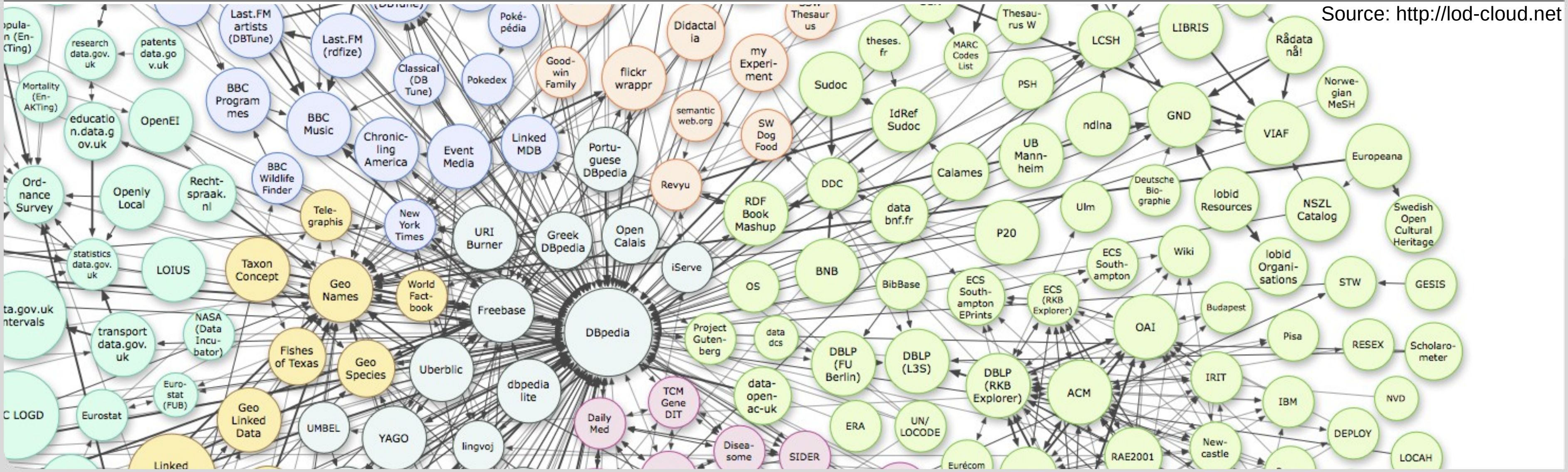
C03 The Resource Description Framework
How to represent data on the web as graphs?

Version 2021-04-25

Lecturer: Prof. Dr. Andreas Harth, FAU Erlangen-Nürnberg

CHAIR OF TECHNICAL INFORMATION SYSTEMS



Source: http://lod-cloud.net

CC - Creative Commons Licensing

- This set of slides is part of the lecture „Semantic Web Technologies“ held at Karlsruhe Institute of Technology
- The content of the lecture was prepared by Andreas Harth based on his book „Introduction to Linked Data“
- The slides were prepared by Tobias Käfer, Andreas Harth, and Lars Heling

- **This content is licensed under a Creative Commons Attribution 4.0 International license (CC BY 4.0):** http://creativecommons.org/licenses/by/4.0/

Agenda

1. **Graph-structured Data**
2. The RDF Vocabulary
3. RDF Abstract Syntax: Terms, Triples and Graphs
4. Relations Between RDF Graphs
5. Handling Multiple RDF Graphs in an RDF Dataset

Desiderata for a Standardised Data Model for Data on the Web

- Low level of surprise (=low entropy) for machines[1] and those who program them
  - The higher the entropy, the more energy needed to process information (computing power, lines of code, memory, …) [1]
- “Energy” could be put into:
  - Integrating data from different sources (merge operation, term disambiguation)
  - Writing processors
  - Validating processors
  - Running processors

- Contrast the “energy” required to process files with MIME types:
  - **`text/uri-list`**
  - **`text/plain`**
  - **`text/html`**
  - **`application/xml`**
  - **`application/json`**

[1] Cf. Mike Amundsen: “Autonomous Agents on the Web”, Keynote at the Workshop for Services an Applications over Linked Data, 2013 https://www.slideshare.net/rnewton/autonomous-agents-on-the-web-22078931

From Trees to Graphs

- XML and OEM are based on a tree structure, with a dedicated root element
- A graph-structured data model allows for merging data from multiple sources

- If the data publishers agree on a graph-structured data model, the data consumers can easily combine data from multiple publishers

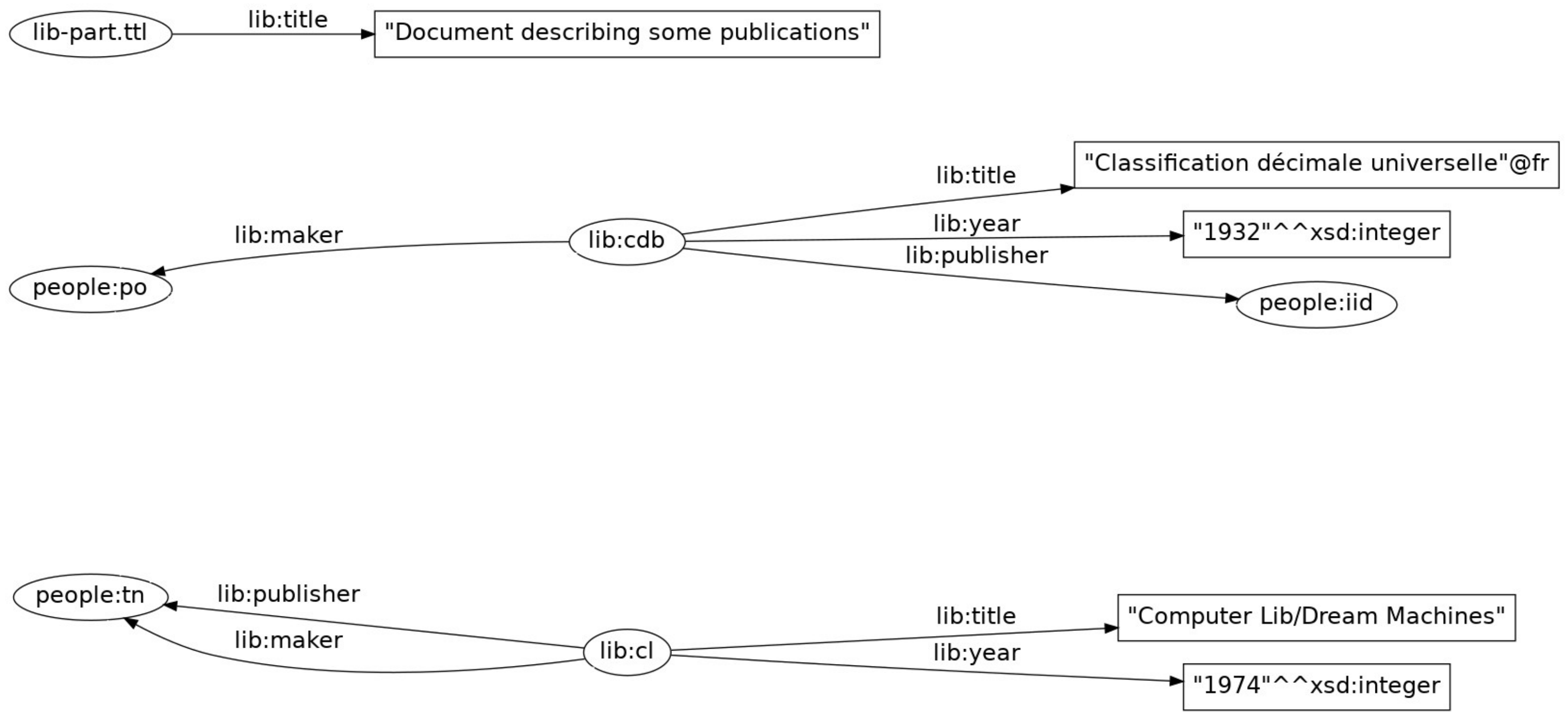
Example: Merging Graphs

- We consider three graphs
  - lib-part.ttl
  - people-part.ttl
  - topics-part.ttl

lib-part.ttl

```
@prefix : <lib-part#> .
@prefix people: <people-part#> .

<> :title "Document describing some publications" .

:cdb :title "Classification décimale universelle"@fr ;
:maker people:po ; :publisher people:iid ; :year
1932 .

:cl :title "Computer Lib/Dream Machines" ;
:publisher people:tn ; :maker people:tn ; :year 1974 .
```

