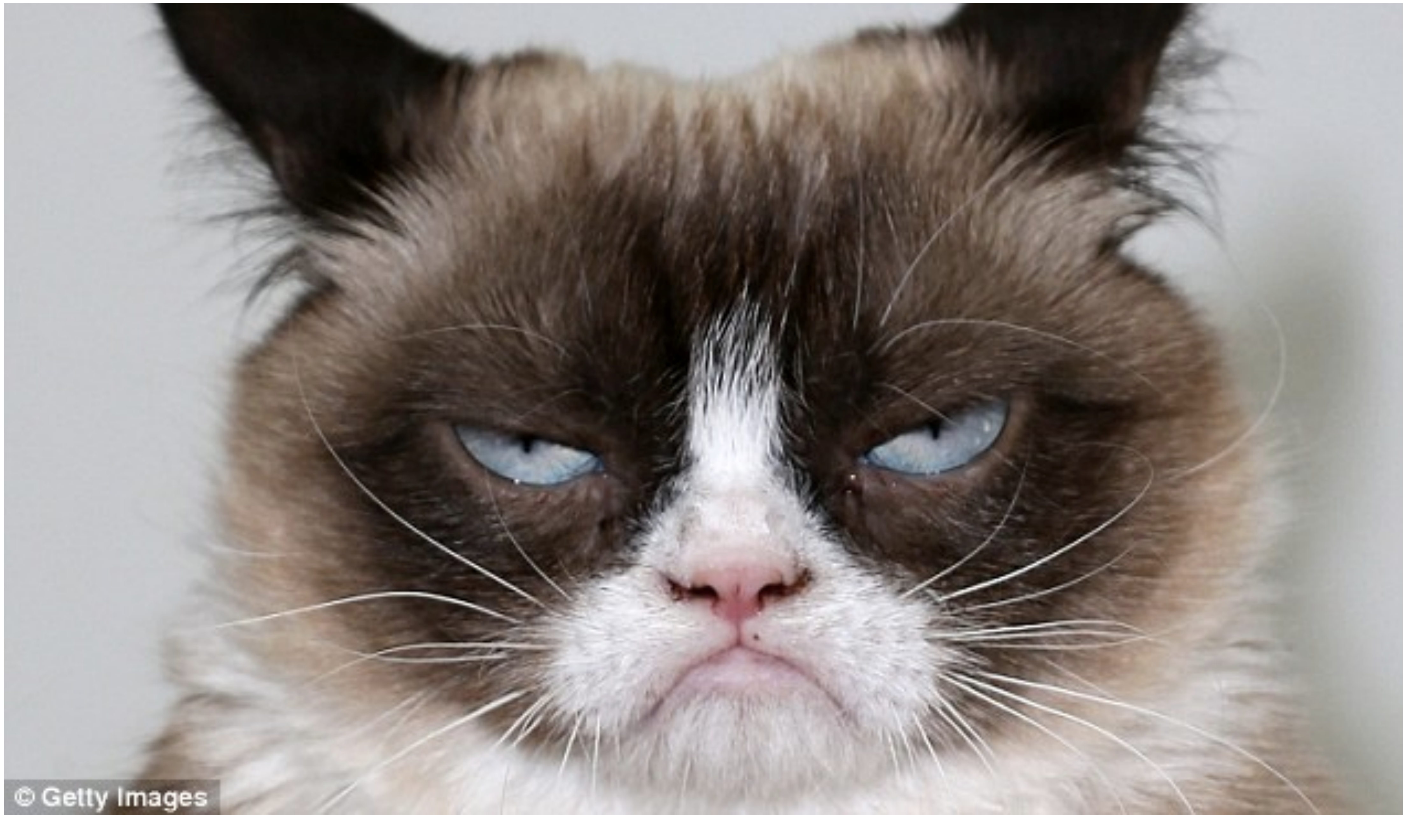
Spark Streaming Data with Wings

Never stop the movement

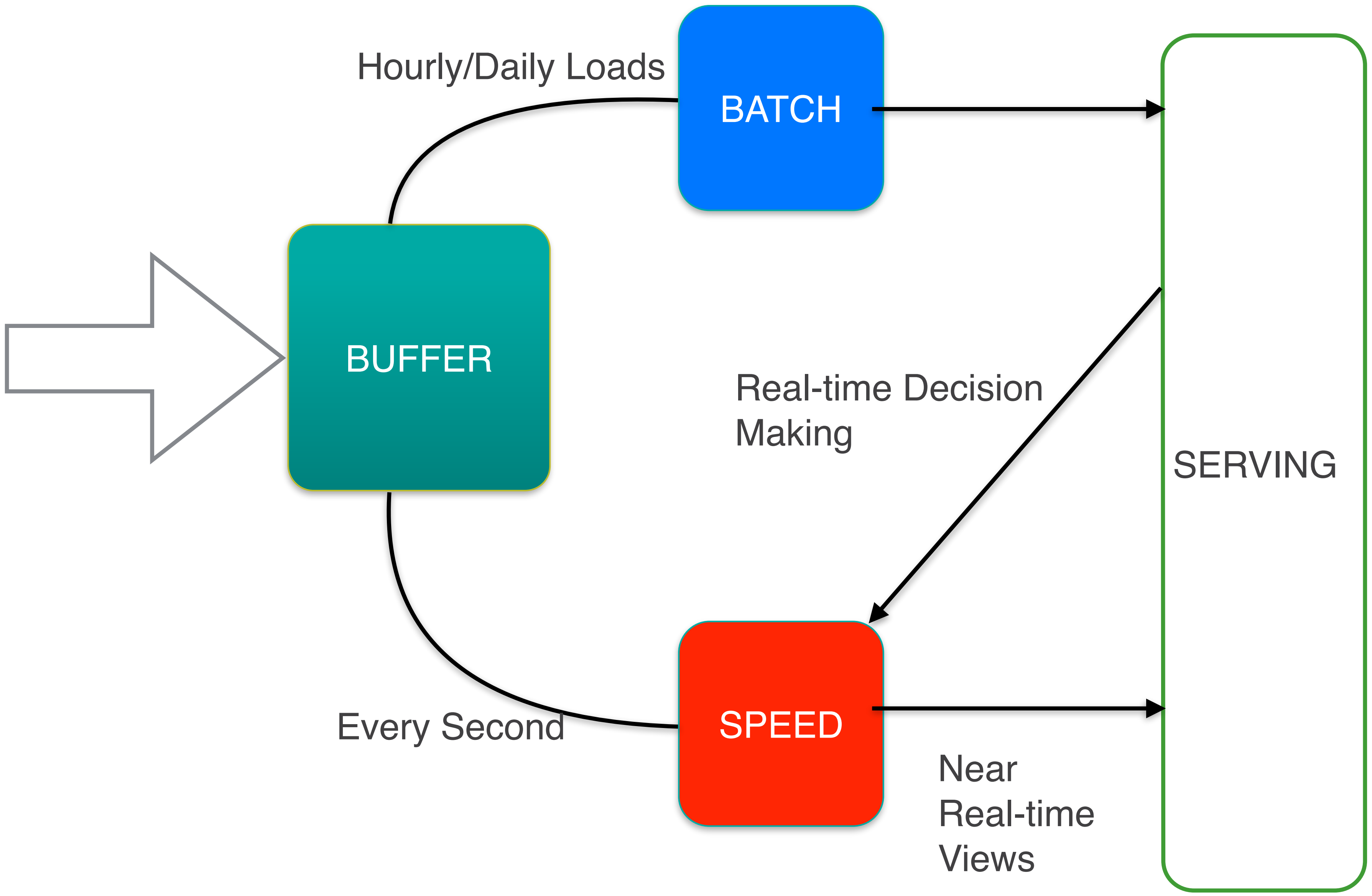
Agenda

- Spark overview
- Spark Streaming overview
- Processing streams from meetup.com
- Little intro to stateful stream processing
- All together

There will be no pictures of cats!



Lambda Architecture

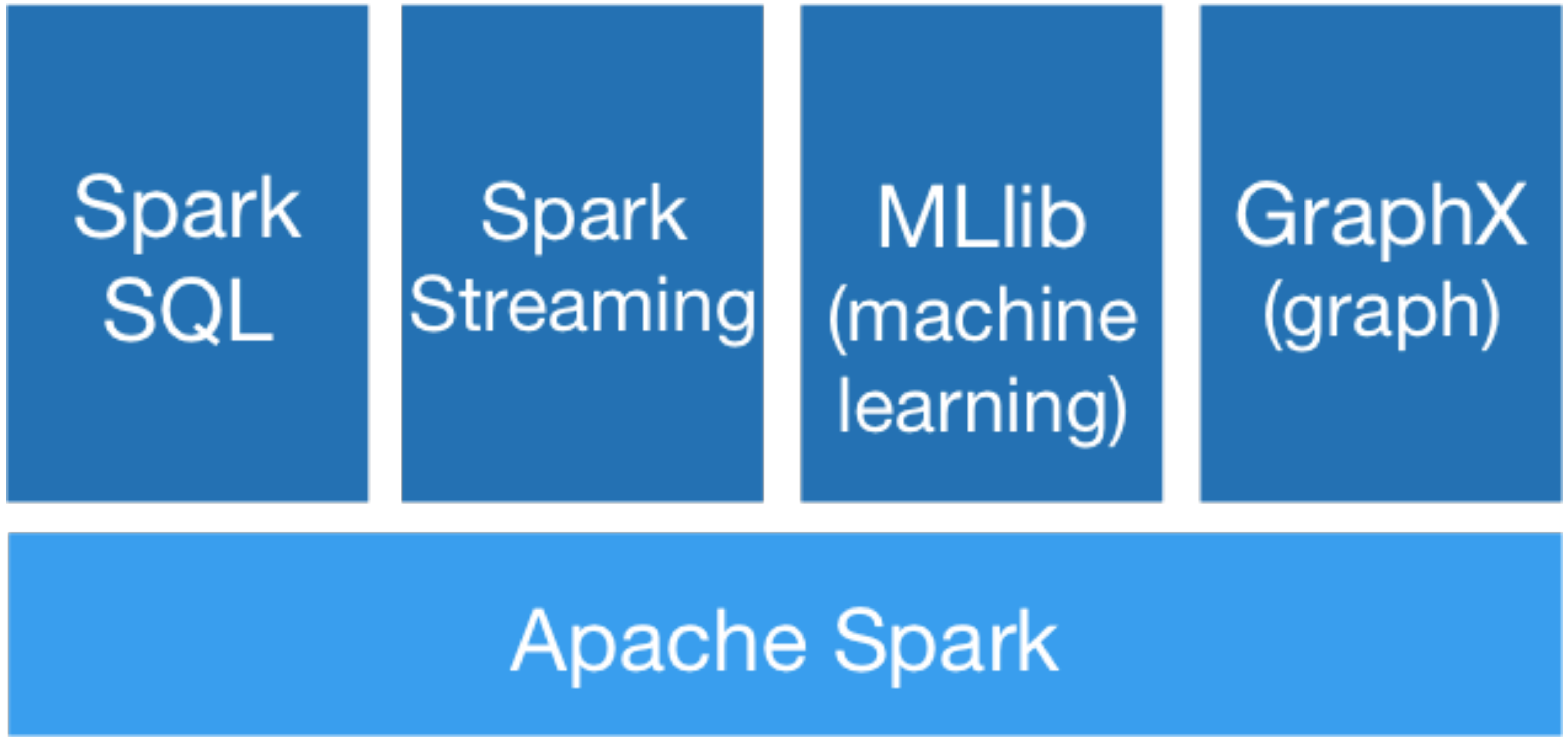


Do you see any problems with Lambda Architecture?

Spark Overview

What if you could write programs operating with petabyte size **datasets** and large **streams** same way you operate Iterable collections?

Apache Spark



Resilient **D**istributed **D**ataset

Resilient - resilience is addressed by tracking the log of operations performed on the dataset. Because of the side effects are eliminated, every lost partition can be recalculated in case of a loss.

Distributed - the dataset is partitioned. We can specify partitioning scheme for every operation.

Dataset - can be built from regular files, HDFS large files, Cassandra table, HBase table, etc.

