Conceptualizing the Processing Model for Apache Spark Structured Streaming

GETTING STARTED WITH STRUCTURED STREAMING

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

Overview

Streaming and its place in the Apache Spark stack

RDDs and DataFrames

Structured Streaming in Spark 2.x

Batch as a prefix of stream

ClusterManager, SparkSession, and Executors

Prerequisites and Course Outline

Prerequisites

Comfortable programming in Python

Some exposure to Apache Spark and DataFrames

Exposure to streaming data would be helpful, but is not required

Prerequisite Courses

Apache Spark Fundamentals

Getting Started with Spark 2

Course Outline

Getting Started with Structured Streaming

Executing Streaming Queries

Understanding Scheduling and Checkpointing

Configuring Processing Models

Understanding Query Planning

Introducing Apache Spark

Hadoop

HDFS

MapReduce

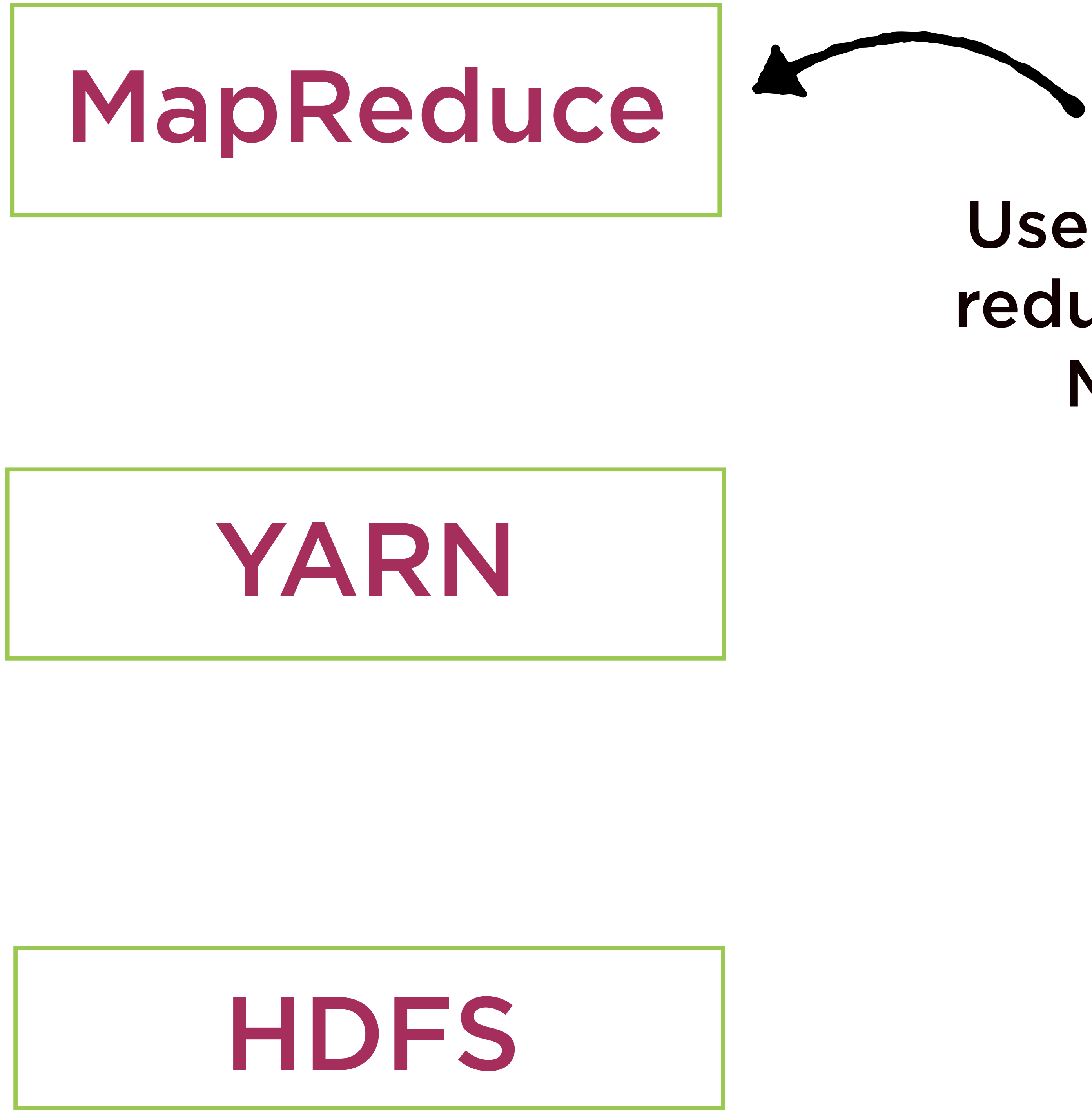
YARN

A file system to manage the storage of data

A framework to define a data processing task

A framework to run the data processing task

Co-ordination Between Hadoop Blocks



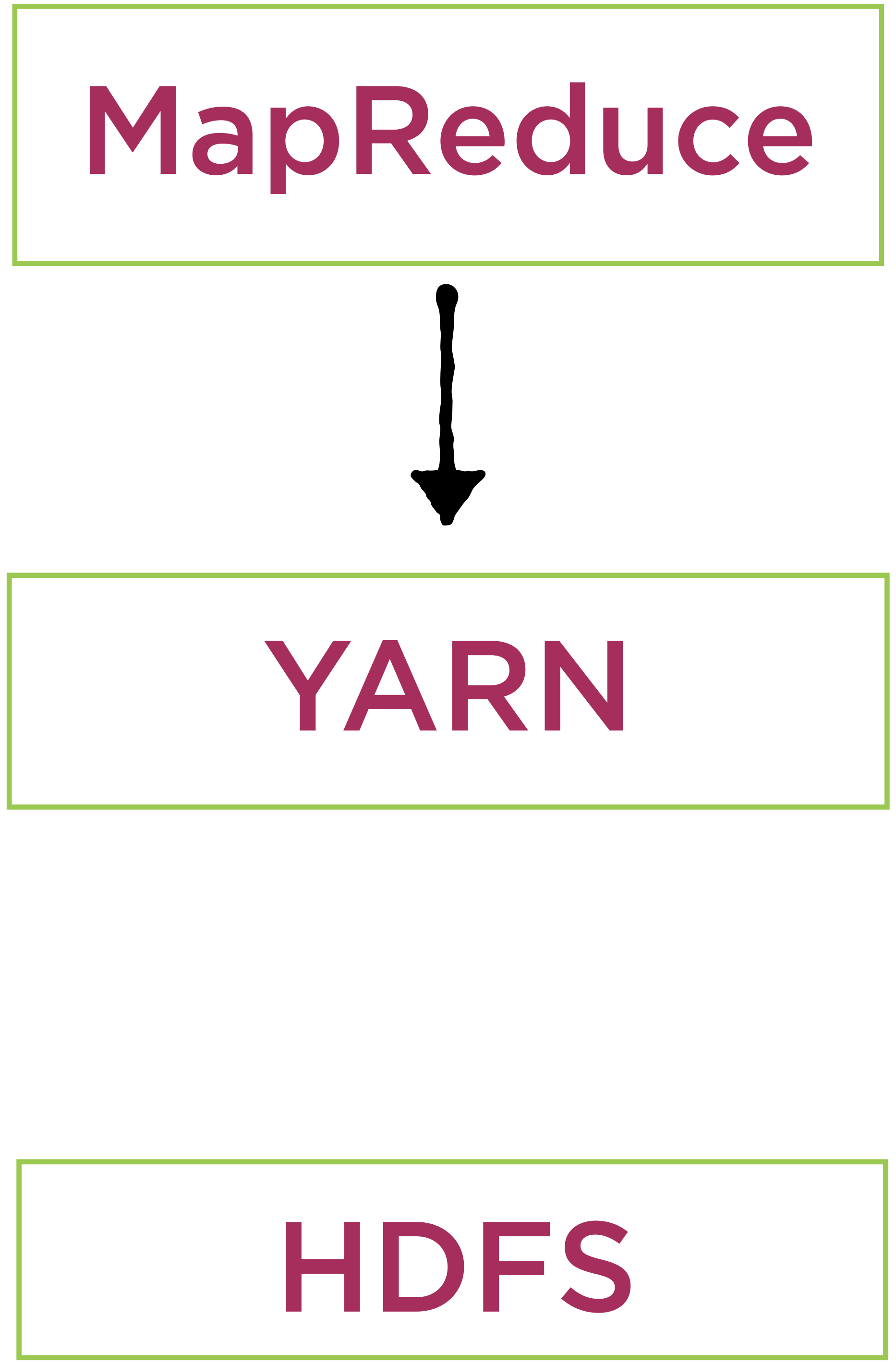
MapReduce

YARN

HDFS

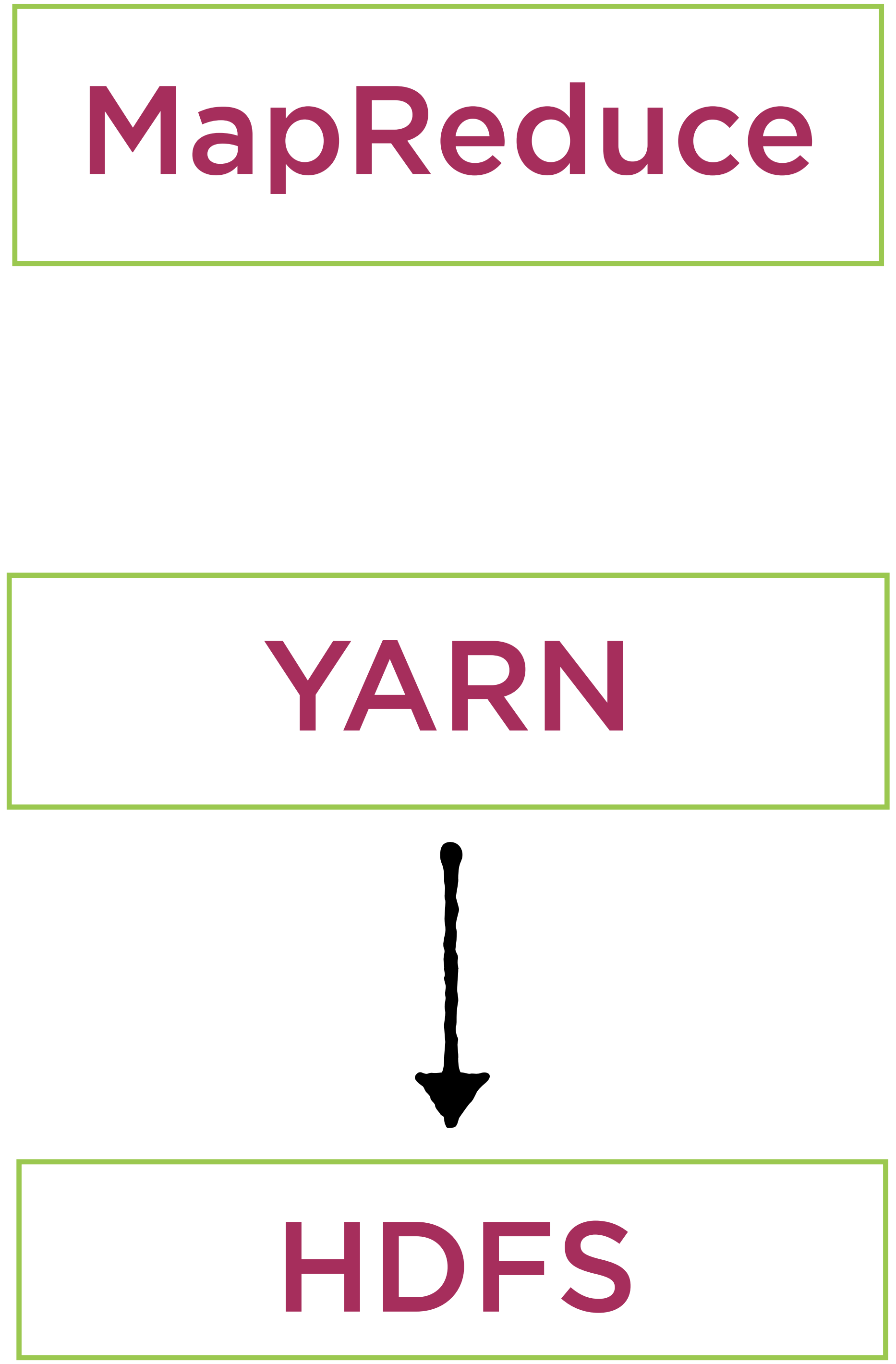
User defines map and reduce tasks using the MapReduce API

