DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF OXFORD

# Semantic Technology Tutorial

Part 3: Languages

# SemTech Languages

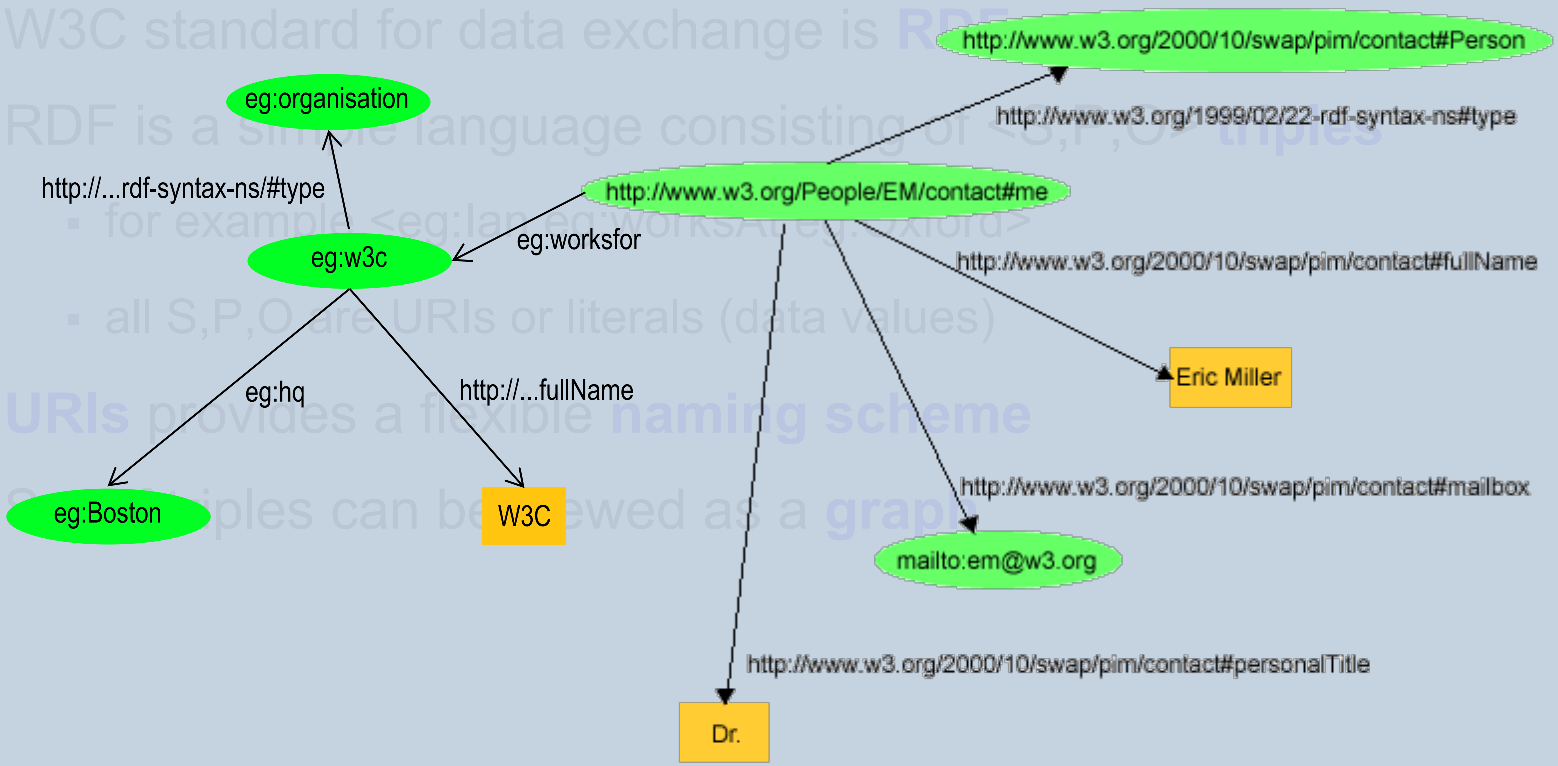
1 **Standardised language for data**

- W3C standard for data exchange is **RDF**
- RDF is a simple language consisting of <S P O> **triples**
  - for example <eg:Ian eg:worksAt eg:Oxford>
  - all S,P,O are URIs or literals (data values)
- **URIs** provides a flexible **naming scheme**
- Set of triples can be viewed as a **graph**

# SemTech Languages

## 1 Standardised language for **data**

- W3C standard for data exchange is RDF
- RDF is a simple language consisting of <S,P,O> triples
  - for example <eg:ian eg:worksAt eg:oxford>
  - all S,P,O are URIs or literals (data values)
- URIs provides a flexible naming scheme
- Set of triples can be viewed as a graph



# SemTech Languages

RDF

## 1 Standardised language for **data**

- W3C s…
- RDF i… triples
  - fo…
  - all…
- **URIs** …
- Set of…

| Triple | | |
|---|---|---|
| **S** | **P** | **O** |
| em1234 | rdf:type | Person |
| em1234 | name | “Eric Miller” |
| em1234 | title | “Dr” |
| em1234 | mailbox | mailto:em@w3.org |
| em1234 | worksfor | w3c |
| w3c | rdf:type | organisation |
| w3c | hq | Boston |
| w3c | name | “W3C” |
| … | … | … |

# SemTech Languages

RDF

## 1 Standardised language for **data**

- W3C standard for data exchange is RDF

| PERSON | | | | |
|---|---|---|---|---|
| **ID** | **NAME** | **TITLE** | **MAILBOX** | **WORKSFOR** |
| em1234 | “Eric Miller” | “Dr” | mailto:em@w3.org | w3c |
| ... | ... | ... | ... | ... |

| ORGANISATION | | |
|---|---|---|
| **ID** | **NAME** | **HQ** |
| w3c | “W3C” | Boston |
| ... | ... | ... |

...

# SemTech Languages

## 2 Standardised language for vocabularies/schemas

- W3C standard for vocabulary/schema exchange is **OWL**
- OWL provides for rich conceptual schemas, aka **ONTOLOGIES**

$$
\begin{aligned}
\mathsf{Heart} &\sqsubseteq \mathsf{MuscularOrgan} \sqcap \\
&\quad \exists \mathsf{isPartOf}.\mathsf{CirculatorySystem} \\
\mathsf{HeartDisease} &\equiv \mathsf{Disease} \sqcap \\
&\quad \exists \mathsf{affects}.\mathsf{Heart} \\
\mathsf{VascularDisease} &\equiv \mathsf{Disease} \sqcap \\
&\quad \exists \mathsf{affects}.(\exists \mathsf{isPartOf}.\mathsf{CirculatorySystem})
\end{aligned}
$$

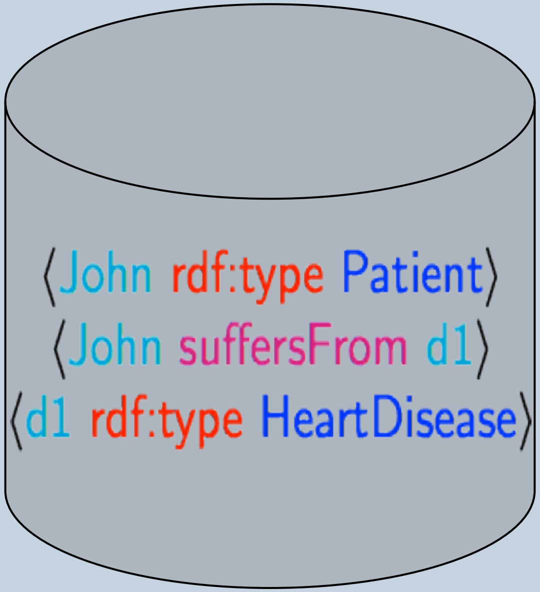
# SemTech Languages

3 **Standardised language for queries**

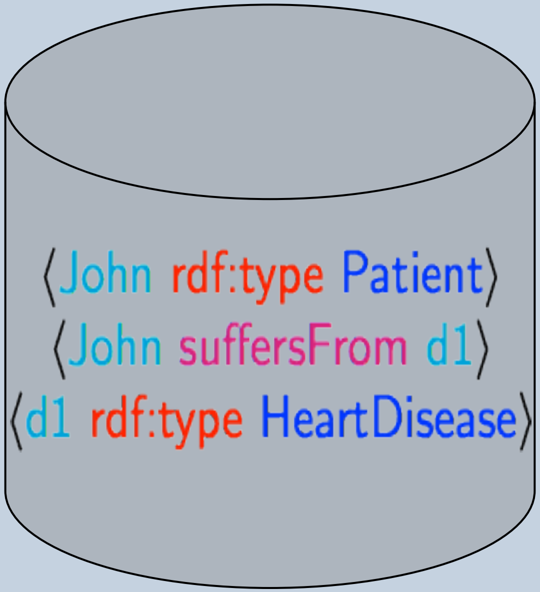
- W3C standard for queries is **SPARQL**
- SPARQL provides a rich query language comparable to **SQL**

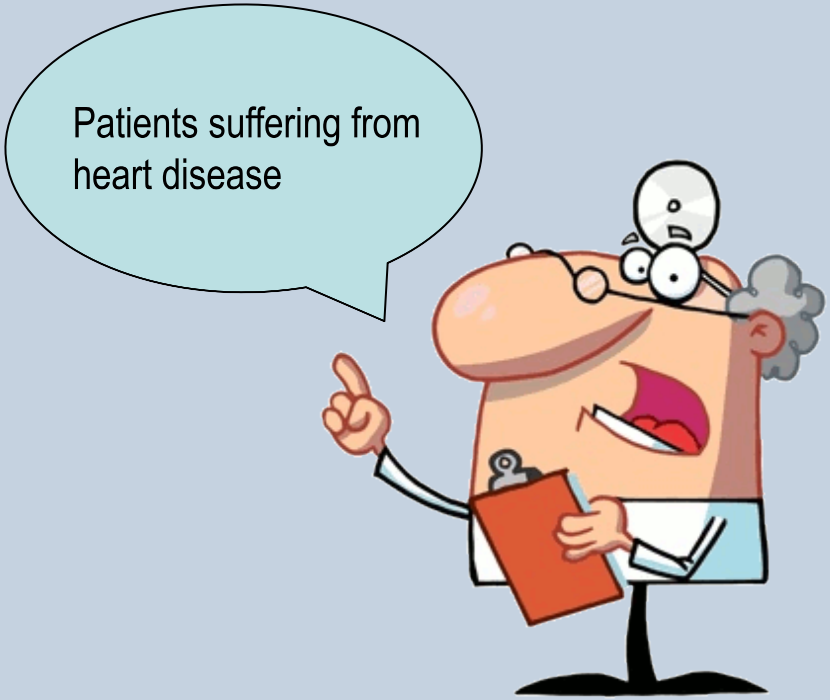
```
SELECT ?x
WHERE
  { ?x rdf:type Patient .
    ?x suffersFrom ?y .
    ?y rdf:type VascularDisease }
```

