dataArtisans

Apache Flink® Training

System Overview

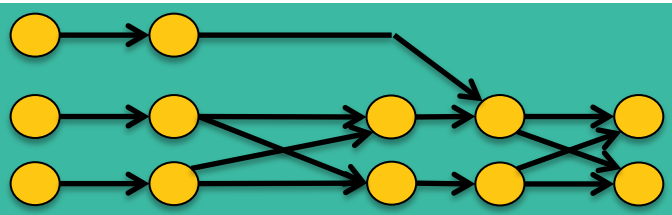
June 15th, 2015

What is Apache Flink?

A Top-Level project of the Apache Software Foundation

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

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



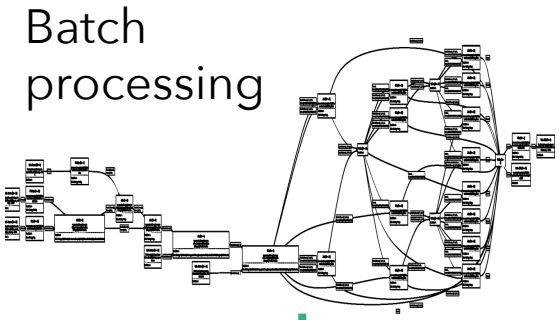
Streaming dataflow runtime

Local | Remote | Yarn | Tez | Embedded

What is Apache Flink?

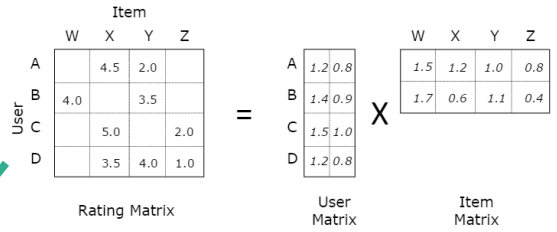
- Large-scale data processing engine
- Easy and powerful APIs for *batch and streaming* analysis (Java / Scala / Python)
- Backed by a robust execution backend
  - with true streaming capabilities,
  - sophisticated windowing mechanisms,
  - custom memory manager,
  - native iteration execution,
  - and a cost-based optimizer.

Native workload support

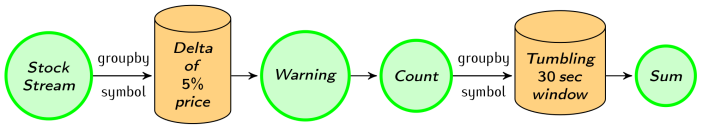


Batch processing

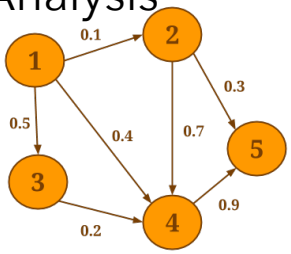
Machine Learning at scale



Stream processing



Graph Analysis



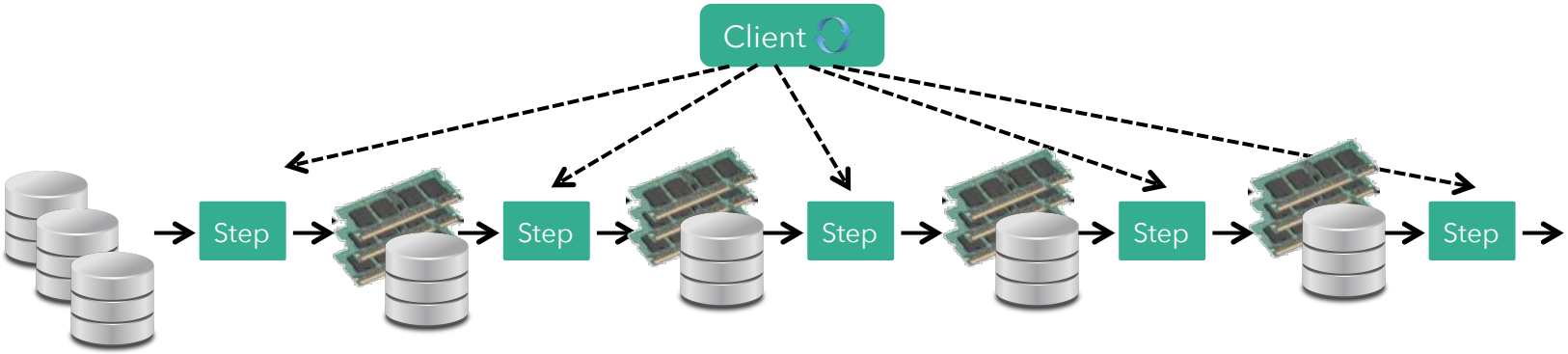
Flink

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

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

E.g.: Non-native iterations

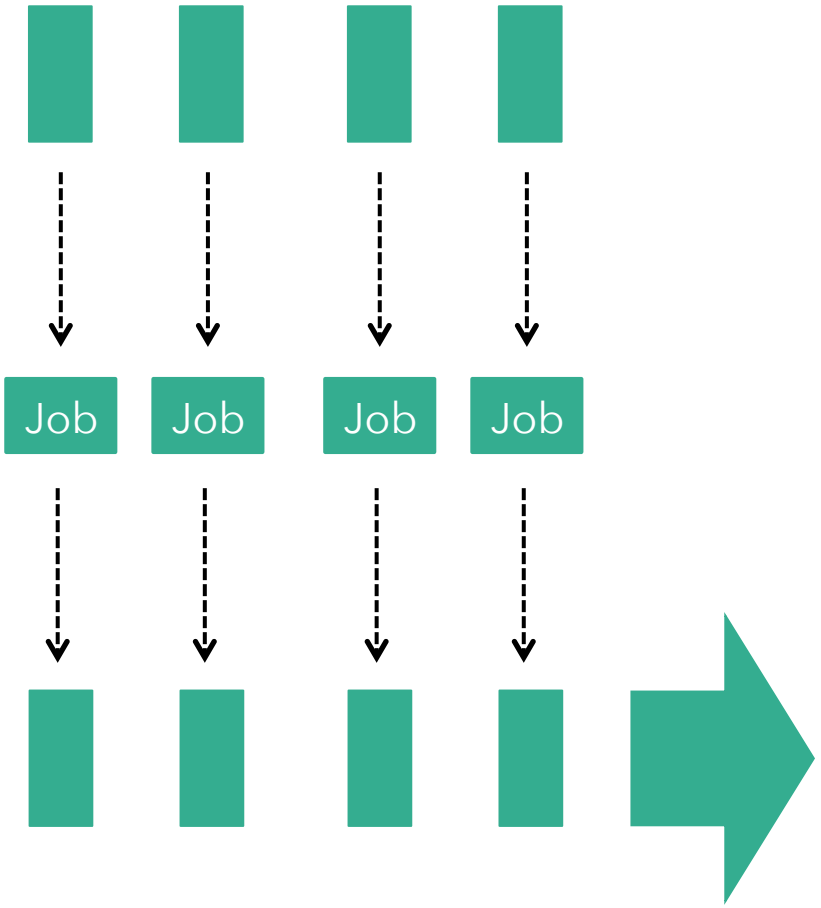
```
for (int i = 0; i < maxIterations; i++) {
    // Execute MapReduce job
}
```



