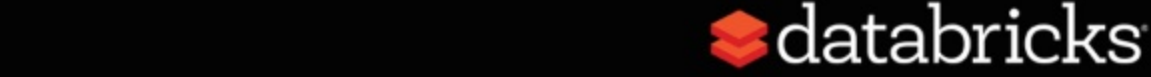
Looking for a comprehensive guide on going from zero to Apache Spark hero in steps? Look no further! Written by our friends at Databricks, this exclusive guide provides a solid foundation for those looking to master Apache Spark 2.0.

By Jules S. Damji & Sameer Farooqui, Databricks.

Not a week goes by without a mention of Apache Spark in a blog, news article, or webinar on Spark's impact in the big data landscape. Not a meetup or conference on big data or advanced analytics is without a speaker that expounds on aspects of Spark—touting of its rapid adoption; speaking of its developments; explaining of its uses cases, in enterprises across industries.

databricks

How did we get here…?
Where we going…?



A Brief History

Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica
University of California, Berkeley

2010 — Started @ UC Berkeley

2012 — Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

2013 — Databricks started & donated to ASF

2014 — Spark 1.0 & libraries (SQL, ML, GraphX)

2015 — DataFrames/Datasets, Tungsten, Catalyst Optimizer, ML Pipelines

2016-17 — Apache Spark 2.0, 2.1, 2.2, Structured Streaming, Cost Based Optimizer, Deep Learning Pipelines

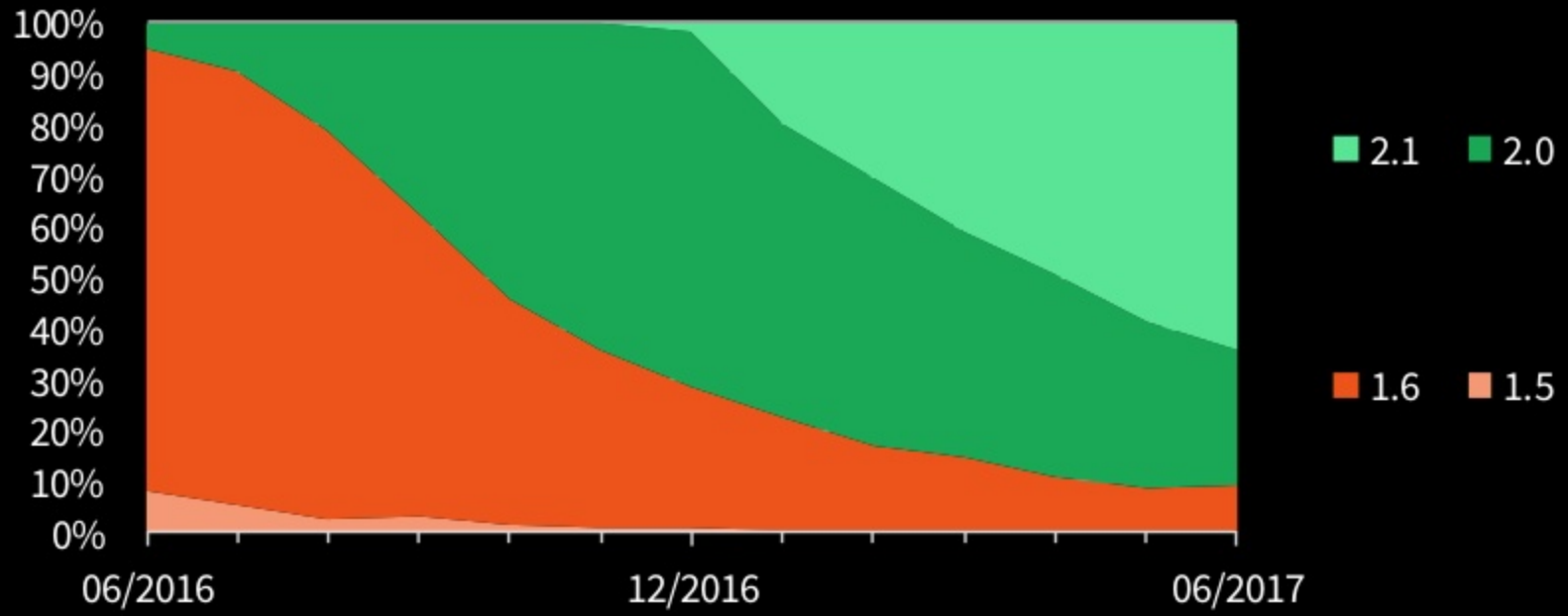
Easier Smarter Faster

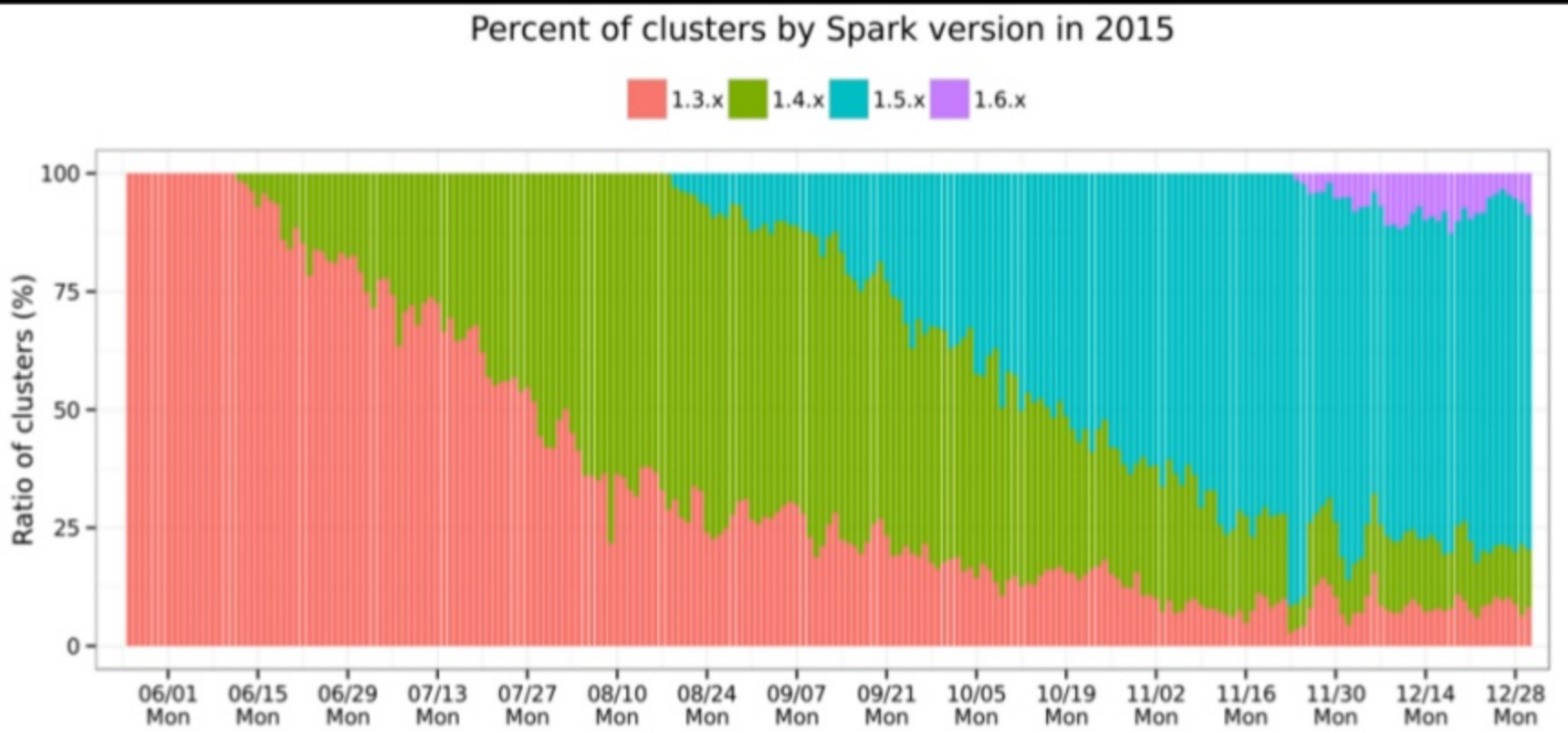
Apache Spark 2.X

- Steps to Bigger & Better Things….

Builds on all we learned in past 2 years

Spark Version Usage in Databricks





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

Major Themes in Apache Spark 2.x

Unifying Datasets and DataFrames & SparkSessions

Easier

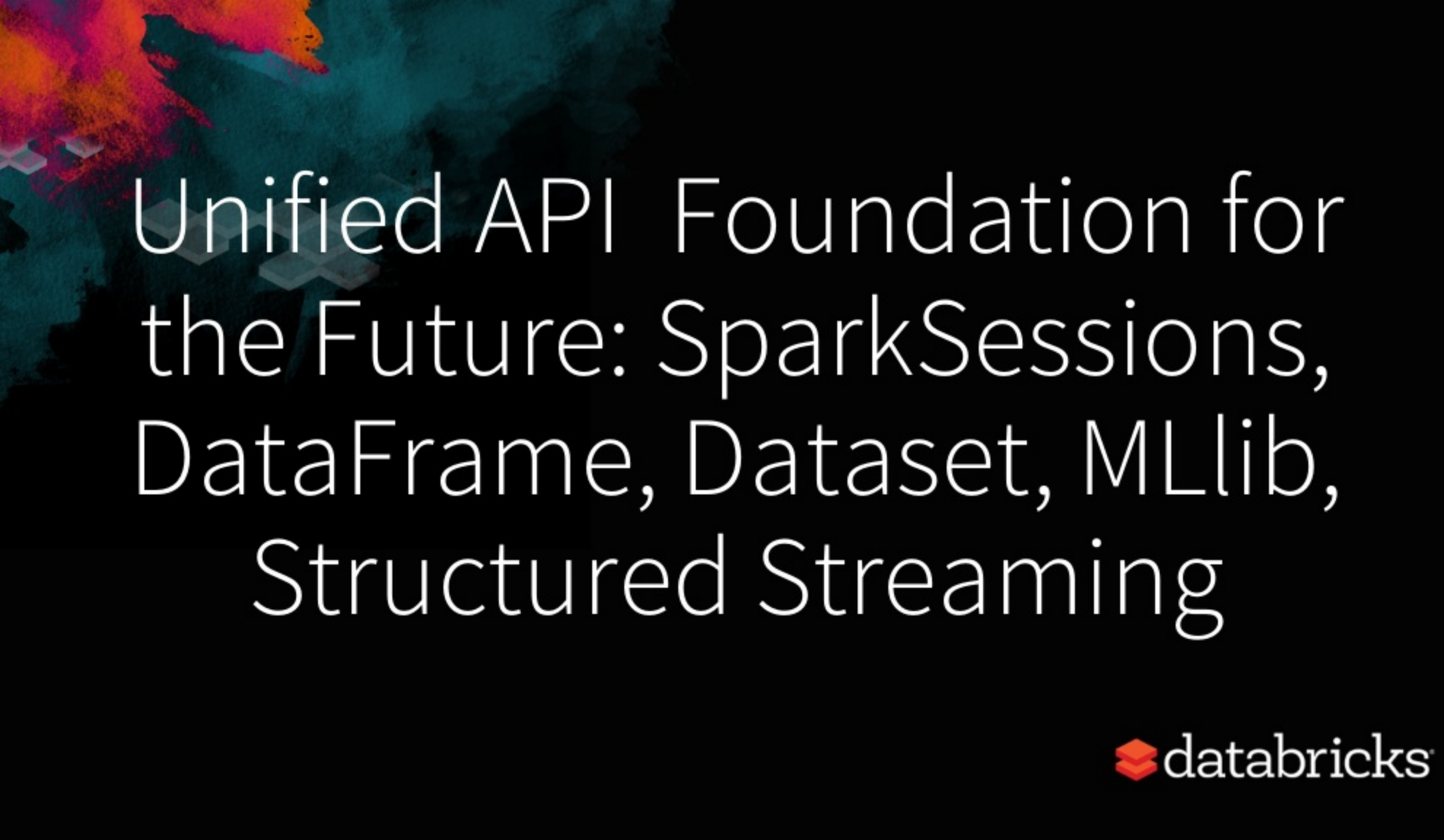
Tungsten Phase 2 speedups of 5-10x & Catalyst Optimizer

