

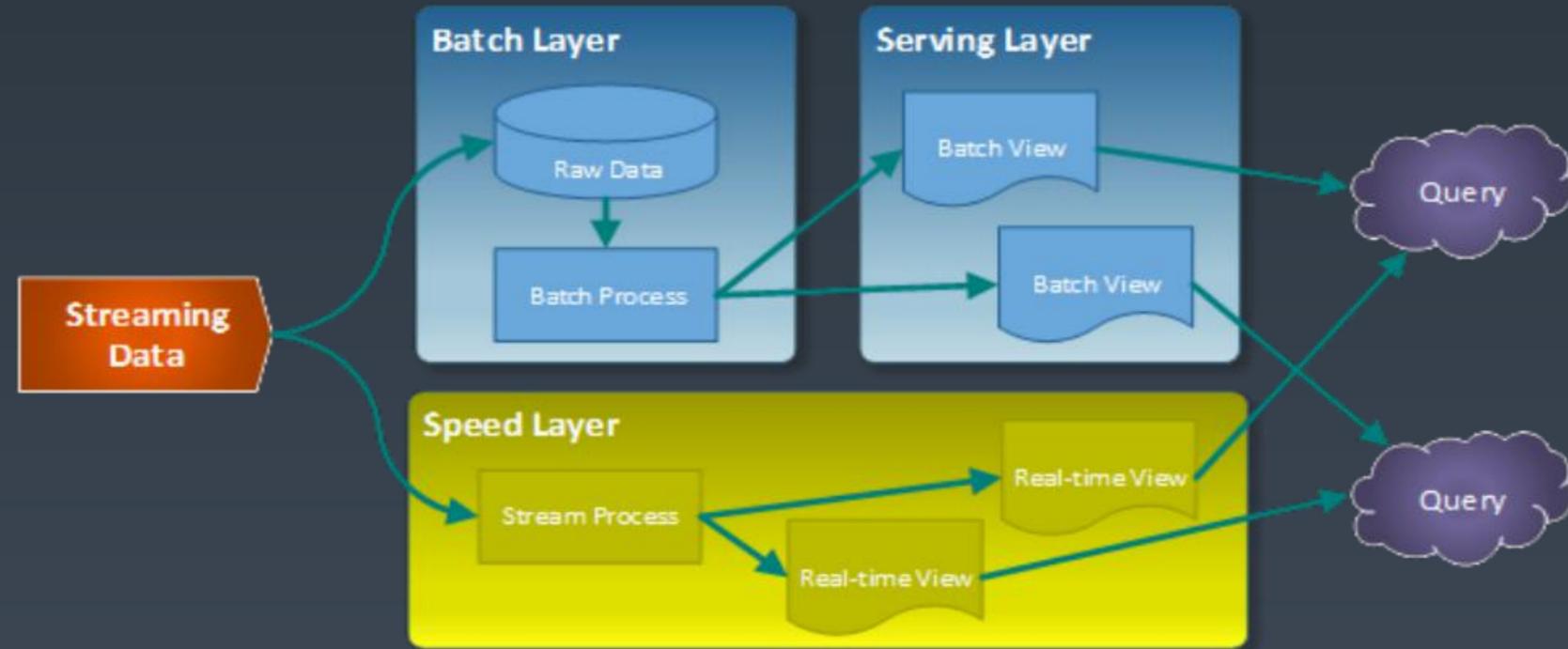
Flink 流批一体的技术架构以及在阿里的实践

目录

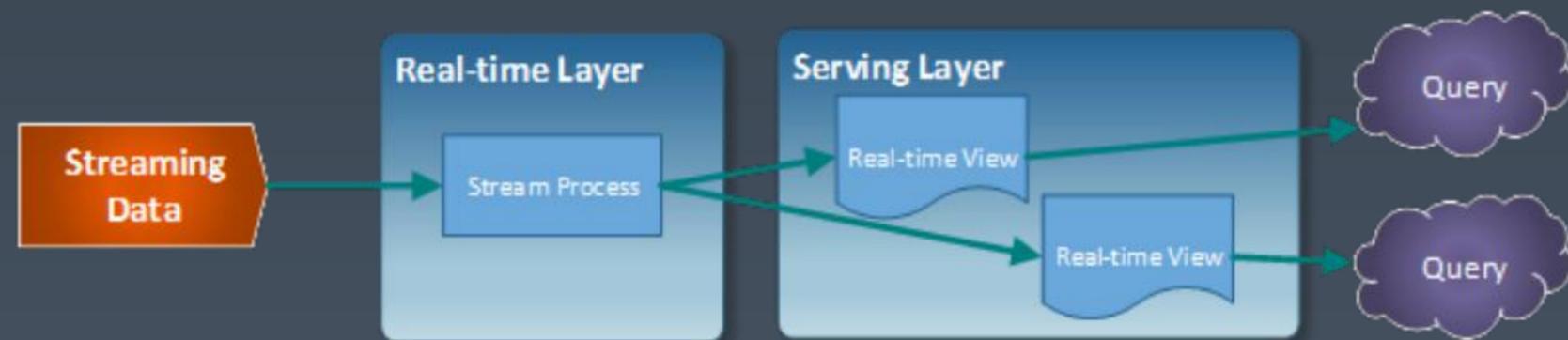
- 需求和挑战
- Flink 架构简介
- 流批一体的入口 – SQL
- 流批一体的大规模实践 – 在线机器学习平台
- 总结和展望