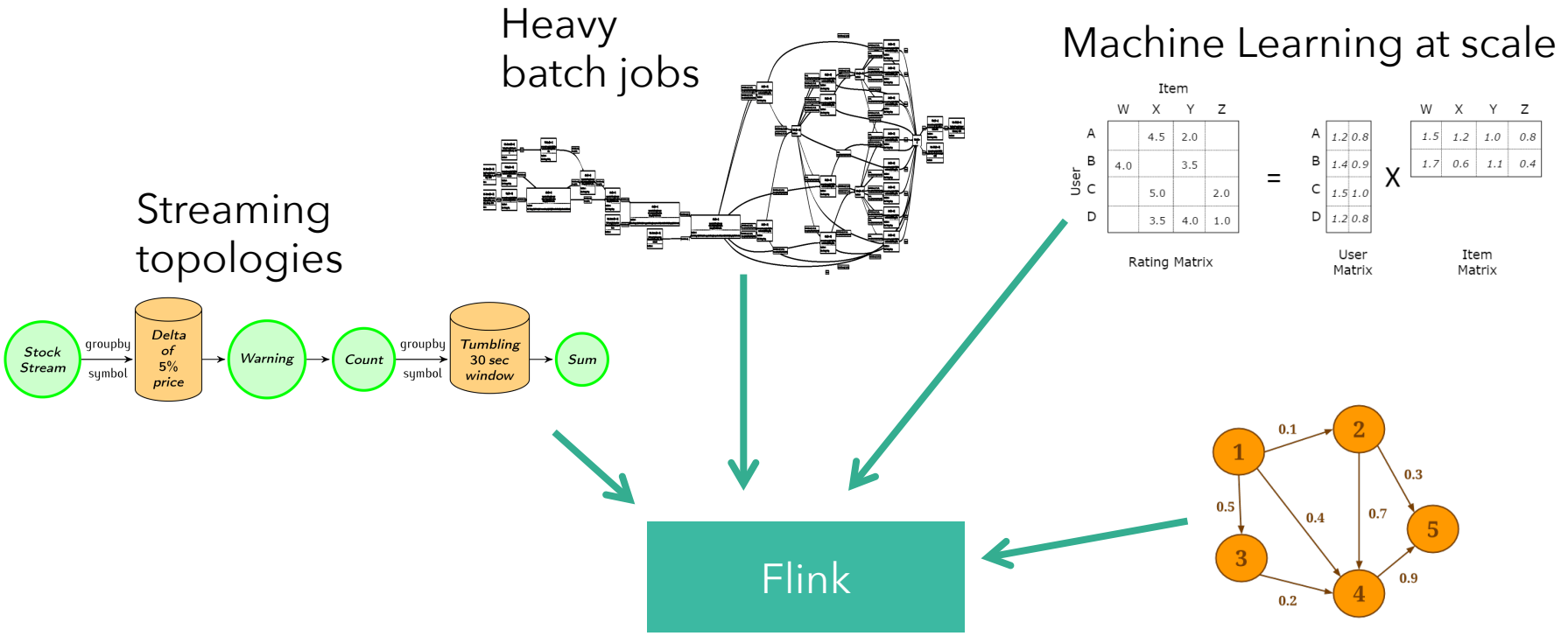
E.g.: Non-native streaming

stream discretizer

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



Native workload support



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

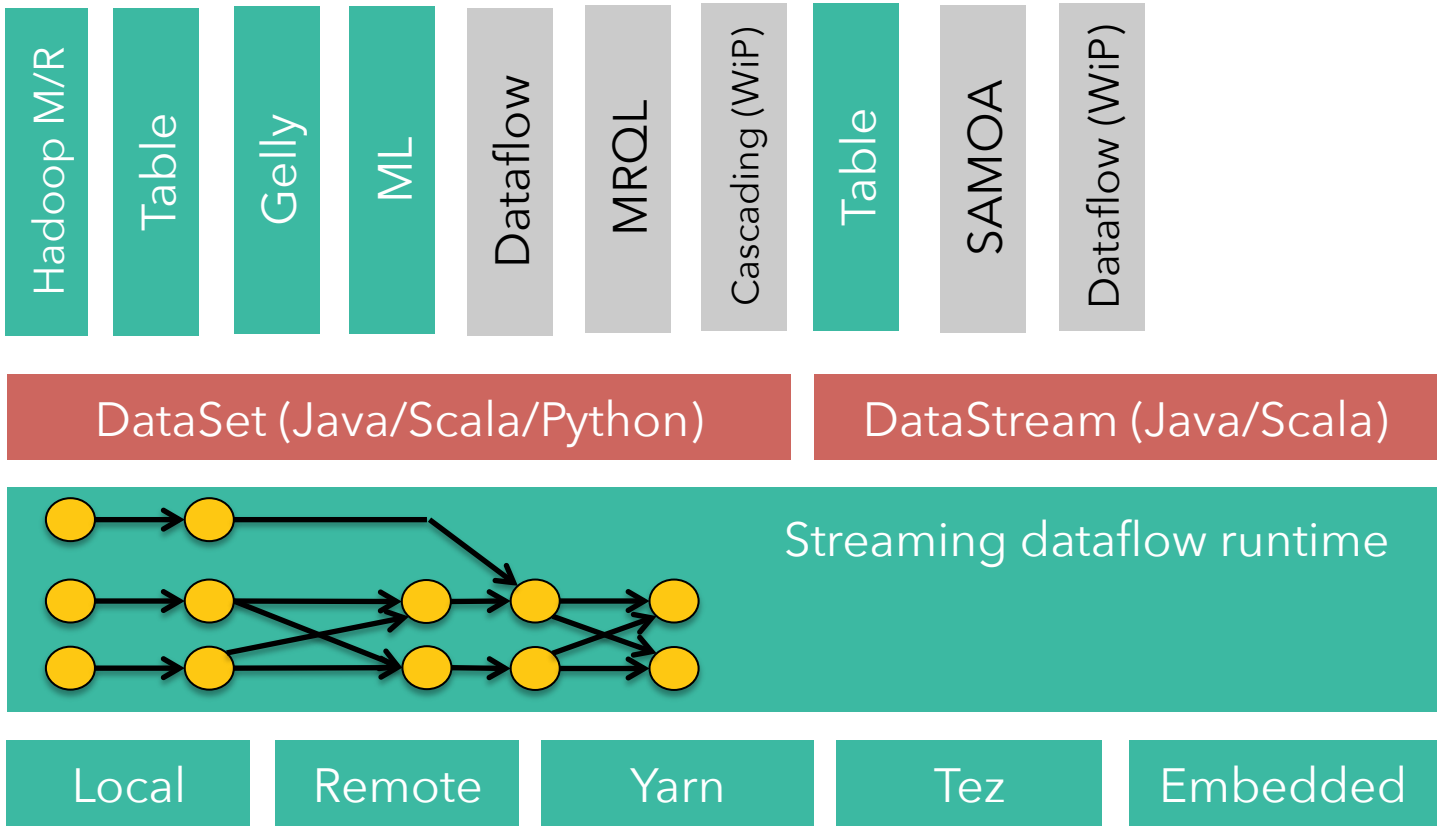
And what does native **mean**?

