Foundations of Semantic Knowledge Graphs

Prof. Dr. Stefan Linus Zander

A Framework for Encoding Semantic Knowledge Graphs – The Resource Description Framework (aka RDF)

Preamble: What makes a data graph a knowledge graph ?

We have learned about the different types of data graphs and the elements that constitute a data graph.

In order to transform a data graph into a knowledge graph, we need additional features:

- Identity
- Schema knowledge
- Contextual knowledge
- Semantics

...and we need a knowledge representation framework in which these features can be encoded

⇝ We therefore discuss **RDF** and the **RDF Schema data-modelling vocabulary** that allow for the creation of knowledge graphs

RDF – The Lingua Franca for Data Integration

- RDF is **simple**
- RDF enables a **data-model-agnostic integration** ~> it **encodes and combines all kinds of data models** (relational, taxonomic, graphs, object-oriented, hierarchical, schema-driven, etc...)
- RDF supports **distributed data** and **schemas**
- RDF allows to **seamlessly evolve** simple semantic representations (e.g. vocabularies) into more complex and expressive ones (e.g. ontologies)
- Small **representational units** (URIs / IRIs / triples) facilitate mixing and mashing
- RDF can be viewed from **many perspectives**: facts, graphs, ER, logical axioms, objects, frames etc.
- RDF **integrates well with other formalisms** such as HTML (RDFa), XML (RDF/XML), JSON (JSON-LD), CSV, ...
- Linking and referencing between different knowlegde bases, systems, and platforms facilitates the creation of **sustainable data ecosystems** (e.g. Web of Data, DBpedia, etc.)
- RDF can be used to create **meta data annotation frameworks** for Web resources

Source: Adapted and complemented from https://www.slideshare.net/soeren1611/knowledge-graph-introduction

Outline

Part I: RDF

- Concepts and Elements
- How to identify things
- Reification
- Serialisation Formats

RDF Schema

- Motivation
- Classes and Properties
- Examples and Pitfalls
- RDF Schema Inferencing

Part II: Deductive Knowledge

- Ontologies
- Interpretations and Models
- Ontology Language Features
- Reasoning using DLs and Rules

Part III: Querying RDF Graphs

- Introduction to SPARQL
- Concepts and Elements
- SPARQL Examples
- SPARQL Query Endpoints

- **KG Applications** (if time)
  - Wikidata
  - DBpedia
  - Linked Open Data

In order to use RDF for encoding KGs, the following questions need to be answered

- How to represent vertices and edges in RDF KGs ?
- How to represent data and data values ?
- How can we express n-ary relationships ?
- How to provide identity for elements in the graph ?
- How to identify "things" ?
- How to distinguish "things" from representations about those "things" ?
- How to encode / serialize RDF graphs ?
- How to implement and integrate semantics ?
- How to share RDF graphs among applications and systems ?

The Resource Description Framework (RDF)

RDF is a W3C standard for exchanging graphs

- First proposed in 1999
- Updated in 2004 (RDF 1.0) and in 2014 (RDF 1.1)
- Originally built for Web data exchange
- Meanwhile used in many graph database applications
- Supported by many other W3C standards (RDFa, SPARQL, OWL, SHACL, . . .)

In this course: **focus on graph representation features of RDF 1.1**

W3C creates open standards: patent-free & freely accessible

- Gentle RDF 1.1 introduction: https://www.w3.org/TR/rdf11-primer/
- Specification of graph model: https://www.w3.org/TR/rdf11-concepts/
- Specific file formats are defined in other documents, linked from those

W3C = World Wide Web Consortium

What does RDF stand for ?

Ressource ⇝ can be or refer to **everything** that is **uniquely identifiable** by an URI/IRI and must be **referencable**

Description ⇝ representation of **properties** and **relationships** among resources in form of **directed graphs**

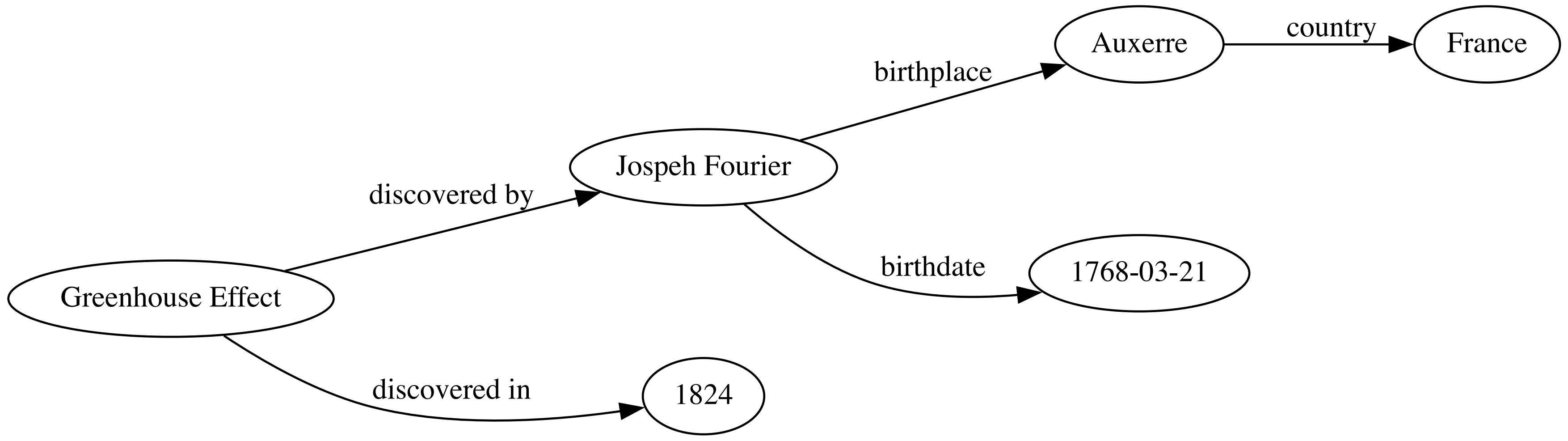
Framework ⇝ conflation of **Web technologies**, **protocols**, and **standards** (URI, HTTP, XML, JSON) and **formal logics** (semantics)

