



Lambda Architecture with Spark Streaming, Kafka, Cassandra, Akka, Scala

Helena Edelson
[@helenaedelson](https://twitter.com/helenaedelson)



Who Is This Person?

- 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

```
/** 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?

```
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

What Delivering Meaning

Why Spark, Kafka, Cassandra & Akka

How Composable Pipelines

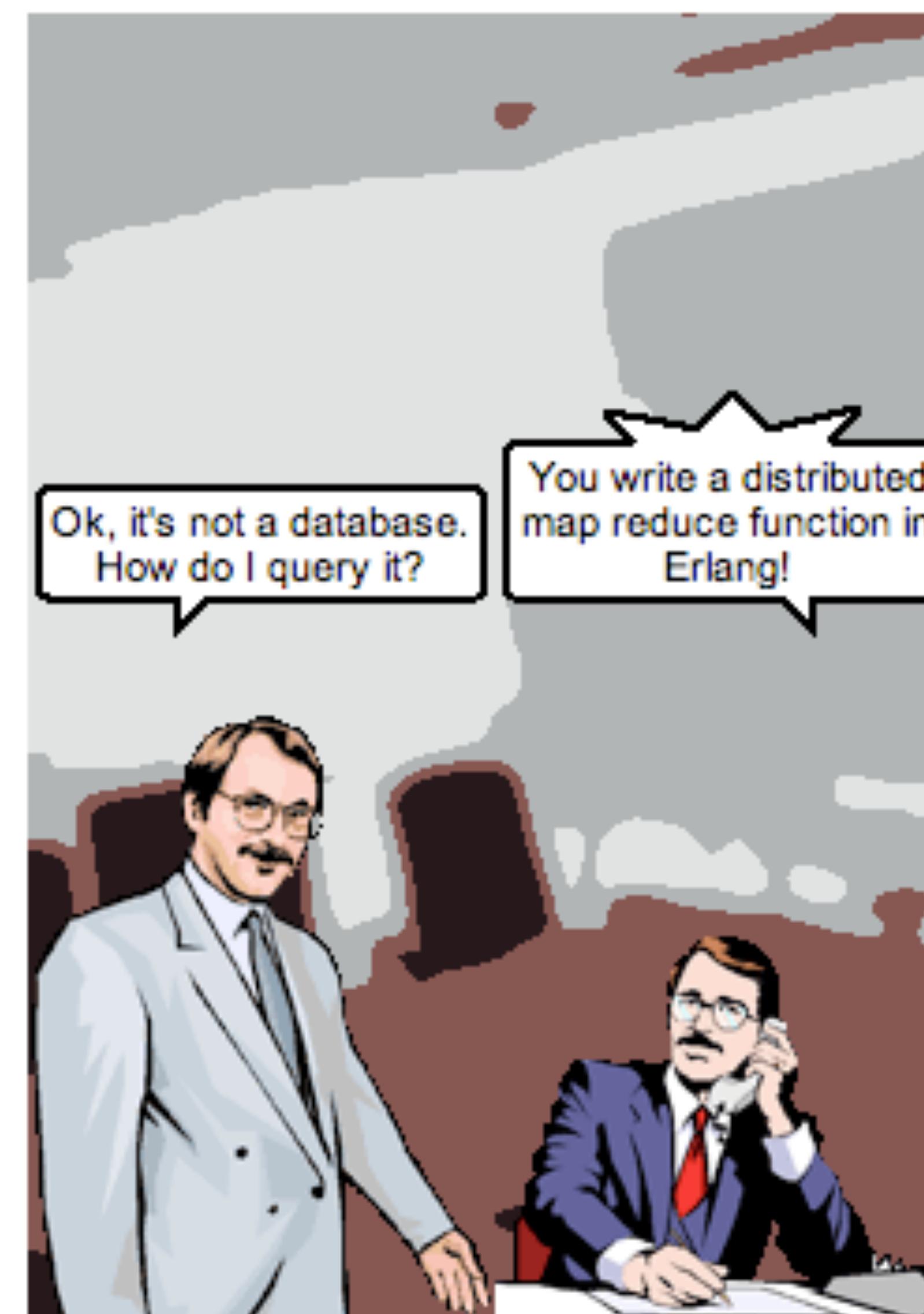
App Robust Implementation

Sending Data Between Systems Is *~~Difficult~~ Risky*

DATASTAX

Fault-tolerance

by @jrecursive



Strategies

- 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

The screenshot shows the official Apache Spark website at spark.apache.org. The header includes the Spark logo and the tagline "Lightning-fast cluster computing". Navigation links for Download, Related Projects, Documentation, Community, and FAQ are available. A sidebar on the right lists "Latest News" items such as "Spark 1.0.2 released (Aug 05, 2014)" and "Spark Summit 2014 videos posted (Jul 18, 2014)". A "Related Projects" section lists Spark SQL, Spark Streaming, MLlib (machine learning), and GraphX (graph). A "Download Spark" button is also present.

Apache Spark™ is a fast and general engine for large-scale data processing.

Speed
Run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.
Spark has an advanced DAG execution engine that supports cyclic data flow and in-memory computing.

Ease of Use
Write applications quickly in Java, Scala or Python.
Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it interactively from the Scala and Python shells.

```
file = spark.textFile("hdfs://...")  
file.flatMap(lambda line:  
    line.split()  
        .map(lambda word: (word, 1))  
        .reduceByKey(lambda a, b:  
            a+b)
```

Generality
Combine SQL, streaming, and complex analytics.
Spark powers a stack of high-level tools including [Spark SQL](#), [MLlib](#) for machine learning, [GraphX](#), and [Spark Streaming](#). You can combine these frameworks seamlessly in the same application.

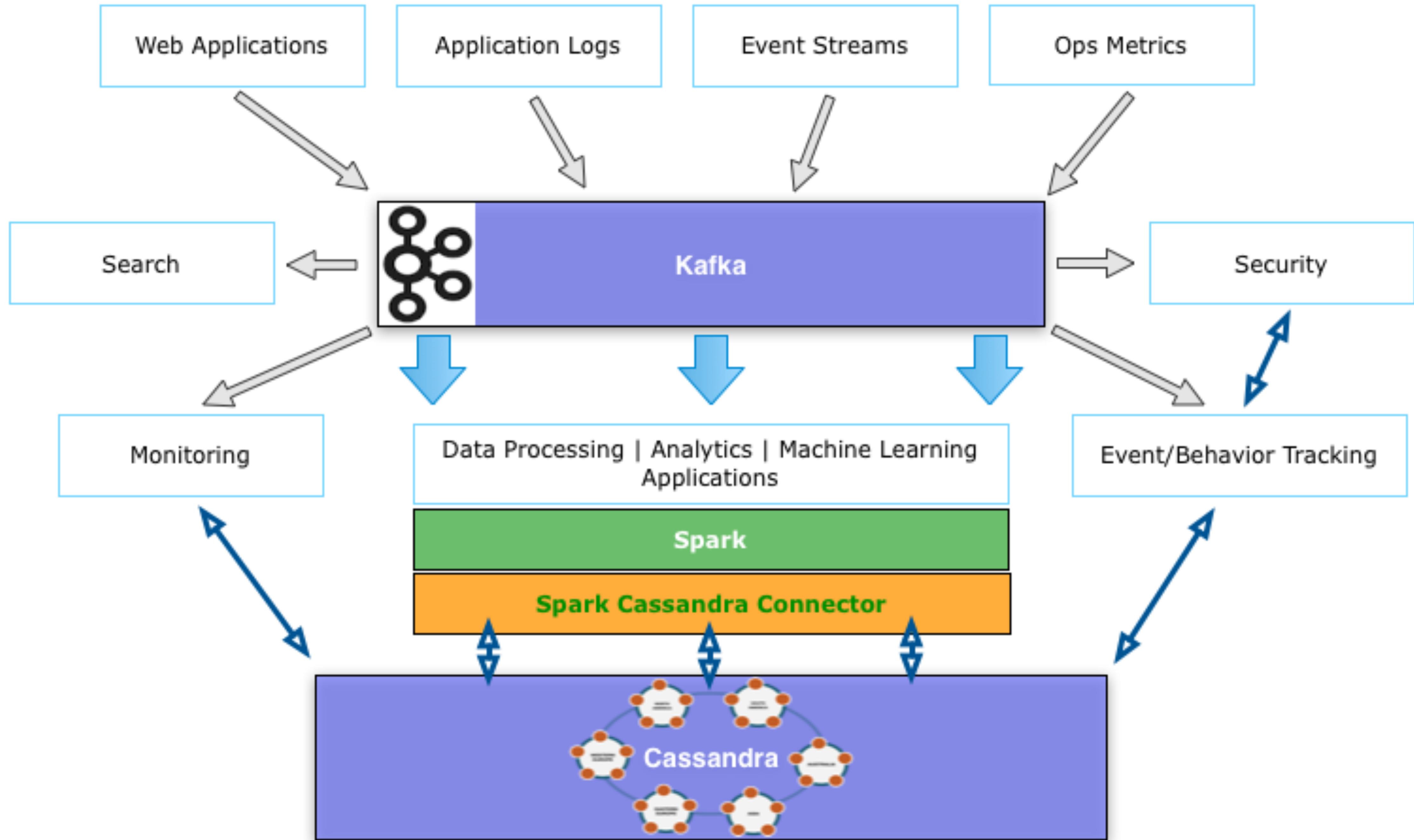
Integrated with Hadoop
Spark can run on Hadoop 2's YARN cluster manager, and can read any existing

The logo for Apache Hadoop, featuring a yellow bulldog icon followed by the word "hadoop" in a blue, lowercase, sans-serif font.



