Build a Table-centric Apache Flink Ecosystem

Shaoxuan Wang
Director, Senior Staff Engineer at Alibaba

2019 Flink Forward San Francisco

Shaoxuan Wang
Director, Senior Staff Engineer at Alibaba

shaoxuan.wsx@alibaba-inc.com
shaoxuan@apache.org

Peking University

EECS

University of California at San Diego

PhD, Computer Engineer

Broadcom

High-Perf Platform
Senior Software Engineer

Facebook

Core Infra
Software Engineer

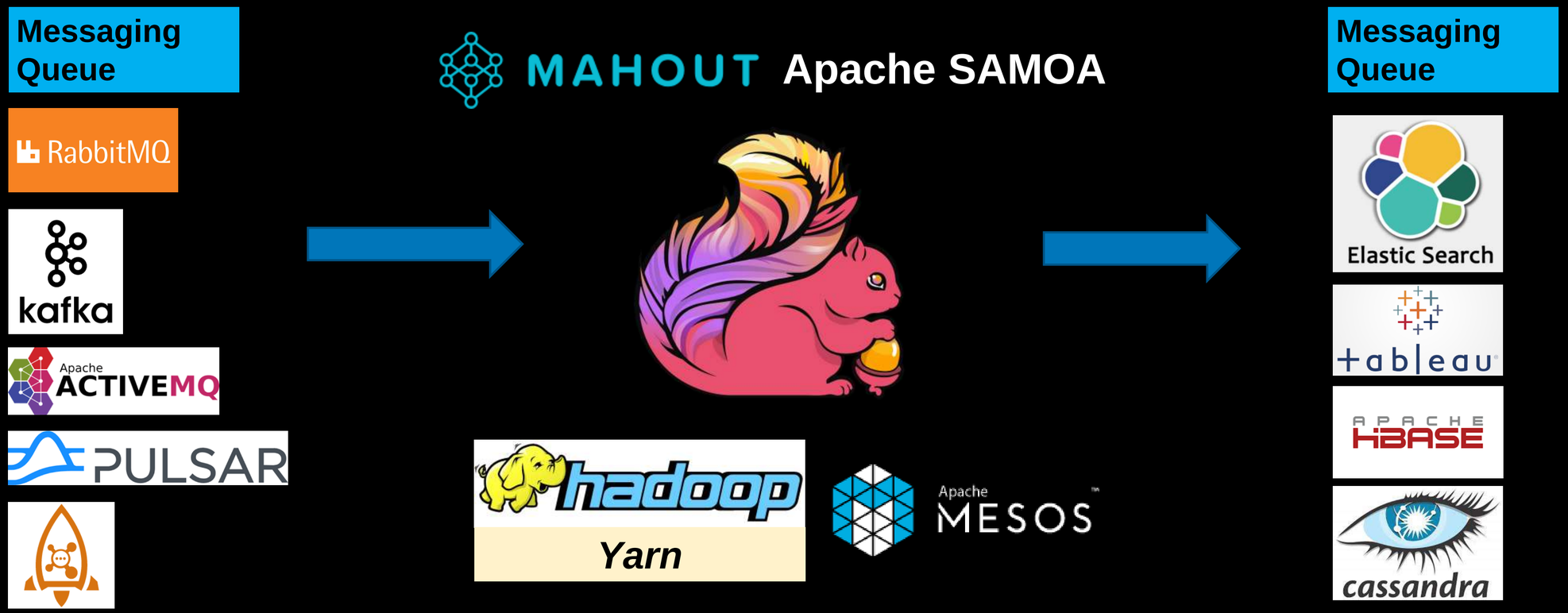