# How Does it Work?

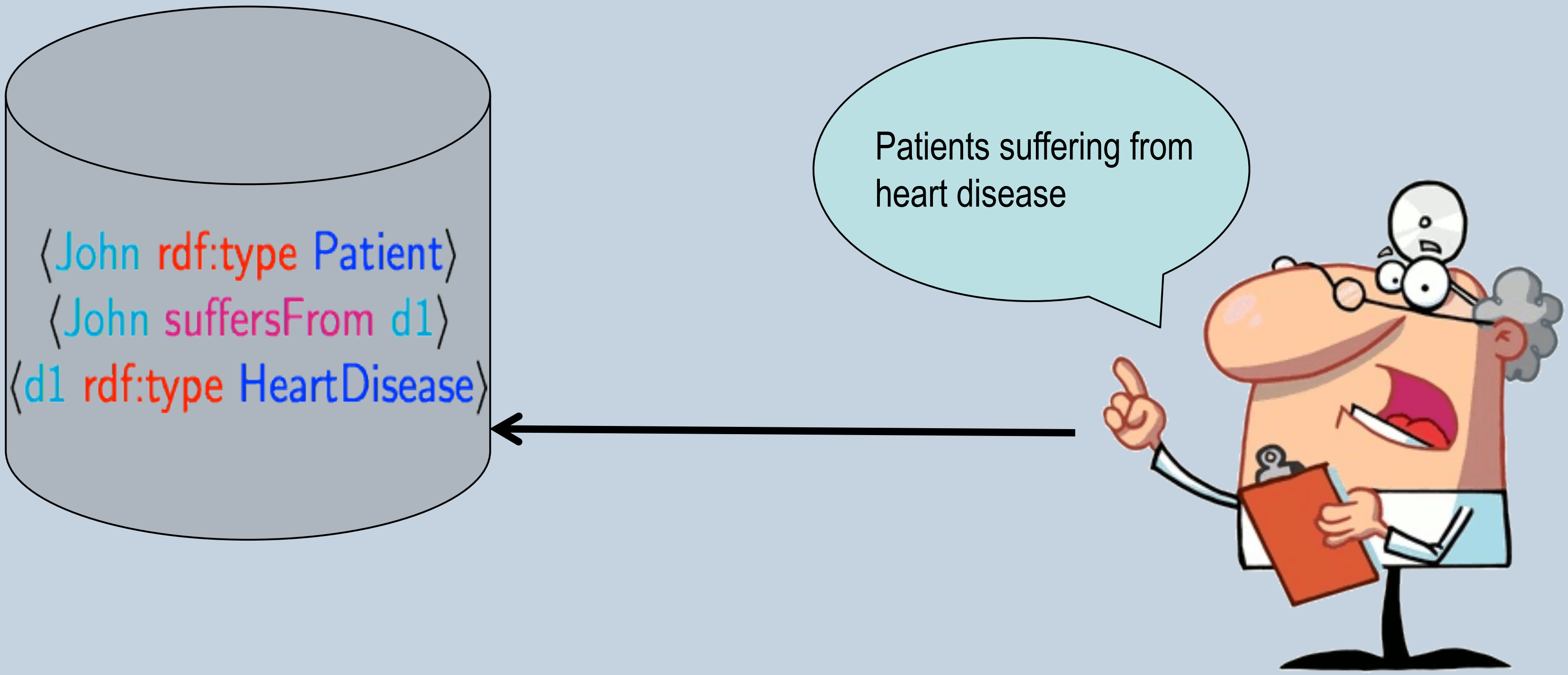


# How Does it Work?

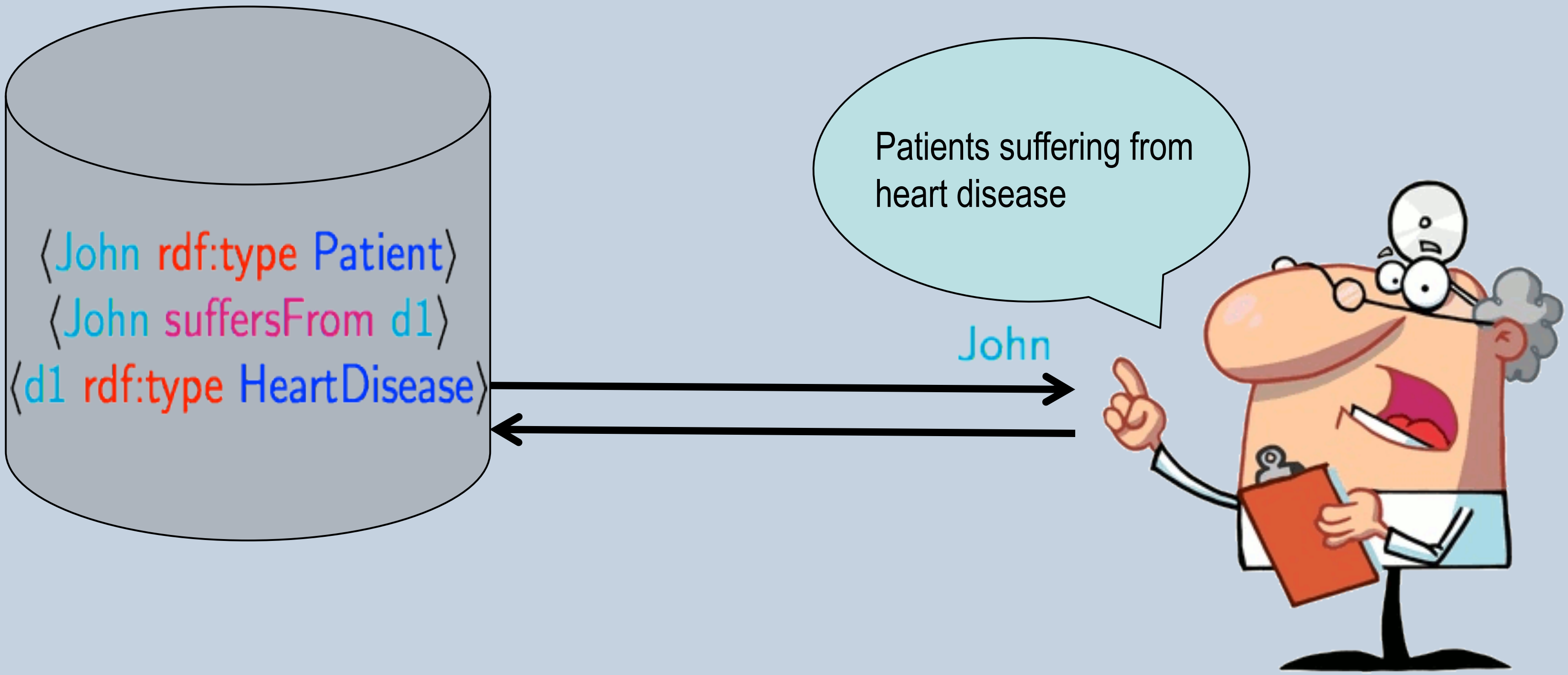


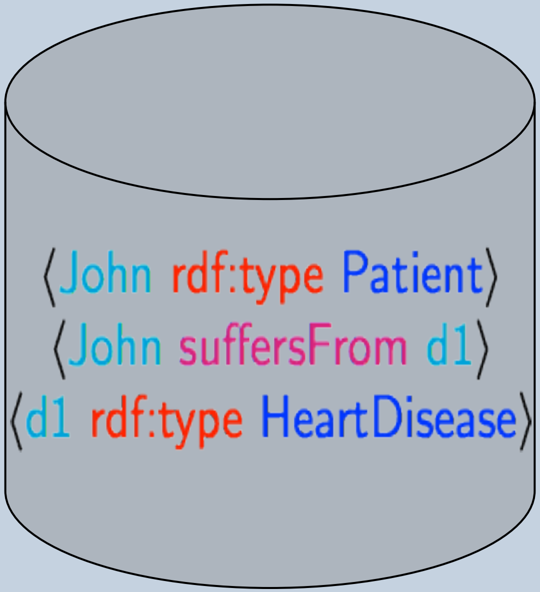
# How Does it Work?

