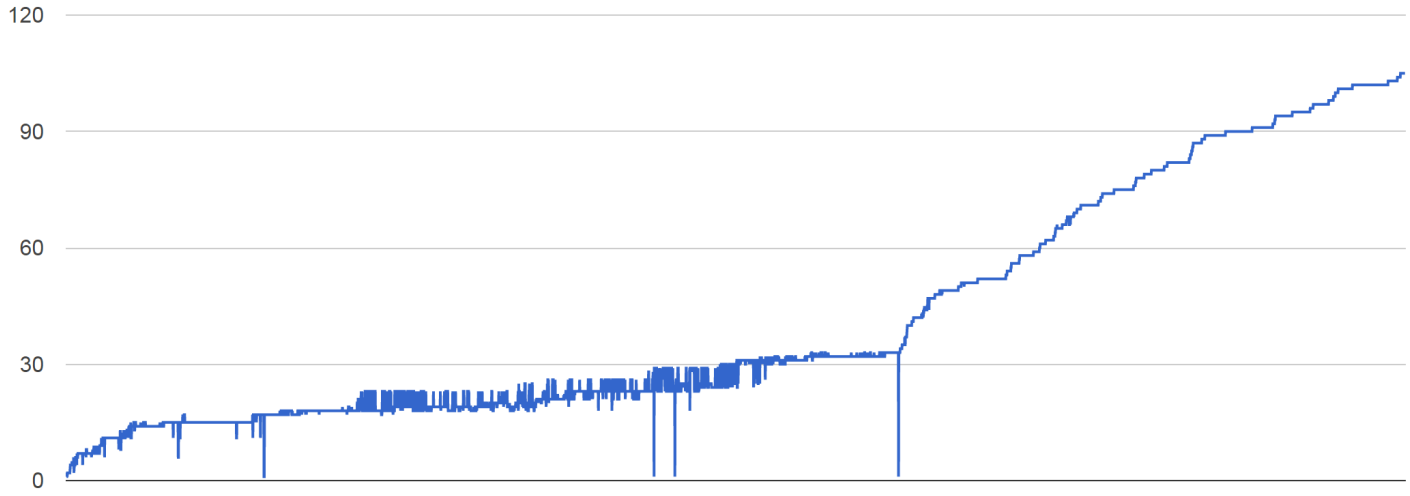
Flink internals

Kostas Tzoumas
Flink committer &
Co-founder, data Artisans
ktzoumas@apache.org
@kostas_tzoumas

Welcome

- **Last talk:** how to program PageRank in Flink, and Flink programming model
- **This talk:** how Flink works internally
- Again, a big bravo to the Flink community



Recap: Using Flink

DataSet and transformations


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

DataSet<String> first = input
    .filter (str -> str.contains("Apache Flink"));
DataSet<String> second = first
    .filter (str -> str.length() > 40);

second.print()
env.execute();
```

Available transformations

- map
- flatMap
- filter
- reduce
- reduceGroup
- join
- coGroup
- aggregate
- cross
- project
- distinct
- union
- iterate
- iterateDelta
- repartition
- …

Other API elements & tools

- Accumulators and counters
  - Int, Long, Double counters
  - Histogram accumulator
  - Define your own
- Broadcast variables
- Plan visualization
- Local debugging/testing mode

Data types and grouping

```
public static class Access {
  public int userId;
  public String url;
  ...
}
```

```
public static class User {
  public int userId;
  public int region;
  public Date customerSince;
  ...
}
```

```
DataSet<Tuple2<Access,User>> campaign = access.join(users)
  .where("userId").equalTo("userId")

DataSet<Tuple3<Integer,String,String> someLog;
someLog.groupBy(0,1).reduceGroup(...);
```

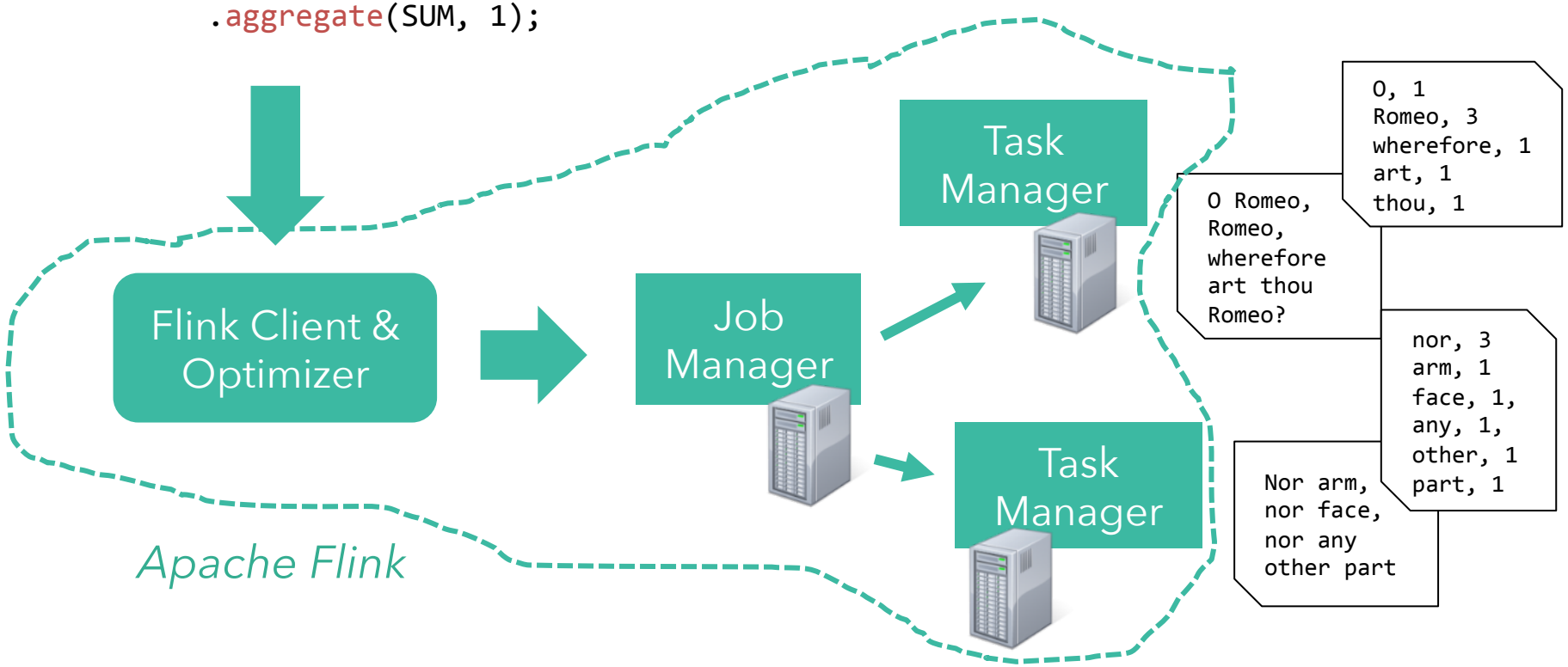
- Bean-style Java classes & field names
- Tuples and position addressing
- Any data type with key selector function

Other API elements

- Hadoop compatibility
  - Supports all Hadoop data types, input/output formats, Hadoop mappers and reducers
- Data streaming API
  - `DataStream` instead of `DataSet`
  - Similar set of operators
  - Currently in alpha but moving very fast
- Scala and Java APIs (mirrored)
- Graph API (Spargel)

Intro to internals

```
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);
```



If you want to know **one** thing about Flink is that you don’t need to know the internals of Flink.

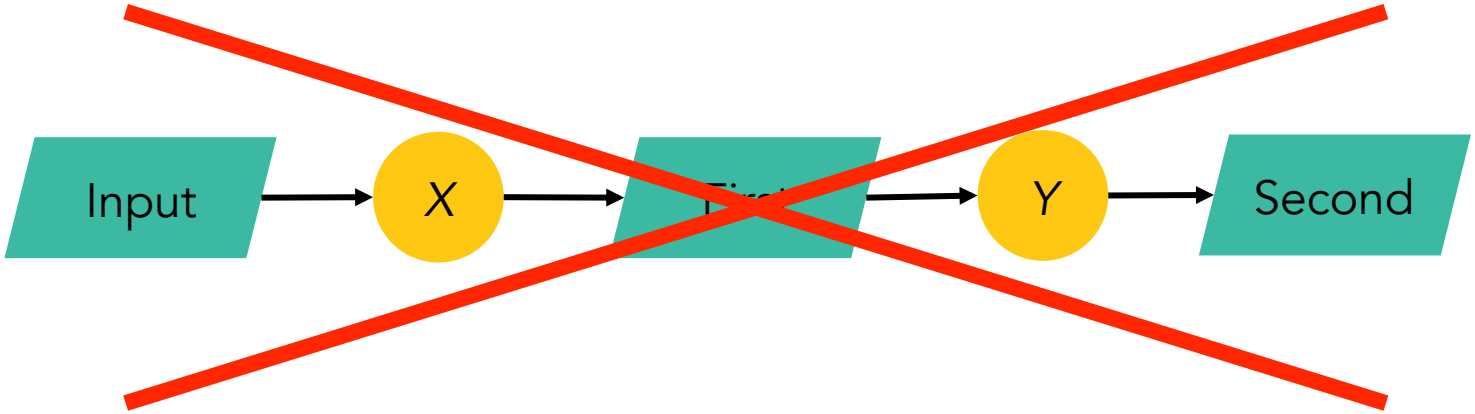
Philosophy

- Flink “hides” its internal workings from the user
- This is **good**
  - User does not worry about how jobs are executed
  - Internals can be changed without breaking changes
- … and **bad**
  - Execution model more complicated to explain compared to MapReduce or Spark RDD

Recap: DataSet


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

DataSet<String> first = input
    .filter (str -> str.contains("Apache Flink"));
DataSet<String> second = first
    .filter (str -> str.length() > 40);

second.print()
env.execute();
```