Obligatory word count

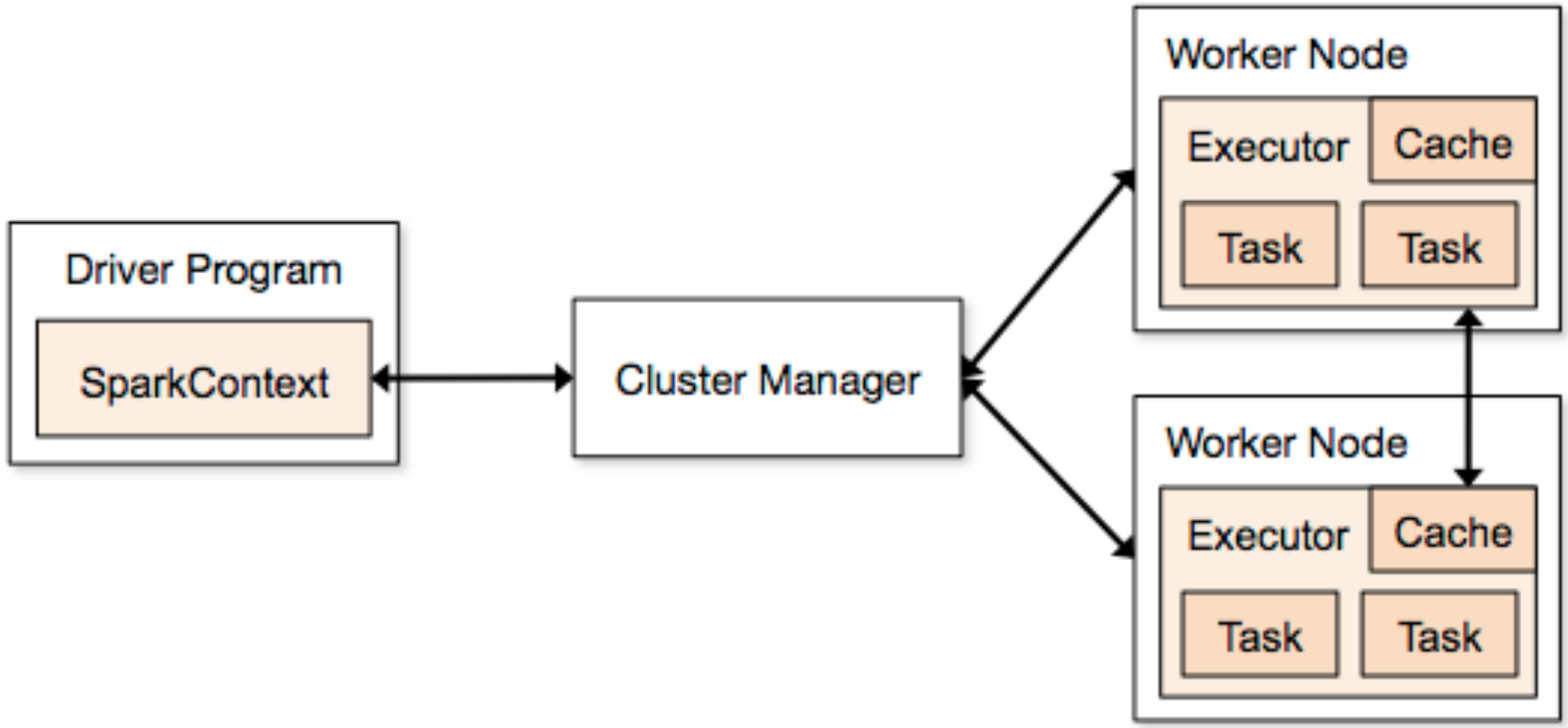
```
lines = spark.textFile("hdfs://...")

lines.flatMap{line: line.split(“ “)}

    .map({word: (word, 1)})

    .reduceByKey(_+_)
```

