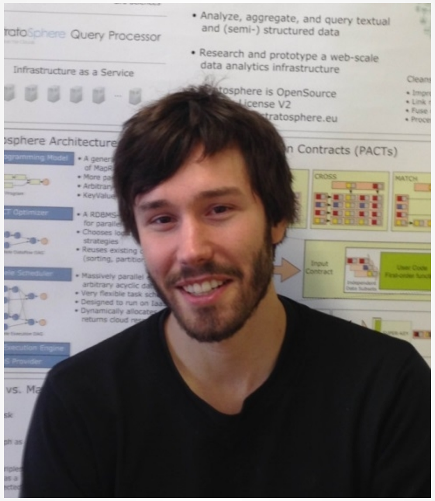
Stratosphere / Flink

Next-Gen Data Analytics Platform

Berlin Buzzwords, May 26th

Stephan Ewen
sewen@apache.org / @stratosphere_eu

About me


Query Processor
Infrastructure as a Service
• Analyze, aggregate, and query textual and (semi-) structured data
• Research and prototype a web-scale data analytics infrastructure
Contracts (PACTs)
CROSS
MATCH
Input Contract

Stephan Ewen
Last days Ph.D. student at TU Berlin
Stratosphere core developer

Big Data looks tiny from
Stratosphere

Apache Flink

What is Stratosphere?

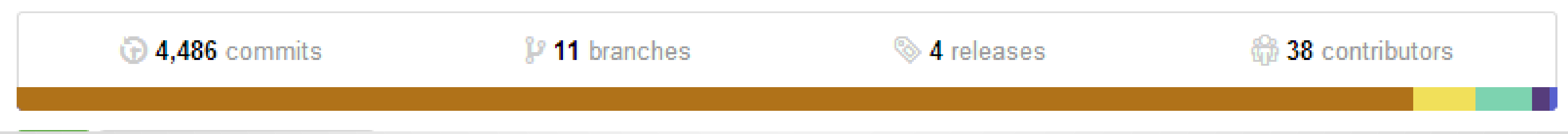
An efficient distributed general-purpose data analysis platform.

Built on top of HDFS and YARN.

Focusing on ease of programming.

Project status

- Research project started in 2009 by TU Berlin, HU Berlin, HPI
- Now a growing open source project with first industrial installations
- Moving to Apache as "Flink"
- v0.4 - stable & documented, v0.5 release candidate 2 out



4,486 commits

11 branches

4 releases

38 contributors

Introducing Stratosphere

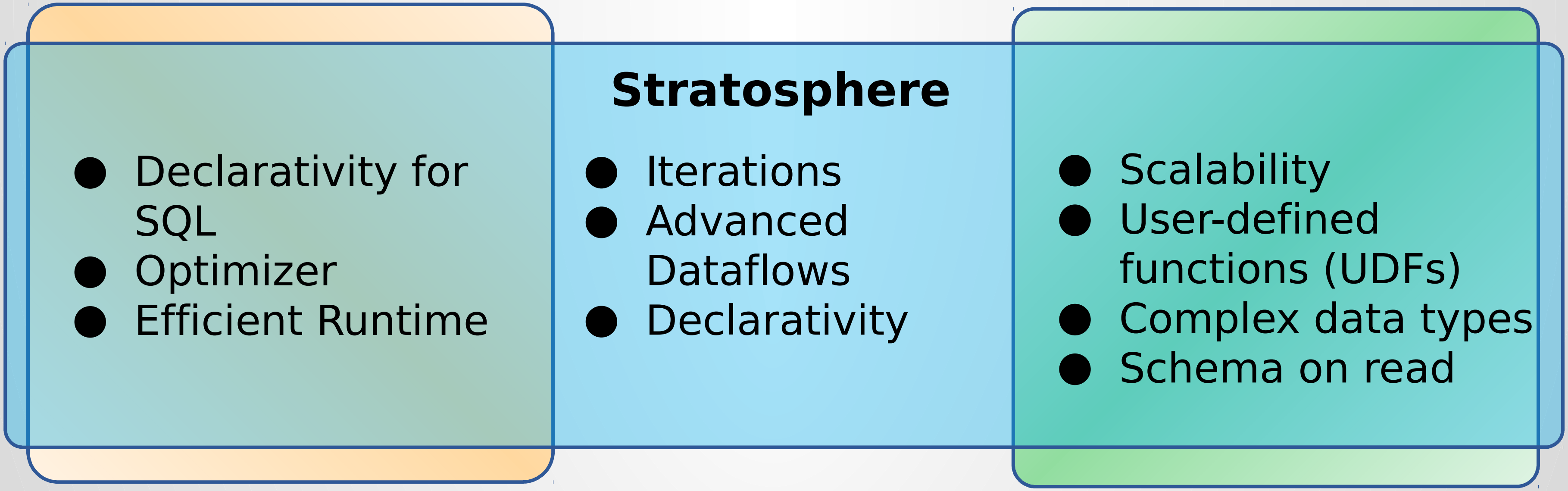
General Purpose Data Analytics Platform.

Database Technology

APACHE DRILL TERADATA

MapReduce-style Technology

hadoop Spark



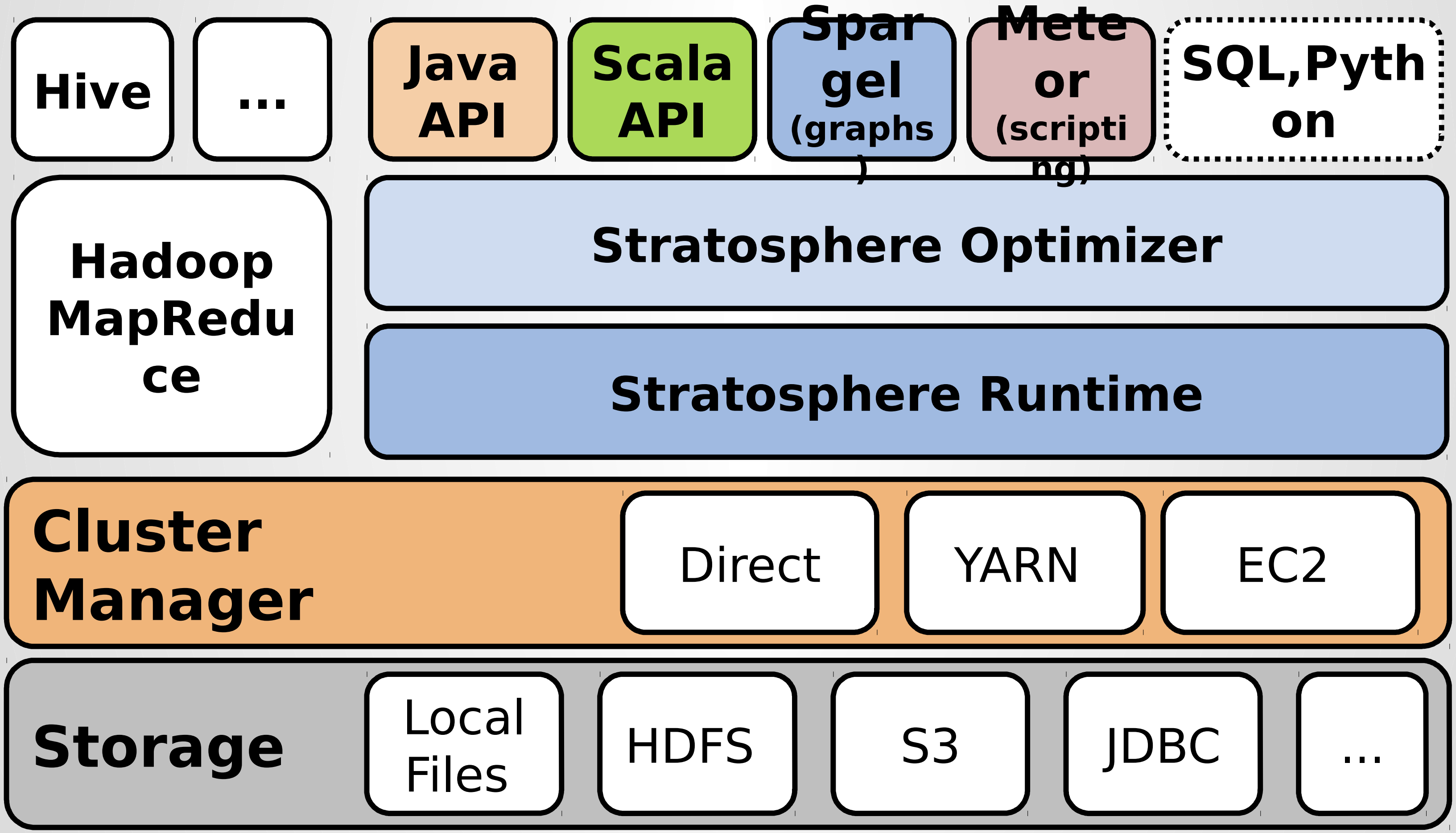
- Declarativity for SQL
- Optimizer
- Efficient Runtime

Stratosphere

- Iterations
- Advanced Dataflows
- Declarativity

- Scalability
- User-defined functions (UDFs)
- Complex data types
- Schema on read

Stratosphere Stack



Hive

...

Java API

Scala API

Spargel (graphs)

Meteor (scripting)

SQL,Python

Hadoop MapReduce

Stratosphere Optimizer

Stratosphere Runtime

Cluster Manager

Direct

YARN

EC2

Storage

Local Files

HDFS

S3

JDBC

...

Key Features

Easy to use developer APIs

- Java, Scala, Graphs, Nested Data
  (Python & SQL under development)
- Flexible composition of large programs

Automatic Optimization

- Join algorithms
- Operator chaining
- Reusing partitioning/sorting

High Performance Runtime

Complex DAGs of operators

In memory & out-of-core

Data streamed between operations

Native Iterations

- Embedded in the APIs
- Data streaming / in-memory
- Delta iterations speed up many programs by orders of mag.

Stratosphere Features

Concise & rich APIs

Word Count in Stratosphere, new Java API

```
DataSet<String> text = env.readTextFile(input);

DataSet<Tuple2<String, Integer>> result = text
                  .flatMap(new Splitter())
                  .groupBy(0).aggregate(SUM, 1);


// map function implementation
class Splitter extends FlatMap<String, Tuple2<String, Integer>> {

    public void flatMap(String value, Collector out){
        for (String token : value.split("\\W")) {
            out.collect(new Tuple2<String, Integer>(token, 1));
        }
    }
}
```

Can use regular POJOs!

Concise & rich APIs

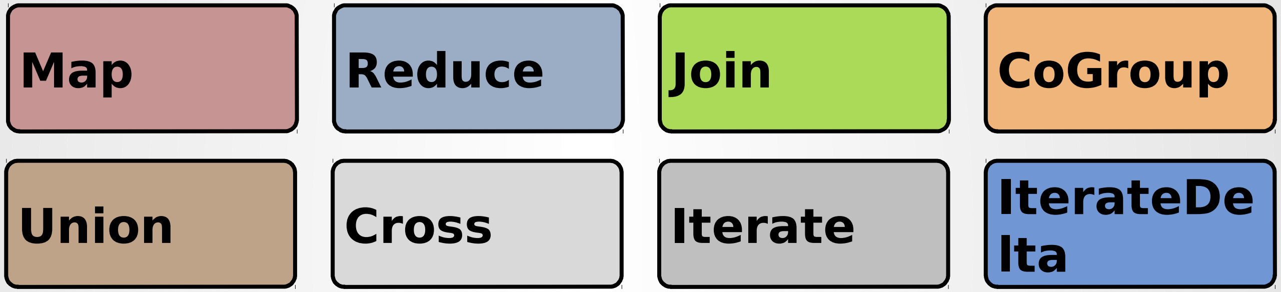
Word Count in Stratosphere

Scala API

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

Concise & rich APIs

Basic Operators



Map

Reduce

Join

CoGroup

Union

Cross

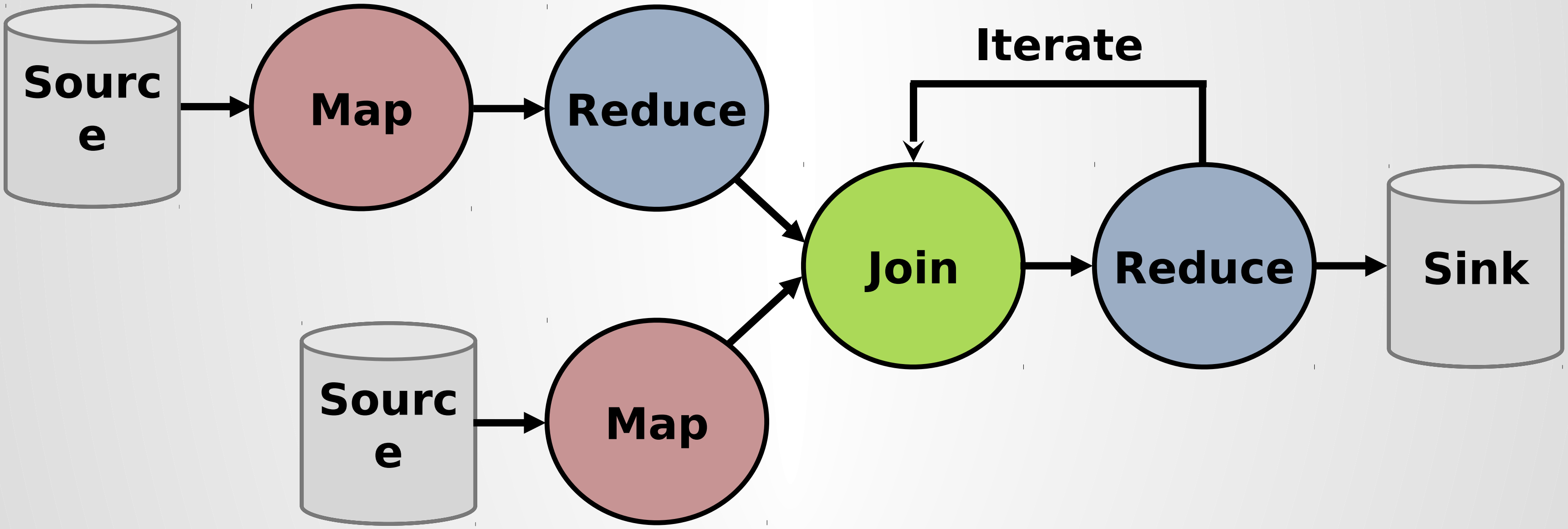
Iterate

IterateDelta

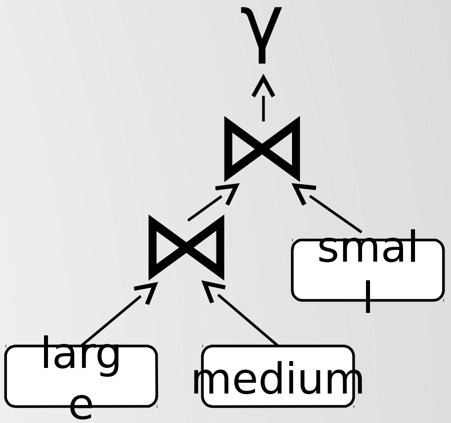
Derived Operators

- Filter, FlatMap, Project
- Aggregate, Distinct
- Outer-Join, Semi-Join, Anti-Join
- Vertex-Centric Graphs computation (Pregel style)
- ....

