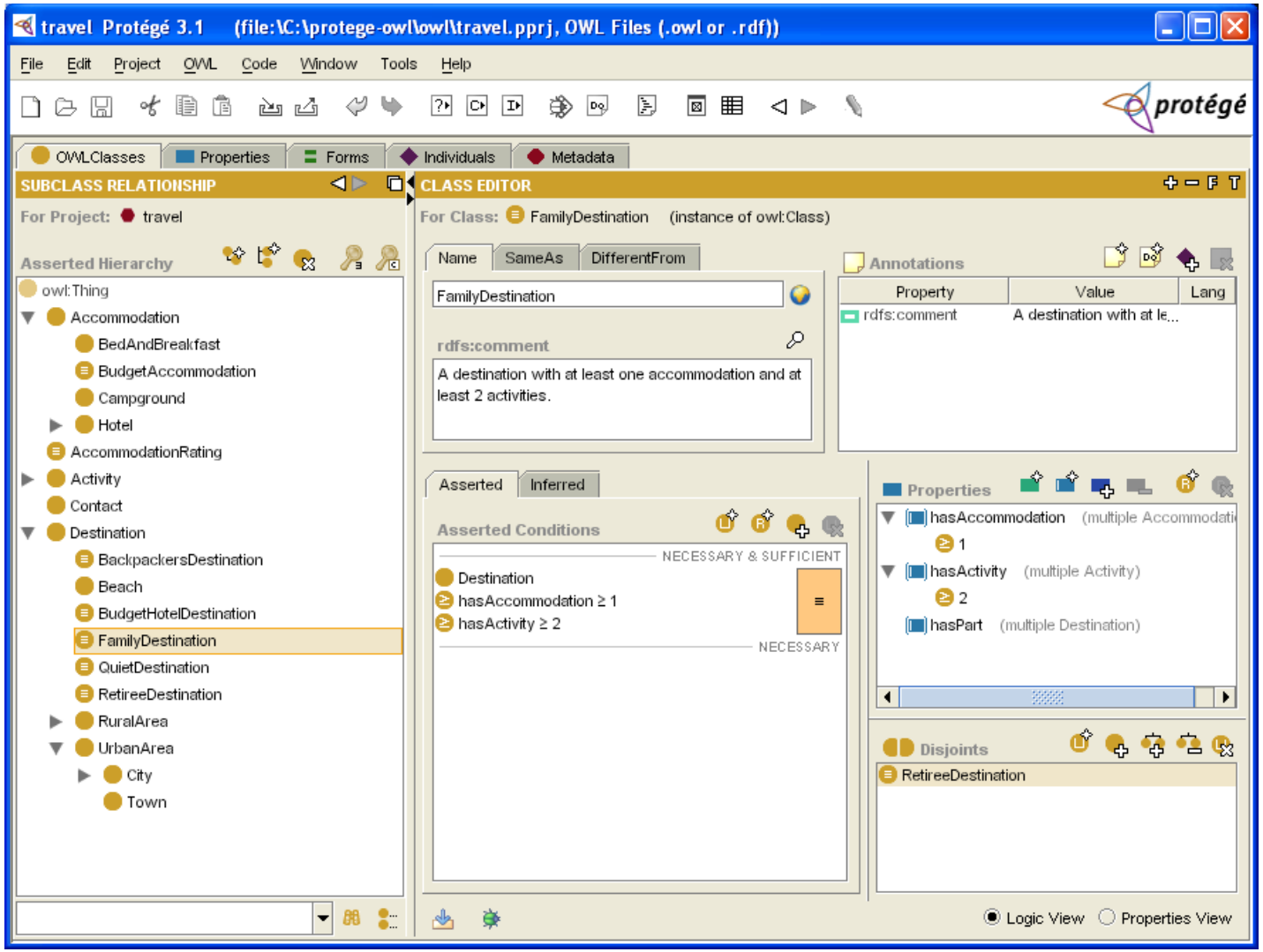
Chapter 3

RDF Schema

Introduction

- RDF has a very simple data model
- RDF Schema (RDFS) enriches the data model, adding vocabulary & associated semantics for
  - Classes and subclasses
  - Properties and sub-properties
  - Typing of properties
- Support for describing simple ontologies
- Adds an object-oriented flavor
- But with a logic-oriented approach and using “open world” semantics

RDFS is a simple KB Language



Several widely used Knowledge-Base tools can import and export in RDFS, including Stanford's Protégé KB editor

RDFS Vocabulary

RDFS introduces the following terms, giving each a meaning w.r.t. the rdf data model

- Terms for classes
  - rdfs:Class
  - rdfs:subClassOf
- Terms for properties
  - rdfs:domain
  - rdfs:range
  - rdfs:subPropertyOf
- Special classes
  - rdfs:Resource
  - rdfs:Literal
  - rdfs:Datatype
- Terms for collections
  - rdfs:member
  - rdfs:Container
  - rdfs:ContainerMembershipProperty
- Special properties
  - rdfs:comment
  - rdfs:seeAlso
  - rdfs:isDefinedBy
  - rdfs:label

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

Modeling the semantics in logic

- We could represent any RDF triple with a binary predicate in logic, e.g.

  type(john, human)
  age(john, 32)
  subclass(human, animal)

- But traditionally we model a class or type as a **unary** predicate

  human(john)
  age(john, 32)
  subclass(human, animal)

SW Classes and Instances

- We distinguish between
  - Concrete "things" (individual objects) in the domain: *CMSC201 (a course)*, *Richard Chang (a person),* etc.
  - **Sets of individuals** sharing properties called **classes**: courses, people, students, faculty, etc.
- Individual objects belonging to a class are referred to as **instances** of that class
- Relationship between instances and classes in RDF is through **rdf:type**
- Note similarity to *classes* and *objects* in an OO prog. language (but RDF classes stand for **sets**)

Classes are Useful

Classes let us impose restrictions on what can be stated in an RDF document using the schema

- Like datatypes in a programming language

  e.g,: A+1, where A is an array
- Disallow nonsense from being stated by detecting contradictions

  e.g.: CMSC201 teaches RichardChang
- Enables type inference from use

  e.g.: RichardChang teaches CMSC999 imples that CMSC999 is a course

Preventing nonsensical Statements

- *CMSC201* is taught by *Calculus*
  - We want courses to be taught by lecturers only
  - Restriction on values of the property *"is taught by"* (**range restriction**)
- *Room ITE228* is taught by *Richard Chang*
  - Only *courses* can be taught
  - This imposes a restriction on the objects to which the property can be applied (**domain restriction**)

