Graph Flow

Using Spark and ~~Shark~~ SparkSQL to Power a Real-time Recommendation and Customer Intelligence Platform

About

- @MLnick
- Co-founder @graphflow - big data & machine learning applied to recommendations, consumer behaviour & insights
- Apache Spark committer
- Author of “Machine Learning with Spark”
    - *Packt RAW: http://www.packtpub.com/machine-learning-with-spark/book*

GraphFlow

Agenda

- A bit about Graphflow
- Architecture overview:
  - *Graphflow Platform*
  - *Why Spark?*
  - *Evolution of our architecture*
- Future

GraphFlow

1

Graphflow's platform captures interactions between users and items:

Items can be almost anything, from ***products* to *movies* to *music* to *apps.***

Interactions can include ***view*, *purchase*, *click, like*** and anything in between.

2

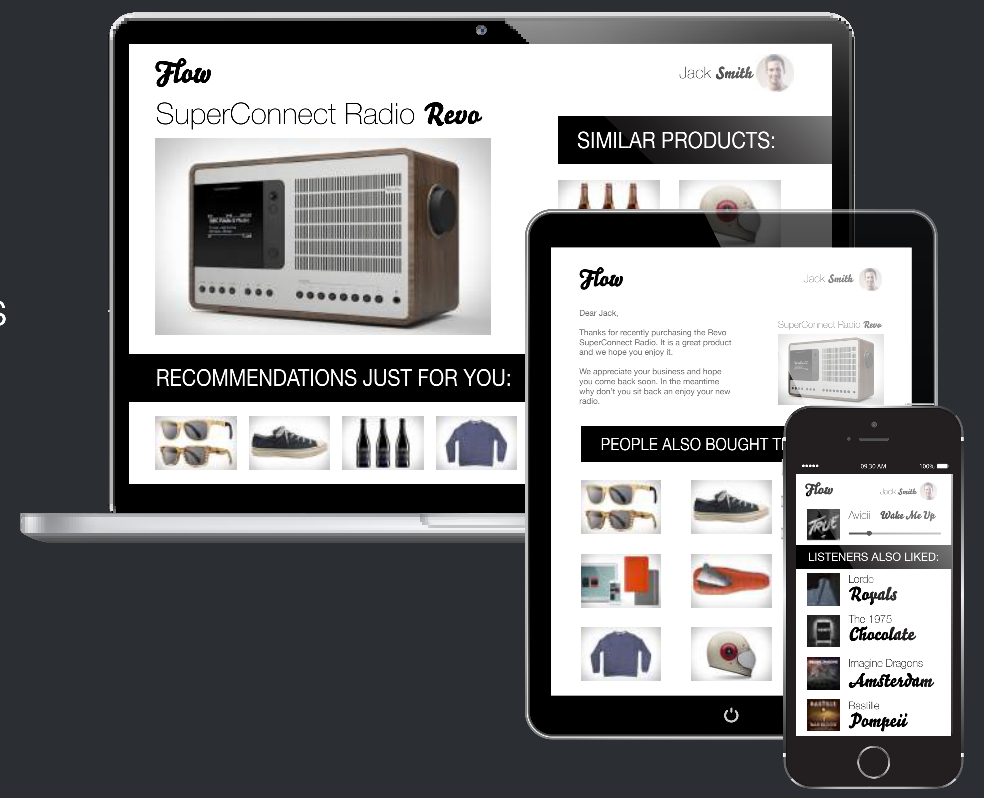
We use state-of-the art **machine learning** and **large-scale analytics techniques** to analyse this behaviour and generate recommendations for users and items.

