

# Stratosphere

## v0.4

Stephan Ewen  
(stephan.ewen@tu-berlin.de)

# Release Preview

• • •

Official release coming end of November

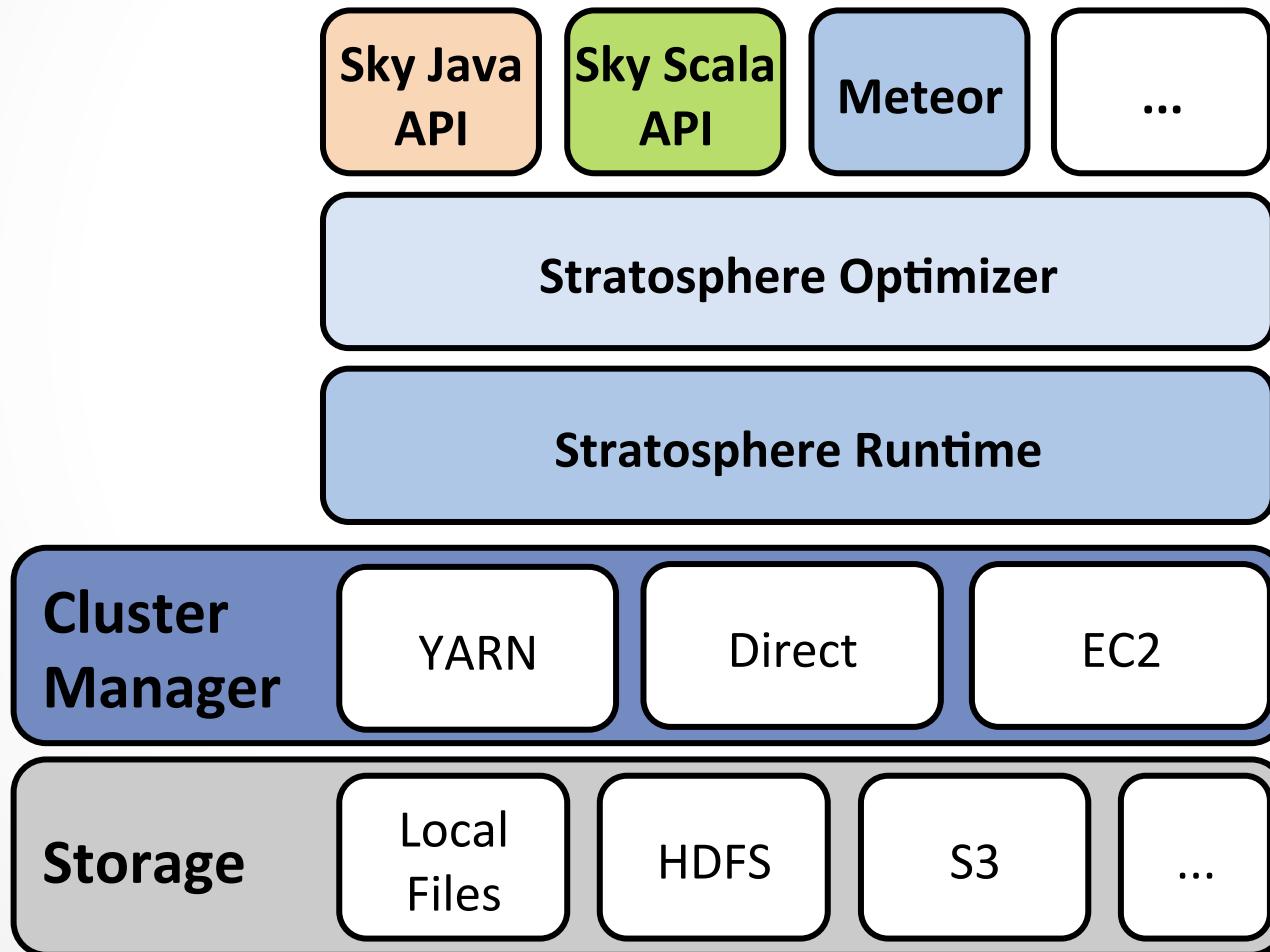
Hands on sessions today with the latest code snapshot

# New Features in a Nutshell



- Declarative Scala Programming API
- Iterative Programs
  - Bulk (batch-to-batch in memory) and Incremental (Delta Updates)
  - Automatic caching and cross-loop optimizations
- Runs on top of YARN (Hadoop Next Gen)
- Various deployment methods
  - VMs, Debian packages, EC2 scripts, ...
- Many usability fixes and of bugfixes

# Stratosphere System Stack

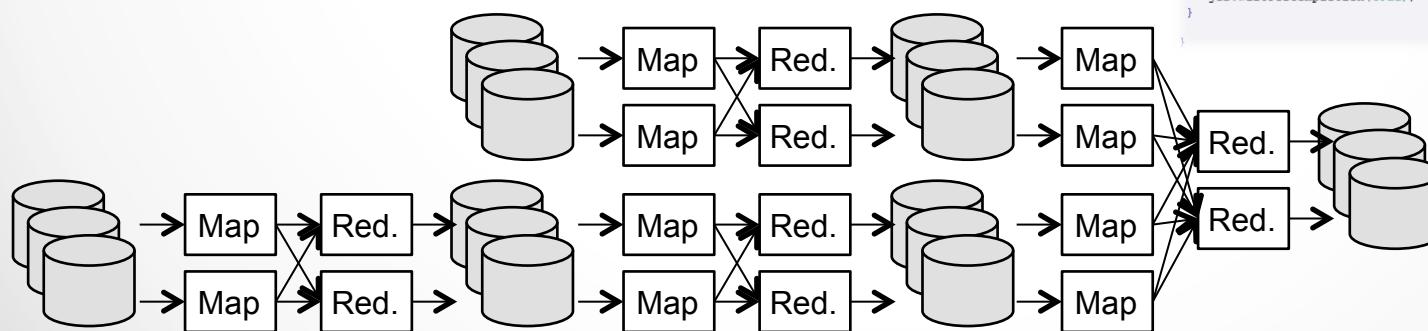


# MapReduce

## It is nice and good, but...

Very verbose and low level. Only usable by system programmers.

