Big Data Analytics for Semantic Data
BigSem Tutorial

Module 3: Semantic Data Analytic Engines and Frameworks

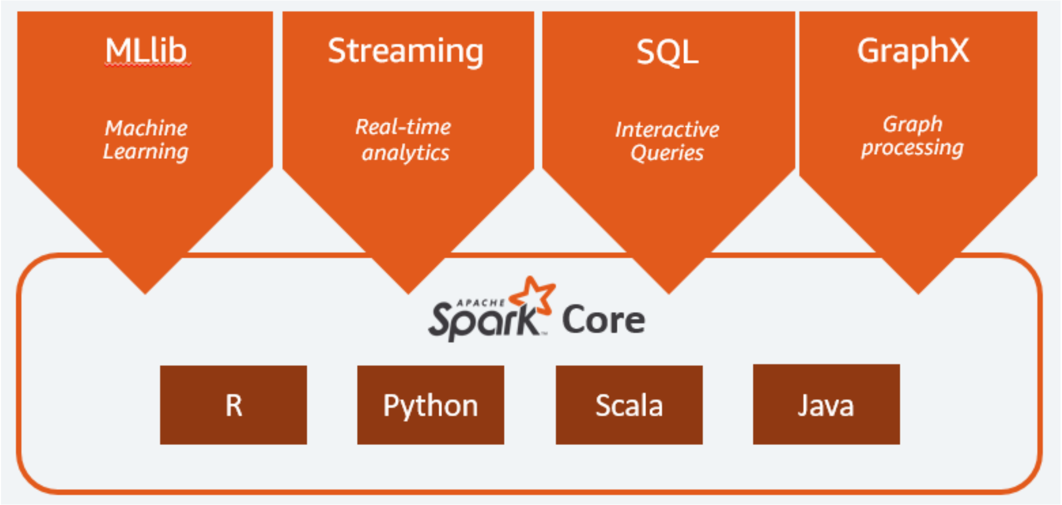
Chelmis Charalampos, Bedirhan Gergin
University at Albany, SUNY

ISWC 2024

goo.gl/6Nrc1r

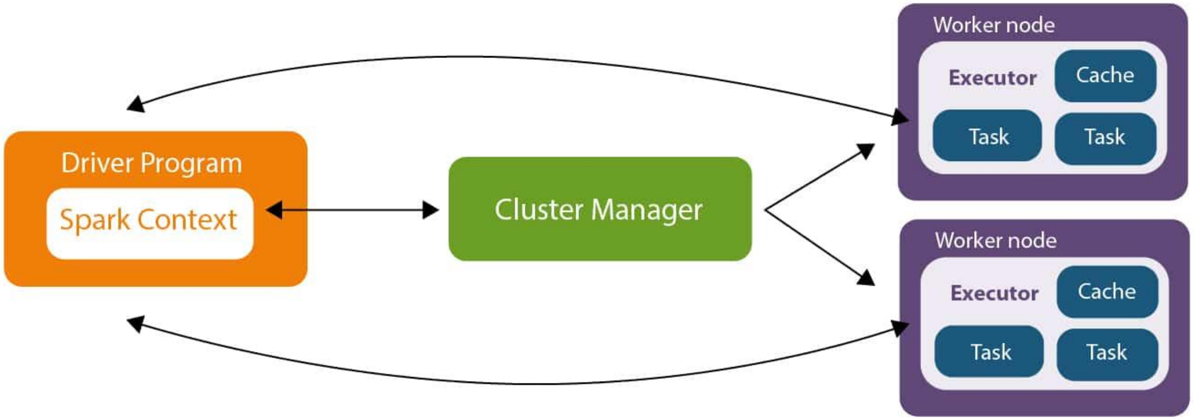
Apache Spark

- Apache Spark is an open-source cluster computing system that provides high-level API in Java, Scala, Python



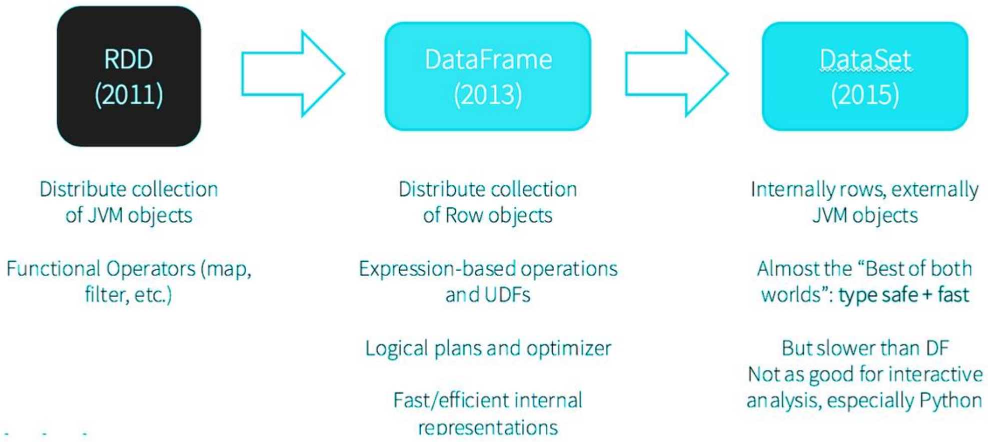
Apache Spark: Architecture

- **Driver Program:** The Spark Context manages job execution and distributes tasks to the worker nodes
- **Cluster Manager:** Allocates resources across the cluster. It communicates with both the driver program and the worker nodes to manage resource allocation
- **Worker Nodes:** where the actual execution of tasks happens. Each worker node contains executors, which are responsible for running the tasks



