Jump Start into Apache Spark 2.0

Denny Lee,
Technology Evangelist
denny@databricks.com, @dennylee

About Me: Denny Lee

Technology Evangelist, Databricks (Working with Spark since v0.5)

Former:

- Senior Director of Data Sciences Engineering at Concur (now part of SAP)
- Principal Program Manager at Microsoft

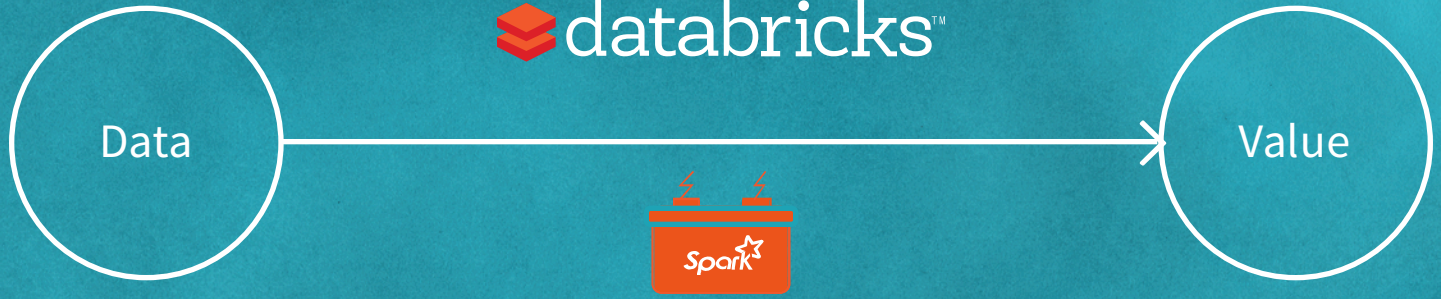
Hands-on Data Engineer, Architect more than 15y developer internet-scale infrastructure for both on-premises and cloud including Bing's Audience Insights, Yahoo's 24TB SSAS cube, and Isotope Incubation Team (HDInsight)

We are Databricks, the company behind 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.**

So if Spark were the Justice League...

SPARK CORE API
(R, SQL, Python, Scala, Java)

SPARK SQL + DATAFRAMES

SPARK STREAMING

MLLIB

GRAPHX

Spark

LEARN MORE HERE:
https://databricks.com/blog/2016/01/05/spark-2015-year-in-review.html

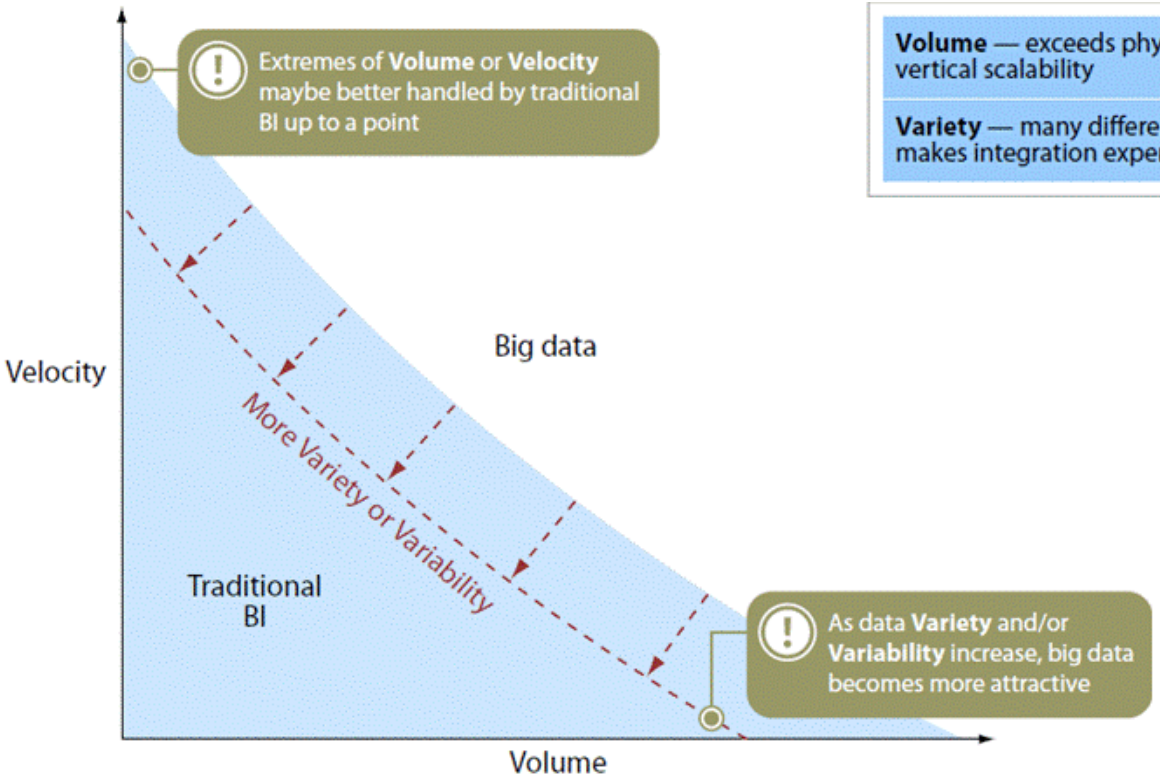
MOST IMPORTANT ASPECTS OF SPARK

91% Performance	77% Ease of programming
71% Ease of deployment	64% Advanced analytics
52% Real-time streaming	

Source: Databricks Spark Survey Result 2015

Big Data Primer

Big Data: The 4Vs



Volume — exceeds physical limits of vertical scalability	**Velocity** — decision window small compared to data change rate
Variety — many different formats makes integration expensive	**Variability** — many options or variable interpretations confound analysis

Big Data: techniques and technologies that make handling data at extreme scale economical

Source: Big Data, Brewer, And A Couple Of Webinars http://bit.ly/1rl4gKR

Scale Up

With the power of the Hubble telescope, we can take amazing pictures 45M light years away

Amazing image of the Antennae Galaxies (NGC 4038-4039)

Analogous with scale up:

- non-commodity
- specialized equipment
- single point of failure*

Scale Out › Distribution

Hubble can provide an amazing view Giant Galactic Nebula (NGC 3503) but how about radio waves?

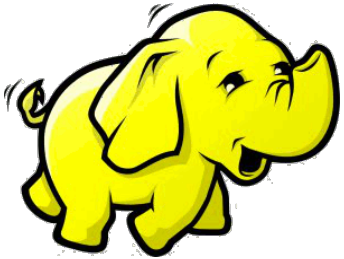
- Not just from one area but from all areas viewed by observatories
- SETI @ Home: 5.2M participants, 10^{21} floating point operations[1], 769 teraFLOPS[2]

Analogous with commoditized distributed computing

- Distributed and calculated locally
- Engage with hundreds, thousands, + machines
- Many points of failure, auto-replication prevents this from being a problem

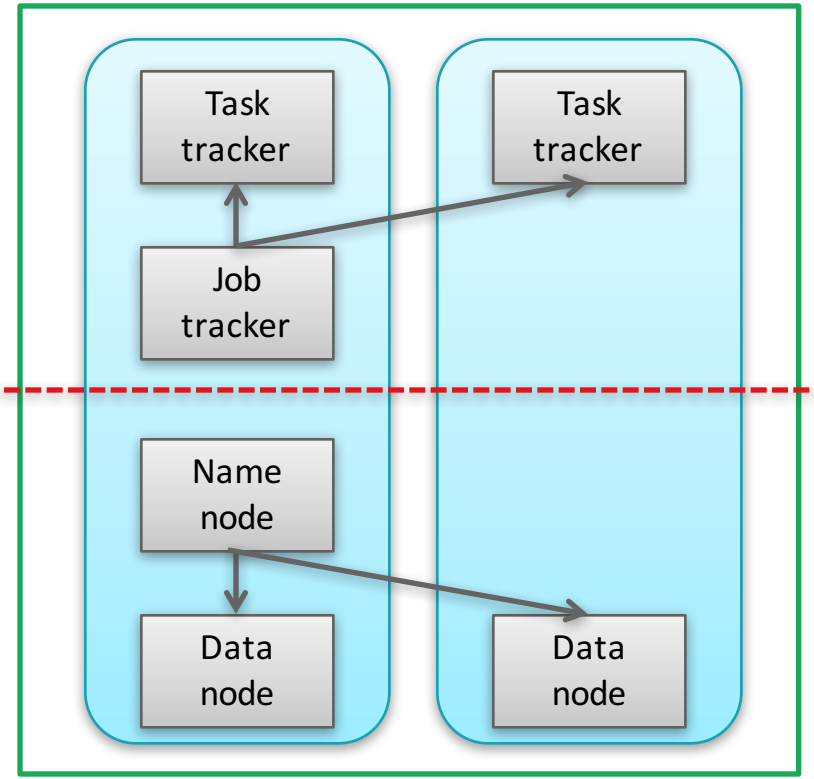
databricks

What is Hadoop?



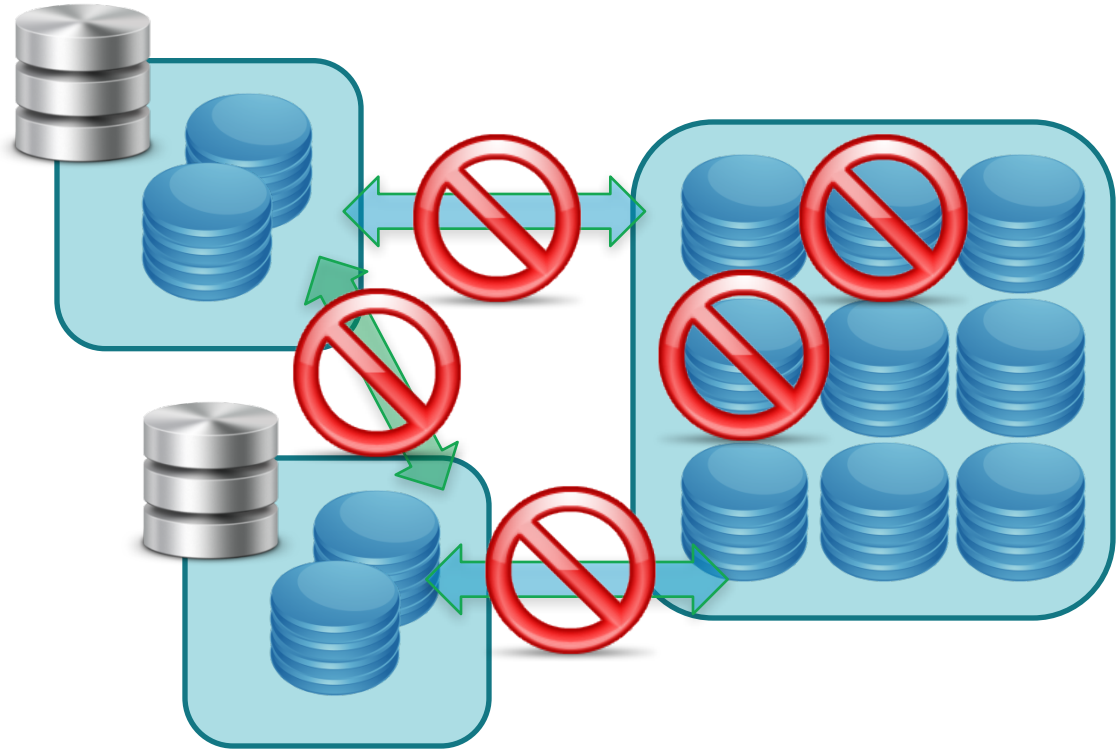
Map Reduce Layer

HDFS Layer

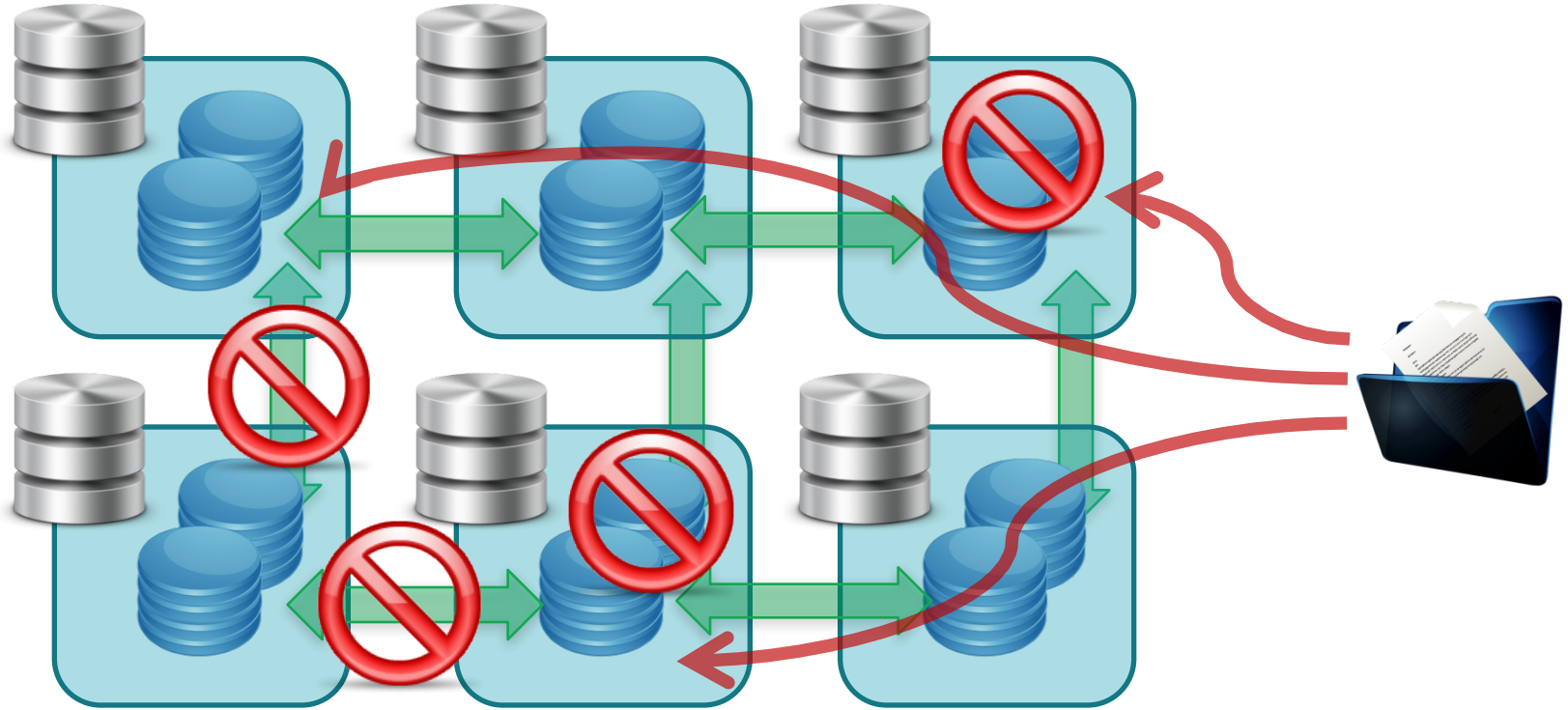


Reference: http://en.wikipedia.org/wiki/File:Hadoop_1.png

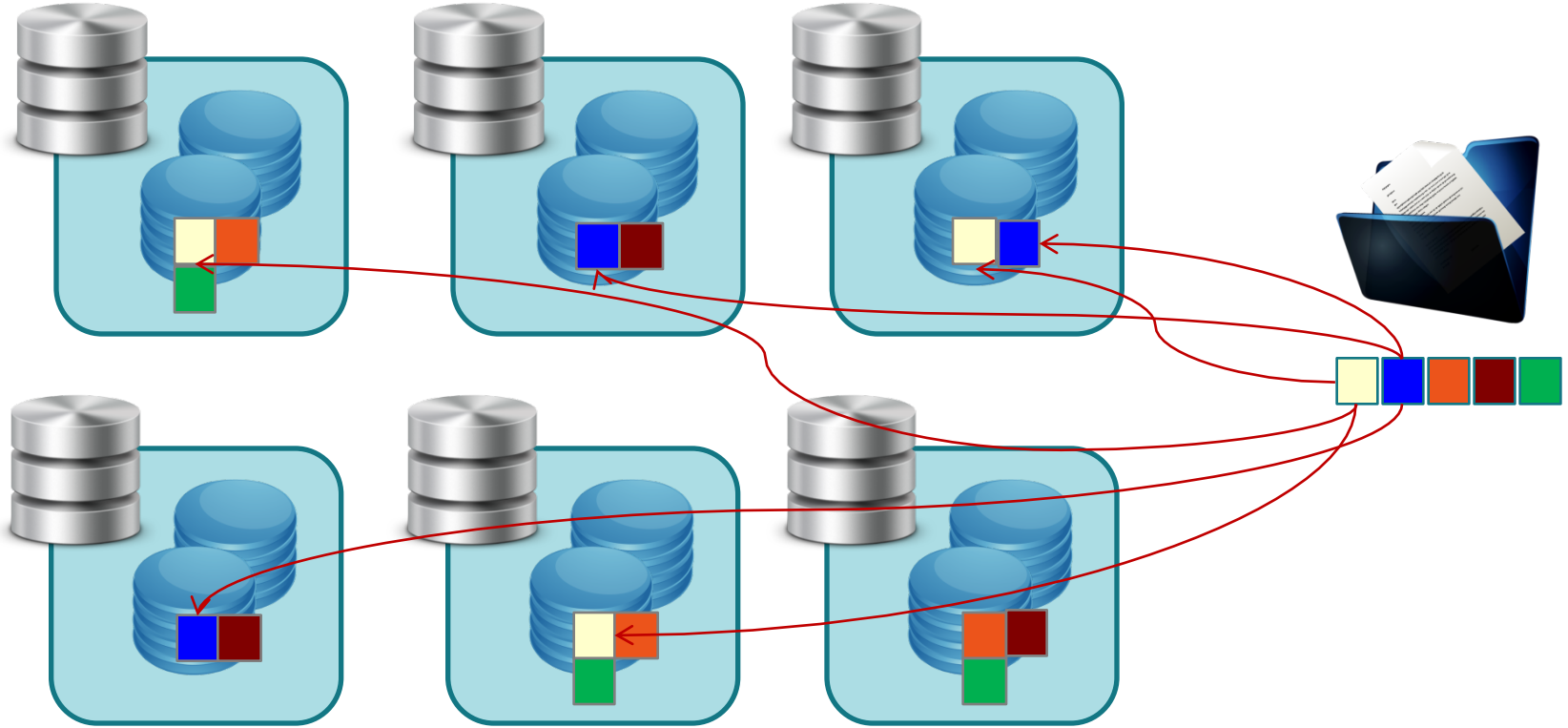
Traditional RDBMS: Move Data to Compute



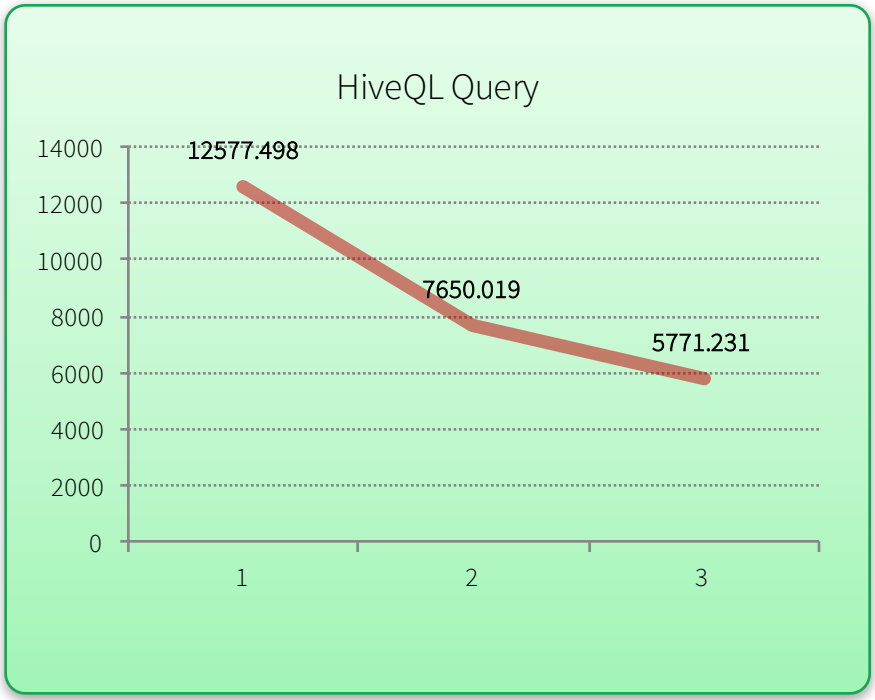
Hadoop: Move Compute to the Data

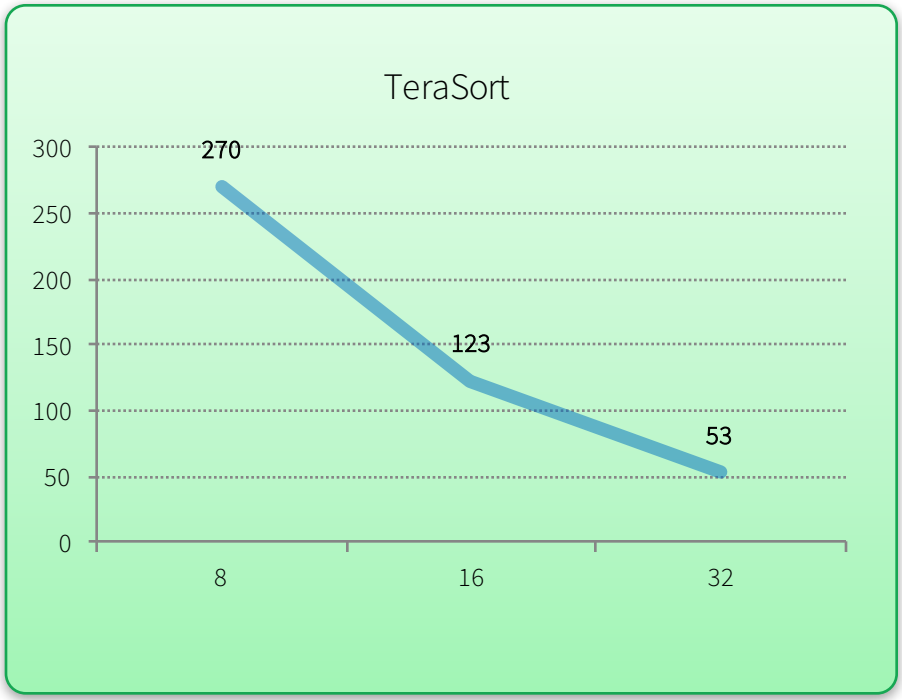


Auto-replication



Distribution, Linear Scalability





Comparing RDBMS and MapReduce

	Traditional RDBMS	MapReduce
Data Size	Gigabytes (Terabytes)	Petabytes (Exabytes)
Access	Interactive and Batch	Batch
Updates	Read / Write many times	Write once, Read many times
Structure	Static Schema	Dynamic Schema
Integrity	High (ACID)	Low (BASE)
Scaling	Nonlinear	Linear
DBA Ratio	1:40	1:3000

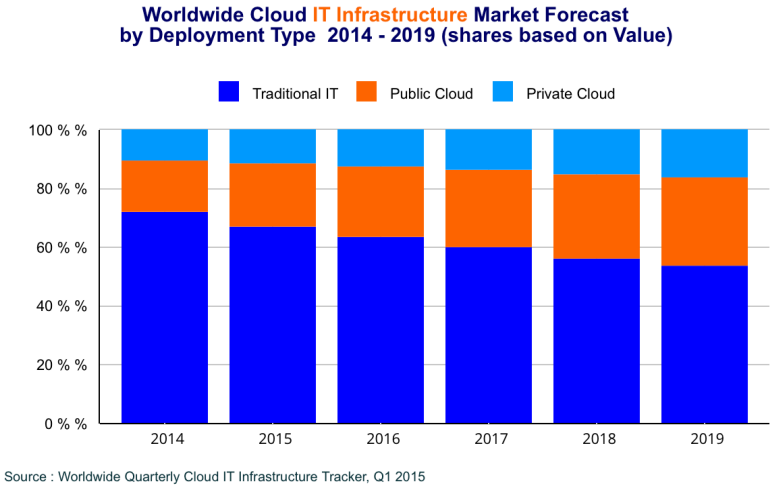
Reference: Tom White's Hadoop: The Definitive Guide

This illustrates a new thesis or collective wisdom emerging from the Valley: If a technology is not your core value-add, it should be open-sourced because then others can improve it, and potential future employees can learn it. This rising tide has lifted all boats, and is just getting started.

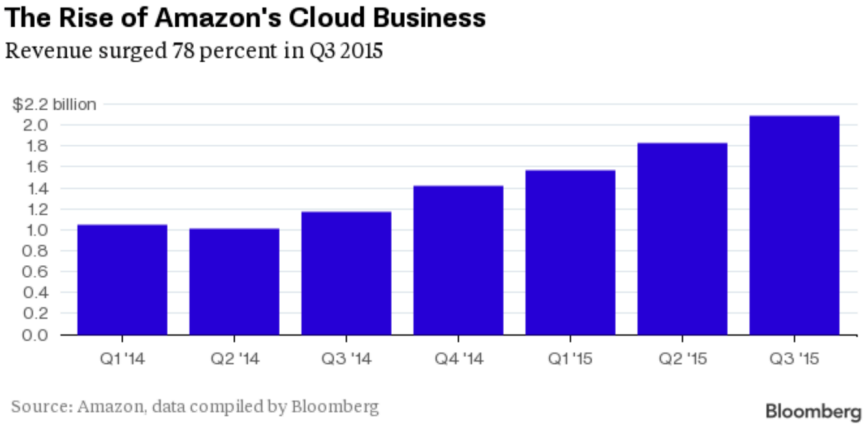
Kovas Boguta: Hadoop & Startups: Where Open Source Meets Business Data.

Jump into the Cloud

Enterprise Growth in Cloud



Source : Worldwide Quarterly Cloud IT Infrastructure Tracker, Q1 2015



Source: IDC Cloud IT Infrastructure Spending Forecast to Grow 26% YOY in 2015 http://bit.ly/1JJIskN

Source: The Cloud is Raining Cash on Amazon, Google, and Microsoft http://bloom.bg/1Mdzw9e

databricks

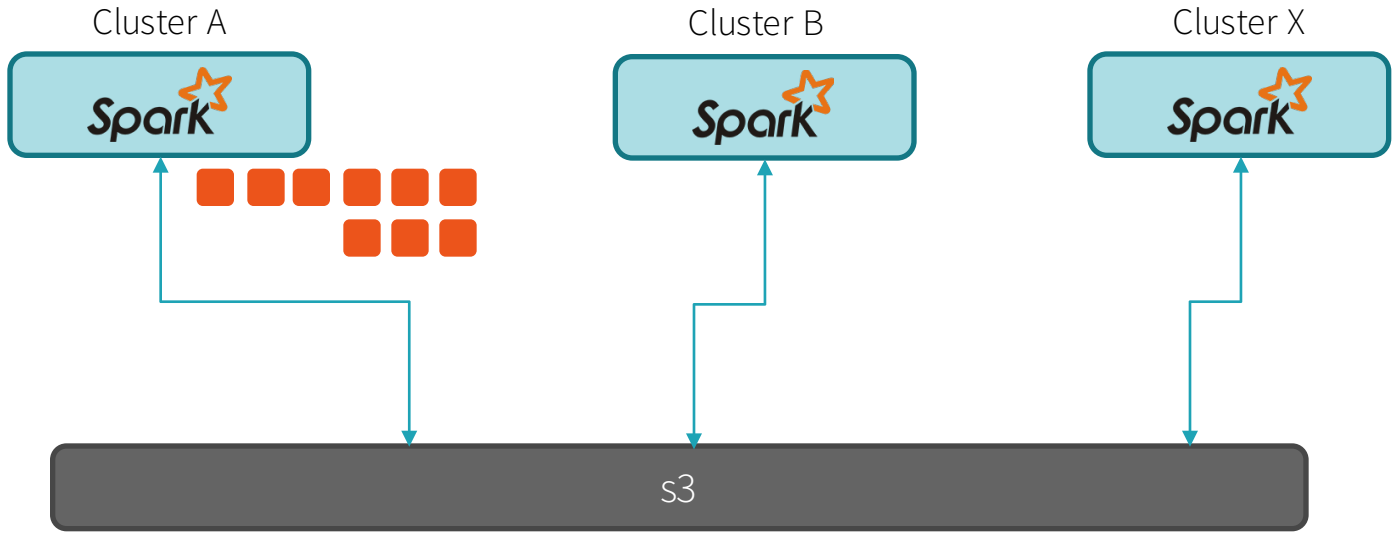
Transitioning to the Cloud

Beth Israel Deaconess Medical Center is increasingly moving to cloud infrastructure services with the hopes of closing its data center when the hospital's lease is up in the next five years. ***CIO John Halamka*** *says he's decommissioning HP and Dell servers as he moves more of his compute workloads to Amazon Web Services, where he's currently using 30 virtual machines to test and develop new applications. "It is no longer cost effective to deal with server hosting ourselves because our challenge isn't real estate, it's power and cooling," he says.*

Why Jump into the Cloud?

- On-demand nodes clusters
  - Spin up and down clusters when individuals or teams need them
  - Scale out (and back in) clusters when resources are needed
- Combination of simple UIs, CLIs, and REST API calls to manage nodes
- Elasticity means having resources when you need them, not paying for resources when you do not
- Cost Effective, Faster Time to Market, and Scalability

Cloud Scenarios: Spark on S3

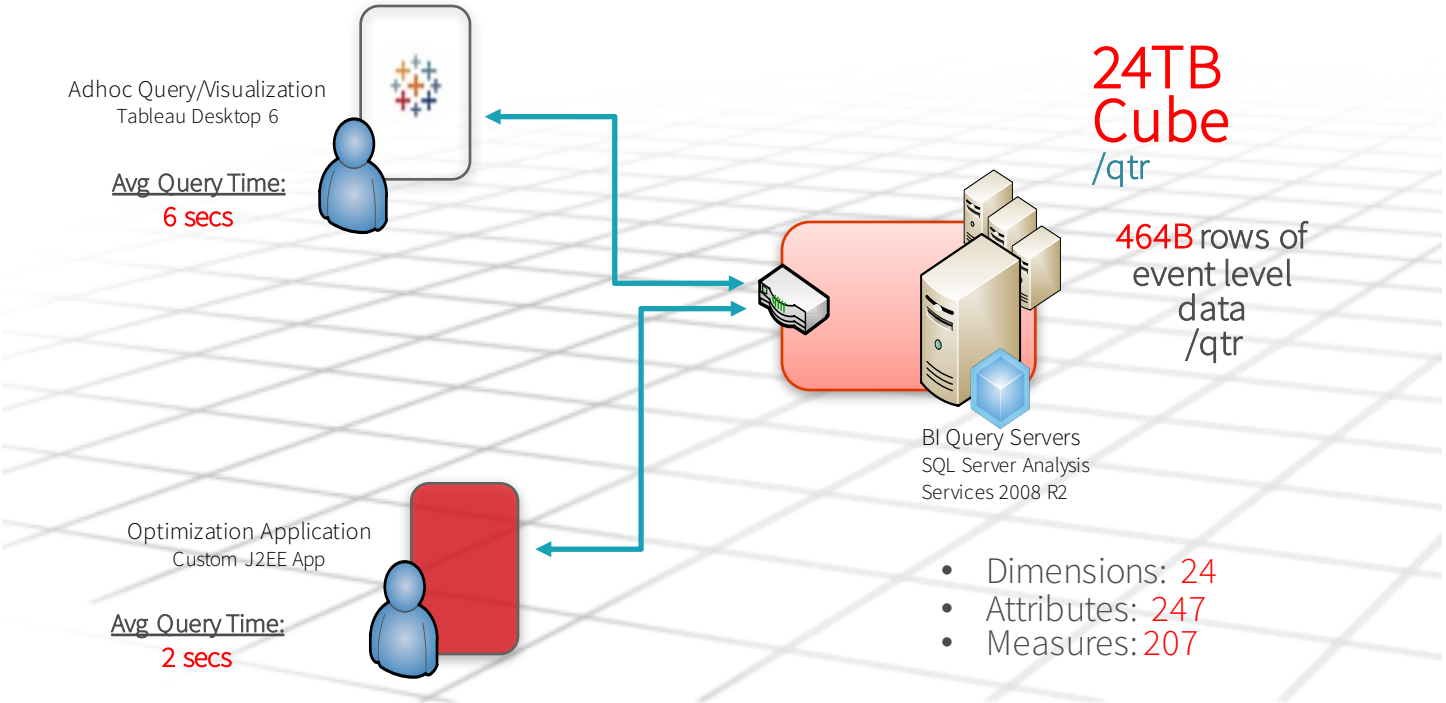


My Motivation to Big Data and Spark

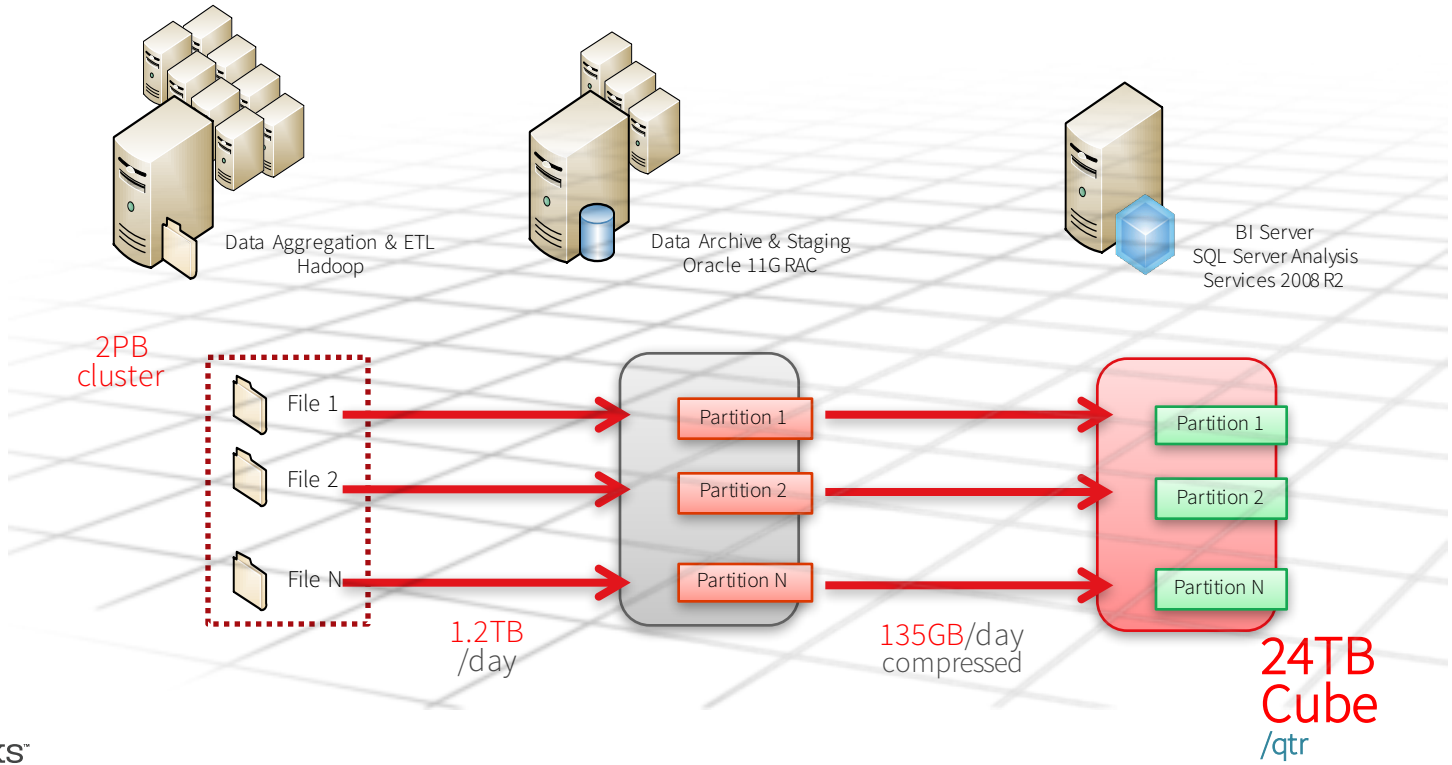
Motivation: Building Yahoo!'s 24TB SSAS Cube

Visitors to Yahoo! Branded sites:	680,000,000
Ad Impressions:	3,500,000,000 (per day)
Rows Loaded:	464,000,000,000 (per qtr)
Refresh Frequency:	Hourly
Average Query Time:	<10 seconds

Motivation: Queries at the speed of thought



Motivation: Movement of Data

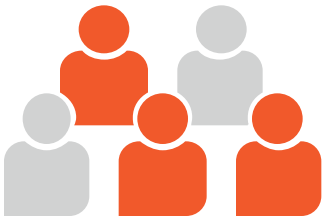


Spark Community Growth

- Spark Survey 2015 Highlights
- End of Year Spark Highlights

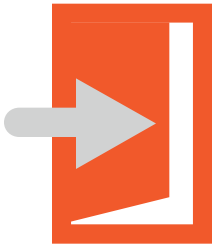
Spark Survey Report 2015 Highlights

TOP 3 APACHE SPARK TAKEAWAYS



Spark adoption is growing rapidly

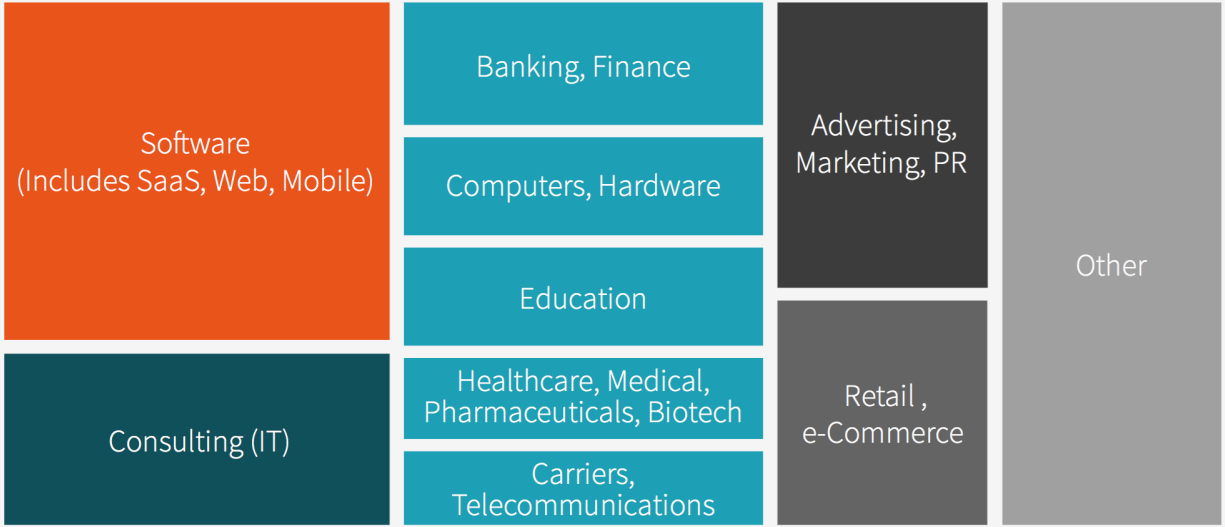
Spark use is growing beyond Hadoop



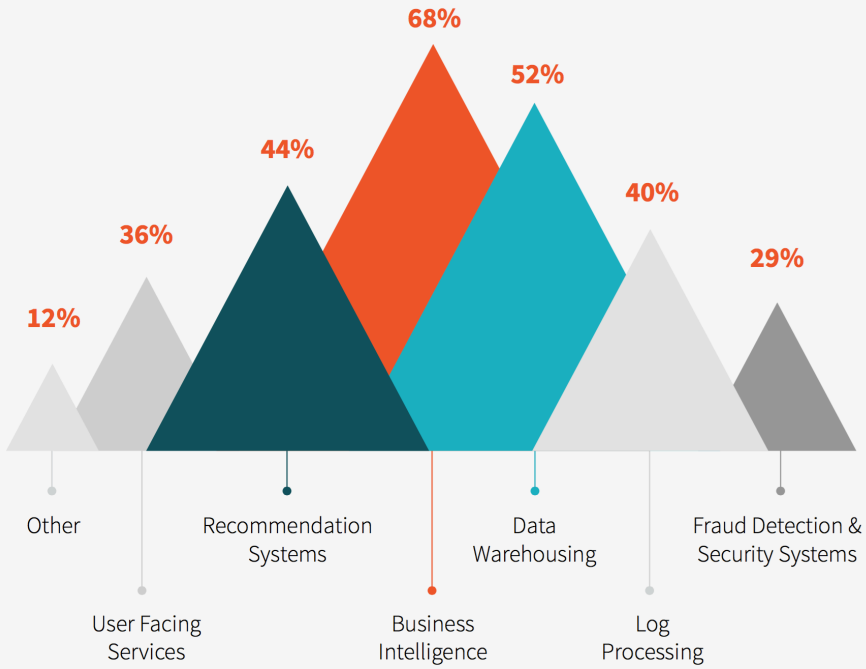
Spark is increasing access to big data

As of 2015 EOY: 1000 total contributors

TOP 10 INDUSTRIES USING SPARK



SPARK IS USED TO CREATE MANY TYPES OF PRODUCTS INSIDE OF DIFFERENT ORGANIZATIONS



MOST COMMON SPARK DEPLOYMENT ENVIRONMENTS (CLUSTER MANAGERS)

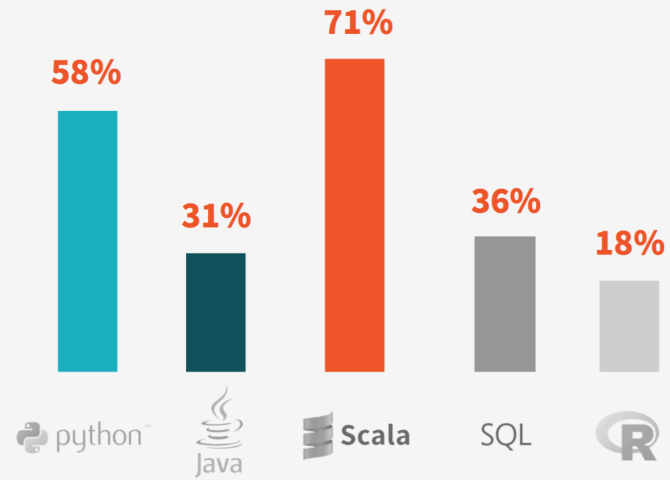
48% Standalone mode

40% YARN

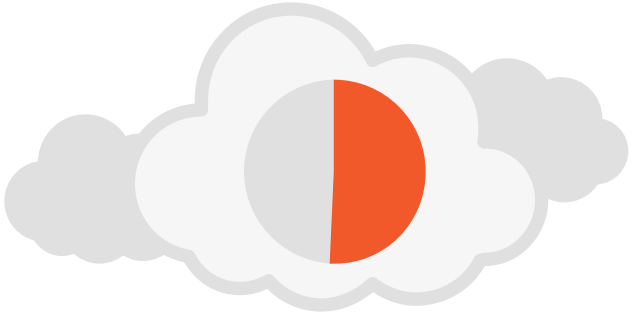
11% Mesos

PROGRAMMING LANGUAGES USED WITH SPARK

Survey respondents can choose multiple languages.



