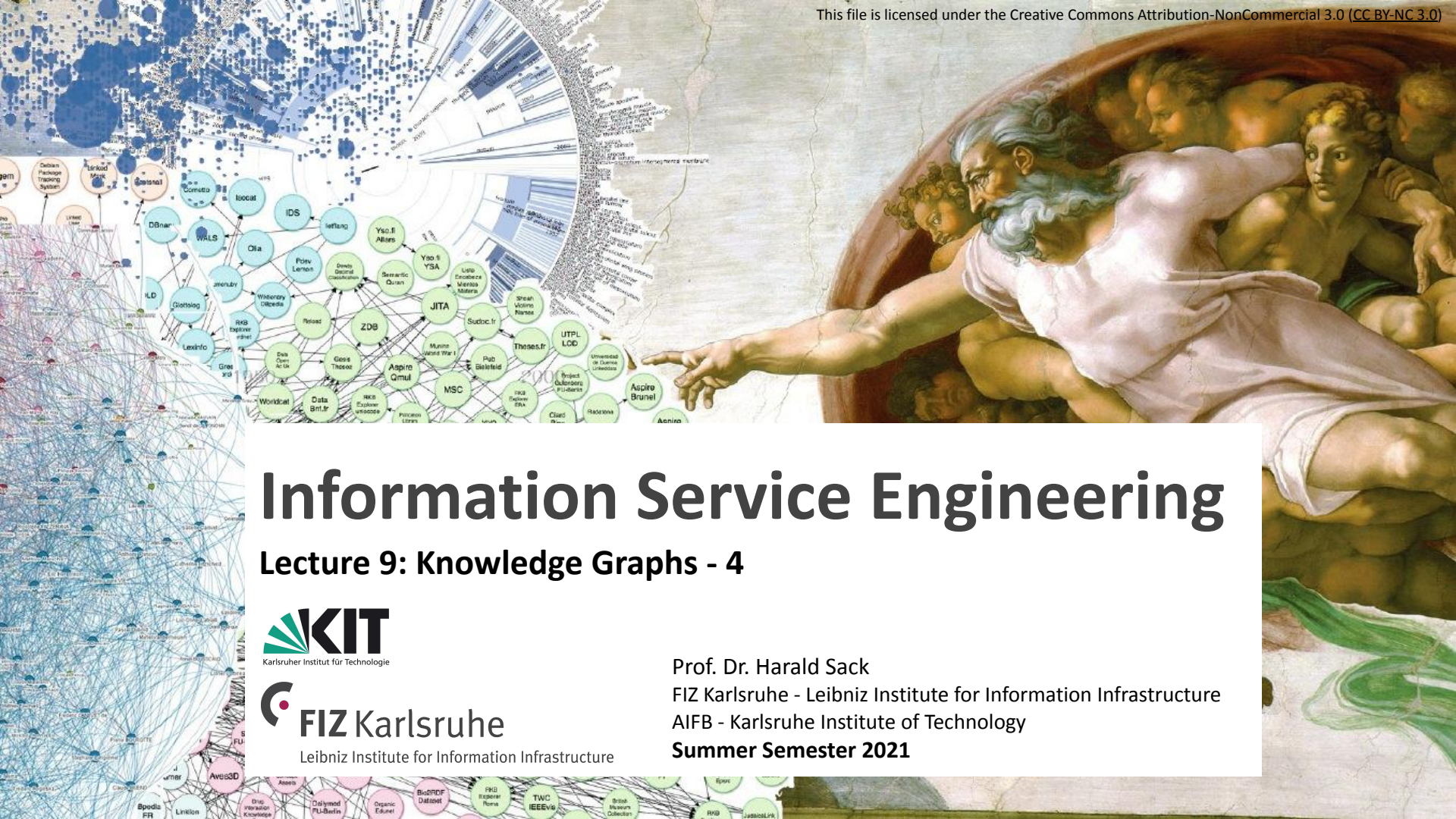


Information Service Engineering

Lecture 9: Knowledge Graphs - 4

Karlsruher Institut für Technologie

FIZ Karlsruhe
Leibniz Institute for Information Infrastructure

Prof. Dr. Harald Sack
FIZ Karlsruhe - Leibniz Institute for Information Infrastructure
AIFB - Karlsruhe Institute of Technology

Summer Semester 2021

Information Service Engineering

Last Lecture: Knowledge Graphs - 3

3.1 Knowledge Representations and Ontologies
3.2 Semantic Web and the Web of Data
3.3 Linked Data Principles
3.4 How to identify Things - URIs
3.5 Resource Description Framework (RDF) as simple Data Model
3.6 Creating new Models with RDFS
3.7 Knowledge Graphs
3.8 Querying Knowledge Graphs with SPARQL
3.9 More Expressivity with Web Ontology Language (OWL)
3.10 Knowledge Graph Programming

- DBpedia & wikidata
- SPARQL Query Language
- Graph Patterns
- SPARQL General Query Format
- SPARQL Protocol
- SPARQL with Regular Expressions
- SPARQL Aggregations
- SPARQL Federated Queries

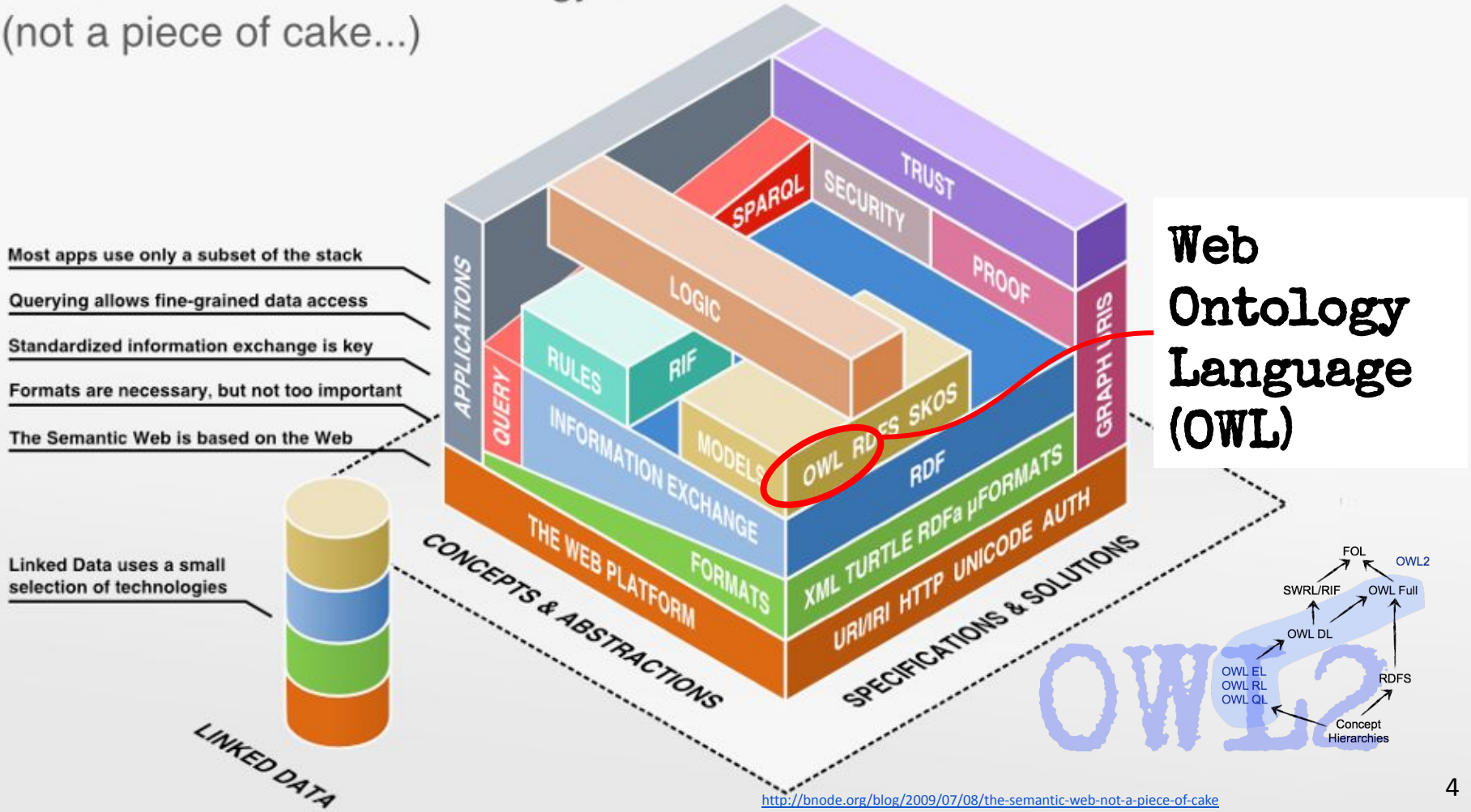
Information Service Engineering

Lecture 9: Knowledge Graphs - 4

Karlsruher Institut für Technologie

FIZ Karlsruhe
Leibniz Institute for Information Infrastructure

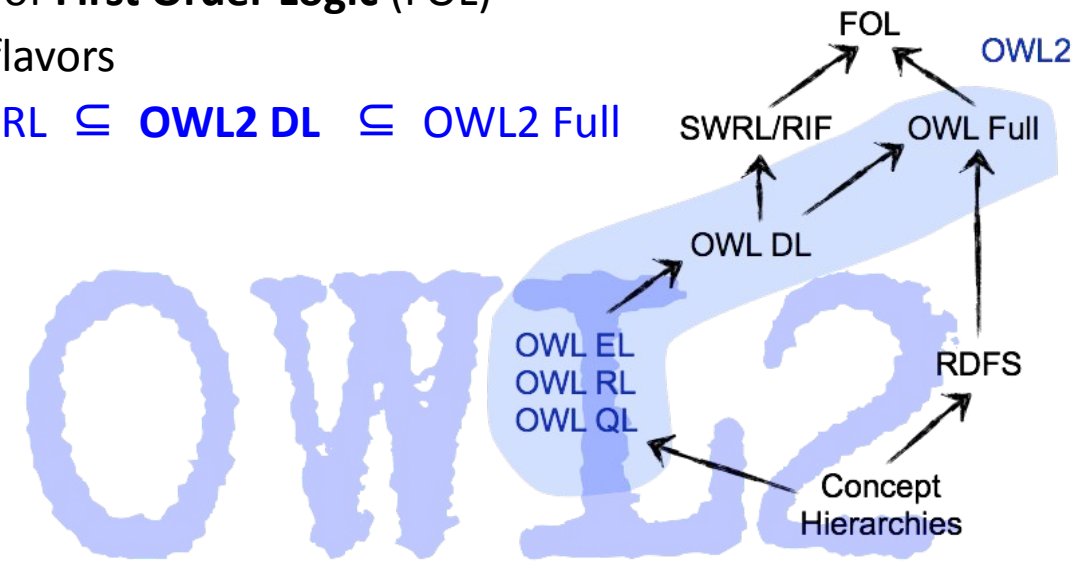
The Semantic Web Technology Stack
(not a piece of cake...)