lib-part.ttl as Graph

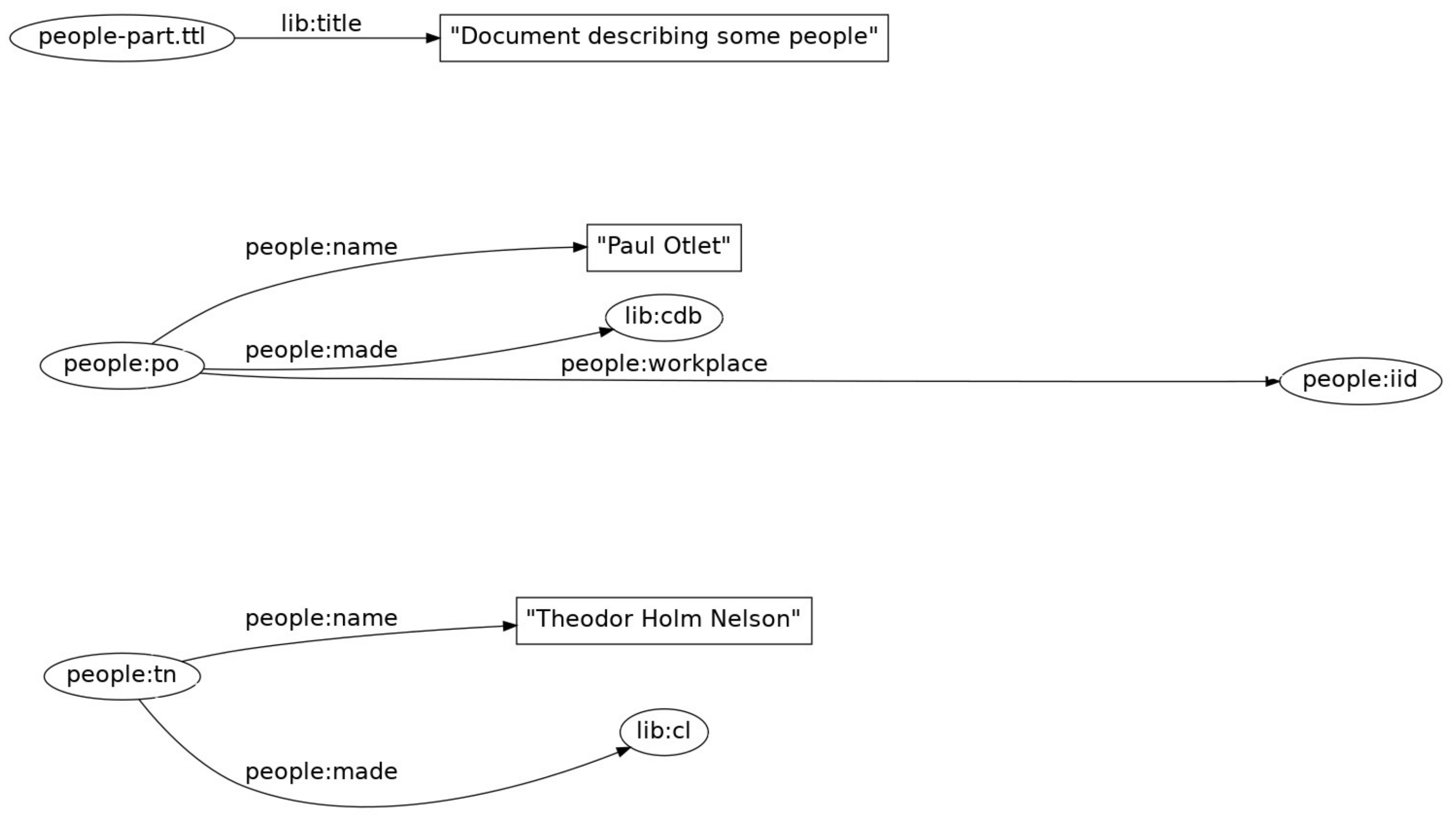


```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix people: <people-part#> .
@prefix lib: <lib-part#> .
@prefix topics: <topic-part#> .
```

people-part.ttl

```
@prefix : <people-part#> .
@prefix lib: <lib-part#> .

<> lib:title "Document describing some people" .

:po :name "Paul Otlet" ; :made
lib:cdb ; :workplace :iid .

:iid :name "Institut international de
documentation"@fr .

:tn :name "Theodor Holm Nelson" ; :made lib:cl .
```

people-part.ttl as Graph

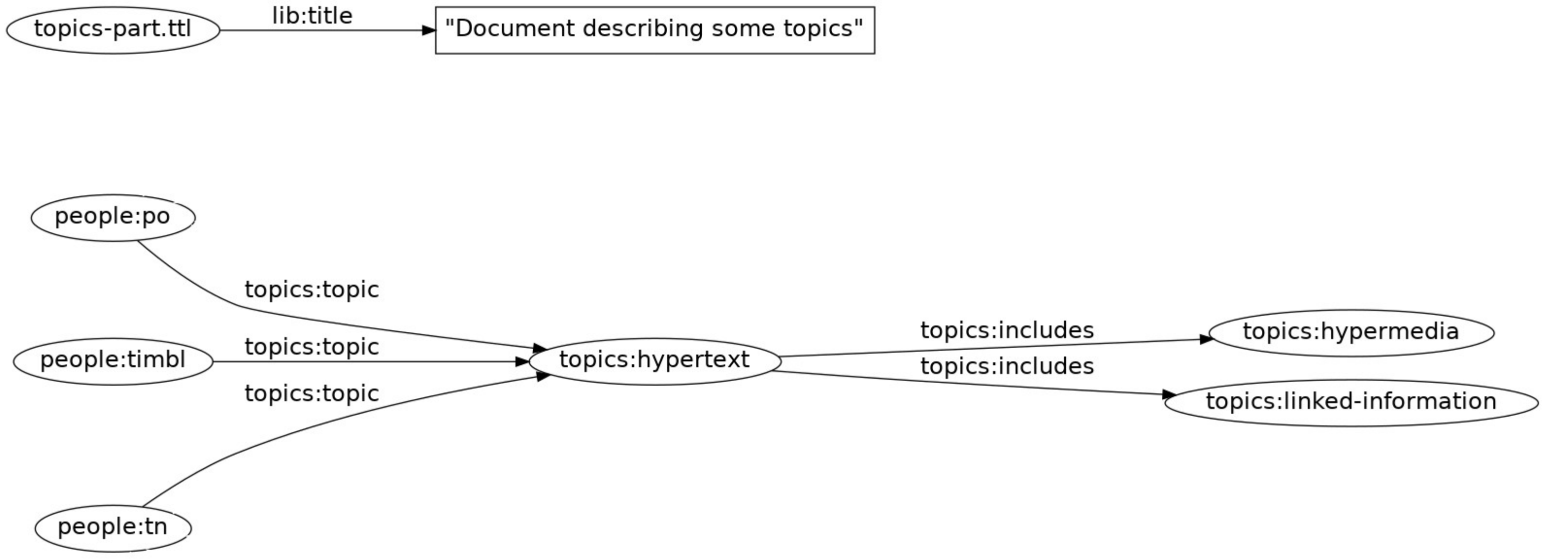


```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix people: <people-part#> .
@prefix lib: <lib-part#> .
@prefix topic: <topics-part#> .
```

topics-part.ttl

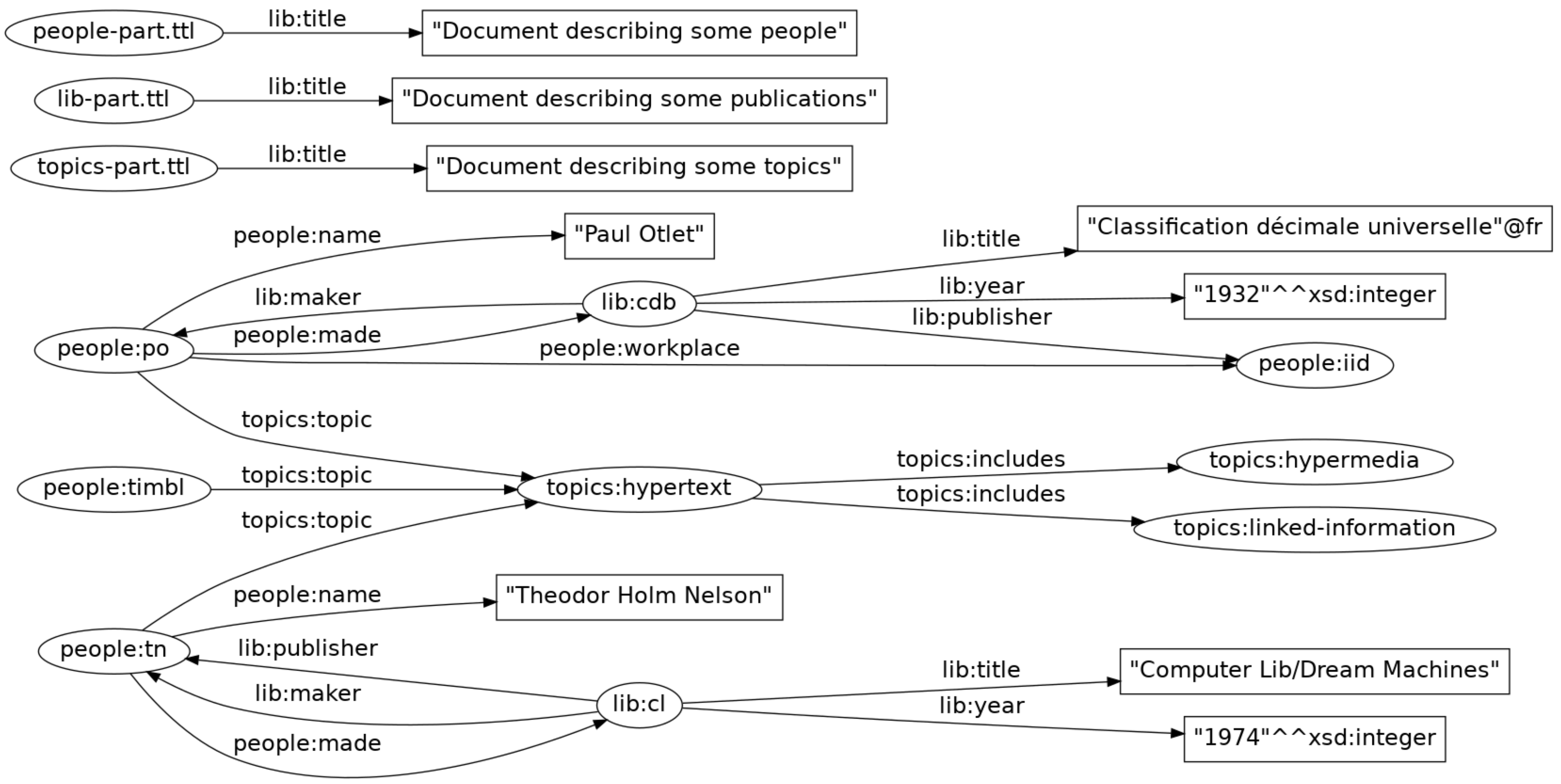
```
@prefix people: <people-part#> .
@prefix : <topics-part#> .
@prefix lib: <lib-part#> .

<> lib:title "Document describing some topics" .

people:timbl :topic :hypertext .

people:po :topic :hypertext .

people:tn :topic :hypertext .

:hypertext :includes :hypermedia , :linked-information
.
```

topics-part.ttl as Graph



```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix people: <people-part#> .
@prefix lib: <lib-part#> .
@prefix topics: <topic-part#> .
```

The Merge of lib-part.ttl, people-part.ttl and topics-part.ttl



```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix people: <people-part#> .
@prefix lib: <lib-part#> .
@prefix topic: <topic-part#> .
```

RDF/XML: Example

```
<?xml version="1.0" encoding="utf-8"?>

<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:lib="lib-part#"
  xmlns:people="people-part#"
  xmlns="topics-part#">

<rdf:Description rdf:about="topics-part#hypertext">
  <includes rdf:resource="topics-part#hypermedia"/>
  <includes rdf:resource="topics-part#linked-information"/>
</rdf:Description>

<rdf:Description rdf:about="">
  <lib:title>Document describing some topics</lib:title>
</rdf:Description>

</rdf:RDF>
```