How to represent data in RDF

RDF allows for specifying graphs that are:

- directed (edges have a source and a target)
- edge-labelled (edges have one label)
- a restricted form of multi-graphs (multiple edges can exist between same vertices, but only if they have different labels)

Example



A example RDF graph

How data are represented in RDF

- Facts in RDF are expressed as **statements** in the form of `<subject> <predicate> <object>`-triples
- Statement (ie RDF-triples)

Subject	i.e.	Resource	$\rightarrow$	URI/IRI or Blank node
Predicate	i.e.	Property	$\rightarrow$	URI/IRI
Object	i.e.	Object or Value	$\rightarrow$	URI/IRI or Blank node or (Typed) Literal

- All RDF statements follow the same **logical schema** and are represented as a **list of triples**
- An RDF document can be **represented graphically** in different forms
    - Common representation form is a Node-Edge-Node graph
- Due to the **uniqueness** of node and edge **identifiers**, an RDF graph can be **reconstructed** from the list of triples

Elements of RDF graphs

- **URIs/IRIs**
  - enable the unique identification of resources
- **Literals**
  - describe data values that do not have a specific existence; can be typed or untyped (=plain)
- **Blank Nodes**
  - enable statements about the existence of individuals and their properties without naming them explicitly

How to identify things...

Information resources versus non-information resources ☆

In the Web, we distinguish between Information Resources and Non-Information Resources.

These concepts allow us to distinguish between the actual "thing" (ie designatum) and representations about it (ie designators).

A "thing" can be anything – a physical object, a city, a person, an animal, a feeling, an event etc.

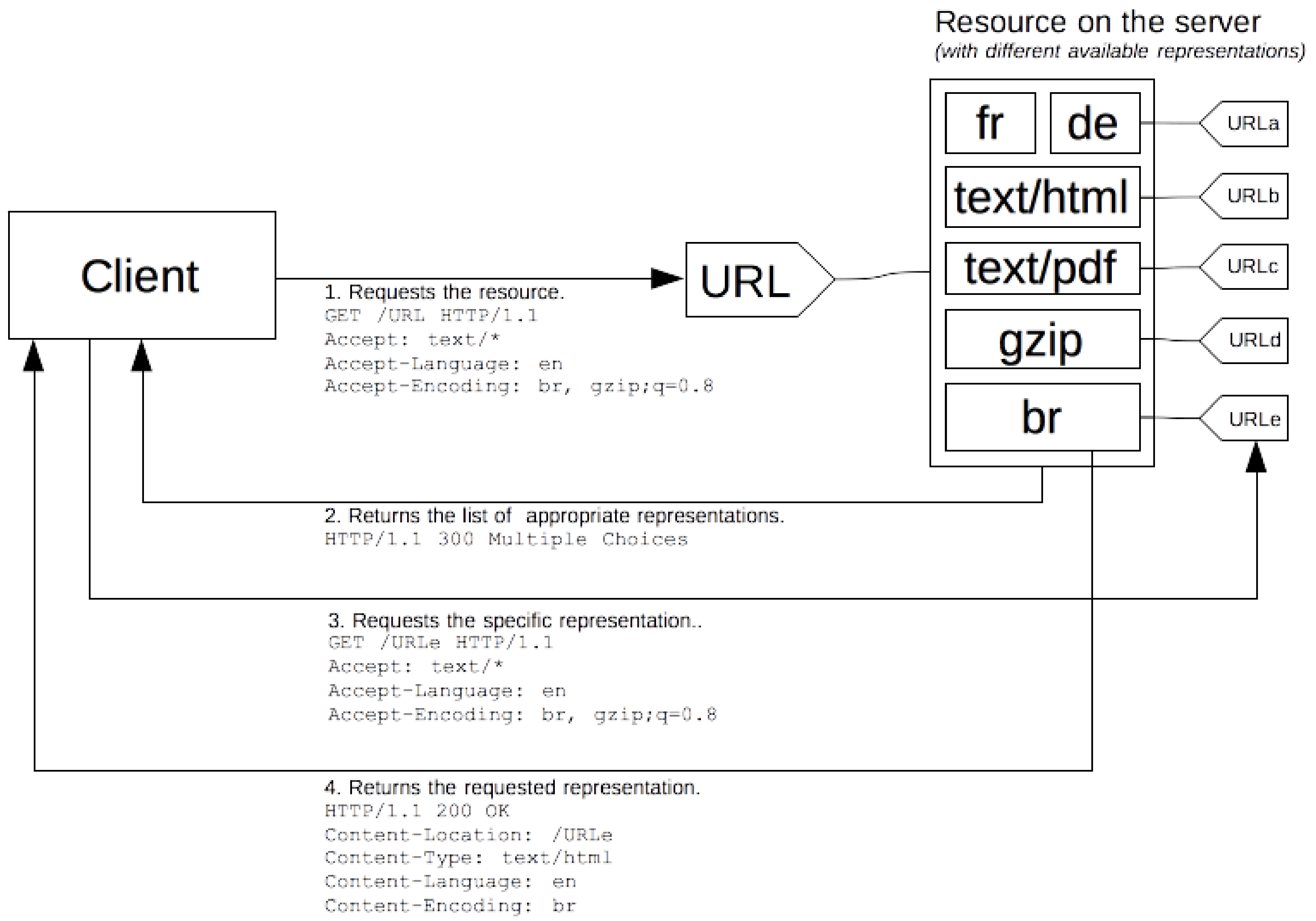
URIs/IRIs are used to identify things.

