基于Streaming构建统一的数据处理引擎的挑战与实践

公司：阿里巴巴 Alibaba

职位：高级技术专家 / 高级开发工程师

演讲者：杨克特 / 伍翀

About me

- 杨克特（鲁尼，Kurt）
- Apache Flink Committer
- 2011年硕士毕业加入阿里，参与多个计算平台核心项目的设计和研发，包括搜索引擎、调度系统、监控分析等。目前负责Blink SQL的研发和优化

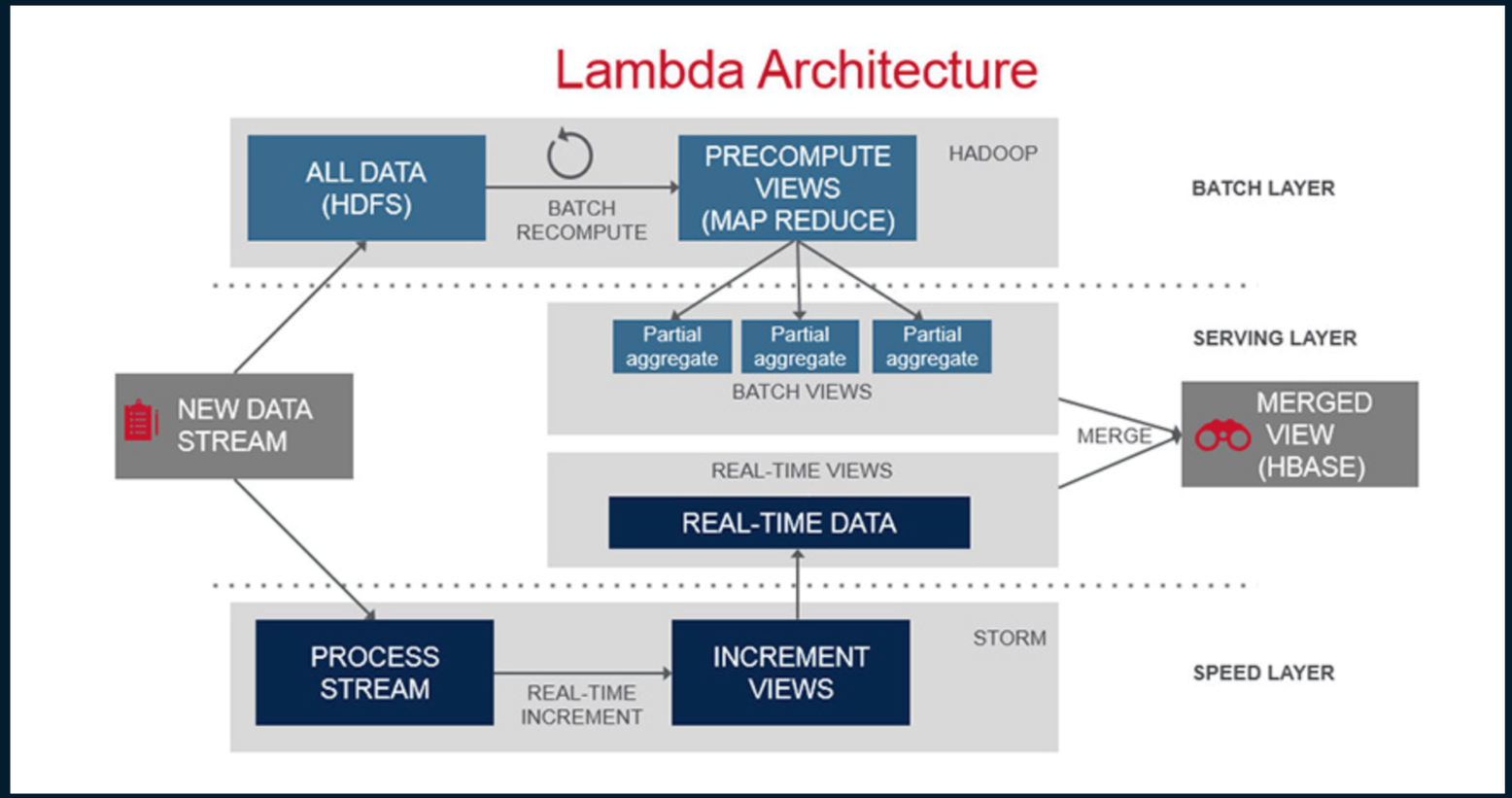
Agenda

1. Why
2. What
3. How
4. Achievement
5. Future

Why to Unify Batch and Stream

陡峭的学习曲线：两个引擎，两份代码

Steep learning curve: two engines, two codes

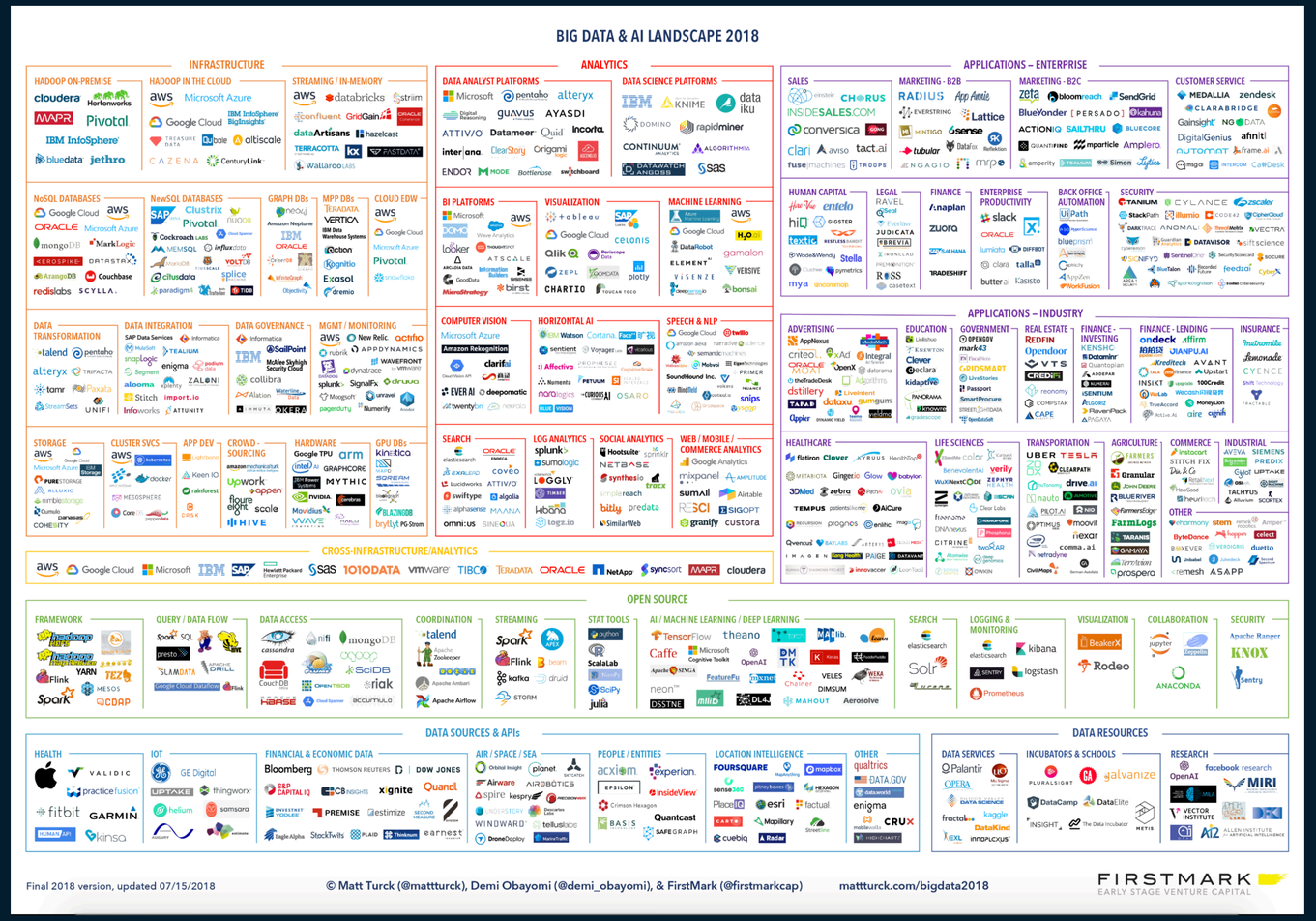


https://mapr.com/developercentral/lambda-architecture/

Why to Unify Batch and Stream

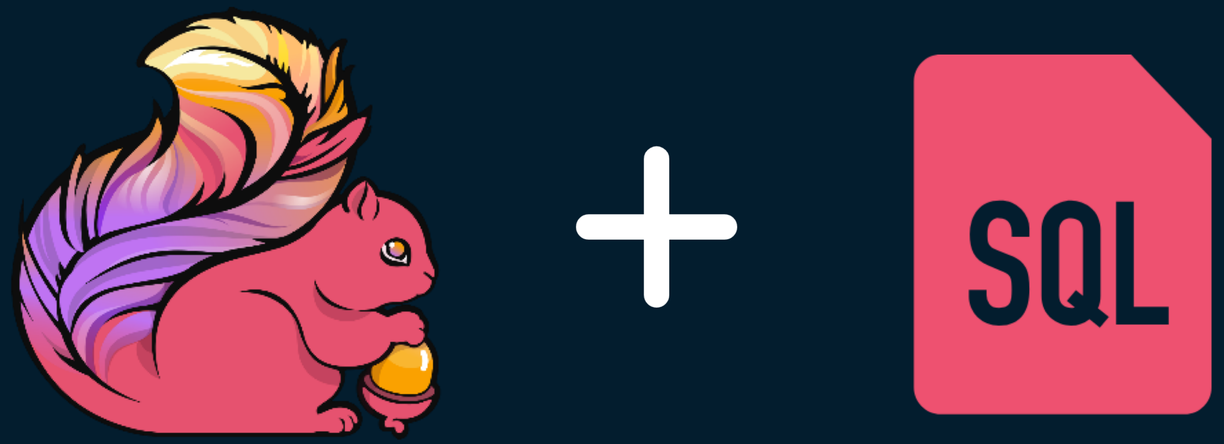
大数据和 AI 全景 – 2018

BIG DATA & LANDSCAPE 2018



http://mattturck.com/bigdata2018/

Why to Unify Batch and Stream



学习一套引擎，写一份代码，同时运行流处理和批处理，甚至更多

Learn One Engine, Write One Code, Run both Stream and Batch, Even More

Agenda

1. Why
2. What
3. How
4. Achievement
5. Future

