Engaging Content
Engaging People

A World Leading SFI Research Centre

Science Foundation Ireland sfi For what's next

Querying Multiple Semantic Data Sources

Damien Graux, ADAPT Centre, Trinity College Dublin

December 14th 2020

HOST INSTITUTION
Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

HOST INSTITUTION
DCU Ollscoil Chathair Bhaile Átha Cliath Dublin City University

PARTNER INSTITUTIONS
TU Dublin
UCD Dublin
CIT Cork Institute of Technology
Institiúid Teicneolaíochta Chorcaí
AIT Connect & Discover
Maynooth University National University of Ireland Maynooth

FUNDED BY
Ireland's European Structural and Investment Funds Programmes 2014-2020
Co-funded by the Irish Government and the European Union
European Union
European Regional Development Fund

Who am I?

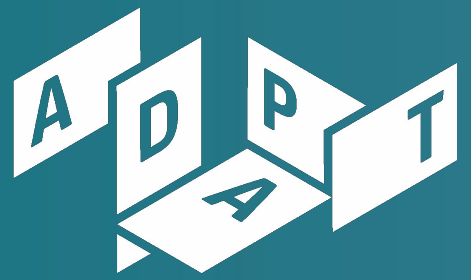
Damien Graux

Research Fellow at Trinity College Dublin

`https://dgraux.github.io/`

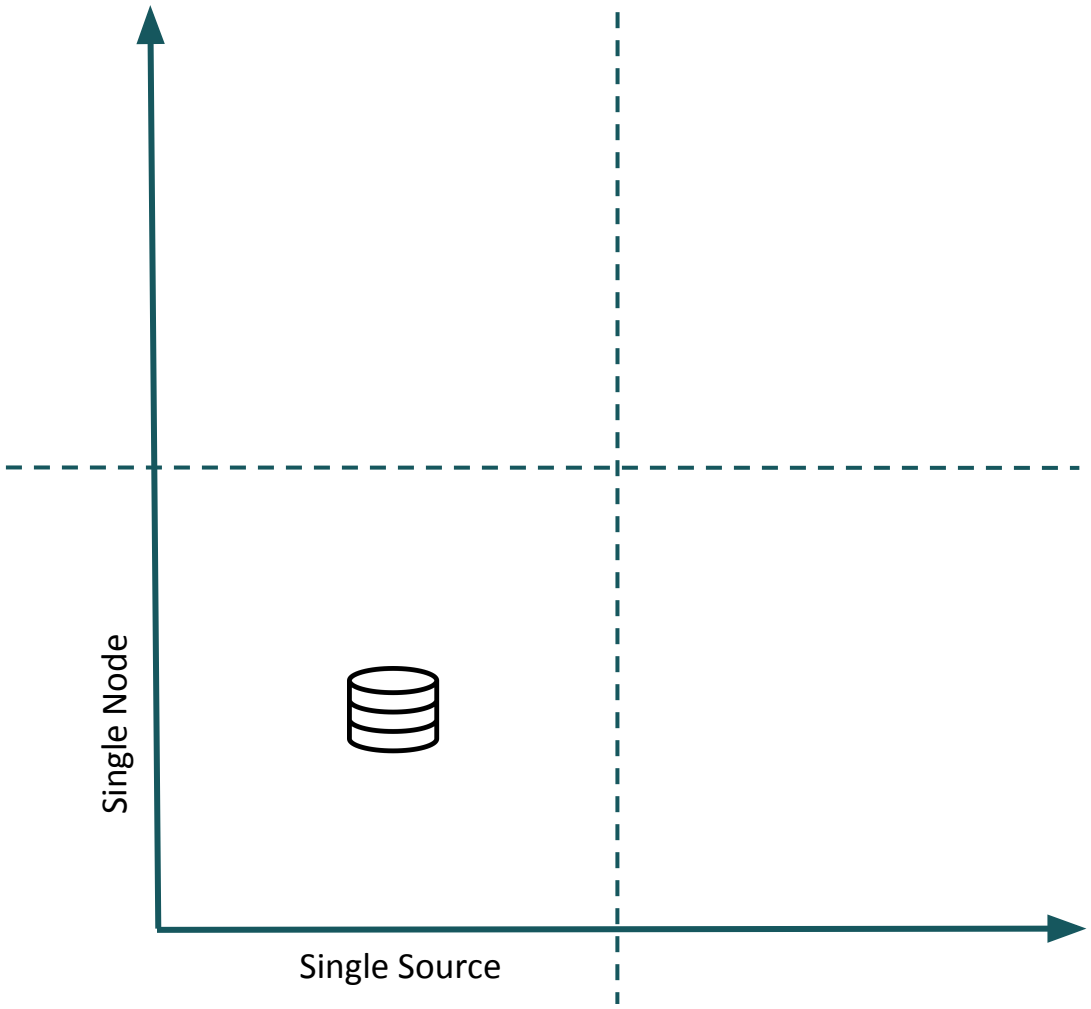
Focused on the Semantic Web domain especially on distributed query evaluation and on complex data pipelines.

Multiple Datasources…!



Engaging Content
Engaging People

So far, the **simplest** case

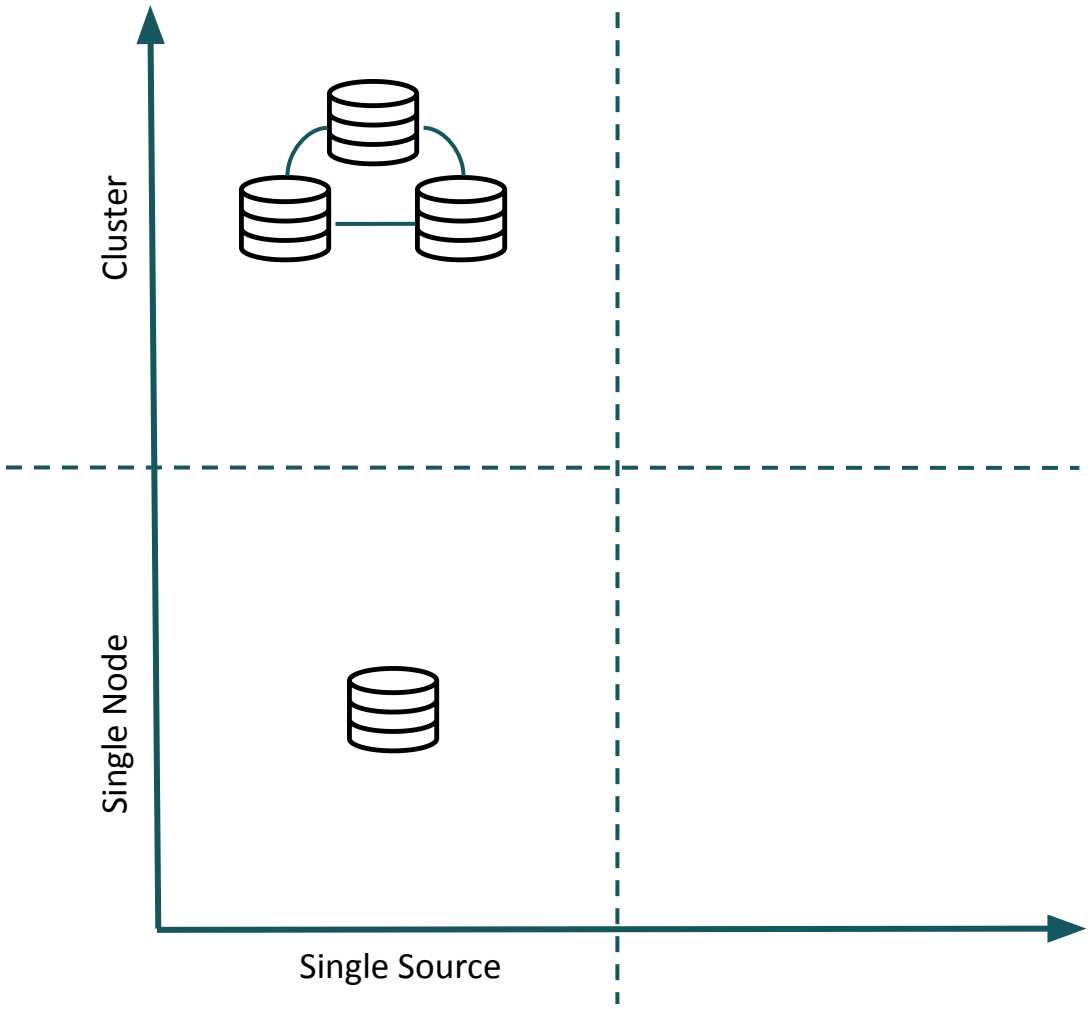


Datasource types

Various paradigms:

1. **One** graph stored on **one** machine

When datasets are **distributed** across nodes…

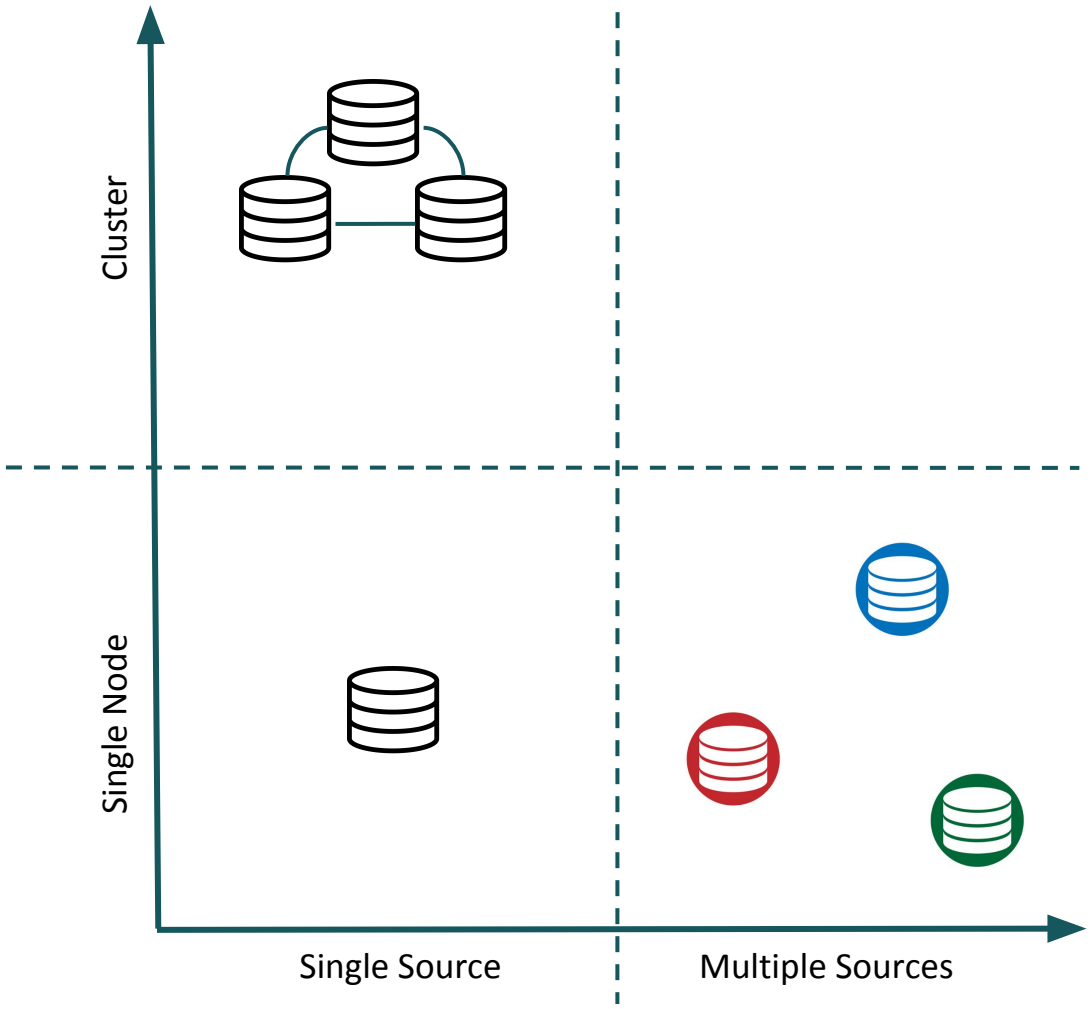


Datasource types

Various paradigms:

1. One graph stored on one machine
2. **One** graph distributed on a **cluster** of nodes

…and when datasets have **different origins**,

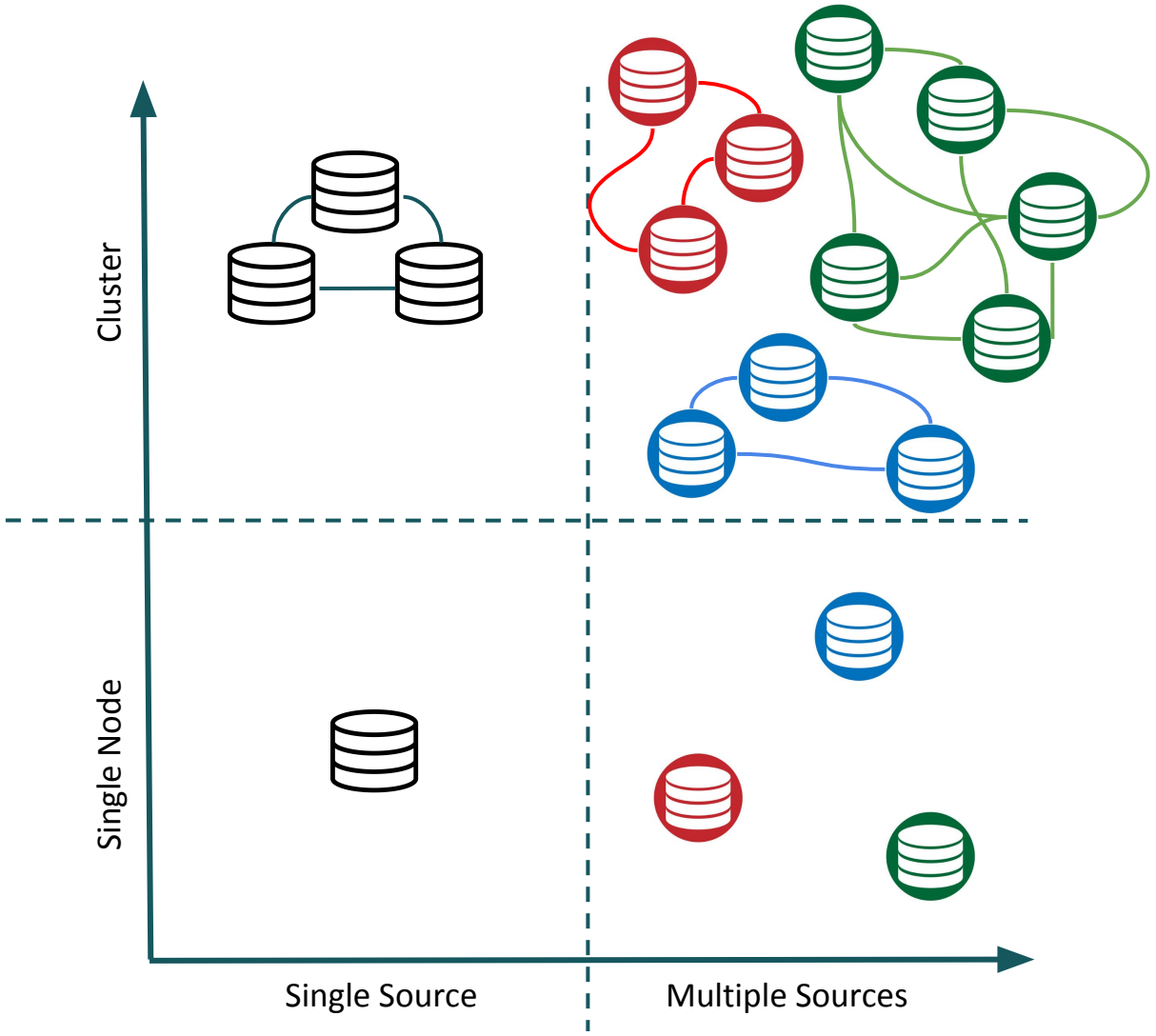


Datasource types

Various paradigms:

1. One graph stored on one machine
2. One graph distributed on a cluster of nodes
3. **Several graphs** available

…finally, distributed datasets have different origins

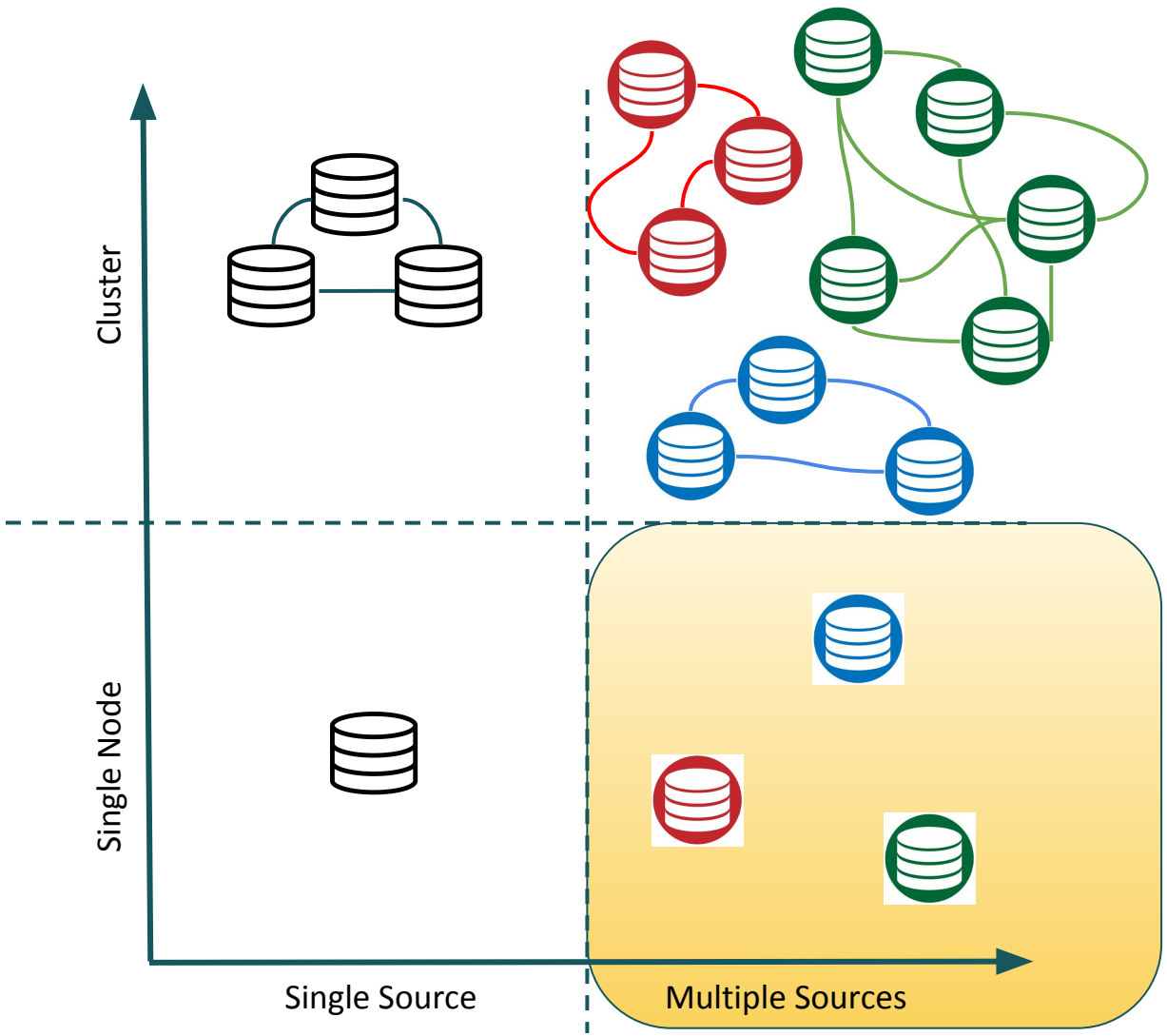


Datasource types

Various paradigms:

1. One graph stored on one machine
2. One graph distributed on a cluster of nodes
3. Several graphs available
4. **Several distributed** graphs

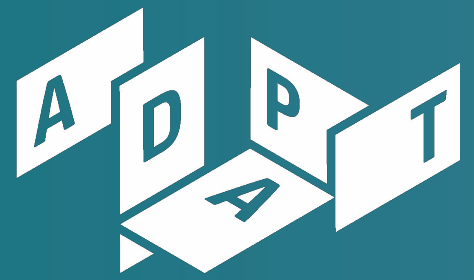
Today's focus ⇒ A **Federation** of Sources



Challenges

- How to access several sources at once?
- How to efficiently retrieve information?

SPARQL Federation



Engaging Content
Engaging People

An RDF graph, so far

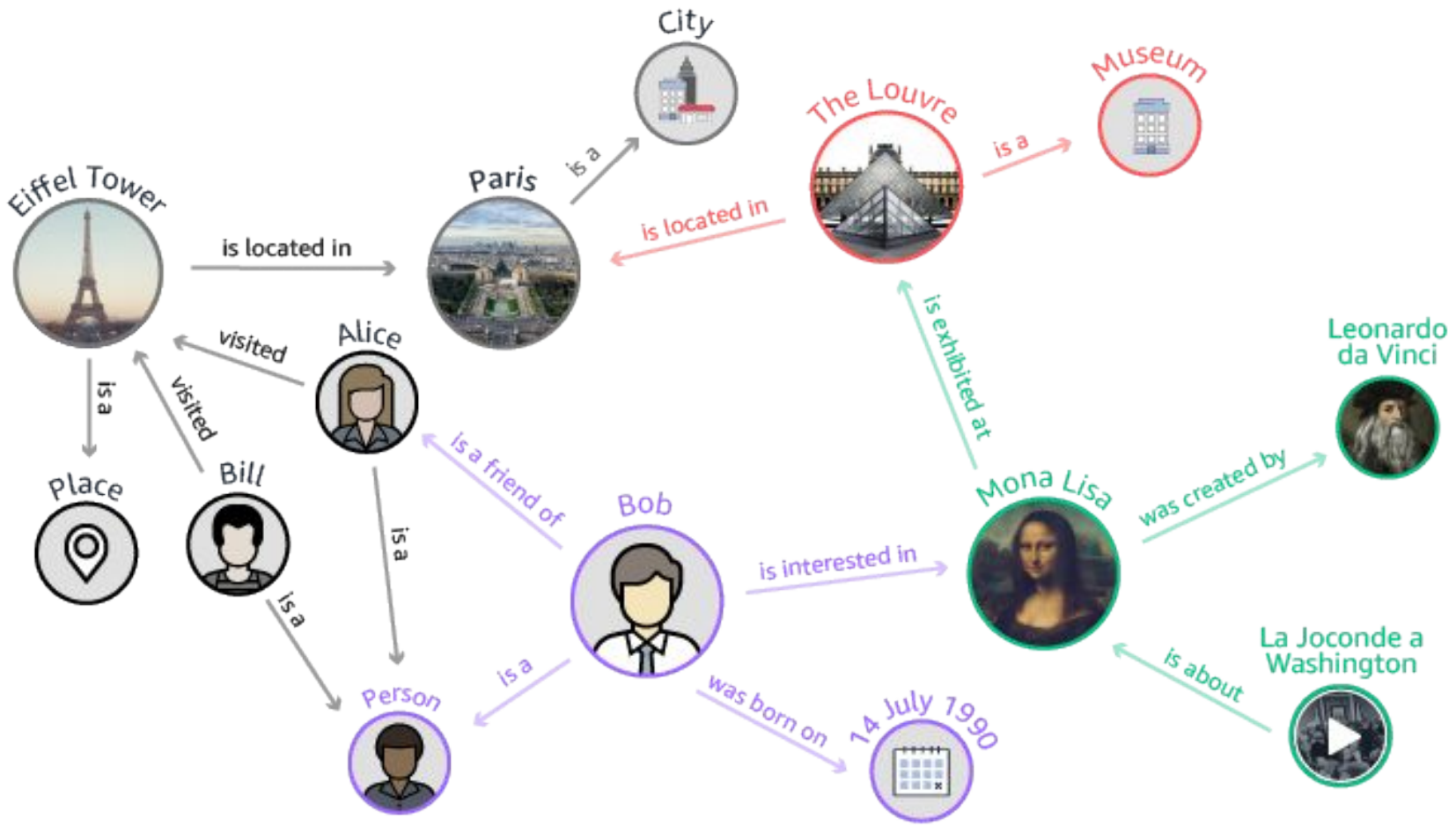


Image : https://aws.amazon.com/neptune/

RDF Graph

Structured as triples

- **subject** is the described resource
- **predicate** is the property applied to the resource
- **object** is either a literal or a resource

But, **practically**!

