Apache Flink

Fast and Reliable Large-Scale Data Processing

Fabian Hueske @fhueske

What is Apache Flink?

Distributed Data Flow Processing System

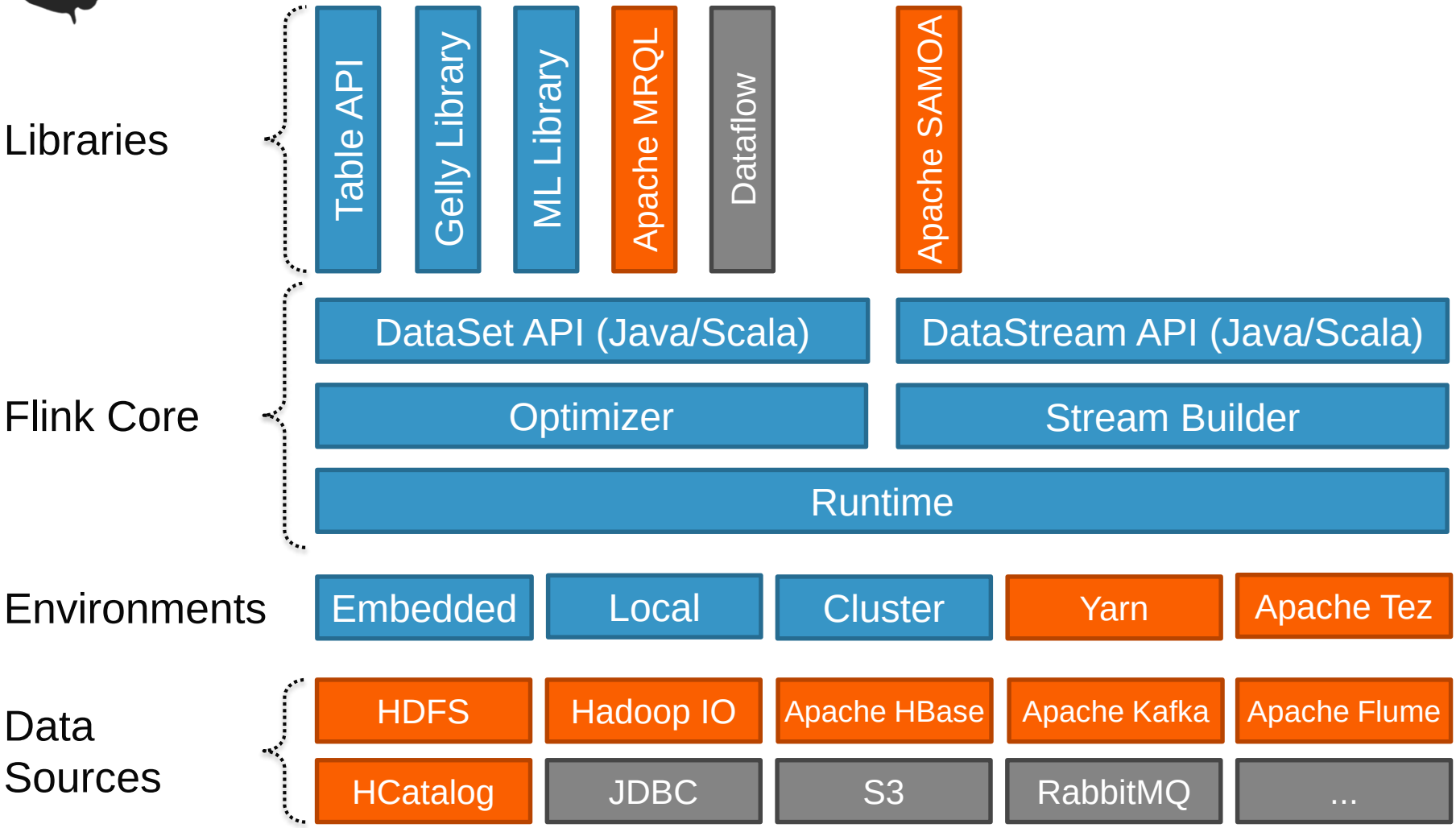
- Focused on large-scale data analytics
- Real-time stream and batch processing
- Easy and powerful APIs (Java / Scala)
- Robust execution backend

What is Flink good at?