Common misconception

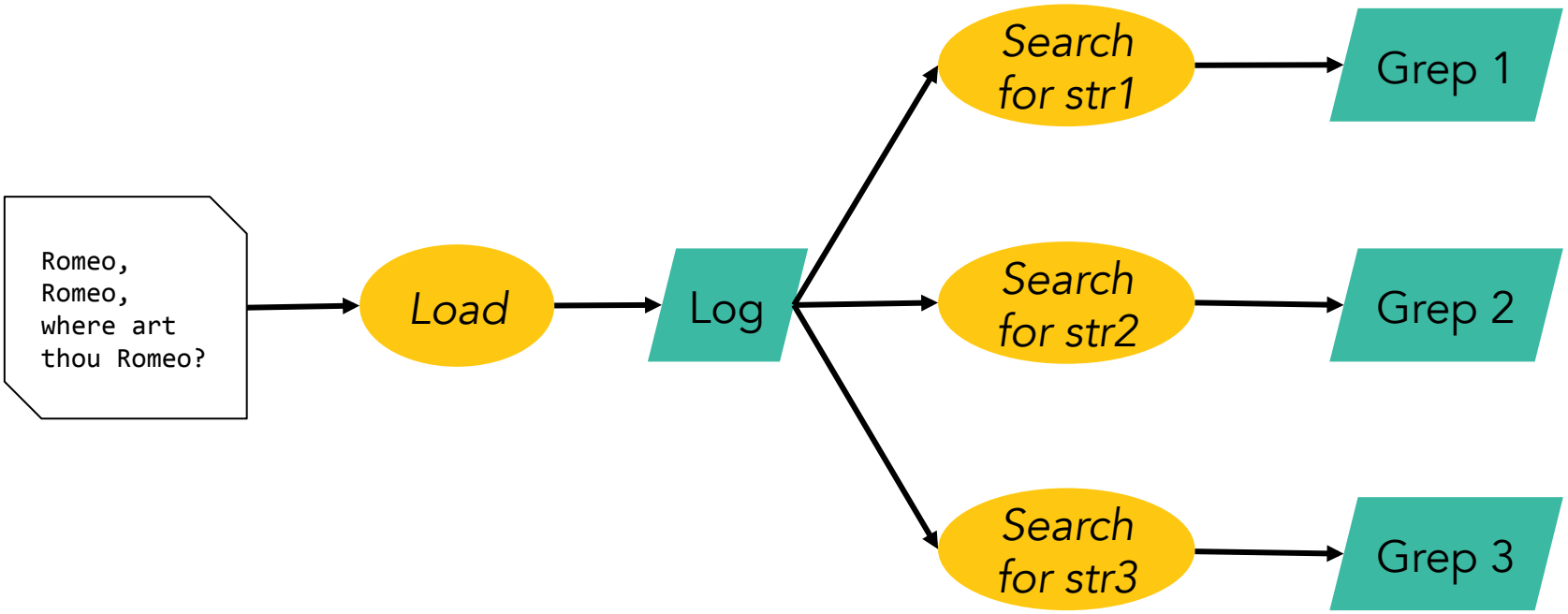


- Programs are not executed eagerly
- Instead, system compiles program to an execution plan and executes that plan

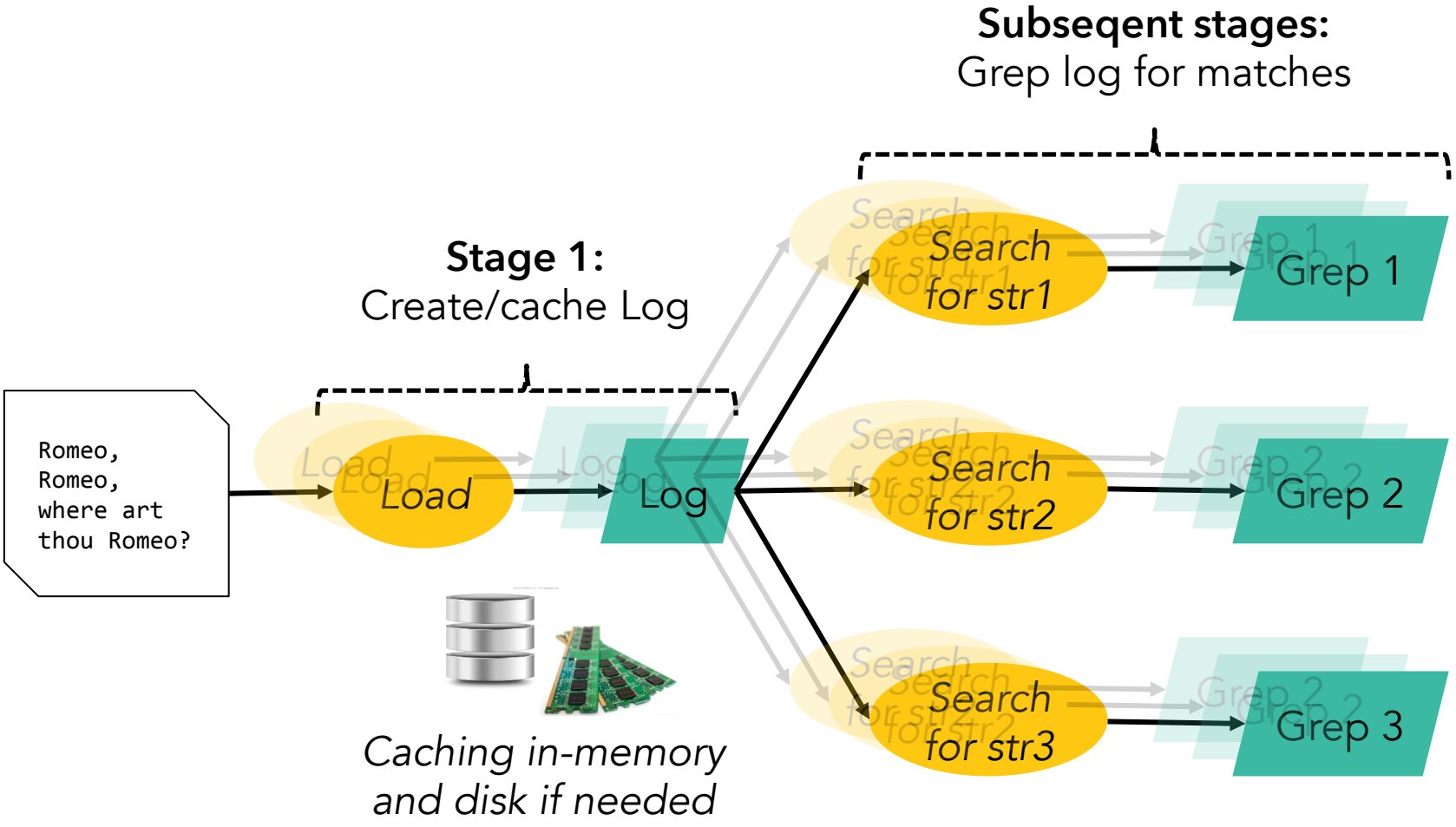
DataSet<String>

- Think of it as a PCollection<String>, or a Spark RDD[String]
- With a major difference: it can be produced/ recovered in several ways
  - … like a Java collection
  - … like an RDD
  - … perhaps it is never fully materialized (because the program does not need it to)
  - … implicitly updated in an iteration
- And this is transparent to the user

