

CC6202-1

LA WEB DE DATOS

PRIMAVERA 2015

Lecture 4: Web Ontology Language (I)

Aidan Hogan

aidhog@gmail.com

**PREVIOUSLY ON
“LA WEB DE DATOS”**

(1) Data, (2) Rules/Ontologies, (3) Query,

INPUT: “ (x, partOf, y) ”

DATA:

<http://ex.org/Ireland>

Ireland



$(\text{Ireland}, \text{partOf}, \text{Europe})$
 $(\text{Ireland}, \text{a}, \text{Country})$
 $(\text{Ireland}, \text{capital}, \text{Dublin})$

<http://ex.org/Dublin>

Dublin



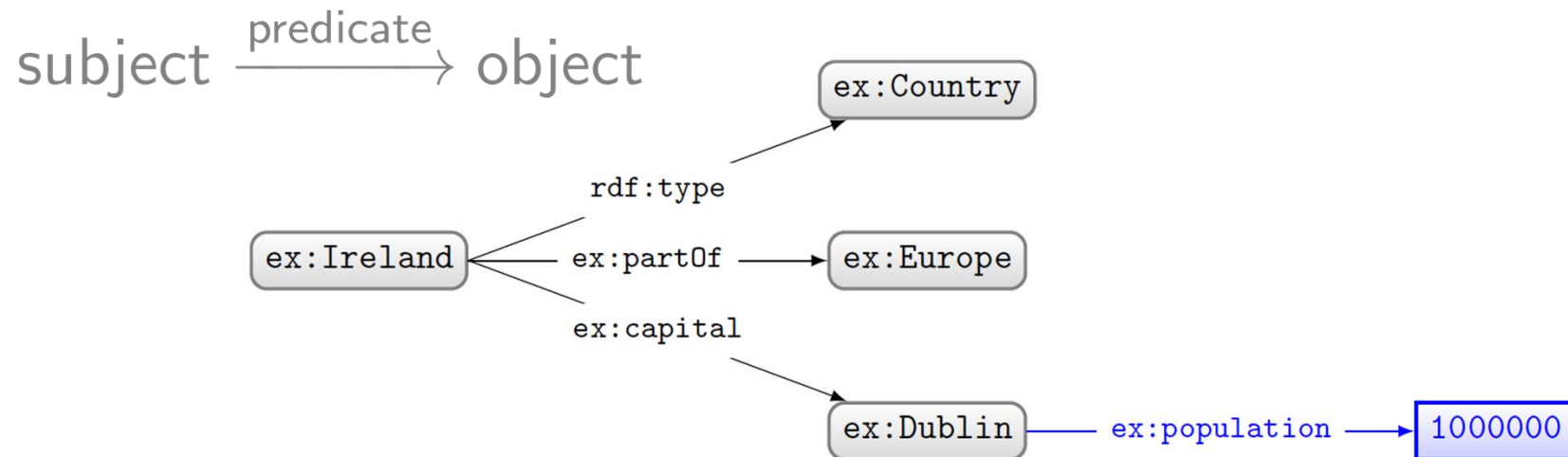
$(\text{Dublin}, \text{population}, 1000000)$

RULES: $(a, \text{capital}, b) \rightarrow (b, \text{partOf}, a)$
 $(c, \text{partOf}, d), (d, \text{partOf}, e) \rightarrow (c, \text{partOf}, e)$

OUTPUT: $\{(x \mapsto \text{Ireland}, y \mapsto \text{Europe}), (x \mapsto \text{Dublin}, y \mapsto \text{Ireland})$
 $(x \mapsto \text{Dublin}, y \mapsto \text{Europe})\}$

RDF: Resource Description Framework

<i>subject</i>	<i>predicate</i>	<i>object</i>
ex:Ireland	ex:partOf	ex:Europe
ex:Ireland	rdf:type	ex:Country
ex:Ireland	ex:capital	ex:Dublin
ex:Dublin	ex:population	1,000,000



(1) Data, (2) Rules/Ontologies, (3) Query

INPUT: “ (x, partOf, y) ”

DATA:

<http://ex.org/Ireland>

Ireland



(Ireland, partOf, Europe)
(Ireland, a, Country)
(Ireland, capital, Dublin)

<http://ex.org/Dublin>

Dublin



(Dublin, population, 1000000)

RULES: $(a, \text{capital}, b) \rightarrow (b, \text{partOf}, a)$
 $(c, \text{partOf}, d), (d, \text{partOf}, e) \rightarrow (c, \text{partOf}, e)$

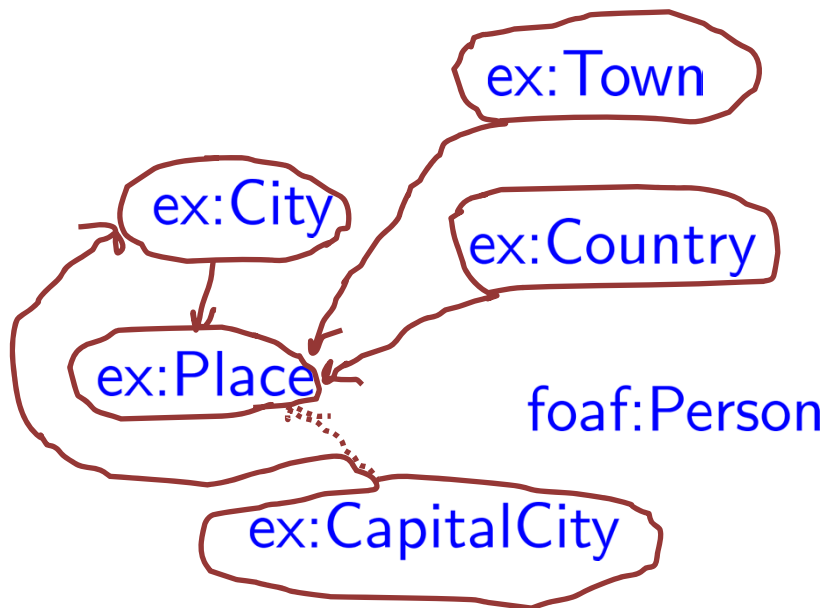
OUTPUT: $\{(x \mapsto \text{Ireland}, y \mapsto \text{Europe}), (x \mapsto \text{Dublin}, y \mapsto \text{Ireland})$
 $(x \mapsto \text{Dublin}, y \mapsto \text{Europe})\}$

Class hierarchy

- Class $c1$ is a sub-class of Class $c2$
 - If $(x, \text{rdf:type}, c1)$ then $(x, \text{rdf:type}, c2)$,

*Example: if ex:CapitalCity sub-class of ex:City
and if $(\text{ex:Dublin}, \text{rdf:type}, \text{ex:CapitalCity})$
then $(\text{ex:Dublin}, \text{rdf:type}, \text{ex:City})$*

Which classes would be **sub-classes** of each other?

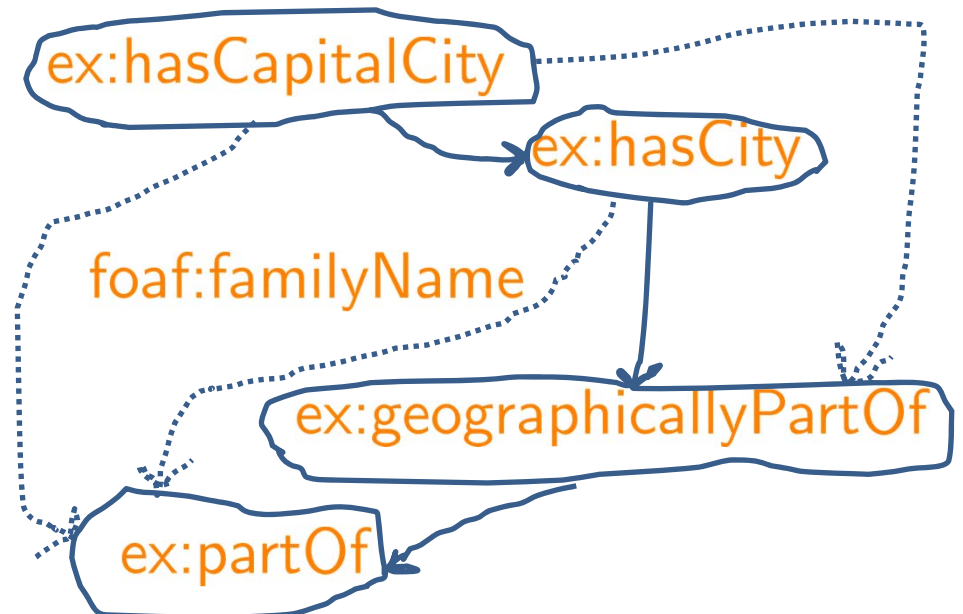


Property hierarchy

- Property $p1$ is a sub-property of $p2$
 - If $(x, p1, y)$ then $(x, p2, y)$

*Example: if `ex:hasCapitalCity` sub-property of `ex:hasCity`
and if $(\text{ex:Ireland}, \text{ex:hasCapitalCity}, \text{ex:Dublin})$
then $(\text{ex:Ireland}, \text{ex:hasCity}, \text{ex:Dublin})$*

Which properties would be **sub-properties** of each other?

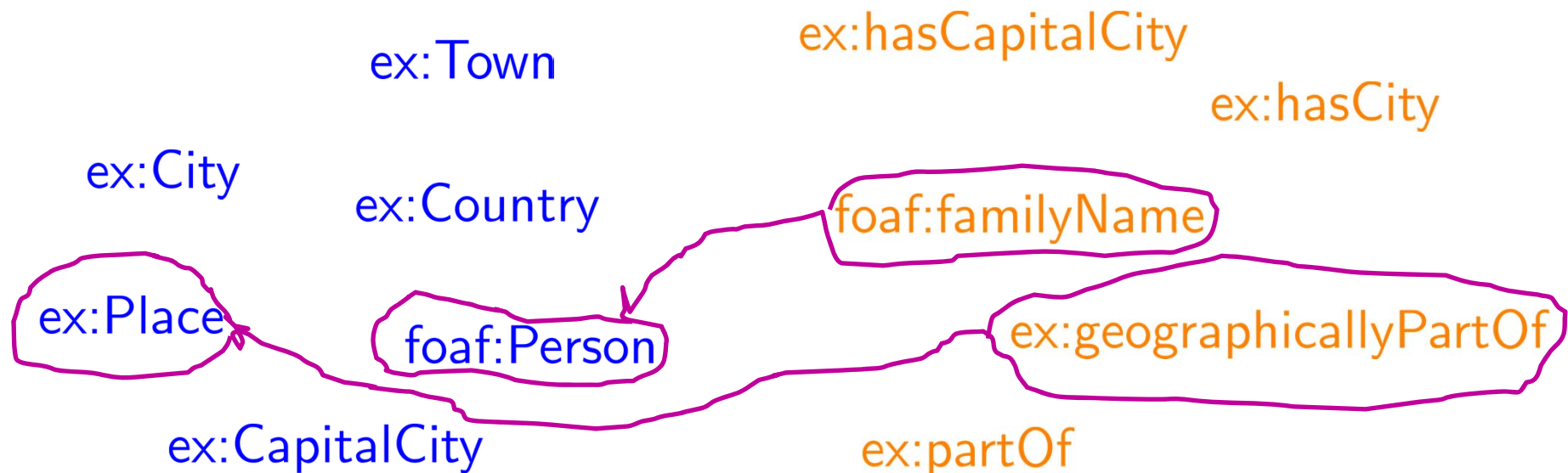


Domain of properties

- Property p has domain class c
 - If (x, p, y) then $(x, \text{rdf:type}, c)$

*Example: if foaf:familyName has domain foaf:Person
and if $(\text{ex:Aidan}, \text{foaf:familyName}, \text{'Hogan'})$
then $(\text{ex:Aidan}, \text{rdf:type}, \text{foaf:Person})$*

Which properties would have which classes as **domain**?

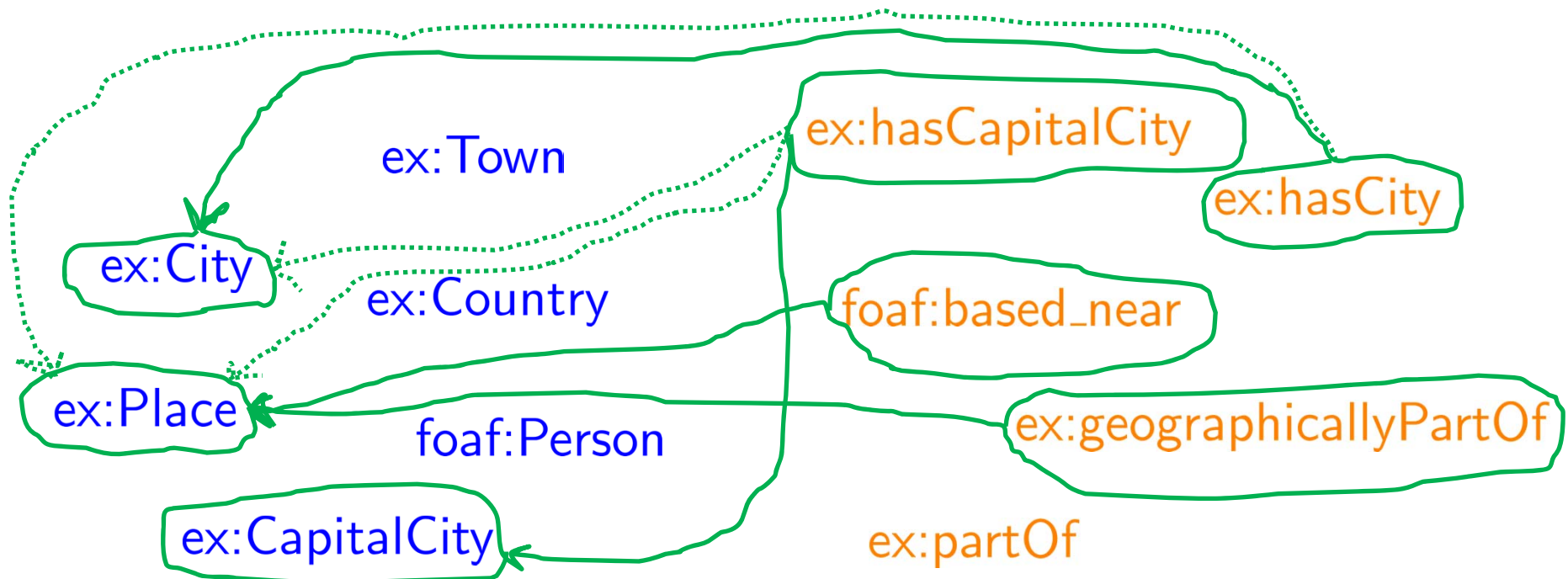


Range of properties

- Property p has range class c
 - If (x, p, y) then $(y, \text{rdf:type}, c)$

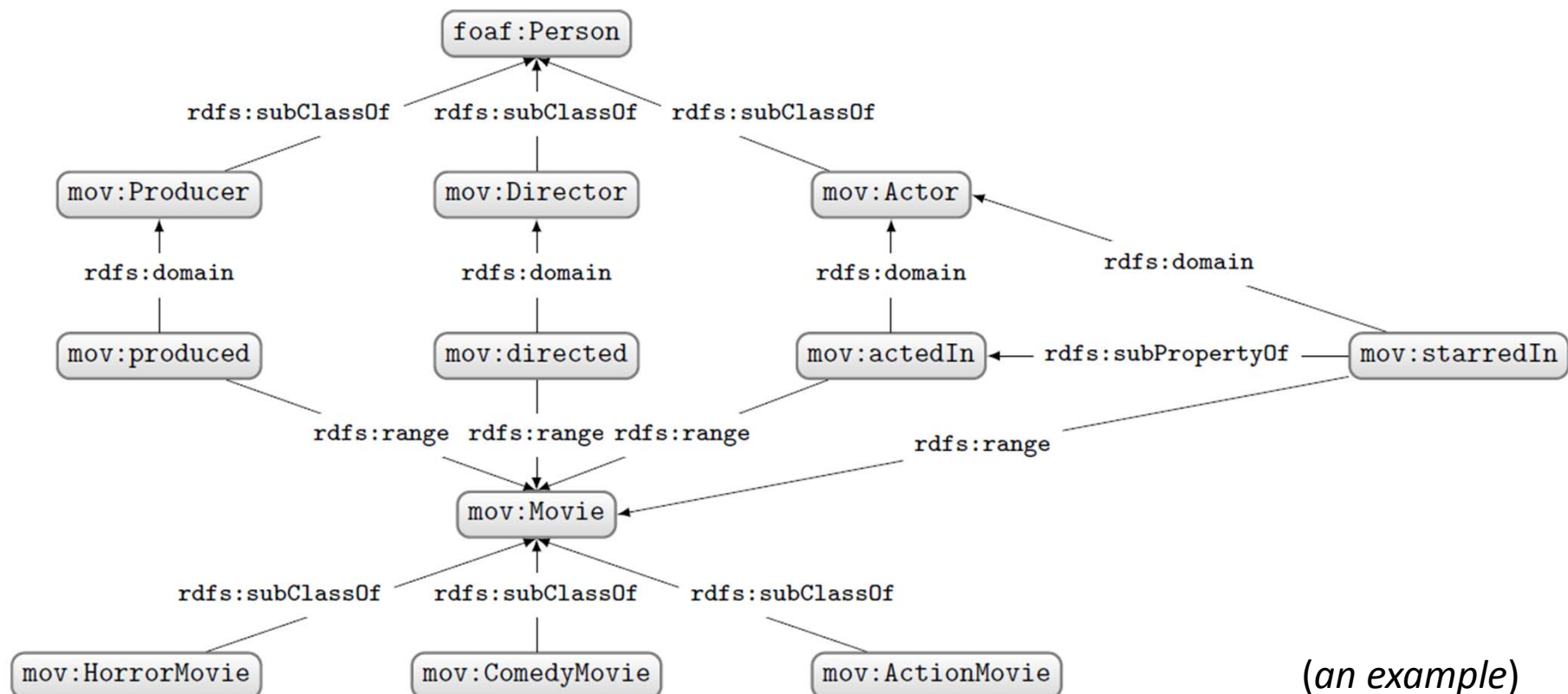
*Example: if `ex:hasCity` has range `ex:City`
and if $(\text{ex:Ireland}, \text{ex:hasCity}, \text{ex:Dublin})$
then $(\text{ex:Dublin}, \text{rdf:type}, \text{ex:City})$*

Which properties would have which classes as **range**?



RDF Schema ...

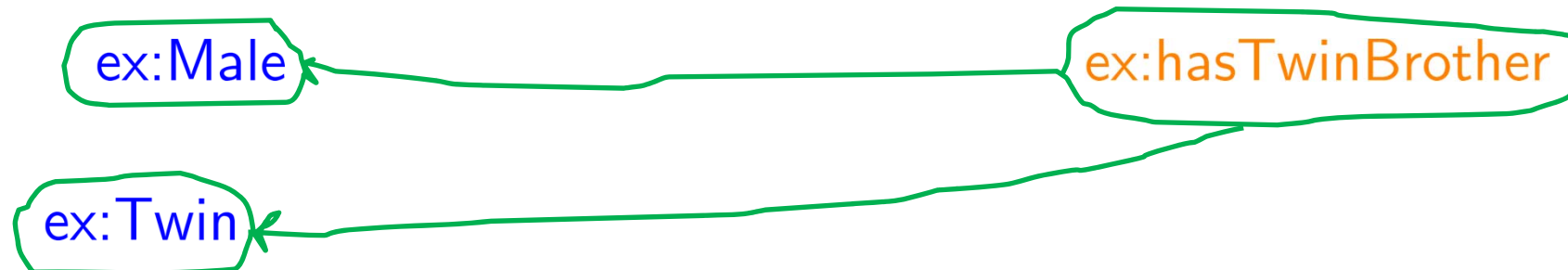
Let's model an RDF Schema for Movies, including different types of movies, some different types of people involved, and how they are related.



(an example)

On a side note

- A property can have multiple **domains** or **ranges**



TODAY'S TOPIC ...

