Structured Streaming in Apache Spark 2

UNDERSTANDING THE HIGH LEVEL STREAMING API IN SPARK 2.X

Janani Ravi
CO-FOUNDER, LOONYCORN
www.loonycorn.com

Overview

Understand the need for stream processing

Identify differences between batch and stream architectures

Understand Spark DStreams and streaming in Spark 1.x

Analyze advances made in Spark 2.x

Introduce Structured Streaming

Streams modeled as unbounded datasets

Prerequisites and Course Outline

Prerequisite Courses

Beginning Data Exploration and Analysis with Apache Spark

- Programming in Spark 1.x using Python

Getting Started with Spark 2

- Programming in Spark 2.x using Python

Software and Skills

Be very comfortable programming in Python (Python 3)

Understand the basics of distributed computing

Understand the basics of Spark - transformations and aggregations

Course Outline

Introduction to streaming

- Difference between batch and stream architectures
- Structured streaming in Spark 2

Streaming pipelines

- Selections, projections, aggregations of streaming data
- Querying streaming data
- Windowing, event-time aggregations, joins
- Working with Twitter streaming data

Integrating Kafka with Spark

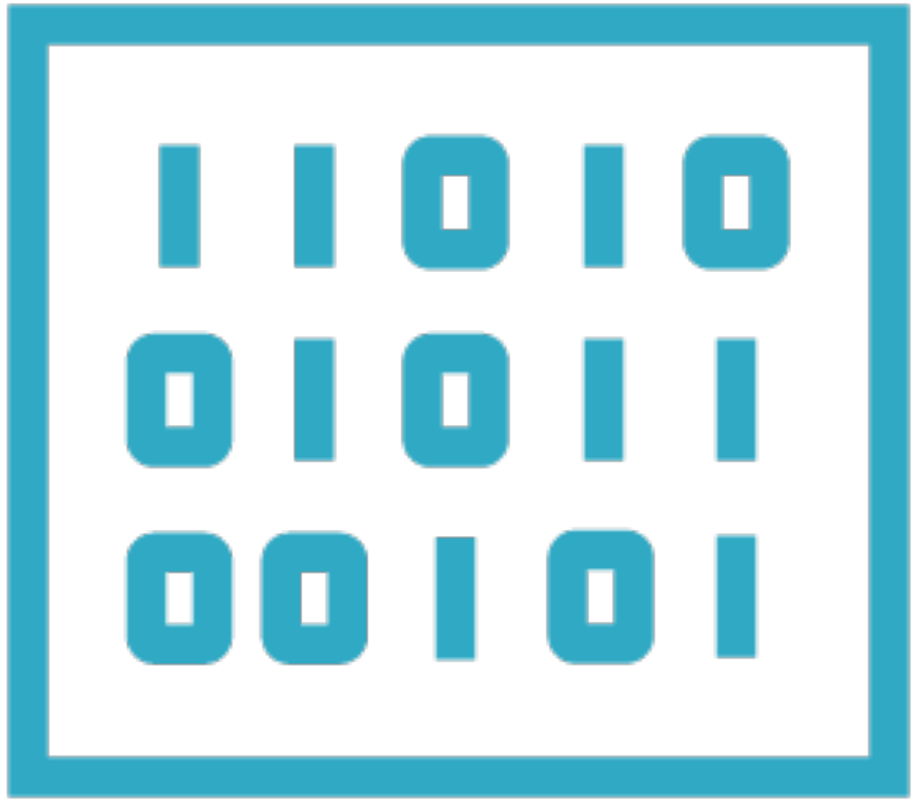
- Stream processing from a Kafka source

RDDs and Spark 1.x

Why is this relevant in Spark 2?

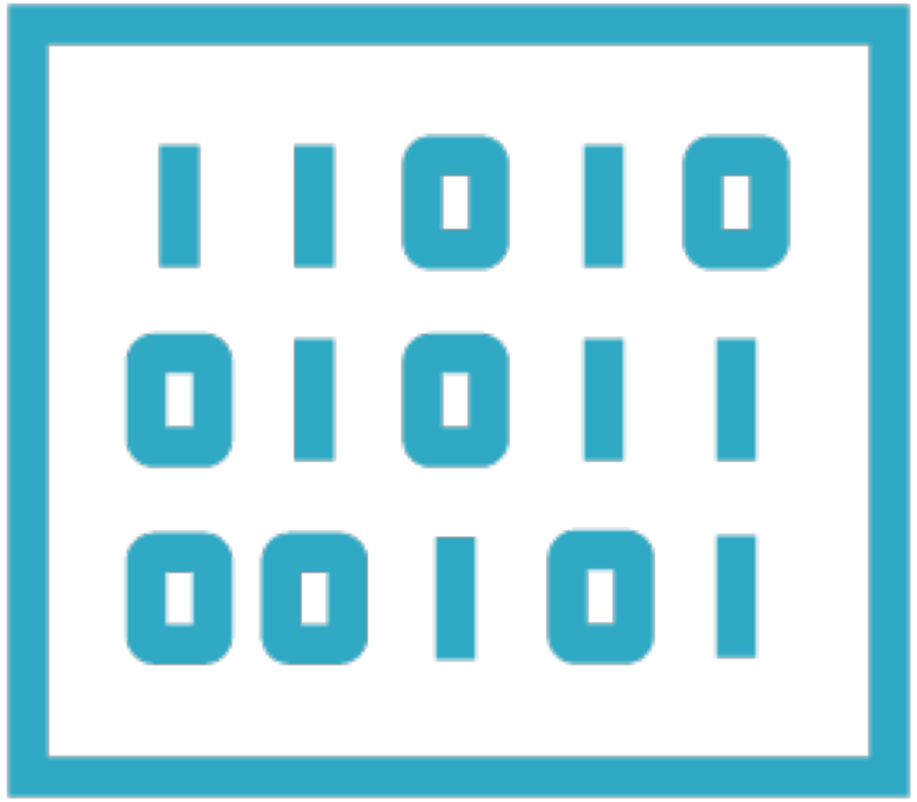
RDDs are still the **fundamental building blocks** of Spark

Resilient Distributed Datasets



All operations in Spark are performed on in-memory objects

Resilient Distributed Datasets



An RDD is a collection of entities - rows, records

Characteristics of RDDs

Partitioned

Split across data nodes in a cluster

Immutable

RDDs, once created, cannot be changed

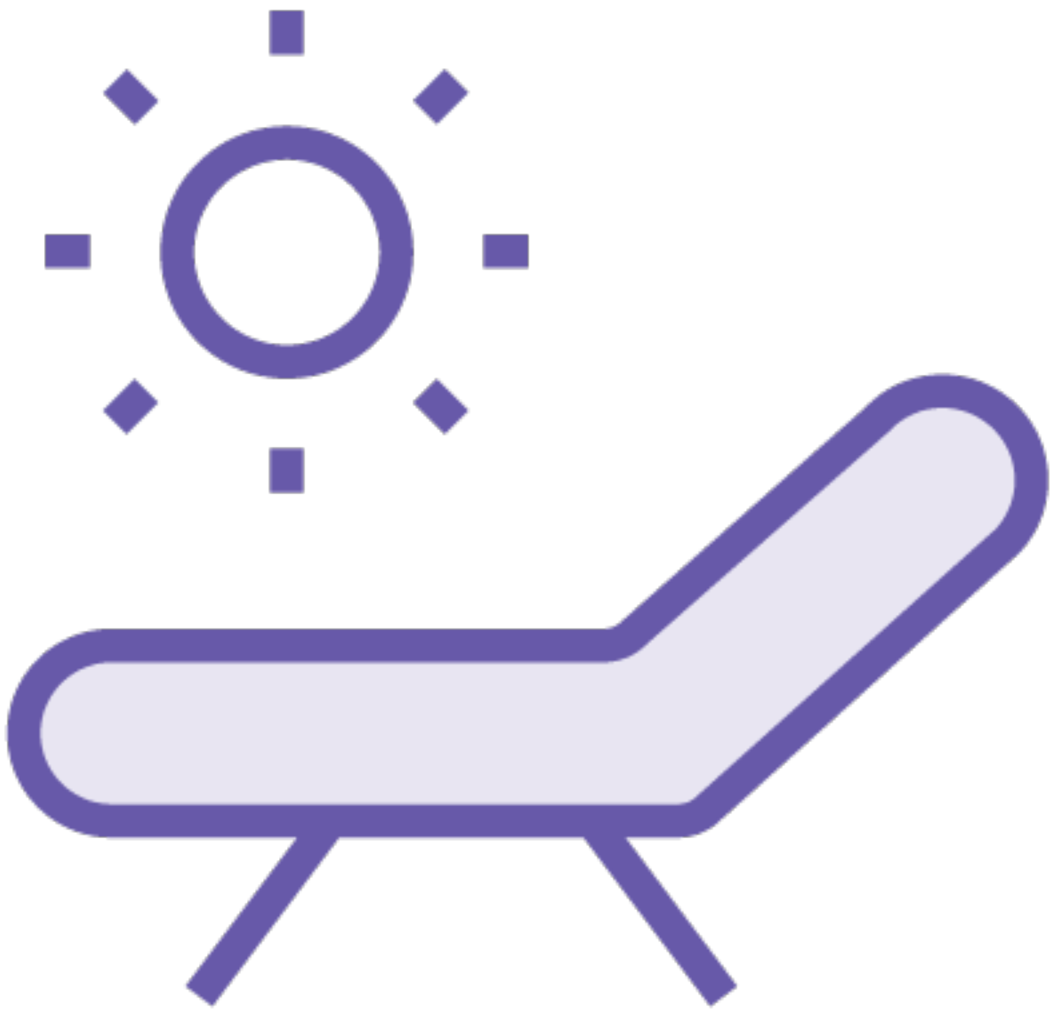
Resilient

Can be reconstructed even when a node crashes

Only Two Types of Operations

Transformation	Action
Transform into another RDD	Request a result

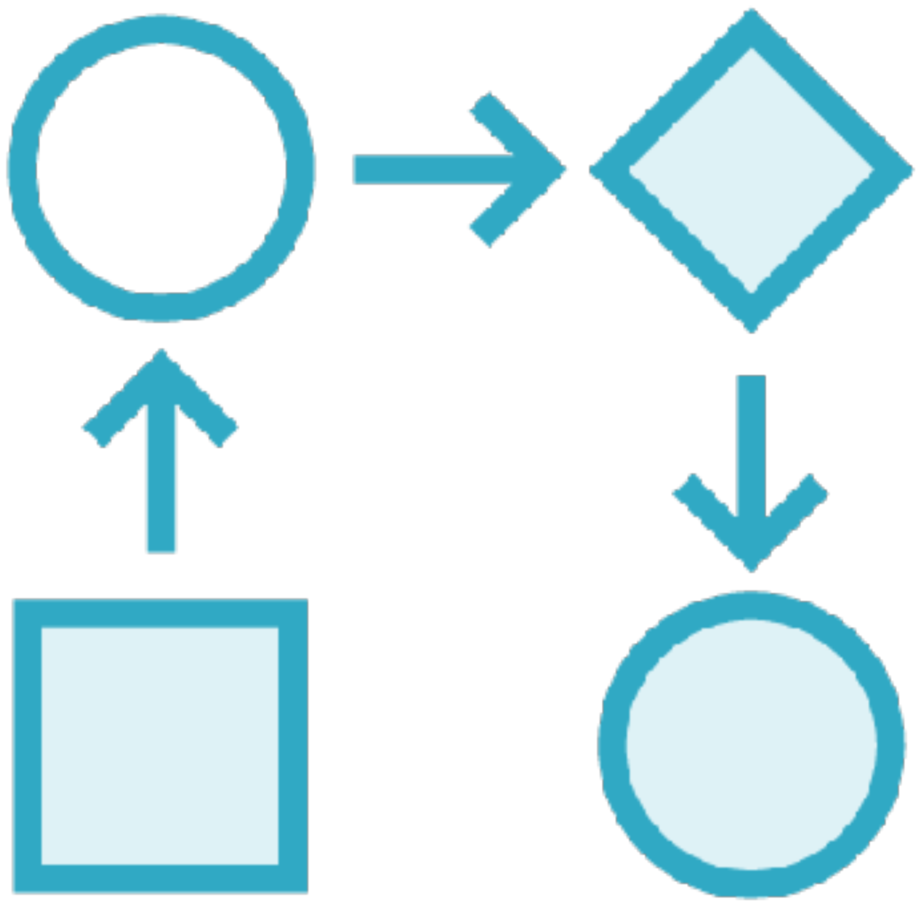
Lineage and Lazy Evaluation



Spark keeps a record of the series of transformations requested by the user

The transformations are materialized or evaluated only when the user requests a result

Lineage and Lazy Evaluation



The record of transformations is called an RDDs lineage

Allows RDDs to be reconstructed on node crashes

Batch and Stream Processing

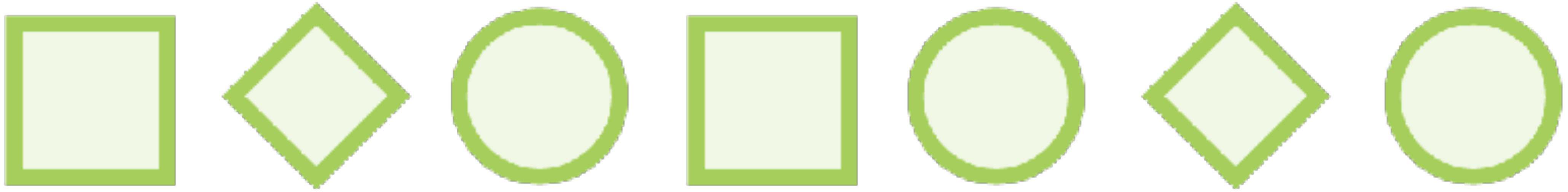
Bounded datasets are processed in **batches**

Unbounded datasets are processed as **streams**

Batch vs. Stream Processing

Batch	Stream
Bounded, finite datasets	Unbounded, infinite datasets
Slow pipeline from data ingestion to analysis	Processing immediate, as data is received
Periodic updates as jobs complete	Continuous updates as jobs run constantly
Order of data received unimportant	Order important, out of order arrival tracked
Single global state of the world at any point in time	No global state, only history of events received

Stream Processing



Data is received as a stream

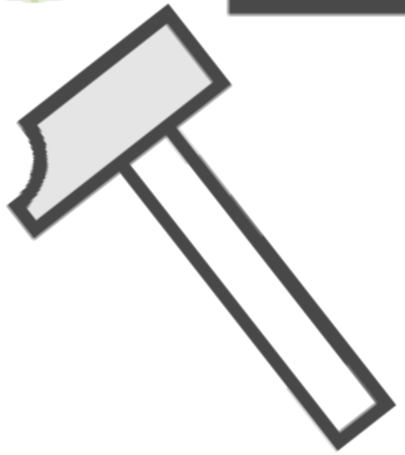
- Log messages
- Tweets
- Climate sensor data

Stream Processing

Process the data one entity at a time

- Filter error messages
- Find references to the latest movies
- Track weather patterns

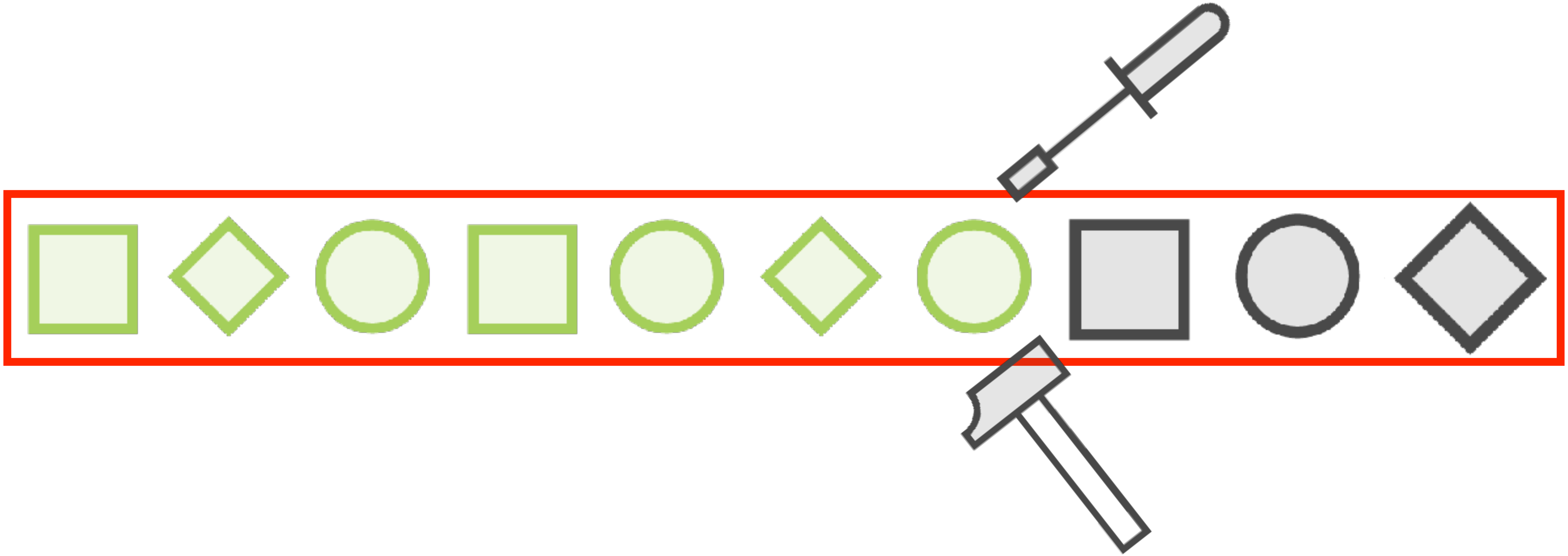
Stream Processing



Store, display, act on filtered messages

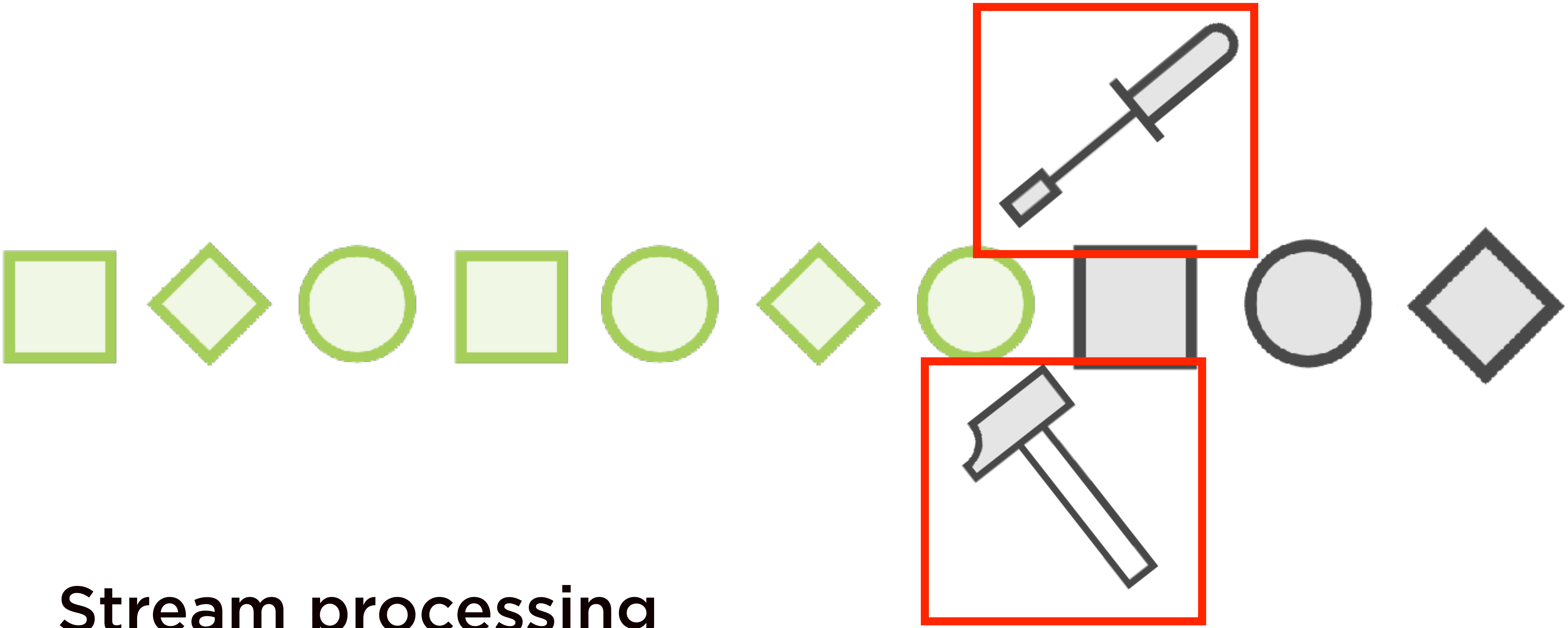
- Trigger an alert
- Show trending graphs
- Warn of sudden squalls