JSON-LD: Example

```
{
  "@context": {
    "lib": "lib-part#",
    "people": "people-part#",
    "topics": "topics-part#"
  },
  "@graph": [
    {
      "@id": "",
      "lib:title": "Document describing some topics"
    },
    {
      "@id": "topics:hypertext",
      "topics:includes": [
        {
          "@id": "topics:hypermedia"
        },
        {
          "@id": "topics:linked-information"
        }
      ]
    }
  ]
}
```

Linked Data

- Linked Data uses RDF, which has a graph-structured data model
- RDF graphs can be serialised in RDF documents (in many different serialisation formats: Turtle, RDF/XML, JSON-LD and others)
- The RDF documents have URIs
- User agents can lookup URIs of RDF documents

- Per convention, the URI of a thing and the URI of the document about the thing are related
  - Hash-URIs, which provide the connection via syntactic means
  - Slash-URIs, which provide the connection via the protocol

Agenda

1. Graph-structured Data
2. **The RDF Vocabulary**
3. RDF Abstract Syntax: Terms, Triples and Graphs
4. Relations Between RDF Graphs
5. Handling Multiple RDF Graphs in an RDF Dataset

Formal Instances (rdf:type)

- The URI `rdf:type` allows to specify that a resource is an instance of a class

- For example, the following describes `:Berlin` as belonging to the class `:City`, as follows:

```
:Berlin rdf:type :City .
```

What was the shortcut for rdf:type in the Turtle syntax?

rdf:Property

- The term `rdf:Property` denotes the resource that contains as members all resources occurring on predicate position in RDF triples

- Given an RDF graph

```
:s :p :o .
```

- we can conclude

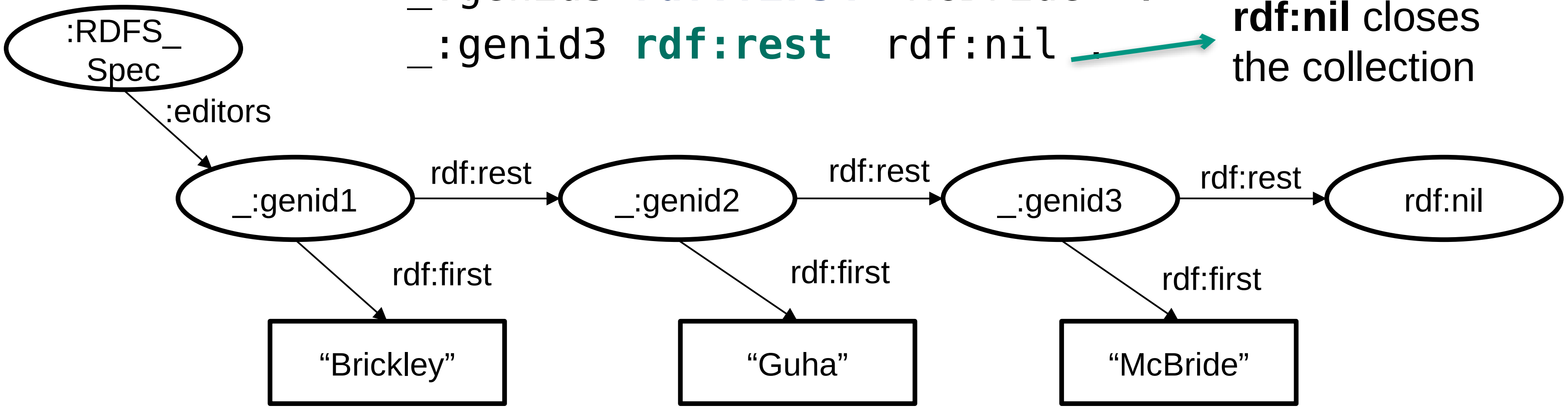
```
:p rdf:type rdf:Property .
```

RDF Collections aka rdf:Lists

- An RDF collection is a **closed group** of elements
- Example: Editors of the RDFS spec “Brickley”, “Guha”, “McBride”

```
:RDFS_Spec :editors _:genid1 .
_:genid1 rdf:first "Brickley" .
_:genid1 rdf:rest _:genid2 .
_:genid2 rdf:first "Guha" .
_:genid2 rdf:rest _:genid3 .
_:genid3 rdf:first "McBride" .
_:genid3 rdf:rest  rdf:nil .
```

rdf:nil closes the collection



RDF Lists

- Lists can only appear in subject or object position of a triple
- The class **`rdf:List`** contains the RDF lists
- Turtle provides a syntax abbreviation for specifying collections (“lists structures”) by enclosing the RDF terms with ( )

```
#the object of this triple is the RDF collection blank node
:RDFS_Spec :editors ( “Brickley” “Guha” “McBride” ) .
```

RDF Containers

- An RDF container is an **open group** of elements
  - It is possible to add more elements to the container
- Containers are not widely used and were discussed for depreciation, but are actually still in use for RDF 1.1[1]
- A container is a resource that contains RDF terms
- The contained things are called **members**
- There are three kinds of containers in RDF:
  - Bags (`rdf:Bag`)
  - Sequences (`rdf:Seq`)
  - Alternatives (`rdf:Alt`)

[1] http://lists.w3.org/Archives/Public/public-rdf-wg/2011Aug/0193.html

RDF Containers

- **`rdf:Bag`** represents an **unordered group** of RDF terms which may contain duplicates
- **`rdf:Seq`** represents an **ordered group** of RDF terms which may contain duplicates
- **`rdf:Alt`** can represent a group of RDF terms, and is "used conventionally to indicate to a human reader that typical processing will be to **select one of the members** of the container[1]"

- The properties **`rdf:_1`**, **`rdf:_2`**, **`rdf_3`**... are instances of the `rdfs:ContainerMembershipProperty`, and state that an RDF term is a member of a container
- Please observe that there is no equivalent to a membership property for lists

[1] http://www.w3.org/TR/rdf-schema/#ch_alt

RDF Containers

- The Turtle syntax does not provide a special syntax for containers
- The following Turtle document represents an `rdf:Bag` with things that the monkey likes. Note that in contrast to RDF list, this container is not closed.

```
:Monkey :likes _:genid1 .
_:genid1 rdf:type rdf:Bag ;
         rdf:_1 "apple" ;
         rdf:_2 "banana" ;
         rdf:_3 "orange" .
```

Reification

- From Latin res "thing" + facere "to make", reification can be loosely translated as thing-making; the turning of something abstract into a concrete thing or object[1]
- Reification is a construct **to use a triple in another triple** on subject or object position
- For example, consider the statement

  “:bob says that :i is of type :Person. “
- To be able to reference a triple in another one, while staying inside the triple data model, we use terms specified in the RDF vocabulary, namely `rdf:subject, rdf:predicate` and `rdf:object`

[1] https://en.wikipedia.org/wiki/Reification_%28fallacy%29#Etymology

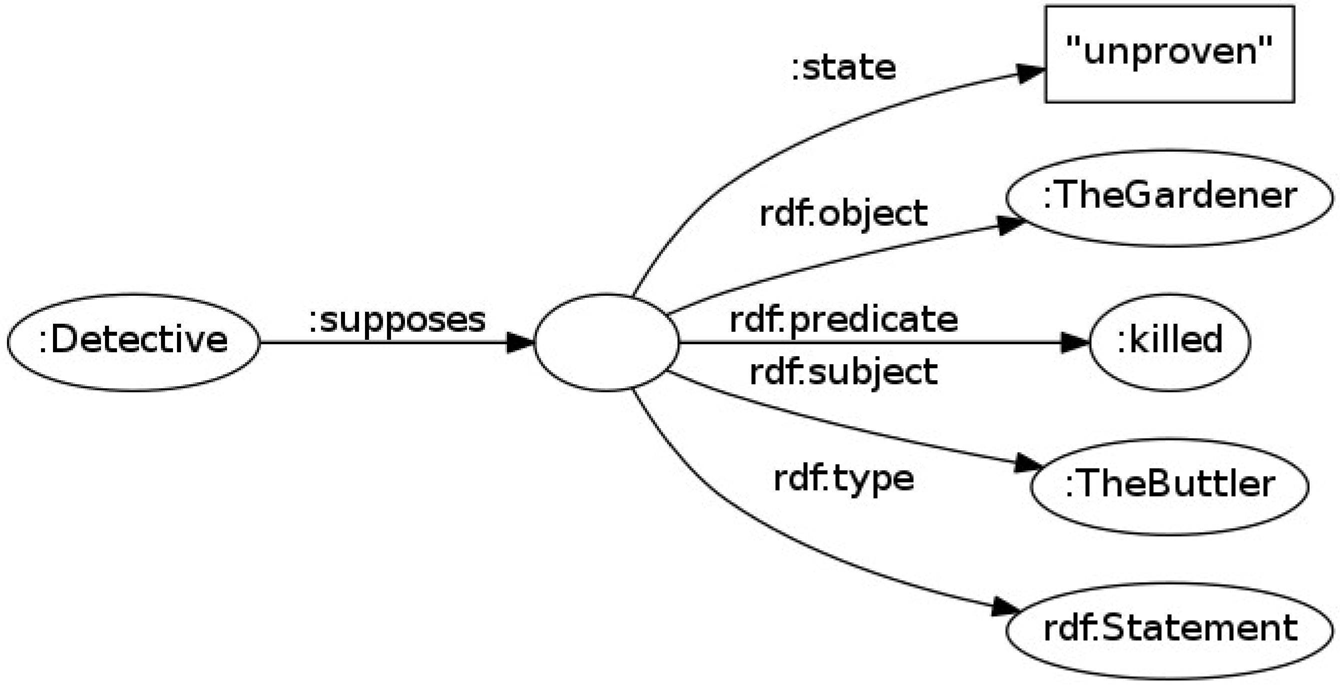
Reification: Example

“:bob says that :i is of type :Person. “

```
:bob :says _:bn .
_:bn rdf:subject :i .
_:bn rdf:predicate rdf:type .
_:bn rdf:object :Person .
_:bn rdf:type rdf:Statement .
```

- **Attention:**
  querying for “all resources of rdf:type :Person” will not deliver :i !
- Reified triples are members of the class `rdf:Statement`, but the reified triples are themselves not asserted.
- In other words, to include the triple `:i rdf:type :Person` in the RDF graph, one has to explicitly write that triple.