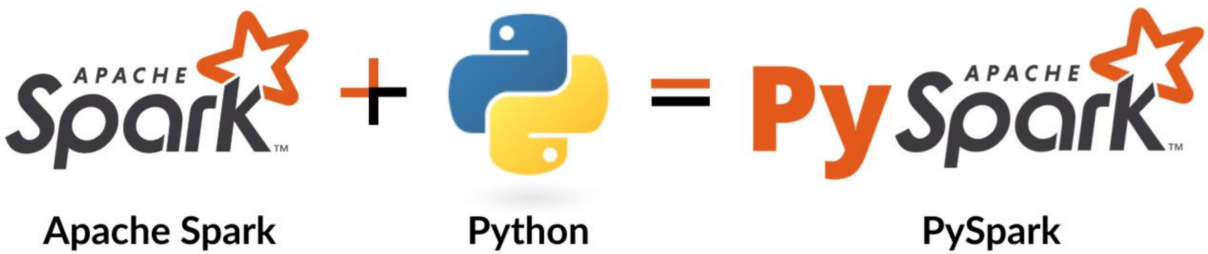
Apache Spark: RDD

- An RDD (Resilient Distributed Dataset) in Apache Spark is a fundamental data structure that represents an immutable, distributed collection of objects that can be processed in parallel across a cluster



Apache Spark in Python: PySpark

- PySpark is the Python API for Apache Spark.
- It enables you to perform real-time, large-scale data processing in a distributed environment using Python



PySpark: SparkSession

- **config**: This allows users to specify additional Spark configuration settings to customize the application further
- **spark.executor.memory**: This parameter specifies the amount of memory allocated per executor process, such as 2g for 2 gigabytes
- **spark.executor.cores**: This defines the number of CPU cores allocated to each executor, impacting the parallelism and speed of the application
- **spark.driver.memory**: This indicates the amount of memory reserved for the driver process, with a common setting being 1g

```python
import findspark
findspark.init()
from pyspark.sql import SparkSession

# Initialize SparkSession with additional configurations
spark = SparkSession.builder \
    .appName('PySpark Beginner Tutorial') \
    .master('local[*]') \
    .config('spark.executor.memory', '2g') \
    .config('spark.executor.cores', '2') \
    .config('spark.driver.memory', '1g') \
    .getOrCreate()
```

PySpark: Creating Dataframe

- Creating spark dataframe manually

```
data = [('John', 28, 'M'), ('Anna', 23, 'F'), ('Mike', 35, 'M'), ('Sara', 31, 'F')]
columns = ['Name', 'Age', 'Sex']

# Create a DataFrame with additional columns
df = spark.createDataFrame(data, columns)

# Show DataFrame
df.show()
```

```
+----+---+---+
|Name|Age|Sex|
+----+---+---+
|John| 28|  M|
|Anna| 23|  F|
|Mike| 35|  M|
|Sara| 31|  F|
+----+---+---+
```

PySpark: Dataframe Operations

- Here are some common DataFrame operations such as checking schema, selecting columns, filtering rows, and computing basic statistics

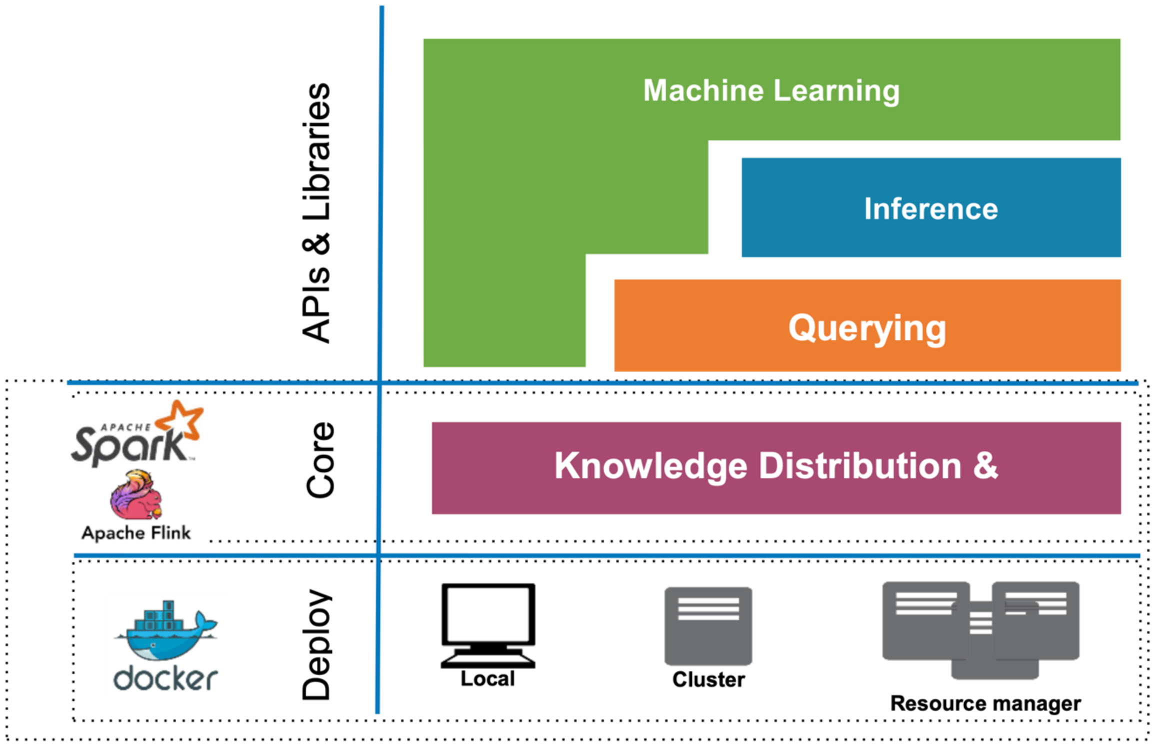
```python
# Check the schema of the DataFrame
df.printSchema()

# Select the 'Name' column
df.select('Name').show()

# Filter rows where age > 30
df.filter(df.Age > 30).show()

# Compute basic statistics
df.describe().show()
```

```
root
 |-- Name: string (nullable = true)
 |-- Age: long (nullable = true)
 |-- Sex: string (nullable = true)

+----+
|Name|
+----+
|John|
|Anna|
|Mike|
|Sara|
+----+
```

```
+----+---+---+
|Name|Age|Sex|
+----+---+---+
|Mike| 35|  M|
|Sara| 31|  F|
+----+---+---+

+-------+----+
|summary| Age|
+-------+----+
|  count|   4|
|   mean|29.25|
| stddev| 4.573474244670772|
|    min|  23|
|    max|  35|
+-------+----+
```

