TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Macroarea di Ingegneria
Dipartimento di Ingegneria Civile e Ingegneria Informatica

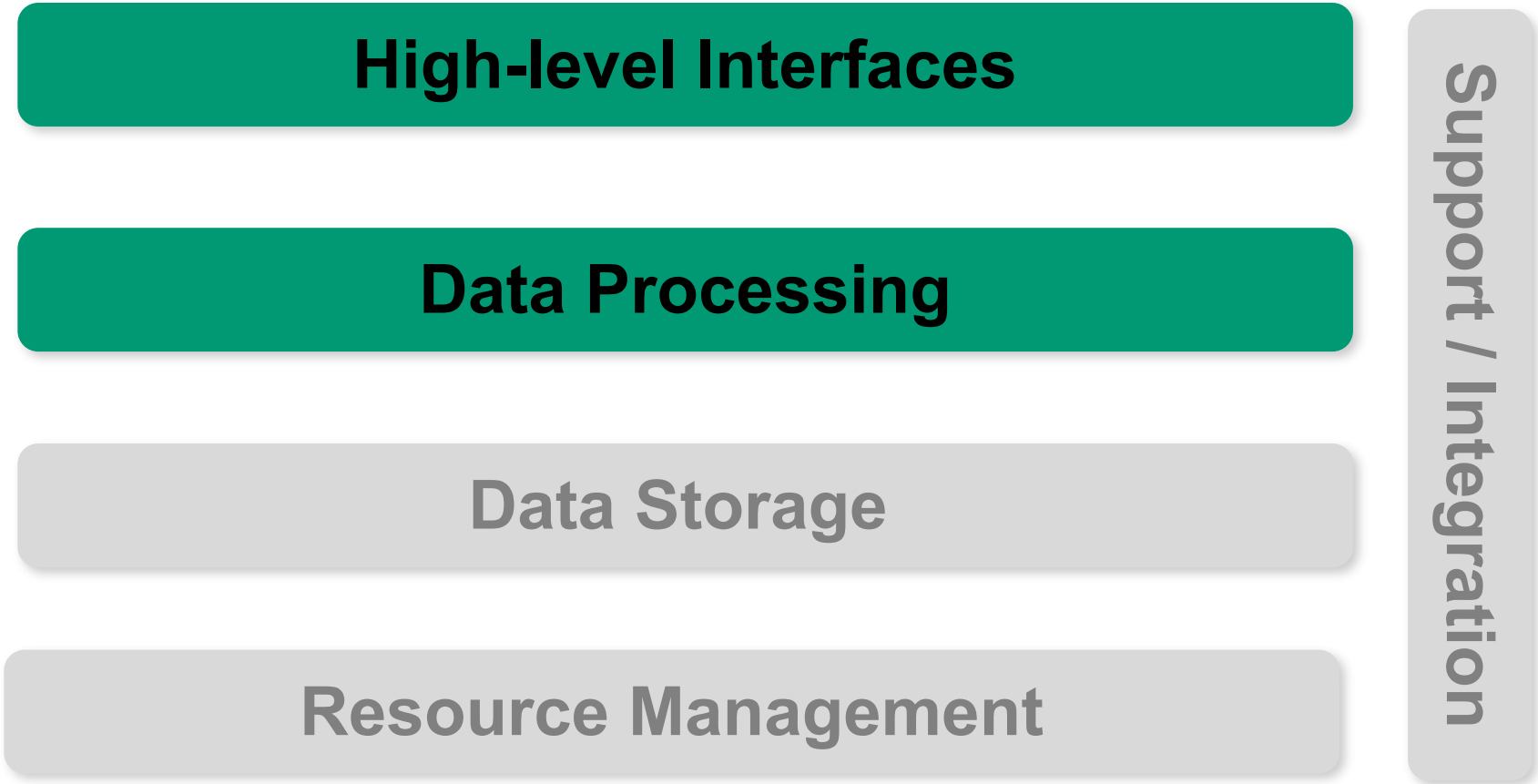
Spark Streaming: Hands-on Session

A.A. 2025/26

Matteo Nardelli

Laurea Magistrale in
Ingegneria Informatica - II anno

The reference Big Data stack



Apache Spark

Book

- Learning Spark, 2nd Edition
  - by Jules S. Damji, Brooke Wenig, Tathagata Das, Denny Lee
  - Released July 2020
  - Publisher(s): O'Reilly Media, Inc.

