Streaming Big Data with Spark Streaming, Kafka, Cassandra and Akka

Helena Edelson

@helenaedelson

©2014 DataStax Confidential. Do not distribute without consent.

Who Is This Person?

DATASTAX

- Spark Cassandra Connector committer
- Akka contributor (Akka Cluster)
- Scala & Big Data conference speaker
- Sr Software Engineer, Analytics @ DataStax
- Sr Cloud Engineer, VMware,CrowdStrike,SpringSource…
- (Prev) Spring committer - Spring AMQP, Spring Integration

Use Case: Hadoop + Scalding

DATASTAX

```
/** Reads SequenceFile data from S3 buckets, computes then persists to Cassandra. */
class TopSearches(args: Args) extends TopKDailyJob[MyDataType](args) with Cassandra {

 PailSource.source[Search](rootpath, structure, directories).read
  .mapTo('pailItem -> 'engines) { e: Search ⇒ results(e) }
  .filter('engines) { e: String ⇒ e.nonEmpty }
  .groupBy('engines) { _.size('count).sortBy('engines) }
  .groupBy('engines) { _.sortedReverseTake[(String, Long)](('engines, 'count) -> 'tcount, k) }
  .flatMapTo('tcount -> ('key, 'engine, 'topCount)) { t: List[(String, Long)] ⇒
      t map { case (k, v) ⇒ (jobKey, k, v) }}
  .write(CassandraSource(connection, "top_searches", Scheme(‘key, ('engine, ‘topCount))))

}
```

Use Case: Spark?

DATASTAX

```
20  object SparkWordCount extends WordCountBlueprint {
21
22    sc.textFile("./src/main/resources/data/words")
23      .flatMap(_.split("\\s+"))
24      .map(word => (clean(word), 1))
25      .reduceByKey(_ + _)
26      .saveAsTextFile(s"./wordcount-$timestamp")
27  }
```

Talk Roadmap

DATASTAX

What Delivering Meaning

Why Spark, Kafka, Cassandra & Akka

How Composable Pipelines

App Robust Implementation

Sending Data Between Systems Is

~~Difficult~~ Risky

©2014 DataStax Confidential. Do not distribute without consent.

Fault-tolerance

by @jrecursive

Strategies

DATASTAX

- Scalable Infrastructure
- Partition For Scale
- Replicate For Resiliency
- Share Nothing
- Asynchronous Message Passing
- Parallelism
- Isolation
- Location Transparency

Strategy	Technologies
Scalable Infrastructure / Elastic scale on demand	Spark, Cassandra, Kafka
Partition For Scale, Network Topology Aware	Cassandra, Spark, Kafka, Akka Cluster
Replicate For Resiliency span racks and datacenters, survive regional outages	Spark,Cassandra, Akka Cluster all hash the node ring
Share Nothing, Masterless	Cassandra, Akka Cluster both Dynamo style
Fault Tolerance / No Single Point of Failure	Spark, Cassandra, Kafka
Replay From Any Point Of Failure	Spark, Cassandra, Kafka, Akka + Akka Persistence
Failure Detection	Cassandra, Spark, Akka, Kafka
Consensus & Gossip	Cassandra & Akka Cluster
Parallelism	Spark, Cassandra, Kafka, Akka
Asynchronous Data Passing	Kafka, Akka, Spark
Fast, Low Latency, Data Locality	Cassandra, Spark, Kafka
Location Transparency	Akka, Spark, Cassandra, Kafka

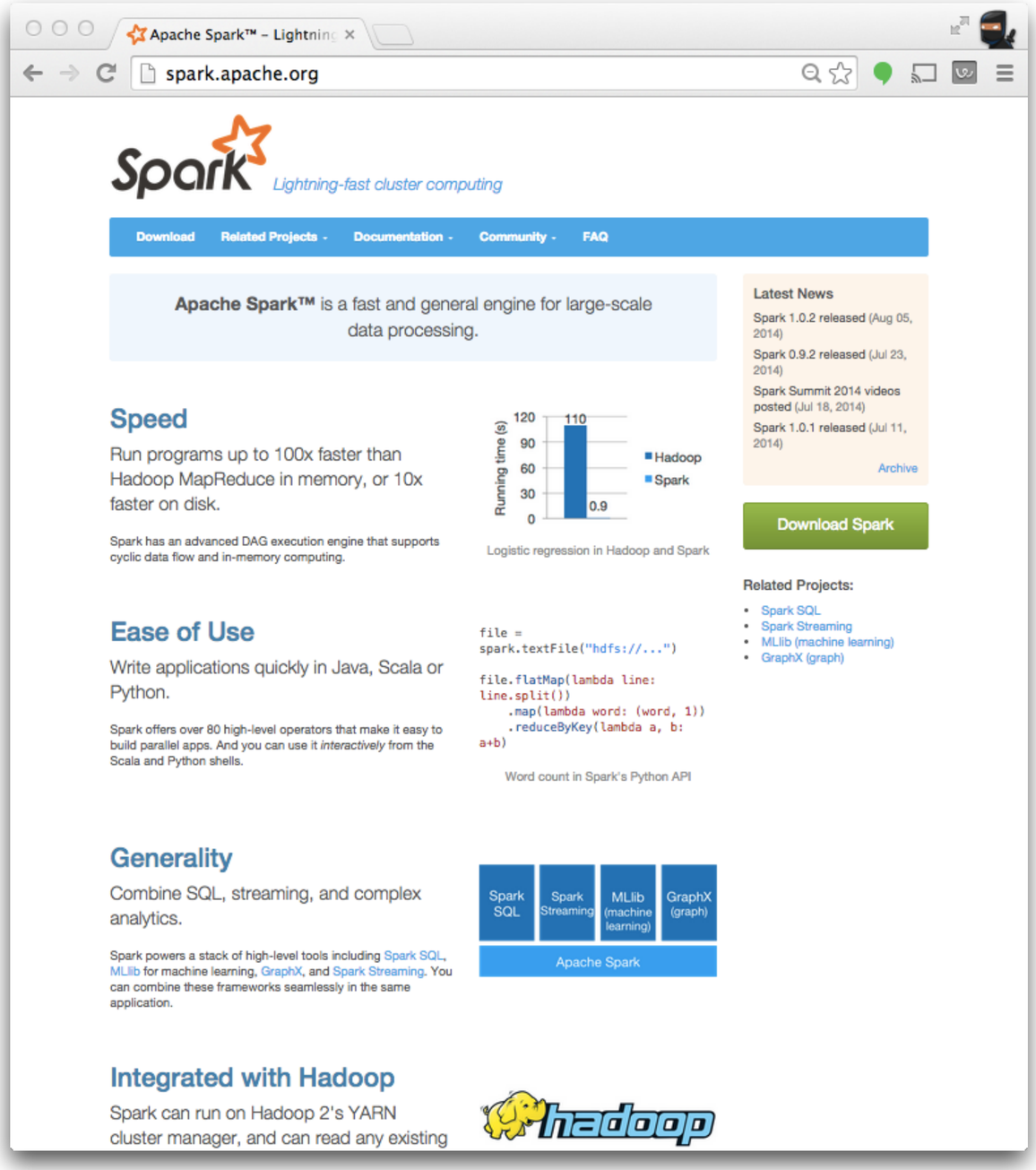
Lambda Architecture

A data-processing architecture designed to handle *massive quantities* of data by taking advantage of both batch and stream processing methods.

- Spark is one of the few data processing frameworks that allows you to seamlessly integrate batch and stream processing
  - Of petabytes of data
  - In the same application

I need fast access to **historical data** on the fly for predictive modeling with **real time data** from the **stream**

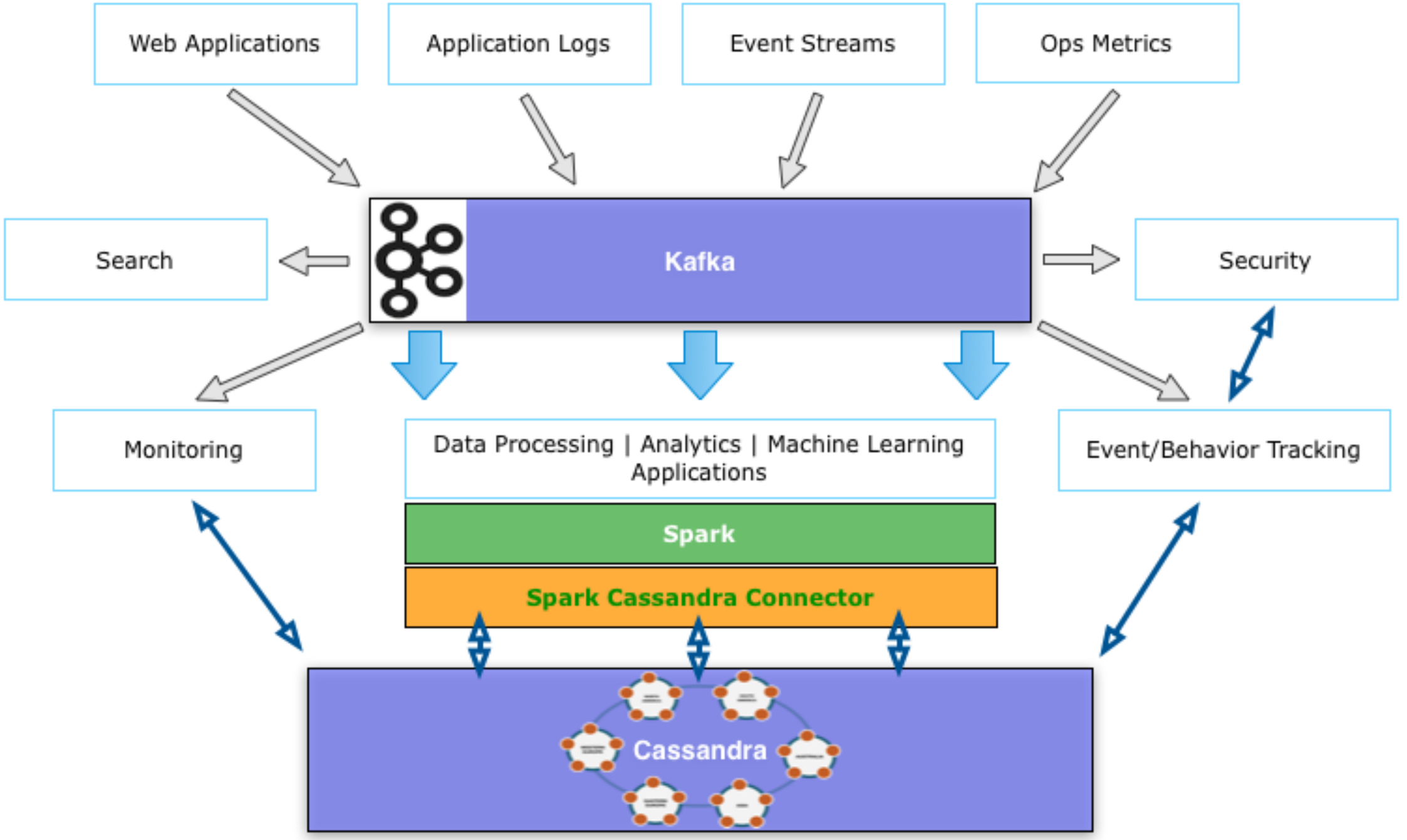
- Fast, distributed, scalable and fault tolerant cluster compute system
- Enables Low-latency with complex analytics
- Developed in 2009 at UC Berkeley AMPLab, open sourced in 2010, and became a top-level Apache project in February, 2014



