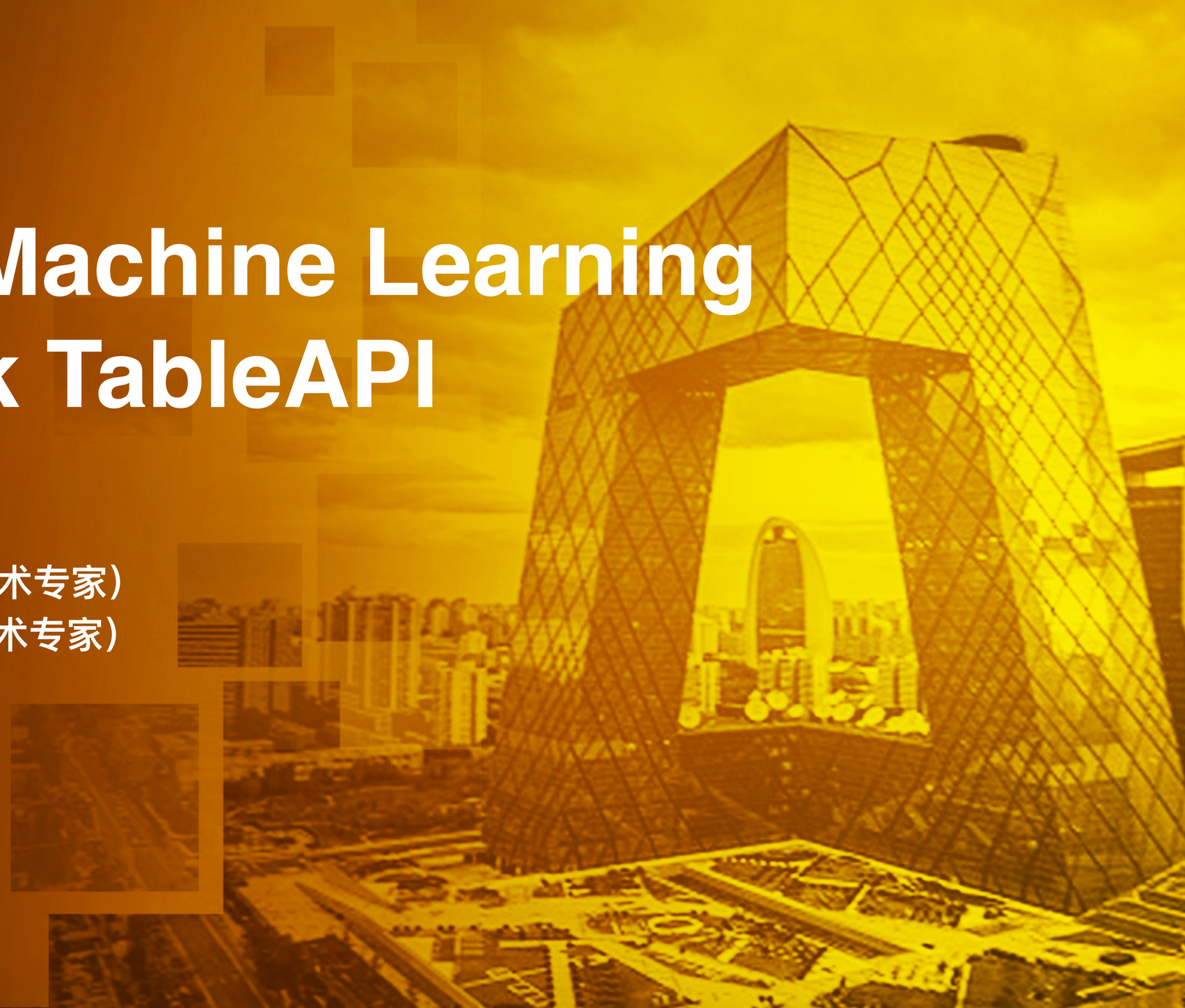
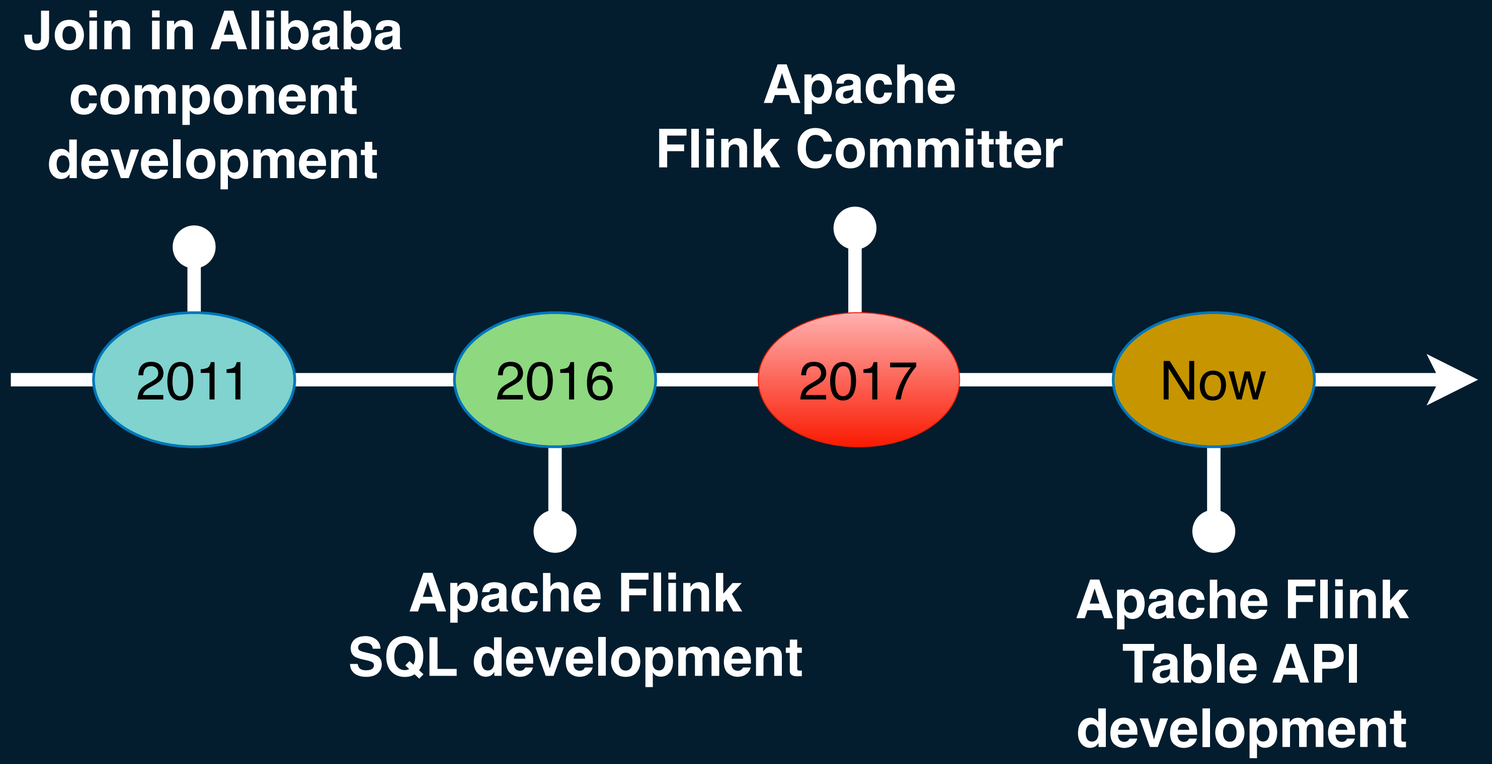
Simplify Machine Learning With Flink TableAPI

