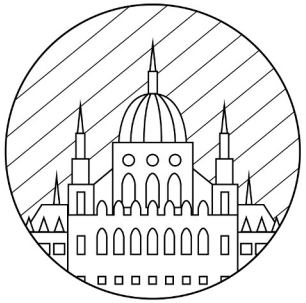
The Flink Big Data Analytics Platform

Marton Balassi, Gyula Fora
{mbalassi, gyfora}@apache.org



APACHECON EUROPE

CORINTHIA HOTEL
BUDAPEST, HUNGARY
NOVEMBER 17-21, 2014

What is Apache Flink?

Open Source

- Started in 2009 by the Berlin-based database research groups
- In the Apache Incubator since mid 2014
- 61 contributors as of the 0.7.0 release

Fast + Reliable

- Fast, general purpose distributed data processing system
- Combining batch and stream processing
- Up to 100x faster than Hadoop

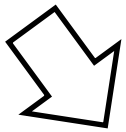
Ready to use

- Programming APIs for Java and Scala
- Tested on large clusters

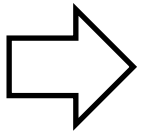
What is Apache Flink?

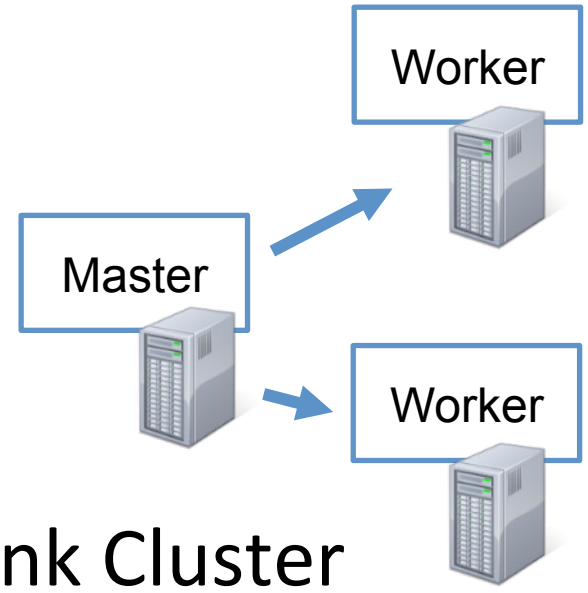
Analytical Program

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

DataSet<Tuple2<String, Integer>> result = text
        .flatMap((str, out) -> {
            for (String token : value.split("\\W")) {
                out.collect(new Tuple2<>(token, 1));
            })
        .groupBy(0)
        .aggregate(SUM, 1);
```



Flink Client & Optimizer





Flink Cluster

This Talk

- Introduction to Flink
- API overview
- Distinguishing Flink
- Flink from a user perspective
- Performance
- Flink roadmap and closing

Open source Big Data Landscape

Applications

Hive

Cascading

Mahout

Pig

Crunch

Data processing engines

MapReduce

Flink

Spark

Storm

Tez

App and resource management

Yarn

Mesos

Storage, streams

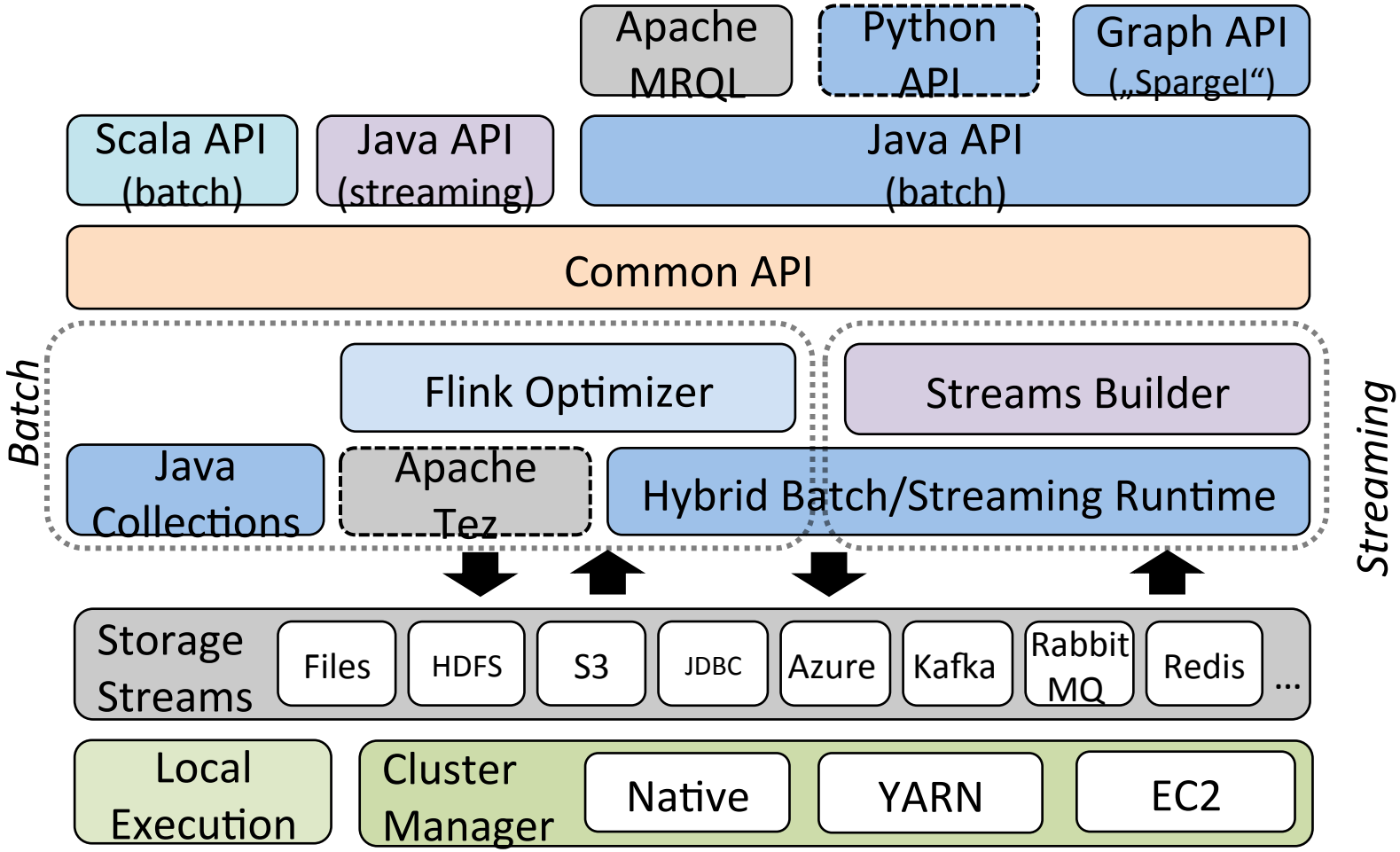
HDFS

HBase

Kafka

...

Flink stack

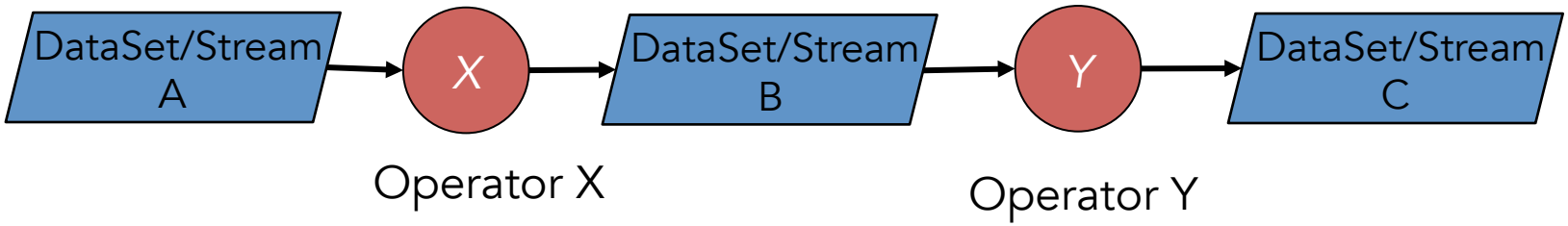

Apache MRQL
Python API
Graph API („Spargel“)
Scala API (batch)
Java API (streaming)
Java API (batch)
Common API
Batch
Flink Optimizer
Streams Builder
Java Collections
Apache Tez
Hybrid Batch/Streaming Runtime
Streaming
Storage Streams
Files
HDFS
S3
JDBC
Azure
Kafka
Rabbit MQ
Redis
...
Local Execution
Cluster Manager
Native
YARN
EC2

Flink APIs

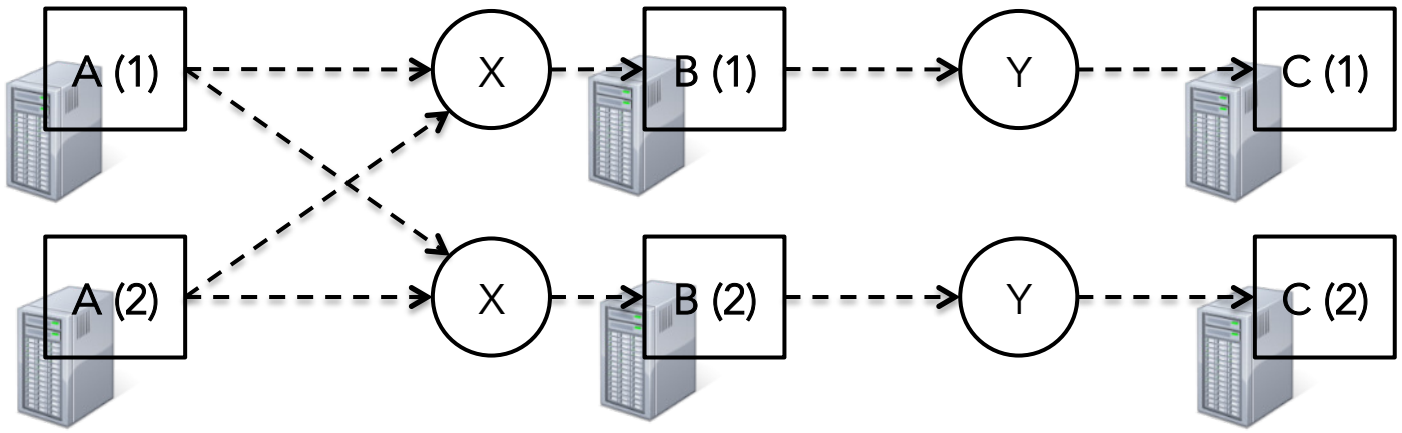
Programming model

Data abstractions: Data Set, Data Stream

Program

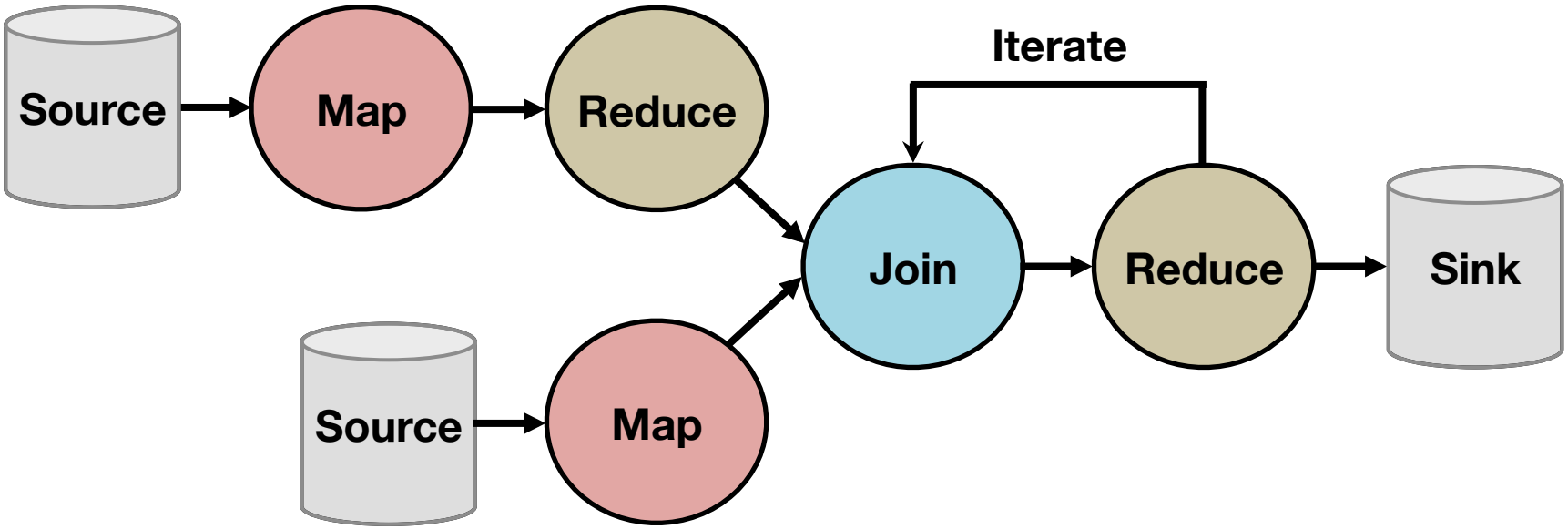


Parallel Execution



Flexible pipelines

Map, FlatMap, MapPartition, Filter, Project, Reduce, ReduceGroup, Aggregate, Distinct, Join, CoGoup, Cross, Iterate, Iterate Delta, Iterate-Vertex-Centric, Windowing



WordCount, Java API

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

DataSet<Tuple2<String, Integer>> result = text
          .flatMap((str, out) -> {
                for (String token : value.split("\\W")) {
                     out.collect(new Tuple2<>(token, 1));
                })
          .groupBy(0)
          .sum(1);
```