Flexible Data Pipelines



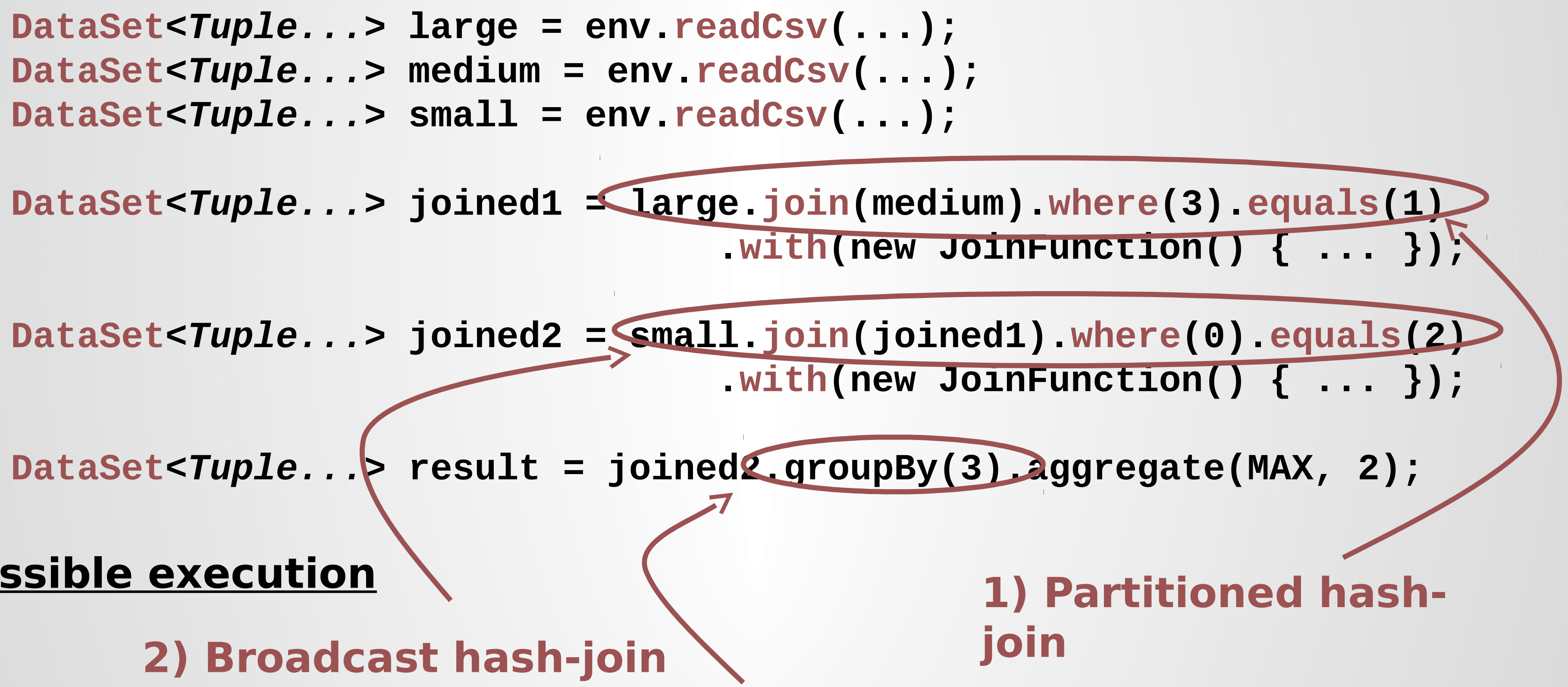
Joins in Stratosphere



```
DataSet<Tuple...> large = env.readCsv(...);
DataSet<Tuple...> medium = env.readCsv(...);
DataSet<Tuple...> small = env.readCsv(...);

DataSet<Tuple...> joined1 = large.join(medium).where(3).equals(1)
                                   .with(new JoinFunction() { ... });

DataSet<Tuple...> joined2 = small.join(joined1).where(0).equals(2)
                                   .with(new JoinFunction() { ... });

DataSet<Tuple...> result = joined2.groupBy(3).aggregate(MAX, 2);
```

ilt-in strategies include *partitioned* join and *replicated* join w
cal *sort-merge* or *hybrid-hash* algorithms.

Automatic Optimization



```
DataSet<Tuple...> large = env.readCsv(...);
DataSet<Tuple...> medium = env.readCsv(...);
DataSet<Tuple...> small = env.readCsv(...);

DataSet<Tuple...> joined1 = large.join(medium).where(3).equals(1)
                                 .with(new JoinFunction() { ... });

DataSet<Tuple...> joined2 = small.join(joined1).where(0).equals(2)
                                 .with(new JoinFunction() { ... });

DataSet<Tuple...> result = joined2.groupBy(3).aggregate(MAX, 2);
```

Possible execution

2) Broadcast hash-join

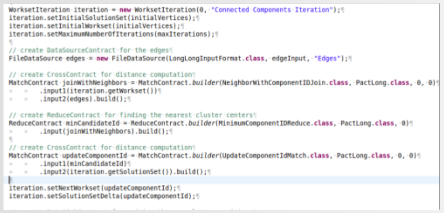
1) Partitioned hash-join

3) Grouping /Aggregation reuses the partitioning from step (1) ⇒ No shuffle!!!

Partitioned ≈ Reduce-side
Broadcast ≈ Map-side

Running Programs

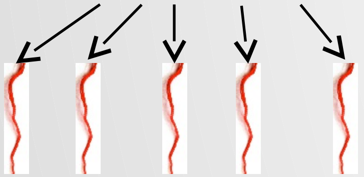
Local Environment



```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);
```

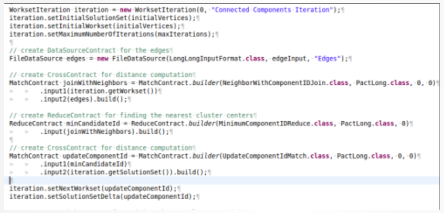
`LocalEnvironment.execute()`

Spawn embedded multi-threaded environment



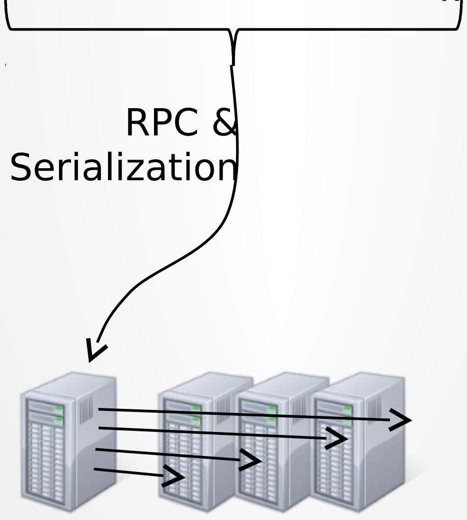
JVM

Remote Environment



```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);
```

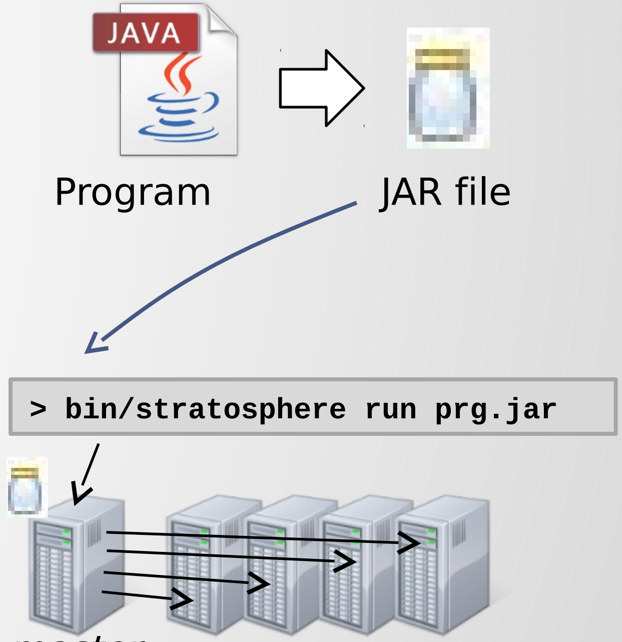
`RemoteEnvironment.execute()`



RPC & Serialization

master

Packaged Programs



Program

JAR file

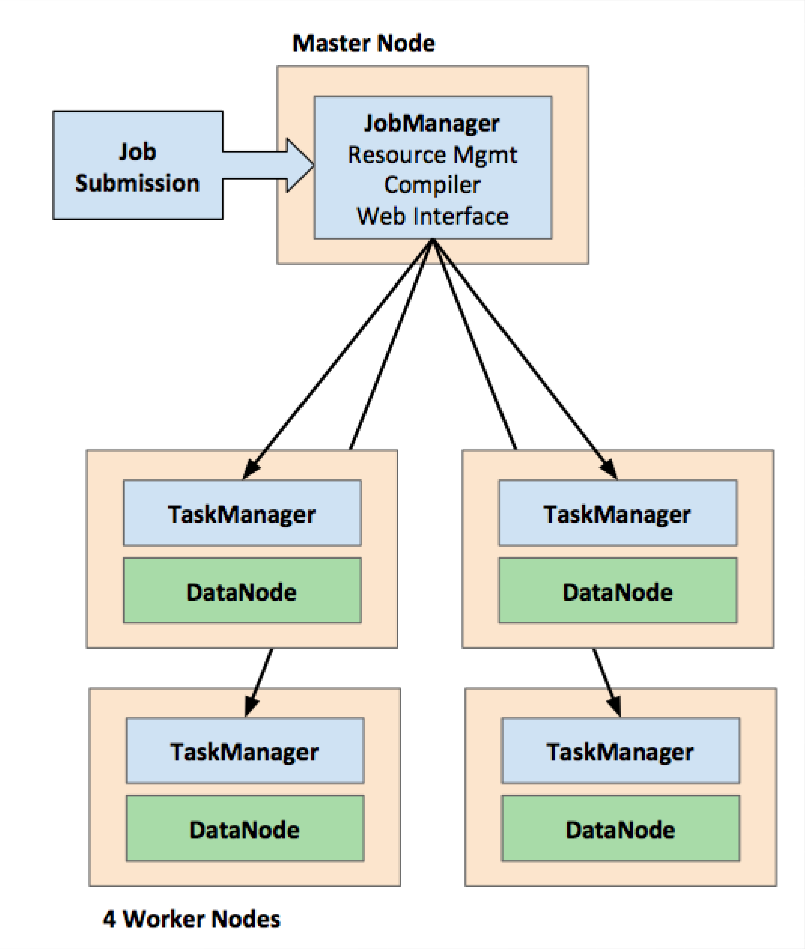
```
> bin/stratosphere run prg.jar
```

master

Stratosphere Runtime

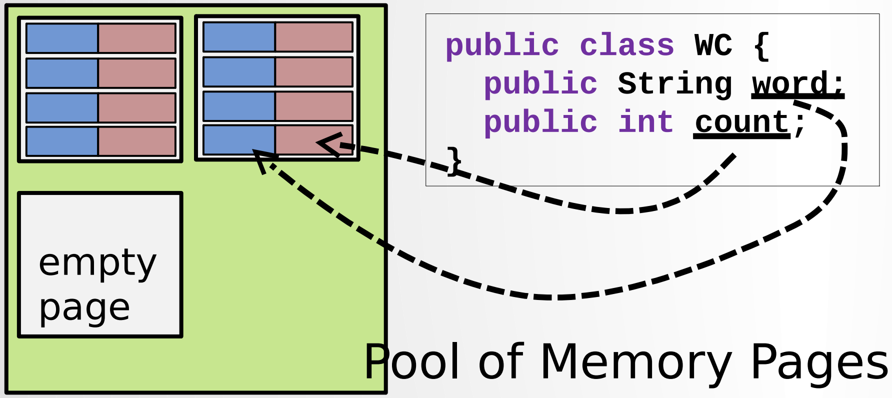
Distributed Runtime

- Master (Job Manager) handles job submission, scheduling, and metadata
- Workers (Task Managers) execute operations
- Data can be streamed between nodes
- All operators start in-memory and gradually go out-of-core



Runtime Architecture *(comparison)*

Big Data looks tiny from Stratosphere


```
public class WC {
  public String word;
  public int count;
}
```

- Works on pages of bytes
- Maps objects transparently to these pages
- Full control over memory, out-of-core enabled
- Algorithms work on binary representation
- Address individual fields (not deserialize

Spark

Distributed Collection

List[WC]

- Collections of objects
- General-purpose serializer (Java / Kryo)
- Limited control over memory & less efficient spilling
- Deserialize all or nothing

Iterative Programs

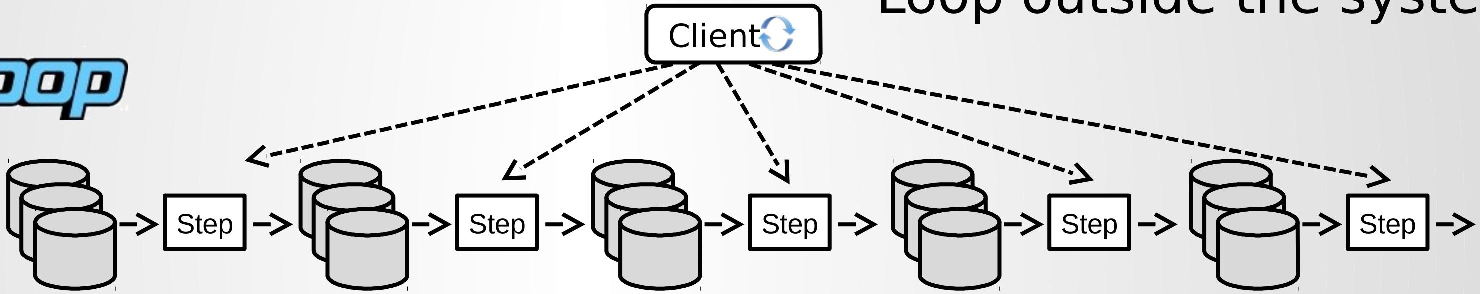
Why Iterative Algorithms

- Algorithms that need iterations
  - Clustering (K-Means, Canopy, …)
  - Gradient descent (e.g., Logistic Regression, Matrix Factorization)
  - Graph Algorithms (e.g., PageRank, Line-Rank, components, paths, reachability, centrality,)
  - Graph communities / dense sub-components
  - Inference (believe propagation)
  - …
- Loop makes multiple passes over the data