URLs allow to retrieve the content of information resources.
$\rightsquigarrow$ information resources are dereferencable; non-information resources are not

Non-information resources and information resources are "linked" together via content negotiation.

By using globally defined IRIs, the Web can be transformed into a linked data space ($\longrightarrow$ see Linked Data Cloud)

Content Negotiation



1. Requests the resource.

```
GET /URL HTTP/1.1
Accept: text/*
Accept-Language: en
Accept-Encoding: br, gzip;q=0.8
```

2. Returns the list of appropriate representations.

```
HTTP/1.1 300 Multiple Choices
```

3. Requests the specific representation..

```
GET /URLe HTTP/1.1
Accept: text/*
Accept-Language: en
Accept-Encoding: br, gzip;q=0.8
```

4. Returns the requested representation.

```
HTTP/1.1 200 OK
Content-Location: /URLe
Content-Type: text/html
Content-Language: en
Content-Encoding: br
```

Source:
https://www.w3.org/blog/2006/02/content-negotiation/

Identifiers in RDF: How should we refer to vertices?

Definition A Uniform Resource Identifier (URI) is a sequence (string) of a subset of ASCII characters as defined in RFC 3986 (link). Every (absolute) URI consists of a string that defines a scheme, followed by a colon (:) and another sequence of characters specifying an authority, path, query, and fragment, where all parts other than the path are optional.

A International Resource Identifier (IRI) is a generalised form of URI that allows for an expanded range of Unicode glyphs in part of its syntax.

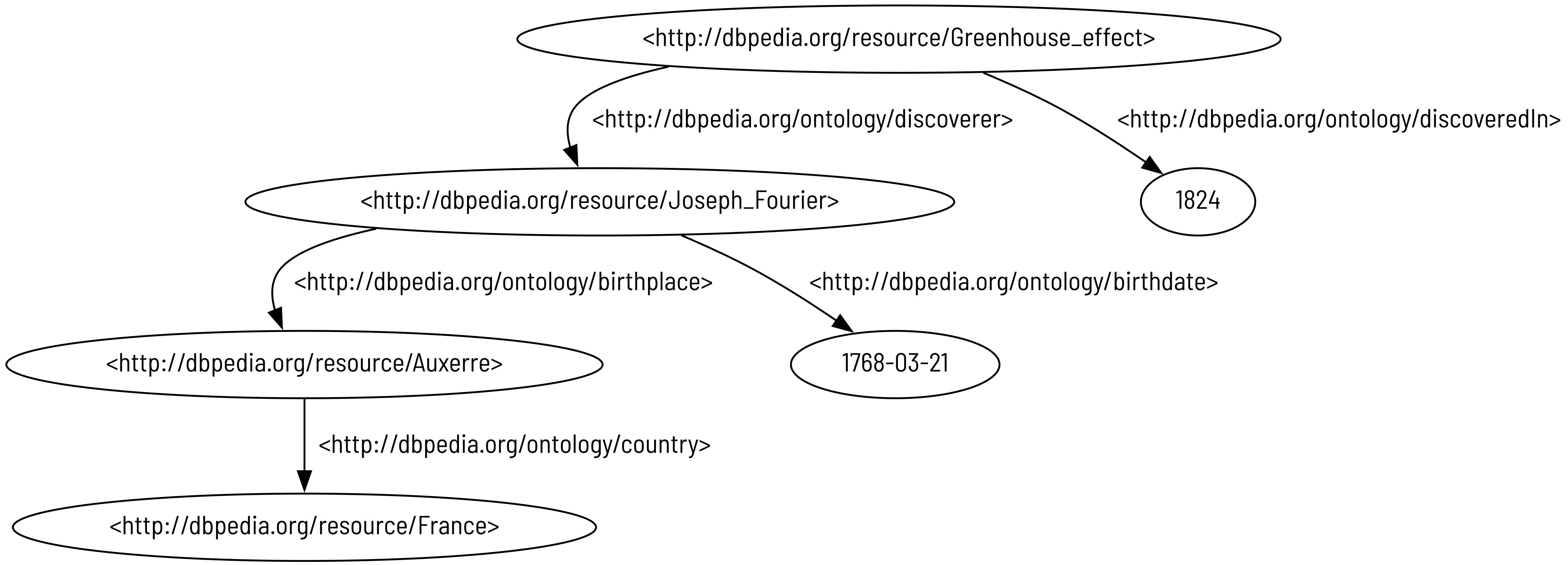
Source: Adapted from Krötzsch

Example

```
URI = scheme ":" ["//" authority] path ["?" query] ["#" fragment]
```

```
      |-------------------- Schema-spezifischer Teil ----------------------|
      |                                                                    |
https://maxmuster:geheim@www.example.com:8080/index.html?p1=A&p2=B#ressource
\___/   \_______/ \____/ \_____________/ \__/\_________/ \_______/ \_______/
  |         |       |           |         |       |          |         |
Schema¹ Benutzer Kennwort      Host      Port    Pfad      Query    Fragment
```

RDF uses IRIs in two ways: to define resources that appear as vertices, and as edge labels