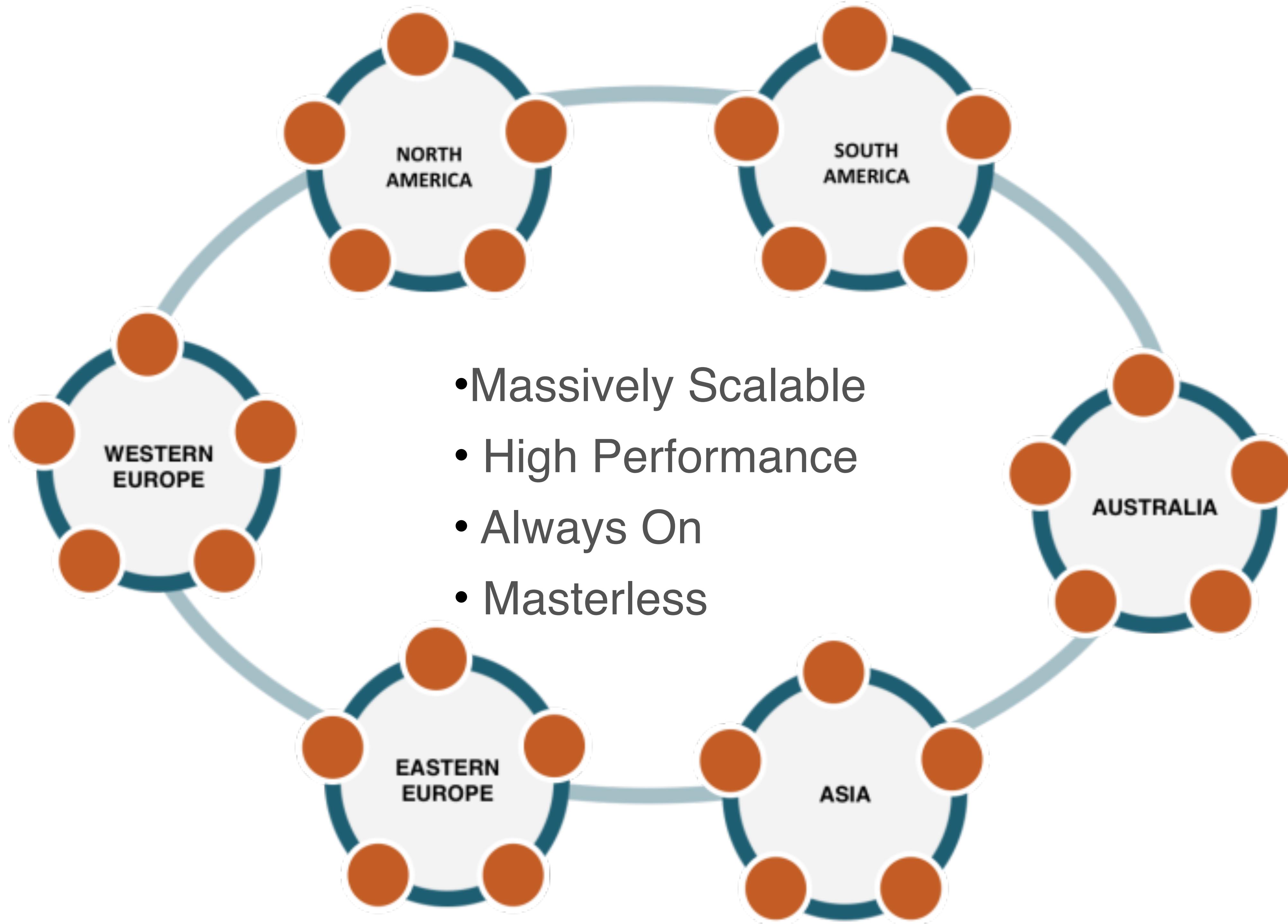
kafka

- 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



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







akka

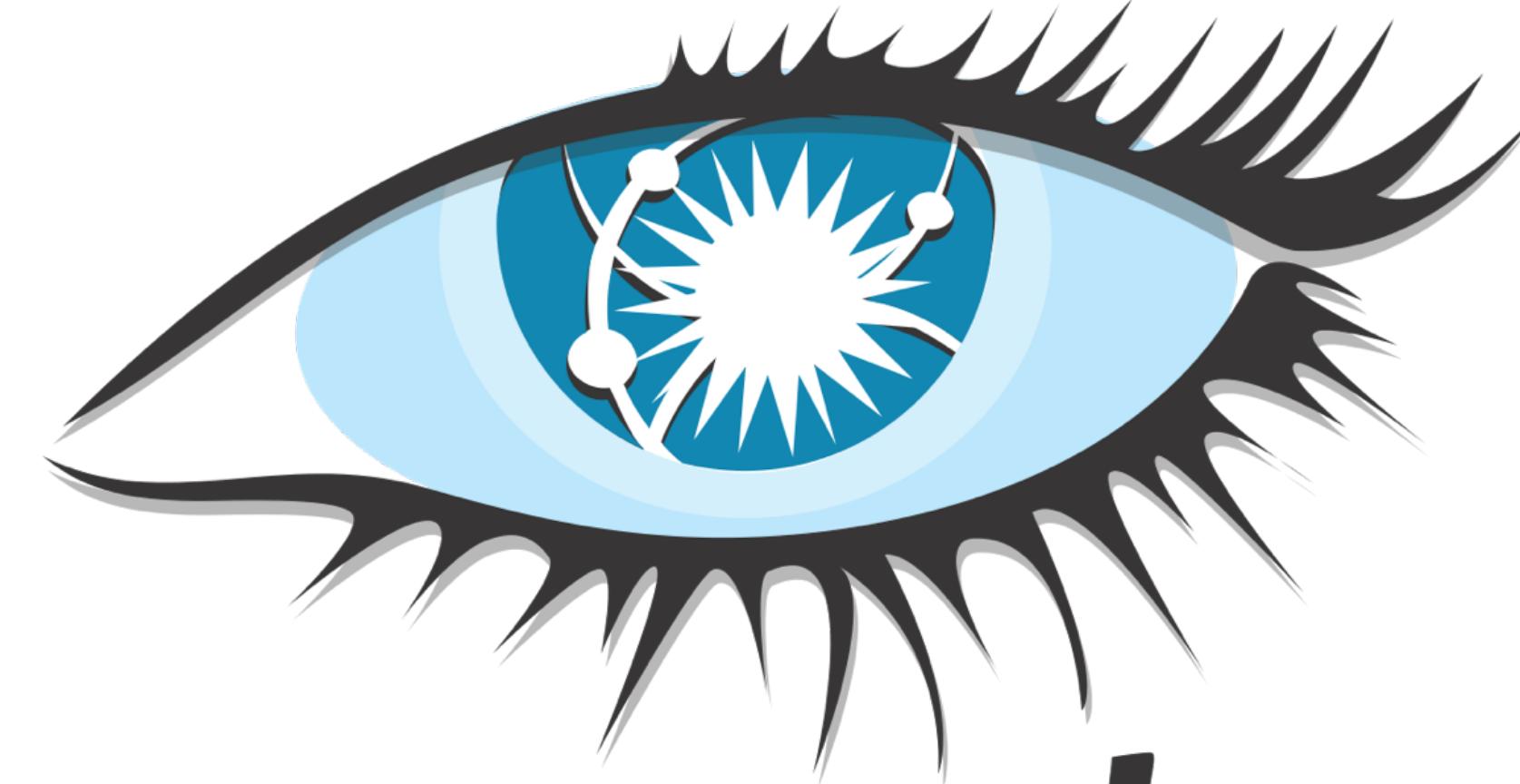


Spark

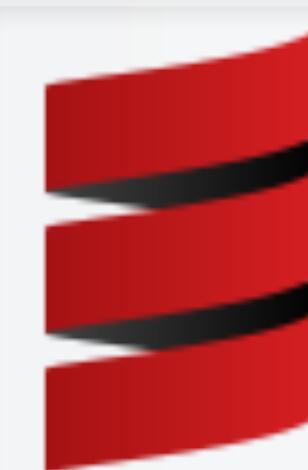


kafka

*Spark Cassandra
Connector*



cassandra



Scala



- 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 Core



Akka Cluster



Apache Kafka Cluster



Apache Cassandra Cluster



Alpha / Pre-alpha

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



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
	10 Merged Pull Requests
	154 Proposed Pull Requests
	0 Active Issues
	0 Closed Issues
	0 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**.

The chart displays the number of pull requests proposed by 154 different people. The y-axis represents the count of pull requests, ranging from 0 to 150. The x-axis shows 154 individual contributors, each represented by a small profile picture and a corresponding orange bar indicating their contribution count.

154 Pull requests proposed by 82 people

Apache Spark - Easy to Use API

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



<code>def aggregate[U](zeroValue: U)(seqOp: (U, T) → U, combOp: (U, U) → U): U</code> Aggregate the elements of each partition, and then the results for all the partitions.	<code>def glom(): RDD[Array[T]]</code> Return an RDD created by coalescing all elements within each partition.	<code>def min()(implicit ord: Ordering[T]): T</code> Returns the min of this RDD as defined by the implicit Ordering[T].
<code>def cache(): RDD.this.type</code> Persist this RDD with the default storage level (MEMORY_ONLY).	<code>def groupBy[K](f: (T) → K, p: Partitioner)(implicit kt: KeyTypeTag[K]): RDD[K, T]</code> Return an RDD of grouped items.	<code>var name: String</code> A friendly name for this RDD
<code>def cartesian[U](other: RDD[U])(implicit arg0: ClassTag[U]): RDD[(U, U)]</code> Return the Cartesian product of this RDD and another one, that is, the product of their elements.	<code>def groupBy[K](f: (T) → K, numPartitions: Int)(implicit kt: KeyTypeTag[K]): RDD[K, T]</code> Return an RDD of grouped elements.	<code>val partitioner: Option[Partitioner]</code> Optionally overridden by subclasses to specify how they are partitioned.
<code>def checkpoint(): Unit</code> Mark this RDD for checkpointing.	<code>def groupBy[K](f: (T) → K)(implicit kt: ClassTag[K]): RDD[K, T]</code> Return an RDD of grouped items.	<code>def partitions: Array[Partition]</code> Get the array of partitions of this RDD, taking into account whether the RDD is checked for partitions.
<code>def coalesce(numPartitions: Int, shuffle: Boolean = false): RDD[T]</code> Return a new RDD that is reduced into numPartitions partitions.	<code>val id: Int</code> A unique ID for this RDD (within its SparkContext).	<code>def persist(): RDD.this.type</code> Persist this RDD with the default storage level (MEMORY_ONLY).
<code>def collect[U](f: PartialFunction[T, U])(implicit arg0: ClassTag[U]): RDD[U]</code> Return an RDD that contains all matching values by applying f.	<code>def intersection(other: RDD[T], numPartitions: Int): RDD[T]</code> Return the intersection of this RDD and another one.	<code>def persist(newLevel: StorageLevel): RDD.this.type</code> Set this RDD's storage level to persist its values across operations after the first time it is computed.
<code>def collect(): Array[T]</code> Return an array that contains all of the elements in this RDD.	<code>def intersection(other: RDD[T], partitioner: Partitioner): RDD[T]</code> Return the intersection of this RDD and another one.	<code>def pipe(command: Seq[String], env: Map[String, String] = Map()): RDD[String]</code> Return an RDD created by piping elements to a forked external process.
<code>def context: SparkContext</code> The <code>org.apache.spark.SparkContext</code> that this RDD was created on.	<code>def intersection(other: RDD[T]): RDD[T]</code> Return the intersection of this RDD and another one.	<code>def pipe(command: String, env: Map[String, String]): RDD[String]</code> Return an RDD created by piping elements to a forked external process.
<code>def count(): Long</code> Return the number of elements in the RDD.	<code>def isCheckpointed: Boolean</code> Return whether this RDD has been checkpointed or not.	<code>def pipe(command: String): RDD[String]</code> Return an RDD created by piping elements to a forked external process.
<code>def countApprox(timeout: Long, confidence: Double = 0.95): Long</code> Approximate version of count() that returns a potentially incomplete result.	<code>def iterator(split: Partition, context: TaskContext): Iterator[T]</code> Internal method to this RDD; will read from cache if applicable, or otherwise read from disk.	<code>def preferredLocations(split: Partition): Seq[String]</code> Get the preferred locations of a partition (as hostnames), taking into account whether the partitioner specifies them.
<code>def countApproxDistinct(relativeSD: Double = 0.05): Long</code> Return approximate number of distinct elements in the RDD.	<code>def keyBy[K](f: (T) → K): RDD[(K, T)]</code> Creates tuples of the elements in this RDD by applying f.	<code>def randomSplit(weights: Array[Double], seed: Long = Utils.random.nextInt()): RDD[Dataset[_]]</code> Randomly splits this RDD with the provided weights.
<code>def countApproxDistinct(p: Int, sp: Int): Long</code> Return approximate number of distinct elements in the RDD.	<code>def map[U](f: (T) → U)(implicit arg0: ClassTag[U]): RDD[U]</code> Return a new RDD by applying a function to all elements of this RDD.	<code>def reduce(f: (T, T) → T): T</code> Reduces the elements of this RDD using the specified commutative and associative reduce function.
<code>def countByValue()(implicit ord: Ordering[T] = null): Map[T, Long]</code> Return the count of each unique value in this RDD as a map of (value, count).	<code>def mapPartitions[U](f: (Iterator[T]) → Iterator[U], p: Partitioner): RDD[U]</code> Return a new RDD by applying a function to each partition of this RDD.	<code>def repartition(numPartitions: Int)(implicit ord: Ordering[T] = null): RDD[T]</code> Return a new RDD that has exactly numPartitions partitions.
<code>def countByValueApprox(timeout: Long, confidence: Double): Map[T, Long]</code> Approximate version of countByValue().	<code>def mapPartitionsWithContext[U](f: (TaskContext, Iterator[T]) → Iterator[U], p: Partitioner): RDD[U]</code> Return a new RDD by applying a function to each partition of this RDD.	<code>def sample(withReplacement: Boolean, fraction: Double, seed: Long): RDD[T]</code> Return a sampled subset of this RDD.
<code>def dependencies: Seq[Dependency[_]]</code> Get the list of dependencies of this RDD, taking into account whether they are resolved.	<code>def mapPartitionsWithIndex[U](f: (Int, Iterator[T]) → Iterator[U], p: Partitioner): RDD[U]</code> Return a new RDD by applying a function to each partition of this RDD.	<code>def saveAsObjectFile(path: String): Unit</code> Save this RDD as a SequenceFile of serialized objects.
<code>def distinct(): RDD[T]</code> Return a new RDD containing the distinct elements in this RDD.	<code>def max()(implicit ord: Ordering[T]): T</code> Returns the max of this RDD as defined by the implicit Ordering[T].	<code>def saveAsTextFile(path: String, codec: Class[_ <: CompressionCodec]): Unit</code> Save this RDD as a compressed text file, using string representations of elements.

