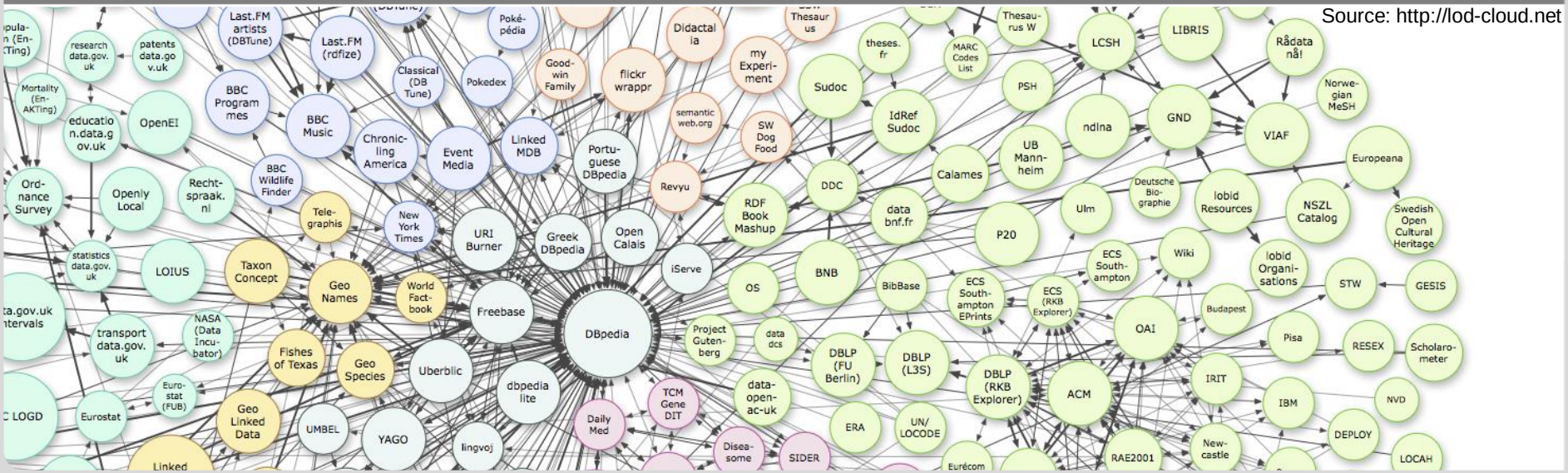
RDF Concepts and Syntax
How to represent data on the web?

Dr. Tobias Käfer

AI4INDUSTRY SUMMER SCHOOL



CC - Creative Commons Licensing

- 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/

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`**

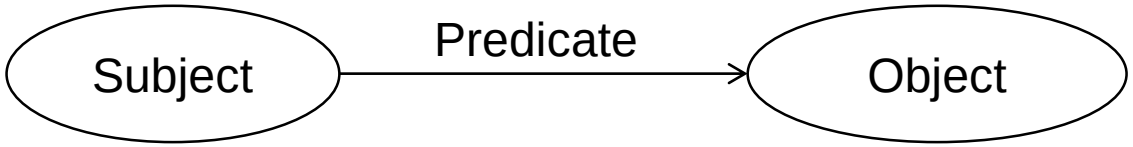
[1] Cf. Mike Amundsen: "Autonomous Agents on the Web", Keynote at the Workshop for Services an Applications over Linked Data, 2013 https://www.slideshare.net/rnewton/autonomous-agents-on-the-web-22078931

ENTER: THE RESOURCE DESCRIPTION FORMAT

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:

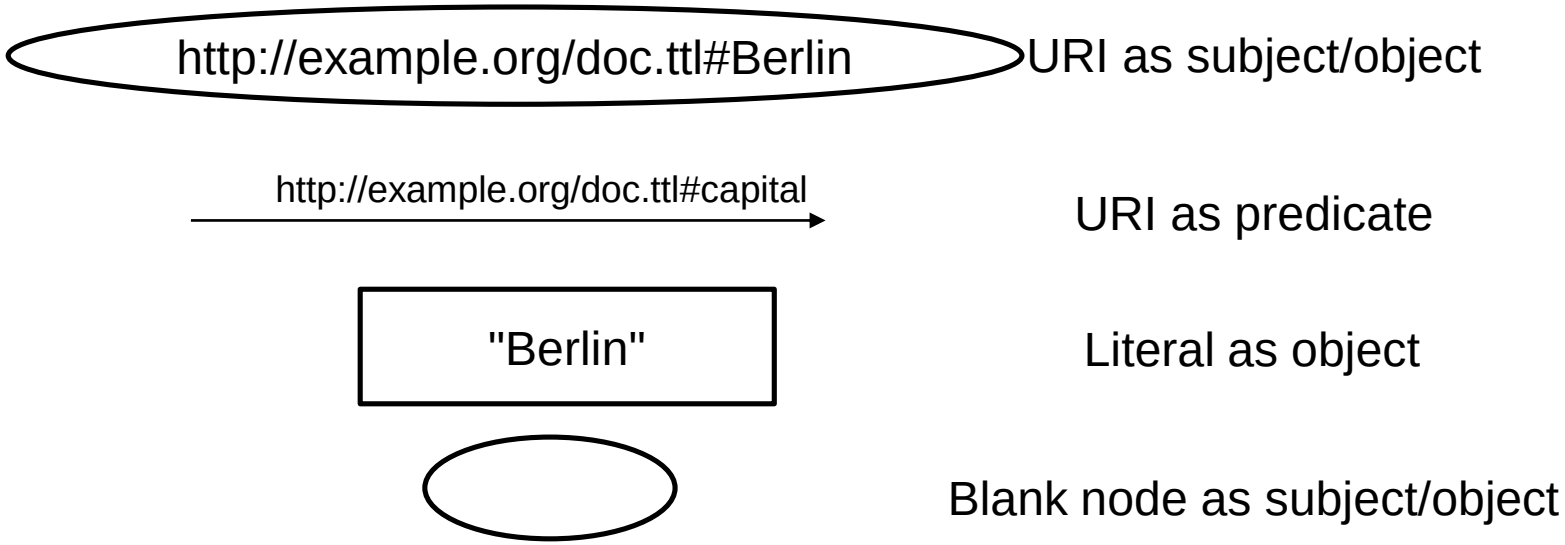


- RDF graphs can be presented as directed labelled graph

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

RDF Term Overview: URIs - Blank Nodes - Literals

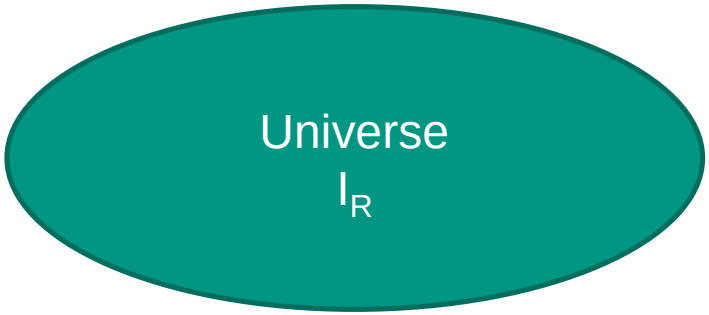
- **URIs** are used to globally identify resources
- **Blank nodes** refer to resources, too, but these resources can only be identified within a file and are not globally addressable (later more)
- **Literals** refer to concrete data values such as strings, integers, floats or dates. In RDF, we can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



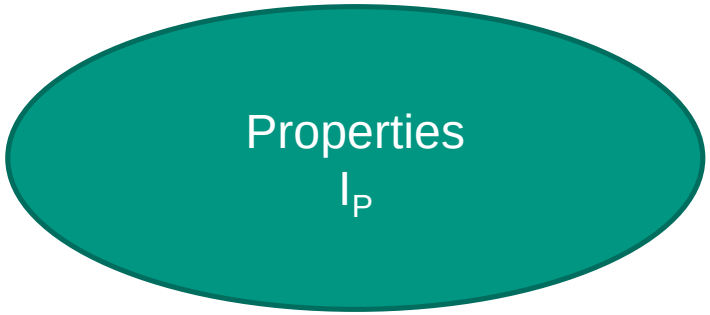
RESOURCE DENOTATION USING RDF TERMS

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.*



The set of all things we want to talk about



The set of resources for the URIs in predicate position

URIs – Uniform Resource Identifiers (Recap and Terminology)

- 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 `http://example.org/doc.ttl#Berlin`
- IRIs
  - Standard to allow for using characters outside of US-ASCII in URIs
  - We typically use “URI“ and “IRI“ interchangably

URI as subject/object

URI as predicate

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

URI Recap, Some Terminology and Practices

- 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/doc.ttl#Berlin`
- 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://dbpedia.org/resource/Berlin` redirects (303) to `http://dbpedia.org/data/Berlin.ttl` 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 $< lex, dt >$
      - For simple literals: `xsd:string`
      - For language-tagged literals: `rdf:langString` → triples $< lex, dt, lang >$
    - 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

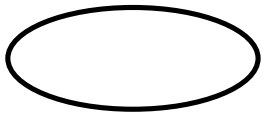
"01"^^xsd:integer

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

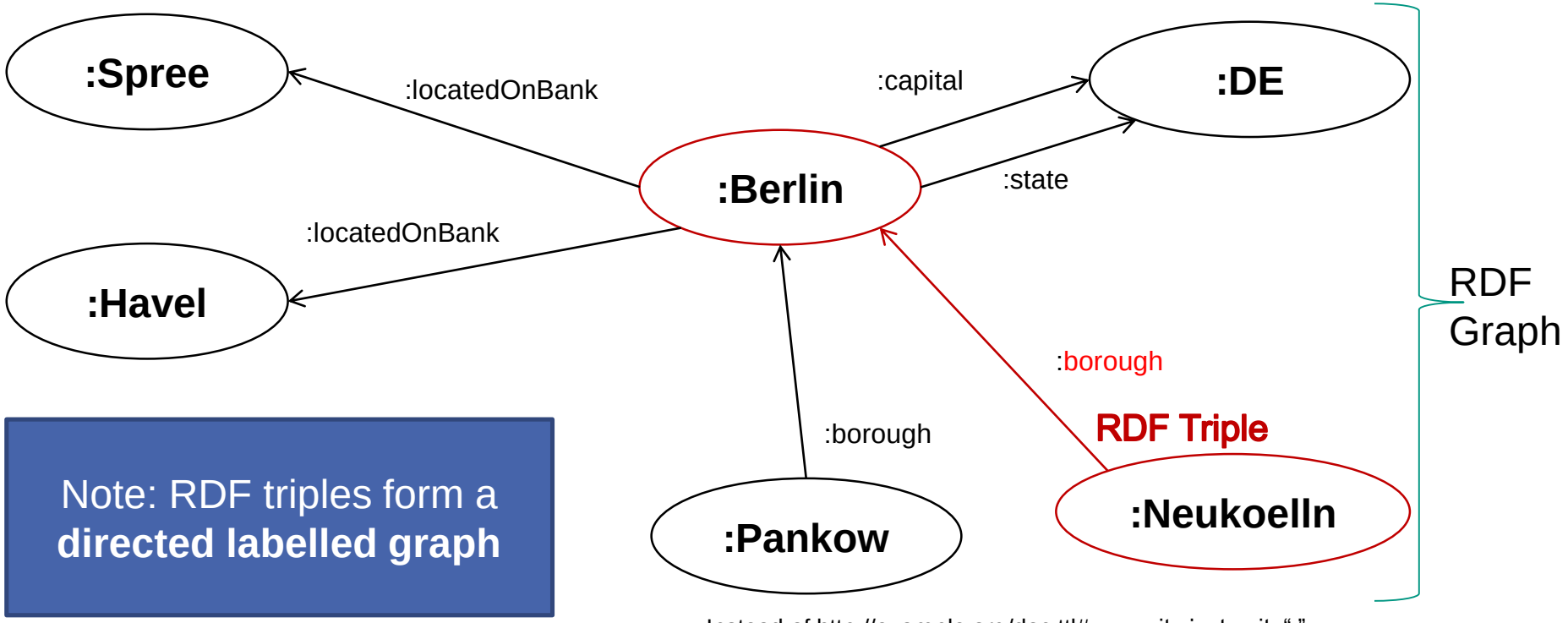
https://www.w3.org/TR/rdf11-concepts/#section-blank-nodes

RESOURCE DESCRIPTIONS IN RDF GRAPHS

RDF – A Graph-based Data Model

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

Definition (RDF Triple, RDF Graph). Let $\mathcal{U}$ be the set of URIs, $\mathcal{B}$ the set of blank nodes, and $\mathcal{L}$ the set of RDF literals. A tuple $< s, p, o > \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$ is called an RDF triple, where s is the subject, p is the predicate and o is the object. A set of RDF triples is called RDF graph.