Alibaba Group

Data Infra / Cloud
Senior Staff Engineer

Flink Committer

Since 2017

Apache Flink Ecosystem - Present

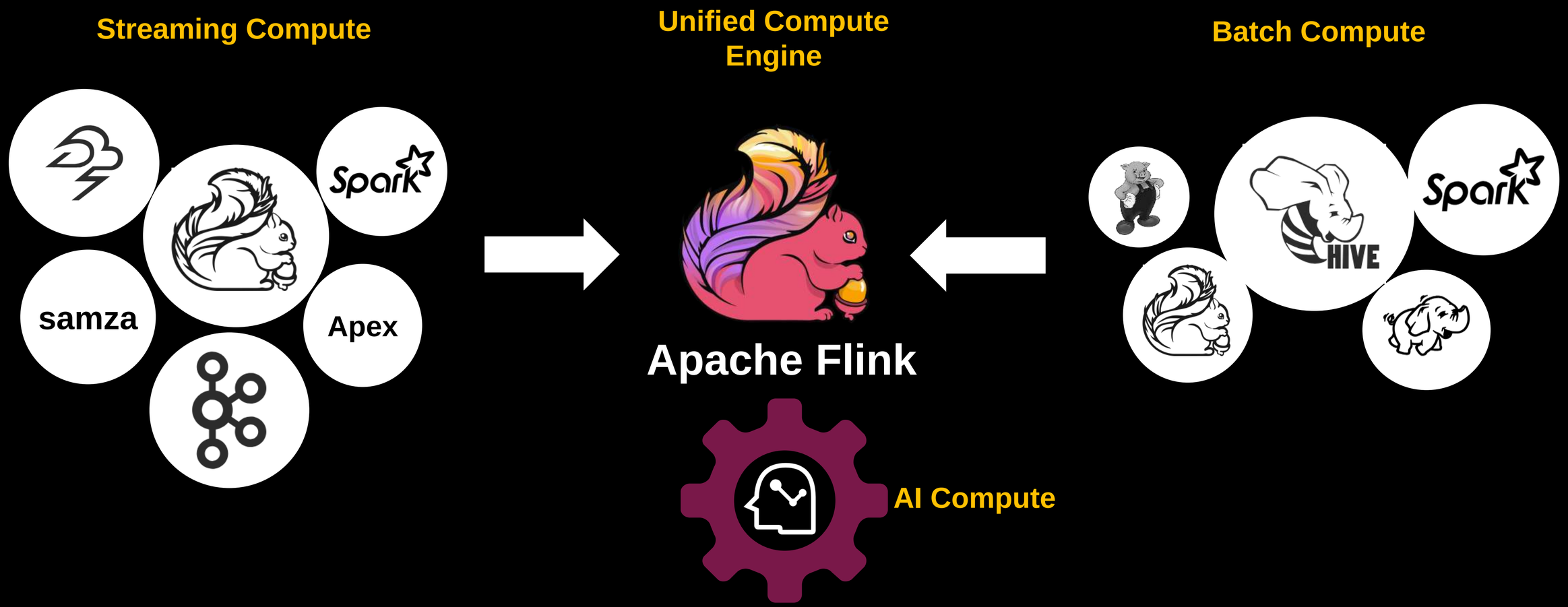


Apache Flink is the most sophisticated open-source Stream Processor

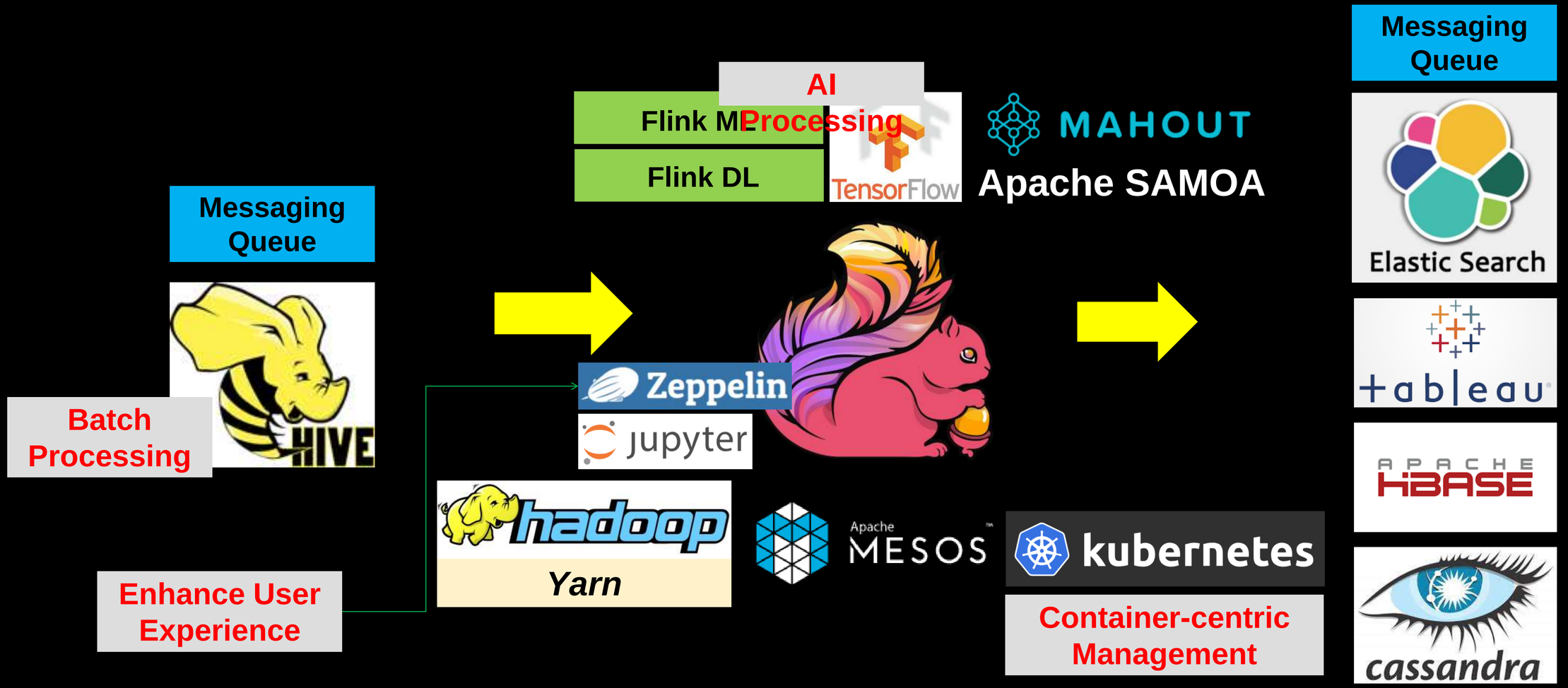
Intelligent Big Data Computing

Can Apache Flink become an unified engine for intelligent big data computing?

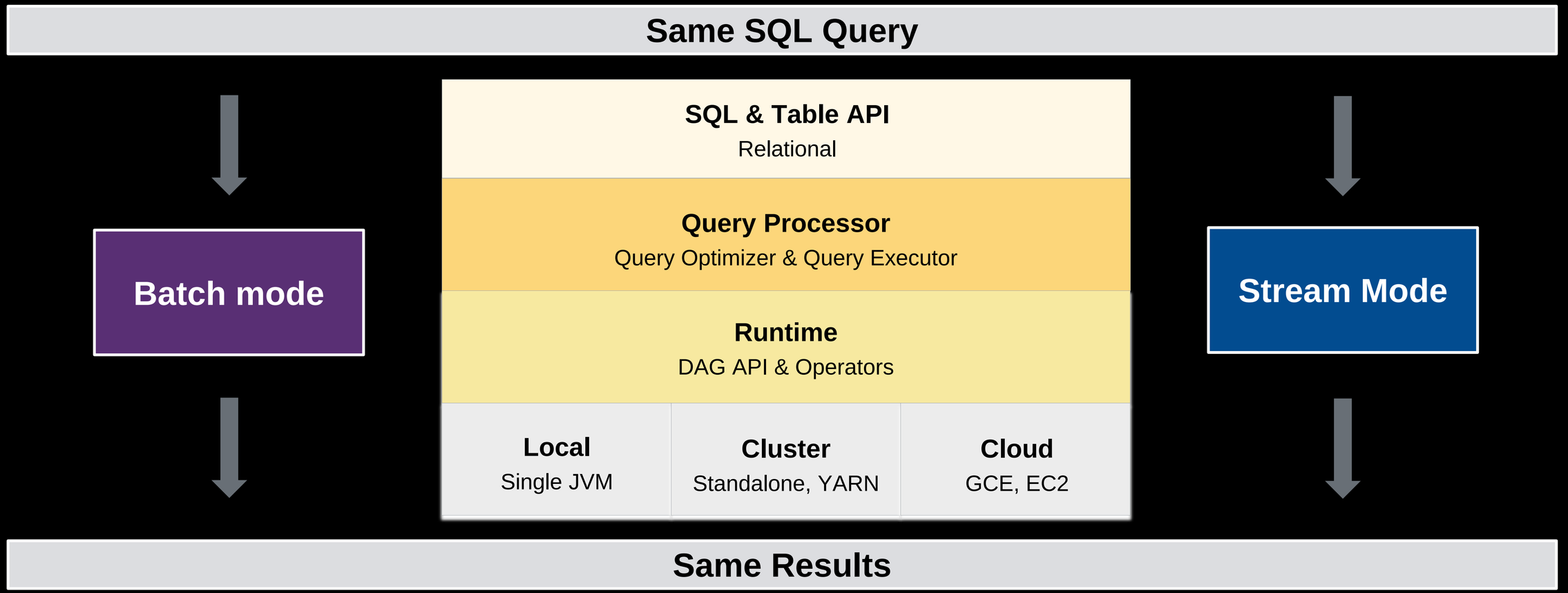
Build Intelligent Big Data Platform with Apache Flink



