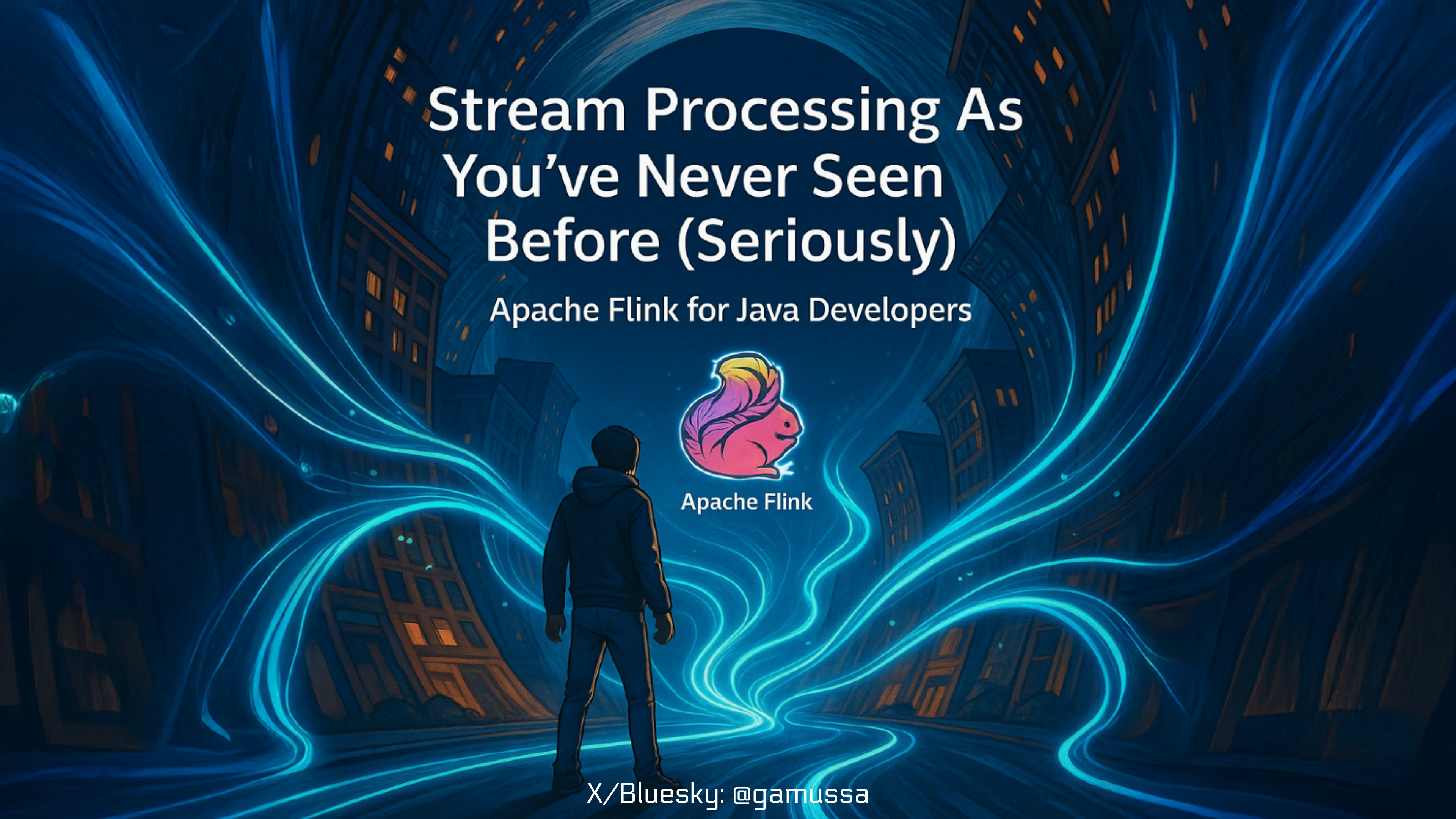
Stream Processing As You've Never Seen Before (Seriously)

Apache Flink for Java Developers



Apache Flink

X/Bluesky: @gamussa

Viktor

GAMOV

Principal Developer Advocate | Confluent
co-author of Manning's Kafka in Action

X/Bluesky: @gamussa

Slides and Video

https://speaking.gamov.io/

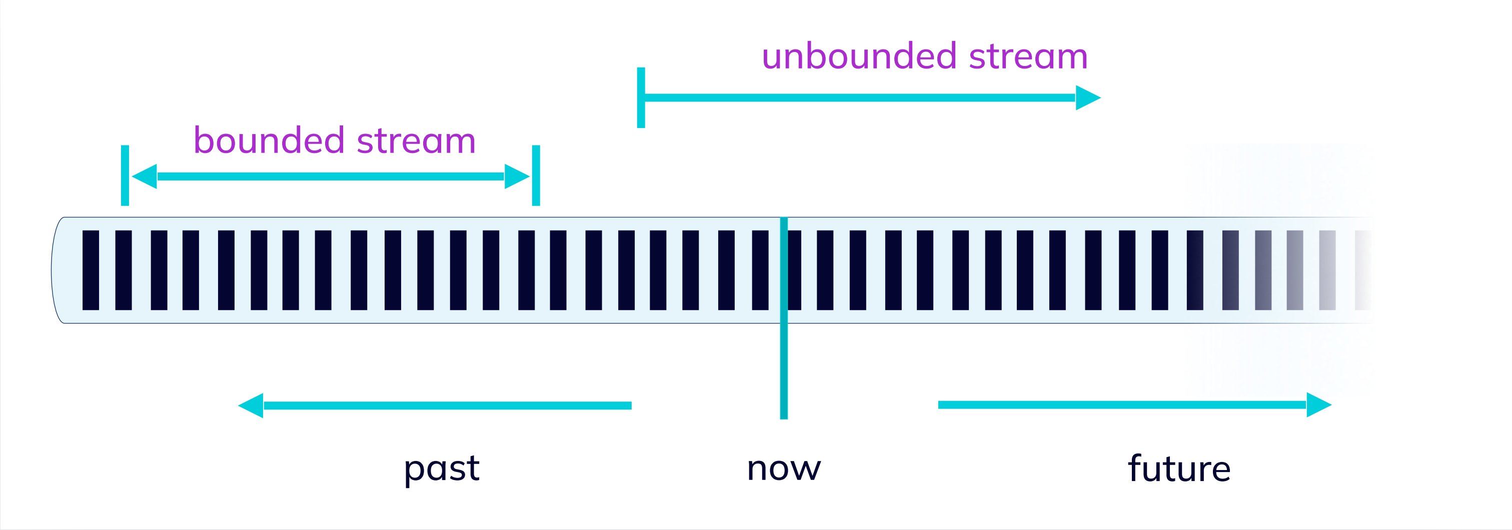
What is Apache Flink?

What is Flink?

Apache Flink is a **framework** and distributed processing engine for **stateful** computations over **unbounded** and **bounded** data streams.

