# COMP318: Introduction to OWL

**www.csc.liv.ac.uk/~valli/Comp318**


**Dr Valentina Tamma**
**Room: Ashton 2.12**
**Dept of computer science**
**University of Liverpool**
**V.Tamma@liverpool.ac.uk**

# Where were we

- Limitations of RDF(S) as a modelling language
  - Characteristics
  - Wishlist for a Web ontology language

# Reasoning support

- Formal semantics allows the automatic deduction of new facts and possible conflicts between class definitions (consistency):

An ontology language can be provided with formal semantics and reasoning support by mapping it to a known logical formalism and by using the reasoning tools developed for the chosen formalism.

  - These checks are extremely valuable for designing large ontologies, for collaborative ontology design and for sharing and integrating ontologies from various sources:
    - check the consistency of the ontology;
    - check for unintended relations between classes;
    - check for the unintended classification of instances.

# OWL 1 and OWL 2

- OWL: OWL 1 (`http://www.w3.org/TR/owl-features/`) and OWL 2 (`http://www.w3.org/TR/owl2-overview/`)
  - Rationale for OWL
    - **Open world assumption:** the absence of a particular statement means that the statement has not been made explicitly yet.
      - Whether the statement is true or not, and whether it is believed that it is (or would be) true or not is irrelevant.
      - Thus, from the absence of a statement alone, a deductive reasoner cannot infer that the statement is false.
    - Reasonable trade-off between expressivity and scalability
    - Fully declarative semantics.
- OWL 2 DL
  - Fragment of first order predicate logic, decidable
  - Known complexity classes
  - Reasonably efficient for real ontologies + instances

# OWL 1: Three species of OWL

- OWL Lite:
  - Sublanguage of OWL DL but without nominals and XML datatypes:
  - Classification hierarchy
  - Simple constraints
  - It excludes enumerated classes, disjointness statements, and arbitrary cardinality.
  - Reasoning still not tractable.
- OWL DL
  - Sublanguage of OWL Full, it imposes restrictions on the use of OWL/RDFS constructors
  - Application of OWL's constructors to each other not permitted
  - Provides reasonably efficient reasoning support.

# OWL 1: Three species of OWL

- OWL Full
  - Very high expressiveness, uses all of the OWL primitives
  - Fully upward compatible with RDF
  - Losing decidability: no complete or efficient reasoning support
  - All syntactic freedom of RDF (self-modifying):
    - primitives can be combined in arbitrary ways with RDF(S)