Example: grep

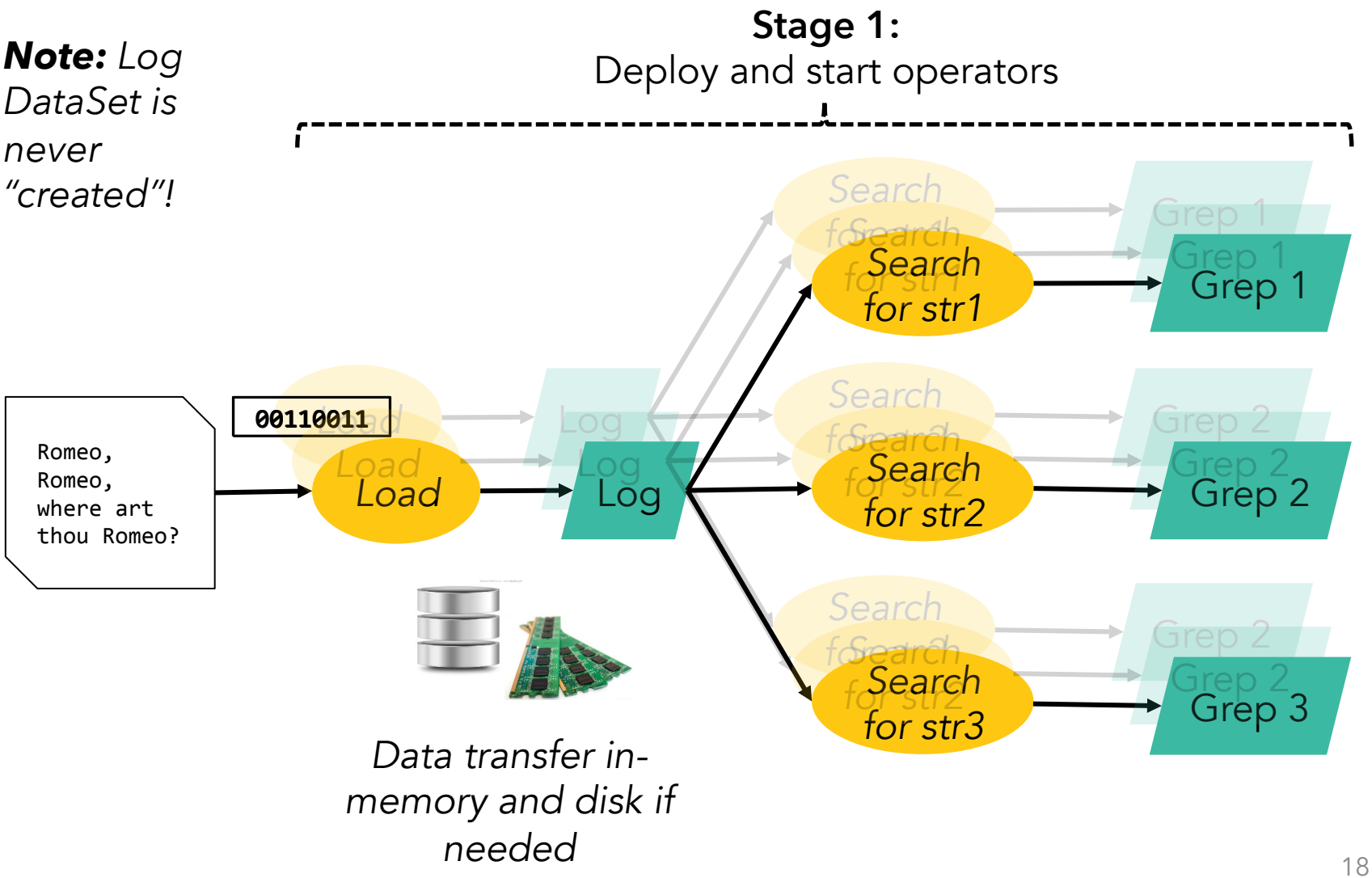


Staged (batch) execution

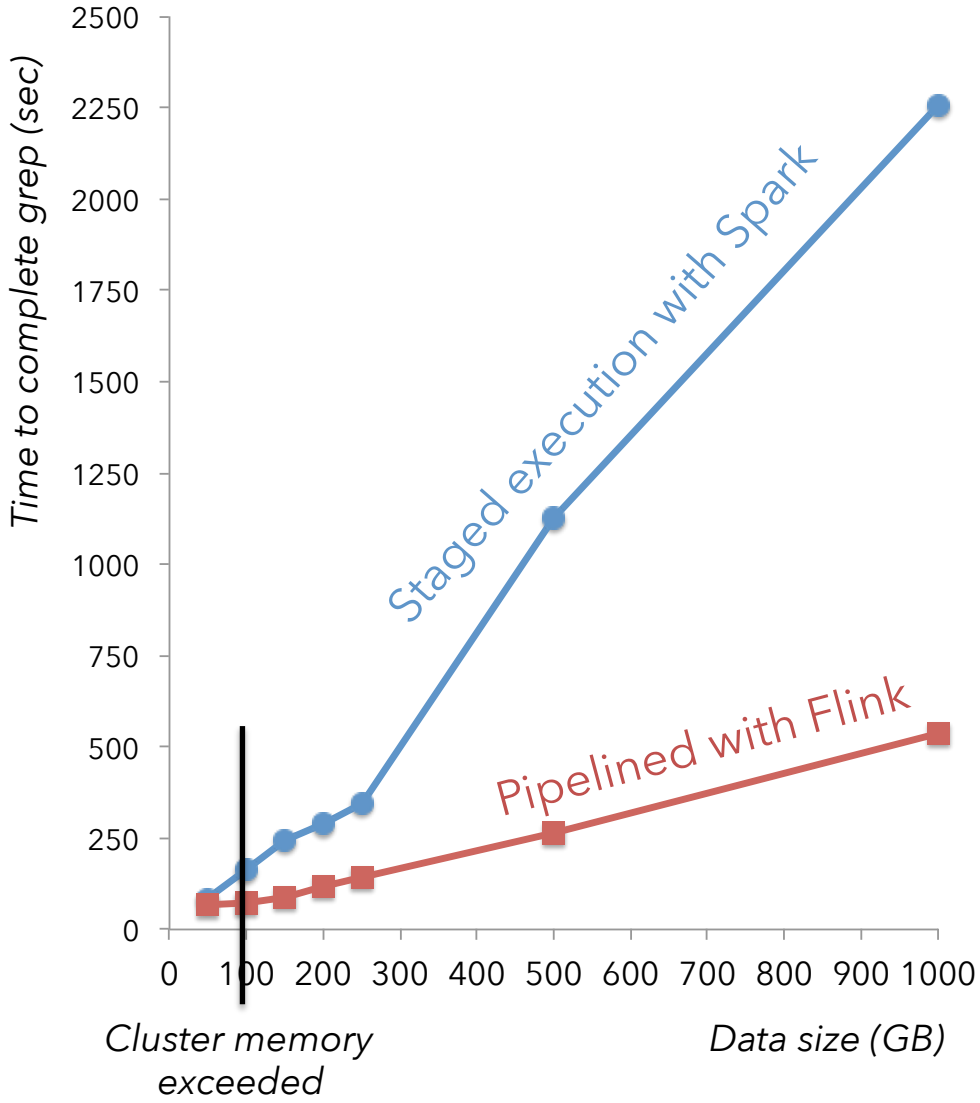


Pipelined execution

Note: *Log DataSet is never "created"!*



Benefits of pipelining



- 25 node cluster
- Grep log for 3 terms
- Scale data size from 100GB to 1TB

▶ Running Jobs

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

cancel

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

Drawbacks of pipelining

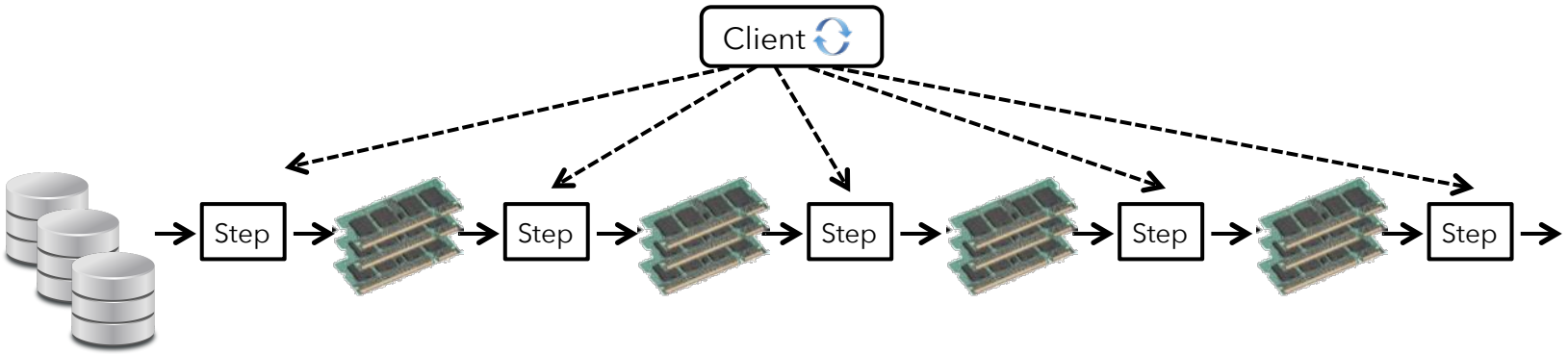
- Long pipelines may be active at the same time leading to memory fragmentation
  - FLINK-1101: Changes memory allocation from static to adaptive
- Fault-tolerance harder to get right
  - FLINK-986: Adds intermediate data sets (similar to RDDS) as first-class citizen to Flink Runtime. Will lead to fine-grained fault-tolerance among other features.

