TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Macroarea di Ingegneria
Dipartimento di Ingegneria Civile e Ingegneria Informatica

Introduction to Data Stream Processing

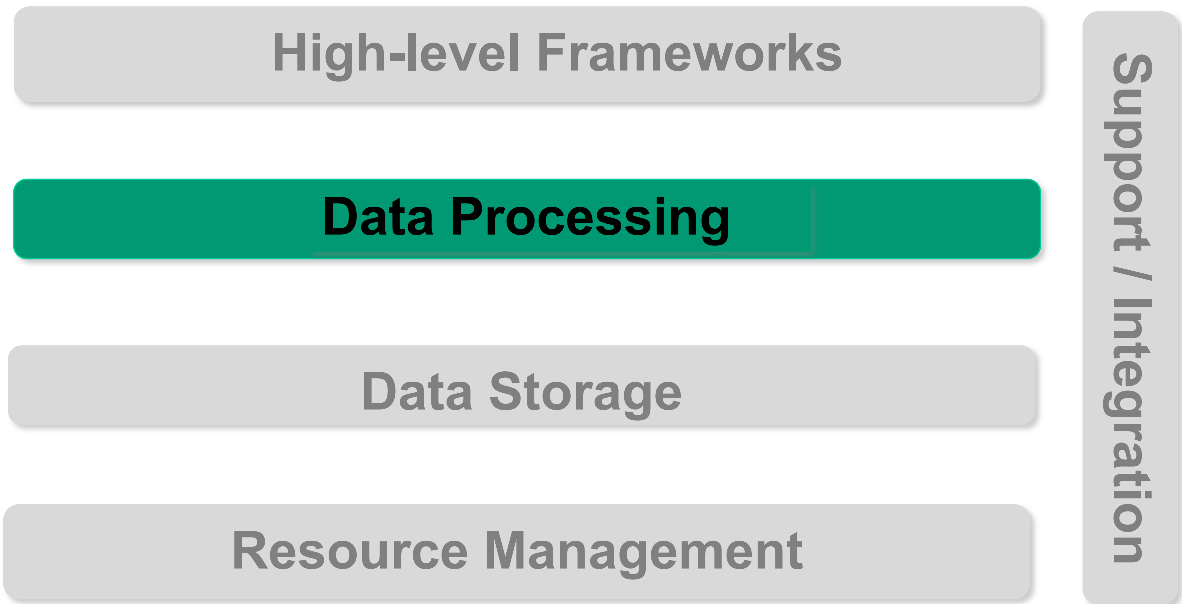
Corso di Sistemi e Architetture per Big Data

A.A. 2021/22

Valeria Cardellini

Laurea Magistrale in Ingegneria Informatica

The reference Big Data stack

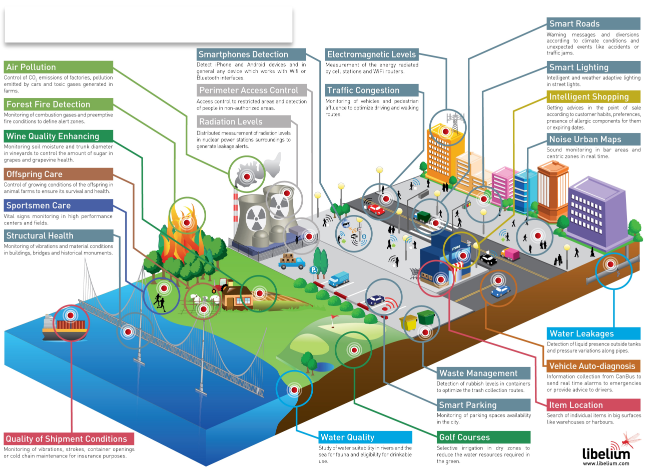


Why data stream processing?

- Applications such as:
  – Sentiment analysis on tweet streams @Twitter
  – User profiling @Yahoo!
  – Tracking of query trend evolution @Google
  – Fraud detection in financial transactions
  – Real-time advertising
  – Healthcare analytics involving IoT medical sensors
- Require:
  – Continuous **processing** of unbounded **data streams** generated by multiple and distributed sources
  – In (near) **real-time** fashion

Why data stream processing?

- In the early years data stream processing (**DSP**) was considered a solution for very specific problems (e.g., financial tickers)
- Now we have more general settings
  – E.g., Internet of Things



Why data stream processing?