What is an Unified SQL Engine

01 用户角度：一份代码，一样的结果

User Perspective: One Query, Same Result

02 开发角度：架构统一、代码复用、流批融合

Dev Perspective: Unified Arch, Code Reuse

User: One Query, Same Result

```
CREATE TABLE USER_SCORES (
  Name    VARCHAR,
  Score   DOUBLE,
  Time    TIMESTAMP
) WITH (
  ...
);

CREATE TABLE TOTOAL_SCORES (
  Name         VARCHAR,
  TotalScore   DOUBLE,
  MaxTime      TIMESTAMP
) WITH (
  ...
);

INSERT INTO TOTAL_SOURCES
SELECT Name, SUM(Score), MAX(Time)
FROM USER_SCORES
GROUP BY Name;
```

SQL is exactly the same in

Streaming and Batch

User: One Query, Same Result

FLINK FORWARD

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

Stream Mode:

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

[-inf, 12:01)

Name	Score	Time

[12:01, 12:04)

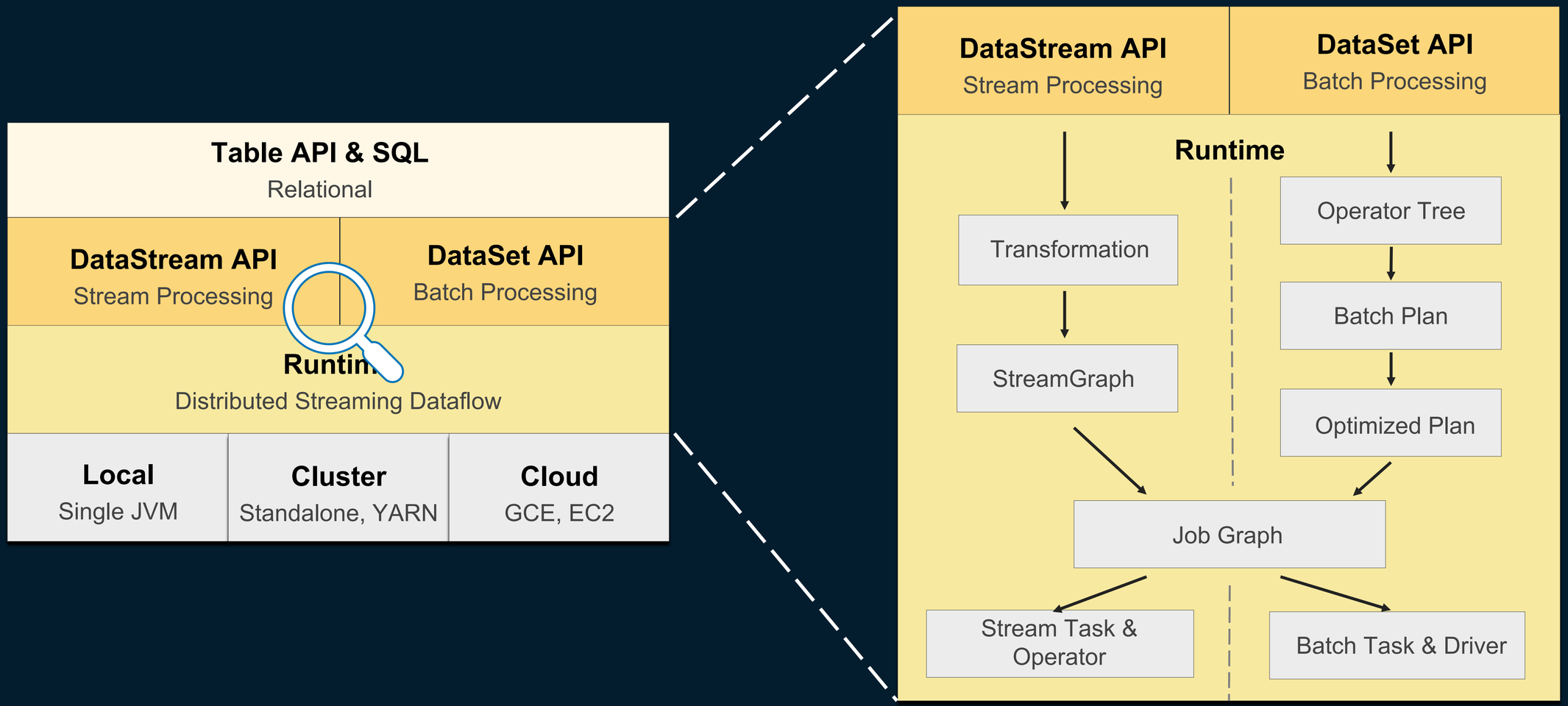
Name	Score	Time
Julie	8	12:03
Frank	3	12:03

