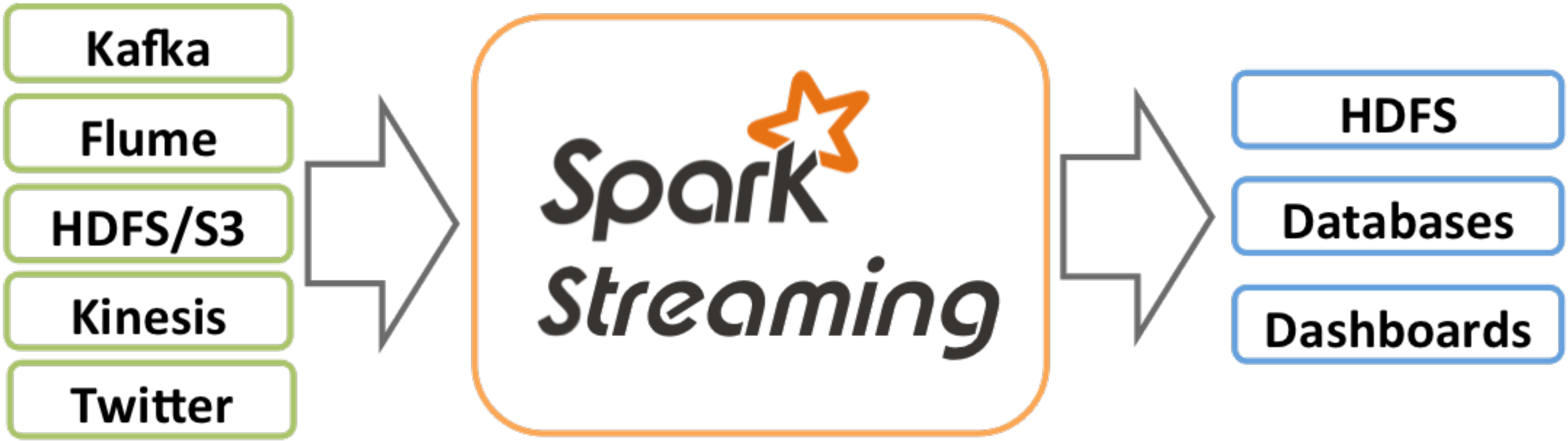


# Introduction to Spark Streaming

Surendra Panpaliya

# Agenda

- What is Spark streaming?
- High level streaming Architecture
- Spark streaming Sources
- Spark streaming's place in spark
- Spark structured streaming place in spark

# Agenda

- Dstream, Transformations
- Discretized Steams (DStreams)
- Receivers
- Batching
- Dstream processing

# Agenda

- Spark structured streaming
- How does it work?
- Steps for structured streaming
- Supported sources and sinks

# What is Spark Streaming?

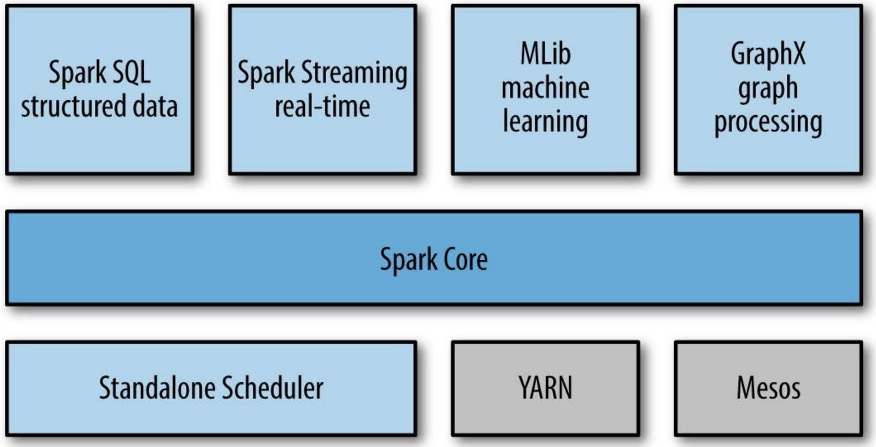
An extension of the core Spark API

enables scalable,

high-throughput,

fault-tolerant

stream processing of live data streams.



Spark SQL structured data

Spark Streaming real-time

MLib machine learning

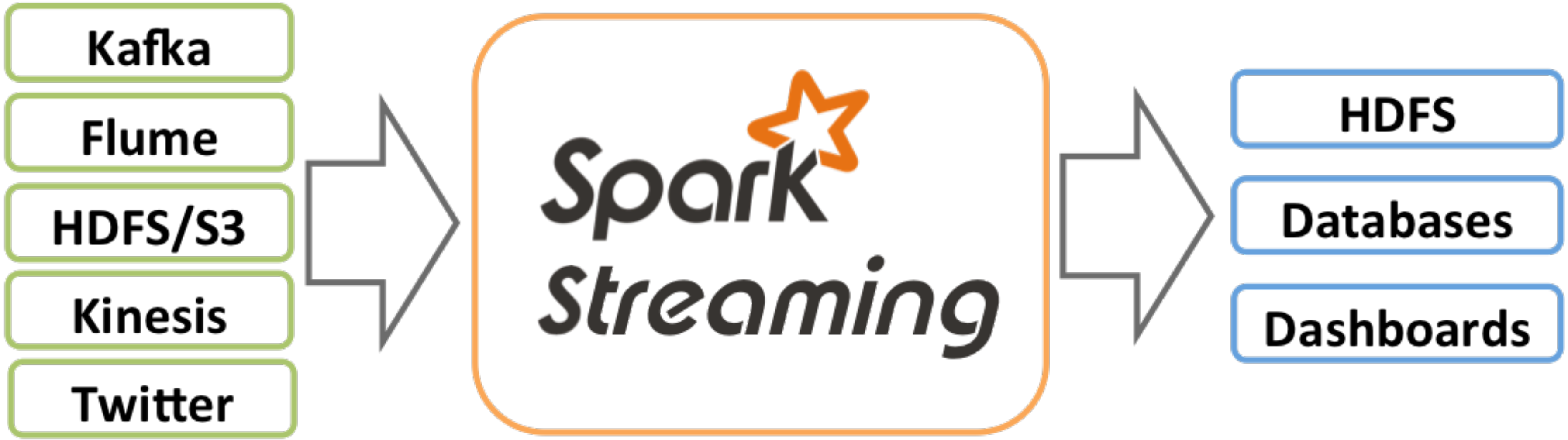
GraphX graph processing

Spark Core

Standalone Scheduler

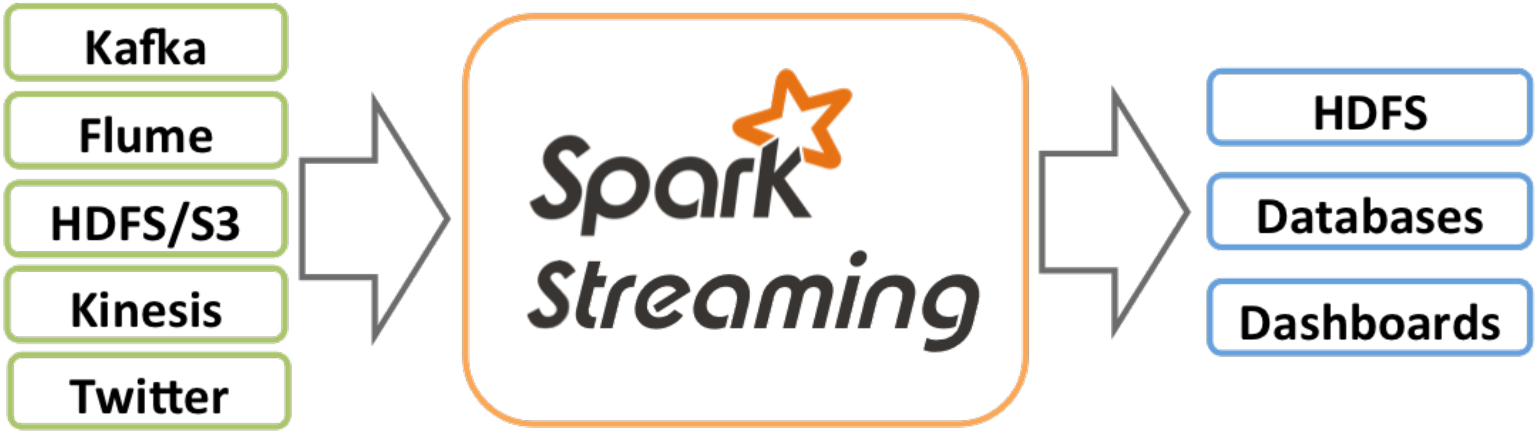
YARN

Mesos



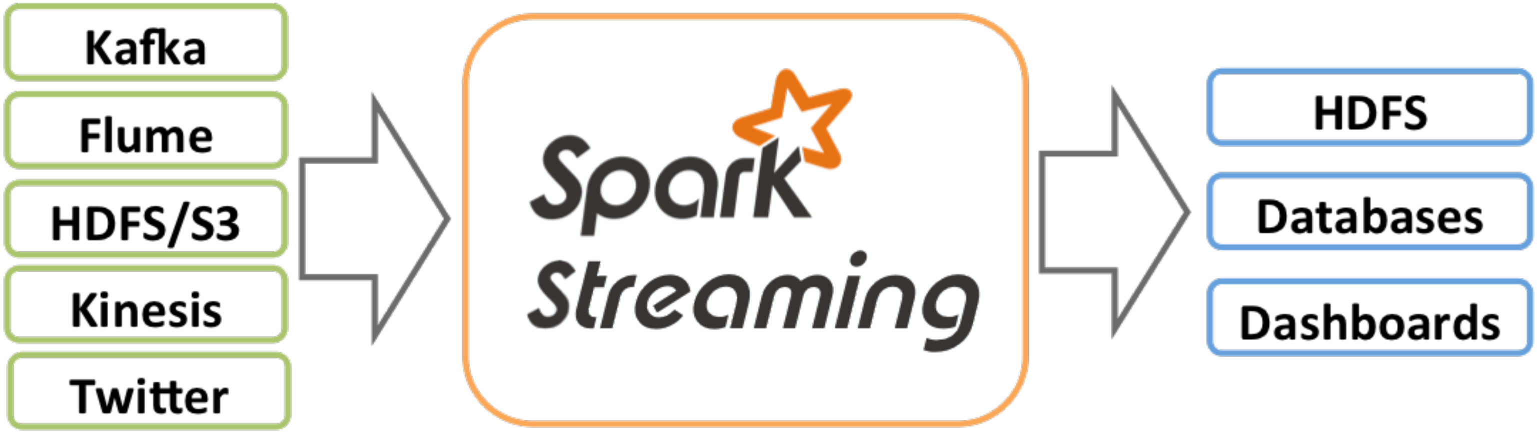
# What is Spark Streaming?