Flink Engine

1. Execute everything as streams
2. Allow some iterative (cyclic) dataflows
3. Allow some mutable state
4. Operate on managed memory
5. Special code paths for batch

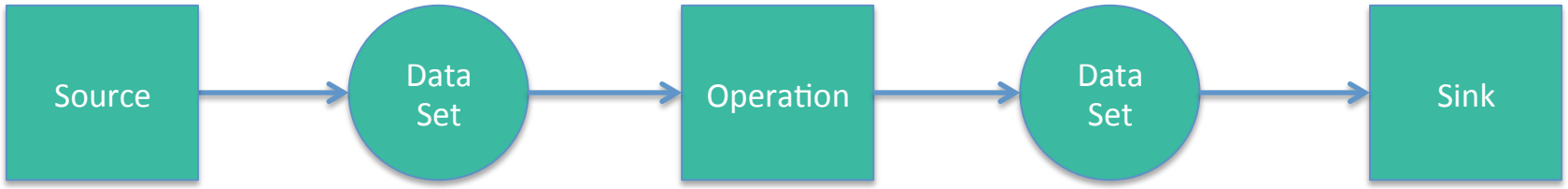
What is a Flink Program?

Flink stack

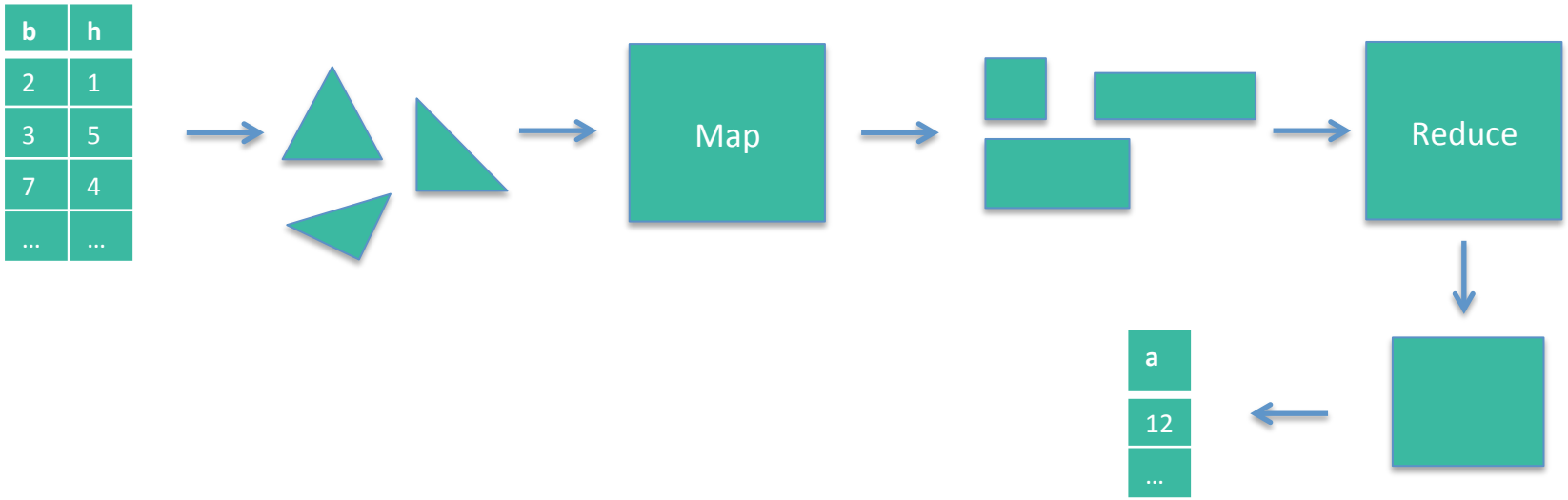

Hadoop M/R
Table
Gelly
ML
Dataflow
MRQL
Cascading (WiP)
Table
SAMOA
Dataflow (WiP)
DataSet (Java/Scala/Python)
DataStream (Java/Scala)
Streaming dataflow runtime
Local
Remote
Yarn
Tez
Embedded

