Architecture of Flink's Streaming Runtime

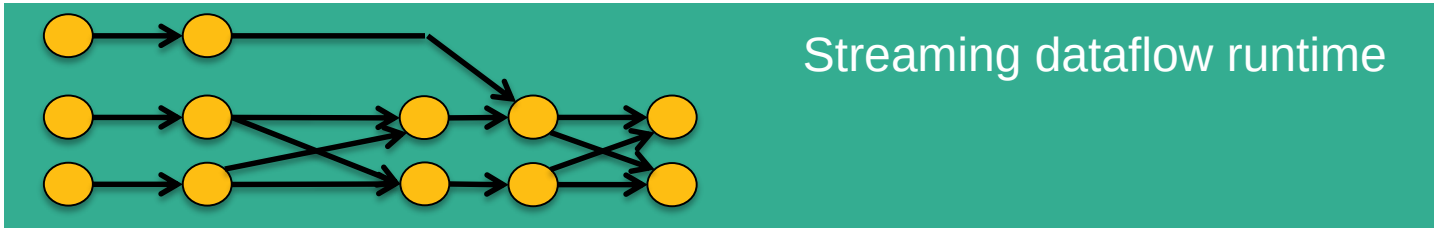
Robert Metzger
@rmetzger_
rmetzger@apache.org

dataArtisans

What is stream processing

- Real-world data is unbounded and is pushed to systems
- Right now: people are using the batch paradigm for stream analysis (there was no good stream processor available)
- New systems (Flink, Kafka) embrace streaming nature of data

Flink is a **stream processor** with **many faces**



Flink's streaming runtime

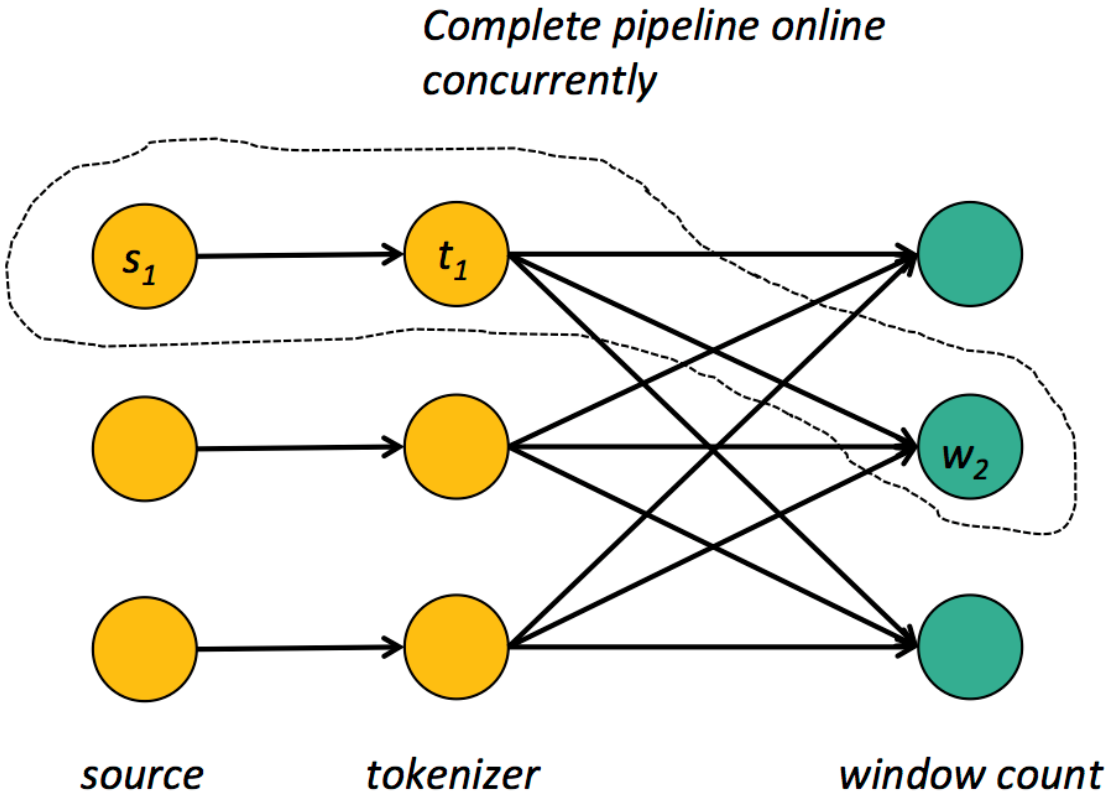
Requirements for a stream processor

- Low latency
  - Fast results (milliseconds)
- High throughput
  - handle large data amounts (millions of events per second)
- Exactly-once guarantees
  - Correct results, also in failure cases
- Programmability
  - Intuitive APIs

Pipelining

Basic building block to “keep the data moving”

- Low latency
- Operators push data forward
- Data shipping as buffers, not tuple-wise
- Natural handling of back-pressure



Fault Tolerance in streaming

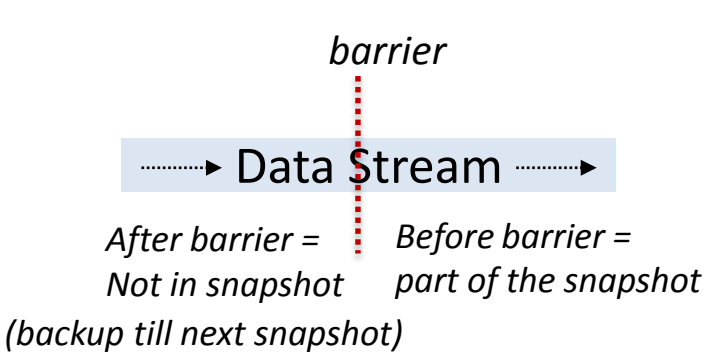
- **at least once:** ensure all operators see all events
  - Storm: Replay stream in failure case
- **Exactly once:** Ensure that operators do not perform duplicate updates to their state
  - Flink: Distributed Snapshots
  - Spark: Micro-batches on batch runtime

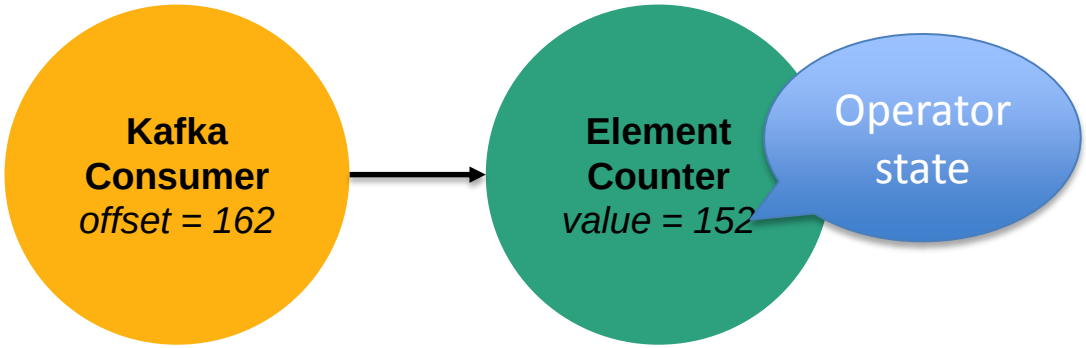
Flink's Distributed Snapshots

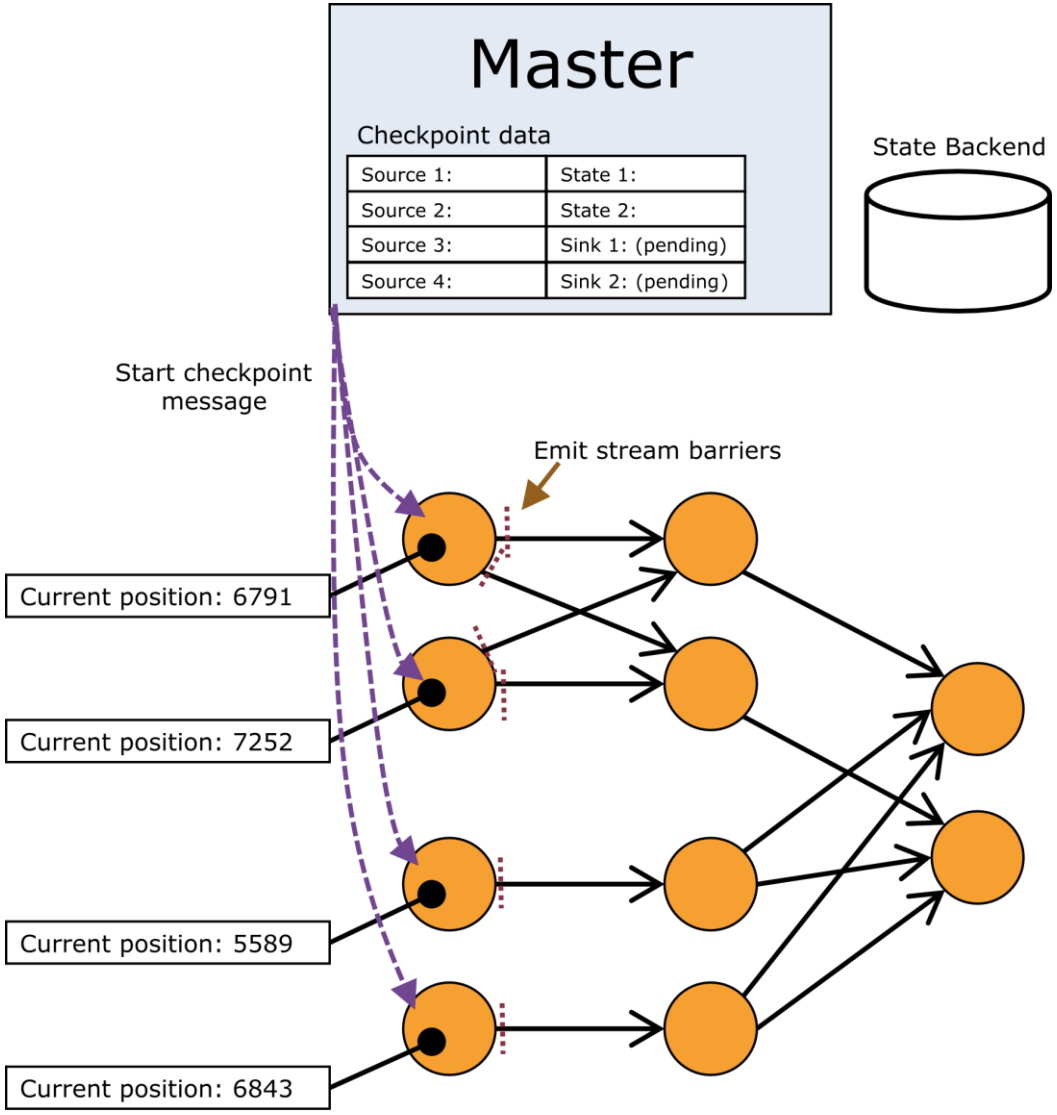
- Lightweight approach of storing the state of all operators without pausing the execution

