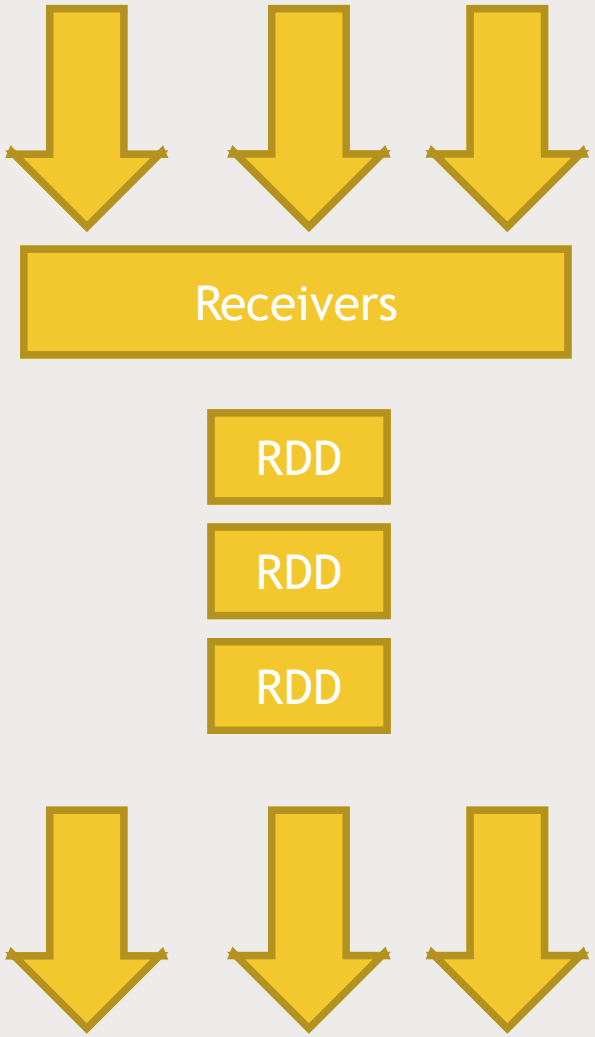
SPARK STREAMING

Processing continuous streams of data in near-real-time

Why Spark Streaming?

- “Big data” never stops!
- Analyze data streams in real time, instead of in huge batch jobs daily
- Analyzing streams of web log data to react to user behavior
- Analyze streams of real-time sensor data for “Internet of Things” stuff

Spark Streaming: High Level



Data Streams

Batches of data
For a given time increment

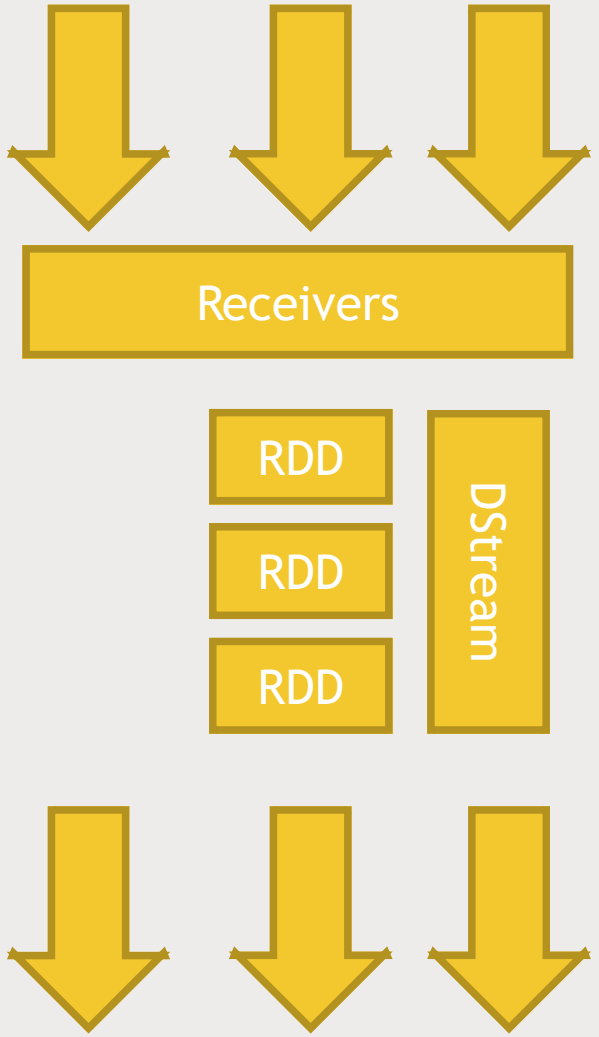
Transform & output to other systems

This work can be distributed

- Processing of RDD’s can happen in parallel on different worker nodes

DStreams (Discretized Streams)

- Generates the RDD's for each time step, and can produce output at each time step.
- Can be transformed and acted on in much the same way as RDD's
- Or you can access their underlying RDD's if you need them.



