dataArtisans

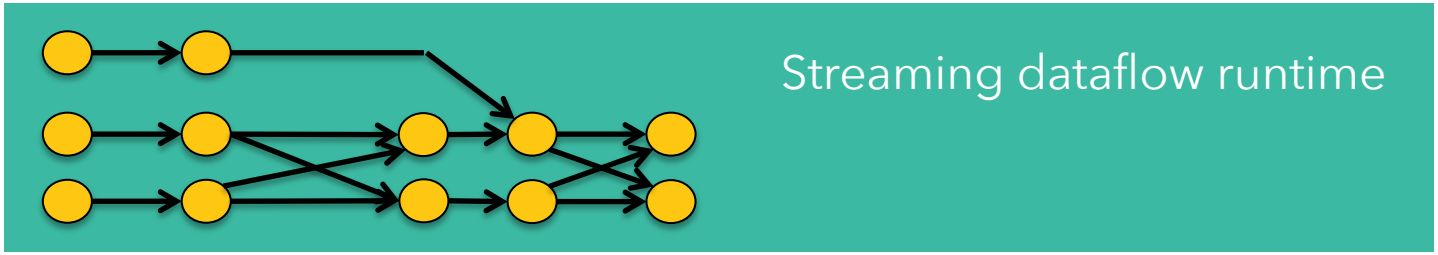
Apache Flink® Training

System Overview

Apache Flink

A stream processor

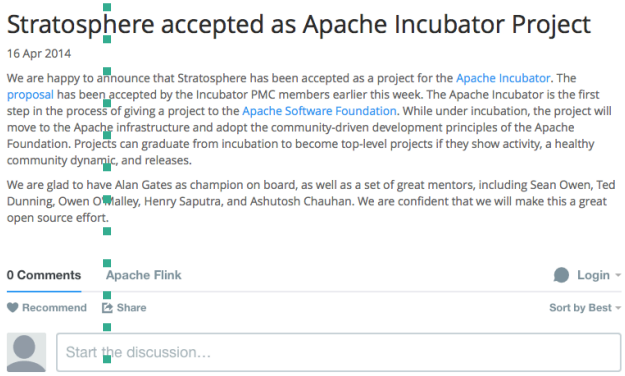
with many applications



1 year of Flink - code

April 2014

April 2015



Stratosphere accepted as Apache Incubator Project

16 Apr 2014

We are happy to announce that Stratosphere has been accepted as a project for the Apache Incubator. The proposal has been accepted by the Incubator PMC members earlier this week. The Apache Incubator is the first step in the process of giving a project to the Apache Software Foundation. While under incubation, the project will move to the Apache infrastructure and adopt the community-driven development principles of the Apache Foundation. Projects can graduate from incubation to become top-level projects if they show activity, a healthy community dynamic, and releases.

We are glad to have Alan Gates as champion on board, as well as a set of great mentors, including Sean Owen, Ted Dunning, Owen O'Malley, Henry Saputra, and Ashutosh Chauhan. We are confident that we will make this a great open source effort.

0 Comments Apache Flink Login

Recommend Share Sort by Best

Start the discussion...

DataSet API (Java/Scala)

Flink core

Local Remote Yarn

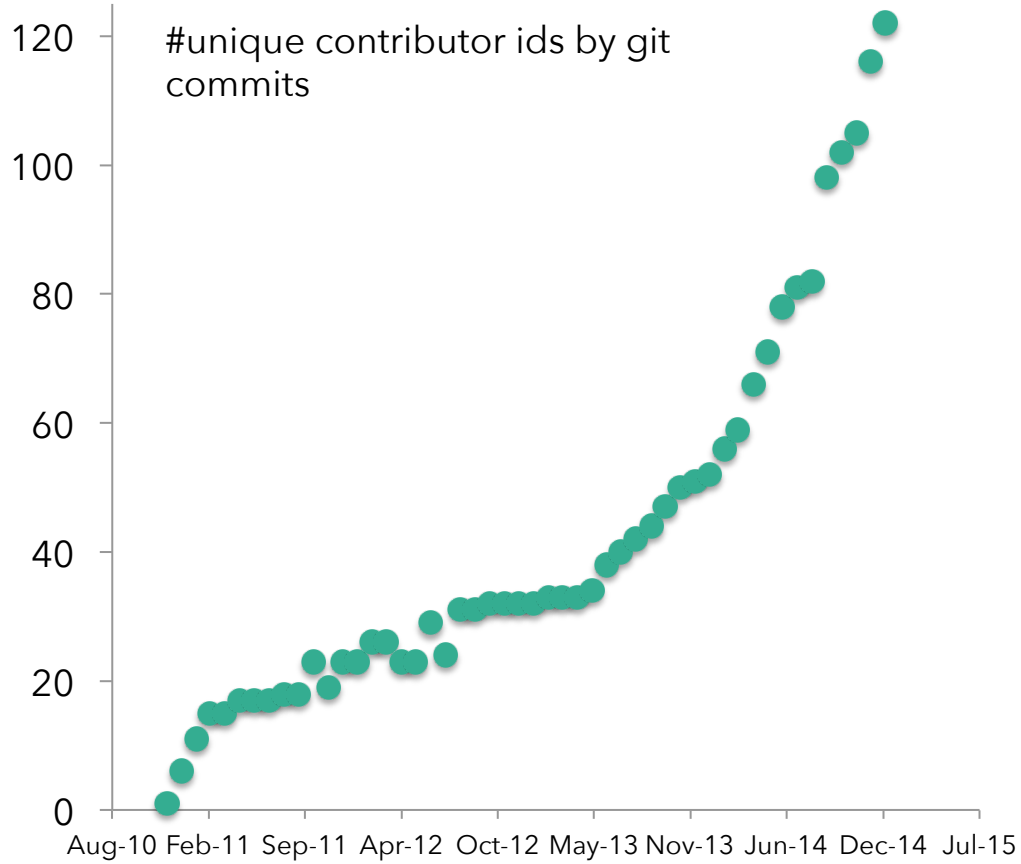
Hadoop M/R Python Gelly Table ML Dataflow MRQL Table SAMOA Dataflow

DataSet (Java/Scala) DataStream (Java/Scala)

Flink core

Local Remote Yarn Tez Embedded

Flink Community



In top 5 of Apache's big data projects after one year in the Apache Software Foundation

The Apache Way

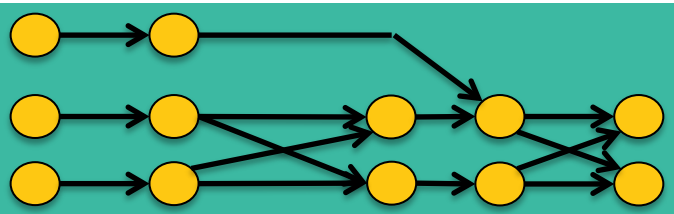
- Flink is an Apache **top-level project**

- Independent, **non-profit** organization
- **Community-driven** open source software development approach
- **Public communication** and open to new contributors

What is Apache Flink?