Example: Iterative processing

```
DataSet<Page> pages = ...
DataSet<Neighborhood> edges = ...
DataSet<Page> oldRanks = pages; DataSet<Page> newRanks;

for (i = 0; i < maxIterations; i++) {
  newRanks = update(oldRanks, edges)
  oldRanks = newRanks
}

DataSet<Page> result = newRanks;
```

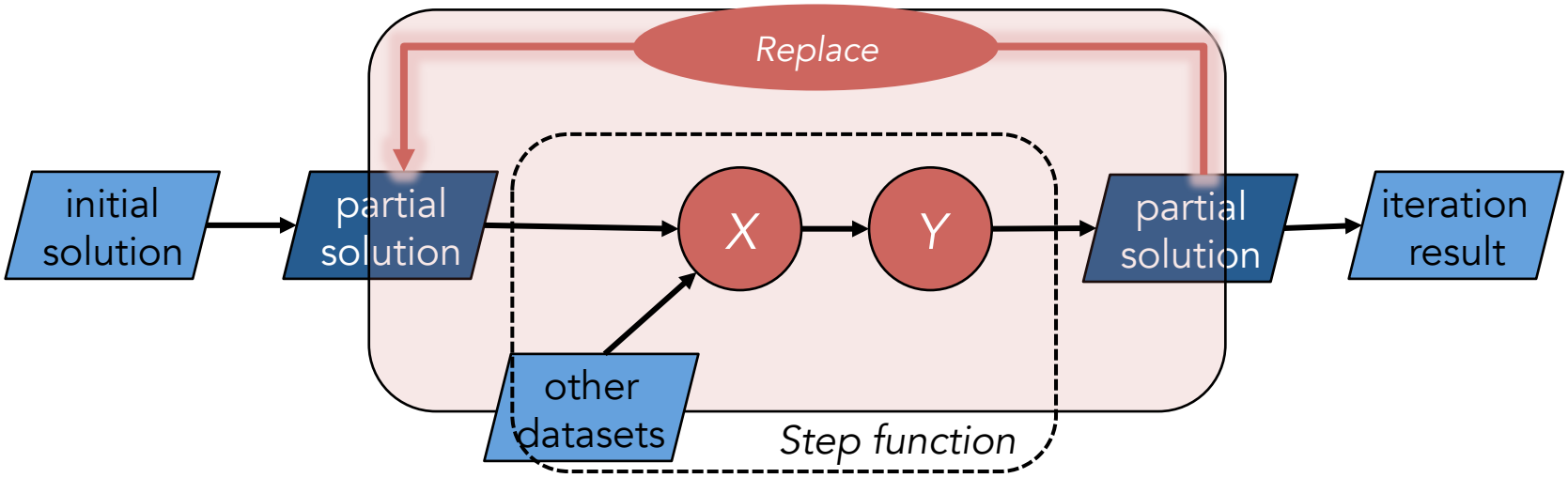
```
DataSet<Page> update (DataSet<Page> ranks, DataSet<Neighborhood> adjacency) {
  return oldRanks
    .join(adjacency)
    .where("id").equalTo("id")
    .with ( (page, adj, out) -> {
      for (long n : adj.neighbors)
        out.collect(new Page(n, df * page.rank / adj.neighbors.length))
    })
    .groupBy("id")
    .reduce ( (a, b) -> new Page(a.id, a.rank + b.rank) );
```

Iterate by unrolling



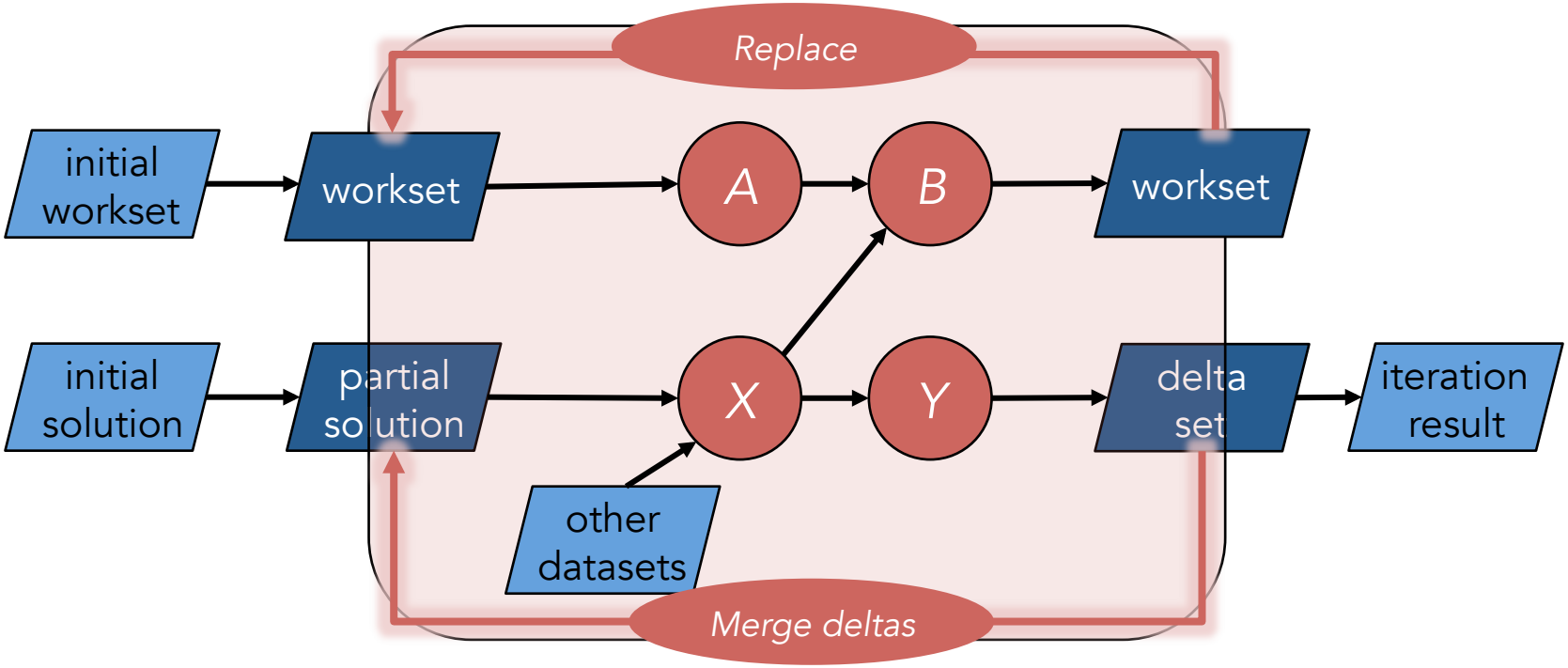
- for/while loop in client submits one job per iteration step
- Data reuse by caching in memory and/or disk

Iterate natively




```
DataSet<Page> pages = ...
DataSet<Neighborhood> edges = ...

IterativeDataSet<Page> pagesIter = pages.iterate(maxIterations);
DataSet<Page> newRanks = update (pagesIter, edges);
DataSet<Page> result = pagesIter.closeWith(newRanks)
```

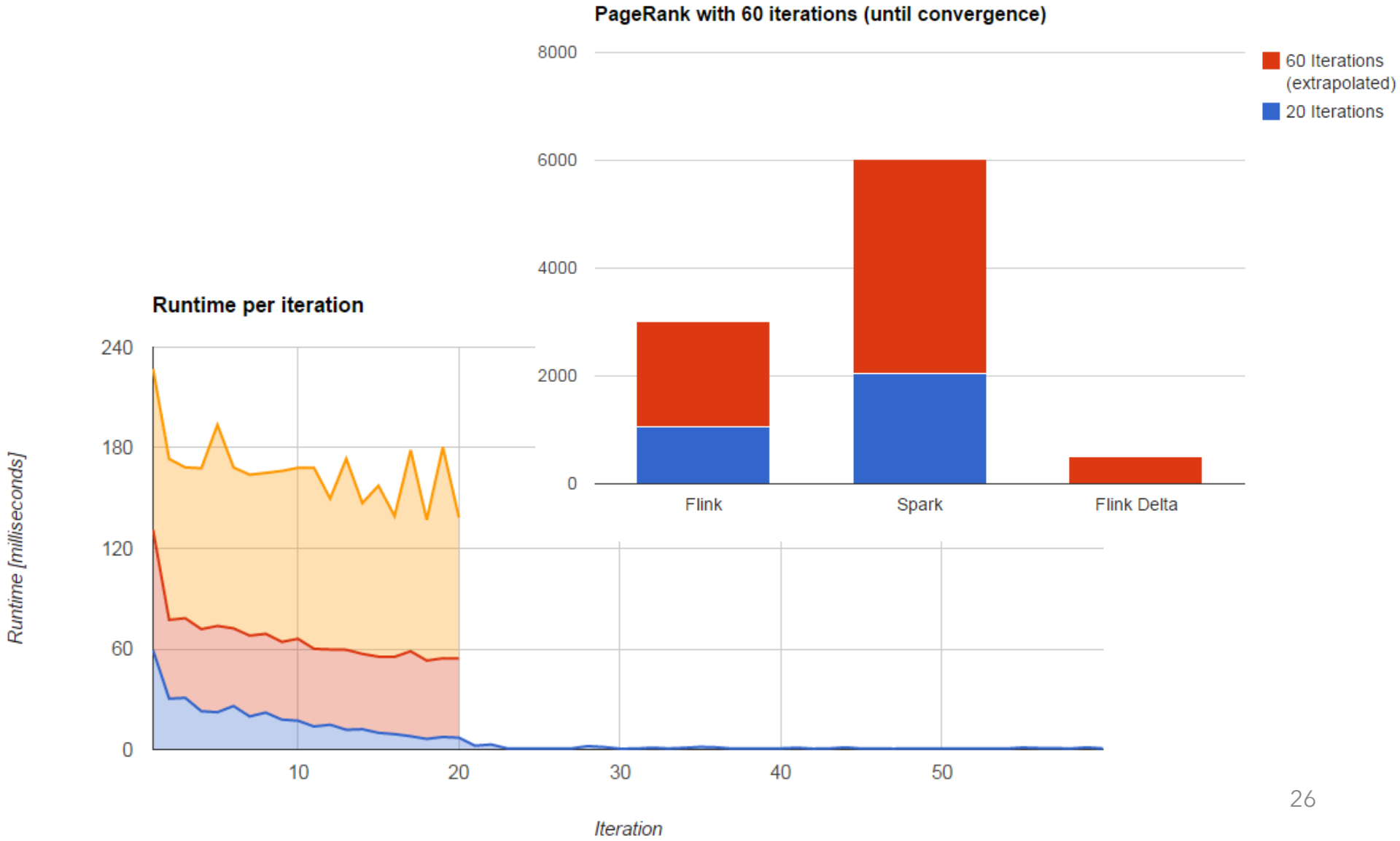
Iterate natively with deltas




```
DeltaIteration<...> pagesIter = pages.iterateDelta(initialDeltas, maxIterations, 0);
DataSet<...> newRanks = update (pagesIter, edges);
DataSet<...> newRanks = ...
DataSet<...> result = pagesIter.closeWith(newRanks, deltas)
```