- Data can be ingested
- from many sources like
- Kafka, Kinesis,
- TCP sockets



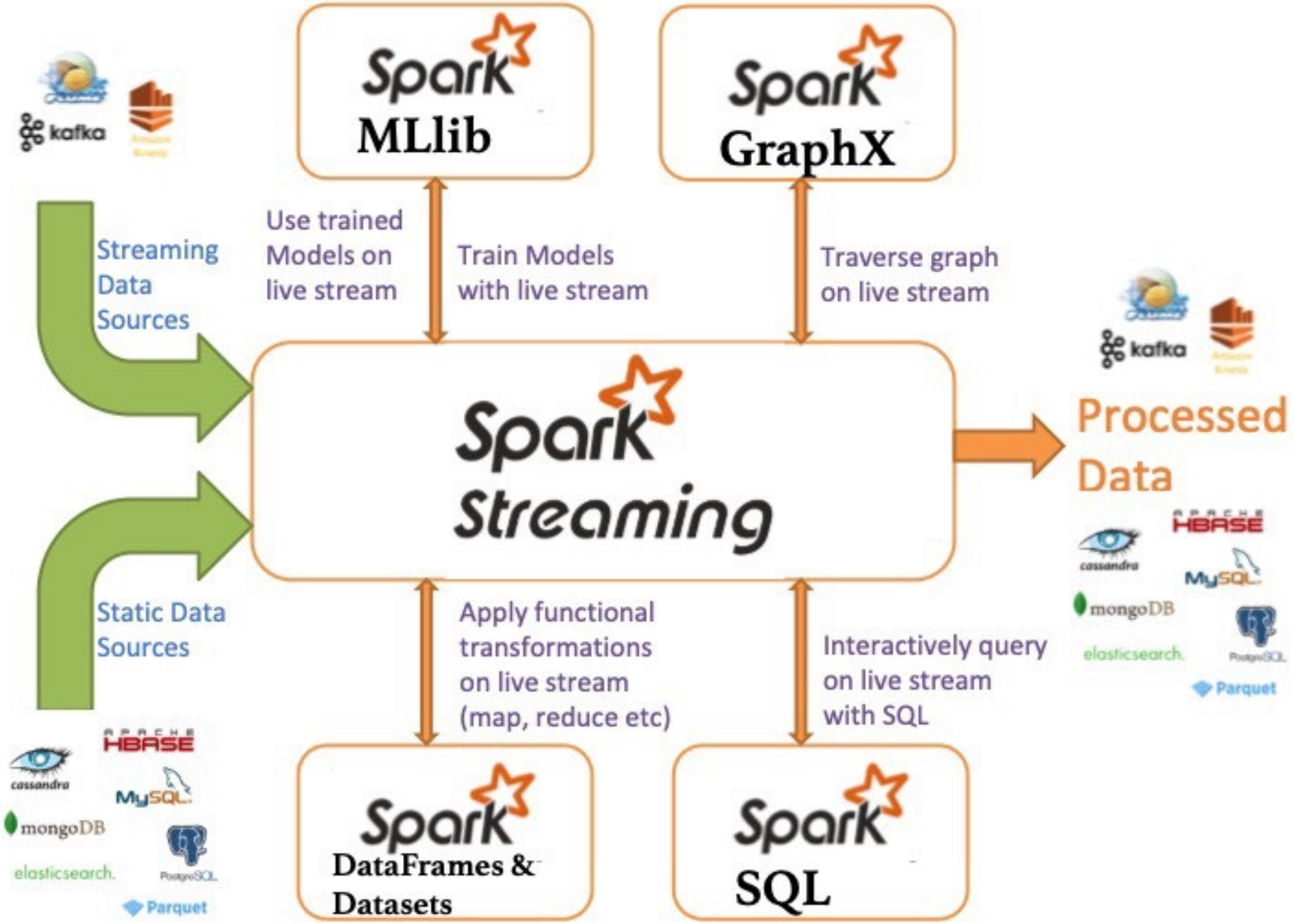
# What is Spark Streaming?

- Can be processed
- using complex algorithms
- with high-level functions like
- map, reduce, join and window.



# What is Spark Streaming?

- Finally, processed data
- can be pushed out to
- filesystems, databases, and
- live dashboards.



Spark MLlib

Use trained Models on live stream

Train Models with live stream

Spark GraphX

Traverse graph on live stream

Streaming Data Sources

Spark Streaming

Processed Data

Static Data Sources

Apply functional transformations on live stream (map, reduce etc)

Interactively query on live stream with SQL

Spark DataFrames & Datasets

Spark SQL

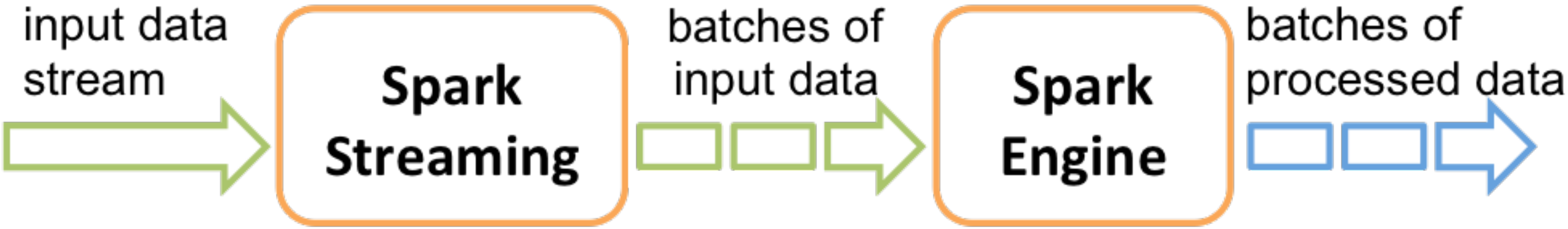
# What is Spark Streaming?

Can apply Spark’s

machine learning and

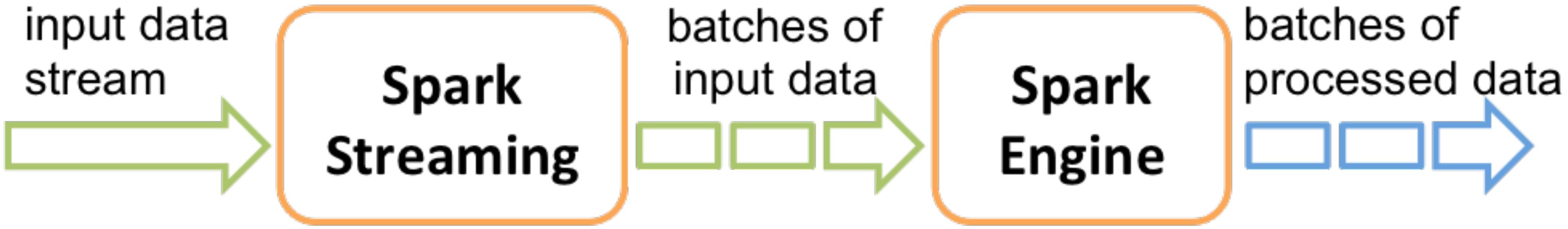
graph processing algorithms

on data streams.



# How it works?

- Spark Streaming
- receives live input data streams and
- divides the data into batches



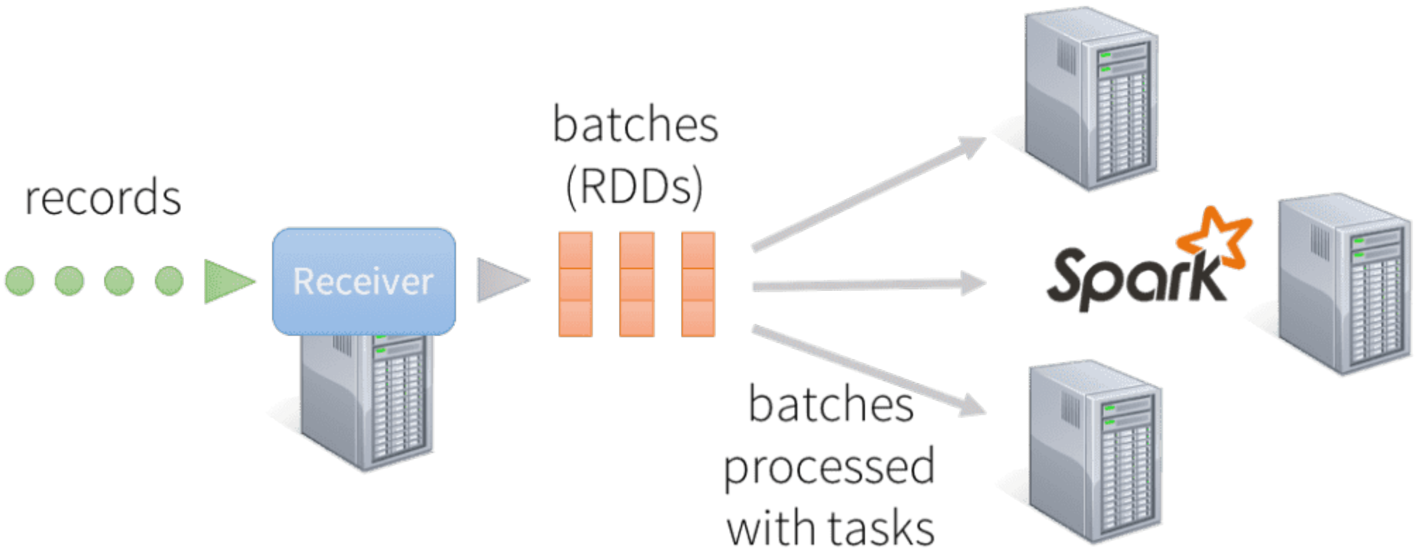
# How it works?

- processed by the Spark engine
- to generate the
- final stream of results in batches.

# High level streaming Architecture

## Spark Streaming

*discretized stream processing*



records processed in batches with short tasks
each batch is a RDD (partitioned dataset)

# High level streaming Architecture

At a high level, modern distributed stream processing pipelines execute as follows:

**Receive** streaming data from data sources

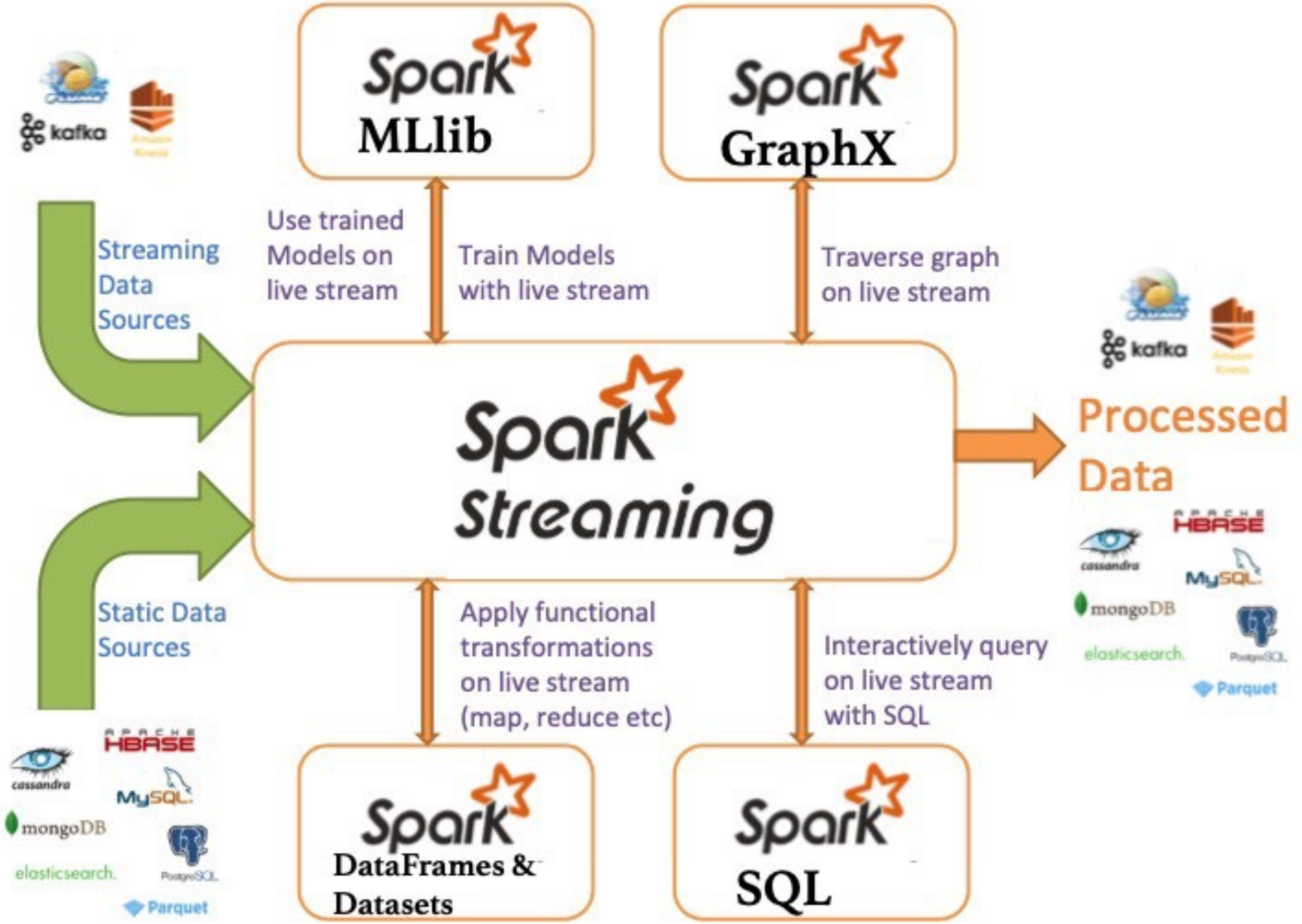
(e.g. live logs, system telemetry data, IoT device data, etc.)

into some data ingestion system like

Apache Kafka, Amazon Kinesis, etc.