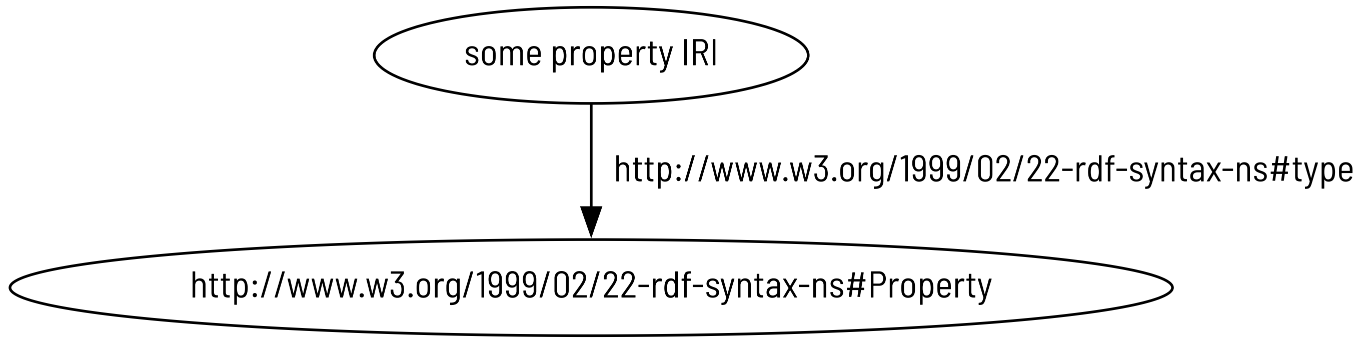

<http://dbpedia.org/resource/Greenhouse_effect>
<http://dbpedia.org/ontology/discoverer>
<http://dbpedia.org/ontology/discoveredIn>
<http://dbpedia.org/resource/Joseph_Fourier>
1824
<http://dbpedia.org/ontology/birthplace>
<http://dbpedia.org/ontology/birthdate>
<http://dbpedia.org/resource/Auxerre>
1768-03-21
<http://dbpedia.org/ontology/country>
<http://dbpedia.org/resource/France>

RDF Properties

RDF uses IRIs in **predicate** positions

- Resources represented by predicates are called properties
- We can make statements about properties by using their IRIs as subjects in triples

Example: It is common to assign labels to properties, and many applications display these labels to show triples that use these properties as their predicate.


some property IRI
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.w3.org/1999/02/22-rdf-syntax-ns#Property

We can declare a resource as a property using special RDF vocabulary:

Much further information about a property can be specified using properties of RDF and other standard vocabularies (esp. OWL)

Which IRIs to use in an RDF Graph?

Where do the **IRIs** that we use in graphs come from?

- They can be newly created for an application ⇝ avoid confusion with resources in other graphs 💣
- They can be IRIs that are already in common use ⇝ support information integration and re-use across graphs 👍

Guidelines for creating new IRIs:

1. Check if you could re-use an existing IRI ⇝ avoid duplication if feasible
2. Use http(s) IRIs ⇝ useful protocols, registries, resolution mechanisms
3. Create new IRIs based on domains that you own ⇝ clear ownership; no danger of clashing with other people's IRIs
4. Don't use URLs of existing web pages, unless you want to store data about pages ⇝ avoid confusion between pages and more abstract resources
5. Make your IRIs return some useful content via http(s) ⇝ helps others to get information about your resources

Source: Lecture slide from Prof. Dr. Markus Krötzsch

Excursus: Why IRIs ?

IRIs/URIs may seem a bit complicated

- They look a bit technical and complex
- They are hard to display or draw in a graph
- The guidelines just given may seem quite demanding to newcomers
- They can not be dereferenced just like URLs

However, it’s not that hard

- RDF can work with any form of IRI (most tools would probably accept any Latin-letter string with a colon inside!)
- The guidelines help sharing graphs across applications – a strength of RDF
- Internet domain name registration is a very simple way to define ownership in a global data space
- IRIs should not be shown to users (we will introduce human-readable labels soon)

Source: Lecture slide from Prof. Dr. Markus Krötzsch

How to represent data values in RDF ?

How to represent data values in RDF ?

IRIs should not be used to represent data values

- IRIs can represent almost anything
- but data values (numbers, strings, times, . . .) should not be represented by IRIs!

Why not use IRIs here too?

1. Data values are the same everywhere $\rightsquigarrow$ no use in application-specific IRIs
2. Many RDF-based applications need a built-in understanding of data values (e.g., for sorting content)
3. Data values are usually more "interpreted" than IRIs.
   - Example: Using a hypothetical scheme "integer", the IRIs `integer:42` and `integer:+42` would be different, but intuitively they should represent the same number.

RDF Datatypes

Definition A datatype in RDF is specified by the following components:

- The value space is the set of possible values of this type.
- The lexical space is a set of (Unicode) strings that can be used to denote values of this type.
- the lexical-to-value mapping is a function that maps each string from the lexical space to an element of the value space.

Source: Definition taken from Krötzsch, 2021

Datatypes for RDF must be identified by IRIs (known to software that supports them).

Example:
The W3C standard XML Schema defines the datatype **integer**, identified by the IRI http://www.w3.org/2001/XMLSchema#integer. It has the **value space** of all integer numbers (of arbitrarily large absolute value), the **lexical space** of finite-length strings of decimal digits (`0-9`) with an optional leading sign (`-` or `+`), and the expected **lexical-to-value mapping**.

