POLITECNICO
MILANO 1863

QUANTIA
consulting

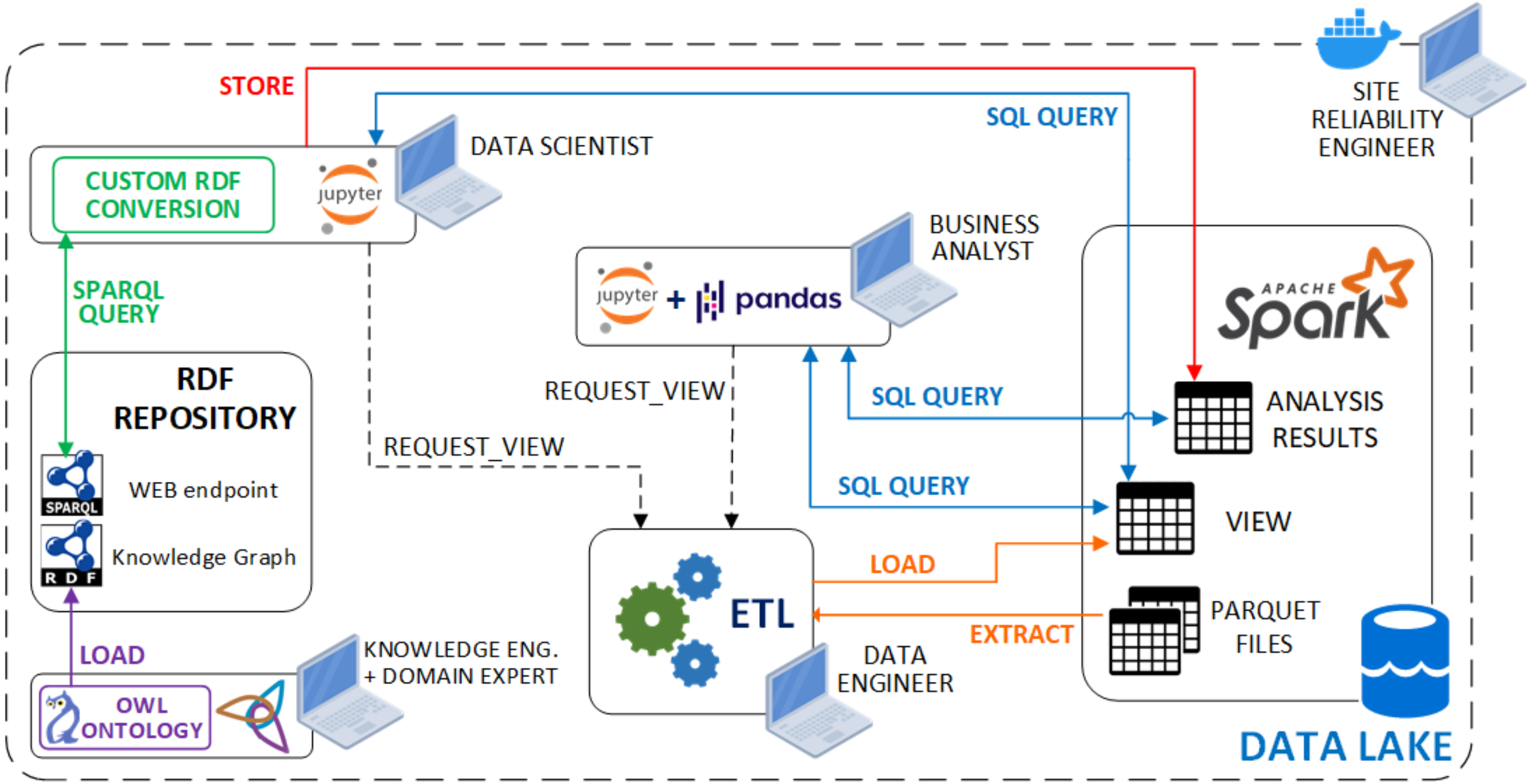
Web Stream Processing with OntopStream

AUTORI E DETTAGLI
AUTORI E DETTAGLI

Business Scenario

Semantic and Big Data technologies are separated

- Data lakes : store the whole enterprise data. Analysts need custom **Extract Transform Load** (ETL) jobs to access the data
- Knowledge Graphs : queried with **SPARQL** to extract semantical information