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

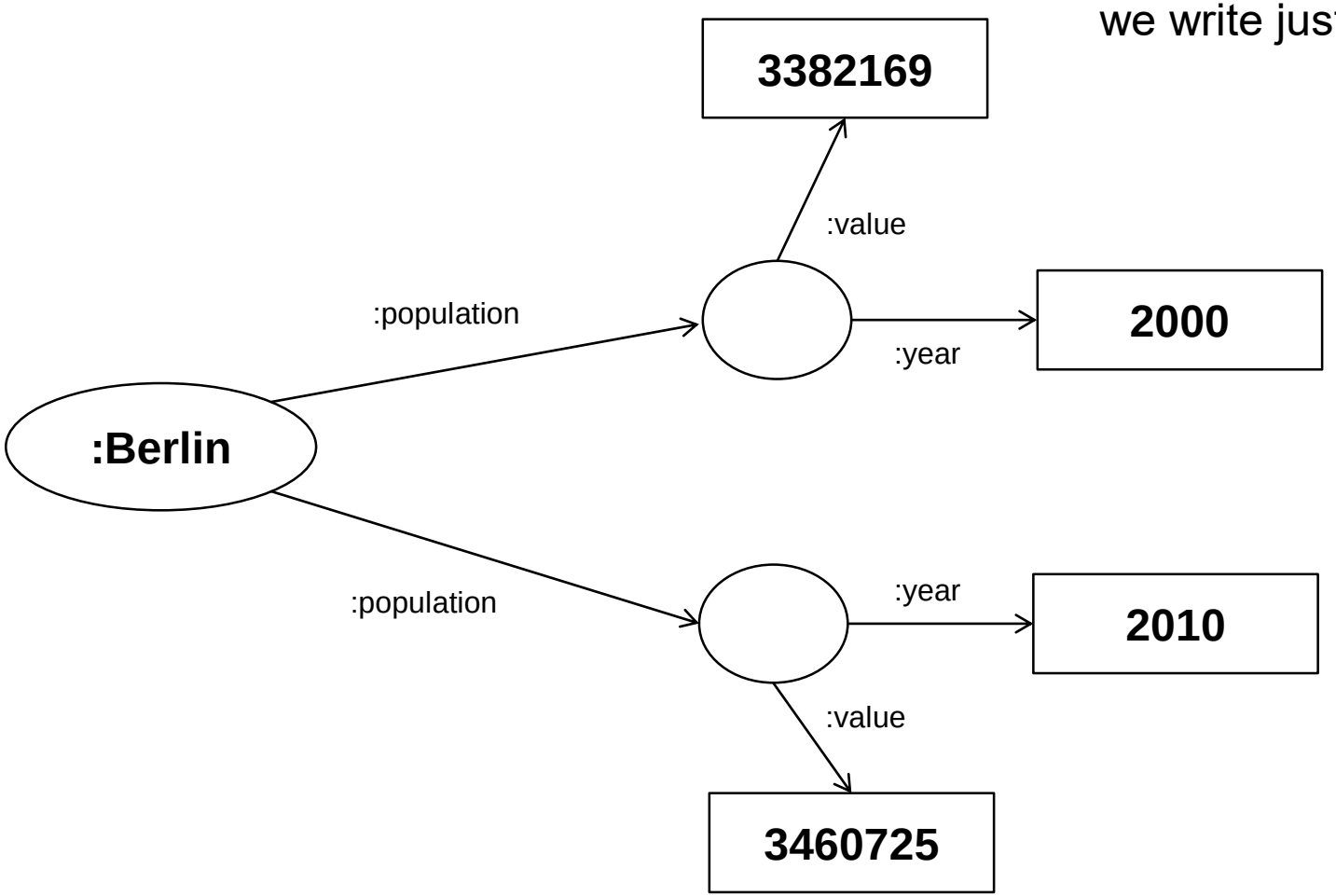
Instead of http://example.org/doc.ttl# we write just write":"

N-ary Relations

- An RDF property represents a binary relation between resources
- But there are cases where we want to model relations between more than two resources
- So-called n-ary relations can be modelled as binary relations, if we think of the relation itself as a resource
- Often, we use blank nodes to identify the relation (as resource)

Example: N-ary Relation in RDF

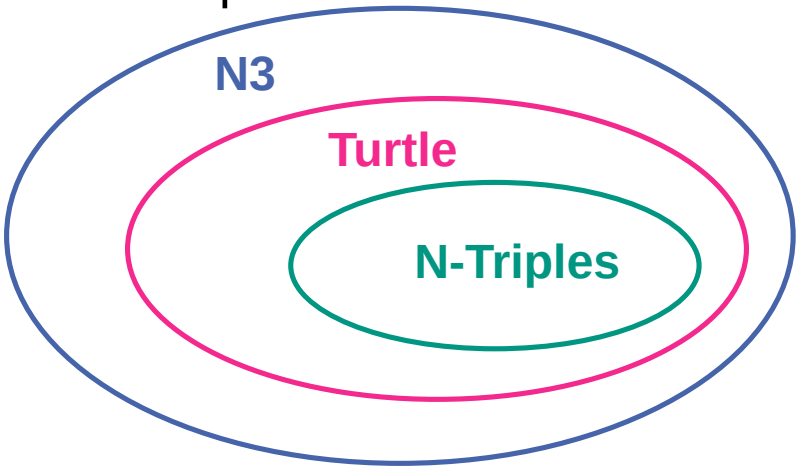
Instead of
http://example.org/doc.ttl#
we write just write“:”