公司：阿里巴巴
演讲者：孙金城 （高级技术专家）
秦江杰 （高级技术专家）



About us

Jincheng Sun(JinZhu)
孙金城(金竹)



About us

Jiangjie (Becket) Qin
秦江杰

Alibaba Realtime Computing Platform, 2018.5 -
2018年5月加入阿里巴巴实时计算平台

Worked at LinkedIn and IBM
曾就职于 LinkedIn 和 IBM

M.S Information Networking, CMU, 2014
2014年硕士毕业于卡耐基梅陇大学

Apache Kafka PMC Member
Apache Kafka 项目管理委员会成员

AGENDA
提要

Brief introduction to Flink Table API
Table API 简介

The API requirements from Machine Learning Algorithms
机器学习算法对 API 的核心需求

The enhancement to Flink Table API
Flink Table API 的扩展

Algorithm examples based on Flink Table API
基于 Flink Table API 的算法实现

AGENDA
提要

Brief introduction to Flink Table API
Table API 简介

The API requirements from Machine Learning Algorithms
机器学习算法对 API 的核心需求

The enhancement to Flink Table API
Flink Table API 的扩展

Algorithm examples based on Flink Table API
基于 Flink Table API 的算法实现

Table API is high-level analytics API

FLINK FORWARD

Apache Flink Layered APIs

High-level Analytics API	SQL / Table API (dynamic tables)	Descriptive API/描述性API With Optimization/基于优化
Stream- & Batch Data Processing	DataStream API (streams, windows)	Richer Processing API/更丰富的API
Stateful Event-Driven Applications	ProcessFunction (events, state, time)	Raw Processing API/底层API

− Conciseness +

+ Expressiveness −

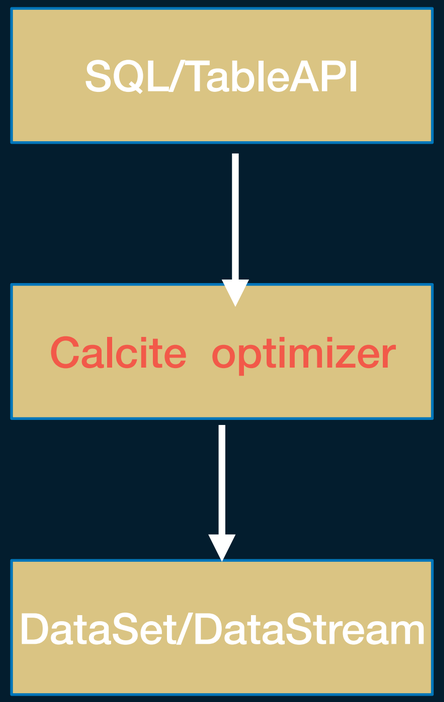
E.g.: Count the numbers of people by Region
示例：按地区统计人口数量

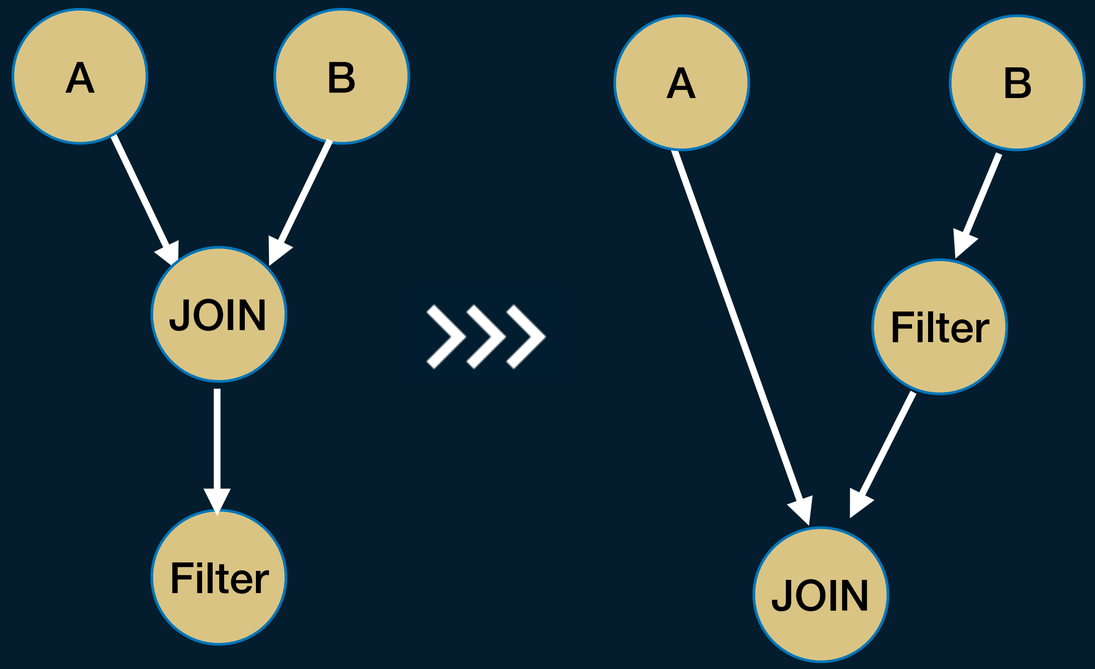
```
table
    .groupBy('region)
    .select('region, COUNT(1))
```

```
def main(args: Array[String]): Unit = {
  val env = StreamExecutionEnvironment.getExecutionEnvironment
  env.setParallelism(1)

  val dataStream = env.addSource(new WordCountSourceFunction)
  val tokenizerWord = dataStream.flatMap(new TokenizerFlatMap)
  val result = tokenizerWord
    .map{(_, 1)}
    .keyBy( fields = 0)
    .sum( position = 1)

  result.addSink(new PrintSinkFunction[(String, Int)])
  env.execute()
}

class TokenizerFlatMap extends FlatMapFunction[String, String] {
  override def flatMap(
    t: String,
    out: Collector[String]): Unit = {
    val stringTokenizer = new StringTokenizer(t,  delim = ",!' .;",  returnDelims = false)
    while(stringTokenizer.hasMoreElements()) {
      val word = stringTokenizer.nextToken.trim.toLowerCase
      out.collect(word)
    }
  }
}
```