RDF supports different forms of data type literals

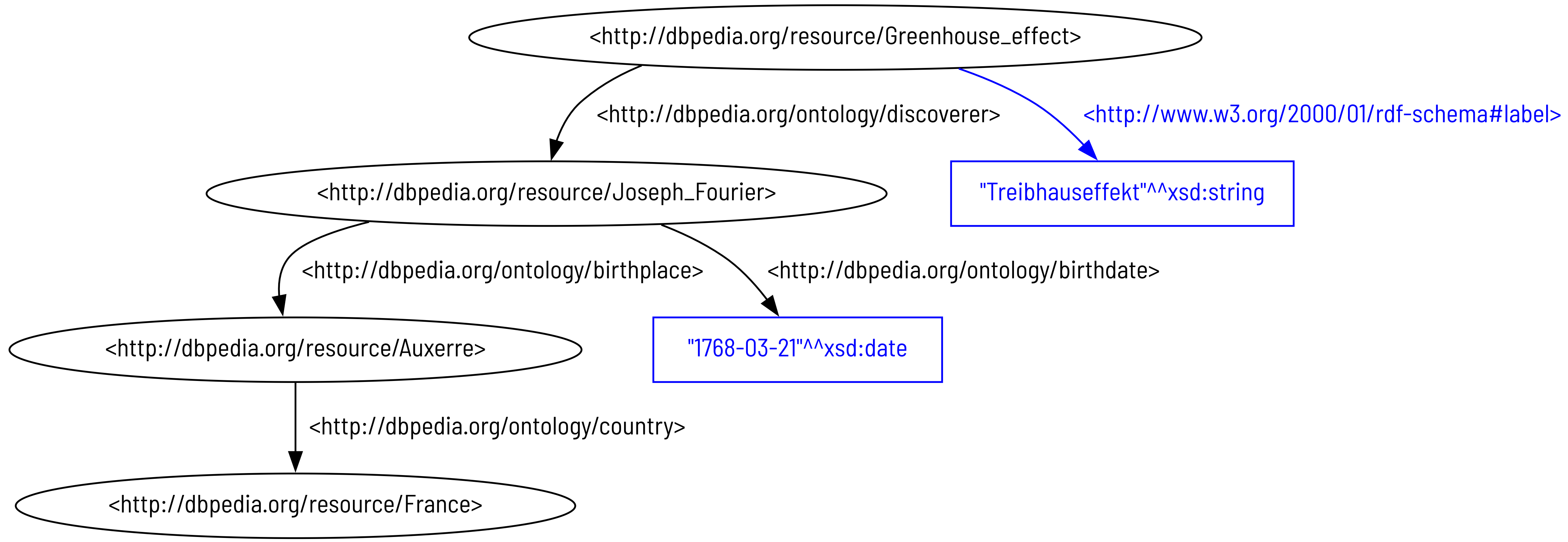
Literals

- Used for the representation of **data values**
- Representation as **strings**
- Interpretation depending on the **data type** associated with a Literal
- Literals without type information are **untyped** and treated as **plain strings**
- Represented as **boxes** in visualized RDF graphs

Typed Literals

- Typed literals are expressed via XML Schema data types
  - Namespace: `http://www.w3.org/2001/XMLSchema#`
- Language tags indicate the natural language of a text
  - Example: `"Semantik"@de`, `"Semantics"@en`
- Data types allow for a **semantic interpretation** of object values
- Data types are represented by **URIs** and can be arbitrarily chosen, but **XML Schema data types** are commonly used in RDF graphs
- Syntax: `"Data_value"^^Data_Type_URI`