Common stateless transformations on DStreams

- Map
- Flatmap
- Fliter
- reduceByKey

Stateful data

- You can also maintain a long-lived state on a Dstream
- For example – running totals, broken down by keys
- Another example: aggregating session data in web activity

WINDOWING

Windowed Transformations

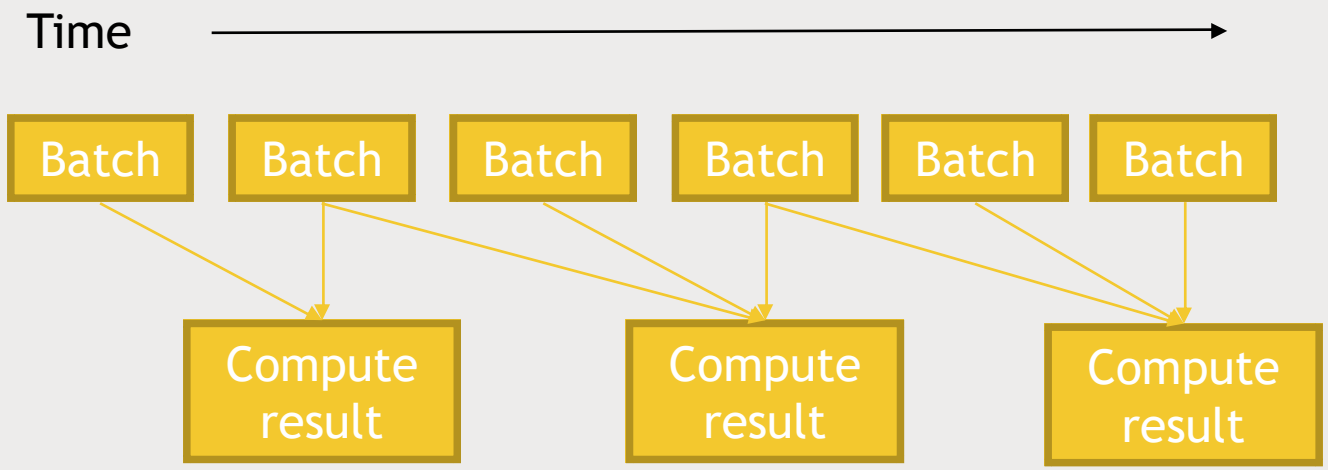
- Allow you to compute results across a longer time period than your batch interval
- Example: top-sellers from the past hour
  - *You might process data every one second (the batch interval)*
  - *But maintain a window of one hour*
- The window “slides” as time goes on, to represent batches within the window interval

Batch interval vs. slide interval vs. window interval

- The batch interval is how often data is captured into a Dstream
- The slide interval is how often a windowed transformation is computed
- The window interval is how far back in time the windowed transformation goes

Example

- Each batch contains one second of data (the batch interval)
- We set up a window interval of 3 seconds and a slide interval of 2 seconds



Windowed transformations: code

- The batch interval is set up with your SparkContext:

  ssc = StreamingContext(sc, 1)

- You can use reduceByWindow() or reduceByKeyAndWindow() to aggregate data across a longer period of time!

  hashtagCounts = hashtagKeyValues.reduceByKeyAndWindow(lambda x, y: x + y, lambda x, y : x - y, 300, 1)

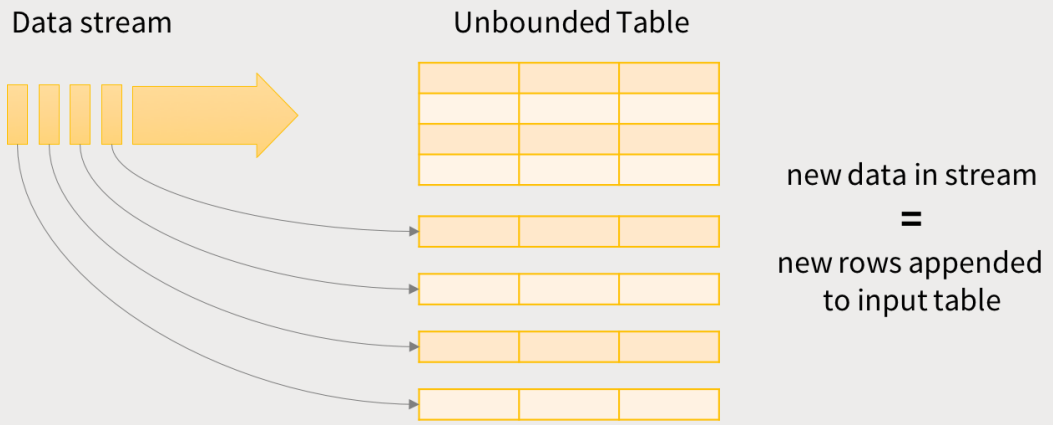
STRUCTURED STREAMING

What is structured streaming?

- A new, higher-level API for streaming structured data
  - *Available in Spark 2.0 and 2.1 as an experimental release*
  - *But it's the future.*
- Uses DataSets
  - *Like a DataFrame, but with more explicit type information*
  - *A DataFrame is really a DataSet[Row]*

Imagine a DataFrame that never ends

- New data just keeps getting appended to it
- Your continuous application keeps querying updated data as it comes in



Data stream as an unbounded Input Table

Advantages of Structured Streaming

- Streaming code looks a lot like the equivalent non-streaming code
- Structured data allows Spark to represent data more efficiently
- SQL-style queries allow for query optimization opportunities – and even better performance.
- Interoperability with other Spark components based on DataSets
  - *MLLib is also moving toward DataSets as its primary API.*
- DataSets in general is the direction Spark is moving

Once you have a SparkSession, you can stream data, query it, and write out the results.

2 lines of code to stream in structured JSON log data, count up “action” values for each hour, and write the results to a database.

```
val inputDF = spark.readStream.json("s3://logs")
inputDF.groupBy($"action", window($"time", "1 hour")).count()
       .writeStream.format("jdbc").start("jdbc:mysql//...")
```

LET’S PLAY

Spark Streaming with Flume

- We'll set up Flume to use a spooldir source as before
- But use an Avro sink to connect it to our Spark Streaming job!
  - *Use a window to aggregate how often each unique URL appears from our access log.*
- Using Avro in this manner is a "push" mechanism to Spark Streaming
  - *You can also "pull" data by using a custom sink for Spark Streaming*