- Decrease overall latency to obtain results
  - No data persistence on stable storage
    Recall "Latency numbers every programmer should know"!
  - No periodic batch analysis

- Simplify Big data infrastructure

Data stream: example

- "A data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamp) sequence of items. It is impossible to control the order in which items arrive, nor is it feasible to locally store a stream in its entirety. Queries over streams run continuously over a period of time and incrementally return new results as new data arrive."

Source: Golab and Özs, Issues in data stream management, ACM *SIGMOD Rec.* 32, 2, 2003. http://bit.ly/2rp3sJn

Data stream: example

- Data stream related to maritime traffic

```
0x3b62baab6210a8e69d3e7f9df53d000c83d00fd0,2,
15.247220,37.287770,163,511,01-06-15 0:00,AUGUSTA,
0x0fe9acdb3675a8a2942fafbd4af61bc37e44c0ec,146,
23.694910,37.313620,13,15,01-06-15 0:00,SALERNO,88
0xb35dc6acdc29f2241296c44384fa2b0f7044d257,20,
15.669920,38.387740,339,339,01-06-15 0:00,MESSINA,66
…
```

Tuple fields:

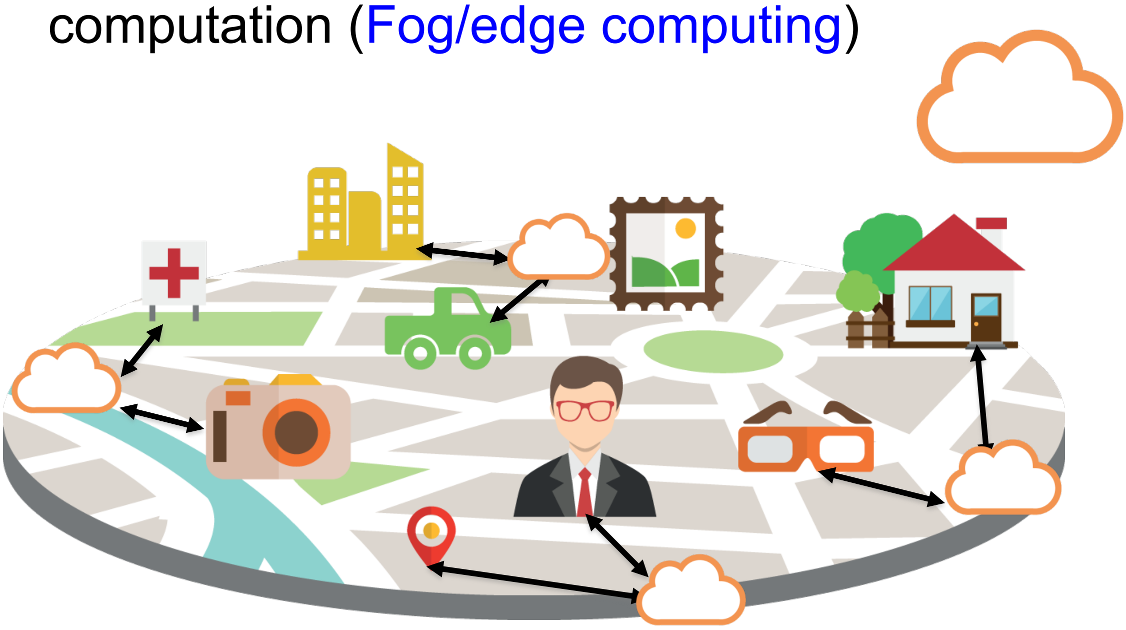
```
SHIP_ID,SPEED,LON2,LAT2,COURSE,HEADING,TIMESTAMP,
departurePortName,Reported_Draught
```


Traditional DSP challenges

- Stream data rates can be high, data arrive in large volumes and data arrival patterns can be highly variable
  - High resource requirements for processing (clusters, data centers, distributed Clouds)
- Processing stream data has real-time aspects
  - Stream processing applications have QoS requirements, e.g., end-to-end latency
  - Must be able to react to events as they occur

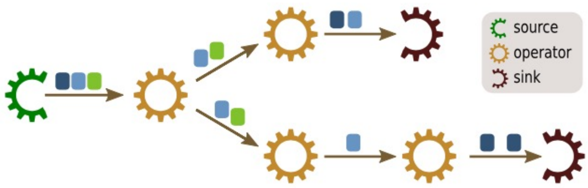
New challenge for large-scale DSP

- Goals: increase scalability and reduce latency
- How? Rely on distributed and near-edge computation (Fog/edge computing)