→ high throughput, low latency

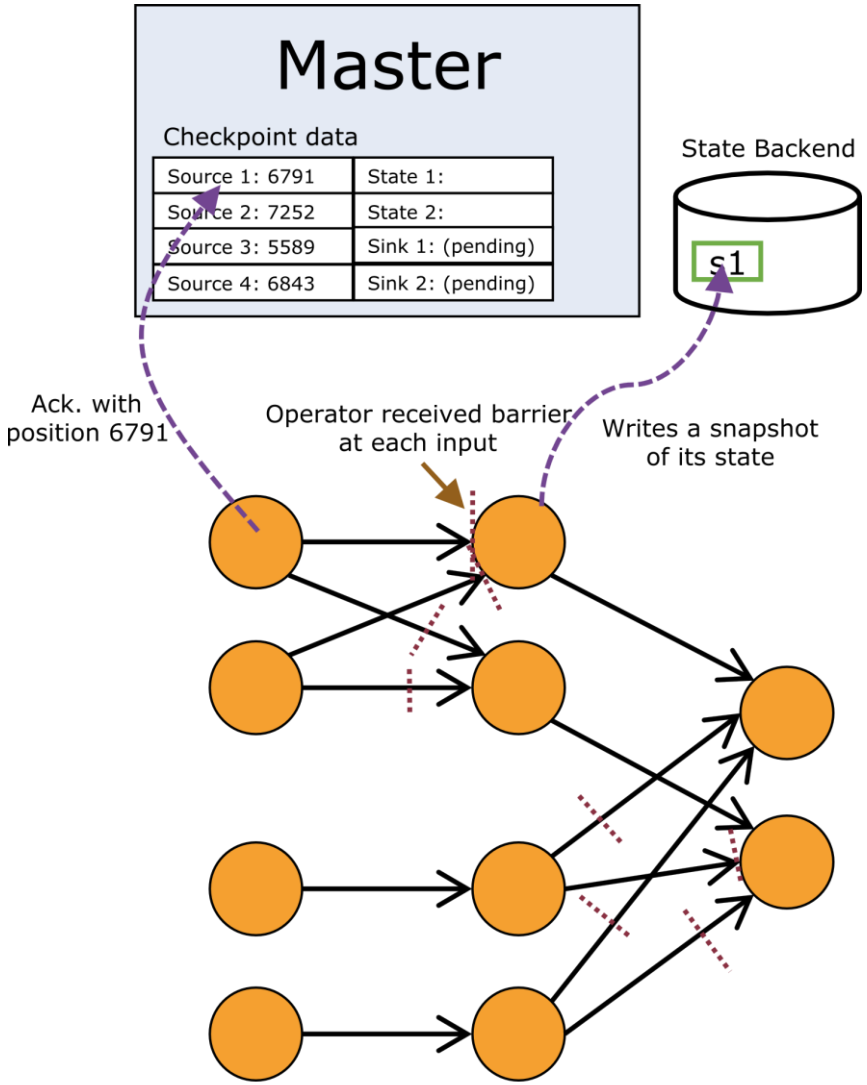
- Implemented using barriers flowing through the topology