Stream Processing



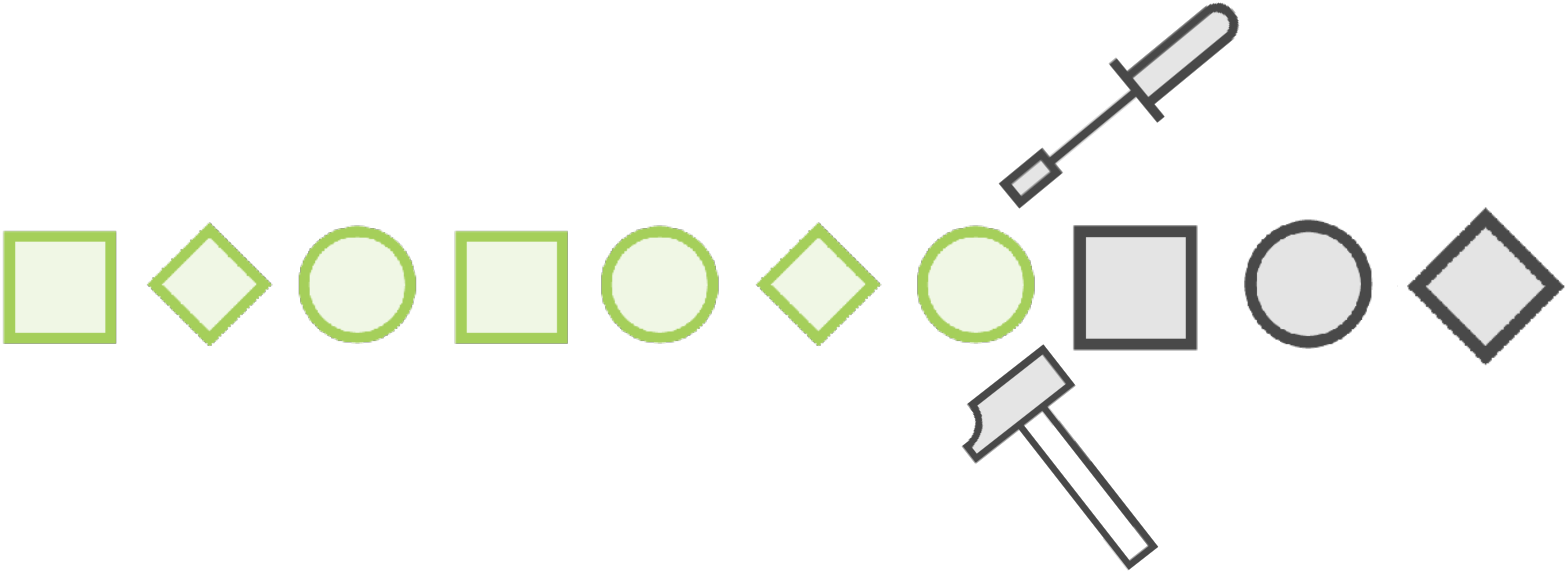
Streaming data

Stream Processing



Stream processing

Stream Processing



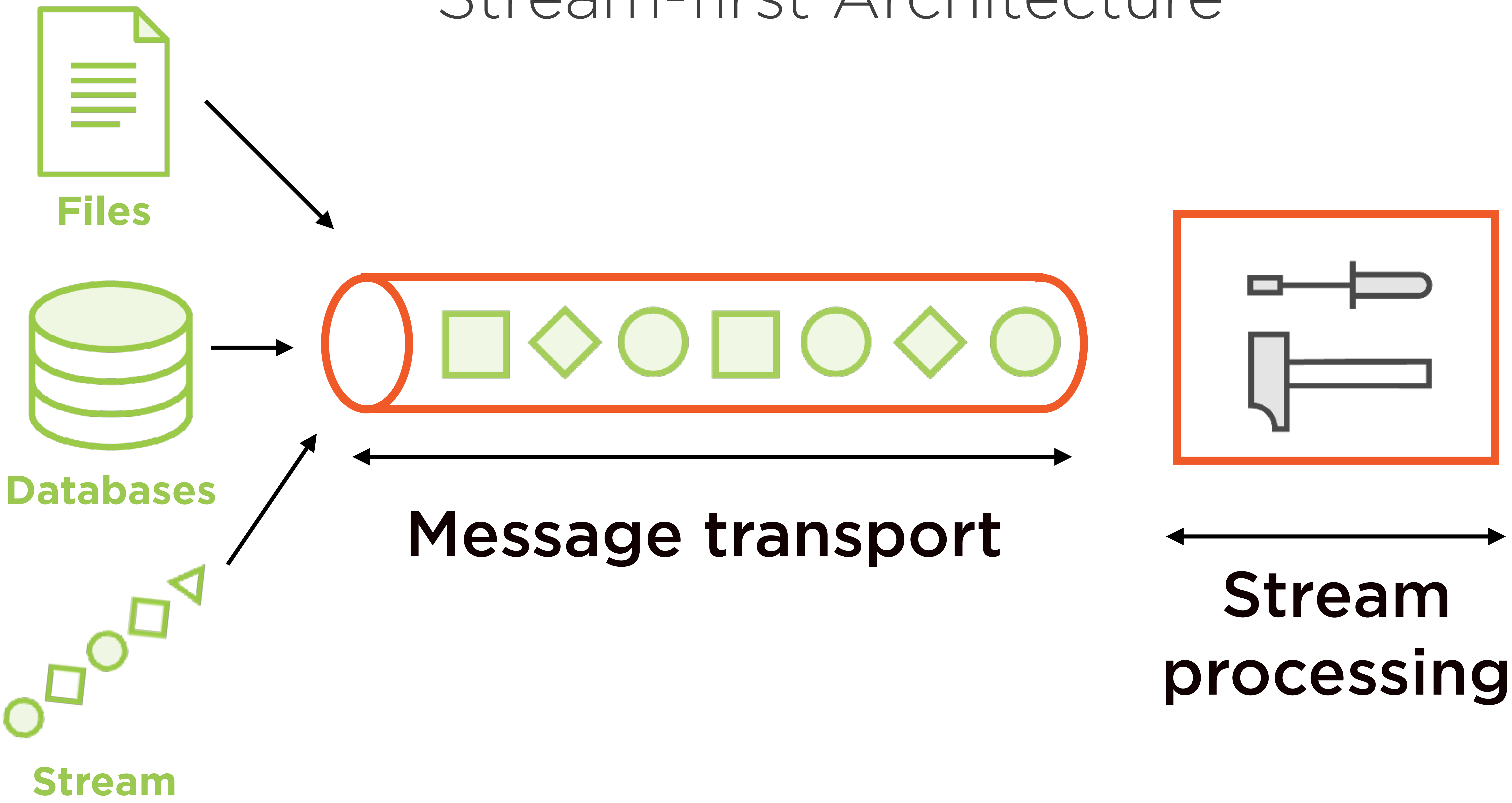
Traditional Systems

Files

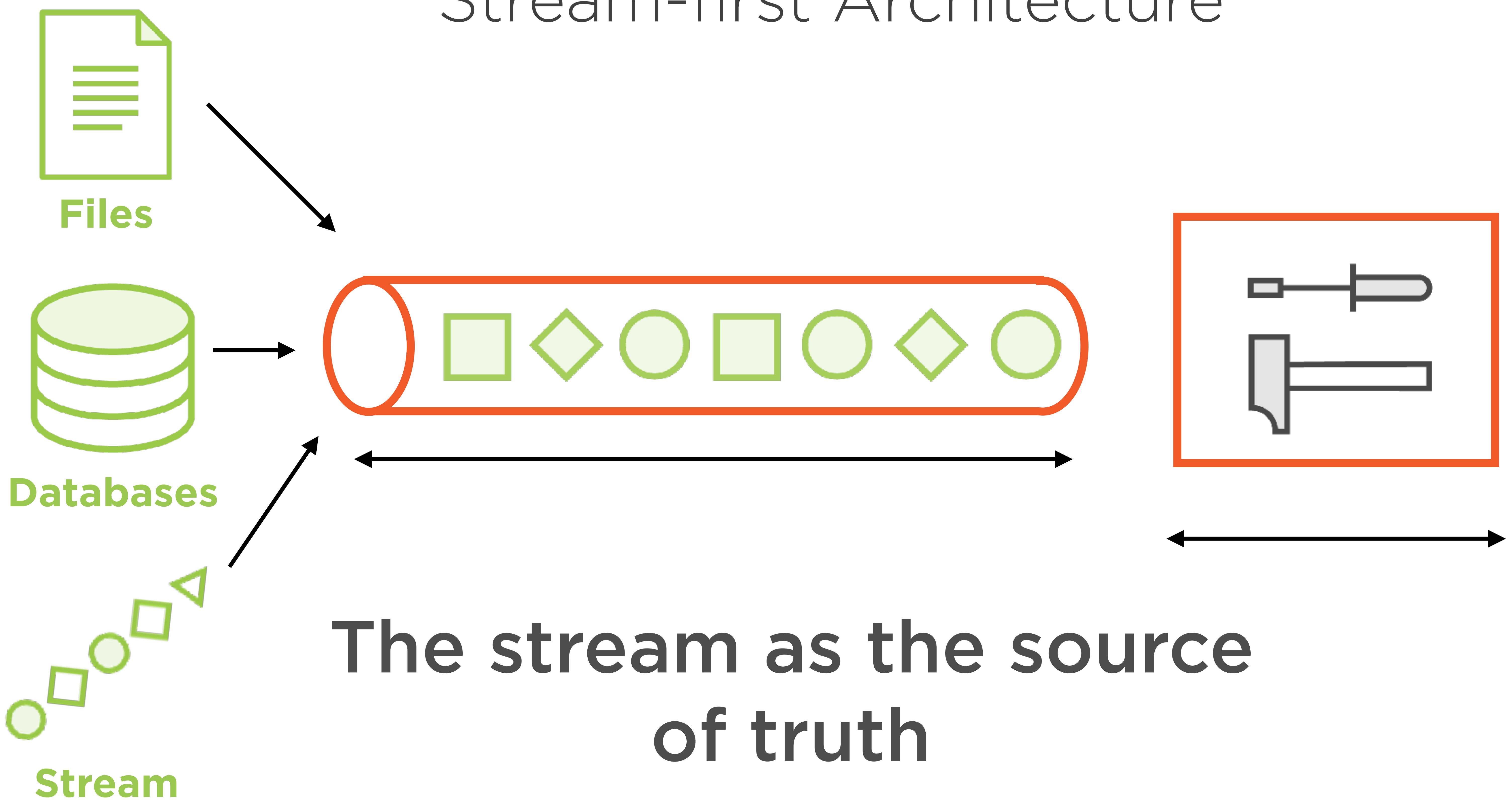
Databases

Reliable storage as the source of truth

Stream-first Architecture



Stream-first Architecture



The stream as the source of truth

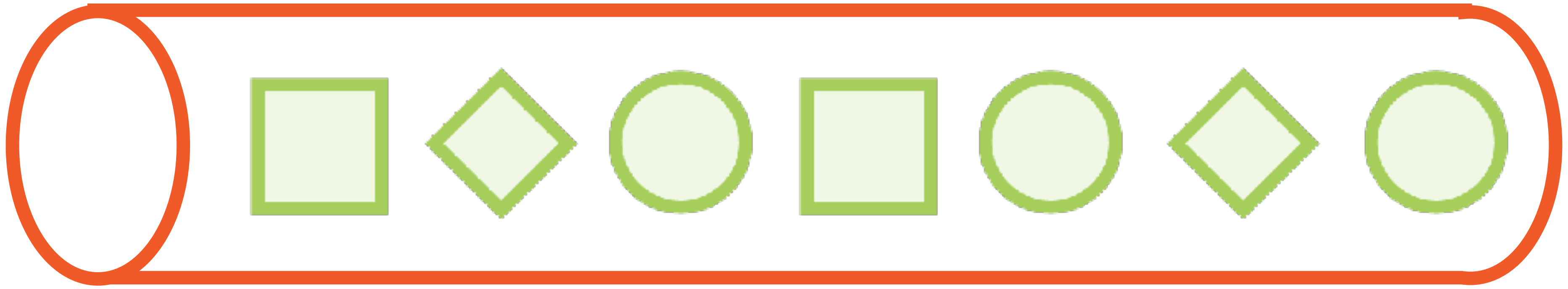
Message Transport

Buffer for event data

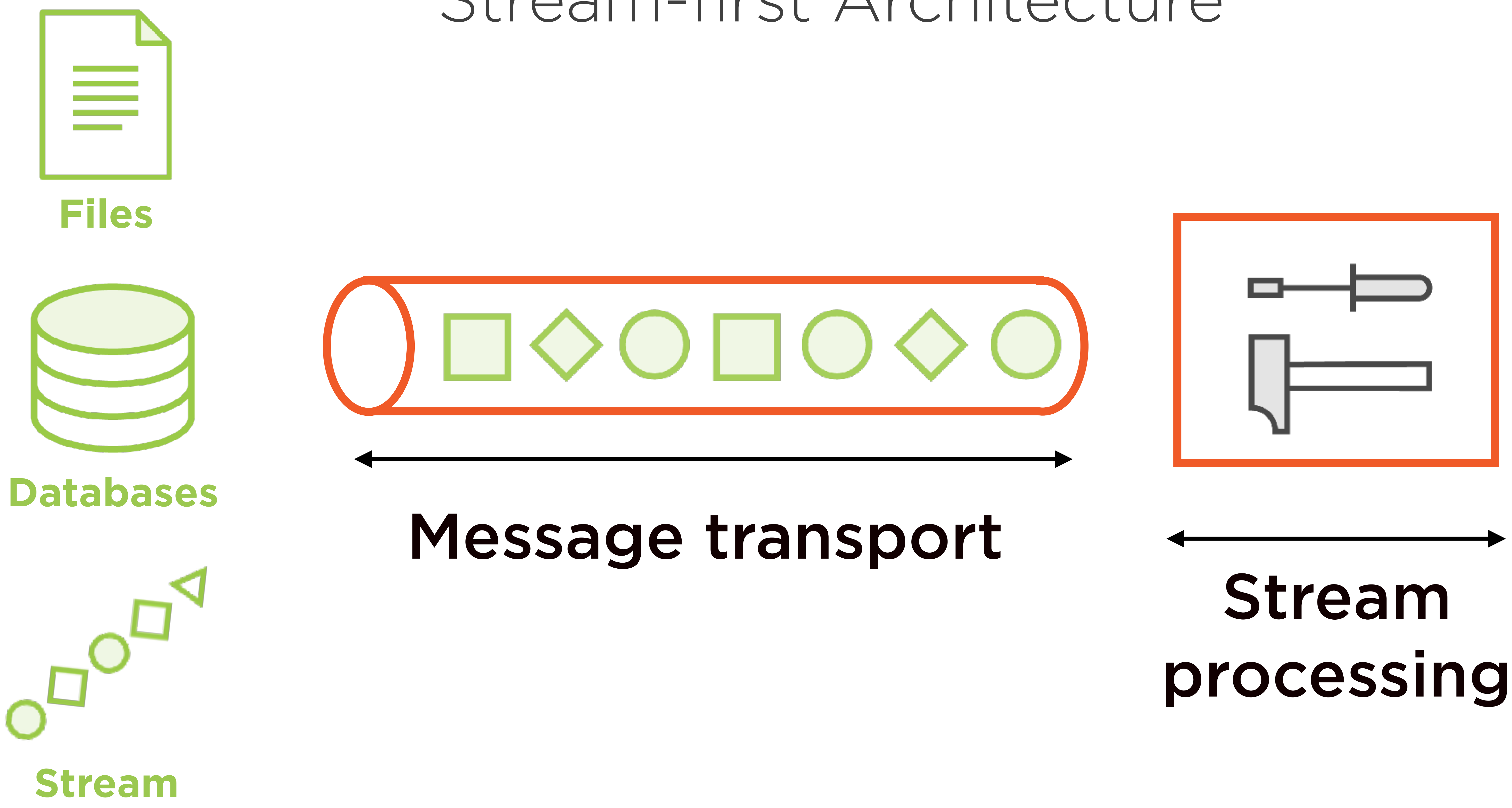
Performant and persistent

Decoupling multiple sources from processing

Kafka, MapR streams



Stream-first Architecture



High throughput, low latency

Fault tolerance with low overhead

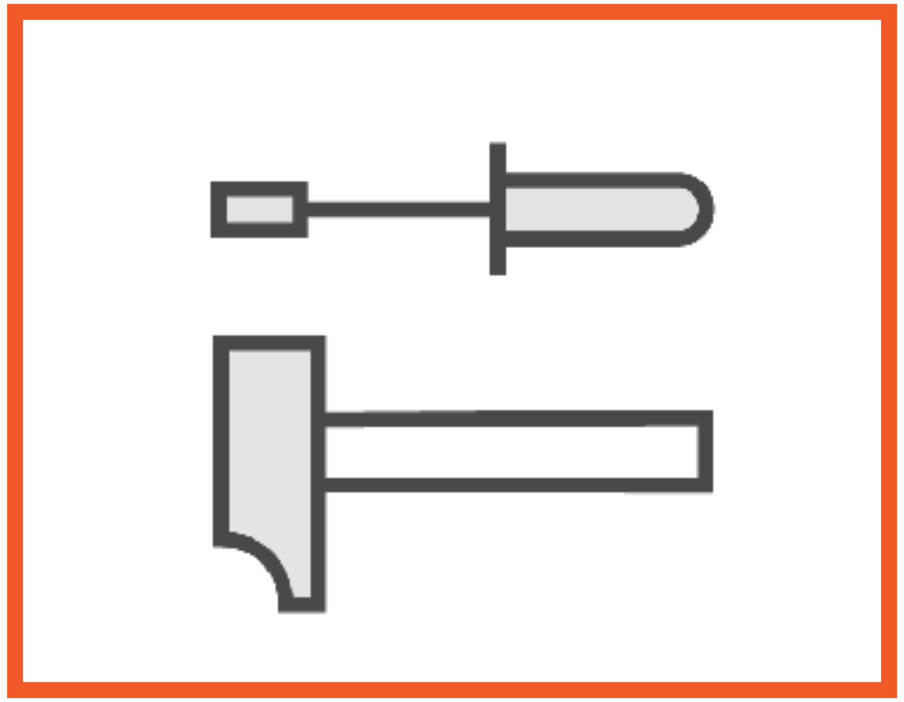
Manage out of order events

Easy to use, maintainable

Replay streams

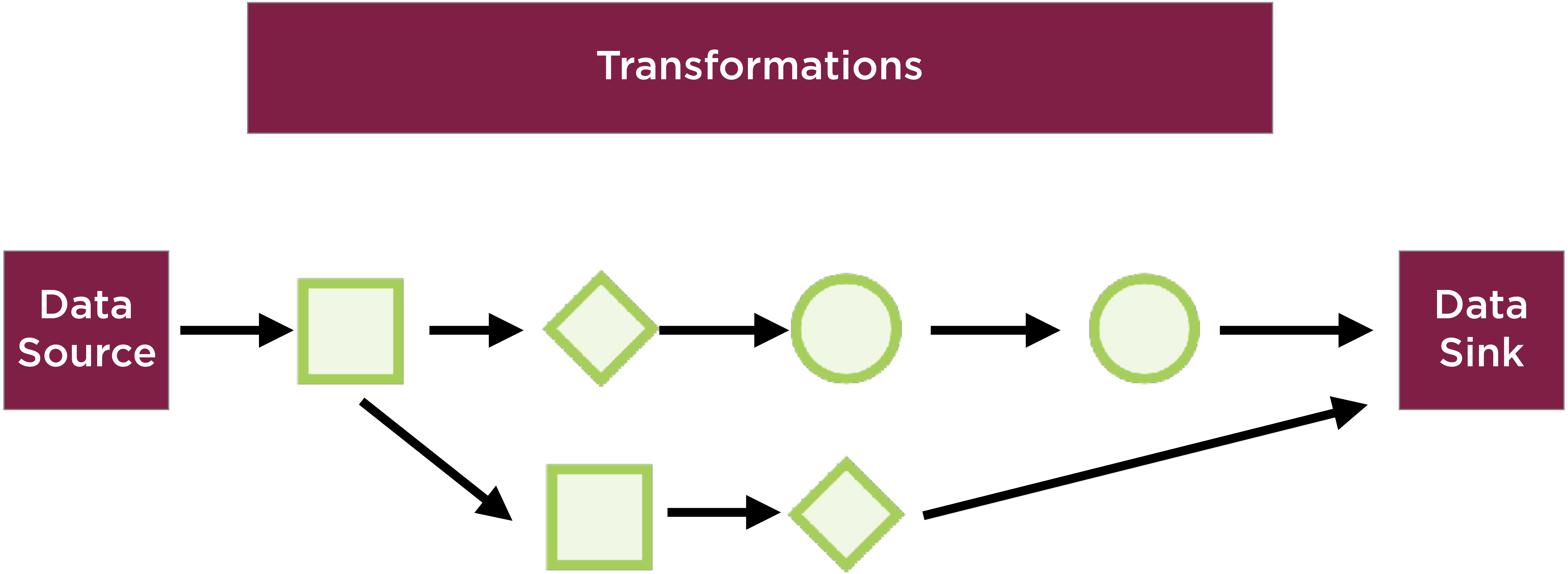
Spark Streaming, Storm, Flink

Stream Processing



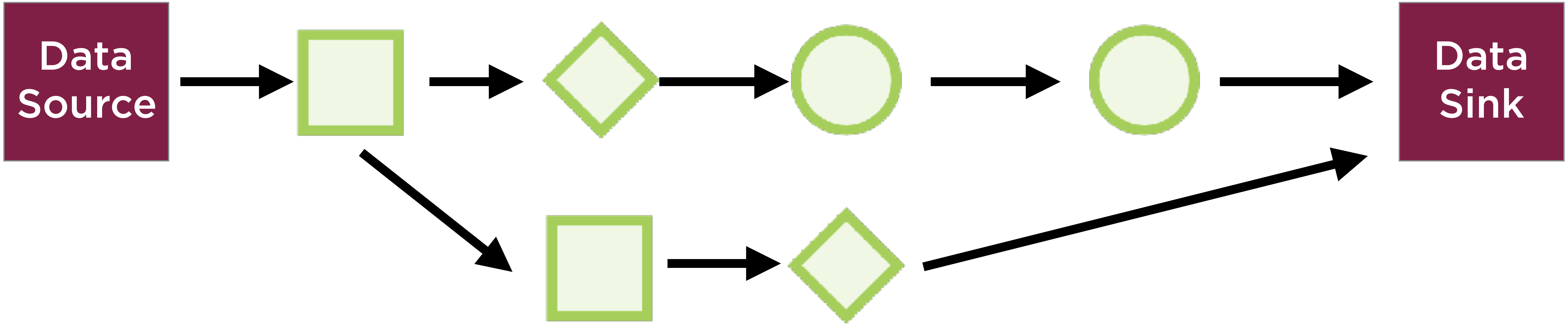
Stream Processing Model

Stream Processing Model



Transformations

A directed-acyclic graph



Streaming in Spark 1.x

Log messages received from a server can be thought of as a stream

Streaming Data

```
2016-12-30 09:09:57,862 INFO
org.apache.hadoop.http.HttpServer2: Jetty bound to port
56745
2016-12-30 09:09:57,862 INFO org.mortbay.log: jetty-6.1.26
2016-12-30 09:09:58,037 INFO org.mortbay.log: Started
HttpServer2$SelectChannelConnectorWithSafeStartup@localhost:
56745
2016-12-30 09:09:58,124 INFO
org.apache.hadoop.hdfs.server.datanode.web.DatanodeHttpServe
r: Listening HTTP traffic on /0.0.0.0:50075
2016-12-30 09:09:58,239 INFO
```

Streaming Data

```
2016
-12-
30
09:0
9:58
,239
INFO
```

```
org.apache.hado
op.hdfs.server.
datanode.web.Da
tanodeHttpServe
r: Listening
HTTP traffic on
/0.0.0.0:50075
```

```
HttpServer
2$SelectCh
annelConne
ctorWithSa
feStartup@
localhost:
56745
```

```
2016-12-3
0
09:09:58,
037 INFO
org.mortb
ay.log:
Started
```

```
2016-12-30
09:09:57,8
62 INFO
org.mortba
y.log:
jetty-6.1.
26
```

```
2016-12-30
09:09:57,862
INFO
org.apache.hadoo
p.http.HttpServe
r2: Jetty bound
to port 56745
```

Each message is one entity in this stream

Streaming Data

2016 -12- 30 09:0 9:58 ,239 INFO	org.apache.hadoop.hdfs.server.datanode.web.DatanodeHttpServer: Listening HTTP traffic on /0.0.0.0:50075	HttpServer2$SelectChannelConnectorWithSafeStartup@localhost:56745	2016-12-30 09:09:58,037 INFO org.mortbay.log: Started	2016-12-30 09:09:57,862 INFO org.mortbay.log: jetty-6.1.26	2016-12-30 09:09:57,862 INFO org.apache.hadoop.http.HttpServer2: Jetty bound to port 56745

Spark works with stream data using the same batch RDD abstraction

Streaming Data

2016
-12-
30
09:0
9:58
,239
INFO

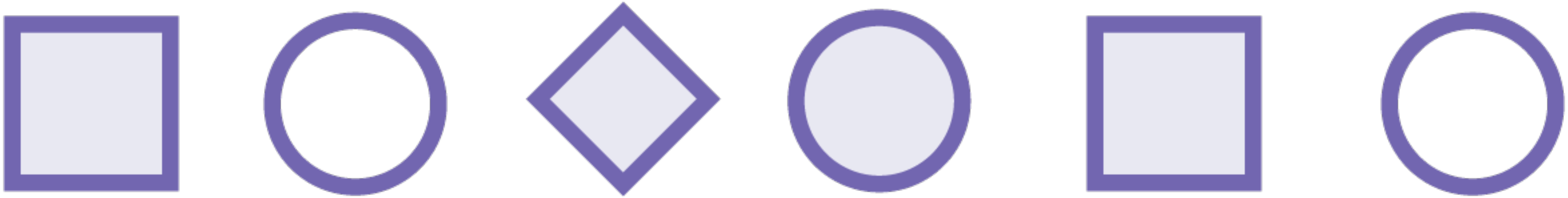
org.apache.hado
op.hdfs.server.
datanode.web.Da
tanodeHttpServe
r: Listening
HTTP traffic on
/0.0.0.0:50075

HttpServer
2$SelectCh
annelConne
ctorWithSa
feStartup@
localhost:
56745

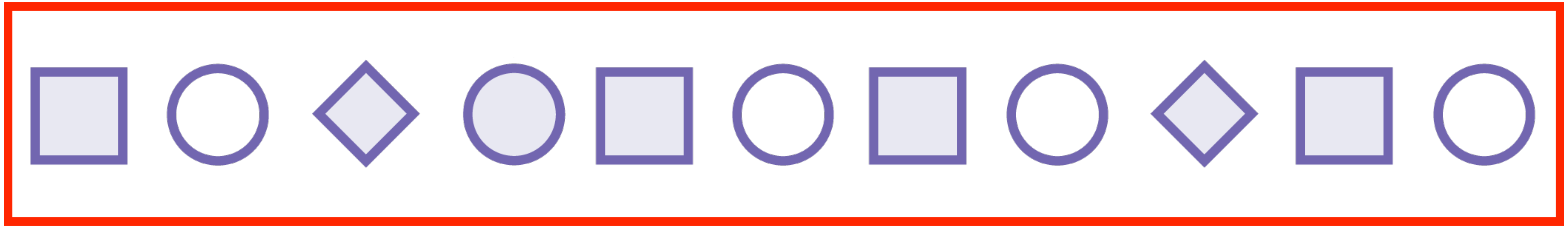
2016-12-3
0
09:09:58,
037 INFO
org.mortb
ay.log:
Started

2016-12-30
09:09:57,8
62 INFO
org.mortba
y.log:
jetty-6.1.
26

2016-12-30
09:09:57,862
INFO
org.apache.hadoo
p.http.HttpServe
r2: Jetty bound
to port 56745



Streaming Data



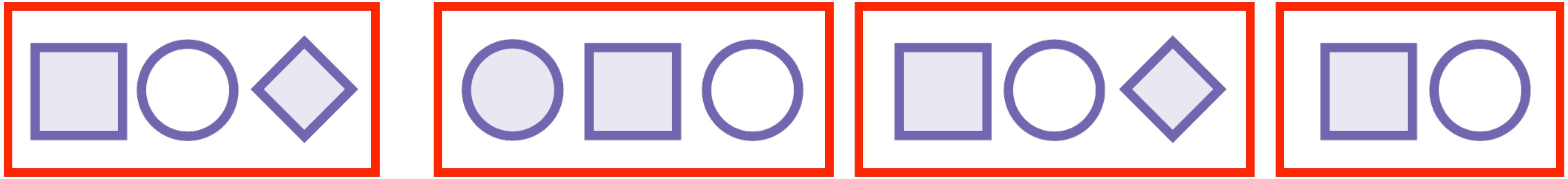
This stream of entities is represented as a discretized stream or DStream

DStream = Sequence of RDDs

Streaming Data

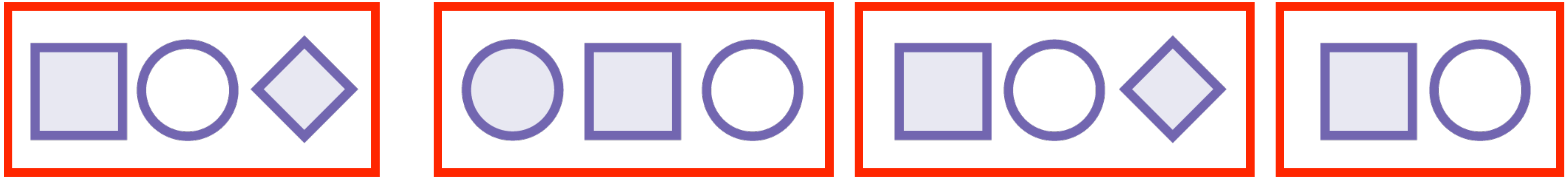
Entities => RDDs => DStream ?

Streaming Data



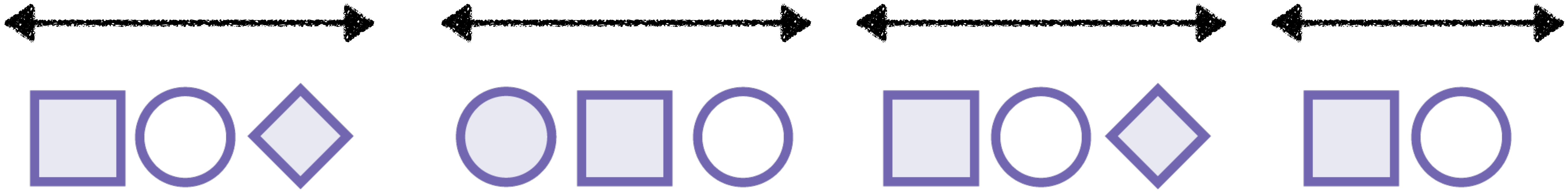
Entities in a Stream are grouped into batches

Streaming Data



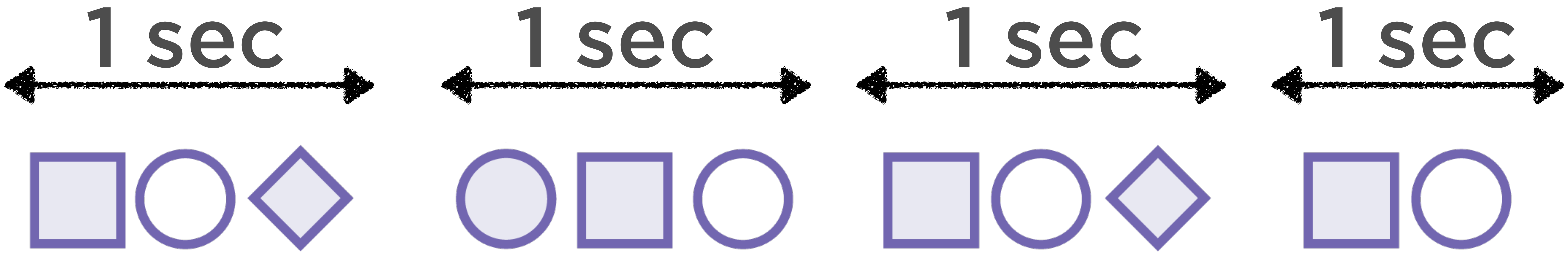
Each batch = 1 RDD

Streaming Data



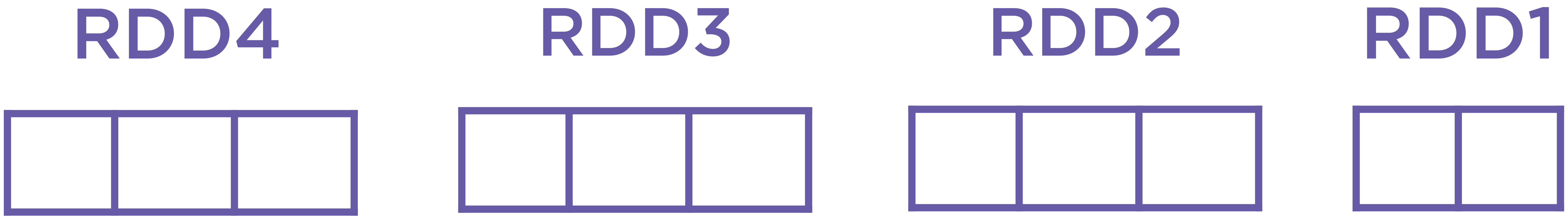
Batches are formed based on a **batch interval**

Streaming Data



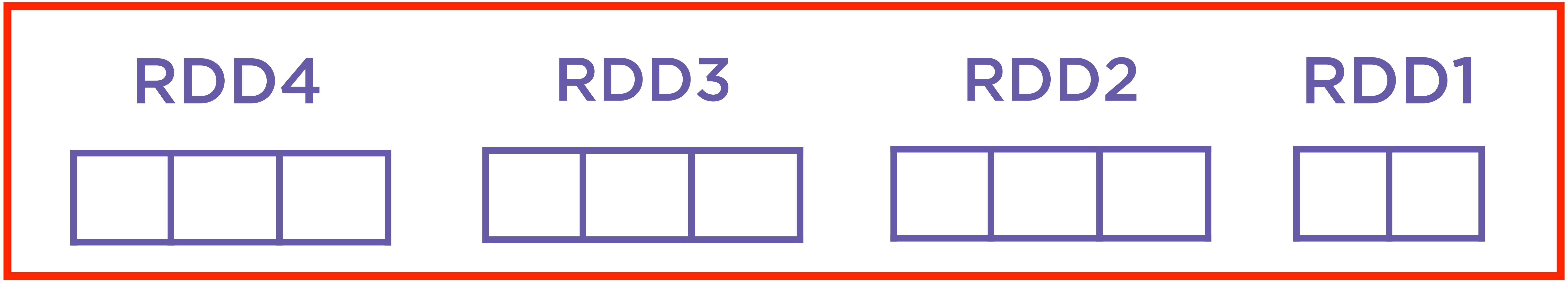
All logs received **within the batch interval** make one RDD

