ANALYZE AND GET START

FLINK SQL

@时金魁 2016/11/04

HUAWEI TECHNOLOGIES CO., LTD.

OUTLINE

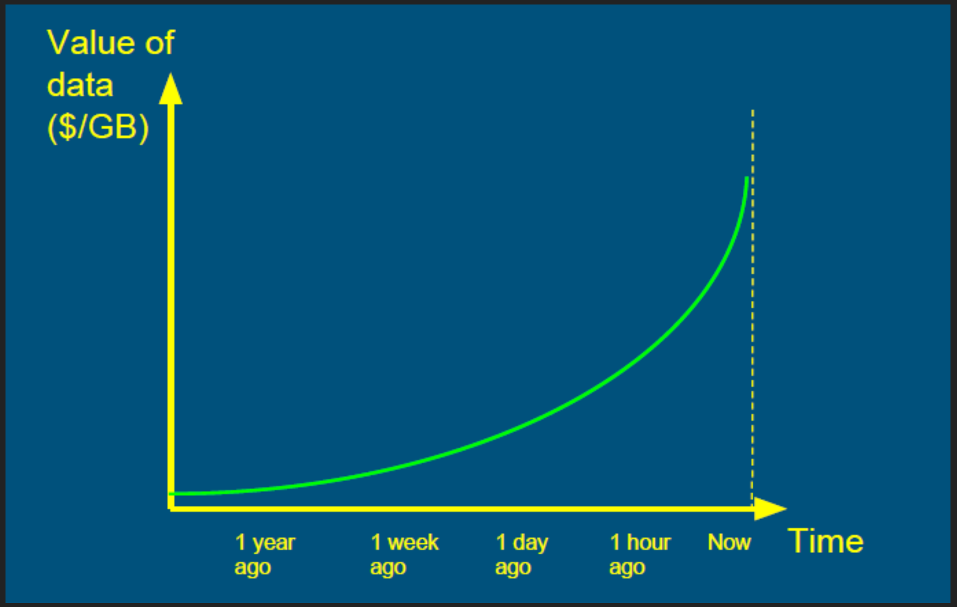
- Why Flink, Why Flink SQL?
- What Flink SQL look like in depth
- How Flink SQL executed on runtime
- QA

WHO AM I

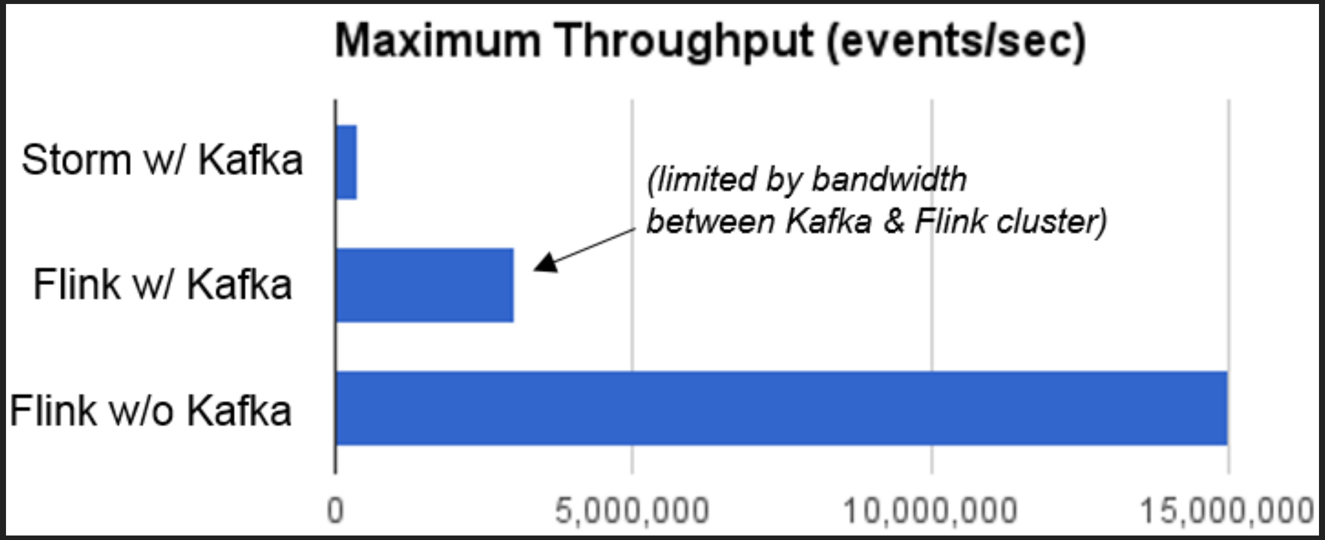
- 时金魁 Jinkui Shi
- ==> Hexun，Sohu，Alibaba，Huawei

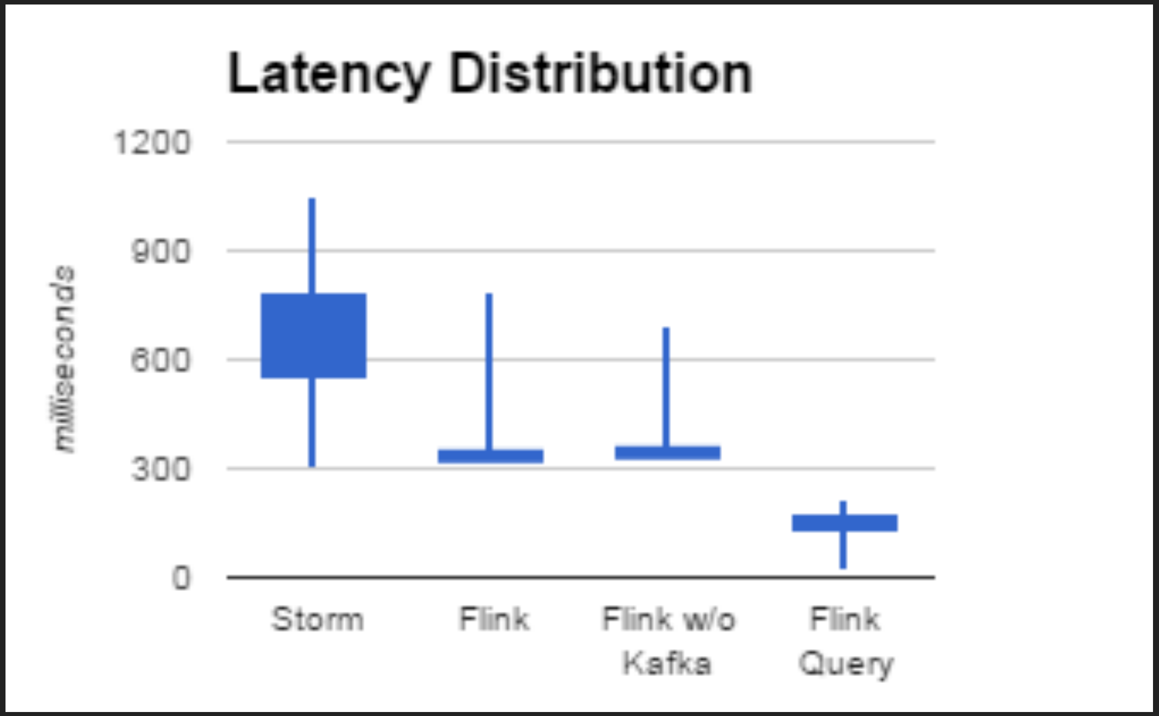
Why Flink

- your users are always online
- stream data are everywhere: web, services, logs, IoT, devices, DB..
- product require low latency: risk-management, online-xxx..
- mixed user fast response and CEP、Gelly、FlinkML