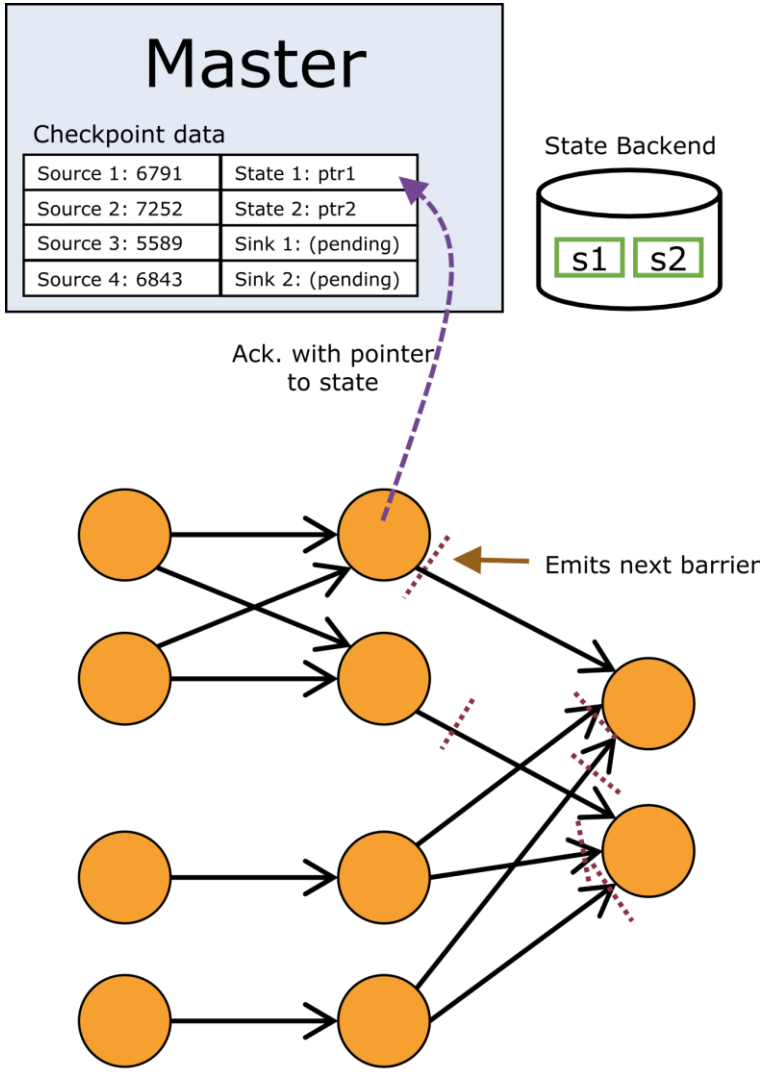




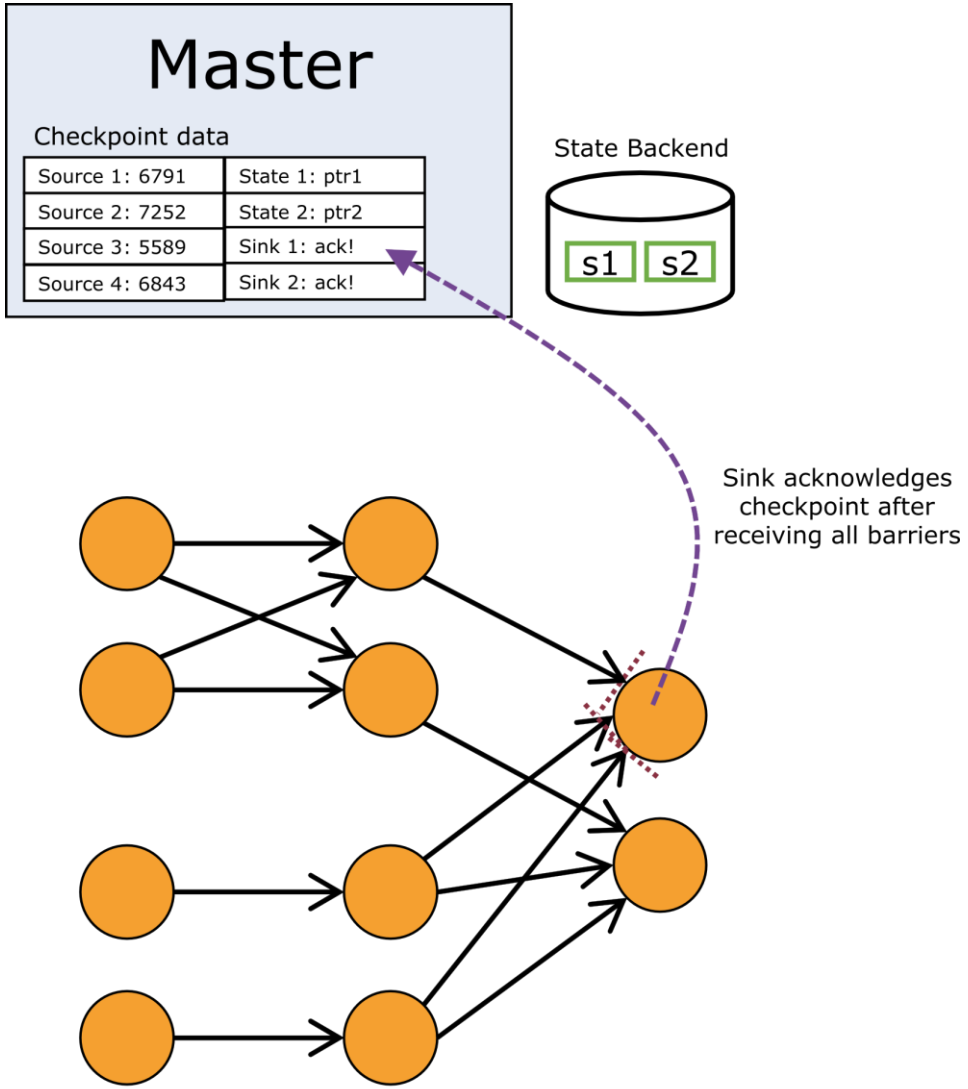
Starting Checkpoint



Checkpoint in Progress



Checkpoint in Progress

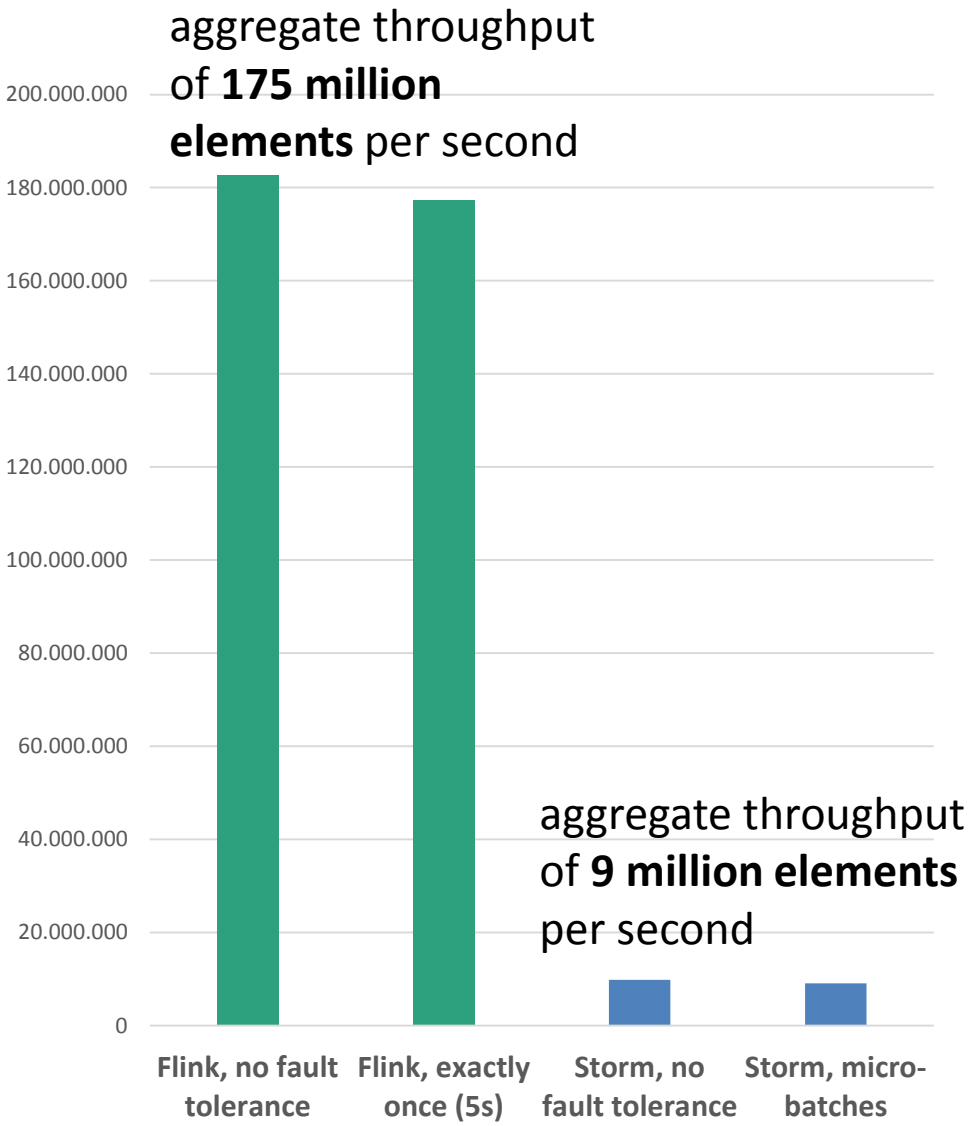


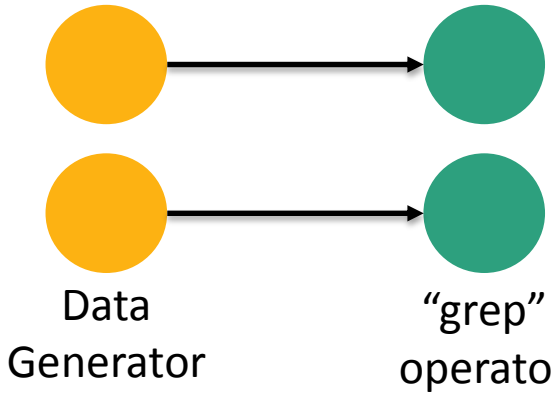
Checkpoint Completed

Best of all worlds for streaming

- Low latency
  - Thanks to pipelined engine
- Exactly-once guarantees
  - Distributed Snapshots
- High throughput
  - Controllable checkpointing overhead
- Separates app logic from recovery
  - Checkpointing interval is just a config parameter

Throughput of distributed grep

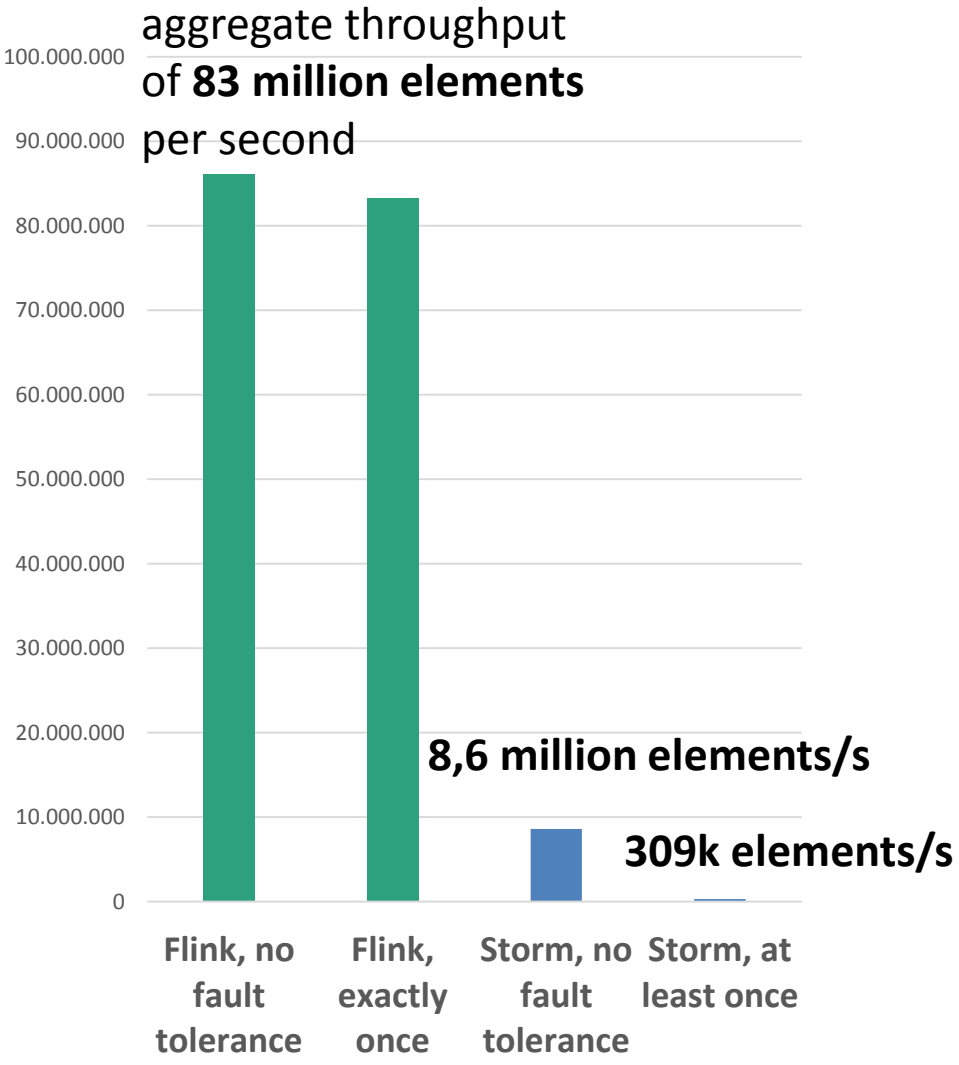


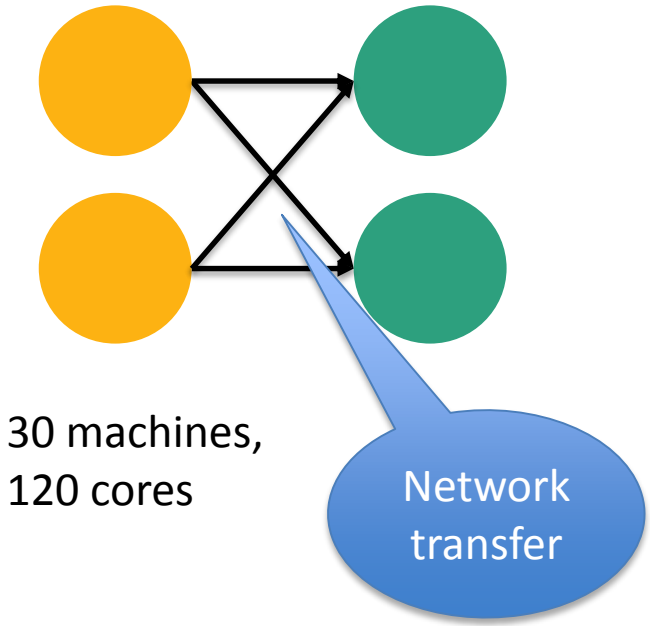


30 machines, 120 cores

- Flink achieves 20x higher throughput
- Flink throughput almost the same with and without exactly-once

Aggregate throughput for stream record grouping

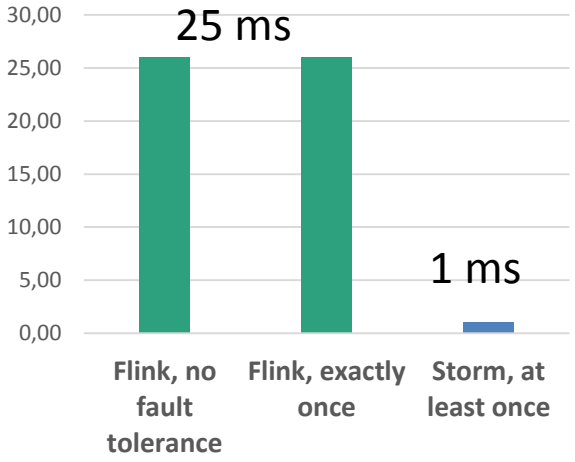


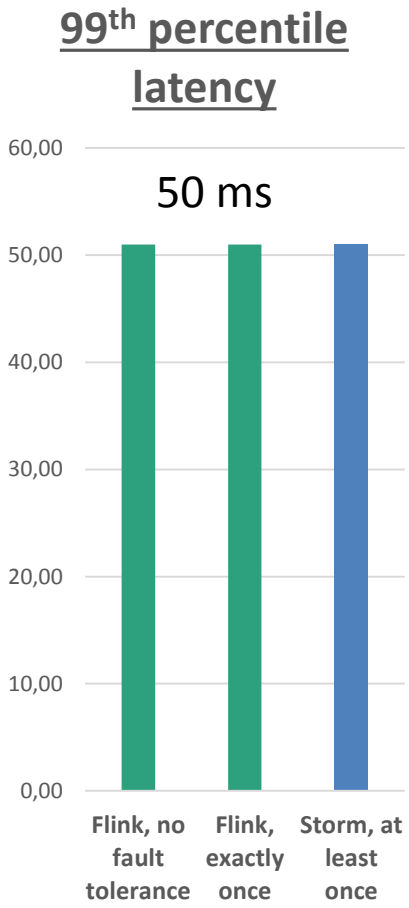


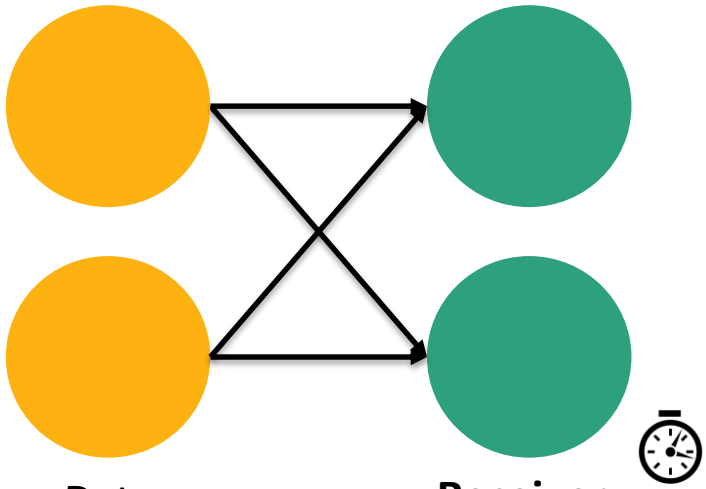
→ Flink achieves 260x higher throughput with fault tolerance