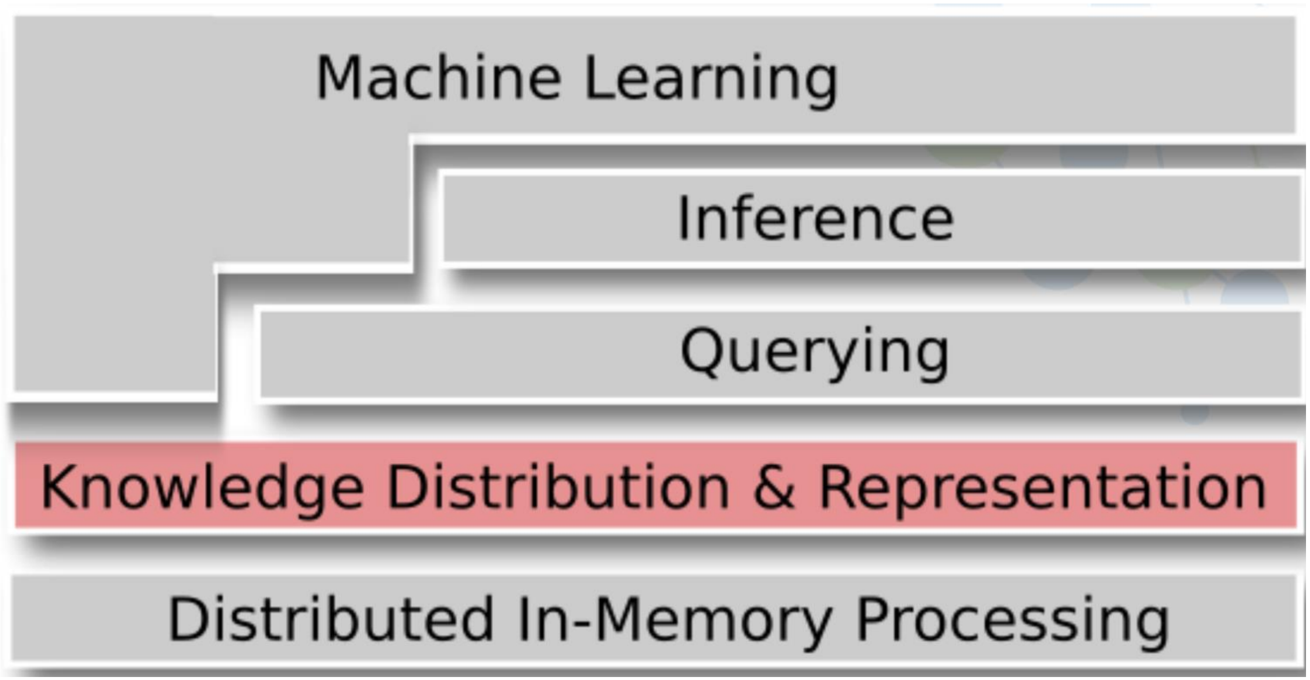
SANSA Stack [9]

- Scalable Semantic Analytic Stack (SANSA) framework is an open-source distributed data flow engine which allows scalable analysis of large-scale RDF datasets



SANSA Stack – RDF Processing Layer

- SANSA provide mechanism of reading RDF model in the format of RDD/DataFrame/Dataset of triples



SANSA Stack – RDF Processing Layer

- SANSA provide mechanism of reading RDF model in the format of RDD/DataFrame/Dataset of triples

Listing 1. Triple reader example.