Streaming

CONFLUENT Developer

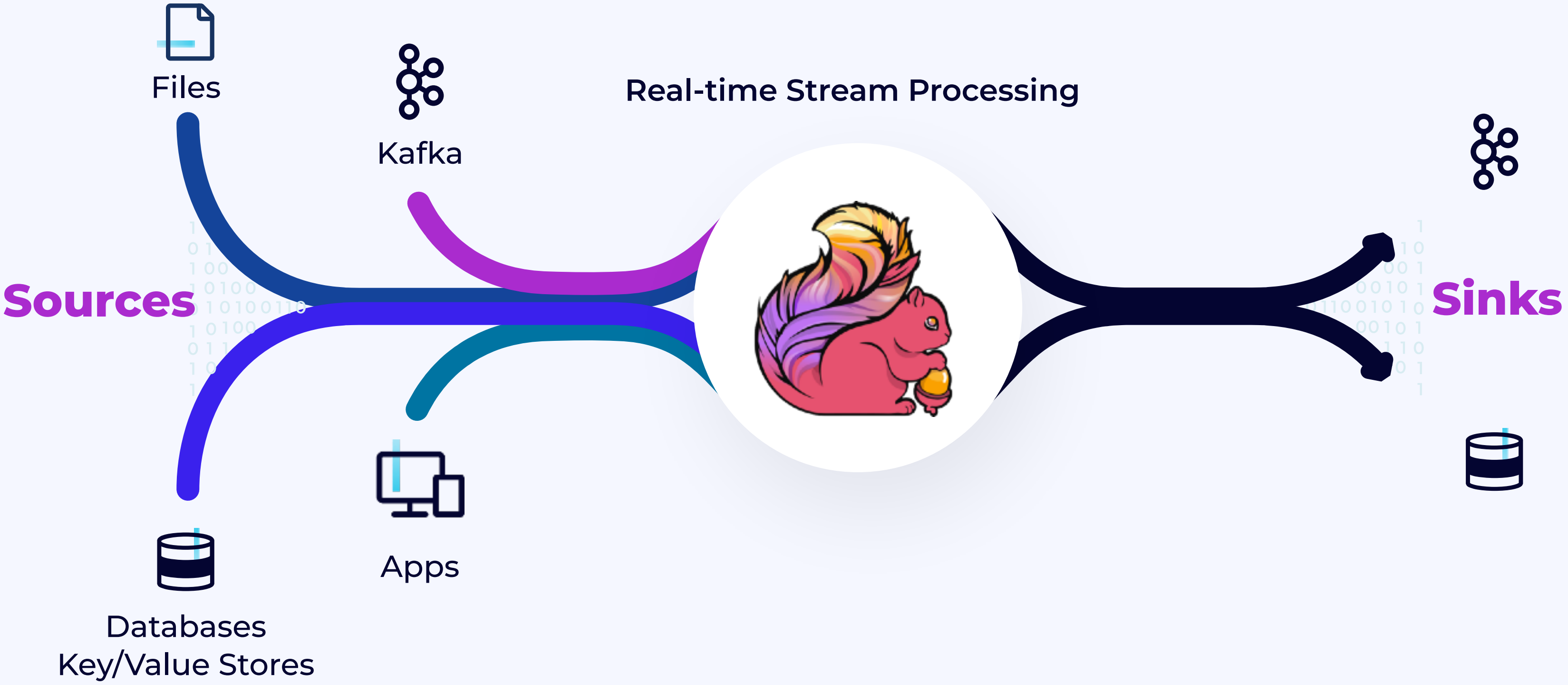


- A stream is a **sequence** of events
- Business data is always a stream: **bounded** or **unbounded**
- For Flink, batch processing is just a **special case** in the runtime