WordCount, Scala API

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

WordCount, Streaming API

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

DataStream<Tuple2<String, Integer>> result = text
          .flatMap((str, out) -> {
                for (String token : value.split("\\W")) {
                     out.collect(new Tuple2<>(token, 1));
                })
          .groupBy(0)
          .sum(1);
```

Is there anything beyond WordCount?

Beyond Key/Value Pairs

```
// outputs pairs of pages and impressions

class Impression {
    public String url;
    public long count;
}

class Page {
    public String url;
    public String topic;
}

DataSet<Page> pages = ...;
DataSet<Impression> impressions = ...;

DataSet<Impression> aggregated =
 impressions
  .groupBy("url")
  .sum("count");

pages.join(impressions).where("url").equalTo("url")
     .print()
// outputs pairs of matching pages and impressions
```

Preview: Logical Types

```
DataSet<Row> dates = env.readCsv(...).as("order_id", "date");
DataSet<Row> sessions = env.readCsv(...).as("id", "session");

DataSet<Row> joined = dates
                      .join(session).where("order_id").equals("id");

joined.groupBy("date").reduceGroup(new SessionFilter())
```

```
class SessionFilter implements GroupReduceFunction<SessionType> {

    public void reduce(Iterable<SessionType> value, Collector out){
        ...
    }
}
```

```
public class SessionType {
  public String order_id;
  public Date date;
  public String session;
}
```

Distinguishing Flink

Hybrid batch/streaming runtime

- Batch and stream processing in the same system
- No micro-batches, unified runtime
- Competitive performance
- Code reusable from batch processing to streaming, making development and testing a piece-of-cake

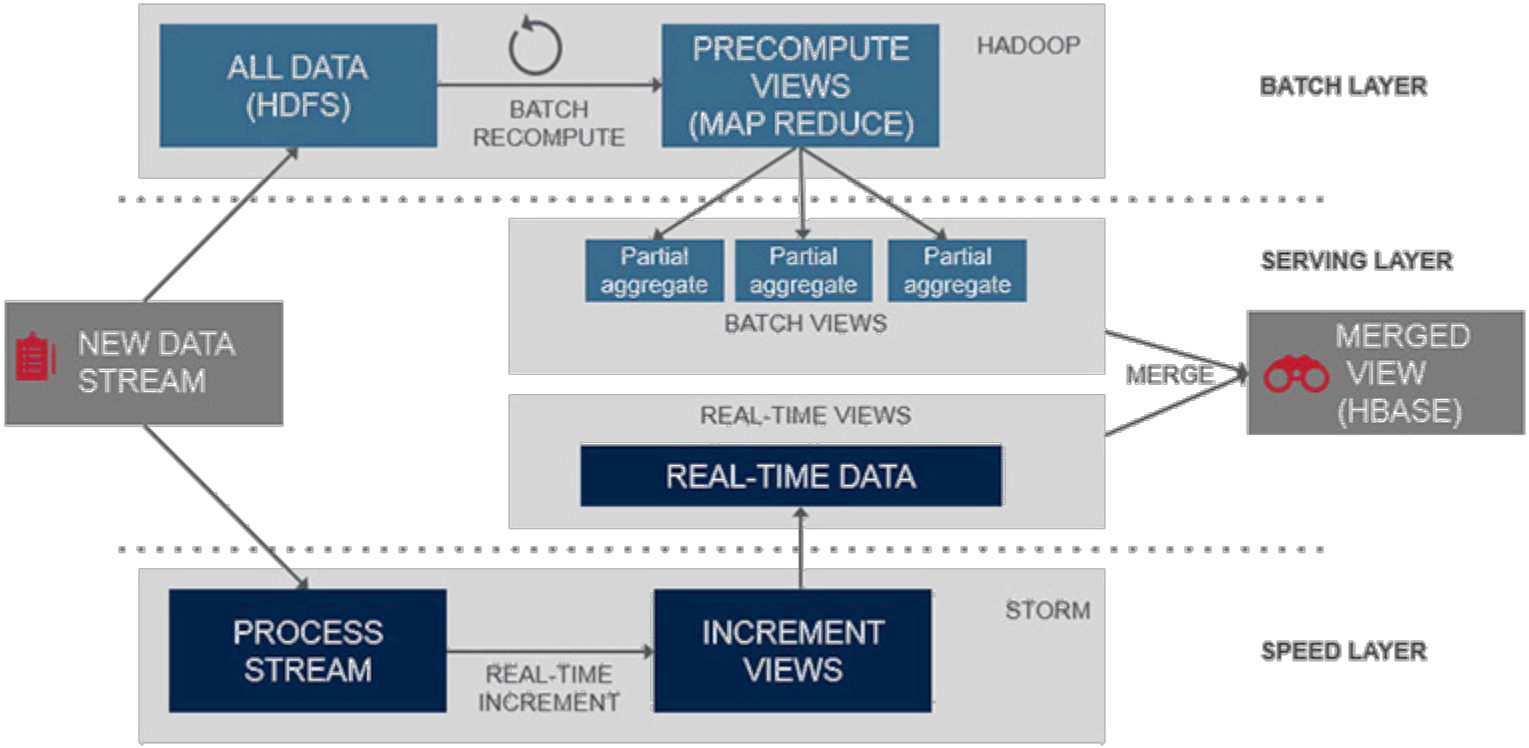
Flink Streaming

- Most Data Set operators are also available for Data Streams
- Temporal and streaming specific operators
  - Window/mini-batch operators
  - Window join, cross etc.
- Support for iterative stream processing
- Connectors for different data sources
  - Kafka, Flume, RabbitMQ, Twitter etc.

Flink Streaming

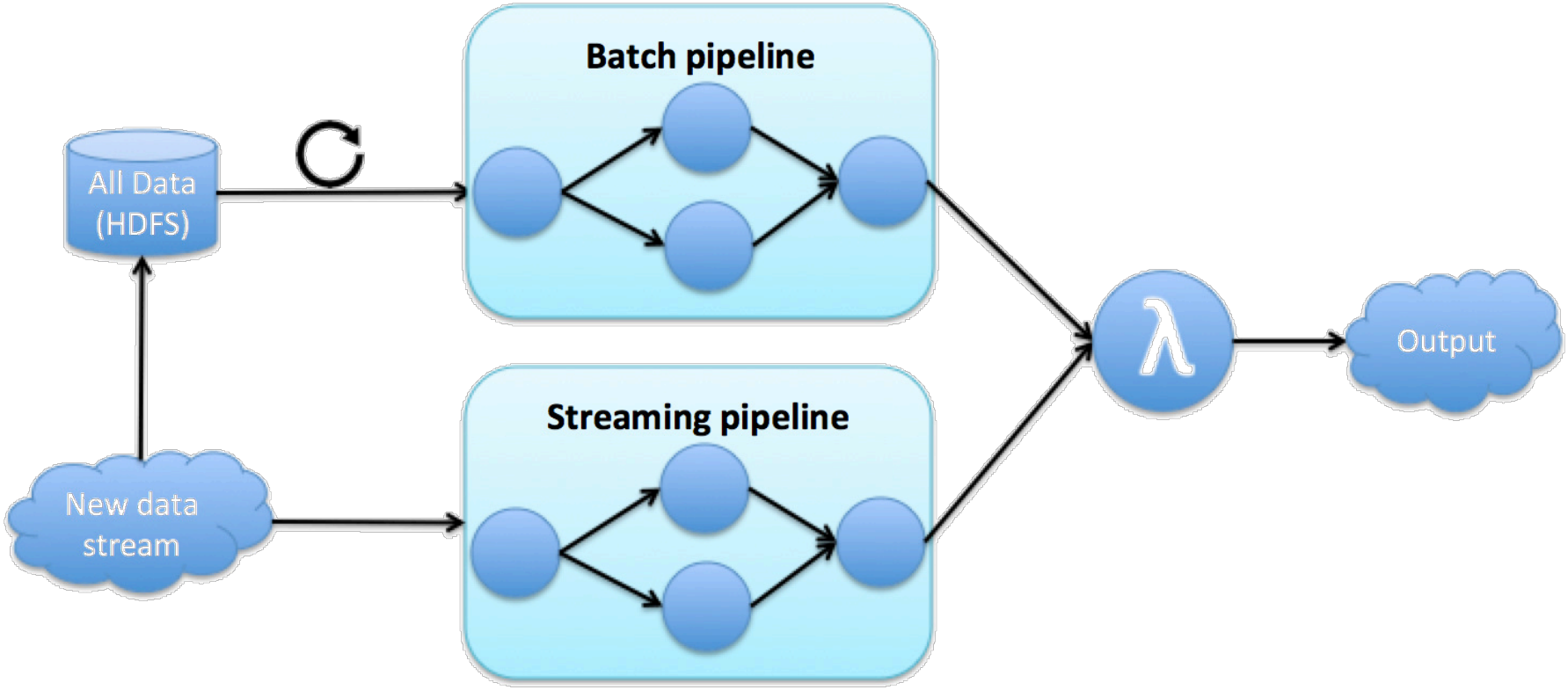
```
//Build new model on every second of new data
DataStream<Double[]> model= env
      .addSource(new TrainingDataSource())
      .window(1000)
      .reduceGroup(new ModelBuilder());