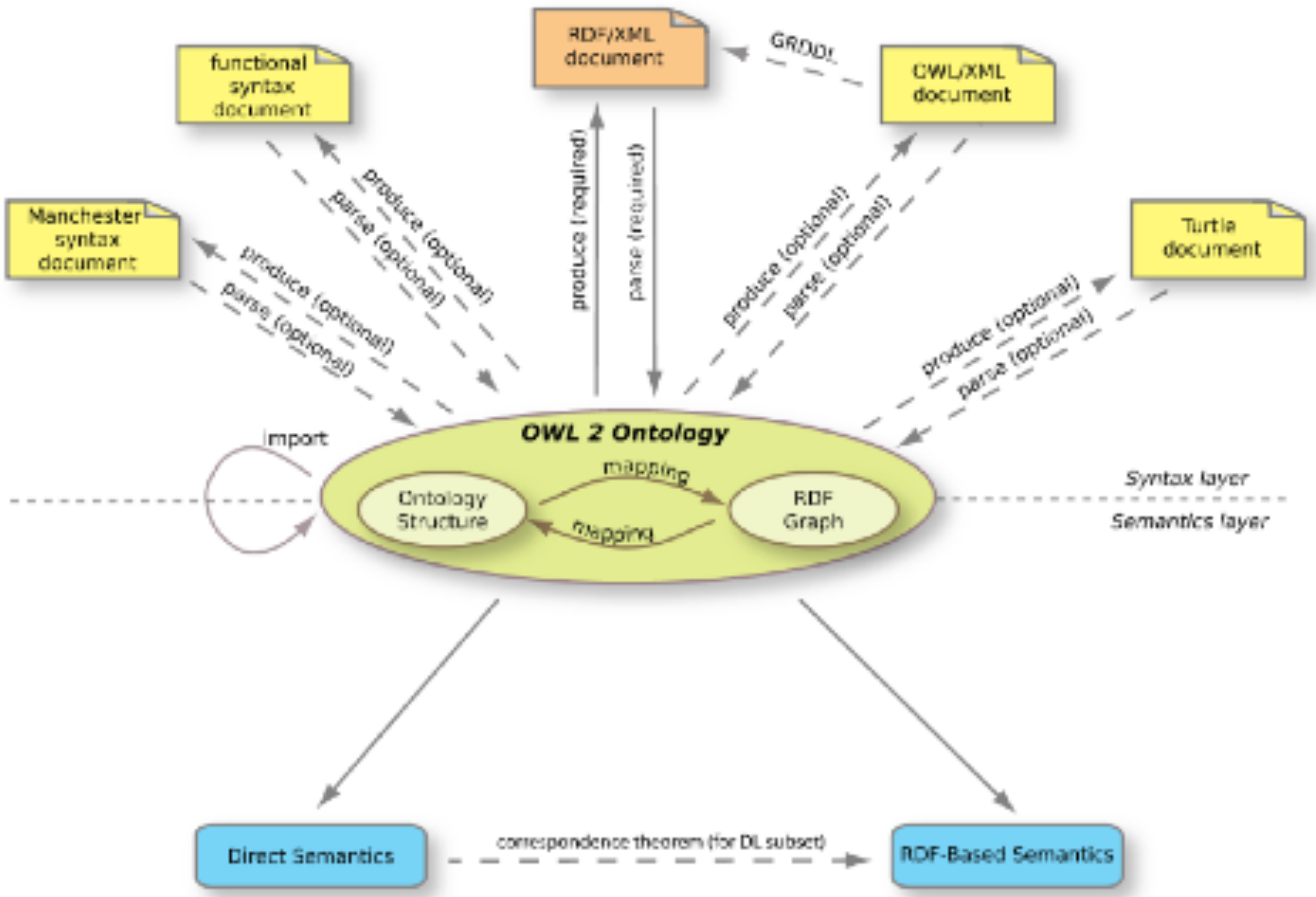
# OWL 2 - Profile

- Sublanguages of OWL2 trading expressive power for efficient reasoning
  - Each supports different application scenarios
- OWL 2 EL
  - very large ontologies, efficient reasoning performance guaranteed at the expenses of expressive power;
- OWL 2 RL
  - subclass axioms understood as rule like implication, with head - superclass and body - subclass
  - different restrictions on subclasses and superclasses
  - allows the integration of OWL with rules

# OWL 2 - Profile

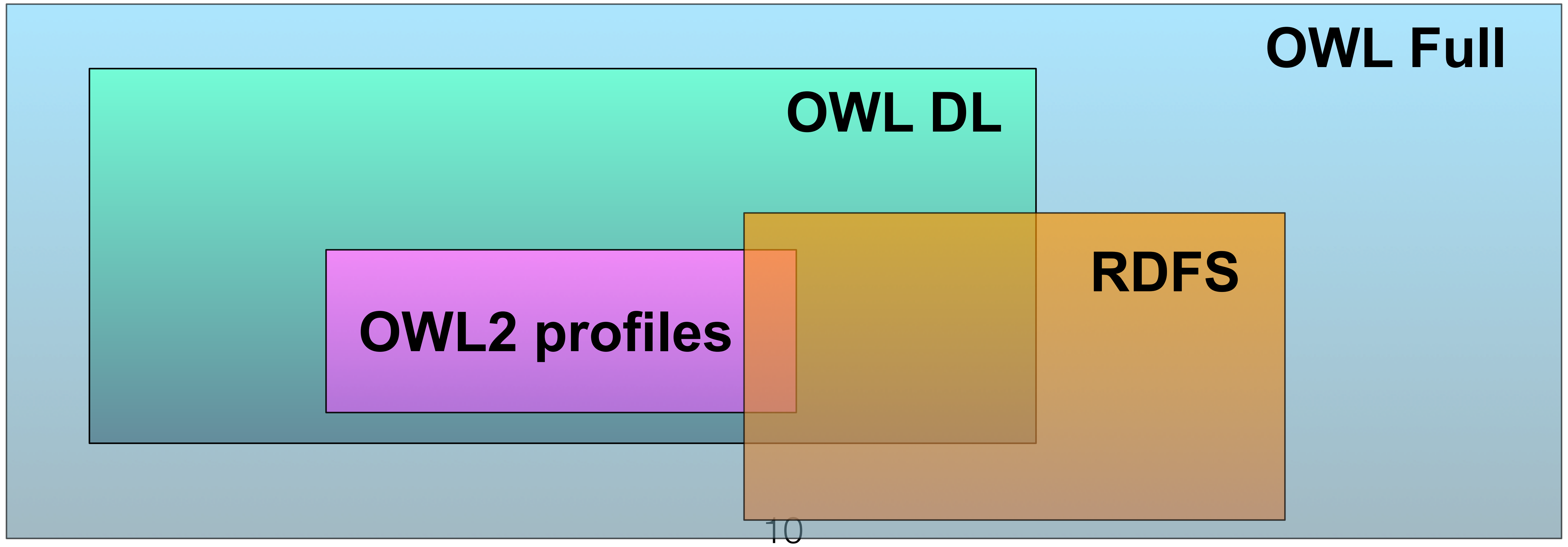
- Sublanguages of OWL2 trading expressive power for efficient reasoning
  - Each supports different application scenarios
- OWL 2 QL
  - useful to query data rich applications
  - different restrictions on subclasses and superclasses
  - suitable for simple, lightweight ontologies with a large number of individuals and it is necessary to access the data directly via SQL queries
  - fast implementation on top of legacy DB systems, relational or RDF

# OWL 2 structure



# Compatibility between OWL and RDF(S)

- OWL uses to a great extent RDF(S)
  - One of the possible syntax formats for OWL is in RDF/XML
  - instances are declared in RDF
    - using `rdf:Description` and `rdf:type`
  - The OWL constructs `owl:Class`, `owl:DatatypeProperty` and `owl:ObjectProperty` are specialisations of the corresponding RDFS constructs