Co-ordination Between Hadoop Blocks



A job is triggered on the cluster

Co-ordination Between Hadoop Blocks



MapReduce

YARN

HDFS

YARN figures out where and how to run the job, and stores the result in HDFS

Apache Spark

General Purpose

Interactive

Distributed Computing

An engine for data processing and analysis

Apache Spark

Analytics and ML on Big Data

Extremely powerful and popular Big Data technology

Distributed computing framework for general-purpose computing

Open-source from Apache

Written in Scala

Apache Spark

Real-time as well as batch

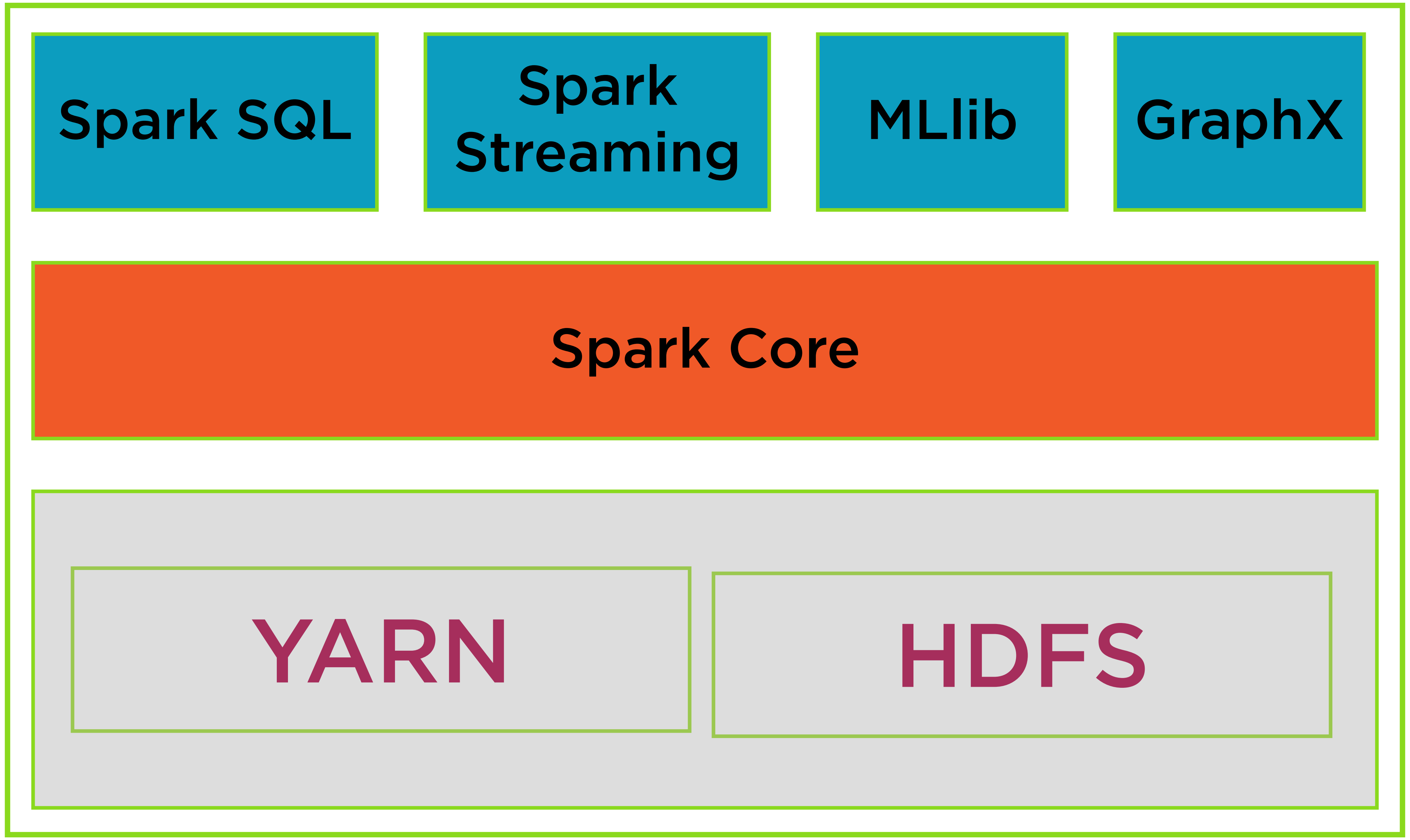
Interactive REPL environment

Read-Evaluate-Print-Loop

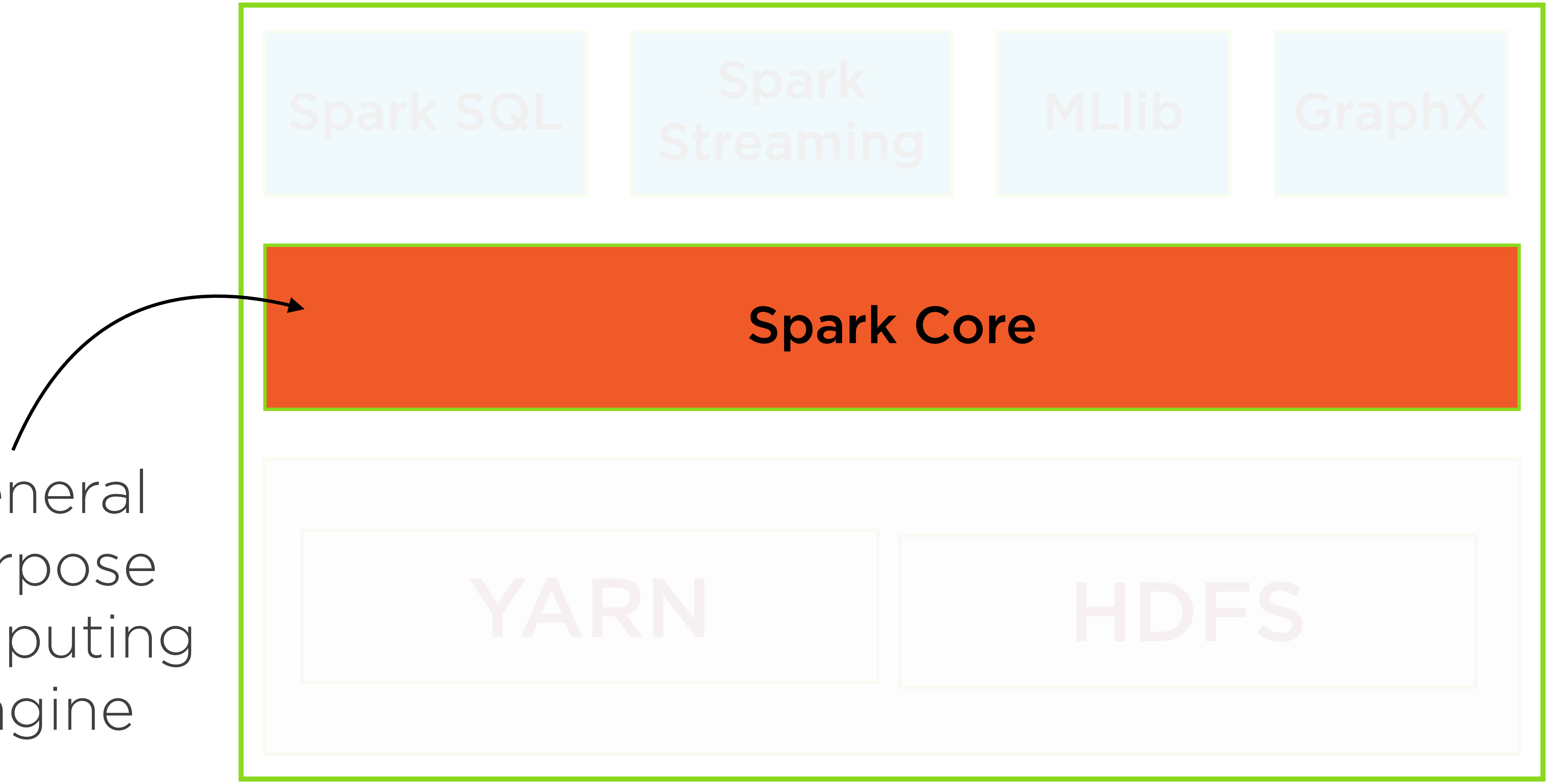
Support for multiple programming languages

- Python, Scala, R, ...