Another Example of Reification



n-ary Relations and rdf:value

- n-ary relations represent relations between more than two resources
- One way of modeling n-ary relations in RDF is using blank nodes
  - Example:

```
:AIFB :address [ :streetAndNumber “Kaiserstr. 89”;
                 :postalCode “76135”;
                 :city :Karlsruhe ]
```

- Note that in the previous example, none of the individual parts of the n-ary relation can be considered the “main” value
- Now consider the following example:

$$\texttt{[:price [}\underbrace{\texttt{rdf:value “20”}}_{\textbf{Main value}}\texttt{ ; }\underbrace{\texttt{:currency :EUR}}_{\textbf{Complement}}\texttt{]] .}$$

- The property `rdf:value` relates a value to a resource[1] and can be used to represent the “main” value in an n-ary relation

[1] For the varied history of `rdf:value` see https://lists.w3.org/Archives/Public/semantic-web/2010Jul/0252.html

Datatype URIs in RDF

- `rdf:langString` – datatype of language-tagged string values
- `rdf:HTML` – datatype of RDF literals storing fragments of HTML content
- `rdf:XMLLiteral` – datatype of XML literal values
- `rdf:PlainLiteral` – class of plain (i.e. untyped) literal values, as used in RIF and OWL 2

Summary of RDF Vocabulary Terms

- The following table lists all RDF terms, other than the container membership properties `rdf:_1, rdf:_2, rdf:_3 ...`

Class URIs	Property URIs	Datatype URIs	Instance URIs
rdf:Property	rdf:type	rdf:langString	rdf:nil
rdf:List	rdf:first	rdf:HTML	
rdf:Bag	rdf:rest	rdf:XMLLiteral	
rdf:Alt	rdf:value	rdf:PlainLiteral	
rdf:Seq	rdf:subject		
rdf:Statement	rdf:predicate		
	rdf:object		

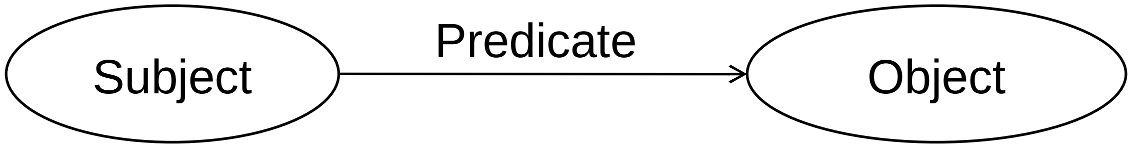
Agenda

1. Graph-structured Data
2. The RDF Vocabulary
3. **RDF Abstract Syntax: Terms, Triples and Graphs**
4. Relations Between RDF Graphs
5. Handling Multiple RDF Graphs in an RDF Dataset

Resource Description Framework (RDF)

[1]

- RDF is the foundational data format for both Semantic Web and Linked Data
- An RDF triple is the basic RDF concept describing information as a subject-property-object structure
- Property (or predicate) specifies relation between subject and object
- Triples can be viewed graphically:

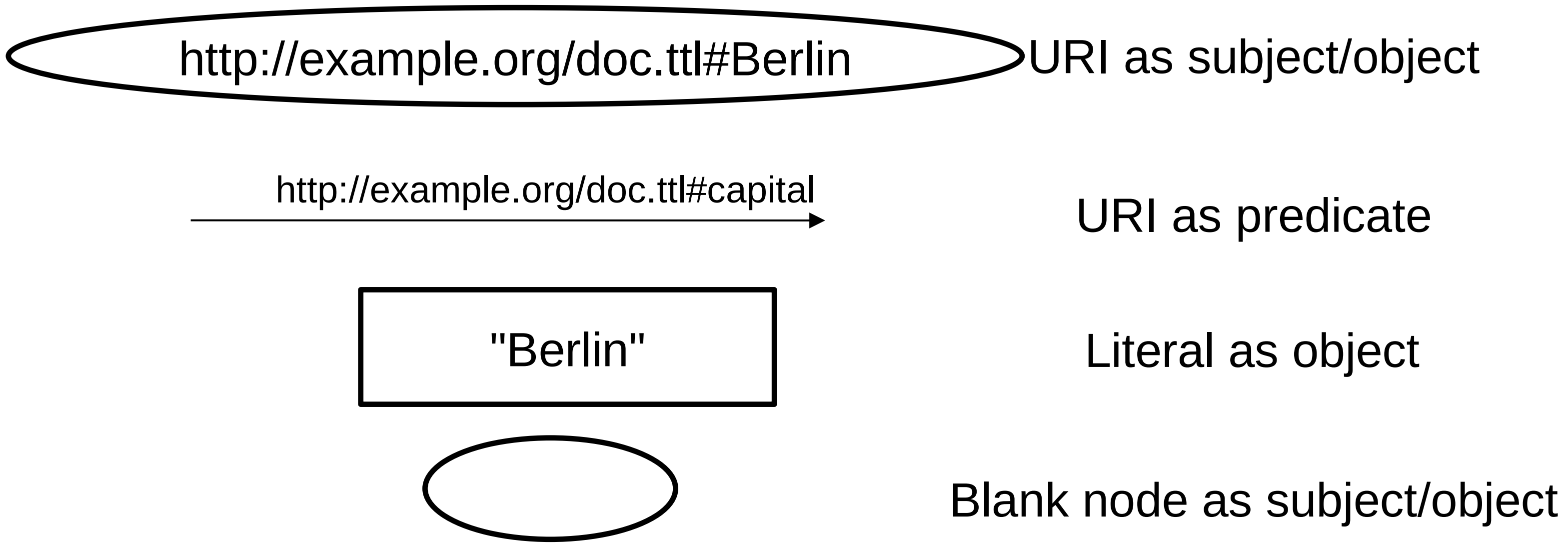


- RDF graphs can be presented as directed labelled graph

[1] http://www.w3.org/RDF/icons/

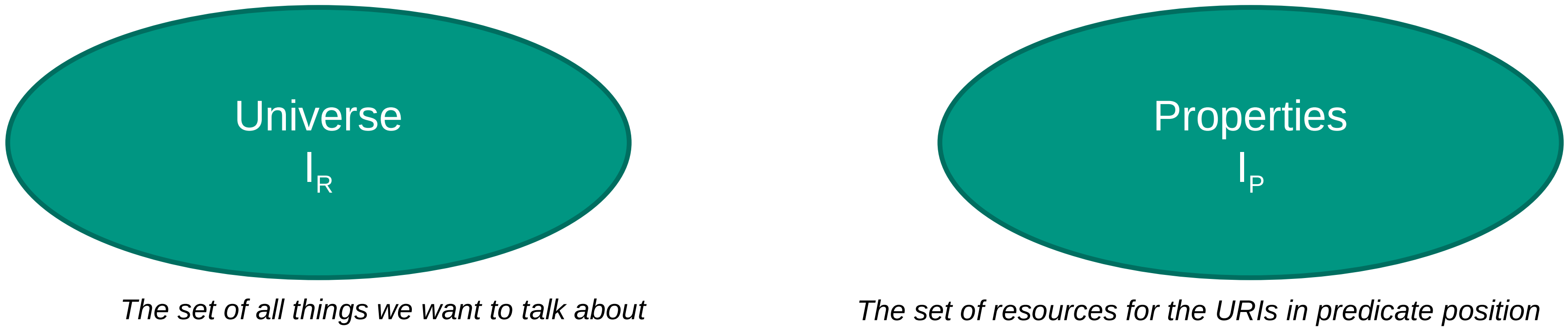
RDF Term Overview: URIs - Blank Nodes - Literals

- **URIs** are used to globally identify resources
- **Blank nodes** refer to resources, too, but these resources can only be identified within a file and are not globally addressable (later more)
- **Literals** refer to concrete data values such as strings, integers, floats or dates. In RDF, we can use the datatypes defined as part of the XML Schema recommendation


http://example.org/doc.ttl#Berlin
URI as subject/object
http://example.org/doc.ttl#capital
URI as predicate
"Berlin"
Literal as object
Blank node as subject/object

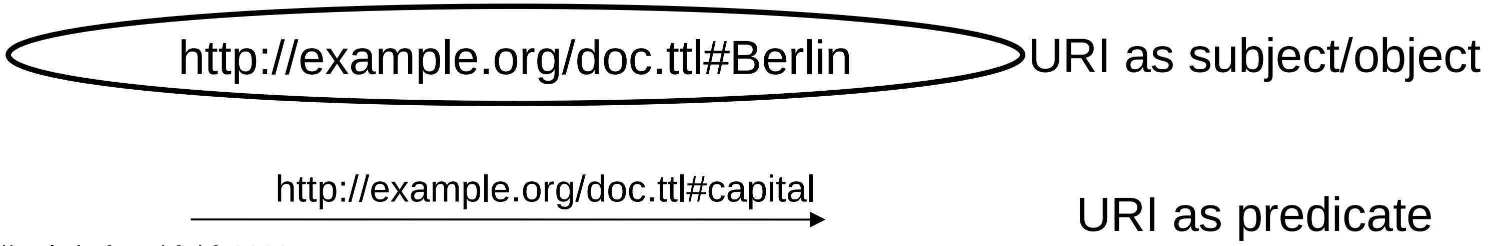
Domain of Discourse

- Characterisation (Resource, Property, Universe): *A resource is a notion for things of discourse, be they abstract or concrete, physical or virtual. We write I_R for the set of resources, also called the universe or domain. We write I_P for the set of property resources.*


Universe
I_R
Properties
I_P
The set of all things we want to talk about
The set of resources for the URIs in predicate position

URIs

- We use URIs as identifiers
- Full URIs
  - Start with a scheme
  - Example: http://example.org/doc.ttl#Berlin
- CURIEs
  - Allow for abbreviating URIs
  - Example: with doc: being short for http://example.org/doc.ttl#, we can write doc:Berlin for the URI from the full URI example
- IRIs
  - Standard to allow for using characters outside of US-ASCII in URIs
  - We typically use URI and IRI interchangably


http://example.org/doc.ttl#Berlin
URI as subject/object
http://example.org/doc.ttl#capital
URI as predicate

https://tools.ietf.org/rfc/rfc3986.txt

URIs (cont‘d)

- Hierarchical URIs:
  - HTTP URIs are hierarchical in the path part of the URI
  - Example: http://example.org/path/to/resource
- Relative URIs
  - With hierarchical URIs you can have relative URIs that traverse the path
  - Example:../../relative/../path/to/resource
- Reference Resolution
  - Relative URIs can be converted to absolute URIs by resolving them
  - Example: resolving ../../relative/../path/to/resource against http://example.org/path/to/resource yields the same URI
- Hash URIs
  - In Linked Data, we make the difference between a thing and the document about the thing. One way of expressing the difference is to use hash URIs
  - Example: http://example.org/document#thing
- Slash URIs
  - Another way of making the difference is to use slash URIs for the thing and then use HTTP redirection to the document
  - Example: http://example.org/things/thing redirects (303) to http://example.org/data/thing which in turn serves RDF

RDF Literals

- Kinds of Literals:
  - Simple Literals
  - Language-tagged Literals
    - BCP47 language tags
  - Typed literals
    - All literals have an (implicit) datatype ⇒ literals are pairs
      - For simple literals: `xsd:string`
      - For language-tagged literals: `rdf:langString` ⇒ triples
    - Eg. XML Schema datatypes
- Lexical forms and value space
- *Term Equality* of two Literals:
  - Need to be equal, character by character:
    - Lexical forms
    - Datatype IRIs
    - Language tags (if any)

⇒ Two literals can have the same value without being the same RDF term.