主流架构对比

Lambda



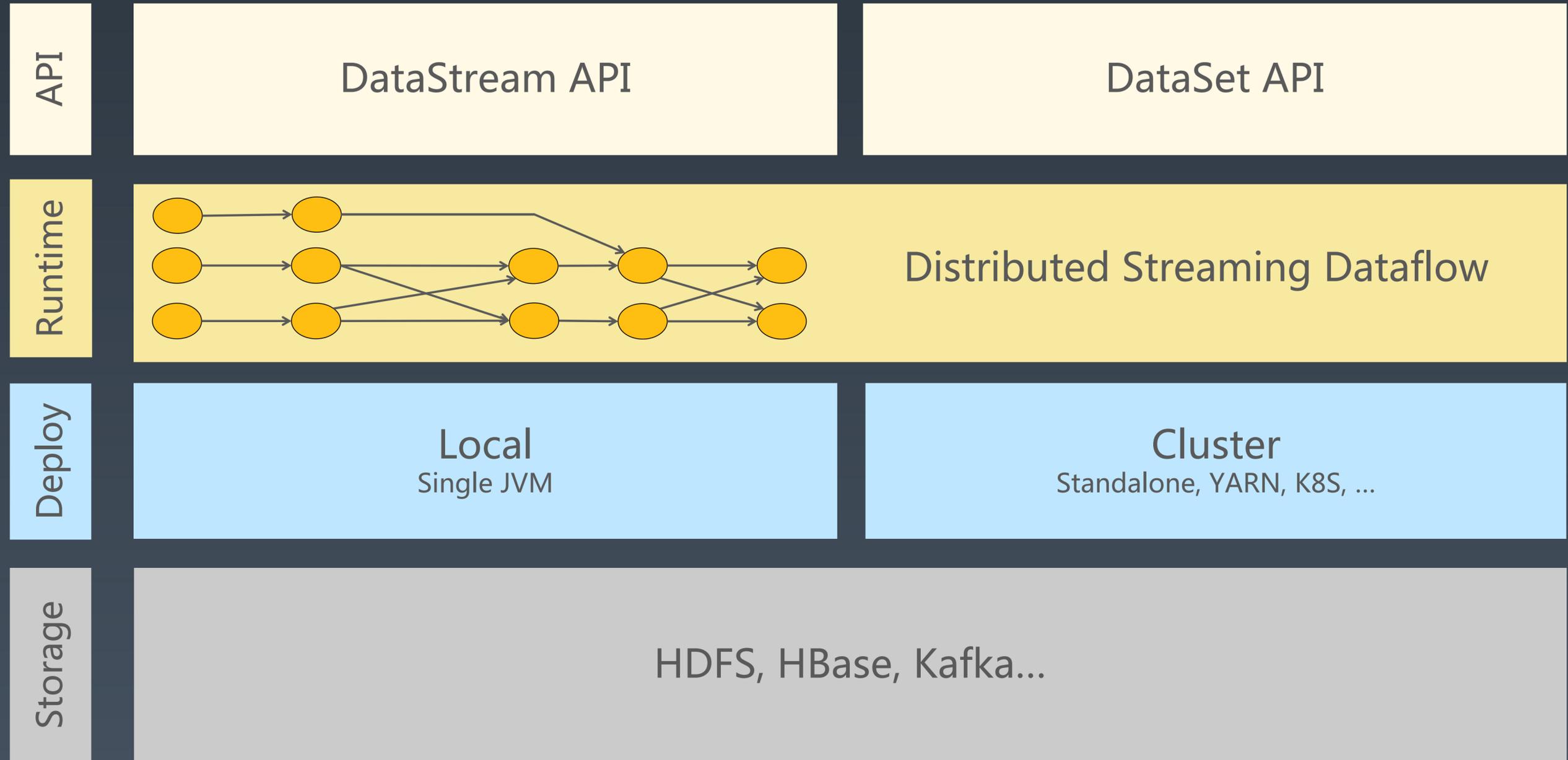
Kappa



流批一体系统的需求和挑战

- 执行引擎同时具备多种能力
 1. 低延迟的流计算
 2. 高吞吐、高稳定性的批处理
- 用户角度：编程接口统一。一份代码，一样的结果
- 开发人员角度：架构统一，代码复用

Flink 架构简介



Word Count

```
val lines: DataStream[String] = env.readFromQueue(address)