//Predict new data using the most up-to-date model
DataStream<Integer> prediction = env
      .addSource(new NewDataSource())
      .connect(model)
      .map(new Predictor());
```

Lambda architecture



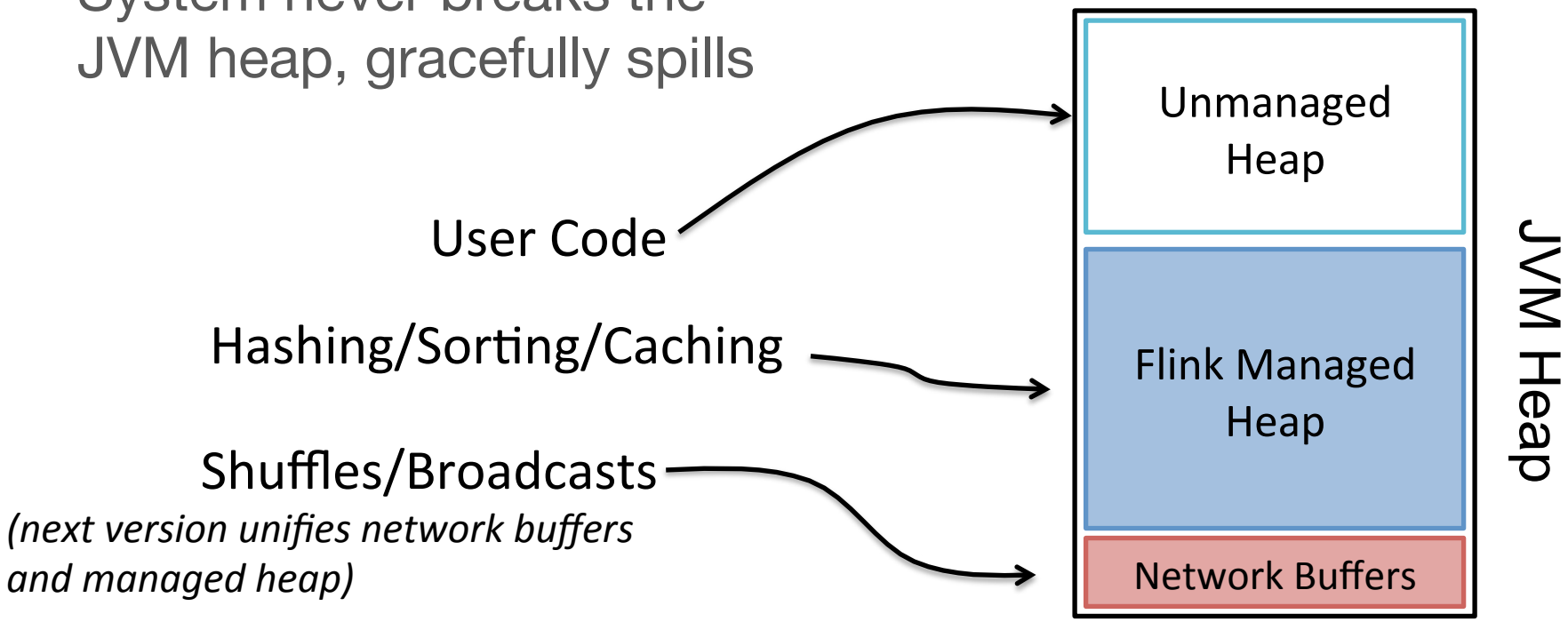
Source: https://www.mapr.com/developercentral/lambda-architecture

Lambda architecture in Flink

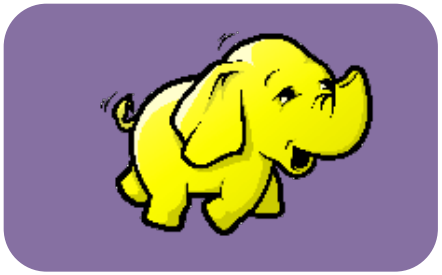


Dependability

- Flink manages its own memory
- Caching and data processing happens in a dedicated memory fraction
- System never breaks the JVM heap, gracefully spills



Operating on Serialized Data



- serializes data every time

→ **Highly robust, never gives up on you**

- works on objects, RDDs may be stored serialized

→**Serialization considered slow, only when needed**

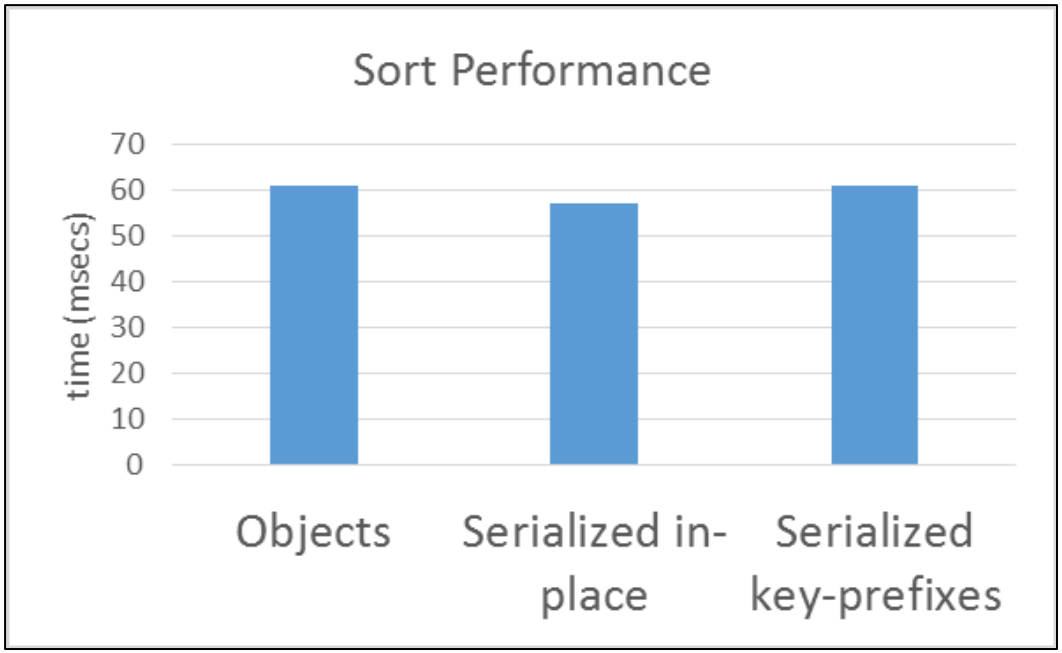
- makes serialization really cheap:

→ partial deserialization, operates on serialized form
→ **Efficient and robust!**

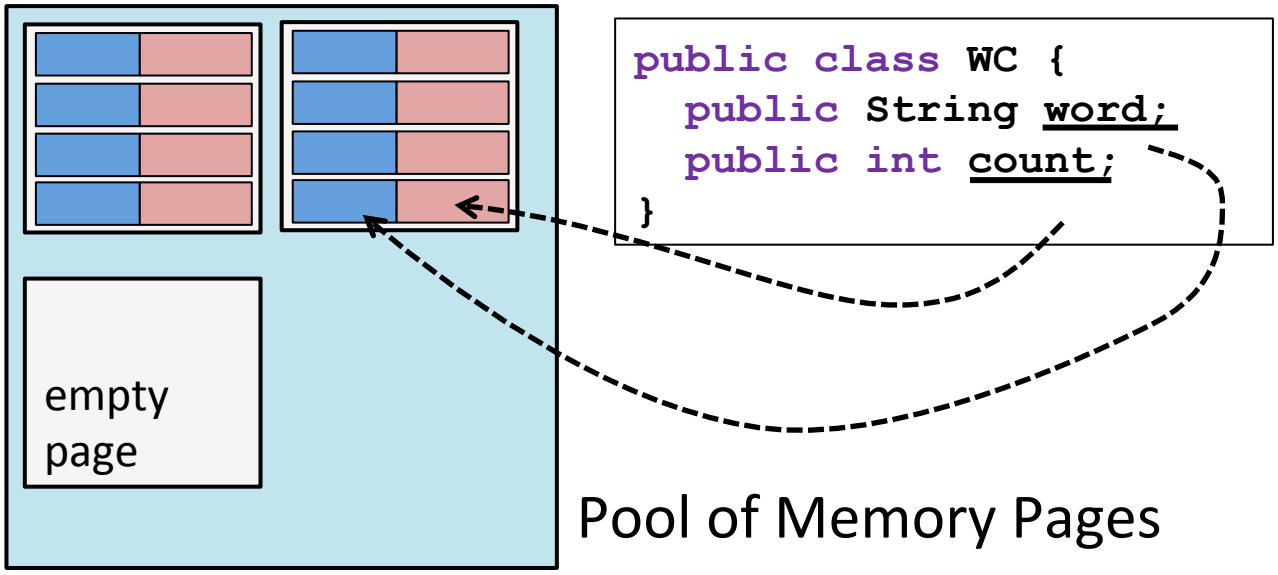
Operating on Serialized Data

Microbenchmark

- Sorting 1GB worth of (long, double) tuples
- 67,108,864 elements
- Simple quicksort



Memory Management



- Works on pages of bytes, maps objects transparently
- Full control over memory, out-of-core enabled
- Algorithms work on binary representation
- Address individual fields (not deserialize whole object)
- Move memory between operations

Flink from a user perspective

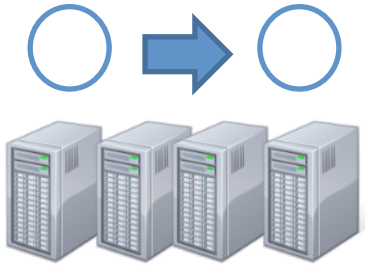
Flink programs run everywhere

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

DataSet<Tuple2<String, Integer>> result = text
          .flatMap((str, out) -> {
               for (String token : value.split("\\W")) {
                    out.collect(new Tuple2<>(token, 1));
               })
          .groupBy(0)
          .aggregate(SUM, 1);
```

Local
Debugging

As Java Collection Programs

Embedded
(e.g., Web Container)



Cluster (Batch)