Apache Flink Ecosystem - Future



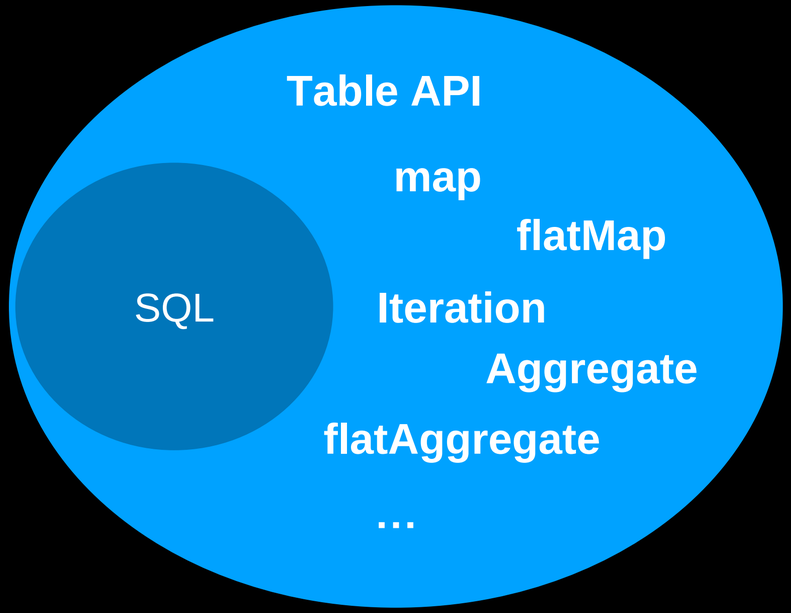
TableAPI: declaritive API with nature Optimization



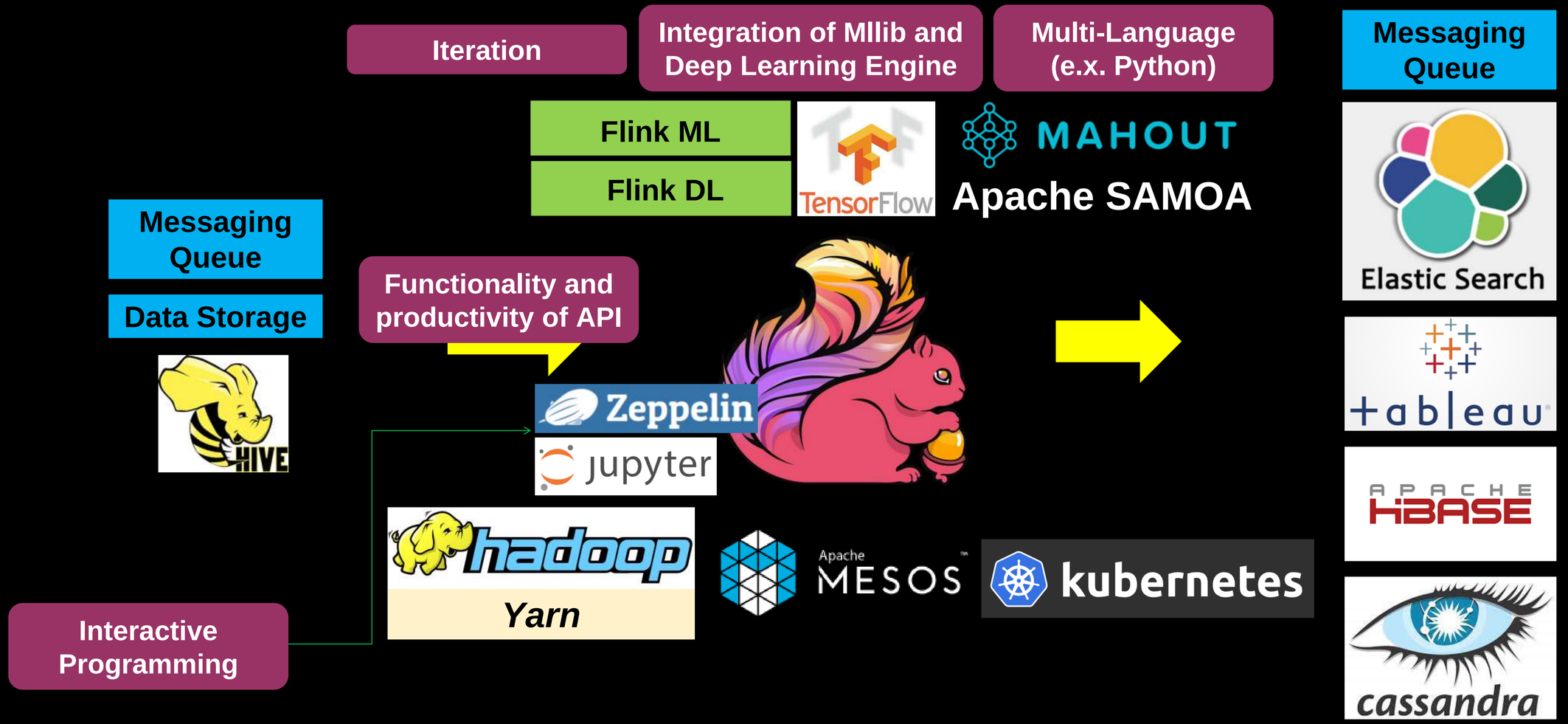
similar as SQL, batch&stream unified, declarative API with nature optimization framework

TableAPI is more than SQL

	TableAPI	SQL	e.g.
Stream and batch unified	Y	Y	SELECT/AGG/WINDOW etc.
Functional scalability	Y	N	flatAgg/Column operations etc.
Rich expression	Y	N	map/flatMap/intersect etc.
Compile check	Y	N	Java/Scala IDE



What Else are Needed on TableAPI



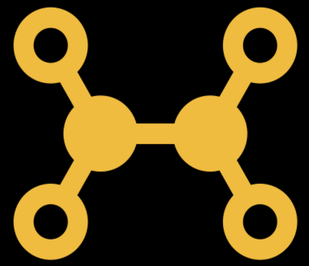
What Else are Needed on TableAPI

Functionality and Productivity Enhancement

Interactive programming

Multi-language

Iteration

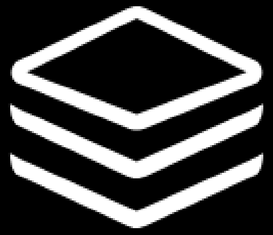


Execute MLlib and DL engine

Agenda

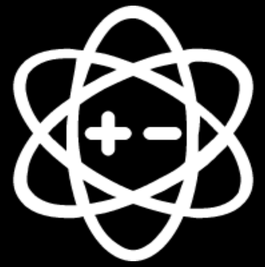
- **TableAPI enhancement: functionality and productivity**
- **Interactive programming**
- **Multi-language support**
- **Brief introduction for AI support**

TableAPI Enhancement



Row-based data processing API

Functionality, productivity



Column operation

Productivity

Hint

optimization instruction, Resource configuration

Introduce Row-based TableAPI (FLIP29)

UDF	Single Row Input	Multiple Row Input
Single Row Output	ScalarFunction	AggregateFunction
Multiple Row Output	TableFunction	*TableAggregateFunction (new)*

Table Method	Single Column Output	Multiple Column Output
ScalarFunction	Table.select	*Table.map*
AggregateFunction	Table.select	*GroupedTable.agg*
TableFunction	N/A	*Table.flatmap*
TableAggregateFunction	N/A	*GroupedTable.flatagg*

Map Operator in Table API

Method signature

def map(scalaFunction: Expression): Table

Pros

table.select(udf(‘c1), udf(’c2), udf(’c3))
VS
table.map(udf(’c1,’c2,’c3))

