第九章 批流融合基础

徐 辰
cxu@dase.ecnu.edu.cn
華東師範大學

大纲

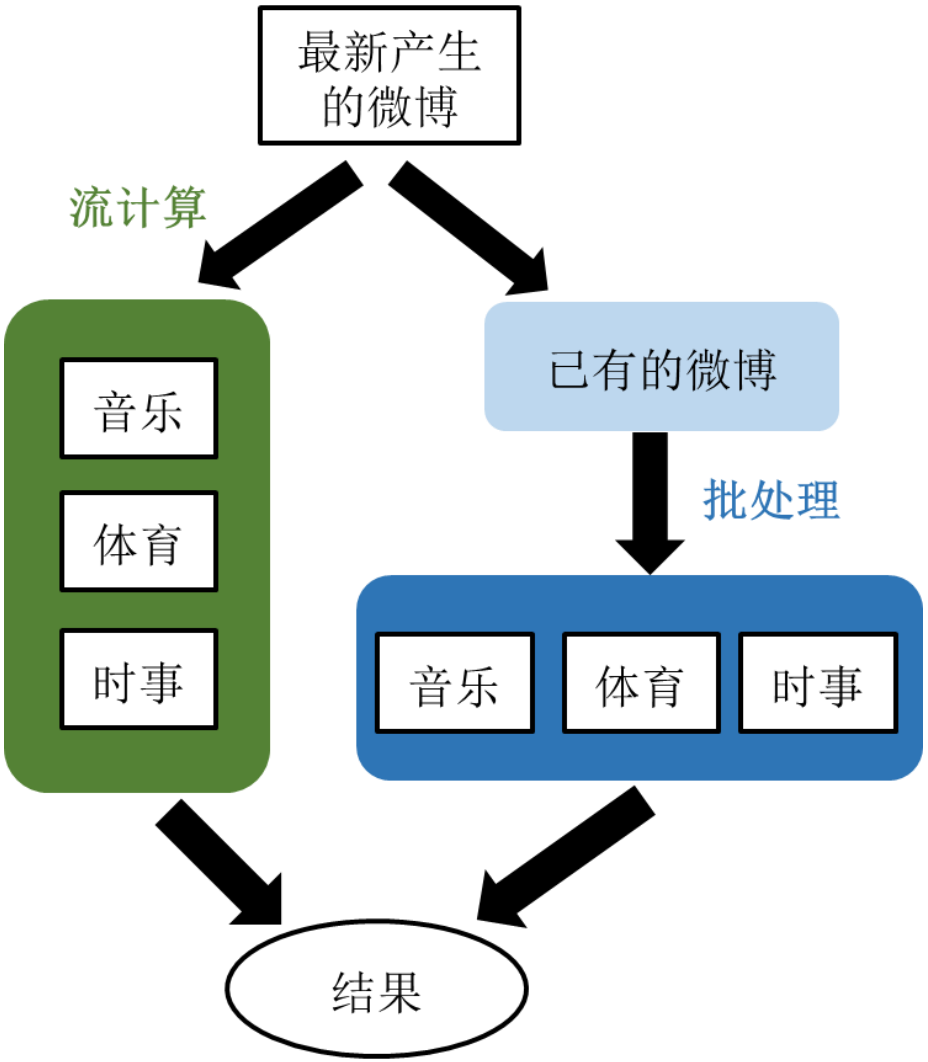
- 批流融合的背景
  - 应用需求
  - Lambda架构及其局限性
- 批处理与流计算的统一性
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎

批处理 vs. 流计算

- 批处理系统适合处理大批量数据、实时性要求不高的场景
- 流计算系统适合处理快速产生的数据、实时性要求高的场景
- 但是，同一场景可能既有大批量数据、又有快速产生的数据，某些模块实时性要求高，某些模块实时性要求低

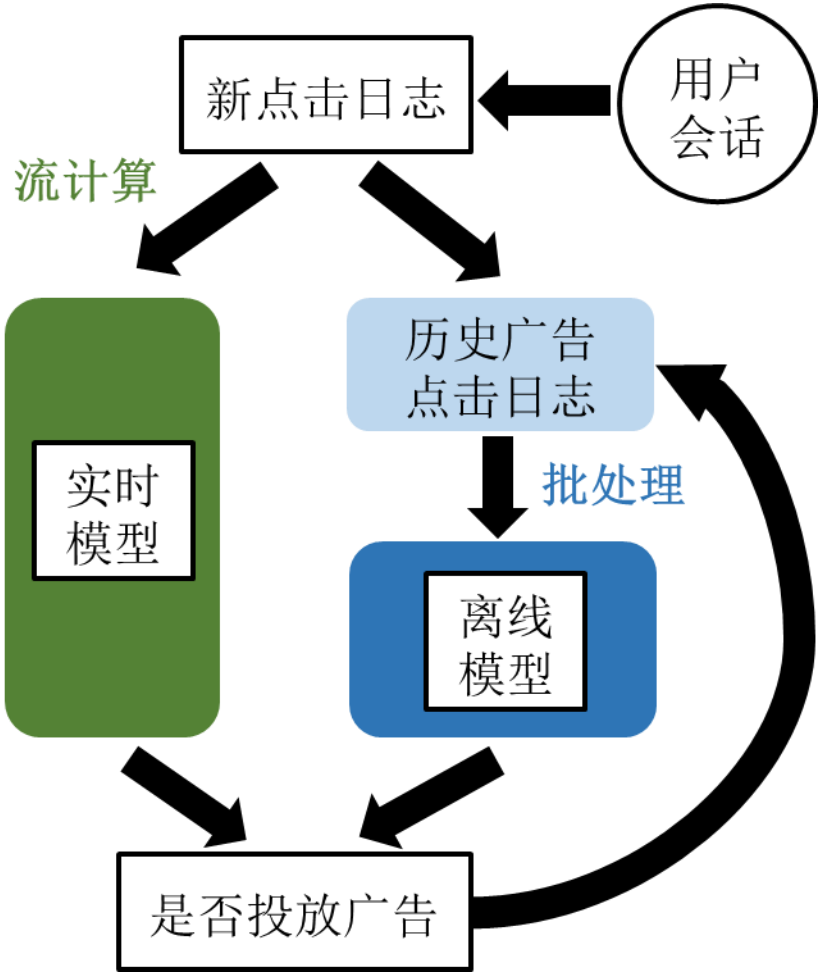
热点话题统计

- 根据已有的微博（例如，近一周）和最新产生的微博统计不同话题数量来判断热门程度



在线广告投放

- 模型训练：根据大量历史数据离线训练
- 模型预测：依据当前用户会话的数据进行决策，这些数据变成历史数据用于更新模型



Volume和Velocity

- 批处理系统：应对数据量大volume
- 流计算系统：应对数据产生速度快velocity
- 问题：如何同时应对volume和velocity?

大纲

- 批流融合的背景
  - 应用需求
  - Lambda架构及其局限性
- 批处理与流计算的统一性
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎

理想情况

- 理想状态下，任何针对数据的查询都可以用以下式子表达

$$Query = function(All\ Data)$$

- 如果数据达到相当大的一个级别（例如，PB级），那么查询难以得到快速响应，意味着满足实时查询需要消耗大量的资源

预先物化视图

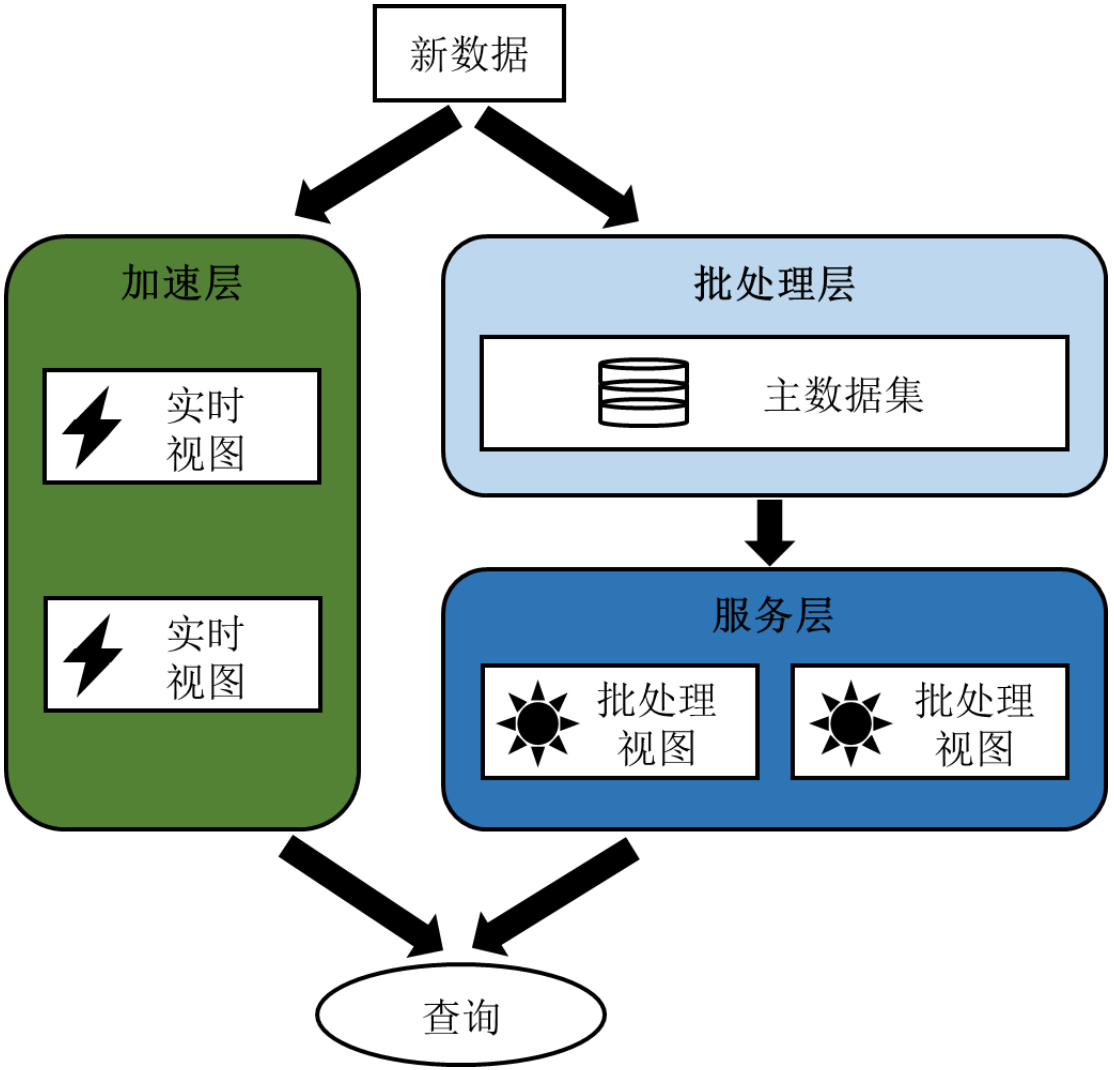
- 针对查询进行预先运算，得到批处理视图（Batch View）
  - 当需要执行查询时，从批处理视图中读取结果
  - 通过建立索引的方式支持随机读取，加快响应

$$Batch\ View = function_1(All\ Data)$$

$$Query = function_2(Batch\ View)$$

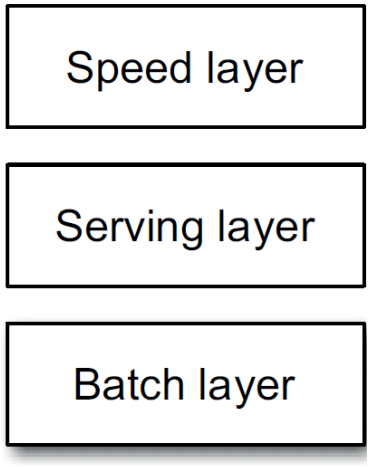
- 忽略了一个重要问题：数据往往是快速、动态增加的，使得批处理视图的结果存在一定的滞后性

Lambda架构



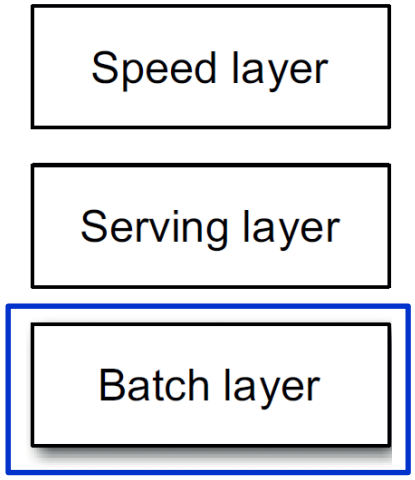
Lambda架构

- 批处理层（Batch Layer）：离线批量处理数据，生成批处理视图
- 加速层（Speed Layer）：实时处理新的数据，增量补偿批处理视图
- 服务层（Serving Layer）：响应用户的查询请求



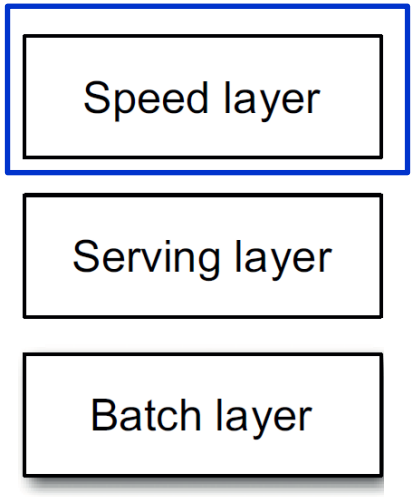
批处理层Batch Layer

- 功能：全量计算
  - 存储Master Dataset
    - 不可变、持续增长的数据集
  - 针对这个Master Dataset进行预运算
- Batch View = function$_1$(Master Dataset)



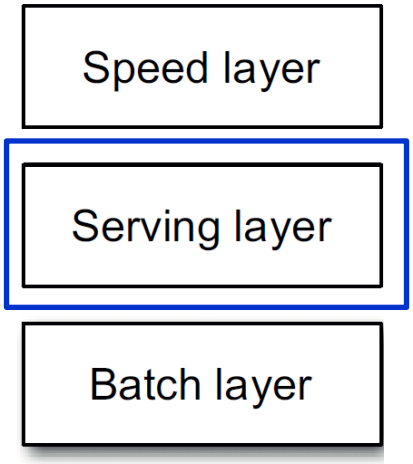
加速层Speed Layer

- 功能：增量计算
  - 只处理最近的数据，而不是所有数据
  - 使用增量算法针对批处理视图的陈旧结果进行补偿，从而降低查询延迟
- Real-time View = function$_2$(Real-time View, New Data)



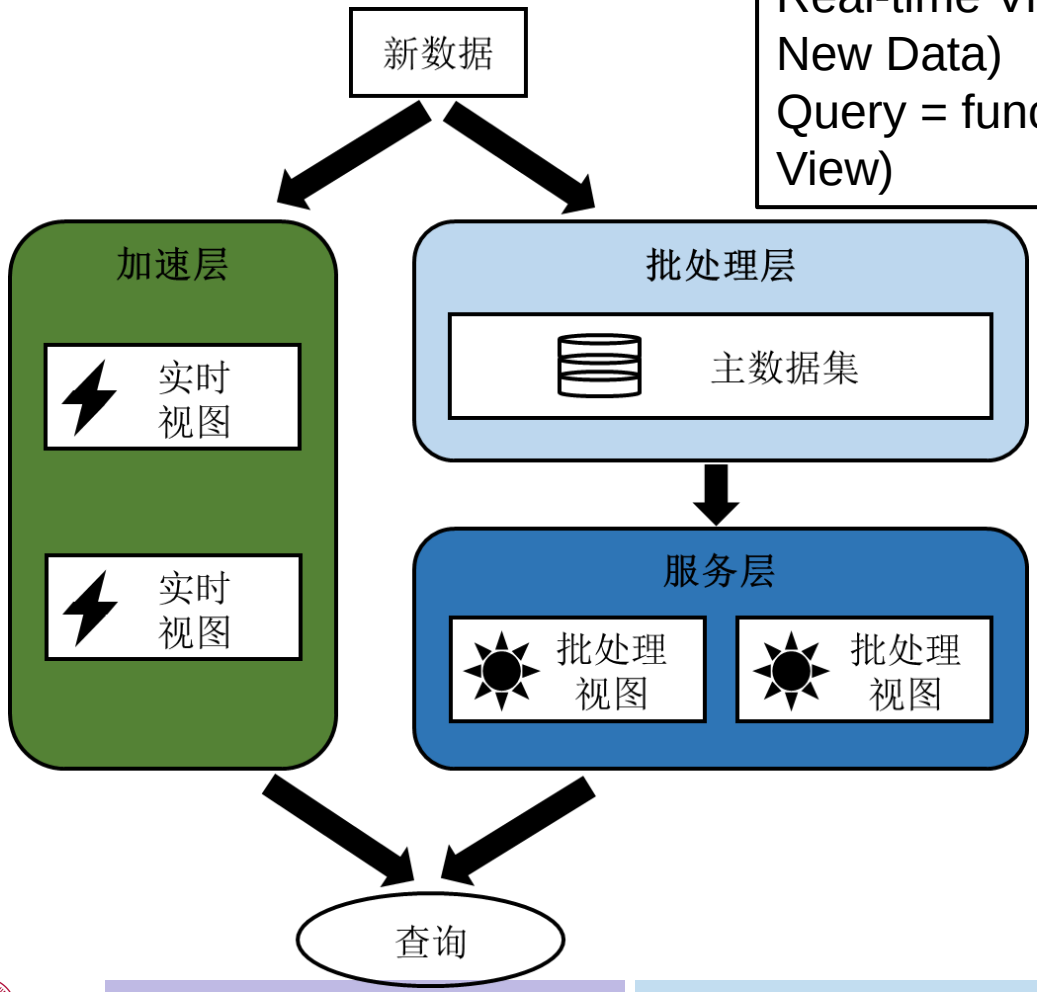
服务层Serving Layer

- 功能：通过合并批处理视图和实时视图最快地得出结果
- Query = function$_3$(Batch View, Real-time View)