RDF SERIALISATION FORMATS

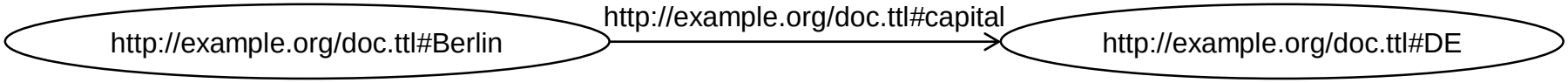
RDF Syntaxes

- Based on Notation3 (an expressive textual syntax for a superset of the RDF data model)
  - **N-Triples** is a very simple syntax, in which one triple is written in one line
  - **Turtle** adds syntactical features to N-Triples that increase human readability and writeability (eg. CURIEs, abbreviations)
- **RDF/XML** is a XML-based syntax with high historical relevance and practical prevalence
- **JSON-LD** is a JSON-based syntax, which allows for eased interoperability of RDF with JavaScript code



N-Triples – A Simple Syntax for RDF Triples

- N-Triples provides a very straight-forward way to write down RDF triples



- The basic structure consists of subject property object triples, followed by a dot and a newline
- URIs are enclosed in angle brackets (“<>”). No relative URIs!
- Blank nodes are prefixed with an underscore and a colon (“_:”)
- Literals are enclosed in quotation marks (“""”)
- Comments are marked with a hash character (“#”)

```
<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#label> "Berlin" .

<http://example.org/doc.ttl#Berlin> <http://example.org/doc.ttl#population> _:bn .  # part of n-ary rel.
```

- The s p o . representation is also called simple triple form

https://www.w3.org/TR/n-triples/

Simple, Datatyped, and Language-tagged Literals in N-Triples

- Simple, Datatyped and Language-tagged Literals enclose the lexical form in quotation marks (“"”)
- Datatyped literals use two caret characters (“^^”) to specify the datatype URI
- For example, a literal denoting the integer value 3460725 is written as

```
"3460725"^^<http://www.w3.org/2001/XMLSchema#integer>
```

- Language-tagged literals use the at character (“@”) to specify the language
- For example, the literal denoting *Berlin* in German is written as

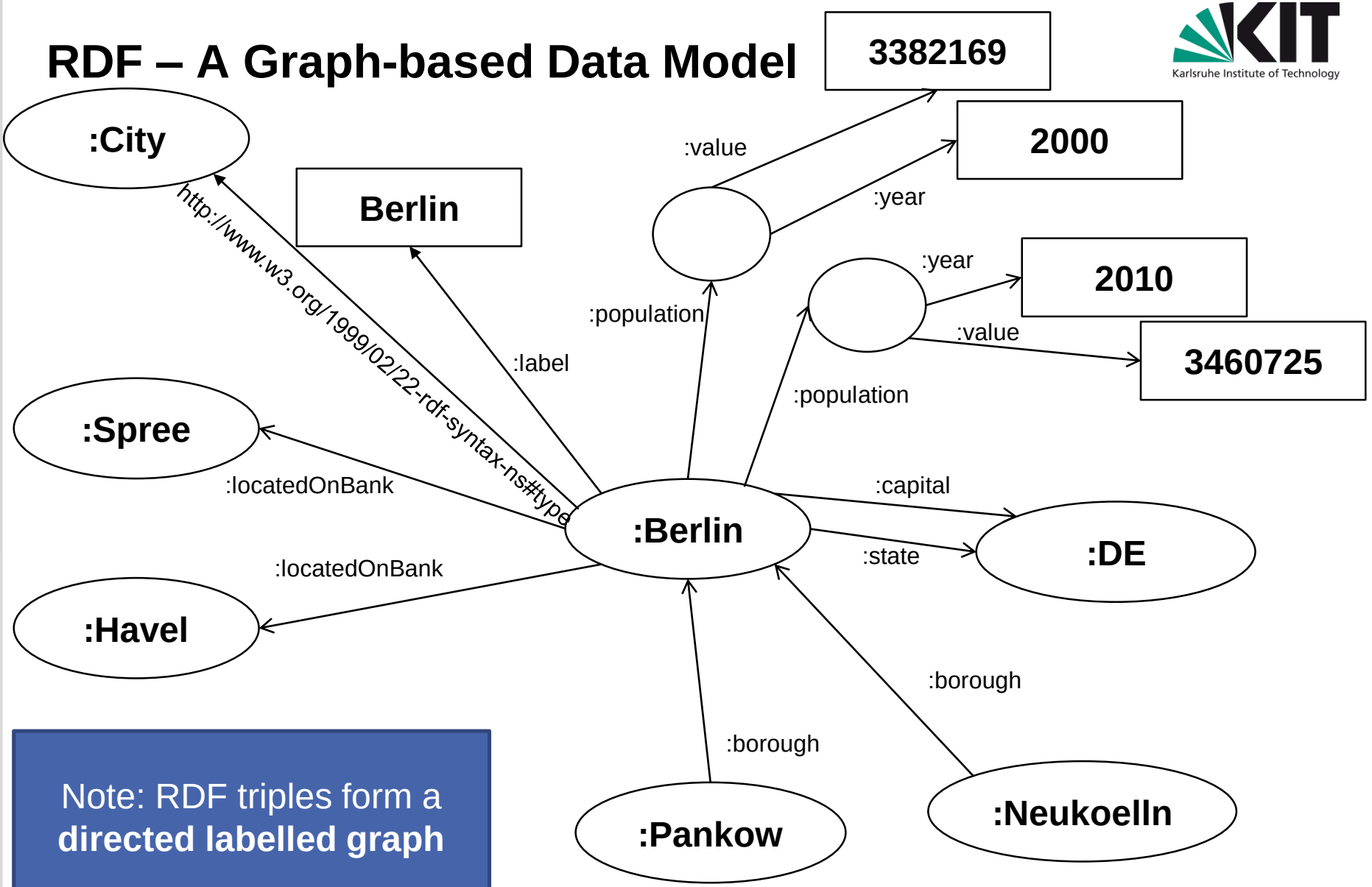
```
"Berlin"@de
```

Turtle

- Turtle – the “Terse RDF Triple Language”
- Easier for a human to read and write than N-Triples
- Will be used throughout the lecture
- Supports:
  1. CURIEs
  2. Relative URIs
  3. Abbreviation for rdf:type (“a”)
  4. Abbreviations for literals with some XML Schema datatypes
  5. Abbreviations for repetition of subject (“;”) and subject+predicate (“,”)
  6. Abbreviations for RDF lists (later)