Everything slightly more complex must result in a cascade of jobs. Loses performance and optimization potential.



```
public class WordCount {  
    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {  
        private final static IntWritable one = new IntWritable(1);  
        private Text word = new Text();  
  
        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {  
            StringTokenizer tokenizer = new StringTokenizer(value.toString());  
            while (tokenizer.hasMoreTokens()) {  
                word.set(tokenizer.nextToken());  
                context.write(word, one);  
            }  
        }  
    }  
  
    public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {  
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {  
            int sum = 0;  
            for (IntWritable val : values) {  
                sum += val.get();  
            }  
            context.write(key, new IntWritable(sum));  
        }  
    }  
  
    public static void main(String[] args) throws Exception {  
        Configuration conf = new Configuration();  
  
        Job job = new Job(conf, "wordcount");  
  
        job.setOutputKeyClass(Text.class);  
        job.setOutputValueClass(IntWritable.class);  
  
        job.setMapperClass(Map.class);  
        job.setReducerClass(Reduce.class);  
  
        job.setInputFormatClass(TextInputFormat.class);  
        job.setOutputFormatClass(TextOutputFormat.class);  
  
        TextInputFormat.addInputPath(job, new Path(args[0]));  
        TextOutputFormat.setOutputPath(job, new Path(args[1]));  
  
        job.waitForCompletion(true);  
    }  
}
```

# SQL (or Hive or Pig) It is nice and good, but...

- Allow you to do a subset of the tasks efficiently and elegantly
- What about the cases that do not fit SQL?
  - Custom types
  - Custom non-relational functions (they occur a lot!)
  - Iterative Algorithms → Machine learning, graph analysis
- How does it look to mix SQL with MapReduce?

# SQL (or Hive or Pig) is nice and good, but...

- *Program Fragmentation*
- *Impedance Mismatch*
- *Breaks optimization*

```
FROM (
    FROM pv_users
    MAP pv_users.userid, pv_users.date
    USING 'map_script'
    AS dt, uid
    CLUSTER BY dt) map_output
INSERT OVERWRITE TABLE pv_users_reduced
REDUCE map_output.dt, map_output.uid
USING 'reduce_script'
AS date, count;
```

```
A = load 'WordcountInput.txt';
B = MAPREDUCE wordcount.jar store A into 'inputDir' load
    'outputDir' as (word:chararray, count: int)
    'org.myorg.WordCount inputDir outputDir';
C = sort B by count;
```

Hive

Pig

# Sky Language

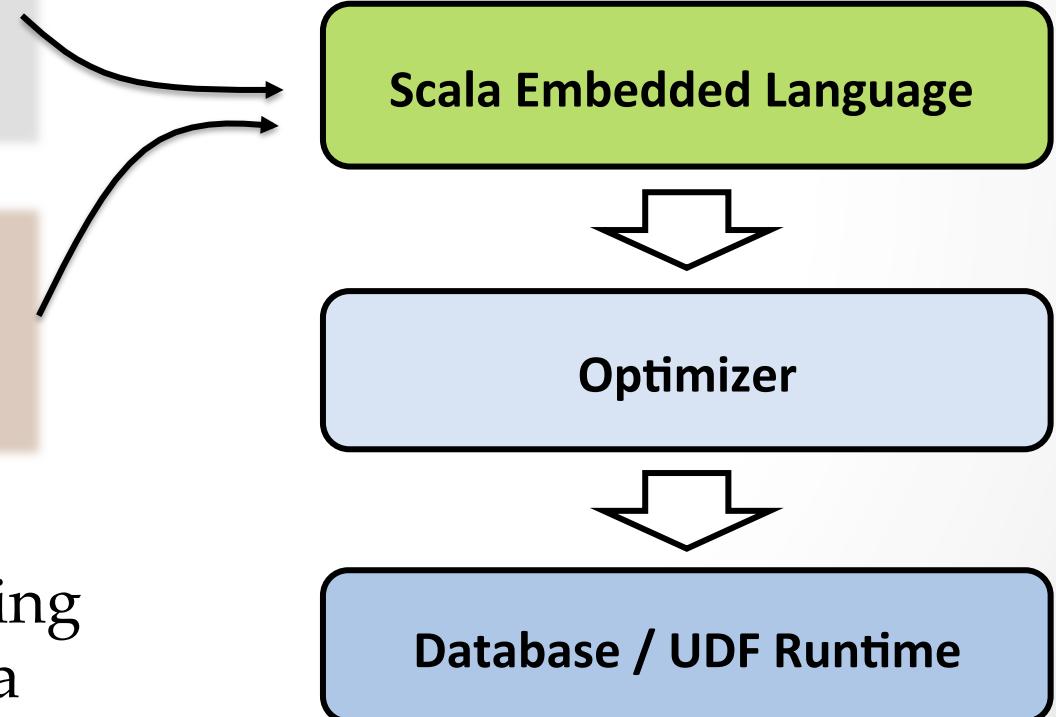
MapReduce style functions

*(Map, Reduce, Join, CoGroup,  
Cross, ...)*

Relational Set Operations

*(filter, map, group, join,  
aggregate, ...)*

Write like a programming  
language, execute like a  
database...