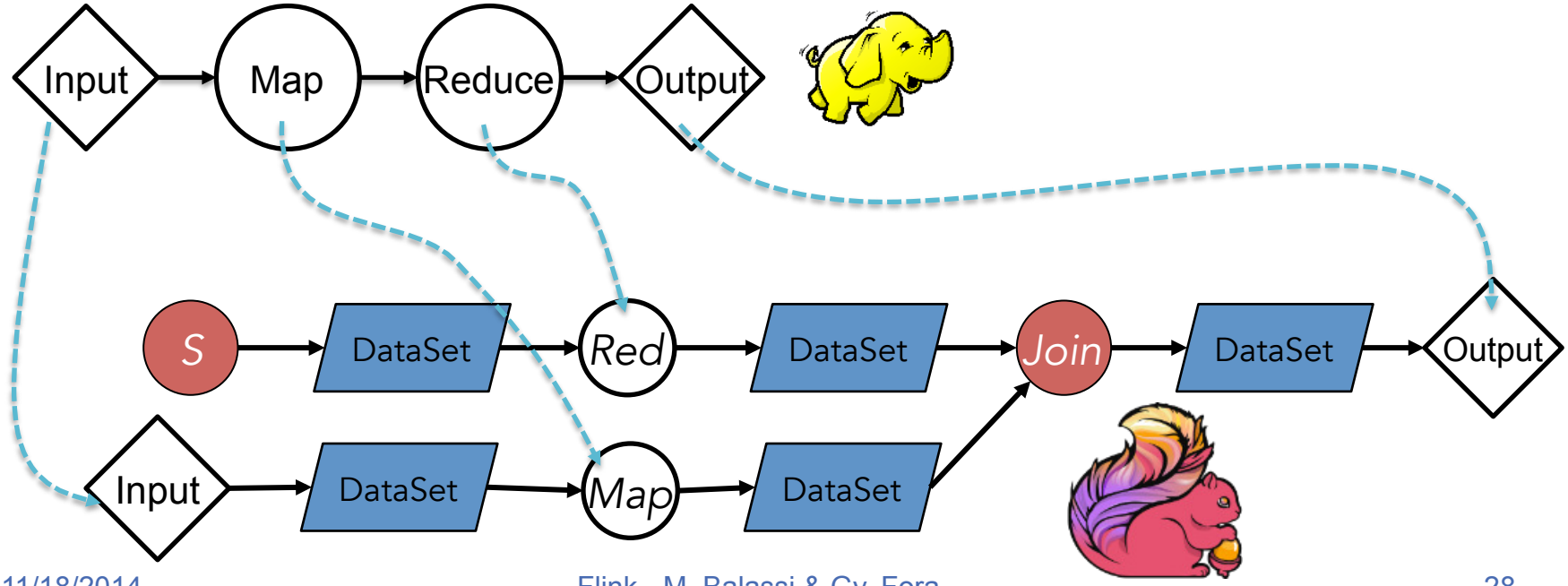
Fink Runtime or Apache Tez

Cluster (Streaming)

Migrate Easily

Flink out-of-the-box supports

- Hadoop data types (writables)
- Hadoop Input/Output Formats
- Hadoop functions and object model

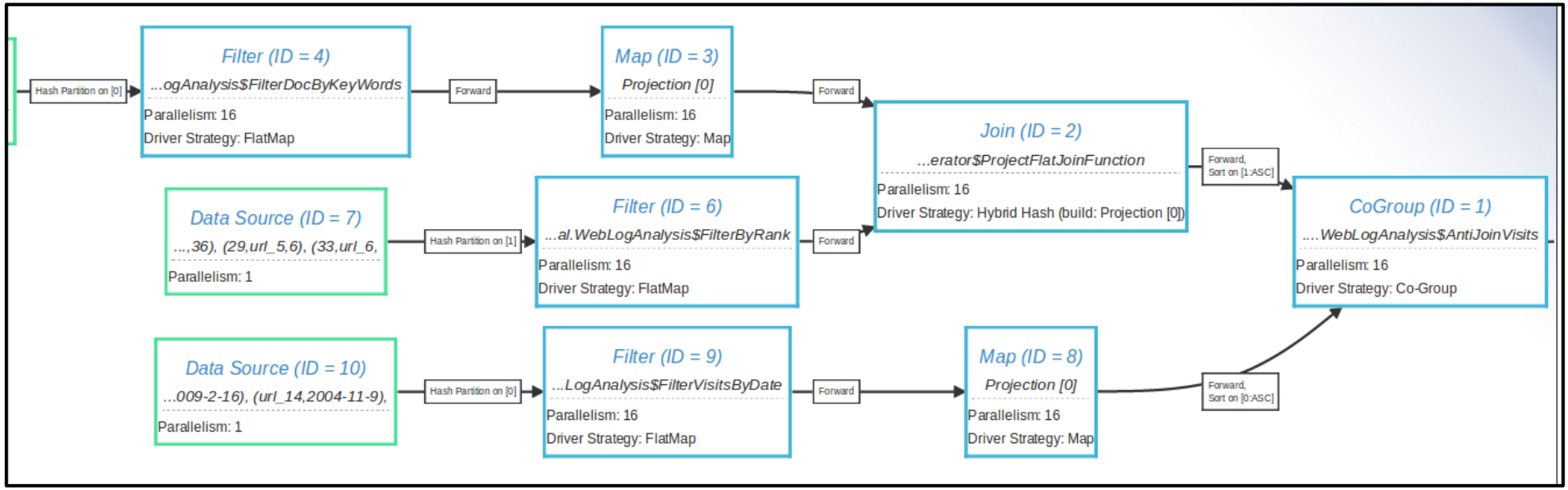


Little tuning or configuration required

- Requires no memory thresholds to configure
  - Flink manages its own memory
- Requires no complicated network configs
  - Pipelining engine requires much less memory for data exchange
- Requires no serializers to be configured
  - Flink handles its own type extraction and data representation
- Programs can be adjusted to data automatically
  - Flink's optimizer can choose execution strategies automatically

Understanding Programs

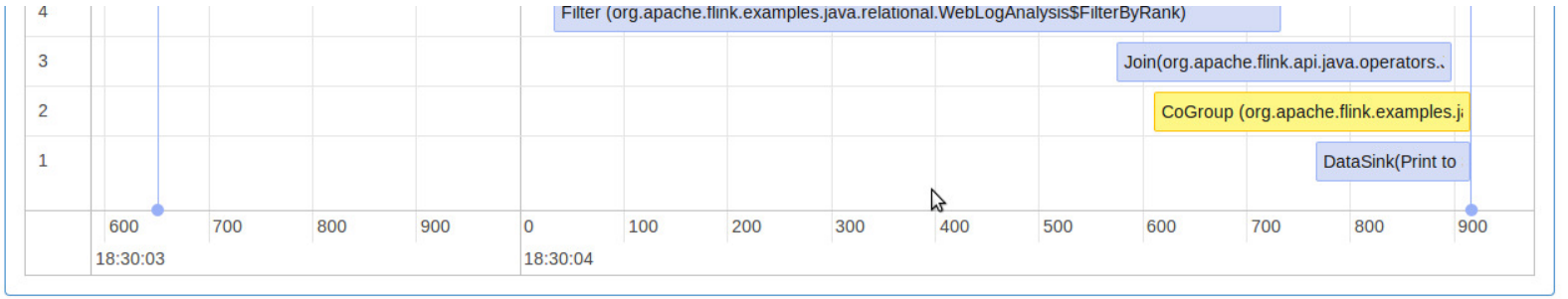
Visualizes the operations and the data movement of programs

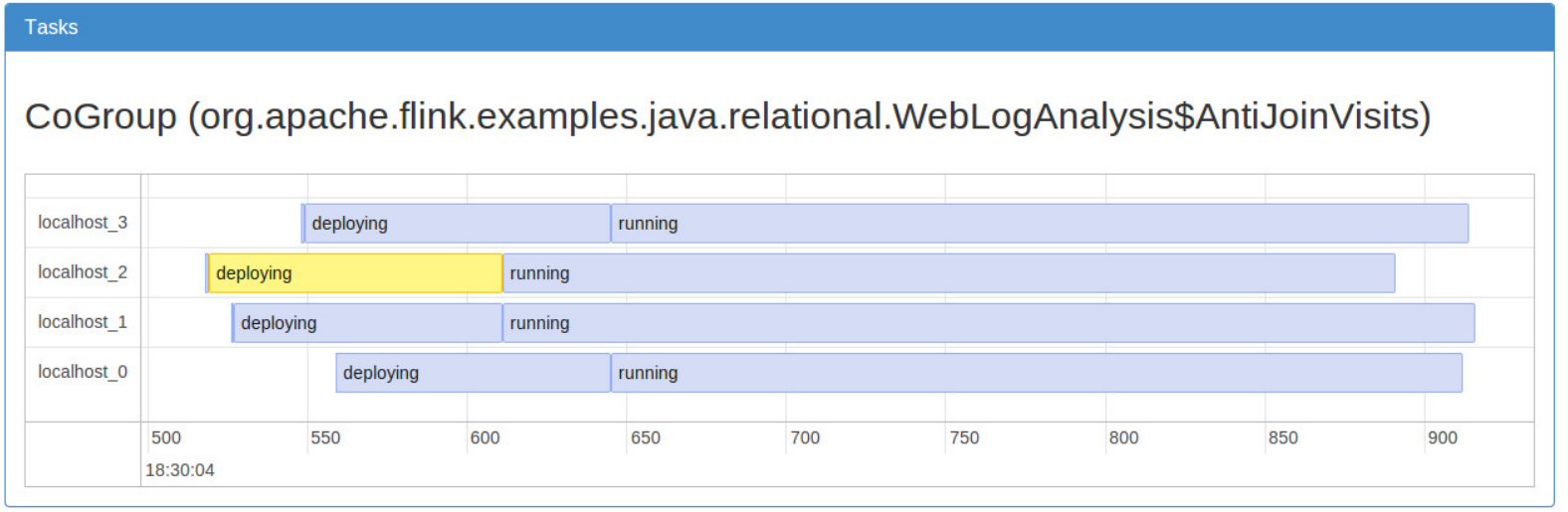


Screenshot from Flink's plan visualizer

Understanding Programs

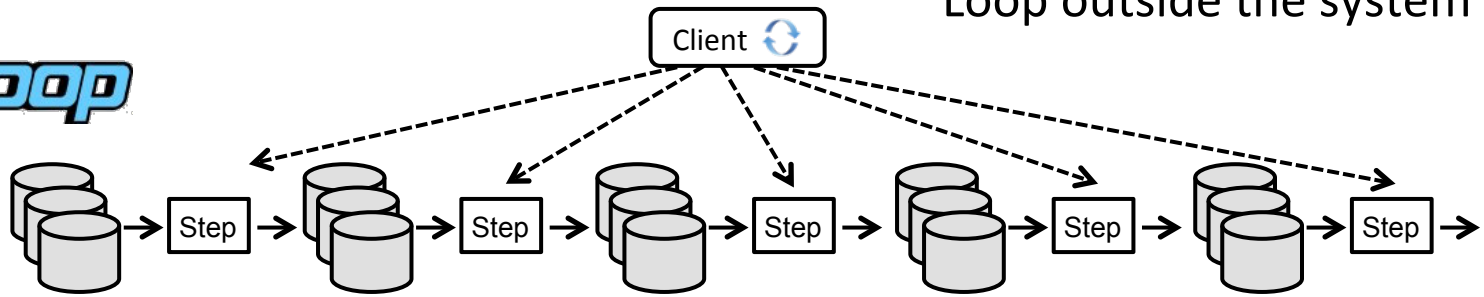
Analyze after execution (times, stragglers, …)





