You've got the lighter, let's start a fire: Spark and Cassandra

Robert Stupp

Solutions Architect @ DataStax, Committer to Apache Cassandra

@snazy

A journey through Distributed Computing

KillrPowr Application

Show current power consumption in Germany in real time

- Power consumption of all cities on a map
- Graph of data for one city

cassandra

Spark™

kafka

One word…

THE APACHE® SOFTWARE FOUNDATION

cassandra

Spark™

kafka

Apache Cassandra

Distributed & **resilient** database

Cassandra Use-Cases

- Time Series
- Personalization
- (or a mix of both)

Cassandra?

- Joining data … hmm
- Analyzing the whole data set … hmm
- Ad-Hoc Queries (SQL) … hmm

Apache Spark

Distributed & **resilient** analytics

"Spark lets you run programs up to 100x faster in memory, or 10x faster on disk, than Hadoop."

Spark : Simplify the…

challenging and compute-intensive task of processing
high volumes of real-time or archived data,
both structured and unstructured,

seamlessly integrating relevant
complex capabilities
such as machine learning and
graph algorithms.

Source: http://www.toptal.com/spark/introduction-to-apache-spark

Spark Use Cases

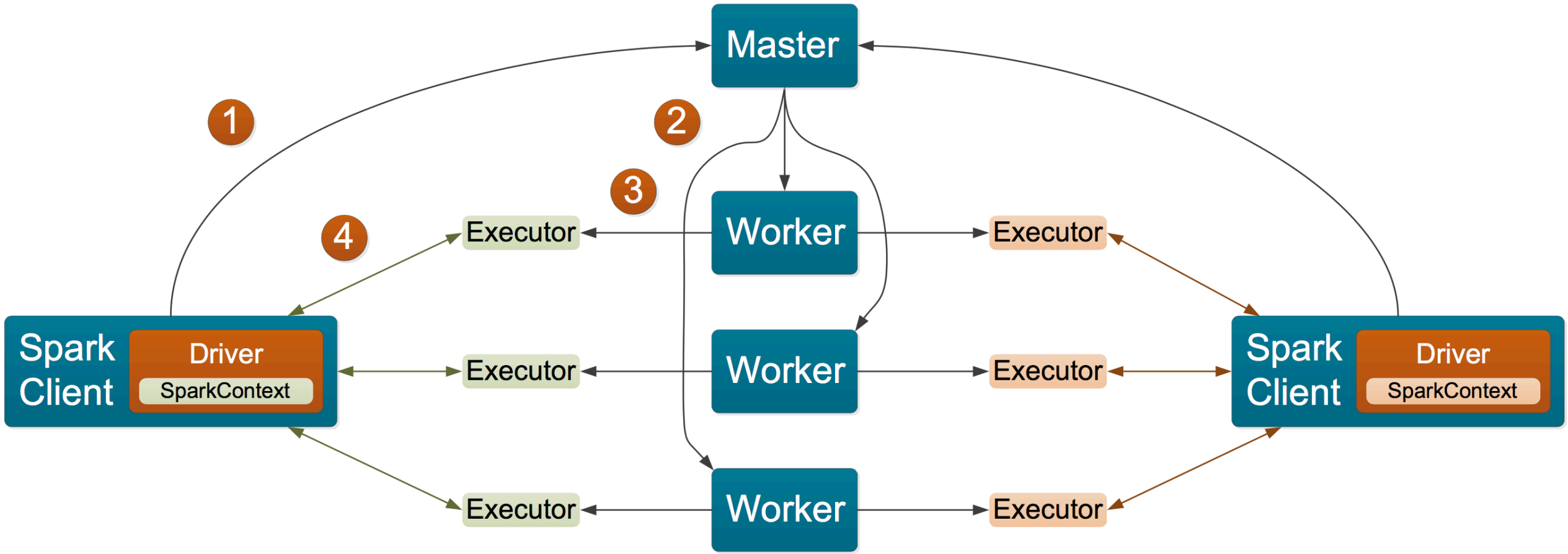
- Event detection
- Behavior / anomalies detection
- Fraud & intrusion detection
- Targeted advertising
- E-commerce transaction processing
- Recommendations

Spark Use Cases

- Data migration
- Ad hoc queries
- Business Intelligence

Spark

Spark Architecture



Spark RDDs

DATASTAX

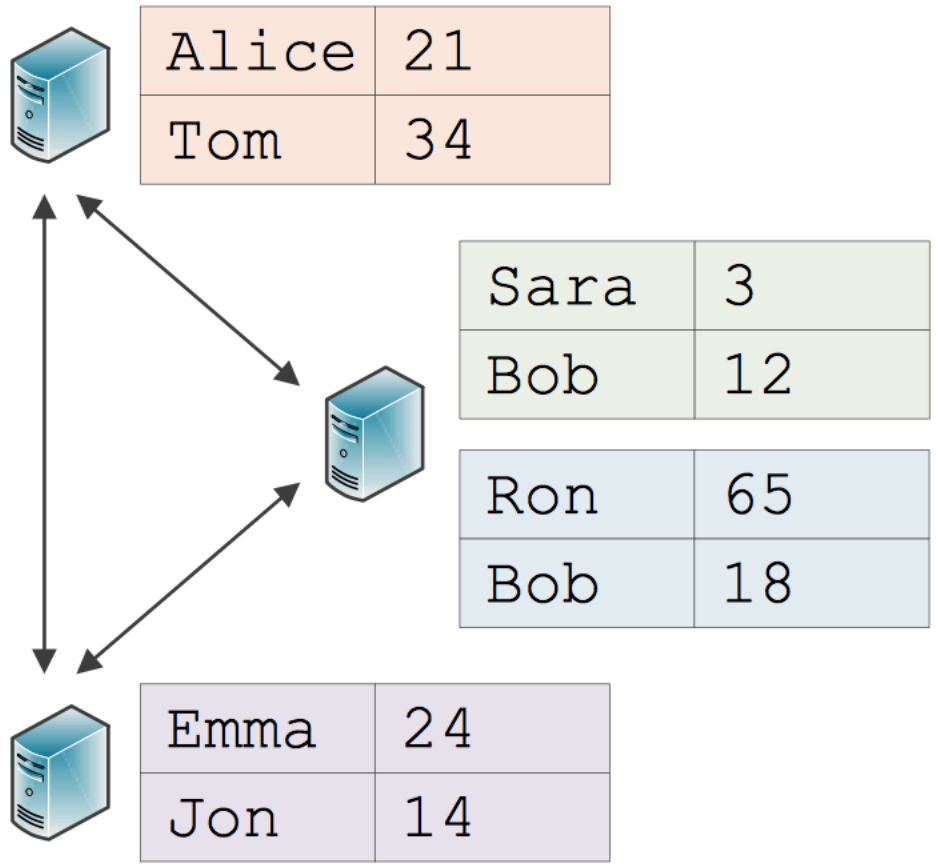
Resilient Distributed Datasets

- Fault-tolerant collection of elements
- Can be operated on in parallel
- Immutable
- In-Memory (as much as possible)