[12:04, now)

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

Reference: https://qconsf.com/sf2017/system/files/presentation-slides/foundations_of_streaming_sql_qcon_sf_2017.pdf

Developer: Unified Architecture ?



Developer: What about Code Reuse

```
SELECT * FROM t1, t2 WHERE t1.a = t2.b where t1.a < 1
```

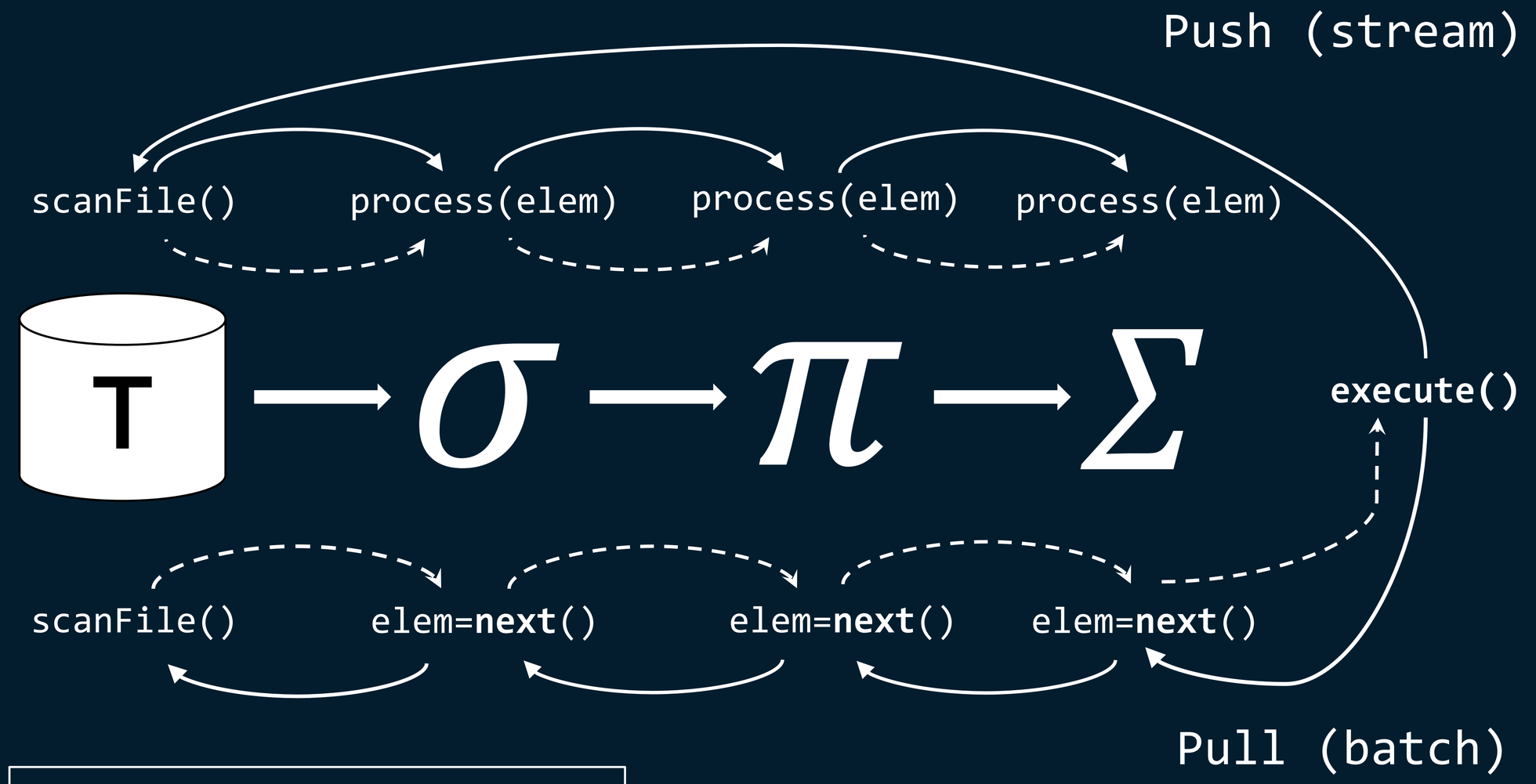
Streaming

```
DataStream[Row] t1;
DataStream[Row] t2;

t1.connect(t2)
  .keyBy(a, b)
  .transform(KeyedCoProcessOperator)

class CoProcessFunction<IN1, IN2, OUT>
```

Batch

```
DataSet[Row] t1;
DataSet[Row] t2;

t1.join(t2)
  .where(a)
  .equalTo(b)
  .with(joinFunc) // t1.a < 1
```

Developer: What about Runtime




```
SELECT SUM(T.B)
FROM T
WHERE T.A < 10
```

控制流 (Control Flow) ——→

数据流 (Data Flow) - - - - - - →

Agenda

1. Why
2. What
3. How
4. Achievement
5. Future

About me

- 伍翀（云邪，Jark）
- Apache Flink Committer
  - Contributing since Flink v1.0
  - Focusing on Flink Table & SQL since 3 years
- Software Engineer at Alibaba in Blink SQL team