10 +

TableAPI is optimized



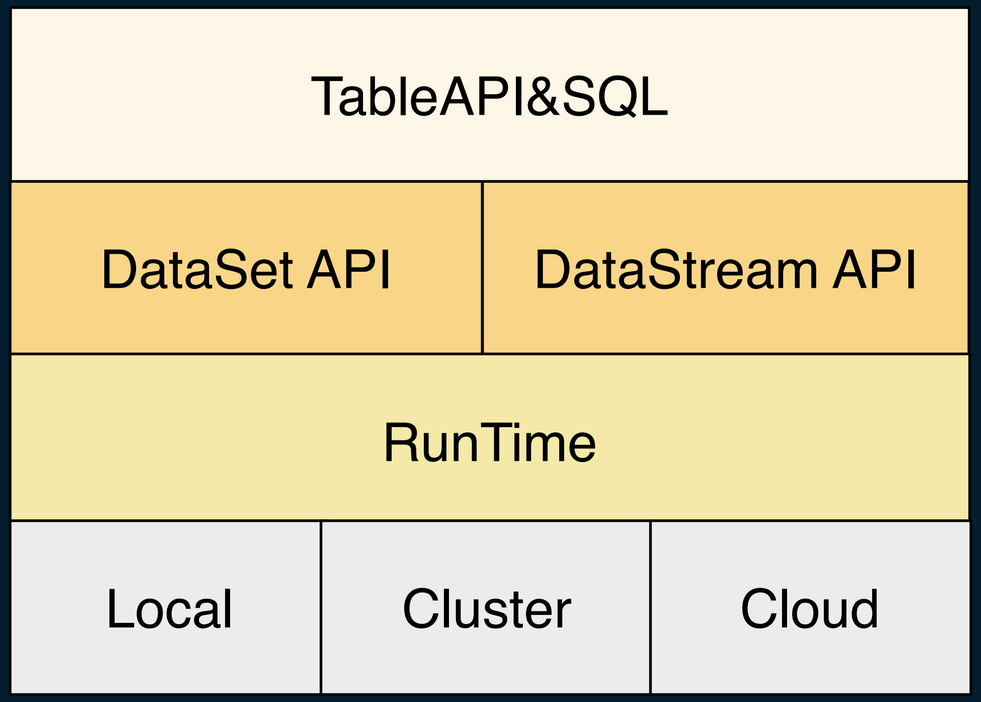


Get better execution performance
获取更好执行性能

TableAPI unifies batch and stream

FLINK FORWARD

Component Stack
组件栈




```
table
    .groupBy('region)
    .select('region,  COUNT(1))
```

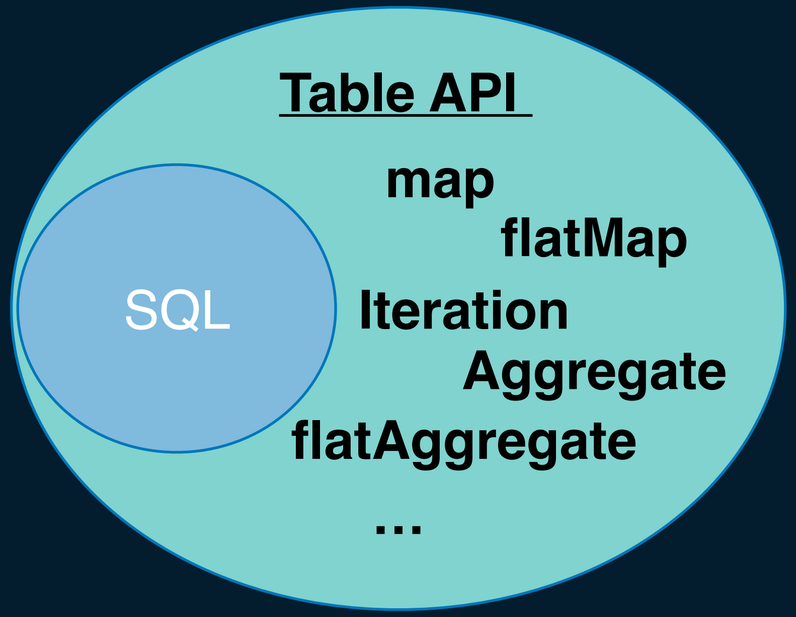
For Bath and Streaming
支持流批两种运行模式

Table API is a super set of the SQL

FLINK FORWARD

Table API = SQL + … Specially designed by Apache Flink

	TableAPI	SQL	e.g.
Stream and batch unified 流批统一	Y	Y	SELECT/AGG/ WINDOW etc.
Functional scalability 功能扩展性	Y	N	FlatAGG/Iteration/ Column operations etc.
Expressive extensibility 表达方式扩展性	Y	N	map/flatMap/ Row.flatten()/minus/ intersect etc.
Compile check 编译检查	Y	N	IDE Java/Scala

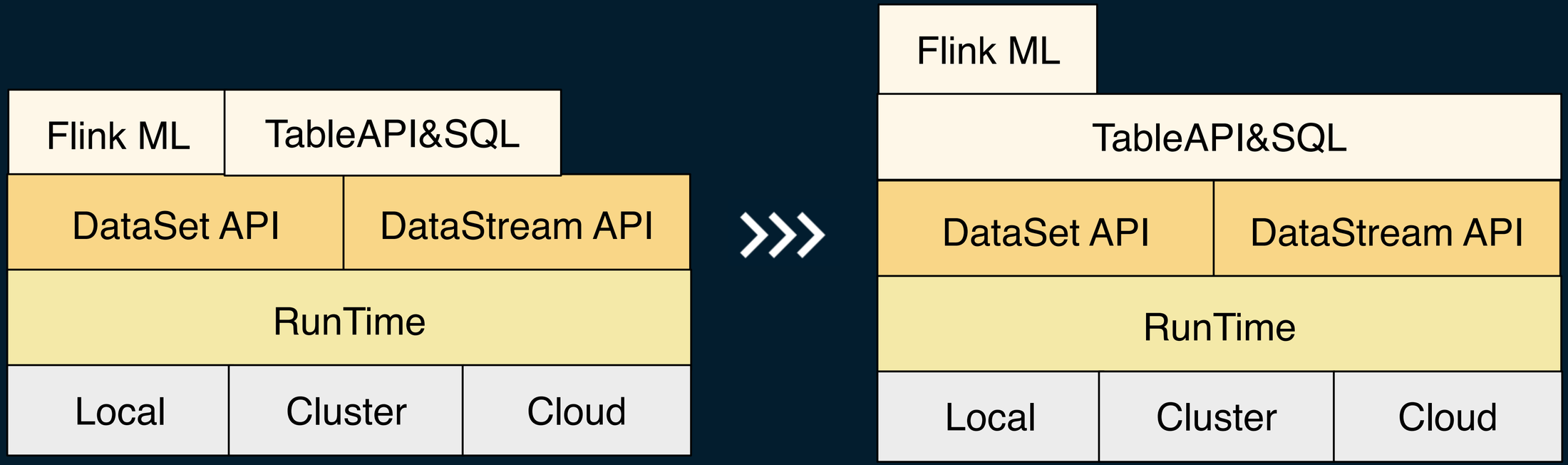


Summarize what is tableAPI

- **Stream and batch unified**
  流批统一
- **With optimization**
  可被优化
- **Concise and easy to use**
  简洁易用

So，How about Flink ML on tableAPI?
那么可以基于TableAPI开发Flink ML吗?

