Flinq

Aljoscha Krettek

Flink committer

co-founder @ *data Artisans*

aljoscha@apache.org

What is Flink

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

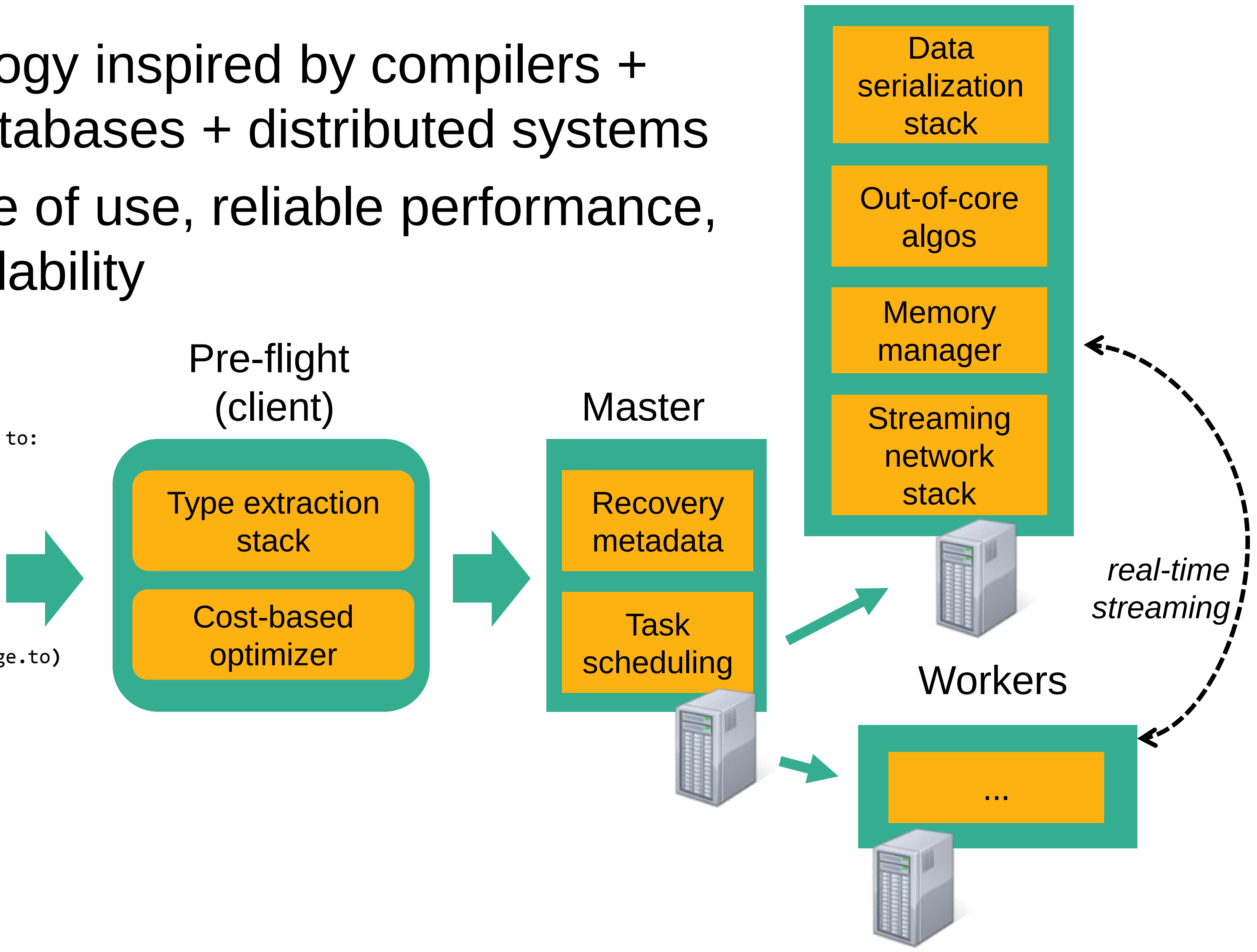
The case for Apache Flink

- Performance and ease of use
  - Exploits in-memory and pipelining, language-embedded logical APIs
- Unified batch and real streaming
  - Batch and Stream APIs on top of a streaming engine
- A runtime that "just works" without tuning
  - custom memory management inside the JVM
- Predictable and dependable execution
  - Bird’s-eye view of what runs and how, and what failed and why

Technology inside Flink

- Technology inspired by compilers + MPP databases + distributed systems
- For ease of use, reliable performance, and scalability

```
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
  }
```



How do you use Flink?