How to Unify Batch and Stream

01 理论基础： 动态表
Dynamic Table

02 架构改进
Improve Architecture

03 优化器的统一
Unification of the Optimizer

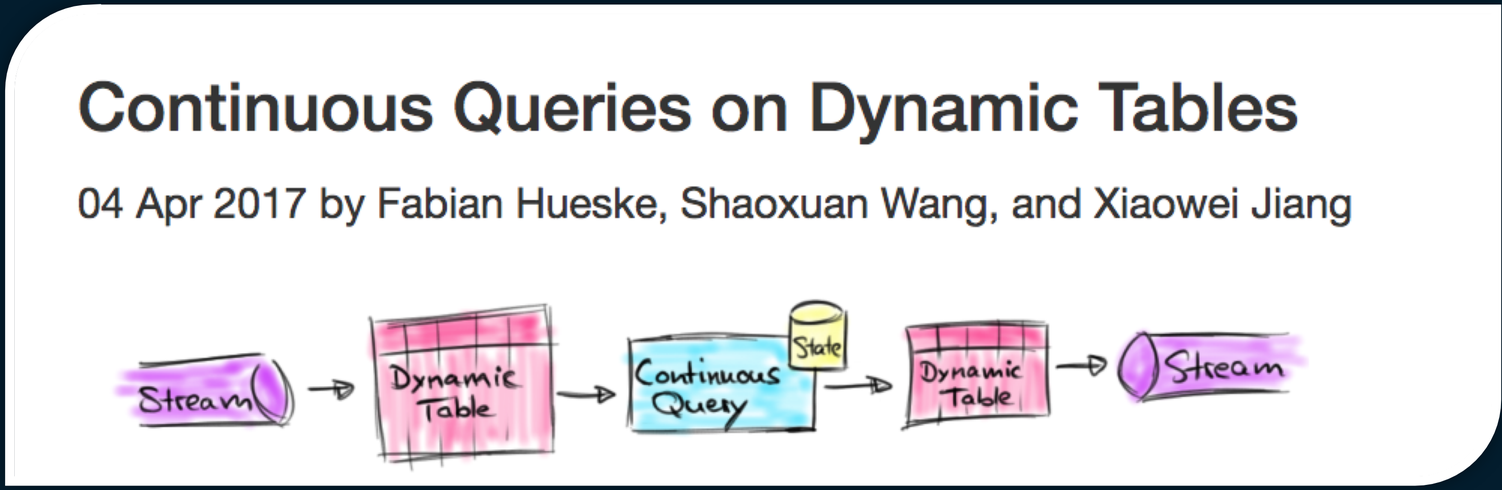
04 基础数据结构的统一
Unification of Basic Data Structure