Example

```
val res = tab
   .map(fun(‘e)).as(‘a, ‘b, ‘c)
   .select(‘a, ‘c)
```

```
class MyMap extends ScalarFunction {
  var param : String = ""

  override def open(context: FunctionContext): Unit
   =  param = context.getJobParameter("paramKey","")

  def eval(index: Int): Row = {
    val result = new Row(3)
    // Business processing based on data and parameters
    result
  }

  override def getResultType(signature: Array[Class[_]]): TypeInformation[_]
= {
    Types.ROW(Types.STRING, Types.INT, Types.LONG)
  }
}
```

INPUT(Row)	OUTPUT(Row)
1	1

FatMap Operator in Table API

Method signature

def flatMap(tableFunction: Expression): Table

Pros

table.join(udtf) VS table.flatMap(udtf())

Example

```
val res = tab
   .flatMap(fun(‘e,’f)).as(‘name, ‘age)
   .select(‘name, ‘age)
```

```
case class User(name: String, age: Int)
class MyFlatMap extends TableFunction[User] {
  def eval(user: String): Unit = {
    if (user.contains("#")) {
      val splits = user.split("#")
      collect(User(splits(0),splits(1).toInt))
    }
  }
}
```

INPUT(Row)	OUTPUT(Row)
1	N(N>=0)

Aggregate Operator in Table API

Method signature

def aggregate(aggregateFunction: Expression): AggregatedTable
class AggregatedTable(table: Table, groupKeys: Seq[Expression], aggFunction: Expression)

Pros

table.select(agg1(), agg2(), agg3()….)
VS
table.aggregate(agg())

Example

```
val res = groupedTab
    .groupBy(‘a)
    .aggregate(
      aggFun(‘e,’f) as (‘a, ‘b, ‘c))
    .select(‘a, ‘c)
```

```
class CountAccumulator extends JTuple1[Long] {
  f0 = 0L //count
}
class CountAgg extends AggregateFunction[JLong, CountAccumulator] {
  def accumulate(acc: CountAccumulator): Unit = {
    acc.f0 += 1L
  }

  override def getValue(acc: CountAccumulator): JLong = {
    acc.f0
  }
  ... retract()/merge()
}
```

INPUT(Row)	OUTPUT(Row)
N(N>=0)	1

FlatAggregate Operator in Table API

Method signature

def flatAggregate(tableAggregateFunction: Expression): GroupedFlatAggregateTable
class GroupedFlatAggTable(table: Table, groupKey: Seq[Expression], tableAggFun: Expression)

Pros

A completely new feature

Example

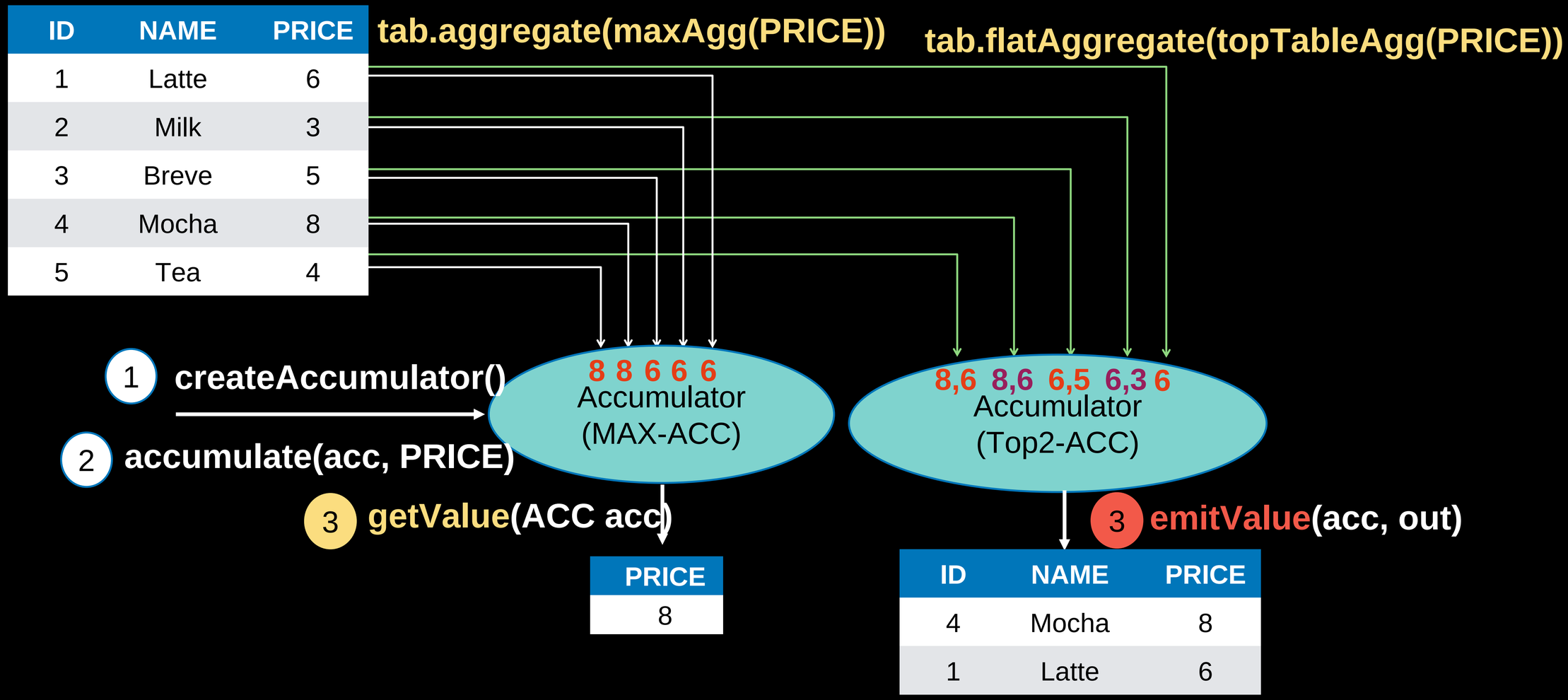
```
val res = groupedTab
    .groupBy(‘a)
    .faltAggregate(
        flatAggFun(‘e,’f) as (‘a, ‘b, ‘c))
    .select(‘a, ‘c)
```

```
class TopNAcc {
  var data: MapView[JInt, JLong] = _  // (rank -> value)
  ...
}
class TopN(n: Int) extends TableAggregateFunction[(Int, Long), TopNAccum] {

  def accumulate(acc: TopNAcc, category: Int, value: Long) {
    ...
  }
   def emitValue(acc: TopNAcc, out: Collector[(Int, Long)]): Unit = {
    ...
  }
  ...retract/merge
}
```

INPUT(Row)	OUTPUT(Row)
N(N>=0)	M(M>=0)

