SPARK STREAMING

AMEL BOUZEGHOUB

Amel.Bouzeghoub@telecom-sudparis.eu

TELECOM SudParis

IP PARIS

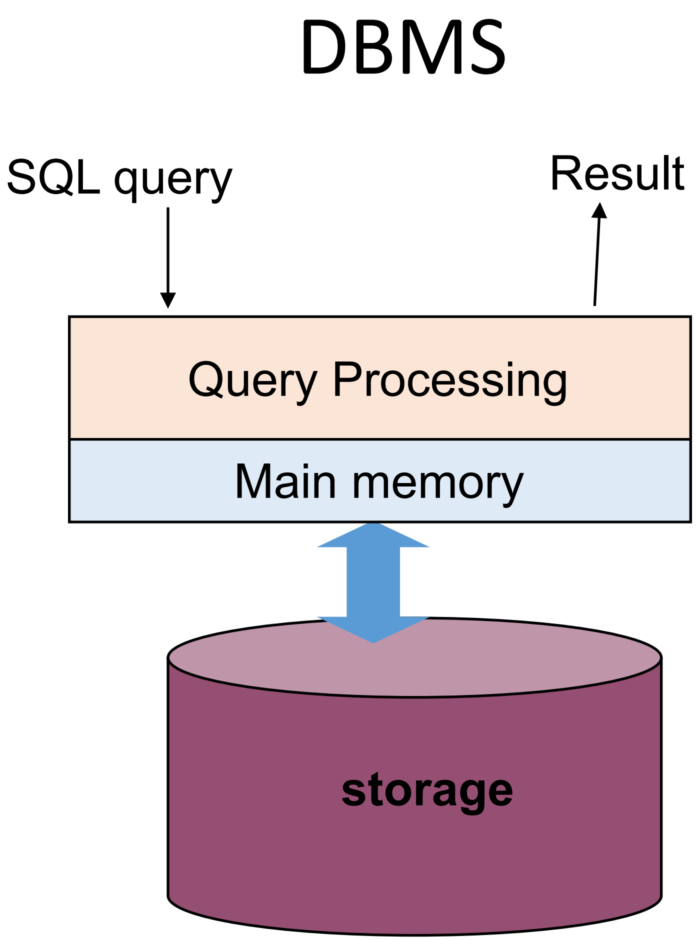
Institut Mines-Télécom

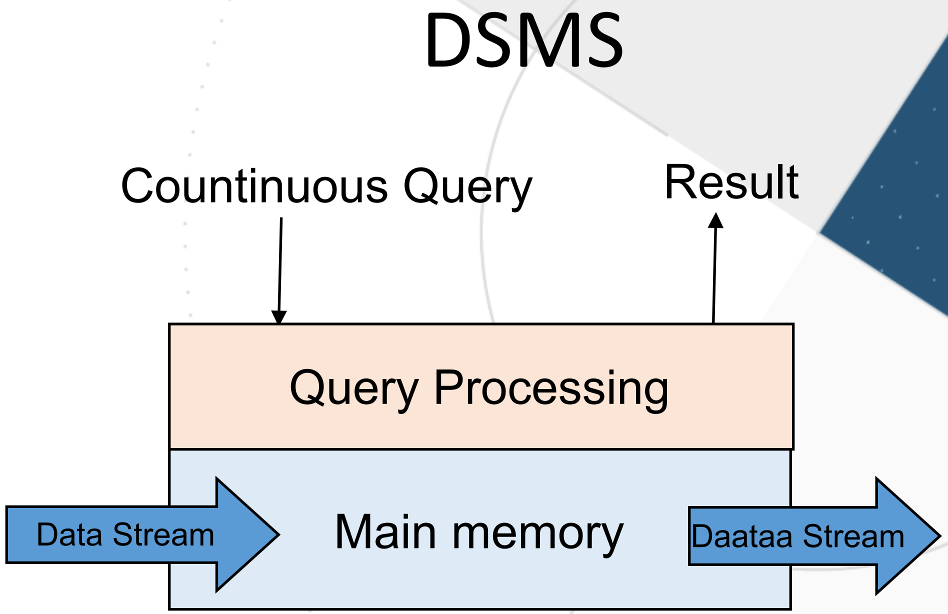
CSC5003

MOTIVATION

1. Many important applications need to process large data streams and provide near real-time results
   - Social network trends
   - Website statistics
   - Intrusion detection systems
   - etc.
2. Requires large clusters to cope with the workload
3. Requires latency times of a few seconds