Streaming Data



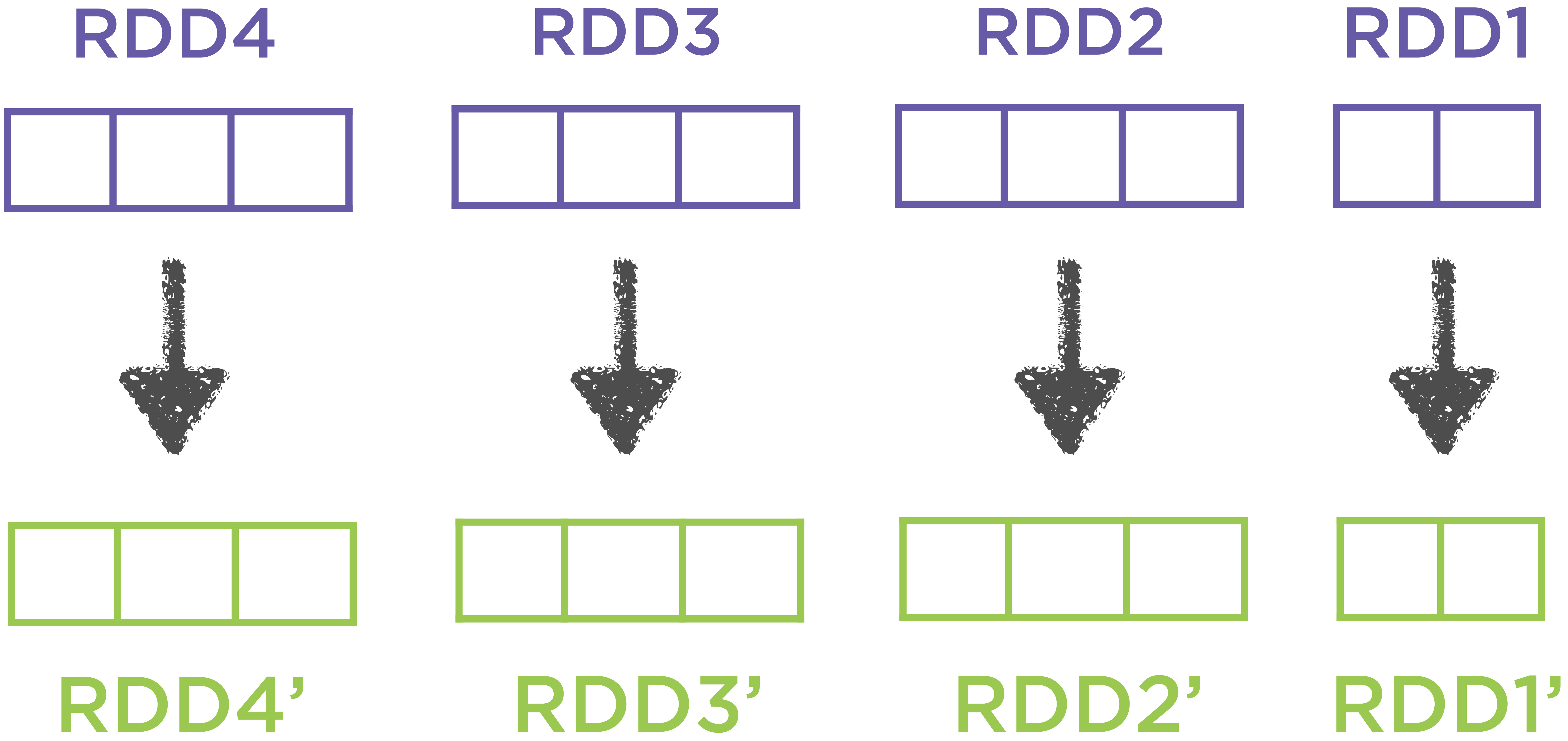
All logs received within the batch interval make one RDD

Streaming Data

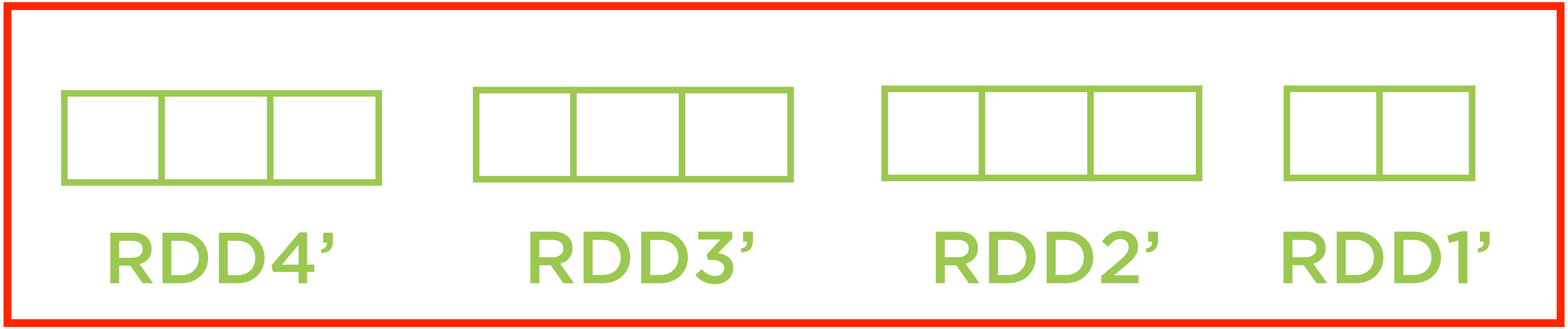


Within a DStream, Spark still performs operations on individual RDDs

Streaming Data



Streaming Data



Transformed DStream

Within DStreams, Spark 1.x does **batch processing on individual RDDs**

Spark 1.x to Spark 2.x

Changes Starting Spark 2.0

Easier

Unifying Datasets and DataFrames, SQL support…

Faster

Optimize like a compiler, not a DBMS

Execution in Spark 1.x

In Spark 1.x, execution optimization resembled traditional DBMS

“Volcano Iterator Model”

Missed several code/compiler optimizations

Execution in Spark 2.x

Tungsten engine (2nd generation)

Eliminate virtual function calls

Store data in registers, not RAM/cache

Compiler loop unrolling, pipelining

Spark 2.0 uses Tungsten, an engine that speeds up execution 10-20X

Performance Improvements

Comparison of time per row, on 1 billion records on single thread

Primitive	Spark 1.6	Spark 2.0	Speedup Factor
filter	15ns	1.1ns	13.6
sum w/o group	14ns	0.9ns	15.6
sum w/ group	79ns	10.7ns	7.4
hash join	115ns	4.0ns	28.8
sort (8-bit)	620ns	5.3ns	117.0
sort (64-bit)	620ns	40ns	15.5
sort-merge-join	750ns	700ns	1.1

Source: https://databricks.com/blog/2016/07/26/introducing-apache-spark-2-0.html

The basic data structure for records in Spark 2.x is the DataFrame

DataFrame: Data in Rows and Columns

DATE	OPEN	...	PRICE
2016-12-01	772	...	779
2016-11-01	758	...	747
2006-01-01	302	...	309

Each row represents
1 observation

DataFrame: Data in Rows and Columns

DATE	OPEN	...	PRICE
2016-12-01	772	...	779
2016-11-01	758	...	747
2006-01-01	302	...	309

Each column represents 1 variable (a list or vector)

From File to DataFrame

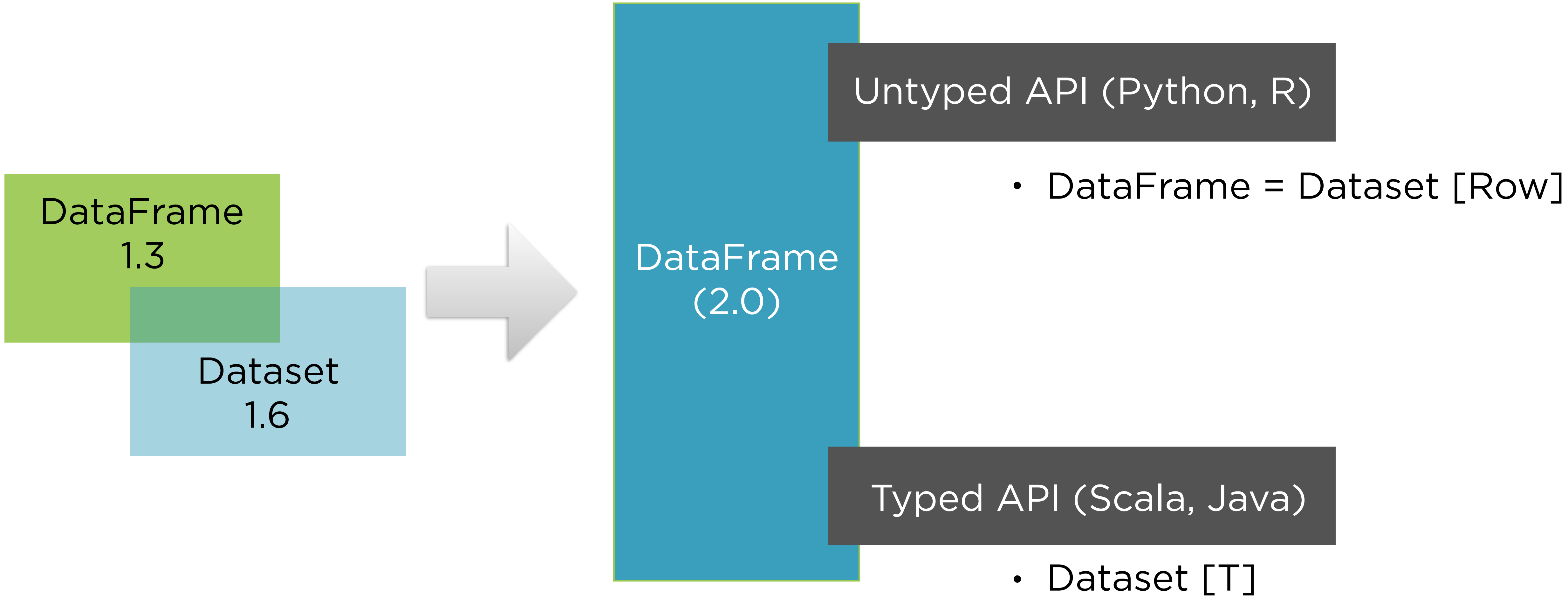
DATE	OPEN	...	PRICE
2016-12-01	772	...	779
2016-11-01	758	...	747
2006-01-01	302	...	309

read

DATE	OPEN	...	PRICE
2016-12-01	772	...	779
2016-11-01	758	...	747
2006-01-01	302	...	309

File **DataFrame**

Unified API for DataFrames



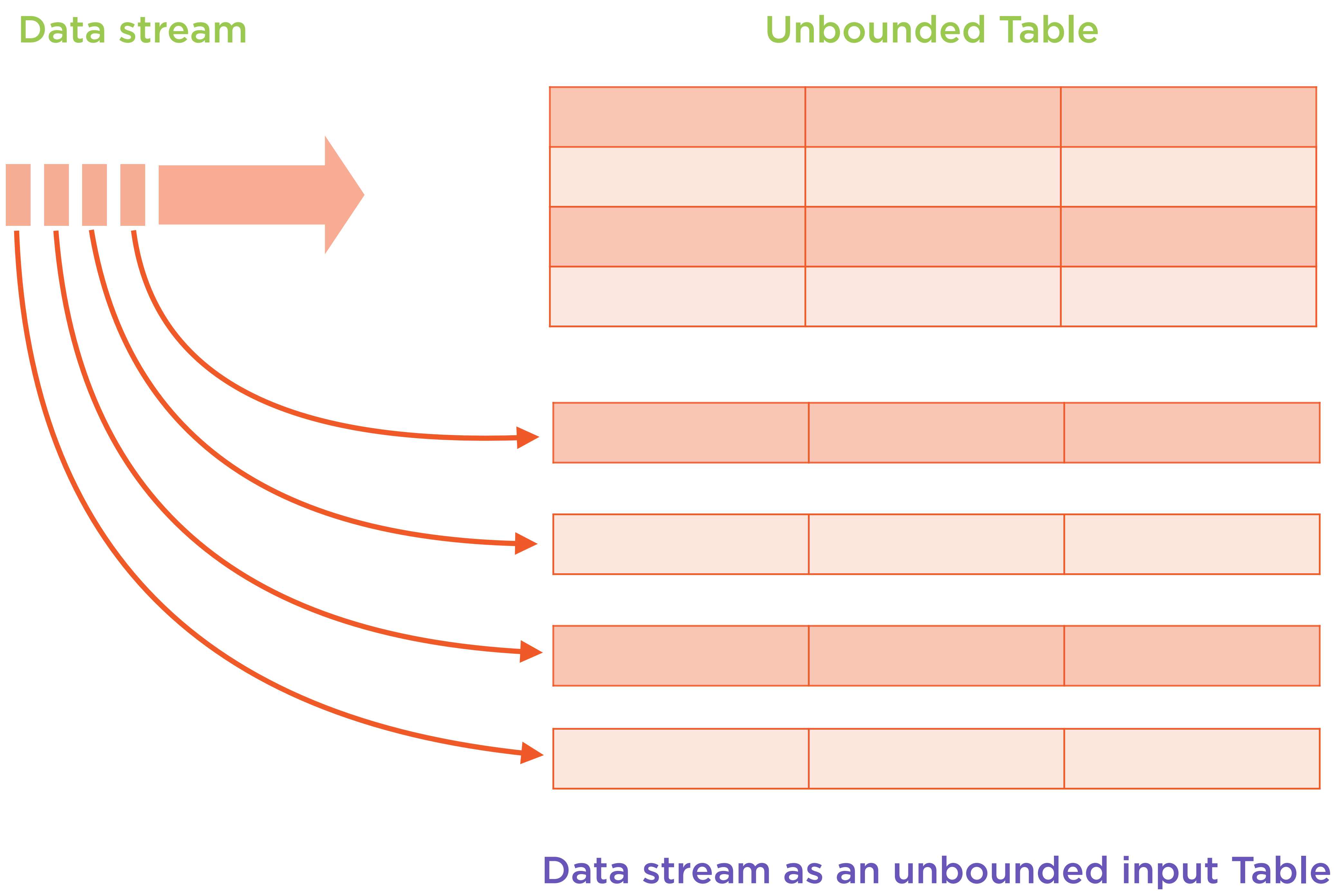
Starting Spark 2.0, APIs for Datasets and DataFrames have merged

DataFrames Built on Top of RDDs

Partitioned	Immutable	Resilient
Split across data nodes in a cluster	Once created, cannot be changed	Can be reconstructed even when a node crashes

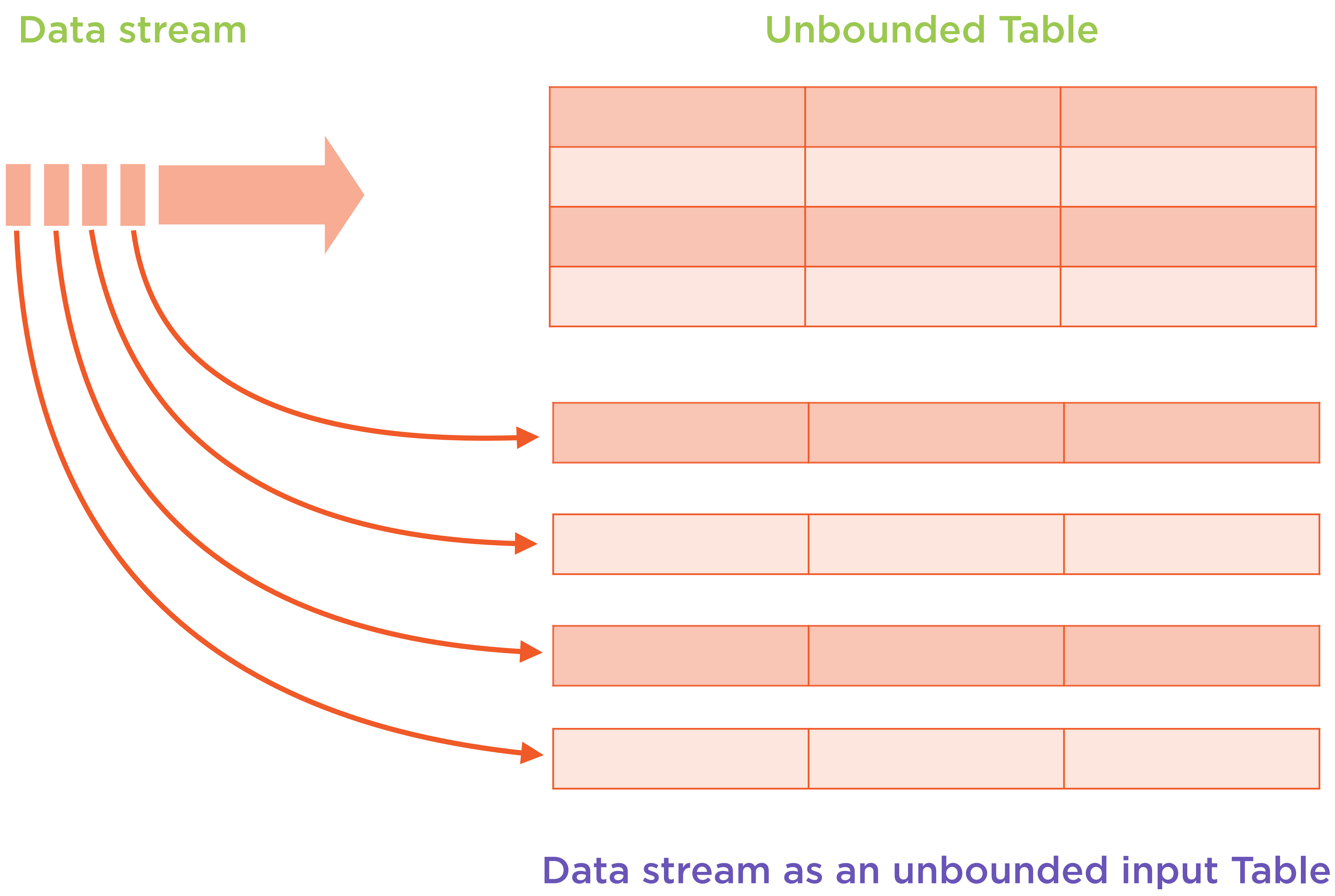
The Intuition Behind Structured Streaming

Batch is Simply Prefix of Stream



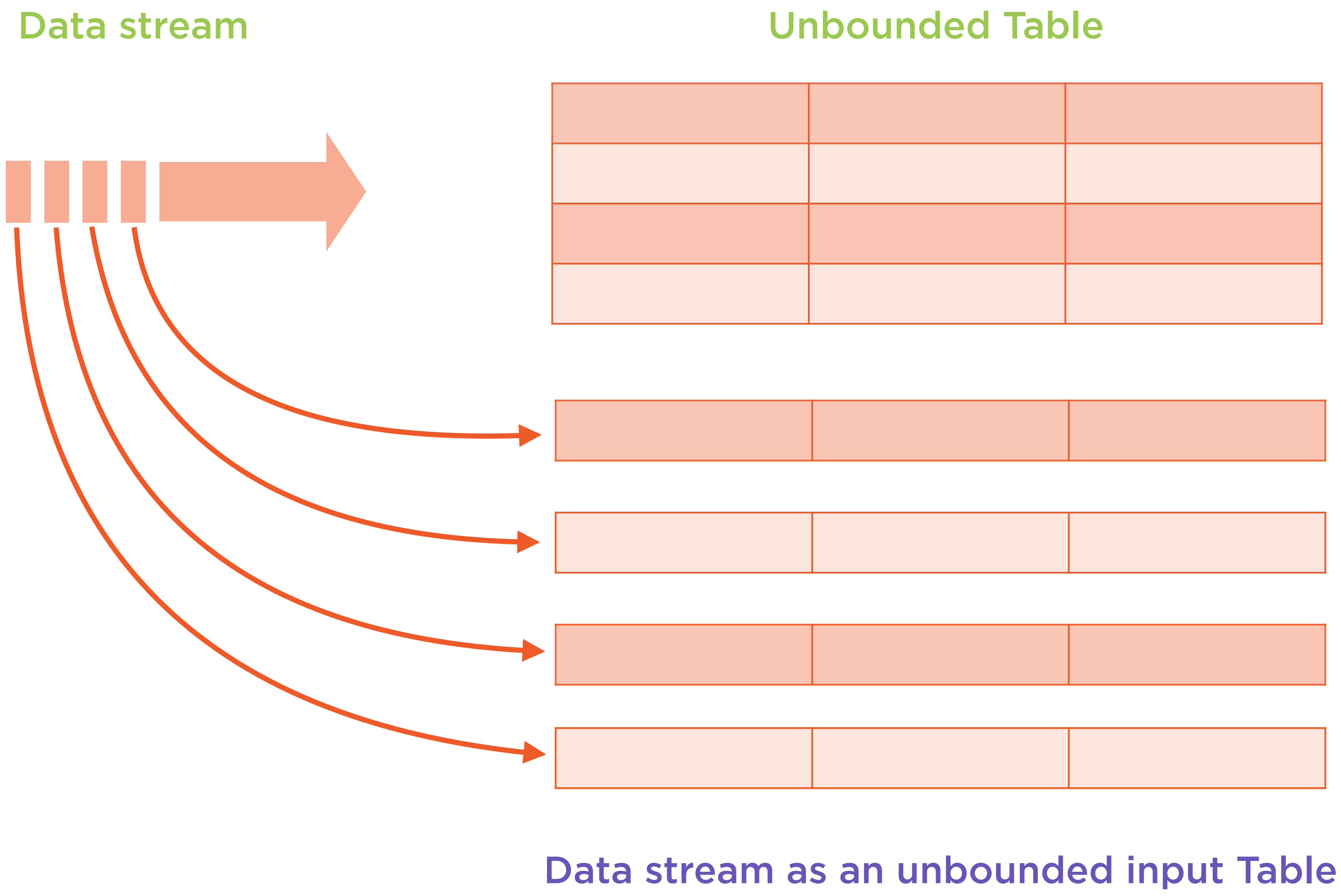
Every data item that is arriving on the stream is like a new row being appended to the input table