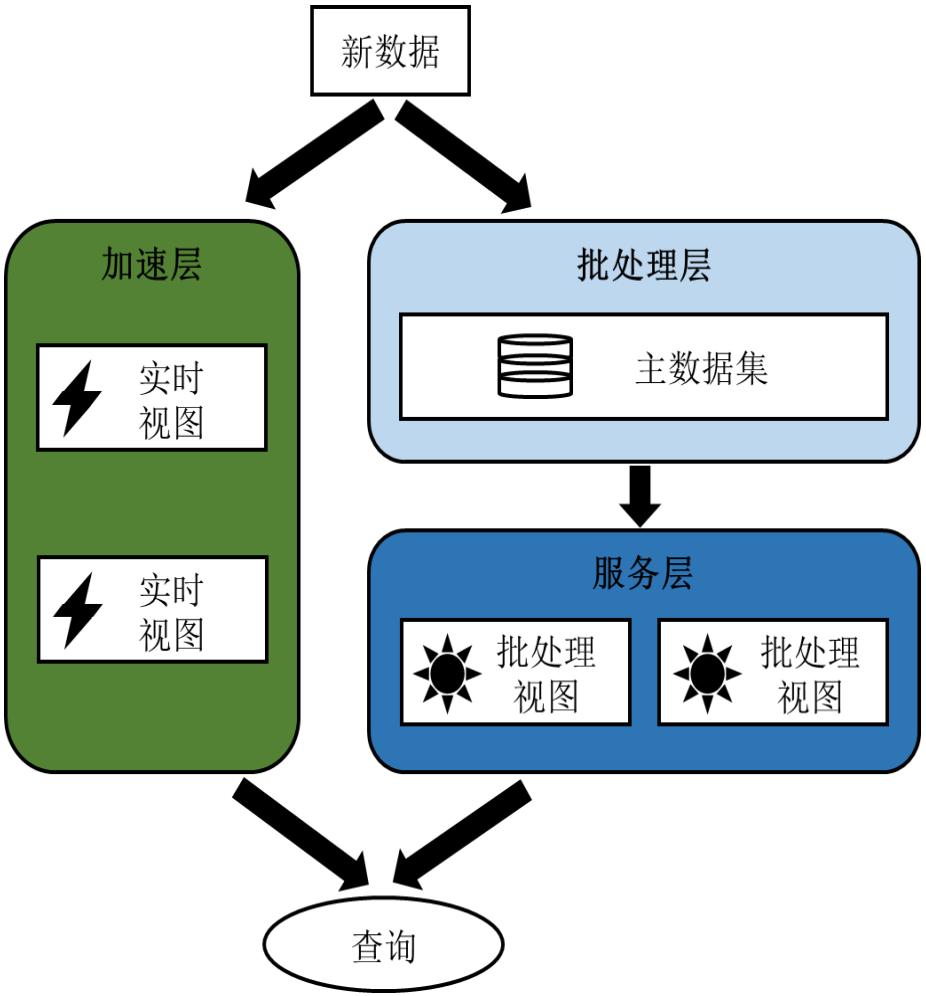


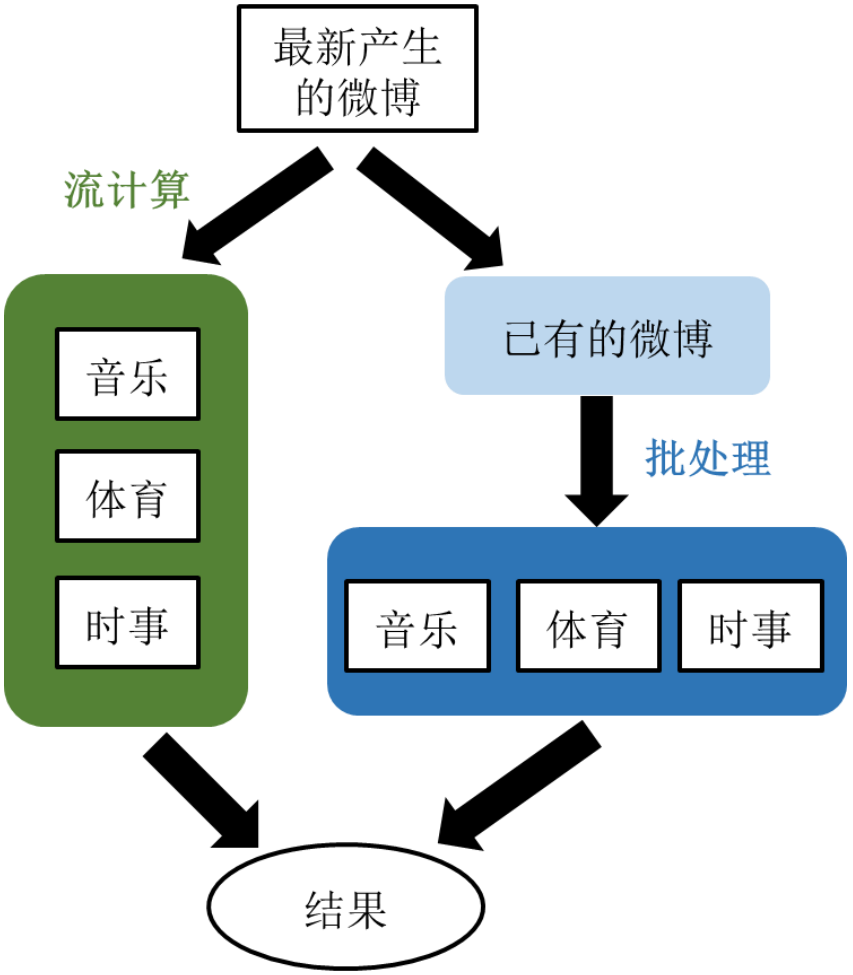
Lambda架构

Batch View = function$_1$(Master Dataset)
Real-time View = function$_2$(Real-time View, New Data)
Query = function$_3$(Batch View, Real-time View)



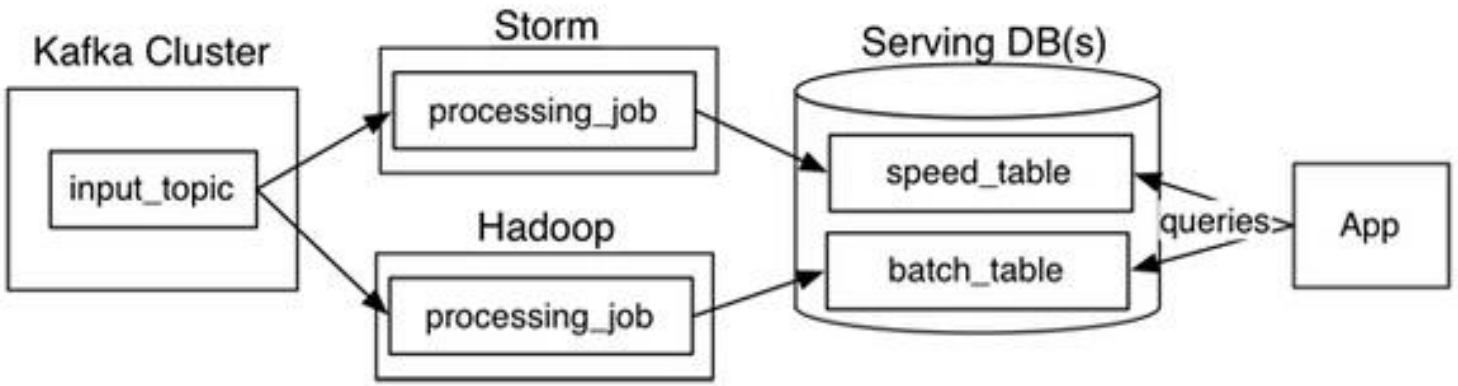
热点话题统计


新数据
加速层
实时
视图
实时
视图
批处理层
主数据集
服务层
批处理
视图
批处理
视图
查询


最新产生
的微博
流计算
音乐
体育
时事
已有的微博
批处理
音乐
体育
时事
结果

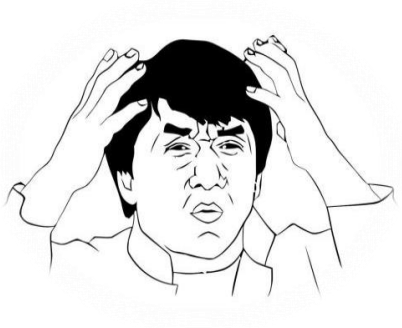
系统选型

- Lambda架构只是个架构
- 批处理层：批处理系统
- 加速层：流计算系统
- 服务层：数据库



Lambda架构优缺点

- 优点：
  - 平衡了重新计算高开销和需求低延迟的矛盾
- 缺点：
  - 开发复杂：需要将所有的算法实现两次，批处理系统和实时系统分开编程，还要求查询得到的是两个系统结果的合并
    - 不同编程模型
    - 编写两套代码
    - 潜在的bugs变多
  - 运维复杂：同时维护两套执行引擎