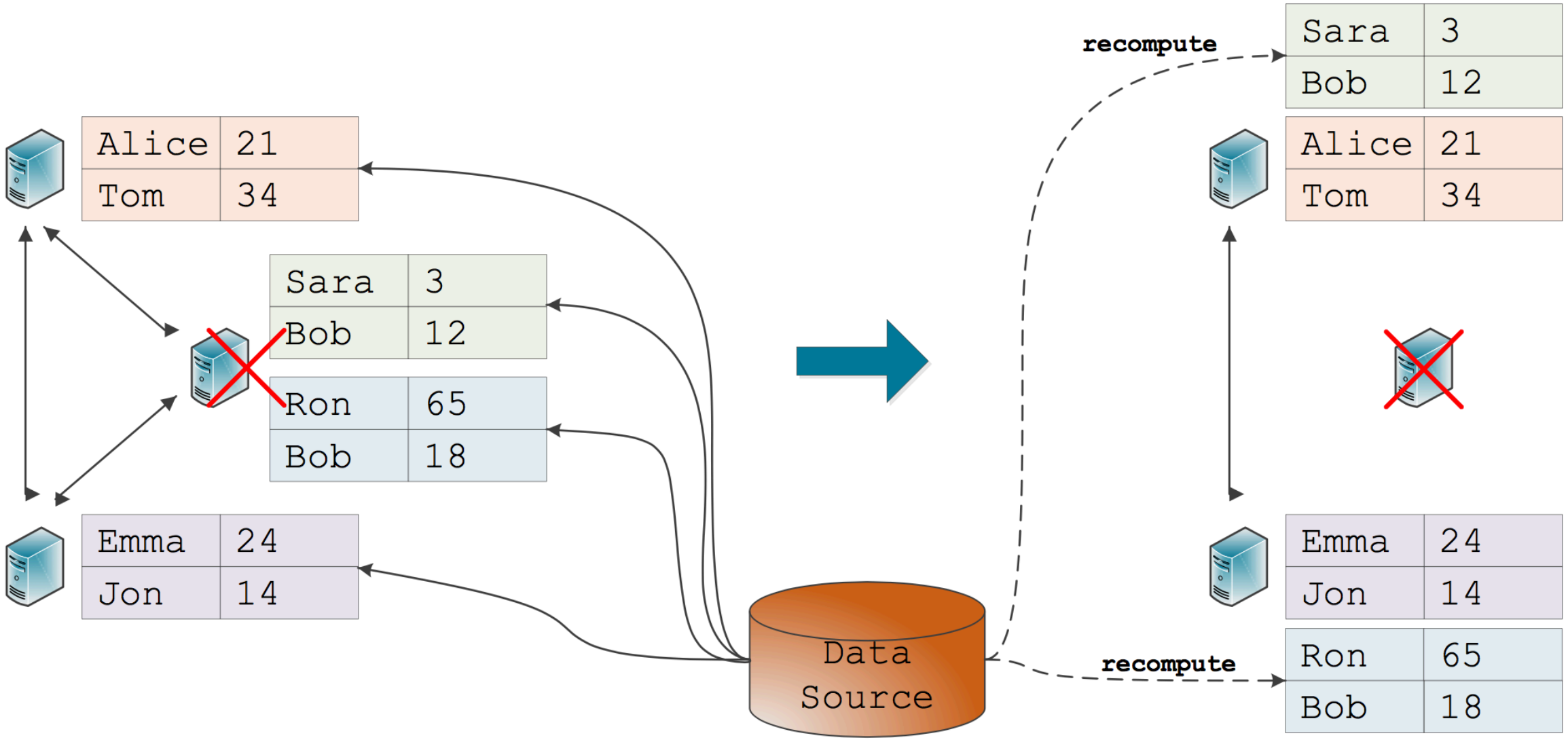
Spark RDD Partitioning

RDD[(String,Int)]