from Rado



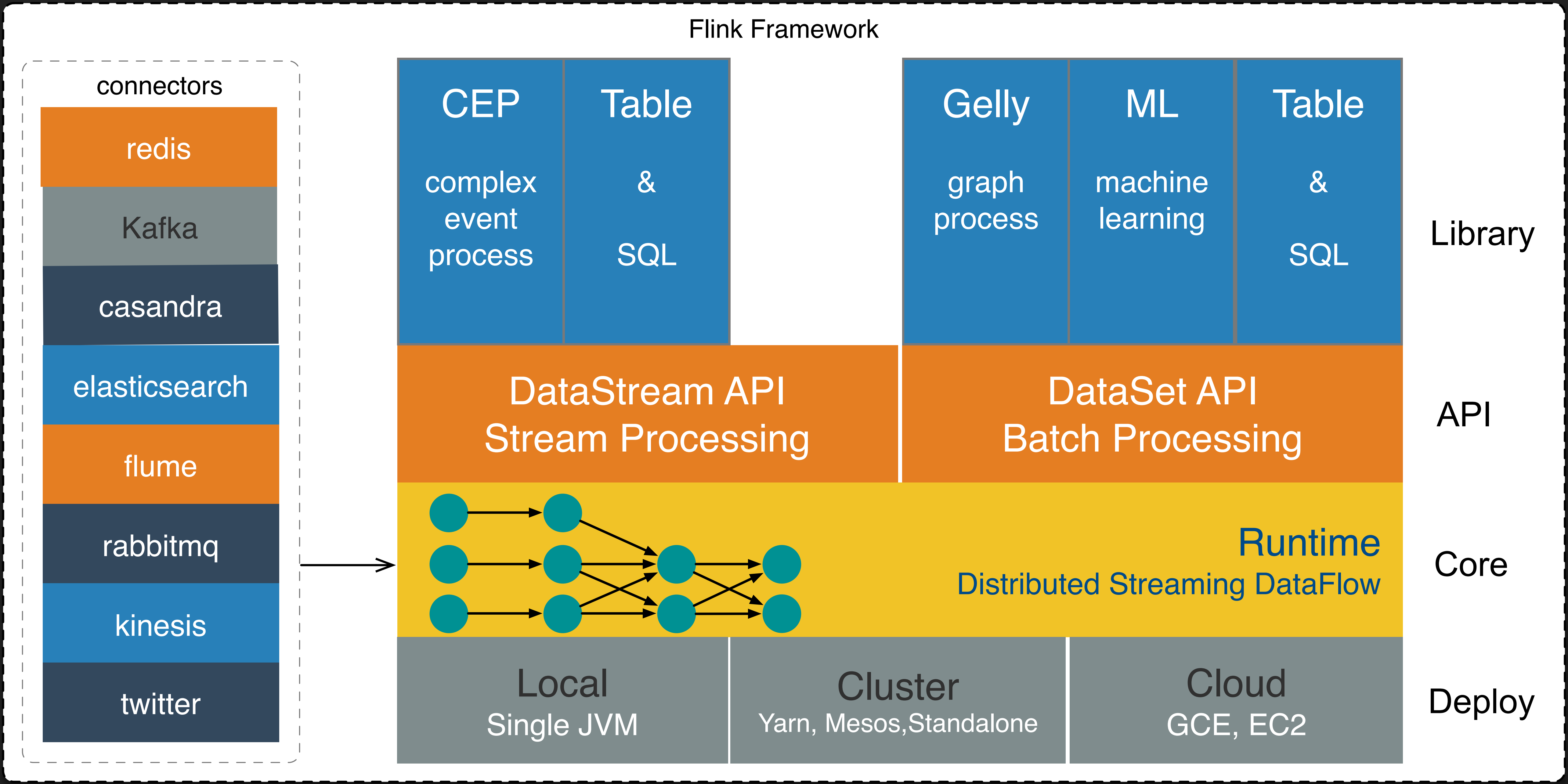


http://data-artisans.com/extending-the-yahoo-streaming-benchmark

Why Flink SQL

- Standard API
- Query planner optimizer converts logical plan to physical plan
- “Your database is just a cache of my stream”
- “Your stream is just change-capture of my database”
- “Data is the new oil” –*Julian Hyde*
- *apache calcite will be streaming sql standard, used by Drill, Hive, Kylin, Phoenix, Cascading, Flink, Storm, Samza..*

Flink stack

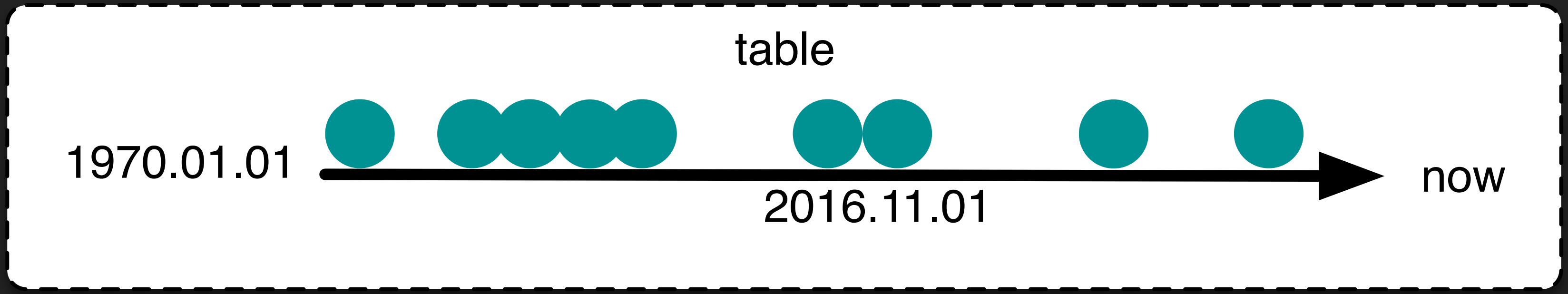


How to use Flink:

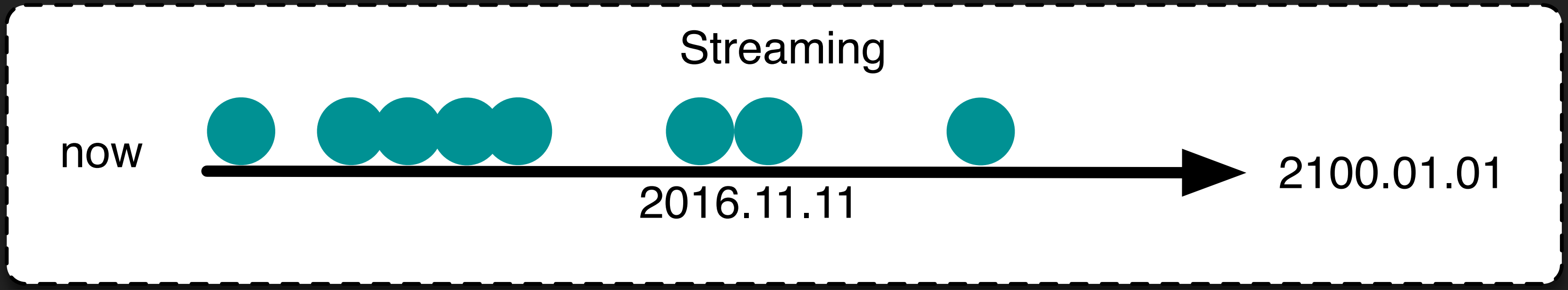
1. Connectors
2. Table API and SQL
3. CEP
4. Gelly
5. FlinkML