融合的思路

- 如何将批流系统分开编程变为统一编程?
  - Google Dataflow
  - Apache Beam

- 如何将两套执行引擎变为一体化的批流融合执行引擎并支持统一编程?
  - Spark Structured Streaming
  - Apache Flink

大纲

- 批流融合的背景
- 批处理与流计算的统一性
  - 有界数据集与无界数据集
  - 窗口操作
  - 时间域
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎

批处理 vs. 流计算引擎

- 问题：批处理引擎仅能处理批数据，流计算引擎仅能处理流数据，对吗？

执行引擎 \ 数据	批数据	流数据
Batch	✓	?
Streaming	?	✓

有界 vs. 无界数据集

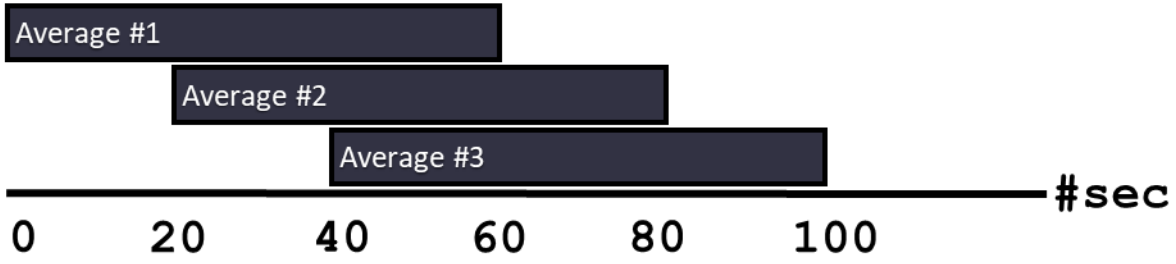
- 数据： bounded/unbounded
  - 有界数据集意味着系统处理的数据是有限的，对应于我们之前所说的批数据
  - 无界数据集意味着系统处理的是无限的数据，对应于我们之前所说的流数据
- 处理引擎： Batch/Streaming

执行引擎 \ 数据	bounded	unbounded
Batch	✓	?
Streaming	?	✓

大纲

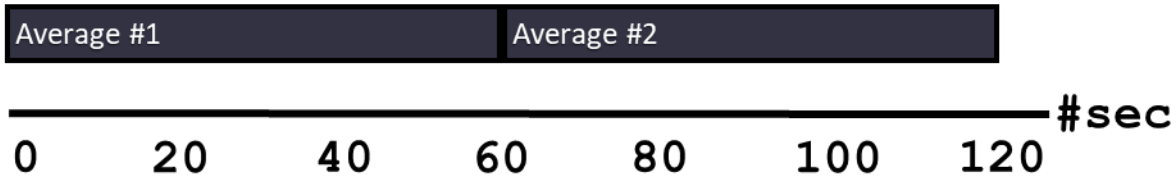
- 批流融合的背景
- 批处理与流计算的统一性
  - 有界数据集与无界数据集
  - 窗口操作
  - 时间域
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎

窗口类型



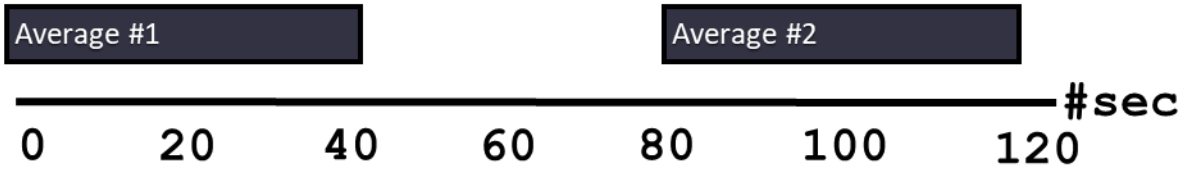
Sliding　　*range > slide*

滑动窗口



Tumbling　　*range = slide*

滚动窗口

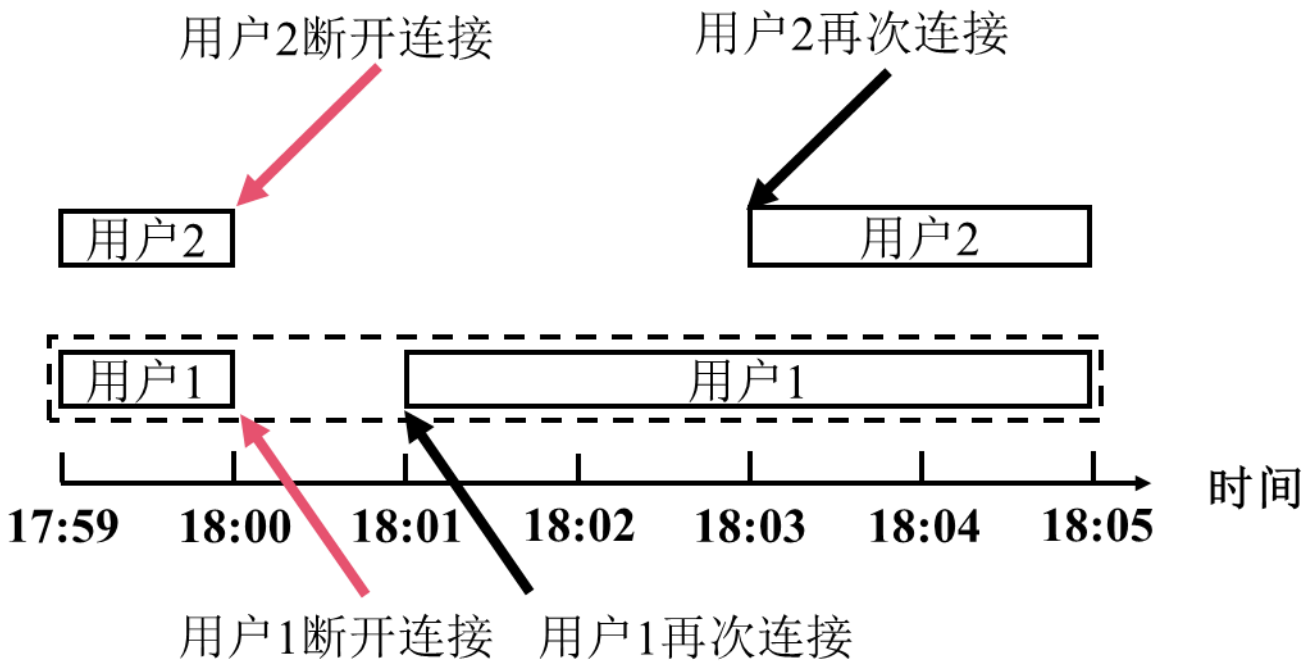


Jumping　　*range < slide*

跳跃窗口

会话窗口

- 会话超时时长：2分钟



窗口与数据集

- 窗口操作可以把数据集切分为有限的数据片以便于针对该数据片进行处理
  - 对于无界数据集来说，诸如聚合（aggregate）等操作需要窗口来定义边界，例如最近1分钟等；另一些则不需要（如filter，map等）
  - 对于有界数据集来说，窗口是可选的，或者我们认为是有一个全局窗口（Global Window）涵盖有界数据集中的所有数据

大纲

- 批流融合的背景
- 批处理与流计算的统一性
  - 有界数据集与无界数据集
  - 窗口操作
  - 时间域
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎

时间域

- 我们将一条数据记录视为一个事件
  - 事件时间 Event Time：该事件实际发生的时间
    - 当该事件发生时，其所在系统（可能是传感器等数据源）的当前时间
  - 处理时间 Processing Time：一个事件在数据处理的过程中被数据处理系统观察到的时间
    - 当该事件被数据处理系统处理时，数据处理系统的当前时间
  - 一个记录的事件时间是永远不变的，但是处理时间随着该记录在系统中被各个节点处理时而持续变化

有界数据集的时间域

- 例子：系统日志分析
  - 假设Hadoop平台从学期开始到目前为止运行了两个月，产生了大量的日志文件
    - 日志中的每一行开头的时间（事件时间）
    - 现在（处理时间）统计error的数量
- 需求：统计每周出现各类故障及其数量
  - 目前为止，我们所学到的窗口是基于处理时间的，那么如何设置该需求的窗口？
    - 需要根据事件时间来确定窗口

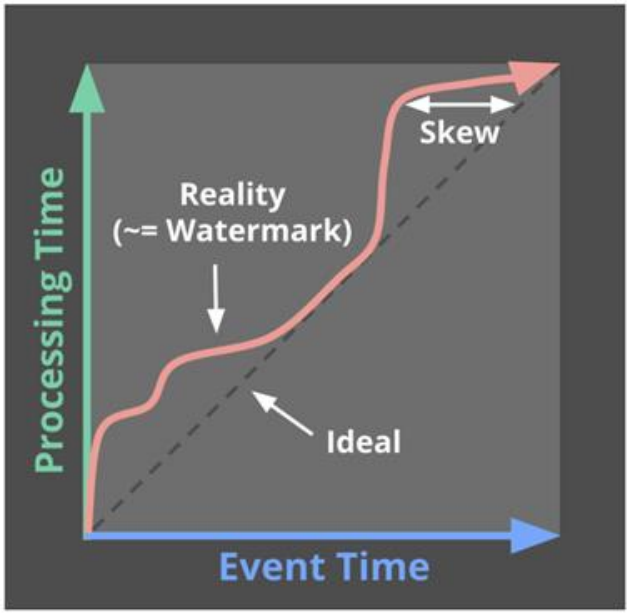
无界数据集的时间域

- 例子：传感器温度值统计
  - 传感器周期性(1-2s)采集温度信息，产生(time, temperature)的记录并向数据处理系统发送
  - 半小时统计一次最近1分钟的平均值：系统当前时间12:01:00，窗口[12:00:00-12:01:00)
- 假如：当前时间12:01:30(处理时间),传感器12:00:00(事件时间)产生的记录由于网络阻塞等原因才到达
  - 12:01:00（处理时间）触发计算的结果不准确
  - 到底还有没有其它记录没有到达?