http://bnode.org/blog/2009/07/08/the-semantic-web-not-a-piece-of-cake

Web Ontology Language OWL - OWL Flavors

- OWL is a semantic fragment of **First Order Logic** (FOL)
- OWL also exists in different flavors
  - OWL EL, OWL QL, OWL RL ⊆ **OWL2 DL** ⊆ OWL2 Full



OWL2 is based on the Description Logic $\mathcal{SROIQ}(\mathcal{D})$

Class Expressions

- Class names A, B
- Conjunction $C \sqcap D$
- Disjunction $C \sqcup D$
- Negation $\neg C$
- Exist. property restriction $\exists R.C$
- Univ. property restriction $\forall R.C$
- Self $\exists S.Self$
- Greater-than $\geq n\ S.C$
- Less-than $\leq n\ S.C$
- Enumerated classes $\{a\}$

Properties

- Property names R, S, T
- Simple properties S, T
- Inverse properties R^-
- Universal property U

Tbox (Class axioms)

- Inclusion $C \sqsubseteq D$
- Equivalence $C \equiv D$

Rbox (Property Axioms)

- Inclusion $R_1 \sqsubseteq R_2$
- General Inclusion $R^{(-)}_1 \circ R^{(-)}_2 \circ \ldots \circ R^{(-)}_n \sqsubseteq R$
- Transitivity
- Symmetry
- Reflexivity
- Irreflexivity
- Disjunctiveness

Abox (Facts)

- Class membership $C(a)$
- Property relation $R(a,b)$
- Negated property relation $\neg S(a,b)$
- Equality $a=b$
- Inequality $a \neq b$

OWL Basic Building Blocks

- OWL **namespace**:
  `@prefix owl: <http://www.w3.org/2002/07/owl#>`
- There is a **Turtle Syntax** for OWL.
- OWL axioms consist of the following three **building blocks**:
  - **Classes**
    - Comparable with classes in RDFS.
  - **Individuals**
    - Comparable with class instances in RDFS.
  - **Properties**
    - Comparable with properties in RDFS.

OWL Classes

- There exist two **predefined classes**
  - `owl:Thing` (class that contains all individuals)
  - `owl:Nothing` (empty class)

equivalent expression in description logics

$\top \equiv C \sqcup \neg C$

$\bot \equiv C \sqcap \neg C$

- **Definition of a class**

```
:GreenhouseGas a owl:Class .
```

This is OWL in RDF/Turtle serialization

OWL Individuals

- **Definition of individuals** via class membership

```
:JosephFourier a :Person .
```

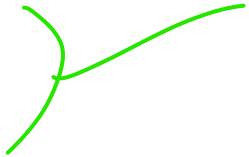
Person(JosephFourier)

- Individuals can also be defined **without class membership** as **named individuals**

```
:HaraldSack a owl:NamedIndividual .
```

OWL Object Properties

- There exist two **property variants**:
  - object properties
  - datatype properties



- **Object properties** have classes as range

```
:discoverer a owl:ObjectProperty .
```

- **Domain** and **Range** of object properties

```
:discoverer a owl:ObjectProperty ;
            rdfs:domain owl:Thing ;
            rdfs:range  :Person .
```

$\exists \texttt{discoverer}.\top \sqsubseteq \top$

$\top \sqsubseteq \forall \texttt{discoverer}.\texttt{Person}$

OWL Datatype Properties

- **Datatype properties** have datatypes as range

```
:discoveredIn a owl:DatatypeProperty .
```

- **Domain** and **Range** of datatype properties

```
:discoveredIn a owl:DatatypeProperty ;
              rdfs:domain owl:Thing ;
              rdfs:range  xsd:date .
```

$\exists \text{discoveredIn}.\top \sqsubseteq \top$

$\top \sqsubseteq \forall \text{discoveredIn}.\text{Date}$

OWL Properties and Individuals - Example

OWL TBox

```
:AtmosphericProcess a owl:Class .
:Person a owl:Class .

:discoverer a owl:ObjectProperty ;
            rdfs:domain owl:Thing ;
            rdfs:range  :Person .

:discoveredIn a owl:DatatypeProperty ;
              rdfs:domain owl:Thing ;
              rdfs:range  xsd:date .
```

```
:JosephFourier a Person .
:GreenhouseEffect a :AtmosphericProcess ;
              :discoverer :JosephFourier ;
              :discoveredIn “1824-00-00”^^xsd:date .
```

OWL ABox

OWL Class Hierarchies

```
:Physicist a owl:Class ;
       rdfs:subClassOf :Scientist .
:Scientist a owl:Class ;
      rdfs:subClassOf :Person .
:Person a owl:Class .
```

we don't need to define a new subClassOf property for OWL, we simply reuse `rdfs:subClassOf`

$$\text{Physicist} \sqsubseteq \text{Scientist}$$
$$\text{Scientist} \sqsubseteq \text{Person}$$

- Via **inference** it can be entailed that `:Physicist` is also a subclass of `:Person`.

OWL Class Hierarchies and Disjunctiveness

```
:ChemicalSubstance a owl:Class .
:Person a owl:Class .
:GreenhouseGas a owl:Class ;
       rdfs:subClassOf :ChemicalSubstance .
:Scientist a owl:Class ;
      rdfs:subClassOf :Person .

:ChemicalSubstance owl:disjointWith :Person .
```

In OWL everything might be potentially identical if we don't explicitly state the difference

$$\texttt{GreenhouseGas} \sqsubseteq \texttt{ChemicalSubstance}$$
$$\texttt{Scientist} \sqsubseteq \texttt{Person}$$
$$\texttt{ChemicalSubstance} \sqcap \texttt{Person} \sqsubseteq \bot$$

- Via **inference** it can be entailed that `:GreenhouseGas` and `:Scientist` are also disjoint classes.

OWL Class Hierarchies and Equivalence

```
:Scientist  a owl:Class .
:Researcher a owl:Class .
:Physicist  a owl:Class ;
      rdfs:subClassOf :Scientist .

:Scientist  owl:equivalentClass :Researcher .
```

$$\texttt{Physicist} \sqsubseteq \texttt{Scientist}$$
$$\texttt{Scientist} \equiv \texttt{Researcher}$$

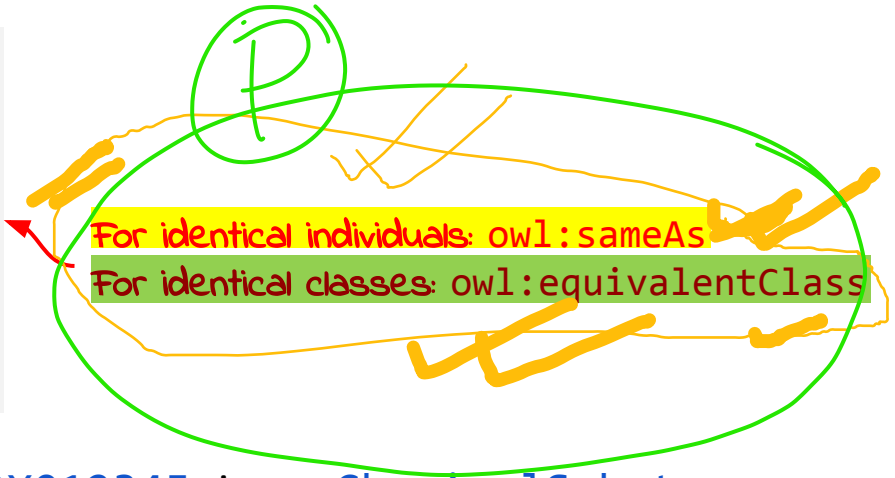
- Via **inference** it can be entailed that `:Physicist` is also a `:Researcher`.

OWL Individuals - Identity and Distinctiveness

```
:CarbonDioxide a :GreenhouseGas ;
    :discoverer :JosephBlack ;
    :discoveredIn "1750-00-00"^^xsd:date ;
    owl:sameAs :ARX012345 .

:GreenhouseGas a owl:Class ;
       rdfs:subClassOf :ChemicalSubstance .

:ChemicalSubstance a owl:Class.
```



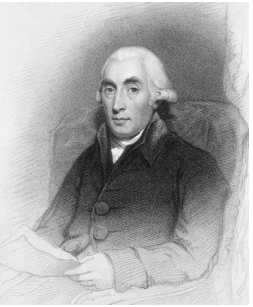
- Via **inference** it can be entailed that `:ARX012345` is a `:ChemicalSubstance`.
- **Difference of Individuals** via `owl:differentFrom`.

```
:ARX012345 a :GreenhouseGas ;
           owl:differentFrom :ARX012346 .
```

OWL Complex Classes - Nominals

```
:Chemist   a owl:Class .
:Physicist a owl:Class .
:JosephFourier a :Physicist .
:JanBaptistVanHelmont a :Physicist .
:JosephBlack a Chemist .

:CarbonDioxideClub a owl:Class ;
     owl:oneOf
     ( :JosephFourier
       :JanBaptistVanHelmont
       :JosephBlack ) .
```



CarbonDioxideClub $\sqsubseteq$ { JosephFourier, JanBaptistVanHelmont, JosephBlack }

No Comma inside.

- There are only three scientists in the Carbon Dioxide Club.

OWL Logical Class Constructors

- logical AND (conjunction): `owl:intersectionOf` ⊓
- logical OR (disjunction): `owl:unionOf` ⊔
- logical negation: `owl:complementOf` ¬

Used to create complex classes from atomic classes.

OWL Logical Class Constructors - Intersection

```
:Scientist a owl:Class .

:ClimateActivist a owl:Class .

:Scientist4Future a owl:Class ;
      owl:intersectionOf (:Scientist :ClimateActivist) .
```

$$\texttt{Scientist4Future} \equiv \texttt{Scientist} \sqcap \texttt{ClimateActivist}$$

- Scientists4Future are Scientists who are also ClimateActivists.