05 物理实现的共享
Sharing of the Runtime Implementation

01 Dynamic Table

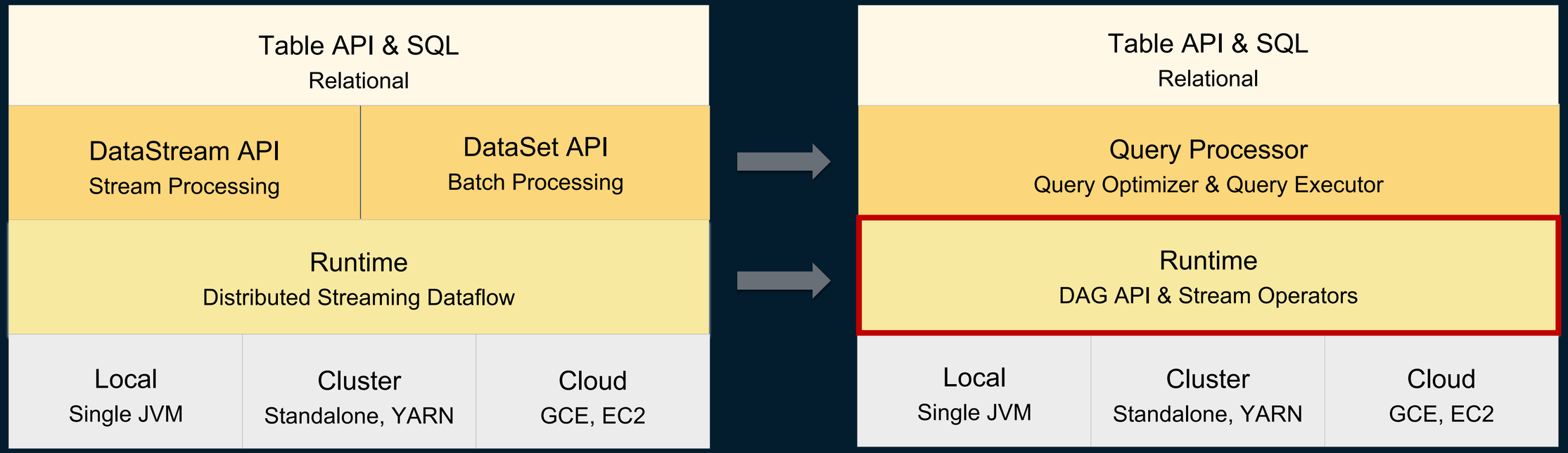
理论基础：流表对偶性，动态表

Theoretical Basis: Duality of Streams and Tables, Dynamic Table

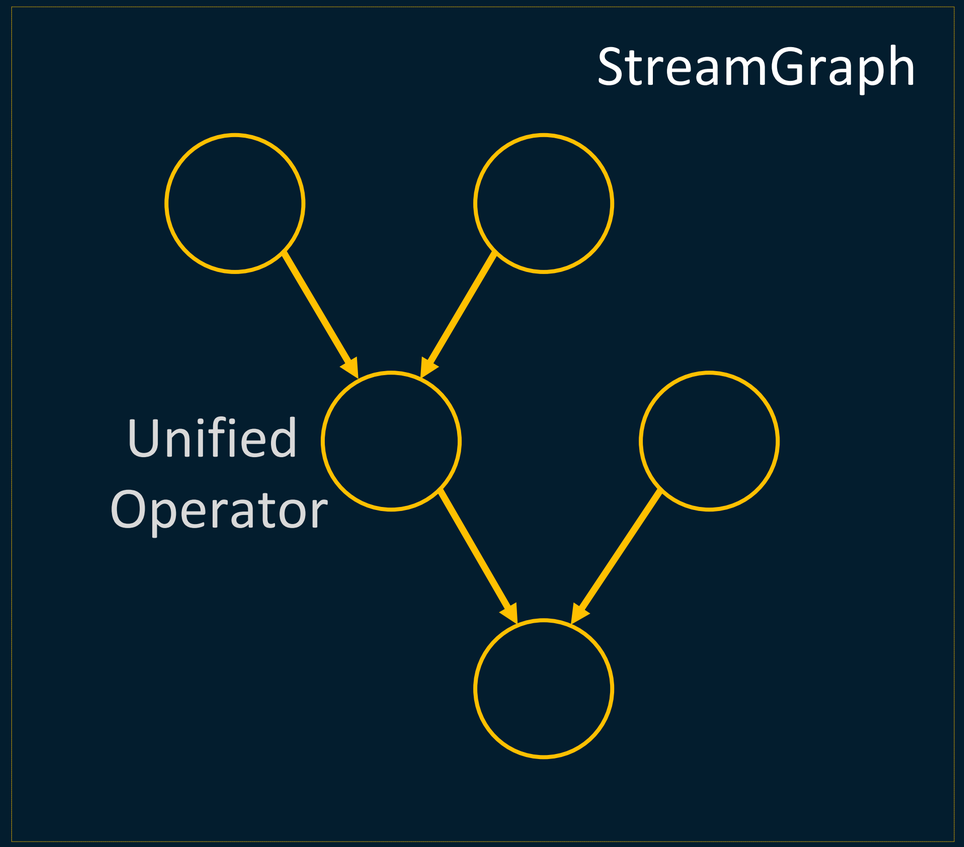


https://flink.apache.org/news/2017/04/04/dynamic-tables.html

02 Improve Architecture



02.1 Unified Operator Framework

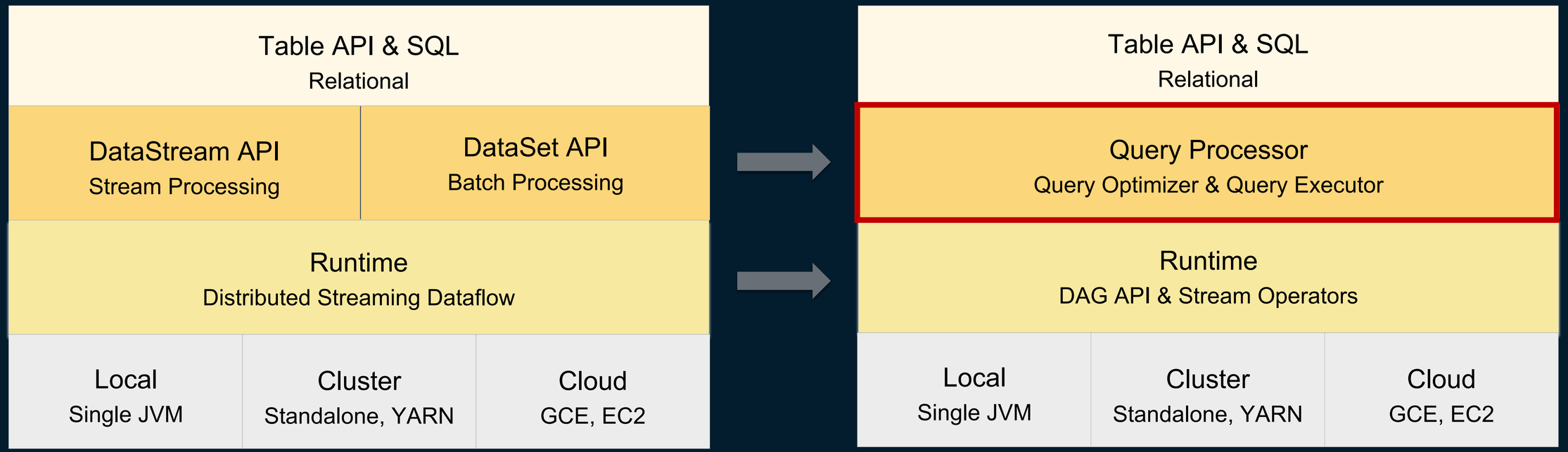


Unified Operator Abstraction

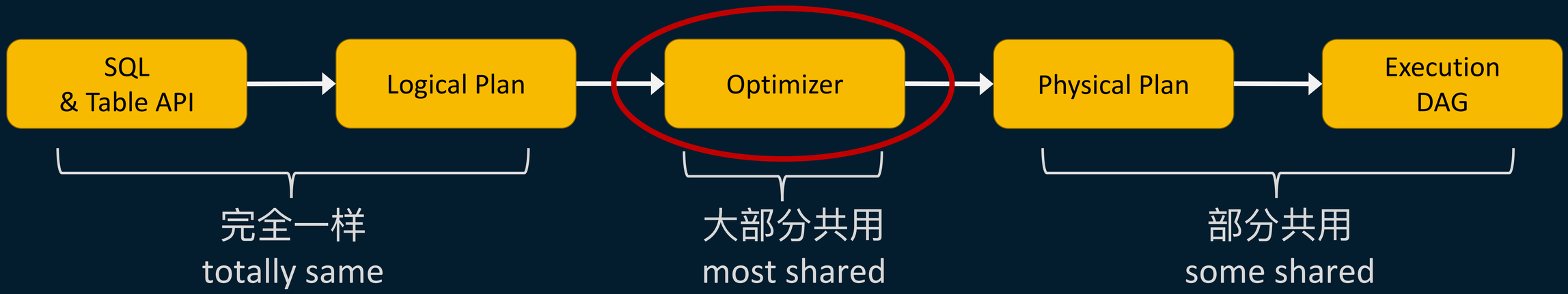
- Driver => StreamOperator
- Operators can choose inputs
- Flexible Chaining