TableAPI may unify the implementation of Flink ML

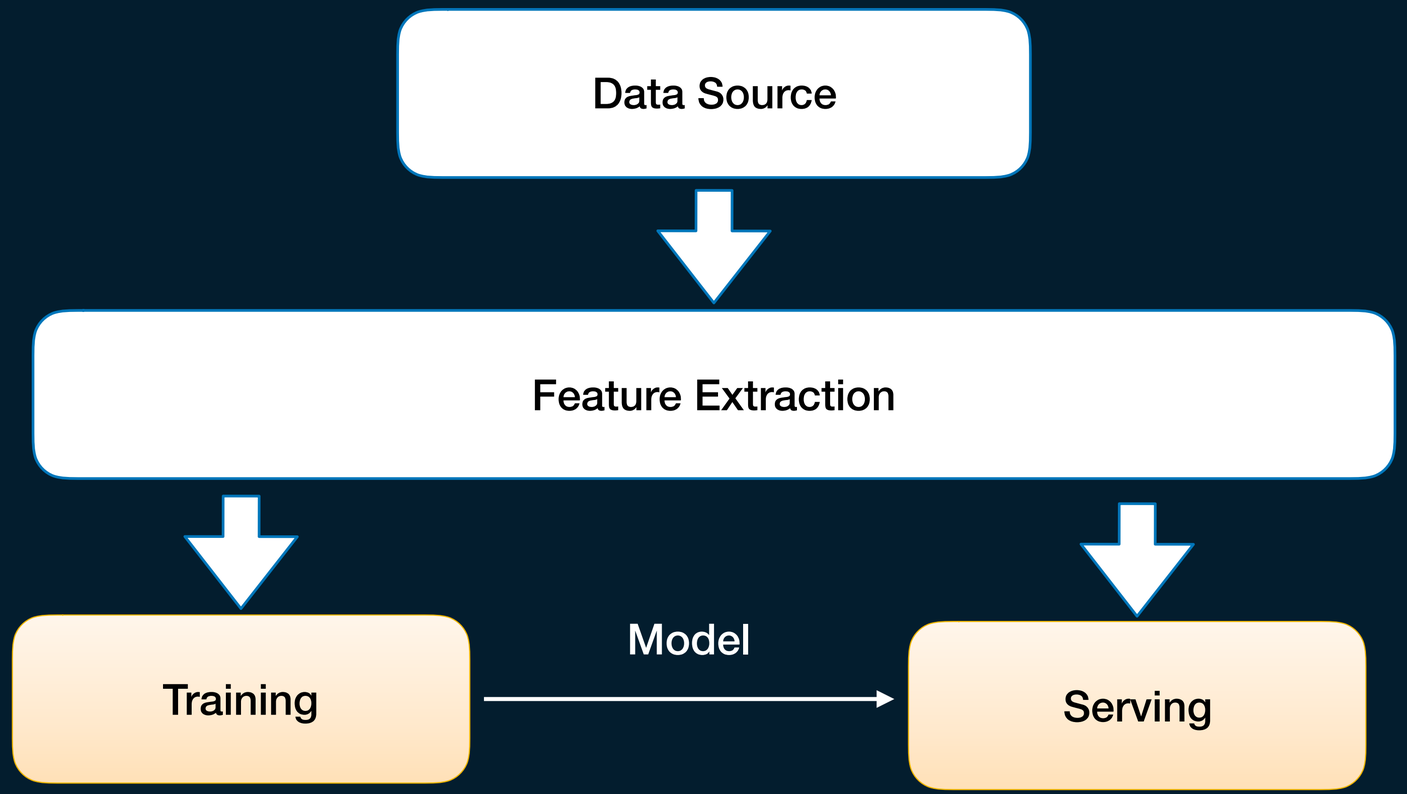


What are the requirements of Flink ML for the Table API?
Flink 机器学习对TableAPI有哪些需求呢?

AGENDA
提要

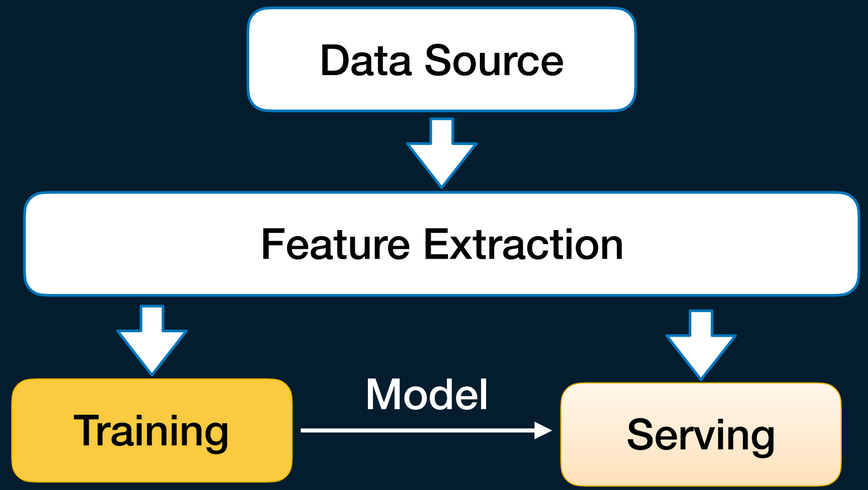
- Brief introduction to Flink Table API
  Table API 简介
- **The API requirements from Machine Learning Algorithms**
  机器学习算法对 API 的核心需求
- **The enhancement to Flink Table API**
  Flink Table API 的扩展
- **Algorithm examples based on Flink Table API**
  基于 Flink Table API 的算法实现

Core API requirements from ML



Model Training

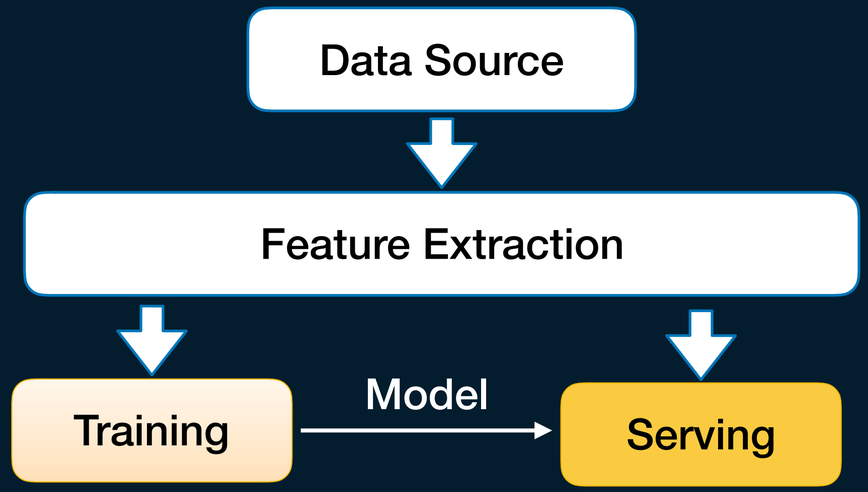
- **Batch processing in most cases**
  主要为批计算
- **Ad-hoc algorithm experiments**
  算法实验
- **Iterate until convergence**
  迭代收敛



Model Serving

批计算和流计算
Batch processing and stream processing

模型的动态更新部署
Dynamic deployment of the model



From Scenarios to API Requirements

算法实验
Ad-hoc Algorithm Experiments

迭代收敛
Iterate until converge

批计算+流计算
Batch + Stream processing

行计算
Row-based processing

动态模型更新部署
Dynamic model update and deployment

交互式编程
Interactive programming

迭代计算
Iterative processing

批流统一的 API
Unified API for batch and stream processing

基于整行的计算
Row-based API

From Scenarios to API Requirements

算法实验
Ad-hoc Algorithm Experiments

迭代收敛
Iterate until converge

批计算+流计算
Batch + Stream processing

行计算
Row-based processing

动态模型更新部署
Dynamic model update and deployment

交互式编程（FLINK–11199）
Interactive programming (FLINK-11199)

迭代计算（coming soon）
Iterative processing

批流统一的 API
Unified API for batch and stream processing

基于整行的计算（FLIP–29）
Row-based API

AGENDA
提要

- Brief introduction to Flink Table API
  Table API 简介
- The API requirements from Machine Learning Algorithms
  机器学习算法对API的核心需求
- The enhancement to Flink Table API
  Flink Table API 的扩展
- Algorithm examples based on Flink Table API
  基于 Flink Table API 的算法实现

