CC7220-1
La Web de Datos
Primavera 2024

Lecture 6: Web Ontology Language (OWL) [I]

Aidan Hogan
aidhog@gmail.com

Last time …

Semantic Web: Logic

Data:

Logic: “$(b$,capital,$a) \rightarrow (a$,partOf,$b)$”
“$(a$,partOf,$b)$, $(b$,partOf,$c) \rightarrow (a$,partOf,$c)$”

Query: “$(x$,partOf,$y)$?”

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 Schema: RDFS

Class c is a **sub-class** of Class d

If (x, rdf:type, c) then (x, rdf:type, d)

Property p is a **sub-property** of q

If (x, p, y) then (x, q, y)

Property p has **domain** class c

If (x, p, y) then (x, rdf:type, c)

Property p has **range** class c

If (x, p, y) then (y, rdf:type, c)

Today's topic ...

* More or less

Semantic Web: Logic

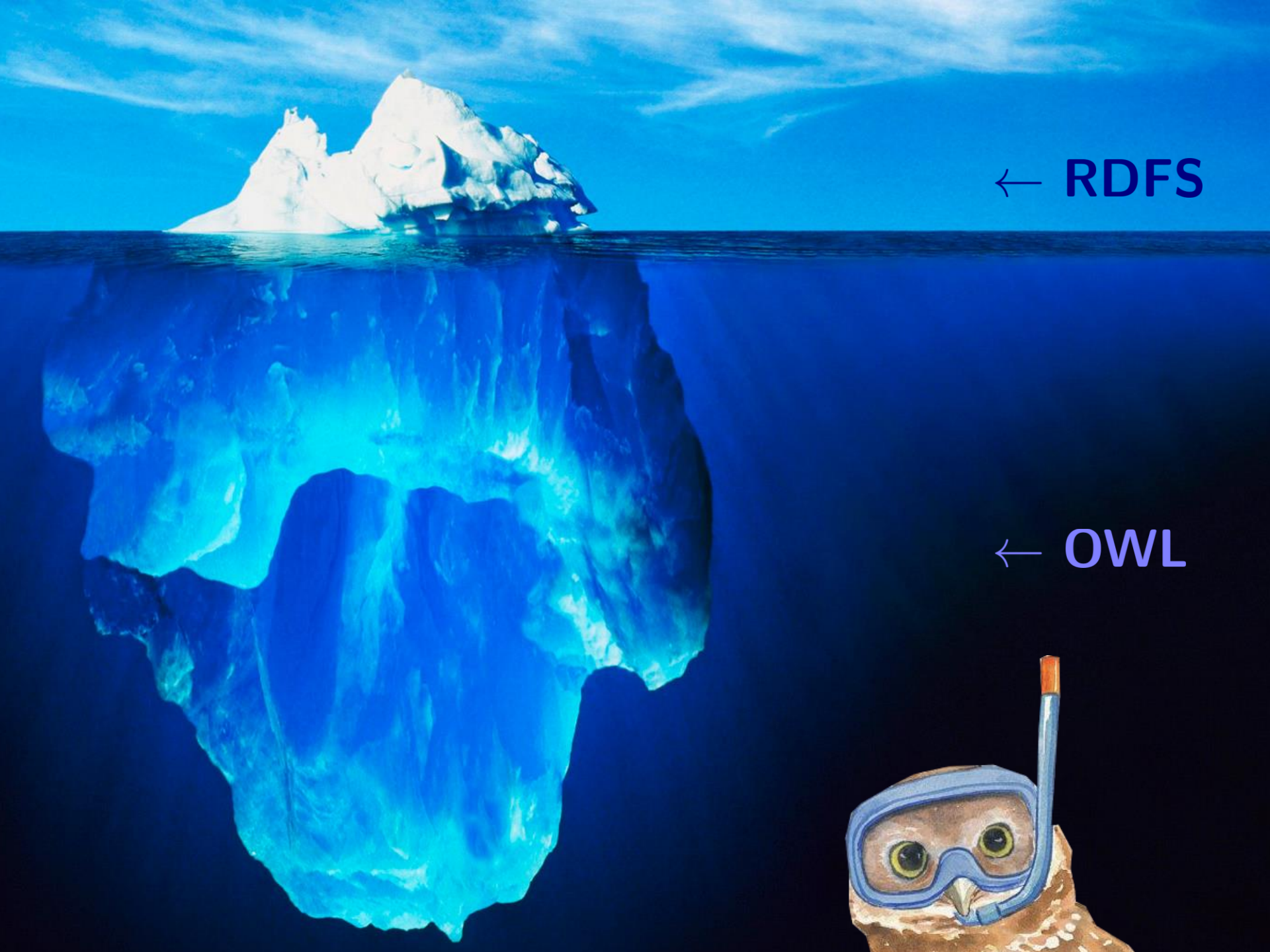
Data:

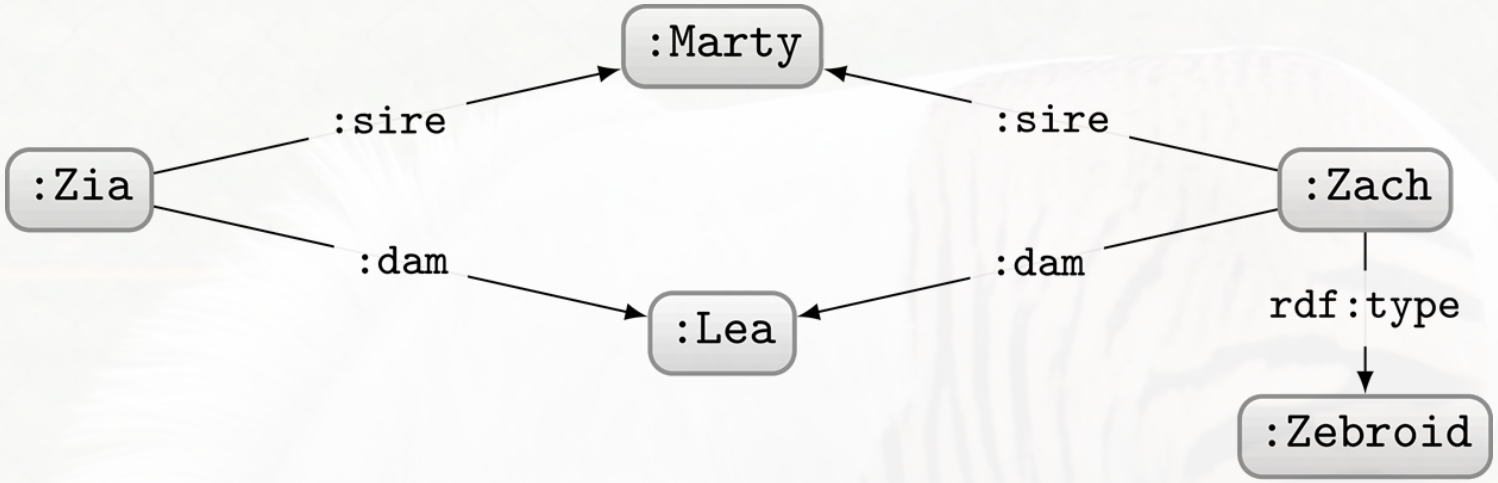
Logic: “$(b$,capital,$a) \rightarrow (a$,partOf,$b)$”

“$(a$,partOf,$b)$, $(b$,partOf,$c) \rightarrow (a$,partOf,$c)$”

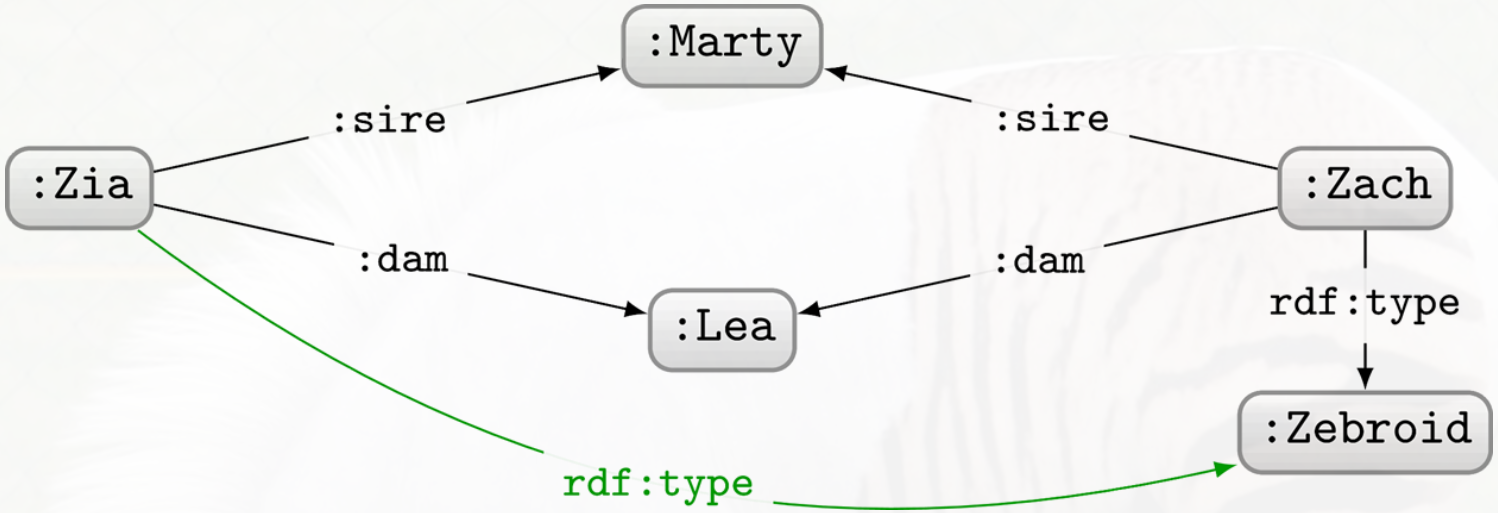
Query: “$(x$,partOf,$y)$?”

