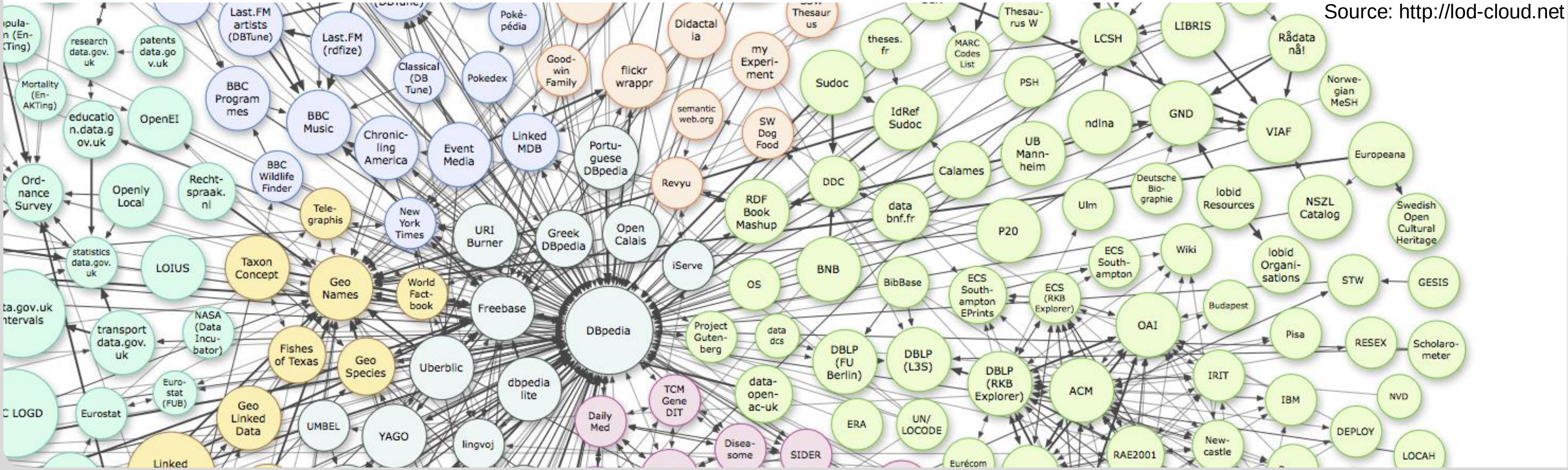
Karlsruhe Institute of Technology

Distributed Knowledge Graphs IV

Data Integration, Link Following, and Programming in Rules

Dr. Tobias Käfer

AI4INDUSTRY SUMMER SCHOOL


Source: http://lod-cloud.net

Creative Commons Licensing

- The slides have been prepared by Tobias Käfer, Andreas Harth, and Lars Heling

Agenda

Rules for:

- **Data Integration**
- Link following
- Programming

Web Standards

- Data providers publish data on web servers
- Data consumers access data with user agents

- Resource Description Framework
  - Graph-structured data: nodes (URIs, literals, blank nodes) and edges (URIs)
  - Interlink information (relationships)

- How can groups of people use RDF to
  - encode a shared understanding of a domain,
  - organise knowledge in a machine-processable way and
  - give meaning to data that can be exploited?

Ontology in Informatics

"An Ontology is a

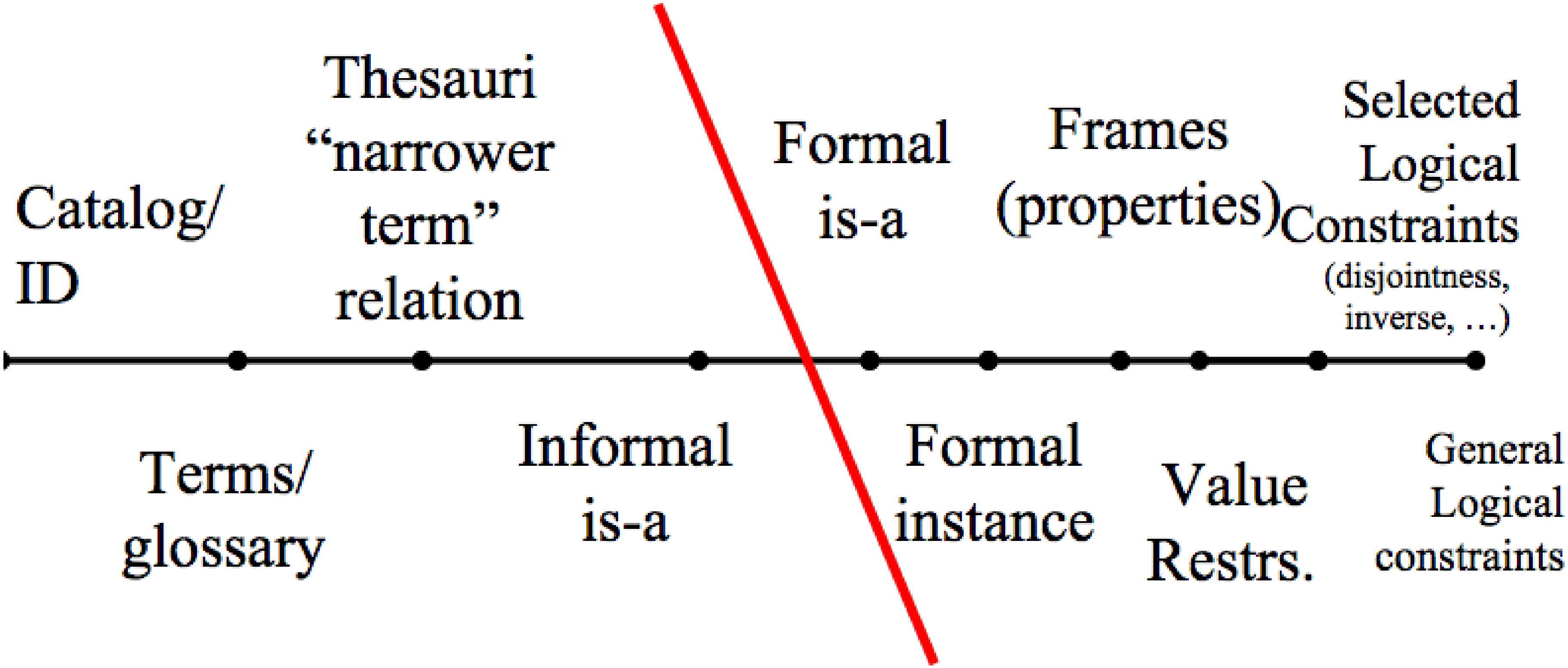
formal specification	> interpretable by machines
of a shared	> based on consensus
conceptualisation	> describes terminology
of a domain of interest"	> models a specific topic

Studer, Benjamins and Fensel (1998) based on Gruber (1993) and Borst (1997)

- An ontology is an engineering artefact, consisting of:
  - A specific vocabulary (set of terms - URIs and literals) used to describe a certain reality, plus
  - A set of explicit assumptions regarding the intended meaning of the vocabulary

Ontology Spectrum

From 99 AAAI panel with Gruninger, Lehmann, McGuinness, Ushold, Welty, 2000 Dagstuhl talk by McGuinness


Catalog/ ID
Thesauri "narrower term" relation
Formal is-a
Frames (properties)
Selected Logical Constraints (disjointness, inverse, …)
Terms/ glossary
Informal is-a
Formal instance
Value Restrs.
General Logical constraints

Less expressive

More expressive

Vocabularies and Vocabulary Descriptions/Ontologies

- Vocabularies are sets of terms, eg.
  - **Individuals:**
    Entities identified via a URI or blank node; a vocabulary description may include descriptions of identity (comes later)
  - **Classes:**
    Sets of individuals identified via URIs or blank nodes; a vocabulary description may include the characteristics of classes
  - **Properties:**
    Properties identified via URIs; a vocabulary description may include the characteristics of properties
- Ontologies (vocabulary descriptions) are collections of terms together with their (logically) defined meaning

Core Semantic Web Vocabularies

- To bootstrap meaning of vocabulary terms, we could use terms that are widely agreed; how about we use mathematics?
- The W3C standardised fundamental vocabularies (based on mathematics) that can be used to express other vocabularies.
- **RDF[1]:** We consider the RDF vocabulary, i.e., the URIs defined as part of the RDF W3C Recommendation.
- **RDFS[2]:** We examine RDF Schema, a simple ontology language that offers means to describe characteristics of classes and properties.