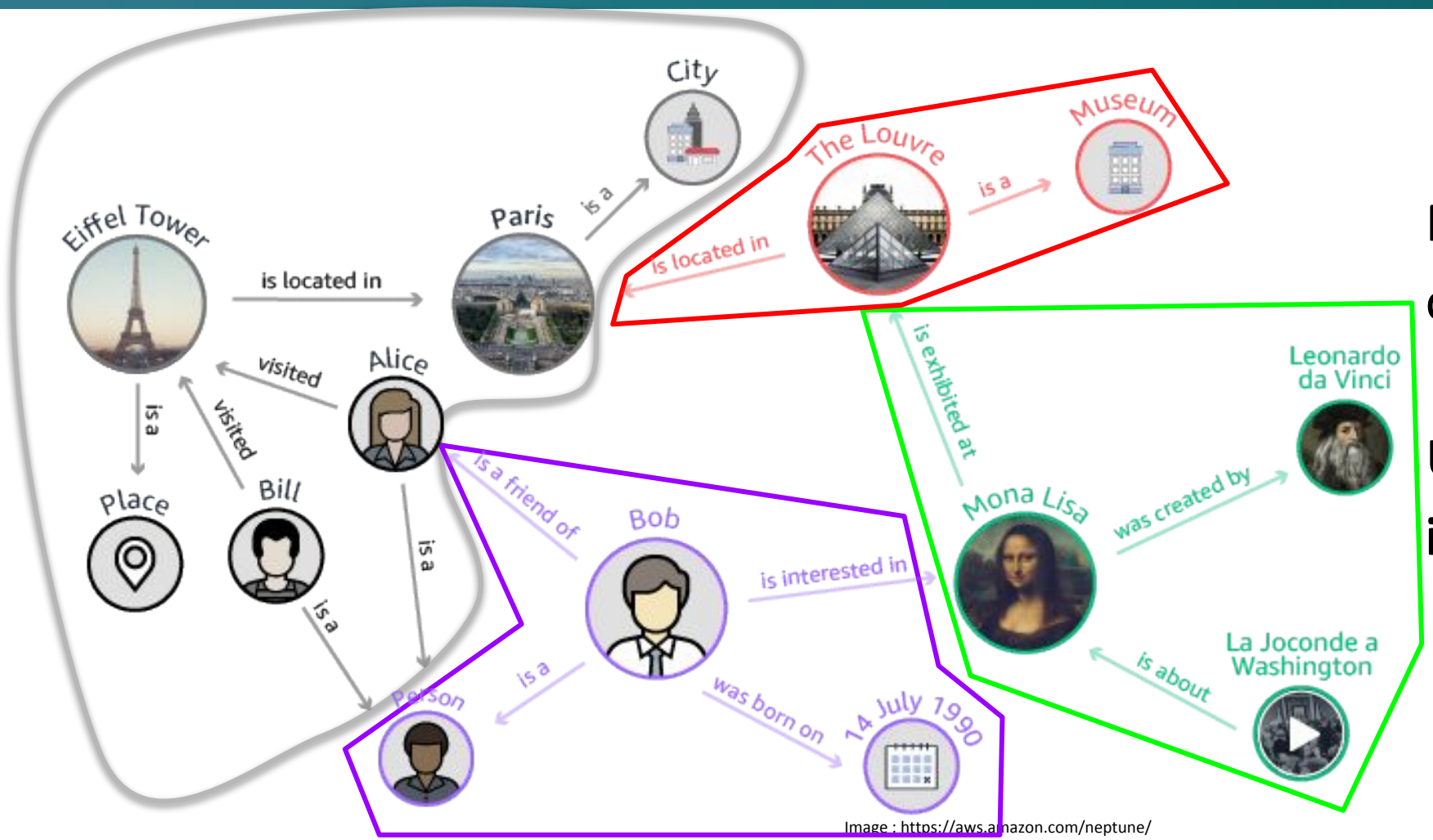


Image : https://aws.amazon.com/neptune/

RDF Graph

Made of **different** sources connected together.

Using the Semantic Web **interlinking** features.

Querying **naively** several RDF data sources

For a standalone and unique application, one could:

1. Detect where the datasets sit;
2. Query each SPARQL endpoint separately;
3. Collect the results and aggregate them.

Querying **naively** several RDF data sources

For a standalone and unique application, one could:

1. Detect where the datasets sit;
2. Query each SPARQL endpoint separately;
3. Collect the results and aggregate them.

Practically, it implies a lot of scripting and debugging, + prior knowledge on data sources!

The following steps have to be coded:

- Start a python script
- Write several SPARQL queries (you need to know the structure and ontology of each source)
- Run them on their endpoints
- Parse the results
- Join them and perform further computations
- Return the final result

Querying **naively** several RDF data sources

For a standalone and unique application, one could:

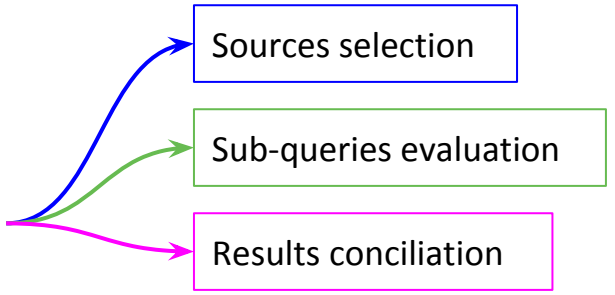
1. Detect where the datasets sit;
2. Query each SPARQL endpoint separately;
3. Collect the results and aggregate them.

Practically, it implies a lot of scripting and debugging, + prior knowledge on data sources!

Goal ⇒ How to automatise as much as possible?

The following steps have to be coded:

- Start a python script
- Write several SPARQL queries (you need to know the structure and ontology of each source)
- Run them on their endpoints
- Parse the results
- Join them and perform further computations
- Return the final result



Querying several data sources with the W3C standard

W3C Recommendation

SPARQL 1.1 Federated Query

W3C Recommendation 21 March 2013

This version:
http://www.w3.org/TR/2013/REC-sparql11-federated-query-20130321/
Latest version:
http://www.w3.org/TR/sparql11-federated-query/
Previous version:
http://www.w3.org/TR/2012/PR-sparql11-federated-query-20121108/
Editors:
Eric Prud'hommeaux, W3C <eric@w3.org>
Carlos Buil-Aranda, Ontology Engineering Group, UPM, Spain; currently at Universidad Pontificia Católica de Chile
Contributors:
Andy Seaborne, The Apache Software Foundation
Axel Polleres, Siemens AG <axel.polleres@siemens.com>
Lee Feigenbaum, Cambridge Semantics <lee@thefigtrees.net>
Gregory Todd Williams, Rensselaer Polytechnic Institute <greg@evilfunhouse.com>

Please refer to the **errata** for this document, which may include some normative corrections.

See also **translations**.

Abstract

RDF is a directed, labeled graph data format for representing information in the Web. SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. This specification defines the syntax and semantics of SPARQL 1.1 Federated Query extension for executing queries distributed over different SPARQL endpoints. The `SERVICE` keyword extends SPARQL 1.1 to support queries that merge data distributed across the Web.

Querying several data sources with the W3C standard

- The SPARQL federated feature is available in SPARQL 1.1 from

  <https://www.w3.org/TR/2013/REC-sparql11-federated-query-20130321/>
- Goals:
  - extension for executing queries distributed over different SPARQL endpoints
  - support of queries that merge data distributed across the Web
- Provides the dedicated `SERVICE` keyword:

```
SELECT ?var WHERE {
     conditions
     SERVICE <external/graph/address> {
          conditions
     }
}
```

- SERVICE instructs a query processor to invoke a portion of a SPARQL query against a remote SPARQL endpoint

Simple SERVICE in action

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
@prefix : <http://example.org/> .

:people15  foaf:name     "Alice" .
:people16  foaf:name     "Bob" .
:people17  foaf:name     "Charles" .
:people18  foaf:name     "Daisy" .
```

http://people.example.org/sparql

http://example.org/myfoaf.rdf

```
<http://example.org/myfoaf/I> <http://xmlns.com/foaf/0.1/knows>  <http://example.org/people15> .
```

Simple SERVICE in action

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
@prefix : <http://example.org/> .

:people15  foaf:name     "Alice" .
:people16  foaf:name     "Bob" .
:people17  foaf:name     "Charles" .
:people18  foaf:name     "Daisy" .
```

http://people.example.org/sparql

http://example.org/myfoaf.rdf

```
<http://example.org/myfoaf/I> <http://xmlns.com/foaf/0.1/knows>  <http://example.org/people15> .
```

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name
FROM <http://example.org/myfoaf.rdf>
WHERE {
  <http://example.org/myfoaf/I> foaf:knows ?person .
  SERVICE <http://people.example.org/sparql> {
    ?person foaf:name ?name . }
}
```

Simple SERVICE in action

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
@prefix : <http://example.org/> .

:people15  foaf:name     "Alice" .
:people16  foaf:name     "Bob" .
:people17  foaf:name     "Charles" .
:people18  foaf:name     "Daisy" .
```

http://people.example.org/sparql

http://example.org/myfoaf.rdf

```
<http://example.org/myfoaf/I> <http://xmlns.com/foaf/0.1/knows>  <http://example.org/people15> .
```

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name
FROM <http://example.org/myfoaf.rdf>
WHERE {
  <http://example.org/myfoaf/I> foaf:knows ?person .
  SERVICE <http://people.example.org/sparql> {
    ?person foaf:name ?name . }
}
```

name
"Alice"

Nested & complex SERVICE in action

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
@prefix : <http://example.org/> .

:people15  foaf:name     "Alice" .
:people16  foaf:name     "Bob" .
:people17  foaf:name     "Charles" .
:people17  foaf:interest :rdb2rdf .
```

http://people.example.org/sparql

http://people2.example.org/sparql

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
@prefix : <http://example.org/> .