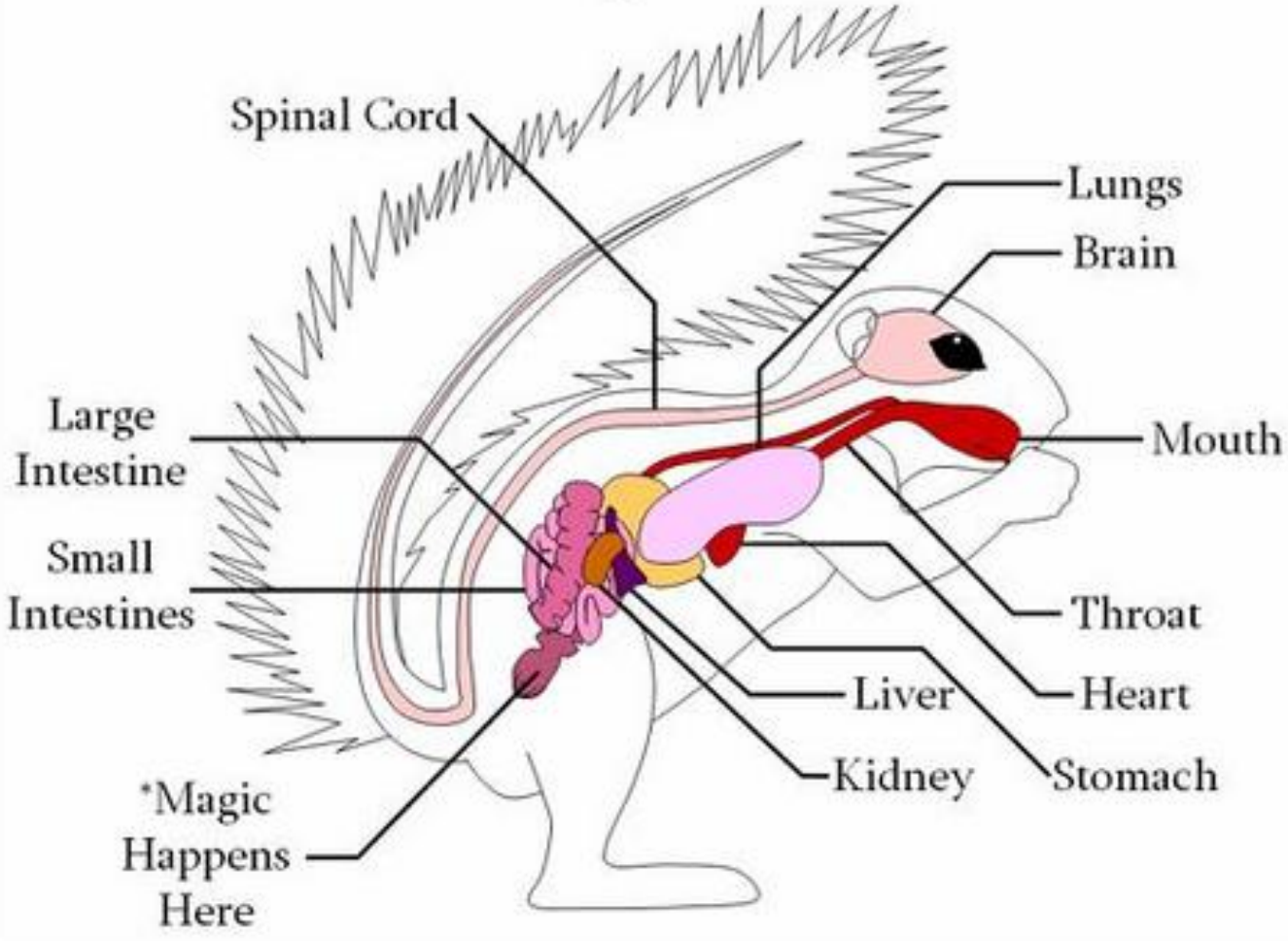
See http://data-artisans.com/data-analysis-with-flink.html

Native, unrolling, and delta

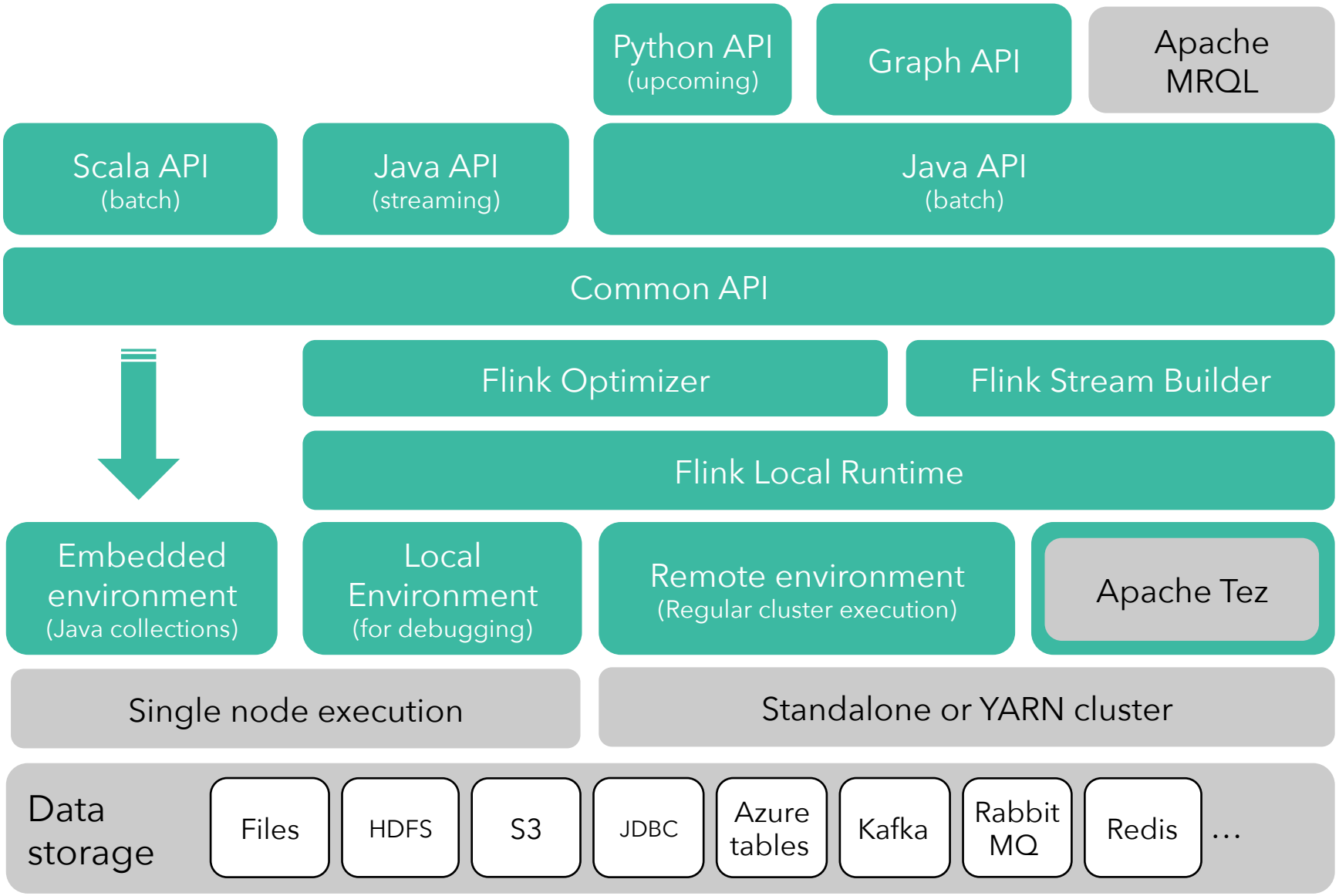


Dissecting Flink

Anatomy of a Squirrel



The growing Flink stack



Stack without Flink Streaming

Focus on regular (batch) processing…

Python API (upcoming)

Graph API

Apache MRQL

Scala API

Java API

Common API

Embedded environment (Java collections)

Flink Optimizer

Flink Local Runtime

Local Environment (for debugging)

Remote environment (Regular cluster execution)