Hadoop M/R | Table | Gelly | ML | Dataflow | MRQL | Cascading (WiP) | Table | SAMOA | Dataflow (WiP)

DataSet (Java/Scala/Python) | DataStream (Java/Scala)

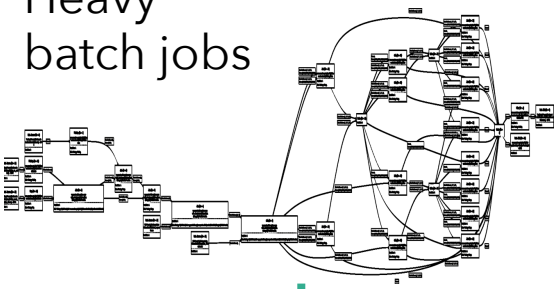


Streaming dataflow runtime

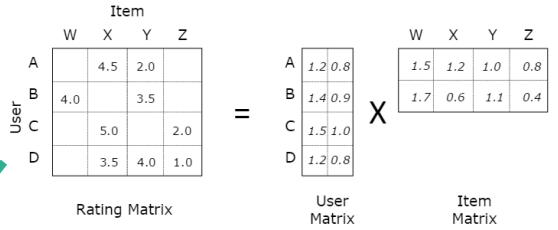
Local | Remote | Yarn | Tez | Embedded

Native workload support

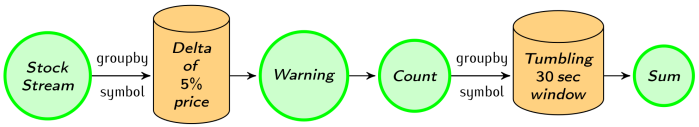
Heavy batch jobs

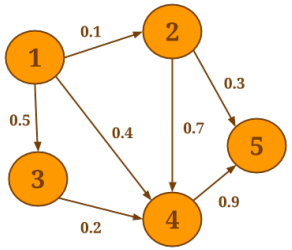


Machine Learning at scale



Streaming topologies



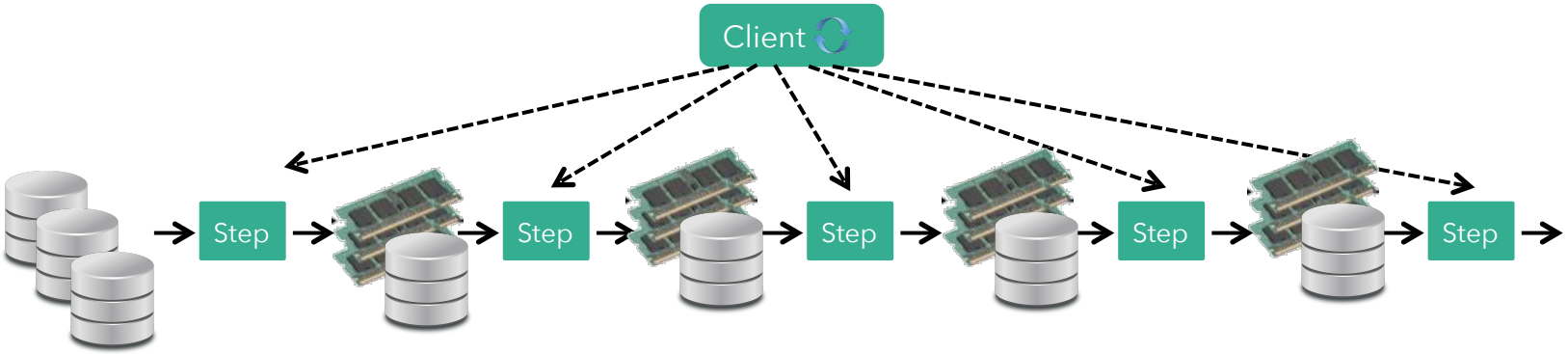


How can an engine **natively** support all these workloads?

And what does native **mean**?

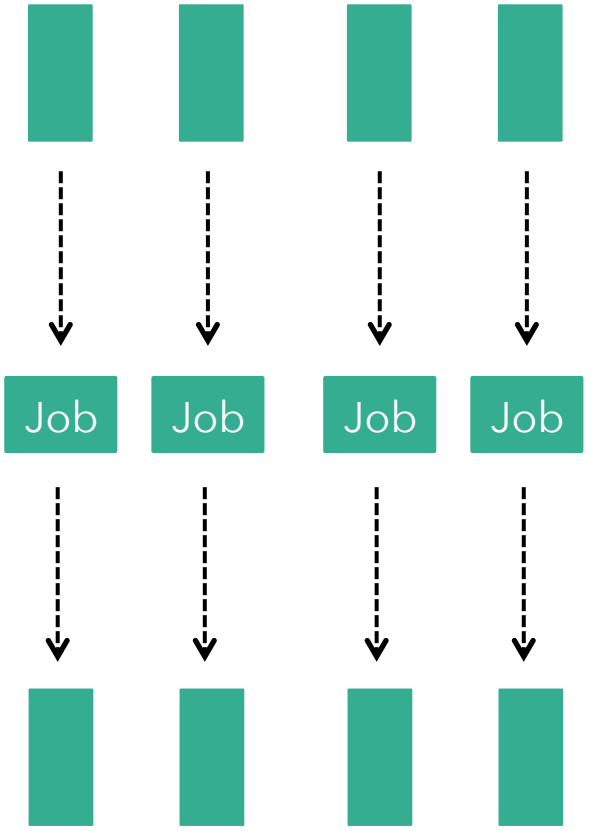
E.g.: Non-native iterations

```
for (int i = 0; i < maxIterations; i++) {
    // Execute MapReduce job
}
```



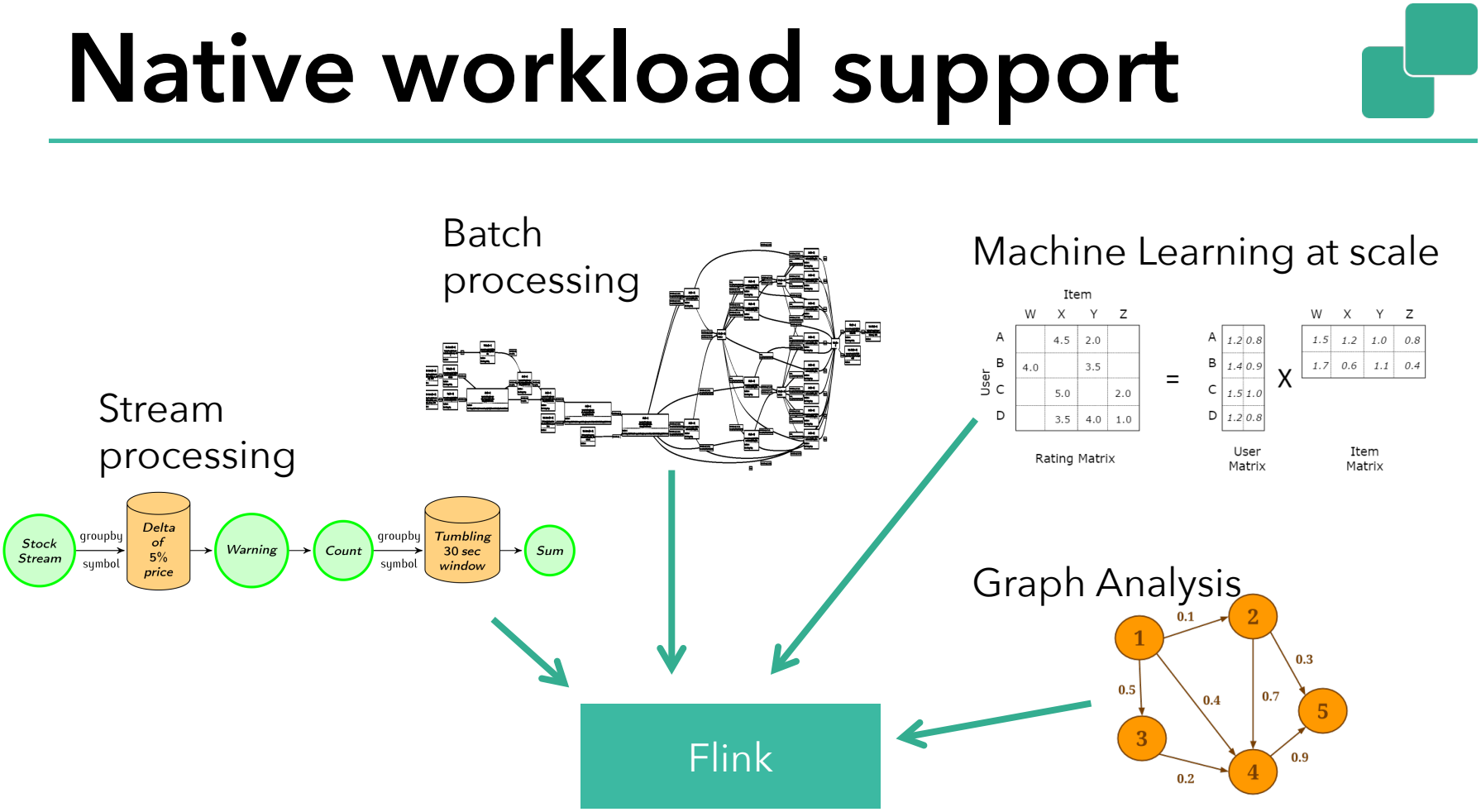
E.g.: Non-native streaming

stream discretizer



```
while (true) {
  // get next few records
  // issue batch job
}
```

Native workload support

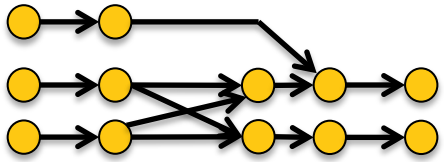


How can an engine **natively** support all these workloads?