DSP application model

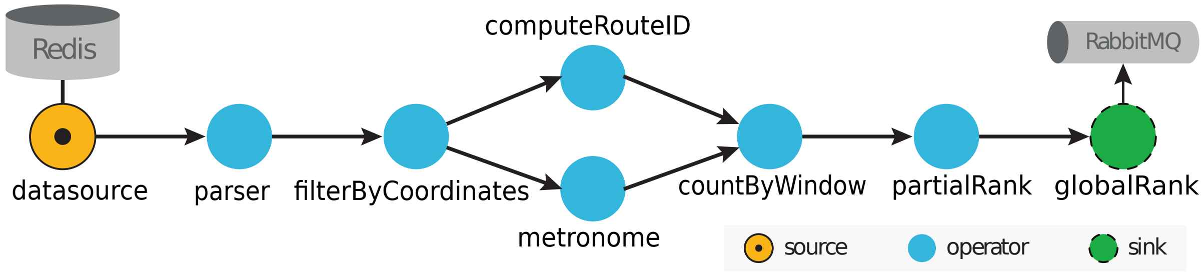
- A DSP application is made of a network of *operators* (processing elements) connected by streams, at least one *data source* and at least one *data sink*
- Represented by a **directed graph**
  - Graph vertices: operators
  - Graph edges: streams
  - Graph is often referred to as topology



- Graph is typically acyclic: directed acyclic graph (DAG)
  - Most systems only support DAGs, few support also cyclic computations (e.g., Flink)
- Application topology does not usually change during processing

DSP application model: example

- Example of DAG for a DSP application



DSP programming model

- Dataflow programming
  - Programming paradigm that models a program as a directed graph of data flowing between operations
  - Pioneered by Jack Dennis and his students at MIT in the 1960s
- Examples
  - Apache NiFi: automates the flow of data between systems
  - Apache Flink: stream and batch processing
  - Apache Beam: unifies batch and streaming data processing on top of several execution engines
  - TensorFlow: ML library based on dataflow programming

DSP programming model

- **Flow composition**: how to create the topology associated with the directed graph for a DSP application

- **Flow manipulation**: use of processing elements (i.e., operators) to perform transformations on data

Data flow manipulation

- How the streaming data is manipulated by the operators in the flow graph?
- Operator properties:
  - Operator type
  - Operator state
  - Windowing

DSP operator

- Self-contained processing element that
  - Transforms one or more input streams into another stream
  - Can execute a generic user-defined code
    - Algebraic operation (filter, aggregate, join, ..)
    - User-defined and possibly complex operation (POS-tagging, machine learning algorithm, …)
  - Can execute in parallel with other operators

Types of operators

- Edge adaptation: converting data from external sources into tuples that can be consumed by downstream operators
- Aggregation: collecting and summarizing a subset of tuples from one or more streams
- Splitting: partitioning a stream into multiple streams
- Merging: combining multiple input streams

Types of operators

- Logical and mathematical operations: applying different logical processing, relational processing, and mathematical functions to tuple attributes
- Sequence manipulation: reordering, delaying, or altering the temporal properties of a stream
- Custom data manipulations: applying data mining, machine learning, ...

DSP operator: state

- Operator can be stateless or stateful
- **Stateless**: solely depends on current input, knows nothing about state and thus processes tuples independently of each other, independently of prior history, or even from tuple arrival order
  - E.g., filter, map
  - Easily parallelizable
  - No synchronization in a multi-threaded context
  - Restart upon failures without the need of any recovery procedure

DSP operator: state

- **Stateful**: keeps some sort of state and thus involves maintaining information across different tuples to detect complex patterns
  - E.g., some aggregation or summary of processed elements, or state-machine for detecting patterns for fraudulent financial transaction
  - State might be shared between operators