Stream-table duality

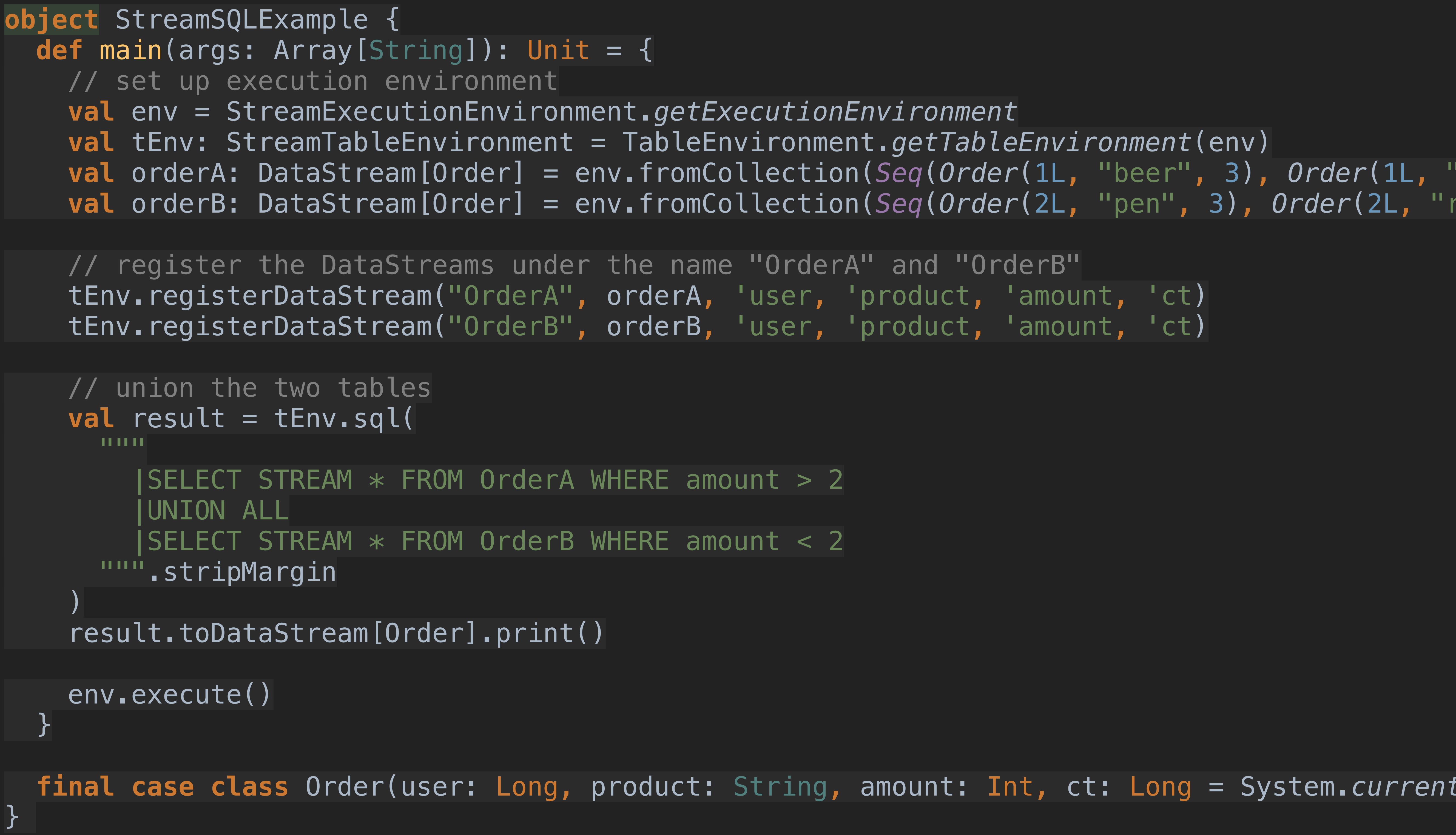
SELECT * FROM TABLE_1 WHERE ID > 1000



SELECT STREAM * FROM TABLE_1 WHERE ID > 1000



Flink SQl example to get start



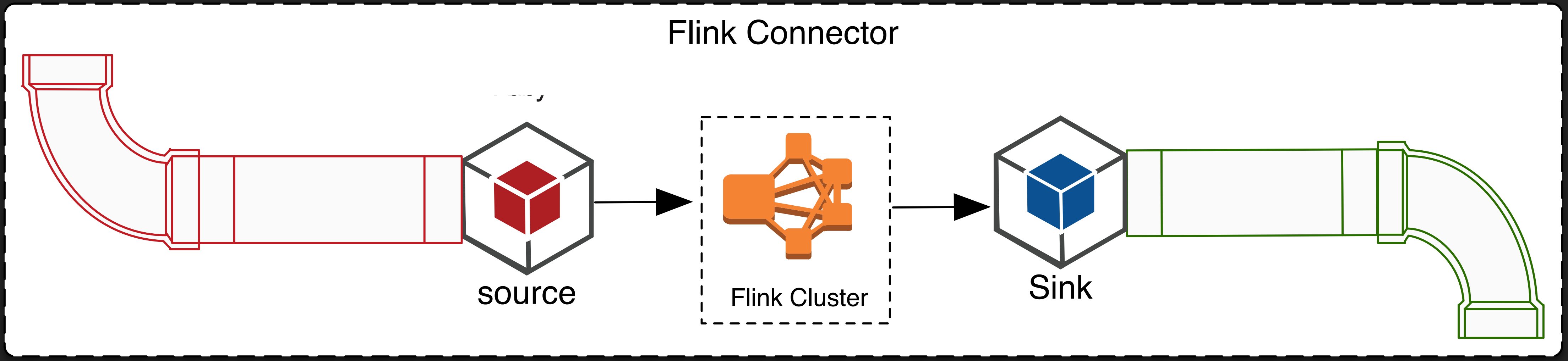
```
object StreamSQLExample {
  def main(args: Array[String]): Unit = {
    // set up execution environment
    val env = StreamExecutionEnvironment.getExecutionEnvironment
    val tEnv: StreamTableEnvironment = TableEnvironment.getTableEnvironment(env)
    val orderA: DataStream[Order] = env.fromCollection(Seq(Order(1L, "beer", 3), Order(1L, "
    val orderB: DataStream[Order] = env.fromCollection(Seq(Order(2L, "pen", 3), Order(2L, "r

    // register the DataStreams under the name "OrderA" and "OrderB"
    tEnv.registerDataStream("OrderA", orderA, 'user, 'product, 'amount, 'ct)
    tEnv.registerDataStream("OrderB", orderB, 'user, 'product, 'amount, 'ct)

    // union the two tables
    val result = tEnv.sql(
      """
        |SELECT STREAM * FROM OrderA WHERE amount > 2
        |UNION ALL
        |SELECT STREAM * FROM OrderB WHERE amount < 2
      """.stripMargin
    )
    result.toDataStream[Order].print()

    env.execute()
  }

  final case class Order(user: Long, product: String, amount: Int, ct: Long = System.current
}
```

SELECT STREAM * FROM ORDERA WHERE AMOUNT > 2
UNION ALL
SELECT STREAM * FROM ORDERB WHERE AMOUNT < 2

Internal Flink SQL

stream	java	scala 2.10	scala 2.11
table	java	scala 2.10	scala 2.11

General process



main:

=> Source

=> DataStream[Any]

=> SQL Query -> RelNode -> translateToPlan() -> codegen() -> function object

=> DataStream[Any] *(include function)*

=> *Runtime*

step 1:

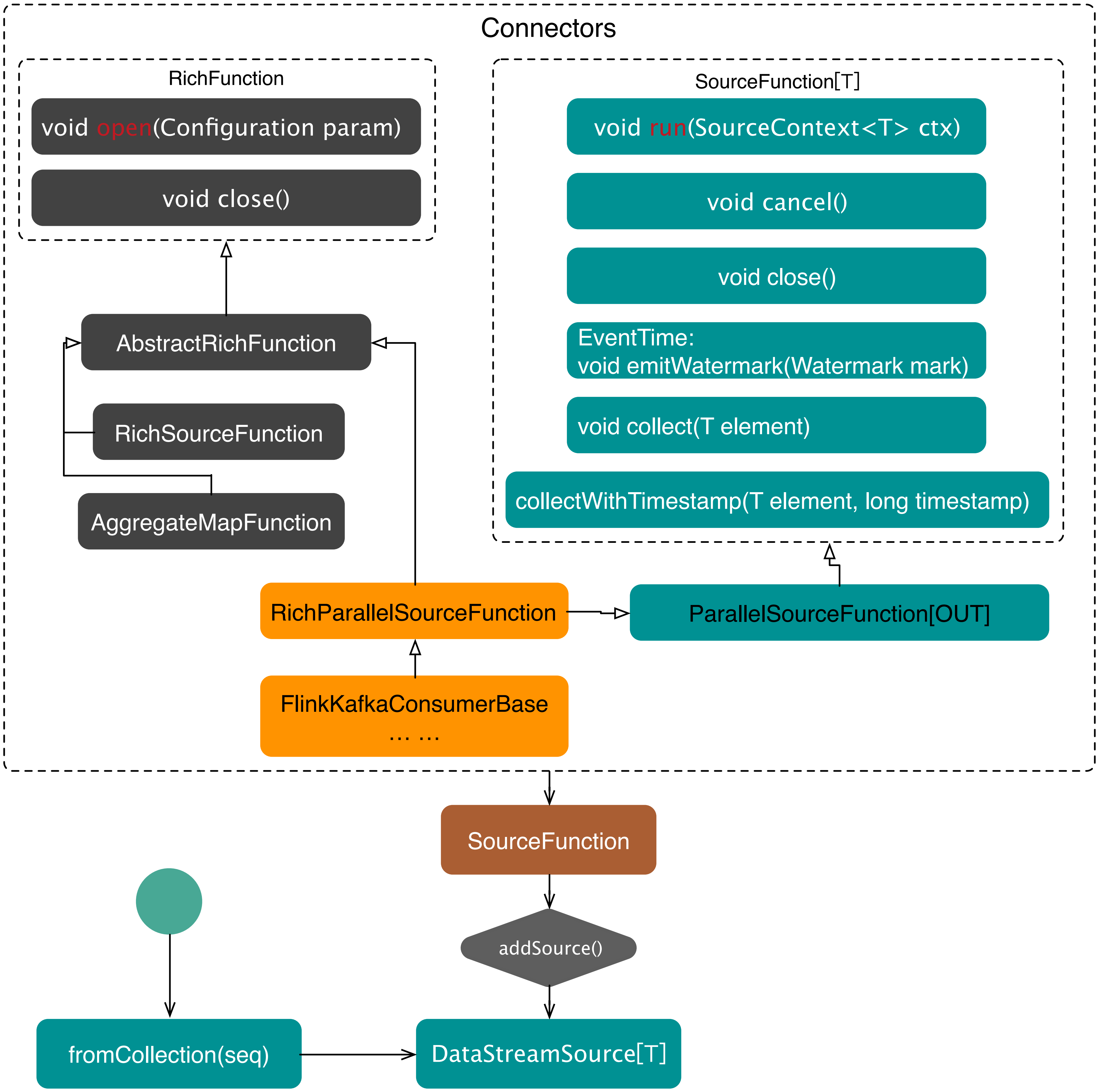
generate DataStrem

from source of

connector

```
env.fromCollection(Seq(
Order(1L, "beer", 3),
Order(1L, "diaper", 4),
Order(3L, "rubber", 2)
))
```

generate data stream from source



Connectors

RichFunction

void open(Configuration param)

void close()

AbstractRichFunction

RichSourceFunction

AggregateMapFunction

SourceFunction[T]

void run(SourceContext<T> ctx)

void cancel()

void close()

EventTime:
void emitWatermark(Watermark mark)

void collect(T element)

collectWithTimestamp(T element, long timestamp)

RichParallelSourceFunction

ParallelSourceFunction[OUT]

FlinkKafkaConsumerBase
… …

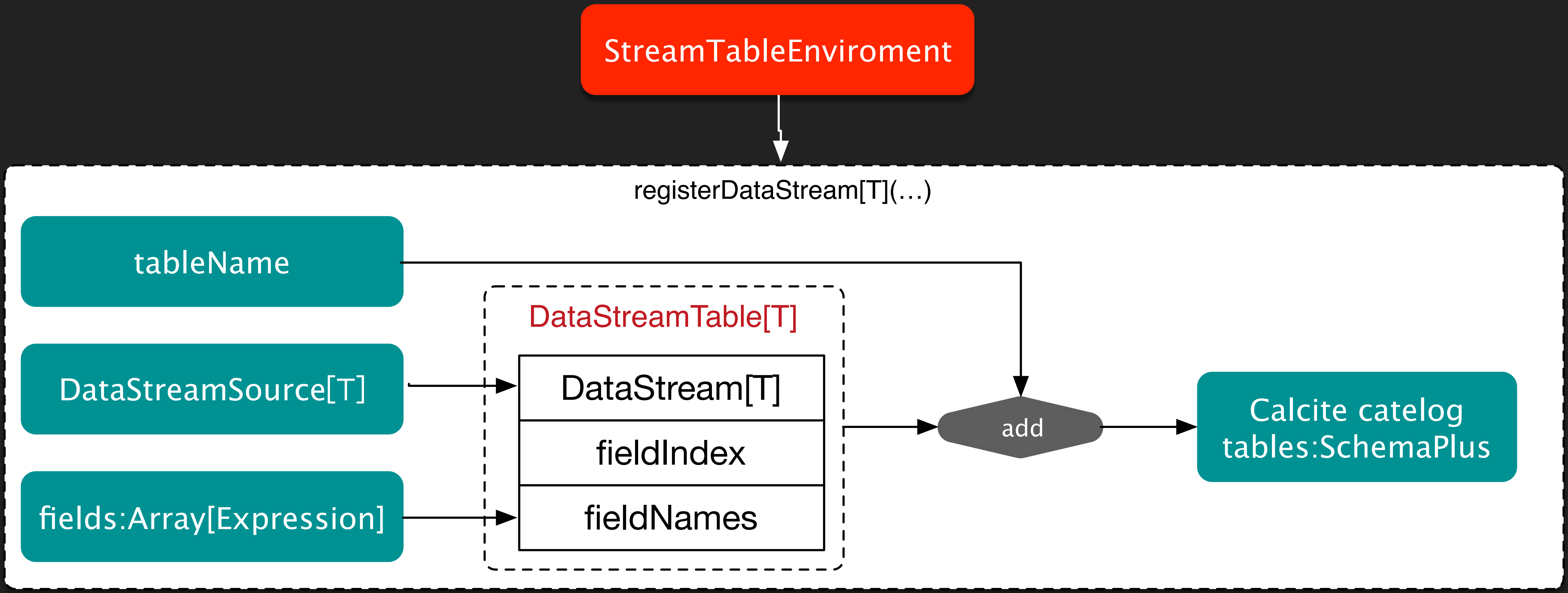
SourceFunction

addSource()

fromCollection(seq)

DataStreamSource[T]

Step 2: Register table to calcite



register the schema(field name and field index) info to calcite

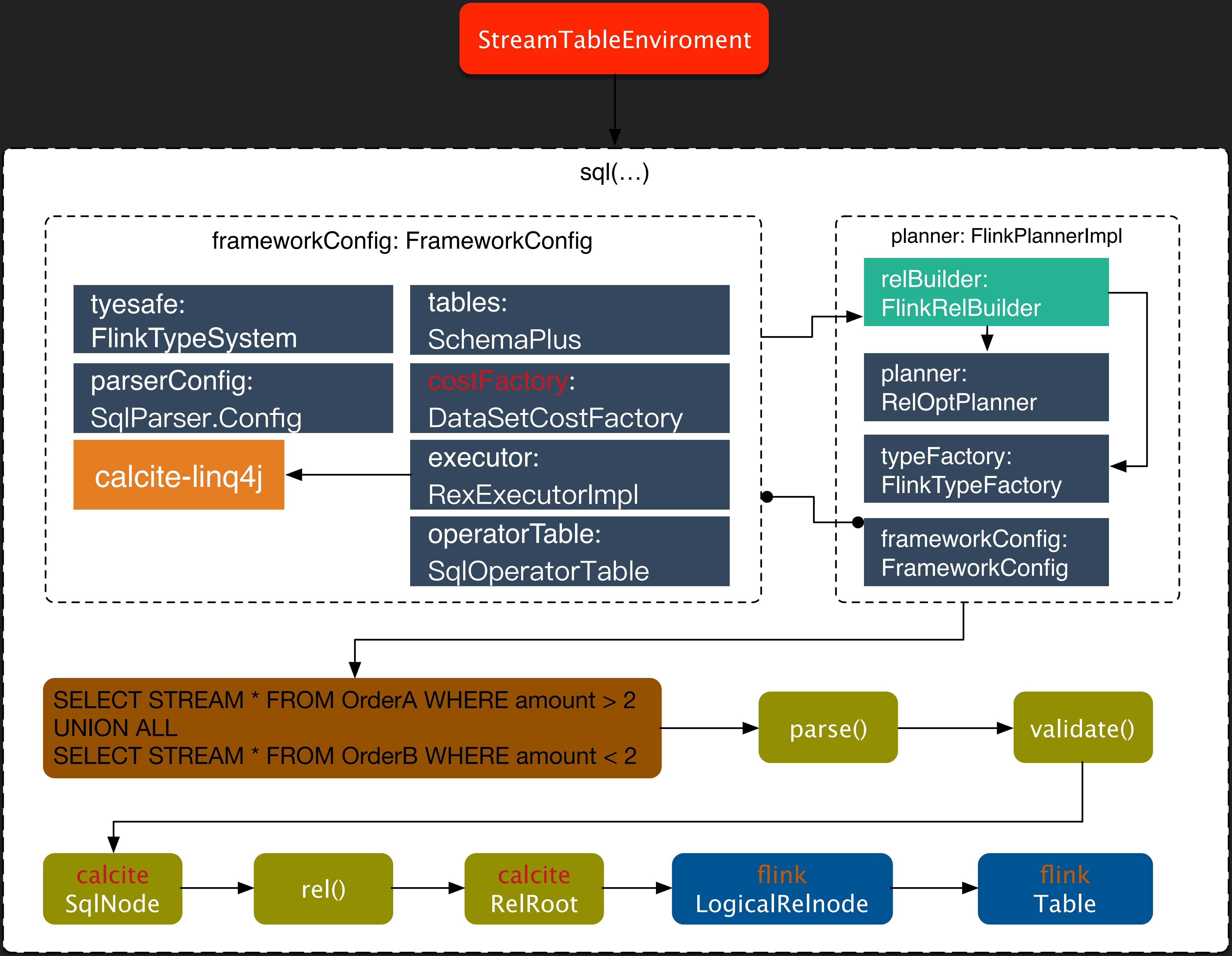
code:

```
tEnv.registerDataStream("OrderA", orderA, 'user, 'product, 'amount, 'ct)
tEnv.registerDataStream("OrderB", orderB, 'user, 'product, 'amount, 'ct)
```