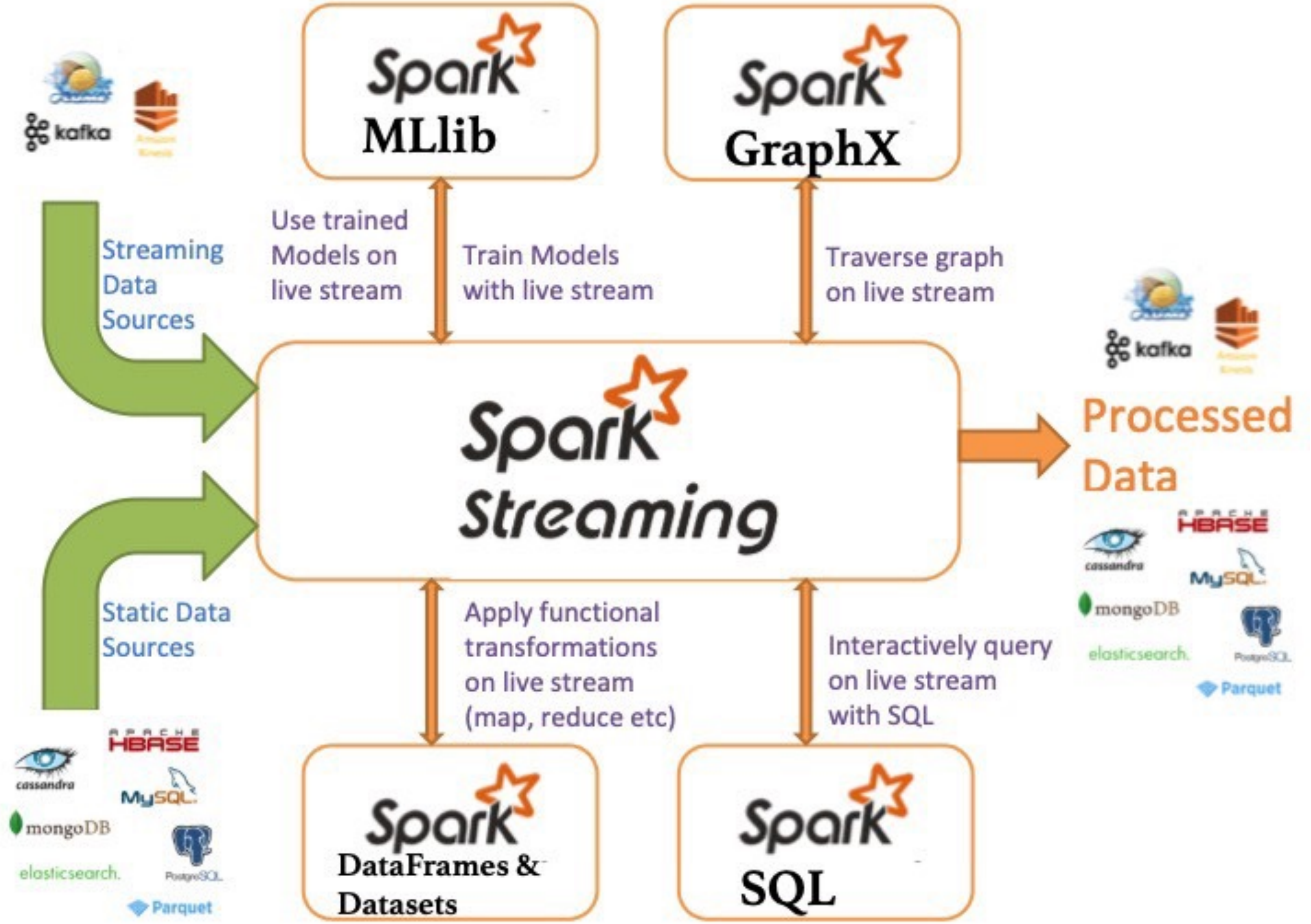
# High level streaming Architecture

- Stream Process Engine
- **Process** the data in
- parallel on a cluster.

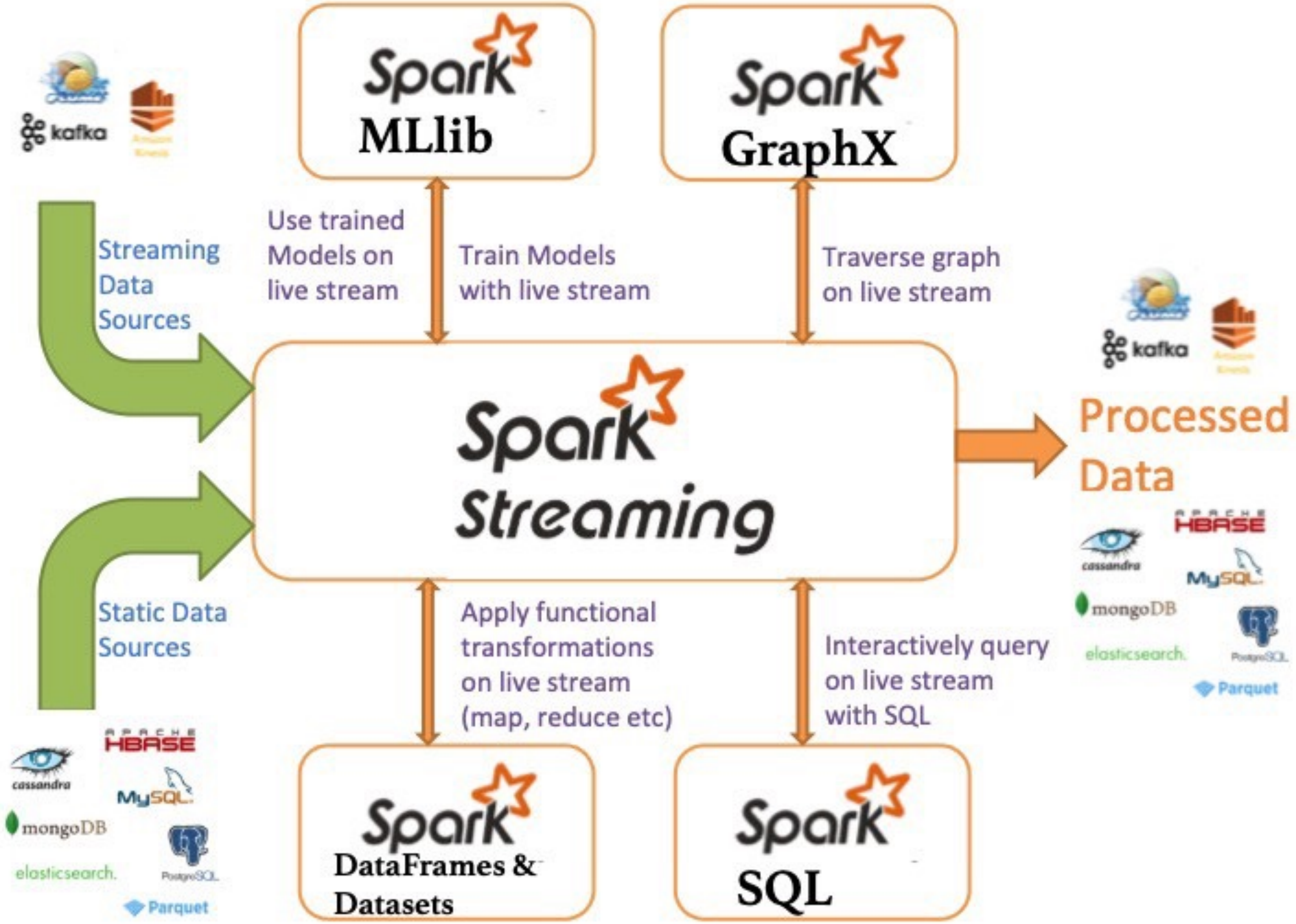

kafka
Amazon Kinesis
Streaming Data Sources
Static Data Sources
cassandra
HBASE
MySQL
mongoDB
elasticsearch
PostgreSQL
Parquet
Spark MLlib
Spark GraphX
Use trained Models on live stream
Train Models with live stream
Traverse graph on live stream
Spark Streaming
Processed Data
Apply functional transformations on live stream (map, reduce etc)
Interactively query on live stream with SQL
Spark DataFrames & Datasets
Spark SQL

# High level streaming Architecture

- **Output** the results
- out to downstream systems like
- HBase, Cassandra, Kafka, etc.



kafka
Amazon Kinesis
Streaming Data Sources
Static Data Sources
cassandra
HBASE
MySQL
mongoDB
elasticsearch
PostgreSQL
Parquet
Spark MLlib
Spark GraphX
Use trained Models on live stream
Train Models with live stream
Traverse graph on live stream
Spark Streaming
Processed Data
Apply functional transformations on live stream (map, reduce etc)
Interactively query on live stream with SQL
Spark DataFrames & Datasets
Spark SQL



Spark MLlib

Spark GraphX

Streaming Data Sources

Use trained Models on live stream

Train Models with live stream

Traverse graph on live stream

Spark Streaming

Processed Data

Static Data Sources

Apply functional transformations on live stream (map, reduce etc)

Interactively query on live stream with SQL

Spark DataFrames & Datasets

Spark SQL

# Spark streaming Sources

**File Streams:**

Reading data from files

on any file system

compatible with the HDFS API

HDFS, S3, NFS

DStream = streamingContext.fileStream<...>(directory);

# Spark streaming Sources

**Streams based on Custom Receivers:**

DStreams can be created

with data streams received

through custom receivers,

extending the Receiver<T> class...

streamingContext.queueStream(queueOfRDDs)

# Spark streaming Sources

**Queue of RDDs as a Stream:**

For testing a Spark Streaming application

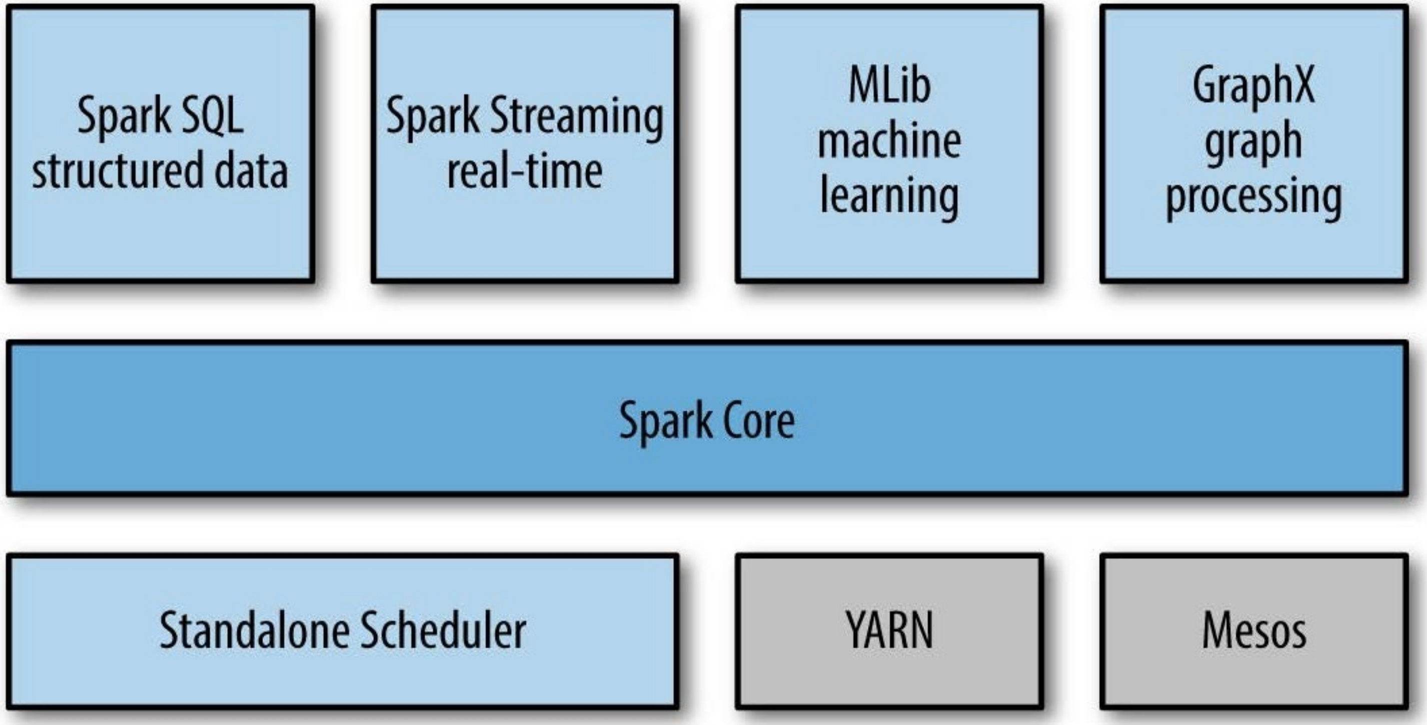
with test data

DStream based on a queue of RDDs

streamingContext.queueStream(queueOfRDDs)