Apache Spark



Apache Spark



Spark SQL

Spark Streaming

MLlib

GraphX

Spark Core

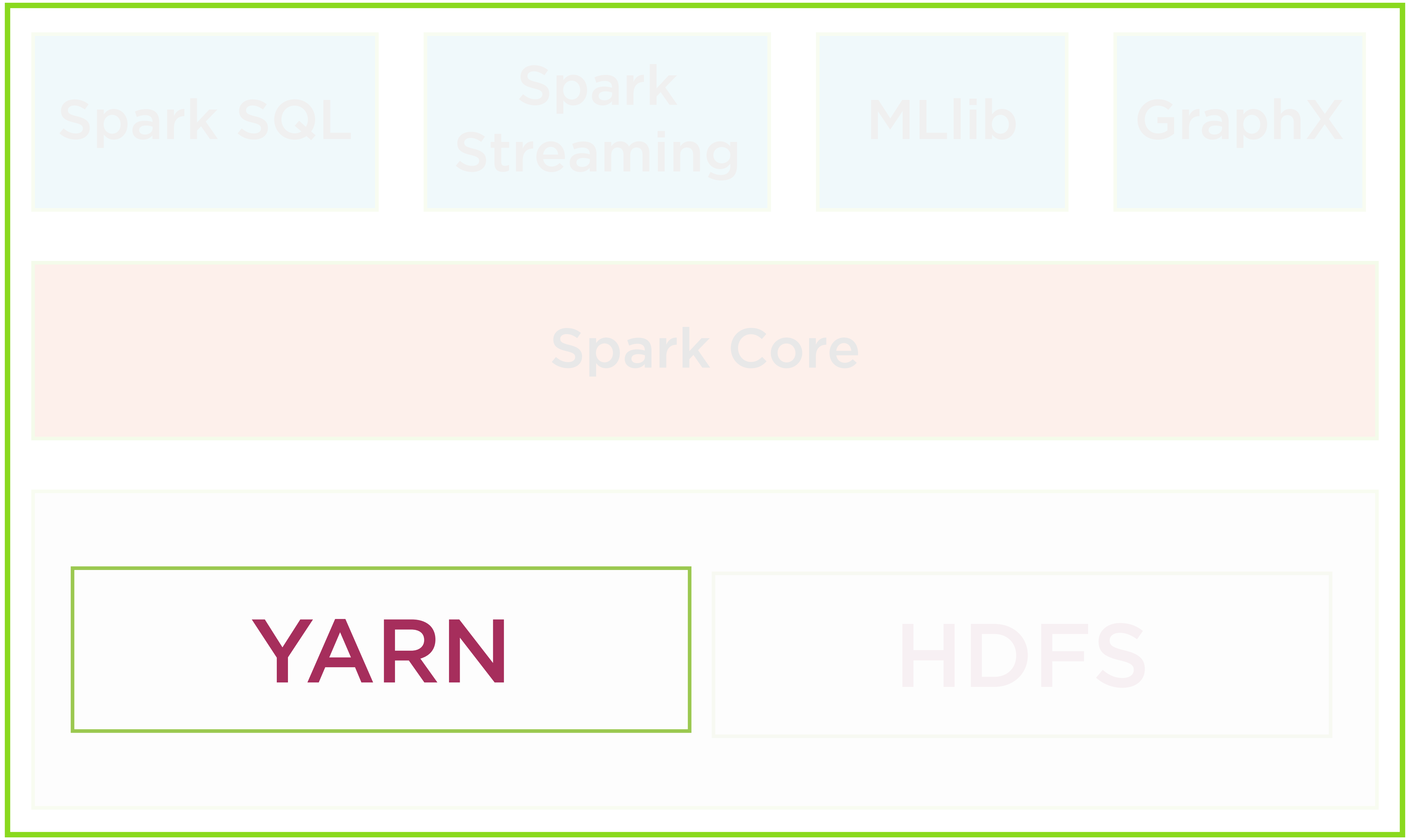
YARN

HDFS

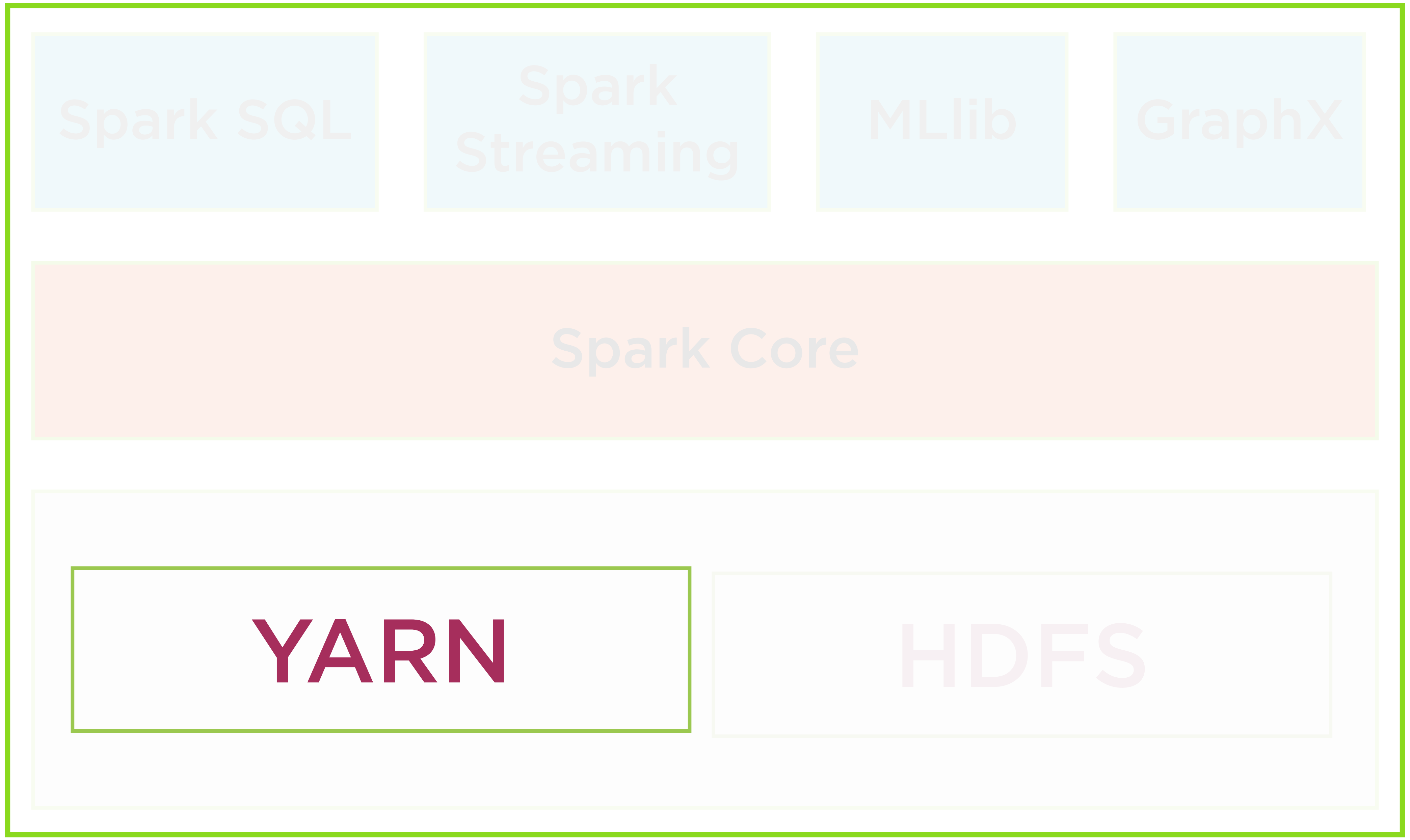
General purpose computing engine

Apache Spark

Cluster manager

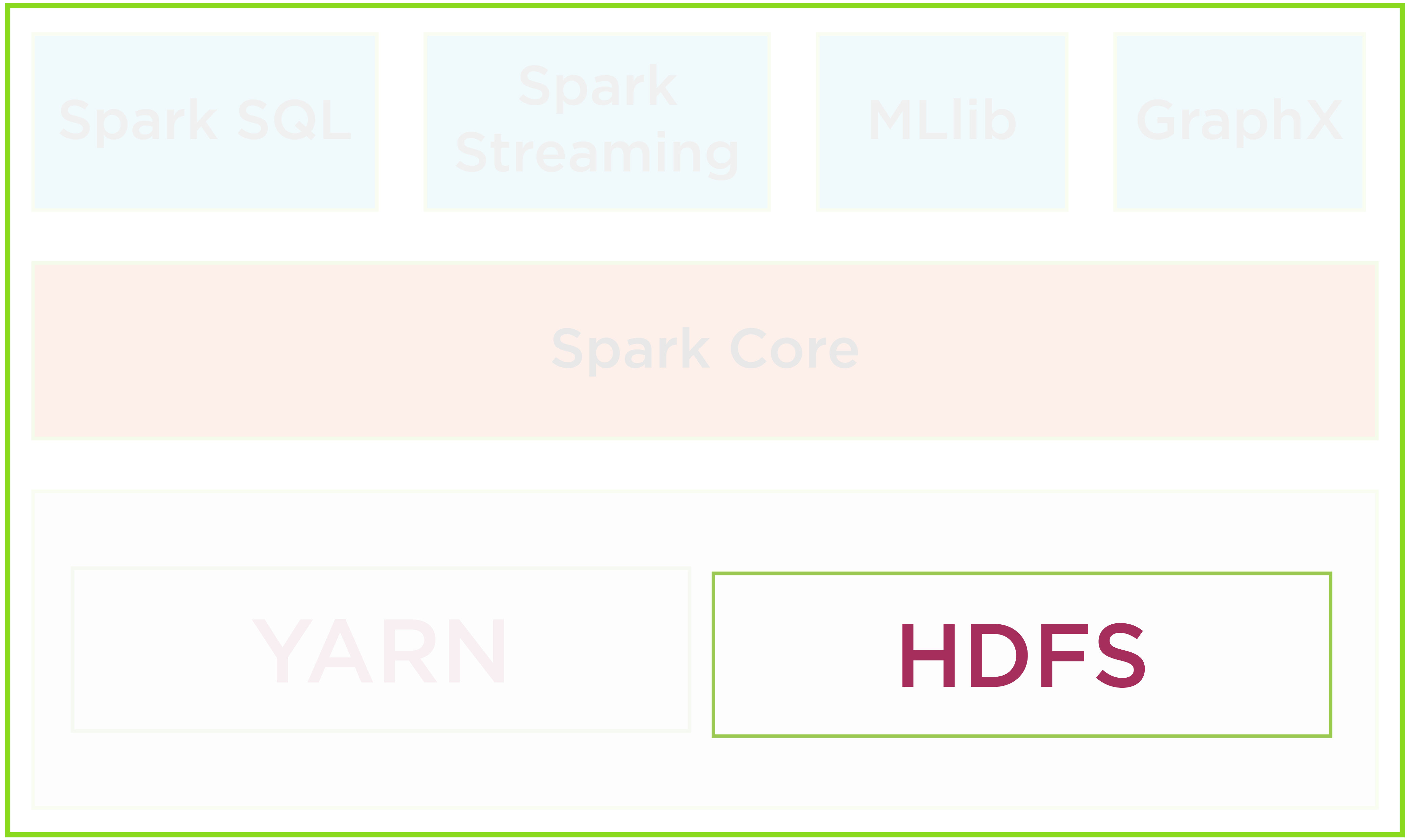


Apache Spark



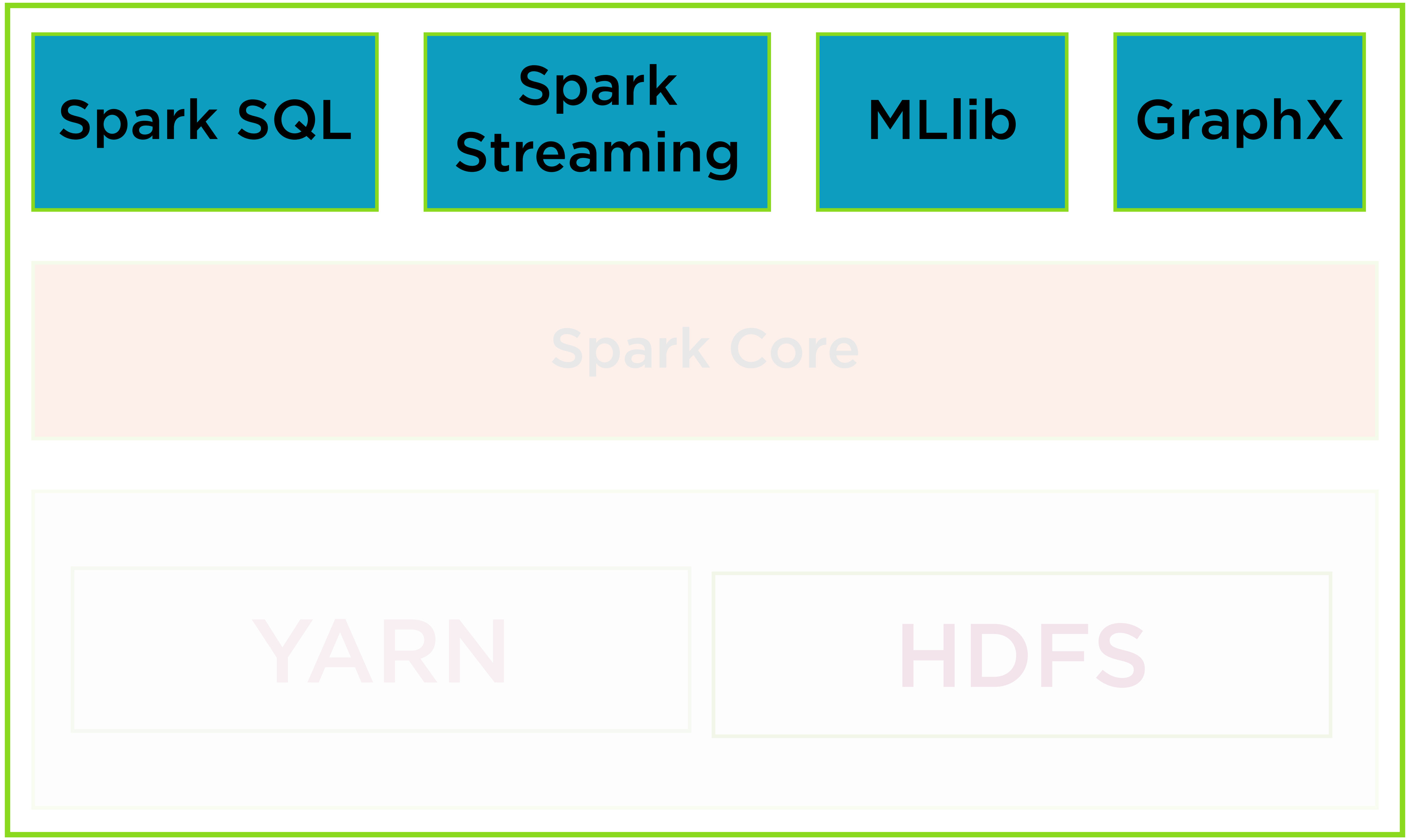
Alternatives: Mesos, Spark Standalone, Kubernetes

Apache Spark



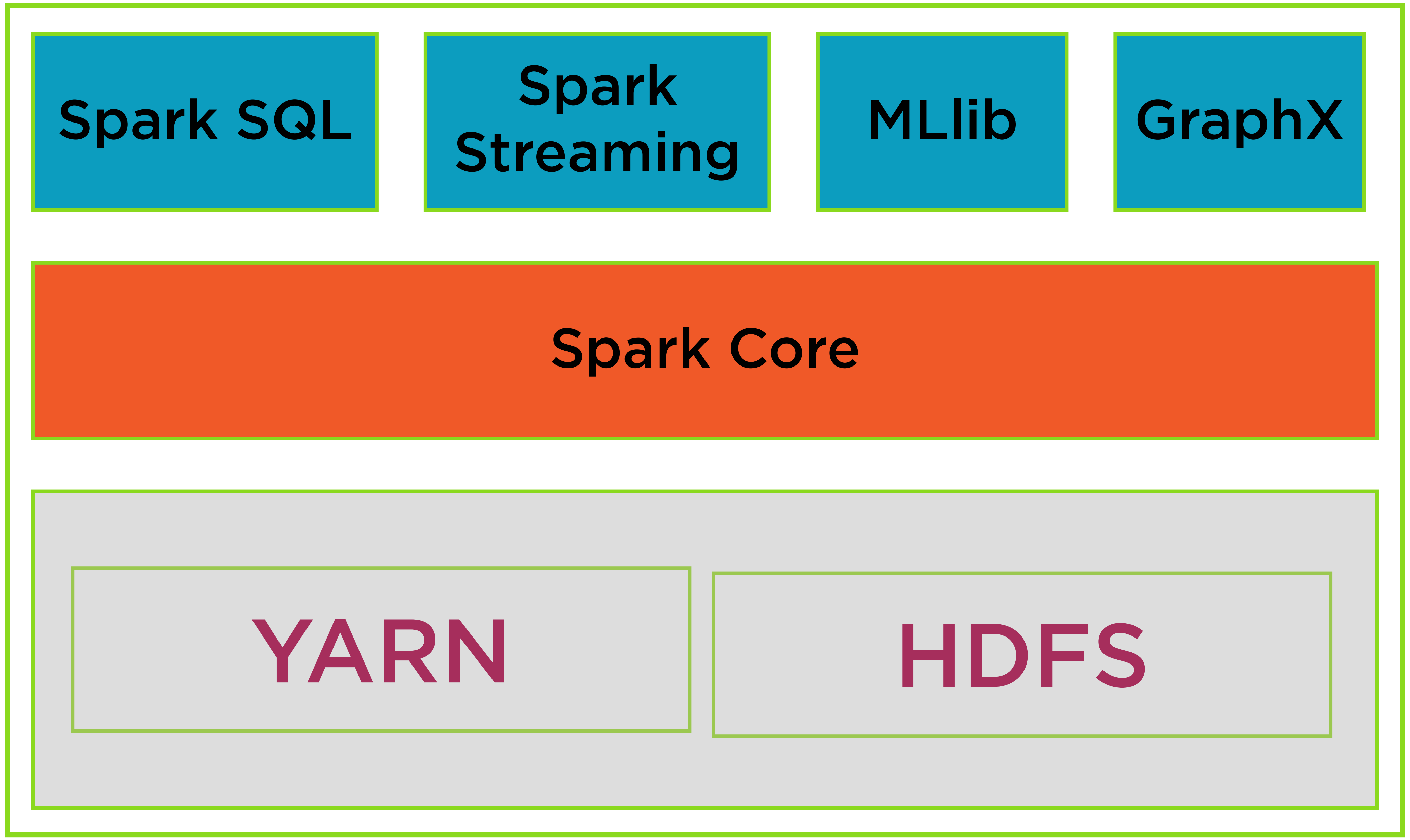
Distributed Storage system

Apache Spark



Spark libraries

Apache Spark

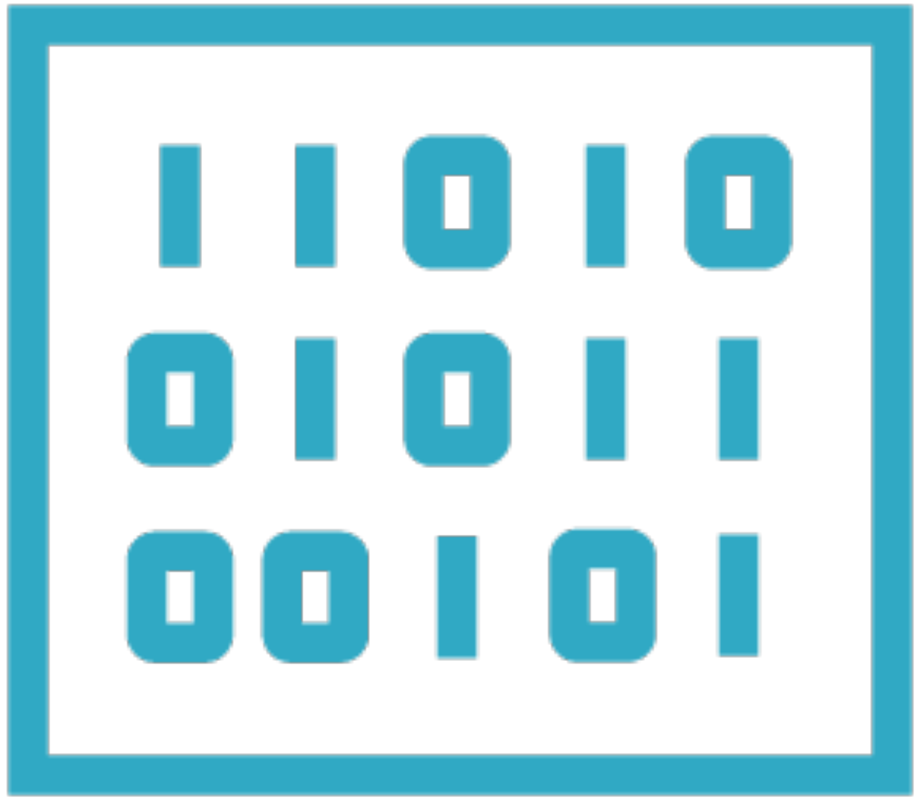


RDDs and DataFrames

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, an RDD is the basic data structure used in Spark 1.x

Why is this relevant in Spark 2?

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

Characteristics of RDDs

Partitioned

RDDs are split across nodes in a cluster

Immutable

RDDs, once created, cannot be changed

Resilient

Can be reconstructed on node crashes

RDDs Support Two Operations

Transformation

Transform input RDDs into another RDD

Action

Request a result, to a file, to console window

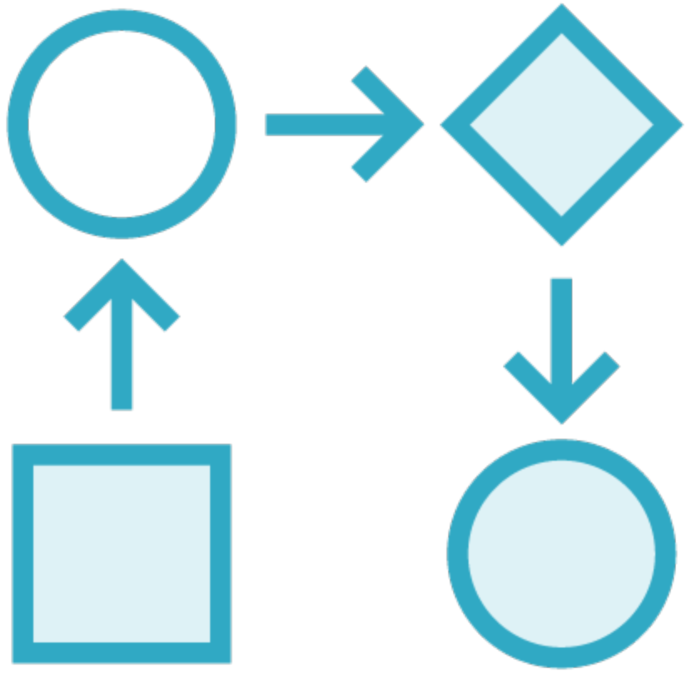
Lazy Evaluation

Spark keeps a record of the series of transformations

Transformations are not performed when defined

Transformations are materialized only when the user requests a result

Lineage



The record of transformations is called lineage

Allows RDDs to be reconstructed in case of node crashes

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

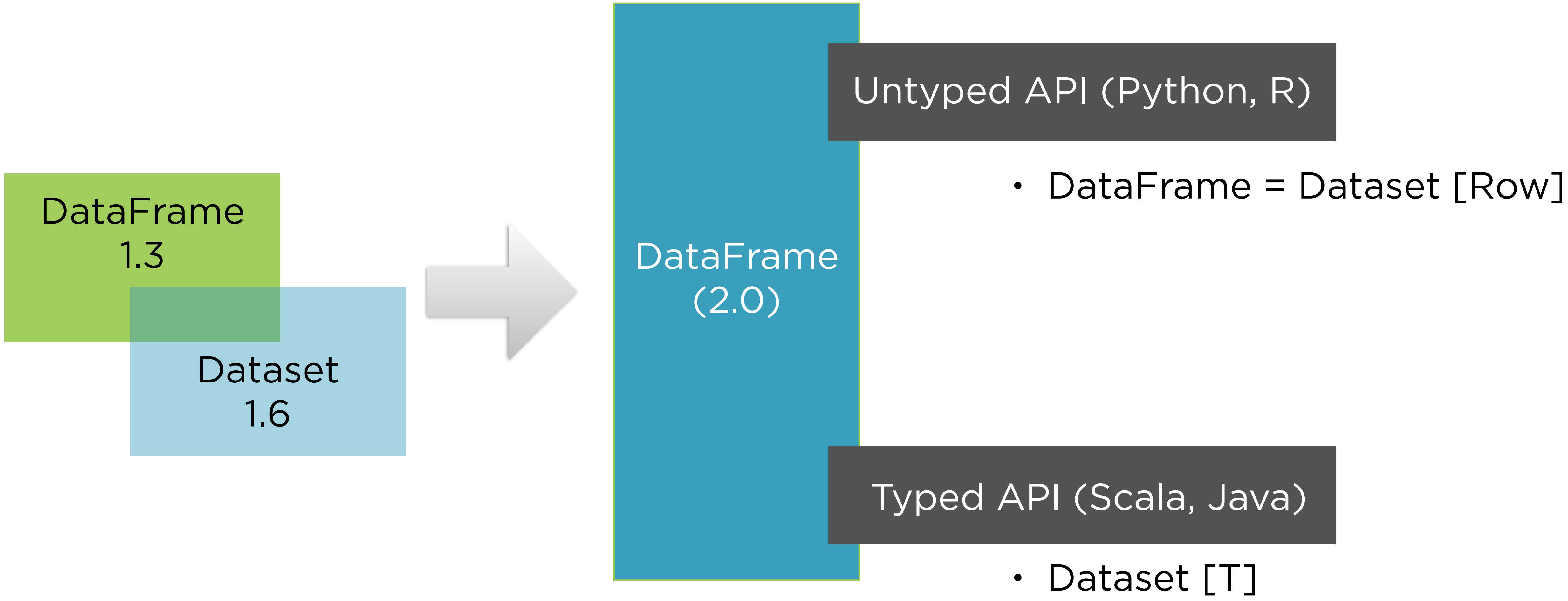
Each row represents 1 observation

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)

Unified API for DataFrames



Untyped API (Python, R)

- DataFrame = Dataset [Row]

Typed API (Scala, Java)

- Dataset [T]