:people15  foaf:knows    :people18 .
:people18  foaf:name     "Mike" .
:people17  foaf:knows    :people19 .
:people19  foaf:name     "Daisy" .
```

Nested & complex SERVICE in action

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
@prefix : <http://example.org/> .

:people15  foaf:name     "Alice" .
:people16  foaf:name     "Bob" .
:people17  foaf:name     "Charles" .
:people17  foaf:interest :rdb2rdf .
```

http://people.example.org/sparql

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
@prefix : <http://example.org/> .

:people15  foaf:knows    :people18 .
:people18  foaf:name     "Mike" .
:people17  foaf:knows    :people19 .
:people19  foaf:name     "Daisy" .
```

http://people2.example.org/sparql

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?person ?interest ?known WHERE {
  SERVICE <http://people.example.org/sparql> {
    ?person foaf:name ?name .
    OPTIONAL {
      ?person foaf:interest ?interest .
      SERVICE <http://people2.example.org/sparql> {
        ?person foaf:knows ?known . } }
  }
}
```

Nested & complex SERVICE in action

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
@prefix : <http://example.org/> .

:people15  foaf:name     "Alice" .
:people16  foaf:name     "Bob" .
:people17  foaf:name     "Charles" .
:people17  foaf:interest :rdb2rdf .
```

http://people.example.org/sparql

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
@prefix : <http://example.org/> .

:people15  foaf:knows    :people18 .
:people18  foaf:name     "Mike" .
:people17  foaf:knows    :people19 .
:people19  foaf:name     "Daisy" .
```