Traditional Data Analysis - PROBLEMS

- Problem-dependent tasks:
  - new analytical query → new ETL task from scratch
  - ETLs require several days of work and meetings
  - requires a lot of Data Engineers workforce

- Semantical analyses persistence in the data lake, for later re-use, is difficult

Solvable using a combination of *multiple tools*, which *increase the required skills*

↓

need for ***single***, ***user-friendly***, ***straightforward*** tool

Ontology-Based Data Access

- **Ontology-Based Data Access (OBDA)** softwares aim to solve data integration problems…

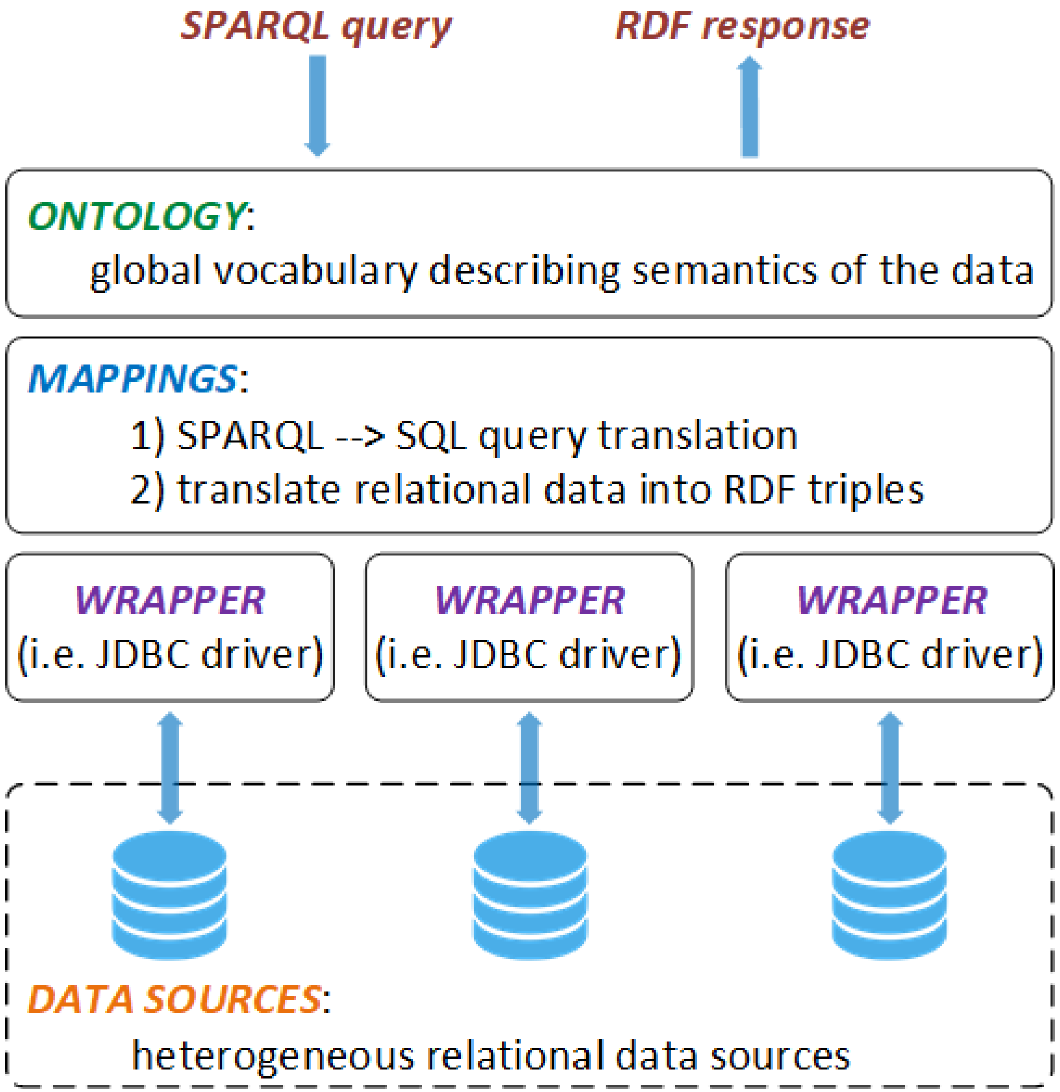
- **Virtual Knowledge Graph (VKG)** approach:
  - additional semantic layer on top of the data
  - relational data sources abstraction, exposed as RDF triples
  - SPARQL queries to access the data
  - automatic SPARQL → SQL query rewritings