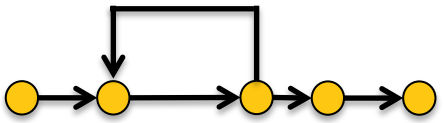
And what does "native" **mean**?

Flink Engine

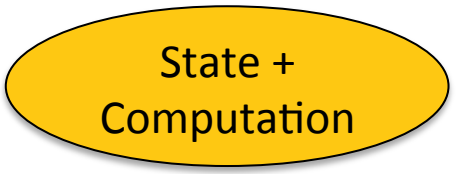
1. Execute everything as streams



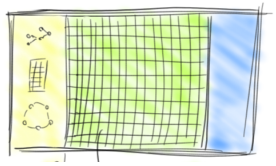
2. Iterative (cyclic) dataflows



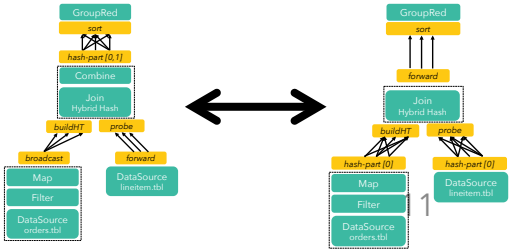
3. Mutable state



4. Operate on managed memory

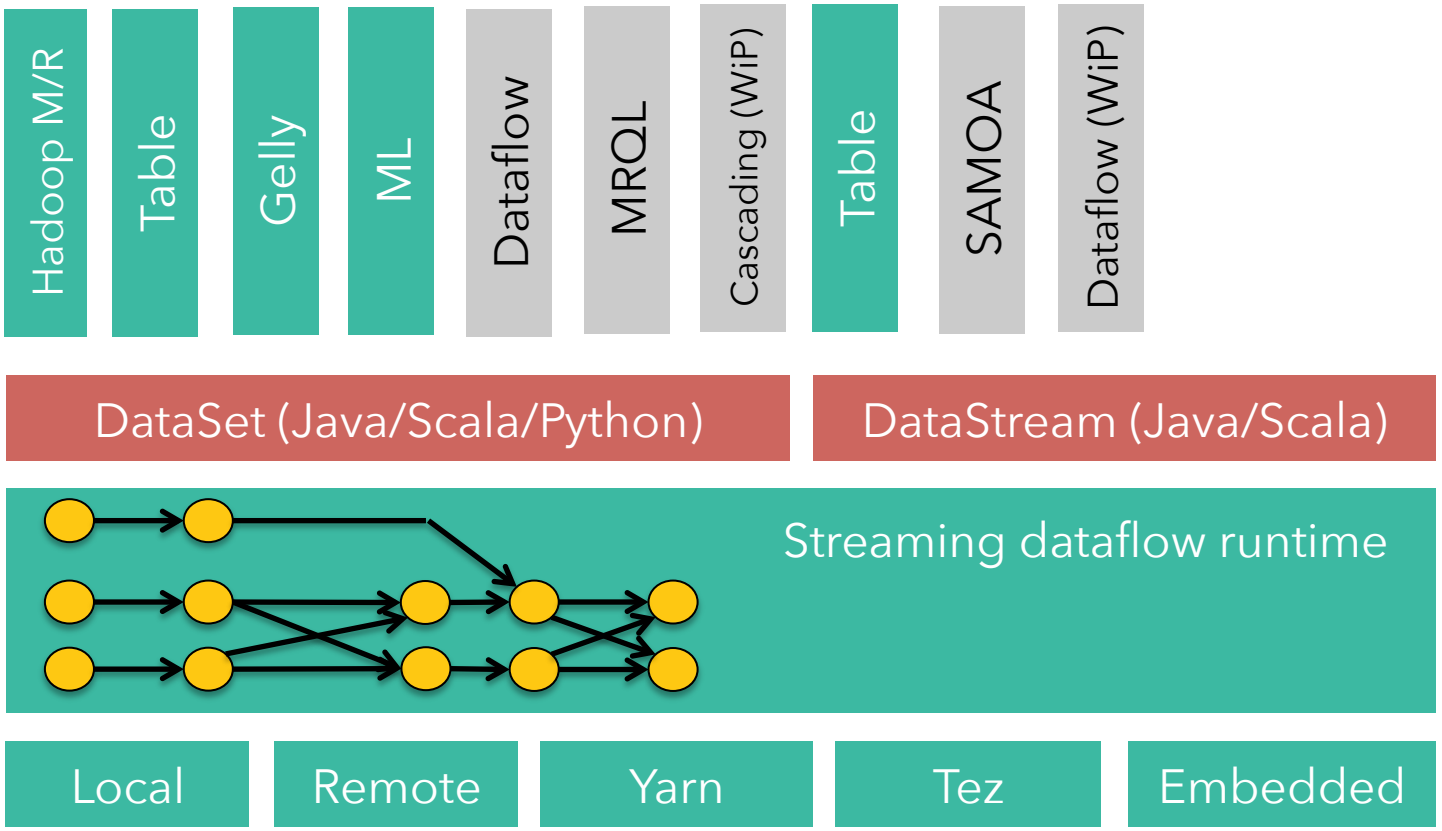


5. Special code paths for batch

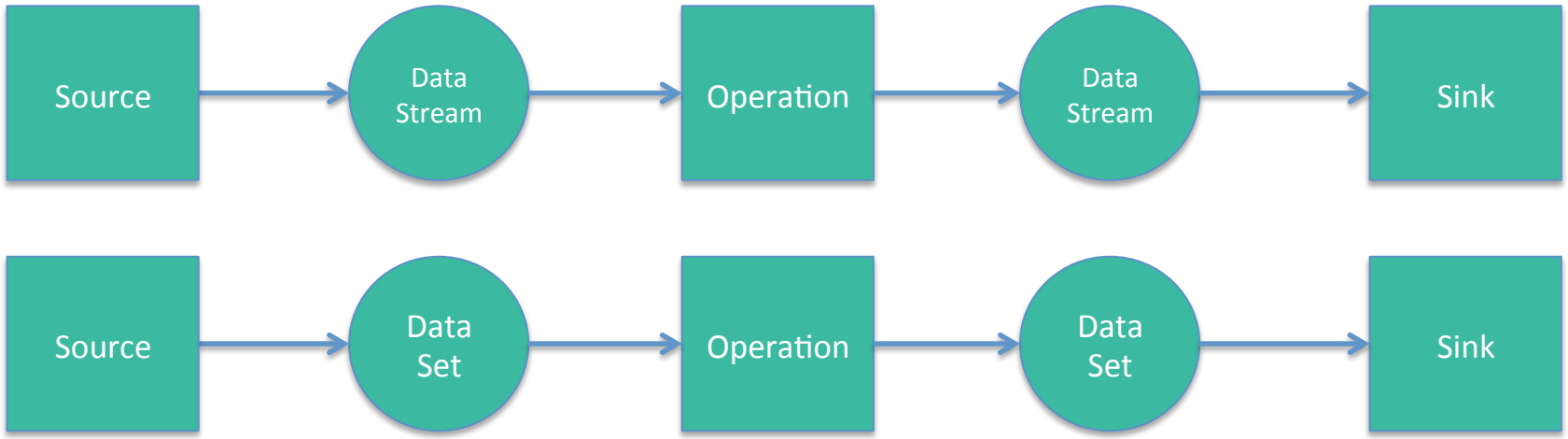


What is a Flink Program?