水位线(watermark)

- 水位线所指示的事件时间表示早于该事件时间的记录已经完全被系统观察到
- 系统根据水位线“认定”当前事件时间域的所处时间

水位线是由系统根据预定义或用户指定的启发式规则生成的，因此“认定”实际上只是一种猜测

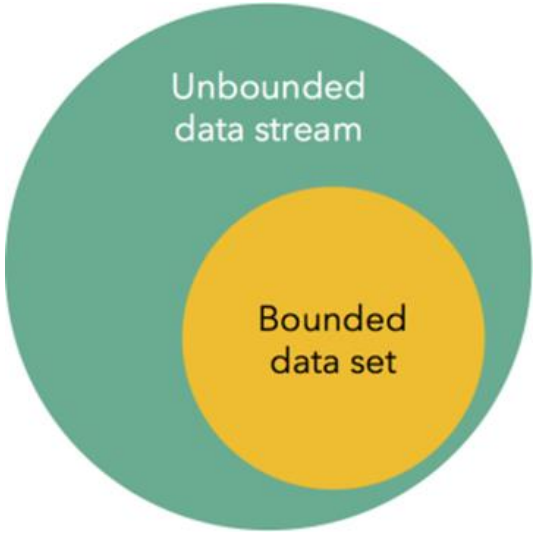


大纲

- 批流融合的背景
- 批处理与流计算的统一性
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎

Dataflow模型的观点

- 基本假设
  - 将所有输入数据均视为无界数据集
  - 有界数据集仅仅是无界数据集的一个特例
- 是否可以反过来?



无界数据集的特点

- 无界(Unbounded)：记录(Record)在数据处理系统执行计算过程中不断地动态到达并且永无止境
- 延迟(Delay)：由于网络传输等原因，记录从产生到进入数据处理系统，通常会产生延迟
- 乱序(Out-of-order)：由于数据记录从产生到进入系统产生的延迟可能不同，所以数据记录的原始顺序和进入系统的处理顺序可能不一致

WWWH模型

- 操作描述：需要对输入的数据执行什么(*What*)操作?

- 窗口定义：如何按记录的事件时间域定义窗口对输入数据进行划分，即在哪里(*Where*)进行数据切分?

WWWH模型

- 触发器：当输入数据无界时，由于输入数据存在延迟，系统在处理时间域何时(*When*)触发基于事件时间定义的窗口？

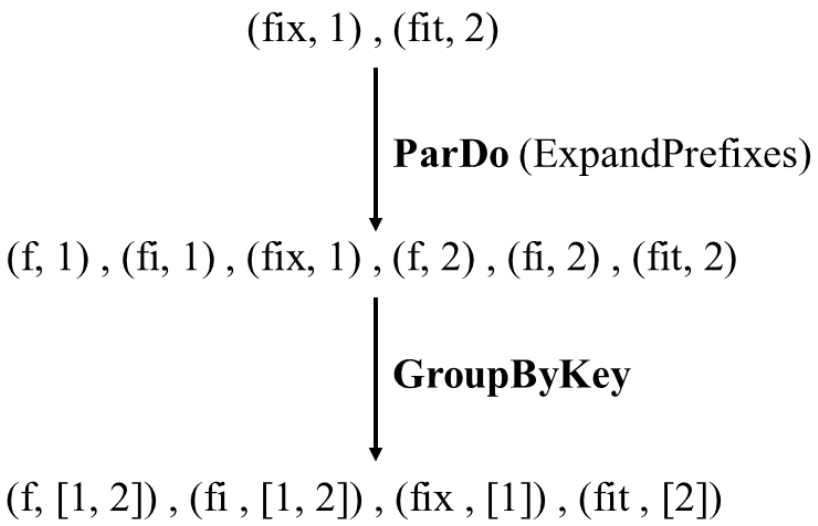
- 结果修正：当输入数据无界时，由于输入数据的乱序，系统触发窗口计算后仍然可能有迟到的数据抵达，那么如何(*How*)修正已经触发的窗口的计算结果？

大纲

- 批流融合的背景
- 批处理与流计算的统一性
- Dataflow统一编程模型
  - 操作描述
  - 窗口定义
  - 触发器
  - 结果修正
- 关系化Dataflow编程模型
- 一体化执行引擎

核心原语

- ParDo: 该操作对每个键值对都执行相同的处理，获得0或多个输出键值对
- GroupByKey: 用来按键把元素重新分组



大纲

- 批流融合的背景
- 批处理与流计算的统一性
- Dataflow统一编程模型
  - 操作描述
  - 窗口定义
  - 触发器
  - 结果修正
- 关系化Dataflow编程模型
- 一体化执行引擎

无界数据集与窗口

- 如果输入是无界数据集，那么无法确定何时才能收集到所有键值对

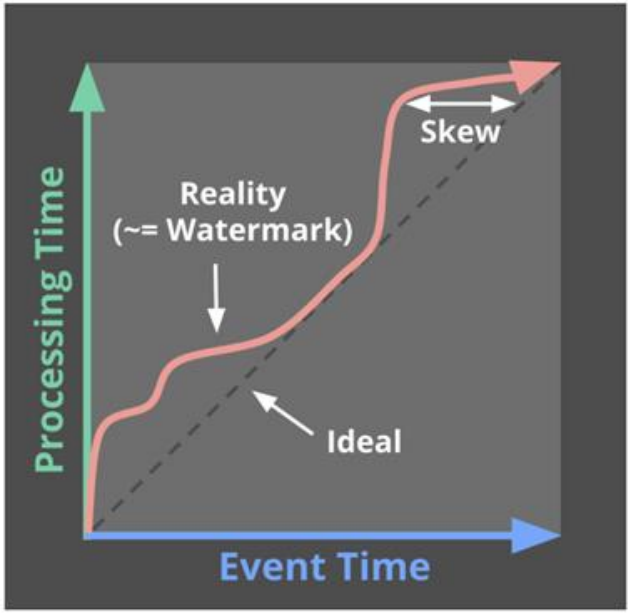
- 针对无界数据集的操作通常需要先定义窗口，在确定的窗口中执行操作

大纲

- 批流融合的背景
- 批处理与流计算的统一性
- Dataflow统一编程模型
  - 操作原语
  - 窗口表达
  - 触发器
  - 结果修正
- 关系化Dataflow编程模型
- 一体化执行引擎

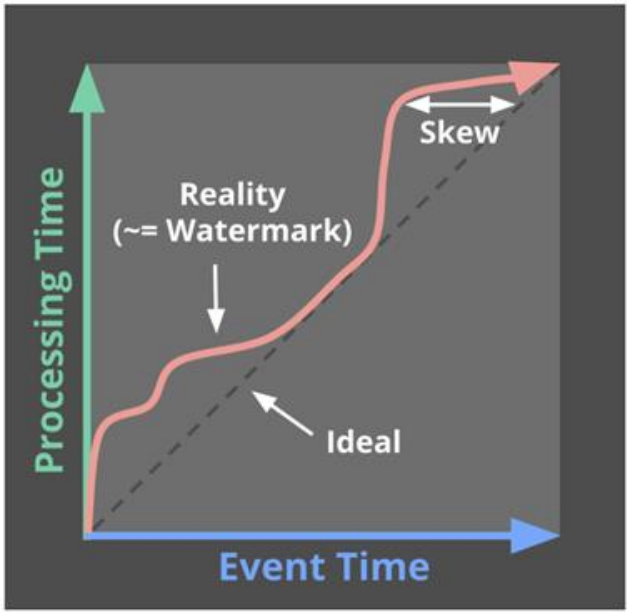
触发器

- 在某一处理时间决定处理窗口的聚合结果并输出
- 窗口的语义要求根据事件时间
- 触发器需要利用水位线



水位线过慢

- 水位线是对事件时间的猜测，可能过慢
  - 窗口操作结束前，需要提前触发窗口操作并输出结果
  - 例如：再定义一个触发器，每隔1min（处理时间）触发窗口计算