- We now derive Turtle starting with N-Triples considering above points except 2 and 6

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":"

N-Triples

```
<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 ] .
```

Now You Know Turtle

- Language features not covered in the previous slides:
  - Relative URIs (see slides on URIs)
  - RDF list syntax (see later)

```
@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 ] .
```

Turtle Exercise

- Describe the facts from → as RDF Triples in Turtle Syntax
- Invent URIs as necessary
- Type your questions into the chat and raise your hand once you‘re done
- Steps:
  1. Open a text editor, eg. Nano
     ```
     nano production.nt
     ```
  2. Don‘t use abbreviations, ie. write N-Triples
  3. Double-check your solution using rapper or RDFShape [1]
     ```
     rapper -i ntriples production.nt
     ```
  4. Copy the file to a new file
     ```
     cp production.nt production.ttl
     ```
  5. Edit the new file and apply as many Turtle abbreviations as possible
  6. Double-check your solution using rapper or RDFShape [1]
     ```
     rapper -i turtle production.ttl
     ```

- myProductionSystem isA System
- myProductionSystem hasSubSystem roboticArm1
- myProductionSystem hasSubSystem conveyorBelt2
- roboticArm1 isA System
- roboticArm1 isA RoboticArm
- roboticArm1 hasManufacturer ABB
- conveyorBelt2 isA System
- conveyorBelt2 hasSpeed 0.1

[1] http://rdfshape.herokuapp.com/dataConversions

Sample Solution – Step 2

```
<http://example.org/#myProductionSystem> <http://example.org/#isA> <http://example.org/#System> .
<http://example.org/#myProductionSystem> <http://example.org/#hasSubSystem> <http://example.org/#roboticArm1> .
<http://example.org/#myProductionSystem> <http://example.org/#hasSubSystem> <http://example.org/#conveyorBelt2> .
<http://example.org/#roboticArm1> <http://example.org/#isA> <http://example.org/#System> .
<http://example.org/#roboticArm1> <http://example.org/#isA> <http://example.org/#RoboticArm> .
<http://example.org/#roboticArm1> <http://example.org/#hasManufacturer> <http://example.org/#ABB> .
<http://example.org/#conveyorBelt2> <http://example.org/#isA> <http://example.org/#System> .
<http://example.org/#conveyorBelt2> <http://example.org/#hasSpeed> "0.1" .
```

Sample Solution – Step 5

```
@prefix : <http://example.org/#> .
:myProductionSystem a :System ; # <- let‘s pretend on the previous slide, we had rdf:type
  :hasSubSystem :roboticArm1 , :conveyorBelt2 .
:roboticArm1 a :System , :RoboticArm ; # <- let‘s pretend on the previous slide, we had rdf:type
  :hasManufacturer :ABB .
:conveyorBelt2 a :System ;
  :hasSpeed "0.1" .
#  :hasSpeed 0.1 . <- In the previous slide, we had a string so we can‘t write this line or the two following:
# @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
#  :hasSpeed "0.1"^^xsd:decimal .
```

