# Online Learning with Structured Streaming

Ram Sriharsha, Vlad Feinberg
@halfabrane

Spark Summit, Brussels
27 October 2016

databricks™

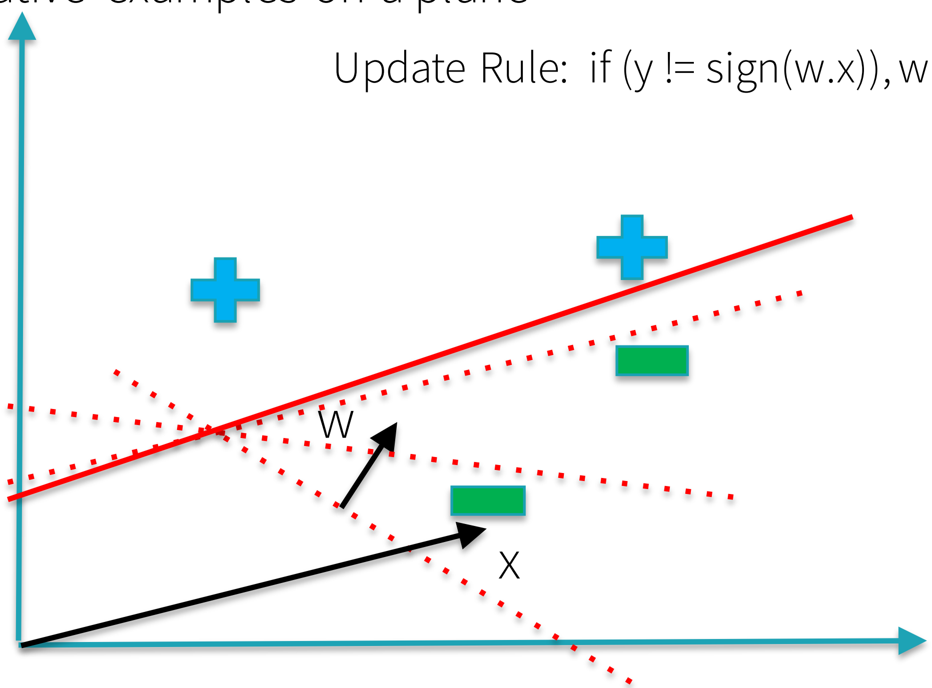
# What is online learning?

- Update model parameters on each data point
  - In batch setting get to see the entire dataset before update
- Cannot visit data points again
  - In batch setting, can iterate over data points as many times as we want!

# An example: the perceptron

Goal: Find the best line separating positive
From negative examples on a plane

Update Rule: if (y != sign(w.x)), w -> w + y(w.x)



# Why learn online?

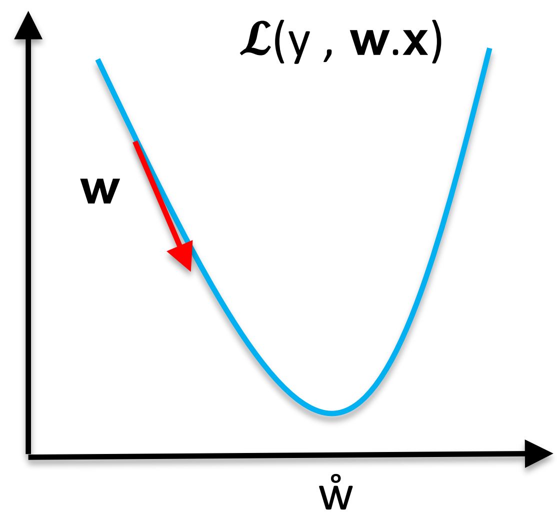
- I want to adapt to changing patterns *quickly*
  - data distribution can change
    - e.g, distribution of features that affect learning might change over time
- I need to learn a good model *within resource + time* constraints (*large-scale learning)*
  - Time to a given accuracy might be faster for certain online algorithms

