Large scale stream processing with Apache Flink

Nikolay Stoitsev

Sr. Software Engineer at Uber Tech Sofia

Uber

Stream Processing?

Stream Processing?

User Interaction Logs

Stream Processing?

User Interaction Logs

Application Logs

Stream Processing?

User Interaction Logs

Application Logs

Sensor Data

Stream Processing?

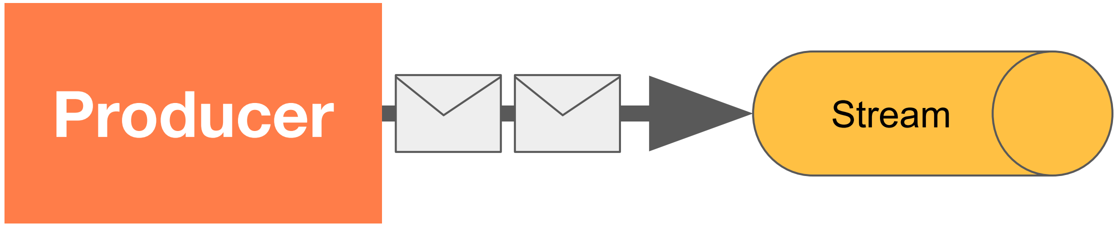
User Interaction Logs

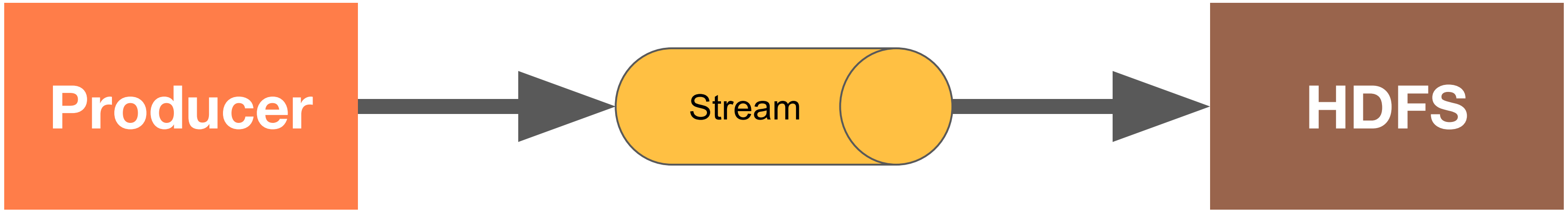
Application Logs

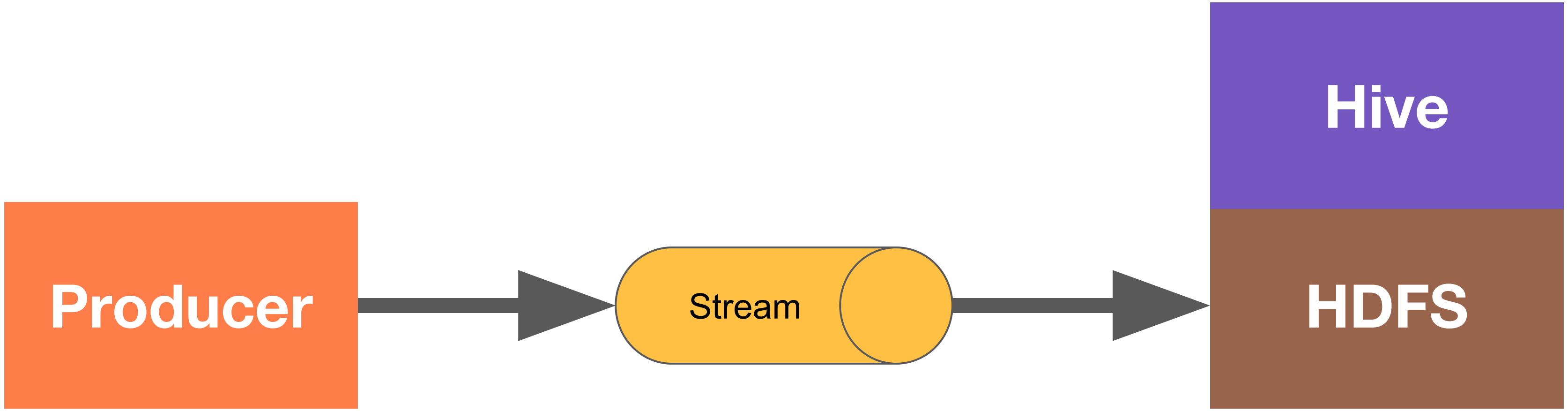
Sensor Data

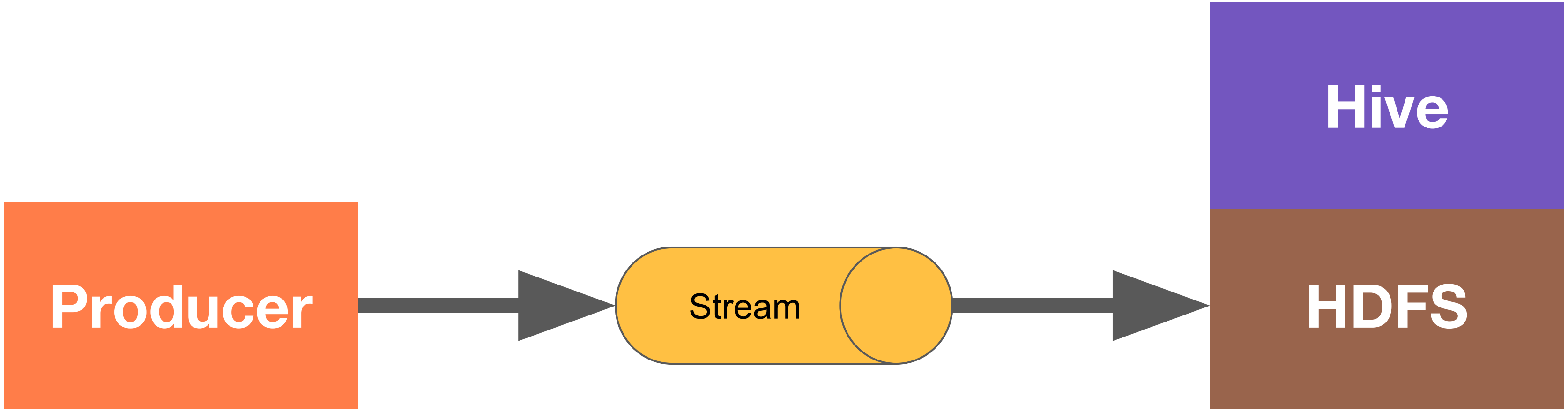
Database Commit Logs

Infinite Dataset

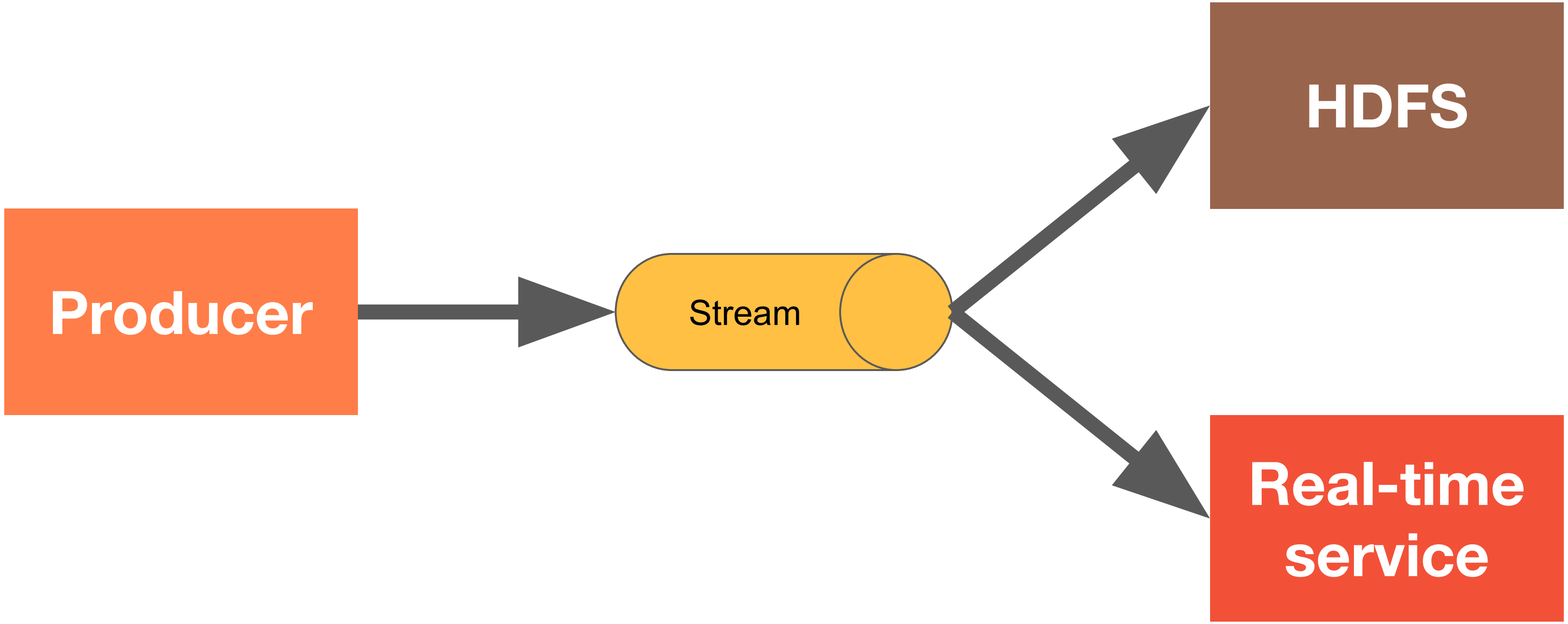

Producer
Stream


Producer
Stream
HDFS


Producer
Stream
Hive
HDFS



Big Latency


Producer
Stream
HDFS
Real-time service

Apache Storm

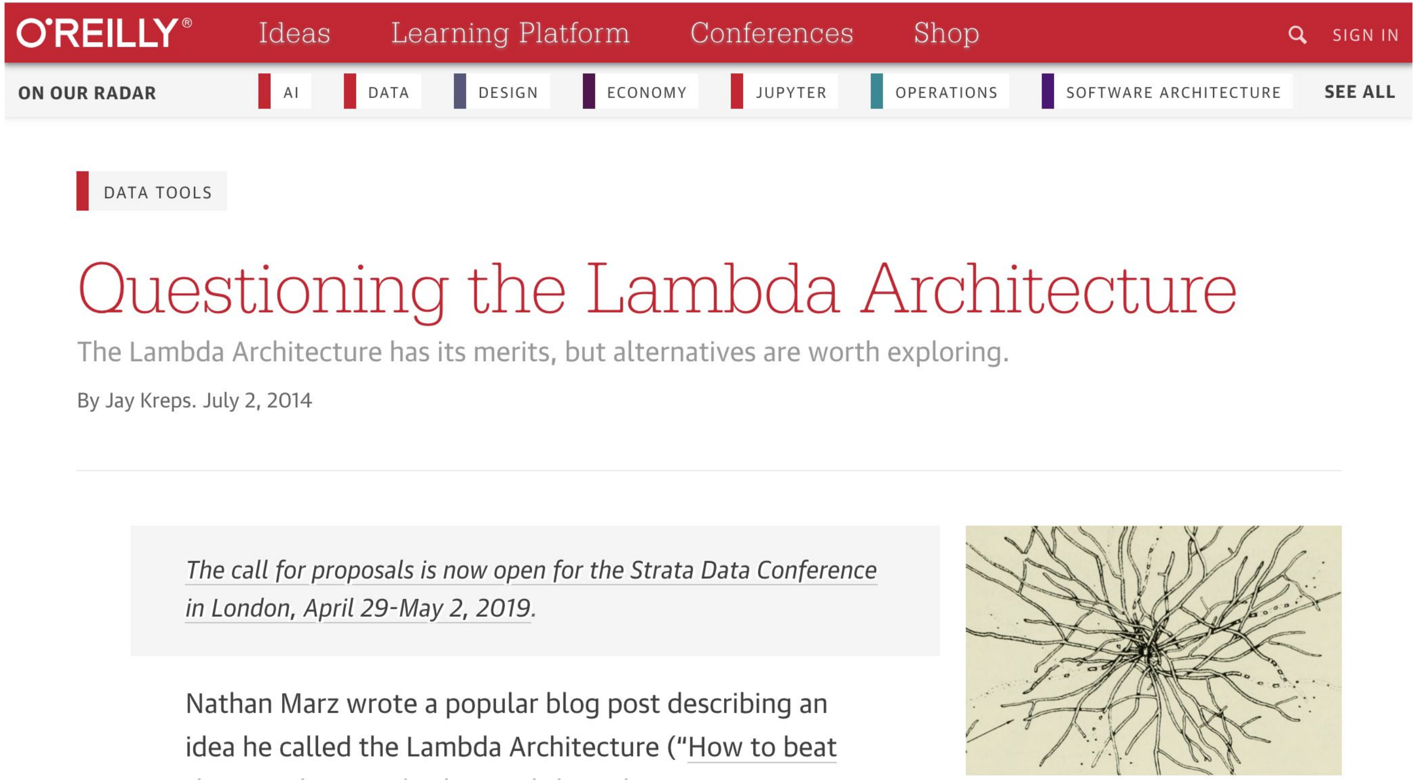
storm.apache.org

High-latency & accurate

vs.