Class Hierarchies

- Classes can be organized in hierarchies
  - A is a **subclass** of B if and only if every instance of A is also an instance of B
  - We also say that B is a **superclass** of A
- The subclass graph needn't be a tree
  - A class may have multiple superclasses
- In logic:
  - subclass(p, q) $\Leftrightarrow$ p(x) => q(x)
  - subclass(p, q) $\wedge$ p(x) => q(x)

Domain and Range

- The domain & range properties let us associate classes with a property's subject and object
- Only a course can be taught
  - domain(isTaughtBy, course)
- Only an academic staff member can teach
  - range(isTaughtBy, academicStaffMember)
- Semantics in logic:
  - domain(pred, aclass) $\wedge$ pred(subj, obj) $\Rightarrow$ aclass(subj)
  - range(pred, aclass) $\wedge$ pred(subj, obj) $\Rightarrow$ aclass(obj)

Property Hierarchies

- Hierarchical relationships for properties
  - E.g., “is taught by” is a subproperty of “involves”
  - If a course C is taught by an academic staff member A, then C also involves A
- The converse is not necessarily true
  - E.g., A may be the teacher of the course C, or a TA who grades student homework but doesn’t teach
- Semantics in logic
  - subproperty(p, q) $\wedge$ p(subj, obj) $\Rightarrow$ q(sub,obj)
  - e.g, subproperty(mother,parent), mother(p1, p2) $\Rightarrow$ parent(p1, p2)

RDF Schema in RDF

- RDFS's modelling primitives are defined using resources and properties (RDF itself is used!)
- To declare that *"lecturer"* is a subclass of *"academic staff member"*
  - Define resources **lecturer**, **academicStaffMember**, and **subClassOf**
  - define property **subClassOf**
  - Write triple (**subClassOf, lecturer,academicStaffMember**)

Core RDFS Classes

- **rdfs:Resource**: class of all resources
- **rdfs:Class**: class of all classes
- **rdfs:Literal**: class of all literals (strings)
- **rdf:Property**: class of all properties
- **rdf:Statement**: class of all reified statements

Core Properties

- **rdf:type** relates a resource to its class

  A resource is declared to be an instance of a class

- **rdfs:subClassOf** relates a class to a superclass

  All instances of a class are also instances of its superclasses

- **rdfs:subPropertyOf** relates a property to one of its superproperties

  If a property holds between two items, the property's superproperties also hold

Core Properties

- **rdfs:domain** specifies domain of property P
  - The class of those resources that may appear as subjects in a triple with predicate P
  - If domain not specified, any resource can be subject
- **rdfs:range** specifies range of a property P
  - The class of those resources that may appear as object in a triple with predicate P
  - If range not specified, any resource can be object

Examples (in Turtle)

```
# every :lecturer is also a :staffMember
:lecturer a rdfs:Class;
    rdfs:subClassOf :staffMember .

# anything with a phone must be a :staffMember
:phone a rdfs:Property;
    rdfs:domain :staffMember;
    rdfs:range rdfs:Literal .
```

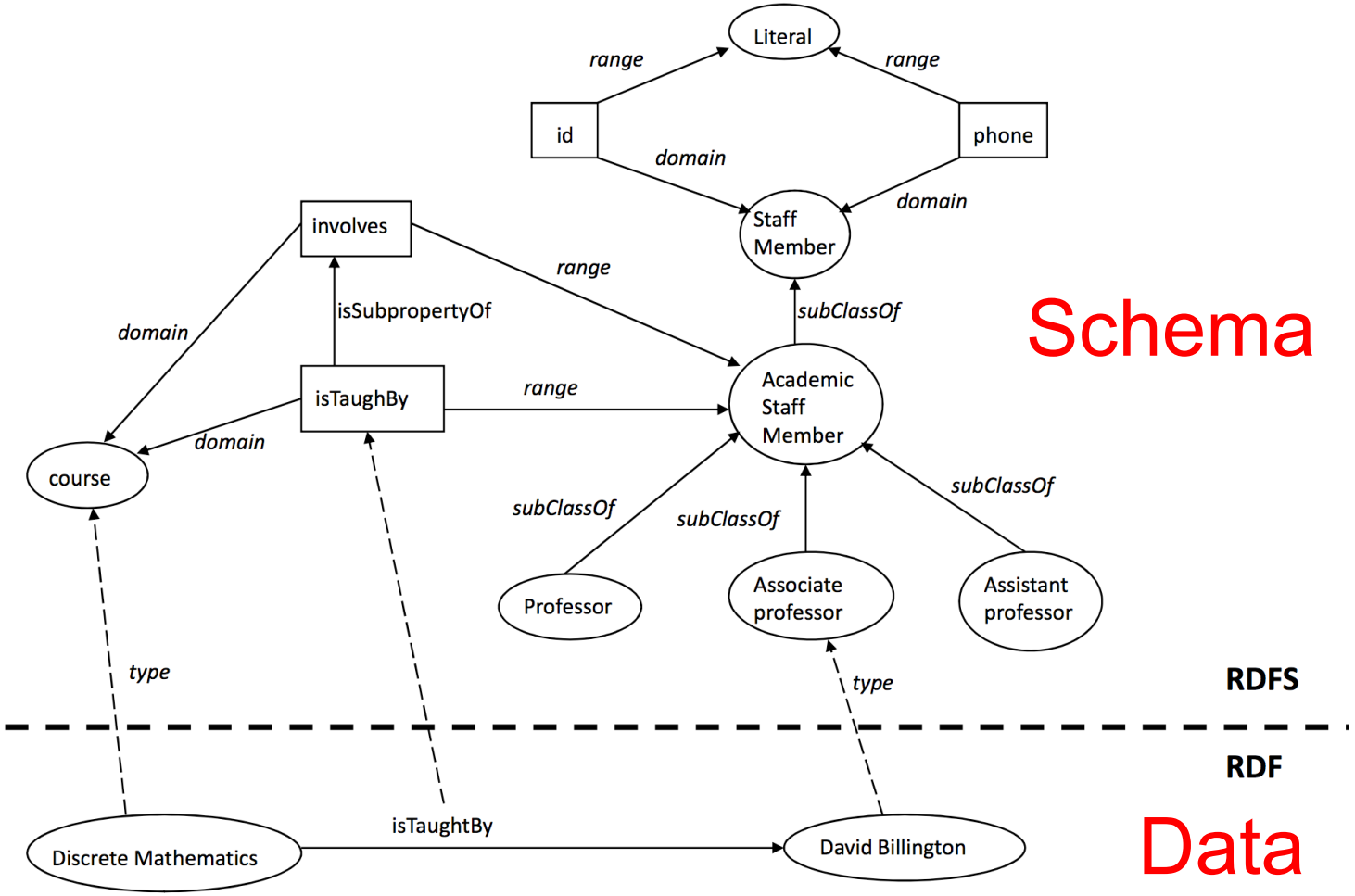
Relationships: Core Classes & Properties