HOW RESPONDENTS ARE RUNNING SPARK



51%

on a public cloud

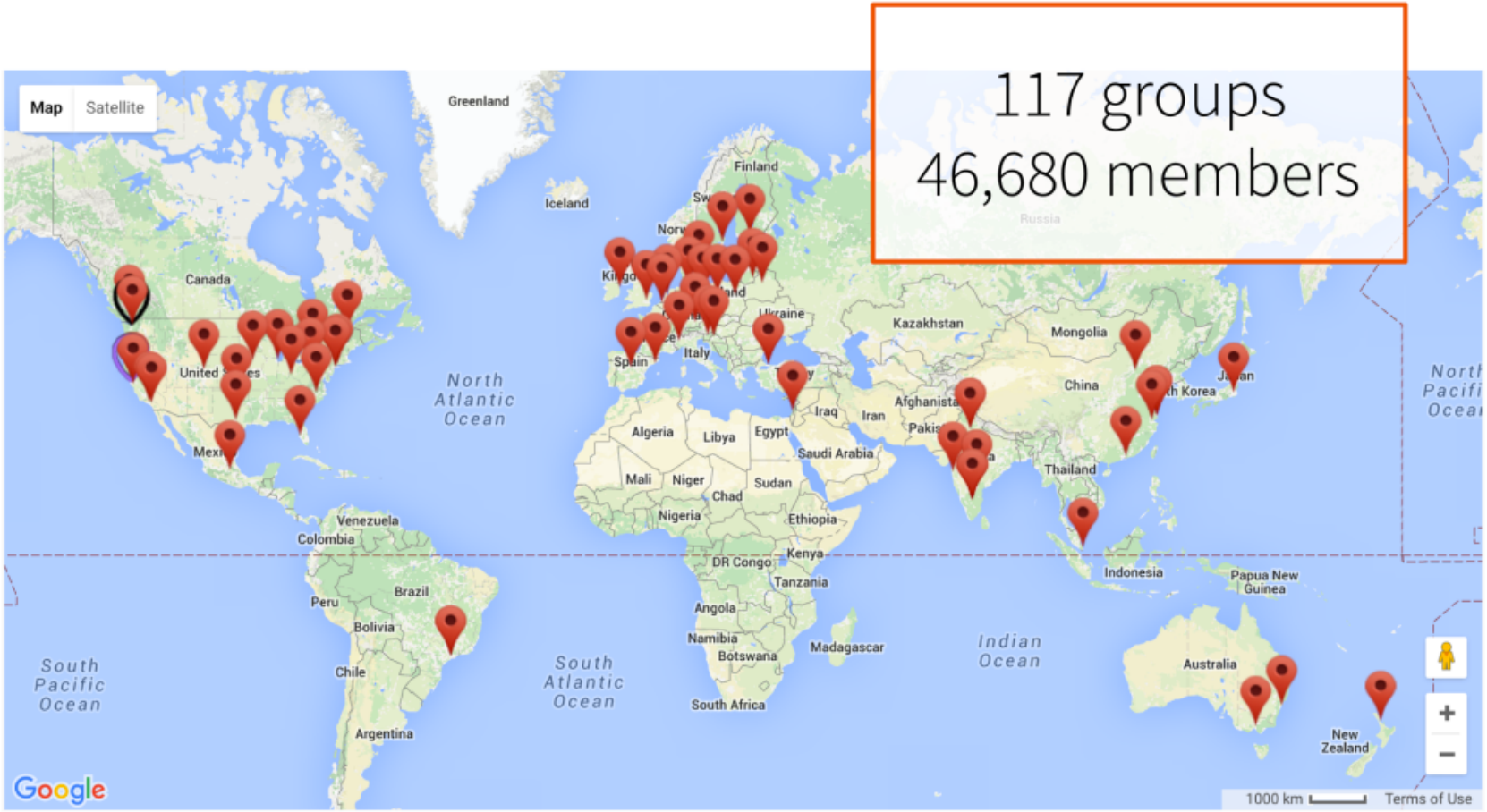
TOP ROLES USING SPARK

41%

of respondents identify themselves as Data Engineers

22%

of respondents identify themselves as Data Scientists



*As of November 19th, 2015

Spark Customer Highlights

NOTABLE USERS THAT PRESENTED AT SPARK SUMMIT 2015 SAN FRANCISCO

Source: Slide 5 of Spark Community Update

Large-Scale Usage

Largest cluster:
8000 Nodes (Tencent)

Largest single job:
1 PB (Alibaba, Databricks)

Top Streaming Intake:
1 TB/hour (HHMI Janelia Farm)

2014 On-Disk Sort Record
Fastest Open Source Engine for sorting a PB

Goldman Sachs

Source: How Spark is Making an Impact at Goldman Sachs

- Started with Hadoop, RDBMS, Hive, HBASE, PIG, Java MR, etc.
- Challenges: Java, Debugging PIG, Code-Compile-Deploy-Debug
- Solution: Spark
  - Language Support: Scala, Java, Python, R
  - In-Memory: Faster than other solutions
  - SQL, Stream Processing, ML, Graph
  - Both Batch and stream processing

Source: Big Telco Real-time Network Analysis

- 250 TB/day, 1400+ node Hadoop cluster migrated from MPP RDBMS
- Use cases:
  - Real-time Analytics of Base Stations
  - Network Enterprise DW
- Why Spark:
  - Event Stream Processing, Fast Queries for Large Datasets (incl RT), Improved Programmer Productivity

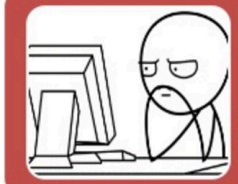
Source: Perspectives on Big Data & Analytics

- Analytics for Government
  - To deliver the most advanced analytic environment in support of the mission
- Government is Changing:
  - Adoption of C2S cloud in the Intelligence community
- Spark Advantage:
  - Fast – Streaming, Distributed, Interoperable, Learning

- Scale:
  - > 1000 nodes (20,000 cores, 100TB RAM)
- Daily Jobs: 2000-3000
- Supports: Ads, Search, Map, Commerce, etc.
- Cool project: Enabling Interactive Queries with Spark and Tachyon
  - >50X acceleration of Big Data Analytics workloads

Baidu 百度

Need for Interaction (Speed)



Problem

- John is a PM and he needs to keep track of the top queries submitted to Baidu everyday
- Based on the top queries of the day, he will perform additional analysis
- But John is very frustrated that each query takes ***tens of minutes***

Requirements for Baidu Interactive Query Engine

- Manages Petabytes of data
- Able to finish 95% of queries within ***30 seconds***

Baidu 百度

databricks

Office 365 Customer Fabric

Source: Spark and Cassandra: An Amazing Apache Love Story

- 10TB of high frequency event data daily
- Constantly increasing volume
- More info:
  - Using Spark to Power the Office 365 Delve Organization Analytics (Spark Summit East 2016)
  - A Primer into Jupyter, Spark on HDInsight, and Office 365 Analytics with Spark (Seattle Spark Meetup, March 9th 2016 @ MS City Center)

Machine Learning: What and Why?

ML uses data to identify patterns and make decisions.

Core value of ML is automated decision making

- Especially important when dealing with TB or PB of data

Many Use Cases including:

- Marketing and advertising optimization
- Security monitoring / fraud detection
- Operational optimizations

Why Spark MLlib

Provide general purpose ML algorithms on top of Spark

- Let Spark handle the distribution of data and queries; scalability
- Leverage its improvements (e.g. DataFrames, Datasets, Tungsten)

Advantages of MLlib's Design:

- Simplicity
- Scalability
- Streamlined end-to-end
- Compatibility

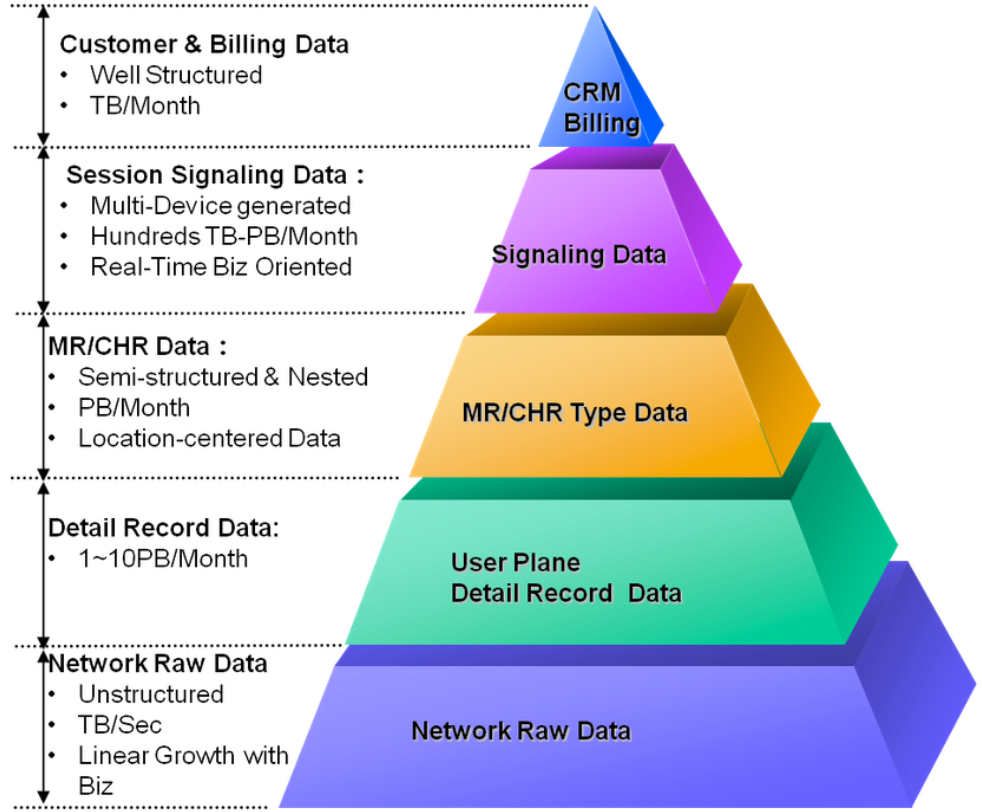
Machine Learning Highlights

Source: [Why you should use Spark for Machine Learning]

HUAWEI

Why Huawei chose Spark:

- Scale-out, parallel data flow model efficiently supporting diverse workloads on the same execution engine
- Open framework can support diverse complex data sources in a consistent way. It significantly shortens the lifecycle of application development and onsite deployment.



Note：In a typical network of ~30M subscribers and the data flow around 1TB/Sec.
MR/CHR: Measurement Report/Call History Data generated during calls

databricks

Source: Toyota Customer 360 Insights on Apache Spark and MLlib

- Performance
  - Original batch job: 160 hours
  - Same Job re-written using Apache Spark: 4 hours
- Categorize
  - Prioritize incoming social media in real-time using Spark MLlib (differentiate campaign, feedback, product feedback, and noise)
  - ML life cycle: Extract features and train:
    - V1: 56% Accuracy -> V9: 82% Accuracy
    - Remove False Positives and Semantic Analysis (distance similarity between concepts)

What is Spark Streaming?

Spark Streaming

Scalable, fault-tolerant stream processing system

High-level API

joins, windows, …
often 5x less code

Fault-tolerant

Exactly-once semantics,
even for stateful ops

Integration

Integrate with MLlib, SQL,
DataFrames, GraphX

What can you use it for?

Real-time fraud detection in transactions

React to anomalies in sensors in real-time

Cat videos in tweets as soon as they go viral

Streaming Highlights

Use Cases

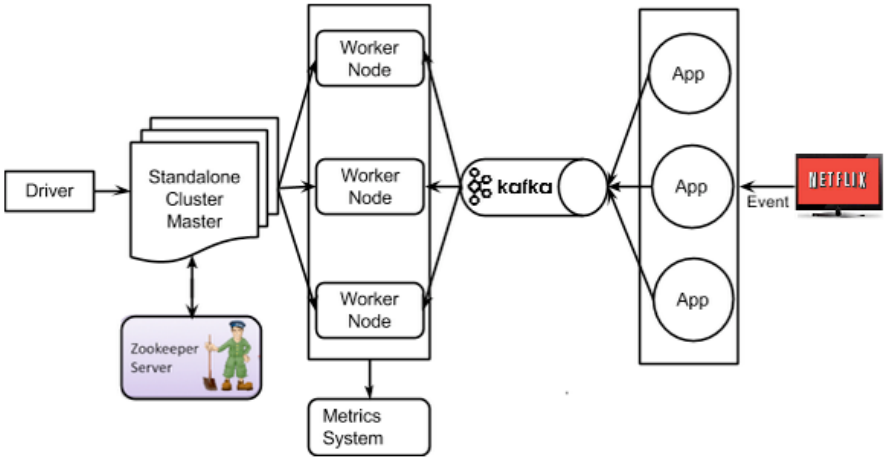
- Streaming ETL
- Triggers
- Data Enrichment
- Complex sessions and continuous learning

Source: Spark Streaming: What Is It and Who's Using It?

NETFLIX

Source: Can Spark Streaming Survive Chaos Monkey

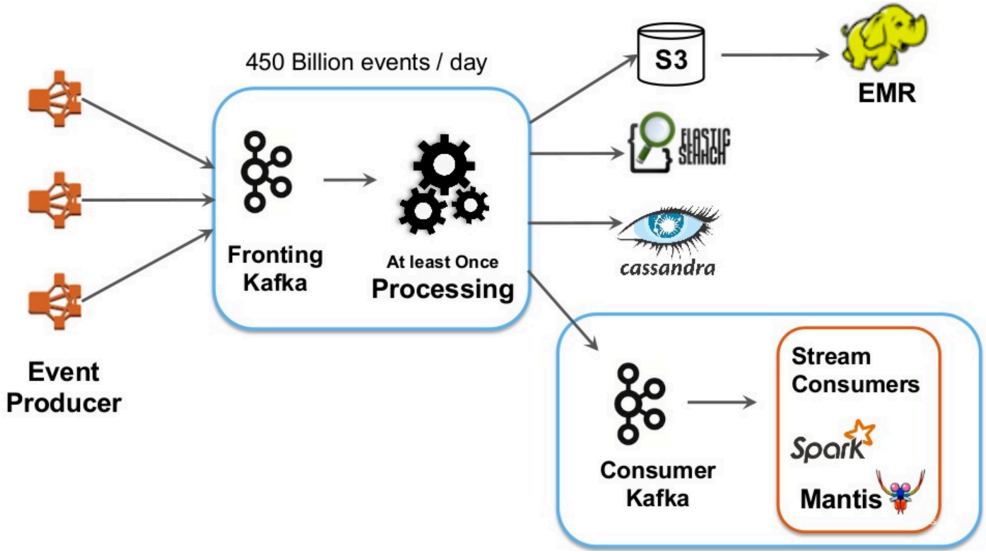
"Overall, we are happy with the resiliency of spark standalone for our use cases and excited to take it to the next level where we are working towards building a unified Lambda Architecture that involves a combination of batch and real-time streaming processing.



NETFLIX

Source: Spark and Spark Streaming at Netflix

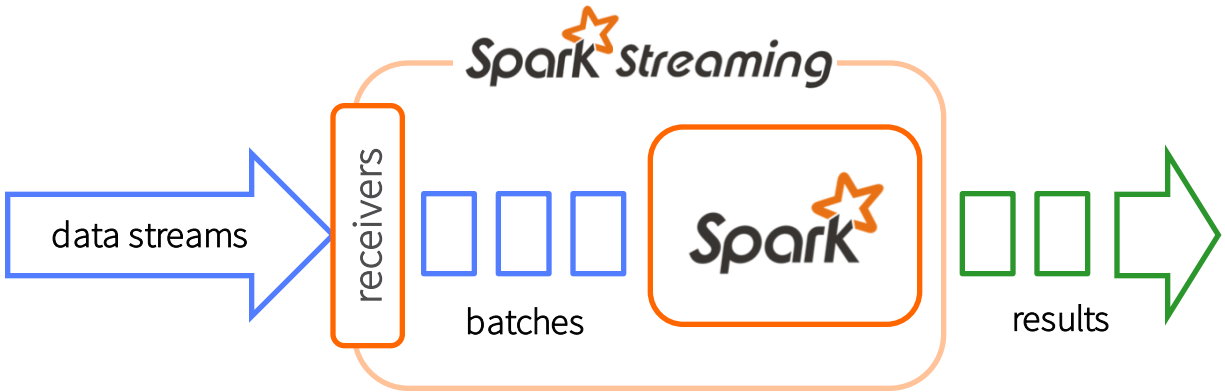
- 450 Billion Events/Day
- 8 Million (17GB) / Second Peak



How does it work?

Receivers receive data streams and chop them up into batches

Spark processes the batches and pushes out the results



Word Count

Entry point

Batch Interval

```
val context = new StreamingContext(conf, Seconds(1))

val lines = context.socketTextStream(...)
```

DStream: represents a data stream

Word Count

```
val context = new StreamingContext(conf, Seconds(1))

val lines = context.socketTextStream(...)

val words = lines.flatMap(_.split(" "))
```

Transformations: transform data to create new DStreams

Word Count

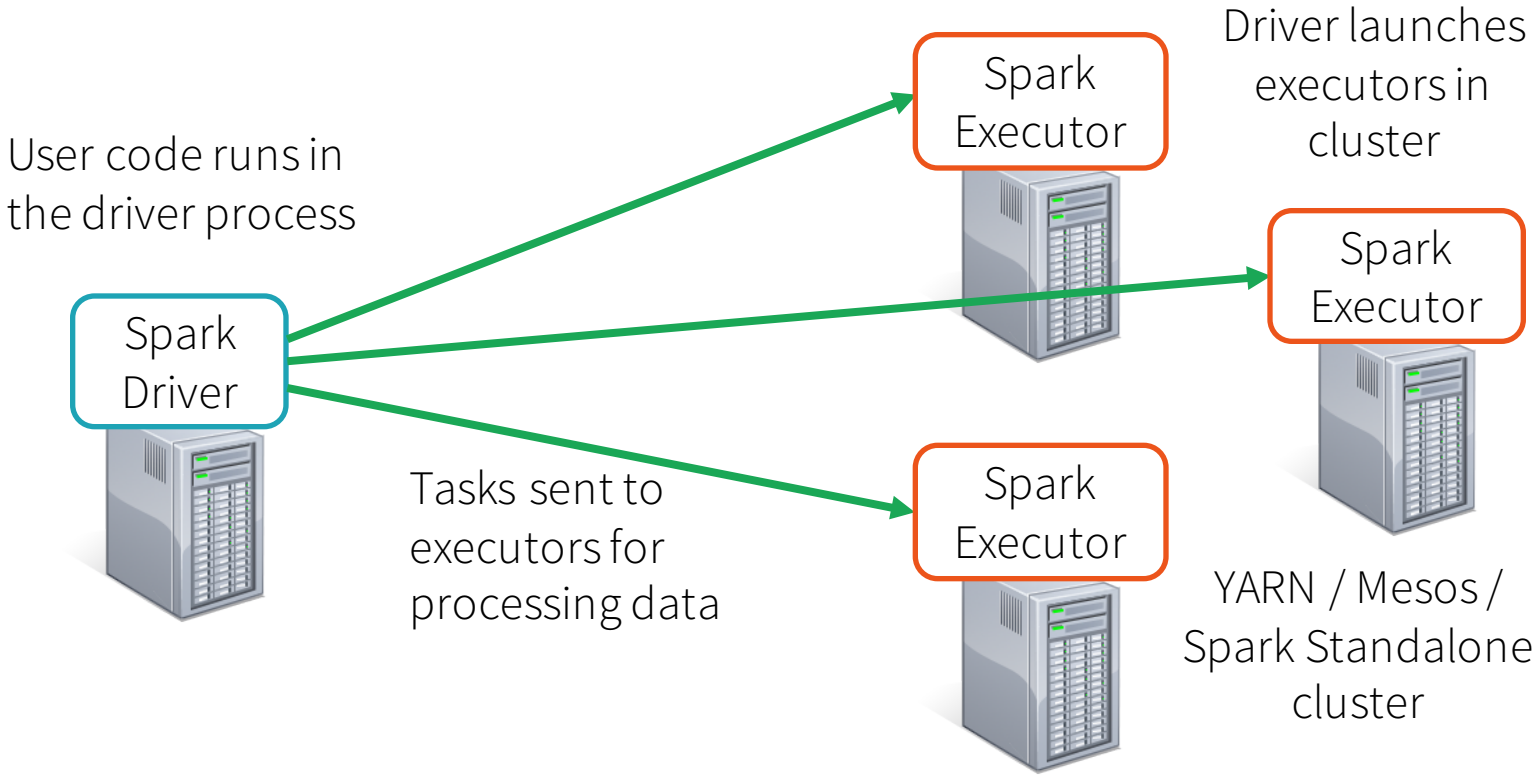
```
val context = new StreamingContext(conf, Seconds(1))
val lines = context.socketTextStream(...)
val words = lines.flatMap(_.split(" "))
val wordCounts = words.map(x => (x, 1)).reduceByKey(_+_)
wordCounts.print()
context.start()
```

Print the DStream contents on screen

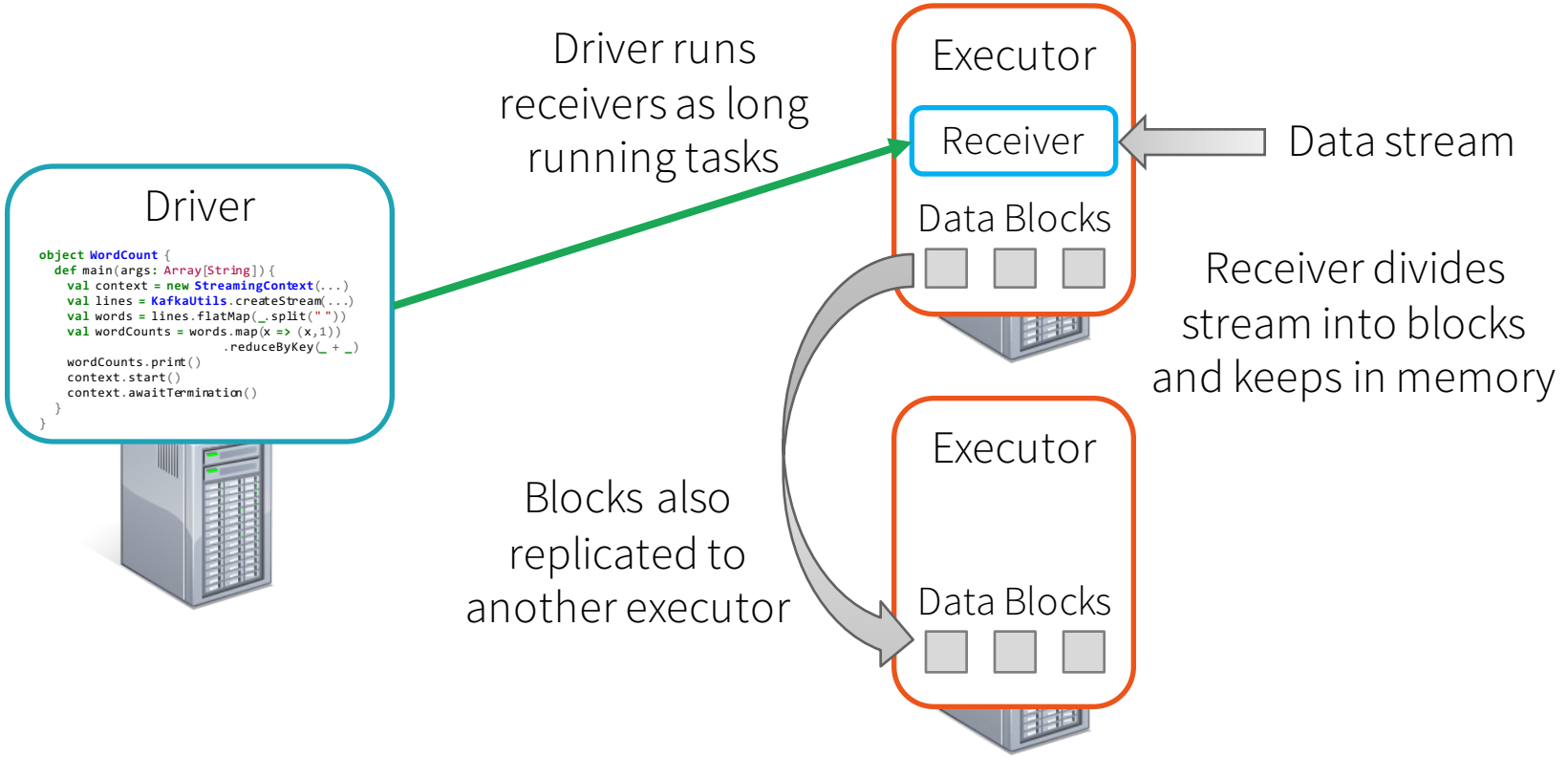
Start the streaming job

Lifecycle of Streaming Application

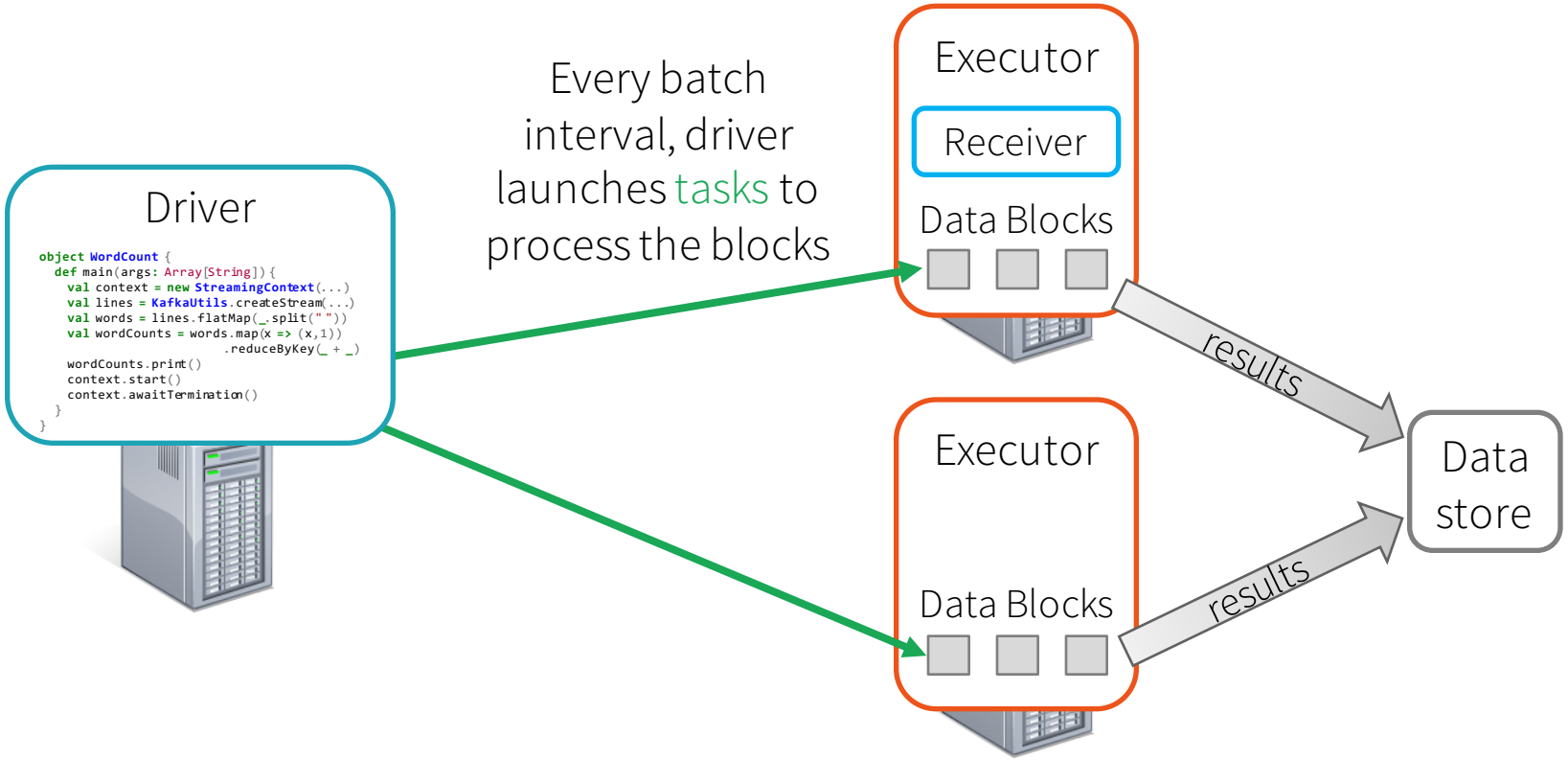
Spark Application Execution



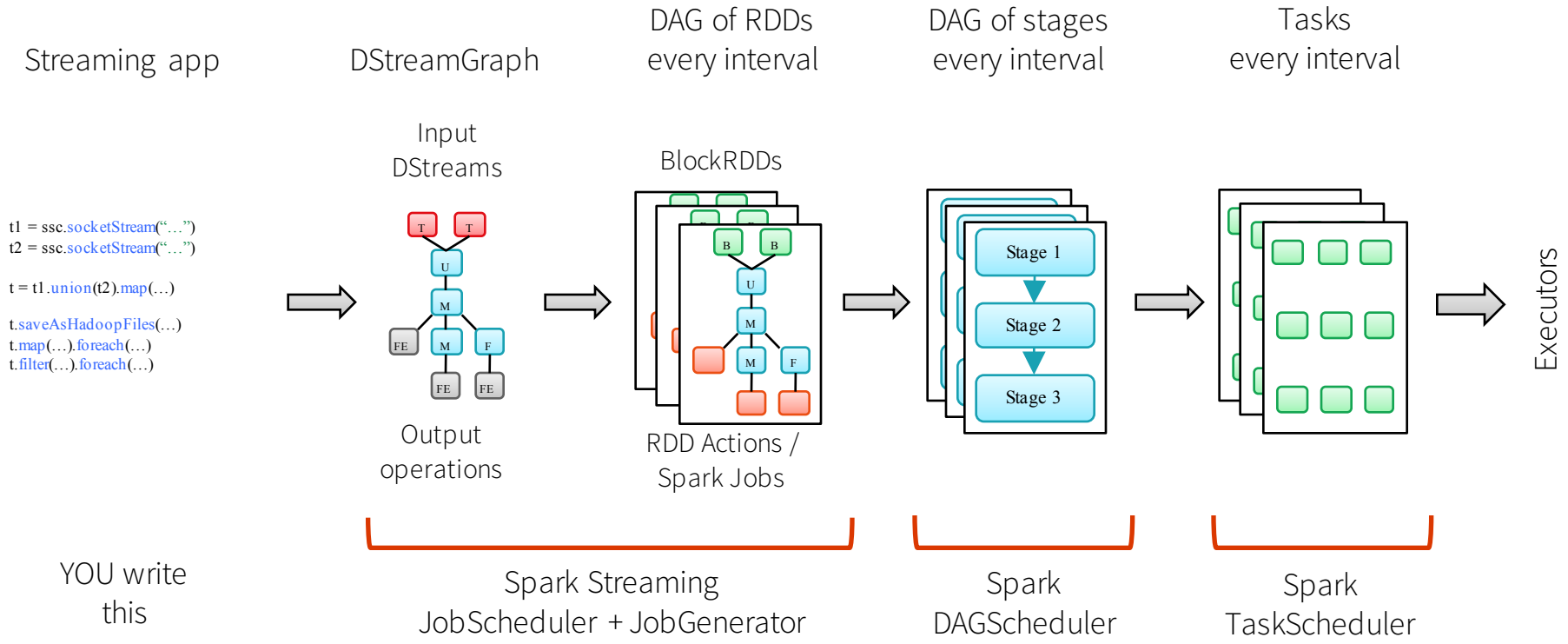
Receiving Data



Process Data



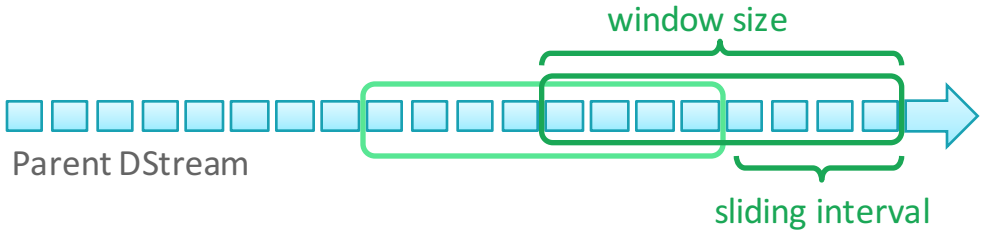
Streaming End-to-End View



databricks

Aggregations

Word count over a time window



Reduces over a time window

```
val wordCounts = wordStream.reduceByKeyAndWindow((x:
  Int, y:Int) => x+y, windowSize, slidingInterval)
```

Word count over a time window

Scenario: Word count for the last 30 minutes

How to optimize for good performance?

- Increase batch interval, if possible
- Incremental aggregations with inverse reduce function

```
val wordCounts = wordStream.reduceByKeyAndWindow(
    (x: Int, y:Int) => x+y, (x: Int, y: Int) =>
     x-y, windowSize, slidingInterval)
```

- Checkpointing

```
wordStream.checkpoint(checkpointInterval)
```

Stateful: Global Aggregations

Scenario: Maintain a global state based on the input events coming in. Ex: Word count from beginning of time.

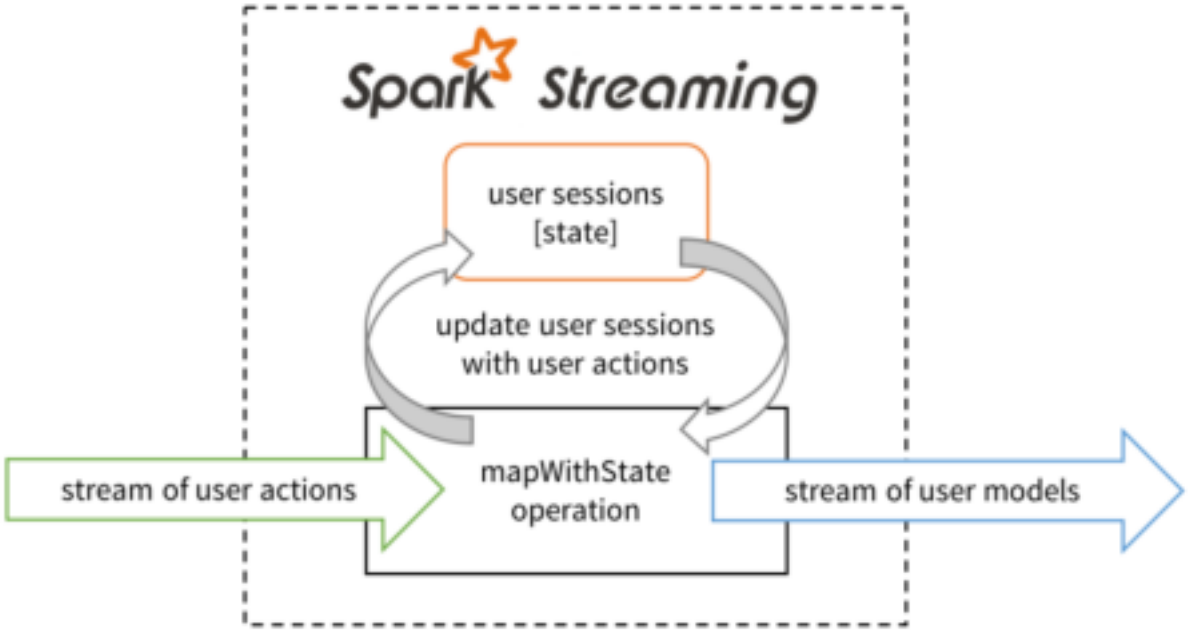
~~updateStateByKey (Spark 1.5 and before)~~

- Performance is proportional to the size of the state.

mapWithState (Spark 1.6+)

- Performance is proportional to the size of the batch.

Stateful: Global Aggregations



Stateful: Global Aggregations

Key features of mapWithState:

- An initial state - Read from somewhere as a RDD
- # of partitions for the state - If you have a good estimate of the size of the state, you can specify the # of partitions.
- Partitioner - Default: Hash partitioner. If you have a good understanding of the key space, then you can provide a custom partitioner
- Timeout - Keys whose values are not updated within the specified timeout period will be removed from the state.

Stateful: Global Aggregations: Word Count

```
val stateSpec = StateSpec.function(updateState _)
   .initialState(initialRDD)
   .numPartitions(100)
   .partitioner(MyPartitioner())
   .timeout(Minutes(120))
val wordCountState = wordStream.mapWithState(stateSpec)
```

Stateful: Global Aggregations: Word Count

```
def updateState(batchTime: Time,
                key: String,
                value: Option[Int],
                state: State[Long])
              : Option[(String, Long)]
```

Current batch time

A Word in the input stream

Current value (= 1)

Counts so far for the word

The word and its new count

Good References

- Four things to know about reliable Spark Streaming
- Productionizing your Streaming Jobs

Quick Start

Quick Start

- Quick Start Using Scala
- Quick Start Using Python

Data Import

- Data Import Guide
- Data Import Notebook

Data Exploration on Databricks

- Tutorial Video
- Setup Notebook
- Data Exploration Notebook

Scenarios

Population vs. Price

MLlib and DataFrames Visualizations

- Simplifying Machine Learning with Databricks Blog Post
- Population vs. Price Multi-chart Spark SQL Notebook
- Population vs. Price Linear Regression Python Notebook

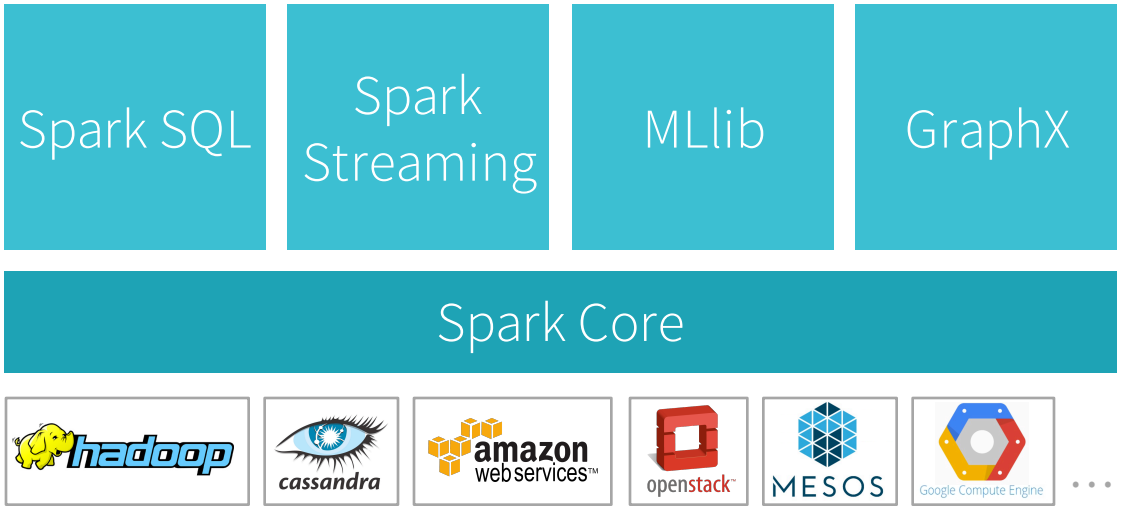
Miscellaneous

DataFrames, UDFs, etc.