Latency in stream record grouping

- Measure time for a record to travel from source to sink

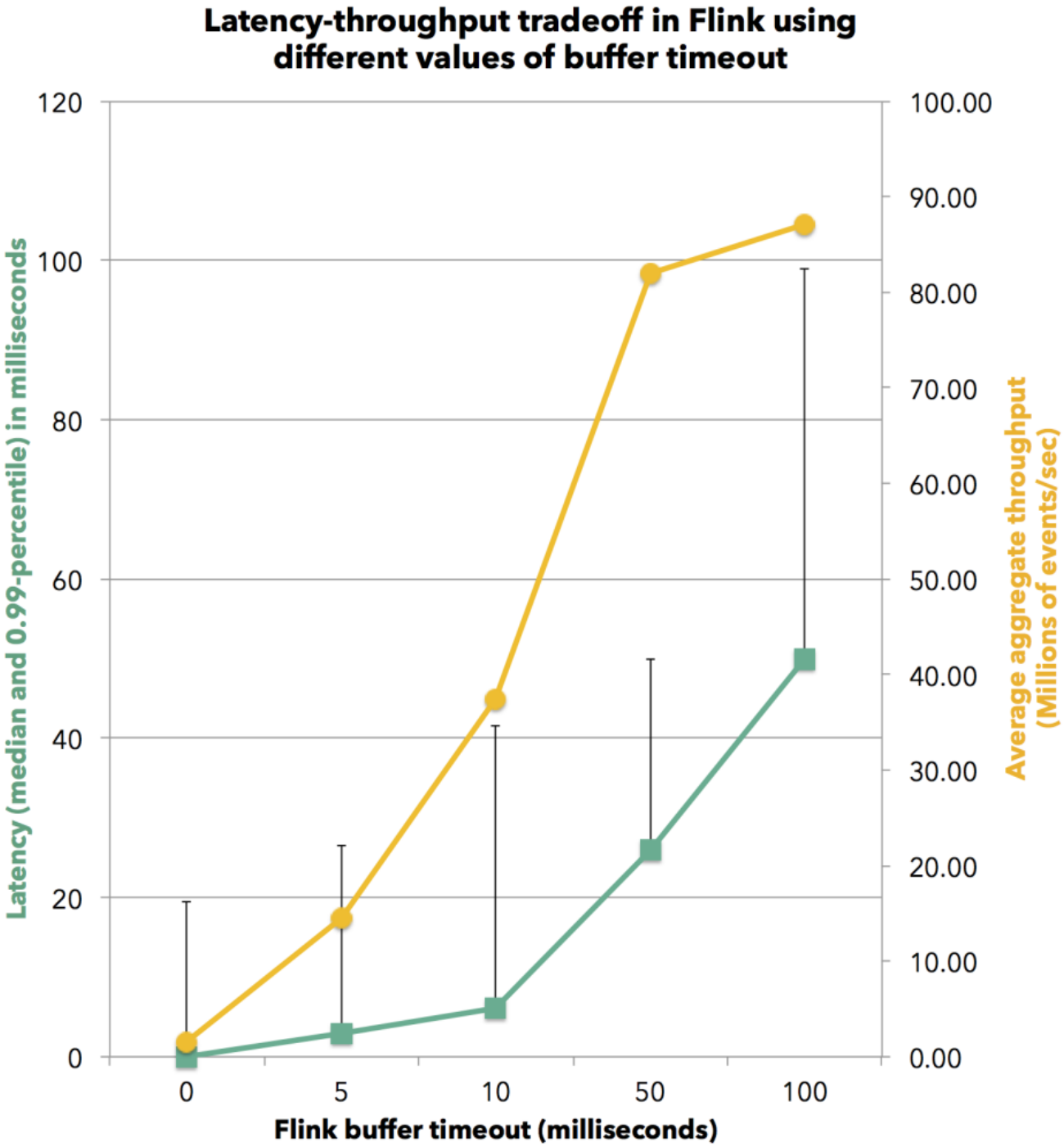






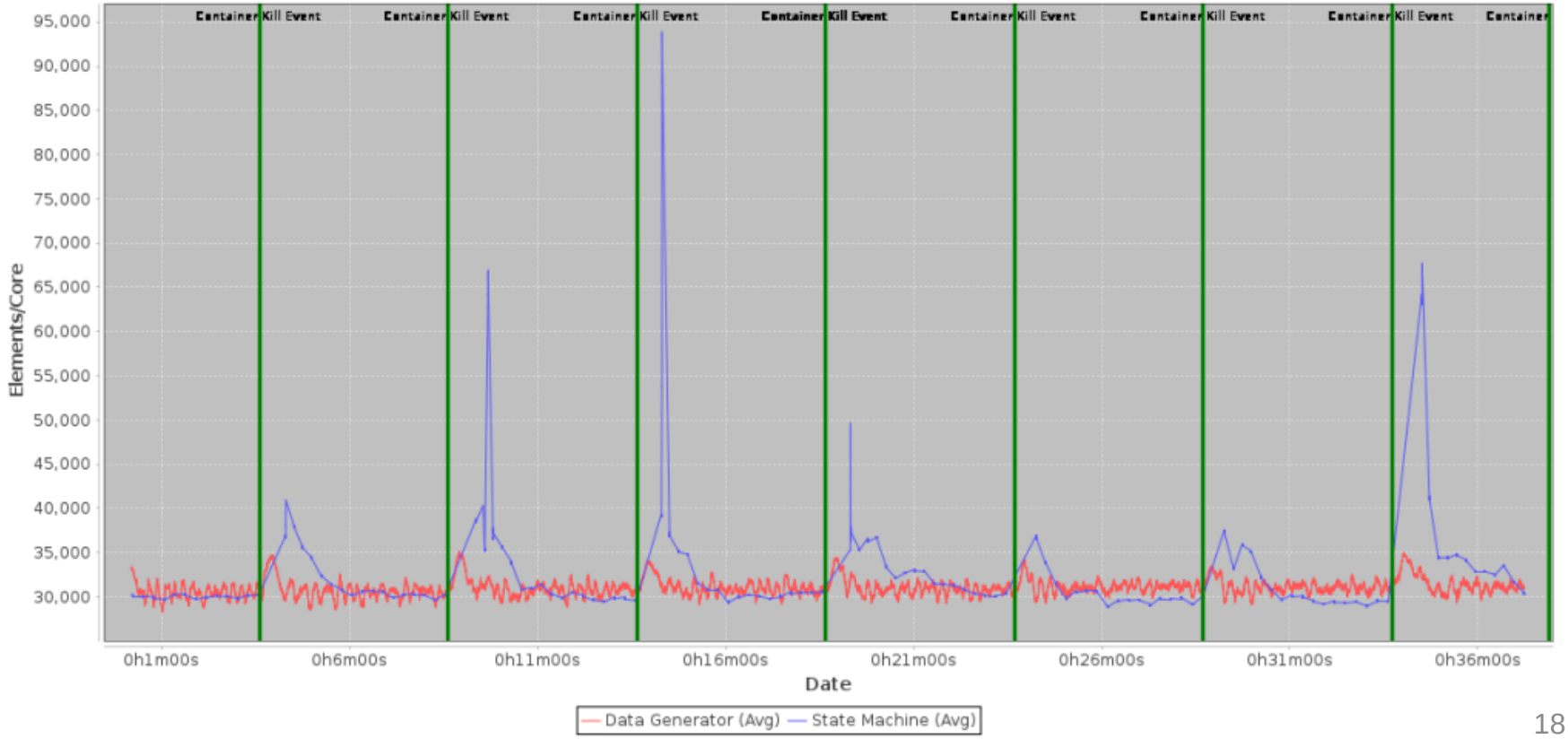
Data Generator

Receiver: Throughput / Latency measure


Latency-throughput tradeoff in Flink using different values of buffer timeout
Latency (median and 0.99-percentile) in milliseconds
Average aggregate throughput (Millions of events/sec)
Flink buffer timeout (milliseconds)
120
100
80
60
40
20
0
100.00
90.00
80.00
70.00
60.00
50.00
40.00
30.00
20.00
10.00
0.00
0
5
10
50
100

Exactly-Once with YARN Chaos Monkey

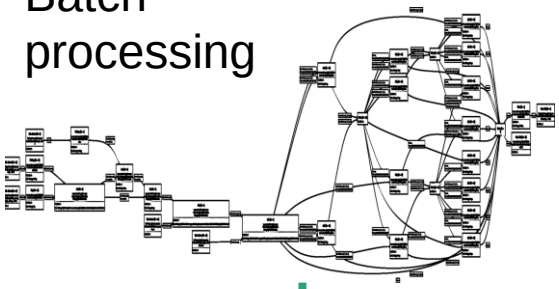
- Validate exactly-once guarantees with state-machine



