OWL, SPARQL, and Mappings

John Beverley

Assistant Professor, *University at Buffalo*
Co-Director, National Center for Ontological Research
Affiliate Faculty, *Institute of Artificial Intelligence and Data Science*

Outline

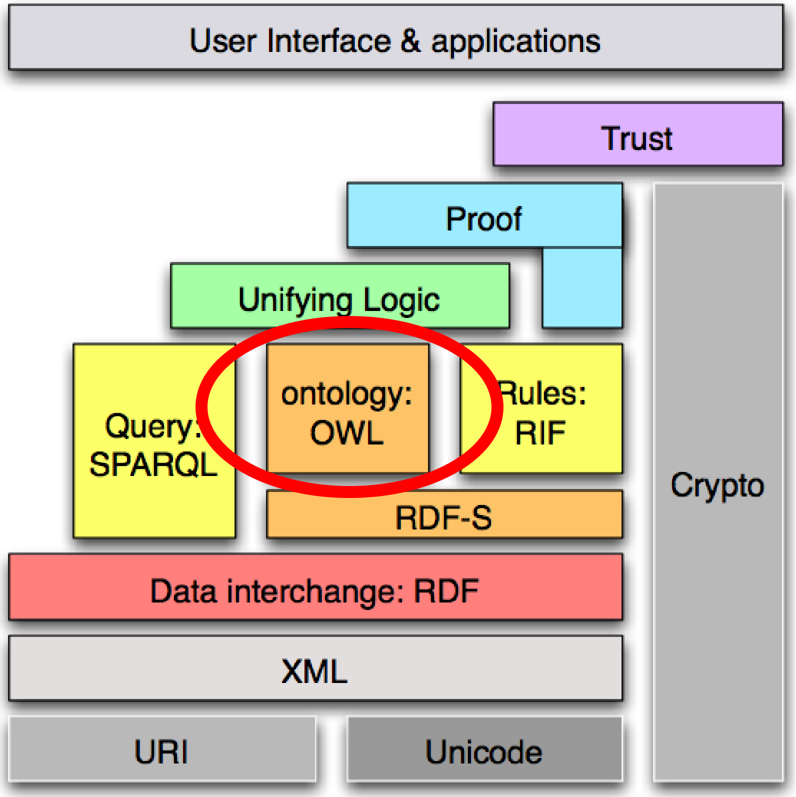
- OWL 2 Direct Semantics
- SPARQL
- Mapping

Outline

- OWL 2 Direct Semantics
- SPARQL
- Mapping

Semantic Web Stack

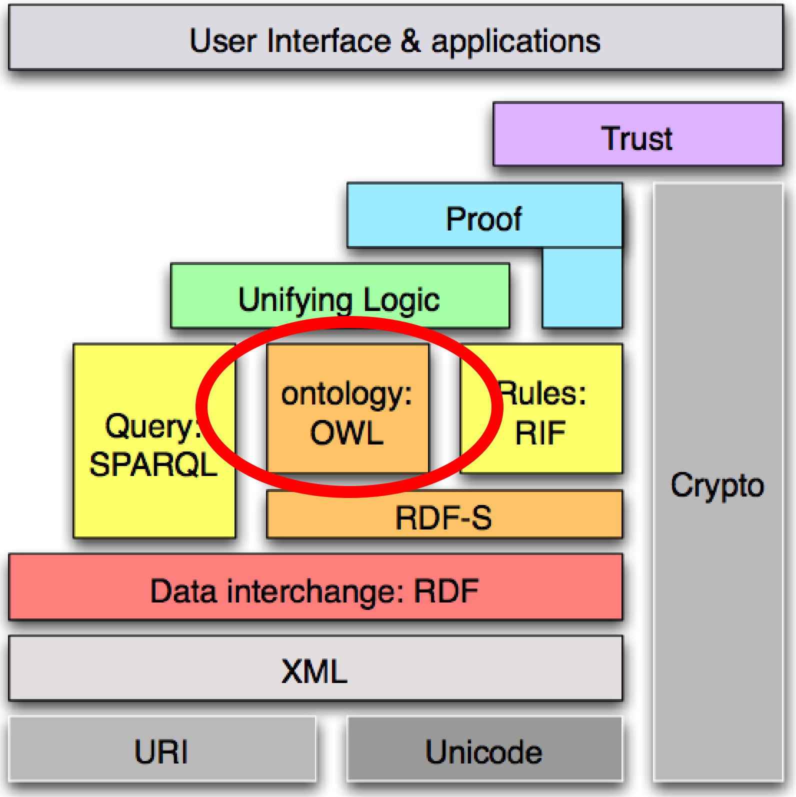
- "OWL" stands for:
  Web
  Ontology
  Language



Semantic Web Stack

- "OWL" stands for:
  Web
  Ontology
  Language

- OWL is:
  A *family of vocabularies*
  That extend *RDF and RDFS*
  Which provide semantics for constructing *logical relationships among resources*



OWL2

- We will focus on the most widely used OWL language, called “OWL2”
- OWL2 has a clear connection to description logics
- The vocabulary includes terms for representing contradictions, logical truths, disjunction, conjunction etc. but also complex combinations of logic such as disjointness

OWL2

- We will focus on the most widely used OWL language, called “OWL2”
- OWL2 has a clear connection to description logics
- The vocabulary includes terms for representing contradictions, logical truths, disjunction, conjunction etc. but also complex combinations of logic such as disjointness

```
owl:unionOf
    a rdf:Property ;
    rdfs:comment "The property that determines the collection of classes or data ranges that build a union." ;
    rdfs:domain rdfs:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "unionOf" ;
    rdfs:range rdf:List .
```

```
owl:disjointUnionOf
    a rdf:Property ;
    rdfs:comment "The property that determines that a given class is equivalent to the disjoint union of a collection of other classes." ;
    rdfs:domain owl:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "disjointUnionOf" ;
    rdfs:range rdf:List .

owl:disjointWith
    a rdf:Property ;
    rdfs:comment "The property that determines that two given classes are disjoint." ;
    rdfs:domain owl:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "disjointWith" ;
    rdfs:range owl:Class .
```

```
owl:Nothing
    a owl:Class ;
    rdfs:comment "This is the empty class." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "Nothing" ;
    rdfs:subClassOf owl:Thing .
```

```
owl:topObjectProperty
    a owl:ObjectProperty ;
    rdfs:comment "The object property that relates every two individuals." ;
    rdfs:domain owl:Thing ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "topObjectProperty" ;
    rdfs:range owl:Thing .
```

```
owl:unionOf
    a rdf:Property ;
    rdfs:comment "The property that determines the collection of classes or data ranges that build a union." ;
    rdfs:domain rdfs:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "unionOf" ;
    rdfs:range rdf:List .
```

```
owl:disjointUnionOf
    a rdf:Property ;
    rdfs:comment "The property that determines that a given class is equivalent to the disjoint union of a collection of other classes." ;
    rdfs:domain owl:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "disjointUnionOf" ;
    rdfs:range rdf:List .

owl:disjointWith
    a rdf:Property ;
    rdfs:comment "The property that determines that two given classes are disjoint." ;
    rdfs:domain owl:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "disjointWith" ;
    rdfs:range owl:Class .
```

```
owl:Nothing
    a owl:Class ;
    rdfs:comment "This is the empty class." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "Nothing" ;
    rdfs:subClassOf owl:Thing .
```

```
owl:topObjectProperty
    a owl:ObjectProperty ;
    rdfs:comment "The object property that relates every two individuals." ;
    rdfs:domain owl:Thing ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "topObjectProperty" ;
    rdfs:range owl:Thing .
```

OWL2

- We will focus on the most widely used OWL language, called "OWL2"
- OWL2 has a clear connection to description logics
- The vocabulary includes terms for representing contradictions, logical truths, disjunction, conjunction etc. but also complex combinations of logic such as disjointness

```
owl:unionOf
    a rdf:Property ;
    rdfs:comment "The property that determines the collection of classes or data ranges that build a union." ;
    rdfs:domain rdfs:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "unionOf" ;
    rdfs:range rdf:List .
```

```
owl:disjointUnionOf
    a rdf:Property ;
    rdfs:comment "The property that determines that a given class is equivalent to the disjoint union of a collection of other classes." ;
    rdfs:domain owl:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "disjointUnionOf" ;
    rdfs:range rdf:List .

owl:disjointWith
    a rdf:Property ;
    rdfs:comment "The property that determines that two given classes are disjoint." ;
    rdfs:domain owl:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "disjointWith" ;
    rdfs:range owl:Class .
```

```
owl:Nothing
    a owl:Class ;
    rdfs:comment "This is the empty class." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "Nothing" ;
    rdfs:subClassOf owl:Thing .
```

```
owl:topObjectProperty
    a owl:ObjectProperty ;
    rdfs:comment "The object property that relates every two individuals." ;
    rdfs:domain owl:Thing ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "topObjectProperty" ;
    rdfs:range owl:Thing .
```

OWL2

- We will focus on the most widely used OWL language, called "OWL2"
- OWL2 has a clear connection to description logics
- The vocabulary includes terms for representing contradictions, logical truths, disjunction, conjunction etc. but also complex combinations of logic such as disjointness

```
owl:unionOf
    a rdf:Property ;
    rdfs:comment "The property that determines the collection of classes or data ranges that build a union." ;
    rdfs:domain rdfs:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "unionOf" ;
    rdfs:range rdf:List .
```

```
owl:disjointUnionOf
    a rdf:Property ;
    rdfs:comment "The property that determines that a given class is equivalent to the disjoint union of a collection of other classes." ;
    rdfs:domain owl:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "disjointUnionOf" ;
    rdfs:range rdf:List .

owl:disjointWith
    a rdf:Property ;
    rdfs:comment "The property that determines that two given classes are disjoint." ;
    rdfs:domain owl:Class ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "disjointWith" ;
    rdfs:range owl:Class .
```

```
owl:Nothing
    a owl:Class ;
    rdfs:comment "This is the empty class." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "Nothing" ;
    rdfs:subClassOf owl:Thing .
```

```
owl:topObjectProperty
    a owl:ObjectProperty ;
    rdfs:comment "The object property that relates every two individuals." ;
    rdfs:domain owl:Thing ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "topObjectProperty" ;
    rdfs:range owl:Thing .
```

OWL2

- We will focus on the most widely used OWL language, called "OWL2"
- OWL2 has a clear connection to description logics
- The vocabulary includes terms for representing contradictions, logical truths, disjunction, conjunction etc. but also complex combinations of logic such as disjointness
- Also: identity, existentials, properties of relations such as functional and inverse functional, instance declarations and negative property assertions

OWL2

- We will focus on the most widely used OWL language, called "OWL2"
- OWL2 has a clear connection to description logics
- The vocabulary includes terms for representing contradictions, logical truths, disjunction, conjunction etc. but also complex combinations of logic such as disjointness
- Also: identity, existentials, properties of relations such as functional and inverse functional, instance declarations and negative property assertions

```
owl:sameAs
    a rdf:Property ;
    rdfs:comment "The property that determines that two given individuals are equal." ;
    rdfs:domain owl:Thing ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "sameAs" ;
    rdfs:range owl:Thing .

owl:someValuesFrom
    a rdf:Property ;
    rdfs:comment "The property that determines the class that an existential property restriction refers to." ;
    rdfs:domain owl:Restriction ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "someValuesFrom" ;
    rdfs:range rdfs:Class .
```

```
owl:FunctionalProperty
    a rdfs:Class ;
    rdfs:comment "The class of functional properties." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "FunctionalProperty" ;
    rdfs:subClassOf rdf:Property .

owl:InverseFunctionalProperty
    a rdfs:Class ;
    rdfs:comment "The class of inverse-functional properties." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "InverseFunctionalProperty" ;
    rdfs:subClassOf owl:ObjectProperty .
```

```
owl:NamedIndividual
    a rdfs:Class ;
    rdfs:comment "The class of named individuals." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "NamedIndividual" ;
    rdfs:subClassOf owl:Thing .

owl:NegativePropertyAssertion
    a rdfs:Class ;
    rdfs:comment "The class of negative property assertions." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "NegativePropertyAssertion" ;
    rdfs:subClassOf rdfs:Resource .
```

OWL2

- We will focus on the most widely used OWL language, called "OWL2"
- OWL2 has a clear connection to description logics
- The vocabulary includes terms for representing contradictions, logical truths, disjunction, conjunction etc. but also complex combinations of logic such as disjointness
- Also: identity, existentials, properties of relations such as functional and inverse functional, instance declarations and negative property assertions

```
owl:sameAs
    a rdf:Property ;
    rdfs:comment "The property that determines that two given individuals are equal." ;
    rdfs:domain owl:Thing ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "sameAs" ;
    rdfs:range owl:Thing .

owl:someValuesFrom
    a rdf:Property ;
    rdfs:comment "The property that determines the class that an existential property restriction refers to." ;
    rdfs:domain owl:Restriction ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "someValuesFrom" ;
    rdfs:range rdfs:Class .
```

```
owl:FunctionalProperty
    a rdfs:Class ;
    rdfs:comment "The class of functional properties." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "FunctionalProperty" ;
    rdfs:subClassOf rdf:Property .

owl:InverseFunctionalProperty
    a rdfs:Class ;
    rdfs:comment "The class of inverse-functional properties." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "InverseFunctionalProperty" ;
    rdfs:subClassOf owl:ObjectProperty .
```

```
owl:NamedIndividual
    a rdfs:Class ;
    rdfs:comment "The class of named individuals." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "NamedIndividual" ;
    rdfs:subClassOf owl:Thing .

owl:NegativePropertyAssertion
    a rdfs:Class ;
    rdfs:comment "The class of negative property assertions." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "NegativePropertyAssertion" ;
    rdfs:subClassOf rdfs:Resource .
```

OWL2

- We will focus on the most widely used OWL language, called "OWL2"
- OWL2 has a clear connection to description logics
- The vocabulary includes terms for representing contradictions, logical truths, disjunction, conjunction etc. but also complex combinations of logic such as disjointness
- Also: identity, existentials, properties of relations such as functional and inverse functional, instance declarations and negative property assertions

```
owl:sameAs
    a rdf:Property ;
    rdfs:comment "The property that determines that two given individuals are equal." ;
    rdfs:domain owl:Thing ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "sameAs" ;
    rdfs:range owl:Thing .

owl:someValuesFrom
    a rdf:Property ;
    rdfs:comment "The property that determines the class that an existential property restriction refers to." ;
    rdfs:domain owl:Restriction ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "someValuesFrom" ;
    rdfs:range rdfs:Class .
```

```
owl:FunctionalProperty
    a rdfs:Class ;
    rdfs:comment "The class of functional properties." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "FunctionalProperty" ;
    rdfs:subClassOf rdf:Property .

owl:InverseFunctionalProperty
    a rdfs:Class ;
    rdfs:comment "The class of inverse-functional properties." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "InverseFunctionalProperty" ;
    rdfs:subClassOf owl:ObjectProperty .
```

```
owl:NamedIndividual
    a rdfs:Class ;
    rdfs:comment "The class of named individuals." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "NamedIndividual" ;
    rdfs:subClassOf owl:Thing .

owl:NegativePropertyAssertion
    a rdfs:Class ;
    rdfs:comment "The class of negative property assertions." ;
    rdfs:isDefinedBy <http://www.w3.org/2002/07/owl#> ;
    rdfs:label "NegativePropertyAssertion" ;
    rdfs:subClassOf rdfs:Resource .
```

OWL2

- We will focus on the most widely used OWL language, called "OWL2"
- OWL2 comes in two 'flavors' of differing logical strengths:
    - OWL2 Full
    - OWL2 DL: Direct Semantics

OWL2

- We will focus on the most widely used OWL language, called “OWL2”

- OWL2 comes in two ‘flavors’ of differing logical strengths:

    - OWL2 Full
    - OWL2 DL: Direct Semantics

OWL2 Full

- OWL2 was designed for reasoning support, hence the focus on representing logical connections among resources

- However, combining the RDFS and OWL2 vocabularies without restriction, results in a language too expressive to accommodate efficient reasoning support

- *OWL2 Full* uses all the OWL2 vocabulary and allows arbitrary combinations of this vocabulary with RDF and RDFS

OWL2 Full

- This undermines efficient reasoning in part because arbitrary combinations of OWL2 and RDF **allow one to change the meaning of predefined RDF or OWL2** vocabulary items:

- For example, in **OWL2** Full you can:

    - Impose a cardinality constraint on the class of all classes, thereby limiting the number of classes that can be represented in an ontology
    - Relate any instance of a class directly to another class

OWL2

- We will focus on the most widely used OWL language, called "OWL2"
- OWL2 has a clear connection to description logics
- OWL2 comes in two 'flavors' of differing logical strengths:
  - OWL2 Full
  - OWL2 DL: Direct Semantics

OWL2 DL Direct Semantics

- *OWL2 DL* is a restriction of the OWL2 vocabulary by mapping it directly to a decidable description logic

OWL2 DL Direct Semantics

- *OWL2 DL* is a restriction of the OWL2 vocabulary by mapping it directly to a **decidable description logic**

- Notable consequences of this restriction include:

  - OWL2 vocabulary terms **cannot** be applied to each other
  - All OWL2 classes are **instances of owl:Class** rather than rdfs:Class
  - OWL2 properties are either owl:ObjectProperty or owl:DatatypeProperty but not both
  - OWL2 resources **cannot simultaneously** be class, property, and instance

Summary

- OWL2 DL **can be used** by standard reasoners one finds in Protege, but OWL2 Full **cannot**

- Important takeaways thus far -

    - Simple RDF will often need to be extended to be useful
    - Extended RDF will need to be restricted to be usable
    - Any legal OWL2 DL file is a legal RDF file
    - It is not the case any legal OWL2 file is a legal RDF file

OWL2 DL and SROIQ

- OWL2 DL corresponds to the description logic SROIQ

- SROIQ:
  - S = ALC
  - R = Role Axiom Extension
  - O = Nominals
  - I = Inverses
  - Q = Qualified Cardinalities

OWL2 DL and SROIQ

- OWL2 DL corresponds to the description logic SROIQ

- SROIQ:
  - S = ALC
  - R = Role Axiom Extension
  - O = Nominals
  - I = Inverses
  - Q = Qualified Cardinalities

ALC Syntax

Definition 2.1. Let $\mathbf{C}$ be a set of *concept names* and $\mathbf{R}$ be a set of *role names* disjoint from $\mathbf{C}$. The set of $\mathcal{ALC}$ *concept descriptions* over $\mathbf{C}$ and $\mathbf{R}$ is inductively defined as follows:

- Every concept name is an $\mathcal{ALC}$ concept description.
- $\top$ and $\bot$ are $\mathcal{ALC}$ concept descriptions.
- If C and D are $\mathcal{ALC}$ concept descriptions and r is a role name, then the following are also $\mathcal{ALC}$ concept descriptions:

  $C \sqcap D$ (conjunction),

  $C \sqcup D$ (disjunction),

  $\neg C$ (negation),

  $\exists r.C$, (existential restriction), and

  $\forall r.C$ (value restriction).

ALC Semantics

Definition 2.2. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$, called the *interpretation domain*, and a mapping $\cdot^{\mathcal{I}}$ that maps

- every concept name $A \in \mathbf{C}$ to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and
- every role name $r \in \mathbf{R}$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

$$
\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}}, \\
\bot^{\mathcal{I}} &= \emptyset, \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}}, \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}}, \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}, \\
(\exists r.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \text{there is an } e \in \Delta^{\mathcal{I}} \text{ with } (d,e) \in r^{\mathcal{I}} \text{ and } e \in C^{\mathcal{I}}\}, \\
(\forall r.C)^{\mathcal{I}} &= \{d \in \Delta^{\mathcal{I}} \mid \text{for all } e \in \Delta^{\mathcal{I}},\ \text{if } (d,e) \in r^{\mathcal{I}},\ \text{then } e \in C^{\mathcal{I}}\}.
\end{aligned}
$$

OWL2 DL and SROIQ

- OWL2 DL corresponds to the description logic SROIQ

- SROIQ:
  - S = ALC
  - R = Role Axiom Extension
  - O = Nominals
  - I = Inverses
  - Q = Qualified Cardinalities

ALCO (nominal) Syntax/Semantics

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"
- **Note**
  - Nominals allow for defining classes by enumerations of instances

ALCO (nominal) Syntax/Semantics

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"
- **Note**
  - Nominals allow for defining classes by enumerations of instances

    The Beatles consist of john, paul, ringo, and george

ALCO (nominal) Syntax/Semantics

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"

- **Note**
  - Nominals allow for defining classes by enumerations of instances

    The Beatles consist of john, paul, ringo, and george

  - In ALC, the connective $\sqcup$ can be used to combine *classes*

ALCO (nominal) Syntax/Semantics

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"
- **Note**
  - Nominals allow for defining classes by enumerations of instances

    The Beatles consist of john, paul, ringo, and george
  - In ALC, the connective ⊔ can be used to combine *classes*

    Great Bands = The Beatles ⊔ The Eagles ⊔ Metallica ⊔…

ALCO (nominal) Syntax/Semantics

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"
- **Note**
  - Nominals allow for defining classes by enumerations of instances

    The Beatles consist of john, paul, ringo, and george
  - In ALC, the connective ⊔ can be used to combine *classes*

    Great Bands = The Beatles ⊔ The Eagles ⊔ Metallica ⊔…
  - But no native connectives link *instances*

ALCO (nominal) Syntax/Semantics

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"

- **Note**
  - Nominals allow for defining classes by enumerations of instances

    The Beatles consist of john, paul, ringo, and george

  - In ALC, the connective ⊔ can be used to combine *classes*

    Great Bands = The Beatles ⊔ The Eagles ⊔ Metallica ⊔…

  - But no native connectives link *instances*

    Beatles = john ? paul ? ringo ? george

ALCO (nominal) Syntax/Semantics

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"
- **Note**
  - Nominals allow for defining classes by enumerations of instances

    The Beatles consist of john, paul, ringo, and george
  - In ALCO, treating instances as singleton classes permits using ⊔

ALCO (nominal) Syntax/Semantics

ALCO Signature = ALC Signature + {{a}, {b}, …}

- {a} – Corresponds to the instance mapped to by the name "a"

- **Note**
  - Nominals allow for defining classes by enumerations of instances

    The Beatles consist of john, paul, ringo, and george
  - In ALCO, treating instances as singleton classes permits using $\sqcup$

    The Beatles = {john} $\sqcup$ {paul} $\sqcup$ {ringo} $\sqcup$ {george}

ALCO (nominal) Syntax/Semantics

ALCO Signature = ALC Signature + { {a}, {b}, … }

- {a} – Corresponds to the instance mapped to by the name "a"
- **Note**
  - Nominals allow for defining classes by enumerations of instances

    The Beatles consist of john, paul, ringo, and george
  - In ALCO, treating instances as singleton classes permits using ⊔

    The Beatles = {john} ⊔ {paul} ⊔ {ringo} ⊔ {george}
  - Since ⊔ can only be used to combine classes

OWL2 DL and SROIQ

- OWL2 DL corresponds to the description logic SROIQ

- SROIQ:
  - S = ALC
  - R = Role Axiom Extension
  - O = Nominals
  - I = Inverses
  - Q = Qualified Cardinalities

ALCI (inverses) Syntax/Semantics

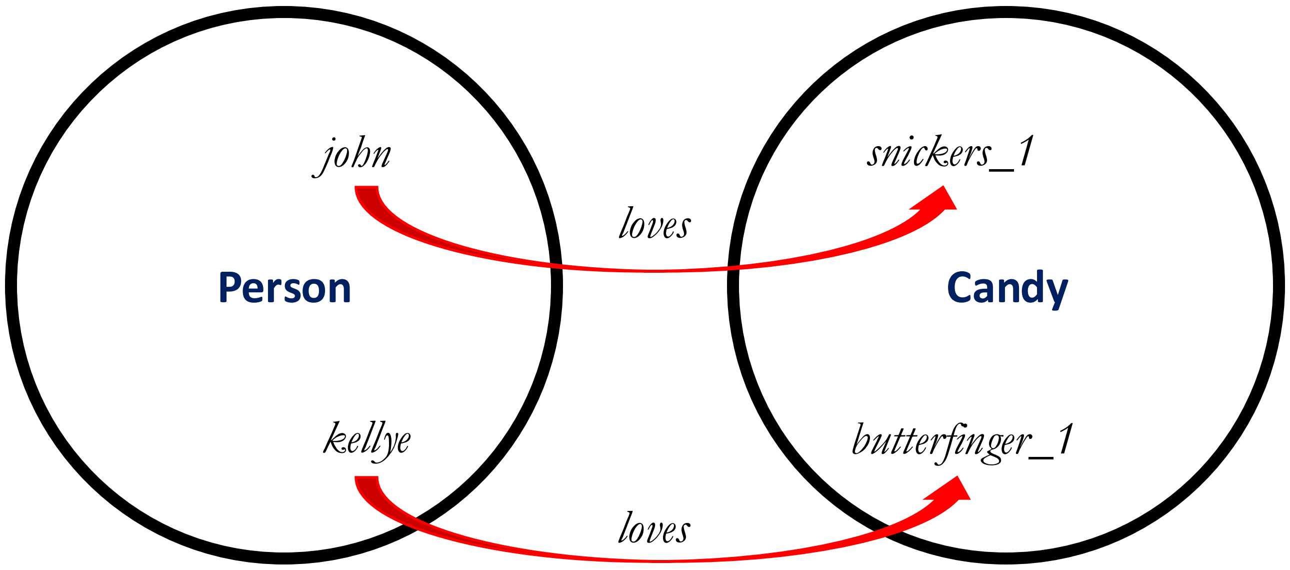
ALCI Signature = ALC Signature + $\{r^{-}_{1\ldots n}\}$

- $r^{-}_{1\ldots n}$ – Corresponds to inversions of relations such as r between instances, such as the inverse of 'loves' being 'loves^{-}', i.e. 'loved by'

ALCI (inverses) Syntax/Semantics

ALCI Signature = ALC Signature + $\{r^{-}_{1\ldots n}\}$

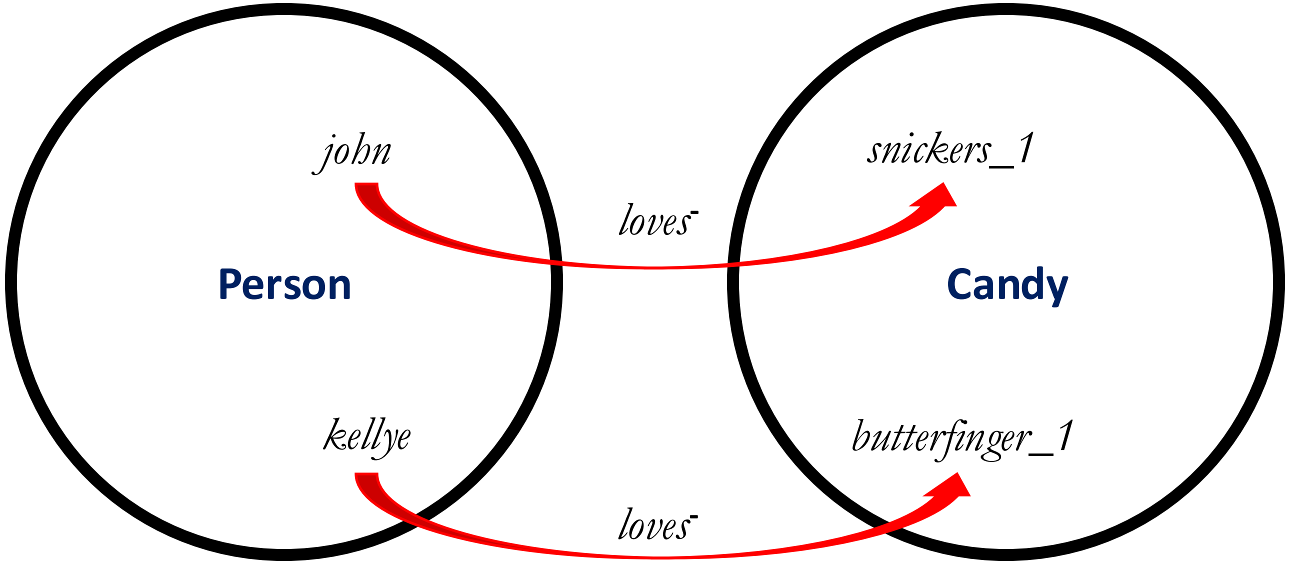
- $r^{-}_{1\ldots n}$ – Corresponds to inversions of relations such as r between instances, such as the inverse of 'loves' being 'loves⁻', i.e. 'loved by'



