Flink 流处理简介

左元

2021 年 8 月 28 日

尚硅谷大数据组

主要内容

- Flink 是什么
- 为什么要用 Flink
- 流处理的发展和演变
- Flink 的主要特点
- Flink vs Spark Streaming

Flink 是什么

图 1: Apache Logo

图 2: Flink Logo

- Apache Flink is a framework and distributed processing engine for stateful computations over unbounded and bounded data streams.
- Apache Flink 是一个框架和分布式处理引擎，用于对无界和有界数据流进行状态计算。

Flink 目前在国内企业的应用

图 3: Flink 目前在国内企业的应用

为什么选择 Flink

- 流数据更真实地反映了我们的生活方式
- 传统的数据架构是基于有限数据集的
- 我们的目标
  - 低延迟（Spark Streaming 的延迟是秒级，Flink 延迟是毫秒级）
  - 高吞吐（阿里每秒钟使用 Flink 处理 4.6PB，双十一大屏）
  - 结果的准确性和良好的容错性（exactly-once）

哪些行业需要处理流数据

- 电商和市场营销
  - 数据报表、广告投放、业务流程需要
- 物联网（IOT）
  - 传感器实时数据采集和显示、实时报警，交通运输业（自动驾驶）
- 电信业
  - 基站流量调配
- 银行和金融业
  - 实时结算和通知推送，实时检测异常行为（信用卡盗卡）