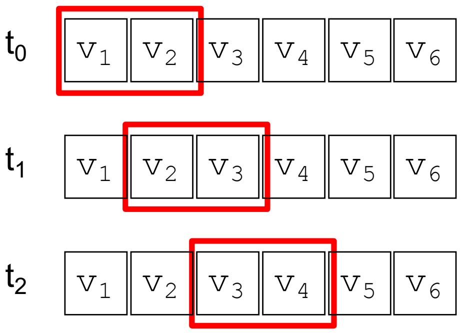
Windowing

- **Window**: buffer associated with an operator input port to retain incoming tuples over which we can apply computations so to process them as a whole
  - E.g., the most frequently purchased items over the last hour
- Window is characterized by:
  - **Size**: amount of data that should be buffered before triggering operator execution
    - Statically defined: time-based (e.g., 30 seconds) or count-based (e.g., the last 100 tuples)
    - Dynamically defined: session-based
  - **Sliding interval**: how the window moves forward
    - Time-based or count-based

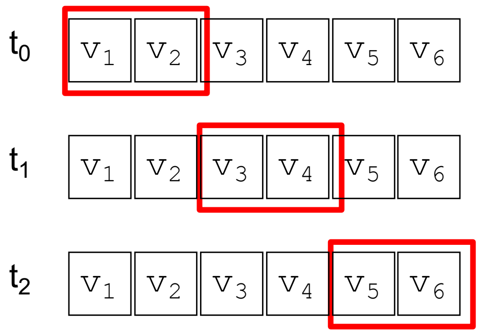
Windowing patterns

- Different windowing patterns by combining window size and sliding interval:
  - **Sliding window**: static window size and sliding interval with value different from window size, single tuples may be included in multiple consecutive windows
  - **Tumbling window**: sliding interval equal to window size, no overlapping of windows

Sliding window (size:2; slide:1)

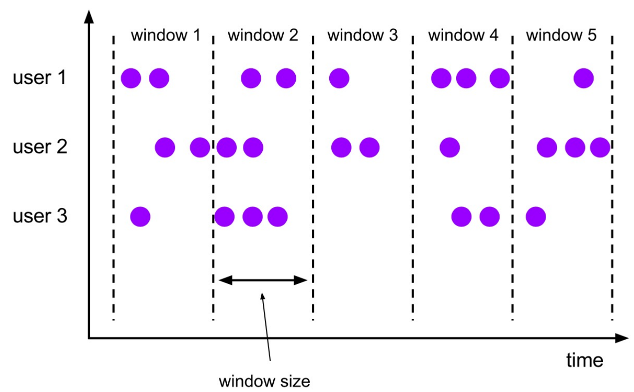


Tumbling window (size:2; slide:2)

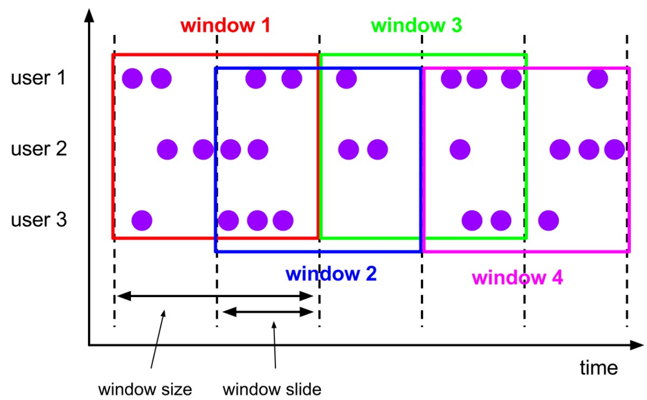


Windowing patterns

Tumbling windows



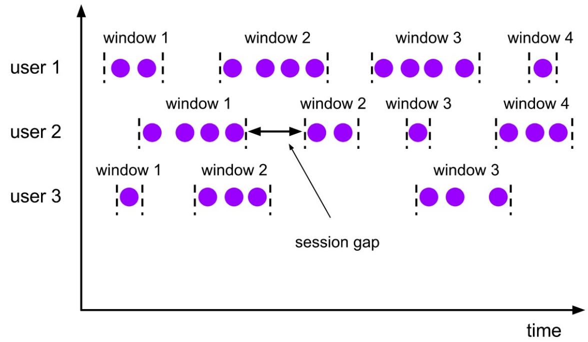
Sliding windows



Windowing patterns

- Window can be also dynamically defined: session window
  - Dynamic size of window length, depending on inputs
  - Starts with an input and expands itself if the following input has been received within the gap duration
  - Closes when there's no input received within the gap duration after receiving the latest input
  - Enables to group events until there are no new events for specified time duration (inactivity)