"Berlin"

Literal as object

Simple Literal

"Berlin"@de

Language-tagged Literal

"1"^^xsd:integer

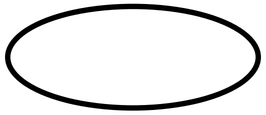
"01"^^xsd:integer

Two typed literals with different lexical forms denoting the same value

https://www.w3.org/TR/rdf11-concepts/#section-Graph-Literal http://tools.ietf.org/html/bcp47

Blank Nodes

- Blank nodes say that something […] exists, without explicitly naming it
- Blank nodes *denote* resources
- Blank nodes do not *identify* resources
- Blank nodes do not have identifiers in the RDF abstractly speaking (see depiction below)
- In *implementations and serialisations*, blank nodes have identifiers (which are only locally scoped)



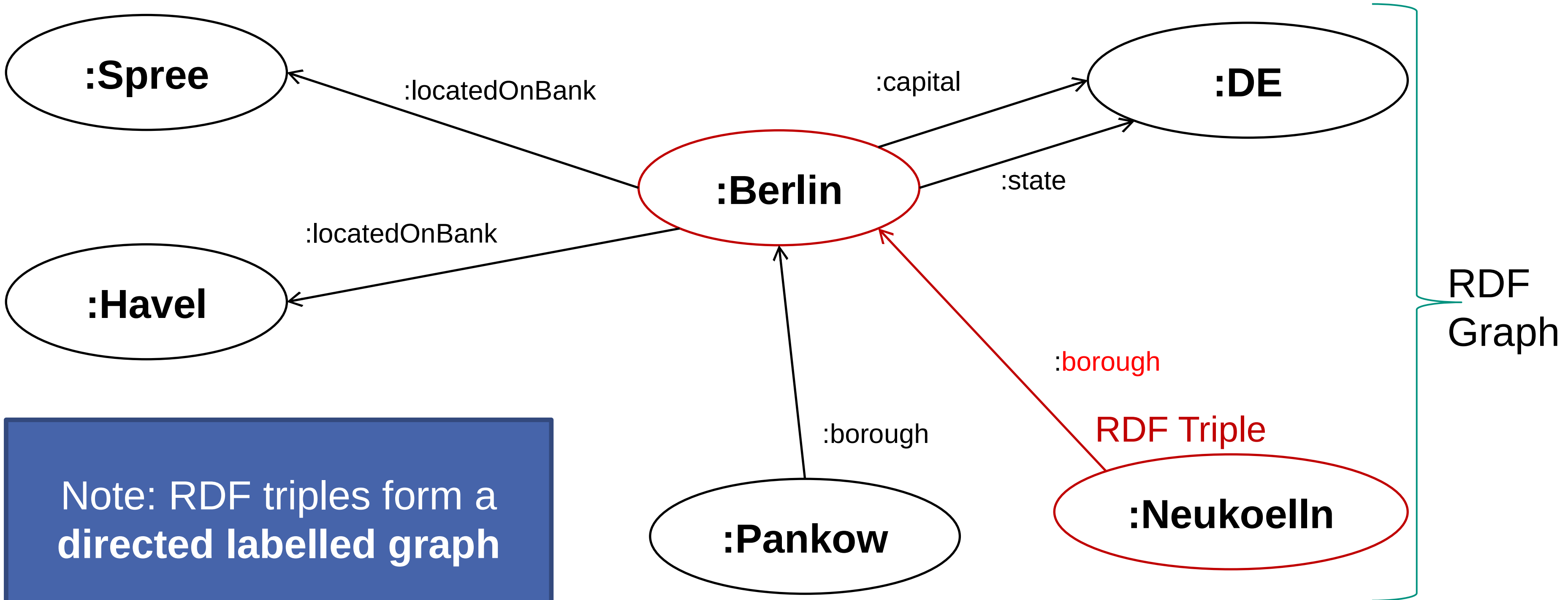
Blank node as subject/object

https://www.w3.org/TR/rdf11-concepts/#section-blank-nodes

RDF – A Graph-based Data Model

- We arrange RDF Terms in RDF Triples ⩦ the edges in RDF Graphs

Definition (RDF Triple, RDF Graph). Let be the set of URIs, the set of blank nodes, and the set of RDF literals. A tuple is called an RDF triple, where is the subject, is the predicate and is the object. A set of RDF triples is called RDF graph.



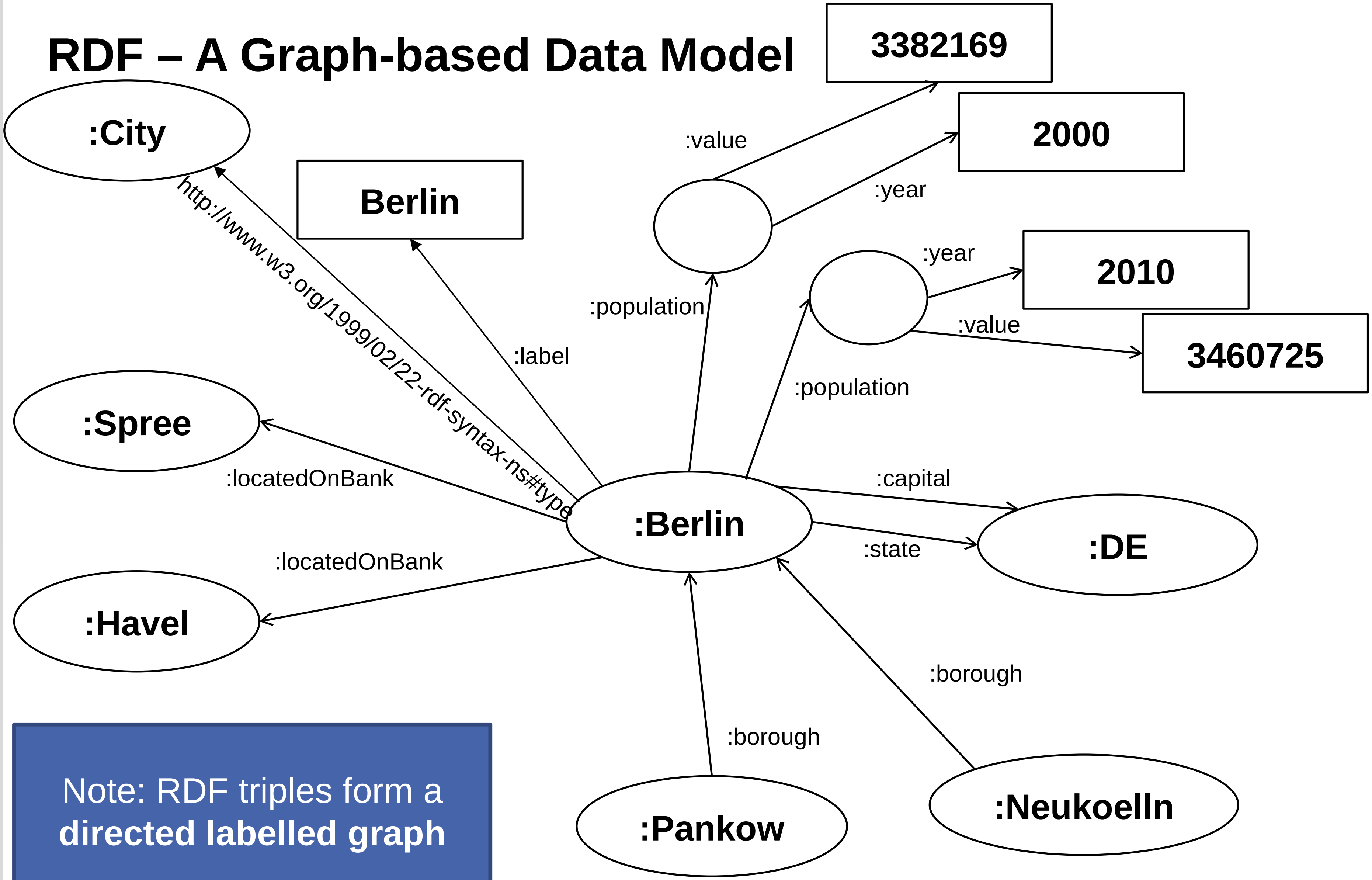
Note: RDF triples form a **directed labelled graph**

Instead of http://example.org/doc.ttl# we write just write“:”

Agenda

1. Graph-structured Data
2. The RDF Vocabulary
3. RDF Abstract Syntax: Terms, Triples and Graphs
4. **Relations Between RDF Graphs**
5. Handling Multiple RDF Graphs in an RDF Dataset