It‘s a general-purpose data analytics system

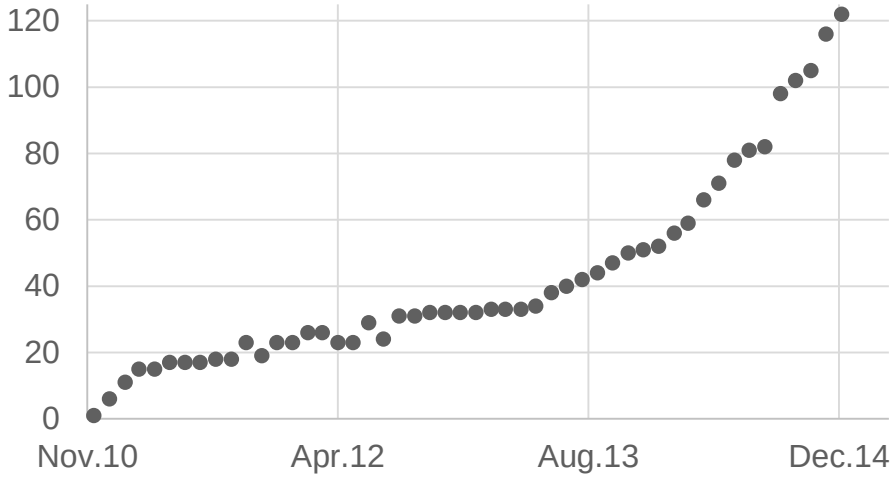
- Real-time stream processing with flexible windows
- Complex and heavy ETL jobs
- Analyzing huge graphs
- Machine-learning on large data sets
- ...

Flink in the Hadoop Ecosystem



Flink in the ASF

- Flink entered the ASF about one year ago
  - 04/2014: Incubation
  - 12/2014: Graduation
- Strongly growing community



#unique git committers (w/o manual de-dup)

Where is Flink moving?

A "use-case complete" framework to unify batch & stream processing

Data Streams
- *Kafka*
- *RabbitMQ*
- *...*

“Historic” data
- *HDFS*
- *JDBC*
- *...*

Analytical Workloads
- *ETL*
- *Relational processing*
- *Graph analysis*
- *Machine learning*
- *Streaming data analysis*

Goal: Treat batch as finite stream

Programming Model & APIs

HOW TO USE FLINK?

Unified Java & Scala APIs

- Fluent and mirrored APIs in Java and Scala
- Table API for relational expressions
- Batch and Streaming APIs almost identical ...
  - ... with slightly different semantics in some cases

DataSets 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
    .map(str -> str.toLowerCase());
second.print();

env.execute();
```

Expressive Transformations

- Element-wise
  - `map, flatMap, filter, project`
- Group-wise
  - `groupBy, reduce, reduceGroup, combineGroup, mapPartition, aggregate, distinct`
- Binary
  - `join, coGroup, union, cross`
- Iterations
  - `iterate, iterateDelta`
- Physical re-organization
  - `rebalance, partitionByHash, sortPartition`
- Streaming
  - `Window, windowMap, coMap, ...`

Rich Type System

- Use any Java/Scala classes as a data type
  - Tuples, POJOs, and case classes
  - Not restricted to key-value pairs
- Define (composite) keys directly on data types
  - Expression
  - Tuple position
  - Selector function

Counting Words in Batch and Stream

```
case class Word (word: String, frequency: Int)
```

DataSet API (batch):

```
val lines: DataSet[String] = env.readTextFile(...)

lines.flatMap {line => line.split(" ")
                       .map(word => Word(word,1))}
     .groupBy("word").sum("frequency")
     .print()
```

DataStream API (streaming):

```
val lines: DataStream[String] = env.fromSocketStream(...)

lines.flatMap {line => line.split(" ")
                       .map(word => Word(word,1))}
     .window(Count.of(1000)).every(Count.of(100))
     .groupBy("word").sum("frequency")
     .print()
```

Table API

- Execute SQL-like expressions on table data
  - Tight integration with Java and Scala APIs
  - Available for batch and streaming programs

```
val orders = env.readCsvFile(…)
  .as('oId, 'oDate, 'shipPrio)
  .filter('shipPrio === 5)

val items = orders
  .join(lineitems).where('oId === 'id)
  .select('oId, 'oDate, 'shipPrio,
          'extdPrice * (Literal(1.0f) - 'discnt) as 'revenue)