AGENDA
提要

- The enhancement to Flink Table API
  Flink Table API 的扩展
  - **基于整行的计算及聚合功能**
    Row-based processing and aggregation function
  - 交互式编程
    Interactive programming
  - 迭代计算
    Iterative processing

Map Operator in Table API

FLINK FORWARD

Method signature
方法签名

def map(scalaFunction: Expression): Table

Usage
用法

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

Benefit
好处

table.select(udf1(), udf2(), udf3()….)
VS
table.map(udf())

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

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

  def eval([user defined inputs]): Row = {
    val result = new Row(3)
    // Business processing based on data and parameters
    // 根据数据和参数进行业务处理
    result
  }

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

INPUT(Row)	OUTPUT(Row)
1	1

FlatMap Operator in Table API

FLINK FORWARD

Method signature
方法签名

def flatMap(tableFunction: Expression): Table

Usage
用法

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

Benefit
好处

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

```
case class User(name: String, age: Int)

class MyFlatMap extends TableFunction[User] {
  def eval([user defined inputs]): Unit = {
    for(..){
      collect(User(name, age))
    }
  }
}
```

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

Aggregate Operator in Table API

FLINK FORWARD

Method signature
方法签名

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

Usage
用法

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

Benefit
好处

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

```
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

FLINK FORWARD

Method signature
方法签名

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

Usage
用法

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

Benefit
好处

New features on the TableAPI
TableAPI 上面的新增功能

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

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

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

Aggregate VS FlatAggregate

Scenes using Max and Top2 compare the difference between Aggregate and FlatAggregate
使用 Max 和 TopN 的场景比较 Aggregate 和 FlatAggregate 之间的差异

ID	NAME	PRICE
1	Latte	6
2	Milk	3
3	Breve	5
4	Mocha	8
5	Tea	4

tab.aggregate(maxAgg(PRICE))

tab.flatAggregate(topTableAgg(PRICE))

Both AGG and TableAGG represents its state using accumulator
AGG 和 TableAGG 都是利用 accumulator 来处理 state 的