Our flexible integration layer can deliver **real-time**, **personalised** recommendations across any channel, including **online**, **mobile**, **email** and **SMS.**

Our analytics portal provides **deep insight** into customer behaviour and product performance

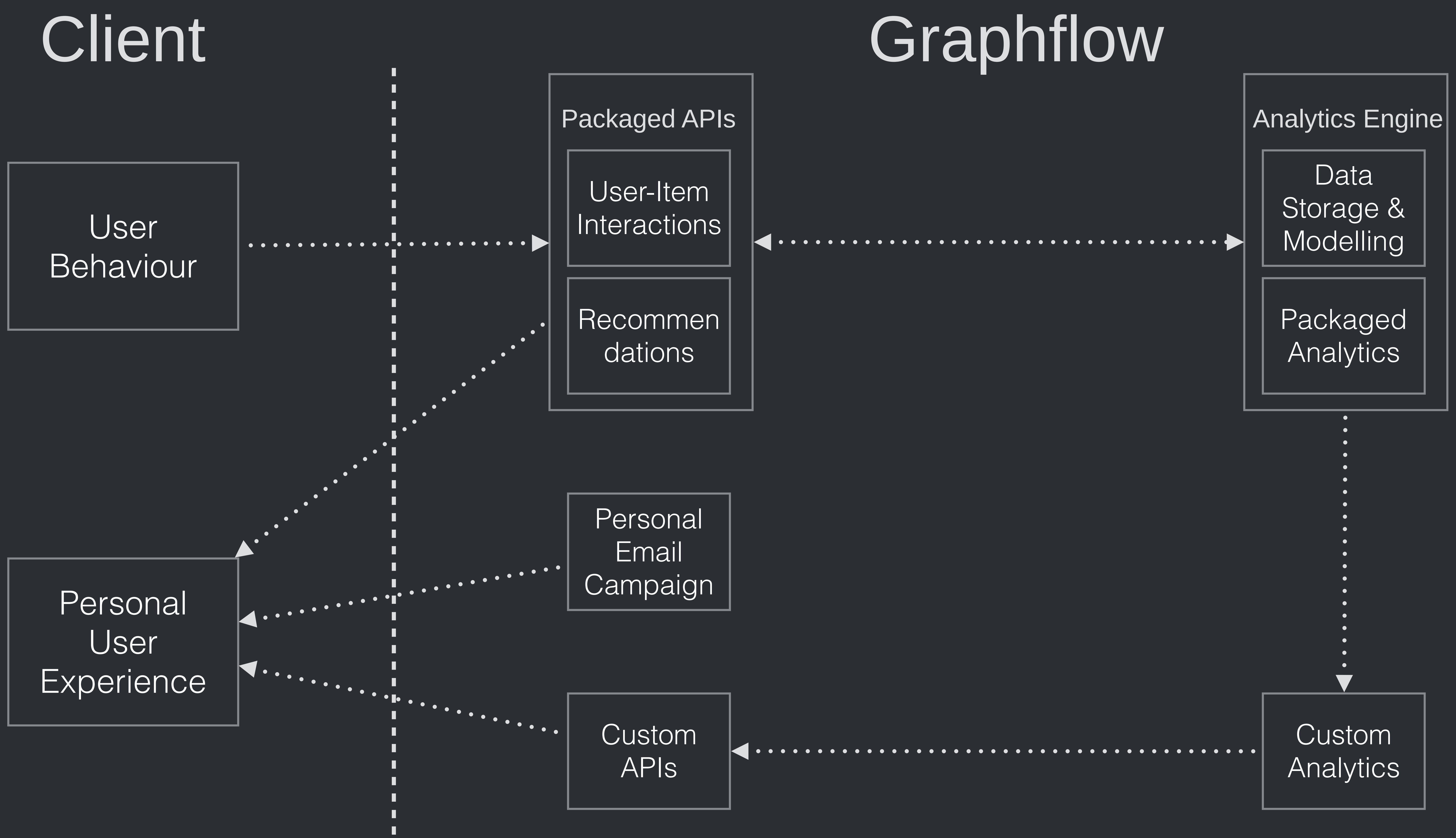
3

Omni-channel Recommendations



Customer Insight

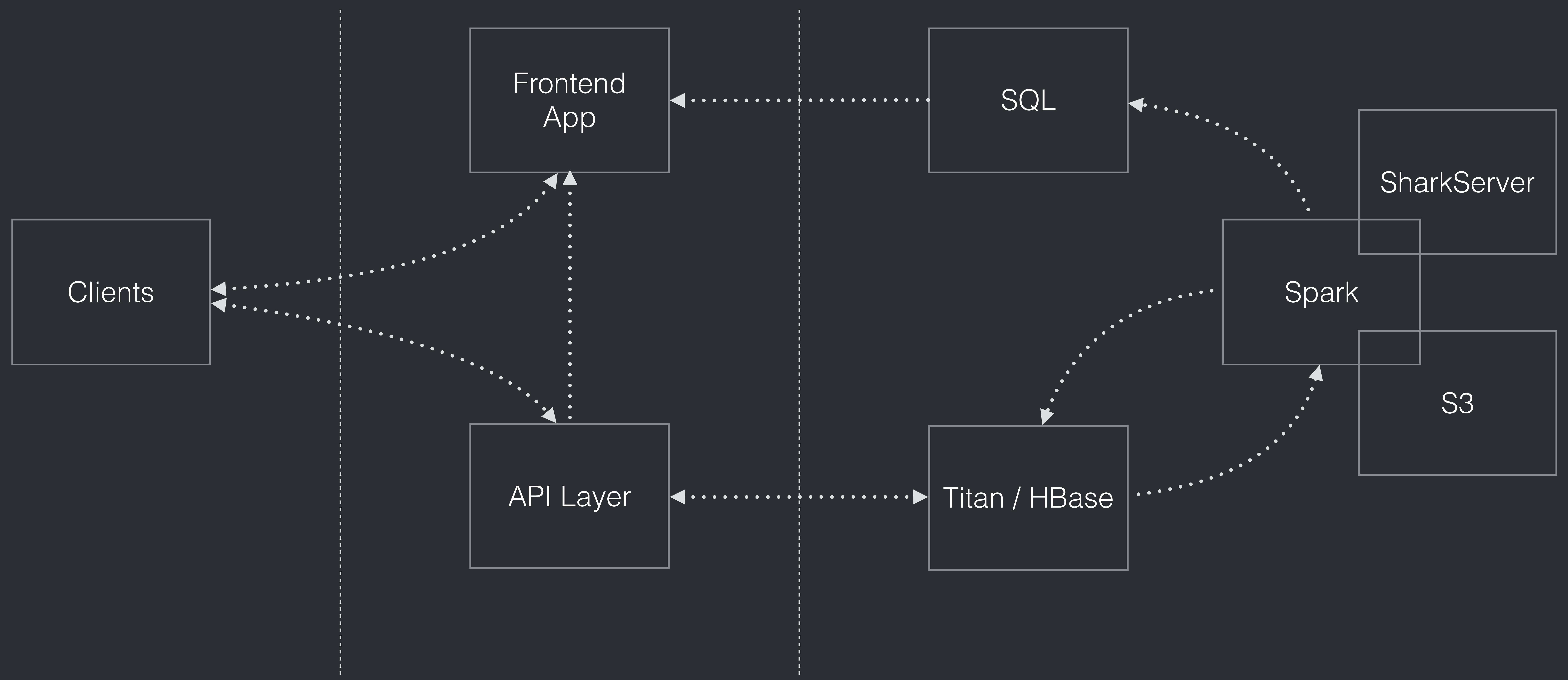
The Graphflow Platform



Why Spark?

- In-memory caching + broadcast variables == machine learning
- Scala + Python APIs + SQL (Shark)
- Spark Streaming
- (more recently)
  - *MLlib*
  - *SparkSQL*

Architecture, Take One



Titan - Spark Integration

- Faunus: Gremlin DSL on Hadoop (in Groovy)

```
g = FaunusFactory.open(conf)
g.E.has(“label”, “purchase”).transform(...)
```

- Spark

```
val graph = sc.newAPIHadoopRDD(
               faunusGraph.getConf,
               fClass = classOf[TitanHBaseInputFormat],
               kClass = classOf[NullWritable],
               vClass = classOf[FaunusVertex])
graph.flatMap { vertex =>
     val edges = vertex.getEdges(Direction.OUT).filter(e => e.getLabel == “purchase”)
     edges.map { edge => (...) }
}
```