RDF/XML Example

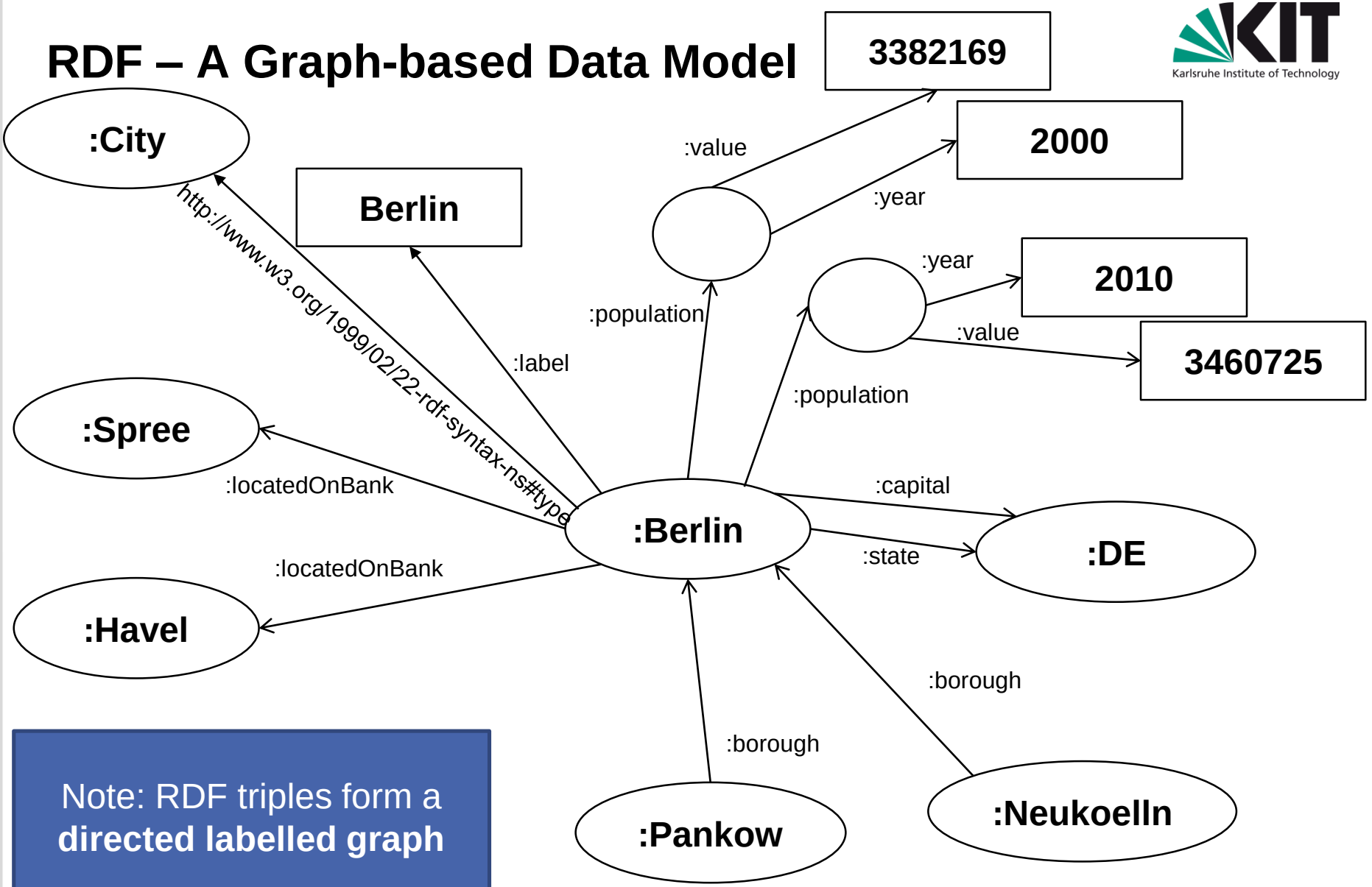
```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns="http://example.org/doc.ttl#">
  <rdf:Description rdf:about="http://example.org/doc.ttl#Berlin">
    <rdf:type rdf:resource="http://example.org/doc.ttl#City"/>
    <capital rdf:resource="http://example.org/doc.ttl#DE"/>
    <state rdf:resource="http://example.org/doc.ttl#DE"/>
    <locatedOnBank rdf:resource="http://example.org/doc.ttl#Spree"/>
    <locatedOnBank rdf:resource="http://example.org/doc.ttl#Havel"/>
    <borough rdf:resource="http://example.org/doc.ttl#Berlin"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/doc.ttl#Pankow">
    <borough rdf:resource="http://example.org/doc.ttl#Berlin"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/doc.ttl#Neukoelln">
    <borough rdf:resource="http://example.org/doc.ttl#Berlin"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/doc.ttl#Berlin">
    <label xml:lang="de">Berlin</label>
    <population rdf:nodeID="genid1"/>
    <population rdf:nodeID="genid2"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="genid1">
    <value rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">3382169</value>
    <year rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">2000</year>
  </rdf:Description>
  <rdf:Description rdf:nodeID="genid2">
    <value rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">3460725</value>
    <year rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">2010</year>
  </rdf:Description>
</rdf:RDF>
```

JSON-LD Example

```
{
  "@context": {
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "doc": "http://example.org/doc.ttl#",
    "people": "doc:population",
    "country": { "@id": "doc:state", "@type": "@id" },
    "borough": { "@reverse": "doc:borough", "@type": "@id" },
    "locatedOnBank": { "@id": "doc:locatedOnBank", "@type": "@id" }
  },
  "@graph": [
    {
      "@id": "doc:Berlin",
      "http://example.org/doc.ttl#label": { "@value": "Berlin", "@language": "de“ },
      "http://example.org/doc.ttl#capital": { "@id": "doc:DE" },
      "rdf:type": { "@id": "doc:City" },
      "locatedOnBank": [ "doc:Havel", "doc:Spree" ],
      "people": [{ "doc:value": 3382169, "doc:year": 2000 },
                 { "doc:value": 3460725, "doc:year": 2010 }],
      "country": "doc:DE",
      "borough": [ "doc:Neukoelln", "doc:Pankow“ ]
    }
  ]
}
```