- **rdfs:subClassOf** and **rdfs:subPropertyOf** are transitive, by definition
- **rdfs:Class** is a subclass of **rdfs:Resource**
  - Because every class is a resource
- **rdfs:Resource** is an instance of **rdfs:Class**
  - **rdfs:Resource** is class of all resources, so it is a class
- Every class is an instance of **rdfs:Class**
  - For the same reason

Utility Properties

- **rdfs:seeAlso** relates a resource to another resource that “explains” it. It’s mostly a help for humans and maybe smart AI systems.
- **rdfs:isDefinedBy:** a subproperty of **rdfs:seeAlso** that relates a resource to the place where its definition, typically an RDF schema, is found
- **rfds:comment**. Comments, typically longer text, can be associated with a resource
- **rdfs:label**. A human-friendly label (name) is associated with a resource

Data and schema



Syntactically it's all just RDF. The data part only uses RDF vocabulary and the schema part uses RDFS vocabulary

RDF and RDFS Namespaces

- The RDF, RDFS and OWL namespaces specify some constraints on the 'languages'
  - http://www.w3.org/1999/02/22-rdf-syntax-ns#
  - http://www.w3.org/2000/01/rdf-schema#
  - http://www.w3.org/2002/07/owl#
- Strangely, each uses terms from all three to define its own terms
- Don't be confused: the real semantics of the terms isn't specified in the namespace files

Example

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

bio:Animal a **rdfs:**Class.  This can be inferred

bio:offspring a **rdfs:**Property;

rdfs:domain bio:Animal;  From this!

rdfs:range bio:Animal.

bio:Human **rdfs:**subClassOf bio:Animal.

bio:Dog **rdfs:**subClassOf bio:Animal.

:fido a bio:Dog.

:john a bio:Human;

bio:offspring :fido.  Is this weird?

Ontology and Data

Let's follow best practice and separate our ontology (i.e., schema) file from the data

bio0.ttl is the schema

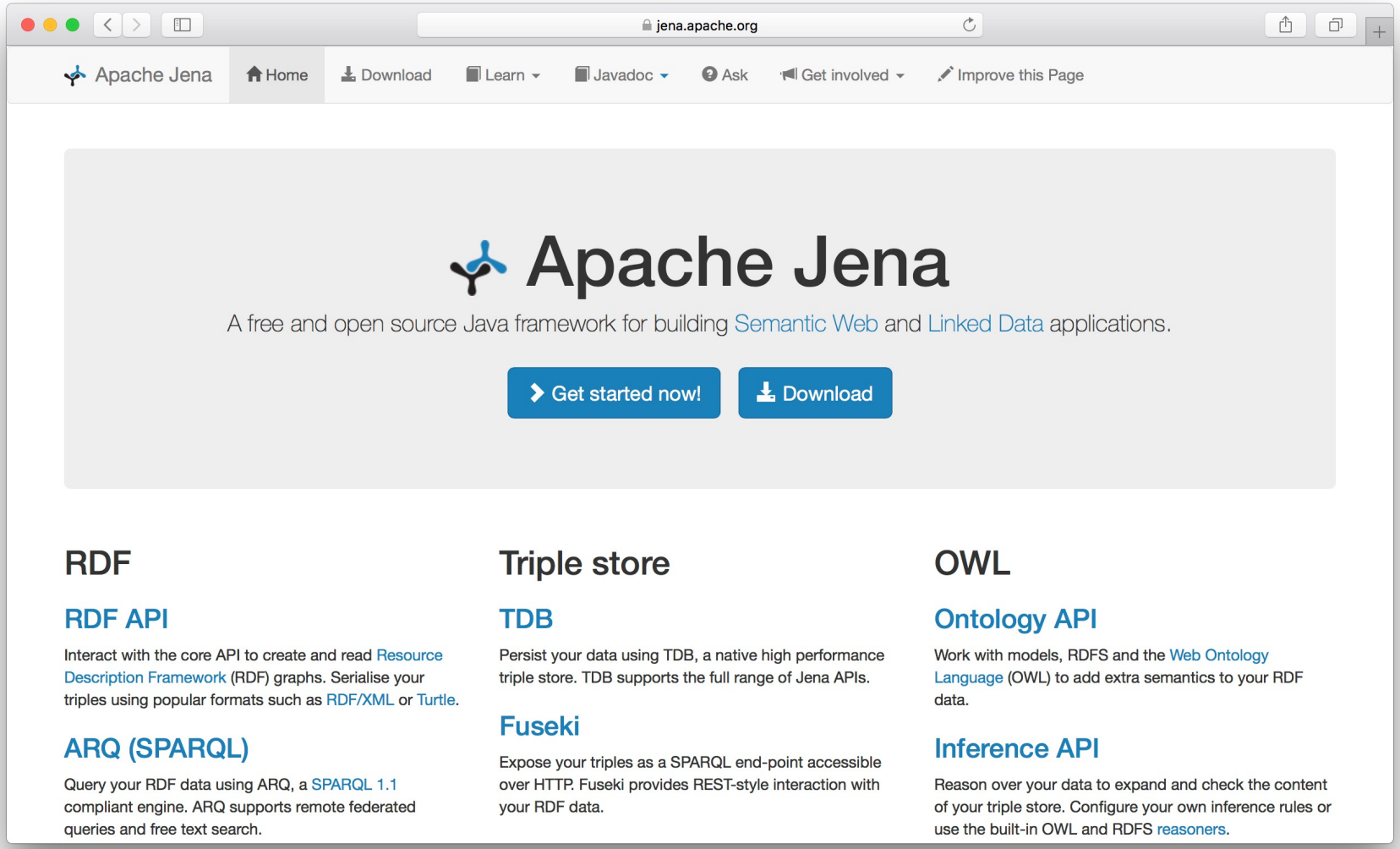
```
# A simple Biology ontology

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

bio:Animal a rdfs:Class.
bio:offspring a rdfs:Property;
   rdfs:domain bio:Animal;
   rdfs:range bio:Animal.
bio:Human rdfs:subClassOf bio:Animal.
bio:Dog rdfs:subClassOf bio:Animal.
```