# OWL syntax

- RDF
  - Official exchange syntax
    - Hard for humans
    - RDF parsers are hard to write!
- UML
  - Large user base
- XML
  - Not the RDF syntax, it is not based on RDF conventions
    - Better for humans
    - More XML than RDF tools available
      - `http://www.w3.org/TR/owl-xmlsyntax/`
      - `http://www.w3.org/TR/owl2-mapping-to-rdf`
- Human readable syntax
  - Manchester syntax, used by Protege
  - Functional syntax, more compact and readable
    - `http://www.w3.org/2007/OWL/wiki/ManchesterSyntax`
    - `http://www.w3.org/TR/owl2-manchester-syntax/`
  - Turtle syntax for OWL 2
    - `http://www.w3.org/TR/owl2-primer/`

# OWL ontology header

```
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
         xmlns:xsd ="http://www.w3.org/2001/XMLSchema#"
         xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xml:base="http://www.dcs.bbk.ac.uk/">

<owl:Ontology rdf:about="""">
   <rdfs:comment>An example OWL ontology</rdfs:comment>
   <owl:priorVersion rdf:resource="http://www.dcs.bbk.ac.uk/uni-old-ns"/>
   <owl:imports rdf:resource="http://www.dcs.bbk.ac.uk/person"/>
   <rdfs:label>SCSIS Ontology</rdfs:label>
</owl:Ontology>
...

</rdf:RDF>
```

OWL namespace

Assertions for housekeeping purposes

# Namespaces vs import

```
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
         xmlns:xsd ="http://www.w3.org/2001/XMLSchema#"
         ...

<owl:Ontology rdf:about="""">
   <rdfs:comment>An example OWL ontology</rdfs:comment>
   <owl:priorVersion rdf:resource="http://www.dcs.bbk.ac.uk/uni-old-ns"/>
   <owl:imports rdf:resource="http://www.dcs.bbk.ac.uk/person"/>
   <rdfs:label>SCSIS Ontology</rdfs:label>
</owl:Ontology>

...

</rdf:RDF>
```

