深入探索 Flink SQL 流批统一的查询引擎与最佳实践

Deep Dive into Unified Query Processor of Flink SQL and Best Practice

伍翀（云邪）
PMC & Committer @ Apache Flink
技术专家 @ 阿里巴巴

李劲松（之信）
Committer @ Apache Beam
技术专家 @ 阿里巴巴

FLINK FORWARD # ASIA
实时即未来 # Real-time Is The Future

FLINK FORWARD

About me

- 伍翀（云邪，Jark）
- Apache Flink Committer since 2017.02
- Apache Flink PMC since 2019.11
- 阿里巴巴 Blink SQL 团队

Contents
目录

01 Flink SQL 架构

Flink SQL Architecture

02 深入探索 Flink SQL 流处理

Deep Dive into Flink SQL Stream Processing

03 深入探索 Flink SQL 批处理

Deep Dive into Flink SQL Batch Processing

04 未来计划

Roadmap

Apache Flink 架构

Apache Flink Architecture

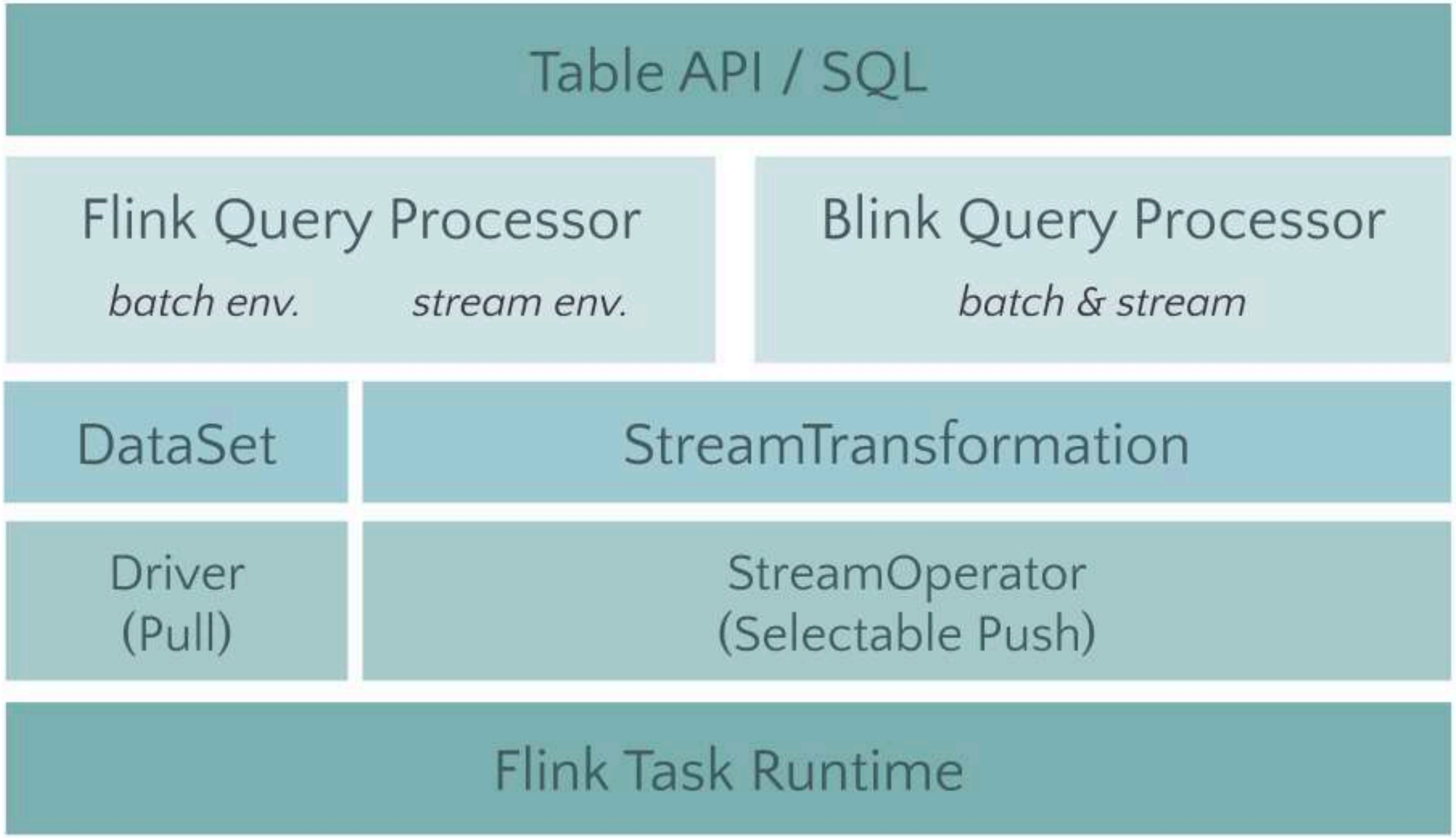


Table API / SQL

Flink Query Processor

batch env. *stream env.*

Blink Query Processor

batch & stream

DataSet

StreamTransformation

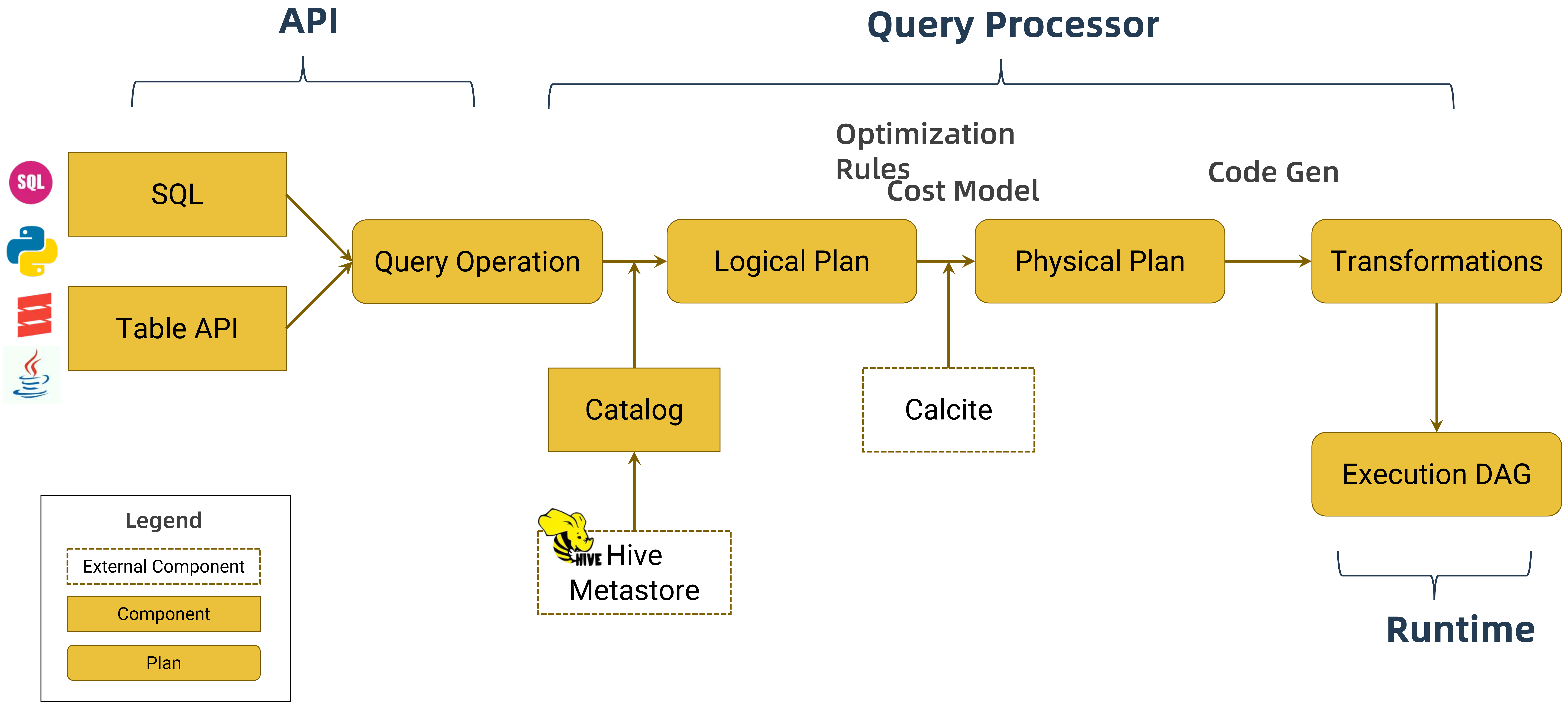
Driver
(Pull)

StreamOperator
(Selectable Push)

Flink Task Runtime

Flink SQL 工作流程

Flink SQL Workflow



深入探索 Flink SQL 流处理

Deep Dive into Flink SQL Stream Processing

02

统计每小时全网的访问量?

Count PV/hour of the entire site?

统计每小时全网的访问量? 窗口聚合

Count PV/hour of the entire site? Window Aggregation

FLINK FORWARD

```
CREATE TABLE user_log (
  user_id STRING,
  content STRING,
  ts TIMESTAMP(3),
  WATERMARK FOR ts AS ts - INTERVAL '2' SECOND
) with (
 'connector.type' = 'kafka',
 'connector.version' = 'universal',
 'format.type' = 'json',
 ...
)

SELECT
  TUMBLE_START(ts, INTERVAL '1' HOUR) hour
  COUNT(*) AS pv
FROM user_log
GROUP BY TUMBLE(ts, INTERVAL '1' HOUR)
```

Specify rowtime attribute

Specify watermark generation expression

watermark statement (available in 1.10)

延迟高!

High Latency!

统计每小时全网的访问量？常规聚合

Count PV/hour of the entire site? Regular Aggregation

```
SELECT
  TUMBLE_START(ts, INTERVAL '1' HOUR) hour,
  COUNT(*) AS pv
FROM user_log
GROUP BY TUMBLE(ts, INTERVAL '1' HOUR)
```

```
SELECT
  DATE_FORMAT(ts, 'yyyy-MM-dd HH') hour,
  COUNT(*) AS pv
FROM user_log
GROUP BY DATE_FORMAT(ts, 'yyyy-MM-dd HH')
```

Window Aggregate

延迟高！
High latency!

Regular Aggregate

吞吐低!
Low throughput!

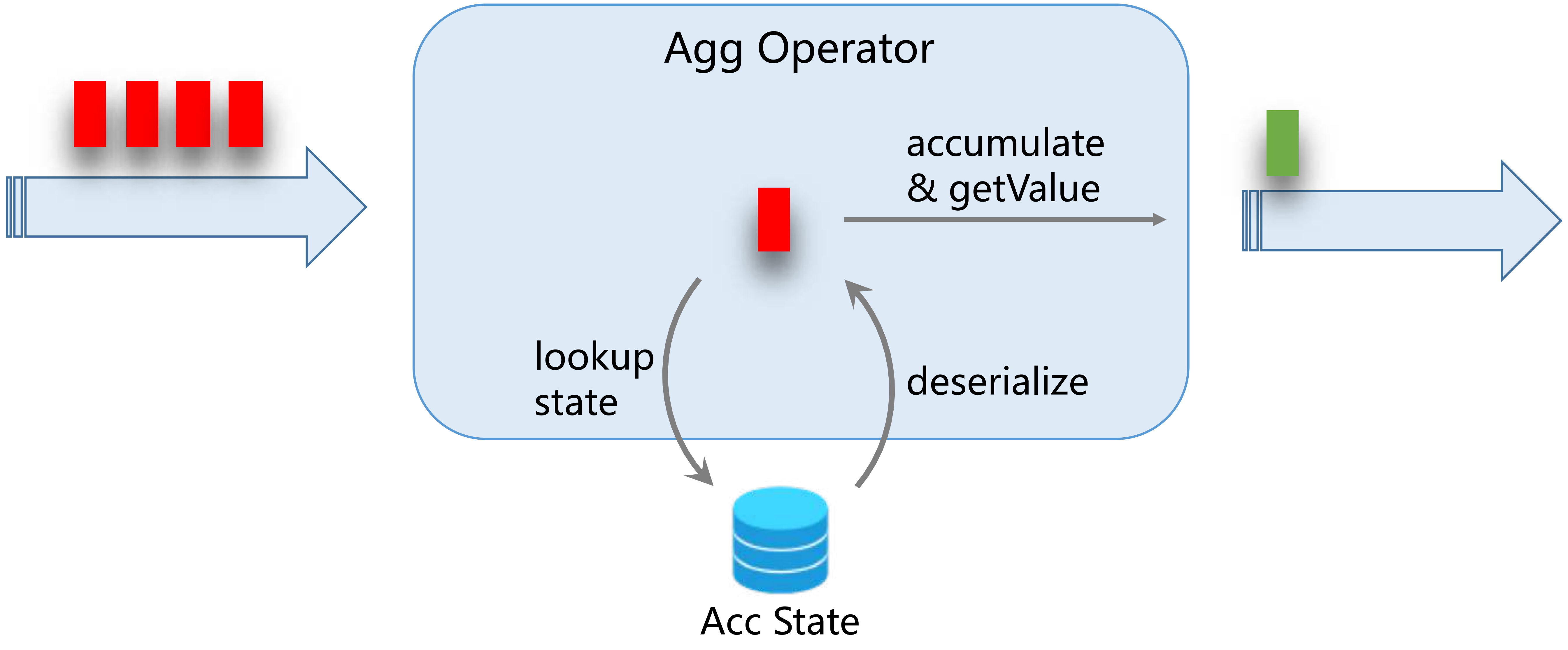
输出压力大
High output pressure!