@dev [DISCUSS] Unified Core API for Streaming and Batch: https://goo.gl/CvfKuZ

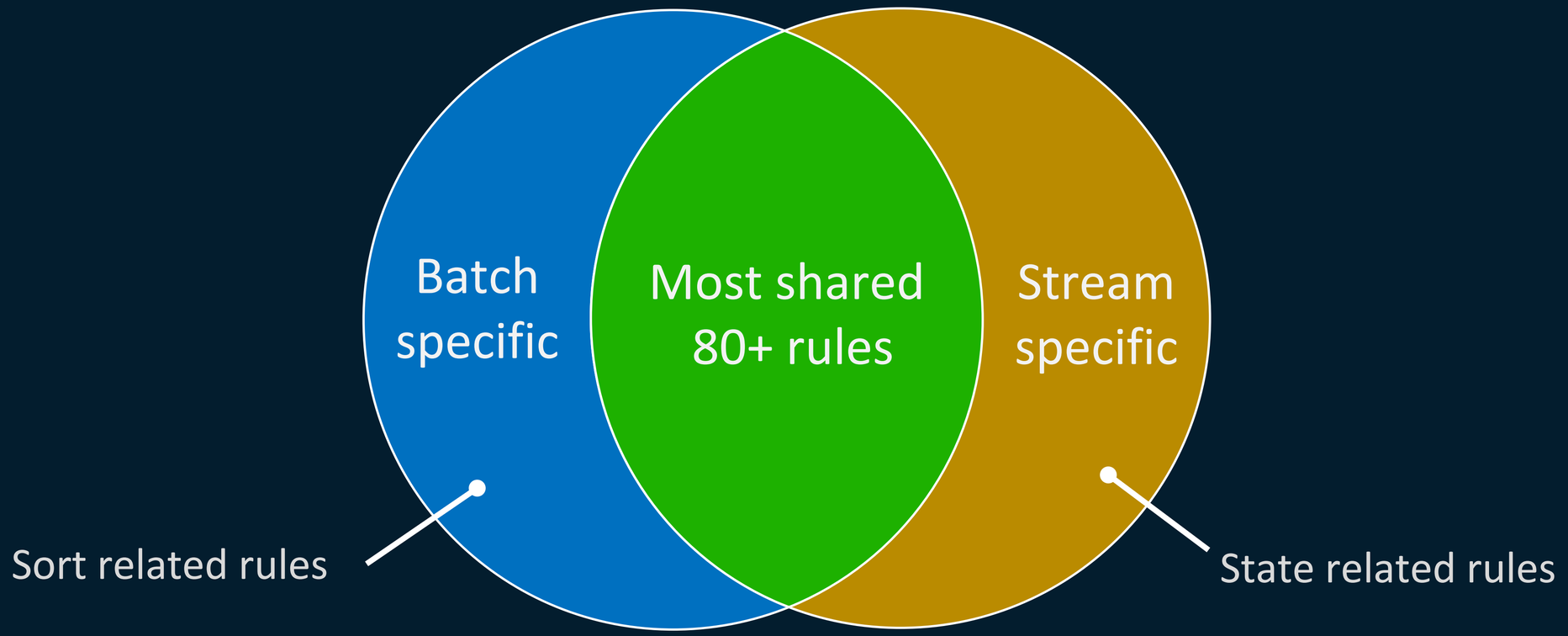
02 Improve Architecture



02.2 Unified Query Processing



03 Unification of the Optimizer



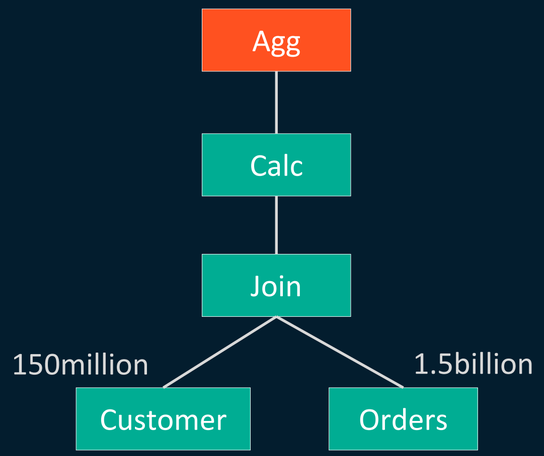
03.1 push Aggregate past Join

6 hours -> 14min (25x) in stream

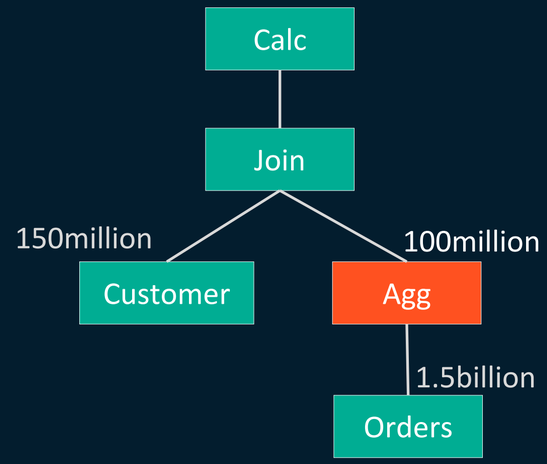
Simplified TPCH13

```
SELECT
  c.c_custkey,
  COUNT(o.o_orderkey)
FROM customer c
LEFT JOIN orders o
ON c.c_custkey = o.o_custkey
GROUP BY c.c_custkey
```

Before Optimization



After Optimization



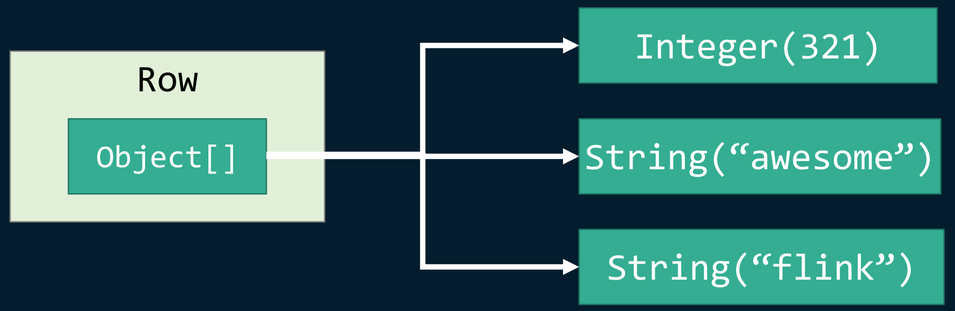
流与批都适用于这个优化规则

Batch and Stream both use this rule .

04 Unification of Basic Data Structure

Old row format: Row

- Java 对象的空间开销高
  High space overhead of Java objects
- 主类型的装箱和拆箱开销
  Boxing & Unboxing for primitive types
- 昂贵的 hashCode() 和 (反)序列化
  Expensive hashCode() & (de)serialization