Session windows

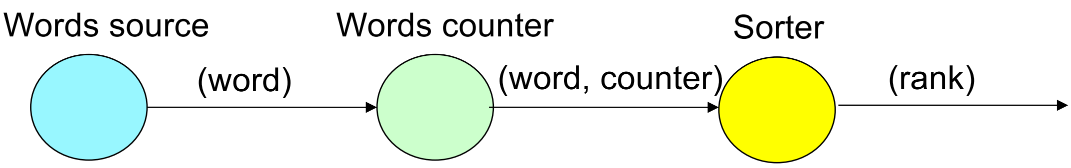


How to define a DSP application

- **Topology description**
  - Explicitly defines operators (built-in or user-defined) and links through a DAG
  - Used in Flink, Storm, Spark Sstreaming, …
- **Formal language**
  - Declarative language that specifies result (SQL-like)
    - e.g., Streams Processing Language (SPL) in IBM Streams
  - Imperative language that specifies composition of basic operators
    - e.g., SQuAl (Stream Query Algebra) used in Aurora/Borealis
- The first offers more flexibility, the latter more rigor and expressiveness

“Hello World”: a variant of WordCount

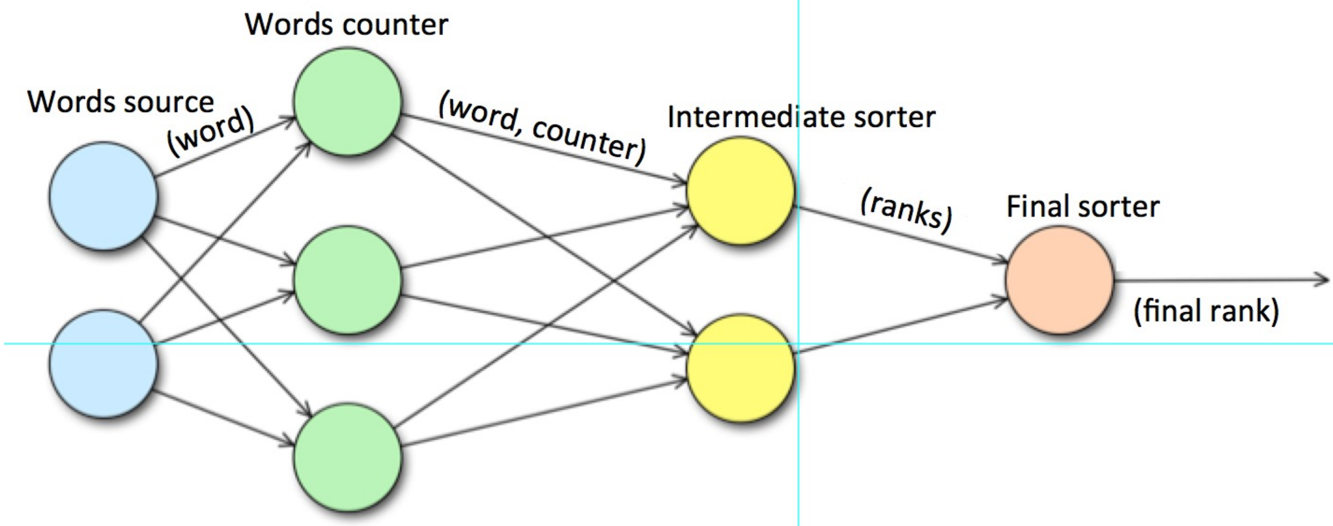
- Goal: emit the top-k words in terms of occurrence when there is a rank update



- Where are the bottlenecks?
- How to scale the DSP application in order to sustain the traffic load?

“Hello World”: a variant of WordCount

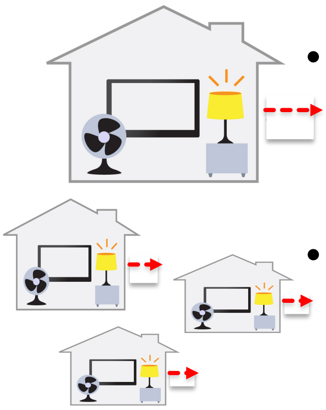
- The usual answer: replication!
- Let’s use data parallelism (aka operator fission) and redesign the application