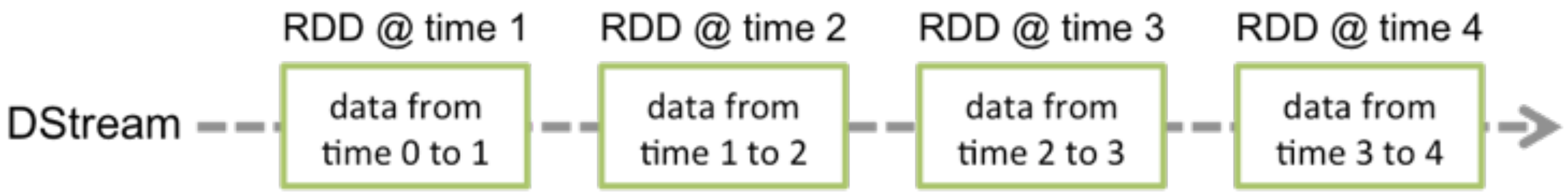
Spark Runtime



Spark is like Yoda and Hulk combined

Discretized **S**tream

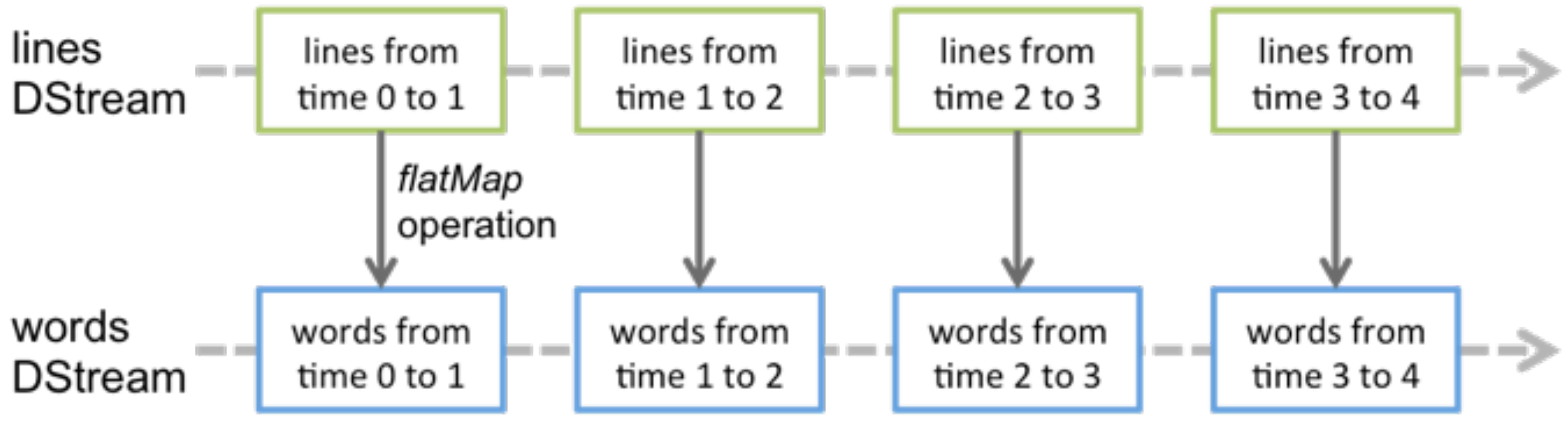
New RDD every second



https://spark.apache.org/docs/1.2.0/streaming-programming-guide.html

DStream - still pretty

You operate on stream just like on collections



https://spark.apache.org/docs/1.2.0/streaming-programming-guide.html

meetup.com Streams

Let’s play with meetup.com streams:

- Events
- RSVPs

RSVP Schema

```
{ "event" : { "event_id" : "220993343",

      "event_name" : "Paper Collage",

      "event_url" : "http://www.meetup.com/ELEOS-Art-School-Studio/events/220993343/",

      "time" : 1427551200000

    },

  "group" : {...},

  "guests" : 0,

  "member" : { "member_id" : 120762942,

      "member_name" : "Esther",

      "photo" : "http://photos1.meetupstatic.com/photos/member/b/b/0/thumb_159962992.jpeg"

    },

  ...

}
```

Event Schema

```
{ "description" : "<p>90 Minute walking tour with your dog!  Please arrive early.</p>\n<p>Tour of Balboa Parks spookiest
locations on ...>",

  "duration" : 5400000,

  "event_url" : "http://www.meetup.com/SanDiegoDogWalkers/events/220302036/",

  "group" : {...},

  "id" : "220302036",

  "maybe_rsvp_count" : 0,

  "mtime" : 1425785739616,

  "name" : "After Dark Ghost Walking Tour in Balboa Park with your dog!",

  "payment_required" : "0",

  "rsvp_limit" : 20,

  "status" : "upcoming",

  "time" : 1426993200000,

  "utc_offset" : -25200000,

  "venue" : {....},
```

Meetup Receiver

Receiver is the way to pump data to spark streaming.

In our case, we connect to meetup streaming api and send json to Spark.

Code is a bit long, but you can explore it:

https://github.com/actions/meetup-stream/blob/master/src/main/scala/receiver/MeetupReceiver.scala

Intro to Stateful Stream Processing

```
def locaitonCounts(eventsStream: DStream[(Venue,Event)])=
  liveEvents
    .filter{case(venue,event)=>venue.country==“usa”}
    .map{case(venue,event)=>(venue.toCityState,1)}
    .updateStateByKey(countForLocaiton)
    .print


 def countForLocaiton(counts: Seq[Int], initCount: Option[Int])
=Some(initCount.getOrElse(0) + counts.sum)
```

How it works again…?