ALCI (inverses) Syntax/Semantics

ALCI Signature = ALC Signature + $\{r^{-}_{1\ldots n}\}$

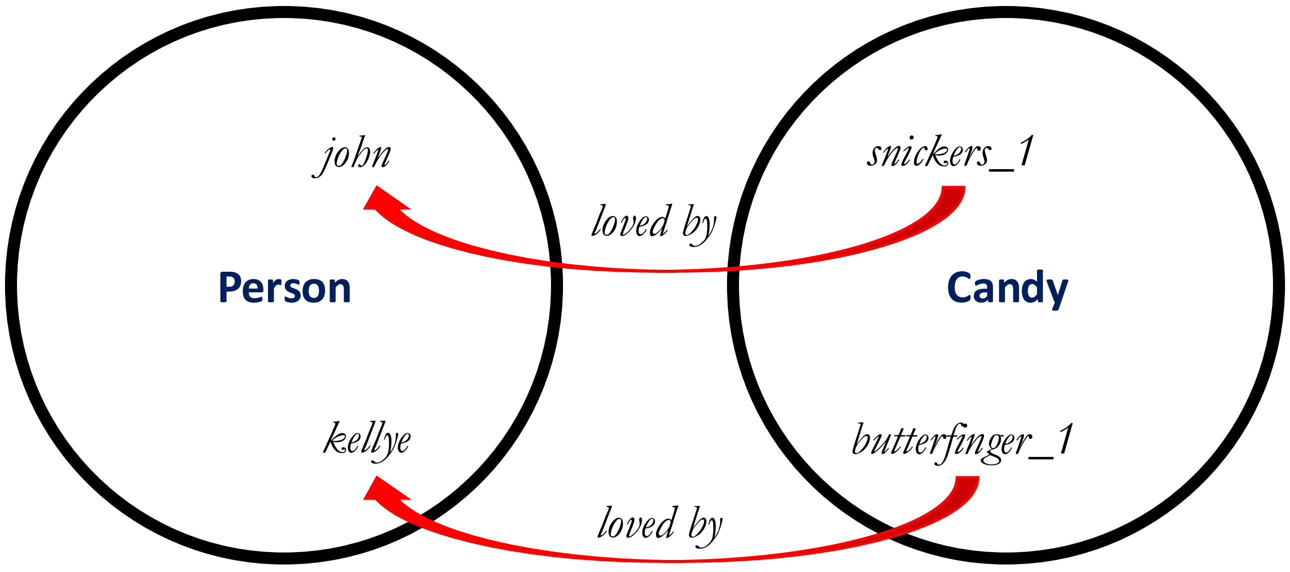
- $r^{-}_{1\ldots n}$ – Corresponds to inversions of relations such as r between instances, such as the inverse of 'loves' being 'loves^{-}', i.e. 'loved by'



ALCI (inverses) Syntax/Semantics

ALCI Signature = ALC Signature + $\{r^{-}_{1\ldots n}\}$

- $r^{-}_{1\ldots n}$ – Corresponds to inversions of relations such as r between instances, such as the inverse of 'loves' being 'loves^{-}', i.e. 'loved by'



OWL2 DL and SROIQ

- OWL2 DL corresponds to the description logic SROIQ

- SROIQ:
  - S = ALC
  - R = Role Axiom Extension
  - O = Nominals
  - I = Inverses
  - Q = Qualified Cardinalities

ALCQ (qual. cardinality) Syntax/Semantics

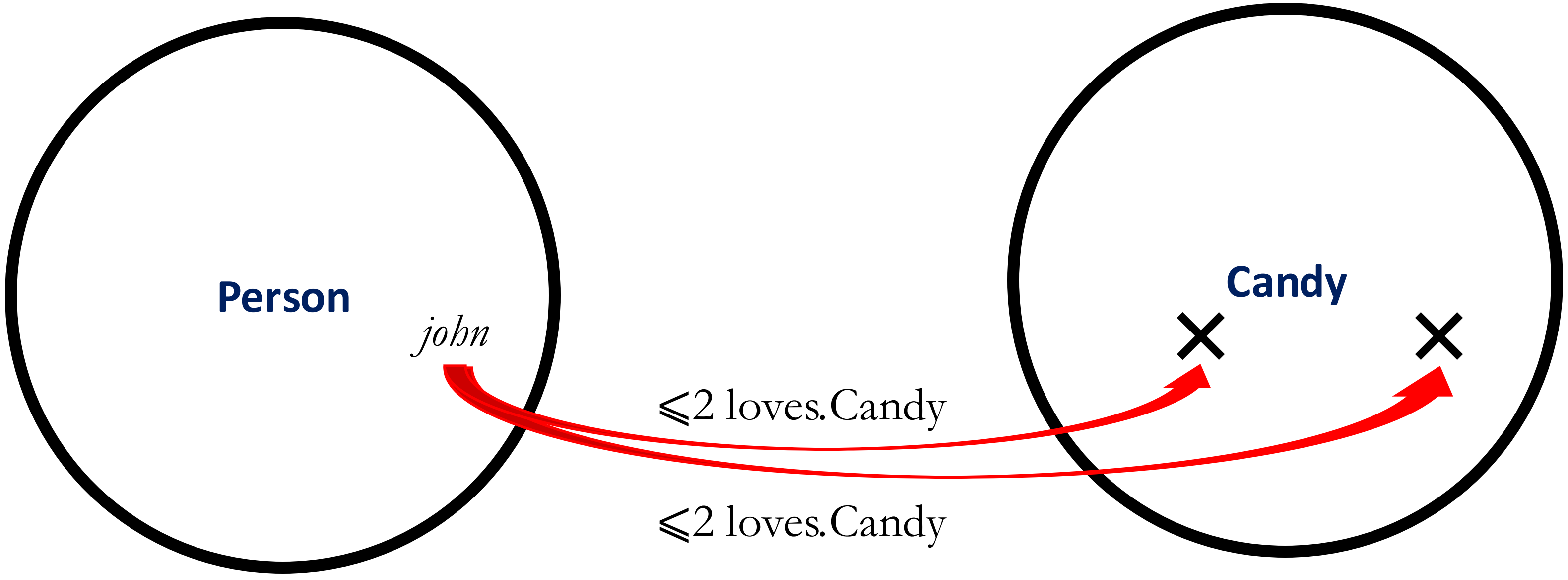
ALCQ Signature = ALC Signature + {$\leqslant$n r.C, $\geqslant$n r.C}

- $\leqslant$n r.C – Corresponds to restriction that r is related to no more than n Cs
- $\geqslant$n r.C – Corresponds to restriction that r is related to no fewer than n Cs

ALCQ (qual. cardinality) Syntax/Semantics

ALCQ Signature = ALC Signature + {⩽n r.C, ⩾n r.C}

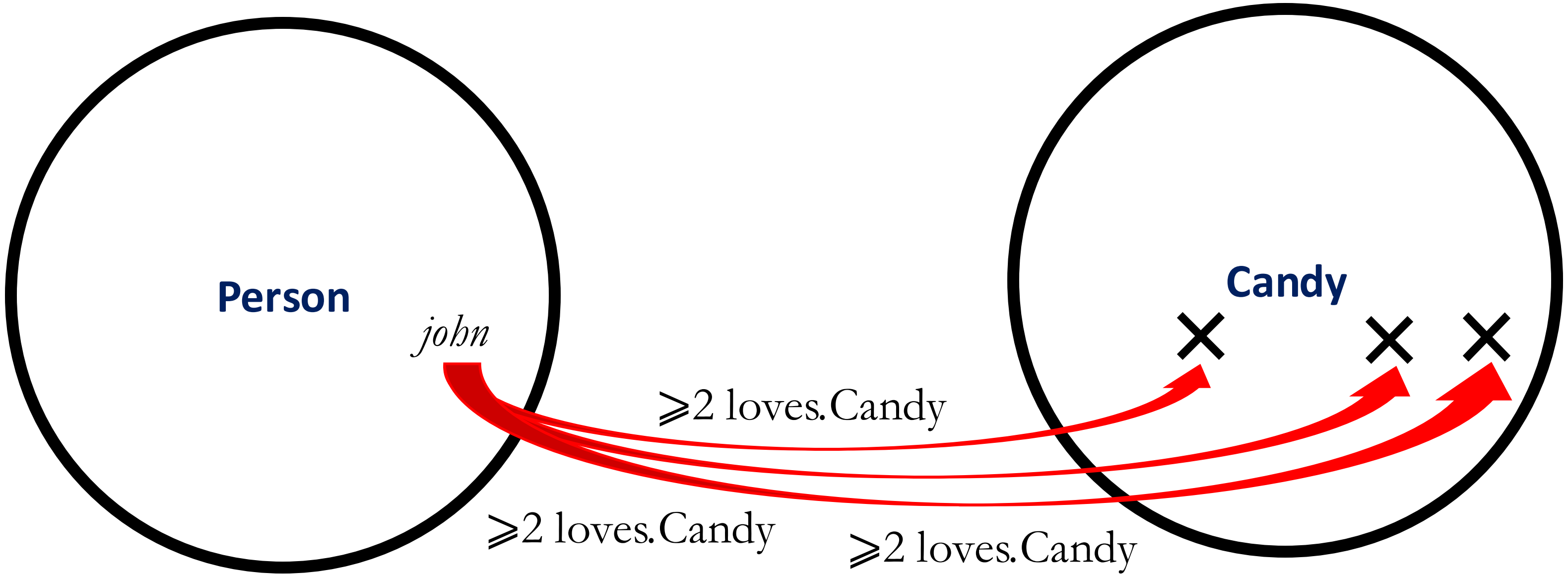
- ⩽n r.C – Corresponds to restriction that r is related to no more than n Cs
- ⩾n r.C – Corresponds to restriction that r is related to no fewer than n Cs



ALCQ (qual. cardinality) Syntax/Semantics

ALCQ Signature = ALC Signature + $\{\leqslant n\ r.C, \geqslant n\ r.C\}$

- $\leqslant$n r.C – Corresponds to restriction that r is related to no more than n Cs
- $\geqslant$n r.C – Corresponds to restriction that r is related to no fewer than n Cs



OWL2 DL and SROIQ

- OWL2 DL corresponds to the description logic SROIQ

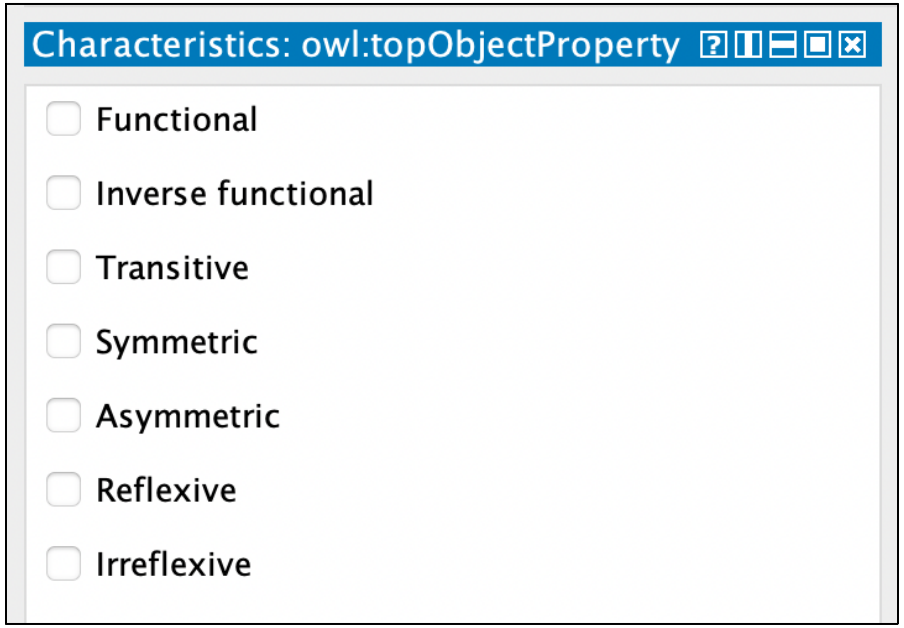
- SROIQ:
    - S = ALC
    - R = Role Axiom Extension
    - O = Nominals
    - I = Inverses
    - Q = Qualified Cardinalities

Role Axiom Extension

- SROIQ’s role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity



Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity

Role Inclusion

- Allowing role inclusion axioms facilitates chaining of roles, such as:

If x owns y and y part of z then x owns z

- Role inclusion axioms have the form:

$$\mathbf{r_1 \circ \ldots \circ r_n \sqsubseteq r}$$

- But **not just any role chains** are allowed...to see why, we first define *simple* and *non-simple roles*

Role Inclusion

- Role inclusion axioms have the form:

$$\mathbf{r_1 \circ \ldots \circ r_n \sqsubseteq r}$$

- Any role inclusion axiom in which n=1, is *simple*

- For any role inclusion axiom:
  1. Every role in $\mathbf{r_1 \circ \ldots \circ r_n \sqsubseteq r}$ with n>1, is *non-simple*
  2. Every role in a simple role inclusion $\mathbf{s \sqsubseteq r}$ with a non-simple s, is *non-simple*
  3. Every r- where r is *non-simple*, is *non-simple*
  4. No other role is *non-simple*

Example

- Which of the following are *simple* and which are *non-simple*?
  - motherOf $\sqsubseteq$ parentOf
  - parentOf $\sqsubseteq$ ancestorOf
  - ancesterOf $\circ$ ancestorOf $\sqsubseteq$ ancestorOf
  - ancestorOf$^-$ $\sqsubseteq$ descendantOf$^-$

Example

1. Every role in $r_1 \circ \ldots \circ r_n \sqsubseteq r$ with $n>1$, is ***non-simple***
2. Every role in a simple role inclusion $s \sqsubseteq r$ with a non-simple s, is ***non-simple***
3. Every r- where r is ***non-simple***, is ***non-simple***
4. No other role is ***non-simple***

- Which of the following are *simple* and which are *non-simple*?
  - motherOf ⊑ parentOf
  - parentOf ⊑ ancestorOf
  - ancesterOf ∘ ancestorOf ⊑ ancestorOf
  - ancestorOf− ⊑ descendantOf−

1. Every role in $r_1 \circ \ldots \circ r_n \sqsubseteq r$ with $n>1$, is ***non-simple***
2. Every role in a simple role inclusion $s \sqsubseteq r$ with a non-simple s, is ***non-simple***
3. Every r- where r is ***non-simple***, is ***non-simple***
4. No other role is ***non-simple***

Example

- Which of the following are *simple* and which are *non-simple*?

  - motherOf $\sqsubseteq$ parentOf
  - parentOf $\sqsubseteq$ ancestorOf
  - ancesterOf $\circ$ ancestorOf $\sqsubseteq$ ancestorOf
  - ancestorOf− $\sqsubseteq$ descendantOf−

According to 1, if there is a role chain consisting of more than one occurrence of a given role, that role is non-simple

Example

1. Every role in $r_1 \circ \ldots \circ r_n \sqsubseteq r$ with $n>1$, is ***non-simple***
2. Every role in a simple role inclusion $s \sqsubseteq r$ with a non-simple s, is ***non-simple***
3. Every r- where r is ***non-simple***, is ***non-simple***
4. No other role is ***non-simple***

- Which of the following are *simple* and which are *non-simple*?
  - motherOf ⊑ parentOf
  - parentOf ⊑ ancestorOf
  - ancesterOf ∘ ancestorOf ⊑ ancestorOf
  - ancestorOf− ⊑ descendantOf−

We can then conclude that ancestorOf is non-simple

1. Every role in $r_1 \circ \ldots \circ r_n \sqsubseteq r$ with $n>1$, is *non-simple*
2. Every role in a simple role inclusion $s \sqsubseteq r$ with a non-simple s, is *non-simple*
3. Every r- where r is *non-simple*, is *non-simple*
4. No other role is *non-simple*

Example

- Which of the following are *simple* and which are *non-simple*?

  - motherOf ⊑ parentOf
  - parentOf ⊑ ancestorOf
  - ancesterOf ∘ ancestorOf ⊑ ancestorOf
  - ancestorOf− ⊑ descendantOf−

Moreover, according to 3 and the fact that ancestorOf is non-simple, it follows that ancestorOf− is also non-simple

1. **Every role in $r_1 \circ \ldots \circ r_n \sqsubseteq r$ with $n > 1$, is *non-simple***
2. **Every role in a simple role inclusion $s \sqsubseteq r$ with a non-simple s, is *non-simple***
3. **Every r- where r is *non-simple*, is *non-simple***
4. **No other role is *non-simple***

Example

- Which of the following are *simple* and which are *non-simple*?
  - motherOf ⊑ parentOf
  - parentOf ⊑ ancestorOf
  - ancesterOf ∘ ancestorOf ⊑ ancestorOf
  - ancestorOf− ⊑ descendantOf−

And by 2, because ancestorOf− is non-simple, so too is descendentOf−

Example

1. **Every role in $r_1 \circ \ldots \circ r_n \sqsubseteq r$ with n>1, is *non-simple***
2. **Every role in a simple role inclusion $s \sqsubseteq r$ with a non-simple s, is *non-simple***
3. **Every r- where r is *non-simple*, is *non-simple***
4. **No other role is *non-simple***

- Which of the following are *simple* and which are *non-simple*?

  - motherOf ⊑ parentOf
  - parentOf ⊑ ancestorOf
  - ancesterOf ∘ ancestorOf ⊑ ancestorOf
  - ancestorOf− ⊑ descendantOf−

Lastly, by 4 all other roles are simple

Regularity

- To maintain decidability, we have to restrict any non-simple role inclusion axioms to those that have the property of being “regular”

- Without going into much detail, this just means if you’re going to have role inclusion chains, then then any combination of those roles and their inverses must exhibit a *strict partial order*

- In other words, a simple tree structure

Regularity

- To maintain decidability, we have to restrict any non-simple role inclusion axioms to those that have the property of being "regular"

- Without going into much detail, this just means if you're going to have role inclusion chains, then then any combination of those roles and their inverses must exhibit a *strict partial order*

- In other words, a simple tree structure

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity

Role Axiom Extension

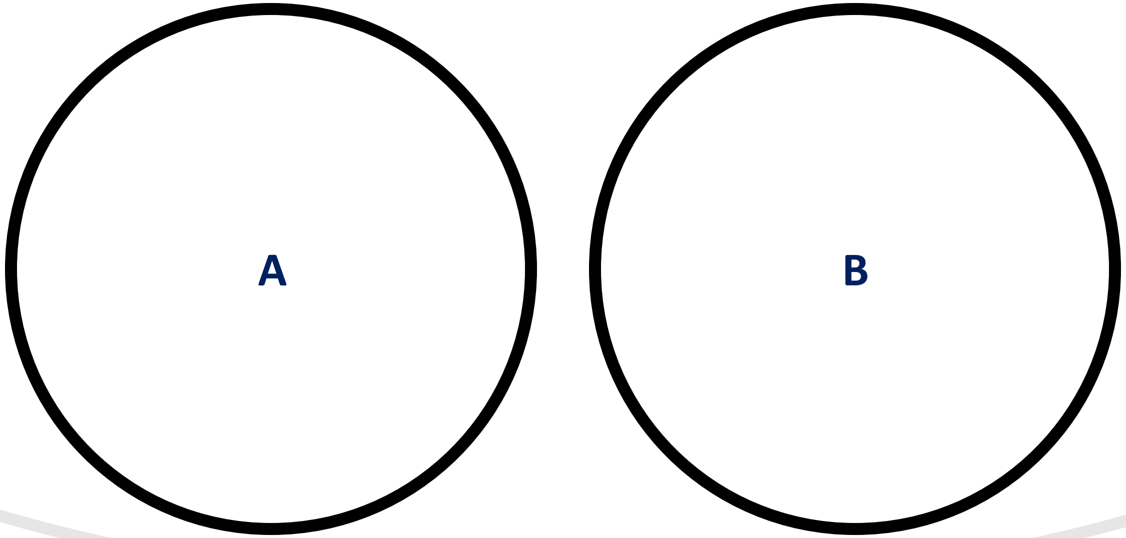
- SROIQ's role axioms include:
  - Inclusion
  - Disjointness – A and B are disjoint just in case they share no individuals

$$\textbf{DisjointWith(A, B)} = \mathbf{A^I} \cap \mathbf{B^I} = \emptyset$$

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness – A and B are disjoint just in case they share no individuals

DisjointWith(A, B) = $A^I \cap B^I = \emptyset$



Role Axiom Extension

- SROIQ's role axioms include:
    - Inclusion
    - Disjointness
    - Transitivity
    - Symmetry
    - Asymmetry
    - Reflexivity
    - Irreflexivity

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity – If x related to y and y related to z, then x related to z

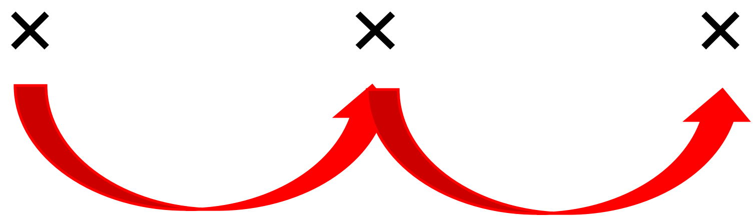
$$\mathbf{Trans(R) = R^{I} \circ R^{I} \subseteq R^{I}}$$

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity – If x related to y and y related to z, then x related to z

THING

$$\mathbf{Trans(R) = R^I \circ R^I \subseteq R^I}$$

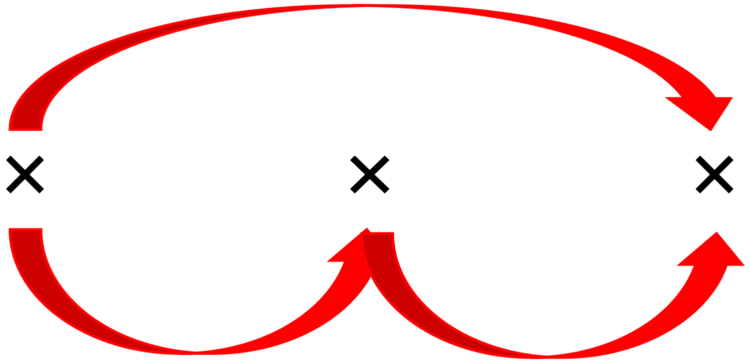


Role Axiom Extension

- SROIQ’s role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity – If x related to y and y related to z, then x related to z

THING

$$\mathbf{Trans(R) = R^I \circ R^I \subseteq R^I}$$



Role Axiom Extension

- SROIQ’s role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry – If x related to y then y related to x

$$(x,y) \in R^I \Rightarrow (y,x) \in R^I$$

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry – If x related to y then y related to x

THING

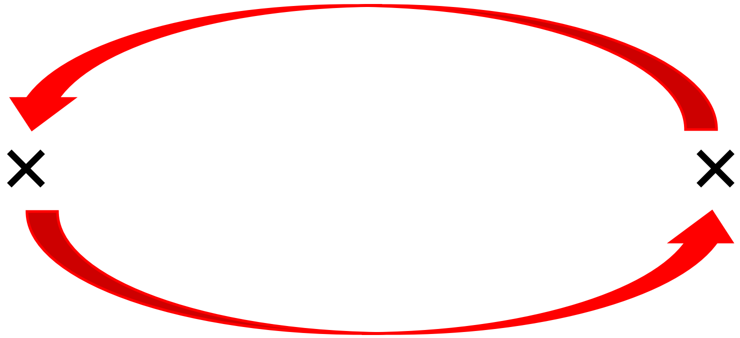
$$\mathbf{(x,y) \in R^I \Rightarrow (y,x) \in R^I}$$

Role Axiom Extension

- SROIQ’s role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry – If x related to y then y related to x

THING

$$(x,y) \in R^I \Rightarrow (y,x) \in R^I$$



Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry – If x is related to y then y is not related to x

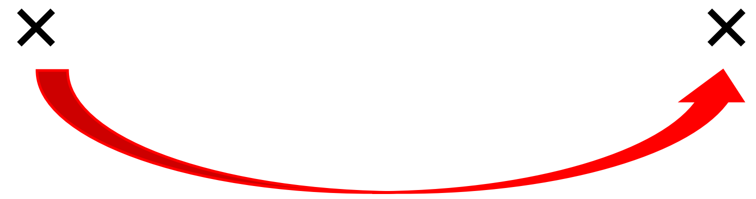
$$(x,y) \in R^I \Rightarrow (y,x) \notin R^I$$

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry – If x is related to y then y is not related to x

THING

$$(x,y) \in R^I \Rightarrow (y,x) \notin R^I$$

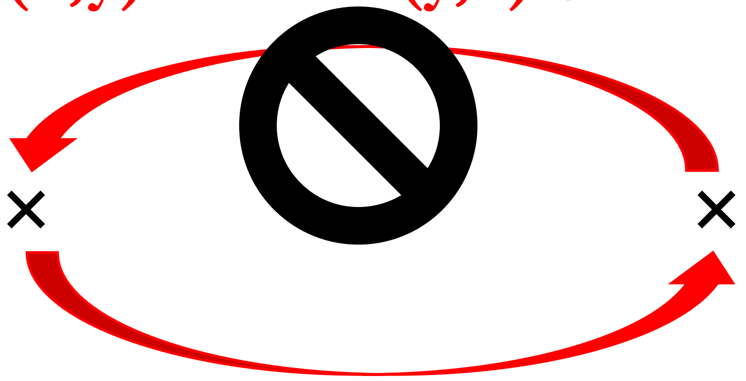


Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry – If x is related to y then y is not related to x

THING

$$(x,y) \in R^{I} \Rightarrow (y,x) \notin R^{I}$$



Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity – For all x, x is related to x

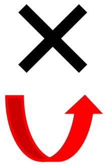
$$\{(x,x) \mid x \in \mathrm{Domain}\} \subseteq R^I$$

Role Axiom Extension

- SROIQ’s role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity – For all x, x is related to x

THING

$$\{(x,x) \mid x \in \text{Domain}\} \subseteq R^{I}$$



Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity – There is no x such that x is related to x

$$\{(x,x) \mid x \in \text{Domain}\} \cap R^I = \emptyset$$

Role Axiom Extension

- SROIQ's role axioms include:
  - Inclusion
  - Disjointness
  - Transitivity
  - Symmetry
  - Asymmetry
  - Reflexivity
  - Irreflexivity – There is no x such that x is related to x