Papers

- Zaharia, Das, Li, Hunter, Shenker, Stoica. Discretized Streams: A Fault-Tolerant Model for Scalable Stream Processing. Berkeley EECS
- Zaharia, Chowdhury, Franklin, Shenker, Stoica. Spark: Cluster Computing with Working Sets. HotCloud 10

Online References

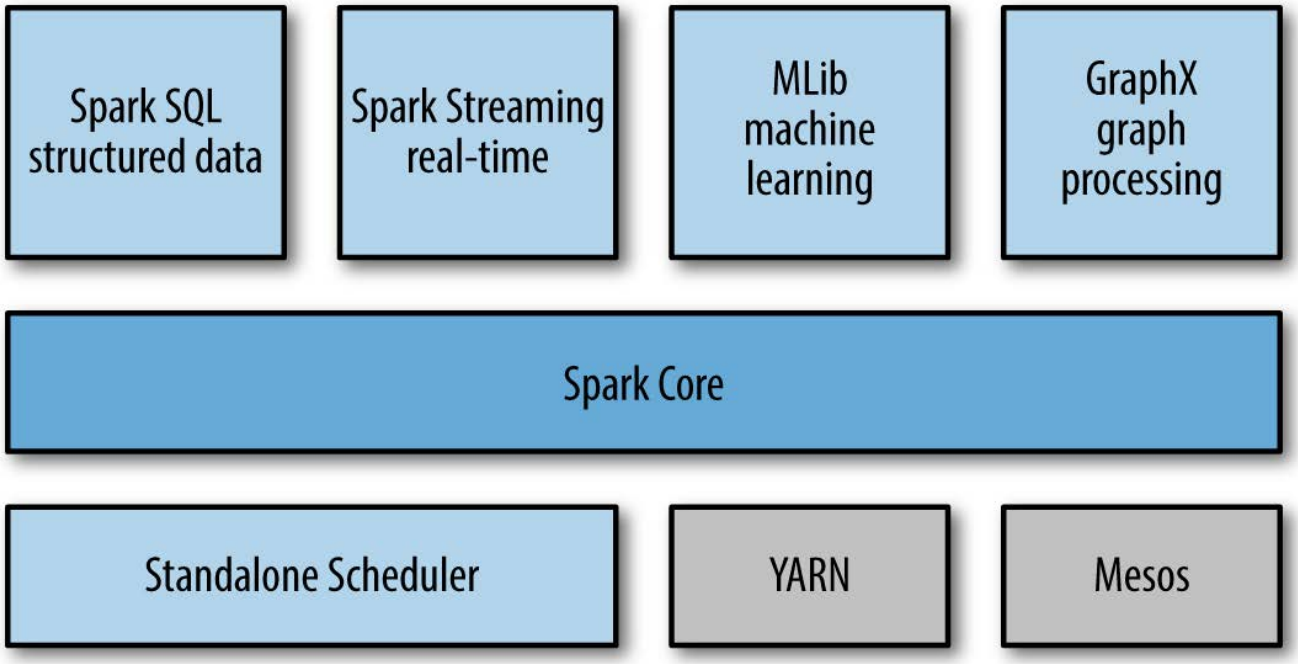
- Apache Spark: http://spark.apache.org/
- Spark Documentation: http://spark.apache.org/docs/latest/api/python/index.html

Apache Spark

- Fast and general-purpose engine for large-scale data processing
  - Not a modified version of Hadoop
  - The leading candidate for “successor to MapReduce”
- Spark can efficiently support **more types of computations**
  - interactive queries, stream processing
- Can read/write to any Hadoop-supported system (e.g., HDFS)
- **Speed:**
  - in-memory data storage for very fast iterative queries
  - the system is also more efficient than MapReduce for complex applications running on disk
  - up to 40x faster than Hadoop