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


← RDFS
← OWL



What can we intuitively conclude about Zia?

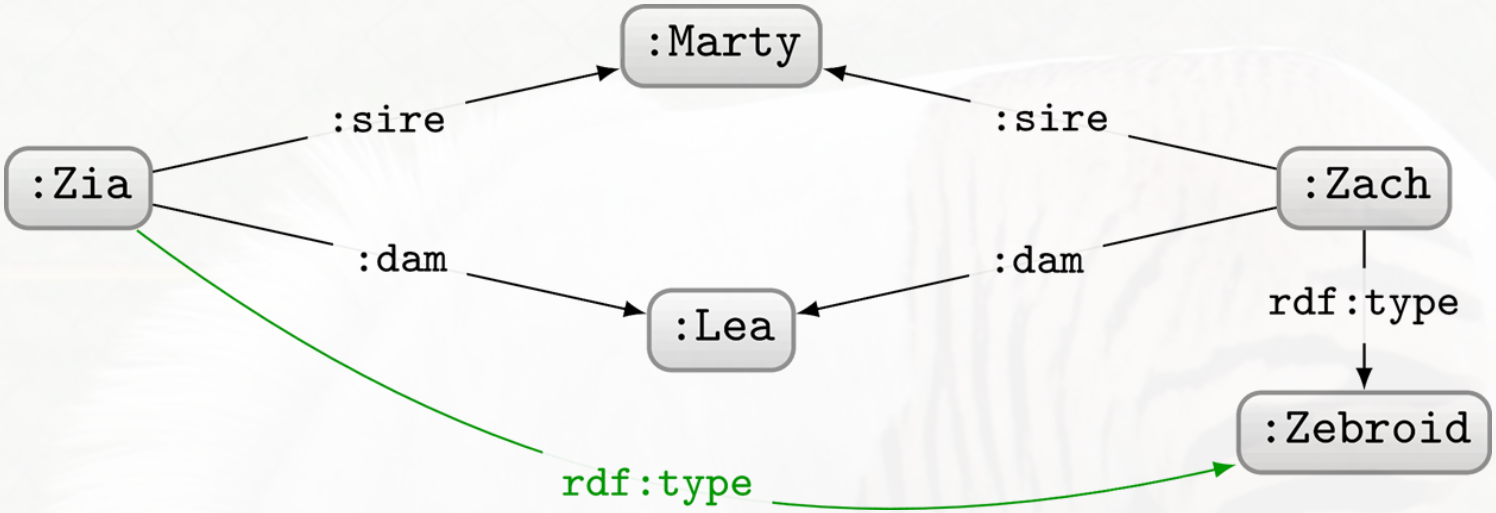


What can we intuitively conclude about `Zia`?

`Zia` is also a `Zebroid`!

What kind of reasoning are we using here?

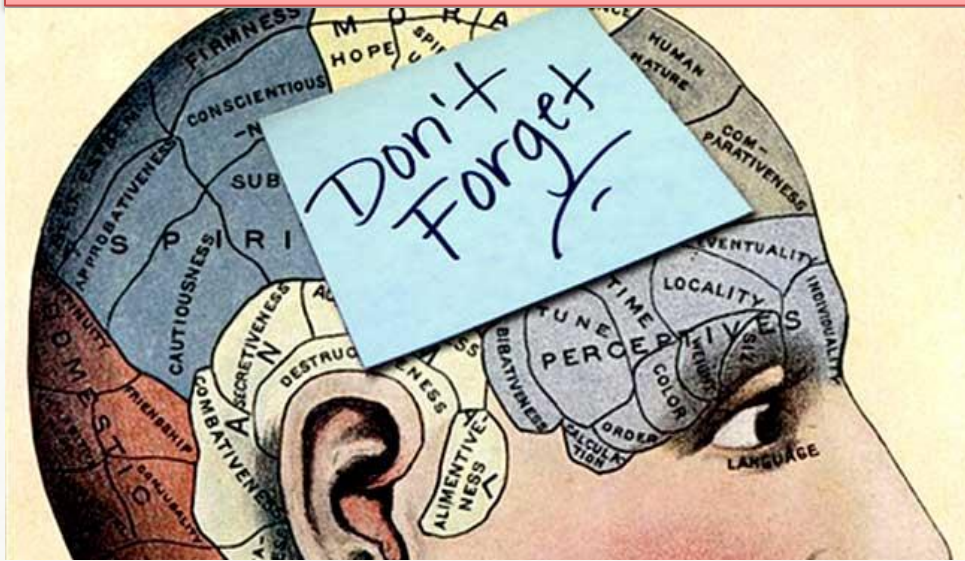
Deductive (mostly)