Batch is Simply Prefix of Stream



In other words, the input table (batch) is simply a prefix of the stream

Batch is Simply Prefix of Stream



All operations that can be performed on data frames can be performed on the stream

Structured Streaming treats a live data stream as a table that is being **continuously** appended

Burden of stream-processing shifts from user to system

Streaming and Structured Streaming

Streaming	Structured Streaming
Older	Newer
RDDs	DataFrames
No optimizations	Optimizations on DataFrames
Batch and streaming support not unified	Unified support for batch and streaming

Demo

Installations for this course

- Spark 2
- Kafka

Python libraries

- Tweepy
- Pykafka
- Afinn

Structured Streaming in Spark 2.x

Spark Streaming

What

A high-level API that takes burden off user

How

Micro-batch processing with exactly-once fault-tolerance

Why

Code virtually identical for batch and streaming

Spark Streaming

What

A high-level API that takes burden off user

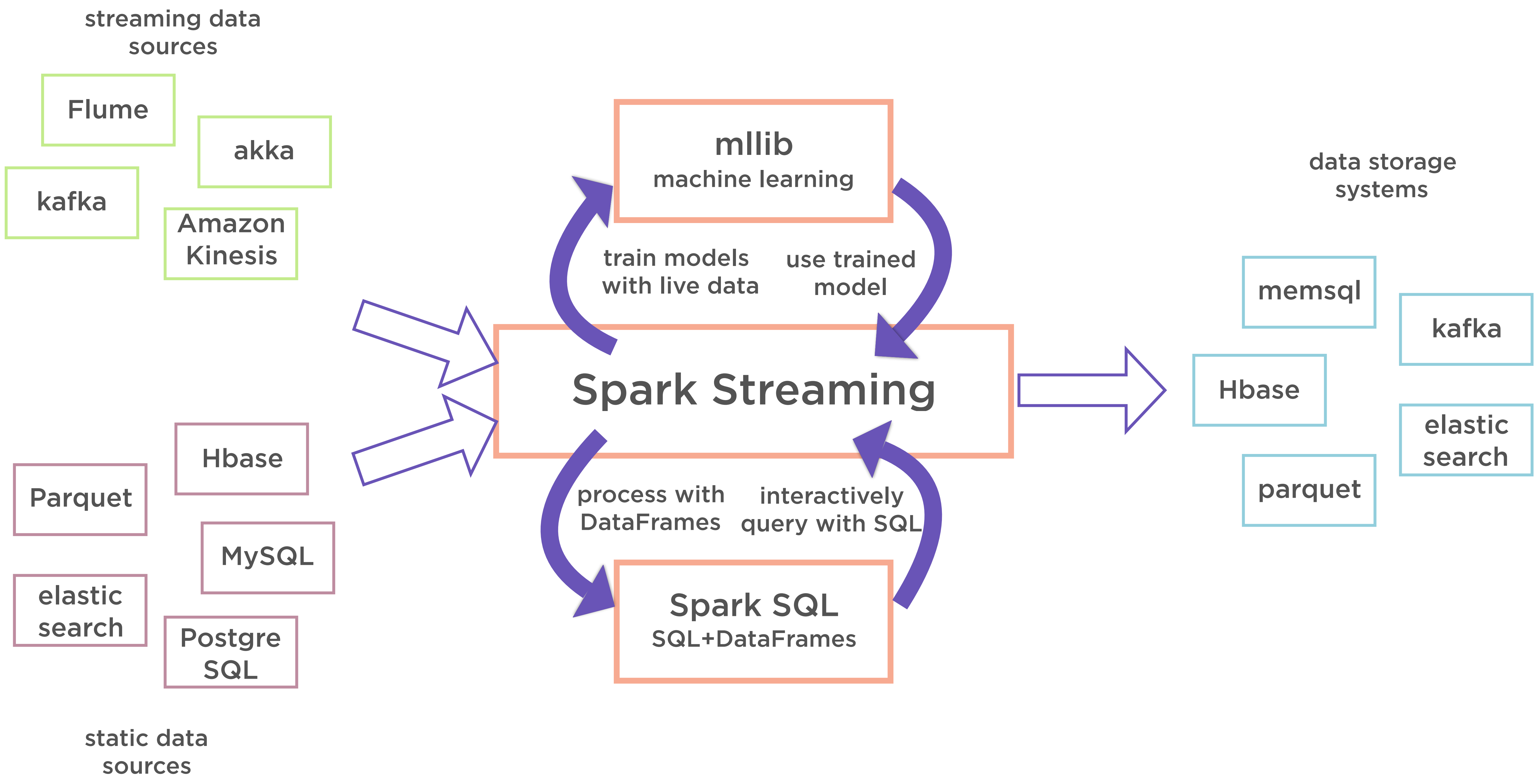
How

Micro-batch processing with exactly-once fault-tolerance

Why

Code virtually identical for batch and streaming

Spark 1.x: Spark Streaming



Continuous Application

End-to-end application that reacts to data in real-time

https://databricks.com/blog/2016/07/28/continuous-applications-evolving-streaming-in-apache-spark-2-0.html

Continuous Applications

A stream is not a stream

- It is simply an unbounded dataset

End-to-end application

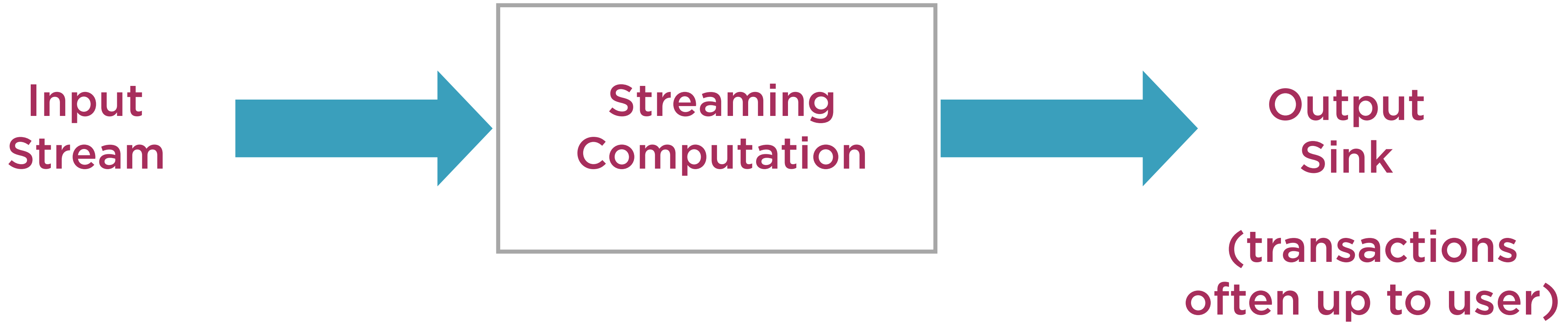
- Unify batch and streaming pipelines
- With same queries for both

Structured Streaming

New high-level API in Apache Spark 2.0 that supports continuous applications

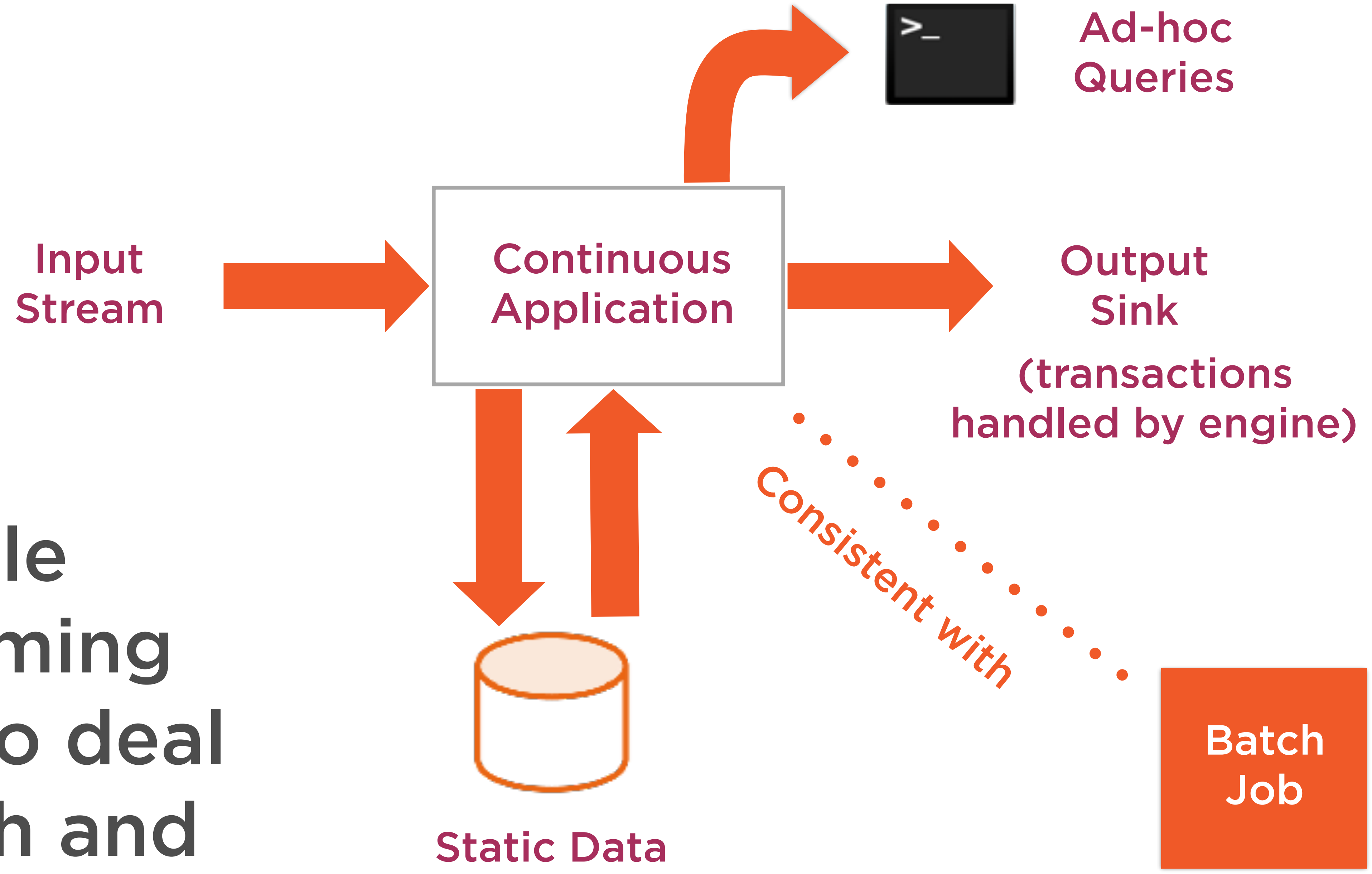
https://databricks.com/blog/2016/07/28/continuous-applications-evolving-streaming-in-apache-spark-2-0.html

Pure Streaming System



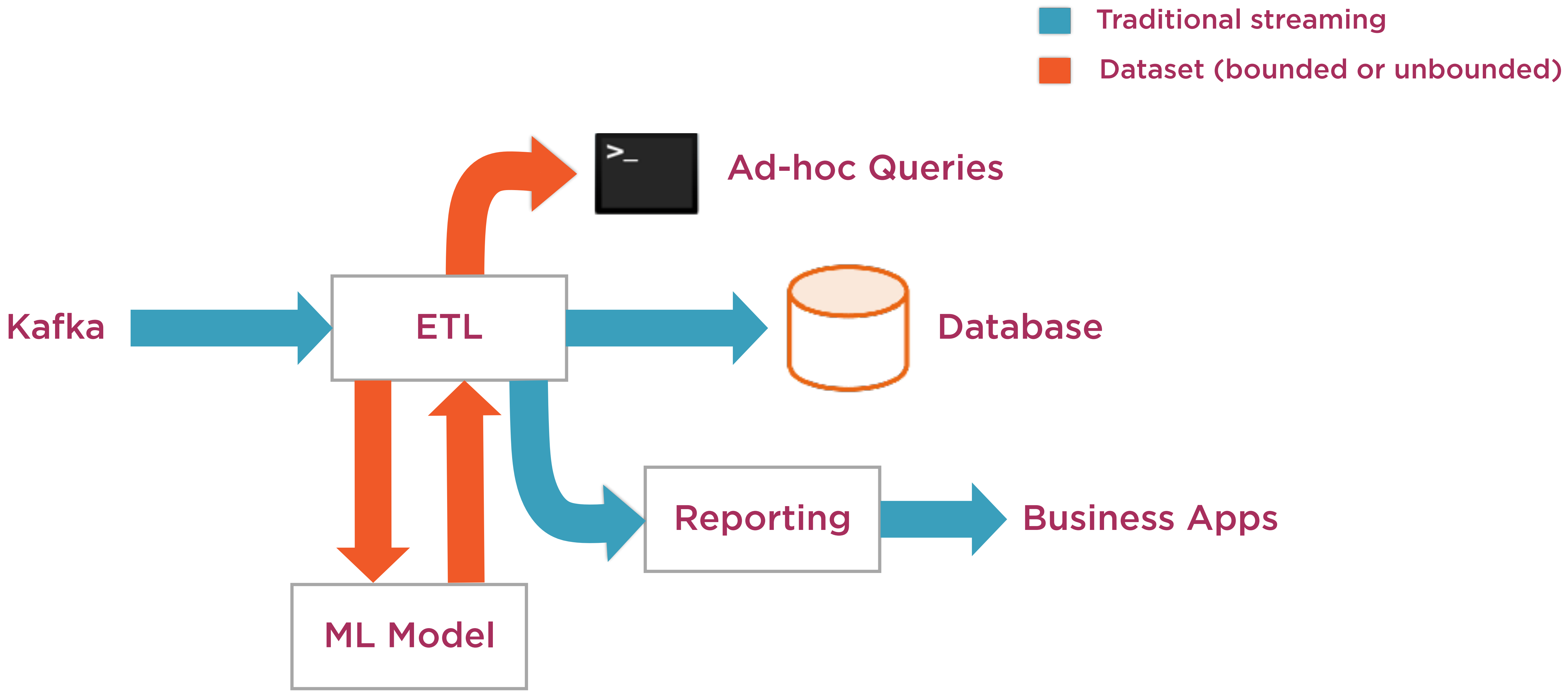
Interactions with other systems are completely up to the user