1 createAccumulator()

2 accumulate(acc, PRICE)

8 8 6 6 6
Accumulator
(MAX-ACC)

8,6 8,6 6,5 6,3 6
Accumulator
(Top2-ACC)

3 getValue(ACC acc)

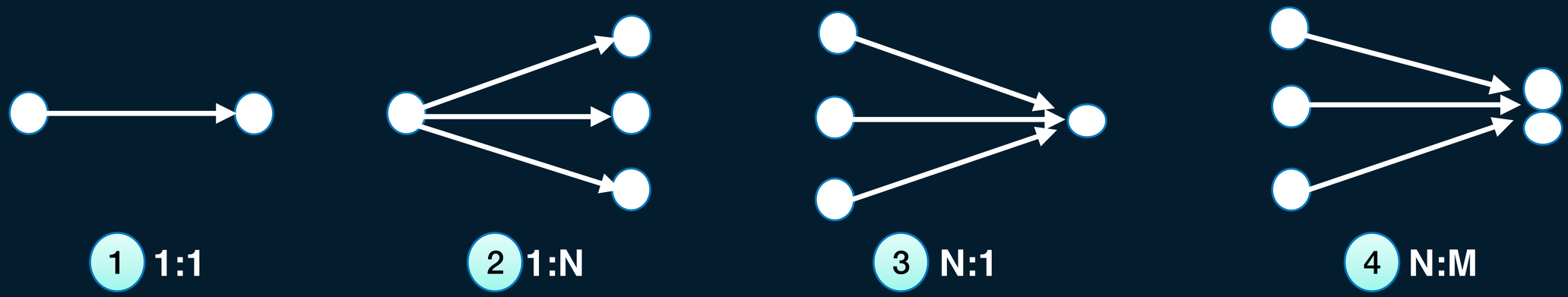
ID	NAME	PRICE
4	Mocha	8

3 emitValue(acc, out)

ID	NAME	PRICE
4	Mocha	8
1	Latte	6

Summarize the enhancements to TableAPI

	Single Row Input 单行输入	Multiple Row Input 多行输入
Single Row Output **单行输出**	ScalarFunction (select/map)	AggregateFunction (select/aggregate)
Multiple Row Output **多行输出**	TableFunction (cross join/flatmap)	TableAggregateFunction (flatAggregate)



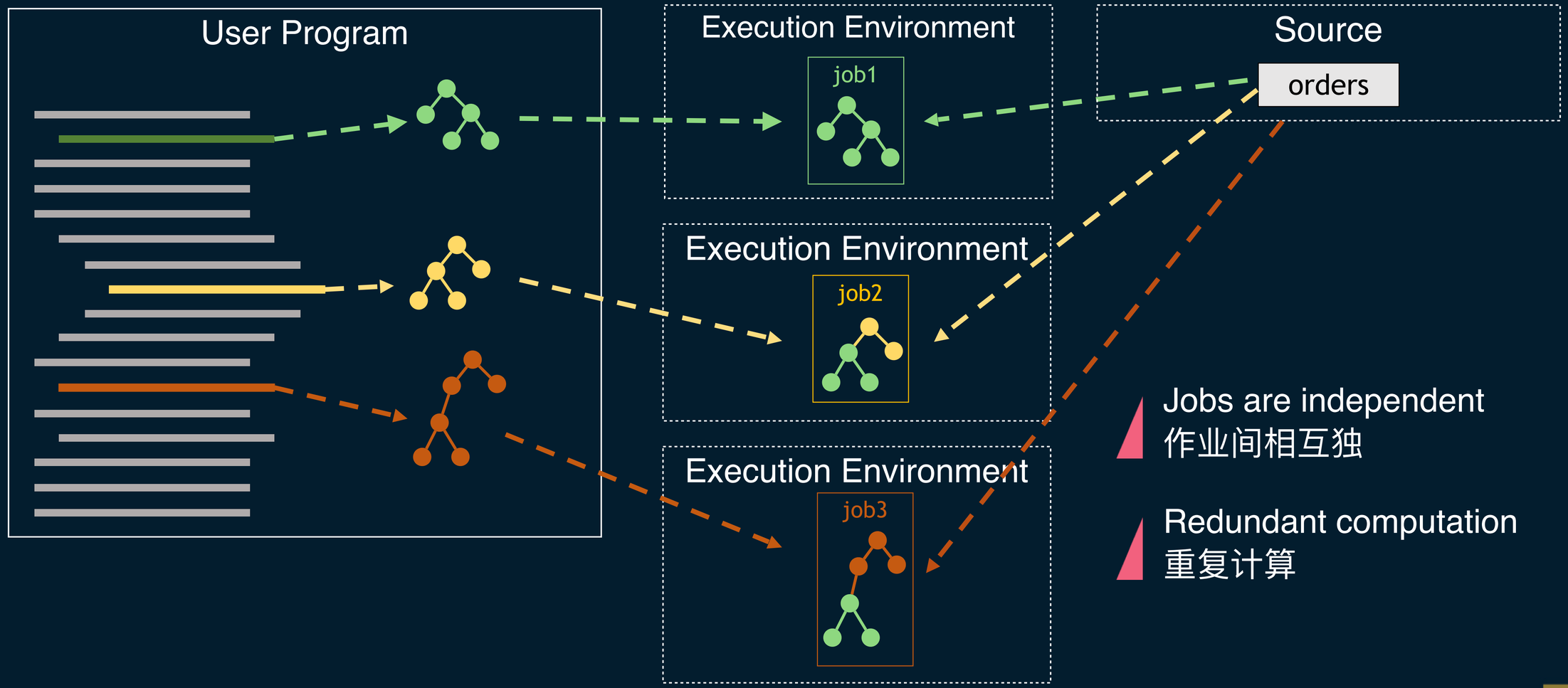
AGENDA
提要

- The enhancement to Flink Table API
  Flink Table API的扩展
  - 基于整行的计算及聚合功能
    Row-based processing and aggregation function
  - Interactive programming
    交互式编程
  - Iterative processing
    迭代计算

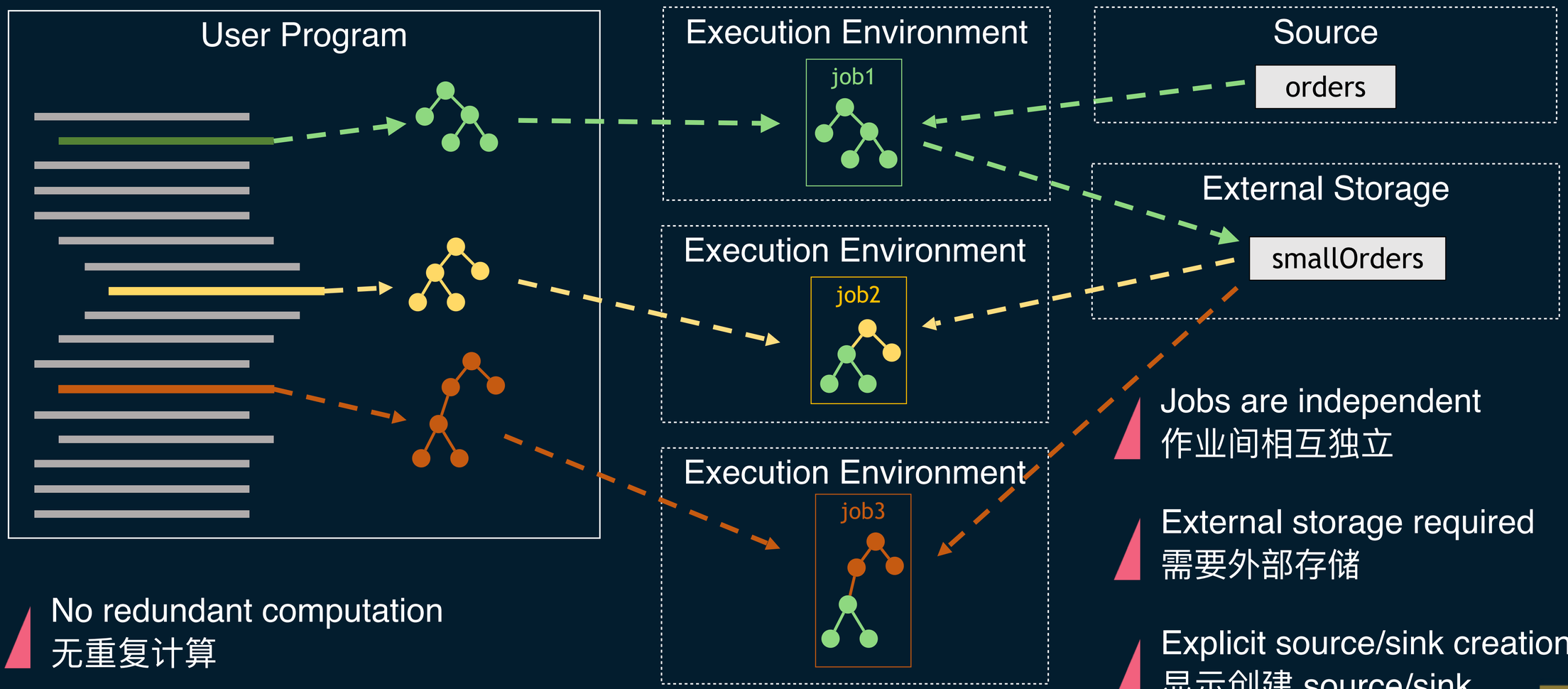
A example code snippet

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

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

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

  val smallOrdersByCountry = smallOrders.groupBy('country).select('country, 'quantity.sum as 'TotalSales)
  smallOrdersByCountry.print()

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

Static programming

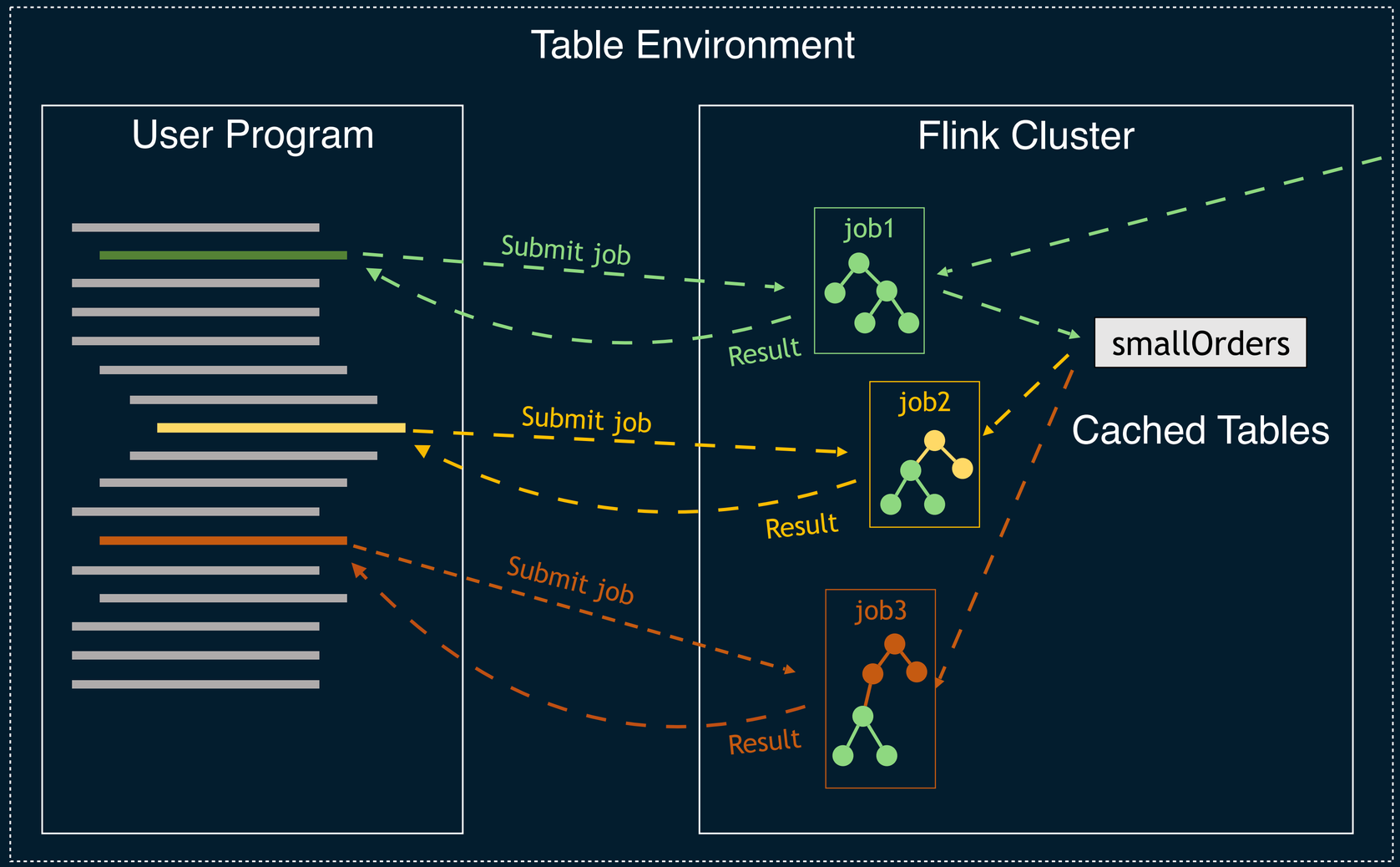


A workaround



- No redundant computation
  无重复计算
- Jobs are independent
  作业间相互独立
- External storage required
  需要外部存储
- Explicit source/sink creation
  显示创建 source/sink

Interactive Programming



- Jobs in the same context
  作业共享环境
- No external storage
  无外部存储
- Better performance
  更好的性能

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 smallOrdersByCountry = smallOrders.groupBy('country).select('country, 'quantity.sum as 'TotalSales)
  smallOrdersByCountry.print()

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

AGENDA
提要

- **The enhancement to Flink Table API**
  Flink Table API 的扩展
  - 基于整行的计算及聚合功能
    Row-based processing and aggregation function
  - Interactive programming
    交互式编程
  - **Iterative processing**
    迭代计算

Iterative processing

- Execute same processing logic repeatedly
  重复执行相同的计算逻辑
- Input of each round is from the output of the previous roun
  每次计算的变量输入来自于上次计算的结果

Components of iteration

- **Input variables (updated in iterations)**
  迭代变量（迭代中更新）
- **Step function（update input variables)**
  单次计算逻辑（更新迭代变量）
- **Termination condition**
  迭代终止条件