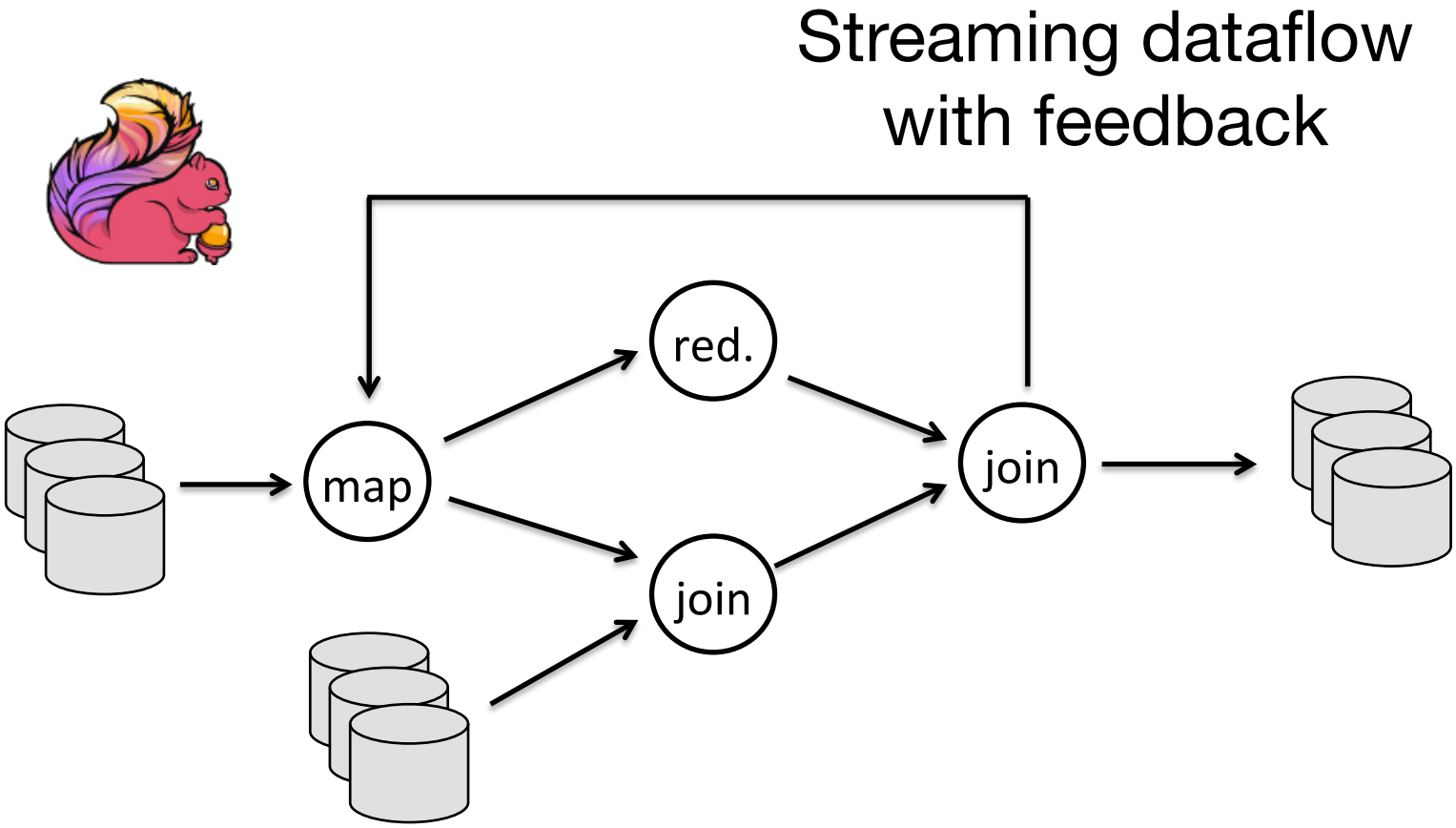
Iterations in other systems

Loop outside the system



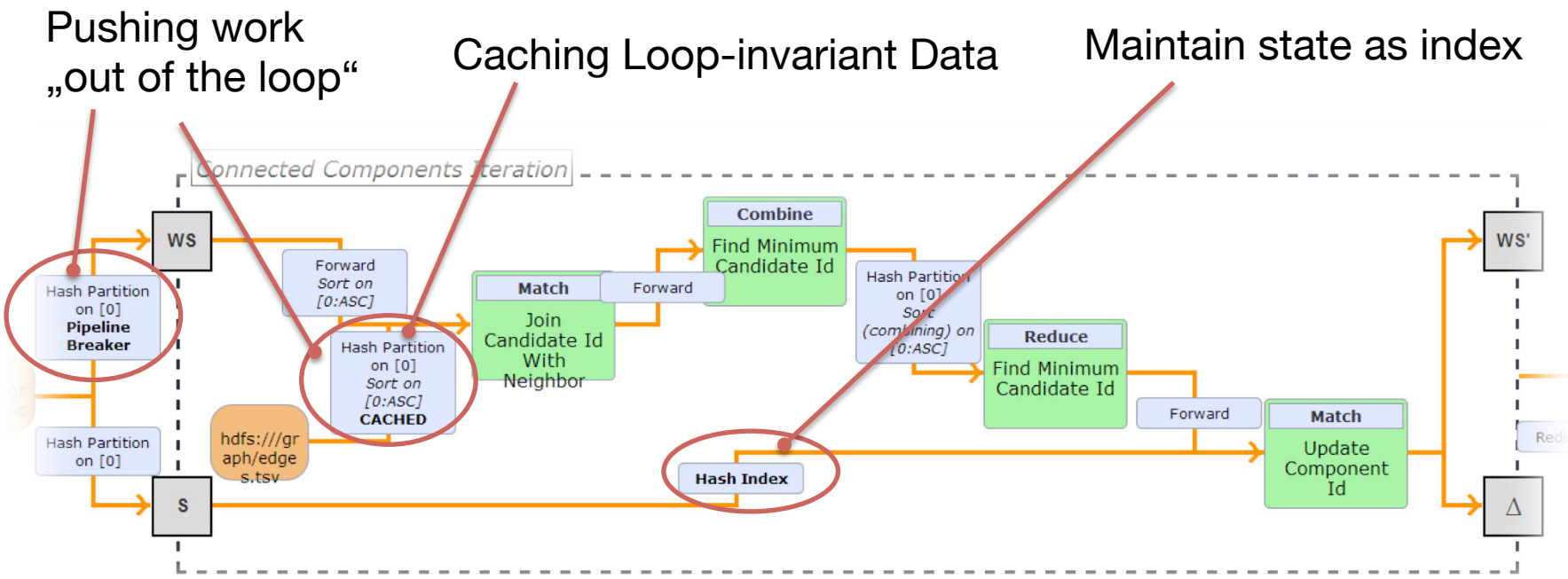
Loop outside the system

Iterations in Flink



System is iteration-aware, performs automatic optimization

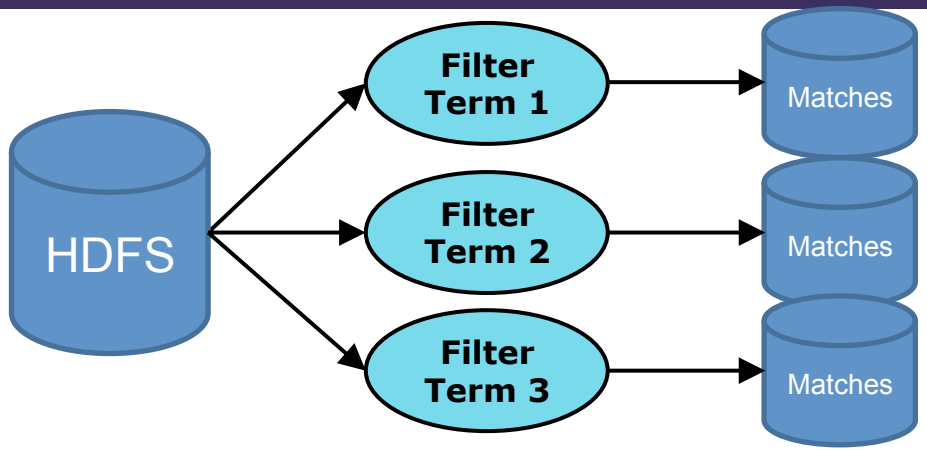
Automatic Optimization for Iterative Programs