Encoding Data Values in RDF

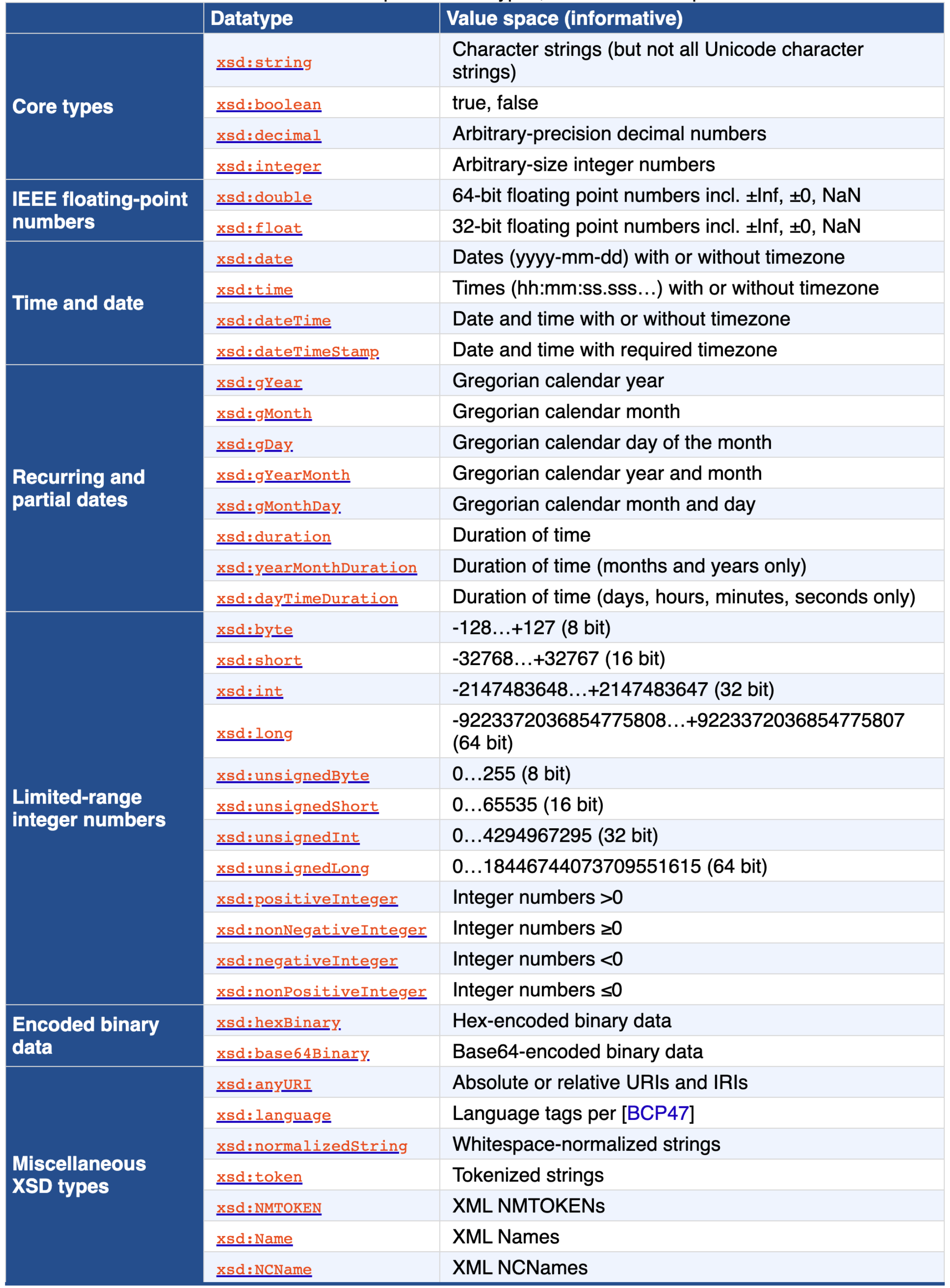

<http://dbpedia.org/resource/Greenhouse_effect>
<http://dbpedia.org/ontology/discoverer>
<http://www.w3.org/2000/01/rdf-schema#label>
<http://dbpedia.org/resource/Joseph_Fourier>
"Treibhauseffekt"^^xsd:string
<http://dbpedia.org/ontology/birthplace>
<http://dbpedia.org/ontology/birthdate>
<http://dbpedia.org/resource/Auxerre>
"1768-03-21"^^xsd:date
<http://dbpedia.org/ontology/country>
<http://dbpedia.org/resource/France>

- Data values in RDF are written in the format `"lexical value"^^datatype-IRI`.
- They are drawn as rectangular nodes in graphs.

An Overview of available XSD datatypes in RDF

More Information:

https://www.w3.org/TR/xmlschema11-2/



	Datatype	Value space (informative)
Core types	xsd:string	Character strings (but not all Unicode character strings)
	xsd:boolean	true, false
	xsd:decimal	Arbitrary-precision decimal numbers
	xsd:integer	Arbitrary-size integer numbers
IEEE floating-point numbers	xsd:double	64-bit floating point numbers incl. ±Inf, ±0, NaN
	xsd:float	32-bit floating point numbers incl. ±Inf, ±0, NaN
Time and date	xsd:date	Dates (yyyy-mm-dd) with or without timezone
	xsd:time	Times (hh:mm:ss.sss…) with or without timezone
	xsd:dateTime	Date and time with or without timezone
	xsd:dateTimeStamp	Date and time with required timezone
Recurring and partial dates	xsd:gYear	Gregorian calendar year
	xsd:gMonth	Gregorian calendar month
	xsd:gDay	Gregorian calendar day of the month
	xsd:gYearMonth	Gregorian calendar year and month
	xsd:gMonthDay	Gregorian calendar month and day
	xsd:duration	Duration of time
	xsd:yearMonthDuration	Duration of time (months and years only)
	xsd:dayTimeDuration	Duration of time (days, hours, minutes, seconds only)
Limited-range integer numbers	xsd:byte	-128…+127 (8 bit)
	xsd:short	-32768…+32767 (16 bit)
	xsd:int	-2147483648…+2147483647 (32 bit)
	xsd:long	-9223372036854775808…+9223372036854775807 (64 bit)
	xsd:unsignedByte	0…255 (8 bit)
	xsd:unsignedShort	0…65535 (16 bit)
	xsd:unsignedInt	0…4294967295 (32 bit)
	xsd:unsignedLong	0…18446744073709551615 (64 bit)
	xsd:positiveInteger	Integer numbers >0
	xsd:nonNegativeInteger	Integer numbers ≥0
	xsd:negativeInteger	Integer numbers <0
	xsd:nonPositiveInteger	Integer numbers ≤0
Encoded binary data	xsd:hexBinary	Hex-encoded binary data
	xsd:base64Binary	Base64-encoded binary data
Miscellaneous XSD types	xsd:anyURI	Absolute or relative URIs and IRIs
	xsd:language	Language tags per [BCP47]
	xsd:normalizedString	Whitespace-normalized strings
	xsd:token	Tokenized strings
	xsd:NMTOKEN	XML NMTOKENs
	xsd:Name	XML Names
	xsd:NCName	XML NCNames

Advanced Features

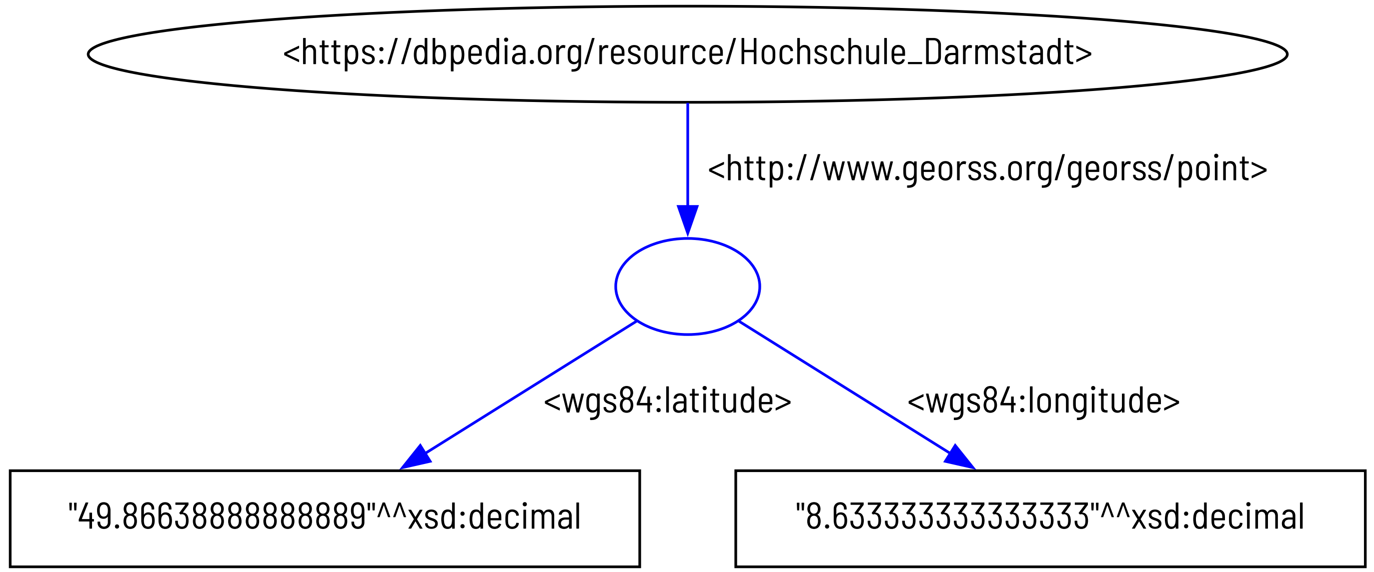
Blank Nodes

RDF also supports vertices that are not identified by a IRI, called blank nodes or bnodes.