Low-latency & approximation

Lambda architecture



Questioning the Lambda Architecture

The Lambda Architecture has its merits, but alternatives are worth exploring.

By Jay Kreps. July 2, 2014

The call for proposals is now open for the Strata Data Conference in London, April 29-May 2, 2019.

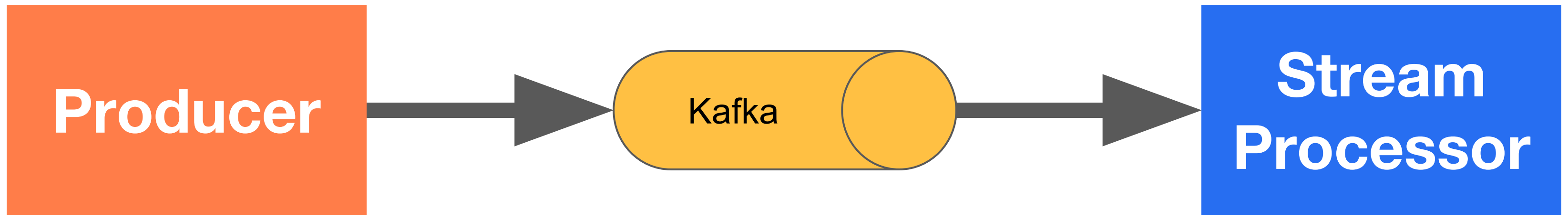
Nathan Marz wrote a popular blog post describing an idea he called the Lambda Architecture ("How to beat

https://www.oreilly.com/ideas/questioning-the-lambda-architecture

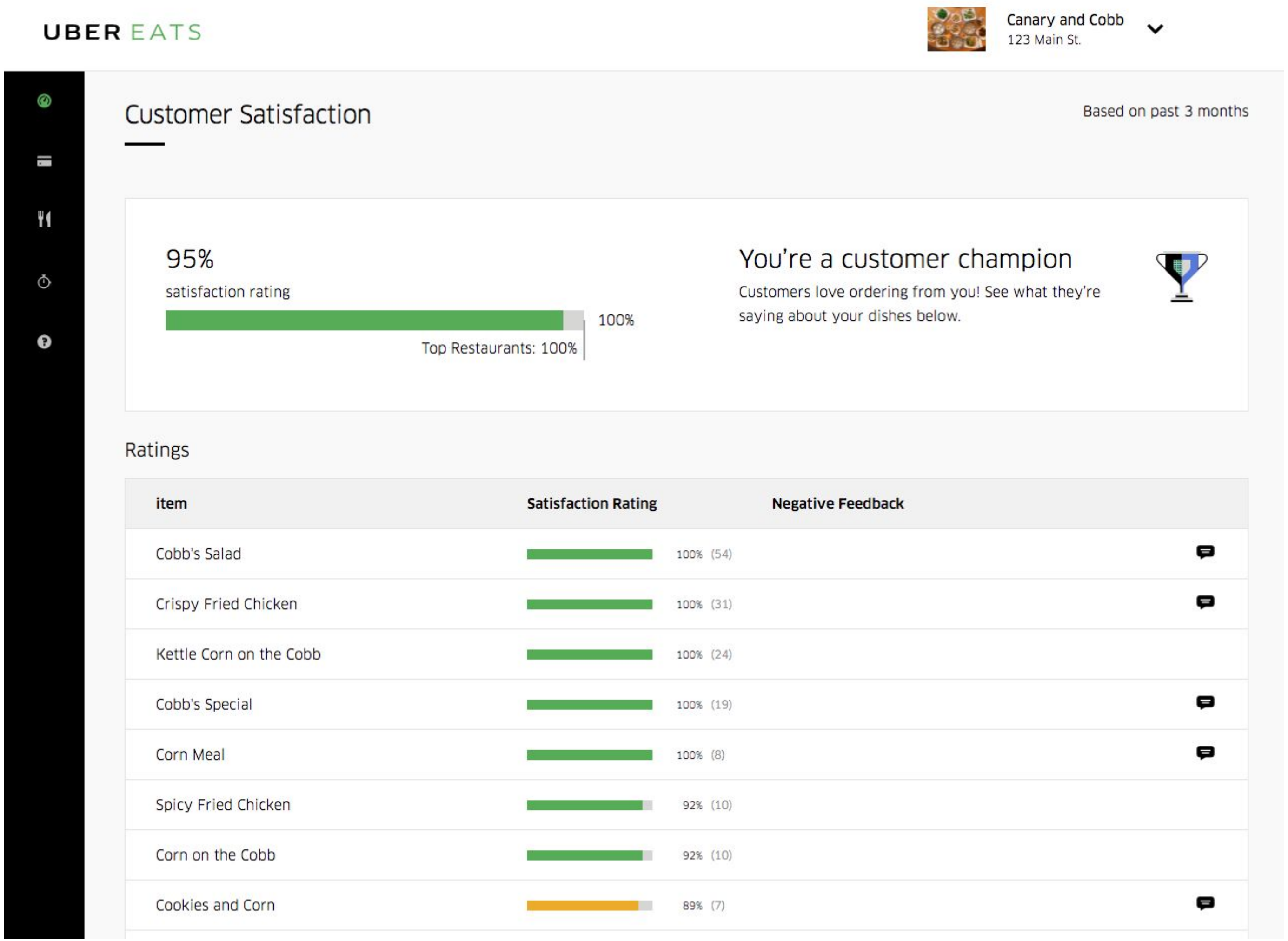
Kappa Architecture

Use Apache Kafka

Durable, scalable, fault-tolerant


Producer
Kafka
Stream Processor

UBER
eats


UBER EATS

Canary and Cobb
123 Main St.

Customer Satisfaction

Based on past 3 months

95%
satisfaction rating

100%

Top Restaurants: 100%

You're a customer champion
Customers love ordering from you! See what they're saying about your dishes below.

Ratings

item	Satisfaction Rating	Negative Feedback
Cobb's Salad	100% (54)	
Crispy Fried Chicken	100% (31)	
Kettle Corn on the Cobb	100% (24)	
Cobb's Special	100% (19)	
Corn Meal	100% (8)	
Spicy Fried Chicken	92% (10)	
Corn on the Cobb	92% (10)	
Cookies and Corn	89% (7)	

Metrics we want to track

Net payout

Daily items sold

Weekly items sold

Order acceptance rate

Order preparation speed

Item rating

Real time

Scalable

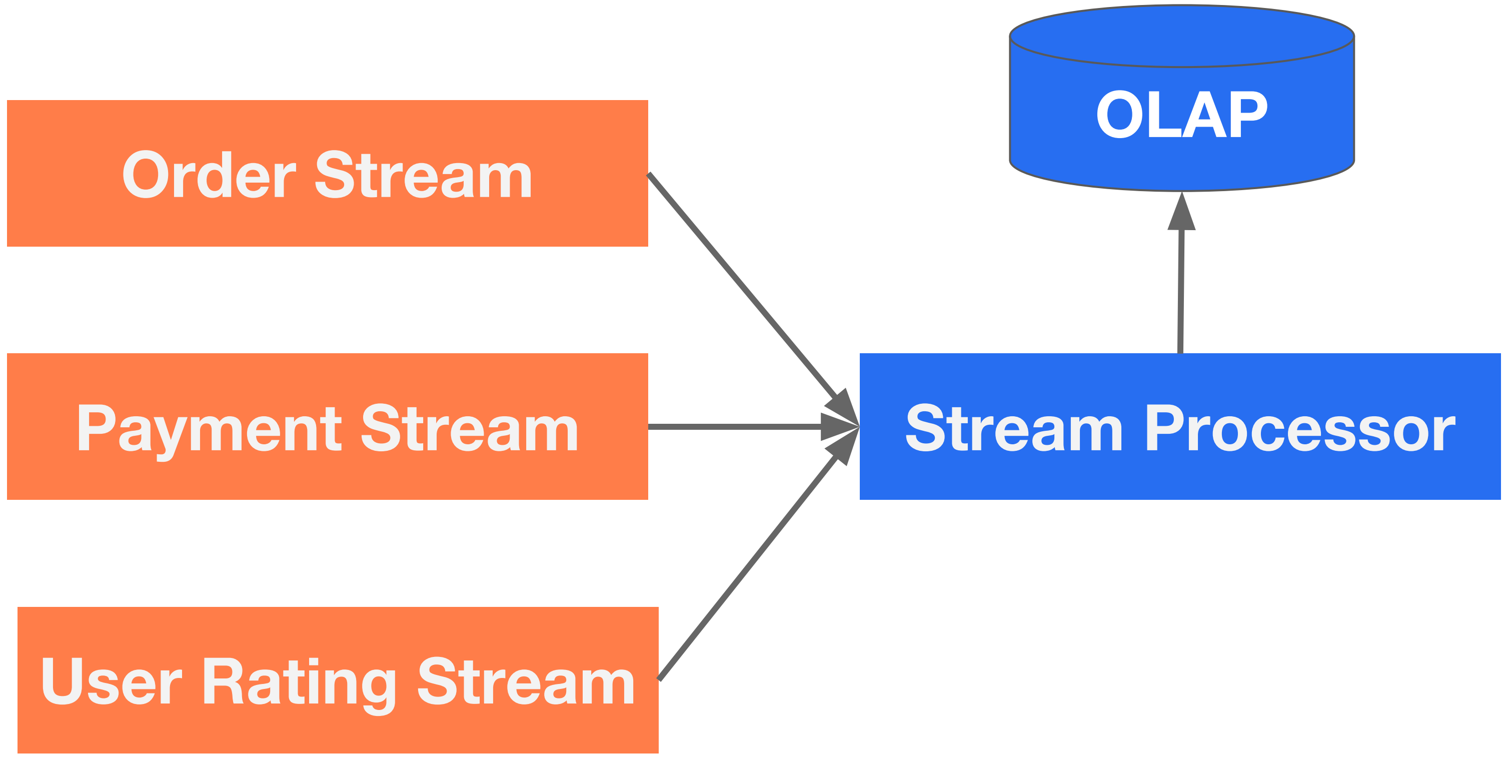
Granular

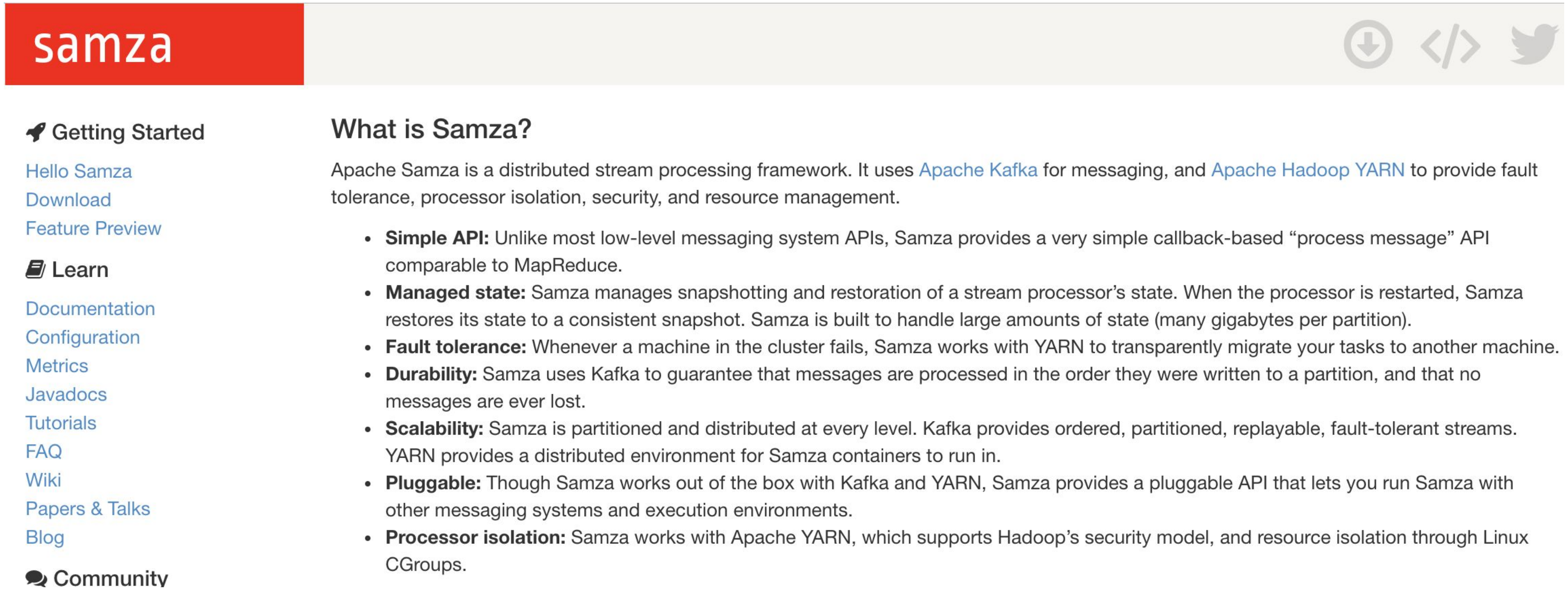
Highly available

Order Stream

Payment Stream

User Rating Stream


Order Stream
Payment Stream
User Rating Stream
Stream Processor
OLAP



What is Samza?