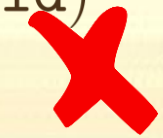
What assumptions do we make to conclude that `Zia` is a `Zebroid`?

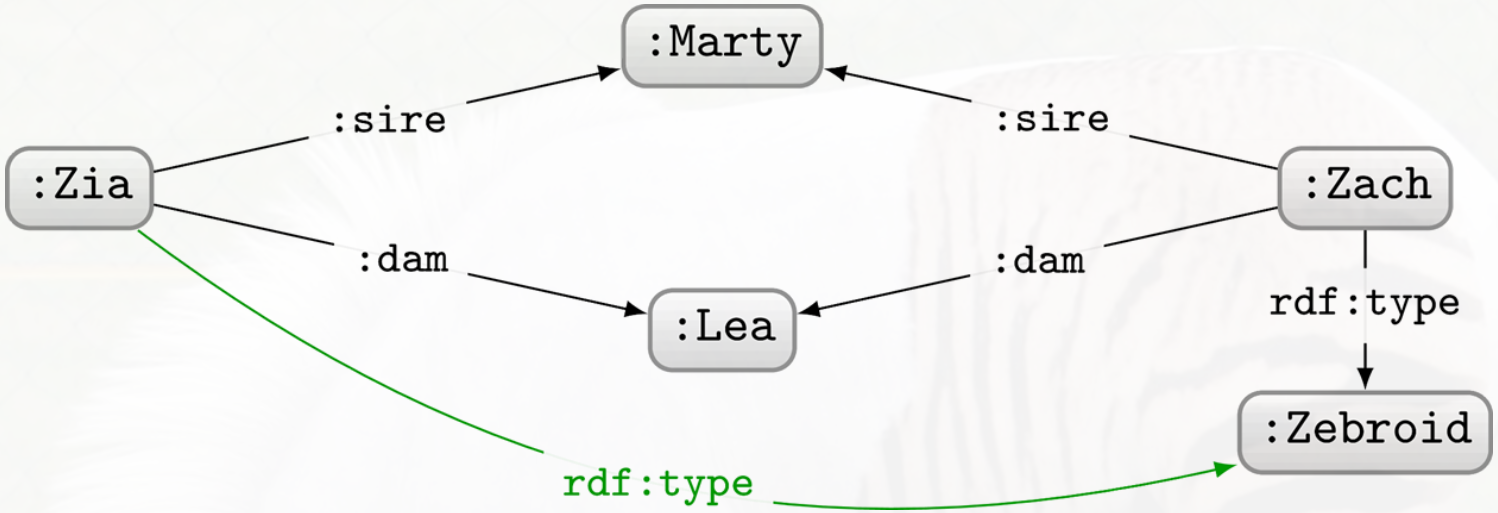
If x has same `sire` and `dam` as y and y is a `Zebroid` then x is a `Zebroid`!