统计每小时全网的访问量？常规聚合

Count PV/hour of the entire site? Regular Aggregation

每条数据访问状态
Access state for every record

```
SELECT
  DATE_FORMAT(ts, 'yyyy-MM-dd HH') hour,
  COUNT(*) AS pv
FROM user_log
GROUP BY DATE_FORMAT(ts, 'yyyy-MM-dd HH')
```



Mini-Batch 优化

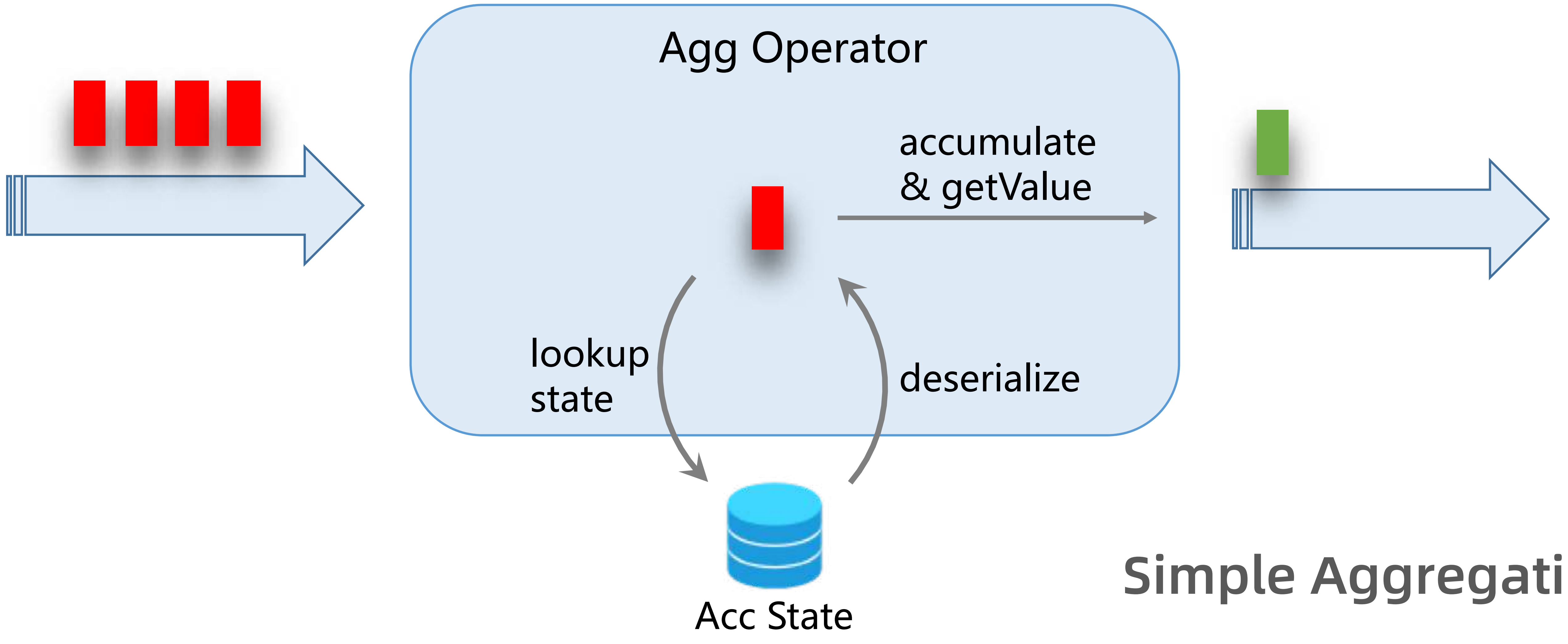
Mini-Batch Optimization

Mini-Batch 的优势：
Mini-Batch Pros:

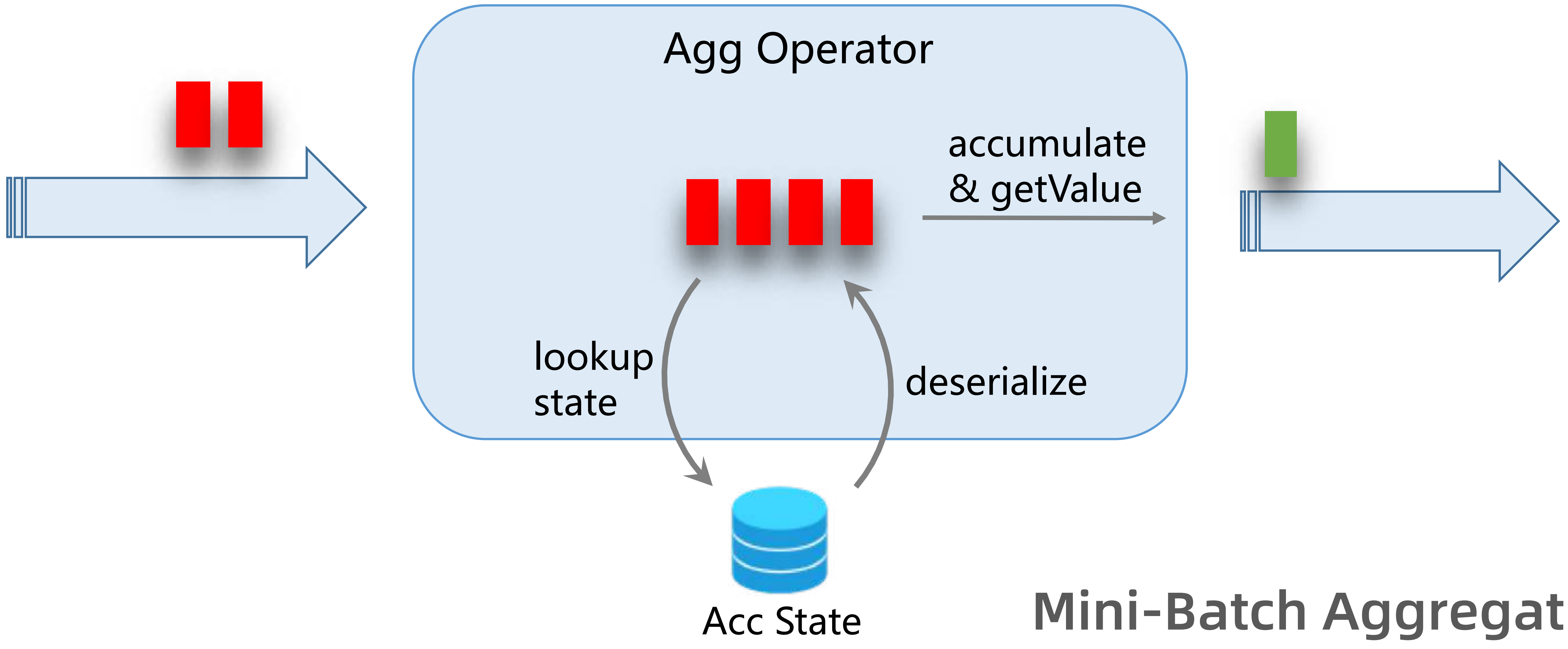
1. 提升吞吐
   Improve throughput
2. 减少 state 访问
   Reduce state access
3. 减少序列化/反序列化的开销
   Reduce de/serialization
4. 减少输出，降低对下游压力
   Reduce downstream pressure

```
# 开启 mini-batch
table.exec.mini-batch.enabled=true
# mini-batch的时间间隔，即作业需要额外忍受的延迟
table.exec.mini-batch.allow-latency=5s
# 一个节点中允许最多缓存的数据
table.exec.mini-batch.size=5000
```

Available from Flink 1.9



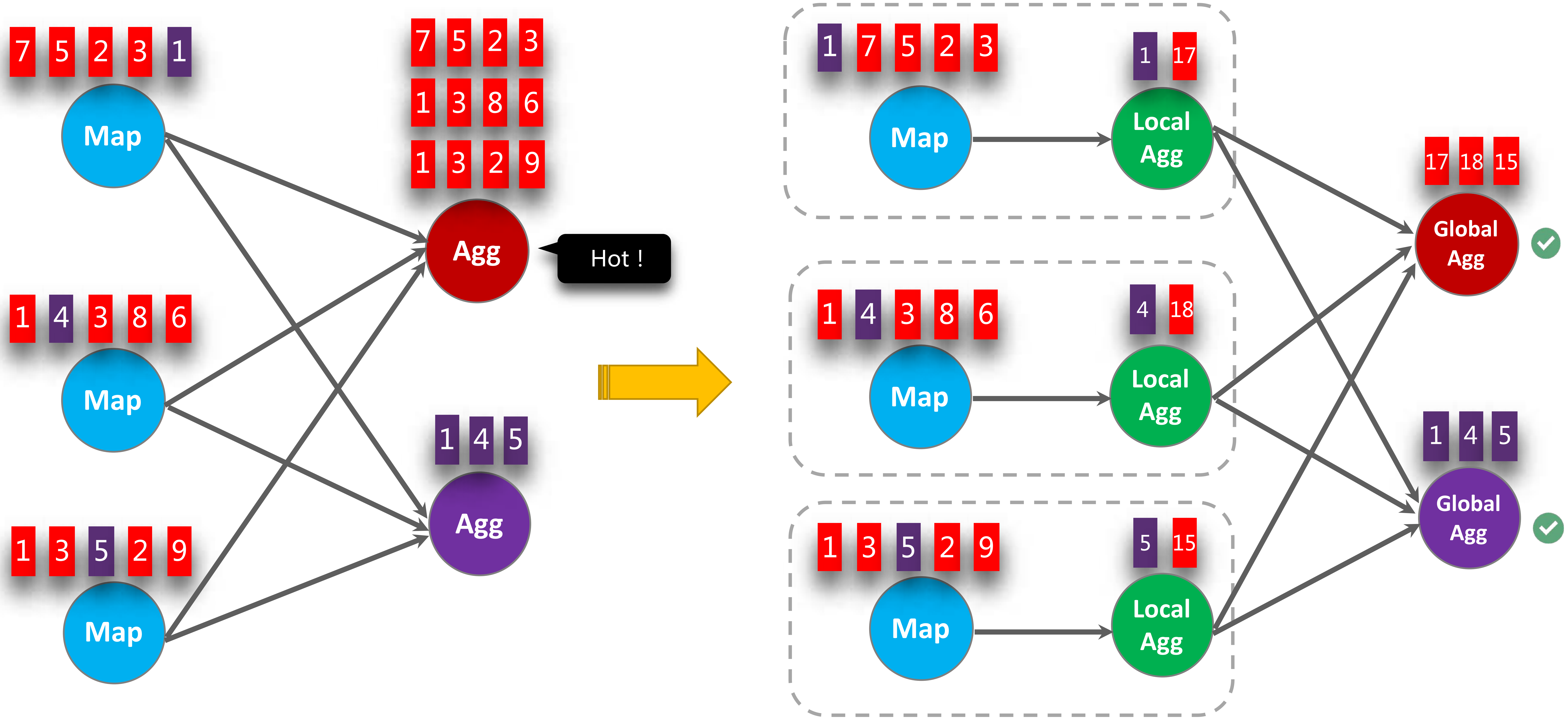
Simple Aggregation



Mini-Batch Aggregation

Local-Global 优化

Local-Global Optimization



Local-Global 优化

Local-Global 优势：
Local-Global Pros:

Local-Global Optimization

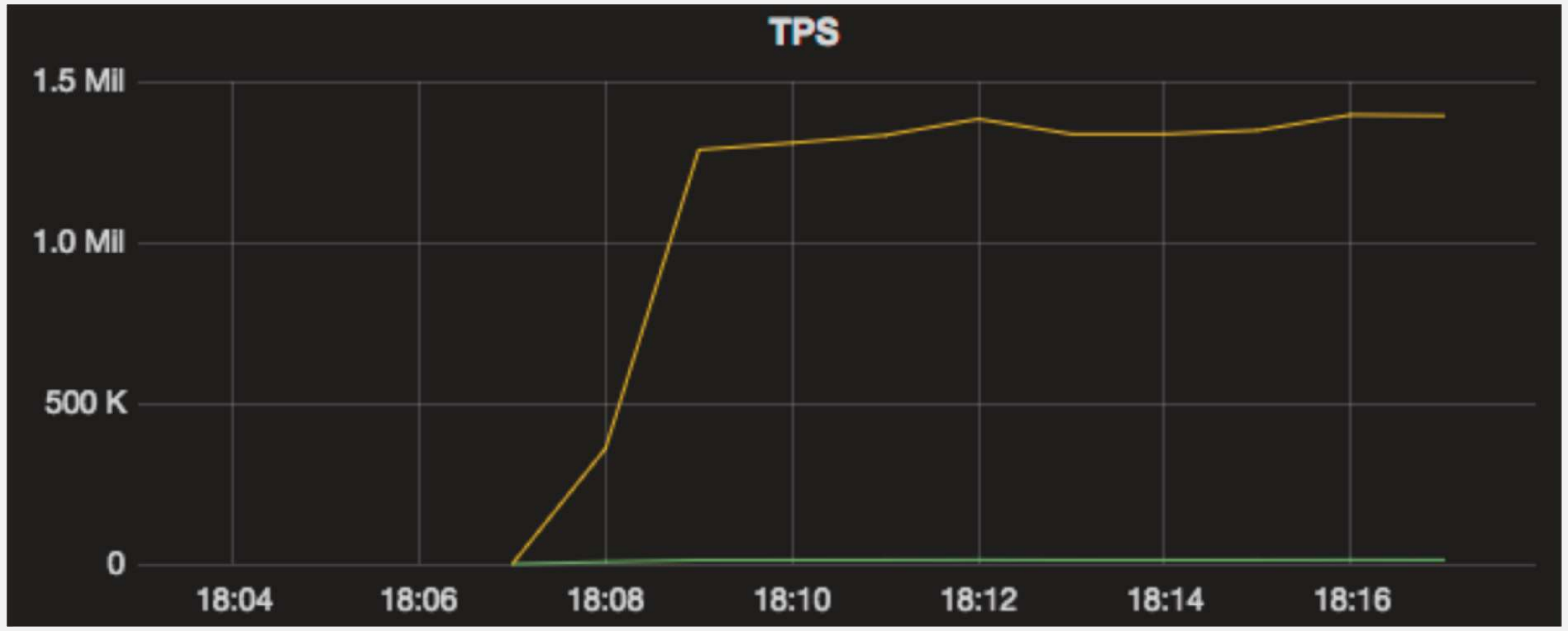
1. 对于简单聚合，性能提升明显
   Significant improvement for simple aggregates
2. 缓解数据倾斜
   Ease data skew
3. 减少网络传输
   Reduce network shuffle

注：所有的聚合函数都需要实现 merge() 方法。
Note: all aggregate function should implement merge() method.

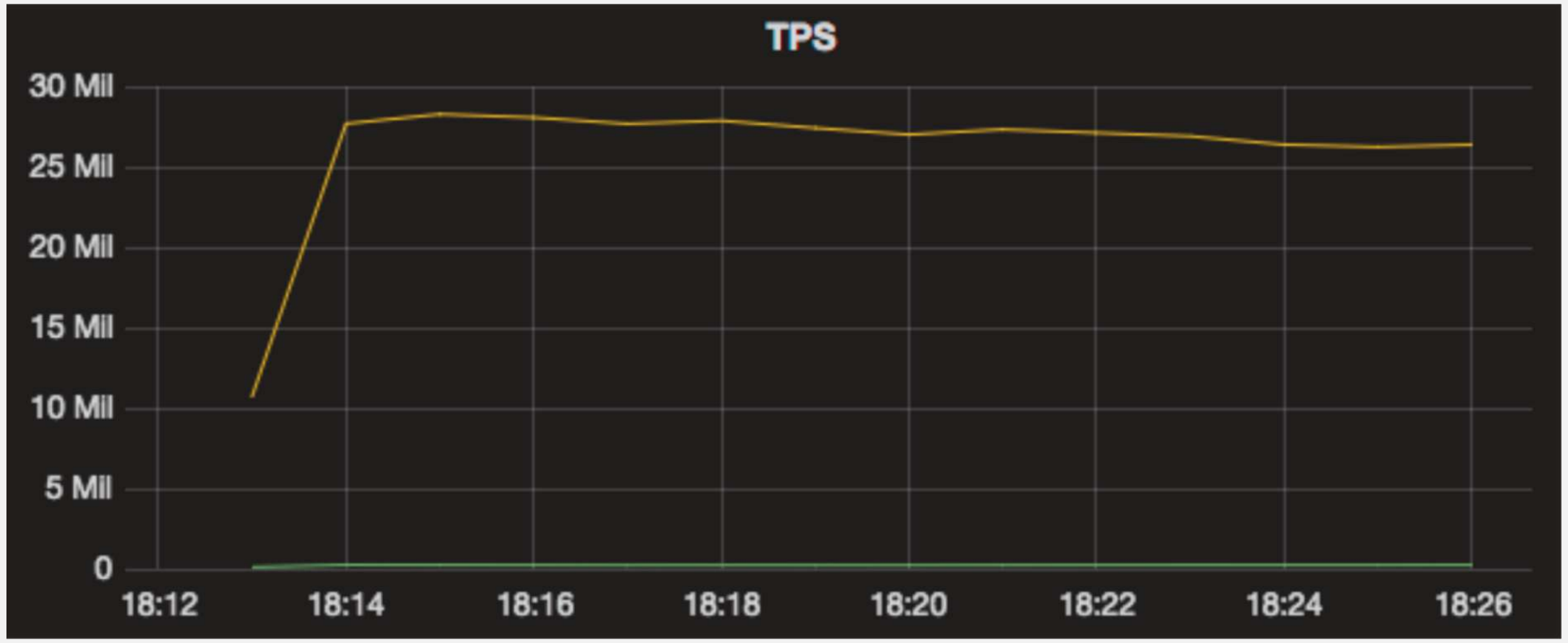
```
# 开启两阶段, 即 local-global 优化
table.optimizer.agg-phase-strategy=TWO_PHASE

# 还需开启 mini-batch
table.exec.mini-batch.enabled=true
table.exec.mini-batch.allow-latency=5s
table.exec.mini-batch.size=5000
```

Available from Flink 1.9

Before



After

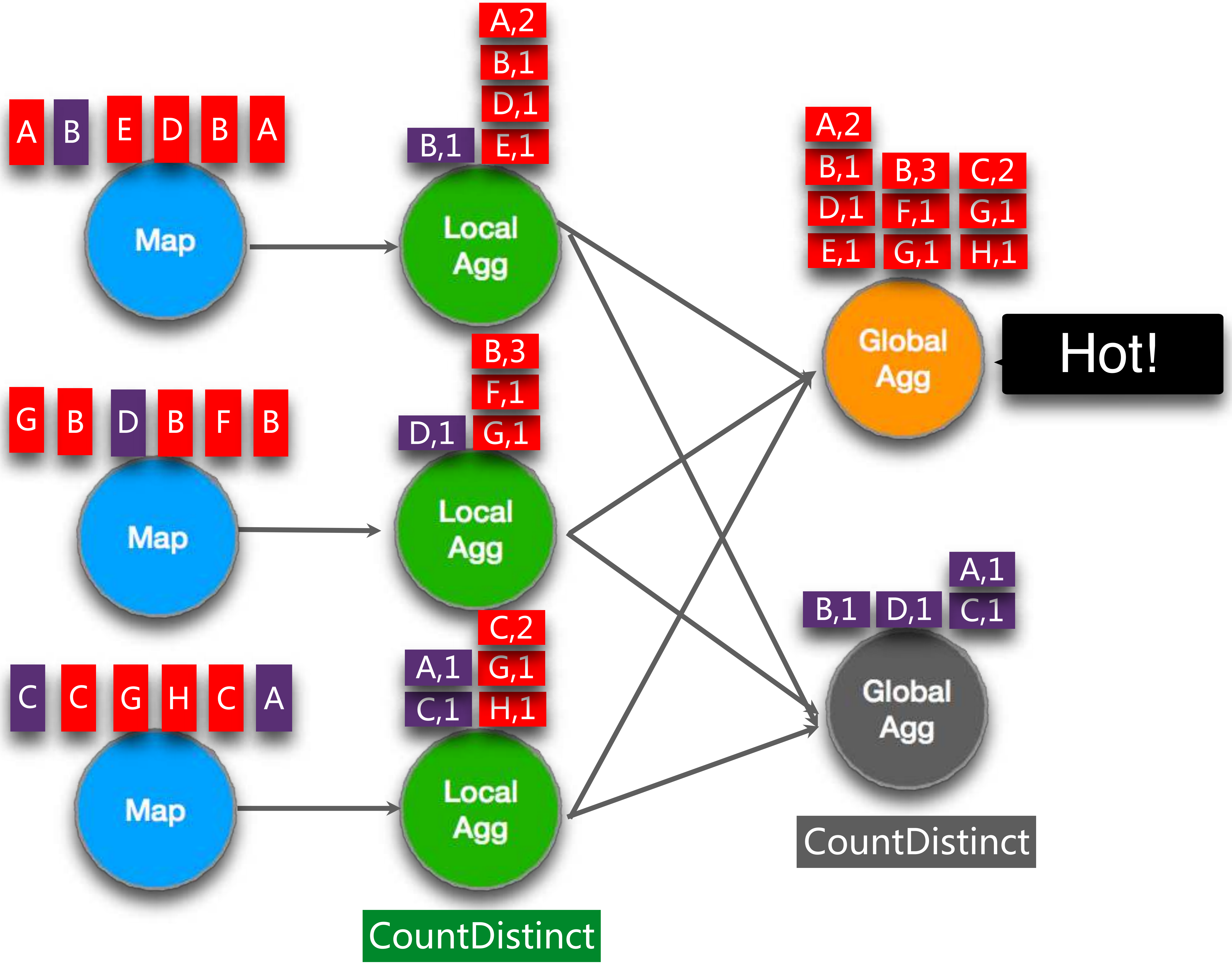


统计每小时全网的独立用户数?

Count UV/hour of the entire site?