Apache Samza is a distributed stream processing framework. It uses Apache Kafka for messaging, and Apache Hadoop YARN to provide fault tolerance, processor isolation, security, and resource management.

- **Simple API:** Unlike most low-level messaging system APIs, Samza provides a very simple callback-based "process message" API comparable to MapReduce.
- **Managed state:** Samza manages snapshotting and restoration of a stream processor's state. When the processor is restarted, Samza restores its state to a consistent snapshot. Samza is built to handle large amounts of state (many gigabytes per partition).
- **Fault tolerance:** Whenever a machine in the cluster fails, Samza works with YARN to transparently migrate your tasks to another machine.
- **Durability:** Samza uses Kafka to guarantee that messages are processed in the order they were written to a partition, and that no messages are ever lost.
- **Scalability:** Samza is partitioned and distributed at every level. Kafka provides ordered, partitioned, replayable, fault-tolerant streams. YARN provides a distributed environment for Samza containers to run in.
- **Pluggable:** Though Samza works out of the box with Kafka and YARN, Samza provides a pluggable API that lets you run Samza with other messaging systems and execution environments.
- **Processor isolation:** Samza works with Apache YARN, which supports Hadoop's security model, and resource isolation through Linux CGroups.

samza.apache.org

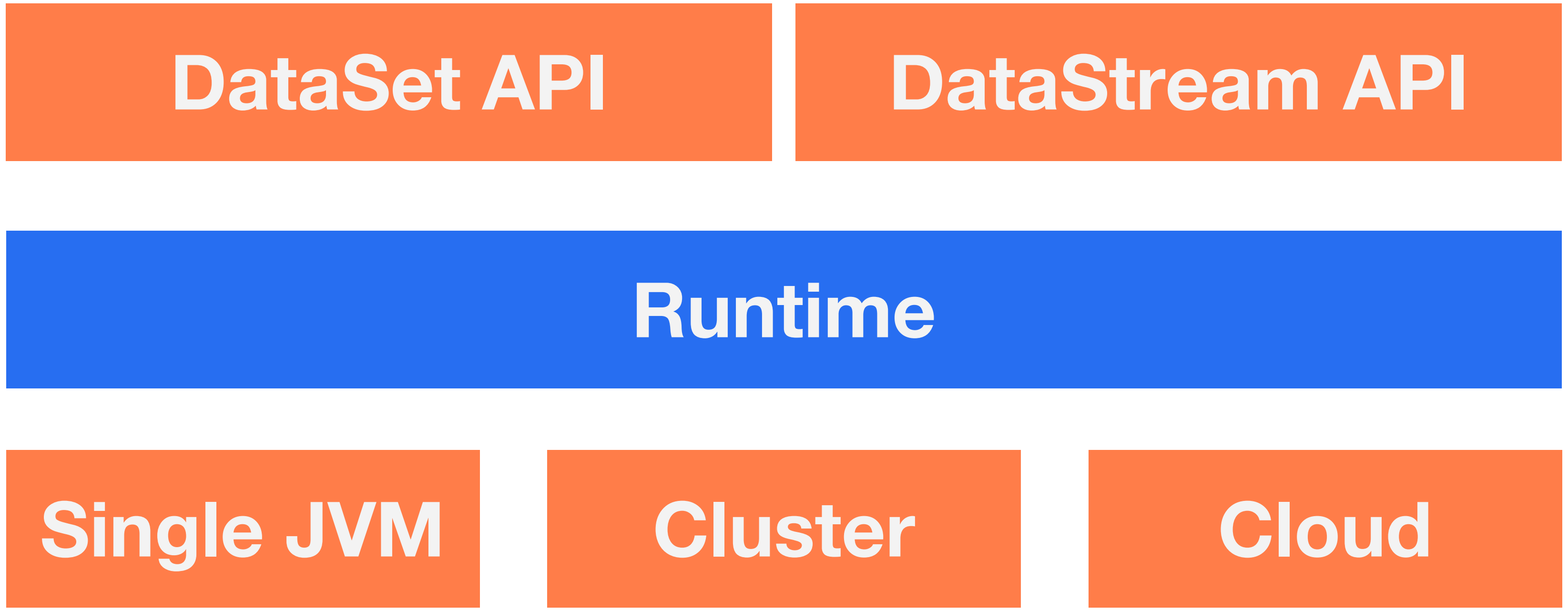
Apache Flink

flink.apache.org

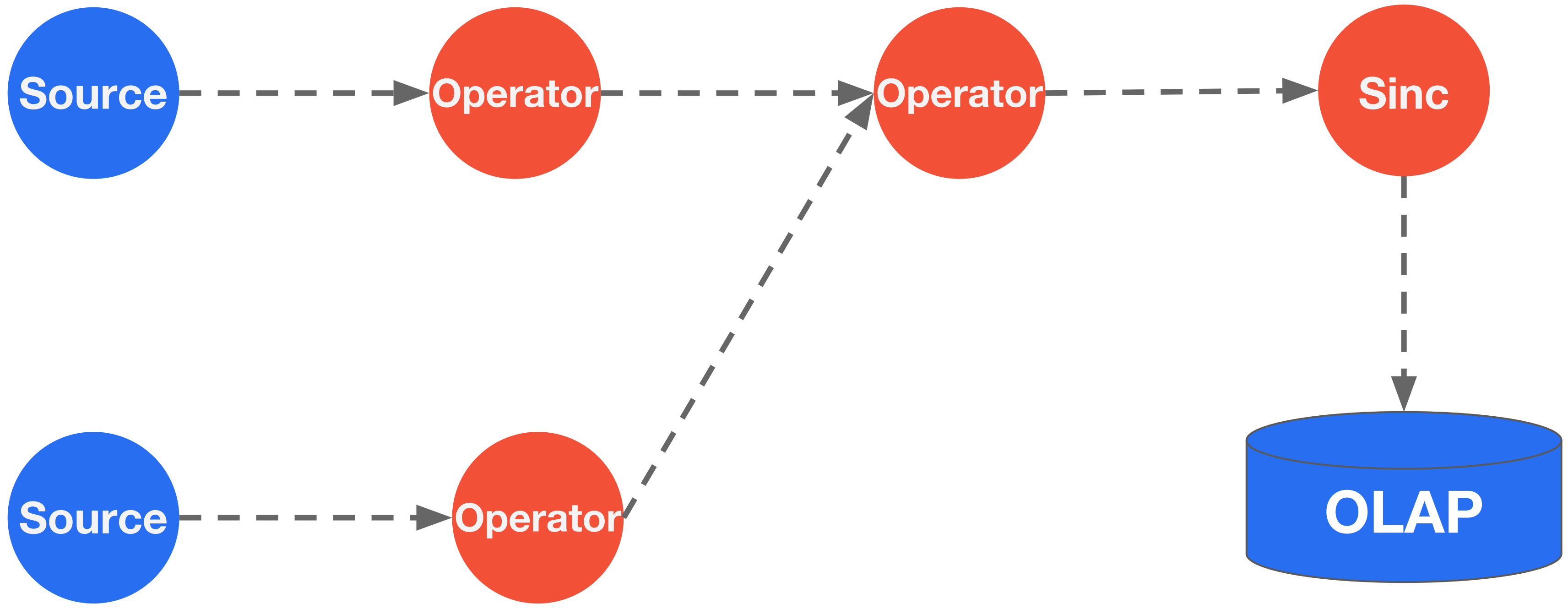
Everything is a batch

vs.

Everything is a stream


DataSet API
DataStream API
Runtime
Single JVM
Cluster
Cloud

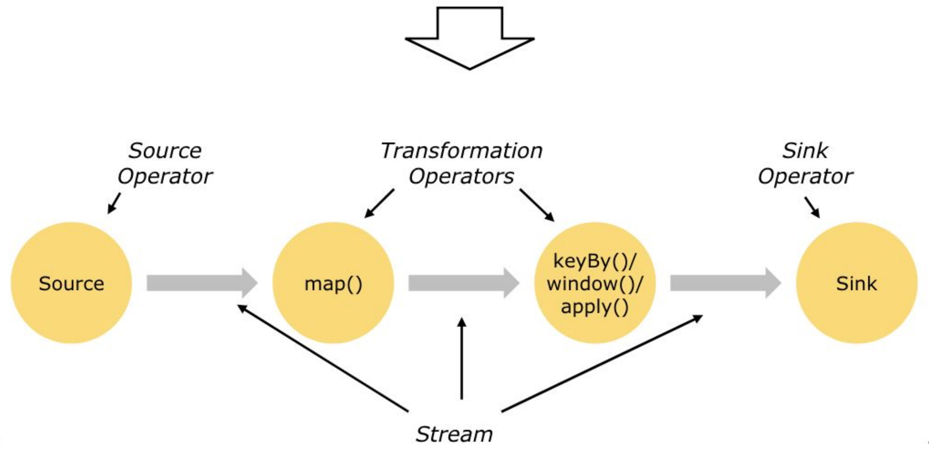
Dataflow graph


Source
Operator
Operator
Sinc
Source
Operator
OLAP

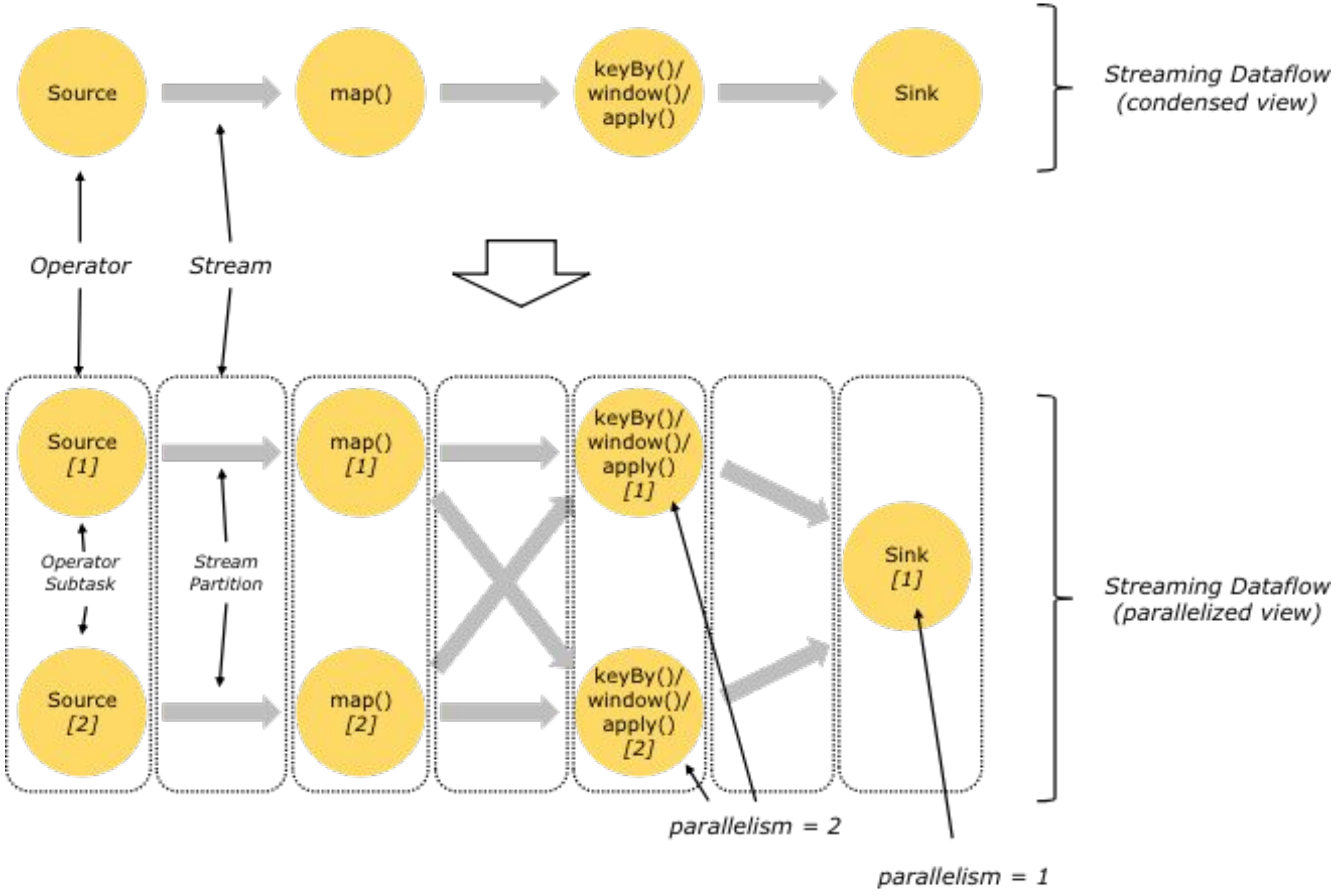
```java
DataStream<String> lines = env.addSource(
                                  new FlinkKafkaConsumer<>(…));

DataStream<Event> events = lines.map((line) -> parse(line));

DataStream<Statistics> stats = events
        .keyBy("id")
        .timeWindow(Time.seconds(10))
        .apply(new MyWindowAggregationFunction());

stats.addSink(new RollingSink(path));
```

https://ci.apache.org/projects/flink/flink-docs-release-1.6/concepts/programming-model.html