Spark Basic Word Count

```
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

```
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

```
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 }
```

Transformation

Action



When Batch Is Not Enough

Spark Streaming

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

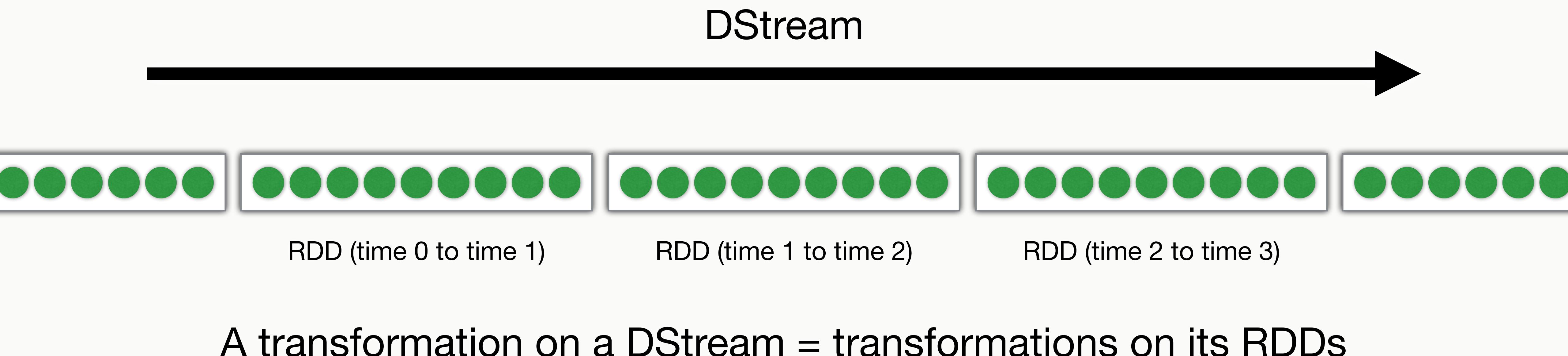
Spark Streaming Setup

```
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)

Continuous stream of micro batches

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

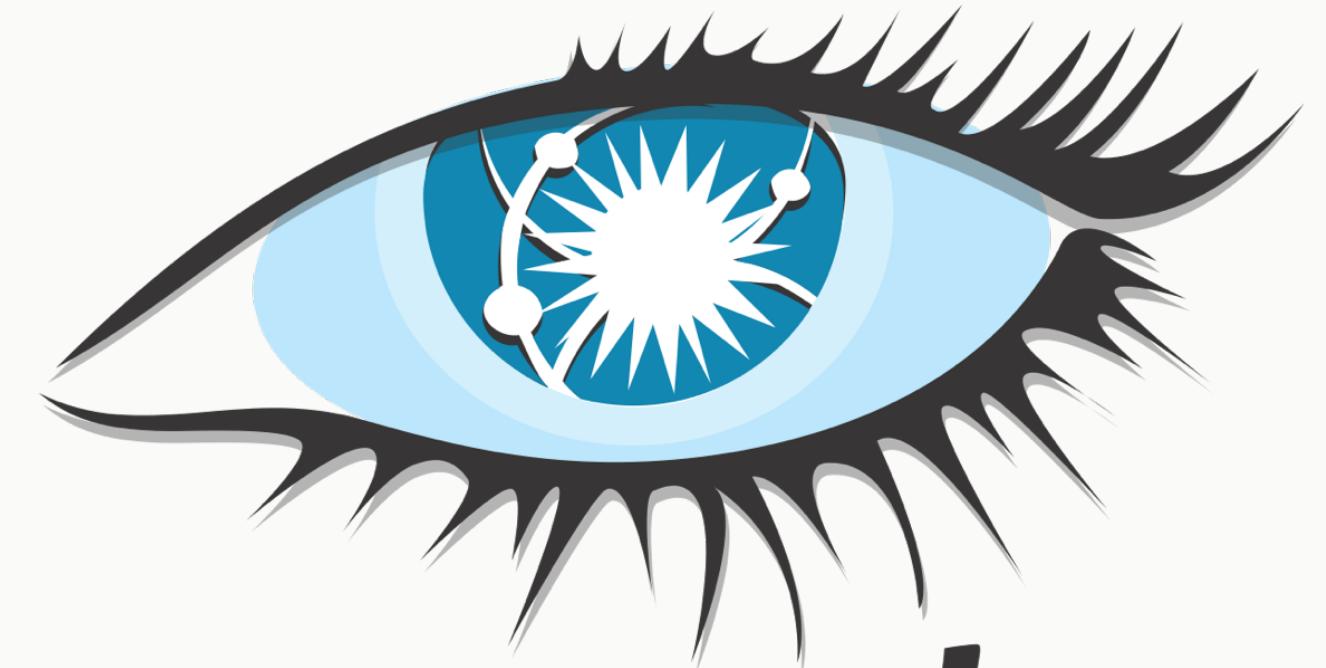


A transformation on a DStream = transformations on its RDDs

Spark Streaming External Source/Sink



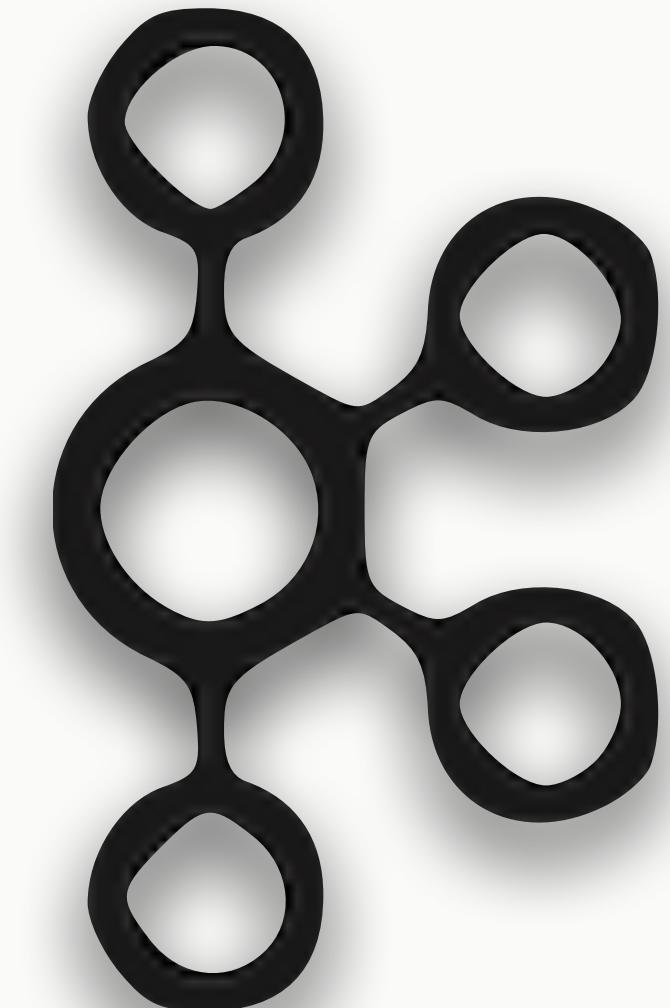
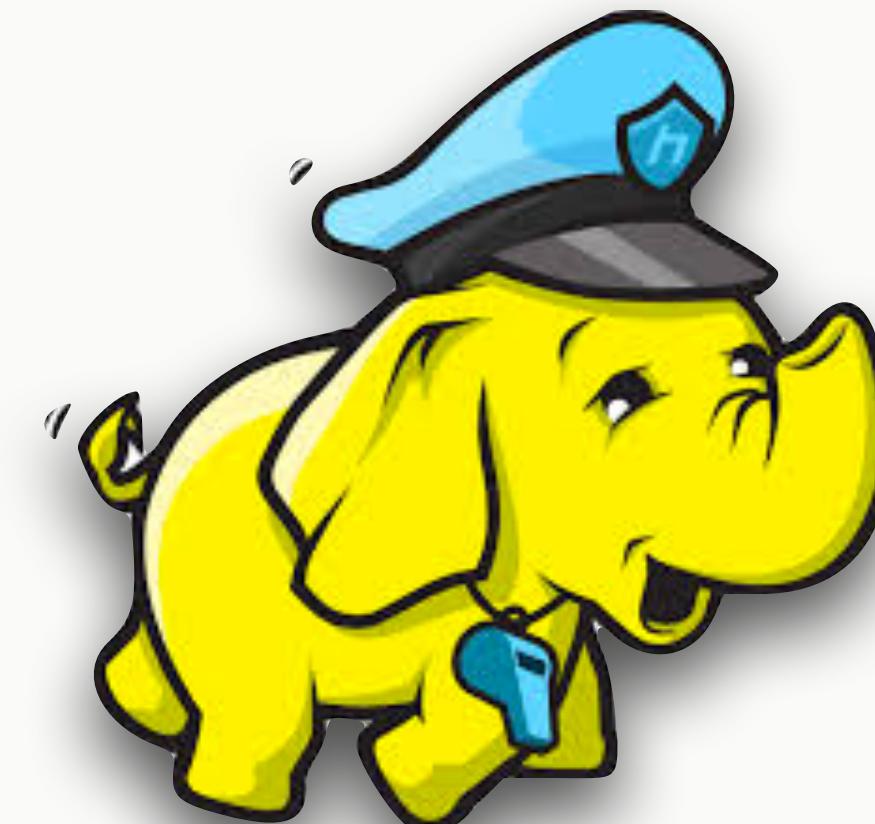
ØMQ



cassandra



akka