http://people2.example.org/sparql

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name ?interest ?known WHERE {
  SERVICE <http://people.example.org/sparql> {
    ?person foaf:name ?name .
    OPTIONAL {
      ?person foaf:interest ?interest .
      SERVICE <http://people2.example.org/sparql> {
        ?person foaf:knows ?known . } }
  }
}
```

name	interest	known
"Alice"		
"Bob"		
"Charles"	:rdb2rdf	:people19

Advantages of SPARQL query federation

- Allows to collect information from different sources
- Datasets remain where they are and can be further updated
- Easy to deploy using the SPARQL standard if we know
  - where the endpoints are
  - the content/structure of the endpoints
- Feature supported by most of the open-source tools *e.g.* Apache Jena

Querying several data sources with a dedicated tool

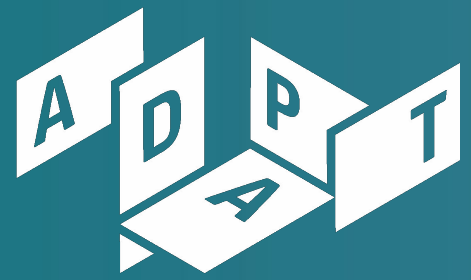
To ease the tedious process previously described, tools have been developed to enable users to execute queries over a federation of SPARQL endpoints.

To name a few:

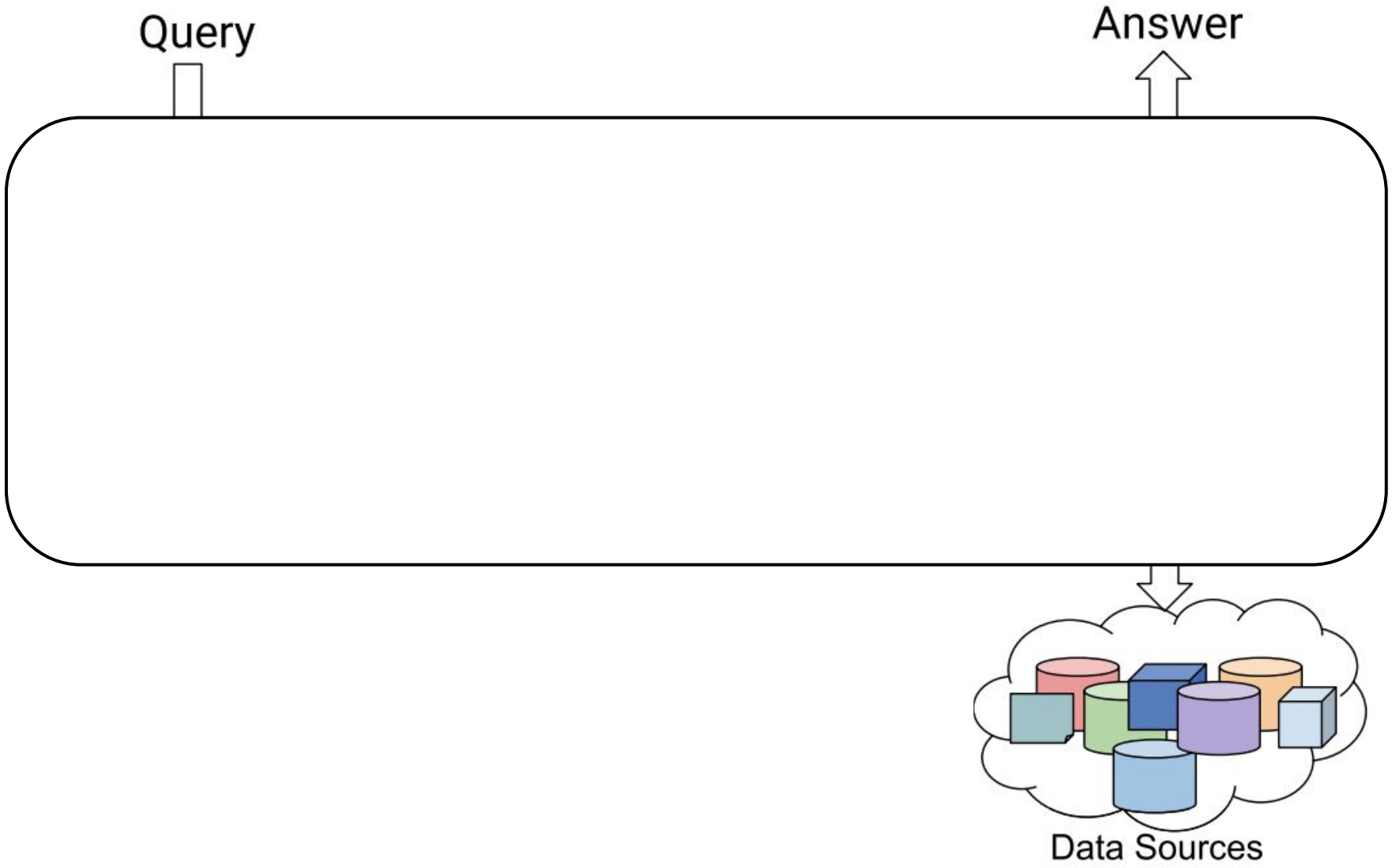
- **FedX** implements adaptive techniques to identify relevant sources to evaluate a query. It is able to contact SPARQL endpoints on the fly to choose the subqueries of the original query
- **ANAPSID** makes use of metadata about RDF vocabularies
- **MULDER** resorts to description of the RDF datasets based on the classes and relations of the dataset vocabularies

Integrating RDF sources

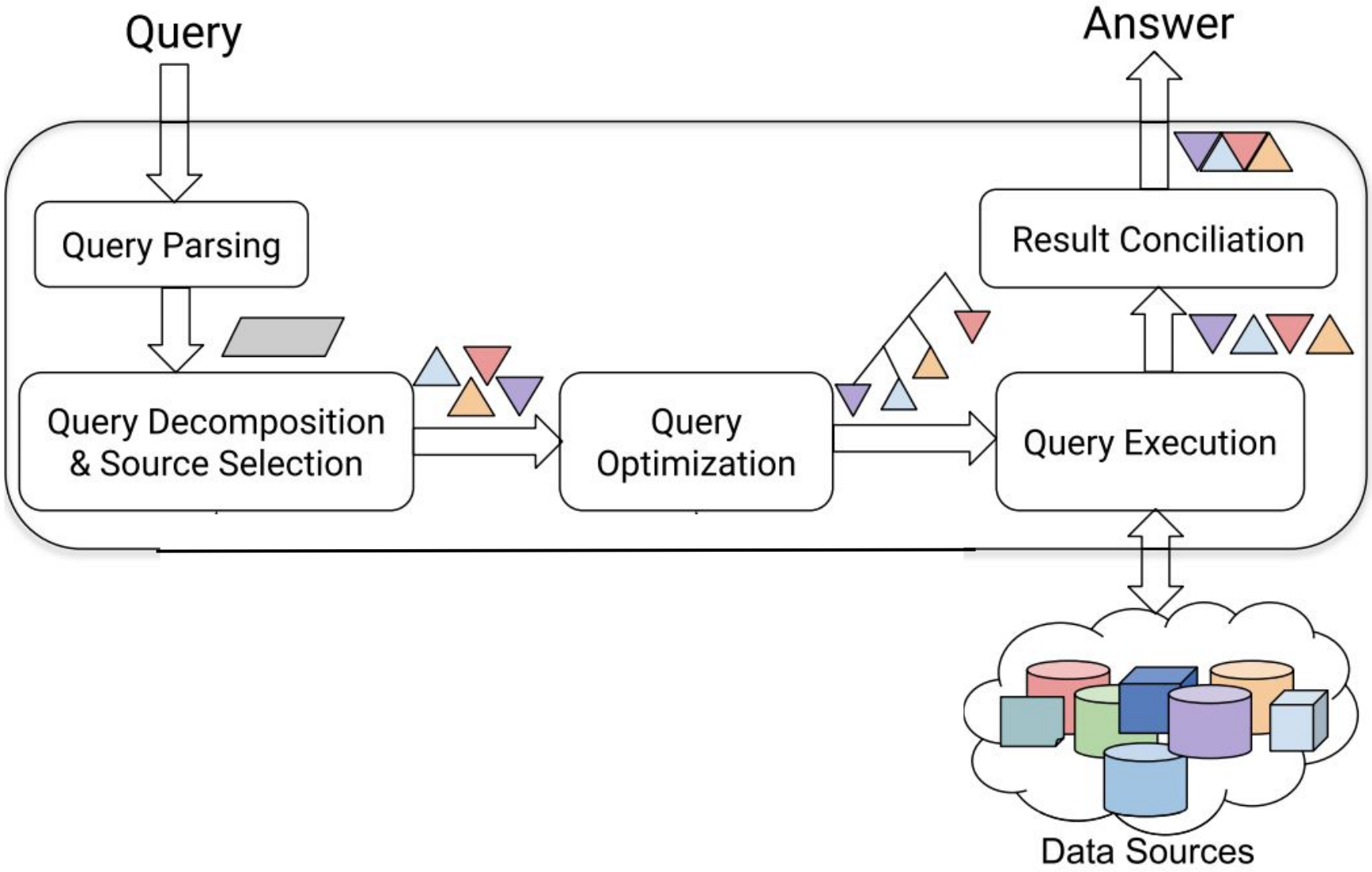
Zoom on a federated query processing



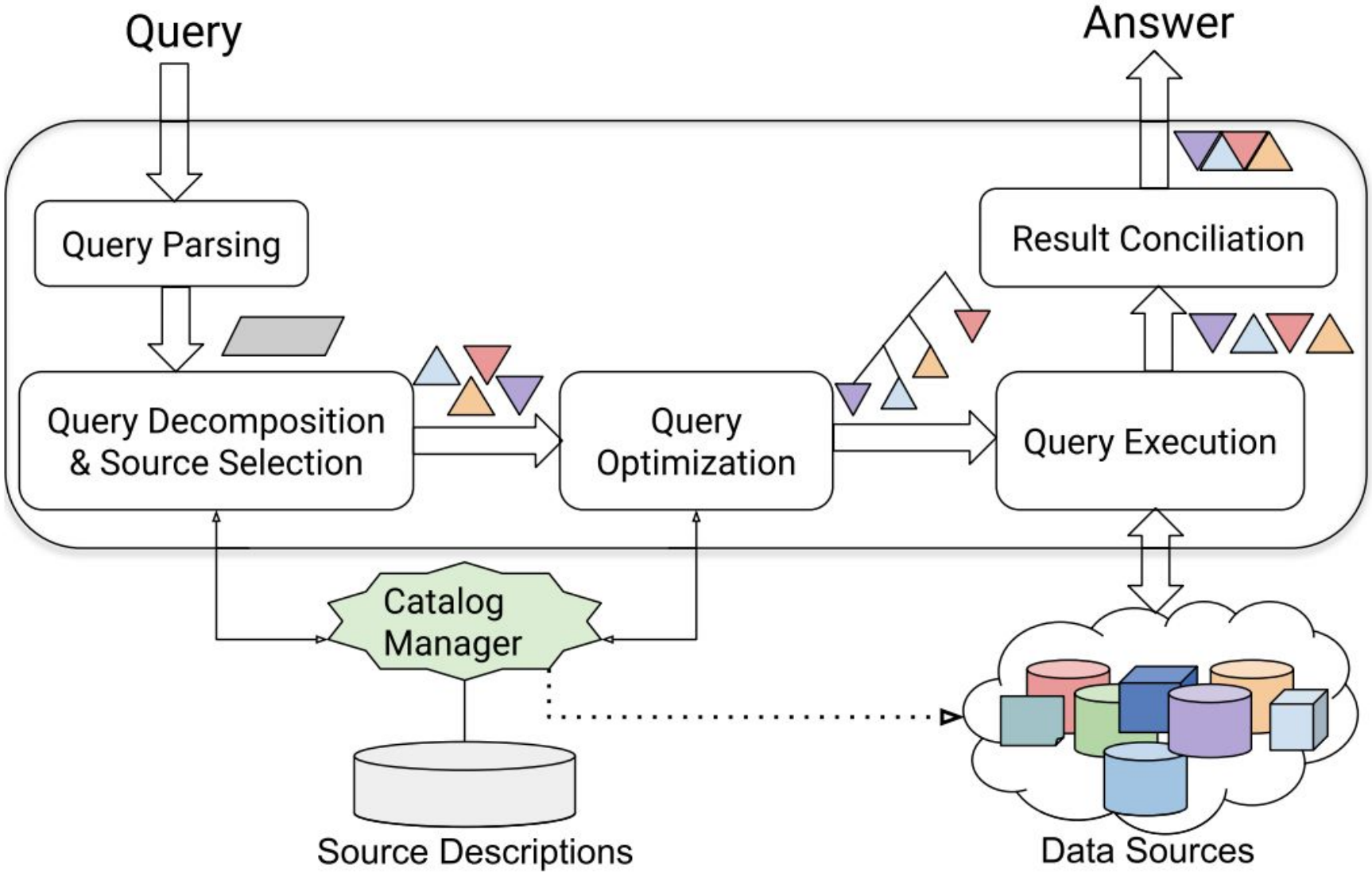
Federated query processing basic components



Federated query processing basic components



Federated query processing basic components



Data Source Description

- **Goal**: describing the data available in data sources and managing catalogs about data sources that are participating in the federation.

- The descriptions can encode information about availability, data types, access method, privacy or access policies.
- Schema mappings also represent privacy and access control restrictions as well as statistics on the available data in each data source.

⇒ Could be pre-computed or obtained on-the-fly with ASK queries

Query Decomposition & Source Selection

- **Goal**: decomposes the federated query into subqueries associated with data sources in the federation that are selected for executing the subqueries.

- The number of data sources considered for selection are bounded by the data source description given to the federated query processing engine. Each sub-query may be associated to zero or more data sources.

Query Planning and Optimisation