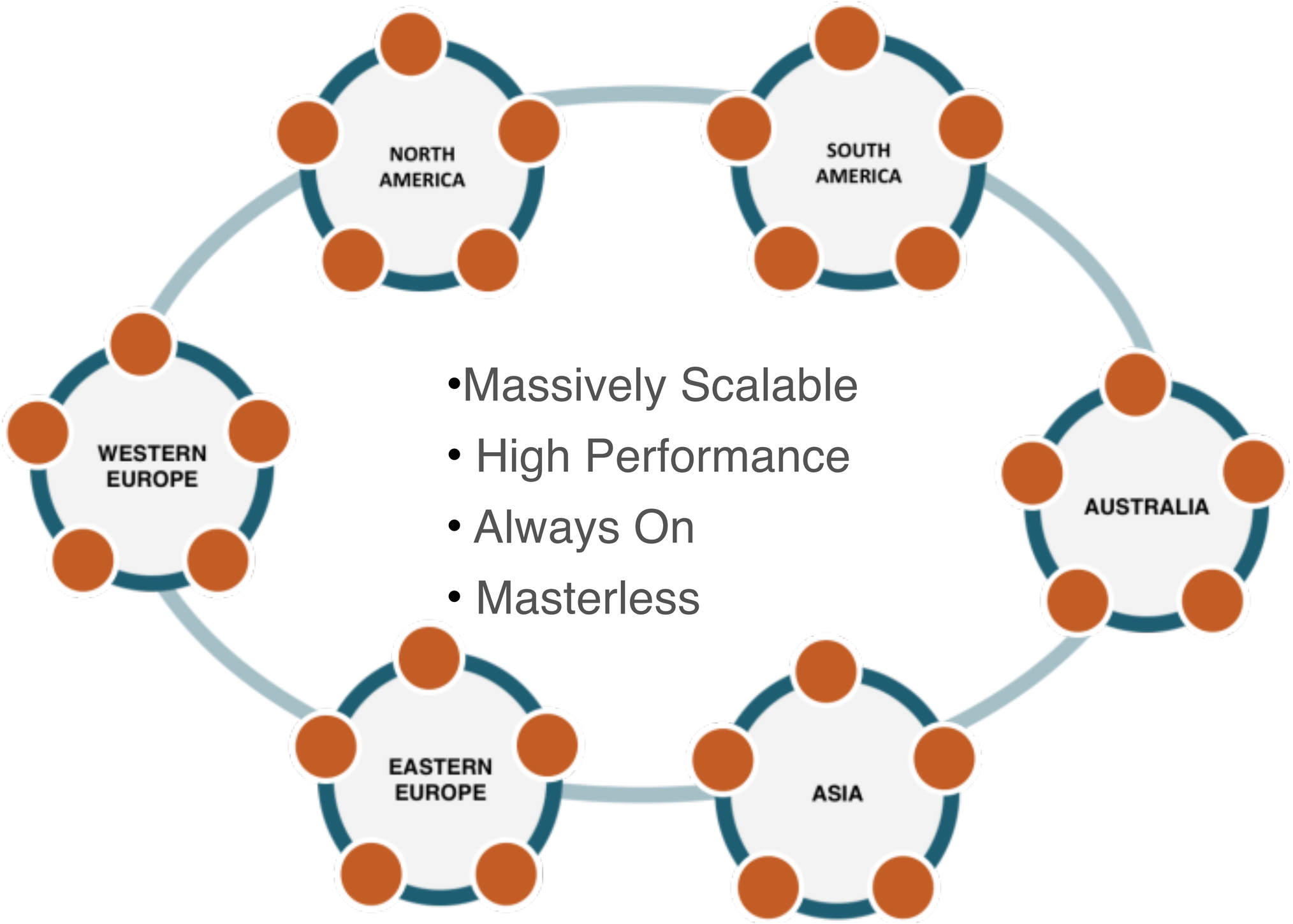
kafka

DATASTAX

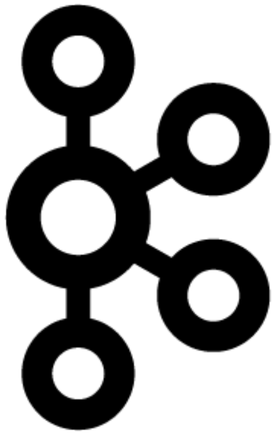
- High Throughput Distributed Messaging
- Decouples Data Pipelines
- Handles Massive Data Load
- Support Massive Number of Consumers
- Distribution & partitioning across cluster nodes
- Automatic recovery from broker failures


Web Applications
Application Logs
Event Streams
Ops Metrics
Search
Kafka
Security
Monitoring
Data Processing | Analytics | Machine Learning Applications
Event/Behavior Tracking
Spark
Spark Cassandra Connector
Cassandra

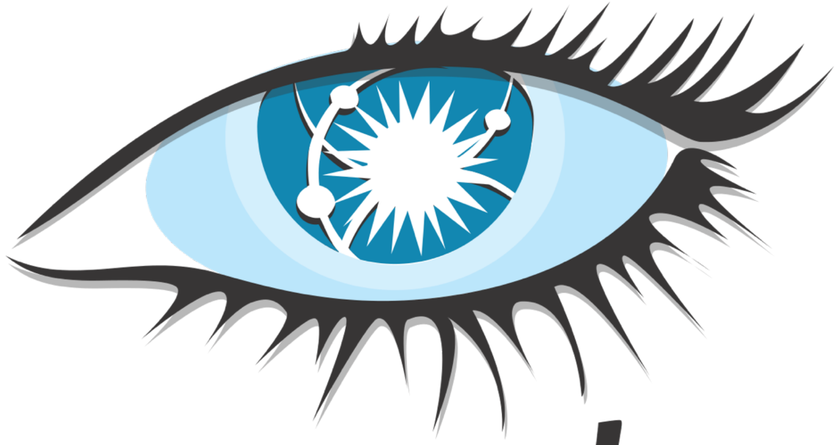
The one thing in your infrastructure you can *always* rely on.


NORTH AMERICA
SOUTH AMERICA
WESTERN EUROPE
AUSTRALIA
EASTERN EUROPE
ASIA
•Massively Scalable
• High Performance
• Always On
• Masterless