(1) Data, (2) Rules/Ontologies, (3) Query

INPUT: “ (x, partOf, y) ”

DATA:

<http://ex.org/Ireland>

Ireland



$(\text{Ireland}, \text{partOf}, \text{Europe})$
 $(\text{Ireland}, a, \text{Country})$
 $(\text{Ireland}, \text{capital}, \text{Dublin})$

<http://ex.org/Dublin>

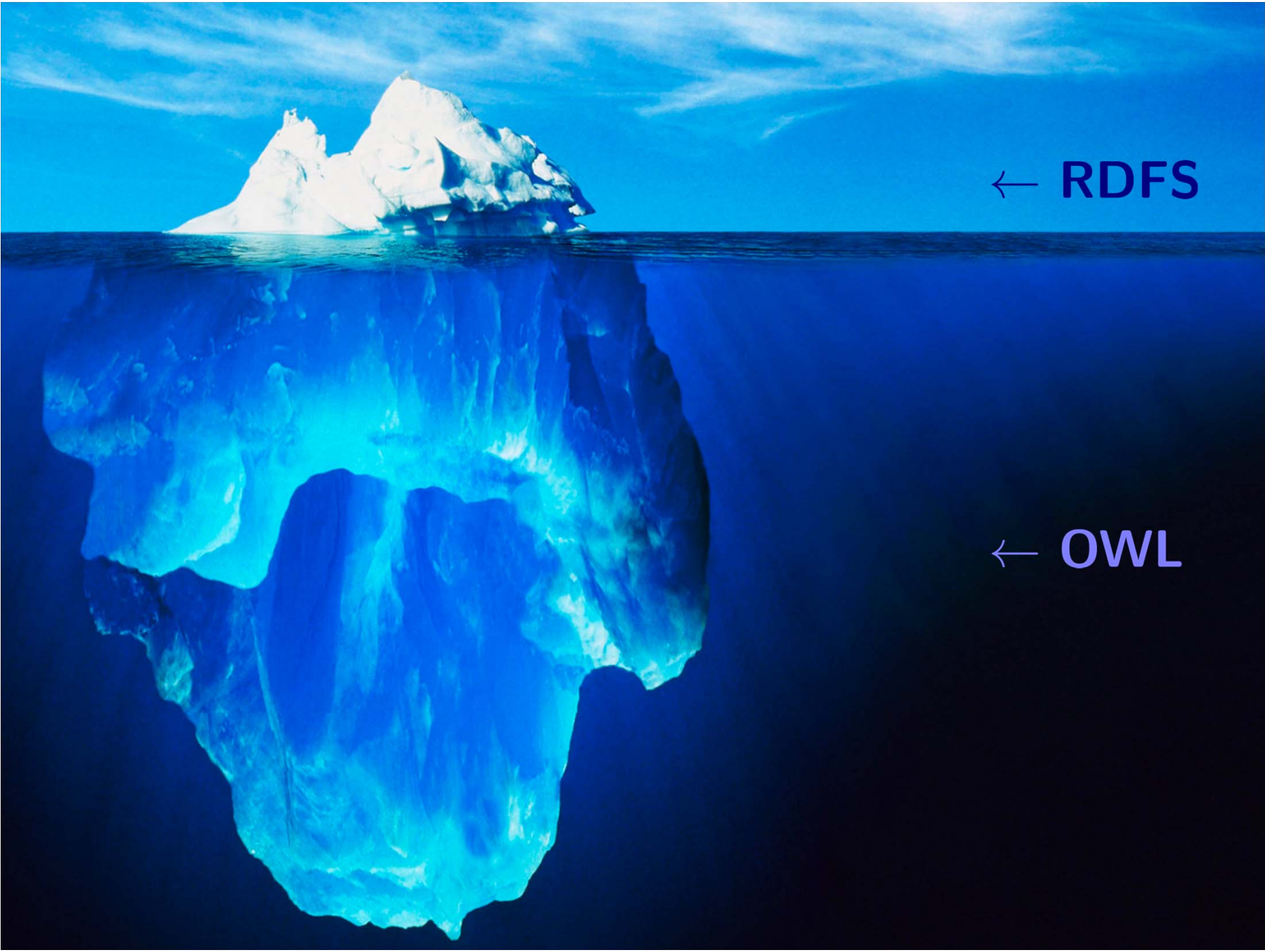
Dublin



$(\text{Dublin}, \text{population}, 1000000)$

RULES: $(a, \text{capital}, b) \rightarrow (b, \text{partOf}, a)$
 $(c, \text{partOf}, d), (d, \text{partOf}, e) \rightarrow (c, \text{partOf}, e)$

OUTPUT: $\{(x \mapsto \text{Ireland}, y \mapsto \text{Europe}), (x \mapsto \text{Dublin}, y \mapsto \text{Ireland})$
 $(x \mapsto \text{Dublin}, y \mapsto \text{Europe})\}$

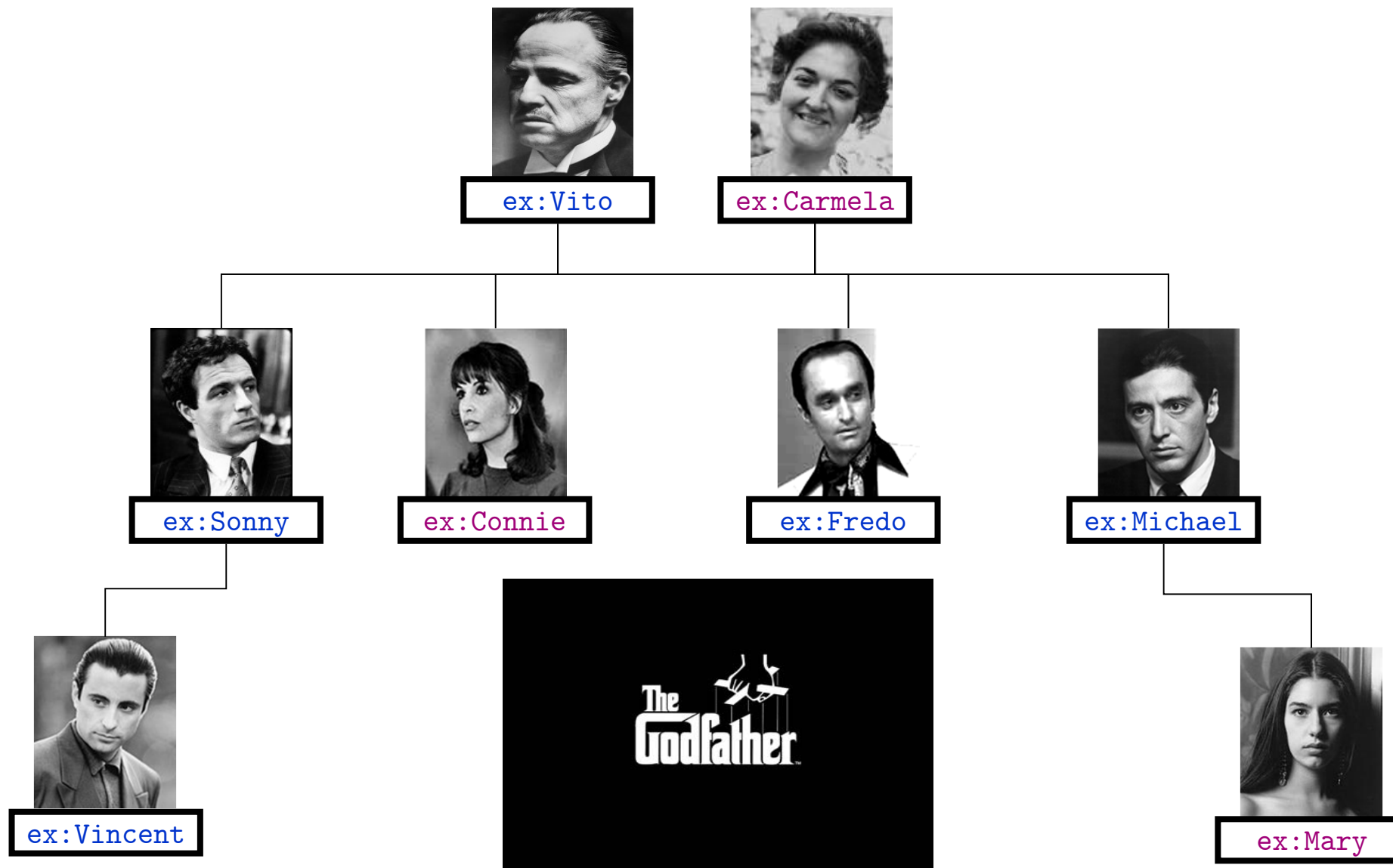
An iceberg floating in a deep blue ocean under a clear blue sky. The small tip of the iceberg is above the water line, while the much larger, more complex structure is submerged below. The water is a deep, dark blue, and the sky is a lighter, clear blue with some wispy clouds. The iceberg's surface is jagged and textured, showing various shades of white and light blue. The submerged part is also jagged and textured, showing various shades of blue and white. The overall image conveys the idea that what is visible is only a small fraction of the whole.

← RDFS

← OWL

LET'S DIVE INTO OWL

A special family ...

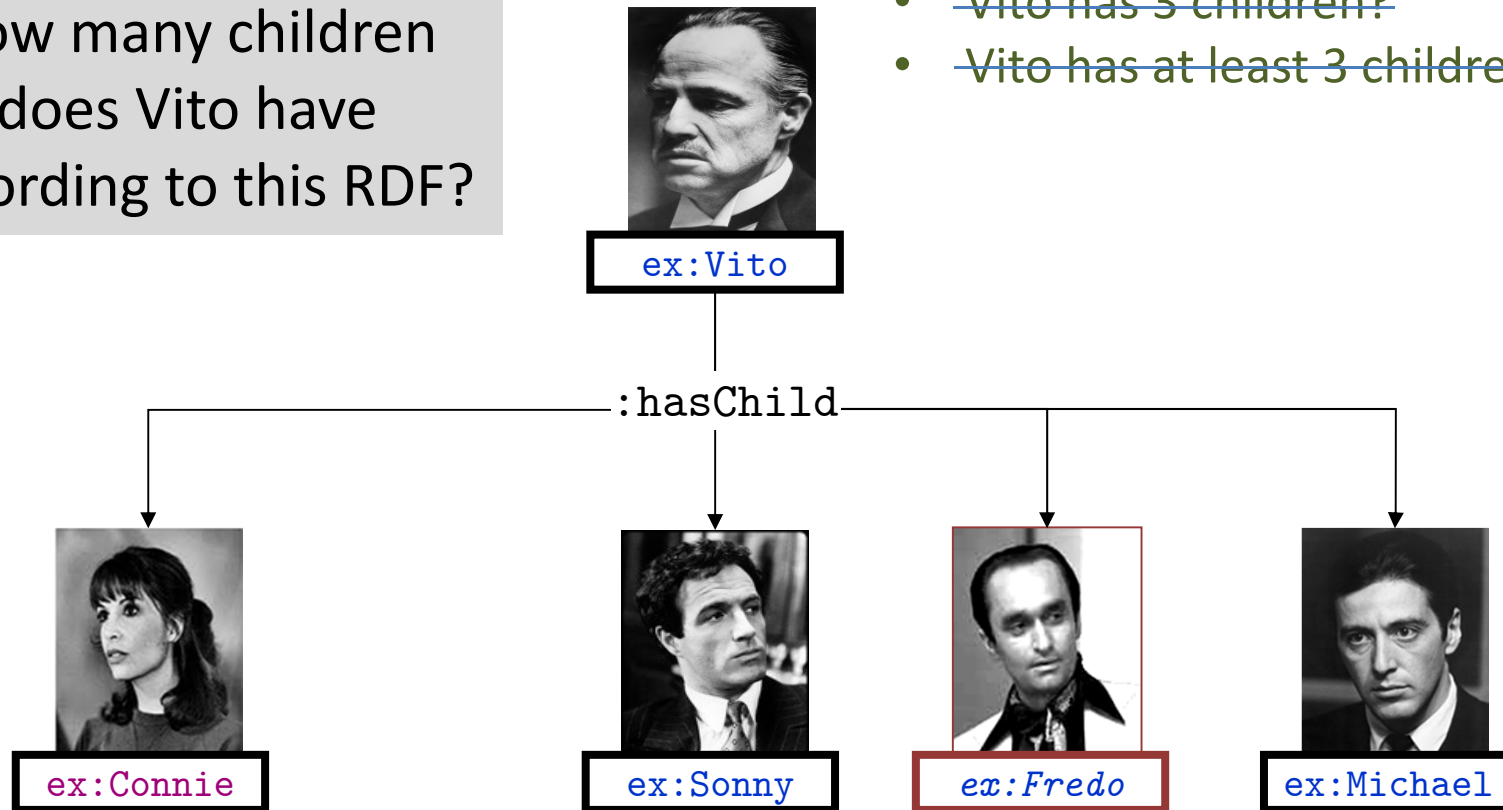


FIRST SOME PRELIMINARIES

Open World Assumption (OWA)

How many children does Vito have according to this RDF?

- ~~Vito has 3 children?~~
- ~~Vito has at least 3 children?~~



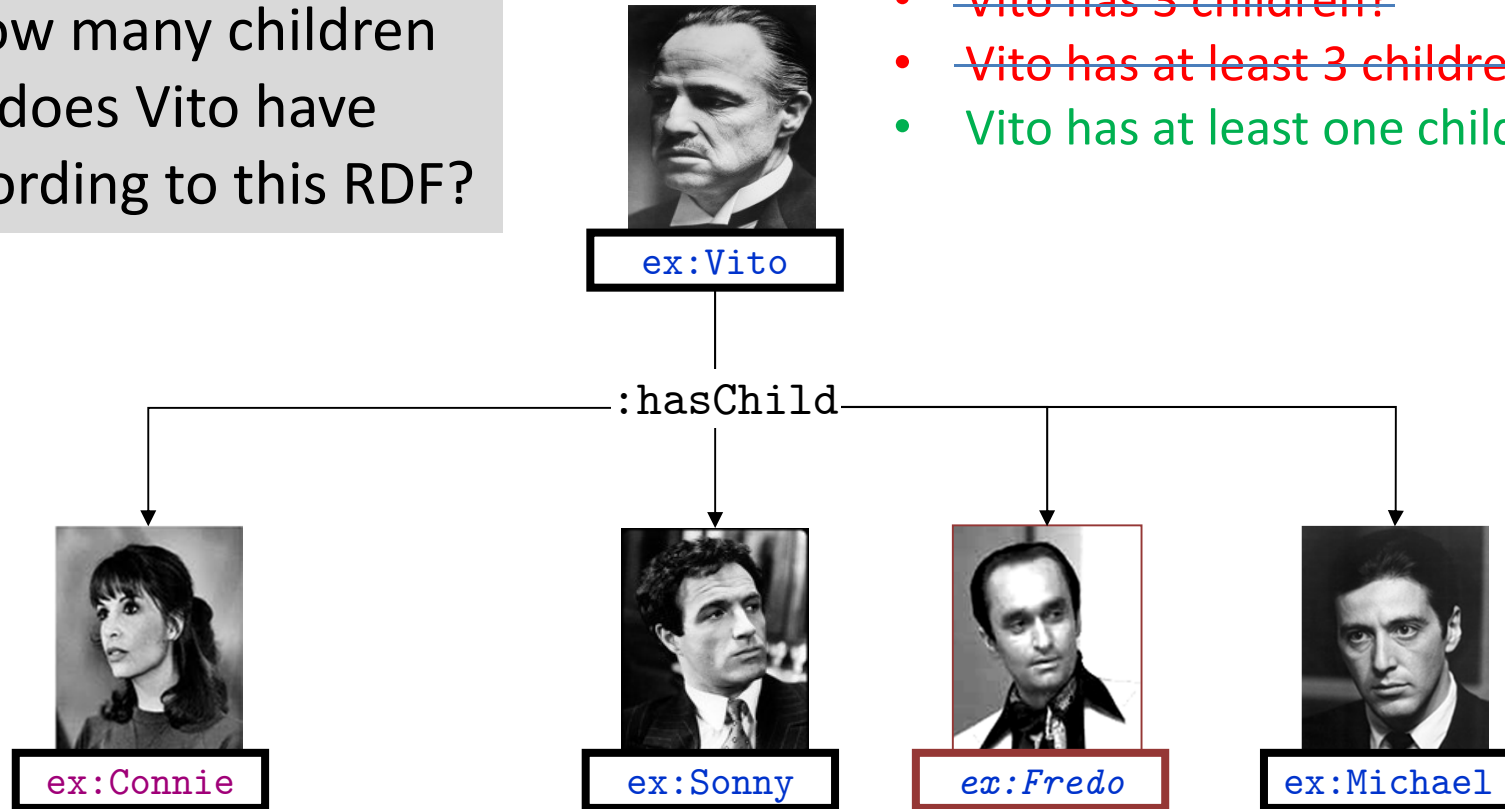
`ex:Vito :hasChild ex:Connie, ex:Sonny, ex:Michael .`

`ex:Vito :hasChild ex:Fredo .`

`... ?`

No Unique Name Assumption (No UNA)

How many children does Vito have according to this RDF?



- ~~Vito has 3 children?~~
- ~~Vito has at least 3 children?~~
- Vito has at least one child!

`ex:Vito :hasChild ex:Connie, ex:Sonny, ex:Michael .`

`ex:Vito :hasChild ex:Fredo .`

`... ?`

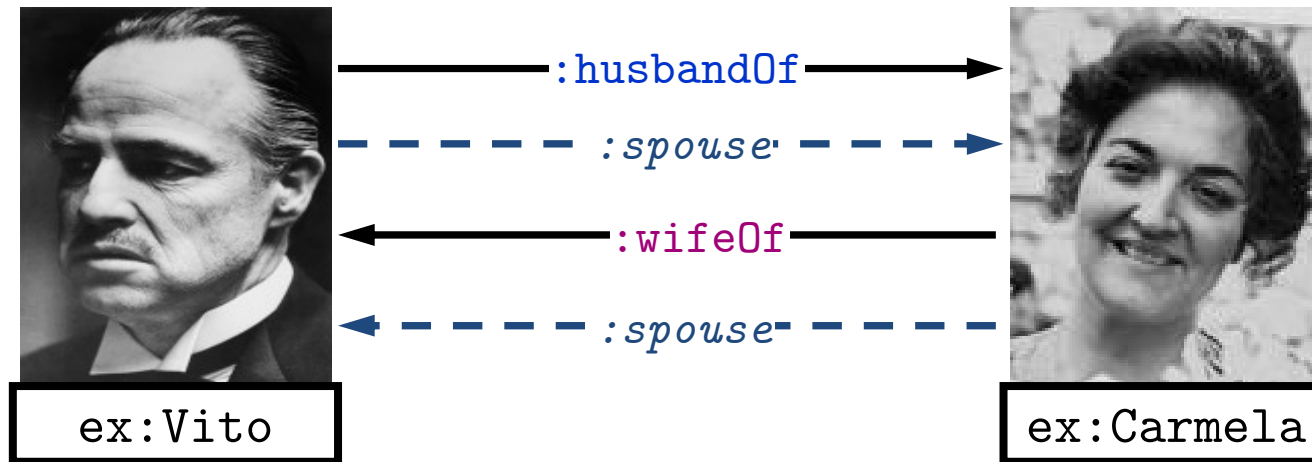
... in words

- RDF(S) and OWL:
 - Take an **Open World Assumption**:
 - Anything not known is not assumed to be false, simply unknown
 - Without further information, Vito may have children that we don't know about!
 - Do not take a **Unique Name Assumption**:
 - Two or more IRIs may refer to the same thing!
 - Without further information, the IRIs we know to be Vito's children may refer to one thing in the real-world!

Why might this be important for the Web?

LET'S START WITH SOME RDFS ...

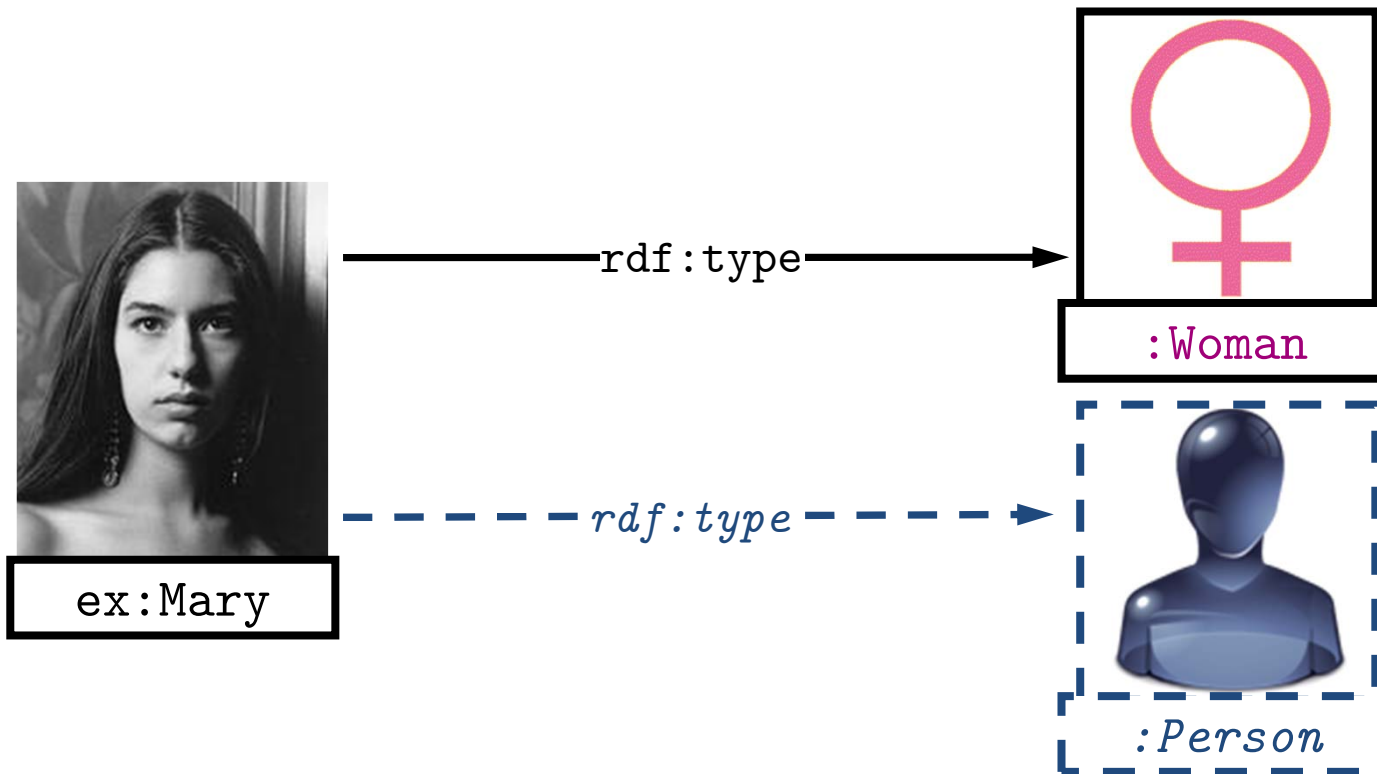
rdfs:subPropertyOf



```
ex:Vito :husbandOf ex:Carmela .  
:husbandOf rdfs:subPropertyOf :spouse .  
⇒ ex:Vito :spouse ex:Carmela .
```