Step 3: sql parse, validate, generate relational tree => Flink Table

```
tEnv.sql(
  """
    |SELECT STREAM * FROM OrderA WHERE amount > 2
    |UNION ALL
    |SELECT STREAM * FROM OrderB WHERE amount < 2
  """.stripMargin
)
```

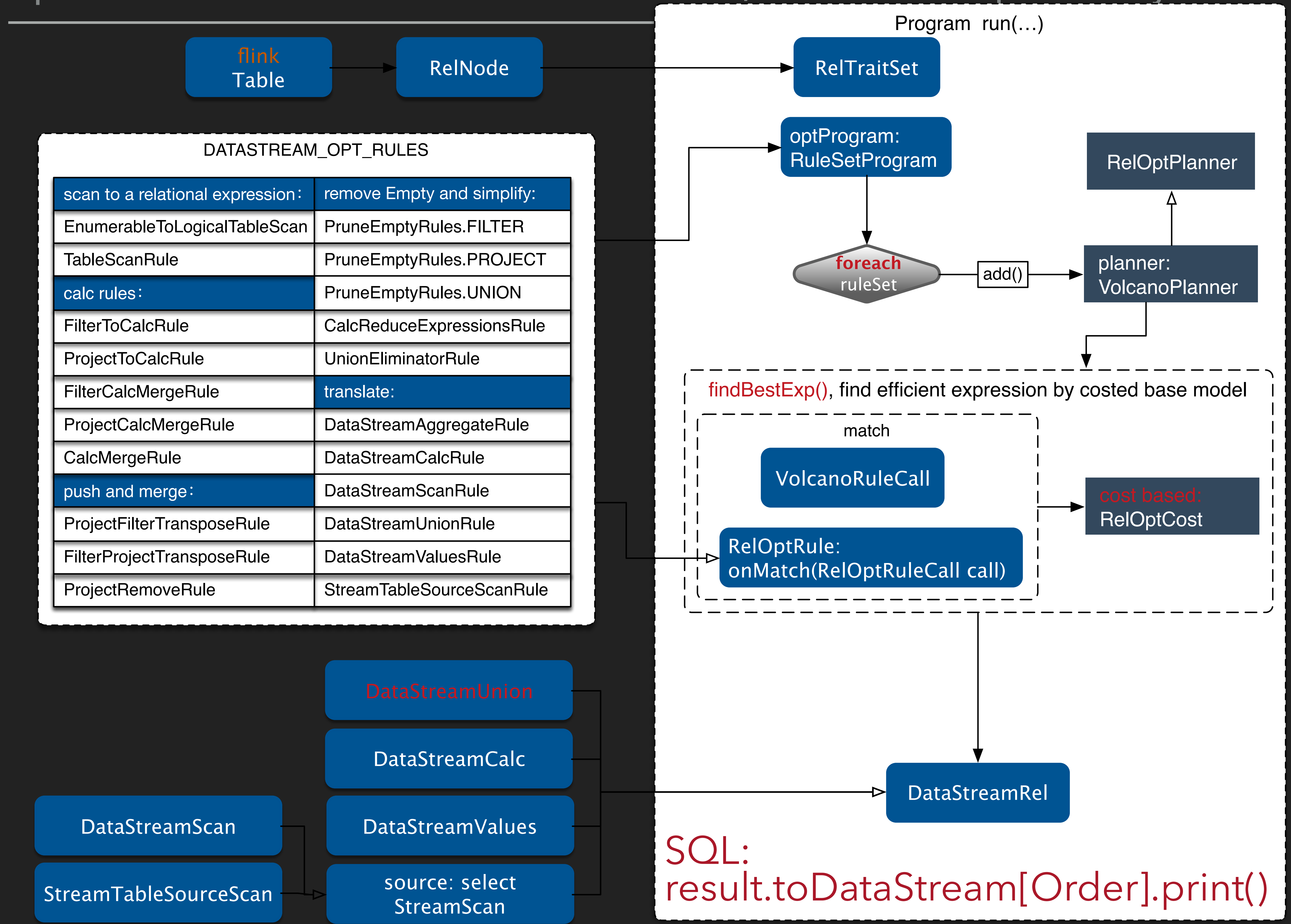
Step 3: sql parse, validate, generate relational tree => Flink Table



StreamTableEnviroment
sql(…)
frameworkConfig: FrameworkConfig
tyesafe: FlinkTypeSystem
tables: SchemaPlus
parserConfig: SqlParser.Config
costFactory: DataSetCostFactory
calcite-linq4j
executor: RexExecutorImpl
operatorTable: SqlOperatorTable
planner: FlinkPlannerImpl
relBuilder: FlinkRelBuilder
planner: RelOptPlanner
typeFactory: FlinkTypeFactory
frameworkConfig: FrameworkConfig
parse()
validate()
calcite SqlNode
rel()
calcite RelRoot
flink LogicalRelnode
flink Table

SELECT STREAM * FROM OrderA WHERE amount > 2
UNION ALL
SELECT STREAM * FROM OrderB WHERE amount < 2