Very specific to this example



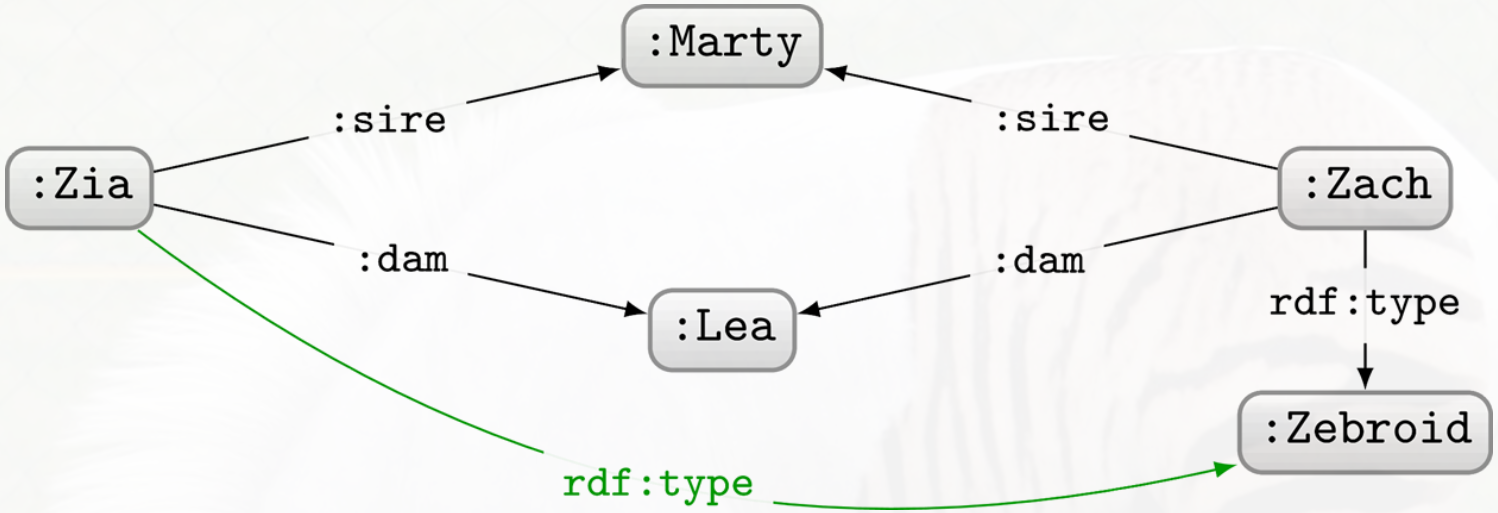
$$(x, \texttt{:dam}, z_1),\ (x, \texttt{:sire}, z_2),$$
$$(y, \texttt{:dam}, z_1),\ (y, \texttt{:sire}, z_2),$$
$$(y, \texttt{rdf:type}, \texttt{:Zebroid})$$
$$\rightarrow (x, \texttt{rdf:type}, \texttt{:Zebroid})$$





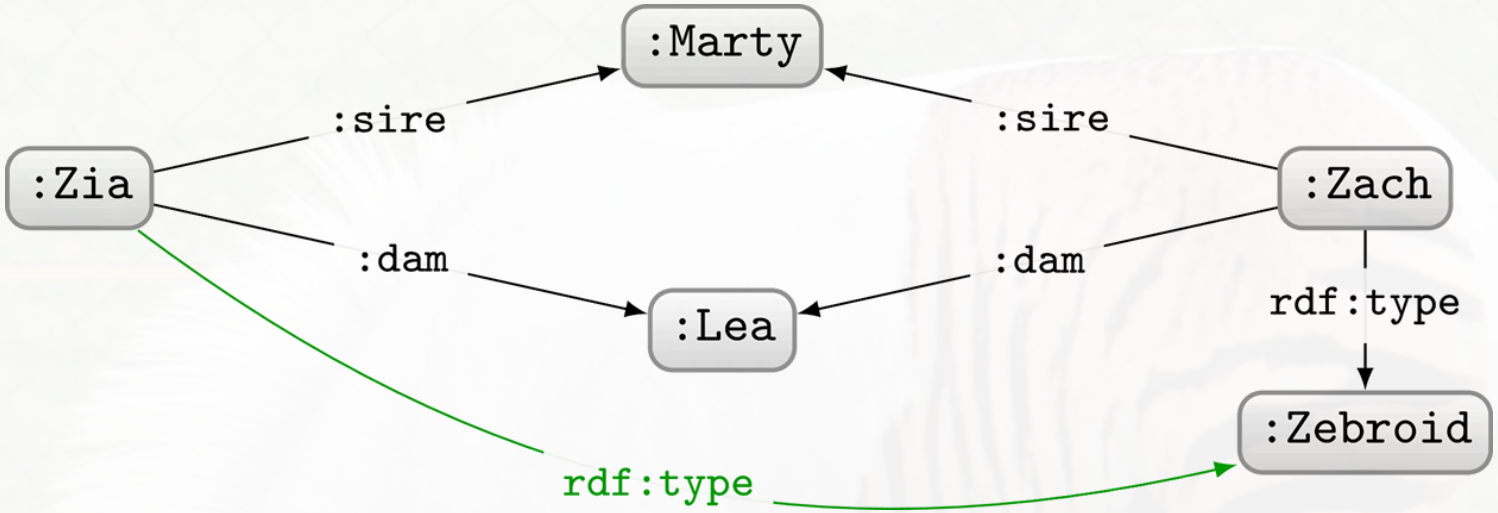
What assumptions do we make to conclude that `Zia` is a `Zebroid`?