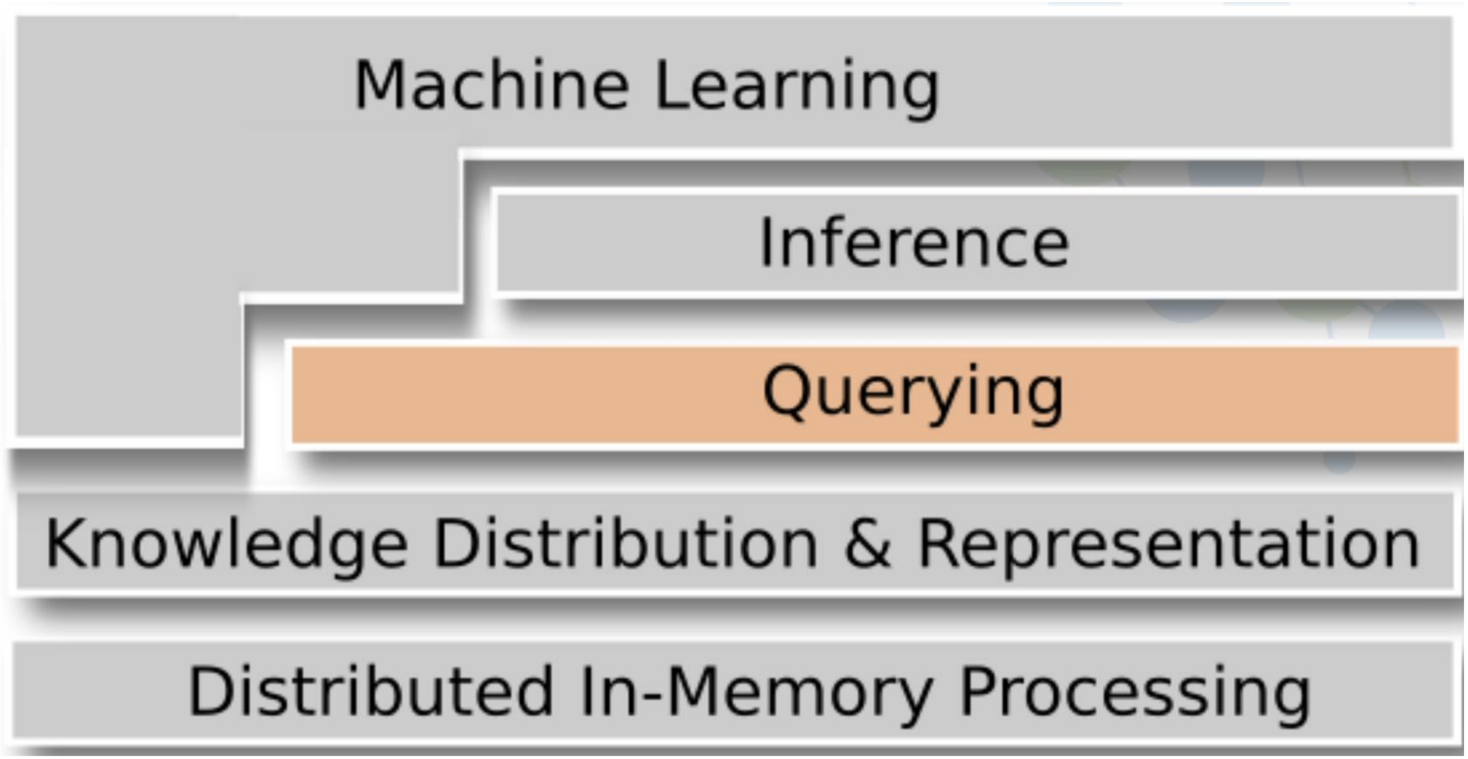
```
import net.sansa_stack.rdf.spark.io._
import org.apache.jena.riot.Lang

val input = "hdfs://namenode:8020/data/rdf.nt"
val lang = Lang.NTRIPLES

val triples = spark.rdf(lang)(input)

triples.take(5).foreach(println(_))
```

Listing 2. Triple writer example.

```
import net.sansa_stack.rdf.spark.io._
import org.apache.jena.riot.Lang

val input = "hdfs://namenode:8020/data/rdf.nt"
val lang = Lang.NTRIPLES

val triples = spark.rdf(lang)(input)

triples.saveAsNTriplesFile(output)
```

SANSA Stack – Querying Layer

- The default approach for querying RDF data in SANSA is based on SPARQL-to-SQL. It uses a flexible triple-based partitioning strategy on top of RDF (such as predicate tables with sub partitioning by data types)