# Sky Language

Add a bit of  
"languages and compilers"  
sauce to the database  
stack



940697 [RF] © www.visualphotos.com

# Scala API by Example

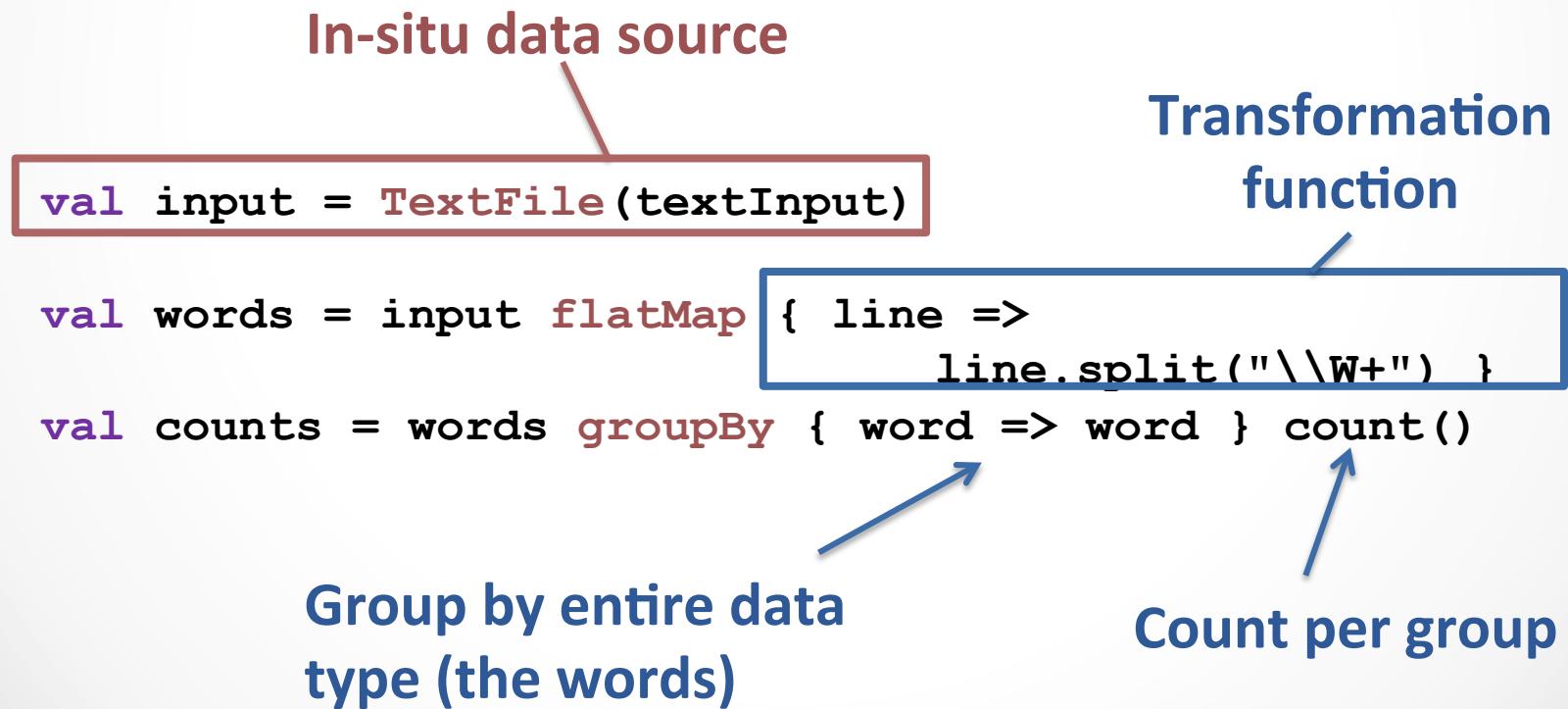
- The classical word count example

```
val input = TextFile(textInput)

val words = input flatMap { line =>
    line.split("\\w+")
}
val counts = words groupBy { word => word } count()
```

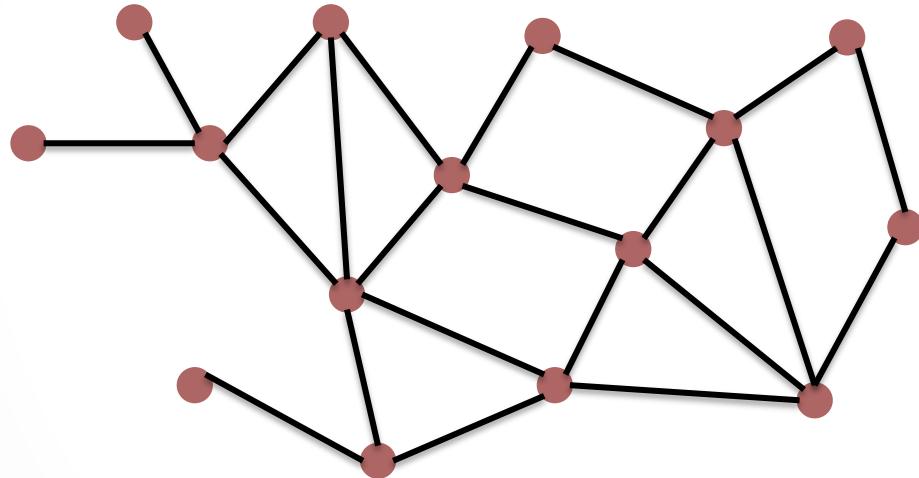
# Scala API by Example

- The classical word count example



# Scala API by Example

- Graph Triangles (Friend-of-a-Friend problem)
  - Recommending friends, finding important connections



- 1) Enumerate candidate triads
- 2) Close as triangles

# Scala API by Example

```
case class Edge(from: Int, to: Int)
case class Triangle(apex: Int, base1: Int, base2: Int)

val vertices = DataSource("hdfs://...", CsvFormat[Edge])

val byDegree = vertices map { projectToLowerDegree }

val byID = byDegree map { (x) => if (x.from < x.to) x
                           else Edge(x.to, x.from) }

val triads = byDegree groupBy { _.from } reduceGroup { buildTriads }

val triangles = triads join byID
                  where { t => (t.base1, t.base2) }
                  isEqualTo { e => (e.from, e.to) }
                  map { (triangle, edge) => triangle }
```

# Scala API by Example

## Custom Data Types

```
case class Edge(from: Int, to: Int)
case class Triangle(apex: Int, base1: Int, base2: Int)
```

## In-situ data source

```
val vertices = DataSource("hdfs://...", CsvFormat[Edge])
```

```
val byDegree = vertices map { projectToLowerDegree }
```

```
val byID = byDegree map { (x) => if (x.from < x.to) x
                           else Edge(x.to, x.from) }
```

```
val triads = byDegree groupBy { _.from } reduceGroup { buildTriads }
```

```
val triangles = triads join byID
                  where { t => (t.base1, t.base2) }
                  isEqualTo { e => (e.from, e.to) }
                  map { (triangle, edge) => triangle }
```

# Scala API by Example

```
case class Edge(from: Int, to: Int)
case class Triangle(apex: Int, base1: Int, base2: Int)

val vertices = DataSource("hdfs://...", CsvFormat[Edge])

val byDegree = vertices map { projectToLowerDegree }

val byID = byDegree map { (x) => if (x.from < x.to) x
                           else Edge(x.to, x.from) }

val triads = byDegree groupBy { _.from } reduceGroup { buildTriads }

val triangles = triads join byID
  where { t => (t.base1, t.base2) }
  isEqualTo { e => (e.from, e.to) }
  map { (triangle, edge) => triangle }
```