- `sire` is a sub-property of `parent`
- `dam` is a sub-property of `parent`



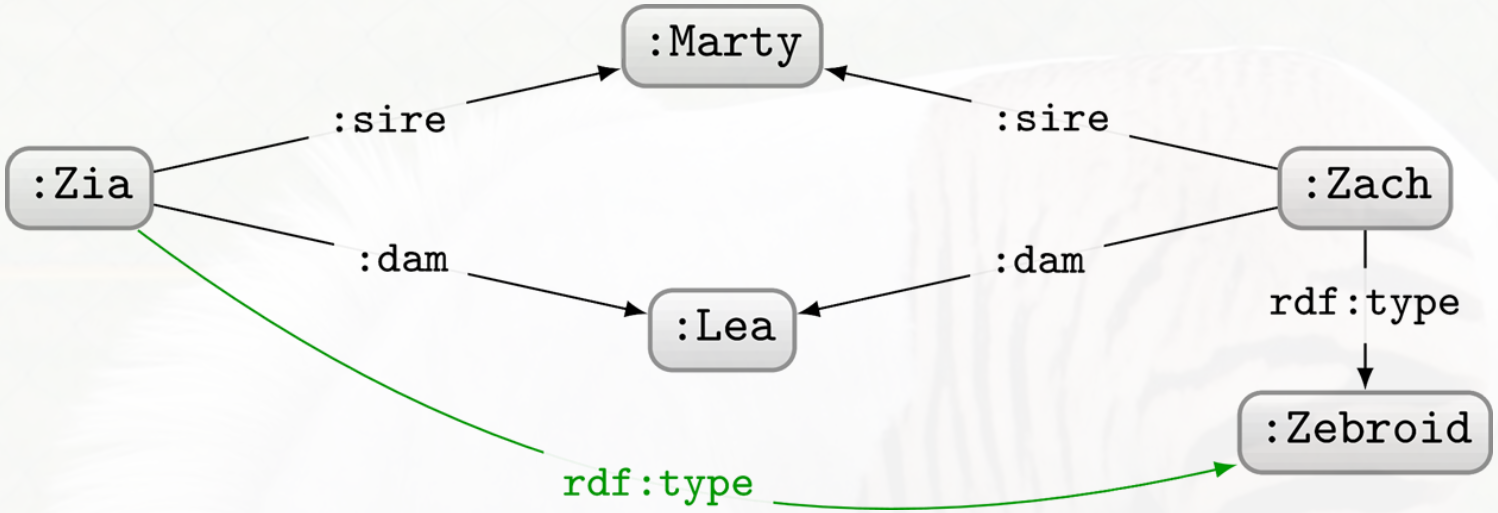
What assumptions do we make to conclude that `Zia` is a `Zebroid`?

- `sire` is a sub-property of `parent`
- `dam` is a sub-property of `parent`
- A `Zebroid` has exactly one `parent` a `Zebra`
- A `Zebroid` has exactly one `parent` a (¬`Zebra` and a `Equine`)



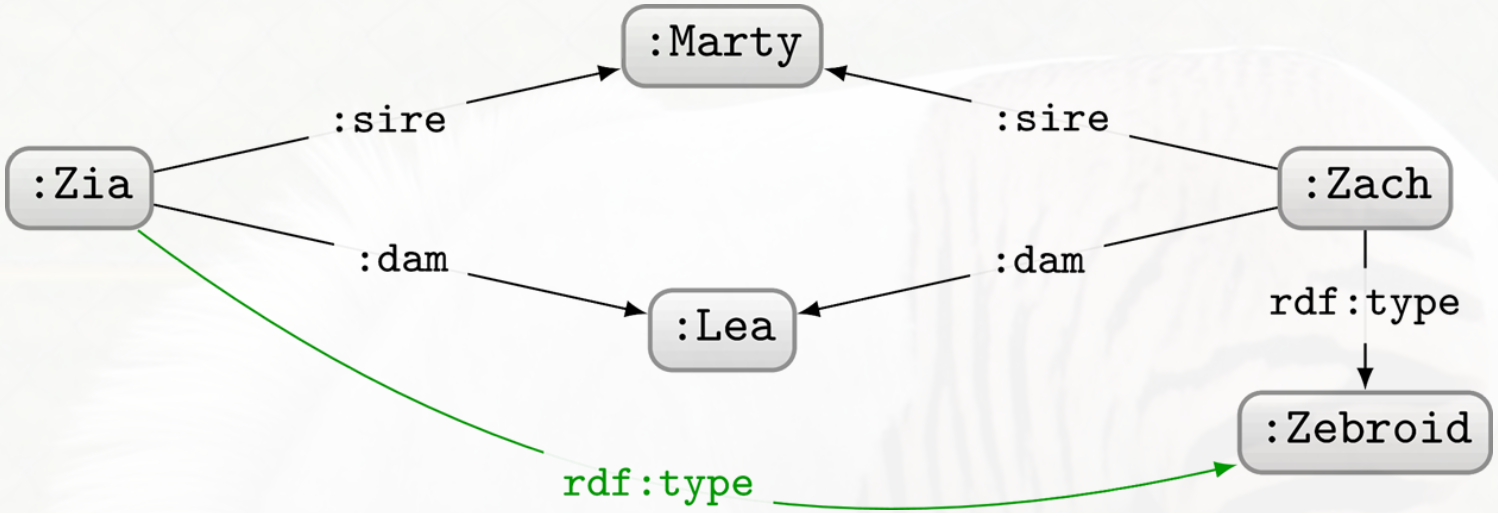
What assumptions do we make to conclude that Zia is a Zebroid?