Apache Tez

Single node execution

Standalone or YARN cluster

Data storage

Files

HDFS

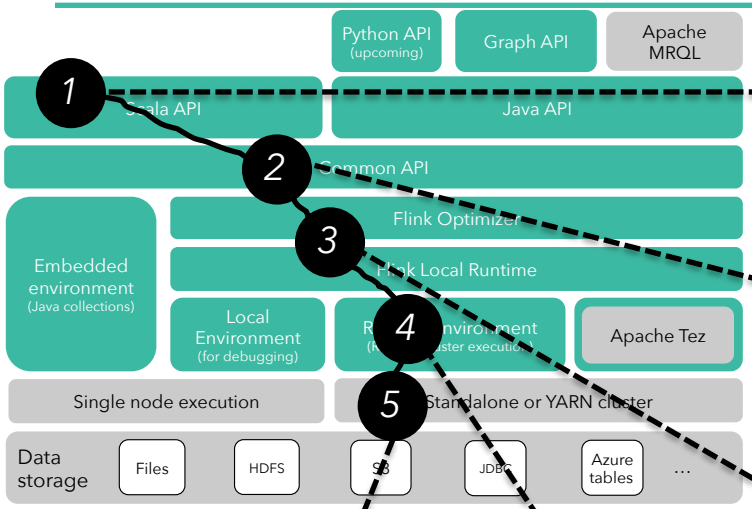
S3

JDBC

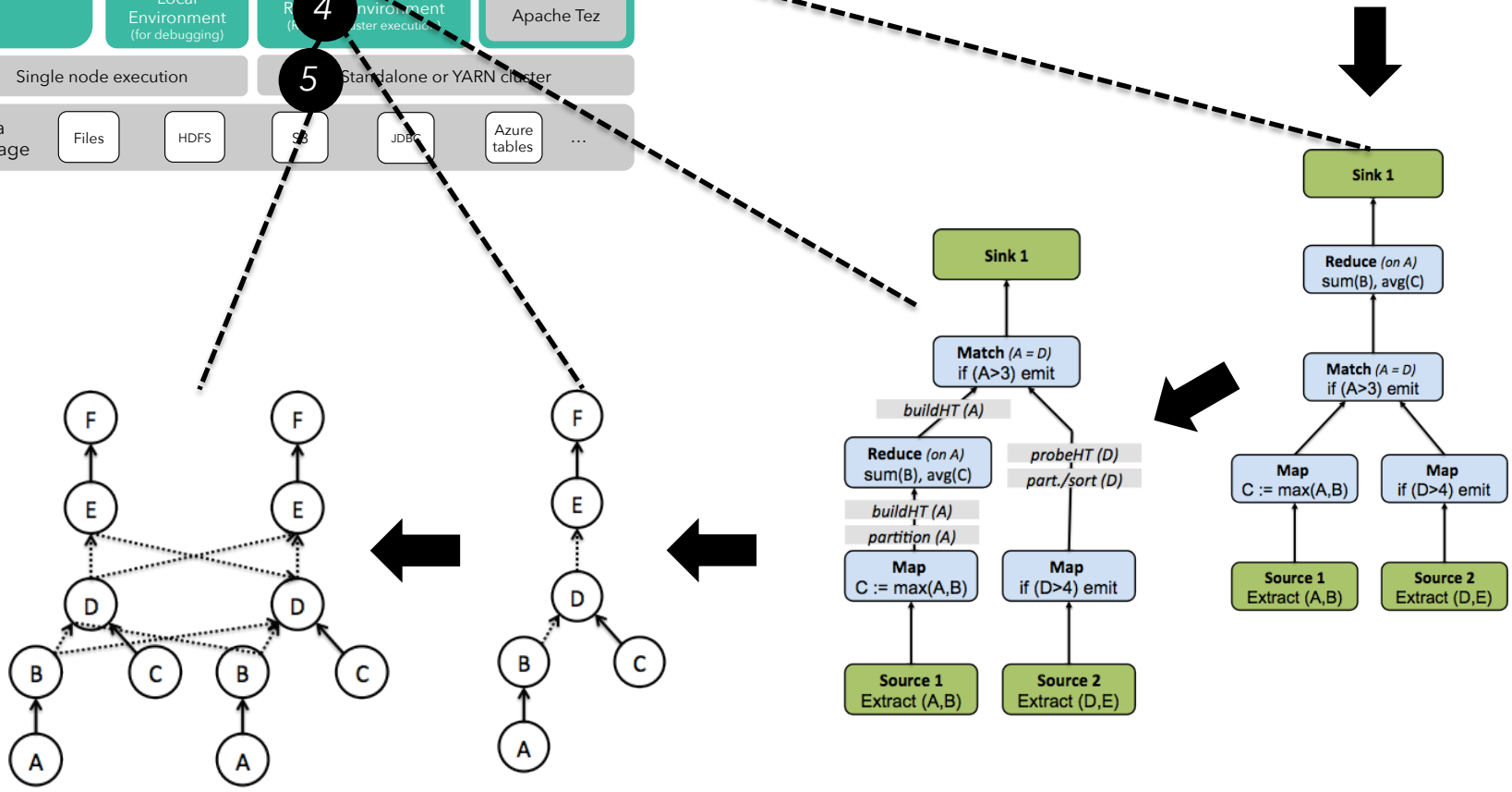
Azure tables

…

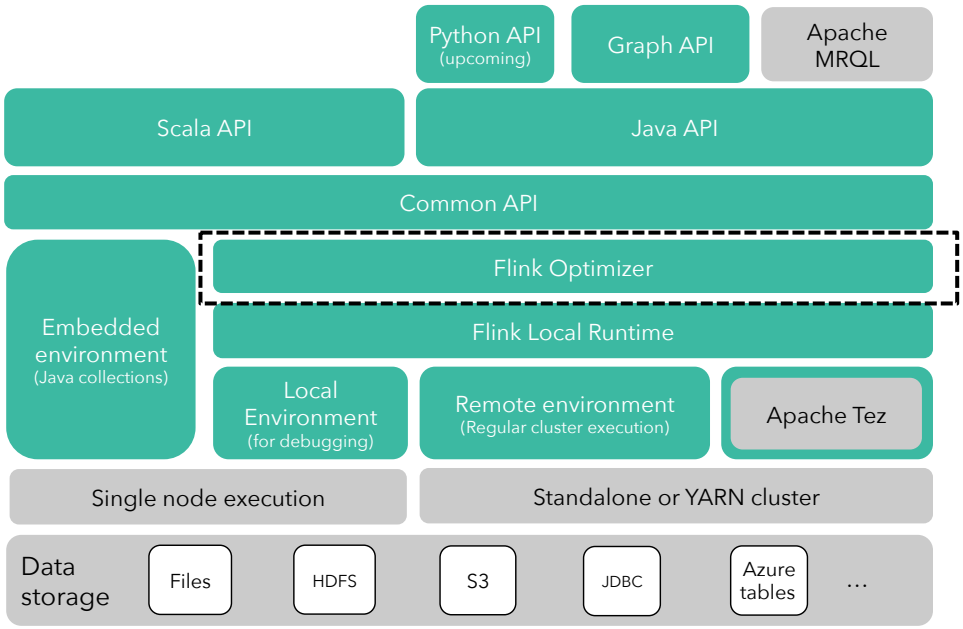
Program lifecycle




```
val source1 = …
val source2 = …
val maxed = source1
  .map(v => (v._1,v._2,
             math.max(v._1,v._2))
val filtered = source2
  .filter(v => (v._1 > 4))
val result = maxed
  .join(filtered).where(0).equalTo(0)
  .filter(_1 > 3)
  .groupBy(0)
  .reduceGroup {……}
```