# Online Classification Setting

- Pick a hypothesis
- For each labeled example ($\mathbf{x}$, y):
  - Predict label ỹ using hypothesis
  - Observe the loss $\mathcal{L}$(y, ỹ) (and its gradient)
  - Learn from mistake and update hypothesis
- Goal: to make as few mistakes as possible in comparison to the *best* hypothesis in *hindsight*

# An example: Online SGD

- Initialize weights **w**
- Loss function $\mathcal{L}$ is known.
- For each labeled example (**x**, y):
  - Perform update **w** -> **w** – $\eta \nabla \mathcal{L}$(y , **w**.**x**)
- For each new example x:
  - Predict $\tilde{y}$ = $\sigma$(**w**.**x**) ($\sigma$ is called link function)




databricks

# Distributed Online Learning

- *Synchronous*
  - On each worker:
    - Load training data, compute gradients and update model, push model to driver
  - On some node:
    - Perform model merge
- Asynchronous
  - On each worker:
    - Load training data, compute gradients and push to server
  - On each server:
    - Aggregate the gradients, perform update step

# Challenges

- Not all algorithms admit *efficient* online versions
- Lack of infrastructure
  - (Single machine) Vowpal Wabbit works great but hard to use from Scala, Java and other languages.
  - (Distributed) No implementation that is *fault tolerant*, *scalable*, *robust*
- Lack of framework in open source to provide extensible algorithms
  - Adagrad, normalized learning, L1 regularization,. . .
  - Online SGD, FTRL, ...

# Structured Streaming

# Structured Streaming

1. One single API `DataFrame` for everything
   - Same API for machine learning, batch processing, graphX
   - Dataset is a typed version of DataFrame for Scala and Java

2. End-to-end exactly-once guarantees
   - The guarantees extend into the sources/sinks, e.g. MySQL, S3

3. Understands external event-time
   - Handling late arriving data
   - Support sessionization based on event-time

# How does it work?

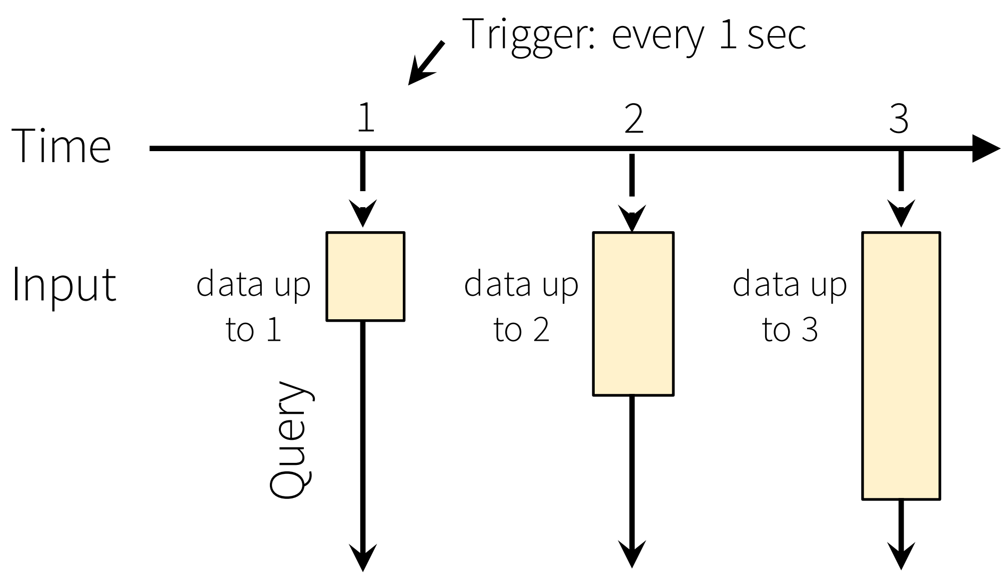
*at any time, the output of the application is equivalent to executing a batch job on a prefix of the data*