OWL namespace

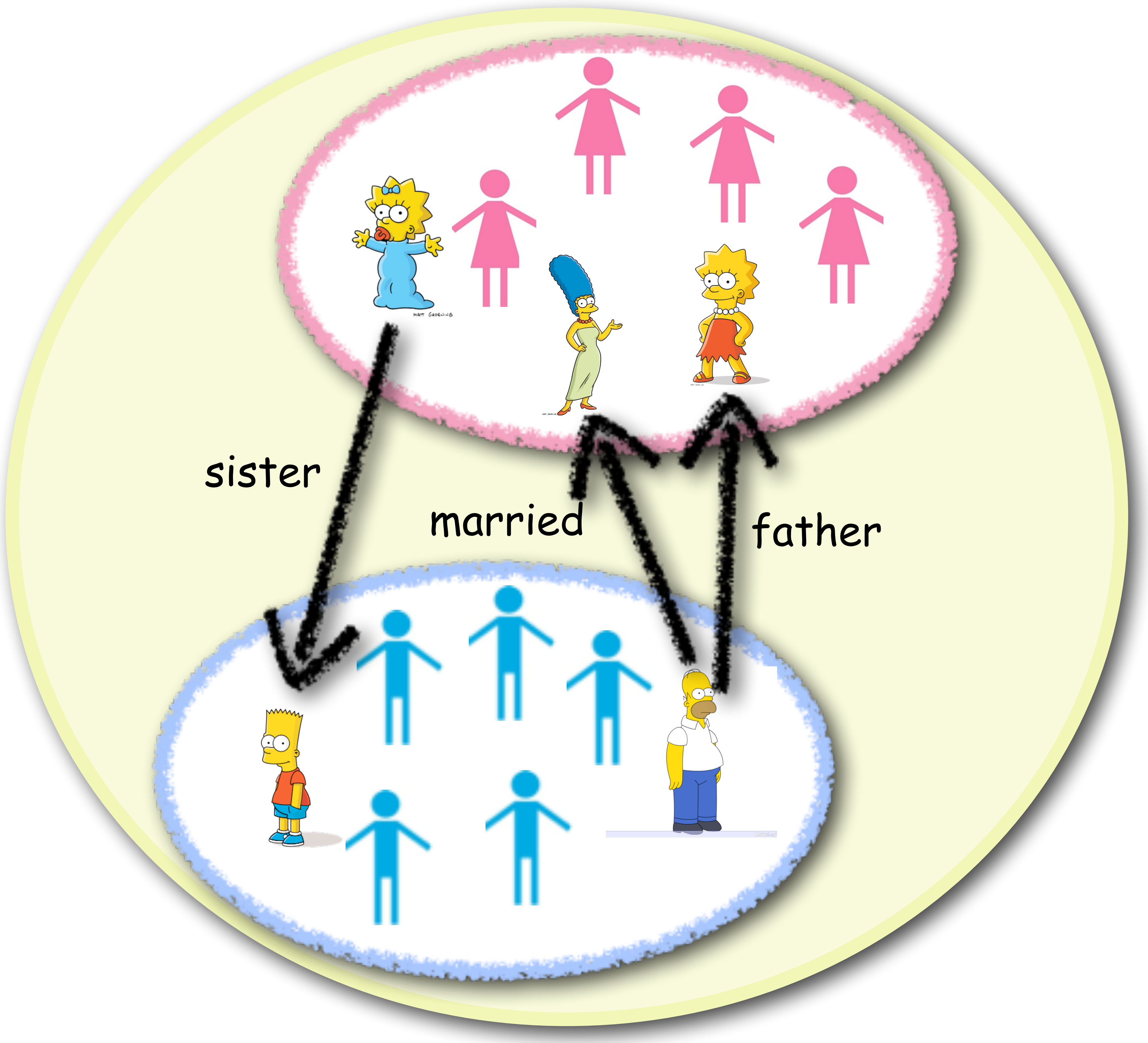
Import of an ontology

Definitions from imported ontologies can be used as if they were asserted in the ontology where they are imported;

Namespaces allow users to disambiguate between similar class or property names defined in different ontologies.

# What do we describe with OWL

- OWL (we assume DL) ontologies describe a world in terms of:
  - individuals (constants): *homer*, *lisa*, ...
  - classes (unary predicates): *man(x)*, *woman(x)*, *lazy(x)*, *clever(x)*, ...
  - properties/roles (binary predicates): *sister_of(x,y)*, *works_for(x,y)*...



# Assertional knowledge (instances)

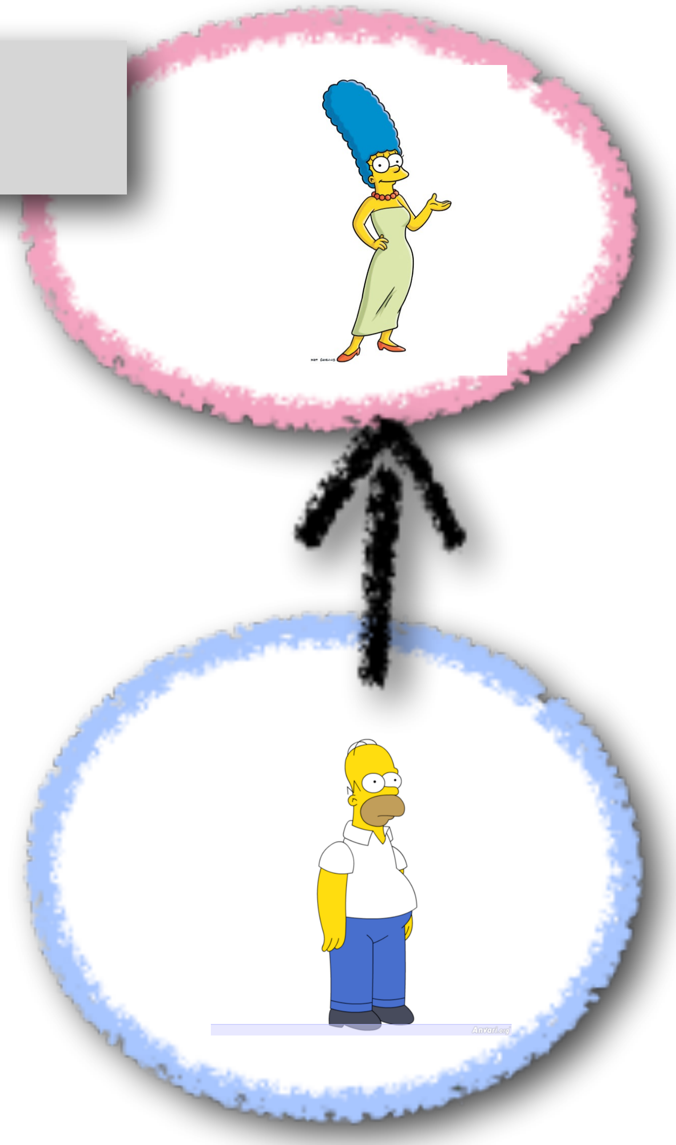
- Instances assert information about named individuals
- It is restricted to what can be stated in RDF
  - class membership: *female(marge)*
  - property membership: *married(marge, homer)*
    - use **`rdf:ID`** and **`rdf:about`** ***almost*** interchangeably

```
<rdf:Description rdf:ID=″marge″>
   <rdf:type rdf:resource=″#woman″/ >
</rdf:Description>
```

It *declares* the individual, it can be used only once in the document

```
<woman rdf:about=″#marge″/>
```

It *references* the individual, it can be used as many times as needed



```
<rdf:Description rdf:about=″marge″>
   <married rdf:resource=″homer″/ >
</rdf:Description>
```

# Unique Name Assumption

- In logics with the unique name assumption, different names always refer to different entities in the world.

```
<rdf:Description rdf:about="#marge"/>
   <owl:sameAs rdf:resource="#margeSimpson">
</rdf:Description>

<rdf:Description rdf:about="#homer">
   <owl:differentFrom rdf:resource="#marge"/>
</rdf:Description>
```