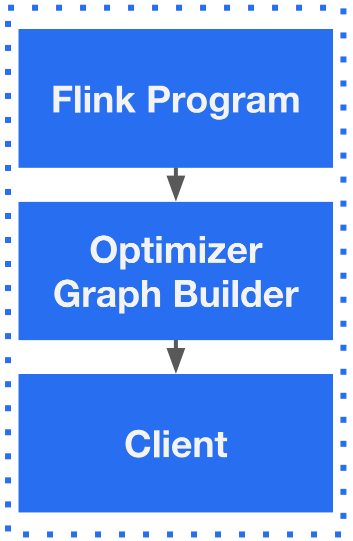
```
DataStream<String> lines = env.addSource(
                               new FlinkKafkaConsumer<>(…));         Source

DataStream<Event> events = lines.map((line) -> parse(line));          Transformation

DataStream<Statistics> stats = events
        .keyBy("id")
        .timeWindow(Time.seconds(10))                                  Transformation
        .apply(new MyWindowAggregationFunction());

stats.addSink(new RollingSink(path));                                  Sink
```

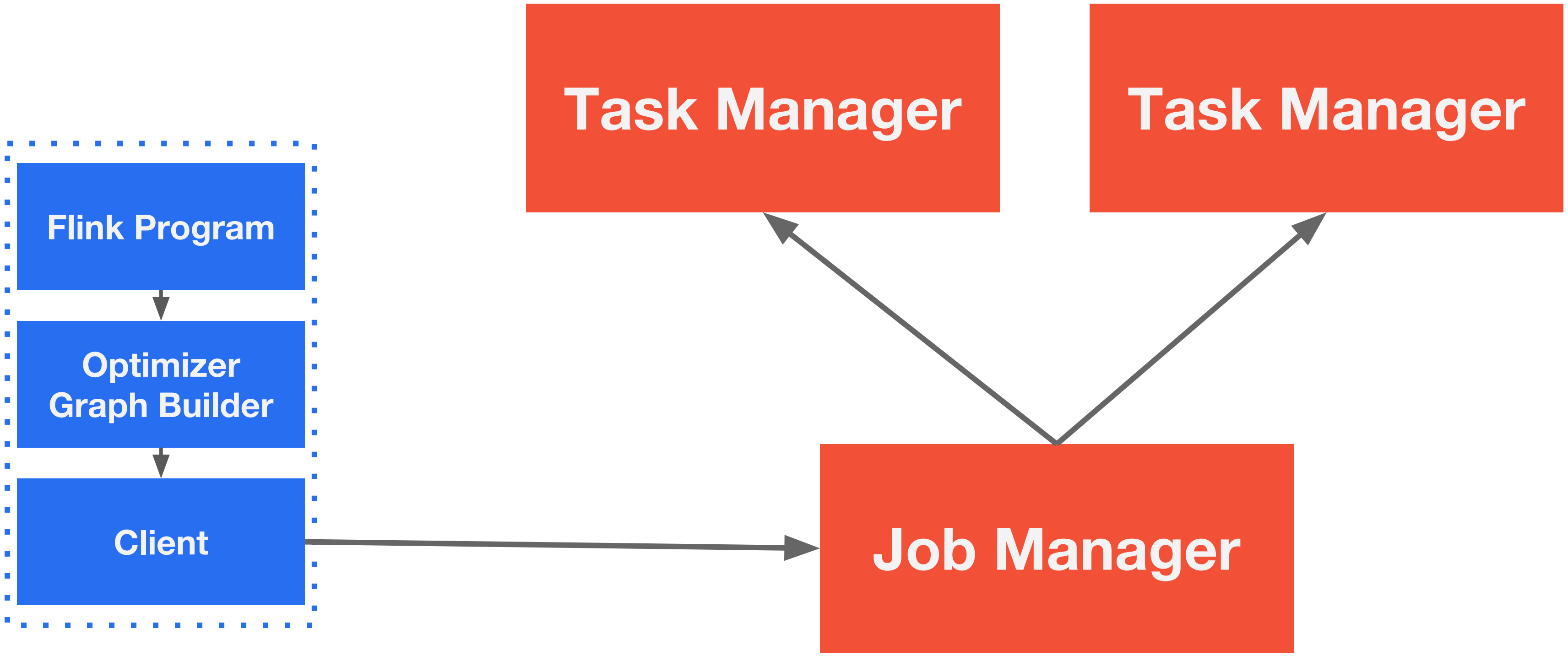


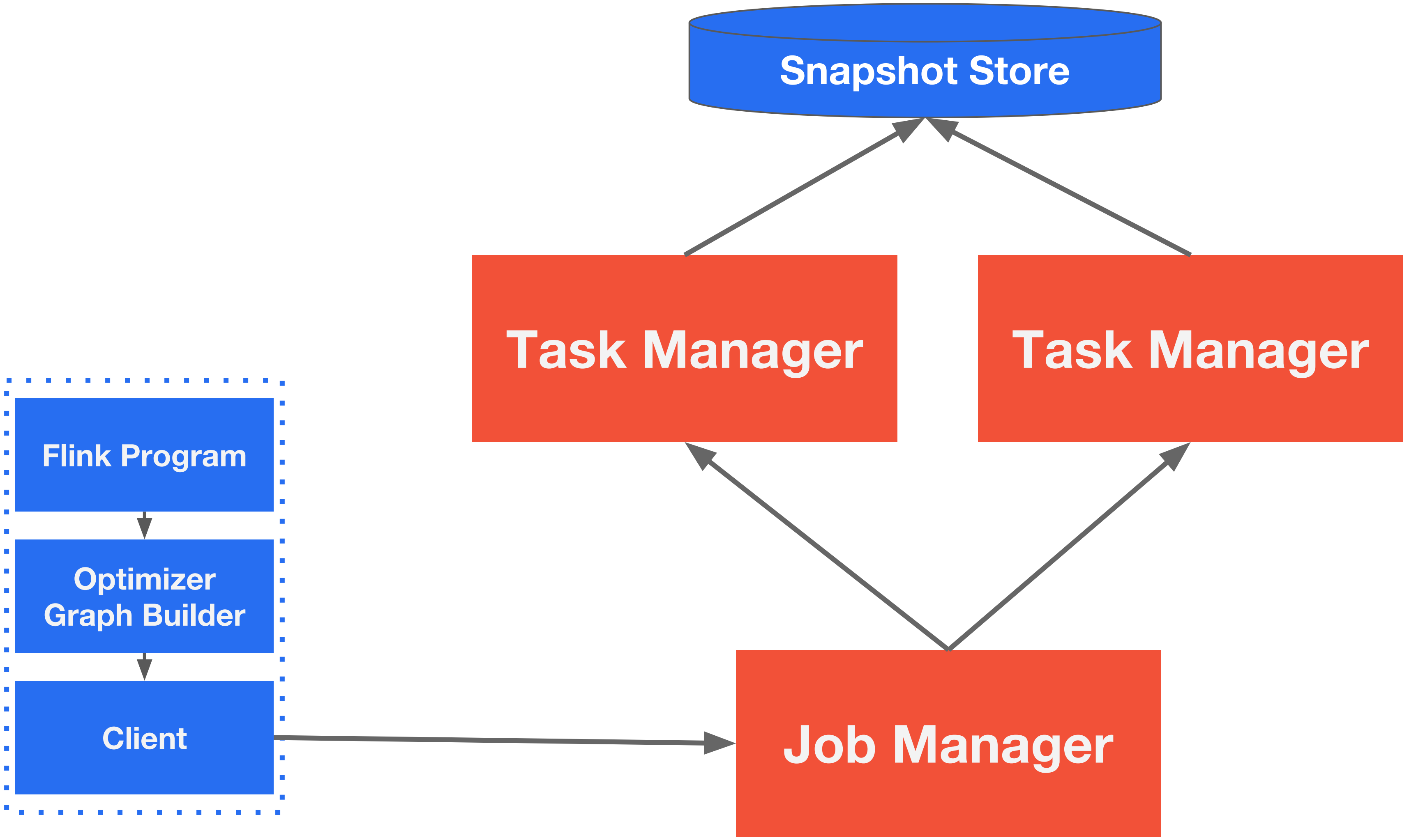
https://ci.apache.org/projects/flink/flink-docs-release-1.6/concepts/programming-model.html


Source
map()
keyBy()/ window()/ apply()
Sink
Streaming Dataflow (condensed view)
Operator
Stream
Source [1]
Source [2]
Operator Subtask
Stream Partition
map() [1]
map() [2]
keyBy()/ window()/ apply() [1]
keyBy()/ window()/ apply() [2]
Sink [1]
Streaming Dataflow (parallelized view)
parallelism = 2
parallelism = 1

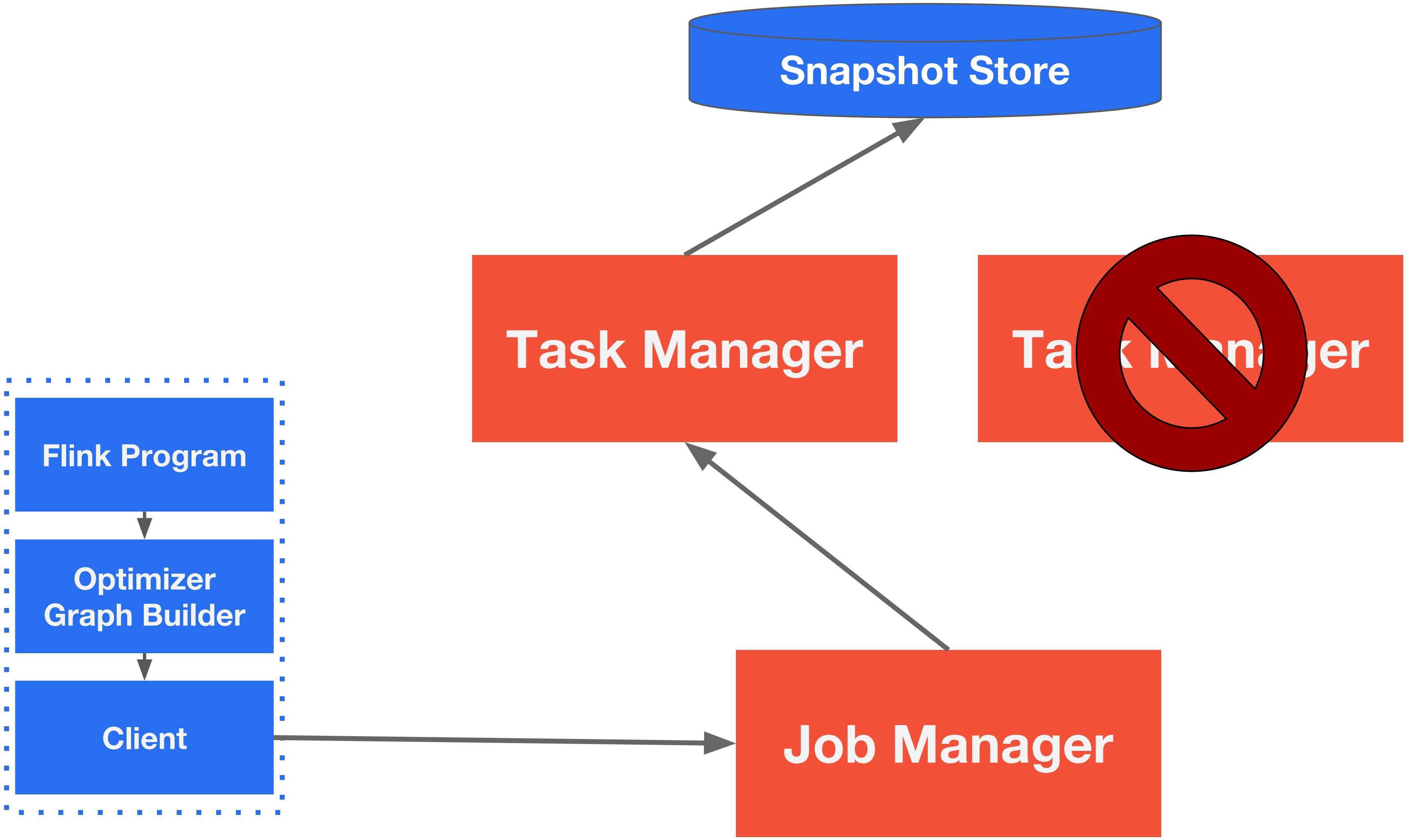
https://ci.apache.org/projects/flink/flink-docs-release-1.6/concepts/programming-model.html