Flink stack


Hadoop M/R
Table
Gelly
ML
Dataflow
MRQL
Cascading (WiP)
Table
SAMOA
Dataflow (WiP)
DataSet (Java/Scala/Python)
DataStream (Java/Scala)
Streaming dataflow runtime
Local
Remote
Yarn
Tez
Embedded

Basic API Concept



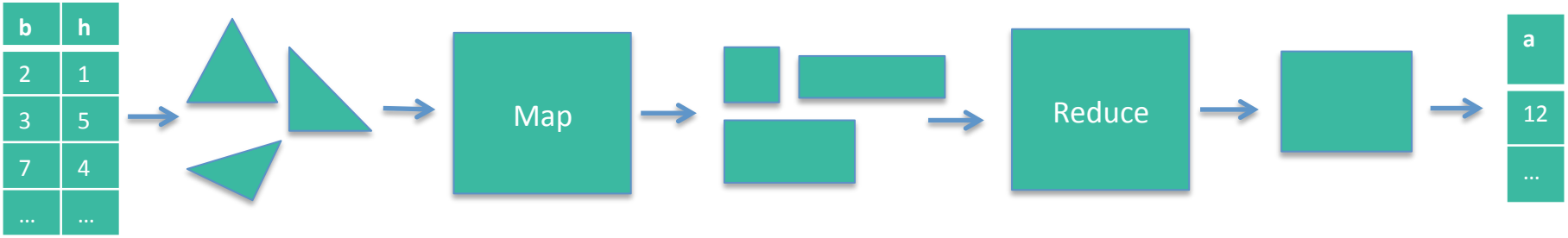
How do I write a Flink program?