Continuous Applications

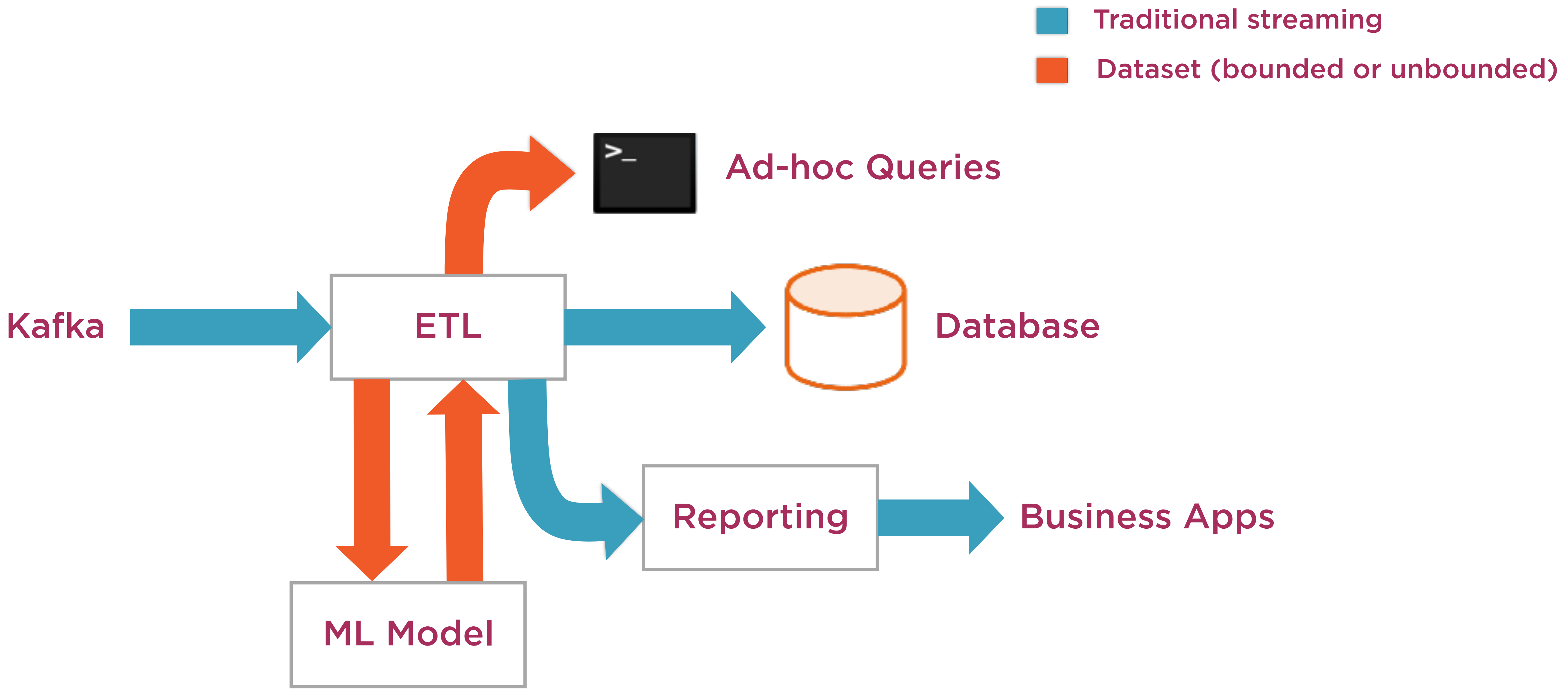


A single programming interface to deal with batch and realtime jobs

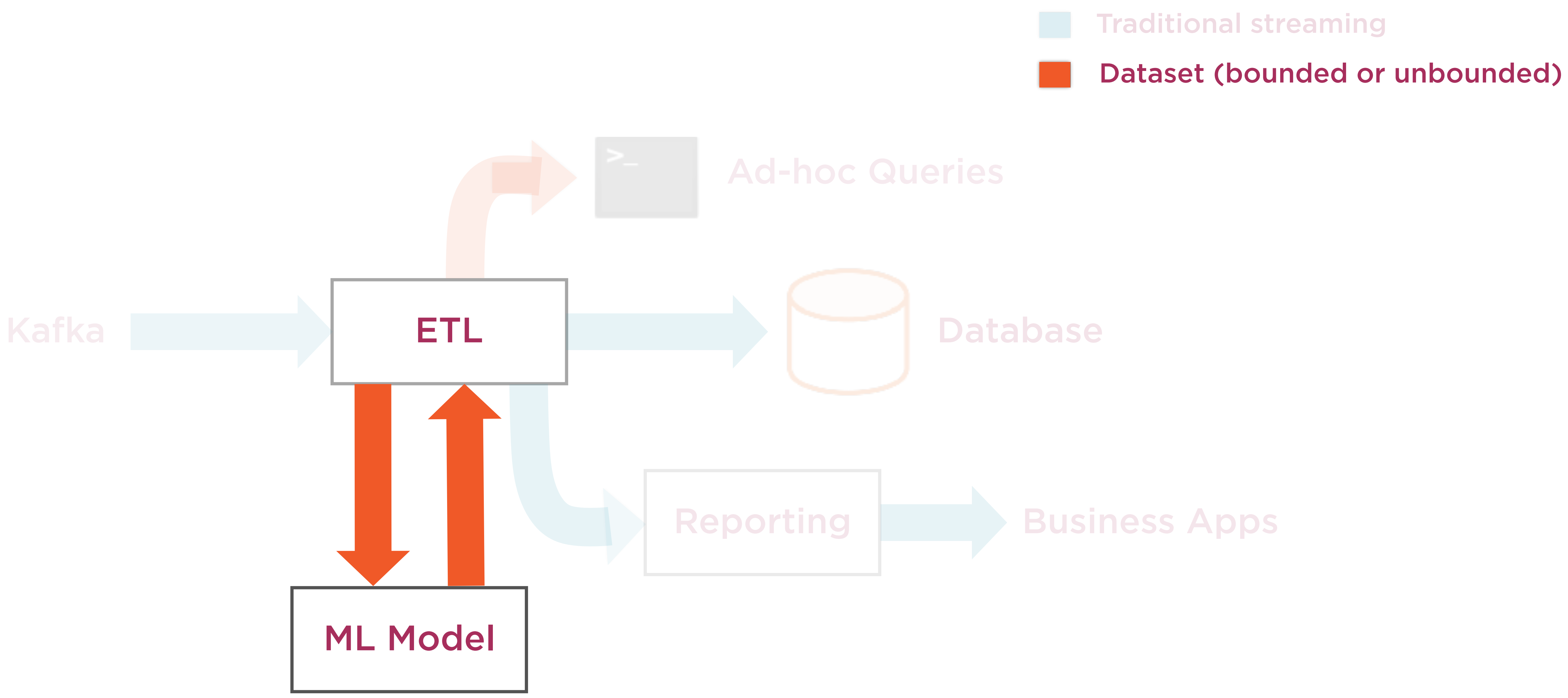
Spark 2.x: Continuous Applications



Structured Streaming

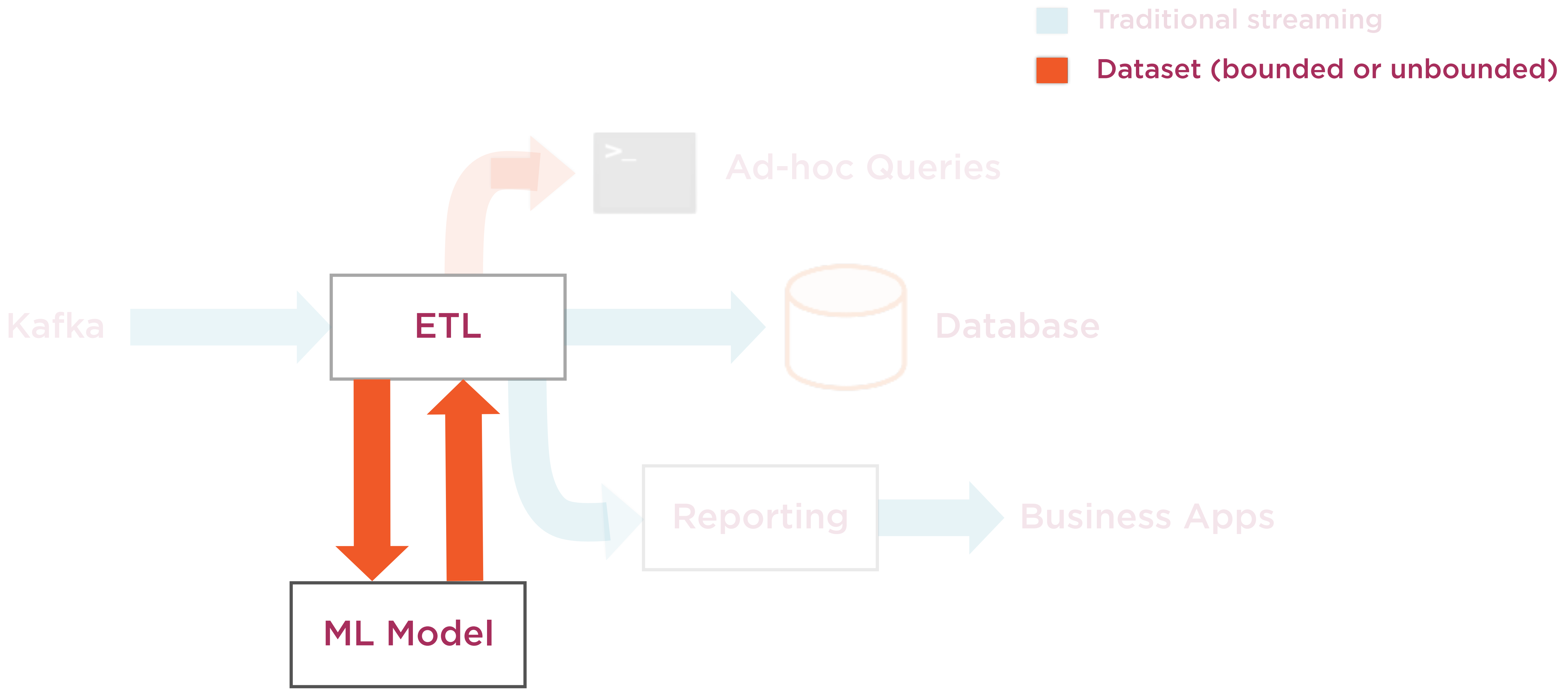


High-level User API



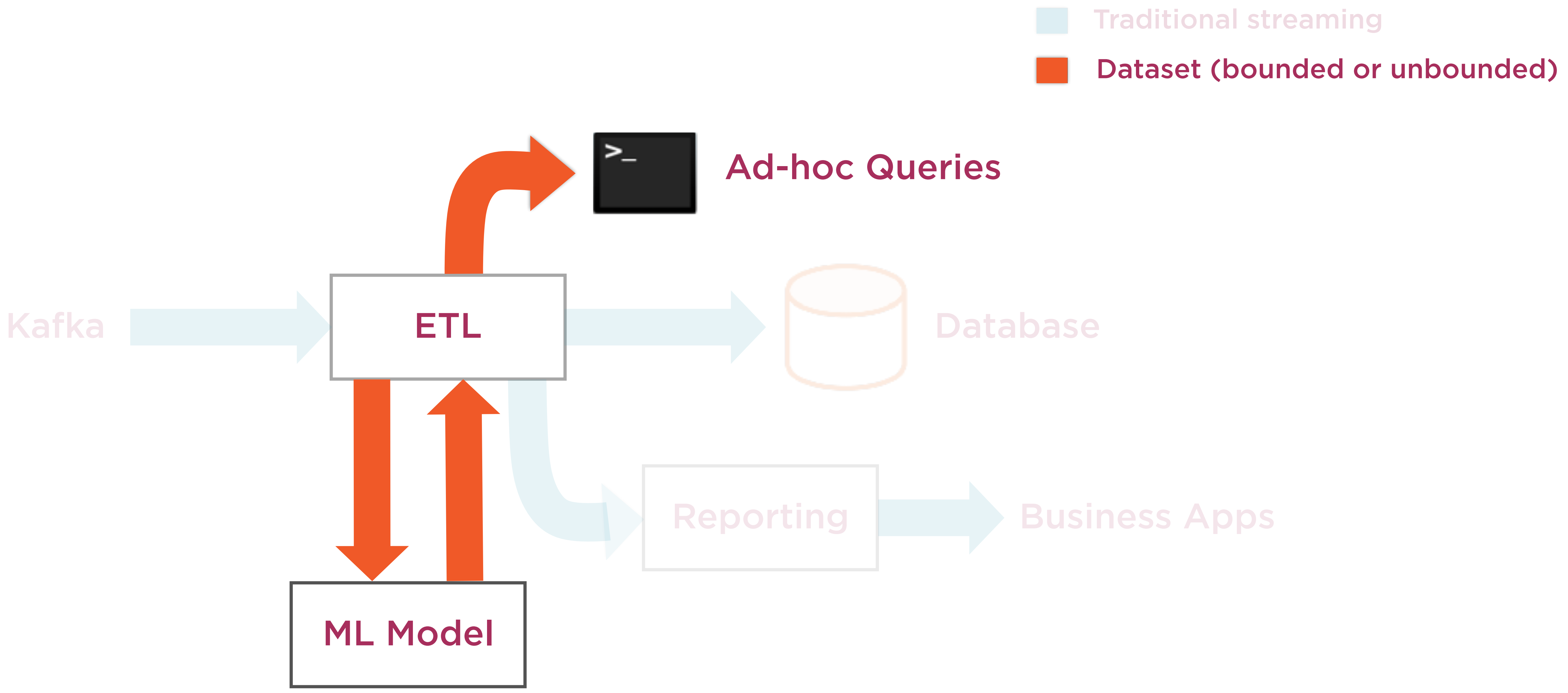
User implements batch computation using DataFrame/Dataset API

Automatic Support for Continuous Apps



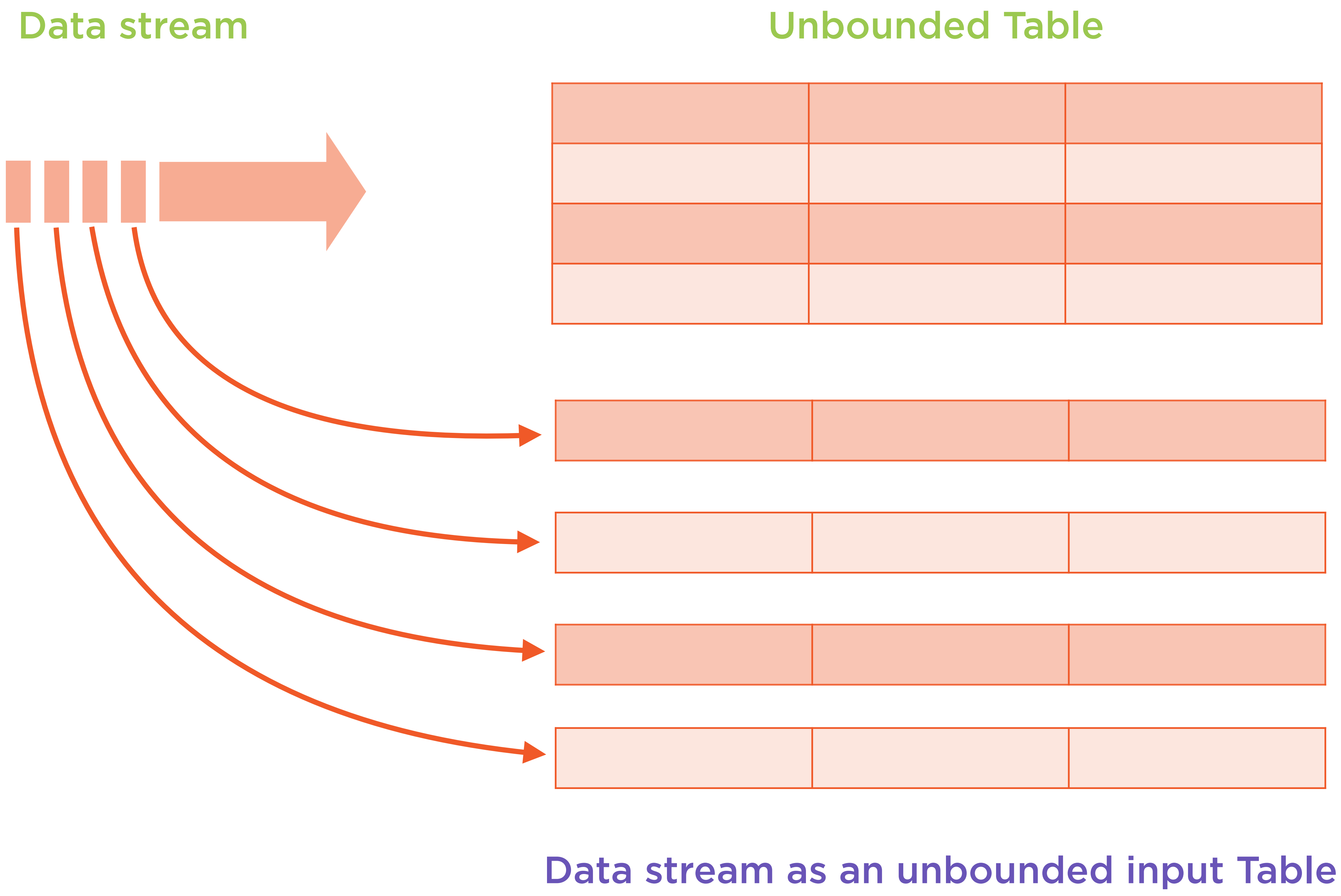
Spark automatically incrementalizes the batch computation

Automatic Support for Continuous Apps



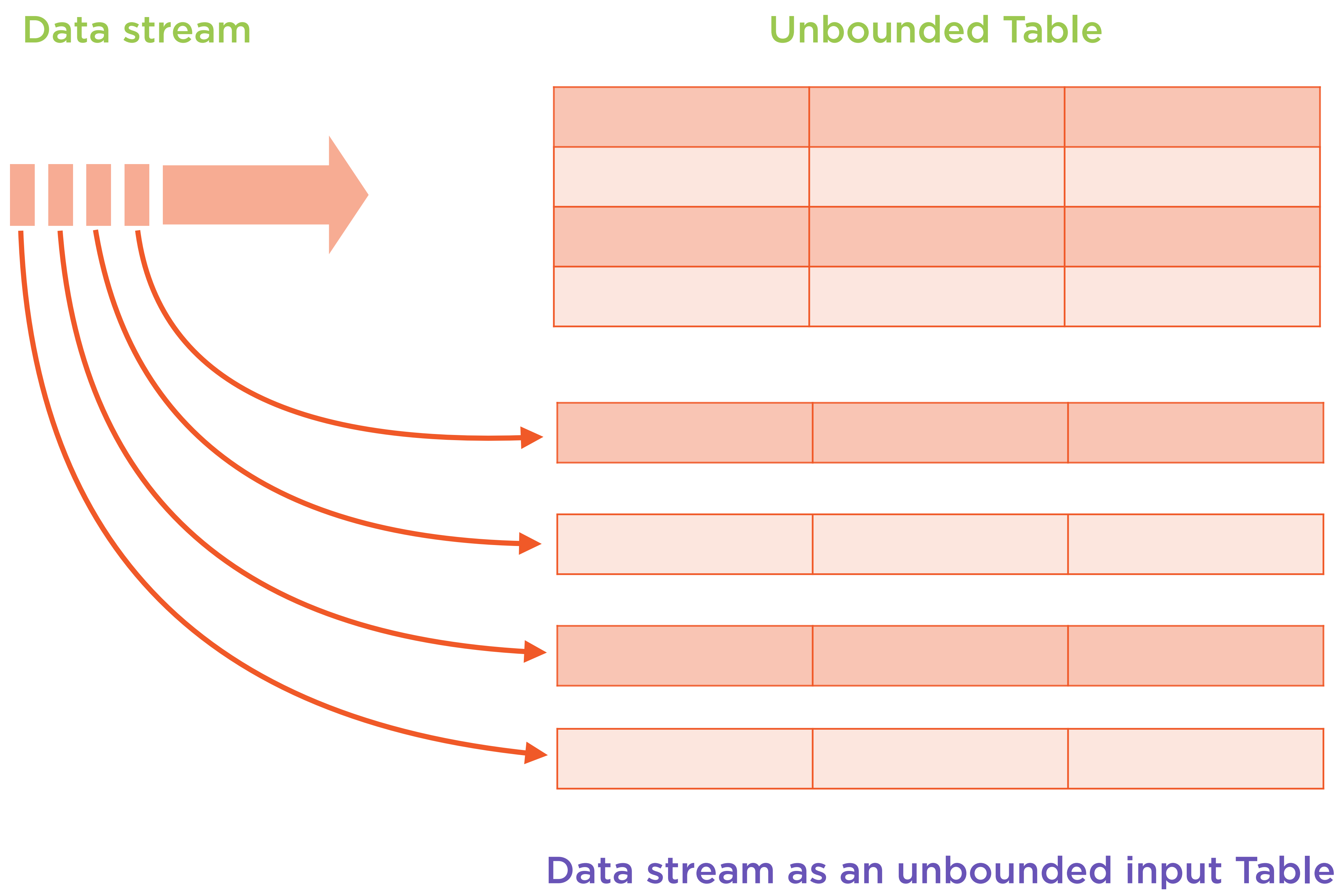
i.e. Spark automatically converts the job from batch to continuous