Faster

Structured Streaming real-time engine on SQL / DataFrames

Smarter

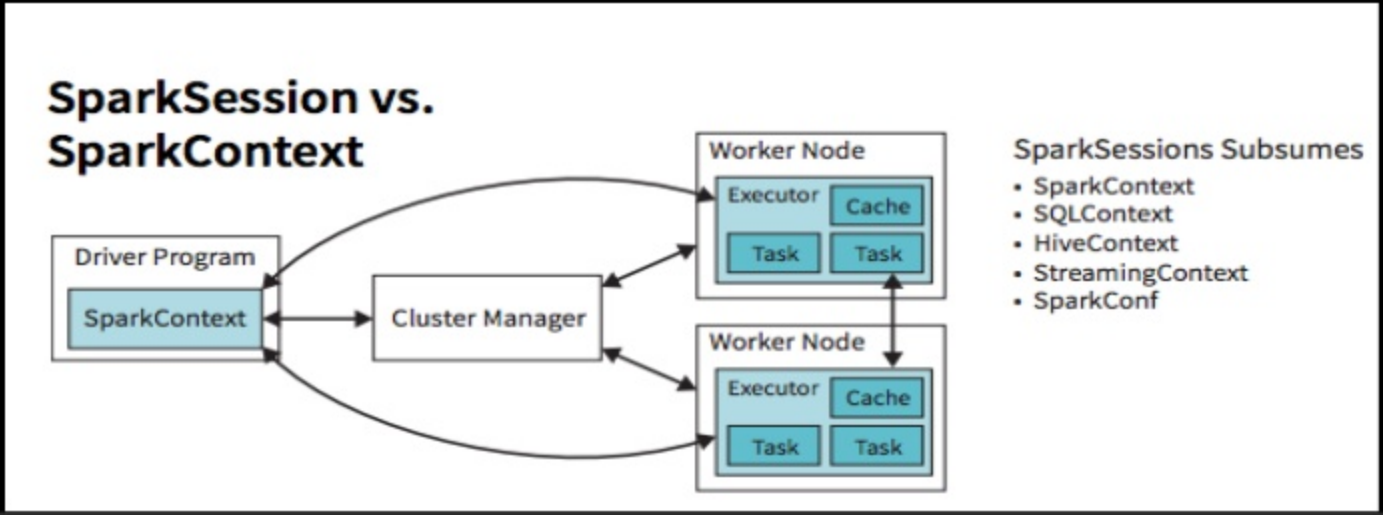
Unified API Foundation for the Future: SparkSessions, DataFrame, Dataset, MLlib, Structured Streaming



SparkSession – A Unified entry point to Spark

- Conduit to Spark
  - Creates Datasets/DataFrames
  - Reads/writes data
  - Works with metadata
  - Sets/gets Spark Configuration
  - Driver uses for Cluster resource management

SparkSession vs SparkContext




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

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

SparkSession – A Unified entry point to Spark

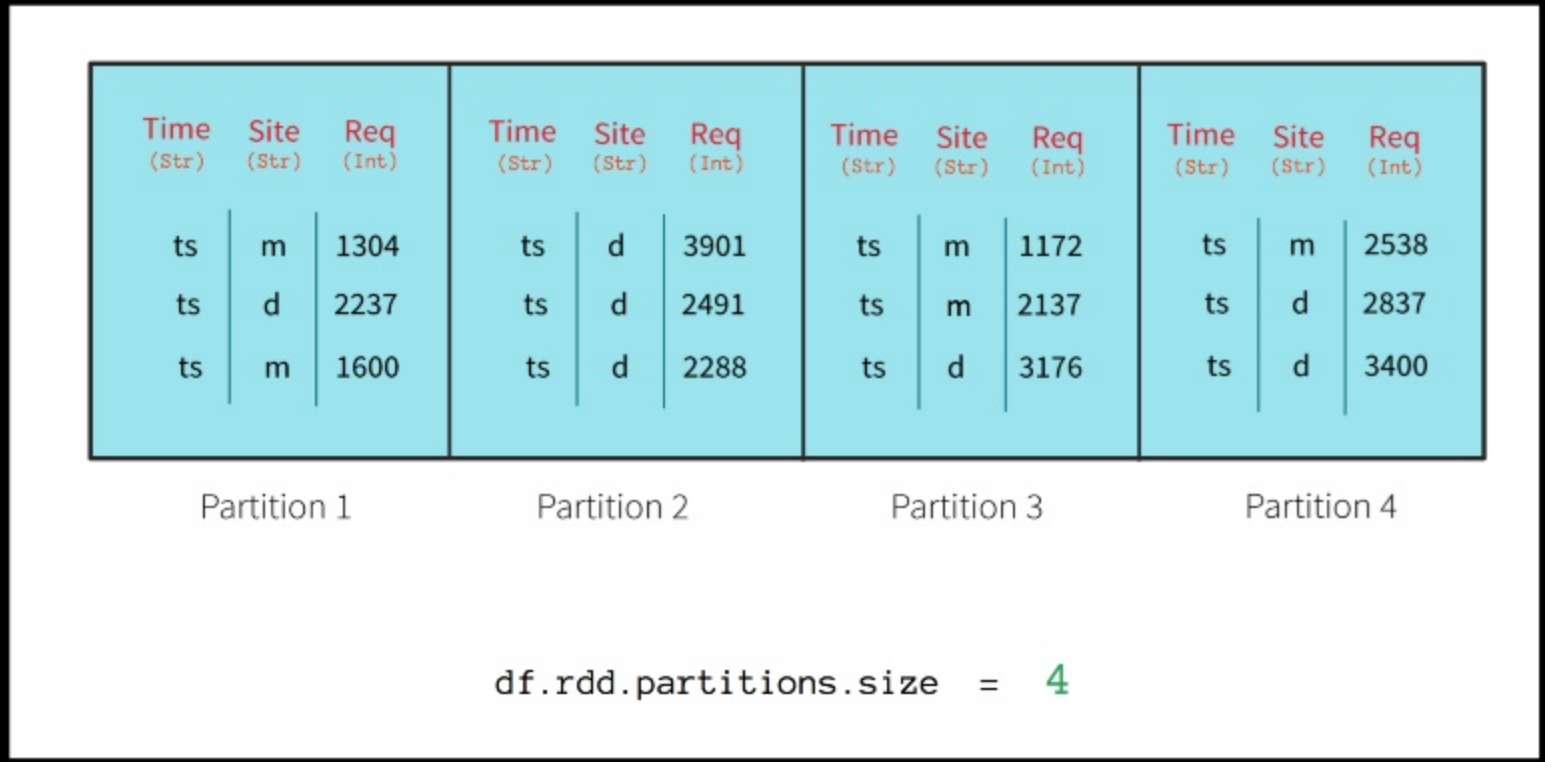
How to use SparkSession in Apache Spark 2.0