DISTINCT 数据倾斜问题

DISTINCT Data Skew Problem



```sql
SELECT
  hour,
  COUNT(DISTINCT user) AS uv
FROM T
GROUP BY hour
```

对于中间结果会存储明细的agg，

（例如 count distinct）

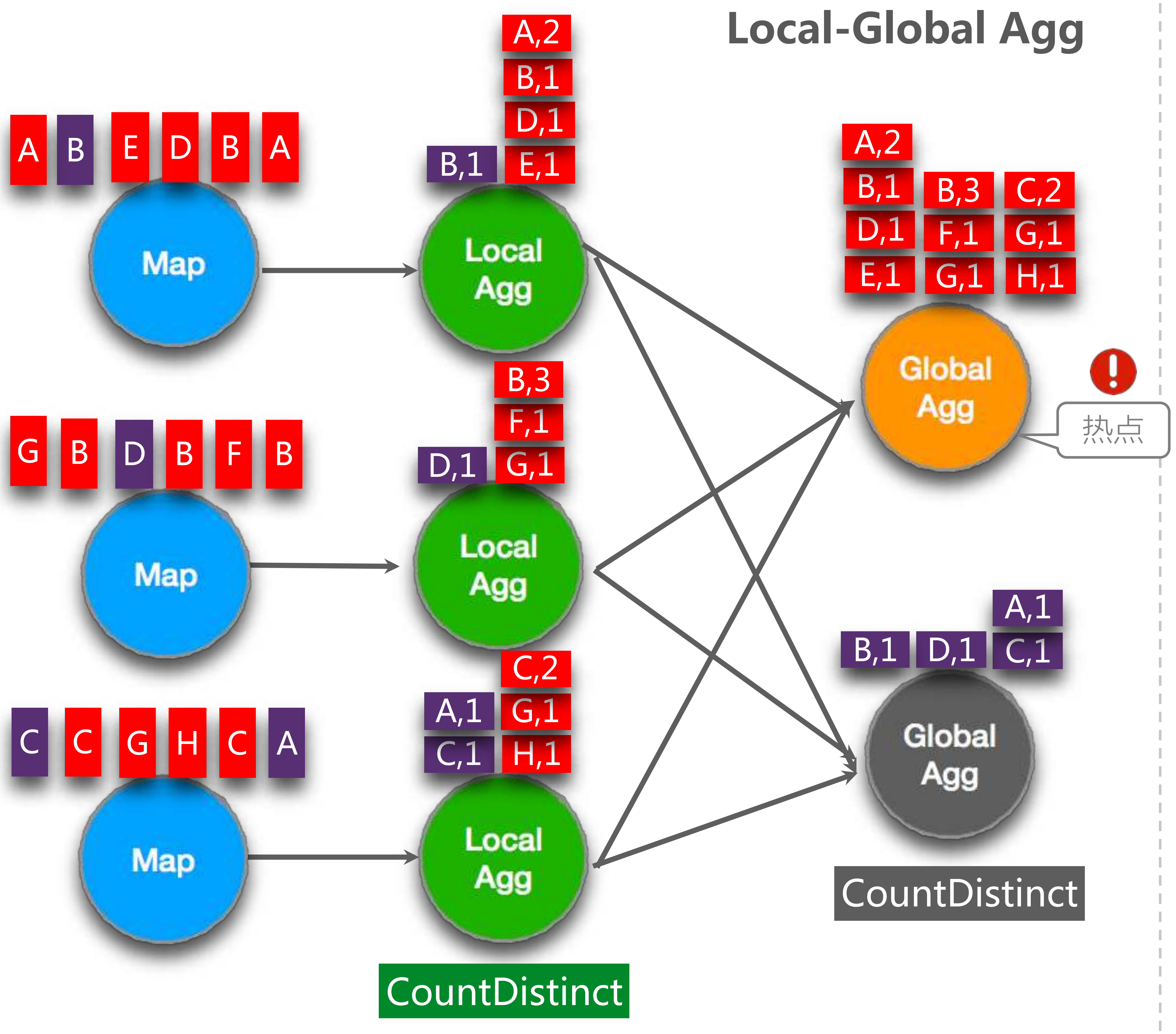
LocalGlobal无法有效解决热点问题。

For aggregates need detail data, (e.g. count distinct), Local-Global cannot resolve data skew problem.

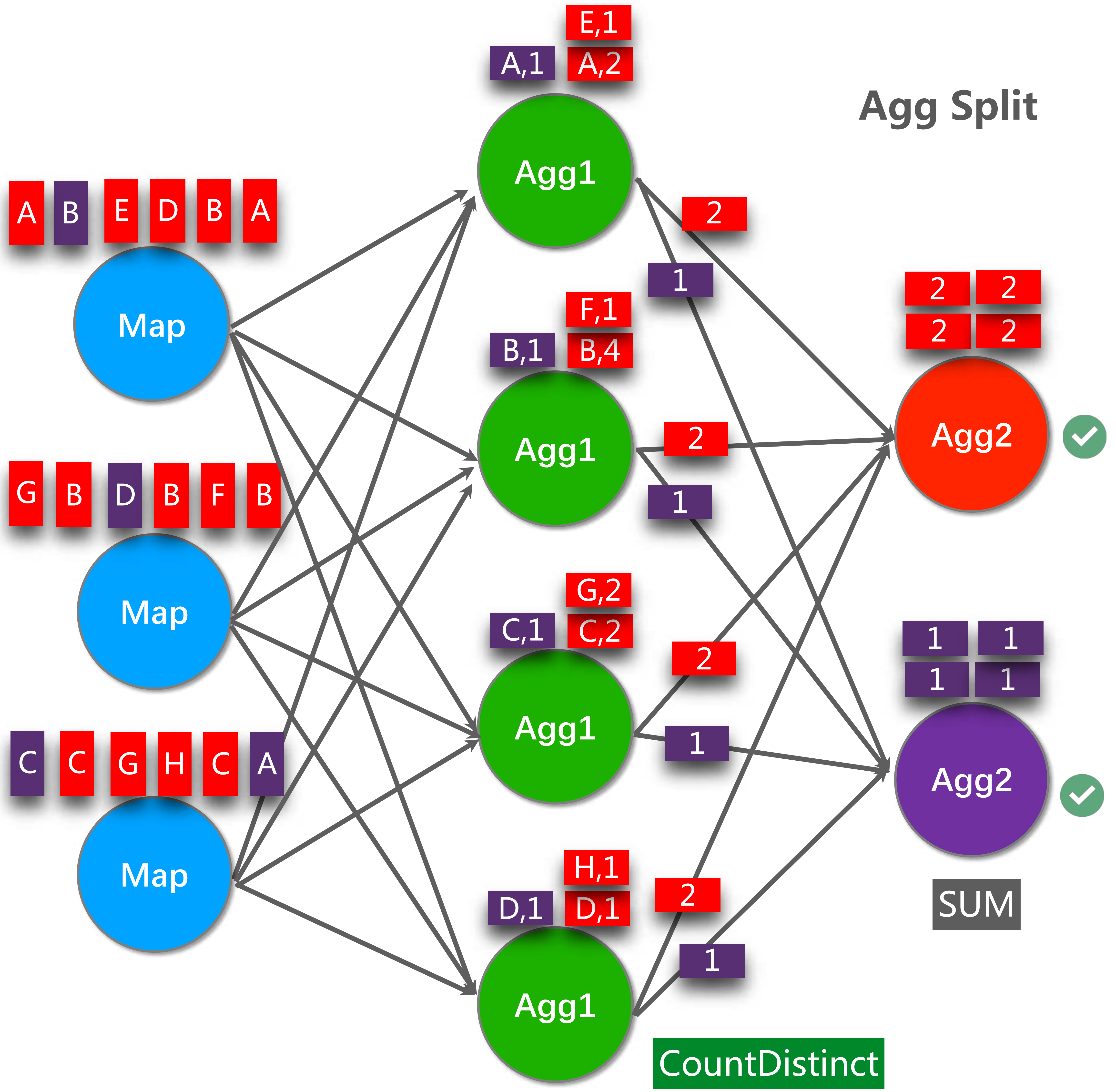
DISTINCT 数据倾斜问题

DISTINCT Data Skew Problem

Local-Global Agg



Agg Split



DISTINCT 数据倾斜问题

DISTINCT Data Skew Problem

1. 解决 COUNT DISTINCT 数据倾斜问题

   Resolve data skew problem of COUNT DISTINCT

2. 亦适用于多 DISTINCT 场景

   Can be used for multiple DISTINCT

注：不能有自定义聚合函数（UDAF）。
Note: there should be no UDAF.

```
# 开启 distinct agg 切分
table.optimizer.distinct-agg.split.enabled=true
```

Available from Flink 1.9

```
SELECT
  hour,
  COUNT(DISTINCT user) AS uv
FROM T
GROUP BY hour
```

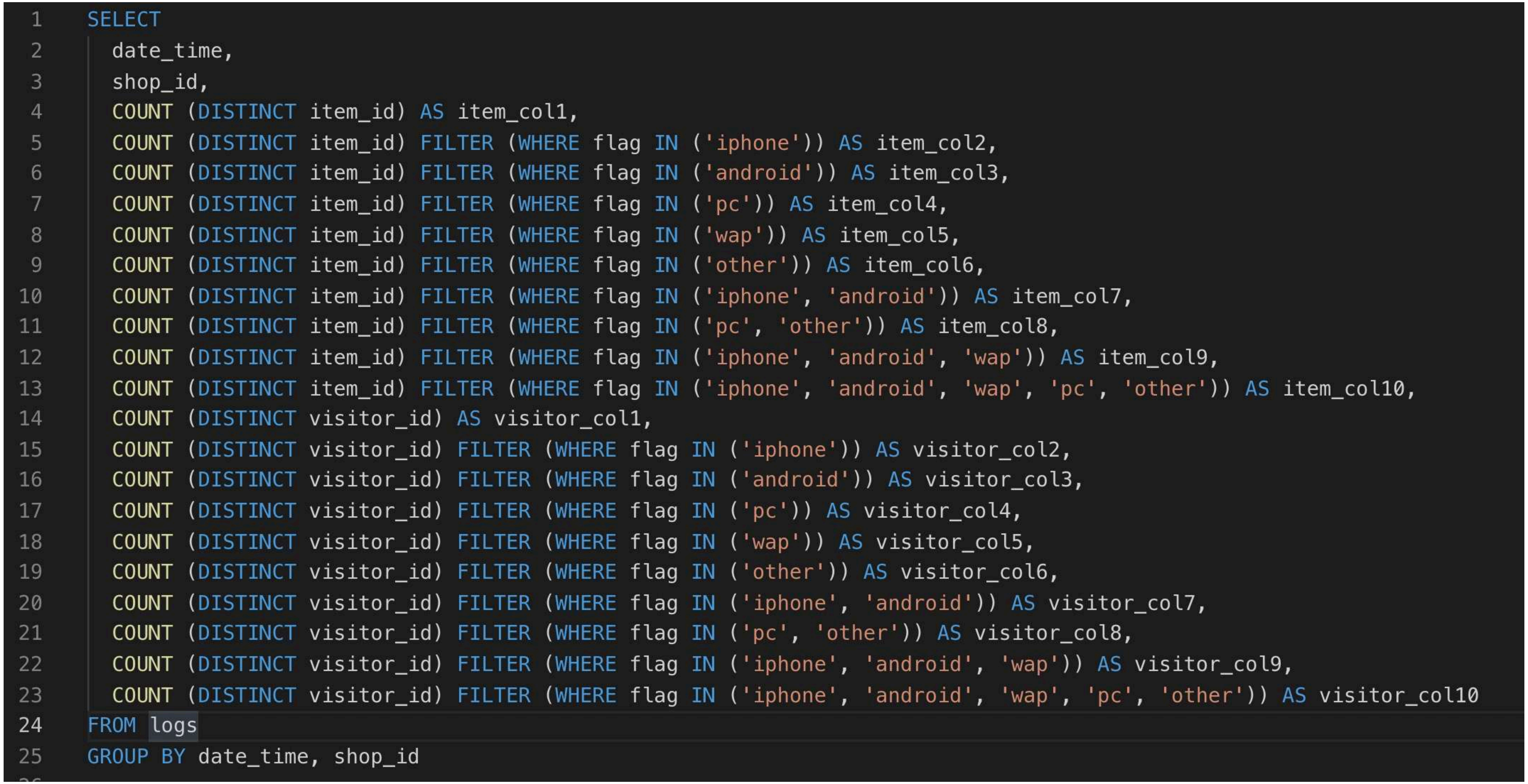
Before Split

```
SELECT hour, SUM(cnt) AS uv
FROM (
  SELECT hour, COUNT(DISTINCT user) AS cnt
  FROM T
  GROUP BY hour, MOD(HASH_CODE(user), 1024)
) GROUP BY hour
```