# Spark streaming's place in spark

- **Spark Streaming:**
- A component that
- enables processing of
- live streams of data



# Spark structured streaming place in spark

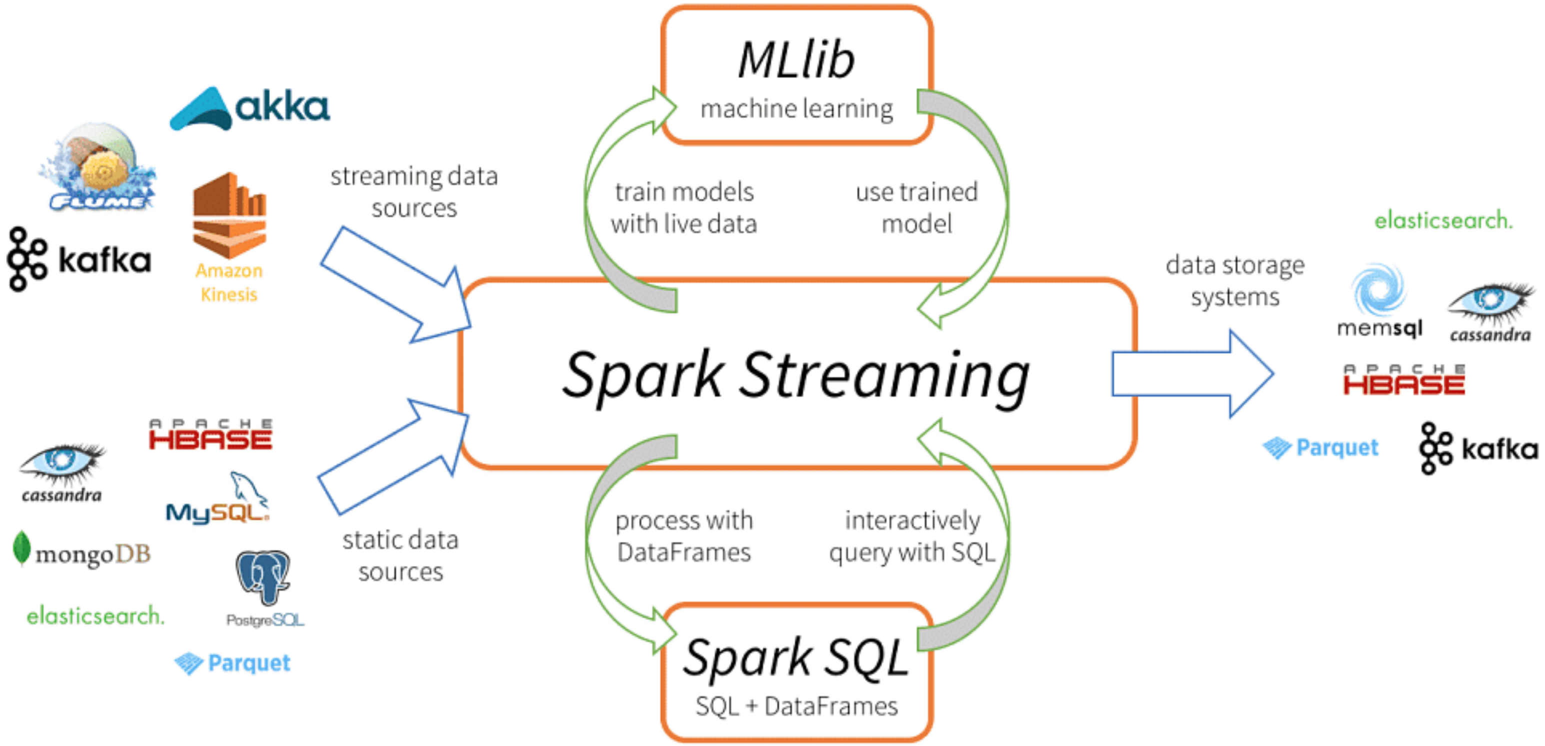
The Spark SQL engine

will take care of running it incrementally

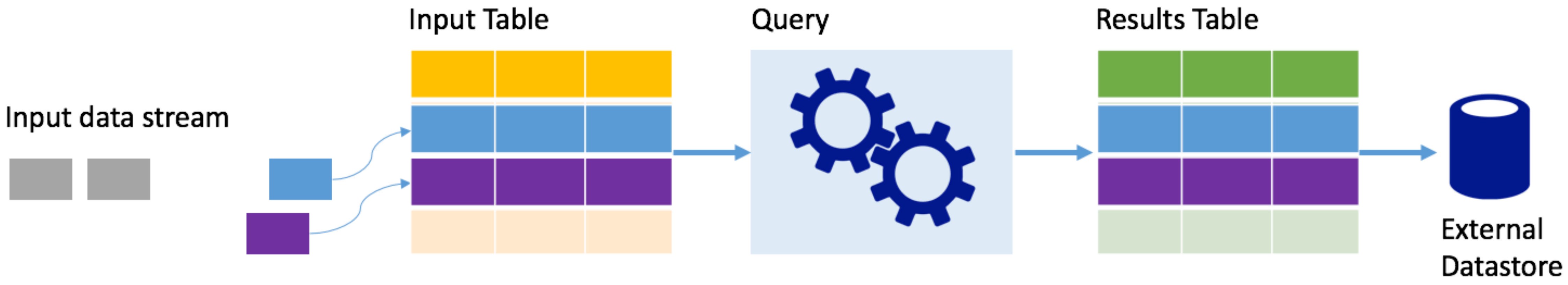
Continuously and updating the final result

as streaming data continues to arrive.

# Spark structured streaming

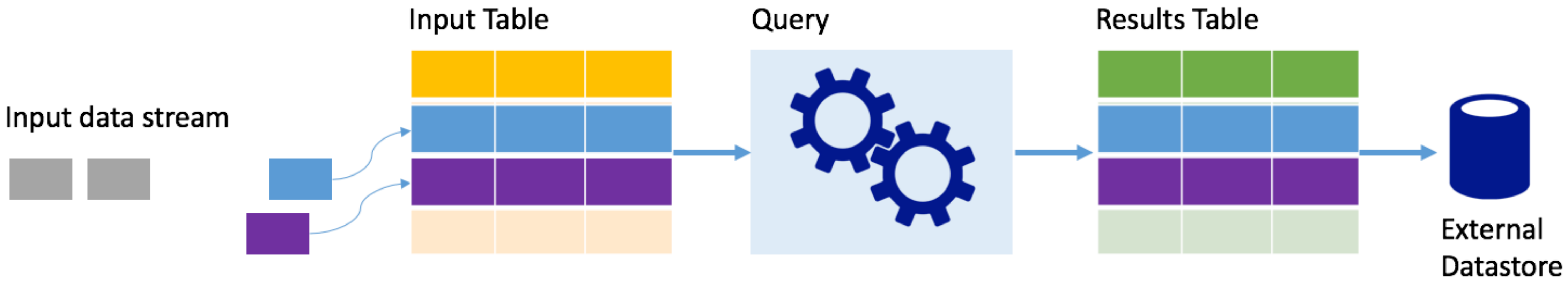


Structured Streaming is a scalable and fault-tolerant stream processing engine built on the Spark SQL engine.



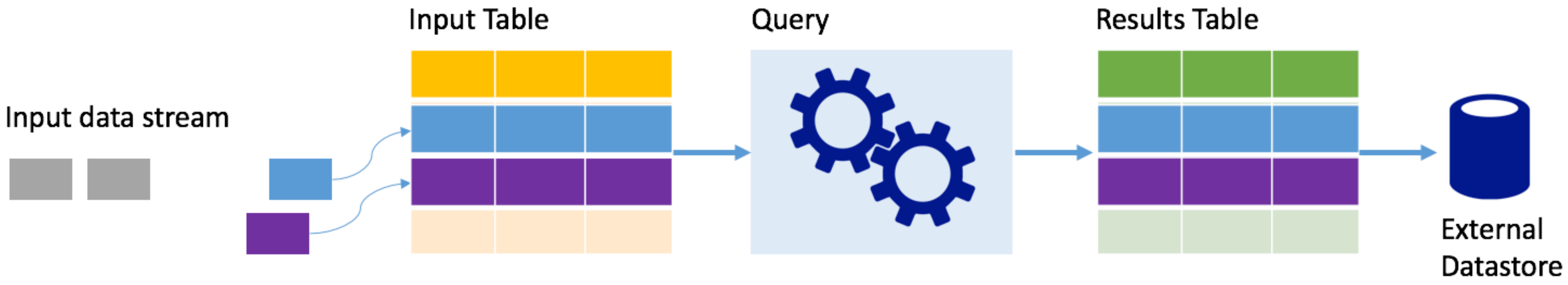
# Streams as tables

- Spark Structured Streaming represents
- a stream of data as a table



# Streams as tables

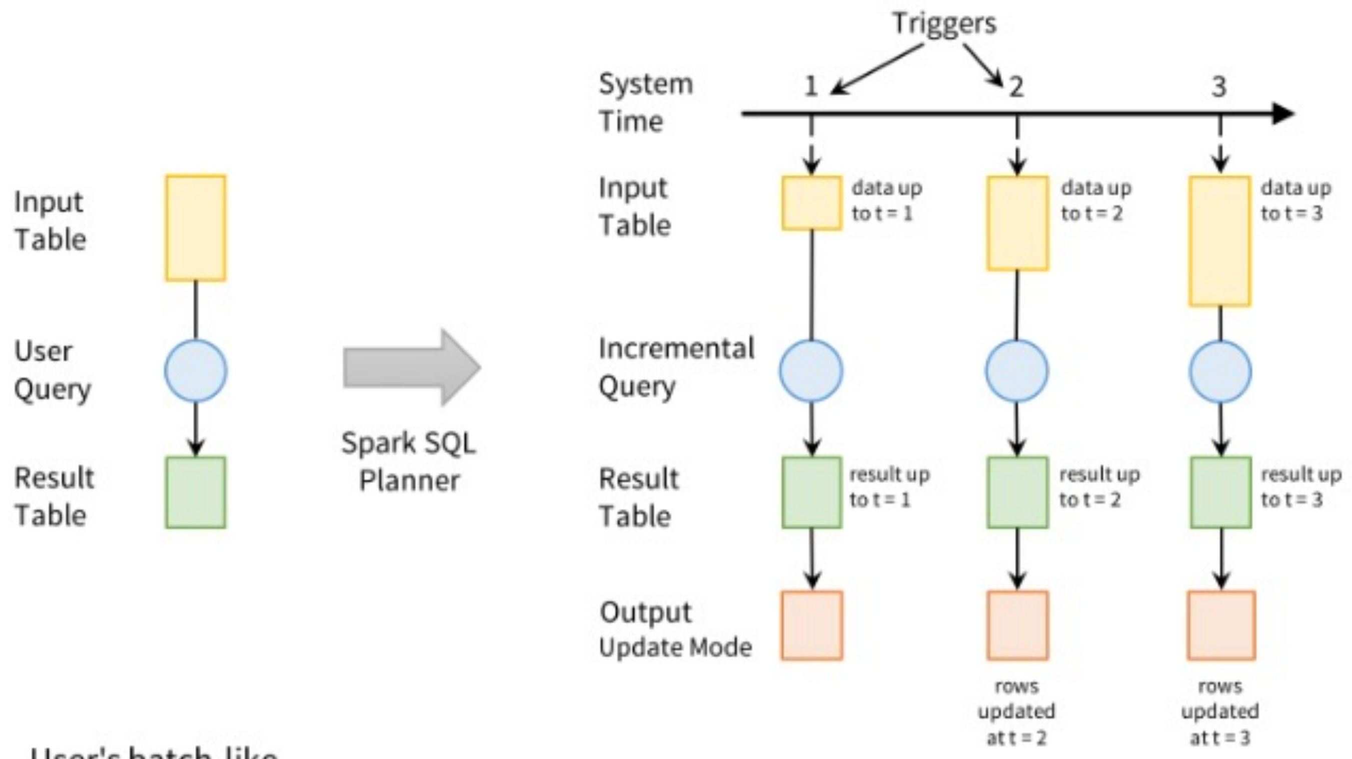
- Unbounded in depth,
- the table continues to grow as new data arrives.