- Despite being based on description logic, for which UNA holds, OWL does not make this assumption
  - explicit constructs are used to express whether two names refer to the same or different entities
    - `owl:sameAs` - URIs refer to the same entity or individual
    - `owl:differentFrom` - URIs refer to different entities or individual

# Terminological knowledge: classes and subclasses

- Classes are defined using owl:class
  - subclass of **rdfs:class**

```
<owl:Class rdf:ID="parents">
   <rdfs:subClassOf rdf:resource="#people"/>
</owl:Class>
```

```
<owl:Class rdf:about="#children">
   <owl:disjointWith rdf:resource="#parents"/>
</owl:Class>
<owl:Class rdf:ID="offspring">
   <owl:equivalentClass rdf:resource="#children"/>
</owl:Class>
```

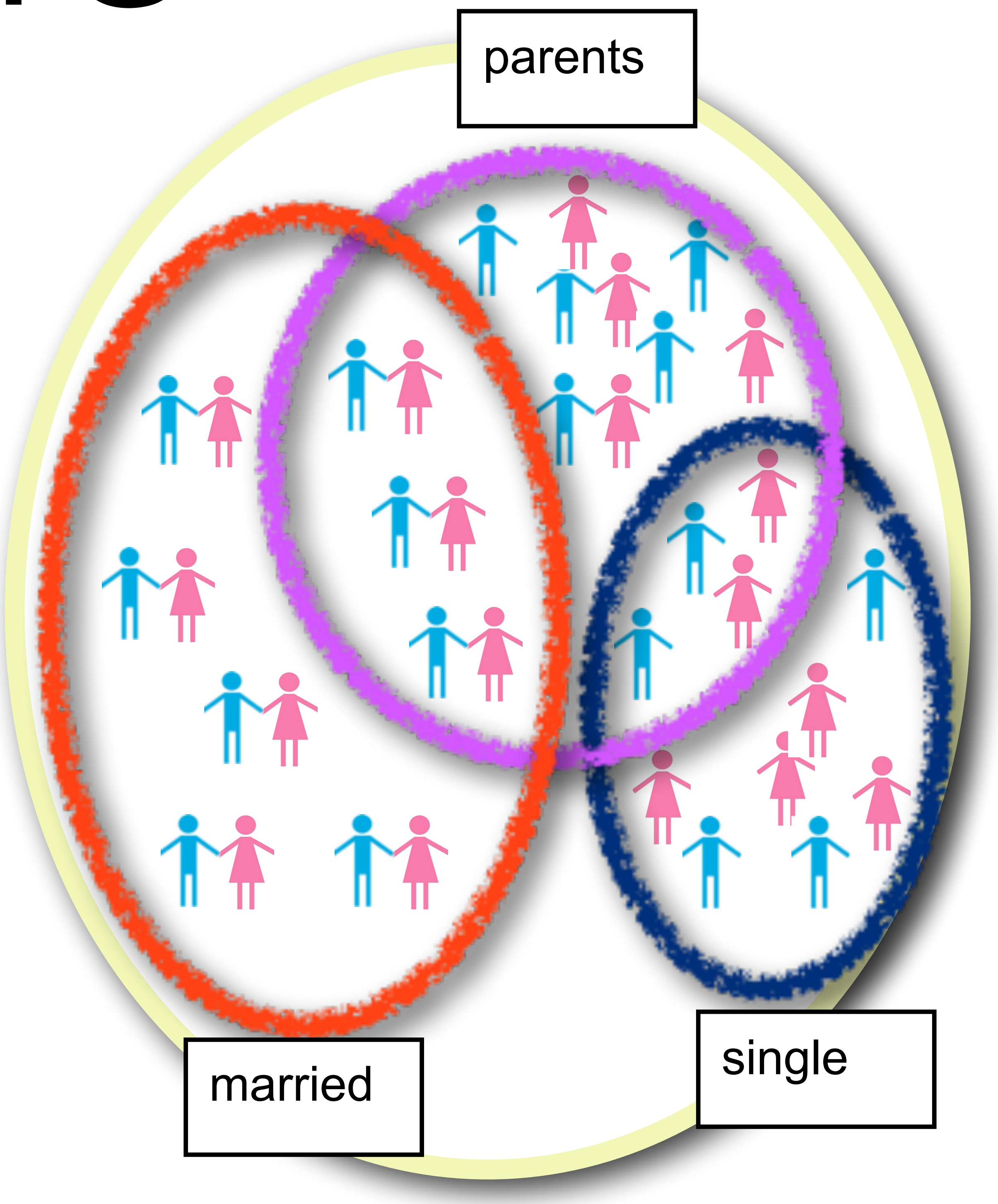
# owl:Thing and owl:Nothing

- OWL has two predefined classes
  - `owl:Thing`
    - $\top$ (in DL formalism)
    - class containing all individuals
  - `owl:Nothing`
    - $\bot$ (in DL formalism)
    - “empty” class containing no individuals

- For every class C
  - `owl:Nothing` is a subclass of C
  - C is a subclass of `owl:Thing`

# OWL class constructors

- Boolean combinations are used to define classes
  - married and single are disjoint but parent and single are not!




```
Class: Married
   EquivalentTo: not Single
```