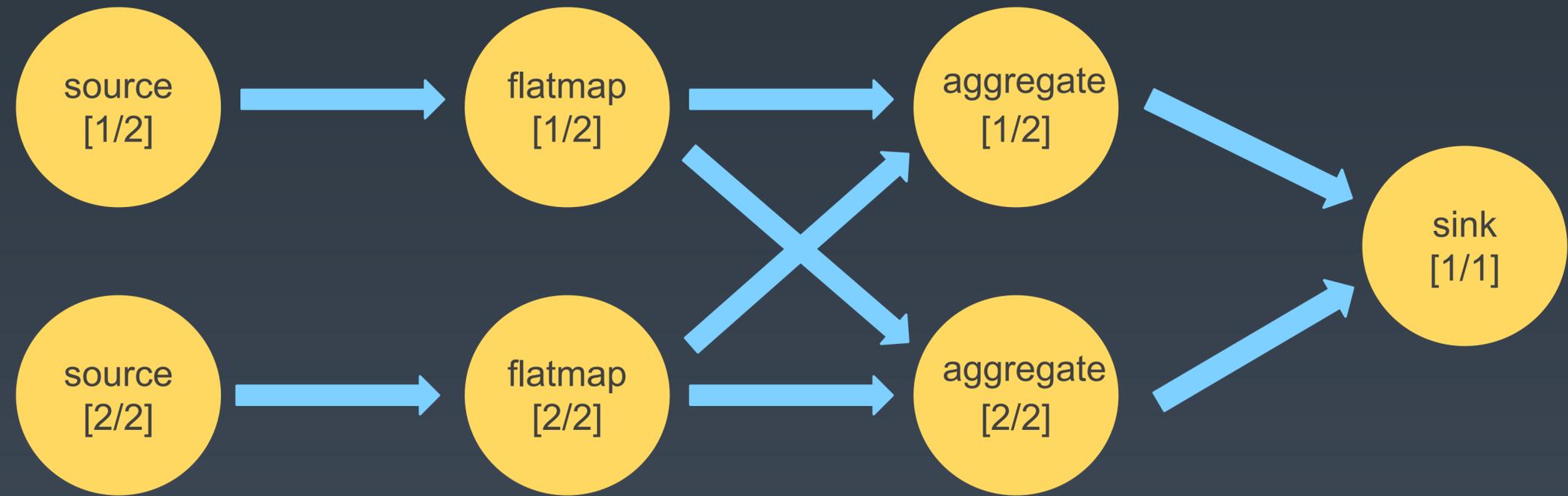
val words: DataStream[Word] = lines.flatMap((line) => split(line))

val counts: DataStream[Int] = words
    .keyBy("word")
    .sum("frequency")

counts.addSink(new RollingSink(path))
```