DataFrames Built on Top of RDDs

Partitioned

RDDs are split across nodes in a cluster

Immutable

RDDs, once created, cannot be changed

Resilient

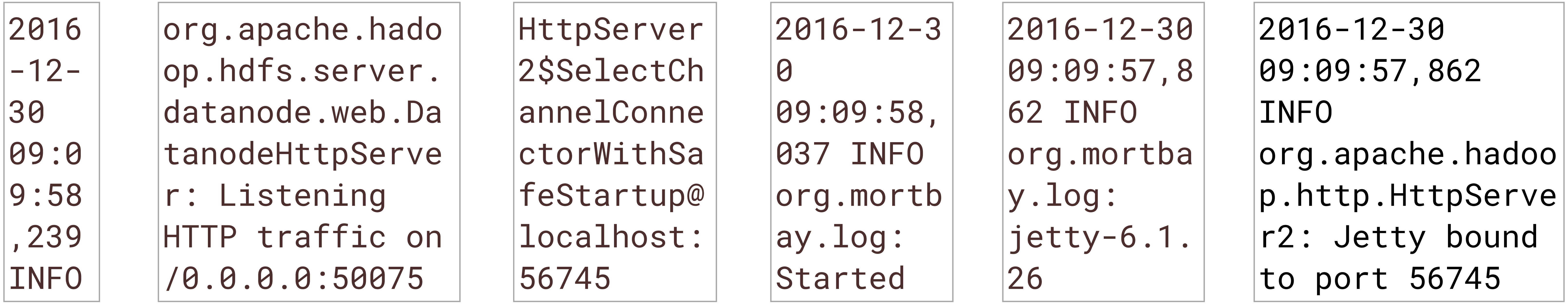
Can be reconstructed on node crashes

Streaming in Spark 1.x and Spark 2.x

Streaming Data Spark 1.x

```
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 Spark 1.x



```
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 log message is one entity in this stream

Streaming Data Spark 1.x

```
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
```

Spark works with stream data using the same batch RDD abstraction

Streaming Data Spark 1.x

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

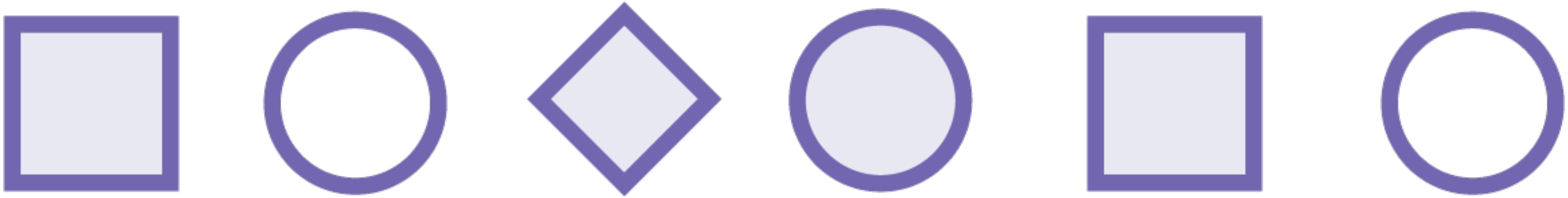
```
org.apache.hado
op.hdfs.server.
datanode.web.Da
tanodeHttpServe
r: Listening
HTTP traffic on
/0.0.0.0:50075
```