Vitual Knowledge Graph approach

Data Analyst/Scientist

Knowledge Engineer

Data Engineer



Virtual Knowledge Graph engines

- “traditional” VKG engines (Mastro, Morph-RDB, UltraWrap)

- **Ontop** is considered the state-of-the-art reference VKG engine:
  - the only one offered as a commercial solution
  - active Github community (weekly-based issues)
  - relevant industrial-grade implementations
    - Statoil (Equinor)
    - Siemens Electric
    - Ricerca sul Sistema Energetico s.p.a

However, none of the tool is designed for supporting streams of data

Streaming Technologies

- Streaming technologies are becoming very popular…

- Data Streams can be:
  - continuously generated
  - incrementally processed
  - segmented by their time (window)

- Stream Processing engines enables real-time processing and querying of multiple data streams

State of the art streaming technologies…

OPEN-SOURCE

- APACHE kafka®
- Spark Streaming
- Flink
- APACHE nifi
- ksqDB
- APACHE calcite™

CLOUD-BASED (proprietary)

- Cloud DataFlow
- amazon Kinesis
- Azure Stream Analytics

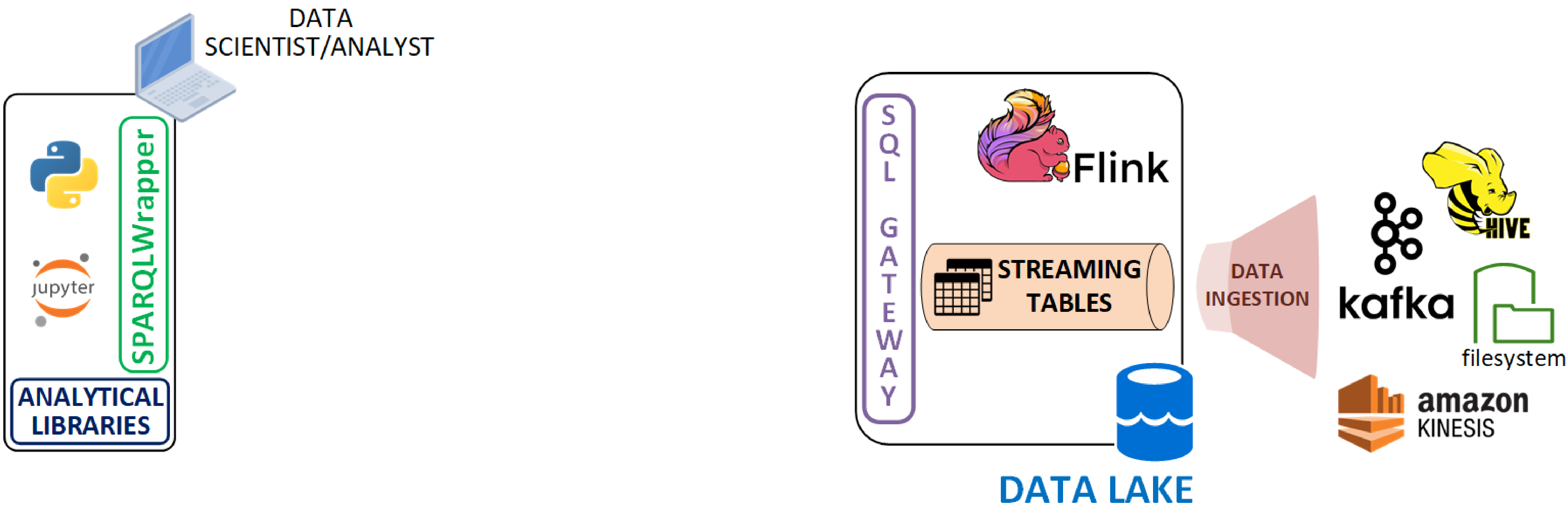
Towards real-time analytics

- Stream Processing engines enables real-time processing and querying of multiple data streams