Flink Optimizer

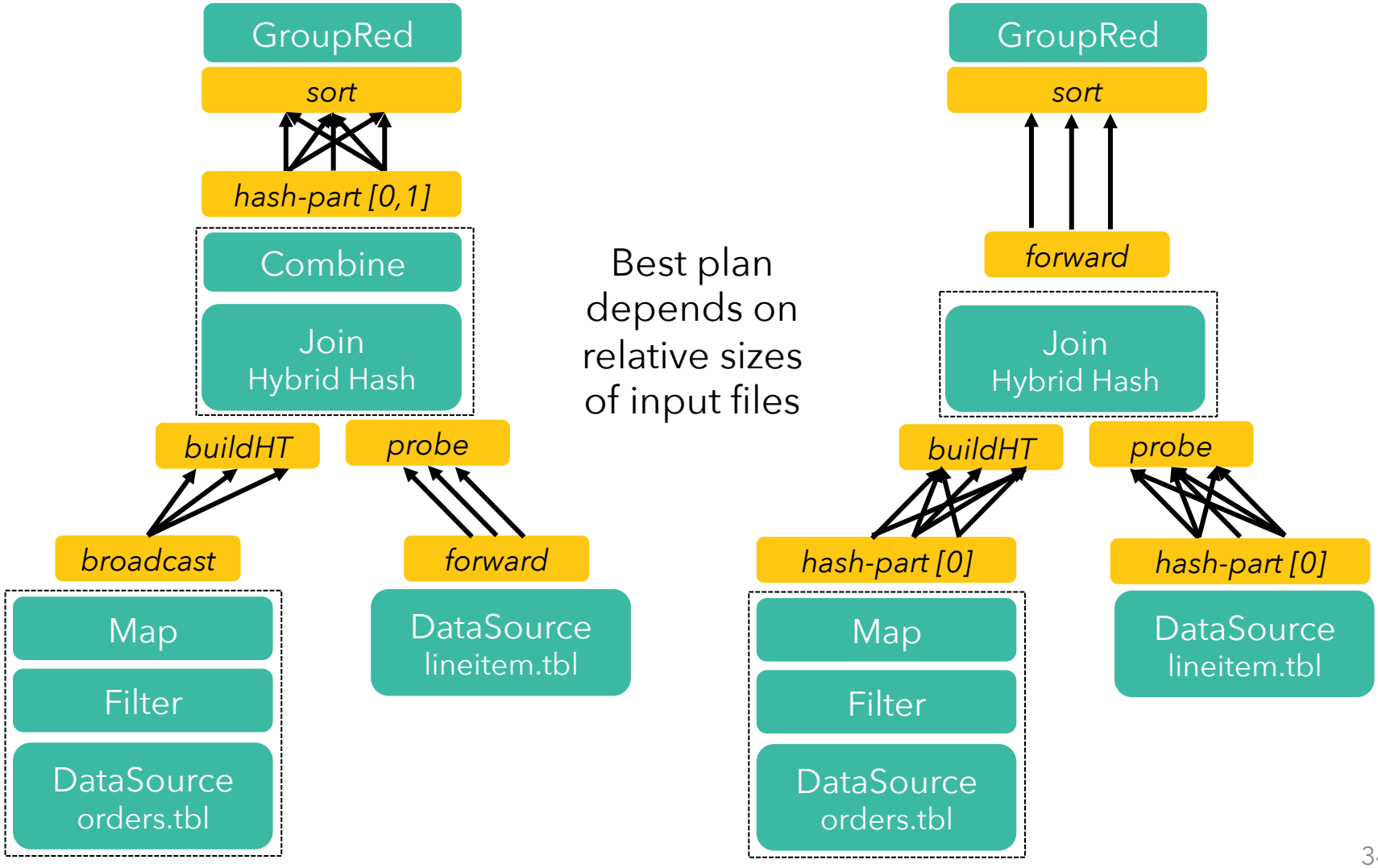


- The optimizer is the component that selects an execution plan for a Common API program
- Think of an AI system manipulating your program for you ☺
- But don't be scared – it works
  - Relational databases have been doing this for decades – Flink ports the technology to API-based systems

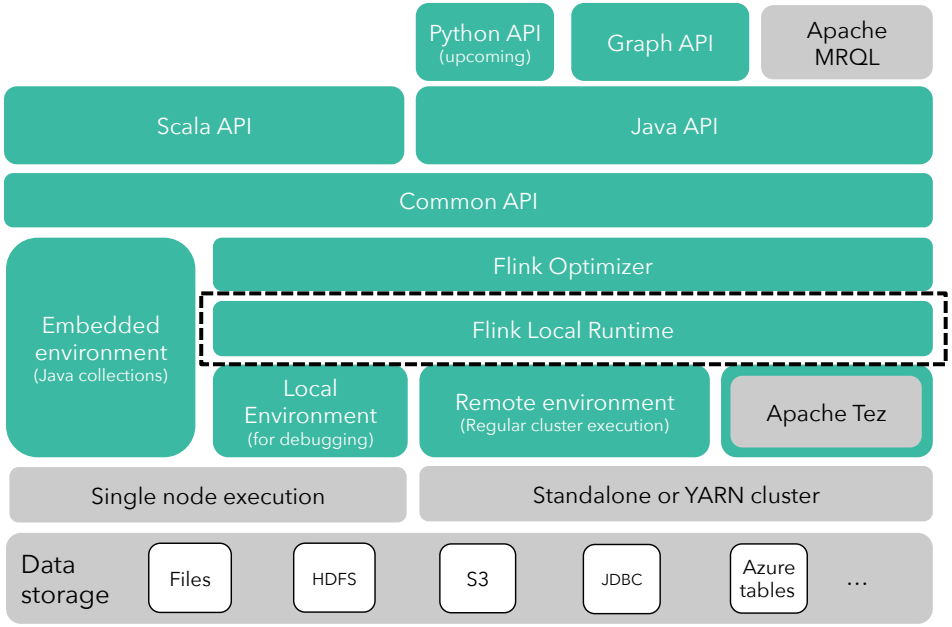
A simple program

```
DataSet<Tuple5<Integer, String, String, String, Integer>> orders = …
DataSet<Tuple2<Integer, Double>> lineitems = …

DataSet<Tuple2<Integer, Integer>> filteredOrders = orders
  .filter(. . .)
  .project(0,4).types(Integer.class, Integer.class);

DataSet<Tuple3<Integer, Integer, Double>> lineitemsOfOrders = filteredOrders
  .join(lineitems)
  .where(0).equalTo(0)
  .projectFirst(0,1).projectSecond(1)
  .types(Integer.class, Integer.class, Double.class);

DataSet<Tuple3<Integer, Integer, Double>> priceSums = lineitemsOfOrders
  .groupBy(0,1).aggregate(Aggregations.SUM, 2);

priceSums.writeAsCsv(outputPath);
```

Two execution plans



Flink Local Runtime



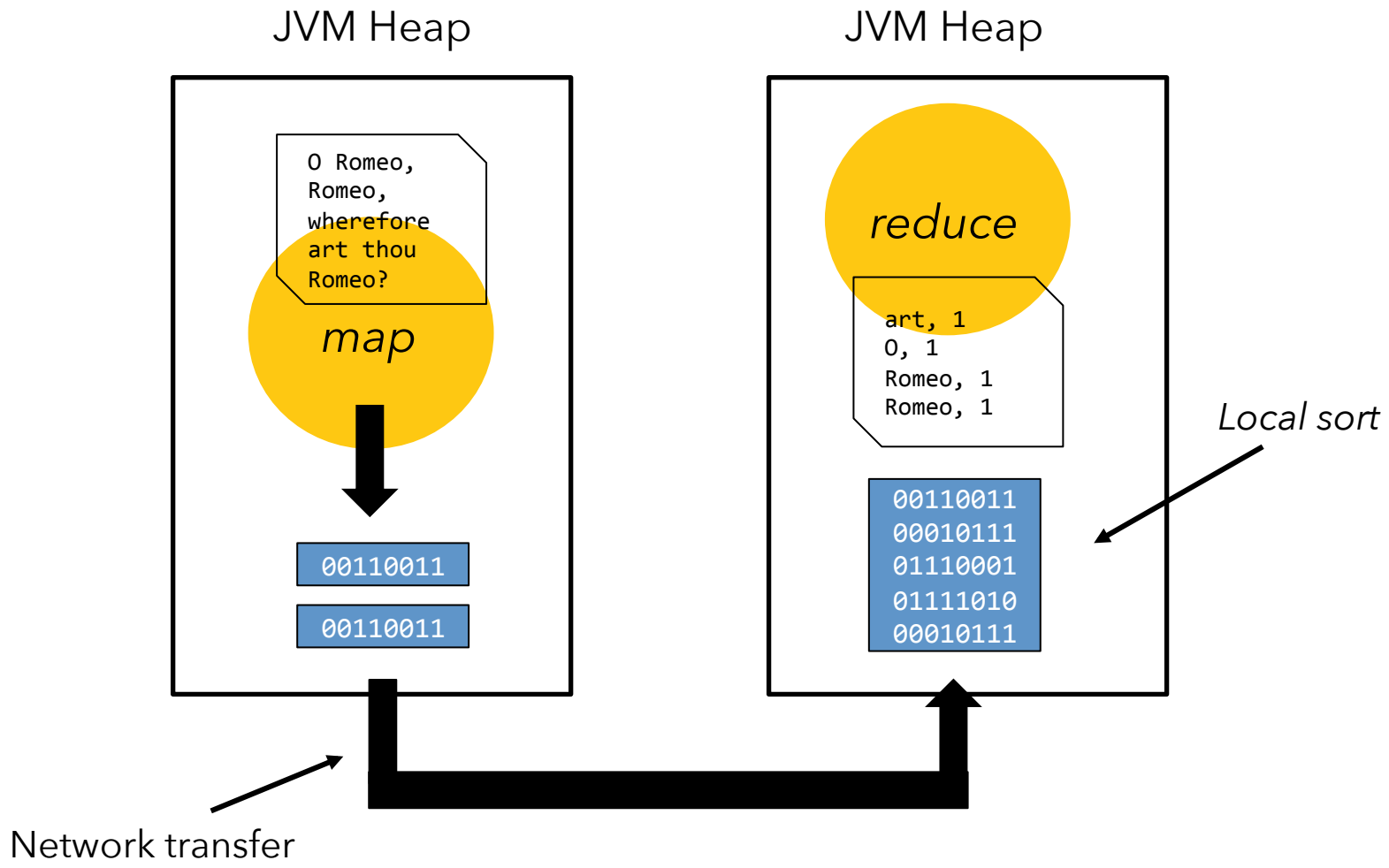
- *Local* runtime, not the distributed execution engine
- Aka: what happens inside every parallel task

Flink runtime operators

- Sorting and hashing data
  - Necessary for grouping, aggregation, reduce, join, cogroup, delta iterations
- Flink contains tailored implementations of hybrid hashing and external sorting in Java
  - Scale well with both abundant and restricted memory sizes