- sire is a sub-property of parent
- dam is a sub-property of parent
- A Zebroid has exactly one parent a Zebra
- A Zebroid has exactly one parent a (¬Zebra and a Equine)
- A Zebroid is a sub-class of Equine
- An Equine has exactly two parents



What assumptions do we make to conclude that `Zia` is a `Zebroid`?

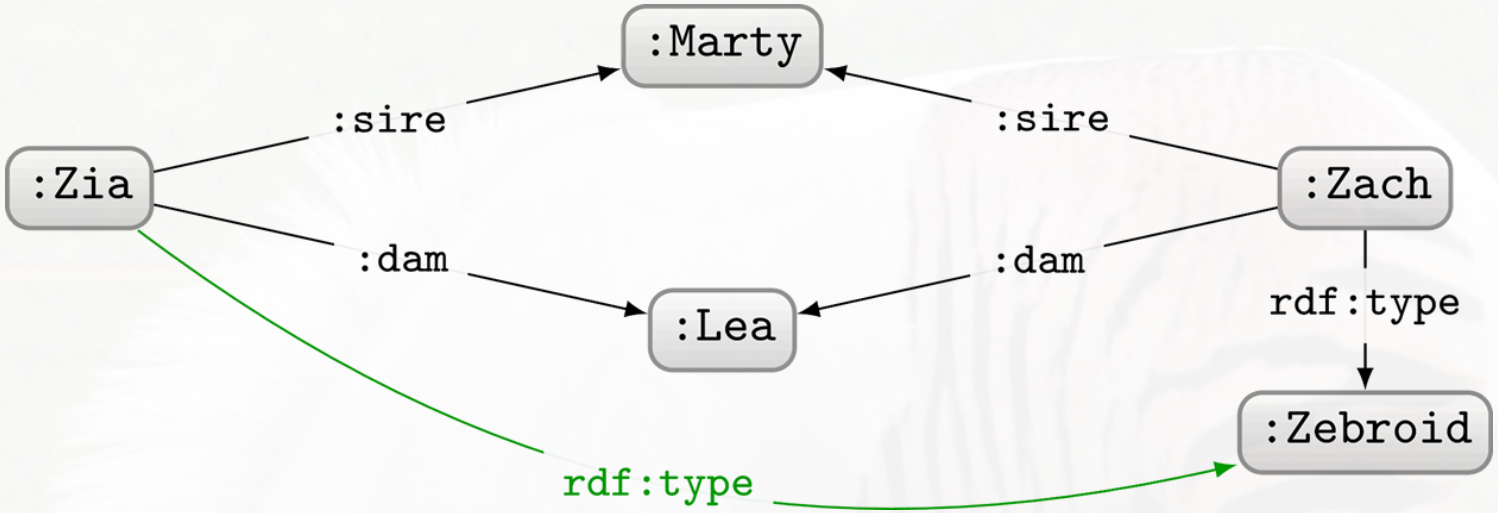
- `sire` is a sub-property of `parent`
- `dam` is a sub-property of `parent`
- A `Zebroid` has exactly one `parent` a `Zebra`
- A `Zebroid` has exactly one `parent` a (¬`Zebra` and a `Equine`)
- A `Zebroid` is a sub-class of `Equine`
- An `Equine` has exactly two `parents`
- Two things cannot be related by `sire` and `dam` at the same time



What assumptions do we make to conclude that Zia is a Zebroid?

Which are expressible in RDFS?