OWL Logical Class Constructors - Union

```
:Environmentalist a owl:Class ;
      owl:equivalentClass [
            owl:unionOf ( :ClimateActivist
                          :AnimalRightsActivist
                          :EnergySaver )
] .
```

$$\texttt{Environmentalist} \equiv \texttt{ClimateActivist} \sqcup \texttt{AnimalRightsActivist} \sqcup \texttt{EnergySaver}$$

- Climate Activists, Animal Rights Activists, and Energy Savers are all Environmentalists.

OWL Logical Class Constructors - Negation

```
:Pacifist a owl:Class .

:Warmonger a owl:Class ;
     owl:complementOf (:Pacifist) .
```

$$Warmonger \equiv \neg Pacifist$$

- Warmongers are the opposite of Pacifists.

OWL Property Restrictions

- **OWL property restrictions** are used to describe **complex classes** via **properties**
- Restrictions on values:
  - `owl:hasValue`
  - `owl:allValuesFrom`
  - `owl:someValuesFrom`
- Restrictions on cardinality:
  - `owl:cardinality`
  - `owl:minCardinality`
  - `owl:maxCardinality`

OWL Property Restrictions with Constants

```
:FouriersDiscoveries a owl:Class ;
                     rdfs:subClassOf
                       [ a owl:Restriction ;
                         owl:onProperty :discoverer ;
                         owl:hasValue :JosephFourier ] .
```

$$\text{FouriersDiscoveries} \sqsubseteq \text{discoverer.}(\text{JosephFourier})$$

- The class `:FouriersDiscoveries` is described via fixed value assignment (=constant) of the individual `:JosephFourier` to the property `:discoverer`.
- Fourier's Discoveries have been discovered by Joseph Fourier.

OWL Properties Restriction with Strict Binding

```
:VegetarianDish a owl:Class ;
     rdfs:subClassOf
        [ a owl:Restriction ;
          owl:onProperty :ingredient ;
          owl:allValuesFrom :VegetarianFood ] .
```

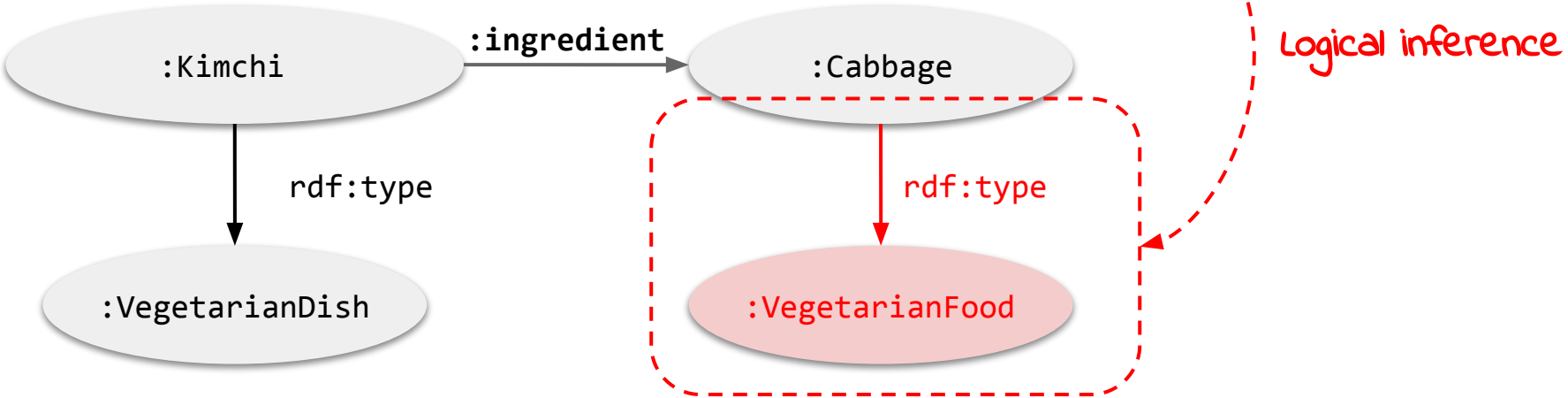
$$\text{VegetarianDish} \sqsubseteq \forall \text{ingredient}.\text{VegetarianFood}$$

- `owl:allValuesFrom` fixes all instances of a specific class C as allowed range for a property p (strict binding) $\forall p.C$

- The **ingredients** of a **vegetarian dish** are only **vegetarian food.**

OWL Properties Restriction with Strict Binding

```
:VegeratianDish a owl:Class ;
        rdfs:subClassOf
          [ a owl:Restriction ;
            owl:onProperty :ingredient ;
            owl:allValuesFrom :VegetarianFood ] .
```

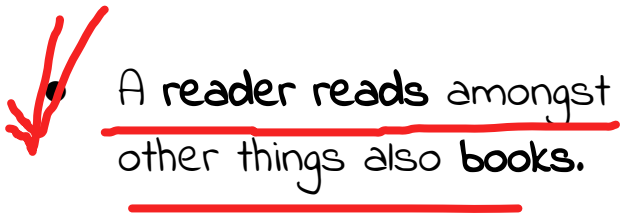


OWL Property Restriction with Loose Binding

```
:Reader a owl:Class ;
        rdfs:subClassOf
          [ a owl:Restriction ;
            owl:onProperty :reads ;
            owl:someValuesFrom :Book ] .
```

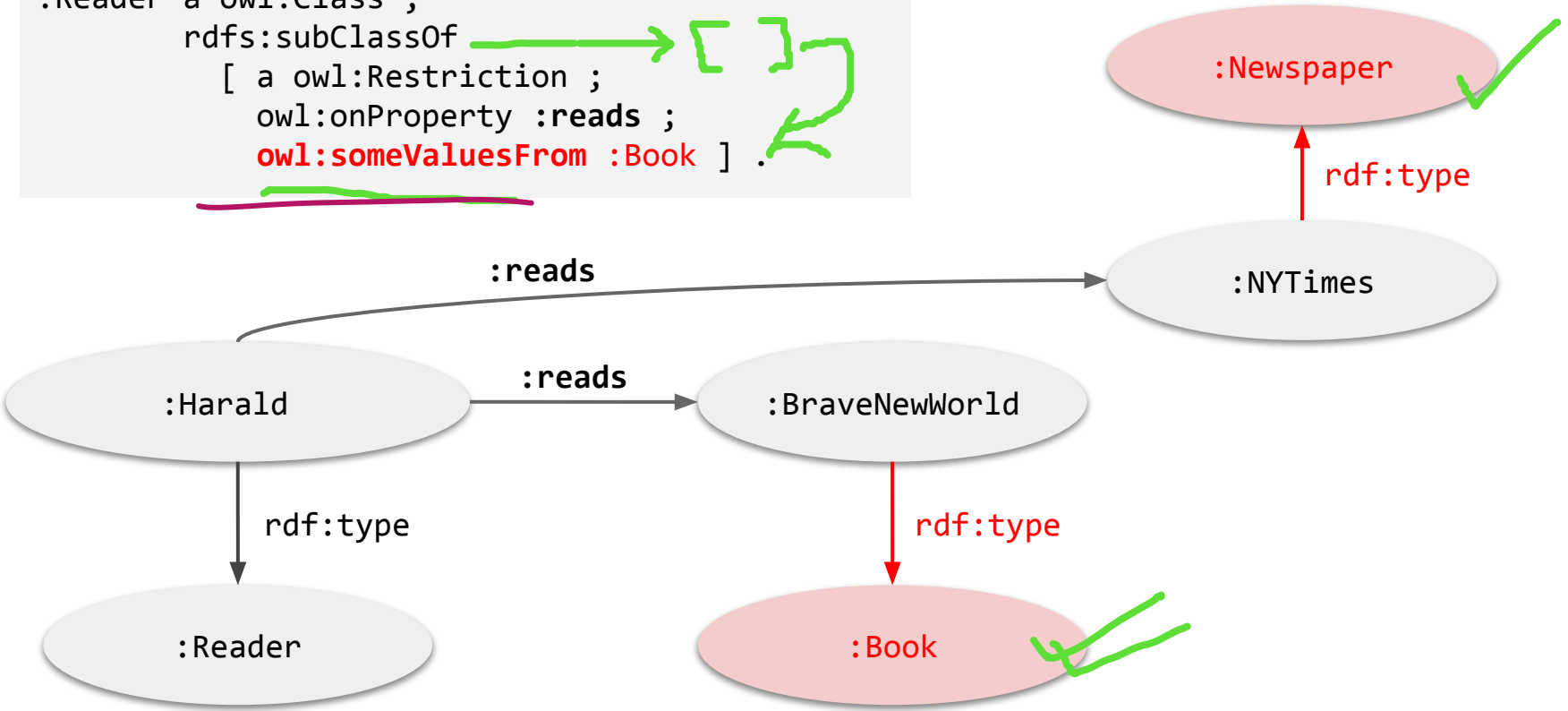
$\texttt{Reader} \sqsubseteq \exists \texttt{reads.Book}$

- owl:someValuesFrom describes that there must exist an individual for p and fixes its range to class C (loose binding) $\exists p.C$



OWL Property Restriction with Loose Binding

```
:Reader a owl:Class ;
        rdfs:subClassOf
          [ a owl:Restriction ;
            owl:onProperty :reads ;
            owl:someValuesFrom :Book ] .
```



OWL Property Restrictions

- OWL property restrictions are used to describe complex classes via properties.
- Restrictions on values:
  - `owl:hasValue`
  - `owl:allValuesFrom`
  - `owl:someValuesFrom`
- Restrictions on cardinality:
  - `owl:cardinality`
  - `owl:minCardinality`
  - `owl:maxCardinality`

OWL Property Restrictions with Cardinality

```
:StringQuartet a owl:Class ;
      rdfs:subClassOf
         [ a owl:Restriction ;
           owl:onProperty :member ;
           owl:cardinality "4"^^<http://www.w3.org/2001/XMLSchema#int> ] .
```

$StringQuartett \sqsubseteq =4.member.\top$

- A String Quartet has exactly 4 members.
- Any instance of Class `:StringQuartet` must have exactly 4 values for the property `:member`.
- For `owl:maxCardinality` and `owl:minCardinality` the restriction gives **upper and lower bounds** on property value cardinalities.

Loose or Strict Binding?

- "A vegetarian does not eat animals"
- How to model this restriction with the classes :Vegetarian, :Animal and the property :eats ?

```
Vegetarian a owl:Class ;
      rdfs:subClassOf [
            a owl:Restriction ;
            owl:onProperty :eats ;
            owl:allValuesFrom [
                  owl:complementOf :Animal .
            ] .
]
```

Loose or Strict Binding?