传统数据处理架构

- 事务处理（OLTP）

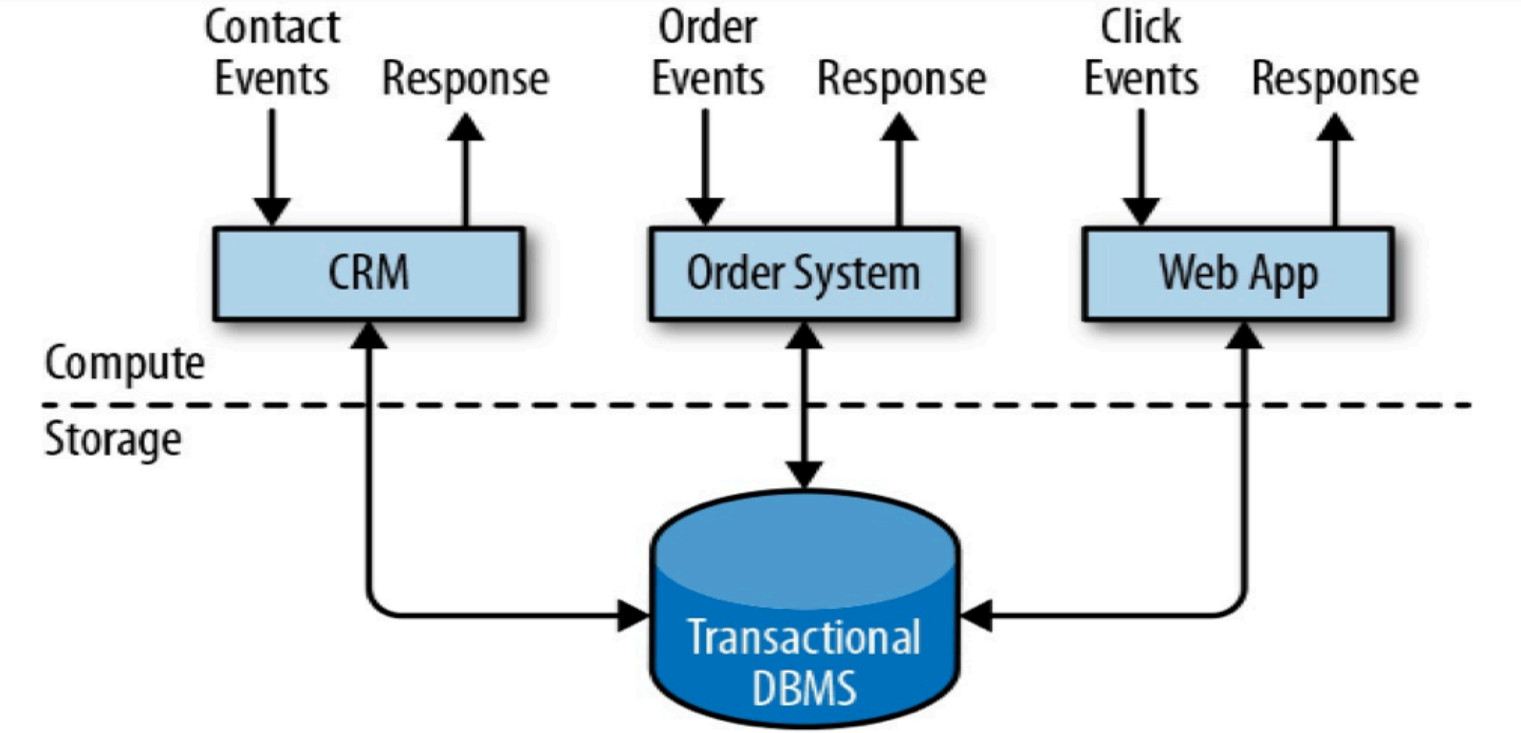


图 4: OLTP 架构

传统数据处理架构

- 分析处理