Key Features of Flink

Stream processing with Flink

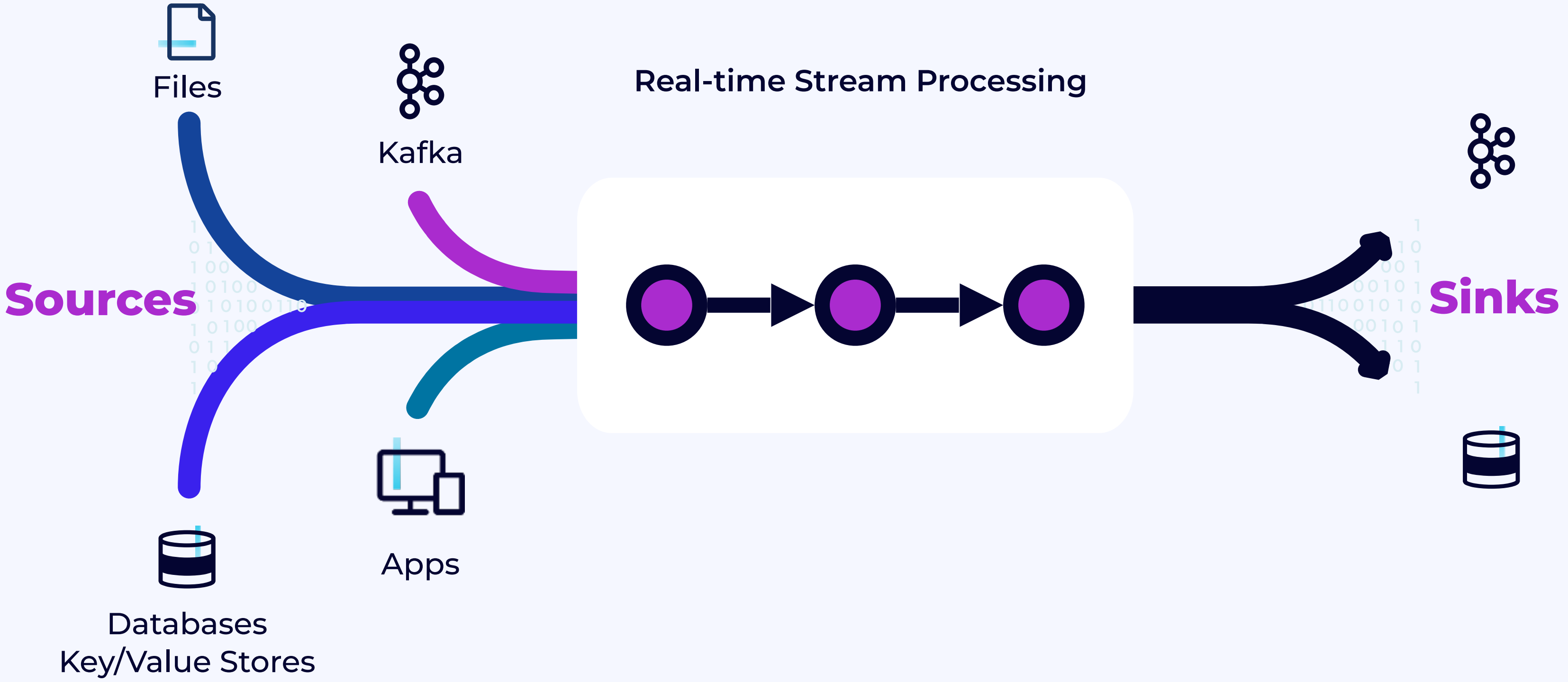
CONFLUENT Developer



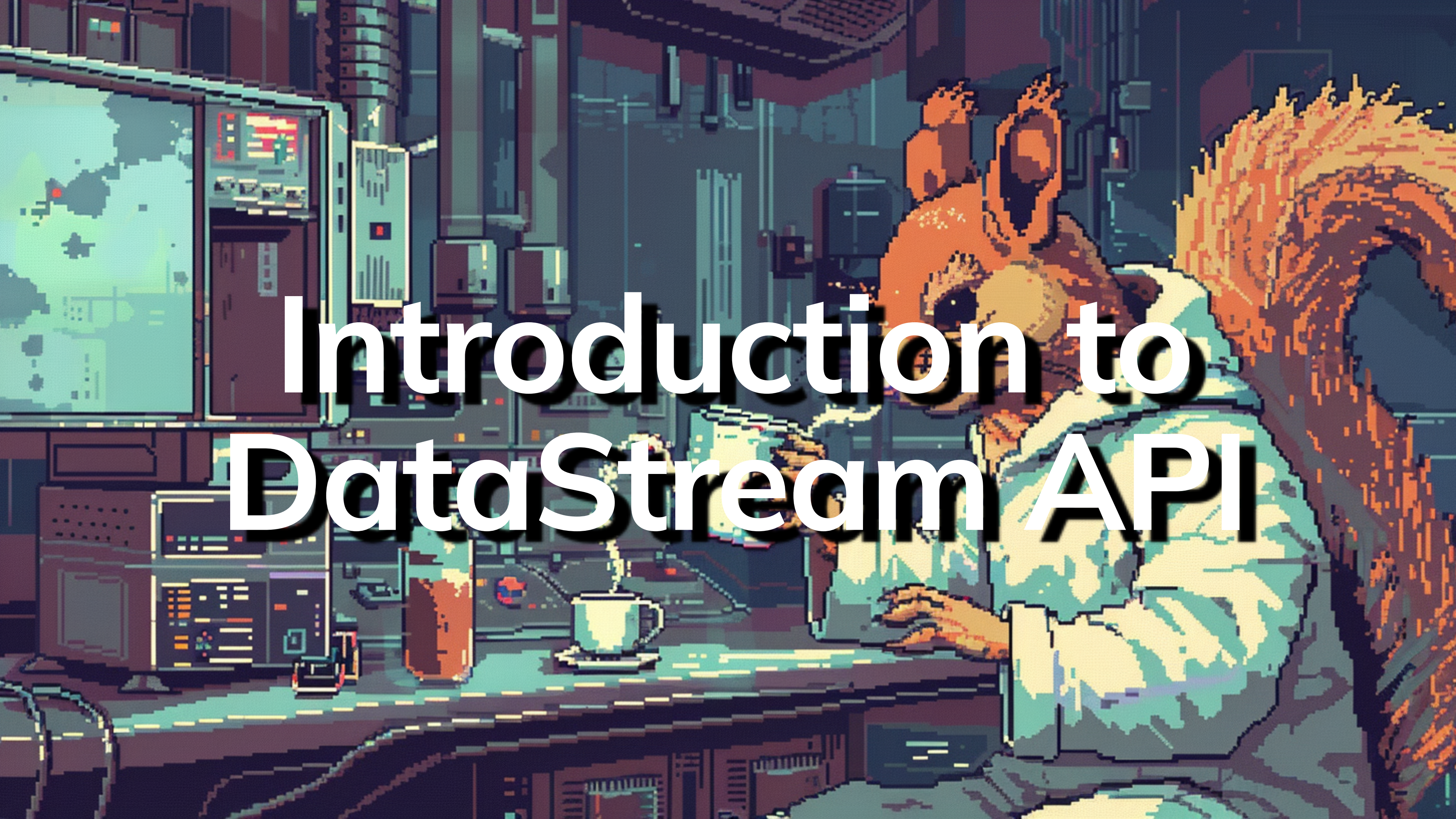
X/Bluesky: @gamussa

Stream processing with Flink

CONFLUENT Developer



Introduction to DataStream API



```
// Set up the execution environment
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

// Create a DataStream from some elements
DataStream<String> inputStream = env.fromData("apple", "banana", "cherry", "date", "elderberry");

// Perform a transformation
DataStream<Tuple2<String, Integer>> resultStream = inputStream
    .map(value -> new Tuple2<>(value, value.length()))
    .returns(Types.TUPLE(Types.STRING, Types.INT));

// Print the results to the console
resultStream.print();

// Execute the Flink job
env.execute("Simple Flink Job");
```

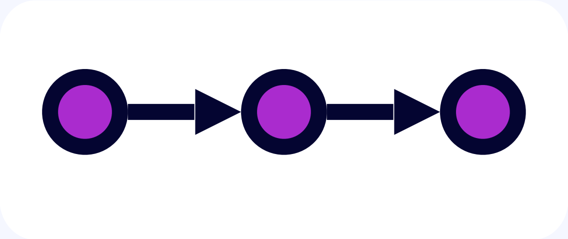
```java
// Set up the execution environment
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

// Create a DataStream from some elements
DataStream<String> inputStream = env.fromData("apple", "banana", "cherry", "date", "elderberry");

// Perform a transformation
DataStream<Tuple2<String, Integer>> resultStream = inputStream
    .map(value -> new Tuple2<>(value, value.length()))
    .returns(Types.TUPLE(Types.STRING, Types.INT));

// Print the results to the console
resultStream.print();

// Execute the Flink job
env.execute("Simple Flink Job");
```

```java
// Set up the execution environment
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

// Create a DataStream from some elements
DataStream<String> inputStream = env.fromData("apple", "banana", "cherry", "date", "elderberry");

// Perform a transformation
DataStream<Tuple2<String, Integer>> resultStream = inputStream
    .map(value -> new Tuple2<>(value, value.length()))
    .returns(Types.TUPLE(Types.STRING, Types.INT));

// Print the results to the console
resultStream.print();

// Execute the Flink job
env.execute("Simple Flink Job");
```

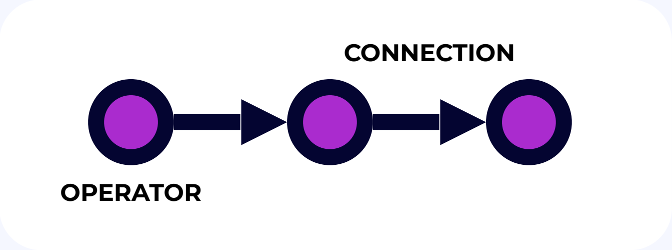
```
// Set up the execution environment
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();

// Create a DataStream from some elements
DataStream<String> inputStream = env.fromData("apple", "banana", "cherry", "date", "elderberry");

// Perform a transformation
DataStream<Tuple2<String, Integer>> resultStream = inputStream
    .map(value -> new Tuple2<>(value, value.length()))
    .returns(Types.TUPLE(Types.STRING, Types.INT));

// Print the results to the console
resultStream.print();

// Execute the Flink job
env.execute("Simple Flink Job");
```

The JobGraph (or topology)