- Used to model **multi-valued relations** (e.g. `rdf:value`)
- Used for auxiliary resources that do not need a name
- Similar to existentially quantified variables in logic

Blank nodes indicate the existence of an individual with specific attributes, but without providing **external identification** or URI-based reference information.

Example: Blank nodes have historically been used for auxiliary vertices


<https://dbpedia.org/resource/Hochschule_Darmstadt>
<http://www.georss.org/georss/point>
<wgs84:latitude>
<wgs84:longitude>
"49.86638888888889"^^xsd:decimal
"8.633333333333333"^^xsd:decimal

Note:
Today, bnodes are largely avoided. They still occur in the RDF-encoding of the OWL Web Ontology Language, but specialised tools are used in this application anyway.

Modelling in RDF using Blank Nodes

Task: Please represent the following statement as an RDF graph

The lecture "Knowledge Graphs" takes place twice a week – on TUE in Room D14/1.04 and on WED in room D14/4.03.

RDF Reification

- **Question**

  How do we model propositions about propositions ?

- **Example**

  How can we model the following assumption ?

  "The detective supposes that the buttler killed the gardener"

- This is problematic in RDF
- In the german language, such circumstance is often indicated by the word "*dass*"

RDF Reification: Solution #1

- **Example**

  "The detective supposes that the buttler killed the gardener"

- **Solution #1**

```
ex:detective      ex:supposes     "The butler killed the gardener." .
```

Shortcomings

- Literal will not be referenced in other triples
- Inherent meaning (semantics) of statement is lost

RDF Reification: Solution #2

- **Example**

  "The detective supposes that the buttler killed the gardener"

- **Solution #1**

```
ex:detective      ex:supposes     ex:theButlerKilledTheGardener .
```

Shortcomings

- Does not capture the full meaning of the proposition
- Semantics and inner structure of the assertion is lost

RDF Reification

Solution: **nested triples**

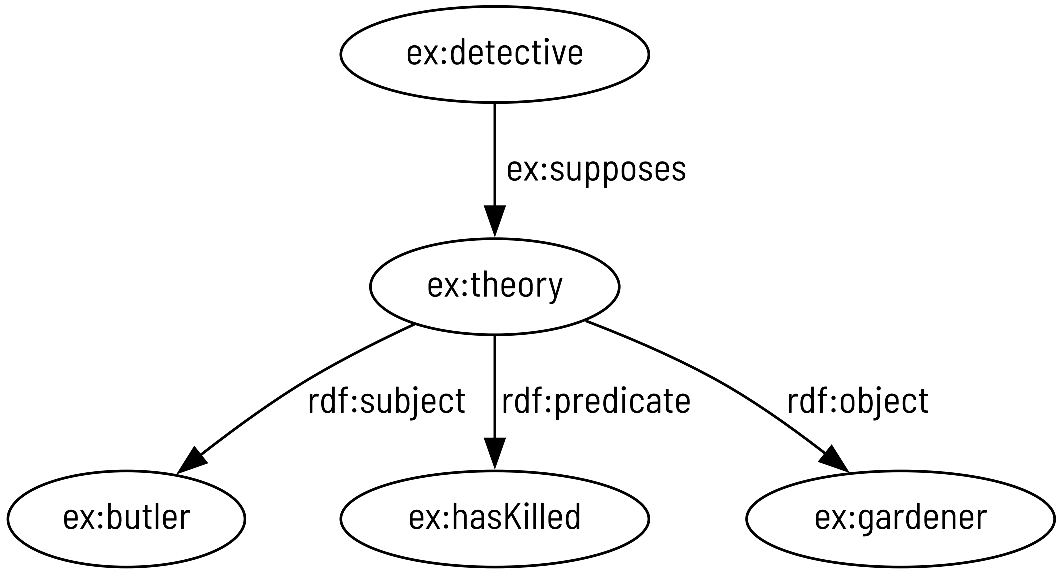
- Object of the previous triple is a triple of its own
- Draws the idea from many-valued relations (blank nodes)

Auxiliary node is used to refer to the whole statement

- Access to inner structure of represented triple is connected via a blank node with
  - `rdf:subject`, referring to a statement's subject
  - `rdf:predicate`, referring to a statement's predicate
  - `rdf:object`, referring to the object properties
- Corresponding triple is called reified

Correct Model

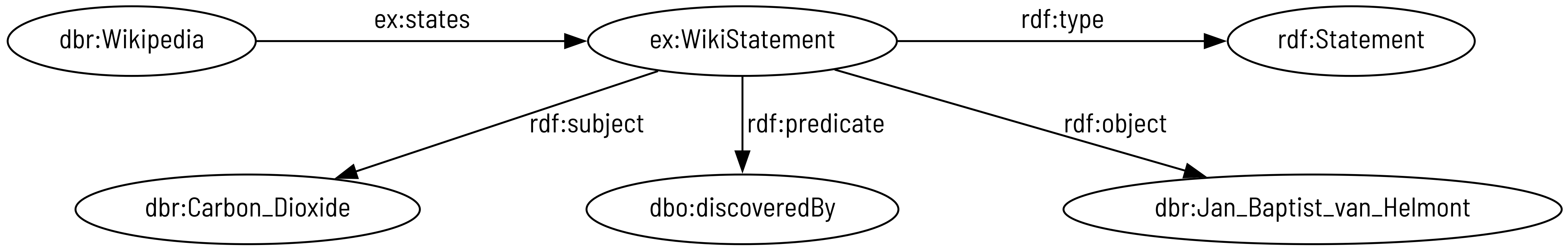
```
ex:theory       rdf:subject      ex:butler .
ex:theory       rdf:predicate    ex:hasKilled .
ex:theory       rdf:object       ex:gardener .

ex:detective    ex:supposes      ex:theory .
```



Reification: Claims about Facts

Example

"Wikipedia states that carbon dioxide was discovered by Jan Baptist van Helmont."