- Mobile Phone Sample Notebook
- Salesforce Leads with Machine Learning, Spark SQL, and UDFs Notebook
- AdTech Sample Notebook (Part 1)

Spark Quick Primer

Apache Spark Engine



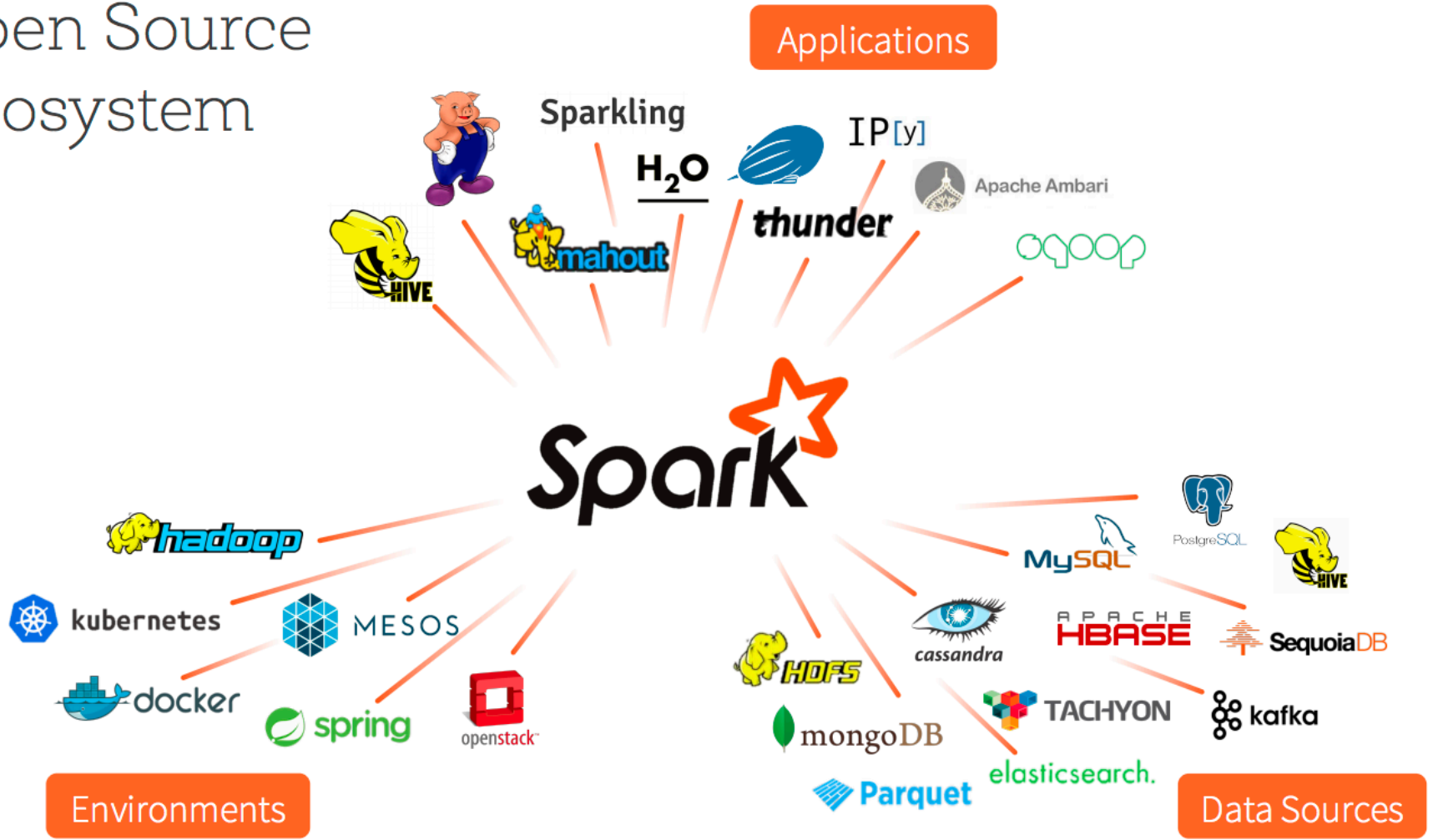
Scale out, fault tolerant

Python, Java, Scala, and R APIs

Standard libraries

Unified engine across diverse workloads & environments

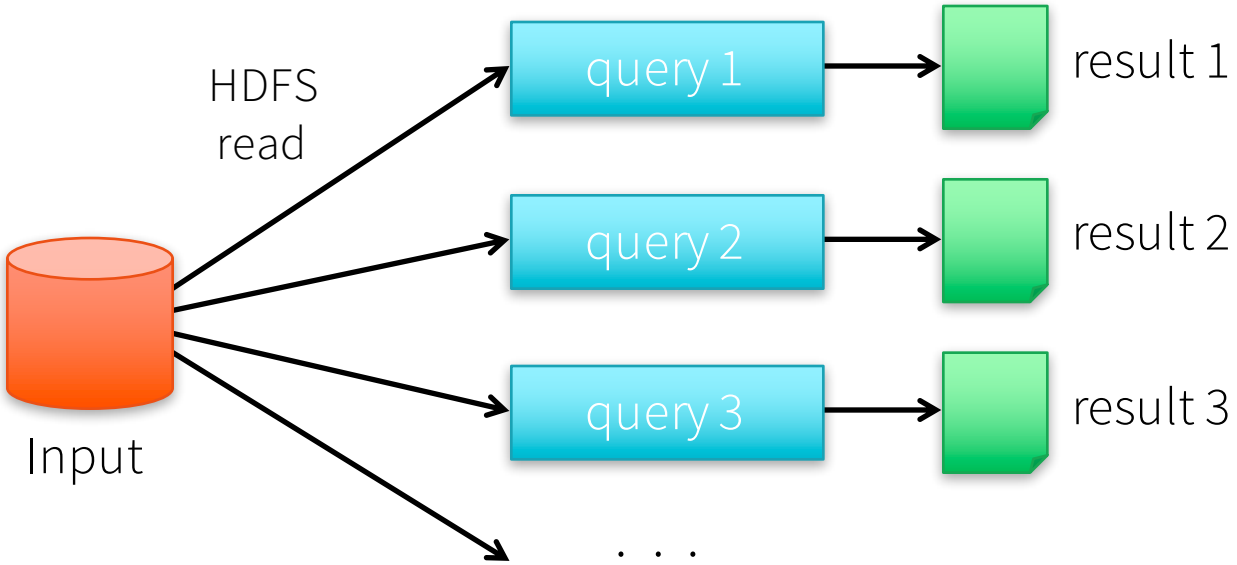
Open Source Ecosystem



NOTABLE USERS THAT PRESENTED AT SPARK SUMMIT 2015 SAN FRANCISCO

Source: Slide 5 of Spark Community Update

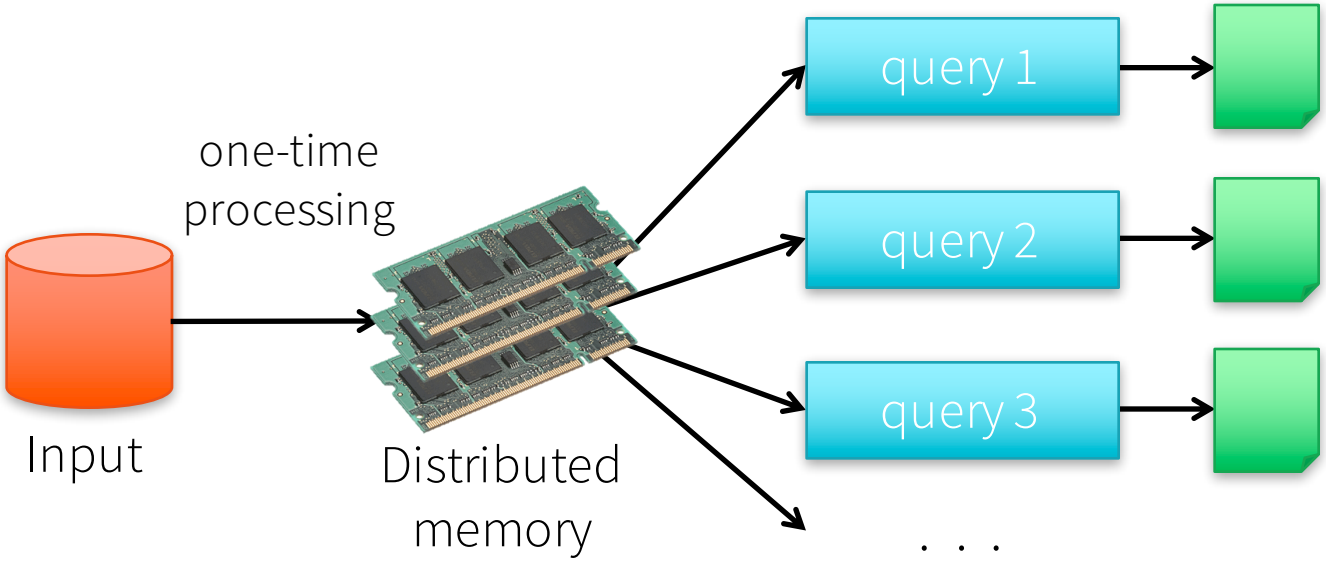
Data Sharing in MapReduce



Slow due to replication, serialization, and disk IO

Source: Seattle Spark Meetup Presentation by Matei Zaharia and Pat McDonough

Data Sharing in Spark



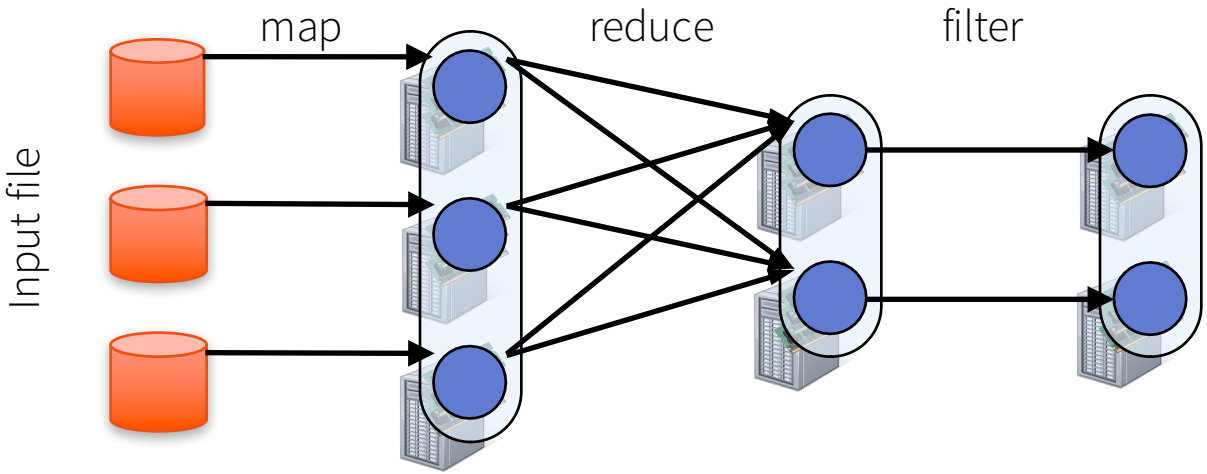
10-100× faster than network and disk

Source: Seattle Spark Meetup Presentation by Matei Zaharia and Pat McDonough

Fault Tolerance

RDDs track lineage info to rebuild lost data

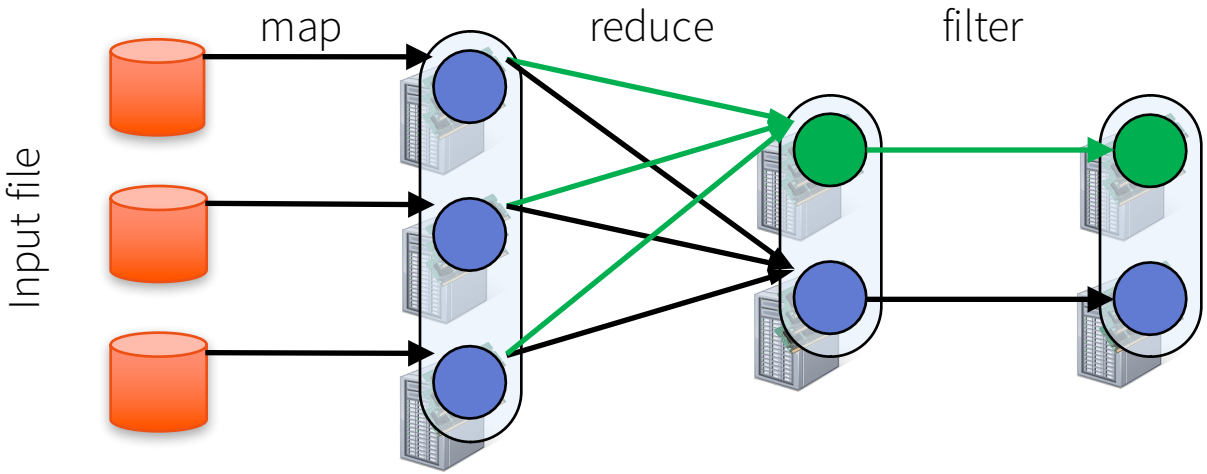
```
file.map(lambda rec: (rec.type, 1))
        .reduceByKey(lambda x, y: x + y)
        .filter(lambda (type, count): count > 10)
```



Fault Tolerance

RDDs track lineage info to rebuild lost data

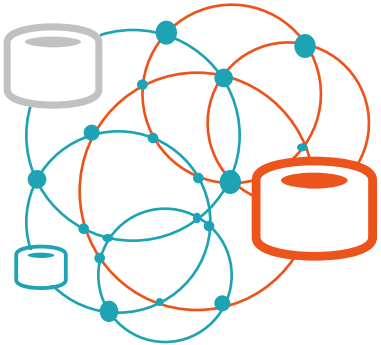
```
file.map(lambda rec: (rec.type, 1))
        .reduceByKey(lambda x, y: x + y)
        .filter(lambda (type, count): count > 10)
```



PROBLEM

Building infrastructure and data pipelines is complex

3 main causes of this problem:

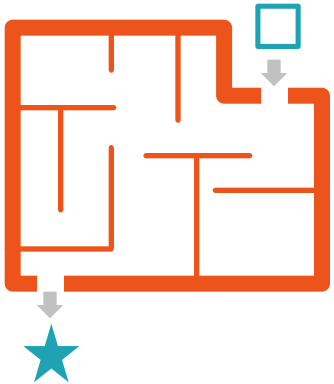


Infrastructure is complex to build and maintain

- Expensive upfront investment
- Months to build
- Dedicated DevOps to operate

Tools are slow, clunky, and disparate

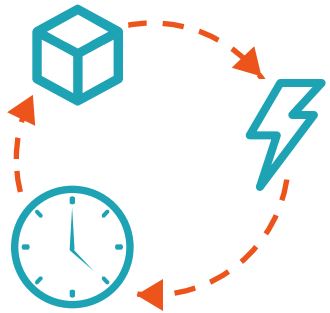
- Not user-friendly
- Long time to compute answers
- Lots of integration required



Re-engineering of prototypes for deployment

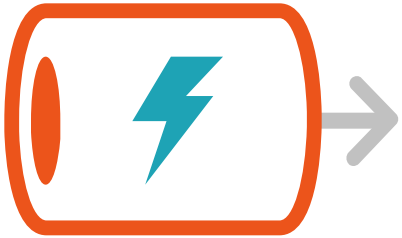
- Duplicated effort
- Complexity to achieve production quality

Databricks benefits



Higher productivity

- Maintenance-free infrastructure
- Real time processing
- Easy to use tools



Faster deployment of data pipelines

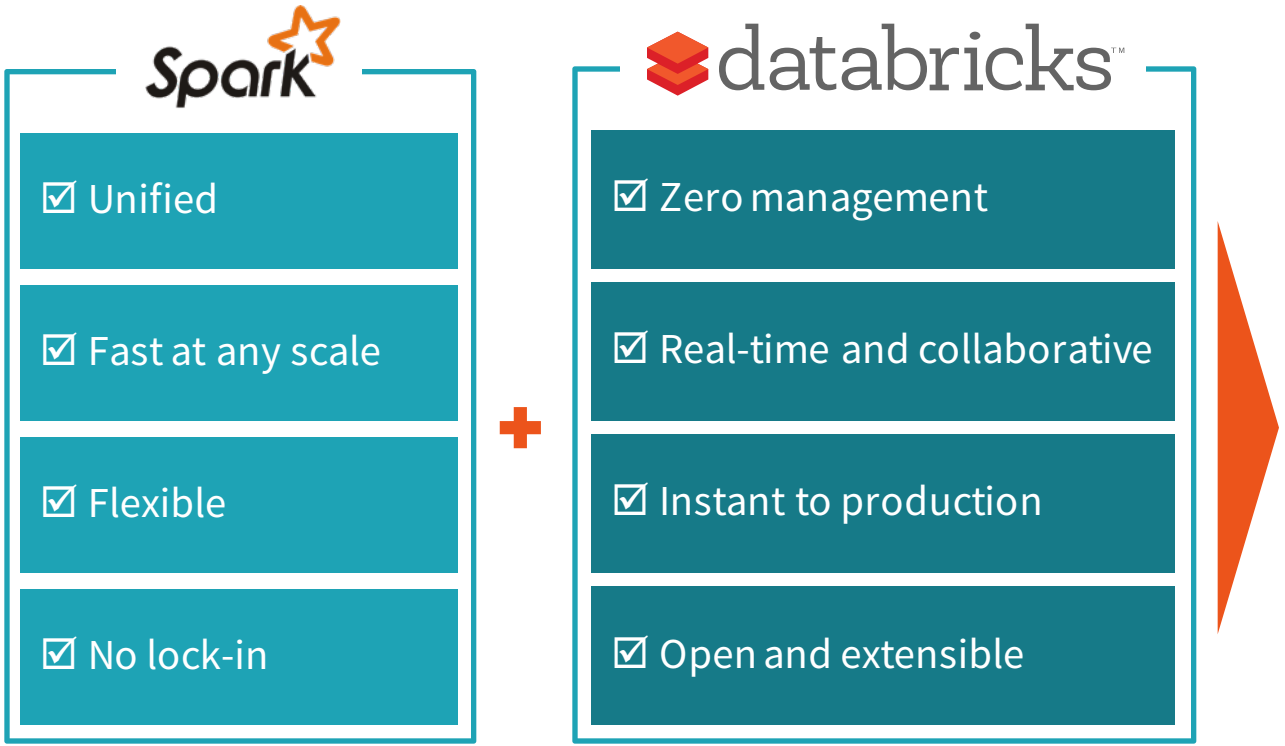
- Zero management Spark clusters
- Instant transition from prototype to production

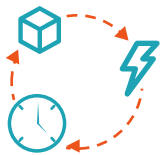
Data democratization

- One shared repository
- Seamless collaboration
- Easy to build sophisticated dashboards and notebooks

Data science made easy with Apache Spark

From ingest to production





Higher productivity

Faster deployment
of data pipelines

Data democratization

Demos

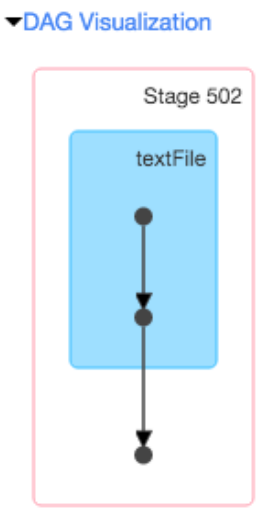
Quick Start | Data Import | Ad Tech Example | Population vs. Price Example | Spark API Performance | Meetup Streaming RSVPs | Transitioning from DW to Data Sciences | Movie Recommendations with MLlib

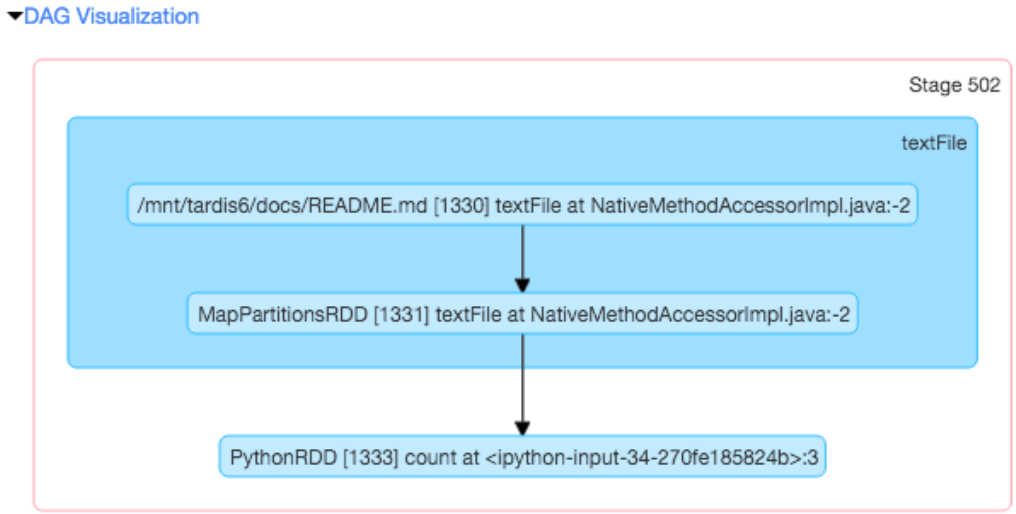
Quick Start

Quick Start Using Python | Quick Start Using Scala

Quick Start with Python

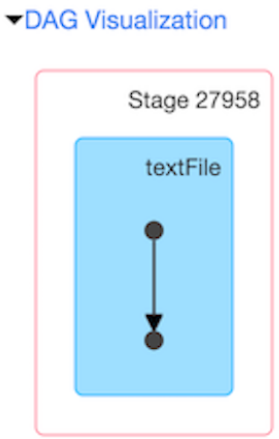
```
textFile = sc.textFile("/mnt/tardis6/docs/README.md")
textFile.count()
```

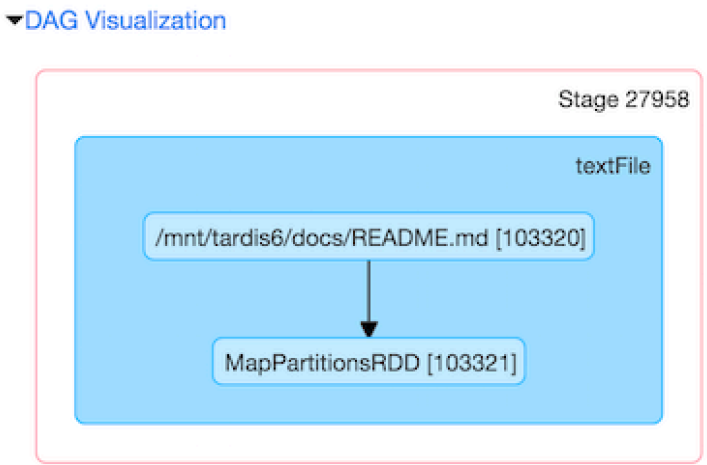




Quick Start with Scala

```
textFile = sc.textFile("/mnt/tardis6/docs/README.md")
textFile.count()
```





RDDs

- RDDs have *actions*, which return values, and *transformations*, which return pointers to new RDDs.
- Transformations are *lazy* and executed when an *action* is run
  - Transformations: `map()`, `flatMap()`, `filter()`, `mapPartitions()`, `mapPartitionsWithIndex()`, `sample()`, `union()`, `distinct()`, `groupByKey()`, `reduceByKey()`, `sortByKey()`, `join()`, `cogroup()`, `pipe()`, `coalesce()`, `repartition()`, `partitionBy()`, `...`
  - Actions: `reduce()`, `collect()`, `count()`, `first()`, `take()`, `takeSample()`, `takeOrdered()`, `saveAsTextFile()`, `saveAsSequenceFile()`, `saveAsObjectFile()`, `countByKey()`, `foreach()`, `...`
- Persist (cache) distributed data in memory or disk

Data Import

Data Import How-To Guide | Data Import Notebook

Enter your AWS access keys

Configure your AWS key settings

```
> import urllib
  ACCESS_KEY = "XHFKDFAEHFKASHFFDFAE"
  SECRET_KEY = "XDNn42u2-8a+rlkje597kd0937f3fcasdaqrjgya"
  ENCODED_SECRET_KEY = urllib.quote(SECRET_KEY, "")
  AWS_BUCKET_NAME = "my-data-for-databricks"
  MOUNT_NAME = "my-data"
```

Mount your bucket

```
> dbutils.fs.mount("s3n://%s:%s@%s" % (ACCESS_KEY, ENCODED_SECRET_KEY, AWS_BUCKET_NAME), "/mnt/%s" % MOUNT_NAME)

Successfully mounted to /mnt/my-data!
Out[11]: True
```

databricks

Access your data
Using dbutils

```
> display(dbutils.fs.ls("/mnt/my-data"))
```

path	name	size
dbfs:/mnt/my-data/apache/	apache/	0
dbfs:/mnt/my-data/iis/	iis/	0
dbfs:/mnt/my-data/map/	map/	0
dbfs:/mnt/my-data/response/	response/	0

Count the number of rows in all of the files in your Apache Access Web Logs folder

```
> myApacheLogs = sc.textFile("/mnt/my-data/apache")
  myApacheLogs.count()
```

```
Out[2]: 4468
```

databricks

Review ten rows from your Apache Access web logs

```
> myApacheLogs.take(10)
```

```
Out[3]:
[u'10.0.0.127 - 2696232 [14/Aug/2015:00:00:26 -0800] "GET /index.html HTTP/1.1" 304 428',
 u'10.0.0.104 - 2404465 [14/Aug/2015:00:01:14 -0800] "GET /Cascades/rss.xml HTTP/1.1" 304 514',
 u'10.0.0.108 - 2404465 [14/Aug/2015:00:04:21 -0800] "GET /Olympics/rss.xml HTTP/1.1" 200 499',
 u'10.0.0.213 - 2185662 [14/Aug/2015:00:05:15 -0800] "GET /Hurricane+Ridge/rss.xml HTTP/1.1" 200 288',
 u'10.0.0.203 - 2185662 [14/Aug/2015:00:05:17 -0800] "GET /index.html HTTP/1.1" 200 212',
 u'10.0.0.104 - 2696232 [14/Aug/2015:00:06:09 -0800] "GET /Cascades/rss.xml HTTP/1.1" 304 420',
 u'10.0.0.206 - 2576242 [14/Aug/2015:00:08:40 -0800] "GET /index.html HTTP/1.1" 304 343',
 u'10.0.0.213 - 2185662 [14/Aug/2015:00:09:07 -0800] "GET /Olympics/rss.xml HTTP/1.1" 304 323',
 u'10.0.0.212 - 2404465 [14/Aug/2015:00:10:29 -0800] "GET /index.html HTTP/1.1" 304 530',
 u'10.0.0.114 - 2575718 [14/Aug/2015:00:11:22 -0800] "GET /index.html HTTP/1.1" 304 341']
```

Setup Apache Access Log DataFrame

```
# sc is an existing SparkContext.
from pyspark.sql import SQLContext, Row

# Load the space-delimited web logs (text files)
parts = myApacheLogs.map(lambda l: l.split(" "))
apachelogs = parts.map(lambda p: Row(ipaddress=p[0], clientidentd=p[1], userid=p[2], datetime=p[3], tmz=p[4], method=p[5], endpoint=p[6], protocol=p[7],
responseCode=p[8], contentSize=p[9]))
```

```
# Infer the schema, and register the DataFrame as a table.
schemaApacheLogs = sqlContext.createDataFrame(apachelogs)
schemaApacheLogs.registerTempTable("apachelogs")
```

```
# Access the table using Spark SQL within the sqlContext
sqlContext.sql("select ipaddress, endpoint from apachelogs").take(10)
```

```
Out[6]:
[Row(ipaddress=u'10.0.0.127', endpoint=u'/index.html'),
 Row(ipaddress=u'10.0.0.104', endpoint=u'/Cascades/rss.xml'),
 Row(ipaddress=u'10.0.0.108', endpoint=u'/Olympics/rss.xml'),
 Row(ipaddress=u'10.0.0.213', endpoint=u'/Hurricane+Ridge/rss.xml'),
 Row(ipaddress=u'10.0.0.203', endpoint=u'/index.html'),
 Row(ipaddress=u'10.0.0.104', endpoint=u'/Cascades/rss.xml'),
 Row(ipaddress=u'10.0.0.206', endpoint=u'/index.html'),
 Row(ipaddress=u'10.0.0.213', endpoint=u'/Olympics/rss.xml'),
 Row(ipaddress=u'10.0.0.212', endpoint=u'/index.html'),
 Row(ipaddress=u'10.0.0.114', endpoint=u'/index.html')]
```

Ad Tech Example

AdTech Sample Notebook (Part 1)

Create External Table with RegEx

```
CREATE EXTERNAL TABLE accesslog (
  ipaddress STRING,
  ...
)
ROW FORMAT
  SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
  "input.regex" = '^(\\S+) (\\S+) (\\S+) \\[([\\w:/]+\\s[+\\-]\\d{4})\\]
\\"(\\S+) (\\S+) (\\S+)\\" (\\d{3}) (\\d+) \\"(.*)\\" \\"(.*)\\" (\\S+)
\\"(\\S+), (\\S+), (\\S+), (\\S+)\\"'
)
LOCATION
  "/mnt/mdl/accesslogs/"
```

External Web Service Call via Mapper

```
# Obtain the unique agents from the accesslog table
ipaddresses = sqlContext.sql("select distinct ip1 from \
 accesslog where ip1 is not null").rdd

# getCCA2: Obtains two letter country code based on IP address
def getCCA2(ip):
  url = 'http://freegeoip.net/csv/' + ip
  str = urllib2.urlopen(url).read()
  return str.split(",")[1]

# Loop through distinct IP addresses and obtain two-letter country codes
mappedIPs = ipaddresses.map(lambda x: (x[0], getCCA2(x[0])))
```

Join DataFrames and Register Temp Table

```
# Join countrycodes with mappedIPsDF so we can have IP address and
# three-letter ISO country codes
mappedIP3 = mappedIP2 \
  .join(countryCodesDF, mappedIP2.cca2 == countryCodesDF.cca2, "left_outer") \
  .select(mappedIP2.ip, mappedIP2.cca2, countryCodesDF.cca3, countryCodesDF.cn)

# Register the mapping table
mappedIP3.registerTempTable("mappedIP3")
```

Add Columns to DataFrames with UDFs

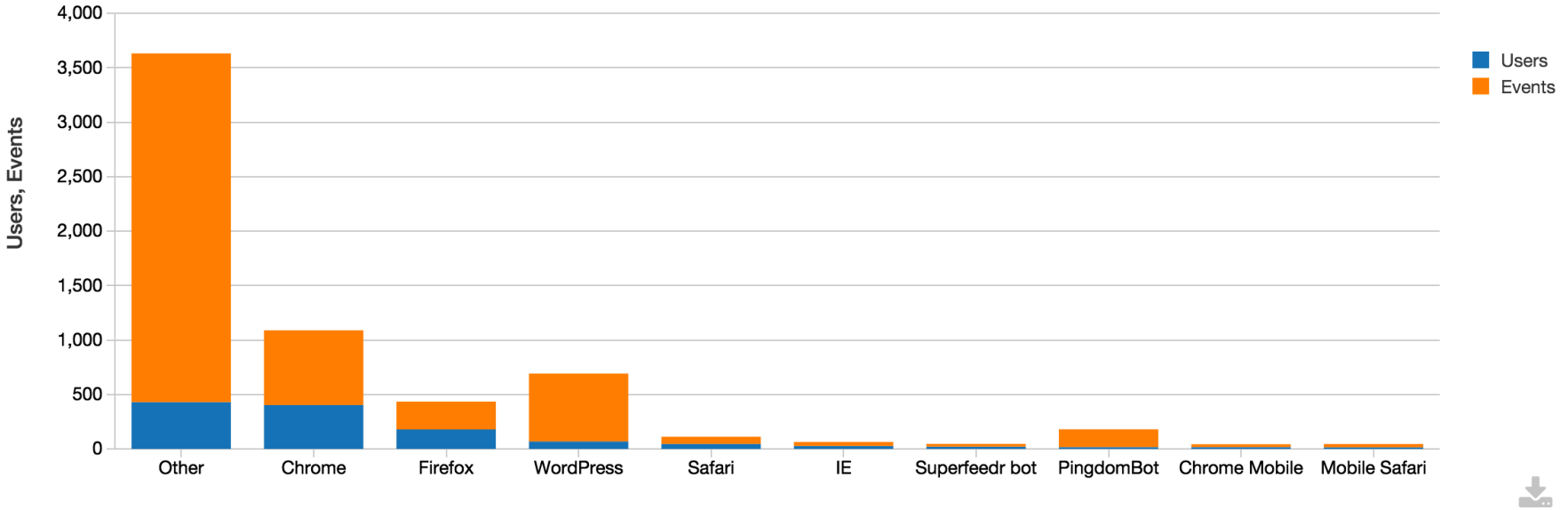
```
from user_agents import parse
from pyspark.sql.types import StringType
from pyspark.sql.functions import udf

# Create UDFs to extract out Browser Family information
def browserFamily(ua_string) : return xstr(parse(xstr(ua_string)).browser.family)
udfBrowserFamily = udf(browserFamily, StringType())

# Obtain the unique agents from the accesslog table
userAgentTbl = sqlContext.sql("select distinct agent from accesslog")

# Add new columns to the UserAgentInfo DataFrame containing browser information
userAgentInfo = userAgentTbl.withColumn('browserFamily', \
   udfBrowserFamily(userAgentTbl.agent))
```

Use Python UDFs with Spark SQL

```
# Define function (converts Apache web log time)
def weblog2Time(weblog_timestr): ...

# Define and Register UDF
udfWeblog2Time = udf(weblog2Time, DateType())
sqlContext.registerFunction("udfWeblog2Time", lambda x: weblog2Time(x))

# Create DataFrame
accessLogsPrime = sqlContext.sql("select hash(a.ip1, a.agent) as UserId,
   m.cca3, udfWeblog2Time(a.datetime) ...")
```

```
> %sql select browserFamily, count(distinct UserID) as Users, count(1) as Events from accessLogsPrime group by browserFamily order by Users desc limit 10;
```

▸ (1) Spark Jobs



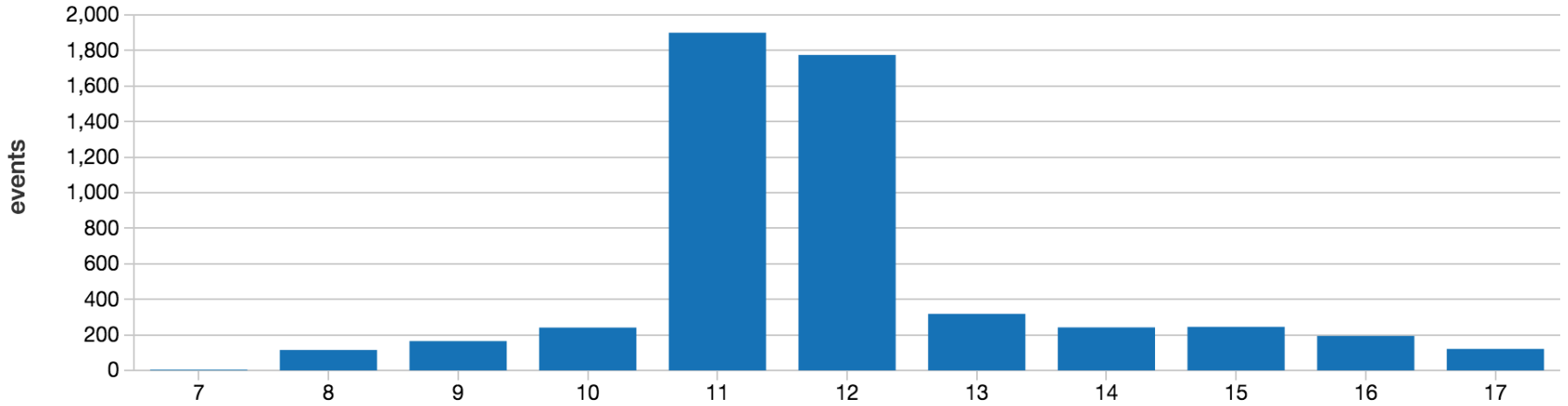
Plot Options...

Command took 3.12s

databricks

```
> %sql select hour(datetime) as Hour, count(1) as events from accessLogsPrime group by hour(datetime) order by hour(datetime)
```

▸ (1) Spark Jobs

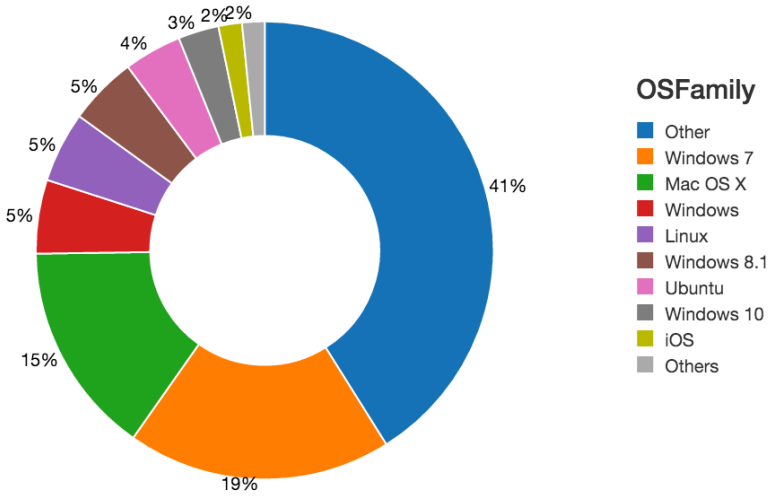


Plot Options...

Command took 2.45s

```
> %sql select OSFamily, count(distinct UserID) as Users from accessLogsPrime group by OSFamily order by Users desc limit 10;
```

▸ (1) Spark Jobs



Plot Options...

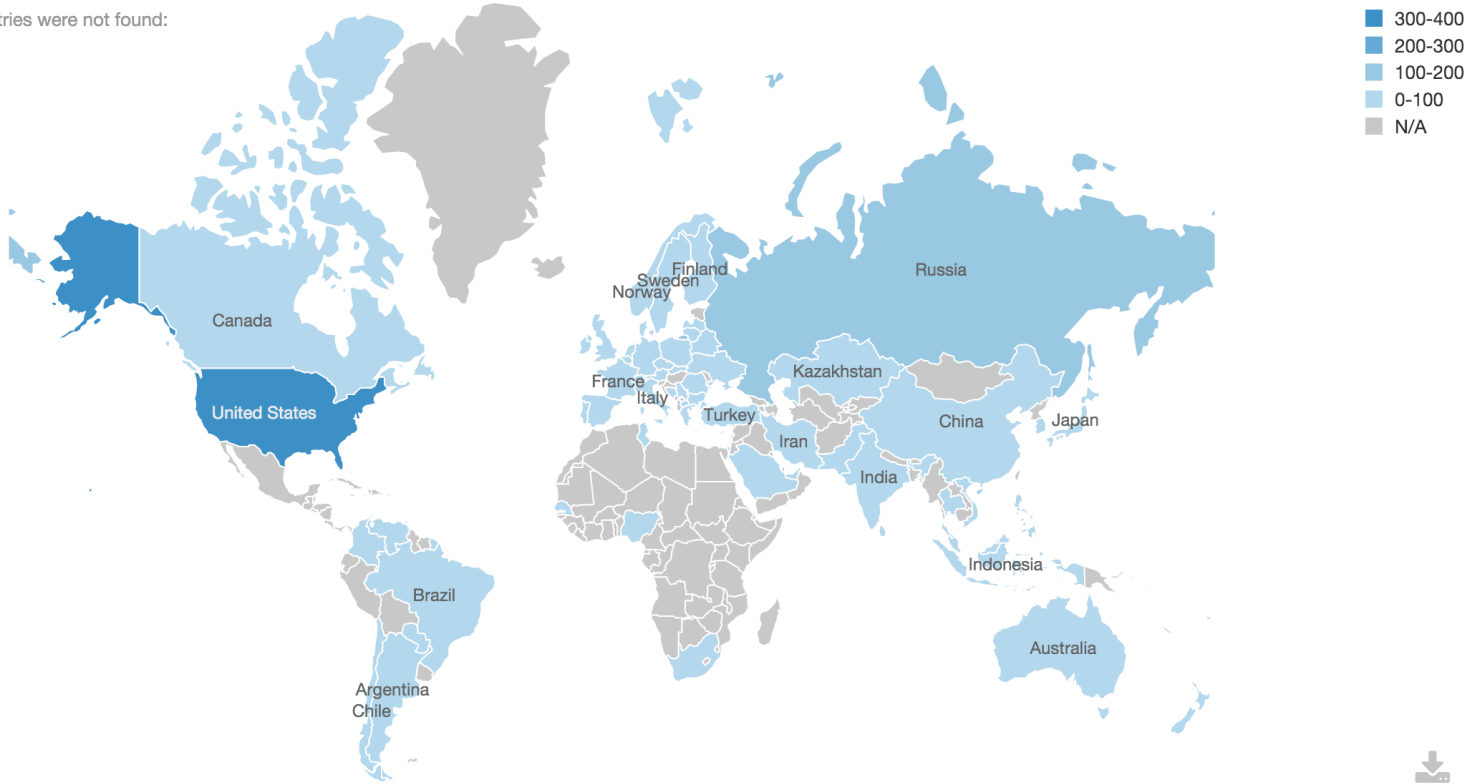
Command took 4.02s

databricks

```
> %sql select cca3, count(distinct UserID) as users from accessLogsPrime group by cca3
```

▸ (2) Spark Jobs

Following countries were not found:



Plot Options...