DataSet

- Used for Batch Processing

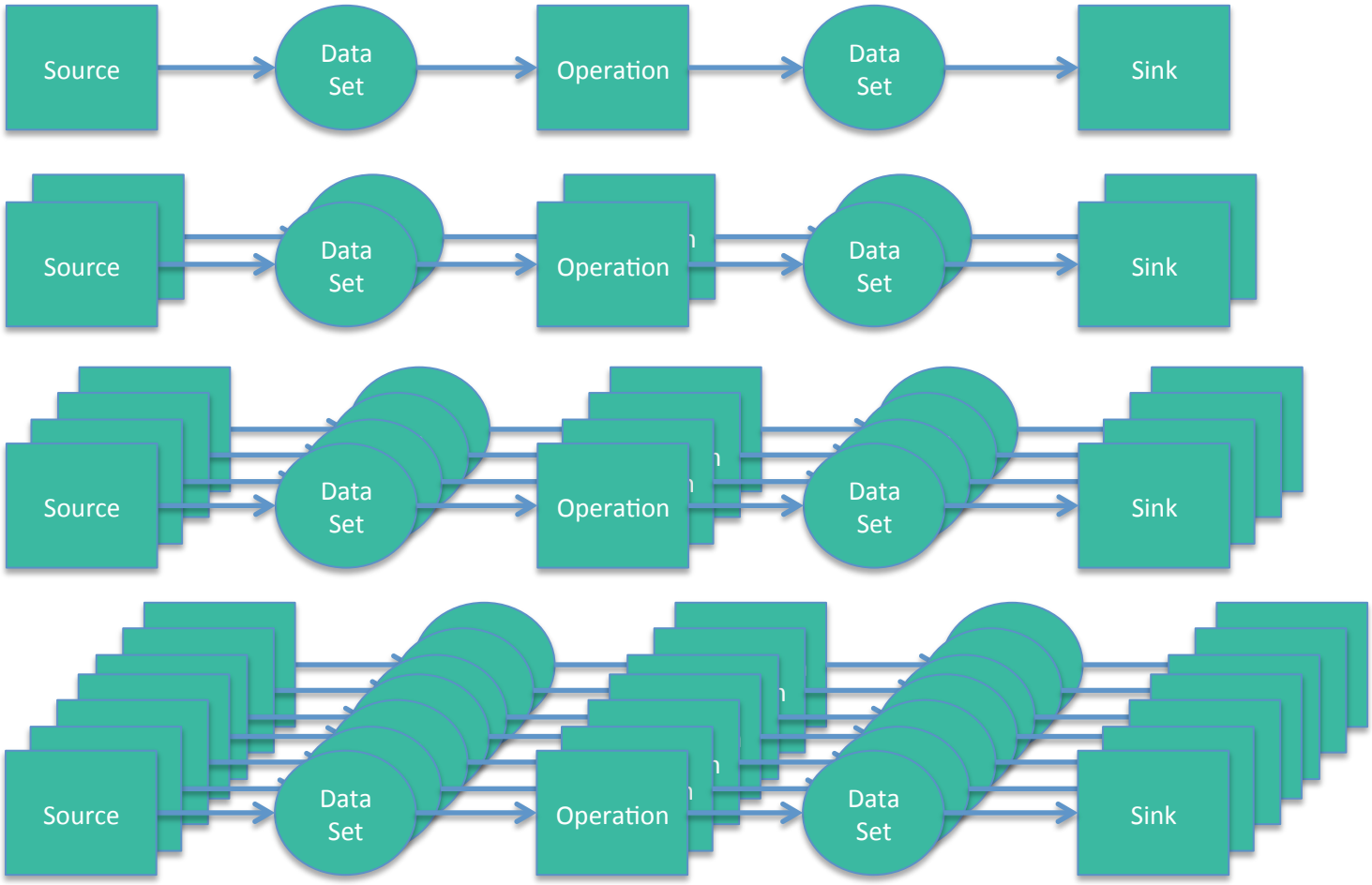


Example: Map and Reduce operation

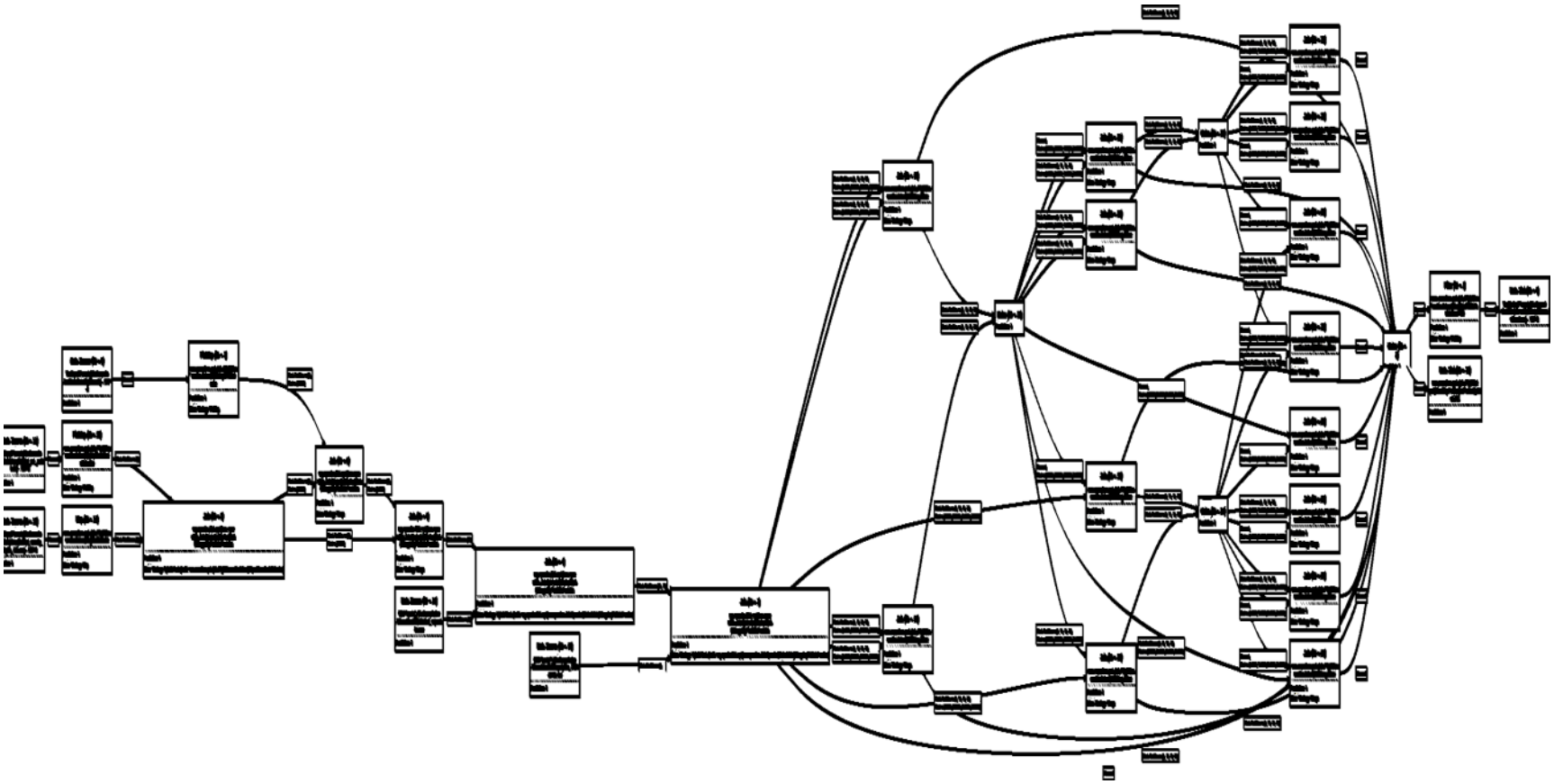


Scaling out

- Scale out arbitrarily by setting the parallelism

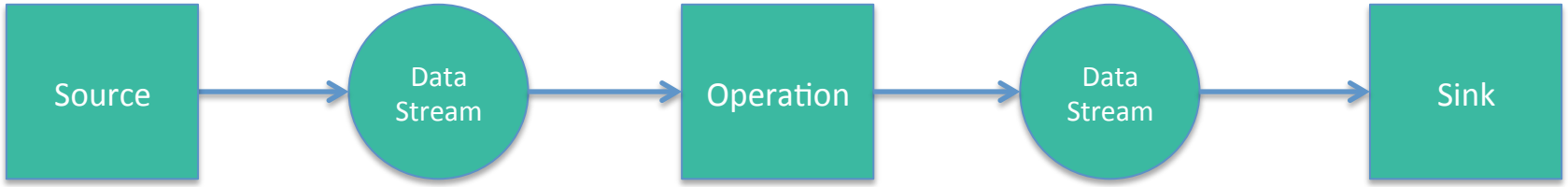


Scaling up

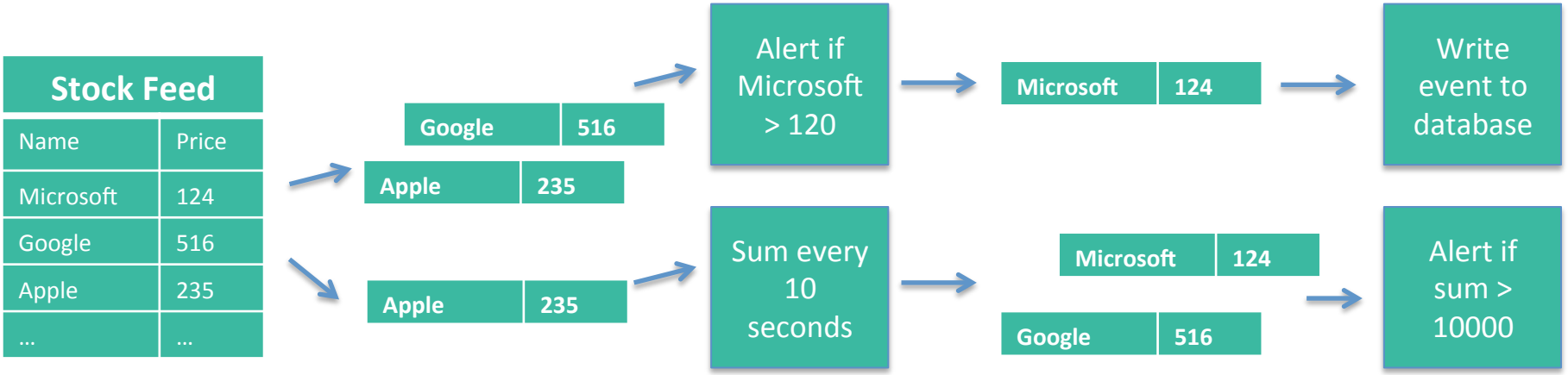


DataStream

- Real-time event streams



Example: Stream from a live stock feed



Sources (selection)

File-based

- TextInputFormat
- CsvInputFormat

Collection-based

- fromCollection
- fromElements

Sinks (selection)

File-based

- TextOuputFormat
- CsvOutputFormat
- PrintOutput

Hadoop Integration

Out of the box

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

With a thin wrapper

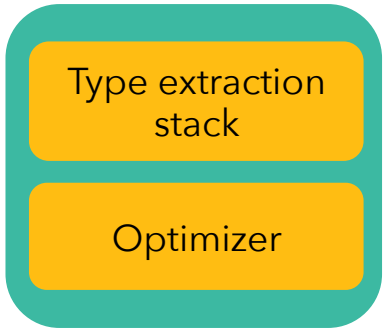
- Reuse Hadoop input and output formats
- Reuse functions like Map and Reduce

What's the Lifecycle of a Program?