  - 将数据从业务数据库复制到数仓，再进行分析和查询

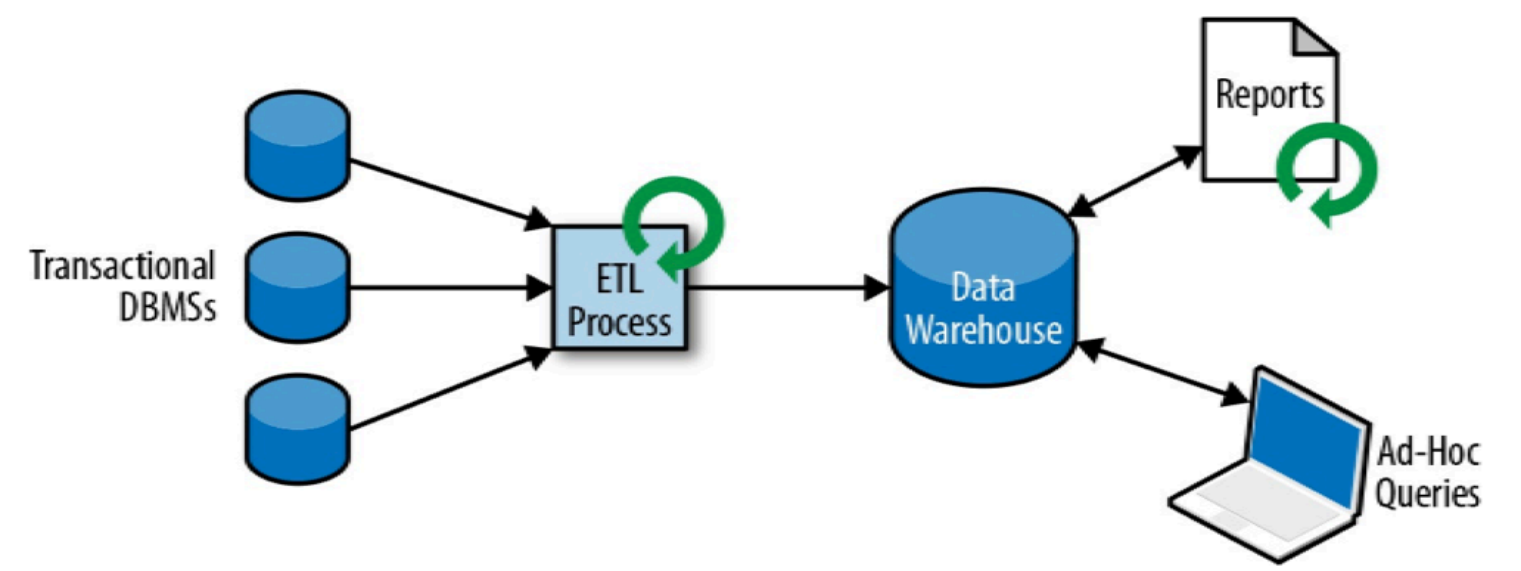


图 5: OLAP 架构

有状态的流式处理