ReceiverInputDStreams

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: FileInputStream



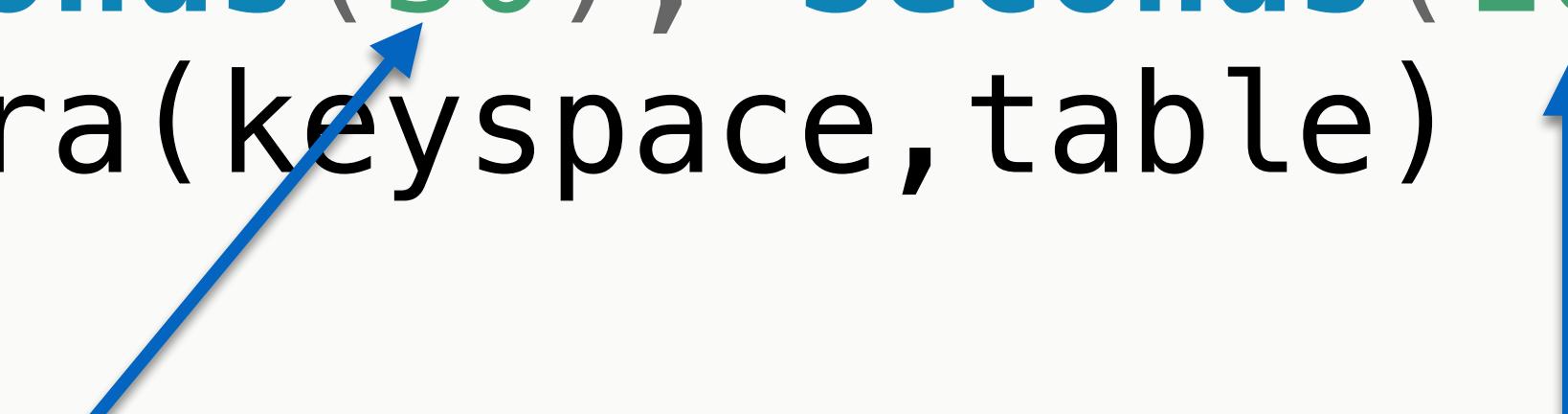
```
// Creates new DStreams
ssc.textFileStream("s3n://raw_data_bucket/")
  .flatMap(_.split("\\s+"))
  .map(_.toLowerCase, 1))
  .countByValue()
  .saveAsObjectFile("s3n://analytics_bucket/")
```

Streaming Window Operations

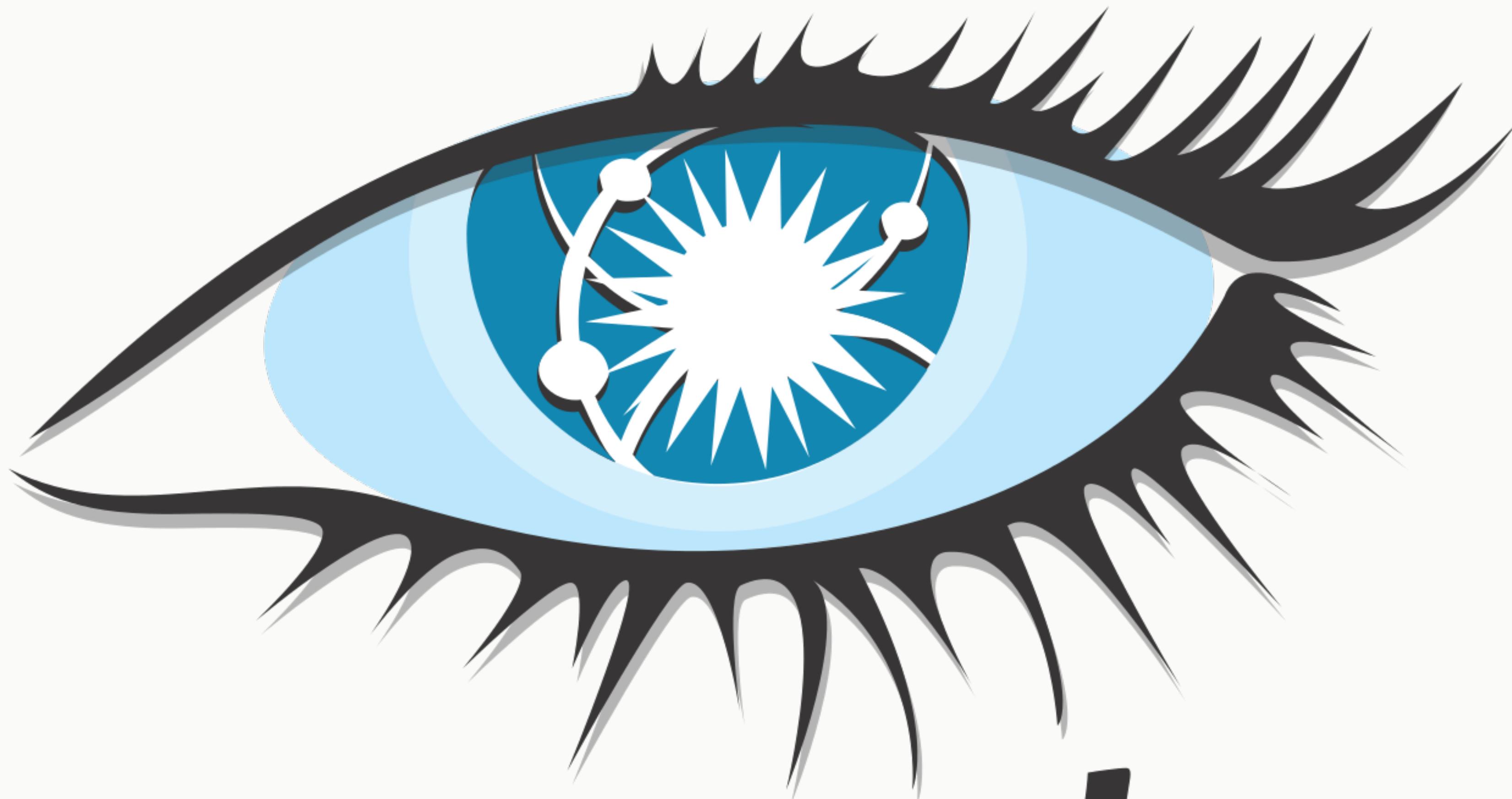
kvStream

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

Window Length:
Duration = every 10s



Sliding Interval:
Interval at which the window operation
is performed = every 10 s



cassandra

Public Cassandra Users



Apache Cassandra

- 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

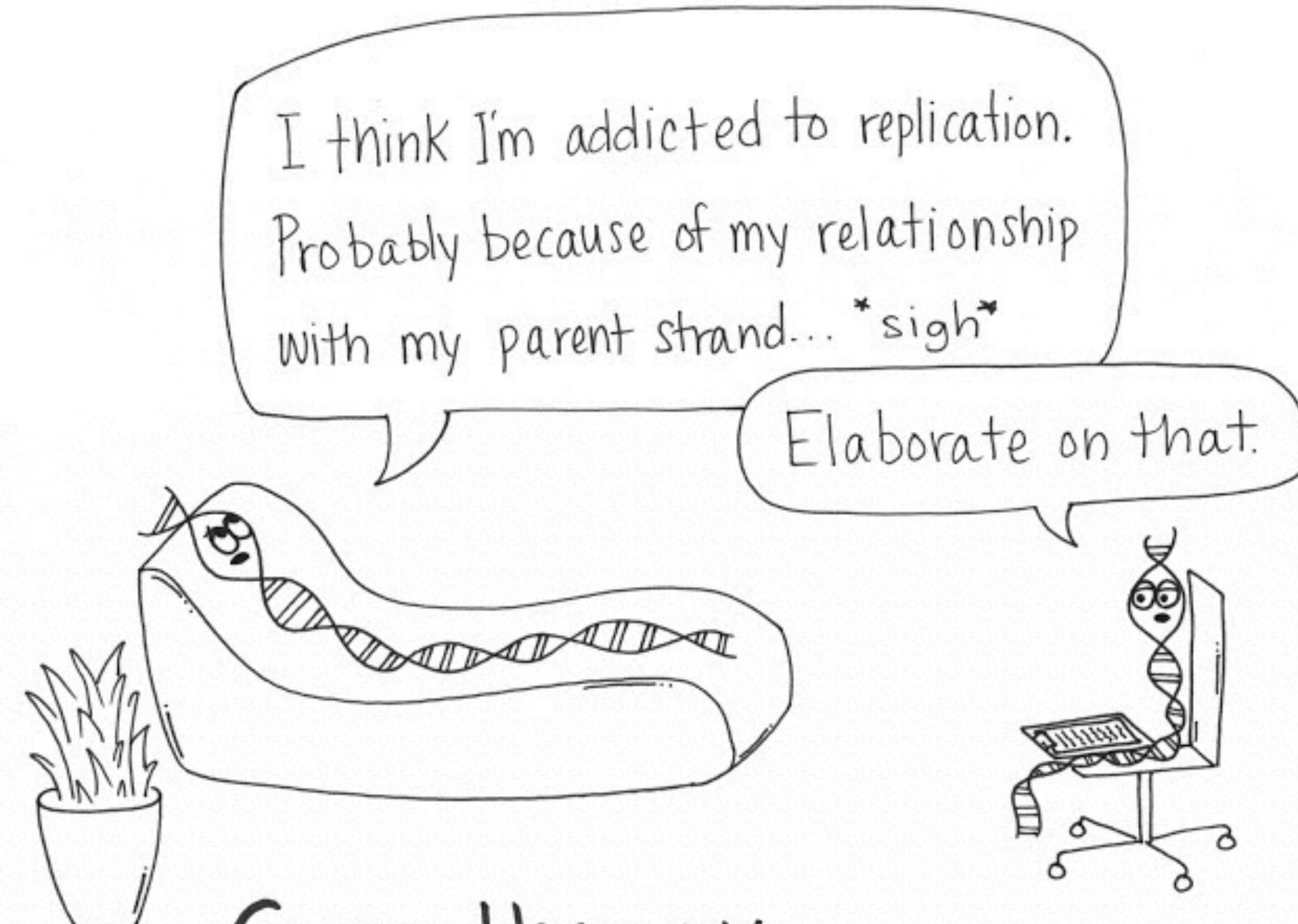
Security, Machine Learning



Physics: Astro Physics / Particle Physics..