SOME DEFINITIONS: DBMS VS DSMS





SOME DEFINITIONS: STREAMS

Stream: succession of data of the same type, incoming at constant or variable intervals

➔ Massive data: **high throughput** of the stream, not **all** the data can be processed regularly

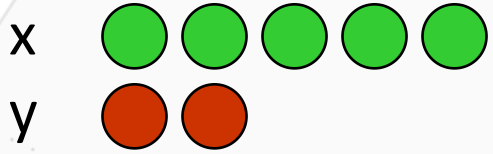
1. The processing of data slices must be performed in real time
2. RAM is limited and mass storage is too slow
3. The modeling must be incremental

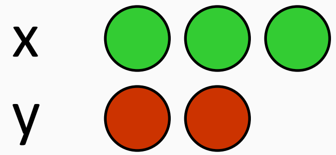
➔ **Solutions** : approximate answers to queries

- Sliding windows
- Batch processing
- Sampling and load shedding
- Synopsis

SOME DEFINITIONS: DATA MODELING

1. **Observation $\neq$ Modeling**
   - Observation: data that we can read on the stream
   - Modeling: signal that we want to rebuild from these data
2. Time series model
   - Sequence of observations (ti, ei)
3. Cash register model (arrivals only)
   - Ei = (j, vi) vi always > 0
   - Multi-dimentionnel flow in one pass
   - Example: <x, 3>, <y, 2>, <x, 2> means
     - Arrival of 3 copies of item x,
     - 2 copies of y, then 2 copies of x.
4. Turnstile model (arrivals and departures)
   - vi is either > 0 or <0
   - The flow elements update the data of the series
   - Each element of the flow is an update
   - Example: <x, 3>, <y,2>, <x, -2> means
     final state of <x, 1>, <y, 2>.





SOME DEFINITIONS: BATCH VS STREAM PROCESSING

1. Data processing can be achieved in three different ways :
   - Batch : available data are processed at a specific time T.
   - Micro-Batch : available data is processed every n seconds.
   - Real Time : data is processed as soon as it becomes available.

BATCH	STREAM
- Process a full (large) dataset from scratch - Focus on throughput (time / size) - Takes a long time (minutes, hours ...) to obtain results - Complex analysis requiring multiple pass over data (e.g. machine learning) - Good for analyzing a static dataset (post-mortem)	- Process recent data (small window) to continuously update results - Focus on latency (time between data production and results update) - Near real-time - Incremental analysis, see data only once - Good to analyze live data (e.g. what is trending on Twitter?)

SOME DEFINITIONS: WINDOWS

Logical window vs. physical window
Fixed window vs. sliding window

Timestamps

- Used to order the instances
- Useful for the DSMS to define the size of the windows
- Useful for the user to know the arrival date of the data
- Explicit: given by the source
- Implicit: given by the DSMS

Sliding:

Jumping:

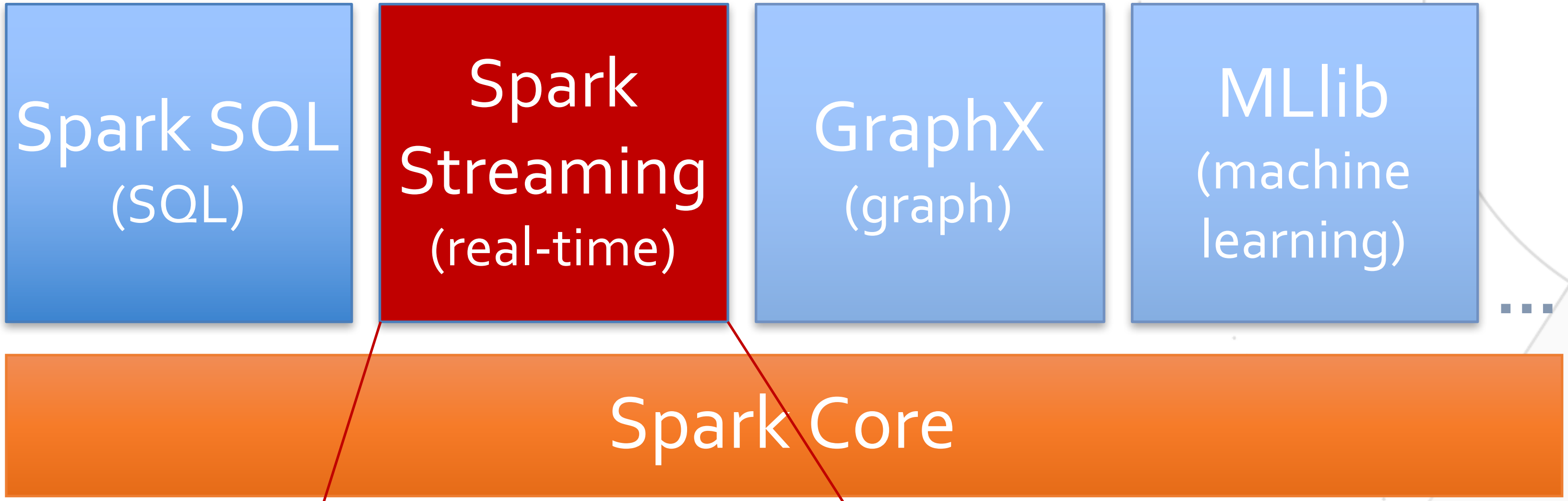
Overlapping

(adapted from Jarle Søberg)

SPARK STREAMING

Institut Mines-Télécom

SPARK COMPONENTS

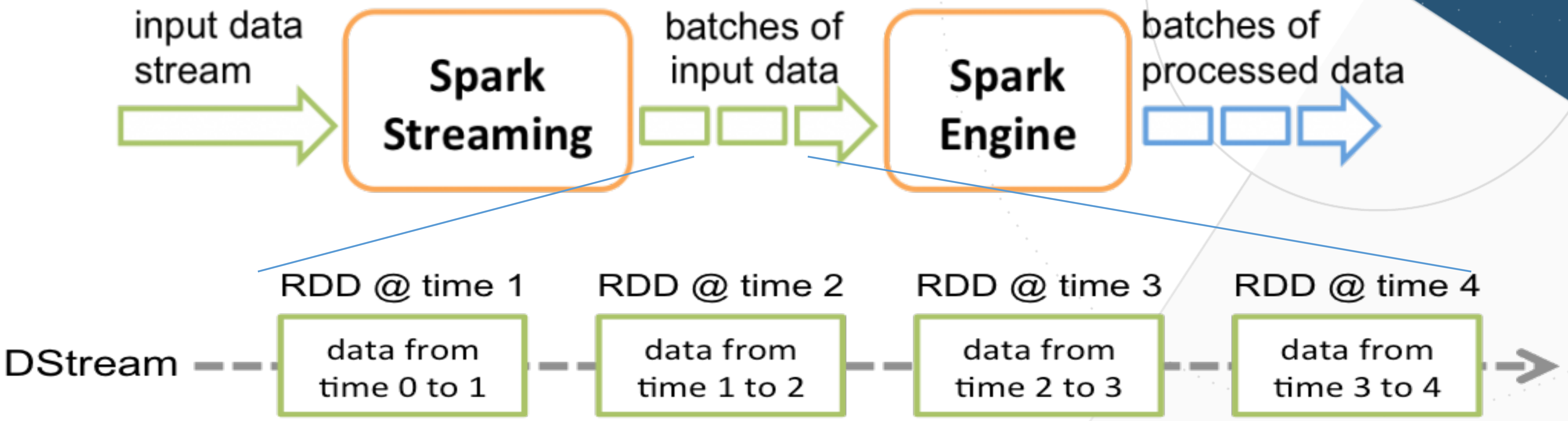


scalable, **high-throughput**, **fault-tolerant** stream processing of live data streams