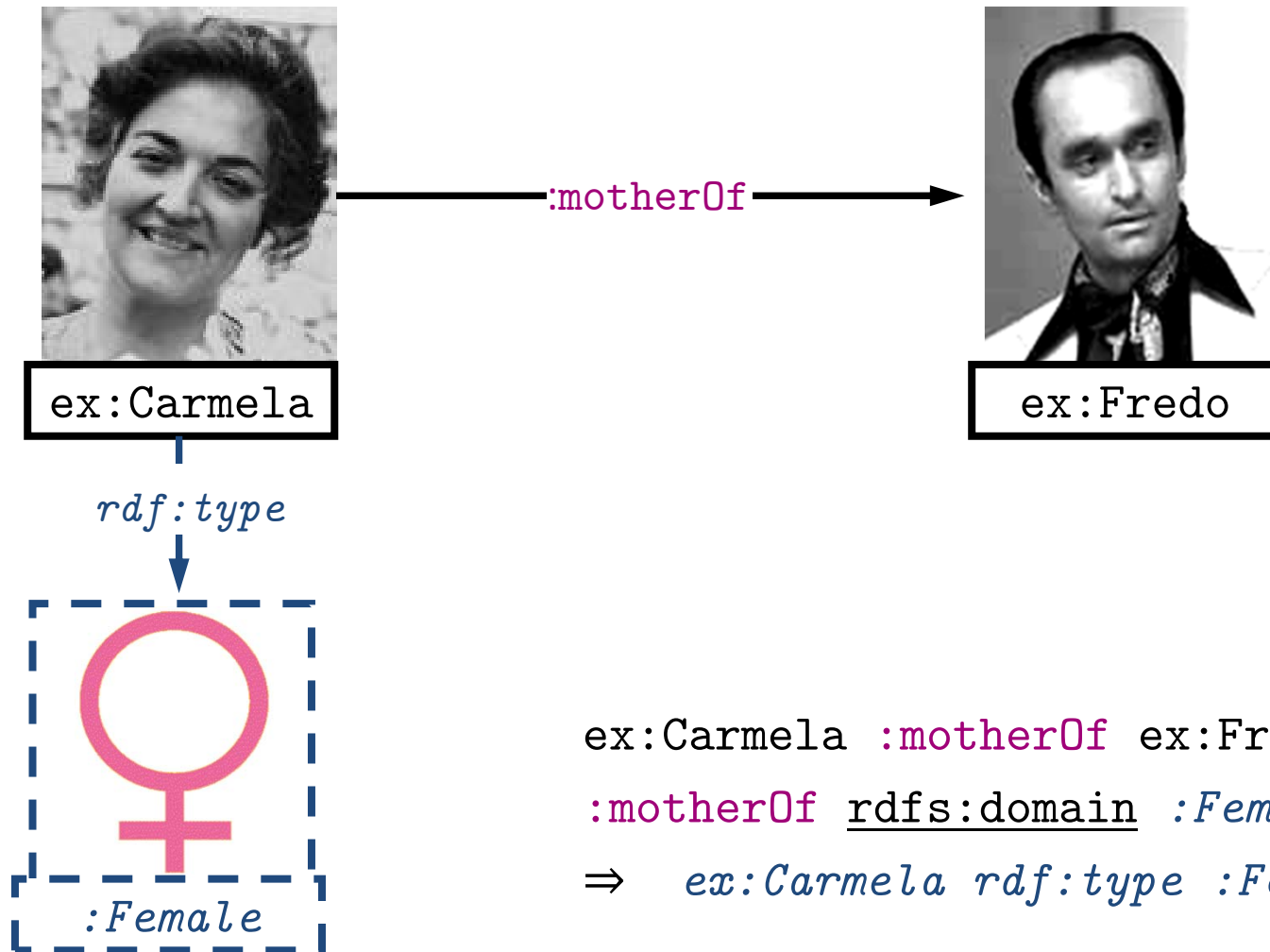
```
ex:Carmela :wifeOf ex:Vito .  
:wifeOf rdfs:subPropertyOf :spouse .  
⇒ ex:Carmela :spouse ex:Vito .
```

rdfs:subClassOf



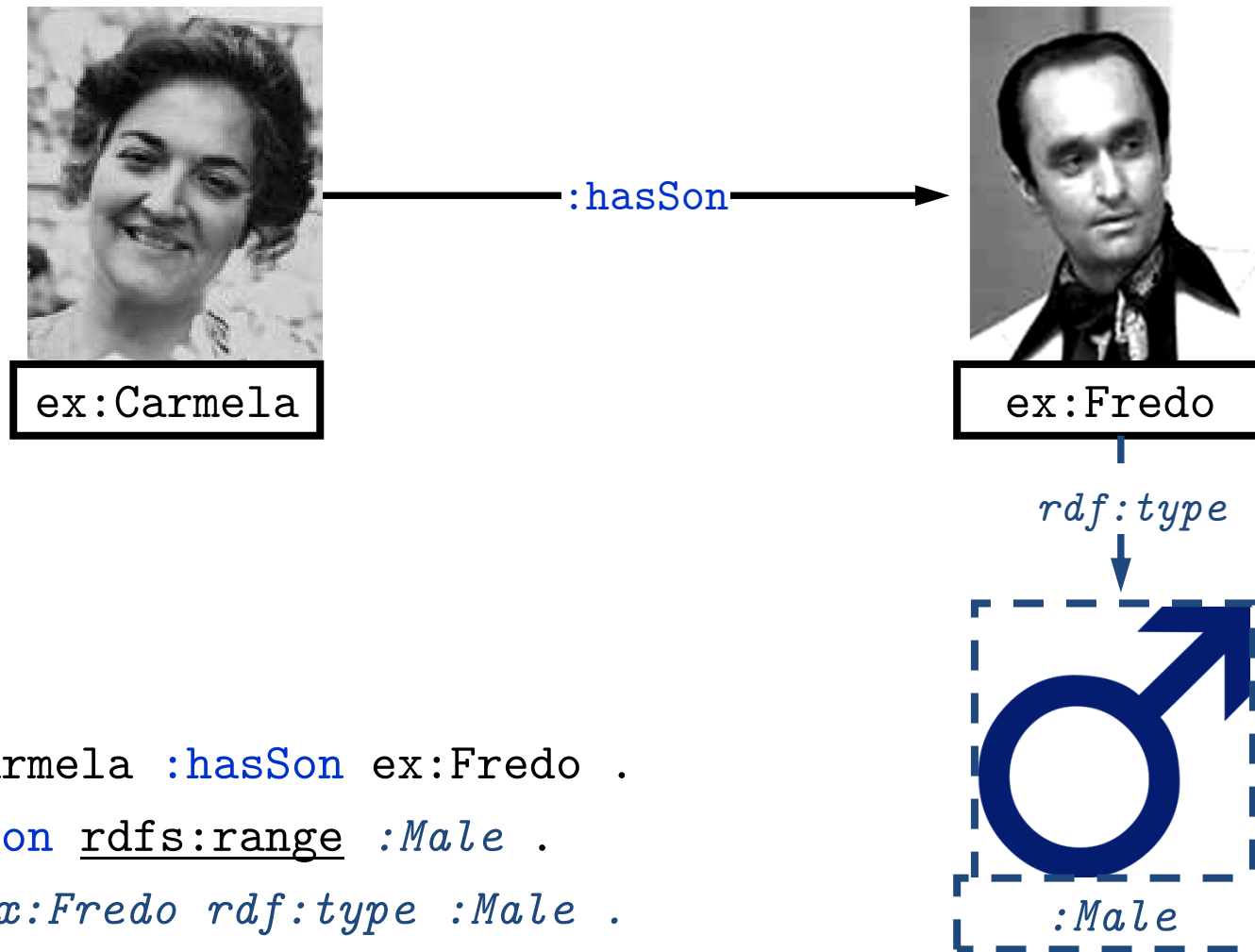
```
ex:Mary rdf:type :Woman .  
:Woman rdfs:subClassOf :Person .  
⇒ ex:Mary rdf:type :Person .
```

rdfs:domain



```
ex:Carmela :motherOf ex:Fredo .  
:motherOf rdfs:domain :Female.  
⇒ ex:Carmela rdf:type :Female .
```

rdfs:range

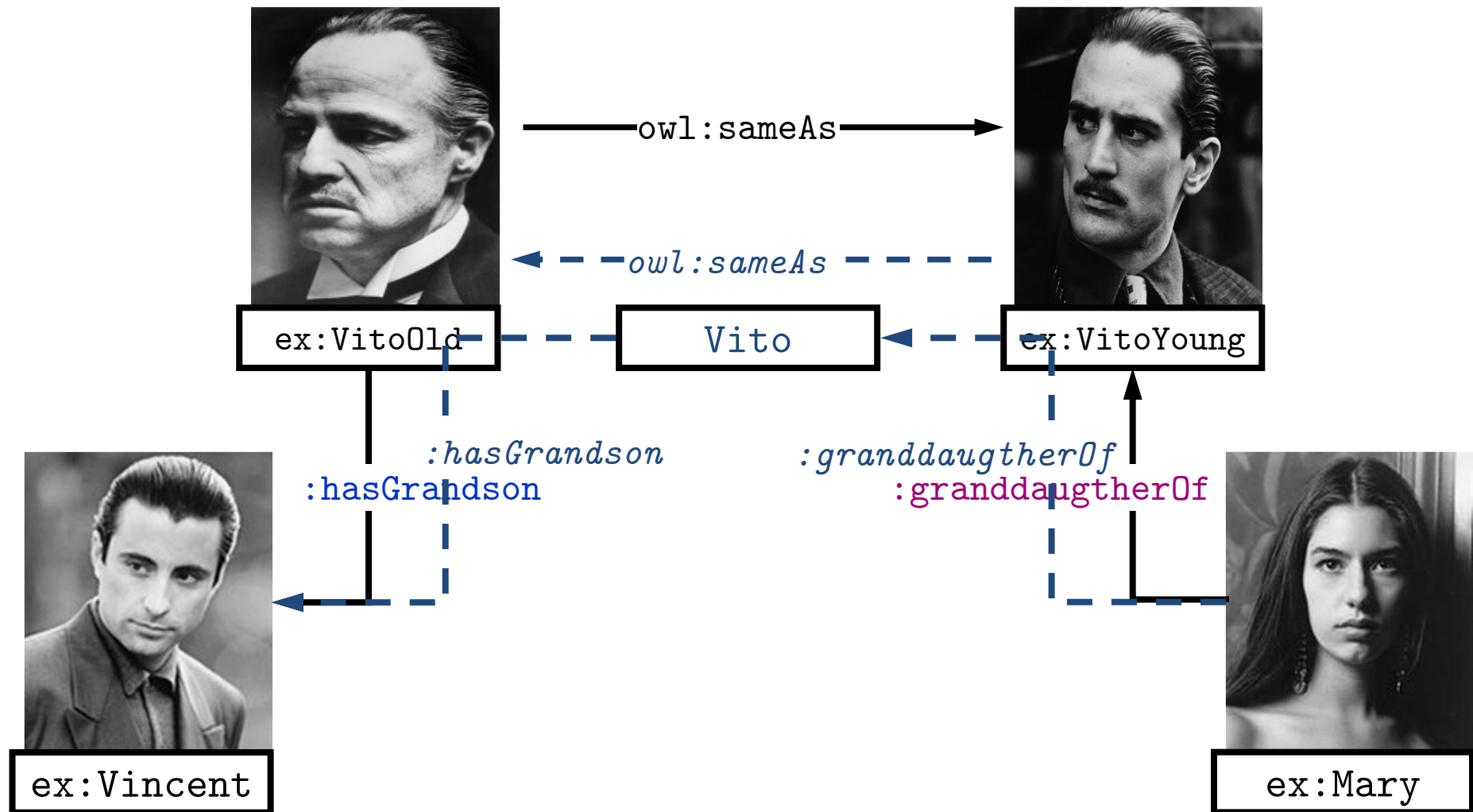


```
ex:Carmela :hasSon ex:Fredo .  
:hasSon rdfs:range :Male .  
⇒ ex:Fredo rdf:type :Male .
```

(IN)EQUALITY IN OWL ...

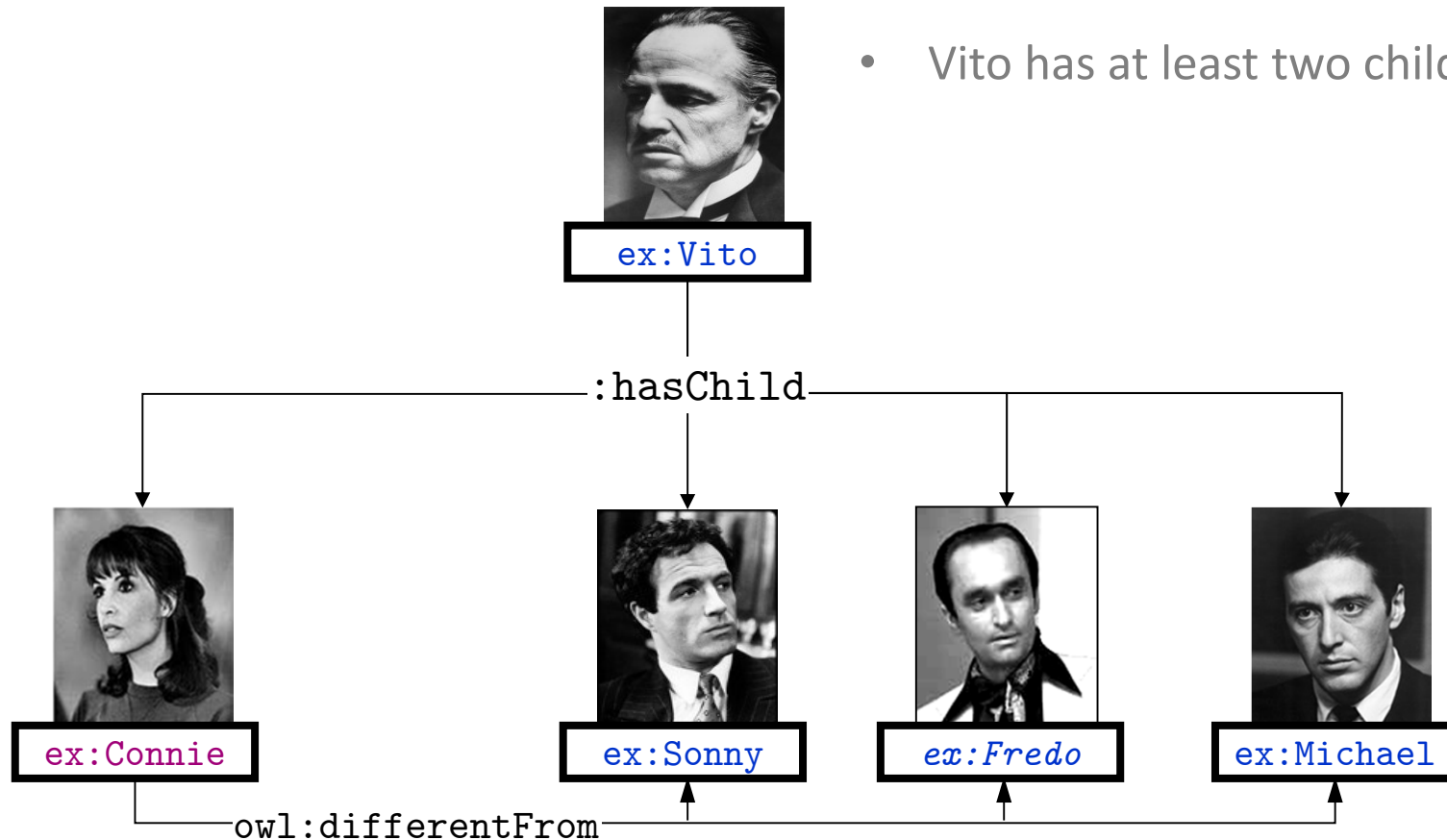


owl:sameAs



```
ex:VitoOld owl:sameAs ex:VitoYoung .
```

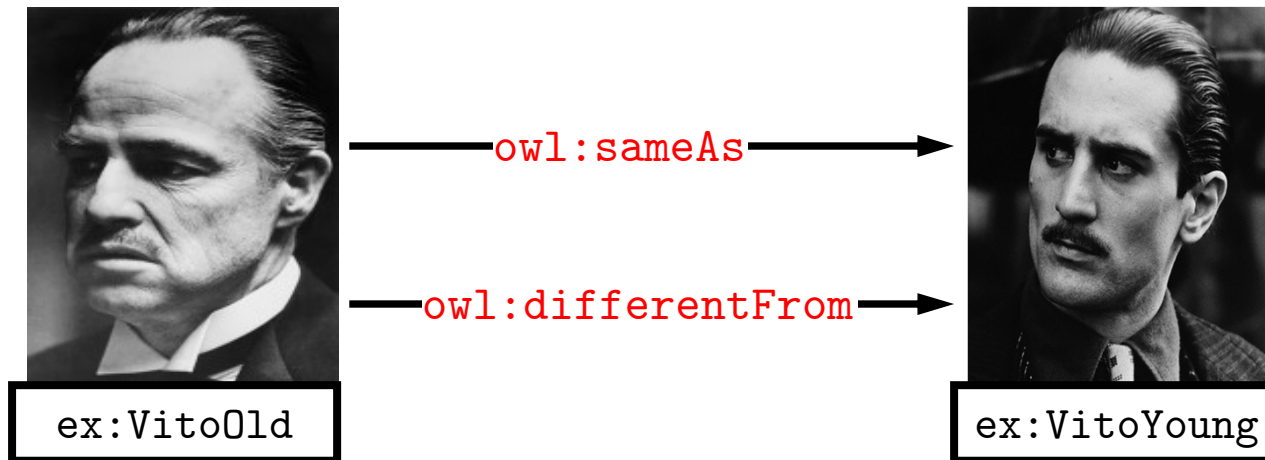
owl:differentFrom



```
ex:Vito :hasChild ex:Connie, ex:Sonny, ex:Michael, ex:Fredo .  
ex:Connie owl:differentFrom ex:Sonny, ex:Michael, ex:Fredo .
```

Inconsistency in OWL ...