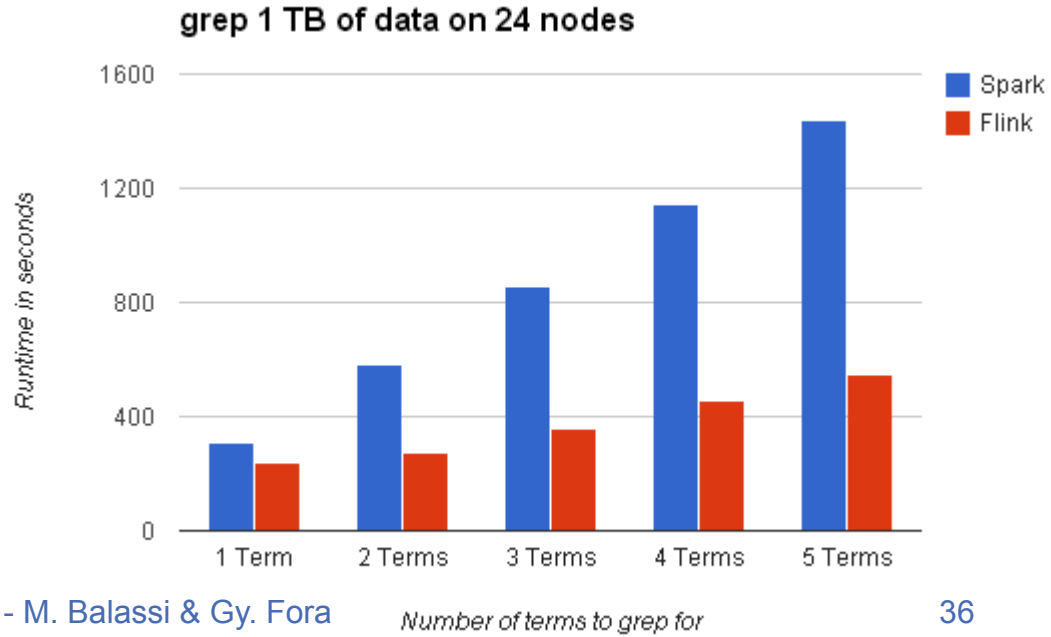


Performance

Distributed Grep

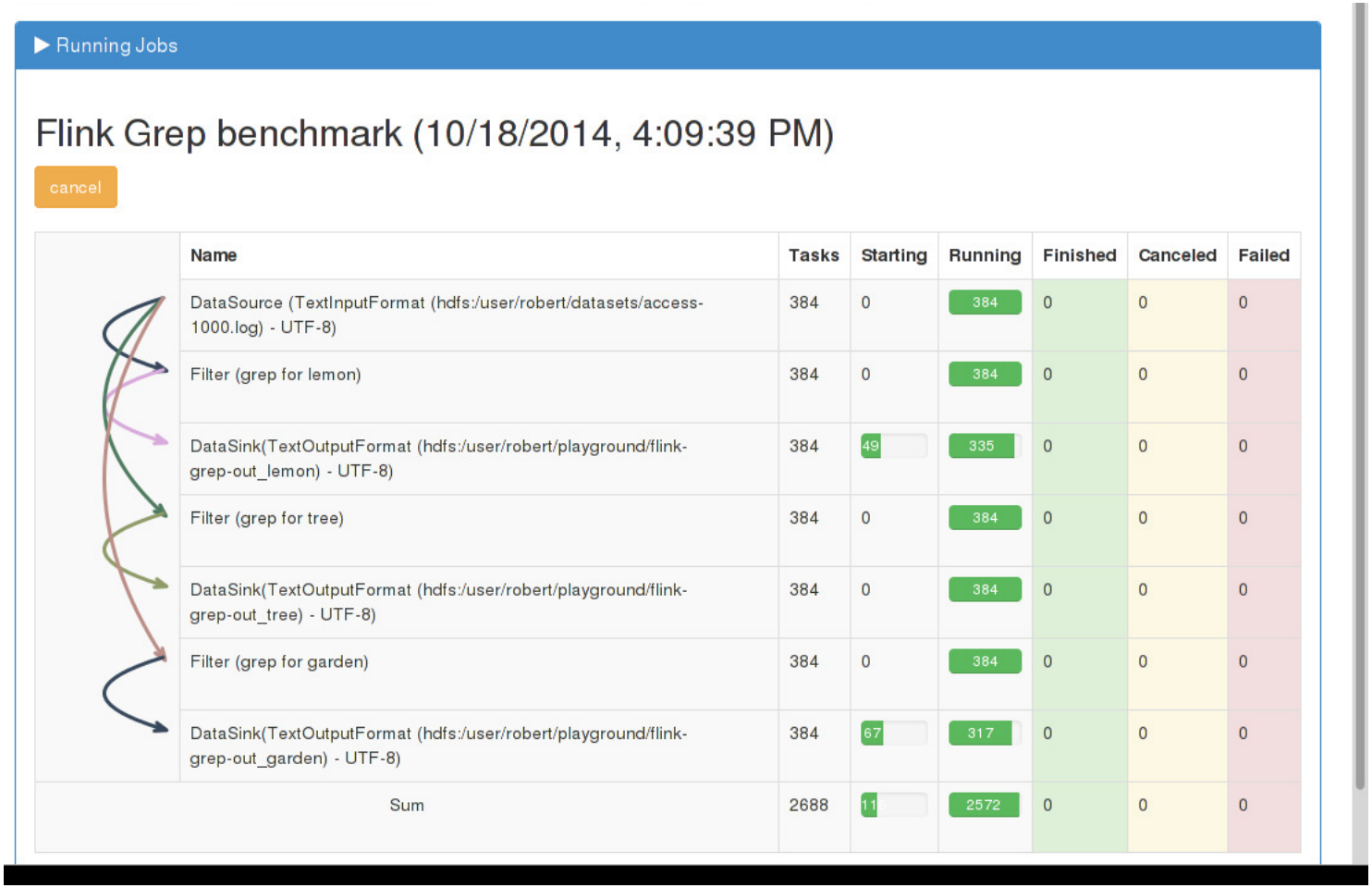
- 1 TB of data (log files)
- 24 machines with
  - 32 GB Memory
  - Regular HDDs
  - HDFS 2.4.0
- Flink 0.7-incubating-SNAPSHOT
- Spark 1.2.0-SNAPSHOT





➔Flink up to 2.5x faster

Distributed Grep: Flink Execution



▶ Running Jobs

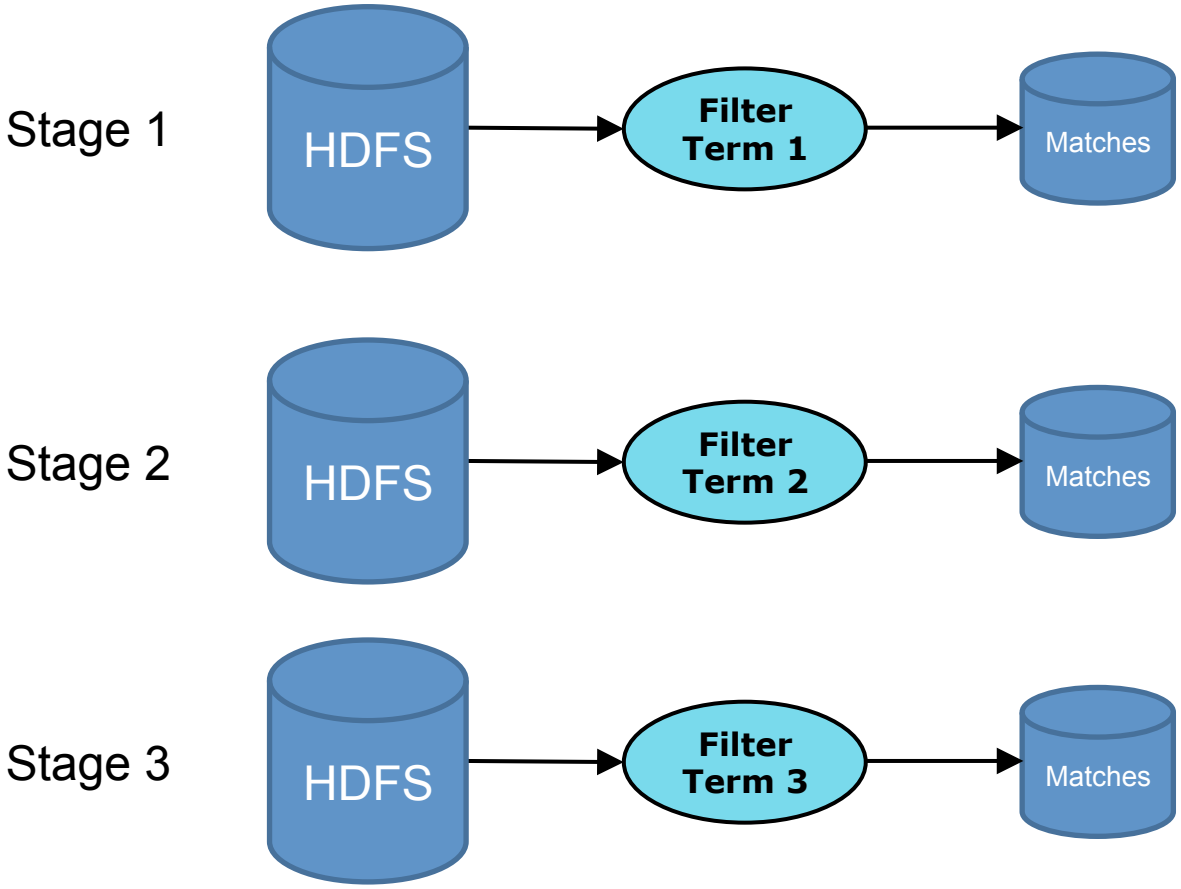
Flink Grep benchmark (10/18/2014, 4:09:39 PM)

cancel

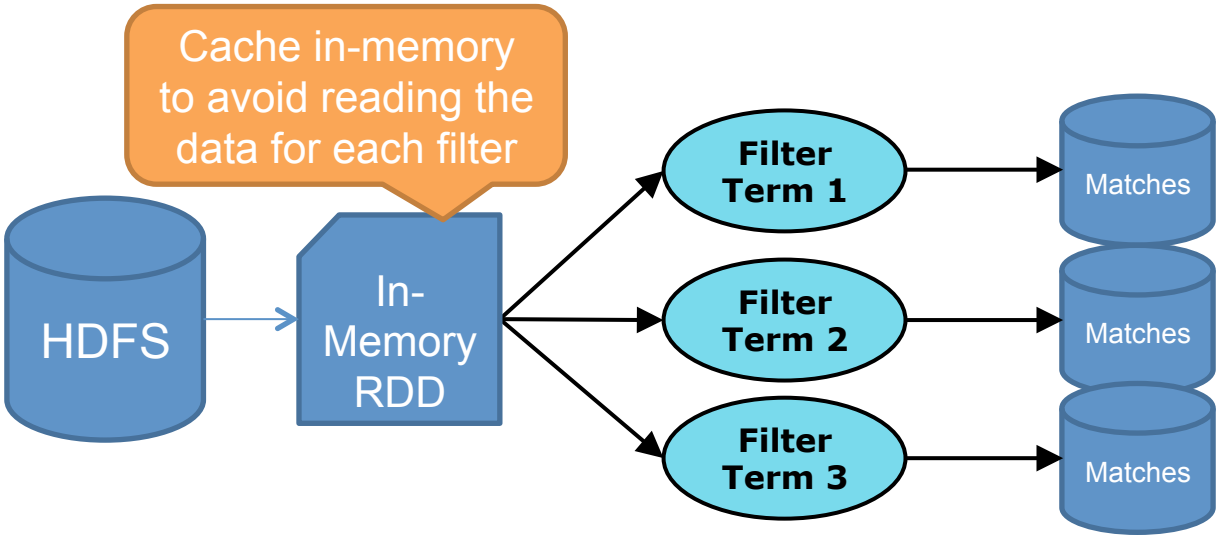
Name	Tasks	Starting	Running	Finished	Canceled	Failed
DataSource (TextInputFormat (hdfs:/user/robert/datasets/access-1000.log) - UTF-8)	384	0	384	0	0	0
Filter (grep for lemon)	384	0	384	0	0	0
DataSink(TextOutputFormat (hdfs:/user/robert/playground/flink-grep-out_lemon) - UTF-8)	384	49	335	0	0	0
Filter (grep for tree)	384	0	384	0	0	0
DataSink(TextOutputFormat (hdfs:/user/robert/playground/flink-grep-out_tree) - UTF-8)	384	0	384	0	0	0
Filter (grep for garden)	384	0	384	0	0	0
DataSink(TextOutputFormat (hdfs:/user/robert/playground/flink-grep-out_garden) - UTF-8)	384	67	317	0	0	0
Sum	2688	11	2572	0	0	0

Distributed Grep: Spark Execution

Spark executes the job in 3 stages:



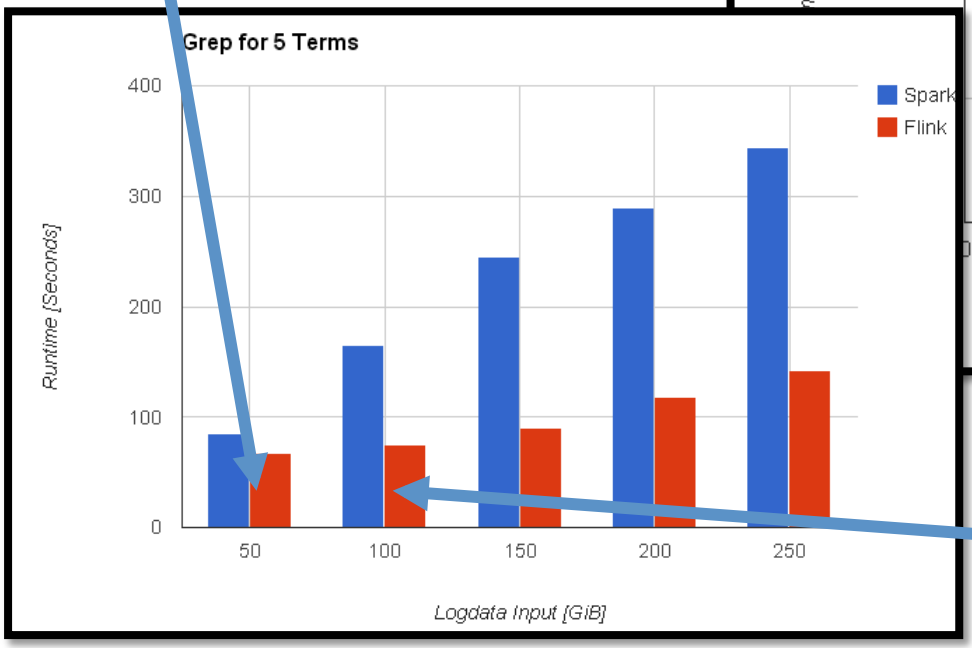
Spark in-memory pinning




```
JavaSparkContext sc = new JavaSparkContext(conf);
JavaRDD<String> file =
sc.textFile(inFile).persist(StorageLevel.MEMORY_AND_DISK());
for(int p = 0; p < patterns.length; p++) {
    final String pattern = patterns[p];
    JavaRDD<String> res = file.filter(new Function<String, Boolean>() { … }); }
```

Spark in-memory pinning

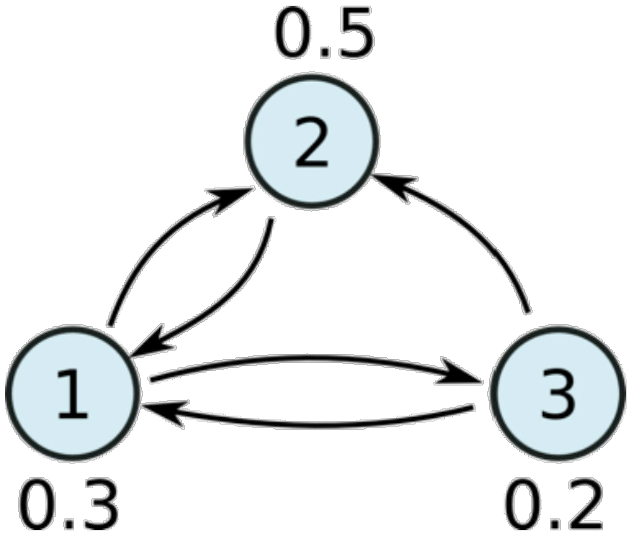
RDD is 100% in-memory



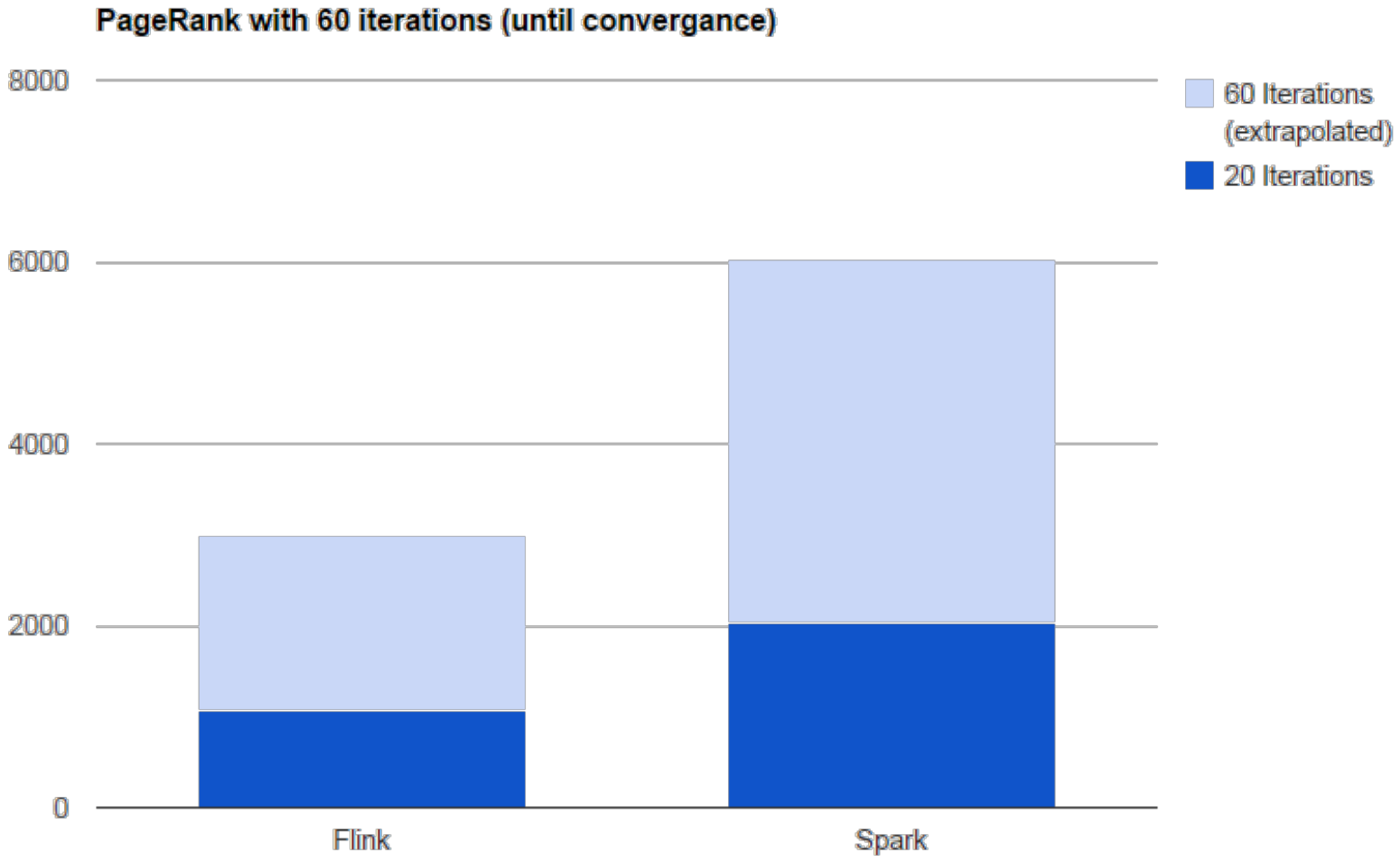
Spark starts spilling RDD to disk

PageRank

- Dataset:
  - Twitter Follower Graph
  - 41,652,230 vertices (users)
  - 1,468,365,182 edges (followings)
  - 12 GB input data



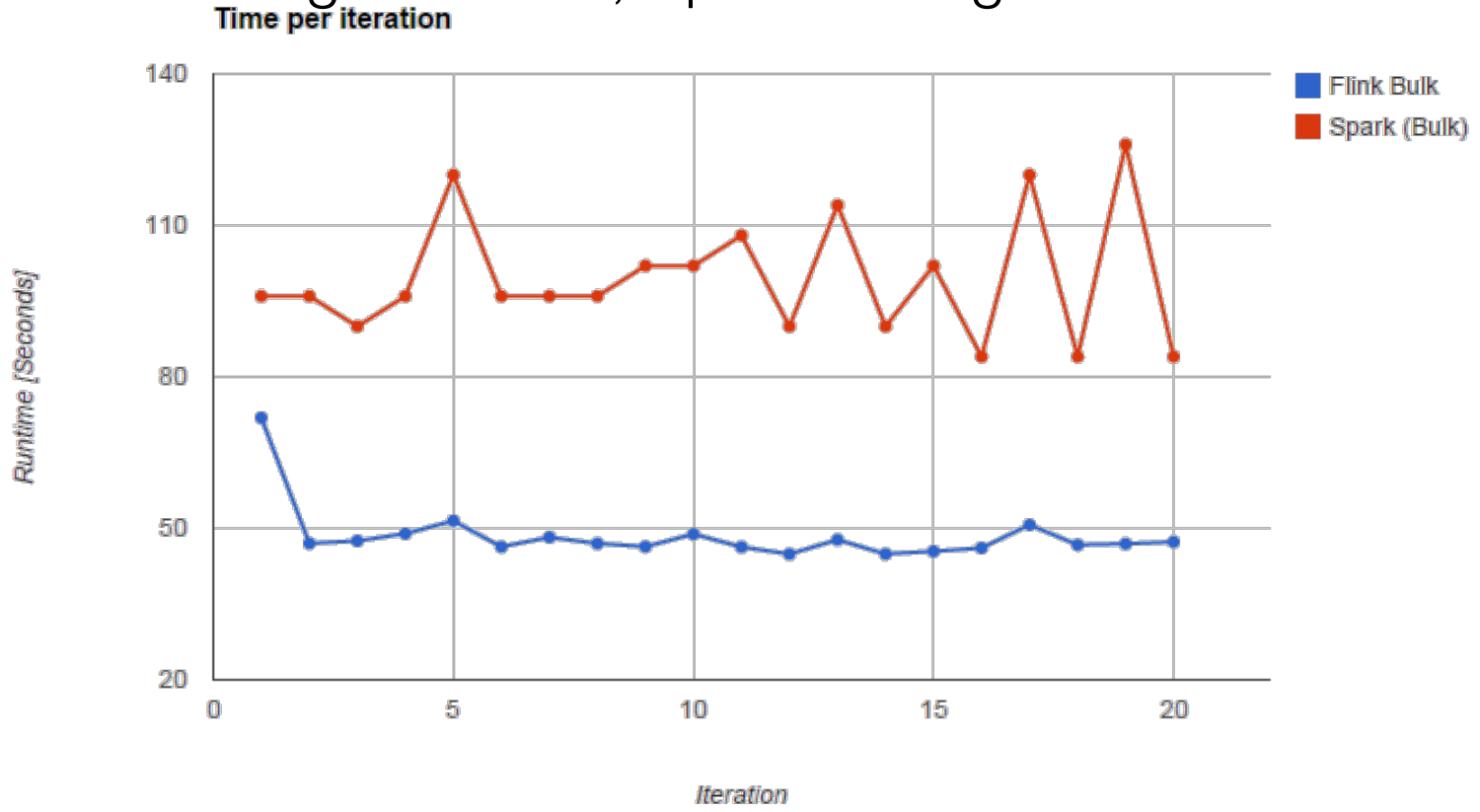
PageRank results



Why is there a difference?