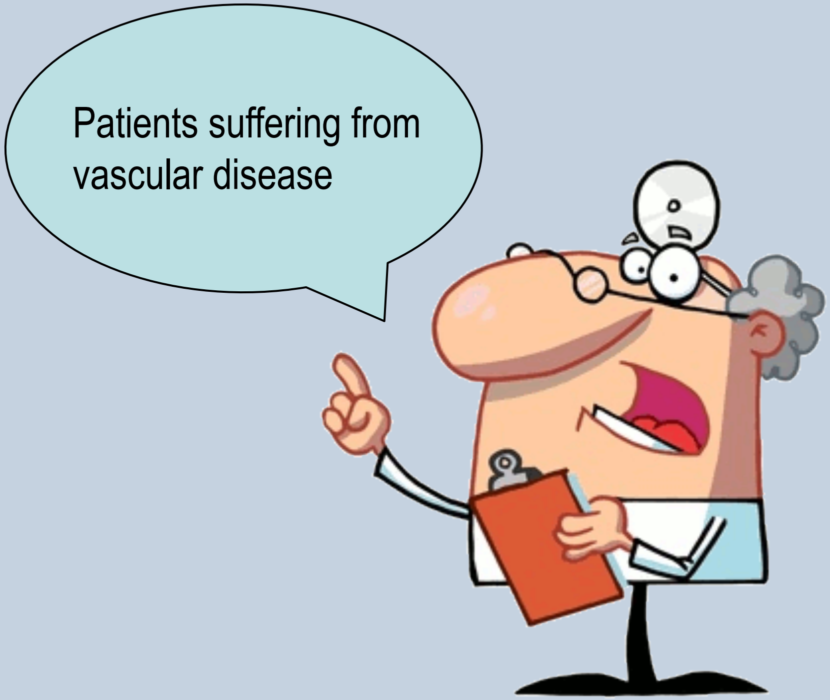


# How Does it Work?

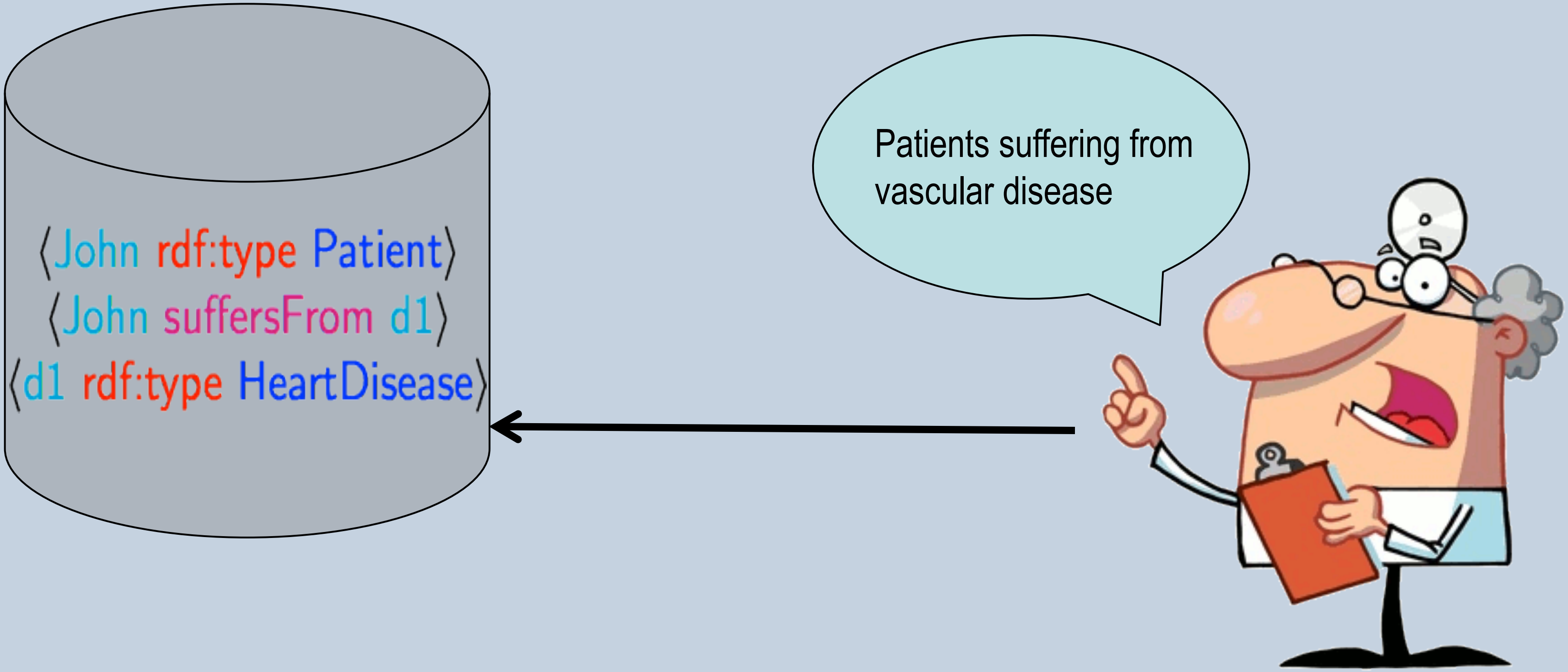


# How Does it Work?



# How Does it Work?

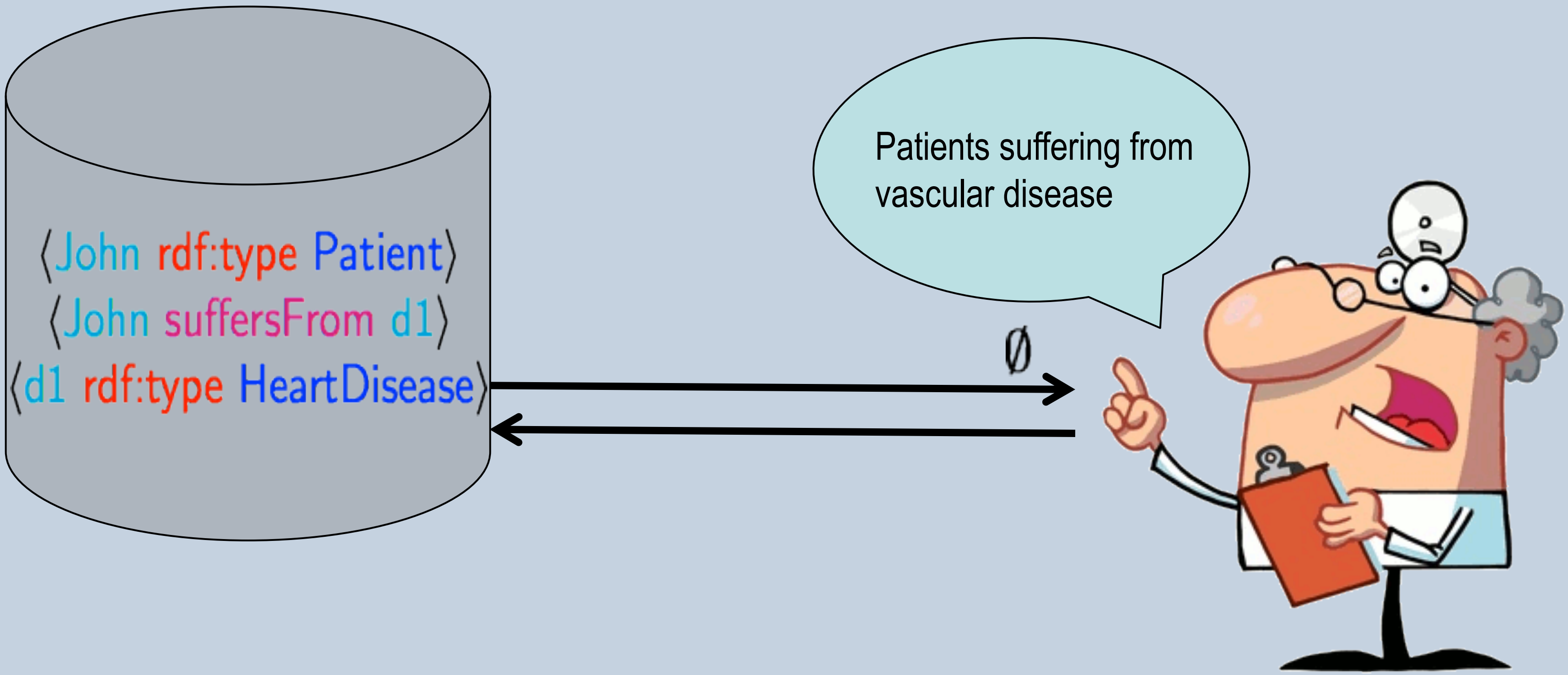



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF OXFORD

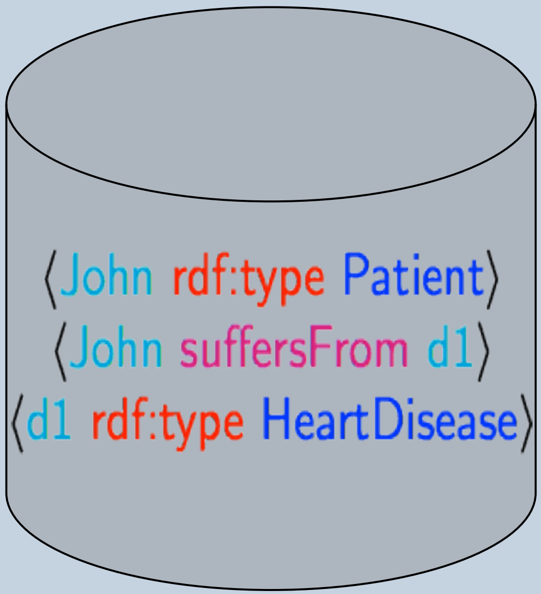
DBOnto

SIRIUS

# How Does it Work?



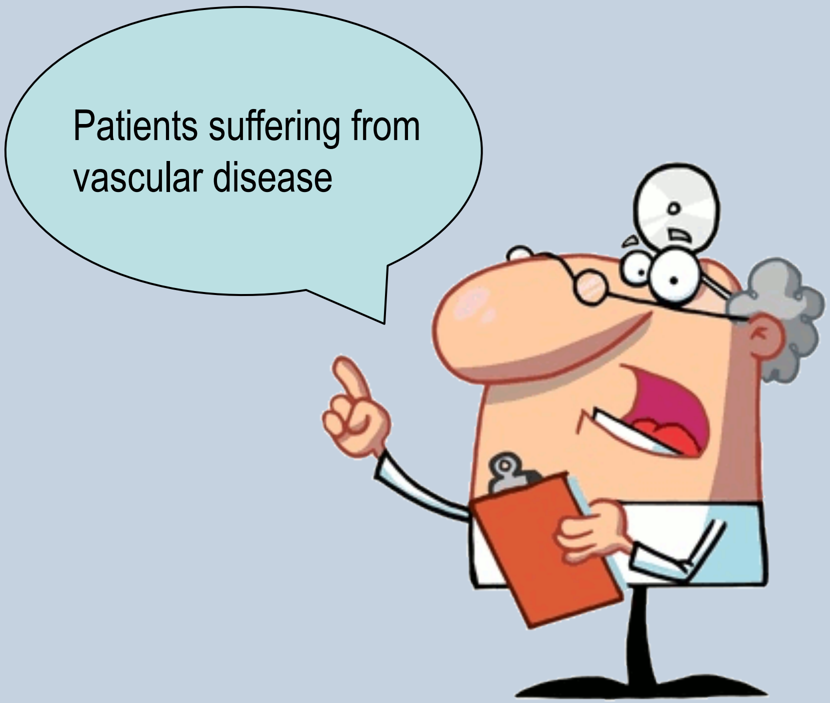
# How Does it Work?



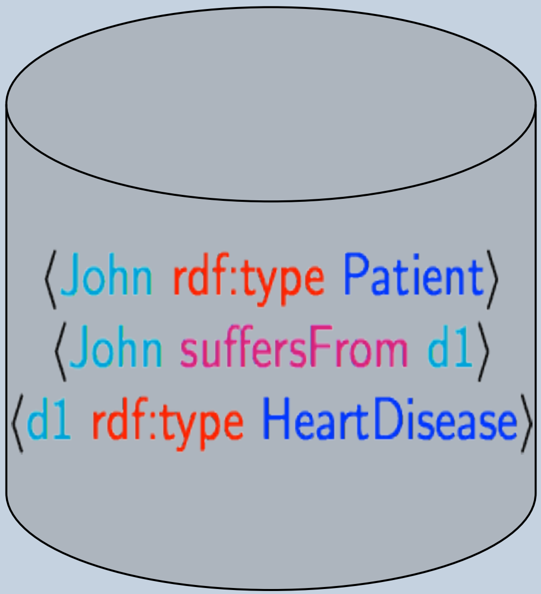

$$\mathsf{Heart} \sqsubseteq \mathsf{MuscularOrgan} \sqcap \exists \mathsf{isPartOf}.\mathsf{CirculatorySystem}$$

$$\mathsf{HeartDisease} \equiv \mathsf{Disease} \sqcap \exists \mathsf{affects}.\mathsf{Heart}$$

$$\mathsf{VascularDisease} \equiv \mathsf{Disease} \sqcap \exists \mathsf{affects}.(\exists \mathsf{isPartOf}.\mathsf{CirculatorySystem})$$



# How Does it Work?



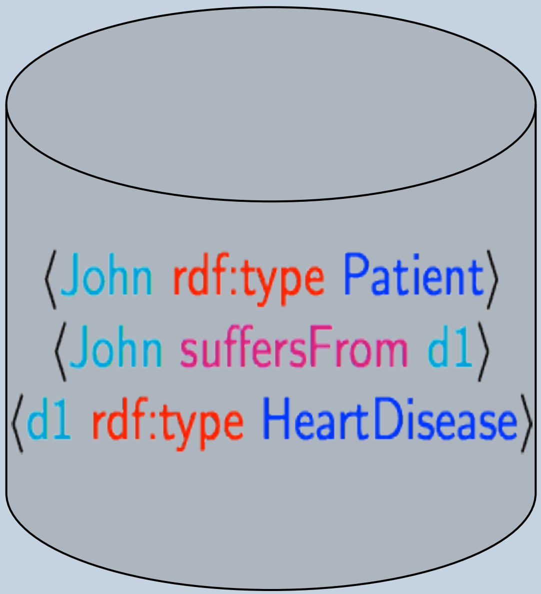

$$\begin{aligned}
\mathsf{Heart} &\sqsubseteq \mathsf{MuscularOrgan} \sqcap \\
&\quad \exists \mathsf{isPartOf}.\mathsf{CirculatorySystem} \\
\mathsf{HeartDisease} &\equiv \mathsf{Disease} \sqcap \\
&\quad \exists \mathsf{affects}.\mathsf{Heart} \\
\mathsf{VascularDisease} &\equiv \mathsf{Disease} \sqcap \\
&\quad \exists \mathsf{affects}.(\exists \mathsf{isPartOf}.\mathsf{CirculatorySystem})
\end{aligned}$$


UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE

DBOnto

SIRIUS

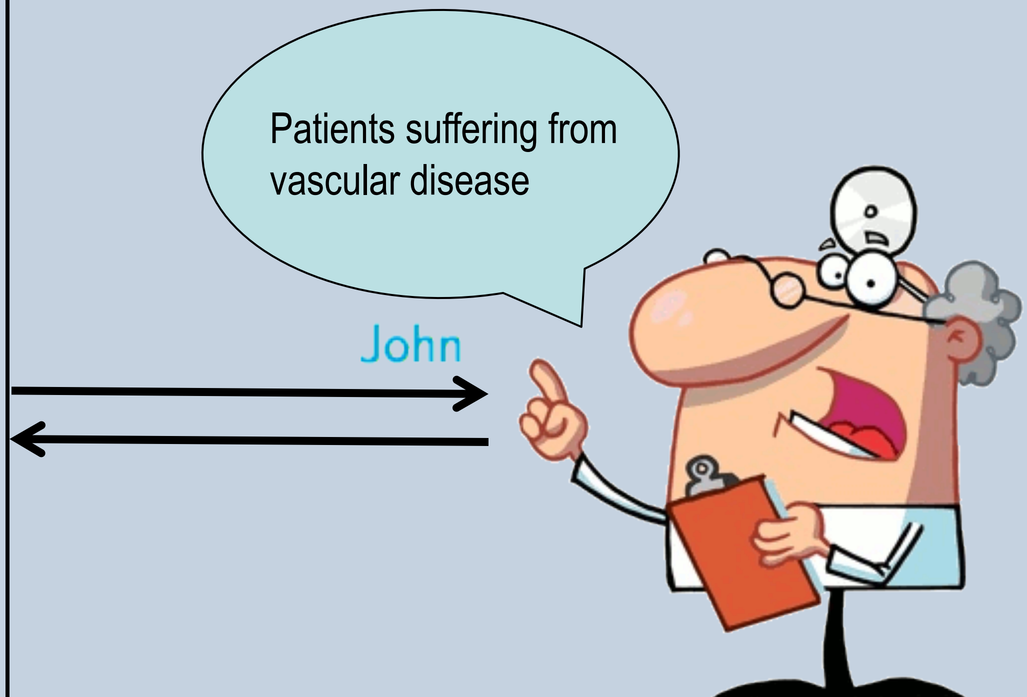
# How Does it Work?



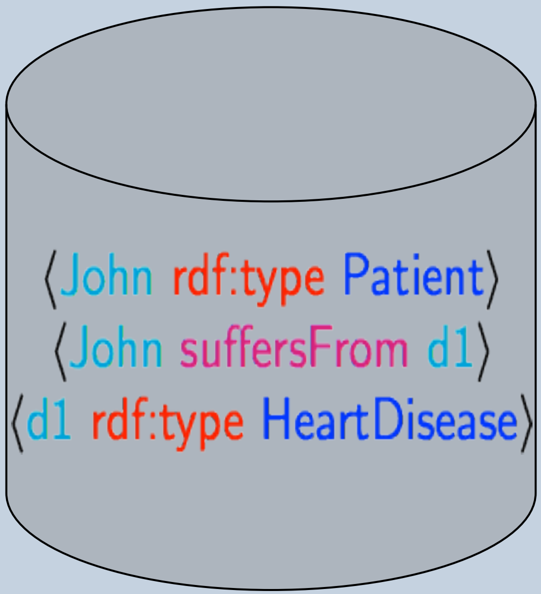

$$\mathsf{Heart} \sqsubseteq \mathsf{MuscularOrgan} \sqcap \exists \mathsf{isPartOf}.\mathsf{CirculatorySystem}$$

$$\mathsf{HeartDisease} \equiv \mathsf{Disease} \sqcap \exists \mathsf{affects}.\mathsf{Heart}$$

$$\mathsf{VascularDisease} \equiv \mathsf{Disease} \sqcap \exists \mathsf{affects}.(\exists \mathsf{isPartOf}.\mathsf{CirculatorySystem})$$