Example: WordCount

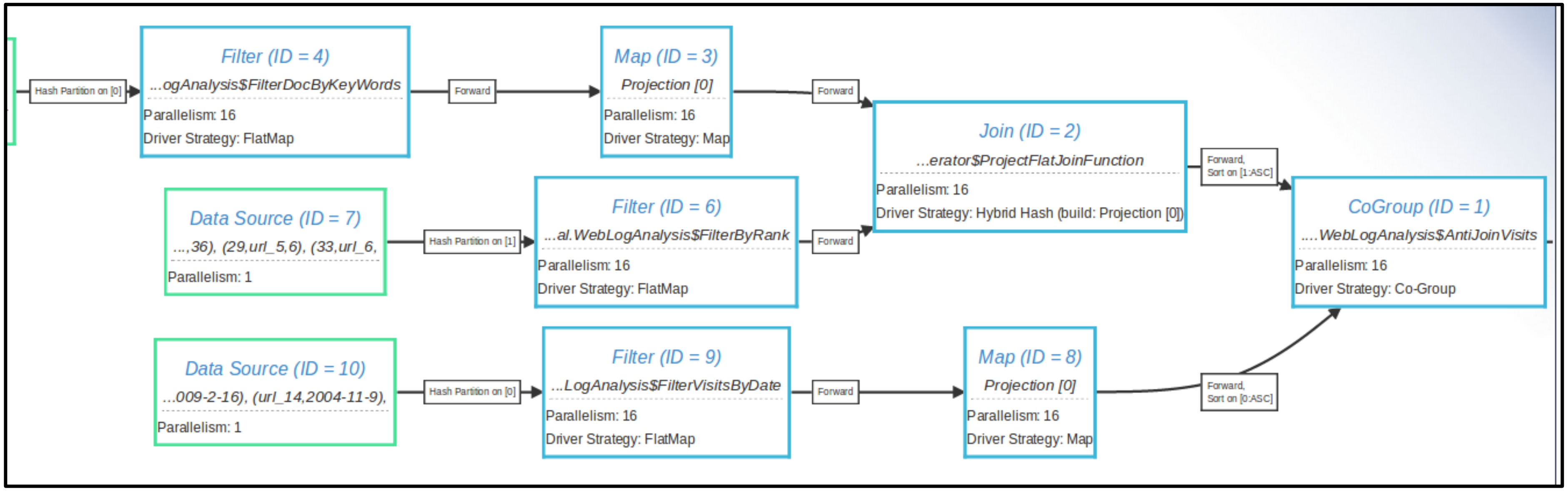
```
case class Word (word: String, frequency: Int)

val env = ExecutionEnvironment.getExecutionEnvironment()

val lines = env.readTextFile(...)

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

env.execute()
```

Flink has mirrored Java and Scala APIs that offer the same functionality, including by-name addressing.

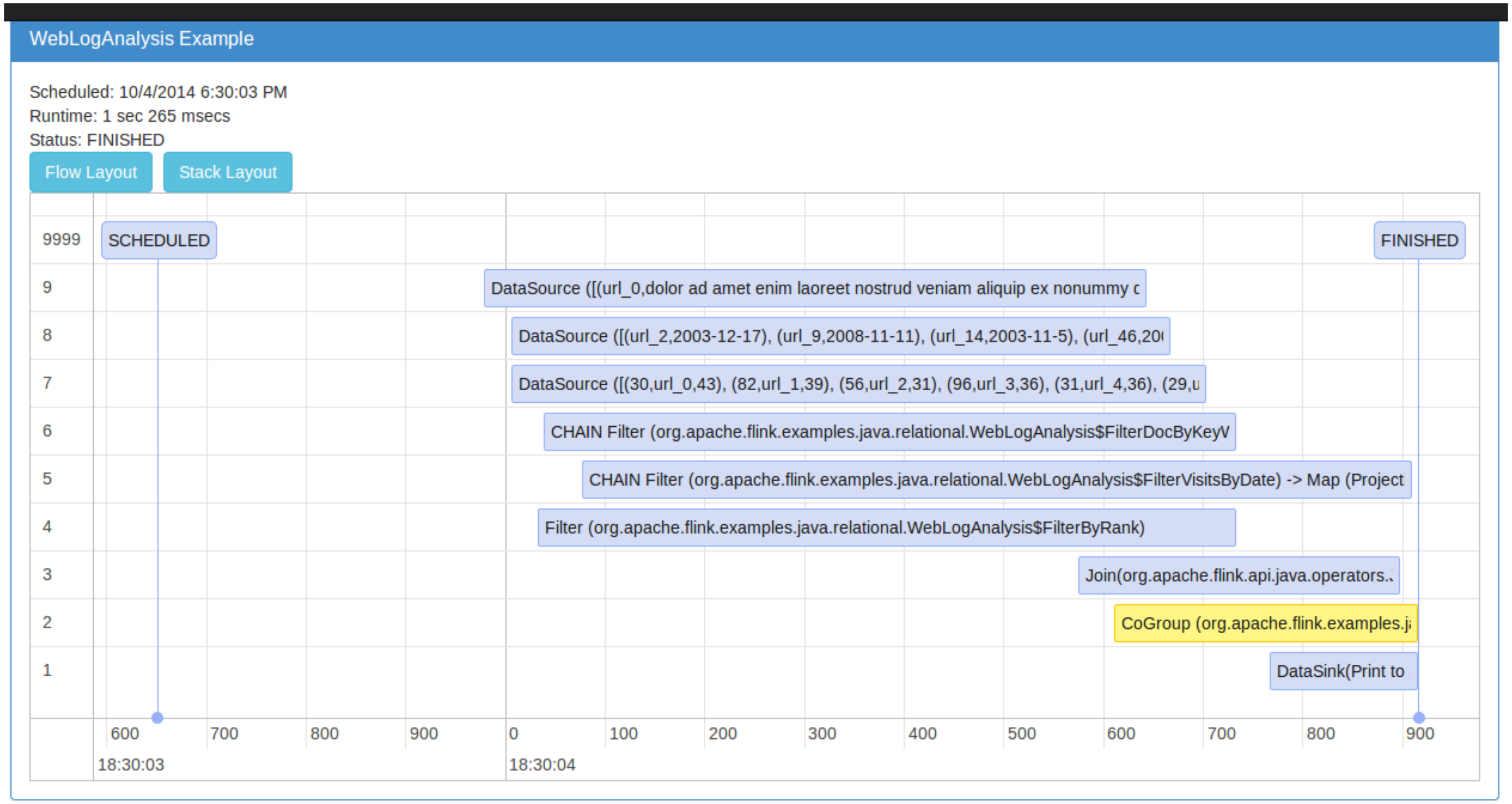
Flink API in a Nutshell

- map, flatMap, filter, groupBy, reduce, reduceGroup, aggregate, join, coGroup, cross, project, distinct, union, iterate, iterateDelta, ...
- All Hadoop input formats are supported
- API similar for data sets and data streams with slightly different operator semantics
- Window functions for data streams
- Counters, accumulators, and broadcast variables