# The Model

**Input:** data from source as an append-only table

**Trigger:** how frequently to check input for new data

**Query:** operations on input
usual map/filter/reduce
new window, session ops

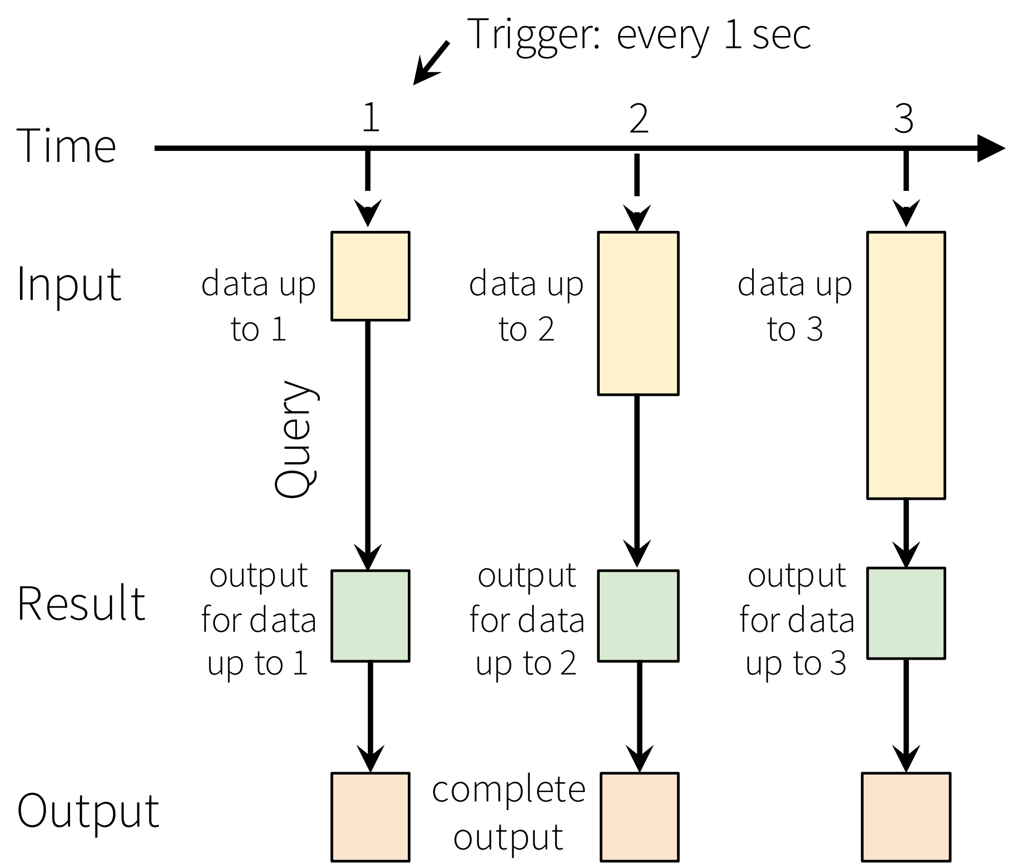


# The Model

**Result:** final operated table
updated every trigger interval

**Output:** what part of result to write
to data sink after every trigger