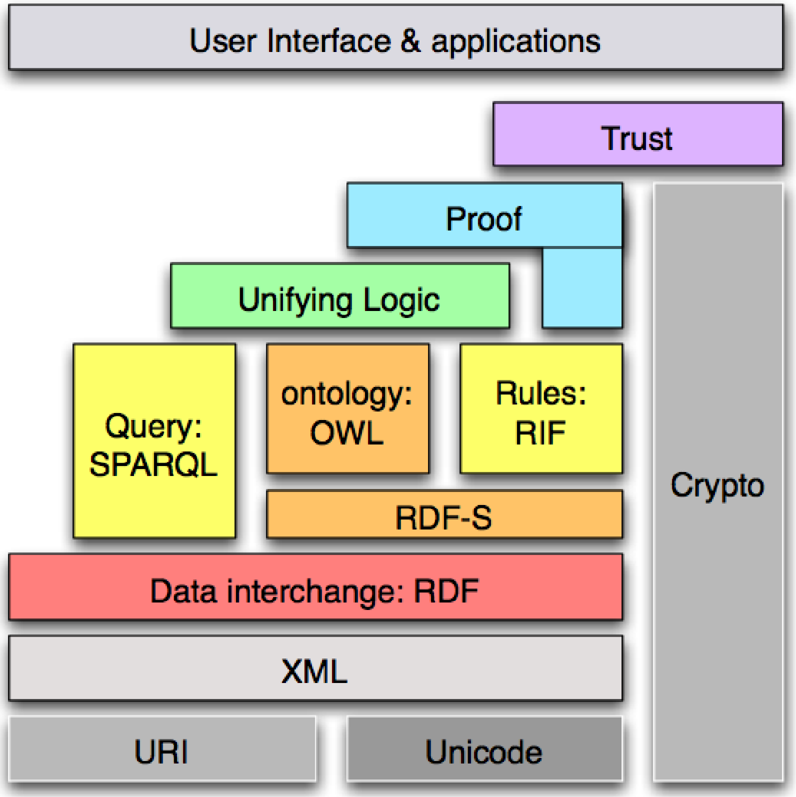
THING

$$\{(x,x) \mid x \in Domain\} \cap R^I = \emptyset$$

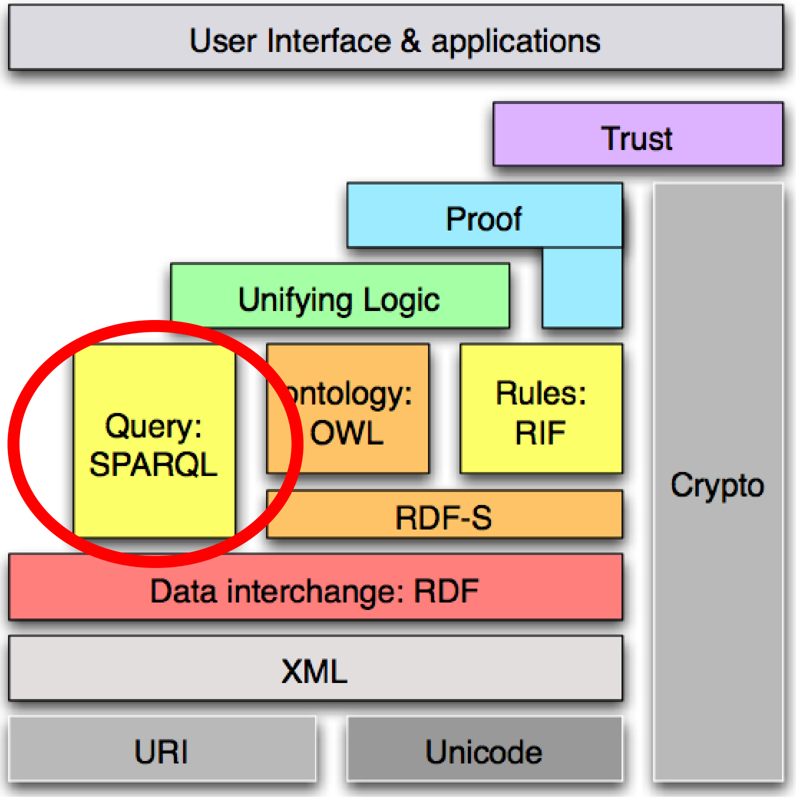
Outline

- OWL 2 Direct Semantics
- SPARQL
- Mapping

Semantic Web Stack

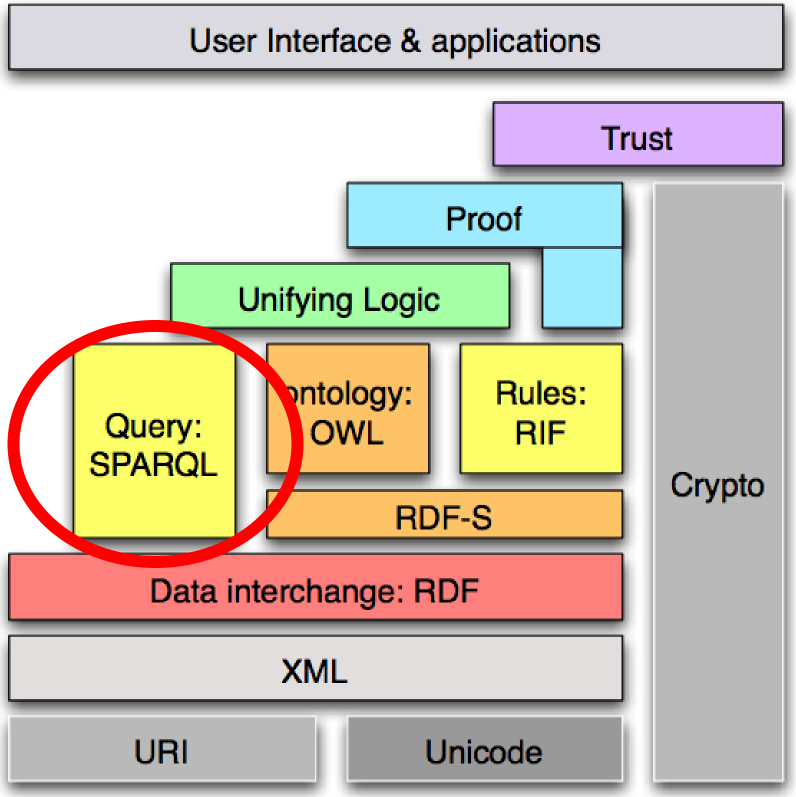


Semantic Web Stack



Semantic Web Stack

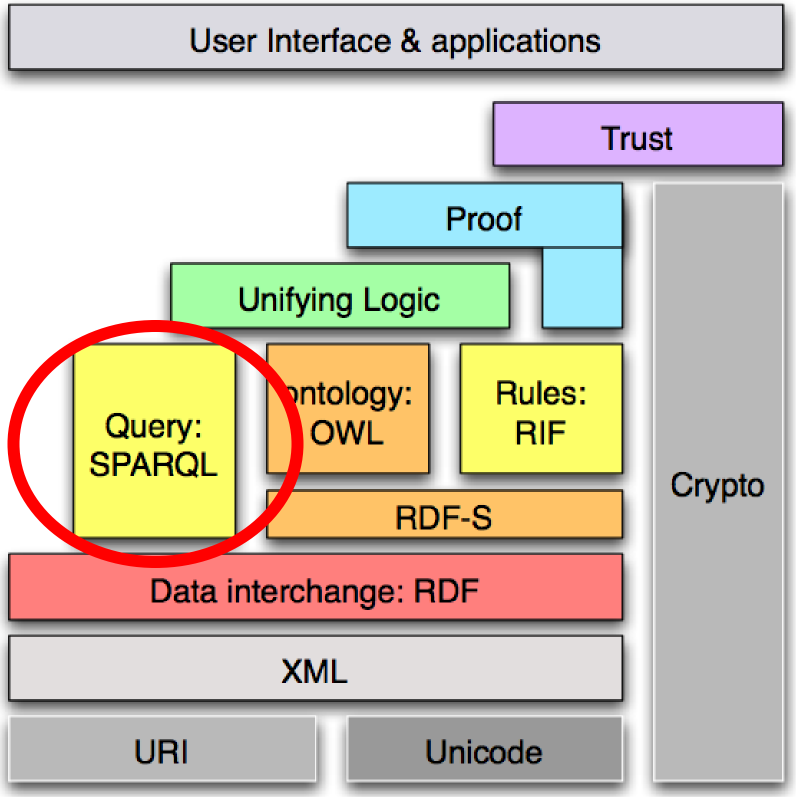
- "SPARQL" stands for:
  - **S**PARQL **P**rotocol
  - **A**nd **R**DF
  - **Q**uery **L**anguage



Semantic Web Stack

- "SPARQL" stands for:
  - **S**PARQL **P**rotocol
  - **A**nd **R**DF
  - **Q**uery **L**anguage

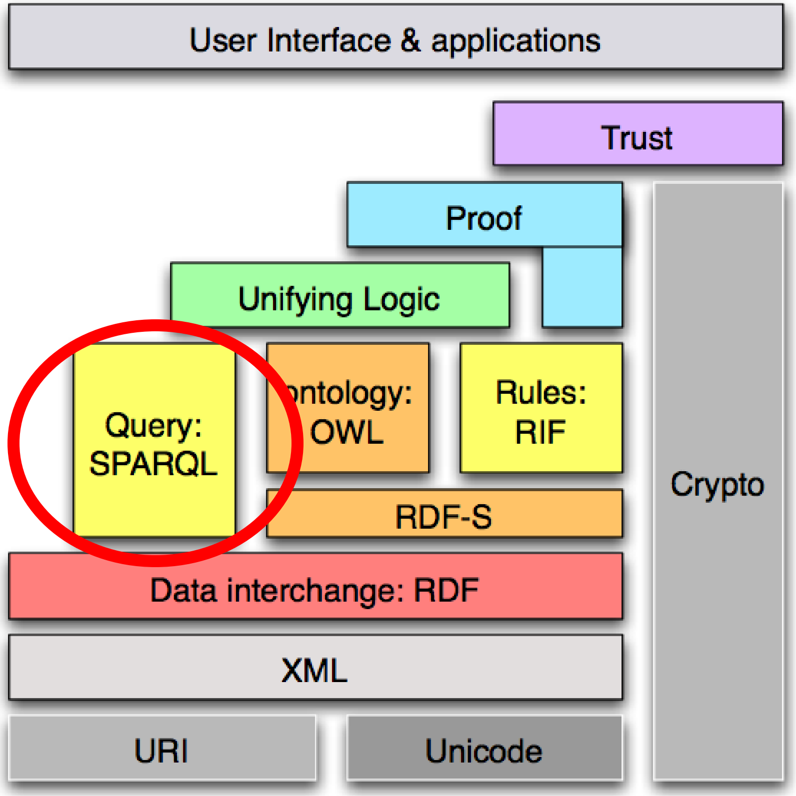
- SPARQL is a:
  - Core semantic web technology
  - *Query language* for RDF
  - A *protocol* for transmitting queries over HTTP



Semantic Web Stack

- "SPARQL" stands for:
  - **S**PARQL **P**rotocol
  - **A**nd **R**DF
  - **Q**uery **L**anguage

- SPARQL is a:
  - Core semantic web technology
  - *Query language* for RDF
  - A *protocol* for transmitting queries over HTTP



Query Languages

- Database query languages are languages used to extract from and manipulate data in information systems

- Traditionally, data was stored in *relational databases*, which were fixed, built on the closed world assumption, but somewhat easy to query using well-known languages like SQL

- RDF databases are flexible, adopt the open world assumption, and querying requires a language suited to these features

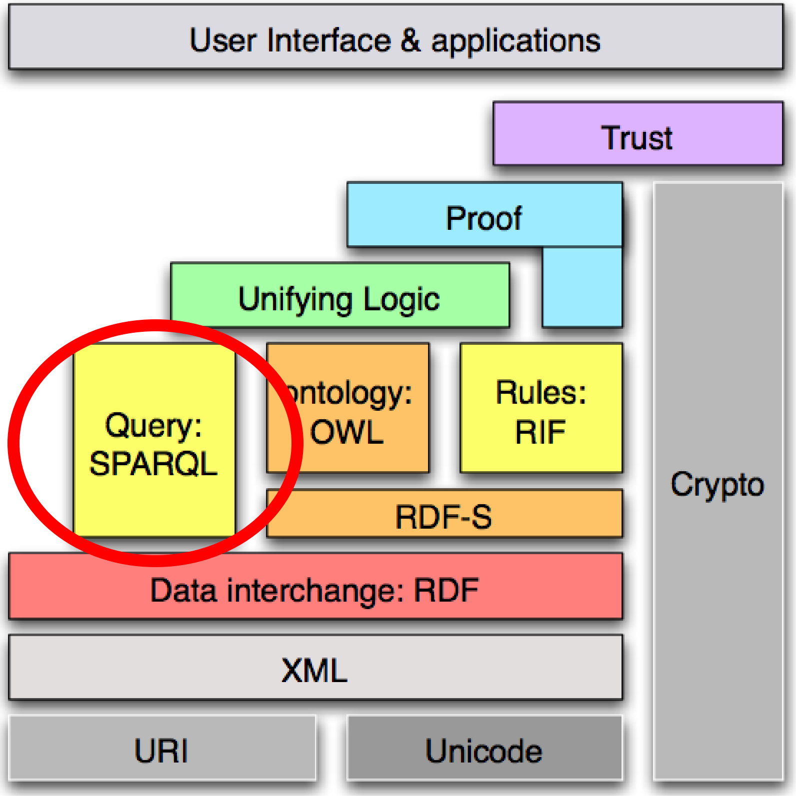
SPARQL Query Language

- SPARQL is a query language for RDF databases
- SPARQL shares much in common with query languages like SQL, and many differences
- SPARQL queries focus on what users want to know about the data; SQL queries focus on how the data is structured
- You will become a SPARQL Ninja™ by course end

Semantic Web Stack

- "SPARQL" stands for:
  - **S**PARQL **P**rotocol
  - **A**nd **R**DF
  - **Q**uery **L**anguage

- SPARQL is a:
  - Core semantic web technology
  - *Query language* for RDF
  - A *protocol* for transmitting queries over HTTP



HTTP Protocol

- SPARQL is protocol that specified how to send queries over the web to an endpoint using HTTP requests

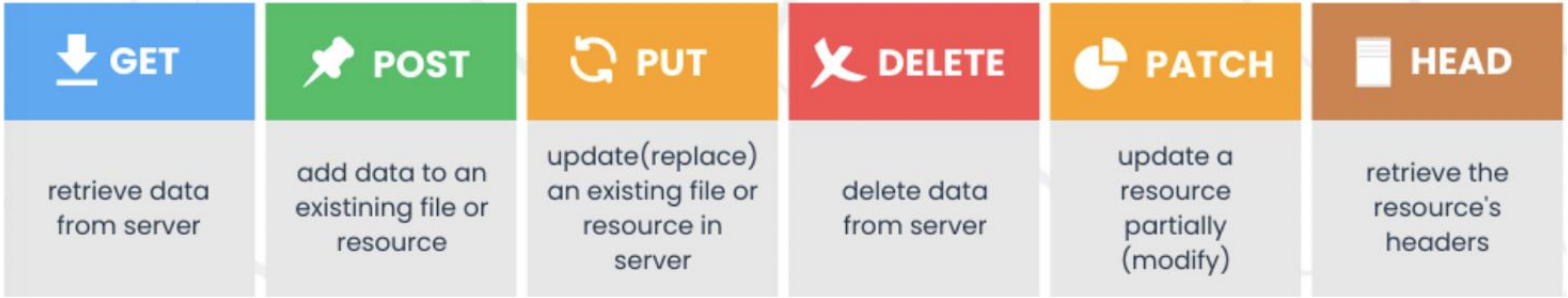
- A *protocol* in this context is a set of rules that prescribe how parties communicate

- HTTP is a protocol defined between a client – software that reads data from a server – and a server – where the data is stored

HTTP Protocol

- HTTP is a *request-response* protocol, a client has to send a request for data before the server will respond with that data

- HTTP request methods:



SPARQL Protocol

- Unlike other query languages, SPARQL is designed with a protocol that enables it to natively query over HTTP requests

- What this means is that data exposed by SPARQL on *any* server can be queried by any SPARQL client

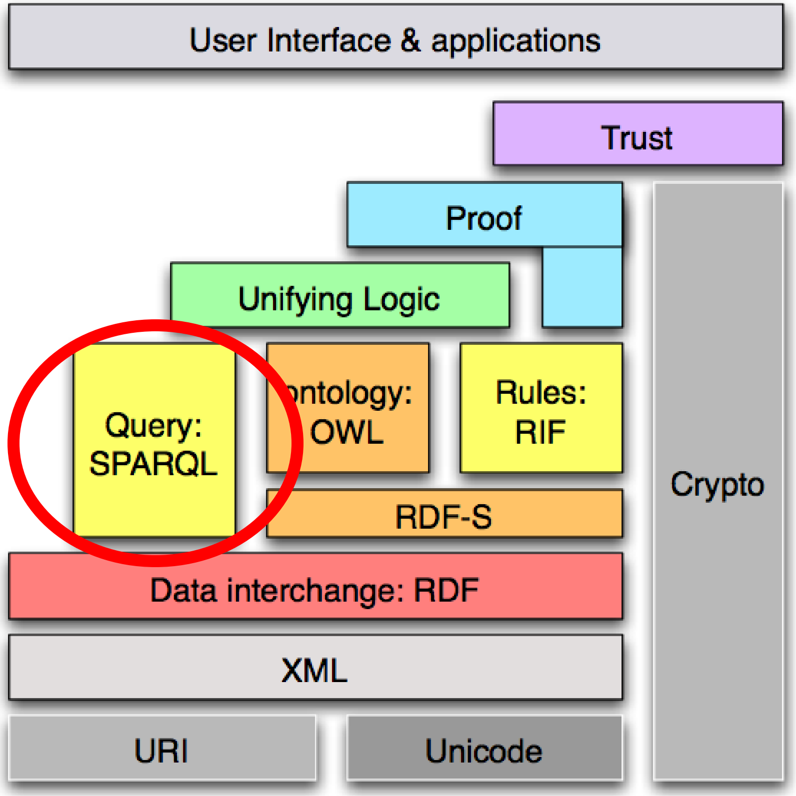
- In contrast, query languages like SQL can only be queried locally

- SPARQL allows, for example, combining data from many different sources, dynamically

Semantic Web Stack

- "SPARQL" stands for:
  - **S**PARQL **P**rotocol
  - **A**nd **R**DF
  - **Q**uery **L**anguage

- SPARQL is a:
  - Core semantic web technology
  - *Query language* for RDF
  - A *protocol* for transmitting queries over HTTP



SPARQL Query Structure

```
PREFIX rdfs: http://www.w3.org/2000/01/rdf-schema#

SELECT ?subject ?label
WHERE
  {
    ?subject rdfs:label ?label .
   }
```

SPARQL Query Structure

```
PREFIX rdfs: http://www.w3.org/2000/01/rdf-schema#
```

Declare the namespace

```
SELECT ?subject ?label
WHERE
  {
    ?subject rdfs:label ?label .
   }
```

SPARQL Query Structure

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

Declare the namespace

SELECT ?subject ?label **then return any data...**

WHERE

{

?subject rdfs:label ?label .

}

SPARQL Query Structure

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

Declare the namespace

SELECT ?subject ?label **then return any data...**

WHERE **...that satisfies the condition...**

```
{
  ?subject rdfs:label ?label .
}
```

SPARQL Query Structure

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

Declare the namespace

SELECT ?subject ?label **then return any data...**

WHERE **...that satisfies the condition...**

{

?subject rdfs:label ?label . **...bears rdfs:label to something.**

}

SPARQL Query Structure

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

Declare the namespace

SELECT ?subject ?label

then return any data...

WHERE

...that satisfies the condition...

{

?subject rdfs:label ?label .

...bears rdfs:label to something.

}

Returns list of subjects and their labels

SELECT and WHERE

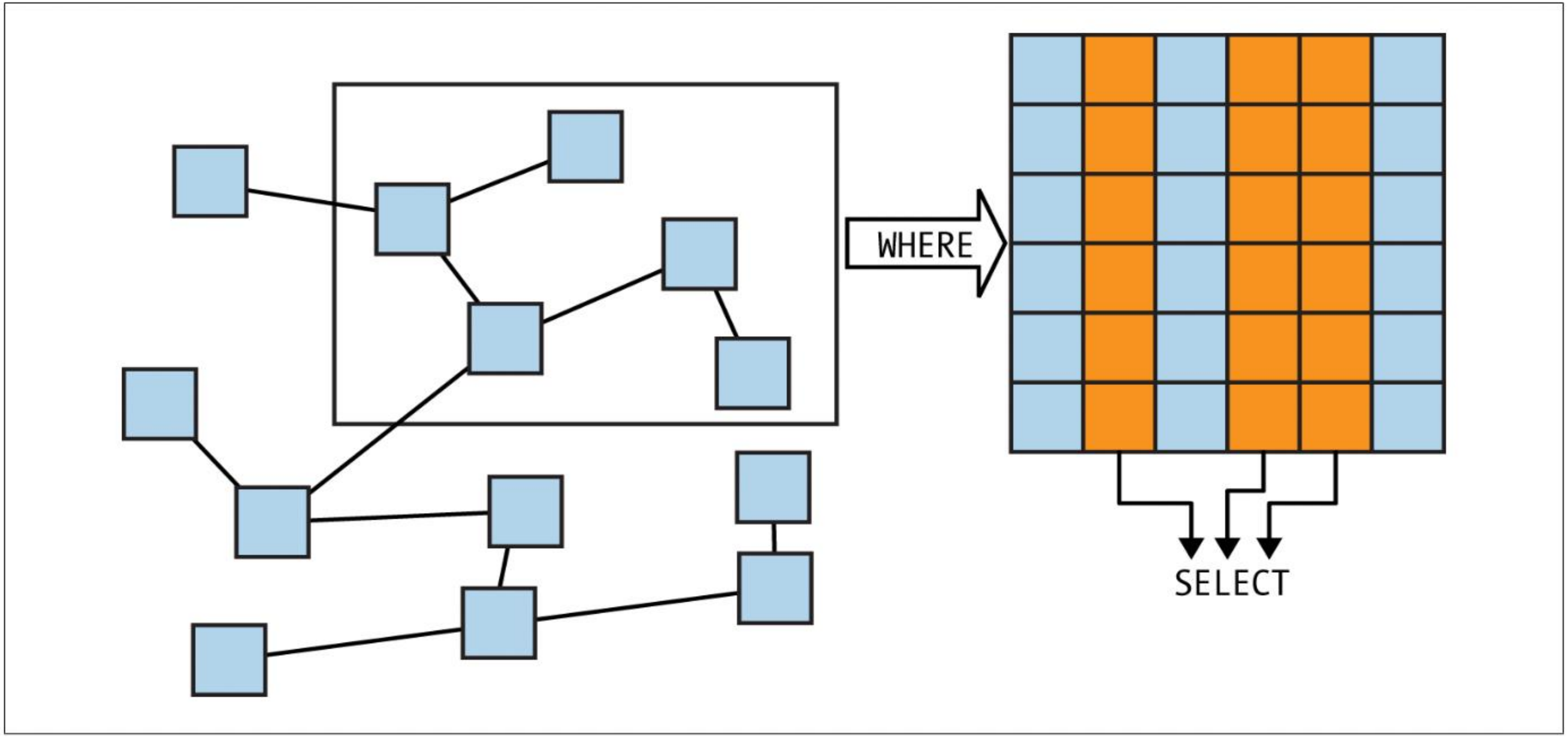


Figure 1-1. WHERE specifies data to pull out; SELECT picks which data to display

Virtuoso SPARQL Query Editor

Default Data Set Name (Graph IRI)

Query Text

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

SELECT DISTINCT ?subject ?label
WHERE {
 ?subject rdfs:label ?label .
} LIMIT 10
```

Endpoint

https://makg.org/sparql

Virtuoso SPARQL Query Editor

Default Data Set Name (Graph IRI)

Query Text

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

SELECT DISTINCT ?subject ?label
WHERE {
 ?subject rdfs:label ?label .
} LIMIT 10
```

Endpoint
https://makg.org/sparql

subject	label
[illegible]w3.org/2002/07/owl#equivalentClass	"equivalentClass"
ht[illegible]y3.org/2002/07/owl#equivalentProperty	"equivalentProperty"
htt[illegible]w.w3.org/2002/07/owl#InverseFunctionalProperty	"InverseFunctionalProperty"
http://www.w3.org/2002/07/owl#SymmetricProperty	"SymmetricProperty"
http://www.w3.org/2002/07/owl#FunctionalProperty	"FunctionalProperty"
http://www.w3.org/2002/07/owl#inverseOf	"inverseOf"
http://www.w3.org/2002/07/owl#TransitiveProperty	"TransitiveProperty"
http://www.w3.org/2002/07/owl#Thing	"Thing"
http://www.w3.org/2002/07/owl#Class	"Class"
http://www.w3.org/2002/07/owl#Nothing	"Nothing"

OPTIONAL

- Because the basic unit of RDF is a triple, SPARQL queries by default return only triples that satisfy conditions in the WHERE clause

- That is, partial matches are not returned by default

- Consider, you might want to return the birth and death dates of everyone in a database...

OPTIONAL

```
SELECT ?subject ?birthday ?deathday
WHERE
  {
    ?subject ex:has_birthday ?birthday ;
             ex:has_deathday ?deathday .
  }
```

OPTIONAL

```
SELECT ?subject ?birthday ?deathday
WHERE
  {
    ?subject ex:has_birthday ?birthday ;
             ex:has_deathday ?deathday .
  }
```

Suppose the database is of my family, and includes my grandmother's birthday and deathday

OPTIONAL

```
SELECT ?subject ?birthday ?deathday
WHERE
  {
    ?subject ex:has_birthday ?birthday ;
             ex:has_deathday ?deathday .
  }
```

This query will return that information for my grandmother, but it will only return information for individuals who have *both* a birthday and a deathday

OPTIONAL

```
SELECT ?subject ?birthday ?deathday
WHERE
  {
    ?subject ex:has_birthday ?birthday ;
             ex:has_deathday ?deathday .
  }
```

It is plausible, however, that one might want to return individuals and their birthdays, even if they are still alive…

OPTIONAL

```
SELECT ?subject ?birthday ?deathday
WHERE
  {
    ?subject ex:has_birthday ?birthday .
    OPTIONAL
      {
        ?subject ex:has_deathday ?deathday .
      }
  }
```

OPTIONAL operates like a conditional; return everyone with a birthday and *if they have a deathday*, return that too

FILTER

- The FILTER keyword restricts the conditions in the WHERE clause using boolean and other functions

```
PREFIX ex: <https://example.com/>
SELECT ?person ?name
WHERE
  {
     ?person rdf:type ex:Person ;
                  ex:name ?name ;
                   ex:age ?age .
   FILTER (xsd:integer(?age) >= 18)
  }
```

FILTER

- The FILTER keyword restricts the conditions in the WHERE clause using boolean and other functions

```
PREFIX ex: <https://example.com/>
SELECT ?person ?name
WHERE
  {
     ?person rdf:type ex:Person ;
                    ex:name ?name ;
                     ex:age ?age .
   FILTER (xsd:integer(?age) >= 18)
  }
```

Declare the namespace

FILTER

- The FILTER keyword restricts the conditions in the WHERE clause using boolean and other functions

```
PREFIX ex: <https://example.com/>
SELECT ?person ?name
WHERE
  {
     ?person rdf:type ex:Person ;
                   ex:name ?name ;
                    ex:age ?age .
   FILTER (xsd:integer(?age) >= 18)
  }
```

Declare the namespace

SELECT variables…

FILTER

- The FILTER keyword restricts the conditions in the WHERE clause using boolean and other functions

```
PREFIX ex: <https://example.com/>
SELECT ?person ?name
WHERE
  {
     ?person rdf:type ex:Person ;
                    ex:name ?name ;
                     ex:age ?age .
   FILTER (xsd:integer(?age) >= 18)
  }
```

Declare the namespace

SELECT variables…

…WHERE…

FILTER

- The FILTER keyword restricts the conditions in the WHERE clause using boolean and other functions

```
PREFIX ex: <https://example.com/>
SELECT ?person ?name
WHERE
  {
    ?person rdf:type ex:Person ;
                   ex:name ?name ;
                    ex:age ?age .
   FILTER (xsd:integer(?age) >= 18)
  }
```

Declare the namespace

SELECT variables…

…WHERE…

…someone is a Person…

FILTER

- The FILTER keyword restricts the conditions in the WHERE clause using boolean and other functions

```
PREFIX ex: <https://example.com/>
SELECT ?person ?name
WHERE
  {
    ?person rdf:type ex:Person ;
            ex:name ?name ;
            ex:age ?age .
  FILTER (xsd:integer(?age) >= 18)
  }
```

Declare the namespace

SELECT variables…

…WHERE…

…someone is a Person…

…with a name…

FILTER

- The FILTER keyword restricts the conditions in the WHERE clause using boolean and other functions

```
PREFIX ex: <https://example.com/>
SELECT ?person ?name
WHERE
  {
    ?person rdf:type ex:Person ;
                ex:name ?name ;
                ex:age ?age .
  FILTER (xsd:integer(?age) >= 18)
  }
```

Declare the namespace
SELECT variables…
…WHERE…
…someone is a Person…
…with a name…
…and age…

FILTER

- The FILTER keyword restricts the conditions in the WHERE clause using boolean and other functions

```
PREFIX ex: <https://example.com/>
SELECT ?person ?name
WHERE
  {
    ?person rdf:type ex:Person ;
                    ex:name ?name ;
                     ex:age ?age .
   FILTER (xsd:integer(?age) >= 18)
  }
```

Declare the namespace

SELECT variables…

…WHERE…

…someone is a Person…

…with a name…

…and age…

…but keep only the people over 18.

FILTER

- FILTER functions include:

Comparators: `<, >, =, <=, >=, !=`

Regular expressions: `regex(?x, "A.*")`

Test variable values: `isURI(?x), isBlank(?x), isLiteral(?x), bound(?x)`

And: `&&`

Or: `||`

Not: `!`

`()`

`YEAR(Date), MONTH(Date), DAY(Date)`
`HOURS(Date), MINUTES(Date), SECONDS(Date)`
`NOW()`

FILTER

Logical combinations of filter clauses, e.g.
FILTER (xsd:integer(?age)>18
&& xsd:integer(?age)<25)

- FILTER functions include:

Comparators: <, >, =, <=, >=, !=

Regular expressions: regex(?x, "A.*")

Test variable values: isURI(?x), isBlank(?x), isLiteral(?x), bound(?x)

And: &&

Or: ||

Not: !