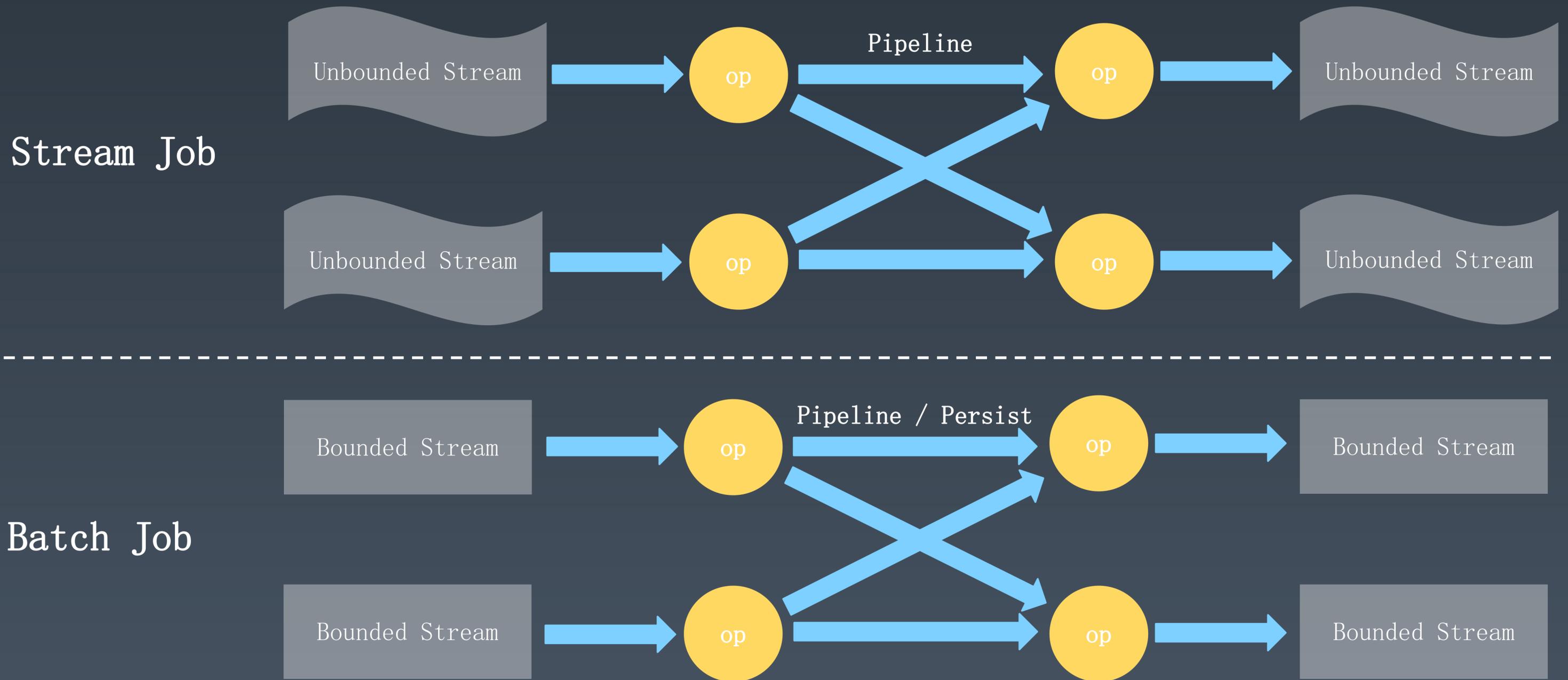
Streaming Dataflow

Word Count :



- 计算模型的核心抽象
- 表达作业逻辑的DAG主要由点和边构成
 1. 点：算子（operator），包含主要的计算逻辑
 2. 边：数据流通管道，可以运行在多种介质上（网络、文件、内存）

批处理是流计算的特例



流批统一的入口 - SQL

USER_SCORES		
User	Score	Time
Julie	7	12:01
Frank	3	12:03
Julie	1	12:03
Frank	2	12:06
Julie	4	12:07

USER_SCORES is a source table/stream.