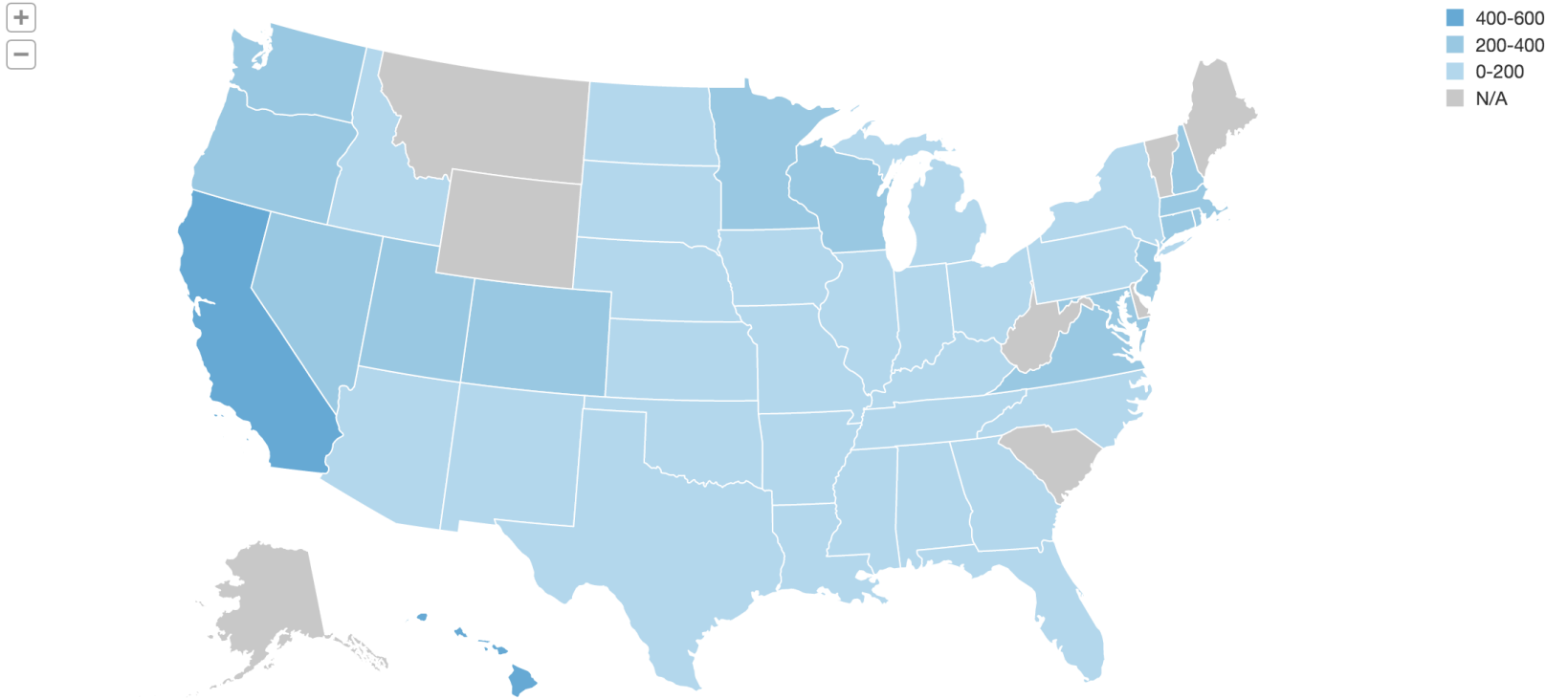
databricks

Population vs. Price Example

Simplifying Machine Learning with Databricks Blog Post
Population vs. Price Multi-chart Spark SQL Notebook
Population vs. Price Linear Regression Python Notebook
Spark MLlib: From Quick Start to Scikit-Learn (Add Link)

```
> select `State Code`, `2015 median sales price` from data_geo
```

▸ (2) Spark Jobs

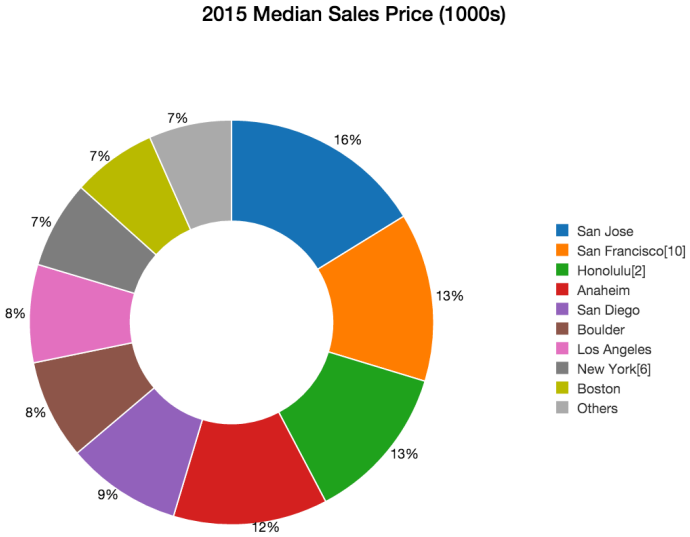


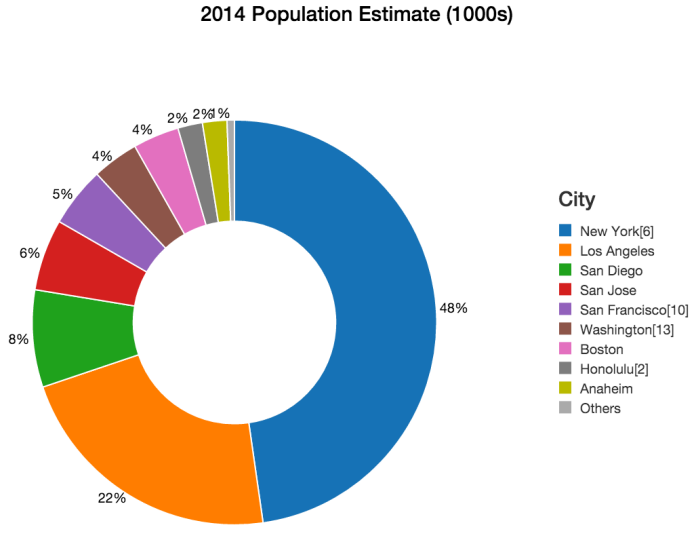
Plot Options...

databricks

```
> select City, `2014 Population estimate`/1000 as `2014 Population Estimate (1000s)`, `2015 median sales price` as `2015 Median Sales Price (1000s)` from data_geo
  order by `2015 median sales price` desc limit 10;
```

▸ (1) Spark Jobs



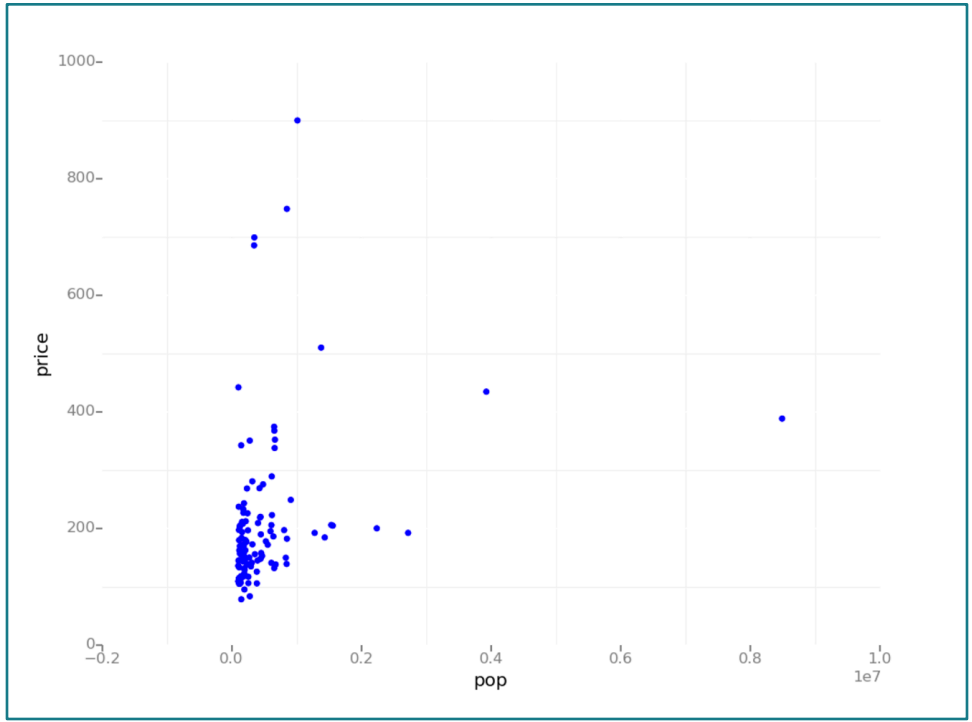


Plot Options...

databricks

Scatterplot

```
import numpy as np
import matplotlib.pyplot as plt

x = data.map(lambda p:
(p.features[0])).collect()
y = data.map(lambda p:
(p.label)).collect()

from pandas import *
from ggplot import *
pydf = DataFrame({'pop':x,'price':y})
p = ggplot(pydf, aes('pop','price')) + \
    geom_point(color='blue')
display(p)
```



Linear Regression with SGD
Define and Build Models

```
# Import LinearRegression class
from pyspark.ml.regression import LinearRegression

# Define LinearRegression model
lr = LinearRegression()

# Build two models
modelA = lr.fit(data, {lr.regParam:0.0})
modelB = lr.fit(data, {lr.regParam: 100.0})
```

Linear Regression with SGD
Make Predictions

```
# Make predictions
predictionsA = modelA.transform(data)
display(predictionsA)
```

▸ (2) Spark Jobs

features	label	prediction
▸ {"type":1,"size":1,"indices":[],"values":[212247]}	162.9	199.31676595846622
▸ {"type":1,"size":1,"indices":[],"values":[188226]}	157.7	198.40882267887176
▸ {"type":1,"size":1,"indices":[],"values":[194675]}	122.5	198.65258131548575
▸ {"type":1,"size":1,"indices":[],"values":[200481]}	129	198.8720359044423
▸ {"type":1,"size":1,"indices":[],"values":[1537058]}	206.1	249.39183544694856
▸ {"type":1,"size":1,"indices":[],"values":[527972]}	178.1	211.2505069330287
▸ {"type":1,"size":1,"indices":[],"values":[197706]}	131.8	198.76714674075743
▸ {"type":1,"size":1,"indices":[],"values":[346997]}	685.7	204.41003255541705
▸ {"type":1,"size":1,"indices":[],"values":[392886	434.7	220.70707185640

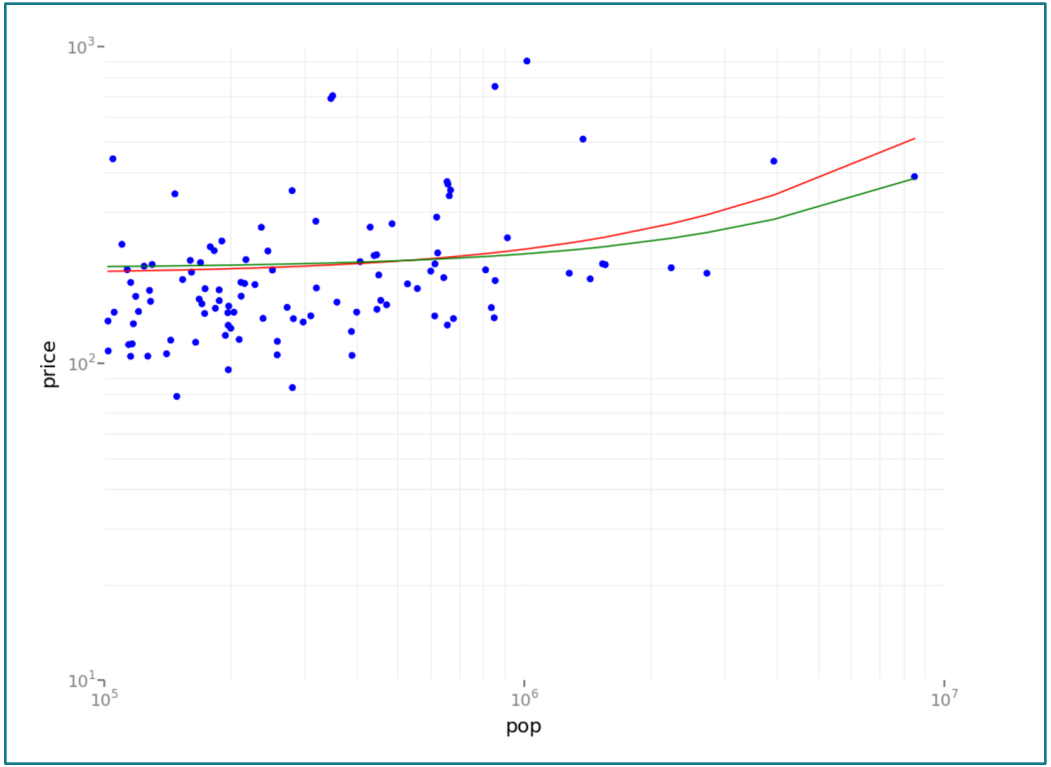
Linear Regression with SGD
Evaluate the Models

```
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(metricName="mse")
MSE = evaluator.evaluate(predictionsA)
print("ModelA: Mean Squared Error = " + str(MSE))
```

```
ModelA: Mean Squared Error = 16538.4813081
ModelB: Mean Squared Error = 16769.2917636
```

Scatterplot with plotting Regression Models

```
p = ggplot(pydf, aes('pop','price')) + \
    geom_point(color='blue') + \
    geom_line(pydf, aes('pop','predA'),
      color='red') + \
    geom_line(pydf, aes('pop','predB'),
      color='green') + \
    scale_x_log(10) + scale_y_log10()
display(p)
```



Learning more about MLlib

Guides & examples

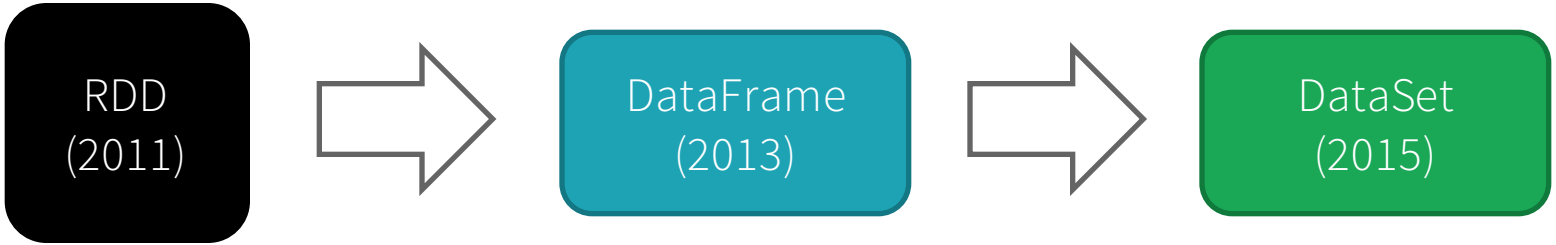
- Example workflow using ML Pipelines (Python)
- Power plant data analysis workflow (Scala)
- The above 2 links are part of the Databricks Guide, which contains many more examples and references.

References

- Apache Spark MLlib User Guide
  - The MLlib User Guide contains code snippets for almost all algorithms, as well as links to API documentation.
- Meng et al. "MLlib: Machine Learning in Apache Spark." 2015. http://arxiv.org/abs/1505.06807 (*academic paper*)

Spark API Performance

History of Spark APIs



Distribute collection of JVM objects

Functional Operators (map, filter, etc.)

Distribute collection of Row objects

Expression-based operations and UDFs

Logical plans and optimizer

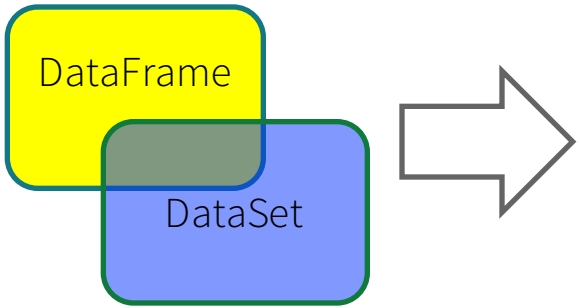
Fast/efficient internal representations

Internally rows, externally JVM objects

Almost the "Best of both worlds": type safe + fast

But slower than DF
Not as good for interactive analysis, especially Python

Apache Spark 2.0 API



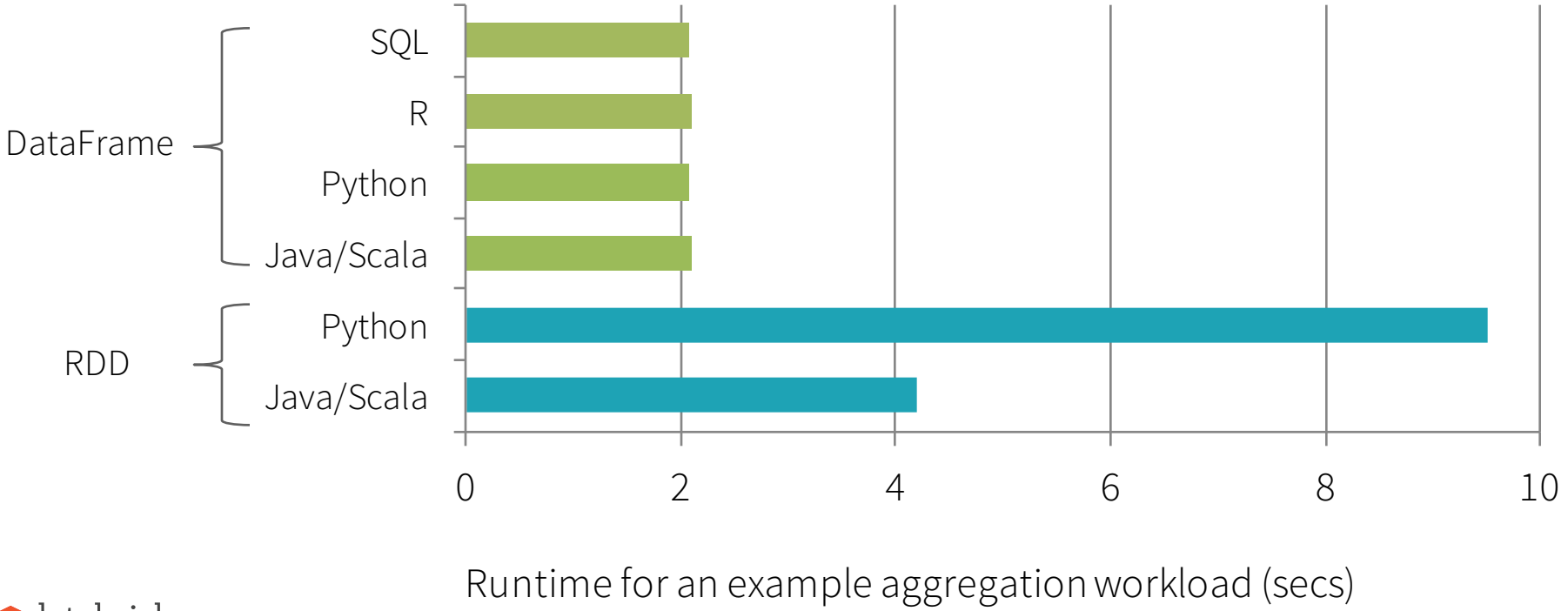
Untyped API

- `DataFrame = Dataset[Row]`
- Convenient for interactive analysis
- Faster

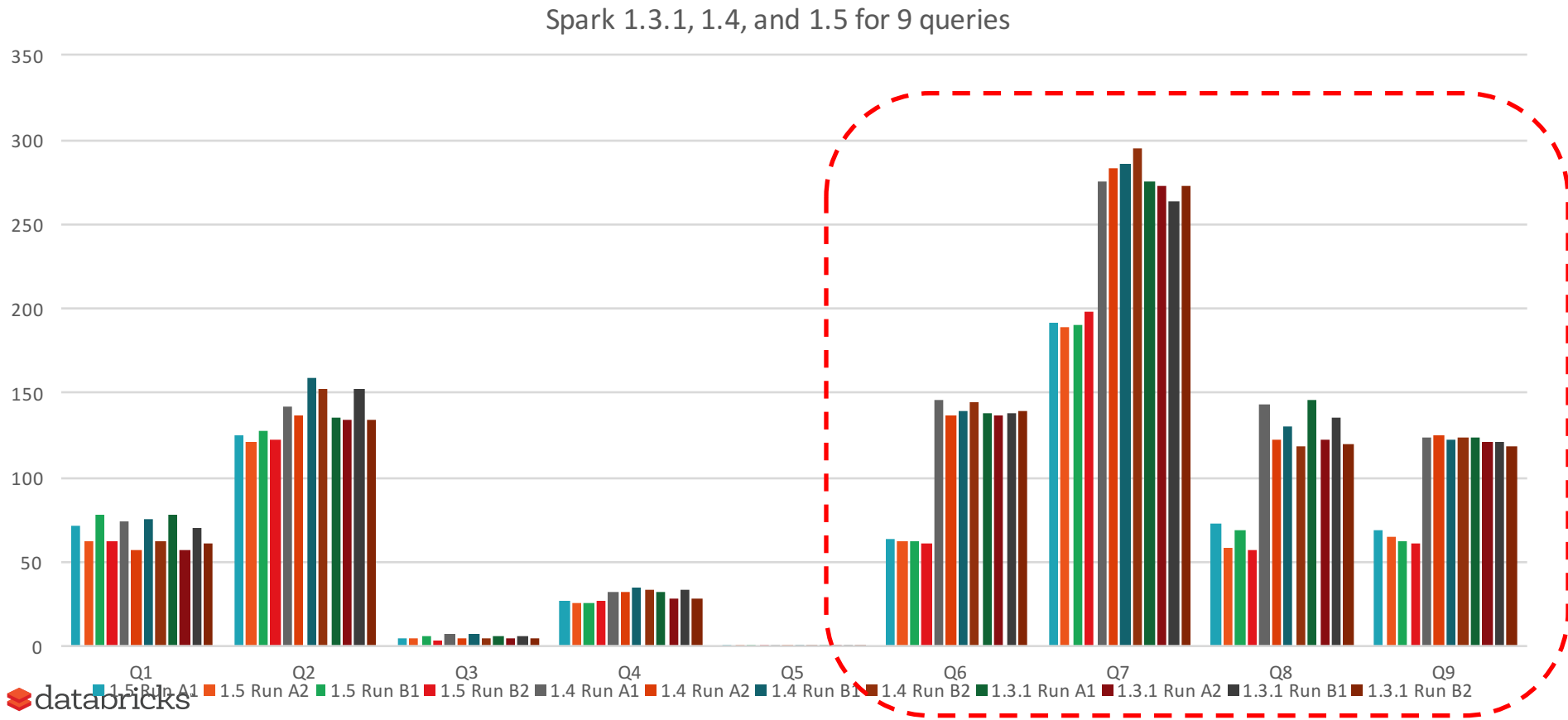
Typed API

- Optimized for data engineering
- Fast

Benefit of Logical Plan: Performance Parity Across Languages



NYC Taxi Dataset



databricks

Dataset API in Spark 1.6

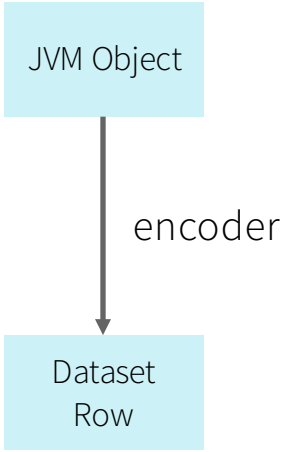
Typed interface over DataFrames / Tungsten

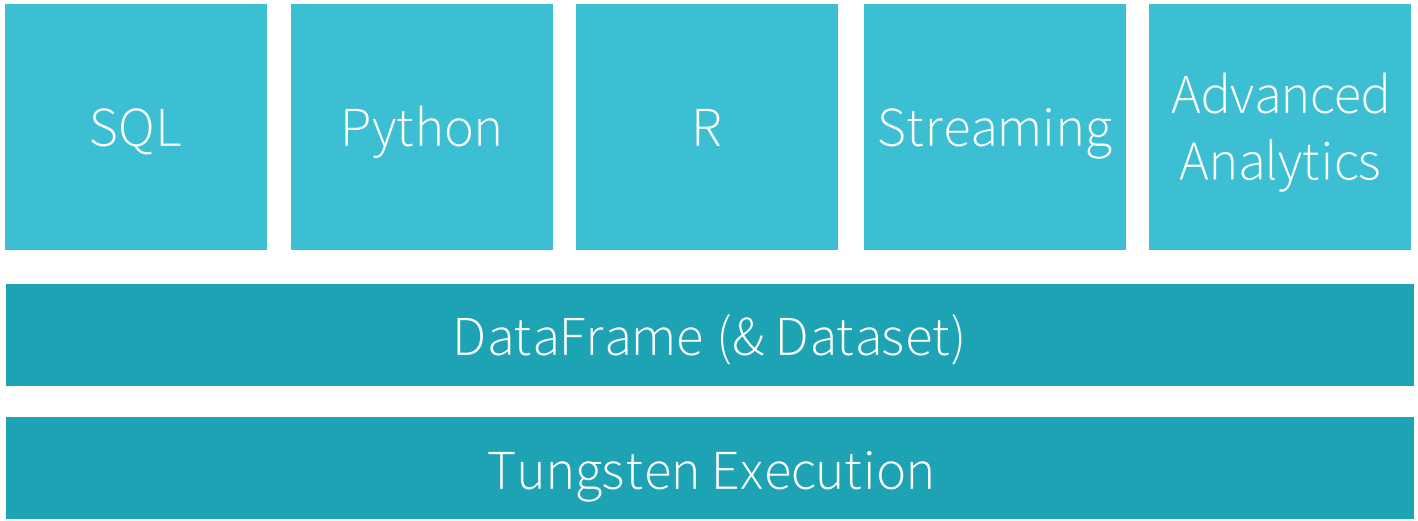
```
case class Person(name: String, age: Long)

val dataframe = read.json("people.json")
val ds: Dataset[Person] = dataframe.as[Person]

ds.filter(p => p.name.startsWith("M"))
      .toDF()
      .groupBy($"name")
      .avg("age")
```

Dataset

"Encoder" converts from JVM Object into a Dataset Row

Checkout [SPARK-9999]




SQL
Python
R
Streaming
Advanced Analytics
DataFrame (& Dataset)
Tungsten Execution

Meetup Streaming RSVPs

60 RSVPs / minute

Claire will meetup with
Texas Rock Climbing
in Austin, TX

Giovanna Meza will meetup with
Energy Coaching Barcelona
in Barcelona, ES
1 second ago

Quinn Eurich will meetup with
Hai An Pagoda Meditation
in New Britain, CT
4 seconds ago

Maria will meetup with
Social Fit Hub
in Toronto, ON
5 seconds ago

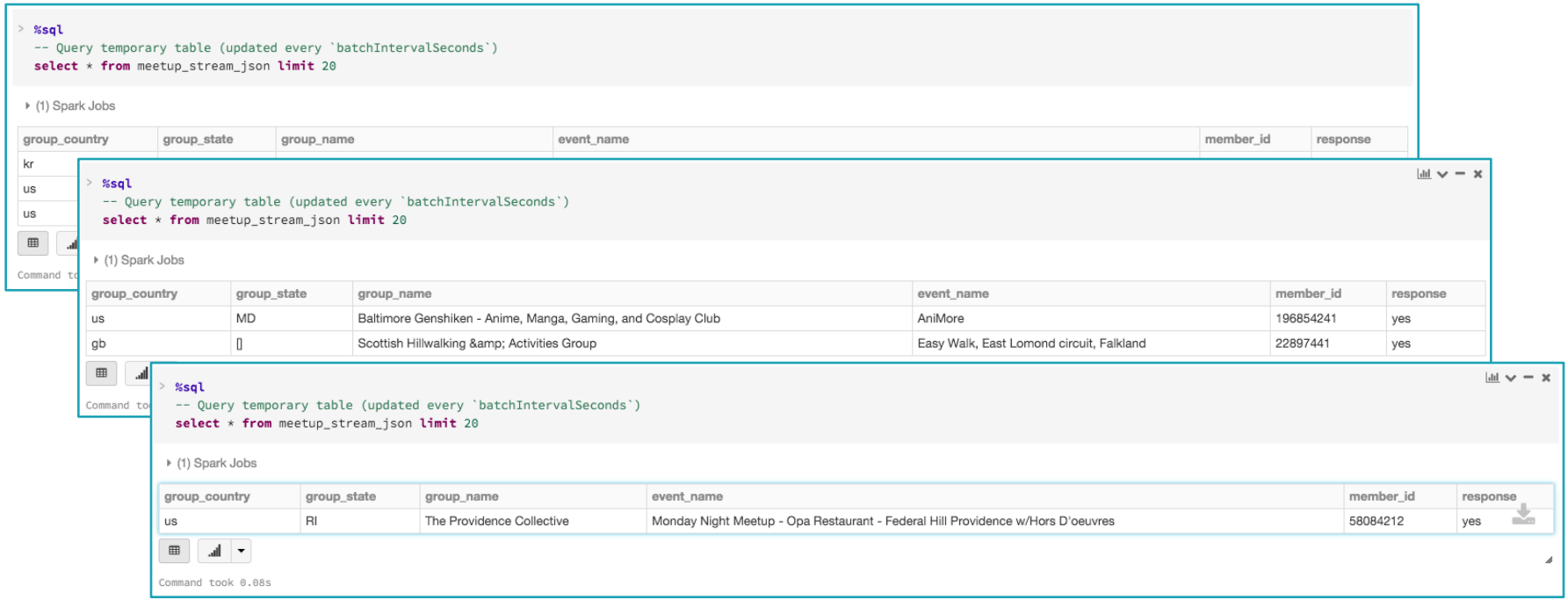
Lisa M Hamm will meetup with
Cribbage Portland/Vancouver
in Vancouver, WA
7 seconds ago

W W will meetup with
Los Angeles Free Concerts

Streaming Context Function

```
val ssc = new StreamingContext(sc, Seconds(batchIntervalSeconds))

val stream = ssc.receiverStream(new MeetupReceiver("..."))

stream.foreachRDD { rdd =>
   if (rdd.toLocalIterator.nonEmpty) {
    val sdf = sqlContext.read.json(rdd)
    sdf.select(...).registerTempTable("meetup_stream_json")
}
```

Query Streaming DataFrame



```
%sql
-- Query temporary table (updated every `batchIntervalSeconds`)
select * from meetup_stream_json limit 20
```

(1) Spark Jobs

group_country	group_state	group_name	event_name	member_id	response
kr					
us					
us					

```
%sql
-- Query temporary table (updated every `batchIntervalSeconds`)
select * from meetup_stream_json limit 20
```

(1) Spark Jobs

group_country	group_state	group_name	event_name	member_id	response
us	MD	Baltimore Genshiken - Anime, Manga, Gaming, and Cosplay Club	AniMore	196854241	yes
gb	[]	Scottish Hillwalking & Activities Group	Easy Walk, East Lomond circuit, Falkland	22897441	yes

```
%sql
-- Query temporary table (updated every `batchIntervalSeconds`)
select * from meetup_stream_json limit 20
```

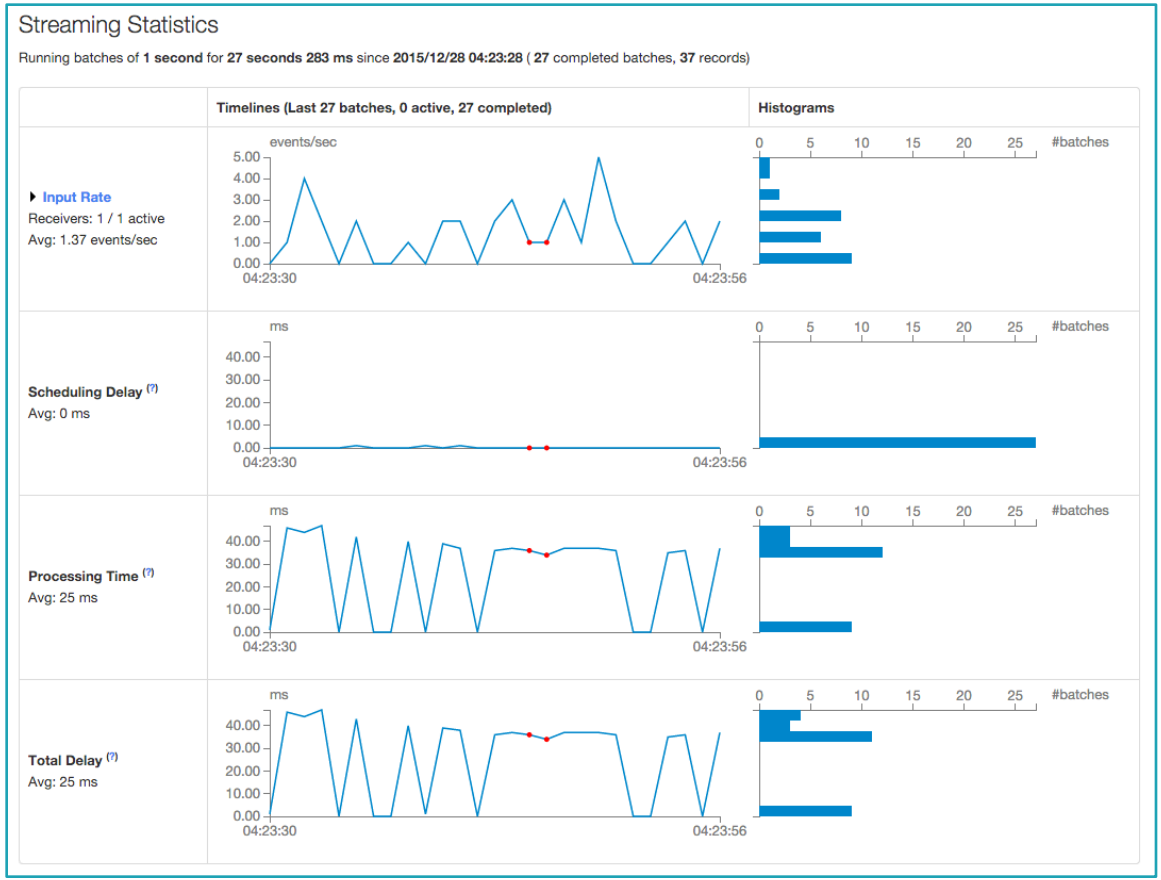
(1) Spark Jobs

group_country	group_state	group_name	event_name	member_id	response
us	RI	The Providence Collective	Monday Night Meetup - Opa Restaurant - Federal Hill Providence w/Hors D'oeuvres	58084212	yes

Command took 0.08s

databricks

Review Spark UI › Streaming Tab



databricks

Streaming Context Function Updated

```
val ssc = new StreamingContext(sc, Seconds(batchIntervalSeconds))

val stream = ssc.receiverStream(new MeetupReceiver("..."))

stream.foreachRDD { rdd =>
    if (rdd.toLocalIterator.nonEmpty) {
     val sdf = sqlContext.read.json(rdd)
     sdf.select(...).registerTempTable("meetup_stream_json")

     // Populate `meetup_stream` table
     sqlContext.sql("insert into meetup_stream select * from meetup_stream_json")
}
```

Streaming DataFrame Reports

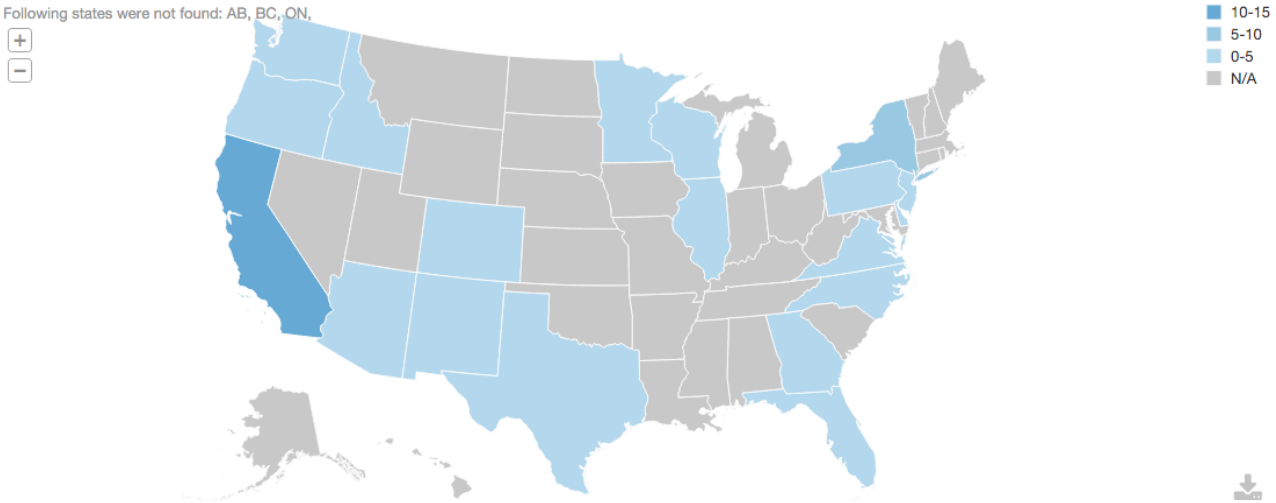
```
> select group_state, count(1) as responses from meetup_stream group by group_state
```


```
> select group_state, count(1) as responses from meetup_stream group by group_state
```

```
> select group_state, count(1) as responses from meetup_stream group by group_state
```


▸ (2) Spark Jobs

Following states were not found: AB, BC, ON,



Plot Options...

Command took 5.59s

databricks

Review Spark UI › Streaming Tab

databricks

Using mapWithState

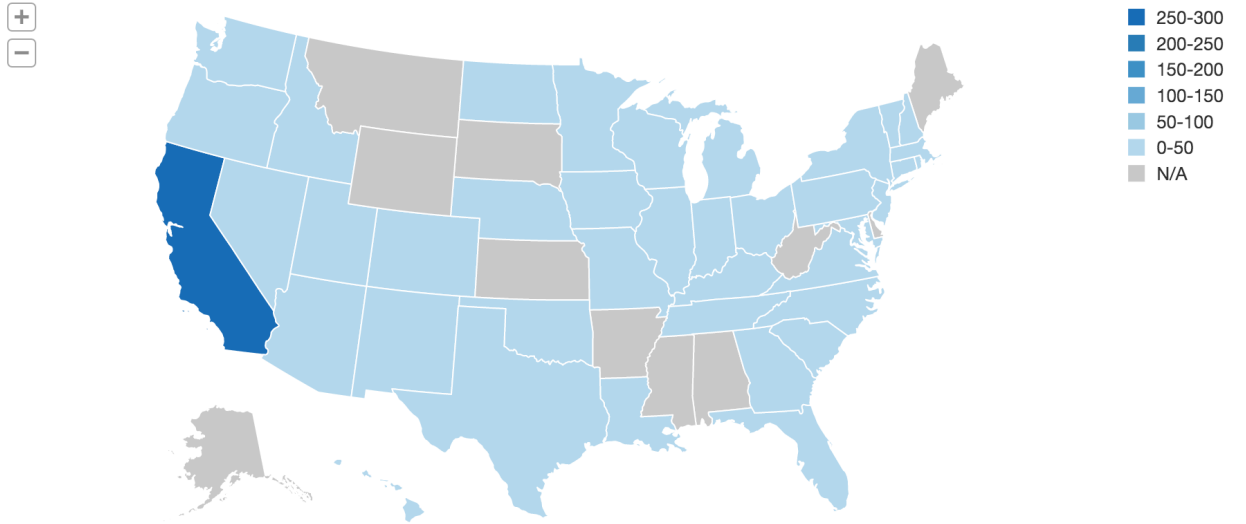
```
def trackStateFunc(batchTime: Time, key: String, value: Option[Int], state:
    State[Long]): Option[(String, Long)] = { ... }

val stateSpec = StateSpec.function(trackStateFunc _)...

val ssc = new StreamingContext(sc, Seconds(batchIntervalSeconds))

val stream = ssc.receiverStream(new MeetupReceiver("..."))

val meetupStream = stream.map(...)

val meetupStateStream = meetupStream.mapWithState(stateSpec)
...
```

Streaming Reports via mapWithState

```
> select group_state as `State Code`, count as `RSVPs` from batch_meetup_stream where group_state <> '{}' order by count desc
```


```
> select group_state as `State Code`, count as `RSVPs` from batch_meetup_stream where group_state <> '{}' order by count desc
```

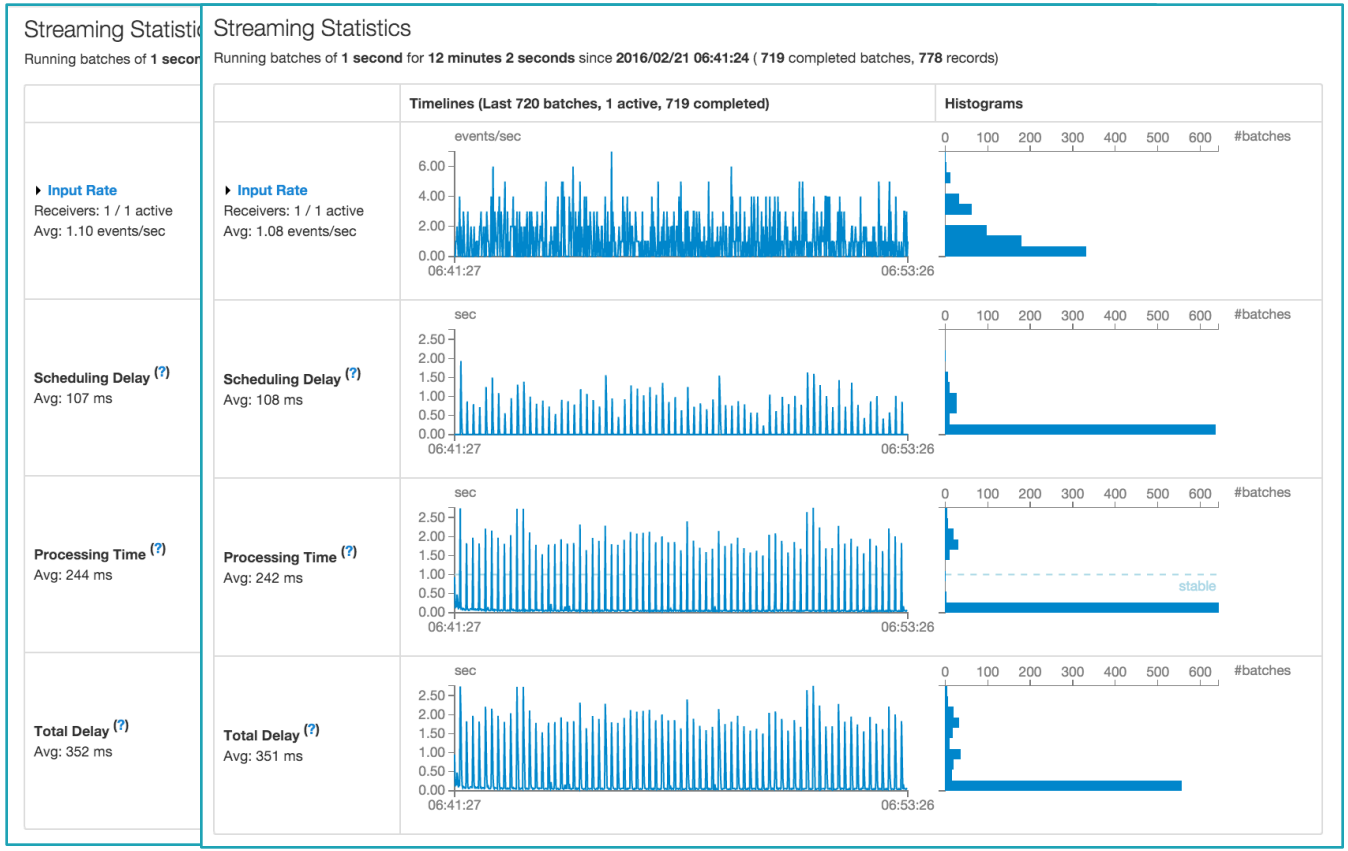

▸ (1) Spark Jobs



Plot Options...

Command took 0.03s

Review Spark UI › Streaming Tab



databricks

The simplest way to perform streaming analytics is not having to ***reason*** about streaming

Reynold Xin

Spark Summit East 2016

Day 3 Keynote

Apache Spark 2.0: Structured Streaming

High-level streaming API built on Spark SQL Engine

- Runs the same queries on DataFrames
- Event time, windowing, sessions, sources, and sinks

Unifies streaming, interactive, and batch queries

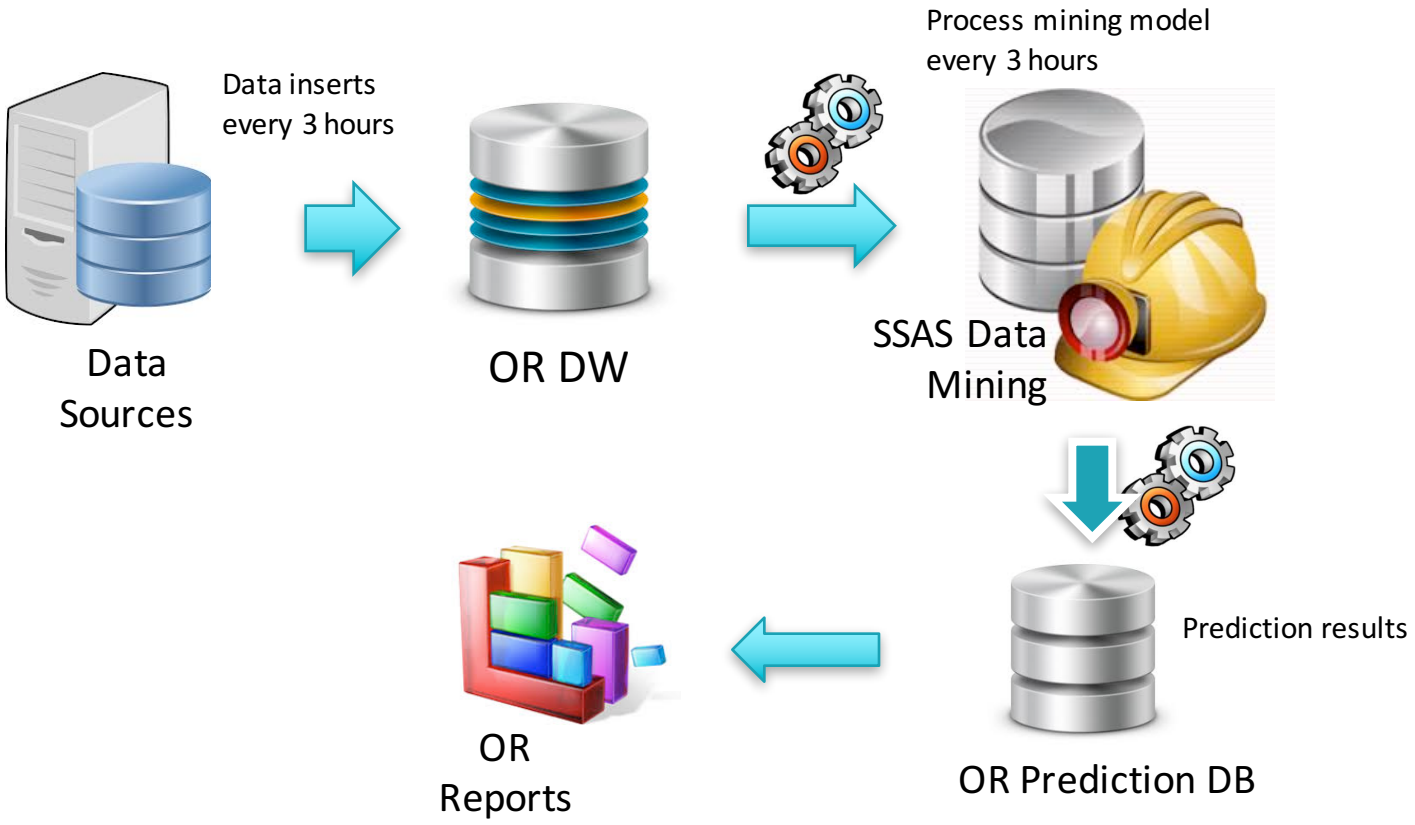
- Aggregate data in a stream, then serve using JDBC
- Change queries at runtime
- Build and apply ML mldeos

Transitioning from DW to Data Sciences

Webinar: Transitioning from Traditional DW to Spark in OR Predictive Modeling

Using Traditional Data Warehousing Techniques

Traditional Data Warehousing & Data Mining OR Predictive Model



Original Design

Multiple data sources pushing data into SQL Server and SQL Server Analysis Server Data Mining

Hand built 225 different DM modules (5 days, 15 business days ahead, 3 different groups)

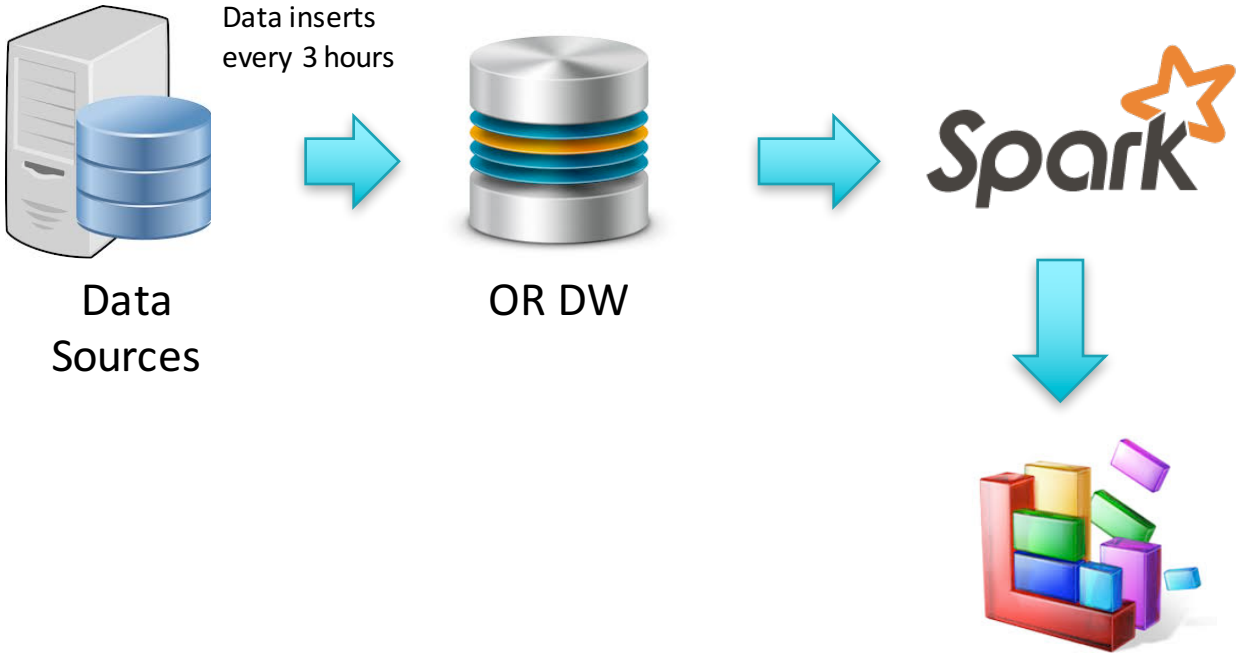
Pipeline process had to run 225 times / day (3 pools x 75 modules)

Regression Calculations

SSAS Data Mining	T-SQL Code
Intercept	R2
Mean	Adjusted R2
Coefficients	Standard Deviation
Variance	Standard Error

Taking advantage of Spark’s DW Capabilities and MLlib

OR Predictive Model in Spark



Build Data Frame

```
case class ORBookingTimeHistory(KeyID: Long, AltKeyID: Long, GroupID: Long, Case_Date: String, DayName_short: String, DayOfWeekNum: Long, DaysAhead: Long,
Day30: Long, Day20: Long, DayX: Long, Day5: Long, RecCreateDt: String)
```

```
defined class ORBookingTimeHistory
```

Command took 0.28s

```
val dfOR = sc.textFile("/mnt/cg.montreal/history/ORBookingTimeHistory.csv").map(_.split(",")).map(m => ORBookingTimeHistory(m(0).toLong, m(1).toLong,
m(2).toLong, m(3), m(4), m(5).toLong, m(6).toLong, m(7).toLong, m(8).toLong, m(9).toLong, m(10).toLong, m(11))).toDF()
```

```
dfOR: org.apache.spark.sql.DataFrame = [KeyID: bigint, AltKeyID: bigint, GroupID: bigint, Case_Date: string, DayName_short: string, DayOfWeekNum: bigint, DaysAhead: bigint,
Day30: bigint, Day20: bigint, DayX: bigint, Day5: bigint, RecCreateDt: string]
```

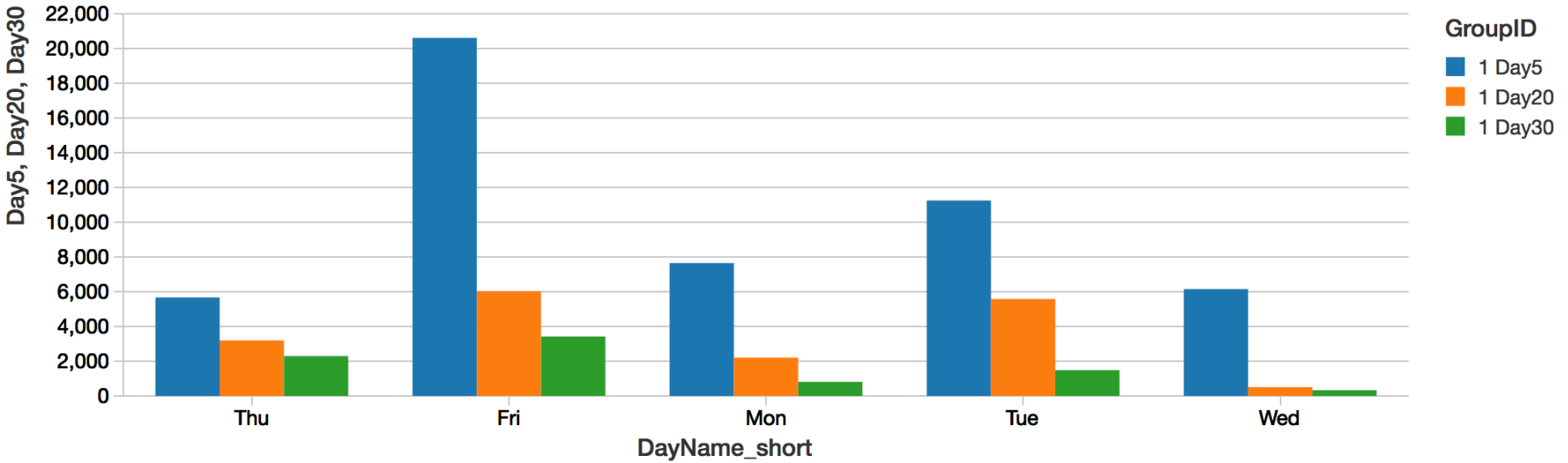
Command took 0.31s

```
dfOR.registerTempTable("ORBookingTimeHistory")
```

Command took 0.14s

```
> %sql SELECT * FROM ORBookingTimeHistory where Case_Date > '4/1/15' and GroupID = 1 limit 100;
```

▸ (2) Spark Jobs



Plot Options...