- Contradictory statements



```
ex:VitoOld owl:sameAs ex:VitoYoung .  
ex:VitoOld owl:differentFrom ex:VitoYoung .
```

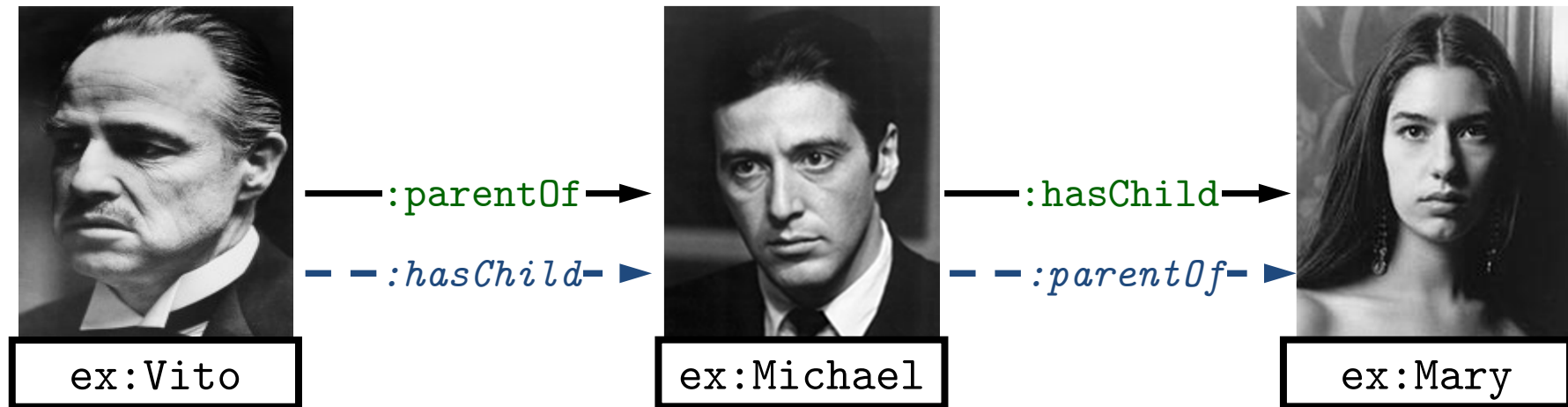
⇒ *FALSE*



PROPERTY AXIOMS IN OWL ...



owl:equivalentProperty



`ex:Vito :parentOf ex:Michael .`

`ex:Michael :hasChild ex:Mary .`

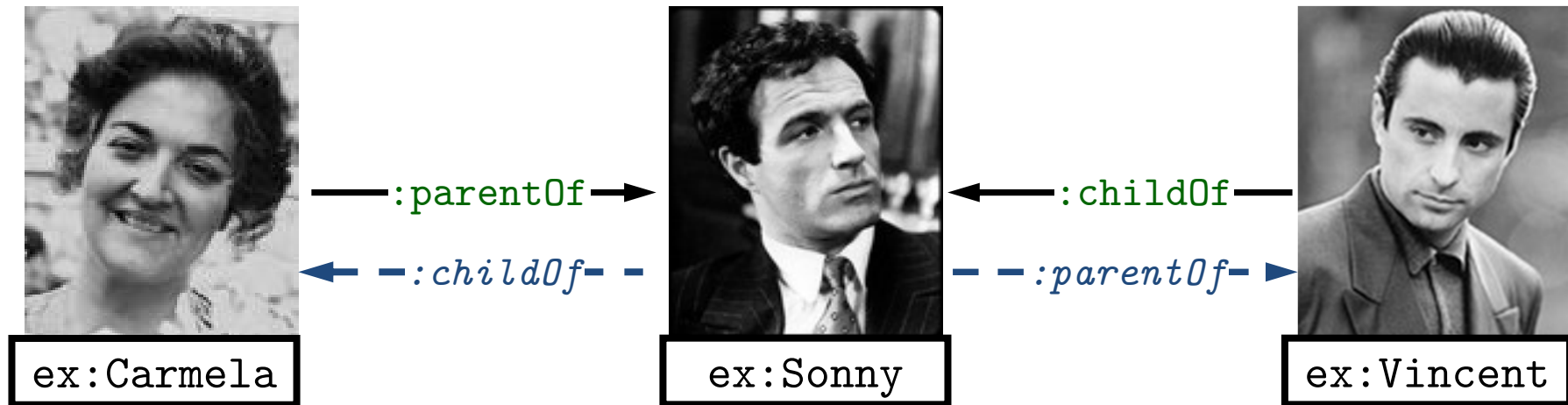
`:parentOf owl:equivalentProperty :hasChild .`

$\Rightarrow$ `ex:Vito :hasChild ex:Michael .`

$\Rightarrow$ `ex:Michael :parentOf ex:Mary .`



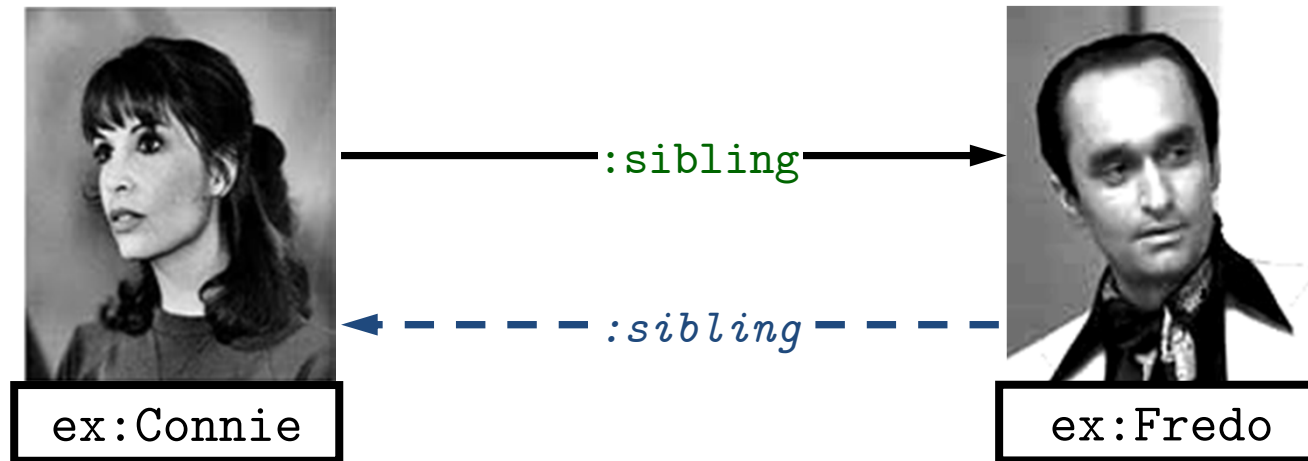
owl:inverseOf



```
ex:Carmela :parentOf ex:Sonny .  
ex:Vincent :childOf ex:Sonny .  
:parentOf owl:inverseOf :childOf .  
⇒ ex:Sonny :parentOf ex:Vincent .  
⇒ ex:Sonny :childOf ex:Carmela .
```



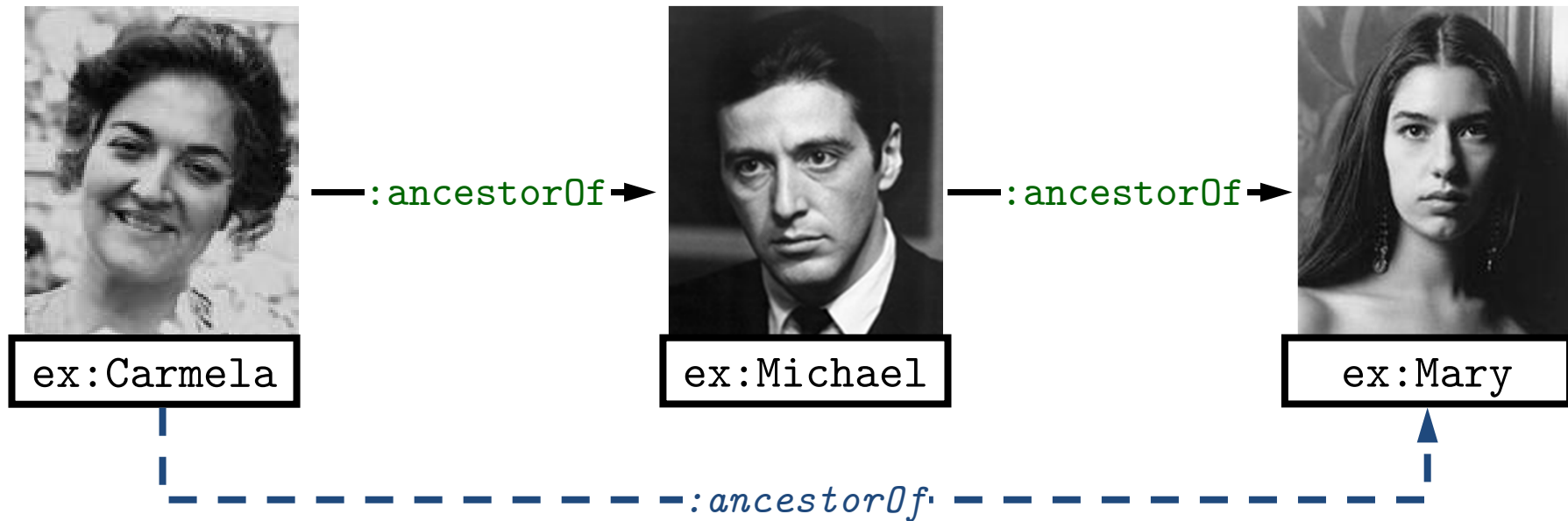
owl:SymmetricProperty



```
ex:Connie :sibling ex:Fredo .  
:sibling rdf:type owl:SymmetricProperty .  
⇒ ex:Fredo :sibling ex:Connie .
```



owl:TransitiveProperty



ex:Carmela :ancestorOf ex:Michael .

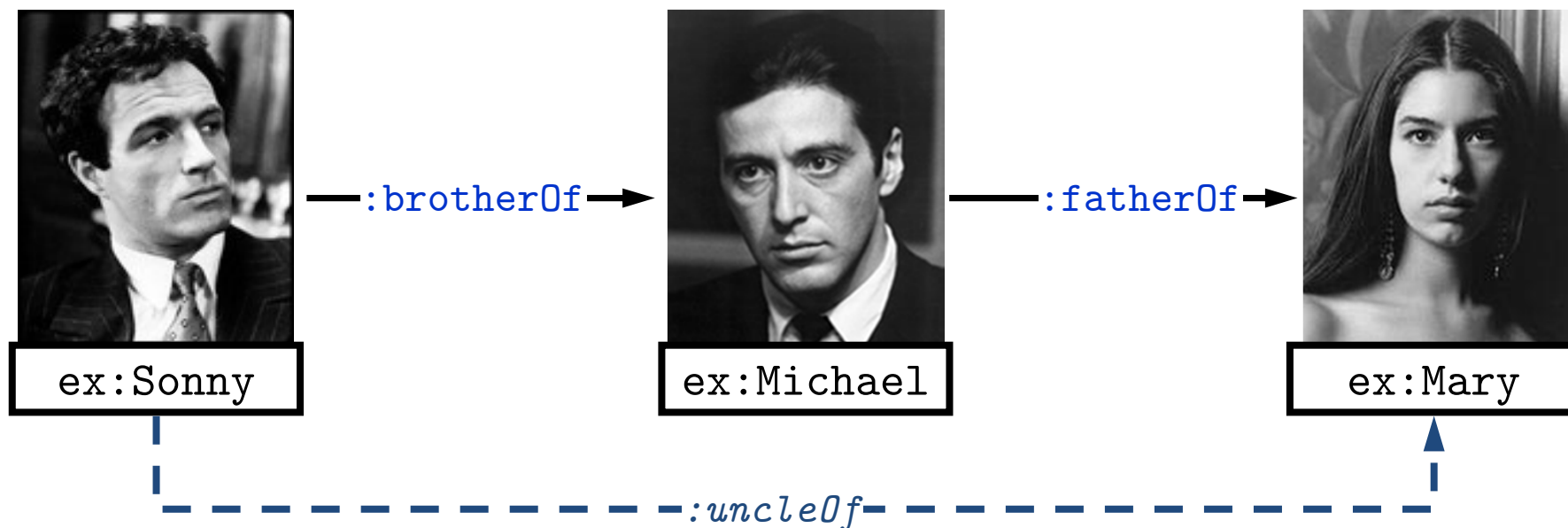
ex:Michael :ancestorOf ex:Mary .

:ancestorOf rdf:type owl:TransitiveProperty .

⇒ *ex:Carmela :ancestorOf ex:Mary .*

owl:propertyChainAxiom

Means new to OWL version 2.0!



ex:Sonny :brotherOf ex:Michael .

ex:Michael :fatherOf ex:Mary .

:uncleOf owl:propertyChainAxiom (:brotherOf :fatherOf) .

⇒ ex:Sonny :uncleOf ex:Mary .

owl:ReflexiveProperty



```
:similarTo rdf:type owl:ReflexiveProperty .
```

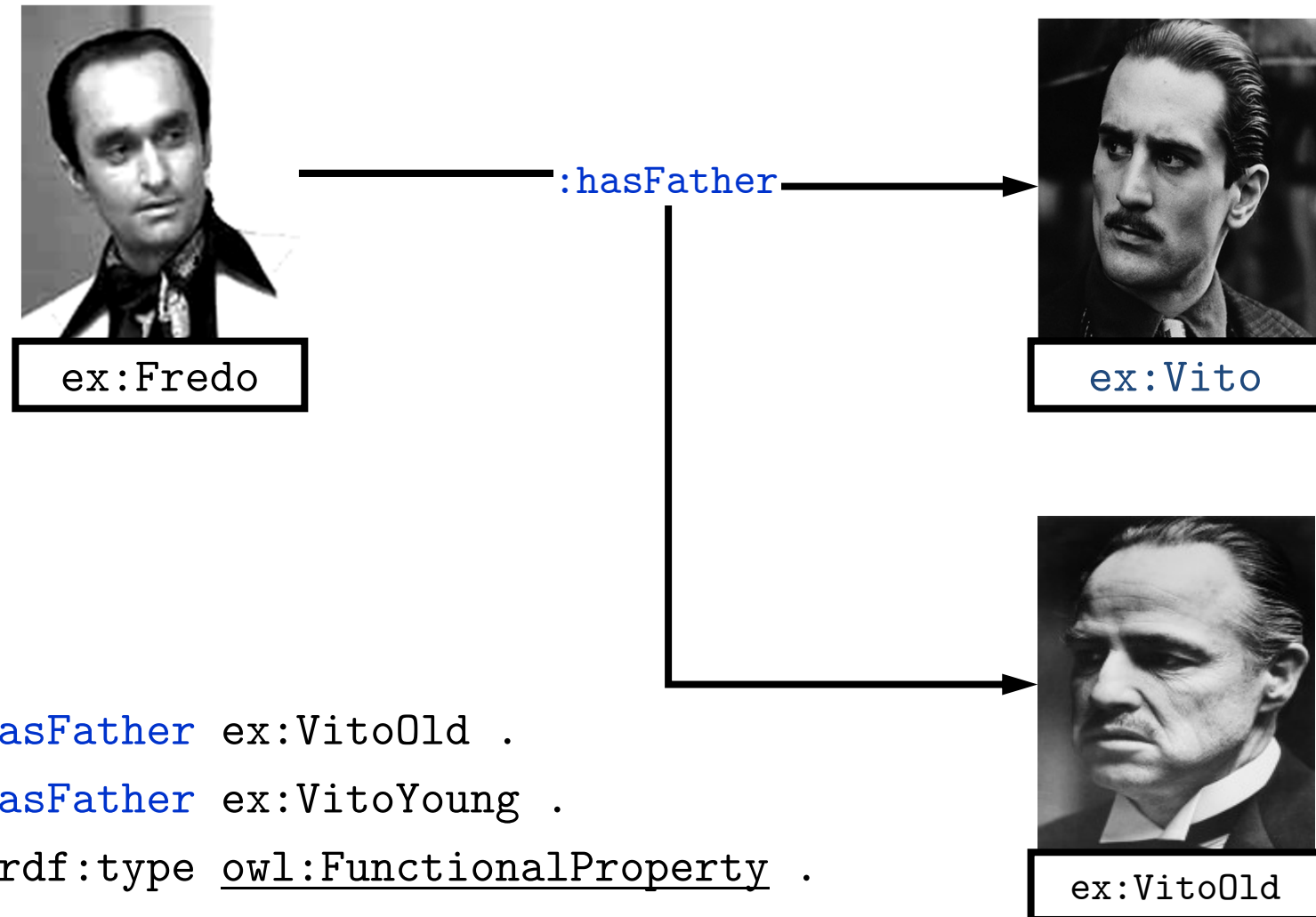
```
⇒ ex:Connie :similarTo ex:Connie .
```

```
ex:Freddie :similarTo ex:Freddie .
```

```
(everything :similarTo itself)
```



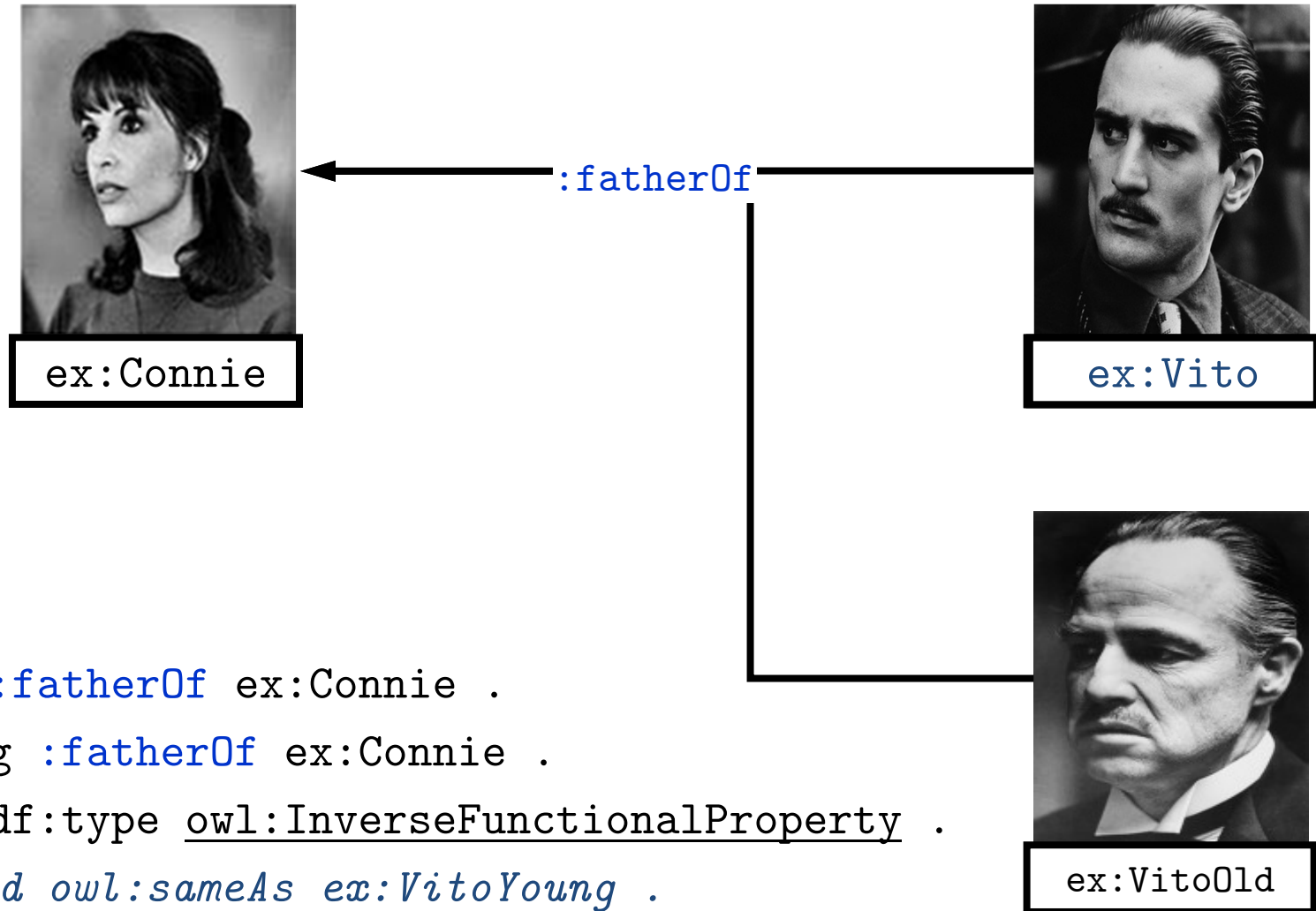
owl:FunctionalProperty



```
ex:Fredo :hasFather ex:VitoOld .  
ex:Fredo :hasFather ex:VitoYoung .  
:hasFather rdf:type owl:FunctionalProperty .  
⇒ ex:VitoOld owl:sameAs ex:VitoYoung .
```

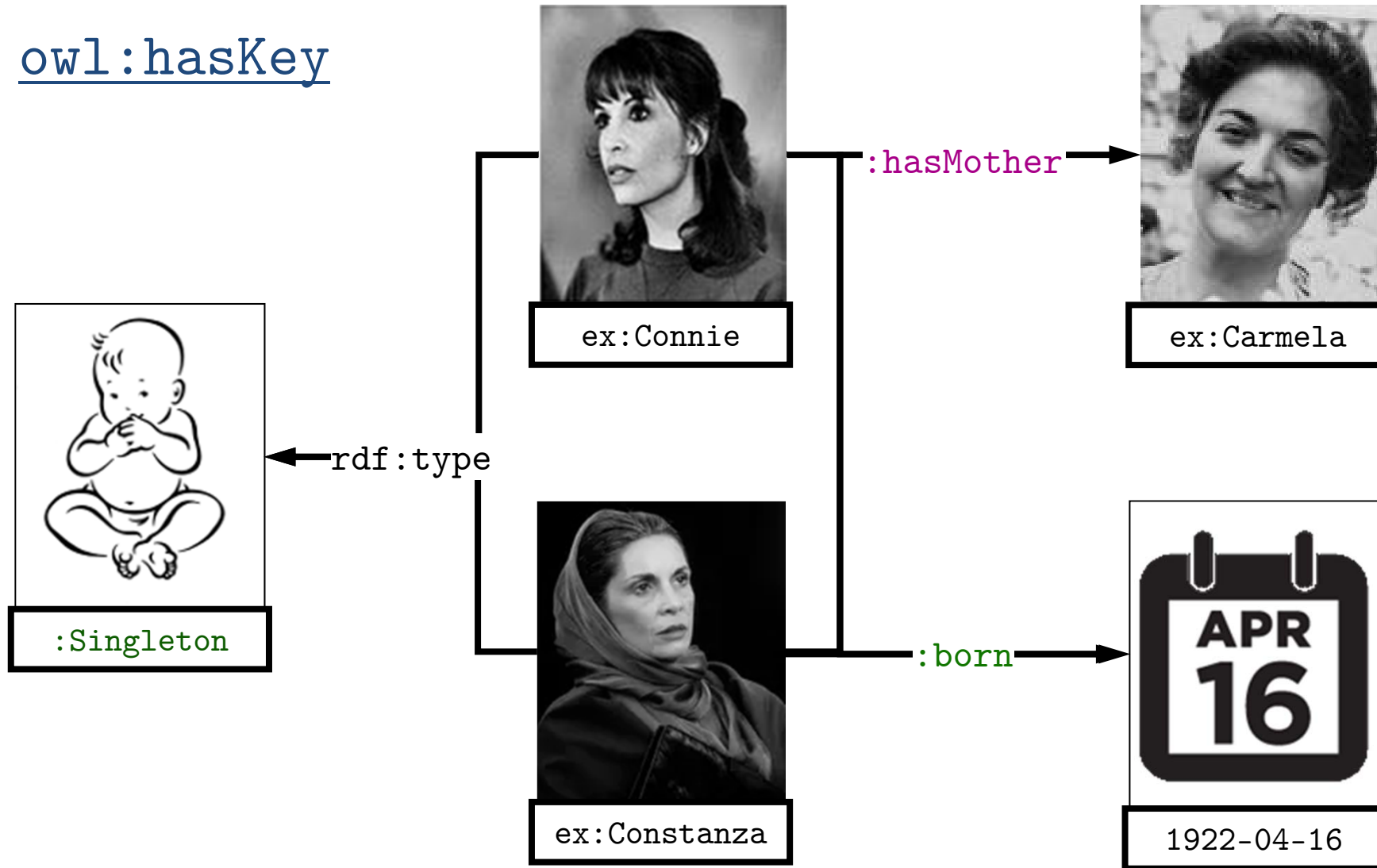


owl:InverseFunctionalProperty



```
ex:VitoOld :fatherOf ex:Connie .  
ex:VitoYoung :fatherOf ex:Connie .  
:fatherOf rdf:type owl:InverseFunctionalProperty .  
⇒ ex:VitoOld owl:sameAs ex:VitoYoung .
```

owl:hasKey

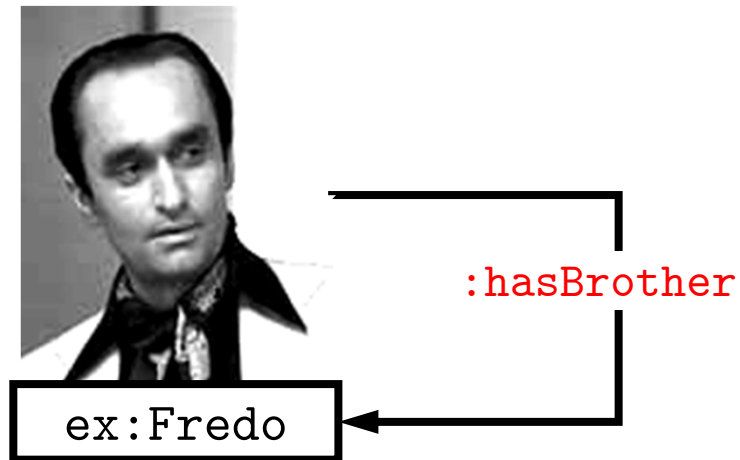


```

ex:Connie a :Singleton ; :hasMother ex:Carmela ; :born "1922-04-16"^^xsd:date .
ex:Constanza a :Singleton ; :hasMother ex:Carmela ; :born "1922-04-16"^^xsd:date .
:Singleton owl:hasKey ( :hasMother :born ) .
⇒ ex:Connie owl:sameAs ex:Constanza .
  