After Split

大量 COUNT DISTINCT 场景的优化

Lots of COUNT DISTINCT

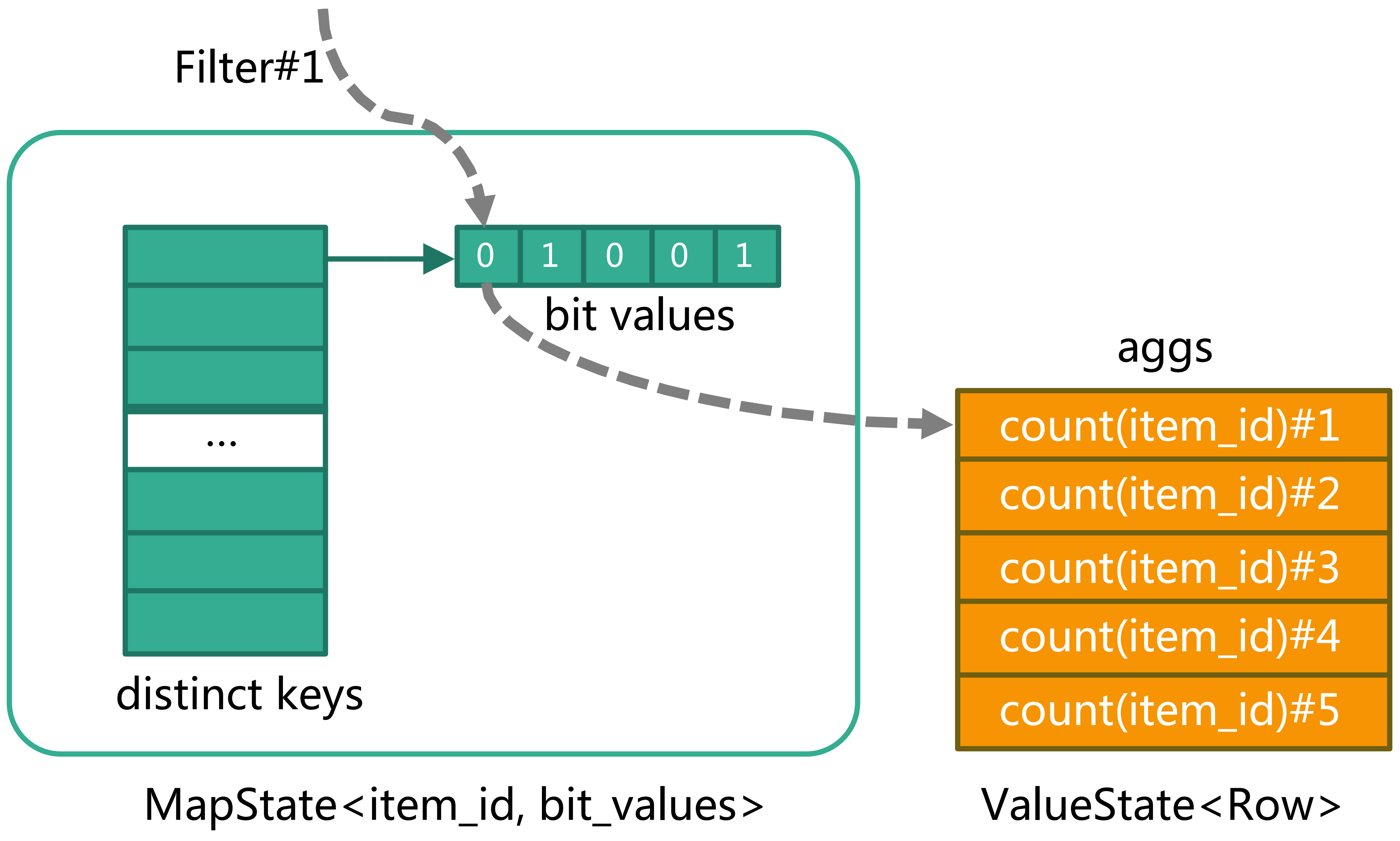


```sql
SELECT
  date_time,
  shop_id,
  COUNT (DISTINCT item_id) AS item_col1,
  COUNT (DISTINCT item_id) FILTER (WHERE flag IN ('iphone')) AS item_col2,
  COUNT (DISTINCT item_id) FILTER (WHERE flag IN ('android')) AS item_col3,
  COUNT (DISTINCT item_id) FILTER (WHERE flag IN ('pc')) AS item_col4,
  COUNT (DISTINCT item_id) FILTER (WHERE flag IN ('wap')) AS item_col5,
  COUNT (DISTINCT item_id) FILTER (WHERE flag IN ('other')) AS item_col6,
  COUNT (DISTINCT item_id) FILTER (WHERE flag IN ('iphone', 'android')) AS item_col7,
  COUNT (DISTINCT item_id) FILTER (WHERE flag IN ('pc', 'other')) AS item_col8,
  COUNT (DISTINCT item_id) FILTER (WHERE flag IN ('iphone', 'android', 'wap')) AS item_col9,
  COUNT (DISTINCT item_id) FILTER (WHERE flag IN ('iphone', 'android', 'wap', 'pc', 'other')) AS item_col10,
  COUNT (DISTINCT visitor_id) AS visitor_col1,
  COUNT (DISTINCT visitor_id) FILTER (WHERE flag IN ('iphone')) AS visitor_col2,
  COUNT (DISTINCT visitor_id) FILTER (WHERE flag IN ('android')) AS visitor_col3,
  COUNT (DISTINCT visitor_id) FILTER (WHERE flag IN ('pc')) AS visitor_col4,
  COUNT (DISTINCT visitor_id) FILTER (WHERE flag IN ('wap')) AS visitor_col5,
  COUNT (DISTINCT visitor_id) FILTER (WHERE flag IN ('other')) AS visitor_col6,
  COUNT (DISTINCT visitor_id) FILTER (WHERE flag IN ('iphone', 'android')) AS visitor_col7,
  COUNT (DISTINCT visitor_id) FILTER (WHERE flag IN ('pc', 'other')) AS visitor_col8,
  COUNT (DISTINCT visitor_id) FILTER (WHERE flag IN ('iphone', 'android', 'wap')) AS visitor_col9,
  COUNT (DISTINCT visitor_id) FILTER (WHERE flag IN ('iphone', 'android', 'wap', 'pc', 'other')) AS visitor_col10
FROM logs
GROUP BY date_time, shop_id
```

DISTINCT 状态复用

DISTINCT State Reuse

```
SELECT
  key,
  count(distinct item_id) filter (…),
  count(distinct item_id) filter (…),
  count(distinct item_id) filter (…),
  count(distinct item_id) filter (…),
  count(distinct item_id) filter (…)
FROM T
GROUP BY key
```



节省了约1倍的 state size, 4倍的额外读写

Save 1x state size, 4x extra write & read

实际测试性能提升1倍

1x improved actual test performance

Window Aggregation VS Regular Aggregation

		Window Aggregation	Regular Aggregation
功能 Functionality	Input stream	Only Append Stream	Append/Retract Stream
	Output mode	Output when window close	Per record by default
	Output stream	Append Stream	Update Stream
	Supported Sink	All (DB, MQ, File)	Updatable sink (DB)
性能 Performance	MiniBatch	NO	YES
	Local-Global	NO	YES
	DISTINCT Split	NO	YES
	DISTINCT State Reuse	YES	YES
	Latency	window size + watermark delay	LOW
稳定性 Stability	Throughput	No tuning options	High after optimization
	Data skew	No good solutions	Have solutions
	State cleanup	Cleanup when window expired	None by default ❗

精确性 ←→ 状态大小
Accuracy ←→ State Size

State TTL Config

```
TableConfig
  .withIdleStateRetentionTime(Time.day(1), Time.day(2))
```

计算店铺的实时引流能力

（店铺UV/类目UV）

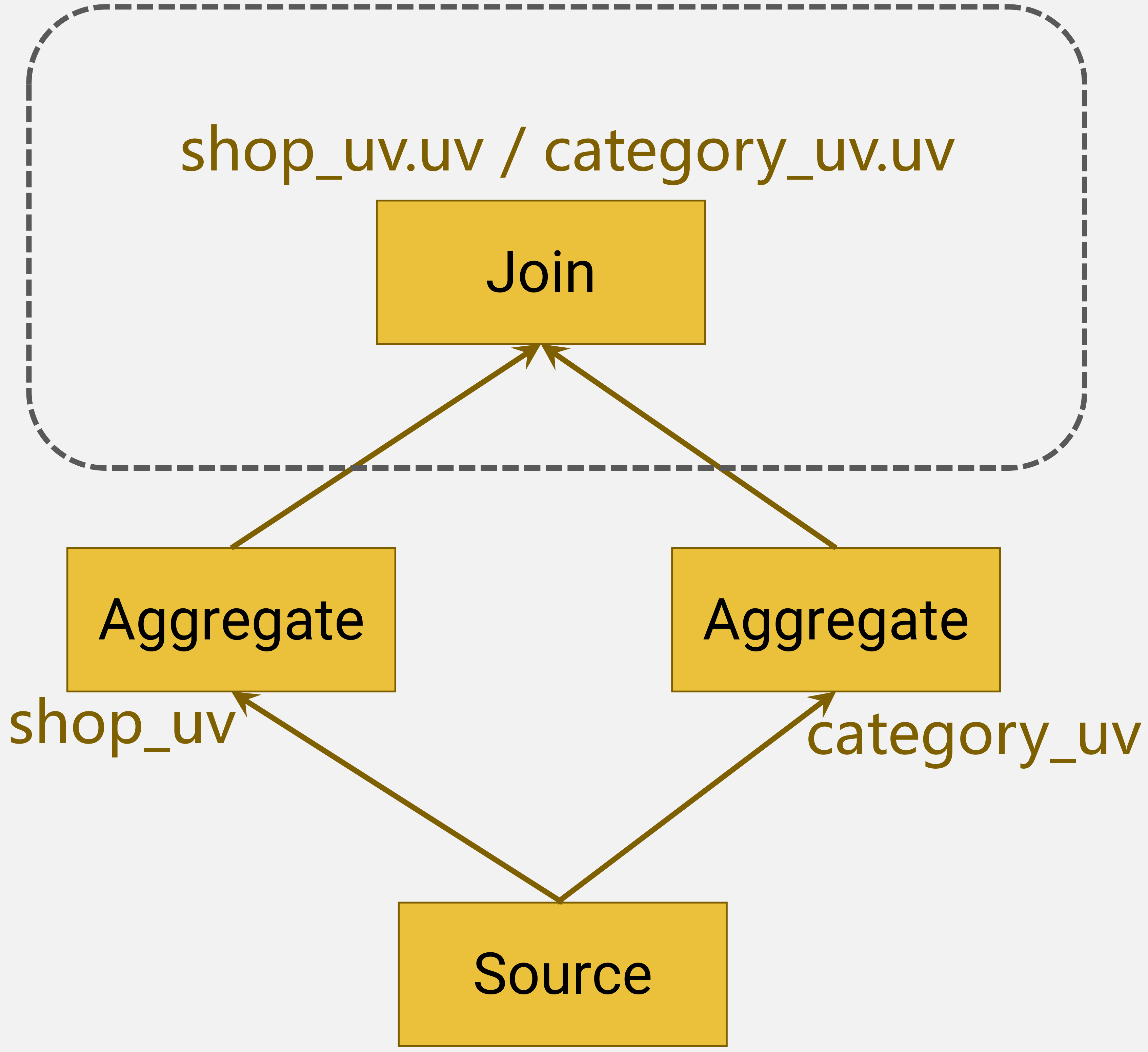
Calculate the Score of Shops

(Shop UV / Category UV)

店铺引流能力（店铺 UV / 类目 UV）

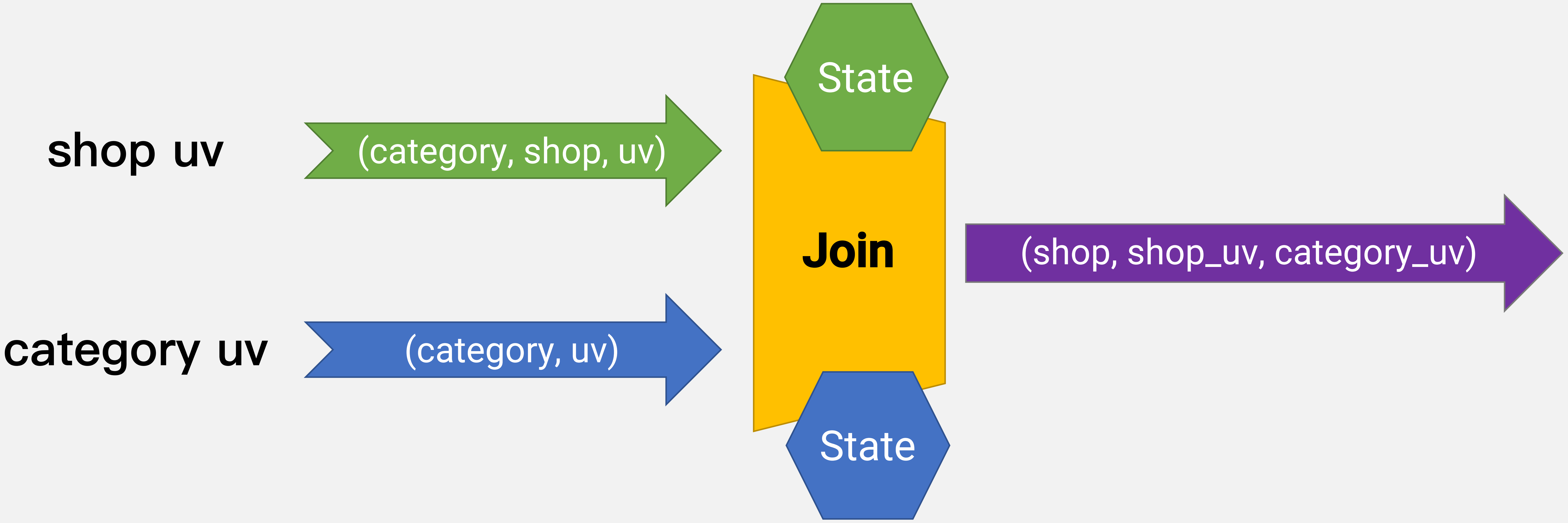
Score of Shops (Shop UV / Category UV)

```
SELECT
  S.hour, S.category, S.shop, S.uv/C.uv AS score
FROM
  ( SELECT hour, category, shop, COUNT(DISTINCT user) AS uv
    FROM T
    GROUP BY hour, category, shop
  ) AS S JOIN (
    SELECT hour, category, COUNT(DISTINCT user) AS uv
    FROM T
    GROUP BY hour, category
  ) AS C
  ON S.hour = C.hour AND S.category = C.category;
```