Command took 1.21s

databricks

Review Features and Labels (i.e. x and y variables)

```
> val df = sqlContext.sql("SELECT GroupID, DaysAhead, Day30, Day20, DayX, Day5, KeyID FROM ORBookingTimeHistory WHERE GroupID = 1 AND DaysAhead = 1")
df.count
```

```
df: org.apache.spark.sql.DataFrame = [GroupID: bigint, DaysAhead: bigint, Day30: bigint, Day20: bigint, DayX: bigint, Day5: bigint, KeyID: bigint]
res1: Long = 126
```

Command took 1.11s

```
> df.collect.take(10).foreach(println)
```

```
[1,1,27,162,867,747,2878591]
[1,1,48,207,573,519,2878606]
[1,1,264,369,864,615,2878621]
[1,1,333,432,1140,858,2878636]
[1,1,207,300,840,750,2878651]
[1,1,201,318,915,633,2878666]
[1,1,129,225,789,642,2878681]
[1,1,96,189,648,597,2878696]
[1,1,105,171,435,393,2878831]
[1,1,48,147,594,408,2878846]
```

Command took 0.44s

Setup MLLib

```
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LinearRegressionWithSGD
```

```
import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LinearRegressionWithSGD
```

Command took 0.10s

Defintions

- Features: Day30 (2), Day20 (3), DayX (4),
- Label: Day5 (5)

```
val parsedData = data.map { p =>
  val label = p(5).toString.toDouble
  val features = Array(p(2), p(3), p(4)) map (_.toString.toDouble)
  LabeledPoint(label, Vectors.dense(features))
}
```

```
parsedData: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[3383] at map at
```

Command took 0.32s

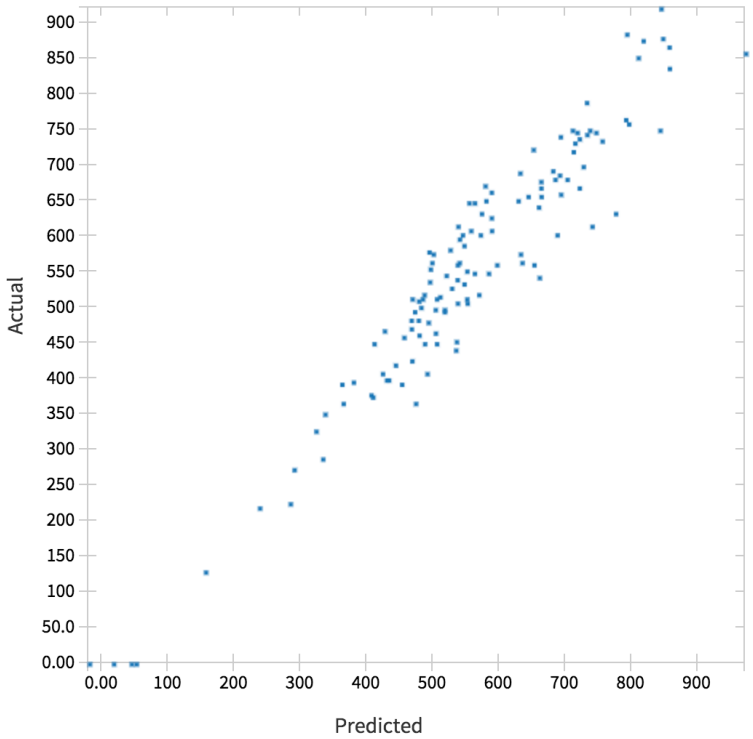
Feature Scaling (Regularization)

```
> import org.apache.spark.mllib.feature.StandardScaler
  val scaler = new StandardScaler(withMean = true, withStd = true).fit(parsedData.map(x => x.features))
  val scaledData = parsedData.map(x => LabeledPoint(x.label, scaler.transform(Vectors.dense(x.features.toArray))))

  import org.apache.spark.mllib.feature.StandardScaler
  scaler: org.apache.spark.mllib.feature.StandardScalerModel = org.apache.spark.mllib.feature.StandardScalerModel@4d50448
  scaledData: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] = MapPartitionsRDD[3386] at map at <console>:34
  Command took 0.43s
```

Train the Model

```
> val numIterations = 3000
  val alpha = 0.1
  val algorithm = new LinearRegressionWithSGD()
  algorithm.setIntercept(true)
  algorithm.optimizer.setNumIterations(numIterations)
  algorithm.optimizer.setStepSize(alpha)
  val model = algorithm.run(scaledData)


  numIterations: Int = 3000
  alpha: Double = 0.1
  algorithm: org.apache.spark.mllib.regression.LinearRegressionWithSGD = org.apache.spark.mllib.regression.LinearRegressionWithSGD@19bc16bf
  model: org.apache.spark.mllib.regression.LinearRegressionModel = (weights=[-7.266071208249522,25.566012025772352,156.2382377166797], intercept=554.6559567707233)
  Command took 56.74s
```

```
> %sql select `_1` as Actual, `_3` as Predicted from LRResults;
```



Plot Options...

Command took 0.27s

Apply Previous Linear Regression with SGD model

For GroupID = 1, DaysAhead = 1 (KeyID = 151), predicted booking is 483

```
> val valuesAndPredsFuture = scaledFutureData.map { point =>
    val prediction = model.predict(point.features)
    (point.label, point.features, prediction)
  }
valuesAndPredsFuture.take(10).foreach({case (v, f, p) =>
      println(s"Features: ${f}, Predicted: ${p}, Actual: ${v}")})
```

```
Features: [-0.24464267654515323,0.04851883110991891,-0.5092994337292287], Predicted: 478.1019349027521, Actual: 546.0
valuesAndPredsFuture: org.apache.spark.rdd.RDD[(Double, org.apache.spark.mllib.linalg.Vector, Double)] = MapPartitionsRDD[9524]
```

```
Command took 0.46s
```

MLLib LinearRegressionSGD predcition: 478.1019349027521

Key Learnings when Migrating from Traditional DW to Spark

Transitioning to the Cloud

Beth Israel Deaconess Medical Center is increasingly moving to cloud infrastructure services with the hopes of closing its data center when the hospital's lease is up in the next five years. ***CIO John Halamka*** *says he's decommissioning HP and Dell servers as he moves more of his compute workloads to Amazon Web Services, where he's currently using 30 virtual machines to test and develop new applications. "It is no longer cost effective to deal with server hosting ourselves because our challenge isn't real estate, it's power and cooling," he says.*

Transitioning to the Cloud

Need time for engineers, analysts, and data scientists to learn how to build for the cloud

Build for security right from start – process heavy, a lot of documentation, audits / reviews

Differentiating data engineers and engineers (REST APIs, services, elasticity, etc.)

Transitioning to Spark

No more stored procedures or indexes

- Good for Spark SQL, services design

Prototype, prototype, prototype

- Leverage existing languages and skill sets
- Leverage the MOOCs and other Spark training
- Break down the silos of data engineers, engineers, data scientists, and analysts

Transitioning DW to Spark

Understand Partitioning, Broadcast Joins, and Parquet

Not all Hive functions are available in Spark (99% of the time that is okay) due to Hive context

Don’t limit yourself to build star-schemas / snowflake schemas

Expand outside of traditional DW: machine learning, streaming

Movie Recommendations with MLlib

Notebook will be available soon

Movie Recommendations

- Reference blog post: Scalable Collaborative Filtering with Spark MLlib
- Based on the Spark Summit 2014 Training Archive Movie Recommendation with MLlib Hands-on Exercises
- Datasets:
  - The MovieLens dataset which contains 72,000 users on 10,000 movies with 10 million ratings.
  - You can find the dataset within the Spark Summit 2014 Training Archive USB drive (download) within the *$usb/data/moviellens/large* folder.

Review mounted MovieLens datasets

```
> display(dbutils.fs.ls("/mnt/tardis6/movielens/"))
```

▸ (3) Spark Jobs

path	name	size
dbfs:/mnt/tardis6/movielens/movies/	movies/	0
dbfs:/mnt/tardis6/movielens/personalRatings/	personalRatings/	0
dbfs:/mnt/tardis6/movielens/ratings/	ratings/	0
dbfs:/mnt/tardis6/movielens/tags/	tags/	0

Command took 6.67s

Load data

```
> from pyspark.mllib.recommendation import ALS, Rating

  # Parses a rating record in MovieLens format userId::movieId::rating::timestamp .
  def parseRating(line):
      fields = line.strip().split("::")
      return long(fields[3]) % 10, Rating(int(fields[0]), int(fields[1]), float(fields[2]))

  # Parses a movie record in MovieLens format movieId::movieTitle .
  def parseMovie(line):
      fields = line.strip().split("::")
      return int(fields[0]), fields[1]


  # ratings is an RDD of (last digit of timestamp, (userId, movieId, rating))
  ratings = sc.textFile("/mnt/tardis6/movielens/ratings/ratings.dat").map(lambda l: parseRating(l))

  # movies is an Dictionary RDD of (movieId, movieTitle)
  movies = dict(sc.textFile("/mnt/tardis6/movielens/movies/movies.dat").map(lambda l: parseMovie(l)).collect())

  # personalRatings is an RDD of (last digit of timestamp, (userId, movieId, rating))
  personalRatings = sc.textFile("/mnt/tardis6/movielens/personalRatings/personalRatings.txt").map(lambda l: parseRating(l))
```

Calculate summary statistics

```
> # Calculate the number of ratings, distinct users, and distinct movies
  numRatings = ratings.count()
  numUsers = ratings.values().map(lambda r: r[0]).distinct().count()
  numRatedMovies = ratings.values().map(lambda r: r[1]).distinct().count()

  # Number of movies
  numMovies = sum(len(v) for v in movies.itervalues())

  # Specify movieId = 1
  movieId1 = movies[1]

  # Print out the values
  print "Got %d ratings from %d users on %d movies." % (numRatings, numUsers, numRatedMovies)
  print "There are a total of %d movies with movieId = 1 as '%s'." % (numMovies, movieId1)
```

▸ (3) Spark Jobs

```
Got 10000054 ratings from 69878 users on 10677 movies.
There are a total of 277363 movies with movieId = 1 as 'Toy Story (1995)'.
```

Command took 102.34s

Create the Training and Test Datasets

```
> # Randomly split data between training and test
  training, test = ratings.map(lambda x: x[1]).randomSplit([4, 1], 8)

  # Calculate summary statistics
  numTraining = training.count()
  numTest = test.count()

  print "Training: %d, test: %d" % (numTraining, numTest)
```

▸ (2) Spark Jobs

```
Training: 8000011, test: 2000043
```

Command took 65.19s

Determining the best ALS model

```
model = ALS.train(training, rank, numIteration, lambda_)

# Get predicted ratings on test existing user-product pairs
testData = test.map(lambda p: (p.user, p.product))
predictions = model.predictAll(testData).map(lambda r: ((r.user, r.product), r.rating))
ratingsTuple = test.map(lambda r: ((r.user, r.product), r.rating))
scoreAndLabels = predictions.join(ratingsTuple).map(lambda tup: tup[1])

# Instantiate regression metrics to compare predicted and actual ratings
metrics = RegressionMetrics(scoreAndLabels)

# Print out RMSE and R2 based on different variables
testRmse = metrics.rootMeanSquaredError
testR2 = metrics.r2
print(" >> Rank: %s, Lambda: %s, Iterations: %s, RMSE: %s, R2: %s" % (rank, lambda_, numIteration, testRmse, testR2))
```

TODO: Add personal recommendations

On-time Flight Performance with GraphFrames

On-time Flight Performance with GraphFrames Notebook

Why Graph?

- Graph structures are a more intuitive approach to many classes of data problems: social networks, restaurant recommendations, or flight paths, etc.
- Easier to understand these data problems within the context of graph structures: vertices, edges, and properties.
- e.g. flight data analysis is a classic graph problem:
  - airports are represented by *vertices*
  - flights are represented by *edges*.
  - numerous *properties* associated with these flights including but not limited to departure delays, plane type, and carrier.

Preparing the Flight Dataset

```
# Import graphframes (from Spark-Packages)
from graphframes import *

# Create Vertices (airports) and Edges (flights)
tripVertices = airports.withColumnRenamed("IATA", "id").distinct()
tripEdges = departureDelays.select("tripid", "delay", "src", "dst",
   "city_dst", "state_dst")

# This GraphFrame builds upon the vertices and edges
# based on our trips (flights)
tripGraph = GraphFrame(tripVertices, tripEdges)
```

Viewing the edges (flights between airports)

```
> # Edges
  #  The edges of our graph are the flights between airports
  display(tripEdges)
```

▸ (1) Spark Jobs

tripid	delay	src	dst	city_dst	state_dst
3010630	4	BFL	IAH	Houston	TX
3020630	-3	BFL	IAH	Houston	TX
3021124	27	BFL	IAH	Houston	TX
3030630	-3	BFL	IAH	Houston	TX
3031124	34	BFL	IAH	Houston	TX
3040630	-8	BFL	IAH	Houston	TX
3041124	105	BFL	IAH	Houston	TX
3050630	-11	BFL	IAH	Houston	TX
3051124	5	BFL	IAH	Houston	TX

Showing the first 1000 rows.

Simple Queries against the GraphFrame

```
print "Airports: %d" % tripGraph.vertices.count()
print "Trips: %d" % tripGraph.edges.count()
```

```
Airports: 279
Trips: 1361141
```

What flights departing from SFO are most likely to have significant delays?

```
tripGraph.edges\
  .filter("src = 'SFO' and delay > 0")\
  .groupBy("src", "dst")\
  .avg("delay")\
  .sort(desc("avg(delay)"))
```

```
> display(tripGraph.edges.filter("src = 'SFO' and delay > 0").groupBy("src",
  "dst").avg("delay").sort(desc("avg(delay)")))
```

▸ (1) Spark Jobs

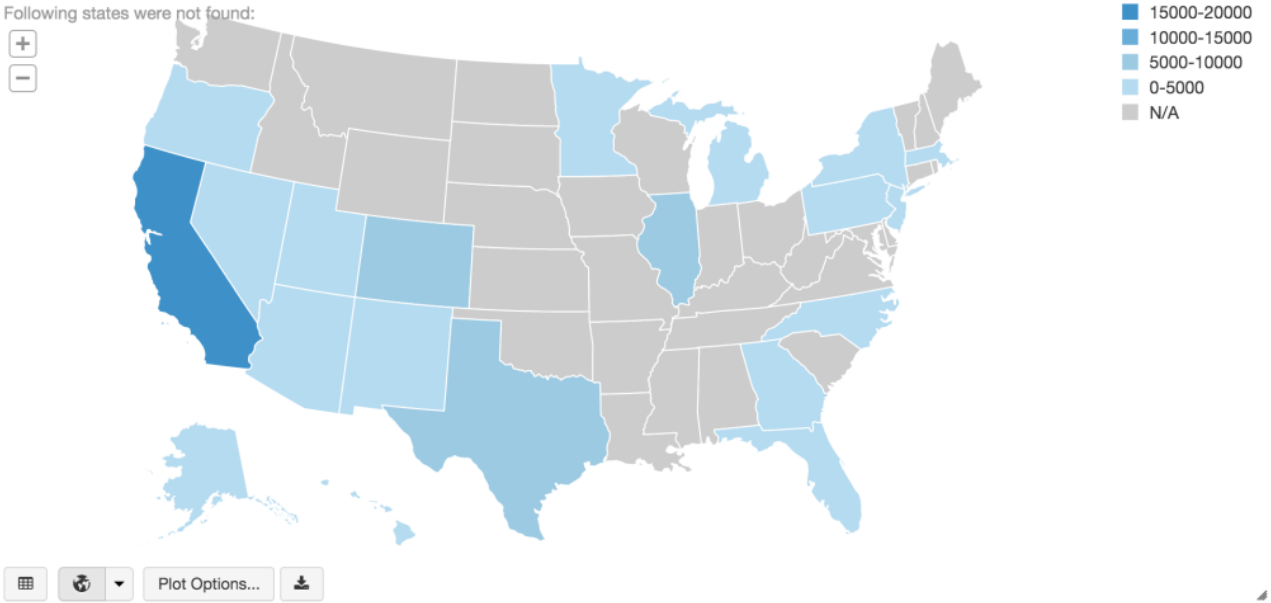
src	dst	avg(delay)
SFO	OKC	59.073170731707314
SFO	JAC	57.13333333333333
SFO	COS	53.976190476190474
SFO	OTH	48.09090909090909
SFO	SAT	47.625
SFO	MOD	46.80952380952381

What destination states tend to have significant delays departing from SEA?

```
> # States with the longest cummulative delays (with individual delays > 100 minutes) (origin: Seattle)
  display(tripGraph.edges.filter("src = 'SEA' and delay > 100"))
```

▸ (3) Spark Jobs

Following states were not found:



databricks

Using Motif Finding to understand flight delays

What delays might we blame on SFO?

```
motifs = tripGraphPrime.find("(a)-[ab]->(b); (b)-[bc]->(c)")\
  .filter("(b.id = 'SFO') and (ab.delay > 500 or bc.delay > 500)
    and bc.tripid > ab.tripid and bc.tripid > ab.tripid +
    10000")
display(motifs)
```

Using Motif Finding to understand flight delays

Below is an abridged subset from this query where the columns are the respective motif keys.

a	ab	b	bc	c
Houston (IAH)	IAH -> SFO (-4) [1011126]	San Francisco (SFO)	SFO -> JFK (536) [1021507]	New York (JFK)
Tuscon (TUS)	TUS -> SFO (-5) [1011126]	San Francisco (SFO)	SFO -> JFK (536) [1021507]	New York (JFK)

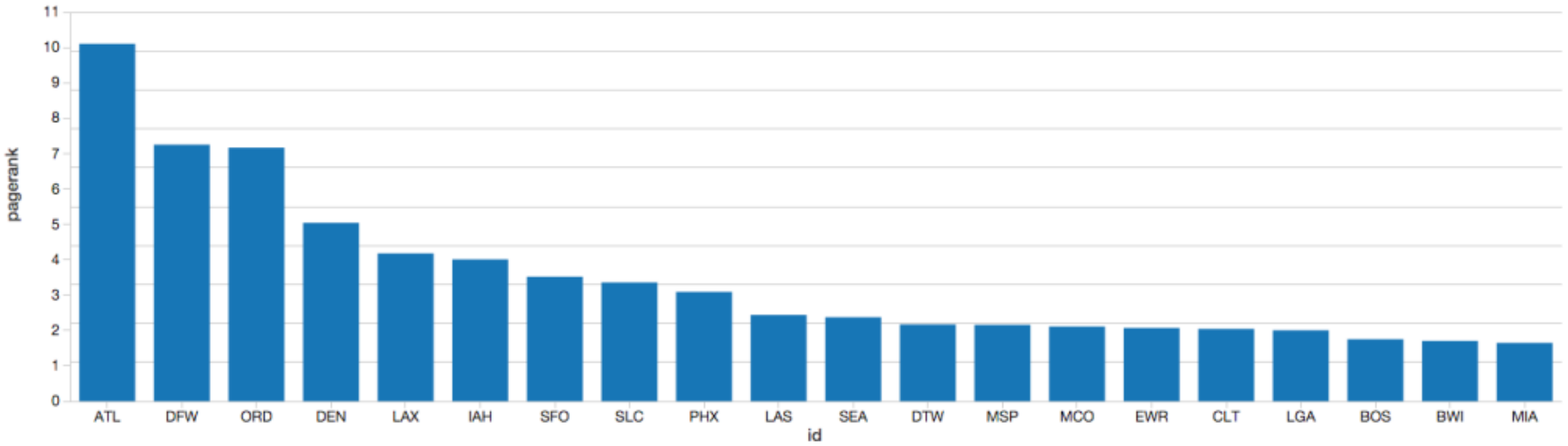
Using PageRank to find the most important airport

```
# Determining Airport ranking of importance using pageRank
ranks = tripGraph.pageRank(resetProbability=0.15, maxIter=5)

display(ranks.vertices.orderBy(ranks.vertices.pagerank.desc()).
  limit(20))
```

Using PageRank to find the most important airport

Corresponding to the fact that Atlanta is the busiest airport in the world by passenger traffic

▸ (6) Spark Jobs

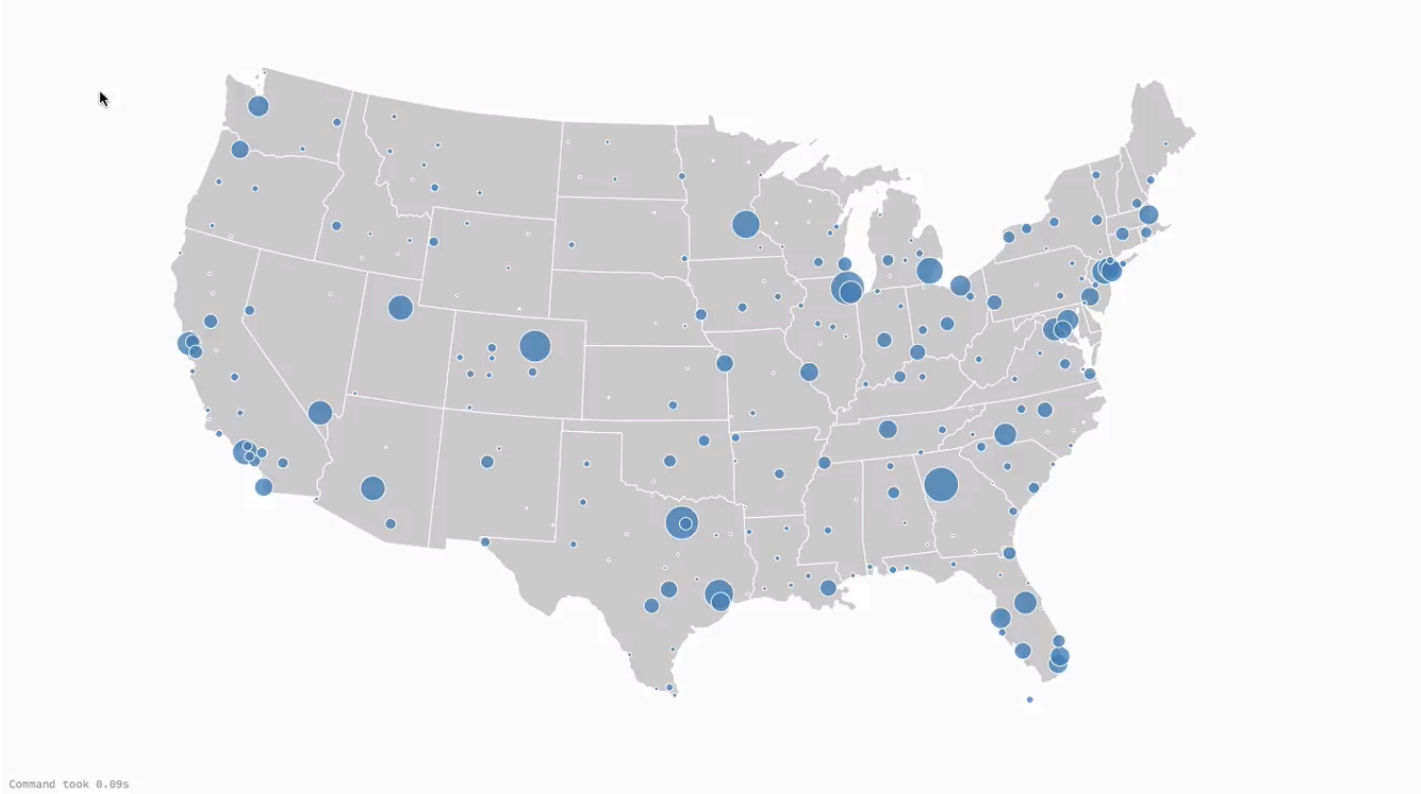


Determining Flight Connections

```
filteredPaths = tripGraph.bfs(
  fromExpr = "id = 'SFO'",
  toExpr = "id = 'BUF'",
  maxPathLength = 2
)
display(filteredPaths)
```

from	v1	to
SFO	MSP (Minneapolis)	BUF
SFO	EWR (Newark)	BUF
SFO	JFK (New York)	BUF
SFO	ORD (Chicago)	BUF
SFO	ATL (Atlanta)	BUF
SFO	LAS (Las Vegas)	BUF
SFO	BOS (Boston)	BUF
…	…	…

Visualizing Flights Using D3



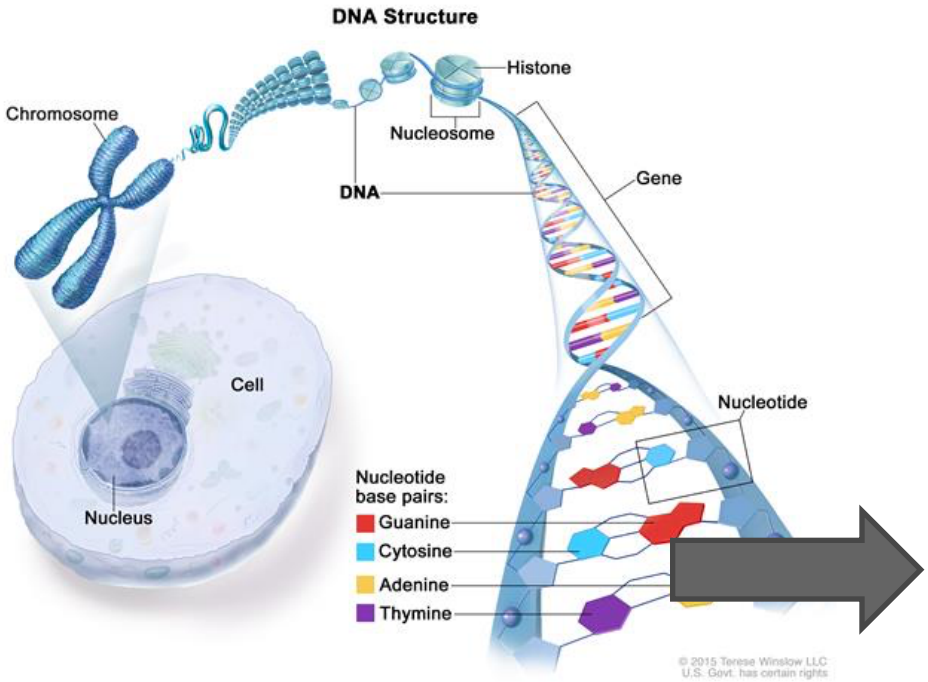
Predicting Population Sample via Genomic Sequences and Spark MLlib

Notebook: Will be available soon

Background

Genome Sequencing

Genome Sequencing



In a sense, a genome sequence is simply a very long string of letters in a mysterious language.

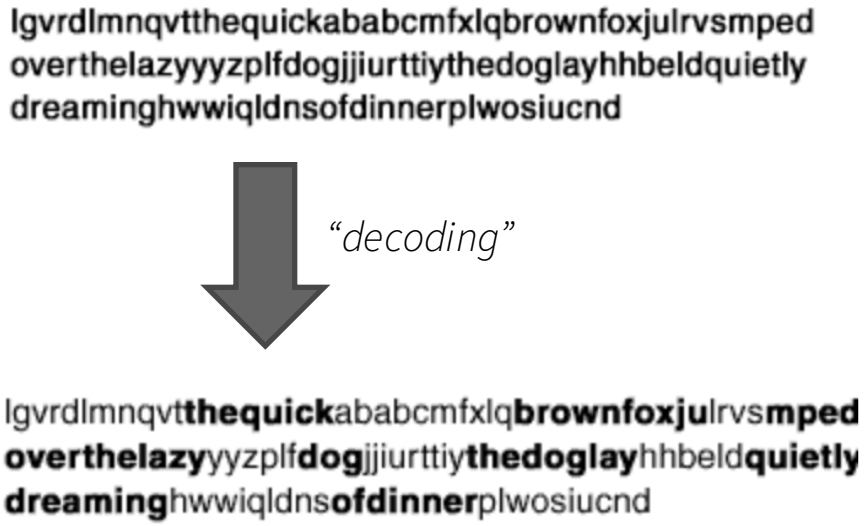
AGTCCGCGAATACAGGCTCGGT

databricks

Sources: NCI Dictionary of Cancer Terms, What is a genome

Genome Sequencing Analogy

- Genome sequence is a set of letters representing nucleotides (A, G, C, T, U)
- Need to extract meaning out of it.
- e.g. The sentence to the right had words and various random letters included
- We need to extract the words from the noise

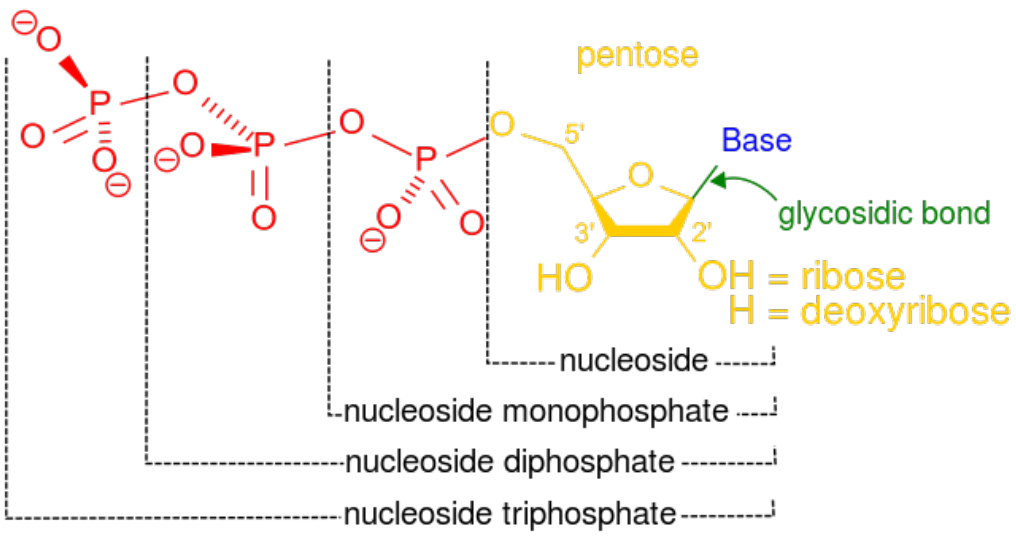


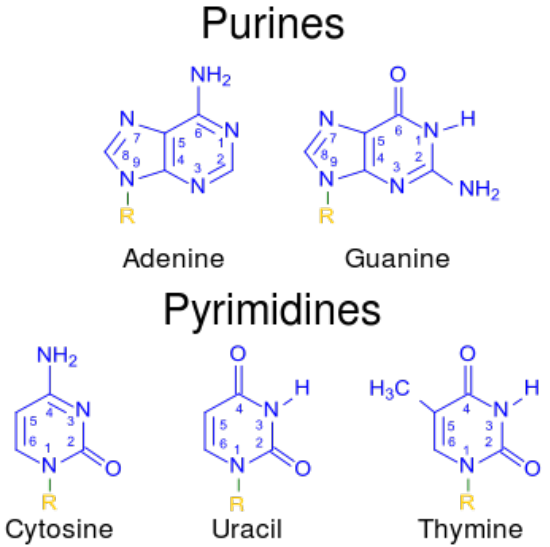
"The quick brown fox jumped over the lazy dog. The dog lay quietly dreaming of dinner."

databricks

Source: What is a genome

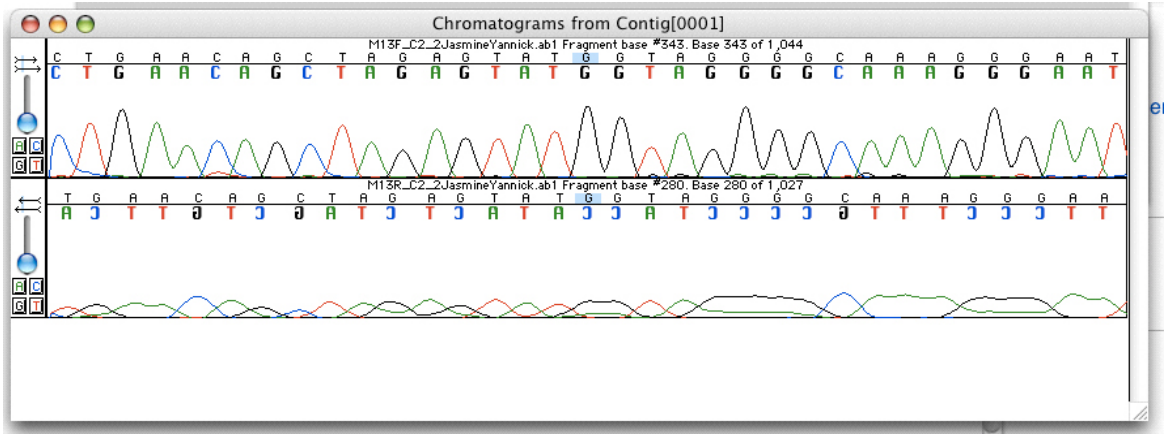
Nucleotides





Source: [Wikipedia: Nucleotide]

Whole Genome Sequencing



Electropherograms are commonly used to sequence portions of genomes

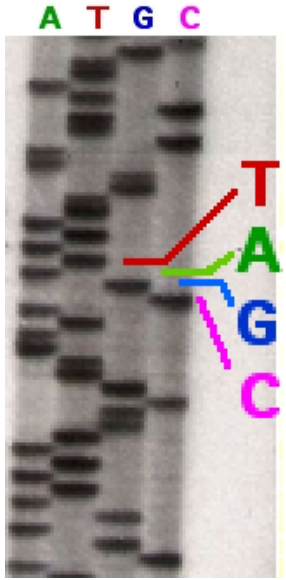
High throughput sequencing through nanopore technology, fluorophere technology, pyrosequencing, etc.

Source: Whole Genome Sequencing

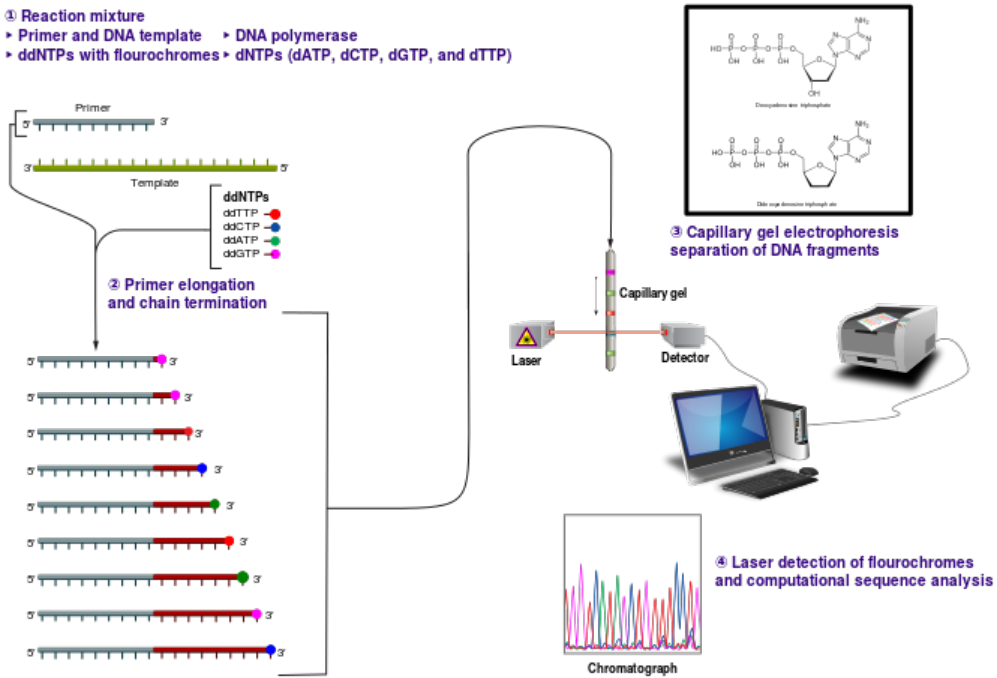
Background

Genome Sequencing Techniques

Sanger Sequencing Technique

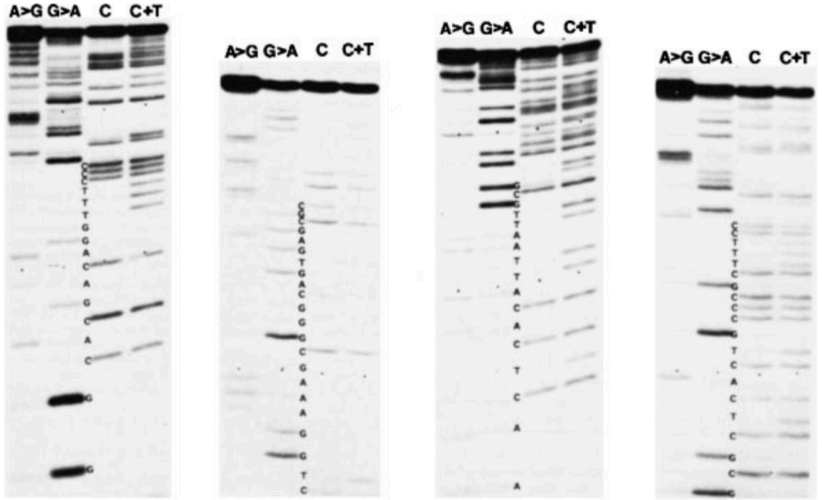


Sanger method is the classic technique and still remains in wide use, for smaller-scale projects, validation of Next-Gen results and for obtaining especially long contiguous DNA sequence reads (>500 nucleotides).