```



owl:IrreflexiveProperty



ex:Fredo :hasBrother ex:Fredo .

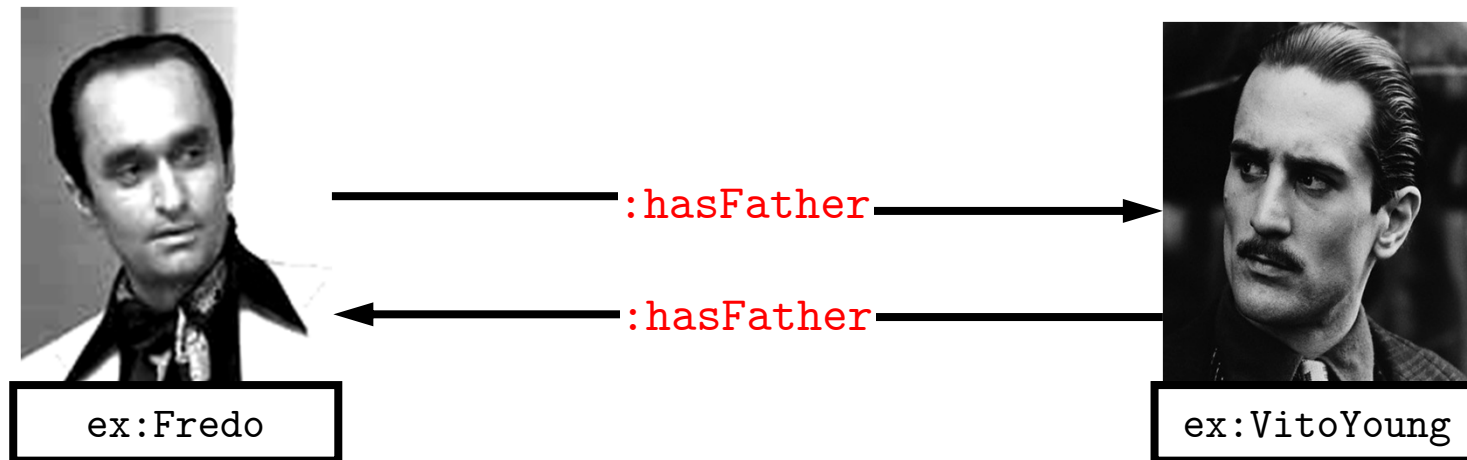
:hasBrother rdf:type owl:IrreflexiveProperty .

⇒ *FALSE*





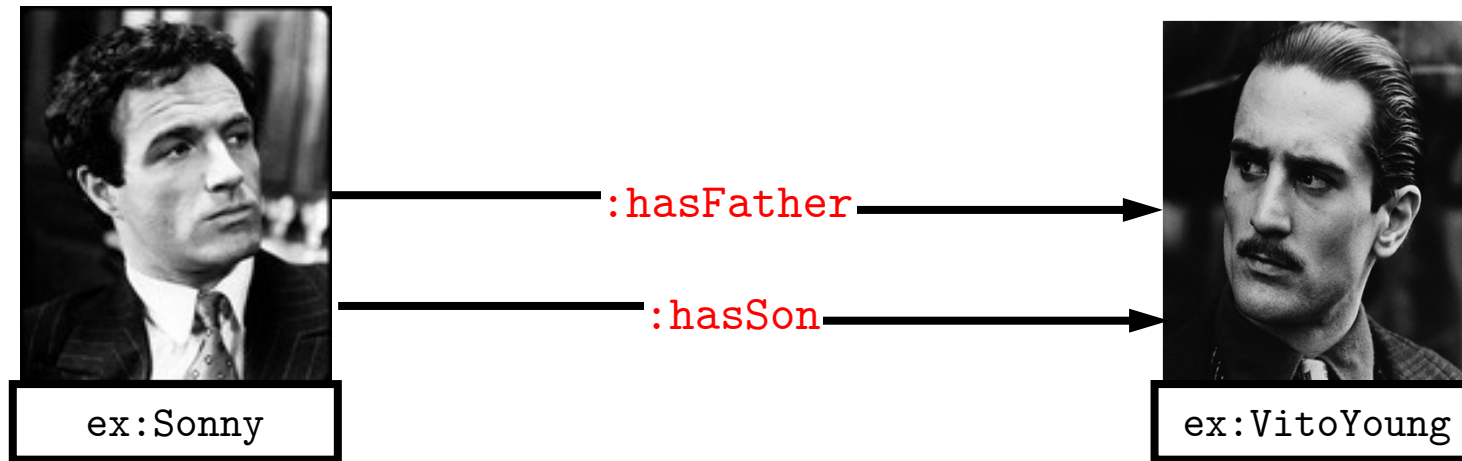
owl:AsymmetricProperty



```
ex:Fredo :hasFather ex:VitoYoung .  
ex:VitoYoung :hasFather ex:Fredo .  
:hasFather rdf:type owl:AsymmetricProperty .  
⇒ FALSE
```



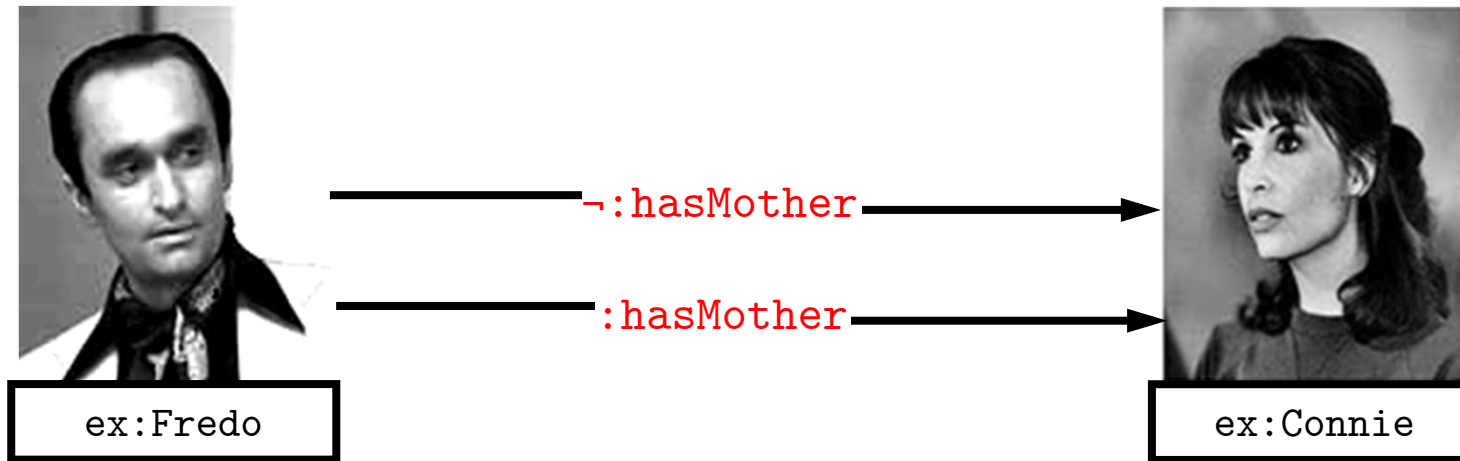
owl:disjointPropertyWith



```
ex:Sonny :hasFather ex:VitoYoung .
ex:VitoYoung :hasSon ex:Sonny .
:hasSon owl:disjointPropertyWith :hasFather .
⇒ FALSE
```



Negative property assertions



```
[ ] owl:sourceIndividual ex:Fredo ;  
    owl:assertionProperty :hasMother ;  
    owl:targetIndividual ex:Connie .
```

```
ex:Fredo :hasMother ex:Connie .
```

⇒ *FALSE*



Recap OWL property axioms

What would be the
owl:inverseOf the
property :fatherOf?

Name an owl:SymmetricProperty
for family relations?

Name an owl:TransitiveProperty
for family relations?

Give an owl:propertyChainAxiom
for :hasNiece?

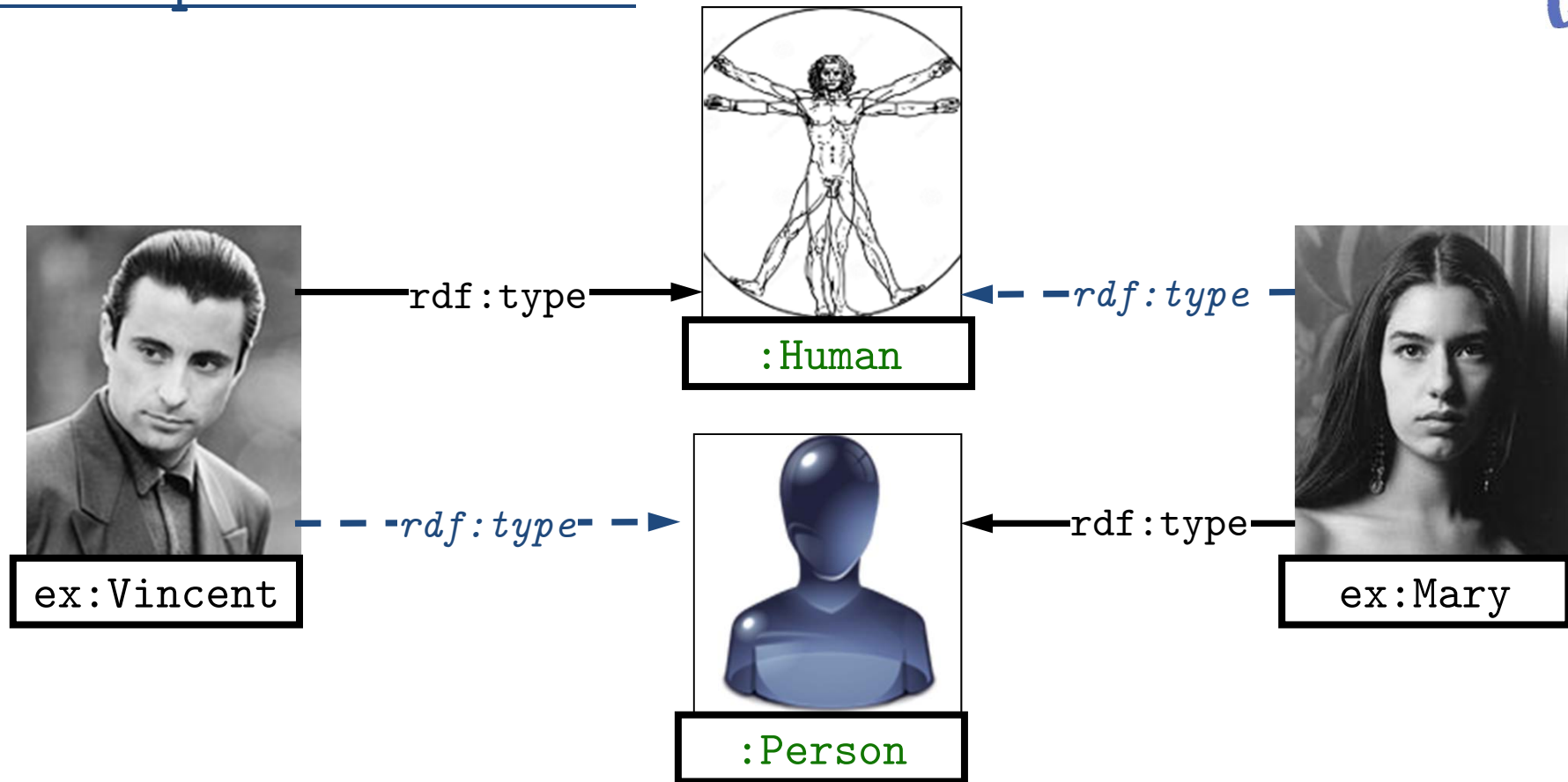
Name an owl:AsymmetricProperty
for family relations?

Name an owl:FunctionalProperty for family relations?

CLASS AXIOMS IN OWL



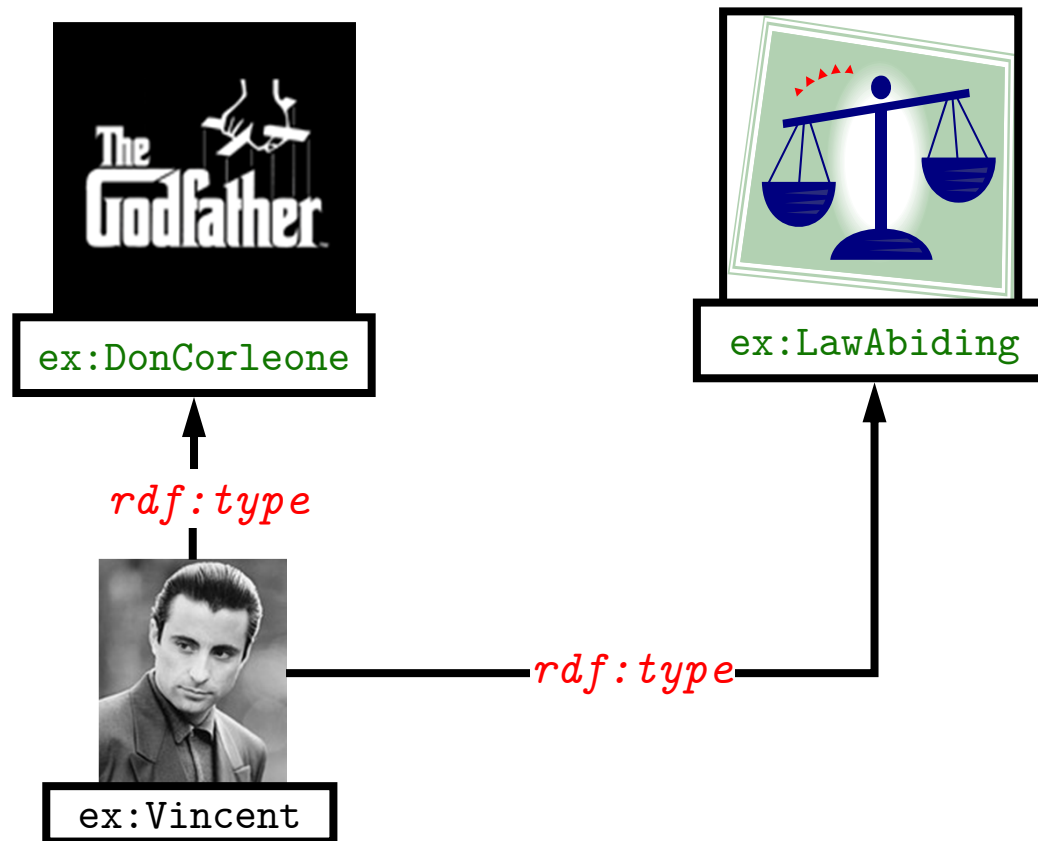
owl:equivalentClass



```
ex:Vincent rdf:type :Human .  
ex:Mary rdf:type :Person .  
:Human owl:equivalentClass :Person .  
⇒ ex:Vincent rdf:type :Person .  
ex:Mary rdf:type :Human .
```



owl:disjointWith



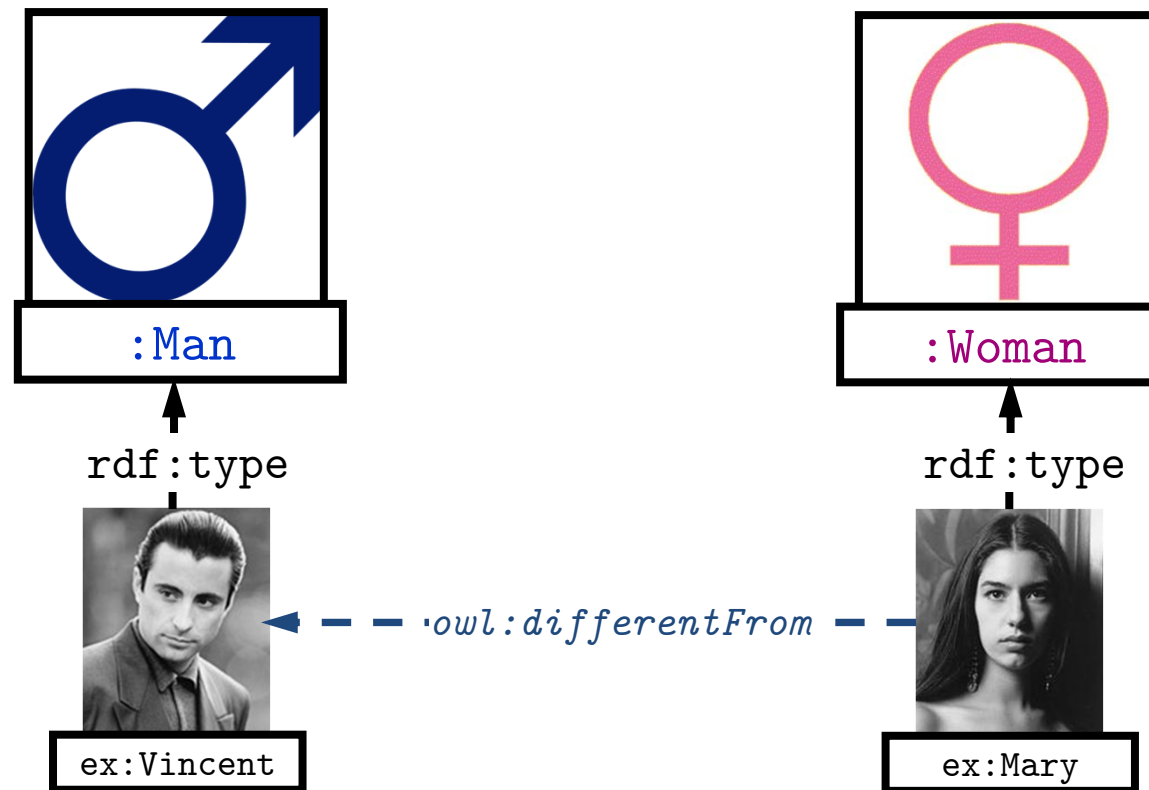
`ex:Vincent rdf:type ex:DonCorleone , ex:LawAbiding .`

`ex:DonCorleone owl:disjointWith ex:LawAbiding .`

$\Rightarrow$ **FALSE**



owl:disjointWith (ii)



```
ex:Vincent rdf:type :Man . ex:Mary rdf:type :Woman .  
:Man owl:disjointWith :Woman .  
⇒ ex:Mary owl:disjointWith ex:Vincent .
```

CLASS DEFINITIONS IN OWL

Things start to get a little more “tricky”

- Property axioms say something about properties
- Class axioms say something about classes
- Class definitions define new classes
 - e.g., the class of humans with at least one human child
 - often these classes are not named, but a class axiom is used to relate them to a named class
 - e.g., the class of human parent is equivalent to the class of humans with at least one human child
 - e.g., the class of human mother is a sub-class of the class of humans with at least one human child
- Will become more clear 😊