A unified entry point for manipulating data with Spark

by Jules Damji
Posted in **ENGINEERING BLOG** | August 15, 2016

Try this notebook in Databricks

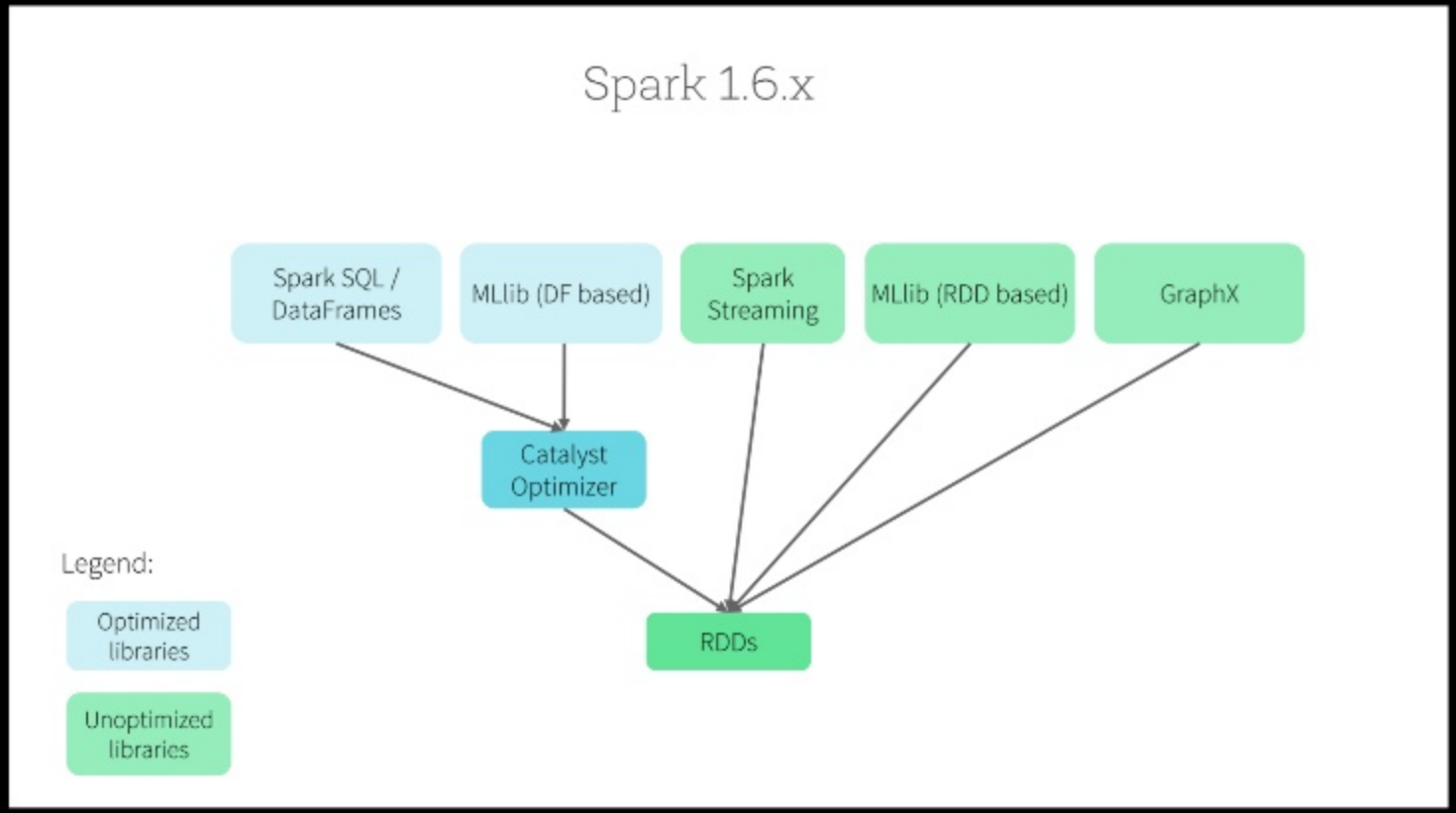
DataFrame & Dataset Structure



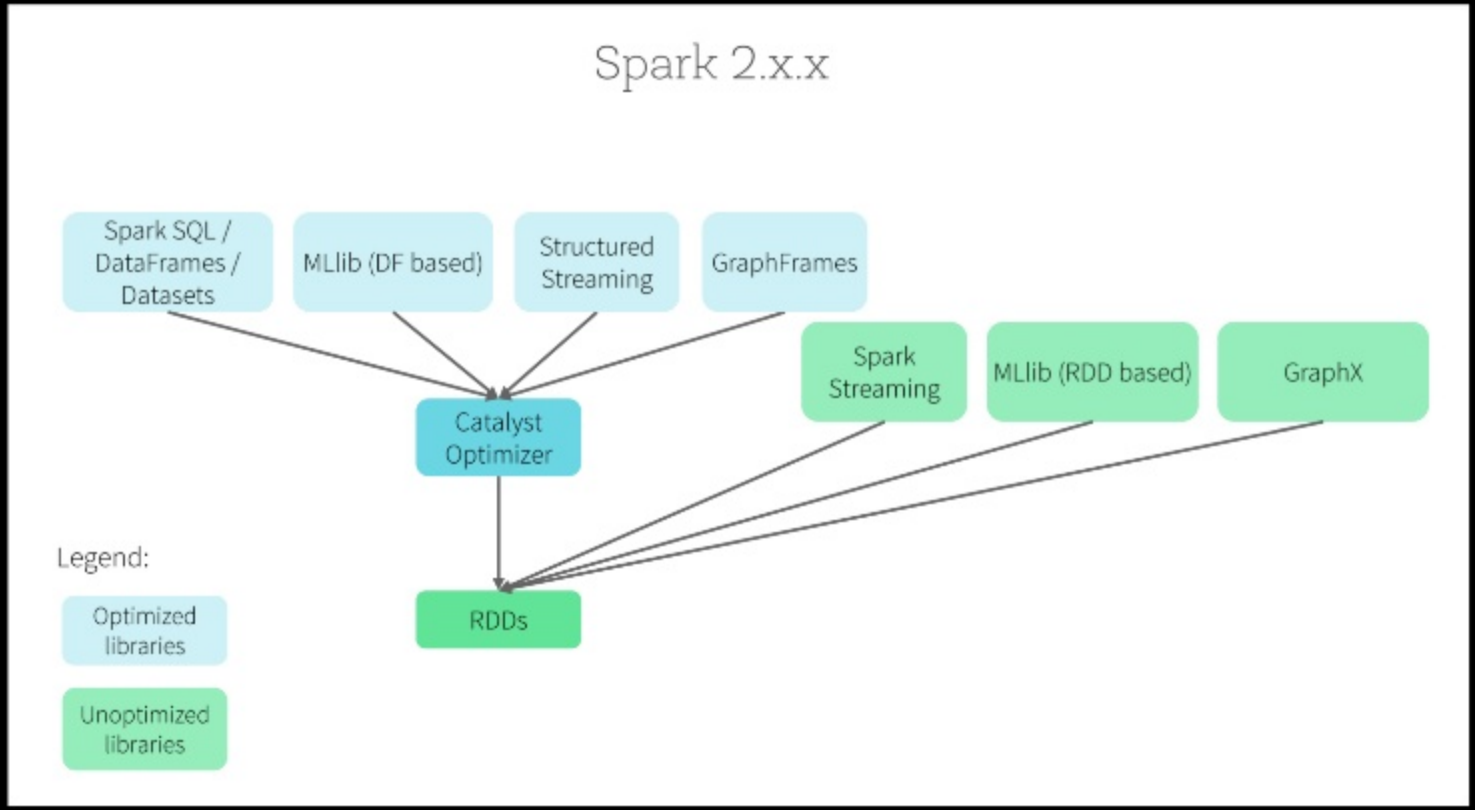
Long Term

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

Spark 1.6 vs Spark 2.x

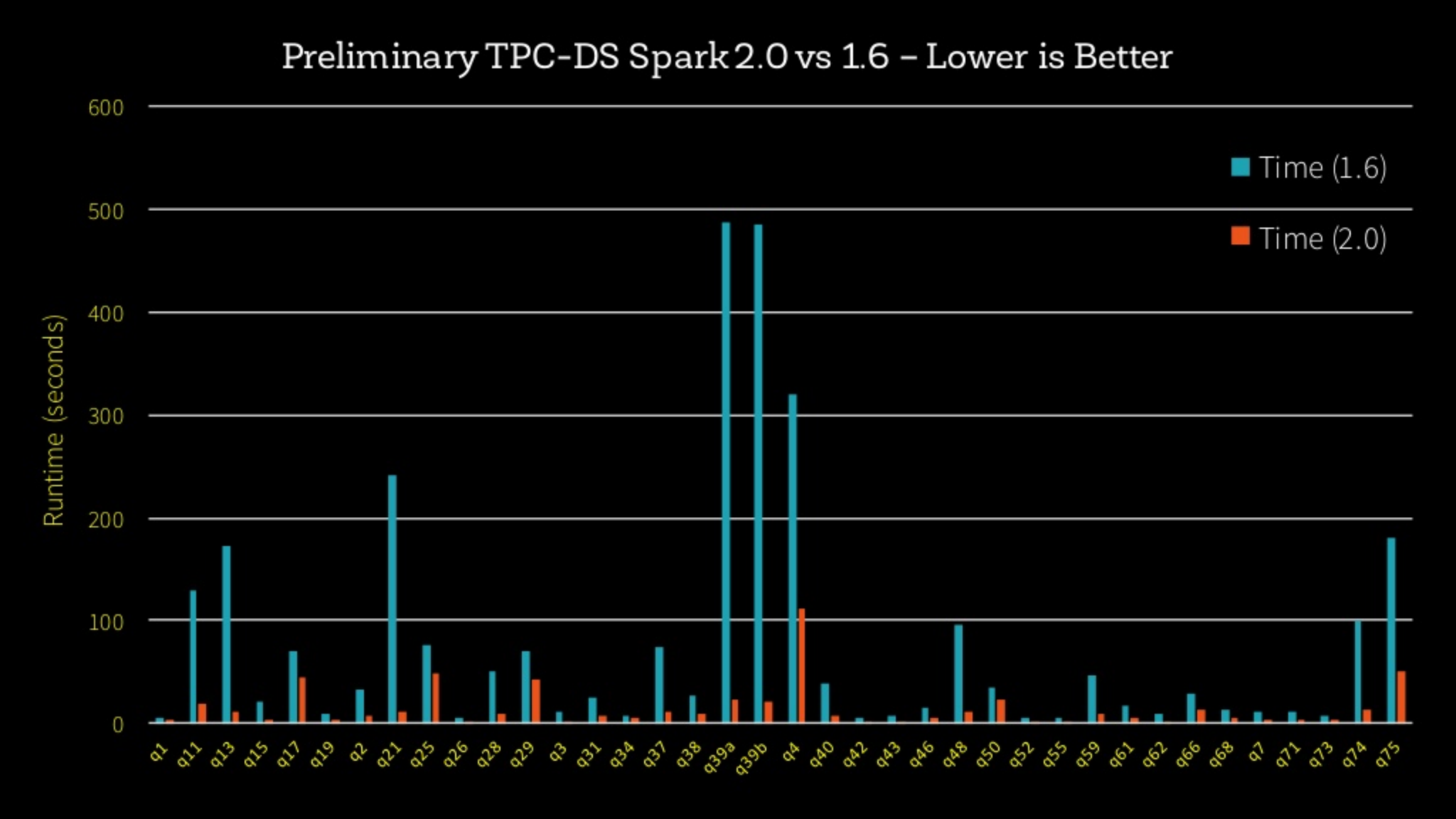


Spark 1.6 vs Spark 2.x



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/2016/06/17/sql-subqueries-in-apache-spark-2-0.html


Preliminary TPC-DS Spark 2.0 vs 1.6 – Lower is Better
Runtime (seconds)
600
500
400
300
200
100
0
Time (1.6)
Time (2.0)
q1
q11
q13
q15
q17
q19
q2
q21
q25
q26
q28
q29
q3
q31
q34
q37
q38
q39a
q39b
q4
q40
q42
q43
q46
q48
q50
q52
q55
q59
q61
q62
q66
q68
q7
q71
q73
q74
q75

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
- Structured Streaming Features & Production Readiness
- https://databricks.com/blog/2017/07/11/introducing-apache-spark-2-2.html

Workshop: Notebook on SparkSession

- Import Notebook into your Spark 2.2 Cluster
  - http://dbricks.co/ss_wkshp1
  - 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/Dataset, Spark SQL & Catalyst Optimizer

The not so secret truth…

SQL

is not about SQL

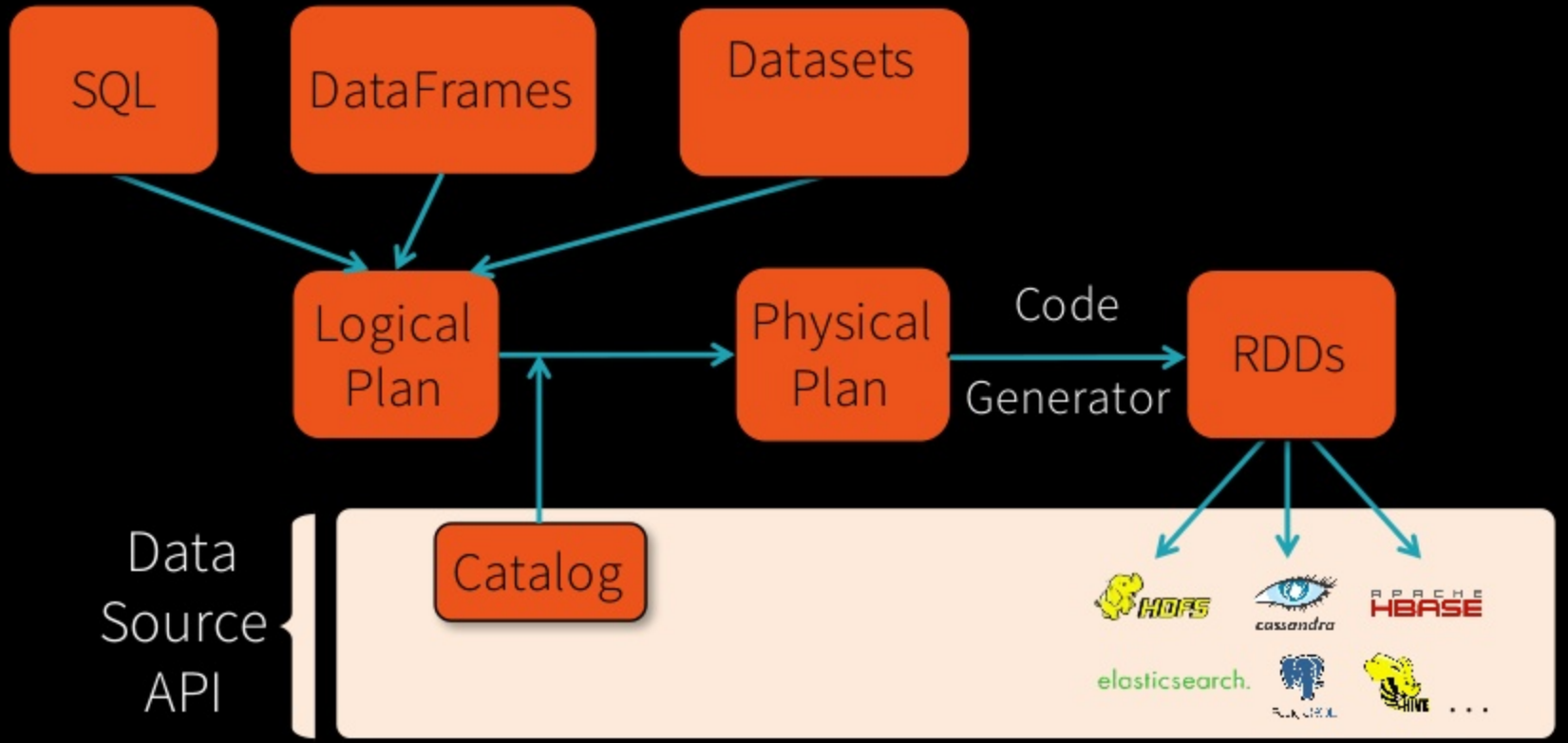
is about more than SQL

Spark SQL: The whole story

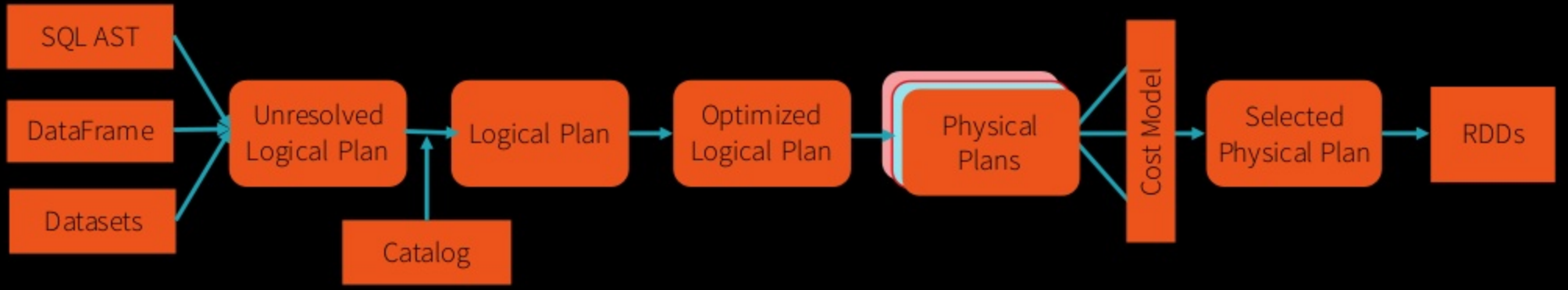
Is About Creating and Running Spark Programs Faster:

- Write less code
- Read less data
- Do less work
  - optimizer does 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