**Non-relational library function**

**Non-relational function**

**Relational Join**

# Scala API by Example

```
case class Edge(from: Int, to: Int)
case class Triangle(apex: Int, base1: Int, base2: Int)

val vertices = DataSource("hdfs://...", CsvFormat[Edge])

val byDegree = vertices map { projectToLowerDegree }

val byID = byDegree map { (x) => if (x.from < x.to) x
                           else Edge(x.to, x.from) }

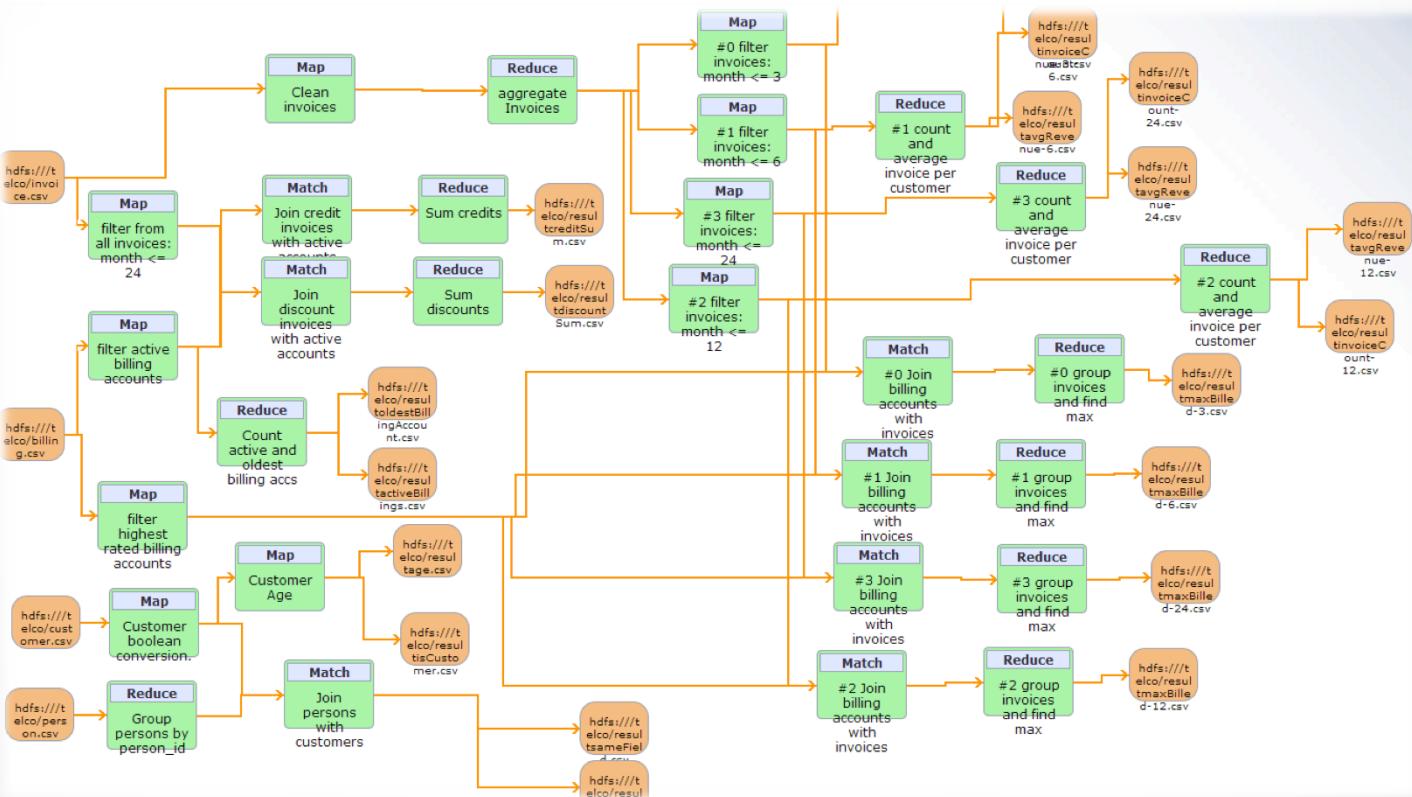
val triads = byDegree groupBy { _.from } reduceGroup { buildTriads }

val triangles = triads join byID
  where { t => (t.base1, t.base2) }
  isEqualTo { e => (e.from, e.to) }
  map { (triangle, edge) => triangle }
```

Key References

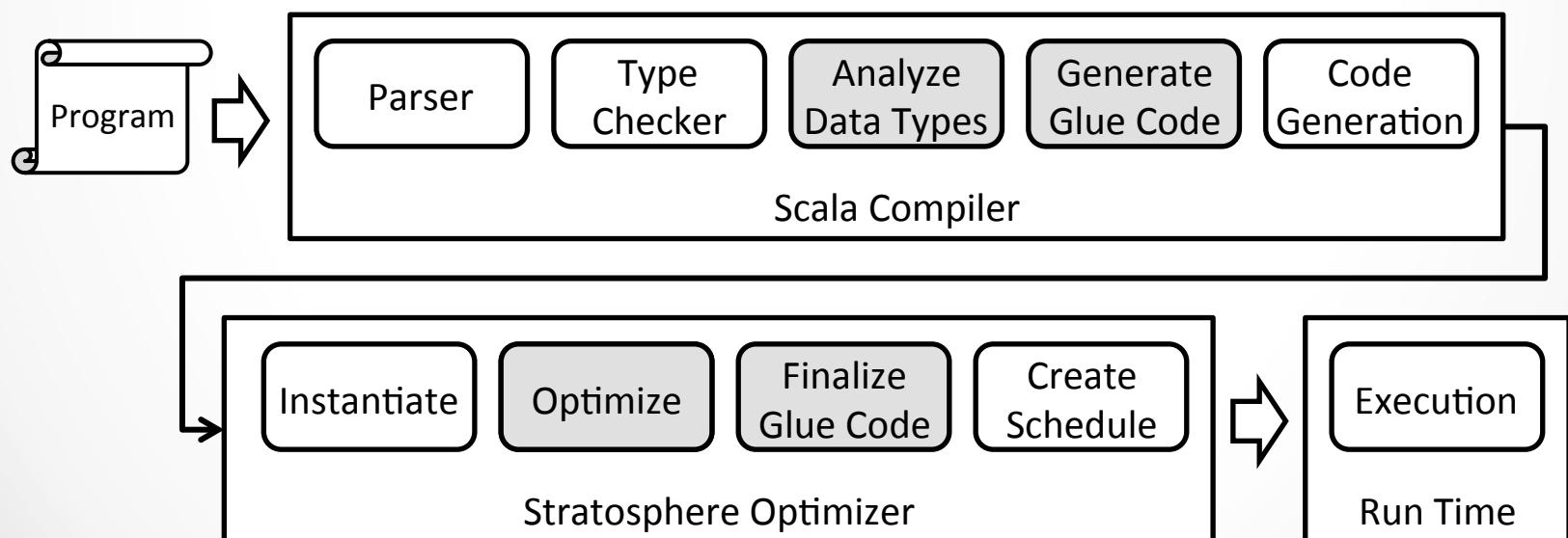
# Analysis in the Real World...