1. Bootstrap sources
2. Apply operations
3. Output to source

Batch & Stream Processing

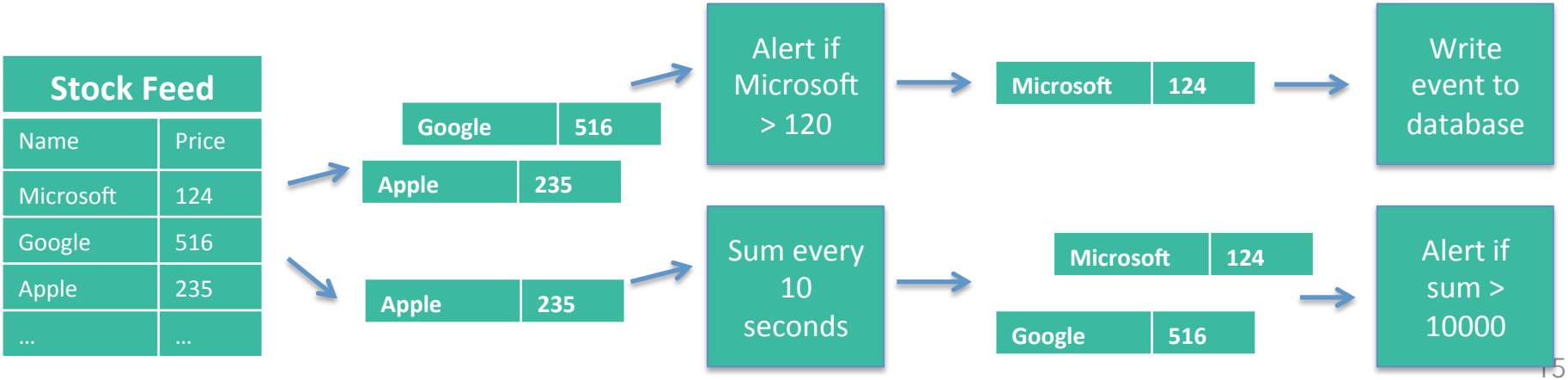
- DataSet API

  Example: Map/Reduce paradigm



- DataStream API

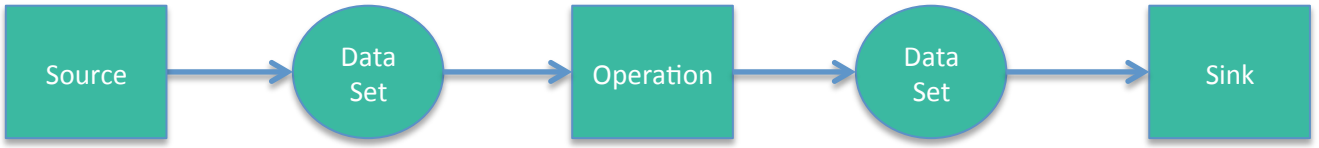
  Example: Live Stock Feed

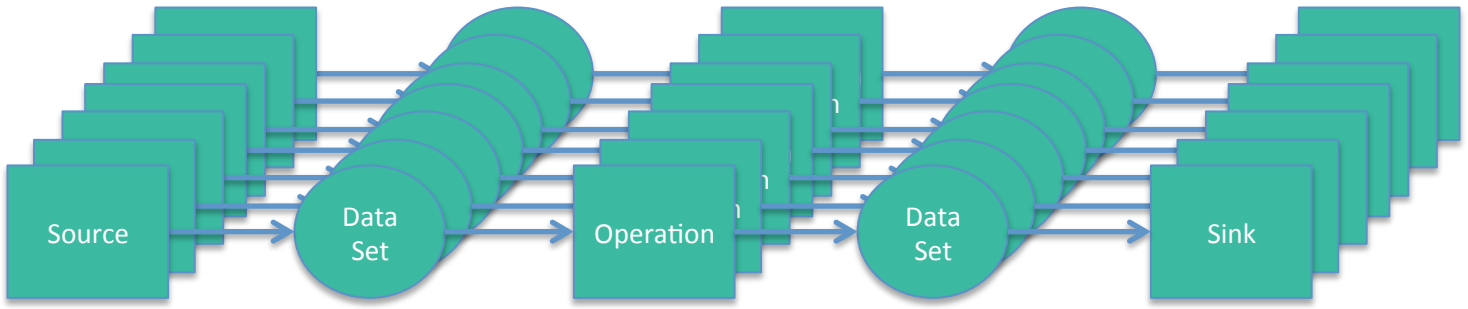


Streaming & Batch

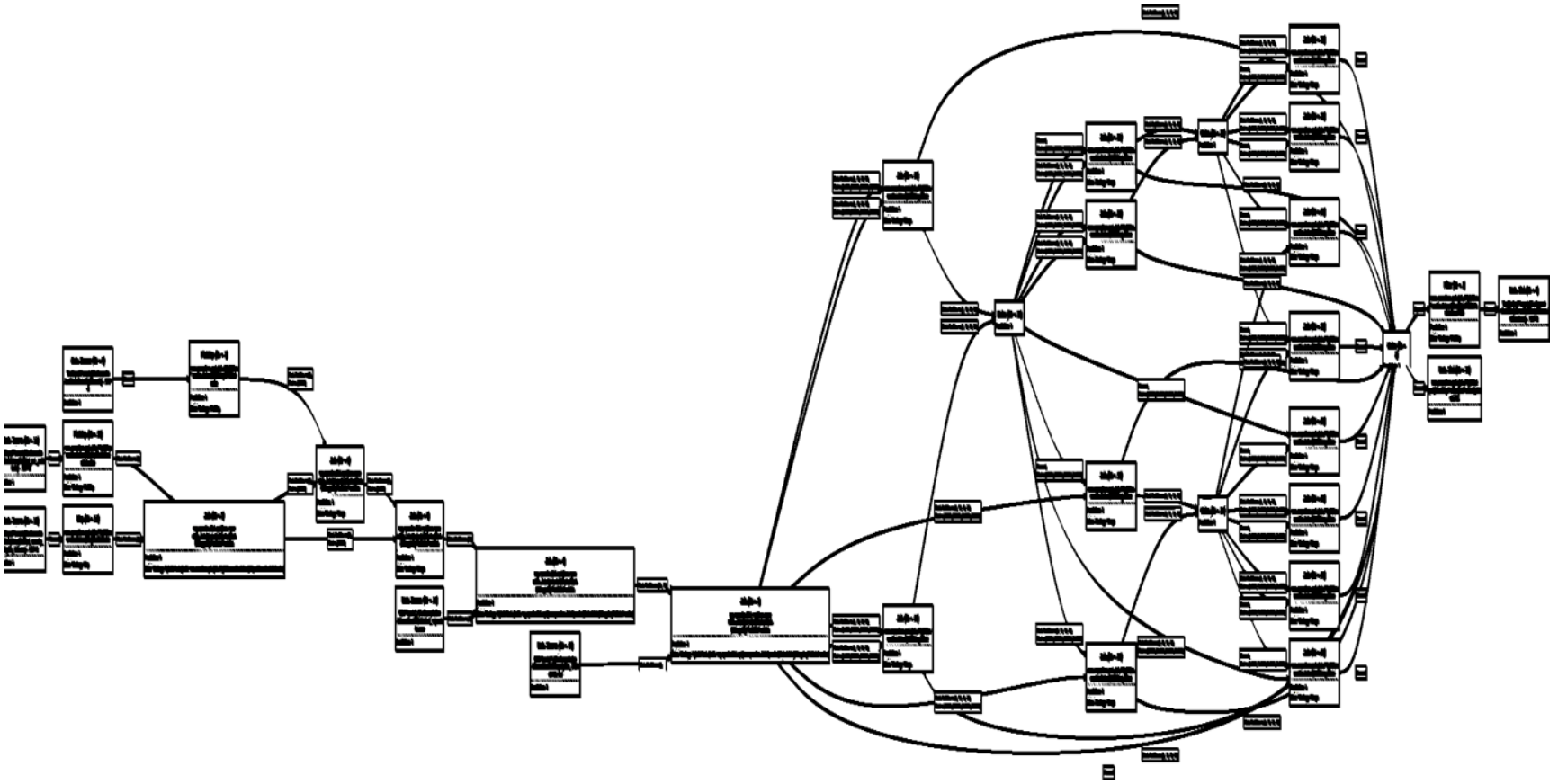
	Streaming	Batch
Input	infinite	finite
Data transfer	pipelined	blocking or pipelined
Latency	low	high

Scaling out


Source
Data Set
Operation
Data Set
Sink


Source
Data Set
Operation
Data Set
Sink

Scaling up



Sources (selection)

Collection-based

- fromCollection
- fromElements

File-based

- TextInputFormat
- CsvInputFormat

Other

- SocketInputFormat
- KafkaInputFormat
- Databases

Sinks (selection)

File-based

- TextOutputFormat
- CsvOutputFormat
- PrintOutput

Others

- SocketOutputFormat
- KafkaOutputFormat
- Databases

Hadoop Integration

Out of the box

- Access HDFS
- Yarn Execution (covered later)
- Reuse data types (Writables)

With a thin wrapper

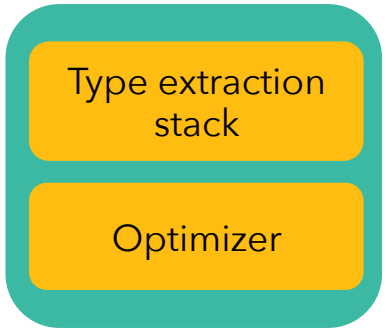
- Reuse Hadoop input and output formats
- Reuse functions like Map and Reduce

What's the Lifecycle of a Program?

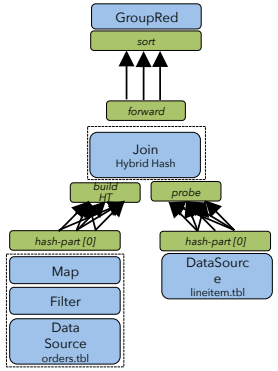
From Program to Dataflow

```
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
  }
```

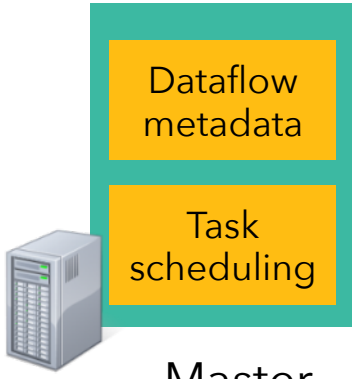
Program



Pre-flight (Client)