SANSA Stack – Querying Layer

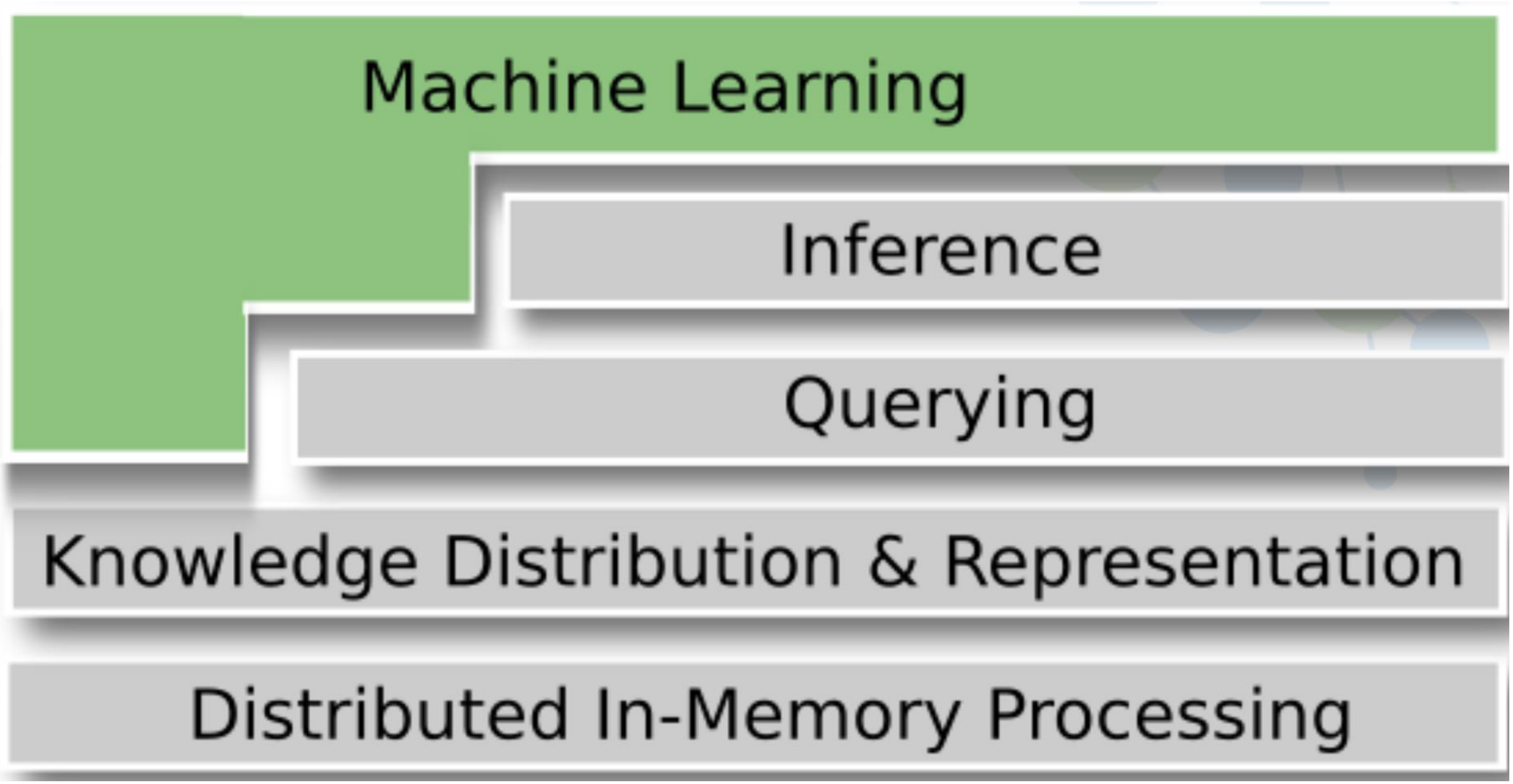
- The default approach for querying RDF data in SANSA is based on SPARQL-to-SQL. It uses a flexible triple-based partitioning strategy on top of RDF (such as predicate tables with sub partitioning by data types)

Listing 8. Sparklify example.

```
import org.apache.jena.riot.Lang
import net.sansa_stack.rdf.spark.io._
import net.sansa_stack.query.spark.query._

val input = "hdfs://namenode:8020/data/rdf.nt"
val lang = Lang.NTRIPLES

val triples = spark.rdf(lang)(input)
val sparqlQuery = """SELECT ?s ?p ?o
                            WHERE {?s ?p ?o }
                     LIMIT 10"""
val result = triples.sparql(sparqlQuery)
z.show(result)
```

SANSA Stack – ML Layer

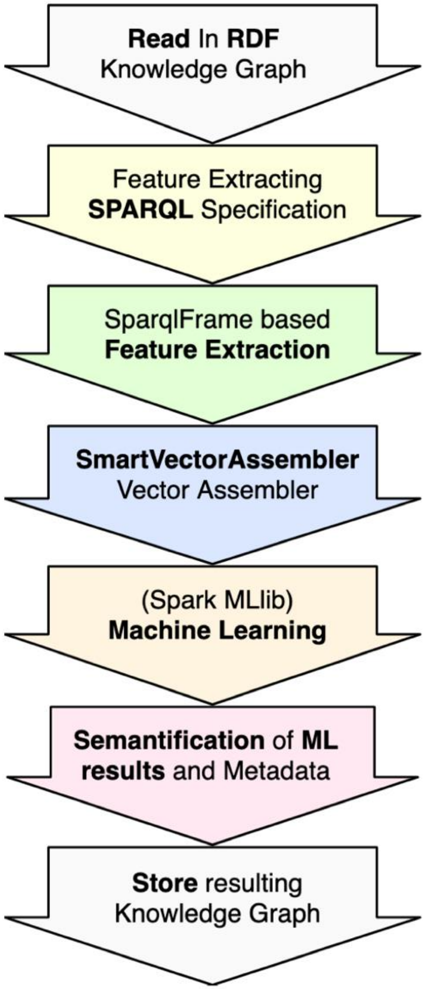
- Includes distributed ML models that works on and make use of RDF.