Visualization of an analysis program from a partner use-case



# Optimizing Programs

- Program optimization happens in two phases
  1. Data type and function code analysis inside the Scala Compiler
  2. Relational-style optimization of the data flow



# Type Analysis

- Types and Key Selectors are mapped to flat schema
- Generated code for interaction with runtime

*Primitive Types,  
Arrays, Lists*

```
Int, Double,  
Array[String], .  
..
```



*Single Value*

*Tuples /  
Classes*

```
(a: Int, b: Int, c: String)  
class T(x: Int, y: Long)
```



*Tuples*

```
(a: Int, b: Int, c: String)  
(x: Int, y: Long)
```

*Nested  
Types*

```
class T(x: Int, y: Long)  
class R(id: String, value: T)
```

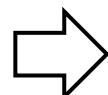


*Recursively  
flattened*

```
(x: Int, y: Long)  
(id:String, x:Int, y:Long)
```

*recursive  
types*

```
class Node(id: Int, left: Node,  
          right: Node)
```

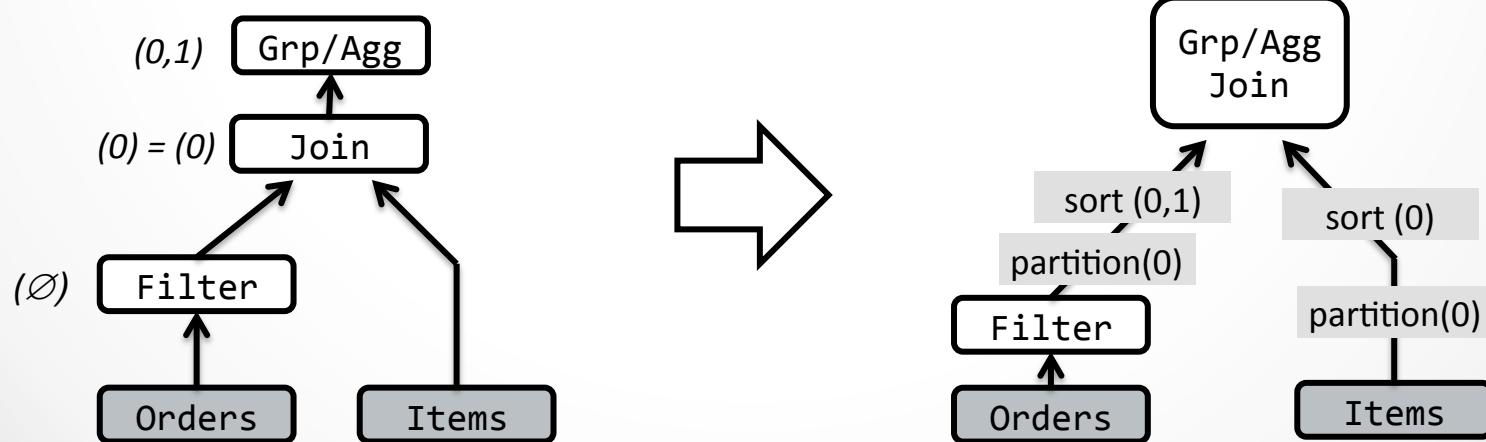


*Tuples  
(w/ BLOB for  
recursion)*

```
(id:Int, left:BLOB,  
right:BLOB)
```

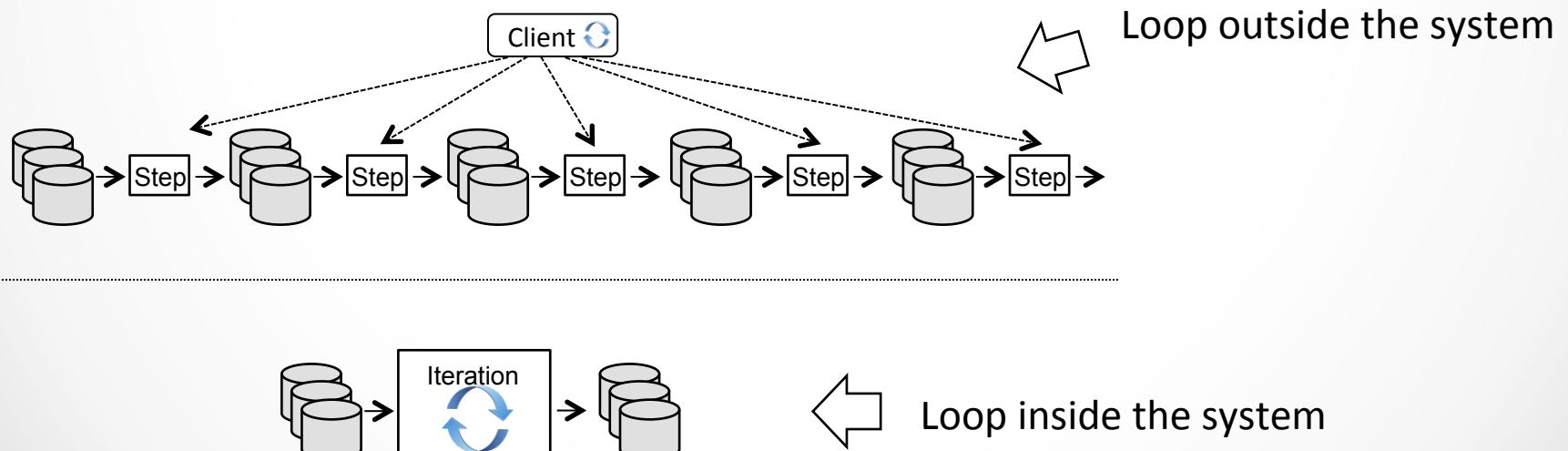
# Optimization

```
case class Order(id: Int, priority: Int, ...)  
case class Item(id: Int, price: Double, )  
case class PricedOrder(id, priority, price)  
  
val orders = DataSource(...)  
val items = DataSource(...)  
  
val filtered = orders filter { ... }  
  
val prio = filtered join items where { _.id } isEqualTo { _.id }  
    map { (o, li) => PricedOrder(o.id, o.priority, li.price) }  
  
val sales = prio groupBy { p => (p.id, p.priority) } aggregate {_.price}, SUM
```