Apache Spark

- Spark Core: basic functionality of Spark (task scheduling, memory management, fault recovery, storage systems interaction).
- Spark SQL: package for working with structured data queried via SQL as well as HiveQL
- **Spark Streaming**: a component that enables processing of live streams of data (e.g., logfiles, status updates messages)



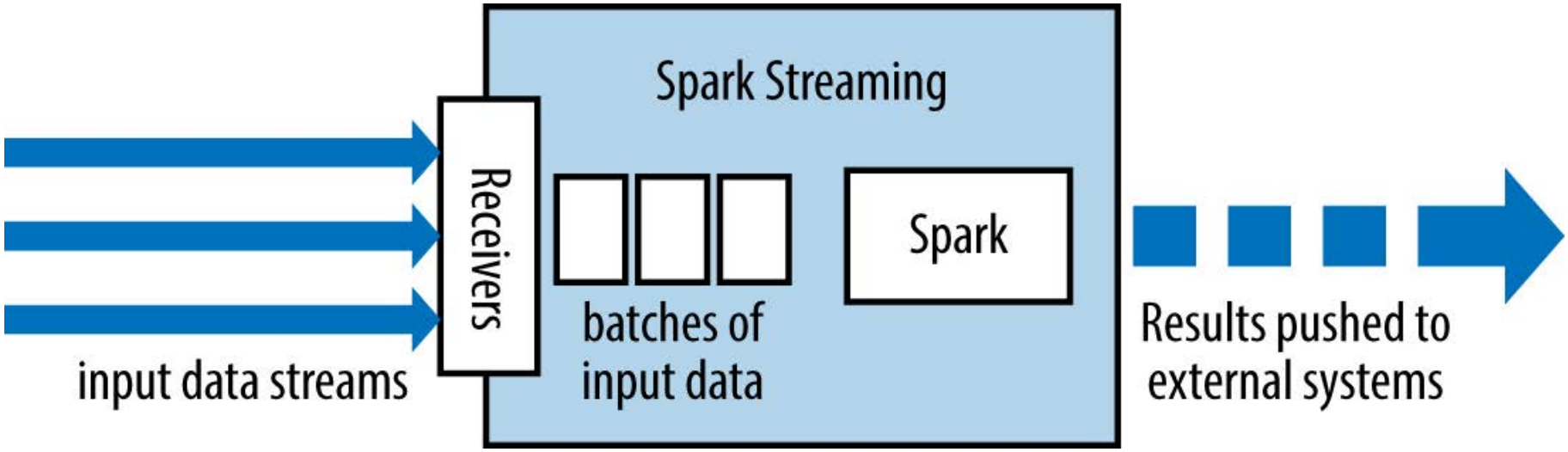
Spark Streaming

- Data can be ingested from many sources: Kafka, Twitter, HDFS, TCP sockets
- Results can be pushed out to file-systems, databases, live dashboards, but not only

Spark Streaming: Abstractions

micro-batch architecture

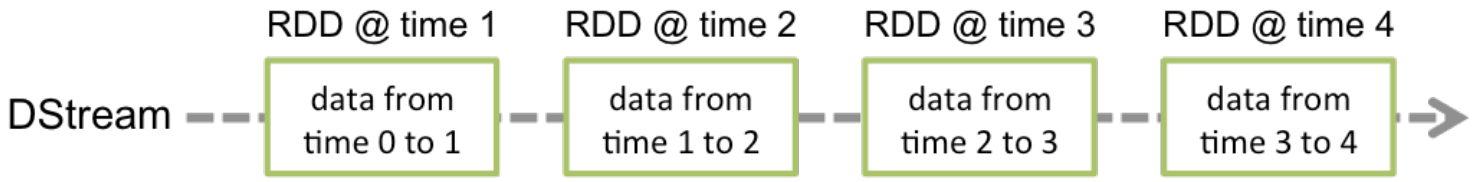
- the stream is treated as a series of batches of data
- new batches are created at regular time intervals
- the size of the time intervals is called the **batch interval**
- the batch interval is typically between 500 ms and several seconds