*Complete output:* Write full result table every time



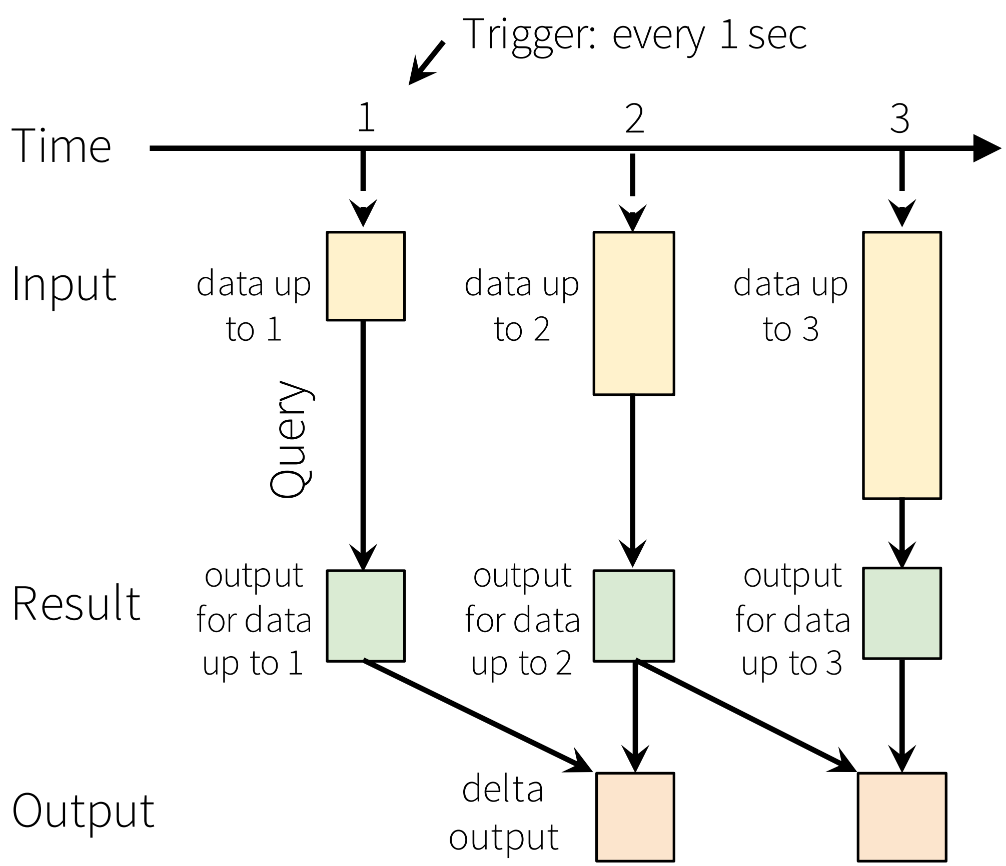
# The Model

**Result:** final operated table
updated every trigger interval

**Output:** what part of result to write
to data sink after every trigger

*Complete output:* Write full result table every time
*Delta output:* Write only the rows that changed
in result from previous batch
*Append output:* Write only new rows