Step 4: translate Flink Table to DataStreamRel, costed base optimize by rules

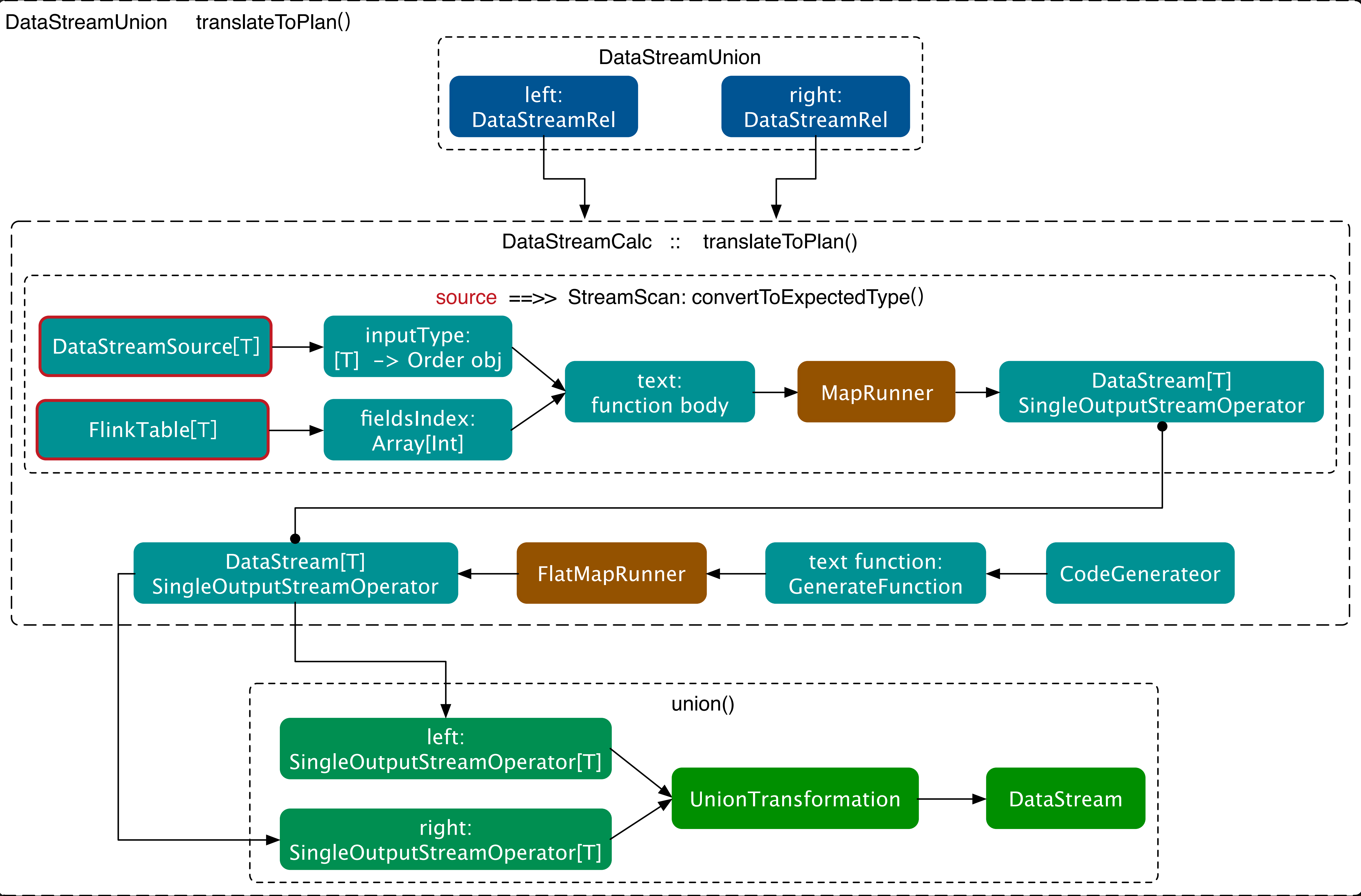


DATASTREAM_OPT_RULES

scan to a relational expression：	remove Empty and simplify:
EnumerableToLogicalTableScan	PruneEmptyRules.FILTER
TableScanRule	PruneEmptyRules.PROJECT
calc rules：	PruneEmptyRules.UNION
FilterToCalcRule	CalcReduceExpressionsRule
ProjectToCalcRule	UnionEliminatorRule
FilterCalcMergeRule	translate:
ProjectCalcMergeRule	DataStreamAggregateRule
CalcMergeRule	DataStreamCalcRule
push and merge：	DataStreamScanRule
ProjectFilterTransposeRule	DataStreamUnionRule
FilterProjectTransposeRule	DataStreamValuesRule
ProjectRemoveRule	StreamTableSourceScanRule

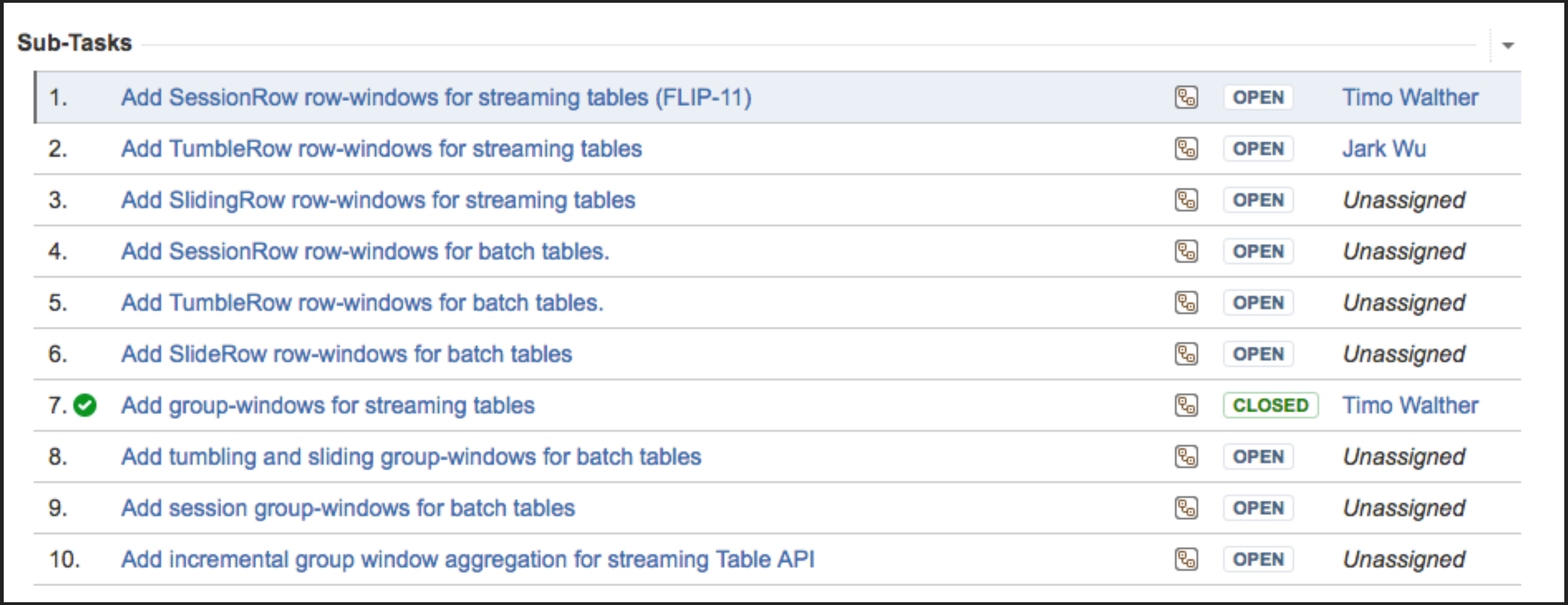
SQL:
result.toDataStream[Order].print()

Step 5: translate DataStreamRel to final DataStream[T]


DataStreamUnion translateToPlan()
DataStreamUnion
left: DataStreamRel
right: DataStreamRel
DataStreamCalc :: translateToPlan()
source ==>> StreamScan: convertToExpectedType()
DataStreamSource[T]
inputType: [T] –> Order obj
FlinkTable[T]
fieldsIndex: Array[Int]
text: function body
MapRunner
DataStream[T] SingleOutputStreamOperator
DataStream[T] SingleOutputStreamOperator
FlatMapRunner
text function: GenerateFunction
CodeGenerateor
union()
left: SingleOutputStreamOperator[T]
right: SingleOutputStreamOperator[T]
UnionTransformation
DataStream

FLIP-11: Table API Stream Aggregations

aggregate: groupBy, window, rowWindow



Sub-Tasks

1.	Add SessionRow row-windows for streaming tables (FLIP-11)	OPEN	Timo Walther
2.	Add TumbleRow row-windows for streaming tables	OPEN	Jark Wu
3.	Add SlidingRow row-windows for streaming tables	OPEN	*Unassigned*
4.	Add SessionRow row-windows for batch tables.	OPEN	*Unassigned*
5.	Add TumbleRow row-windows for batch tables.	OPEN	*Unassigned*
6.	Add SlideRow row-windows for batch tables	OPEN	*Unassigned*
7. ✓	Add group-windows for streaming tables	CLOSED	Timo Walther
8.	Add tumbling and sliding group-windows for batch tables	OPEN	*Unassigned*
9.	Add session group-windows for batch tables	OPEN	*Unassigned*
10.	Add incremental group window aggregation for streaming Table API	OPEN	*Unassigned*