Spark Streaming: DStream

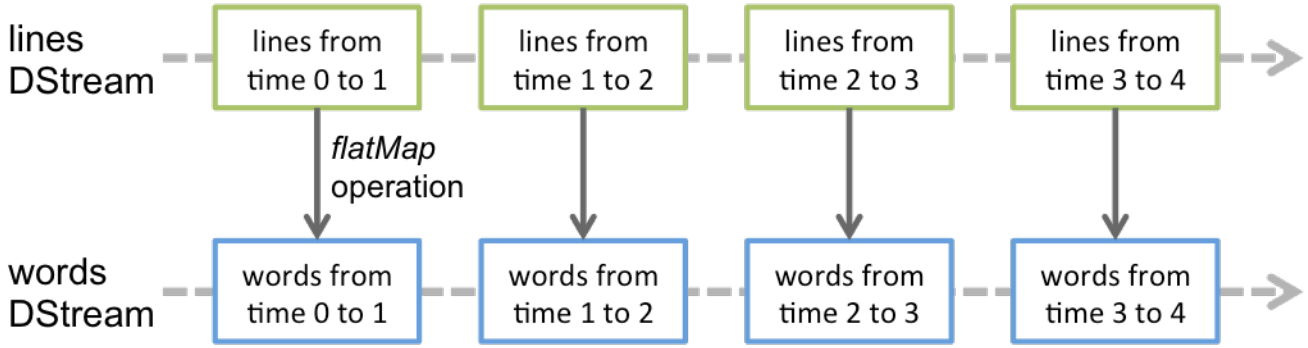
Discretized Stream (DStream)

- basic abstraction provided by Spark Streaming
- represents a continuous stream of data
  - either input stream or generated by transforming the input stream
- internally, a DStream is represented by a continuous series of RDDs. Each RDD in a DStream contains data from a certain interval.



Spark Streaming: DStream

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



- RDD transformations are computed by the Spark engine
- the DStream operations hide most of these details

Spark Streaming: Streaming Context

To execute a SparkStreaming application, we need to define the StreamingContext

- specializes SparkContext for streaming applications
- in Java can be defined as follows

```
JavaStreamingContext ssc =
        new JavaStreamingContext(sparkConf, batchInterval);
```

where:

- **master** is a Spark, Mesos or YARN cluster URL; to run your code in local mode, use "local[K]" where K>=2 represents the parallelism
- **appname** is the name of your application
- **batchinterval** time interval (in seconds) of each batch

Spark Streaming: DStream

Basic data sources

- **File Streams:** For reading data from files on any file system compatible with the HDFS API (that is, HDFS, S3, NFS, etc.), a DStream can be created as:

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

- **Streams based on Custom Receivers:** DStreams can be created with data streams received through custom receivers, extending the Receiver<T> class
- **Queue of RDDs as a Stream:** For testing a Spark Streaming application with test data, one can also create a DStream based on a queue of RDDs, using

```
... = streamingContext.queueStream(queueOfRDDs)
```

Spark Streaming: DStream

Once built, they offer two types of operations:

- **Transformations** which yield a **new** DStream from a previous one. For example, one common transformation is filtering data.
  - **stateless transformations**: the processing of each batch does not depend on the data of its previous batches.
    Examples are: map(), filter(), and reduceByKey()
  - **stateful transformations**: use data from previous batches to compute the results of the current batch.
    They include sliding windows, and tracking state across time
- **Output operations** which write data to an external system.
  Each streaming application has to define an output operation.

Note that a streaming context can be started only once, and must be started after we set up all the DStreams and output operations.

Spark Streaming: DStream

Most of the transformations have the same syntax as the one applied to RDDs

Transformation	Meaning
map(func)	Return a new DStream by passing each element of the source DStream through a function func.
flatMap(func)	Similar to map, but each input item can be mapped to 0 or more output items.
filter(func)	Return a new DStream by selecting only the records of the source DStream on which func returns true.
union(otherStream)	Return a new DStream that contains the union of the elements in the source DStream and otherDStream.
join(otherStream)	When called on two DStreams of (K, V) and (K, W) pairs, return a new DStream of (K, (V, W)) pairs with all pairs of elements for each key.