*Not all output modes are feasible with all queries



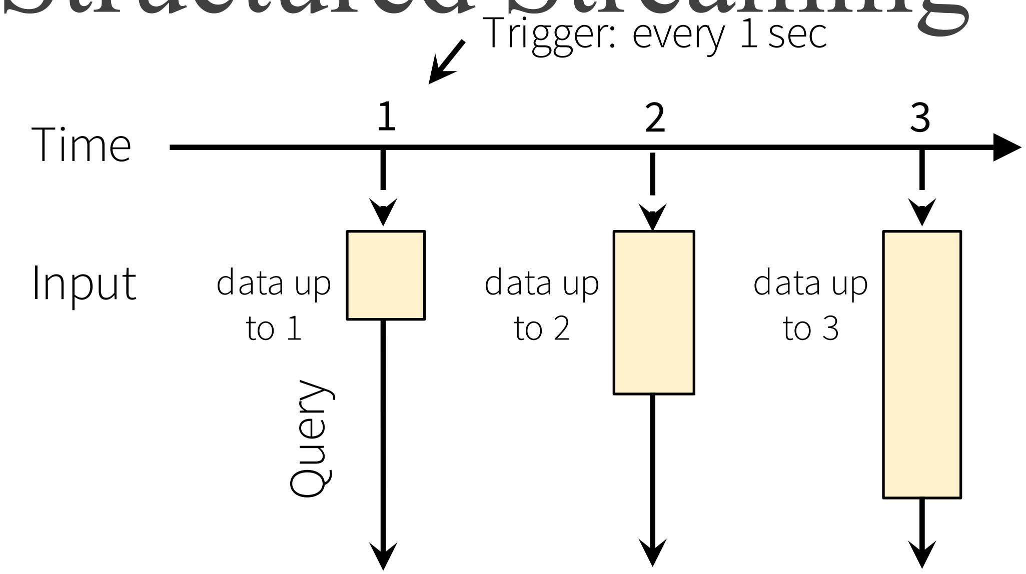

databricks

# Streaming ML on Structured Streaming

# Streaming ML on Structured Streaming

**Input:** append only table containing labeled examples

**Query:** Stateful aggregation query: picks up the last trained model, performs a distributed update + merge

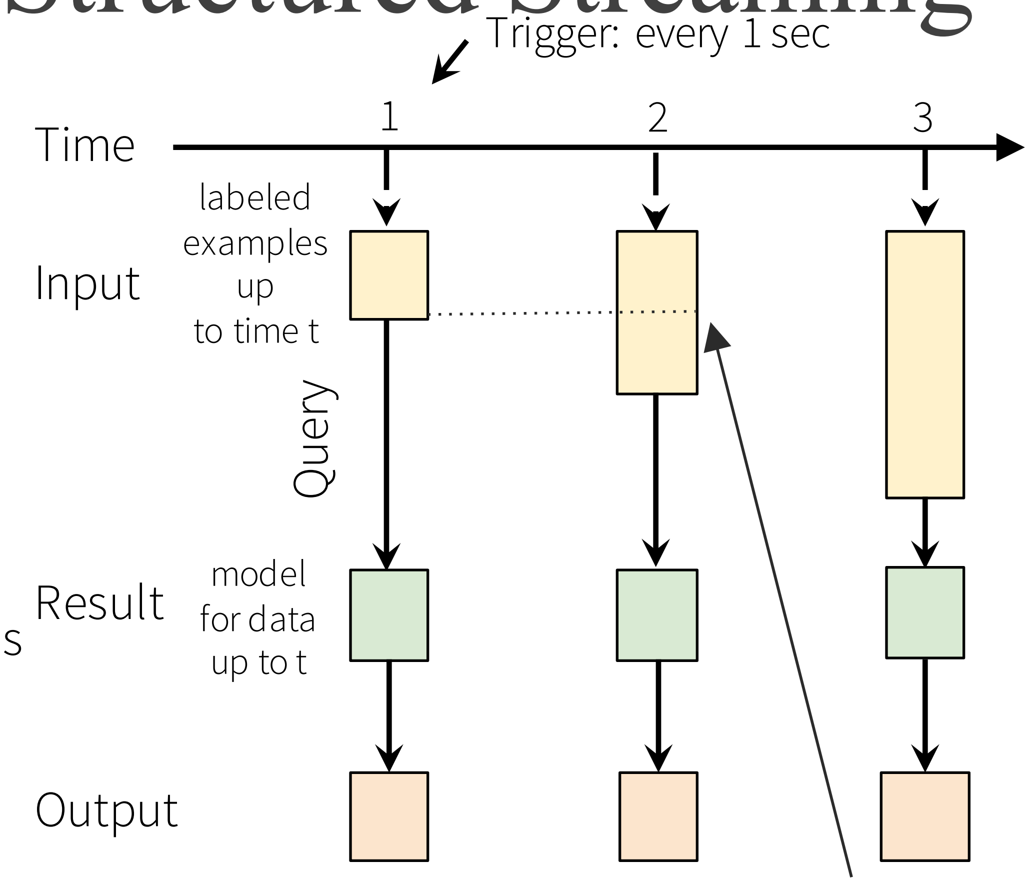


# Streaming ML on Structured Streaming

**Result:** table of model parameters updated every trigger interval

**Complete mode:** table has one row, constantly being updated

**Append mode (in the works):** table has timestamp-keyed model, one row per trigger




intermediate models would have the same state at this point of computation for the (abstract) queries #1 and #2

databricks

# Why is this hard?

- Need to update model, i.e
  - Update(previousModel, newDataPoint) = newModel
- Typical aggregation is associative, commutative
  - e.g. sum( P1: sum(sum(0, data[0]), data[1]), P2: sum(sum(0, data[2]), data[3]))
- General model update violates associativity + commutativity!