()

YEAR(Date), MONTH(Date), DAY(Date)
HOURS(Date), MINUTES(Date), SECONDS(Date)
NOW()

FILTER

Filter functions to restrict results, e.g.
FILTER (regex(?x, “hello”, “i”))

- FILTER functions include:

Comparators: `<, >, =, <=, >=, !=`

Regular expressions: `regex(?x, "A.*")`

Test variable values: `isURI(?x), isBlank(?x), isLiteral(?x), bound(?x)`

And: `&&`

Or: `||`

Not: `!`

`()`

`YEAR(Date), MONTH(Date), DAY(Date)`
`HOURS(Date), MINUTES(Date), SECONDS(Date)`
`NOW()`

Outline

- OWL 2 Direct Semantics
- SPARQL
- Mapping

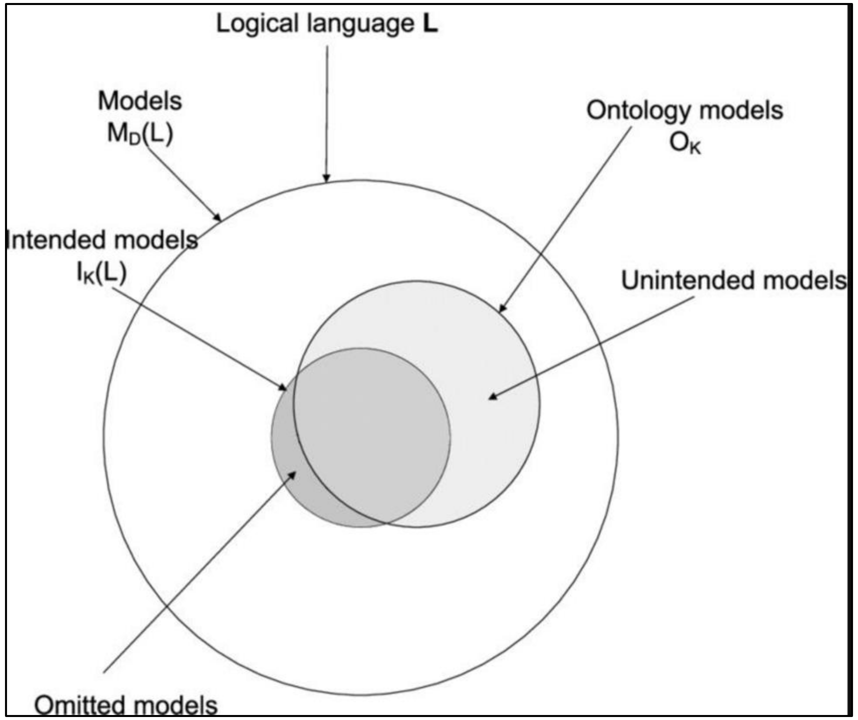
Definitions

Given two ontologies, determine which entities and relations represent share common intended semantics

- **Ontology matching** is the process of identifying similarity relations between ontologies

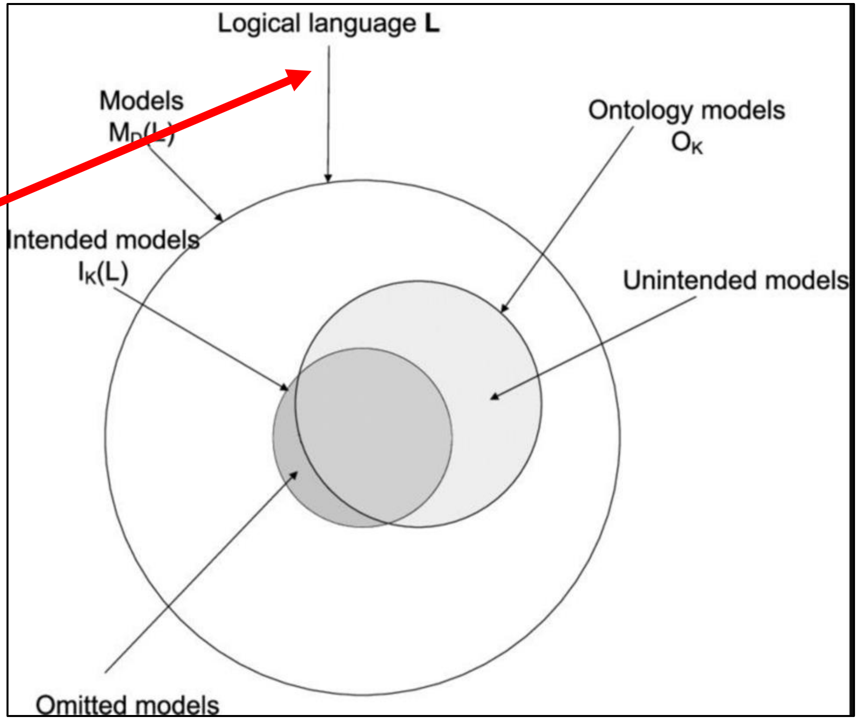
- **Ontology alignment** similarity relations that result from matching

	Ontology A	Ontology B
Different names for the same concept	`Private First Class`	`PFC`
Same term for different concepts	`Facility` (Building or Infrastructure)	`Facility` (Target)
Scope	Air and Field Assets	Air and Field Operations
Different modeling conventions	`Operation` is a class	`Operation` is a relation
Granularity	`Ground Vehicle XYZ`	`Ground Vehicle`



	Ontology A	Ontology B
Different names for the same concept	`Private First Class`	`PFC`
Same term for different concepts	`Facility` (Building or Infrastructure)	`Facility` (Target)
Scope	Air and Field Assets	Air and Field Operations
Different modeling conventions	`Operation` is a class	`Operation` is a relation
Granularity	`Ground Vehicle XYZ`	`Ground Vehicle`

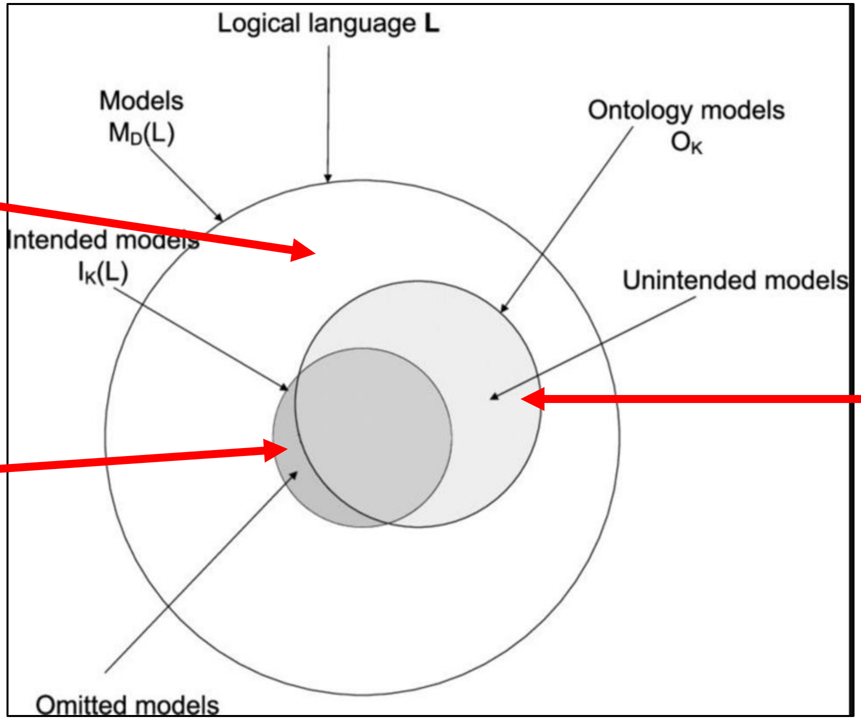
Web Ontology Language (OWL)



	Ontology A	Ontology B
Different names for the same concept	`Private First Class`	`PFC`
Same term for different concepts	`Facility` (Building or Infrastructure)	`Facility` (Target)
Scope	Air and Field Assets	Air and Field Operations
Different modeling conventions	`Operation` is a class	`Operation` is a relation
Granularity	`Ground Vehicle XYZ`	`Ground Vehicle`

What you could say in OWL

What you want to say in OWL



What you do not want to say in OWL

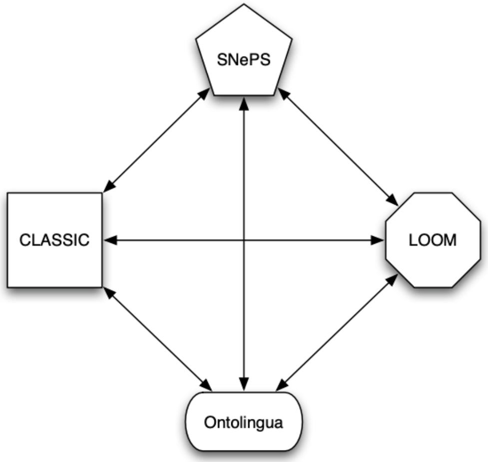
History of Mapping: 90s – 00s

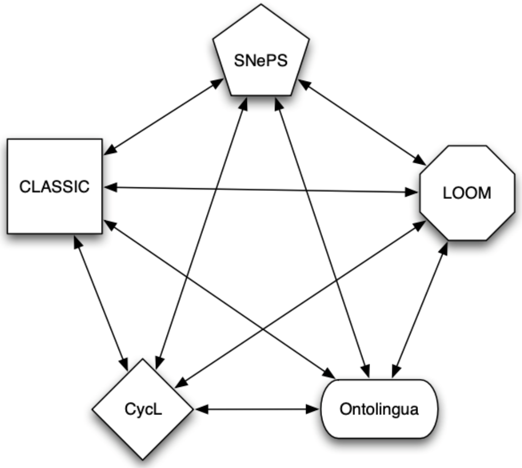
- Emerged from the knowledge representation and AI communities, largely driven by the need to integrate heterogeneous knowledge bases
- Strategies were manual: experts compared class hierarchies, identified overlaps, and wrote crosswalks or bridging axioms between ontologies
- Early efforts (e.g., UMLS Metathesaurus, Gene Ontology cross-links) focused on lexical matching (shared labels, synonyms) and human curation

Minimizing Mappings

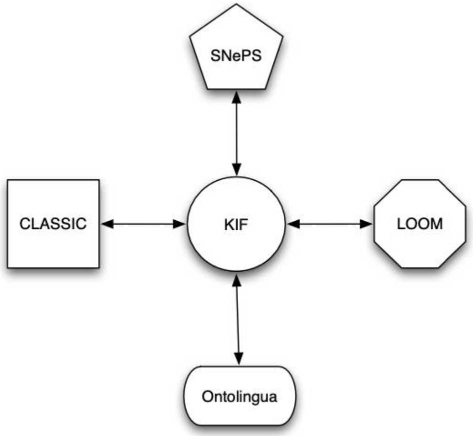
- Connecting disparate datasets requires two-way mappings:
  - 2 datasets – 2 mappings
  - 3 datasets – 6 mappings
  - 4 datasets – 12 mappings

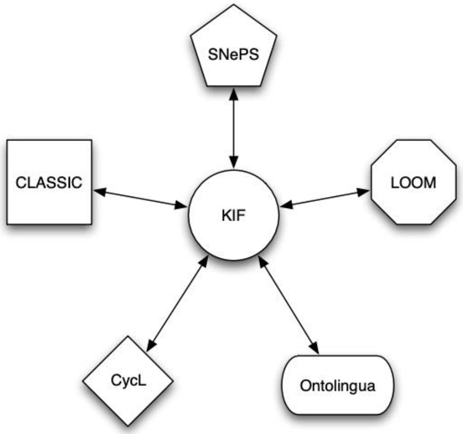
  ….





- Ontologies with common semantics help minimize the number of needed mappings for interoperability



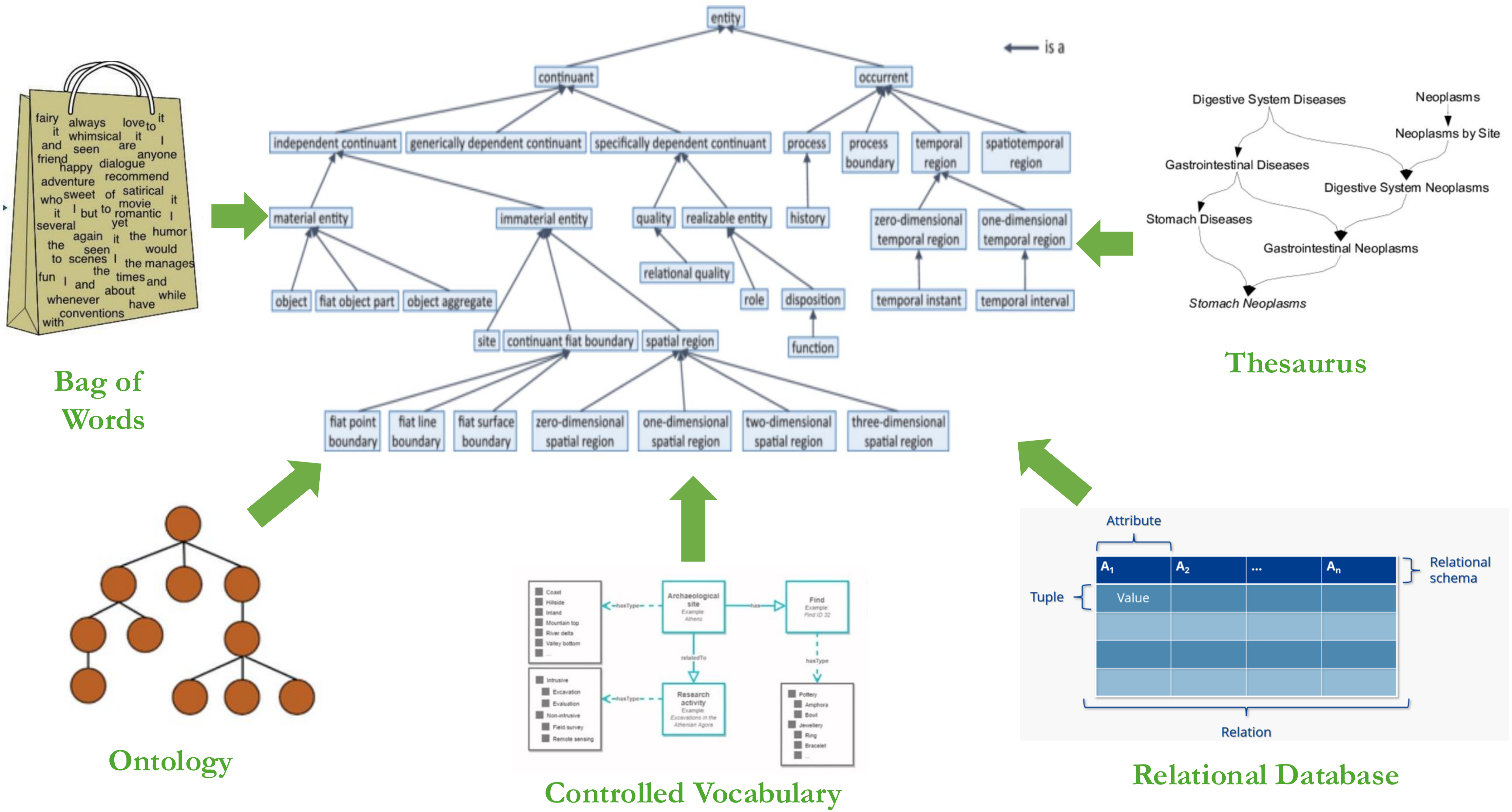


History of Mapping: 05s

- Semantic Web stack matured with tools like PROMPT (Protégé plugin) and FOAM
- Strategies emphasized:
  - Lexical similarity (string, edit distance, WordNet expansion)
  - Structural similarity (aligning hierarchies, subsumption patterns)
  - Instance-based similarity (when data annotations were available)
- This period introduced the formalization of benchmarks and comparing system performance

History of Mapping: 2005 - 2015

- Upper level ontologies such as **Basic Formal Ontology (BFO)**, were increasingly used as **anchors for alignment**
- Committing domain ontologies to a common top-level framework, reducing the number of arbitrary cross-maps
- For example, the **OBO Foundry** adopted BFO as a unifying basis, which meant mappings were often recast as **subclass axioms under BFO categories**


Bag of Words
Ontology
Controlled Vocabulary
Relational Database
Thesaurus
entity
is a
continuant
occurrent
independent continuant
generically dependent continuant
specifically dependent continuant
process
process boundary
temporal region
spatiotemporal region
material entity
immaterial entity
quality
realizable entity
history
zero-dimensional temporal region
one-dimensional temporal region
object
fiat object part
object aggregate
relational quality
role
disposition
temporal instant
temporal interval
site
continuant fiat boundary
spatial region
function
fiat point boundary
fiat line boundary
fiat surface boundary
zero-dimensional spatial region
one-dimensional spatial region
two-dimensional spatial region
three-dimensional spatial region
Digestive System Diseases
Neoplasms
Neoplasms by Site
Gastrointestinal Diseases
Digestive System Neoplasms
Stomach Diseases
Gastrointestinal Neoplasms
Stomach Neoplasms
Archaeological site
Find
Research activity
Attribute
Relational schema
Tuple
Value
Relation

gistBFO: An Open-Source, BFO-Compatible Version of gist

Dylan ABNEY [a,1], Katherine STUDZINSKI [a], Giacomo DE COLLE [b,c],
Finn WILSON [b,c], Federico DONATO [b,c], and John BEVERLEY [b,c]
[a] *Semantic Arts, Inc.*
[b] *University at Buffalo*
[c] *National Center for Ontological Research*
ORCiD ID: Dylan Abney https://orcid.org/0009-0005-4832-2900, Katherine Studzinski https://orcid.org/0009-0001-3933-0643, Giacomo De Colle https://orcid.org/0000-0002-3600-6506, Finn Wilson https://orcid.org/0009-0002-7282-0836, Federico Donato https://orcid.org/0009-0001-6600-240X, John Beverley https://orcid.org/0000-0002-1118-1738

Abstract. gist is an open-source, business-focused ontology actively developed by Semantic Arts. Its lightweight design and use of everyday terminology has made it a useful tool for kickstarting domain ontology development in a range of areas including finance, government, and pharmaceuticals. The Basic Formal Ontology (BFO) is an ISO/IEC standard upper ontology that has similarly found practical application across a variety of domains, especially biomedicine and defense. Given its demonstrated utility, BFO was recently adopted as a baseline standard in the U.S. Department of Defense and Intelligence Community.

Because BFO sits at a higher level of abstraction than gist, we see an opportunity to align gist with BFO and get the benefits of both: one can kickstart domain ontology development with gist, all the while maintaining an alignment with the BFO standard. This paper presents such an alignment, which consists primarily of subclass relations from gist classes to BFO classes and includes some subproperty axioms. The union of gist, BFO, and this alignment is what we call "gistBFO." The upshot is that one can model instance data using gist and then instances of gist classes can be mapped to BFO. This not only achieves compliance with the BFO standard; it also enables interoperability with other domains already modeled using BFO. We describe a methodology for aligning gist and BFO, provide rationale for decisions we made about mappings, and detail a vision for future development.

Keywords. Ontology, upper ontology, ontology alignment, gist, BFO

Article | Open access | Published: 17 February 2025

A semantic approach to mapping the Provenance Ontology to Basic Formal Ontology

Tim Prudhomme, Giacomo De Colle, Austin Liebers, Alec Sculley, Peihong "Karl" Xie, Sydney Cohen & John Beverley

Scientific Data **12**, Article number: 282 (2025) | Cite this article

Abstract

The Provenance Ontology (PROV-O) is a World Wide Web Consortium (W3C) recommended ontology used to structure data about provenance across a wide variety of domains. Basic Formal Ontology (BFO) is a top-level ontology ISO/IEC standard used to structure a wide variety of ontologies, such as the OBO Foundry ontologies and the Common Core Ontologies (CCO). To enhance interoperability between these two ontologies, their extensions, and data organized by them, a mapping methodology and set of alignments are presented according to specific criteria which prioritize semantic and logical principles. The ontology alignments are evaluated by checking their logical consistency with canonical examples of PROV-O instances and querying terms that do not satisfy the alignment criteria as formalized in SPARQL. A variety of semantic web technologies are used in support of FAIR (Findable, Accessible, Interoperable, Reusable) principles.

FOWG Subgroup Report

IES-BFO Cross-Ontology Working Group

FOUST

Authors	Title
Accepted Papers	
Laure Vieu and Adrien Barton	Order in the Mereology of Slots
Brandon Bennett	Building Up: foundations and material for definitional ontology construction
Fumiaki Toyoshima and Ludger Jansen	Malfunctioning artifacts: A step towards a realizable-centered unifying account
Fumiaki Toyoshima and Satoru Niki	Subsumption in the Mirror of Ontological and Logical Choices
Ian Bailey, John Beverley, Helene Blackmore, Andreas Cola, Paul Cripps, Giacomo de Colle, Federico Donato, Amanda Hicks, David Limbaugh, Elena Milivinti, Chris Patridge, Rebecca Rafferty and Barry Smith	Comparing Information Exchange Standard and Basic Formal Ontology Design Patterns
Michel Dumontier, Remzi Çelebi, Komal Gilani, Isabelle de Zegher, Katerina Serafimova, Catalina Martínez Costa and Stefan Schulz	SULO — a simplified upper-level ontology
Lucas V. Vieira, Cauã R. Antunes, Mara Abel, Fabrício H. Rodigues and Lisa Stright	Towards a reference ontology of the spatial location of physical objects

https://www.dmi.unict.it/fois2025/?page_id=867

History of Mapping: 2005 - 2015

- Instead of one-off mappings, ontologists began creating **import modules** (via MIREOT, OntoFox, ROBOT extract) to bring terms across ontologies

- **Bridging ontologies** were created e.g., for anatomy across species, or disease–phenotype mappings

- In biomedicine, major mapping efforts included:
  - **Ontology of Biomedical Investigations (OBI)** importing across OBO
  - SNOMED CT, LOINC, and other clinical terminologies to research ontologies

History of Mapping: 2010 - Present

- With the explosion of biomedical data and knowledge graphs, mapping began leveraging **NLP**, **embeddings**, and **ML-based similarity**

- Strategies include:
  - **Vector-space alignment**: word embeddings (e.g., BioBERT, SciBERT) to propose mappings
  - **Graph embeddings**: aligning ontologies based on structural patterns
  - **Hybrid pipelines**: combining lexical, structural, and ML similarity scores, filtered by logical consistency checks

- Example: **BioPortal mappings**

Ambiguity

- Despite a cottage industry of research on this topic, the field is underdeveloped with few obvious successes

- This seems partly to do with deep ambiguity at the heart of the research program

- In short: “mapping” can mean many things to many people

SSSOM

- Simple Standard for Sharing Ontological Mappings is a lightweight, tabular format (usually TSV/CSV) for representing and exchanging ontology mappings

- Each row records a mapping between two entities, with fields for IDs, labels, mapping predicate (e.g., skos:exactMatch), and provenance (creator, method, confidence)

- Considered a "mapping"

Formal Equivalence

- Such mappings bury semantics in strings such as “narrow match” making them opaque to machines

- When I think of mappings being useful, it is insofar as they can be used in specific applications or as a foundation for code

- Which is to say, the semantics must be spelled out carefully

Generating Implicit Hierarchy

- owl:Thing
  - entity
    - continuant
      - generically dependent continuant
      - independent continuant
        - immaterial entity
          - continuant fiat boundary
          - site
          - spatial region
        - material entity
          - fiat object part
          - object
          - object aggregate
      - specifically dependent continuant

- owl:Thing
  - military artifact
    - ground vehicle
      - tank
      - operation area
    - watercraft
      - naval vessel aircraft
    - aircraft
      - flight path

Suppose there is a need to maintain an application ontology that is not aligned to BFO or CCO

Generating Implicit Hierarchy

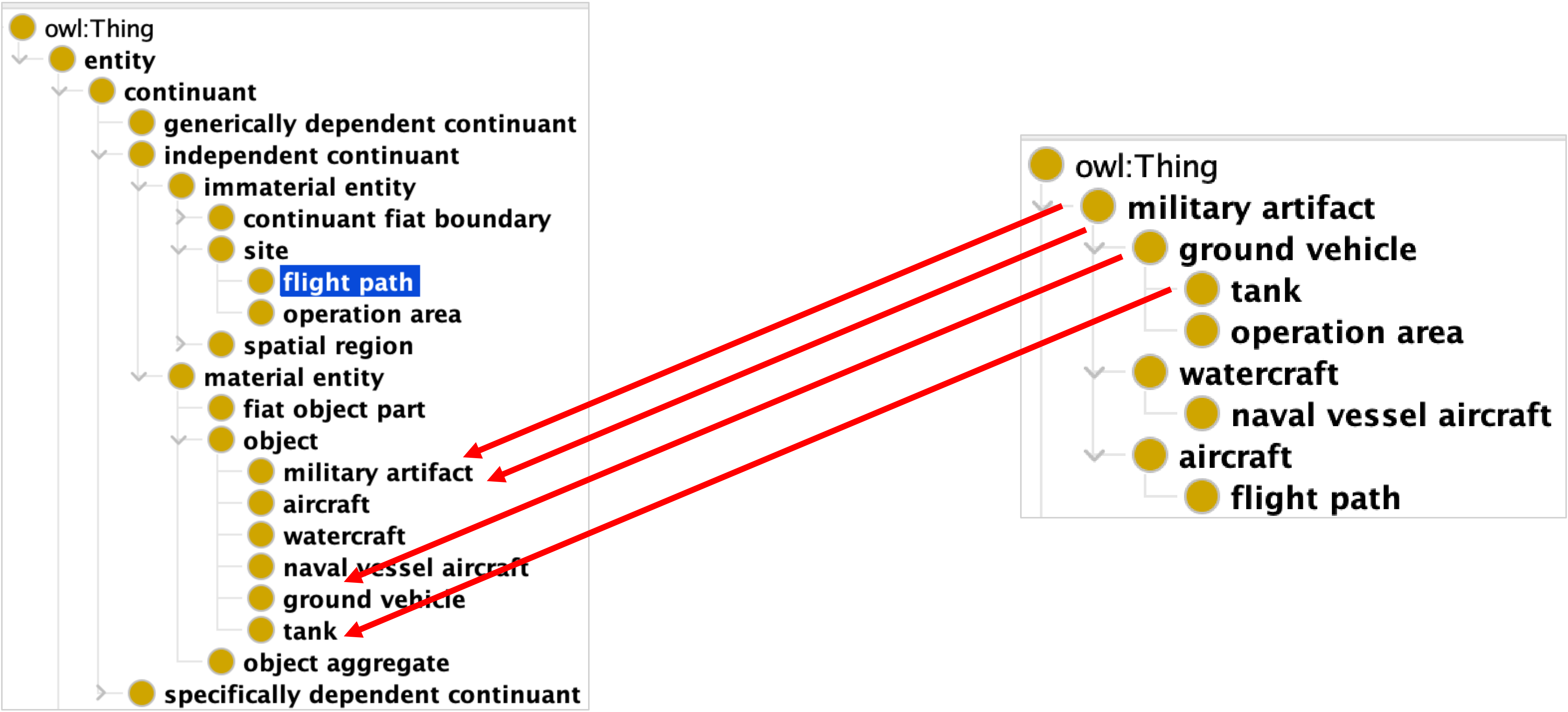
- owl:Thing
  - entity
    - continuant
      - generically dependent continuant
      - independent continuant
        - immaterial entity
          - continuant fiat boundary
          - site
          - spatial region
        - material entity
          - fiat object part
          - object
          - object aggregate
      - specifically dependent continuant