Batch Is Simply Prefix of Stream



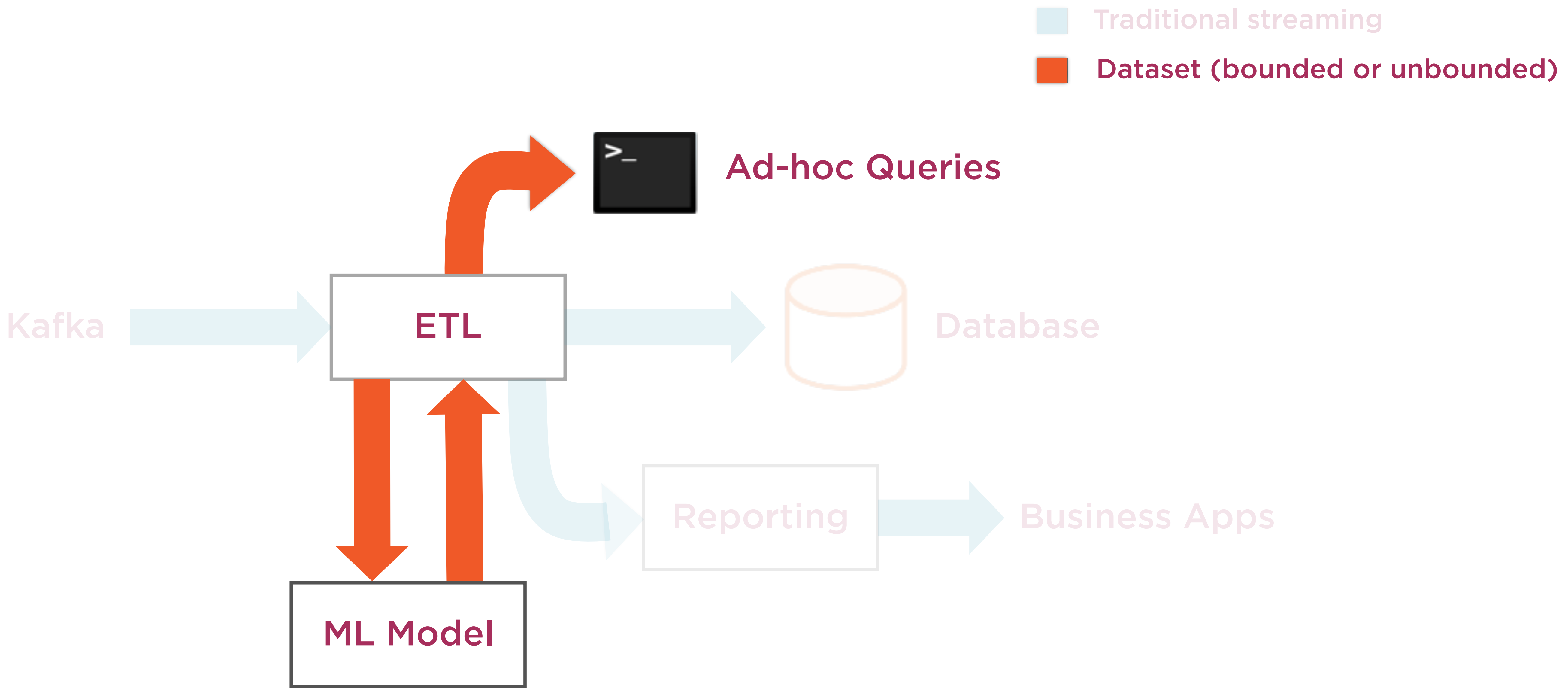
Every data item that is arriving on the stream is like a new row being appended to the input table

Batch Is Simply Prefix of Stream



In other words, the input table (batch) is simply a prefix of the stream

Batch Is Simply Prefix of Stream



Output of Structured Streaming job is same as running batch job on prefix (subset) of data

Prefix Integrity

Running job on continuous data yields same result as running job on batch data (where the batch is a prefix or snapshot of continuous data)

Prefix Integrity

Running job on continuous data yields **same result** as running job on batch data (where the batch is a prefix or snapshot of continuous data)

Prefix Integrity

Running job on continuous data yields same result as running job on batch data (where the batch is a prefix or snapshot of continuous data)

Structured Streaming treats a live data stream as a table that is being continuously appended

Burden of stream-processing shifts from user to system

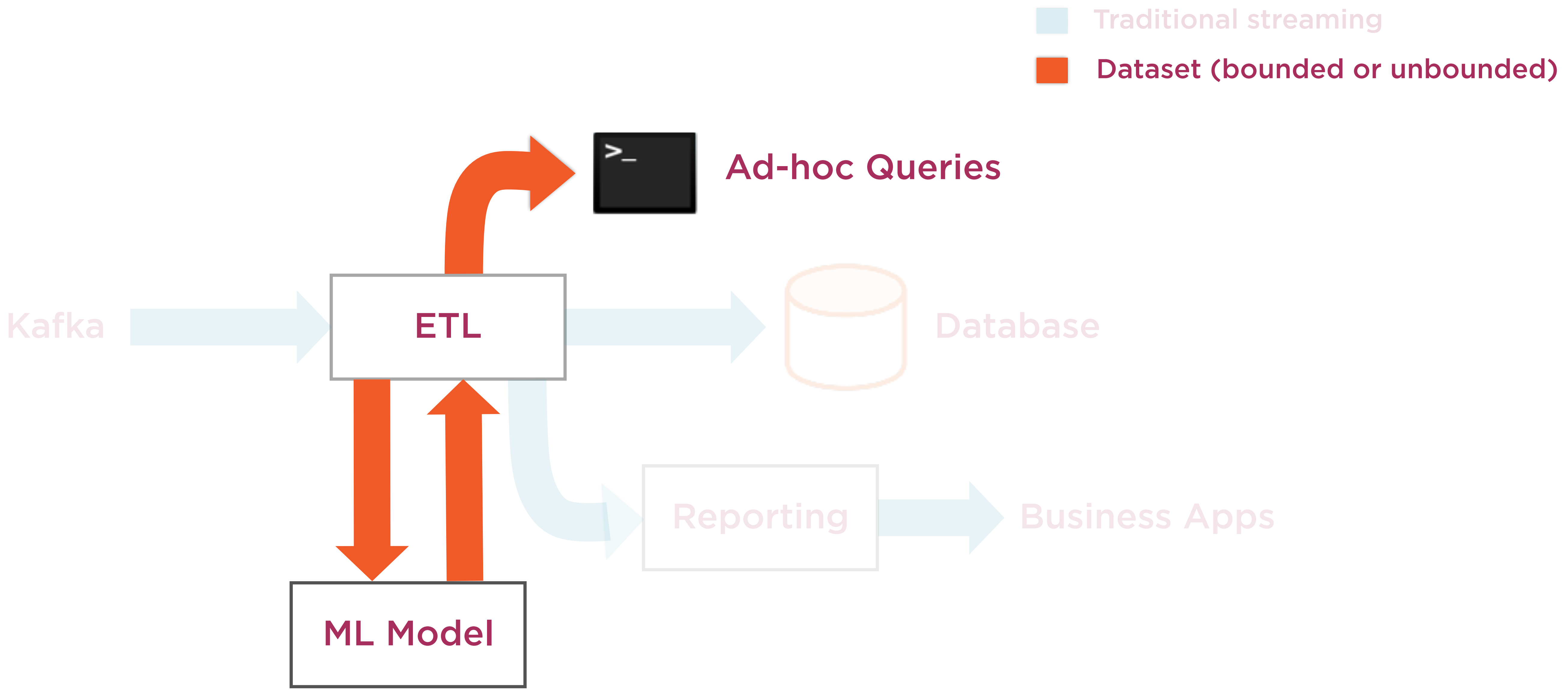
Structured Streaming

Structured Streaming maintains prefix integrity

Most other technologies do not

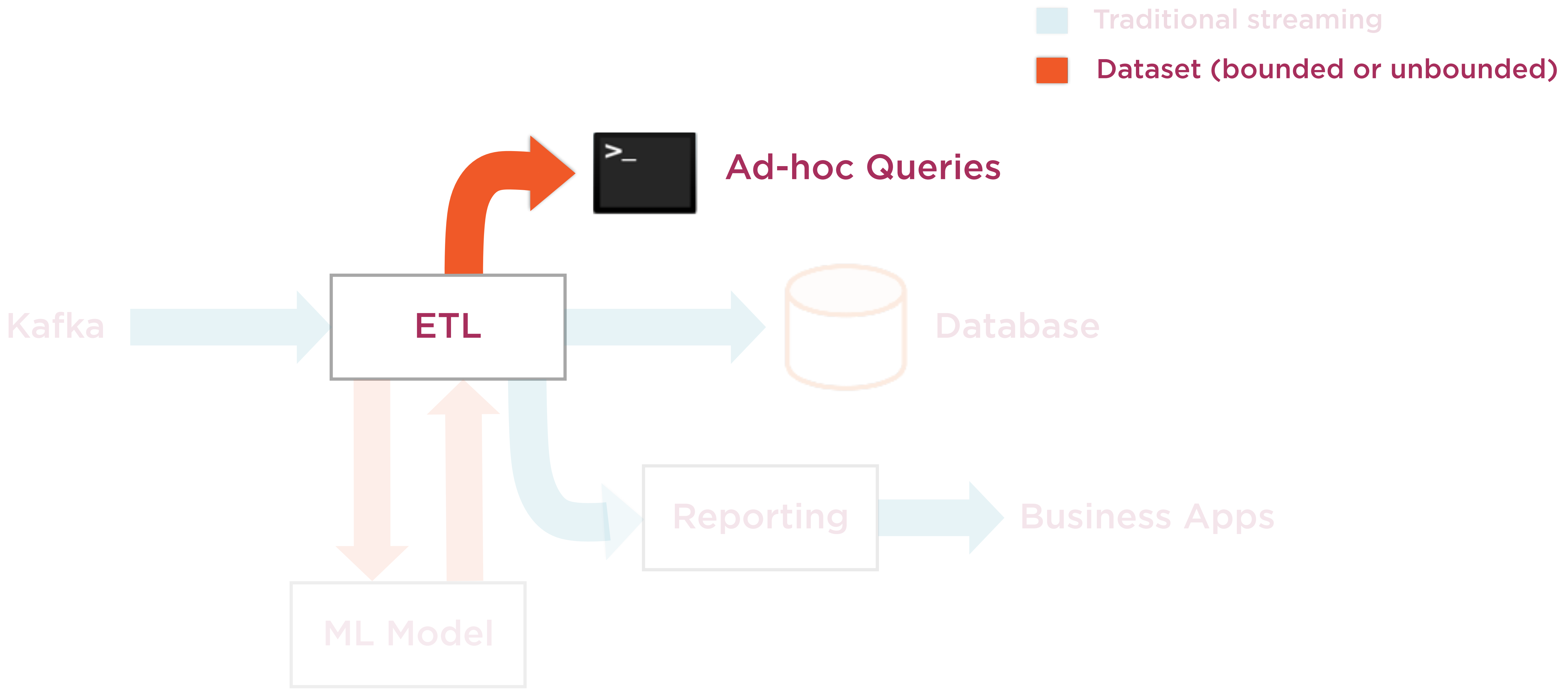
- Apache Flink
- Apache Kafka
- Apache Storm
- Google Dataflow

Tight Spark Integration



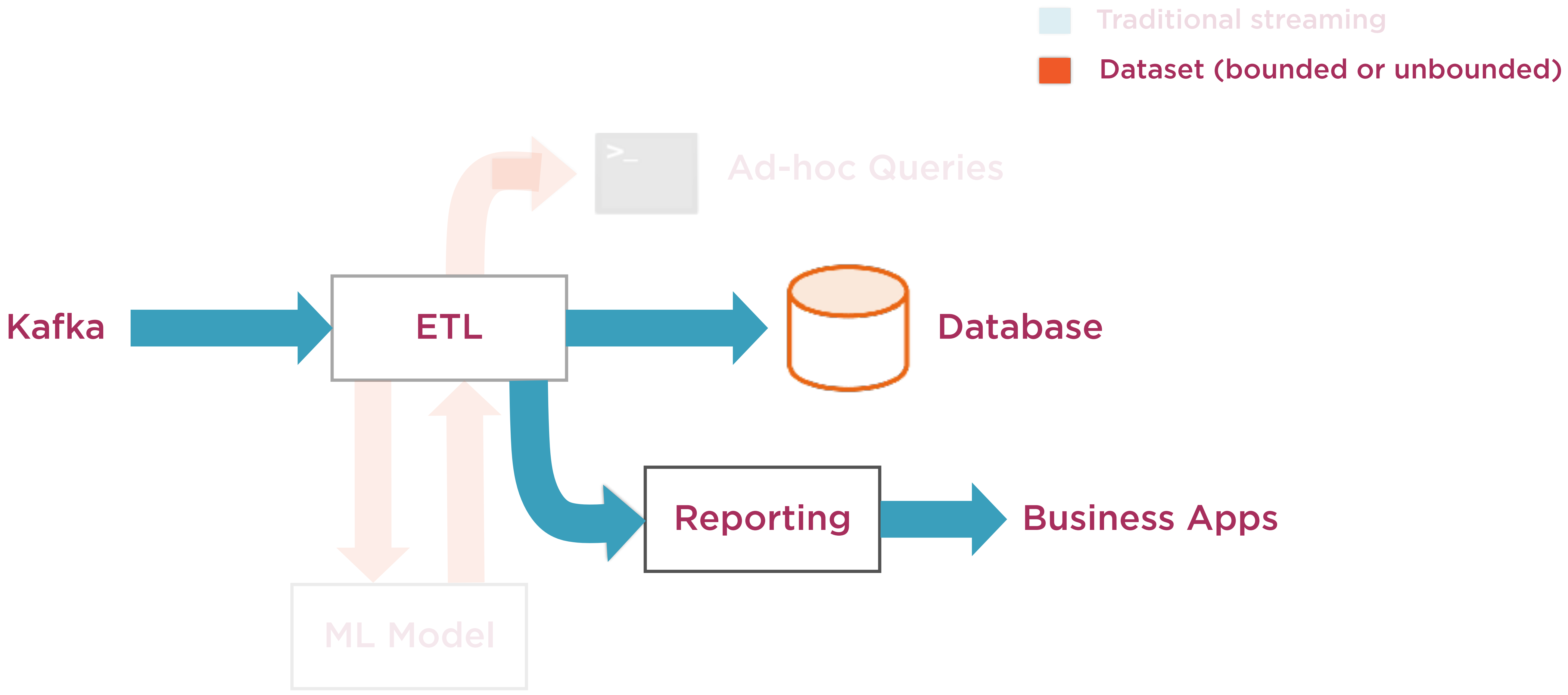
Structured Streaming retains tight integration with rest of Spark

Tight Spark Integration



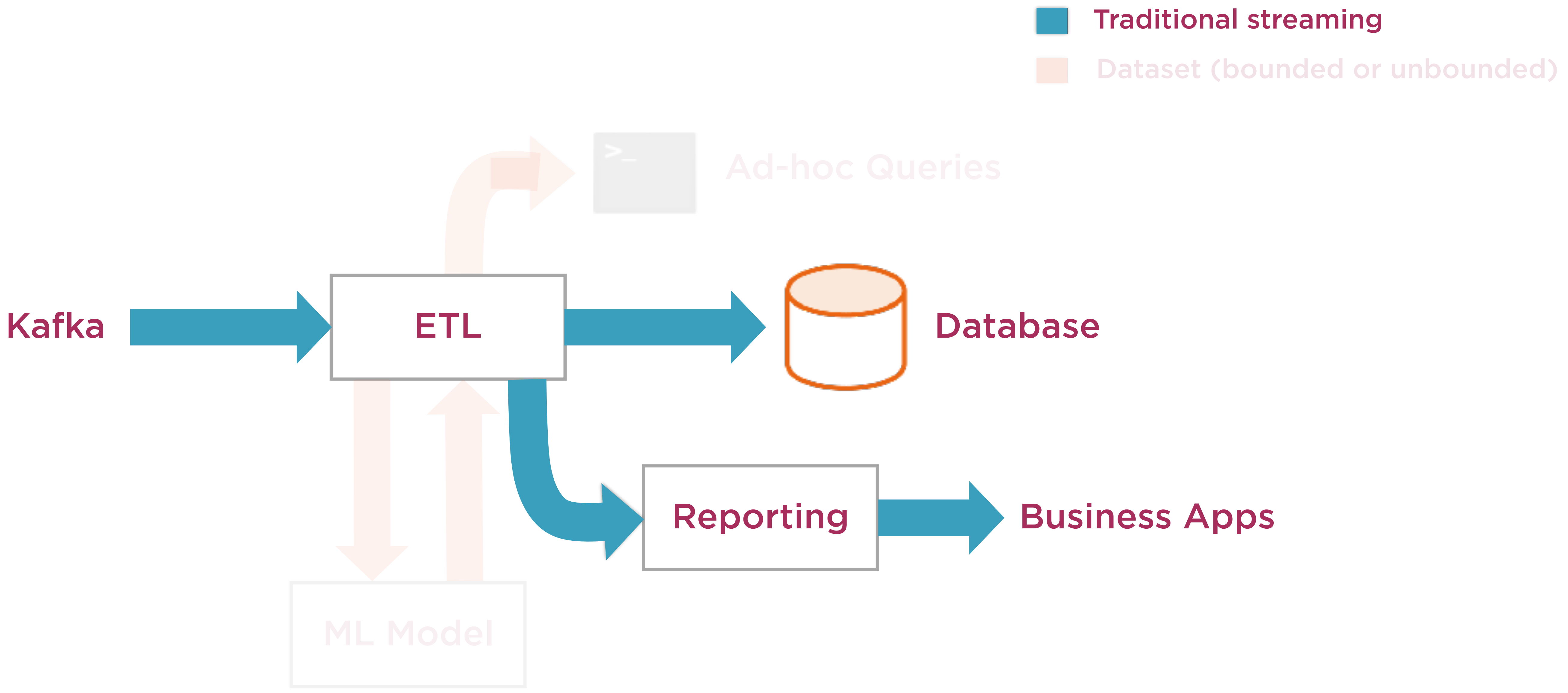
Serves interactive queries on the streaming state

Tight Storage Integration



Also maintains transactional integration with traditional storage

Spark 2.x: Continuous Applications



Combines advantages of traditional streaming with high-level API

A stream is just an unbounded dataset

Spark Streaming

What

A high-level API that takes burden off user

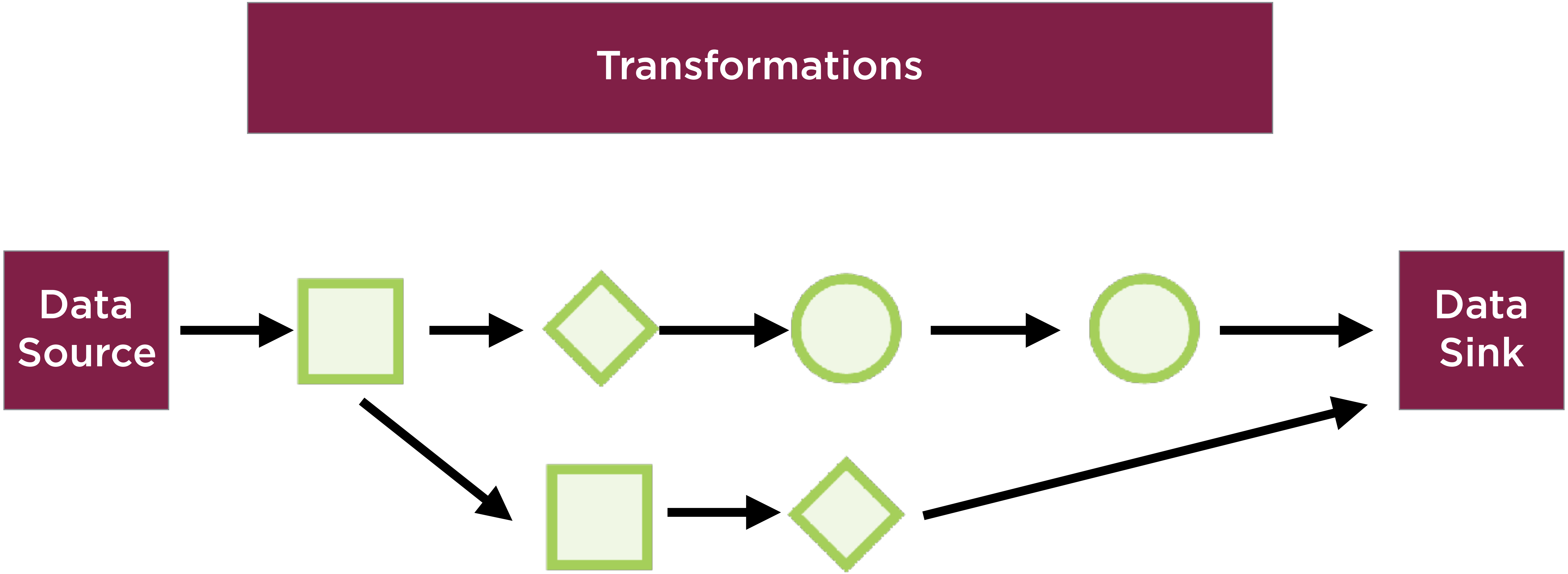
How

Micro-batch processing with exactly-once fault-tolerance

Why

Code virtually identical for batch and streaming

Stream Processing Model



Trigger

Events that determine when transformations on accumulated input data need to be re-performed. Each trigger event emits new data into the Result Table