- Throughout the slides, assume the following prefix declarations:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix : <#> .
```

[1] https://www.w3.org/TR/rdf11-primer/
[2] https://www.w3.org/TR/rdf-schema/

Why Formal Semantics?

- After introduction of RDF and RDFS, critcism of tool developers: different tools were incompatible (despite the existing specification)

- E.g.:
  - Same RDF document
  - Same entailment relation
  - Different results

- Thus, a model-theoretic semantics was defined for entailment:
  - provides a formal specification of when truth is preserved by transformations of RDF or operations which derive RDF triples from other RDF triples (logical consequence).

A Classical Example for Entailment

- Premise: All men are mortal
- Premise: Socrates is a man
- Conclusion: Socrates is mortal

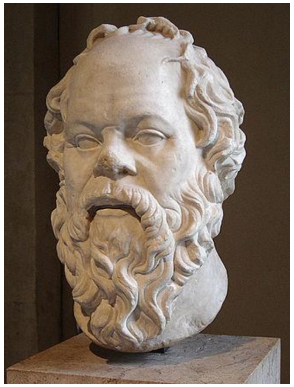


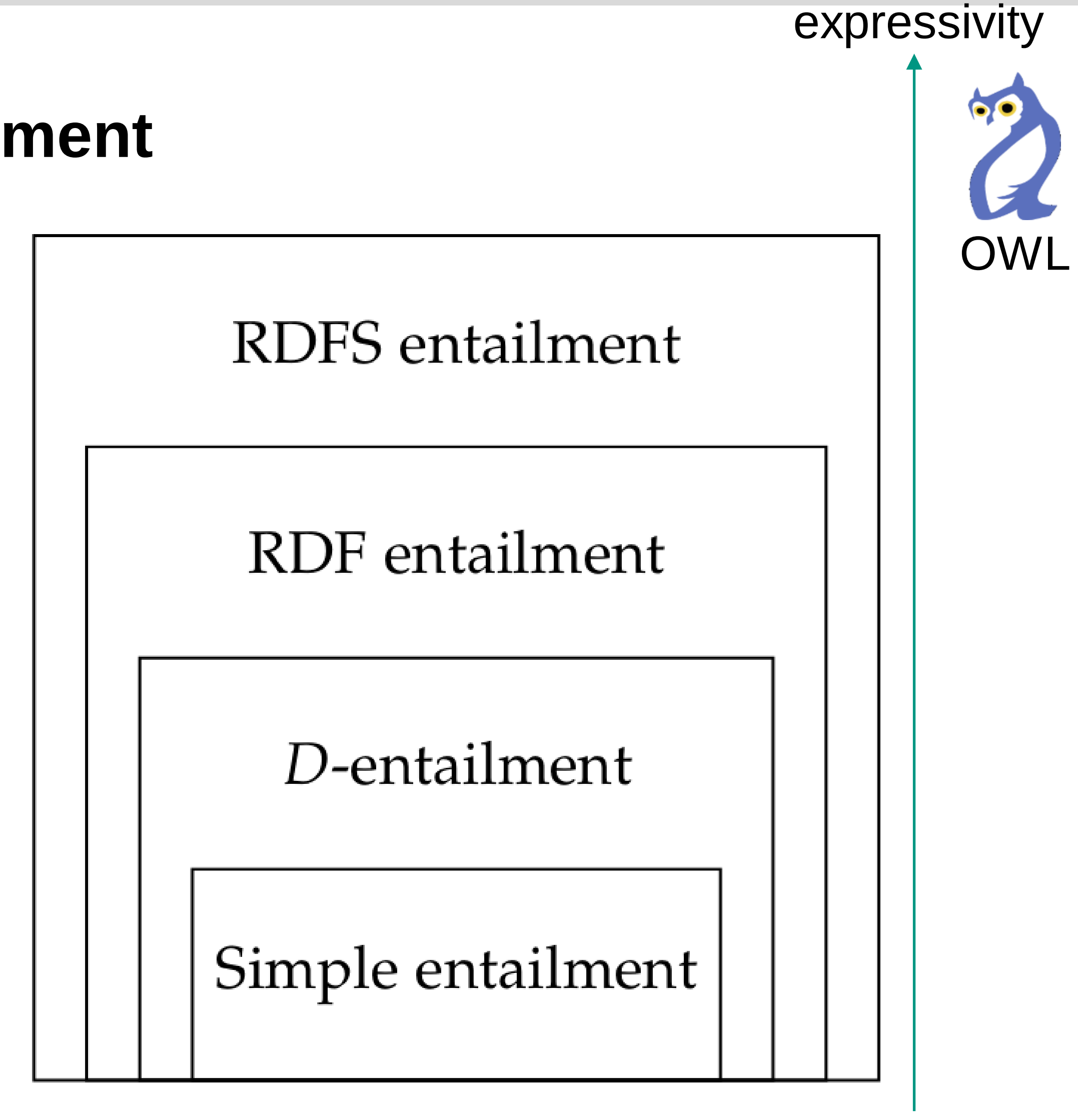
Photo from Wikipedia

- In RDF using RDFS vocabulary:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix : <#> .

:Man rdfs:subClassOf :Mortal .      # premise
:Socrates a :Man .                  # premise
```

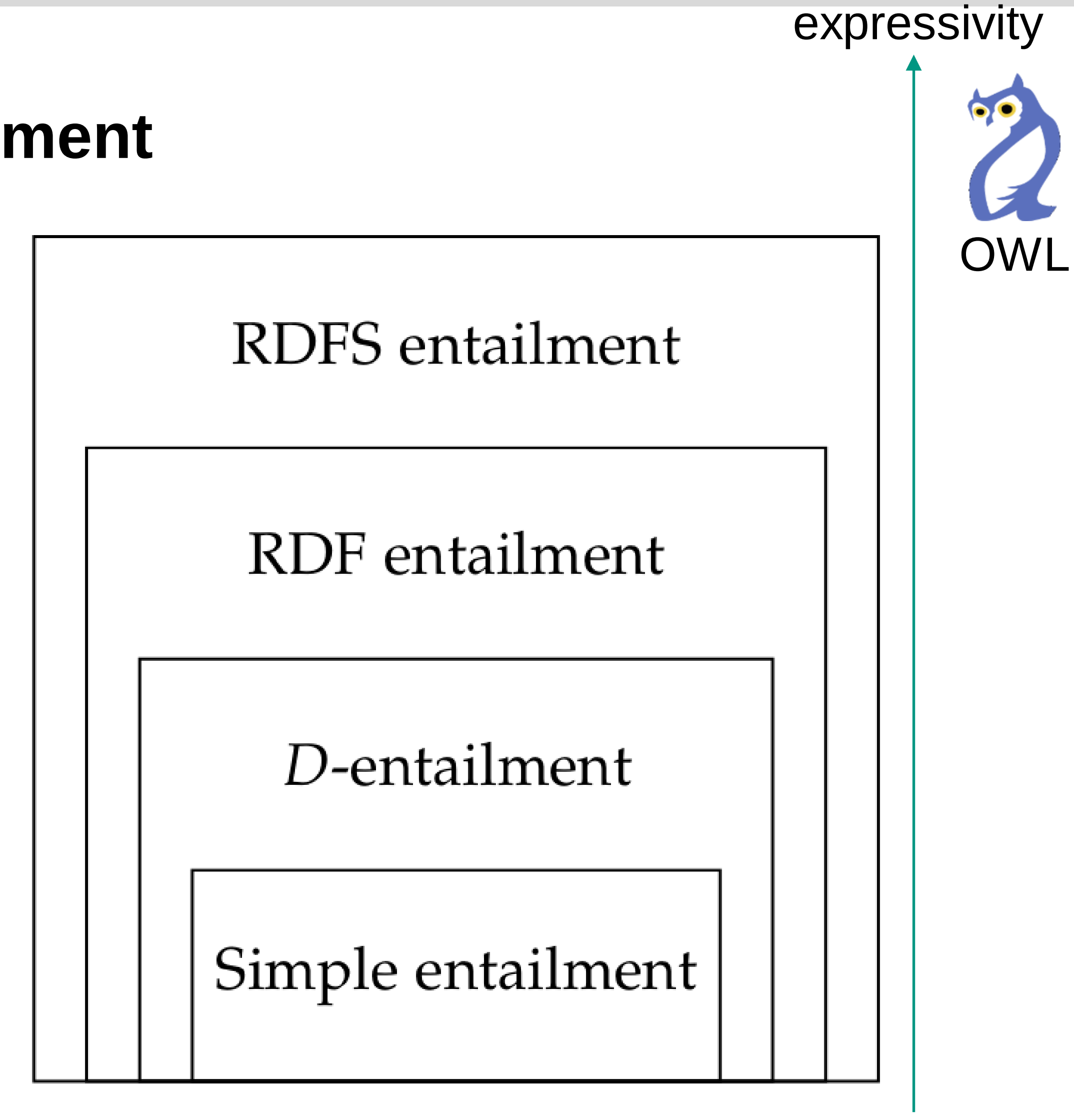
```
:Socrates a :Mortal .               # conclusion
```

Layered Entailment



- Higher expressivity → More logical conclusions (entailments) and higher computational complexity.
- Defined mathematically via sets and functions using model theory
- Rules as way to implement the mentioned entailment regimes.

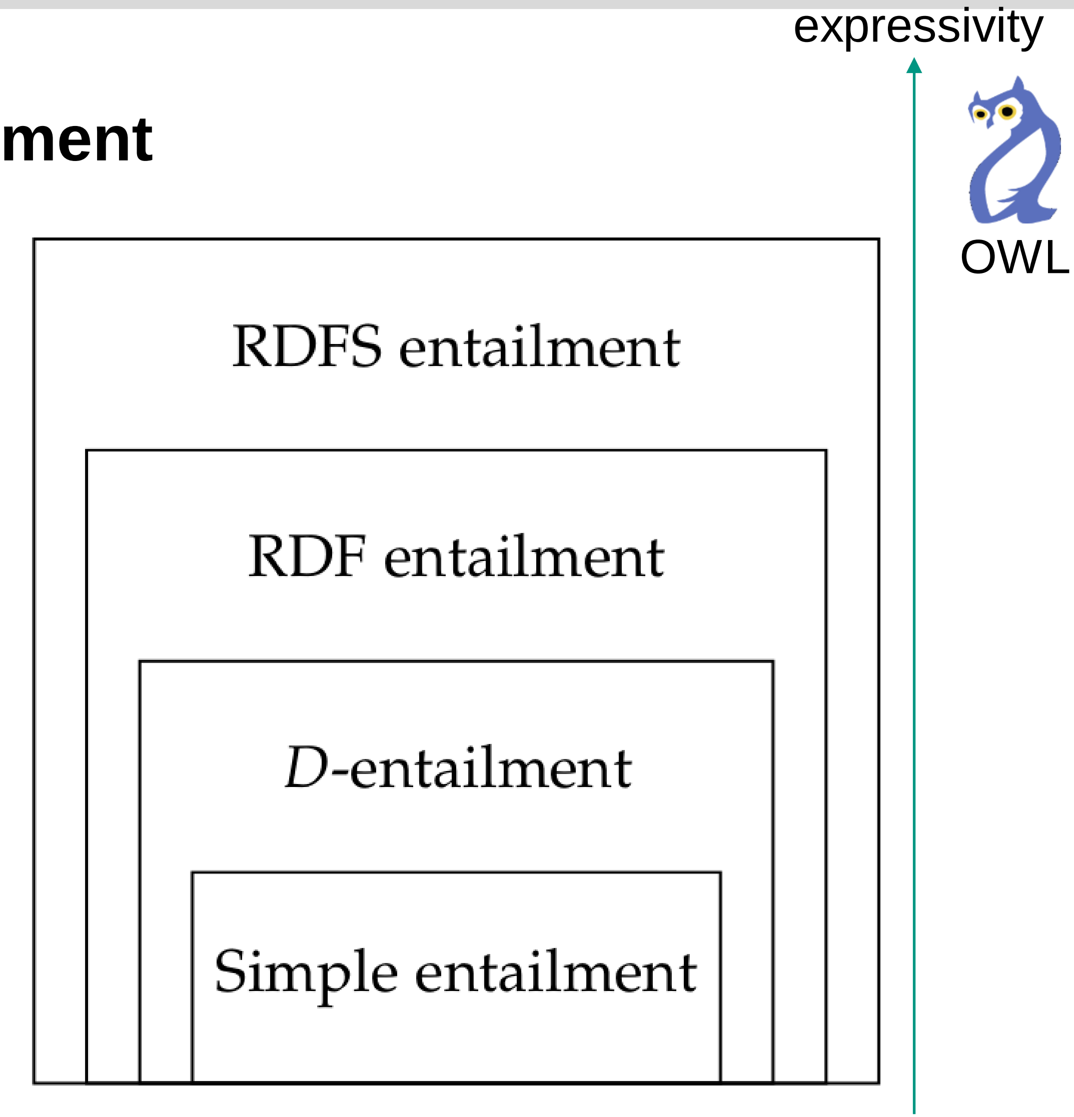
Layered Entailment



Interesting for bootstrapping the definitions via sets and functions

- Higher expressivity → More logical conclusions (entailments) and higher computational complexity.
- Defined mathematically via sets and functions using model theory
- Rules as way to implement the mentioned entailment regimes.