Incoming Stream

CityState	Count
San Francisco,CA	1
Miami, FL	1

CityState	Count
San Francisco,CA	1
Miami, FL	1

10 sec

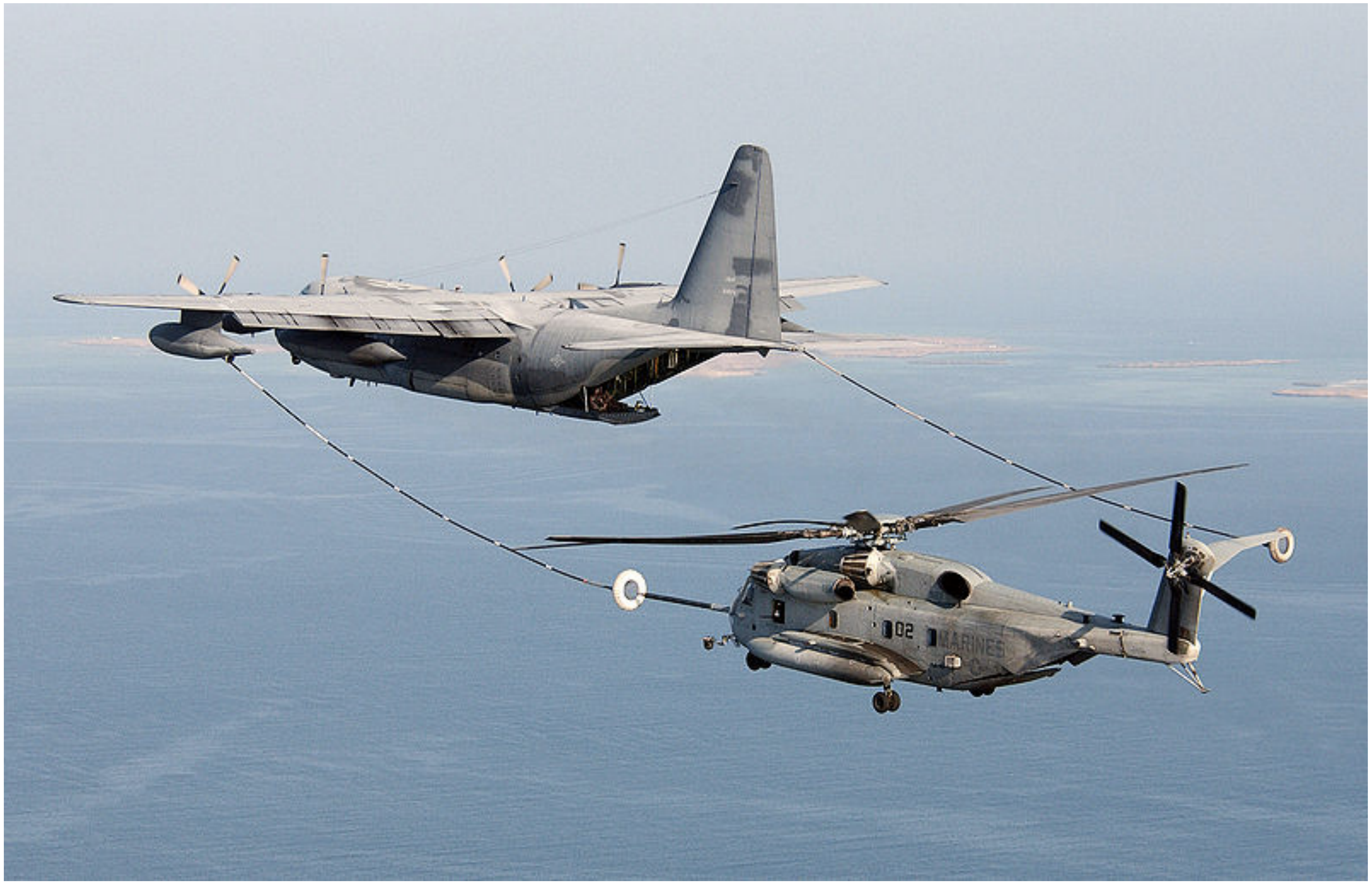
11 sec

CityState	Count
San Francisco, CA	5
LA, CA	3
Portland, OR	2

CityState	Count
San Francisco, CA	6
LA, CA	3
Portland, OR	2