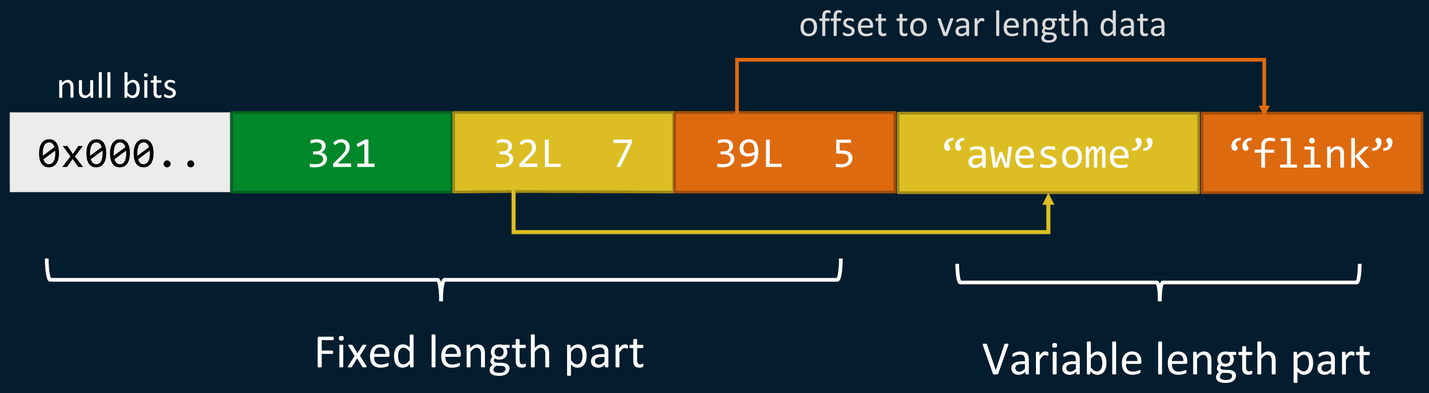
04 Unification of Basic Data Structure

New row format: BinaryRow

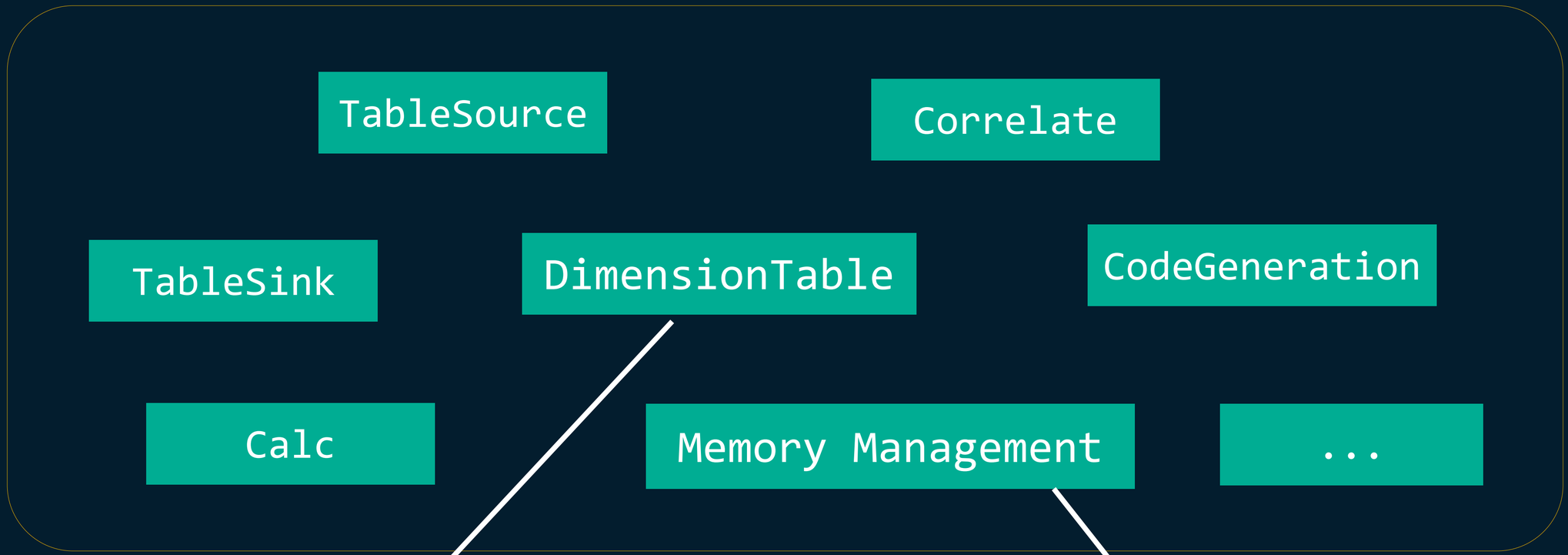
- 避免了很多反序列化开销
  Reduce lots of deserialization cost
- 与内存管理紧密结合
  Tight integration with memory management
- CPU 缓存友好
  CPU Cache Friendly

不仅在批处理中表现出色，
在流处理中也收获了一倍的提升

Not only worked perfectly in batch,
but also increase 1X throughput in streaming cases

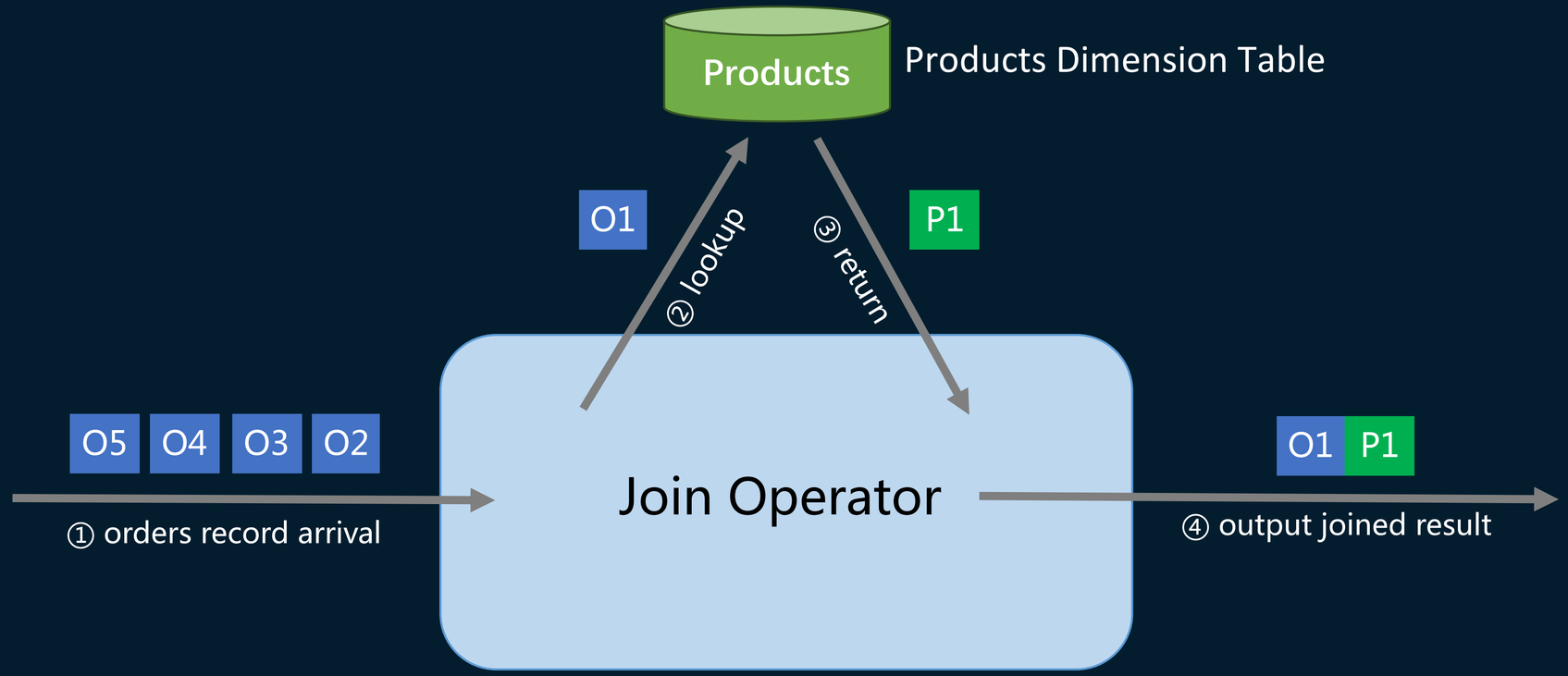


05 Sharing of Runtime Implementation



- 维表关联
  Join a Dimension Table
- Micro-Batch的内存管理
  Memory Management of Micro Batch