COLLEGE OF NANOTECHNOLOGY, SCIENCE, AND ENGINEERING
UNIVERSITY AT ALBANY STATE UNIVERSITY OF NEW YORK

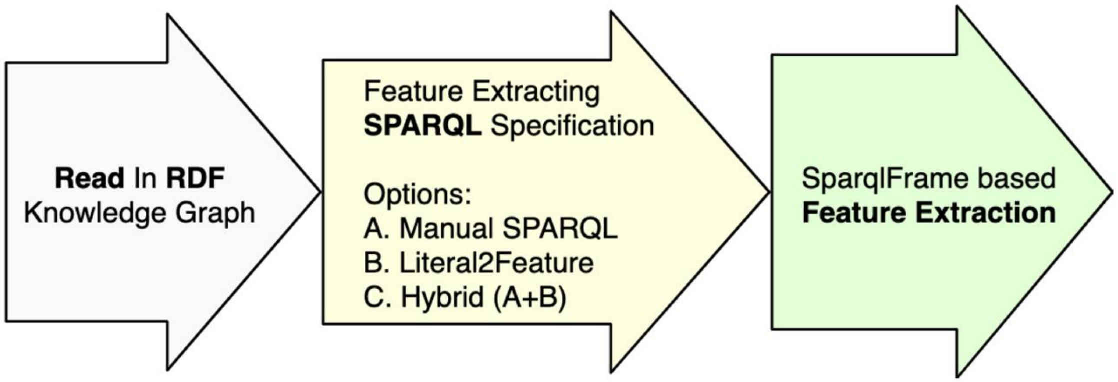
SANSA Stack – ML Layer (DistRDF2ML [10])

- Introduces software modules that transform large-scale RDF data into ML-ready fixed-length numeric feature vectors



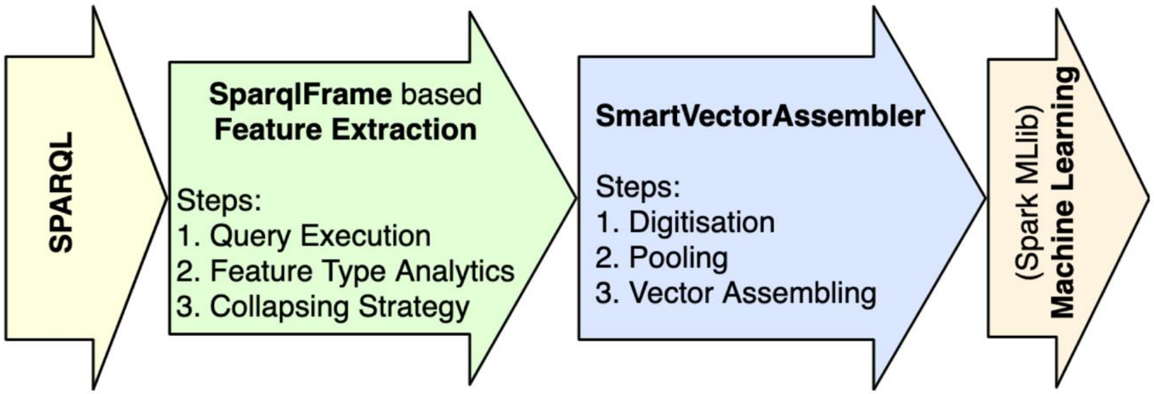
SANSA Stack – DistRDF2ML (SparqlFrame)

- **Manual SPARQL creation:** A knowledge graph expert manually crafts the SPARQL query to extract relevant features
- **Literal2Feature:** automatically generates a SPARQL query by deep traversing the RDF graph and extracting literals
- **Hybrid approach:** Literal2Feature generates a query, which is then manually refined, balancing automation with manual control to create a clean and focused SPARQL query



SANSA Stack – DistRDF2ML (SmartVectorAssembler)

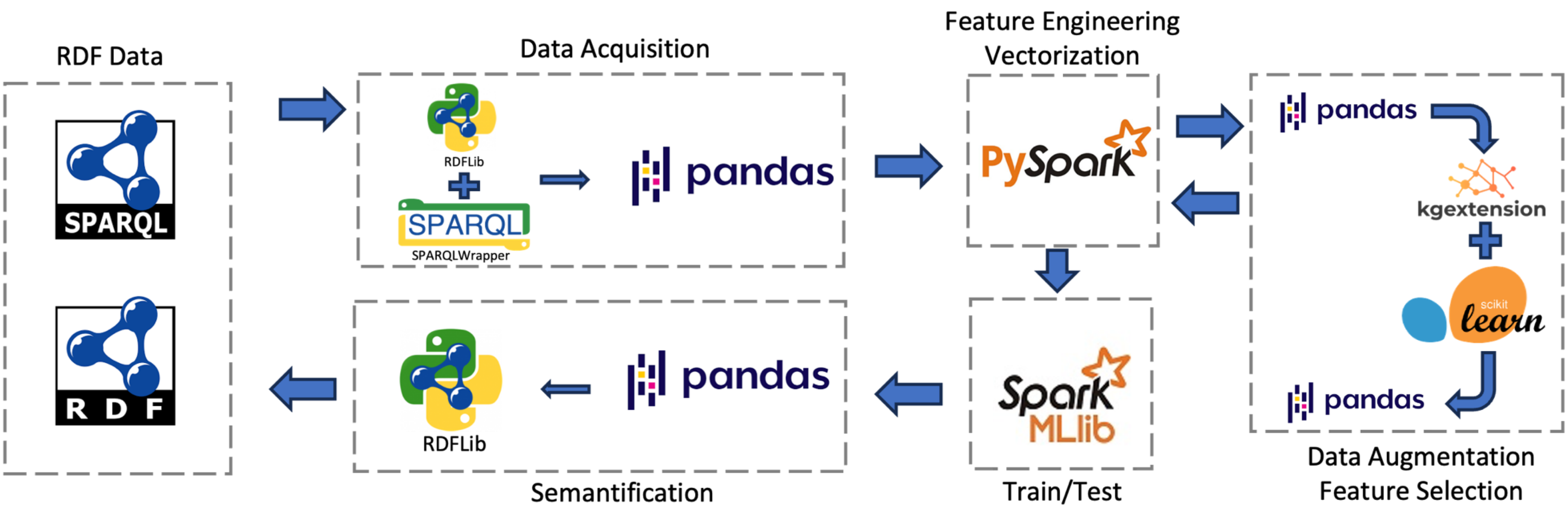
- **SparqlFrame**: Convert SPARQL query results into Spark DataFrames
- **SmartVectorAssembler**: Features are converted into numeric representations based on their type (e.g., Word2Vec for strings, indices for categorical lists, and datetime transformations for timestamps)