Visualization tools



Visualization tools



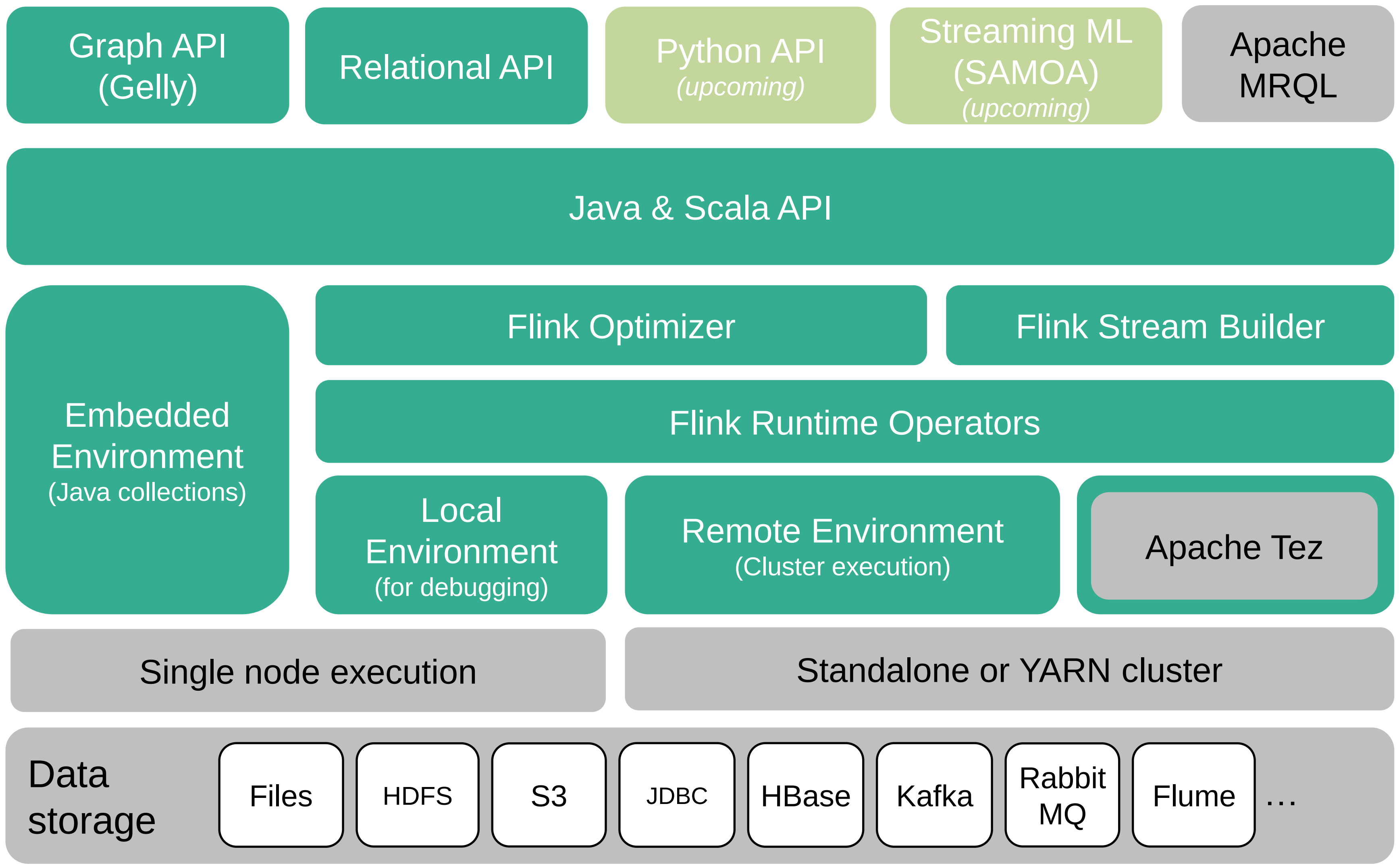
WebLogAnalysis Example

Scheduled: 10/4/2014 6:30:03 PM
Runtime: 1 sec 265 msecs
Status: FINISHED

Flow Layout
Stack Layout
9999
SCHEDULED
FINISHED
9
DataSource ([(url_0,dolor ad amet enim laoreet nostrud veniam aliquip ex nonummy c
8
DataSource ([(url_2,2003-12-17), (url_9,2008-11-11), (url_14,2003-11-5), (url_46,20
7
DataSource ([(30,url_0,43), (82,url_1,39), (56,url_2,31), (96,url_3,36), (31,url_4,36), (29,u
6
CHAIN Filter (org.apache.flink.examples.java.relational.WebLogAnalysis$FilterDocByKeyV
5
CHAIN Filter (org.apache.flink.examples.java.relational.WebLogAnalysis$FilterVisitsByDate) -> Map (Project
4
Filter (org.apache.flink.examples.java.relational.WebLogAnalysis$FilterByRank)
3
Join(org.apache.flink.api.java.operators.
2
CoGroup (org.apache.flink.examples.j
1
DataSink(Print to
600
700
800
900
0
100
200
300
400
500
600
700
800
900
18:30:03
18:30:04

Flink Architecture

Flink stack



Graph API (Gelly)
Relational API
Python API (upcoming)
Streaming ML (SAMOA) (upcoming)
Apache MRQL
Java & Scala API
Embedded Environment (Java collections)
Flink Optimizer
Flink Stream Builder
Flink Runtime Operators
Local Environment (for debugging)
Remote Environment (Cluster execution)
Apache Tez
Single node execution
Standalone or YARN cluster
Data storage
Files
HDFS
S3
JDBC
HBase
Kafka
Rabbit MQ
Flume
…