双流 Join State 优化

Optimization for State of Stream-Stream Join



使用 State 来存储所有已经到到达的数据，
因为另一条流的匹配的数据可能任何时候都会到来

Use state to store all arrived records

双流 Join State 优化

Optimization for State of Stream-Stream Join

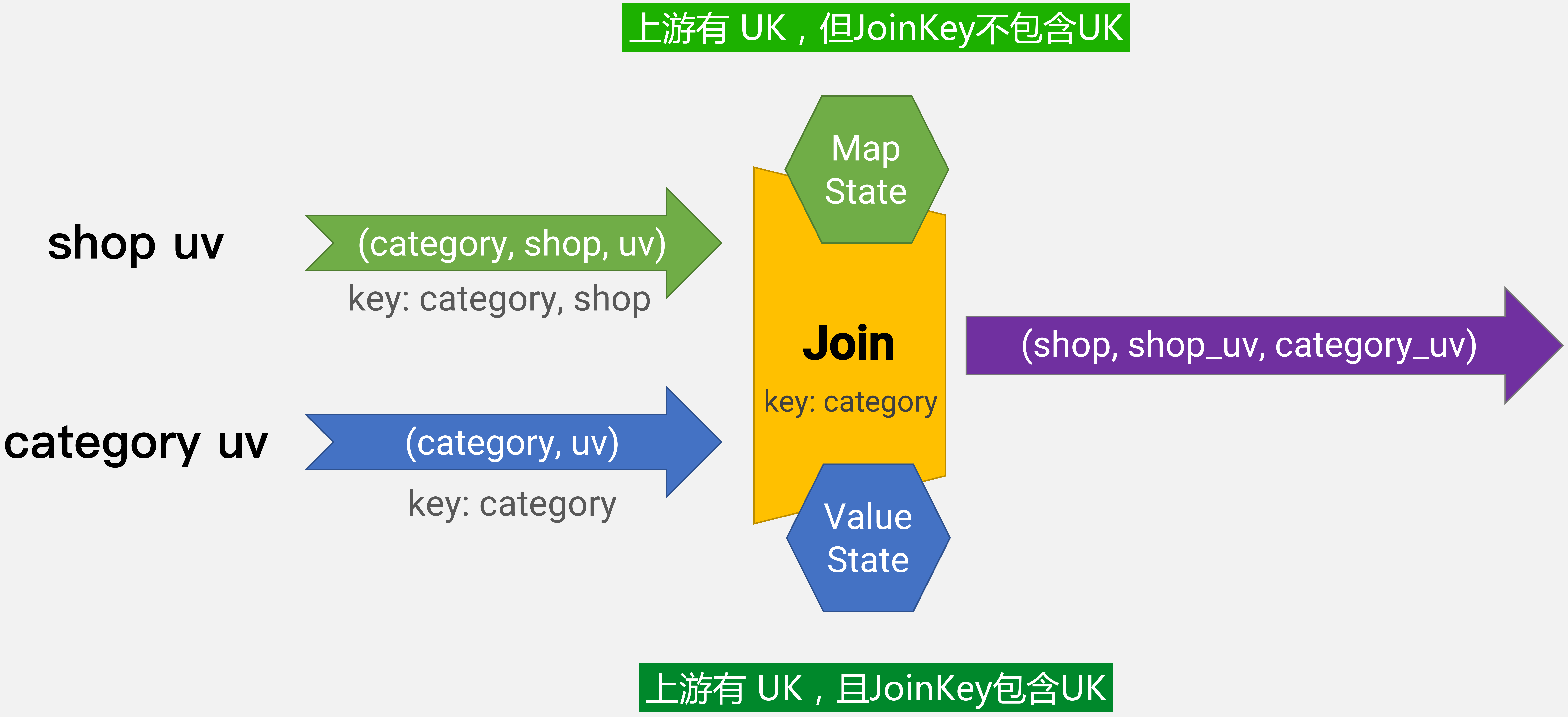
	State Structure	Update Row	Query by JK	Note
上游有UK, JK 包含UK	<JK, ValueState<Row>>	O(1)	O(1)	
上游有UK, JK 不包含UK	<JK, MapState<UK, ROW>>	O(2)	O(N)	N = size of MapState
上游没有UK	<JK, ListState<Row>>	O(N)	O(N)	N = size of MapState

性能分析：
Performance：

上游有 UK，且JoinKey包含UK > 上游有 UK，但JoinKey不包含UK > 上游没有 UK > 无 JoinKey

双流 Join State 优化

Optimization for State of Stream-Stream Join



深入探索 Flink SQL 批处理

Deep Dive into Flink SQL Batch Processing
- v1.10 Blink planner

03

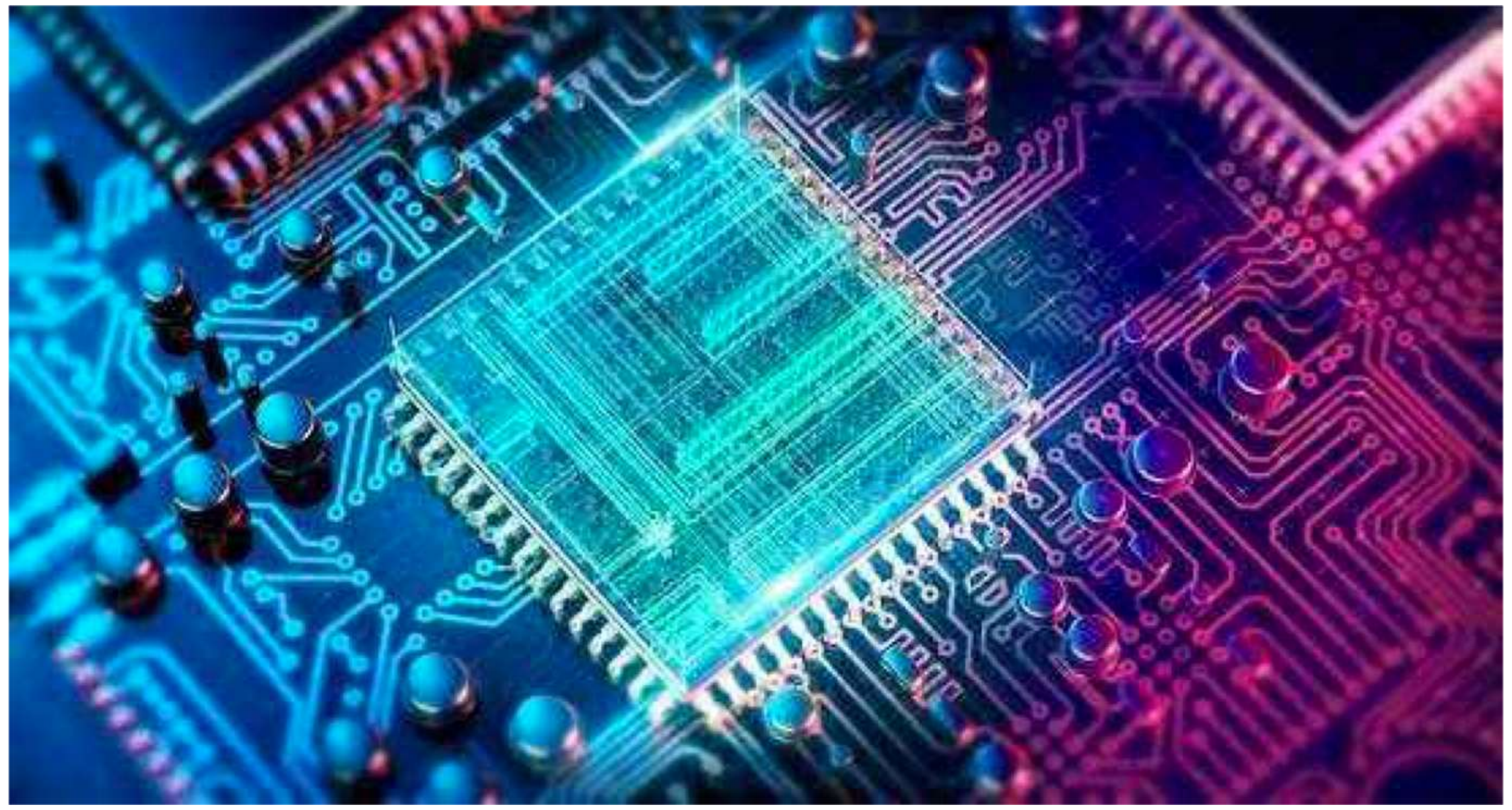
About me

- 李劲松（之信，Jingsong Li）
- Apache Flink Contributor & Apache Beam Committer
- 阿里巴巴 Blink SQL 团队

批计算

Batch processing

- 容错 – Batch shuffle mode
- 资源模型 – 提高资源利用率
- Hive 集成 – Meta/Format/UDF 兼容
- SQL 优化 – 丰富完善的Rules
- SQL 支持 – 完善的SQL支持，包括TPC-DS等测试集
- 算子 – 高性能优化



1.10 Production ready!

Batch
批处理

01 案例

Example

02 批处理模型和架构

Batch mode and Architecture

03 批Connector支持

Batch Connector support

04 批处理算子

Batch operators

案例

Example

```
TableEnvironment tEnv =
    TableEnvironment.create(EnvironmentSettings.newInstance().useBlinkPlanner().inBatchMode().build());
tEnv.registerCatalog(”hiveCatalog“, new HiveCatalog(…));
tEnv.useCatalog(”hiveCatalog");
tEnv.sqlUpdate("INSERT OVERWRITE OUTPUT_TABLE ....");
tEnv.execute(“MyJob”);
```

Batch作业建议使用Catalog

```
SELECT
  S.hour, S.category, S.shop, S.uv/C.uv AS score
FROM
  ( SELECT hour, category, shop, COUNT(DISTINCT user) AS uv
    FROM T
    GROUP BY hour, category, shop
  ) AS S JOIN (
    SELECT hour, category, COUNT(DISTINCT user) AS uv
    FROM T
    GROUP BY hour, category
  ) AS C
  ON S.hour = C.hour AND S.category = C.category;
```

完全与流SQL一致

案例
Example

```
Project
+- Join
   :- Aggregate
   :  +- Aggregate(Distinct)
   :     +- TableScan
   +- Aggregate
      +- Aggregate(Distinct)
         +- TableScan
```