# Streams as tables

- This *input table* is continuously processed
- by a long-running query
- the results sent to an *output table*

# Spark structured streaming

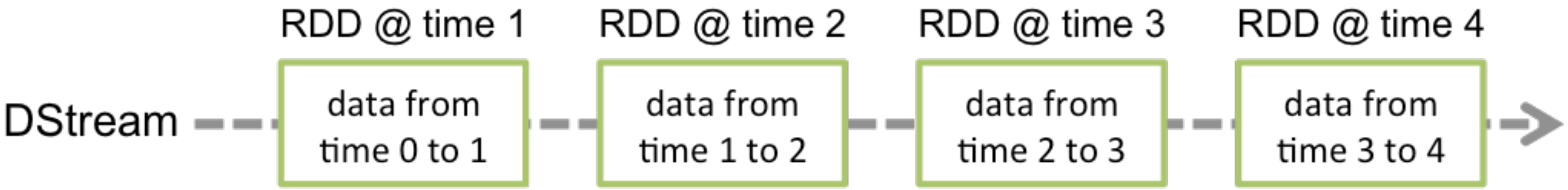


User's batch-like query on input table

Incremental execution on streaming data

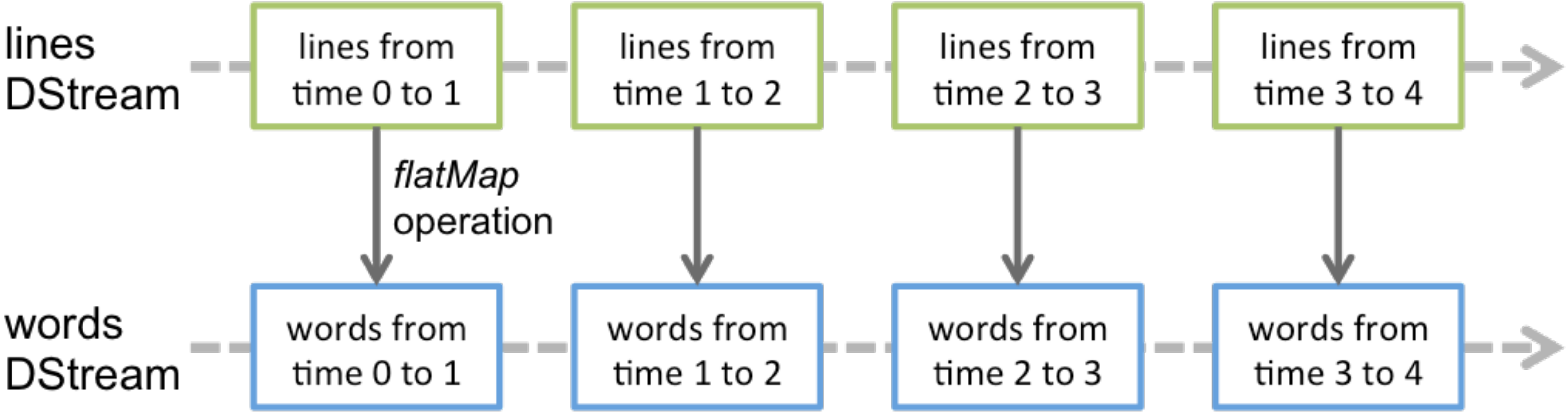
Structured Streaming Processing Model

Users express queries using a batch API; Spark incrementalizes them to run on streams



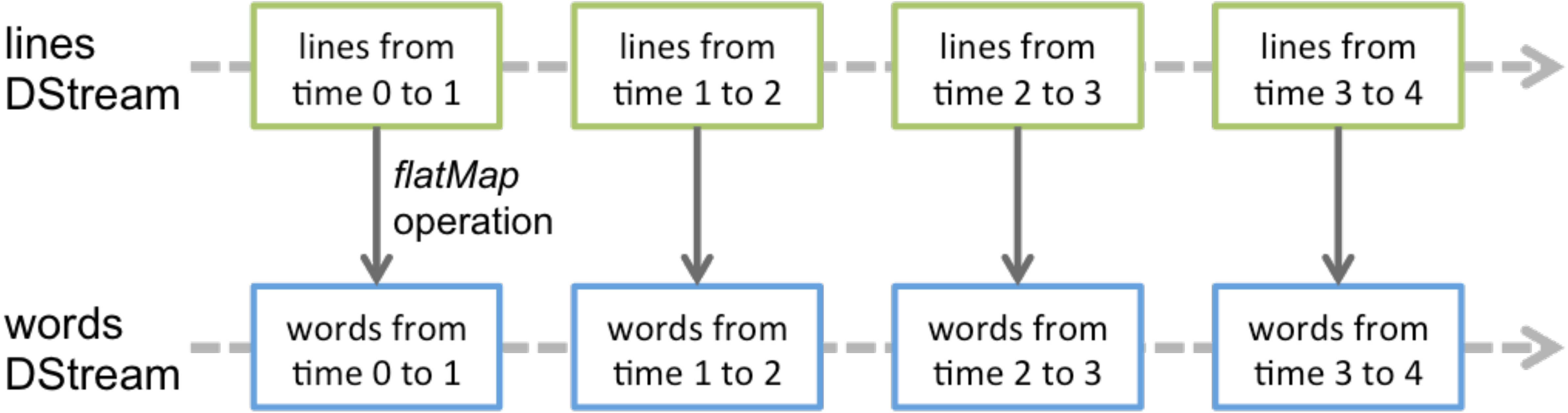
# Discretized Streams (DStreams)

- Basic abstraction provided by Spark Streaming.



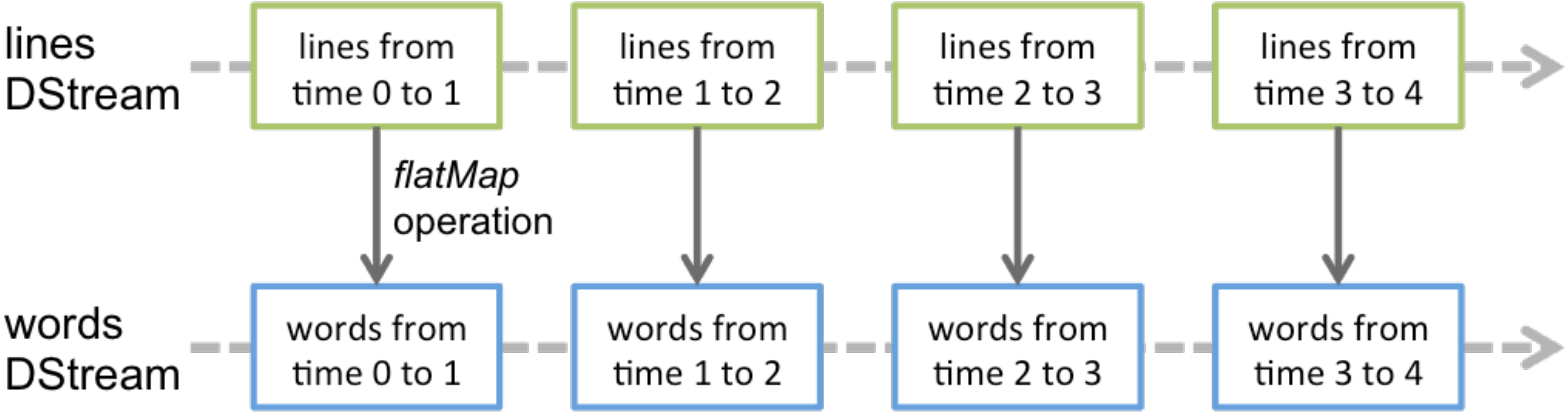
# Discretized Streams (DStreams)

- Continuous stream of data,
- input data stream received from source



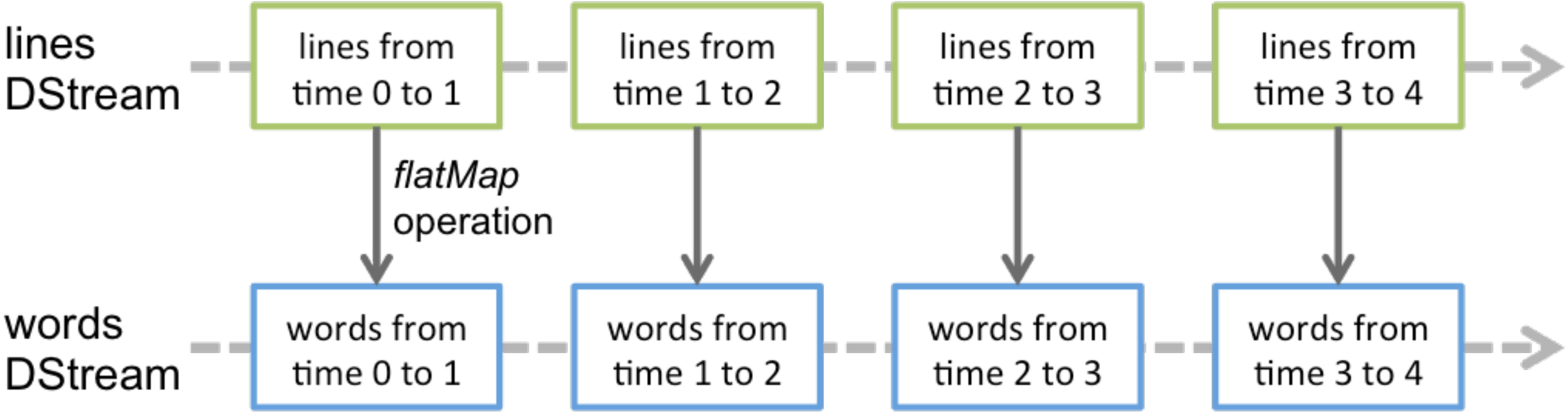
# Discretized Streams (DStreams)

- The processed data stream
- generated by transforming
- the input stream.



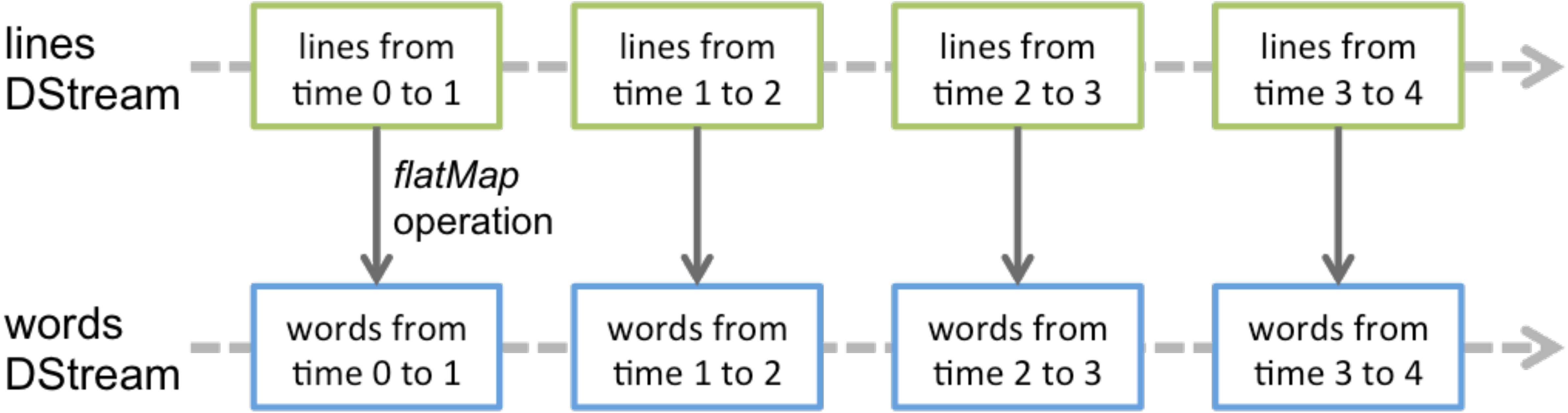
# Discretized Streams (DStreams)