Spark Cassandra Connector



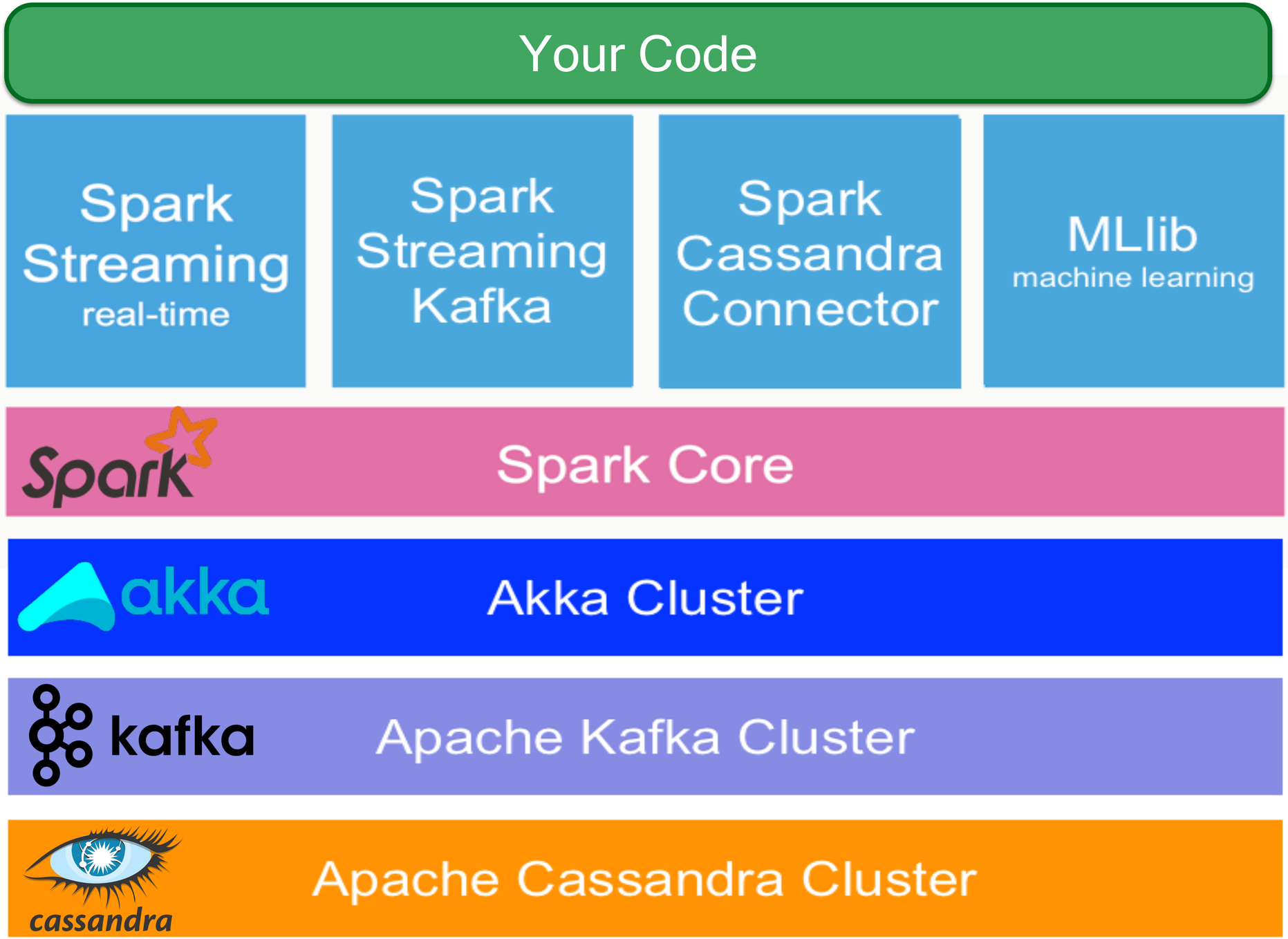
kafka

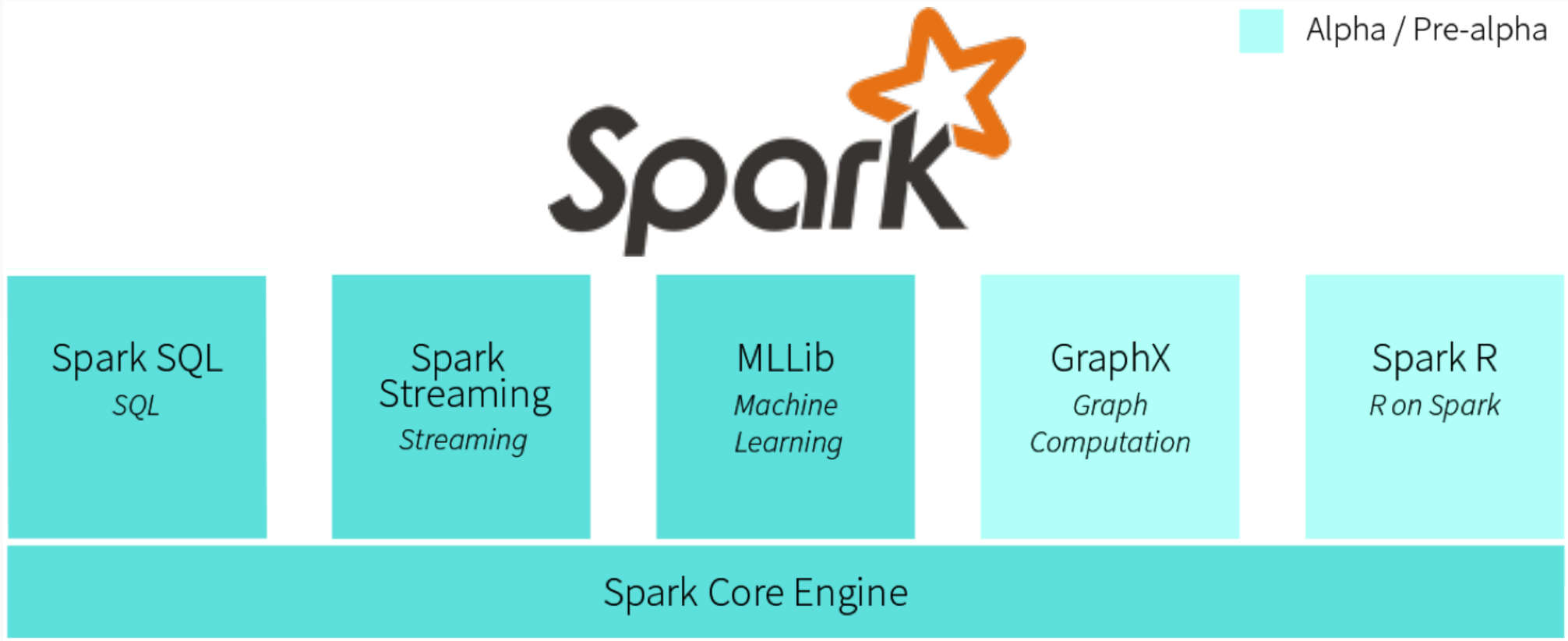


cassandra

- Fault tolerant
  - Hierarchical Supervision
  - Customizable Failure Strategies & Detection
- Asynchronous Data Passing
- Parallelization - Balancing Pool Routers
- Akka Cluster
  - Adaptive / Predictive
  - Load-Balanced Across Cluster Nodes

- Stream data from Kafka to Cassandra
- Stream data from Kafka to Spark and write to Cassandra
- Stream from Cassandra to Spark - coming soon!
- Read data from Spark/Spark Streaming Source and write to C*
- Read data from Cassandra to Spark


Your Code
Spark Streaming
real-time
Spark Streaming Kafka
Spark Cassandra Connector
MLlib
machine learning
Spark
Spark Core
akka
Akka Cluster
kafka
Apache Kafka Cluster
cassandra
Apache Cassandra Cluster


Alpha / Pre-alpha
Spark
Spark SQL
SQL
Spark Streaming
Streaming
MLLib
Machine Learning
GraphX
Graph Computation
Spark R
R on Spark
Spark Core Engine

Most Active OSS In Big Data

DATASTAX

apache / spark
mirrored from git://git.apache.org/spark.git

Watch 427 | Star 2,285 | Fork 2,044

November 11, 2014 – December 11, 2014

Period: **1 month**

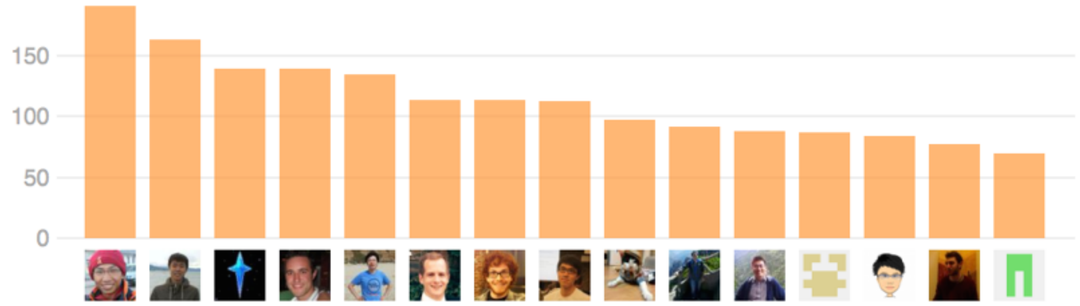
Overview

154 Active Pull Requests

0 Active Issues

0	154	0	0
Merged Pull Requests	Proposed Pull Requests	Closed Issues	New Issues

Excluding merges, **317 authors** have pushed **297 commits** to master and **3,546 commits** to all branches. On master, **702 files** have changed and there have been **38,458 additions** and **18,021 deletions**.



154 Pull requests proposed by **82** people

Apache Spark - Easy to Use API

DATASTAX

Returns the top (k) highest temps for any location in the year

```
def topK(aggregate: Seq[Double]): Seq[Double] =
  sc.parallelize(aggregate).top(k).collect
```

Returns the top (k) highest temps … in a Future

```
def topK(aggregate: Seq[Double]): Future[Seq[Double]] =
  sc.parallelize(aggregate).top(k).collectAsync
```

Not Just MapReduce

DATASTAX

def **aggregate**[U](zeroValue: U)(seqOp: (U, T) ⇒ U, combOp
Aggregate the elements of each partition, and then the results for all th

def **cache**(): RDD.this.type
Persist this RDD with the default storage level (MEMORY_ONLY).

def **cartesian**[U](other: RDD[U])(*implicit* arg0: ClassTag[
Return the Cartesian product of this RDD and another one, that is, the

def **checkpoint**(): Unit
Mark this RDD for checkpointing.

def **coalesce**(numPartitions: Int, shuffle: Boolean = fals
Return a new RDD that is reduced into numPartitions partitions.

def **collect**[U](f: PartialFunction[T, U])(*implicit* arg0:
Return an RDD that contains all matching values by applying f.

def **collect**(): Array[T]
Return an array that contains all of the elements in this RDD.

def **context**: SparkContext
The org.apache.spark.SparkContext that this RDD was created on.

def **count**(): Long
Return the number of elements in the RDD.

def **countApprox**(timeout: Long, confidence: Double = 0.95
Approximate version of count() that returns a potentially incomplete res

def **countApproxDistinct**(relativeSD: Double = 0.05): Long
Return approximate number of distinct elements in the RDD.

def **countApproxDistinct**(p: Int, sp: Int): Long
Return approximate number of distinct elements in the RDD.

def **countByValue**()(*implicit* ord: Ordering[T] = null): Ma
Return the count of each unique value in this RDD as a map of (value,

def **countByValueApprox**(timeout: Long, confidence: Double
Approximate version of countByValue().

def **dependencies**: Seq[Dependency[_]]
Get the list of dependencies of this RDD, taking into account whether t

def **distinct**(): RDD[T]
Return a new RDD containing the distinct elements in this RDD.

def **glom**(): RDD[Array[T]]
Return an RDD created by coalescing all elements within each parti

def **groupBy**[K](f: (T) ⇒ K, p: Partitioner)(*implicit* kt
Return an RDD of grouped items.

def **groupBy**[K](f: (T) ⇒ K, numPartitions: Int)(*implici*
Return an RDD of grouped elements.

def **groupBy**[K](f: (T) ⇒ K)(*implicit* kt: ClassTag[K]):
Return an RDD of grouped items.

val **id**: Int
A unique ID for this RDD (within its SparkContext).

def **intersection**(other: RDD[T], numPartitions: Int): R
Return the intersection of this RDD and another one.

def **intersection**(other: RDD[T], partitioner: Partition
Return the intersection of this RDD and another one.

def **intersection**(other: RDD[T]): RDD[T]
Return the intersection of this RDD and another one.

def **isCheckpointed**: Boolean
Return whether this RDD has been checkpointed or not

def **iterator**(split: Partition, context: TaskContext):
Internal method to this RDD; will read from cache if applicable, or otl

def **keyBy**[K](f: (T) ⇒ K): RDD[(K, T)]
Creates tuples of the elements in this RDD by applying f.

def **map**[U](f: (T) ⇒ U)(*implicit* arg0: ClassTag[U]): RD
Return a new RDD by applying a function to all elements of this RDD

def **mapPartitions**[U](f: (Iterator[T]) ⇒ Iterator[U], p
Return a new RDD by applying a function to each partition of this RD

def **mapPartitionsWithContext**[U](f: (TaskContext, Itera
Return a new RDD by applying a function to each partition of this RD

def **mapPartitionsWithIndex**[U](f: (Int, Iterator[T]) ⇒
Return a new RDD by applying a function to each partition of this RD

def **max**()(*implicit* ord: Ordering[T]): T
Returns the max of this RDD as defined by the implicit Ordering[T].

def **min**()(*implicit* ord: Ordering[T]): T
Returns the min of this RDD as defined by the implicit Ordering[T].

var **name**: String
A friendly name for this RDD

val **partitioner**: Option[Partitioner]
Optionally overridden by subclasses to specify how they are partitioned.

def **partitions**: Array[Partition]
Get the array of partitions of this RDD, taking into account whether the RDD is chec

def **persist**(): RDD.this.type
Persist this RDD with the default storage level (MEMORY_ONLY).

def **persist**(newLevel: StorageLevel): RDD.this.type
Set this RDD's storage level to persist its values across operations after the first tim

def **pipe**(command: Seq[String], env: Map[String, String] = Map(), p
Return an RDD created by piping elements to a forked external process.

def **pipe**(command: String, env: Map[String, String]): RDD[String]
Return an RDD created by piping elements to a forked external process.

def **pipe**(command: String): RDD[String]
Return an RDD created by piping elements to a forked external process.

def **preferredLocations**(split: Partition): Seq[String]
Get the preferred locations of a partition (as hostnames), taking into account whethe

def **randomSplit**(weights: Array[Double], seed: Long = Utils.random.
Randomly splits this RDD with the provided weights.

def **reduce**(f: (T, T) ⇒ T): T
Reduces the elements of this RDD using the specified commutative and associative

def **repartition**(numPartitions: Int)(*implicit* ord: Ordering[T] = nu
Return a new RDD that has exactly numPartitions partitions.

def **sample**(withReplacement: Boolean, fraction: Double, seed: Long
Return a sampled subset of this RDD.

def **saveAsObjectFile**(path: String): Unit
Save this RDD as a SequenceFile of serialized objects.

def **saveAsTextFile**(path: String, codec: Class[_ <: CompressionCode
Save this RDD as a compressed text file, using string representations of elements.