SANSA Notebook

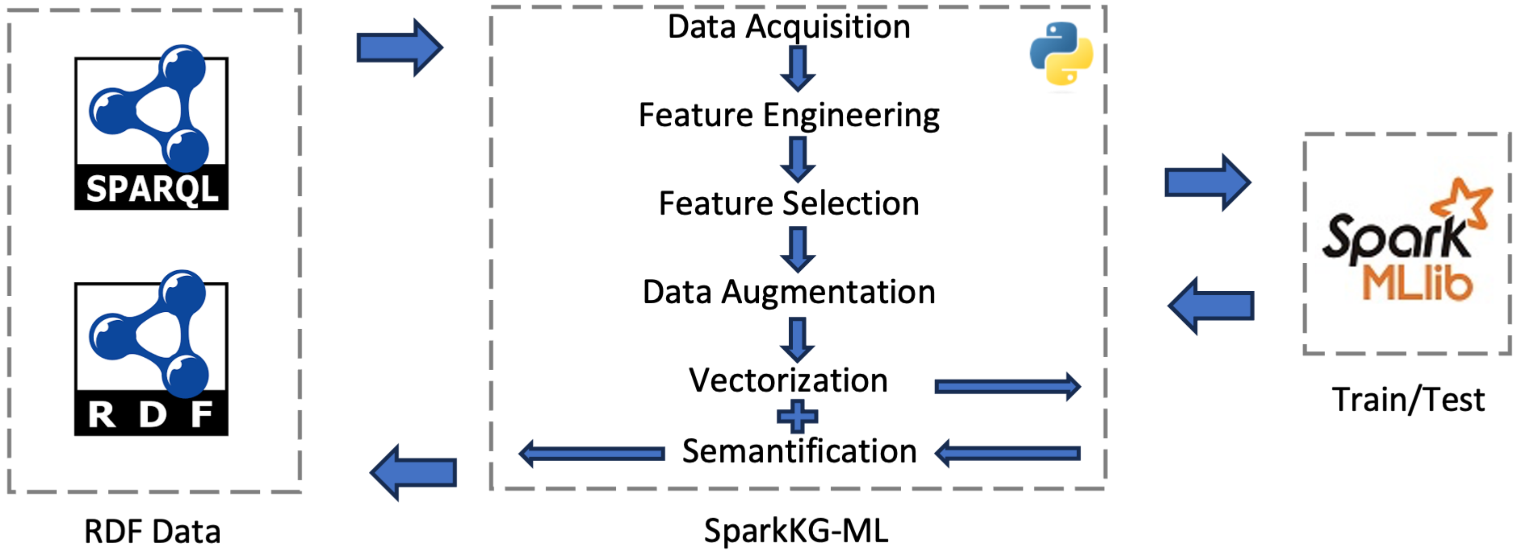
- Feel free to explore more!!
- You can find the hands-on notebook for SANSA in our Github repository

Typical Workflow



SparkKG-ML [11]*

- A Library to Facilitate end–to–end Large–scale Machine Learning over Knowledge Graphs in Python



*It will be presented on Thursday at 4pm in Main Tracks (7): Machine Learning for Graphs session

SparkKG-ML

Data Acquisition: Transform RDF data into the tabular format that Spark can process.

Feature Eng.: Gathers feature characteristics and a collapsed DataFrame is created.

Feature Selection: Focuses on identifying and retaining attributes while discarding redundant ones.

Data Augmentation: Enables augmenting a given KG with data from public KGs, allowing the extraction of additional features from these KGs.

Vectorization: Produces a ML-ready DataFrame by transforming all features according to their feature type into numeric representations.

Semantification: Transform ML results into RDF data.

SparkKG-ML: Data Acquisition

- Transform RDF data into the tabular format that Spark can process

a. Knowledge Graph

```
PREFIX schema: <https://schema.org/>
PREFIX recipeKG: <http://purl.org/recipekg/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX food: <http://purl.org/heals/food/>
SELECT ?recipe ?ingredientName ?fat
WHERE { ?recipe a schema:Recipe.
        ?recipe food:hasIngredient ?ingredient.
        ?ingredient recipeKG:ingredientName ?ingredientName.
        ?recipe recipeKG:hasNutritionalInformation ?a.
        ?a recipeKG:hasFatData ?b.
        ?b recipeKG:hasAmount ?fat. }
```