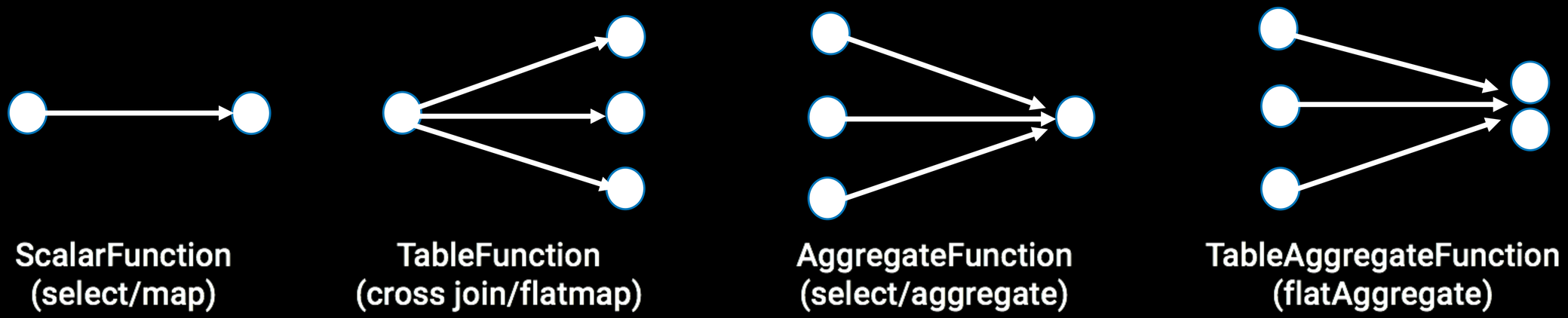
Examples of Aggregate and FlatAggregate

Examples of Max (aggregate) and Top2 (flatAggregate)



Summary

	Single Row Input	Multiple Row Input
Single Row Output	ScalarFunction (select/map)	AggregateFunction (select/aggregate)
Multiple Row Output	TableFunction (cross join/flatmap)	TableAggregateFunction (flatAggregate)



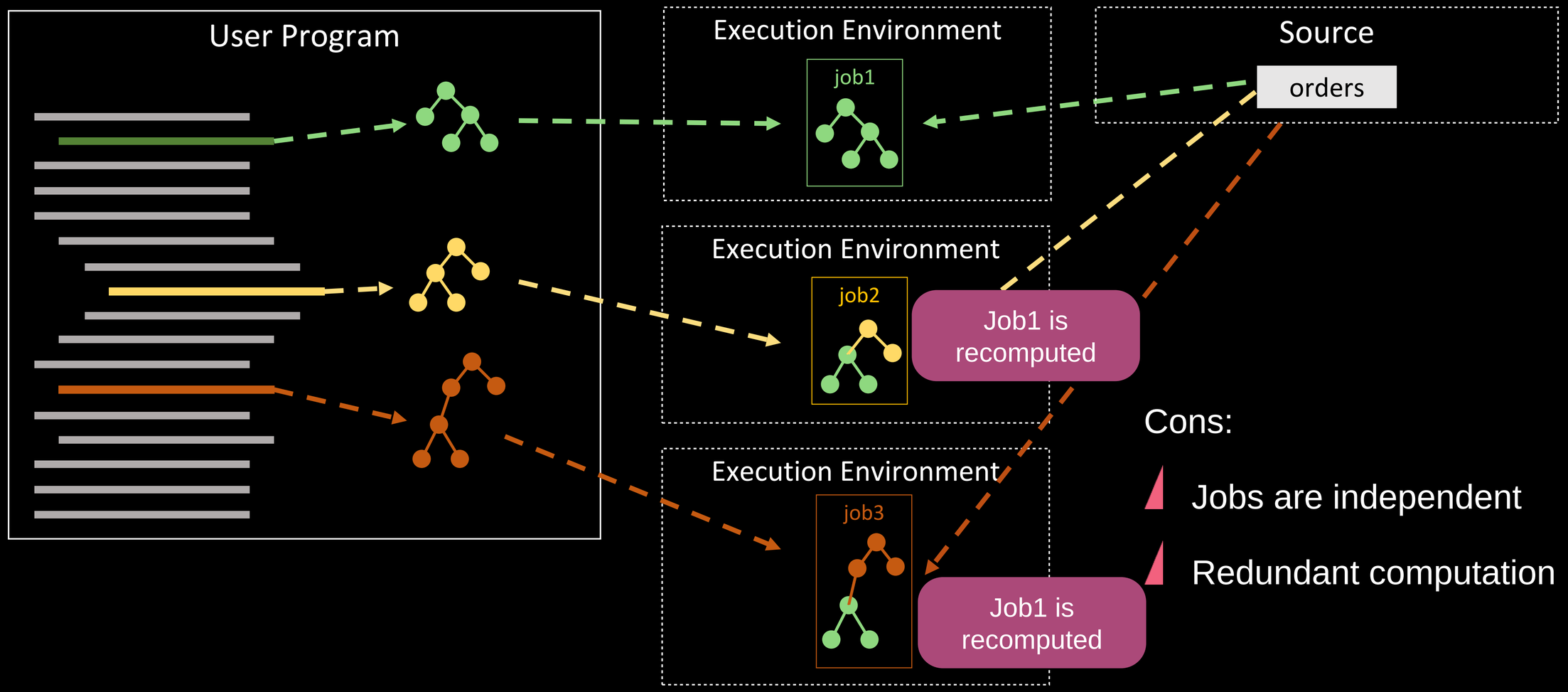
Agenda

- TableAPI enhancement: functionality and productivity
- Interactive programming
- Multi-language support
- Brief introduction of AI support

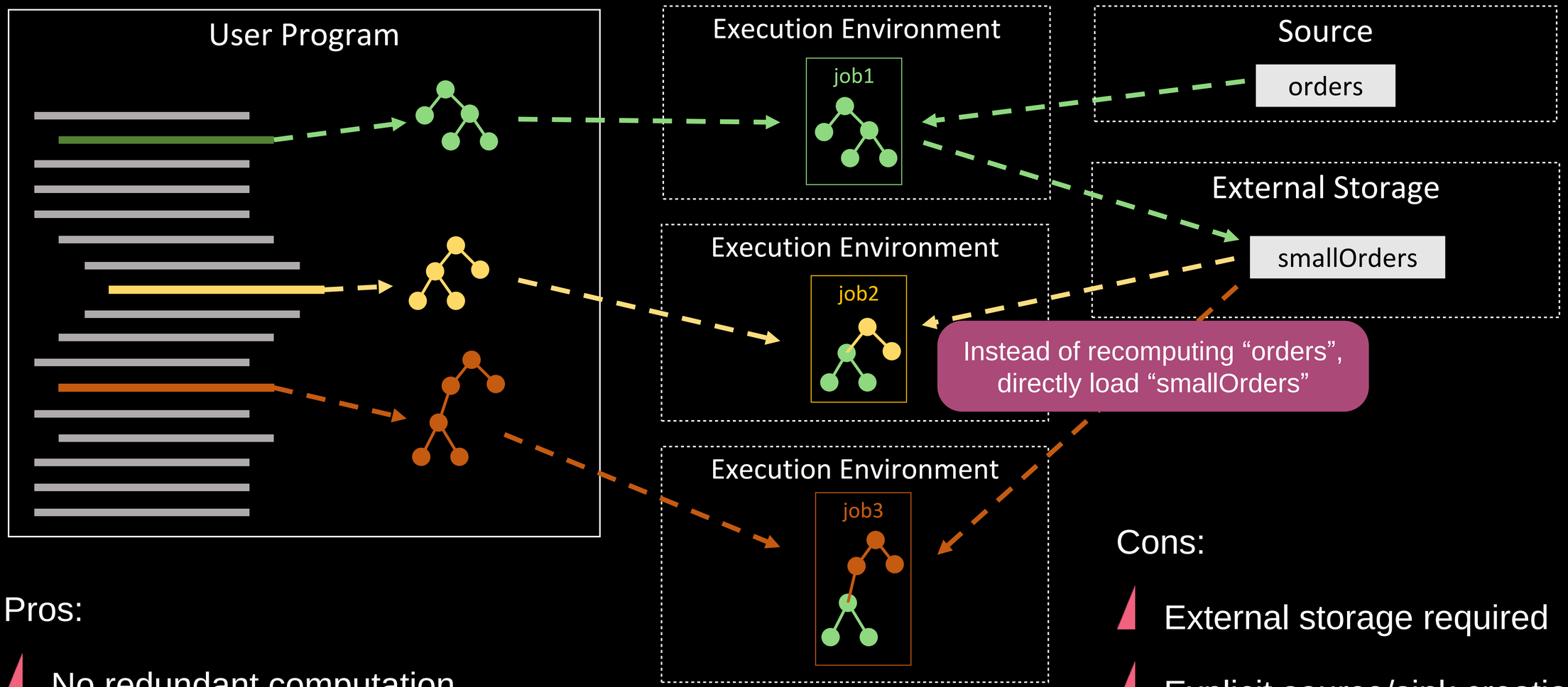
A example code snippet

```
{
  val orders = tEnv.fromCollection(data).as ('country, 'color, 'quantity)

  val smallOrders = orders.filter('quantity < 100)

  val countriesOfSmallOrders = smallOrders.select('country).distinct()
  countriesOfSmallOrders.print()

  val smallOrdersByColor = smallOrders.groupBy('color).select('color, 'quantity.avg as 'avg)
  smallOrdersByColor.print()
}
```