CONFLUENT Developer

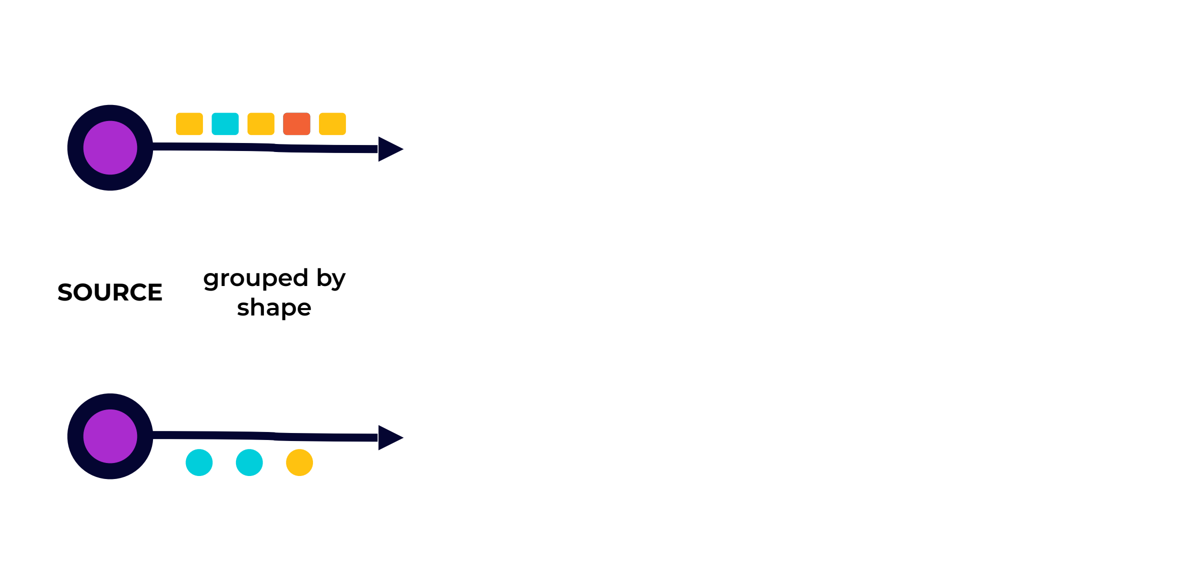


The JobGraph (or topology)



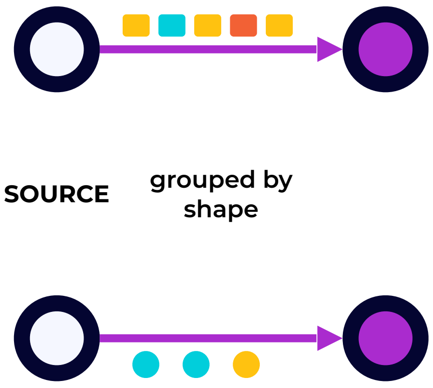
Stream processing

- Parallel
- Forward
- Repartition
- Rebalance



Stream processing

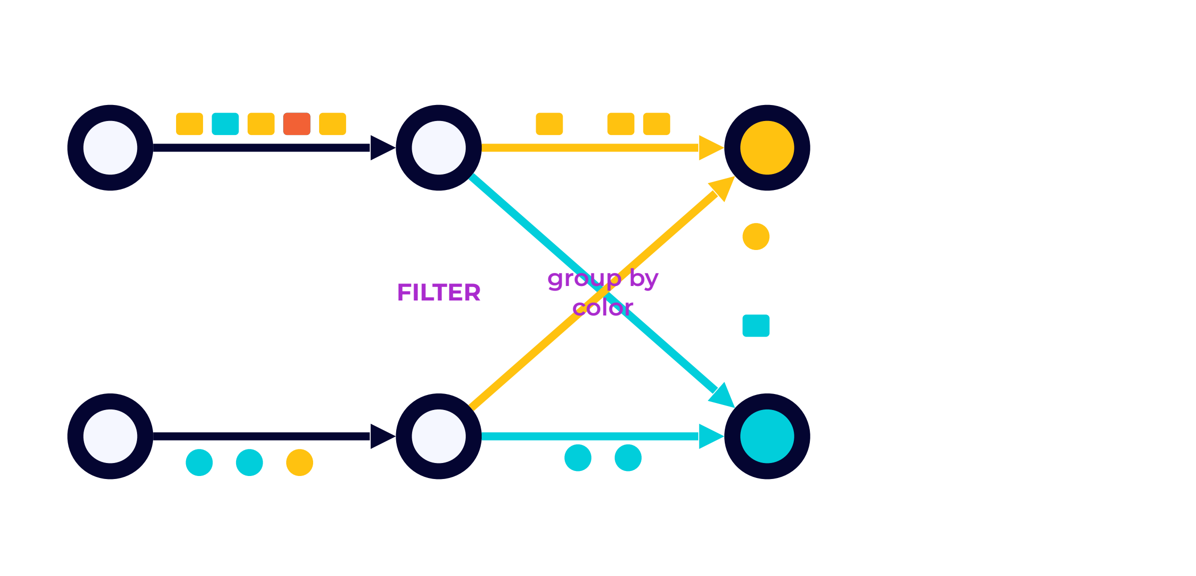
- Parallel
- Forward
- Repartition
- Rebalance