Genetics / Biological Computations



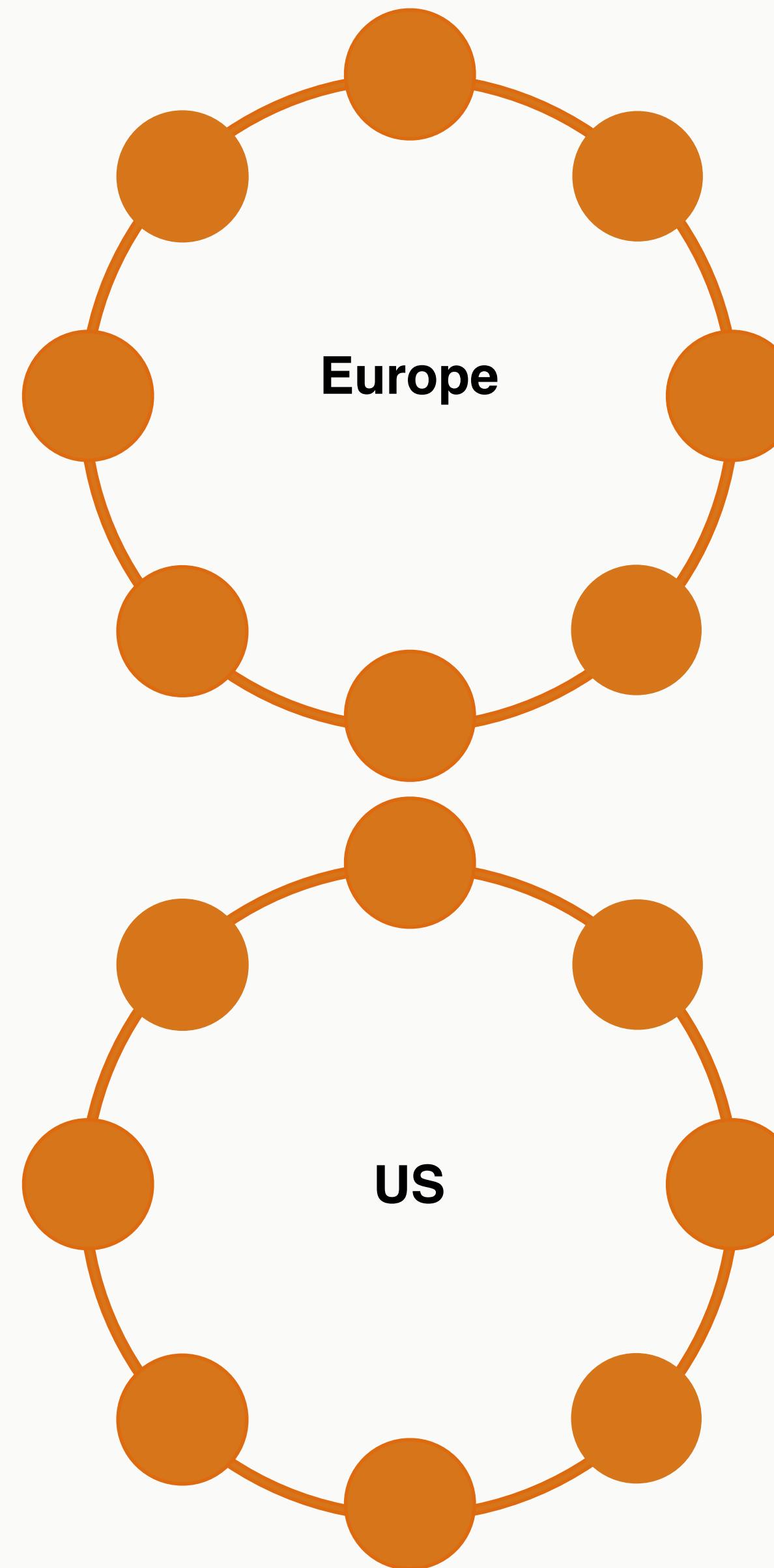
Gene therapy.

Beatrice the Biologist

IoT

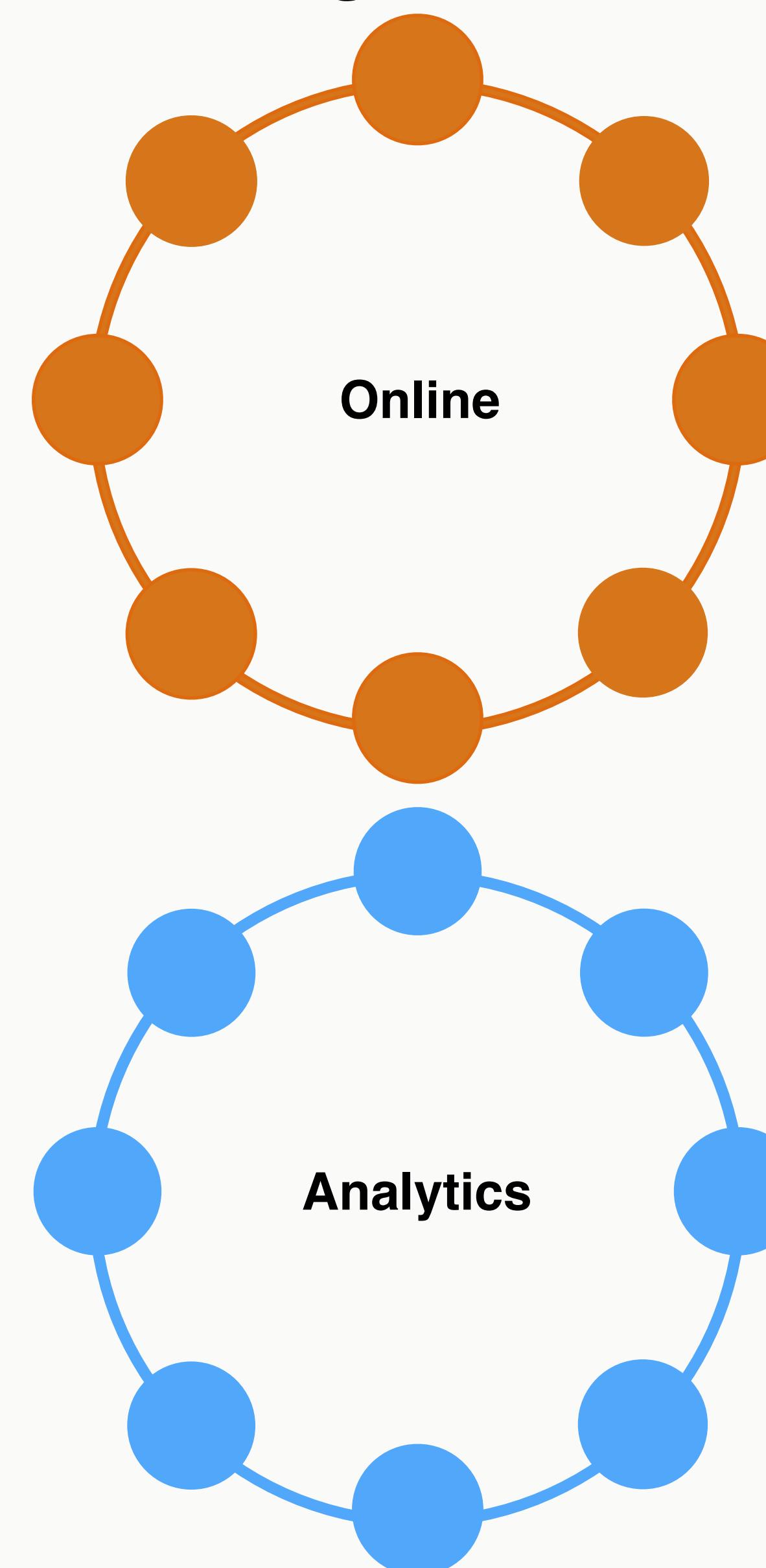


Cassandra



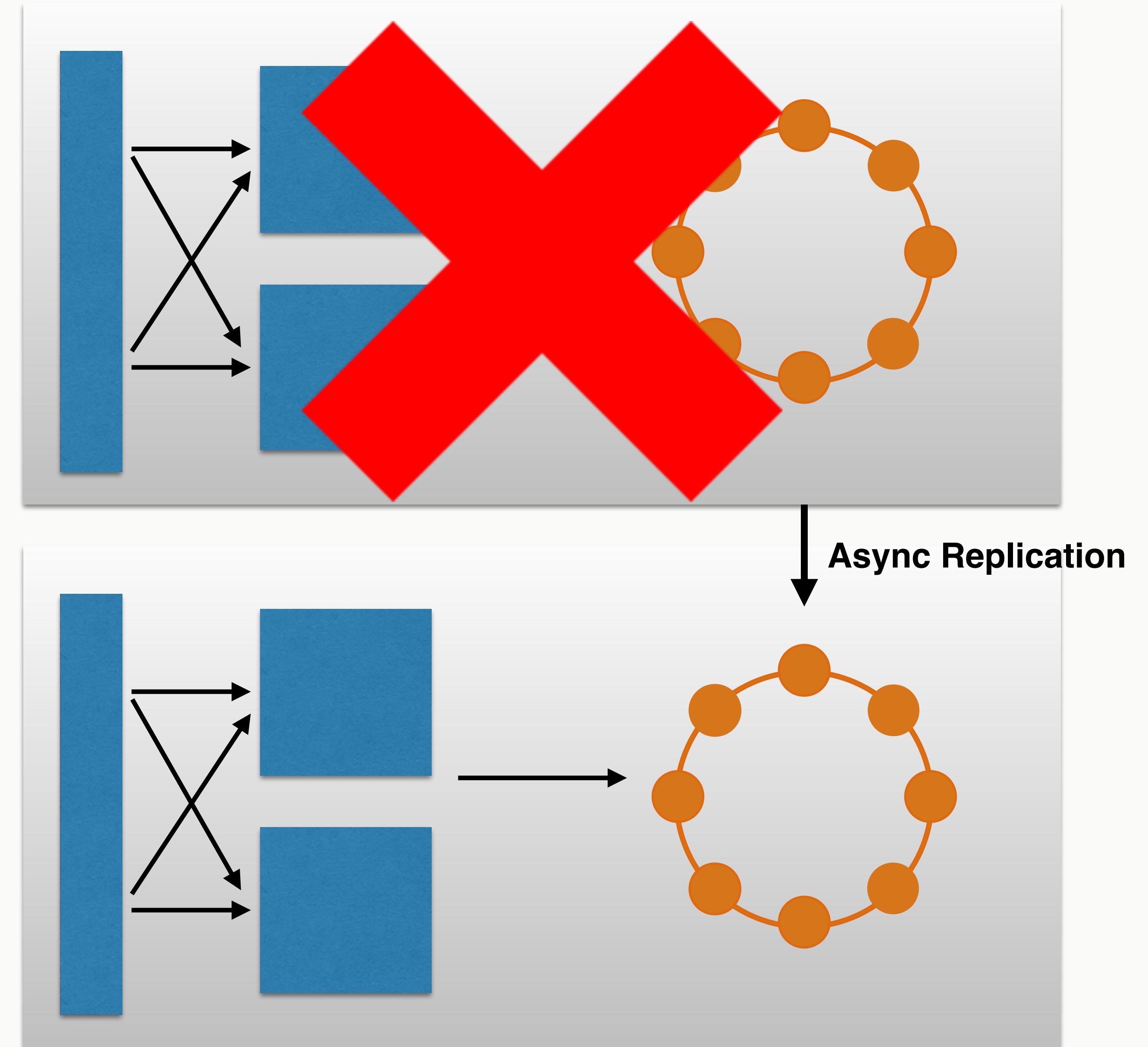
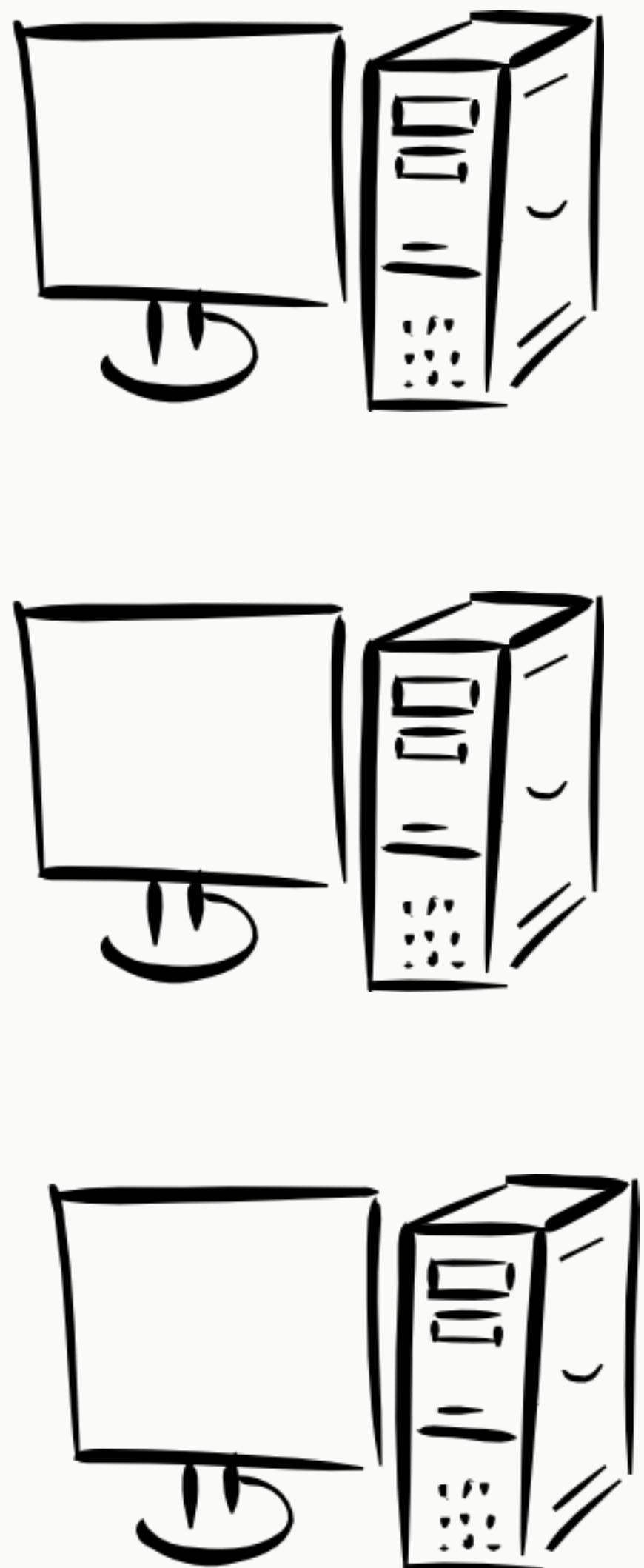
- Availability Model
 - Amazon Dynamo
 - Distributed, Masterless
- Data Model
 - Google Big Table
- Network Topology Aware
- Multi-Datacenter Replication

Analytics with Spark Over Cassandra



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

Handling Failure



What Happens When Nodes Come Back?

- Hinted handoff to the rescue
- Coordinators keep writes for downed nodes for a configurable amount of time



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



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