- **Goal**: generate an execution plan that represent the steps on how the query is executed and which algorithms (operators) are used.

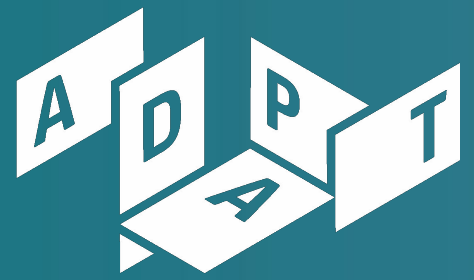
- This task produces query execution plans (a tree-based plan).
- In a federated setting, the number of intermediate results and the communication costs impact the performance of query execution.
- Optimization techniques include making decisions on selection of the join methods, ordering, and adapting to the condition of the sources.

Query Execution

- **Goal**: sub-queries executed in each data source are then optimized using the local schema (and index) of the data source and executed.

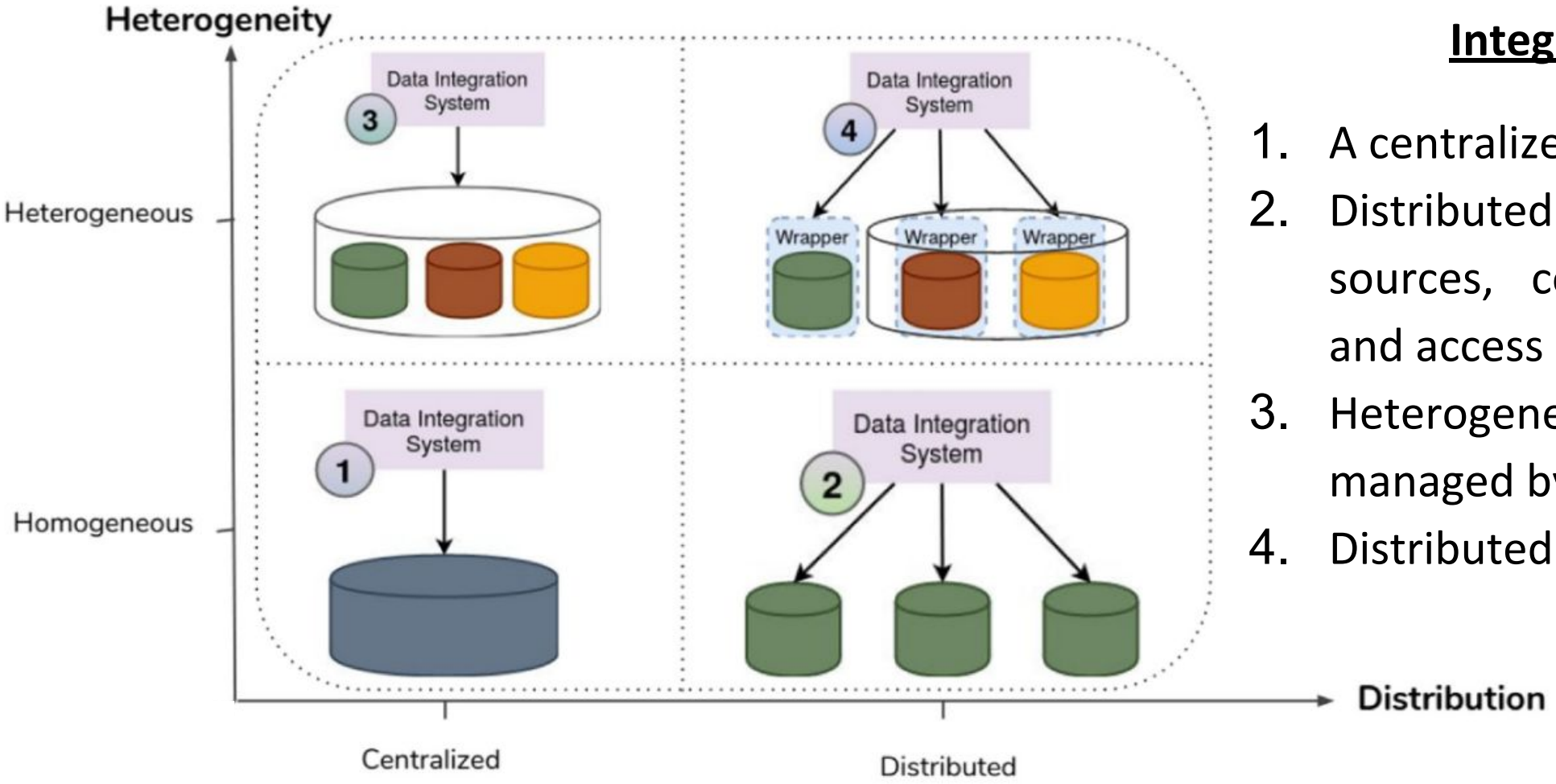
- Five different join methods are used in federated query engines:
  - Nested-loop join: executes the inner sub-query for every single binding of the intermediate results
  - Bound-join: executes inner sub-query for the set of bindings
  - Hash-join method: each sub-query is executed in parallel and the join is performed locally using a single hash table at the query engine
  - Symmetric (hash) join: non-blocking hash-based join that pipelines parallel execution of the operands
  - Multiple (hash) join: uses multiple hash tables to join more than two sub-queries running at the same time

And what if we have heterogeneous data?!



Engaging Content
Engaging People

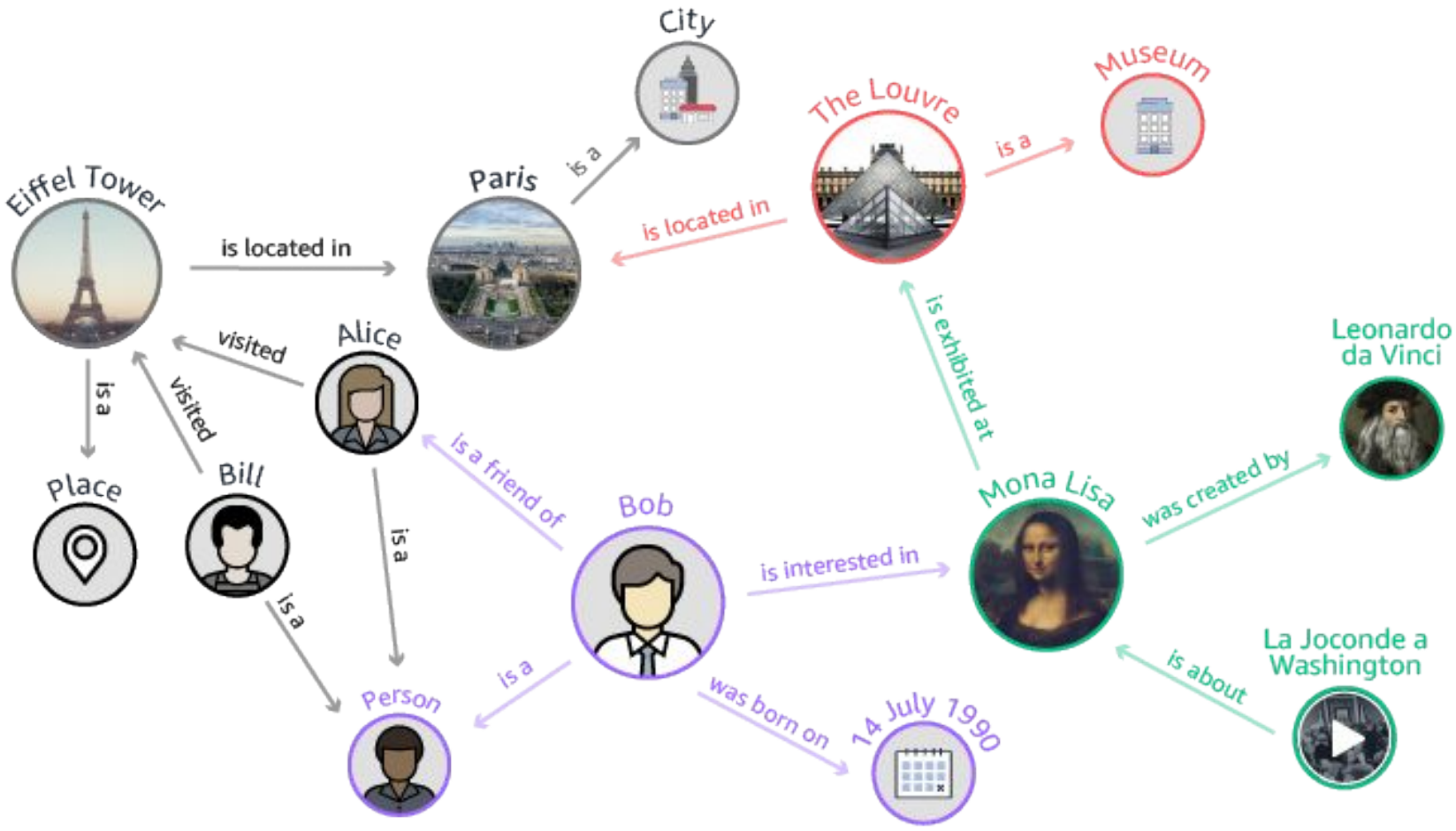
Data Integration — A classification



Integration types

1. A centralized data source
2. Distributed homogeneous data sources, common data model and access methods
3. Heterogeneous data sources managed by a centralized system
4. Distributed heterogeneous data

Let's increase the complexity



Data landscape

Variety of data sources:

- Heterogeneous structures
- Several domains
- Different size

GitHub

PubMed.gov

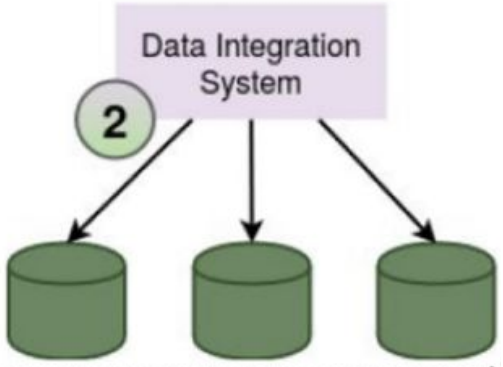
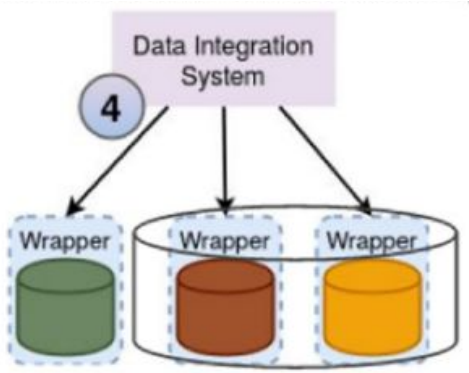
OpenStreetMap
The Free Wiki World Map

Copernicus
Europe's eyes on Earth

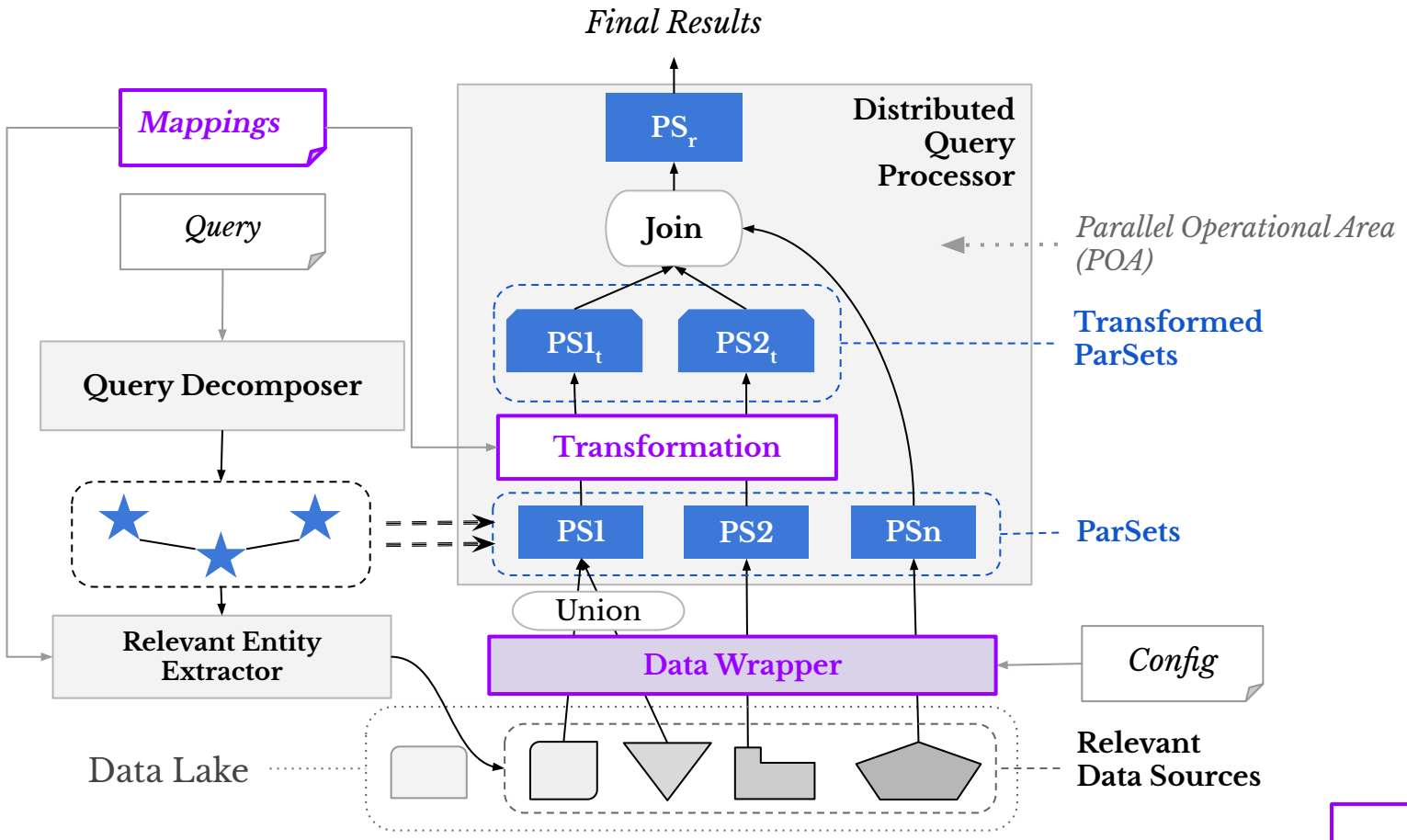
Wikipédia
L'encyclopédie libre

DBpedia

From *SPARQL Federation* To *Semantic Data Lake*

SPARQL Federation	Semantic Data Lake
	
Characteristics: • Several RDF data sources • SPARQL as common query language	Characteristics: • Several independent heterogeneous data sources • Multiple query languages

Heterogeneous datasets ⟶ Semantic Data lake



Strategy: Ontology-based data access

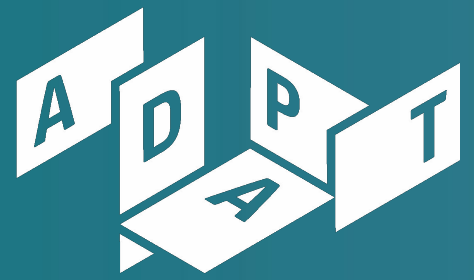
Semantic Standards for Virtual Data Integration

- Ontology terms to create a cross-data sources schema
  - A general schema abstracting from the underlying data models
  - High-level view of the data, to be queried uniformly