# Solution: Make Assumptions

- Result may be partition-dependent, but we don't care as long as we get some valid result.

```
average-models(
  P1: update(update(previous  model, data[0]),  data[1]),
  P2: update(update(previous  model, data[2]),  data[3]))
```

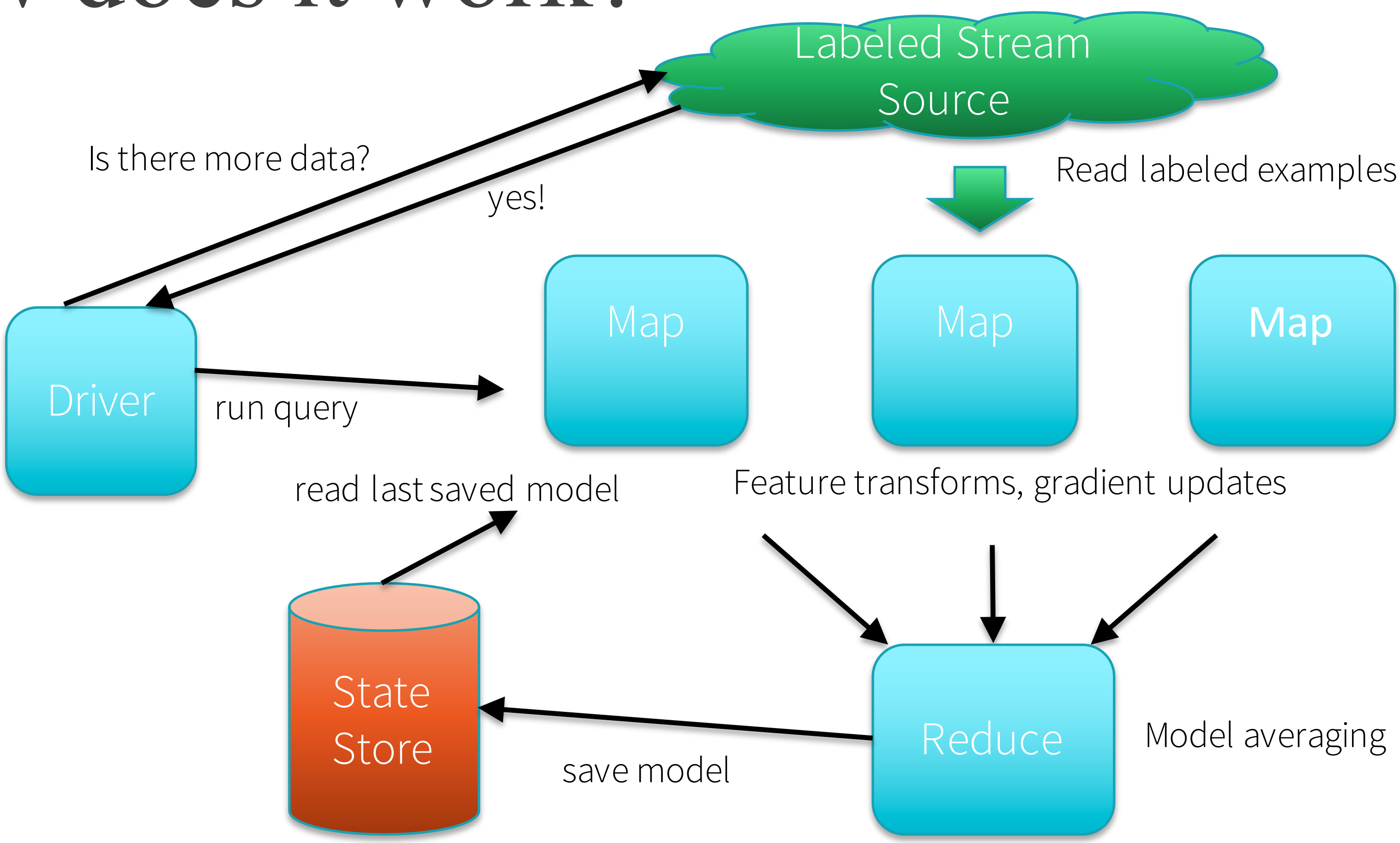
- Only partition-dependent if update and average don't commute - can still be deterministic otherwise!