Batch Mode:

```
12:07> SELECT Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

Name	Score	Time
Julie	12	12:07
Frank	5	12:06

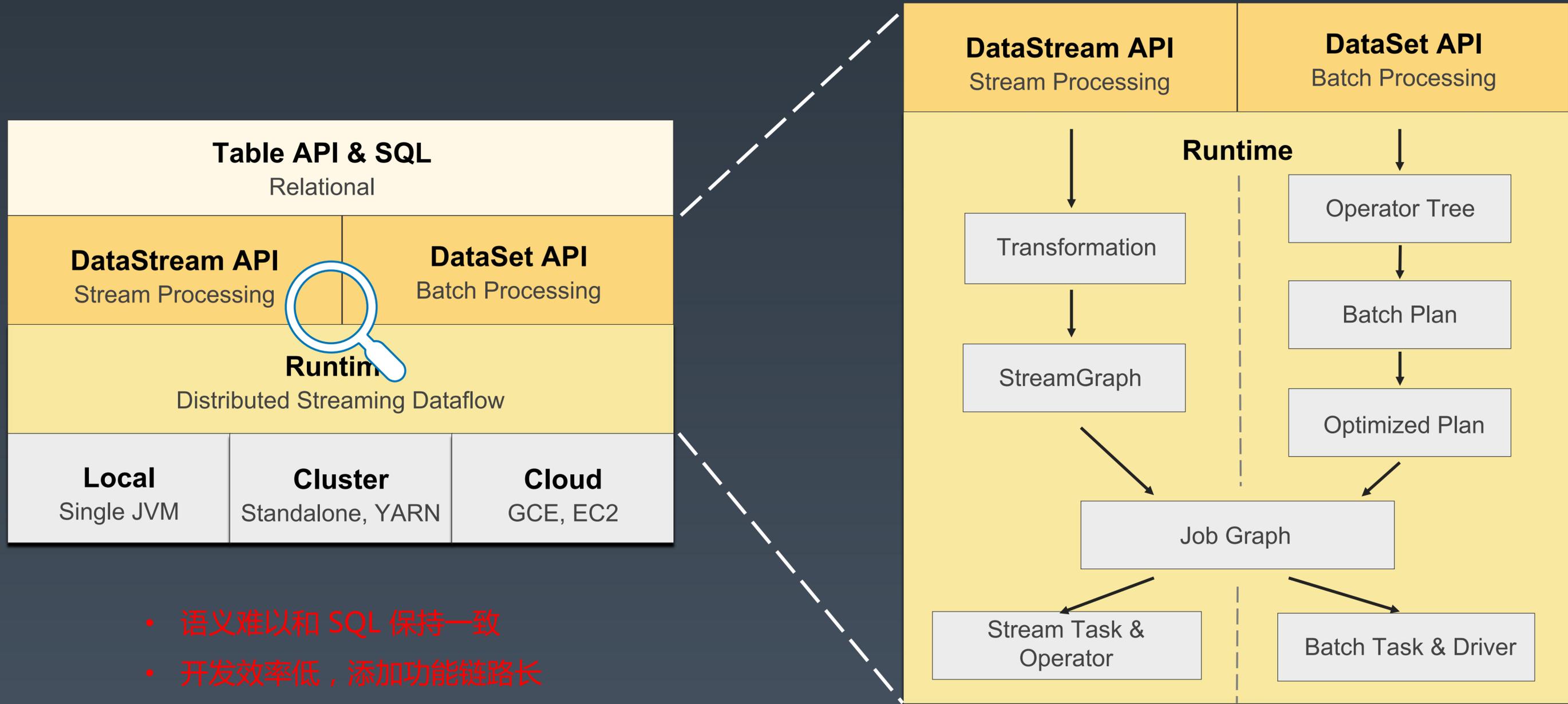
1. 流有 Early fire
2. 最终结果一致

Stream Mode:

```
12:01> SELECT Name, SUM(Score), MAX(Time) FROM USER_SCORES GROUP BY Name;
```

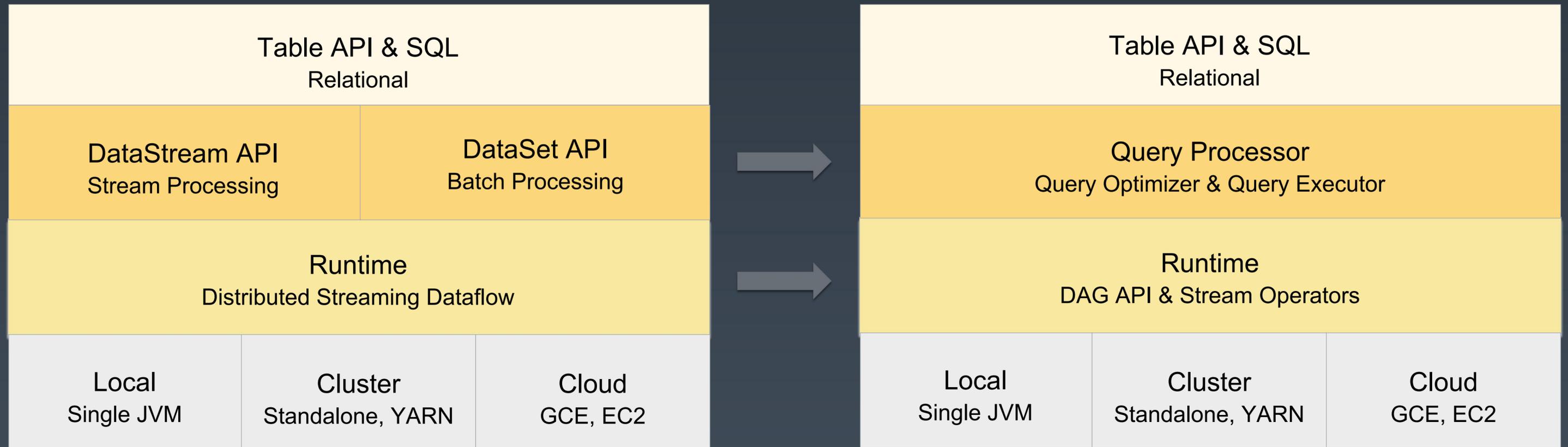
[-inf, 12:01)			[12:01, 12:04)			[12:04, now)		
Name	Score	Time	Name	Score	Time	Name	Score	Time
			Julie	8	12:03	Julie	12	12:07
			Frank	3	12:03	Frank	5	12:06

流批统一的架构

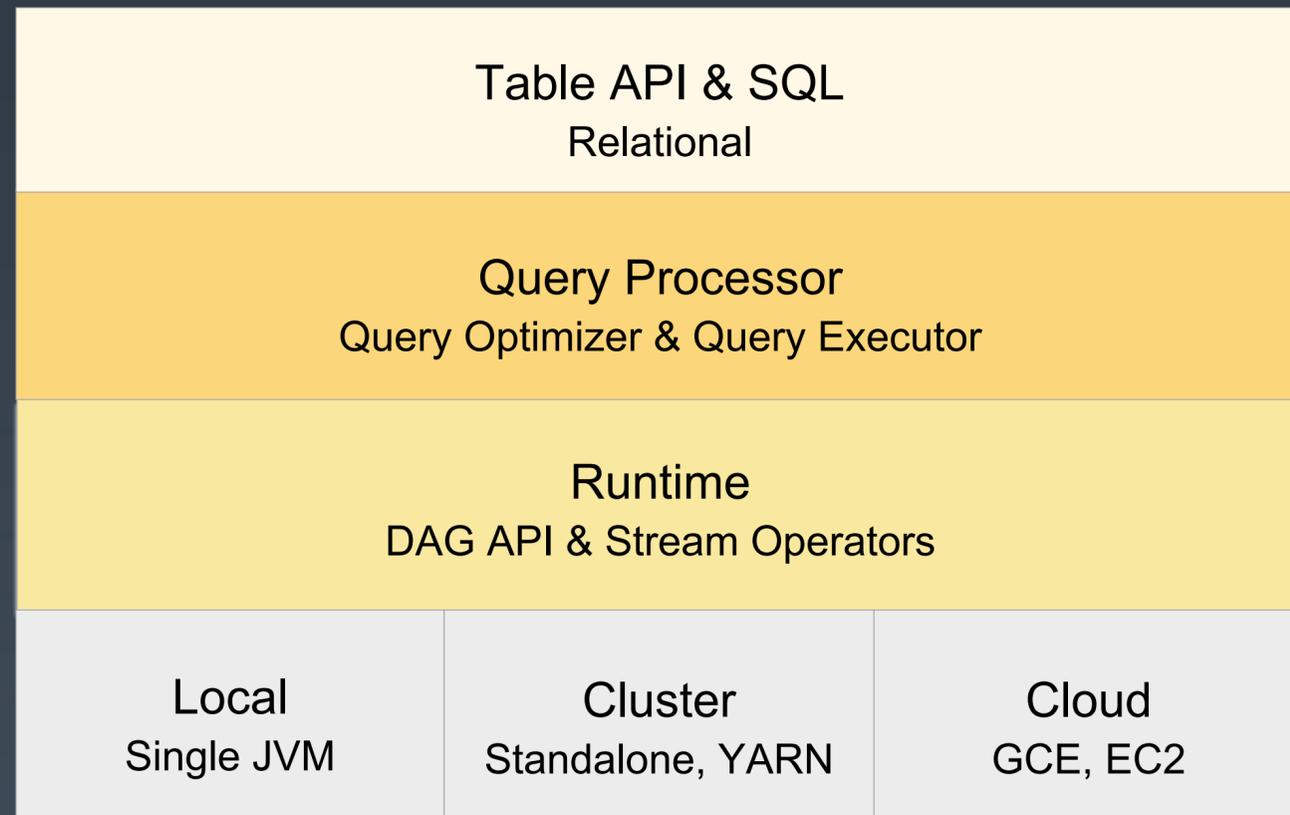


- 语义难以和 SQL 保持一致
- 开发效率低，添加功能链路长
- 执行模式不同，代码难以复用

新架构



新架构主要修改点



1. Table API 和 SQL 变成一级 API
2. 引入 Query Processor 模块统一流和批的处理
3. 使用相同的 DAG 和 Stream Operator 来描述流批作业
4. Runtime 统一到流上 push based 实现
5. 未来可以考虑和 DataStream 共享算子

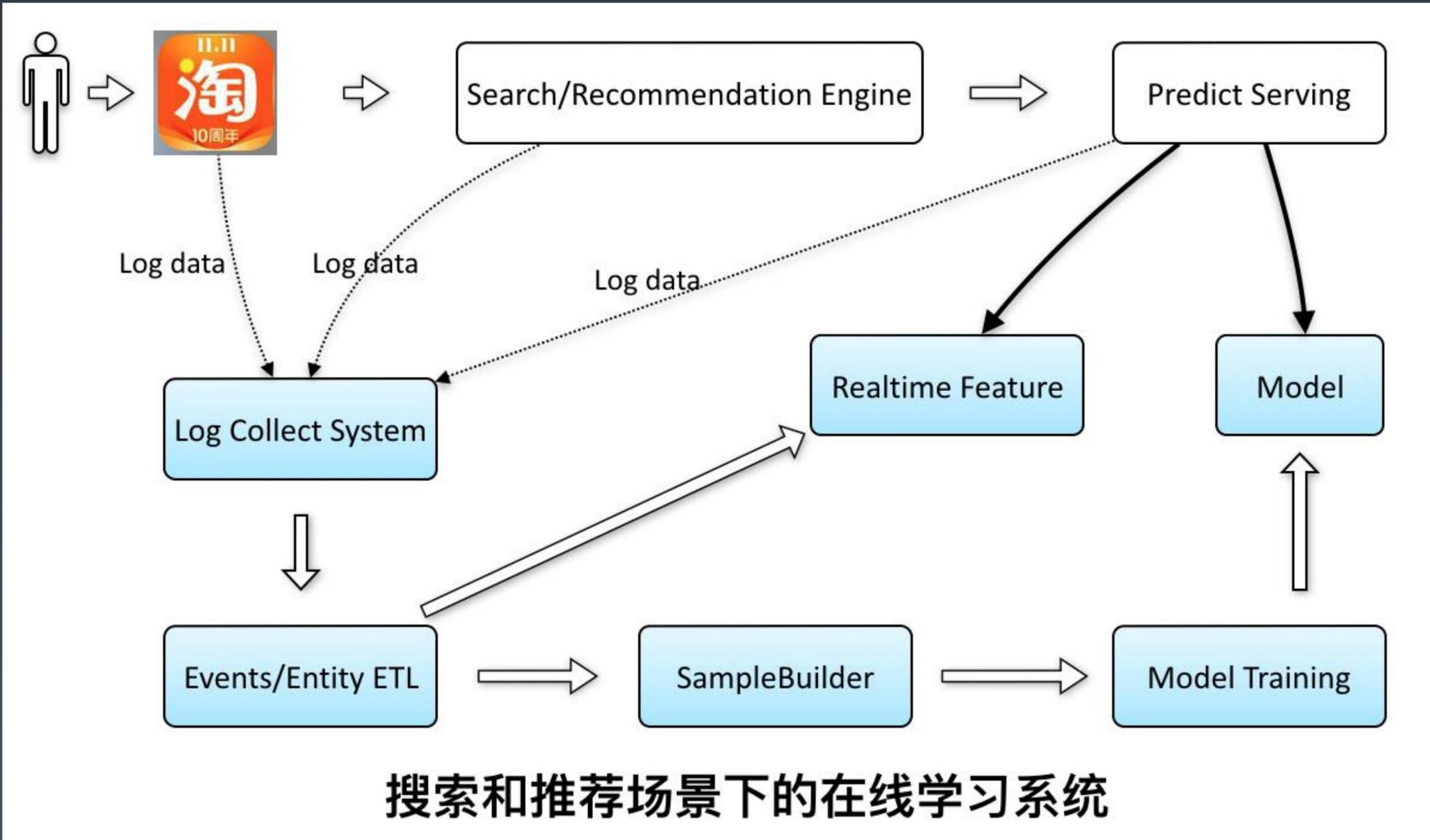
Query Processor 简介



回顾需求

- 执行引擎同时具备多种能力 
 1. 低延迟的流计算
 2. 高吞吐、高稳定性的批处理
- 用户角度：编程接口统一。一份代码，一样的结果 
- 开发人员角度：架构统一，代码复用 

大规模实践 - 在线机器学习平台



问题和挑战

1. Event 和 Entity 的存储选择
2. 时效性：ETL 作业 + 实时训练作业
3. 样本一致性：规避 Early fire 导致的错误样本