val result = items
  .groupBy('oId, 'oDate, 'shipPrio)
  .select('oId, 'revenue.sum, 'oDate, 'shipPrio)
```

Libraries are emerging

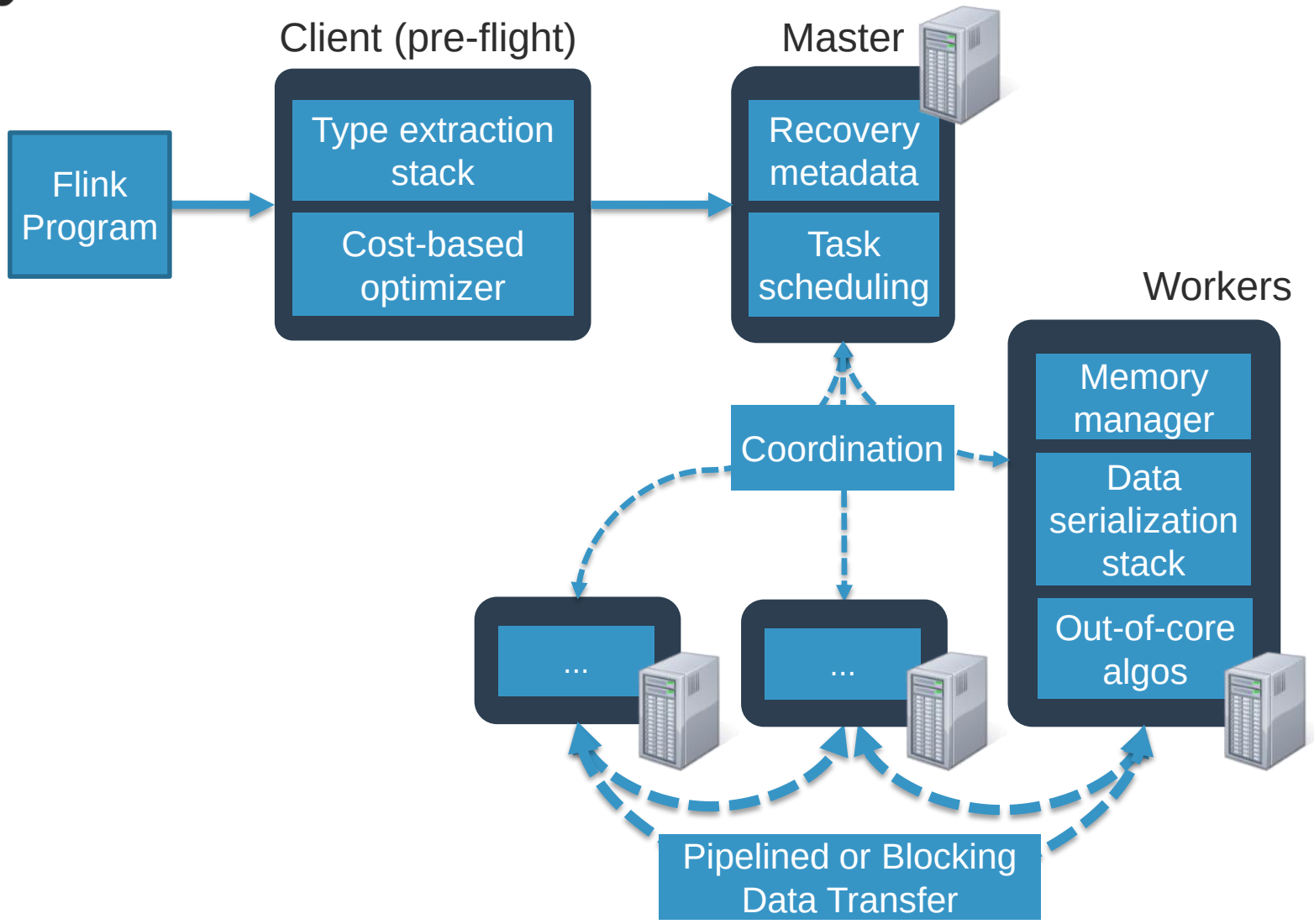
- As part of the Apache Flink project
  - Gelly: Graph processing and analysis
  - Flink ML: Machine-learning pipelines and algorithms
  - Libraries are built on APIs and can be mixed with them

- Outside of Apache Flink
  - Apache SAMOA (incubating)
  - Apache MRQL (incubating)
  - Google DataFlow translator

Processing Engine

WHAT IS HAPPENING INSIDE?

System Architecture



Cool technology inside Flink

- Batch and Streaming in one system
- Memory-safe execution
- Built-in data flow iterations
- Cost-based data flow optimizer
- Flexible windows on data streams
- Type extraction and serialization utilities
- Static code analysis on user functions
- and much more...

Pipelined Data Transfer