$\neq$

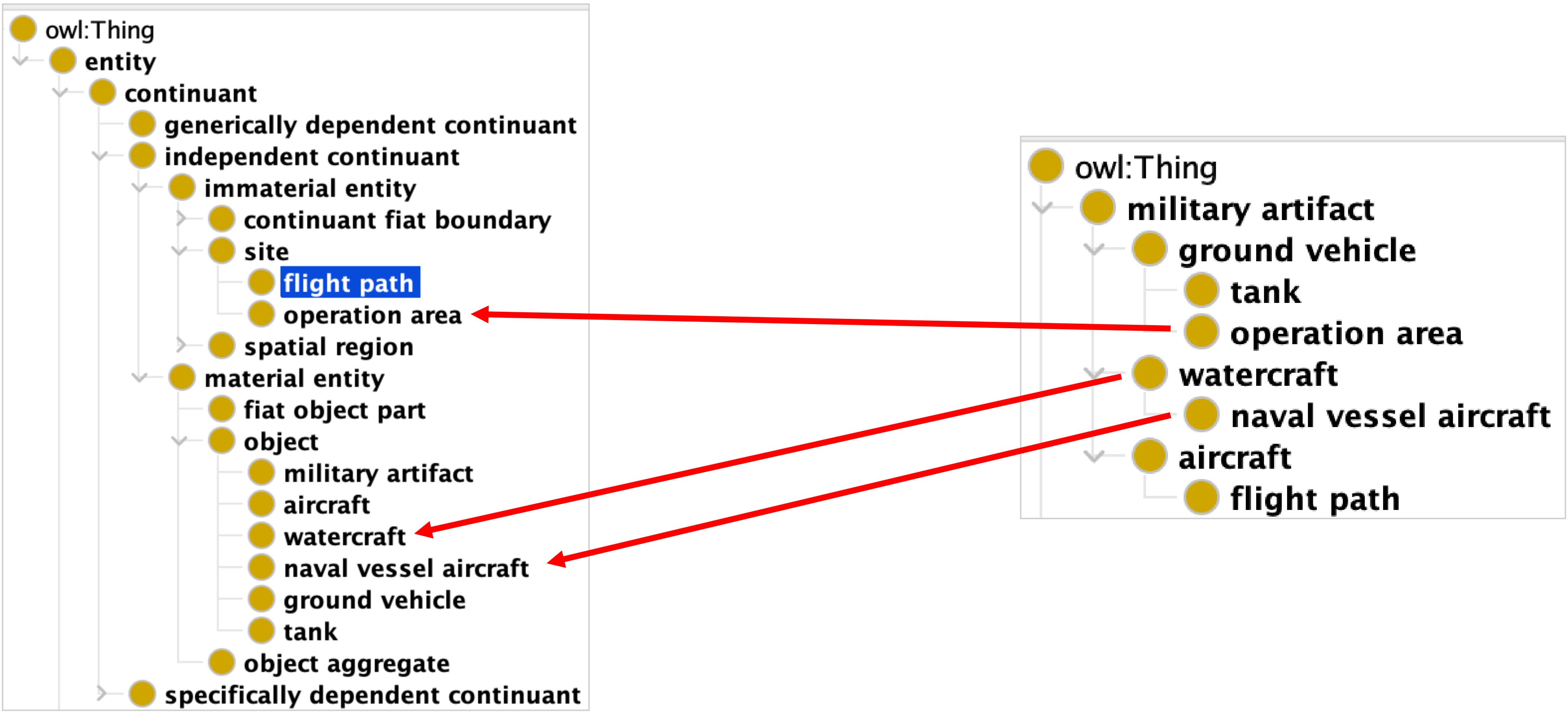
- owl:Thing
  - military artifact
    - ground vehicle
      - tank
      - operation area
    - watercraft
      - naval vessel aircraft
    - aircraft
      - flight path

Suppose there is a need to maintain an application ontology that is not aligned to BFO or CCO

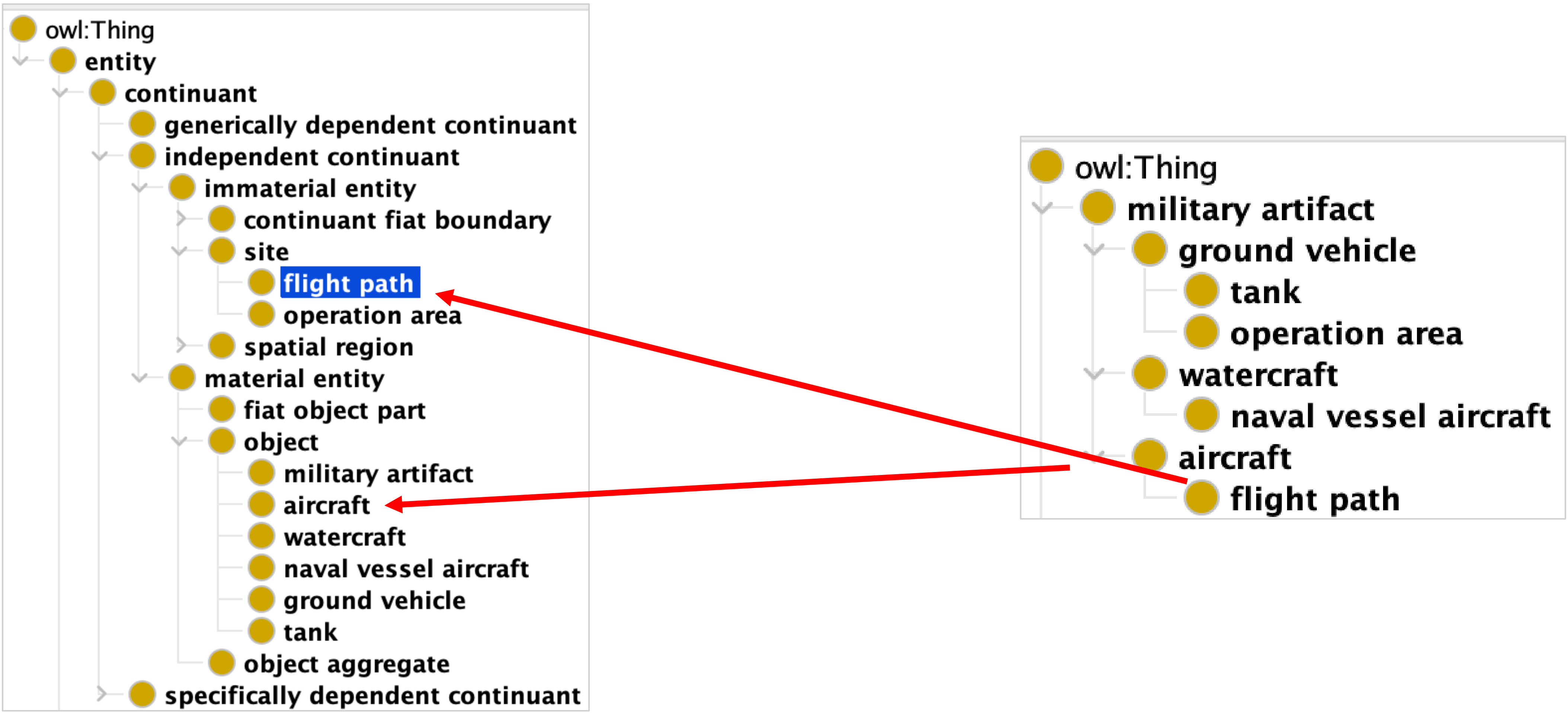
Generating Implicit Hierarchy



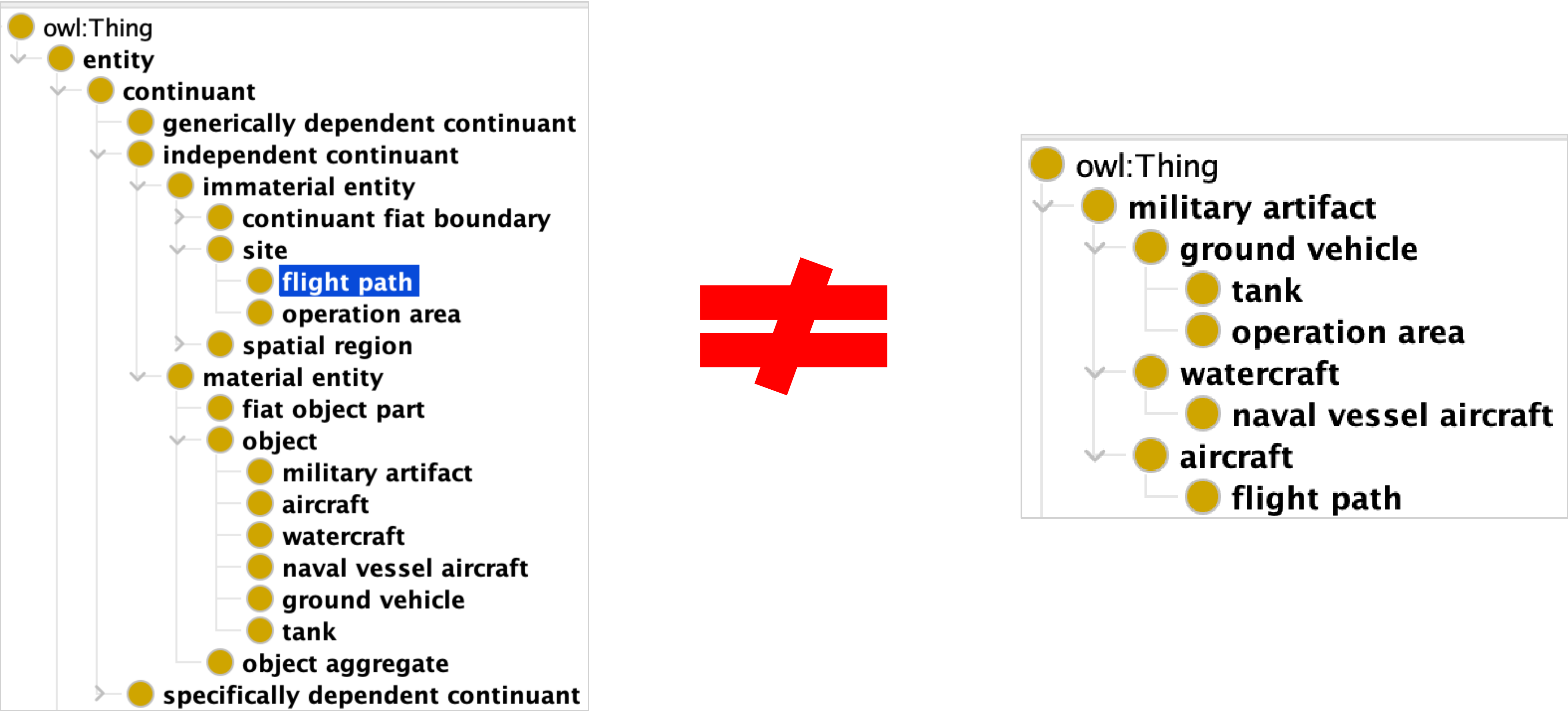
Generating Implicit Hierarchy



Generating Implicit Hierarchy



Generating Implicit Hierarchy



military taxonomy relation

- One way forward is to introduce an object property – call it **military taxonomy** – that connects instances of the reference ontology to those of the military artifact class

- To simulate the **subclass of** relation, we assert that **military taxonomy** is **reflexive** and **transitive**

- And is such that any entity in the domain can be related **only** to instances under military artifact

owl:allValuesFrom

- All instances that are related to **only** some C

THING

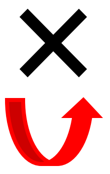
$$\{<x,y> \mid <x,y> \in R \text{ implies } y \in C\}$$

owl:ReflexivityProperty

- For all x, x is related to x

THING

$$\{\langle x,x\rangle \mid x \in \text{Domain}\} \subseteq R^I$$

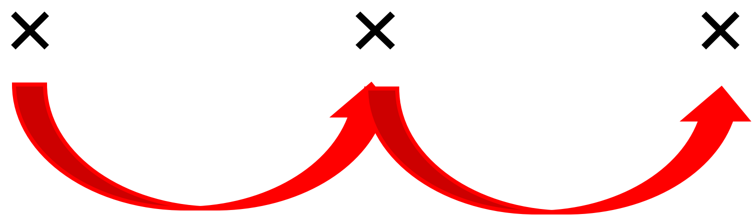


owl:TransitiveObjectProperty

- If x related to y and y related to z, then x related to z

THING

Trans(R) = $R^I \circ R^I \subseteq R^I$

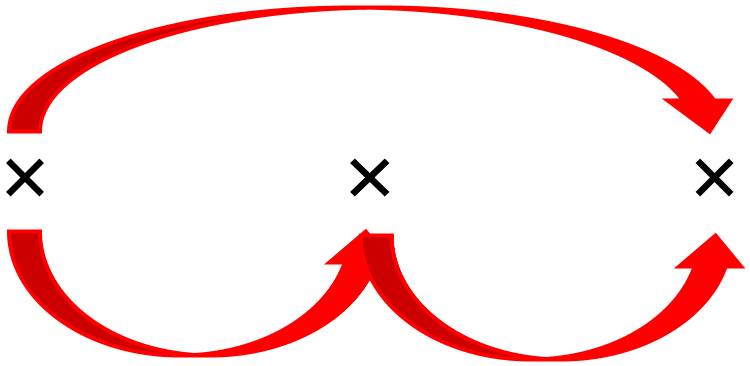


owl:TransitiveObjectProperty

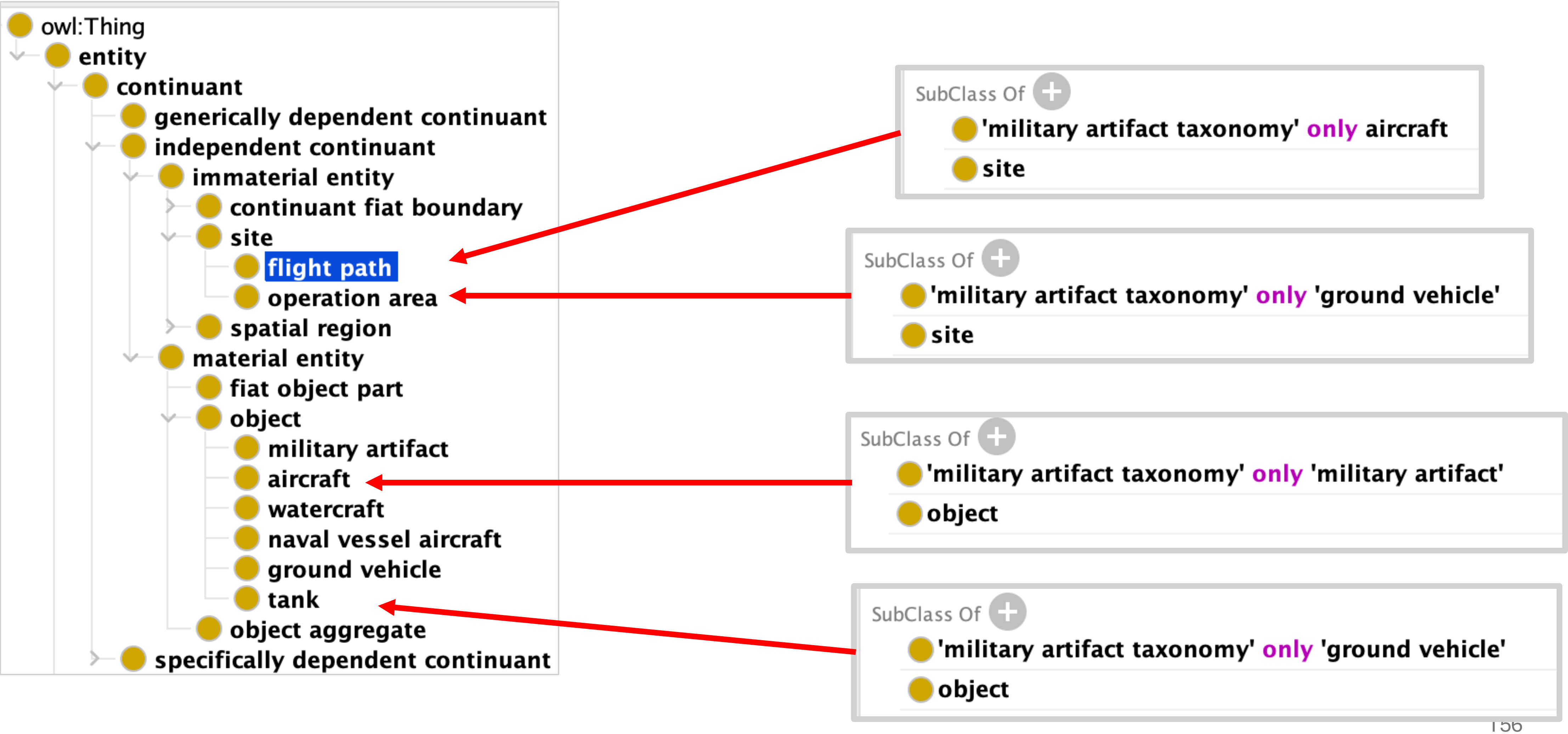
- If x related to y and y related to z, then x related to z

THING

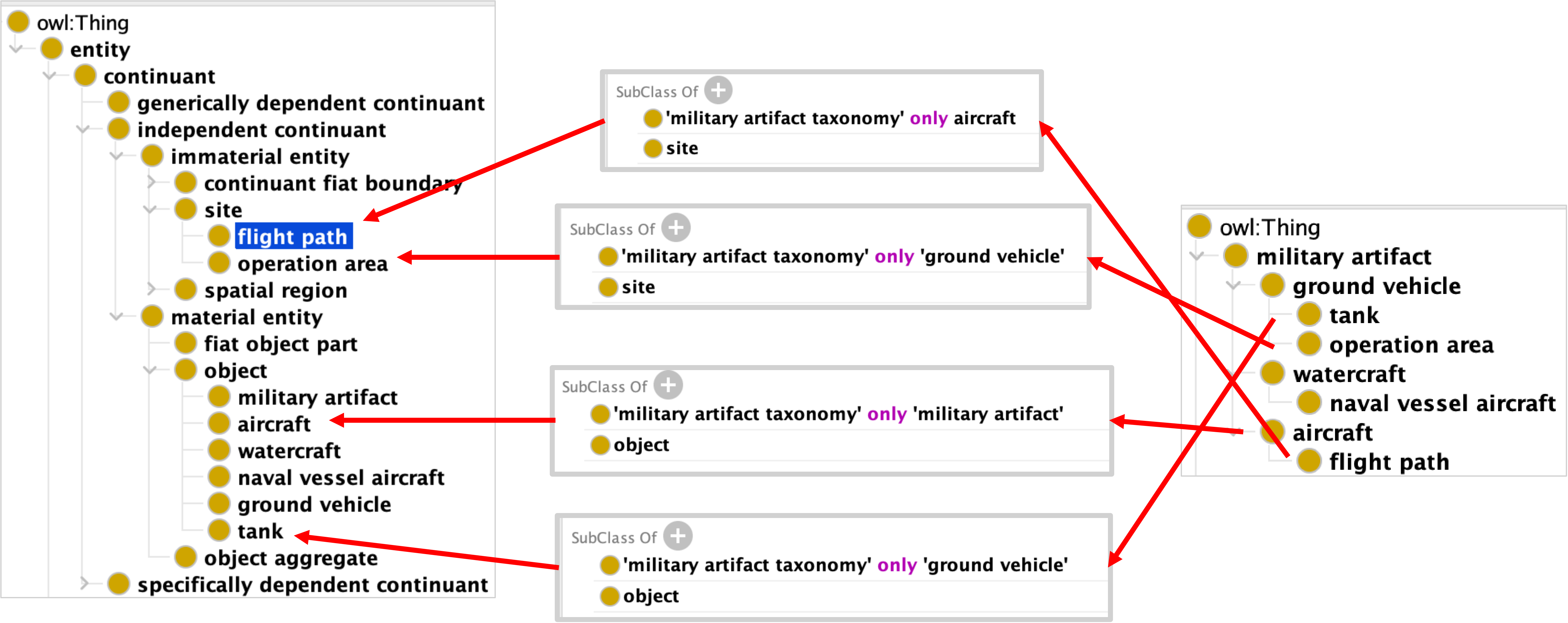
$$\mathbf{Trans(R) = R^I \circ R^I \subseteq R^I}$$



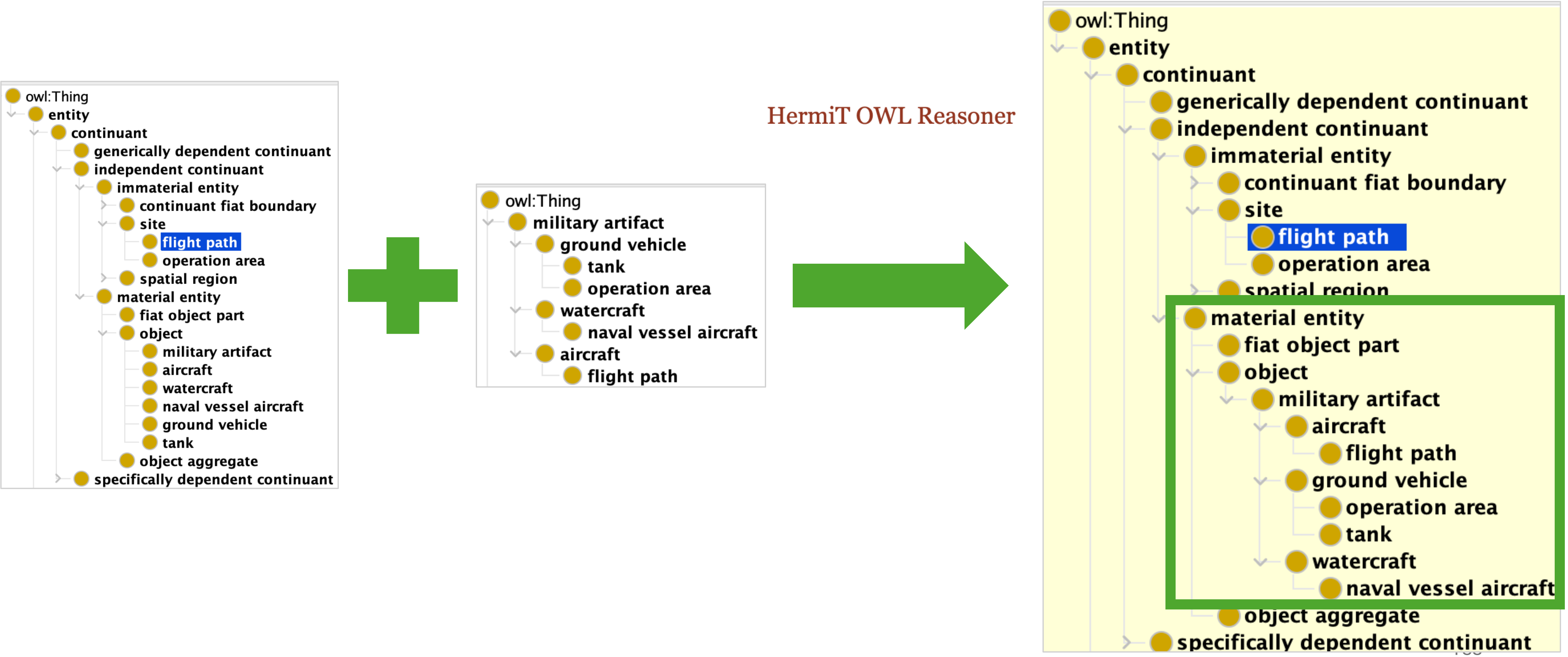
Generating Implicit Hierarchy



Generating Implicit Hierarchy



Generating Implicit Hierarchy

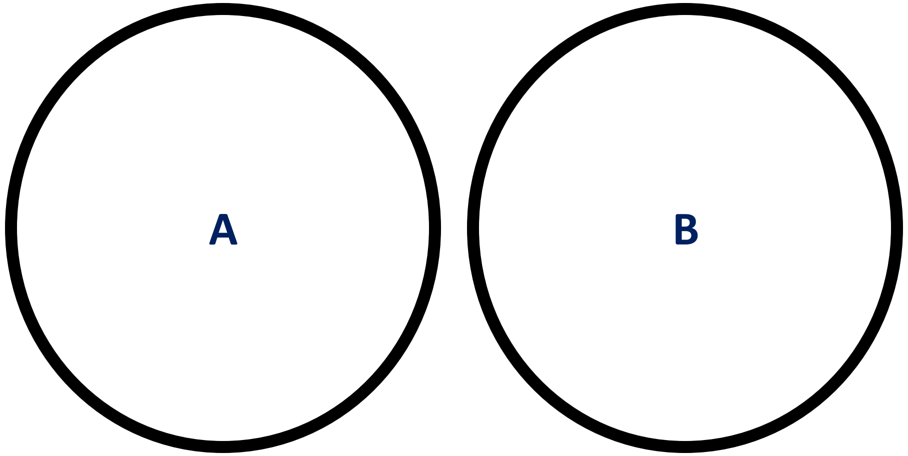


Caveat: Disjointness

- Disjointness – A and B are disjoint just in case they share no individuals

THING

DisjointWith(A, B) = $A^I \cap B^I = \emptyset$



Caveat: Disjointness Must be Dropped

- In BFO, the class **site** and the class **object** are **disjoint**, which means they may share no instances in common

- Consequently, **operation area** cannot – strictly speaking – be an asserted subclass of **object** and inferred subclass of **site**

- Importantly, such constraints should be understood as applying at the level of reference ontologies, **not necessarily** application ontologies

Mapping Files

- Mapping files need not abide by the principles governing ontology design, insofar as those principles undermine mission goals

- When principles are not adhered to owing to application needs, keep the new application ontology distinct from relevant reference ontologies and **annotate deviations from reference ontology in detail**

ANNOTATIONS

Semantic Alignment

- The Web Ontology Language (OWL) is based on a description logic; ontologies can be checked for semantic similarity using **bisumulation techniques**
- **Bisimulation** allows one to identify whether one ontology is more or less expressive than another
- Examples can be suggestive but fall short of proof; it is challenging to prove a negative, i.e. "You cannot define XYZ in ontology A"

Bisimulation

- ϱ is a *bisimulation* if and only if:

  i. $d_1 \varrho d_2$ implies $d_1 \in A^{I1}$ if and only if:
     $d_2 \in A^{I2}$ for all $d_1 \in \Delta^{I1}$, $d_2 \in \Delta^{I2}$ and $A \in C$
  i. $d_1 \varrho d_2$ and $(d_1, d'_1) \in r^{I1}$ implies the existence of $d'_2 \in \Delta^{I2}$ such that:
     $d'_1 \varrho d'_2$ and $(d_2, d'_2) \in r^{I2}$ for all $d_1, d'_1 \in \Delta^{I1}$, $d_2 \in \Delta^{I2}$ and $r \in R$
  ii. $d_1 \varrho d_2$ and $(d_2, d'_2) \in r^{I2}$ implies the existence of $d'1 \in \Delta^{I1}$ such that:
     $d'_1 \varrho d'_2$ and $(d_1, d'_1) \in r^{I1}$ for all $d_1 \in \Delta^{I1}$, $d_2, d'_2 \in \Delta^{I2}$ and $r \in R$

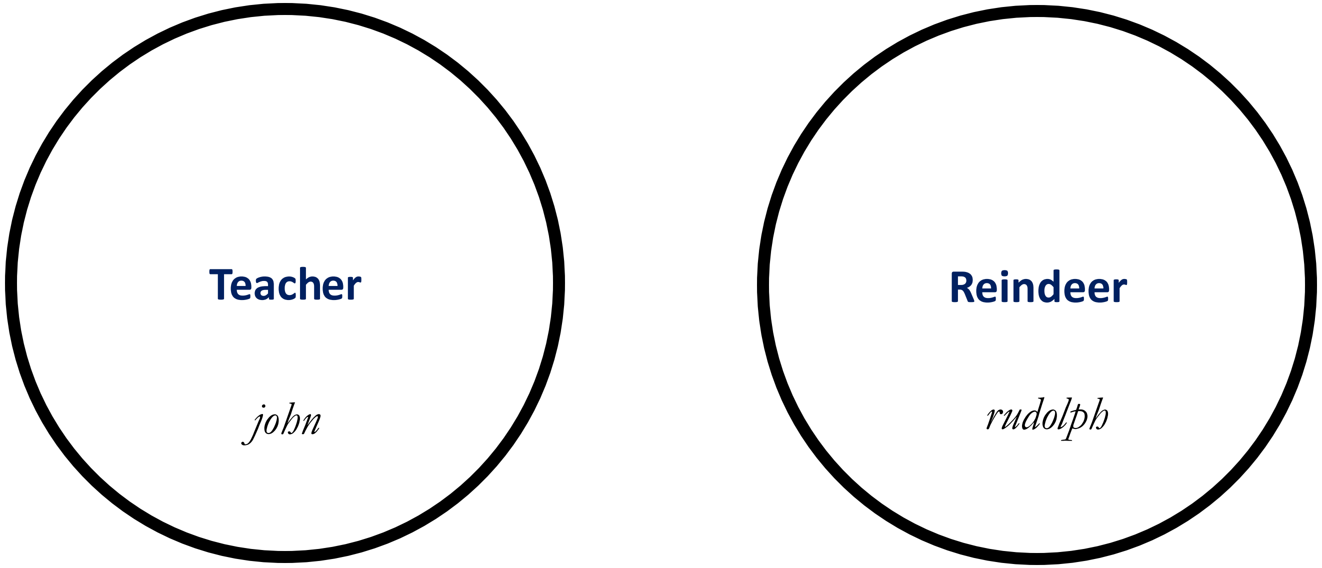
ALC Syntax

Signature = $\{\top, \bot, \sqcup, \sqcap, \neg, \exists, \forall, r_{1\ldots n}, C_{1\ldots n}\}$