Some basic Description Logics symbols

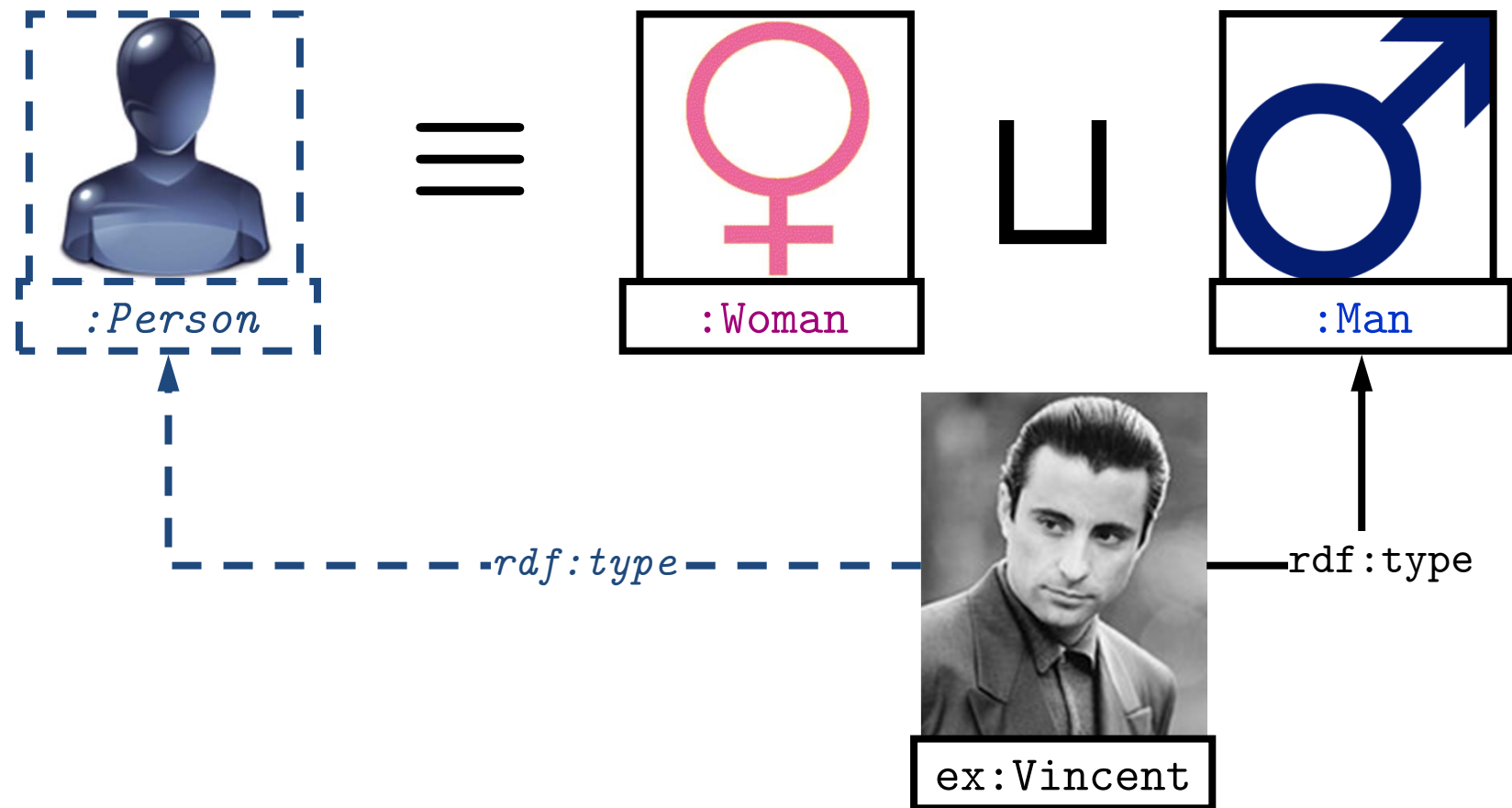
- **Description Logics (DL)** a theory underlying OWL
- ‘ $\equiv$ ’ for **equivalent** class or property
 - $\text{:Person} \equiv \text{:Human}$
 - $\text{:hasChild} \equiv \text{:parentOf}$
- ‘ $\sqsubseteq$ ’ for **sub**-class or sub-property
 - $\text{:Woman} \sqsubseteq \text{:Person}$
 - $\text{:hasSon} \sqsubseteq \text{:hasChild}$

Some built in classes and properties

- `owl:Thing`: the class of “everything”
 - Relation to `rdfs:Resource` is complicated ... let's not get into that now 😊
 - Symbol: '`T`'
- `owl:Nothing`: the empty class
 - Symbol: '`⊥`'



owl:unionOf (⊔)



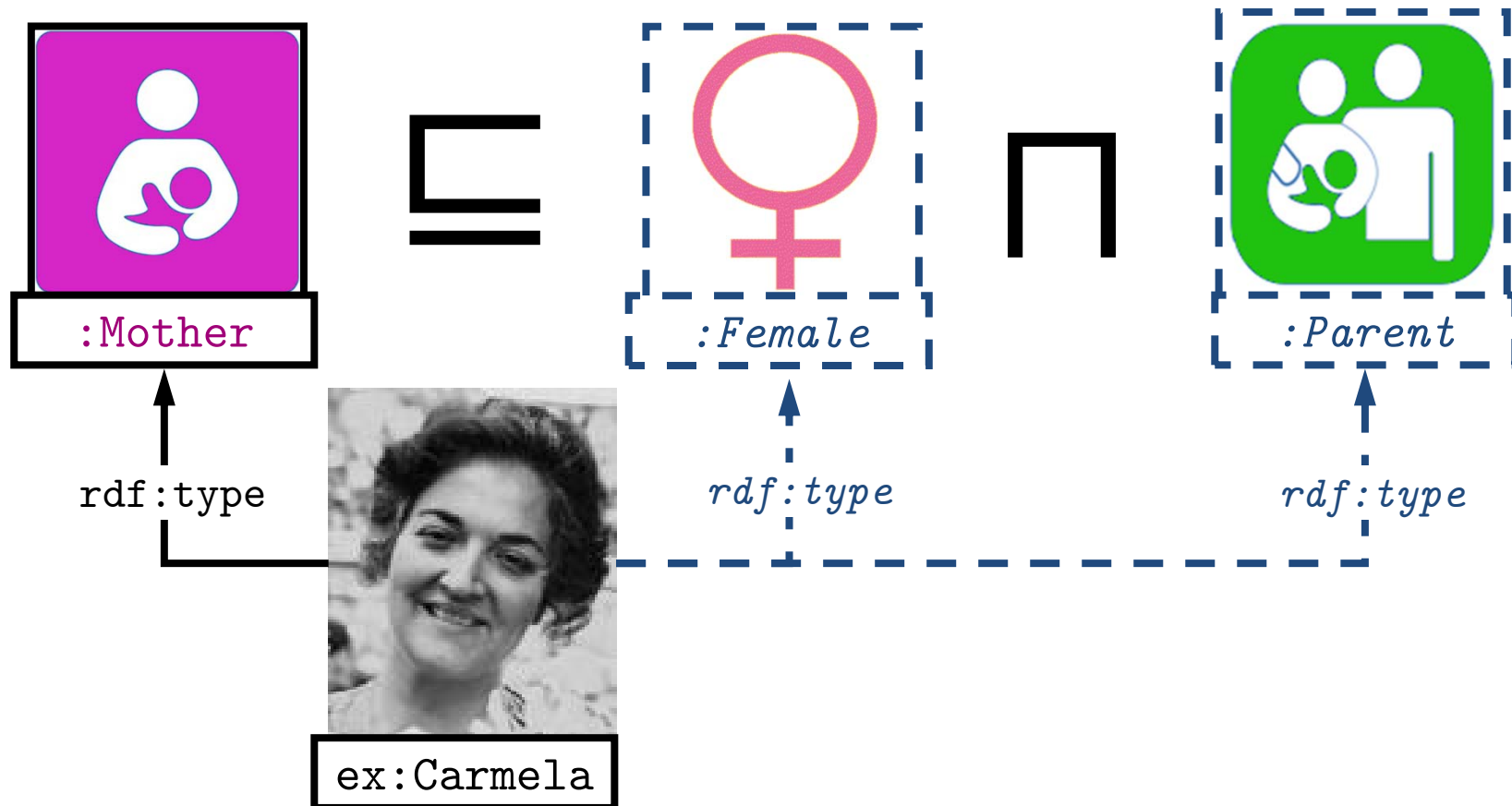
`ex:Vincent rdf:type :Man .`

`:Person owl:equivalentClass [ owl:unionOf ( :Woman :Man ) ]`

$\Rightarrow$ `ex:Vincent rdf:type :Person .`



owl:intersectionOf ($\sqcap$) [i]



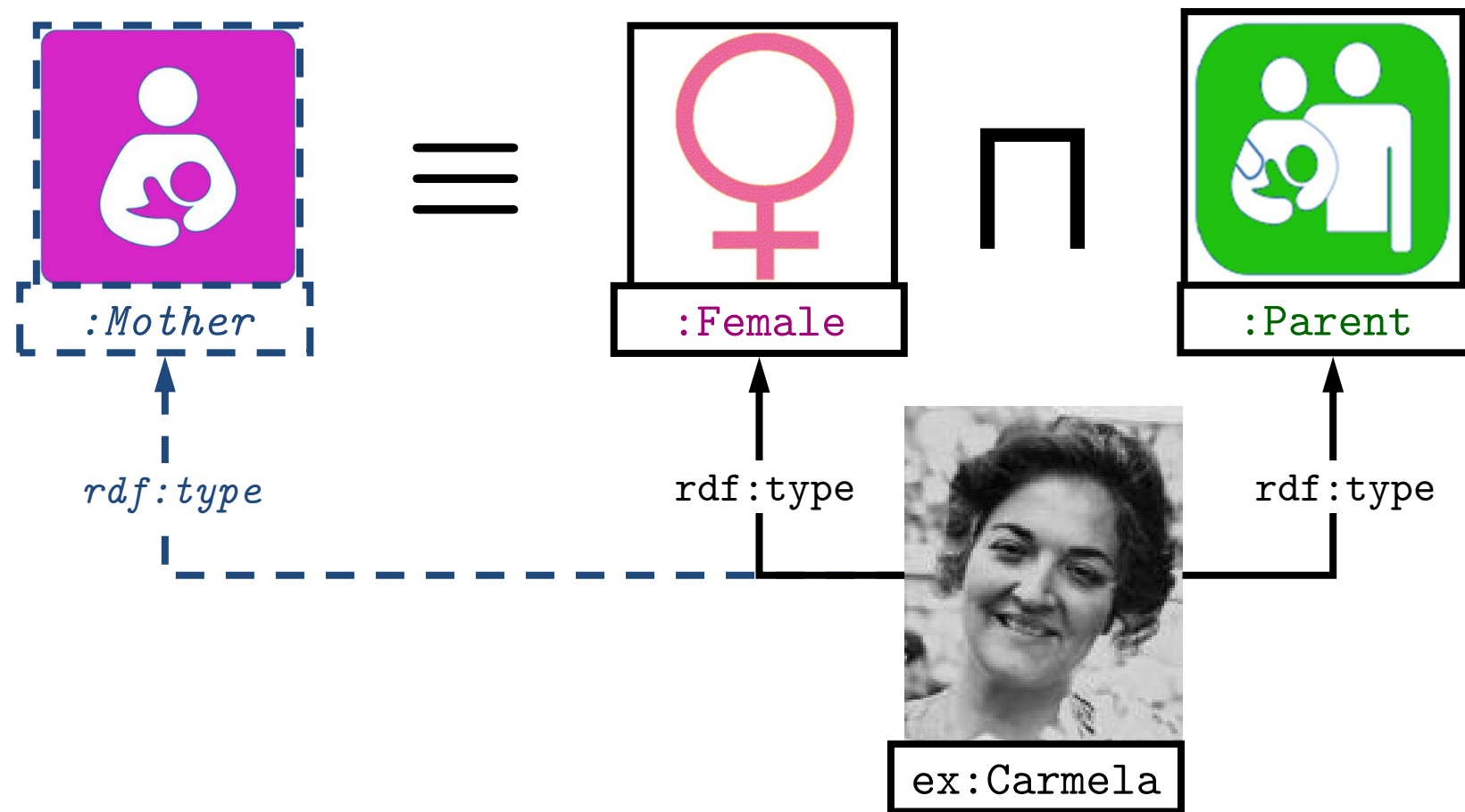
ex:Carmela *rdf:type* *:Mother* .

:Mother *rdfs:subClassOf* [owl:intersectionOf (*:Female* *:Parent*)]

$\Rightarrow$ *ex:Carmela* *rdf:type* *:Female*, *:Parent* .



owl:intersectionOf ($\sqcap$) [ii]



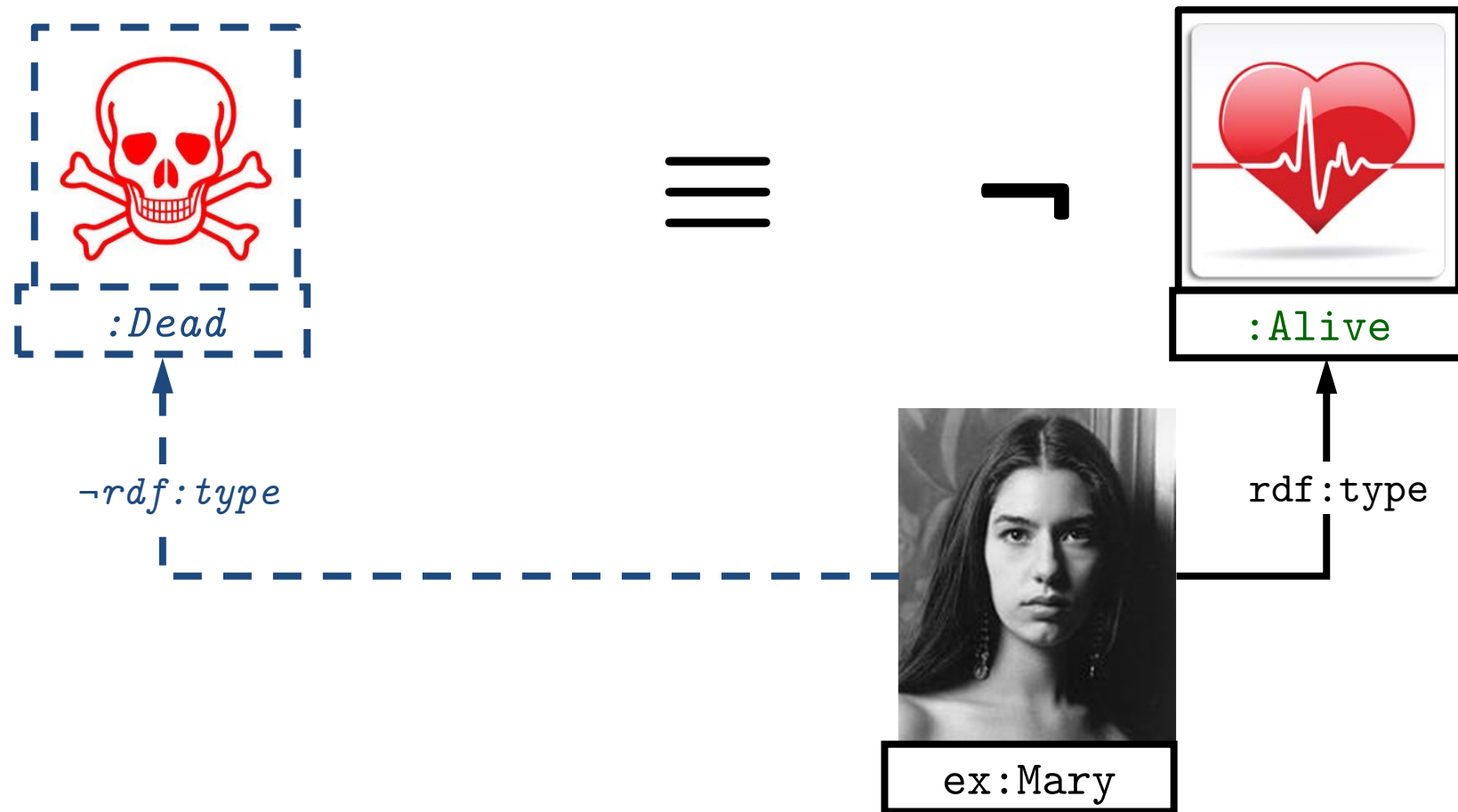
`ex:Carmela rdf:type :Female , :Parent .`

`:Mother owl:equivalentClass [ owl:intersectionOf ( :Female :Parent ) ]`

$\Rightarrow$ `ex:Carmela rdf:type :Mother .`



owl:complementOf (¬) [i]



ex:Mary rdf:type :Alive .

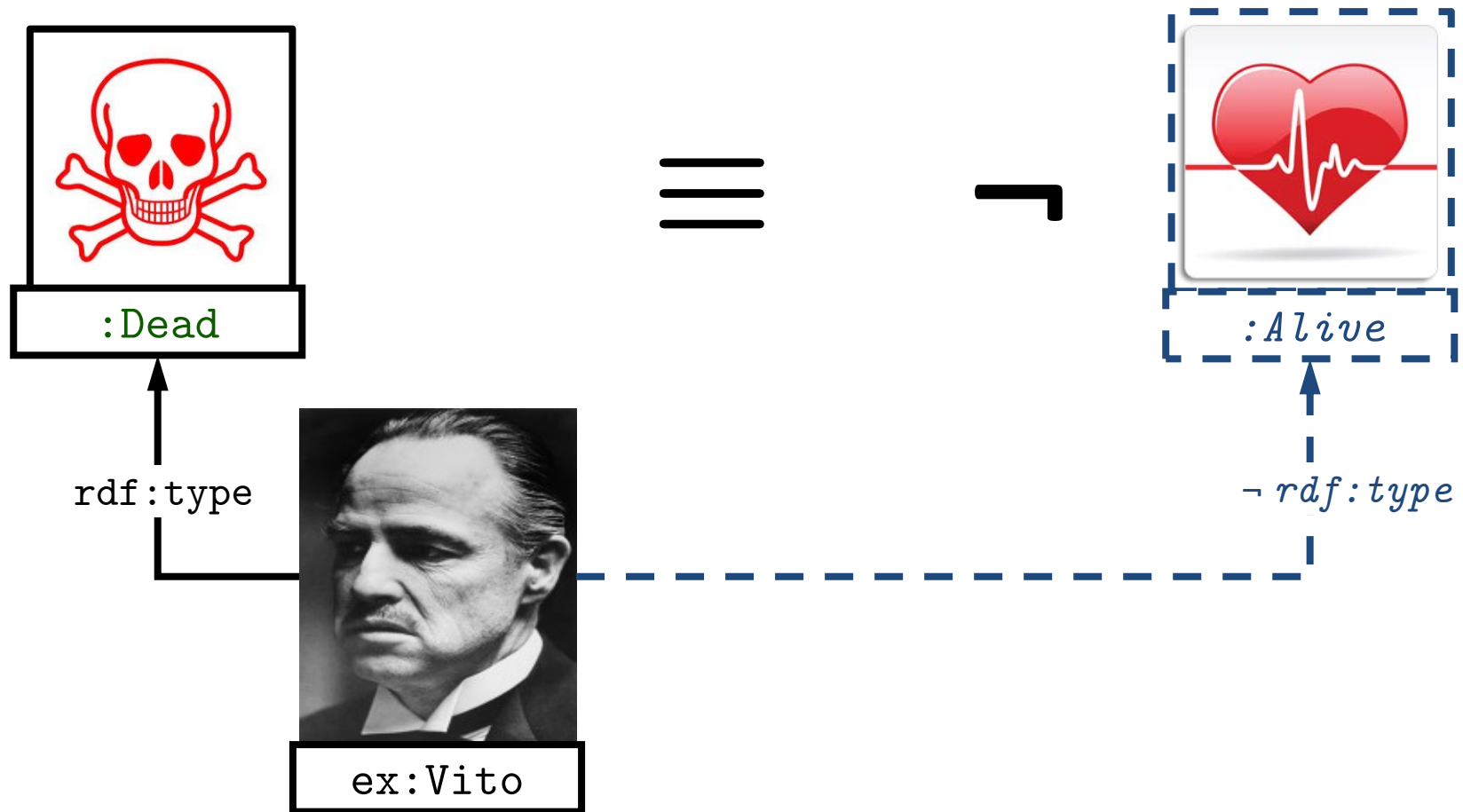
:Dead owl:equivalentClass [owl:complementOf :Alive]

⇒ [] owl:sourceIndividual ex:Mary ; owl:targetProperty rdf:type ;
owl:targetIndividual :Dead .





owl:complementOf ($\neg$) [ii]



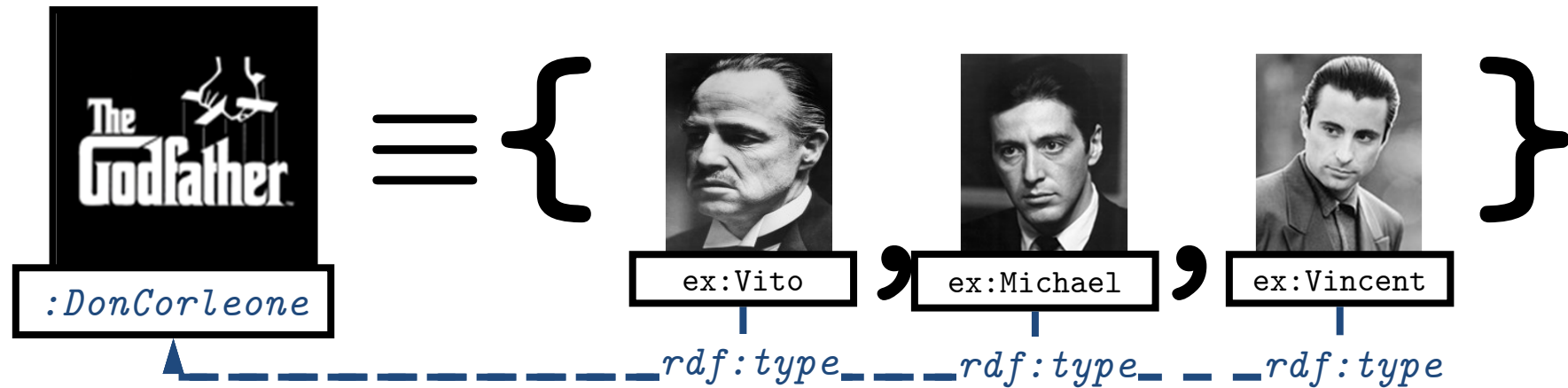
`ex:Vito rdf:type :Dead .`

`:Dead owl:equivalentClass [ owl:complementOf :Alive ]`

$\Rightarrow$ `[] owl:sourceIndividual ex:Vito ; owl:targetProperty rdf:type ; owl:targetIndividual :Alive .`