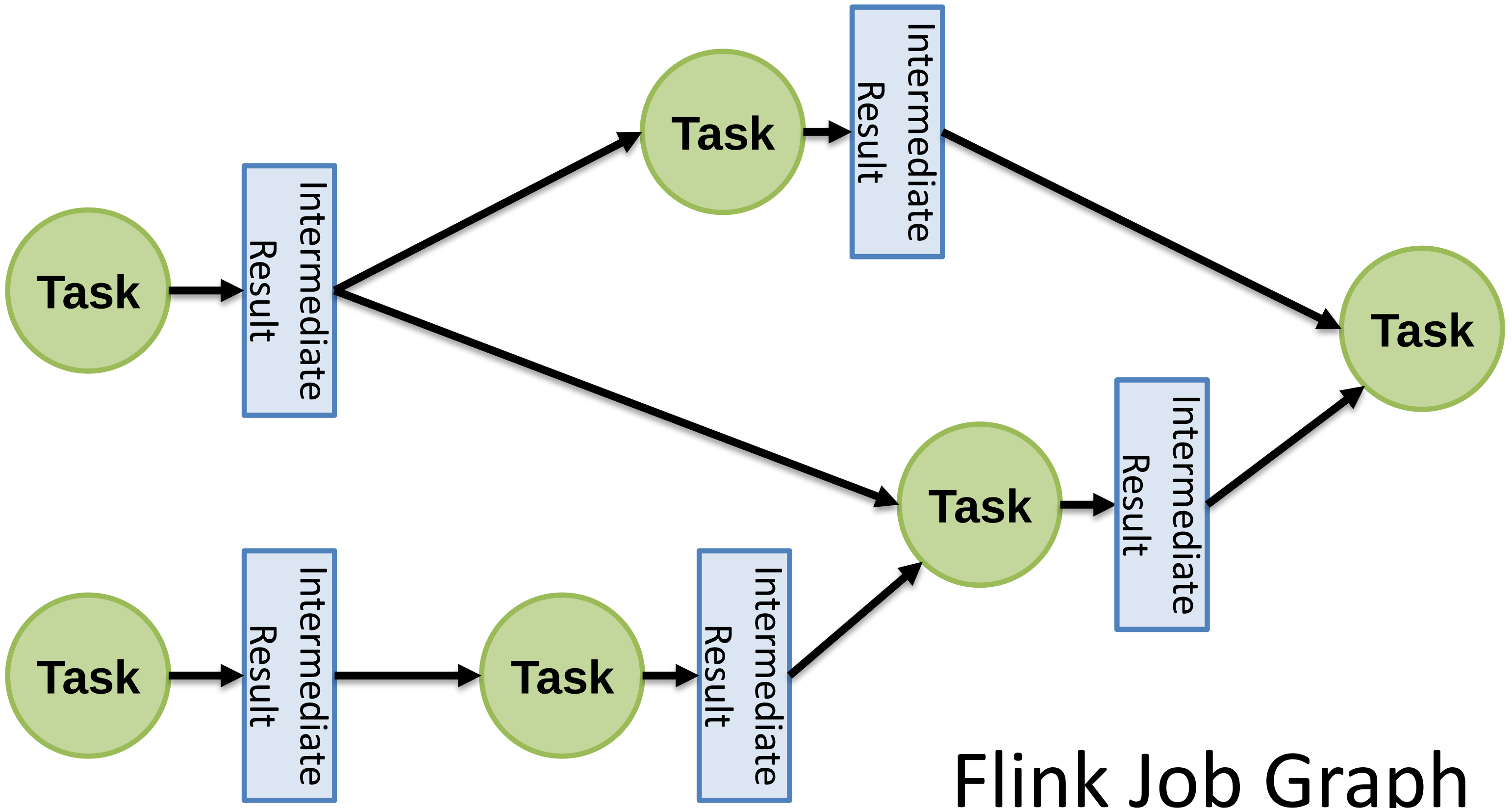
Evolution of Architectures

Operator-centric
(MapReduce / DAGs)

Dataset-centric
(RDDs)