ALC Syntax

$$\text{Signature} = \{\top, \bot, \sqcup, \sqcap, \neg, \exists, \forall, r_{1\ldots n}, C_{1\ldots n}\}$$

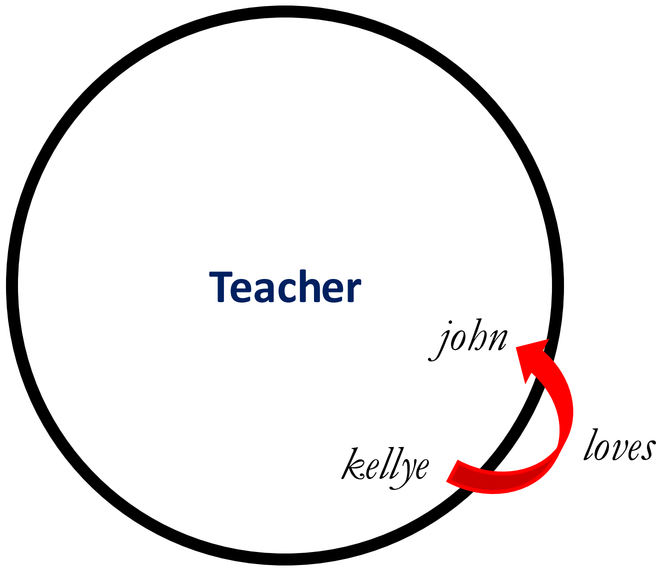
- $C_{1\ldots n}$ - Correspond to classes, such as Teacher, Reindeer, etc. which are often assumed to be collections of similar enough instances in the world, such as John or Rudolph



ALC Syntax

$$\text{Signature} = \{\top, \bot, \sqcup, \sqcap, \neg, \exists, \forall, r_{1\ldots n}, C_{1\ldots n}\}$$

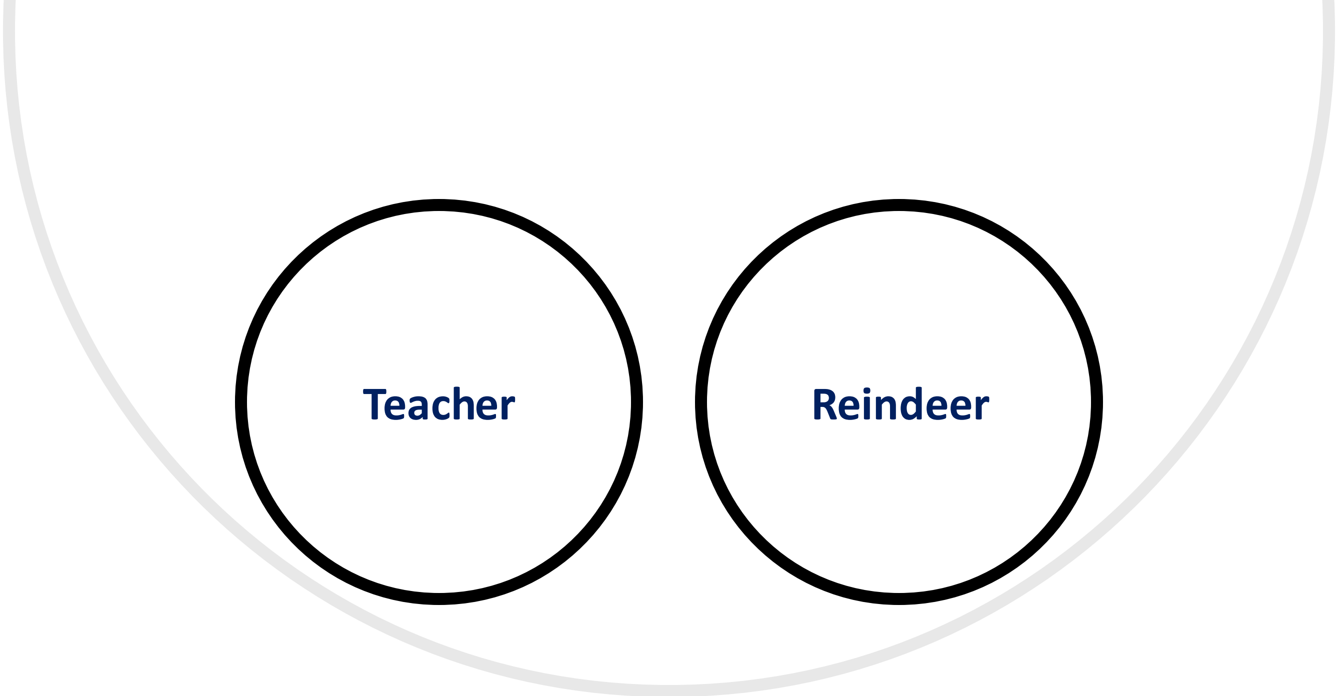
- $r_{1\ldots n}$ - Corresponds to relations holding between instances such as loves or parent of or next to



ALC Syntax

$$\text{Signature} = \{\top, \bot, \sqcup, \sqcap, \neg, \exists, \forall, r_{1\ldots n}, C_{1\ldots n}\}$$

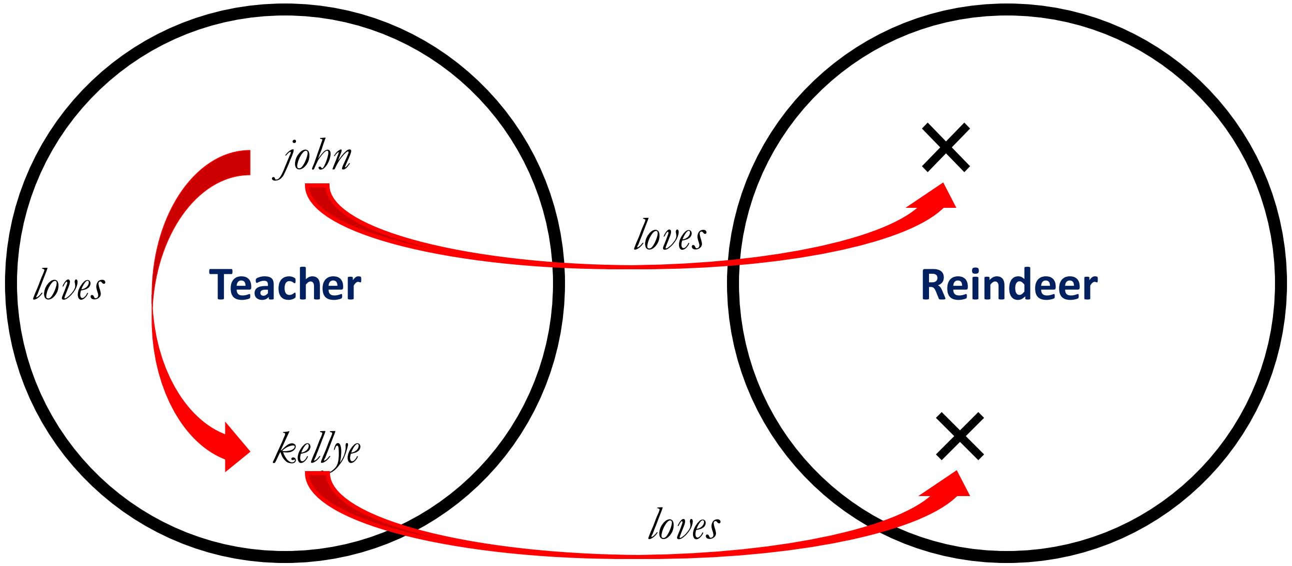
- $\top$ - Corresponds to everything in the domain; in practice this is used to represent the most general class, i.e. the ultimate parent class of every other class



ALC Syntax

$$\text{Signature} = \{\top, \bot, \sqcup, \sqcap, \neg, \exists, \forall, r_{1\ldots n}, C_{1\ldots n}\}$$

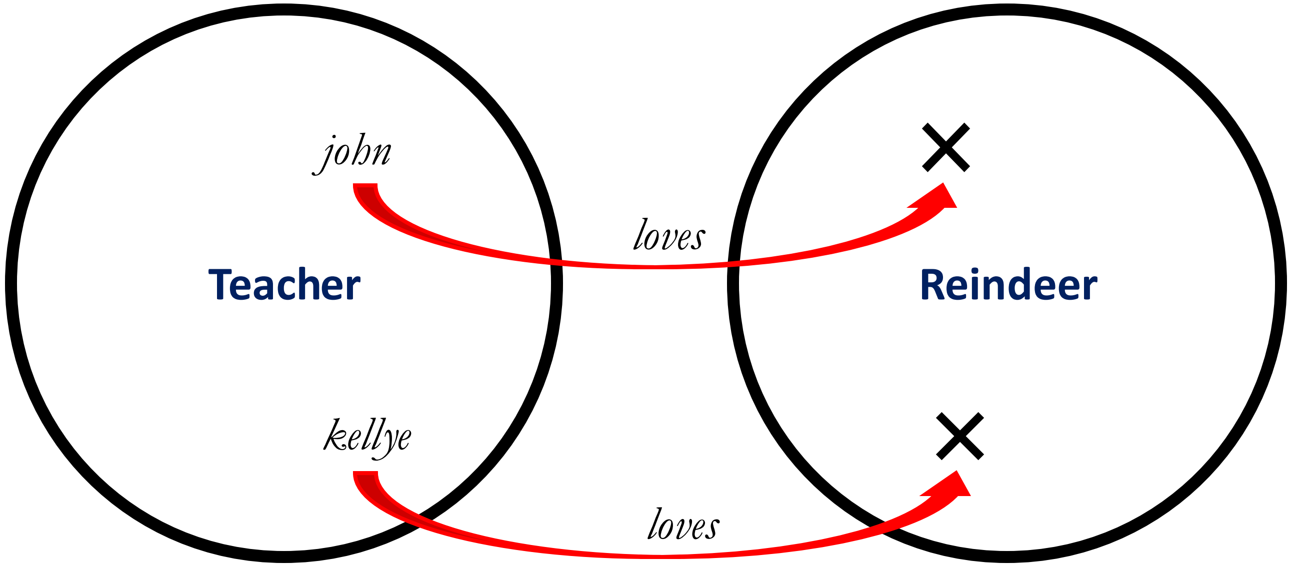
- $\exists$r.C – Corresponds to all instances that are related to some C.



ALC Syntax

$$\text{Signature} = \{\top, \bot, \sqcup, \sqcap, \neg, \exists, \forall, r_{1\ldots n}, C_{1\ldots n}\}$$

- $\forall$r.C – Corresponds to all instances that are related to only C.



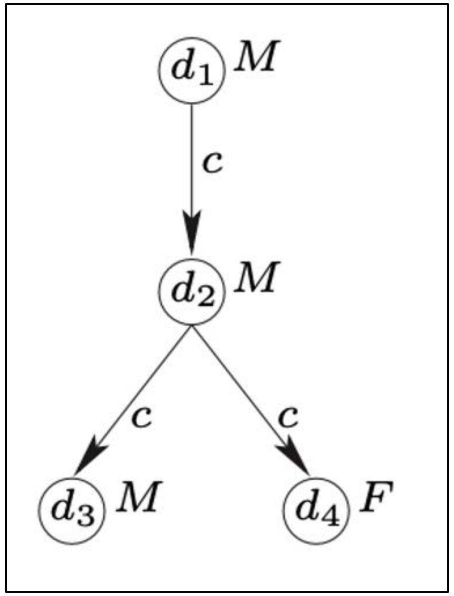
Expressivity

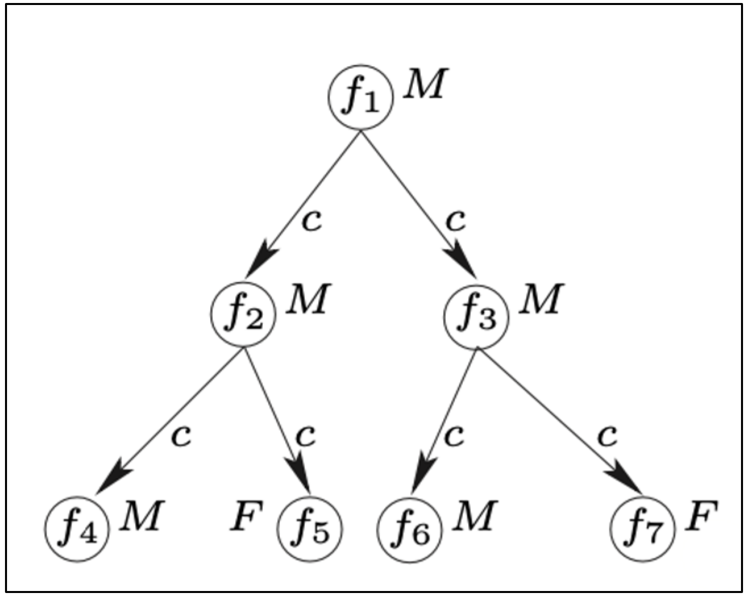
- The language of ALC is not expressive enough to distinguish between bisimilar elements.

- Bisimulation is a relationship between interpretations/elements; interpretations are distinct from syntaxes

- For example, bisimulations between interpretations are distinct from the syntax of ALC

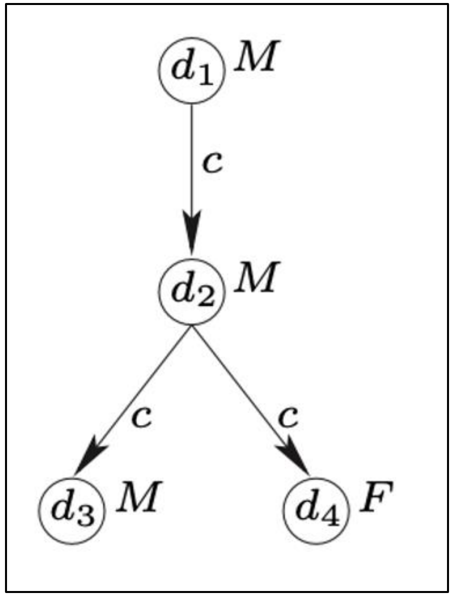
Expressivity

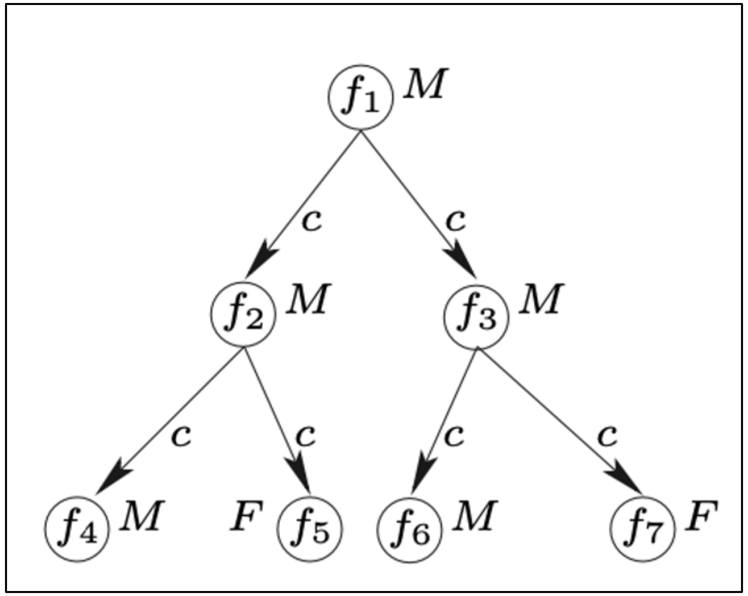




Expressivity

$$\exists c.(M \sqcap \exists c.M \sqcap \exists c.F)$$

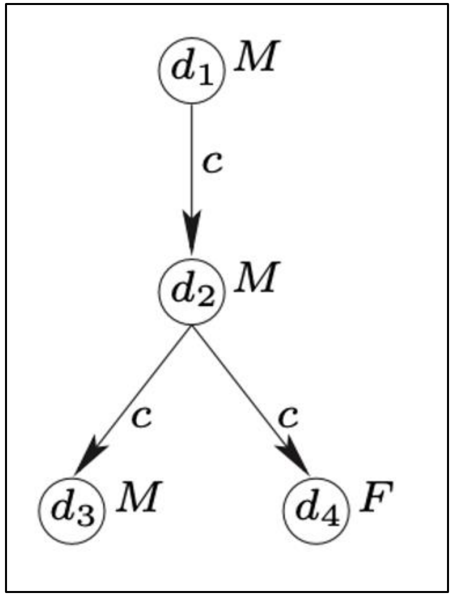


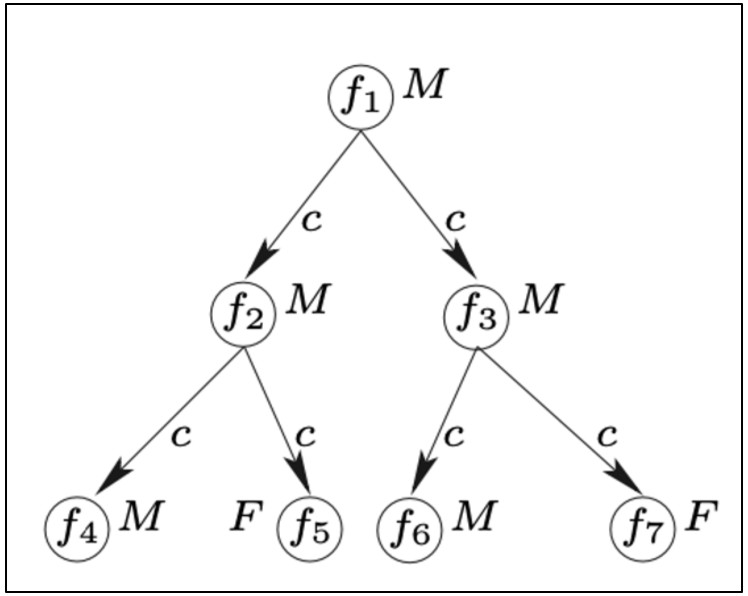


Expressivity

∃c.(M ⊓ ∃c.M ⊓ ∃c.F)

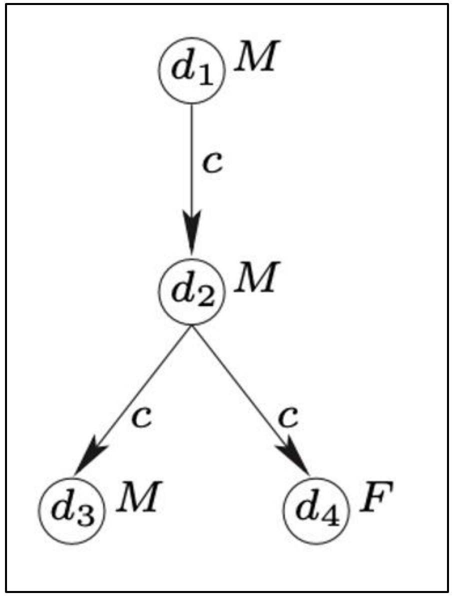
∀x∃y(c(x,y) & M(y) & ∃z(c(y,z) & M(z)) & ∃u(c(y,u) & F(u)))

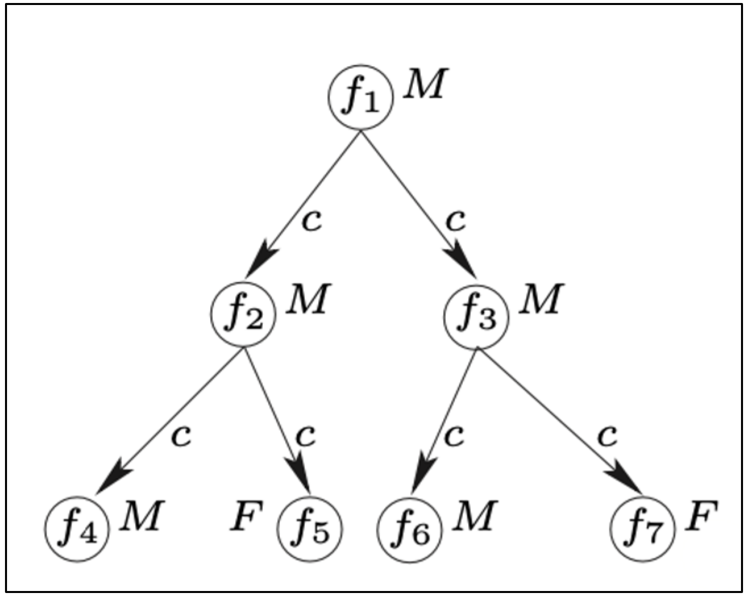




Expressivity

Any x that has *at least one* son y who has *at least one* son z and *at least one* one daughter u

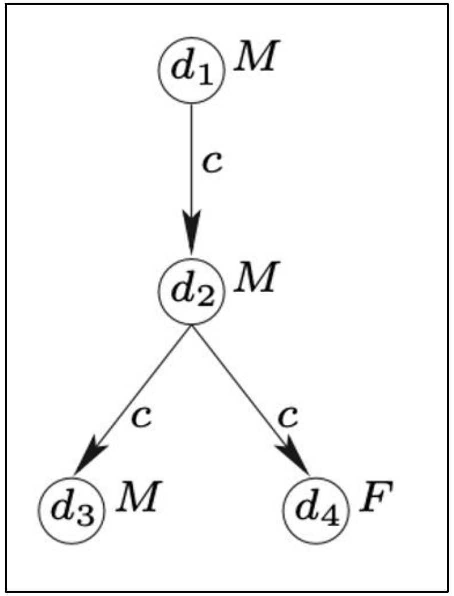


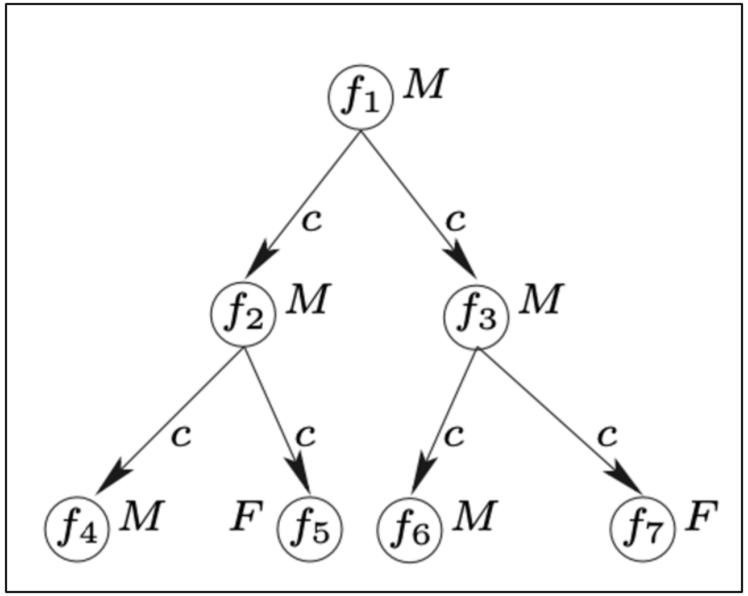


Expressivity

Any x that has *at least one* son y who has *at least one* son z and *at least one* one daughter u

Both graphs satisfy the ALC expression: $\exists c.(M \sqcap \exists c.M \sqcap \exists c.F)$





ALC Extensions: ALCI (inverses)

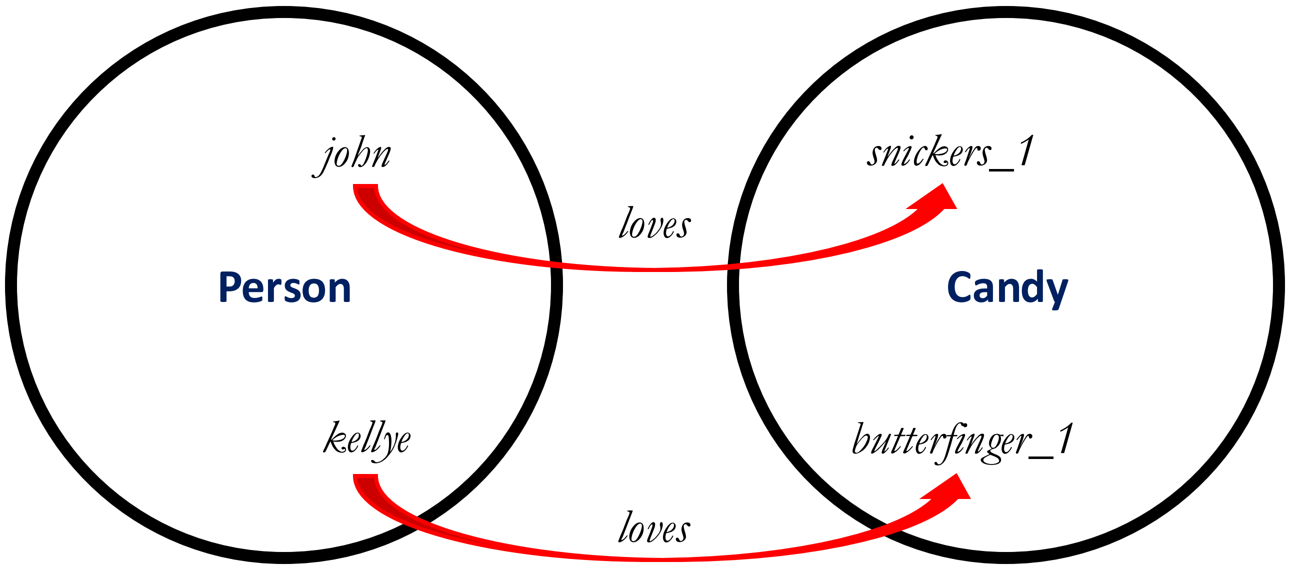
ALCI Signature = ALC Signature + $\{r^{-}_{1\ldots n}\}$

- $r^{-}_{1\ldots n}$ – Corresponds to inversions of relations such as r between instances, such as the inverse of 'loves' being 'loves⁻', i.e. 'loved by'

ALC Extensions: ALCI (inverses)

ALCI Signature = ALC Signature + $\{r^-_{1\ldots n}\}$

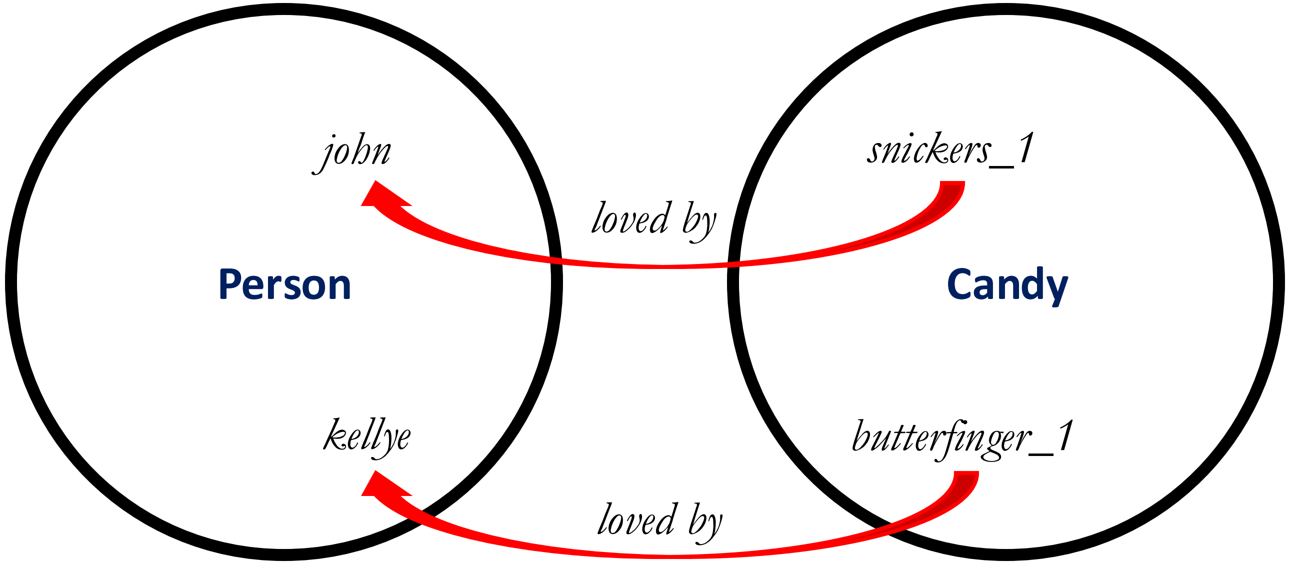
- $r^-_{1\ldots n}$ – Corresponds to inversions of relations such as r between instances, such as the inverse of 'loves' being 'loves$^-$', i.e. 'loved by'



ALC Extensions: ALCI (inverses)

ALCI Signature = ALC Signature + $\{r^{-}_{1\ldots n}\}$

- $r^{-}_{1\ldots n}$ – Corresponds to inversions of relations such as r between instances, such as the inverse of 'loves' being 'loves⁻', i.e. 'loved by'



Expressivity

- There is an *ALCI* concept C such that C≠D holds for all ALC concepts D.