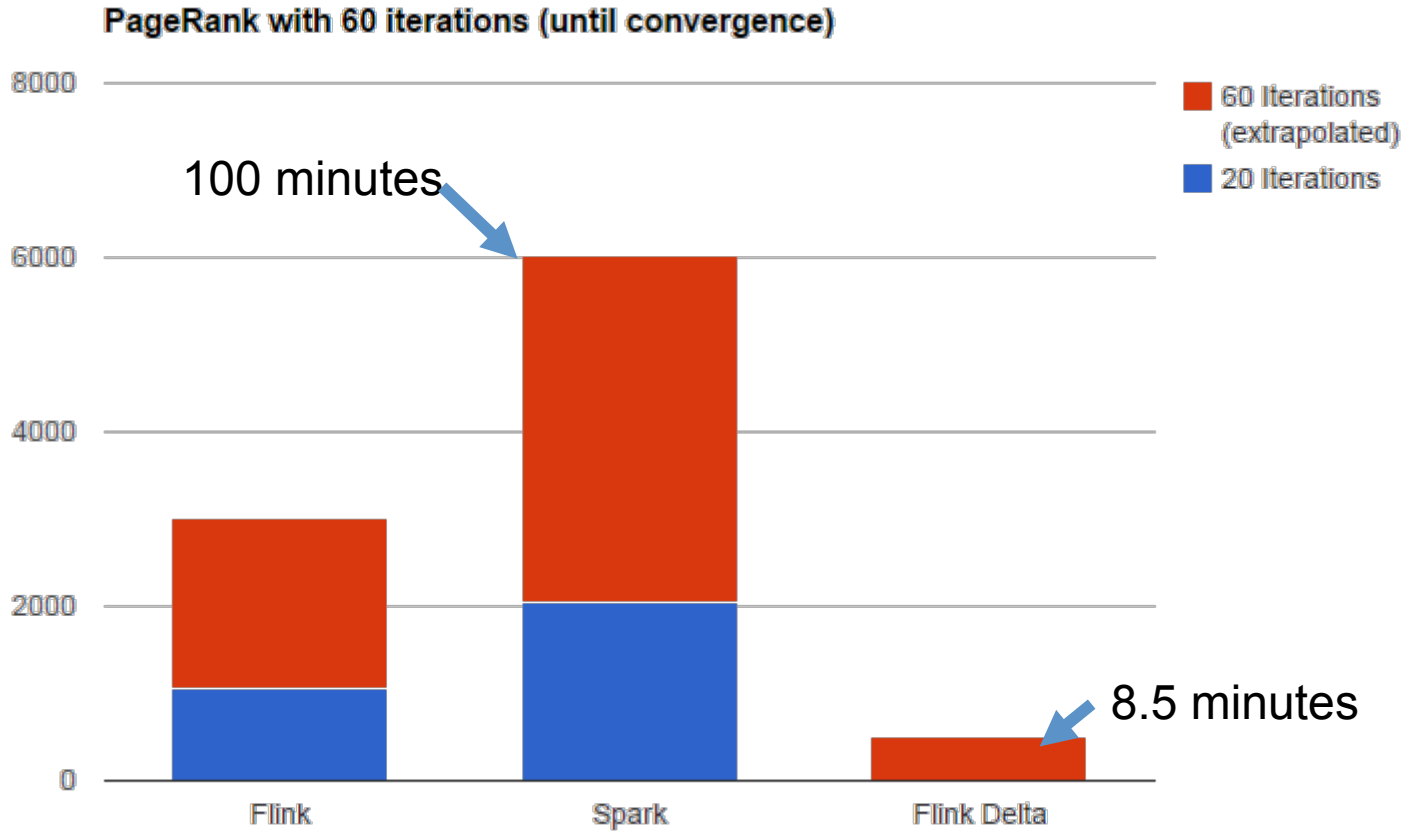
- Lets have a look at the iteration times:

  Flink average: 48 sec., Spark average: 99 sec.



PageRank on Flink with Delta Iterations

- The algorithm runs 60 iterations until convergence (runtime includes convergence check)

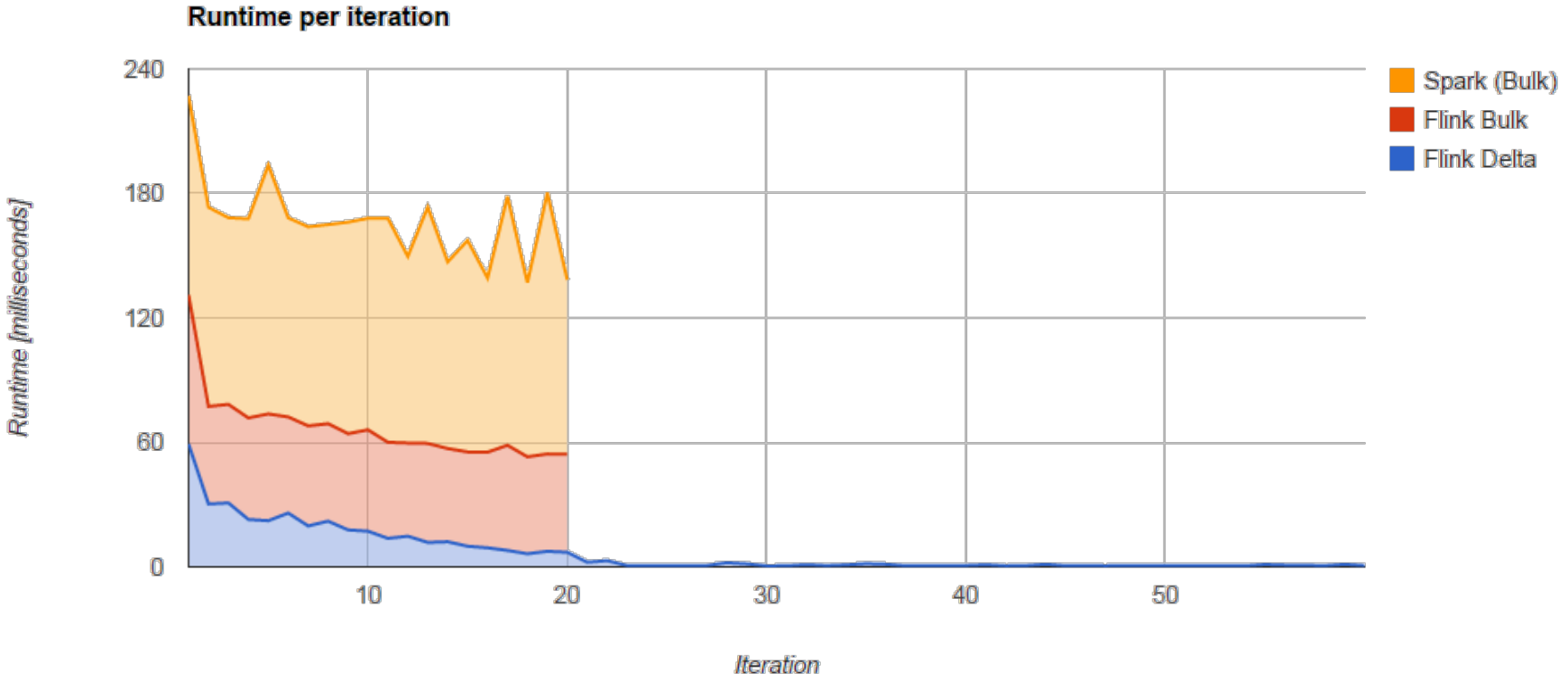


Again, explaining the difference

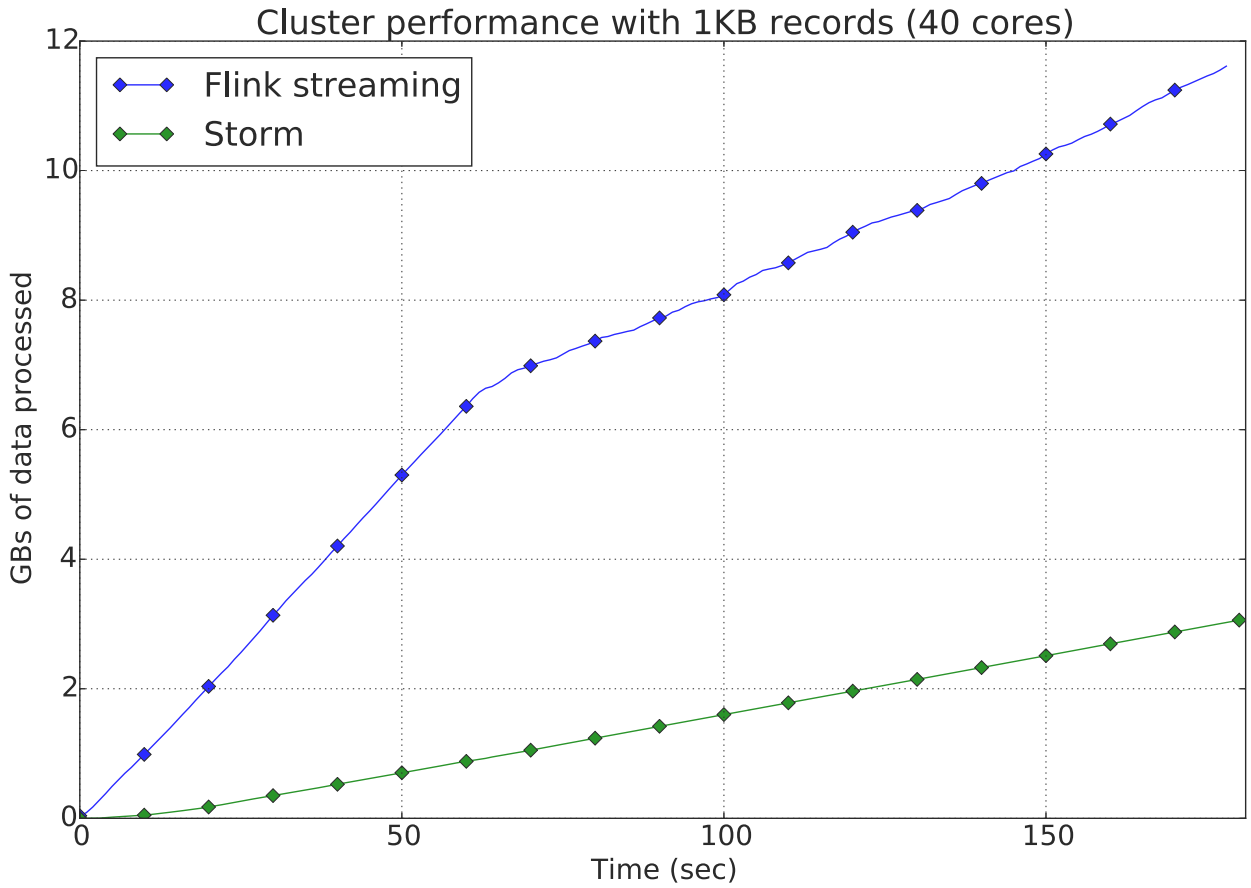
On average, a (delta) interation runs for 6.7 seconds

Flink (Bulk): 48 sec.

Spark (Bulk): 99 sec.



Streaming performance



Flink Roadmap

- Flink has a major release every 3 months, with >=1 big-fixing releases in-between
- Finer-grained fault tolerance
- Logical (SQL-like) field addressing
- Python API
- Flink Streaming, Lambda architecture support
- Flink on Tez
- ML on Flink (e.g., Mahout DSL)
- Graph DSL on Flink
- … and more


dataArtisans

http://flink.incubator.apache.org

github.com/apache/incubator-flink

@ApacheFlink

Marton Balassi, Gyula Fora
{mbalassi, gyfora}@apache.org

APACHECON EUROPE

CORINTHIA HOTEL
BUDAPEST, HUNGARY
NOVEMBER 17-21, 2014