# How Does it Work?



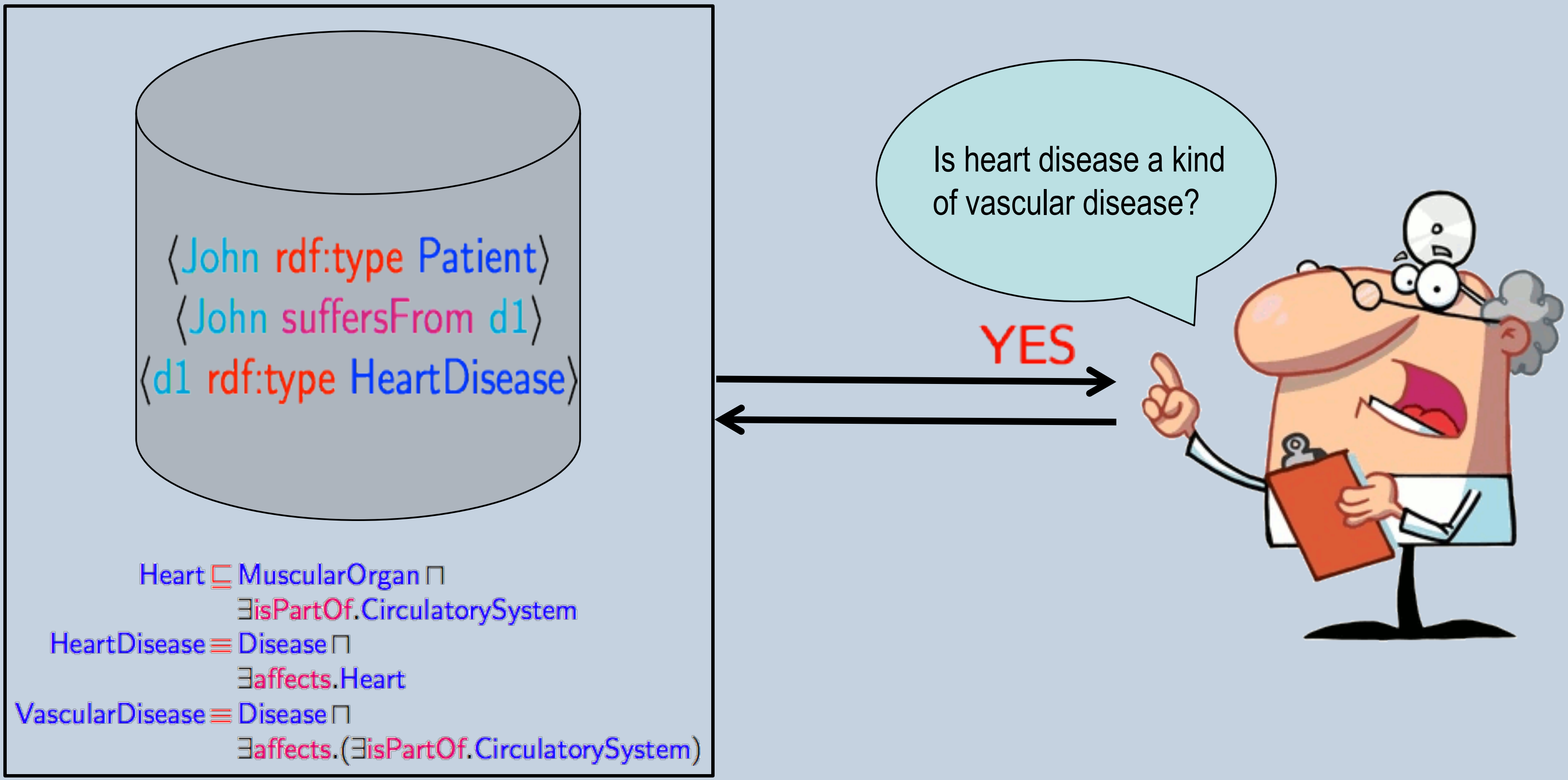

$$\mathsf{Heart} \sqsubseteq \mathsf{MuscularOrgan} \sqcap \exists \mathsf{isPartOf}.\mathsf{CirculatorySystem}$$
$$\mathsf{HeartDisease} \equiv \mathsf{Disease} \sqcap \exists \mathsf{affects}.\mathsf{Heart}$$
$$\mathsf{VascularDisease} \equiv \mathsf{Disease} \sqcap \exists \mathsf{affects}.(\exists \mathsf{isPartOf}.\mathsf{CirculatorySystem})$$


DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF OXFORD

DBOnto

SIRIUS

# How Does it Work?




UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE

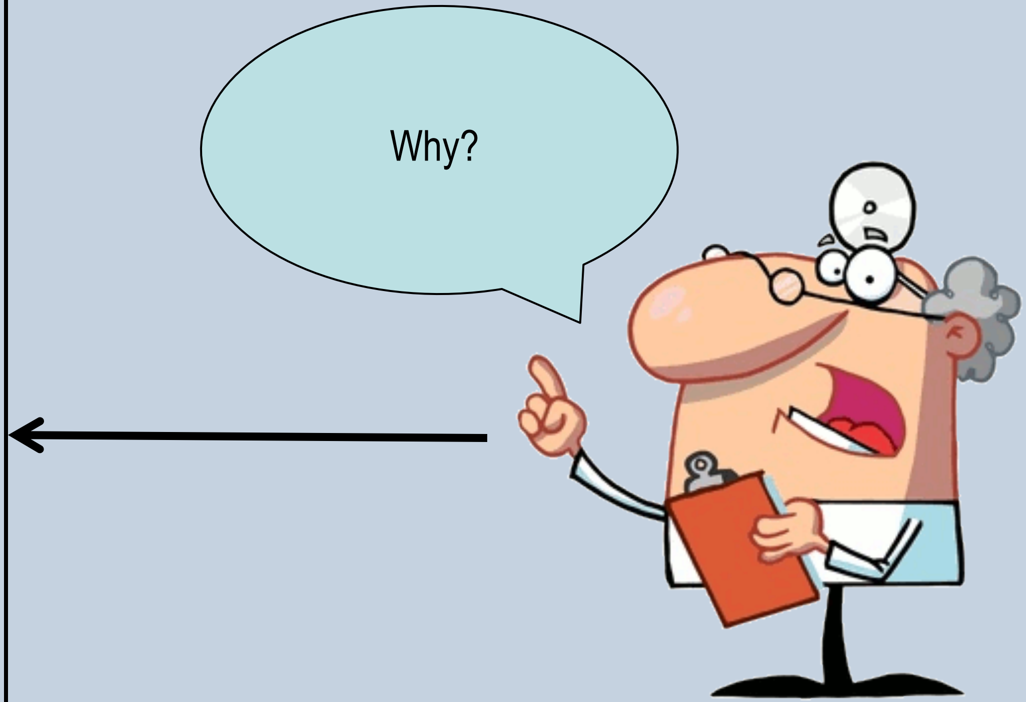
DBOnto

SIRIUS

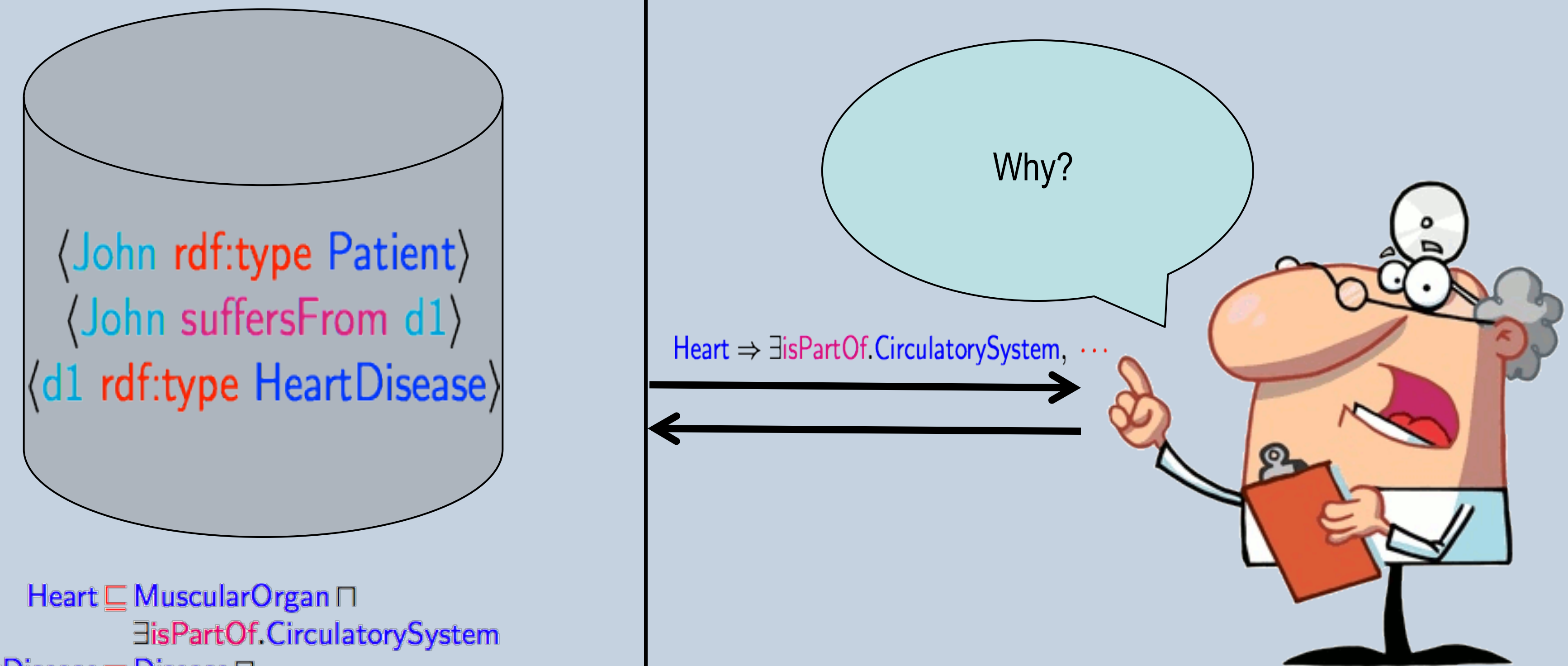
# How Does it Work?

⟨John rdf:type Patient⟩
⟨John suffersFrom d1⟩
⟨d1 rdf:type HeartDisease⟩

$$
\begin{aligned}
\mathsf{Heart} &\sqsubseteq \mathsf{MuscularOrgan} \sqcap \\
&\quad \exists \mathsf{isPartOf}.\mathsf{CirculatorySystem} \\
\mathsf{HeartDisease} &\equiv \mathsf{Disease} \sqcap \\
&\quad \exists \mathsf{affects}.\mathsf{Heart} \\
\mathsf{VascularDisease} &\equiv \mathsf{Disease} \sqcap \\
&\quad \exists \mathsf{affects}.(\exists \mathsf{isPartOf}.\mathsf{CirculatorySystem})
\end{aligned}
$$



# How Does it Work?




$$
\begin{aligned}
\text{Heart} &\sqsubseteq \text{MuscularOrgan} \sqcap \\
&\quad \exists \text{isPartOf}.\text{CirculatorySystem} \\
\text{HeartDisease} &\equiv \text{Disease} \sqcap \\
&\quad \exists \text{affects}.\text{Heart} \\
\text{VascularDisease} &\equiv \text{Disease} \sqcap \\
&\quad \exists \text{affects}.(\exists \text{isPartOf}.\text{CirculatorySystem})
\end{aligned}
$$

# W3C Standards

- An ontology language defines constructs available to modellers
  - E.g., kinds of statements about concepts (conjunction, negation, . . .)
  - Formal semantics specifies mathematically the constructs' meaning
  - Semantics determines the inferences one can draw
- Standard languages facilitate interoperability
- Semantic Web language stack:

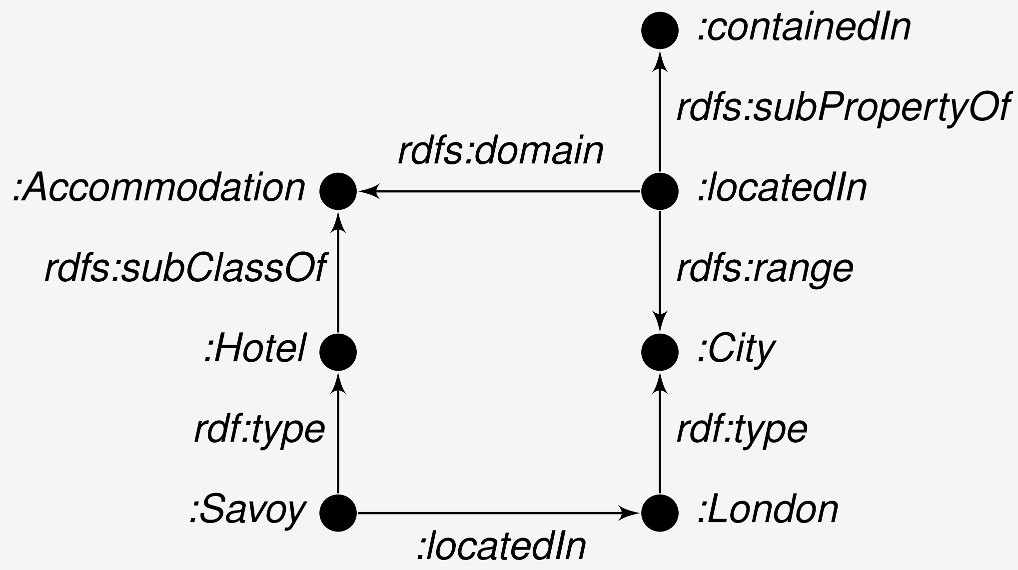
| | |
|---|---|
| Resource Description Framework (RDF) | basic semistructured data model |
| RDF Schema (RDFS) | a simple ontology language over RDF |
| Web Ontology Language (OWL) 2 | extends RDFS to an expressive language |
| ■ OWL 2 Full | undecidable |
| ■ OWL 2 DL | decidable, based on description logics |
| ■ OWL 2 EL | } profiles: trade expressivity for efficiency |
| ■ OWL 2 QL | |
| ■ OWL 2 RL | |
| Semantic Web Rule Language (SWRL) | unofficial rule standard |
| Rule Interchange Format (RIF) | (mainly production) rule standard |