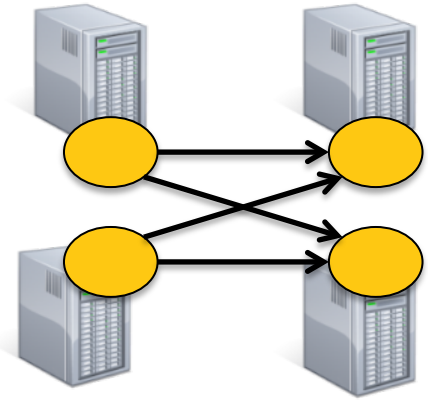
Dataflow Graph



Master

deploy operators

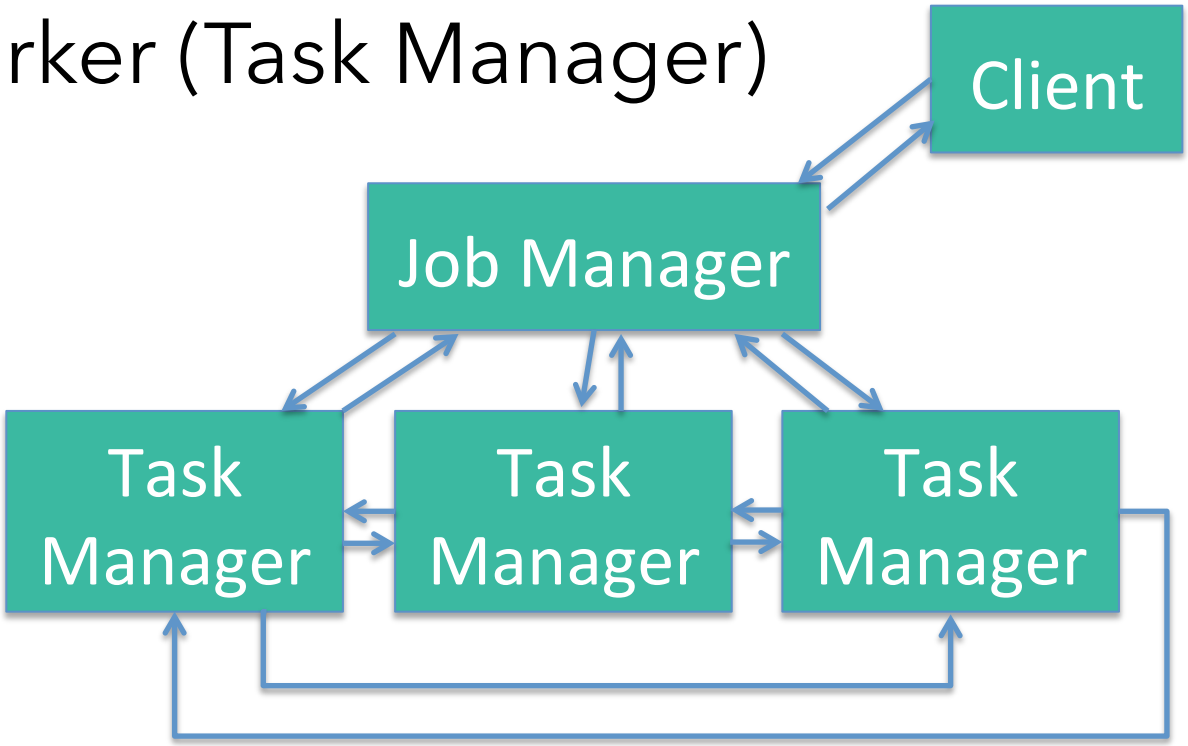
track intermediate results



Workers

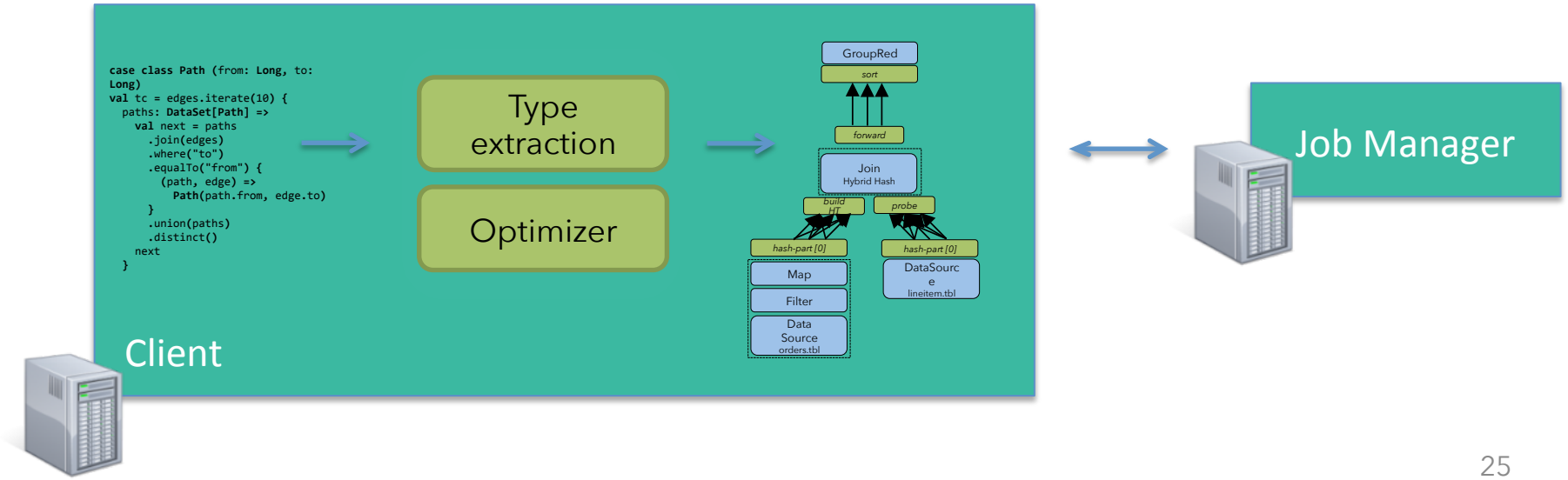
Architecture Overview

- Client
- Master (Job Manager)
- Worker (Task Manager)



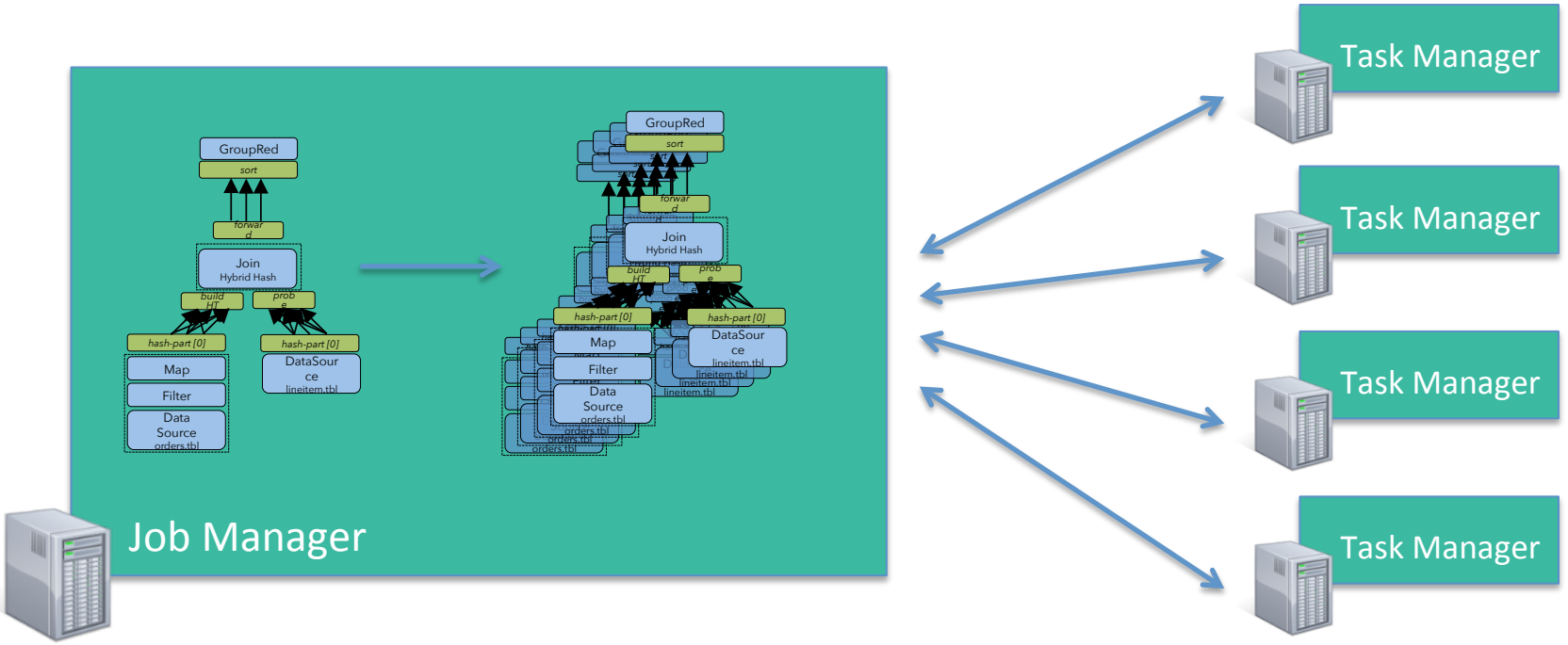
Client

- Optimize
- Construct job graph
- Pass job graph to job manager
- Retrieve job results