String	Int
Alice	21
Tom	34
Sara	3
Bob	12
Ron	65
Bob	18
Emma	24
Jon	14



Spark RDD Resiliency



Spark Transformations

Create a new data set from an existing one

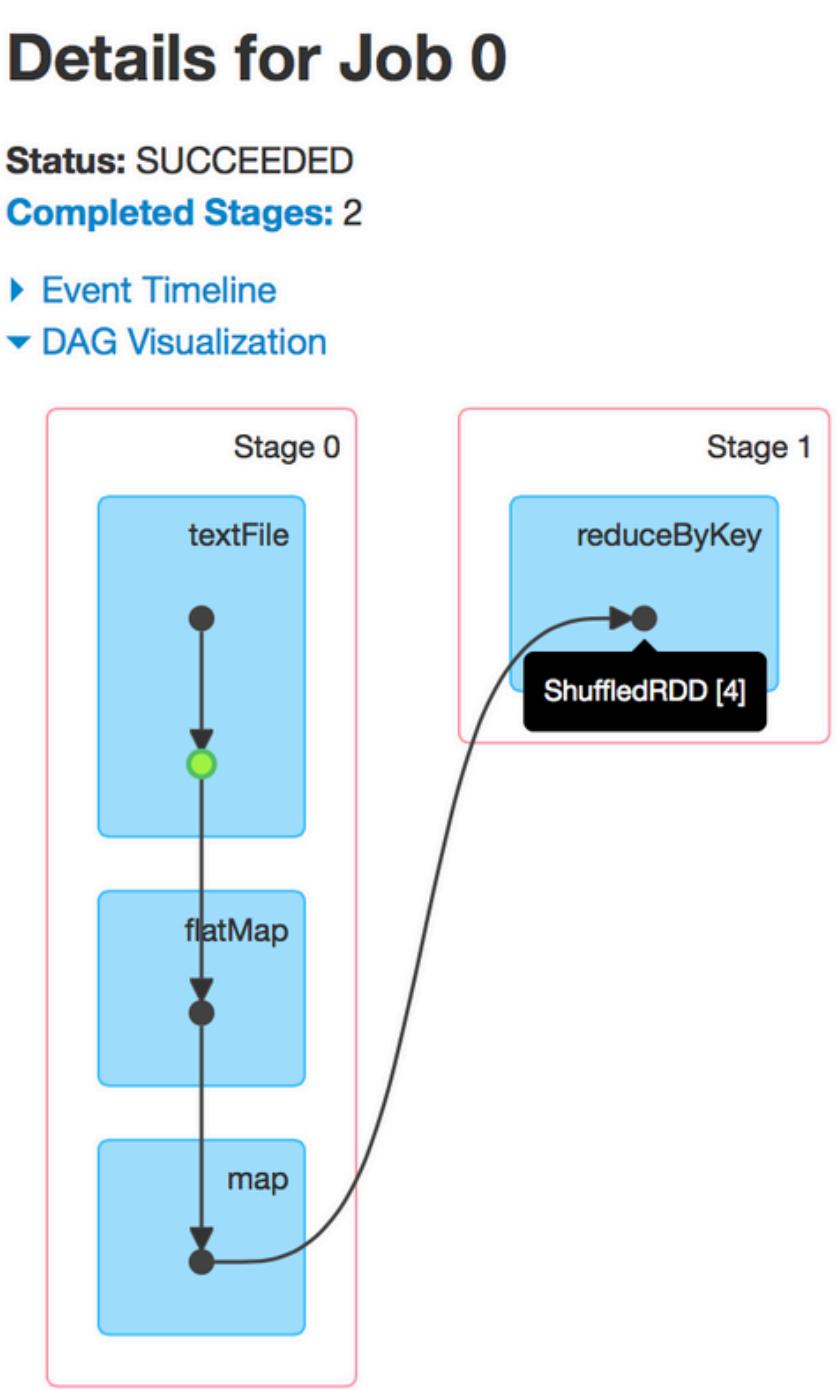
Transformation	Meaning
map(*func*)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
filter(*func*)	Return a new dataset formed by selecting those elements of the source on which *func* returns true.
flatMap(*func*)	Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item).
mapPartitions(*func*)	Similar to map, but runs separately on each partition (block) of the RDD, so *func* must be of type Iterator<T> => Iterator<U> when running on an RDD of type T.
mapPartitionsWithIndex(*func*)	Similar to mapPartitions, but also provides *func* with an integer value representing the index of the partition, so *func* must be of type (Int, Iterator<T>) => Iterator<U> when running on an RDD of type T.

Spark Actions

Return a value after running a computation

Action	Meaning
reduce(*func*)	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset (similar to take(1)).
take(*n*)	Return an array with the first *n* elements of the dataset. Note that this is currently not executed in parallel. Instead, the driver program computes all the elements.

Spark DAG



Spark Example (word count)

Given is a text file with movies and their categories.

We want to know the occurences of each genre.

```
Pirates of the Carribbean,Adventure,Action,Fantasy
Star Trek I,SciFi,Adventure,Action
Die Hard,Action,Thriller
```

Why Scala

Scala is a functional language

Analytics is a functional “problem“

So – use a functional language

Spark Example (word count)

```
sc.textFile("file:///home/videos.csv")
  .flatMap(line => line.split(",").drop(1))
  .map(genre => (genre,1))
  .reduceByKey{case (x,y) => x + y}
  .collect()
  .foreach(println)
```

Spark Example (word count)

```
sc.textFile("file:///home/videos.csv")
  .flatMap(line => line.split(",").drop(1))
  .map(genre => (genre,1))
  .reduceByKey((x,y) => x + y}
```

Anonymous Scala Function

Spark Example (word count)

```
sc.textFile("file:///home/videos.csv")
  .flatMap(line => line.split(",").drop(1))
  .map(genre => (genre,1))
  .reduceB{case (x,y) => x + y}
  .coll
  .fore
```

Flat the collection

Spark Example (word count)

```
sc.textFile("file:///home/videos.csv")
  .flatMap(line => line.split(",").drop(1))
  .map(genre => (genre,1))
  .reduceByKey{case (x,y) => x + y}
  .collect()
```

Pair RDD

Spark Example (word count)

Values of TWO Pair RDDs

```
sc.                                    e/videos.csv")
  .flatMap(line => line.split(",").drop(1))
  .map(genre => (genre,1))
  .reduceByKey{case (x,y) => x + y}
  .collect()
  .foreach(println)
```

Spark Example (word count)

```
sc.textFile("file:///home/videos.csv")
  .flatMap(line => line.split(",").drop(1))
  .map(genre => (genre,1))
  .reduceByKey{case (x,y) => x + y}
  .collect()
  .foreach(println)
```

Spark Example (word count)

```
sc.textFile("file:///home/videos.csv")
  .flatMap(line => line.split(",").drop(1))
  .map(genre => (genre,1))
  .reduceByKey{case (x,y) => x + y}
  .collect()
  .foreach(println)
```

Cassandra-Spark-Driver

Spark-Cassandra-Connector

- Scala, Java
- Open Source (ASF2)
- Developed by DataStax and community guys

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

Spark Batch Processing

Other REALLY useful stuff like

- Re-partition data & processing
- Allows efficient use of Cassandra

SQL and Data Frames

Spark SQL

```
csc.sql(" SELECT COUNT(*) AS total          " +
        " FROM killr_video.movies_by_actor " +
        " WHERE actor = 'Johnny Depp'       ")
   .show
```

```
+-----+
|total|
+-----+
|   54|
+-----+
```

Spark Data Frames

DATASTAX

Same as:

```
SELECT COUNT(*) AS total
FROM killr_video.movies_by_actor
WHERE actor = 'Johnny Depp'
```

```
val movies = csc
  .read
  .format("org.apache.spark.sql.cassandra")
  .options(Map( "keyspace" -> "killr_video",
                "table" -> "movies_by_actor" ))
  .load

movies.filter("actor = 'Johnny Depp'")
  .agg(Map("*" -> "count"))
  .withColumnRenamed("COUNT(1)", "total")
  .show
```