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

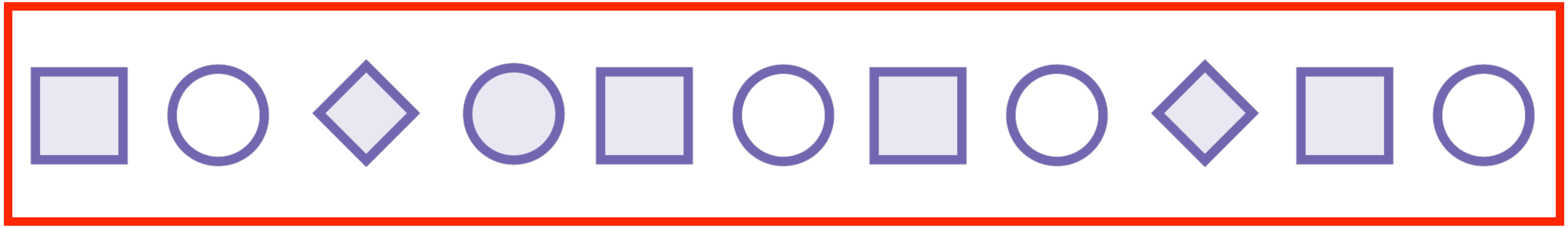
```
2016-12-3
0
09:09:58,
037 INFO
org.mortb
ay.log:
Started
```

```
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 Spark 1.x



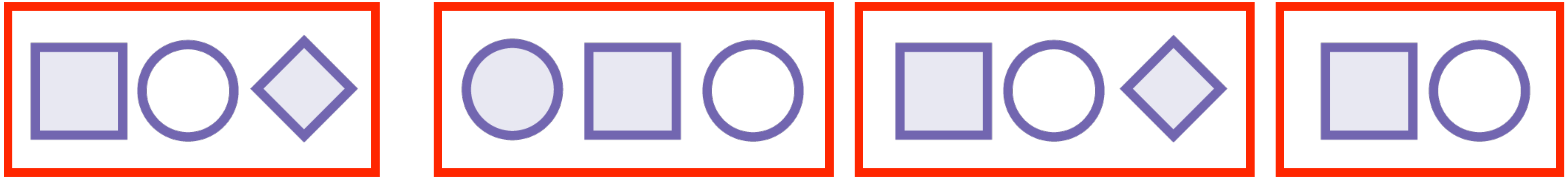
This stream of entities is represented as a **discretized stream or DStream**

DStream = Sequence of RDDs

Streaming Data Spark 1.x

Entities => RDDs => DStream ?

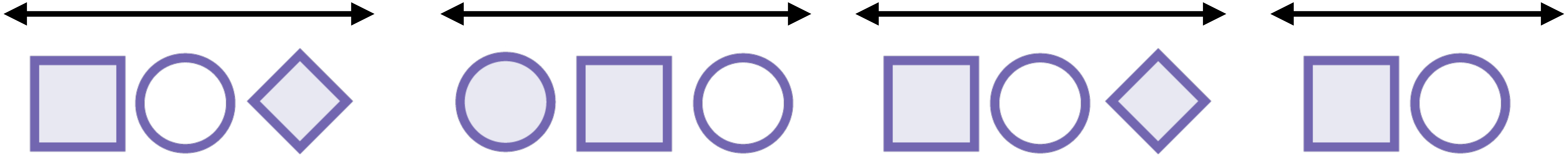
Streaming Data Spark 1.x



Entities in a stream are grouped into batches

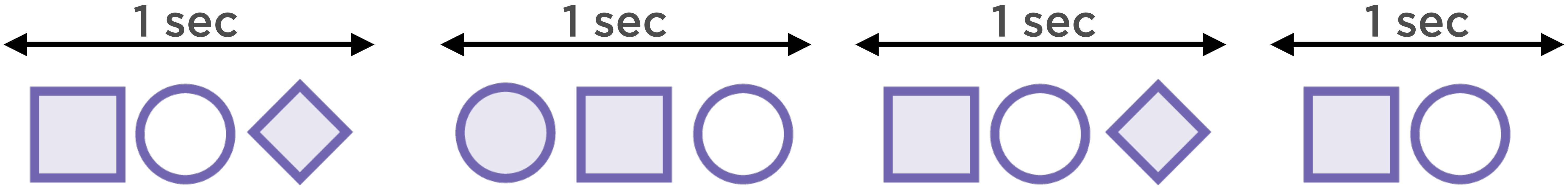
Each batch = 1 RDD

Streaming Data Spark 1.x



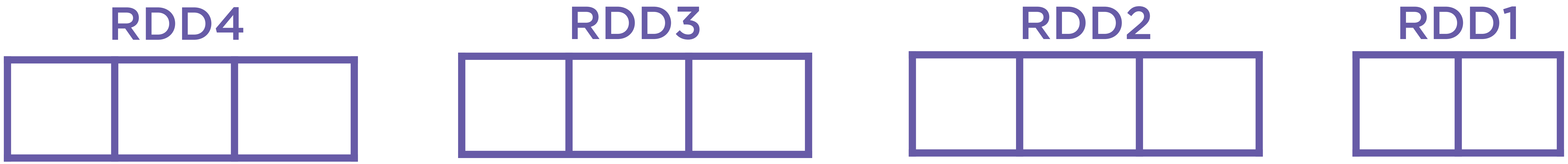
Batches are formed based on a batch interval

Streaming Data Spark 1.x



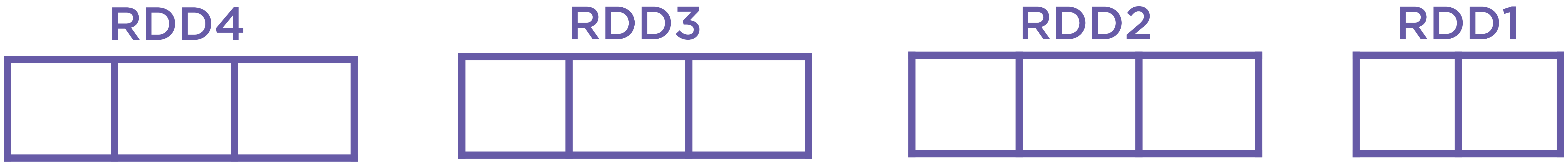
All entities received within the batch interval make one RDD

Streaming Data Spark 1.x



All entities received within the batch interval make one RDD

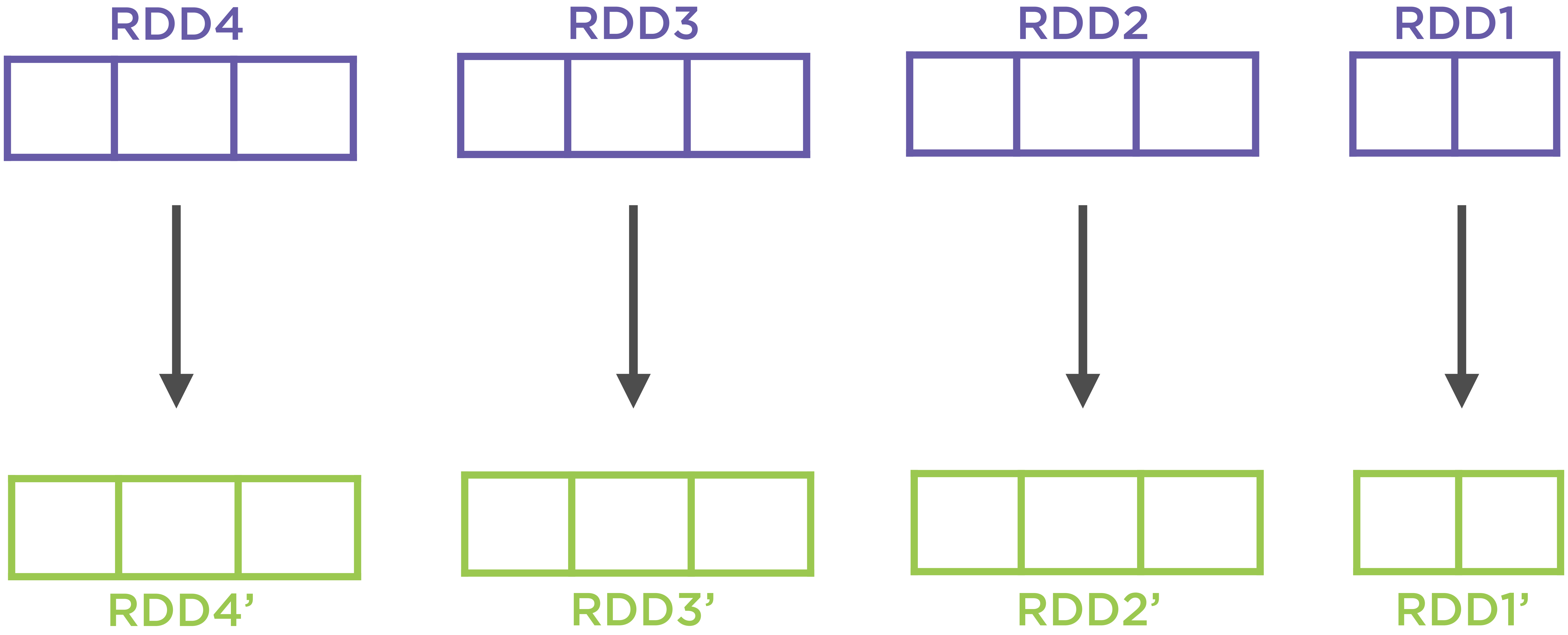
Streaming Data Spark 1.x



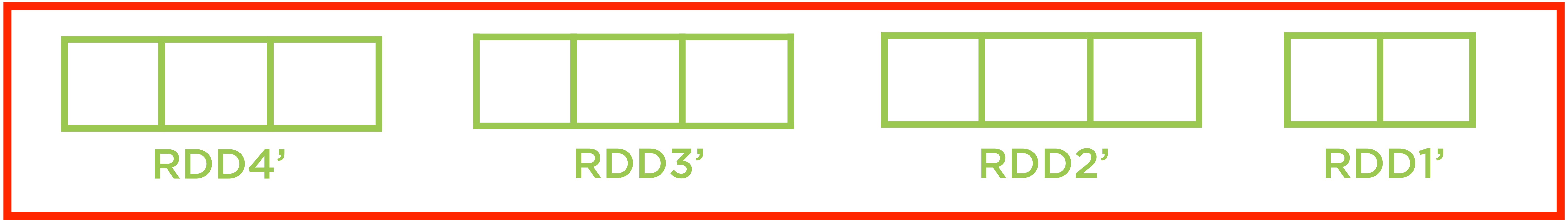
RDD4 RDD3 RDD2 RDD1

Within a DStream, Spark still performs operations on individual RDDs

Streaming Data Spark 1.x



Streaming Data Spark 1.x



Transformed DStream

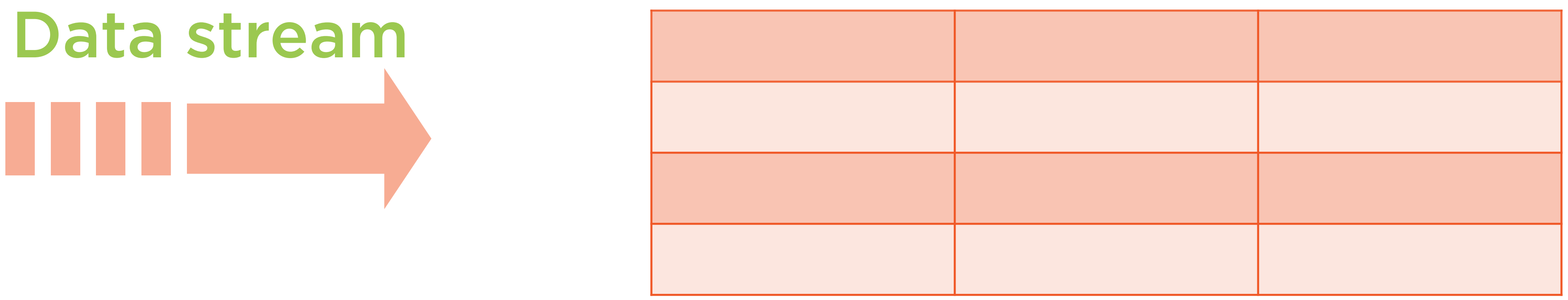
Within DStreams, Spark 1.x does **batch processing on individual RDDs**

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

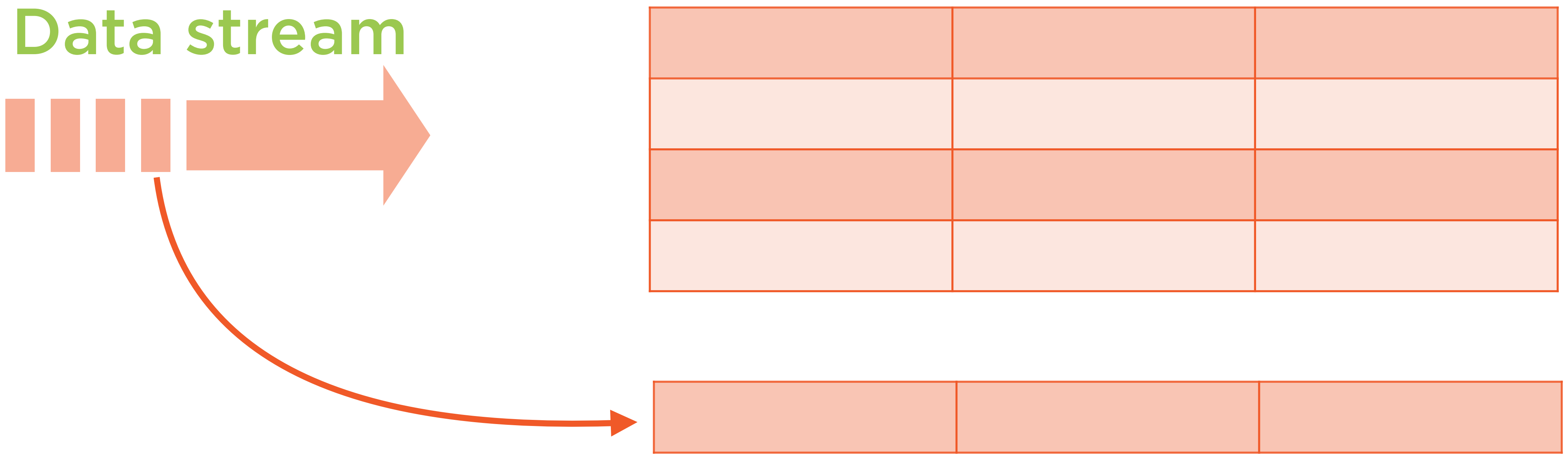
Streaming Data Spark 2.x



Data stream