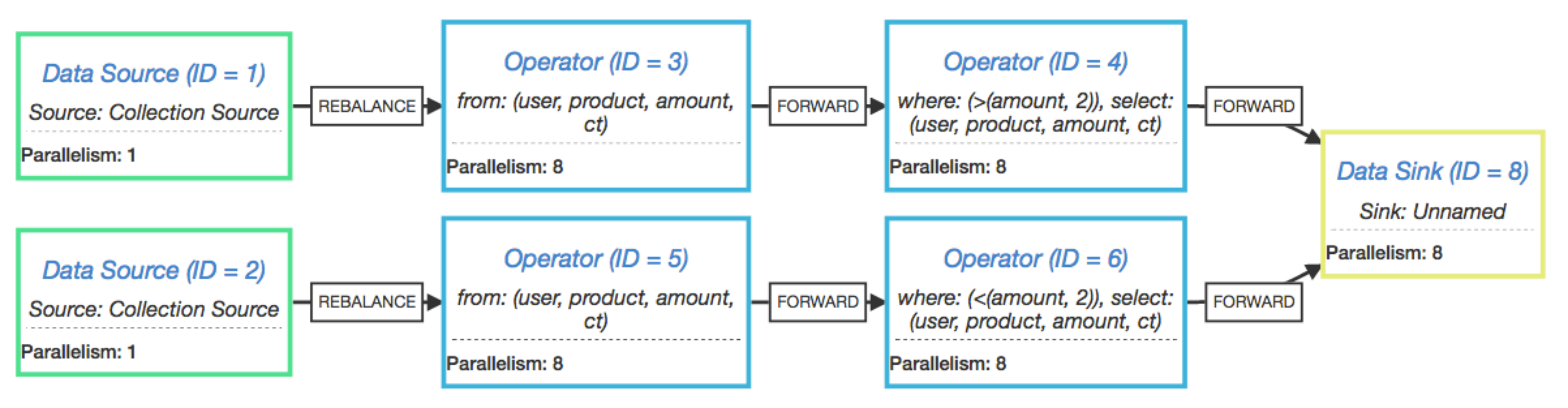
https://issues.apache.org/jira/browse/FLINK-4557

https://cwiki.apache.org/confluence/display/FLINK/FLIP-11%3A+Table+API+Stream+Aggregations

dataflow runtime

Plan Visualizer

env.getExecutionPlan -> http://flink.apache.org/visualizer



Data Source (ID = 1)
Source: Collection Source
Parallelism: 1

REBALANCE

Operator (ID = 3)
from: (user, product, amount, ct)
Parallelism: 8

FORWARD

Operator (ID = 4)
where: (>(amount, 2)), select: (user, product, amount, ct)
Parallelism: 8

FORWARD

Data Source (ID = 2)
Source: Collection Source
Parallelism: 1

REBALANCE

Operator (ID = 5)
from: (user, product, amount, ct)
Parallelism: 8

FORWARD

Operator (ID = 6)
where: (<(amount, 2)), select: (user, product, amount, ct)
Parallelism: 8

FORWARD

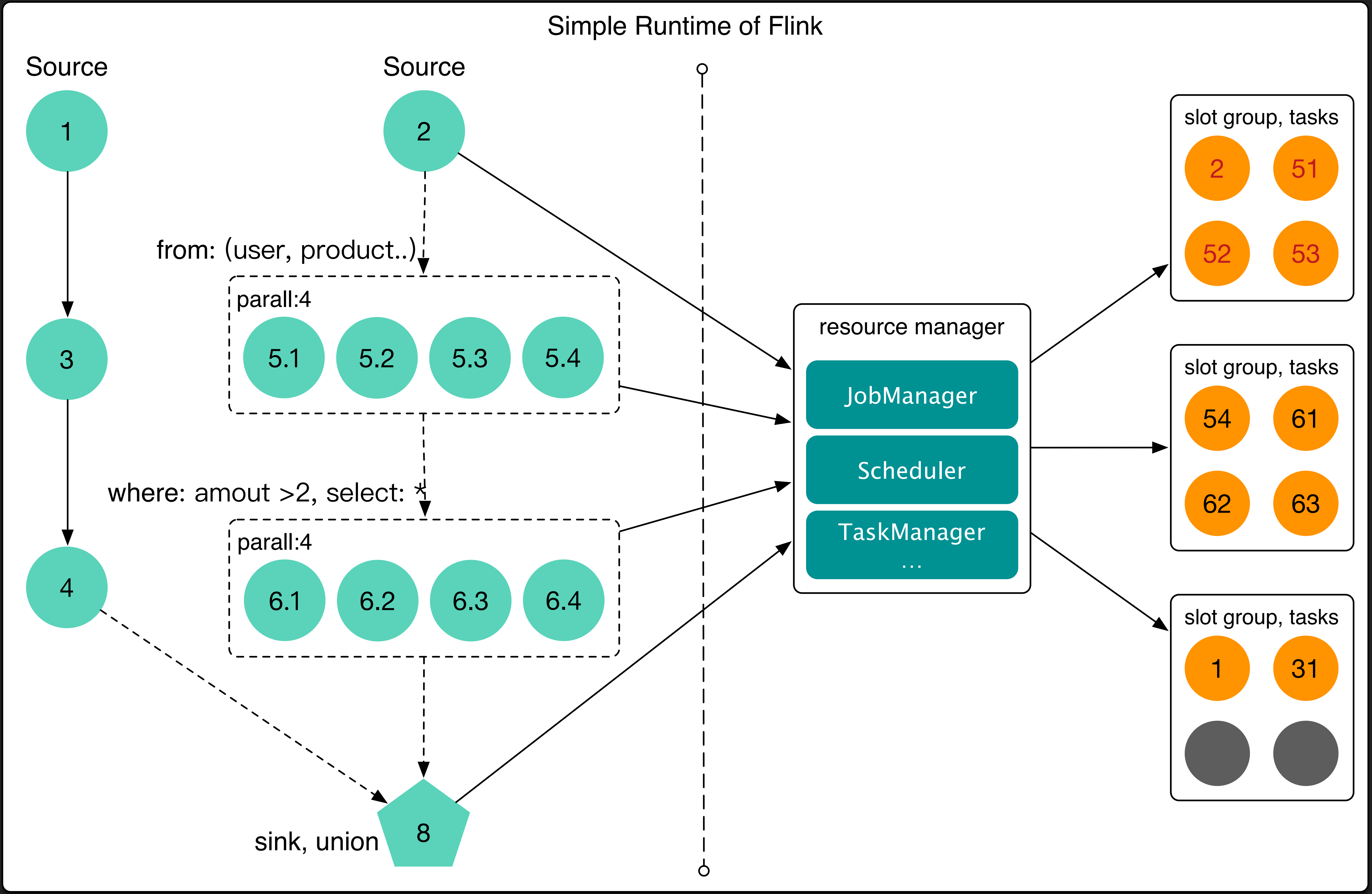
Data Sink (ID = 8)
Sink: Unnamed
Parallelism: 8

==> Source
SELECT STREAM * FROM OrderA WHERE amount > 2
UNION ALL
SELECT STREAM * FROM OrderB WHERE amount < 2
==> Sink

actor system and runtime

- actor: JobManager
- actor: TaskManager
- actor: MemoryArchivist
- actor - yarn: ApplicationClient, YarnJobManager, YarnTaskManager
- actor - mesos: MesosJobManager, MesosTaskManager
- BlobClient, Scheduler, Slot, Task, Instance
- entry: StreamGraph, JobGraph, ExecutionGraph