Iterations in other systems

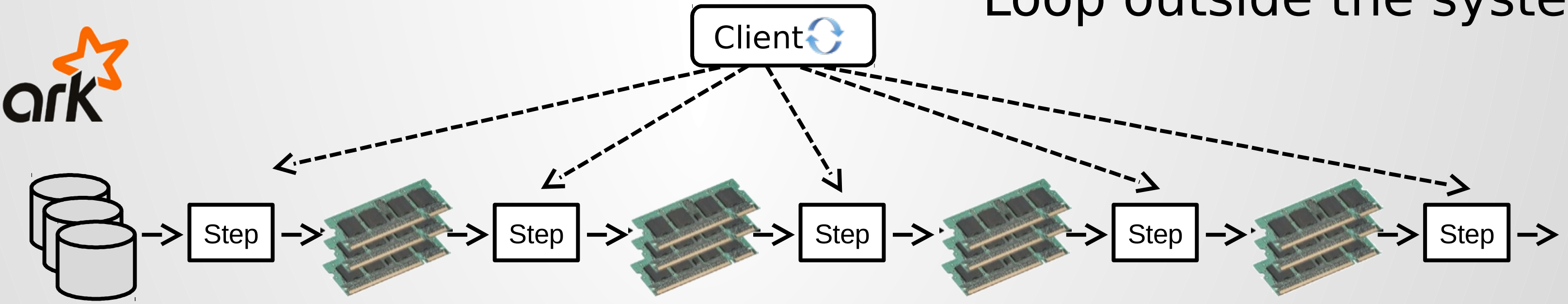
Loop outside the system

hadoop



Loop outside the system

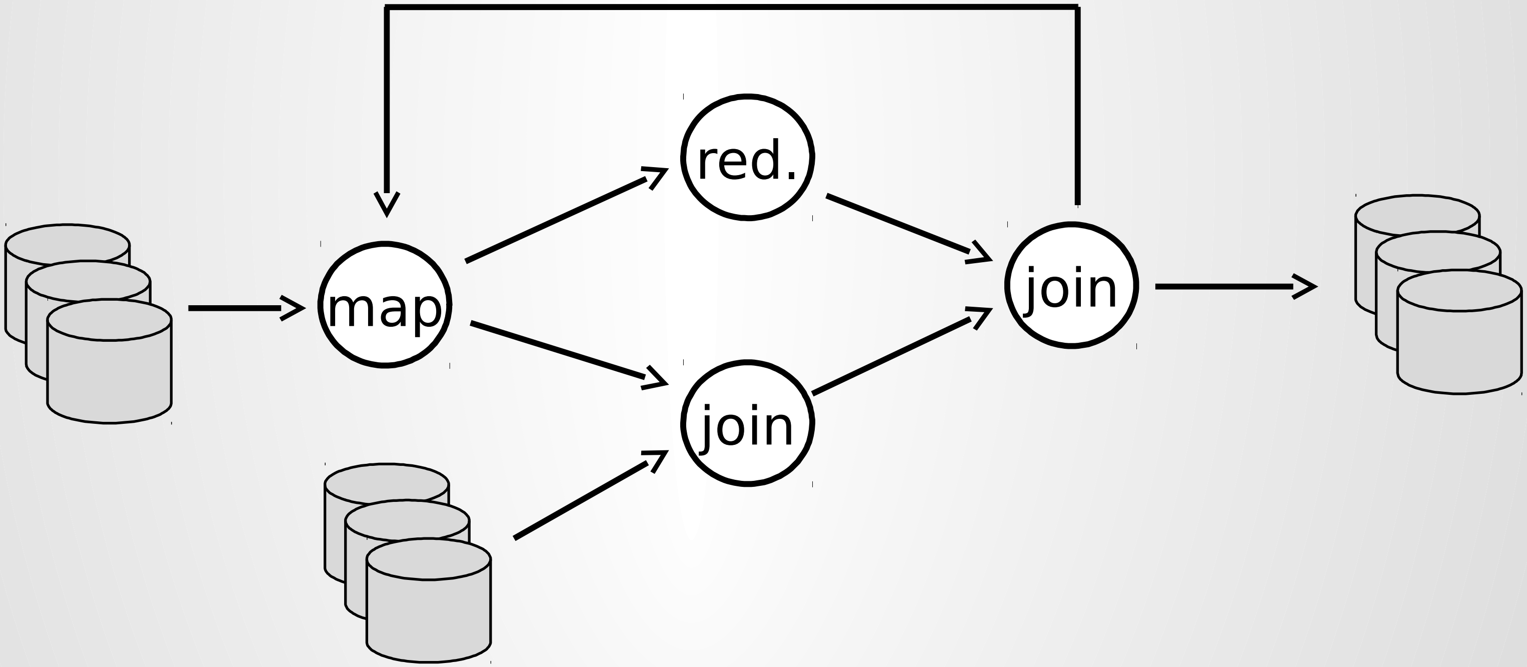
Spark



Iterations in Stratosphere

Big Data looks tiny from Stratosphere

Streaming dataflow with feedback



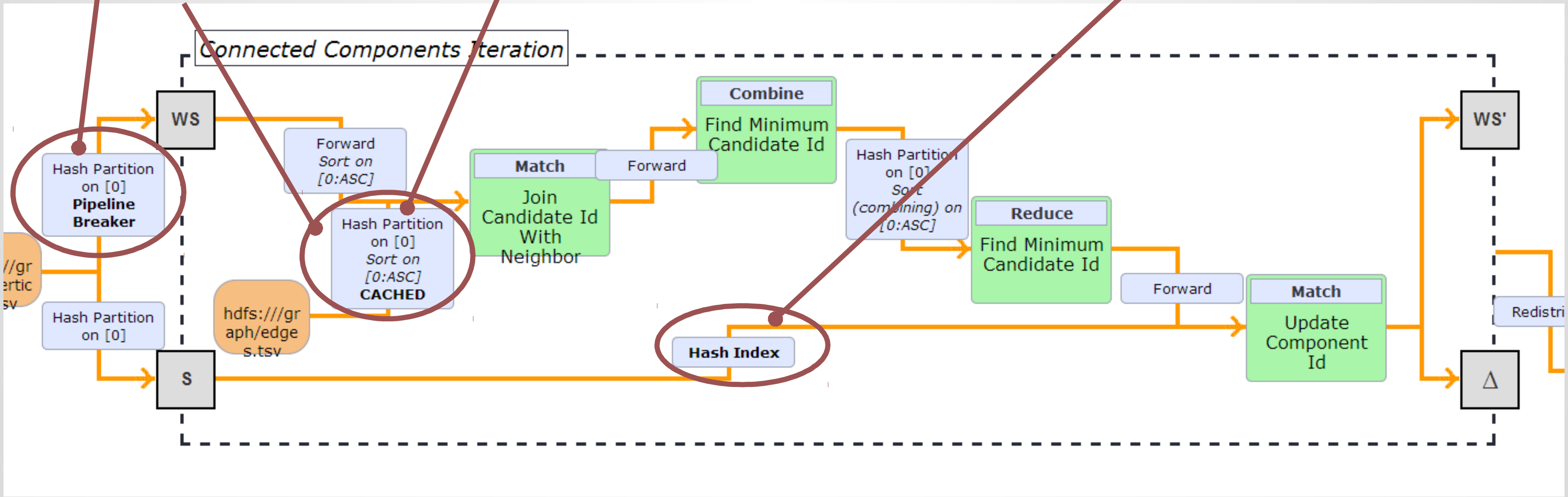
System is iteration-aware, performs automatic optimization

Automatic Optimization for Iterative Programs

Pushing work „out of the loop“

Caching Loop-invariant Data

Maintain state as index


Connected Components Iteration
WS
Hash Partition on [0] Pipeline Breaker
//gr ertic sv
Hash Partition on [0]
S
Forward Sort on [0:ASC]
Hash Partition on [0] Sort on [0:ASC] CACHED
hdfs:///graph/edges.tsv
Match
Join Candidate Id With Neighbor
Forward
Combine
Find Minimum Candidate Id
Hash Index
Hash Partition on [0] Sort (combining) on [0:ASC]
Reduce
Find Minimum Candidate Id
Forward
Match
Update Component Id
WS'
Δ
Redistri

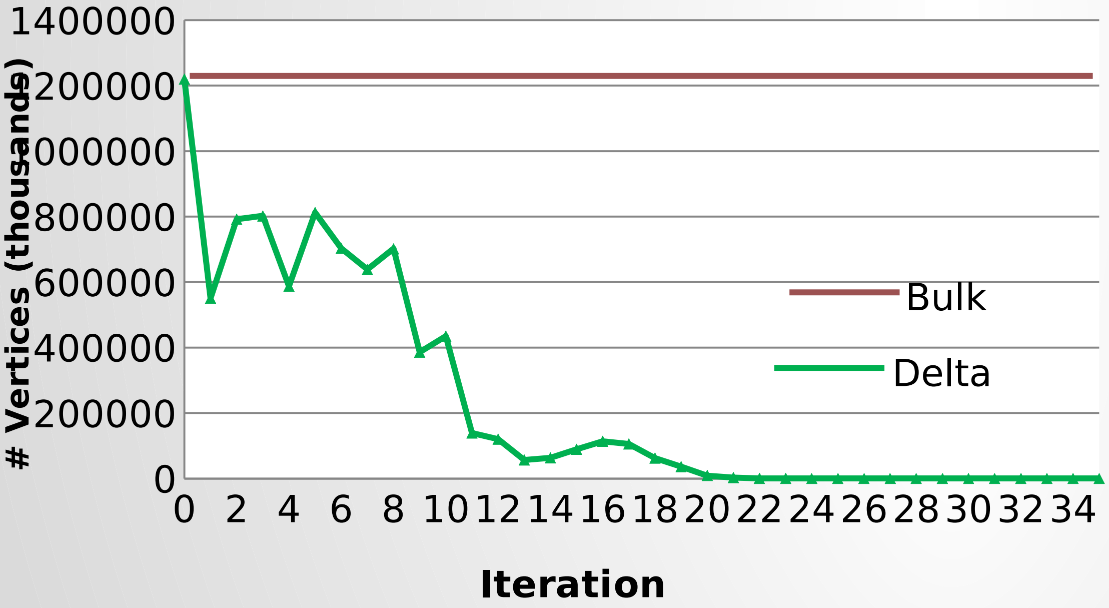
Unifies various kinds of Computations

```
ExecutionEnvironment env = getExecutionEnvironment();

DataSet<Long> vertexIds = ...
DataSet<Tuple2<Long, Long>> edges = ...

DataSet<Tuple2<Long, Long>> vertices = vertexIds.map(new
IdAssigner());

DataSet<Tuple2<Long, Long>> result = vertices .runOperation(
        VertexCentricIteration.withPlainEdges(
                  edges, new CCUpdater(), new CCMessager(), 100));

result.print();
env.execute("Connected Components");
```

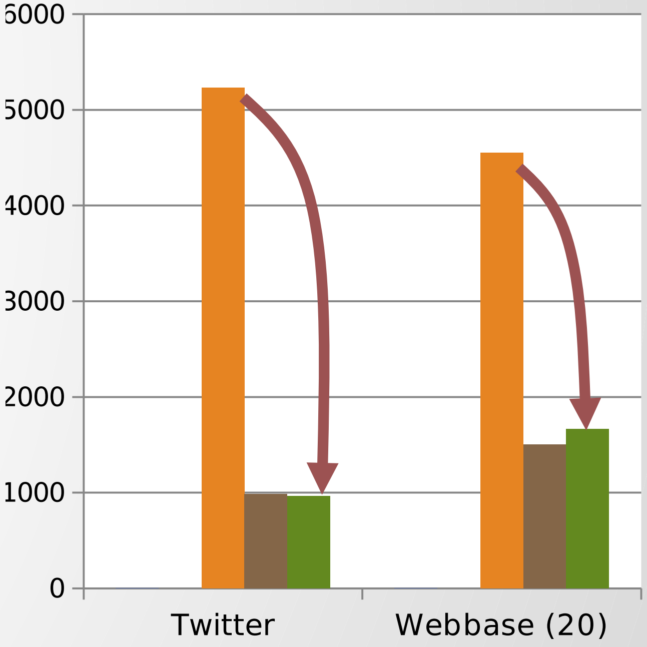
Pregel/Giraph-style Graph Computation

Delta iterations speed up certain problems by a lot

Cover typical use cases of Pregel-like systems with comparable performance in a generic platform and developer API.



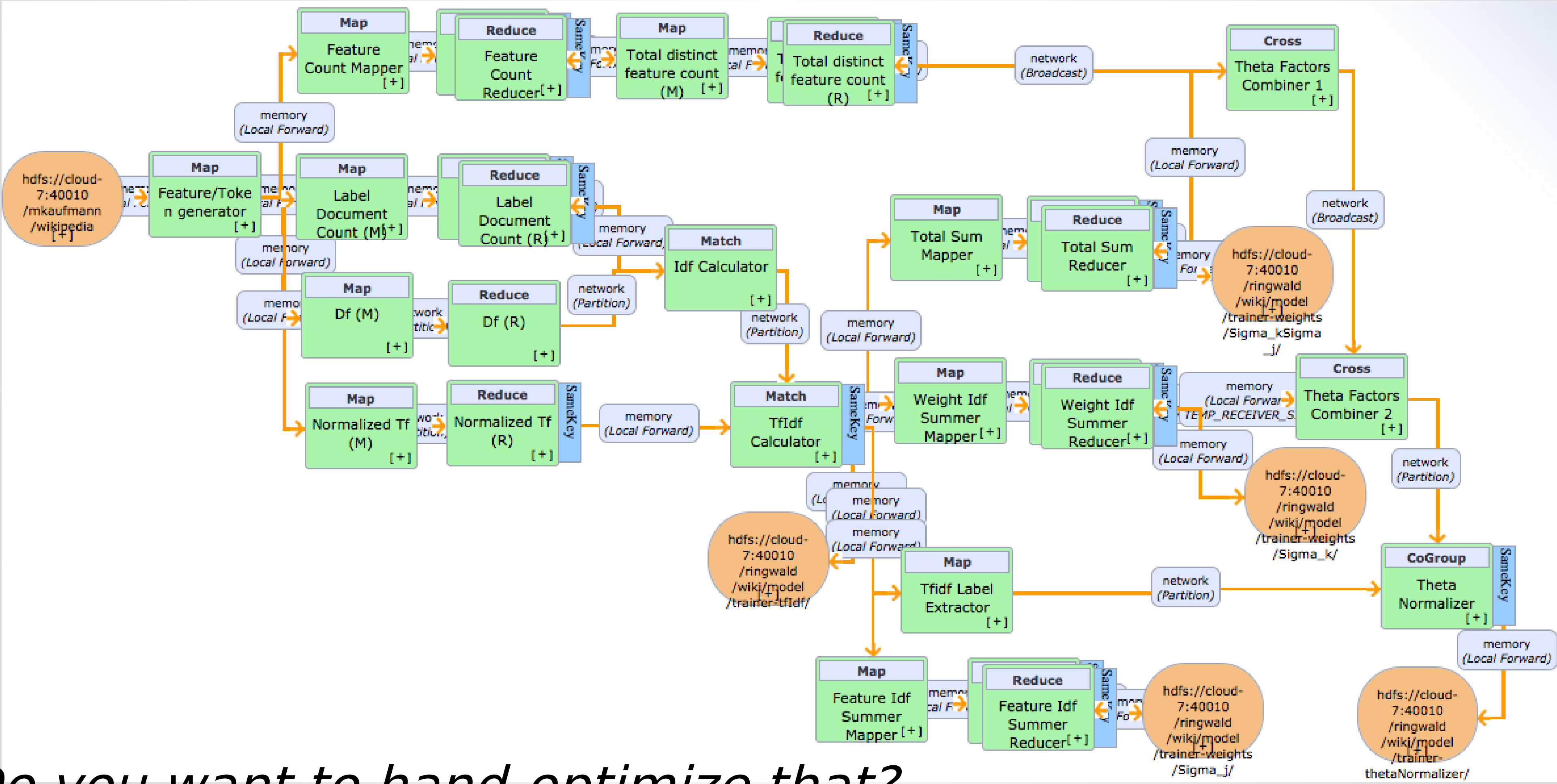
Computations performed in each iteration for connected communities of a social graph