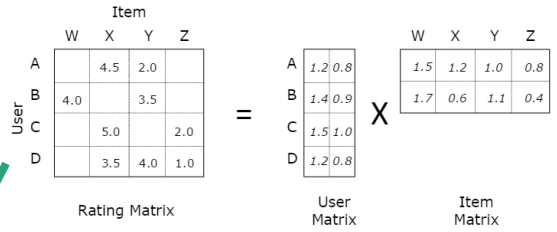
“Faces” of Flink

Faces of a stream processor

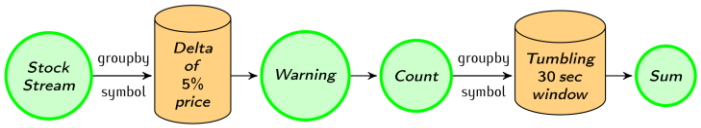
Batch processing



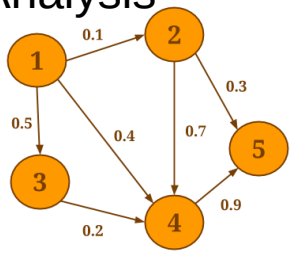
Machine Learning at scale

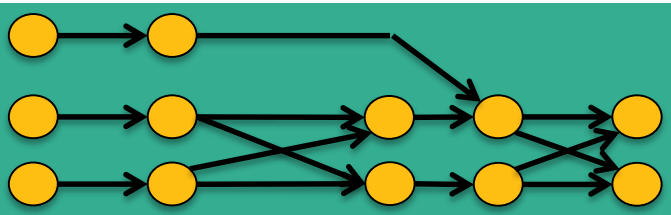


Stream processing



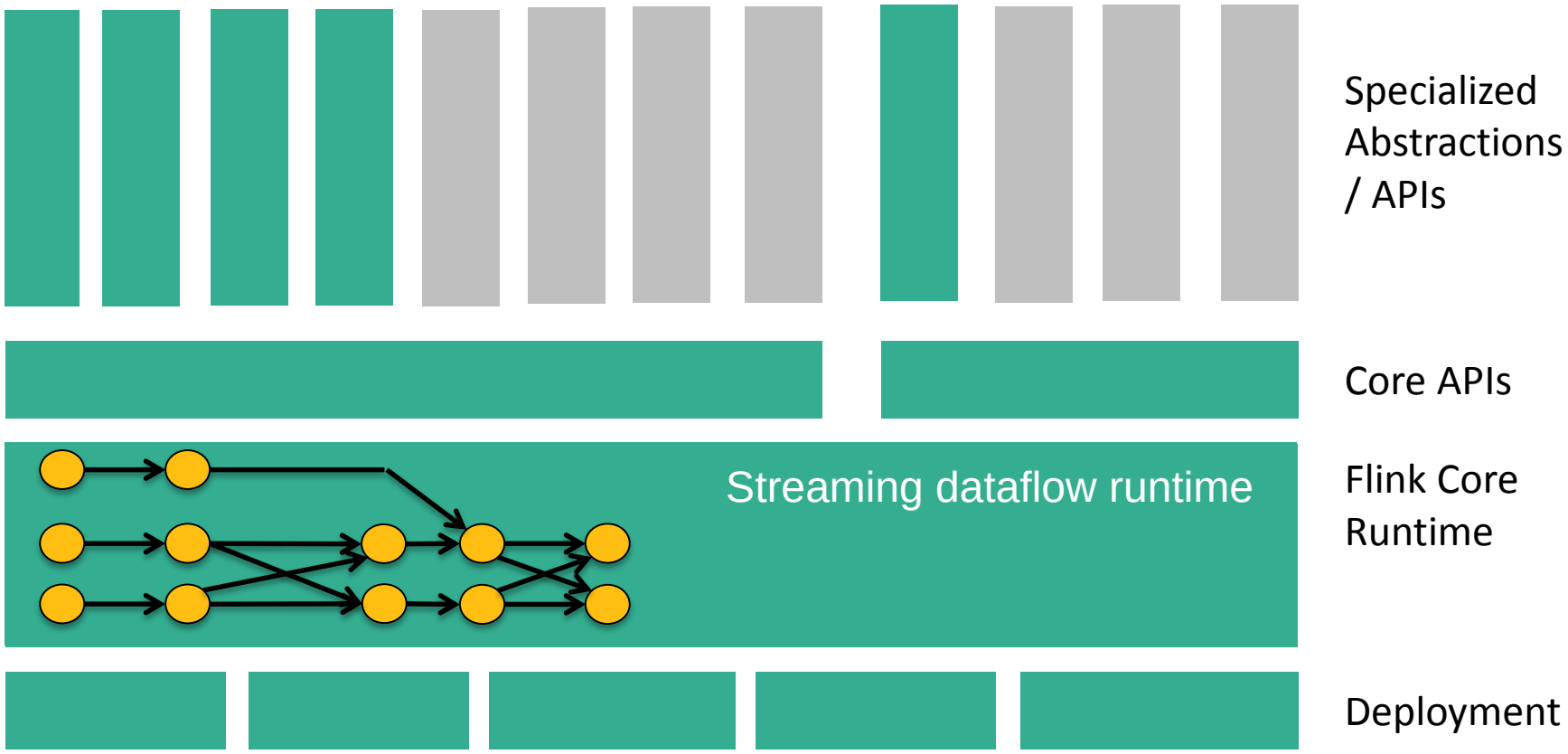
Graph Analysis





Streaming dataflow runtime

The Flink Stack



APIs for stream and batch

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

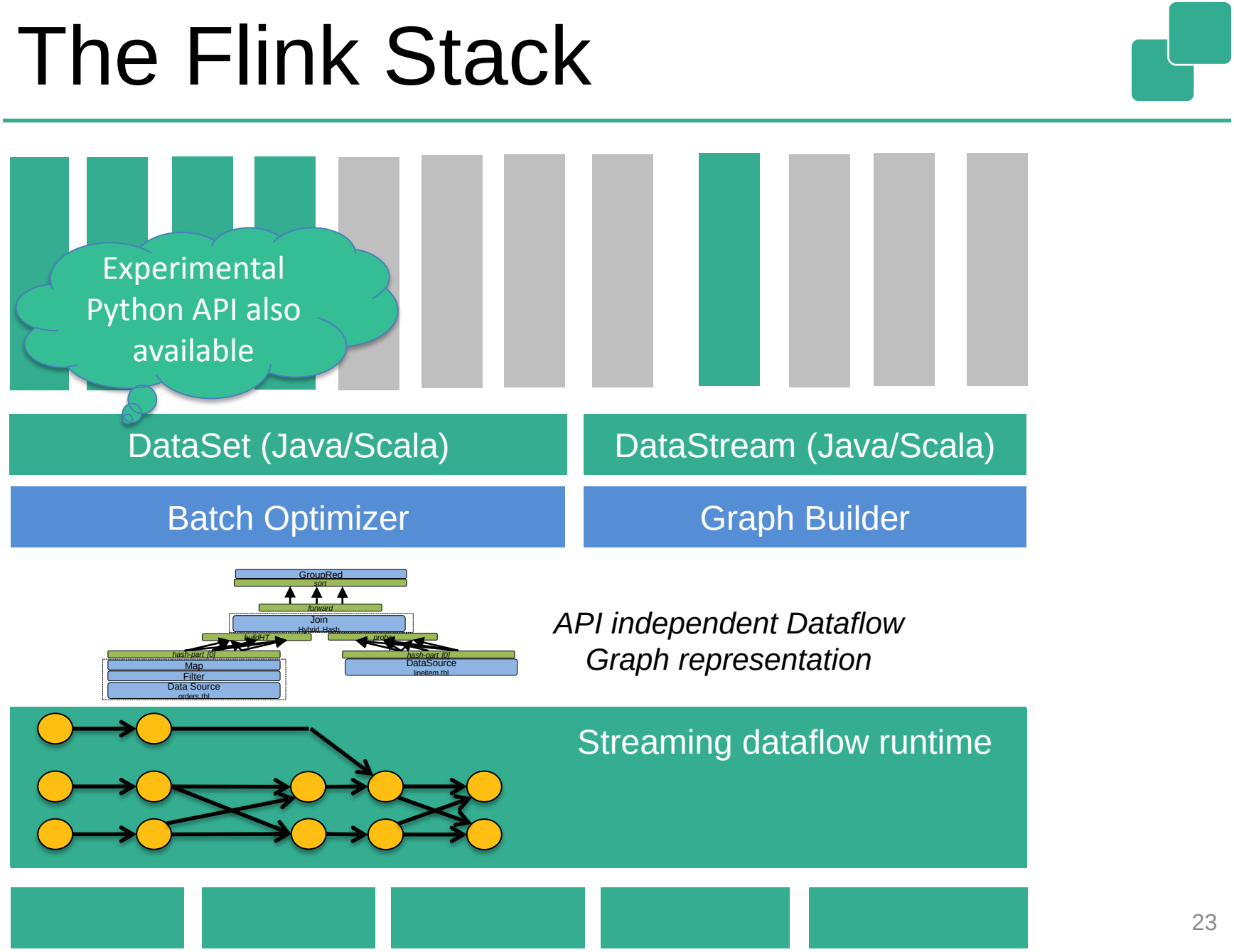
DataSet API (batch):

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

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

DataStream API (streaming):

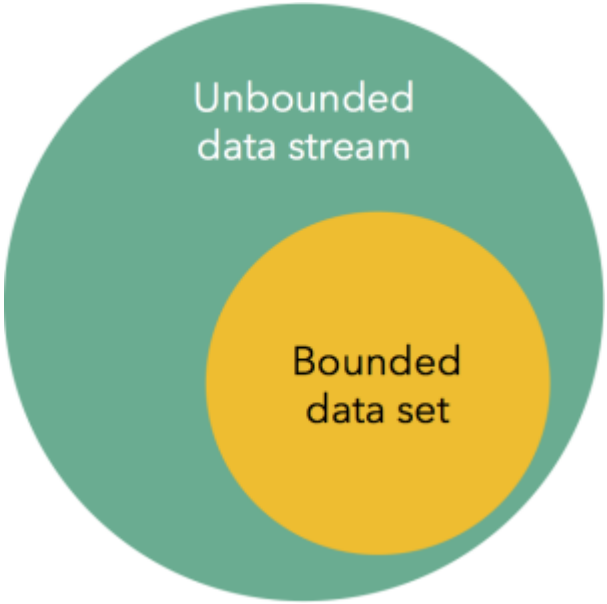
```
val lines: DataStream[String] = env.fromSocketStream(...)

lines.flatMap {line => line.split(" ")
                       .map(word => Word(word,1))}
     .window(Time.of(5,SECONDS)).every(Time.of(1,SECONDS))
     .groupBy("word").sum("frequency")
.print()
```

The Flink Stack



Batch is a special case of streaming

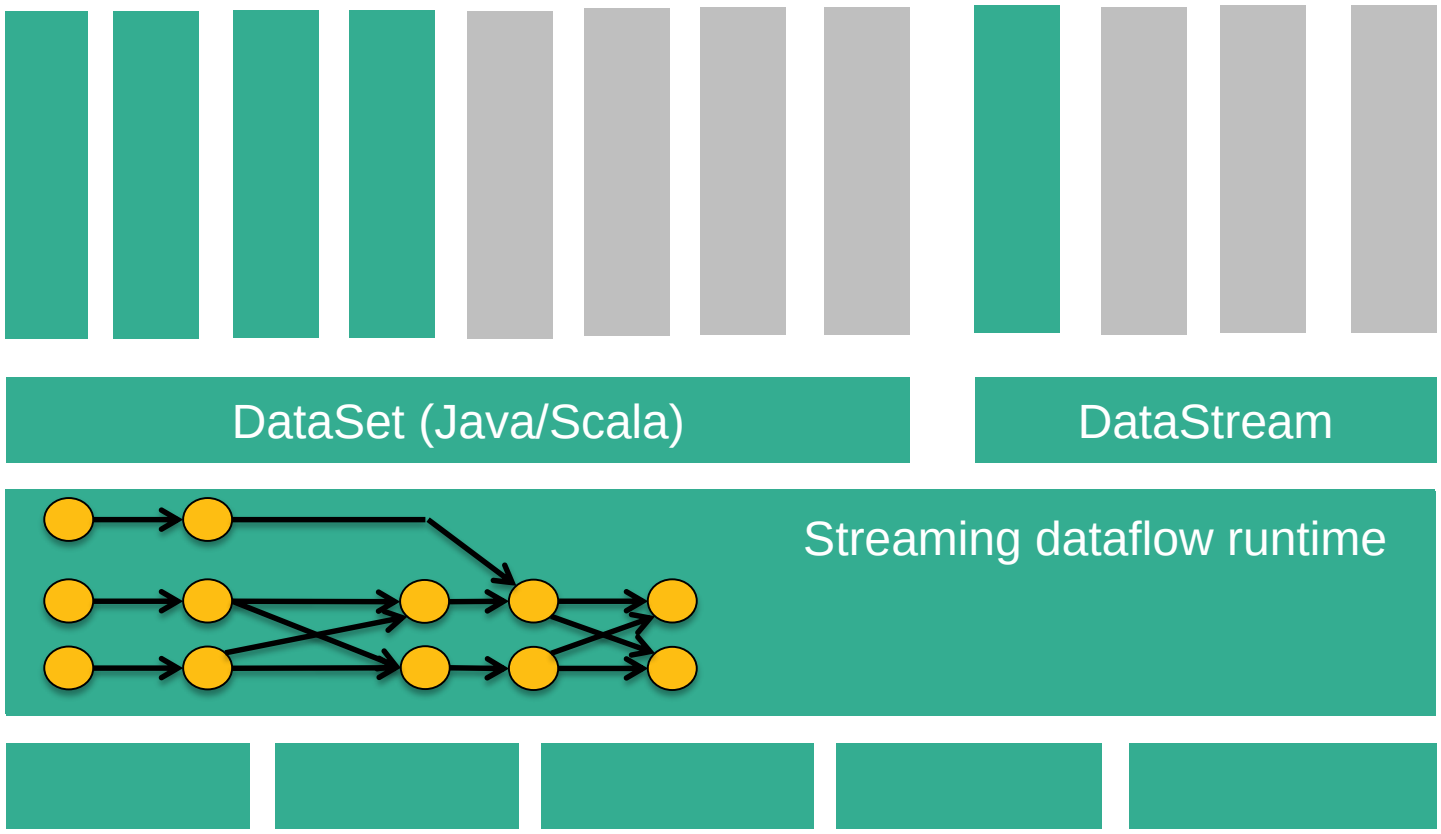
- Batch: run a bounded stream (data set) on a stream processor
- Form a global window over the entire data set for join or grouping operations



Batch-specific optimizations

- **Managed memory** on- and off-heap
  - Operators (join, sort, …) with out-of-core support
  - Optimized serialization stack for user-types
- Cost-based **Optimizer**
  - Job execution depends on data size

The Flink Stack



FlinkML: Machine Learning

- API for ML pipelines inspired by scikit-learn
- Collection of packaged algorithms
  - SVM, Multiple Linear Regression, Optimization, ALS, ...