WORKING WITH MULTIPLE RDF GRAPHS

RDF – A Graph-based Data Model

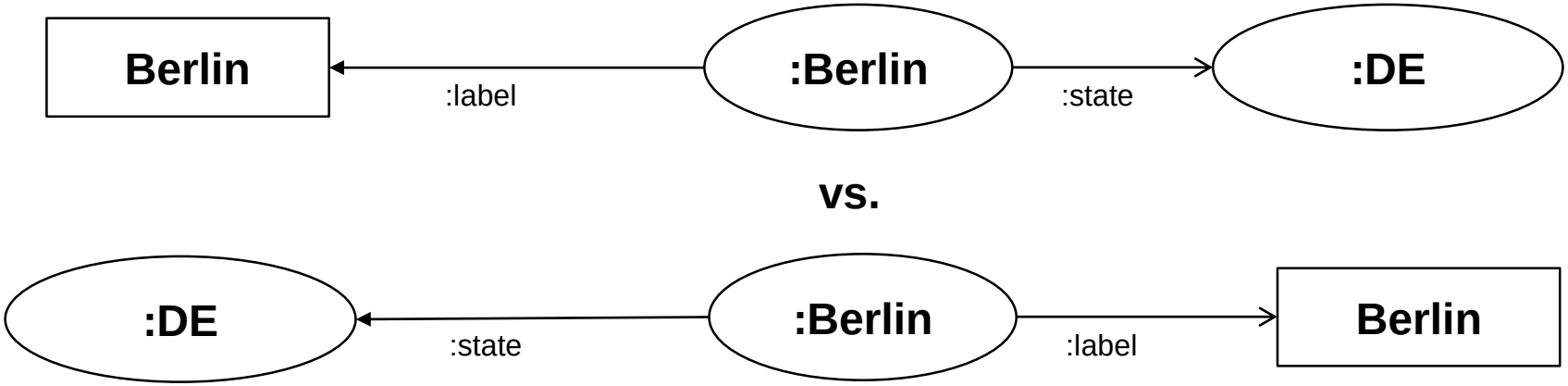


Note: RDF triples form a **directed labelled graph**

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

Isomorphism As Equivalence Relation

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



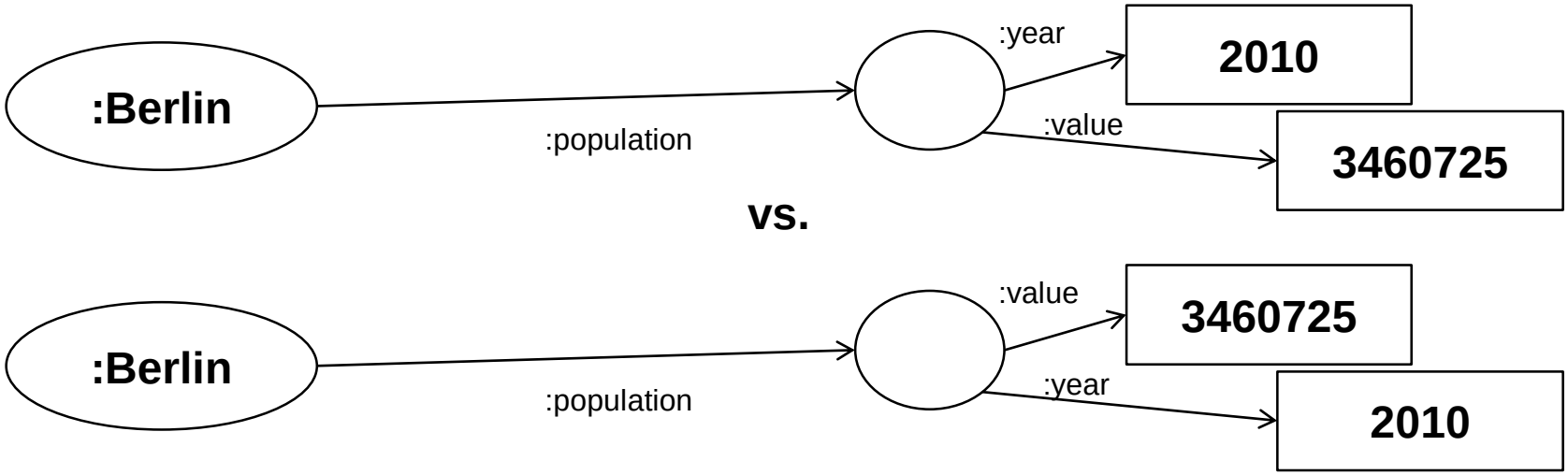

```
@prefix : <http://example.org/doc.ttl#> .
:Berlin :state :DE .
:Berlin :label "Berlin"@de .
```

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 .
```

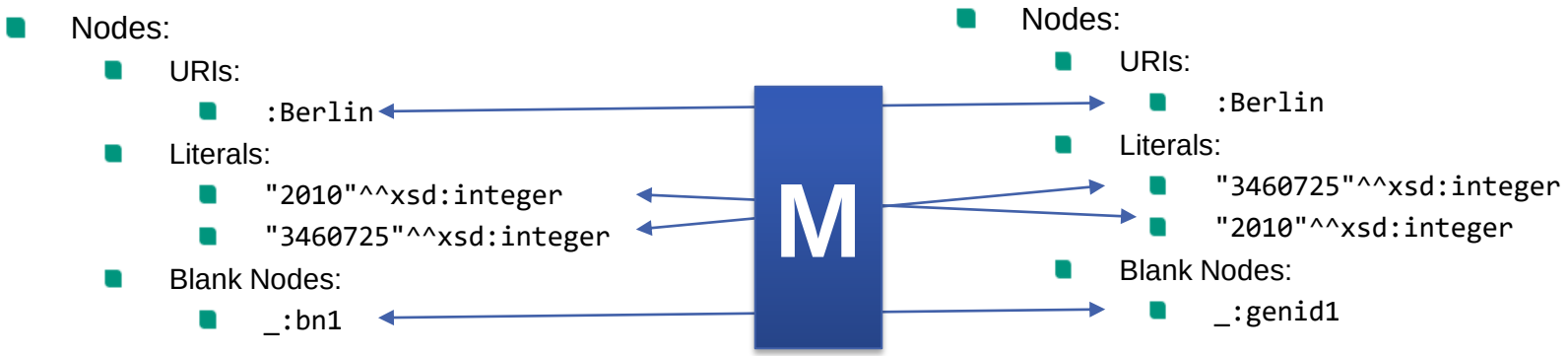
Isomorphism As Equivalence Relation

- Two RDF graphs are isomorphic if there is a bijection M between the two sets of nodes in the graphs G and G' such that:
  - M maps blank nodes to blank nodes.
  - $M(lit) = lit$ for all RDF literals lit which are nodes of G.
  - $M(iri) = iri$ for all IRIs iri which are nodes of G.
  - The triple (s, p, o) is in G if and only if the triple $(M(s), p, M(o))$ is in G'