Runtime (secs)

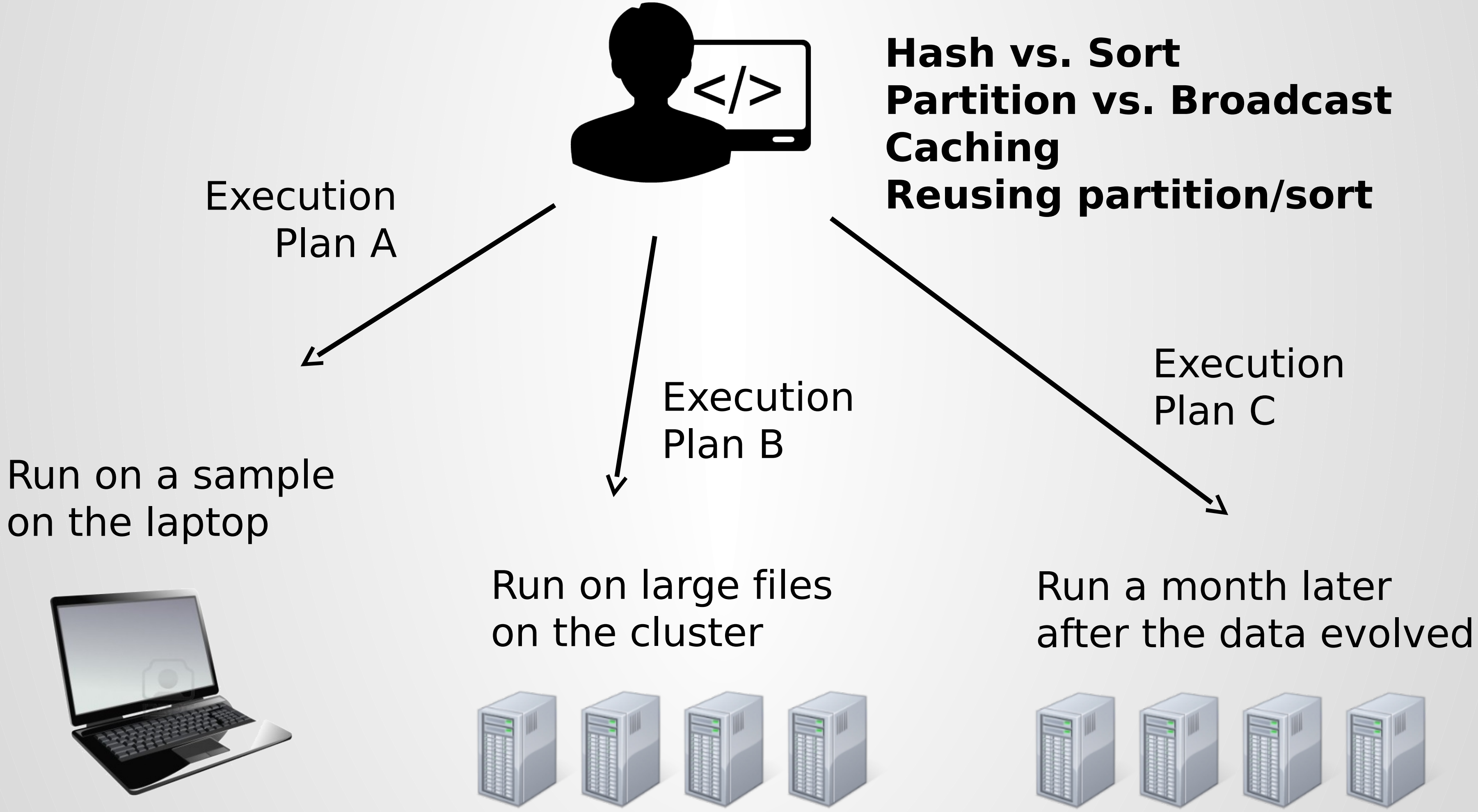
Program Optimization

Why Program Optimization ?


hdfs://cloud-7:40010/mkaufmann/wikipedia
Map Feature/Token generator
Map Feature Count Mapper
Reduce Feature Count Reducer
Map Total distinct feature count (M)
Reduce Total distinct feature count (R)
network (Broadcast)
Cross Theta Factors Combiner 1
memory (Local Forward)
Map Label Document Count (M)
Reduce Label Document Count (R)
Match Idf Calculator
Map Df (M)
Reduce Df (R)
network (Partition)
Map Normalized Tf (M)
Reduce Normalized Tf (R)
Match TfIdf Calculator
SameKey
Map Total Sum Mapper
Reduce Total Sum Reducer
hdfs://cloud-7:40010/ringwald/wiki/model/trainer-weights/Sigma_kSigma_j/
Map Weight Idf Summer Mapper
Reduce Weight Idf Summer Reducer
Cross Theta Factors Combiner 2
TEMP_RECEIVER_S
hdfs://cloud-7:40010/ringwald/wiki/model/trainer-weights/Sigma_k/
hdfs://cloud-7:40010/ringwald/wiki/model/trainer-tfIdf/
Map Tfidf Label Extractor
CoGroup Theta Normalizer
Map Feature Idf Summer Mapper
Reduce Feature Idf Summer Reducer
hdfs://cloud-7:40010/ringwald/wiki/model/trainer-weights/Sigma_j/
hdfs://cloud-7:40010/ringwald/wiki/model/trainer-thetaNormalizer/
[+]

Do you want to hand-optimize that?

What is Automatic Optimization



Hash vs. Sort
Partition vs. Broadcast
Caching
Reusing partition/sort

Execution
Plan A

Execution
Plan B

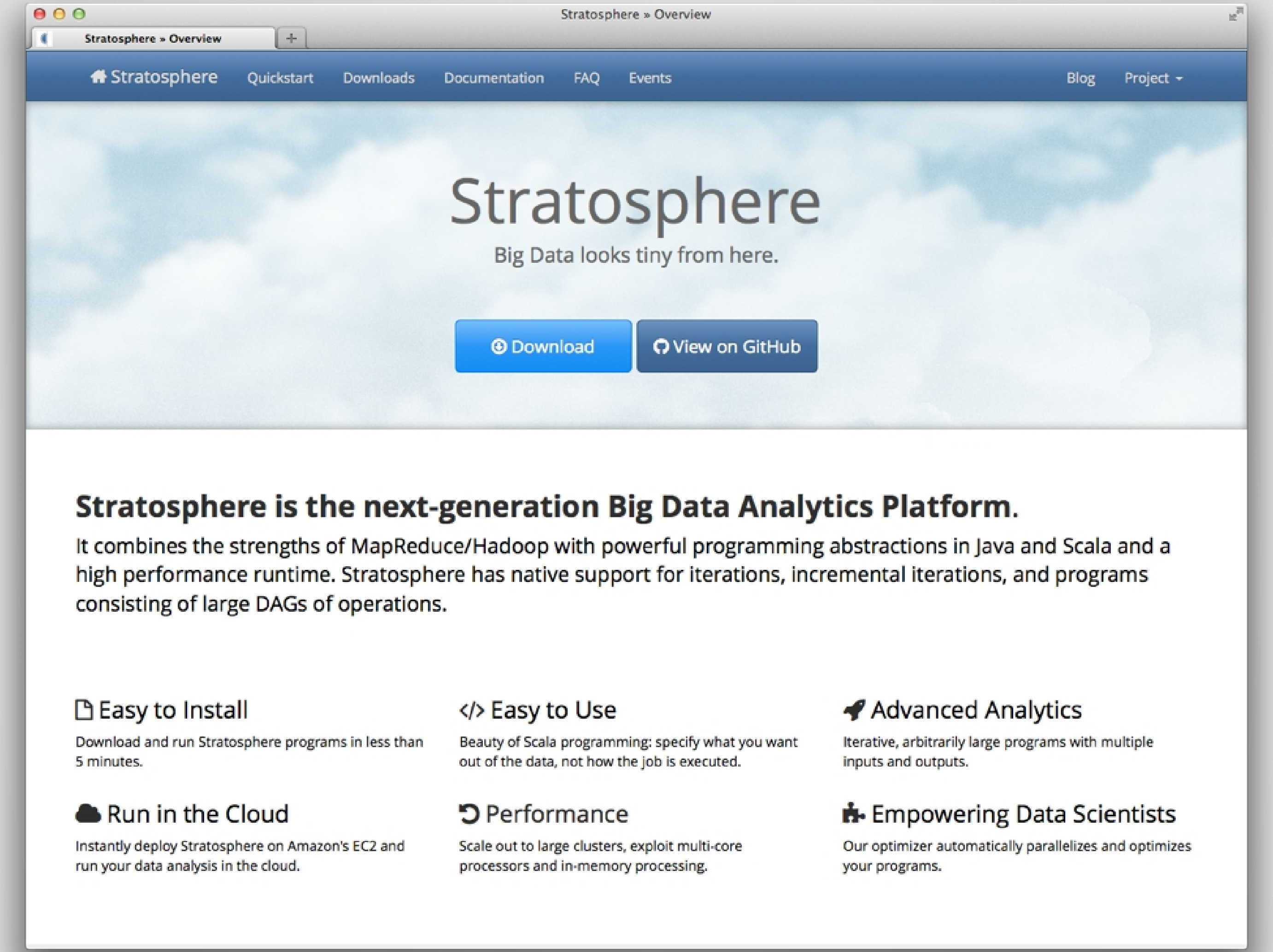
Execution
Plan C

Run on a sample
on the laptop

Run on large files
on the cluster

Run a month later
after the data evolved

Using Stratosphere



Stratosphere

Big Data looks tiny from here.

Stratosphere is the next-generation Big Data Analytics Platform.

It combines the strengths of MapReduce/Hadoop with powerful programming abstractions in Java and Scala and a high performance runtime. Stratosphere has native support for iterations, incremental iterations, and programs consisting of large DAGs of operations.

Easy to Install

Download and run Stratosphere programs in less than 5 minutes.

</> Easy to Use

Beauty of Scala programming: specify what you want out of the data, not how the job is executed.

Advanced Analytics

Iterative, arbitrarily large programs with multiple inputs and outputs.

Run in the Cloud

Instantly deploy Stratosphere on Amazon's EC2 and run your data analysis in the cloud.

Performance

Scale out to large clusters, exploit multi-core processors and in-memory processing.

Empowering Data Scientists

Our optimizer automatically parallelizes and optimizes your programs.

www.stratosphere.eu

Its easy to get started...

trying it out...

Quickstart projects set up a program skeleton, including embedded local execution/debugging environment...

running a local pseudo-clus

```
$ wget https://.../stratosphere-0.4.tgz
$ tar xzf stratosphere-*.tgz
$ stratosphere/bin/start-local.sh
```

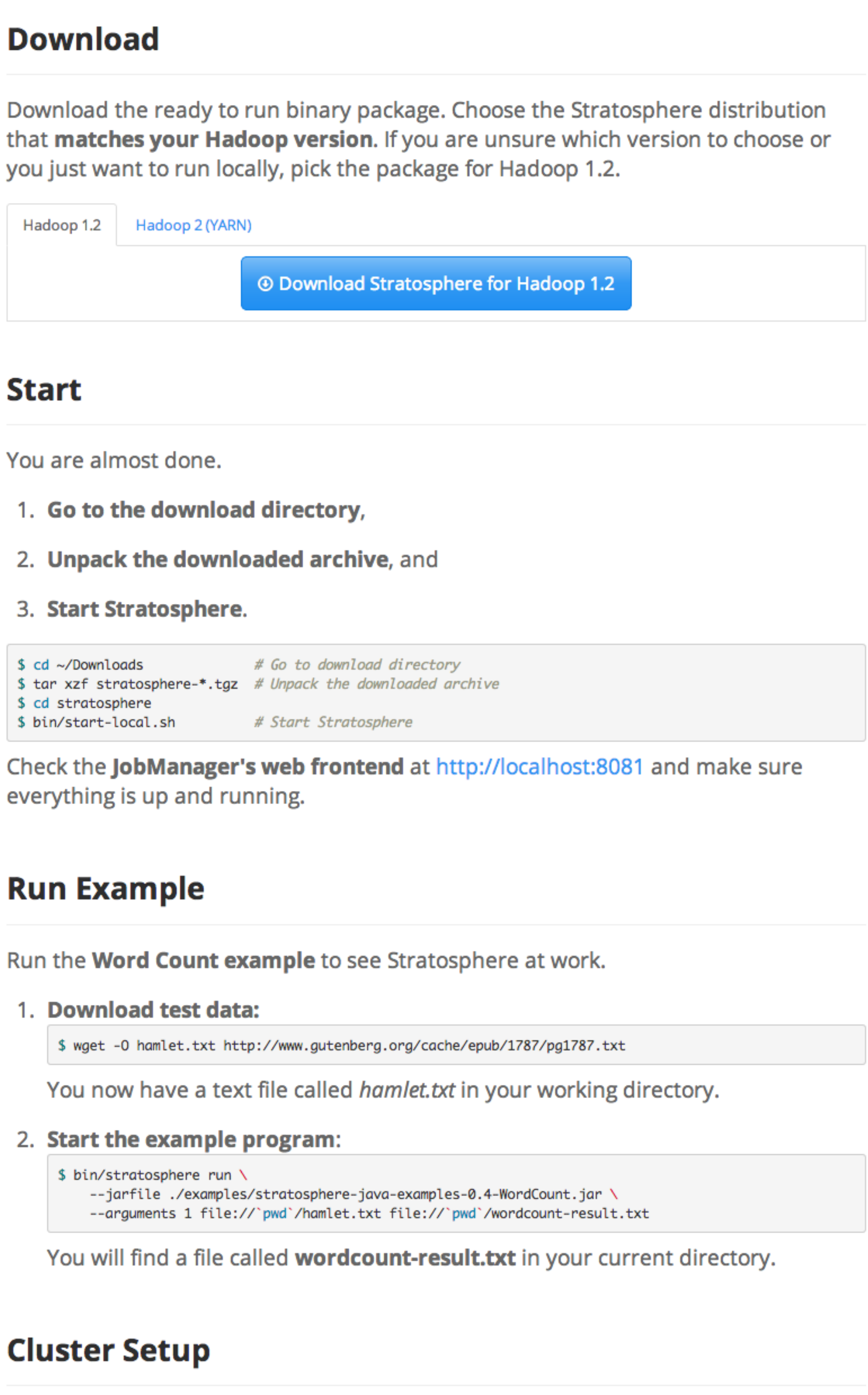
Also available as a Debian package

or the experts...