Spark Challenges, mid-2013

- Spark 0.7.x
- No MLlib …
  - *… so we wrote our own*
  - *Port of Mahout's Alternating Least Squares using Spark + Breeze*
- No JobServer …
  - *… so we wrote our own*
  - *Scalatra REST API + Akka Actors*
  - *Simpler version of Ooyala's JobServer*

Spark ALS

- Spark 0.8.0-SNAPSHOT
- MovieLens 1m dataset
- Implicit prefs, 10 iterations, 10 factors

Model	Mahout 0.8 ALS	Spark Custom ALS
Runtime	6m39s	58s
Lines of Code	1374 + 137 + 121 + 116 = ~1750	~425

Spark ALS

```java
public class ImplicitFeedbackAlternatingLeastSquaresSolver {

  private final int numFeatures;
  private final double alpha;
  private final double lambda;

  private final OpenIntObjectHashMap<Vector> Y;
  private final Matrix YtransposeY;

  public ImplicitFeedbackAlternatingLeastSquaresSolver(int numFeatures, double lambda, double alpha,
      OpenIntObjectHashMap<Vector> Y) {
    this.numFeatures = numFeatures;
    this.lambda = lambda;
    this.alpha = alpha;
    this.Y = Y;
    YtransposeY = getYtransposeY(Y);
  }

  public Vector solve(Vector ratings) {
    return solve(YtransposeY.plus(getYtransponseCuMinusIYPlusLambdaI(ratings)), getYtransponseCuPu(ratings));
  }

  private static Vector solve(Matrix A, Matrix y) {
    return new QRDecomposition(A).solve(y).viewColumn(0);
  }

  double confidence(double rating) {
    return 1 + alpha * rating;
  }

  /* Y' Y */
  private Matrix getYtransposeY(OpenIntObjectHashMap<Vector> Y) {

    IntArrayList indexes = Y.keys();
    indexes.quickSort();
    int numIndexes = indexes.size();

    double[][] YtY = new double[numFeatures][numFeatures];

    // Compute Y'Y by dot products between the 'columns' of Y
    for (int i = 0; i < numFeatures; i++) {
      for (int j = i; j < numFeatures; j++) {
        double dot = 0;
        for (int k = 0; k < numIndexes; k++) {
          Vector row = Y.get(indexes.getQuick(k));
          dot += row.getQuick(i) * row.getQuick(j);
        }
        YtY[i][j] = dot;
        if (i != j) {
          YtY[j][i] = dot;
        }
      }
    }
    return new DenseMatrix(YtY, true);
  }

  /** Y' (Cu - I) Y + λ I */
  private Matrix getYtransponseCuMinusIYPlusLambdaI(Vector userRatings) {
    Preconditions.checkArgument(userRatings.isSequentialAccess(), "need sequential access to ratings!");

    /* (Cu -I) Y */
    OpenIntObjectHashMap<Vector> CuMinusIY = new OpenIntObjectHashMap<Vector>(userRatings.getNumNondefaultElements());
    for (Element e : userRatings.nonZeroes()) {
      CuMinusIY.put(e.index(), Y.get(e.index()).times(confidence(e.get()) - 1));
    }

    Matrix YtransponseCuMinusIY = new DenseMatrix(numFeatures, numFeatures);

    /* Y' (Cu -I) Y by outer products */
    for (Element e : userRatings.nonZeroes()) {
      for (Vector.Element feature : Y.get(e.index()).all()) {
        Vector partial = CuMinusIY.get(e.index()).times(feature.get());
        YtransponseCuMinusIY.viewRow(feature.index()).assign(partial, Functions.PLUS);
      }
    }

    /* Y' (Cu - I) Y + λ I  add lambda on the diagonal */
    for (int feature = 0; feature < numFeatures; feature++) {
      YtransponseCuMinusIY.setQuick(feature, feature, YtransponseCuMinusIY.getQuick(feature, feature) + lambda);
    }

    return YtransponseCuMinusIY;
  }

  /** Y' Cu p(u) */
  private Matrix getYtransponseCuPu(Vector userRatings) {
    Preconditions.checkArgument(userRatings.isSequentialAccess(), "need sequential access to ratings!");

    Vector YtransponseCuPu = new DenseVector(numFeatures);

    for (Element e : userRatings.nonZeroes()) {
      YtransponseCuPu.assign(Y.get(e.index()).times(confidence(e.get())), Functions.PLUS);
    }

    return columnVectorAsMatrix(YtransponseCuPu);
  }

  private Matrix columnVectorAsMatrix(Vector v) {
    double[][] matrix =  new double[numFeatures][1];
    for (Vector.Element e : v.all()) {
      matrix[e.index()][0] =  e.get();
    }
    return new DenseMatrix(matrix, true);
  }

}
```