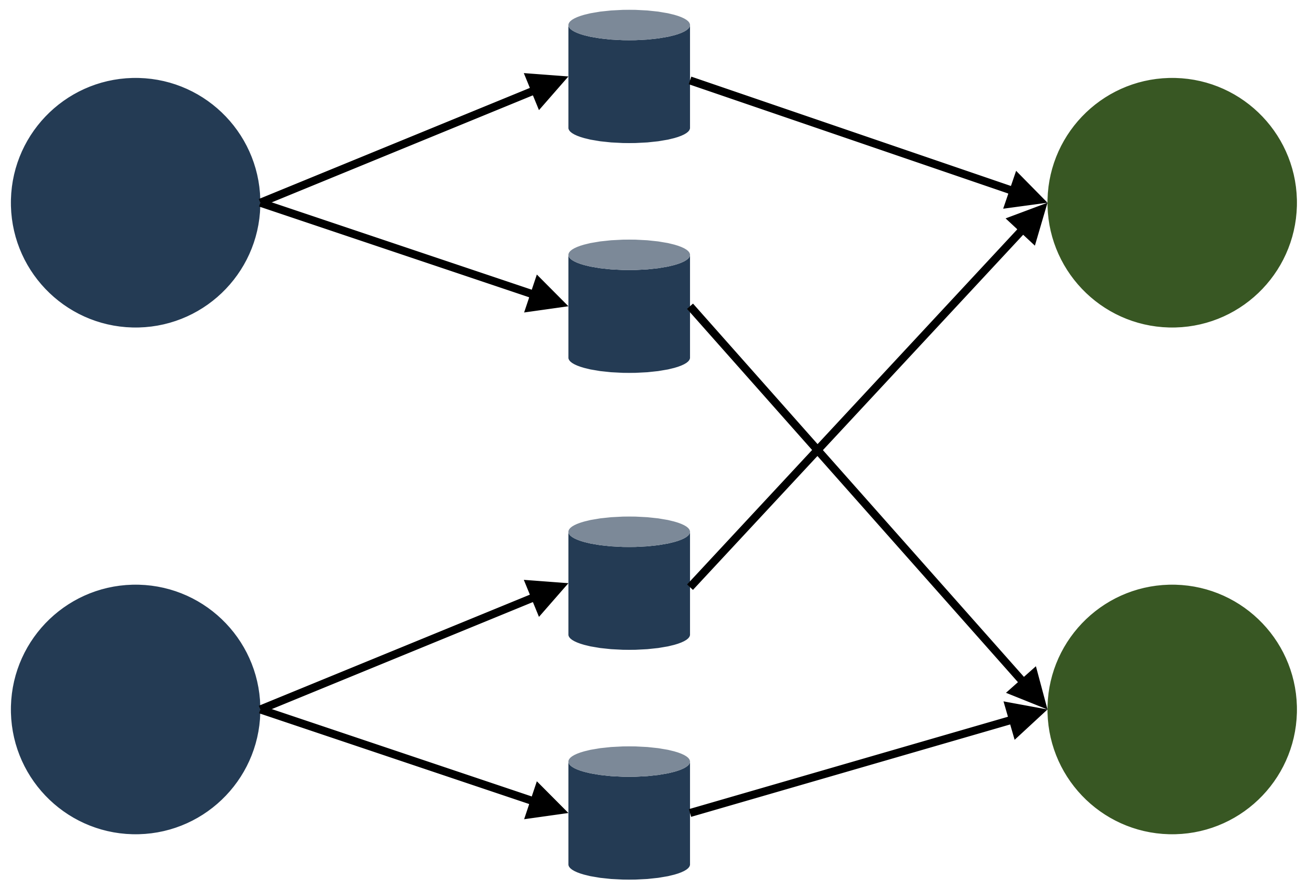
Batch作业如何分布式运行？

How to run Batch job distributed?

批处理Shuffle

Batch Shuffle

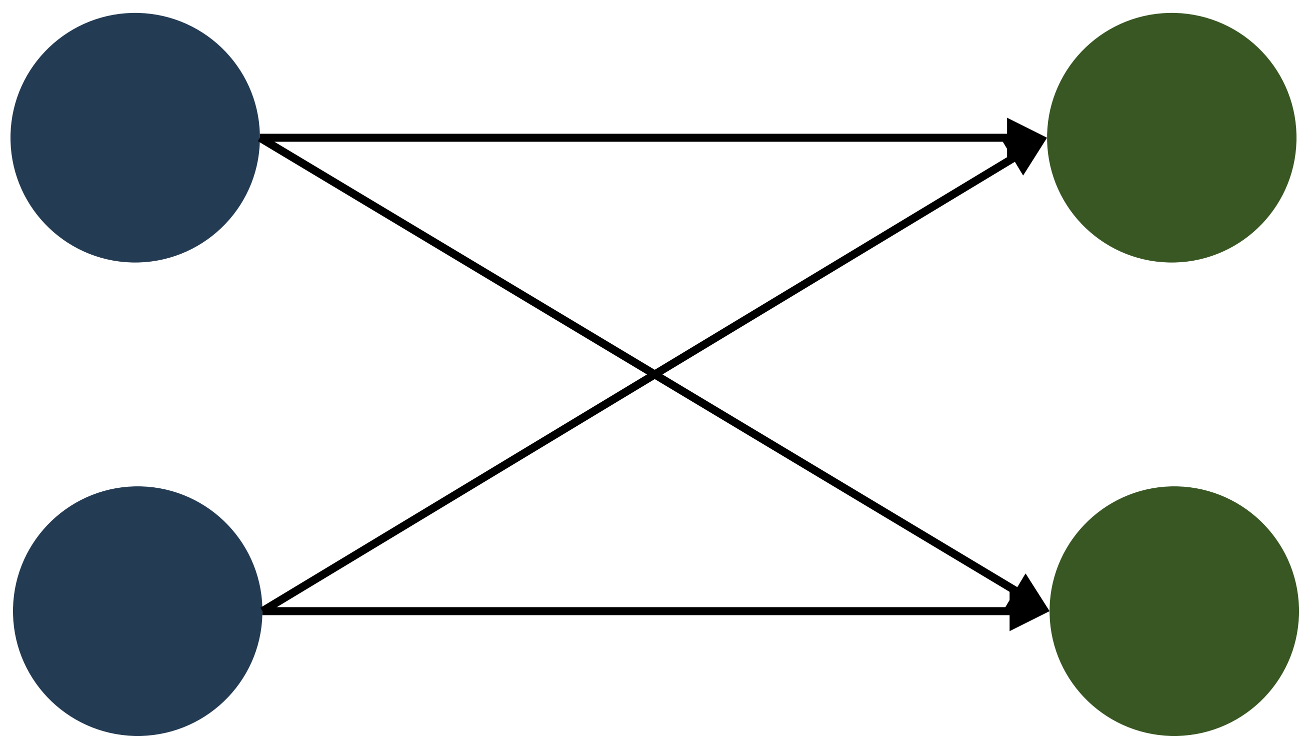
Batch mode(Default)



- 容错好，可以单点恢复
- 调度好，不管多少资源都可以运行
- 性能差，中间数据需要落盘，强烈建议开启压缩

适合传统Batch作业

Pipeline mode

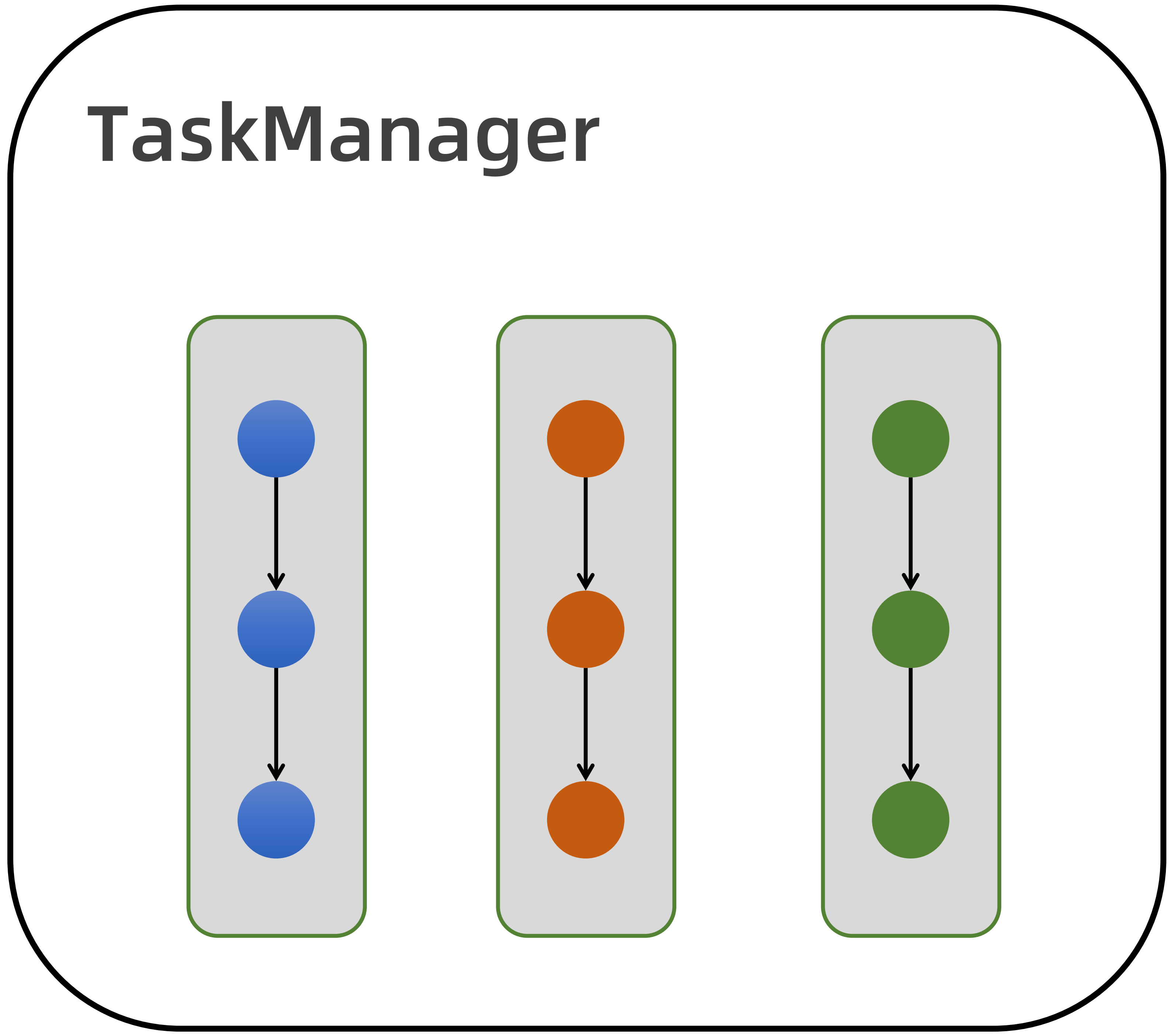


- 容错差，只能全局重来
- 调度差，你得保证有足够的资源
- 性能好，Pipeline执行，完全复用Stream，复用流控反压等功能。

适合OLAP场景

批处理资源模型

Batch resource model



Session mode VS Per job

并发：

- Source个性化并发，下游Task统一并发配置
- Batch关闭SlotShare，一个Task对应一个Slot

内存：

- Batch的内存资源是细粒度分配
- Slot的Manage内存需比单Task内存要大，建议500MB+ (FLIP-56旨在去除Slot的内存资源，由TM统一管理)
- 内存抢占 on going

Source/Sink表从哪里来？

Where do source and sink tables come from?

批Connector支持

Batch Connector Support

- Hive catalog & Hive connector

- 兼容 Stream Connectors：JDBC/HBase

- 高性能Format：ORC
  - Only for Hive 2+版本
  - ORC with Collection Type, ORC for Hive 1.X 和 Parquet 会 Fallback 到 Hadoop reader