- "A Driver is somebody who drives a car."
- How to model this restriction with the classes :Driver, :Car and the property :drives ?

```
Driver a owl:Class ;
       rdfs:subClassOf [
            a owl:Restriction ;
            owl:onProperty :drives ;
            owl:someValuesFrom :Car .
]
```

OWL Property Relationships

- **Property hierarchies** can be created via specializations: `rdfs:subPropertyOf`.
- **Inverse properties** are defined via `owl:inverseOf`.
- **Identical properties** are defined via `owl:equivalentProperty`.

```
:isMadeOf a owl:ObjectProperty ;
        rdfs:subPropertyOf :consistsOf .

:reads a owl:ObjectProperty ;
        owl:inverseOf :isReadBy .

:composedOf a owl:ObjectProperty ;
        owl:equivalentProperty :consistsOf .
```

```
isMadeOf    ⊑ consistsOf
reads-      ≡ isReadBy
composedOf  ≡ consistsOf
```

OWL Property Relationships

- `owl:TransitiveProperty`
  e.g.: if `isPartOf(a,b)` and `isPartOf(b,c)` then it holds that `isPartOf(a,c)`.
- `owl:SymmetricProperty`
  e.g.: if `isNeighborOf(a,b)` then it holds that `isNeighborOf(b,a)`.
- `owl:FunctionalProperty`
  e.g.: if `hasMother(a,b)` and `hasMother(a,c)` then it holds that `b=c`.
- `owl:InverseFunctionalProperty`
  e.g.: if `isMotherOf(b,a)` and `isMotherOf(c,a)` then it holds that `b=c`.

OWL Transitive Properties

```
:isPublishedBefore a owl:ObjectProperty ;
                   a owl:TransitiveProperty ;
                   rdfs:domain owl:Book ;
                   rdfs:range owl:Book .

:AnimalFarm a owl:Book ;
            :isPublishedBefore :NineteenEightyfour .

:BraveNewWorld a :Book ;
            :isPublishedBefore :AnimalFarm .
```

- Via inference it can be entailed that
  `:BraveNewWorld :isPublishedBefore :NineteenEightyfour .`

More Property Relationships

- **Asymmetric properties** via `owl:AsymmetricProperty`
  e.g.: if it holds that `isLeftOf(a,b)`
  then it is not possible that `isLeftOf(b,a)` .
- **Reflexive properties** via `owl:ReflexiveProperty`
  e.g.: `isRelatedTo(x,x)` .
- **Irreflexive properties** via `owl:IrreflexiveProperty`
  e.g.: if `isParentOf(x,y)` then `x≠y` .

Disjunctive Properties

- Two properties R and S are **disjunctive**,
  if two individuals x,y are never related via both properties

```
:hasSibling a owl:ObjectProperty ;
            owl:propertyDisjointWith :hasChild .
```

- **Shortcut** for several **disjunctive properties**

```
[] rdf:type owl:AllDisjointProperties ;
   owl:members
       ( :hasSibling
         :hasChild
         :hasGrandchild ) .
```

Negated Property Instantiations

- Two individuals can explicitly be defined to be **not related with each other** via a given property.

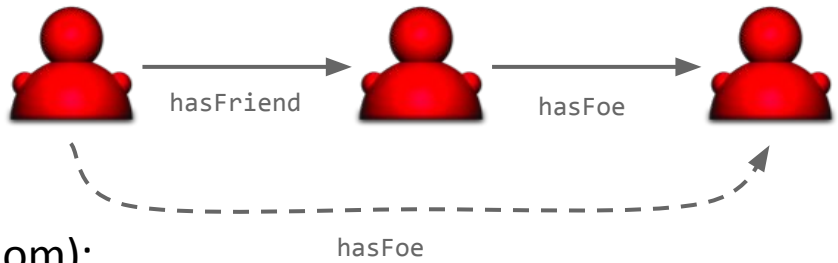
¬isBrother(GeorgeOrwell,AldousHuxley)

```
[] rdf:type owl:NegativePropertyAssertion ;
   owl:sourceIndividual :GeorgeOrwell ;
   owl:assertionProperty :isBrother ;
   owl:targetIndividual :AldousHuxley .
```

- George Orwell is not the brother of Aldous Huxley.

OWL Property Chaining

- **Complex Roles** (properties) can be constructed from simple roles (**RBox**):
  - „The friends of my friends are also my friends."
    ```
    :hasFriend a owl:TransitiveProperty .
    ```
- But: „The foes of my friends are also my foes."
  - cannot be expressed in that way



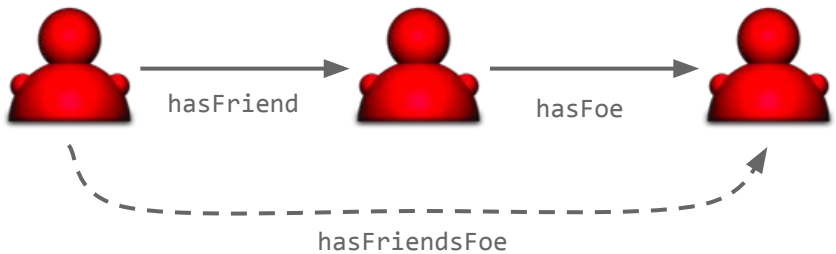
- In FOL it can be expressed as a rule (axiom):
  - $\forall x,y,z : hasFriend(x,y) \wedge hasFoe(y,z) \rightarrow hasFriendsFoe(x,z)$

OWL Property Chaining

- **General Role Inclusion** (**property chaining**):

```
:hasFriendsFoe a owl:ObjectProperty ;
   owl:propertyChainAxiom  ( :hasFriend :hasFoe ) .
```

- Not allowed for datatype properties



OWL Qualified Number Restrictions

- Class constructors with **number restrictions on properties connected with a range constraint**

$$\text{SuccessfulAuthor} \sqsubseteq \geq 1\ \text{notableWork.Bestseller}$$

```
:SuccessfulAuthor a owl:Class;
    rdfs:subClassOf [
      a owl:Restriction ;
      owl:onProperty :notableWork ;
      owl:minQualifiedCardinality  "1"^^xsd:nonNegativeInteger ;
      owl:onClass :Bestseller ] .
```

- owl:maxQualifiedCardinality, owl:minQualifiedCardinality, owl:qualifiedCardinality

OWL Reflexive Property Restriction

- Classes that contain individuals that are **related to themselves** for specific properties

$$\text{Philosopher} \sqsubseteq \text{knows.self}$$

```
:Philosopher a owl:Class ;
    rdfs:subClassOf [
      a owl:Restriction ;
      owl:onProperty :knows ;
      owl:hasSelf "true"^^xsd:boolean ] .
```

- A **Philosopher** is somebody who **knows himself.**

Beyond OWL

- More **expressivity** also means more **complexity**.
  - This might lead to **undecidability** (as for FOL).
- Do we really need more expressivity than OWL DL offers?
- Consider the following example:
  - „A squanderer is a person whose expenses are higher than his income"

- $\mathtt{Squanderer} \sqsubseteq \mathtt{Person}$
- $\mathtt{Squanderer} \sqsubseteq \mathtt{hasExpenses}.\top$
- $\mathtt{Squanderer} \sqsubseteq \mathtt{hasIncome}.\top$

?

- We need a constructor to combine Classes and Properties.
- **Problem:** Mixing of TBox and ABox.

Rules Beyond OWL

- The following example can be expressed via a a **FOL-Rule**:
  - „A squanderer is a person whose expenses are higher than his income"

$$\text{Person}(x) \land \text{hasIncome}(x,y) \land \text{hasExpenses}(x,z) \land (z > y) \rightarrow \text{Squanderer}(x)$$

- Arithmetics can be part of rules and modeled like a predicate:

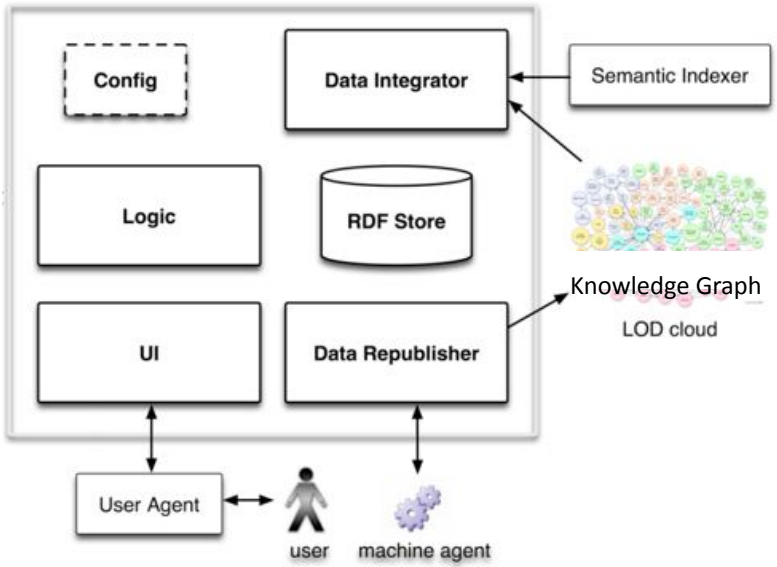
$$(z > y) \triangleq \text{greaterThan}(z,y)$$

Information Service Engineering

Lecture 9: Knowledge Graphs - 4

Knowledge Graph Driven Web Applications

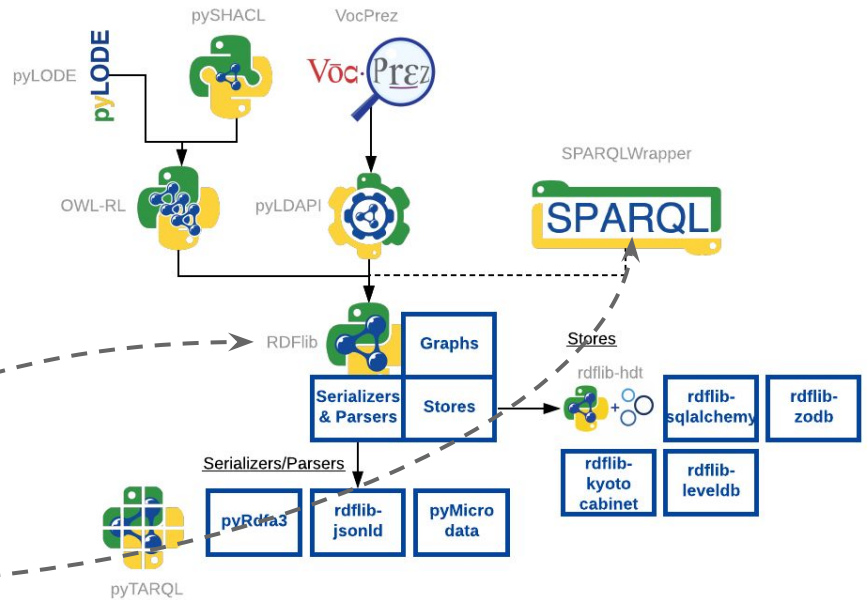
- Required Components:
  - **Local RDF Store**
    - caching of results
    - permanent storage
  - **Logic** (Controller) and
  - **User Interface** (=Business Logic)
    - (not KG specific)
  - **Data Integration component**
    - get data directly from Web of Data or
    - via Semantic Indexer
  - **Data Re-/Publishing component**
    - write back application dependent data into the Web of Data