Spark Data Frames

DATASTAX

Data frame is RDD that has named columns

- Has schema
- Has rows
- Has columns
- Has data types

Spark Straming

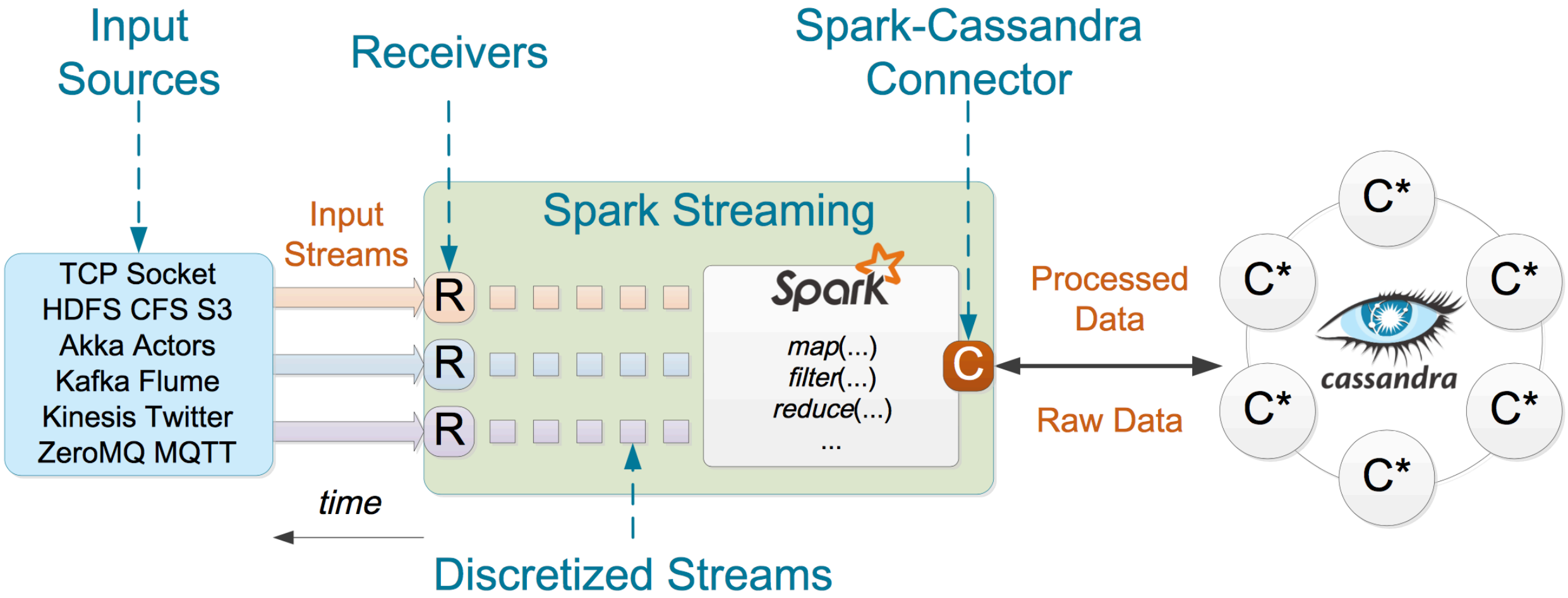
Spark Streaming

DATASTAX

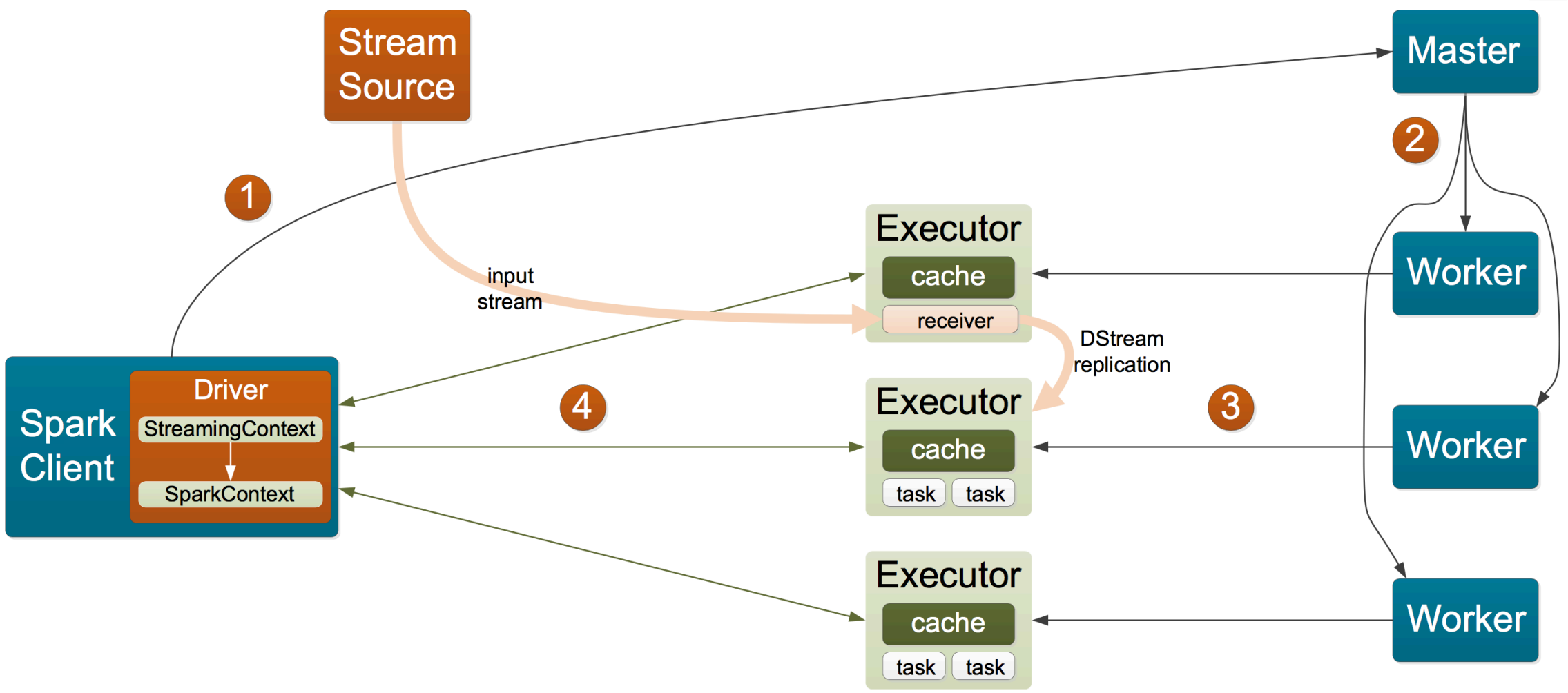
Continuous Micro Batching