# Iterative Programs

- Many programs have a loop and make multiple passes over the data
  - Machine Learning algorithms iteratively refine the model
  - Graph algorithms propagate information one hop by hop



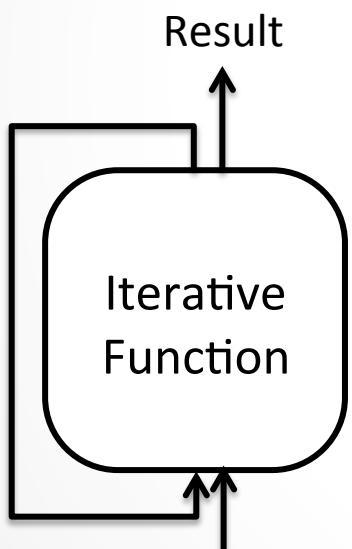
# Why Iterations

- Algorithms that need iterations
  - K-Means
  - Gradient descent
  - Page-Rank
  - Logistic Regression
  - Path algorithms on graphs
  - Graph communities / dense sub-components
  - Inference (believe propagation)
  - ...

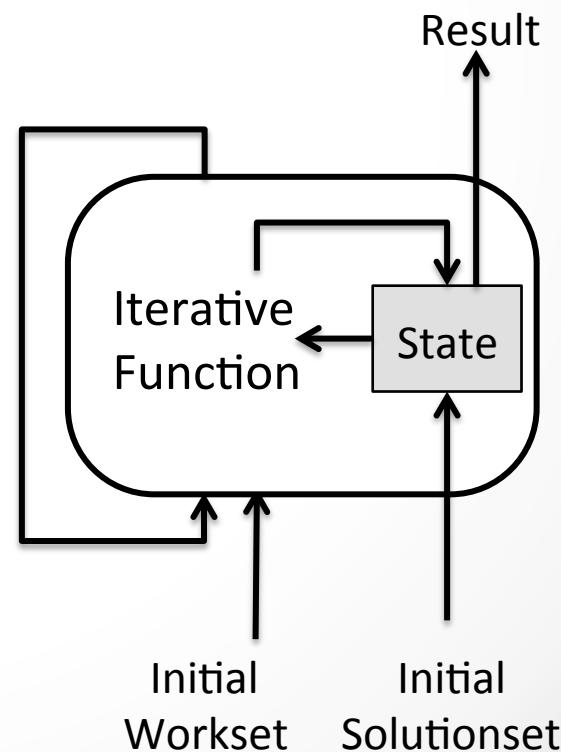
→ All the hot algorithms for building predictive models

# Two Types of Iterations

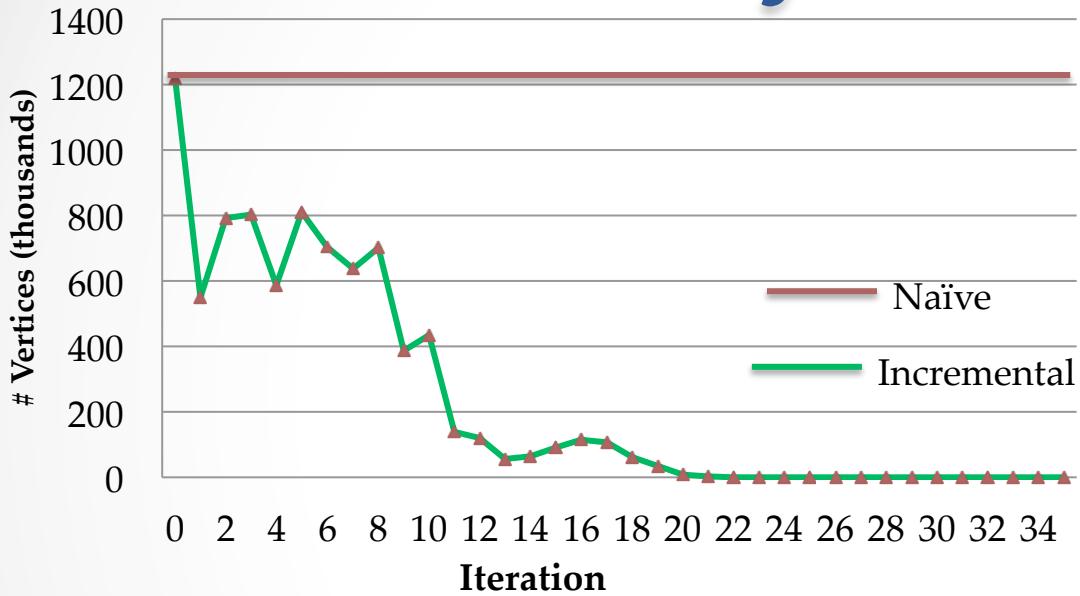
Bulk Iterations



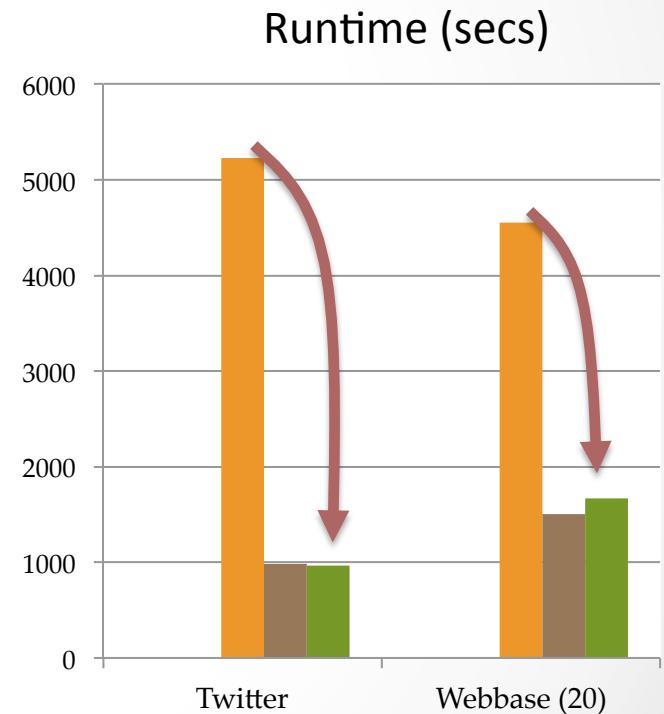
Incremental Iterations  
(aka. Workset Iterations)