# Resource Description Framework (RDF): Basic Concepts

- Node — an object one can make statements about (often called resource)
  - IRI — well-known identifier for an object
    - E.g., ⟨*http://skyscanner.net/Savoy*⟩, often abbreviated as *sky:Savoy*
  - Blank node — an object with an unknown identity (aka labelled null)
    - E.g., *_:x*
  - Literal — concrete value such as a string or an integer
    - E.g., *"abc"^^xsd:string*, *"1"^^xsd:integer*, *"+01"^^xsd:byte*
- Triple — the simplest statement about objects
  - ⟨$s$, $p$, $o$⟩ with $s$, $p$, and $o$ nodes: object $o$ is the value of property $p$ on subject $s$
  - E.g., ⟨*:Savoy*, *:locatedIn*, *:London*⟩, ⟨*:Savoy*, *rdf:type*, *:Hotel*⟩
- RDF graph — a finite set of RDF triples
  - Can be understood as a three-column relation over nodes
- RDF dataset — a finite set of RDF graphs, each associated with a node
- Built-in vocabulary: *rdf:type*, *rdf:Property*, . . .
  - *rdf:type* states that a node is an instance of a class
- More details at `http://www.w3.org/TR/rdf11-concepts/`

# Example RDF Graph

- RDF graphs can be represented graphically
  - Properties are nodes, so one can make statements about them



# RDF/XML Syntax

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns="http://skyscanner.net/">

<rdf:Description rdf:about="http://skyscanner.net/Hotel">
  <rdfs:subClassOf rdf:resource="http://skyscanner.net/Accommodation"/>
</rdf:Description>

<rdf:Description rdf:about="http://skyscanner.net/Savoy">
  <rdf:type rdf:resource="http://skyscanner.net/Accommodation"/>
  <locatedIn rdf:resource="http://skyscanner.net/London"/>
</rdf:Description>

<rdf:Description rdf:about="http://skyscanner.net/London">
  <rdf:type rdf:resource="http://skyscanner.net/City"/>
</rdf:Description>

<rdf:Description rdf:about="http://skyscanner.net/locatedIn">
  <rdfs:domain rdf:resource="http://skyscanner.net/Accommodation"/>
  <rdfs:range rdf:resource="http://skyscanner.net/City"/>
  <rdfs:subPropertyOf rdf:resource="http://skyscanner.net/containedIn"/>
</rdf:Description>

</rdf:RDF>
```

# TURTLE SYNTAX

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

:Hotel rdfs:subClassOf :Accommodation .

:Savoy rdf:type   :Accommodation ;
       :locatedIn :London"       .

:London rdf:type :City .

:locatedIn rdfs:domain        :Accommodation ;
           rdfs:range         :City          ;
           rdfs:subPropertyOf :containedIn   .
```

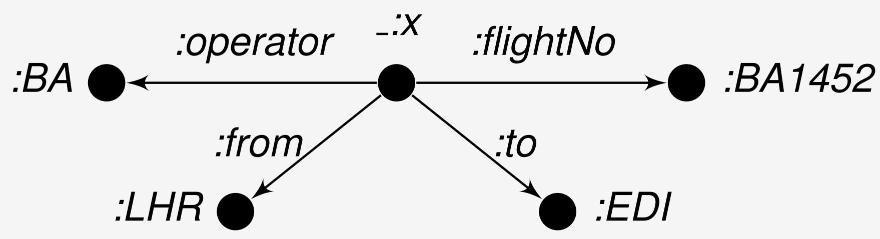
- Much more readable and compact!
- http://www.w3.org/TR/turtle/

# Embedding RDF into Relational Model

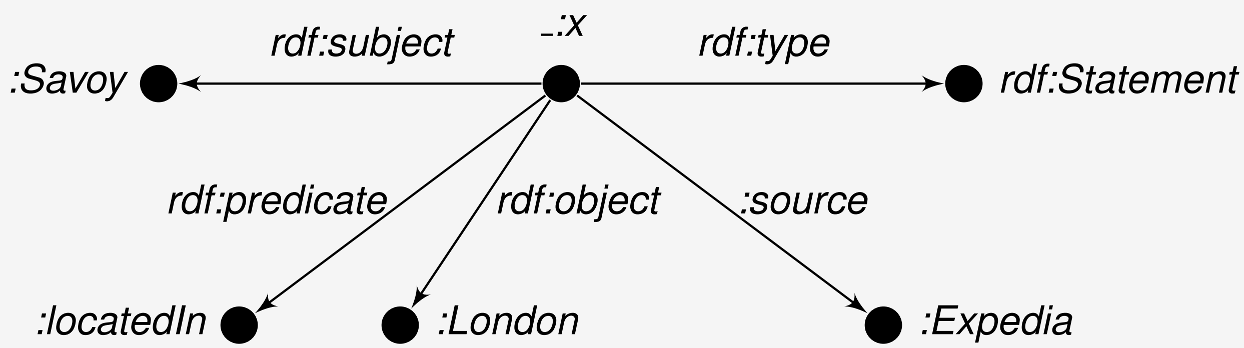
- RDF data can be stored in a relational database in (at least) two ways
- Dictionary encoding commonly used to map nodes to integers
- Triple table approach
  - Store triples in a three-column table
  - Exhaustive indexing can be achieved using only six indexes
  - Often extended to quads $\rightarrow$ triples with additional graph membership node
  - Main benefit: flexibility to support any kind of query
  - Main problem: queries involve many self-joins on the triple table
- Vertical partitioning approach
  - Use binary relations for properties, unary relations for classes
  - Store $\langle s, p, o \rangle$ with $p \neq \mathit{rdf{:}type}$ as tuple $\langle s, o \rangle$ in relation $p$
  - Store $\langle s, \mathit{rdf{:}type}, o \rangle$ as tuple $\langle s \rangle$ in relation $o$
  - Use exhaustive indexing
  - Main benefit: avoids self-joins $\rightarrow$ easier for DBMSs
  - Main problem: does not support queries with variables in predicate position

# Restriction to Binary Relations and Reification

- RDF supports only binary relations $\rightarrow$ often very restrictive in practice
  - E.g., ‘British Airways operates flight BA1452 from LHR to EDI’
- Reification represents a statement as an object




- Can be used to make statements about triples
  - E.g., ‘⟨*:Savoy*, *:locatedIn*, *:London*⟩ was obtained from Expedia’

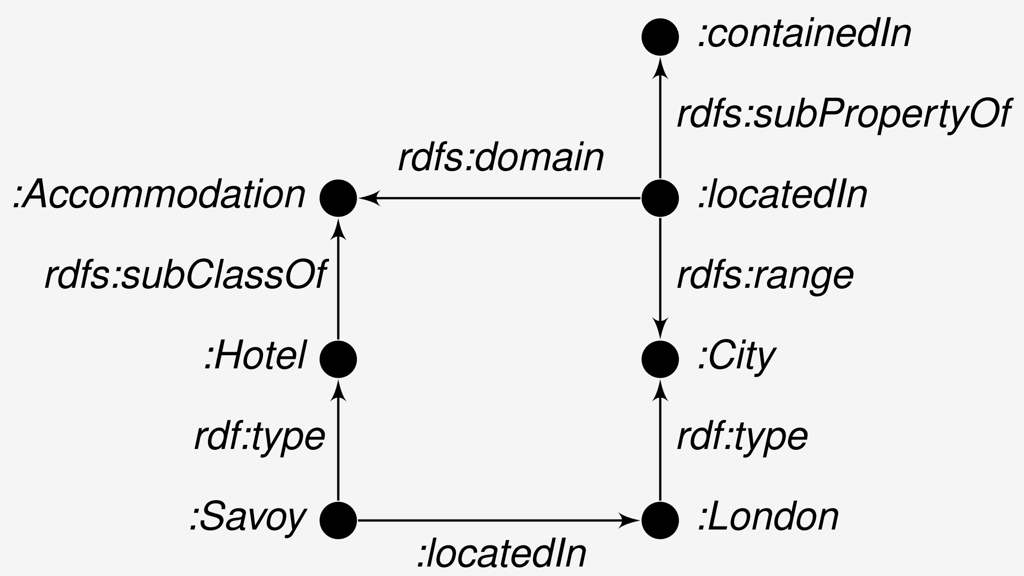


# LITERALS

- *"lexicalValue"^^datatypeIRI* — *datatypeIRI* identifies a datatype that specifies how to map *"lexicalValue"* to a concrete value
  - Many datatypes come from XML Schema 1.1
  - `http://www.w3.org/TR/xmlschema11-2/`
- E.g., *"abc"^^xsd:string*, *"1"^^xsd:integer*, *"+01"^^xsd:byte*
- Syntactic shortcuts:
  - *xsd:string* can be omitted: *"abc"^^xsd:string* → *"abc"*
  - *"abc"@en* supports localisation → equivalent to *"abc@en"^^rdf:PlainLiteral*
- Literal equality and equivalence are different concepts:
  - Equal if lexical values and datatypes are the same
  - Equivalent if mapped to the same value
  - E.g., *"1"^^xsd:integer* and *"+01"^^xsd:byte* are not equal, but are equivalent
- RDF systems often normalise literals on import
  - E.g., *"+01"^^xsd:byte* is stored as *"1"^^xsd:integer*

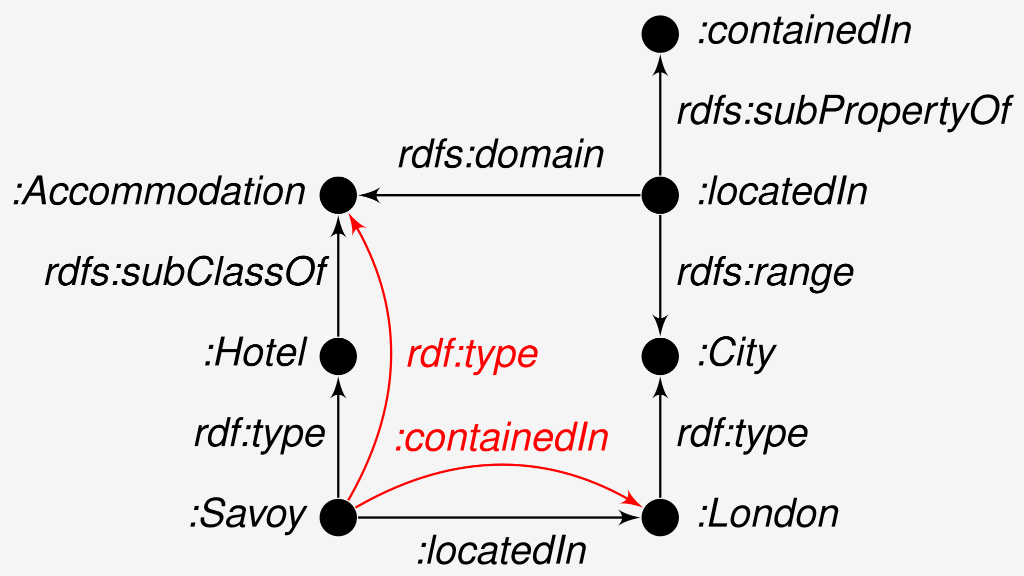
# RDF Schema (RDFS)

- RDFS: a simple ontology language for RDF data
- Introduces special vocabulary
  - E.g., *rdfs:subClassOf*, *rdfs:subPropertyOf*, *rdfs:domain*, *rdfs:range*, ...
  - Schema not separate from data ⇒ schema is data
- RDF(S) semantics specifies consequences of the special vocabulary
  - `http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/`
  - Can be captured using entailment rules
    - E.g., 'If ?X is an instance of ?Y, and ?Y is a subclass of ?Z, then ?X is an instance of ?Z'



# RDF Schema (RDFS)

- RDFS: a simple ontology language for RDF data
- Introduces special vocabulary
  - E.g., *rdfs:subClassOf*, *rdfs:subPropertyOf*, *rdfs:domain*, *rdfs:range*, . . .
  - Schema not separate from data ⇒ schema is data
- RDF(S) semantics specifies consequences of the special vocabulary
  - `http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/`
  - Can be captured using entailment rules
    - E.g., 'If ?X is an instance of ?Y, and ?Y is a subclass of ?Z, then ?X is an instance of ?Z'