mybio0.ttl is data using the schema

```
# Some biological data

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix bio: <http://example.com/biology#> .
@prefix :  <http://finin.org/example/#> .


# fido's  a dog!
:fido a bio:Dog.

# john's human & has offspring fido
:john a bio:Human;
   bio:offspring :fido.
```

Apache Jena

Apache Jena is a suite of high-quality, well-maintained, opensource tools in Java for semantic web technology



Jena's riot command

- Jena has a set of command line tools
- Riot can convert between serializations and also do simple rdfs inference
- Let's try it on the example

```
riot mybio0.ttl --rdfs=bio0.ttl --formatted=ttl mybio0.ttl
```

Riot RDFS inference

```
bio> riot --rdfs=bio0.ttl --formatted=ttl mybio0.ttl

@prefix :  <http://finin.org/example/#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix bio:  <http://example.com/biology#> .

:fido a bio:Animal, bio:Dog .

:john a bio:Animal, bio:Human ;
      bio:offspring :fido .

bio>
```

Can a Human have a Dog as an offspring?

Example after RDFS inference

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix bio: <http://example.com/biology#> .
bio:Animal a rdfs:Class.
bio:offspring a rdfs:Property;
    rdfs:domain bio:Animal;
    rdfs:range bio:Animal.
bio:Human rdfs:subClassOf bio:Animal.
bio:Dog rdfs:subClassOf bio:Animal.
:fido a bio:Dog.
:john a bio:Human;
    bio:offspring :fido.
```

There is no way in RDFS to say that the **offspring** of humans are humans and the **offspring** of dogs are dogs.

Solution? child & puppy subProperies

```
bio:Animal a rdfs:Class.
bio:Human rdfs:subClassOf bio:Animal.
bio:Dog rdfs:subClassOf bio:Animal.

bio:offspring a rdfs:Property;
        rdfs:domain bio:Animal;
        rdfs:range bio:Animal.

bio:child rdfs:subPropertyOf bio:offspring;
        rdfs:domain bio:Human;
        rdfs:range bio:Human.

bio:puppy rdfs:subPropertyOf bio:offspring;
        rdfs:domain bio:Dog;
        rdfs:range bio:Dog.
```

mybio1.ttl

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

# john has child mary
:john bio:child :mary.

# fido has puppy :rover
:fido bio:puppy :rover.
```

mybio1.ttl

```
bio> riot --rdfs=bio1.ttl --formatted=TTL mybio1.ttl
@prefix :      <http://finin.org/example/#> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix bio:   <http://example.com/biology#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .

:rover  a bio:Animal, bio:Dog .

:fido a bio:Animal, bio:Dog ;
      bio:offspring :rover ;
      bio:puppy :rover .

:mary a bio:Animal, bio:Human .

:john a bio:Animal , bio:Human ;
      bio:child :mary ;
      bi0:offspring :mary .
```

More Problems?

```
# john's a Human with children mary & fido and puppy rover
:john a bio:Human;
   bio:child :mary, :fido;
   bio:puppy :rover

# fido has puppy rover
:fido bio:puppy :rover.

# rover has puppies fido & rover
:rover bio:puppy :fido, :rover.
```

Suppose we add:

- :john bio:puppy :rover
- :john bio:child :fido
- :rover bio:puppy :fido.
- :rover bio:puppy :rover

More Problems?

```
# john's a Human with children mary & fido and puppy rover
:john a bio:Human;
   bio:child :mary, :fido;
   bio:puppy :rover

# fido has puppy rover
:fido bio:puppy :rover.

# rover has puppies fido & rover
:rover bio:puppy :fido, :rover.
```

Suppose we add:

- :john bio:puppy :rover
- :john bio:child :fido
- :rover bio:puppy :fido.
- :rover bio:puppy :rover

We can also infer facts:

- :john a :Dog
- :fido a :Human

And the weird facts that rover's his own puppy and both rover & fido are each other's puppy

More Problems!

```
# john's a Human with children mary & fido and puppy rover
:john a bio:Human;
   bio:child :mary, :fido;
   bio:puppy :rover

# fido has puppy rover
:fido bio:puppy :rover.

# rover has puppy fido
:rover bio:puppy :fido.
```

Suppose we add:

- :john bio:puppy :rover
- :john bio:child :fido
- :rover bio:puppy :fido.

We infer:

- :john a :Dog
- :fido a :Human
- :fido bio:puppy :rover.

Not like types in OO systems

- Classes differ from types in OO systems in how they are used
  - They are not *constraints* on well-formedness as in most programming languages
- Lack of *negation* & *open world assumption* in RDF+RDFS makes detecting such contradictions **impossible!**
  - Can't say that Dog and Human are disjoint classes
  - Not knowing any individuals who are both doesn't mean it's not possible

No disjunctions or union types

What does this mean?

```
bio:Human rdfs:subClassOf bio:Animal.
bio:Cat rdfs:subClassOf bio:Animal.
bio:Dog rdfs:subClassOf bio:Animal.
bio:hasPet a rdfs:Property;
  rdfs:domain bio:Human;
  rdfs:range bio:Dog;
  rdfs:range bio:Cat.
```

No disjunctions or union types

What does this mean?

Bio:Human rdfs:subClassOf bio:Animal.

bio:Cat rdfs:subClassOf bio:Animal.

Bio:Dog rdfs:subClassOf bio:Animal.

bio:hasPet a rdfs:Property;
  rdfs:domain bio:Human;
  rdfs:range bio:Dog;
  rdfs:range bio:Cat.