Spark Basic Word Count

DATASTAX

```
val conf = new SparkConf()
 .setMaster(host).setAppName(app)

val sc = new SparkContext(conf)

sc.textFile(words)
   .flatMap(_.split("\\s+"))
   .map(word => (word.toLowerCase, 1))
   .reduceByKey(_ + _)
   .collect
```

Collection To RDD

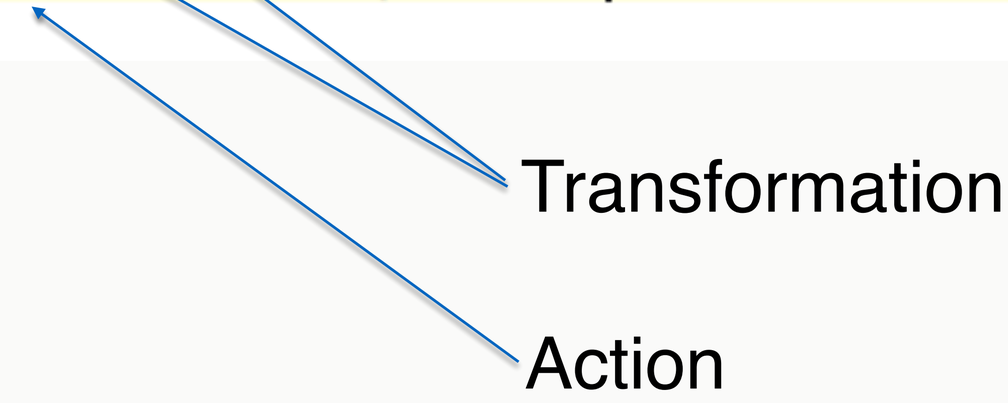
DATASTAX

```
scala> val data = Array(1, 2, 3, 4, 5)
data: Array[Int] = Array(1, 2, 3, 4, 5)

scala> val distributedData = sc.parallelize(data)
distributedData: spark.RDD[Int] =
spark.ParallelCollection@10d13e3e
```

RDD Operations

DATASTAX

```
16  object SparkWordCount extends WordCountBlueprint {
17
18    sc.textFile("./src/main/resources/data/words")
19      .flatMap(_.split("\\s+"))
20      .map(word => (clean(word), 1))
21      .reduceByKey(_ + _)
22      .collect foreach println
23  }
```



When Batch Is Not Enough

Spark Streaming

DATASTAX

- I want results continuously in the event stream
- I want to run computations in my even-driven async apps
- Exactly once message guarantees

Spark Streaming Setup

DATASTAX

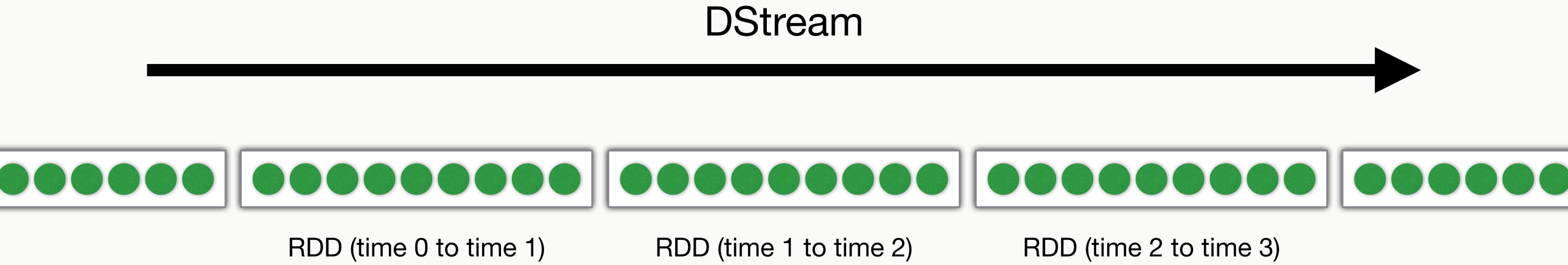
```
val conf = new SparkConf().setMaster(SparkMaster).setAppName(AppName)

val ssc = new StreamingContext(conf, Milliseconds(500))

// Do work in the stream

ssc.checkpoint(checkpointDir)
ssc.start()
ssc.awaitTermination
```

DStream (Discretized Stream)

DATASTAX

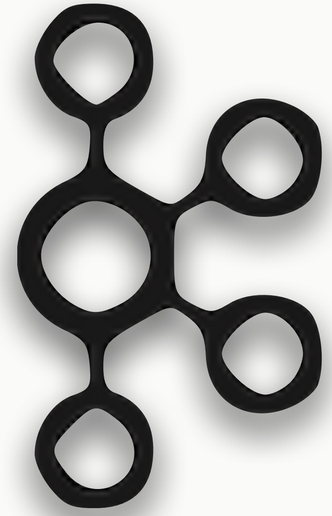
Continuous stream of micro batches

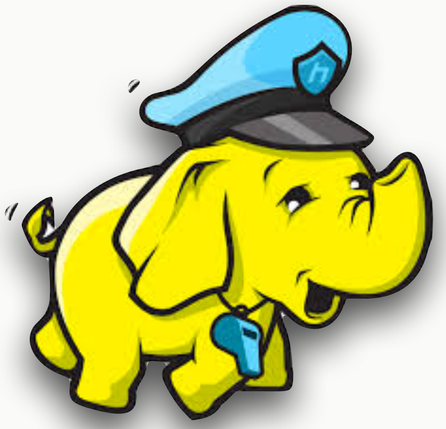
- Complex processing models with minimal effort
- Streaming computations on small time intervals



A transformation on a DStream = transformations on its RDDs

Spark Streaming External Source/Sink





ReceiverInputDStreams

DATASTAX

DStreams - the stream of raw data received from streaming sources:

- Basic Source - in the StreamingContext API
- Advanced Source - in external modules and separate Spark artifacts

Receivers

- Reliable Receivers - for data sources supporting acks (like Kafka)
- Unreliable Receivers - for data sources not supporting acks

Basic Streaming: FileInputDStream

DATASTAX

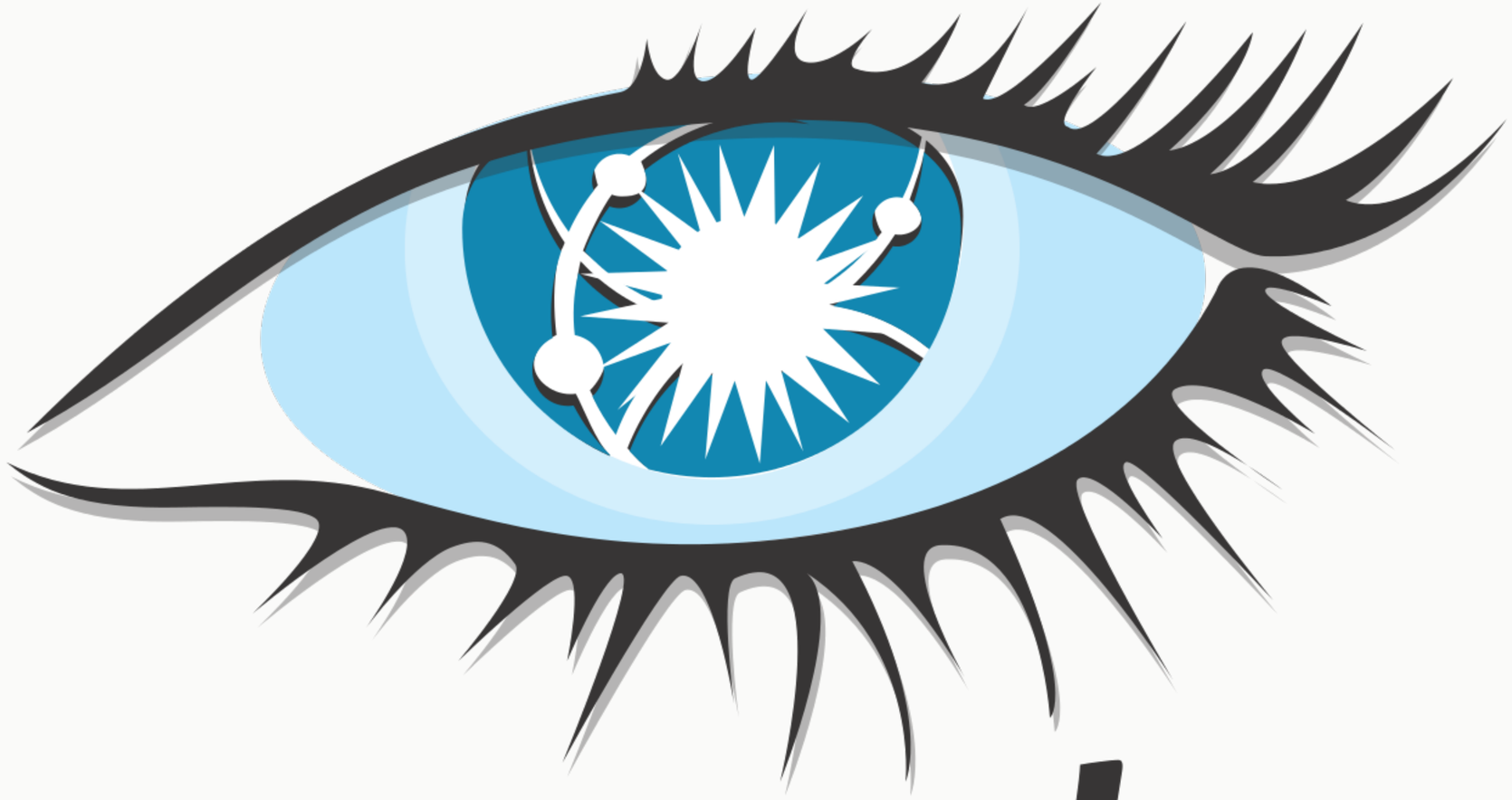
```
// Creates new DStreams
ssc.textFileStream("s3n://raw_data_bucket/")
   .flatMap(_.split("\\s+"))
   .map(_.toLowerCase, 1))
   .countByValue()
   .saveAsObjectFile("s3n://analytics_bucket/")
```