If you have YARN, deploy a full stratosphere setup in 3 command

```
wget http://stratosphere-bin.s3-website-us-east-1
             .amazonaws.com/stratosphere-dist-0.5-SNAPSHOT-yarn.tar.gz
tar xvzf stratosphere-dist-0.5-SNAPSHOT-yarn.tar.gz
./stratosphere-yarn-0.5-SNAPSHOT/bin/yarn-session.sh -n 4 -jm 1024 -tm 3000
```

Also works on Amazon Elastic MapReduce ;-)



Download

Download the ready to run binary package. Choose the Stratosphere distribution that **matches your Hadoop version.** If you are unsure which version to choose or you just want to run locally, pick the package for Hadoop 1.2.

Start

You are almost done.

1. **Go to the download directory,**
2. **Unpack the downloaded archive,** and
3. **Start Stratosphere.**

```
$ cd ~/Downloads              # Go to download directory
$ tar xzf stratosphere-*.tgz  # Unpack the downloaded archive
$ cd stratosphere
$ bin/start-local.sh          # Start Stratosphere
```

Check the **JobManager's web frontend** at http://localhost:8081 and make sure everything is up and running.

Run Example

Run the **Word Count example** to see Stratosphere at work.

1. **Download test data:**

```
$ wget -O hamlet.txt http://www.gutenberg.org/cache/epub/1787/pg1787.txt
```

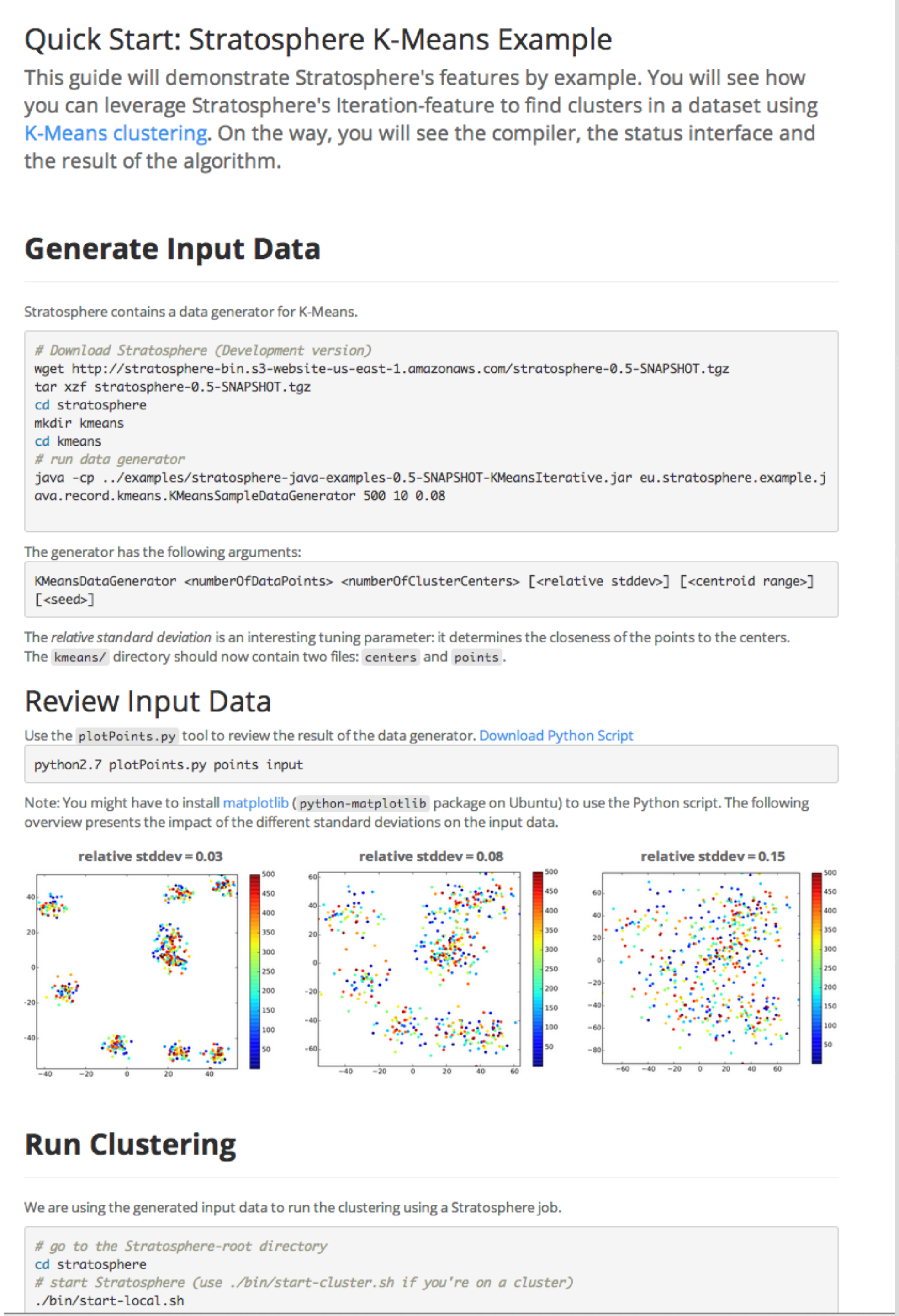
   You now have a text file called *hamlet.txt* in your working directory.

2. **Start the example program:**

```
$ bin/stratosphere run \
    --jarfile ./examples/stratosphere-java-examples-0.4-WordCount.jar \
    --arguments 1 file://`pwd`/hamlet.txt file://`pwd`/wordcount-result.txt
```

   You will find a file called **wordcount-result.txt** in your current directory.

Cluster Setup



Quick Start: Stratosphere K-Means Example

This guide will demonstrate Stratosphere's features by example. You will see how you can leverage Stratosphere's Iteration-feature to find clusters in a dataset using K-Means clustering. On the way, you will see the compiler, the status interface and the result of the algorithm.

Generate Input Data

Stratosphere contains a data generator for K-Means.

```
# Download Stratosphere (Development version)
wget http://stratosphere-bin.s3-website-us-east-1.amazonaws.com/stratosphere-0.5-SNAPSHOT.tgz
tar xzf stratosphere-0.5-SNAPSHOT.tgz
cd stratosphere
mkdir kmeans
cd kmeans
# run data generator
java -cp ../examples/stratosphere-java-examples-0.5-SNAPSHOT-KMeansIterative.jar eu.stratosphere.example.java.record.kmeans.KMeansSampleDataGenerator 500 10 0.08
```

The generator has the following arguments:

```
KMeansDataGenerator <numberOfDataPoints> <numberOfClusterCenters> [<relative stddev>] [<centroid range>] [<seed>]
```

The *relative standard deviation* is an interesting tuning parameter: it determines the closeness of the points to the centers. The `kmeans/` directory should now contain two files: `centers` and `points`.

Review Input Data

Use the `plotPoints.py` tool to review the result of the data generator. Download Python Script

```
python2.7 plotPoints.py points input
```

Note: You might have to install matplotlib (`python-matplotlib` package on Ubuntu) to use the Python script. The following overview presents the impact of the different standard deviations on the input data.

Run Clustering

We are using the generated input data to run the clustering using a Stratosphere job.

```
# go to the Stratosphere-root directory
cd stratosphere
# start Stratosphere (use ./bin/start-cluster.sh if you're on a cluster)
./bin/start-local.sh
```

Roadmap

- Last **pre-Apache release 0.5** coming now, moving to Apache
- **Mid-query fault tolerance**
- **Interactive queries / Cross query data caching**
- Adding **Tez as a distributed runtime backend**
- Add support for the **new Mahout Scala DSL**
- **Streaming** – Initial Storm-like API coming up (SZTAKI Budapest)
- Add **"logical" operations** to Java API

The Infamous WordCount in Stratosphere

Java API – Expression Variant

- (prototype)

```
public class WC {
  public String word;
  public int count;
}
```

```
DataSet<String> text = env.readTextFile(input);

DataSet<WC> words = text.flatMap(

  new FlatMapFunction<String, WC>() {
    public void flatMap(String value, Collector<WC> out){
      for (String token : value.toLowerCase().split("\\W")) {
        out.collect(new WC(token, 1));
      }
    }
  });

words.groupBy("word").aggregate(SUM, "count");
```

"Big Data looks tiny from Stratosphere"

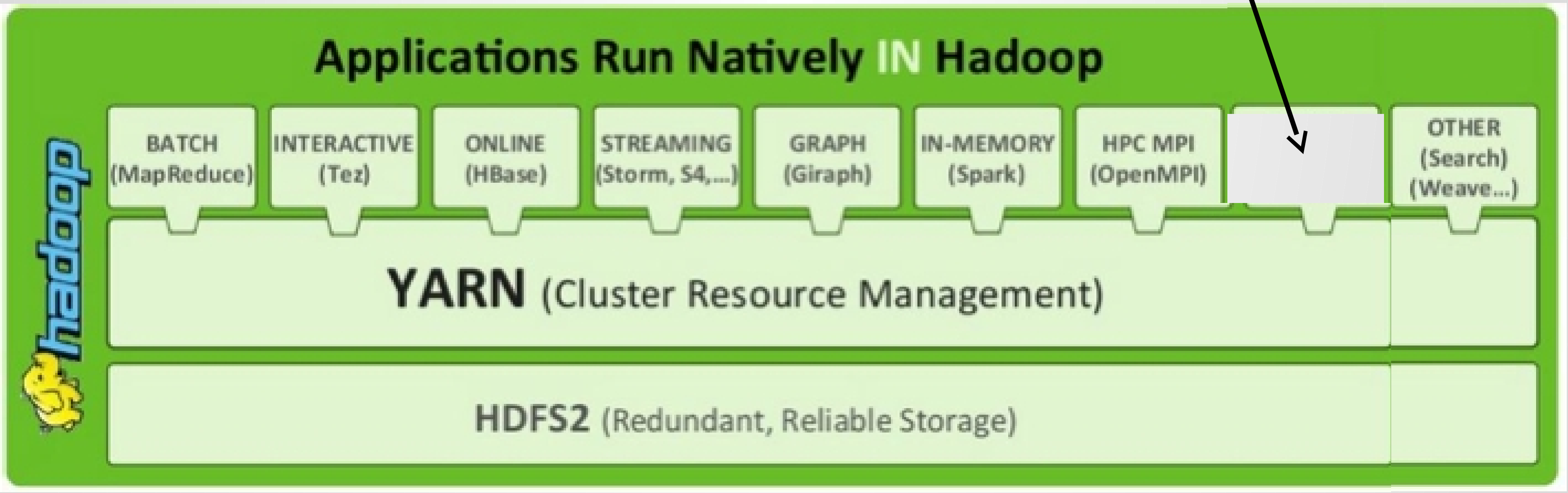
- stratosphere.eu
- github.com/stratosphere/stratosphere
- @stratosphere_eu

Appendix

Fits into Hadoop Stack

- Analyzes HDFS data directly
- Runs on top of YARN

Big Data looks tiny from Stratosphere


Applications Run Natively IN Hadoop
BATCH (MapReduce)
INTERACTIVE (Tez)
ONLINE (HBase)
STREAMING (Storm, S4,...)
GRAPH (Giraph)
IN-MEMORY (Spark)
HPC MPI (OpenMPI)
OTHER (Search) (Weave...)
YARN (Cluster Resource Management)
HDFS2 (Redundant, Reliable Storage)
hadoop

Overview

	hadoop	StratoSphere Above the Clouds	Spark
Paradigm	MapReduce	Iterative Data Flows	Distributed Collections (RDD)
Data Model	Writable Key/Value pairs	Java/Scala type system	Java/Scala types as key/value pairs
Runtime	Batch Parallel Sort	Streaming in-memory & out of core	Batch processing in memory
Compilation/ Optimization	none	holistic planning for data exchange, sort/hash,	none

Data Model
Stratosphere vs. Spark

StratoSphere Above the Clouds	Spark
Arbitrary Java Objects	Arbitrary Java Objects
Tuples as first class citizens	Key/value pairs as first class citizens
Joins / Grouping via field references *(tuple position, selector-function)* *(coming: field name)*	Joins / Grouping via Key/value pairs

The Infamous WordCount in Stratosphere

Java API – POJO Variant

```
public class WC {
  public String word;
  public int count;
}
```

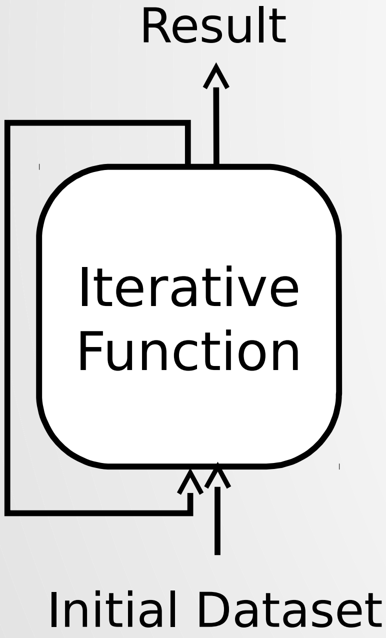
```
DataSet<String> text = env.readTextFile(input);

DataSet<WC> words = text.flatMap(
  new FlatMapFunction<String, WC>() {
    public void flatMap(String value, Collector<WC> out){
      for (String token : value.toLowerCase().split("\\W")) {
        out.collect(new WC(token, 1));
      }
    }
  });

words.groupBy( (WC v) -> return v.word; )

     .reduce(
        new ReduceFunction<WC>() {
          public WC reduce(WC val1, WC val2) {
            return new WC(val1.word, val1.count + val2.count);
          }
        });
```

Simple and self contained Programming/Testing

```
ExecutionEnvironment env = getExecutionEnvironment();

DataSet<String> text = env.fromElements("To be", "or not to be",
    "or to be still", "and certainly not to be not at all", …);

DataSet<Tuple2<String, Integer>> result = text
              .flatMap(new Tokenizer())
              .groupBy(0).aggregate(SUM, 1);

List<Tuple2<String, Integer>> list = new ArrayList<>();
result.output(new CollectingOutput(list));
env.execute();

// validate the list contents
```

Stratosphere offers two types of iterations

Bulk Iterations

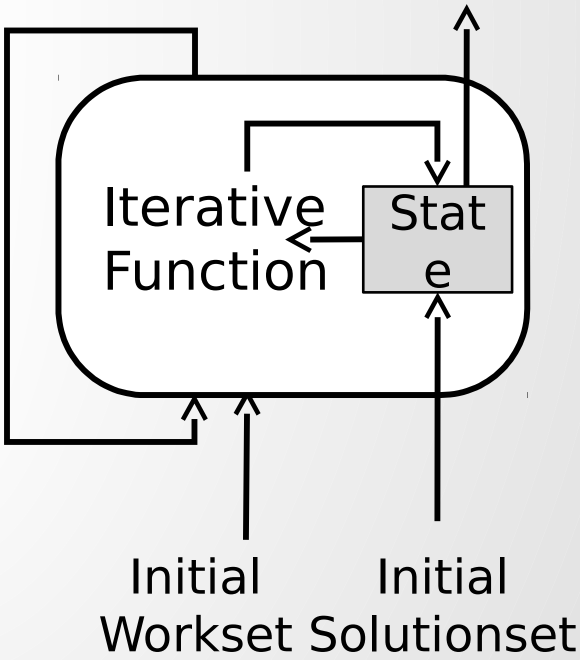


Result

Iterative Function

Initial Dataset

DeltaIterations (aka. Workset Iterations)