- ALCI adds only "$\exists r^{-}.\top$" to the syntax of ALC. To prove ALCI is more expressive than ALC, we must show there is no expression in ALC that is equivalent to $\exists r^{-}.\top$

- Suppose there is such an expression in ALC – call it D - we will show this assumption leads to contradiction.

ALCI > ALC

- Consider d_2 and e_2 in the following diagram:

- There is a bisimulation between them, so $d_2 \in D^{I1}$ just in case $e_2 \in D^{I2}$

ALCI > ALC

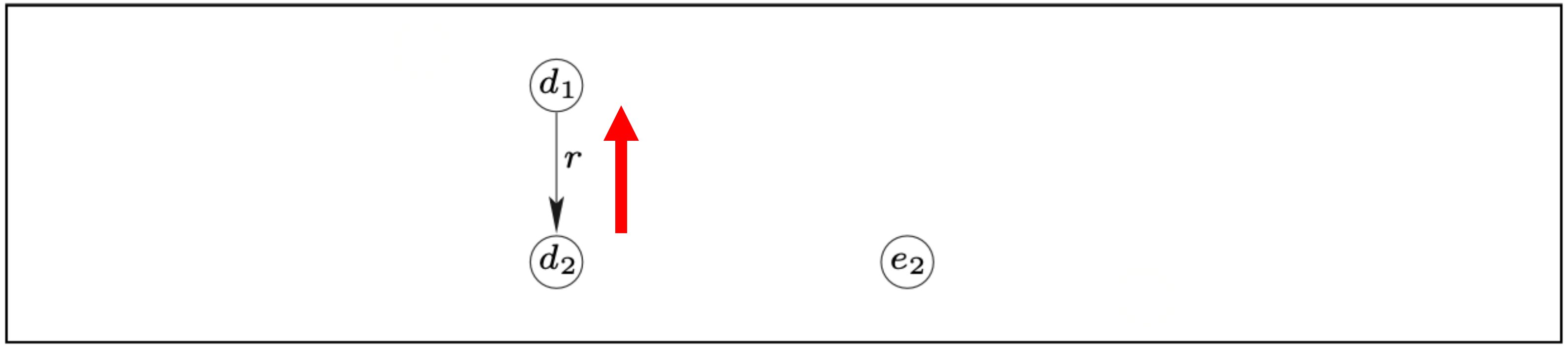
- Consider d_2 and e_2 in the following diagram:

- There is a bisimulation between them, so $d_2 \in D^{I1}$ just in case $e_2 \in D^{I2}$

- However, $d_2 \in (\exists r^{-}.\top)^{I1}$

ALCI > ALC

- Consider d_2 and e_2 in the following diagram:

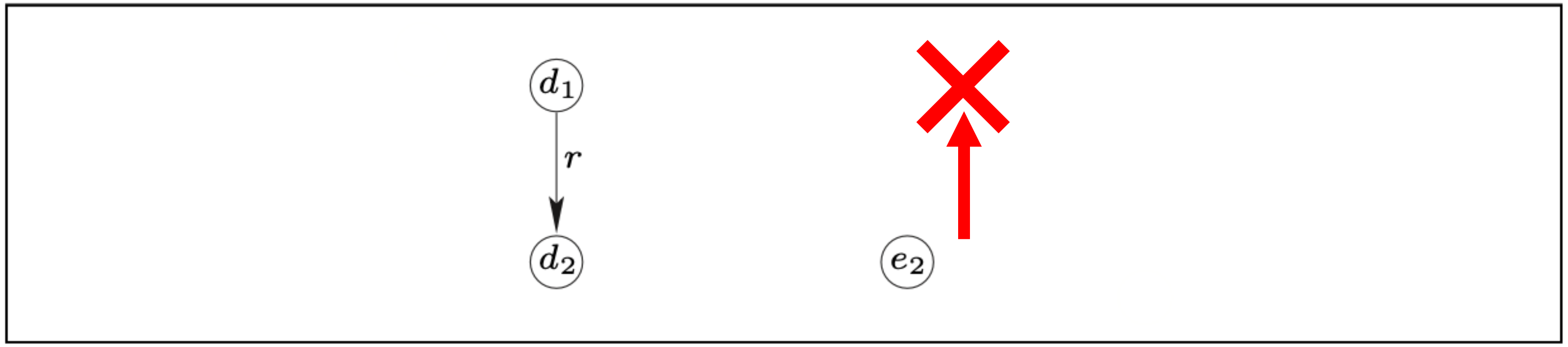


- There is a bisimulation between them, so $d_2 \in D^{I1}$ just in case $e_2 \in D^{I2}$

- However, $d_2 \in (\exists r-.\top)^{I1}$

ALCI > ALC

- Consider d_2 and e_2 in the following diagram:



- There is a bisimulation between them, so $d_2 \in D^{I1}$ just in case $e_2 \in D^{I2}$

- However, $d_2 \in (\exists r^{-}.\top)^{I1}$ and $e_2 \notin (\exists r^{-}.\top)^{I2}$

ALCI > ALC

- Consider d_2 and e_2 in the following diagram:

- That is, the ALCI expression $\exists r^-.\top$ can be satisfied by d_2 in the left graph but not e_2 in the right, since the latter lacks any role to have an inverse

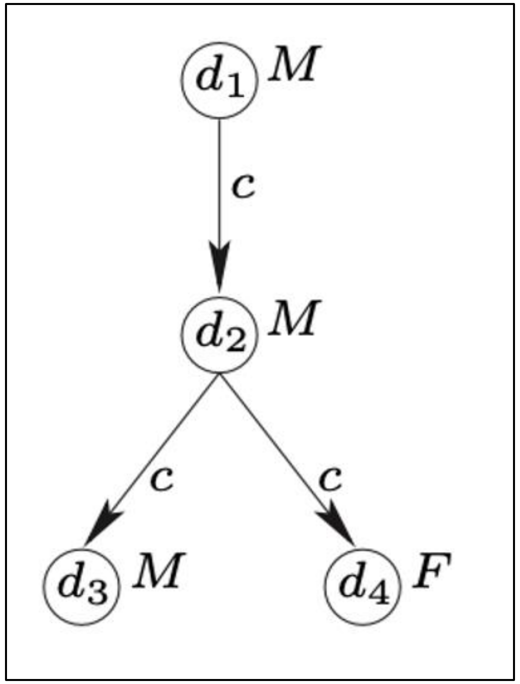
- Because there is a bisimulation between d_2 and e_2 and d_2 satisfies $\exists r^-.\top$ but e_2 doesn't, *ALCI can distinguish between bisimilar graphs that ALC cannot*

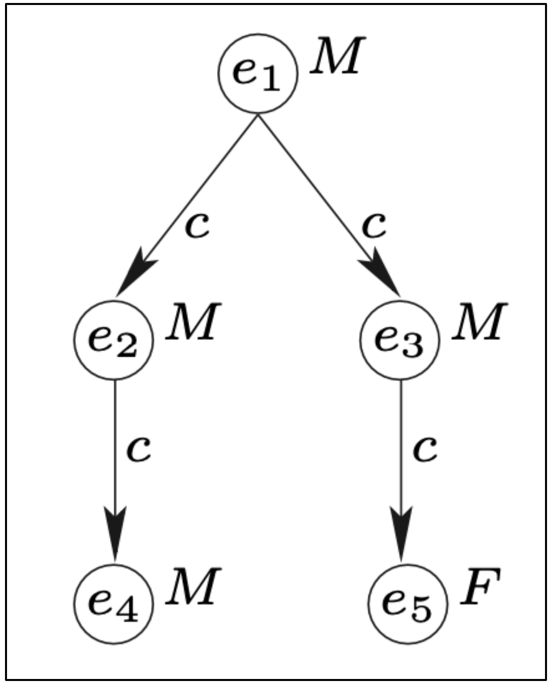
Expressivity to Semantic Similarity

- Much like bisimulation can be leveraged to demonstrate for any concept description C there is no concept description D that is equal to C

- Bisimulation can be leveraged to demonstrate for any ontology element X there is no other ontology element Y that is equal to X

- Bisimulation can thus be a useful strategy for determining when **two ontologies are not aligned**





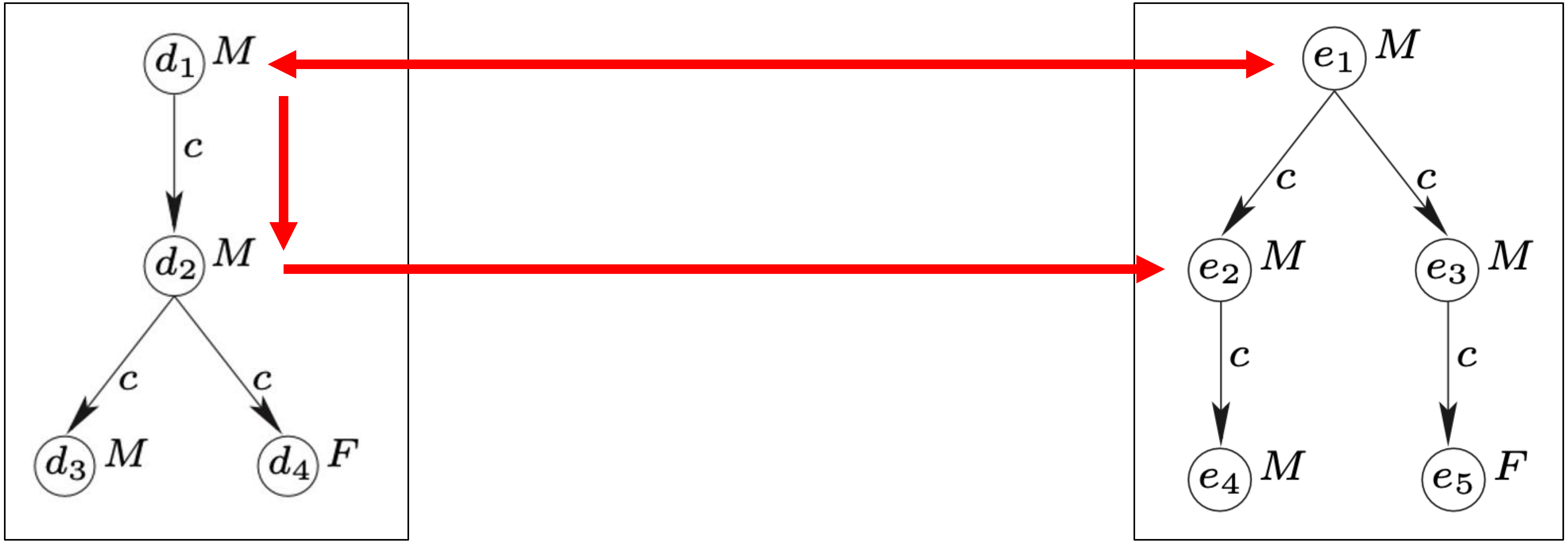
- **Claim:** d_1 is not bisimilar to e_1

- **Claim:** d_1 is not bisimilar to e_1
- **Invariance**: d_1 and e_1 are both instances of M

- **Claim:** d_1 is not bisimilar to e_1

- **Zig:**

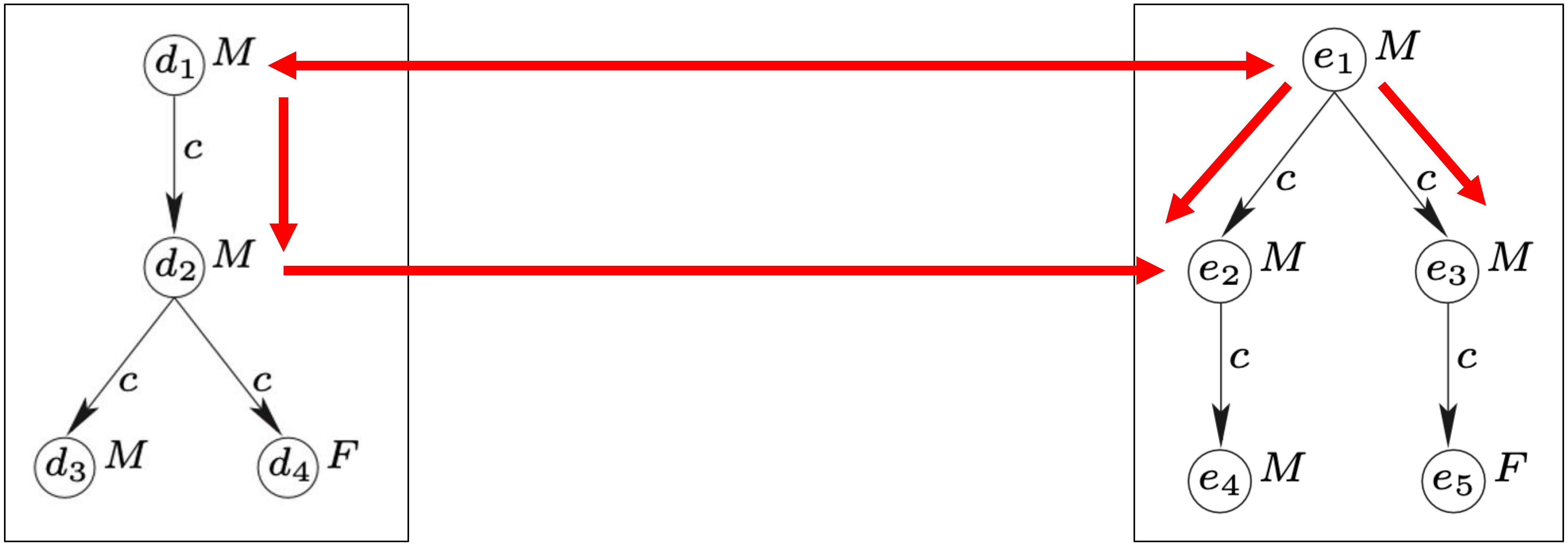
  If role c relates d_1 to d_2



- **Claim:** d_1 is not bisimilar to e_1

- **Zig:**

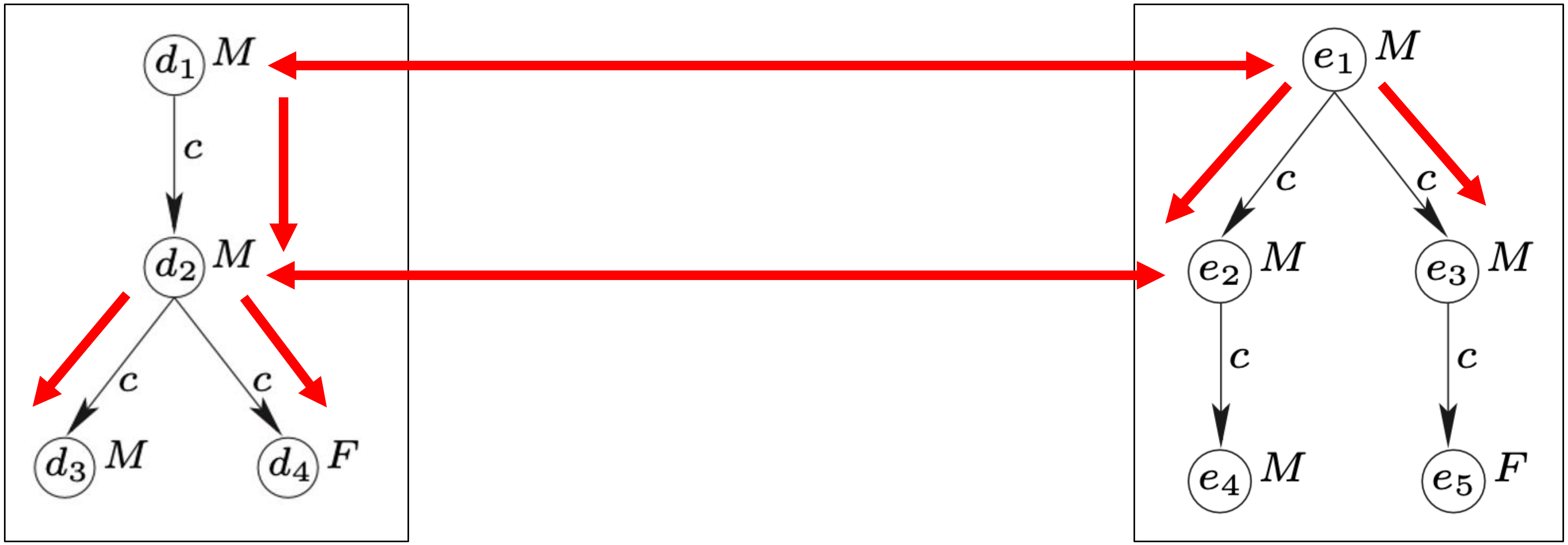
  If role c relates d_1 to d_2 then there is a mapping from d_2 to e_2 where d_2 and e_2 are both instances of M and from d_2 to e_3 where d_2 and e_3 are both instances of M



- **Claim:** d_1 is not bisimilar to e_1

- **Zig:**

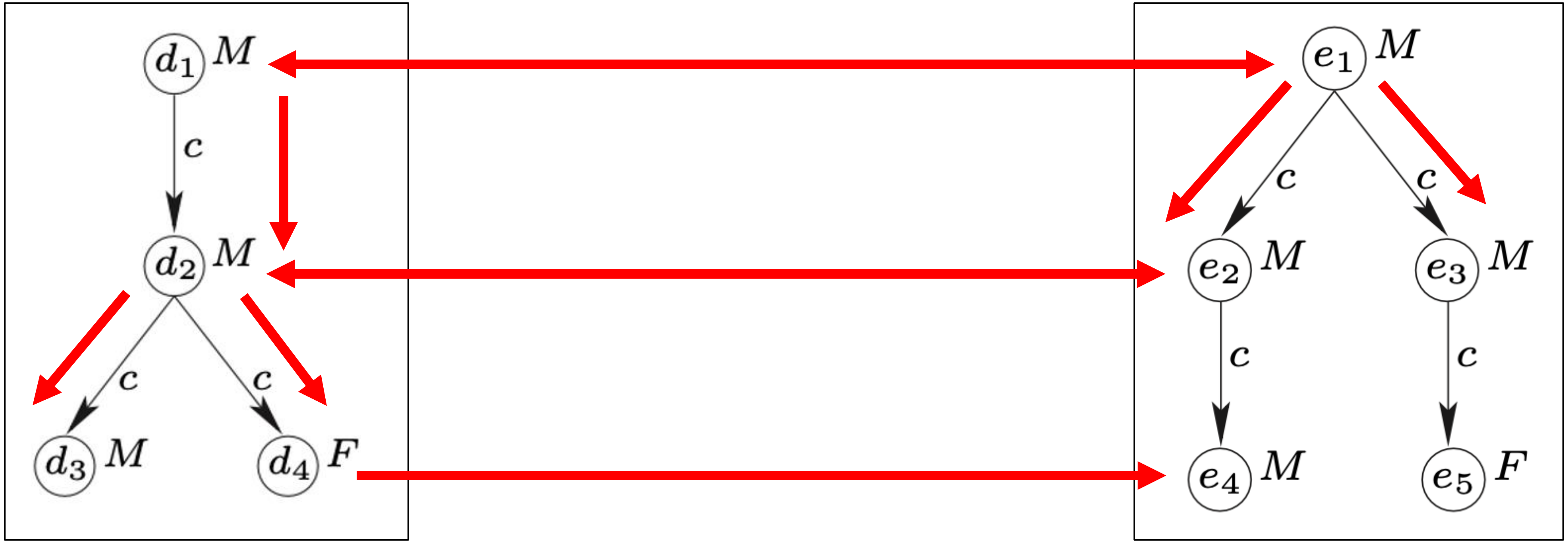
  If role c relates d_1 to d_2 then there is a mapping from d_2 to e_2 where d_2 and e_2 are both instances of M and from d_2 to e_3 where d_2 and e_3 are both instances of M and role c maps e_1 to e_2 and e_1 to e_3



- **Claim:** d_1 is not bisimilar to e_1

- **Zag:**

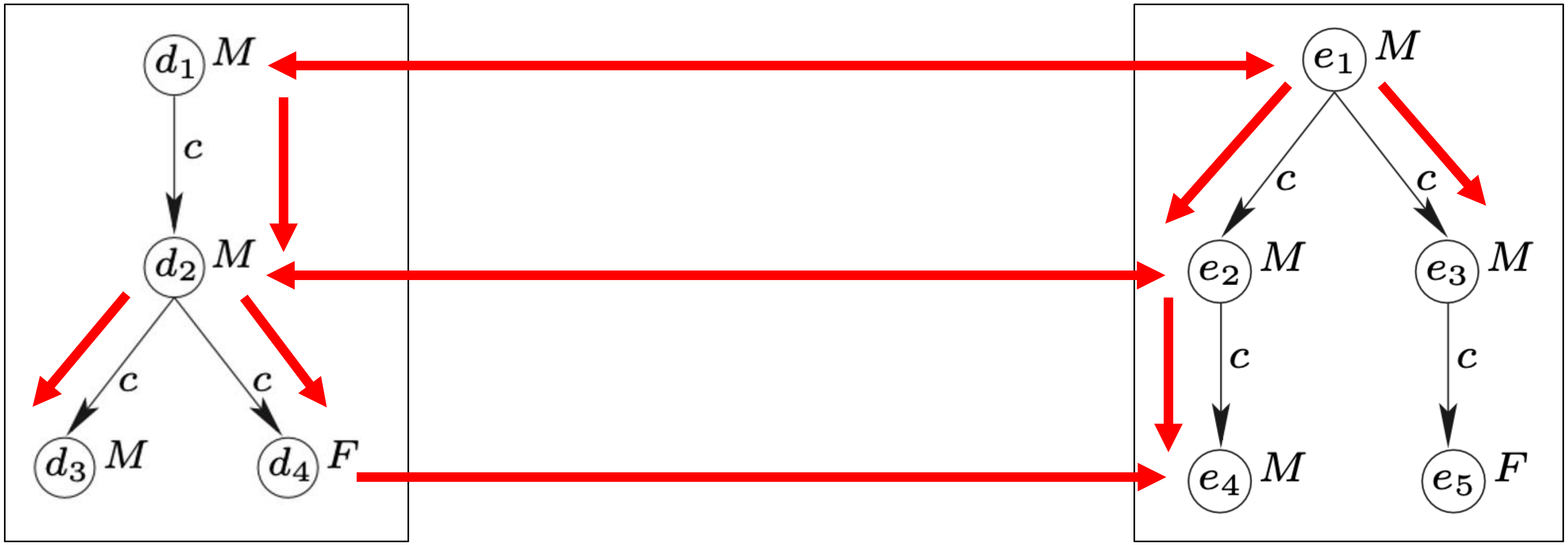
  If role c relates d_2 to d_3 and d_4



- **Claim:** d_1 is not bisimilar to e_1

- **Zag:**

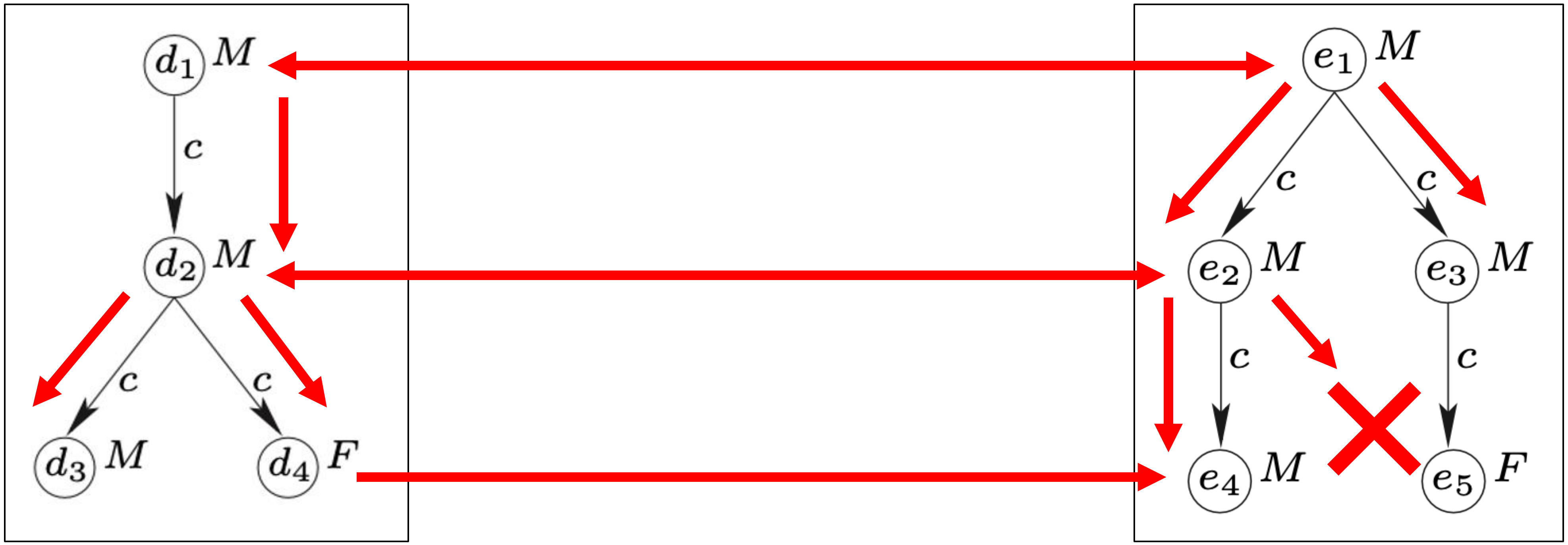
  If role c relates d_2 to d_3 and d_4 then there is a mapping from d_3 to e_4 and from d_3 to e_5 such that d_3 and e_4 are instances of M and e_5 is an instance of F



- **Claim:** d_1 is not bisimilar to e_1

- **Zag:**

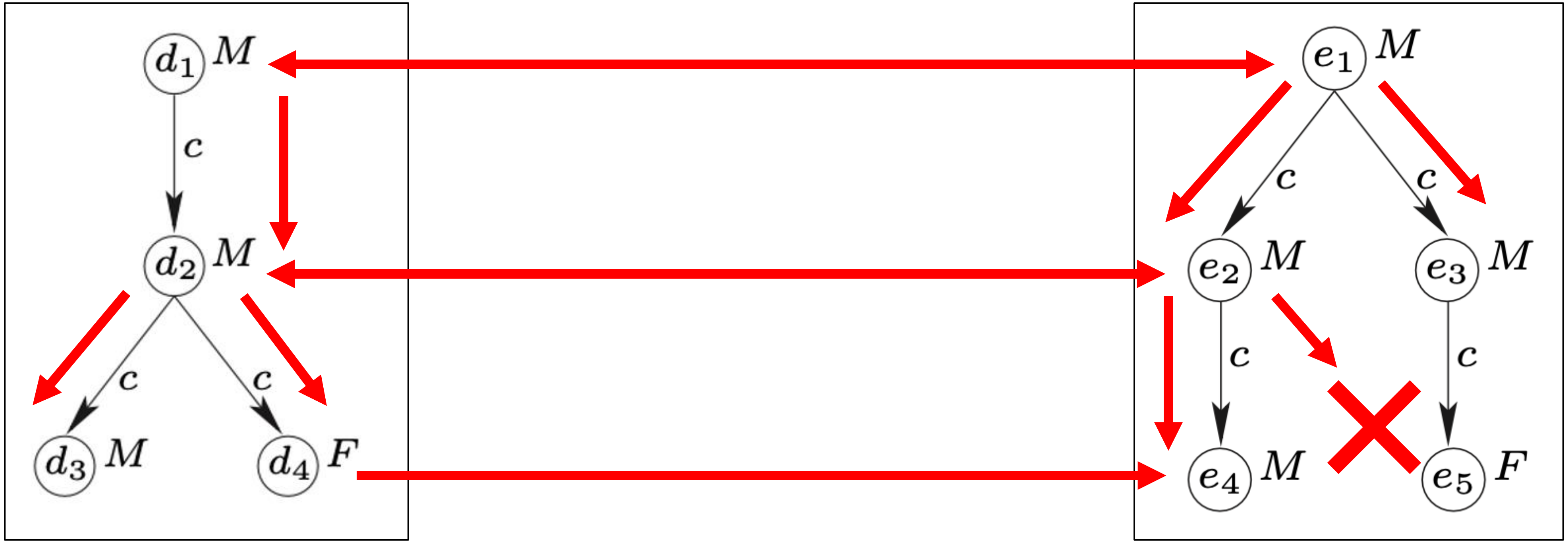
  If role c relates d_2 to d_3 and d_4 then there is a mapping from d_3 to e_4 and from d_3 to e_5 such that d_3 and e_4 are instances of M and e_5 is an instance of F and c relates e_2 to e_4