Flink Program
Optimizer
Graph Builder
Client


Flink Program
Optimizer
Graph Builder
Client
Job Manager
Task Manager
Task Manager

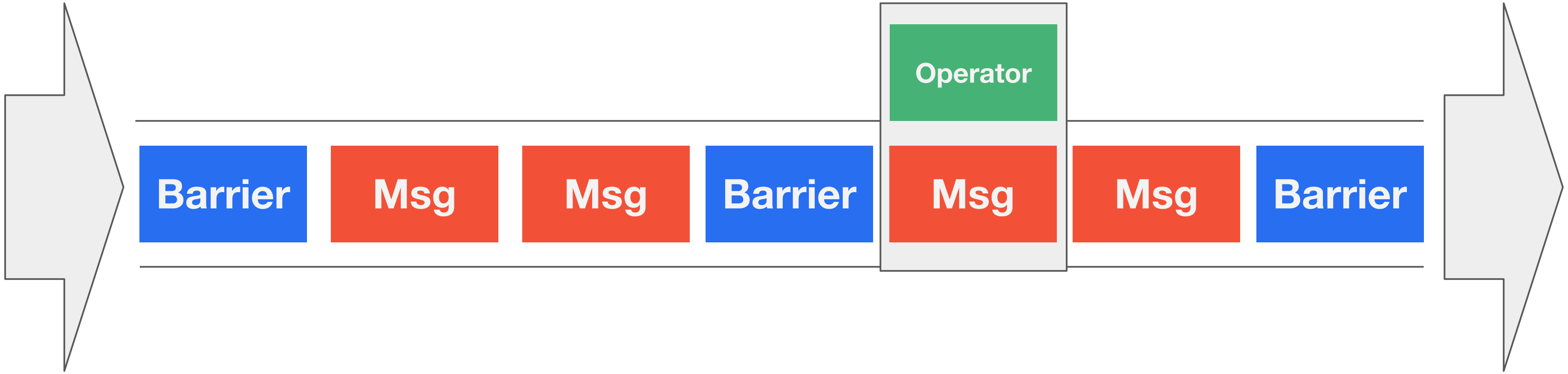

Snapshot Store
Task Manager
Task Manager
Flink Program
Optimizer
Graph Builder
Client
Job Manager

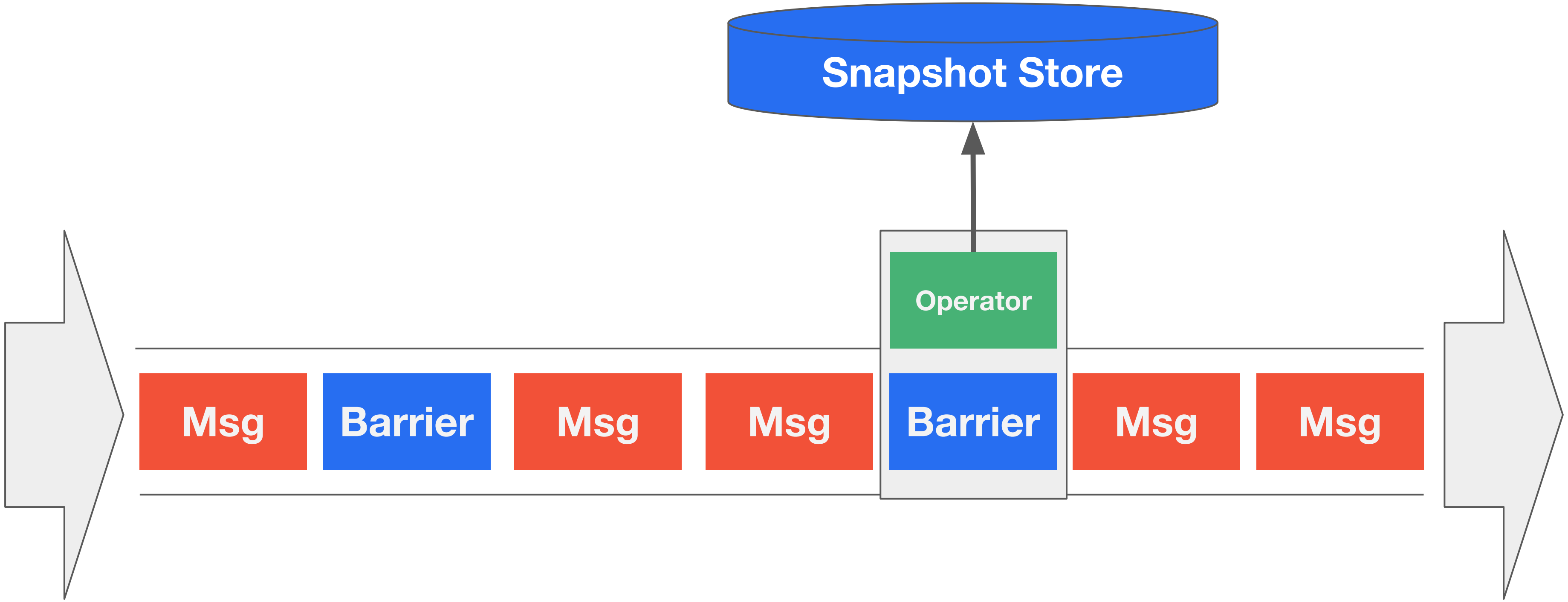
Fault tolerant


Snapshot Store
Task Manager
Task Manager
Flink Program
Optimizer
Graph Builder
Client
Job Manager

Lightweight Asynchronous Snapshots for Distributed Dataflows

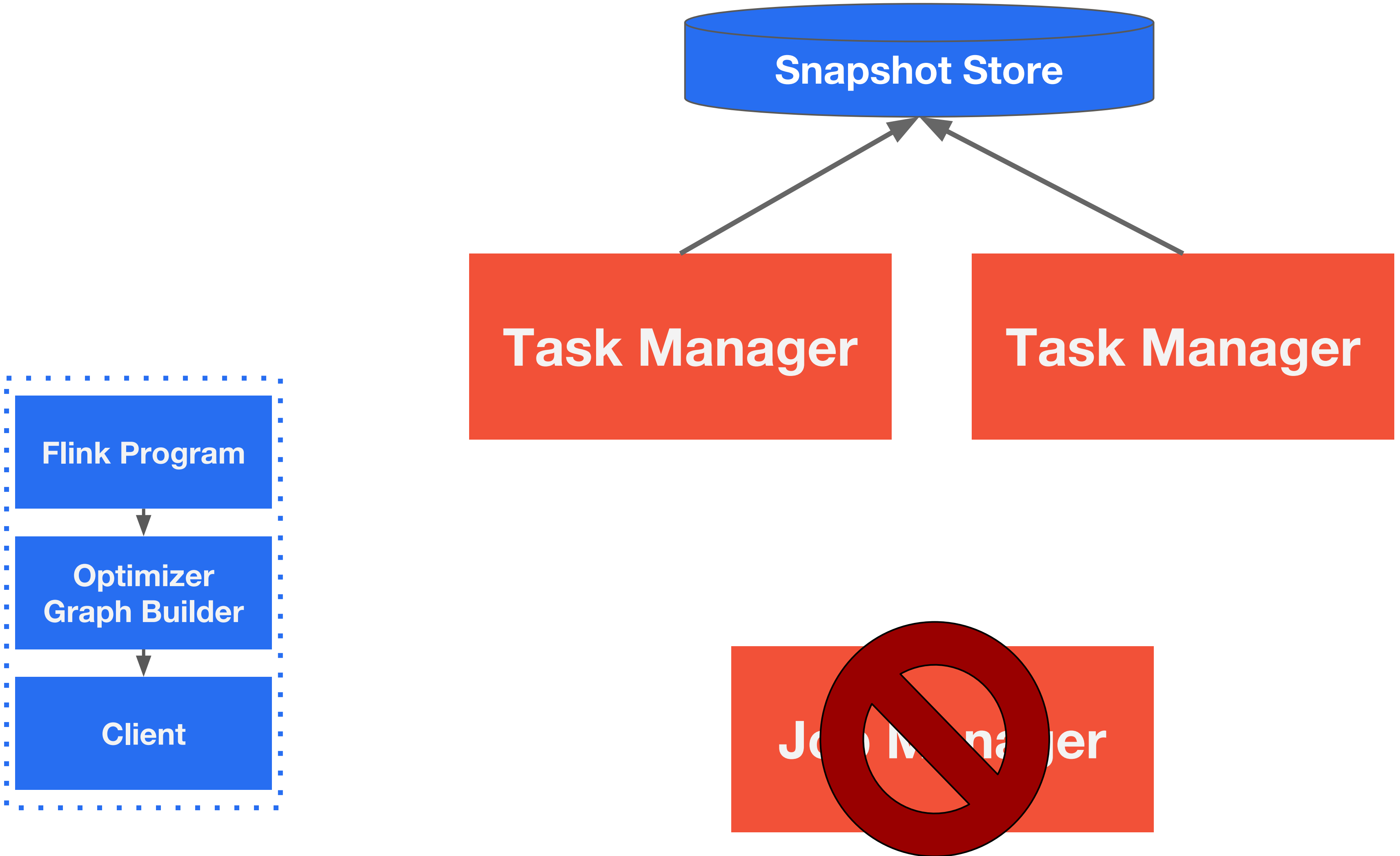
Paris Carbone,
Gyula Fóra,
Stephan Ewen
Seif Haridi
Kostas Tzoumas