Source: Review of DNA Sequencing Techniques, Sanger Sequencing

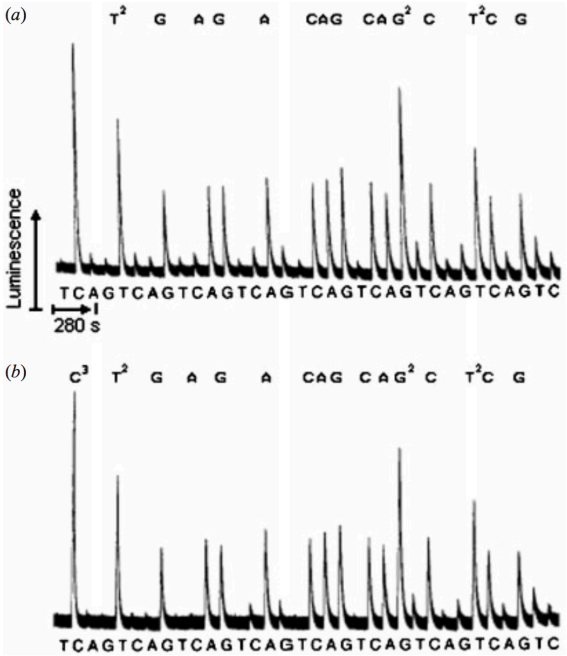
Maxam and Gilbert Sequencing Technique

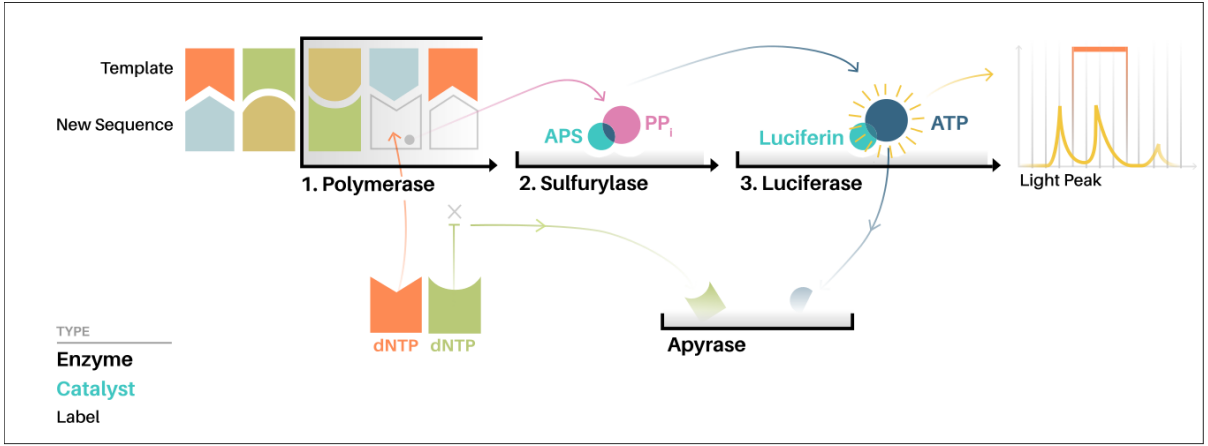


Autoradiograph of a sequencing gel of the complementary strands of a 64-bp DNA fragment using Maxam and Gilbert sequencing method based on chemical degradation

Source: Review of DNA Sequencing Techniques

Pyrosequencing Technique





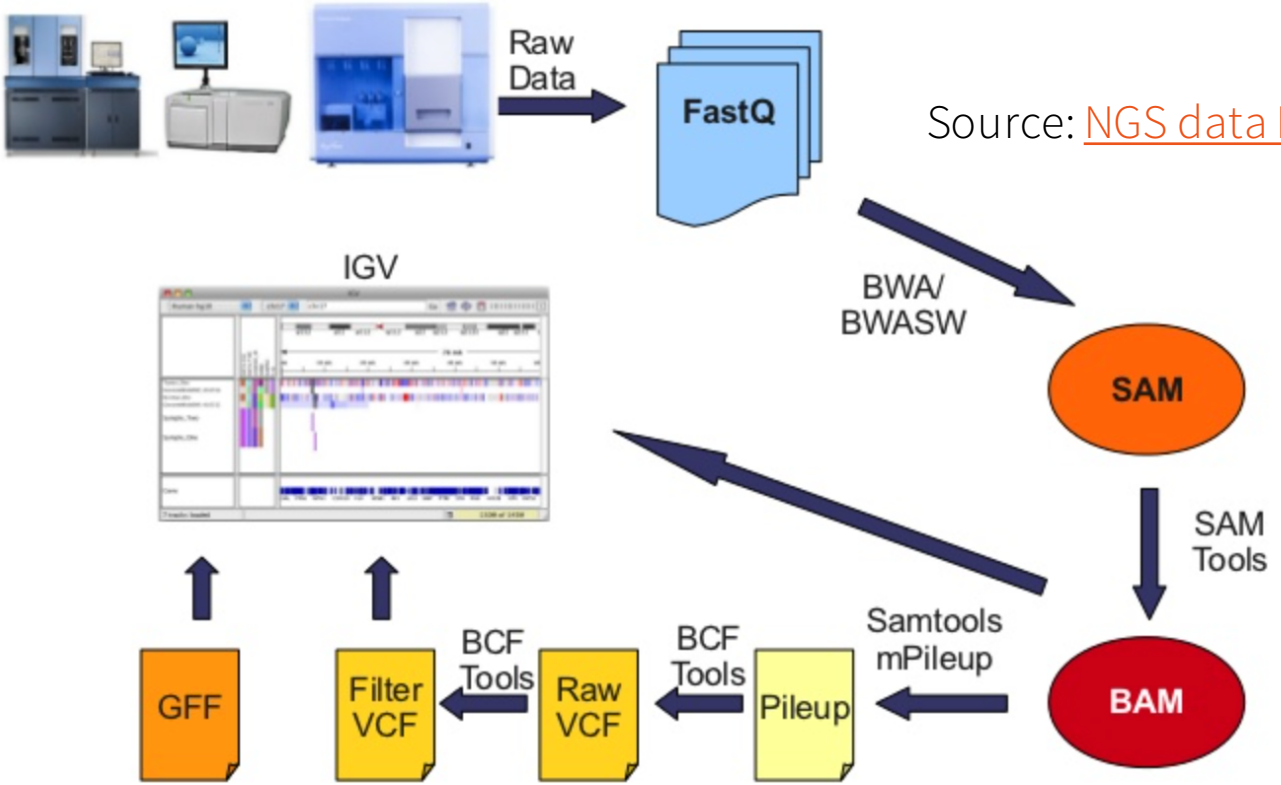
The pyrosequencing method is based on detecting the activity of DNA polymerase (a DNA synthesizing enzyme) with another chemoluminescent enzyme.

Source: Review of DNA Sequencing Techniques, Pyrosequencing

Background

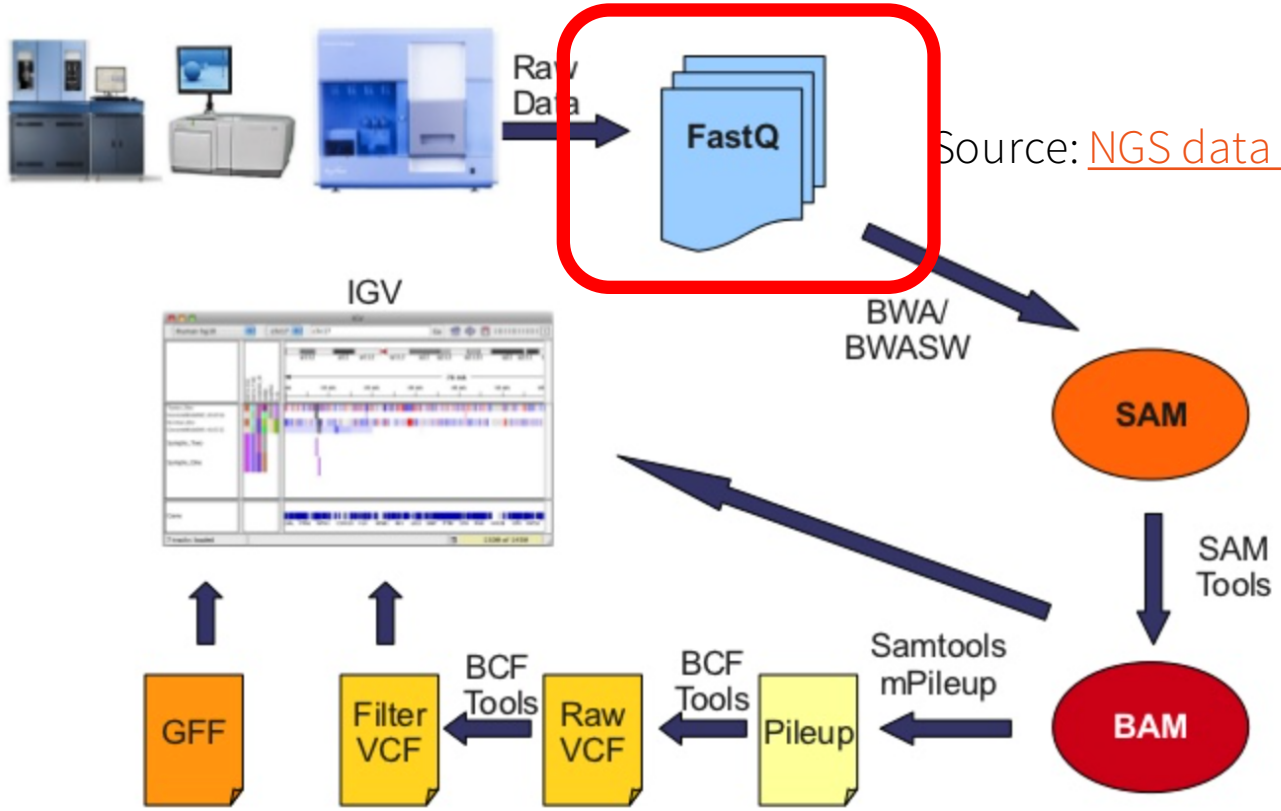
Sequence to Variation Workflow

Sequence to Variation Workflow



Source: NGS data Preprocessing

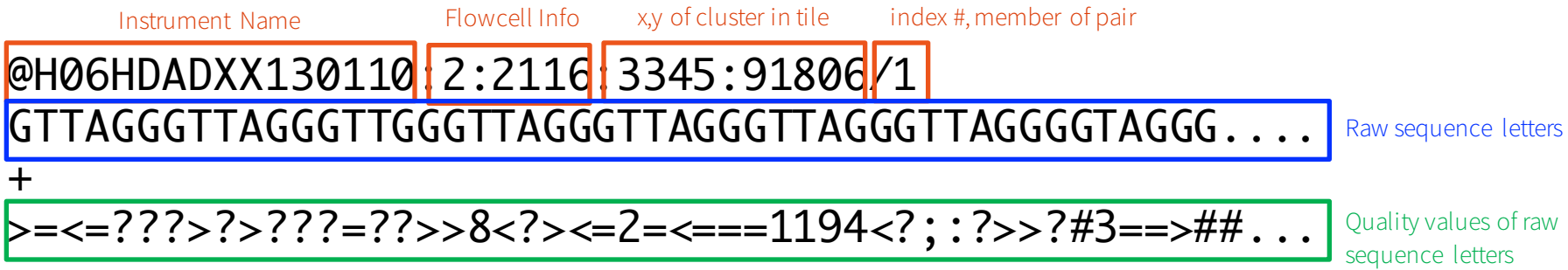
Sequence to Variation Workflow



Source: NGS data Preprocessing

FASTQ Format

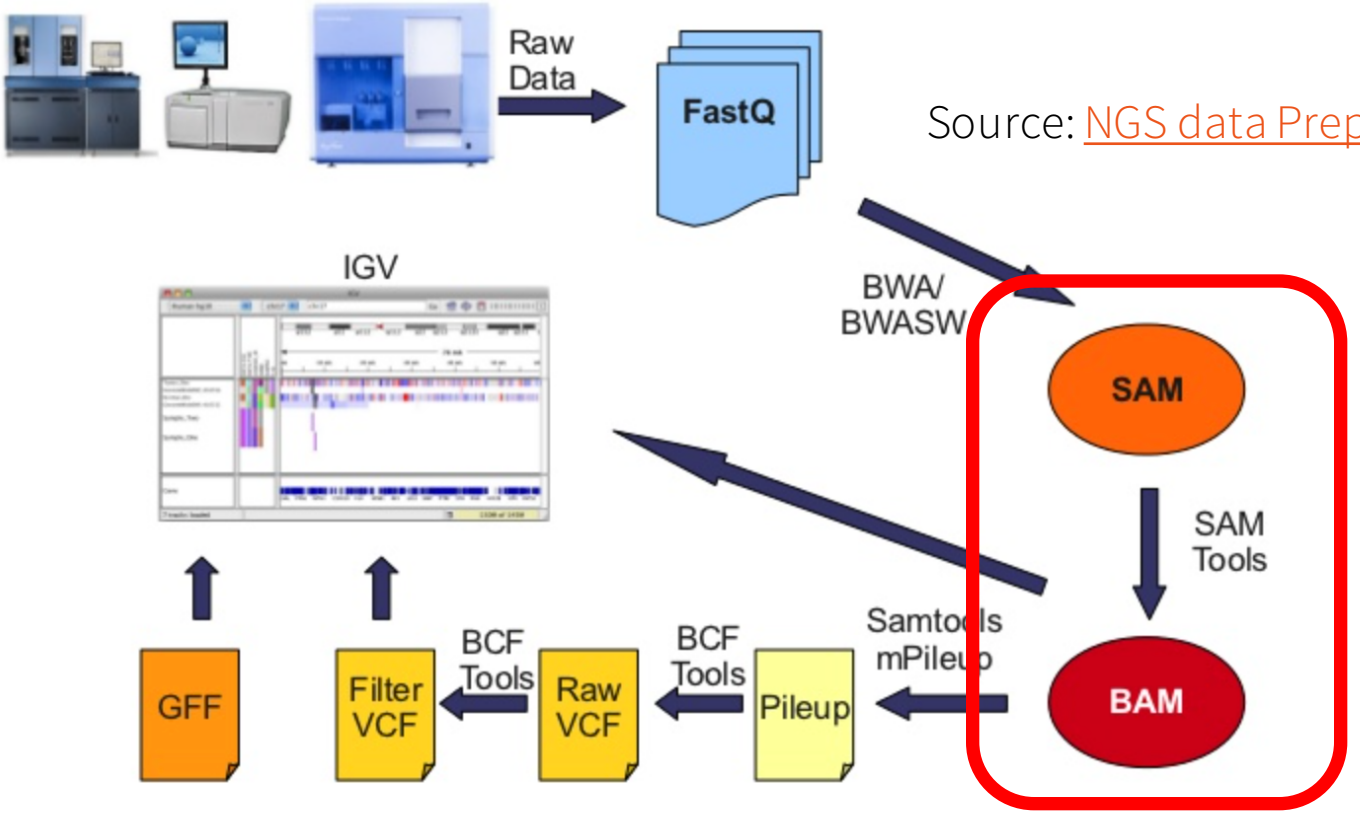
Illumina sequence identifiers



FASTQ format is a text-based format for storing both a biological sequence (usually nucleotide sequence) and its corresponding quality scores. Both the sequence letter and quality score are each encoded with a single ASCII character for brevity.

Source: FASTQ Format

Sequence to Variation Workflow



Source: NGS data Preprocessing

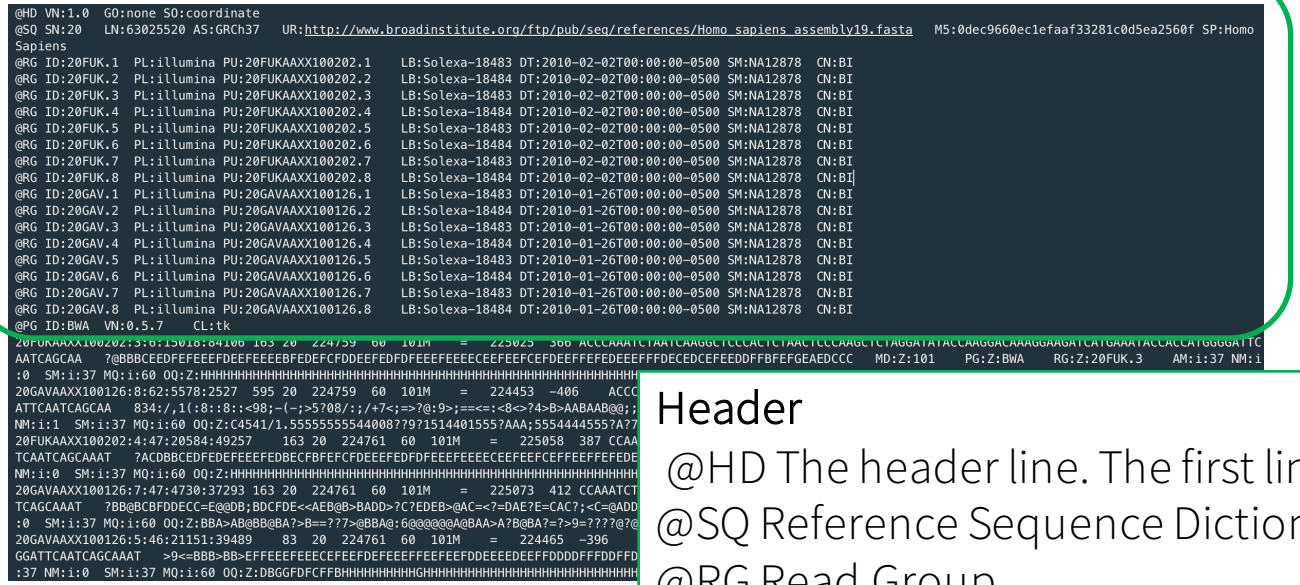
More Info on NA12878

SAM/BAM: NA12878.sam

```
@HD VN:1.0  GO:none SO:coordinate
@SQ SN:20   LN:63025520 AS:GRCh37   UR:http://www.broadinstitute.org/ftp/pub/seq/references/Homo_sapiens_assembly19.fasta   M5:0dec9660ec1efaaf33281c0d5ea2560f SP:Homo
Sapiens
@RG ID:20FUK.1  PL:illumina PU:20FUKAAXX100202.1    LB:Solexa-18483 DT:2010-02-02T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20FUK.2  PL:illumina PU:20FUKAAXX100202.2    LB:Solexa-18484 DT:2010-02-02T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20FUK.3  PL:illumina PU:20FUKAAXX100202.3    LB:Solexa-18483 DT:2010-02-02T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20FUK.4  PL:illumina PU:20FUKAAXX100202.4    LB:Solexa-18484 DT:2010-02-02T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20FUK.5  PL:illumina PU:20FUKAAXX100202.5    LB:Solexa-18483 DT:2010-02-02T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20FUK.6  PL:illumina PU:20FUKAAXX100202.6    LB:Solexa-18484 DT:2010-02-02T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20FUK.7  PL:illumina PU:20FUKAAXX100202.7    LB:Solexa-18483 DT:2010-02-02T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20FUK.8  PL:illumina PU:20FUKAAXX100202.8    LB:Solexa-18484 DT:2010-02-02T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20GAV.1  PL:illumina PU:20GAVAAXX100126.1    LB:Solexa-18483 DT:2010-01-26T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20GAV.2  PL:illumina PU:20GAVAAXX100126.2    LB:Solexa-18484 DT:2010-01-26T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20GAV.3  PL:illumina PU:20GAVAAXX100126.3    LB:Solexa-18483 DT:2010-01-26T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20GAV.4  PL:illumina PU:20GAVAAXX100126.4    LB:Solexa-18484 DT:2010-01-26T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20GAV.5  PL:illumina PU:20GAVAAXX100126.5    LB:Solexa-18483 DT:2010-01-26T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20GAV.6  PL:illumina PU:20GAVAAXX100126.6    LB:Solexa-18484 DT:2010-01-26T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20GAV.7  PL:illumina PU:20GAVAAXX100126.7    LB:Solexa-18483 DT:2010-01-26T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20GAV.8  PL:illumina PU:20GAVAAXX100126.8    LB:Solexa-18484 DT:2010-01-26T00:00:00-0500 SM:NA12878  CN:BI
@PG ID:BWA  VN:0.5.7    CL:tk
20FUKAAXX100202:3:6:15018:84106 163 20  224759  60  101M    =   225025  366 ACCCAAATCTAATCAAGGCTCCCACTCTAACTCCCAAGCTCTAGGATATACCAAGGACAAAGGAAGATCATGAAATACCACCATGGGGATTC
AATCAGCAA   ?@BBBCEEDFEFEEEFDEEFEEEEBFEDEFCFDDEEFEDFDFEEEFEEEECEEFEEFCEFDEEFFEFEDEEEFFFDECEDCEFEEDDFFBFEFGEAEDCCC   MD:Z:101    PG:Z:BWA    RG:Z:20FUK.3    AM:i:37 NM:i
:0  SM:i:37 MQ:i:60 OQ:Z:HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHGHHHHHHHHHHHHHHHFHHHGHHHHHIIHHDHHHHHEHHHHH  UQ:i:0
20GAVAAXX100126:8:62:5578:2527  595 20  224759  60  101M    =   224453  -406    ACCCAAAGCTAATCAAGGCTCCCACTCTAACTCCCAAGCTCTAGGATATACCAAGGACAAAGGAAGATCATGAAATACCACCATGGGG
ATTCAATCAGCAA   834:/,1(:8::8::<98;-(-;>5?08/:;/+7<;=>?@:9>;==<=:<8<>?4>B>AABAAB@@;;<<=>===9>9?=9>=?==;=:;<?>><?@3@;1   MD:Z:7T93   PG:Z:BWA    RG:Z:20GAV.8    AM:i:25
NM:i:1  SM:i:37 MQ:i:60 OQ:Z:C4541/1.5555555544008??9?1514401555?AAA;5554444555?A?7AFEFFFFFFDF5555544445444555444@5@==5555555555  UQ:i:7
20FUKAAXX100202:4:47:20584:49257    163 20  224761  60  101M    =   225058  387 CCAAATCTAATCAAGGCTCCCACTCTAACTCCCAAGCTCTAGGATATACCAAGGACAAAGGAAGATCATGAAATACCACCATGGGGAT
TCAATCAGCAAAT   ?ACDBBCEDFEDEFEEEFEDBECFBFEFCFDEEEFEDFDFEEEFEEEECEEFEEFCEFFEEFFEFEDEAEFFFAECEFCDFEEFBFFDBEEC:@6A?C4>B   MD:Z:101    PG:Z:BWA    RG:Z:20FUK.4    AM:i:37
NM:i:0  SM:i:37 MQ:i:60 OQ:Z:HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHEHHHHDHHHHHIHHHHFHGIGHFE;D9BBD7AH  UQ:i:0
20GAVAAXX100126:7:47:4730:37293 163 20  224761  60  101M    =   225073  412 CCAAATCTAATCAAGGCTCCCACTCTAACTCCCAAGCTCTAGGATATACCAAGGACAAAGGAAGATCATGAAATACCACCATGGGGATTCAA
TCAGCAAAT   ?BB@BCBFDDECC=E@@DB;BDCFDE<<AEB@B>BADD>?C?EDEB>@AC=<?=DAE?E=CAC?;<C=@ADD?ACACCAC>:>4=B676<17@@<:AA<;6   MD:Z:101    PG:Z:BWA    RG:Z:20GAV.7    AM:i:37 NM:i
:0  SM:i:37 MQ:i:60 OQ:Z:BBA>AB@BB@BA?>B==??7>@BBA@:6@@@@@@A@BAA>A?B@BA?=?>9=????@?@>>>@?67@<;??@>?@????@9:96=>2236-39=73@:652  UQ:i:0
20GAVAAXX100126:5:46:21151:39489    83  20  224761  60  101M    =   224465  -396    CCAAATCTAATCAAGGCTCCCACTCTAACTCCCAAGCTCTAGGATATACCAAGGACAAAGGAAGATCATGAAATACCACCATGG
GGATTCAATCAGCAAAT   >9<=BBB>BB>EFFEEEFEEECEFEEFDEFEEEFFEEFEEFDDEEEEDEEFFDDDDFFFDDFFDEFDEEDFFEEEEEEEEEFEEEEEFFEFEFEF=DED=A   MD:Z:101    PG:Z:BWA    RG:Z:20GAV.5    AM:i
:37 NM:i:0  SM:i:37 MQ:i:60 OQ:Z:DBGGFDFCFFBHHHHHHHHHHGHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHEHHHGH  UQ:i:0
```

Source: Sequence Alignment / Map Format Specification

SAM/BAM: NA12878.sam



Header

@HD The header line. The first line if present.
@SQ Reference Sequence Dictionary
@RG Read Group
@Program

Source: Sequence Alignment / Map Format Specification

databricks

SAM/BAM: NA12878.sam

```
@HD VN:1.0  GO:none SO:coordinate
@SQ SN:20   LN:63025520 AS:GRCh37   UR:http://www.broadinstitute.org/ftp/pub/seq/references/Homo_sapiens_assembly19.fasta   M5:0dec9660ec1efaaf33281c0d5ea2560f SP:Homo
Sapiens
@RG ID:20FUK.1  PL:illumina PU:20FUKAAXX100202.1    LB:Solexa-18483 DT:2010-02-02T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20FUK.2  PL:illumina PU:20FUKAAXX100202.2    LB:Solexa-18484 DT:2010-02-02T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20FUK.3  PL:illumina PU:20FUKAAXX100202.3    LB:Solexa-18483 DT:2010-02-02T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20FUK.4  PL:illumina PU:20FUKAAXX100202.4    LB:Solexa-18484 DT:2010-02-02T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20FUK.5  PL:illumina PU:20FUKAAXX100202.5    LB:Solexa-18483 DT:2010-02-02T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20FUK.6  PL:illumina PU:20FUKAAXX100202.6    LB:Solexa-18484 DT:2010-02-02T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20FUK.7  PL:illumina PU:20FUKAAXX100202.7    LB:Solexa-18483 DT:2010-02-02T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20FUK.8  PL:illumina PU:20FUKAAXX100202.8    LB:Solexa-18484 DT:2010-02-02T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20GAV.1  PL:illumina PU:20GAVAAXX100126.1    LB:Solexa-18483 DT:2010-01-26T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20GAV.2  PL:illumina PU:20GAVAAXX100126.2    LB:Solexa-18484 DT:2010-01-26T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20GAV.3  PL:illumina PU:20GAVAAXX100126.3    LB:Solexa-18483 DT:2010-01-26T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20GAV.4  PL:illumina PU:20GAVAAXX100126.4    LB:Solexa-18484 DT:2010-01-26T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20GAV.5  PL:illumina PU:20GAVAAXX100126.5    LB:Solexa-18483 DT:2010-01-26T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20GAV.6  PL:illumina PU:20GAVAAXX100126.6    LB:Solexa-18484 DT:2010-01-26T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20GAV.7  PL:illumina PU:20GAVAAXX100126.7    LB:Solexa-18483 DT:2010-01-26T00:00:00-0500 SM:NA12878  CN:BI
@RG ID:20GAV.8  PL:illumina PU:20GAVAAXX100126.8    LB:Solexa-18484 DT:2010-01-26T00:00:00-0500 SM:NA12878  CN:BI
@PG ID:BWA  VN:0.5.7    CL:tk
20FUKAAXX100202:3:6:15018:84106 163 20  224759  60  101M    =   225025  366 ACCCAAATCTAATCAAGGCTCCCACTCTAACTCCCAAGCTCTAGGATATACCAAGGACAAAGGAAGATCATGAAATACCACCATGGGGATTC
AATCAGCAA   ?@BBBCEEDFEFEEEFDEEFEEEEBFEDEFCFDDEEFEDFDFEEEFEEEECEEFEEFCEFDEEFFEFEDEEEFFFDECEDCEFEEDDFFBFEFGEAEDCCC   MD:Z:101    PG:Z:BWA    RG:Z:20FUK.3    AM:i:37 NM:i
:0  SM:i:37 MQ:i:60 OQ:Z:HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHGHHHHHHHHHHHHHHHFHHHGHHHHHIIHHDHHHHHEHHHHH  UQ:i:0
20GAVAAXX100126:8:62:5578:2527  595 20  224759  60  101M    =   224453  -406    ACCCAAAGCTAATCAAGGCTCCCACTCTAACTCCCAAGCTCTAGGATATACCAAGGACAAAGGAAGATCATGAAATACCACCATGGGG
ATTCAATCAGCAA   834:/,1(:8::8::<98;-(-;>5?08/:;/+7<;=>?@:9>;==<=:<8<>?4>B>AABAAB@@;;<<=>===9>9?=9>=?==;=:;<?>><?@3@;1   MD:Z:7T93   PG:Z:BWA    RG:Z:20GAV.8    AM:i:25
NM:i:1  SM:i:37 MQ:i:60 OQ:Z:C4541/1.55555555544008??9?1514401555?AAA;5554444555?A?7AFEFFFFFFDF5555544454445555444@5@==5555555555  UQ:i:7
20FUKAAXX100202:4:47:20584:49257    163 20  224761  60  101M    =   225058  387 CCAAATCTAATCAAGGCTCCCACTCTAACTCCCAAGCTCTAGGATATACCAAGGACAAAGGAAGATCATGAAATACCACCATGGGGAT
TCAATCAGCAAAT   ?ACDBBCEDFEDEFEEEFEDBECFBFEFCFDEEEFEDFDFEEEFEEEECEEFEEFCEFFEEFFEFEDEAEFFFAECEFCDFEEFBFFDBEEC:@6A?C4>B   MD:Z:101    PG:Z:BWA    RG:Z:20FUK.4    AM:i:37
NM:i:0  SM:i:37 MQ:i:60 OQ:Z:HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHEHHHHDHHHHHIHHHHFHGIGHFE;D9BBD7AH  UQ:i:0
20GAVAAXX100126:7:47:4730:37293 163 20  224761  60  101M    =   225073  412 CCAAATCTAATCAAGGCTCCCACTCTAACTCCCAAGCTCTAGGATATACCAAGGACAAAGGAAGATCATGAAATACCACCATGGGGATTCAA
TCAGCAAAT   ?BB@BCBFDDECC=E@@DB;BDCFDE<<AEB@B>BADD>?C?EDEB>@AC=<?=DAE?E=CAC?;<C=@ADD?ACACCAC>:>4=B676<17@@<:AA<;6   MD:Z:101    PG:Z:BWA    RG:Z:20GAV.7    AM:i:37 NM:i
:0  SM:i:37 MQ:i:60 OQ:Z:BBA>AB@BB@BA?>B==??7>@BBA@:6@@@@@@A@BAA>A?B@BA?=?>9=????@?@>>>@?67@<;??@>?@????@9:96=>2236-39=73@:652  UQ:i:0
20GAVAAXX100126:5:46:21151:39489    83  20  224761  60  101M    =   224465  -396    CCAAATCTAATCAAGGCTCCCACTCTAACTCCCAAGCTCTAGGATATACCAAGGACAAAGGAAGATCATGAAATACCACCATGG
GGATTCAATCAGCAAAT   >9<=BBB>BB>EFFEEEFEEECEFEEFDEFEEEFFEEFEEFDDEEEEDEEFFDDDDFFFDDFFDEFDEEDFFEEEEEEEEEFEEEEEFFEFEFEF=DED=A   MD:Z:101    PG:Z:BWA    RG:Z:20GAV.5    AM:i
:37 NM:i:0  SM:i:37 MQ:i:60 OQ:Z:DBGGFDFCFFBHHHHHHHHHHGHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHEHHHGH  UQ:i:0
```

Alignment Records

Contains 11 mandatory fields and numerous optional fields

Source: Sequence Alignment / Map Format Specification

databricks™

SAM/BAM: NA12878.sam

```
20FUKAAXX100202:3:6:15018:84106    163  20    224759    60    101M
      =     225025    366
      ACCCAAATCTAATCAAGGCTCCCACTCTAACTCCCAAGCTCTAGGATATAC
CAAGGACAAAGGAAGATCATGAAATACCACCATGGGGATTCAATCAGCAA
      ?@BBBCEEDFEFEEEFDEEFEEEEBFEDEFCFDDEEFEDFDFEEEFEEEECEE
FEEFCEFDEEFFEFEDEEEFFFDECEDCEFEEDDFFBFEFGEAEDCCC
      MD:Z:101       PG:Z:BWA       RG:Z:20FUK.3  AM:i:37    NM:i:0
      SM:i:37    MQ:i:60
      OQ:Z:HHHHHHHHHHHHHHHHHHHHHHHHHH
HHHHHHHHHHHHHHHHHHHHHGHHHHHHHHHH
HHDHHHHHEHHHHH      UQ:i:0
```

Alignment Records
Tab-delimited format where the RED is the sequence information.

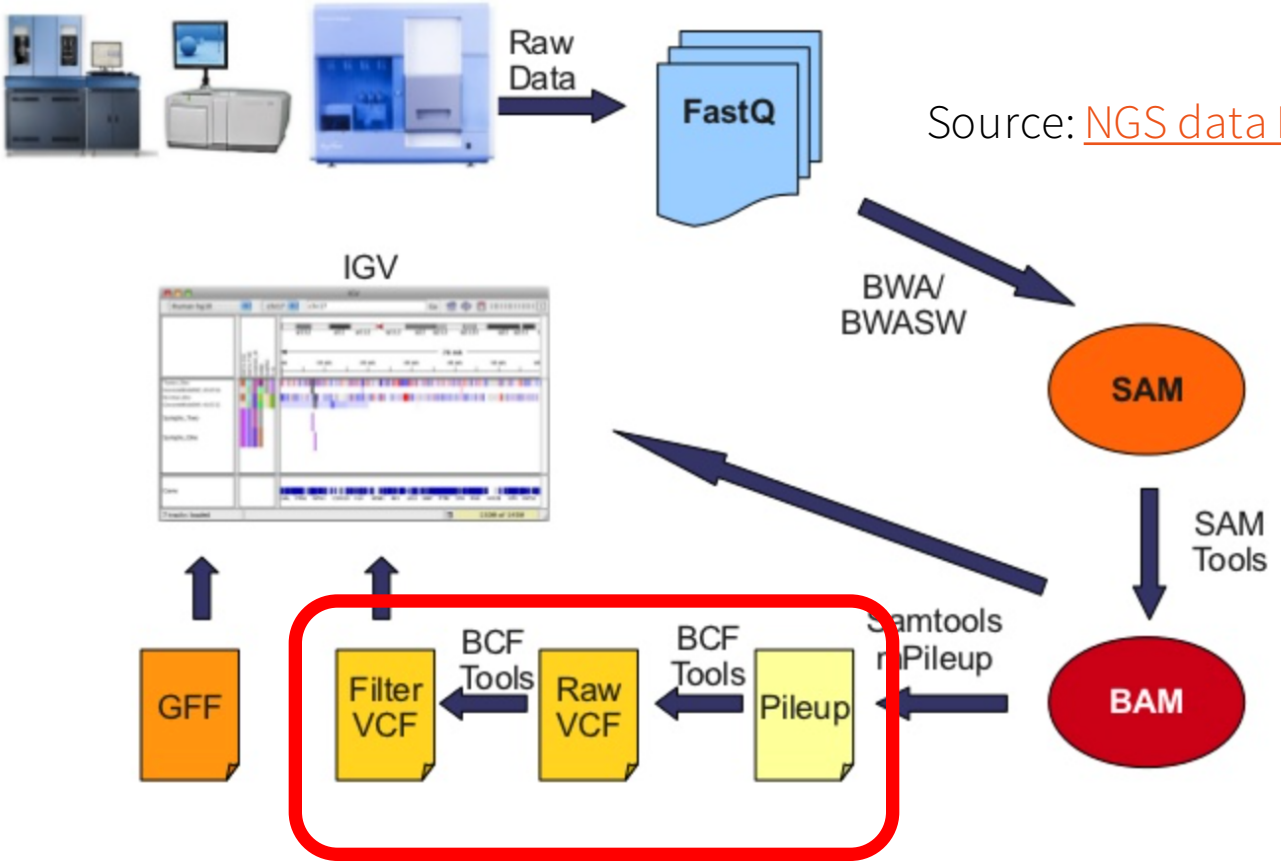
Source: Sequence Alignment / Map Format Specification

Definitions

- FastA are text files containing multiple DNA* seqs each with some text, some part of the text might be a name.
- FastQ files are like fasta, but they also have quality scores for each base of each seq, making them appropriate for reads from an Illumina machine (or other brands)
- SAM holds an alignment of seqs w/qual scores against a template.
- BAM is a compressed binary format for SAM, however it can also be unaligned in which case it's more like a compressed version of fastq.

Source: Bioinformatics: What is the difference between fasta, fastq, and sam files?

Sequence to Variation Workflow



Source: NGS data Preprocessing

Variant Call Format (VCF)

- Take a SAM/BAM file and compare it to the reference sequence
- Any differences to the reference are then *attributed* to individual variation between the sample and the reference sample
- These differences are called *variants* (or mutations) and are stored in a Variant Call Format (VCF) file.

Differences in VCF

Note, this is an example diff. We would use GATK4 / ADAM / SAMtools to do VCF diffs

```
--- small.vcf   2016-04-10 14:14:12.000000000 -0700
+++ small_missing.vcf   2016-04-10 14:14:12.000000000 -0700
@@ -47,7 +47,7 @@
 ##INFO=<ID=VQSLOD,Number=1,Type=Float,Description="Log odds ratio of being a true variant versus being false unde
 ##INFO=<ID=culprit,Number=1,Type=String,Description="The annotation which was the worst performing in the Gaussia
 #CHROM POS ID  REF ALT QUAL    FILTER  INFO    FORMAT  NA12878 NA12891 NA12892
-1  14397   .   CTGT    C   139.12  IndelQD AC=2;AF=0.333;AN=6;BaseQRankSum=1.800;ClippingRankSum=0.138;DP=69;FS=7
+1  14397   .   CTGT    C   139.12  IndelQD AC=2;AF=0.333;AN=6;BaseQRankSum=1.800;ClippingRankSum=0.138;DP=69;FS=7
 1  14522   .   G   A   195.16  VQSRTrancheSNP99.95to100.00 AC=2;AF=0.333;AN=6;BaseQRankSum=2.044;ClippingRankSum=
 1  19190   .   GC  G   1186.88 PASS    AC=3;AF=0.5[illegible]=6;BaseQRankSum=4.157;ClippingRankSum=3.666;DP=74;FS=37.03
 1  63735   rs201888535 CCTA    C   2994.09 PASS    [illegible];AF=0.167;AN=6;BaseQRankSum=1.138;ClippingRankSum=0.448;DB
```

GQ:PL **0/1:16,4:20:rd:99:120,0,827** 0/1:8,2:10:dp;rd:60:60,0,414

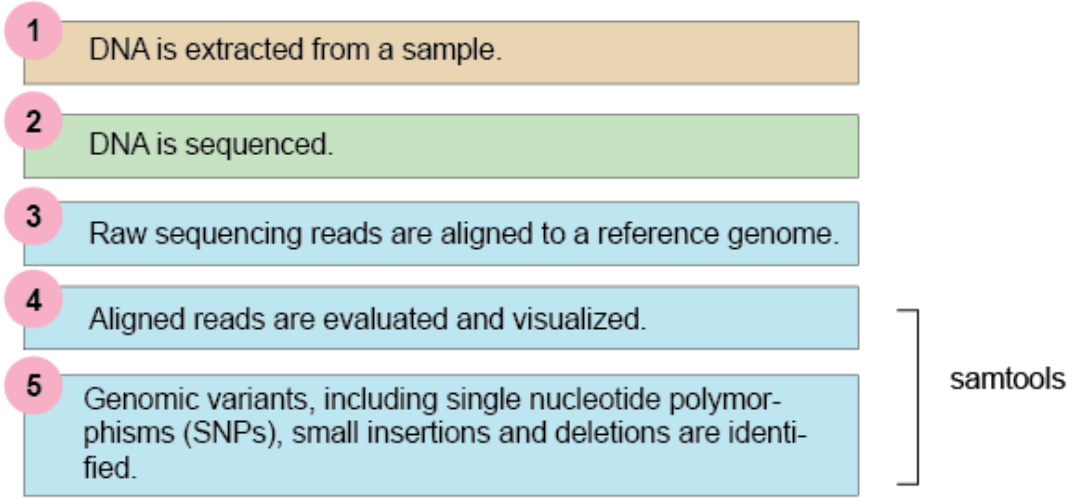
GQ:PL **./.** 0/1:8,2:10:dp;rd:60:60,0,414

databricks

SAM to VCF

- Convert .sam to .vcf for analysis
- Use samtools:

```
samtools mpileup -R -d 1000
-l rangehg18.chr -uf hg18.fa
test.bam | bcftools view -g
- > test.vcf
```



Source: SAMtools: Primer / Tutorial

VCF Format

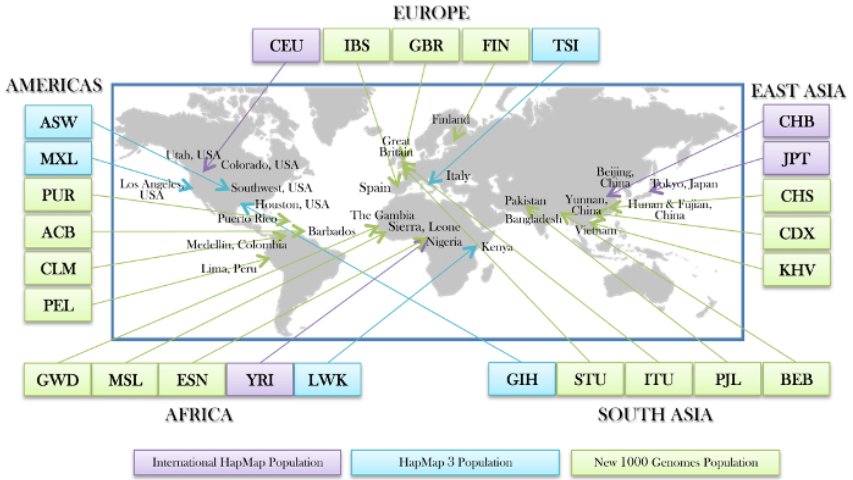
- The Variant Call Format (VCF) specifies the format of a text file used in bioinformatics for storing gene sequence variations
- Good reference: Introduction to Variant Calling

```
##fileformat=VCFv4.1 ##FILTER=<ID=PASS,Description="All filters passed"> ##fileDate=20140730
##reference=ftp://ftp.1000genomes.ebi.ac.uk//vol1/ftp/technical/reference/phase2_reference_assembly_sequence
/hs37d5.fa.gz ##source=1000GenomesPhase3Pipeline ##contig=<ID=1,assembly=b37,length=249250621>
##contig=<ID=2,assembly=b37,length=243199373> ##contig=<ID=3,assembly=b37,length=198022430>
##contig=<ID=4,assembly=b37,length=191154276> ##contig=<ID=5,assembly=b37,length=180915260>
##contig=<ID=6,assembly=b37,length=171115067> ##contig=<ID=7,assembly=b37,length=159138663>
##contig=<ID=8,assembly=b37,length=146364022> ##contig=<ID=9,assembly=b37,length=141213431>
##contig=<ID=10,assembly=b37,length=135534747> ##contig=<ID=11,assembly=b37,length=135006516>
##contig=<ID=12,assembly=b37,length=133851895> ##contig=<ID=13,assembly=b37,length=115169878>
##contig=<ID=14,assembly=b37,length=107349540> ##contig=<ID=15,assembly=b37,length=102531392>
##contig=<ID=16,assembly=b37,length=90354753> ##contig=<ID=17,assembly=b37,length=81195210>
```

Background

1000 Genomes Project

1000 Genomes Project



- Dataset: 1000 Genomes Project
- Largest public catalogue of human variation and genotype data
- Data file formats:
  - Alignment files: BAM and CRAM (Sequence Alignment/Map format)
  - Variant Call Format: stores variant calls and individual genotypes

VCF (Variant Call Format) version 4.0

- VCF 4.0 / 4.1 is the format specification for the VCFs for 1000 genomes project
- Can use ADAM (convert to ADAM Parquet) or GATK4 to make sense of data

```
##fileformat=VCFv4.0
##fileDate=20090805
##source=myImputationProgramV3.1
##reference=1000GenomesPilot-NCBI36
##phasing=partial
##INFO=<ID=NS,Number=1,Type=Integer,Description="Number of Samples With Data">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=AF,Number=.,Type=Float,Description="Allele Frequency">
##INFO=<ID=AA,Number=1,Type=String,Description="Ancestral Allele">
##INFO=<ID=DB,Number=0,Type=Flag,Description="dbSNP membership, build 129">
##INFO=<ID=H2,Number=0,Type=Flag,Description="HapMap2 membership">
##FILTER=<ID=q10,Description="Quality below 10">
##FILTER=<ID=s50,Description="Less than 50% of samples have data">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Read Depth">
##FORMAT=<ID=HQ,Number=2,Type=Integer,Description="Haplotype Quality">
#CHROM POS     ID        REF ALT    QUAL FILTER INFO                              FORMAT      N
A00001        NA00002        NA00003
20     14370   rs6054257 G       A       29   PASS   NS=3;DP=14;AF=0.5;DB;H2           GT:GQ:DP:
HQ 0|0:48:1:51,51 1|0:48:8:51,51 1/1:43:5:.,.
20     17330   .         T       A       3    q10    NS=3;DP=11;AF=0.017               GT:GQ:DP:
HQ 0|0:49:3:58,50 0|1:3:5:65,3   0/0:41:3
20     1110696 rs6040355 A       G,T     67   PASS   NS=2;DP=10;AF=0.333,0.667;AA=T;DB GT:GQ:DP:
HQ 1|2:21:6:23,27 2|1:2:0:18,2   2/2:35:4
20     1230237 .         T       .       47   PASS   NS=3;DP=13;AA=T                   GT:GQ:DP:
HQ 0|0:54:7:56,60 0|0:48:4:51,51 0/0:61:2
20     1234567 microsat1 GTCT    G,GTACT 50   PASS   NS=3;DP=9;AA=G                    GT:GQ:DP
0/1:35:4       0/2:17:2       1/1:40:3
```

databricks

Source: VCF Version 4.0

HERE!