Stream processing

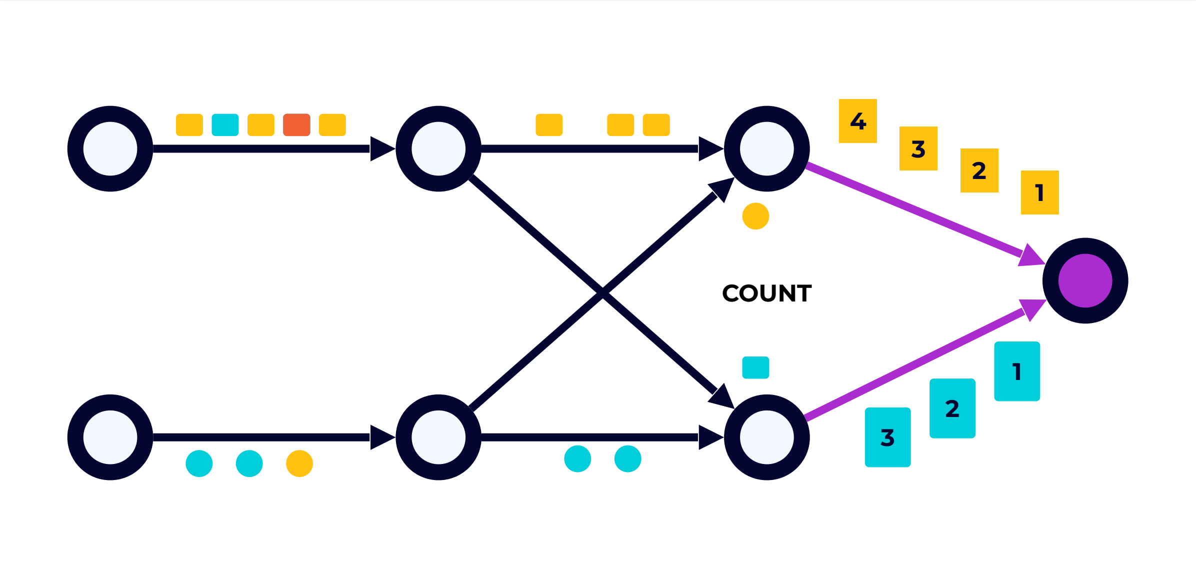
CONFLUENT Developer

- Parallel
- Forward
- Repartition
- Rebalance



Stream processing

- Parallel
- Forward
- Repartition
- Rebalance

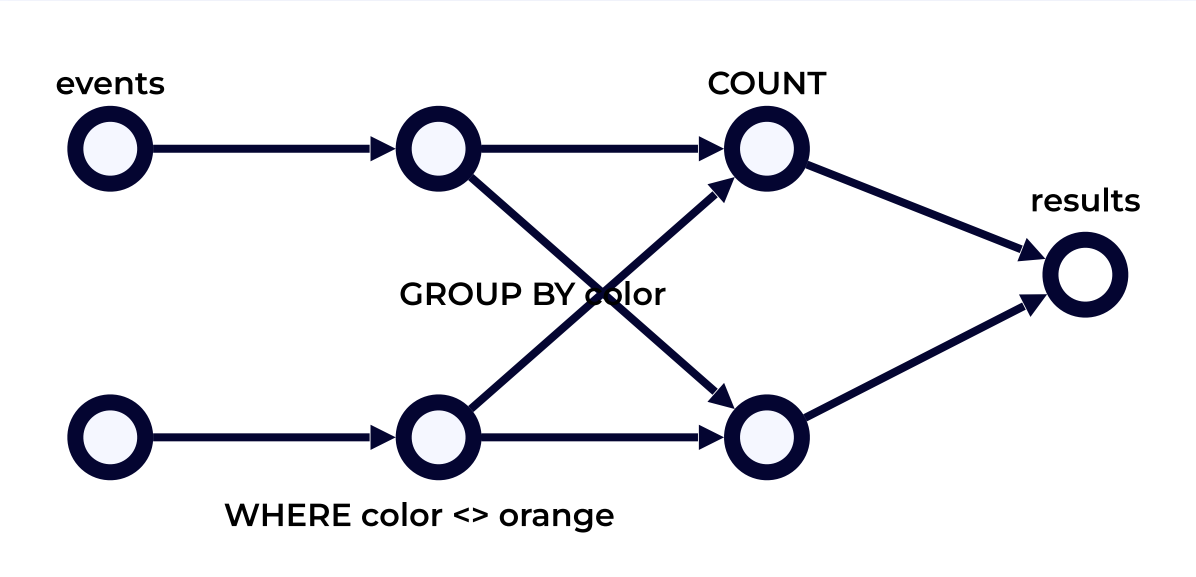


X/Bluesky: @gamussa

Stream processing with SQL

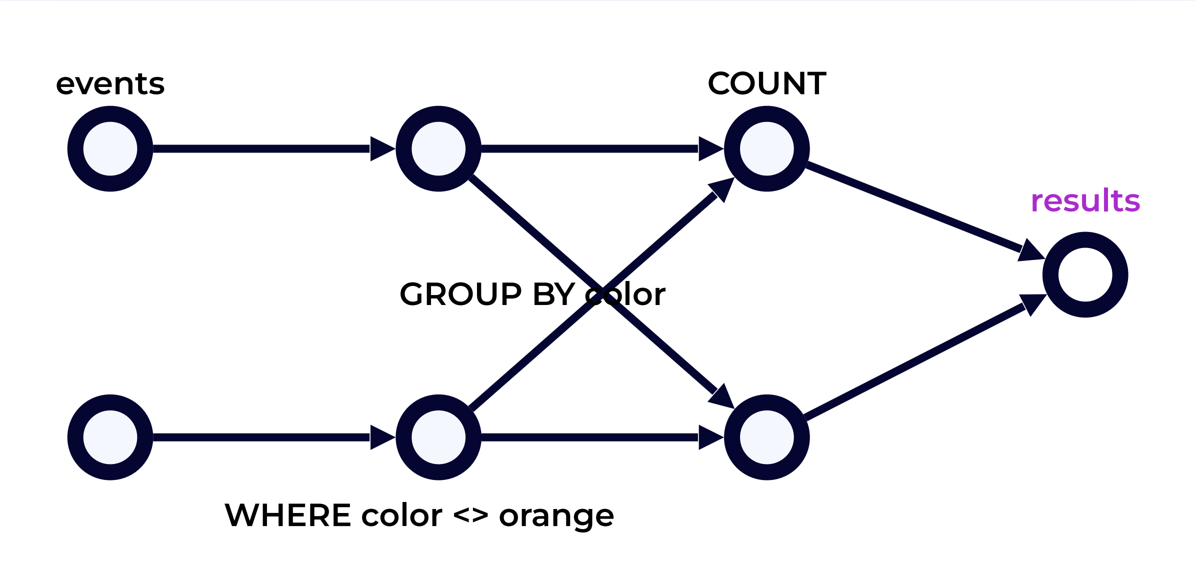
CONFLUENT Developer

```
INSERT INTO results
SELECT color, COUNT(*)
FROM events
WHERE color <> orange
GROUP BY color;
```



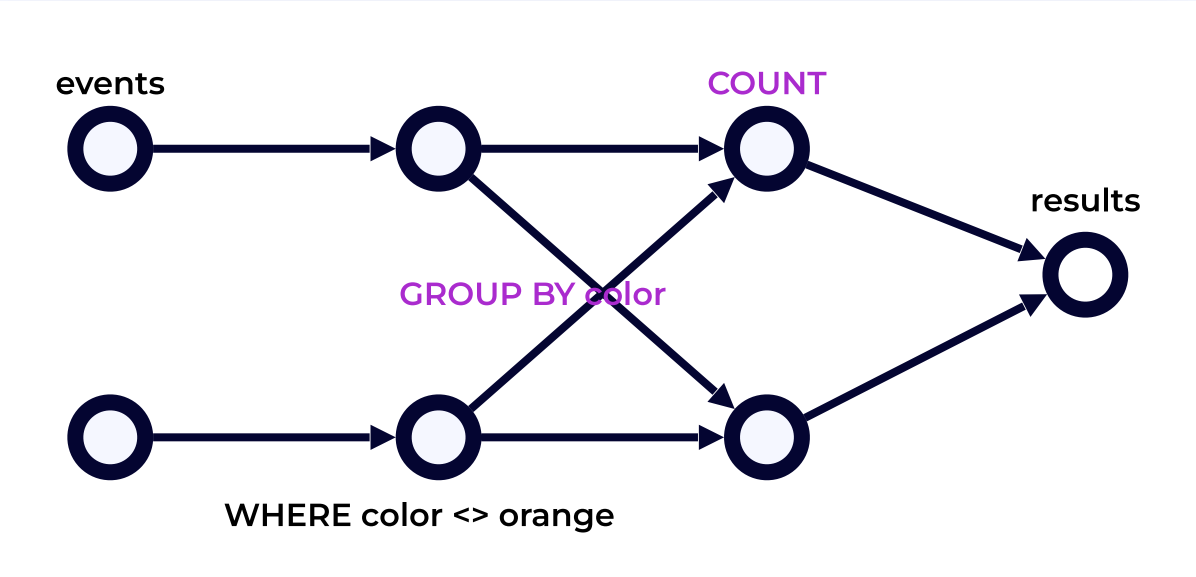
Stream processing with SQL

```
INSERT INTO results
SELECT color, COUNT(*)
FROM events
WHERE color <> orange
GROUP BY color;
```