Spark Streaming: DStream

Transformation	Meaning
reduce(func)	Return a new DStream of single-element RDDs by aggregating the elements in each RDD of the source DStream using a function func (which takes two arguments and returns one). The function should be associative so that it can be computed in parallel.
reduceByKey(func)	When called on a DStream of (K, V) pairs, return a new DStream of (K, V) pairs where the values for each key are aggregated using the given reduce function.
count()	Return a new DStream of single-element RDDs by counting the number of elements in each RDD of the source DStream.
collect()	Return all elements from the RDD.
transform(func)	Return a new DStream by applying a RDD-to-RDD function to every RDD of the source DStream. This can be used to do arbitrary RDD operations on the DStream.
updateStateByKey(func)	Return a new "state" DStream where the state for each key is updated by applying the given function on the previous state of the key and the new values for the key. This can be used to maintain arbitrary state data for each key.

Example: Network Word Count

```
...
SparkConf sparkConf = new SparkConf()
  .setMaster("local[2]").setAppName("NetworkWordCount");
            JavaStreamingContext ssc = ...

            JavaReceiverInputDStream<String> lines =
                                                ssc.socketTextStream( ... );

            JavaDStream<String> words =  lines.flatMap(...);

            JavaPairDStream<String, Integer> wordCounts =
words.mapToPair(s -> new Tuple2<>(s, 1))
                                    .reduceByKey((i1, i2) -> i1 + i2);

            wordCounts.print();
...
```

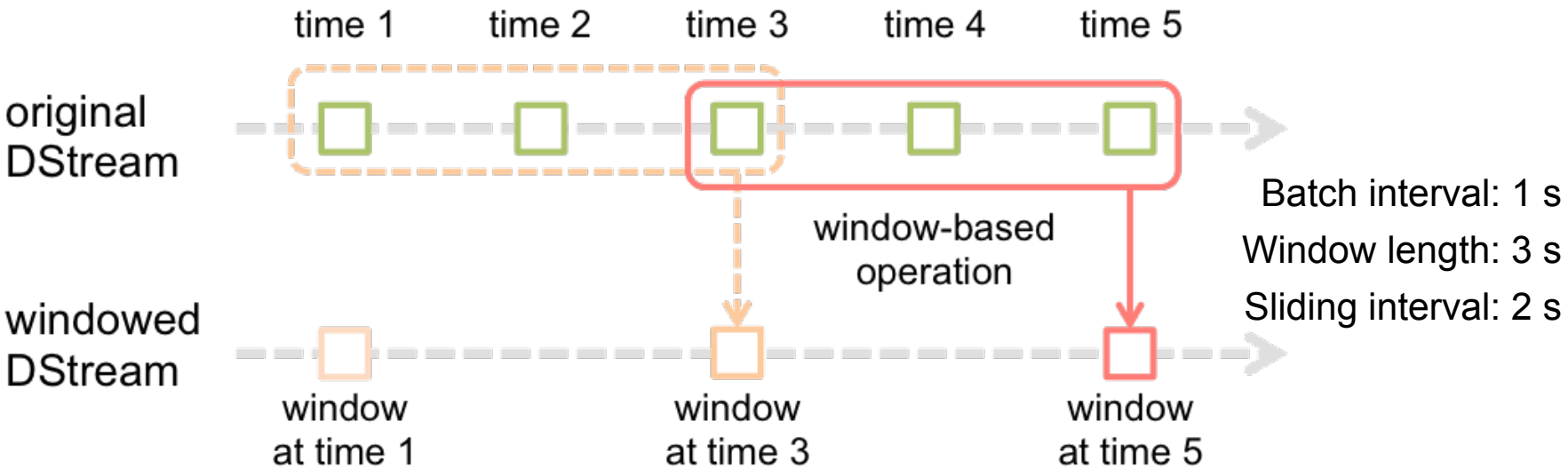
Excerpt of NetworkWordCount.java

Spark Streaming: Window

Windowed computations allow you to apply transformations over a sliding window of data. Any window operation needs to specify two parameters:

- *window length*
  The duration of the window in secs
- *sliding interval*
  The interval at which the window operation is performed in secs

These parameters must be multiples of the batch interval



Spark Streaming: Window

window(windowLength, slideInterval**)**

Return a new DStream which is computed based on windowed batches.

```
...
JavaStreamingContext ssc = ...

JavaReceiverInputDStream<String> lines = ...

JavaDStream<String> linesInWindow =
                    lines.window(WINDOW_SIZE, SLIDING_INTERVAL);

JavaPairDStream<String, Integer> wordCounts =
            linesInWindow.flatMap(SPLIT_LINE)
                    .mapToPair(s -> new Tuple2<>(s, 1))
                    .reduceByKey((i1, i2) -> i1 + i2);
...
```

Excerpt of WindowBasedWordCount.java

Spark Streaming: Window

reduceByWindow(func, InvFunc, windowLength, slideInterval**)**

Return a new single-element stream, created by aggregating elements in the stream over a sliding interval using *func* (which should be associative).

The reduce value of each window is calculated incrementally.

- *func* reduces new data that enters the sliding window
- *invFunc* “inverse reduces” the old data that leaves the window.

reduceByKeyAndWindow(func, InvFunc, windowLength, slideInterval**)**

When called on a DStream of (K, V) pairs, returns a new DStream of (K, V) pairs where the values for each key are aggregated using the given reduce function *func* over batches in a sliding window.

For performing these transformations, we need to define a checkpoint directory

A Window-based WordCount

We now perform only the reduce operation within a sliding window. We change our code as follows.

```
...
JavaPairDStream<String, Integer> wordCountPairs =
ssc.socketTextStream(...)
            .flatMap(x -> Arrays.asList(SPACE.split(x)).iterator())
.mapToPair(s -> new Tuple2<>(s, 1));

JavaPairDStream<String, Integer> wordCounts =
        wordCountPairs.reduceByKeyAndWindow(
                (i1, i2) -> i1 + i2,
                WINDOW_SIZE,
                SLIDING_INTERVAL);

        wordCounts.print();
        wordCounts.foreachRDD(new SaveAsLocalFile());
...
```

A (More Efficient) Window-based WordCount

A more efficient version: the reduce value of each window is calculated incrementally

- a reduce function handles new data that enters the sliding window;
- an “inverse reduce” function handles old data that leaves the window.

Note that checkpointing must be enabled for using this operation.

```
...
ssc.checkpoint(LOCAL_CHECKPOINT_DIR);
...
JavaPairDStream<String, Integer> wordCounts =
            wordCountPairs.reduceByKeyAndWindow(
            (i1, i2) -> i1 + i2,
            (i1, i2) -> i1 - i2,
            WINDOW_SIZE, SLIDING_INTERVAL);
      ...
```