# Stateful Aggregator

- Within each partition
  - Initialize with previous state (instead of zero in regular aggregator)
  - For each item, update state
- Perform reduce step
- Output final state

*Very general abstraction: works for sketches, online statistics (quantiles), online clustering …*

# How does it work?



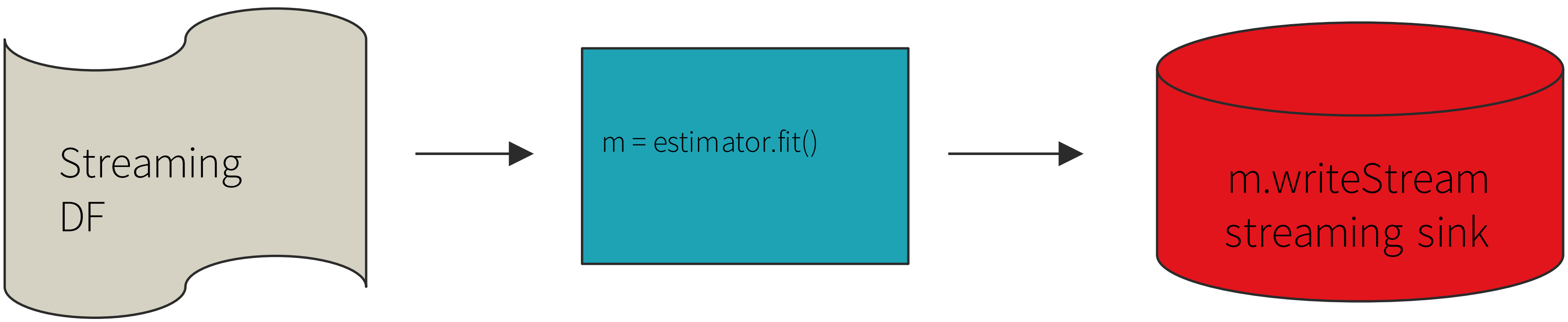

databricks

# APIs

Spark Summit Brussels
27 October 2016

# ML Estimator on Streams

- Interoperable with ML pipelines



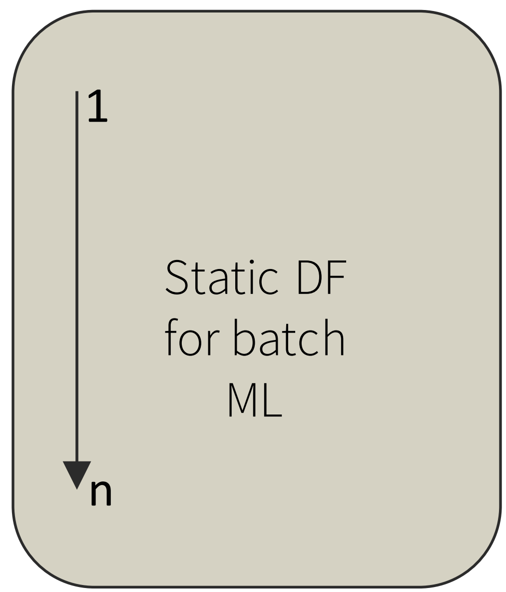

Input: stream of labelled data
Output: stream of models, updated over time.

databricks

# Batch Interoperability

- Seamless application on batch datasets

model = estimator.fit(batchDF)



# Feature Creation

- Handle new features as they appear (ex., IPs in fraud detection)
  - Provide transformers, such as the HashingEncoder, that apply the hashing trick.
  - Encode arbitrary (possibly categorical data) without knowing cardinality ahead of time by using a high-dimensional sparse mapping.

# API Goals

- Provide modern, regret-minimization-based online algorithms.
  - Online Logistic Regression
  - Adagrad
  - Online gradient descent
  - L2 regularization
- Input streams of any kind accepted.
- Streaming aware feature engineering

# What’s next?

Spark Summit Brussels
27 October 2016

databricks™

# What’s next?

- More bells and whistles
  - Adaptive normalization
  - L1 regularization
- More algorithms
  - Online quantile estimation?
  - More general Sketches?
  - Online clustering?
- Scale testing and benchmarking

# Demo

Spark Summit Brussels
27 October 2016

databricks™