*Manchester syntax*

```
:married  owl:subClassOf  [
   rdf:type              owl:Class ;
   owl:intersectionOf  (
                          [ rdf:type           owl:Class ;
                            owl:complementOf   :single ] )
 ] .
```

*turtle syntax*

```
<owl:Class rdf:about="#married">
   <owl:subClassOf>
      <owl:Class>
         <owl:complementOf rdf:resource="#single"/>
      </owl:Class>
   </owl:subClassOf>
</owl:Class>
```

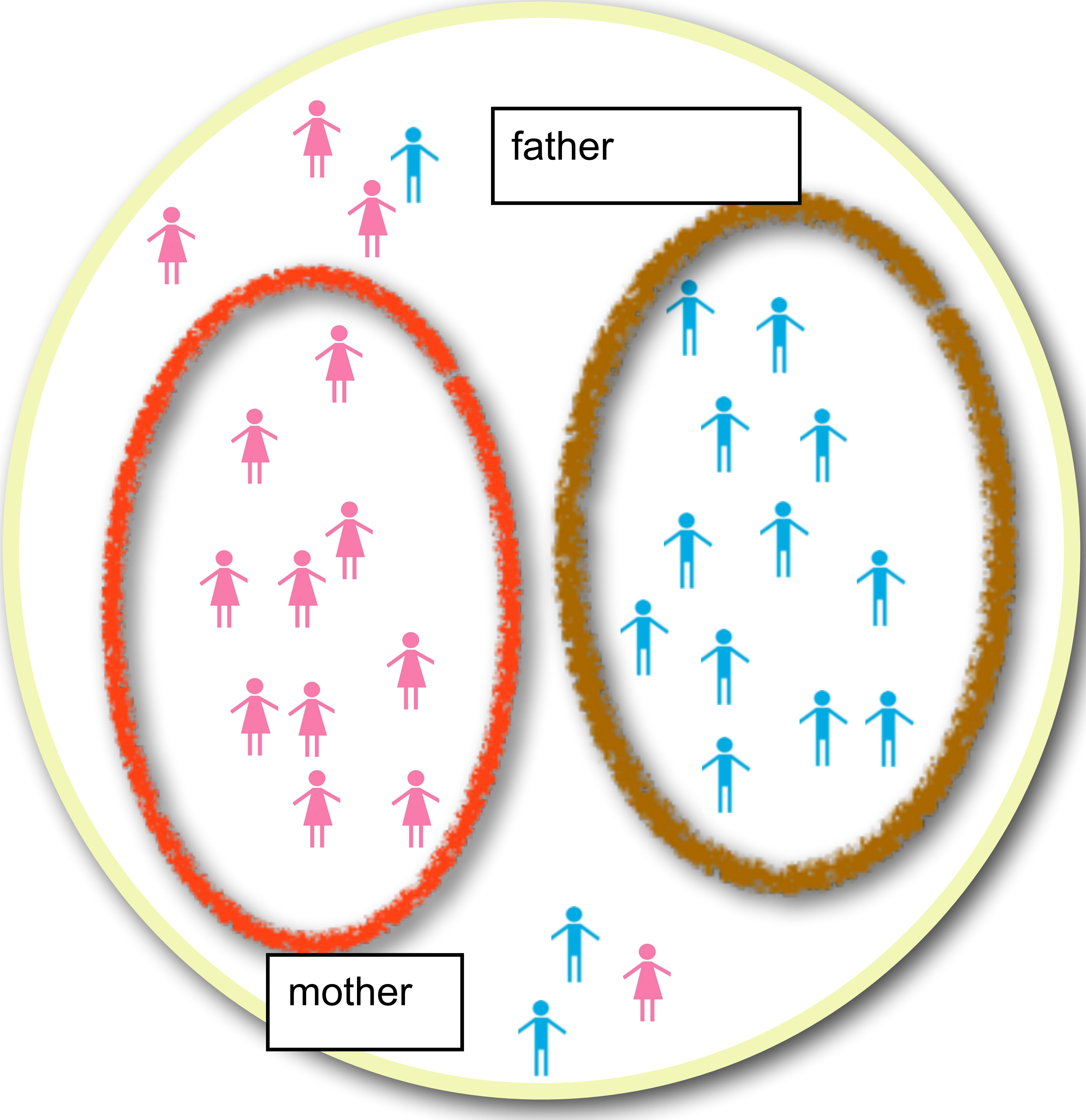
# OWL class constructors

- parents are mothers and father

```
<owl:Class rdf:about="Parent">
   <owl:equivalentClass>
     <owl:Class>
       <owl:unionOf rdf:parseType="Collection">
         <owl:Class rdf:about="Mother"/>
         <owl:Class rdf:about="Father"/>
       </owl:unionOf>
     </owl:Class>
   </owl:equivalentClass>
 </owl:Class>
```

```
Class: Parent
   EquivalentTo: Mother or Father
```

```
:Parent  owl:equivalentClass  [
   rdf:type     owl:Class ;
   owl:unionOf  ( :Mother :Father )
 ] .
```