Result

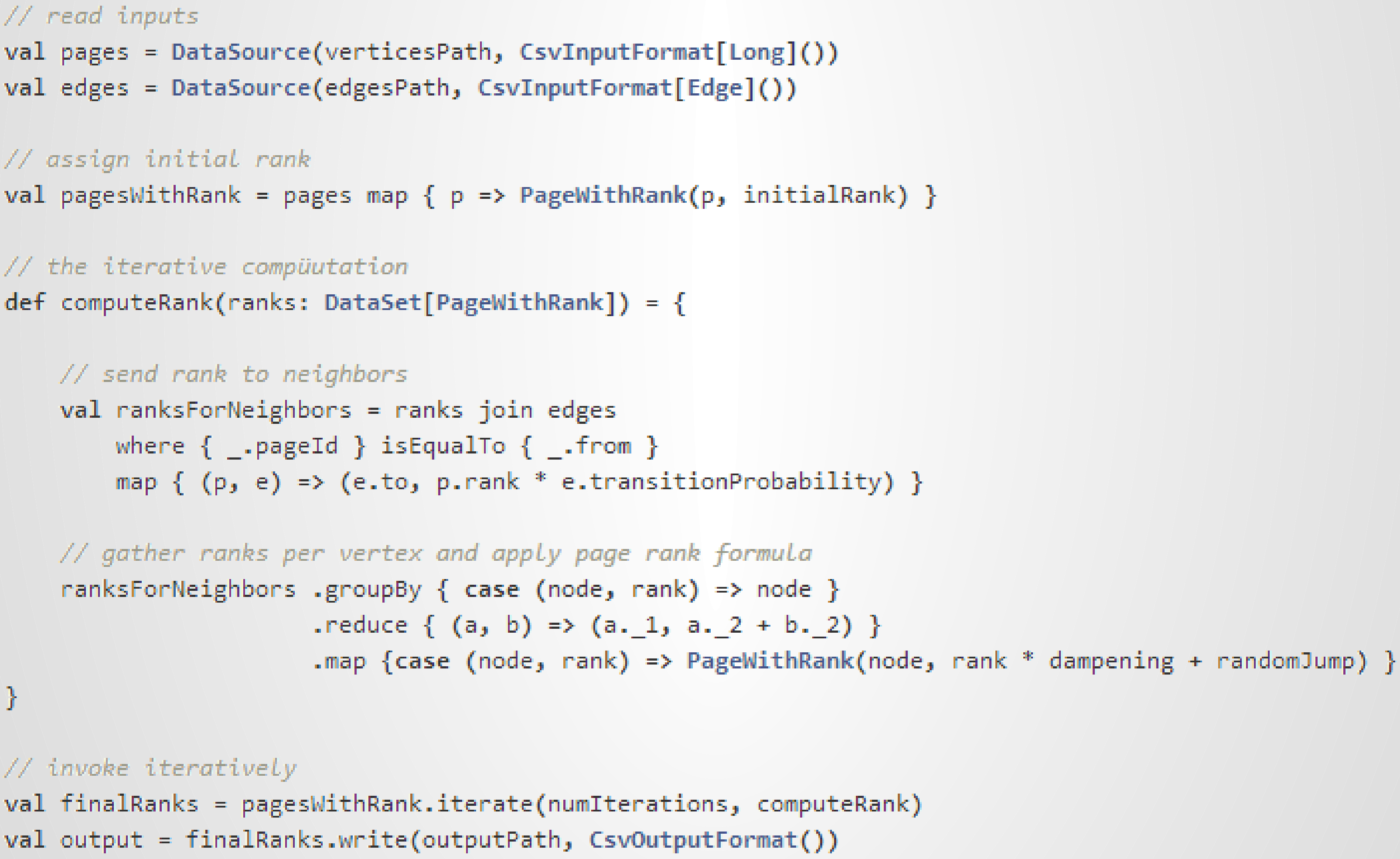
Iterative Function

State

Initial Workset

Initial Solutionset

A Sample Bulk Iteration



```
// read inputs
val pages = DataSource(verticesPath, CsvInputFormat[Long]())
val edges = DataSource(edgesPath, CsvInputFormat[Edge]())

// assign initial rank
val pagesWithRank = pages map { p => PageWithRank(p, initialRank) }

// the iterative compüutation
def computeRank(ranks: DataSet[PageWithRank]) = {

    // send rank to neighbors
    val ranksForNeighbors = ranks join edges
        where { _.pageId } isEqualTo { _.from }
        map { (p, e) => (e.to, p.rank * e.transitionProbability) }

    // gather ranks per vertex and apply page rank formula
    ranksForNeighbors .groupBy { case (node, rank) => node }
                      .reduce { (a, b) => (a._1, a._2 + b._2) }
                      .map {case (node, rank) => PageWithRank(node, rank * dampening + randomJump) }
}

// invoke iteratively
val finalRanks = pagesWithRank.iterate(numIterations, computeRank)
val output = finalRanks.write(outputPath, CsvOutputFormat())
```

A Sample Delta Iteration

Connected Components of a Graph

```
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)
```

A Sample Delta Iteration

Connected Components of a Graph

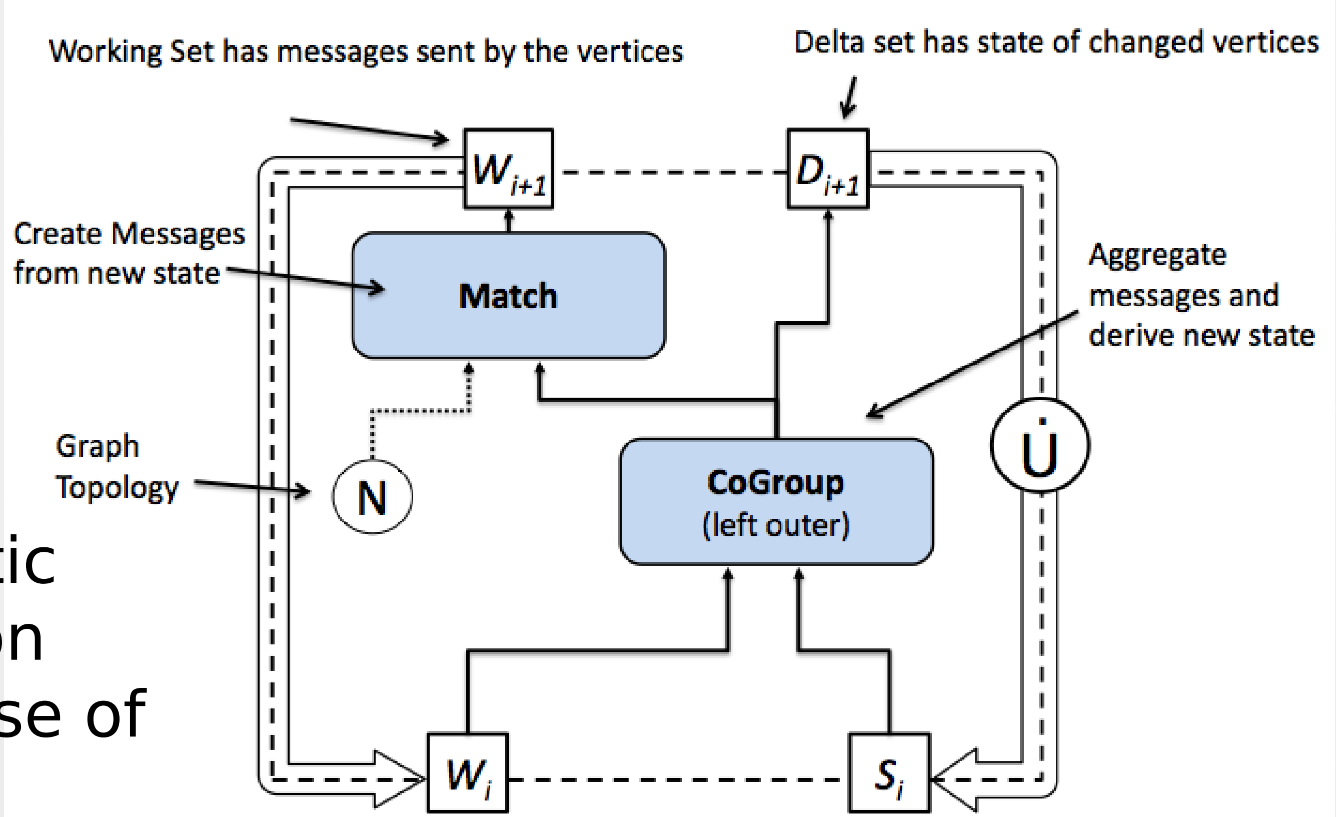
Define Step function

```
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

Spargel: The Graph API

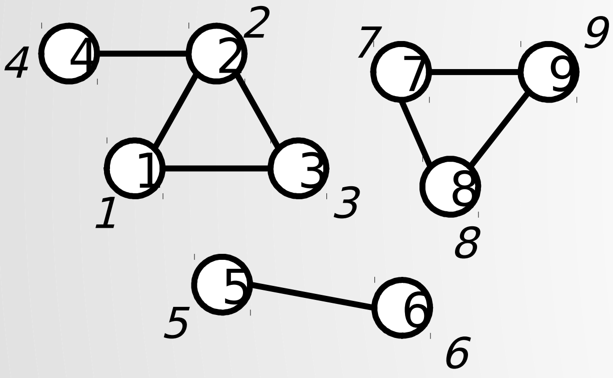


Working Set has messages sent by the vertices
Delta set has state of changed vertices
W_{i+1}
D_{i+1}
Create Messages from new state
Match
Aggregate messages and derive new state
Graph Topology
N
CoGroup (left outer)
$\dot{U}$
W_i
S_i

Vertex Centic computation
is a special case of
a
Delta iteration

A Giraph-style API in < 500 lines of code!

Workset Algorithm Illustrated



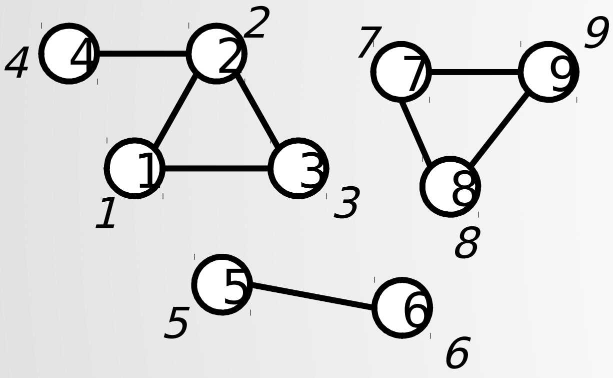
Algorithm: Find connected components of a graph.

Start: Vertices have IDs that respresent the component they belong to. Initially, every vertex has its own id (is its own component).

Step: Each vertex tells its neighbors its component id. Vertices take the min-ID of all candidates from their neighbors. A vertex that did not adopt a new ID needs not participate in the next step, as it has nothing new to tell its

Workset Algorithm Illustrated

Solution Set



Workset

1 (2,2)
(3,3)

2 (1,1)
(3,3)
(4,4)

3 (1,1)
(2,2)

4 (2,2)

5 (6,6)

6 (5,5)

8 (7,7)
(9,9)

9 (7,7)
(8,8)

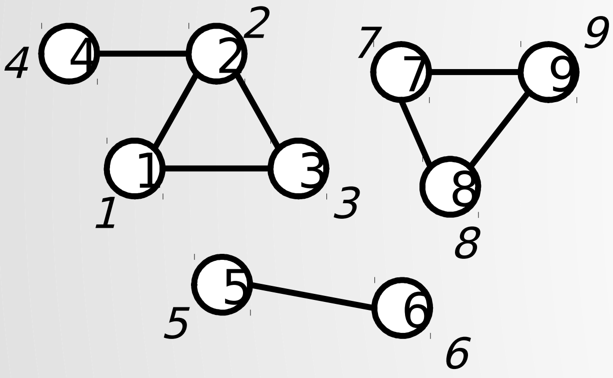
Solution Set Delta

Messages sent to neighbors:

1 (4, 3)   means that vertex 1 receives a candidate id of 3 from vertex 4

Workset Algorithm Illustrated

Solution Set



Workset

1 (2,2)
(3,3)

2 (1,1)
(3,3)
(4,4)

3 (1,1)
(2,2)

4 (2,2)

5 (6,6)

6 (5,5)

8 (7,7)
(9,9)

9 (7,7)
(8,8)

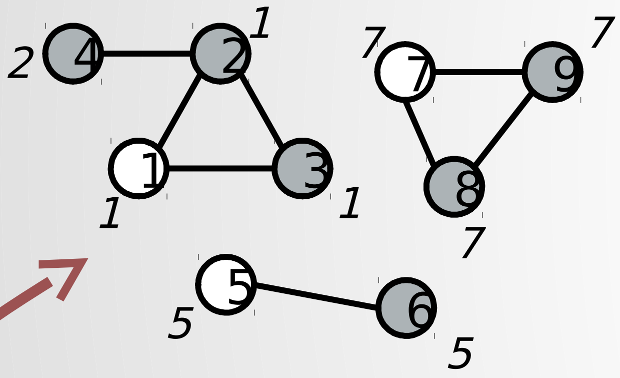
Solution Set Delta

(2,1)
(3,1)
(4,2)

(6, 5)
(8,7)
(9,7)

Workset Algorithm Illustrated

Solution Set



Workset

1 (2,2)
(3,3)

2 (1,1)
(3,3)
(4,4)

3 (1,1)
(2,2)

4 (2,2)

5 (6,6)

6 (5,5)

8 (7,7)
(9,9)

9 (7,7)
(8,8)

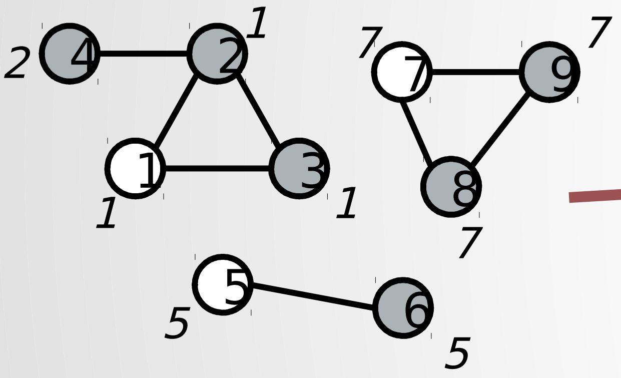
Solution Set Delta

(2,1)
(3,1)
(4,2)

(6, 5)
(8,7)
(9,7)

Workset Algorithm Illustrated

Solution Set



Workset

1 (2,1)
(3,1)

2 (3,1)
(4,2)

3 (2,1)

4 (2,1)

5 (6,5)

7 (8,7)
(9,7)

8 (9,7)

9 (8,7)

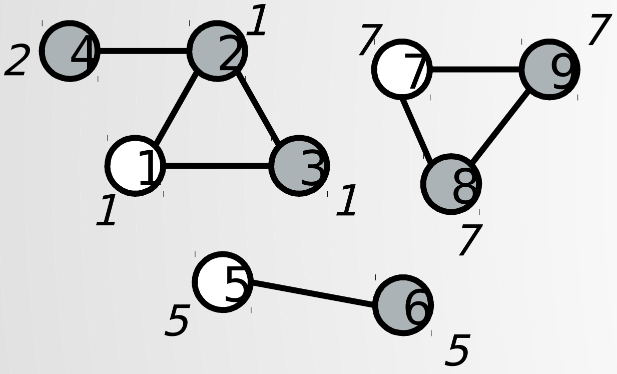
Solution Set Delta

(2,1)
(3,1)
(4,2)

(6, 5)
(8,7)
(9,7)

Workset Algorithm Illustrated

Solution Set



Workset

1 (2,1)
(3,1)

2 (3,1)
(4,2)

3 (2,1)

4 (2,1)

5 (6,5)

7 (8,7)
(9,7)

8 (9,7)

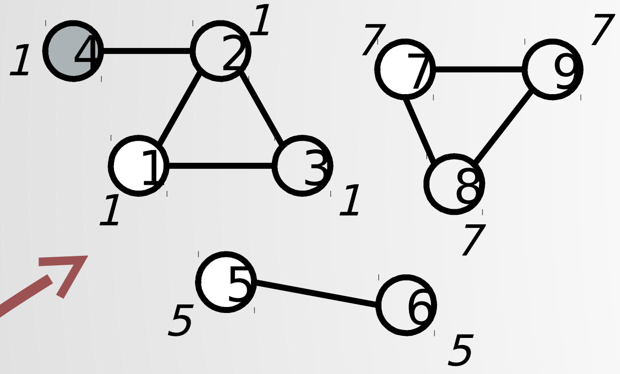
9 (8,7)

Solution Set Delta

(4,1)

Workset Algorithm Illustrated

Solution Set



Workset

1 (2,1)
(3,1)

2 (3,1)
(4,2)

3 (2,1)

4 (2,1)

5 (6,5)

7 (8,7)
(9,7)

8 (9,7)

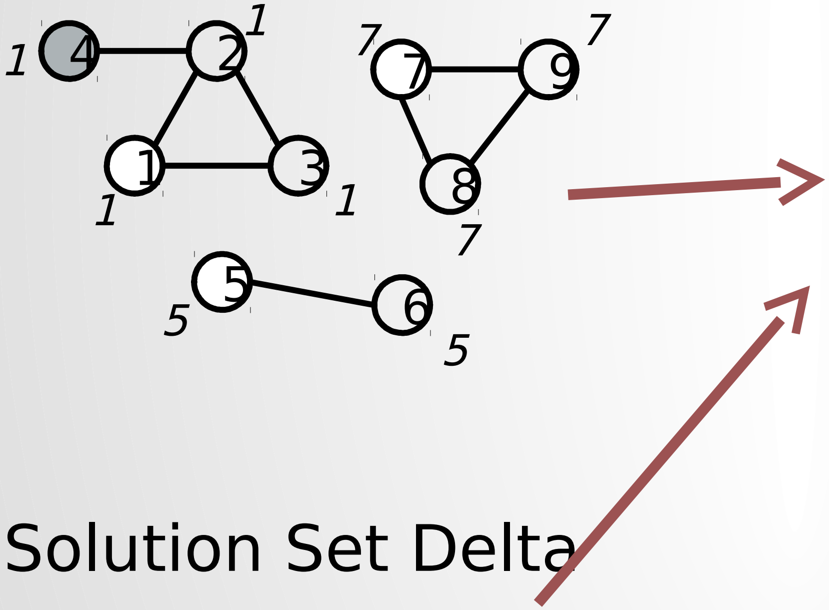
9 (8,7)

Solution Set Delta

(4,1)

Workset Algorithm Illustrated

Solution Set



Workset

2 (4,1)

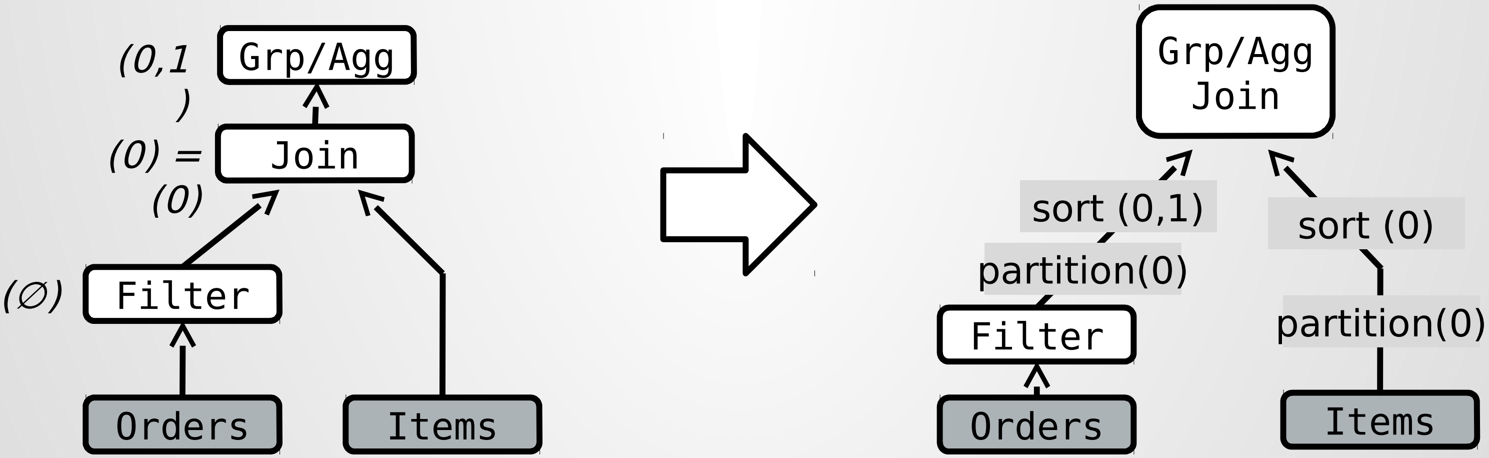
Solution Set Delta

(4,1)

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)
```