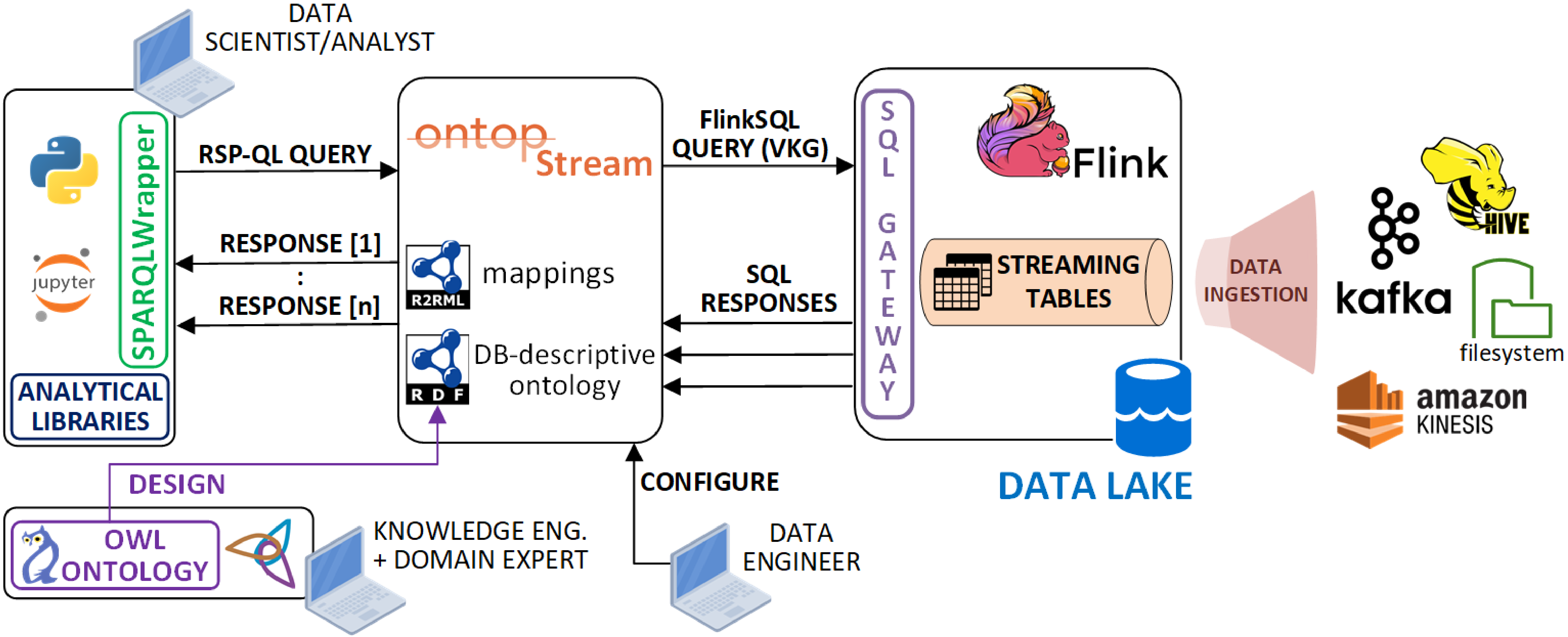
- need for a real-time tool leveraging:
  - State-of-the-art open-source streaming technologies
    - Apache Flink, Apache Kafka, Apache Calcite
  - Streaming extensions of semantic technologies
    - RDF Stream Processing Query Language (RSP-QL)
    - Streaming-VKG

OntopStream

KG-Empowered Continuous Analytics



Streaming-VKGs as a bridge between Stream Processing and Semantic Techs



OntopStream

- Developed as an extension of the Ontop OBDA system (Java)

- Query relational data streams
  - stored and managed in Apache Flink dynamic tables
  - with RSP-QL continuous queries (windowed / not windowed)

- Get RDF streams of responses

- Two distributions:
  - OntopStream-CLI
  - OntopStream-Endpoint (only HTTP calls)

OntopStream: design decisions

- paradigm shift from traditional OBDA to Streaming-OBDA

- design decisions:

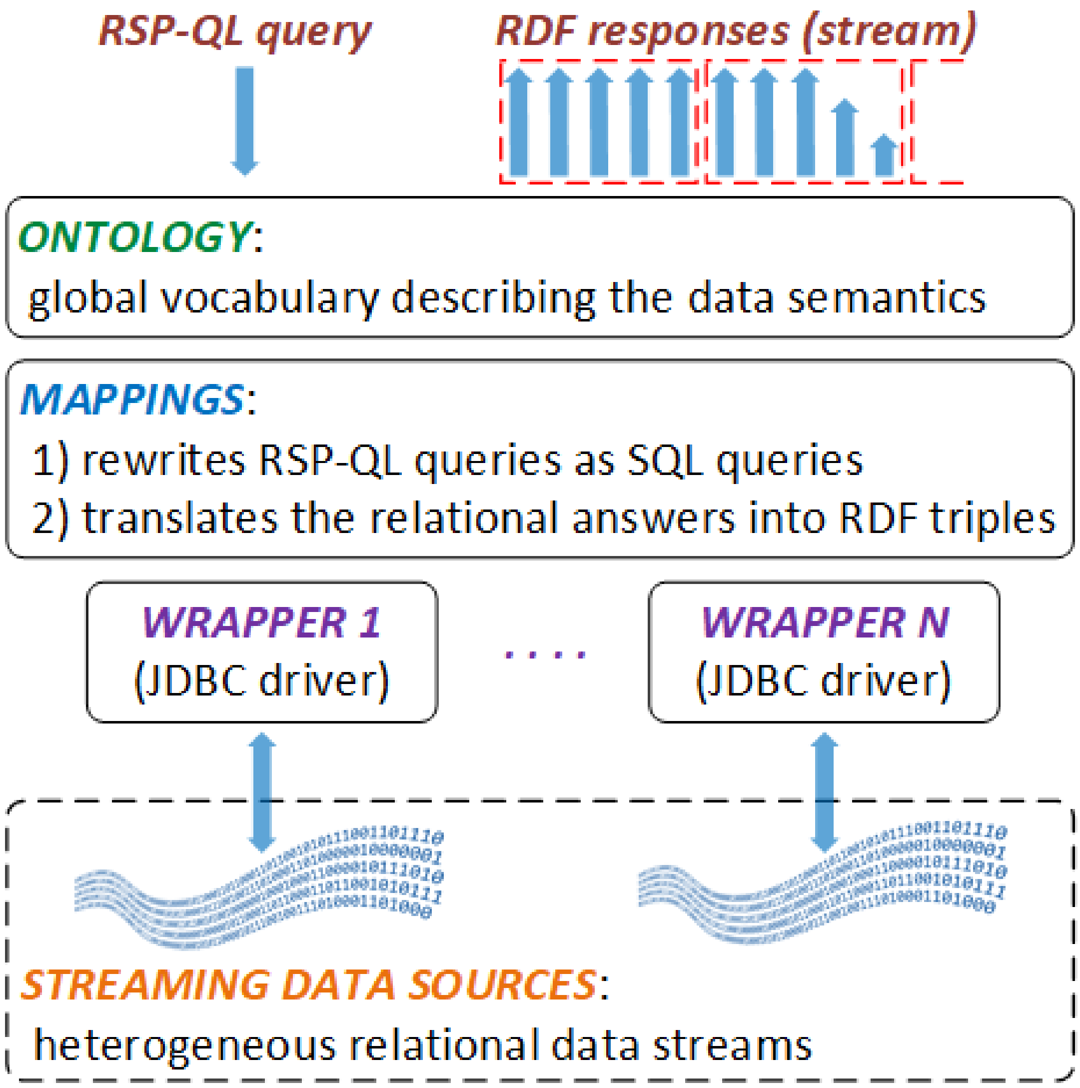
  1. extend the `Flink JDBC driver`
  2. re-design part of the `ontop-engine` to accept `RSP-QL` queries
  3. Streaming Virtual Knowledge Graph query rewriting approach
  4. include support for `RDF streams` of query outputs