STREAM AND BATCH IN ONE SYSTEM

Stream and Batch in one System

- Most systems are either stream or batch systems
- In the past, Flink focused on batch processing
  - Flink‘s runtime has always done stream processing
  - Operators pipeline data forward as soon as it is processed
  - Some operators are blocking (such as sort)
- Stream API and operators are recent contributions
  - Evolving very quickly under heavy development

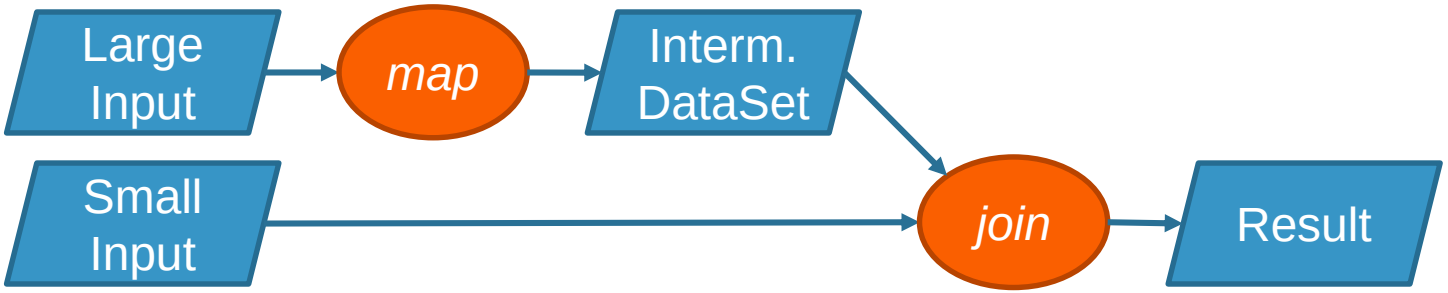
Pipelined Data Transfer

- Pipelined data transfer has many benefits
  - True stream and batch processing in one stack
  - Avoids materialization of large intermediate results
  - Better performance for many batch workloads

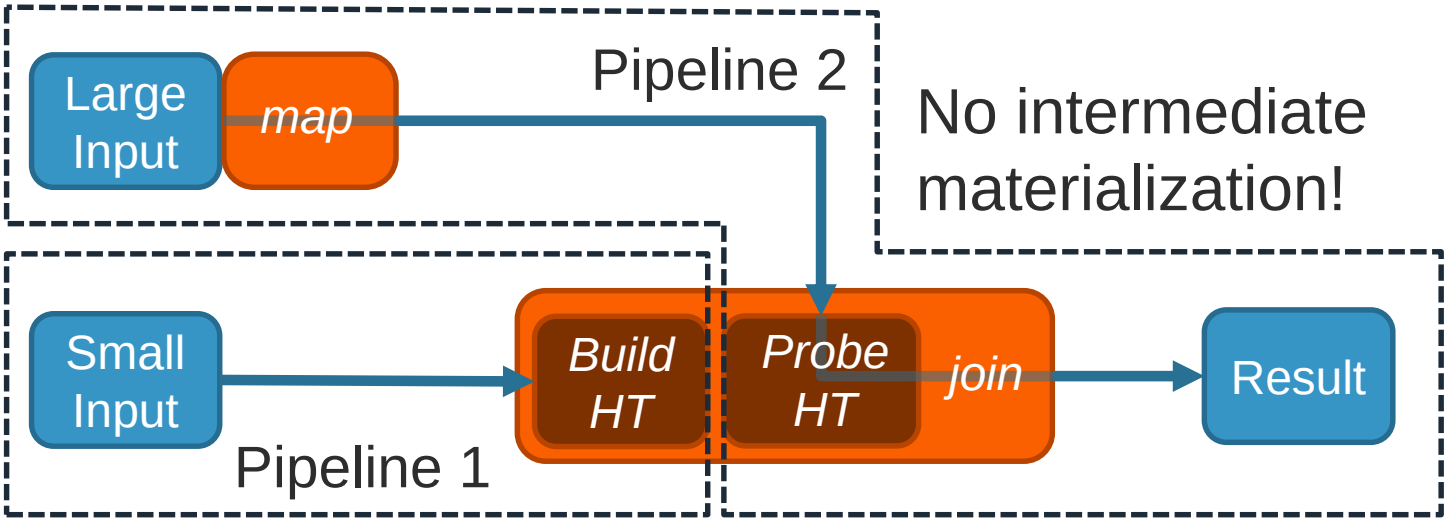
- Flink supports blocking data transfer as well

Pipelined Data Transfer

Program



Pipelined Execution

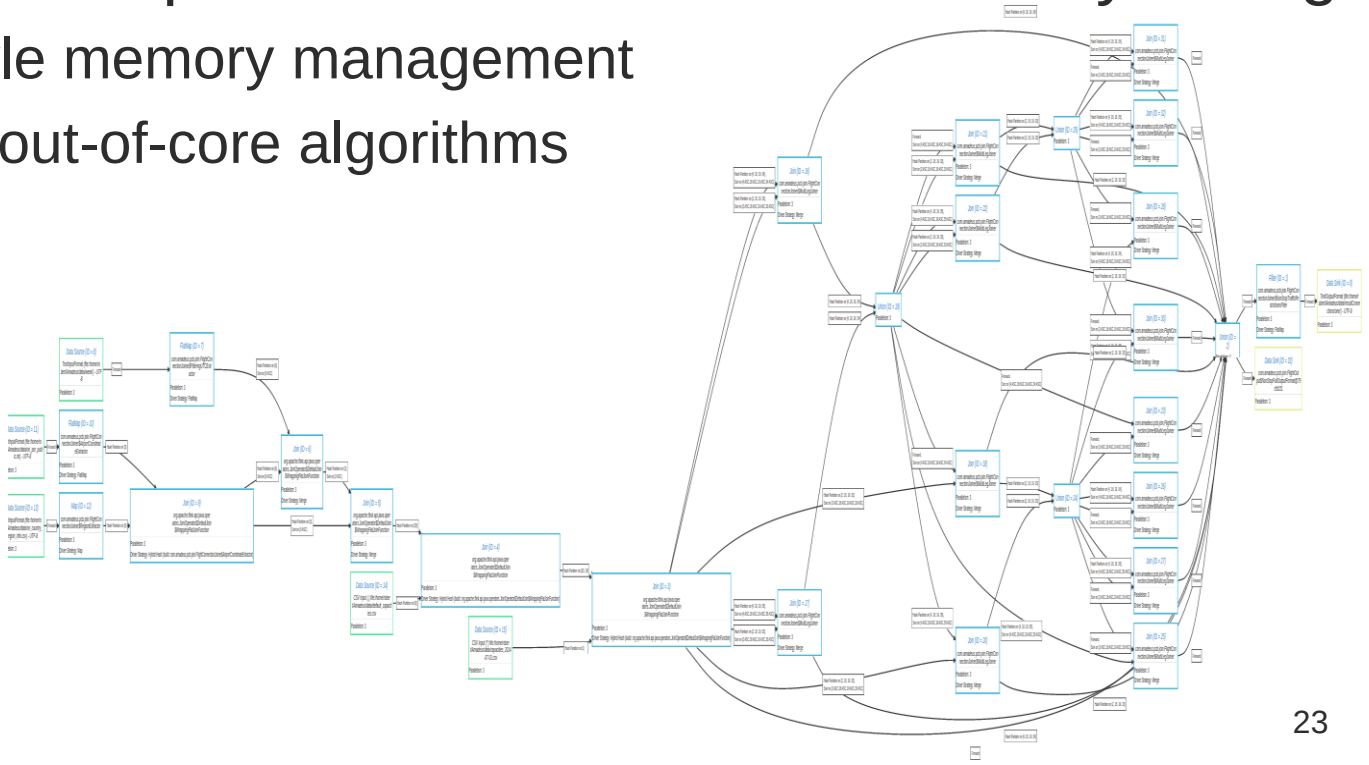