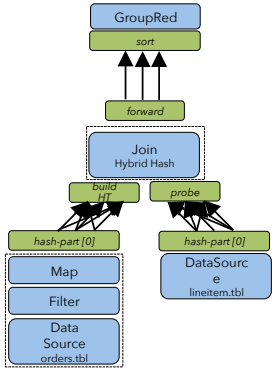
From Program to Dataflow

```
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
  }
```

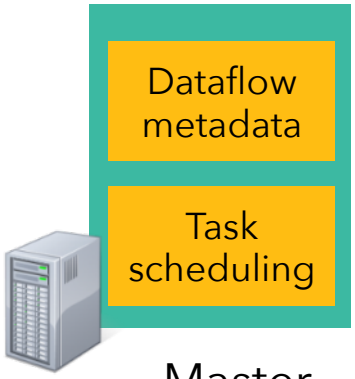
Program



Pre-flight (Client)



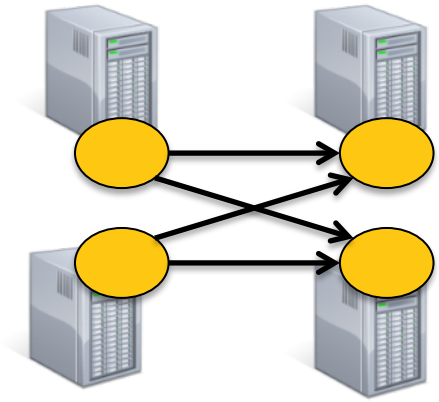
Dataflow Graph



Master

deploy operators

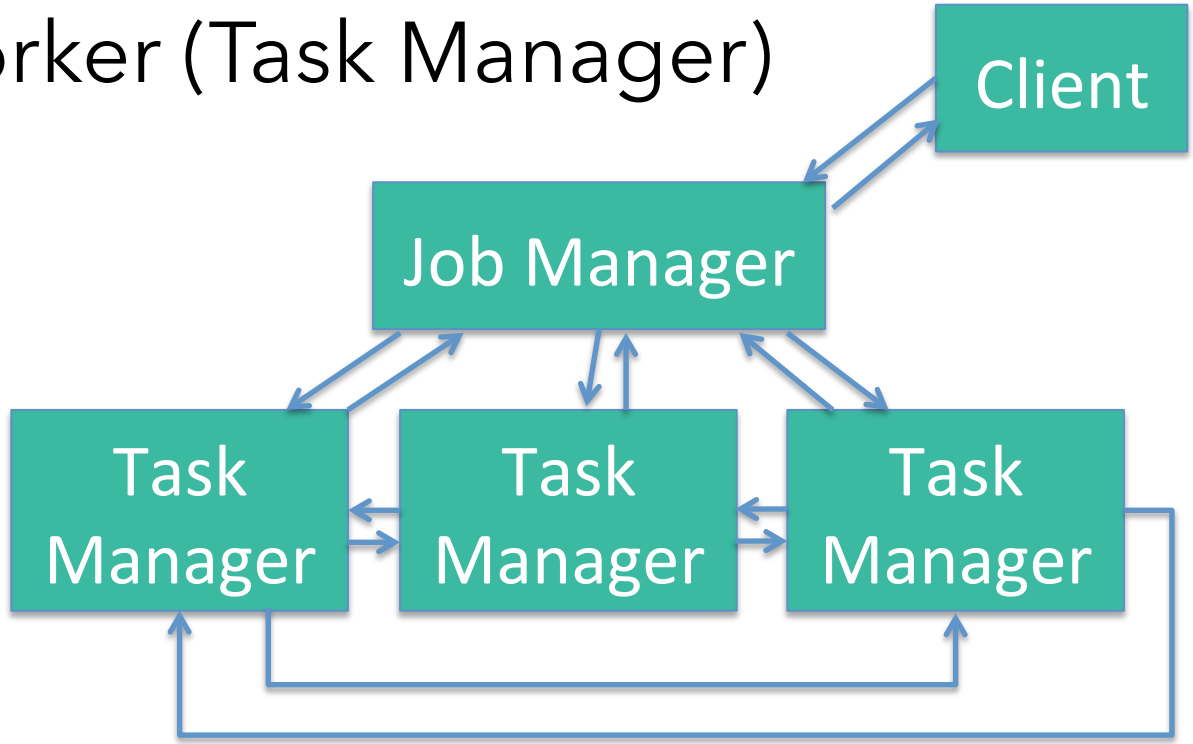
track intermediate results



Workers

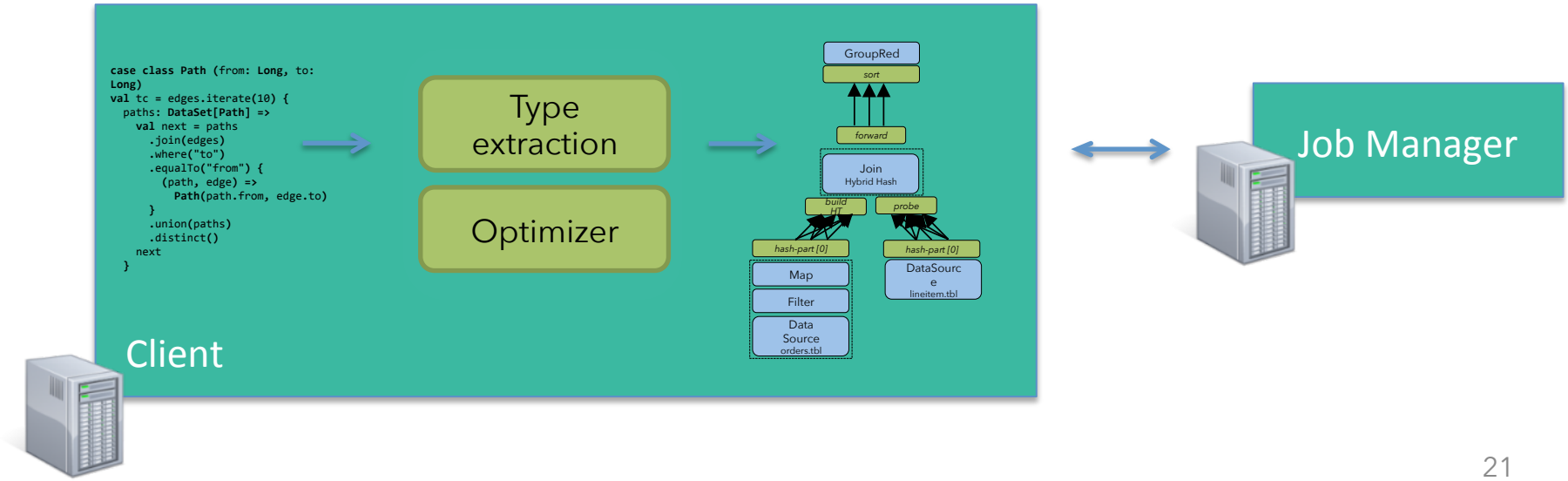
Architecture Overview

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



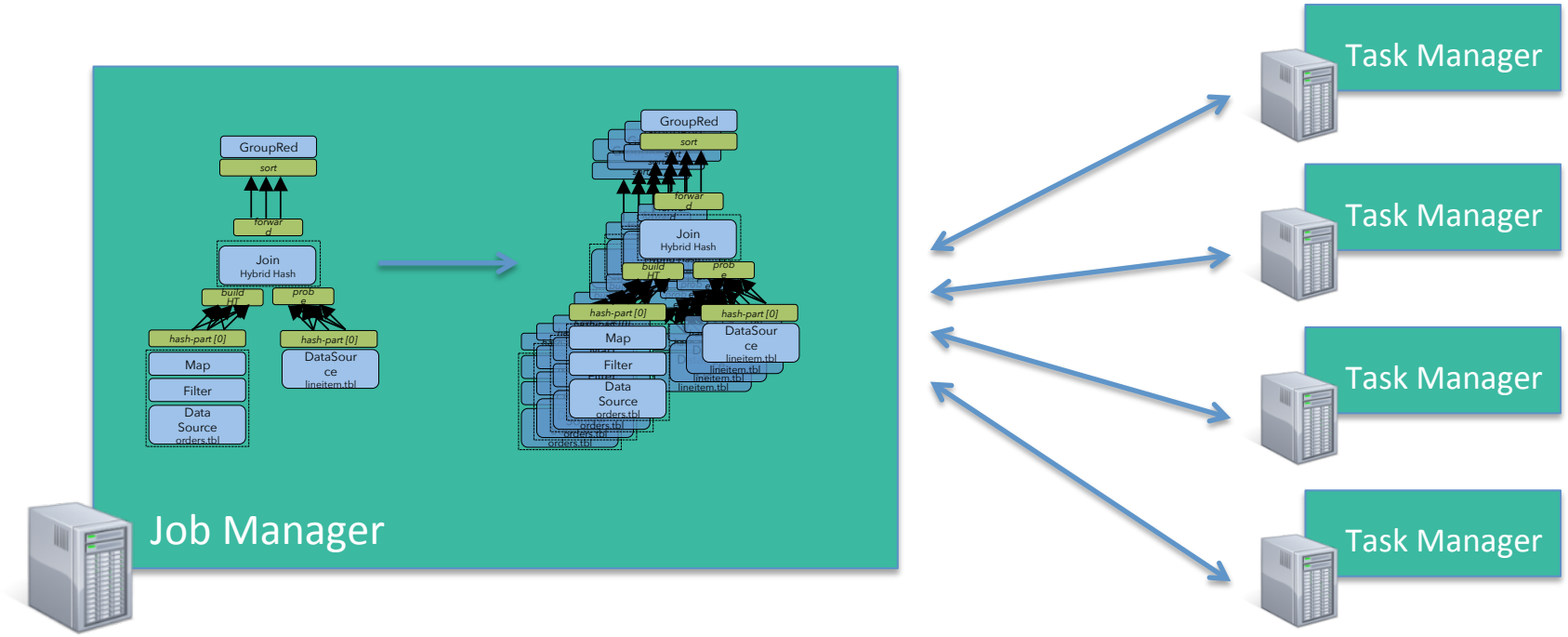
Client

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



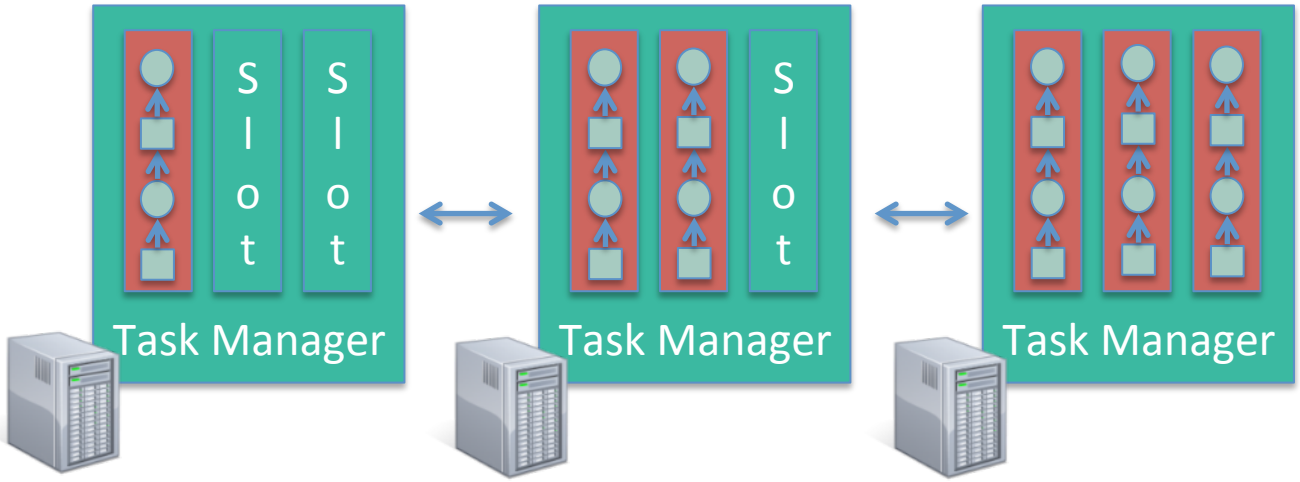
Job Manager

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