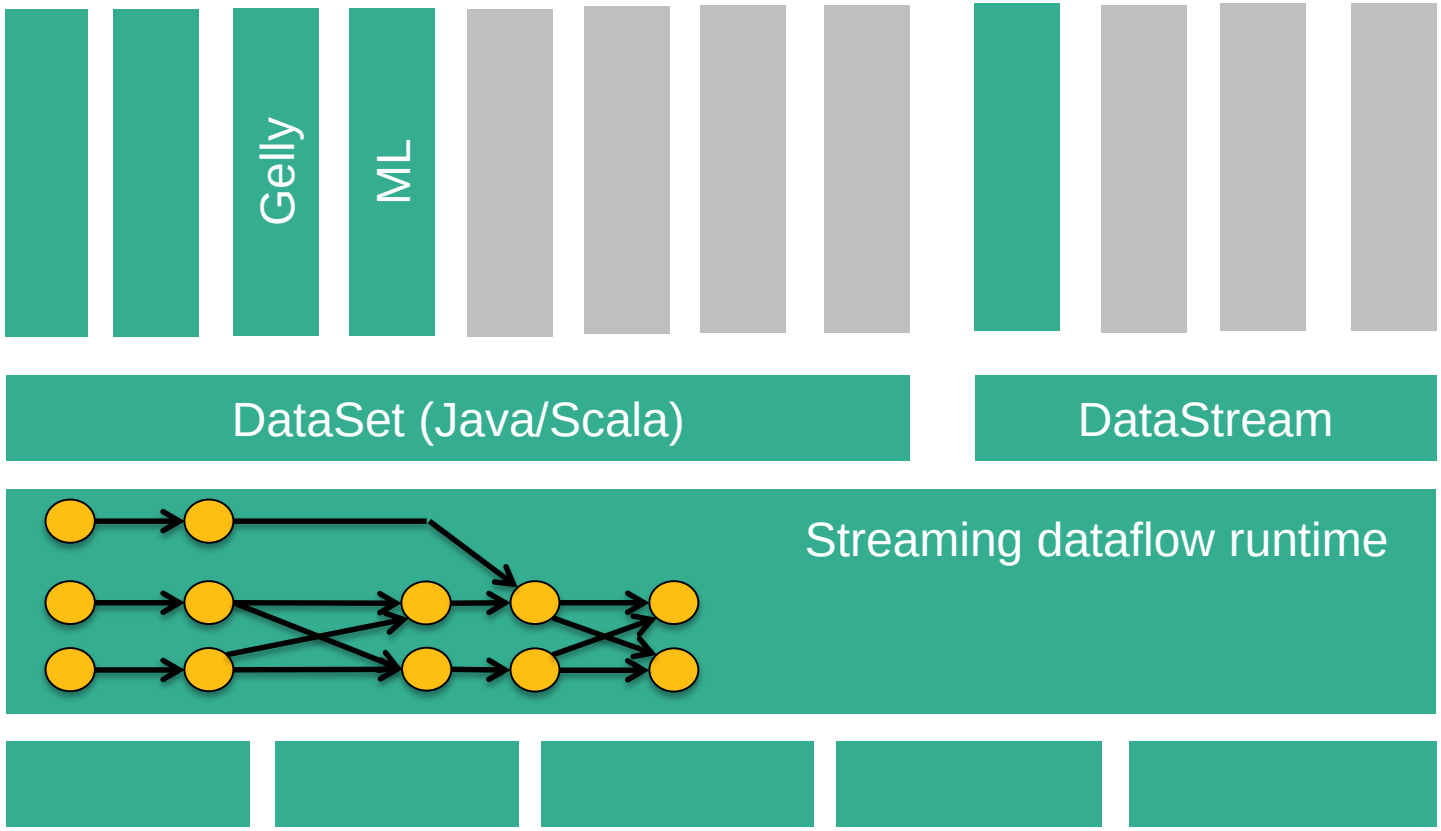
```
val trainingData: DataSet[LabeledVector] = ...
val testingData: DataSet[Vector] = ...

val scaler = StandardScaler()
val polyFeatures = PolynomialFeatures().setDegree(3)
val mlr = MultipleLinearRegression()

val pipeline = scaler.chainTransformer(polyFeatures).chainPredictor(mlr)

pipeline.fit(trainingData)

val predictions: DataSet[LabeledVector] = pipeline.predict(testingData)
```

Gelly: Graph Processing

- Graph API and library
- Packaged algorithms
  - PageRank, SSSP, Label Propagation, Community Detection, Connected Components

```
ExecutionEnvironment env = ExecutionEnvironment.getExecutionEnvironment();

Graph<Long, Long, NullValue> graph = ...

DataSet<Vertex<Long, Long>> verticesWithCommunity = graph.run(
                    new LabelPropagation<Long>(30)).getVertices();

verticesWithCommunity.print();

env.execute();
```

Flink Stack += Gelly, ML



Integration with other systems

Hadoop M/R

- Use Hadoop Input/Output Formats
- Mapper / Reducer implementations
- Hadoop's FileSystem implementations

Google Dataflow

- Run applications implemented against Google's Data Flow API **on premise** with Flink

Cascading

- Run Cascading jobs on Flink, with almost no code change
- Benefit from Flink's vastly better performance than MapReduce

Zeppelin

- Interactive, web-based data exploration

SAMOA

- Machine learning on data streams

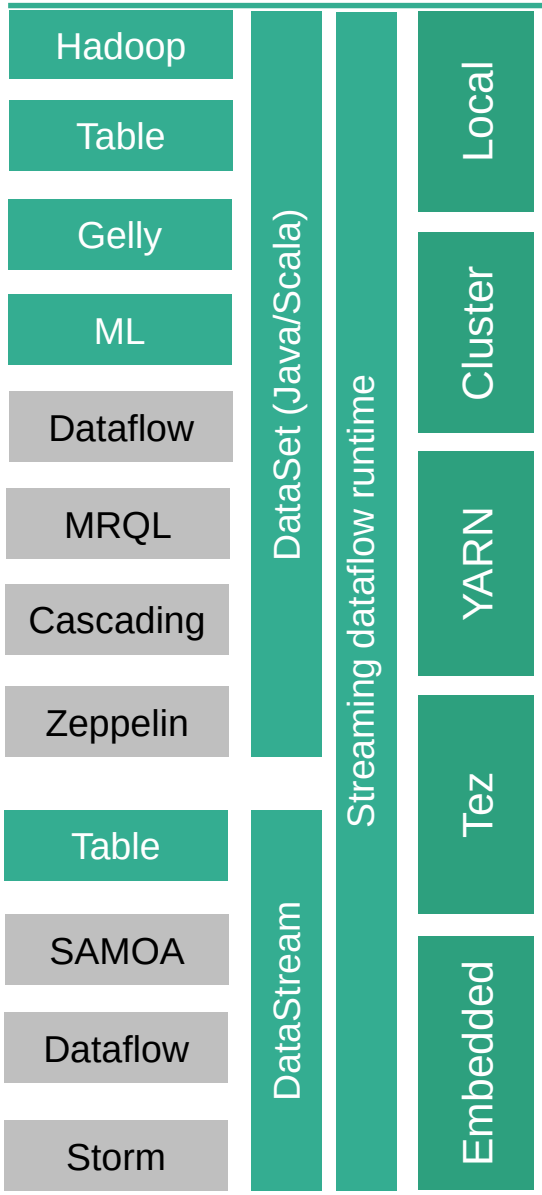
Storm

- Compatibility layer for running Storm code
- FlinkTopologyBuilder: one line replacement for existing jobs
- Wrappers for Storm Spouts and Bolts
- Coming soon: Exactly-once with Storm