Type Analysis/Code Gen

- 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)`

Type Analysis/Code Gen

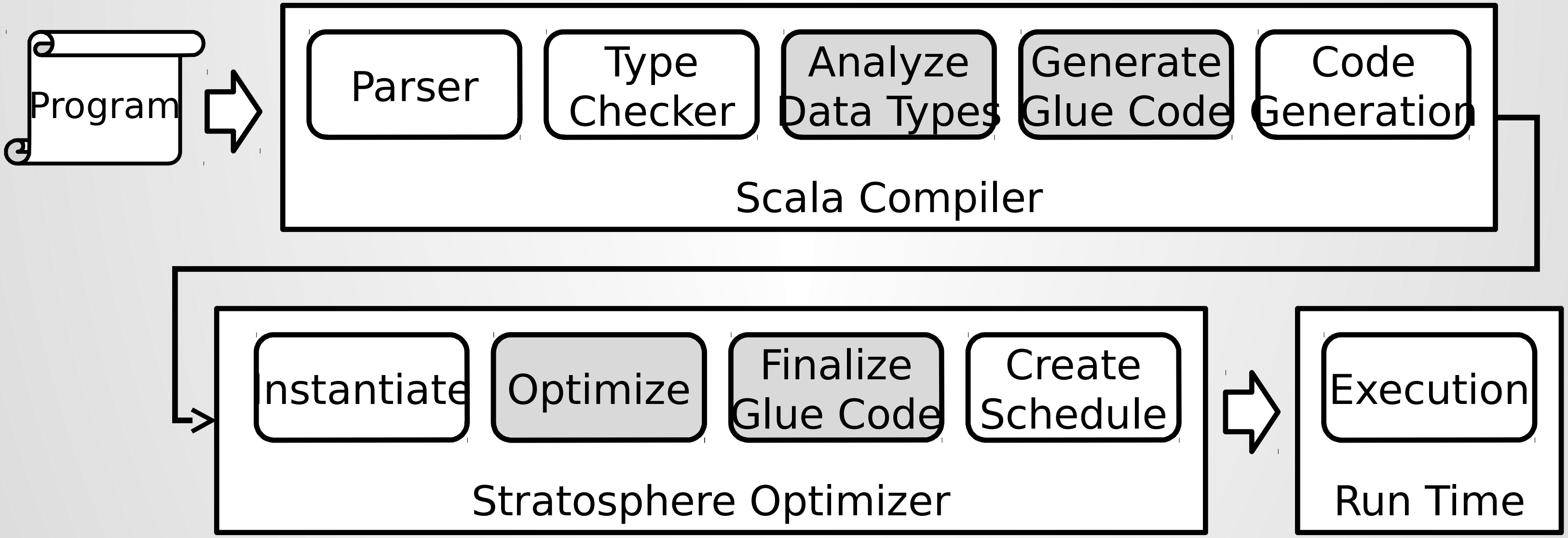
- Implemented in the Scala API via Scala Macros
- Lift the AST, analyze types/code
- Generate type serializers/ accessors and glue code around UDF and type

- Implementation of type analysis via reflection
- Implementation of code generation in Java API in progress

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

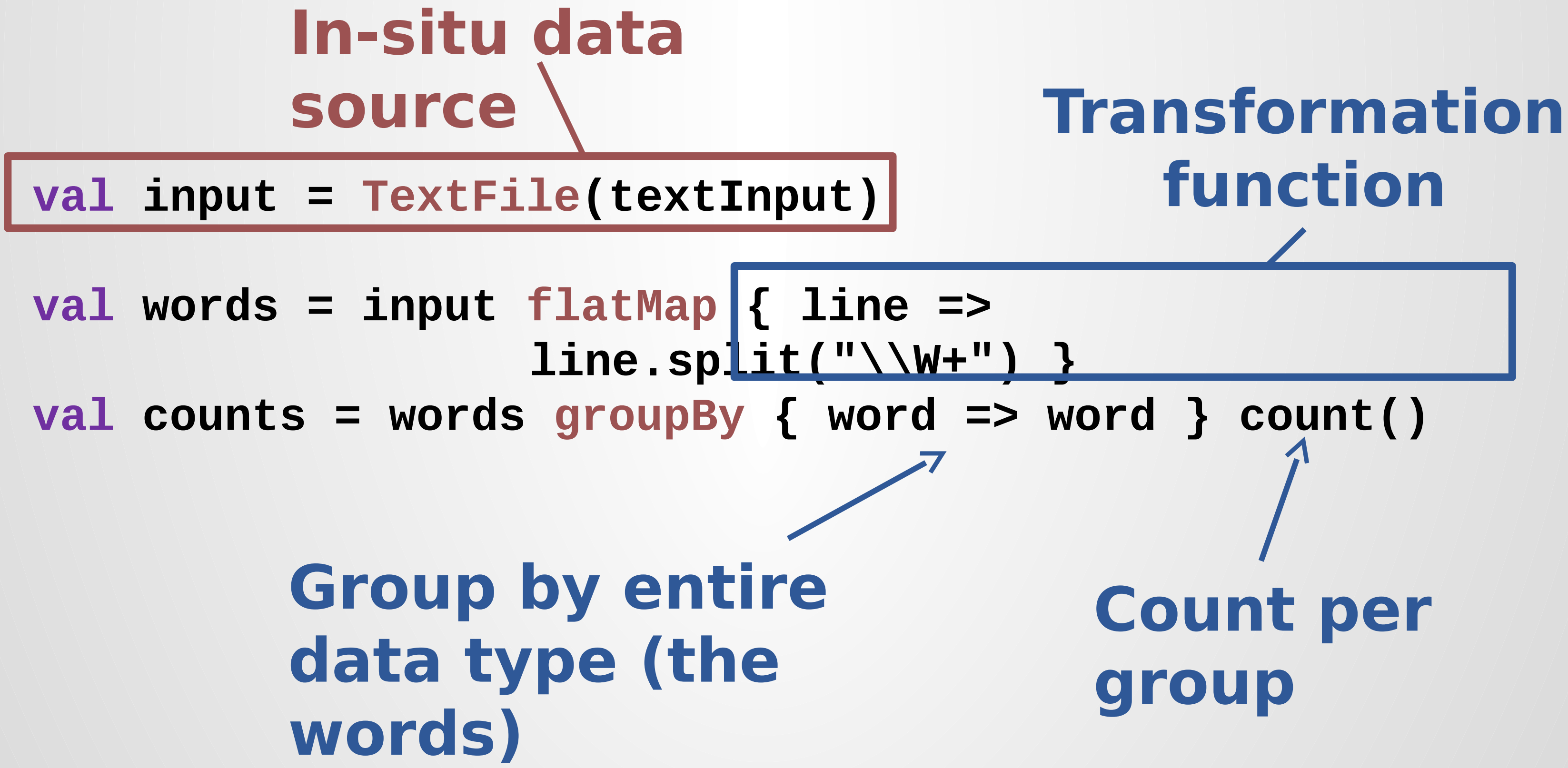


Stratosphere APIs by Example

What does it look like using the system?