Layered Entailment



Tells you how to formally define typed literals and when values are the same

- Higher expressivity → More logical conclusions (entailments) and higher computational complexity.
- Defined mathematically via sets and functions using model theory
- Rules as way to implement the mentioned entailment regimes.

RDF VOCABULARY AND ENTAILMENT

RDF Vocabulary

- The RDF vocabulary allows to make basic statements about resources and triples
- 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`		

Formal Instances (rdf:type)

- The URI `rdf:type` allows to specify that a resource is an instance of something
- For example, the following describes `:Berlin` as being a `: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

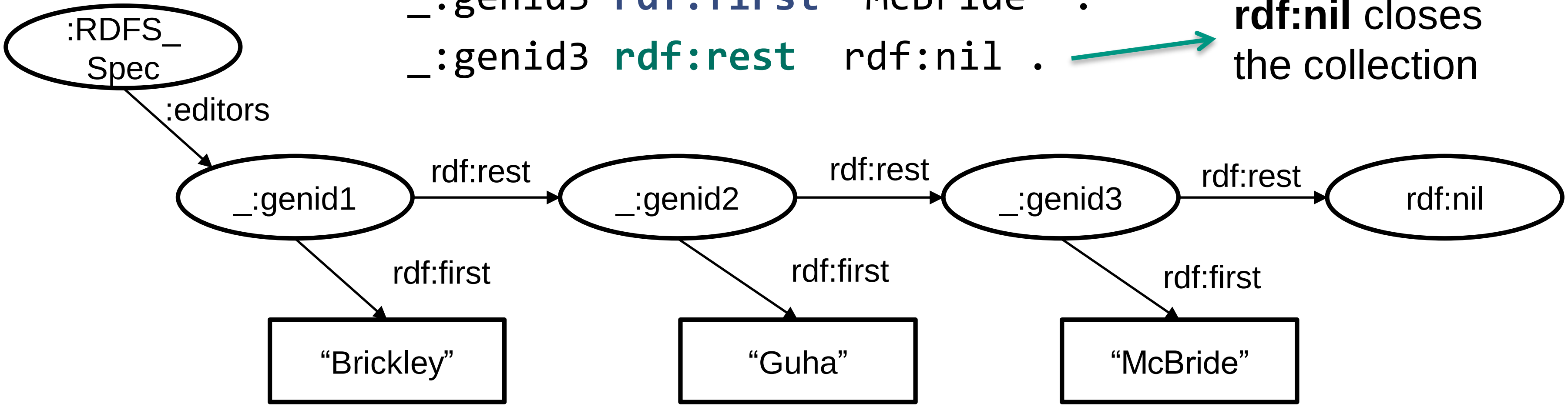
```
:p rdf:type rdf:Property .
```

Collections aka rdf:Lists

- A 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 Axiomatic Triples

- What is an axiom?
  - A self-evident or universally recognised truth [1]
  - An established rule, principle, or law [1]
- The following triples have to be true in any RDF interpretation, by definition:

```
rdf:type rdf:type rdf:Property .
rdf:subject rdf:type rdf:Property .
rdf:predicate rdf:type rdf:Property .
rdf:object rdf:type rdf:Property .
rdf:first rdf:type rdf:Property .
rdf:rest rdf:type rdf:Property .
rdf:value rdf:type rdf:Property .
rdf:nil rdf:type rdf:List .
rdf:_1 rdf:type rdf:Property .
rdf:_2 rdf:type rdf:Property .
...
```

Since the elements of a container may be infinite, the application of the axiomatic triples results in an infinite interpretation

[1] http://www.thefreedictionary.com/axiom

RDF Entailment Patterns

- The following entailment patterns can be used as an easy way to apply the RDF entailment rules to a graph
- Variables are denoted with a “`?`” (as in SPARQL)
- The patterns are applied by assigning values to the variables in the “If” statement and adding (inferring) the “Then” statement
- Patterns*:

	If …	**Then …**
rdfD1	`?x ?p "sss"^^ddd .`	`?x ?p _:n . _:n rdf:type ddd .`
rdfD2	`?x ?p ?y .`	`?p rdf:type rdf:Property .`

- Alternative pattern to rdfD1 (assuming generalised RDF)

	If …	**Then …**
GrdfD1	`?x ?p "sss"^^ddd .`	`"sss"^^ddd rdf:type ddd .`

- For the following examples we consider our graph: http://example.org/cities.ttl

* `"sss"` represents some Unicode string

RDFS VOCABULARY AND ENTAILMENT

RDFS Intuition and Vocabulary

- The RDFS vocabulary allows to make statements about classes of things and properties and to provide documentation to resources
- RDFS entailment is a lot about the semantics of those classes and properties
- RDFS terms are:

Properties:

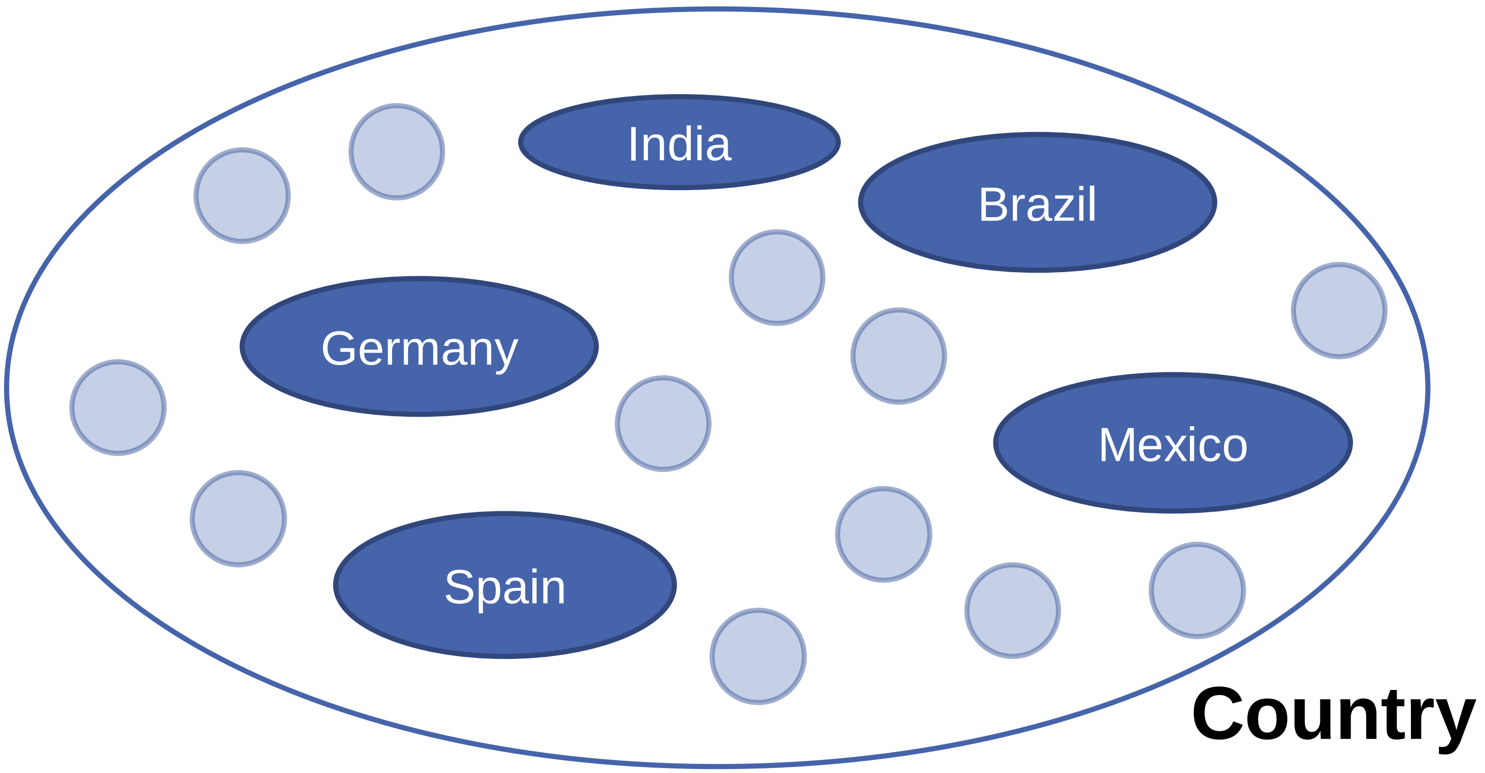
rdfs:domain
rdfs:range
rdfs:subClassOf
rdfs:subPropertyOf
rdfs:member
rdfs:comment
rdfs:seeAlso
rdfs:isDefinedBy
rdfs:label

Classes:

rdfs:Resource
rdfs:Literal
rdfs:Datatype
rdfs:Class
rdfs:Container
rdfs:ContainerMembershipProperty

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

Classes – Analogy to Set Theory

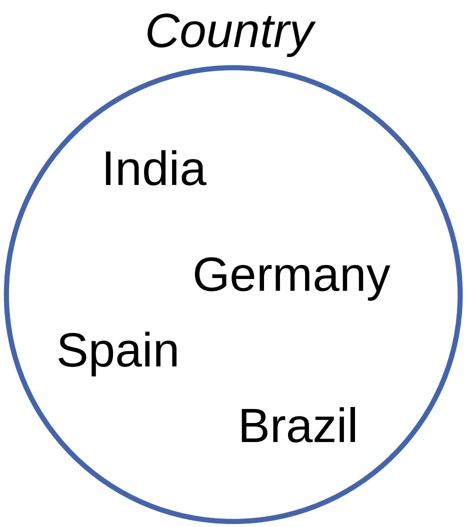


- Individuals represent elements of a set
- Classes represent a set that is identified via a URI or a blank node