Streaming Window Operations

DATASTAX

```
kvStream
  .flatMap { case (k,v) => (k,v.value) }
  .reduceByKeyAndWindow((a:Int,b:Int) =>
    (a + b), Seconds(30), Seconds(10))
  .saveToCassandra(keyspace,table)
```

DATASTAX



cassandra

Public Cassandra Users

Apache Cassandra

DATASTAX

- Elasticity - scale to as many nodes as you need, when you need
- Always On - No single point of failure, Continuous availability
- Masterless peer to peer architecture
- Replication Across DataCenters
- Flexible Data Storage
- Read and write to any node syncs across the cluster
- Operational simplicity - all nodes in a cluster are the same
- Fast Linear-Scale Performance
- Transaction Support

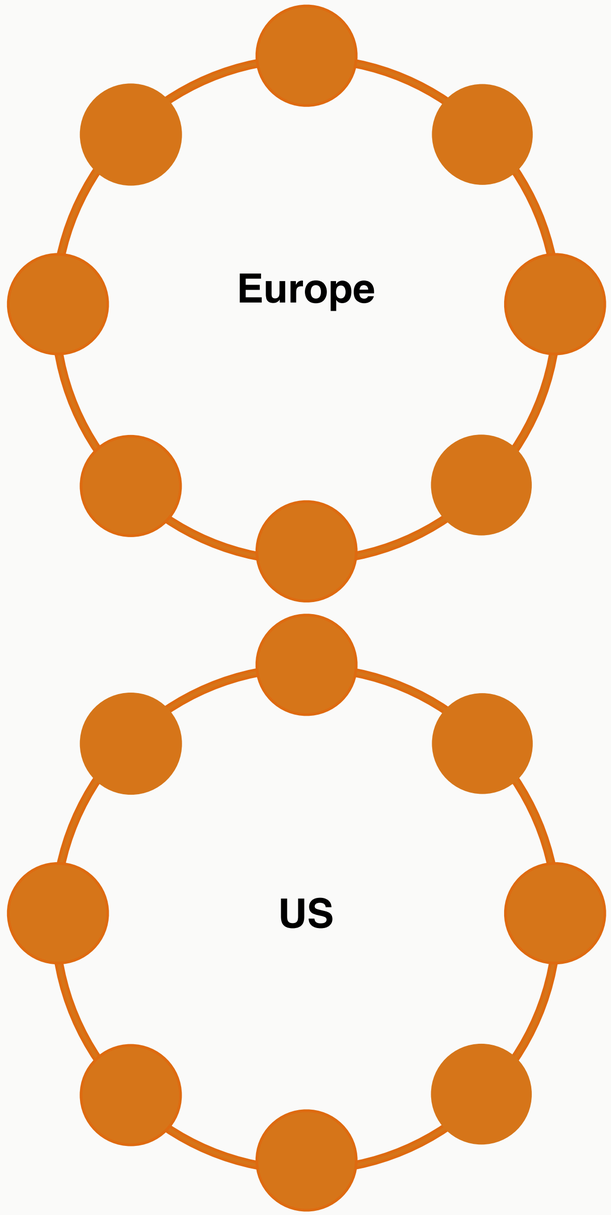
Security, ML, Analytics Workloads

Science

IoT

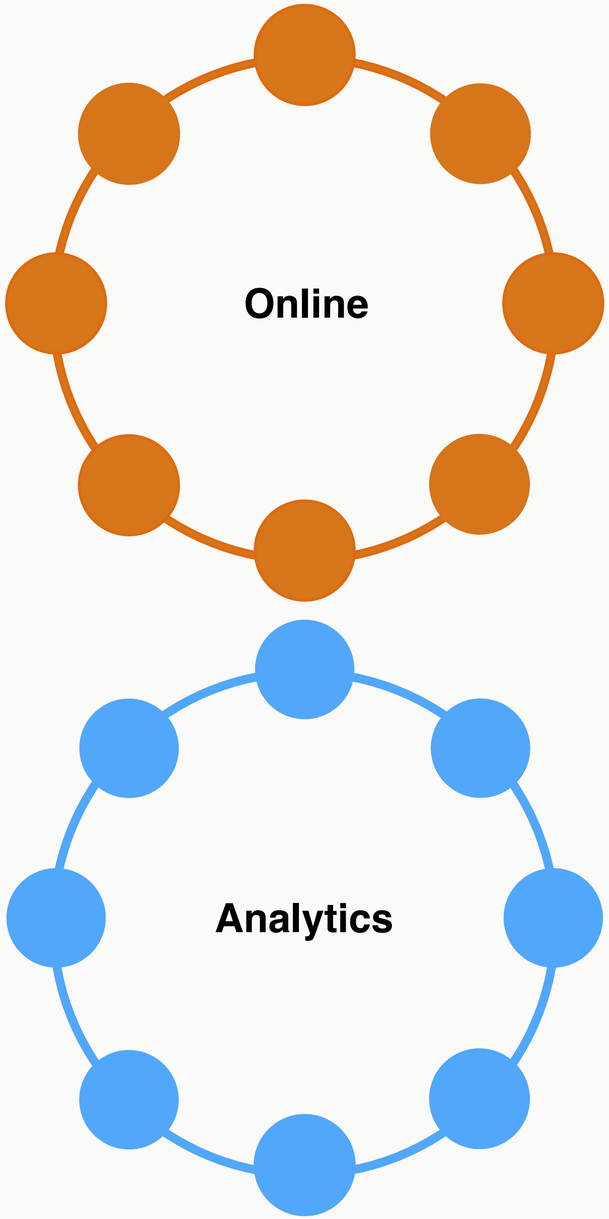

I THINK MY NEST SMOKE ALARM IS GOING OFF. GOOGLE ADWORDS JUST PITCHED ME A FIRE EXTINGUISHER AND AN OFFER FOR TEMPORARY HOUSING.
TOM FISHBURNE
©marketoonist.com

Cassandra



- Availability Model
  - Distributed masterless
- Data Model
  - Google BigTable
- Network Topology Aware
- Multi-Datacenter Replication

Analytics with Spark Over Cassandra



Cassandra enables Spark nodes to transparently communicate across data centers for data

Gossip

Consensus

- Consensus, the agreement among peers on the value of a shared piece of data, is a core building block of Distributed systems
- Cassandra supports consensus via the Paxos protocol

CQL - Easy

DATASTAX

```
CREATE TABLE users (
   username varchar,
   firstname varchar,
   lastname varchar,
   email list<varchar>,
   password varchar,
   created_date timestamp,
   PRIMARY KEY (username)
);
```

- Familiar syntax
- Many Tools & Drivers
- Many Languages
- Friendly to programmers
- Paxos for locking

```
INSERT INTO users (username, firstname, lastname,
 email, password, created_date)
VALUES ('hedelson','Helena','Edelson',
['helena.edelson@datastax.com'],'ba27e03fd95e507daf2937c937d499ab','2014-11-15 13:50:00')
IF NOT EXISTS;
```

Timeseries

DATASTAX

```
CREATE TABLE weather.raw_data (
    wsid text, year int, month int, day int, hour int,
    temperature double, dewpoint double, pressure double,
    wind_direction int, wind_speed double, one_hour_precip
    PRIMARY KEY ((wsid), year, month, day, hour)
) WITH CLUSTERING ORDER BY (year DESC, month DESC, day DESC, hour DESC);
```

C* Clustering Columns

Writes by most recent
Reads return most recent first

Cassandra will automatically sort by most recent for both both write and read

A record of every event, in order in which it happened, per URL:

```
CREATE TABLE IF NOT EXISTS requests_ks.timeline (
  timesegment bigint, url text, t_uuid timeuuid, method text, headers map <text, text>, body text,
  PRIMARY KEY ((url, timesegment) , t_uuid)
);
```

timeuuid protects from simultaneous events over-writing one another.

timesegment protects from writing unbounded partitions.

```
val multipleStreams = (1 to numDstreams).map { i =>
  streamingContext.receiverStream[HttpRequest](new HttpReceiver(port))
}

streamingContext.union(multipleStreams)
.map { httpRequest => TimelineRequestEvent(httpRequest)}
.saveToCassandra("requests_ks", "timeline")
```

Spark Cassandra Connector

©2014 DataStax Confidential. Do not distribute without consent.

Spark Cassandra Connector

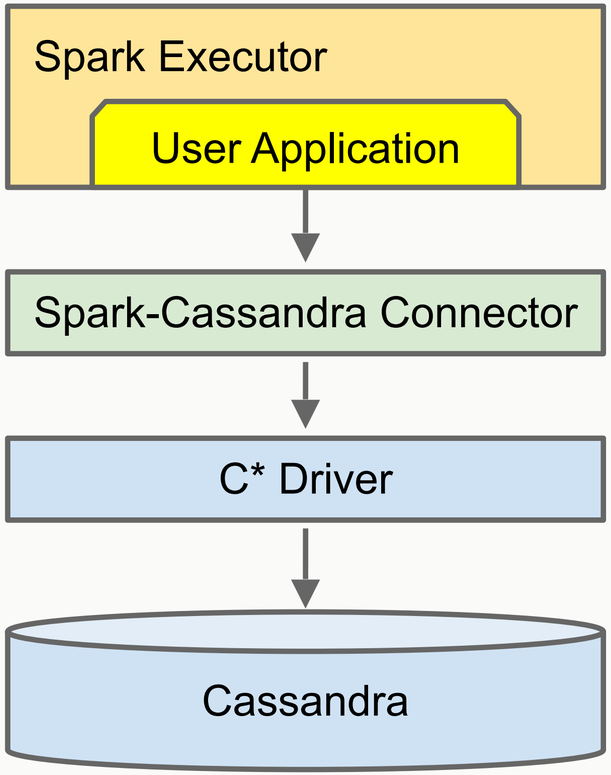
DATASTAX

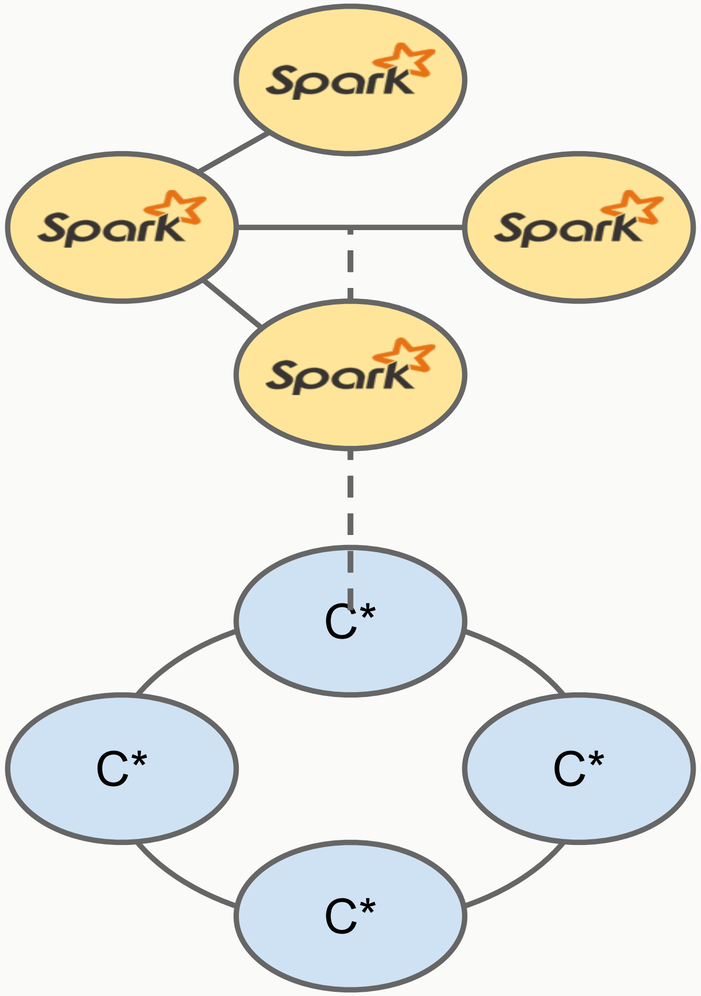
https://github.com/datastax/spark-cassandra-connector