Miami, FL	1

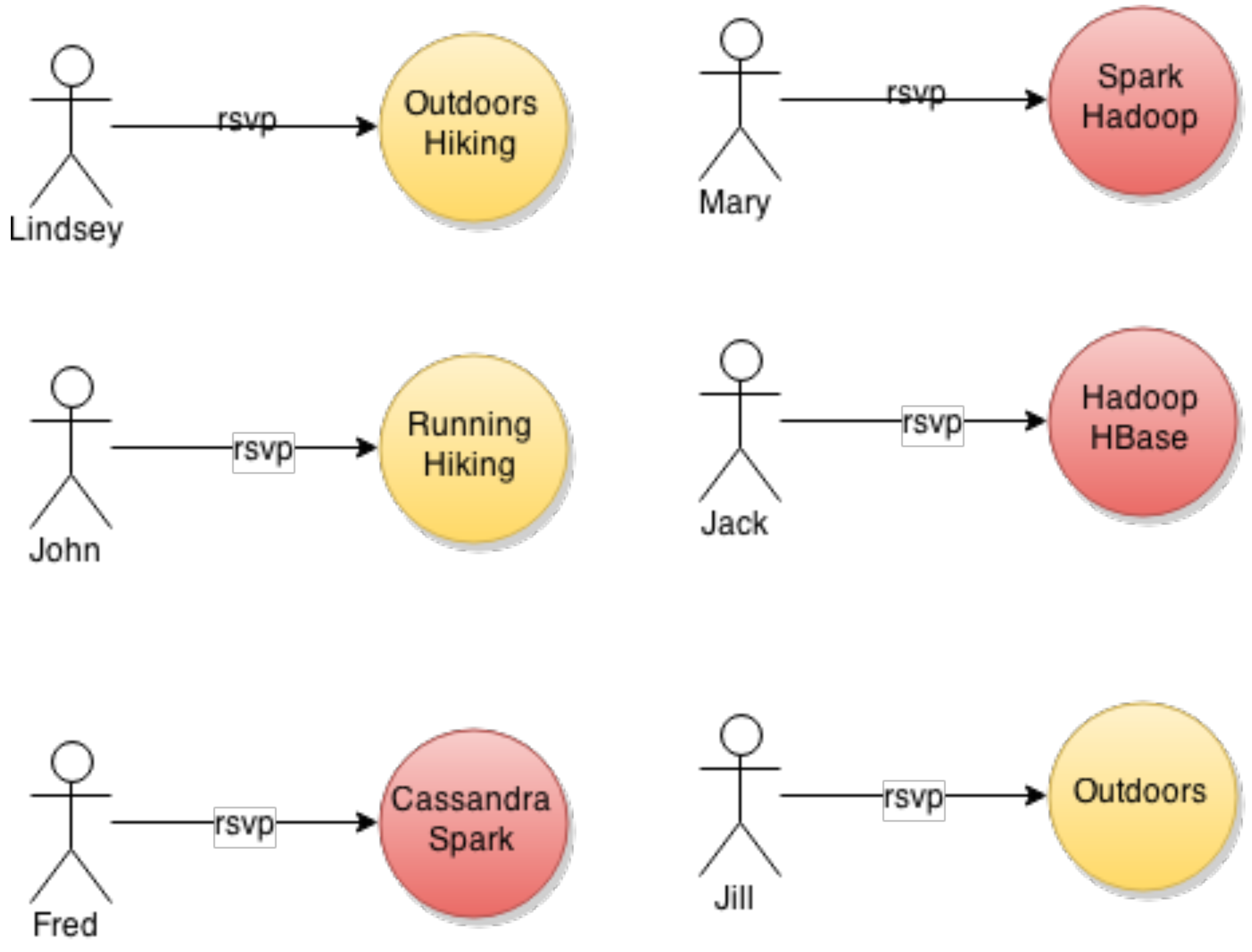
State Stream

Aerial refueling

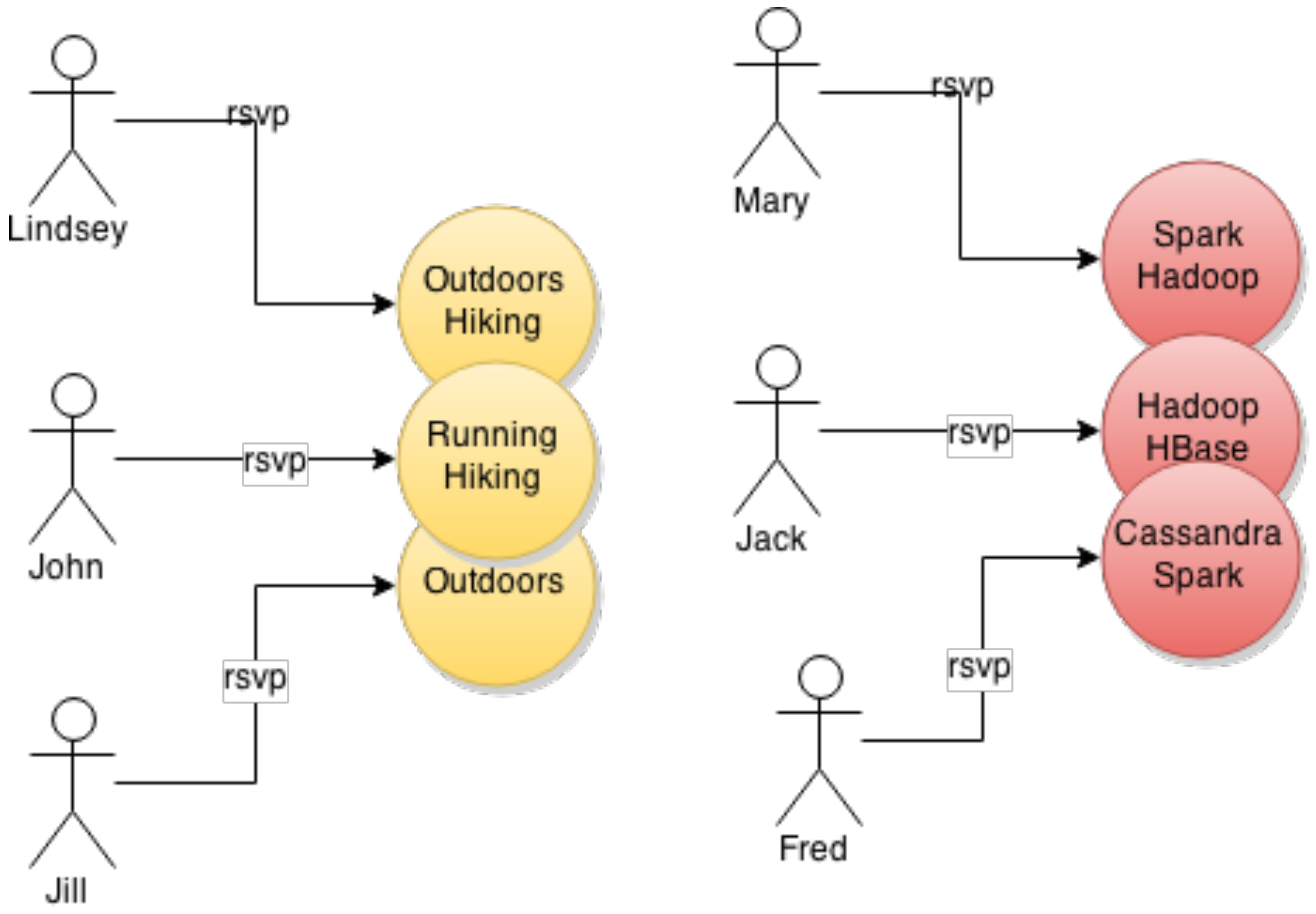


http://commons.wikimedia.org/wiki/File:Aerial_refueling_CH-53_DF-SD-06-02984.JPG

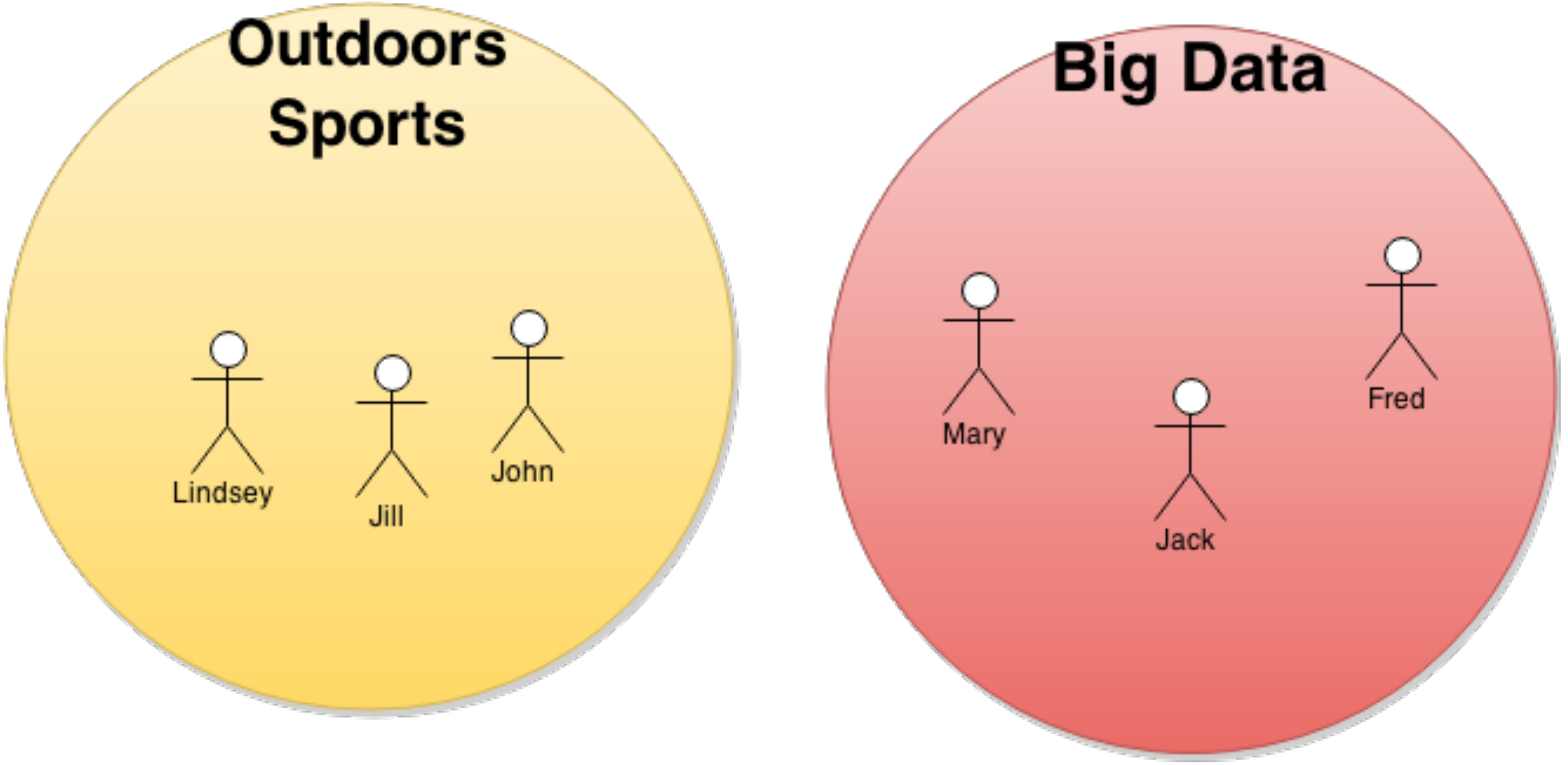
meetup.com connection recommendation app