- Collect data of time slides
- Compute on that data

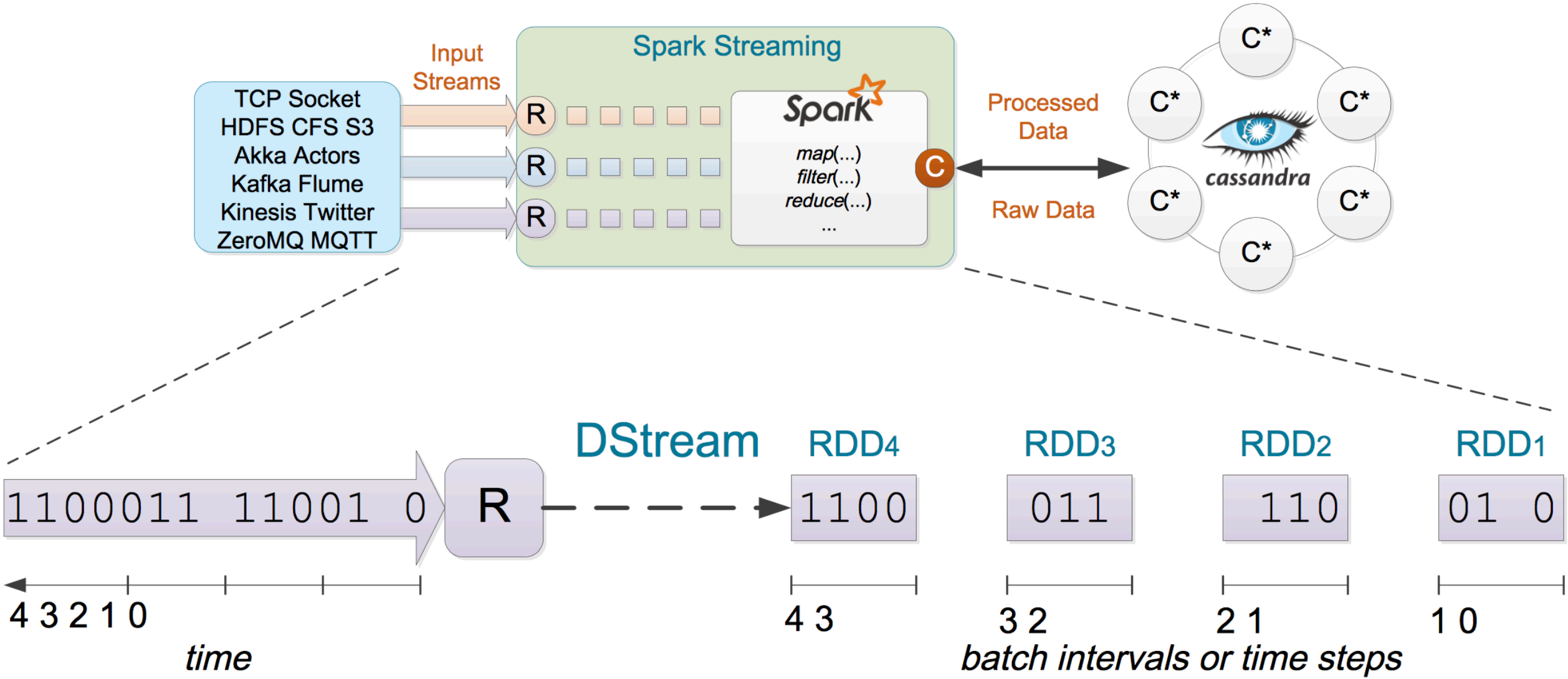
Spark Streaming



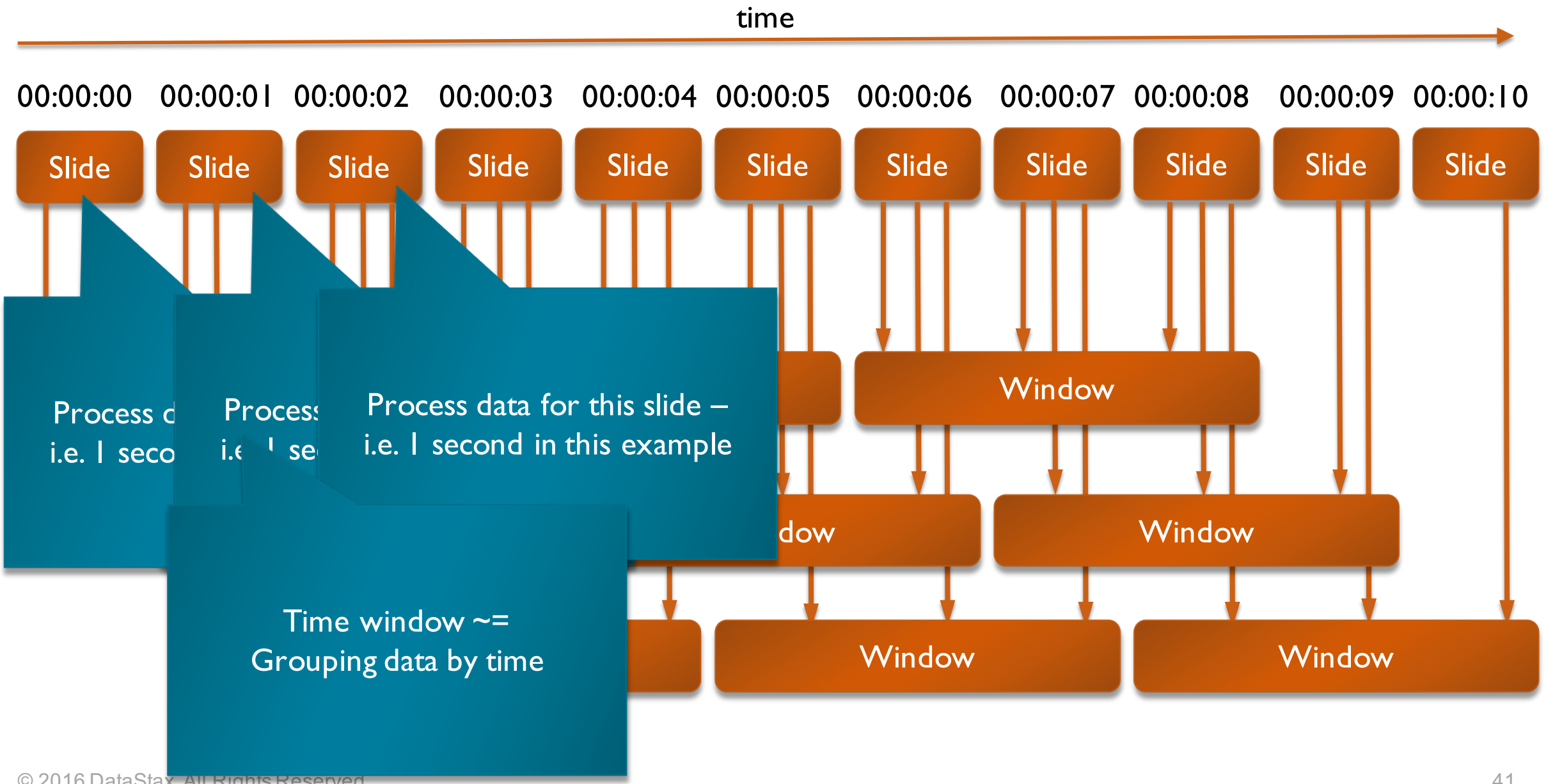
Spark Streaming Architecture



Spark Streaming



Spark Streaming Windows



Apache Kafka

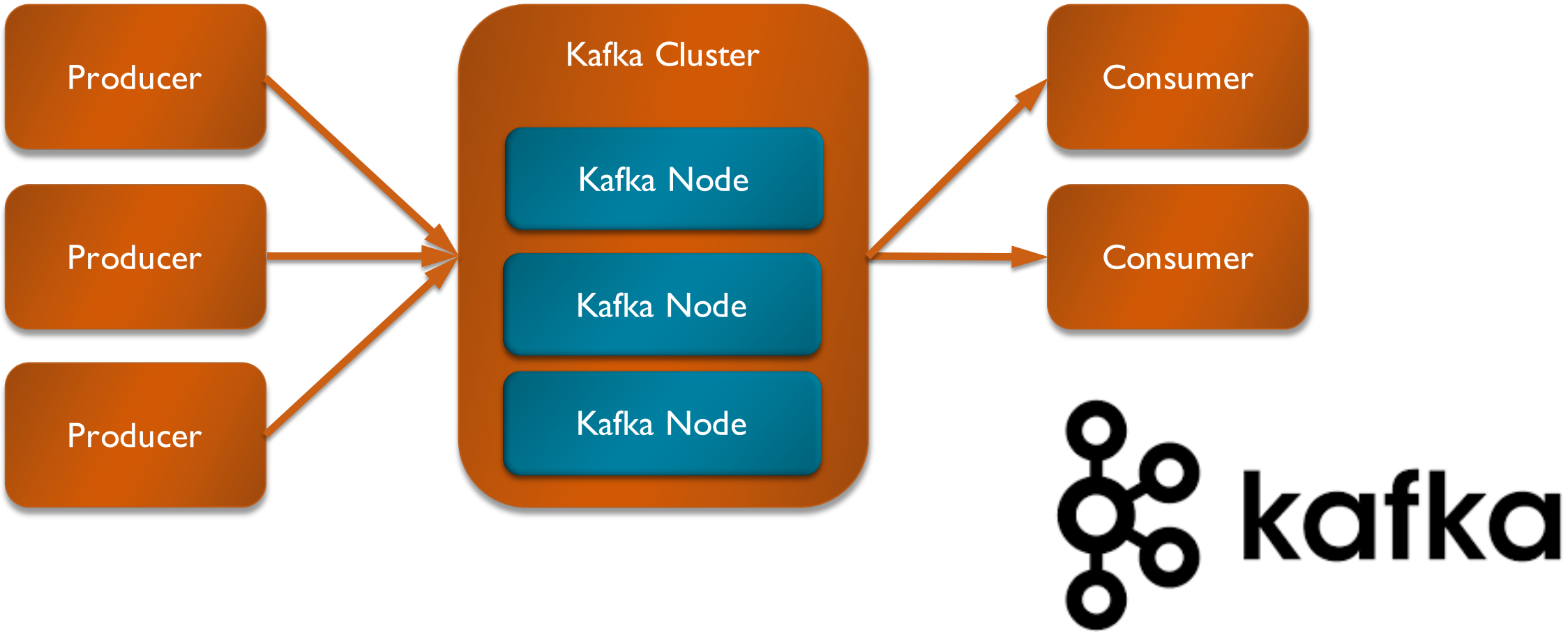
Distributed & **resilient** messaging

Kafka

DATASTAX

Publish-subscribe messaging
rethought as a distributed commit log

Kafka



Kafka Use-Cases

- Messaging
- Website activity tracking
- Metrics
- Log aggregation
- Event sourcing
- **Stream processing**