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

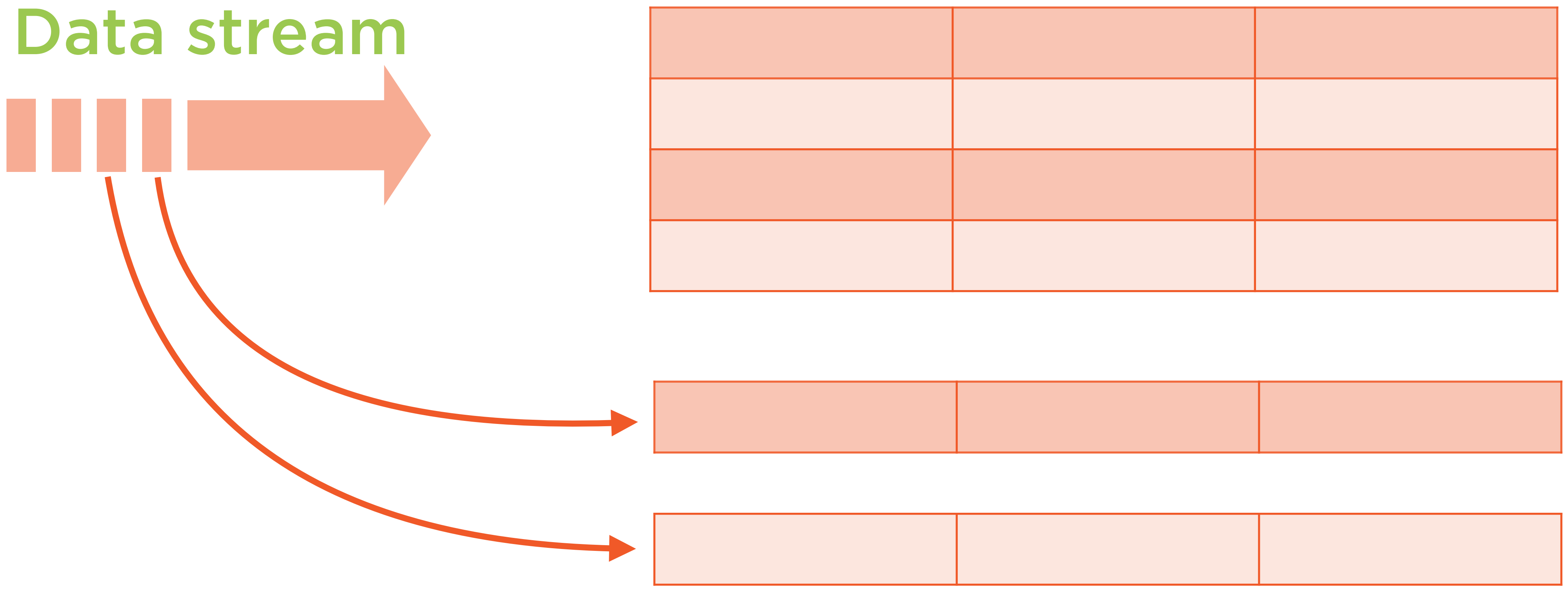
Streaming Data Spark 2.x



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

Data stream as an unbounded input table

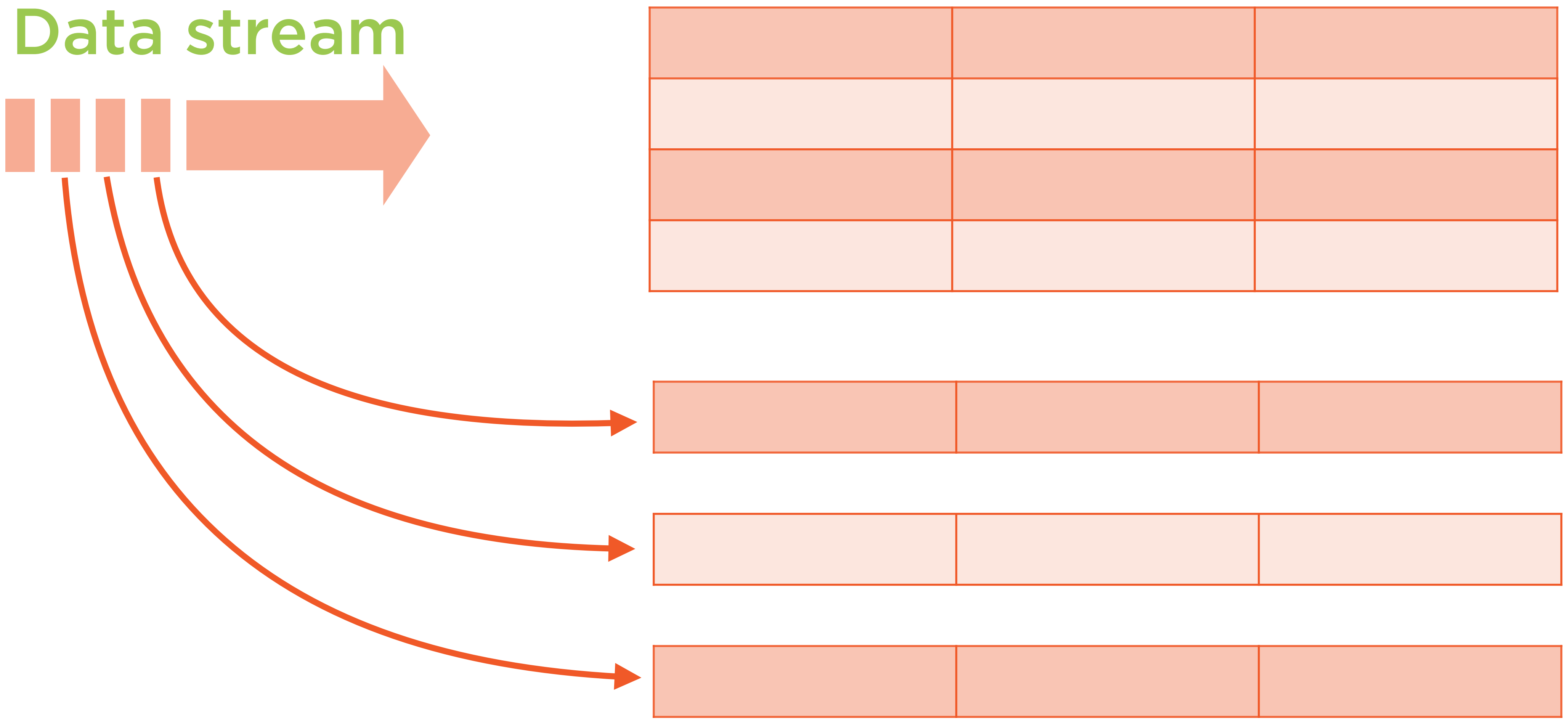
Streaming Data Spark 2.x



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

Data stream as an unbounded input table

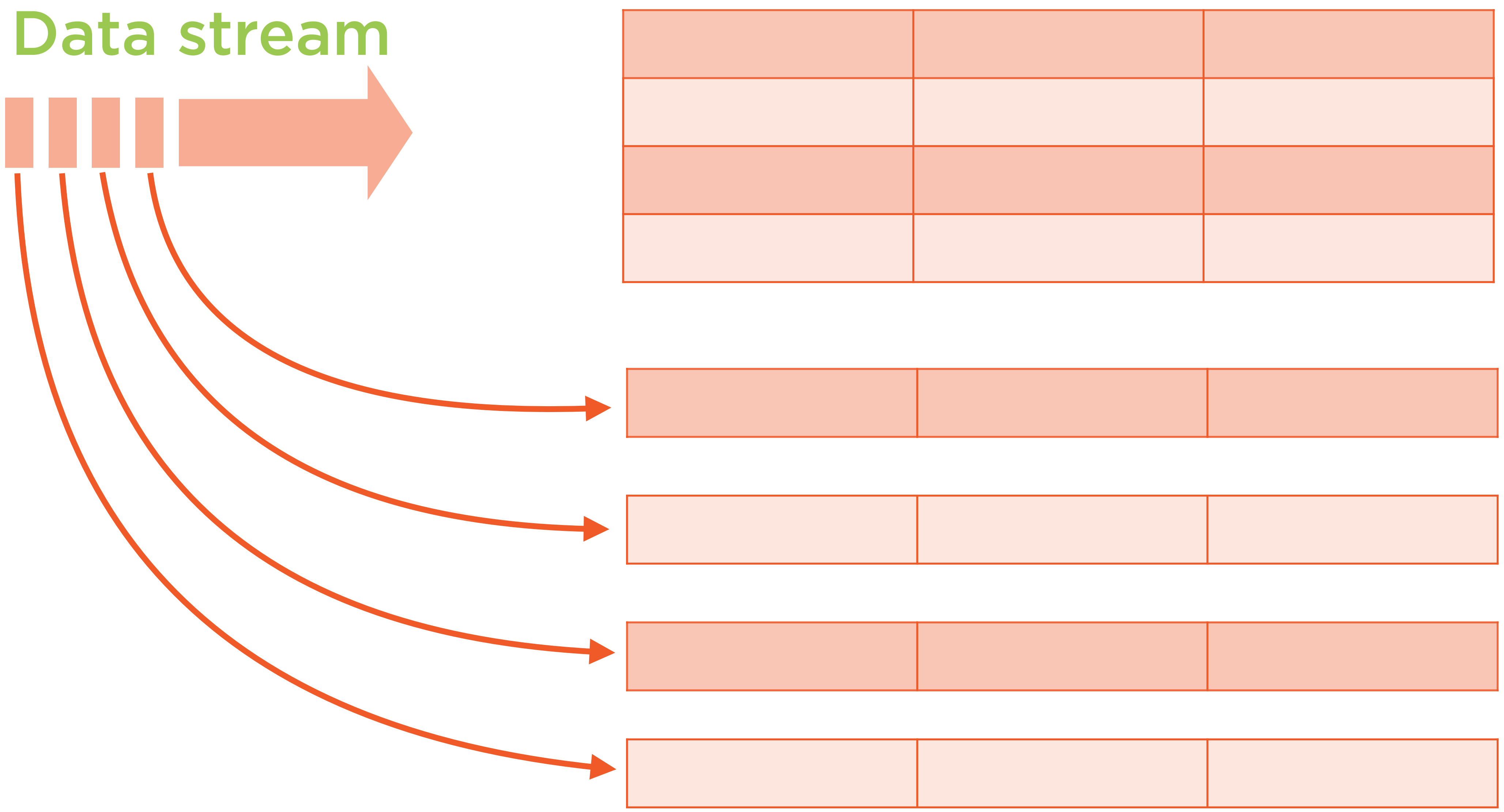
Streaming Data Spark 2.x



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

Data stream as an unbounded input table

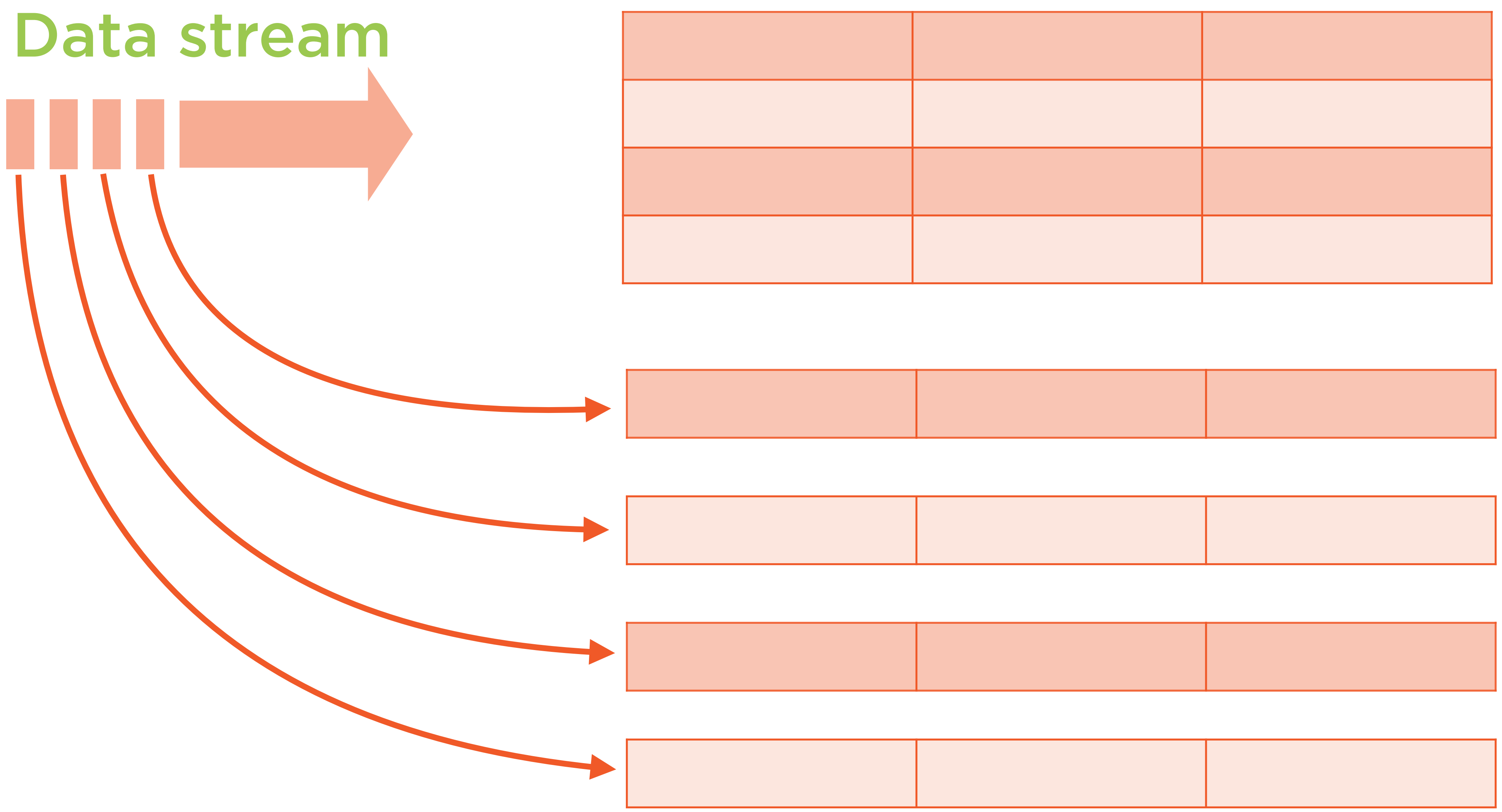
Streaming Data Spark 2.x



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

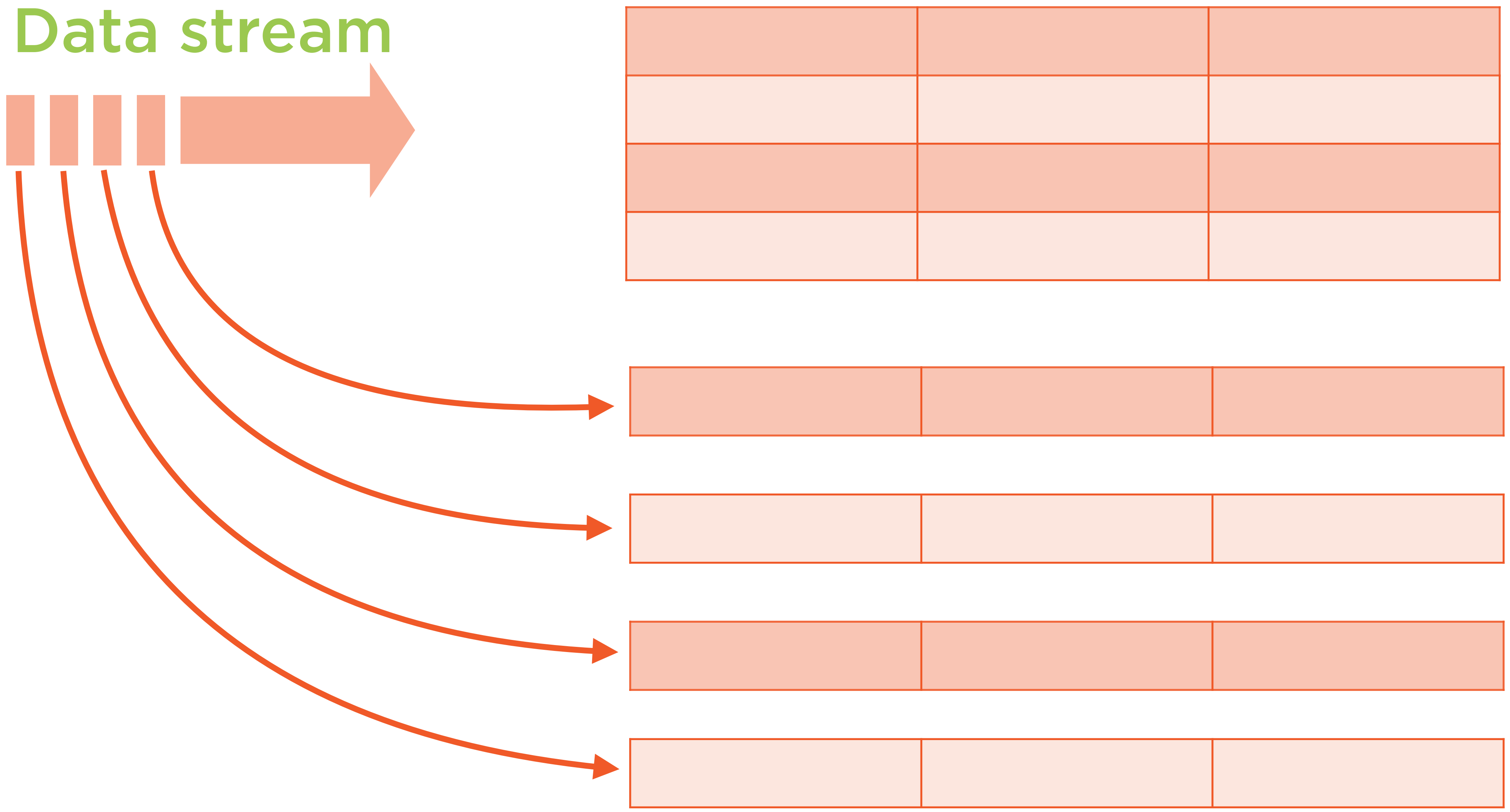
Data stream as an unbounded input table

Batch is Simply A Prefix of Stream



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

Batch is Simply A 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

Structured 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

High-level API in Apache Spark 2.0 that supports continuous applications and replaces Spark Streaming

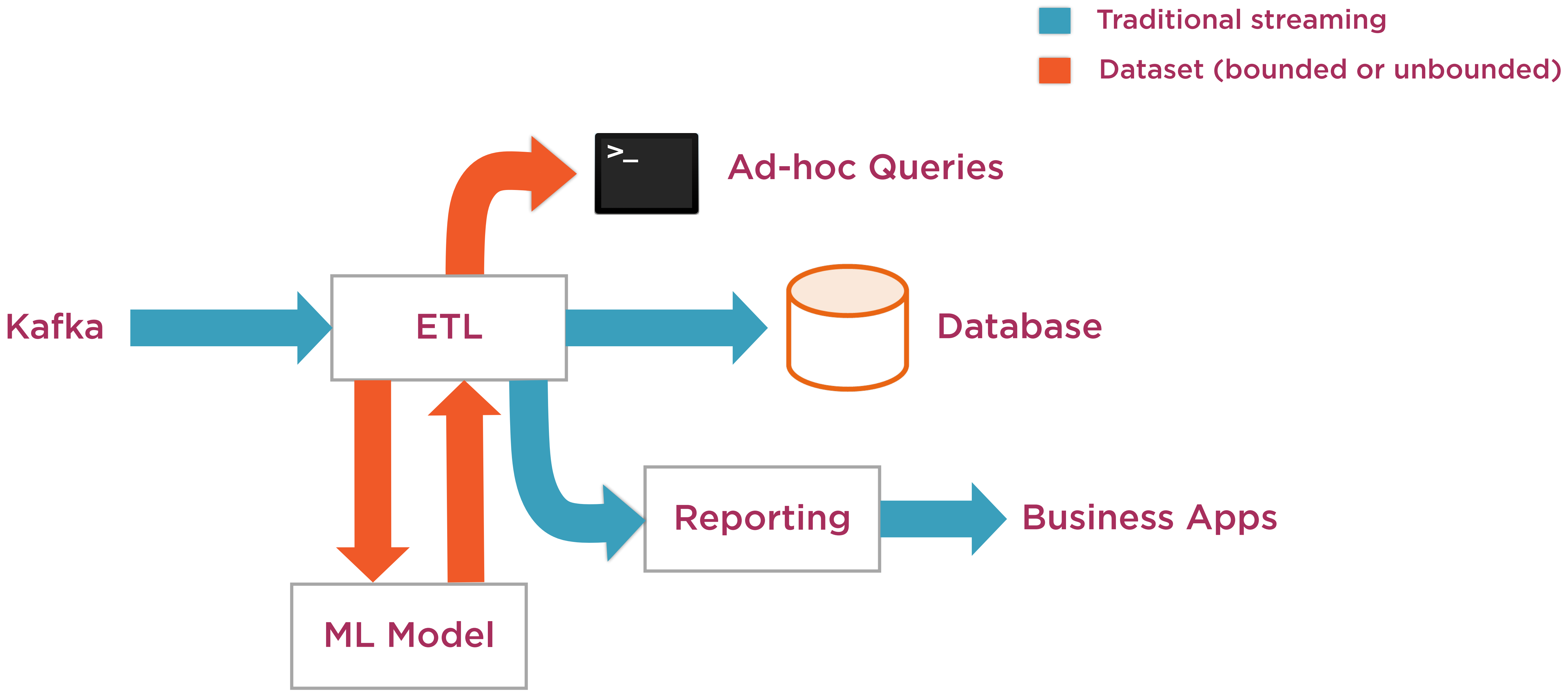
https://databricks.com/blog/2016/07/28/continuous-applications-evolving-streaming-in-apache-spark-2-0.html

Streaming and Structured Streaming

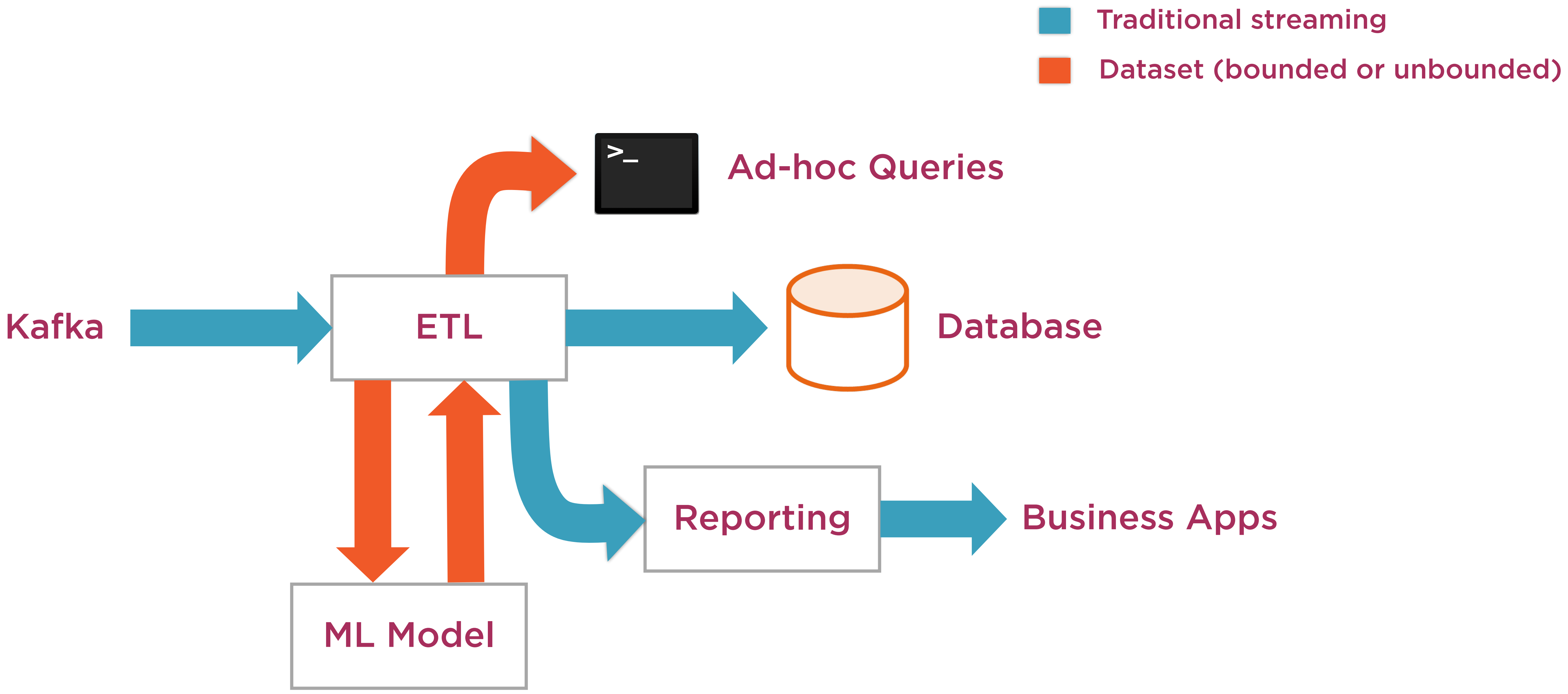
Streaming	Structured Streaming
Older	Newer
RDDs	DataFrames
No optimizations	Optimizations on DataFrames
Batch and streaming support not unified	Unified support for batch and streaming