DataSet

DataStream

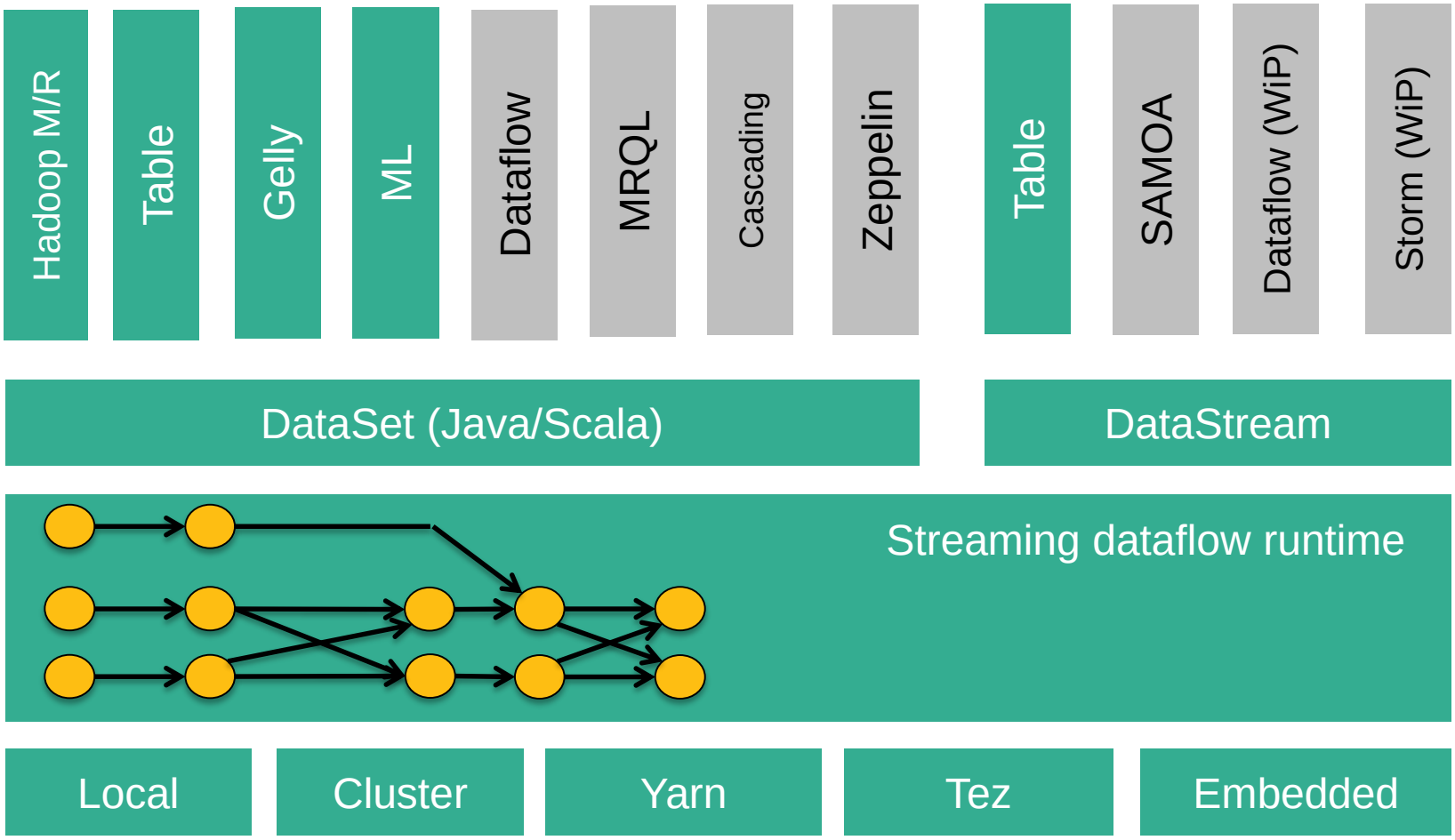
Deployment options



- Start Flink in your IDE / on your machine
- Local debugging / development using the same code as on the cluster
- “bare metal” standalone installation of Flink on a cluster

- Flink on Hadoop YARN (Hadoop 2.2.0+)
- Restarts failed containers
- Support for Kerberos-secured YARN/HDFS setups

The full stack



Closing

tl;dr Summary

Flink is a software stack of

- Streaming runtime
  - low latency
  - high throughput
  - fault tolerant, exactly-once data processing
- Rich APIs for batch and stream processing
  - library ecosystem
  - integration with many systems
- A great community of devs and users
- Used in production

What is currently happening?

- Features in progress:
  - Master High Availability
  - Vastly improved monitoring GUI
  - Watermarks / Event time processing / Windowing rework
  - Graduate Streaming API out of Beta
- 0.10.0-milestone-1 is currently voted

How do I get started?

Mailing Lists: (news | user | dev)@flink.apache.org

Twitter: @ApacheFlink

Blogs: flink.apache.org/blog, data-artisans.com/blog/

IRC channel: irc.freenode.net#flink

Start Flink on YARN in 4 commands:

```
# get the hadoop2 package from the Flink download page at
# http://flink.apache.org/downloads.html
wget <download url>
tar xvzf flink-0.9.1-bin-hadoop2.tgz
cd flink-0.9.1/
./bin/flink run -m yarn-cluster -yn 4 ./examples/flink-java-examples-0.9.1-WordCount.jar
```

Flink Forward

BERLIN 12/13 OCT 2015

Flink Forward: 2 days conference with free training in Berlin, Germany

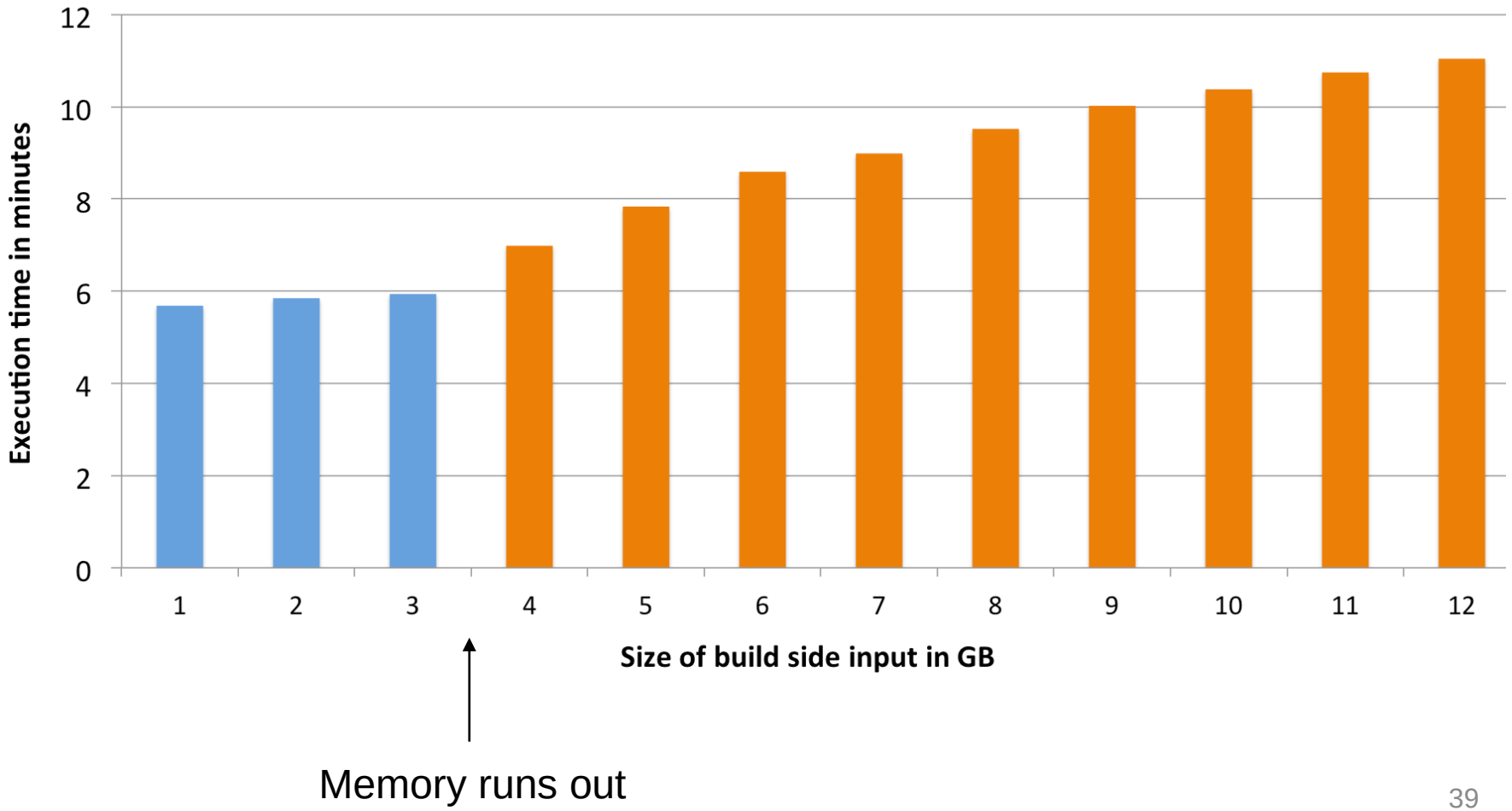
- Schedule: http://flink-forward.org/?post_type=day

Google Capital One aMADEUS Your technology partner ERICSSON

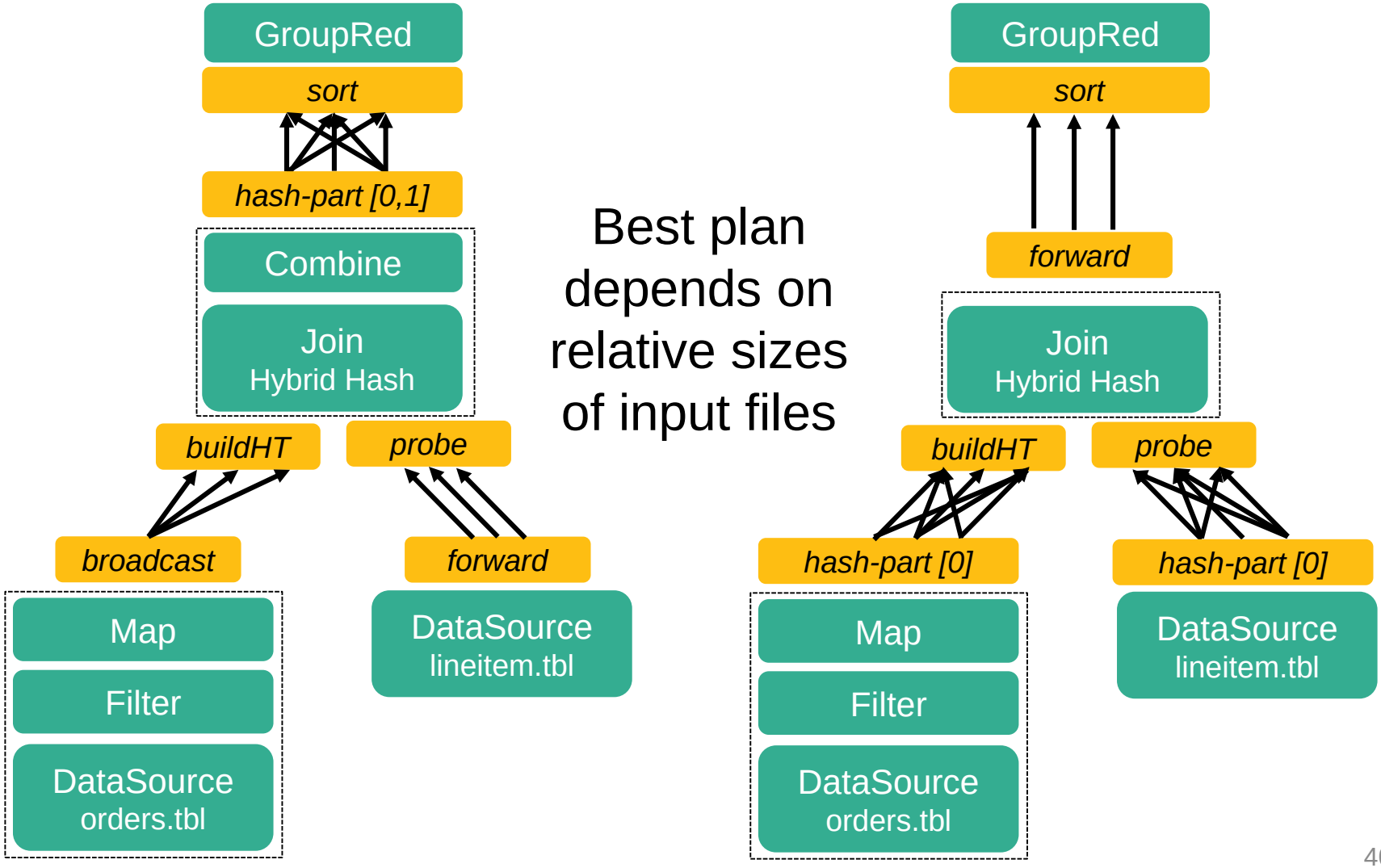
IBM HUAWEI redhat Telefónica

Appendix

Managed (off-heap) memory and out-of-core support

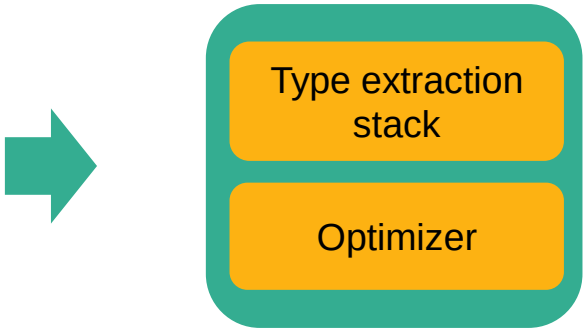


Cost-based Optimizer

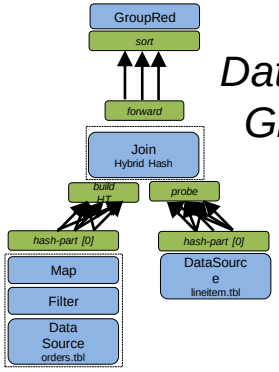


```
case class Path (from: Long, to:
Long)
val tc = edges.iterate(10) {
  paths: DataSet[Path] =>
    val next = paths
      .join(edges)
      .where("to")
      .equalTo("from") {
        (path, edge) =>
          Path(path.from, edge.to)
      }
      .union(paths)
      .distinct()
    next
}
```