Memory Management and Out-of-Core Algorithms

MEMORY SAFE EXECUTION

Memory-safe Execution

- Challenge of JVM-based data processing systems
  - OutOfMemoryErrors due to data objects on the heap
- Flink runs complex data flows without memory tuning
  - C++-style memory management
  - Robust out-of-core algorithms

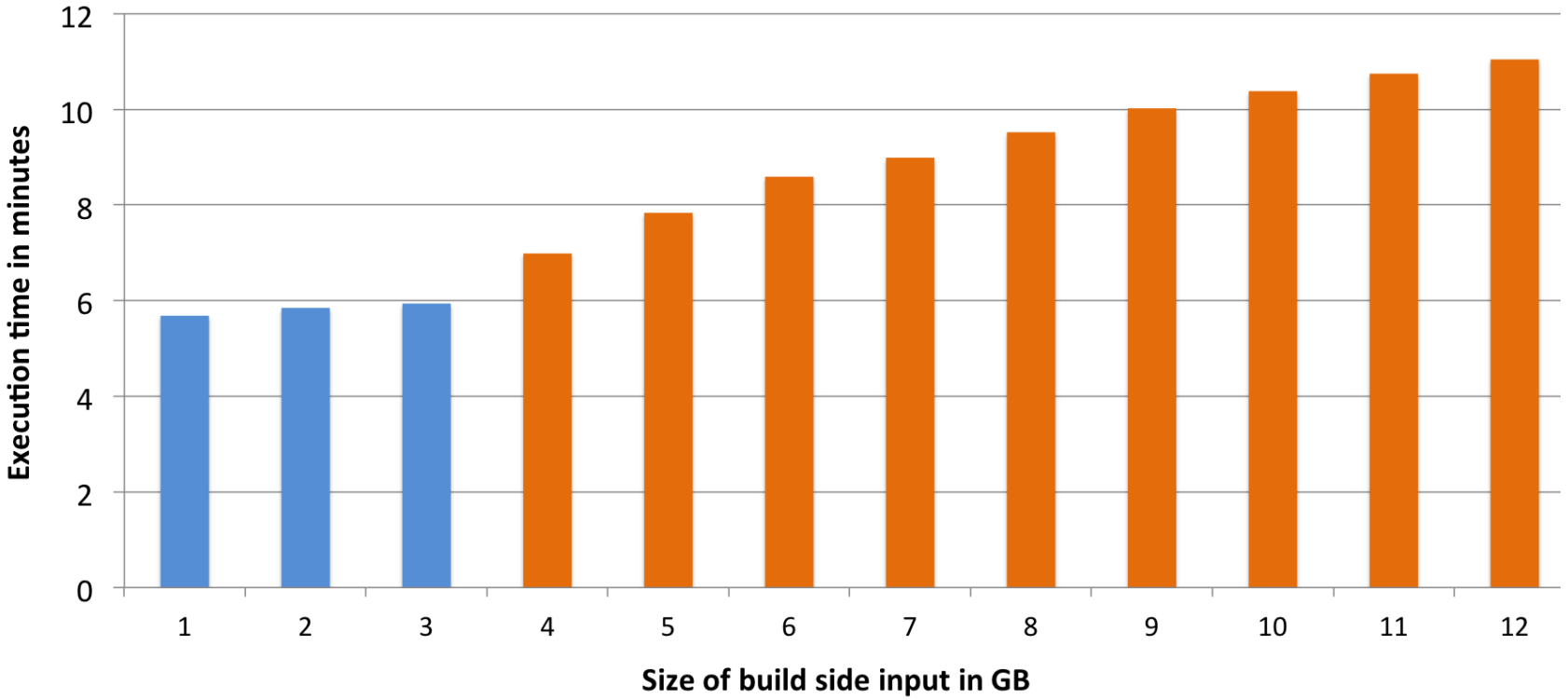


Managed Memory

- Active memory management
  - Workers allocate 70% of JVM memory as byte arrays
  - Algorithms serialize data objects into byte arrays
  - In-memory processing as long as data is small enough
  - Otherwise partial destaging to disk

- Benefits
  - Safe memory bounds (no OutOfMemoryError)
  - Scales to very large JVMs
  - Reduced GC pressure

Going out-of-core



Single-core join of 1KB Java objects beyond memory (4 GB)

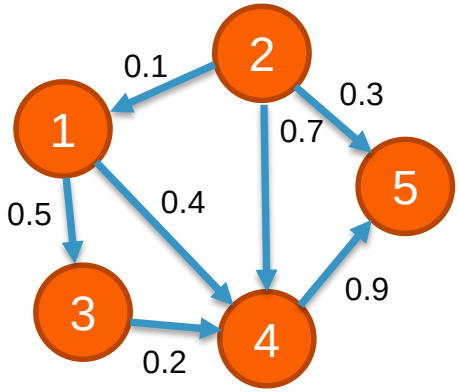
Blue bars are in-memory, orange bars (partially) out-of-core