M.Hausenblas: Linked Data Applications, DERI Technical Report, 2009

Knowledge Graph Driven Web Applications

- The easiest way is to make use of a suitable library:
  - SPARQL Javascript Library
    http://www.thefigtrees.net/lee/blog/2006/04/sparql_calendar_demo_a_sparql.html
  - ARC for SPARQL (PHP)
    https://github.com/semsol/arc2/wiki
  - dotNetRDF (C#)
    https://dotnetrdf.github.io/
  - Jena/ARQ (Java)
    http://jena.apache.org/
  - Sesame (Java)
    http://rdf4j.org/
  - **RDFlib (Python)**
    https://rdflib.dev/
  - **SPARQL Wrapper (Python)**
    http://rdflib.github.io/sparqlwrapper/



Knowledge Graph Programming Example 1

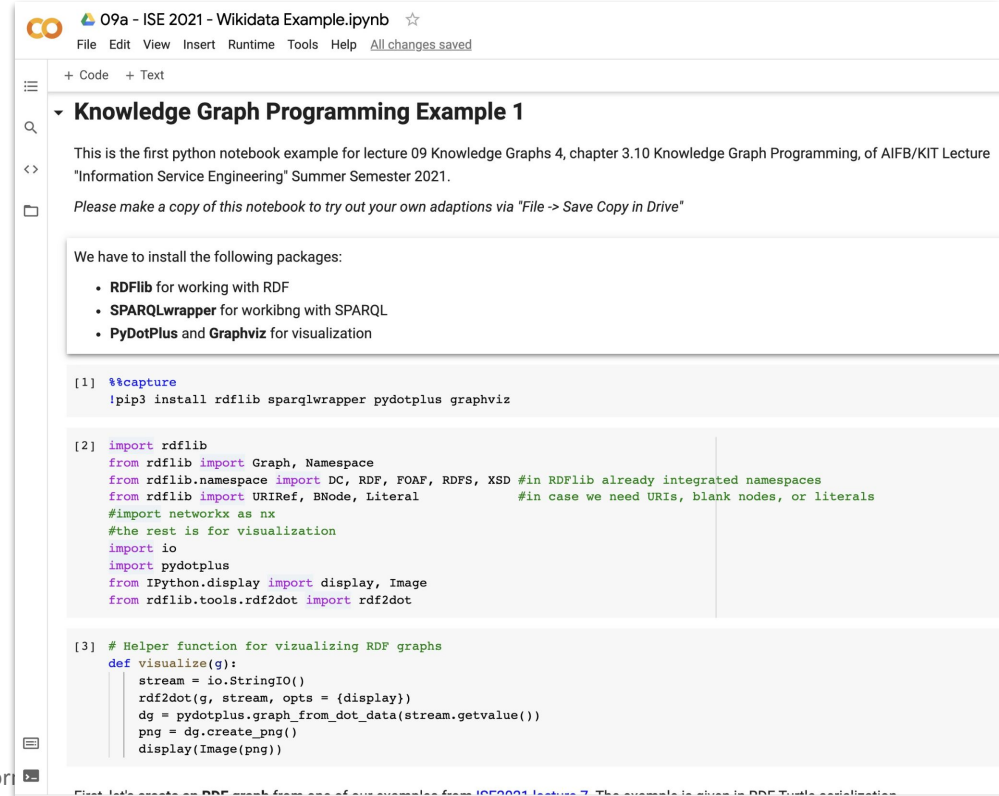
Karlsruher Institut für Technologie

FIZ Karlsruhe
Leibniz Institute for Information Infrastructure

- The easiest way is to make use of a suitable library:
  - **RDFlib (Python)**
    https://rdflib.dev/

- Simple Example Tasks:
  - Reading an RDF graph
  - Displaying an RDF graph
  - Manipulating an RDF graph

Knowledge Graph Programming Example 1



09a - ISE 2021 - Wikidata Example.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

Knowledge Graph Programming Example 1

This is the first python notebook example for lecture 09 Knowledge Graphs 4, chapter 3.10 Knowledge Graph Programming, of AIFB/KIT Lecture "Information Service Engineering" Summer Semester 2021.

Please make a copy of this notebook to try out your own adaptions via "File -> Save Copy in Drive"

We have to install the following packages:

- **RDFlib** for working with RDF
- **SPARQLwrapper** for workibng with SPARQL
- **PyDotPlus** and **Graphviz** for visualization

```
[1] %%capture
    !pip3 install rdflib sparqlwrapper pydotplus graphviz
```

```
[2] import rdflib
    from rdflib import Graph, Namespace
    from rdflib.namespace import DC, RDF, FOAF, RDFS, XSD #in RDFlib already integrated namespaces
    from rdflib import URIRef, BNode, Literal             #in case we need URIs, blank nodes, or literals
    #import networkx as nx
    #the rest is for visualization
    import io
    import pydotplus
    from IPython.display import display, Image
    from rdflib.tools.rdf2dot import rdf2dot
```

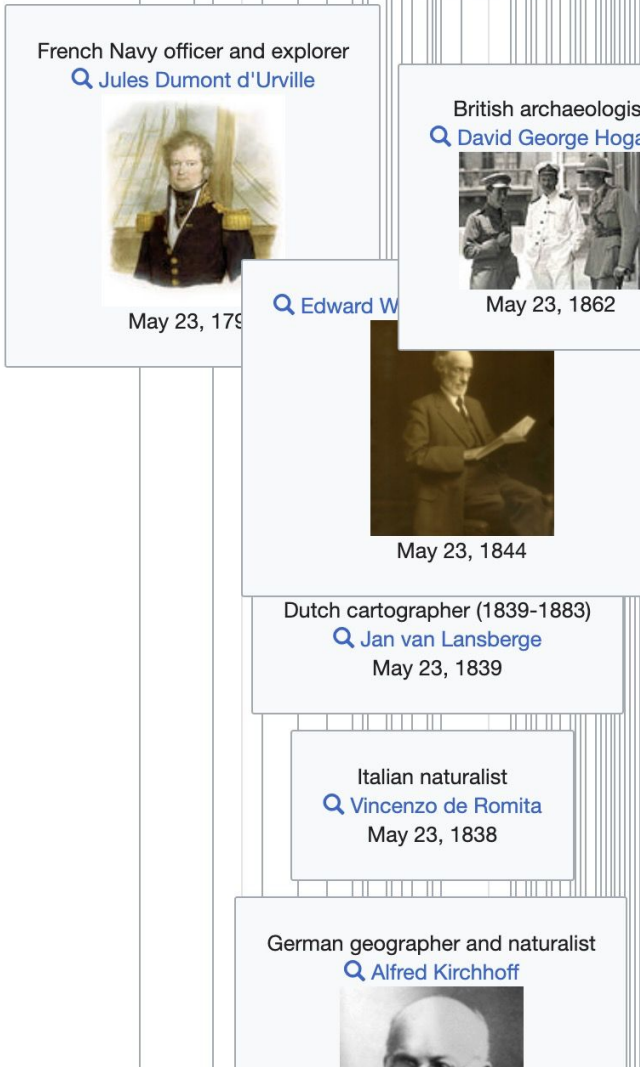
```
[3] # Helper function for vizualizing RDF graphs
    def visualize(g):
        stream = io.StringIO()
        rdf2dot(g, stream, opts = {display})
        dg = pydotplus.graph_from_dot_data(stream.getvalue())
        png = dg.create_png()
        display(Image(png))
```

Knowledge Graph Programming Example 2

- For the 2nd task we will use the SPARQLWrapper library
  - **SPARQL Wrapper (Python)**
    http://rdflib.github.io/sparqlwrapper/
- We will access Linked Data via **SPARQL endpoints** over the Web
  - let‘s choose **DBpedia** (just for simplicity...)
    http://dbpedia.org/sparql
- ...now we have to think of a suitable example task...

Knowledge Graph Programming Example 2

- **Example Application:**
  - Build a simple application that looks for famous people having their birthdays **today(!)**, as e.g. **scientists**, based on **DBpedia**.
  - Create a **list of scientists**, whose birthday is today including some **additional information**, as e.g.
    - Year of Birth
    - Short description, picture (if available), etc.
  - Let‘s create a simple (local) **web page** for the task (i.e. encode results in HTML), which can be displayed in the browser.
  - We use **Python** and the **SPARQL Wrapper library**.



Prerequisites - (Manual) Data Analysis

- Choose a representative example:
  - E.g. **Joseph Fourier** from **DBpedia**

DBpedia Browse using Formats Faceted Browser Sparql Endpoint

About: Joseph Fourier

An Entity of Type : scientist, from Named Graph : http://dbpedia.org, within Data Space : dbpedia.org

Jean-Baptiste Joseph Fourier - (/ˈfʊəriˌeɪ, -iər/; French: [fuʁje]; 21 March 1768 – 16 May 1830) was a French mathematician and physicist born in Auxerre and best known for initiating the investigation of Fourier series and their applications to problems of heat transfer and vibrations. The Fourier transform and Fourier's law are also named in his honour. Fourier is also generally credited with the discovery of the greenhouse effect.

