SJK006 - Master in Intelligent Systems

Data Streaming

Analysing dynamic data

Spark Streaming

Reading and querying continuous data streams

Structured Streaming

- Data frames as streams
- Sources of streams
- Managing streams
- Querying streams

What is a data stream?

A data stream is an infinite sequence of events or observations:

- Data is usually associated with a timestamp
- The observation/event is represented either as raw data (text), tuples (SQL) or as complex objects (JSON)
- Data stream generators:
  - Sensors (Internet Of Things -IoT-)
  - Social networks (Twitter, LinkedIn, Facebook, etc.)
  - News channels (Feeds)
  - Any temporary measure (prices, indices, etc.)

https://www.pubnub.com/demos/real-time-data-streaming/?show=demo

What is data streaming?

They are the techniques to efficiently process data incrementally:

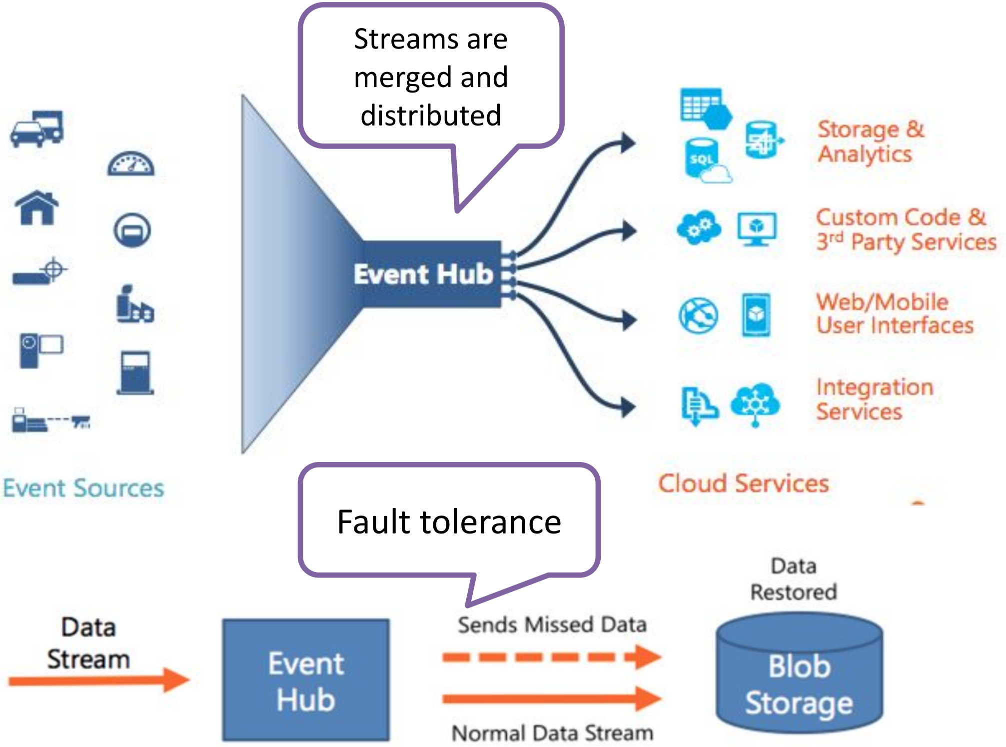
- Real-time processing:
  - buffer the streamed data
  - process buffered data (the result is also data stream)
  - get rid of buffered data
- It requires event-based programming (e.g., Nodejs)
- It requires window operations (time-based buffers)
  - Tumbling and sliding windows
  - Distribution & join of windows
- Real-time vs batch processing

Data streams in Data Science

The data science hypotheses about data streams are associated with:

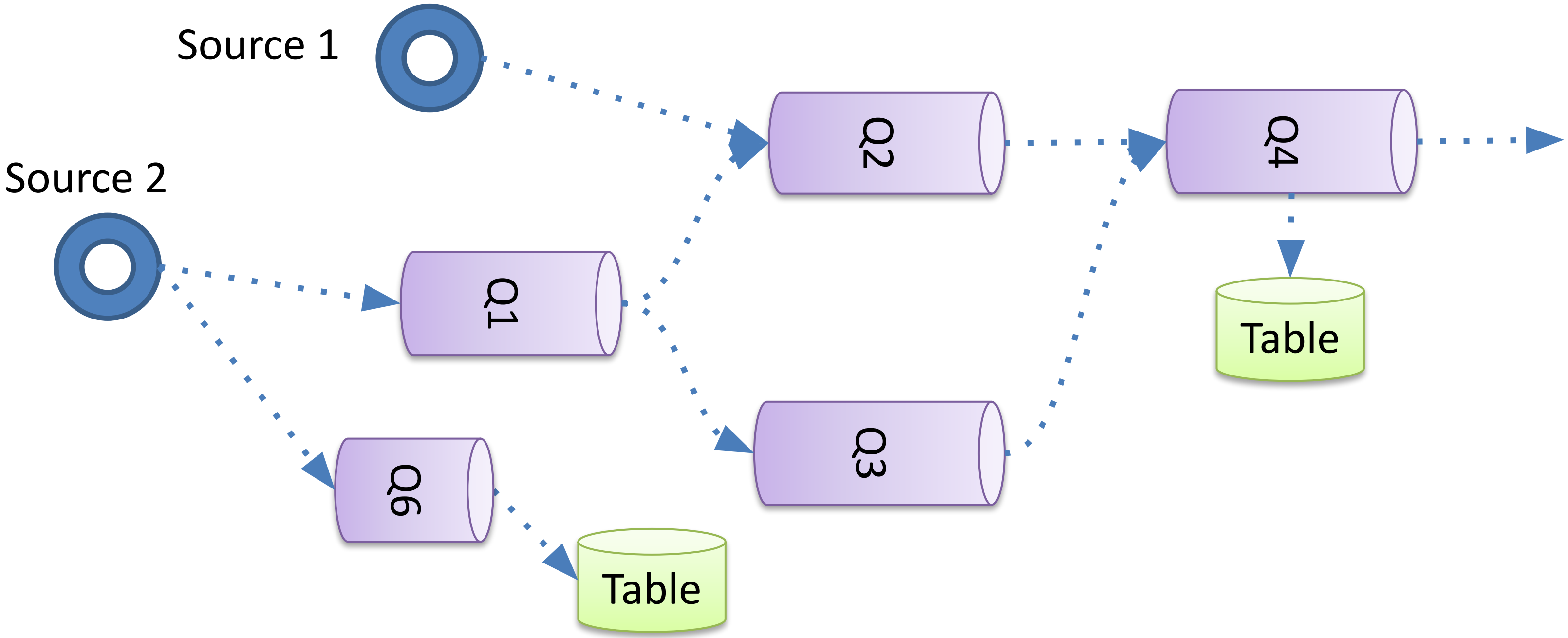
- Monitoring and Prevention (alarms, anti-fraud, cyber-attacks, etc.)
- Agents and decision making (purchases/sales, marketing, etc.)
    - Related to Reinforcement Learning processes
- Concept drift:
    - data changes and makes the analytical models obsolete
    - it triggers the re-training or tuning of current models
- Causal associations between data streams and external complex events
    - Breakdowns, Accidents, Breakages, Lack of stock, etc.
    - Reputation, Risk, Fraud, Trends, etc.

Event hubs



Azure Streaming

Combining streams



Data Streaming in Spark

Spark Streaming

Old API vs. Structured Streaming

OLD WAY: **Discrete Data Streams** (DStream). This is deprecated.

```
…
from pyspark.streaming import StreamingContext

sc = SparkContext(...)
ssc = StreamingContext(sc, 30) #seconds
sourceRDD = ssc.textFileStream(data_dir)

## transformations on sourceRDD

sourceRDD.foreachRDD(process)

ssc.start()
ssc.awaitTermination()
```

CURRENT WAY: **Structured Data Streaming**

```
dfs = spark.readStream\
          .schema(schema) \
          .option("header", True) \
          .format("csv") \
          .load("./stream_dir")

## transformations
res_df = dfs.select("*")

query = (res_df.writeStream \
            .format("memory") \
            .queryName("selectTable") \
            .outputMode("complete") \
            .start() \
            .awaitTermination() )
```