```
{
  val seq = Seq(1, 2, 3)

  for (i <- 1 until 100) {
    seq.foreach(_ + 1)
  }
}
```

Two ways of doing iteration

FLINK FORWARD

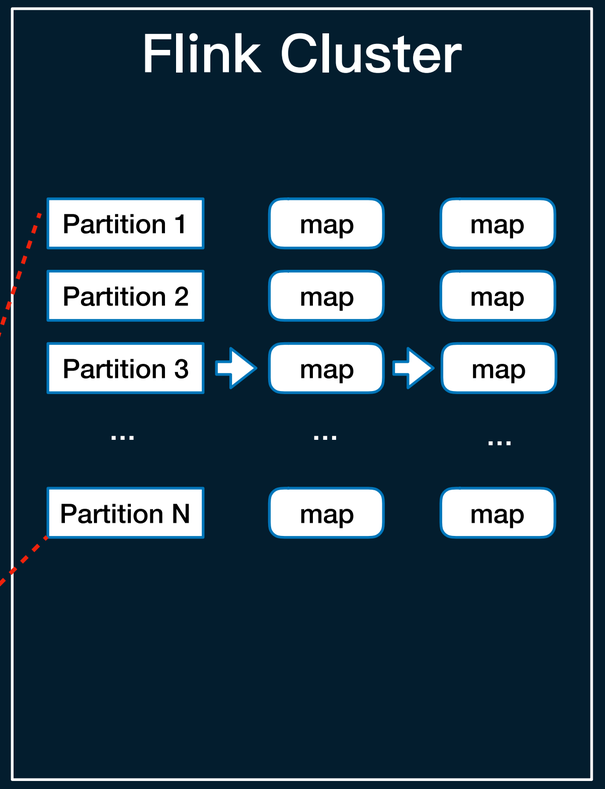
- **Iteration using interactive programming**
  利用交互式编程实现
- **Native iteration with descriptive API**
  描述性API实现的原生迭代

Iterative processing with interactive programming

FLINK FORWARD

User Program

```
{
  val NUM_ITERATIONS = 100
  var parameters = …
  val samples = ...

  samples.cache()
  for (i <- 1 until NUM_ITERATIONS) {
    val gradients = samples.map(sample => computeGradients(sample))
    parameters = gradients.map(grad => updateParameters(grad)).collect()
  }
}
```



Iterative processing with interactive programming

FLINK FORWARD

User Program

```
{
  val NUM_ITERATIONS = 100
  var parameters = …
  val samples = ...

  samples.cache()
  for (i <- 1 until NUM_ITERATIONS) {
    val gradients = samples.map(sample => computeGradients(sample))
    parameters = gradients.map(grad => updateParameters(grad)).collect()
  }
}
```

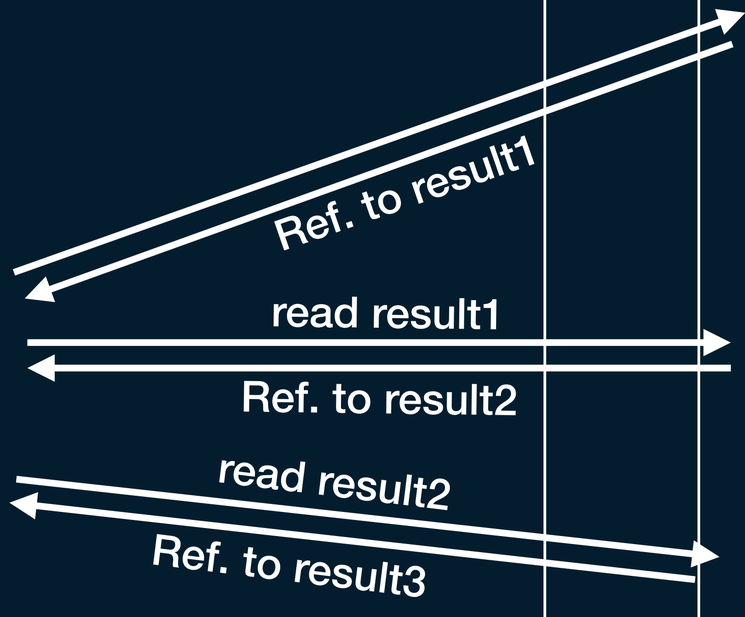
Parameters are updated through repeated serialization.
变量通过每轮迭代中序列化上传进行更新

Iterative processing with interactive programming

FLINK FORWARD

User Program

```
{
  var t = tEnv.fromCollection(data).as ('f1)
  var count = t.select('f1.count).collect().size

  while (count > 0) {
    newT = t.select('f1 + 10).filter('f1 < 1000)
    newT.cache()
    count = t.select('f1.count).collect().size
    t = newT
    t.uncache()
  }
}
```



Variables updated with cached result from last round

通过读取 cache 中的上轮结果更新变量

Iterative processing with interactive programming

- The iterations are implemented by the users
  迭代由用户实现
- Each round of iteration submits a job
  每轮迭代提交一个作业
- Actively manage the lifecycle of the cached tables
  主动cache生命周期维护

Iterative processing with interactive programming

FLINK FORWARD

Pros
优点

Simple and intuitive
简单直观

Good flexibility in code logic
代码逻辑灵活性高

Cons
缺点

Overhead of job submission
任务调度开销

Synchronization after each round
轮次间迭代需回到用户程序同步

Iteration with descriptive API

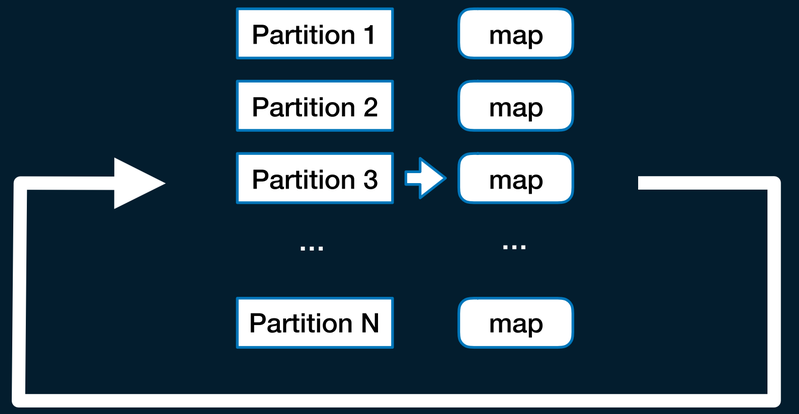
- **Native iteration implemented by the processing engine**
  计算引擎实现的原生迭代
- **Feedback edge on the processing DAG**
  在 DAG 基础上增加反馈边
- **Exist in Flink DataSet and DataStream**
  Flink DataSet 和 DataStream 已经采用

Iteration with descriptive API

User Program