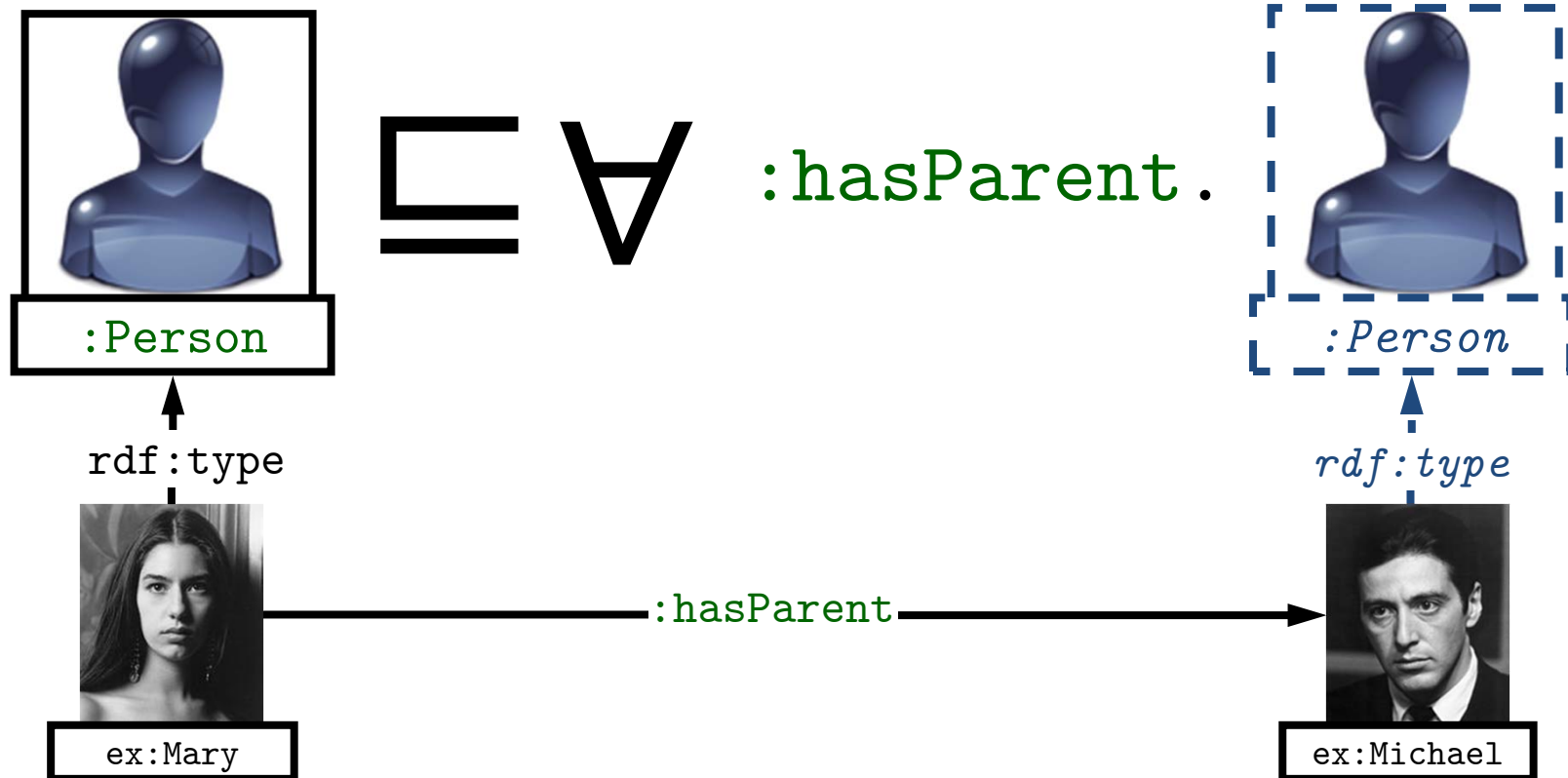
owl:oneOf ({})



```
:DonCorleone owl:equivalentClass  
  [ owl:oneOf (ex:Vito ex:Michael ex:Vincent) ]  
⇒ ex:Vito rdf:type :DonCorleone .  
⇒ ex:Michael rdf:type :DonCorleone .  
⇒ ex:Vincent rdf:type :DonCorleone .
```



owl:allValuesFrom ($\forall$)



```
ex:Mary rdf:type :Person ; :hasParent ex:Michael .
```

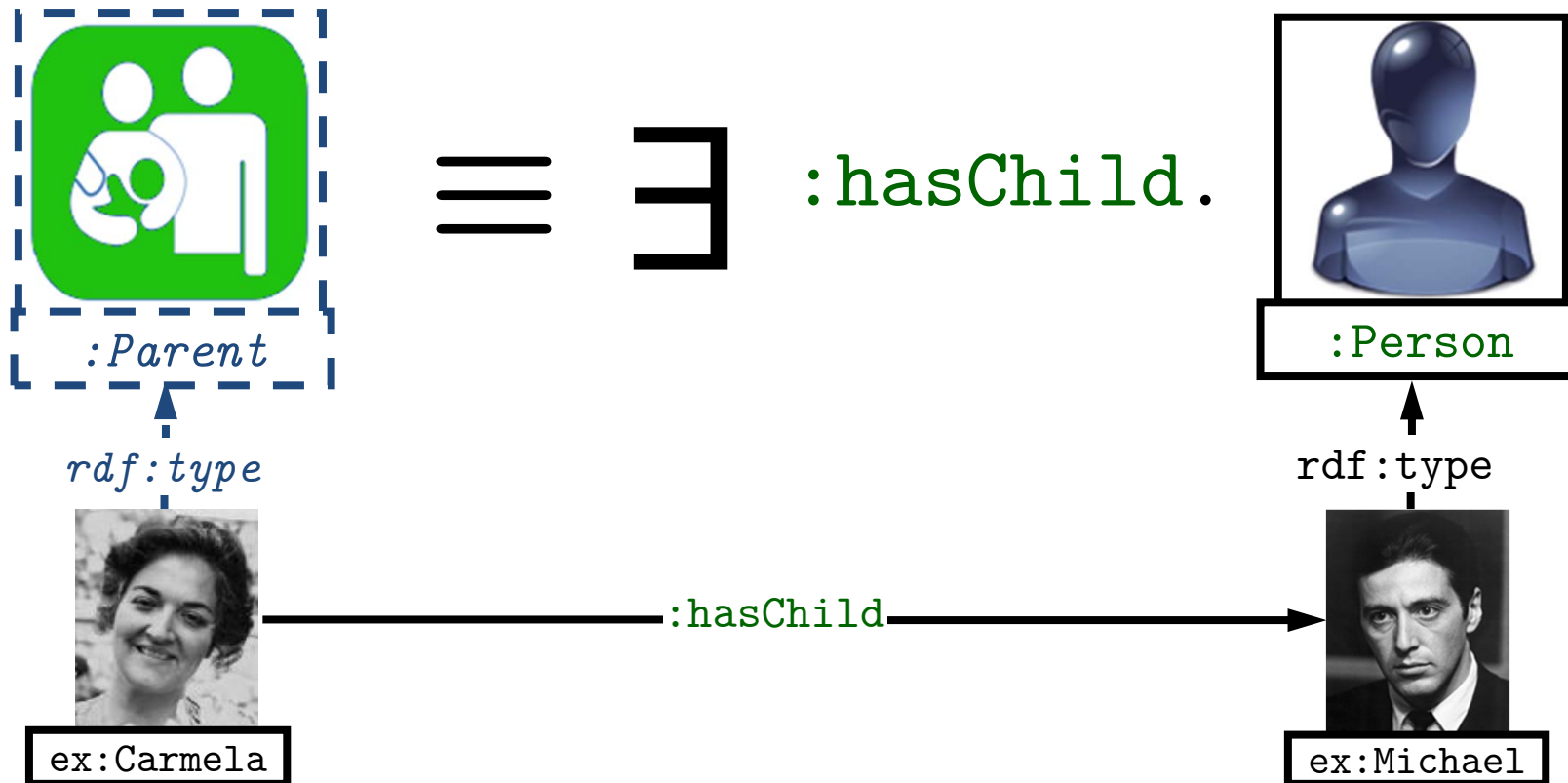
```
:Person rdfs:subClassOf
```

```
[ owl:allValuesFrom :Person ; owl:onProperty :hasParent ]
```

```
⇒ ex:Michael rdf:type :Person .
```



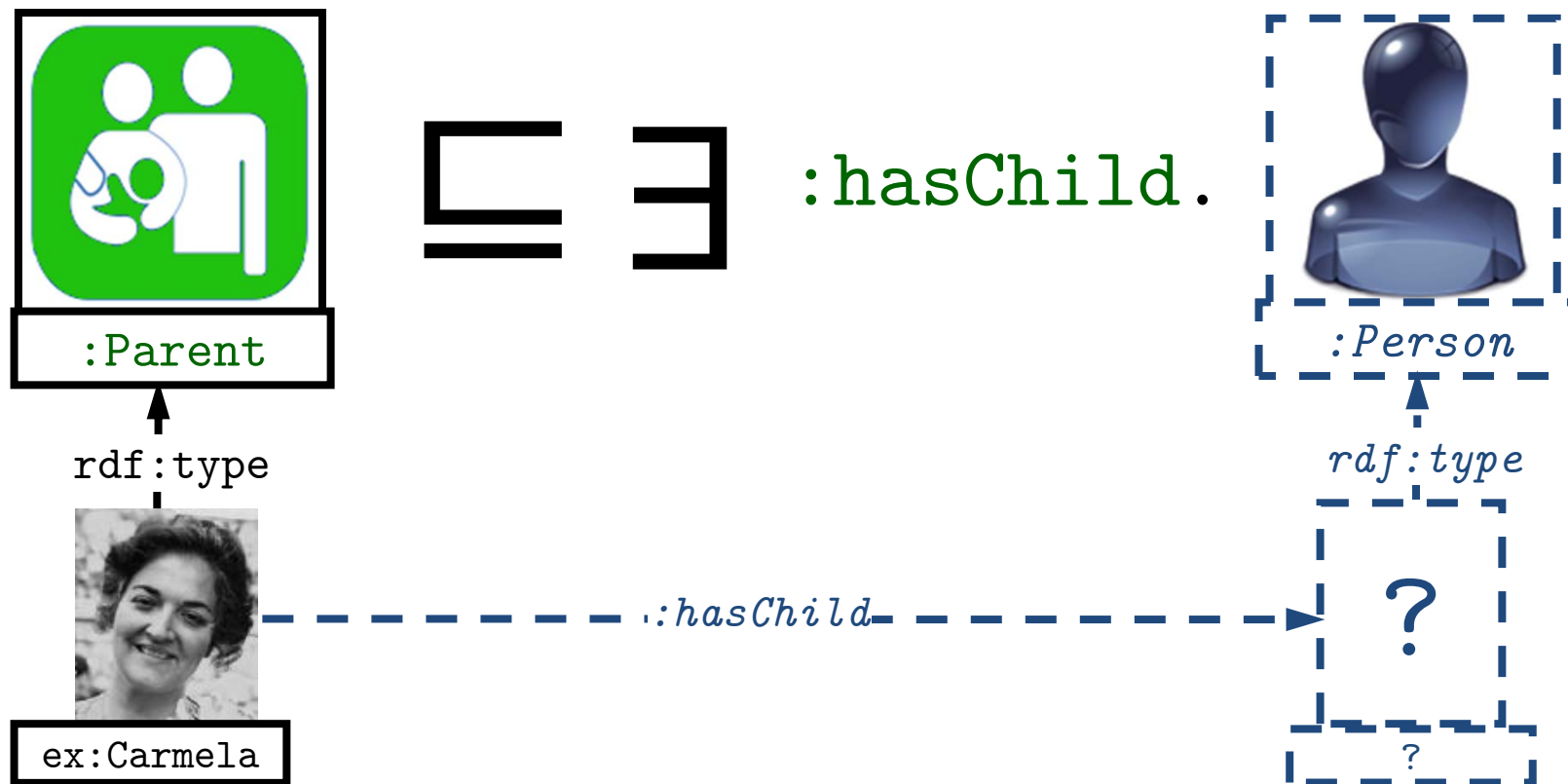
owl:someValuesFrom ($\exists$) [i]



```
ex:Carmela :hasChild ex:Michael . ex:Michael rdf:type :Person .
:Parent owl:equivalentClass
  [ owl:someValuesFrom :Person ; owl:onProperty :hasChild ]
⇒ ex:Carmela rdf:type :Parent .
```



owl:someValuesFrom (∃) [ii]



ex:Carmela rdf:type :Parent .

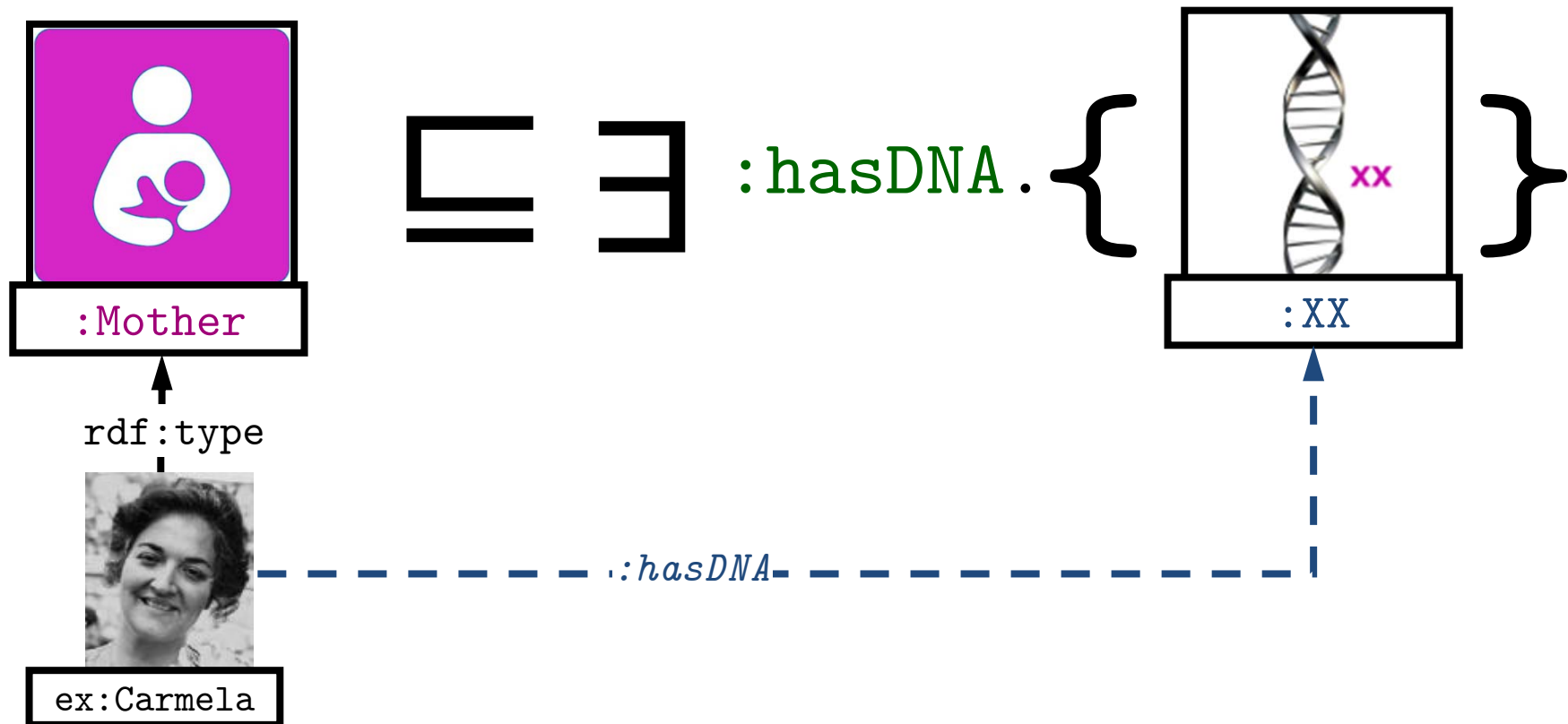
:Parent rdfs:subClassOf

[owl:someValuesFrom :Person ; owl:onProperty :hasChild] .

⇒ ex:Mary :hasChild _:someone . _:someone rdf:type :Person .



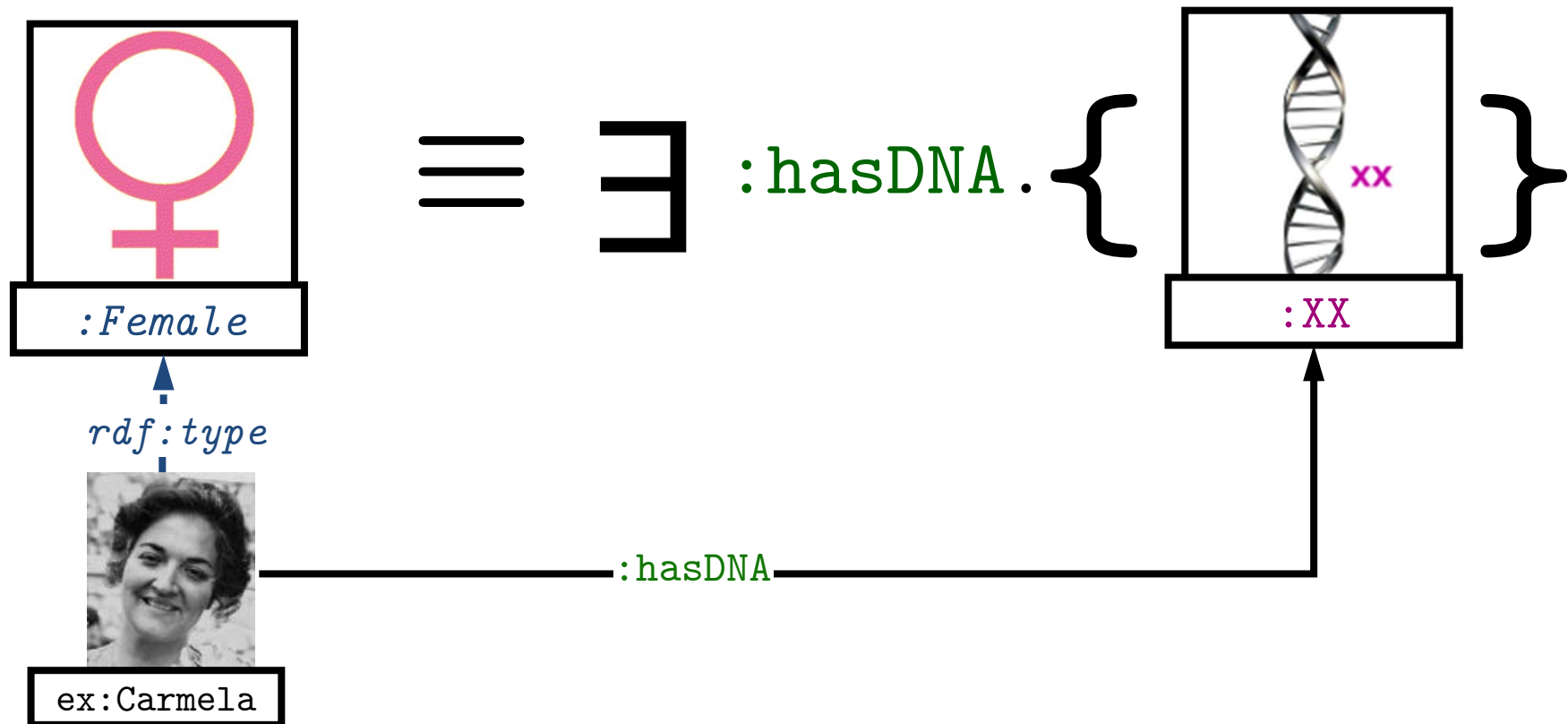
owl:hasValue (∃P.{x}) [i]



```
ex:Carmela rdf:type :Mother .  
:Mother rdfs:subClassOf  
  [ owl:hasValue :XX ; owl:onProperty :hasDNA ]  
⇒ ex:Carmela :hasDNA :XX .
```



owl:hasValue (∃P.{x}) [ii]