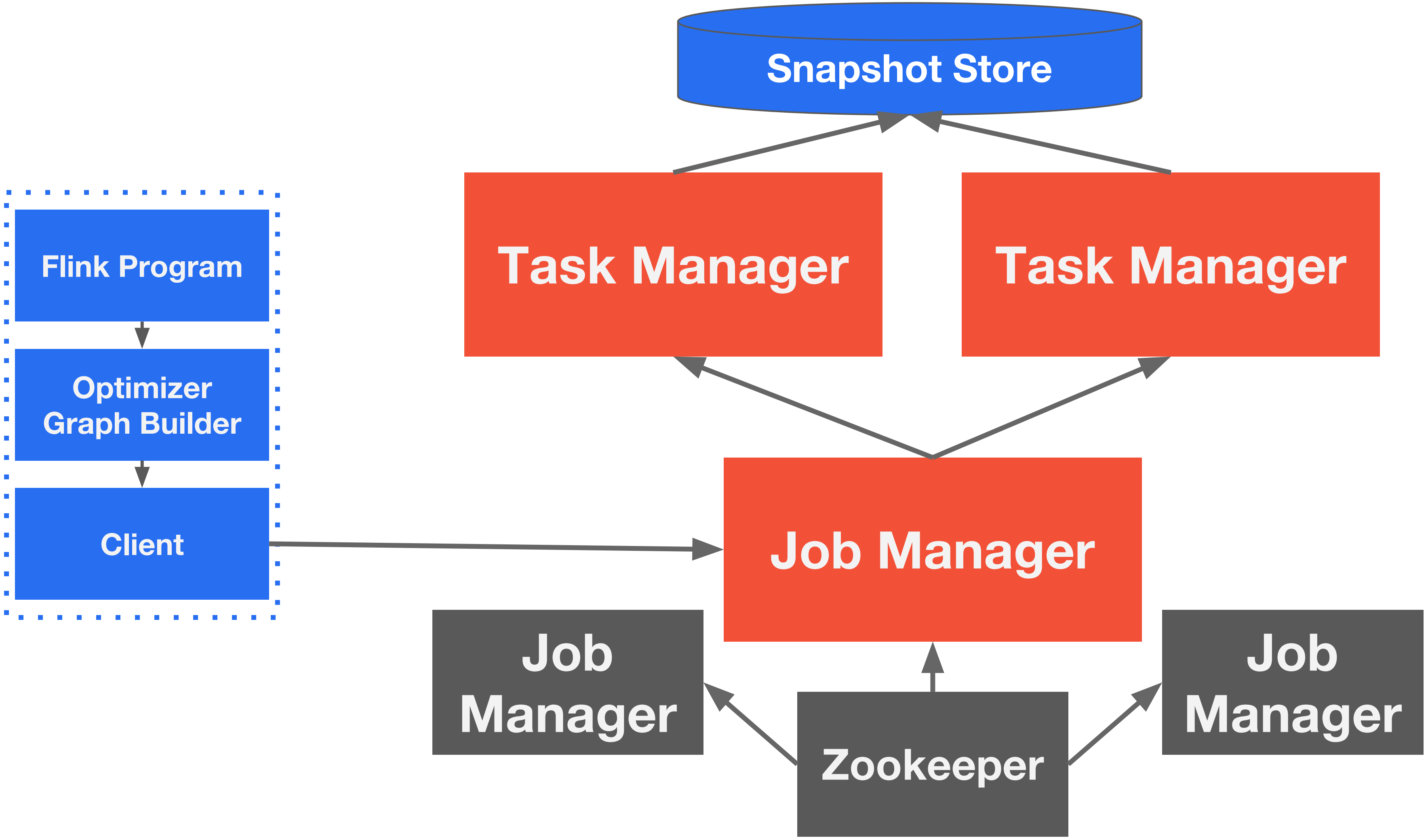

Operator
Barrier
Msg
Msg
Barrier
Msg
Msg
Barrier

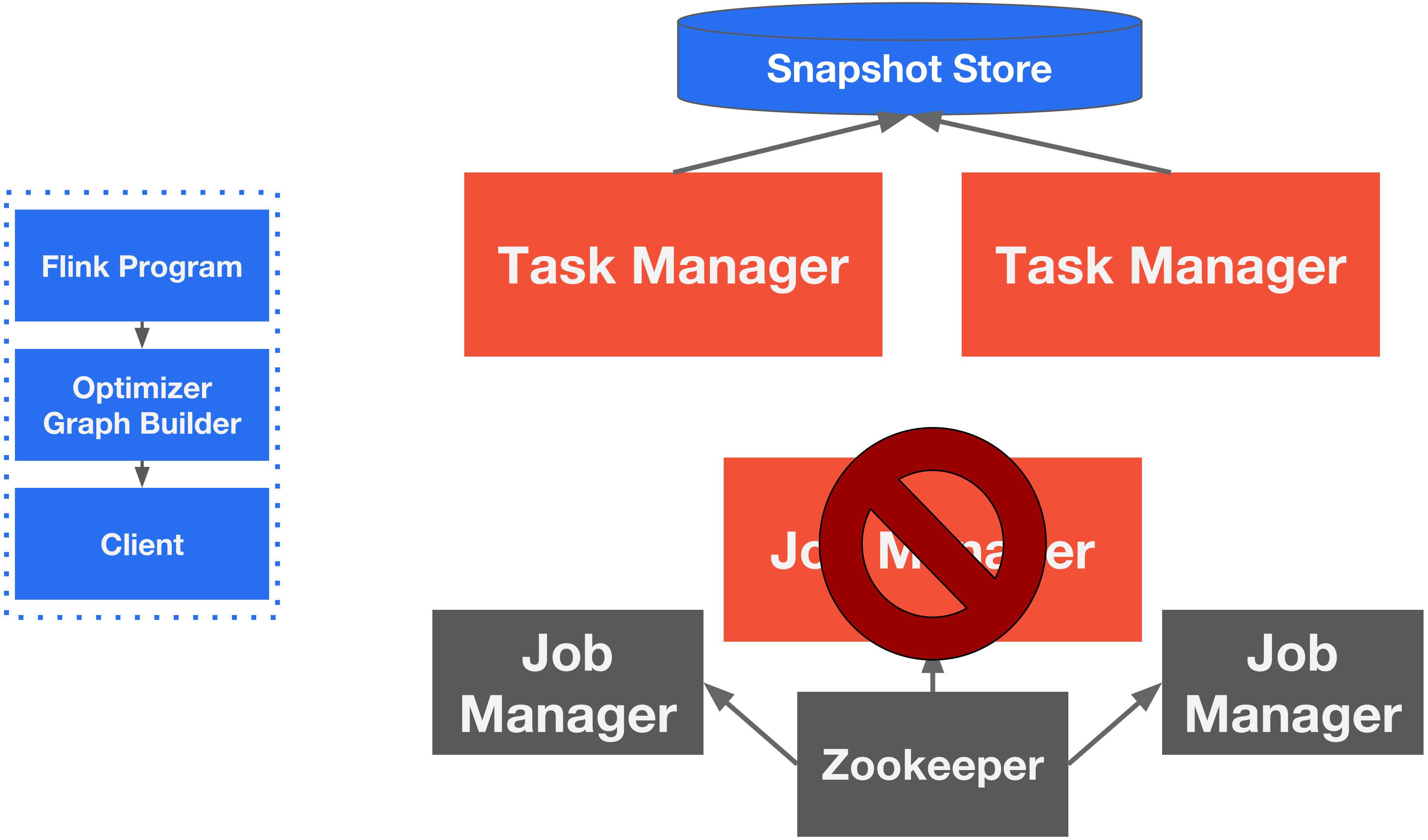

Snapshot Store
Operator
Msg
Barrier
Msg
Msg
Barrier
Msg
Msg

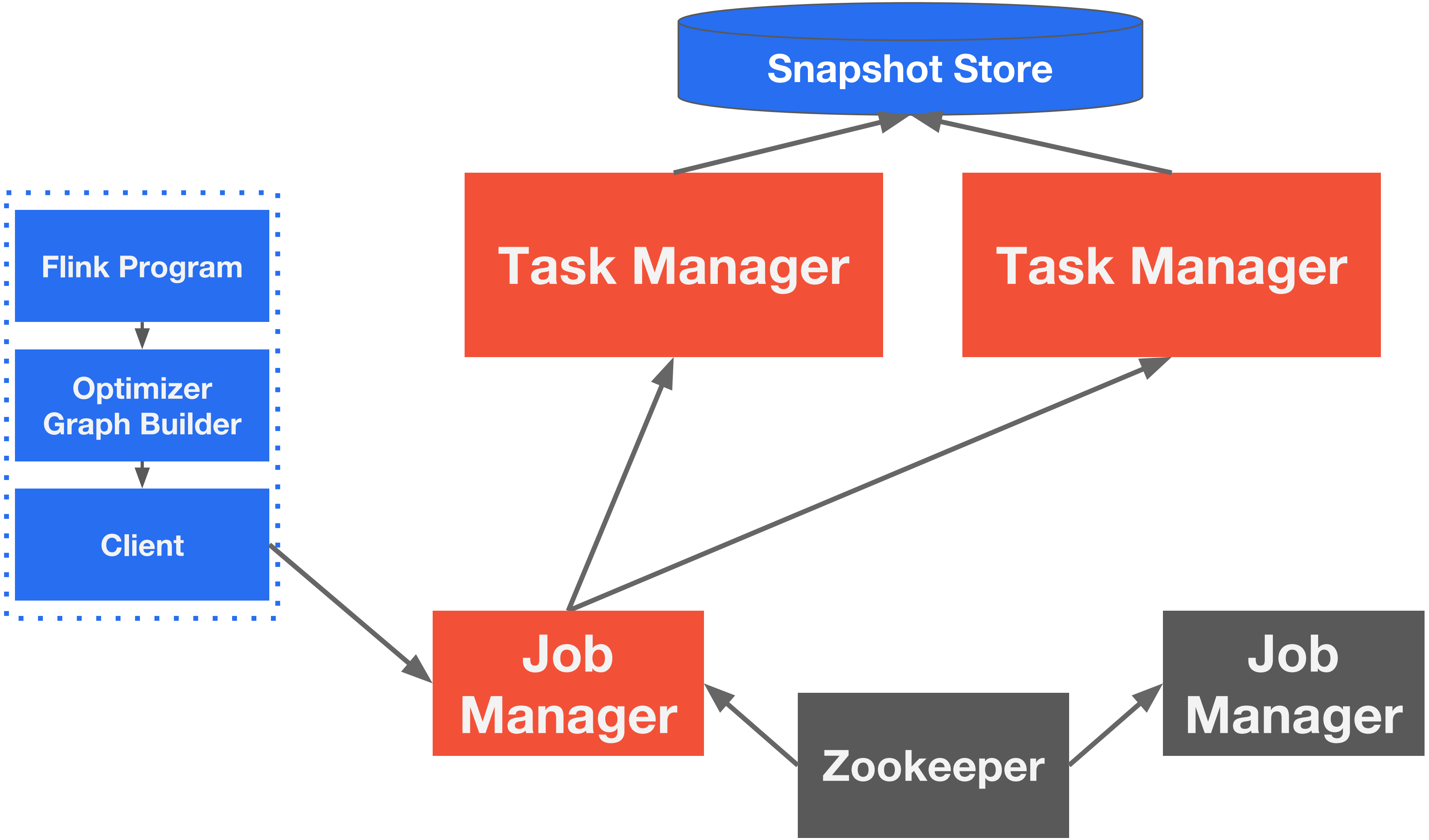
Exactly Once Processing

Can handle very large state

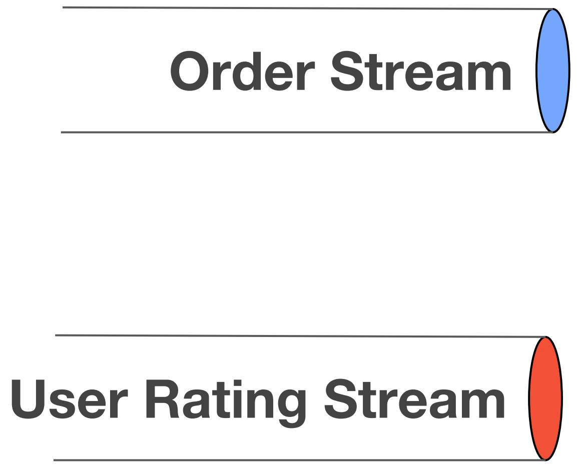

Snapshot Store
Task Manager
Task Manager
Flink Program
Optimizer
Graph Builder
Client
Job Manager

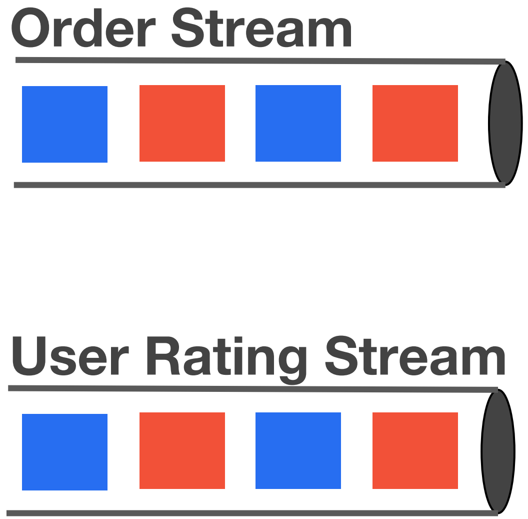

Snapshot Store
Task Manager
Task Manager
Flink Program
Optimizer Graph Builder
Client
Job Manager
Job Manager
Job Manager
Zookeeper