RDF – A Graph-based Data Model

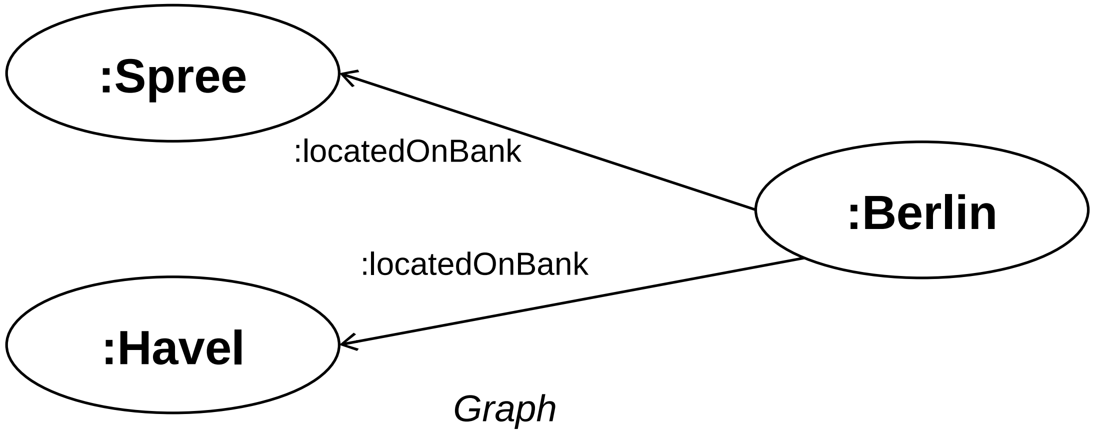


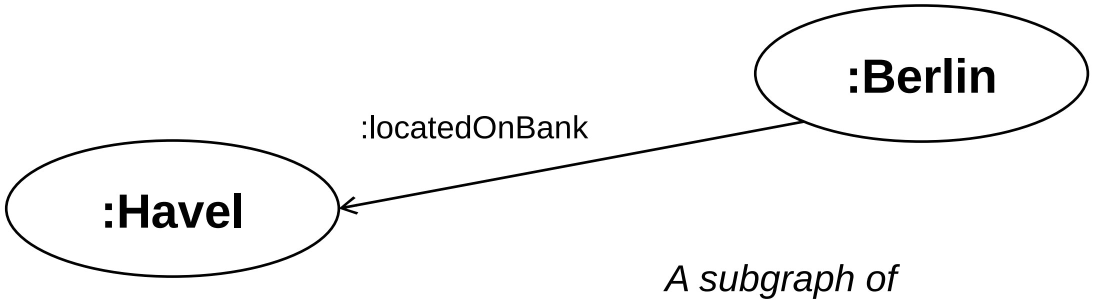
Note: RDF triples form a **directed labelled graph**

Instead of http://example.org/doc.ttl# we write just write“:”

Subgraph

- Definition (Subgraph): *A subgraph of an RDF graph is a subset of the triples in the graph.*
- Example:

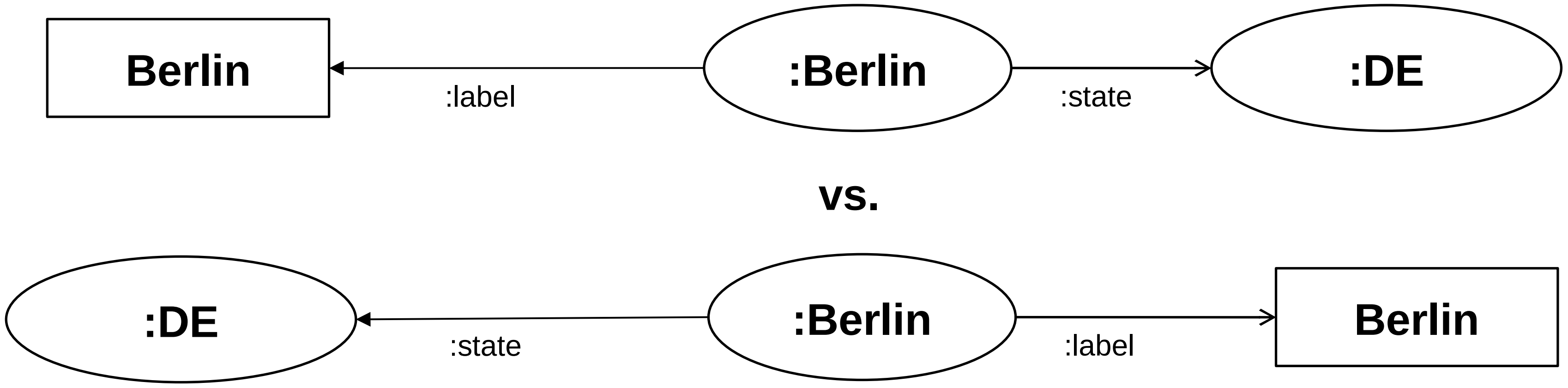




Instead of http://example.org/doc.ttl# we write just write“:”

Isomorphism As Equivalence Relation

- We employ isomorphism to check whether two RDF mean the same
- Are those two graphs the same?



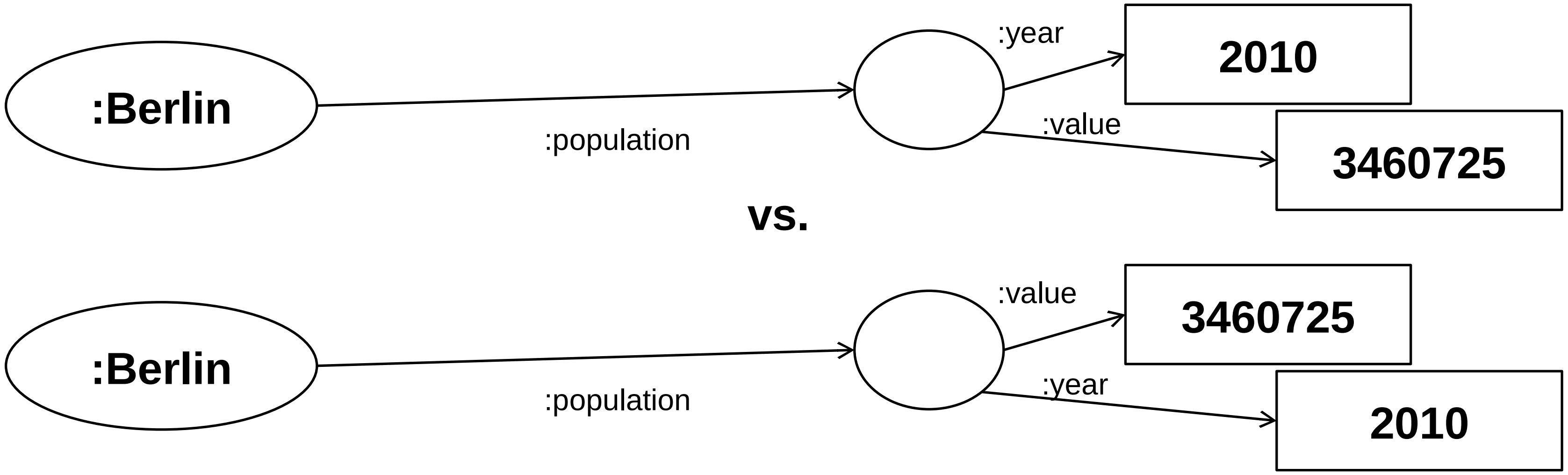
```
@prefix : <http://example.org/doc.ttl#> .
:Berlin :state :DE .
:Berlin :label "Berlin"@de .
```

vs.

```
@prefix : <http://example.org/doc.ttl#>
:Berlin :label "Berlin"@de .
:Berlin :state :DE .
```

Isomorphism As Equivalence Relation

- We employ isomorphism to check whether two RDF mean the same
- Are those two graphs the same?



```
@prefix : <http://example.org/doc.ttl#> .
:Berlin :population _:bn1 .
_:bn1 :year 2010 .
_:bn1 :value 3460725 .
```

vs.

```
@prefix : <http://example.org/doc.ttl#>
_:genid1 :value 3460725 .
:Berlin :population _:genid1 .
_:genid1 :year 2010 .
```

Isomorphism As Equivalence Relation

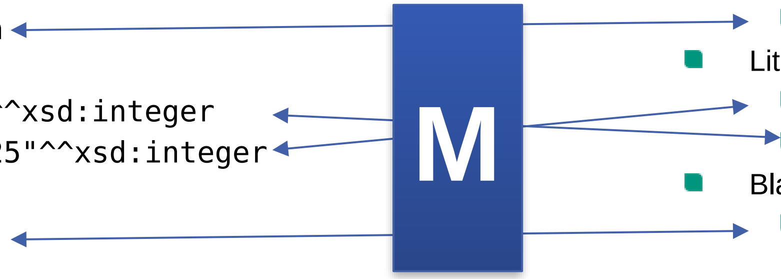
- Two RDF graphs are isomorphic if there is a bijection between the two sets of nodes in the graphs and such that:
  - maps blank nodes to blank nodes.
  - for all RDF literals which are nodes of .
  - for all IRIs which are nodes of .
  - The triple is in if and only if the triple is in

```
@prefix : <http://example.org/doc.ttl#> .
:Berlin :population _:bn1 .
_:bn1 :year 2010 .
_:bn1 :value 3460725 .
```

vs.

```
@prefix : <http://example.org/doc.ttl#>
_:genid1 :value 3460725 .
:Berlin :population _:genid1 .
_:genid1 :year 2010 .
```

- Nodes:
  - URIs:
    - `:Berlin`
  - Literals:
    - `"2010"^^xsd:integer`
    - `"3460725"^^xsd:integer`
  - Blank Nodes:
    - `_:bn1`

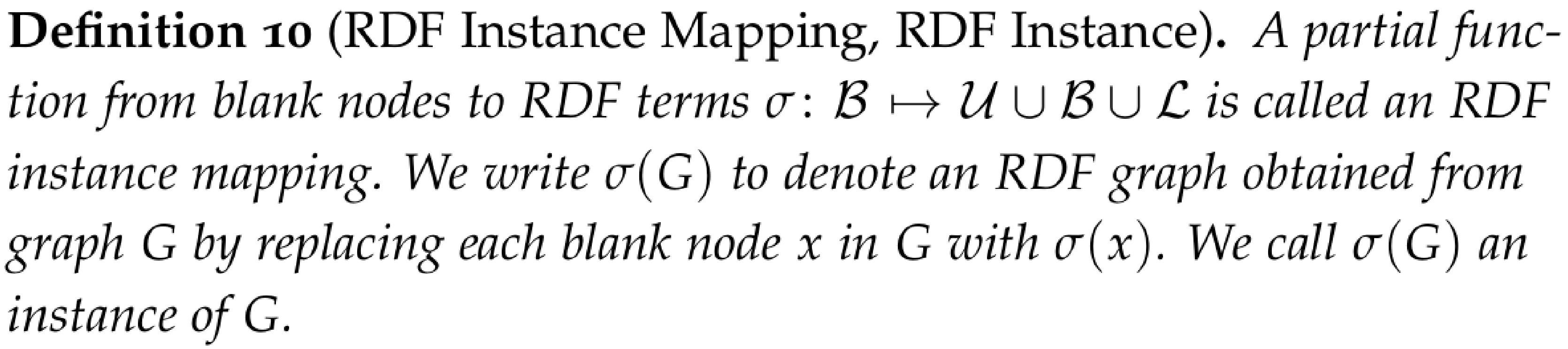


- Nodes:
  - URIs:
    - `:Berlin`
  - Literals:
    - `"3460725"^^xsd:integer`
    - `"2010"^^xsd:integer`
  - Blank Nodes:
    - `_:genid1`

https://www.w3.org/TR/rdf11-concepts/#graph-isomorphism

RDF Instance Mapping

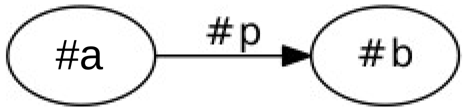
- To understand how blank nodes are handled, we start with the notion of an instance of a graph
- For that, we need the notion of RDF Instance Mapping