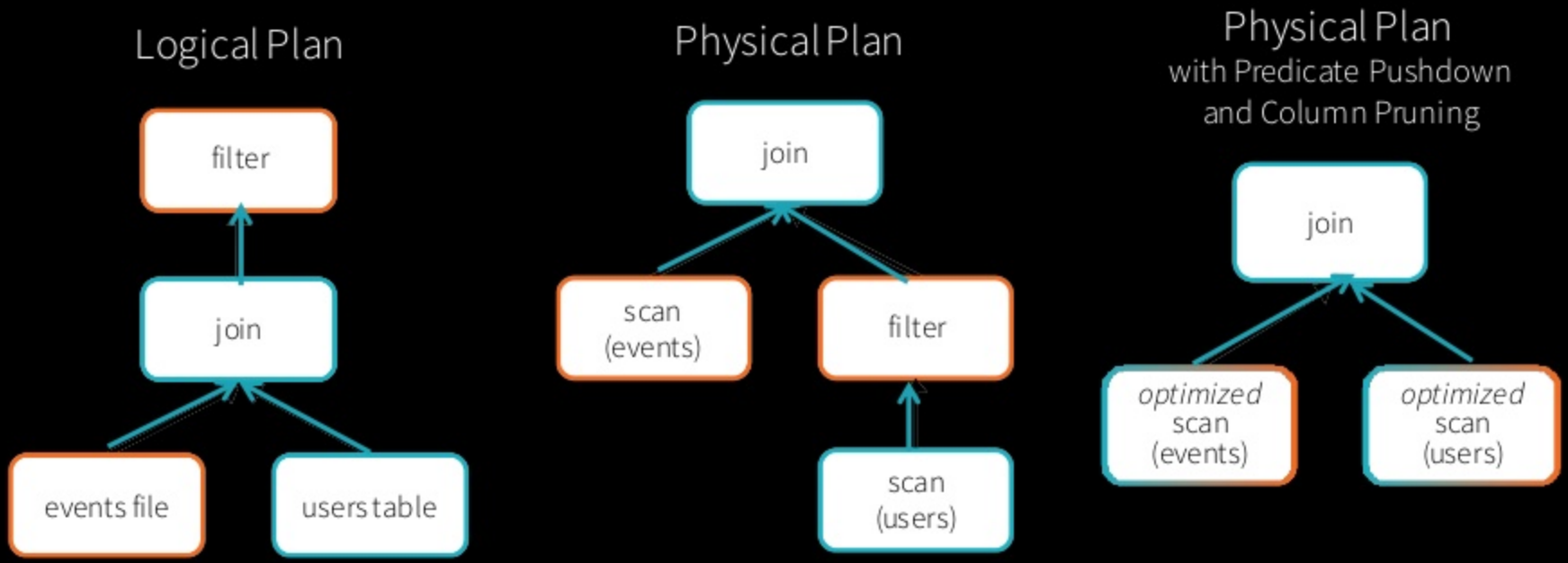
- Push filter predicate down to data source, so irrelevant data can be skipped
- **Parquet:** skip entire blocks, turn comparisons into cheaper integer comparisons via dictionary coding
- RDMS: reduce amount of data traffic by pushing down predicates

PHYSICAL OPTIMIZATIONS

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

DataFrame Optimization

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



Columns: Predicate pushdown

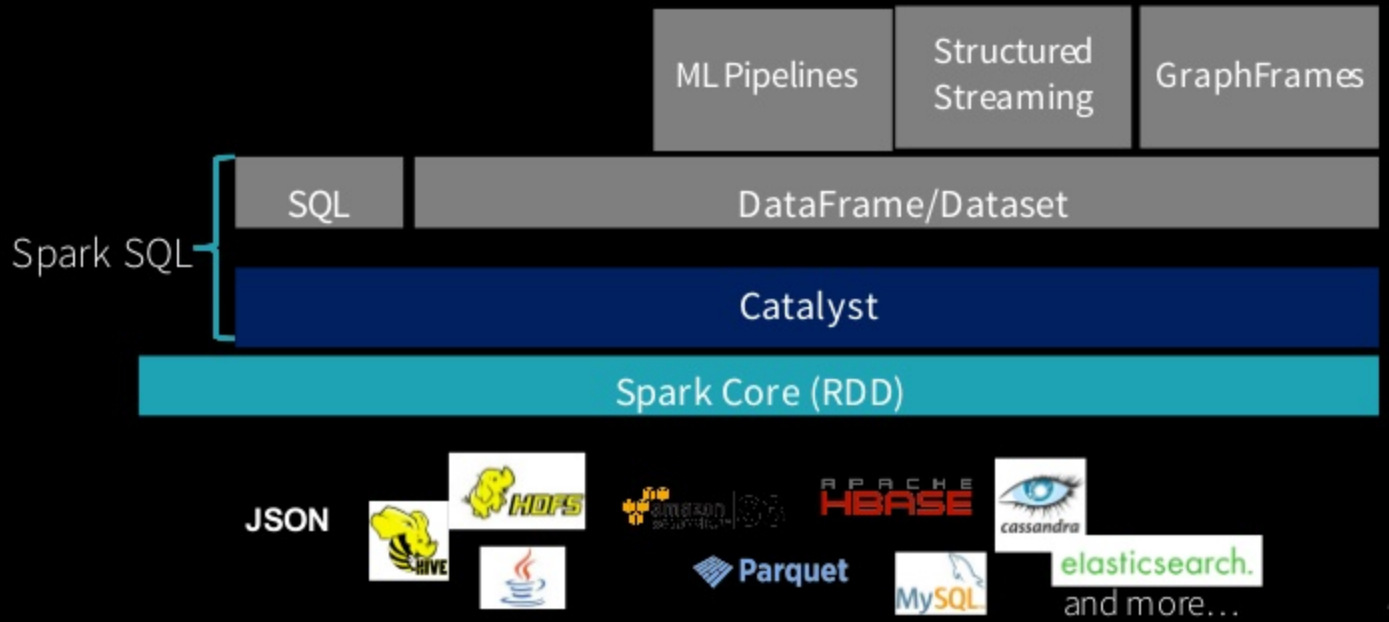
You Write

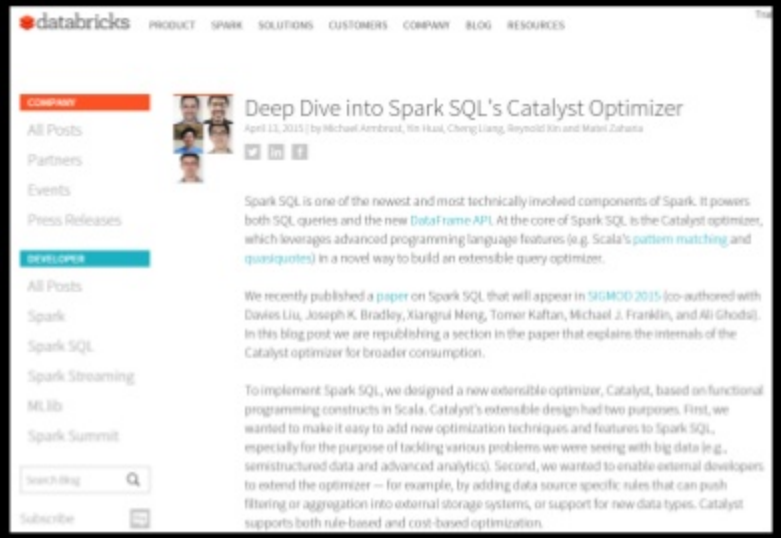
```
spark.read
  .format("jdbc")
  .option("url", "jdbc:postgresql:dbserver")
  .option("dbtable", "people")
  .load()
  .where($"name" === "michael")
```

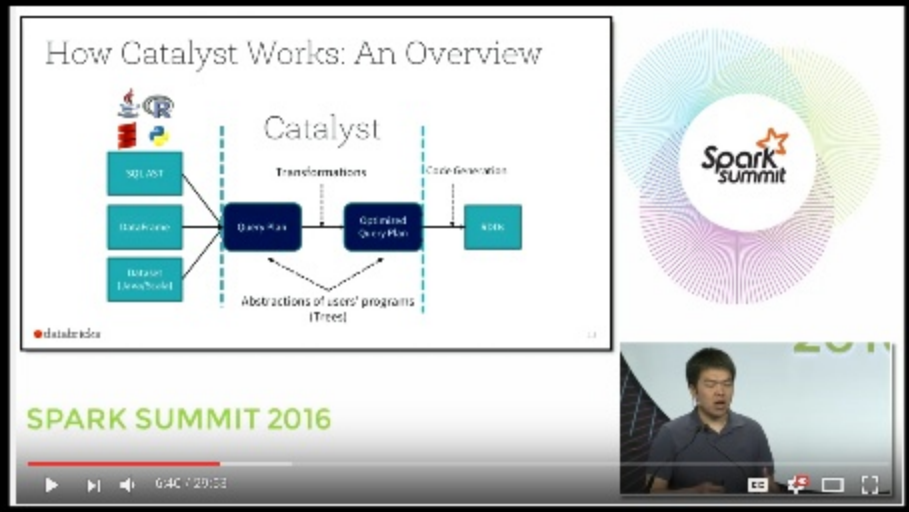
Spark Translates For Postgres

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

Foundational Spark 2.x Components

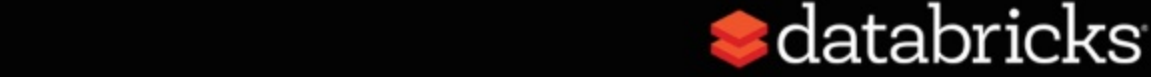






http://people.csail.mit.edu/matei/papers/2015/sigmod_spark_sql.pdf

Datasets Spark 2.x APIs



Background: What is in an RDD?