- Write data from Spark to Cassandra
- Read data from Cassandra to Spark
- Data Locality for Speed
- Easy, and often implicit, type conversions
- Server-Side Filtering - SELECT, WHERE, etc.
- Natural Timeseries Integration
- Implemented in Scala

Spark Cassandra Connector

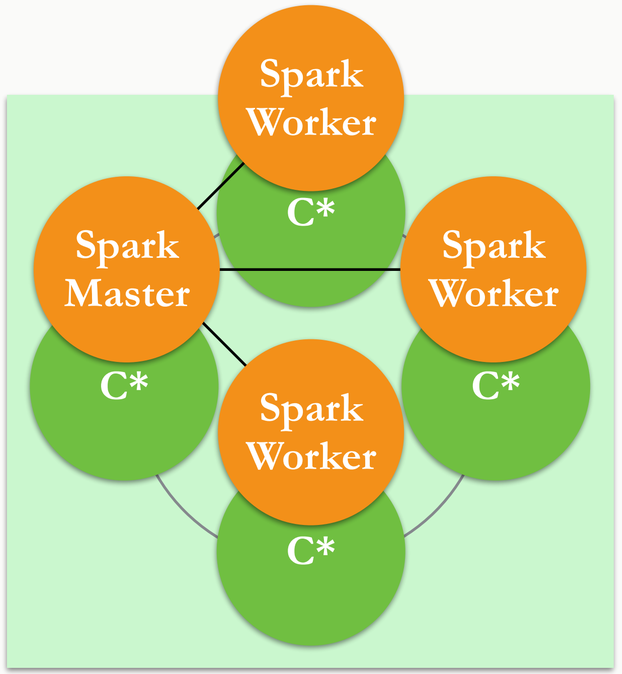
DATASTAX





Co-locate Spark and C* for Best Performance

Running Spark Workers on
the same nodes as your C* Cluster
saves network hops



Writing and Reading

DATASTAX

SparkContext

```
import com.datastax.spark.connector._
```

StreamingContext

```
import com.datastax.spark.connector.streaming._
```

Write from Spark to Cassandra

DATASTAX

SparkContext

Keyspace

Table

```
sc.parallelize(collection).saveToCassandra("keyspace", "raw_data")
```

Spark RDD

JOIN with NOSQL!

```
predictionsRdd.join(music).saveToCassandra("music", "predictions")
```

Read From C* to Spark

DATASTAX

CassandraRDD[CassandraRow]

SparkContext

Keyspace

Table

```
val rdd = sc.cassandraTable("github", "commits")
          .select("user","count","year","month")
          .where("commits >= ? and year = ?", 1000, 2015)
```

Server-Side Column and Row Filtering

Rows: Custom Objects

DATASTAX

StreamingContext CassandraRow Keyspace Table




```
val rdd = ssc.cassandraTable[MonthlyCommits]("github", "commits_aggregate")
              .where("user = ? and project_name = ? and year = ?",
                 "helena", "spark-cassandra-connector", 2015)
```

Rows

DATASTAX

```
val tuplesRdd = sc.cassandraTable[(Int,Date,String)](db, tweetsTable)
 .select("cluster_id","time", "cluster_name")
 .where("time > ? and time < ?",
        "2014-07-12 20:00:01", "2014-07-12 20:00:03”)
```

```
val keyValuesPairsRdd = sc.cassandraTable[(Key,Value)](keyspace, table)
```

Rows

DATASTAX

```
val rdd = ssc.cassandraTable[MyDataType]("stats", "clustering_time")
rdd.where("key = 1").limit(10).collect
rdd.where("key = 1").take(10).collect
```

```
val rdd = ssc.cassandraTable[(Int,DateTime,String)]("stats", "clustering_time")
            .where("key = 1").withAscOrder.collect

val rdd = ssc.cassandraTable[(Int,DateTime,String)]("stats", "clustering_time")
            .where("key = 1").withDescOrder.collect
```

Cassandra User Defined Types

DATASTAX

```
CREATE TYPE address (
  street text,
  city text,
  zip_code int,
  country text,
  cross_streets set<text>
);
```

UDT = Your Custom Field Type In Cassandra

Cassandra UDT's With JSON

DATASTAX

```
{
   "productId": 2,
   "name": "Kitchen Table",
   "price": 249.99,
   "description" : "Rectangular table with oak finish",
   "dimensions": {
      "units": "inches",
      "length": 50.0,
      "width": 66.0,
      "height": 32
   },
   "categories": {
      {
         "category" : "Home Furnishings" {
            "catalogPage": 45,
            "url": "/home/furnishings"
         },
         {
         "category" : "Kitchen Furnishings" {
            "catalogPage": 108,
            "url": "/kitchen/furnishings"
      }
   }
}
```

```
CREATE TYPE dimensions (
   units text,
   length float,
   width float,
   height float
);


CREATE TYPE category (
   catalogPage int,
   url text
);

CREATE TABLE product (
   productId int,
   name text,
   price float,
   description text,
   dimensions frozen <dimensions>,
   categories map <text, frozen <category>>,
   PRIMARY KEY (productId)
);
```

Composable Pipelines
With Spark, Kafka & Cassandra

©2014 DataStax Confidential. Do not distribute without consent.

Spark SQL with Cassandra

DATASTAX

```
import org.apache.spark.sql.cassandra.CassandraSQLContext

val cc = new CassandraSQLContext(sparkContext)
cc.setKeyspace(keyspaceName)
cc.sql("""
  SELECT table1.a, table1.b, table.c, table2.a
  FROM table1 AS table1
  JOIN table2 AS table2 ON table1.a = table2.a
  AND table1.b = table2.b
  AND table1.c = table2.c
  """)
.map(Data(_))
.saveToCassandra(keyspace1, table3)
```

Spark SQL with Cassandra & JSON

DATASTAX

```
cqlsh> CREATE TABLE github_stats.commits_aggr(user VARCHAR PRIMARY KEY, commits INT…);
```

```
val sql = new SQLContext(sparkContext)

val json = Seq(
  """{"user":"helena","commits":98, "month":3, "year":2015}""",
  """{"user":"jacek-lewandowski", "commits":72, "month":3, "year":2015}""",
  """{"user":"pkolaczk", "commits":42, "month":3, "year":2015}""")

// write
sql.jsonRDD(json)
   .map(CommitStats(_))
   .flatMap(compute)
   .saveToCassandra("stats","monthly_commits")

// read
val rdd = sc.cassandraTable[MonthlyCommits]("stats","monthly_commits")
```

Spark Streaming, Kafka, C* and JSON

```
KafkaUtils.createStream[String, String, StringDecoder, StringDecoder](
  ssc, kafkaParams, topicMap, StorageLevel.MEMORY_ONLY)
  .map { case (_,json) => JsonParser.parse(json).extract[MonthlyCommits]}
  .saveToCassandra("github_stats","commits_aggr")
```

```
cqlsh> select * from github_stats.commits_aggr;

 user              | commits | month | year
-------------------+---------+-------+------
           pkolaczk |      42 |     3 | 2015
 jacek-lewandowski |      43 |     3 | 2015
             helena |      98 |     3 | 2015
(3 rows)
```

Spark Streaming, Kafka & Cassandra

DATASTAX

```
sparkConf.set("spark.cassandra.connection.host", "10.20.3.45")
val streamingContext = new StreamingContext(conf, Seconds(30))


KafkaUtils.createStream[String, String, StringDecoder, StringDecoder](
 streamingContext, kafkaParams, topicMap, StorageLevel.MEMORY_ONLY)
  .map(_._2)
  .countByValue()
  .saveToCassandra("my_keyspace","wordcount")
```

Spark Streaming, Twitter & Cassandra

DATASTAX

```
CREATE TABLE IF NOT EXISTS keyspace.table (
  topic text, interval text, mentions counter,
  PRIMARY KEY(topic, interval)
) WITH CLUSTERING ORDER BY (interval DESC)
```

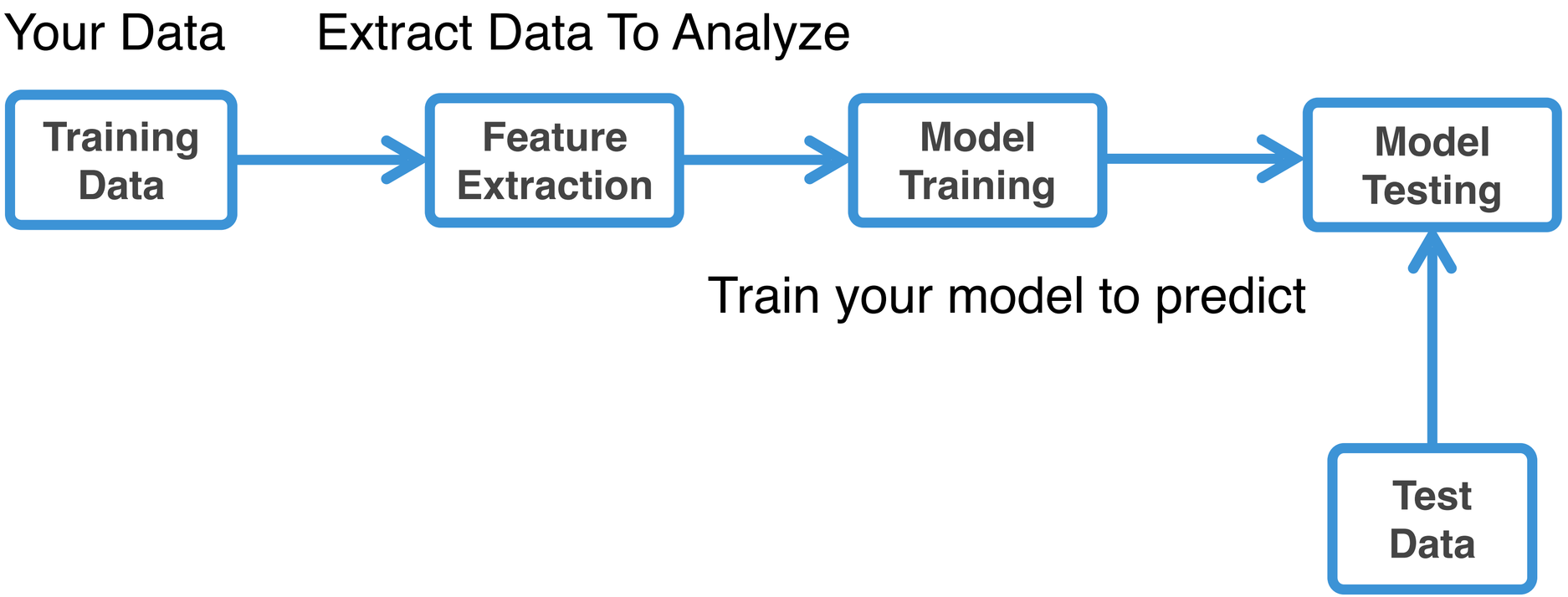
```
/** Cassandra is doing the sorting for you here. */
TwitterUtils.createStream(
    ssc, auth, tags, StorageLevel.MEMORY_ONLY_SER_2)
  .flatMap(_.getText.toLowerCase.split("""\s+"""))
  .filter(tags.contains(_))
  .countByValueAndWindow(Seconds(5), Seconds(5))
  .transform((rdd, time) =>
      rdd.map { case (term, count) => (term, count, now(time))})
  .saveToCassandra(keyspace, table)
```