Consider adding:
:john bio:hasPet :spot

No disjunctions or union types

What does this mean?

```
bio:Human rdfs:subClassOf bio:Animal.
bio:Cat rdfs:subClassOf bio:Animal.
bio:Dog rdfs:subClassOf bio:Animal.
bio:hasPet a rdfs:Property;
  rdfs:domain bio:Human;
  rdfs:range bio:Dog;
  rdfs:range bio:Cat.
```

:john bio:hasPet :spot

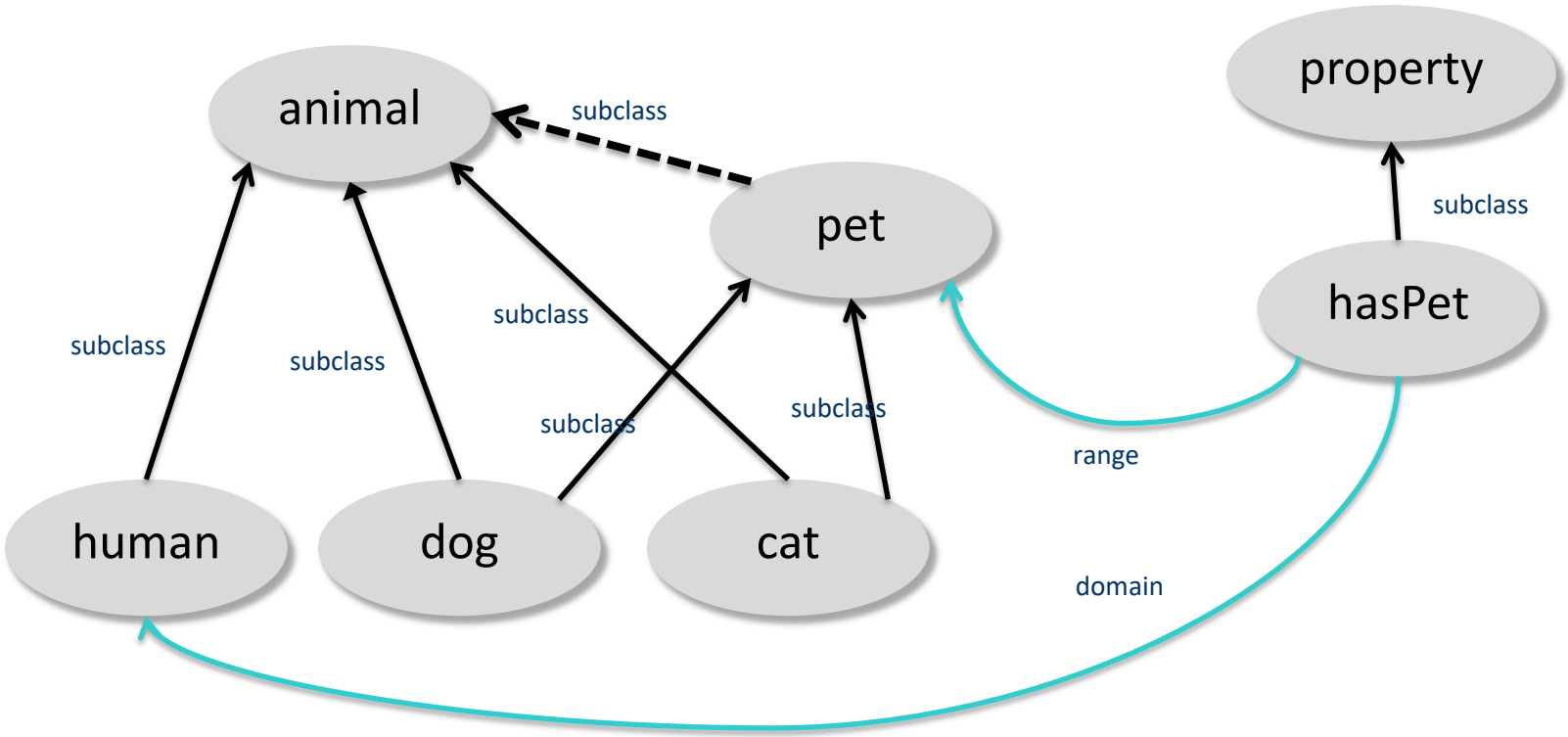
=>

```
:john a bio:Human,
        bio:Animal.
:spot a bio:Dog, bio:Cat,
        bio:Animal.
```