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

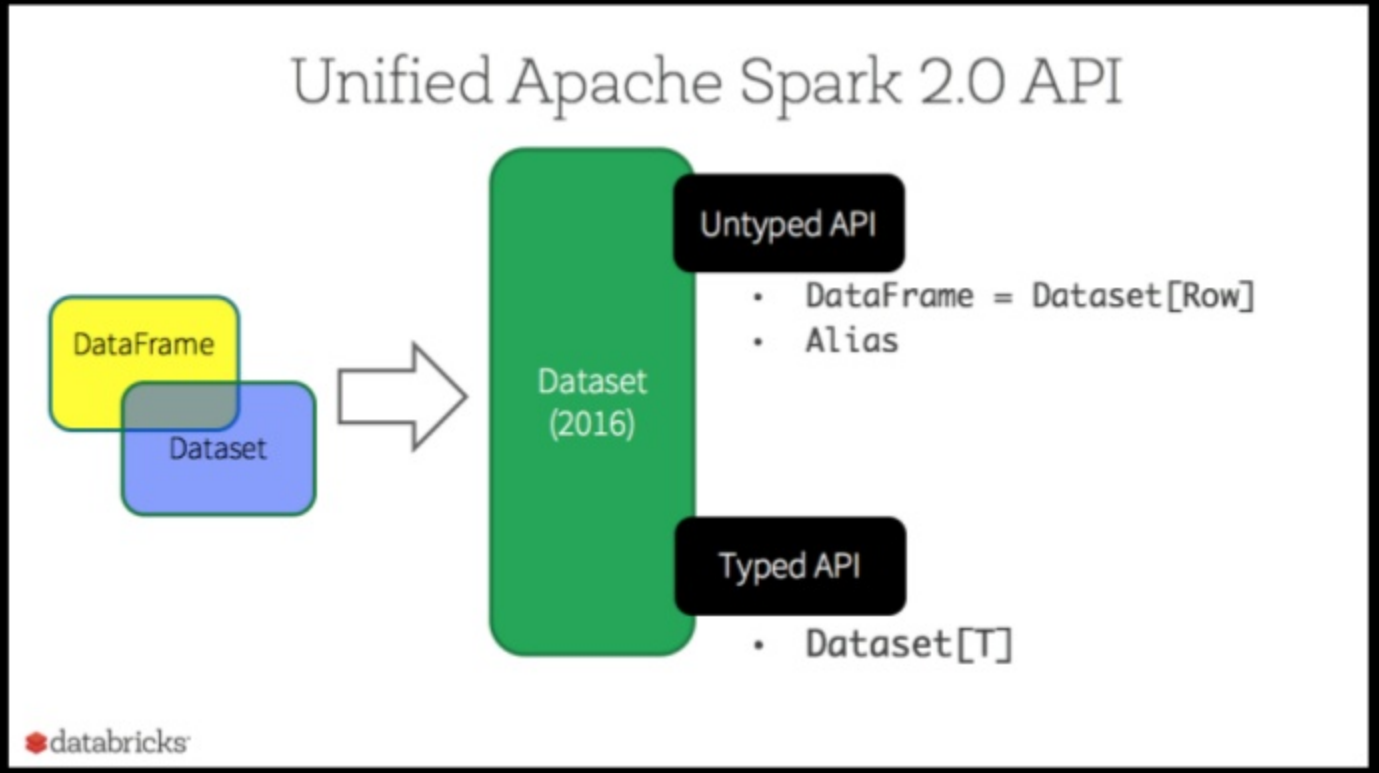
Opaque Computation
& Opaque Data

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

Unification of APIs in Spark 2.0

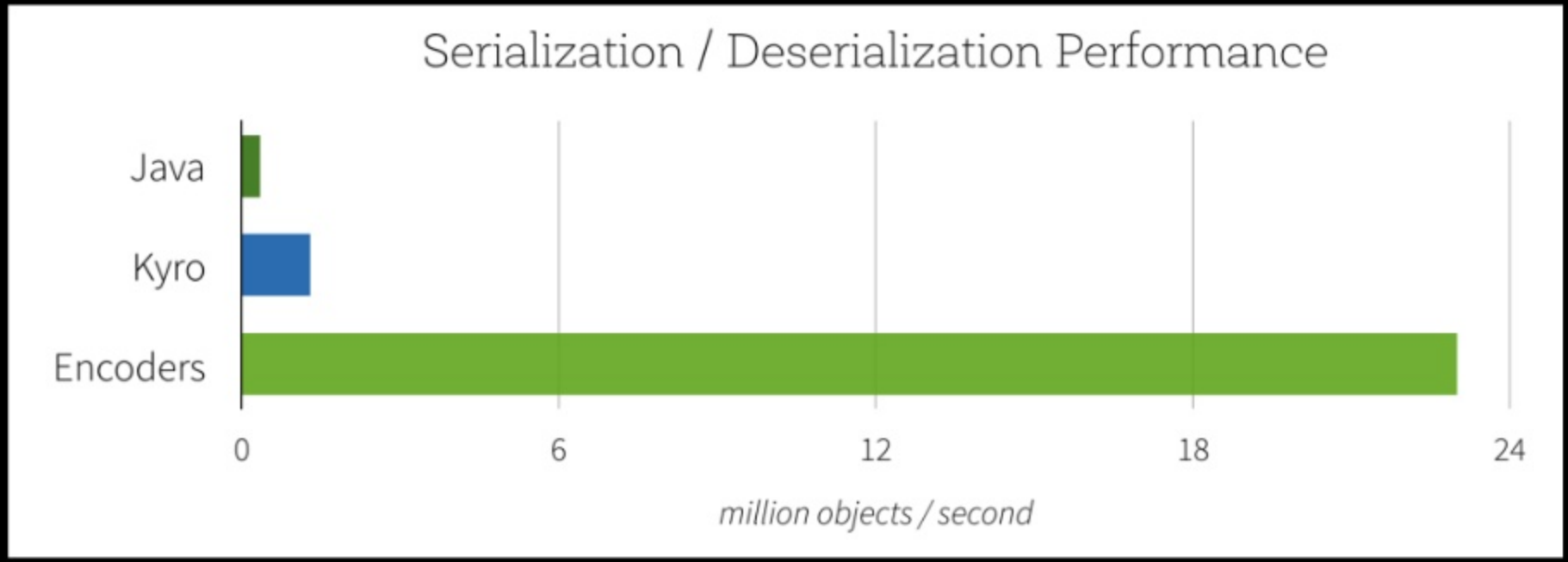


Dataset API in Spark 2.x

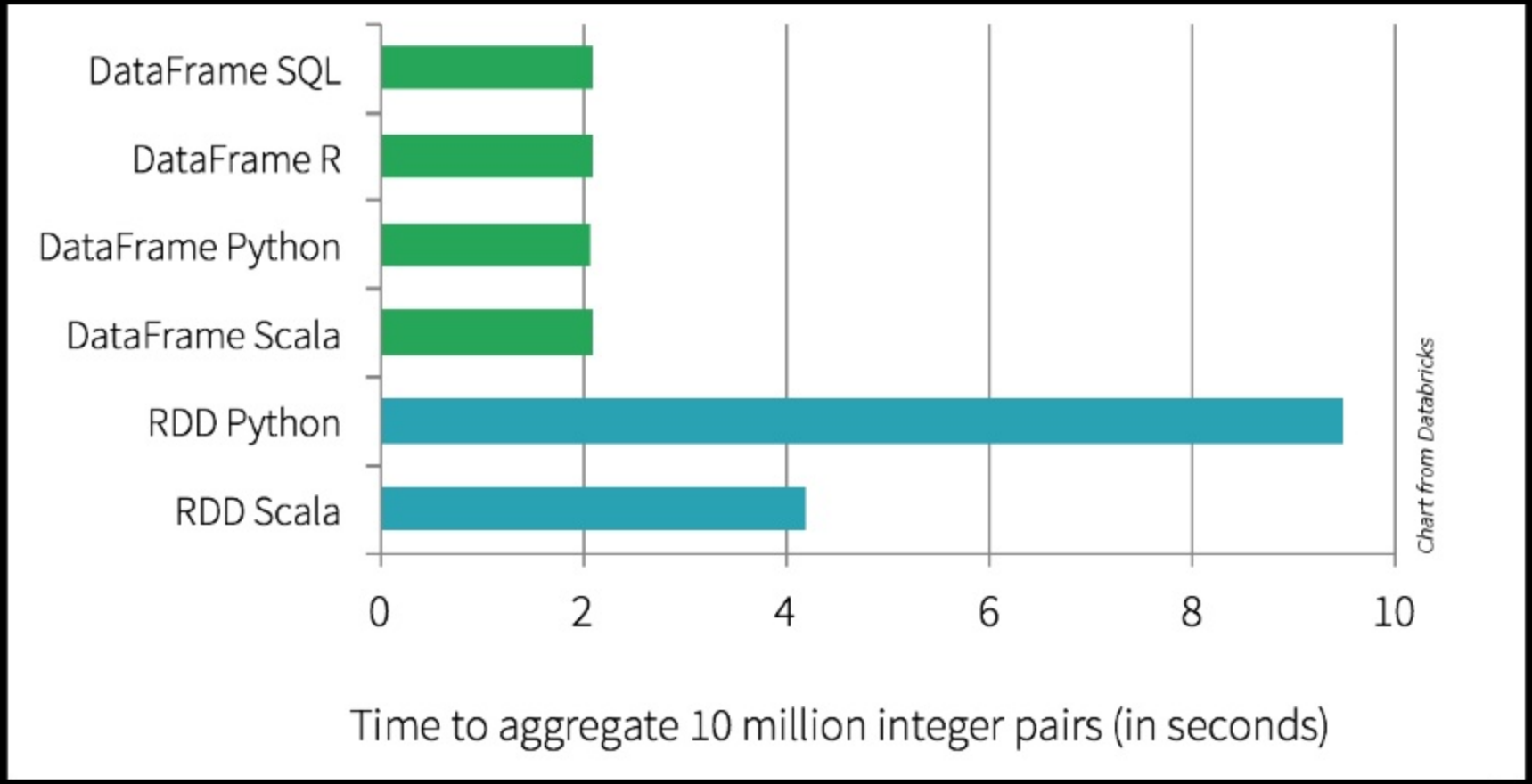
Type-safe: operate on domain objects with compiled lambda functions

```
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]
val filterDS = ds.filter(p => p.age > 30)
```

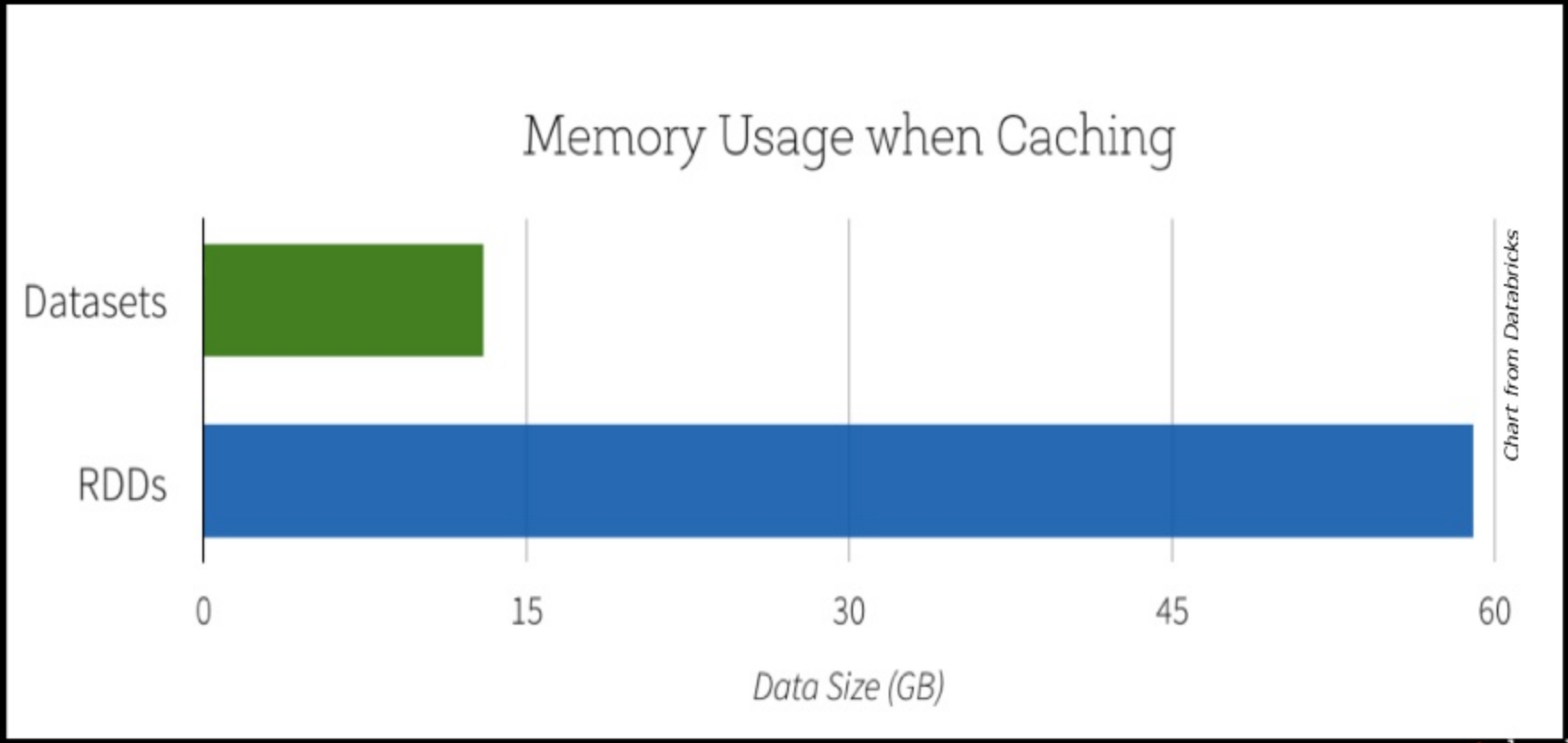
Datasets: Lightning-fast Serialization with Encoders



DataFrames are Faster than RDDs



Datasets < Memory RDDs



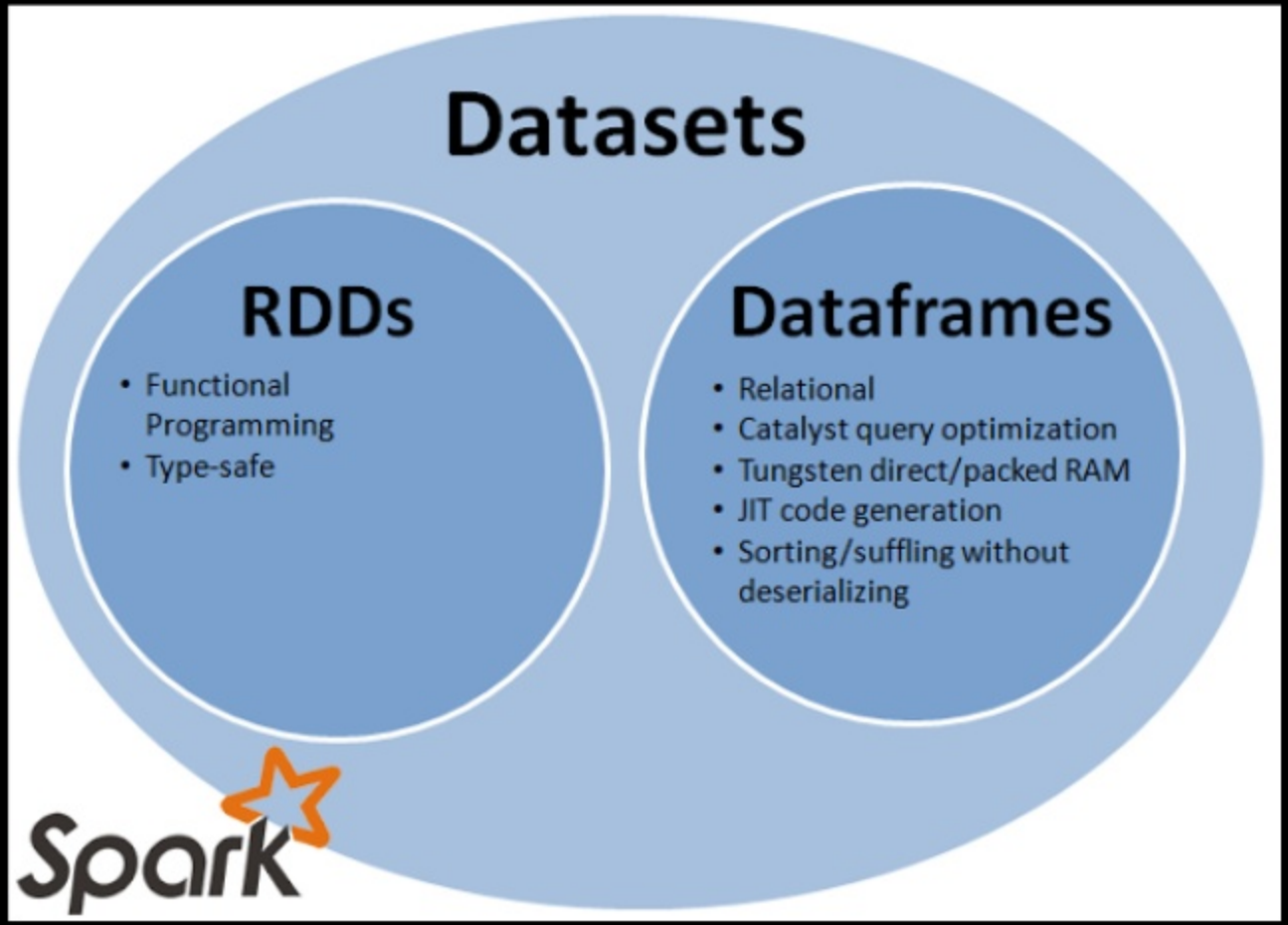
DataFrames & Datasets

Why

- High-level APIs and DSL
- Strong Type-safety
- Ease-of-use & Readability
- ***What-to-do***