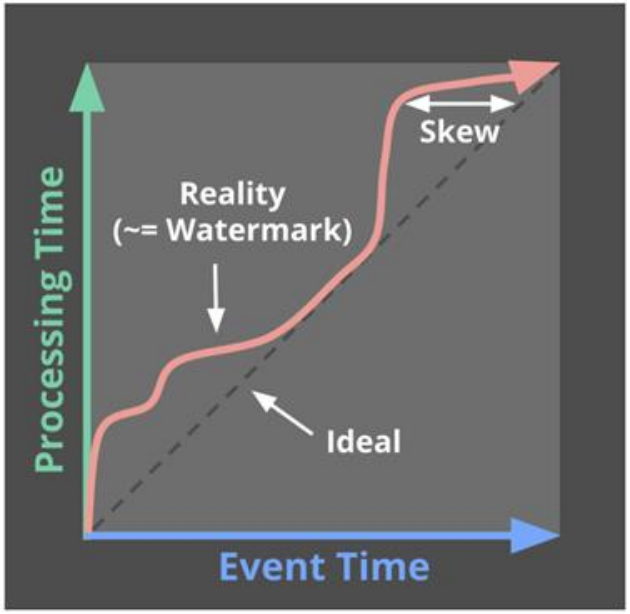


大纲

- 批流融合的背景
- 批处理与流计算的统一性
- Dataflow统一编程模型
  - 操作原语
  - 窗口表达
  - 触发器
  - 结果修正
- 关系化Dataflow编程模型
- 一体化执行引擎

水位线过快

- 水位线是对事件时间的猜测，可能过快
  - 存在“迟到”的数据
  - 需要之后修正原有窗口计算结果，再次触发窗口操作并输出结果



修正方式

- 如何管理当前的窗口内容
  - 抛弃(Discarding)：触发器一旦触发后，窗口内容即被抛弃，之后窗口计算的结果和之前的结果不存在任何相关性
  - 累积(Accumulating)：触发器触发后，窗口内容进行持久化，而新得到的结果成为对已输出结果的一个修正版本
  - 累积和撤回（Accumulating & Retracting）：触发器触发后，不仅将窗口内容持久化，还需记录已经输出的结果。当窗口将来再触发时，先撤回已输出的结果，然后输出新得到的结果

WWWH模型

- ***What*** results are calculated?
  - 计算什么结果? (read, map, reduce)

操作描述

- ***Where*** in event time are results calculated?
  - 在哪儿切分数据? (event time window)

窗口定义

- ***When*** in processing time are results materialized?
  - 什么时候计算数据? (triggers)

触发器

- ***How*** do refinements of results relate?
  - 如何修正相关的数据? (Accumulation)

结果修正

大纲

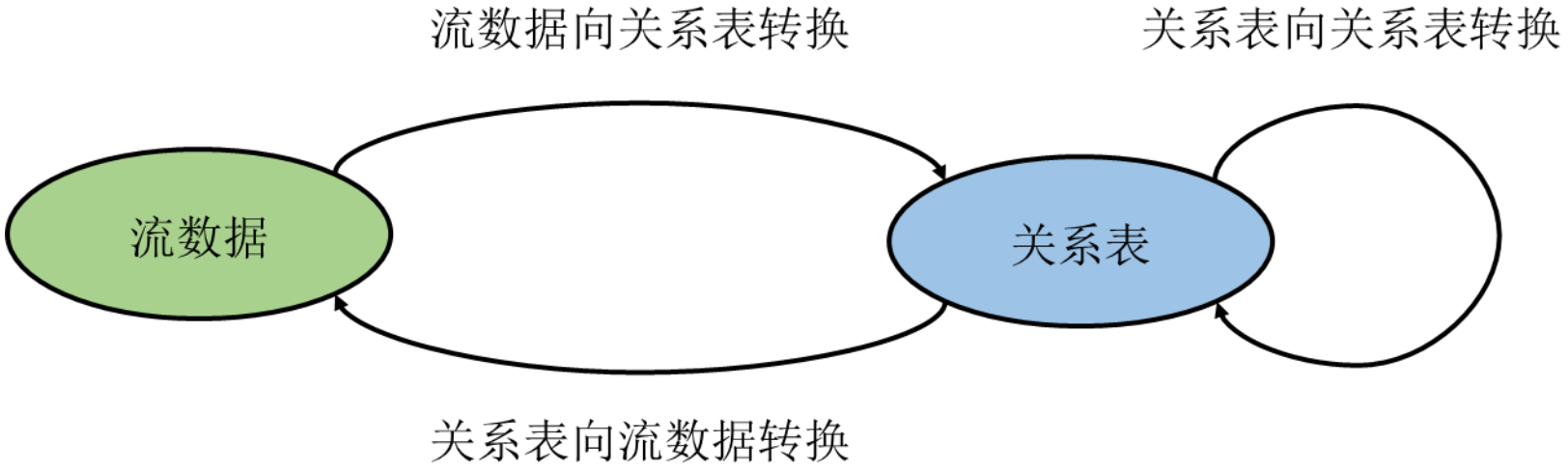
- 批流融合的背景
- 批处理与流计算的统一性
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎

Dataflow编程模型的关系化

- Dataflow编程模型认为输入数据是一系列元组组成的序列
- 从关系数据模型的角度来看，如果将键值对看作元组，那么这一系列的键值对可以构成关系表
- 此时需要提供的操作不再是如何对键值对进行转换，而是针对关系表的SQL

连续查询

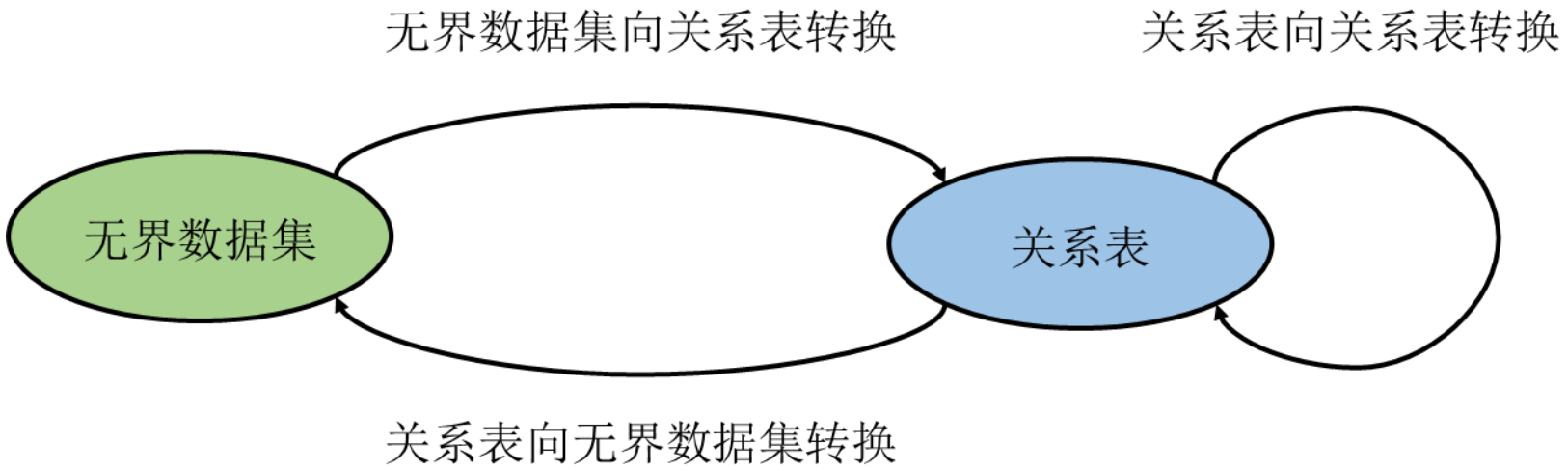
- 流数据转换为关系表
- 在某一时刻针对关系表进行SQL操作得到新的关系表
- 再将新的关系表相应地转换为流数据



The CQL continuous query language: semantic foundations and query execution. VLDB J 2006.