Property	Value
dbo:abstract	▪ Jean Baptiste Joseph Fourier (* 21. März 1768 bei Auxerre; † 16. Mai 1830 in Paris) war ein französischer Mathematiker und Physiker. (de) ▪ Jean-Baptiste Joseph Fourier - (/ˈfʊəriˌeɪ, -iər/; French: [fuʁje]; 21 March 1768 – 16 May 1830) was a French mathematician and physicist born in Auxerre and best known for initiating the investigation of Fourier series and their applications to problems of heat transfer and vibrations. The Fourier transform and Fourier's law are also named in his honour. Fourier is also generally credited with the discovery of the greenhouse effect. (en)
dbo:academicAdvisor	▪ dbr:Joseph-Louis_Lagrange
dbo:almaMater	▪ dbr:École_Normale
dbo:birthDate	▪ 1768-03-21 (xsd:date) ▪ 1768-3-21
dbo:birthPlace	▪ dbr:Burgundy_(region) ▪ dbr:Kingdom_of_France

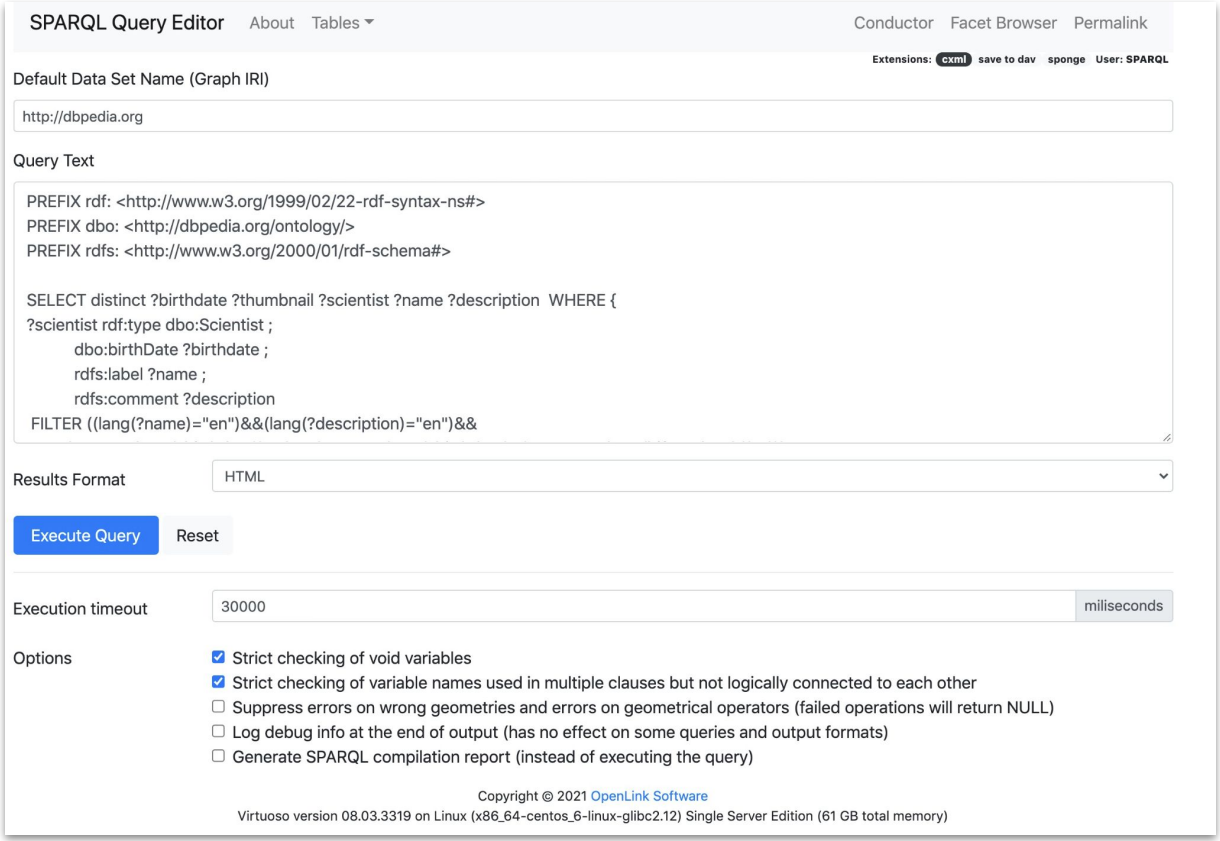
http://dbpedia.org/page/Joseph_Fourier

Prerequisites - (Manual) Data Analysis

- Data Analysis via **SPARQL**
  - What kind of entities are you looking for?
    - `?scientist rdf:type dbo:Scientist .`
  - What information do you need?
    - `?scientist dbo:birthDate ?birthdate .`
    - `?scientist rdfs:label ?name .`
    - `?scientist rdfs:comment ?description .`
    - `OPTIONAL { ?scientist dbo:thumbnail ?thumbnail }`
  - Any filter criteria?
    - `(lang(?name)="en") && (lang(?description)="en")`
    - `(SUBSTR(STR(?birthdate),6)="07-03")`
    - More sophisticated: `(SUBSTR(STR(bif:curdate('')),6)`

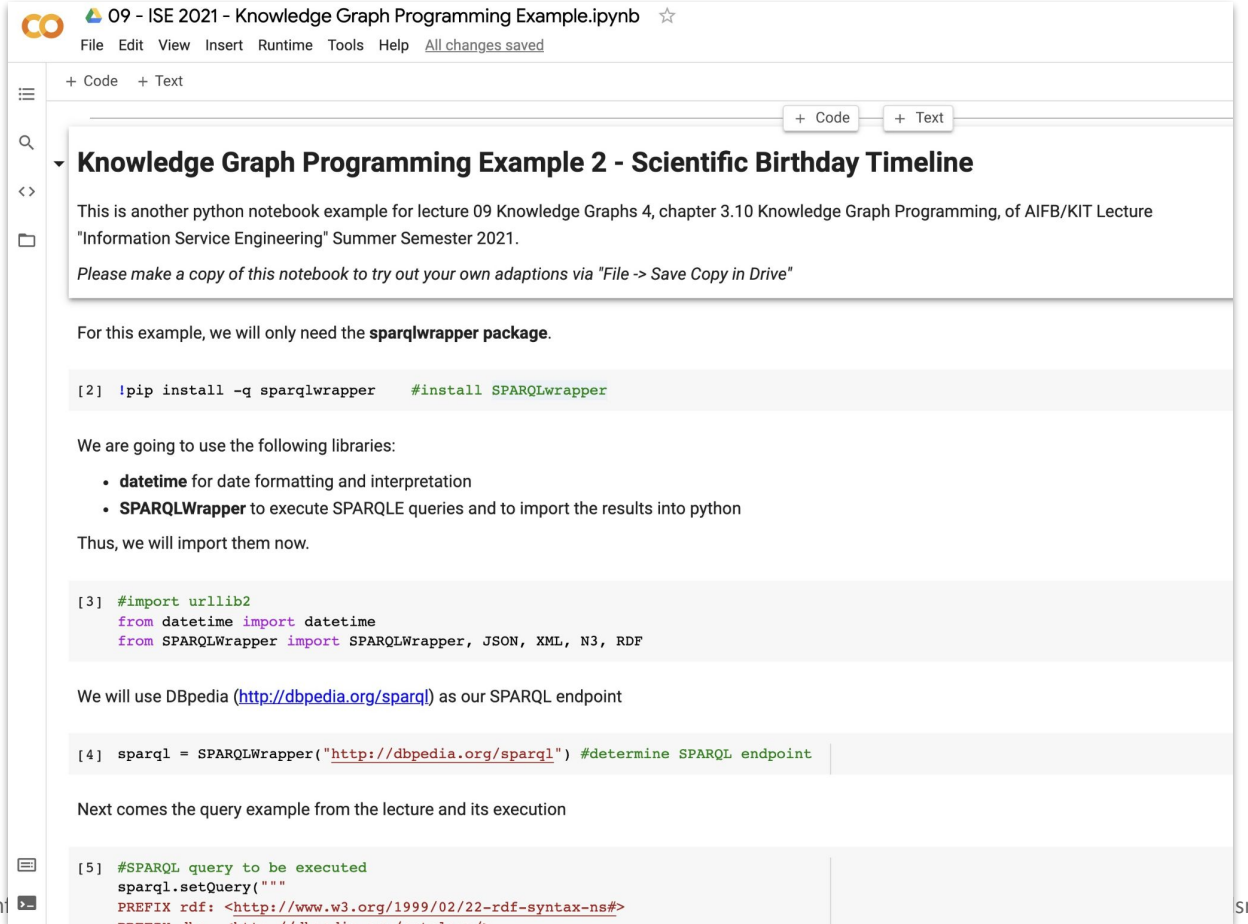
Virtuoso triple store builtin function for current date

Prerequisites - SPARQL Queries



SPARQL query at dbpedia.org

Python Code in a Collaborative Notebook



09 - ISE 2021 - Knowledge Graph Programming Example.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

+ Code + Text

Knowledge Graph Programming Example 2 - Scientific Birthday Timeline

This is another python notebook example for lecture 09 Knowledge Graphs 4, chapter 3.10 Knowledge Graph Programming, of AIFB/KIT Lecture "Information Service Engineering" Summer Semester 2021.