When

- Structured Data schema
- Code optimization & performance
- Space efficiency with Tungsten



Source: michaelmalak

A Tale of Three Apache Spark APIs: RDDs, DataFrames, and Datasets

When to use them and why

by Jules Damji
Posted in **ENGINEERING BLOG** | July 14, 2016

https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html

Project Tungsten II

databricks

Project Tungsten

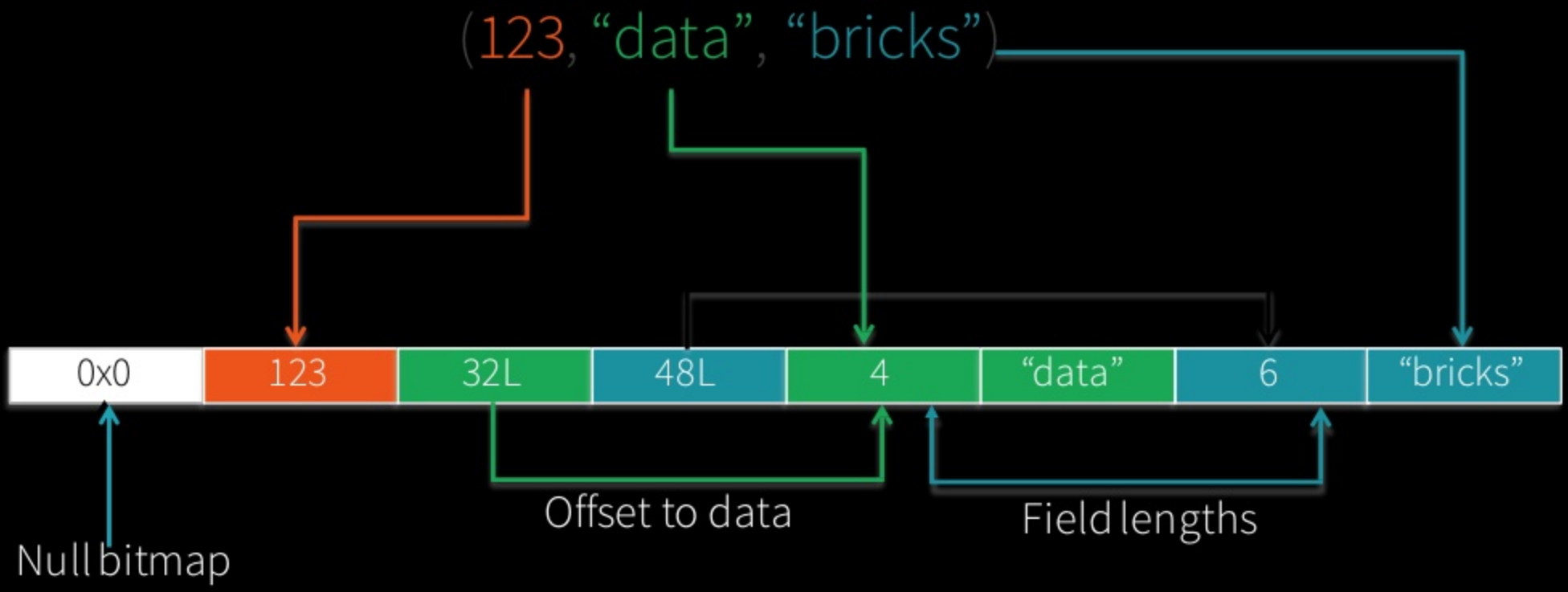
- Substantially speed up execution by optimizing CPU efficiency, via: SPARK-12795

(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

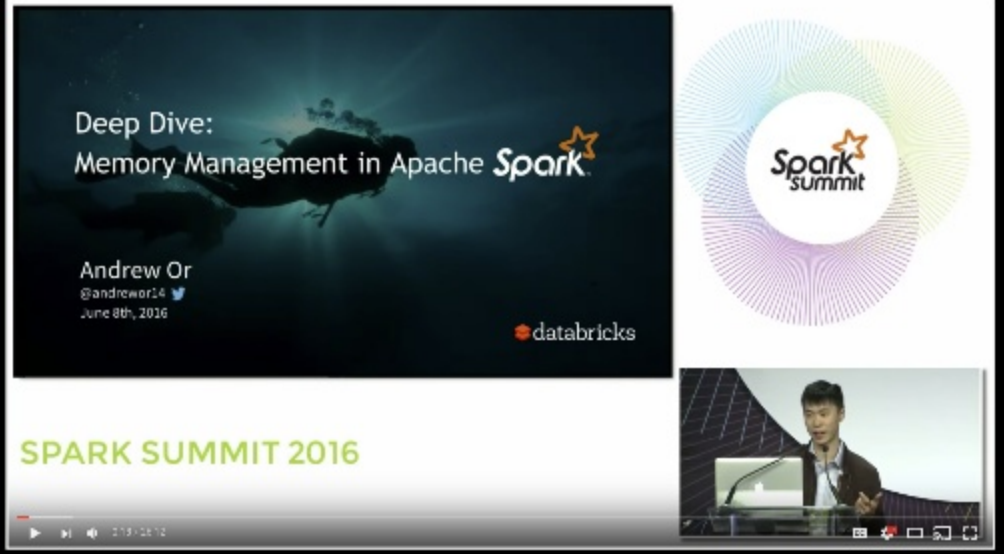
Project Tungsten: Bringing Apache Spark Closer to Bare Metal

by Reynold Xin and Josh Rosen
Posted in **ENGINEERING BLOG** | April 28, 2015

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, however, 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:

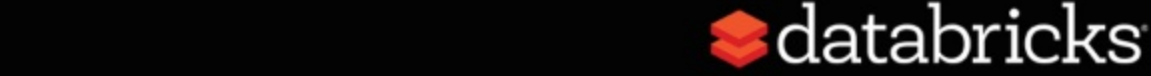
1. *Memory Management and Binary Processing*: leveraging application semantics to manage memory explicitly and eliminate the overhead of JVM object model and garbage collection
2. *Cache-aware computation*: algorithms and data structures to exploit memory hierarchy
3. *Code generation*: using code generation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



Workshop: Notebook on DataFrames/Datasets & Spark SQL

- Import Notebook into your Spark 2.x Cluster
  - **http://dbricks.co/sqlds_wkshp2** (*optional*)
  - **http://dbricks.co/sqldf_wkshp2 (python)** (*optional*)
  - **http://dbricks.co/data_mounts (python)**
  - **http://dbricks.co/iotds_wkshp3**
  - **https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.Dataset**
- Work through each Notebook cell
- Try challenges
- Break..

Introduction to Structured Streaming Concepts



building robust stream processing apps is hard

Complexities in stream processing

COMPLEX DATA

Diverse data formats
(json, avro, binary, …)

Data can be dirty,
late, out-of-order

COMPLEX WORKLOADS

Combining streaming with
interactive queries

Machine learning

COMPLEX SYSTEMS

Diverse storage systems
(Kafka, S3, Kinesis, RDBMS, …)

System failures

Structured Streaming

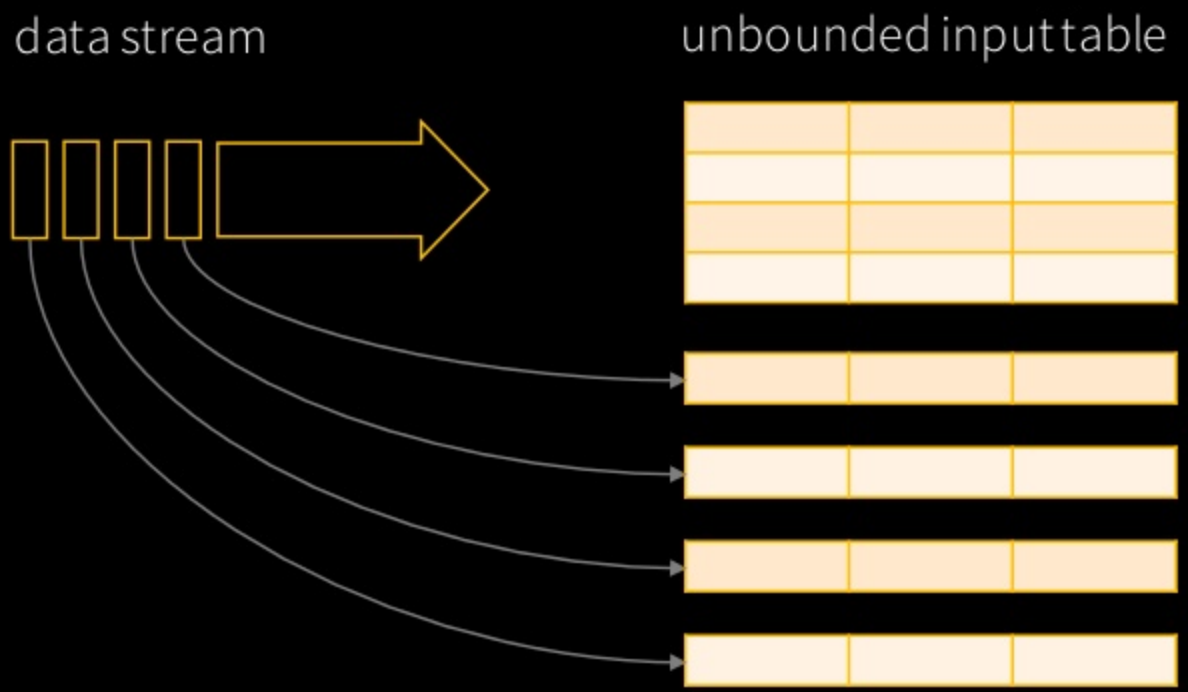
stream processing on Spark SQL engine
fast, scalable, fault-tolerant

rich, unified, high level APIs
deal with *complex data* and *complex workloads*

rich ecosystem of data sources
integrate with many *storage systems*

you
should not have to
reason about streaming

Treat Streams as Unbounded Tables



new data in the
data stream
=
new rows appended
to a unbounded table

you

should write simple queries

&

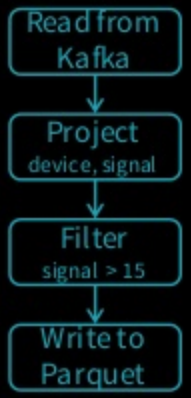
Spark

should continuously update the answer

Spark automatically *streamifies*!

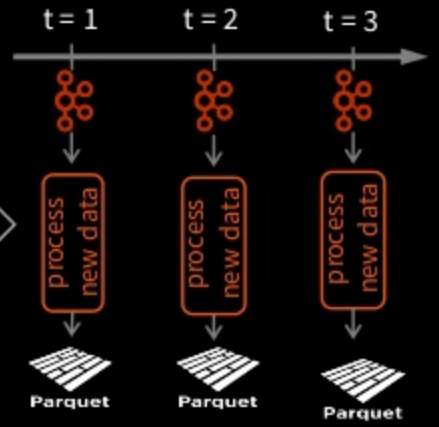
```
input = spark.readStream
  .format("kafka")
  .option("subscribe", "topic")
  .load()

result = input
  .select("device", "signal")
  .where("signal > 15")

result.writeStream
  .format("parquet")
  .start("dest-path")
```

DataFrames, Datasets, SQL



Logical Plan

Optimized Physical Plan



Series of Incremental Execution Plans

Spark SQL converts batch-like query to a series of incremental execution plans operating on new batches of data

Anatomy of a Streaming Query

Streaming word count

Anatomy of a Streaming Query: Step 1

```
spark.readStream
  .format("kafka")
  .option("subscribe", "input")
  .load()
```

Source

- Specify one or more locations to read data from
- Built in support for Files/Kafka/Socket, pluggable.