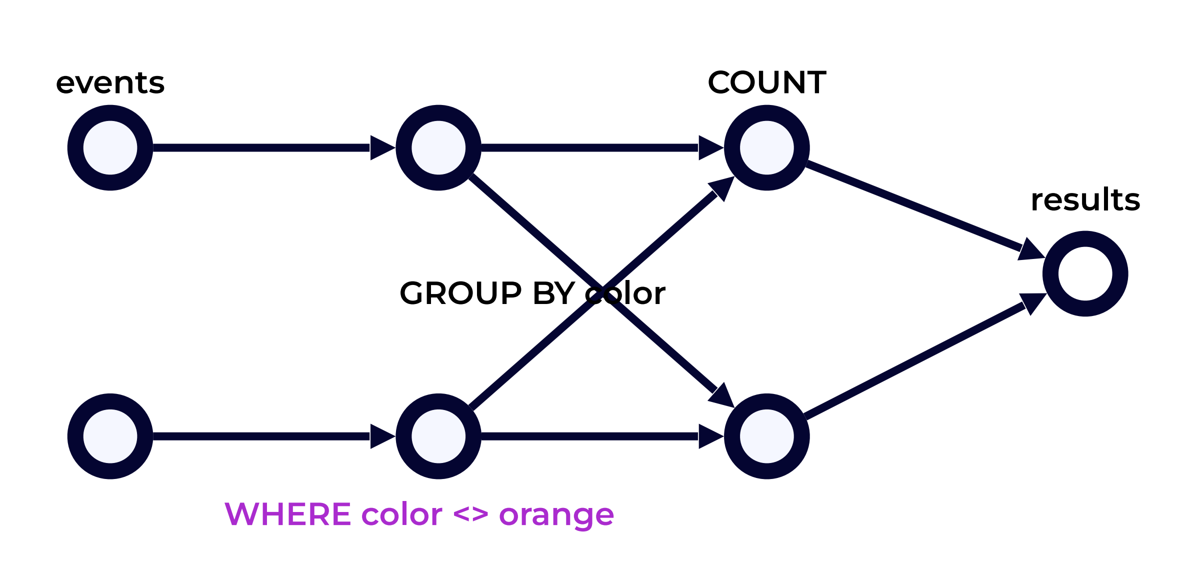
Stream processing with SQL

```
INSERT INTO results
SELECT color, COUNT(*)
FROM events
WHERE color <> orange
GROUP BY color;
```



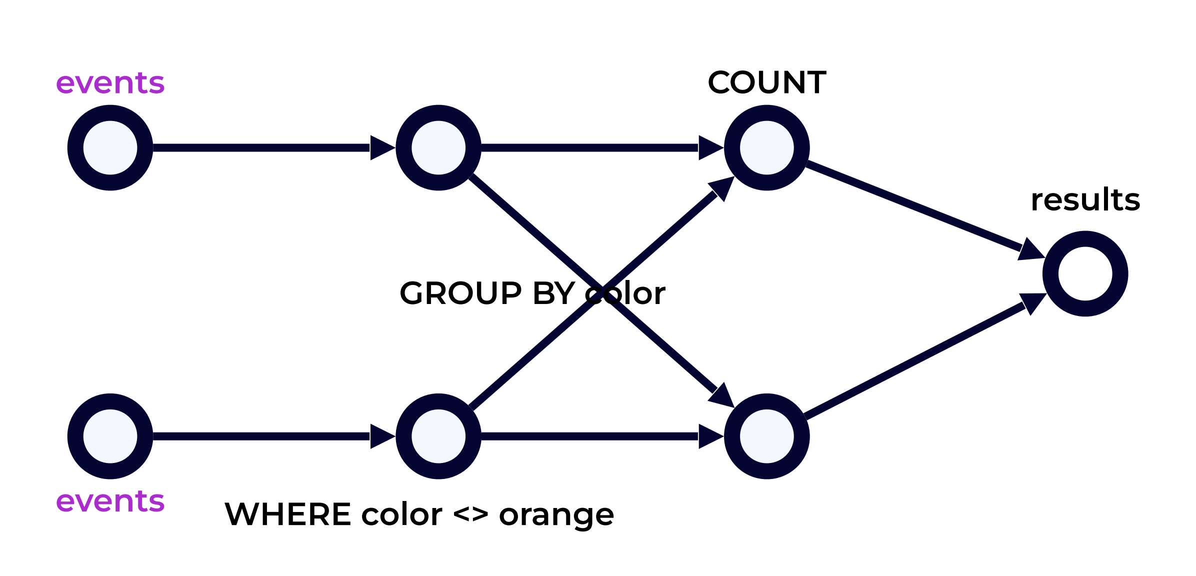
Stream processing with SQL

```
INSERT INTO results
SELECT color, COUNT(*)
FROM events
WHERE color <> orange
GROUP BY color;
```



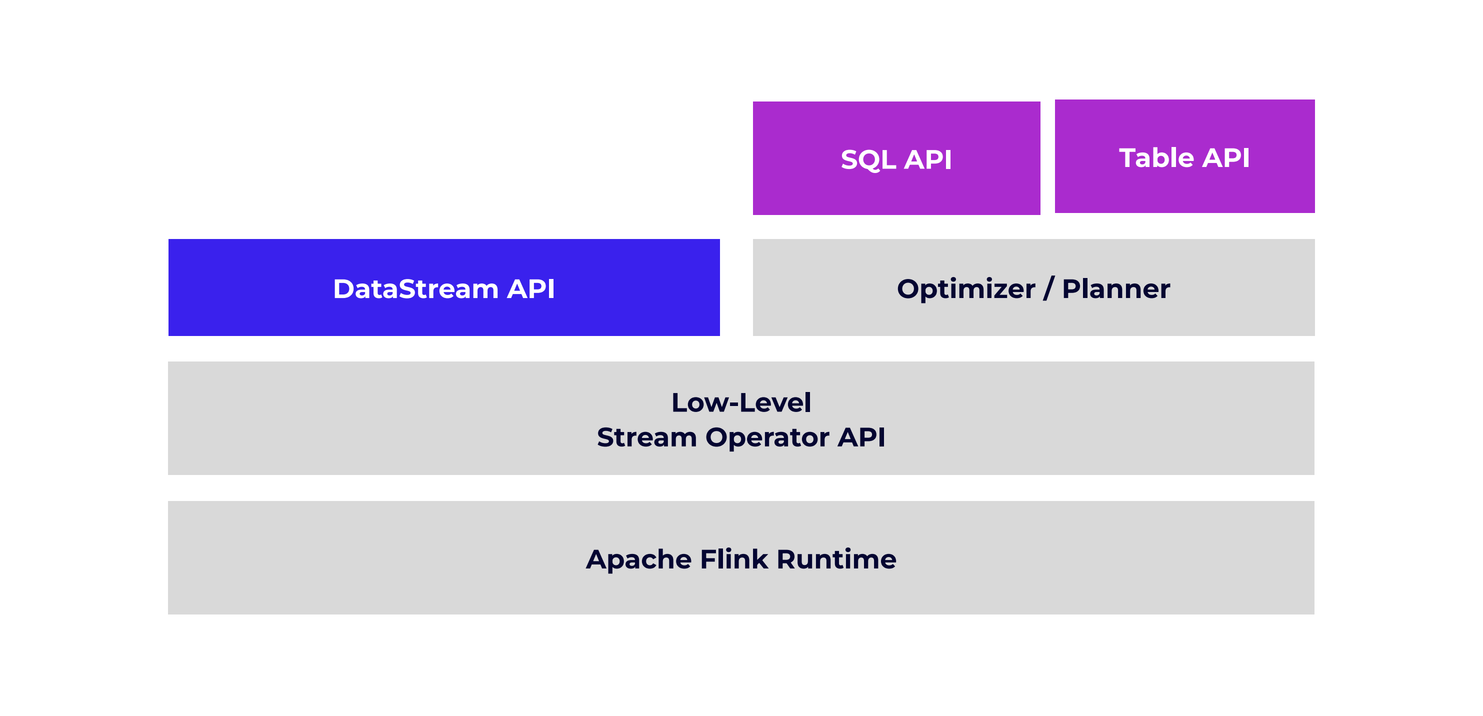
Stream processing with SQL

```
INSERT INTO results
SELECT color, COUNT(*)
FROM events
WHERE color <> orange
GROUP BY color;
```