Scala API

- The infamous word count example



```
val input = TextFile(textInput)

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

In-situ data source

Transformation function

Group by entire data type (the words)

Count per group

DataSet Transformations

- **Operations** are methods on **DataSet**[A]**.**

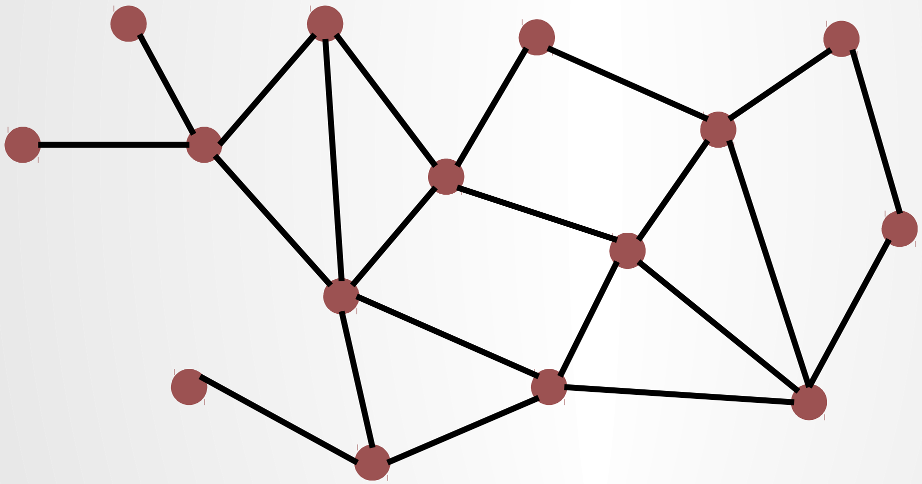
- Working with DataSet[A] **feels like** working with Scala collections.

- DataSet[A] is not an actual collection, but represents **computation on a collection**.

- Stringing together operations creates a **data flow** that can be executed.

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, base1: 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

In-situ data source

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

```
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

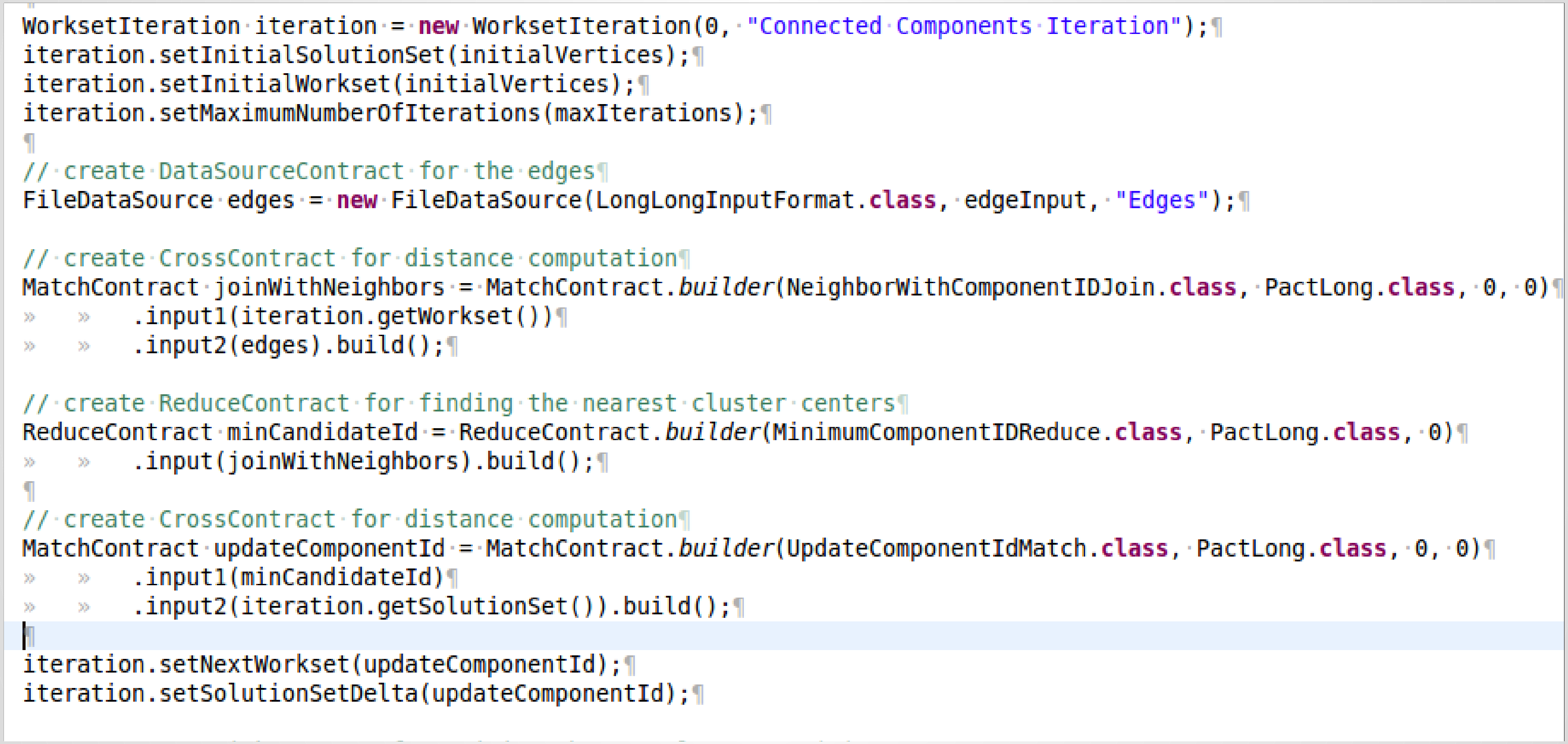
Relational Join

Scala API by Example

```scala
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 }
```

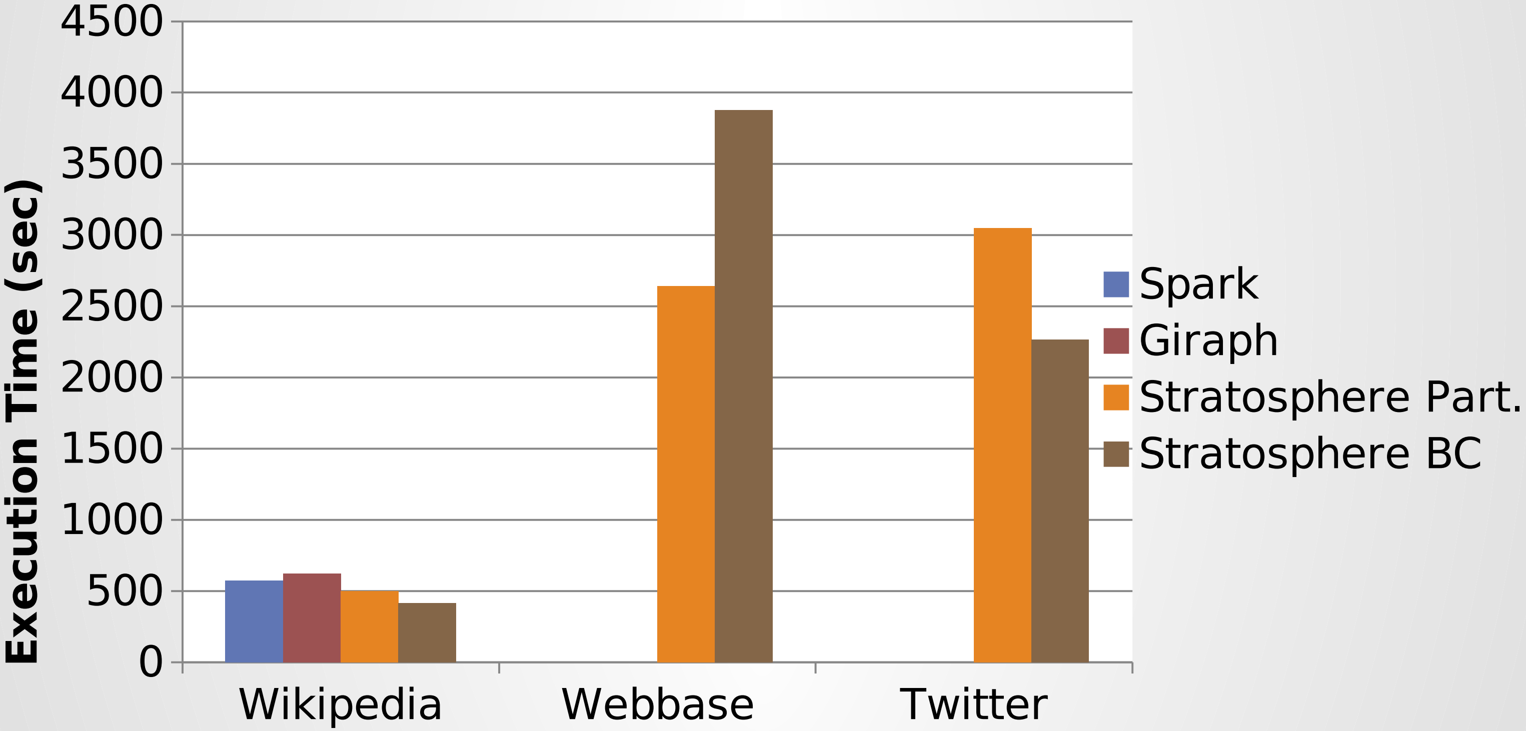
Key References

Iterative Program (Java)



```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);
```

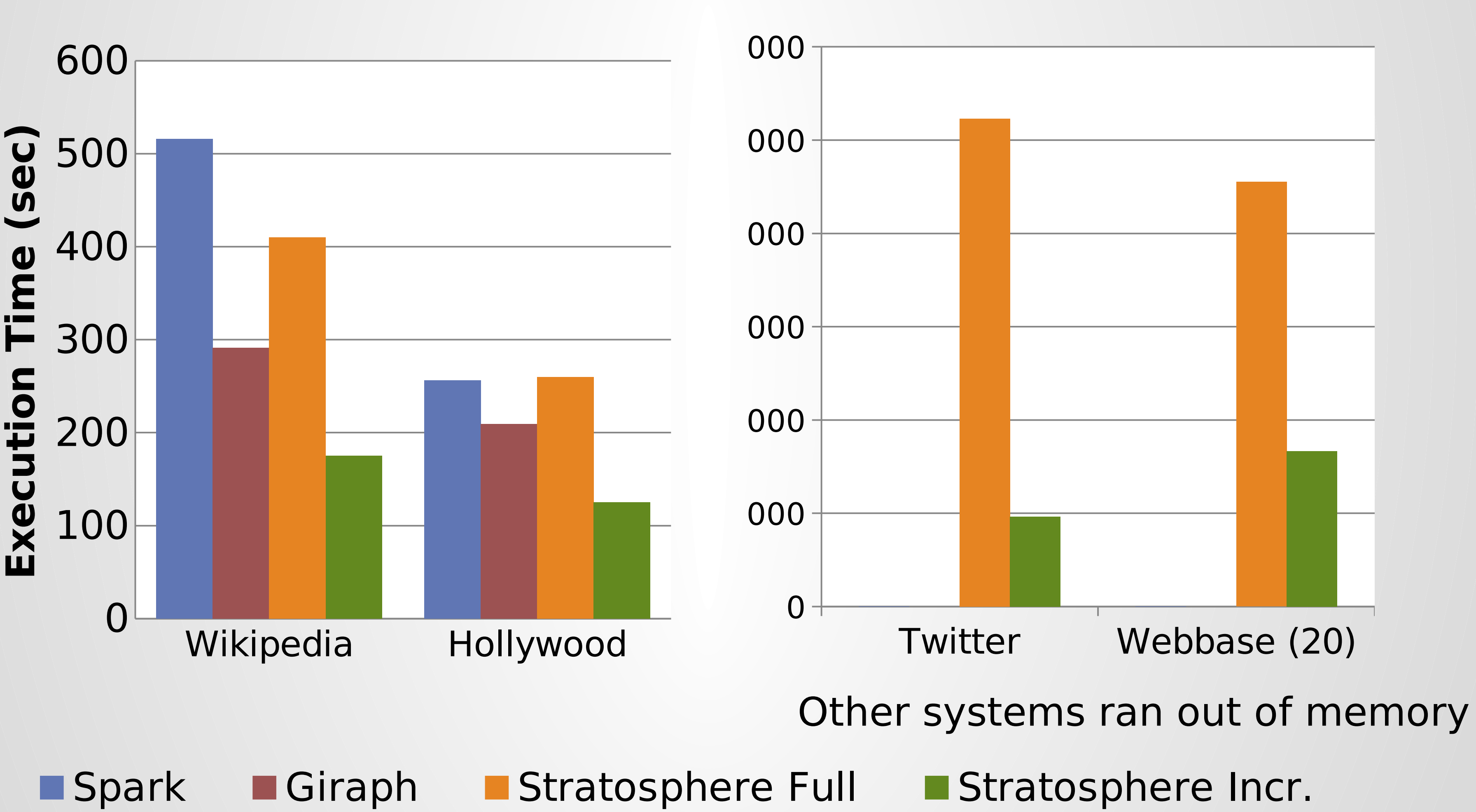
Bulk Iteration Performance



Other systems ran out of memory

Cf. VLDB 2012 Paper "Spinning Fast Iterative Data Flows"

Delta Iteration Performance



Other systems ran out of memory

Cf. VLDB 2012 Paper "Spinning Fast Iterative Data Flows"

Per-Iteration Times

Execution Time (msecs)

■ Spark Ful.

▲ Spark Sim. Incr

(Bagle)

aph

Bulk

eration

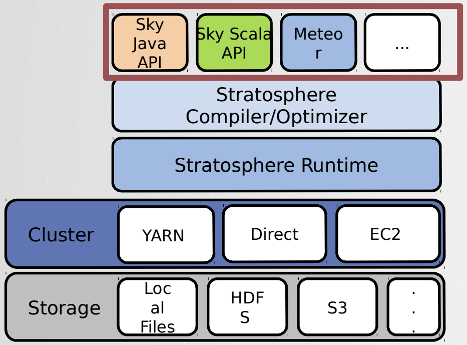
● Stratosphere F

Delta

Stratosphere Incr.

. VLDB 2012 Paper "Spinning Fast Iterative Data Flows"

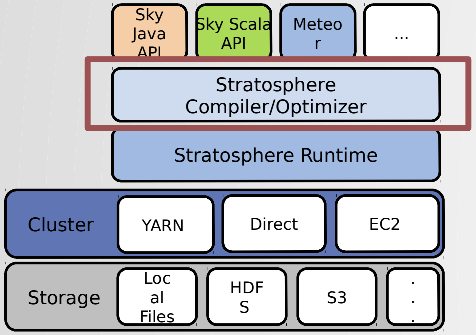
Architecture: Front-Ends



Stratosphere Front-Ends

- Multiple Font-Ends for different target audiences
- Supports operations from both shallow analytics
  (SQL, Hadoop MapReduce) and deep analytics
  (Machine Learning, Data Mining)
- Supports *custom user-defined operations* as required to support the diverse Big Data use cases
- API may be used to create connectors to other application tools (visualization, dashboards, …)

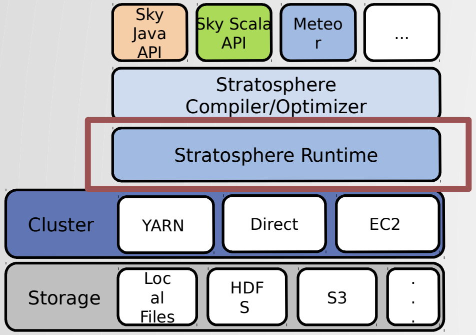
Architecture: Compiler



Stratosphere Compiler

- Inspired by Relational Database Optimizer (optimizes SQL, enables efficient complex queries)
- Extended to non-relational use cases through *code analysis* techniques
- Eliminates costly and time-intensive manual tuning of analysis tasks to the data, automatically adapts programs when the data characteristics change
- Extended to iterative algorithms, optimizes machine learning algorithms and subsumes specialized systems
- *Code generation* techniques efficiently

Architecture: Runtime



Stratosphere Runtime

- Hybrid between a *Parallel Database* and a *MapReduce engine.*
- Supports both database operations, and custom operation defined by users for specialized use cases

- Streaming Engine □ Fast, low latency queries

- Support for *stateful multi-pass algorithms* □ Very efficient for machine learning and graph analysis algorithms

- Heavily in-memory □ Fast on modern computers

- Out-of-core capabilities □ Scales beyond main