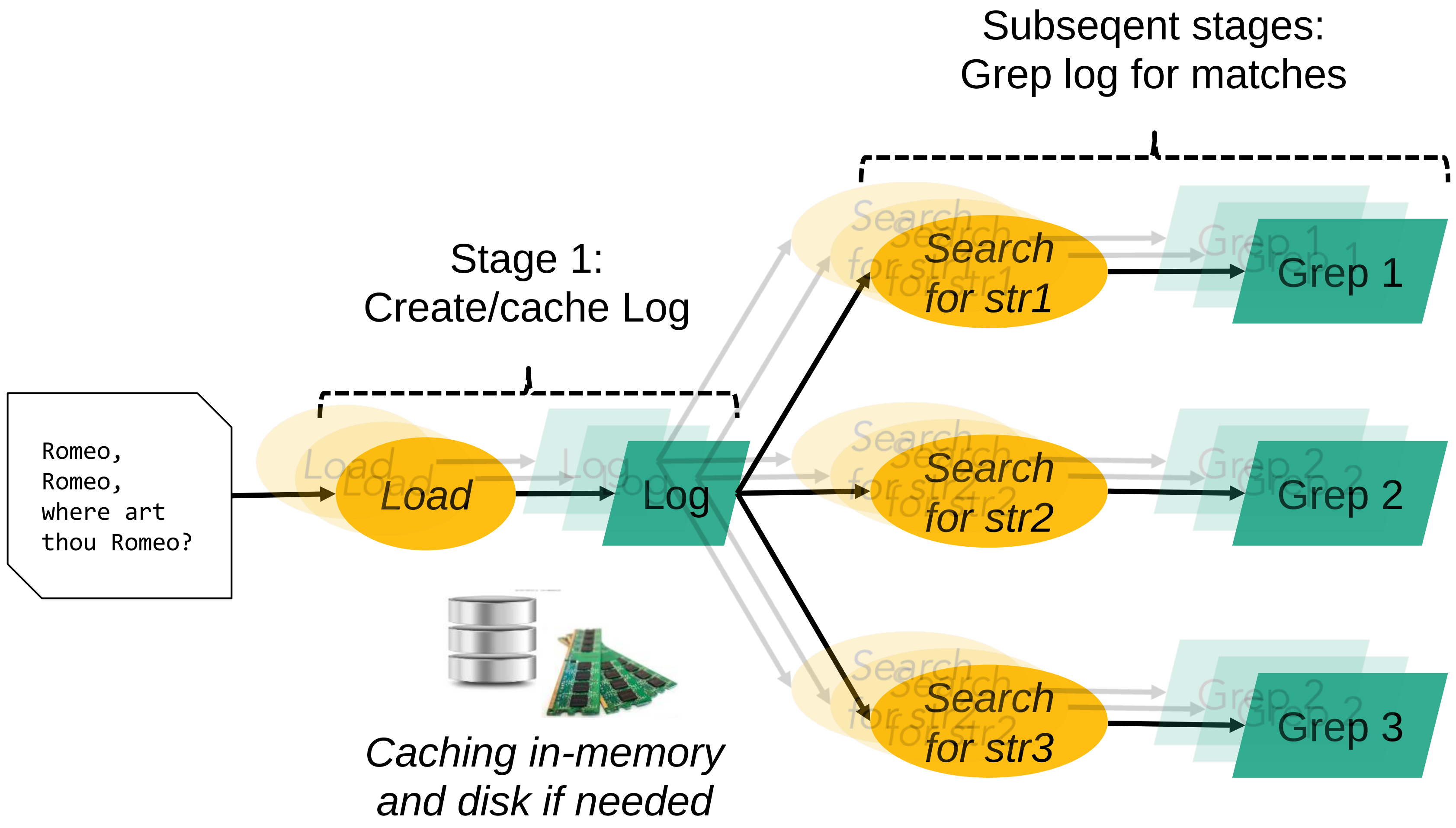
Operators and DataSets

Operators and Data Sets



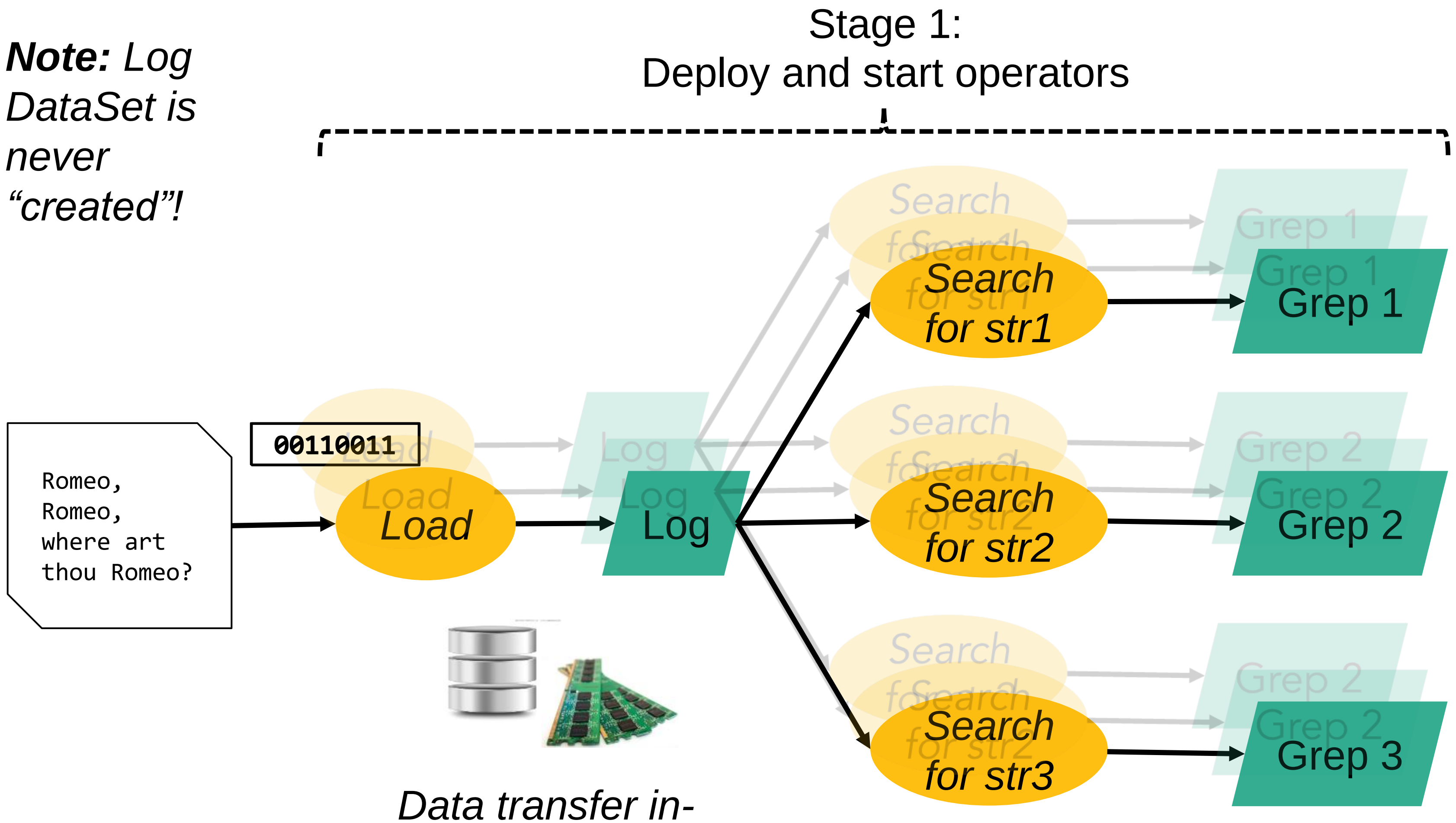
Flink Job Graph

Staged (batch) execution



Pipelined execution

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



Data transfer in-memory and disk if needed

Pipelining in Flink

- Currently the default mode of operation