Structured Streaming

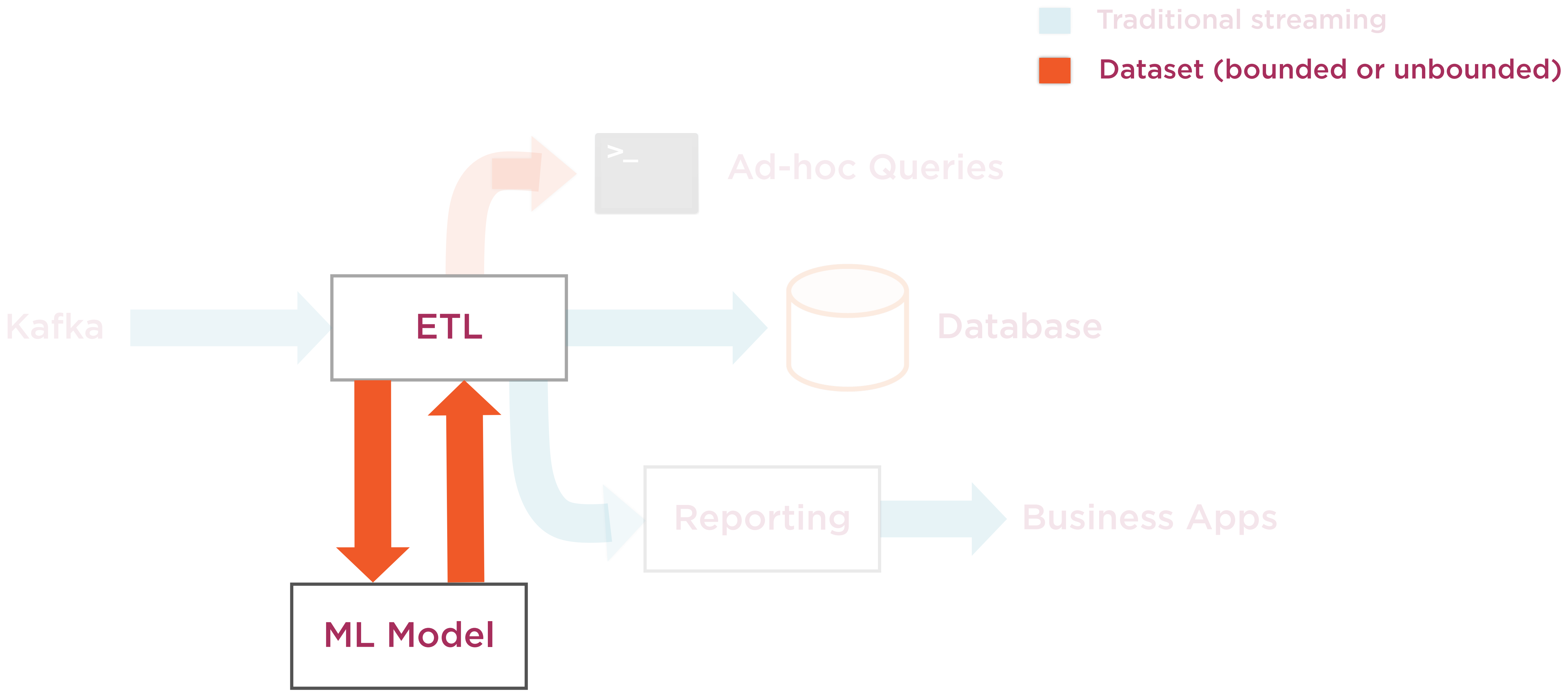
Spark 2.x: Continuous Applications



Structured Streaming

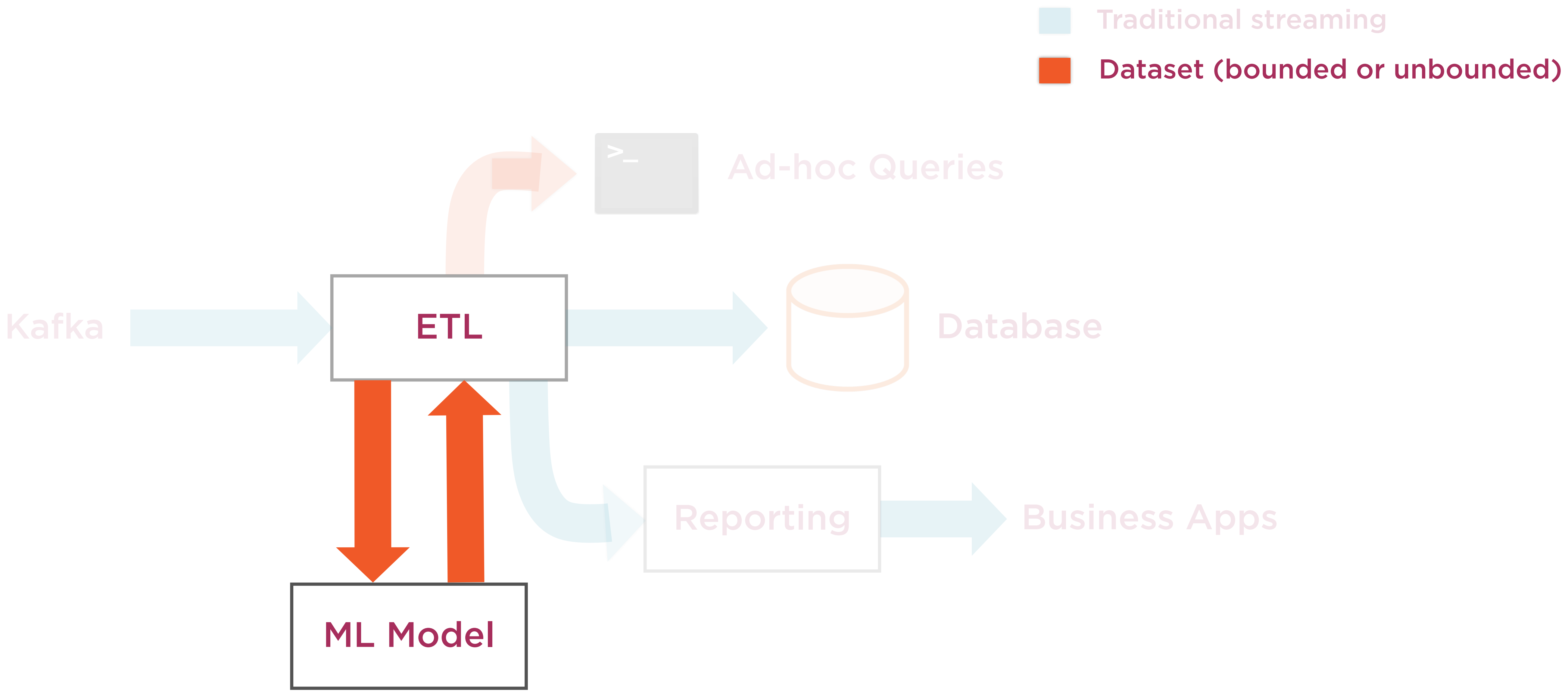


High-level User API



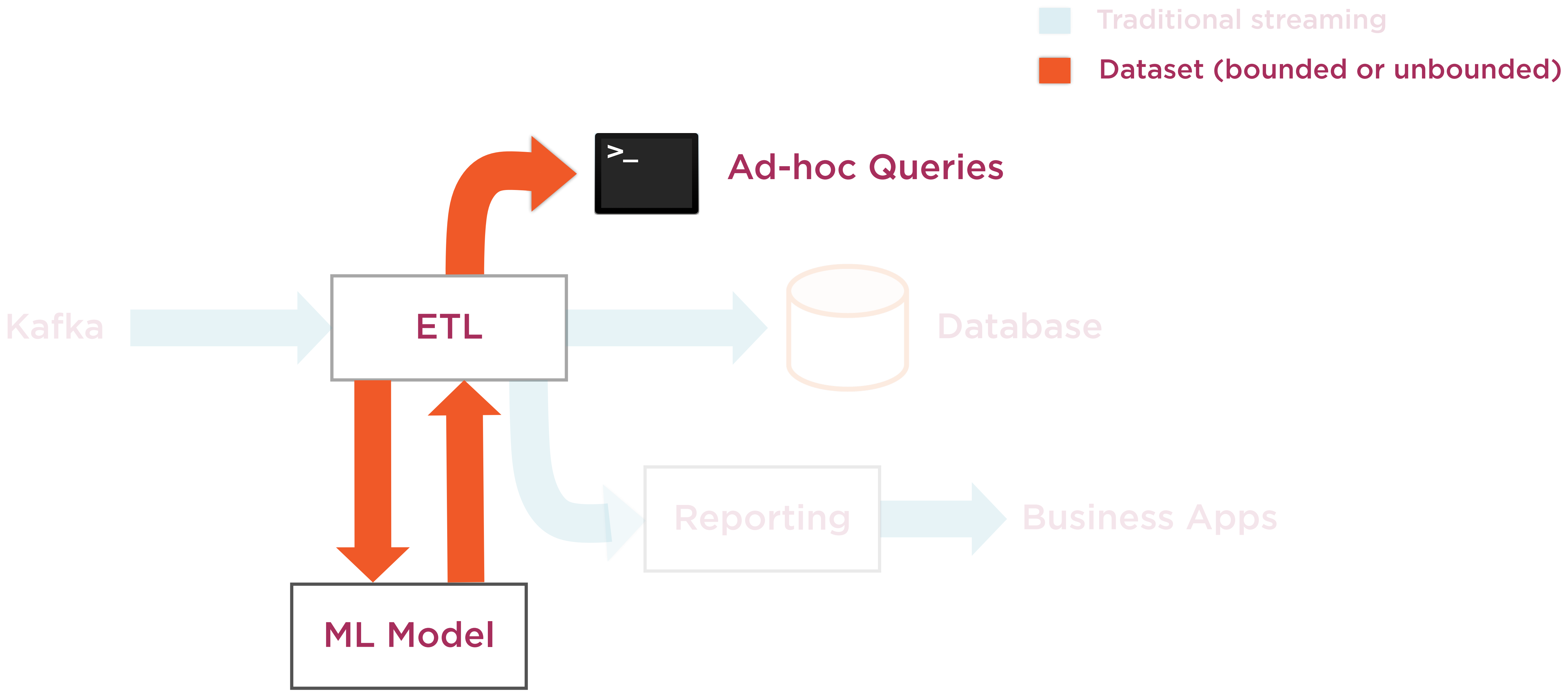
User implements batch computation using DataFrame/Dataset API

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 streaming

Micro-batch Processing

Default stream processing mode in Spark

Data streams processed as a series of batch jobs

End-to-end latencies as low as 100ms

Exactly-once fault-tolerance guarantees

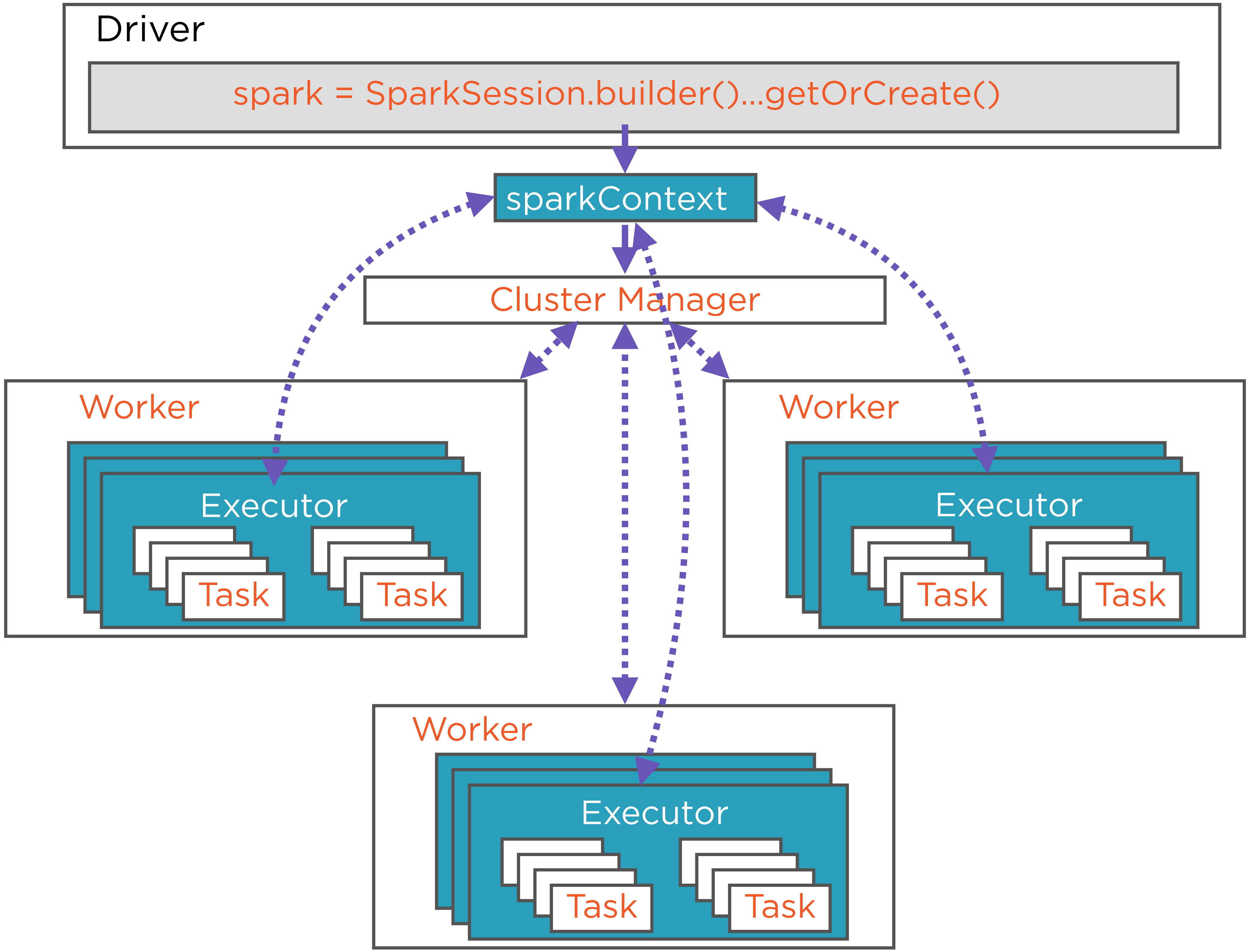
Micro-batch Processing

Since Spark 2.3 there exists a new continuous processing mode

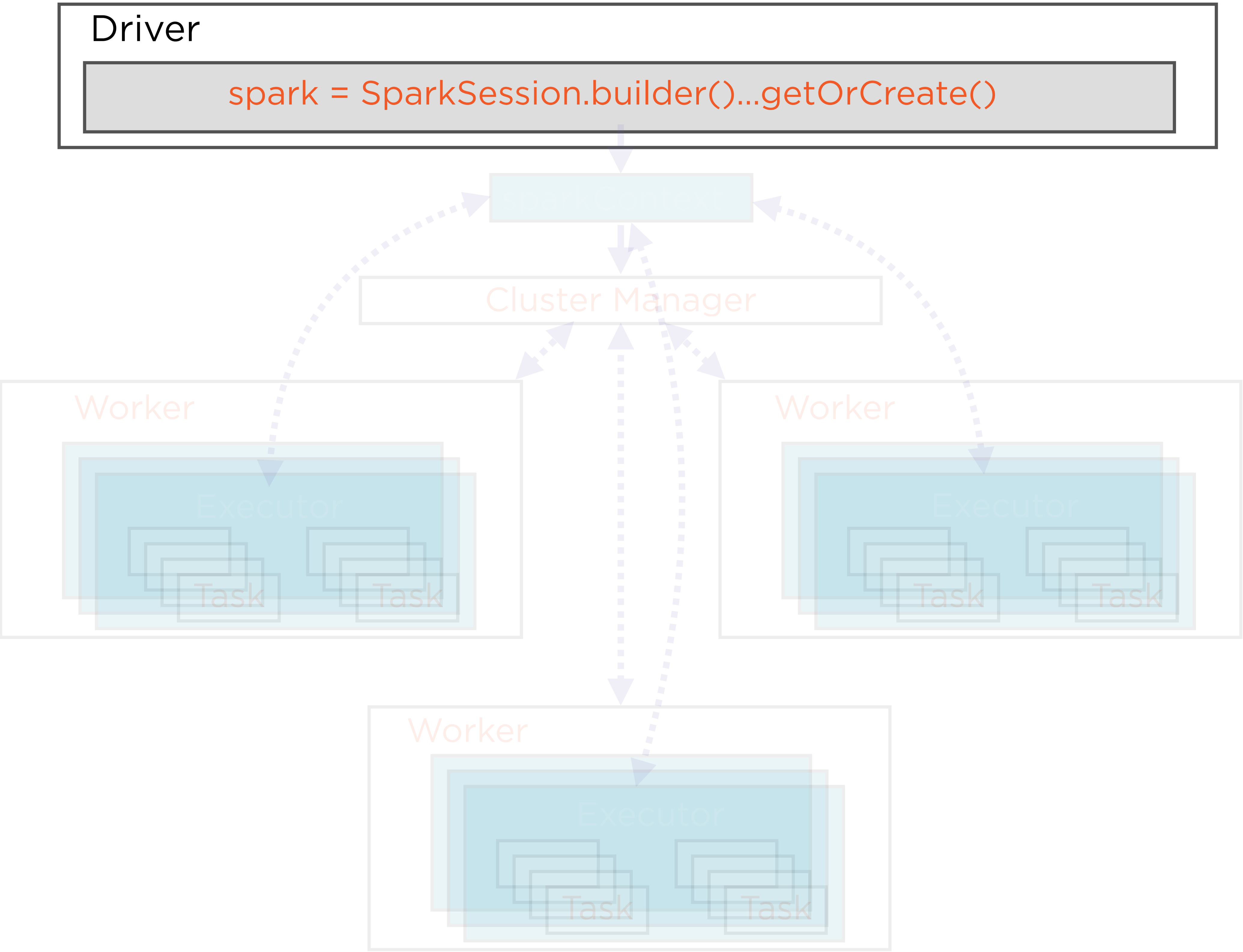
Currently in the experimental stage

Spark Architecture

Spark 2.x Architecture



Spark 2.x Architecture



Driver

spark = SparkSession.builder()...getOrCreate()

Driver

Separate process (JVM)

The master node in a Spark application

Launches tasks

Hosts SparkContext

Driver

Several groups of services run inside the driver

- SparkEnv
- DAGScheduler
- Task Scheduler
- SparkUI
- …

Spark Application

Uses SparkContext as entry point

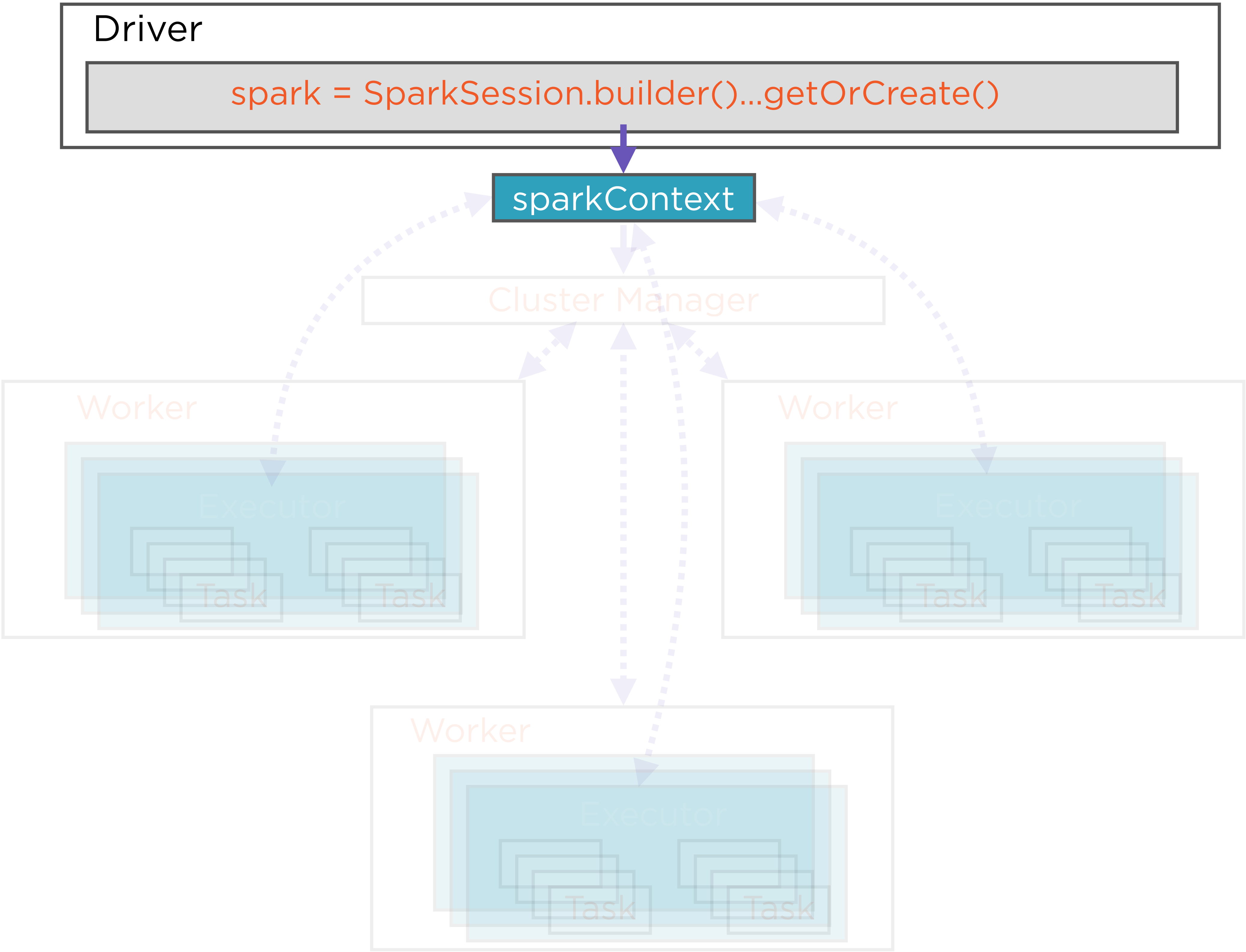
Creates RDD *D*irected *A*cyclic *G*raph

DataFrames run on top of RDDs

Internally, Spark creates *Stages* (physical execution plan)

Each stage is split into operations on RDD partitions called *Tasks*

Spark 2.x Architecture



SparkContext (Spark 1.x)

Familiar code entry point to Spark