Native Data Flow Iterations

GRAPH ANALYSIS

Native Data Flow Iterations

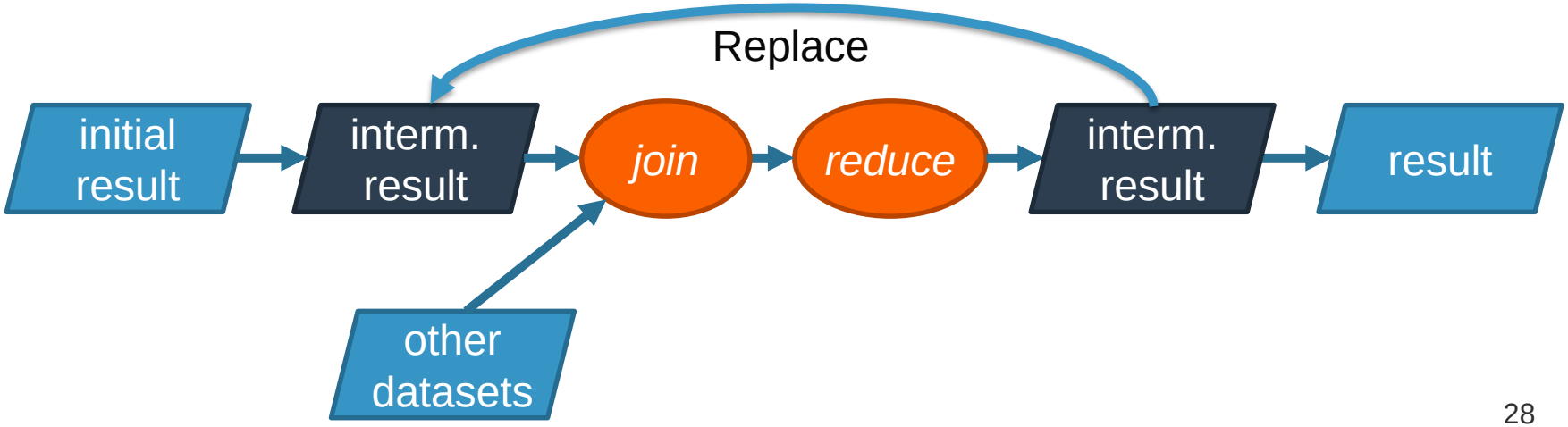
- Many graph and ML algorithms require iterations
- Flink features native data flow iterations
  - Loops are not unrolled
  - But executed as cyclic data flows
- Two types of iterations
  - Bulk iterations
  - Delta iterations



- Performance competitive with specialized systems

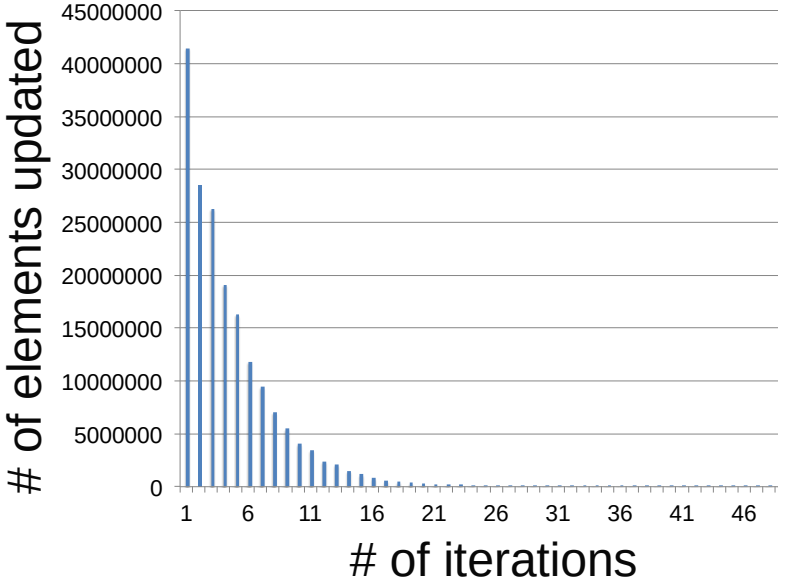
Iterative Data Flows

- Flink runs iterations „natively“ as cyclic data flows
  - Operators are scheduled once
  - Data is fed back through backflow channel
  - Loop-invariant data is cached
- Operator state is preserved across iterations!



Delta Iterations

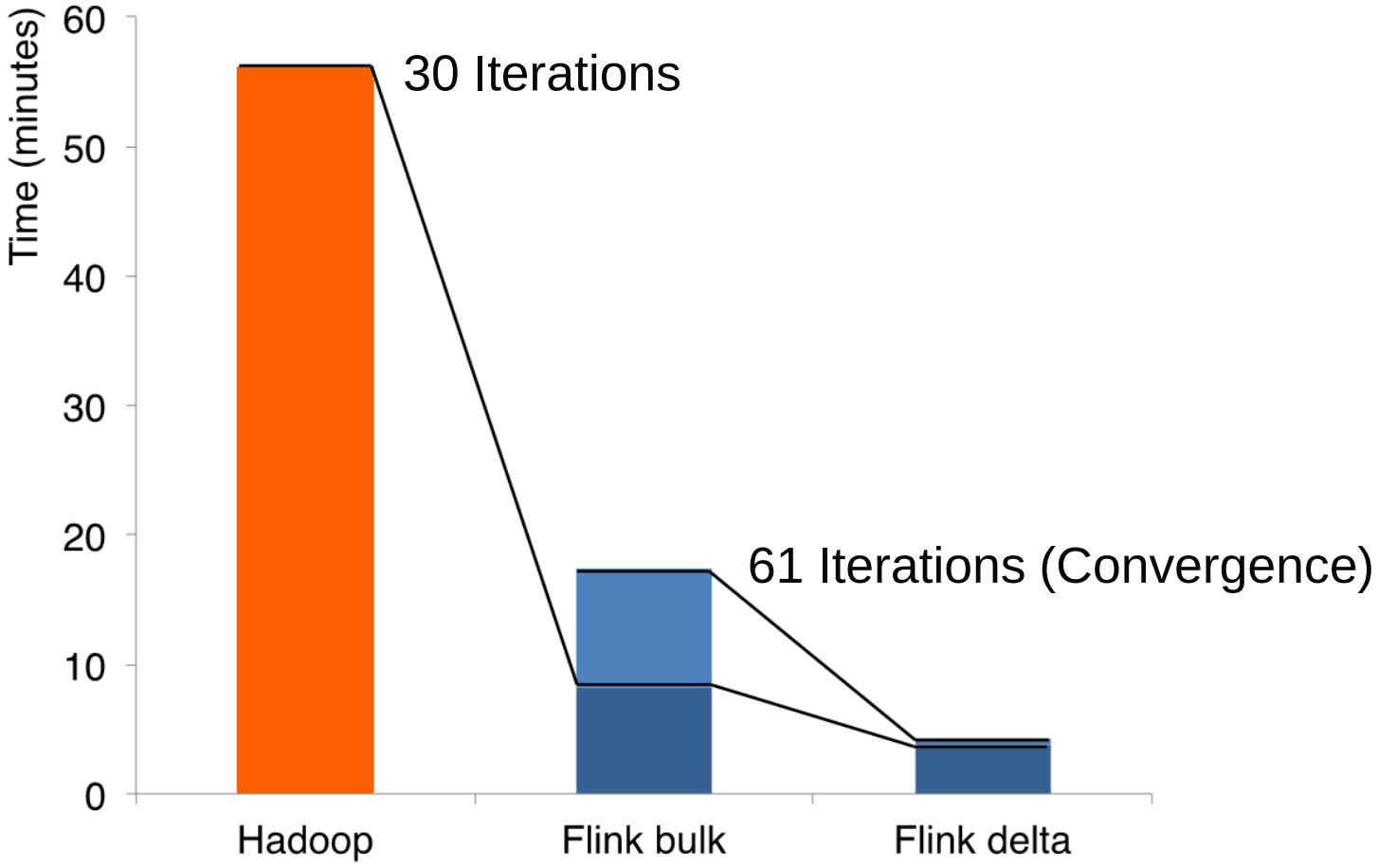
- Delta iteration computes
  - Delta update of solution set
  - Work set for next iteration