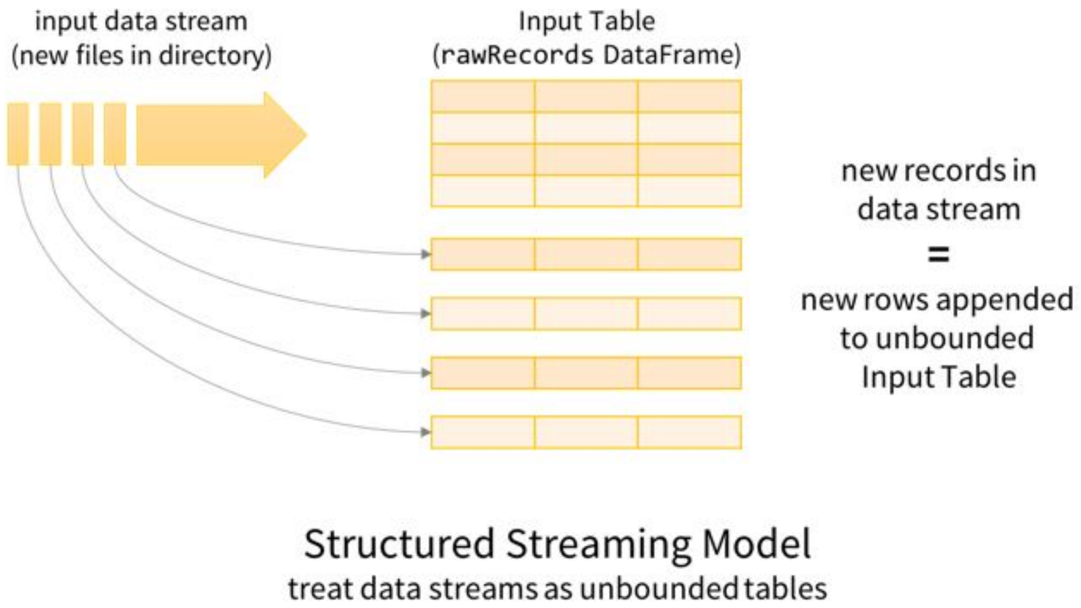
Intermission:
Structured data processing with Spark

Basics of Spark Streaming

- Spark Streaming uses micro-batch processing
  - Buffers are RDDs (deprecated) or Data Frames
  - Micro-batches are triggered by events
- Key concepts:
  - Unbounded tables
  - Triggers
  - Watermarking

Minibatch-based processing

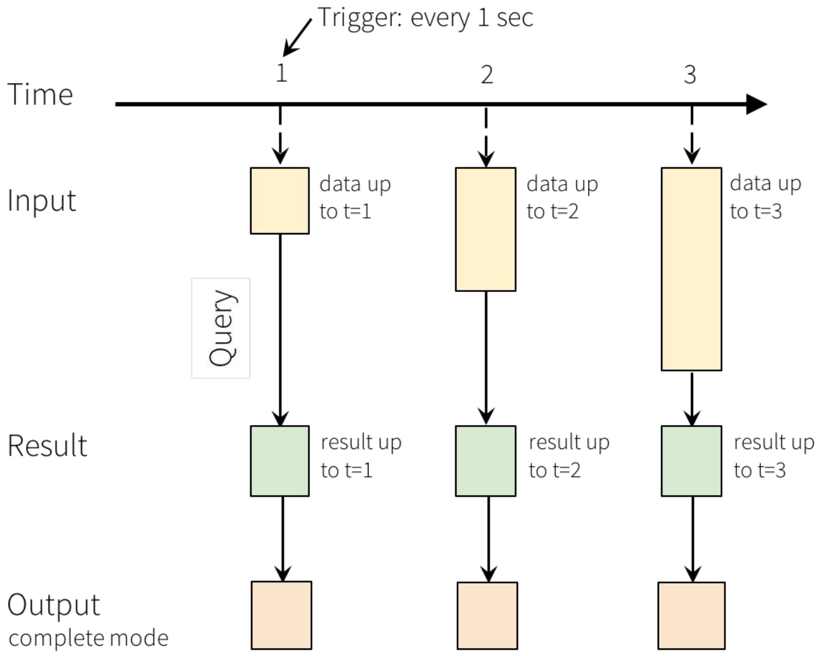
Unbounded tables



Triggers

- Define how often streaming computations are performed and results are produced.
- Controls the latency and throughput of the streaming application..
- Balances resource utilization and result timeliness.
  - Shorter intervals reduce latency but may increase resource usage.
  - Choose triggers based on application requirements.
- Types of Triggers:
  - Fixed Interval Micro-Batch: Executes the query at regular intervals (e.g., every 1 second)
    - `.trigger(ProcessingTime("1 second"))`
  - One-Time (Batch) Trigger: Processes all available data once and then stops.
    - `.trigger(Once())`
  - Continuous Processing (Experimental): Processes data continuously with low latency. Example:
    - `.trigger(Continuous("10 seconds"))`

Triggers



Programming Model for Structured Streaming

Watermarking

- In real-world streams, data can arrive late due to network delays or out-of-order events.
- Watermarking defines how to handle late-arriving data by setting a threshold of how long to wait for late data.
- It helps manage state and ensures timely results.
- Use timestamps from the data itself (event time) instead of processing time.
- The watermark is the maximum allowed lateness
    - .withWatermark("eventTime", "10 minutes")
- Aggregations retain state only for data within the watermark threshold.
- Late data beyond the watermark is discarded or handled separately.

Questions?