# WHAT IS DATALOG?

- Datalog captures entailment rules in a formal way
- Related to Prolog, widely used in databases and Semantic Web
- Term — a node or a variable
  - E.g., ?X, sometimes also written as $\#X$
- (RDF) atom — a triple in which $s$, $p$, and $o$ are terms (not just nodes)
  - E.g., $\langle ?X, \textit{rdf:type}, \textit{:City}\rangle$, $\langle ?X, \textit{:locatedIn}, ?Y\rangle$
  - General atoms have form $R(t_1, \ldots, t_n)$ for $R$ an $n$-ary relation
    - In RDF, there is just one 'triple' relation so we omit it
  - Equivalent logical notation:
    - Classes $\rightarrow$ unary relations: $\langle ?X, \textit{rdf:type}, \textit{:City}\rangle \rightsquigarrow \textit{:City}(?X)$
    - Properties $\rightarrow$ binary relations: $\langle ?X, \textit{:locatedIn}, ?Y\rangle \rightsquigarrow \textit{:locatedIn}(?X, ?Y)$
    - Works if triples do not contain variables in class/property positions
- (Datalog) rule — implication of the form $H \leftarrow B_1 \wedge \ldots \wedge B_n$
  - Also written as $H :- B_1, \ldots, B_n$.
  - $H$ is the head atom
  - $B_1, \ldots, B_n$ are body atoms
  - Each rule must be safe: each variable in the rule must occur in some body atom
- (Datalog) program — a finite set of rules

# Capturing Entailment Rules of RDFS in Datalog

- Entailments about schema:

$$\langle ?X, \mathit{rdfs{:}subClassOf}, ?Z\rangle \leftarrow \langle ?X, \mathit{rdfs{:}subClassOf}, ?Y\rangle \wedge \langle ?Y, \mathit{rdfs{:}subClassOf}, ?Z\rangle$$
$$\langle ?X, \mathit{rdfs{:}subPropertyOf}, ?Z\rangle \leftarrow \langle ?X, \mathit{rdfs{:}subPropertyOf}, ?Y\rangle \wedge \langle ?Y, \mathit{rdfs{:}subPropertyOf}, ?Z\rangle$$
$$\langle ?X, \mathit{rdfs{:}domain}, ?Z\rangle \leftarrow \langle ?X, \mathit{rdfs{:}domain}, ?Y\rangle \wedge \langle ?Y, \mathit{rdfs{:}subPropertyOf}, ?Z\rangle$$
$$\langle ?X, \mathit{rdfs{:}range}, ?Z\rangle \leftarrow \langle ?X, \mathit{rdfs{:}range}, ?Y\rangle \wedge \langle ?Y, \mathit{rdfs{:}subPropertyOf}, ?Z\rangle$$

  - Rules in red are not mentioned in standards, but should be
  - This part of the standard is, IMHO, poorly designed

- Entailments about data:

$$\langle ?X, \mathit{rdf{:}type}, ?Z\rangle \leftarrow \langle ?X, \mathit{rdf{:}type}, ?Y\rangle \wedge \langle ?Y, \mathit{rdfs{:}subClassOf}, ?Z\rangle$$
$$\langle ?X, ?W, ?Z\rangle \leftarrow \langle ?X, ?Y, ?Z\rangle \wedge \langle ?Y, \mathit{rdfs{:}subPropertyOf}, ?W\rangle$$
$$\langle ?X, \mathit{rdf{:}type}, ?Z\rangle \leftarrow \langle ?X, ?W, ?Y\rangle \wedge \langle ?W, \mathit{rdfs{:}domain}, ?Z\rangle$$
$$\langle ?Y, \mathit{rdf{:}type}, ?Z\rangle \leftarrow \langle ?X, ?W, ?Y\rangle \wedge \langle ?W, \mathit{rdfs{:}range}, ?Z\rangle$$

- Rules are fixed $\Rightarrow$ do not depend on the ontology

# Alternative: Ontology-Specific Entailment Rules

- One can use rules created for each ontology separately:

$$\begin{aligned}
\langle ?X, \mathit{rdf{:}type}, \mathit{{:}Accommodation}\rangle &\leftarrow \langle ?X, \mathit{rdf{:}type}, \mathit{{:}Hotel}\rangle \\
\langle ?X, \mathit{rdf{:}type}, \mathit{{:}Accommodation}\rangle &\leftarrow \langle ?X, \mathit{{:}locatedIn}, ?Y\rangle \\
\langle ?Y, \mathit{rdf{:}type}, \mathit{{:}City}\rangle &\leftarrow \langle ?X, \mathit{{:}locatedIn}, ?Y\rangle \\
\langle ?X, \mathit{{:}containedIn}, ?Y\rangle &\leftarrow \langle ?X, \mathit{{:}locatedIn}, ?Y\rangle
\end{aligned}$$

- Often written using logical syntax:

$$\begin{aligned}
\mathit{{:}Accommodation}(?X) &\leftarrow \mathit{{:}Hotel}(?X) \\
\mathit{{:}Accommodation}(?X) &\leftarrow \mathit{{:}locatedIn}(?X, ?Y) \\
\mathit{{:}City}(?X) &\leftarrow \mathit{{:}locatedIn}(?X, ?Y) \\
\mathit{{:}containedIn}(?X, ?Y) &\leftarrow \mathit{{:}locatedIn}(?X, ?Y)
\end{aligned}$$

- More rules, but fewer body atoms
  - More efficient due to shorted rules
  - Can capture only data entailments

- See *B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logic. Proc. WWW 2003, pages 48–57*

# SEMANTIC WEB RULE LANGUAGE (SWRL)

- De facto standard for rules on the Web
- http://www.w3.org/Submission/SWRL/
- Several syntaxes, one of them encodes rules into RDF

```
<ruleml:imp>
  <ruleml:_body>
      <owlx:Class owlx:name="Hotel" />
      <ruleml:var>X</ruleml:var>
    </swrlx:classAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:classAtom>
      <owlx:Class owlx:name="Accommodation" />
      <ruleml:var>X</ruleml:var>
    </swrlx:classAtom>
  </ruleml:_head>
</ruleml:imp>
```

# Rule Interchange Format (RIF)

- A standard for rules on the Web
- `http://www.w3.org/standards/techs/rif#w3c_all`
- IMHO, mostly used in production rule systems, not the Semantic Web

```
Document(
  Prefix(sky <http://skyscanner.net/>)

  Group (
    Forall ?X (
      sky:Accommodation(?X) :- sky:Hotel(?X)
    )
    Forall ?X (
      sky:Accommodation(?X) :- sky:locatedIn(?X ?Y)
    )
    Forall ?Y (
      sky:City(?X) :- sky:locatedIn(?X ?Y)
    )
    Forall ?Y (
      sky:containedIn(?X ?Y) :- sky:locatedIn(?X ?Y)
    )
  )
)
```

# Recursion

- Rules can express recursive queries!
- Significantly more expressive than relational databases
  - WITH clause in SQL-1999 supports limited recursion
  - Not widely (efficiently) implemented

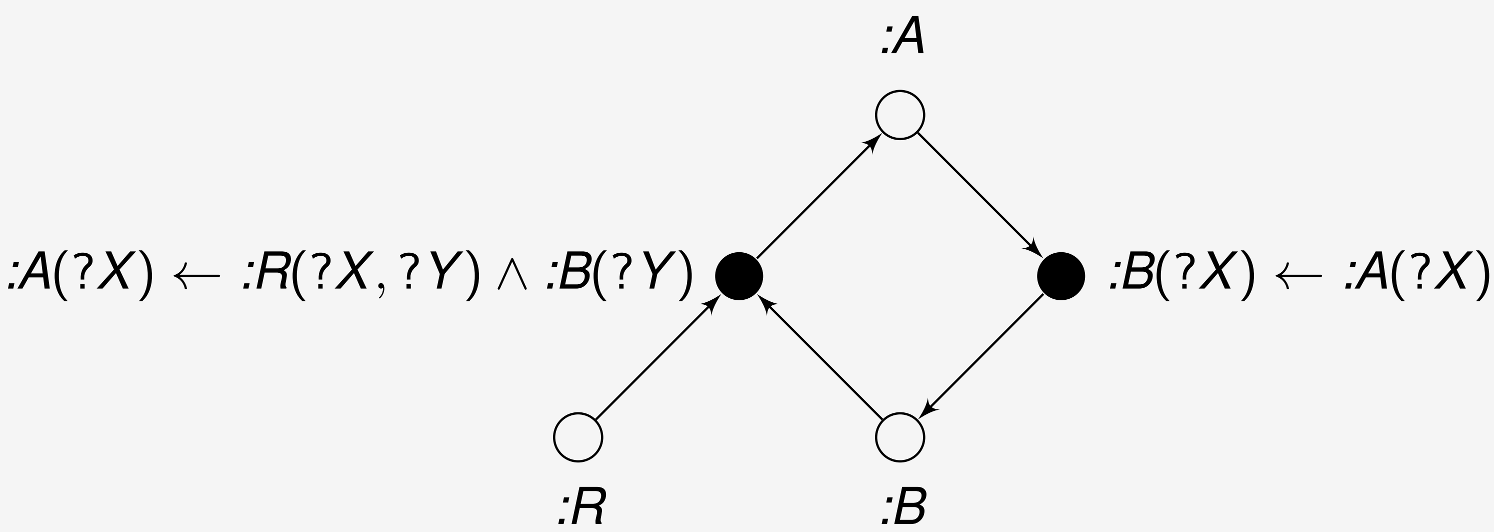
- Reachability:

$$:\!Reachable(?Y) \leftarrow :\!Reachable(X) \wedge :\!connected(?X, ?Y)$$
$$:\!Reachable(:\!source)$$

- Transitivity:

$$:\!connected(?X, ?Z) \leftarrow :\!connected(?X, ?Y) \wedge :\!connected(?Y, ?Z)$$

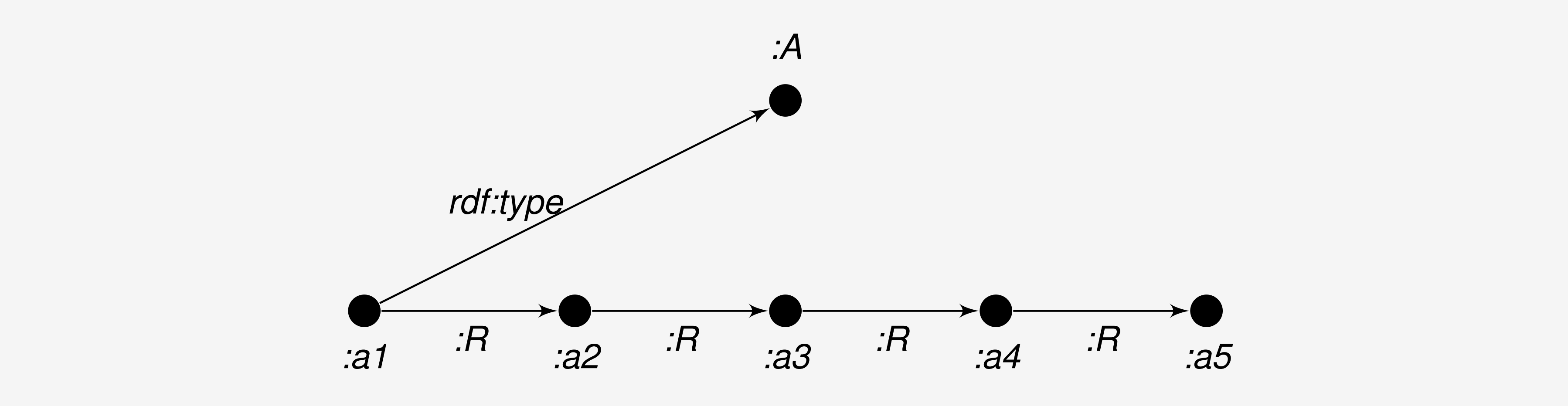
# Rule-Goal Graph




- A program is recursive if its rule-goal graph contains a cycle

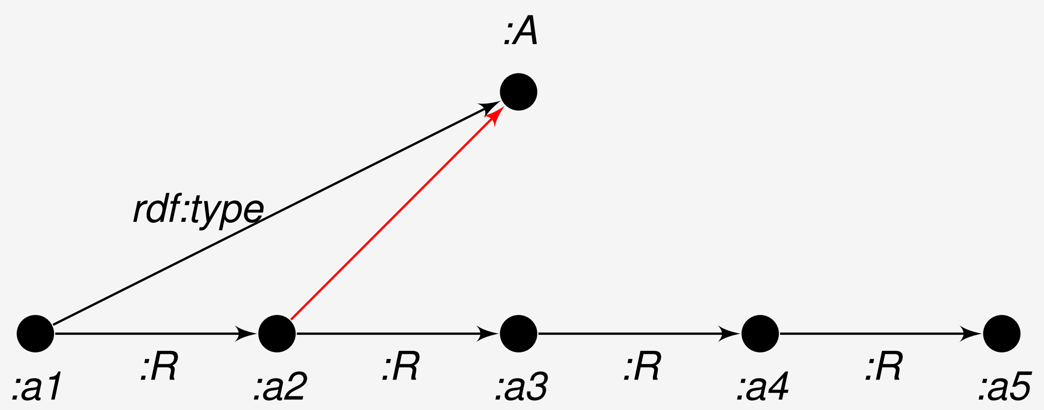
# SEMANTICS OF DATALOG

- Iterative semantics: apply rules as long as new facts are derived
- Example rule: $\langle ?Y, \mathit{rdf{:}type}, {:}A\rangle \leftarrow \langle ?X, \mathit{rdf{:}type}, {:}A\rangle \wedge \langle ?X, {:}R, ?Y\rangle$