Please make a copy of this notebook to try out your own adaptions via "File -> Save Copy in Drive"

For this example, we will only need the **sparqlwrapper package**.

```
[2] !pip install -q sparqlwrapper    #install SPARQLwrapper
```

We are going to use the following libraries:

- **datetime** for date formatting and interpretation
- **SPARQLWrapper** to execute SPARQLE queries and to import the results into python

Thus, we will import them now.

```
[3] #import urllib2
    from datetime import datetime
    from SPARQLWrapper import SPARQLWrapper, JSON, XML, N3, RDF
```

We will use DBpedia (http://dbpedia.org/sparql) as our SPARQL endpoint

```
[4] sparql = SPARQLWrapper("http://dbpedia.org/sparql") #determine SPARQL endpoint
```

Next comes the query example from the lecture and its execution

```
[5] #SPARQL query to be executed
    sparql.setQuery("""
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

Link to collaborative Notebook

Knowledge Graph Programming Example

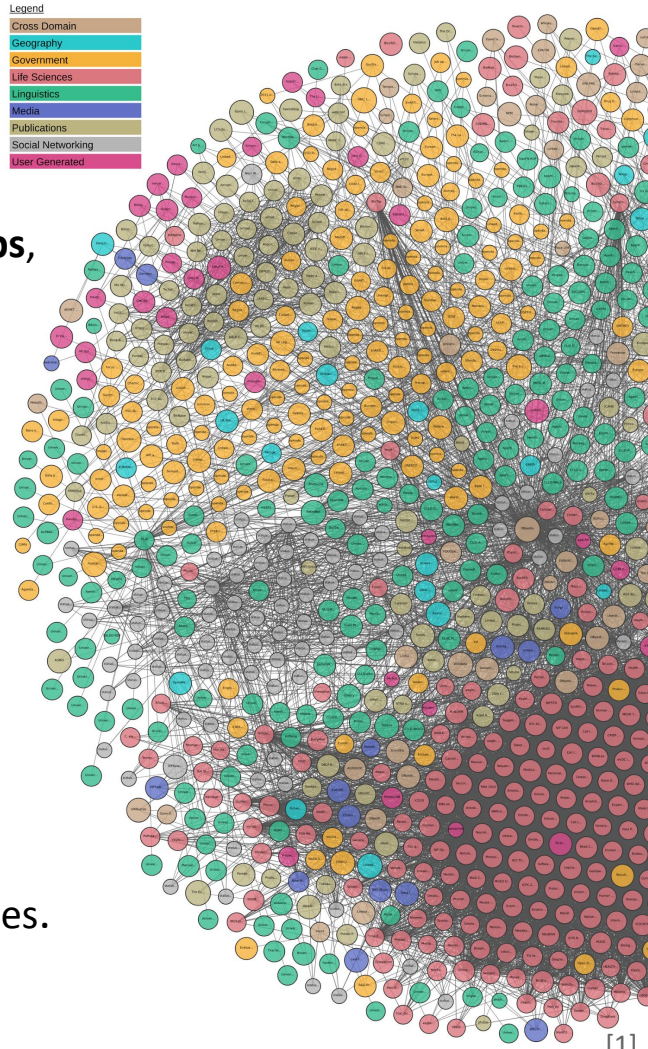
Scientific Birthdays of Wednesday May 19

- **1773** --  Arthur Aikin, Arthur Aikin, FLS, FGS (19 May 1773 – 15 April 1854) was an English chemist, mineralogist and scientific writer, and was a founding member of the Chemical Society (now the Royal Society of Chemistry). He first became its Treasurer in 1841, and later became the Society's second President.
- **1832** --  Edmond Bour, Jacques Edmond Émile Bour (French: [buʁ]; 19 May 1832 – 8 March 1866) was a French engineer famous for the . His parents were Joseph Bour and Gabrielle Jeunet. He was a student at l'École Polytechnique and graduated at the top of his class in 1852. After teaching for a year as a professor at l'École des Mines de Saint-Étienne, he became a professor of engineering at l'École Polytechnique. In 1858 he obtained the grand prize in mathematics from the Académie des Sciences for his treatise on L'intégration des équations aux dérivées partielles des premier et deuxième degrés.
- **1841** --  Adolf Ferdinand Weinhold, Adolf Ferdinand Weinhold (19 May 1841 – 1 July 1917) was a German chemist, physician and inventor.
- **1843** --  Theodor Zincke, Ernst Carl Theodor Zincke (19 May 1843 – 17 March 1928) was a German chemist and the academic adviser of Otto Hahn.
- **1846** --  Otto Beck, Otto Beck (30 March 1846 – 19 May 1908) was a German politician, and former senior mayor for Mannheim from 1891 to 1908. Moreover, by having founded the Handelshochschule Mannheim together with Heinrich Gothein in 1907, he is regarded as one of the founding fathers of the University of Mannheim.
- **1857** --  John Jacob Abel, John Jacob Abel (19 May 1857 – 26 May 1938) was an American biochemist and pharmacologist. He established the pharmacology department at Johns Hopkins University School of Medicine in 1893, and then became America's first full-time professor of pharmacology. During his time at Hopkins, he made several important medical advancements, especially in the field of hormone extraction. In addition to his laboratory work, he founded several significant scientific journals such as the Journal of Biological Chemistry and the Journal of Pharmacology and Experimental Therapeutics.
- **1864** --  Carl Akeley, Carl Ethan Akeley (May 19, 1864 – November 17, 1926) was a pioneering American taxidermist, sculptor, biologist, conservationist, inventor, and nature photographer, noted for his contributions to American museums, most notably to the Field Museum of Natural History and the American Museum of Natural History. He is considered the father of modern taxidermy. He was the founder of the AMNH Exhibitions Lab, the interdisciplinary department that fuses scientific research with immersive design.
- **1865** --  Flora Philip, Flora Philip (19 May 1865 – 14 August 1943) was a Scottish mathematician, one of the first women to receive a degree from the University of Edinburgh and the first female member of the Edinburgh Mathematical Society.
- **1868** --  John Fillmore Hayford, John Fillmore Hayford (May 19, 1868 – March 10, 1925) was an eminent United States geodesist. His work involved the study of isostasy and the construction of a reference ellipsoid for approximating the figure of the Earth. The crater Hayford on the far side of the Moon is named after him. Mount Hayford, a 1,871 m mountain peak near Metlakatla, Alaska, United States, is named after him. A biography of Hayford may be found in the Biographical Memoirs of the National Academy of Sciences, 16 (5), 1935.
- **1869** --  Hans Theodor Bucherer, Hans Theodor Bucherer (19 May 1869 – 29 May 1949) was a German chemist and gave name to several chemical reactions, for example the Bucherer carbazole synthesis, the Bucherer reaction, and the Bucherer–Bergs reaction
- **1869** --  Henry Horatio Dixon, Henry Horatio Dixon FRS (May 19, 1869, Dublin – December 20, 1953, Dublin) was a plant biologist and professor at Trinity College Dublin. Along with John Joly, he put forward the cohesion-tension theory of water and mineral movement in plants. In 1907 he married Dorothea Mary, daughter of Sir John H Franks, with whom he raised three sons. He was the father of Hal Dixon and grandfather of Adrian Dixon, Joly Dixon and Ruth Dixon. In 1908 he was elected a Fellow of the

Information Service Engineering

Lecture 9: Knowledge Graphs - 4

Brief Knowledge Graph Recap



- A **Graph** consisting of **concepts**, **classes**, **properties**, **relationships**, and **entity descriptions**.
- Based on **formal knowledge representations** (RDF(S), OWL).
- Data can be **open** (e.g. DBpedia, WikiData), **private** (e.g. supply chain data), or **closed** (e.g. product models).
- Data can be **original**, **derived**, or **aggregated**.
- We distinguish
  - **instance data** (ground truth),
  - **schema data** (vocabularies, ontologies),
  - **metadata** (e.g. provenance, versioning, licensing).
- **Taxonomies** are used to categorize entities.
- **Links** exist between internal and external data.
- Including **mappings** to data stored in other systems and databases.
- *Fully compliant to **FAIR Data principles***.

The Linked Open Data Cloud from lod-cloud.net

[1]

Knowledge Base Definition

A Knowledge Graph is a **Knowledge Base** that is a Graph.

A **knowledge base** (**KB**) is a technology used to store complex structured and unstructured information used by a computer system. The initial use of the term was in connection with expert systems which were the first knowledge-based systems.

Wikipedia

knowledge base

Free Online Dictionary of Computing

‹*artificial intelligence*›

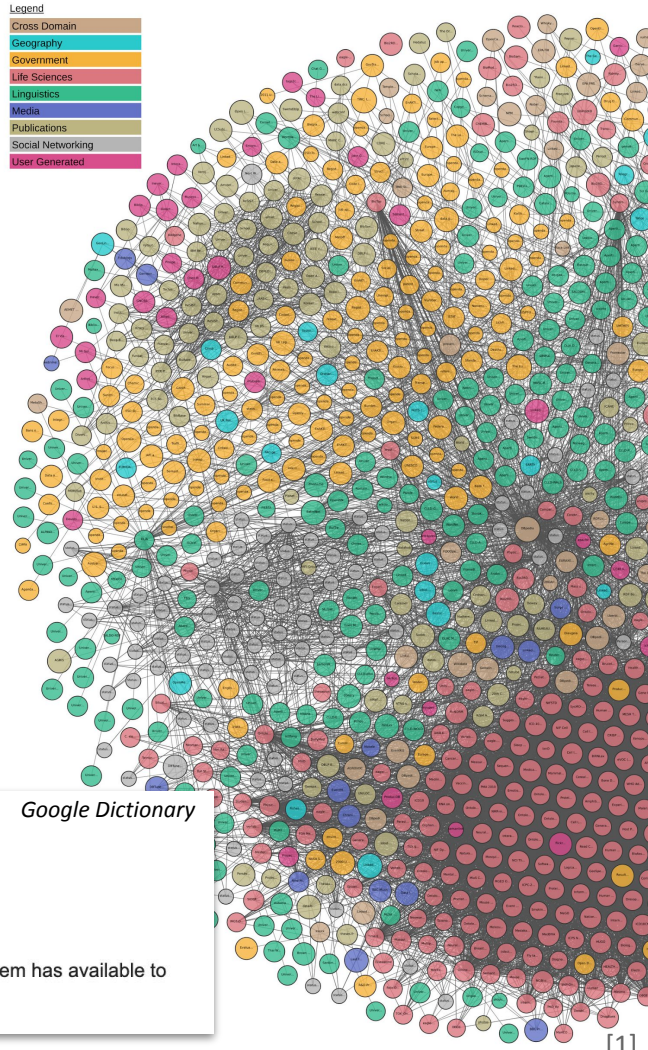
A collection of knowledge expressed using some formal knowledge representation language. A knowledge base forms part of a knowledge-based system (KBS).

knowledge base

Google Dictionary

noun

1. a store of information or data that is available to draw on.
2. the underlying set of facts, assumptions, and rules which a computer system has available to solve a problem.



[1]

Graph Definition

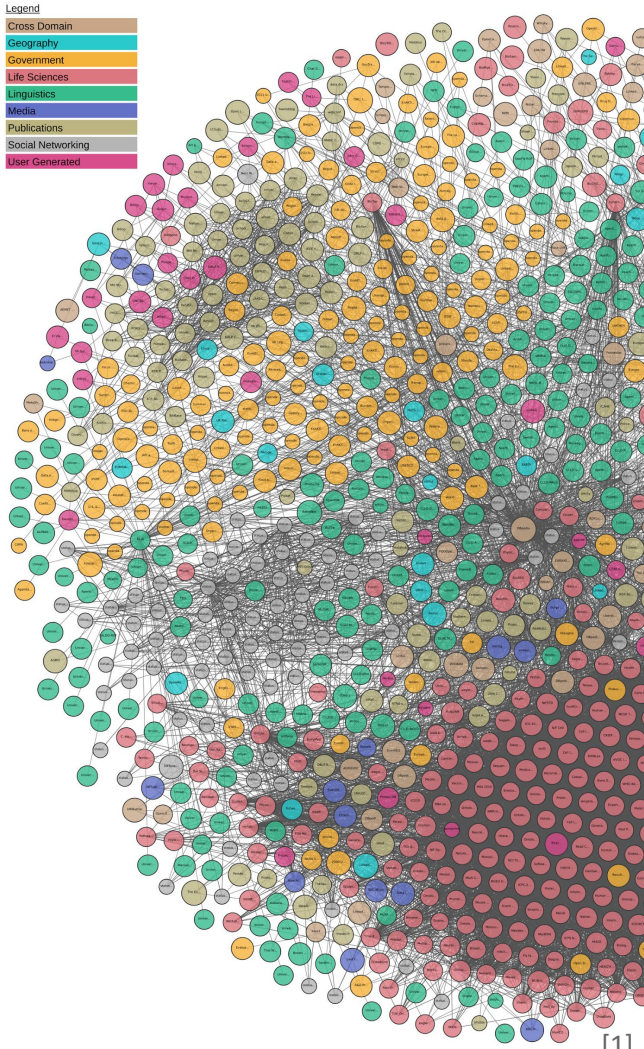
A Knowledge Graph is a Knowledge Base that is a **Graph**.

Definition **1.1**

A **simple directed graph** **G=(V,E)** consists of a set **V** of **vertices**, |V|=n, and a set **E** of **directed edges**, $E \subseteq V \times V$, where each edge $e_i=(v_k, v_l)$, $e_i \in E$ is an ordered pair of two vertices (v_k, v_l) with $v_k, v_l \in V$.

Definition 1.2

- A **graph with self-loops** is a graph extended with the option of having edges that relate a vertex to itself.
- A **multi-graph** is a graph that may have multiple edges with the same vertices.
- An **edge-labelled graph** is a graph that has an additional **labelling function** $\lambda : E \rightarrow L$ that maps each edge in E to an element in a set of labels L (similarly for vertex-labelled graphs).



[1]

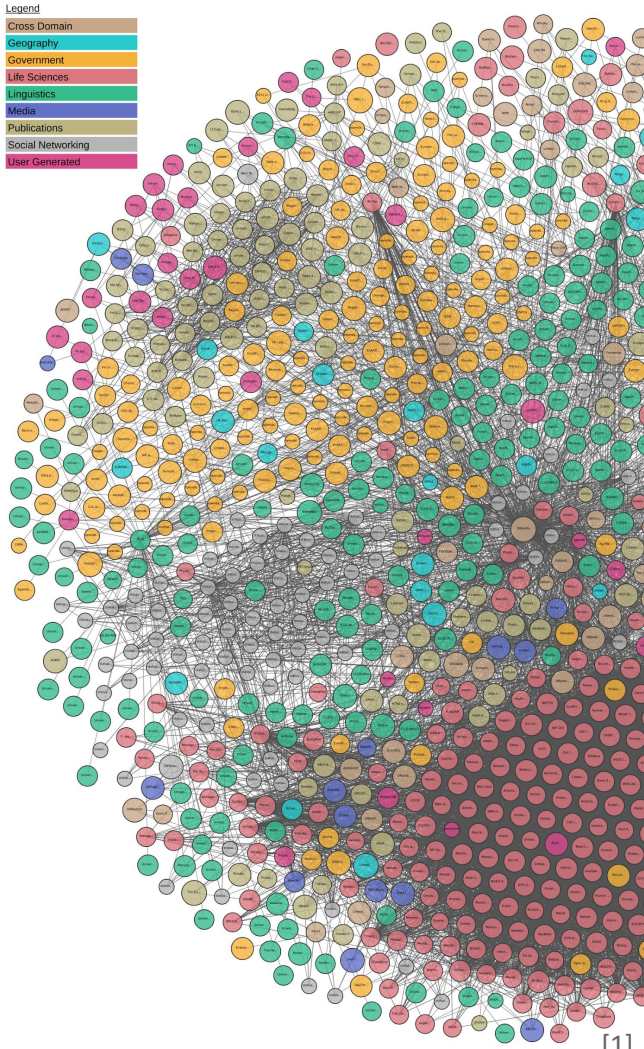
Graph Definition (cont.)

Definition 1.3

- An edge is said to be **incidental** to the vertices it connects.
- The **degree** of a vertex is the number of edges that are incidental to it.
- In a directed graph, the **in-degree** of a vertex is the number of edges pointing towards it; analogously for **out-degree**.

Definition 1.4

- A **directed path** in a directed graph is a sequence of consecutive edges $(e_1, e_2, \ldots, e_n)$ with $e_i=(v_l,v_k)$ and $e_{i+1}=(v_k,v_m)$.
- A directed graph is **strongly connected** if there is a directed path from any vertex to any other vertex.



[1]

How Can You Characterize a Knowledge Graph?

"Should I use Knowledge Graph A or Knowledge Graph B to solve my problem?"

- How to compare two Knowledge Graphs?
  - Size
  - Coverage
  - Completeness
  - Level of Detail
  - Accuracy
  - Reliability
  - etc.

- **Idea**: **Structural Comparison** by just comparing the Graphs

Graph Centrality Measures

- **Network analysis** has developed methods for **finding the most important vertices in a graph**.
- **Vertex importance** based on the structure of such graphs is called **centrality.**

- But, what makes a node important?

What makes a Node important?

- Many networks can be considered to describe a **flow** of something (goods, information, etc.).
- A node might be **important**, if
  - a lot flows from it (in a supply chain),
  - to it (in a network of links), or
  - through it (in a communication network).
- **Flow** might be modelled by **(weighted) paths**, possibly factoring in their length and/or number.
- **Paths** might be more important if they pass through **important nodes**.
- In knowledge graphs, the importance of edges and nodes may also depend on **more complex features** (e.g., edge or vertex labels).

What makes a Node important?

- **Wikidata Example:**
  - A Wikidata entity (node) might be important, if it is referenced by many Wikipedia pages.
  - what are the most important Climatologists?