  - Much better performance in many cases – no need to materialize large data sets
  - Supports both batch and real-time streaming

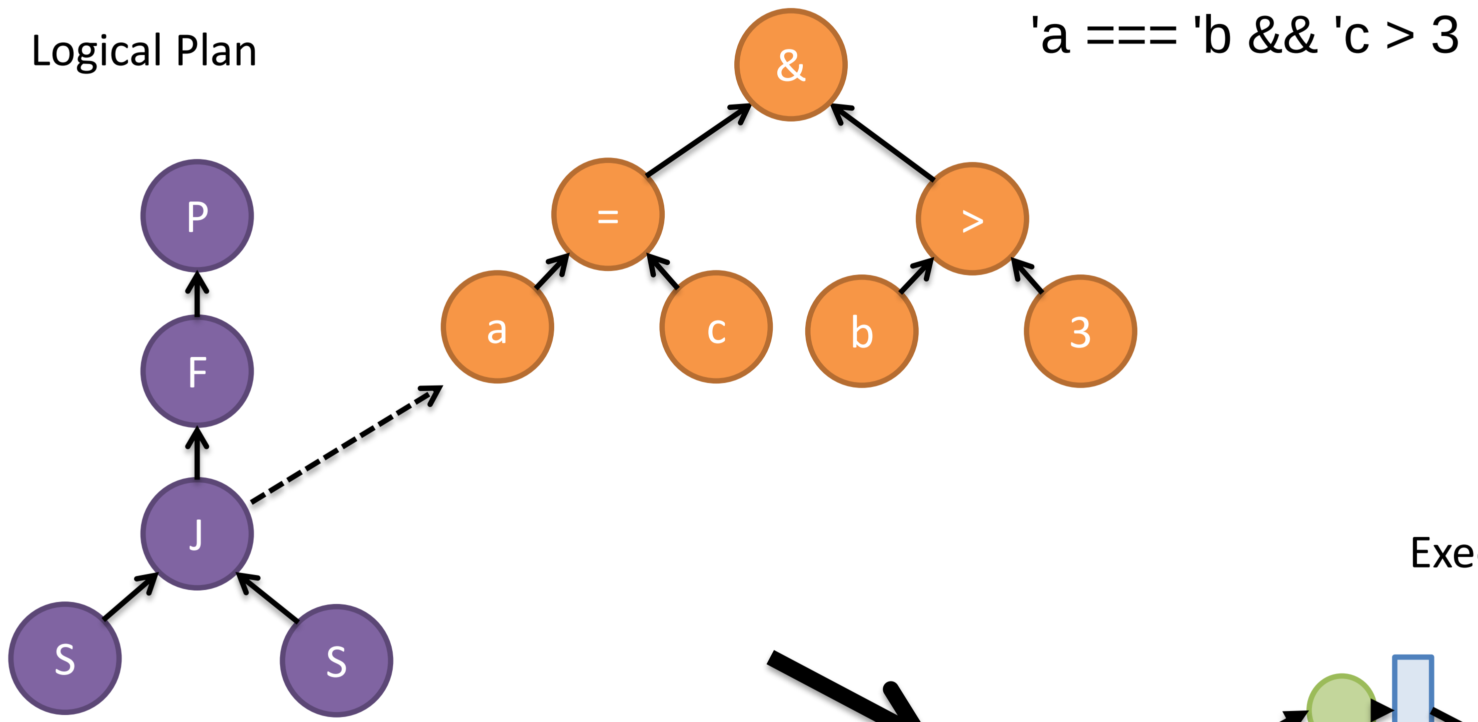
- Currently evolving into a hybrid engine
  - Batch will use combination of blocking and pipelining
  - Streaming will use pipelining
  - Interactive will use blocking