Classes: Example (1)

- To define the class:
  - `rdf:type rdfs:Class`
- To relate instances to the class:
  - `rdf:type`

The class of *countries*



URI of the class

```
:Country rdf:type rdfs:Class .
```

Instances of the class

```
:India   rdf:type  :Country .
:Germany rdf:type  :Country .
:Spain   rdf:type  :Country .
:Brazil  rdf:type  :Country .
```

Class Hierarchies

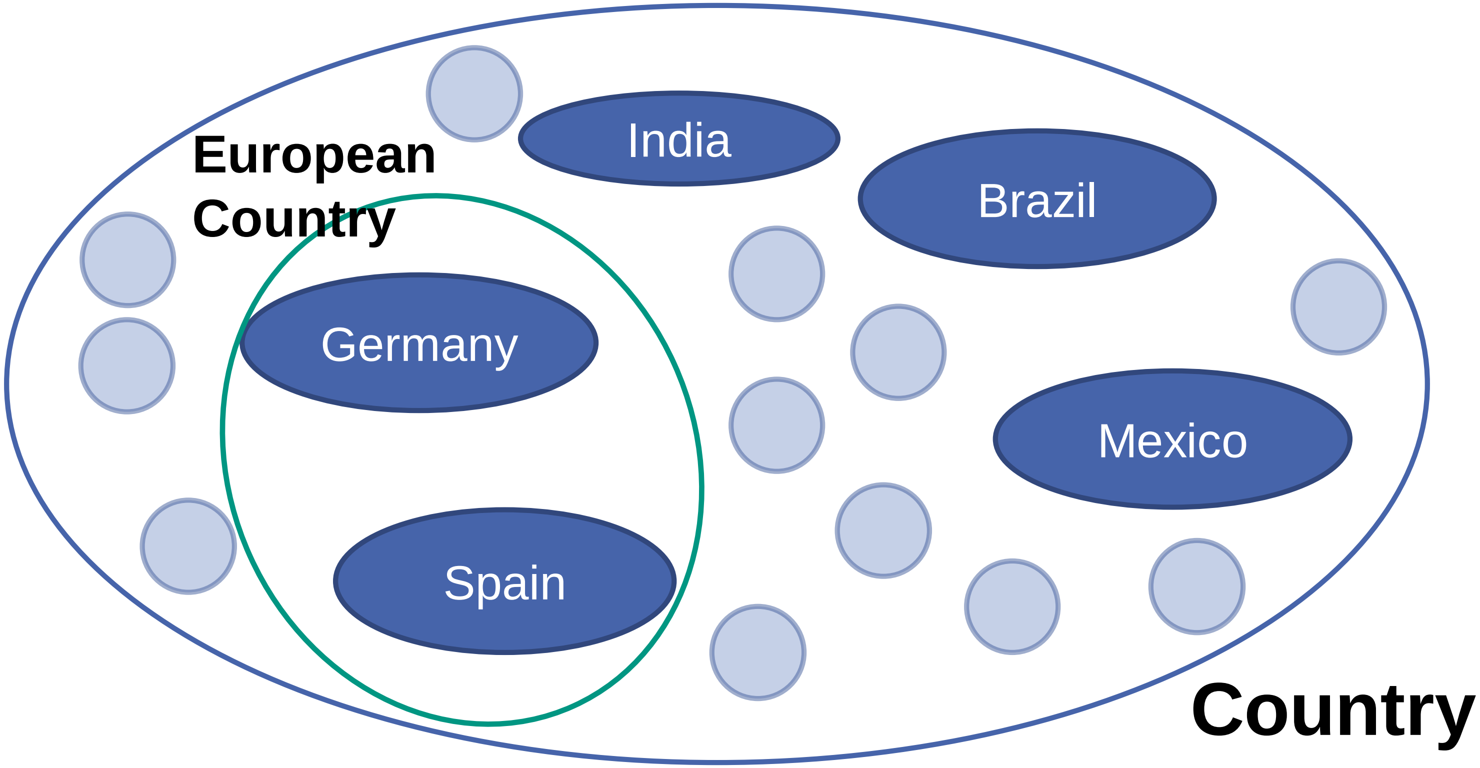
- Given several classes, we can specify a hierarchical relationship between them: the subclass relation

- In RDFS, a class may have several subclasses, and a class can be a subclass of several (super)classes

- Example:
  - We have two classes: `:Country` and `:EuropeanCountry`
  - We want to say that everything that is a European country is also a country
  - That is, `:EuropeanCountry` is a subclass of `:Country`
  - We use `rdfs:subClassOf` to specify the subclass relationship:

```
:EuropeanCountry rdfs:subClassOf :Country .
```

Class Hierarchies – Analogy to Set Theory



- `rdf:type` corresponds to ∈
- `rdfs:subClassOf` corresponds to ⊆

RDFS Axiomatic Triples

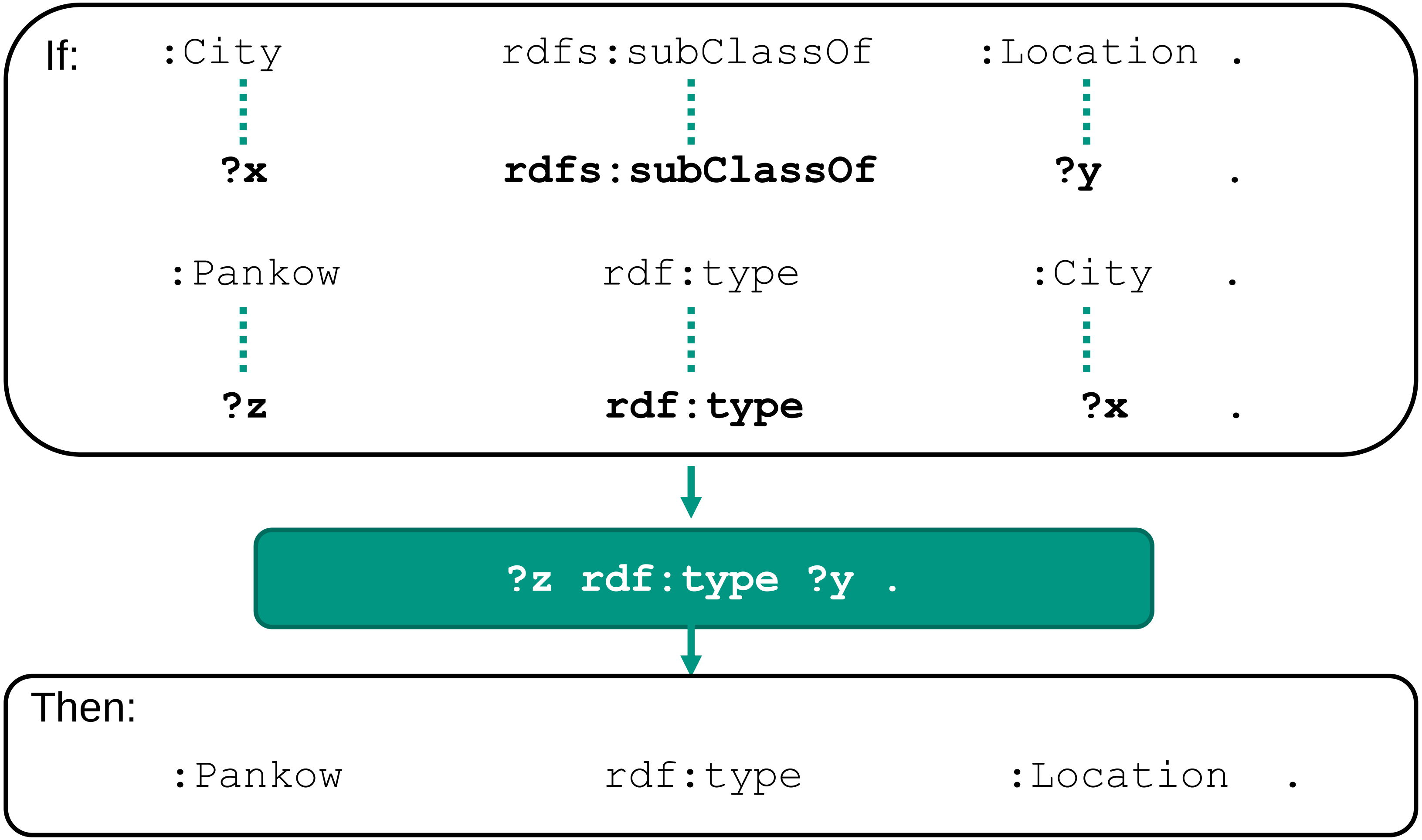
```
rdf:type rdfs:domain rdfs:Resource ; rdfs:range rdfs:Class .
rdfs:domain rdfs:domain rdf:Property ; rdfs:range rdfs:Class .
rdfs:range rdfs:domain rdf:Property ; rdfs:range rdfs:Class .
rdfs:subPropertyOf rdfs:domain rdf:Property ; rdfs:range rdf:Property .
rdfs:subClassOf rdfs:domain rdfs:Class ; rdfs:range rdfs:Class .
rdf:subject rdfs:domain rdf:Statement ; rdfs:range rdfs:Resource .
rdf:predicate rdfs:domain rdf:Statement ; rdfs:range rdfs:Resource .
rdf:object rdfs:domain rdf:Statement ; rdfs:range rdfs:Resource .
rdfs:member rdfs:domain rdfs:Resource ; rdfs:range rdfs:Resource .
rdf:first rdfs:domain rdf:List ; rdfs:range rdfs:Resource .
rdf:rest rdfs:domain rdf:List ; rdfs:range rdf:List .
rdfs:seeAlso rdfs:domain rdfs:Resource ; rdfs:range rdfs:Resource .
rdfs:isDefinedBy rdfs:domain rdfs:Resource ; rdfs:range rdfs:Resource .
rdfs:comment rdfs:domain rdfs:Resource ; rdfs:range rdfs:Literal .
rdfs:label rdfs:domain rdfs:Resource ; rdfs:range rdfs:Literal .
rdf:value rdfs:domain rdfs:Resource ; rdfs:range rdfs:Resource .

rdf:Alt rdfs:subClassOf rdfs:Container .
rdf:Bag rdfs:subClassOf rdfs:Container .
rdf:Seq rdfs:subClassOf rdfs:Container .
rdfs:ContainerMembershipProperty rdfs:subClassOf rdf:Property .

rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso .

rdfs:Datatype rdfs:subClassOf rdfs:Class .

rdf:_1 a rdfs:ContainerMembershipProperty ; rdfs:domain rdfs:Resource ; rdfs:range rdfs:Resource .
rdf:_2 a rdfs:ContainerMembershipProperty ; rdfs:domain rdfs:Resource ; rdfs:range rdfs:Resource .
...
```