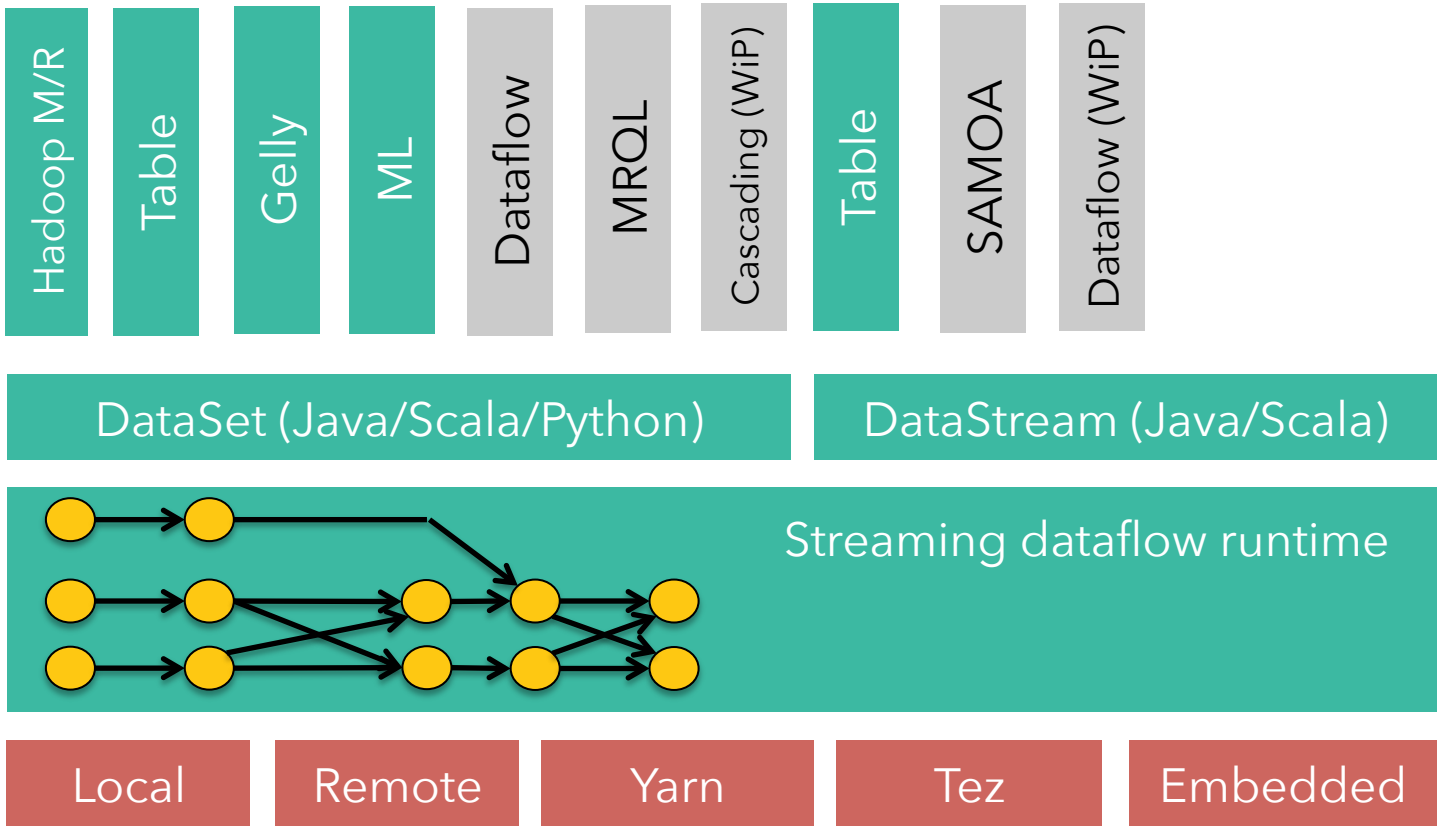
Task Manager

- 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



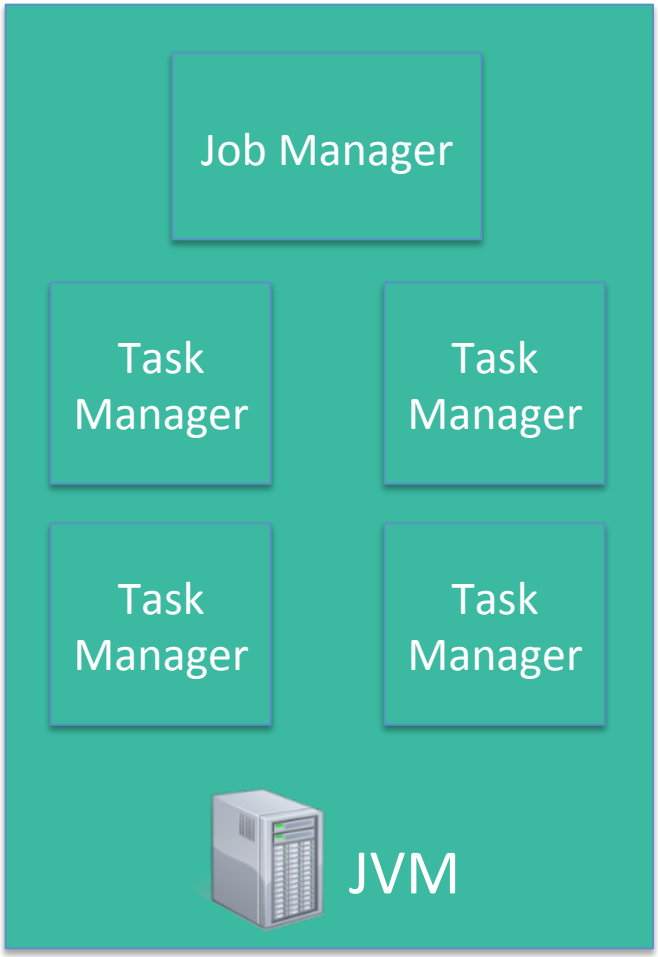
Execution Setups

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

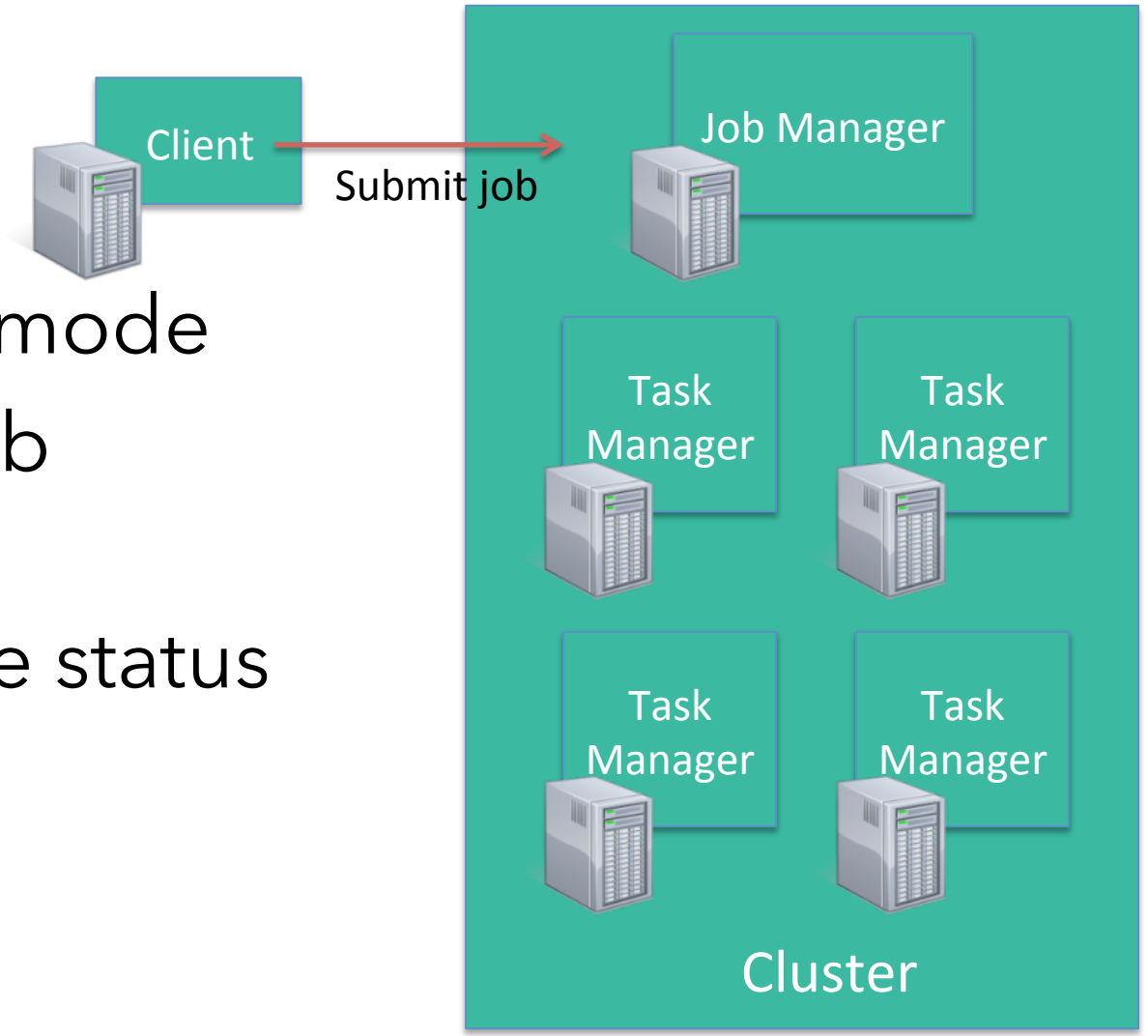


Embedded Execution

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

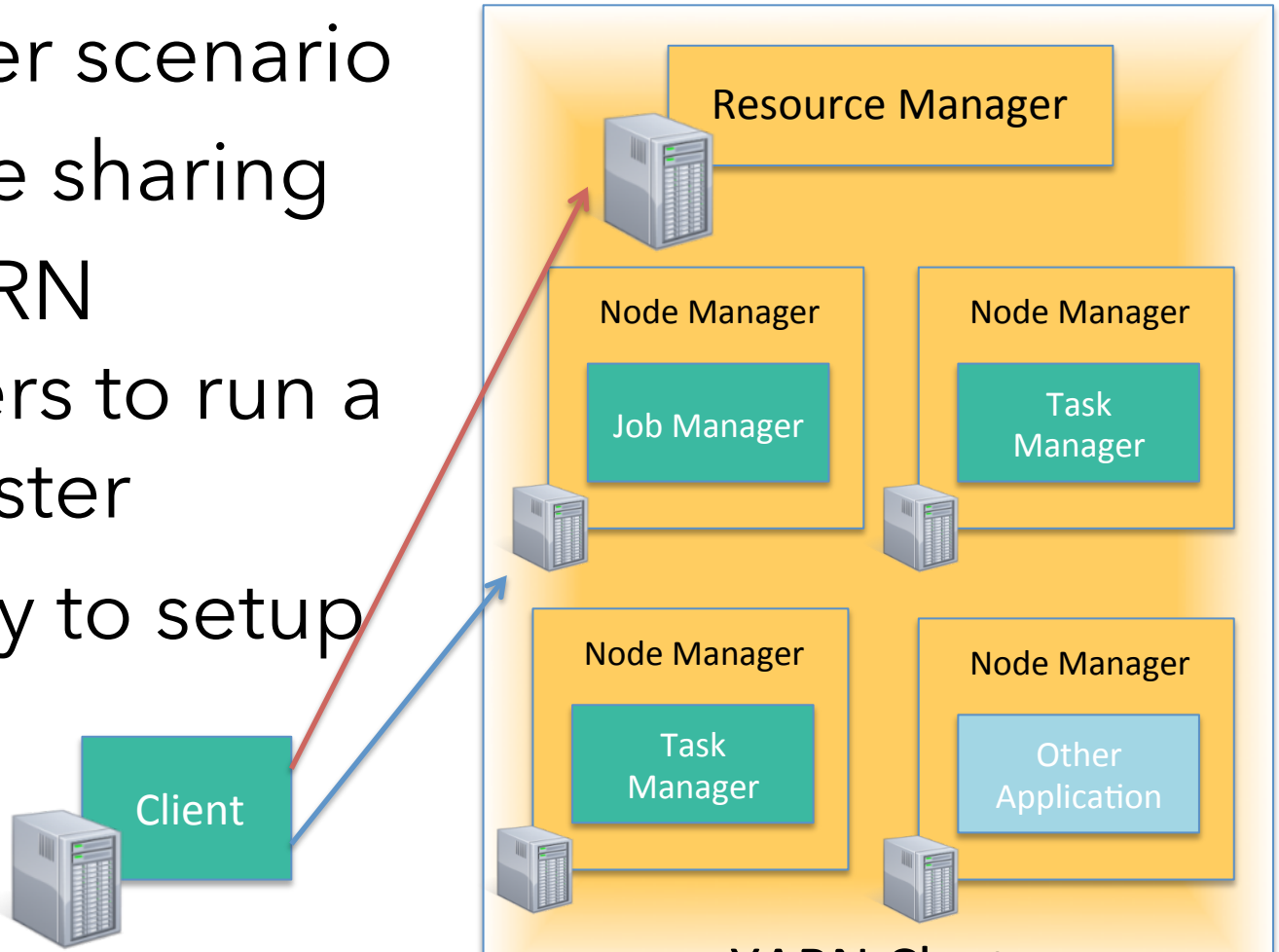
Remote Execution

- The cluster mode
- Submit a Job remotely
- Monitors the status of the job



YARN Execution

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



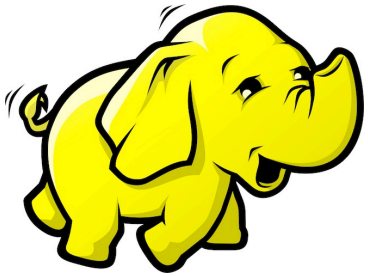
Execution

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

Flink compared to other projects

Batch & Streaming projects

Batch only