Anatomy of a Streaming Query: Step 2

```
spark.readStream
  .format("kafka")
  .option("subscribe", "input")
  .load()
   groupBy('value.cast("string") as 'key)
  .agg(count("*") as 'value)
```

Transformation

- Using DataFrames, Datasets and/or SQL.
- Internal processing always exactly-once.

Anatomy of a Streaming Query: Step 3

```
spark.readStream
  .format("kafka")
  .option("subscribe", "input")
  .load()
  .groupBy('value.cast("string") as 'key)
  .agg(count("*") as 'value)
  .writeStream
  .format("kafka")
  .option("topic", "output")

  .outputMode(OutputMode.Complete())
  .option("checkpointLocation", "…")
  .start()
```

Sink

- Accepts the output of each batch.
- When supported sinks are transactional and exactly once (Files).
- Use `foreach` to execute arbitrary code.

Anatomy of a Streaming Query: Output Modes

```
spark.readStream
  .format("kafka")
  .option("subscribe", "input")
  .load()
  .groupBy('value.cast("string") as 'key)
  .agg(count("*") as 'value)
  .writeStream
  .format("kafka")
  .option("topic", "output")
  .trigger("1 minute")
  .outputMode("update")
       ("checkpointLocation", "…")
  .start()
```

Output mode – What's output

- Complete – Output the whole answer every time
- Update – Output changed rows
- Append – Output new rows only

Trigger – When to output

- Specified as a time, eventually supports data size
- No trigger means as fast as possible

Anatomy of a Streaming Query: Checkpoint

```
spark.readStream
  .format("kafka")
  .option("subscribe", "input")
  .load()
  .groupBy('value.cast("string") as 'key)
  .agg(count("*") as 'value)
  .writeStream
  .format("kafka")
  .option("topic", "output")
  .trigger("1 minute")
  .outputMode("update")
  .option("checkpointLocation", "…")
  .start()
```

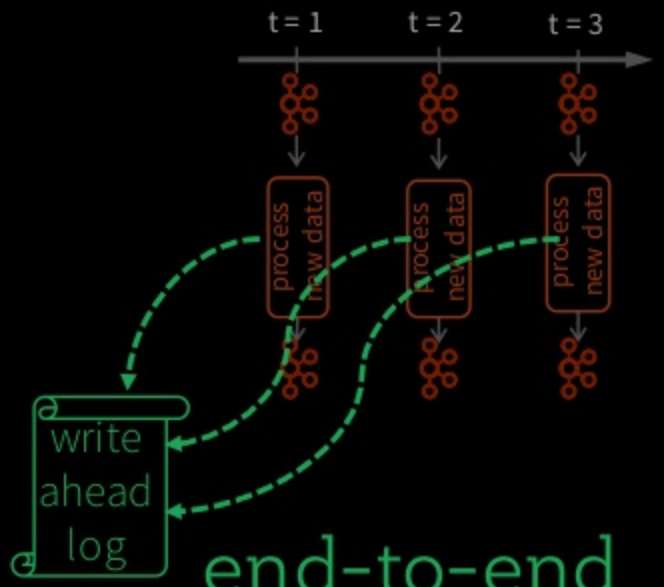
Checkpoint

- Tracks the progress of a query in persistent storage
- Can be used to restart the query if there is a failure.

Fault-tolerance with Checkpointing

Checkpointing – tracks progress (offsets) of consuming data from the source and intermediate state.

- Offsets and metadata saved as JSON
- Can resume after changing your streaming transformations



end-to-end exactly-once guarantees

Complex Streaming ETL

Traditional ETL

Raw, dirty, un/semi-structured is data dumped as files

Periodic jobs run every few hours to convert raw data to structured data ready for further analytics

Traditional ETL

Hours of delay before taking decisions on latest data

Unacceptable when time is of essence
[intrusion detection, anomaly detection, etc.]

Streaming ETL w/ Structured Streaming

Structured Streaming enables raw data to be available as structured data as soon as possible

Streaming ETL w/ Structured Streaming

Example

Json data being received in Kafka

Parse nested json and flatten it

Store in structured Parquet table

Get end-to-end failure guarantees

```
val rawData = spark.readStream
  .format("kafka")
  .option("kafka.boostrap.servers",...)
  .option("subscribe", "topic")
  .load()

val parsedData = rawData
  .selectExpr("cast (value as string) as json"))
  .select(from_json("json", schema).as("data"))
  .select("data.*")

val query = parsedData.writeStream
  .option("checkpointLocation", "/checkpoint")
  .partitionBy("date")
  .format("parquet")
  .start("/parquetTable")
```

Reading from Kafka

Specify options to configure

```
val rawData = spark.readStream
  .format("kafka")
  .option("kafka.boostrap.servers",...)
  .option("subscribe", "topic")
  .load()
```

How?

```
kafka.boostrap.servers => broker1,broker2
```

What?

```
subscribe        =>  topic1,topic2,topic3   // fixed list of topics
subscribePattern =>  topic*                 // dynamic list of topics
assign           =>  {"topicA":[0,1] }      // specific partitions
```

Where?

```
startingOffsets => latest(default) / earliest / {"topicA":{"0":23,"1":345} }
```

Reading from Kafka

rawData dataframe has the following columns

```
val rawDataDF = spark.readStream
  .format("kafka")
  .option("kafka.boostrap.servers",...)
  .option("subscribe", "topic")
  .load()
```

key	value	topic	partition	offset	timestamp
[binary]	*[binary]*	"topicA"	0	345	1486087873
[binary]	*[binary]*	"topicB"	3	2890	1486086721

Transforming Data

Cast binary *value* to string
Name it column *json*

```
val parsedDataDF = rawData
    .selectExpr("cast (value as string) as json")
    .select(from_json("json", schema).as("data"))
    .select("data.*")
```

Transforming Data

Cast binary *value* to string
Name it column *json*

```
val parsedData = rawData
    .selectExpr("cast (value as string) as json")
    .select(from_json("json", schema).as("data"))
    .select("data.*")
```

Parse *json* string and expand into nested columns, name it *data*

Transforming Data

Cast binary *value* to string
Name it column *json*

```
val parsedDataDF = rawData
    .selectExpr("cast (value as string) as json")
    .select(from_json("json", schema).as("data"))
    .select("data.*")
```

Parse *json* string and expand into nested columns, name it *data*

Flatten the nested columns

Transforming Data

Cast binary *value* to string
Name it column *json*

Parse *json* string and expand into nested columns, name it data

Flatten the nested columns

```
val parsedData = rawData
    .selectExpr("cast (value as string) as json")
    .select(from_json("json", schema).as("data"))
    .select("data.*")
```

powerful built-in APIs to perform complex data transformations

from_json, to_json, explode, ...
100s of functions

(see our blog post & tutorial)

Writing to  Parquet

Save parsed data as Parquet table in the given path

Partition files by date so that future queries on time slices of data is fast

e.g. query on last 48 hours of data

```
val query = parsedData.writeStream
      .option("checkpointLocation", ...)
      .partitionBy("date")
      .format("parquet")
      .start("/parquetTable") //pathname
```

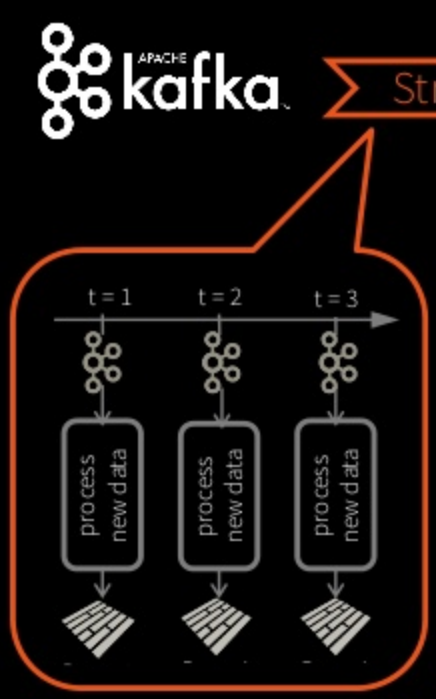
Checkpointing

Enable checkpointing by setting the checkpoint location to save offset logs

`start` actually starts a continuous running StreamingQuery in the Spark cluster

```
val query = parsedData.writeStream
      .option("checkpointLocation", ...)
      .format("parquet")
      .partitionBy("date")
      .start("/parquetTable/")
```

Streaming Query




```
val query = parsedData.writeStream
  .option("checkpointLocation", ...)
  .format("parquet")
  .partitionBy("date")
  .start("/parquetTable")
```

query is a handle to the continuously running StreamingQuery

Used to monitor and manage the execution

Data Consistency on Ad-hoc Queries

Data available for complex, ad-hoc analytics within seconds

Parquet table is updated atomically, ensures *prefix integrity*

Even if distributed, ad-hoc queries will see either all updates from streaming query or none, read more in our blog

https://databricks.com/blog/2016/07/28/structured-streaming-in-apache-spark.html

More Kafka Support [Spark 2.2]

Write out to Kafka

- DataFrame must have binary fields named key and value

```
result.writeStream
  .format("kafka")
  .option("topic", "output")
  .start()
```

Direct, interactive and batch queries on Kafka

- Makes Kafka even more powerful as a storage platform!

```
val df = spark
  .read       // not readStream
  .format("kafka")
  .option("subscribe", "topic")
  .load()

df.createOrReplaceTempView("topicData")
spark.sql("select value from topicData")
```

Amazon Kinesis [Databricks Runtime 3.0]

Configure with options (similar to Kafka)

How?

```
region => us-west-2 / us-east-1 / ...
awsAccessKey (optional) => AKIA...
awsSecretKey (optional) => ...
```

What?

```
streamName => name-of-the-stream
```

Where?

```
initialPosition => latest(default) / earliest / trim_horizon
```

```
spark.readStream
  .format("kinesis")
  .option("streamName”,"myStream")
  .option("region", "us-west-2")
  .option("awsAccessKey", ...)
  .option("awsSecretKey", ...)
  .load()
```

Working With Time

Event Time

Many use cases require aggregate statistics by event time

E.g. what's the #errors in each system in the 1 hour windows?

Many challenges

Extracting event time from data, handling late, out-of-order data

DStream APIs were insufficient for event-time stuff

Event time Aggregations

Windowing is just another type of grouping in Struct. Streaming

number of records every hour

```
parsedData
    .groupBy(window("timestamp","1  hour"))
    .count()
```

avg signal strength of each device every 10 mins

```
parsedData
    .groupBy(
        "device",
        window("timestamp","10  mins"))
    .avg("signal")
```

Support UDAFs!

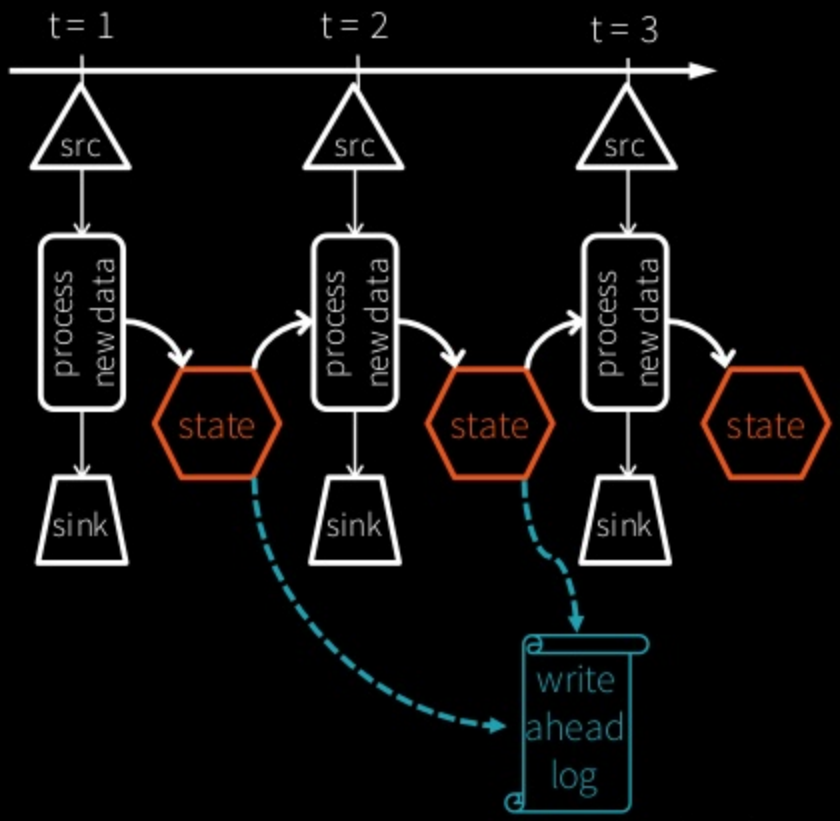
Stateful Processing for Aggregations

Aggregates has to be saved as distributed state between triggers

Each trigger reads previous state and writes updated state

State stored in memory,
backed by *write ahead log* in HDFS/S3

Fault-tolerant, exactly-once guarantee!