Interactive Programming without Cache or External Storage



Interactive Programming with External Storage



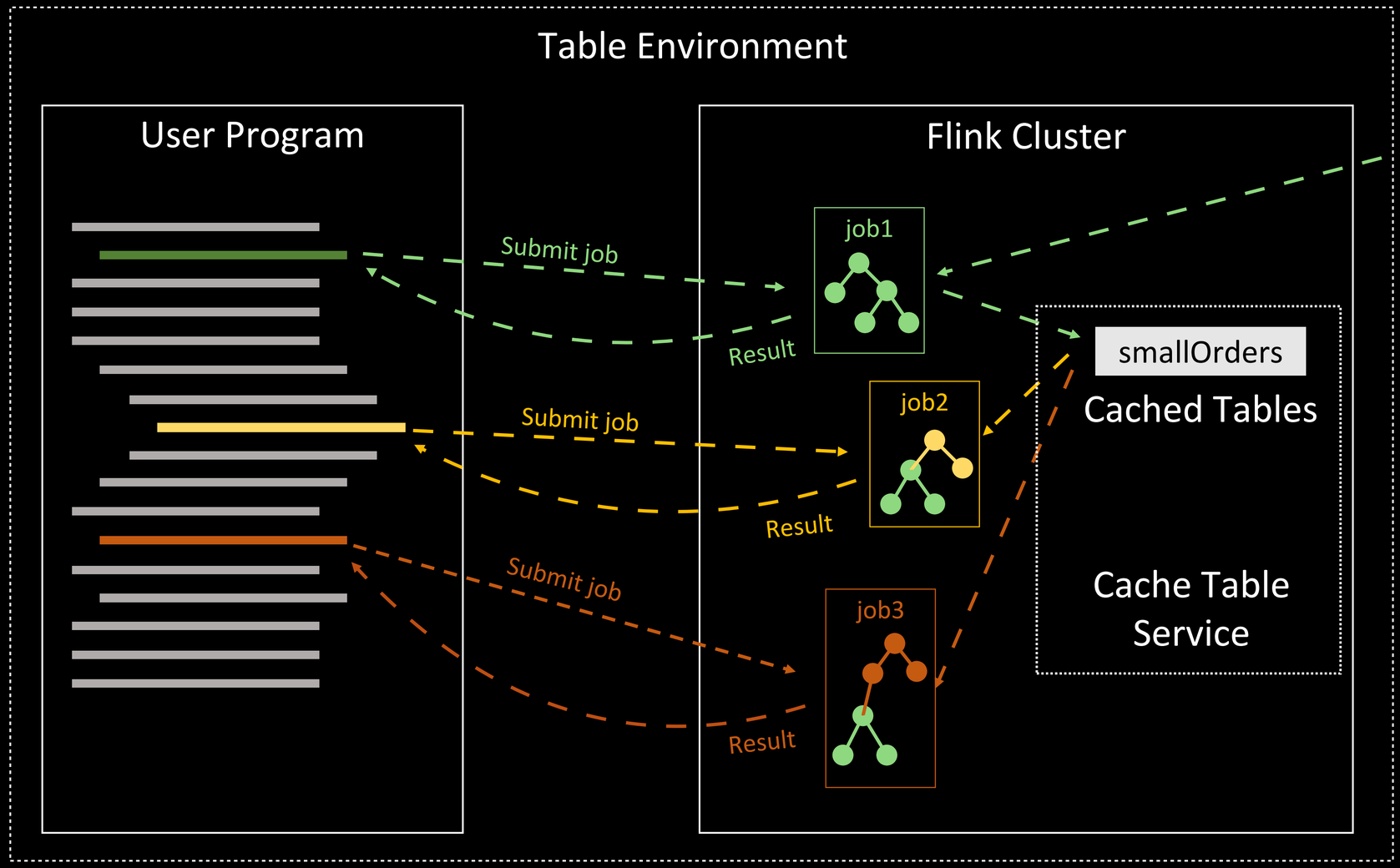
Pros:

- No redundant computation

Cons:

- External storage required
- Explicit source/sink creation

Interactive Programming with Cached Result (FLIP36)



Pros:

- Jobs in the same context
- No external storage
- Better performance

Introducing Table.cache() for Interactive Programming

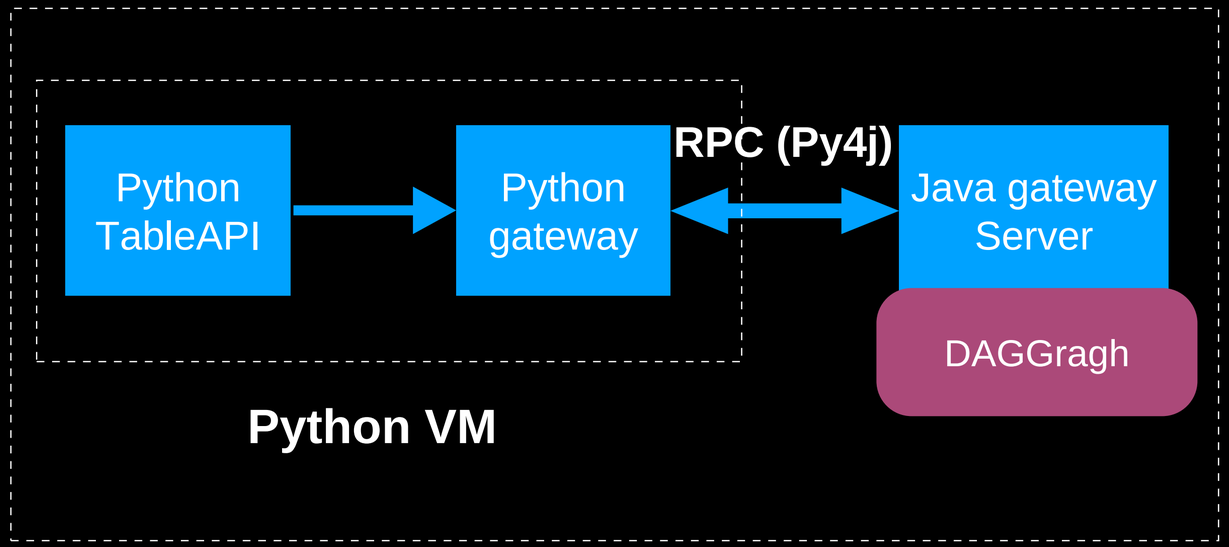
```
{
  val orders = tEnv.fromCollection(data).as ('country, 'color, 'quantity)

  val smallOrders = orders.filter('quantity < 100)
  smallOrders.cache()

  val countriesOfSmallOrders = smallOrders.select('country).distinct()
  countriesOfSmallOrders.print()

  val smallOrdersByColor = smallOrders.groupBy('color).select('color, 'quantity.avg as 'avg)
  smallOrdersByColor.print()
}
```

Agenda

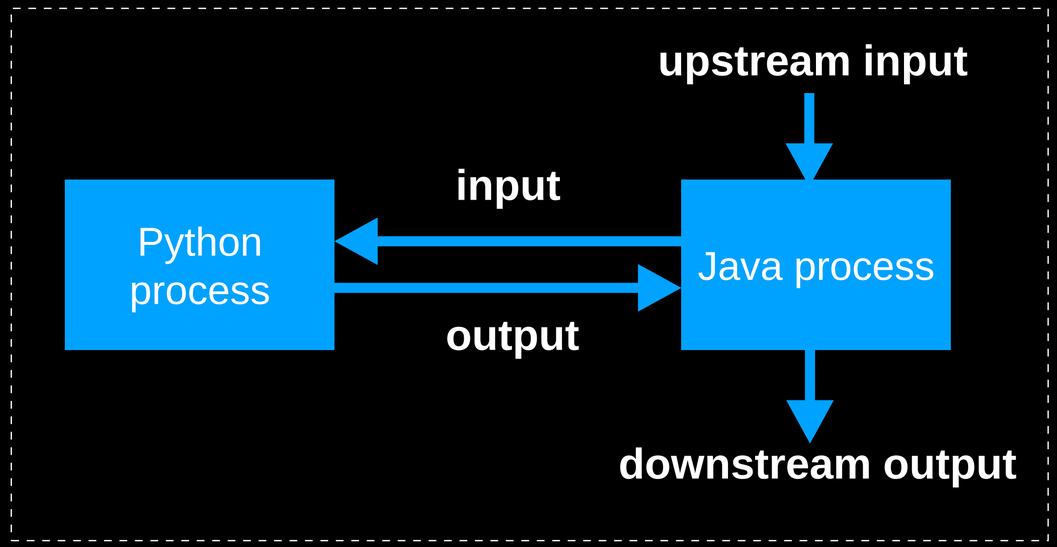
- TableAPI enhancement: functionality and productivity
- Interactive programming
- Multi-language support
- Brief introduction of AI support