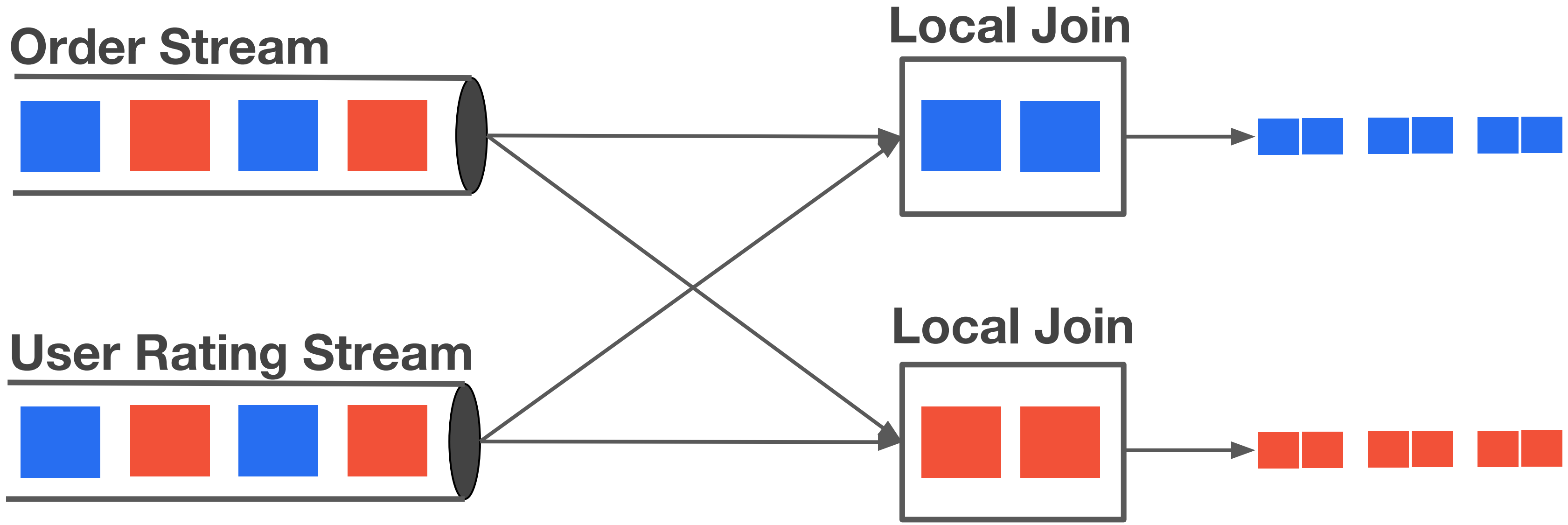

Snapshot Store
Flink Program
Optimizer Graph Builder
Client
Task Manager
Task Manager
Job Manager
Job Manager
Job Manager
Zookeeper

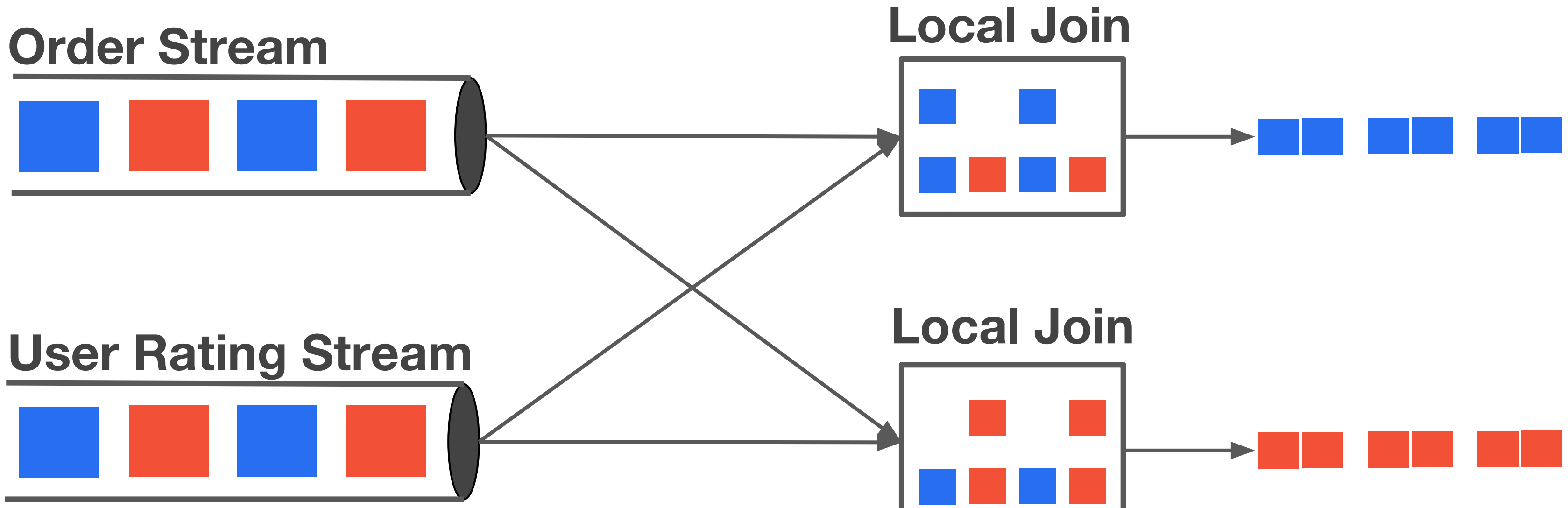

Snapshot Store
Flink Program
Optimizer Graph Builder
Client
Task Manager
Task Manager
Job Manager
Zookeeper
Job Manager

Joining Streams


Order Stream
User Rating Stream


Order Stream
User Rating Stream


Order Stream
User Rating Stream
Local Join
Local Join

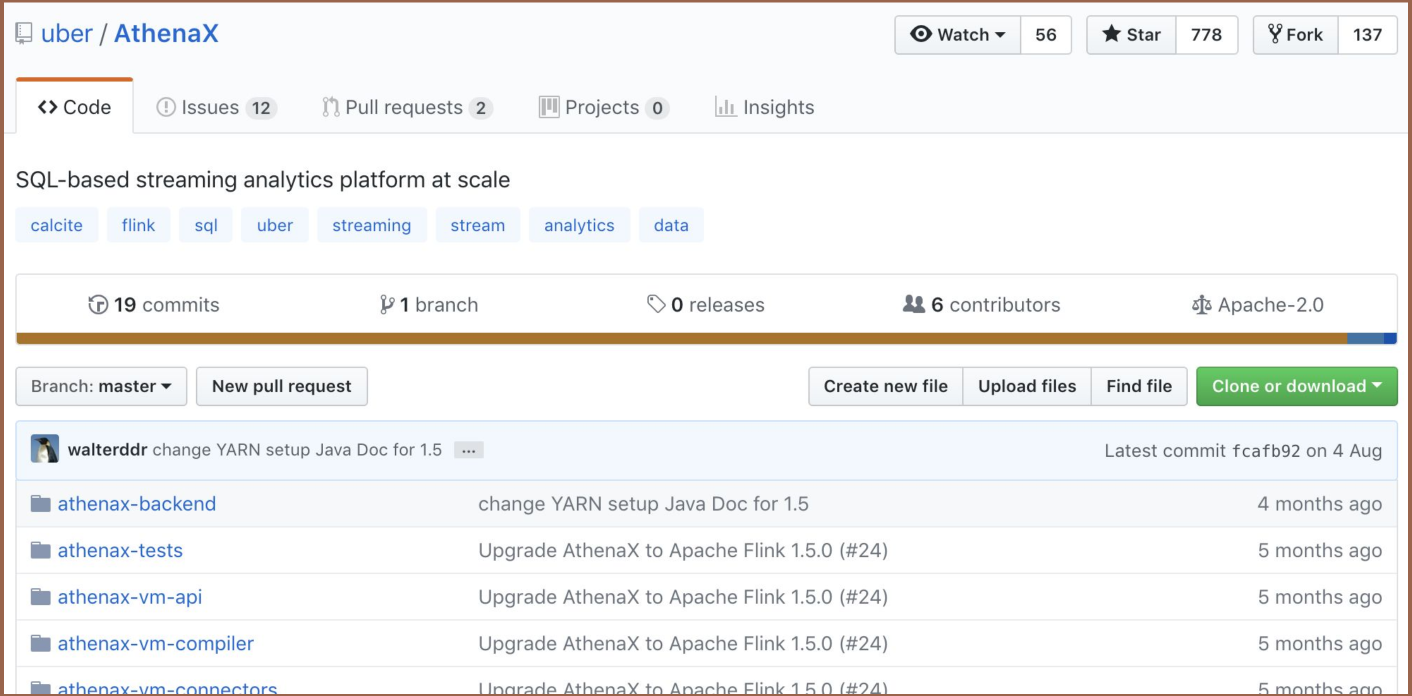

Order Stream
User Rating Stream
Local Join
Local Join

Apache Flink

- Can join streams
- Fault tolerant
- Exactly Once Processing
- Combines stream and batch processing

… but it requires Java/Scala code

Scalable, efficient and robust


uber / AthenaX

Watch 56 | Star 778 | Fork 137

Code | Issues 12 | Pull requests 2 | Projects 0 | Insights

SQL-based streaming analytics platform at scale

calcite flink sql uber streaming stream analytics data

19 commits | 1 branch | 0 releases | 6 contributors | Apache-2.0

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

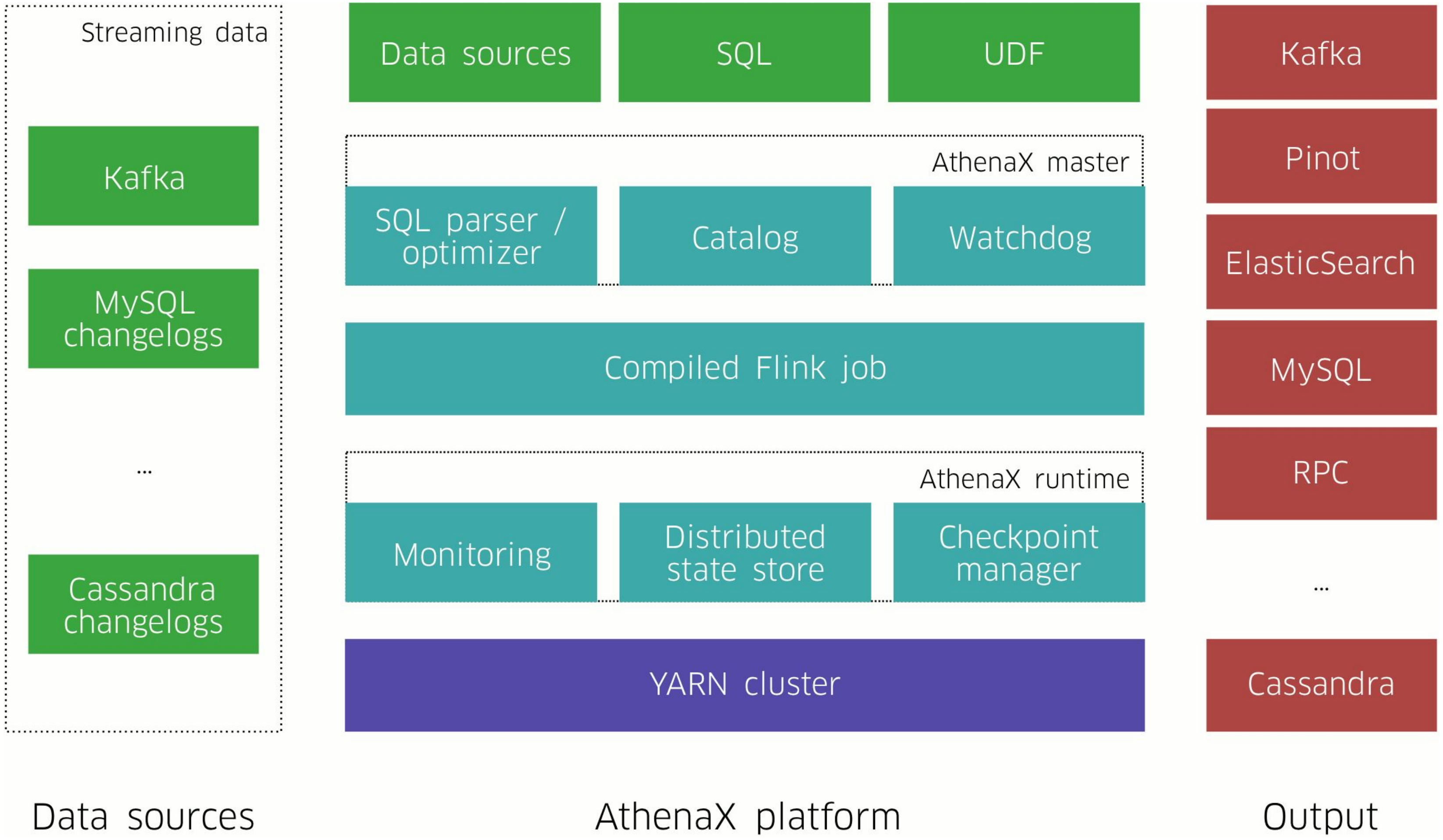
walterddr change YARN setup Java Doc for 1.5 … Latest commit fcafb92 on 4 Aug

athenax-backend	change YARN setup Java Doc for 1.5	4 months ago
athenax-tests	Upgrade AthenaX to Apache Flink 1.5.0 (#24)	5 months ago
athenax-vm-api	Upgrade AthenaX to Apache Flink 1.5.0 (#24)	5 months ago
athenax-vm-compiler	Upgrade AthenaX to Apache Flink 1.5.0 (#24)	5 months ago
athenax-vm-connectors	Upgrade AthenaX to Apache Flink 1.5.0 (#24)	5 months ago