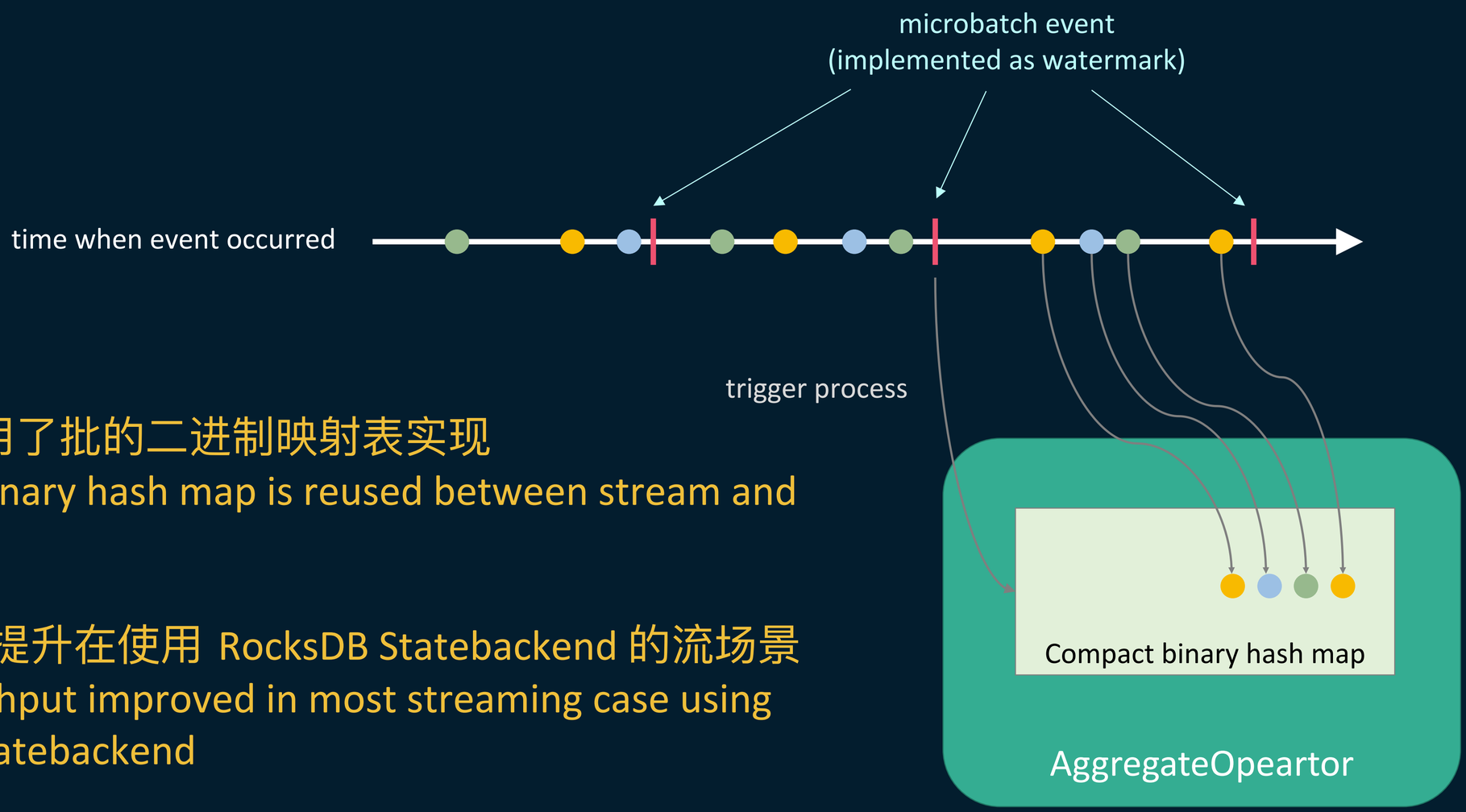
05.1 Join a Dimension Table



批直接复用了流的实现，当维表超大时使用 lookup 要比 scan 更划算

Batch reuses this operator of Streaming
which benefits a lot when dimension table is huge

05.2 Micro-Batch for Streaming



流直接复用了批的二进制映射表实现
Compact binary hash map is reused between stream and batch

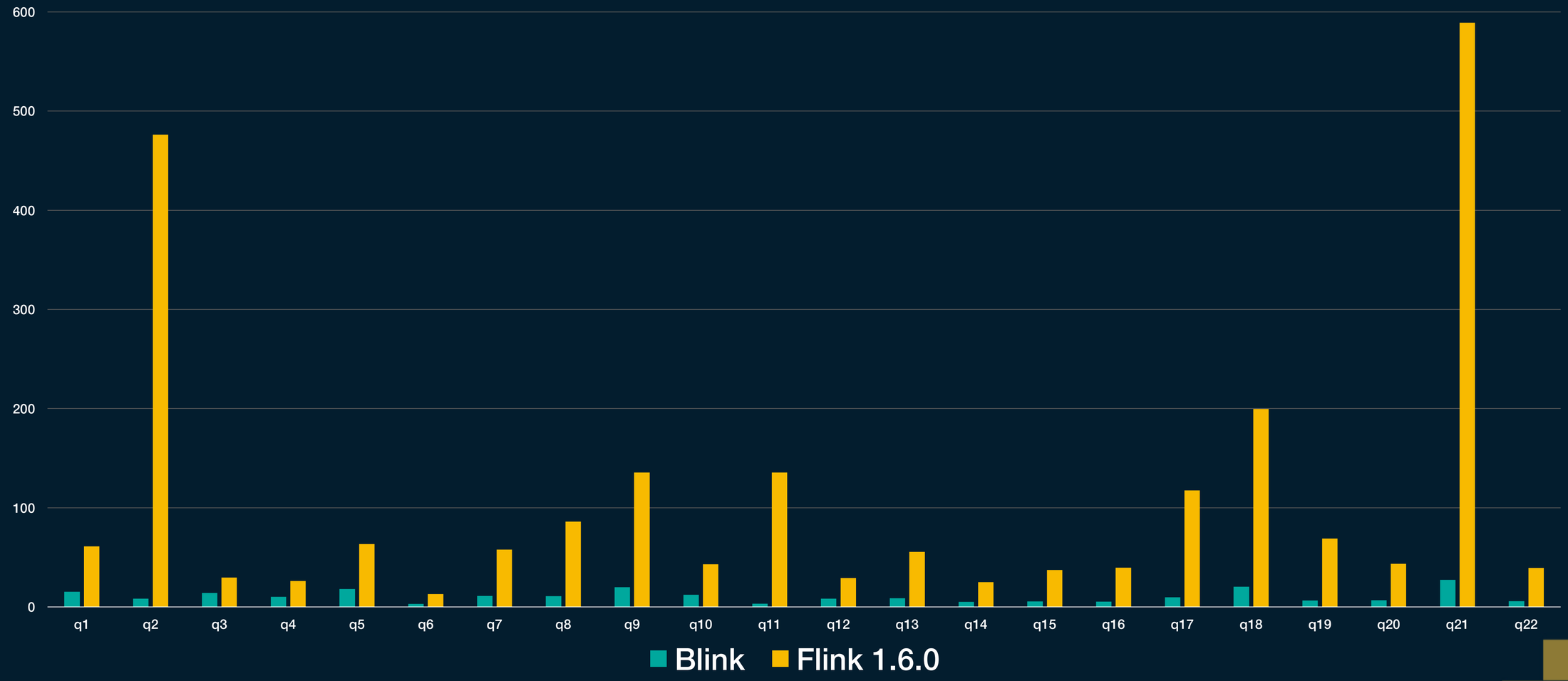
10倍 吞吐提升在使用 RocksDB Statebackend 的流场景
10X throughput improved in most streaming case using RocksDB statebackend

Agenda

1. Why
2. What
3. How
4. Achievement
5. Future

Achievement

TPC-H Results for Batch (lower is better)



Achievement

01 批的性能在 Flink 社区版本上提升 10倍

Outperform Flink By 10X in Batch

02 流也成功攻克了 TPC–H，业内首例

Streaming also successfully conquers TPC-H

03 2018天猫双11，流计算峰值达 17.18亿条/秒

Streaming process exceeded 1.7 billion records/sec, in 2018 Alibaba Global Shopping Festival

Future Plan

01 开源
Open Source

02 流与批的融合，边界模糊化
Hybrid of Stream and Batch


THANKS