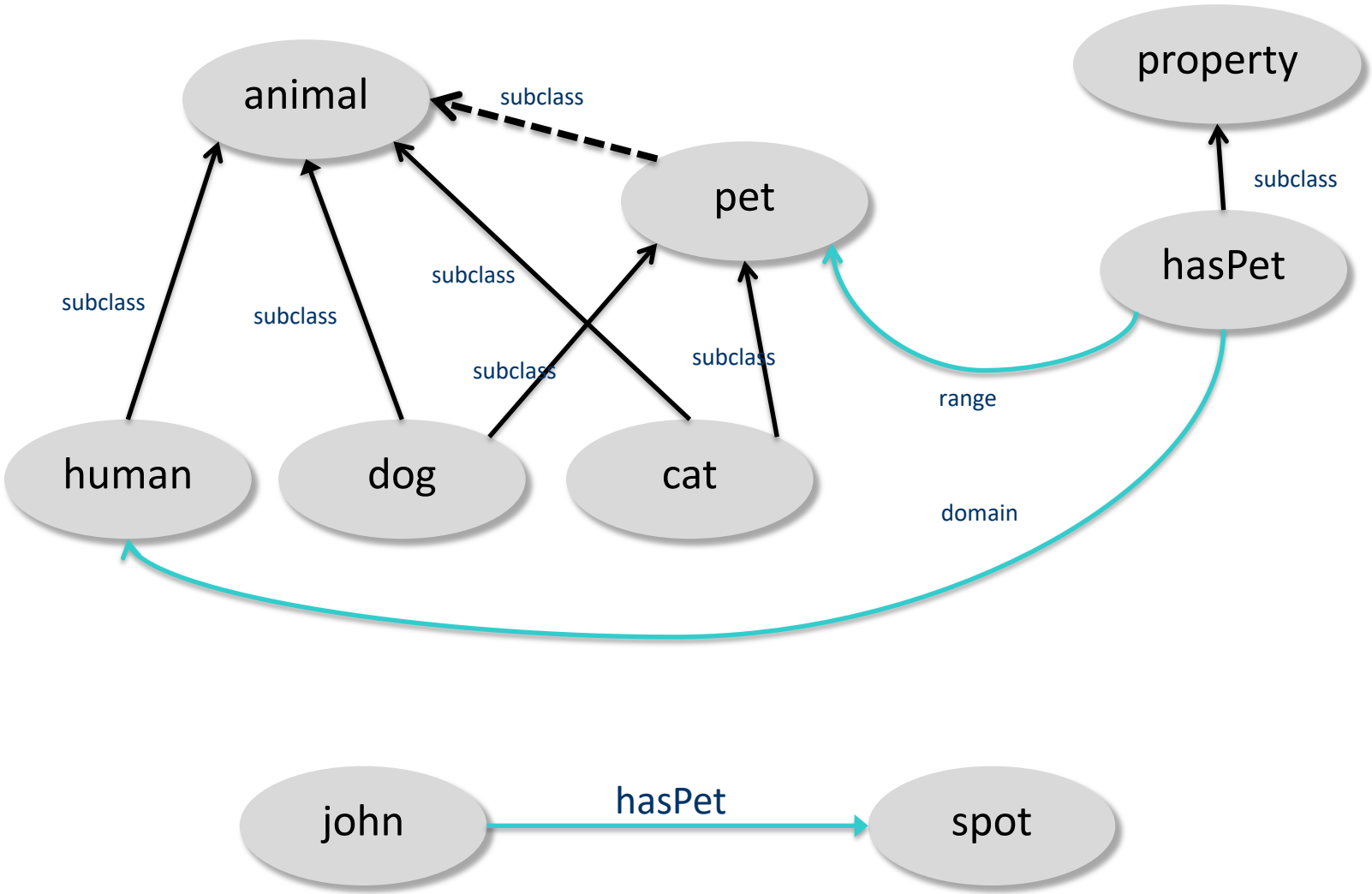
What do we want to say?

- Many different possibilities
  - Only a dog or cat can be an object of hasPet property
  - Dogs and cats and maybe other animals are possible as pets
  - Dogs and cats and maybe other things, not necessarily animals, are possible as pets
  - All dogs and all cats are pets
  - It's possible for some dogs and some cats to be pets
- Not all of these can be said in RDF+RDFS
- We can express all of these in OWL

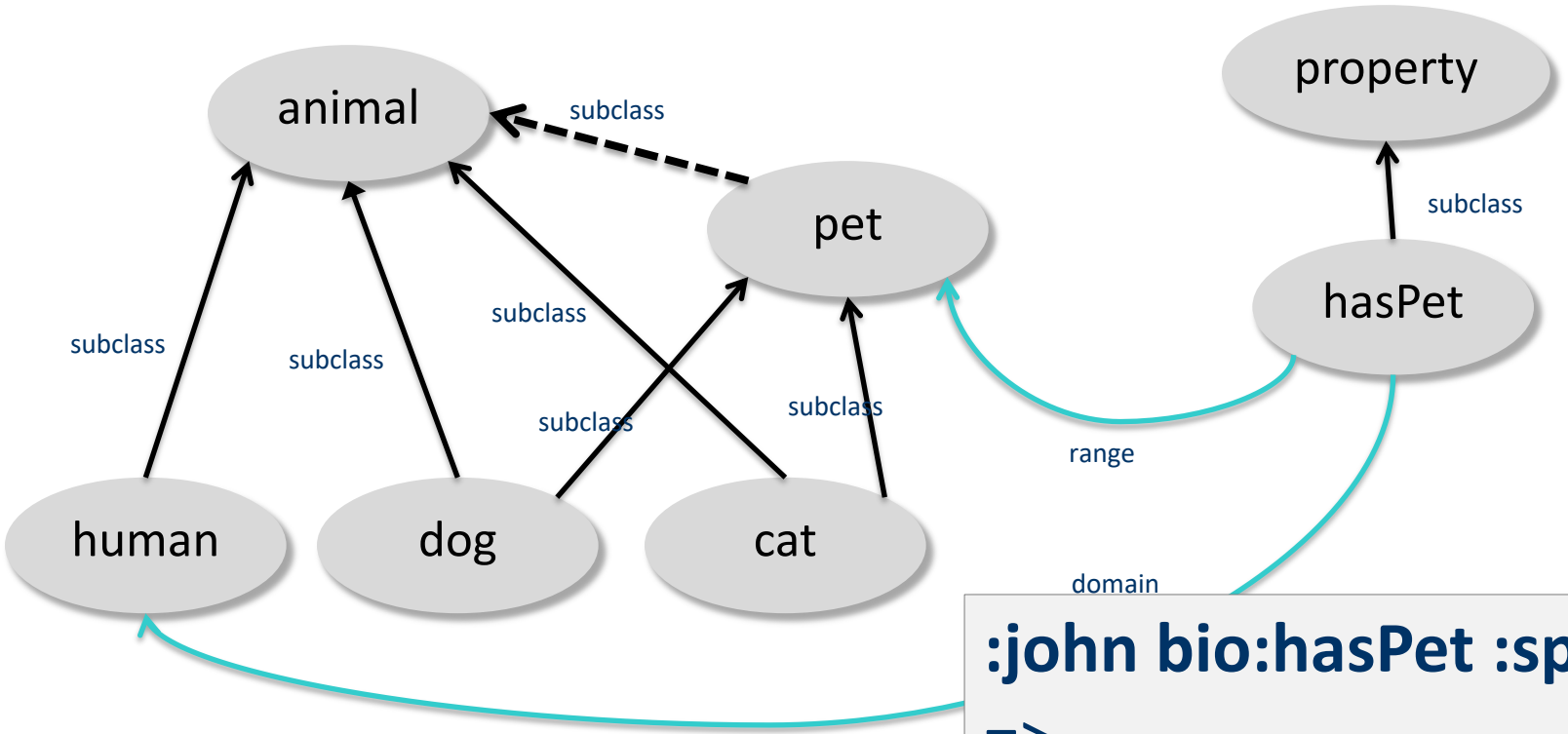
What do we want to say?



What do we want to say?



What do we want to say?