```
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;
```

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

Timeseries

```
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 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

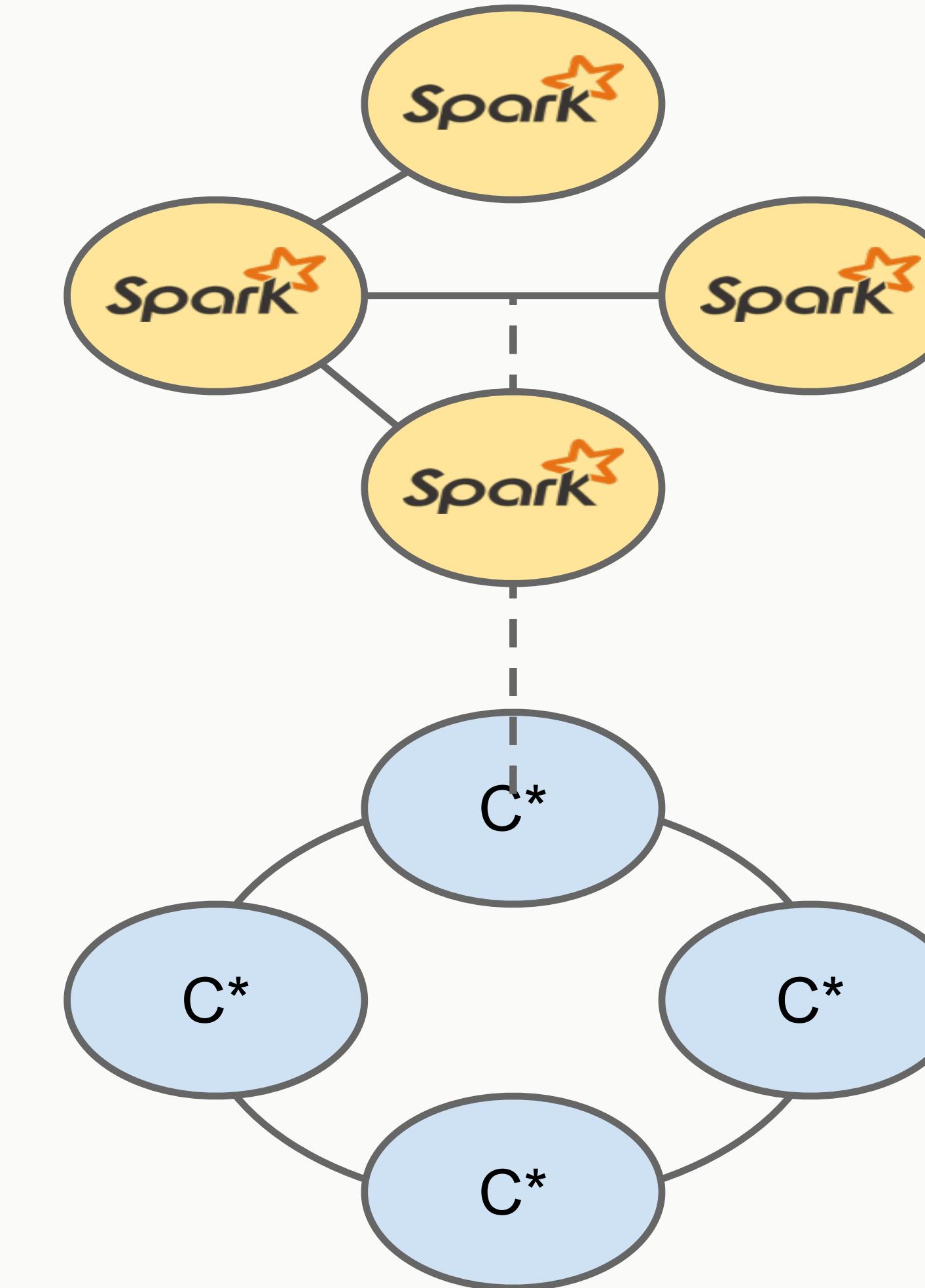
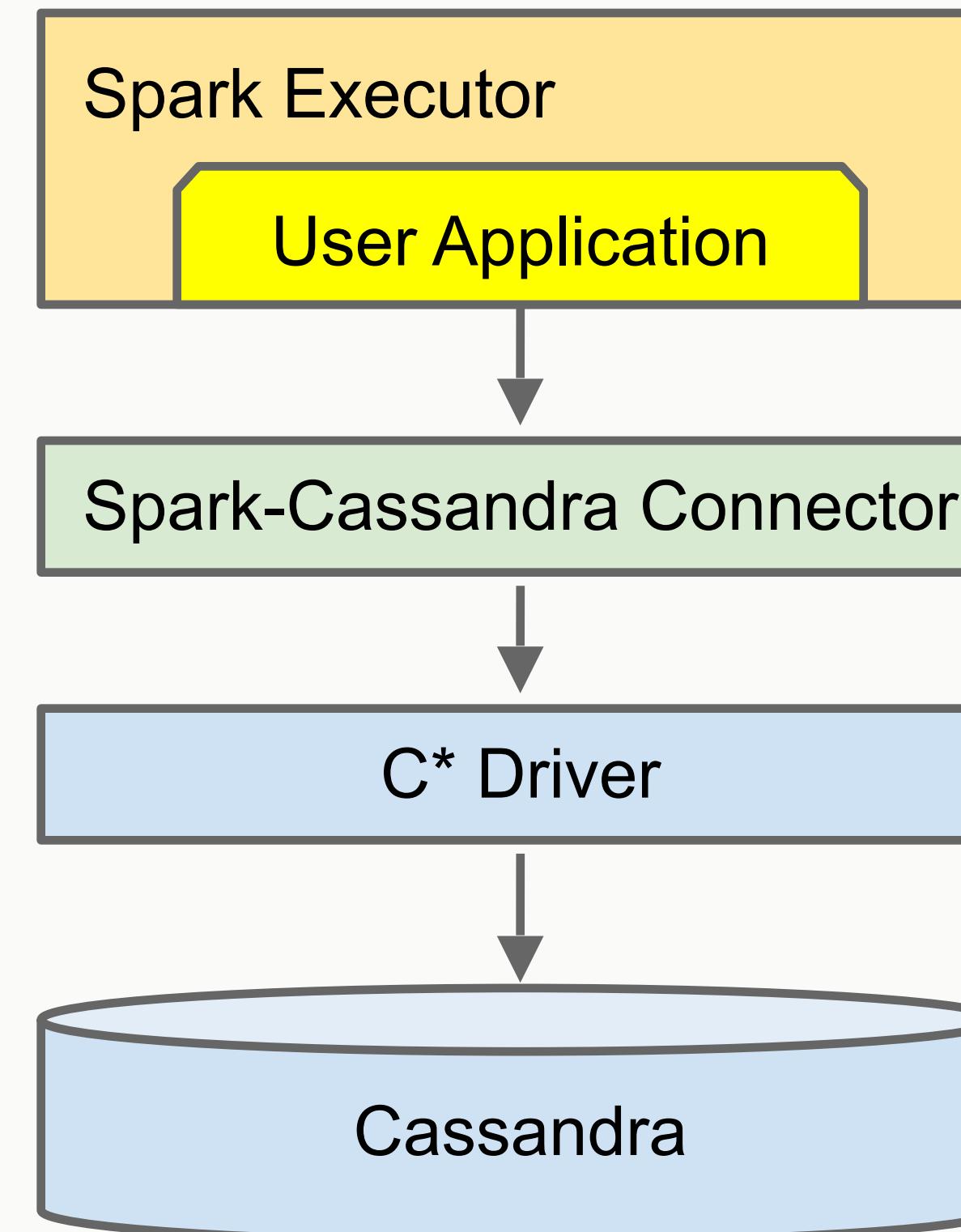


Spark Cassandra Connector

[**https://github.com/datastax/spark-cassandra-connector**](https://github.com/datastax/spark-cassandra-connector)

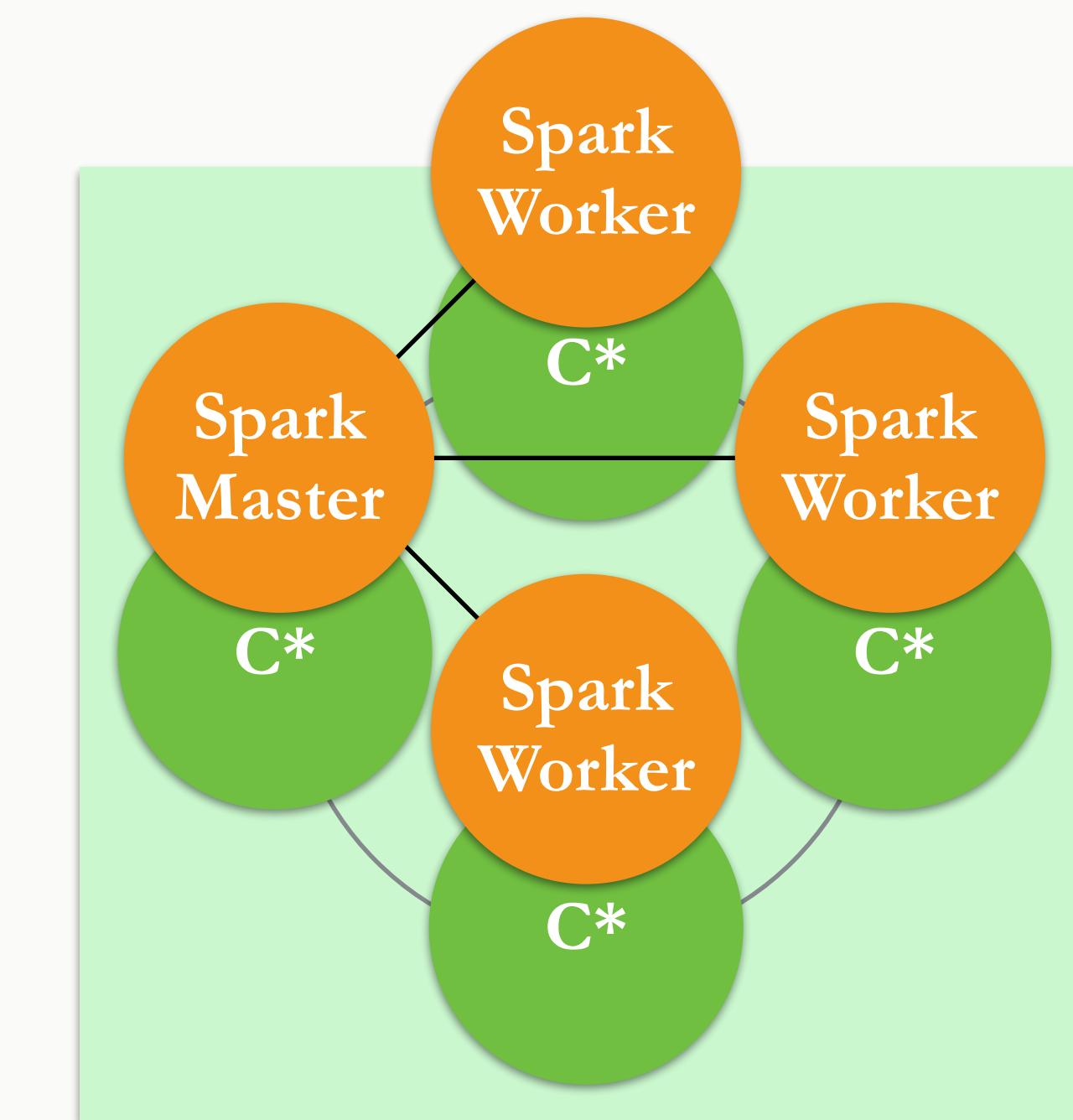
- Write data from Spark to Cassandra
- Read data from Cassandra to Spark
- Data Locality for Speed
- Easy, and often implicit, type conversions
- Server-Side Filtering - SELECT, WHERE, etc.
- Natural Timeseries Integration
- Implemented in Scala

Spark Cassandra Connector



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

SparkContext

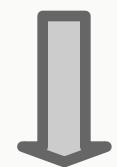
```
import com.datastax.spark.connector._
```

StreamingContext

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

Write from Spark to Cassandra

SparkContext



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

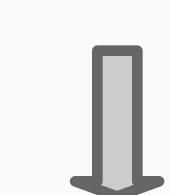
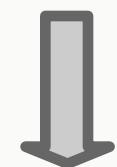
Keyspace