```
{
  val input: DataSet[(String, Int)] = ...
  val iterated = input.iterate(5) { previous =>
    val next = previous.map { x => (x._1, x._2 + 1) }
    next
  }
}
```

Flink Cluster



Single job with a feedback edge
单个有反馈边的作业

Caveats of DataSet and DataStream iteration API

FLINK FORWARD

- **No nested iteration support**
  不支持嵌套迭代
- **No multi-variable iteration support**
  不支持多变量迭代

The iteration API on Flink Table (Multiple Input)

```
}
  val a: Table = ...
  val b: Table = ...
  val resultSeq = Table.iterate(a, b) {

    val next_a = b.select('v_b + 1 as 'v_a)

    val next_b = next_a.select('v_a * 2 as 'v_b)

    Seq(next_a, next_b)

  }.times(10)
}
```

Iteration variables

Step function

Termination condition

The iteration API on Flink Table (Nested Iterations)

FLINK FORWARD

```
}
  val a: Table = ...
  val b: Table = ...
  val resultSeq = iterate(a, b) {

    val next_a = iterate(a) {

      Seq(a.select(‘v_a + 1 as 'v_a))

    }.times(100).head

    val next_b = next_a.select('v_a * 2 as 'v_b)

    Seq(next_a, next_b)

  }.times(10)
}
```

Custom termination condition

```
}
  val a: Table = ...
  val b: Table = ...
  val termCond: AtomicReference[Table] = ...
  val resultSeq = Table.iterate(a, b) {

    val next_a = b.select('v_b + 1 as 'v_a)

    val next_b = next_a.select('v_a * 2 as 'v_b)

    termCond.set(next_a)

    Seq(next_a, next_b)

  }.untilConverge(termCond.get())
}
```

Stop iteration when next_a is empty
当next_a为空时迭代终止

The iteration API on Flink Table

- About to contribute to the community
  即将贡献至社区
- Future work
  后续计划
  - Native support for SGD
    增加对 SGD 的原生支持
  - Native support for online learning scenarios
    增加对 online learning 场景的原生支持

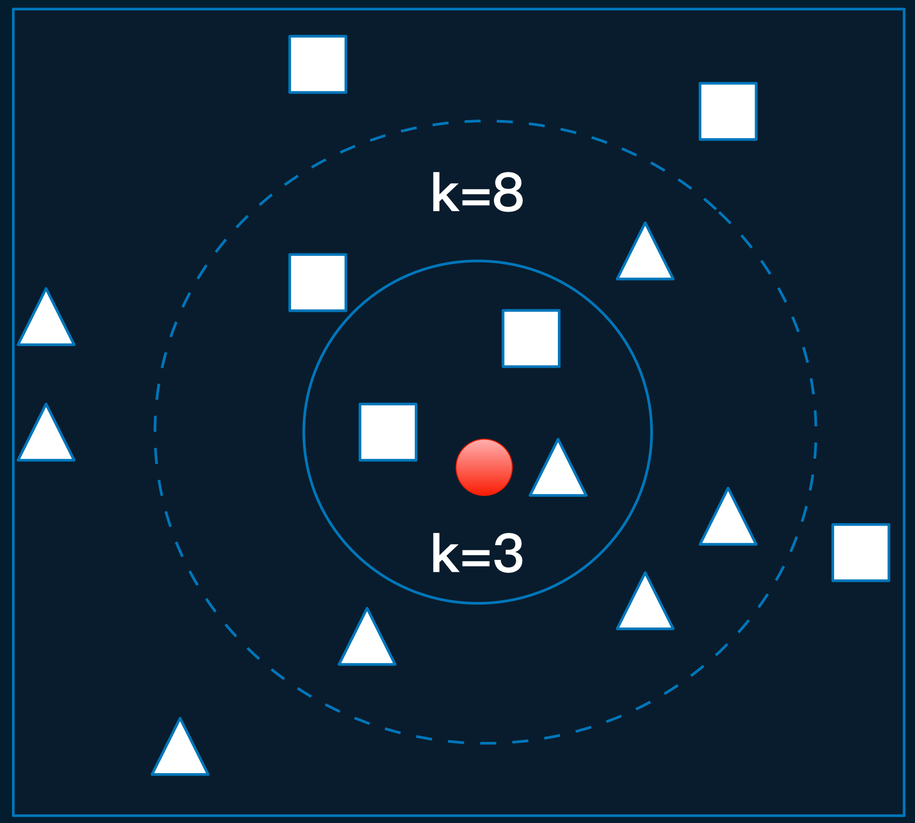
FLINK FORWARD

AGENDA
提要

- Brief introduction to Flink Table API
  Table API 简介
- The API requirements from Machine Learning Algorithms
  机器学习算法对 API 的核心需求
- The enhancement to Flink Table API
  Flink Table API 的扩展
- **Algorithm examples based on Flink Table API**
  基于 Flink Table API 的算法实现

KNN(k-nearest neighbor) on TableAPI

- Classification of red data in the following figure by KNN algorithm
  KNN 算法对下图红色数据进行分类



- TestData 待预测数据
- Triangle 三角形类别
- Square 正方形类别
- K=3, Red data belongs to square
  K=3, 红色数据属于正方形分类
- K=8, Red data belongs to triangle
  K=8, 红色数据属于三角形分类

KNN(k-nearest neighbor) on TableAPI

```
// 1. Assign a unique ID to each test data / 每条测试数据分配唯一ID
val inputWithId = input.as('t).map(uuidMapFun('t))
// 2. Data chunking and concat test and training data/ 数据分块和连接测试和训练数据
val inputSplit = TableFlinkMLTools.block(inputWithId, blocks, Some(partitioner))
val crossTuned = trainingSet.as('train).join(inputSplit.as('test), “1=1")
.select('train, ‘test)
// 3. Calculate the distance between the test data and each training data
// 计算测试数与每条训练数据的距离
val crossed = crossTuned.flatMap(distanceFlatMapFun('train, ‘test))
.select('singleTrain, 'singleTest, 'id, 'distance)
// 4. TopK training data / 计算邻近的K条训练数据
val result = crossed.groupBy(‘id).flatAggregate(topKTableAggFunction(
'singleTrain, 'singleTest, 'id, ‘distance))
```

OutPut: K(3) = (Triangle/三角形, Square/正方形，Square/正方形)

More about Apache Flink

FLINK FORWARD

- Source code for KNN on TableAPI
  KNN on TableAPI 源代码

  Github: http://t.cn/EU9vPF0

- Sun Jincheng's Apache Flink album
  孙金城 的 Apache Flink 专栏


THANKS