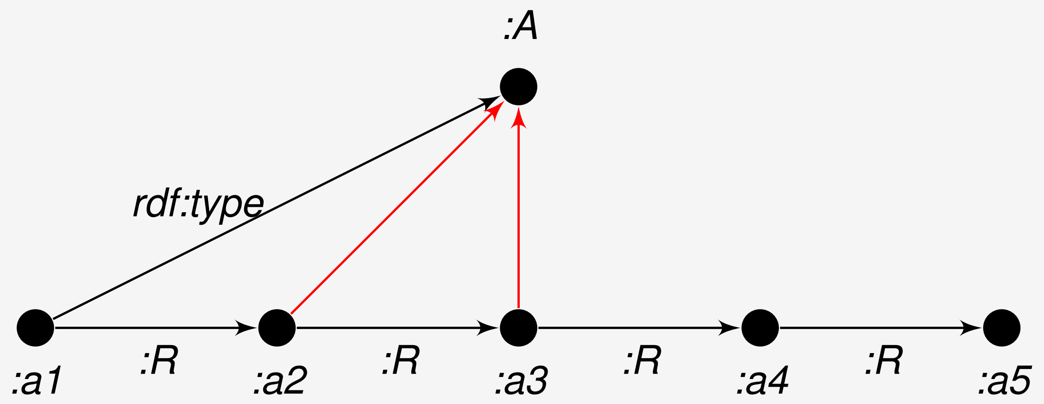
# Semantics of Datalog

- Iterative semantics: apply rules as long as new facts are derived
- Example rule: $\langle ?Y, \textit{rdf:type}, \textit{:A}\rangle \leftarrow \langle ?X, \textit{rdf:type}, \textit{:A}\rangle \wedge \langle ?X, \textit{:R}, ?Y\rangle$



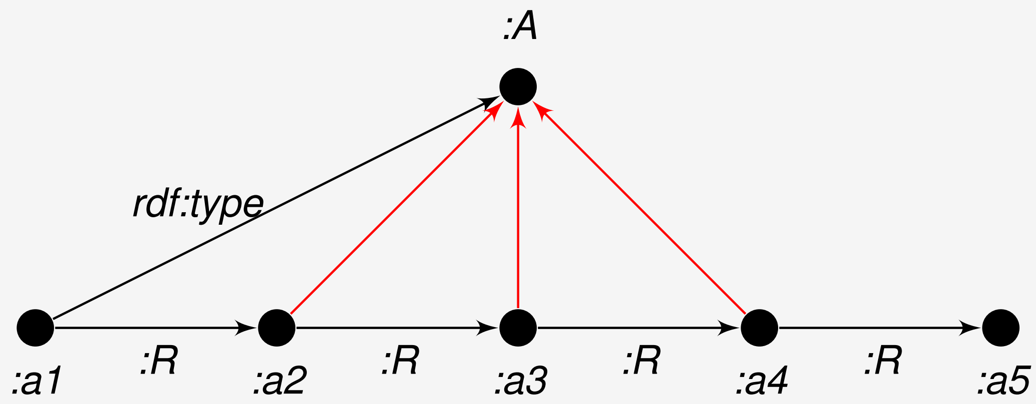
# Semantics of Datalog

- Iterative semantics: apply rules as long as new facts are derived
- Example rule: $\langle ?Y, \mathit{rdf{:}type}, \mathit{{:}A} \rangle \leftarrow \langle ?X, \mathit{rdf{:}type}, \mathit{{:}A} \rangle \wedge \langle ?X, \mathit{{:}R}, ?Y \rangle$



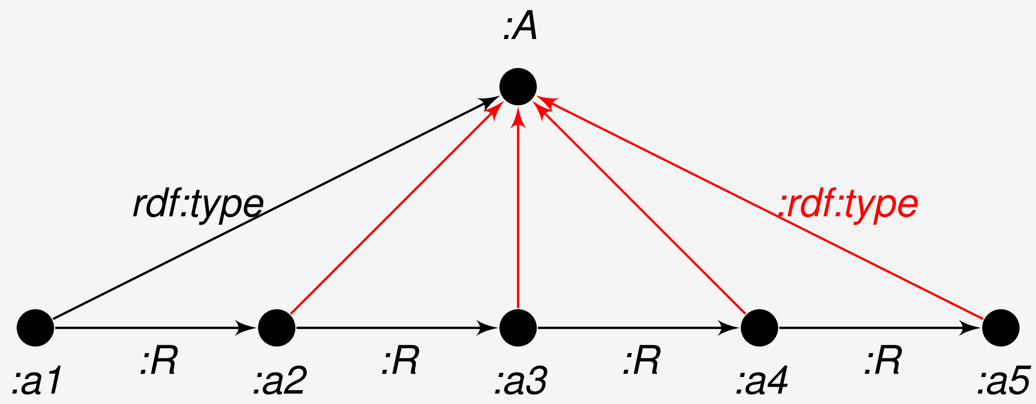
# Semantics of Datalog

- Iterative semantics: apply rules as long as new facts are derived
- Example rule: $\langle ?Y, \mathit{rdf{:}type}, {:}A \rangle \leftarrow \langle ?X, \mathit{rdf{:}type}, {:}A \rangle \wedge \langle ?X, {:}R, ?Y \rangle$



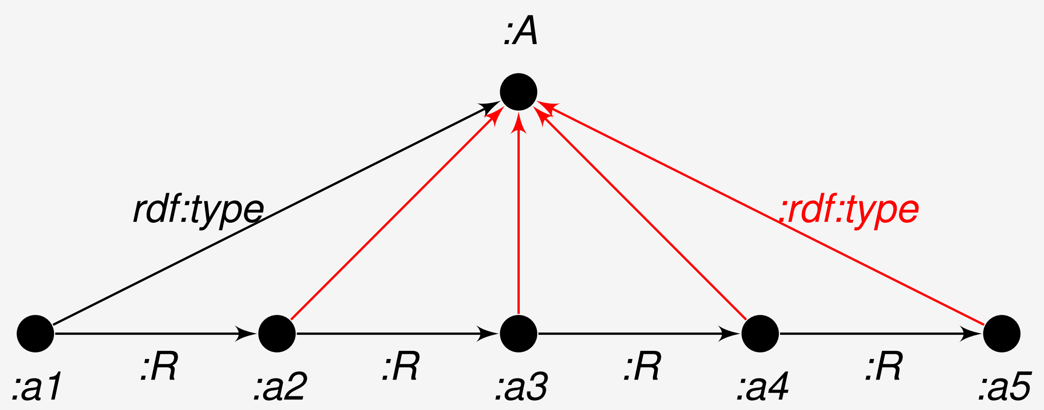
# SEMANTICS OF DATALOG

- Iterative semantics: apply rules as long as new facts are derived
- Example rule: $\langle ?Y, \mathit{rdf{:}type}, \mathit{{:}A}\rangle \leftarrow \langle ?X, \mathit{rdf{:}type}, \mathit{{:}A}\rangle \wedge \langle ?X, \mathit{{:}R}, ?Y\rangle$



# Semantics of Datalog

- Iterative semantics: apply rules as long as new facts are derived
- Example rule: $\langle ?Y, \mathit{rdf{:}type}, {:}A\rangle \leftarrow \langle ?X, \mathit{rdf{:}type}, {:}A\rangle \wedge \langle ?X, {:}R, ?Y\rangle$




- The number of iterative steps depends on the program and the data
  - Cannot be determined in advance by just looking at the program
  - Crucial aspect of recursion
- Semantics just specifies the meaning: implementation can be different

# WEB ONTOLOGY LANGUAGE (OWL)

- Benefits of OWL at a glance:
  - Decidable, but yet very expressive fragment of datalog$^{\pm,\vee}$
  - More user-friendly representation style (no variables)
  - W3C standard (`http://www.w3.org/TR/owl2-overview/`)
- Can describe complex concepts using class expressions
  - E.g., ‘Hotel located at some beach’, ‘Hotel with exactly two swimming pools’, ‘Not a hotel’, ‘Hotel with only non-smoking rooms’, ‘Hotel or B&B’
  - Features: conjunction, disjunction, negation, existential and universal quantification, and cardinality restrictions
- Can describe class expression hierarchies
  - E.g., ‘Each country is headed by a king or a president’, ‘A kingdom is a country headed only by a king’, ‘Nobody is both a king and a president’, ‘A king is a monarch’, ‘A country headed by a monarch is a monarchy’
- Can express complex role properties
  - ‘A friend of a friend is a friend’, ‘An enemy of an enemy is a friend’, ‘A father’s brother is an uncle’, ‘If *A* is reachable from *B*, then *B* is reachable from *A*’

# FUNCTIONAL-STYLE SYNTAX

```
SubClassOf(
  :Country
  ObjectSomeValuesFrom( :headedBy ObjectUnionOf( :King :President ) )
)

SubClassOf(
  :Kingdom
  ObjectIntersectionOf(
    :Country
    ObjectAllValuesFrom( :headedBy :King )
  )
)

DisjointClasses( :King :President )

SubClassOf( :King :Monarch )

SubClassOf(
  ObjectIntersectionOf(
    :Country ObjectSomeValuesFrom( :headedBy :Monarch )
  )
  :Monarchy
)
```

# OWL/XML Syntax

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#">

<rdf:Class rdf:about="http://skyscanner.net/Country">
  <rdfs:subClassOf>
    <rdf:Restriction>
      <owl:onProperty rdf:resource="http://skyscanner.net/headedBy"/>
      <owl:someValuesFrom>
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
            <owl:Class rdf:about="http://skyscanner.net/King"/>
            <owl:Class rdf:about="http://skyscanner.net/President"/>
          </owl:unionOf>
        </owl:Class>
      </owl:someValuesFrom>
    </rdf:Restriction>
  </rdfs:subClassOf>
</rdf:Class>

</rdf:RDF>
```

- Widely used, but awkward and unreadable $\Rightarrow$ mostly machine-generated!

# Manchester Syntax

```
Prefix: : <http://skyscanner.net/>

:Country
    SubClassOf: :headedBy some ( :King or :President )

:Kingdom
    SubClassOf: :Country and ( :headedBy all :King )

:King
    SubClassOf:   :Monarch
    DisjointWith: :President

:Auxiliary
    EquivalentTo: :Country and ( :headedBy some :Monarch )
    SubClassOf:   :Monarchy
```

- Compact and readable
- Does not cover OWL 2 faithfully → hence the *:Auxiliary* class!

# Relationship to Description Logics (DLs)

- Description logics (DLs) provide the formal underpinning of OWL
  - Studied in-depth in theory
  - Tradeoff between complexity and expressivity is well understood
  - Extensive body of research in practical reasoning

- More compact syntax, used mostly by theoreticians in academic publications:

$$\begin{aligned}
\mathit{:Country} &\sqsubseteq \exists \mathit{:headedBy}.(\mathit{:King} \sqcup \mathit{:President}) \\
\mathit{:Kingdom} &\sqsubseteq \mathit{:Country} \sqcap \forall \mathit{:headedBy}.\mathit{:King} \\
\mathit{:King} \sqcap \mathit{:President} &\sqsubseteq \bot \\
\mathit{:King} &\sqsubseteq \mathit{:Monarch} \\
\mathit{:Country} \sqcap \exists \mathit{:headedBy}.\mathit{:Monarch} &\sqsubseteq \mathit{:Monarchy}
\end{aligned}$$

## RELATIONSHIP TO DESCRIPTION LOGICS (DLS)

# Other OWL Features

- XSD datatypes and (in OWL 2) facets, e.g.,
  - integer, string and (in OWL 2) real, float, decimal, datetime, …
  - minExclusive, maxExclusive, length, …
  - PropertyAssertion( hasAge Meg "17"^^xsd:integer )
  - DatatypeRestriction( xsd:integer xsd:minInclusive "5"^^xsd:integer xsd:maxExclusive "10"^^xsd:integer )

  These are equivalent to (a limited form of) **DL concrete domains**

- Keys
  - E.g., HasKey(Vehicle Country LicensePlate)
    - Country + License Plate is a unique identifier for vehicles

  This is equivalent to (a limited form of) **DL safe rules**

# Other OWL Features

**Keys**

- HasKey axioms provide funcionality similar to keys in relational databases.
- A HasKey axiom is of the form:

$$\text{HasKey}(C(p_1 \ldots p_n)(d_1 \ldots d_m))$$

  where $C$ is a class, $p_i$ is an object property and $d_j$ is a data property.

- Axiom states that no two distinct *named* instances of class $C$ can be related to the same set of individuals and literals via the given properties.

# Other OWL Features

**Keys**

More formally, if ontology $\mathcal{O}$ includes an axiom:

$$\text{HasKey}(C(p_1 \dots p_n)(d_1 \dots d_m))$$

then a model $\mathcal{I}$ of $\mathcal{O}$ has to satisfy the following condition:

For each pair $a, b$ of individuals occurring in $\mathcal{O}$, with $\{a^{\mathcal{I}}, b^{\mathcal{I}}\} \subseteq C^{\mathcal{I}}$, and for each $e \in \Delta^{\mathcal{I}}$, $v \in \Delta^{D}$, $1 \leq i \leq n$ and $1 \leq j \leq m$, if:

- $(a^{\mathcal{I}}, e) \in p_i^{\mathcal{I}} \iff (b^{\mathcal{I}}, e) \in p_i^{\mathcal{I}}$ and
- $(a^{\mathcal{I}}, v) \in d_j^{\mathcal{I}} \iff (b^{\mathcal{I}}, v) \in d_j^{\mathcal{I}}$

then $a^{\mathcal{I}} = b^{\mathcal{I}}$.

# Other OWL Features

For example, if an ontology $\mathcal{O}$ includes the following axiom and assertions