Table

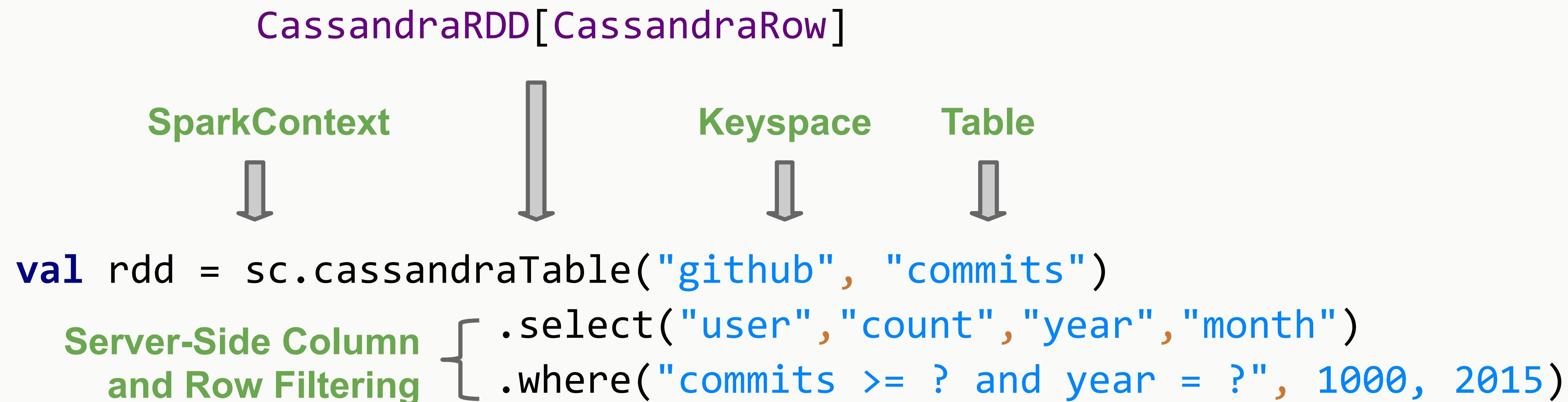


Spark RDD JOIN with NOSQL!

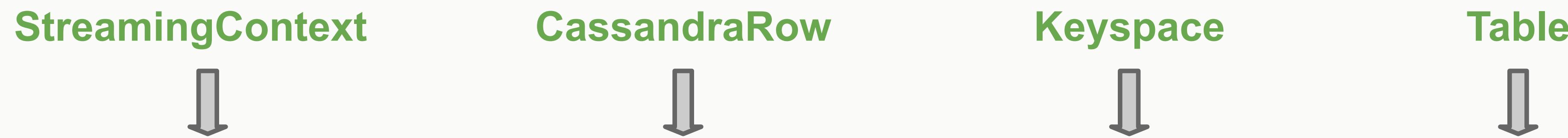


```
predictionsRdd.join(music).saveToCassandra("music", "predictions")
```

Read From C* to Spark



Rows: Custom Objects



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

Rows

```
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

```
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

```
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



```
{  
  "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

DATASTAX

Spark SQL with Cassandra

```
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)
```

Cross Keyspaces (Databases)

```
val schemaRdd1 = cassandraSqlContext.sql("""
    INSERT INTO keyspace2.table3
        SELECT table2.a, table2.b, table2.c
        FROM keyspace1.table2 as table2""")
```

```
val schemaRdd2 = cassandraSqlContext.sql("""
    SELECT table3.a, table3.b, table3.c
    FROM keyspace2.table3 as table3""")
```

```
val schemaRdd3 = cassandraSqlContext.sql("""
    SELECT test1.a, test1.b, test1.c, test2.a
    FROM database1.test1 AS test1
    JOIN database2.test2 AS test2 ON test1.a = test2.a
    AND test1.b = test2.b AND test1.c = test2.c""")
```

Spark SQL with Cassandra & JSON



```
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

```
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



```
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)
```

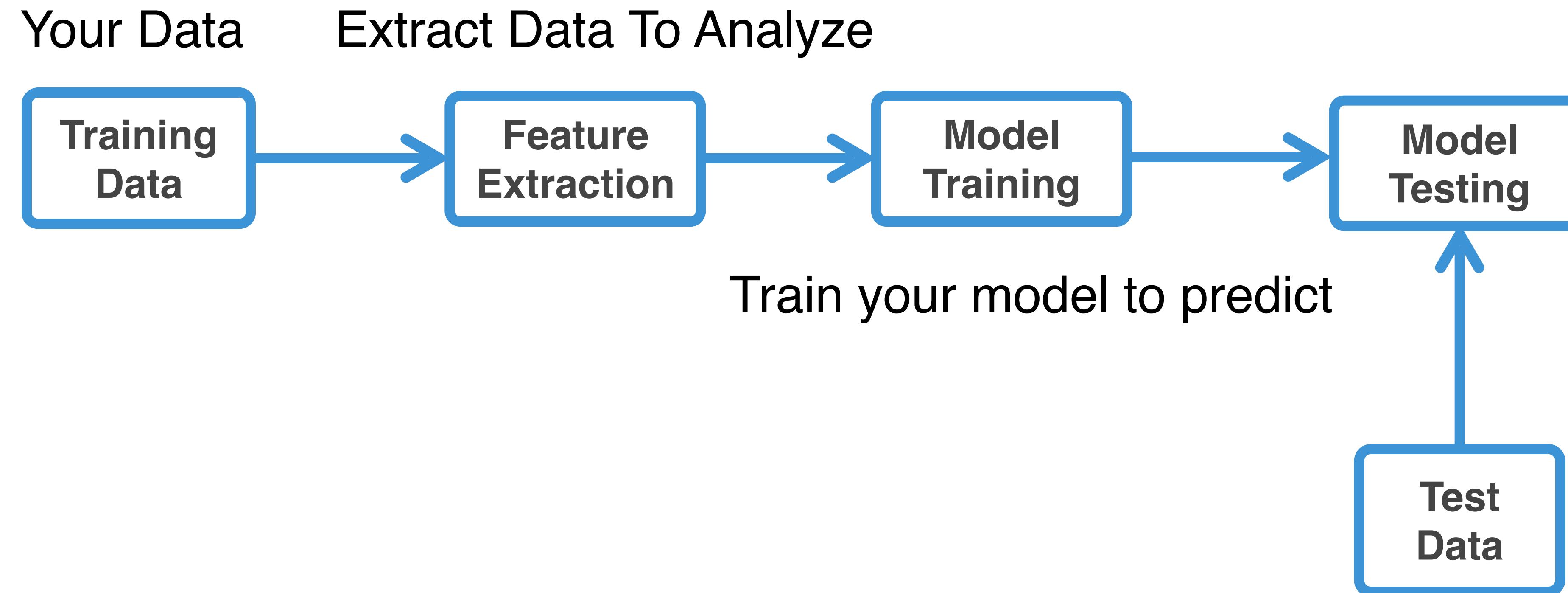
Streaming From Kafka, R/W Cassandra

```
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*

```
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(_.value).map(LabeledPoint.parse)

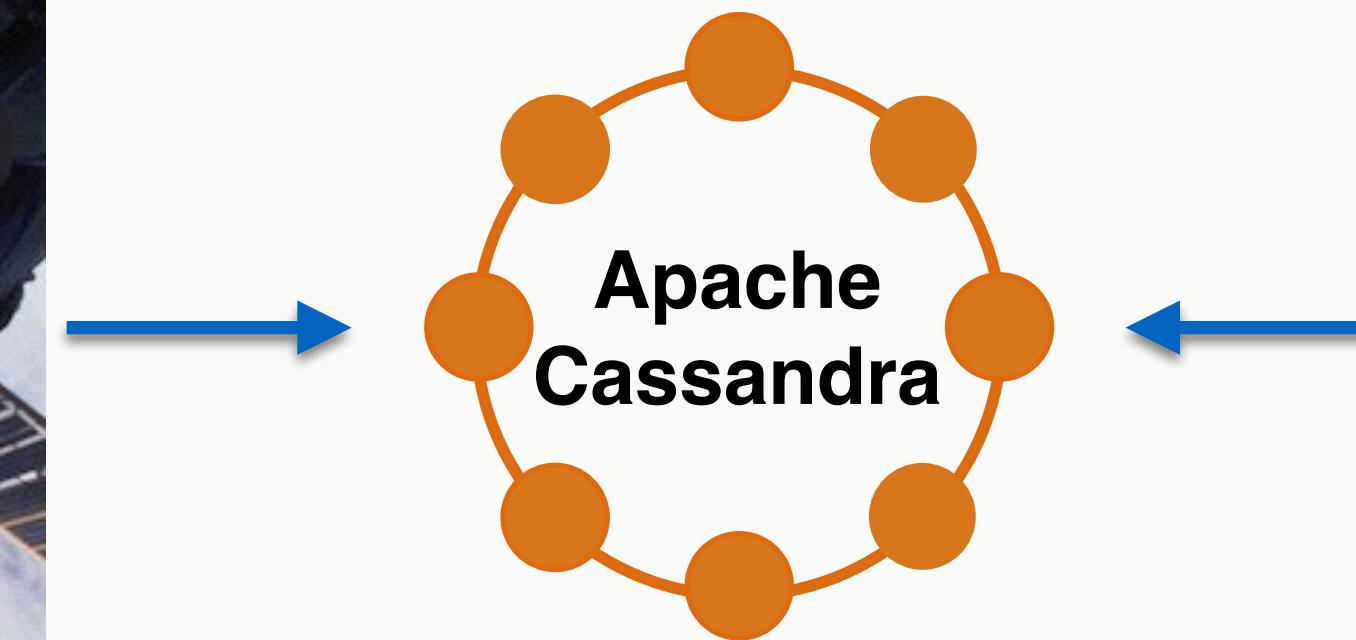
trainingStream.saveToCassandra("ml_keyspace", "raw_training_data")