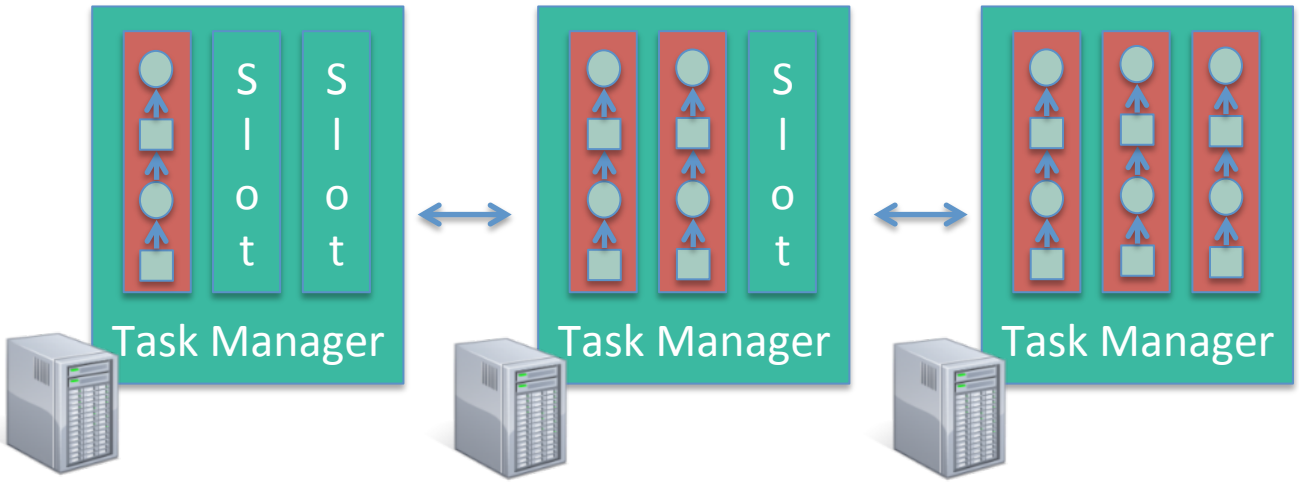
Job Manager

- **Parallelization**: Create Execution Graph
- **Scheduling**: Assign tasks to task managers
- **State**: Supervise the execution



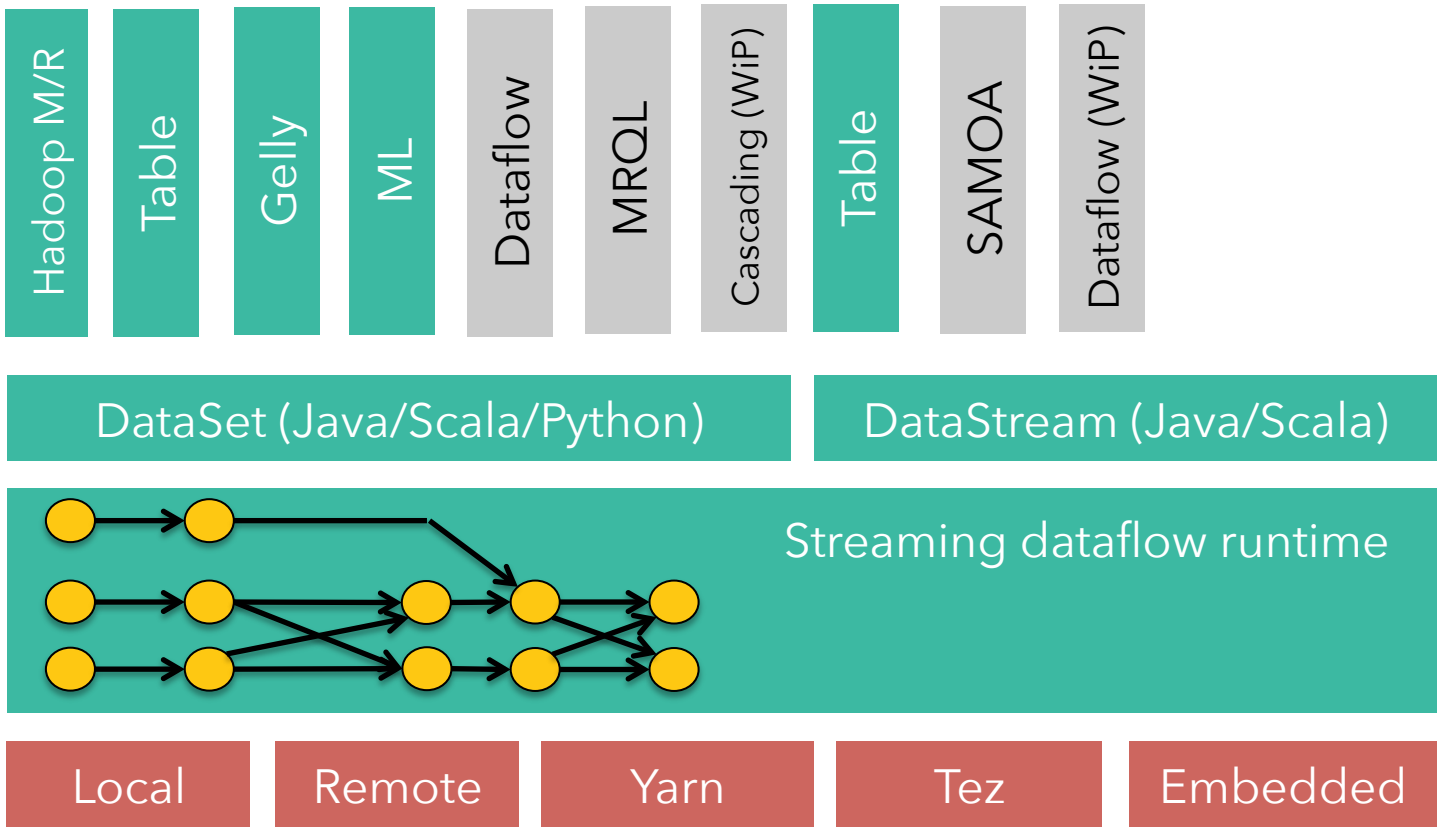
Task Manager

- Operations are split up into **tasks** depending on the specified parallelism
- Each parallel instance of an operation runs in a separate **task slot**
- The scheduler may run several tasks from different operators in one task slot