Simple runtime of Flink Framework



The detail graph of Runtime will be shared next time

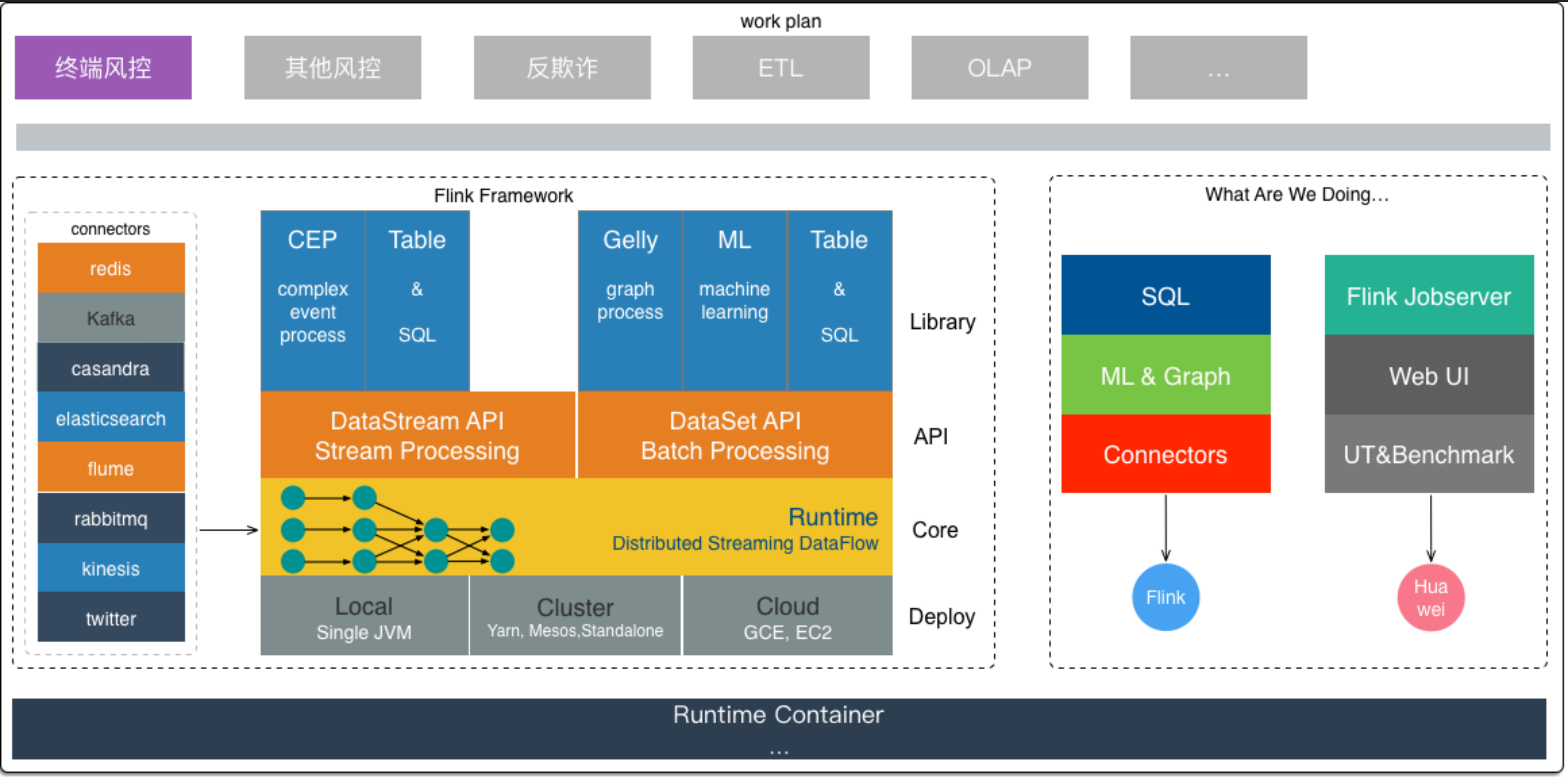
The problem of Flink

Problem of Flink

- user group increasing slowly
- java/scala/batch/streaming API fuzzy
- Gelly and FlinkML are un-matured
- code quality need enhance

What are we doing with Flink

What are we doing



work plan

终端风控

其他风控

反欺诈

ETL

OLAP

…

Flink Framework

connectors

redis

Kafka

casandra

elasticsearch

flume

rabbitmq

kinesis

twitter

CEP

complex event process

Table

&

SQL

Gelly

graph process

ML

machine learning

Table

&

SQL

Library

DataStream API
Stream Processing

DataSet API
Batch Processing

API

Runtime
Distributed Streaming DataFlow

Core

Local
Single JVM

Cluster
Yarn, Mesos,Standalone

Cloud
GCE, EC2

Deploy

What Are We Doing…

SQL

ML & Graph

Connectors

Flink

Flink Jobserver

Web UI

UT&Benchmark

Hua
wei

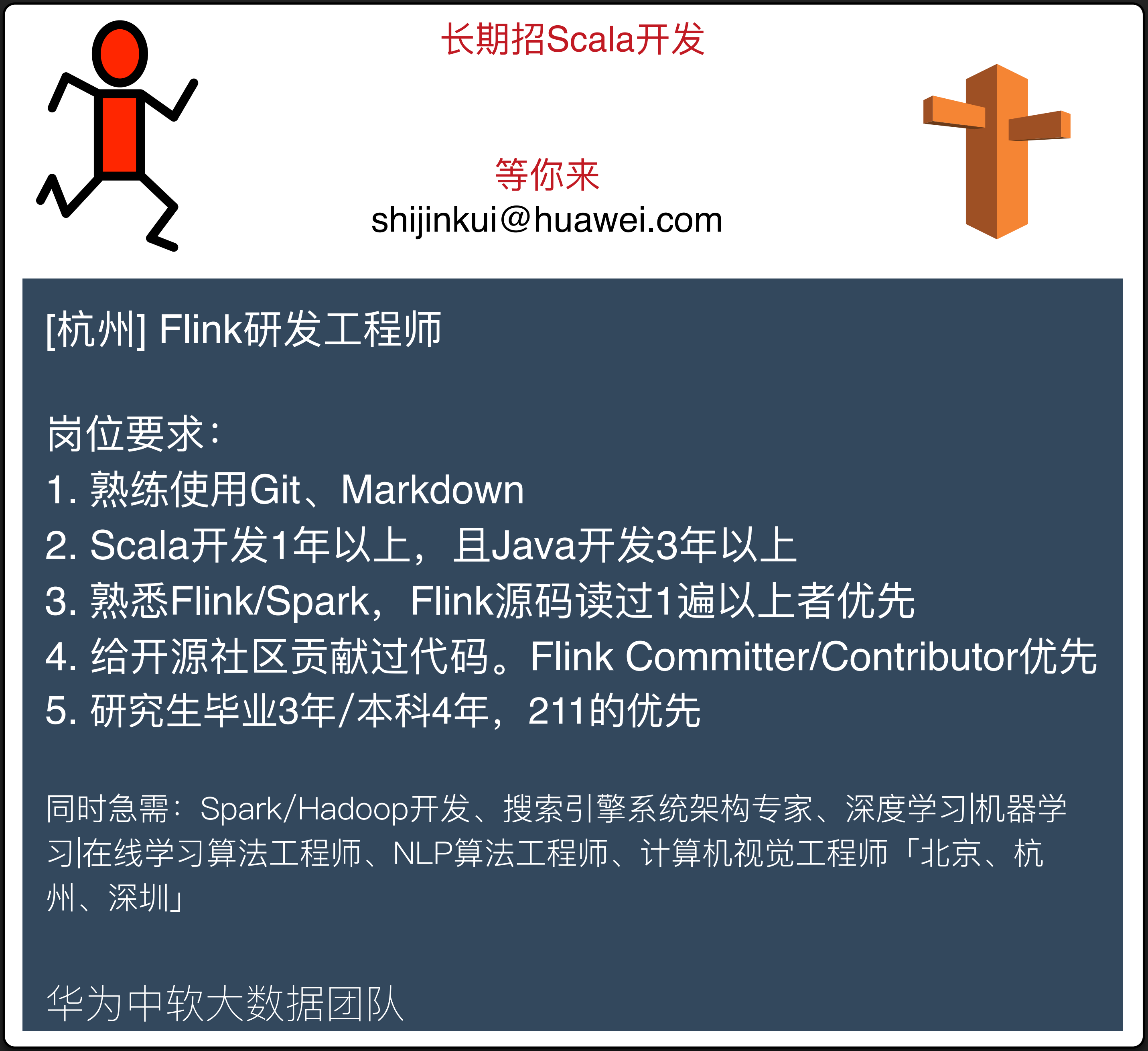
Runtime Container

…

Risk-management

- low latency: less than 100ms
- many security rules
- use Flink Stream API
- expect Flink SQL
- local cache and Redis
- develop Flink Jobserver and Netty(tcp/restful) Connector
- Apache Carbondata connector will be imported

Join us



长期招Scala开发

等你来
shijinkui@huawei.com

[杭州] Flink研发工程师

岗位要求：
1. 熟练使用Git、Markdown
2. Scala开发1年以上，且Java开发3年以上
3. 熟悉Flink/Spark，Flink源码读过1遍以上者优先
4. 给开源社区贡献过代码。Flink Committer/Contributor优先
5. 研究生毕业3年/本科4年，211的优先

同时急需：Spark/Hadoop开发、搜索引擎系统架构专家、深度学习|机器学习|在线学习算法工程师、NLP算法工程师、计算机视觉工程师「北京、杭州、深圳」

华为中软大数据团队

mail to: shijinkui@huawei.com

THANKS