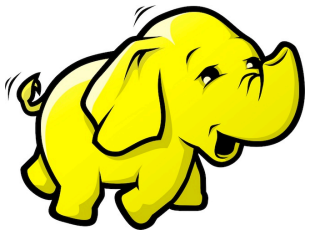


Streaming only

Hybrid

Batch

- Apache Hadoop Map Reduce
  - Low-Level API
  - Batch data transfer
  - Mainly disk based operations
  - File system cached iterations
  - Massive scale out parallelism
  - External libraries
  - Task level fault tolerance



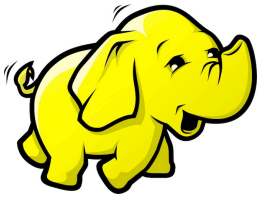
- Apache Spark
  - High-level API
  - Batch data transfer
  - JVM managed memory
  - In memory cached iterations
  - Interactive Data exploration
  - Libraries
  - Task level fault tolerance

- Apache Flink
  - High-level API
  - Pipelined and batch data transfer
  - Active memory management
  - Natively streamed iterations
  - Heavy load backend jobs, iterative data flows
  - Evolving libraries
  - Job level fault tolerance

Streaming

- Apache Storm
  - “True” streaming
  - Low-Level API
  - Costly fault tolerance
  - No built-in state handling
  - At least once semantics
  - No built-in windowing
  - Low Latency
  - Medium throughput
- Apache Spark
  - Mini batch streaming
  - High-level API
  - RDD-based fault tolerance (lineage)
  - External state handling
  - Exactly once semantics
  - Restricted windowing
  - Medium latency
  - High throughput
- Apache Flink
  - “True” streaming
  - High-level API
  - Checkpointing
  - Built-in operator state handling
  - Exactly once semantics
  - Flexible windowing
  - Low latency
  - High throughput

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!