- Internally, a DStream is represented
- by a continuous series of RDDs



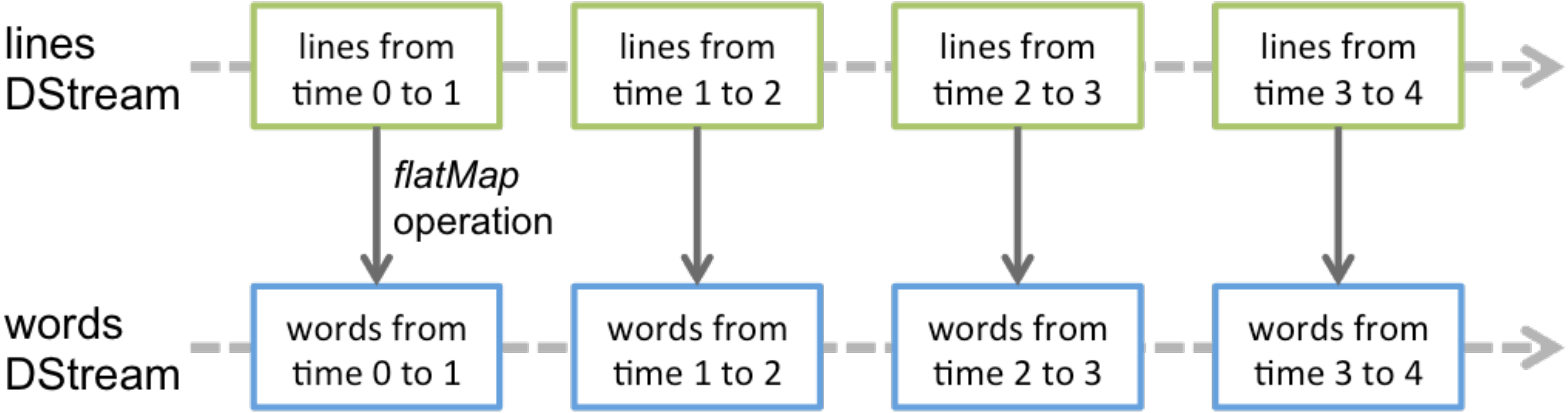
# Discretized Streams (DStreams)

- Internally, a DStream is represented
- by a continuous series of RDDs
- Abstraction of an immutable,
- distributed dataset



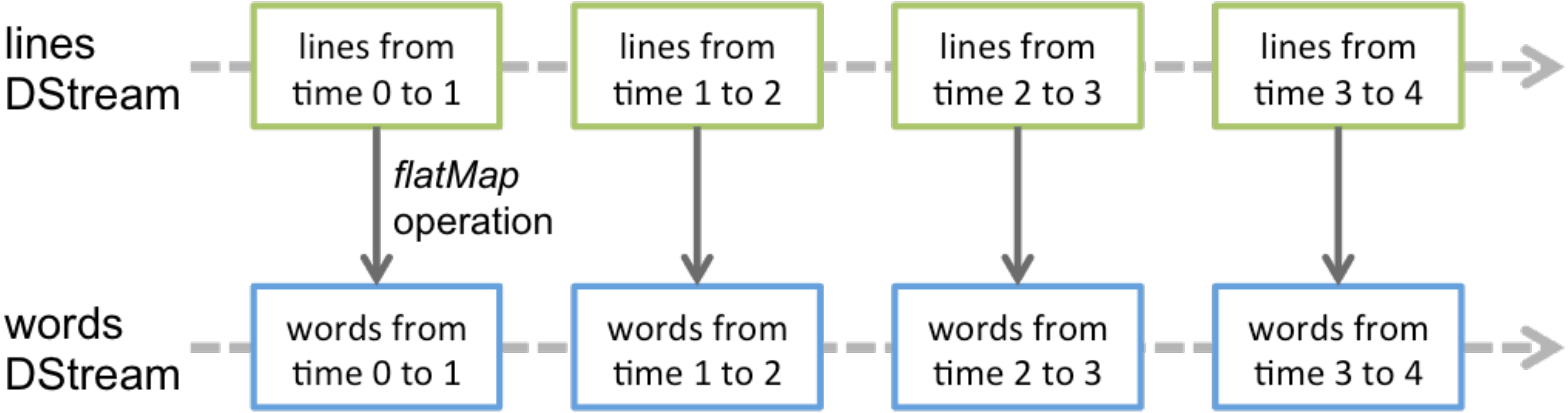
# Discretized Streams (DStreams)

- Each RDD in a DStream
- contains data from
- a certain interval.



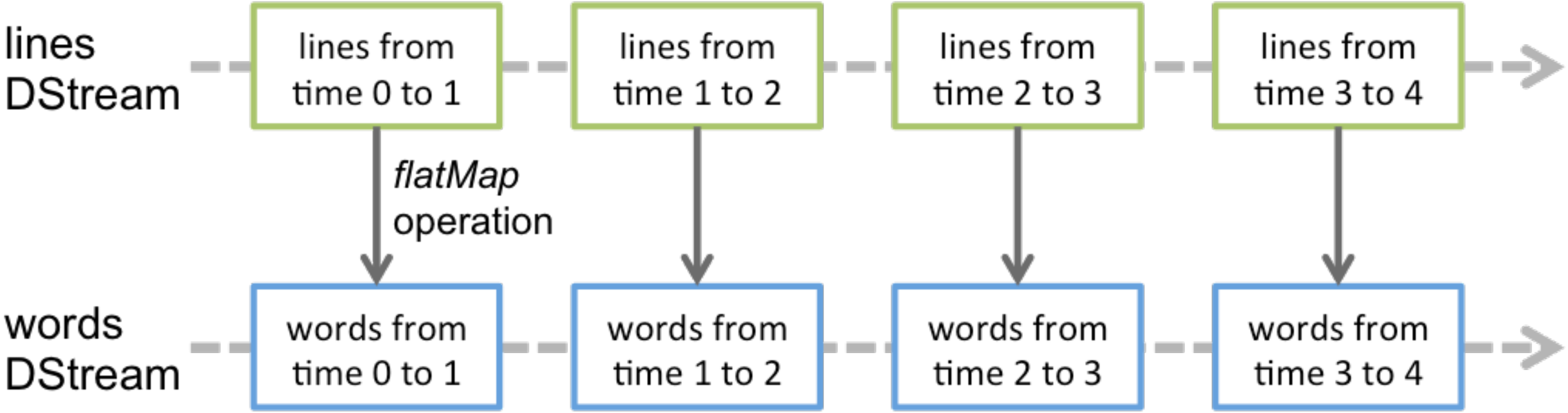
# Discretized Streams (DStreams)

- Any operation applied on a
- DStream translates
- to operations on the underlying RDDs



# Discretized Streams (DStreams)

- Converting a stream of lines to words,
- the flatMap operation is applied on each RDD
- in the lines DStream to generate
- the RDDs of the words DStream.



# Discretized Streams (DStreams)

- Underlying RDD transformations
- are computed by the Spark engine
- The DStream operations hide most of these details
- Provide the developer with a higher-level API for convenience.

# Input DStreams and Receivers

Input DStreams are DStreams

representing the stream of input data

received from streaming sources.

# Input DStreams and Receivers

✓ lines was an input DStream as

it represented the stream of data

received from the netcat server.

# Input DStreams and Receivers

Every input DStream

is associated with

a **Receiver** (Scala ) object

which receives

the data from a source

stores it in Spark’s memory

for processing.

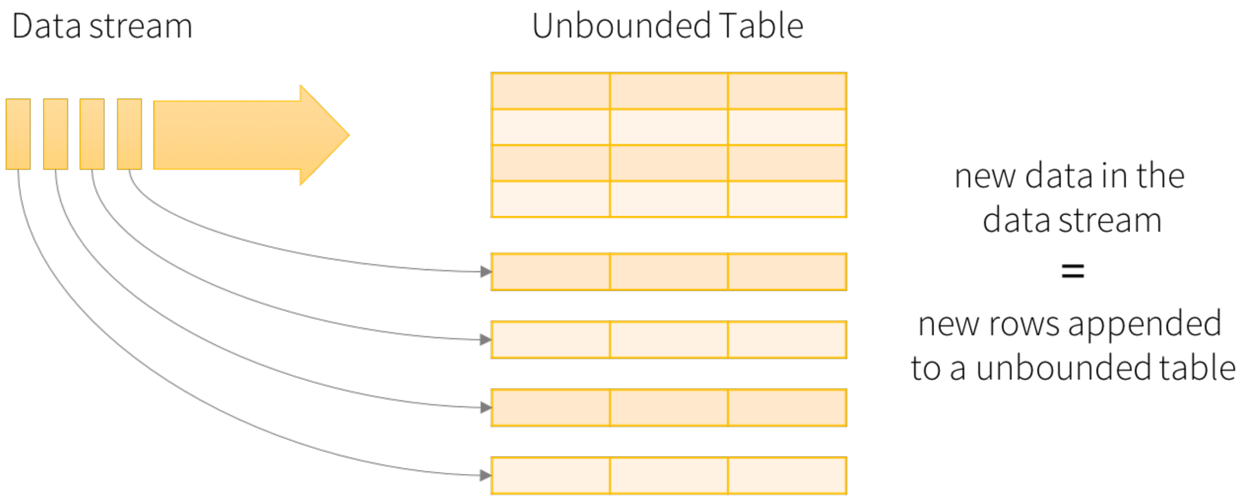
# A sample program

# Spark structured streaming

- Provides fast,
- scalable,
- fault-tolerant,
- end-to-end exactly-once
- stream processing

# How does it work?

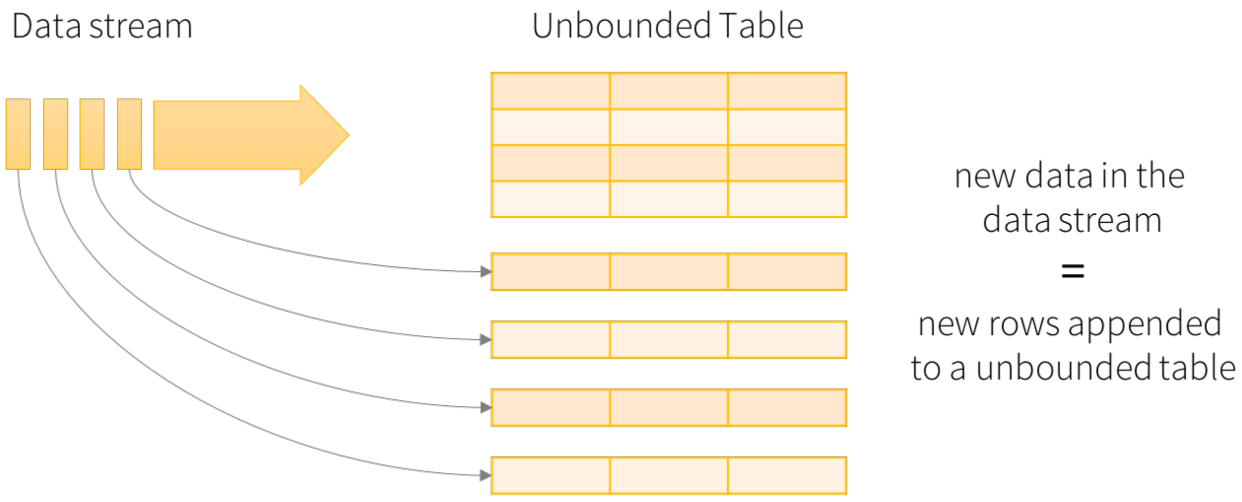
- Structured Streaming queries are
- processed using
- a *micro-batch processing* engine


Data stream
Unbounded Table
new data in the data stream
=
new rows appended to a unbounded table