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

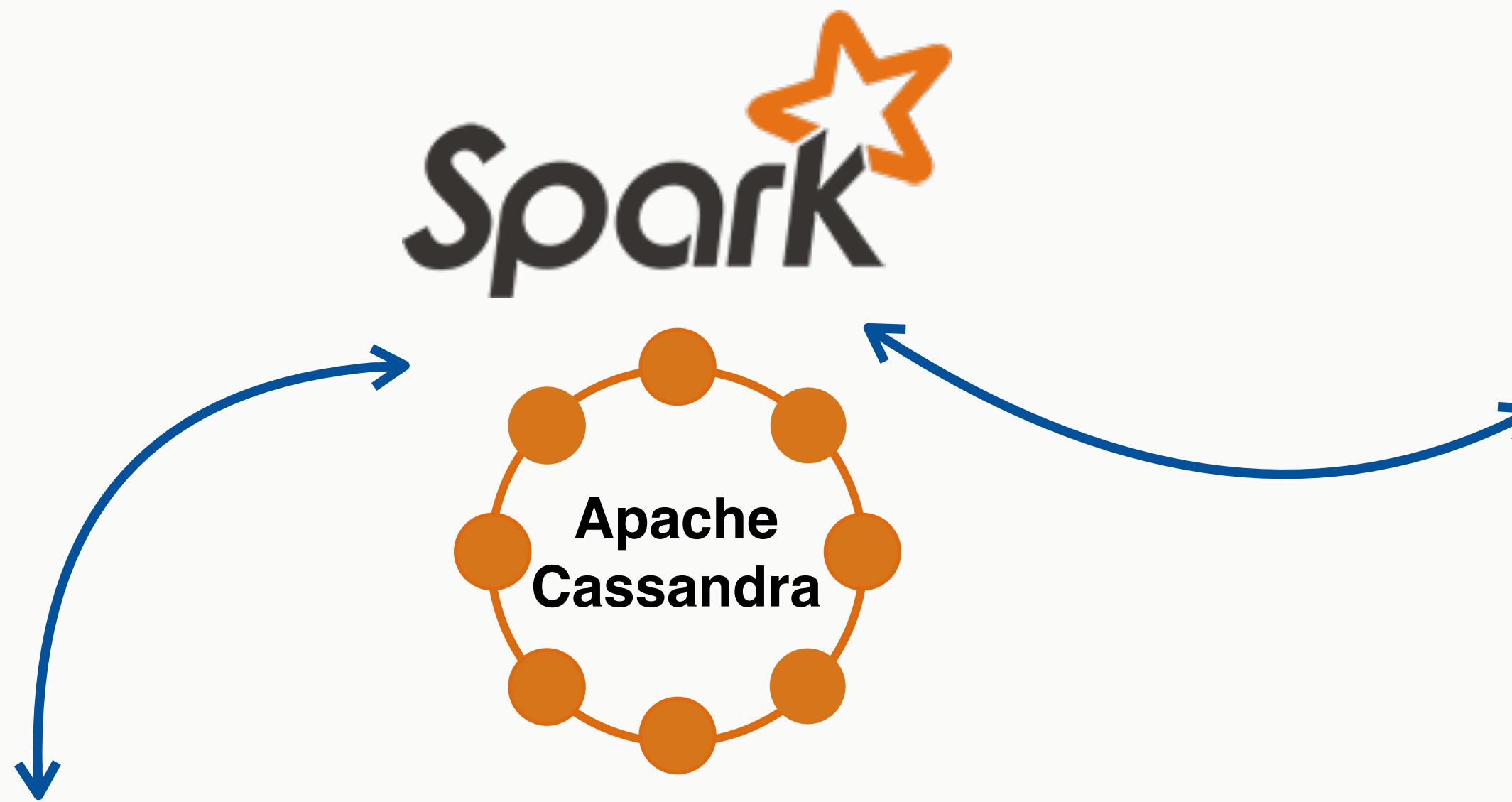
//Making predictions on testData
model
    .predictOnValues(testData.map(lp => (lp.label, lp.features)))
    .saveToCassandra("ml_keyspace", "predictions")
```

Timeseries Data Application

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



Data Analysis



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

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

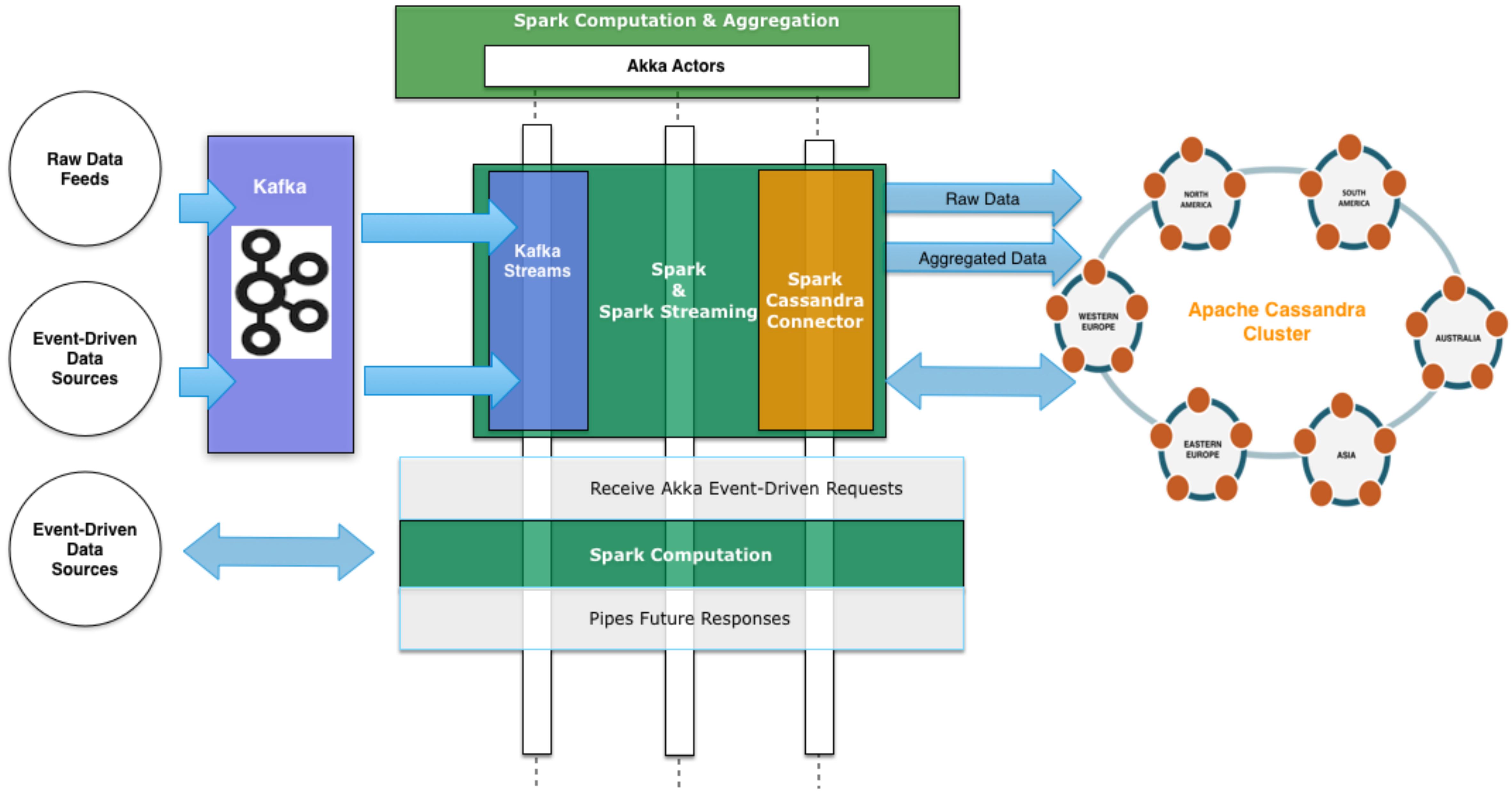
- Weather Station Id and Time are unique
- Store as many as needed

```
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);
```



Kafka Producer as Akka Actor

```
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

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

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

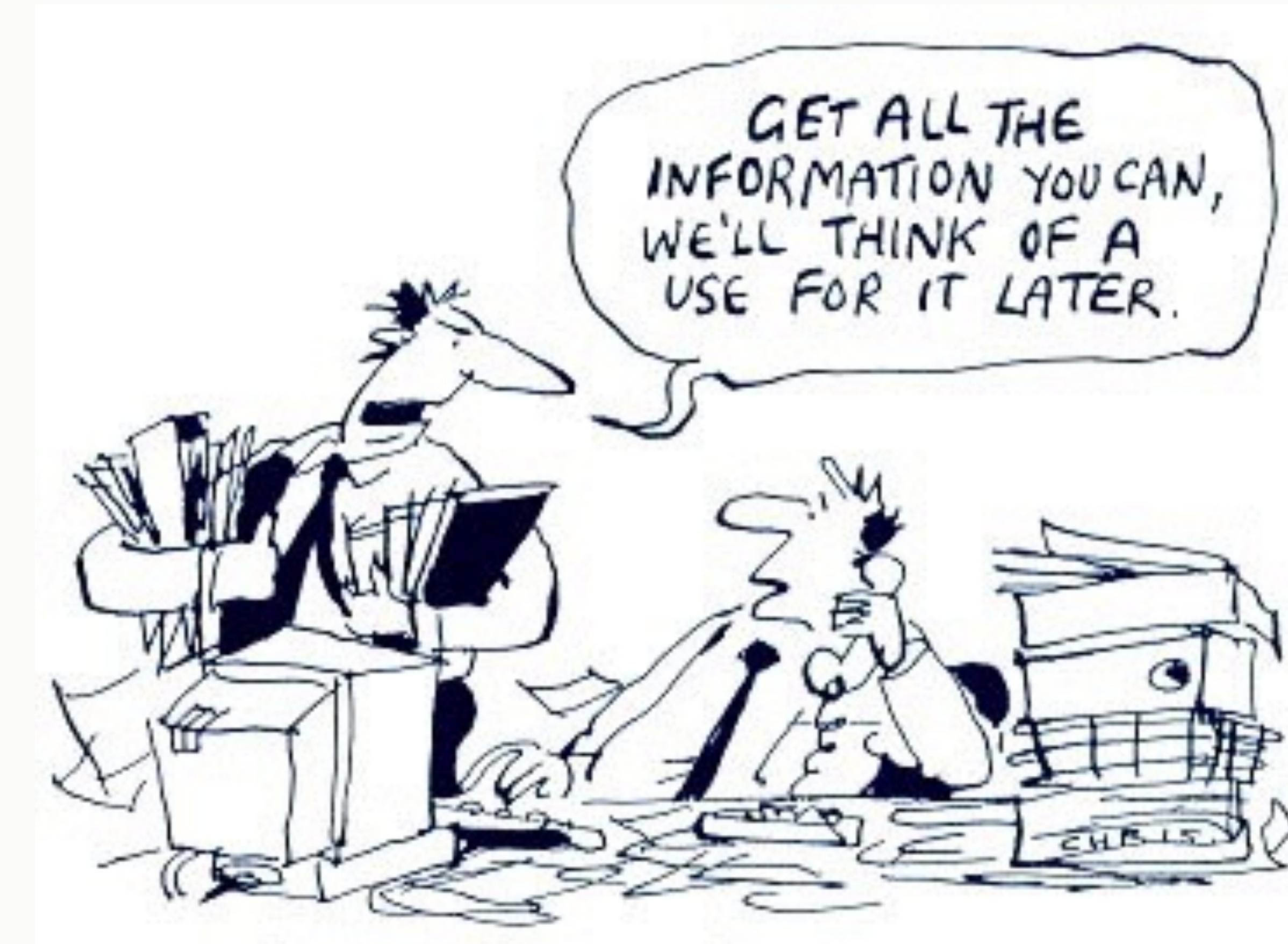
    val requestHandler: HttpRequest => HttpResponse = {
        case HttpRequest(POST, Uri.Path("/weather/v1/hourly-weather"), headers, entity, _) =>
            HttpSource(headers, entity).collect { case s: HeaderSource =>
                for(s <- source.extract)
                    kafka ! KafkaMessageEnvelope[String,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

```
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...

```
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);
```

```
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 precip 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.



This repository Search

Explore Gist Blog Help



helena +



killrweather / killrweather

Unwatch 54

Star 291

Fork 44

KillrWeather is a reference application (in progress) showing how to easily leverage and integrate Apache Spark, Apache Cassandra, and Apache Kafka for fast, streaming computations on time series data in asynchronous Akka event-driven environments. — [Edit](#)

158 commits

3 branches

0 releases

4 contributors

branch: **master** ▾**killrweather** / +

Merge branch 'master' of github.com:killrweather/killrweather

helena authored 11 days ago

latest commit 0d74fc225c



Adding much bigger data file

5 months ago



New diagram

11 days ago



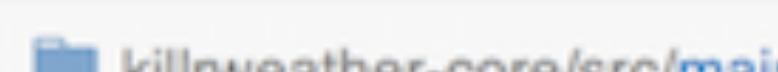
Explicitly listen on 127.0.0.1 so the default of InetAddress.getLocal...

11 days ago



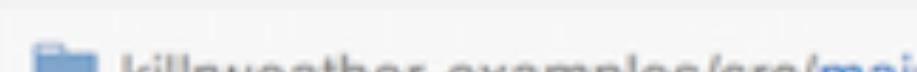
Explicit IP for the client app otherwise KillrWeatherApp can't connec...

11 days ago



Http on balancing pool router.

22 days ago



Added new logback configs and finalized log levels.

23 days ago

Code

Issues 4

Pull requests 3

Wiki

Pulse

Graphs

Settings

SSH clone URL

git@github.com:killrweath

You can clone with [HTTPS](#), [SSH](#), or [Subversion](#).



@helenaedelson



github.com/helena

[slideshare.net/helenaedelson](https://www.slideshare.net/helenaedelson)

D A T A S T A X

Resources

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