```
@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 .
```



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

RDF Dataset

Considering the following prefix declarations:

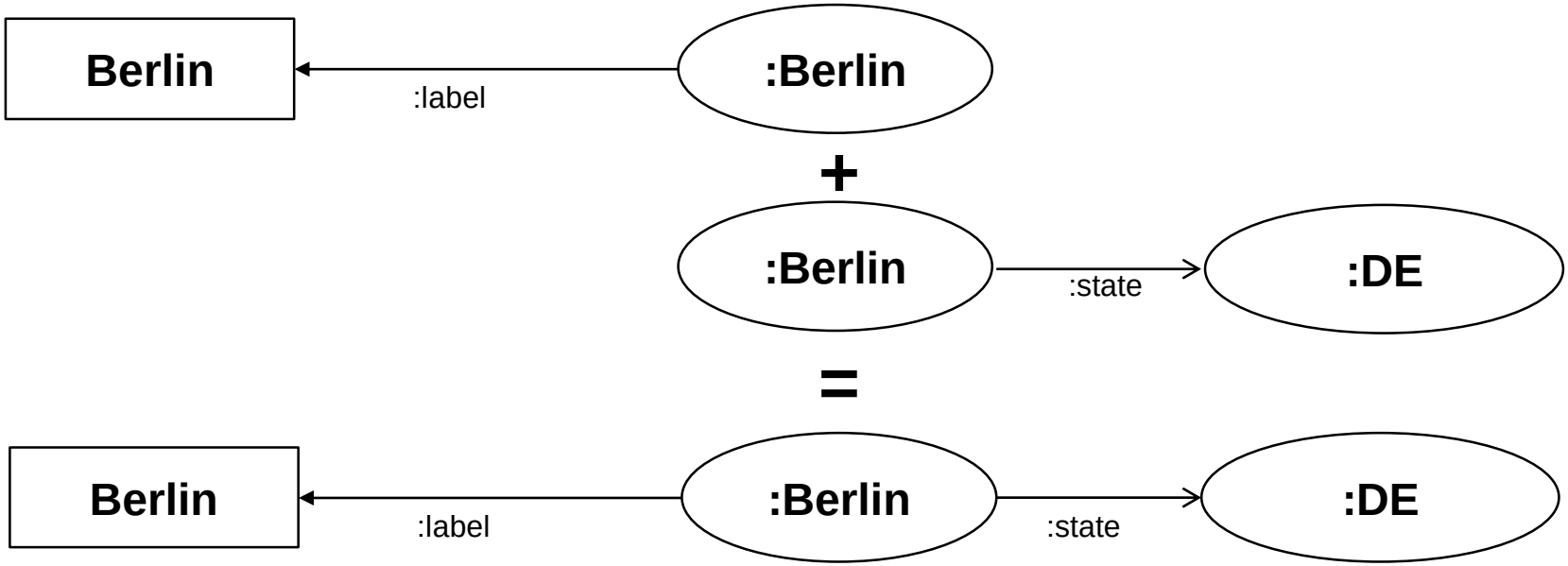
```
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix dbp: <http://dbpedia.org/property/> .
@prefix d: <http://example.org/doc.ttl#> .
```

- 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 $\mathcal{G}$ be the set of RDF graphs and $\mathcal{U}$ be the set of URIs. A pair $< g, u > \in \mathcal{G} \times \mathcal{U}$ is called a named graph. An RDF dataset consists of a (possibly empty) set of named graphs (with distinct names) and a default graph $g \in \mathcal{G}$ 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

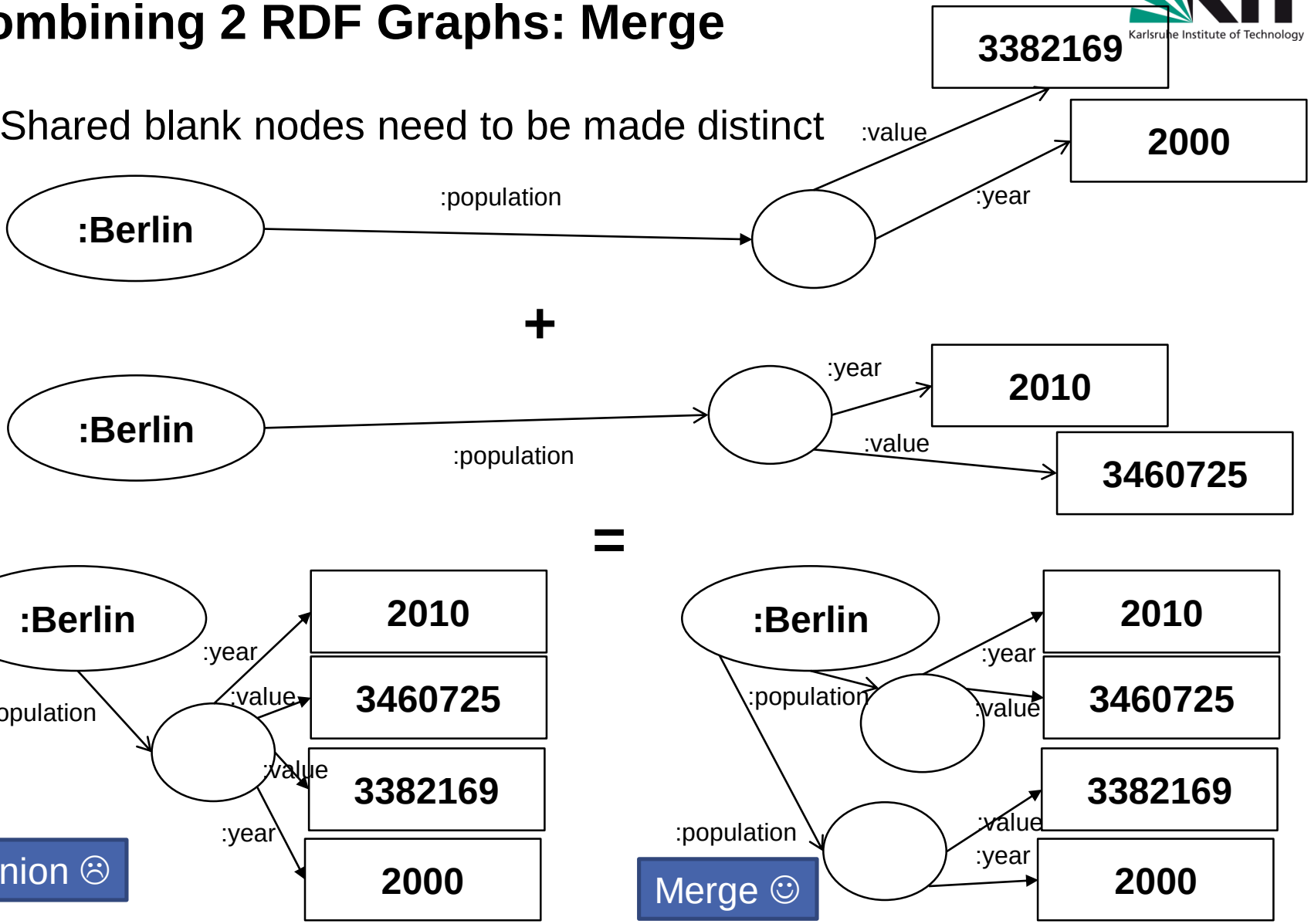
- No blank nodes in the RDF graphs → RDF graph combination trivial



- 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 .
```