Data stream as an unbounded table

# How does it work?

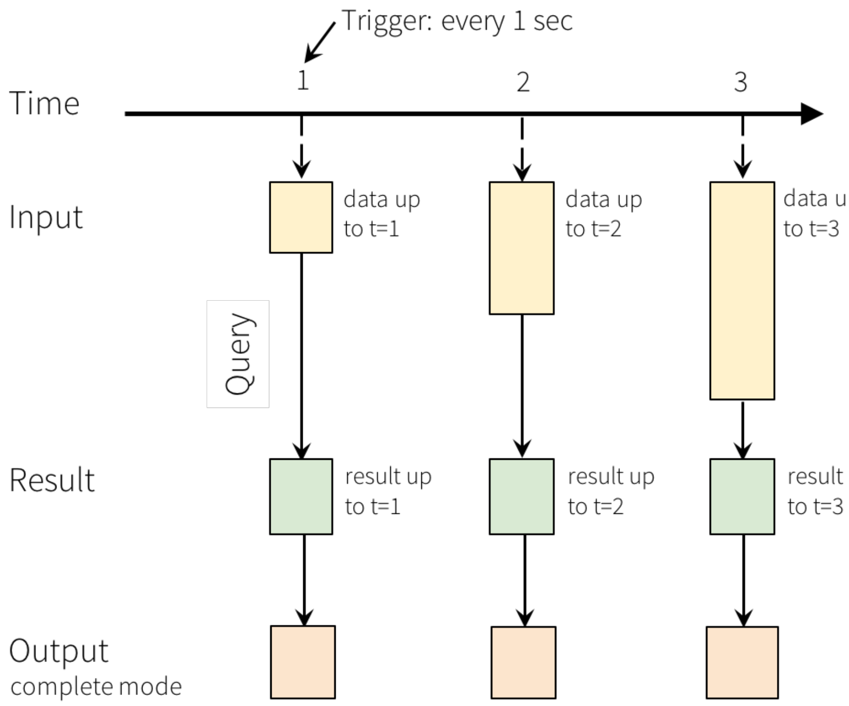
- which processes data streams
- as a series of small batch jobs


Data stream
Unbounded Table
new data in the data stream
=
new rows appended to a unbounded table

Data stream as an unbounded table

# How does it work?

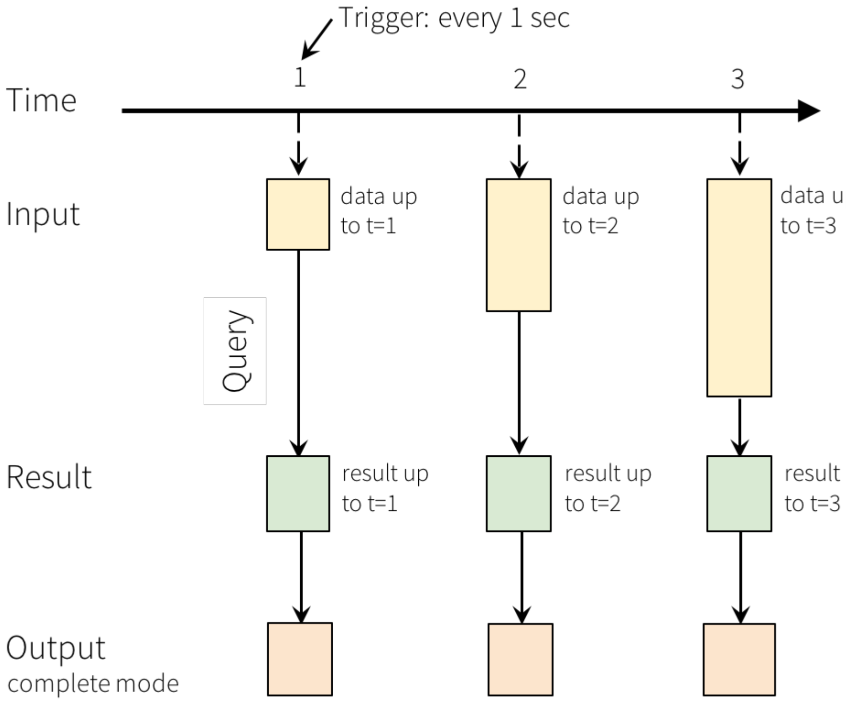
- achieving end-to-end latencies
- as low as 100 milliseconds and
- exactly-once fault-tolerance guarantees.



Programming Model for Structured Streaming

# How does it work?

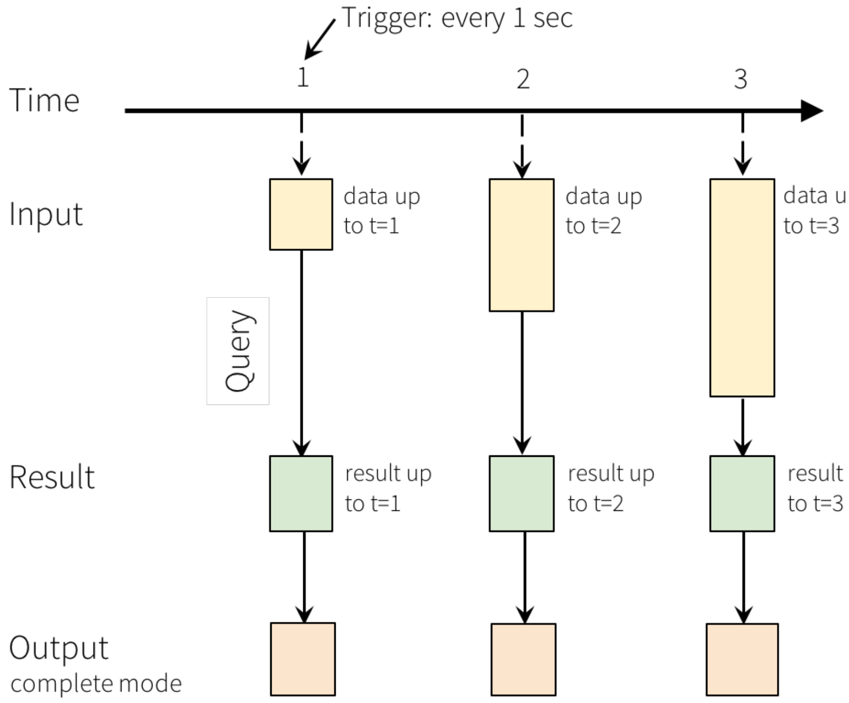
- In Spark 2.3 introduced
- a new low-latency processing mode
- called **Continuous Processing**



Programming Model for Structured Streaming

# How does it work?

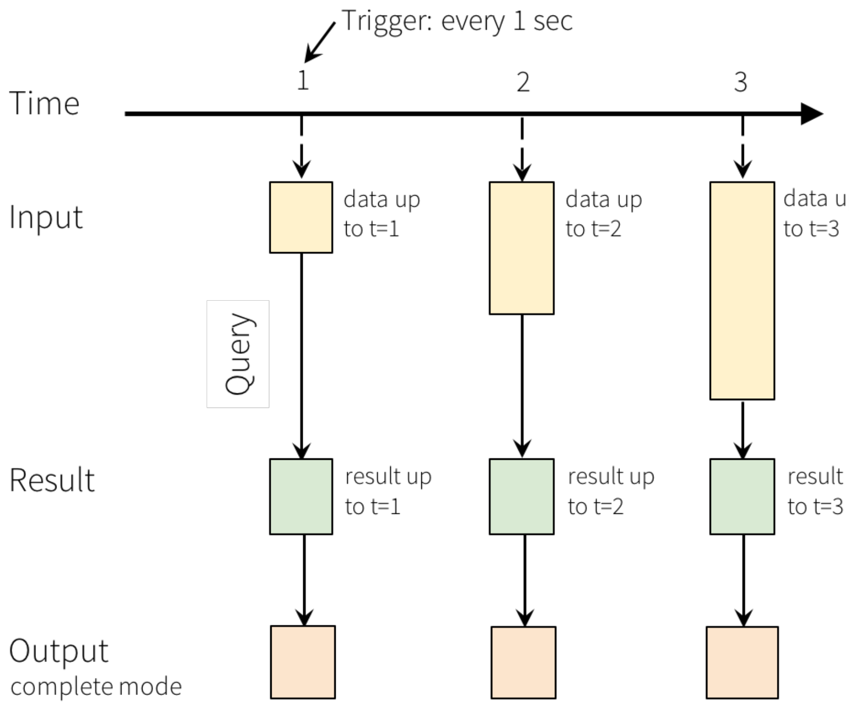
- which can achieve end-to-end latencies
- as low as 1 millisecond
- with at-least-once guarantees.



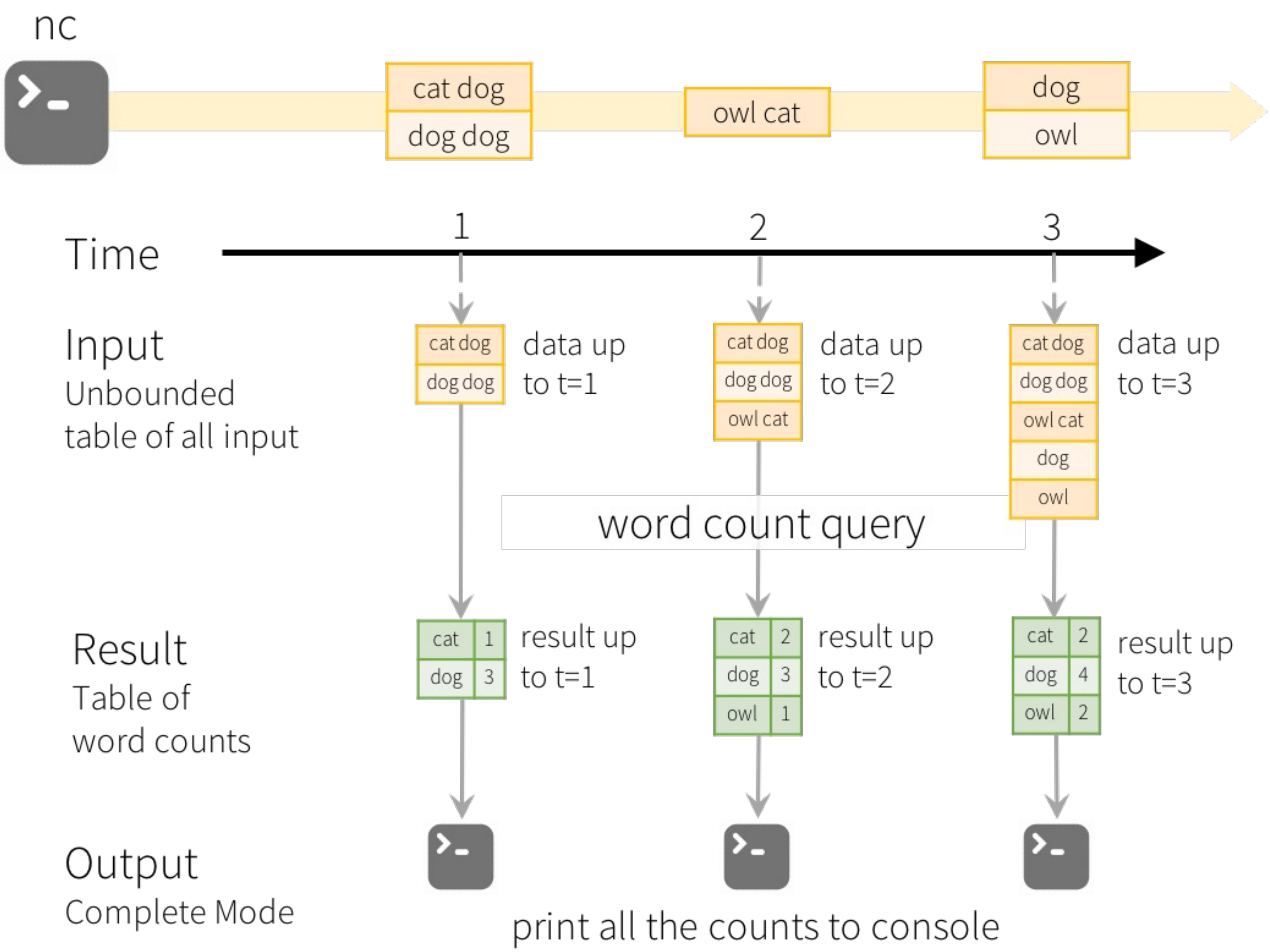
Programming Model for Structured Streaming

# How does it work?

- Without changing the Dataset/DataFrame operations in queries,
- you will be able to choose the mode
- based on your application requirements.