Common Characteristics

Kafka, Spark & Cassandra are

- Scalable
- Distributed
- Partitioned
- Replicated

KillrPowr

DATASTAX

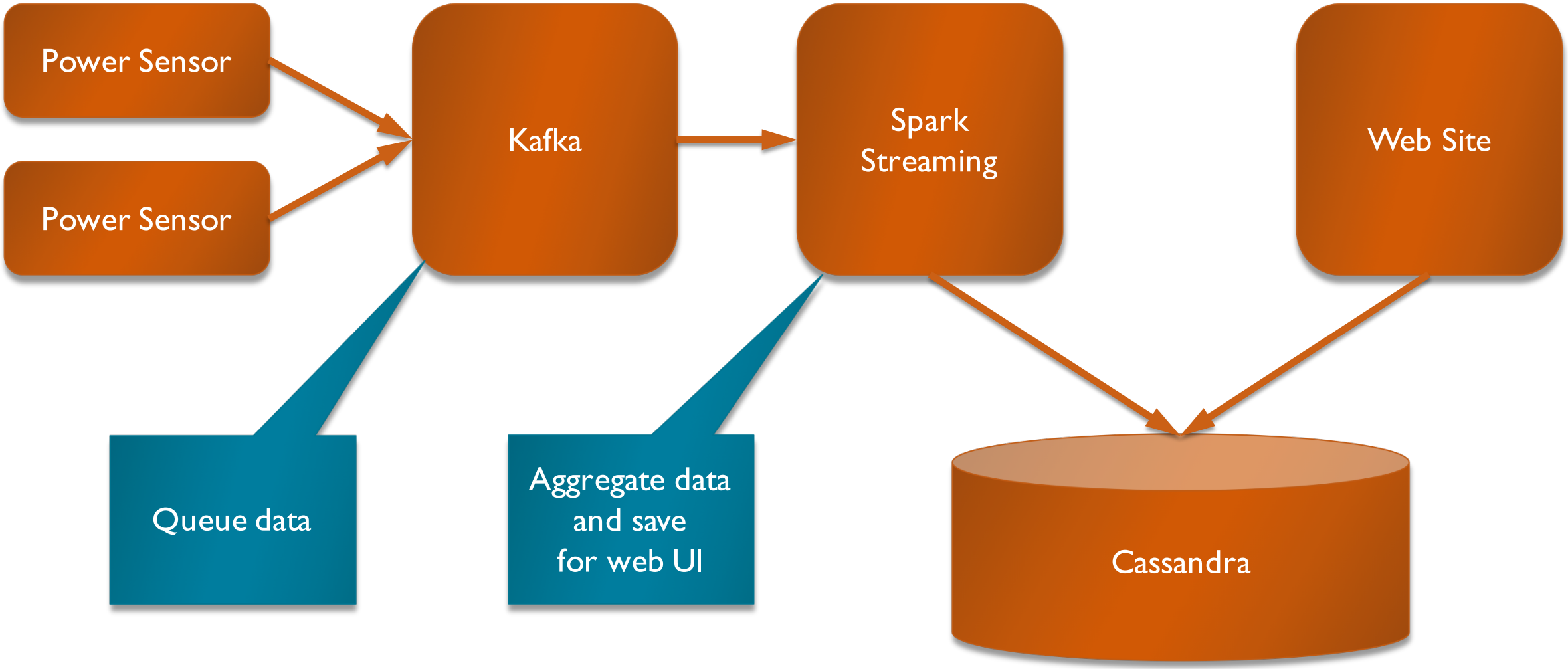
Show current power consumption in Germany in real time

- Power consumption of all cities on a map
- Graph of data for a selected city

KillrPowr – Data modeling 101

1. Collect UI Requirements
2. Model data (flow) per web UI request
3. Build Spark Streaming code to generate that data
4. Connect sensors and Spark Streaming using Kafka

KillrPowr – Tech Stack



KillrPowr – Tech Stack

DATASTAX

Techs used in the demo

- Spray (HTTP for Akka Actors)
- AngularJS
- Kafka
- Spark Streaming
- Cassandra

Tips

- Push down predicates to Cassandra
- Partition your data “right”
- Think distributed

Cassandra + Spark in DSE

DSE integrates
Spark + Cassandra

https://academy.datastax.com/courses/getting-started-apache-spark

KillrPowr Live Demo

You've got the lighter, let's start a fire: Spark and Cassandra

Robert Stupp

Solutions Architect @ DataStax, Committer to Apache Cassandra

@snazy

Thank You

BACKUP SLIDES

BACKUP - DDL

DATASTAX

```
-- cities is just master data
CREATE TABLE cities (
   loc_id              bigint PRIMARY KEY,
   name                text,
   lat                 float,
   lon                 float,
   population          int,
   area                float
);

-- ... preloaded with some data
COPY cities FROM 'cities.csv';
```

BACKUP - DDL

DATASTAX

```
-- Contains the timestamp of the most recent data in
-- power_by_time and power_by_city tables below.
CREATE TABLE lastwindow (
   dummy                text PRIMARY KEY,
   timewindow           timestamp
) WITH default_time_to_live = 3600;
```

BACKUP - DDL

```
-- Contains the power consumption for cities partitioned
-- by timestamp (power consumption by date/time).
CREATE TABLE power_by_time (
  timewindow      timestamp,
  --
  city_id         bigint,
  --
  lat             float,
  lon             float,
  name            text,
  population      int,
  area            float,
  power           float,
  --
  PRIMARY KEY ( timewindow, city_id )
) WITH default_time_to_live = 3600;
```

BACKUP - DDL

DATASTAX

```
-- Contains the power consumption for cities partitioned
-- by city (power consumption by city).
CREATE TABLE power_by_city (
  city_id           bigint,
  --
  timewindow        timestamp,
  --
  lat               float       STATIC,
  lon               float       STATIC,
  name              text        STATIC,
  population        int         STATIC,
  area              float       STATIC,
  --
  power             float,
  --
  PRIMARY KEY ( city_id, timewindow )
) WITH CLUSTERING ORDER BY (timewindow DESC)
  AND default_time_to_live = 3600;
```

BACKUP – Spark Streaming

DATASTAX

```
def main(args: Array[String])
{
  val conf = new SparkConf(true).
    setAppName("cstarSparkStreaming")

  val ssc = new StreamingContext(conf, slideDuration)
```

BACKUP – Spark Streaming

DATASTAX

```
// Setup the Kafka topic Spark DStream
val records: ReceiverInputDStream[(String, Arra
  KafkaUtils.createStream[String, Array[Byte],
    ssc,
    kafkaParams,
    topics,
    StorageLevel.MEMORY_AND_DISK_SER_2)
```

BACKUP – Spark Streaming

DATASTAX

```
val powerdiff = records
  // Deserialize the power record from the Kafka topic.
  .map(keyValue => deserialize[Power](keyValue._2))
  // Extract the ID of the city
  .map(power => (power.id, power))
  // Reduce by ID of the city (aggregate the power consumption
  .reduceByKeyAndWindow((power1: Power, power2: Power) =>
                          power1.copy(...),
                        windowDuration,
                        slideDuration)
  // Inject the current time-window timestamp
  .transform((rdd, time) => rdd.map(power =>
                                    {
                                      /.../
                                      TimestampedPower(...)
                                    }))
  // Map the case class into a tuple to be stored in Cassandra
  .map(...)
  // Cache the result since it's needed multiple times below
  .cache
```