```
sc = new sparkContext(…)
```

Create RDDs, accumulators…

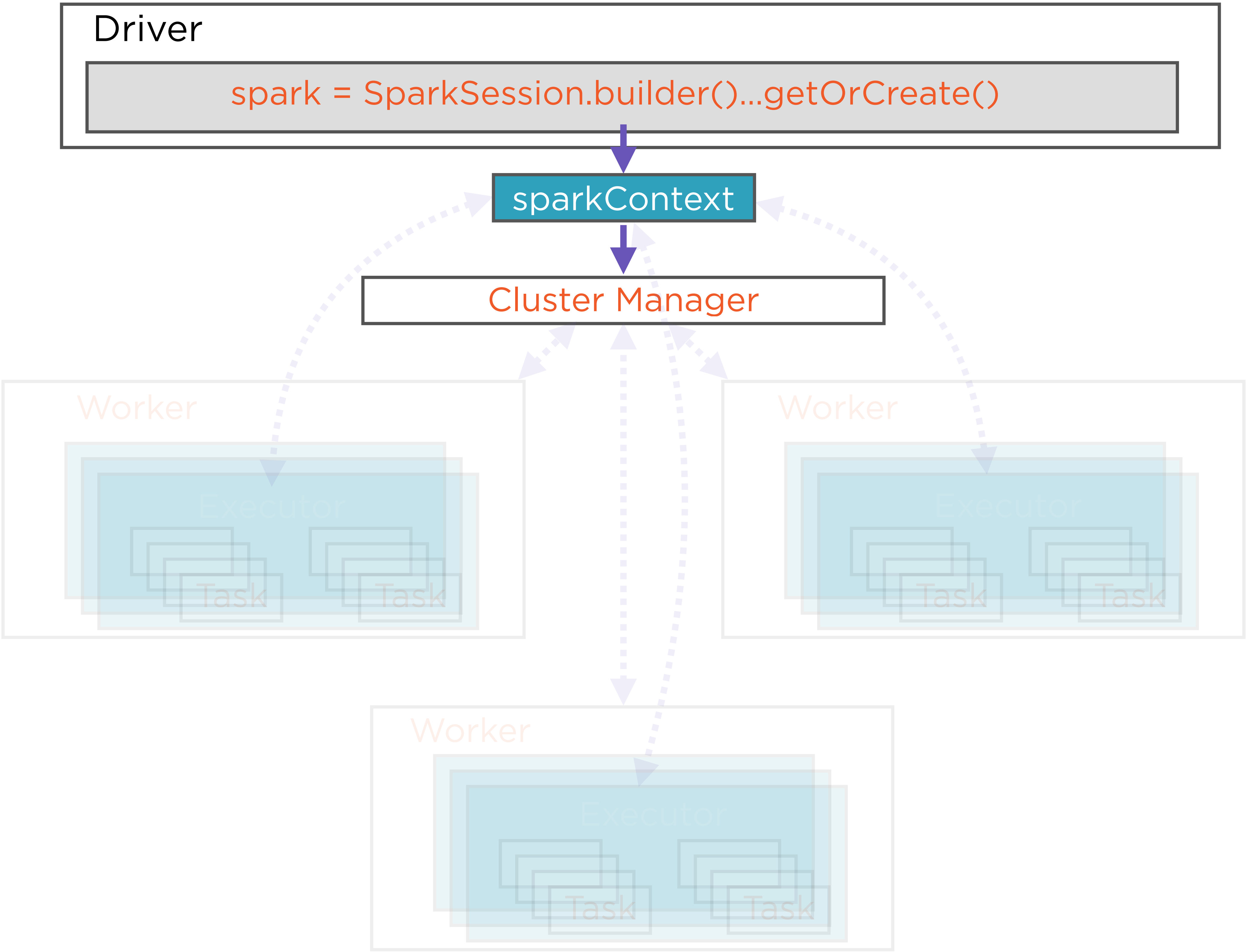
Run jobs

SparkSession (Spark 2.x)

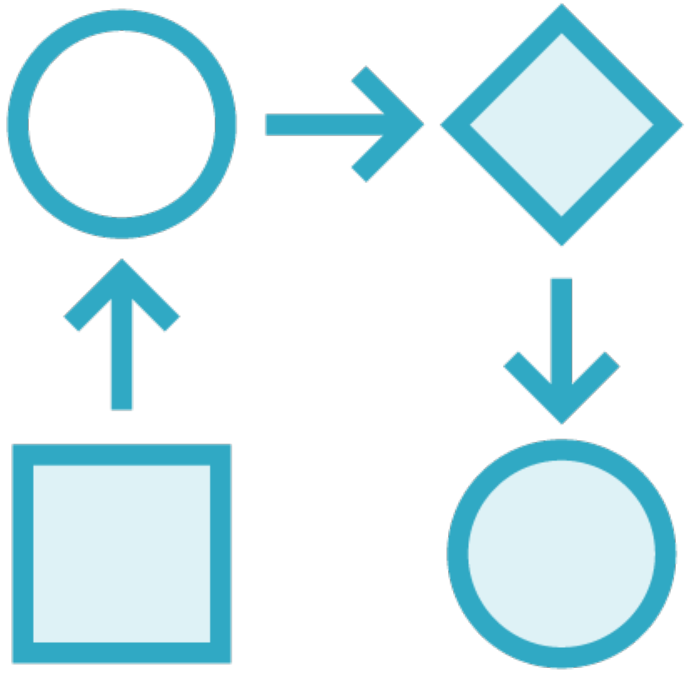
In Spark 2.x, SparkContext is wrapped in SparkSession

Encapsulates SparkContext, SQLContext, HiveContext...

Spark 2.x Architecture



Cluster Manager



Hadoop's YARN

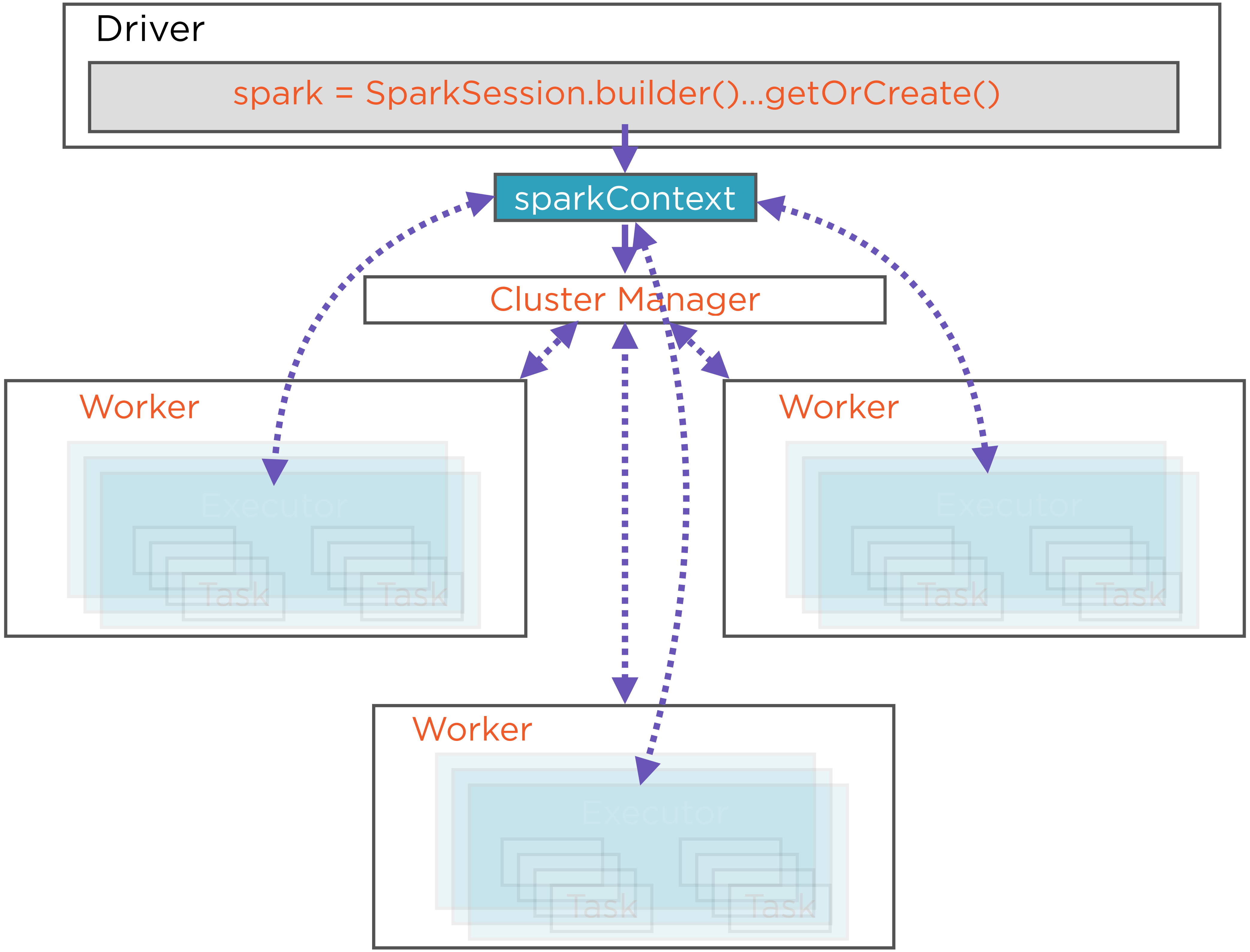
Apache Mesos

Kubernetes

Spark Standalone

Orchestrates execution

Spark 2.x Architecture



Driver

spark = SparkSession.builder()...getOrCreate()

sparkContext

Cluster Manager

Worker

Executor

Task

Task

Worker

Executor

Task

Task

Worker

Executor

Task

Task

Worker

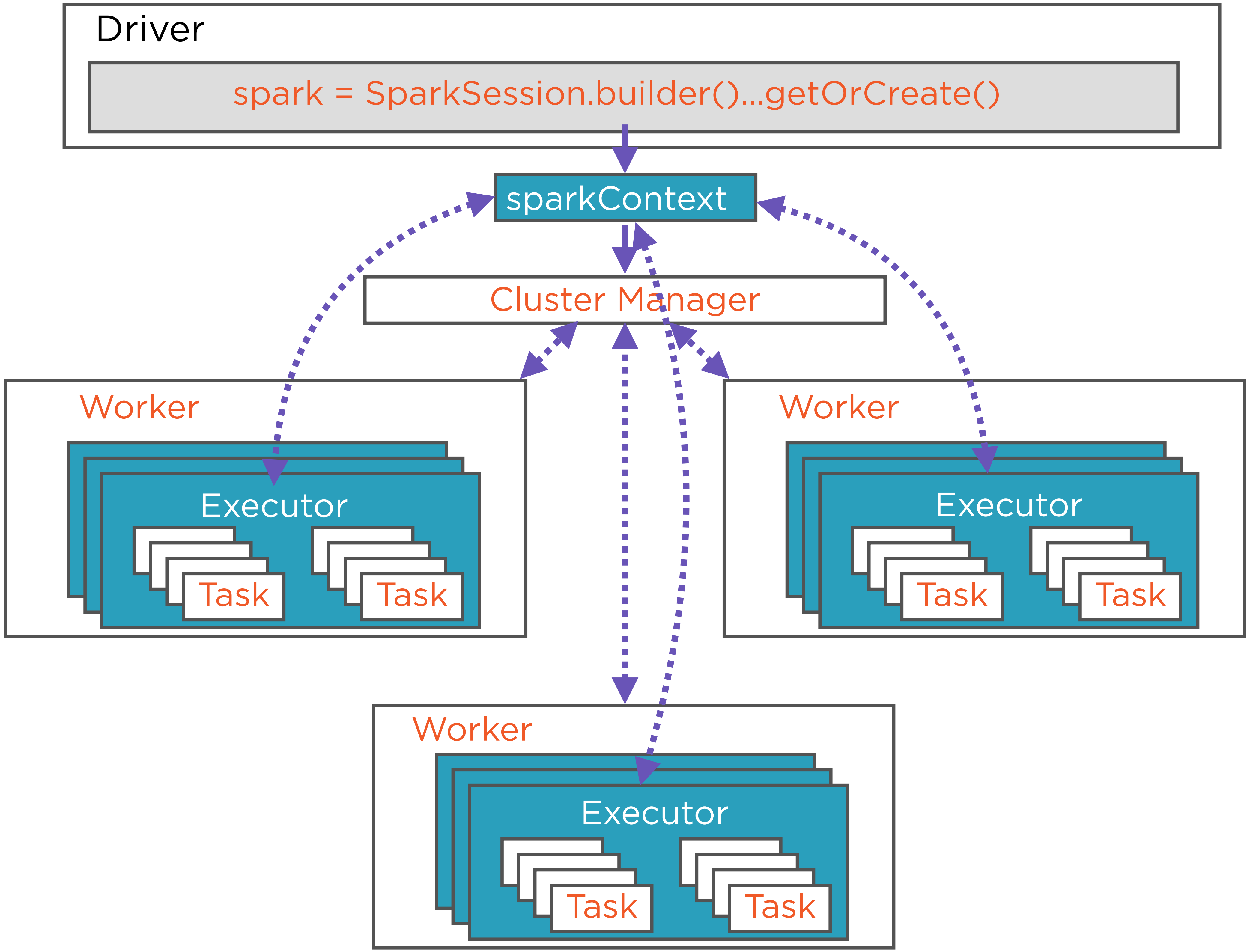
Compute nodes in cluster

Runs the Spark application code

When SparkContext created...

...Each worker starts *executors*

Spark 2.x Architecture



Executors

Distributed agents that execute *tasks*

Tasks are basic units of execution

Tasks belong inside *stages*

Stages are physical units of execution

Demo

Install and set up Apache Spark on MacOS

Demo

Install and set up Apache Spark on Windows

Demo

Streaming word count with a socket source and a console sink

Demo

Aggregations on streaming data with a socket source and console sink

Summary

Streaming and its place in the Apache Spark stack

RDDs and DataFrames

Structured Streaming in Spark 2.x

Batch as a prefix of stream

ClusterManager, SparkSession, and Executors

Up Next:

Executing Streaming Queries