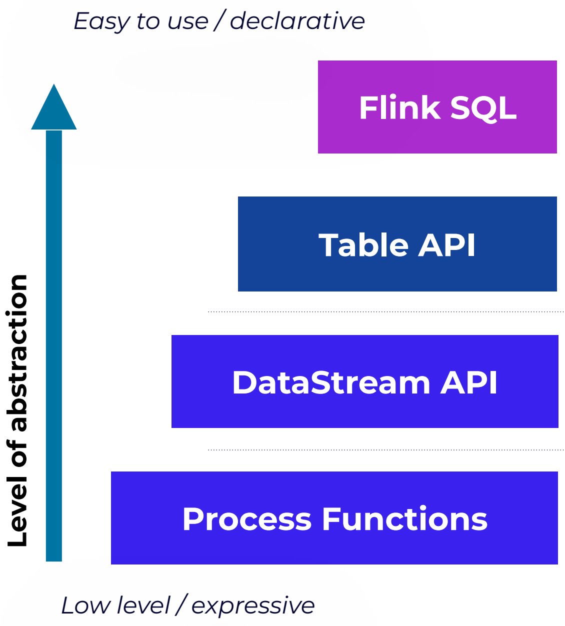


Flink's APIs

CONFLUENT Developer



Flink's APIs: mix & match



Easy to use / declarative

Flink SQL

Table API

DataStream API

Process Functions

Level of abstraction

Low level / expressive

- Code Generation
- Efficient data types
- Cost-based optimizer
- Highly efficient operator implementations (joins, aggregations, deduplications, ...)

  → **Easy to write efficient code with low effort**

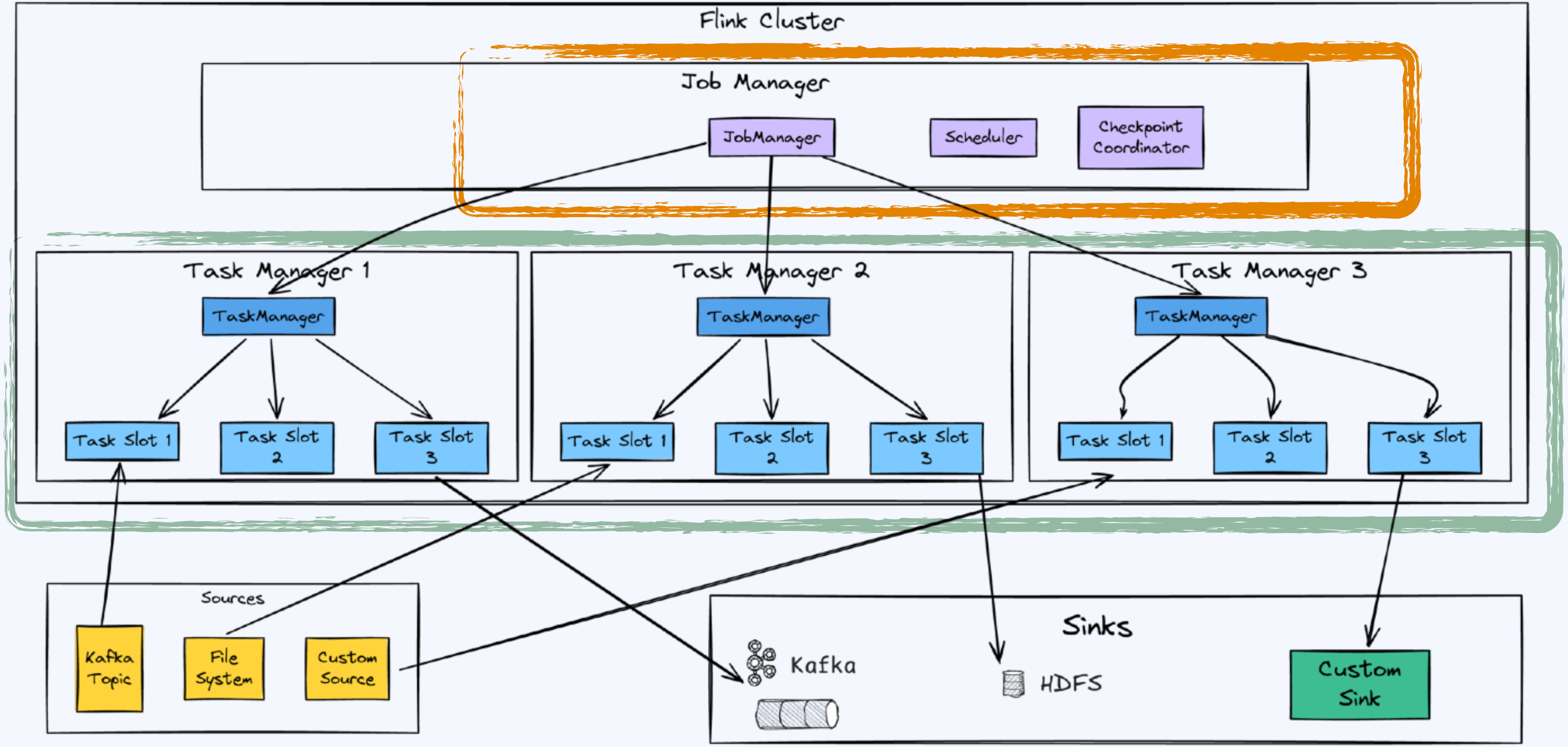
- Ready-made operators for Windowing: sliding, tumbling, session. Late event handling.
- CEP / Async IO operators
- Sources / Sinks / Flink Connectors

- Custom data types
- Raw, low level access to state, time

  → **A lot of potential to make mistakes :)**

State Management

CONFLUENT Developer

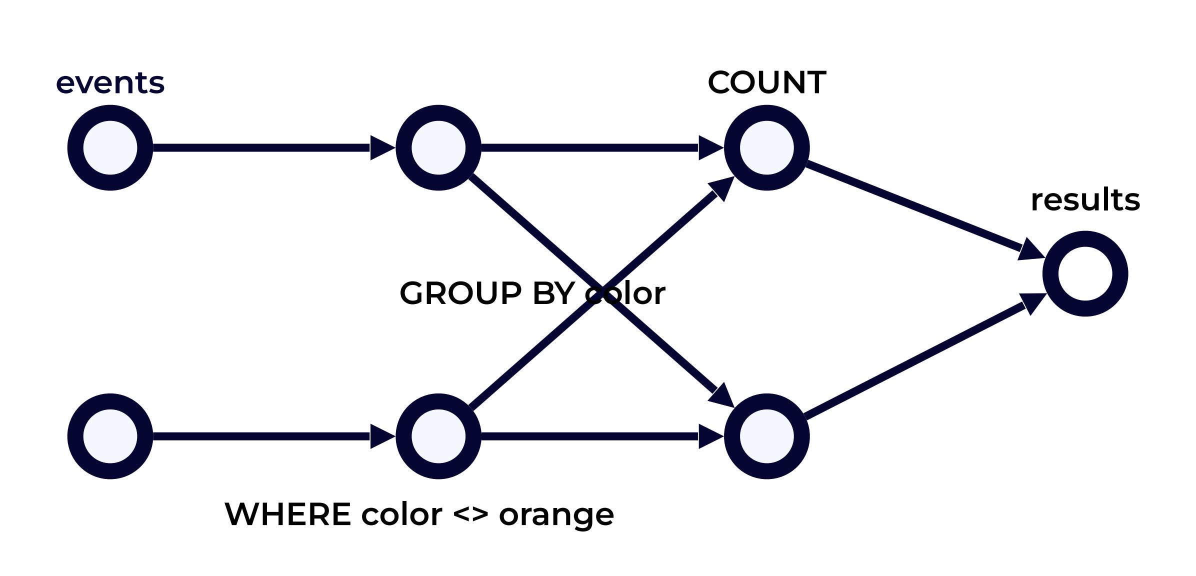


X/Bluesky: @gamussa

Stateful stream processing with Flink SQL

CONFLUENT Developer

```
INSERT INTO results
SELECT color, COUNT(*)
FROM events
WHERE color <> orange
GROUP BY color;
```