Example of DSP application: DEBS'14 GC

https://debs.org/grand-challenges/2014/

- Real-time analytics over high volume sensor data: analysis of energy consumption measurements for smart homes
  - Smart plugs deployed in households and equipped with sensors that measure values related to power consumption



- Input data stream:

  `2967740693, 1379879533, 82.042, 0, 1, 0, 12`

- *Query 1*: make load forecasts based on current load measurements and historical data
  - Output data stream:

    `ts, house_id, predicted_load`

- *Query 2*: find the outliers concerning energy consumption
  - Output data stream:

    `ts_start, ts_stop, household_id, percentage`

Example of DSP application: DEBS'15 GC

https://debs.org/grand-challenges/2015/

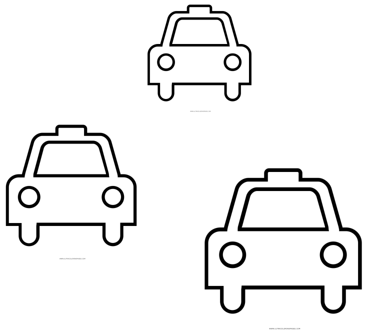
- Real-time analytics over high volume spatio-temporal data streams: analysis of taxi trips based on data streams originating from New York City taxis
- Input data streams: include starting point, drop-off point, corresponding timestamps, and information related to the payment

```
07290D3599E7A0D62097A346EFCC1FB5,E7750A37CAB07D0DFF0AF
7E3573AC141,2013-01-01 00:00:00,2013-01-01
00:02:00,120,0.44,-73.956528,40.716976,-
73.962440,40.715008,CSH,3.50,0.50,0.50,0.00,0.00,4.50
```

Example of DSP application: DEBS'15 GC

https://debs.org/grand-challenges/2015/

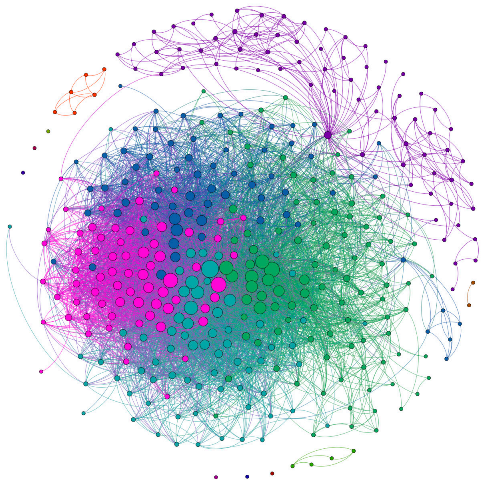
- *Query 1*: identify top-10 most frequent routes during the last 30 minutes
- *Query 2*: identify areas that are currently most profitable for taxi drivers
- Both queries rely on a sliding window operator
  - Continuously evaluate the query results



Example of DSP application: DEBS'16 GC

https://debs.org/grand-challenges/2016/

- Real-time analytics for a dynamic (evolving) social-network graph
- *Query 1*: identify the posts that currently trigger the most activity in the social network
- *Query 2*: identify large communities that are currently involved in a topic
- Require continuous analysis of dynamic graph considering multiple streams that reflect graph updates



Distributed DSP system

- A distributed system that executes stream topologies
  - continuously calculates results for long-standing queries
  - over potentially infinite data streams
  - using operators, that can be stateless or stateful
- System nodes may be heterogeneous
  - Computing capacity, bandwidth, …
- Must be highly optimized and with minimal overhead so to deliver real-time response for high-volume DSP applications
- Must manage a number of issues
  - Operator placement on computing nodes
  - Node and operator failures
  - …

Distributed DSP system

- Usually run in locally distributed clusters within large data centers
- Assumptions:
  - Scale out and not scale up
    - Commodity servers
    - Data-parallelism is king
  - Software designed for failures

Source: Google

- Which software frameworks for distributed DSP systems?

DSP frameworks: processing model

- Main stream processing models:
  - ***One-at-a-time***: each tuple is individually processed
  - ***Micro-batched***: tuples are grouped before being processed

	One-at-a-time (e.g., Apache Storm)	Micro-batched (e.g., Apache Spark Streaming)
Lower latency	✓	
Higher throughput		✓
At-least-once semantics	✓	✓
Exactly-once semantics	In some cases	✓
Simpler programming model	✓	

Source: N. Marz, J. Warren, Big Data, Manning Pub., 2015.