  - A Schema for the schema-less Data Lake
- Mappings to associate data elements with ontology elements
- Query the data in a uniform manner using SPARQL

Directly query original data (no prior transformation needed)
Allow scalable cross-source query execution
Enable query-time transformation to enable joinability
Use existing engine connectors (wrappers)

Similar in structure to the SPARQL Federation!
(Mappings, Wrappers & Transformations added to connect sources)

Let’s wrap up…!



Engaging Content
Engaging People

Conclusion

In a federated context (*i.e.* multiple data sources) there exists several methods to query at once several distinct sources.

The user can benefit from:

- the SPARQL Federated standard to query various endpoints <http://www.w3.org/TR/sparql11-federated-query/>;
- a set of tools to have fully integrated systems *i.e.* able to select relevant sources in addition to distributed sub-queries.

Further Reading

- Acosta, M., Hartig, O., Sequeda, J.F.: **Federated RDF query processing.**
  In: Encyclopedia of Big Data Technologies. Springer, Cham (2019).
  https://doi.org/10.1007/978-3-319-77525-8

- Saleem, M., Khan, Y., Hasnain, A., Ermilov, I., Ngomo, A.N.:
  A fine-grained evaluation of SPARQL endpoint federation systems.
  Semant. Web 7(5), 493–518 (2016). http://dx.doi.org/10.3233/SW-150186

- Valentina Janev, Damien Graux, Hajira Jabeen, and Emanuel Sallinger:
  Knowledge graphs and big data processing.
  Springer (2020): 209. https://doi.org/10.1007/978-3-030-53199-7

SPARQL Federation — Grand Challenges & Future Work

1. Definition of **formal models** to describe not only properties of data sources, but also represent causality relations and bias
2. **Adaptive query processing** techniques to adjust query processing schedules
3. Machine learning models able to **predict the cost of integrating** sources
4. **Hybrid approaches** combining computational methods with human knowledge
5. Query processing able to **interoperate during query execution**
6. Methods capable of **tracing data consumed** from the selected sources
7. **Explainable** federated systems able to justify all the decisions made

The Takeaway 😊

Standard solution for querying multiple SPARQL endpoints
⟶ The SERVICE keyword <http://www.w3.org/TR/sparql11-federated-query/>

Integrating multiple data sources:

- Homogeneous (RDF) ⟶ Tools such FedX, ANAPSID or MULDER
- Heterogeneous ⟶ OBDA strategies to manage a Data Lake

Further questions: <{grauxd,osullivan}@tcd.ie>

ADAPT Engaging Content Engaging People

FUNDED BY

European Union
European Regional Development Fund

Science Foundation Ireland sfi

Damien Graux
https://dgraux.github.io/
Trinity College Dublin, Ireland