Streaming From Kafka, R/W Cassandra

DATASTAX

```
val ssc = new StreamingContext(conf, Seconds(30))

val stream = KafkaUtils.createStream[K, V, KDecoder, VDecoder](
    ssc, kafkaParams, topicMap, StorageLevel.MEMORY_ONLY)

stream.flatMap { detected =>
    ssc.cassandraTable[AdversaryAttack]("behavior_ks", "observed")
       .where("adversary = ? and ip = ? and attackType = ?",
        detected.adversary, detected.originIp, detected.attackType)
       .collect
}.saveToCassandra("profiling_ks", "adversary_profiles")
```

Spark Streaming ML, Kafka & Cassandra



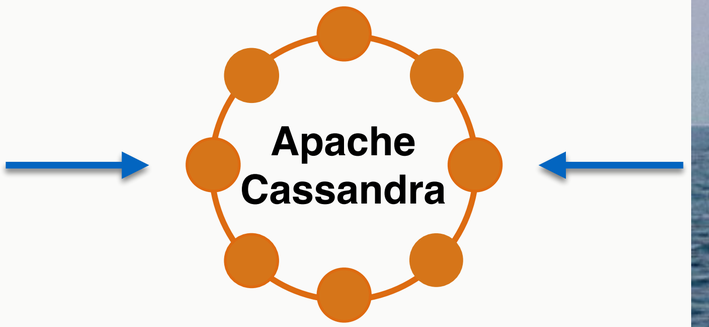
Spark Streaming ML, Kafka & C*

DATASTAX

```
val ssc = new StreamingContext(new SparkConf()…, Seconds(5)

val testData = ssc.cassandraTable[String](keyspace,table).map(LabeledPoint.parse)

val trainingStream = KafkaUtils.createStream[K, V, KDecoder, VDecoder](
                      ssc, kafkaParams, topicMap, StorageLevel.MEMORY_ONLY)
                     .map(_._2).map(LabeledPoint.parse)

trainingStream.saveToCassandra("ml_keyspace", “raw_training_data")

val model = new StreamingLinearRegressionWithSGD()
  .setInitialWeights(Vectors.dense(weights))
  .trainOn(trainingStream)


//Making predictions on testData
  model
  .predictOnValues(testData.map(lp => (lp.label, lp.features)))
  .saveToCassandra("ml_keyspace", "predictions")
```

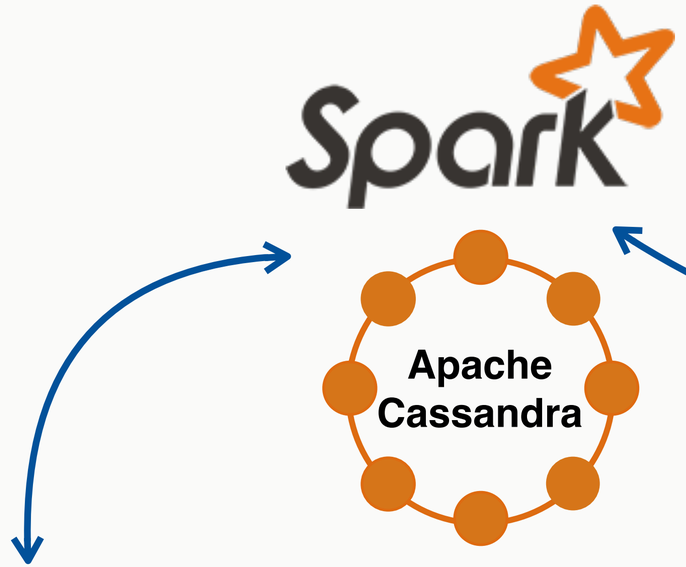
Timeseries Data Application

- Global sensors & satellites collect data
- Cassandra stores in sequence
- Application reads in sequence



Data Analysis

DATASTAX



Application predictive modelling

Data model should look like your queries

Queries I Need

- Get data by ID
- Get data for a single date and time
- Get data for a window of time
- Compute, store and retrieve daily, monthly, annual aggregations

Design Data Model to support queries

- Store raw data per ID
- Store time series data in order: most recent to oldest
- Compute and store aggregate data in the stream
- Set TTLs on historic data

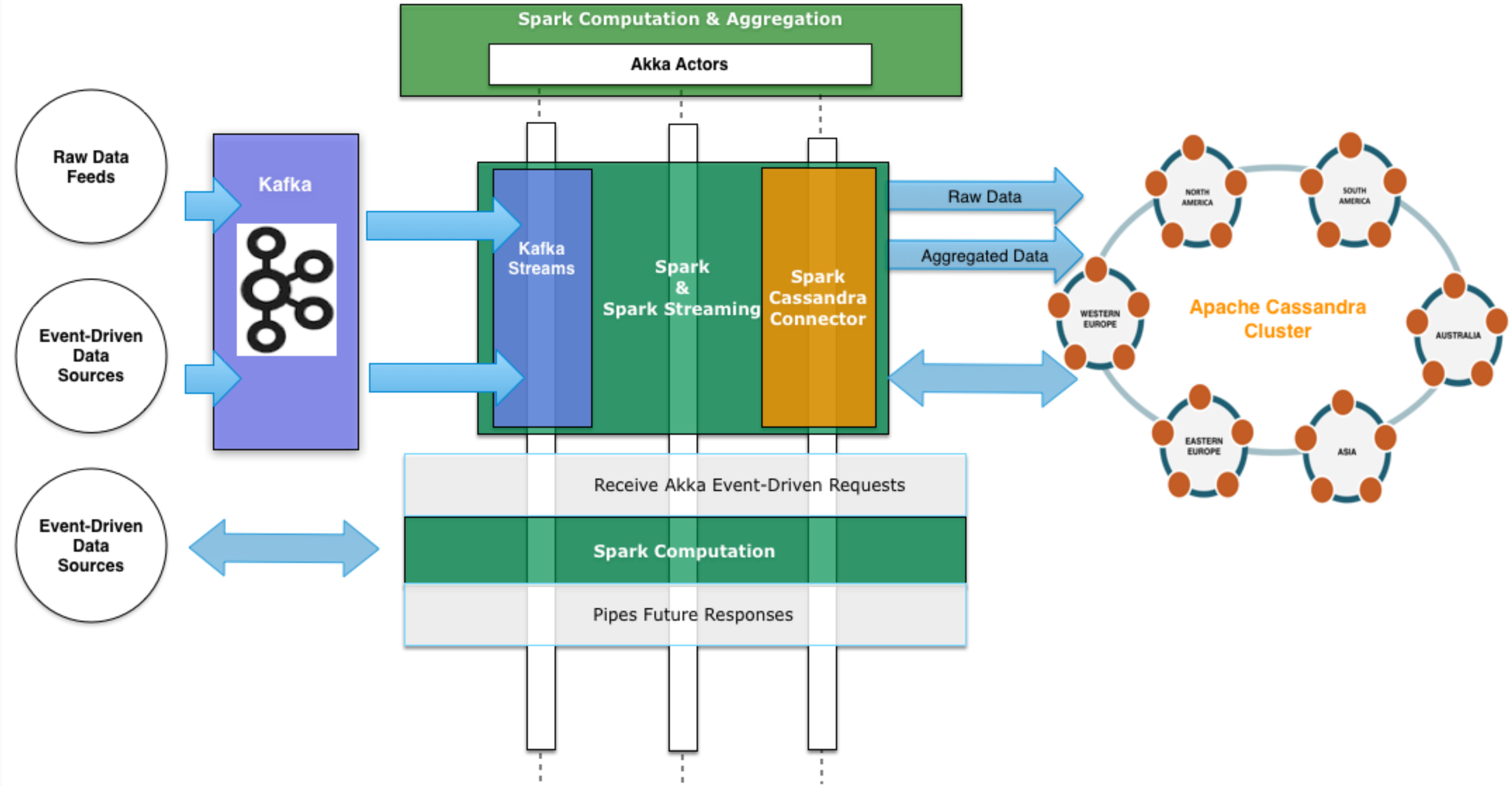
Data Model

DATASTAX

```
CREATE TABLE temperature (
    weather_station text,
    year int,
    month int,
    day int,
    hour int,
    temperature double,
    PRIMARY KEY (weather_station,year,month,day,hour)
);
```

- Weather Station Id and Time are unique
- Store as many as needed

```
INSERT INTO temperature(weather_station,year,month,day,hour,temperature)
VALUES ('10010:99999',2005,12,1,7,-5.6);

INSERT INTO temperature(weather_station,year,month,day,hour,temperature)
VALUES ('10010:99999',2005,12,1,8,-5.1);

INSERT INTO temperature(weather_station,year,month,day,hour,temperature)
VALUES ('10010:99999',2005,12,1,9,-4.9);

INSERT INTO temperature(weather_station,year,month,day,hour,temperature)
VALUES ('10010:99999',2005,12,1,10,-5.3);
```


Spark Computation & Aggregation
Akka Actors
Raw Data Feeds
Event-Driven Data Sources
Event-Driven Data Sources
Kafka
Kafka Streams
Spark & Spark Streaming
Spark Cassandra Connector
Raw Data
Aggregated Data
NORTH AMERICA
SOUTH AMERICA
WESTERN EUROPE
Apache Cassandra Cluster
AUSTRALIA
EASTERN EUROPE
ASIA
Receive Akka Event-Driven Requests
Spark Computation
Pipes Future Responses