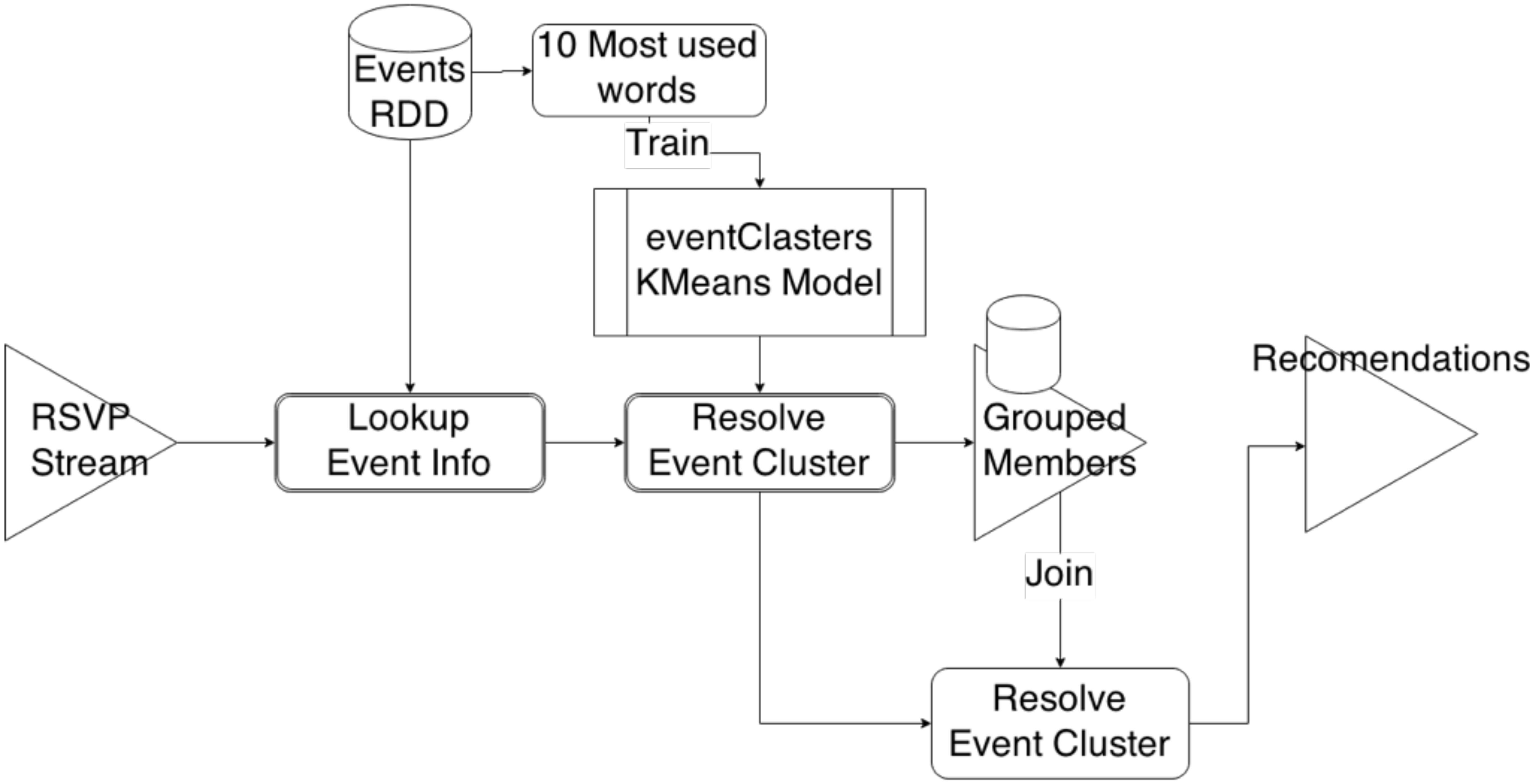
Meetup Event Clustering



Meetup Professional Connection Recommendation



Meetup Recommendations Pipeline



Initializing RSVP Stream and the Event Dataset

```
  val conf = new SparkConf()
      .setMaster("local[4]")
      .setAppName("MeetupExperiments")
      .set("spark.executor.memory", "1g")
      .set("spark.driver.memory", "1g")

val ssc=new StreamingContext(conf, Seconds(1))

val rsvpStream = ssc.receiverStream(
  new MeetupReceiver("http://stream.meetup.com/2/
rsvps")).flatMap(parseRsvp)

val eventsHistory = ssc.sparkContext.textFile("data/events/
events.json", 1).flatMap(parseEvent)
```

Broadcasting Dictionary

```
 val localDictionary=Source
    .fromURL(getClass.getResource("/wordsEn.txt"))
    .getLines
    .zipWithIndex
    .toMap

val dictionary=ssc.sparkContext
                  .broadcast(localDictionary)
```