All dogs are pets
All cats are pets
All pets are animals

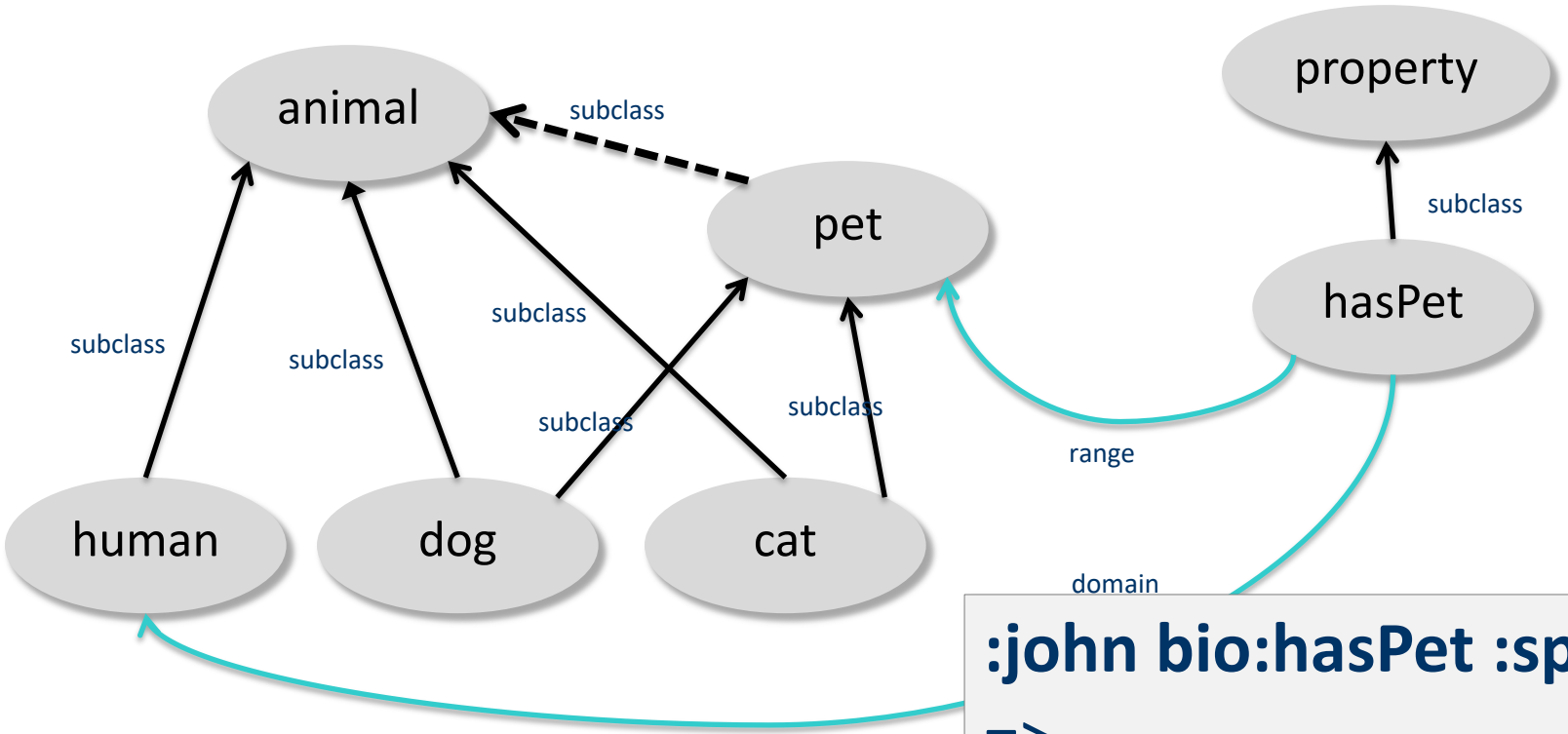
hasPet

:john bio:hasPet :spot

=>

:john a bio:Human,
bio:Animal.
:spot a bio:Pet,
bio:Animal.

What do we want to say?