SPARK STREAMING

1. Framework for large-scale stream processing
   - Can handle hundreds of nodes
   - Can reach latencies of the order of a second
   - Integrated into Spark batch processing
   - Provides a simple batch API for implementing complex algorithms
   - Can ingest live data streams coming from Kafka, Flume, ZeroMQ, etc.

GENERAL PRINCIPLE : DSTREAM



DSTREAMS

1. Spark Streaming provides a high-level abstraction called discretized stream (Dstream)
   - Represents a continuous stream of data.
   - DStreams can be created either from input data streams from sources such as Kafka, Flume, and Kinesis, or by applying high-level operations on other DStreams.
   - Internally, a DStream is represented as a sequence of RDDs.

2. DStreams API very similar to RDD API
   - Functional APIs in Scala, Java
   - Create input DStreams from different sources
   - Apply parallel operations

DSTREAMS TRANSFORMATIONS

Transformations

- map(func), flatMap(func), filter(func), count()
- repartition(numPartitions)
- union(otherStream)
- reduce(func),countByValue (), reduceByKey(func, [numTasks])
- join(otherStream, [numTasks])
- transform(func)
- updateStateByKey(func)

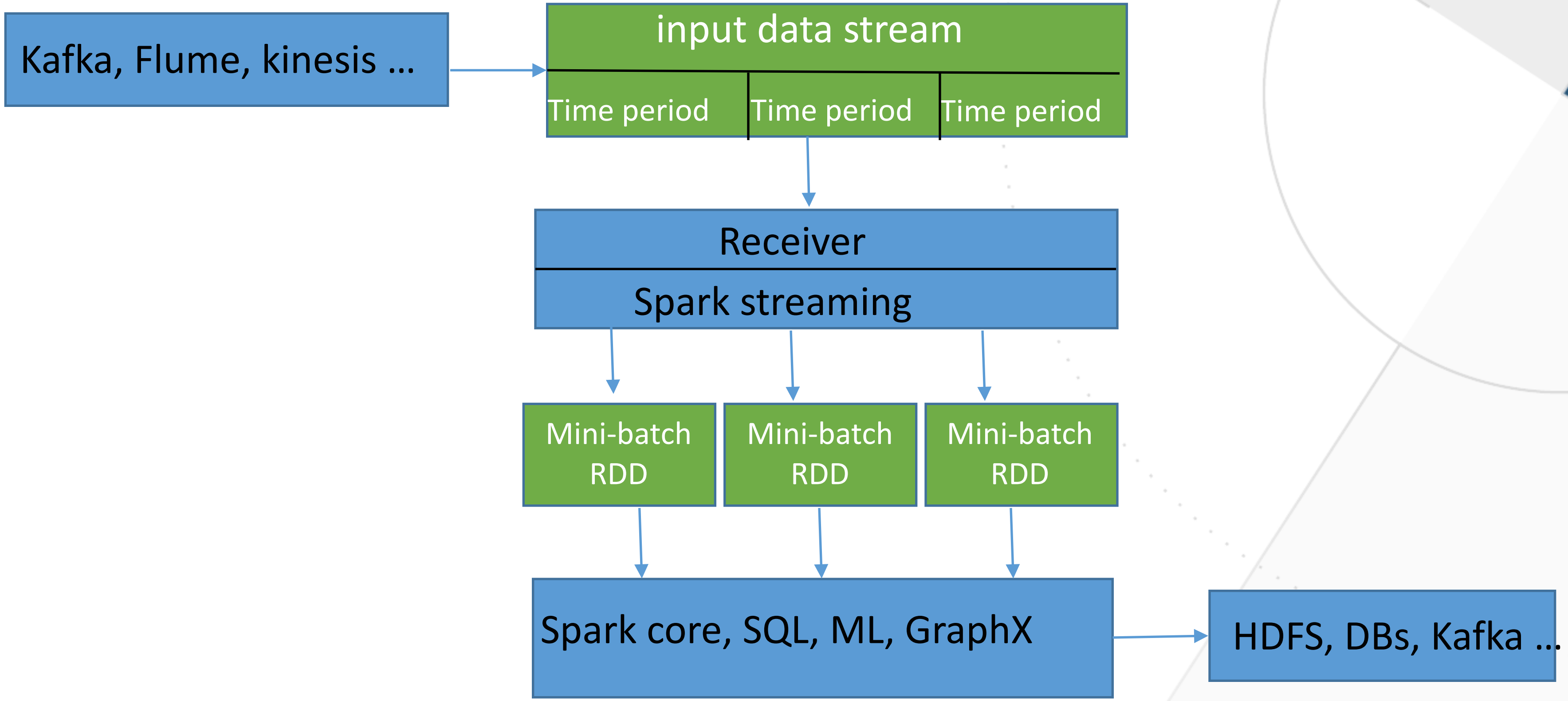
Windows operations

- window(windowLength, slideInterval)
- countByWindow(windowLength , slideInterval)