Automatically handles Late Data

Keeping state allows late data to update counts of old windows

But size of the state increases indefinitely if old windows are not dropped

red = state updated with late data

Watermarking

Useful only in stateful operations
(streaming aggs, dropDuplicates, mapGroupsWithState, ...)

Ignored in non-stateful streaming queries and batch queries

```
parsedDataDF
      .withWatermark("timestamp",  "10 minutes")
      .groupBy(window("timestamp","5  minutes"))
      .count()
```

What else…

Arbitrary Stateful Operations [Spark 2.2]

mapGroupsWithState allows any user-defined stateful function to a user-defined state

Direct support for per-key timeouts in event-time or processing-time

Supports Scala and Java

```
ds.groupByKey(_.id)
  .mapGroupsWithState
    (timeoutConf)
    (mappingWithStateFunc)


def mappingWithStateFunc(
     key: K,
     values: Iterator[V],
     state: GroupState[S]): U = {
       // update or remove state
       // set timeouts
       // return mapped value
}
```

Arbitrary Stateful Operations [Spark 2.2]

mapGroupsWithState allows any user-defined stateful function to a user-defined state

Direct support for per-key timeouts in event-time or processing-time

Supports Scala and Java

```
ds.groupByKey(_.id)
  .mapGroupsWithState
    (timeoutConf)
    (mappingWithStateFunc)


def mappingWithStateFunc(
     key: K,
     values: Iterator[V],
     state: GroupState[S]): U = {
       // update or remove state
       // set timeouts
       // return mapped value
}
```

Other interesting operations

Streaming Deduplication

- Watermarks to limit state

```
parsedDataDF.dropDuplicates("eventId")
```

Stream-batch Joins

```
val batchDataDF = spark.read
     .format("parquet")
     .load("/additional-data")
     //join with stream DataFrame

parsedDataDF.join(batchData, "device")
```

Stream-stream Joins

- Can use mapGroupsWithState
- Direct support coming soon!

More Info

Structured Streaming Programming Guide

http://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

Databricks blog posts for more focused discussions

https://databricks.com/blog/2016/07/28/structured-streaming-in-apache-spark.html

https://databricks.com/blog/2017/01/19/real-time-streaming-etl-structured-streaming-apache-spark-2-1.html

https://databricks.com/blog/2017/02/23/working-complex-data-formats-structured-streaming-apache-spark-2-1.html

https://databricks.com/blog/2017/04/26/processing-data-in-apache-kafka-with-structured-streaming-in-apache-spark-2-2.html

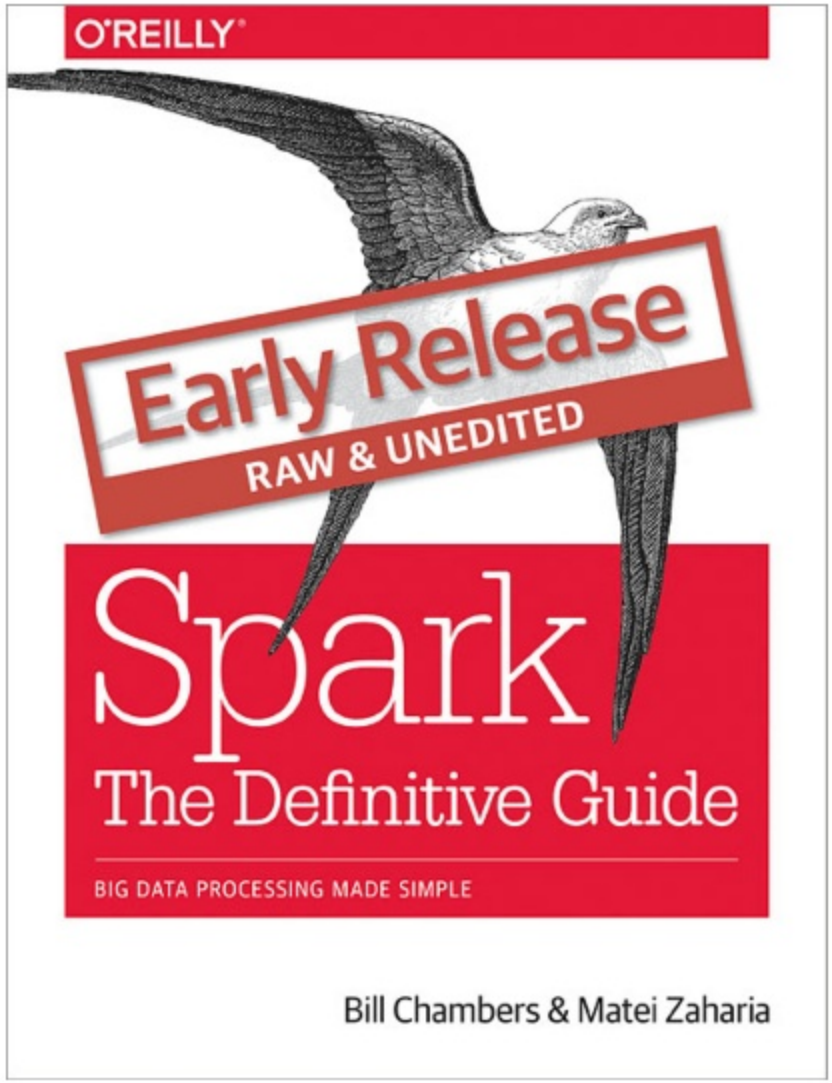
https://databricks.com/blog/2017/05/08/event-time-aggregation-watermarking-apache-sparks-structured-streaming.html

and more to come, stay tuned!!

Resources

- Getting Started Guide with Apache Spark on Databricks
- docs.databricks.com
- Spark Programming Guide
- Structured Streaming Programming Guide
- Databricks Engineering Blogs
- sparkhub.databricks.com
- spark-packages.org

https://spark-summit.org/eu-2017/



source: O'Reilly

Sharing Knowledge with the Community in a Preview of Apache Spark: The Definitive Guide

by Bill Chambers and Matei Zaharia
Posted in **COMPANY BLOG** | June 5, 2017

http://dbricks.co/2sK35XT

databricks

Do you have any questions for my prepared answers?

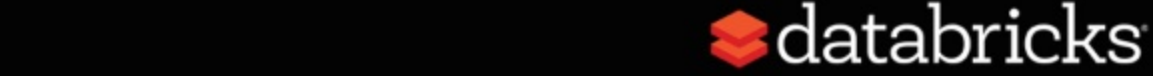
Demo & Workshop: Structured Streaming

- Import Notebook into your Spark 2.2 Cluster
  - http://dbricks.co/iotss_wkshp4
- Done!

Title goes here.
It can be one or two lines.

Author goes here

Date goes here



Here is a basic slide

Suspendisse ullamcorper vel odio a varius

- Pellentesque habitant morbi tristiqu
- enectus et netus et malesuada fames ac turpis egestas
- ut erat dapibus lobortis purus sed gravida augu
- efficitur a risus placerat porta nullam molestie malesuada velit et auctor

Here are some logos

Here is a comparison slide

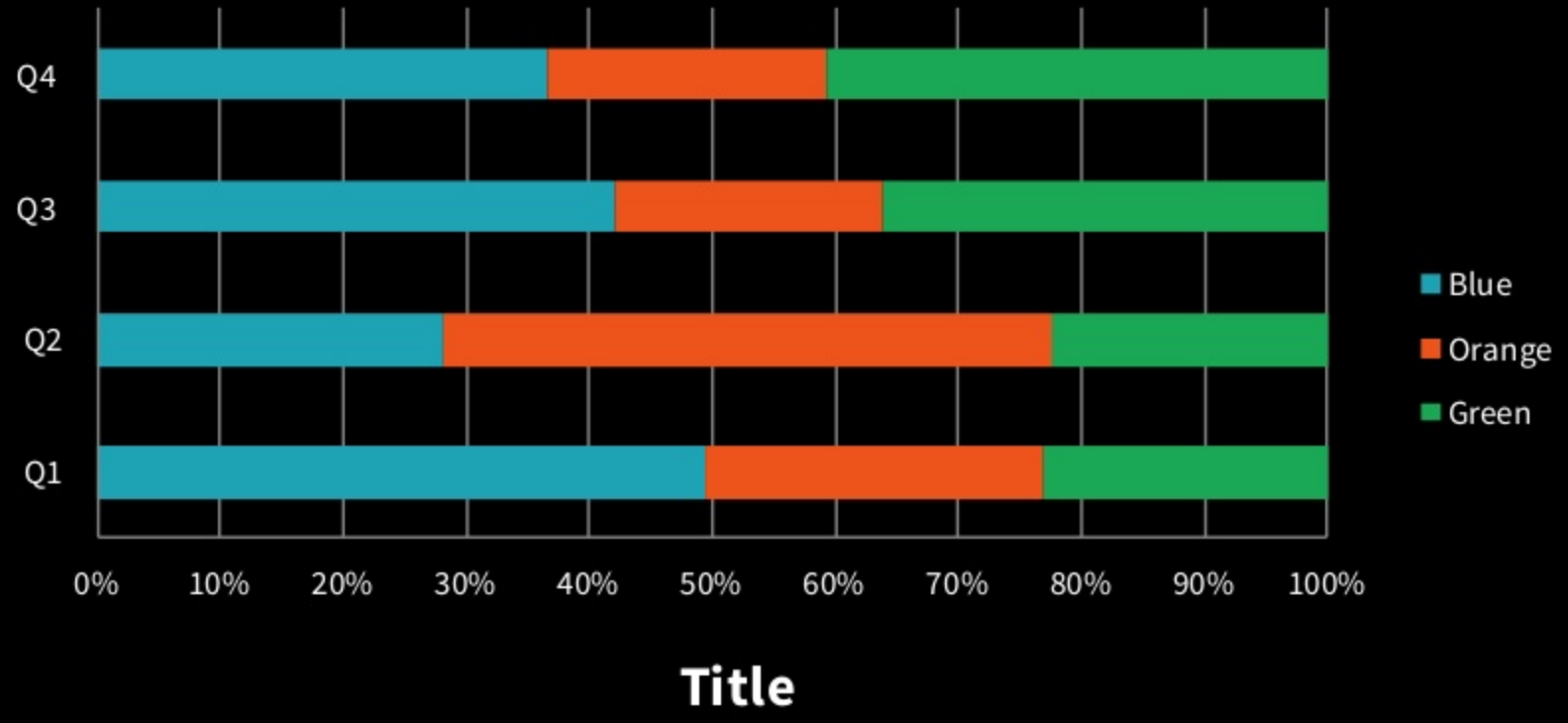
HEADER CAN BE BOLD

- Aliquam purus leo, interdum eu urna vitae
- Etiam in arcu gravida, tincidunt magna ve faucibus
- Donec laoreet vel quam eu condimentum

ALL CAPS LIKE THIS

- Quisque tortor quam, posuere sed sagittis et, iaculis a urna. In malesuada in orci ut lacinia
- Sed bibendum sed mauris egestas pellentesque
- Vestibulum bibendum sagittis odio quis tincidunt augue consequat e

Use this chart to start



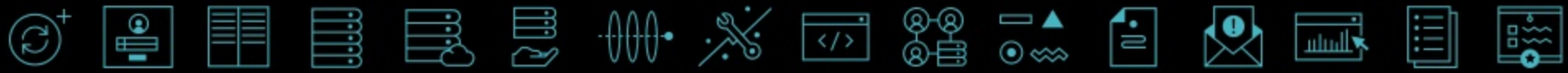
Here are some icons to use - scalable

Icons can be recolored within Powerpoint — see: format picture / picture color / recolor

Orange, Green, and Black versions (no recoloration necessary) can be found in go/icons

DB Benefits

DB Features



General / Data Science

More icons

Spark Benefits

Spark Features

Security

Industries

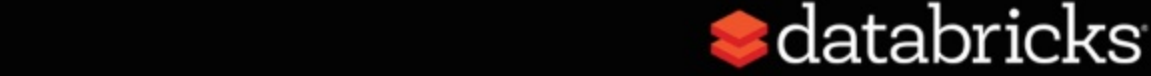
Even more

Misc

Slide for Large Question or Section Headers

Thank You

Parting words or contact information go here.



The Unified Analytics Platform

PEOPLE → databricks → APPLICATIONS

Data Science

Data Engineering

Line of Business

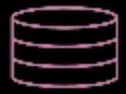
DATABRICKS WORKSPACE | DATABRICKS WORKFLOWS

DATABRICKS RUNTIME

DATABRICKS I/O (DBIO) | APACHE Spark | DATABRICKS SERVERLESS

DATABRICKS ENTERPRISE SECURITY (DBES)

Cloud Storage



Data Warehouses

Hadoop Storage

Deep Learning / ML

Streaming

Data Warehousing

and many others…

databricks