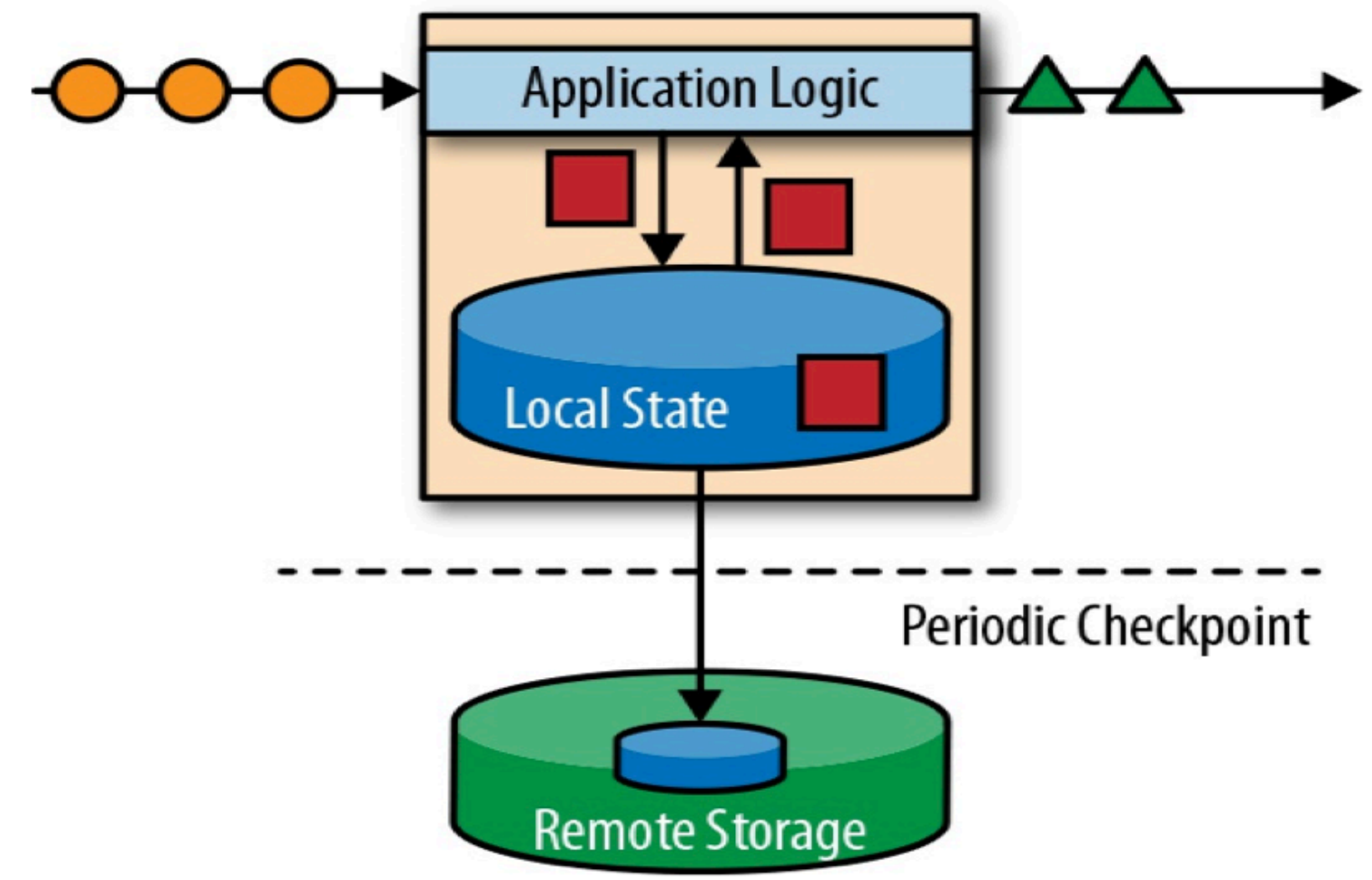


图 6: 有状态的流式处理

流处理的演变

- lambda 架构
  - 用两套系统，同时保证低延迟和结果准确