Need to analyze ADAM Parquet conversion from VCF / BAM

ADAM

- Big Data Genomics: ADAM
- ADAM: Data Alignment/Map
  - Manipulate genomic data on a computing cluster via:
    - Application programming interface (API)
    - Command line interface (CLI)
  - ADAM operates on data stored inside of Parquet with the bdg-formats schemas, using Apache Spark, and provides scalable performance on clusters larger than 100 machines.
- ADAM: Genomics Formats and Processing Patterns for Cloud Scale Computing
- GATK Version 4.0 can read / write ADAM Parquet formatted data

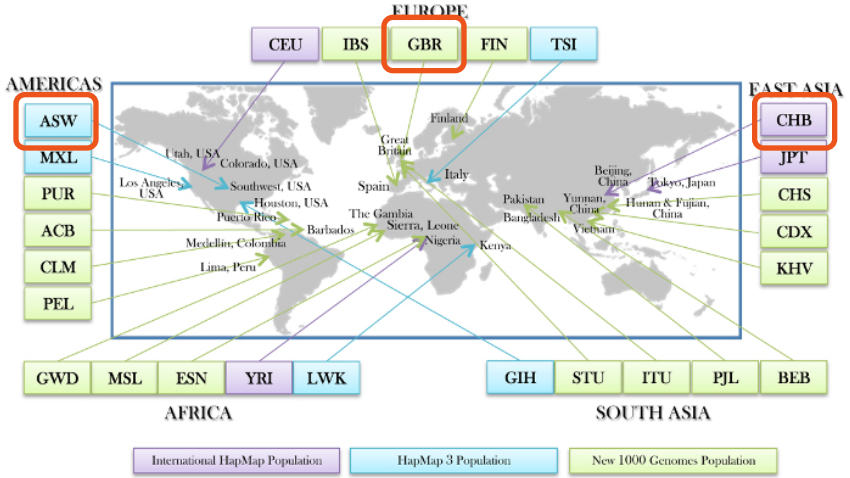
References

In addition, leveraged the following resources

- Lightening Fast Genomics with Spark and ADAM
- Andy Petrella's Spark-Notebook
- Genomics BAM datasets from AMPLab

Purpose: Predicting Population Sample

- Use the genomic sequences to predict where they came from (i.e. GBR, ASW, CHB, etc.)
- Use ADAM to make sense of the VCF / BAM / CRAM data
- Use Spark to run distributed queries
- Use Spark MLlib K-Means to build the model for prediction



Preparation: VCF to ADAM Parquet

VCF -> ADAM Parquet

- Text to compressed columnstore

```
// Load Sample VCF file
val gts:RDD[Genotype]  = sc.loadGenotypes(.)

// Save Sample VCF file into ADAM format
gts.adamParquetSave(.)
```

Load ADAM Parquet

- Load Parquet into memory

```
// Load ADAM Parquet Files
val gts:RDD[Genotype]  =
    sc.loadParquetGenotypes(l_tmp)
gts.cache()
```

Preparation: Load Panel Data

- All 1000 genomes project variant call releases come with a panel file.
- List individuals (i.e. sample) and the population they come from.
- Combined with VCF files to filter population

sample	pop	super_pop	gender
HG00096	GBR	EUR	male
HG00097	GBR	EUR	female
HG00099			female
HG00100			female

sample	pop
HG00096	GBR
HG00097	GBR
HG00099	GBR
HG00100	GBR

```
val filteredpanel = panel.select("sample",
"pop").where("pop IN ('GBR', 'ASW', 'CHB')")
```

Preparation: Filter and obtain complete genomes

```
// Filter out only gts within the panel
val finalGts = gts.filter(g => bPanel.value.contains(g.getSampleId))
...

// Create the complete Gts by filtering out the missingVariants
val completeGts = finalGts.filter {
    g => ! (missingVariants contains variantId(g).hashCode)
}
```

K-means Clustering

```
// Filter out only gts within the panel
import org.apache.spark.mllib.clustering.{KMeans,KMeansModel}

val numClusters = 3
val numIterations = 20

// features was initalized above
val clusters:KMeansModel = KMeans.train(
    features, numClusters, numIterations
)
```

Confusion Matrix

		Predicted		
		GBR	CHB	ASW
Actual	GBR	62	29	
	CHB	13	90	
	ASW	16	3	42

Spark 1.6

Unified Memory Management

SPARK-10000

Memory Management in Spark: < 1.5

- Two separate memory managers:
  - Execution memory: computation of shuffles, joins, sorts, aggregations
  - Storage memory: caching and propagating internal data sources across cluster
- Issues with this:
  - Manual intervention to avoid unnecessary spilling
  - No pre-defined defaults for all workloads
  - Need to partition the execution (shuffle) memory and cache memory fractions
- Goal: Unify these two memory regions and borrow from each other

Unified Memory Management in Spark 1.6

- Can cross between execution and storage memory
  - When execution memory exceeds its own region, it can borrow as much of the storage space as is free and vice versa
  - Borrowed storage memory can be evicted at any time (though not execution memory at this time)
- New configurations to be introduced
- Reference: [SPARK-10000]

Unified Memory Management in Spark 1.6

- Under memory pressure
  - Evict cached data
  - Evict storage memory
  - Evict execution memory
- Notes:
  - Dynamically allocated reserved storage region that execution memory cannot borrow from
  - Cached data evicted only if actual storage exceeds the dynamically allocated reserved storage region

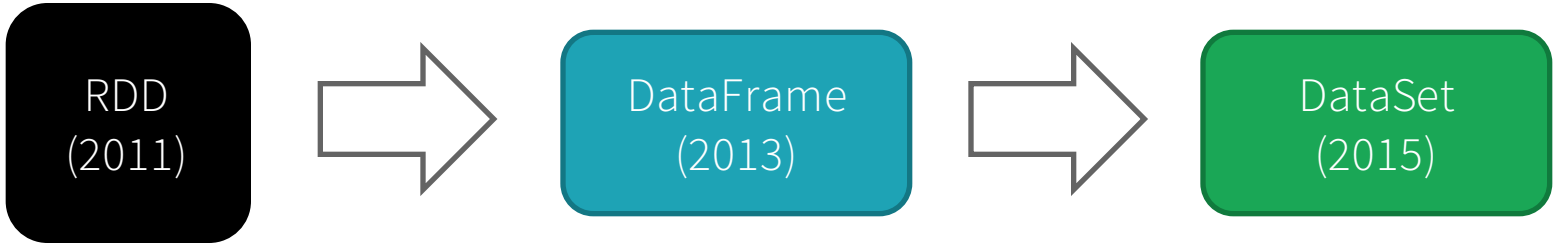
Unified Memory Management in Spark 1.6

- Allows for better memory management for multi-tenancy and applications relying heavily on caching
- No cap on storage memory nor on execution memory
- Dynamic allocation of reserved storage memory will not require user configuration

Dataset API

SPARK-9999

History of Spark APIs



Distribute collection of JVM objects

Functional Operators (map, filter, etc.)

Distribute collection of Row objects

Expression-based operations and UDFs

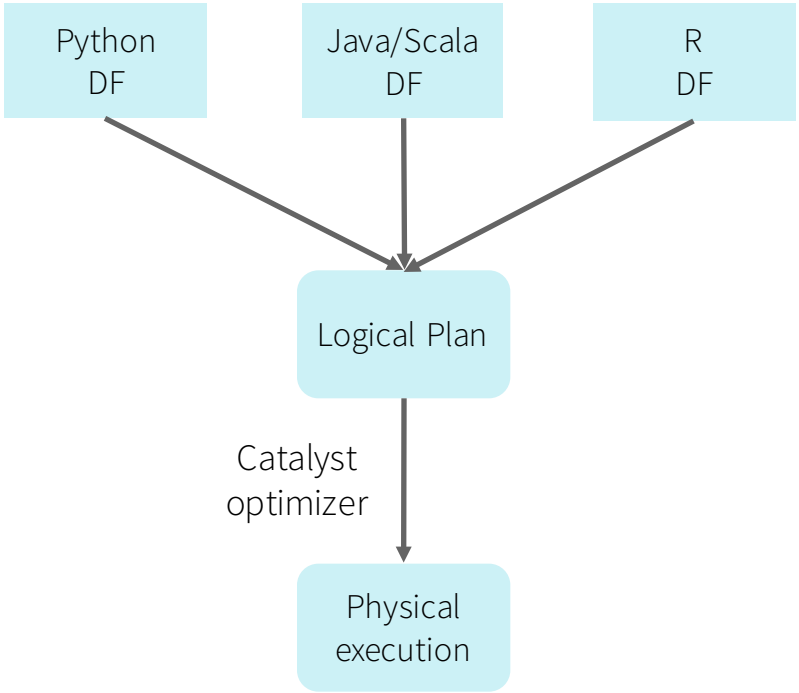
Logical plans and optimizer

Fast/efficient internal representations

Internally rows, externally JVM objects

"Best of both worlds" **type safe + fast**

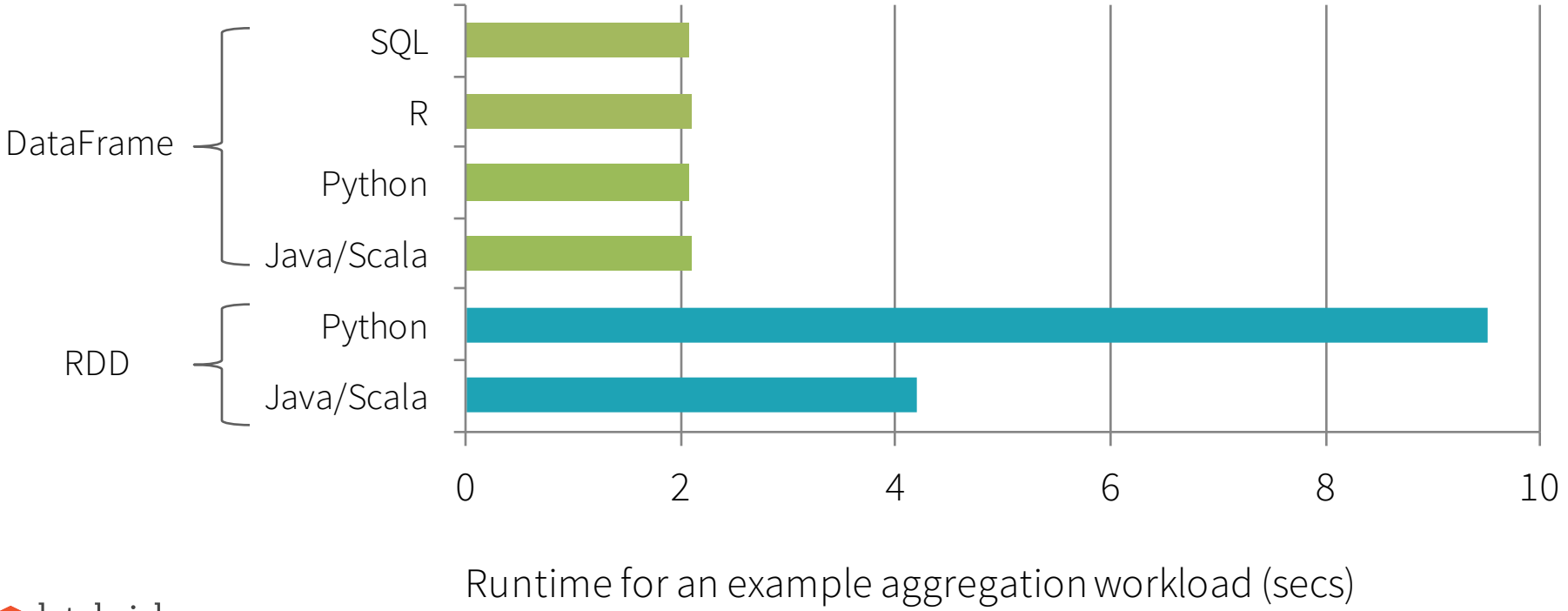
Spark DataFrame Execution



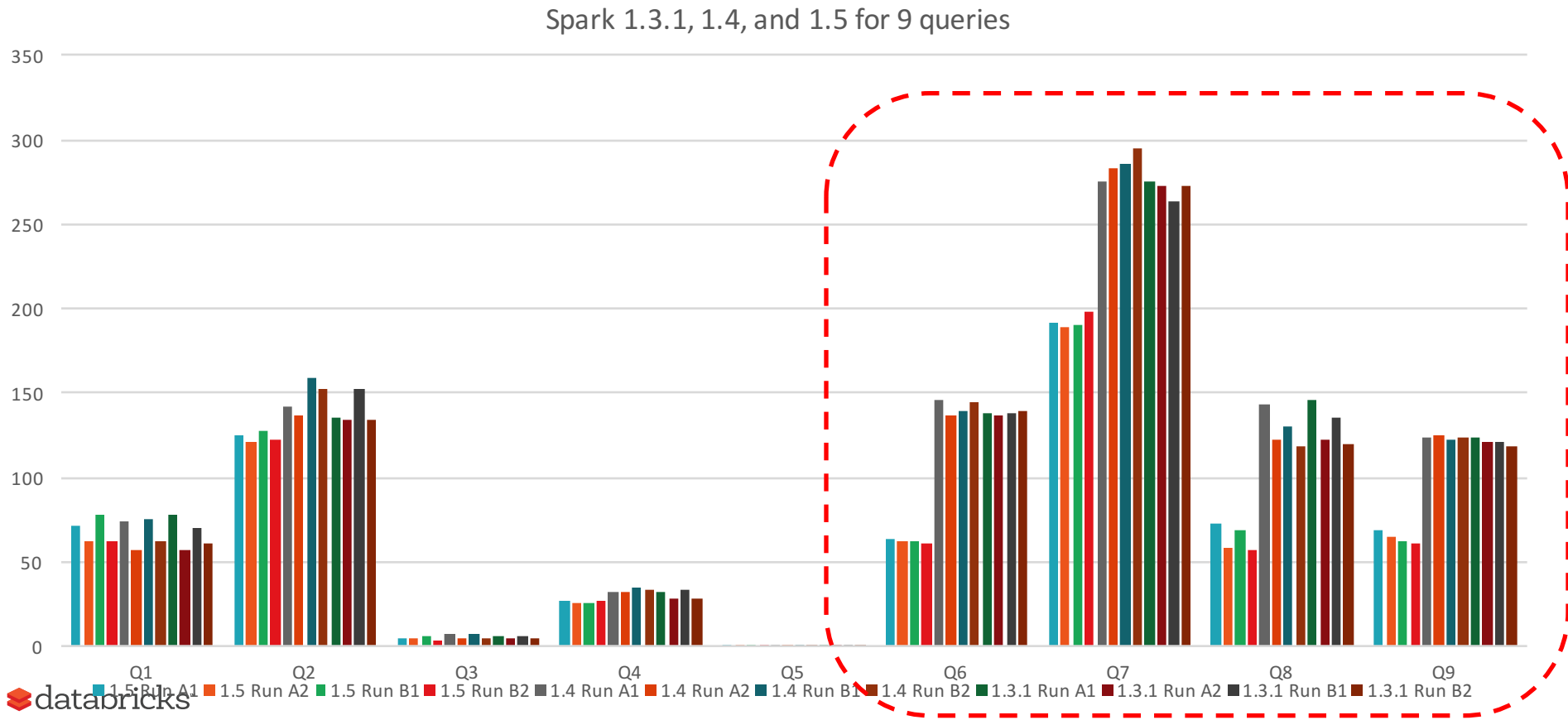
Simple wrappers to create logical plan

Intermediate representation for computation

Benefit of Logical Plan: Performance Parity Across Languages



NYC Taxi Dataset



databricks

Dataset API in Spark 1.6

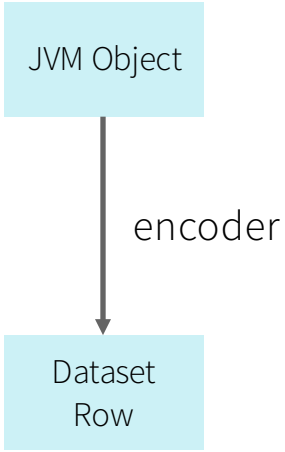
Typed interface over DataFrames / Tungsten

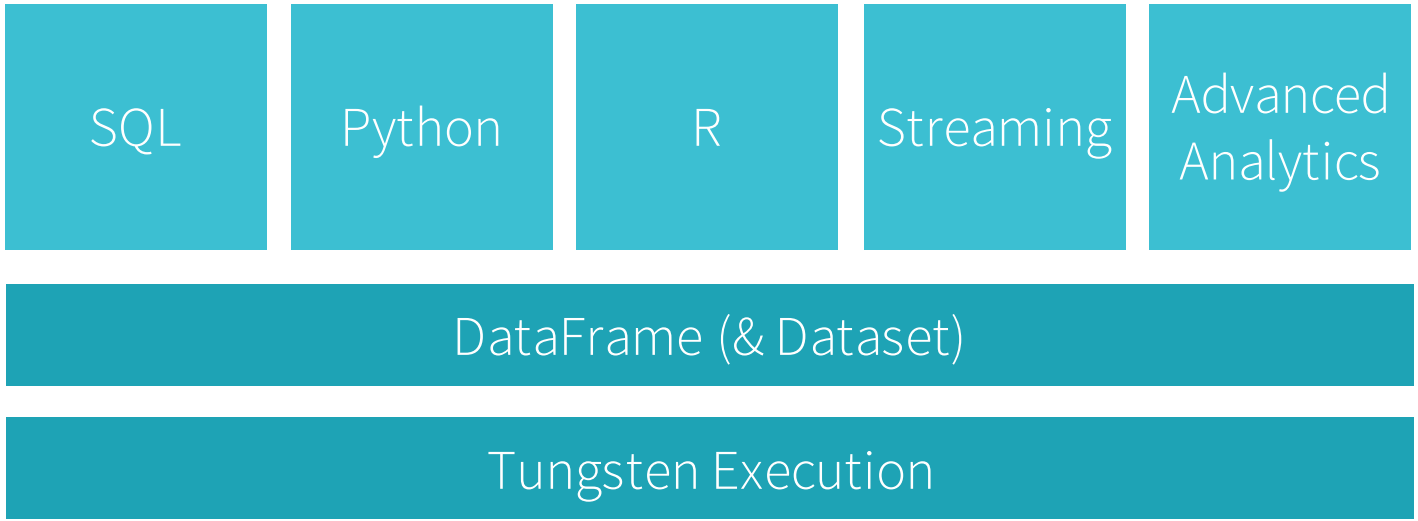
```
case class Person(name: String, age: Long)

val dataframe = read.json("people.json")
val ds: Dataset[Person] = dataframe.as[Person]

ds.filter(p => p.name.startsWith("M"))
      .toDF()
      .groupBy($"name")
      .avg("age")
```

Dataset

"Encoder" converts from JVM Object into a Dataset Row

Checkout [SPARK-9999]




SQL
Python
R
Streaming
Advanced Analytics
DataFrame (& Dataset)
Tungsten Execution

Improved State Management

SPARK-2629

[Streaming] New improved state management

- Introducing a `DStream` transformation for stateful stream processing
  - Does not scan every key
  - Easier to implement common use cases
    - timeout of idle data
    - returning items other than state
- Supercedes `updateStateByKey` in functionality and performance.
- `trackStateByKey` (note, this name may change)
- Check out [SPARK-2629]

[Streaming] trackStateByKey example
(name may change)

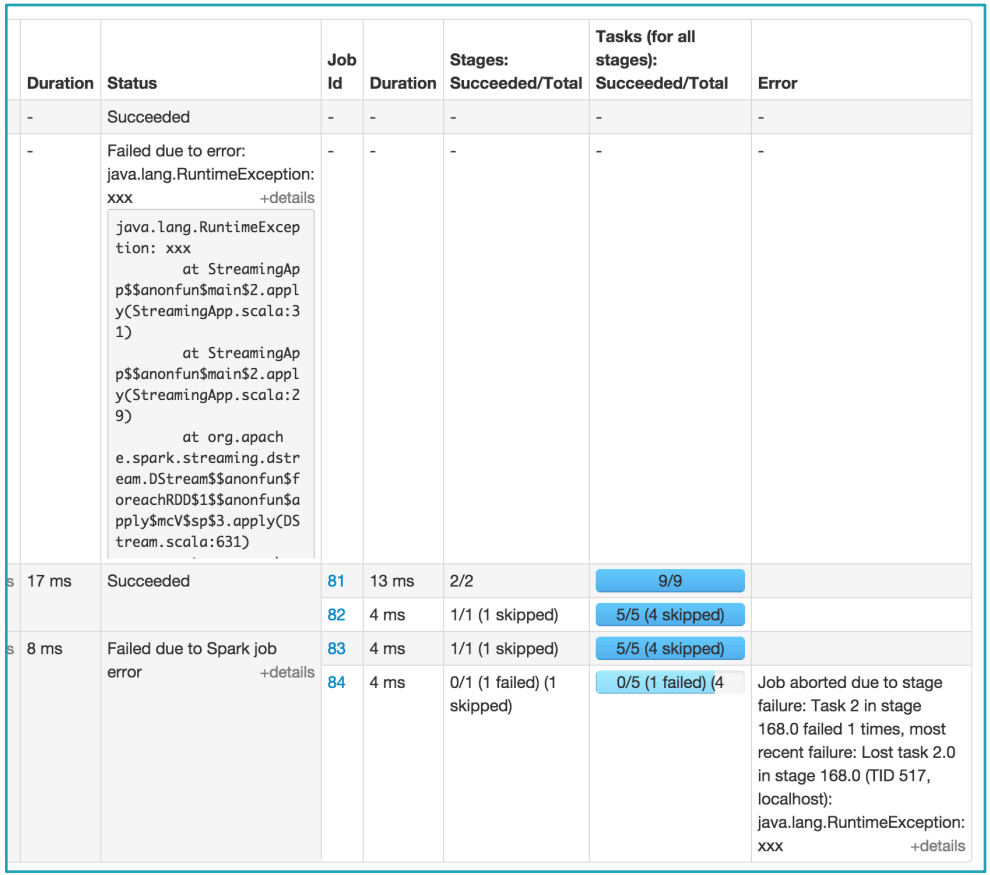
```
// Initial RDD input
val initialRDD = ssc.sparkContext.parallelize(...)

// ReceiverInputDStream
val lines = ssc.socketTextStream(...)
val words = lines.flatMap(...)
val wordDStream = words.map(x => (x, 1))

// stateDStream using trackStateByKey
val trackStateFunc = (...) { ... }
val stateDStream = wordDStream.trackStateByKey(
   StateSpec.function(trackStateFunc).initialState(initialRDD))
```

Streaming Display Failed Output

SPARK-10885 PR#8950

[Streaming] Display the failed output op in Streaming

Check out:
[SPARK-10885] PR#8950



Duration	Status	Job Id	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total	Error
-	Succeeded	-	-	-	-	-
-	Failed due to error: java.lang.RuntimeException: xxx +details java.lang.RuntimeException: xxx at StreamingApp$$anonfun$main$2.apply(StreamingApp.scala:31) at StreamingApp$$anonfun$main$2.apply(StreamingApp.scala:29) at org.apache.spark.streaming.dstream.DStream$$anonfun$foreachRDD$1$$anonfun$apply$mcVsp3.apply(DStream.scala:631)	-	-	-	-	-
17 ms	Succeeded	81	13 ms	2/2	9/9	
		82	4 ms	1/1 (1 skipped)	5/5 (4 skipped)	
8 ms	Failed due to Spark job error +details	83	4 ms	1/1 (1 skipped)	5/5 (4 skipped)	
		84	4 ms	0/1 (1 failed) (1 skipped)	0/5 (1 failed) (4	Job aborted due to stage failure: Task 2 in stage 168.0 failed 1 times, most recent failure: Lost task 2.0 in stage 168.0 (TID 517, localhost): java.lang.RuntimeException: xxx +details

databricks

Advanced Layout of Cached Data

SPARK-4849

Advanced Layout of Cached Data

- Storing partitioning and ordering schemes in In-memory table scan
  - Allows for performance improvements: e.g. in Joins, an extra partition step can be saved based on this information
- Adding `distributeBy` and `localSort` to DF API
  - Similar to HiveQL's `DISTRIBUTE BY`
  - Allows the user to control the partitioning and ordering of a data set
- Check out [SPARK-4849]

[MLlib]: Pipeline persistence

SPARK-6725

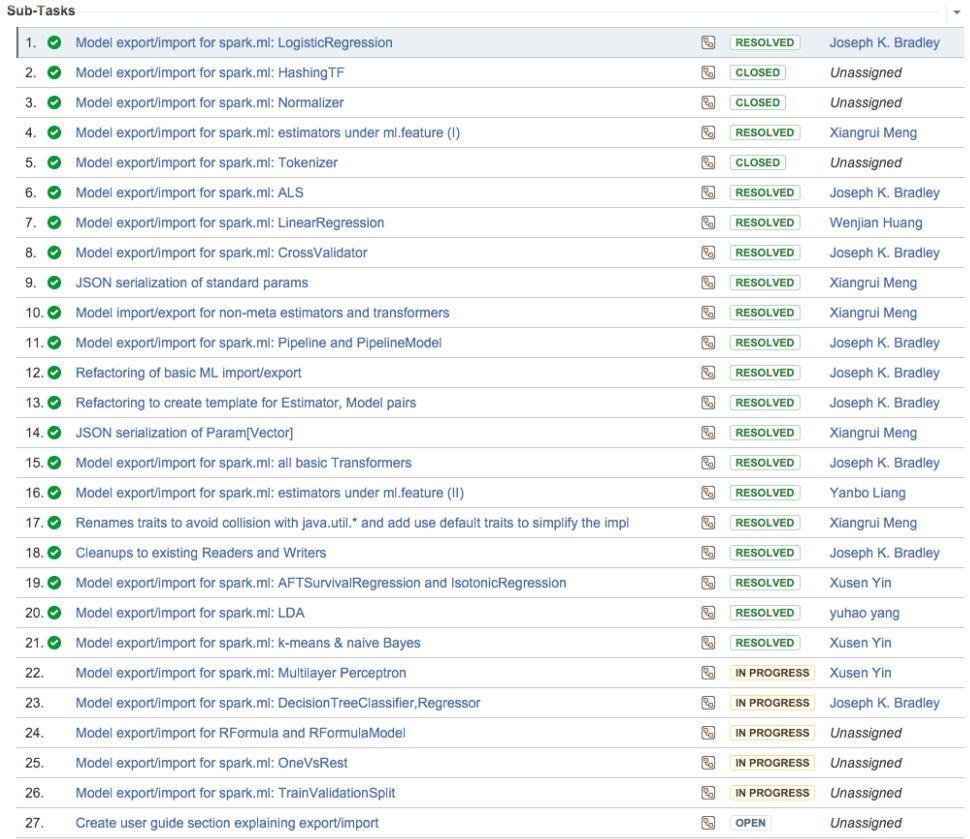
[MLlib]: Pipeline persistence

- Persist ML Pipelines to:
  - Save models in the spark.ml API
  - Re-run workflows in a reproducible manner
  - Export models to non-Spark apps (e.g., model server)
- This is more complex than ML model persistence because:
  - Must persist Transformers and Estimators, not just Models.
  - We need a standard way to persist Params.
  - Pipelines and other meta-algorithms can contain other Transformers and Estimators, including as Params.
  - We should save feature metadata with Models

[MLlib]: Pipeline persistence

Reference [SPARK-6725]

- Adding model export/import to the spark.ml API.
- Adding the internal Saveable/Loadable API and Parquet-based format



Sub-Tasks

1.	Model export/import for spark.ml: LogisticRegression	RESOLVED	Joseph K. Bradley
2.	Model export/import for spark.ml: HashingTF	CLOSED	*Unassigned*
3.	Model export/import for spark.ml: Normalizer	CLOSED	*Unassigned*
4.	Model export/import for spark.ml: estimators under ml.feature (I)	RESOLVED	Xiangrui Meng
5.	Model export/import for spark.ml: Tokenizer	CLOSED	*Unassigned*
6.	Model export/import for spark.ml: ALS	RESOLVED	Joseph K. Bradley
7.	Model export/import for spark.ml: LinearRegression	RESOLVED	Wenjian Huang
8.	Model export/import for spark.ml: CrossValidator	RESOLVED	Joseph K. Bradley
9.	JSON serialization of standard params	RESOLVED	Xiangrui Meng
10.	Model import/export for non-meta estimators and transformers	RESOLVED	Xiangrui Meng
11.	Model export/import for spark.ml: Pipeline and PipelineModel	RESOLVED	Joseph K. Bradley
12.	Refactoring of basic ML import/export	RESOLVED	Joseph K. Bradley
13.	Refactoring to create template for Estimator, Model pairs	RESOLVED	Joseph K. Bradley
14.	JSON serialization of Param[Vector]	RESOLVED	Xiangrui Meng
15.	Model export/import for spark.ml: all basic Transformers	RESOLVED	Joseph K. Bradley
16.	Model export/import for spark.ml: estimators under ml.feature (II)	RESOLVED	Yanbo Liang
17.	Renames traits to avoid collision with java.util.* and add use default traits to simplify the impl	RESOLVED	Xiangrui Meng
18.	Cleanups to existing Readers and Writers	RESOLVED	Joseph K. Bradley
19.	Model export/import for spark.ml: AFTSurvivalRegression and IsotonicRegression	RESOLVED	Xusen Yin
20.	Model export/import for spark.ml: LDA	RESOLVED	yuhao yang
21.	Model export/import for spark.ml: k-means & naive Bayes	RESOLVED	Xusen Yin
22.	Model export/import for spark.ml: Multilayer Perceptron	IN PROGRESS	Xusen Yin
23.	Model export/import for spark.ml: DecisionTreeClassifier,Regressor	IN PROGRESS	Joseph K. Bradley
24.	Model export/import for RFormula and RFormulaModel	IN PROGRESS	*Unassigned*
25.	Model export/import for spark.ml: OneVsRest	IN PROGRESS	*Unassigned*
26.	Model export/import for spark.ml: TrainValidationSplit	IN PROGRESS	*Unassigned*
27.	Create user guide section explaining export/import	OPEN	*Unassigned*

R-like statistics for GLMs

SPARK-9836

R-like statistics for GLMs

```
> # Model summary are returned in a similar format to R's native glm().
  summary(model)
```

▸ (1) Spark Jobs

```
$devianceResiduals
 Min       Max
 -1.307112 1.412532

$coefficients
                   Estimate  Std. Error t value  Pr(>|t|)
(Intercept)        2.251393  0.3697543  6.08889  9.568102e-09
Sepal_Width        0.8035609 0.106339   7.556598 4.187317e-12
Species_versicolor 1.458743  0.1121079  13.01195 0
Species_virginica  1.946817  0.100015   19.46525 0
```

Command took 0.90s

- Provide R-like summary statistics for ordinary least squares via normal equation solver
- Check out [SPARK-9836]

Spark 1.6 Improvements

By Category

Performance

SPARK-10000 **Unified Memory Management** - Shared memory for execution and caching instead of exclusive division of the regions.

SPARK-10917, SPARK-11149 **In-memory Columnar Cache Performance** - Significant (up to 14x) speed up when caching data that contains complex types in DataFrames or SQL.

SPARK-11389 **SQL Execution Using Off-Heap Memory** - Support for configuring query execution to occur using off-heap memory to avoid GC overhead

Performance (continued)

SPARK-4849 **Advanced Layout of Cached Data** - storing partitioning and ordering schemes in In-memory table scan, and adding distributeBy and localSort to DF API

SPARK-9858 **Adaptive query execution** - Initial support for automatically selecting the number of reducers for joins and aggregations.

SPARK-11787 **Parquet Performance** - Improve Parquet scan performance when using flat schemas.

Spark SQL

- SPARK-9999 Dataset API
- SPARK-11197 SQL Queries on Files
- SPARK-11745 Reading non-standard JSON files
- SPARK-10412 Per-operator Metrics for SQL Execution
- SPARK-11329 Star (*) expansion for StructTypes
- SPARK-11111 Fast null-safe joins
- SPARK-10978 Datasource API Avoid Double Filter

Spark Streaming

API Updates

- SPARK-2629 New improved state management
- SPARK-11198 Kinesis record deaggregation
- SPARK-10891 Kinesis message handler function
- SPARK-6328 Python Streamng Listener API

UI Improvements

- Made failures visible in the streaming tab, in the timelines, batch list, and batch details page.
- Made output operations visible in the streaming tab as progress bars

MLlib: New algorithms / models

- SPARK-8518 **Survival analysis** - Log-linear model for survival analysis
- SPARK-9834 **Normal equation for least squares** - Normal equation solver, providing R-like model summary statistics
- SPARK-3147 **Online hypothesis testing** - A/B testing in the Spark Streaming framework
- SPARK-9930 **New feature transformers** - ChiSqSelector, QuantileDiscretizer, SQL transformer
- SPARK-6517 **Bisecting K-Means clustering** - Fast top-down clustering variant of K-Means

MLlib: API Improvements

ML Pipelines

- SPARK-6725 **Pipeline persistence** - Save/load for ML Pipelines, with partial coverage of spark.ml algorithms
- SPARK-5565 **LDA in ML Pipelines** - API for Latent Dirichlet Allocation in ML Pipelines

R API

- SPARK-9836 **R-like statistics for GLMs** - (Partial) R-like stats for ordinary least squares via summary(model)
- SPARK-9681 **Feature interactions in R formula** - Interaction operator ":" in R formula

Python API - Many improvements to Python API to approach feature parity

MLlib: Miscellaneous Improvements

- SPARK-7685, SPARK-9642 **Instance weights for GLMs** - Logistic and Linear Regression can take instance weights
- SPARK-10384, SPARK-10385 **Univariate and bivariate statistics in DataFrames** - Variance, stddev, correlations, etc.
- SPARK-10117 **LIBSVM data source** - LIBSVM as a SQL data source

For More Information

Apache Spark 1.6.0 Release Preview: http://bit.ly/1Ilk9da

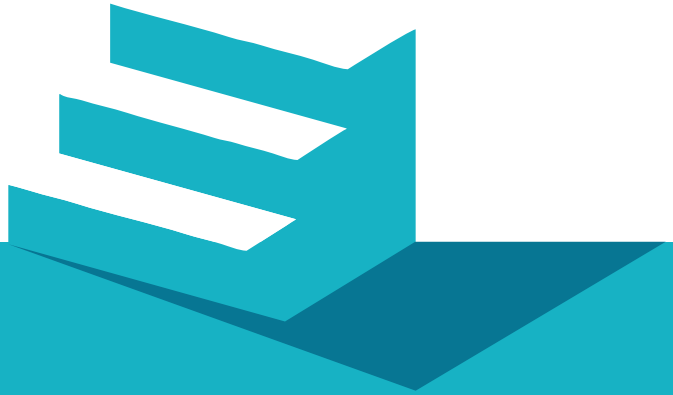
Spark 1.6 Preview available in Databricks: http://bit.ly/1QR0RPC

Spark 1.6 Improvements Notebook (Exported HTML): http://bit.ly/1lrvdLc

Spark 1.6 R Improvements Notebook (Exported HTML): http://bit.ly/1OBkjMM

Spark 2.0

Steps to bigger & better things….



Builds on all we learned in past 2 years

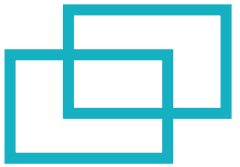
Abridged version of the following

- Reynold Xin's Apache Spark 2.0: Faster, Easier, and Smarter webinar
- Michael Armbrust's Structuring Spark: DataFrames, Datasets, and Streaming
- Tathagata Das' A Deep Dive into Spark Streaming
- Joseph Bradley's Apache Spark MLlib 2.0 Preview: Data Science and Production
- Tim Hunter's TensorFrames: Google Tensorflow on Apache Spark

Major Features in 2.0

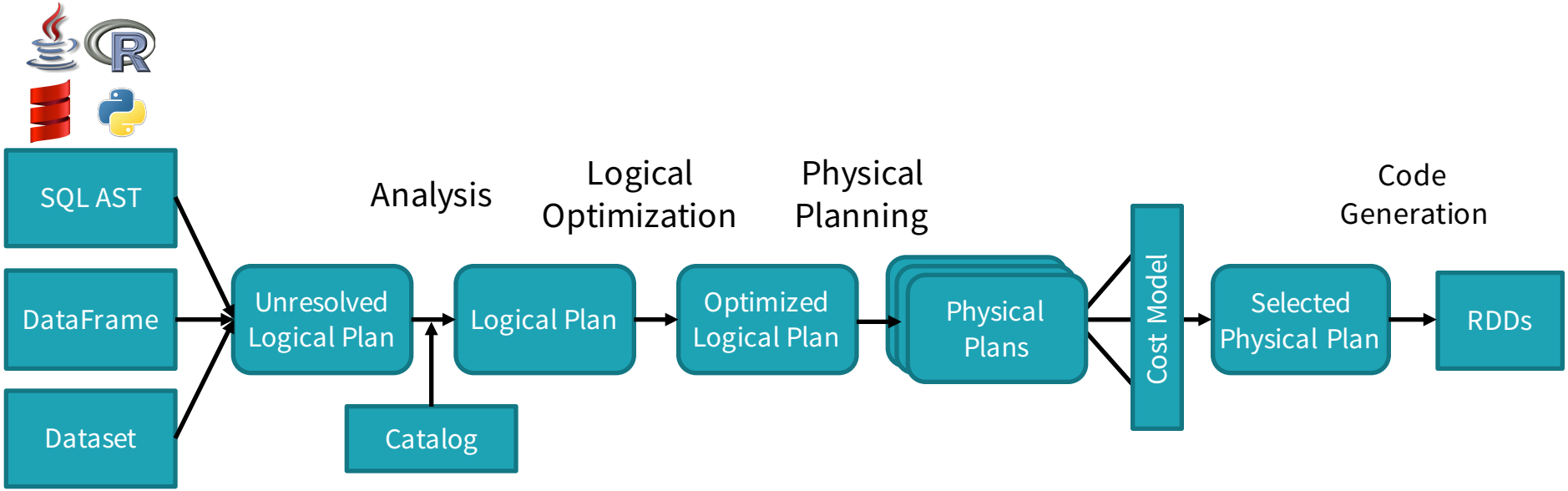
Tungsten Phase 2
speedups of 5-20x

Structured Streaming



SQL 2003
& Unifying Datasets
and DataFrames

Shared Optimization & Execution



DataFrames, Datasets and SQL
share the same optimization/execution pipeline

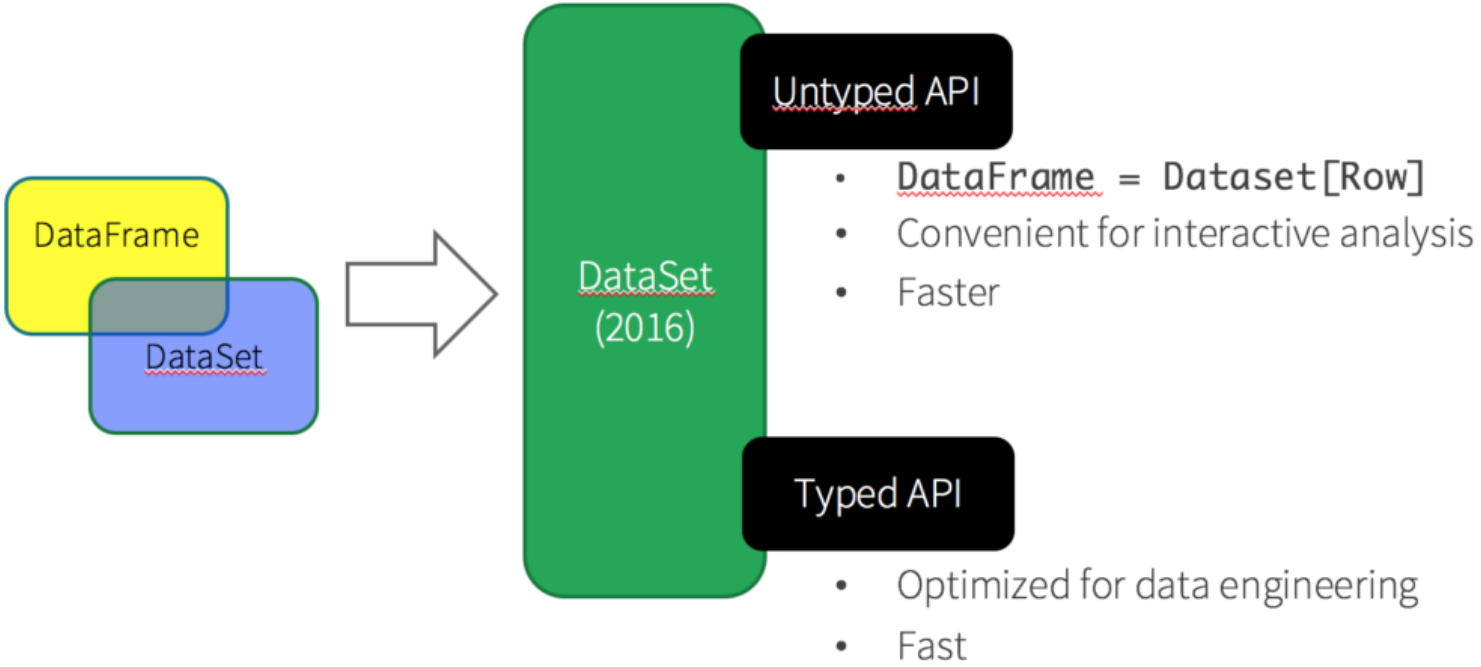
Datasets and DataFrames

In 2015, we added DataFrames & Datasets as structured data APIs

- DataFrames are collections of rows with a schema
- Datasets add static types, e.g. Dataset[Person]
- Both run on Tungsten

Spark 2.0 will merge these APIs: DataFrame = Dataset[Row]

Apache Spark 2.0 API



SparkSession – a new entry point

SparkSession is the “SparkContext” for Dataset/DataFrame

- Entry point for reading data
- Working with metadata
- Configuration
- Cluster resource management