4. 数据可回溯：任意环节的纠错能力
5. 批和流样本的统一生成
6. 模型的 Validation 机制

存储的选择

- 需要同时具备低延迟的流式订阅和高吞吐的历史数据回溯功能
- ETL 作业延迟和数据重复的权衡
 1. Exactly once 由于 Checkpoint barrier 对齐的问题导致延迟波动
 2. At least once 不保证数据不重复输出
- 解决方案：消息队列 + 类 HBase 的 KV 系统
 1. 消息队列提供低延时的流式订阅
 2. ETL 作业使用 At least once，利用 KV 系统的 Update 能力进行去重并提供历史数据 Scan 功能
 3. 平台对数据源进行包装，在不同场景下切换

实时训练的时效性和一致性

- 在 CVR (Conversion Rate , 转化率) 模型中 , 我们会根据点击是否成交而生成相应的样本 :
 1. 假如用户点击后进行了购买 , 则是一个正样本
 2. 假如用户点击后没有购买 , 则是一个负样本
 3. 用户从点击到成交的时间不确定 , 从秒级到小时级不等
- 解决方案 : 利用 SQL 的 Retraction 机制
 1. 在延迟容忍程度内 , 先根据当前数据情况输出结果
 2. 当结果需要有变化时 , 先发送一条之前错误的结果 , 标记为 Retraction , 然后再发送一条最新的正确结果
 3. 用户在算法逻辑中对 Retraction 消息进行处理 , 对错误样本进行修正

批流一体的样本生成机制

- 虽然有了一定的错误修正机制，但还是避免不了产生一些负面影响。需要定期进行样本的批量生成
- 有些模型还不需要很高的时效性，更关注样本准确性
- 解决方案：直接复用实时的样本生成逻辑，一样的 SQL，一样的 UDF
 1. 平台将 Source 自动替换成 KV 系统的历史数据进行 Scan