github.com/uber/AthenaX

SQL → what data to analyze

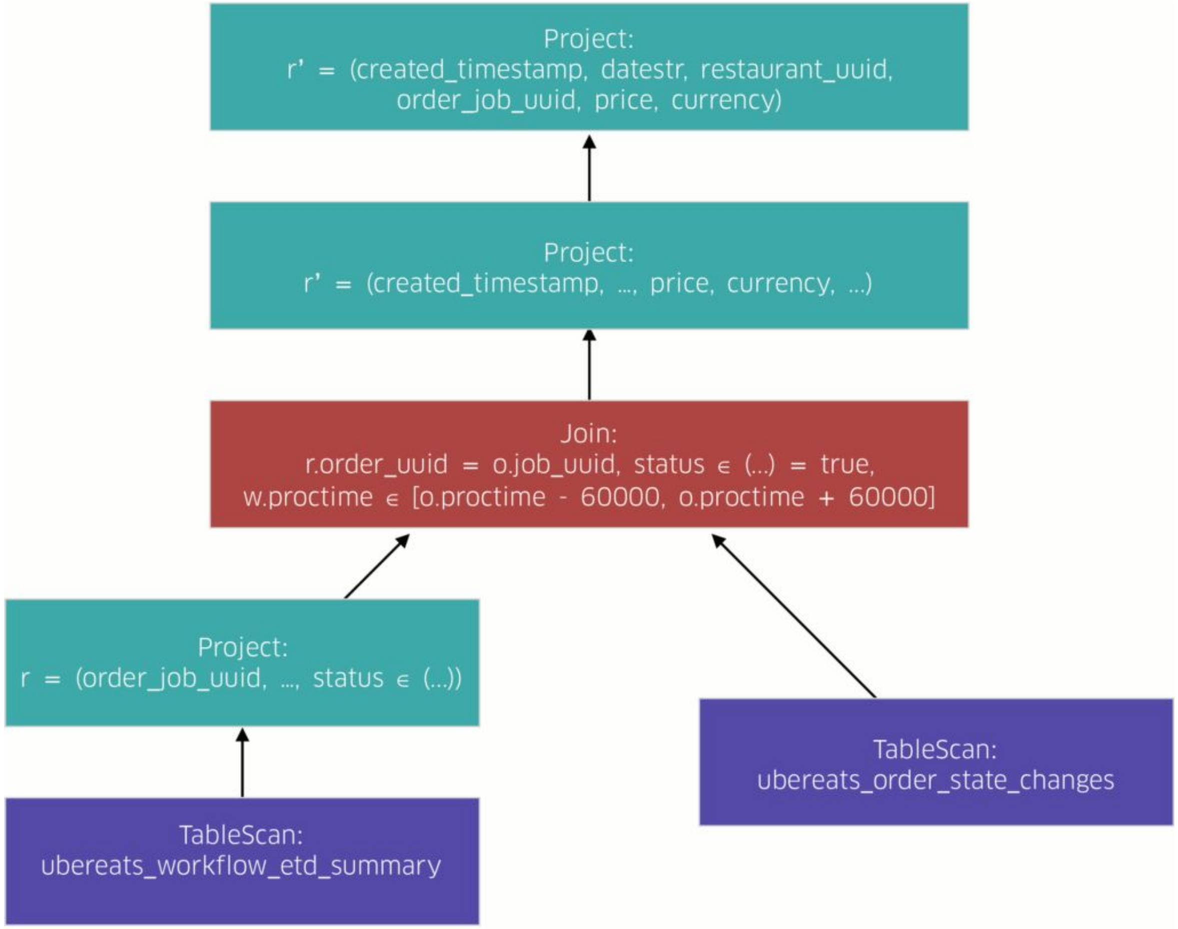
Flink → how to analyze it


Streaming data
Kafka
MySQL changelogs
...
Cassandra changelogs
Data sources
SQL
UDF
AthenaX master
SQL parser / optimizer
Catalog
Watchdog
Compiled Flink job
AthenaX runtime
Monitoring
Distributed state store
Checkpoint manager
YARN cluster
Kafka
Pinot
ElasticSearch
MySQL
RPC
...
Cassandra
Data sources
AthenaX platform
Output

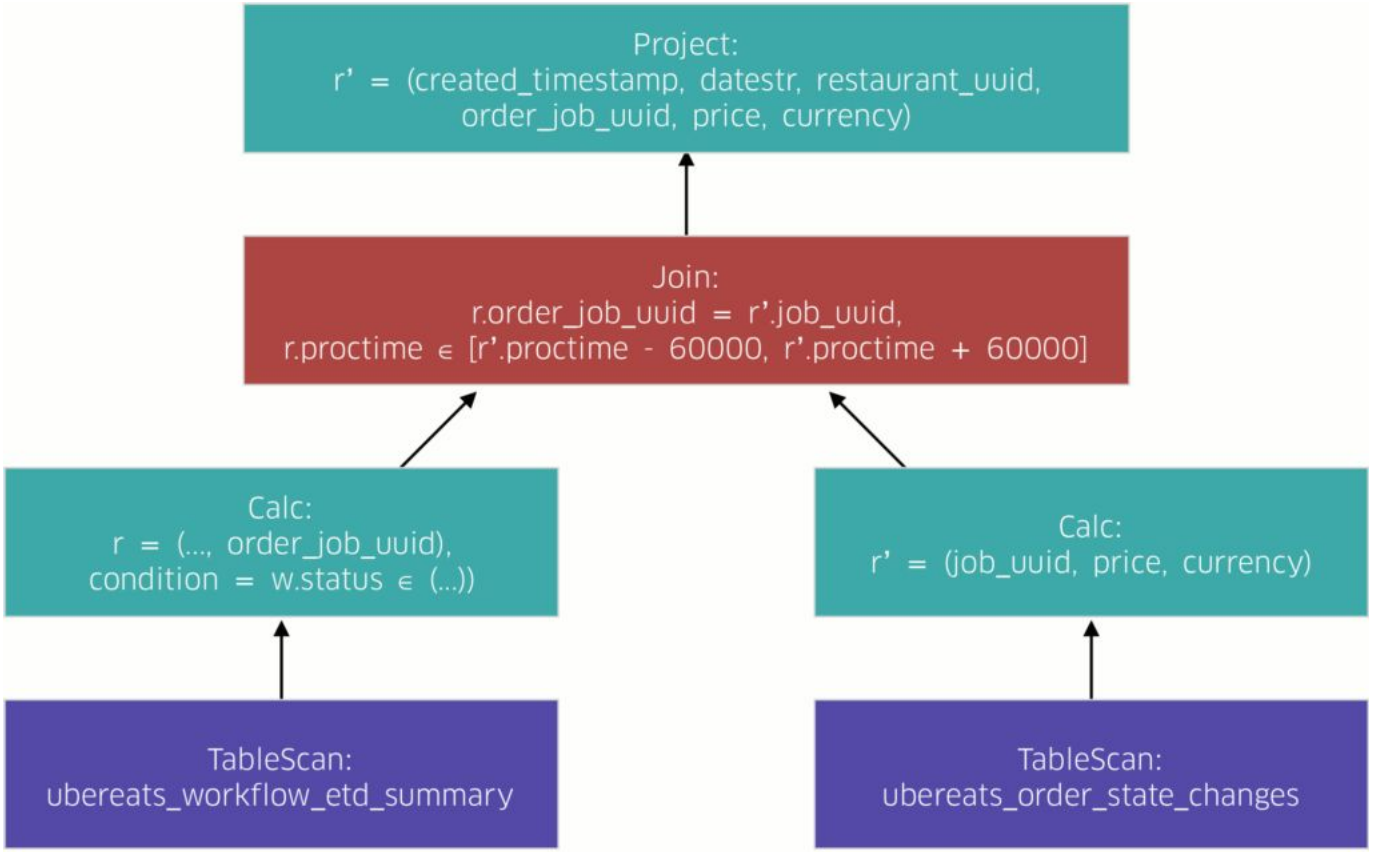
```sql
SELECT
   HOP_START(rowtime, INTERVAL ‘1’ MINUTE, INTERVAL ’15’ MINUTE)
AS window_start,
   restaurant_uuid,
   COUNT(*) AS total_order
FROM ubereats_workflow
WHERE state = ’CREATED’
GROUP BY
   restaurant_uuid,
   HOP(rowtime, INTERVAL ‘1’ MINUTE, INTERVAL ’15’ MINUTE)
```

```
CREATE FUNCTION AirportCode AS …;
SELECT
   AirportCode(location.lng,location.lat) AS airport
   driver_id AS driver_id,
   …
FROM
   event_user_driver_app
WHERE
 NAME =‘trip_start’
```

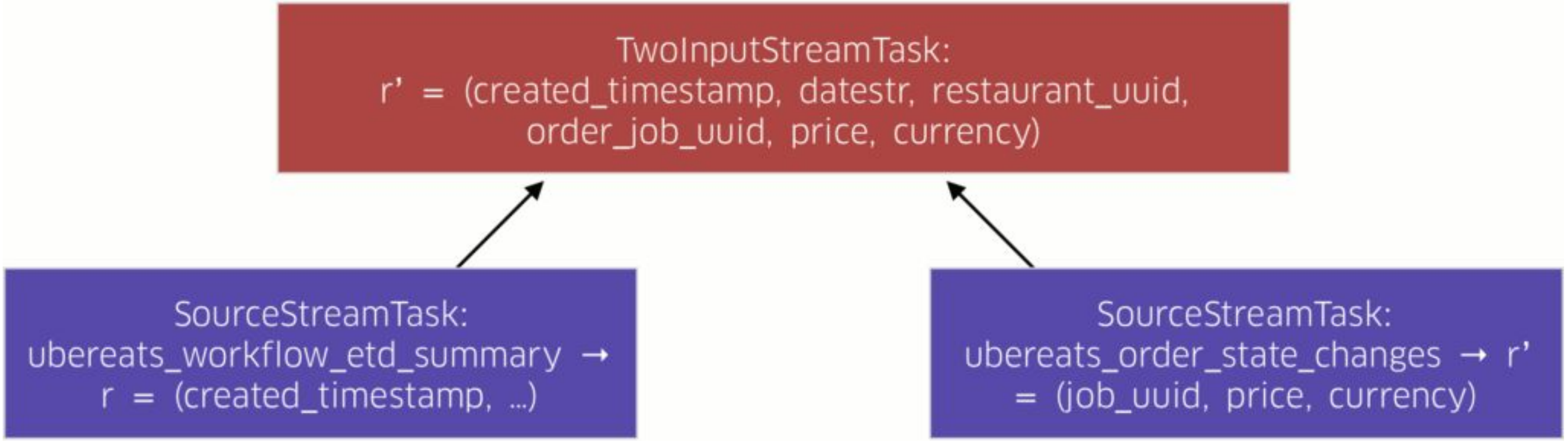
```
SELECT
   w.created_timestamp,
   w.datestr,
   w.restaurant_uuid,
   w.order_job_uuid,
   o.price,
   o.currency,
FROM
   ubereats_workflow_etd_summary w
JOIN
   ubereats_order_state_changes o
ON
   o.job_uuid = w.order_job_uuid
WHERE
   w.status IN ('CANCELED_BY_EATER', 'UNFULFILLED')
AND
   w.proctime
 BETWEEN
   o.proctime – INTERVAL '60' SECOND
 AND
   o.proctime + INTERVAL '60' SECOND
```


Project:
r' = (created_timestamp, datestr, restaurant_uuid, order_job_uuid, price, currency)
Project:
r' = (created_timestamp, ..., price, currency, ...)
Join:
r.order_uuid = o.job_uuid, status ∈ (...) = true, w.proctime ∈ [o.proctime - 60000, o.proctime + 60000]
Project:
r = (order_job_uuid, ..., status ∈ (...))
TableScan:
ubereats_workflow_etd_summary
TableScan:
ubereats_order_state_changes

(a) original logical plan


Project:
r’ = (created_timestamp, datestr, restaurant_uuid, order_job_uuid, price, currency)
Join:
r.order_job_uuid = r’.job_uuid,
r.proctime ∈ [r’.proctime - 60000, r’.proctime + 60000]
Calc:
r = (..., order_job_uuid),
condition = w.status ∈ (...))
Calc:
r’ = (job_uuid, price, currency)
TableScan:
ubereats_workflow_etd_summary
TableScan:
ubereats_order_state_changes

(b) optimized logical plan


TwoInputStreamTask:
r’ = (created_timestamp, datestr, restaurant_uuid, order_job_uuid, price, currency)
SourceStreamTask:
ubereats_workflow_etd_summary → r = (created_timestamp, ...)
SourceStreamTask:
ubereats_order_state_changes → r’ = (job_uuid, price, currency)

(c) compiled data flow program

Resource estimation and auto scaling

Monitoring and automatic failure recovery

Introducing AthenaX, Uber Engineering's Open Source Streaming Analytics Platform

By Haohui Mai, Bill Liu, & Naveen Cherukuri

October 9, 2017

eng.uber.com/athenax

Thanks!

Nikolay Stoitsev @ Uber

Uber