`ex:Carmela :hasDNA :XX .`

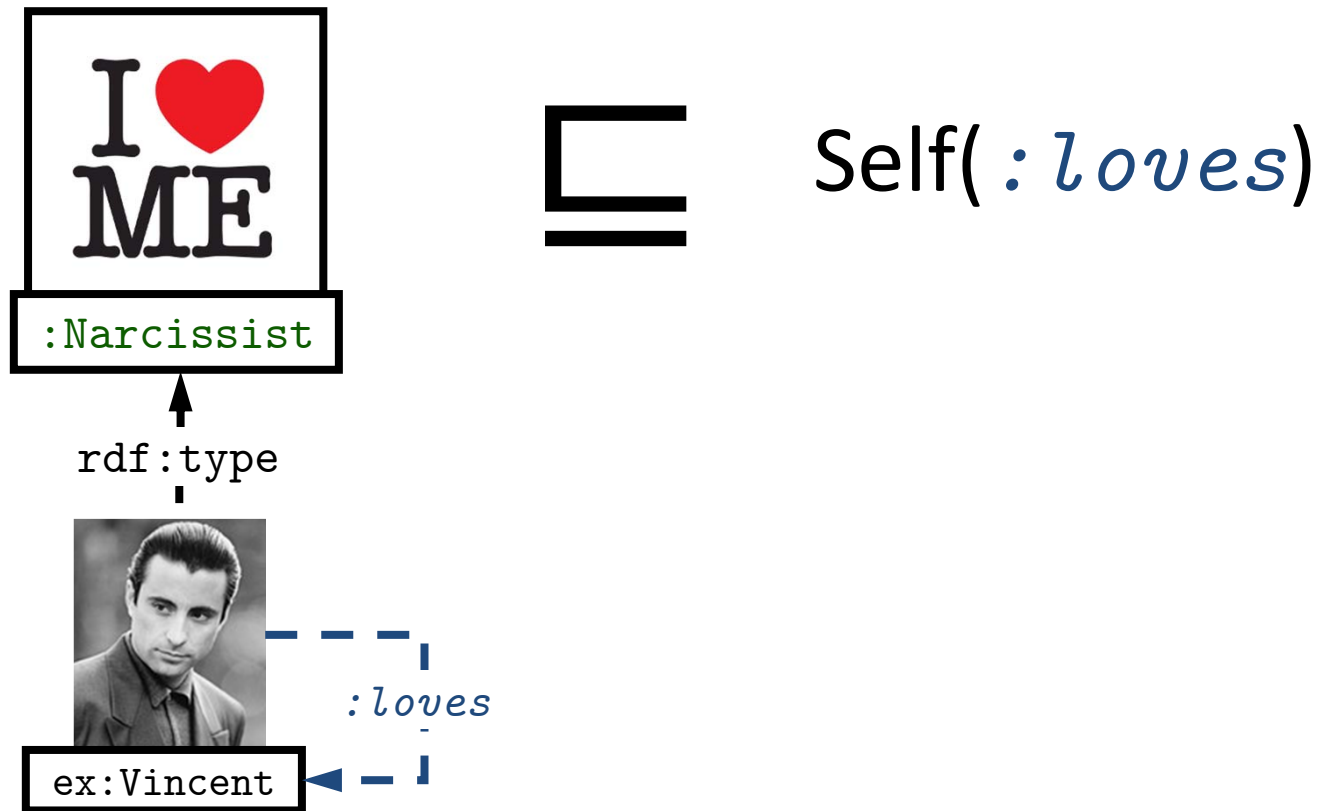
`:Female owl:equivalentClass`

`[ owl:hasValue :XX ; owl:onProperty :hasDNA ]`

$\Rightarrow$ `ex:Carmela rdf:type :Female .`



owl:hasSelf (Self) [i]



```
ex:Vincent rdf:type :Narcissist .
```

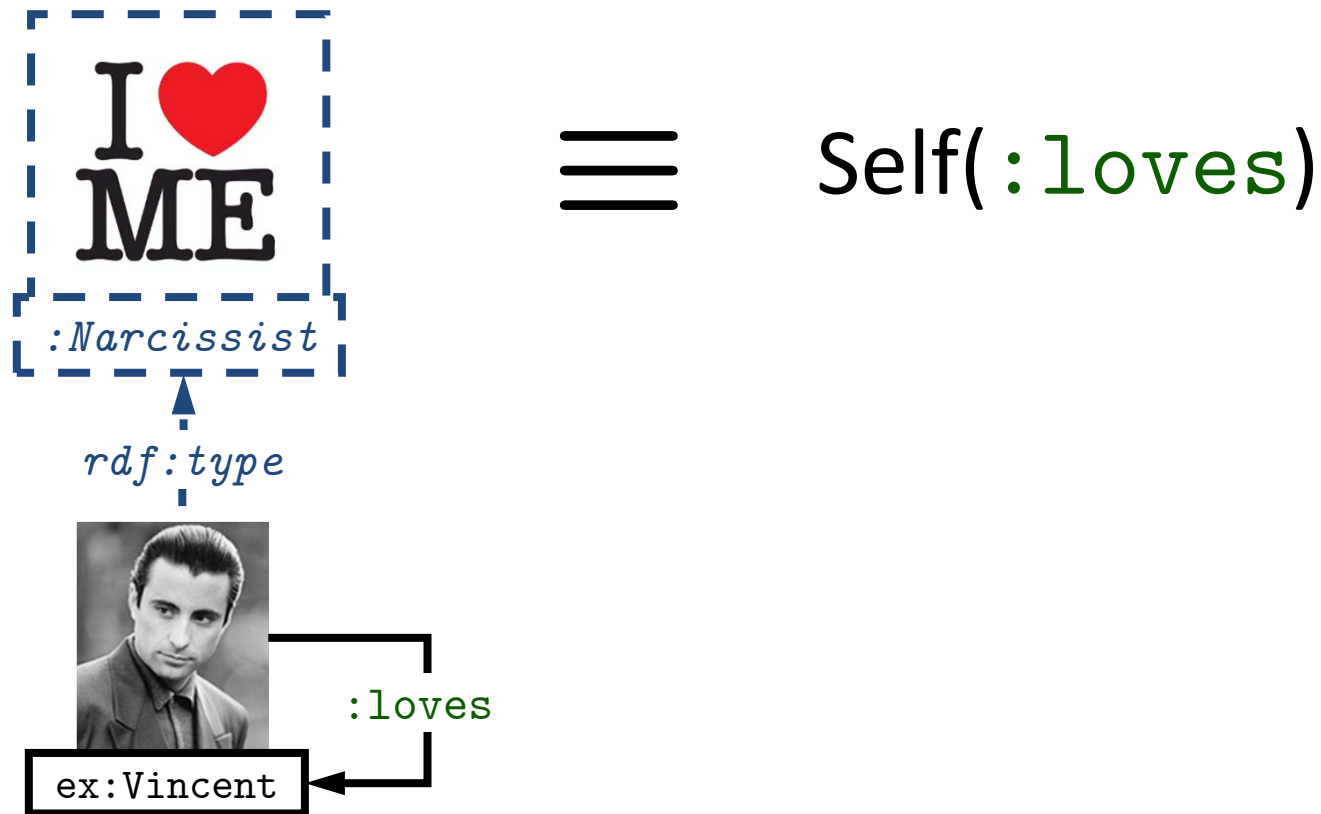
```
:Narcissist rdfs:subClassOf
```

```
[ owl:hasSelf true ; owl:onProperty :loves ]
```

```
⇒ ex:Vincent :loves ex:Vincent .
```



owl:hasSelf (Self) [ii]



```
ex:Vincent :loves ex:Vincent .  
:Narcissist owl:equivalentClass  
  [ owl:hasSelf true ; owl:onProperty :loves ]  
⇒ ex:Vincent rdf:type :Narcissist .
```



Cardinality restrictions ($\geq$, $\leq$, $=$)

- Define a class with a given number of values for a property:

- **Exact:**

- `:Person  $\sqsubseteq$  =2 (:hasParent)`
- `:Parent rdfs:subClassOf [ owl:cardinality 2 ; owl:onProperty :hasParent ] .`

- **Max:**

- `:Monogamist  $\sqsubseteq$   $\leq$  1 (:currentSpouse)`
- `:Monogamist rdfs:subClassOf [ owl:maxCardinality 1 ; owl:onProperty :currentSpouse ] .`

- **Min:**

- `:Parent  $\sqsubseteq$   $\geq$  1 (:hasChild)`
- `:Parent owl:equivalentClass [ owl:minCardinality 1 ; owl:onProperty :hasChild ] .`



Qualified cardinality restrictions ($\geq$, $\leq$, $=$)

- Define a class with a given number of values from a given class for a property:

- **Exact:**

- `:Person  $\equiv$  =2 (:hasParent.Person)`
- `:Parent owl:equivalentClass [ owl:qualifiedCardinality 2 ; owl:onProperty :hasParent ; owl:onClass :Person ] .`
- Now the values in question must be people!

- Analogous versions of **Max** and **Min**.

Recap OWL class axioms/definitions

A class `:HumanParent` might be equivalent to the `owl:unionOf` which classes?

$A \sqsubseteq (B \sqcap \exists P.C) ?$

What is the difference/relation between `owl:complementOf` and `owl:disjointWith`

Give an example use of `owl:allValuesFrom` for family relations

Give an example use of `owl:someValuesFrom` for `:Uncle`.

How might we codify the semantics of a class `:OnlySon` in OWL?

How might we codify the semantics of a class `:InSameSexMarriage` in OWL?

Be careful what you define ...



- Tom Hagen, the adopted son of Vito
- We probably meant *biologicalParent*, etc.



Slides are examples, not definitions



`:DonCorleone owl:equivalentClass`

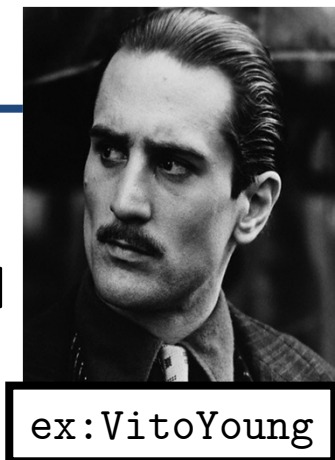
`[ owl:oneOf (ex:Vito ex:Michael ex:Vincent) ]`

$\Rightarrow$ `ex:Vito rdf:type :DonCorleone .`

$\Rightarrow$ `ex:Michael rdf:type :DonCorleone .`

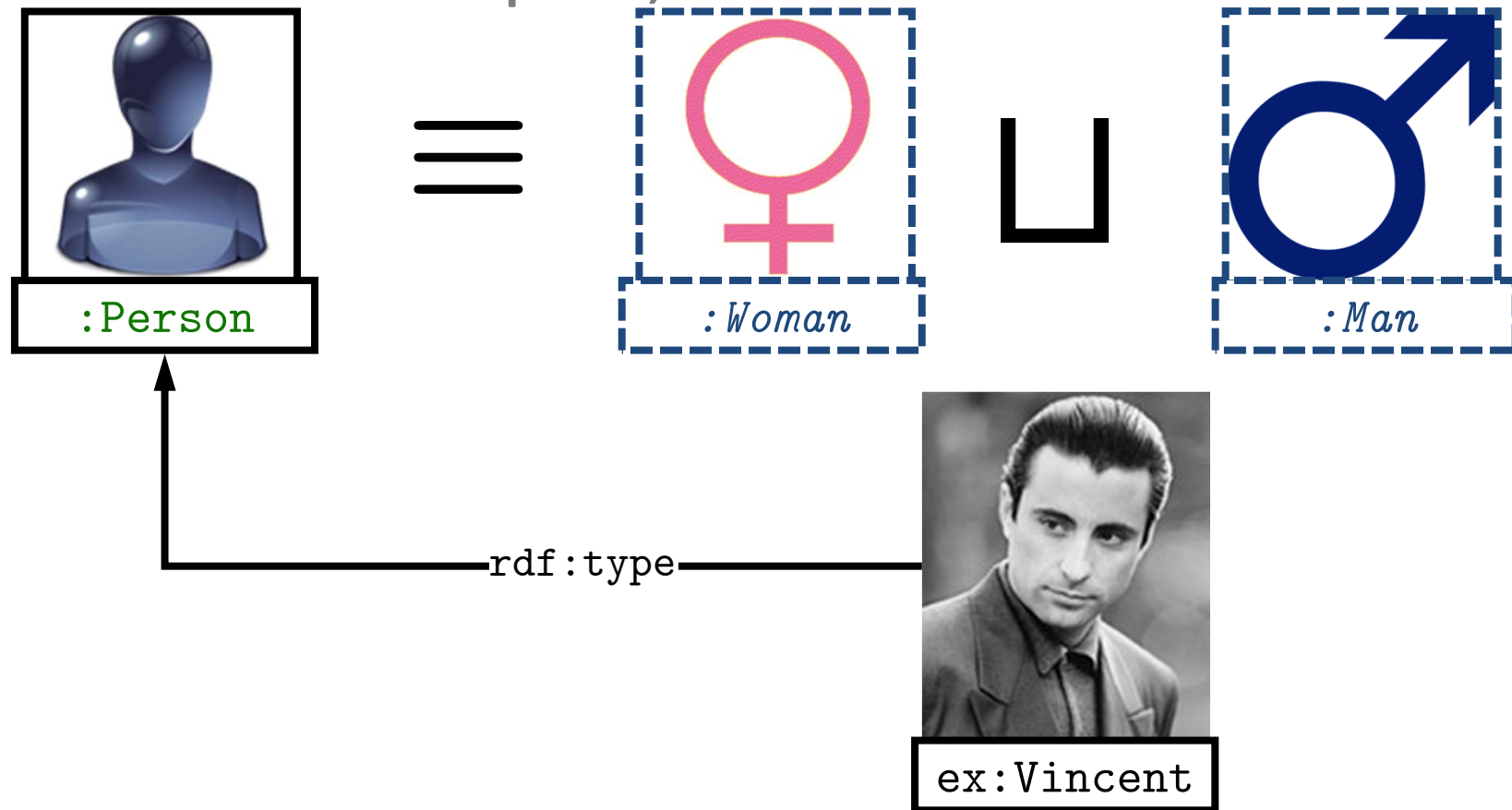
$\Rightarrow$ `ex:Vincent rdf:type :DonCorleone .`

$\Rightarrow$... `ex:VitoYoung` must be the same as one of the three guys!





Slides are examples, not definitions



```
ex:Vincent rdf:type :Person .
```

```
:Person owl:equivalentClass [ owl:unionOf ( :Woman :Man ) ]
```

⇒ ... ex:Vincent must be either of type *:Woman* or *:Man*

RECAP

(1) Data, (2) Rules/Ontologies, (3) Query

INPUT: “ (x, partOf, y) ”

DATA:

<http://ex.org/Ireland>

Ireland



(Ireland, partOf, Europe)
(Ireland, a, Country)
(Ireland, capital, Dublin)

<http://ex.org/Dublin>

Dublin



(Dublin, population, 1000000)

RULES: $(a, \text{capital}, b) \rightarrow (b, \text{partOf}, a)$
 $(c, \text{partOf}, d), (d, \text{partOf}, e) \rightarrow (c, \text{partOf}, e)$

OUTPUT: $\{(x \mapsto \text{Ireland}, y \mapsto \text{Europe}), (x \mapsto \text{Dublin}, y \mapsto \text{Ireland})$
 $(x \mapsto \text{Dublin}, y \mapsto \text{Europe})\}$

RDF(S)/OWL: OWA, no UNA

- RDF(S)/OWL
 - Take an Open World Assumption
 - Do not take a Unique Name Assumption
 - On the Web:
 - information is incomplete
 - having one name for everything would require too much coordination

OWL Features: (In)Equality

- `owl:sameAs`: two names refer to same thing
- `owl:differentFrom`: two names refer to different things

OWL Features: Property Axioms

- `owl:equivalentProperty`: two properties are synonymous
- `owl:inverseOf`: two properties synonymous but in opposite directions
- `owl:SymmetricProperty`: properties that always hold in both directions; i.e., if (a,P,b) then (b,P,a)
- `owl:TransitiveProperty`: if (a,P,b) and (b,P,c) then (a,P,c)

OWL Features: Property Axioms

- `owl:propertyChainAxiom`: define a property as a chain of other properties
- `owl:ReflexiveProperty`: a class of properties that relate everything to itself
- `owl:FunctionalProperty`: if (a, P, x) and (b, P, x) then $x = y$
- `owl:InverseFunctionalProperty`: if (x, P, a) and (y, P, a) then $x = y$
- `owl:hasKey`: define complex keys for instances of a given class

OWL Features: Property Axioms

- `owl:IrreflexiveProperty`: can never relate x to x
- `owl:AsymmetricProperty`: can never relate x to y and y to x
- `owl:disjointPropertyWith`: two properties that can never relate the same two things
- `Negative property assertions`: say that something is false

OWL Features: Class Axioms

- `owl:equivalentClass`: relates two classes that are synonymous
- `owl:disjointWith`: relates two classes that cannot share any instances

OWL Features: Class Definitions

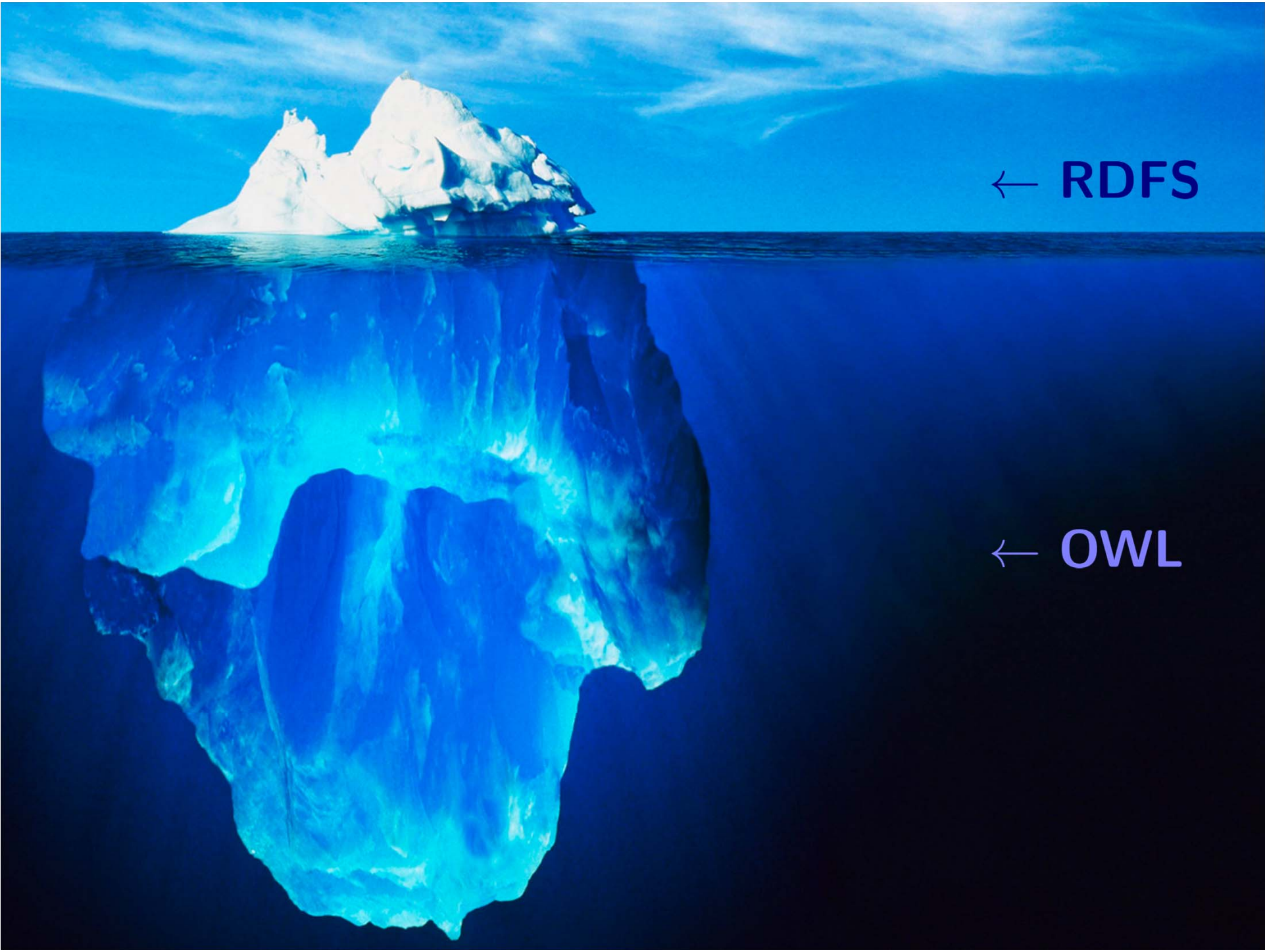
- `owl:unionOf`: defines a class that's the union of one or more other classes
- `owl:intersectionOf`: defines a class that's the intersection of one or more other classes
- `owl:complementOf`: defines a class that's the complement of another class
- `owl:oneOf`: defines a class by enumerating precisely all of its instances

OWL Features: Class Definitions

- `owl:allValuesFrom`: defines a class that contains all things that have values for a given property that are all of a certain type
- `owl:someValuesFrom`: defines a class that contains all things that have at least one value for a given property of a certain type
- `owl:hasValue`: defines a class that contains all things that have a specific value for a given property
- `owl:hasSelf`: defines a class that contains all things that have themselves as a value for a given property
- **Cardinality restrictions** define classes with a given number of values for a property
- **Qualified cardinality restrictions** define classes with a given number of values for a property in a given class

Some Description Logic symbols

- $\sqsubseteq$: sub-class/-property
- $\equiv$: equivalent class/property
- $\sqcup$: union
- $\sqcap$: intersection
- $\top$: top (class of everything)
- $\perp$: bottom (empty class)
- $\exists$: exists (someValuesFrom/hasValue)
- $\forall$: for all (allValuesFrom)
- $\neg$: not (complement, negation)
- $^{-}$ (superscript minus): inverse property
- $\{ \}$: enumeration (owl:oneOf)
- *Self*, *Trans*, *Dom*, etc.: where symbols not available

An iceberg floating in a deep blue ocean under a clear blue sky. The visible tip of the iceberg is small and jagged, while the submerged part is much larger and more complex. The water is a deep blue, and the sky is a lighter blue with some wispy clouds. The horizon line is clearly visible, separating the water from the sky.

← **RDFS**

← **OWL**

Questions?