Programming Model for Structured Streaming



nc

cat dog
dog dog

owl cat

dog
owl

Time

1

2

3

Input

Unbounded table of all input

cat dog
dog dog

data up to t=1

cat dog
dog dog
owl cat

data up to t=2

cat dog
dog dog
owl cat
dog
owl

data up to t=3

word count query

Result

Table of word counts

| | |
|---|---|
| cat | 1 |
| dog | 3 |

result up to t=1

| | |
|---|---|
| cat | 2 |
| dog | 3 |
| owl | 1 |

result up to t=2

| | |
|---|---|
| cat | 2 |
| dog | 4 |
| owl | 2 |

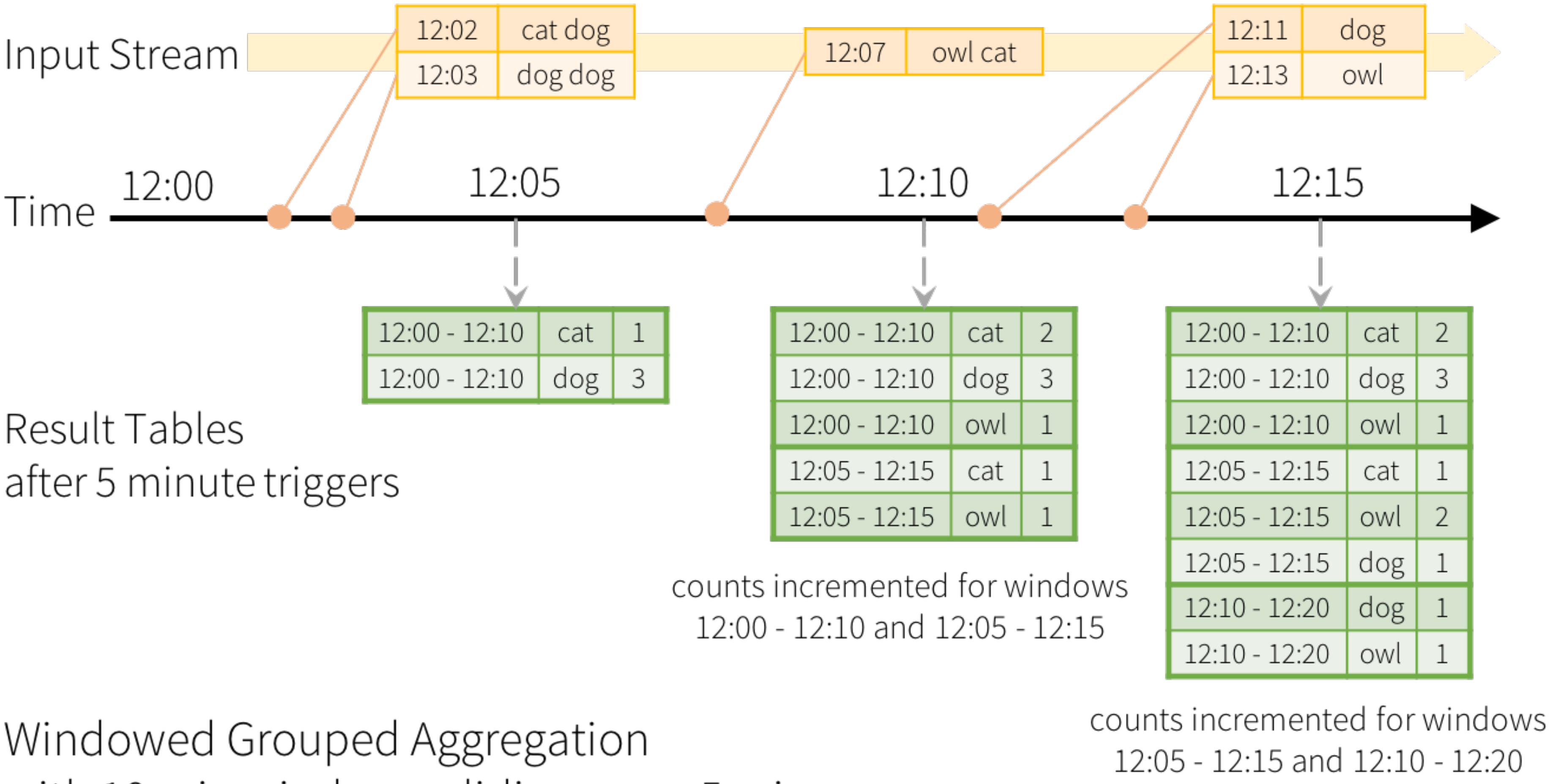
result up to t=3

Output

Complete Mode

print all the counts to console

Model of the Quick Example



Input Stream

| | |
|---|---|
| 12:02 | cat dog |
| 12:03 | dog dog |

| | |
|---|---|
| 12:07 | owl cat |

| | |
|---|---|
| 12:11 | dog |
| 12:13 | owl |

Result Tables
after 5 minute triggers

12:05

| | | |
|---|---|---|
| 12:00 - 12:10 | cat | 1 |
| 12:00 - 12:10 | dog | 3 |

12:10

| | | |
|---|---|---|
| 12:00 - 12:10 | cat | 2 |
| 12:00 - 12:10 | dog | 3 |
| 12:00 - 12:10 | owl | 1 |
| 12:05 - 12:15 | cat | 1 |
| 12:05 - 12:15 | owl | 1 |

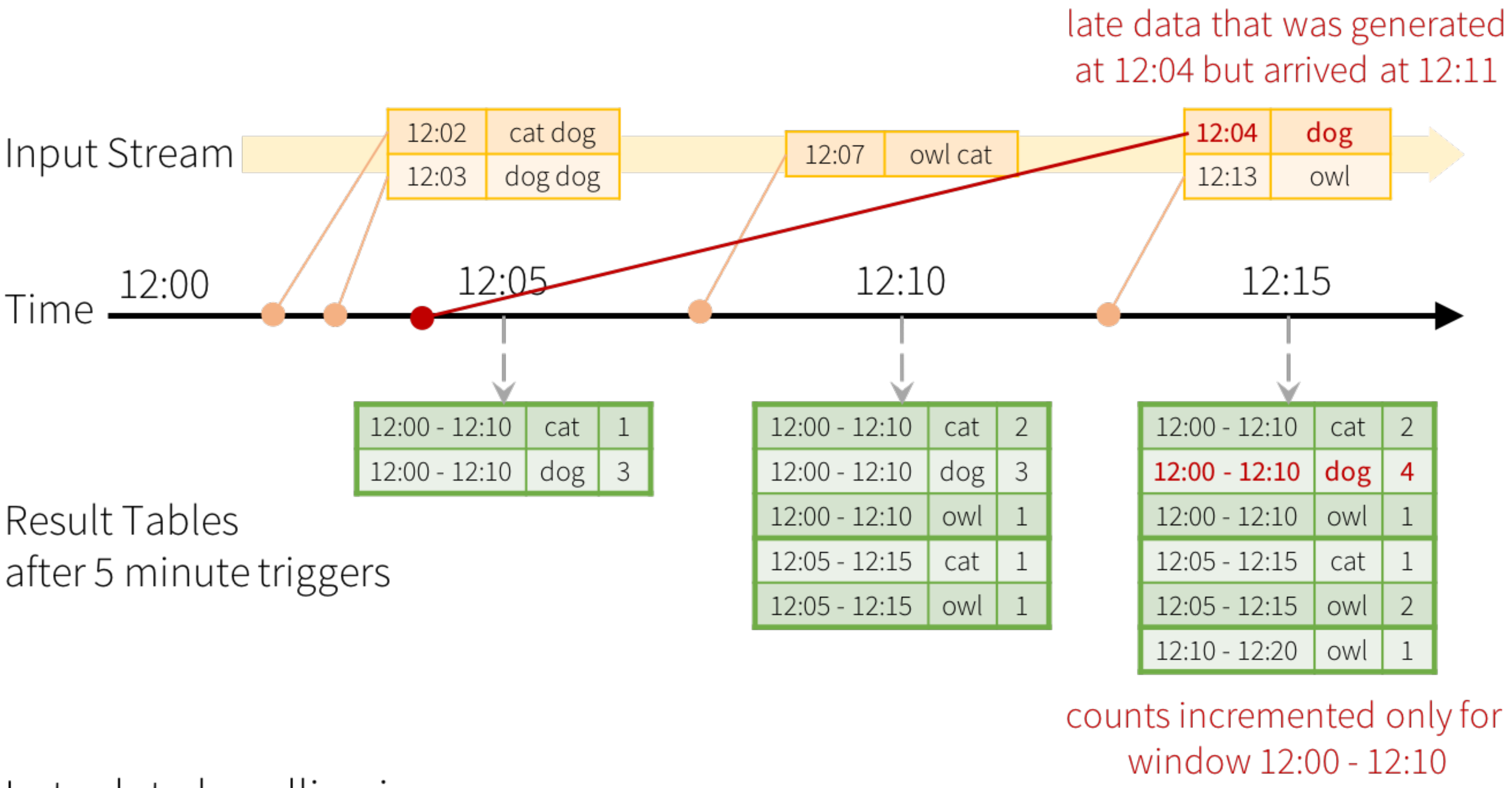
counts incremented for windows
12:00 - 12:10 and 12:05 - 12:15

12:15

| | | |
|---|---|---|
| 12:00 - 12:10 | cat | 2 |
| 12:00 - 12:10 | dog | 3 |
| 12:00 - 12:10 | owl | 1 |
| 12:05 - 12:15 | cat | 1 |
| 12:05 - 12:15 | owl | 2 |
| 12:05 - 12:15 | dog | 1 |
| 12:10 - 12:20 | dog | 1 |
| 12:10 - 12:20 | owl | 1 |

counts incremented for windows
12:05 - 12:15 and 12:10 - 12:20

Windowed Grouped Aggregation
with 10 min windows, sliding every 5 mins


late data that was generated at 12:04 but arrived at 12:11
Input Stream
12:02 cat dog
12:03 dog dog
12:07 owl cat
12:04 dog
12:13 owl
Time
12:00
12:05
12:10
12:15
Result Tables after 5 minute triggers
counts incremented only for window 12:00 - 12:10

| | | |
|---|---|---|
| 12:00 - 12:10 | cat | 1 |
| 12:00 - 12:10 | dog | 3 |

| | | |
|---|---|---|
| 12:00 - 12:10 | cat | 2 |
| 12:00 - 12:10 | dog | 3 |
| 12:00 - 12:10 | owl | 1 |
| 12:05 - 12:15 | cat | 1 |
| 12:05 - 12:15 | owl | 1 |

| | | |
|---|---|---|
| 12:00 - 12:10 | cat | 2 |
| 12:00 - 12:10 | dog | 4 |
| 12:00 - 12:10 | owl | 1 |
| 12:05 - 12:15 | cat | 1 |
| 12:05 - 12:15 | owl | 2 |
| 12:10 - 12:20 | owl | 1 |

Late data handling in
Windowed Grouped Aggregation

# Steps for structured streaming

Let’s say you want

to maintain a running word count

of text data received

from a data server

listening on a TCP socket.

# Steps for structured streaming

- import org.apache.spark.sql.functions._
- import org.apache.spark.sql.SparkSession

# Steps for structured streaming

```
 scala> val spark = SparkSession
 .builder
 .appName("StructuredNetworkWordCount")
 .getOrCreate()

scala> import spark.implicits._
```

# Steps for structured streaming

- // Create DataFrame representing the stream of input lines from connection to localhost:9999

- val lines = spark.readStream
-   .format("socket")
-   .option("host", "localhost")
-   .option("port", 9999)
-   .load()

# Steps for structured streaming

- // Split the lines into words
- val words = lines.as[String].flatMap(_.split(" "))

- // Generate running word count
- val wordCounts = words.groupBy("value").count()

# Steps for structured streaming

- // Start running the query that prints the running counts to the console
- val query = wordCounts.writeStream
- .outputMode("complete")
- .format("console")
- .start()

- query.awaitTermination()

# Steps for structured streaming

- $ nc -lk 9999
- apache spark
- apache hadoop

- apache spark
- apache hadoop

# Steps for structured streaming

```
$ ./bin/run-example org.apache.spark.examples.sql.streaming.StructuredNetworkWordCount localhost 9999
```

```
-------------------------------------------
Batch: 0
-------------------------------------------
+------+-----+
| value|count|
+------+-----+
|apache|    1|
| spark|    1|
+------+-----+
```

# Supported sources and sinks

- https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html