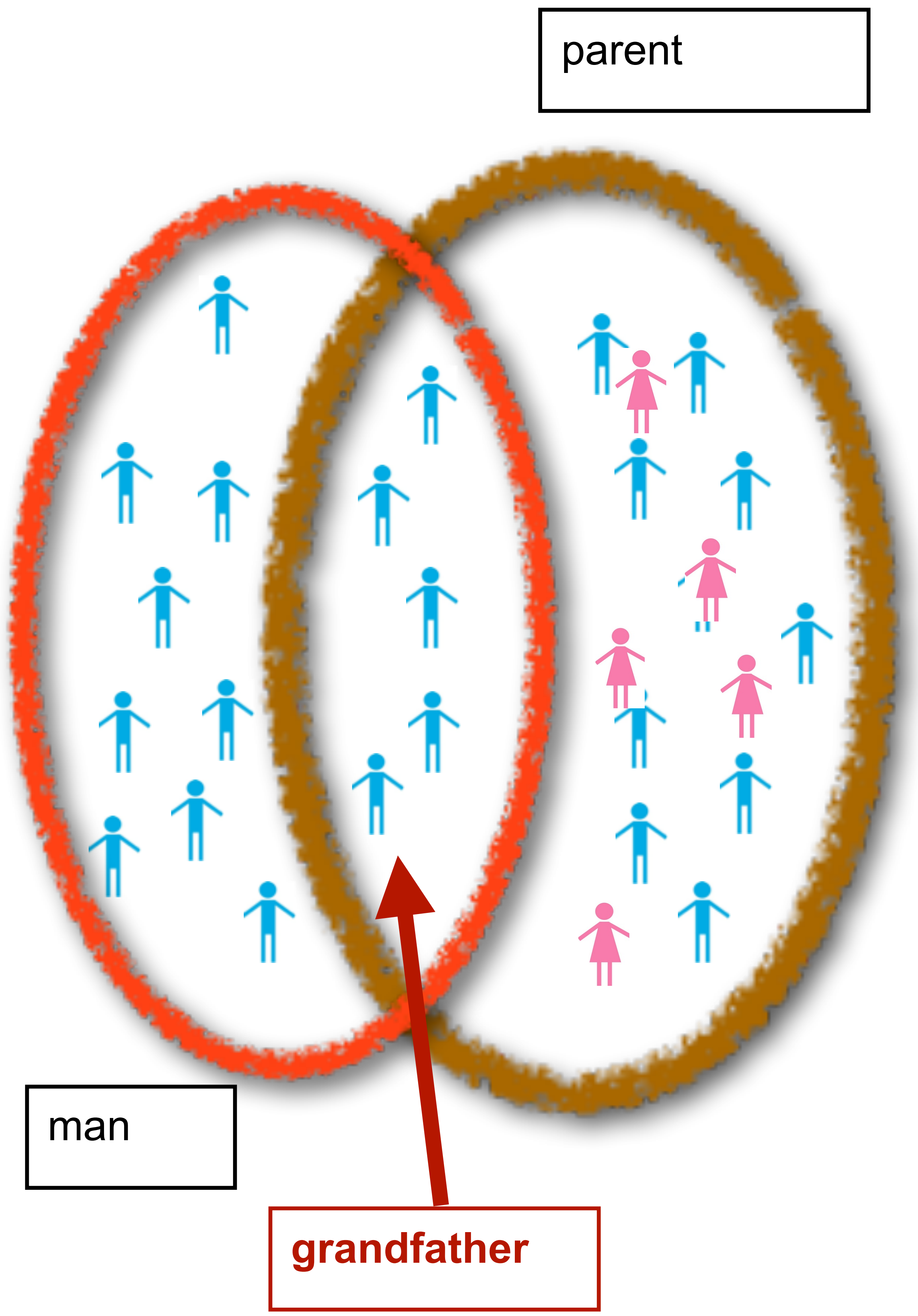
# OWL class constructors

- Grandfather is both a man and a father

```
<owl:Class rdf:about="Grandfather">
   <rdfs:subClassOf>
     <owl:Class>
       <owl:intersectionOf rdf:parseType="Collection">
         <owl:Class rdf:about="Man"/>
         <owl:Class rdf:about="Parent"/>
       </owl:intersectionOf>
     </owl:Class>
   </rdfs:subClassOf>
 </owl:Class>
```

```
  Class: Grandfather
    SubClassOf: Man and Parent
```