VS

```scala
def updateFactorsImplicit(
  UorI: mutable.Map[Int, DenseVector[Double]],
  ratings: Vector[Double],
  YtY: DenseMatrix[Double]) = {

  // set up required intermediate data structures
  val nui = ratings.activeSize
  val UorIMat = DenseMatrix.zeros[Double](nui, numF)
  val CuMinusIY = DenseMatrix.zeros[Double](nui, numF)
  val Cup = DenseVector.zeros[Double](nui)
  var j = 0

  ratings.activeIterator.foreach{ case(i, v) => {
    CuMinusIY(j, ::) := UorI(i) :* alpha :* v
    Cup(j) = alpha * v + 1
    UorIMat(j, ::) := UorI(i)
    j += 1
  }}

  val YtCuY =
    YtY + UorIMat.t * CuMinusIY + (DenseMatrix.eye[Double](numF) :* lambda)
  val YtCup = UorIMat.t * Cup
  YtCuY \ YtCup
}
```

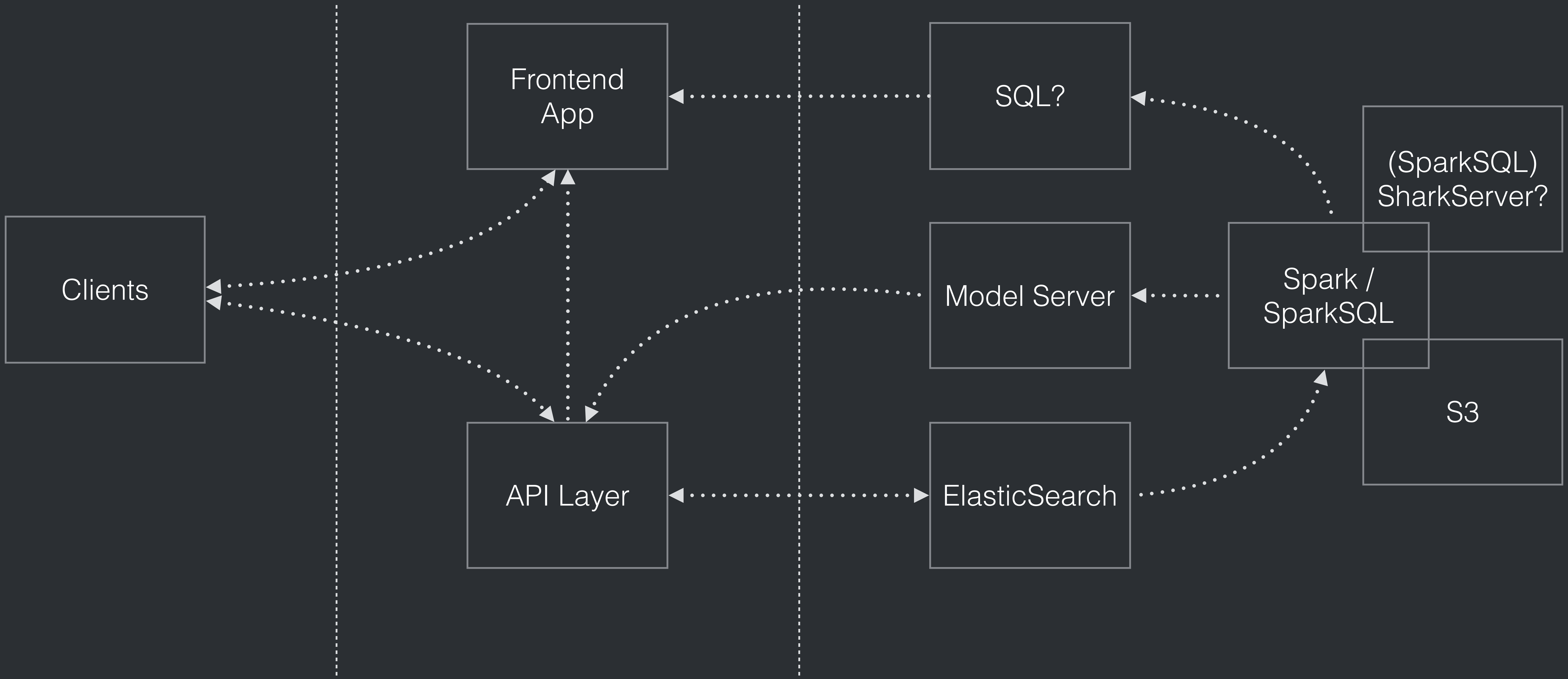
Matrix / vector multiplication

Element-wise operations

Architecture Issues

- Ended up not using Titan’s powerful features
- Search, filtering, aggregation more important
- Titan/HBase + ElasticSearch
  - *Complexity*
  - *Data duplication*
- Always aimed for real-time model serving + incremental updates

Architecture, Take Two



ElasticSearch - Spark Integration

- Faunus / Spark

```
val graph = sc.newAPIHadoopRDD(faunusGraph.getConf,
                               classOf[TitanHBaseInputFormat],
                               classOf[NullWritable],
                               classOf[FaunusVertex])
val events = graph.flatMap { vertex =>
  vertex.getEdges(Direction.OUT).map { edge => (...) }
}
```

- ElasticSearch / Spark

```
conf.set("es.resource", s"$index/$doc")
conf.set("es.query", s"?q=$q")
val esRdd = sc.newAPIHadoopRDD(conf,
                               classOf[EsInputFormat[NullWritable, LinkedMapWritable]],
                               classOf[NullWritable],
                               classOf[LinkedMapWritable])
val events = esRdd.map { case (_, m) => (...) }
```

SparkSQL

- ElasticSearch / SparkSQL

```
val events = esRdd.map { case (_, m) =>
  Event(m("timestamp").toLong, m("event").toString, ...)
}

// complex join and filter in Spark
...

events.registerAsTable("events")
val aggs = hql(
             "select
                from_unixtime(cast(time/1000.0 as bigint), 'yyyy-MM-dd HH:00:00') hour,
                event,
                count(1)
              from events ..."
            )

// save results to S3, ElasticSearch, MySQL, etc…
```