- **Claim:** d_1 is not bisimilar to e_1

- **Zag:**

  If role c relates d_2 to d_3 and d_4 then there is a mapping from d_3 to e_4 and from d_3 to e_5 such that d_3 and e_4 are instances of M and e_5 is an instance of F and c relates e_2 to e_4 and c relates e_2 to e_5



- **Claim:** d_1 is not bisimilar to e_1

- **Zag:**

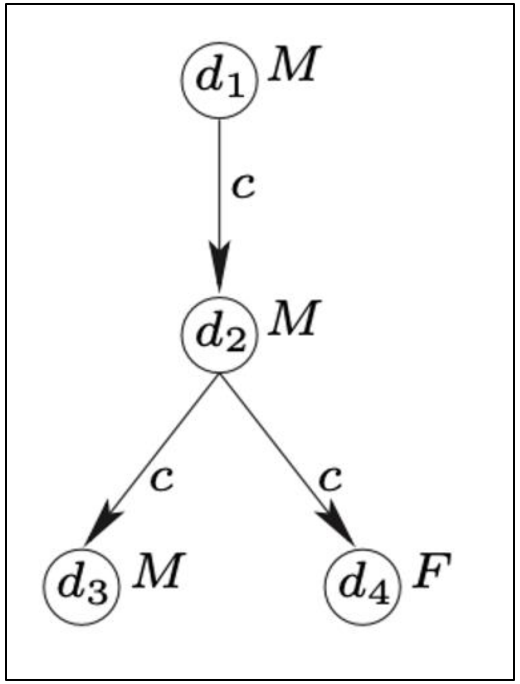
  If role c relates d_2 to d_3 and d_4 then there is a mapping from d_3 to e_4 and from d_3 to e_5 such that d_3 and e_4 are instances of M and e_5 is an instance of F and c relates e_2 to e_4 and c relates e_2 to e_5

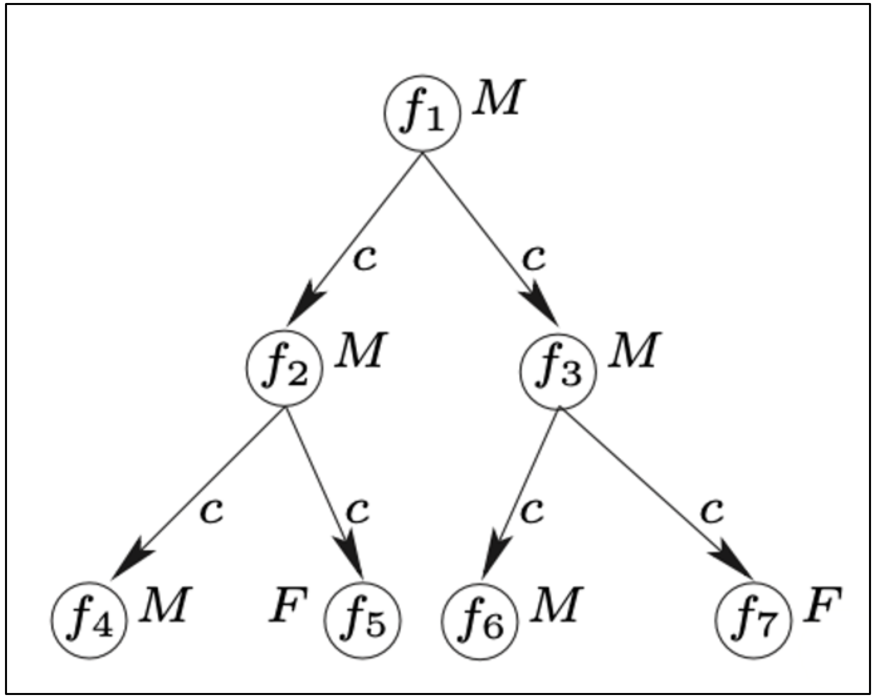
Expressivity to Semantic Similarity

- Bisimulation may also be leveraged to determine when **two ontology elements are aligned**, insofar as they are bisimilar

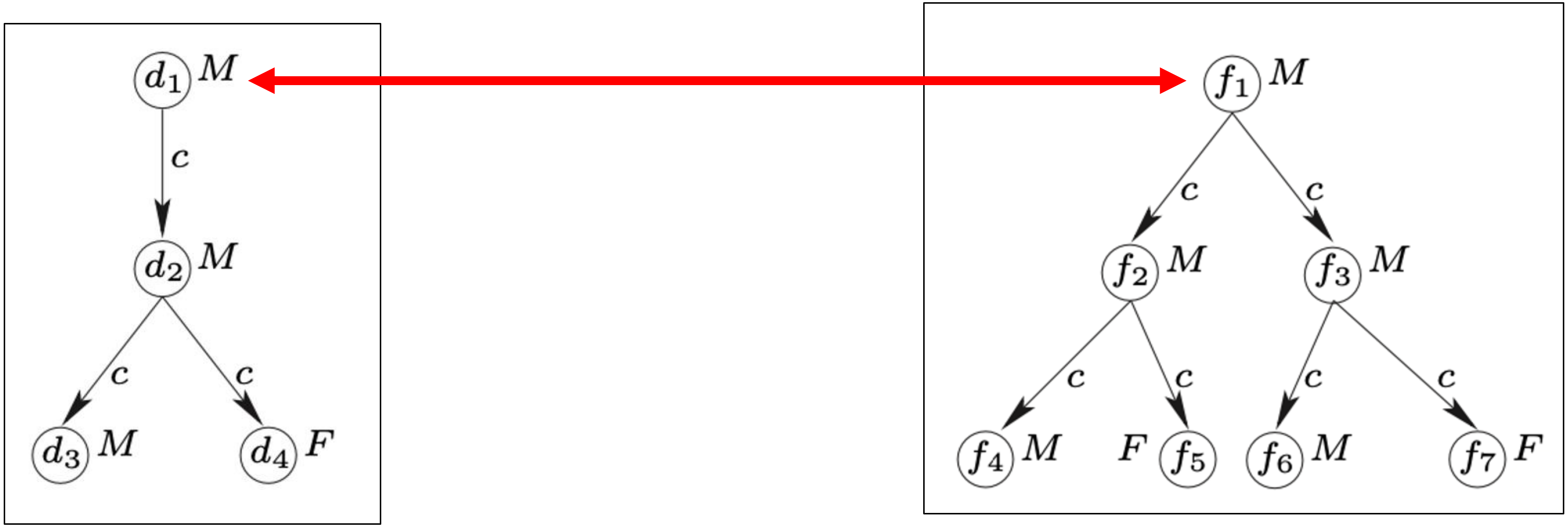
- Such a result may **provide evidence** that the two ontology elements capture the same intended semantics

- On the other hand, such a result may suggest **further work is needed** to capture the intended semantics of the ontology elements



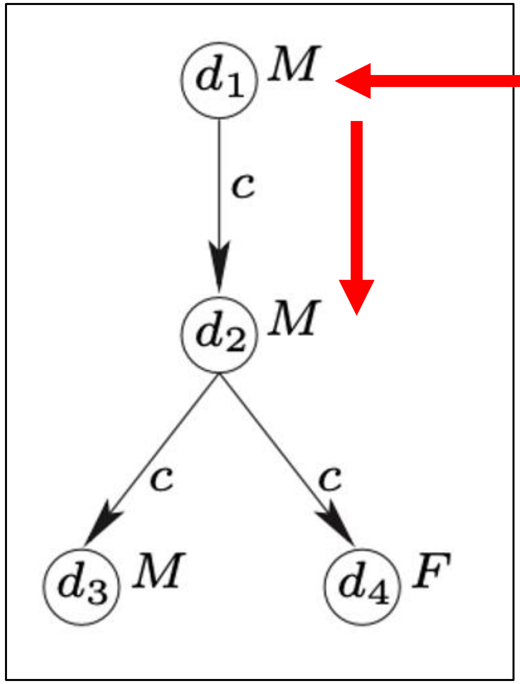


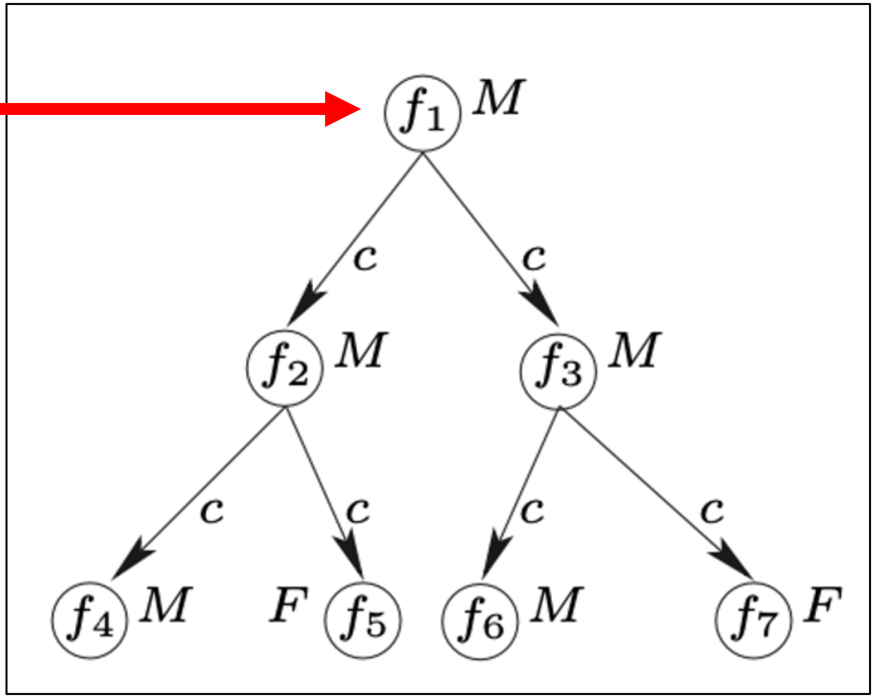
- **Claim:** d_1 is bisimilar to f_1



- **Claim:** d_1 is bisimilar to f_1

- **Invariance:** d_1 and f_1 are both instances of M

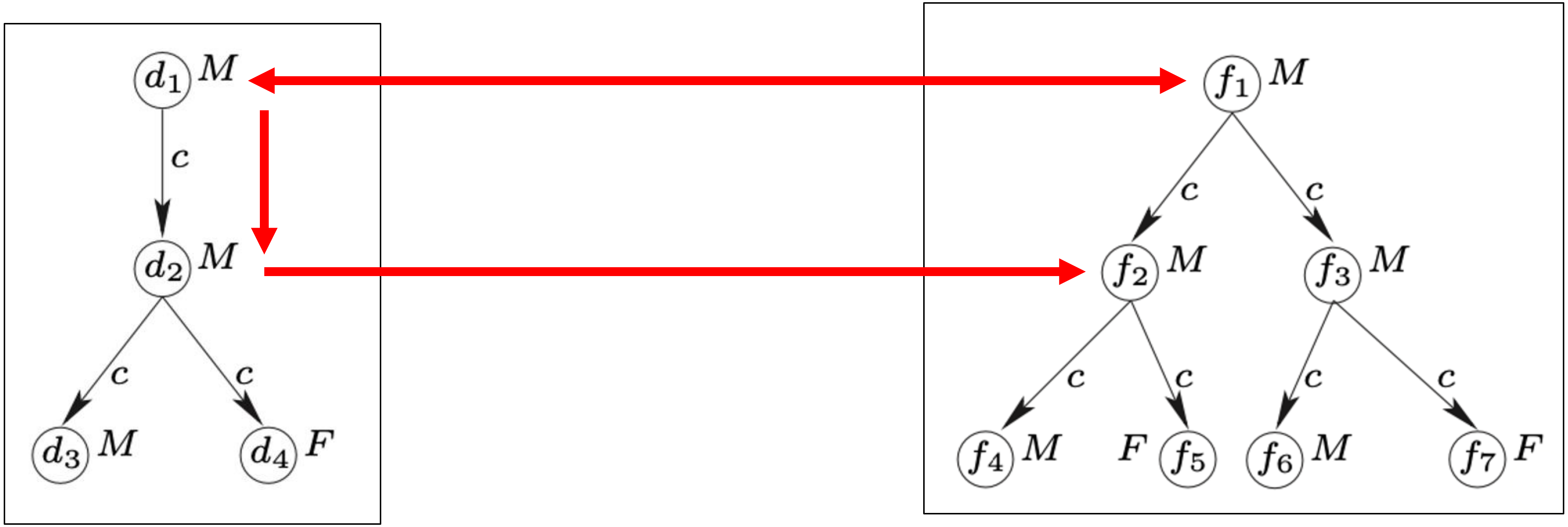




- **Claim:** d_1 is bisimilar to f_1

- **Zig:**

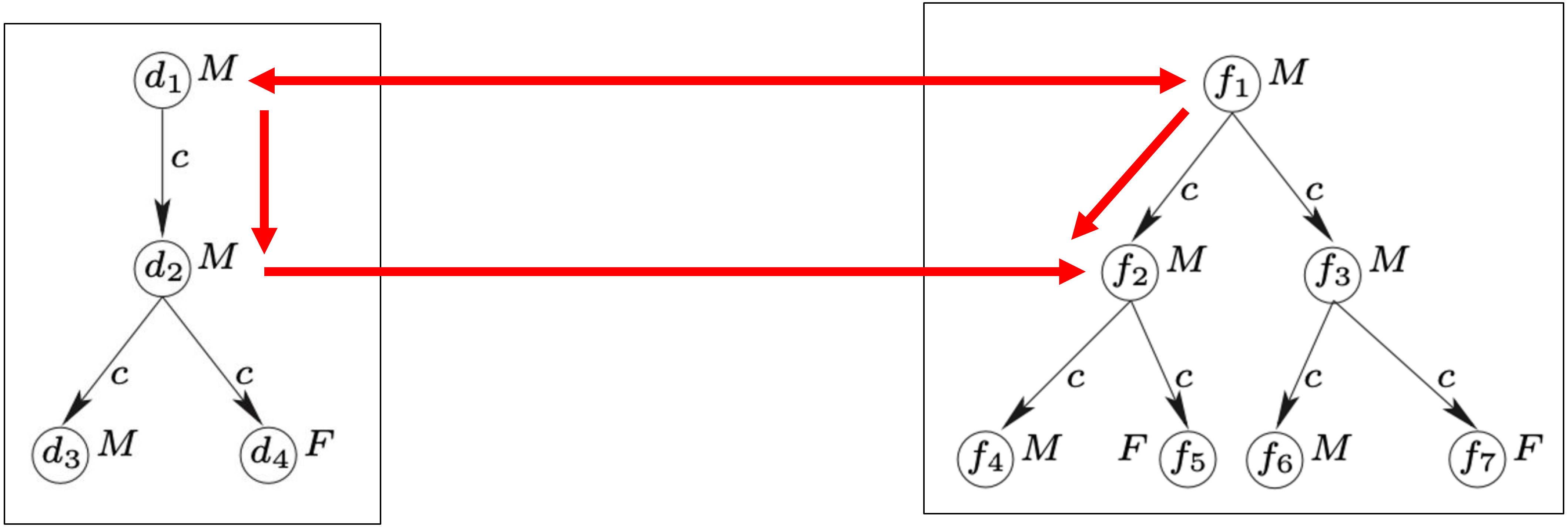
  If role c relates d_1 to d_2



- **Claim:** d_1 is bisimilar to f_1

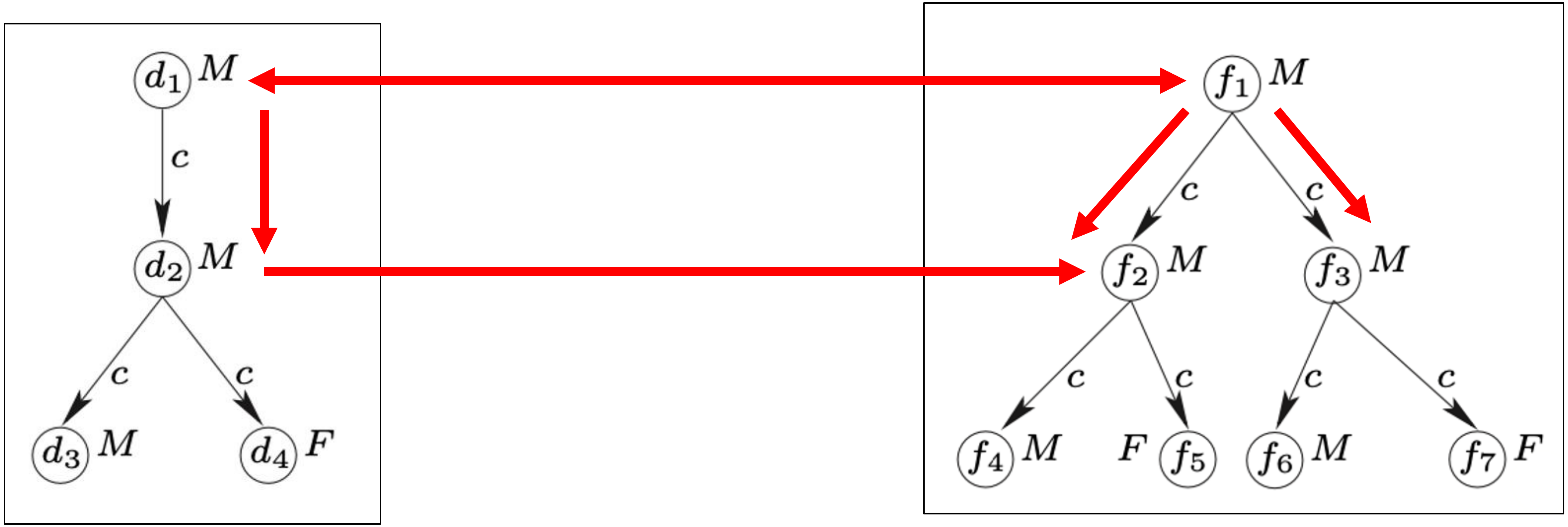
- **Zig:**

  If role c relates d_1 to d_2 then there is a mapping from d_2 to f_2 where d_2 and f_2 are both instances of M



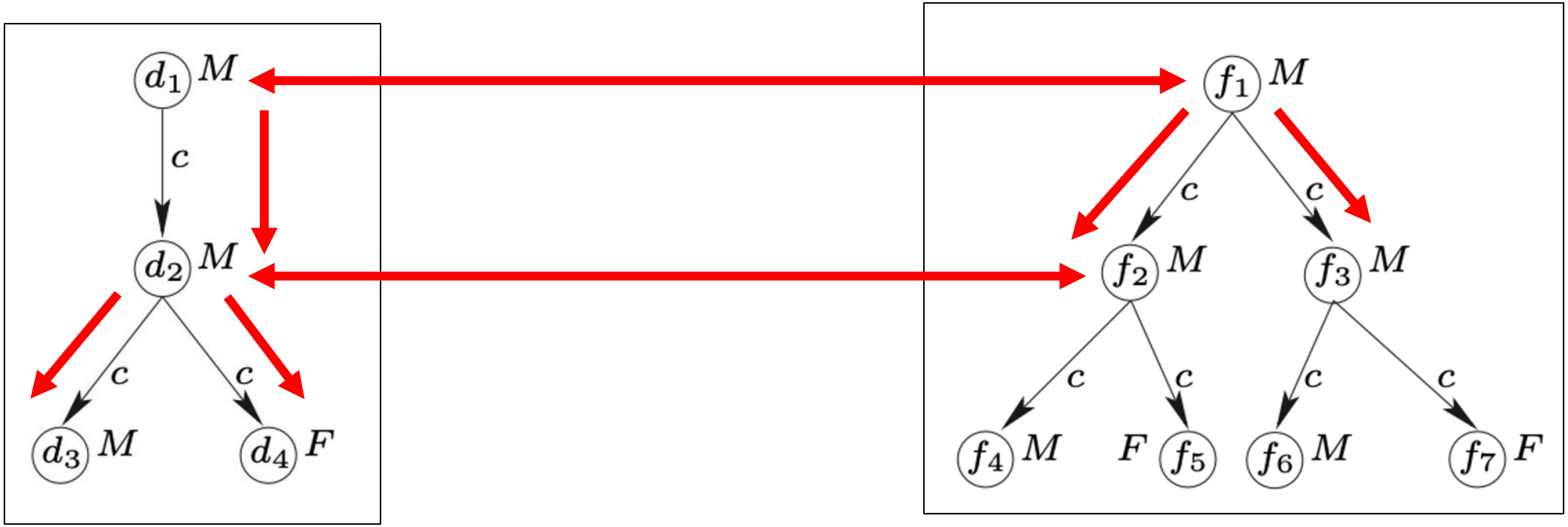
- **Claim:** d_1 is bisimilar to f_1

- **Zig:**
  If role c relates d_1 to d_2 then there is a mapping from d_2 to f_2 where d_2 and f_2 are both instances of M and role c maps f_1 to f_2



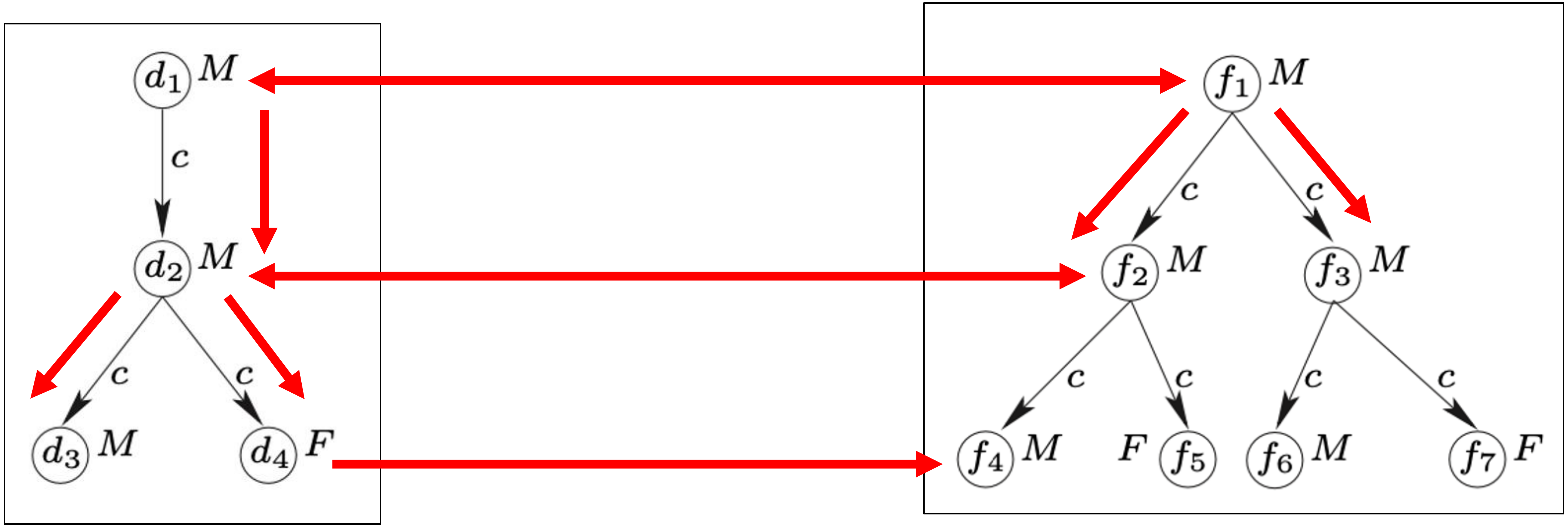
- **Claim:** d_1 is bisimilar to f_1

- **Zig:**
  If role c relates d_1 to d_2 then there is a mapping from d_2 to f_2 where d_2 and f_2 are both instances of M and role c maps f_1 to f_2 and f_1 to f_3



- **Claim:** d_1 is bisimilar to f_1

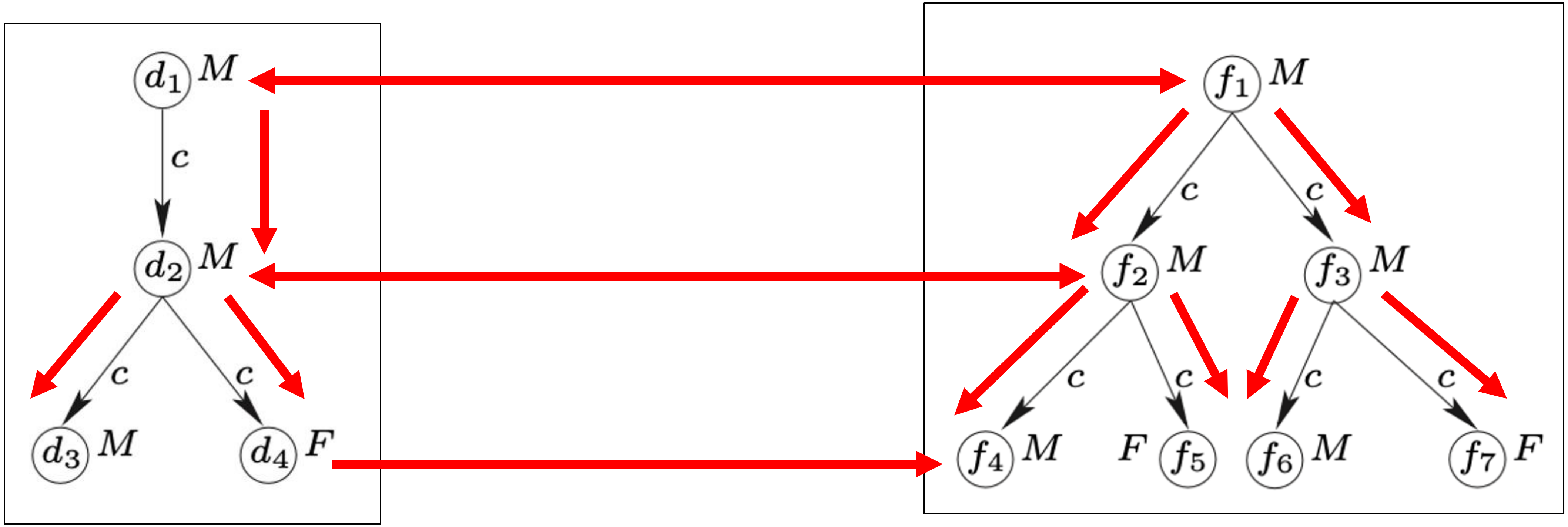
- **Zag:**
  If role c relates d_2 to d_3 and d_4



- **Claim:** d_1 is bisimilar to f_1

- **Zag:**

  If role c relates d_2 to d_3 and d_4 then there is a mapping from d_3 to f_4 and d_4 to f_5 as well as from d_3 to f_6 and d_4 to f_7 where d_3, f_4, and f_6 are instances of M and d_4, f_5, and f_7 are instances of F and



- **Claim:** d_1 is bisimilar to f_1

- **Zag:**
  If role c relates d_2 to d_3 and d_4 then there is a mapping from d_3 to f_4 and d_4 to f_5 as well as from d_3 to f_6 and d_4 to f_7 where d_3, f_4, and f_6 are instances of M and d_4, f_5, and f_7 are instances of F and c relates f_2 to f_4 and f_5 and related f_3 to f_6 and f_7

Summary

- No monolith ontologies are required, needed, or desired
- There are established strategies for constructing paths from user vocabularies back to the ontology whole cloth
- Genuine semantic alignment is challenging, but there are strategies for progress, namely putting the **proof** back in **future-proofing**