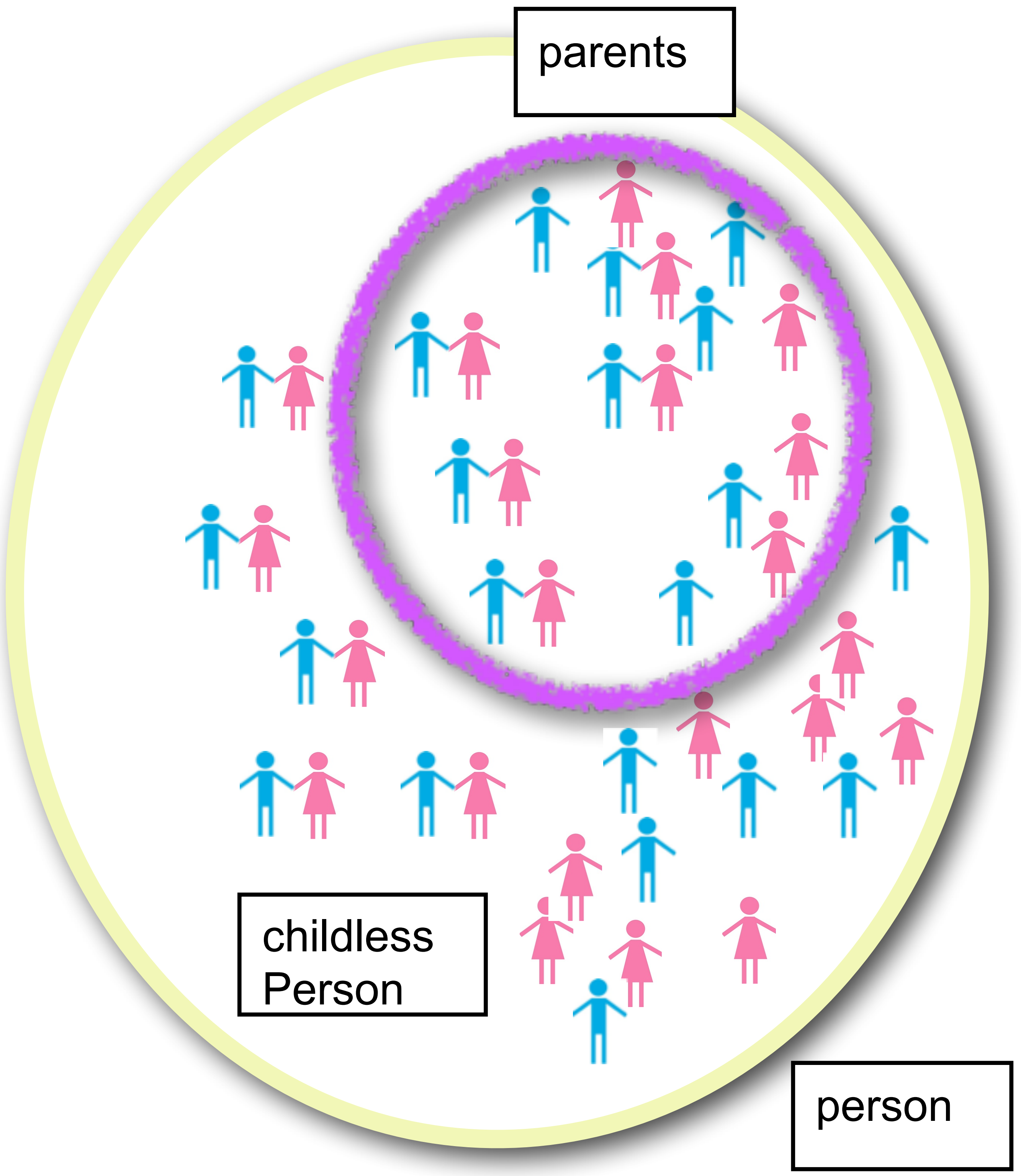
```
:Grandfather  rdfs:subClassOf  [
   rdf:type              owl:Class ;
   owl:intersectionOf  ( :Man  :Parent )
 ] .
```



# OWL class constructors

- a childless person is someone who is a person but not a parent

```
<owl:Class rdf:about="ChildlessPerson">
   <owl:equivalentClass>
     <owl:Class>
       <owl:intersectionOf rdf:parseType="Collection">
         <owl:Class rdf:about="Person"/>
         <owl:Class>
           <owl:complementOf rdf:resource="Parent"/>
         </owl:Class>
       </owl:intersectionOf>
     </owl:Class>
   </owl:equivalentClass>
 </owl:Class>
```




```
:ChildlessPerson  owl:equivalentClass  [
   rdf:type             owl:Class ;
   owl:intersectionOf  ( :Person
                          [ rdf:type          owl:Class ;
                            owl:complementOf  :Parent ] )
 ] .
```

```
Class: ChildlessPerson
  EquivalentTo: Person and not Parent
```

V.Tamma

# OWL class constructors

- Classes can also be defined through enumeration using `owl:oneOf`
  - allows a class to be defined extensionally,
    - with exactly the enumerated individual

```
<owl:Class rdf:about=”#simpsonFamily”>
   <owl:oneOf rdf:parseType=”Collection”>
      <owl:Thing rdf:about=“#marge”/>
      <owl:Thing rdf:about=“#homer”/>
      <owl:Thing rdf:about=“#lisa”/>
      <owl:Thing rdf:about=“#maggie”/>
      <owl:Thing rdf:about=“#bart”/>
   </owl:oneOf>
</owl:Class>
```

```
:simpsonFamily  owl:equivalentClass  [
   rdf:type   owl:Class ;
   owl:oneOf  ( :marge, :homer, :lisa, :maggie, ;bart)
 ] .
```

```
Class: simpsonFamily
   EquivalentTo: { marge, homer, lisa, maggie, bart }
```



V.Tamma

# Recap

- OWL preliminaries
- OWL class constructors

- `https://www.w3.org/TR/owl2-primer/`