BACKUP – Spark Streaming

DATASTAX

```
// Save records per time-window
//
// That's the power consumption over the last 60 seconds (windo
// for each city by time
powerdiff.saveToCassandra(cassandraKeyspace, "power_by_time",
                          SomeColumns("timewindow", "city_id",
                                      "name", "area", "populati

// Save records per city
//
// That's the power consumption over the last 60 seconds (windo
// for time-window by city
powerdiff.saveToCassandra(cassandraKeyspace, "power_by_city",
                          SomeColumns("timewindow", "city_id",
                                      "name", "area", "populati
```

BACKUP – Spark Streaming

DATASTAX

```
// Save last processed timestamp
//
// For this demo we have to save the current (means: "las
// find the current power consumption per city in the tab
powerdiff.map(tup => tup._1)
  .reduce((t, x) => t)
  // at this point we only have the one "row" containing
  // timestamp to be saved to Cassandra
  .map(t => ("dummy", t))
  .saveToCassandra(cassandraKeyspace, "lastwindow",
                   SomeColumns("dummy", "timewindow"))
```

BACKUP – Spark Streaming

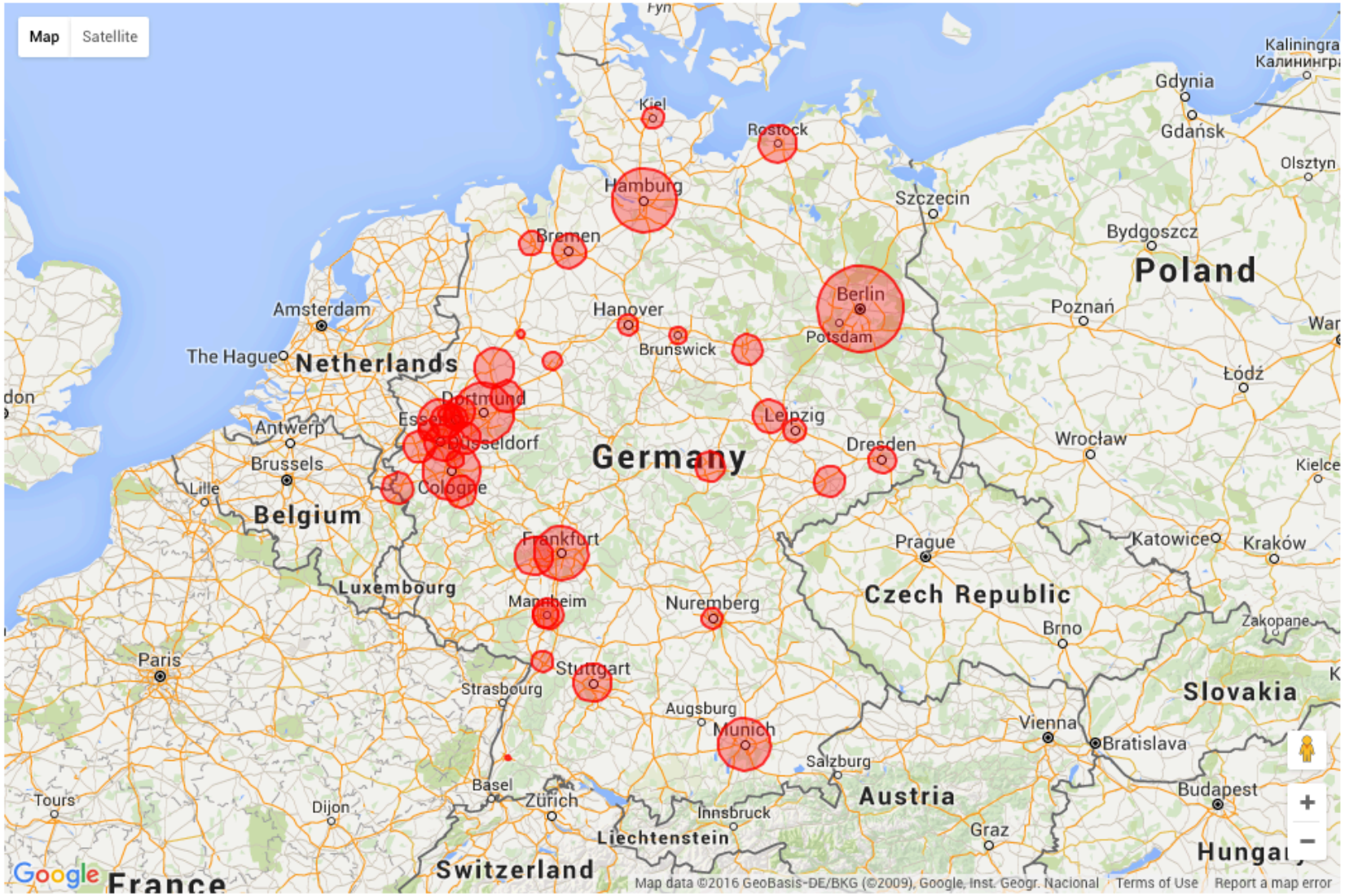
DATASTAX

```
ssc.start()

ssc.awaitTermination()
```

DATASTAX **KillrPowr**

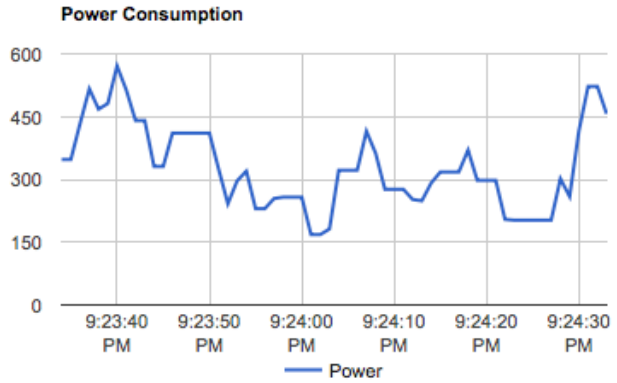
Last updated: Jan 29, 2016 9:24:33 PM



Köln

Power consumption: 456.79 kW

Population: 1,024,373
Area: 405.00 km^2



This is a demo application showing fake data. Thanks to the following projects and products!

Apache Cassandra (database), Apache Spark (analytics), both combined in DataStax Enterprise. Also Apache Kafka for piping data.

Coded in Scala and JavaScript using Spray, Akka, AngularJS, Google Maps & Charts