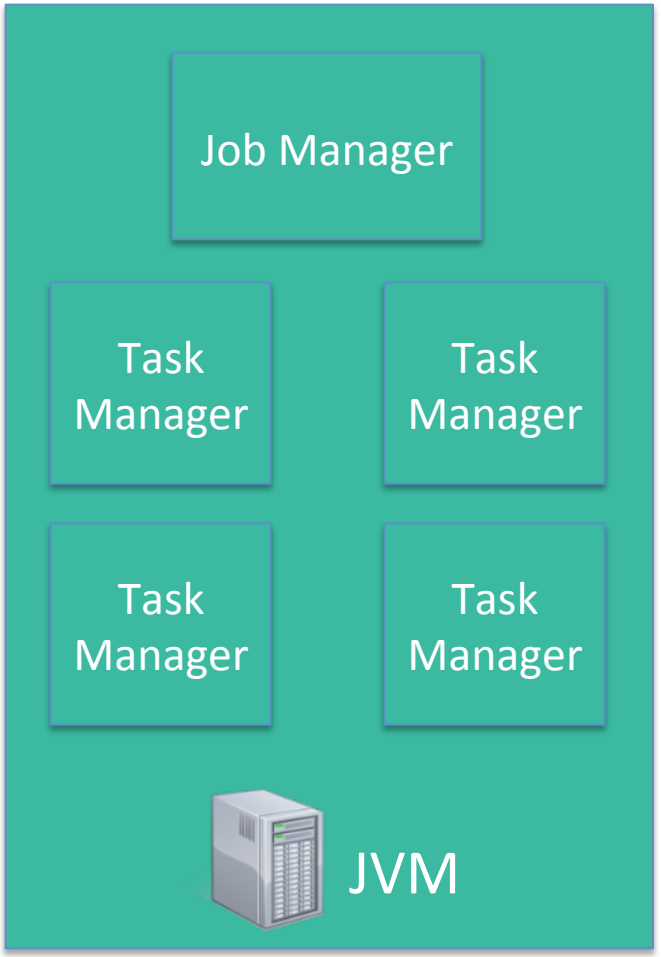
Execution Setups

Ways to Run a Flink Program



Local Execution

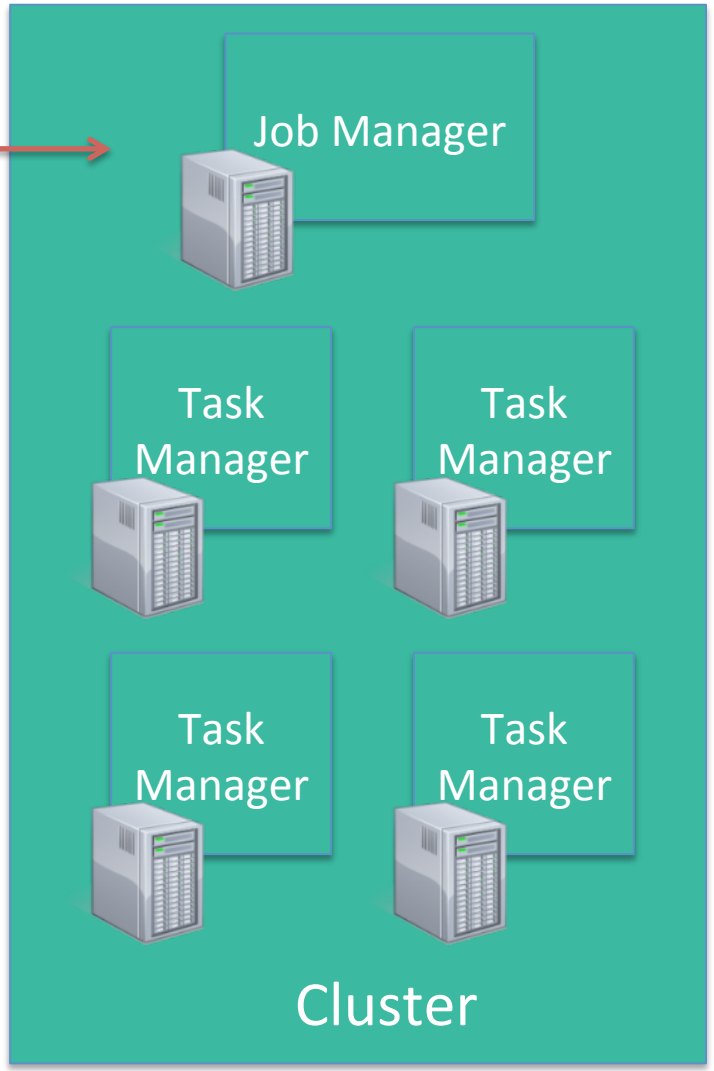
- Starts local Flink cluster
- All processes run in the same JVM
- Behaves just like a regular Cluster
- Very useful for developing and debugging



Embedded Execution

- Runs operators on simple **Java collections**
- Lower overhead
- Does not use memory management
- Useful for testing and debugging

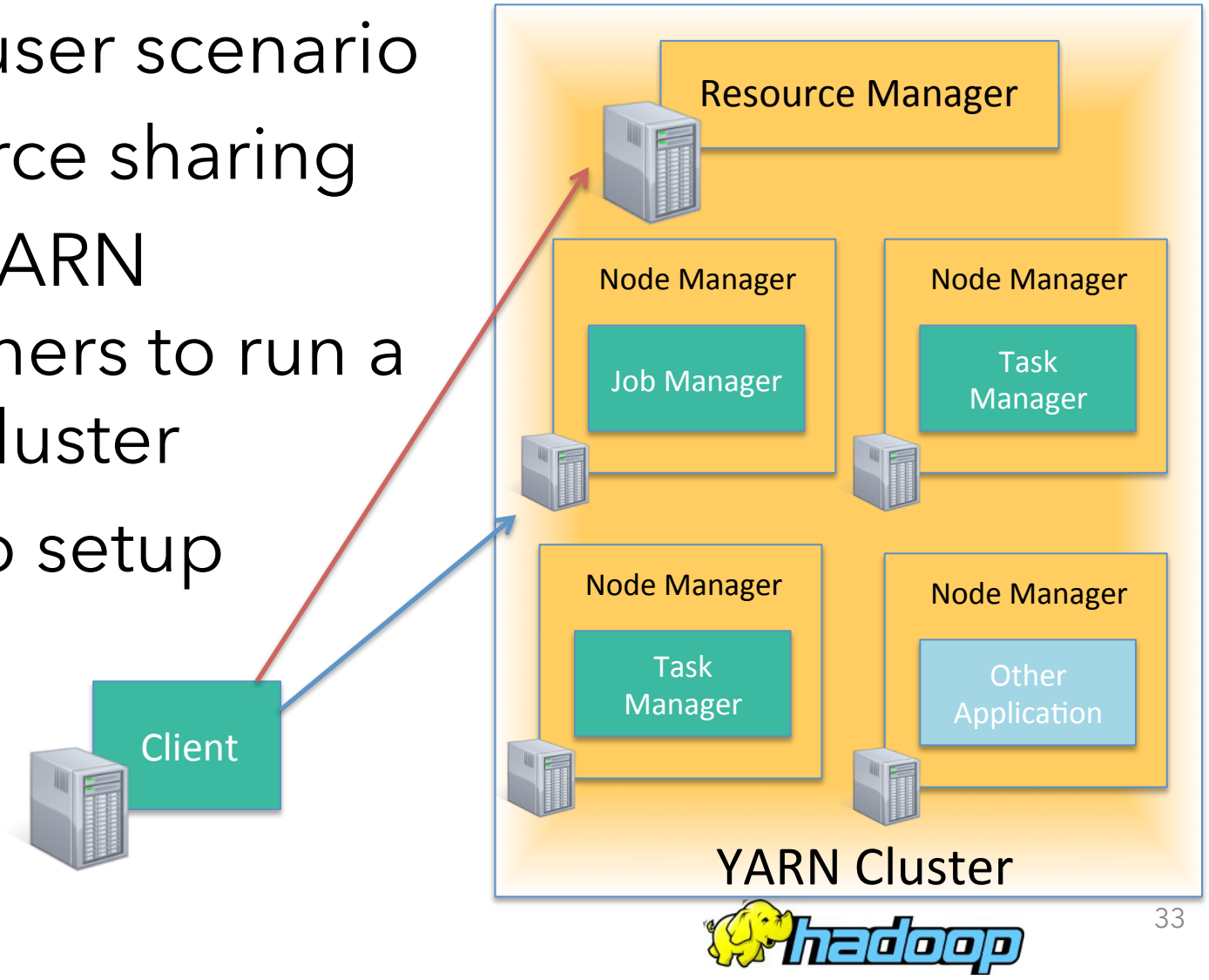
Remote Execution



- **Submit** a Job remotely
- **Monitor** the status of a job

YARN Execution

- Multi-user scenario
- Resource sharing
- Uses YARN containers to run a Flink cluster
- Easy to setup



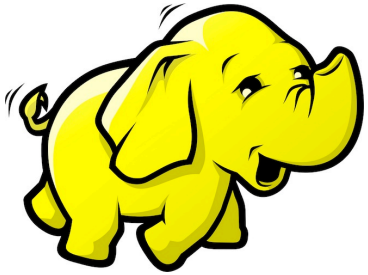
Execution

- Leverages Apache Tez's runtime
- Built on top of YARN
- Good YARN citizen
- Fast path to elastic deployments
- Slower than native Flink

Flink compared to other projects

Batch & Streaming projects

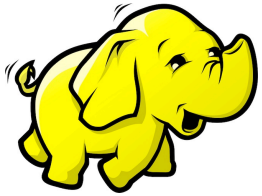
Batch only



Streaming only

Hybrid

Batch comparison



API	low-level	high-level	high-level
Data Transfer	batch	batch	pipelined & batch
Memory Management	disk-based	JVM-managed	Active managed
Iterations	file system cached	in-memory cached	streamed
Fault tolerance	task level	task level	job level
Good at	massive scale out	data exploration	heavy backend & iterative jobs
Libraries	many external	built-in & external	evolving built-in & external

Streaming comparison

Streaming	“true”	mini batches	“true”
API	low-level	high-level	high-level
Fault tolerance	tuple-level ACKs	RDD-based (lineage)	coarse checkpointing
State	not built-in	external	internal
Exactly once	at least once	exactly once	exactly once
Windowing	not built-in	restricted	flexible
Latency	low	medium	low
Throughput	medium	high	high

Thank you for listening!