Streaming Virtual Knowledge Graphs in OntopStream

Data Analyst/Scientist

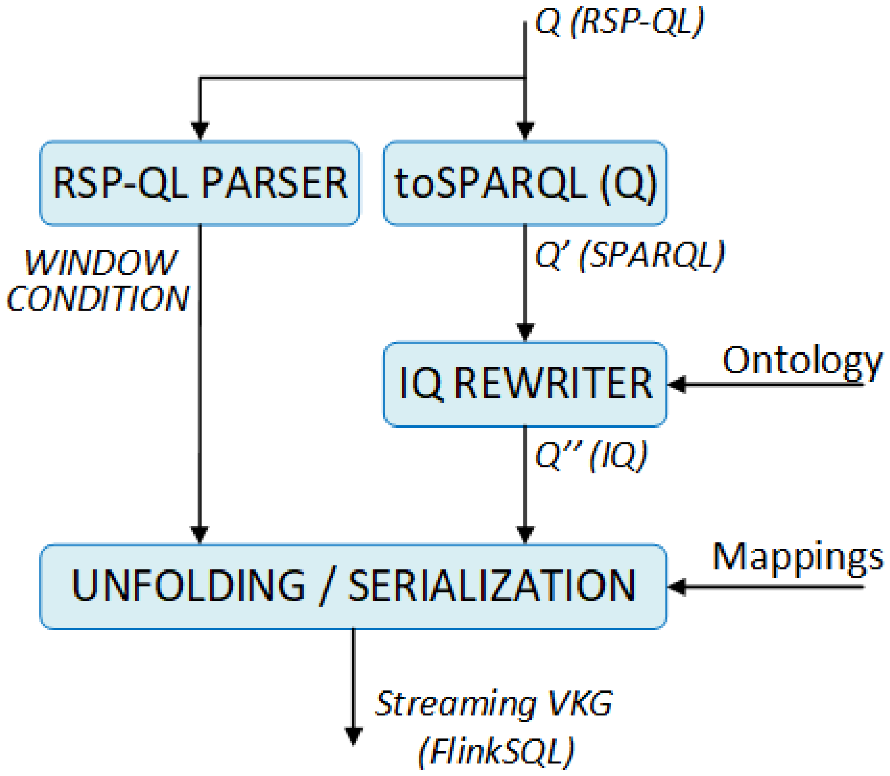
Knowledge Engineer

Data Engineer



Streaming Virtual Knowledge Graph query rewriter

- **`rsp4j`** parser to extract window conditions
- Intermediate Query rewriter unchanged
- **`IQ`** representation:
  - created w.r.t to the Ontology **`O`**
  - unfolded in a **`Streaming VKG`** tree
- Each tree node corresponds to a pseudo-SQL statement
- Streaming VKG serialization in a **`FlinkSQL query`**, add window condition **`W`** if existing



Time to practice

Business Scenario: Rental Company

- A car rental company has recently decided to **unify the information systems** of **two branches** using ontology-based data access techniques.

- Both the branches:
  - have a real-time data management infrastructure
  - store the rental records in Kafka topics

- However, they handles the data differently:
  - *Branch A* uses two separate Kafka topics for trucks and cars
  - *Branch B* stores all the rentals in a single topic, but the users' data are kept in a sperate topic

Business Requirements

The company is booming, and has in plan to acquire soon new branches.

Therefore, the company wants to make the integration process scalable, so that can be easily extended to all its new branches

They need a data integration solution that:

- provides an **unified logical view** of their data
- enables to **query in real-time** their data
- can be used with **python notebooks** for further analyses

Kafka topics: Branch A

user	rid	manufacturer	model	plate	status
Molly Davis	1	Fiat	Panda	FJ7PUJJ	START
Laura Baker	2	Tesla	Model S	JFGJ60A	START
William Diaz	3	Fiat	Tipo	FGL1X62	START
Molly Davis	1	Fiat	Panda	FJ7PUJJ	END
William Diaz	3	Fiat	Tipo	FGL1X62	END

DEALER1_CARS

user	rid	manufacturer	model	plate	status
Laura Baker	1	Iveco	Daily	HHST532	START
Wayne Flower	2	Fiat	Ducato	DM89JKD	START
Richard Tillman	5	Fiat	Ducato	JSDJFI3	START
Richard Tillman	5	Fiat	Ducato	JSDJFI3	END
Wayne Flower	2	Fiat	Ducato	DM89JKD	END