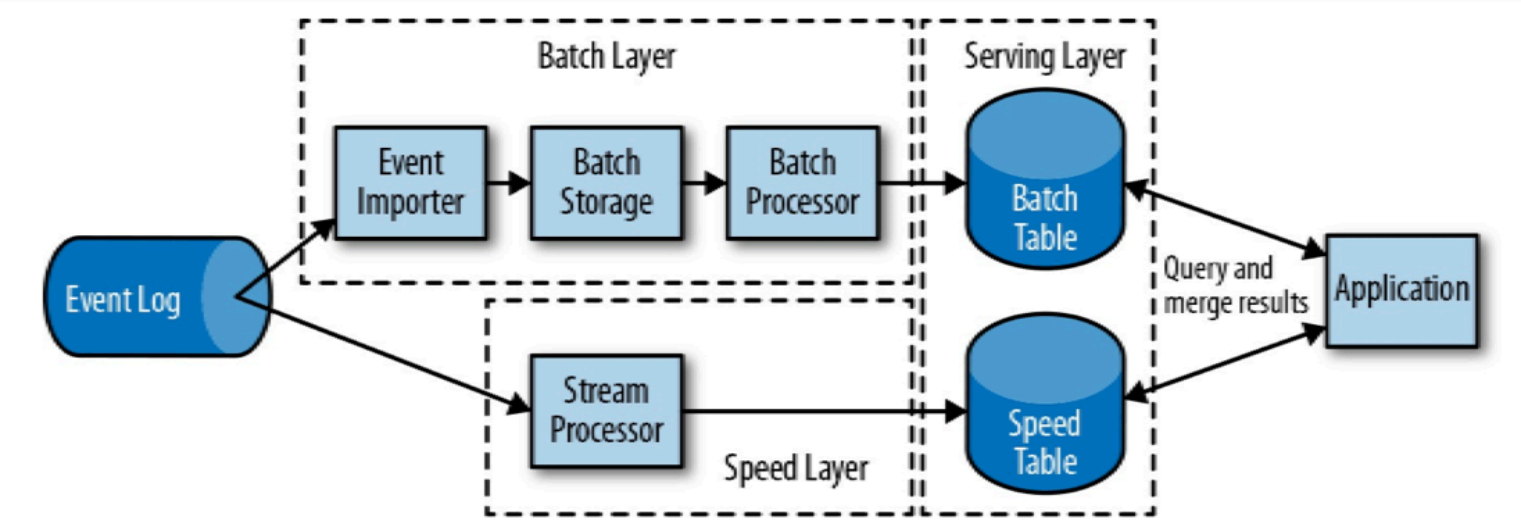
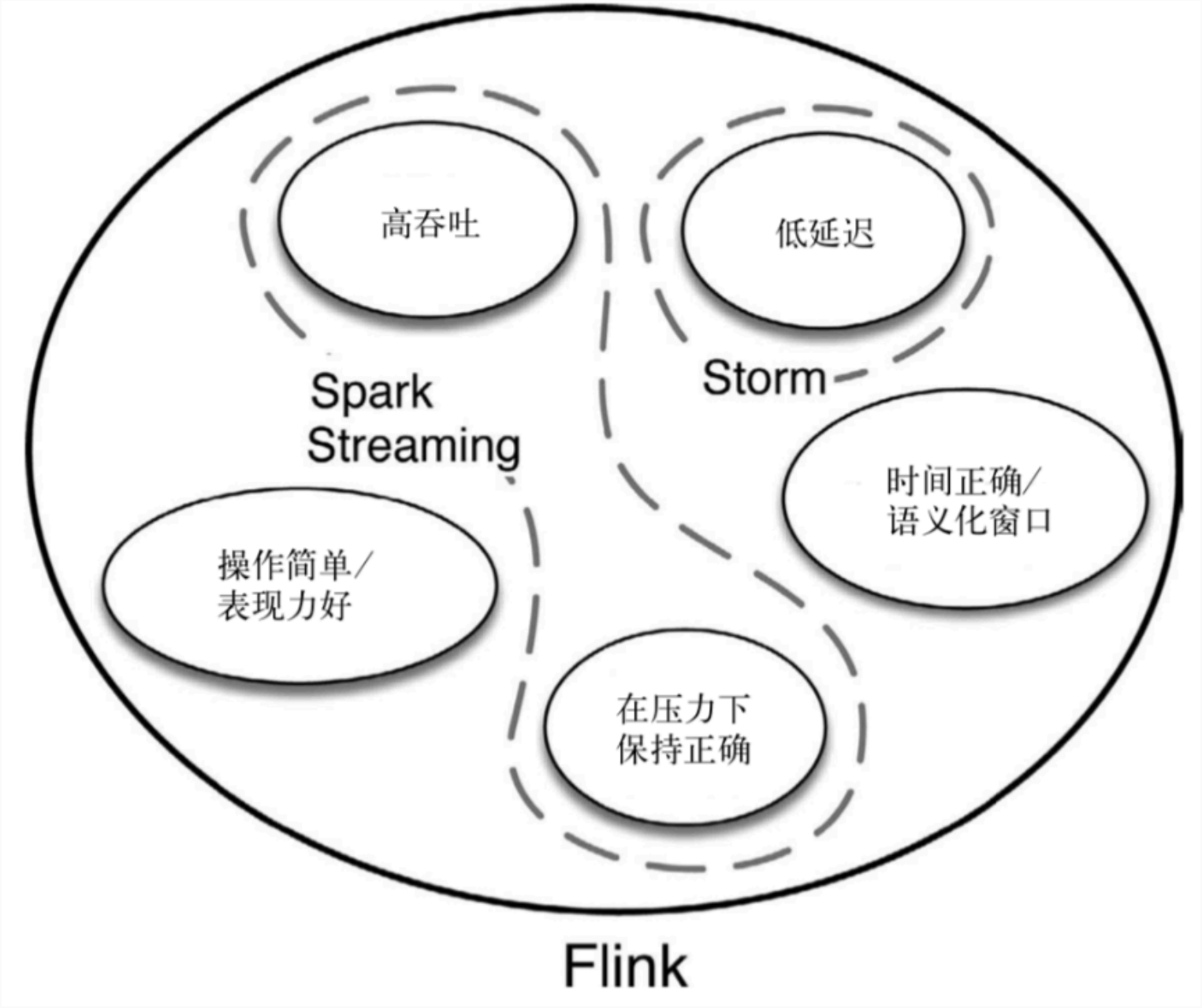