RDFS Entailment Patterns

	If…	Then…
rdfs1	Any URI ddd in D	ddd rdf:type rdfs:Datatype .
rdfs2	?p rdfs:domain ?x . ?y ?p ?z .	?y rdf:type ?x .
rdfs3	?p rdfs:range ?x . ?y ?p ?z .	?z rdf:type ?x .
rdfs4a	?x ?p ?z .	?x rdf:type rdfs:Resource .
rdfs4b	?y ?p ?z .	?z rdf:type rdfs:Resource .
rdfs5	?x rdfs:subPropertyOf ?y . ?y rdfs:subPropertyOf ?z .	?x rdfs:subPropertyOf ?z .
rdfs6	?x rdf:type rdf:Property .	?x rdfs:subPropertyOf ?x .
rdfs7	?p2 rdfs:subPropertyOf ?p1 . ?x ?p2 ?y.	?x ?p1 ?y .
rdfs8	?x rdf:type rdfs:Class .	?x rdfs:subClassOf rdfs:Resource .
rdfs9	?x rdfs:subClassOf ?y . ?z rdf:type ?x .	?z rdf:type ?y .
rdfs10	?x rdf:type rdfs:Class .	?x rdfs:subClassOf ?x .
rdfs11	?x rdfs:subClassOf ?y . ?y rdfs:subClassOf ?z .	?x rdfs:subClassOf ?z .
rdfs12	?x rdf:type rdfs:ContainerMembershipProperty.	?x rdfs:subPropertyOf rdfs:member .
rdfs13	?x rdf:type rdfs:Datatype .	?x rdfs:subClassOf rdfs:Literal .

RDFS Entailment Patterns – rdfs9

- Example:



If: :City rdfs:subClassOf :Location .

?x rdfs:subClassOf ?y .

:Pankow rdf:type :City .

?z rdf:type ?x .

?z rdf:type ?y .

Then:

:Pankow rdf:type :Location .

MORE EXPRESSIVE ENTAILMENT REGIMES

Extending RDFS with other useful features

- OWL is a fairly expressive ontology language

- RDFS plus, RDFS 3.0, OWL LD „extend“ RDFS entailment with the semantics of some terms from OWL such as:
  - owl:sameAs
  - owl:equivalentProperty
  - owl:inverseOf
  - …

IMPLEMENTING ENTAILMENT

Approaches for Evaluating Entailment Patterns

Examples for when users are interested in the derived knowledge

- Queries, eg. of downstream applications
- Conditions for actions outside the realm of the

Approaches:

- Materialization / forward chaining
- Query rewriting / backward chaining
- Hybrid approaches

Algorithm for Materialisation: Extend the Graph with Inferred Triples

```
Require: assertions ▷ Graph
Require: rules ▷ Derivation rules
var data, oldData: set<triple>
var fixpointReached: boolean
data.clear()
data.add(assertions)
repeat ▷ Loop for determining the fixpoint
    fixpointReached ← true
    for rule : rules do
        if rule.matches(data) then
            oldData = data.copy()
            if rule.type==derivation then
                data.add(rule.match(data).data)
            end if
            if ! data.copy().remove(oldData).isEmpty() then
                fixpointReached ← false
            end if
        end if
    end for
until fixpointReached
```

Notation3 Rule Syntax

- We introduce Notation3 (N3), a superset of Turtle syntax

- N3 extends the RDF data model with
  - variables (prefixed with a `?`) and
  - graph quoting (via `{}`) for subject and object of a triple

- Together with a URI for implication (`<http://www.w3.org/2000/10/swap/log#implies>`, shortcut: `=>`), we can encode rules in N3 syntax.

Notation3 Derviation Rules

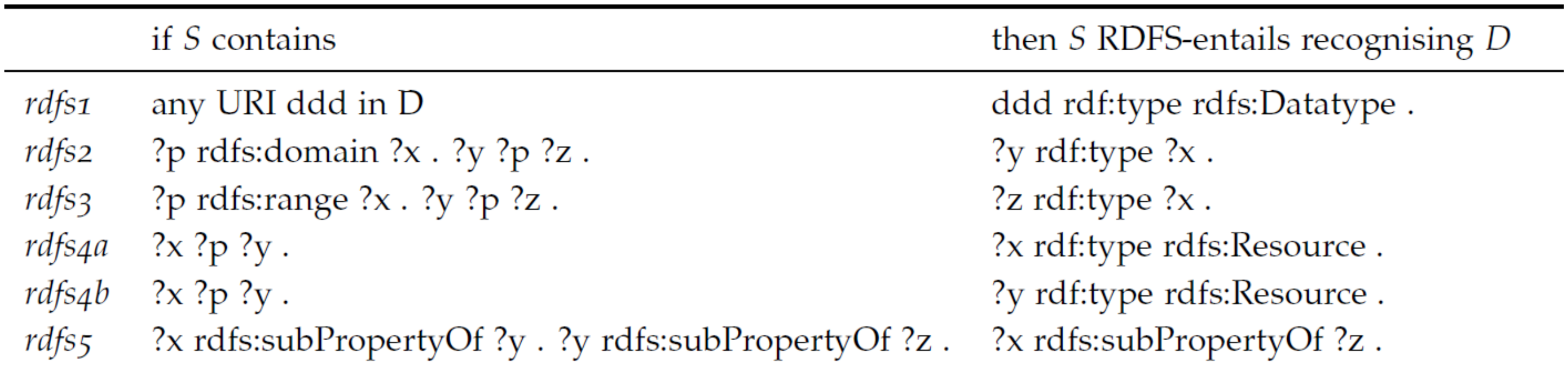
- A N3 rule is of the form `{` body `} => {` head `} .`

- The **body** of a rule (**the „if“ part**) is also called antecedent
- The **head** of a rule (**the „then“ part**) is also called consequent

- The body is a set of triple patterns: a BGP

- The head is a graph template

Example: RDFS Entailment Patterns as Rules



	if *S* contains	then *S* RDFS-entails recognising *D*
rdfs1	any URI ddd in D	ddd rdf:type rdfs:Datatype .
rdfs2	?p rdfs:domain ?x . ?y ?p ?z .	?y rdf:type ?x .
rdfs3	?p rdfs:range ?x . ?y ?p ?z .	?z rdf:type ?x .
rdfs4a	?x ?p ?y .	?x rdf:type rdfs:Resource .
rdfs4b	?x ?p ?y .	?y rdf:type rdfs:Resource .
rdfs5	?x rdfs:subPropertyOf ?y . ?y rdfs:subPropertyOf ?z .	?x rdfs:subPropertyOf ?z .

Entailment pattern rdfs5 as derivation rule:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

{ ?x rdfs:subPropertyOf ?y .
  ?y rdfs:subPropertyOf ?z . } => { ?x rdfs:subPropertyOf ?z . } .
```

Exercise: Query Evaluation with Materialization

- Given the following RDF graph G available at http://example.org/persons and the SPARQL expression E. Assume all the prefix definitions.

```
:Magneto a foaf:Person ;
          foaf:name "Max Eisenhardt" .
foaf:Person rdfs:subClassOf foaf:Agent .
foaf:Person owl:equivalentClass dbo:Person .
```

- Query Q:

```
SELECT ?p WHERE { ?p a foaf:Agent }
```

- Entailment regime R with the following set of rules:

```
{ { ?x owl:equivalentClass  ?y . } => { ?y owl:equivalentClass ?x . },
  { ?x owl:equivalentClass  ?y . ?a rdf:type ?x . } => { ?a rdf:type ?y .} }
```

- Materialise R on the graph G and evaluate Q.

Agenda

Rules for:

- Reasoning
- **Link following**
- Programming

How to Combine Link-Following and Querying?

- The Linked Data principles point towards combining web architecture with knowledge representation

- But all the bits and pieces we have seen so far do not fit yet:

- We can dereference URIs of things via HTTP, view the resulting RDF and follow links (e.g., in the RDF browser)

OR

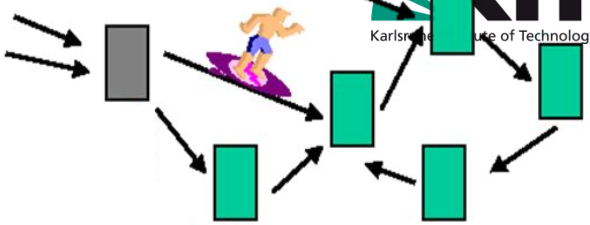
- We can query RDF documents with SPARQL given a fixed set of URIs to documents in FROM/FROM NAMED clauses

BUT

- How how do we query Linked Data while following links?

General User Agent Model

- Characteristics of a generic user agent on the web (e.g., web browser):



http://slideplayer.com/slide/8080871/

1. The user agent starts its interaction based on a specific seed URI

2. The user agent performs HTTP requests on URIs and parses the response

3. Based on the response the user agent has one or multiple choices as to which interaction to perform next

4. The user agent decides which link to follow and initiates a new request

Reduction to What we Learnt: Crawl-Index-Serve

- Crawl-index-serve architecture for Linked Data:
  - Crawl Linked Data (on the level of documents, parse RDF into quads), specify the amount of hops for expansion
  - Load the resulting RDF Dataset (quads) into a SPARQL store
  - Serve query solutions from the SPARQL store

- Materialising the data (crawling, indexing) takes time
- Indexes of Linked Data get outdated [1]
- Indiscriminate expansion of links
- Requires many systems (crawler, SPARQL store), server capacity
- Possibly too much overhead if users are interested in the solution to a single query

- **How about more clever user agents? That run on people's computers?**
- **That access live data?**

[1] Käfer, Umbrich, Abdelgayed, O'Byrne, Hogan: Observing Linked Data Dynamics. Proc. 11th Extended Semantic Web Conference (2013).

Linked Data Principles[1]

Tim Berners-Lee presenting Linked Data. TED CC-BY-ND

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)
4. Include links to other URIs, so that they can discover more things.

[1] http://www.w3.org/DesignIssues/LinkedData.html

Two Perspectives on the Linked Data Principles

Server (Publisher)

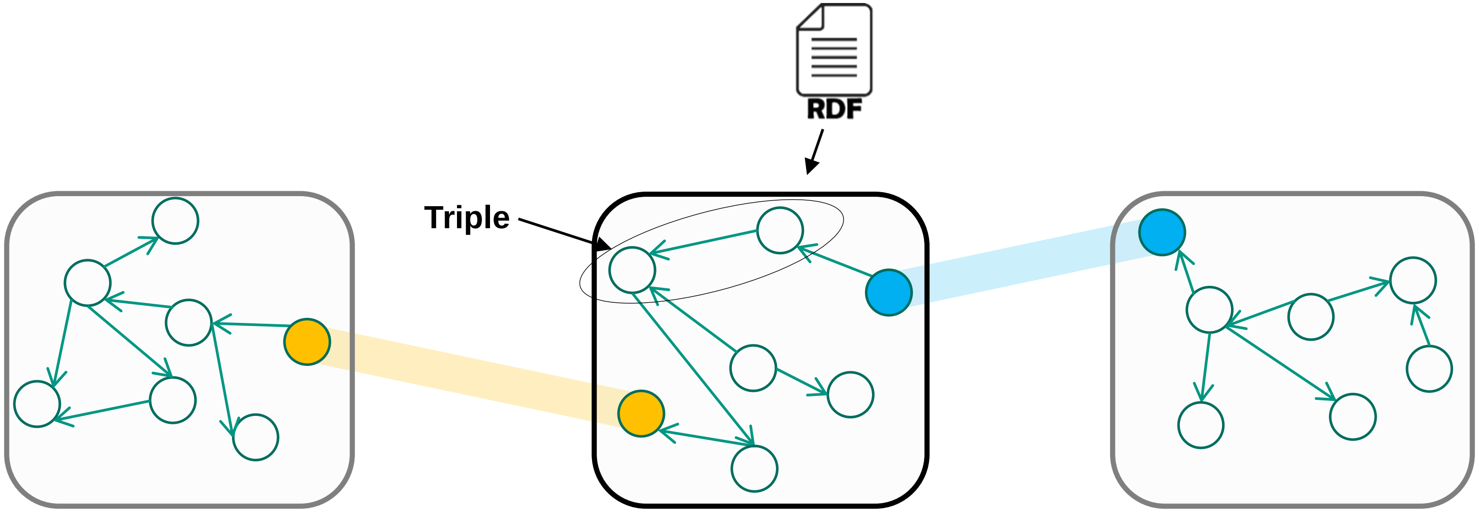
1. Coin URIs to name things. ✓
2. Use a HTTP server to provide access to documents. ✓
3. Upon receiving a request for a URI, the server returns useful information (about the URI in the request) in RDF and RDF Schema. ✓
4. The “useful information” the server returns in the RDF document includes links to other URIs (on other servers). ✓

User Agent (Consumer)

1. Assume URIs as names for things. ✓
2. User agents look up HTTP URIs. ✓
3. User agents process RDF/RDFS documents containing useful information and provide the ability to evaluate SPARQL queries. ✓
4. User agents can discover more things via accessing links to other URIs. ✘

Operating on a Fixed RDF Dataset

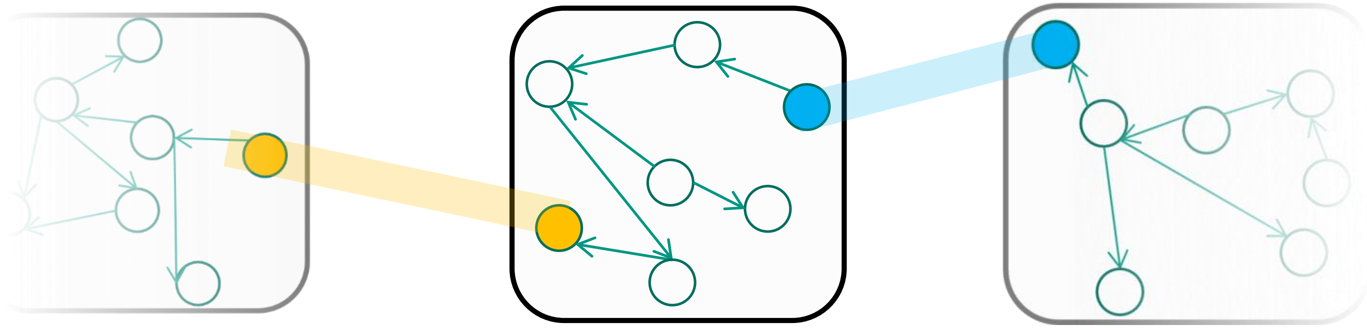
- Until now, both in querying and with entailment, we have assumed that the data over which we operate is fixed at the beginning of the processing.
- That is, we have assumed a fixed RDF Dataset.



Operating on the Web as RDF Dataset

- We would like to use the entire Linked Data web, i.e., a huge RDF dataset *Web*, as basis for querying.
- But the web is too big; downloading the entire web is impractical.

- One of the core features of the web are hyperlinks.
- A user agent starts from an entry point and then follows links.
- Following links can lead to hitherto unknown servers, with unknown data of unknown schema.



- How can we specify a (finite) RDF dataset in a flexible way?

Dereferencing URIs[1]

- We define ways for accessing RDF graphs published on the web as Linked Data

- Linked Data provides a combination of knowledge representation language (RDF, RDFS) and web architecture (HTTP)

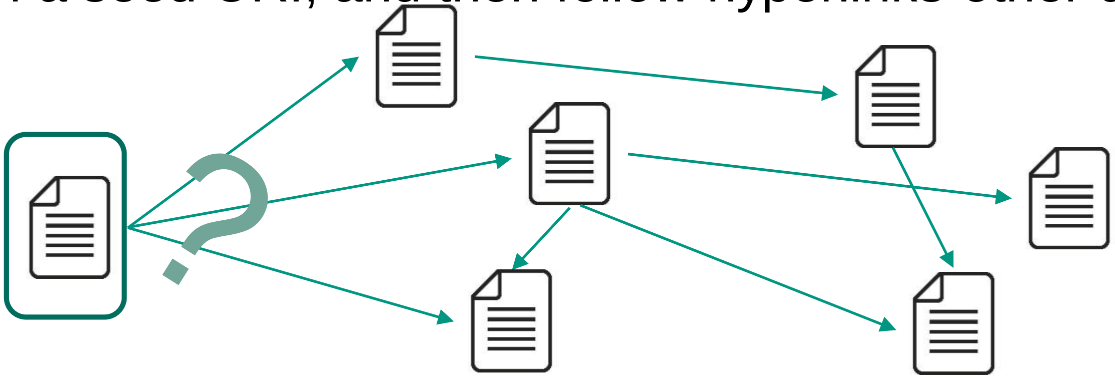
- A key characteristic of Linked Data is the tight connection between an identifier and a source, i.e., the name for a thing[2] is associated with the document where one can find related information

[1] See also in Chapter 2

[2] “non-information” resource, not defined in any RFC, which only know “other resources” and “information resources”.

Motivation for Request Rules

- We want to specify an RDF dataset constructed during query evaluation
- Start with a seed URI, and then follow hyperlinks other data sources



- Given a set of links within a dataset we need to specify:
  - Which links to follow?
  - Order of following links?
  - How far to follow links?

➡ Request rules as a way to specify traversal

Representing HTTP Requests in RDF

- To model HTTP requests in RDF we require a vocabulary for HTTP requests (and headers)
- Namespace for the core terms of HTTP vocabulary[1] in RDF:

```
http://www.w3.org/2011/http#
```

- We also make use of a vocabulary for HTTP methods and HTTP headers
- Using the HTTP vocabulary, we are able to represent any kind of HTTP-interaction using RDF

[1] http://www.w3.org/TR/HTTP-in-RDF10/

HTTP Vocabulary: Example

- Let us consider the simple request:

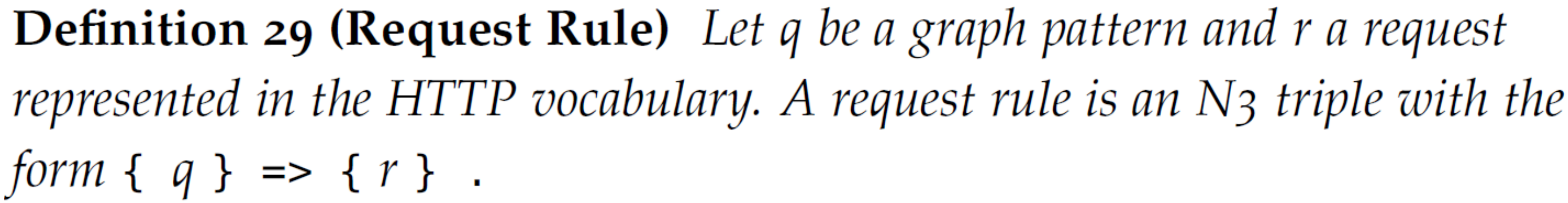
```
GET     /article/420 HTTP/1.1
Host:   example.org
Accept: text/turtle
```

- Represented using the HTTP vocabulary:

```
@prefix http: <http://www.w3.org/2011/http#> .
@prefix httpm: <http://www.w3.org/2011/http-methods#> .
@prefix httph: <http://www.w3.org/2011/http-headers#> .