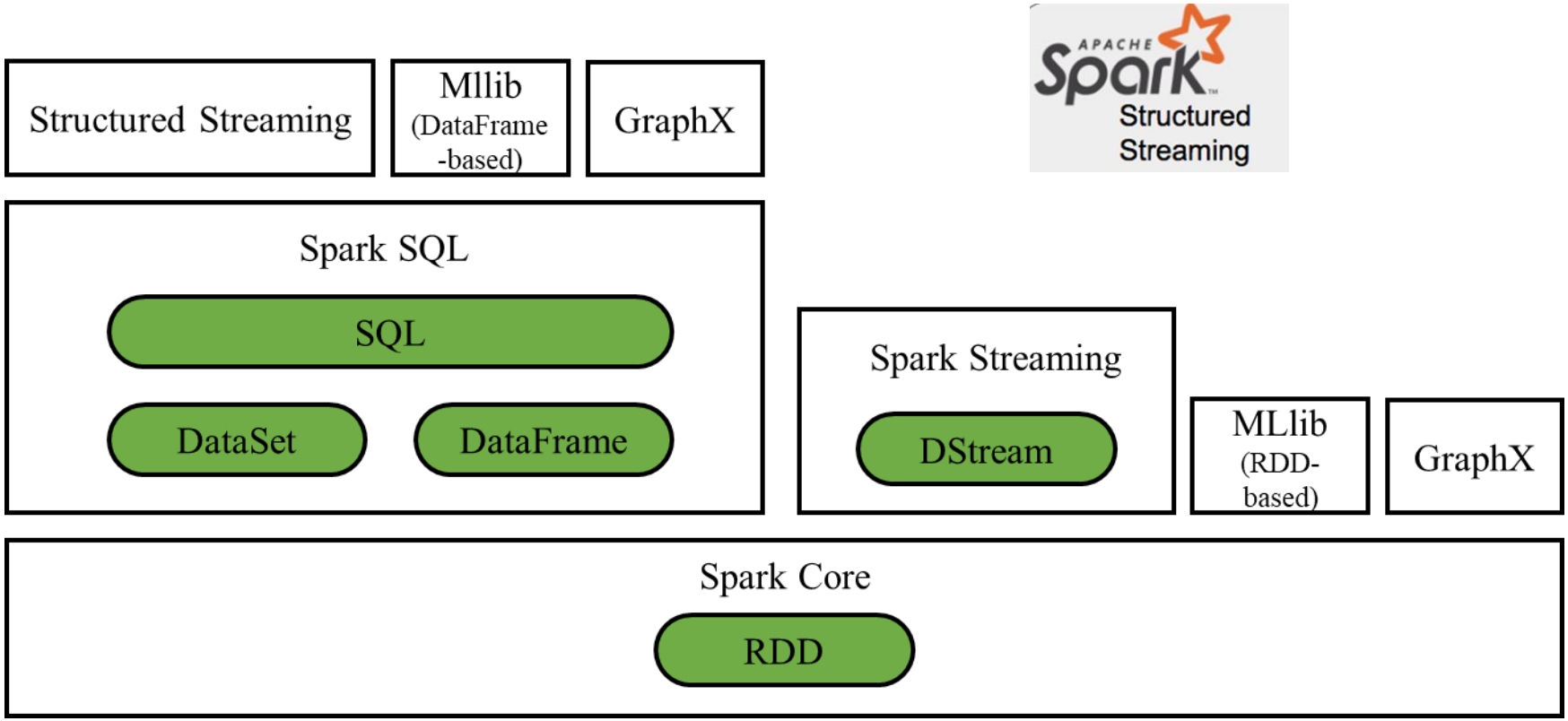
关系化的Dataflow编程模型

- 将输入的无界数据集转化为关系表
- 由于该关系表是动态变化的，在触发器定义的时刻针对关系表进行SQL操作得到新的关系表
- 再将新的关系表相应地转换为无界数据集



Structured Streaming

- Structured Streaming实现了关系化的Dataflow编程模型



大纲

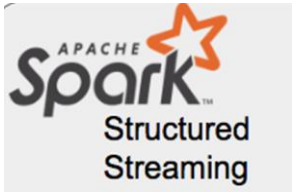
- 批流融合的背景
- 批处理与流计算的统一性
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎

融合的思路

- 如何将批流系统分开编程变为统一编程?
  - Google Dataflow
  - Apache Beam

多个不同执行引擎之间批处理与流计算编程模型进行统一

- 如何将两套执行引擎变为一体化的批流融合执行引擎并支持统一编程?
  - Spark Structured Streaming
  - Apache Flink



在同一个执行引擎中实现批处理与流计算的统一编程

执行引擎一体化的思路

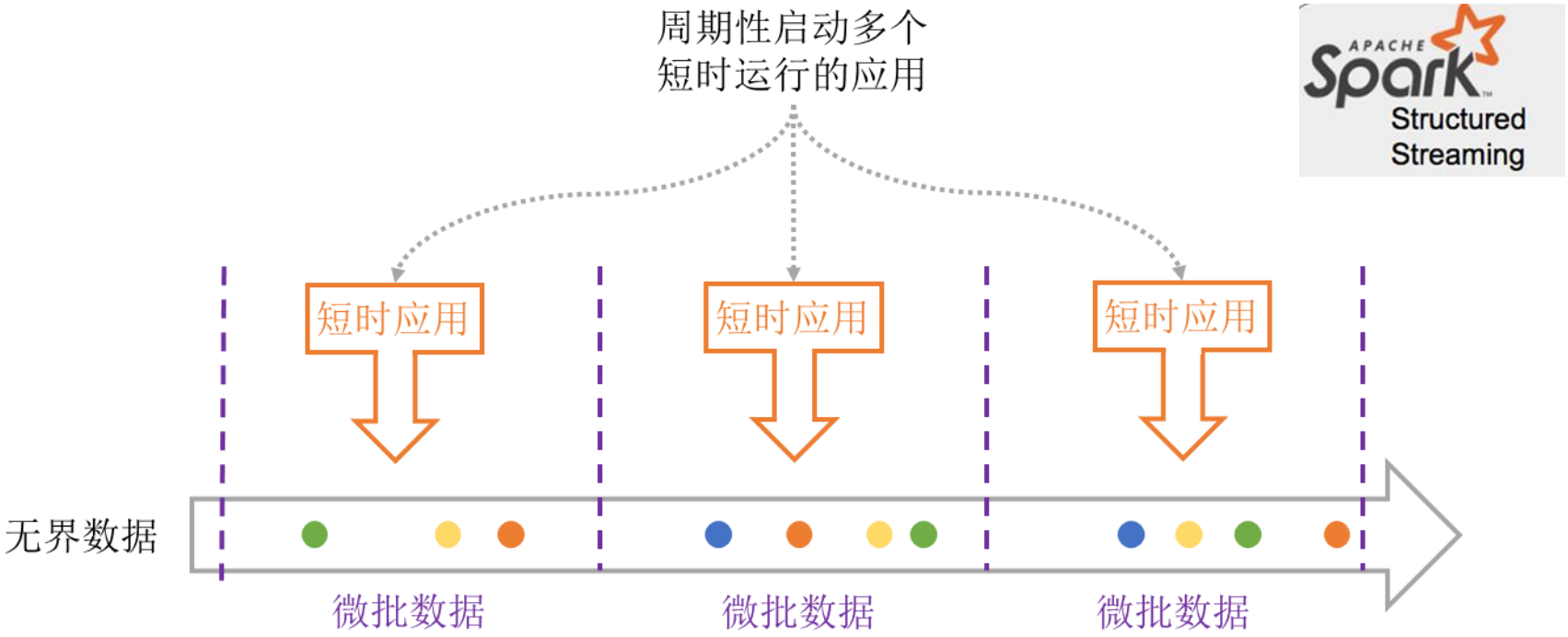
- 当前通用的核心执行引擎是批处理引擎或流计算引擎，因此一体化执行引擎需要选择其中一种作为核心
  - 以批处理为核心：利于处理有界数据集，待解决的问题是如何基于批处理来处理无界数据集
  - 以流计算为核心：利于处理无界数据集，待解决的问题是如何基于流计算来处理有界数据集

大纲

- 批流融合的背景
- 批处理与流计算的统一性
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎
  - 以批处理为核心
  - 以流计算为核心