```
HasKey( :Person ( :hasChild ) ( :hasGender ) )
ClassAssertion( :Person :Elizabeth )
ObjectPropertyAssertion( :hasChild :Elizabeth :Mary )
DataPropertyAssertion( :hasGender :Elizabeth "F" )
ClassAssertion( :Person :Liz )
ObjectPropertyAssertion( :hasChild :Liz :Mary )
DataPropertyAssertion( :hasGender :Liz "F" )
```

then $\mathcal{O}$ entails `SameIndividual( :Elizabeth :Liz )`. If $\mathcal{O}$ additionally includes the following axioms and assertions

# Relationship to Description Logics (DLs)

## Other OWL Features

ClassAssertion( ObjectSomeValuesFrom( *hasFriend* $:P$ ) $:John$ )
SubClassOf( $:P$ ObjectHasValue( *hasChild* $:Mary$ ) )
SubClassOf( $:P$ DataHasValue( *hasGender* "F" ) )
SubClassOf( $:P$ $:Person$ ), SubClassOf( $:P$ $:Happy$ ),
ClassAssertion( ObjectComplementOf( $:Happy$ ) $:Liz$ )

then is $\mathcal{O}$ inconsistent?

# Other OWL Features

## Anonymous Individuals

- Recall that ABox assertions in OWL directly correspond to RDF triples of the form $\langle a, \mathit{rdf{:}type}, C\rangle$ and $\langle a, p, b\rangle$, where $C$ is a class, $p$ is a property, and $a, b$ are IRIs.
- Unlike standard DLs, $a$ and $b$ do not have to be named individuals, but can also be RDF blank nodes.
- Blank nodes are denoted by the use of $\_:$ as an IRI prefix (e.g., $\_ : x$), and are treated as variables that are existentially quantified at the outer level of the ABox.
- In OWL, blank nodes used in ABox assertions are called anonymous individuals.

# Relationship to Description Logics (DLs)

## Other OWL Features

For example, the assertions

$$\texttt{ObjectPropertyAssertion( } :\!hasFriend\ :\!Liz\ \_\!:\!x \texttt{ )}$$
$$\texttt{ObjectPropertyAssertion( } :\!livesIn\ \_\!:\!x\ \_\!:\!y \texttt{ )}$$
$$\texttt{ObjectPropertyAssertion( } :\!livesIn\ :\!Mary\ \_\!:\!y \texttt{ )}$$

assert that :*Liz* has a friend who lives in the same place as :*Mary* without explicitly naming the friend or the place where they live; they are semantically equivalent to a first-order logic sentence of the form

$$\exists x \exists y\, (hasFriend(Liz, x) \wedge livesIn(x, y) \wedge livesIn(Mary, y)).$$

These assertions can also be written as a semantically equivalent $\mathcal{SROIQ}$ concept assertion

$$Liz : \exists hasFriend.(\exists livesIn.(\exists livesIn^{-}.\{Mary\})),$$

# Relationship to Description Logics (DLs)

## Other OWL Features

### Metamodelling

In some applications it may be desirable to use the same name for both a class (or property) and an individual. For example, we might want to state that :*Harry* is an instance of :*Eagle*

$$\texttt{ClassAssertion}(\ :\mathit{Eagle}\ :\mathit{Harry}\ )$$

and that :*Eagle* is an instance of :*EndangeredSpecies*

$$\texttt{ClassAssertion}(\ :\mathit{EndangeredSpecies}\ :\mathit{Eagle}\ ).$$

We could then extend our modelling of the domain to describe classes of classes, e.g., by stating that it is illegal to hunt any class of animal that is an instance of :*EndangeredSpecies*; this is often called *metamodelling*. Metamodelling is not possible in a standard DL, where it is usually assumed that the sets **C**, **R** and **I** (of, respectively, concept, role and individual names) are pairwise disjoint, and where class assertions can only be used to describe individual names; i.e., in an assertion $a : C$, $a$ must be an individual name.

RELATIONSHIP TO DESCRIPTION LOGICS (DLS)

# Other OWL Features

- OWL 2 uses a mechanism known as *punning* to provide a simple form of metamodelling while still retaining the correspondence between OWL ontologies and $\mathcal{SROIQ}$ KBs.
- Punning allows for the same IRI to be used as an individual, a class and a property, but IRIs used in the individual, class and property contexts are semantically unrelated.
- This is equivalent to rewriting the ontology by adding unique prefixes such as $i$ :, $c$ : and $p$ : to IRIs according to the context in which they occur. For example:

$$\texttt{ClassAssertion}(\ c{:}Eagle\ \ i{:}Harry\ )$$
$$\texttt{ClassAssertion}(\ c{:}EndangeredSpecies\ \ i{:}Eagle\ )$$

# Other OWL Features

## Annotations

- OWL includes a flexible annotation mechanism that allows for comments and other "non-logical" information to be included in the ontology.
- An OWL annotation consist of an annotation property and a literal, and zero or more annotations can be attached to class, property and individual names, to axioms and assertions, to datatypes, to the ontology as a whole, and even to annotations themselves; for example:

  ClassAssertion( Annotation( *rdfs:comment* "Liz is a person" )
  :*Person* :*Liz* )

# Other OWL Features

**Imports**

- The OWL Import statement provides a mechanism for "importing" the contents of one ontology document into another
- For example, if :ont1 includes the statement:

  Import( :ont2 )

  then :ont1 is treated as though it also includes all of the contents of :ont2 and, recursively, any ontology documents imported by :ont2.
- The OWL specification defines a parsing procedure that extracts ontological content from the current ontology document and all those that it (possibly recursively) imports, while ensuring termination even if ontology documents (directly or indirectly) import each other cyclically.

# Complete vs. Incomplete Knowledge (I)

## Example

- Known fact: ‘Mary is a woman’
- Question: ‘Does Mary have a daughter?’
  - Database/datalog answer: ‘No’ $\rightarrow$ intuitive!
- Question: ‘Does Mary not have a daughter?’
  - Intuitive answer: ‘Don’t know’ $\rightarrow$ not enough information!
  - Database/datalog answer: ‘No’ $\rightarrow$ not in the database, so ‘No’

# Complete vs. Incomplete Knowledge (II)

- Databases/datalog assume complete knowledge
  - Everything that is not provable is false $\rightarrow$ closed-world assumption
  - Appropriate in some cases: flight schedules, corporate profits, . . .
  - Inappropriate in others: mathematics, certain common-sense reasoning, . . .
- Many situations have incomplete knowledge
  - Negative information must be explicitly provable

## Example

- Known facts: ‘Every man is a person’, ‘Garfield is not a person’
- Can deduce ‘Garfield is not a man’ $\rightarrow$ proof by contradiction
  1. Assume the opposite: ‘Garfield is a man and not a person’
  2. By ‘Every man is a person’, we have ‘Garfield is a man, a person, and not a person’
  3. This is a contradiction, so ‘Garfield is a man’ cannot be true
  4. But ‘Either Garfield is a man, or Garfield is not a man’ (aka law of excluded middle)
  5. Hence, ‘Garfield is not a man’ is true

# Classical Negation

- Classical negation $\neg$ works under incomplete knowledge
  - Comes from propositional and first-order predicate logic
  - Very different from database-style *not* from datalog
  - Used in OWL 2 as ObjectComplementOf

## Example

$$:Man(:garfield) \qquad \neg:Person(:garfield) \qquad \forall ?X.[:Person(?X) \Leftarrow :Man(?X)]$$

- Can use $\neg$ in front of facts or rule heads (e.g., $\neg:Person(:garfield)$)
- Material implication $\Leftarrow$ is different from datalog implication $\leftarrow$

$$\left.\begin{array}{l} A \Leftarrow B \\ A \vee \neg B \\ \bot \Leftarrow \neg A \wedge B \\ \neg B \Leftarrow \neg A \end{array}\right\} \text{all equivalent to each other}$$

# Comparing Two Kinds of Implication

| | Material implication | Datalog implication |
|---|---|---|
| Ontology: | $\forall ?X.[:Person(?X) \Leftarrow :Man(?X)]$ | $:Person(?X) \leftarrow :Man(?X)$ |
| Facts: | $:Man(:peter)$<br>$:Man(:paul)$<br>... | $:Man(:peter)$<br>$:Man(:paul)$<br>... |
| Conclusions: | $:Person(:peter)$<br>$:Person(:paul)$<br>... | $:Person(:peter)$<br>$:Person(:paul)$<br>... |
| $\Rightarrow$ No observable difference on negation-free rules and positive facts. | | |
| More facts: | $\neg :Person(:garfield)$ | Syntax error! |
| More conclusions: | $\neg :Man(:garfield)$ | |
| $\Rightarrow$ Difference observable if facts or rules contain negation. | | |

- Lots of theoretical work on integrating the two $\rightarrow$ very hard problem!

# OWL 2 PROFILES

- Reasoning in OWL 2 is of high worst-case computational complexity
  - Undecidable for the RDF version of OWL 2
  - N2EXPTIME for the DL version of OWL 2
- OWL 2 profiles trade some expressivity for lower complexity
  - `http://www.w3.org/TR/owl2-profiles/`
- OWL 2 RL
  - No support for incomplete information
  - Can be implemented fully using datalog (without negation)
  - Targets mainly database-like warehousing-style applications
- OWL 2 QL
  - Incompleteness via existential quantification, but not disjunction
  - No support for recursion
  - Can be implemented using query rewriting
  - Targets virtual information integration
- OWL 2 EL
  - Incompleteness via existential quantification, but not disjunction
  - Supports recursion
  - Tractable query answering
  - Targets applications that rely on expressive taxonomies

# SPARQL Protocol and RDF Query Language

- Current version 1.1
- `http://www.w3.org/TR/sparql11-query/`
- Used to query RDF and OWL systems
- Uses a familiar SELECT-WHERE paradigm
- Two parts:
  - Basic SPARQL $\rightarrow$ roughly as expressive as SQL
    - No recursive queries
  - Property paths in 1.1 version $\rightarrow$ expressivity beyond SQL
    - Supports property paths $\rightarrow$ a form of recursion

# Basic SPARQL

1. Matching of graph patterns
   - Entailment regimes determine semantics of matches
2. Relational algebra over answers to graph patterns
   - Union, subtraction, subqueries, built-in expressions, aggregate functions
   - No NULL-values, but variables can be unbound

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX :    <http://skyscanner.net/>
SELECT ?H ?N WHERE {
  ?H rdf:type :Hotel ; :hasName ?N ; :hasAmenity :Wifi .
}

SELECT ?H ?N ?D WHERE {
  ?H rdf:type :Hotel ; :name ?N . OPTIONAL { ?H :offersDiscount ?D }
}

SELECT ?A WHERE {
  { ?A rdf:type :Hotel } UNION { ?A rdf:type :Hostel }
}

SELECT ?H WHERE {
  ?H rdf:type :Hotel } MINUS { ?H :locatedIn :Prague }
}
```

# Property Paths

- Terms can be connected by regular expressions over properties

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX :    <http://skyscanner.net/>
SELECT ?H WHERE {
  ?H rdf:type :Hotel ; :inCity/:inCountry :Germany .
}

SELECT ?C1 ?C2 WHERE {
  ?C1 rdf:type :Country (:hasLandBorderWith/:hasLandBorderWith?) ?C2 .
}

SELECT ?C WHERE {
  ?C rdf:type :Country ; :hasLandBorderWith+ :Germany .
}
```

- Regular expressions support a form of recursion
- Blurs the distinction between reasoning and querying
- Such queries are common in graph databases (e.g., Neo4j)