- Table统计信息
  - 支持 Partition pruning
  - 完整统计信息有利于优化器生成更好的 Plan

Aggregate和Join都是怎么执行的？

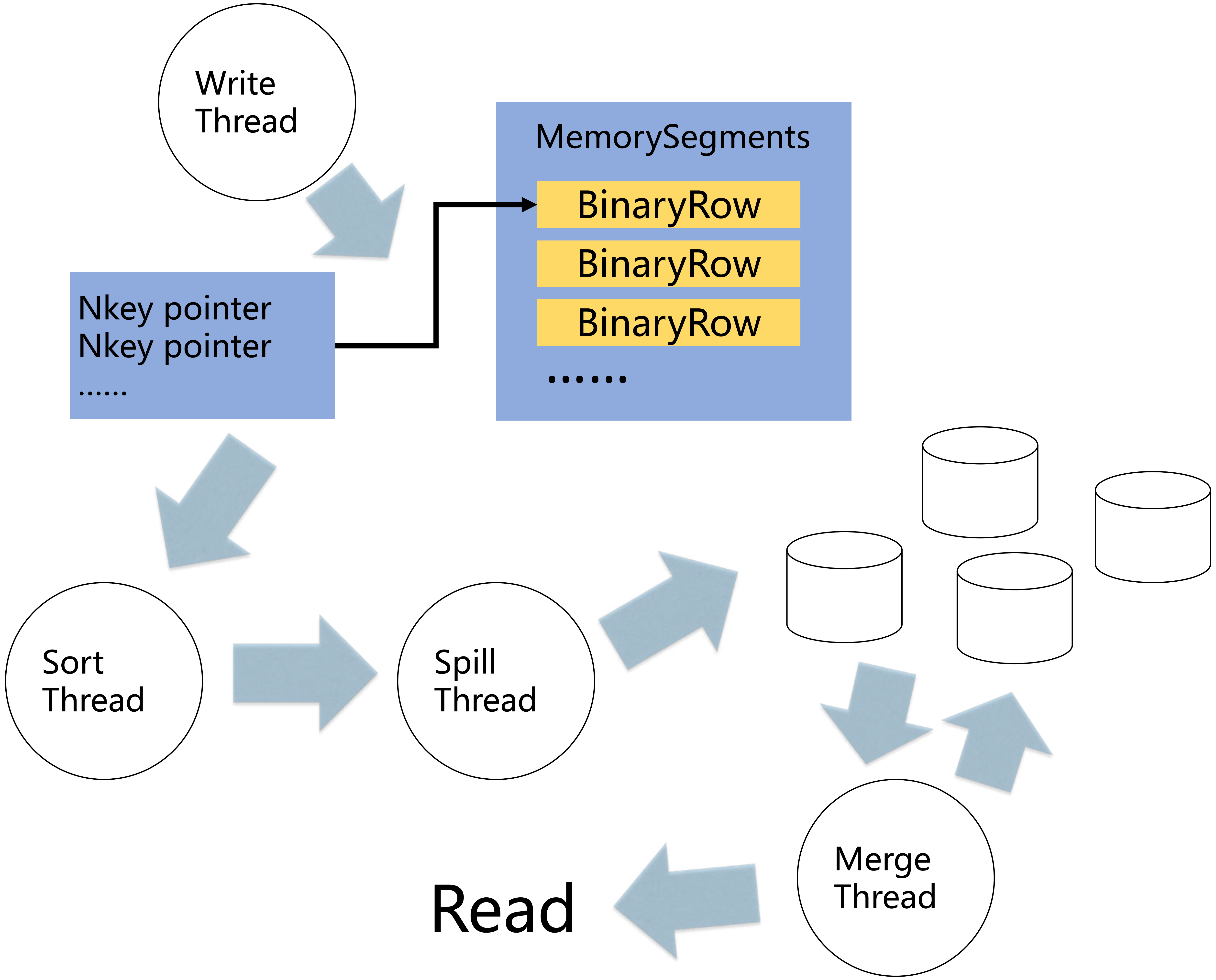
How to execute Aggregate and Join?

批排序算子

Batch Sort operator

Normalized Key + 多线程

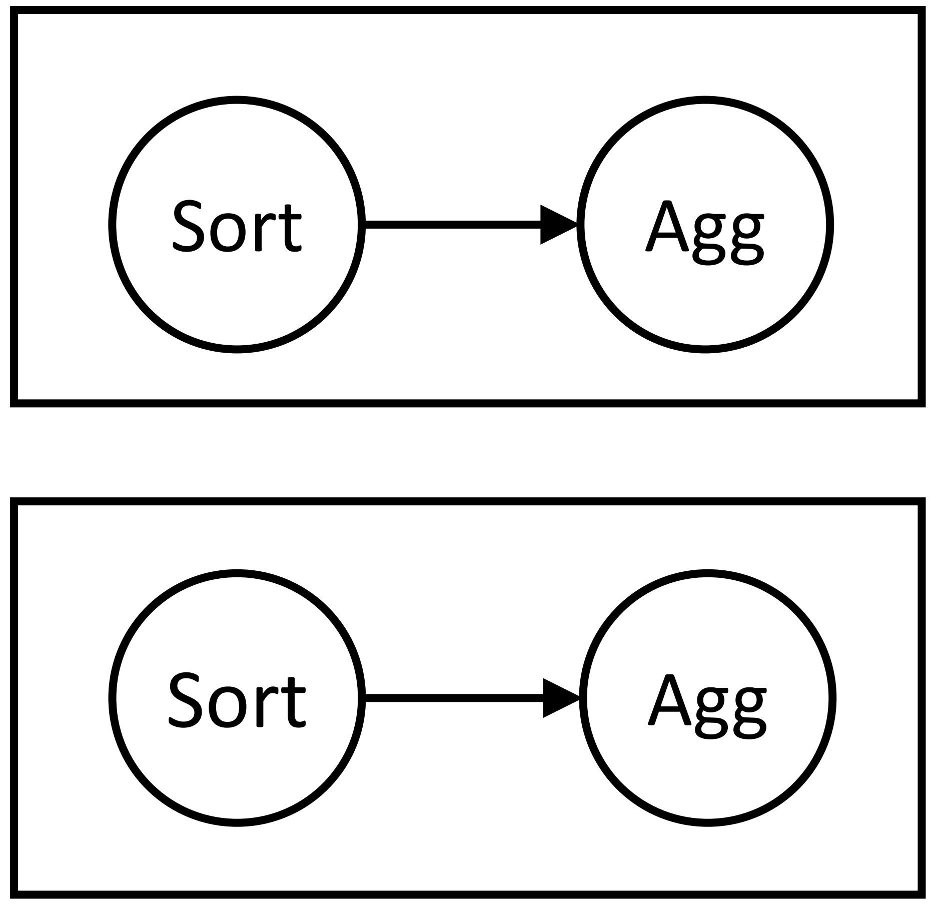
- 强烈建议Key采用Primitive types
- 建议内存大于100MB，利于多线程



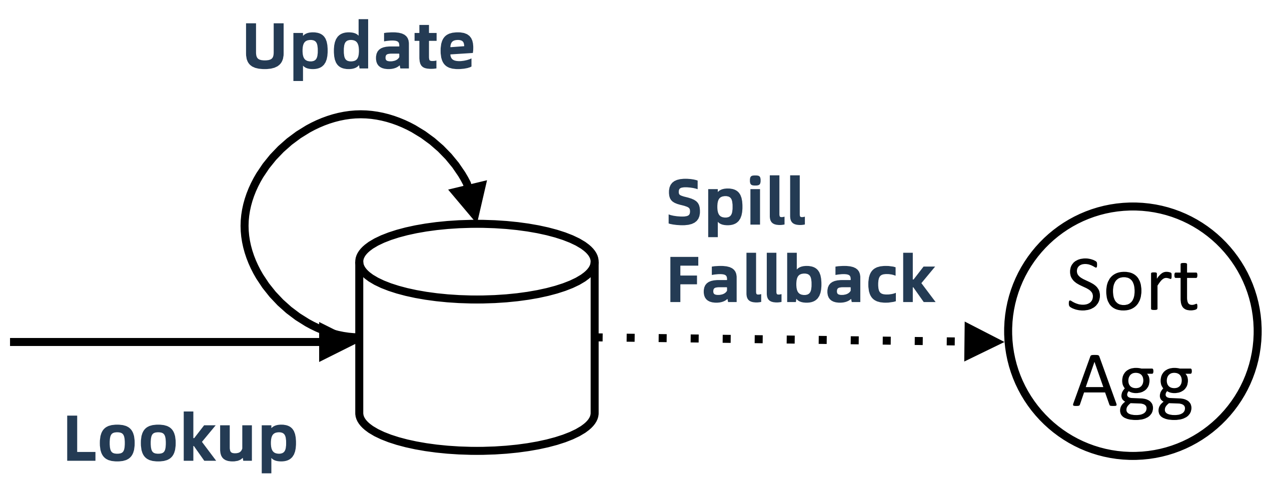
批聚合算子

Batch Aggregate operator

Sort Aggregate



Hash Aggregate



两阶段Aggregate

- AggFuncs全部有Merge功能才能两阶段
- 默认根据Cost选取是否两阶段
- LocalHashAgg不会落盘
- 未来LocalAgg将会动态适应

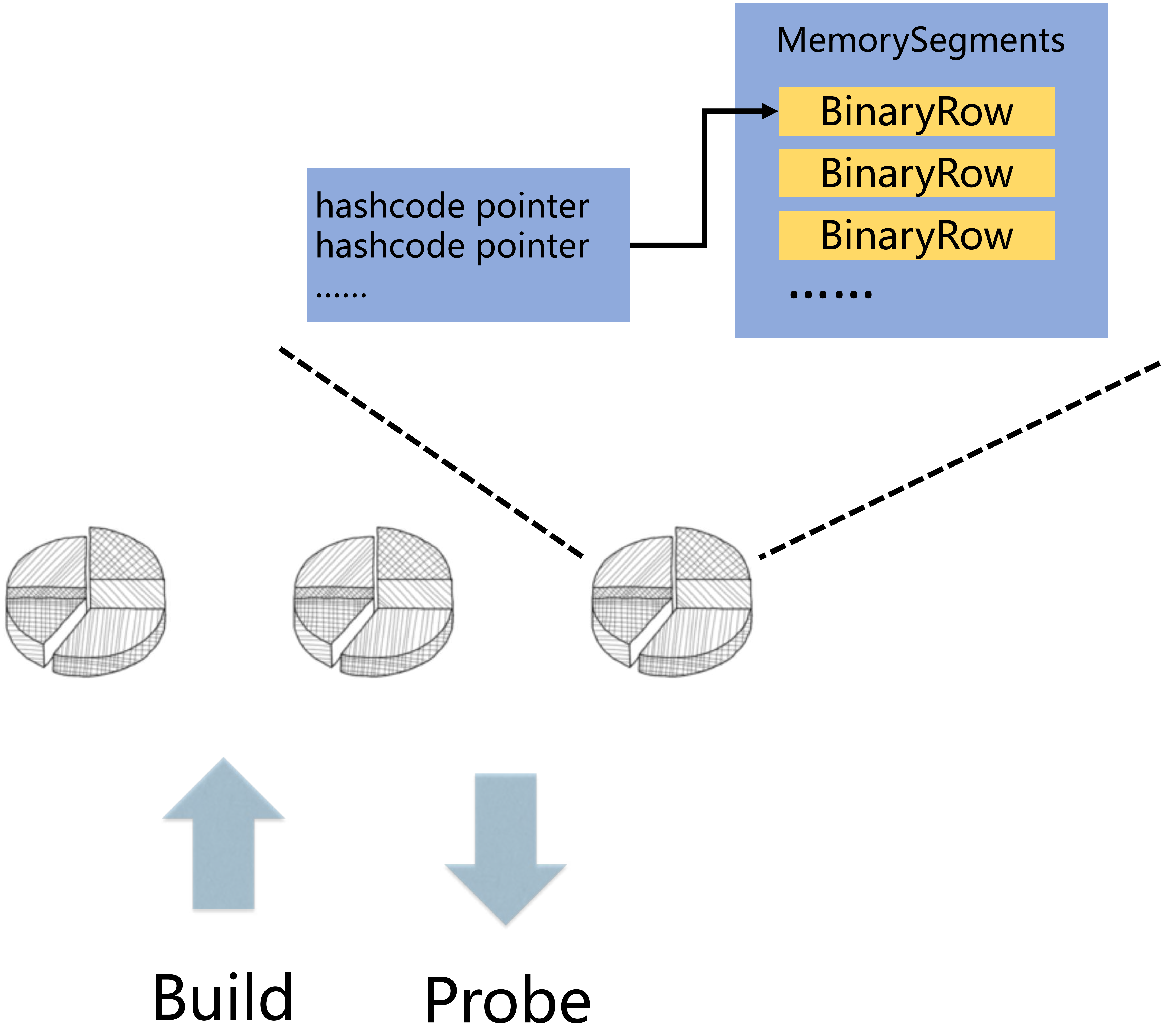
Hash Aggregate

- 强烈建议能选取Hash Agg
- AggFuncs的agg buffer需要都是固定长度
- 不能有UserDefine Agg functions

批Join算子

Batch Join operator

- HybridHashJoin VS SortMergeJoin
- 选错build: exceeded maximum number of recursions
  - 配置禁用HashJoin
  - 后续会引入FallbackSortMergeJoin、Role reversal.
- 强烈建议Key是8个字节能装下的，比如1个long，2个int等
- BroadcastJoin阈值，默认1MB
  - 建议加大至10MB
  - 后续版本会有可观的提升



欢迎大家试用1.10的Batch功能

Welcome to use the batch of flink 1.10

- 全面的 Hive 集成
- 完善的流批统一SQL支持
- 相比 Hive 7X的性能提升

未来计划
Future plan

04

未来计划

Future Plan

- 支持ChangeLog Connector，连接外部 binlog 系统，支持实时数据同步，实时维表关联等场景
- SQL作业的细粒度资源设置
- Hive完整支持（支持完善的版本，支持完善的Format）
- Batch TwoInputOperator Chain
- Runtime filter

THANKS