- reduceByWindow(func, windowLength, slideInterval)
- reduceByKeyAndWindow(func, windowLength, slideInterval, [numTasks])
- countByValueAndWindow(windowLength, slideInterval, [numTasks])

Output operations

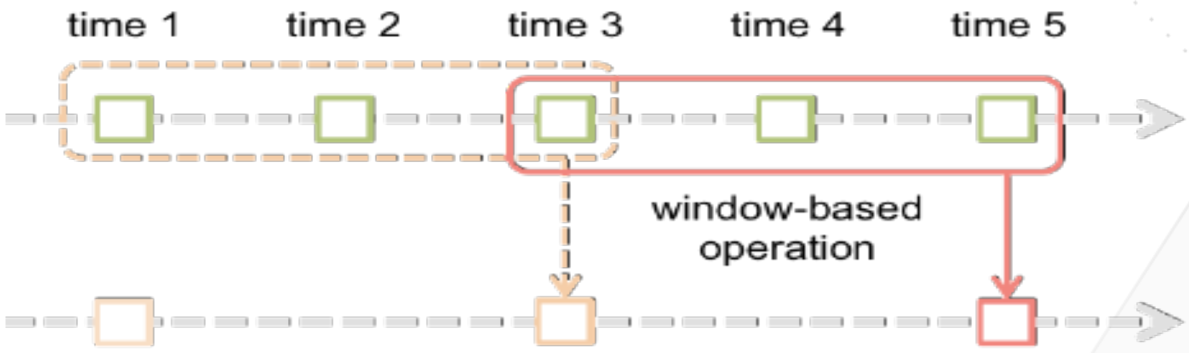
- print()
- foreachRDD(func)
- saveAsObjectFiles(prefix, [suffix])
- saveAsTextFiles(prefix, [suffix])
- saveAsHadoopFiles(prefix, [suffix])
-

ARCHITECTURE



SPARK STREAMING : WINDOW OPERATIONS

1. A window is defined by two numbers
   - How many slices are in the window at any given time (window length)
   - By how many slices the window moves (sliding step)
2. The RDDs corresponding to the slices that are in the window are grouped and processed; the operations applied on the RDDs are extended to the windows (e.g. reduceByKeyAndWindow)



ADVANTAGES

1. Spark Streaming offers failure recovery via the checkpoint method
   - In case of failure, Spark Streaming will start from the last checkpoint
   - The checkpoint must be done periodically
   - The frequency of the backup has a direct impact on the performance
   - On average the backup is done every 10 microbatches.
2. Operations on DStreams are *stateless*, from one batch to another, context is lost. Spark Streaming offers two methods that allow *stateful* processing :
   - reduceByWindow and reduceByKeyAndWindow

Let’s go to the lab!