Spark ALS, Take Two

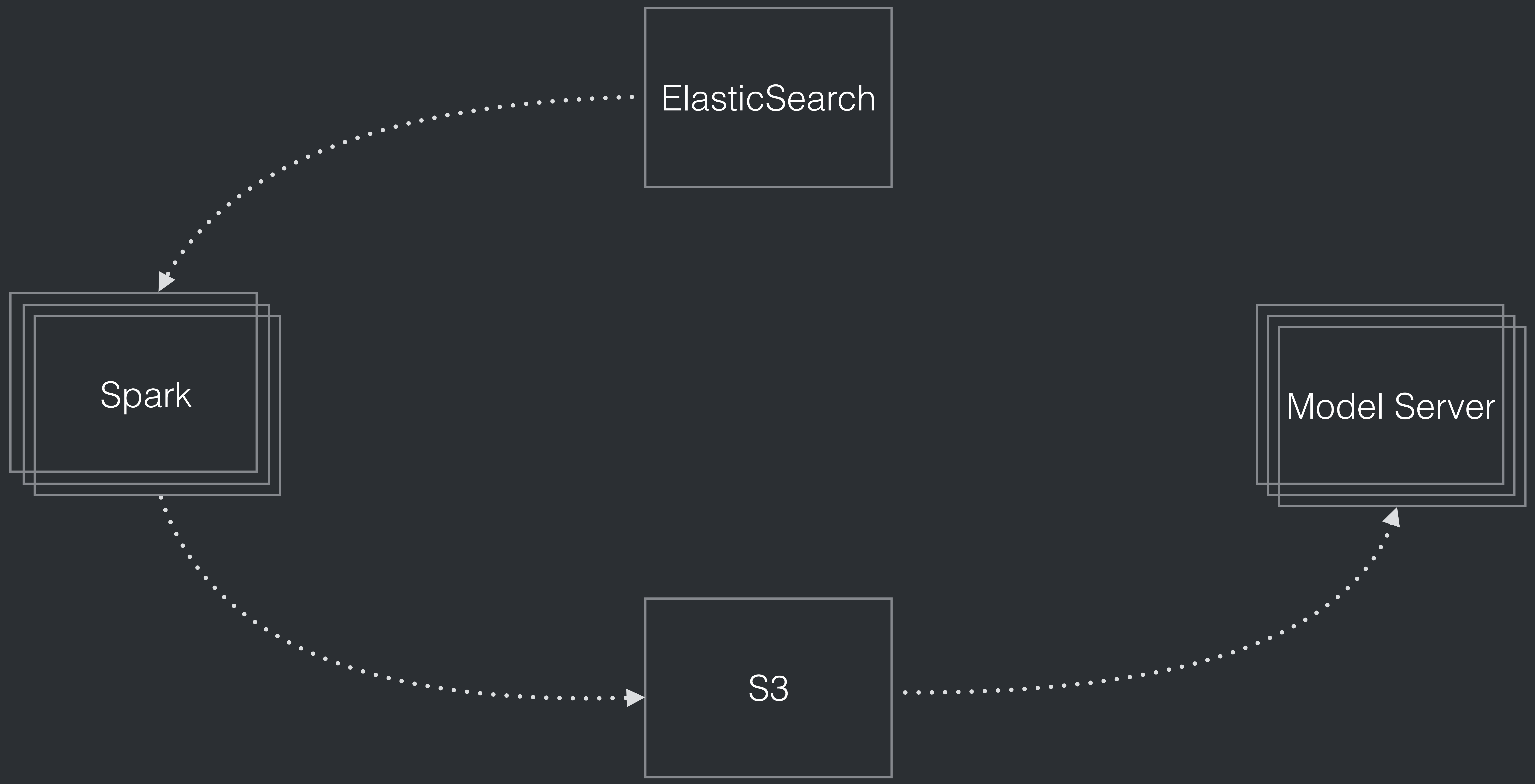
- Spark MLlib
- MovieLens 1m dataset
- Implicit prefs, 10 iterations, 10 factors

Model	Mahout 0.8 ALS	Spark Custom ALS	MLlib ALS
Runtime	6m39s	58s	24.8s
Lines of Code	1374 + 137 + 121 + 116 = ~1750	~425	~880

Model Server

- Requirements:
  - *Serve 100s - 1000s concurrent models*
  - *Fast & horizontally scalable*
  - *Many small models (10s - 100s thousands users/items)*
  - *Some very large models (millions of users/items)*
  - *Fault tolerant*
- Available alternatives didn’t fit: Oryx, Prediction.io

Model Server



Model Server

- Scalatra
- Akka Cluster
- Breeze
- Periodic model refresh
- Incremental updates

Spark Challenges, mid-2014

- Spark 1.0.0!
  - *MLlib*
  - *JobServer*
  - *Spark Submit*
- Main challenges now:
  - *DevOps*
  - *Deployment*
  - *Automation & scheduling*

Future

- Near future:
  - *More complex recommendation models in Spark*
  - *Different types of predictive models*
  - *More complex and compute-intensive aggregations for reporting & insights*
- Farther future:
  - *Spark Streaming for online model updates and aggregations*


Graph Flow