```
SELECT ?climatologistLabel (SUM(?link) AS ?importance)
WHERE {
  ?climatologist wdt:P106 wd:Q1113838 .
  ?climatologist wikibase:sitelinks ?link.
  ?climatologist rdfs:label ?climatologistLabel
   FILTER (lang(?climatologistLabel)="en")
} GROUP BY ?climatologistLabel
ORDER BY DESC(?importance)
```

Wikidata Query Service | Examples | Help | More tools | English

```
SELECT ?climatologistLabel (SUM(?link) AS ?importance)
WHERE {
  ?climatologist wdt:P106 wd:Q1113838 .
  ?climatologist wikibase:sitelinks ?link.
  ?climatologist rdfs:label ?climatologistLabel FILTER (lang(?climatologistLabel)="en")
} GROUP BY ?climatologistLabel
ORDER BY DESC(?importance)
```

SPARQL query

256 results in 550 ms | Code | Download | Link

climatologistLabel	importance
Alexander von Humboldt	119
Paul Josef Crutzen	56
Wladimir Köppen	49
Piers Sellers	27
Andres Tarand	26
Eunice Newton Foote	20
Judith Curry	18
Lučka Kajfež Bogataj	17

Degree Centrality

- A simple form of centrality restricts to incoming/outgoing paths of length one.

Definition 1.5

- The **in-degree centrality** of a directed graph is given by the in-degree of each node.
- The **out-degree centrality** and the **degree centrality** (for undirected graphs) are defined analogously.

- There are more sophisticated forms of centrality, as e.g.
  - Eigenvector centrality, Katz centrality, PageRank, etc.

Further Centrality Measures

- Further measures to characterize a Knowledge Graph:
  - **Sizes**
    - number of nodes
    - number of facts
    - avg number of facts per node
  - **KG diameter**

Definition 1.6

- The **eccentricity** of a node is the maximal distance between a certain node and any other node.
- The **diameter** of a graph is the maximum **eccentricity** of a graph, i.e. the greatest distance between any pair of nodes.
- To find the diameter of a graph, first find the **shortest path** between each pair of nodes. The greatest length of any of these paths is the **diameter of the graph**.

Further Centrality Measures

- Further measures to characterize a Knowledge Graph:
  - **Sizes**
    - number of nodes
    - number of facts
    - avg number of facts per node
  - **KG diameter**
  - **KG radius**

Definition 1.8

- The **radius** of a graph is the minimum eccentricity of a graph, i.e. the shortest of the maximum distances between any pair of nodes.

Further Centrality Measures

- Further (structural) measures to characterize a Knowledge Graph:
  - Sizes
    - number of nodes
    - number of facts
    - avg number of facts per node
  - KG diameter
  - KG radius
  - avg in/out degree
  - avg path length
  - and many more...

Knowledge Graphs and Important Nodes

- In **Knowledge Graphs**, the importance of nodes might further be depending on
  - the **properties** (i.e. edge attributes),
  - the **node labels** (i.e. further attributes of nodes),
  - specific nodes or edges might be **ignored**, as e.g.
    - Basically for every entity in a (OWL encoded) knowledge graph the following fact holds:
      `:entity rdf:type owl:Thing`
    - Therefore, we might ignore this fact if we want to determine the importance of nodes.

Information Service Engineering

Lecture 9: Knowledge Graphs - 4

3. Knowledge Graphs - 4
Bibliography

- P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, S. Rudolph, *OWL 2 Web Ontology Language , Primer (Second Edition)*, W3C Recommendation, 11 December 2012, https://www.w3.org/TR/2012/REC-owl2-primer-20121211/
- Hogan, A., *The Web of Data*, *Springer, 2020*. *(available via KIT network)*
- Hausenblas, M., *Linked Data Applications*, DERI Reports, 2009.
- *The Python Tutorial*, python.org.
- R. Diestel, *Graph Theory*, *Graduate Texts in Mathematics*, Volume 173, Springer, 2016. *(available via KIT network)*

3. Knowledge Graphs - 4

Syllabus Questions

- Why is **OWL** semantically more expressive than RDFS?
- What is the difference between OWL **datatype properties** and OWL **object properties**?
- How can (complex) OWL **classes** be defined?
- What can be defined via an **OWL property constraint**?
- What is the difference between **OWL loose binding** and **strict binding**?
- How much **semantics** can be defined for **OWL datatype/object properties**?
- What are the **main components of Linked Data driven Web applications** and how do they interact?
- What is the **difference between a Knowledge Base and a Knowledge Graph**?
- How can we determine the **importance of a node in a Knowledge Graph**?