- Work set drives computations of next iteration
  - Workload of later iterations significantly reduced
  - Fast convergence

- Applicable to certain problem domains
  - Graph processing

Iteration Performance



PageRank on Twitter Follower Graph

Roadmap

WHAT IS COMING NEXT?

Flink’s Roadmap

Mission: Unified stream and batch processing

- Exactly-once streaming semantics with flexible state checkpointing
- Extending the ML library
- Extending graph library
- Interactive programs
- Integration with Apache Zeppelin (incubating)
- SQL on top of expression language
- And much more…

tl;dr – What's worth to remember?

- Flink is general-purpose analytics system
- Unifies streaming and batch processing
- Expressive high-level APIs
- Robust and fast execution engine

I Flink, do you? ;-)

If you find this exciting,

get involved and start a discussion on Flink‘s ML

or stay tuned by

subscribing to ***news@flink.apache.org*** or
following @***ApacheFlink*** on Twitter

BACKUP

Data Flow Optimizer

- Database-style optimizations for parallel data flows
- Optimizes all batch programs
- Optimizations
  - Task chaining
  - Join algorithms
  - Re-use partitioning and sorting for later operations
  - Caching for iterations

Data Flow Optimizer

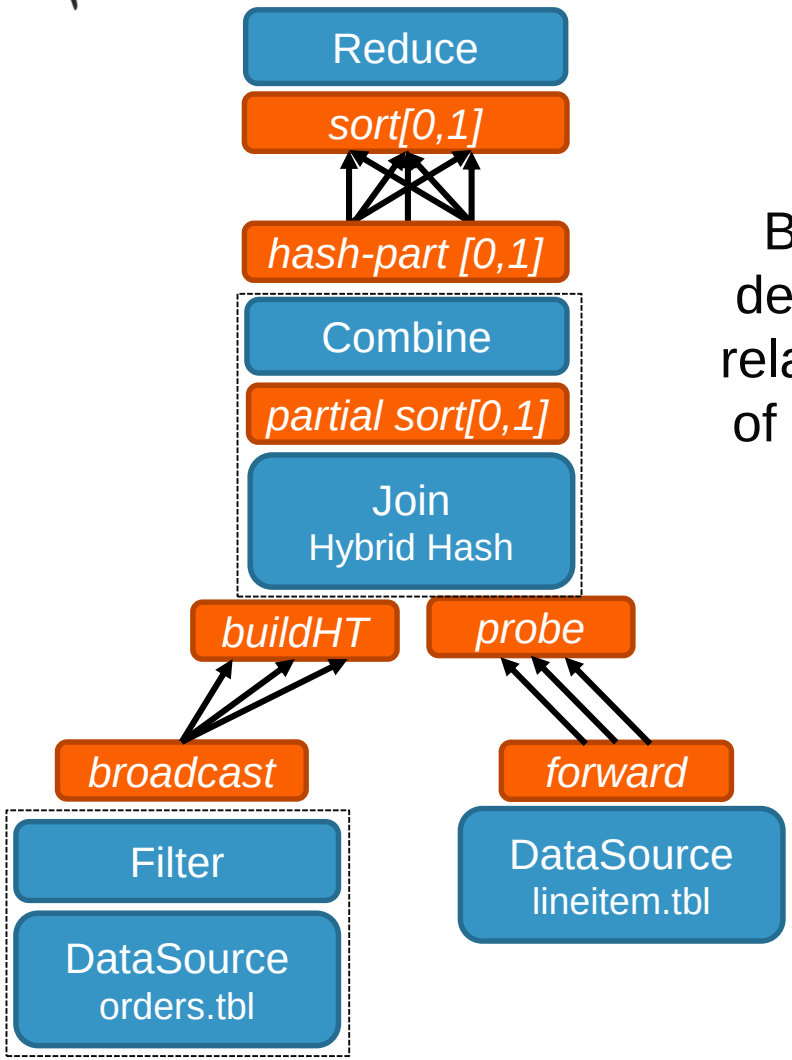
```
val orders = …
val lineitems = …

val filteredOrders = orders
  .filter(o => dataFormat.parse(l.shipDate).after(date))
  .filter(o => o.shipPrio > 2)

val lineitemsOfOrders = filteredOrders
  .join(lineitems)
  .where("orderId").equalTo("orderId")
  .apply((o,l) => new SelectedItem(o.orderDate, l.extdPrice))

val priceSums = lineitemsOfOrders
  .groupBy("orderDate")
  .sum("l.extdPrice");
```

Data Flow Optimizer



Best plan depends on relative sizes of input files