Excerpt of WindowBasedWordCount2.java

Spark Streaming: Output Operations

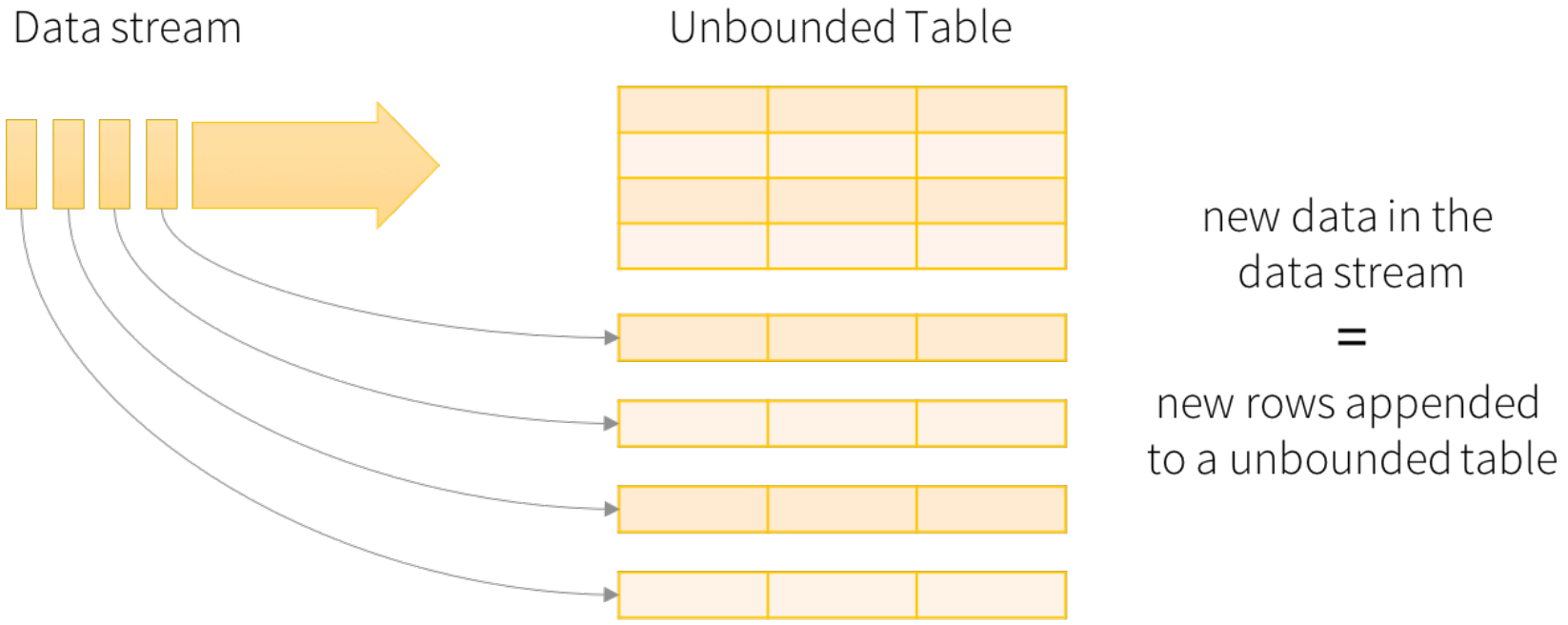
Output operations allow DStream's data to be pushed out to external systems like a database or a file systems

Output Operation	Meaning
print()	Prints the first ten elements of every batch of data in a DStream on the driver node running the application.
saveAsTextFiles(prefix, [suffix])	Save this DStream's contents as text files. The file name at each batch interval is generated based on prefix.
saveAsHadoopFiles(prefix, [suffix])	Save this DStream's contents as Hadoop files.
saveAsObjectFiles(prefix, [suffix])	Save this DStream's contents as SequenceFiles of serialized Java objects.
foreachRDD(func)	Generic output operator that applies a function, func, to each RDD generated from the stream.

Structured Streaming (since Spark 3.4.0)

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

- You can use the Dataset/DataFrame API to express streaming aggregations, event-time windows, stream-to-batch joins, etc.



Data stream as an unbounded table

Structured Streaming (since Spark 3.4.0)

Example: word count

```java
// Create DataFrame representing the stream of input lines from connection to localhost:9999
Dataset<Row> lines = spark
  .readStream()
  .format("socket")
  .option("host", "localhost")
  .option("port", 9999)
  .load();

// Split the lines into words
Dataset<String> words = lines
  .as(Encoders.STRING())
  .flatMap((FlatMapFunction<String, String>) x -> Arrays.asList(x.split(" ")).iterator(), Encoders.STRING());

// Generate running word count
Dataset<Row> wordCounts = words.groupBy("value").count();
```

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

Structured Streaming (since Spark 3.4.0)

Spark Streaming has been marked as “deprecated”

- *“Spark Streaming is the previous generation of Spark’s streaming engine. There are no longer updates to Spark Streaming and it’s a legacy project. There is a newer and easier to use streaming engine in Spark called Structured Streaming.”*

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

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