# Iterations inside the System



Computations performed in each iteration for connected communities of a social graph



# Iterative Program (Scala)

```
def step = (s: DataSet[Vertex], ws: DataSet[Vertex]) => {  
  
    val min = ws groupBy {_.id} reduceGroup { x => x.minBy { _.component } }  
  
    val delta = s join minNeighbor where { _.id } isEqualTo { _.id }  
        flatMap { (c,o) => if (c.component < o.component)  
            Some(c) else None }  
  
    val nextWs = delta join edges where {v => v.id} isEqualTo {e => e.from}  
        map { (v, e) => Vertex(e.to, v.component) }  
  
    (delta, nextWs)  
}  
  
val components = vertices.iterateWithWorkset(initialWorkset, {_.id}, step)
```

# Iterative Program (Scala)

## Define Step function

```
def step = (s: DataSet[Vertex], ws: DataSet[Vertex]) => {  
  
    val min = ws groupBy {_.id} reduceGroup { x => x.minBy { _.component } }  
  
    val delta = s join minNeighbor where { _.id } isEqualTo { _.id }  
        flatMap { (c,o) => if (c.component < o.component)  
            Some(c) else None }  
  
    val nextWs = delta join edges where {v => v.id} isEqualTo {e => e.from}  
        map { (v, e) => Vertex(e.to, v.component) }  
  
    (delta, nextWs)  
}  
  
val components = vertices.iterateWithWorkset(initialWorkset, {_.id}, step)
```