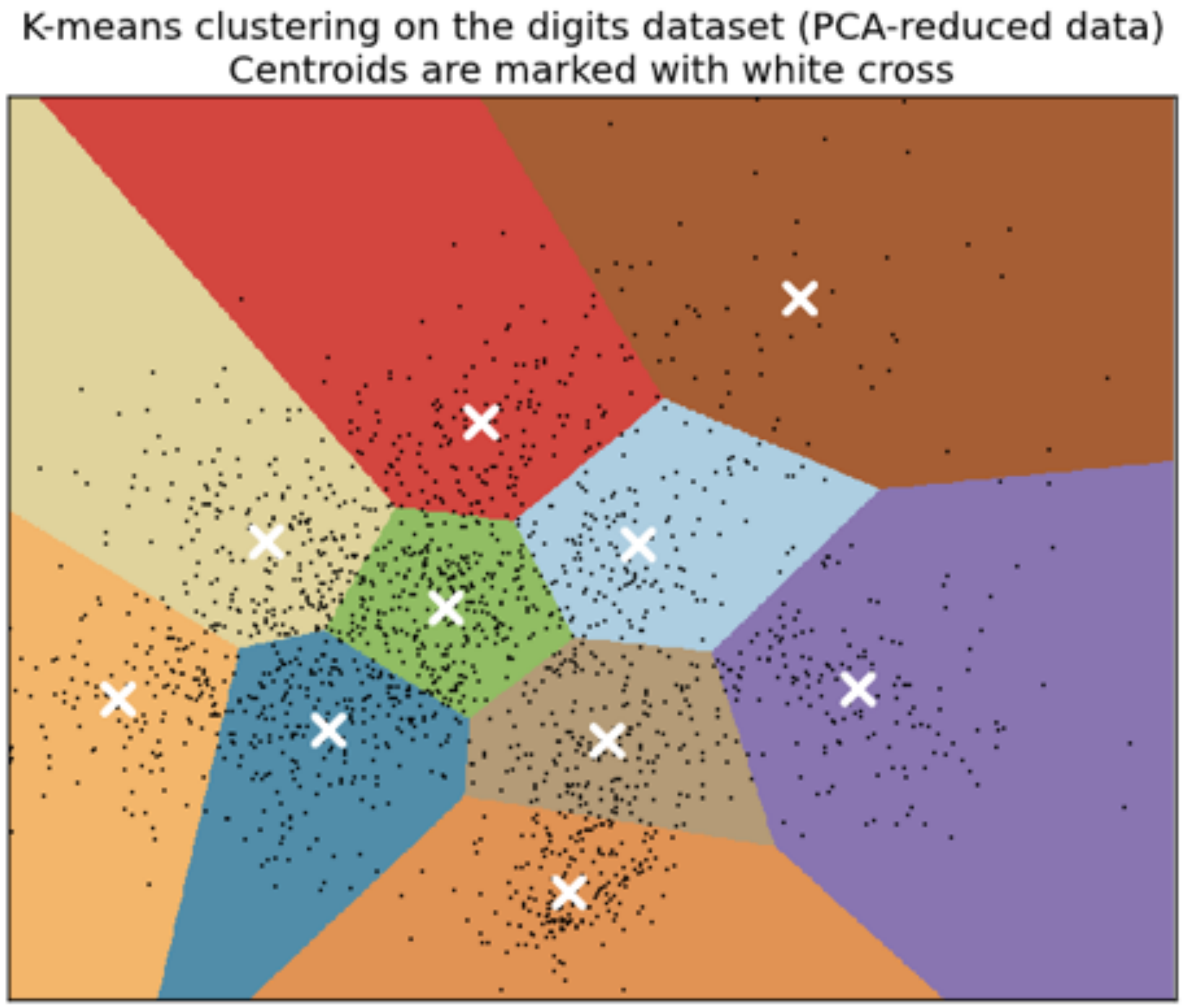
Feature Extraction

10 most popular words in the description.

```
def eventToVector(event: Event): Option[Vector]={
   val wordsIterator =
event.description.map(breakToWords).getOrElse(Iterator())
   val topWords=popularWords(wordsIterator)
   if (topWords.size==10)
Some(Vectors.sparse(dictionary.value.size,topWords)) else None
  }

val eventVectors=eventsHistory.flatMap{event=>eventToVector(event)}
```

Training based on existing dataset



```
val eventClusters = KMeans.train(eventVectors, 10, 2)
```

http://scikit-learn.org/0.11/auto_examples/cluster/plot_kmeans_digits.html

Event History By ID

```
val eventHistoryById=eventsHistory
  .map{event=>(event.id, event.description.getOrElse(""))}
  .reduceByKey{(first: String, second: String)=>first}
```

(220302036,“…description1 …”)
(220302037,“…description2 …”)
(220302038,”…description3 …”)

Streaming lookups

Looking up the event description by eventId from rsvp.

```
val rsvpEventInfo = membersByEventId.transform(
   rdd=>rdd.join(eventHistoryById)
)
```

```
(eventId, (member, response), description)
```

```
(220819928,((Member(Some(cecelia rogers),Some(162556712)),yes),"...")
(221153676,((Member(Some(Carol),Some(183499291)),no),"...")
…
```

Streaming Clustering

```
val memberEventInfo = rsvpEventInfo
   .flatMap{
   case(eventId, ((member, response), event)) => {
    eventToVector(event).map{ eventVector=>
      val eventCluster=eventClusters.predict(eventVector)
      (eventCluster,(member, response))
    }
   }}
```

Clustering members

```
def groupMembers(memberResponses: Seq[(Member, String)], initList: Option[Set[Member]]) =
{
      val initialMemberList=initList.getOrElse(Set())
      val newMemberList=(memberResponses :\ initialMemberList) {
      case((member, response), memberList) =>
        if (response == "yes") memberList + member else memberList - member
      }
      if (newMemberList.size>0) Some(newMemberList) else None
    }
```

```
val memberGroups =
memberEventInfo.updateStateByKey(groupMembers)
```

Recommendations

```
 val recommendations=memberEventInfo
   .join(memberGroups)
   .map{
      case(cluster, ((member, memberResponse), members)) =>
             (member.memberName, members-member)
   }
```

```
(Some(Mike D) -> Set(Member(Some(Sioux),Some(85761302)),
Member(Some(Aileen),Some(12579632)),
Member(Some(Teri),Some(148306762))))
```

Questions