DEALER1_TRUCKS

Kafka topics: Branch B

DEALER2_VEHICLES

userID	rid	type	manufacturer	model	plate	status
3	1	Car	Audi	A3	DFU4HJF	START
4	2	Car	Mercedes	Classe C	784JD93	START
3	7	Truck	Mercedes	Vito	KD94KDS	START
3	1	Car	Audi	A3	DFU4HJF	END
6	8	Truck	Mercedes	Vito	012JKD0	START
3	7	Truck	Mercedes	Vito	KD94KDS	END

DEALER2_USERS

userID	name
1	Douglas Fitch
2	William Diaz
3	Kevin Rodriguez
4	Catherine Crandell
5	Richard Tillman

Kafka topics: Flink ingestion

- Data acquisition in Flink can be automated

- Design the topics ingestion in Flink:
  - Flink streaming tables
    - queried with FlinkSQL continuous queries, recorded in Flink
  - Kafka connector for Flink (Table & SQL API):
    - *files:*
      - ***sql-client-conf.yaml***: Kafka → Flink
      - ***sql-gateway-defaults.yaml***: Flink JDBC Gateway
    - **table schema**: topics fields, datatypes, watermarks, …
    - **connector properties**: Kafka address, schema registry, …

Example: DEALER2_VEHICLES topic

TABLE SCHEMA

```
- name: D2_VEHICLES
  type: source
  update-mode: append
  schema:
  - name: userID
    type: BIGINT
  - name: rid
    type: BIGINT
  - name: type
    type: STRING
  - name: manufacturer
    type: STRING
  - name: model
    type: STRING
  - name: plate
    type: STRING
  - name: status
    type: STRING
  - name: ts
    type: STRING
```

TABLE NAME

data: SOURCE → FLINK

append new data, when is available from the source (Kafka)

Reference: [Apache Kafka SQL Connector]

Example: DEALER2_VEHICLES topic

TABLE SCHEMA

```
- name: parsed_timestamp
    type: TIMESTAMP
    rowtime:
      timestamps:
        type: "from-field"
        from: "ts"
      watermarks:
        type: "periodic-bounded"
        delay: "5"
connector:
    property-version: 1
    type: kafka
    version: universal
    topic: DEALER2_VEHICLES
    startup-mode: earliest-offset
    properties:
    - key: bootstrap.servers
      value: kafka:9092
  format:
    property-version: 1
    type: json
    schema: "ROW(userID BIGINT, rid BIGINT, type STRING, manufacturer STRING,
                model STRING, plate STRING, status STRING, ts STRING)"
```

SOURCE CONNECTOR

reference field for time-based operations (WINDOWS)

source type

source address

topic schema

Relational Streaming Data Integration…

Now, we have a Flink streaming table for each Kafka topic

- *DEALER1_CARS* and *DEALER1_TRUCKS*
- *DEALER2_VEHICLES* and *DEALER2_USERS*

The data streams are still not integrated!!!

Relational Streaming Data Integration…

Now, we have a Flink streaming table for each Kafka topic

- *DEALER1_CARS* and *DEALER1_TRUCKS*
- *DEALER2_VEHICLES* and *DEALER2_USERS*

We can use **OntopStream** to create a **unified logical view** of the data streams…

Flink relational streams:

- exposed to OntopStream using the ***Flink JDBC Gateway***
- can be queried with ***FlinkSQL*** continuous queries

Starting-up the resources

- Requirements: *docker* and *docker-compose*

- Start the tutorial environment
  - Streaming resources (Flink, Kafka) and JupyterLab
    - *sudo docker-compose -f flink-kafka.yml up –d*
  - Flink JDBC Gateway:
    - *sudo docker-compose -f flink-kafka.yml exec sql-client /opt/flink-sql-gateway-0.2-SNAPSHOT/bin/sql-gateway.sh --library /opt/sql-client/lib*
    - **Note**: keep the JDBC endpoint alive until you need the service (don't close the terminal window)
  - OntopStream (new terminal window):
    - *sudo docker-compose -f ontop.yml up -d*