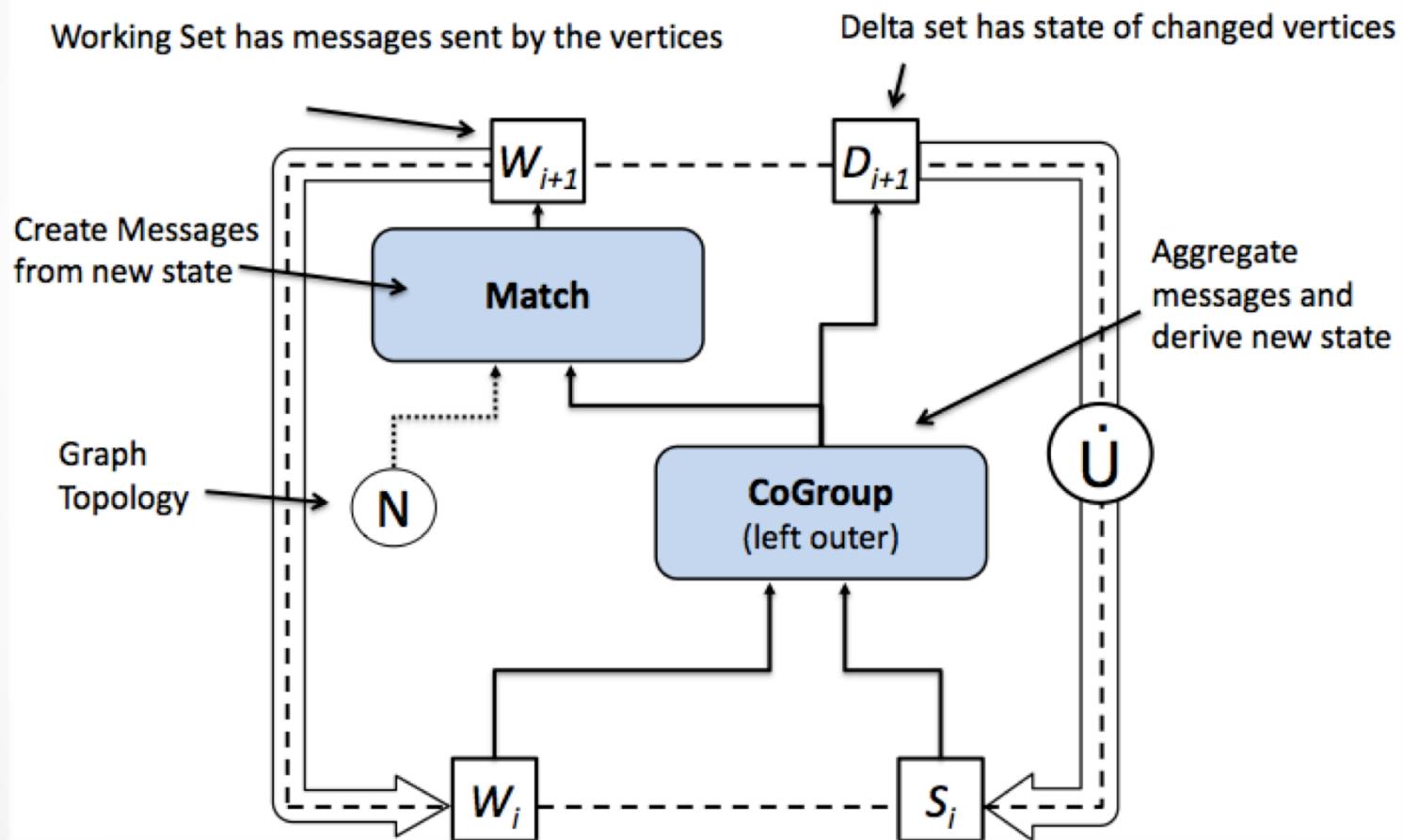
Return Delta and  
next Workset

Invoke Iteration

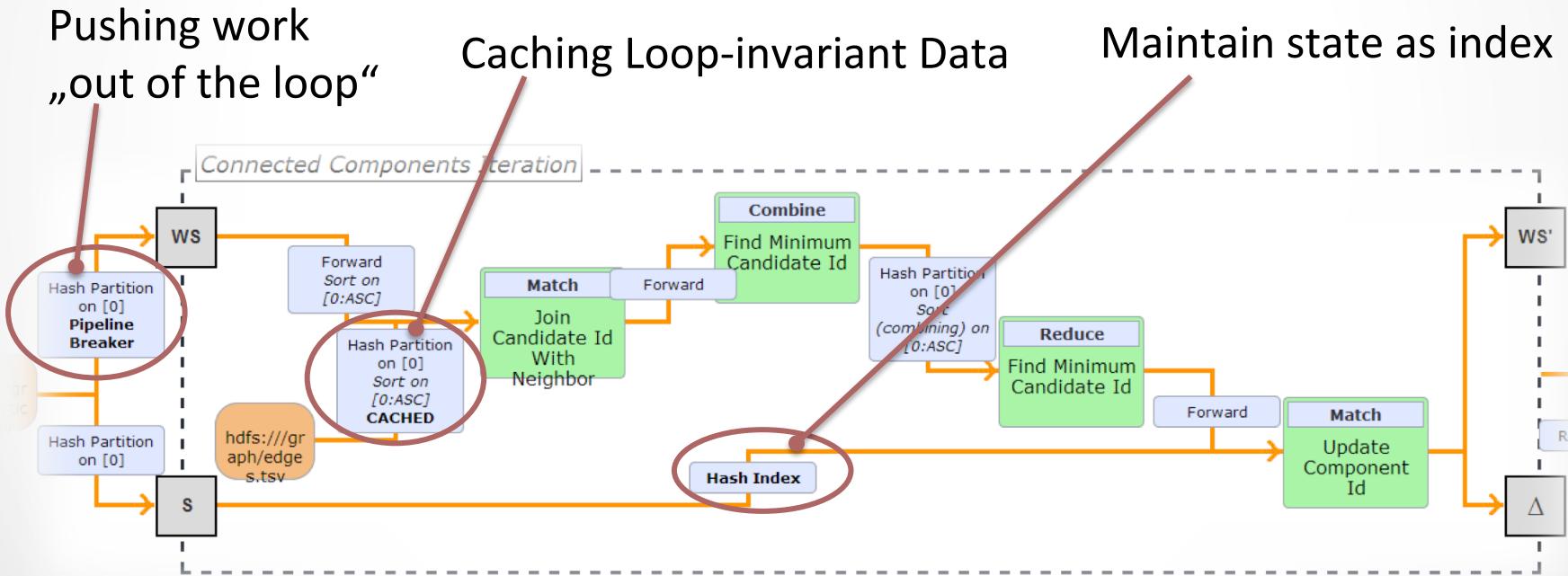
# Iterative Program (Java)

```
WorksetIteration iteration = new WorksetIteration(0, "Connected Components Iteration");  
iteration.setInitialSolutionSet(initialVertices);  
iteration.setInitialWorkset(initialVertices);  
iteration.setMaximumNumberOfIterations(maxIterations);  
  
// create DataSourceContract for the edges  
FileDataSource edges = new FileDataSource(LongLongInputFormat.class, edgeInput, "Edges");  
  
// create CrossContract for distance computation  
MatchContract joinWithNeighbors = MatchContract.builder(NeighborWithComponentIDJoin.class, PactLong.class, 0, 0)  
    .input1(iteration.getWorkset())  
    .input2(edges).build();  
  
// create ReduceContract for finding the nearest cluster centers  
ReduceContract minCandidateId = ReduceContract.builder(MinimumComponentIDReduce.class, PactLong.class, 0)  
    .input(joinWithNeighbors).build();  
  
// create CrossContract for distance computation  
MatchContract updateComponentId = MatchContract.builder(UpdateComponentIdMatch.class, PactLong.class, 0, 0)  
    .input1(minCandidateId)  
    .input2(iteration.getSolutionSet()).build();  
  
iteration.setNextWorkset(updateComponentId);  
iteration.setSolutionSetDelta(updateComponentId);
```

# Graph Processing in Stratosphere

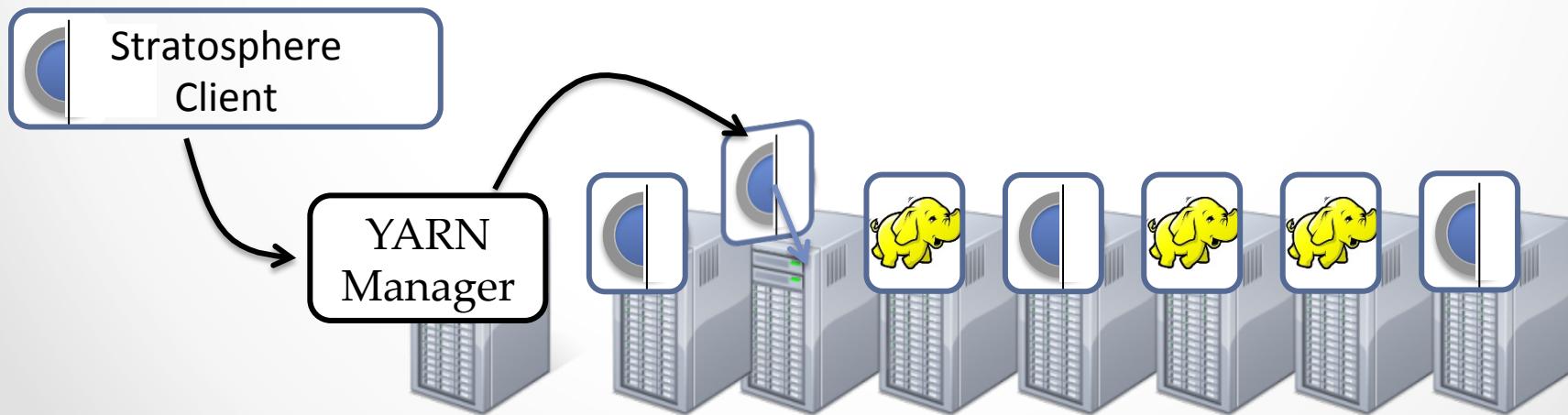


# Optimizing Iterative Programs



# Support for YARN

- Clusters are typically shared between applications
  - Different users
  - Different systems, or different versions of the same system
- YARN manages cluster as a collection of resources
  - Allows systems to deploy themselves on the cluster for a task



# Be Part of a Great Open Source Project

- Use Stratosphere & give us feedback on the experience
- Partner with us and become a pilot user/customer
- Contribute to the system

Project: <http://stratosphere.eu>

Dev: <http://github.com/stratosphere>

Tweet: #StratoSummit