b. SPARQL Query

	recipe	ingredientName	fat
1	recipeKG:recipe/candied-christmas-cookies	"all purpose flour"	"5.8"^^xsd:float
2	recipeKG:recipe/candied-christmas-cookies	"baking soda"	"5.8"^^xsd:float
3	recipeKG:recipe/candied-christmas-cookies	"bourbon"	"5.8"^^xsd:float
4	recipeKG:recipe/candied-christmas-cookies	"brown sugar"	"5.8"^^xsd:float
5	recipeKG:recipe/candied-christmas-cookies	"butter"	"5.8"^^xsd:float
6	recipeKG:recipe/peanut-butter-tandy-bars	"egg"	"9.5"^^xsd:float
7	recipeKG:recipe/peanut-butter-tandy-bars	"butter"	"9.5"^^xsd:float
8	recipeKG:recipe/peanut-butter-tandy-bars	"chocolate"	"9.5"^^xsd:float
9	recipeKG:recipe/peanut-butter-tandy-bars	"baking powder"	"9.5"^^xsd:float
10	recipeKG:recipe/the-best-oatmeal-cookies	"cinnamon"	"7.6"^^xsd:float
	⋮	⋮	⋮

c. Query Result

SparkKG-ML: Data Acquisition

- Transform RDF data into the tabular format that Spark can process

```python
# Import the required module
from sparkkgml.data_acquisition import DataAcquisition

# Create an instance of DataAcquisition
dataAcquisitionObject=DataAcquisition()

# Specify the SPARQL endpoint and query
endpoint = "https://recipekg.arcc.albany.edu/RecipeKG"
query =""" ... """

# Retrieve the data as a Spark DataFrame
spark_df = dataAcquisitionObject.getDataFrame(endpoint=
    endpoint, query=query)
```

```
+----------------+------------------+-------+
| recipe         | ingredient       | fat   |
+----------------+------------------+-------+
| candied-chri...| flour            |  5.8  |
| candied-chri...| baking soda      |  5.8  |
| candied-chri...| bourbon          |  5.8  |
| candied-chri...| brown sugar      |  5.8  |
| candied-chri...| butter           |  5.8  |
| peanut-butte...| egg              |  9.5  |
| peanut-butte...| butter           |  9.5  |
| peanut-butte...| chocolate        |  9.5  |
| peanut-butte...| baking powder    |  9.5  |
| the-best-oat...| cinnamon         |  7.6  |
+----------------+------------------+-------+
```

SparkKG-ML: Feature Engineering

- Gathers feature characteristics and a collapsed DataFrame is created
  - **datatype**: The data type of the feature column.
  - **numberDistinctValues**: The number of distinct values in the feature column.
  - **isListOfEntries**: Flag indicating if the feature is a list of entries.
  - **isCategorical**: The ratio of distinct values and overall dataset size
  - **featureType**: Combine features based on whether they consist of a list or a single value, categorical or non-categorical, and data type.

```
# Import the required module
from sparkkgml.feature_engineering import FeatureEngineering
from sparkkgml.vectorization import Vectorization

# Create an instance of FeatureEngineering
featureEngineeringObject=FeatureEngineering()

# Call the getFeatures function
df2,features=featureEngineeringObject.getFeatures(spark_df)

# Create an instance of Vectorization
vectorizationObject=Vectorization()

# Call vectorize function, digitaze all the columns
digitized_df=vectorizationObject.vectorize(df2,features)
```

```
+----------------+----------------------+-------+
| recipe         | ingredients          | fat   |
+----------------+----------------------+-------+
| candied-chri...| [baking soda, egg, b...| 5.8 |
| peanut-butte...| [egg, butter, chocol...| 9.5 |
| best-oatmeal...| [cinnamon, egg, suga...| 7.6 |
| alfredo-blue...| [salt, egg, pasta, b...| 5.2 |
| millie-pasqu...| [lemon, flour, salt,...| 4.3 |
+----------------+----------------------+-------+
```

NANOTECHNOLOGY, SCIENCE, AND ENGINEERING
UNIVERSITY AT ALBANY STATE UNIVERSITY OF NEW YORK

SparkKG-ML: Vectorization

- Produces a ML–ready DataFrame by transforming all features according to their feature type into numeric representations
  - **Single Categorical String:** indexing or hashing
  - **List of Categorical Strings:** explodes the list and applies string indexing or hashing
  - **Single Non–Categorical String:** Word2Vec (optional stop word removal)
  - **List of Non–Categorical Strings:** the list elements are combined, tokenized, stop words are removed. Embeddings are then calculated using Word2Vec.
  - **Numeric Type :** (i.e., integer, long, float, double)
  - **Boolean Type:** cast to integers (0 or 1).

entity	features	label
candied-chri...	[0.01, 1.34, 6...	5
peanut-butte...	[0.03, 6.34, 7...	6
best-oatmeal...	[0.01, 8.34, 3...	1
alfredo-blue...	[0.05, 3.34, 2...	5
millie-pasqu...	[0.61, 9.34, 1...	4

SparkKG-ML Notebook

- Feel free to explore more on SparkKG-ML!!
- You can find the hands-on notebook for SparkKG-ML in our Github repository

SANSA vs SparkKG-ML

- Functionality
  - SANSA incorporates additional modules (e.g., inferencing)
  - SparkKG-ML specializes on ML pipelines*
- Ease of use
  - Installation process (just pip install vs step 1, 2, 3, …)
  - Programming language (Scala vs Python)
- End-to-End processing
  - SANSA in Scala
  - SparkKG-ML in Python
- Scalability:
  - SparkKG-ML is faster
- Community/Support
  - SANSA (substantial # of contributors, numerous forks)
  - SparkKG-ML is brand new, so help us make it thrive

Short break (5 min)

- Off to discussion and conclusion.
- Any questions??