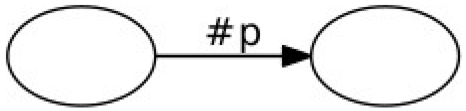
Definition 10 (RDF Instance Mapping, RDF Instance). *A partial function from blank nodes to RDF terms* $\sigma : \mathcal{B} \mapsto \mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$ *is called an RDF instance mapping. We write* $\sigma(G)$ *to denote an RDF graph obtained from graph* G *by replacing each blank node* x *in* G *with* $\sigma(x)$*. We call* $\sigma(G)$ *an instance of* G*.*

Example RDF Instance Mapping

- Graph G0: `<#a> <#p> <#b> .`



- Graph G1: `_:bn1 <#p> _:bn2 .`



- G0 is an instance of G1, assuming the RDF instance mapping :
  - (`_:bn1`) = <#a>
  - (`_:bn2`) = <#b>

Agenda

1. Graph-structured Data
2. The RDF Vocabulary
3. RDF Abstract Syntax: Terms, Triples and Graphs
4. Relations Between RDF Graphs
5. **Handling Multiple RDF Graphs in an RDF Dataset**

Considering the following prefix declarations:

```
@prefix dbr: <http://dbpedia.org/resource/>
@prefix dbp: <http://dbpedia.org/property/>
@prefix d: <http://example.org/doc.ttl#> .
```

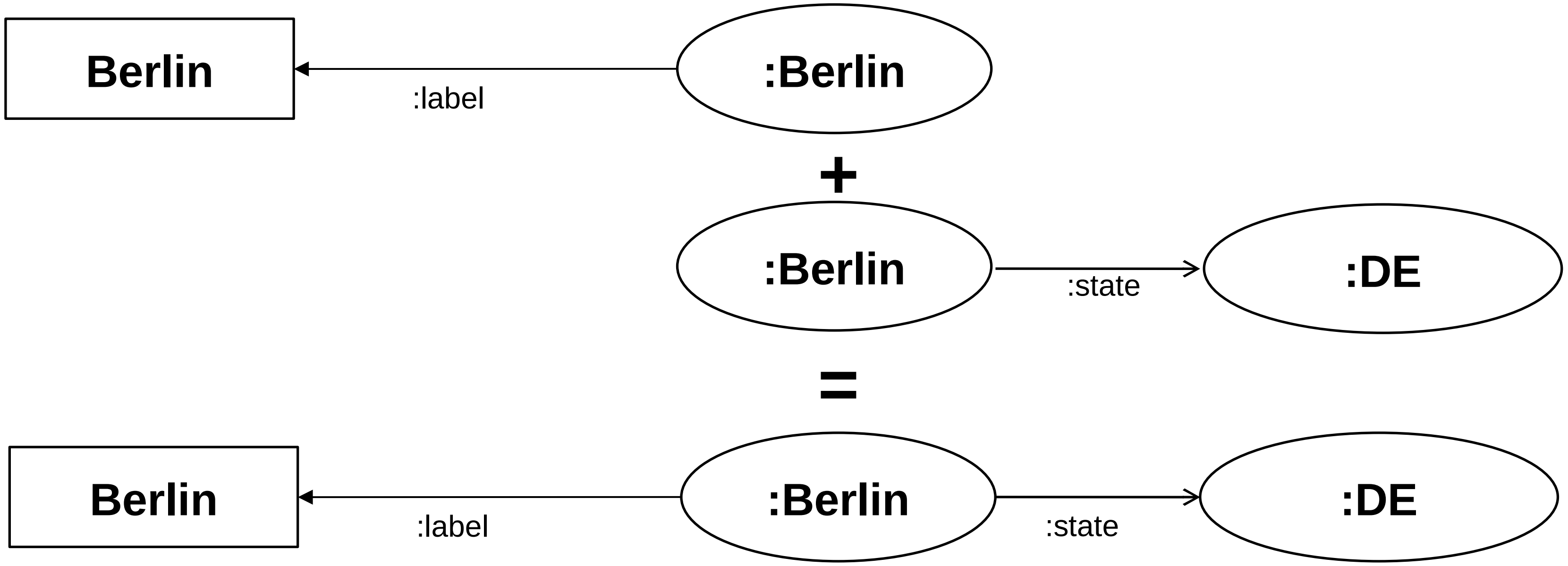
RDF Dataset

- To talk about a collection of RDF graphs, we use the RDF Dataset.
- In an RDF dataset, graphs can be identified using a name
  - The name can be a URI or a blank node
  - There can be one graph without a name, the default graph
  - The name does not need to have any meaning for the graph
- Definition (Named Graph, RDF Dataset): *Let be the set of RDF graphs and be the set of URIs. A pair is called a named graph. An RDF dataset consists of a (possibly empty) set of named graphs (with distinct names) and a default graph without a name.*
- Example:

Name	Graph
`<http://example.org/doc.ttl>`	`d:Berlin d:state d:DE .` `d:Berlin d:label "Berlin"@de .`
`<http://dbpedia.org/data/Berlin.ttl>`	`dbr:Berlin dbo:areaCode 030 .` `dbr:Berlin dbo:kfz "B" .`
	`d:Berlin owl:sameAs dbr:Berlin .`

Combining 2 RDF Graphs: Union

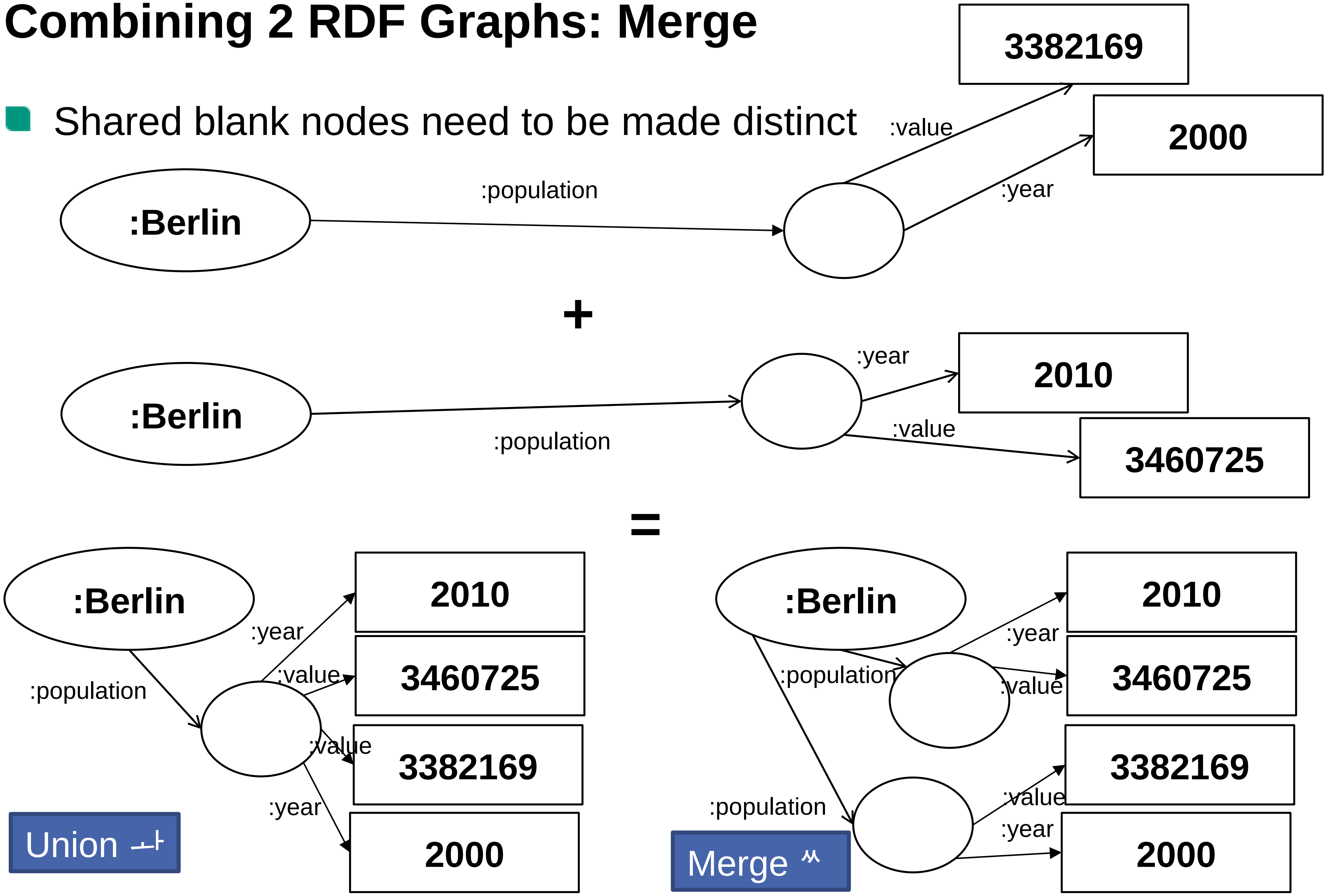
- No blank nodes in the RDF graphs ⇒ RDF graph combination trivial