 2. SQL 作业自动切换成批处理模式执行
 3. 批处理作业使用混部资源运行

大规模批处理作业优化 - JobManager

- 优化资源使用
 1. 避免 $N * M$ 级别（上游并发为 N ，下游并发为 M ）的内存占用消耗
 2. Metrics reporter 的内存占用优化
- 稳定性提升
 1. JobManager Failover [FLINK-4911]
 2. Region-based Task Failover [FLIP1/FLINK-4256]
- 调度性能优化

大规模批处理作业优化 – Shuffle Service

- Flink 自带的 Batch Shuffle 会将数据托管给 TM
- 混部环境中，TM被杀是常态
- 借助 Yarn Auxiliary Service，将数据托管给可靠性更高的Node Manager
- 社区也在尝试更通用的支持方案：[FLIP-31 - Pluggable Shuffle Manager](#)

业务效果

- 日志 ETL 作业双十一峰值接近1亿条每秒
- 手淘主搜索和猜你喜欢样本量每天数百 TB
- 实时模型直接提升了双十一效果
- 最大单一批处理作业处理 400 TB数据，最大单节点并发20000
- 批处理混部运行拉升了集群资源水位

总结

- 经过改造之后，Flink 已经具备了比较完善流批一体的技术架构
- 流批一体的技术架构可以简化业务系统，提升业务效果
- 从扩展性、性能、稳定性这三个维度看，Flink 的批处理已初步通过大规模上线的考验

未来展望

- 流批一体还缺少最后一块拼图：存储
- 流计算 SQL 作业的状态兼容性和作业热升级支持
- 探索介于流批计算之间的其他计算形态
- 批处理大规模推广的关键：Hive 兼容性
- 基于新技术架构的机器学习库

THANKS!