All dogs are pets
All cats are pets
All pets are animals

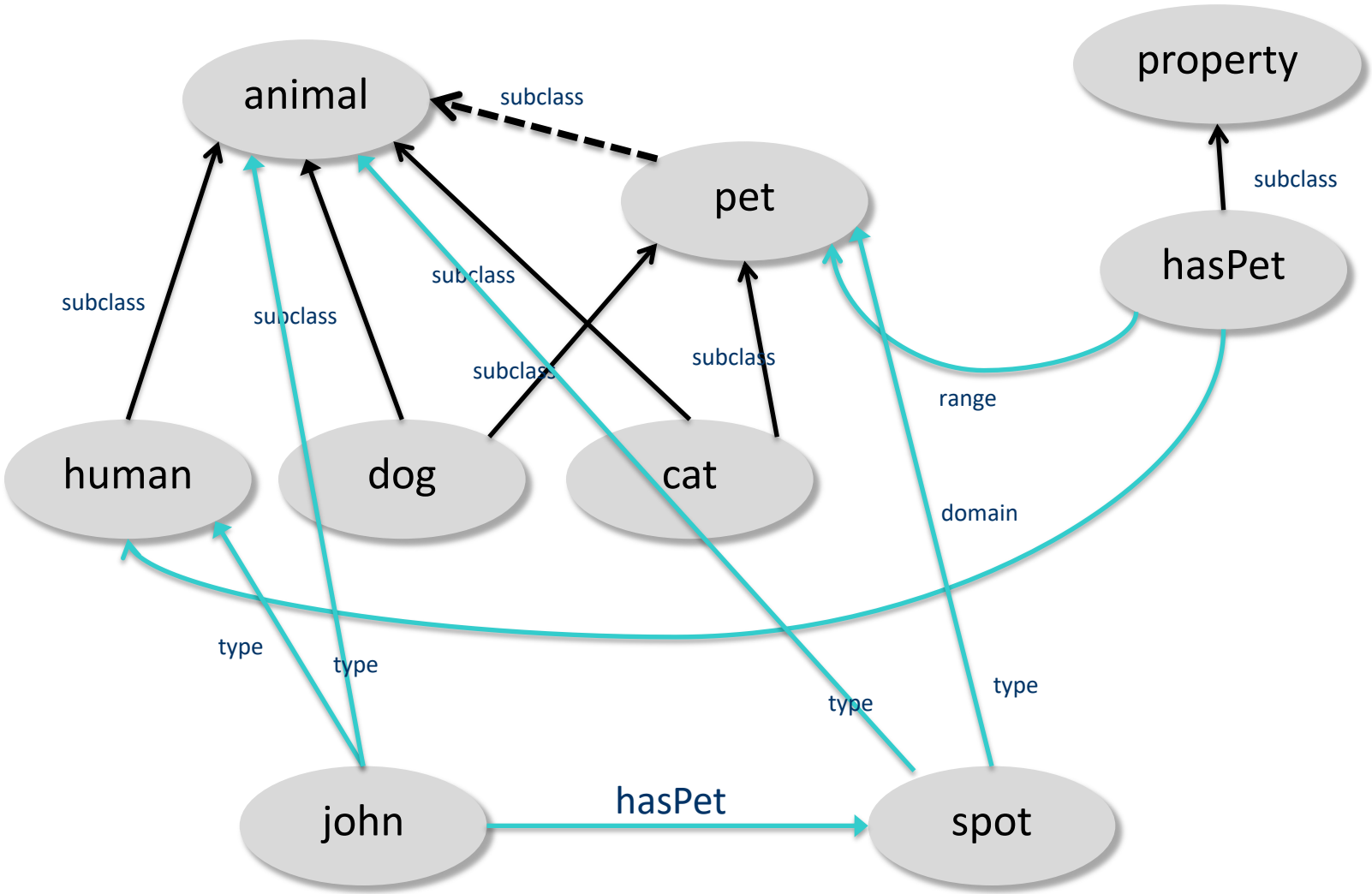
:john bio:hasPet :spot

=>

:john a bio:Human,
bio:Animal.

:spot a bio:Pet,
bio:Animal.

What do we want to say?



Classes and individuals are not disjoint

- In OO systems a thing is either a class or object
  - Many KR systems are like this also
- Not so in RDFS

```
bio:Species rdf:type rdfs:Class.
bio:Dog rdf:type rdfs:Species;
    rdfs:subClassOf bio:Animal.
:fido rdf:type bio:Dog.
```

- rdf:type links an individual to a class it belongs to
- rdfs:subClass links a class to a super-class it is part of

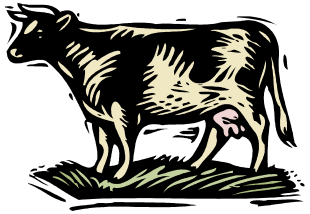
- Adds richness to language but causes problems
  - In OWL DL you can't do this
  - OWL has its own notion of a Class, owl:Class

Inheritance is simple

- No defaults, overriding, shadowing
- What you say about a class is necessarily true of all sub-classes
- A class's properties are not inherited by its members
  - Can't say "Dog's are normally friendly" or even "All dogs are friendly"
  - Meaning of the Dog class is a **set of individuals**
  - Sets cannot be friendly

Set Based Model Theory Example

World



Model

Daisy isA Cow

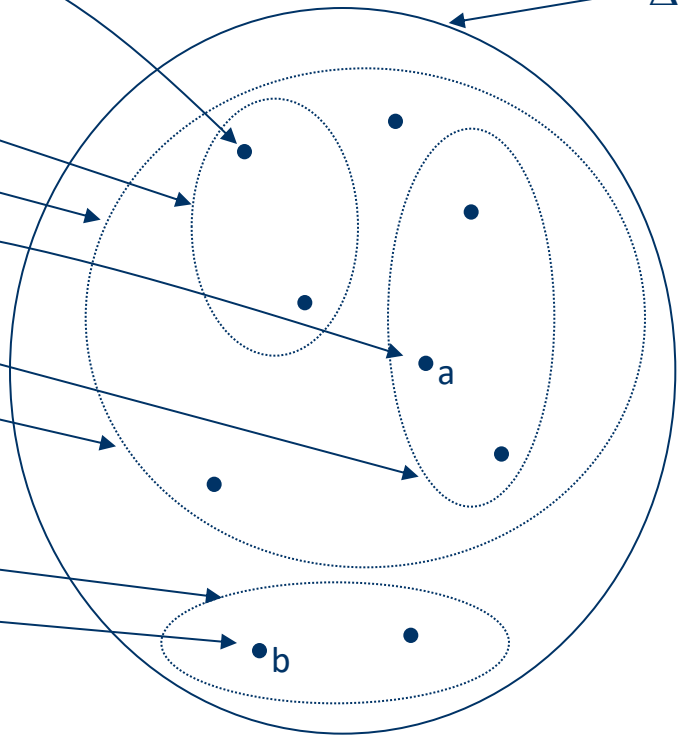
Cow kindOf Animal

Mary isA Person

Person kindOf Animal

Z123ABC isA Car

Interpretation



Mary drives Z123ABC → {... list of facts about individuals ...}

Is RDF(S) better than XML?

Q: For a specific application, should I use XML or RDF?

A: It depends…

- XML's model is
  - a tree, i.e., a strong hierarchy
  - applications may rely on hierarchy position
  - relatively simple syntax and structure
  - not easy to *combine* trees
- RDF's model is
  - a *loose* collections of relations
  - applications may do “database”-like search
  - not easy to recover hierarchy
  - easy to combine relations in one big collection
  - great for the integration of heterogeneous information

RDFS too weak to describe resources in detail

- No *localised range and domain* constraints

  Can't say range of hasChild is person when applied to persons and elephant when applied to elephants

- No *existence/cardinality* constraints

  Can't say all *instances* of person have a mother that is a person, or that persons have exactly two parents

- No *transitive, inverse or symmetrical* properties

  Can't say isPartOf is a transitive property, hasPart is the inverse of isPartOf or that touches is symmetrical

We need RDF terms providing these and other features: this is where OWL comes in

RDF Conclusions

- Simple **data model** based on a **graph,** independent of serializations (e.g., XML or N3)
- Has a **formal semantics** providing a dependable basis for reasoning about the meaning of RDF expressions
- Has an XML serialization, can use XML schema datatypes
- **Open world assumption:** anyone can make statements about any resource
- **RDFS** adds vocabulary with well defined semantics (e.g., Class, subClassOf, etc.)
- OWL addresses some of RDFS's limitations adding richness (and complexity)