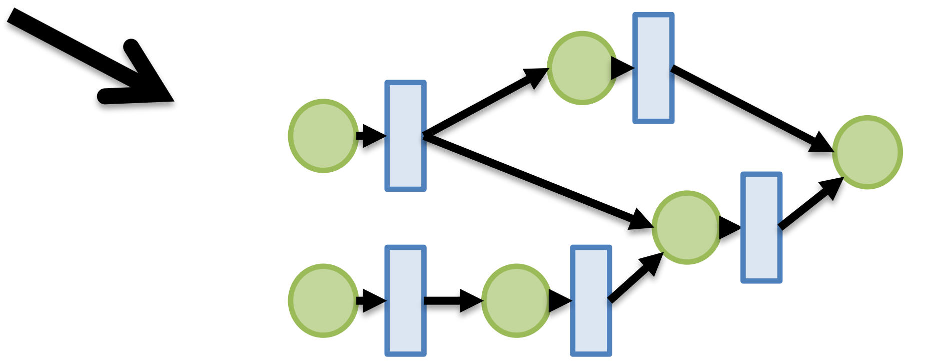
Relational API

First Things First

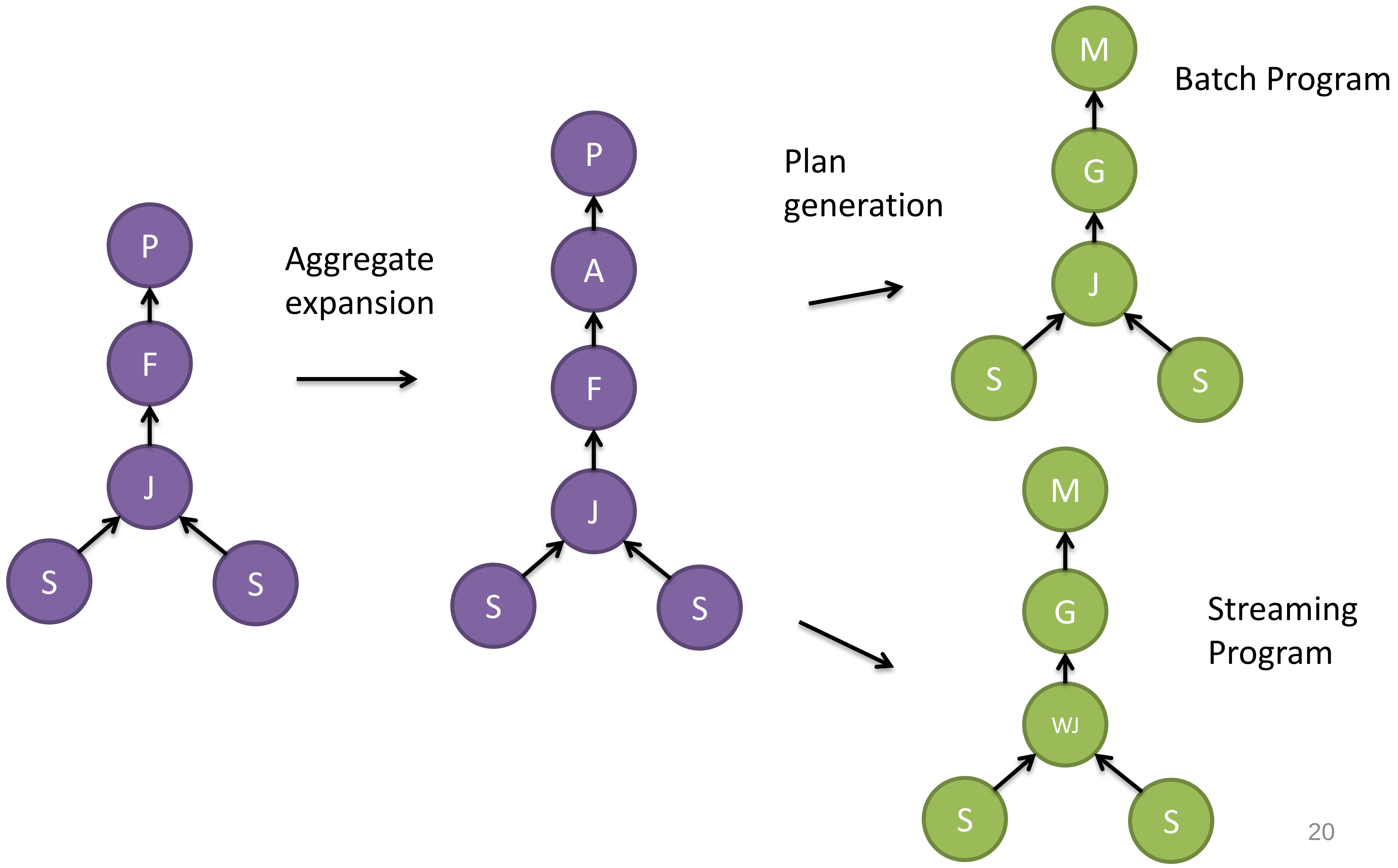
```
val clickCounts= clicks
  .groupBy('user)
  .select('userId, 'url.count as 'count)

val activeUsers = users.join(clickCounts)
  .where('id === 'userId && 'count > 10)
  .select('username, 'count)
```

Under the Hood





Plan Translation



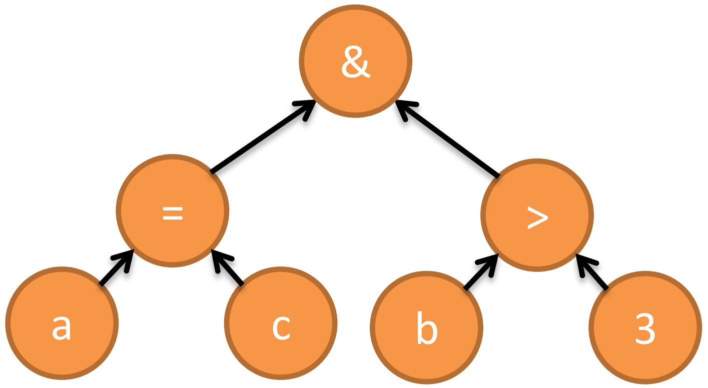
More Complex Expressions

```
in.select('a.substring(0, 'b.avg + 3))
```

```
Table result = in.select("a.count + 'the count'")
```