图 7: lambda 架构

流处理的演变


高吞吐
低延迟
Spark
Streaming
Storm
时间正确/
语义化窗口
操作简单/
表现力好
在压力下
保持正确
Flink

Flink 的主要特点

- 事件驱动（Event-driven）

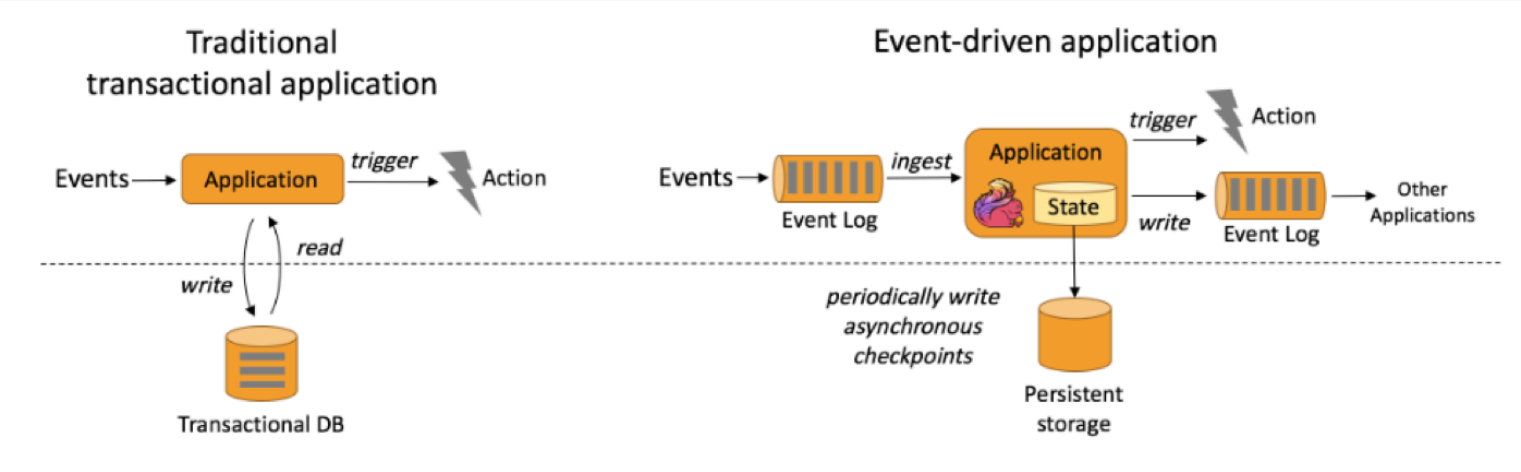


图 9: 事件驱动

Flink 的主要特点

- 基于流的世界观
  - 在 Flink 的世界观中，一切都是由流组成的，离线数据是有界的流；实时数据是一个没有界限的流：这就是所谓的有界流和无界流

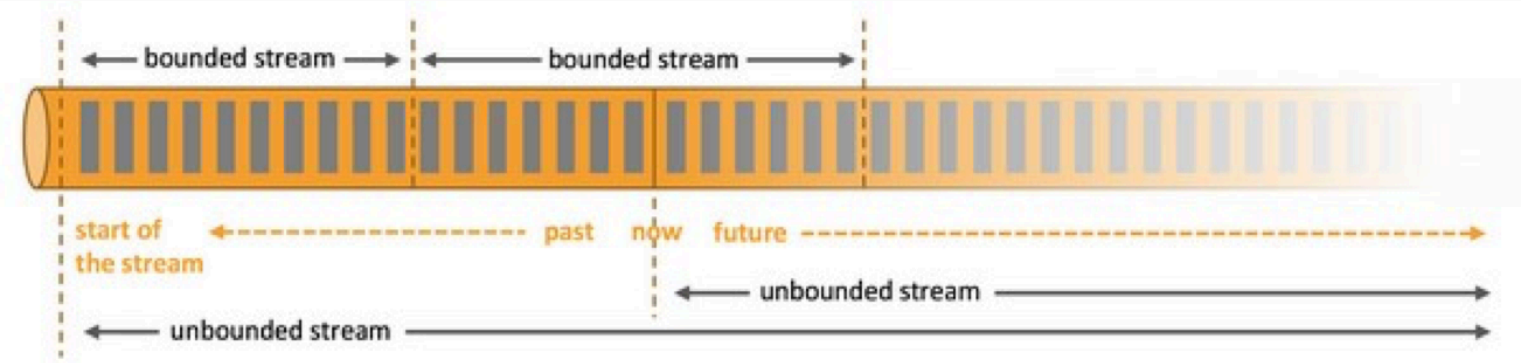


图 10: 流批统一

Flink 的主要特点

- Flink 的分层 API
  - 越顶层越抽象，表达含义越简明，使用越方便
  - 越底层越具体，表达能力越丰富，使用越灵活

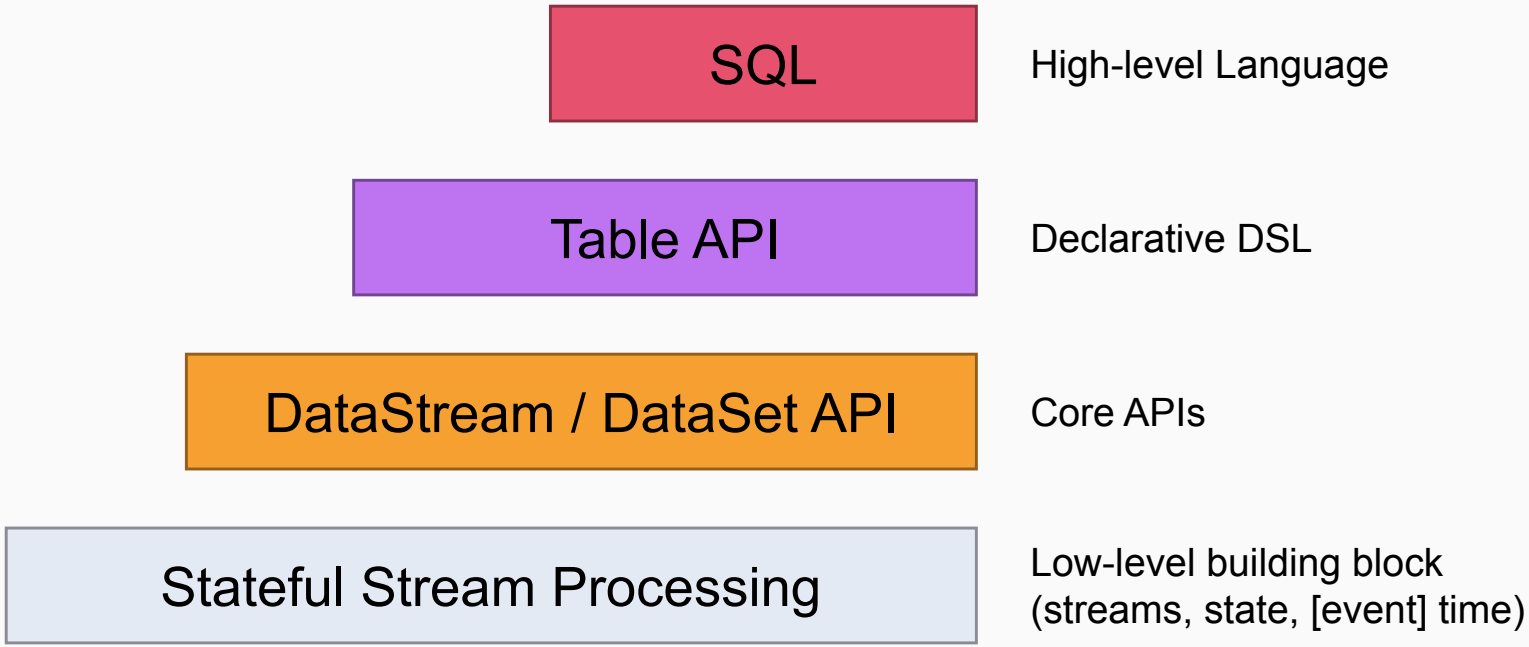


图 11: 分层 API

Flink 的其它特点

- 支持事件时间（event-time）和处理时间（processing-time）语义
- 精确一次（exactly-once）的状态一致性保证
- 低延迟，每秒处理数百万个事件，毫秒级延迟（实际上就是没有延迟）
- 与众多常用存储系统的连接（ES，HBase，MySQL，Redis…）
- 高可用（zookeeper），动态扩展，实现 7*24 小时全天候运行

Flink vs Spark Streaming

- 流（stream）和微批

Flink vs Spark Streaming

- 数据模型
  - Spark 采用 RDD 模型，Spark Streaming 的 DStream 实际上也就是一组组小批数据 RDD 的集合
  - Flink 基本数据模型是数据流，以及事件（Event）序列 (Integer、String、Long、POJO Class)
- 运行时架构
  - Spark 是批计算，将 DAG 划分为不同的 Stage，一个 Stage 完成后才可以计算下一个 Stage
  - Flink 是标准的流执行模式，一个事件在一个节点处理完后可以直接发往下一个节点进行处理

单词计数程序

```
1       public static void main(String[] args) throws Exception {
2           final StreamExecutionEnvironment env =
            ↪  StreamExecutionEnvironment.getExecutionEnvironment();
3           env.setParallelism(1);
4
5           DataStream<String> stream = env.fromElements("Hello
            ↪  World", "Hello World");
6
7           stream
8                   .flatMap(new Tokenizer())
9                   .keyBy(r -> r.f0)
10                  .sum(1)
11                  .print();
12
13          env.execute(" 单词计数");
14      }
```

单词计数程序

```
1       public static class Tokenizer implements
        ↪  FlatMapFunction<String, Tuple2<String, Integer>> {
2           @Override
3           public void flatMap(String value,
            ↪  Collector<Tuple2<String, Integer>> out) throws
            ↪  Exception {
4               String[] stringList = value.split("\\s");
5               for (String s : stringList) {
6                   out.collect(Tuple2.of(s, 1));
7               }
8           }
9       }
```

Q & A