- Simply take the union of the RDF triples in the RDF graphs

Combining 2 RDF Graphs: Merge

- Shared blank nodes need to be made distinct



RDF Merge Example in Triples

- Consider the following RDF graphs:
  - G:

```
@prefix : <http://example.org/doc.ttl#>.
:Berlin :population _:pop .
_:pop :value 3382169 ; :year 2000 .
```

  - E:

```
@prefix : <http://example.org/doc.ttl#>.
:Berlin :population _:pop .
_:pop :value 3460725 ; :year 2010 .
```

- Incorrect merge of G and E:
  - G1:

```
@prefix : <http://example.org/doc.ttl#>.
:Berlin :population _:pop .
_:pop :value 3382169 ; :year 2000 .
_:pop :value 3460725 ; :year 2010 .
```

- Correct merges of G and E:
  - G2:

```
@prefix : <http://example.org/doc.ttl#>.
:Berlin :population _:pop1, _:pop2 .
_:pop1 :value 3382169 ; :year 2000 .
_:pop2 :value 3460725 ; :year 2010 .
```

  - G3:

```
@prefix : <http://example.org/doc.ttl#>.
:Berlin :population _:pop1, _:pop2 .
_:pop2 :value 3382169 ; :year 2000 .
_:pop1 :value 3460725 ; :year 2010 .
```

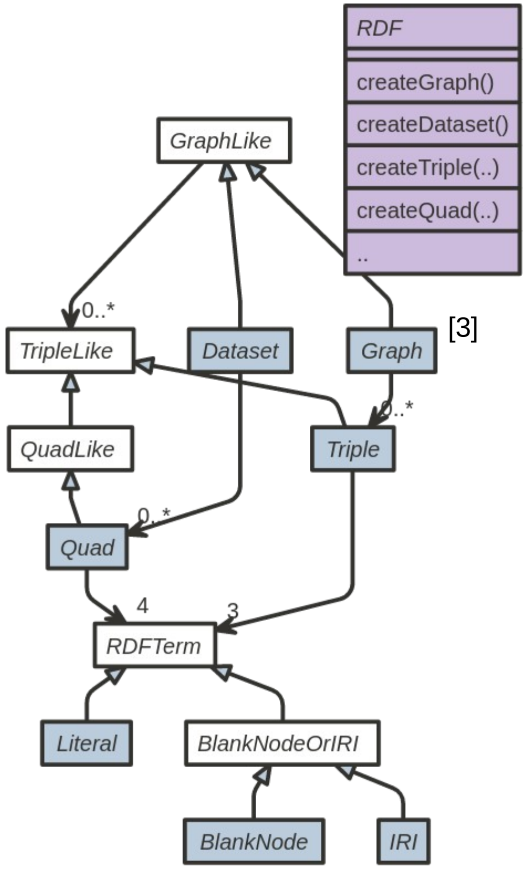
APIs for RDF

[1] [2]

- RDF APIs allow to manipulate and query RDF data adhering to a native programming paradigm.

- Software libraries are available for many programming languages:
  - RDFLib is a RDF API for Python
  - RDF.js is a RDF API for JavaScript

- Commons RDF aims to provide a common library for RDF 1.1 that could be implemented by systems on the Java Virtual Machine.



[3]

[1] https://rdflib.readthedocs.io/en/stable/
[2] https://rdf.js.org/
[3] https://commons.apache.org/proper/commons-rdf/index.html

Learning Goals

- G 3.1 Explain the benefits of a graph-structured data model, and outline different serialisation syntaxes for RDF graphs.
- G 3.2 Correctly use RDF lists in both the Turtle syntax shortcut and the triple representation; correctly use reification in modelling.
- G 3.3 Decide whether two RDF graphs are subgraphs of each other.
- G 3.4 Check whether one graph is an instance of another graph; provide instance mappings between a graph and its instance.
- G 3.5 Construct an RDF dataset from multiple RDF graphs.

Recap: From N-Triples to Turtle

```
<http://example.org/doc.ttl#Berlin> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://example.org/doc.ttl#City> .
<http://example.org/doc.ttl#Berlin> <http://example.org/doc.ttl#capital> <http://example.org/doc.ttl#DE> .
<http://example.org/doc.ttl#Berlin> <http://example.org/doc.ttl#state> <http://example.org/doc.ttl#DE> .
<http://example.org/doc.ttl#Berlin> <http://example.org/doc.ttl#locatedOnBank>
<http://example.org/doc.ttl#Spree> .
<http://example.org/doc.ttl#Berlin> <http://example.org/doc.ttl#locatedOnBank>
<http://example.org/doc.ttl#Havel> .
<http://example.org/doc.ttl#Pankow> <http://example.org/doc.ttl#borough> <http://example.org/doc.ttl#Berlin> .
<http://example.org/doc.ttl#Neukoelln> <http://example.org/doc.ttl#borough>
<http://example.org/doc.ttl#Berlin> .
<http://example.org/doc.ttl#Berlin> <http://example.org/doc.ttl#label> "Berlin"@de .
<http://example.org/doc.ttl#Berlin> <http://example.org/doc.ttl#population> _:genid1 .
<http://example.org/doc.ttl#Berlin> <http://example.org/doc.ttl#population> _:genid2 .
_:genid1 <http://example.org/doc.ttl#value> "3382169"^^<http://www.w3.org/2001/XMLSchema#integer> .
_:genid1 <http://example.org/doc.ttl#year> "2000"^^<http://www.w3.org/2001/XMLSchema#integer> .
_:genid2 <http://example.org/doc.ttl#value> "3460725"^^<http://www.w3.org/2001/XMLSchema#integer> .
_:genid2 <http://example.org/doc.ttl#year> "2010"^^<http://www.w3.org/2001/XMLSchema#integer> .
```

+ CURIEs

- You can allow for CURIEs by issuing @prefix directives of the form:
  - `@prefix` *prefix-label*: <*associated URI*> .

```
@prefix : <http://example.org/doc.ttl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#integer> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
:Berlin rdf:type :City .
:Berlin :capital :DE .
:Berlin :state :DE .
:Berlin :locatedOnBank :Spree .
:Berlin :locatedOnBank :Havel .
:Pankow :borough :Berlin .
:Neukoelln :borough :Berlin .
:Berlin :label "Berlin"@de .
:Berlin :population _:genid1 .
:Berlin :population _:genid2 .
_:genid1 :value "3382169"^^xsd:integer .
_:genid1 :year "2000"^^xsd:integer .
_:genid2 :value "3460725"^^xsd:integer .
_:genid2 :year "2010"^^xsd:integer .
```

+ Abbreviation for rdf:type

- You can abbreviate `rdf:type` with “a”

```
@prefix : <http://example.org/doc.ttl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#integer> .

:Berlin a :City .
:Berlin :capital :DE .
:Berlin :state :DE .
:Berlin :locatedOnBank :Spree .
:Berlin :locatedOnBank :Havel .
:Pankow :borough :Berlin .
:Neukoelln :borough :Berlin .
:Berlin :label "Berlin"@de .
:Berlin :population _:genid1 .
:Berlin :population _:genid2 .
_:genid1 :value "3382169"^^xsd:integer .
_:genid1 :year "2000"^^xsd:integer .
_:genid2 :value "3460725"^^xsd:integer .
_:genid2 :year "2010"^^xsd:integer .
```

+ Abbreviations for Some XML Schema Datatypes

- You can use shorthands for numbers typed with `xsd:integer` , `xsd:decimal` (with “`.`”), and `xsd:float` (written in scientific notation)

```
@prefix : <http://example.org/doc.ttl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#integer> .

:Berlin a :City .
:Berlin :capital :DE .
:Berlin :state :DE .
:Berlin :locatedOnBank :Spree .
:Berlin :locatedOnBank :Havel .
:Pankow :borough :Berlin .
:Neukoelln :borough :Berlin .
:Berlin :label "Berlin"@de .
:Berlin :population _:genid1 .
:Berlin :population _:genid2 .
_:genid1 :value 3382169 .
_:genid1 :year 2000 .
_:genid2 :value 3460725 .
_:genid2 :year 2010 .
```

+ Abbreviations for Repetitions of Subject+Predicate

- Use the colon “,” in consecutive triples to repeat subject and predicate
- Order the triples wisely to benefit; indentation helps for the overview

```
@prefix : <http://example.org/doc.ttl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#integer> .

:Berlin a :City .
:Berlin :capital :DE .
:Berlin :state :DE .
:Berlin :locatedOnBank :Spree , :Havel .
:Pankow :borough :Berlin .
:Neukoelln :borough :Berlin .
:Berlin :label "Berlin"@de .
:Berlin :population _:genid1 , _:genid2 .
_:genid1 :value 3382169 .
_:genid1 :year 2000 .
_:genid2 :value 3460725 .
_:genid2 :year 2010 .
```

+ Abbreviations for Repetitions of Subject

- Use the semicolon "`;`" in consecutive triples to repeat the subject
- Order the triples wisely to benefit; indentation helps for the overview

```
@prefix : <http://example.org/doc.ttl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#integer> .

:Berlin a :City ;
  :capital :DE ;
  :state :DE ;
  :locatedOnBank :Spree , :Havel .
:Pankow :borough :Berlin .
:Neukoelln :borough :Berlin .
:Berlin :label "Berlin"@de ;
  :population _:genid1 , _:genid2 .
_:genid1 :value 3382169 ;
  :year 2000 .
_:genid2 :value 3460725 ;
  :year 2010 .
```

Are the triples in a smart order?

+ Abbreviations for Blank Nodes

- Use brackets “[ ]” to abbreviate blank nodes
- Note that you need to group *all* mentions of the former blank node ID

```
@prefix : <http://example.org/doc.ttl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#integer> .

:Berlin a :City ;
  :capital :DE ;
  :state :DE ;
  :locatedOnBank :Spree , :Havel .
:Pankow :borough :Berlin .
:Neukoelln :borough :Berlin .
:Berlin :label "Berlin"@de ;
  :population [ :value 3382169 ; :year 2000 ] ,
              [ :value 3460725 ; :year 2010 ] .
```

The RDF Graph in Turtle Serialisation

```
@prefix : <http://example.org/doc.ttl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#integer> .

:Berlin a :City ;
  :capital :DE ;
  :state :DE ;
  :locatedOnBank :Spree , :Havel .
:Pankow :borough :Berlin .
:Neukoelln :borough :Berlin .
:Berlin :label "Berlin"@de ;
  :population [ :value 3382169 ; :year 2000 ] ,
              [ :value 3460725 ; :year 2010 ] .
```