Flink Python TableAPI

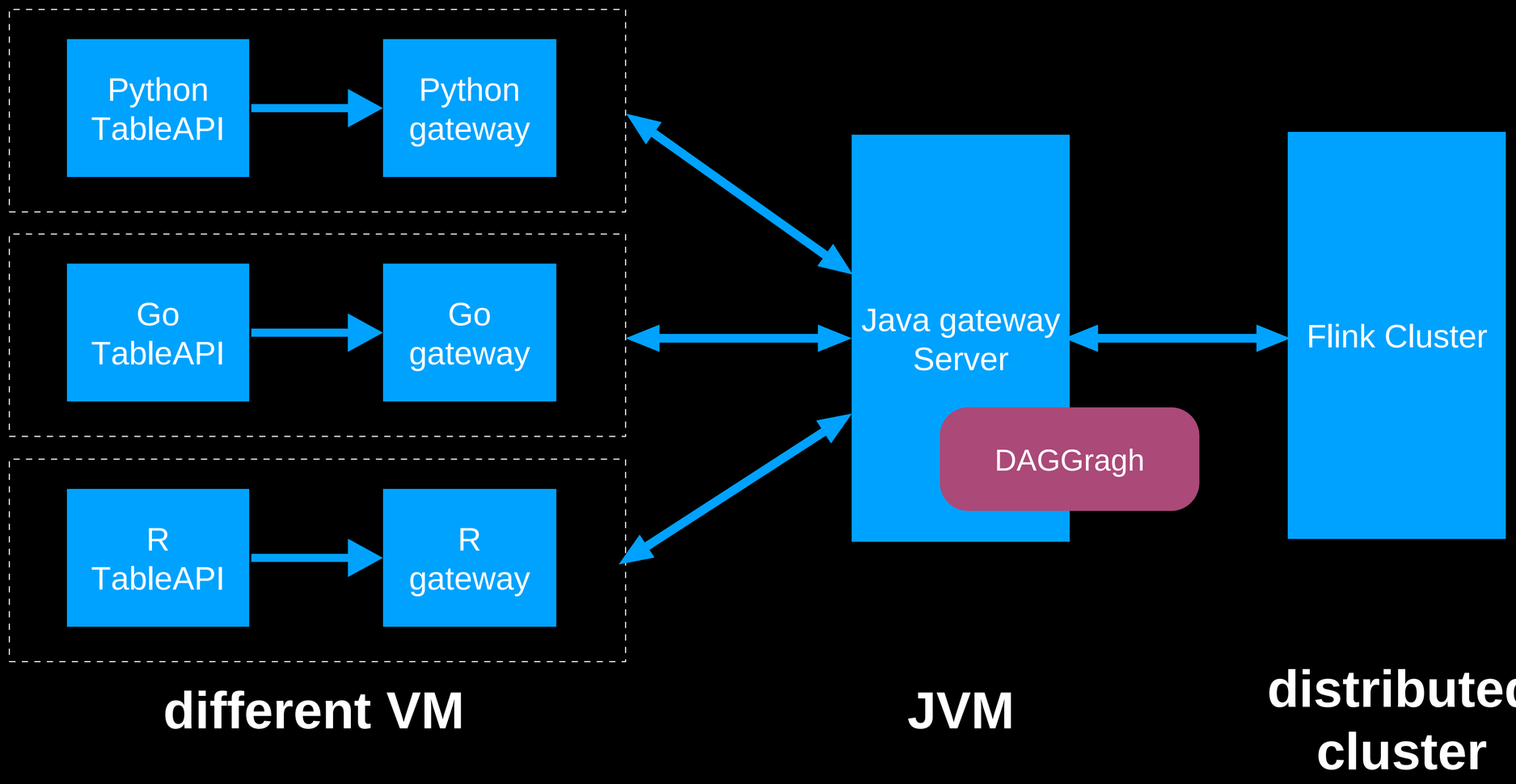
Python TableAPI



Python UDF



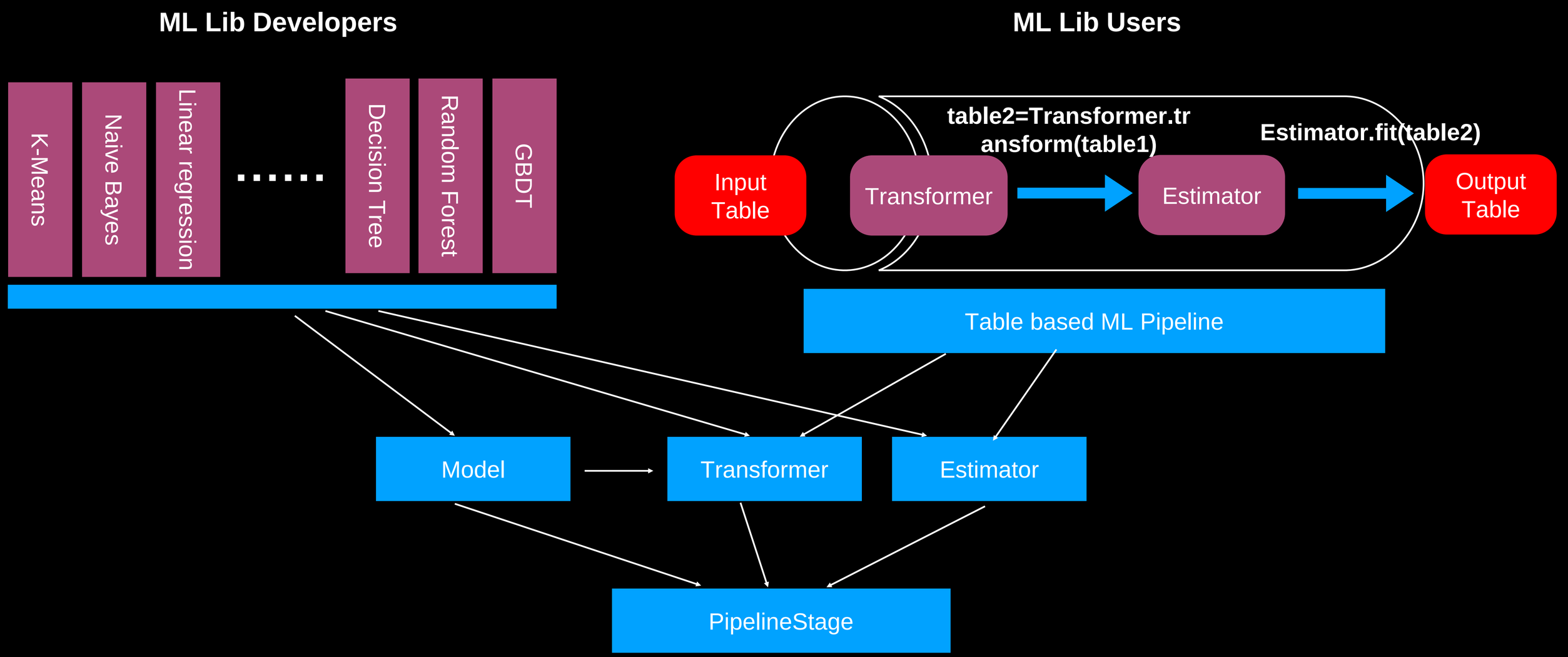
Multi-Language for Flink TableAPI



Agenda

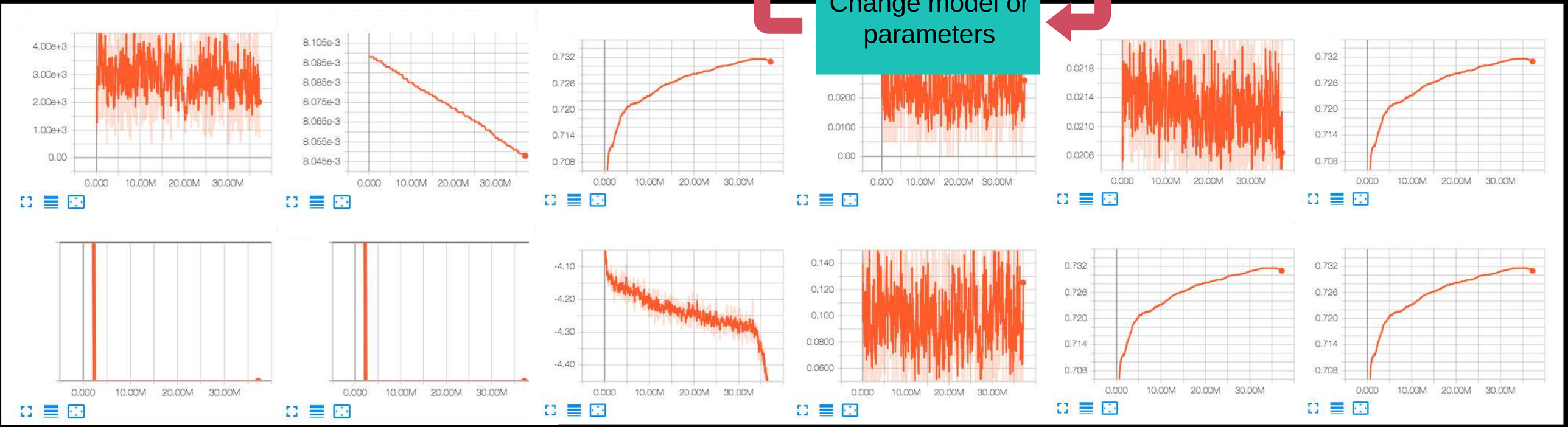
- TableAPI enhancement: functionality and productivity
- Interactive programming
- Multi-language support
- Brief introduction of AI support

ML Pipeline - Overview (Target for Flink 1.9)



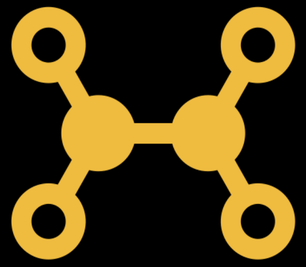
Deep Learning Pipeline on Flink

Change model or parameters



Find More Details in Another Session

Iteration



Execute MLlib and DL engine

When Table meets AI: Build Flink AI Ecosystem on Table API

Shaoxuan Wang,

Director, Senior Staff Engineer at Alibaba

4:30pm - 5:10pm

Nikko II & III

Summary

- TableAPI enhancement: functionality and productivity (FLIP29 etc.)
- Interactive programming (FLIP36, Flink1.9)
- Multi-language support (Python TableAPI, Flink1.9)
- Flink ML pipeline (Flink1.9)
- Deep learning pipeline on Flink (Will be open-sourced)

Welcome to Flink Forward - ASIA

Dec. 2019 @ Beijing

flink-forward-china@list.alibaba-inc.com

Q & A