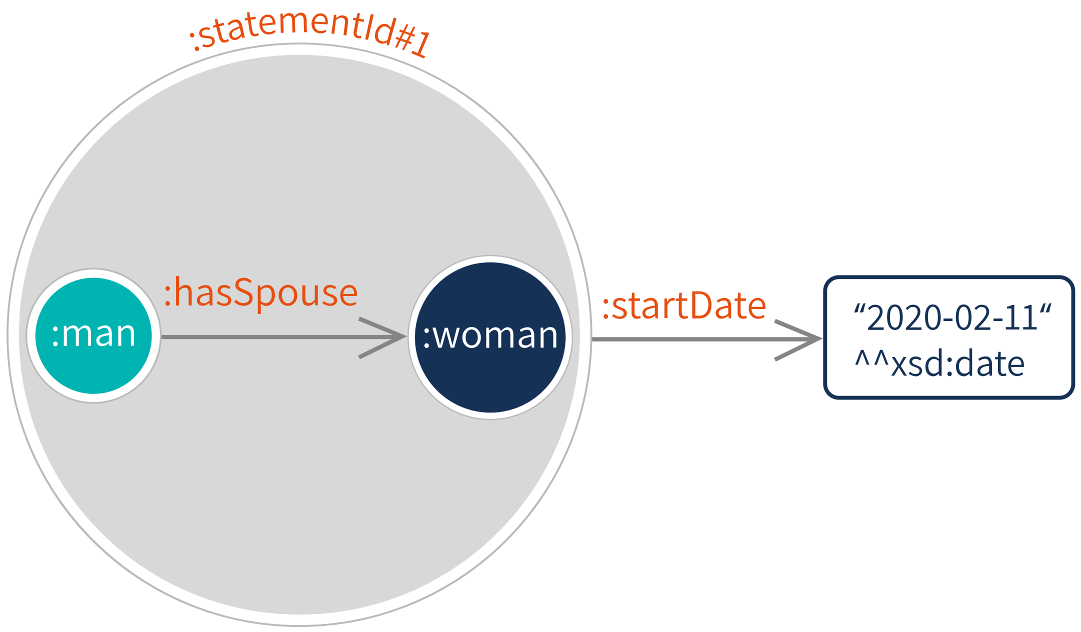
Reification: Summary

What is the use of reification ?

- Modelling **data provenance**
- Formalizing statements about **reliability** and **trust**
- Define **metadata** about statements

But reification should be used with caution...

- Relations and classes can be transformed into instances potentially resulting in **type conflicts**
- Risk to define infinite **recursions** and **cycles**



- Source: Lecture slide adapted from Prof. Dr. Harald Sack;
- Picture taken from: https://www.ontotext.com/knowledgehub/fundamentals/what-is-rdf-star/
- Interesting reads about potential RDF extensions: https://arxiv.org/pdf/1406.3399.pdf and https://www.ontotext.com/knowledgehub/fundamentals/what-is-rdf-star/
- https://github.com/w3c/rdf-star/tree/main/presentations

Design Principles of RDF

- RDF statements use **Uniform Resource Identifiers (URIs)** as names for things.
- RDF statements make use of **RDF properties** and **RDF classes**.
- Properties and classes are defined in **RDF vocabularies**.
- RDF vocabularies defined with **specific axioms** are called ontologies.
- RDF vocabularies are typically published in structured, machine-readable and resolvable schemas on the Web.
- RDF vocabularies are published on the Web in order to encourage their **re-use**.
- RDF is a language designed by humans for **processing by machines**.
  - The RDF language — the grammar together with available RDF vocabularies — does not itself solve the **difficulties** of **human communication** around data and semantics.
  - The specific strength of RDF derives from how the **triple structure** supports the process of **creating knowledge** by providing a **linguistic basis** for **expressing** and **linking data**.

Source: Baker-Coyle-Petiya: Multi-Entity Models of Resource Description in the Semantic Web: A comparison of FRBR, RDA, and BIBFRAME. Published in: Library Hi Tech, v. 32, n. 4, 2014 pp 562-582 DOI:10.1108/LHT-08-2014-0081 (Awarded "Outstanding Paper 2015" by Emerald)

Design Principles of RDF

Design Principles of RDF

- **"Anyone can say anything about anything."**
  - RDF was designed to accommodate multiple sources of information reflecting multiple points of view.
  - The grammar of RDF triples was designed to help merge data from multiple sources by leveraging shared URIs to align layers of information in a unified whole.
  - The specific strength of RDF lies in how its data structure accommodates multiple sources and viewpoints.
- **Open World Assumption (OWA)**
  - As a matter of principle, the information available at any given time may be incomplete.
  - In the Closed World Assumption, the information at hand defines the boundaries of what is known.
  - Closed-world systems are appropriate for information environments designed to capture all known facts; Systems based on the OWA are optimized for environments in which knowledge or scholarly opinion is a moving target and expected to evolve, change, or contradict itself.
- **Non-Unique Naming Assumption (NUNA)**
  - As a matter of principle, things described in RDF data can have more than one name. Because URIs are used in RDF as names, anything may be identified by more than one URI.
  - Things are not assumed to be different because they have different names.