[ ]      a                           http:Request;
         http:requestURI             "/article/420";
         http:httpVersion   "1.1";
         http:mthd          httpm:GET;
         http:headers           (    [ http:hdrName      httph:host ;
                                       http:fieldValue   "example.org" ]
                                     [ http:hdrName      httph:accept ;
                                       http:fieldValue   "text/turtle" ] ) .
```

Syntax of Request Rules in Notation3

- Request with both fixed and variable request targets can appear as the head of a request rule



Definition 29 (Request Rule) *Let q be a graph pattern and r a request represented in the HTTP vocabulary. A request rule is an N3 triple with the form* { *q* } => { *r* } .

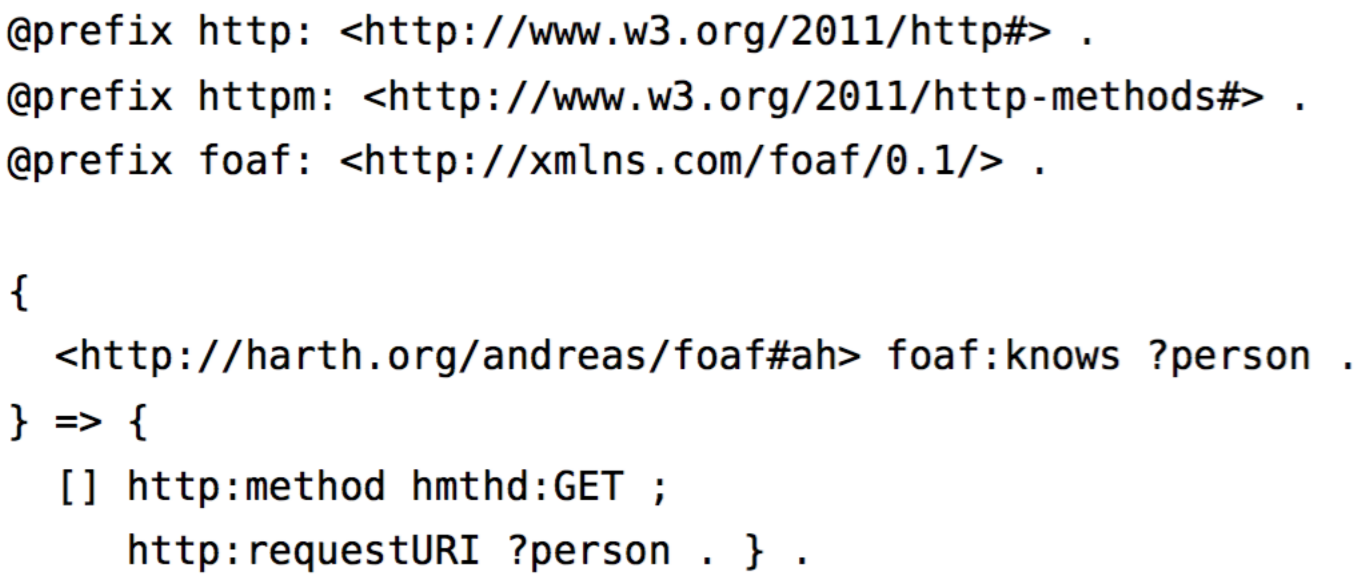
- Form:

$$\overbrace{\texttt{\{ graph pattern \}}}^{\textit{body}}\ \texttt{=>}\ \overbrace{\texttt{\{ request template \}}}^{\textit{head}}\ \texttt{.}$$

- Properties:
  - Existential: *head* contains blank nodes
  - Safe: all variable are part of both *head* and *body* of the rule

Request Rule – Example 1

- Request URIs of people that Andreas knows



```
@prefix http: <http://www.w3.org/2011/http#> .
@prefix httpm: <http://www.w3.org/2011/http-methods#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

{
  <http://harth.org/andreas/foaf#ah> foaf:knows ?person .
} => {
  [] http:method hmthd:GET ;
     http:requestURI ?person . } .
```

Request rules allow for fine-grained manner to determine which resources to retrieve and which links to follow

Request Rule – Example 2

- The following rule dereferences all class URIs that occur in the data:

```
@prefix http: <http://www.w3.org/2011/http#> .
@prefix httpm: <http://www.w3.org/2011/http-methods#> .

{
  ?s a ?c .
} => {
  [] http:method httpm:GET ;
     http:requestURI ?c .
} .
```

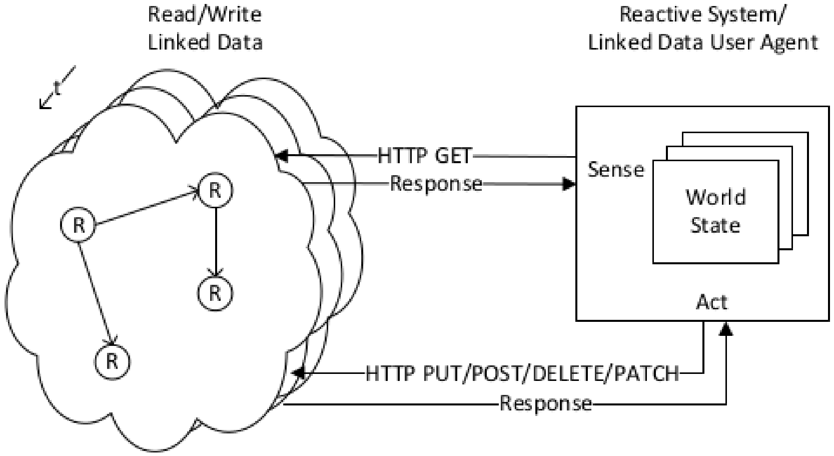
Algorithm for Constructing an RDF Dataset Based on Request Rules (Integrated using Derivation Rules)

```
Require: assertions ▷ Graph
Require: rules ▷ Derivation and GET request rules
var data, oldData: set<triple>
var fixpointReached: boolean
data.clear()
data.add(assertions)
repeat ▷ Loop for determining the fixpoint
    fixpointReached <- true
    for rule : rules do
        if rule.matches(data) then
            oldData = data.copy()
            if rule.type==derivation then
                data.add(rule.match(data).data)
            else ▷ So the rule must be an interaction rule
                if rule.match(data).request.type==GET then
                    data.add(rule.match(data).request.execute())
                end if
            end if
            if ! data.copy().remove(oldData).isEmpty() then
                fixpointReached <- false
            end if
        end if
    end for
until fixpointReached
```

Linked Data-Fu Overview

- Approach for accessing, integrating, querying and manipulating web data
- The language allows developers to specify interactions using rules
- The engine executes desired interactions in parallel

- **Derivation rules** support reasoning constructs, e.g., transitivity, reflexivity of properties
- **Request rules** specify how and when to interact with resources, i.ie., retrieve the state of resources (sense) or manipulate the state of resources (act)



http://linked-data-fu.github.io/

Stadtmüller, Speiser, Harth, Studer: Data-Fu: a language and an interpreter for interaction with read/write linked data. WWW 2013

Linked Data-Fu

- A system to
  - execute programs with request rules to construct a RDF dataset
  - apply entailment patterns expressed in Notation3
  - process SPARQL queries, including entailment, over the RDF dataset created via link-following
- Linked Data-Fu programs run as user agents
  - Request rules can specify link-following based on HTTP GET requests
  - With allowing additional HTTP requests (PUT, POST, DELETE), the user agents can effect change in resource state

Agenda

Rules for:

- Reasoning
- Link following
- **Programming**

From Linked Data to Read-Write Linked Data

- With HTTP GET requests, one can implement systems that answer queries on data published on the web

- But HTTP has more request methods:
  - HTTP POST is used on the web to handle HTML forms and can be used to create resources
  - HTTP PUT can be used to overwrite resource state
  - HTTP DELETE can be used to delete resources

- With POST, GET, PUT and DELETE, one can implement applications that require CRUD (create-read-update-delete) operations on web architecture

Putting the *Web* back into the Semantic Web

- Linked Data Platform (W3C recommendation specified led by IBMers)
  - Read-Write interaction with Linked Data resources and collections of Linked Data resources
- Solid: Social Linked Data
  - Conventions and tools (mainly JavaScript) for building decentralised social applications based on Read-Write Linked Data
  - Users store personal data in "pods" (personal online data stores) hosted wherever the user desires
- Web of Things

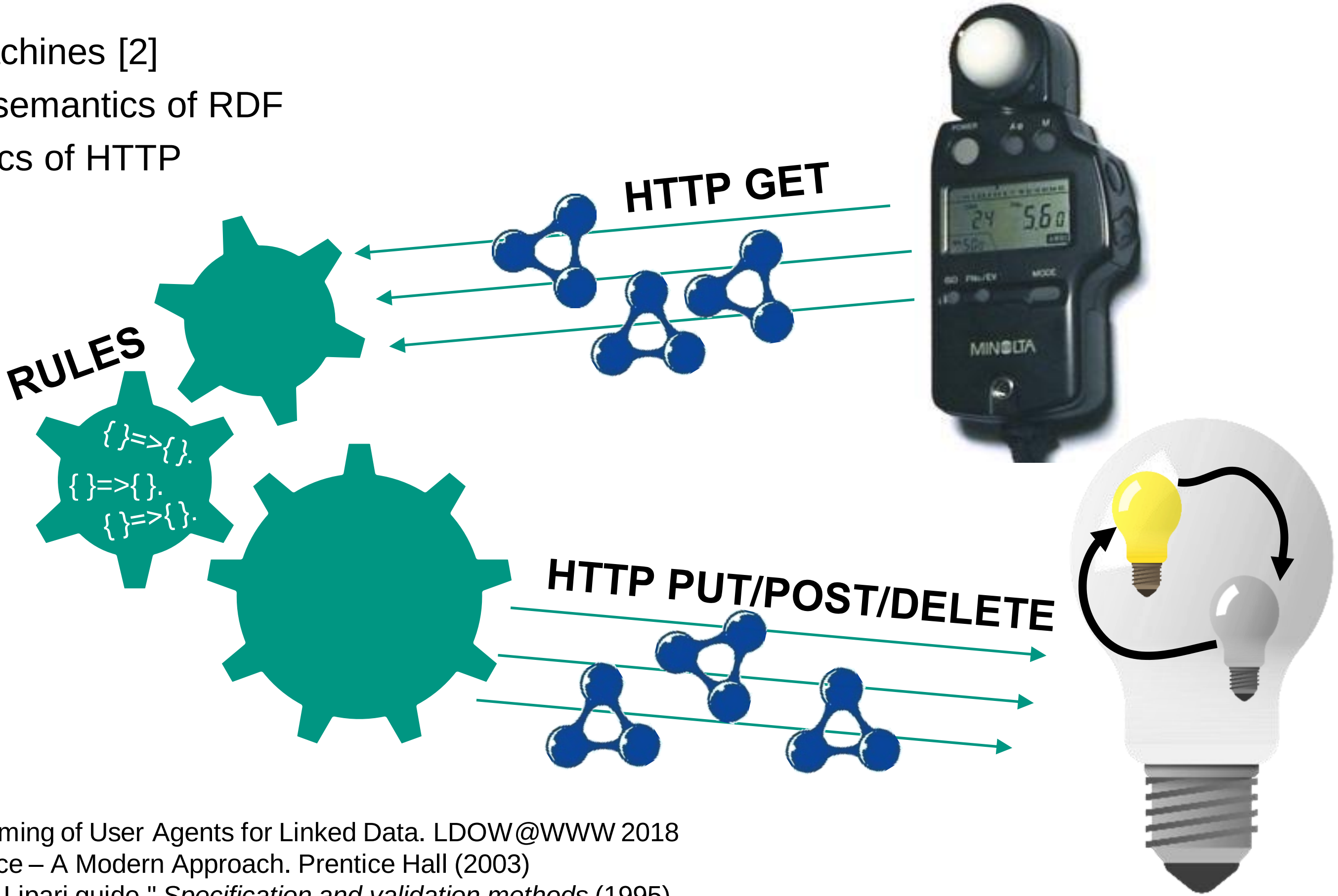
The article in the Scientific American is a lot about ontologies

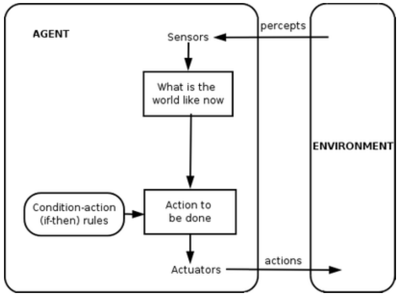
Programming User Agents: ASM4LD [0]

- Aim: Execution of agent specifications on Read-Write Linked Data
- Inspired by Simple Reflex Agents [1]
- Based on:
  - Abstract State Machines [2]
  - Model-theoretics semantics of RDF
  - Message semantics of HTTP

- In a nutshell:

```
while(true):
    sense()
    think()
    act()
```





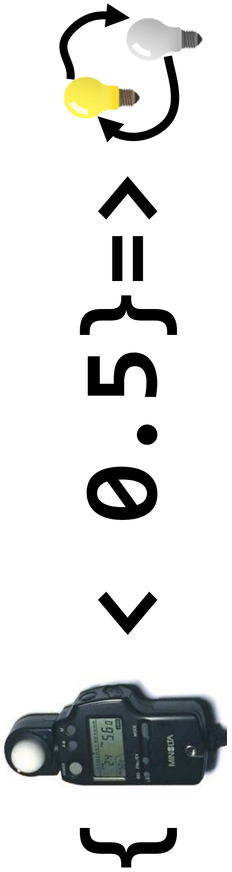
[0] Käfer & Harth: Rule-based Programming of User Agents for Linked Data. LDOW@WWW 2018
[1] Russell & Norvig: Artificial Intelligence – A Modern Approach. Prentice Hall (2003)
[2] Gurevich:. "Evolving algebras 1993: Lipari guide." *Specification and validation methods* (1995)

Algorithm to combine materialisation, link following, and programming

```
Require: assertions ▷ Graph
Require: rules ▷ Derivation and request rules
var data, oldData: set<triple>
var fixpointReached: Boolean
var unsafeRequests: set<request>
while true do ▷ Loop of the ASM steps
    unsafeRequests.clear()
    data.clear()
    data.add(assertions)
    repeat ▷ Loop for determining the fixpoint and the update set
        fixpointReached <- true
        for rule : rules do
            if rule.matches(data) then
                oldData = data.copy()
                if rule.type==derivation then
                    data.add(rule.match(data).data)
                else ▷ So the rule must be an interaction rule
                    if rule.match(data).request.type==GET then
                        data.add(rule.match(data).request.execute())
                    else
                        unsafeRequests.add(rule.match(data).request)
                    end if
                end if
                if ! data.copy().remove(oldData).isEmpty() then
                    fixpointReached <- false
                end if
            end if
        end for
    until fixpointReached
    for request : unsafeRequests do ▷ Enacting the update set
        request.execute()
    end for
end while
```

Turn the Light On in Linked Data-Fu

- Loop



```
{ [] a http:Request ;
     http:hasMethod httpM:GET ;
     http:requestURI </ambient/light> . }
{ [] a http:Request ;
     http:hasMethod httpM:GET ;
     http:requestURI </relay/1> . }

{ </ambient/light> rdf:value ?val .
  ?val math:lessThan 0.5 .
  </relay/1#r> :isOn false . }
=>
{ [] a http:Request ;
     http:hasMethod httpM:PUT ;
     http:requestURI </relay/1> ;
     http:body
       { </relay/1#r> :isOn true . } . } .
```

SENSE:
Retrieve the
world state

THINK:
Conditionally…

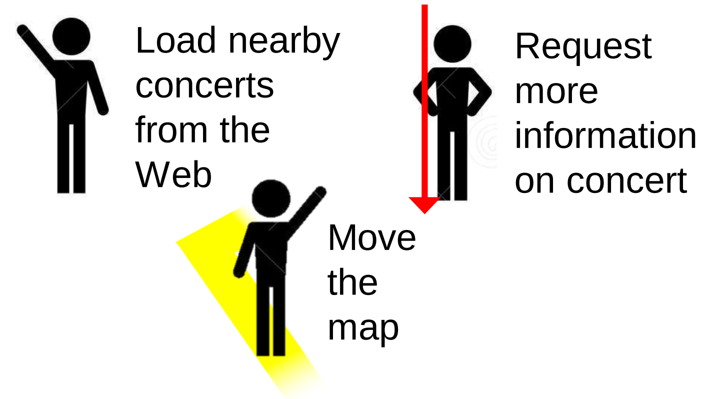
ACT:
…manipulate
the world state

Higher-level Ways of Programming Agents

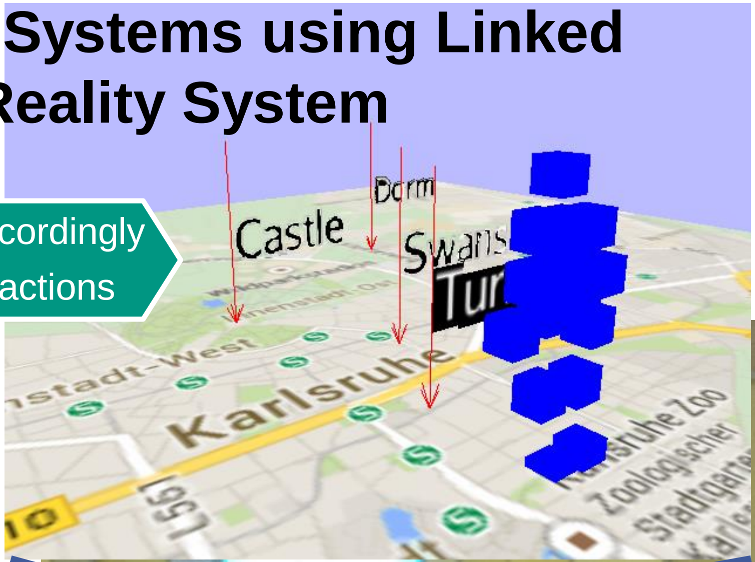
- We can use ASM4LD to give operational semantics to ontologies
- WiLD – Workflows in Linked Data
  - A flow-based workflow language
  - Käfer and Harth: „Specifying, Monitoring, and Executing Workflows in Linked Data“. Proc. ISWC 2018.
- GSM4LD
  - An artifact-centric workflow language
  - Jochum, Nürnberg, Aßfalg, Käfer: „Data-Driven Workflows for Specifying and Executing Agents in an Environment of Reasoning and RESTful Systems”. Proc. WS AI4BPM @ BPM 2019.

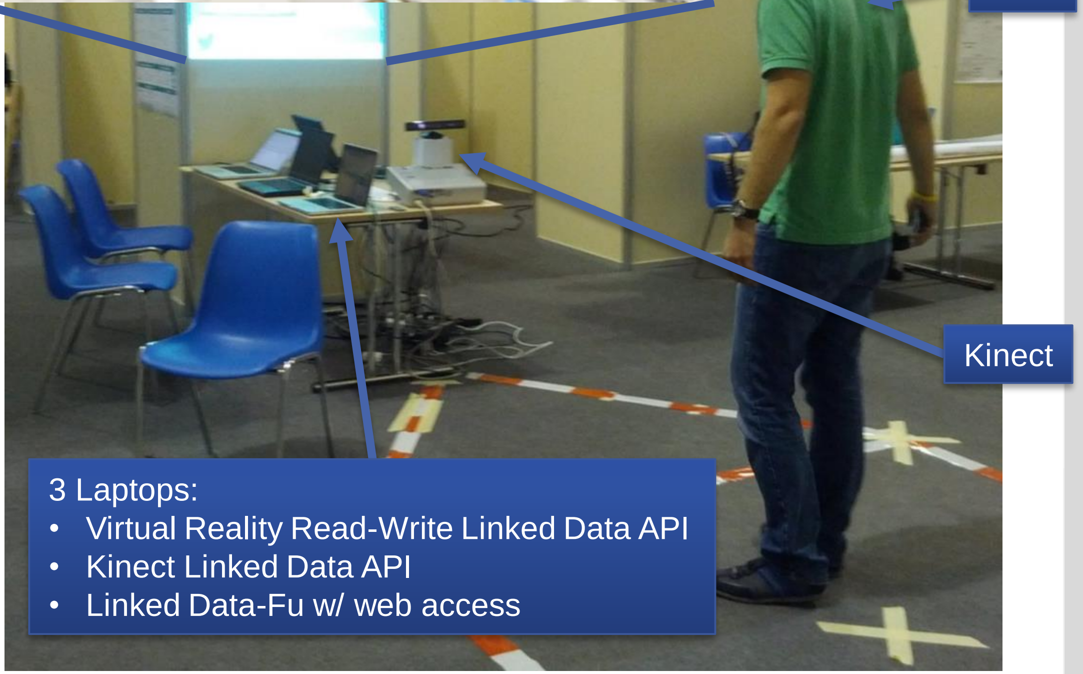
Integration of Distributed Systems using Linked Data: Example: a Virtual Reality System

Kinect tracks user → Avatar moves accordingly / Gestures trigger actions



Load nearby concerts from the Web
Request more information on concert
Move the map





3 Laptops:

- Virtual Reality Read-Write Linked Data API
- Kinect Linked Data API
- Linked Data-Fu w/ web access

- We encoded in Linked Data-Fu rules:
  - Movement of the avatar according to Kinect data
  - Detection of user gestures
  - Movement of the map according to gestures
  - Loading of concert data from the web
  - Data integration between VR RWLD API, concert LD API, Kinect LD API
- Execution at Kinect sensor refresh rate (30Hz)

Keppmann, Käfer, Stadtmüller, Schubotz, Harth: "High Performance Linked Data Processing for Virtual Reality Environments". P&D ISWC 2014.

THANKS FOR YOUR ATTENTION!