Program

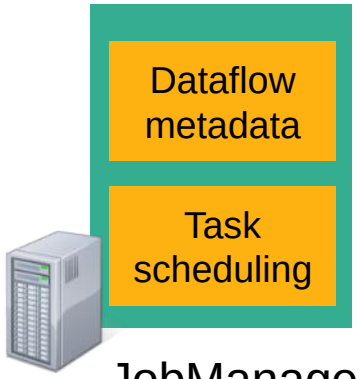


Pre-flight (Client)



Dataflow Graph

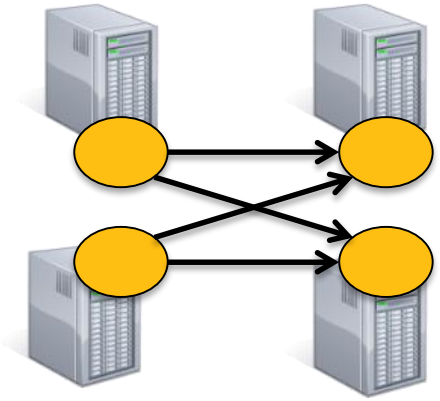
Local



JobManager

deploy operators

track intermediate results

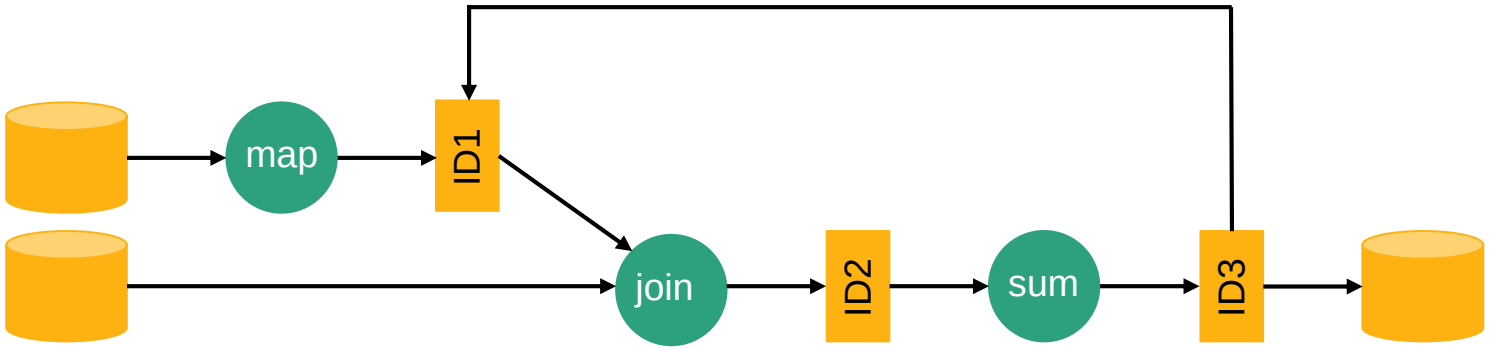


TaskManagers

Cluster: YARN, Standalone

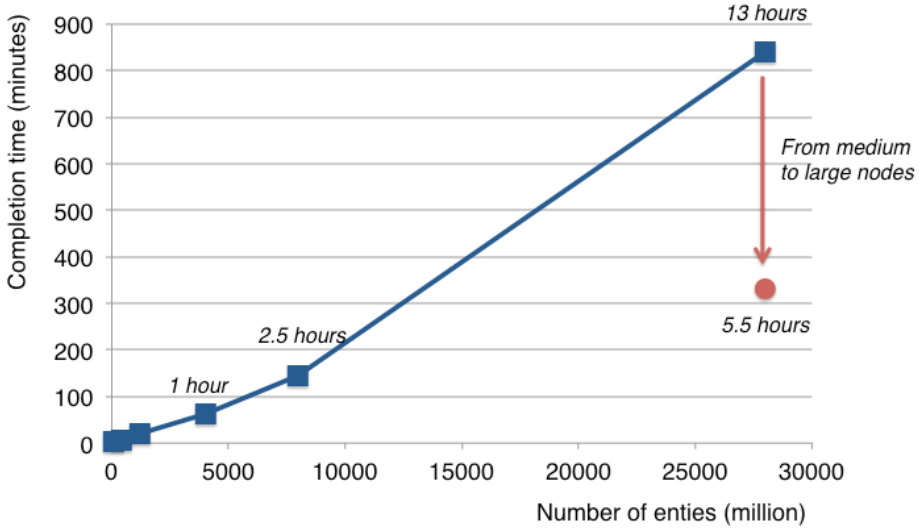
Iterative processing in Flink

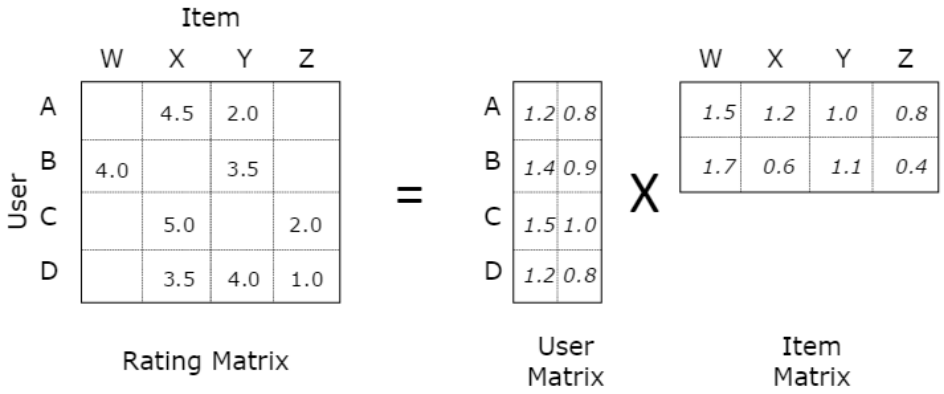
Flink offers built-in iterations and delta iterations to execute ML and graph algorithms efficiently



Example: Matrix Factorization

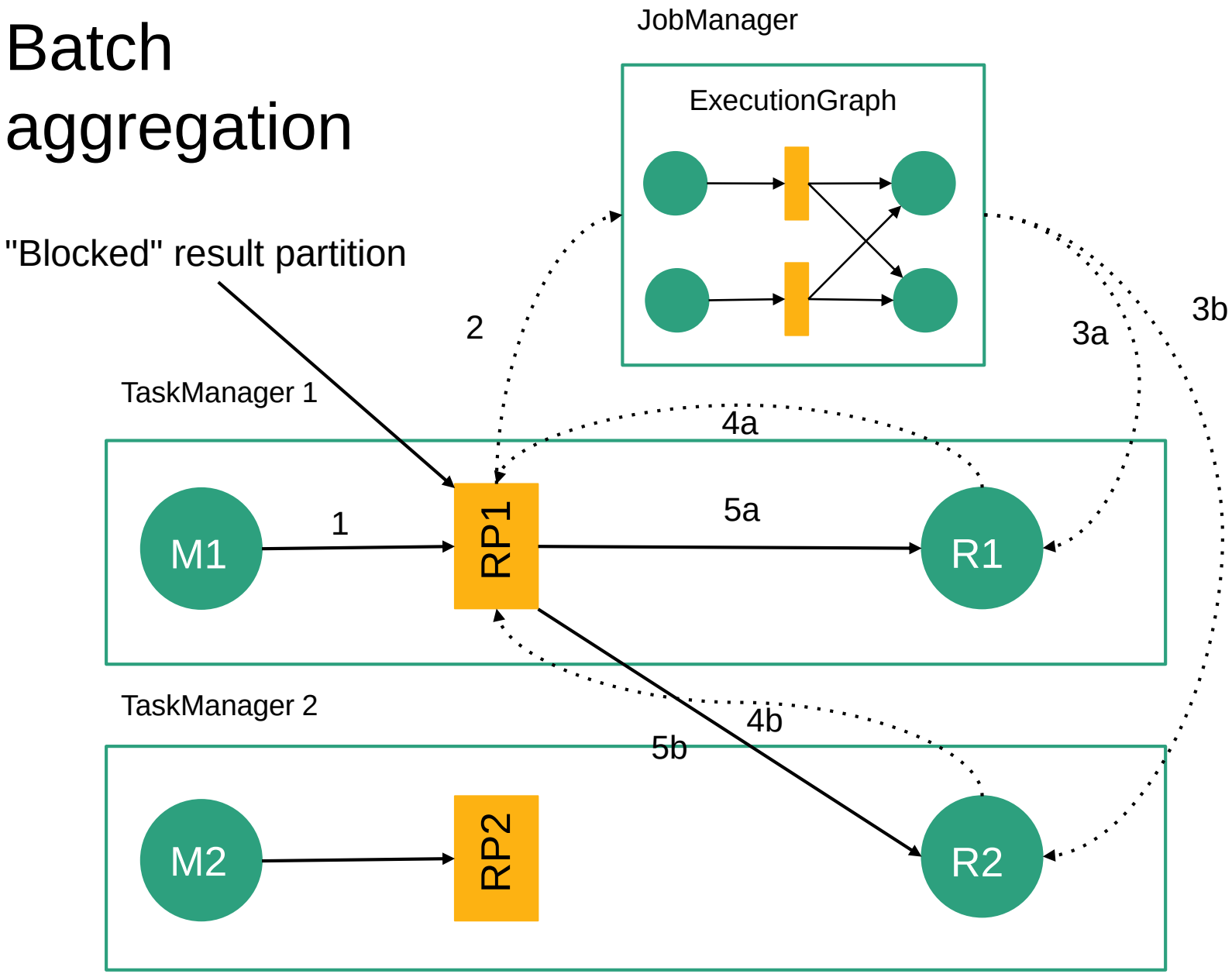
Factorizing a matrix with 28 billion ratings for recommendations





More at: http://data-artisans.com/computing-recommendations-with-flink.html

Batch aggregation



Streaming window aggregation