Expression Translation

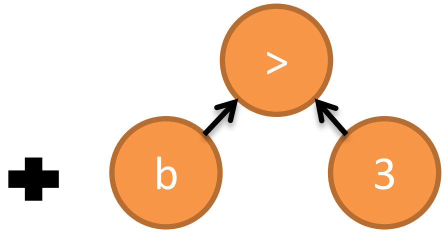
a = c && b > 3



def rules = *Seq*(
  new ResolveFieldReferences,
  new InsertAutoCasts,
  new TypeCheck,
  new VerifyNoAggregates,
  new VerifyBoolean,
  new ExtractEquiJoins
)

```
in1.join(in2).where(0).equalTo(0) {
  (l, r, out: Collector[Row]) =>
    if (r.get(1).asInstanceOf[Int) > 3) {
            val result = Row(4)
            result.set(0, l.get(0))
            …
            result.set(3, r.get(1))
      out.collect(result)
    }
}
```

join({0}, {0})



Expression Translation

```
in1.join(in2).where(0).equalTo(0) {
  (l, r, out: Collector[Row]) =>
    if (r.get(1).asInstanceOf[Int) > 3) {
            val result = Row(4)
            result.set(0, l.get(0))
            …
            result.set(3, r.get(1))
      out.collect(result)
    }
}
```

Back and Forth

```
case class In(a: Int, b: String)
case class Out(c: String, d: Int)
…

val input: DataSet[In] = …
val in = input.toTable()
val result = in.groupBy("b").select("b, a.avg”)

val output = result.as("c, d").toSet[Out]
```

- Supports POJOs, Case classes, Tuples

Back and Forth

```
class In {                    class Out {
  public int a;                 public String c;
  public String b;                public int d;
}                          }
…

DataSet<In> input = …
Table table = TableUtil.toTable(input);
Table result = table.groupBy("b").select("b, a.avg");

DataSet<Out> output =
  TableUtil.toSet(result.as("c, d"), Out.class);
```

- Supports POJOs, Case classes, Tuples

What Works?

- Relational queries from both Java and Scala
- Translation to batch programs
- Preliminary translation to streaming programs

Future Work

- Relational Optimization
  - Filter/Projection push down
  - Join order
- Operator Fusion
- Extend expressions
  - string operations, casting, explode/gather, date/time, ...
- Windowing operations (streaming)
- Columnar execution?

Closing

Flink Roadmap for 2015

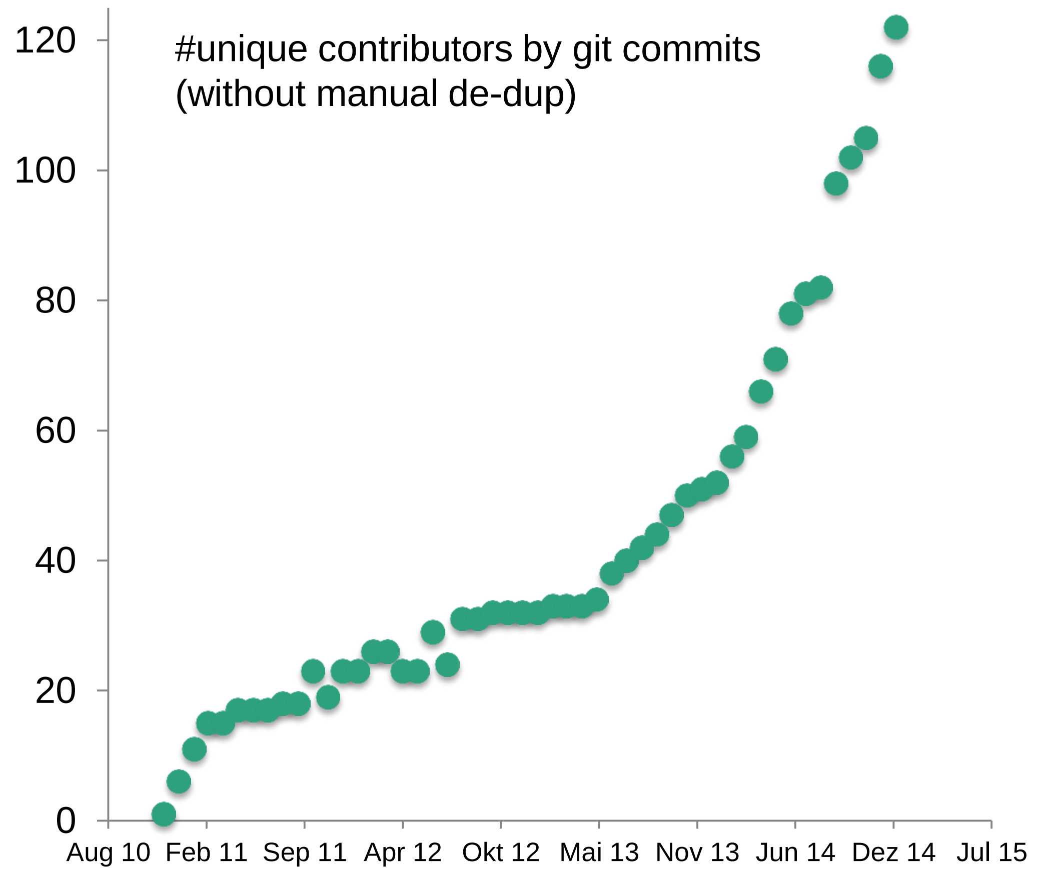
- Exactly-once streaming with flexible state
- Support for Google Dataflow
- Batch Machine Learning library
- Streaming Machine Learning with SAMOA
- Graph library additions (more algorithms)
- Interactive programs and Zeppelin
- SQL on top of Relational API
- and more…

flink.apache.org
@ApacheFlink

Backup Slides

dataArtisans

Flink community


#unique contributors by git commits
(without manual de-dup)
Aug 10
Feb 11
Sep 11
Apr 12
Okt 12
Mai 13
Nov 13
Jun 14
Dez 14
Jul 15