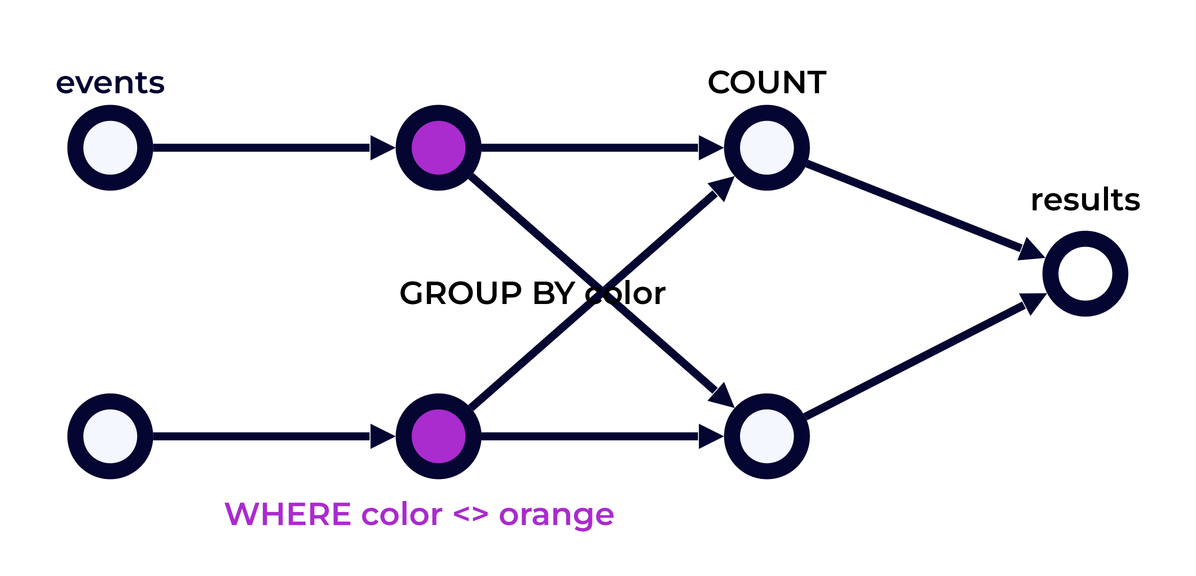


Stateful stream processing with Flink SQL

CONFLUENT Developer

```
INSERT INTO results
SELECT color, COUNT(*)
FROM events
WHERE color <> orange
GROUP BY color;
```

- Filtering is stateless

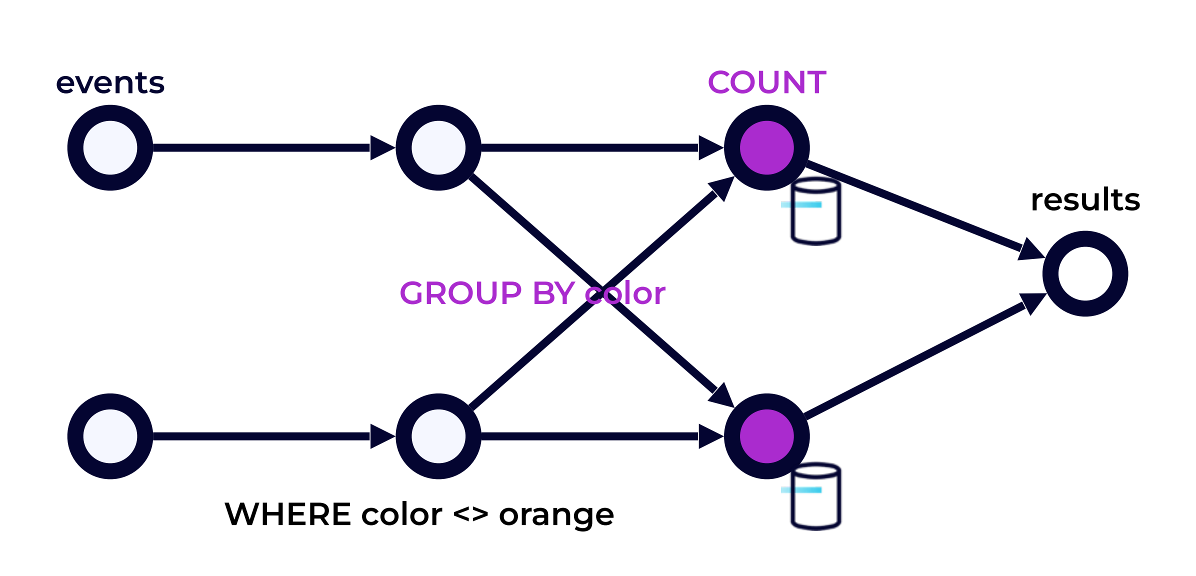


X/Bluesky: @gamussa

Stateful stream processing with FlinkSQL

CONFLUENT Developer

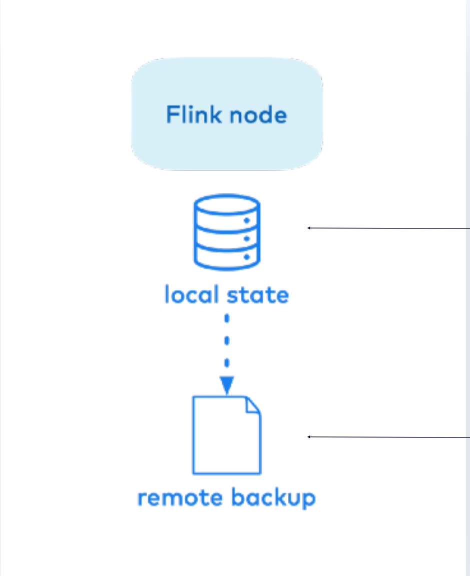
- Counting requires state



State

CONFLUENT Developer

- **Local**
- **Fast**
- **Fault tolerant**



Stored on the heap or on disk using RocksDB (a KV store)

Distributed, reliable storage such as S3 or HDFS

Summary

CONFLUENT Developer

Streaming

A sequence of events.

Unfamiliar to many developers, but ultimately straightforward.

State

Delightfully simple
- local
- key/value
- single-threaded

Event time and watermarks

Watermarks indicate how much progress the time in the stream has made.

State snapshots for recovery

Transparent to application developers, enables correctness and operations.

KAFKA SUMMIT IS NOW

Current

May 20-21 2025 | London

SAVE THE DATE ›

Current

2025

LONDON

THE DATA STREAMING EVENT

May 20-21 | London

ORGANIZED BY CONFLUENT

CONFLUENT
Developer

As Always
Have a Nice Day…

X/Bluesky: @gamussa