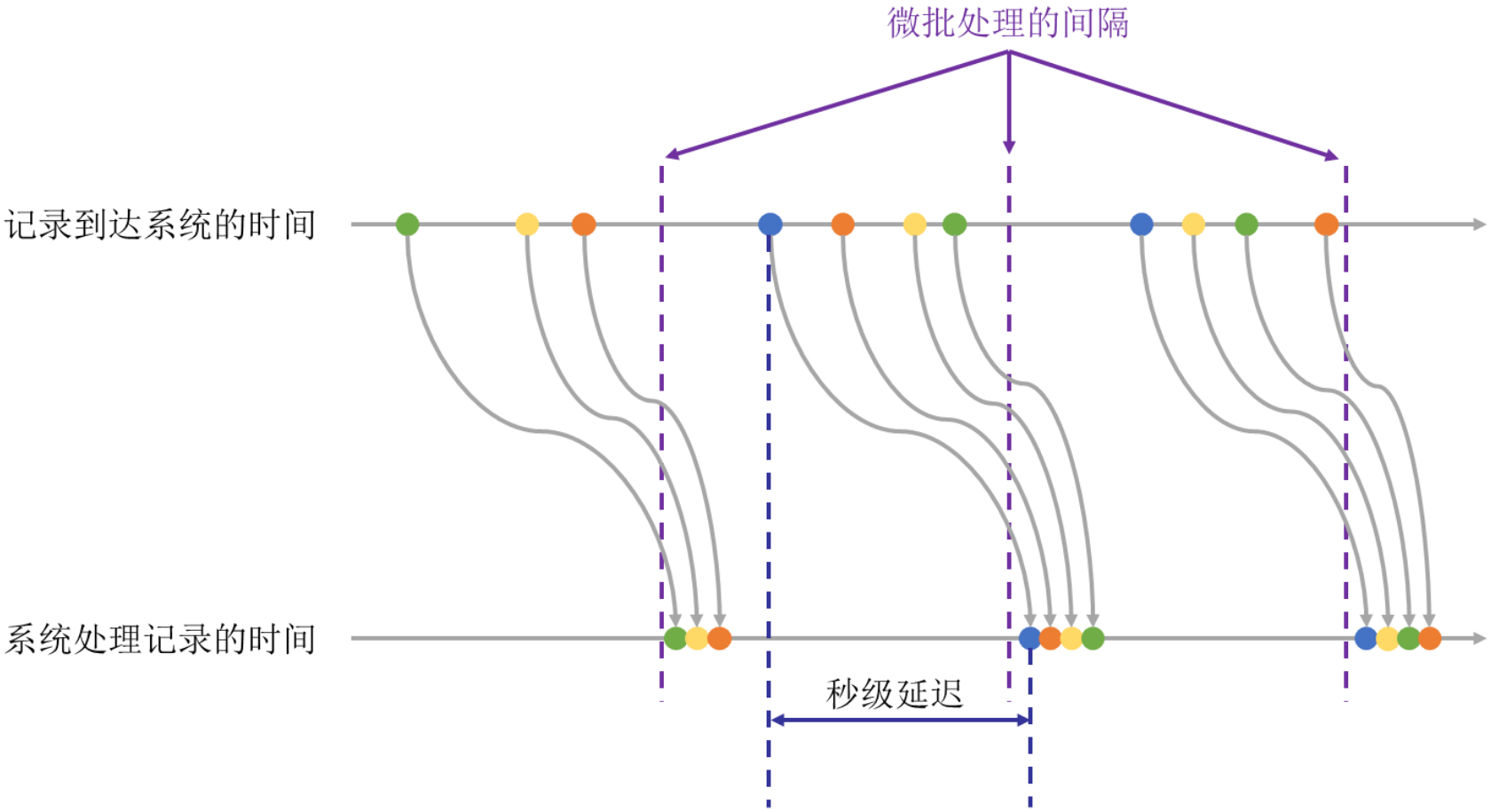
以批处理为核心：微批处理

- 将无界数据集划分为小批量数据，不断地启动短作业来处理这些小批量数据
- 先启动的作业必须执行完，才启动新作业



微批处理的延迟

- 串行执行短作业的方式带来的延迟在秒级



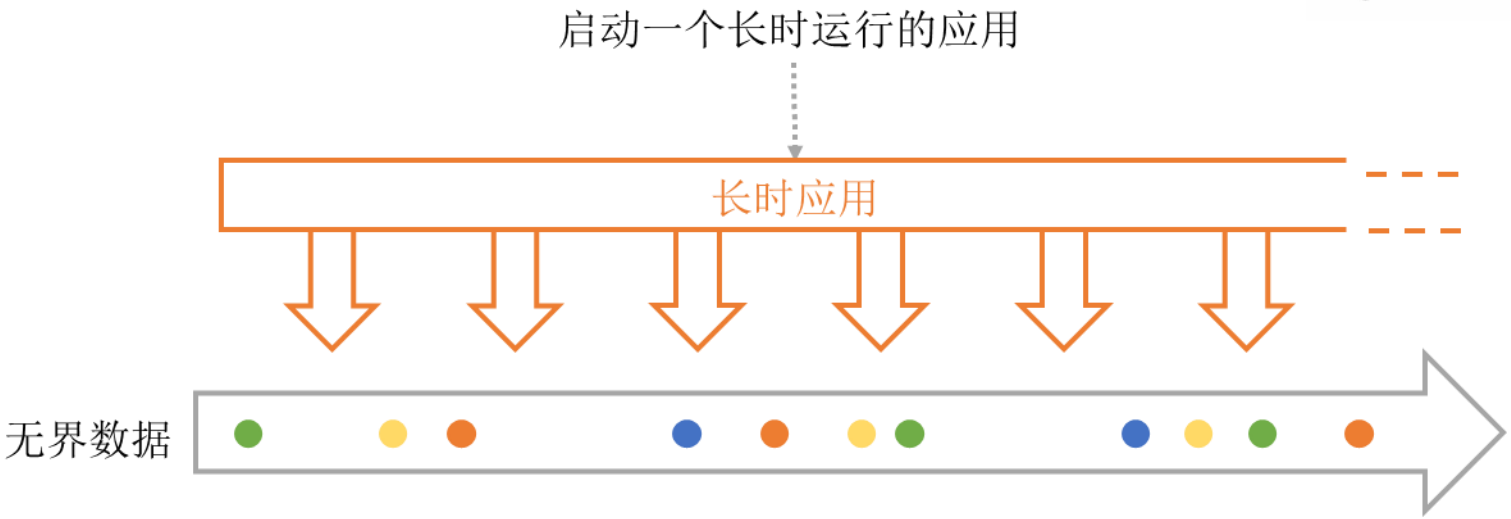
大纲

- 批流融合的背景
- 批处理与流计算的统一性
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎
  - 以批处理为核心
  - 以流计算为核心

以流计算为核心：连续处理

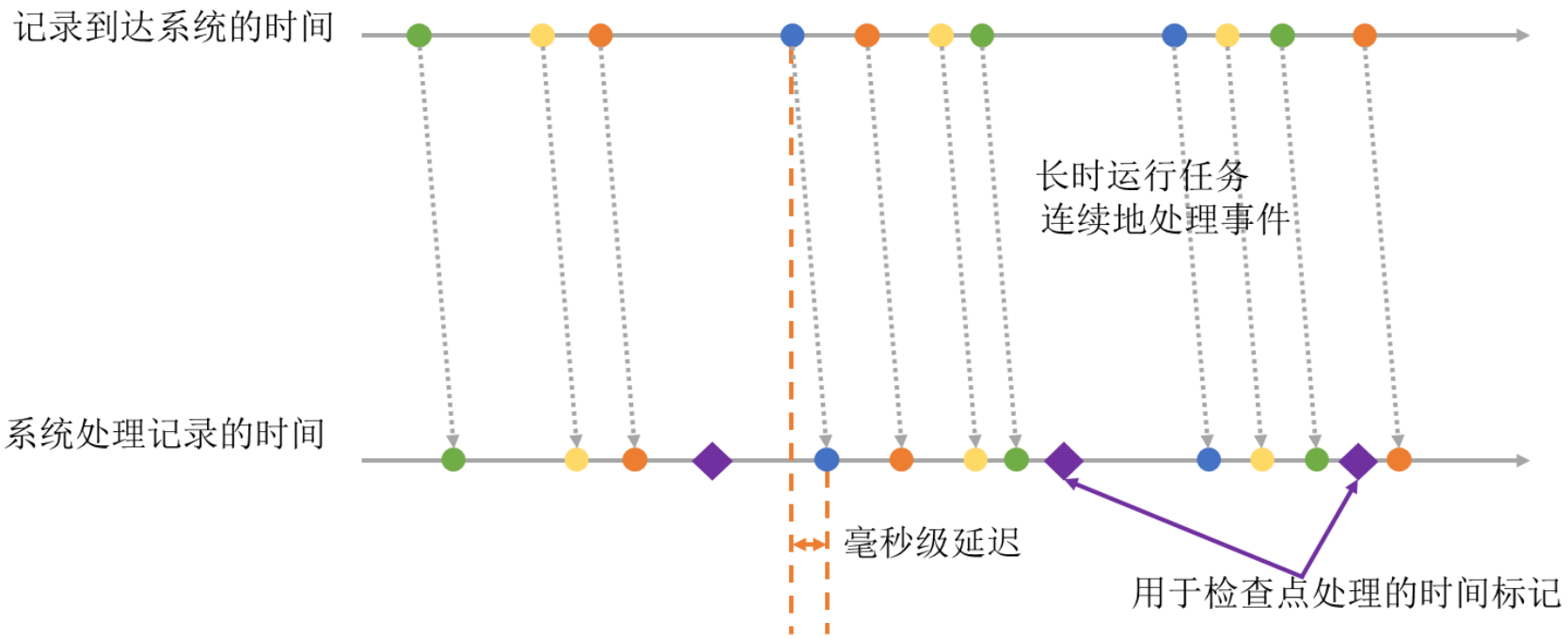
- 对于有界数据集来说，相当于系统接收一定量的记录之后就不再接收新的记录了
- 启动一个长期运行的作业

Apache Flink



连续处理的延迟

- 记录一旦进入系统将立即得到处理，而不是像微批处理方式中那样等到一批数据完全获取并且前一批短作业结束后才会处理



课后阅读

学术论文

- Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fer Andez-Moctezuma, R. J., Lax, R., …, S. W. (2015). The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing. PVLDB, 8(12), 1792–1803.
- Armbrust, M., Das, T., & Torres, J. (2018). Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark. In SIGMOD Conference (pp. 465–476).

课后阅读

- 企业应用
  - Google Cloud、PayPal、Talend
    https://www.sohu.com/a/141488109_470008
  - Twitter
    https://zhuanlan.zhihu.com/p/106676174

本章小结

- 批流融合的背景
- 批处理与流计算的统一性
- Dataflow统一编程模型
- 关系化Dataflow编程模型
- 一体化执行引擎