Internal data representation

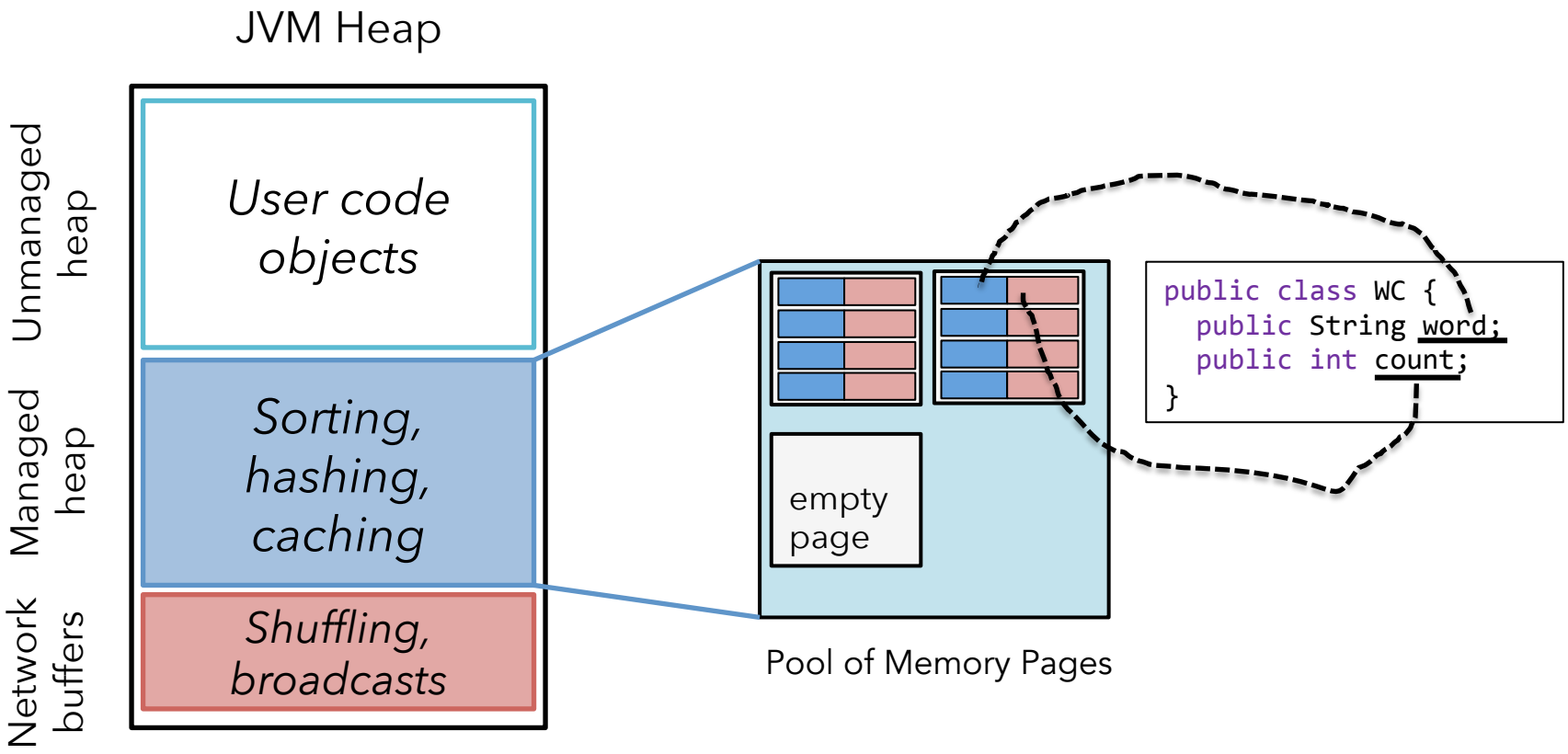
How is intermediate data internally represented?



Internal data representation

- Two options: Java objects or raw bytes
- **Java objects**
  - Easier to program
  - Can suffer from GC overhead
  - Hard to de-stage data to disk, may suffer from "out of memory exceptions"
- **Raw bytes**
  - Harder to program (customer serialization stack, more involved runtime operators)
  - Solves most of memory and GC problems
  - Overhead from object (de)serialization
- Flink follows the **raw byte** approach

Memory in Flink



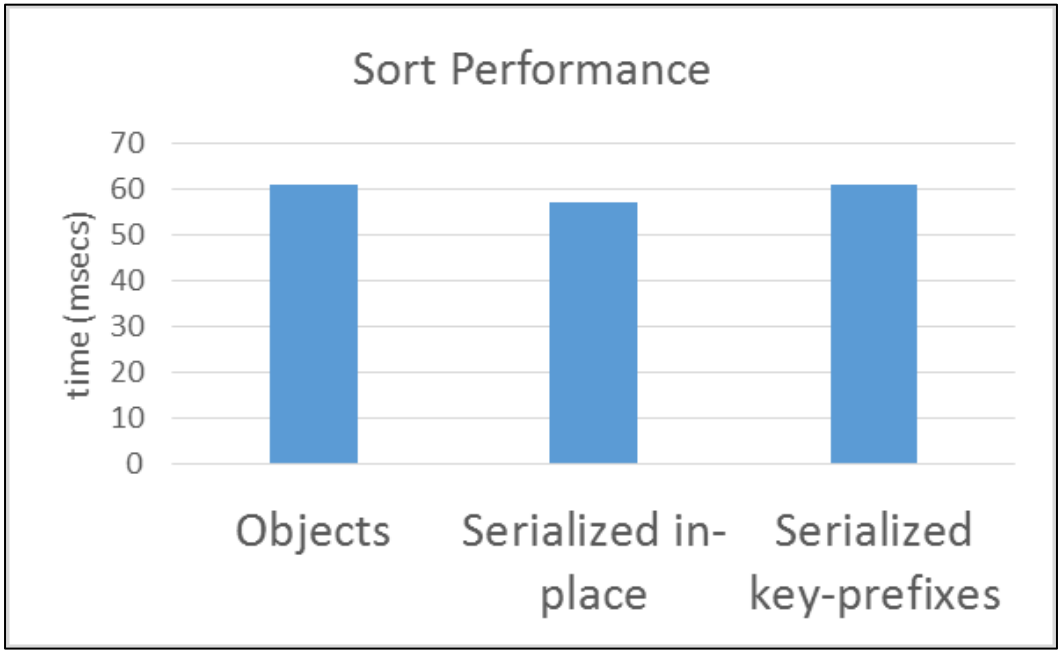
Memory in Flink (2)

- Internal memory management
  - Flink initially allocates 70% of the free heap as byte[] *segments*
  - Internal operators allocate() and release() these segments
- Flink has its own serialization stack
  - All accepted data types serialized to data segments
- Easy to reason about memory, (almost) no OutOfMemory errors, reduces the pressure to the GC (smooth performance)

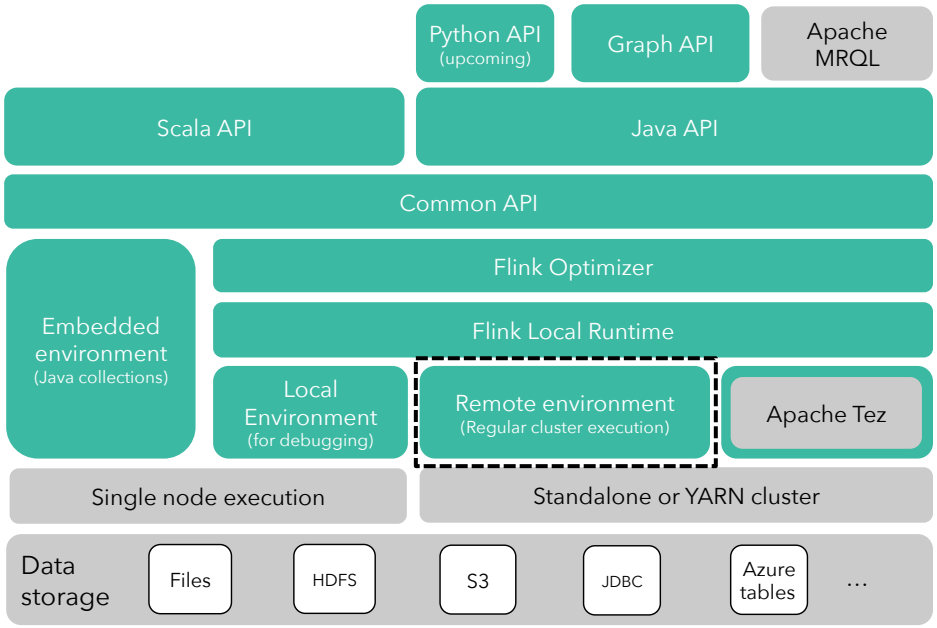
Operating on serialized data

Microbenchmark

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



Flink distributed execution



- Pipelined
  - Same engine for Flink and Flink streaming
- Pluggable
  - Local runtime can be executed on other engines
  - E.g., Java collections and Apache Tez

Closing

Summary

- Flink decouples API from execution
  - Same program can be executed in many different ways
  - Hopefully users do not need to care about this and still get very good performance
- Unique Flink internal features
  - Pipelined execution, native iterations, optimizer, serialized data manipulation, good disk destaging
- Very good performance
  - Known issues currently worked on actively

Stay informed

- flink.incubator.apache.org
  - Subscribe to the mailing lists!
  - http://flink.incubator.apache.org/community.html#mailing-lists
- Blogs
  - flink.incubator.apache.org/blog
  - data-artisans.com/blog
- Twitter
  - follow @ApacheFlink


dataArtisans

That’s it, time for beer