Trigger

Events that determine when transformations on accumulated input data need to be re-performed. Each trigger event emits new data into the Result Table

Trigger

Events that determine when transformations on accumulated input data need to be re-performed. Each trigger event emits new data into the Result Table

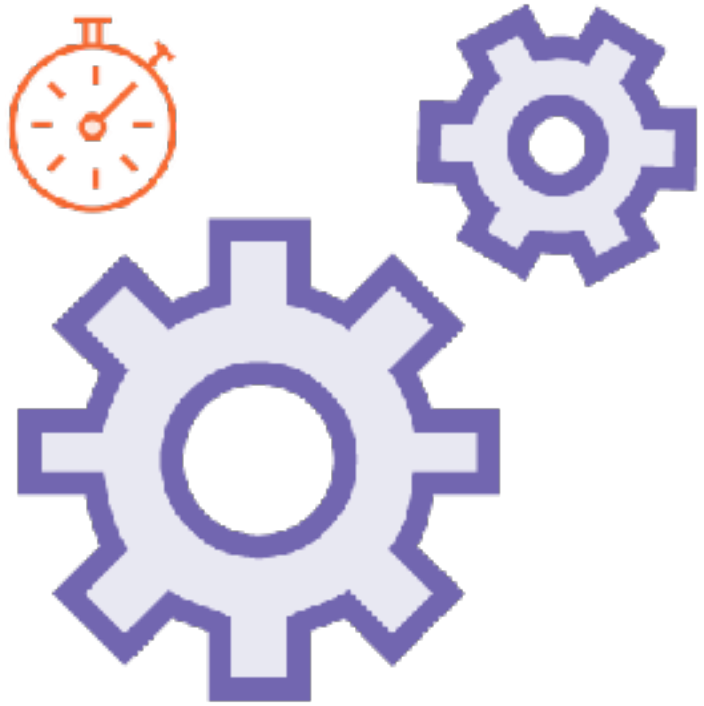
Trigger

Events that determine when transformations on accumulated input data need to be re-performed. Each trigger event emits new data into the Result Table

Time

Event Time

Ingestion Time



Processing Time

Two Types of Triggers

Processing time Triggers

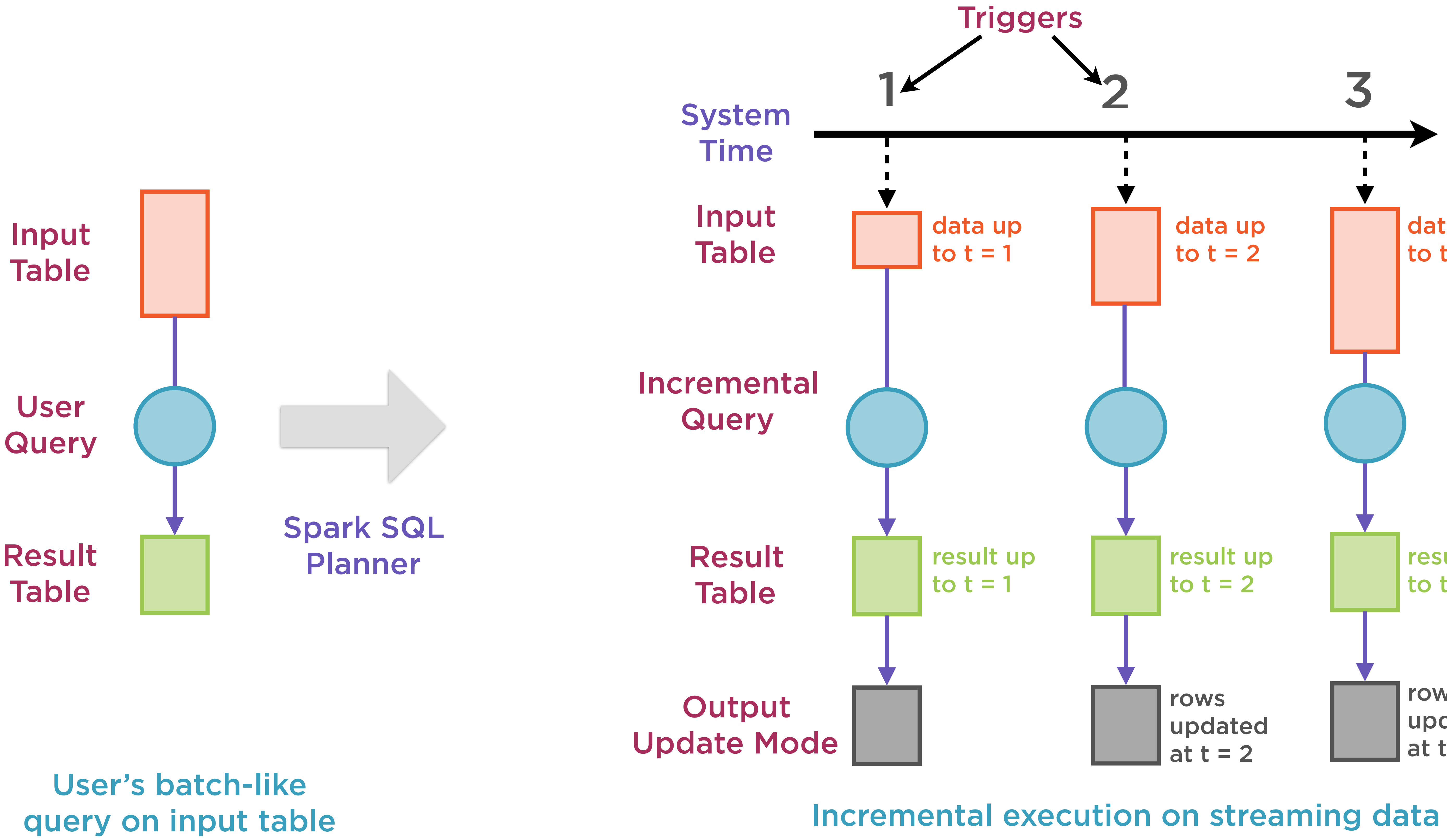
Result Table is recalculated at periodic processing time interval

One-time Trigger

Result Table only calculated once; any new data is ignored in result

Triggers rely on processing time not event time (unless watermarks used)

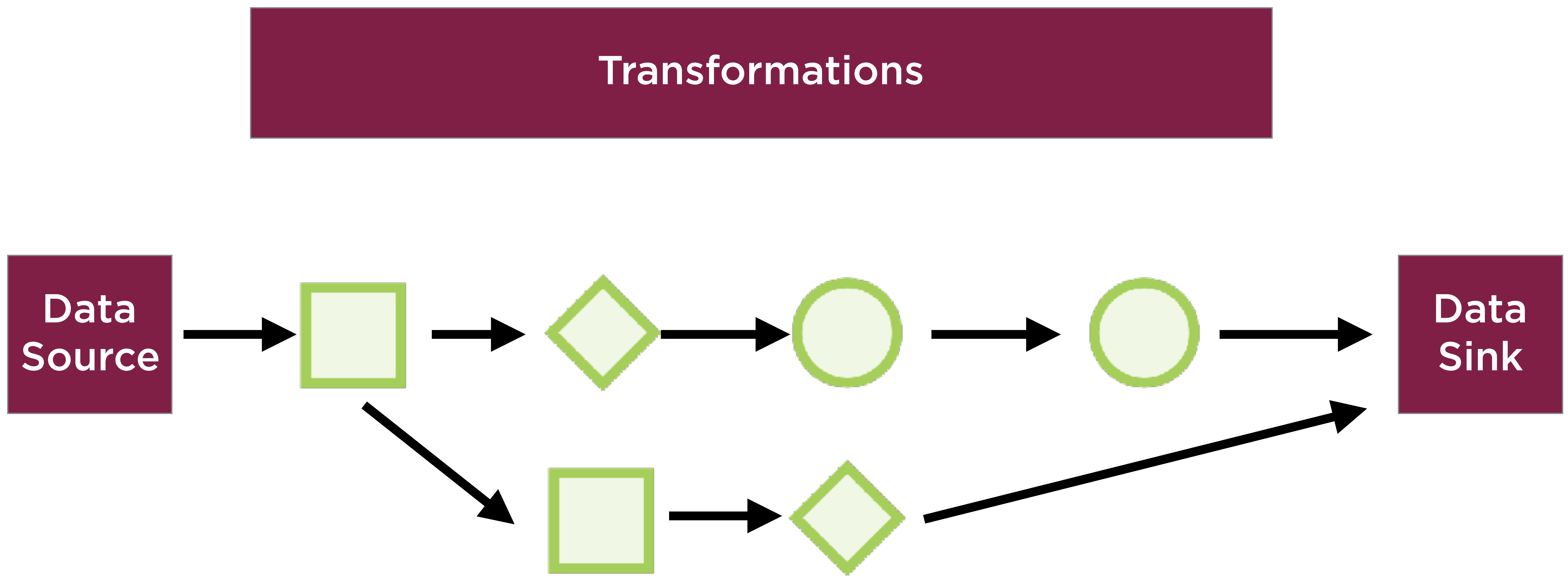
Result Table Is Generated



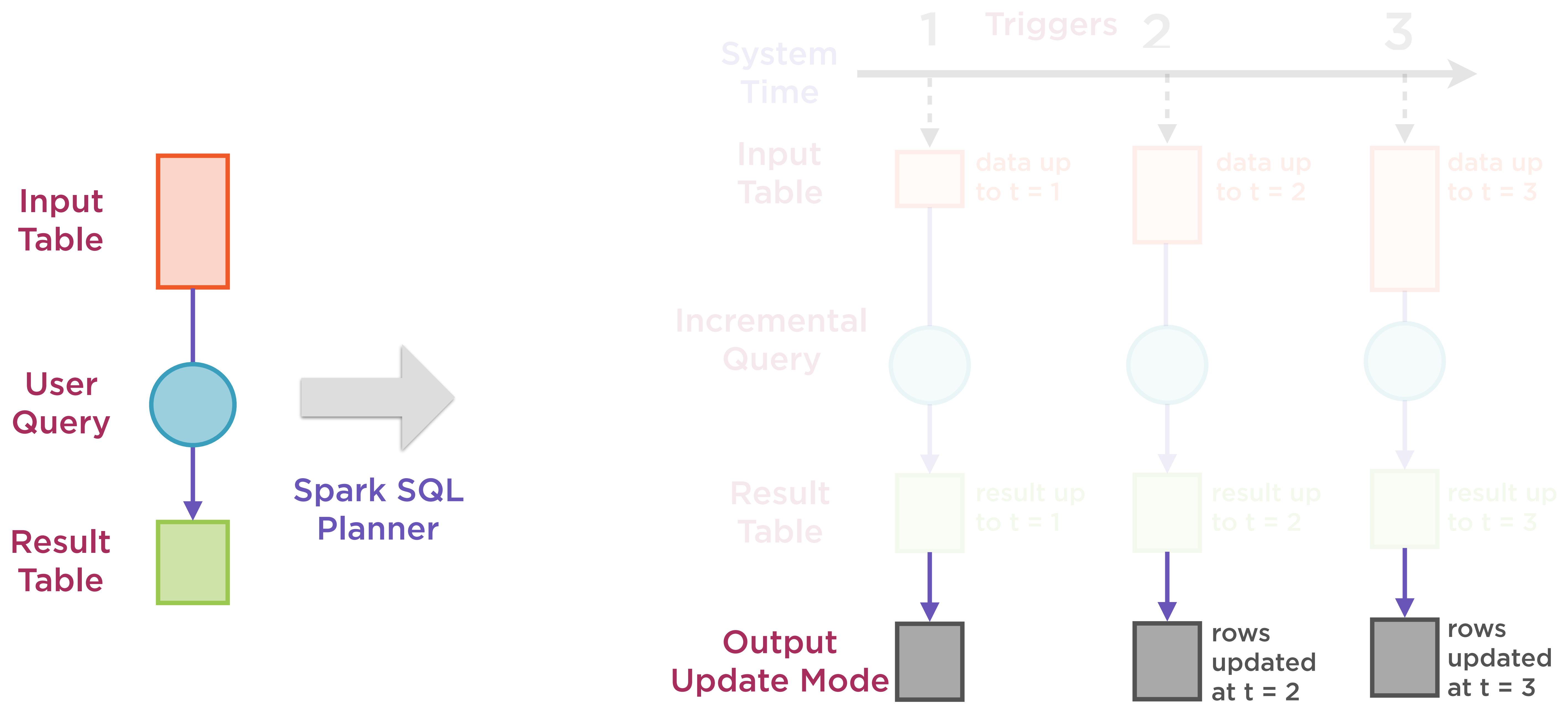
Every trigger, new rows from the input table are processed

These pass through the transformations in the query and update the results table

Stream Processing Model

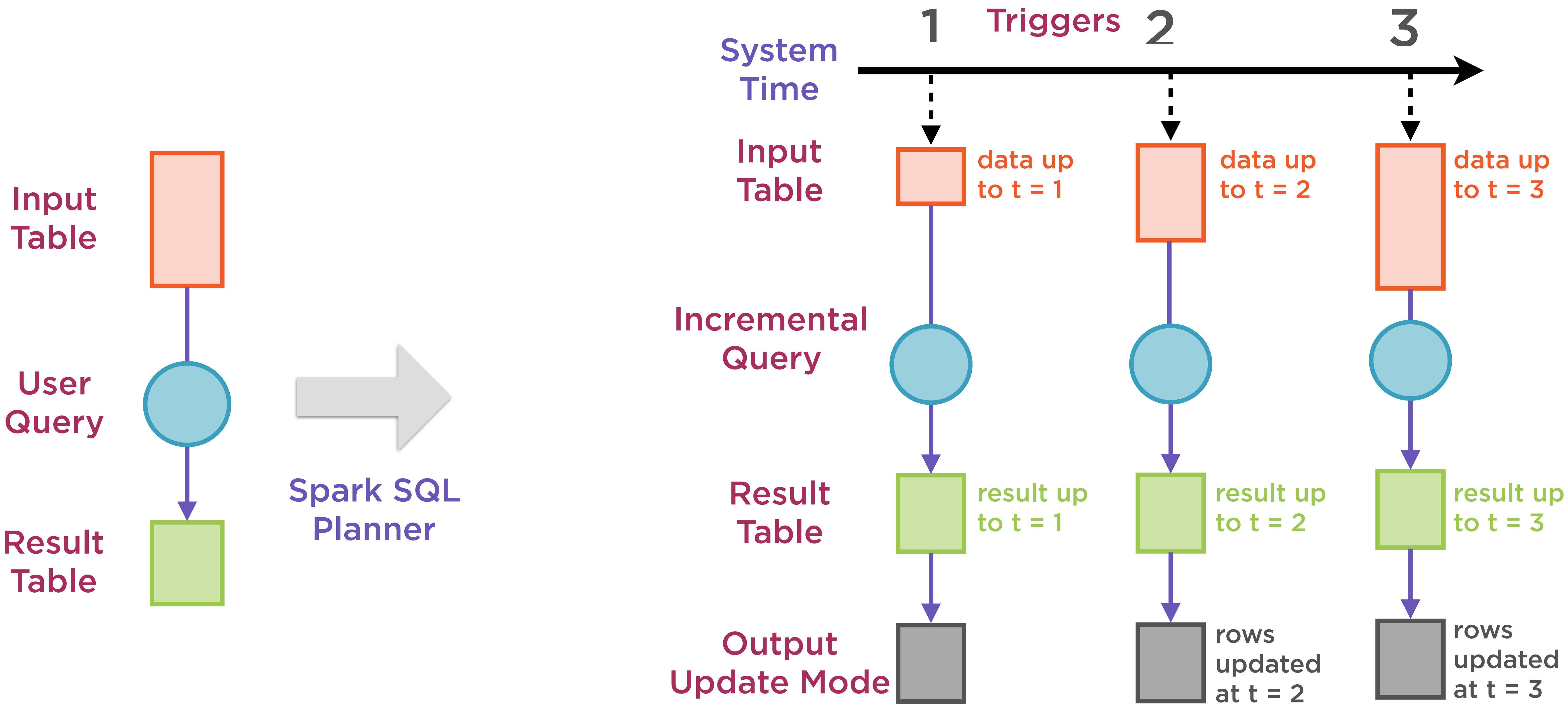


Result Table Is Generated



The Result Table is then written out to data sink

Result Table Is Generated



Changes to input table eventually result in changes to output table

Types of Sources

Data Source

File source

Kafka source

Socket source

Rate source

Types of Sinks

Data Sink

- File sink
- Kafka sink
- Foreach sink
- Console sink
- Memory sink

Data Sink

Materialization

Structured streaming reads data from source

- Processes incrementally
- Updates result
- Discards source

Only minimal intermediate state maintained

When writing to the sink the entire table is not materialized

What is written out depends on the mode

Data Sink

Output Modes

Determines what Result Table rows get sent to storage

- Update mode
- Append mode
- Complete mode

Output Modes

Update mode - only Result Table rows updated since last trigger

Even previous results will be updated in case of aggregations

Append mode - only Result Table rows appended since last trigger

Previous (existing) output rows cannot change

Complete mode - entire updated Result Table is sent across

Storage connector must decide how to use all that data

Spark Streaming

What

A high-level API that takes burden off user

How

Micro-batch processing with exactly-once fault-tolerance

Why

Code virtually identical for batch and streaming

Structured Streaming

Batch and stream code virtually identical

Fault tolerance and exactly-once guarantees

Handles event-time and late data

Spark Streaming

What

A high-level API that takes burden off user

How

Micro-batch processing with exactly-once fault-tolerance

Why

Code virtually identical for batch and streaming

Comparison with Other Engines

Property	Structured Streaming	Spark Streaming	Apache Storm	Apache Flink	Kafka Streams	Google Dataflow
Streaming API	incrementalize batch queries	integrates with batch	separate from batch	separate from batch	separate from batch	integrates with batch
Prefix Integrity Guarantee	✓	✓	✗	✗	✗	✗
Internal Processing	exactly once	exactly once	at least once	exactly once	at least once	exactly once
Transactional Sources/Sinks	✓	some	some	some	✗	✗
Interactive Queries	✓	✓	✗	✗	✗	✗
Joins with Static Data	✓	✓	✗	✗	✗	✗

Demo

A simple streaming application

Count the number of times a word appears in streaming sentences

Summary

Understood attributes of stream processing applications

Identified differences between batch and stream architectures

Understood Spark DStreams and streaming in Spark 1.x

Introduced Structured Streaming

Streams are modeled as unbounded datasets