Long-Term

RDD will remain the low-level API in Spark

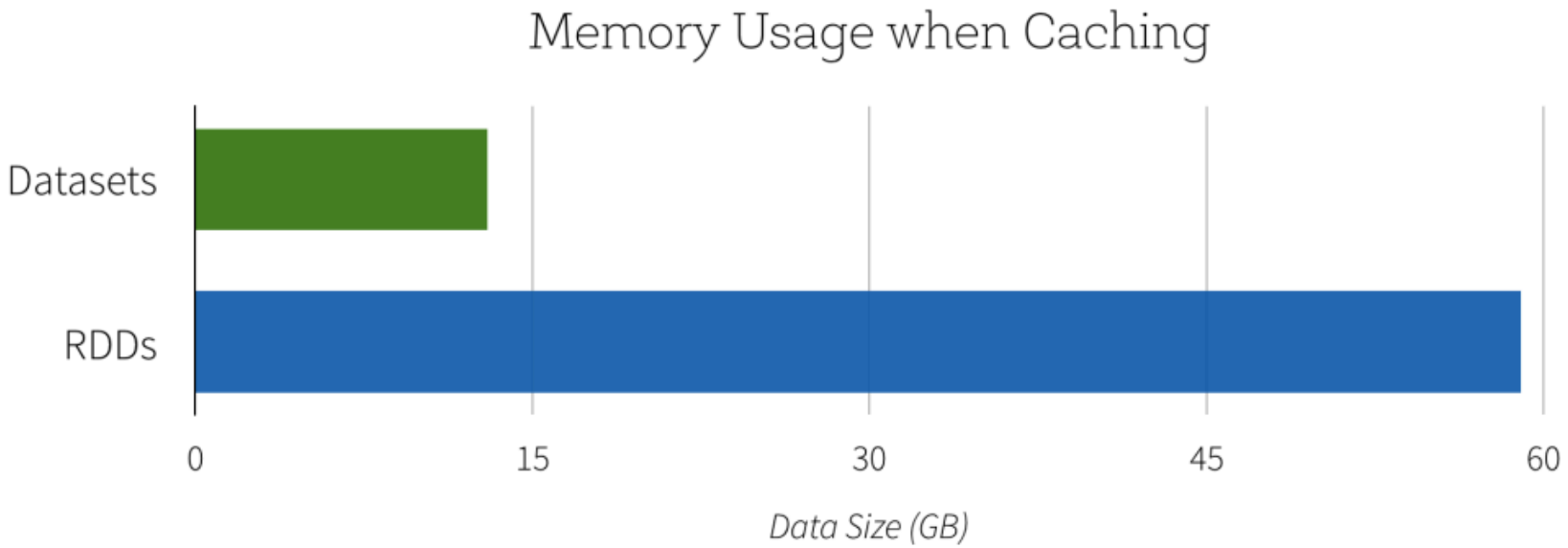
Datasets & DataFrames give richer semantics and optimizations

- New libraries will increasingly use these as interchange format
- Examples: Structured Streaming, MLlib, GraphFrames

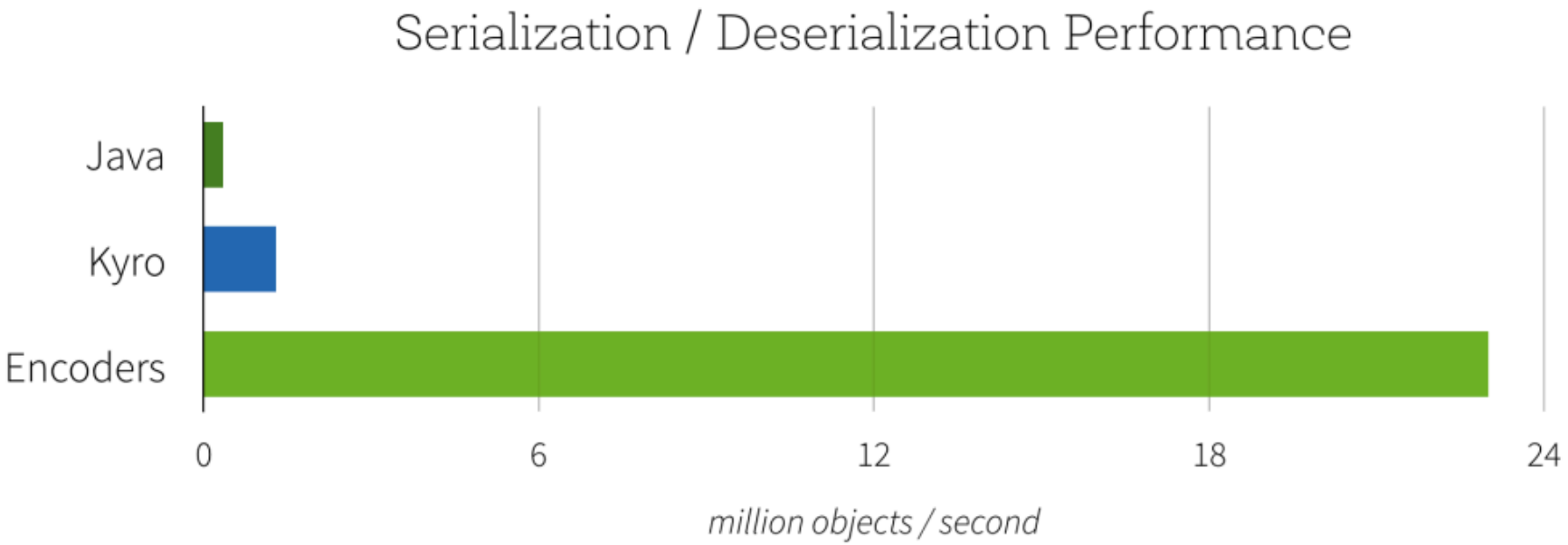
Tungsten Phase 2

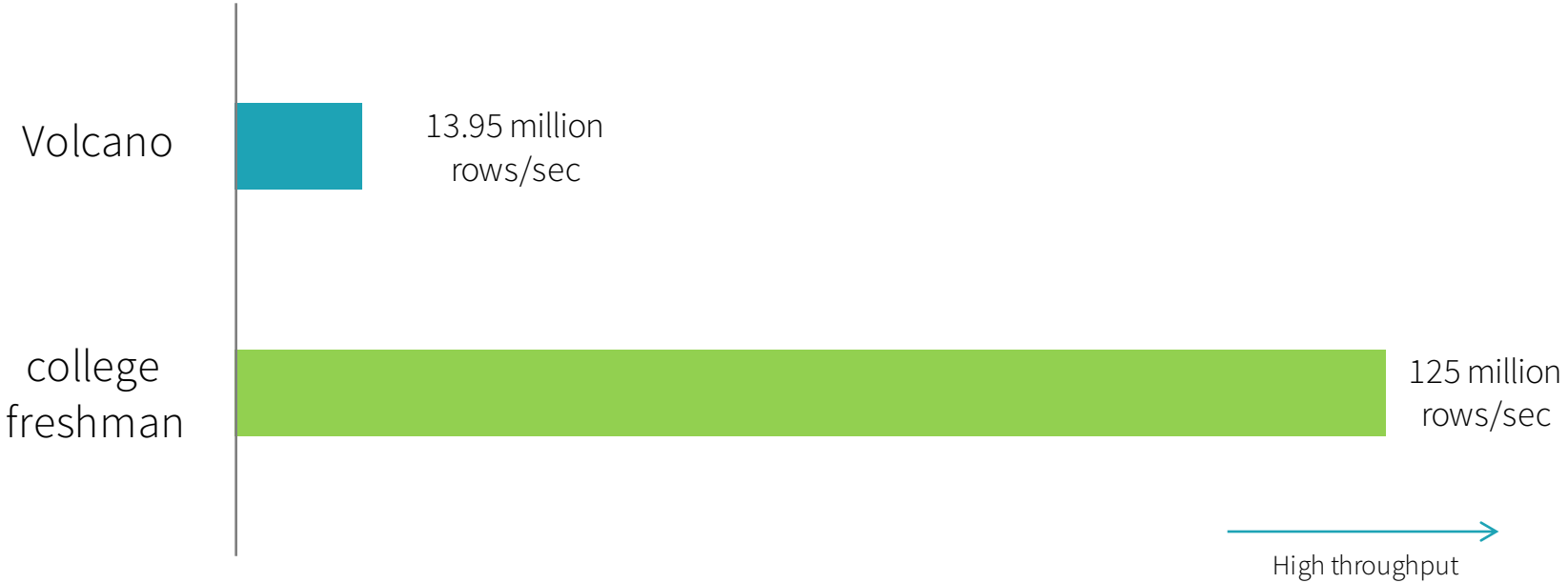
Can we speed up Spark by 10X?

Space Efficiency



Serialization performance





Note: End-to-end, single thread, single column, and data originated in Parquet on disk

How does a student beat 30 years of research?

Volcano

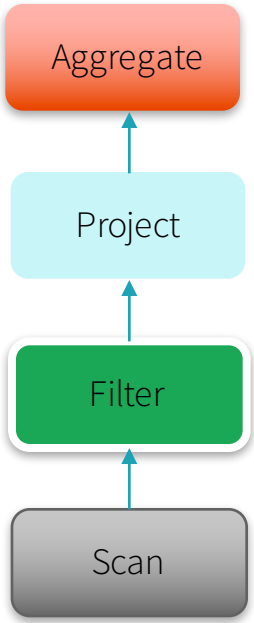
1. Many virtual function calls
2. Data in memory (or cache)
3. No loop unrolling, SIMD, pipelining

hand-written code

1. No virtual function calls
2. Data in CPU registers
3. Compiler loop unrolling, SIMD, pipelining

Take advantage of all the information that is known **after** query compilation

Tungsten Phase 2: Spark as a "Compiler"




```
long count = 0;
for (ss_item_sk in store_sales) {
  if (ss_item_sk == 1000) {
    count += 1;
  }
}
```

Functionality of a general purpose execution engine; performance as if hand built system just to run your query

Structured Streaming

How do we simplify streaming?

Integration Example

Stream

(home.html, 10:08)

(product.html, 10:09)

(home.html, 10:10)

...

Streaming engine

MySQL

Page	Minute	Visits
home	10:09	21
pricing	10:10	30
...	...	...

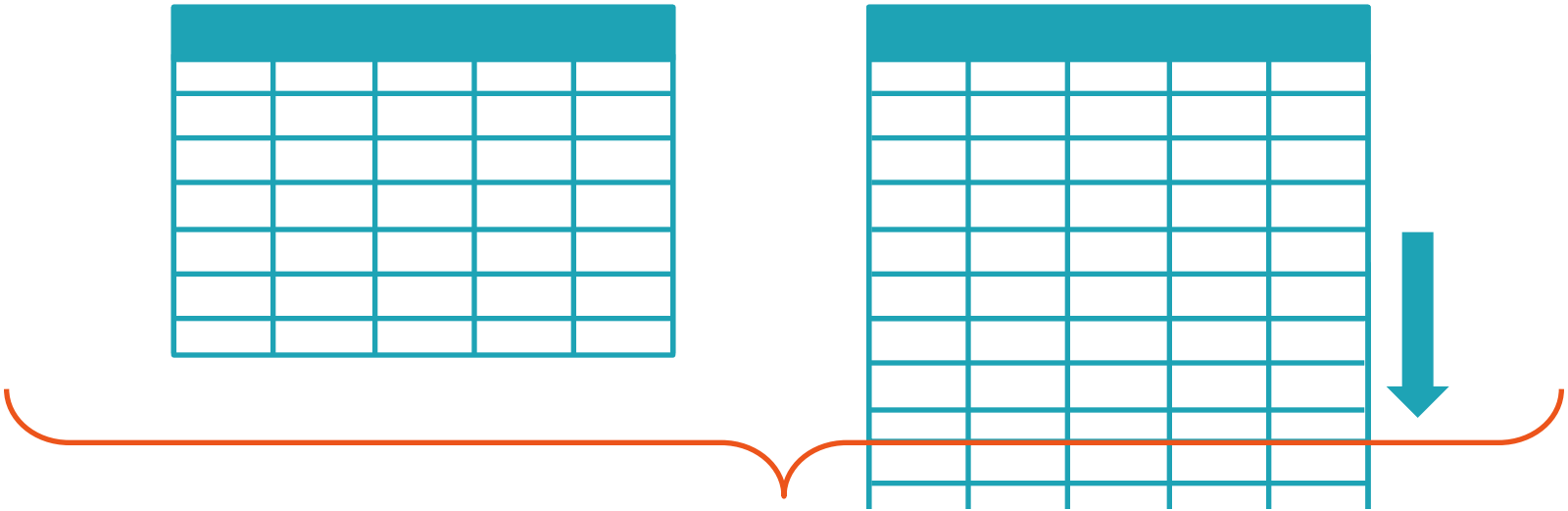
What can go wrong?

- Late events
- Partial outputs to MySQL
- State recovery on failure
- Distributed reads/writes
- ...

The simplest way to perform streaming analytics is not having to ***reason*** about streaming.

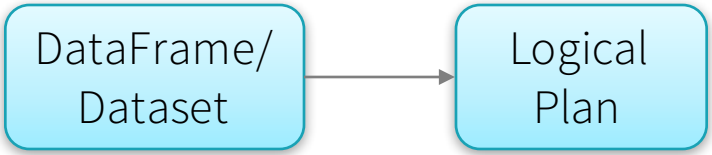
Spark 1.3
Static DataFrames

Spark 2.0
Infinite DataFrames



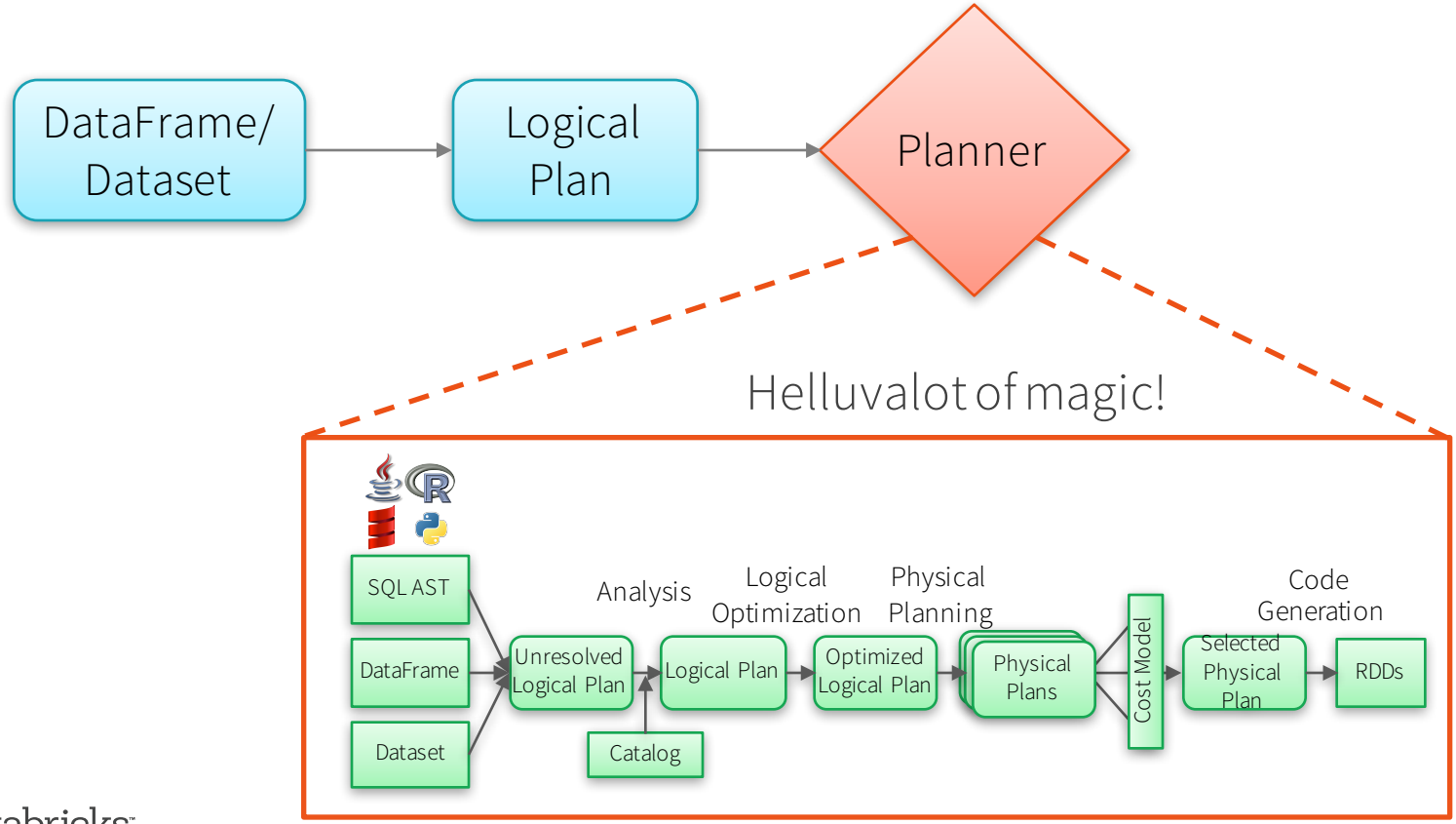
Single API !

Batch Execution on Spark SQL

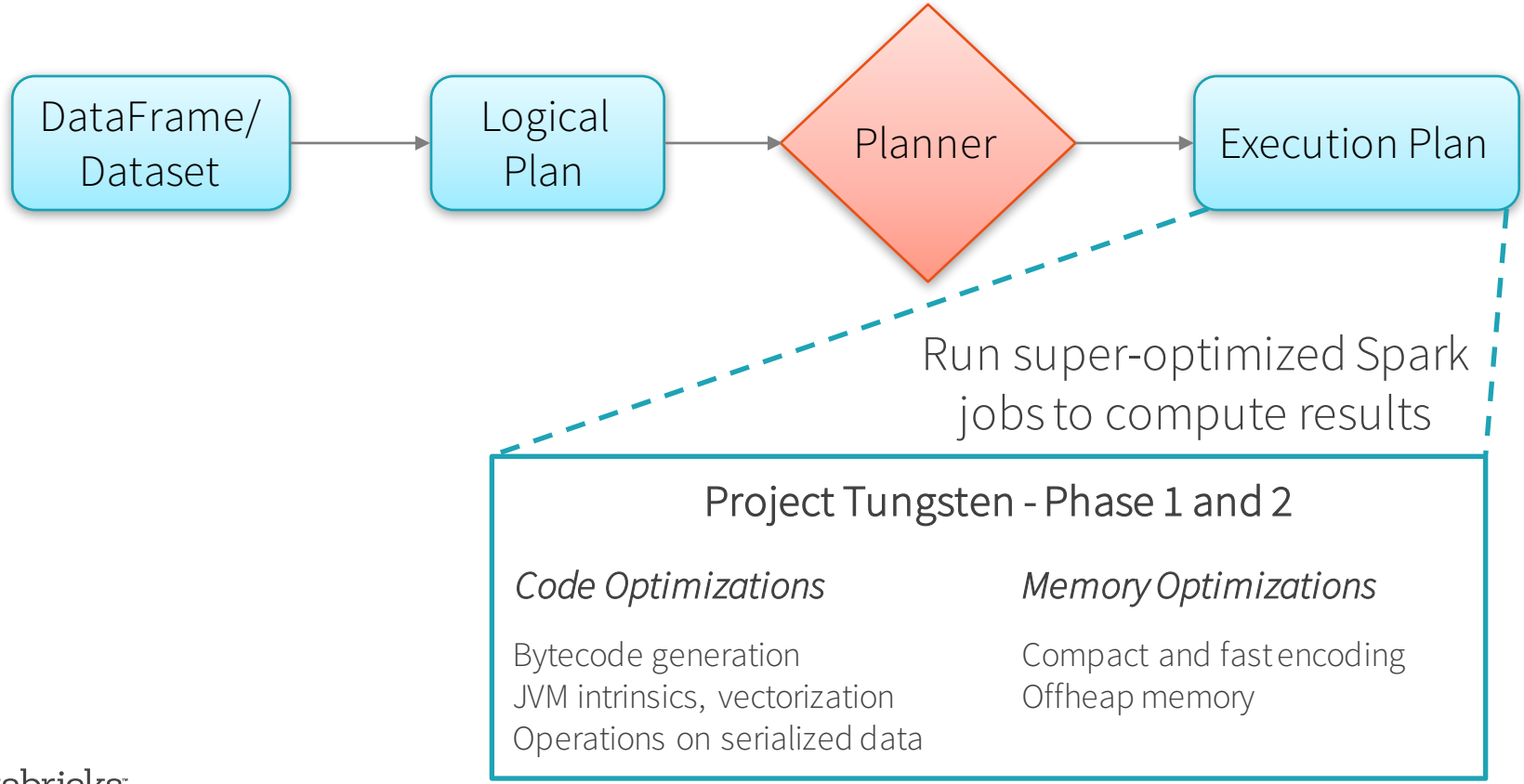


Abstract
representation
of query

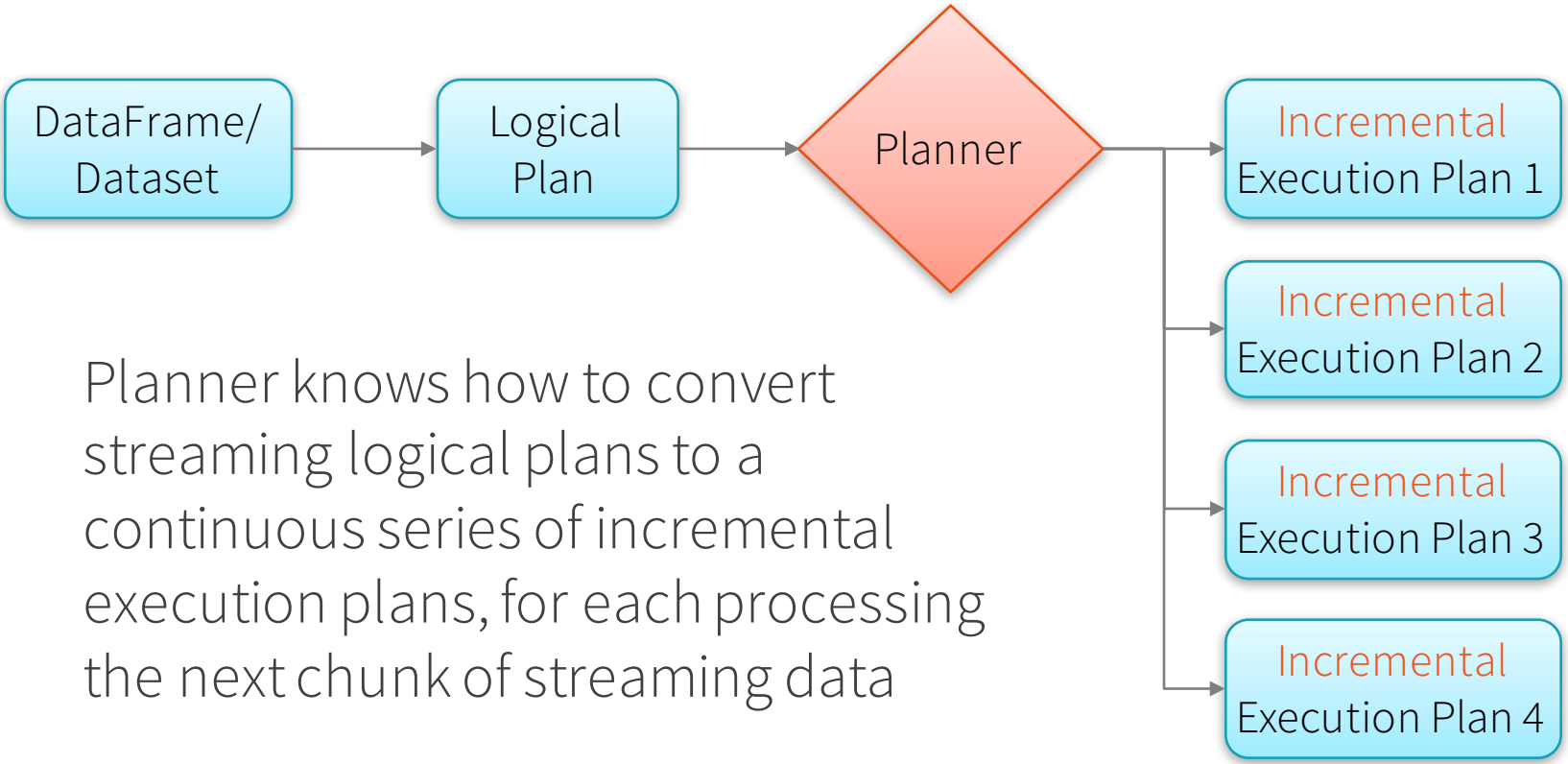
Batch Execution on Spark SQL



Batch Execution on Spark SQL



Continuous Incremental Execution



Planner knows how to convert streaming logical plans to a continuous series of incremental execution plans, for each processing the next chunk of streaming data

Example: Batch Aggregation

```
logs  = ctx.read.format("json").open("s3://logs")

logs.groupBy(logs.user_id).agg(sum(logs.time))
    .write.format("jdbc")
    .save("jdbc:mysql//...")
```

Example: Continuous Aggregation

```
logs  = ctx.read.format("json").stream("s3://logs")

logs.groupBy(logs.user_id).agg(sum(logs.time))
    .write.format("jdbc")
    .startStream("jdbc:mysql//...")
```

Structured Streaming

High-level streaming API built on Spark SQL engine

- Declarative API that extends DataFrames / Datasets
- Event time, windowing, sessions, sources & sinks

Support interactive & batch queries

- Aggregate data in a stream, then serve using JDBC
- Change queries at runtime
- Build and apply ML models

Not just streaming, but "continuous applications"

What's Coming?

Apache Spark 2.0

- Unification of the DataFrame/Dataset & *Context APIs
- Basic streaming API
- Event-time aggregations

Apache Spark 2.1+

- Other streaming sources / sinks
- Machine learning
- Watermarks

Structure in other libraries: MLlib, GraphFrames

MLlib 2.0 Preview

DataFrame-based API for MLlib

a.k.a. “Pipelines” API, with utilities for constructing ML Pipelines

In 2.0, the DataFrame-based API will become the primary API for MLlib.

Voted by community

`org.apache.spark.ml, pyspark.ml`

The RDD-based API will enter maintenance mode.

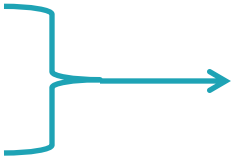
- Still maintained with bug fixes, but no new features
- `org.apache.spark.mllib, pyspark.mllib`

Goals for MLlib in 2.0

Major initiatives

Generalized Linear Models

Python & R API expansion



Exploratory analysis

ML persistence: saving & loading models & Pipelines

Production

Also in 2.0:

Sketching algorithms: http://databricks.com/blog/2016/05/19

For details, see SPARK-12626 roadmap JIRA + mailing list discussions.

GLMs in 2.0

Model family	Supported link functions
Gaussian	Identity, Log, Inverse
Binomial	Logit, Probit, CLogLog
Poisson	Log, Identity, Sqrt
Gamma	Inverse, Identity, Log

Fixes for corner cases

- E.g., handle invalid labels gracefully

`GeneralizedLinearRegression`

- Max 4096 features
- Solved using Iteratively Reweighted Least Squares (IRLS)

`LinearRegression` & `LogisticRegression`

- Millions of features
- Solved using L-BFGS / OWL-QN

Python & R APIs for MLlib

Goal: Expand ML APIs for critical languages for data science

Python

- **Clustering algorithms**: Bisecting K-Means, Gaussian Mixtures, LDA
- **Meta-algorithms**: OneVsRest, TrainValidationSplit
- **GeneralizedLinearRegression**
- **Feature transformers**: ChiSqSelector, MaxAbsScaler, QuantileDiscretizer
- **Model inspection**: summaries for Logistic Regression, Linear Regression, GLMs

R

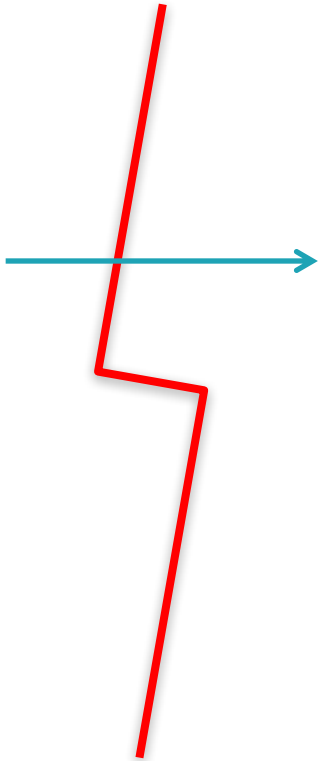
- **Regression & classification**: Generalized Linear Regression, AFT survival regression
- **Clustering**: K-Means

Why ML persistence?

Data Science

Prototype (Python/R)
Create Pipeline

- Extract raw features
- Transform features
- Select key features
- Fit multiple models
- Combine results to make prediction



Software Engineering

Re-implement Pipeline for production (Java)
Deploy Pipeline

- Extra implementation work
- Different code paths
- Synchronization overhead

With ML persistence...

Data Science

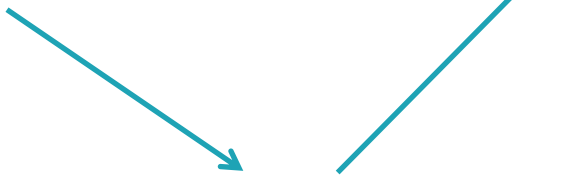
Prototype (Python/R)
Create Pipeline

Software Engineering

Load Pipeline (Scala/Java)

```
Model.load("s3n://...")
```

Deploy in production



Persist model or Pipeline:

```
model.save("s3n://...")
```

Customizing ML Pipelines

MLlib 2.0 will include:

- 29 feature transformers (Tokenizer, Word2Vec, ...)
- 21 models (for classification, regression, clustering, ...)
- Model tuning & evaluation

But some applications require customized Transformers & Models.

What's next?

Prioritized items on the 2.1 roadmap JIRA (SPARK-15581):

- Critical feature completeness for the DataFrame-based API
  - Multiclass logistic regression
  - Statistics
- Python API parity & R API expansion
- Scaling & speed tuning for key algorithms: trees & ensembles

GraphFrames

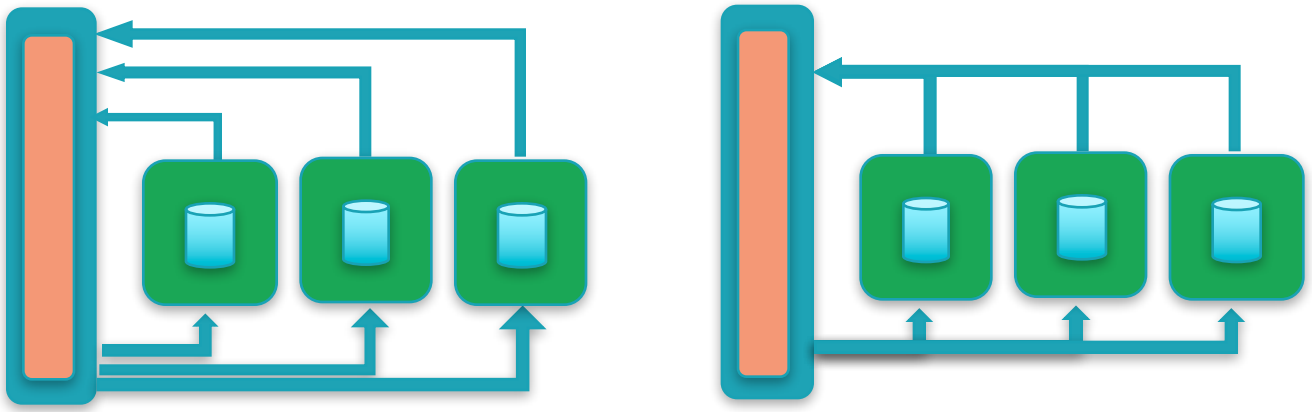
- Release for Spark 2.0
- Speed improvements (join elimination, connected components)

TensorFrames: Google Tensorflow on Apache Spark

Tim Hunter

Spark Summit 2016 - Meetup

Asynchronous vs. synchronous

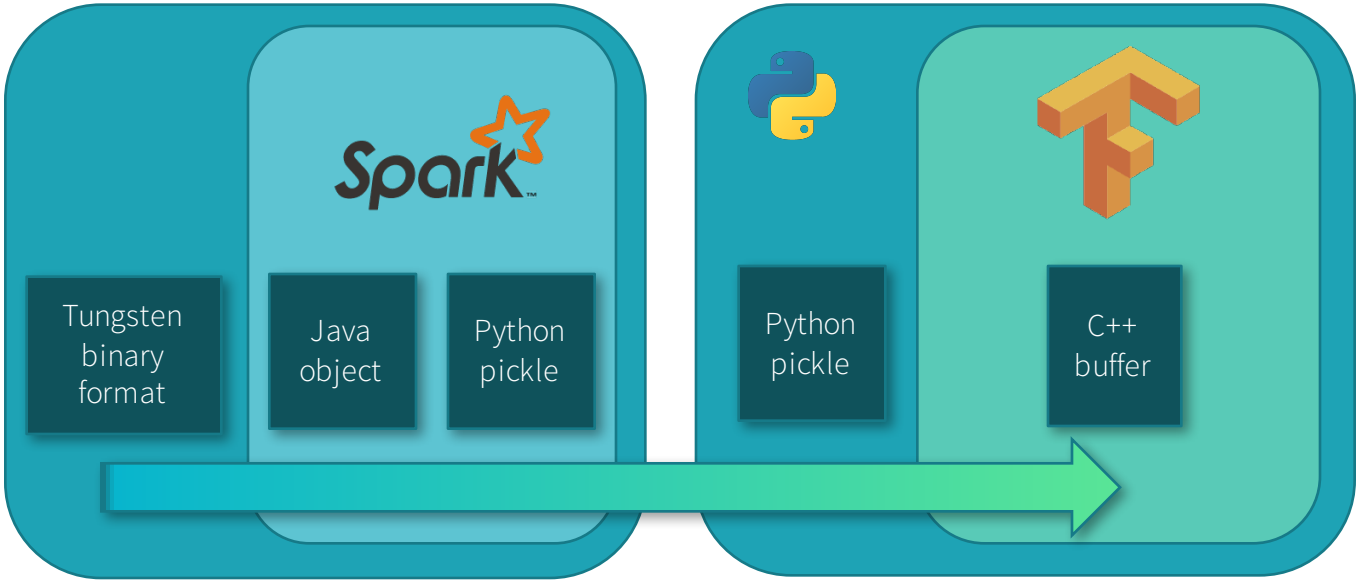


- Asynchronous algorithms perform updates concurrently
- Spark is synchronous model, deep learning frameworks usually asynchronous
- A large number of ML computations are synchronous
- Even deep learning may benefit from synchronous updates

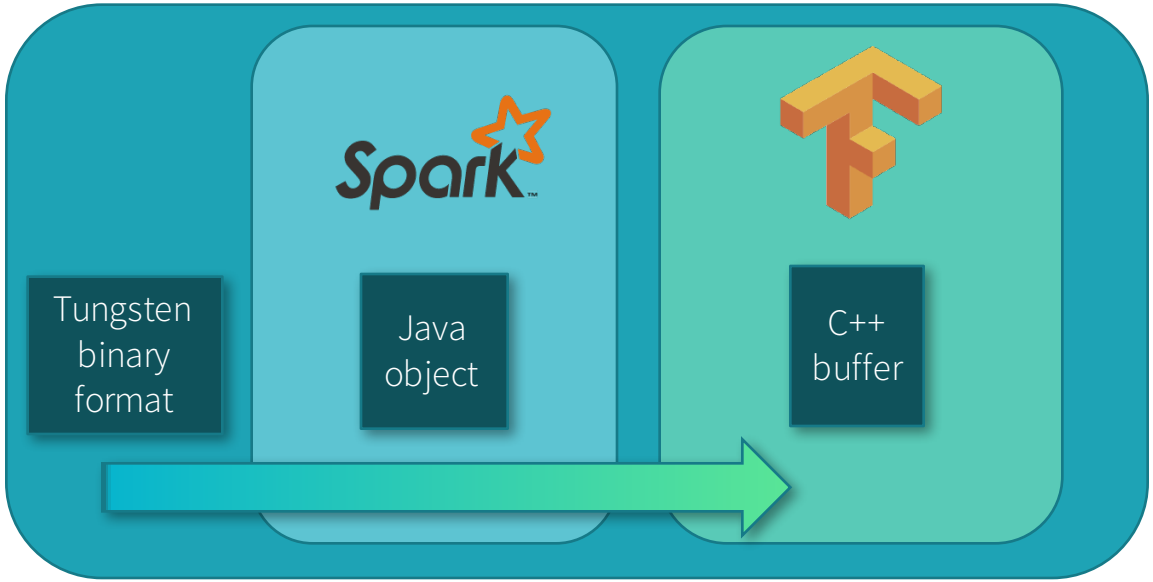
Google TensorFlow

- Library for writing “machine intelligence” algorithms
- Very popular for deep learning and neural networks
- Can also be used for general purpose numerical computations
- Interface in C++ and Python

It is a communication problem

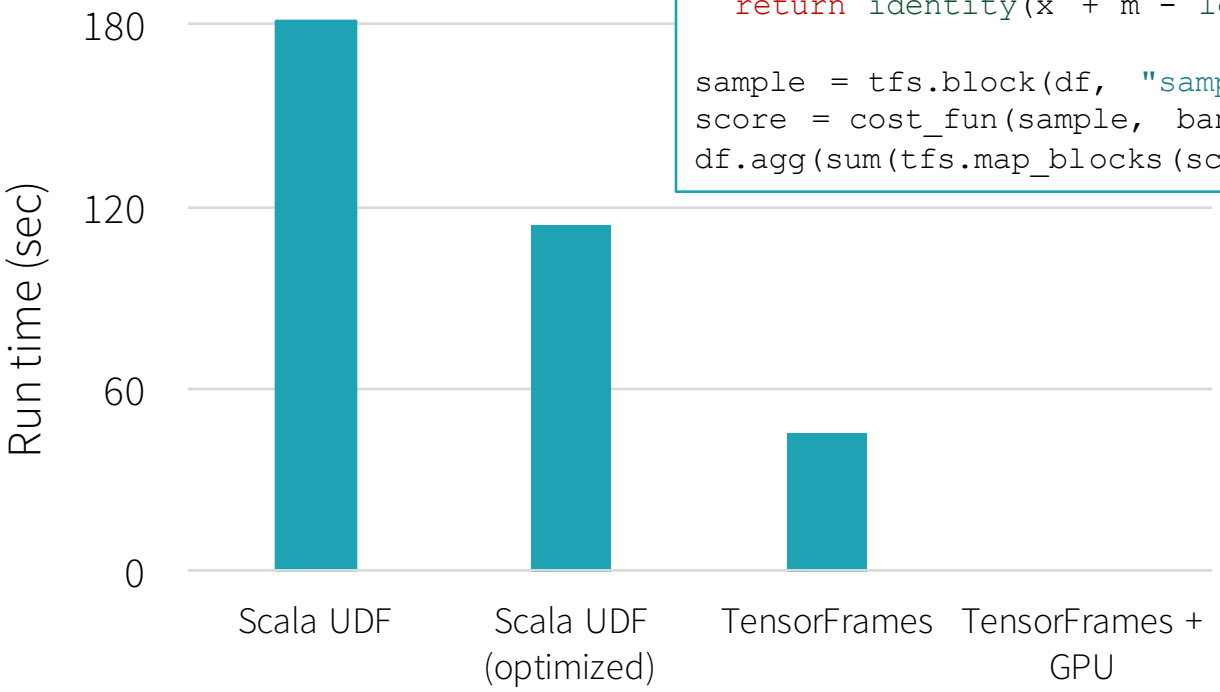


TensorFrames: native embedding of TensorFlow



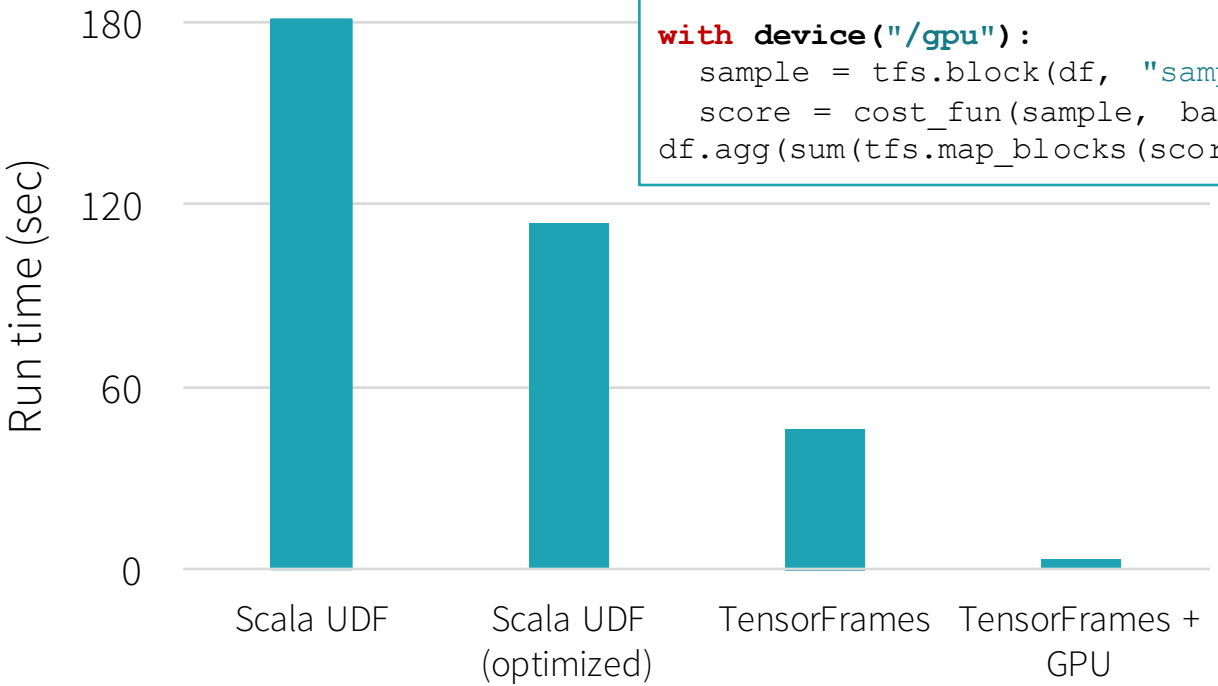
Speedup

```
def cost_fun(block, bandwidth):
  distances = - square(constant(X) - sample) / (2 * b * b)
  m = reduce_max(distances, 0)
  x = log(reduce_sum(exp(distances - m), 0))
  return identity(x + m - log(b * N), name="score”)

sample = tfs.block(df, "sample")
score = cost_fun(sample, bandwidth=0.5)
df.agg(sum(tfs.map_blocks(score, df))).collect()
```



Speedup

```
def cost_fun(block, bandwidth):
  distances = - square(constant(X) - sample) / (2 * b * b)
  m = reduce_max(distances, 0)
  x = log(reduce_sum(exp(distances - m), 0))
  return identity(x + m - log(b * N), name="score”)

with device("/gpu"):
  sample = tfs.block(df, "sample")
  score = cost_fun(sample, bandwidth=0.5)
df.agg(sum(tfs.map_blocks(score, df))).collect()
```



Recap

- Spark: an efficient framework for running computations on thousands of computers
- TensorFlow: high-performance numerical framework
- Get the best of both with TensorFrames:
  - Simple API for distributed numerical computing
  - Can leverage the hardware of the cluster

Try these demos yourself

- TensorFrames source code and documentation:
- github.com/tjhunter/tensorframes
  spark-packages.org/package/tjhunter/tensorframes

- Demo available later on Databricks

- The official TensorFlow website:
  www.tensorflow.org

Join us at Spark Summit Europe

October 25-27, 2016 | Brussels

Apply to the Academic Partners Program

databricks.com/academic

Thanks!