Kafka Producer as Akka Actor

DATASTAX

```
class KafkaProducerActor[K, V](config: ProducerConfig) extends Actor {

  override val supervisorStrategy =
    OneForOneStrategy(maxNrOfRetries = 10, withinTimeRange = 1.minute) {
      case _: ActorInitializationException   => Stop
      case _: FailedToSendMessageException   => Restart
      case _: ProducerClosedException        => Restart
      case _: NoBrokersForPartitionException => Escalate
      case _: KafkaException                 => Escalate
      case _: Exception                      => Escalate
    }

  private val producer = new KafkaProducer[K, V](producerConfig)

  override def postStop(): Unit = producer.close()

  def receive = {
    case e: KafkaMessageEnvelope[K,V] => producer.send(e)
  }
}
```

Akka Actor as REST Endpoint

DATASTAX

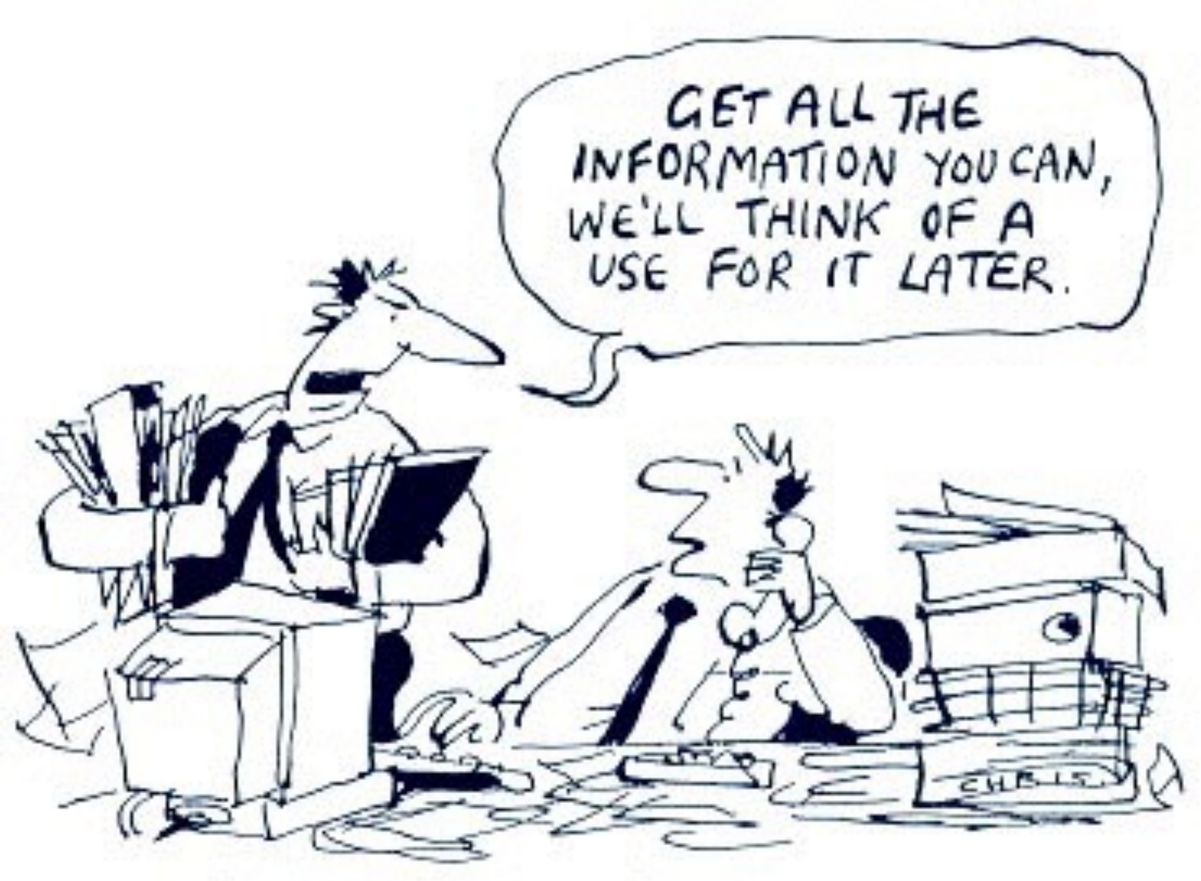
```
class HttpReceiverActor(kafka: ActorRef) extends Actor with ActorLogging {
  implicit val materializer = FlowMaterializer()

  IO(Http) ! Http.Bind(HttpHost, HttpPort)

  val requestHandler: HttpRequest => HttpResponse = {
    case HttpRequest(POST, Uri.Path("/weather/v1/hourly-weather"), headers, entity, _) =>
      HttpSource(headers,entity).collect { case s: HeaderSource =>
        for(s <- source.extract)
          kafka ! KafkaMessageEnvelope[String, String](topic, key, fs.data:_*)

      HttpResponse(200, entity = HttpEntity(MediaTypes.`text/html`, s"POST $entity"))
    }
    getOrElse HttpResponse(404, entity = s"Unsupported request")
  }

  def receive : Actor.Receive = {
    case Http.ServerBinding(localAddress, stream) =>
      Source(stream).foreach({
       case Http.IncomingConnection(remoteAddress, requestProducer, responseConsumer) =>
         log.info("Accepted new connection from {}.", remoteAddress)
         Source(requestProducer).map(requestHandler).to(Sink(responseConsumer)).run()
   })
 }}
```

Akka: Load-Balanced Kafka Work

DATASTAX

```
class HttpNodeGuardian extends ClusterAwareNodeGuardianActor {

  val router = context.actorOf(
    BalancingPool(PoolSize).props(Props(
      new KafkaPublisherActor(KafkaHosts, KafkaBatchSendSize))))


  Cluster(context.system) registerOnMemberUp {

    val router = context.actorOf(
      BalancingPool(PoolSize).props(Props(
        new HttpReceiverActor(KafkaHosts, KafkaBatchSendSize))))
  }

  def initialized: Actor.Receive = { … }

}
```

Store raw data on ingestion



Store Raw Data on Ingestion To Cassandra From Kafka Stream

```
val kafkaStream = KafkaUtils.createStream[K, V, KDecoder, VDecoder]
  (ssc, kafkaParams, topicMap, StorageLevel.DISK_ONLY_2)
  .map(transform)
  .map(RawWeatherData(_))

/** Saves the raw data to Cassandra. */
kafkaStream.saveToCassandra(keyspace, raw_ws_data)
```

Now we can replay: on failure, for later computation, etc

```
/** Now proceed with computations from the same stream.. */
kafkaStream…
```

Our Data Model Again…

DATASTAX

```
CREATE TABLE weather.raw_data (
    wsid text, year int, month int, day int, hour int,
    temperature double, dewpoint double, pressure double,
    wind_direction int, wind_speed double, one_hour_precip
    PRIMARY KEY ((wsid), year, month, day, hour)
) WITH CLUSTERING ORDER BY (year DESC, month DESC, day DESC, hour DESC);
```

```
CREATE TABLE daily_aggregate_precip (
    wsid text,
    year int,
    month int,
    day int,
    precipitation counter,
    PRIMARY KEY ((wsid), year, month, day)
) WITH CLUSTERING ORDER BY (year DESC, month DESC, day DESC);
```

Efficient Stream Computation

```
val kafkaStream = KafkaUtils.createStream[K, V, KDecoder, VDecoder]
  (ssc, kafkaParams, topicMap, StorageLevel.DISK_ONLY_2)
  .map(transform)
  .map(RawWeatherData(_))

kafkaStream.saveToCassandra(keyspace, raw_ws_data)

/** Per `wsid` and timestamp, aggregates hourly pricip by day in the stream. */
kafkaStream.map { weather =>
  (weather.wsid, weather.year, weather.month, weather.day, weather.oneHourPrecip)
}.saveToCassandra(keyspace, daily_precipitation_aggregations)
```

Gets the partition key: Data Locality
Spark C* Connector feeds this to Spark

Cassandra Counter column in our schema, no expensive `reduceByKey` needed. Simply let C* do it: not expensive and fast.

```
class TemperatureActor(sc: SparkContext, settings: WeatherSettings)
  extends AggregationActor {
  import akka.pattern.pipe

  def receive: Actor.Receive = {
    case e: GetMonthlyHiLowTemperature => highLow(e, sender)
  }

  def highLow(e: GetMonthlyHiLowTemperature, requester: ActorRef): Unit =
    sc.cassandraTable[DailyTemperature](keyspace, daily_temperature_aggr)
      .where("wsid = ? AND year = ? AND month = ?", e.wsid, e.year, e.month)
      .collectAsync()
      .map(MonthlyTemperature(_, e.wsid, e.year, e.month)) pipeTo requester

}
```

C* data is automatically sorted by most recent - due to our data model.

Additional Spark or collection sort not needed.

Recent Additions

DATASTAX

- Better write performance
  - Token-aware writes
  - Smarter batching
  - Write throttling
- Better read performance
  - spanBy / spanByKey - timeseries data, better than groupBy
  - Pushing down ORDER BY / LIMIT / COUNT to Cassandra

Recent Additions

DATASTAX

- Scala 2.11 support and cross build
- Mapping user-defined classes to Cassandra UDTs
- Namespace support for multiple Cassandra clusters
- Transform RDD of PrimaryKeys into a CassandraRDD of another Table
- Spark SQL Integration Improvements
    - More predicate pushdowns
    - Support for joins across multiple clusters
- Metrics

Support For

DATASTAX

- Push-down **joins** between generic RDD's and C* Tables
- Partitioning any RDD to the same strategy as a C* Table

What This Means

- Join between an RDD and C* Table without doing a full table scan
- When performed between two C* Tables which share the same partition key this **will not** require movement of data between machines
- Uses the source RDD's partitioning and placement for data locality

Roadmap

DATASTAX

- CassandraInputDStream - stream from a cassandra table (soon)
- Performance Improvements
  - Token-aware data repartitioning
  - Token-aware saving
  - Wide-row support - no costly **groupBy** call
- Python API support
- Official Scala Driver for Cassandra
- Java 8 API

@helenaedelson

github.com/helena

slideshare.net/helenaedelson

DATASTAX

©2014 DataStax Confidential. Do not distribute without consent.

Resources

DATASTAX

Spark Cassandra Connector

github.com/datastax/spark-cassandra-connector

github.com/killrweather/killrweather

groups.google.com/a/lists.datastax.com/forum/#!forum/spark-connector-user

Apache Spark spark.apache.org

Apache Cassandra cassandra.apache.org

Apache Kafka kafka.apache.org

Akka akka.io

Thanks for listening!

SEPTEMBER 22 - 24, 2015 | Santa Clara Convention Center, Santa Clara, CA

[Cassandra Summit]