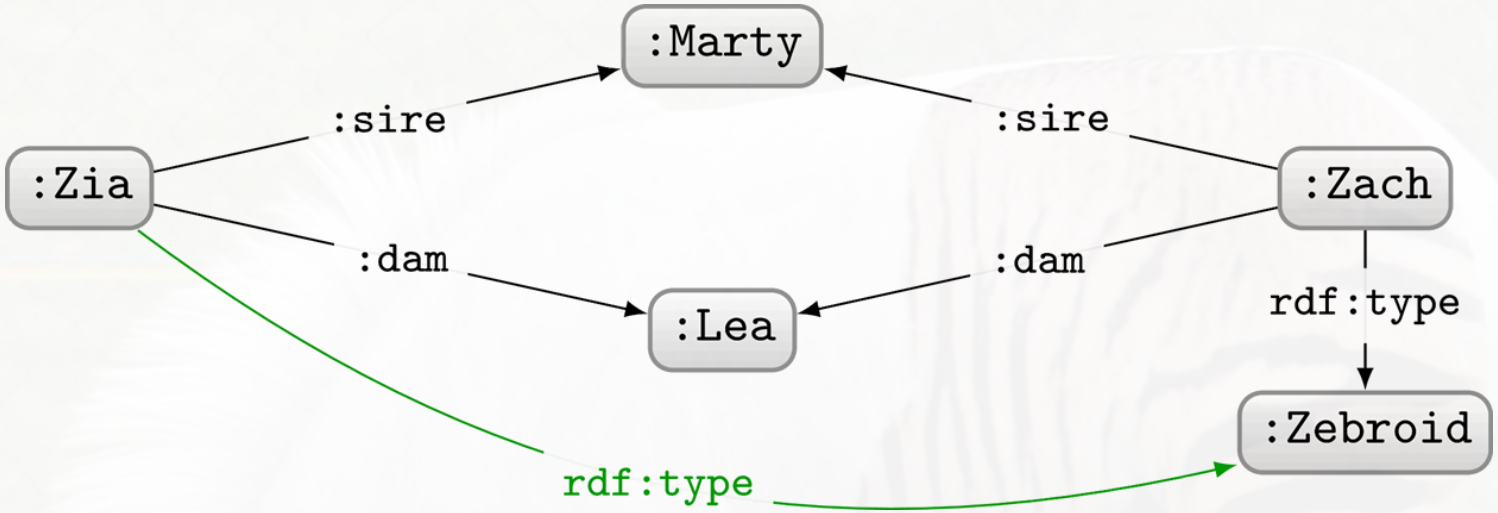
- sire is a sub-property of parent
- dam is a sub-property of parent
- A Zebroid has exactly one parent a Zebra
- A Zebroid has exactly one parent a (¬Zebra and a Equine)
- A Zebroid is a sub-class of Equine
- An Equine has exactly two parents
- Two things cannot be related by sire and dam at the same time



What assumptions do we make to conclude that Zia is a Zebroid?

Which are expressible in RDFS?

- sire is a sub-property of parent
- dam is a sub-property of parent
- A Zebroid has exactly one parent a Zebra
- A Zebroid has exactly one parent a (¬Zebra and a Equine)
- A Zebroid is a sub-class of Equine
- An Equine has exactly two parents
- Two things cannot be related by sire and dam at the same time



What assumptions do we make to conclude that Zia is a Zebroid?

- sire is a sub-property of parent
- dam is a sub-property of parent
- A Zebroid has exactly one parent a Zebra
- A Zebroid has exactly one parent a (¬Zebra and a Equine)
- A Zebroid is a sub-class of Equine
- An Equine has exactly two parents
- Two things cannot be related by sire and dam at the same time

Which are expressible in RDFS?

The rest we can express in OWL

Web Ontology Language: OWL

OWL (2): A Web Standard

W3C Recommendation

https://www.w3.org/TR/owl2-overview/

OWL 2 Web Ontology Language Document Overview (Second Edition)

W3C Recommendation 11 December 2012

This version:
http://www.w3.org/TR/2012/REC-owl2-overview-20121211/
Latest version (series 2):
http://www.w3.org/TR/owl2-overview/
Latest Recommendation:
http://www.w3.org/TR/owl-overview
Previous version:
http://www.w3.org/TR/2012/PER-owl2-overview-20121018/
Editors:
W3C OWL Working Group (see Acknowledgements)

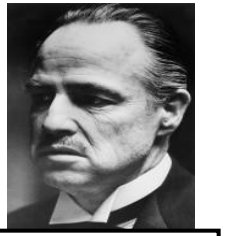
Please refer to the **errata** for this document, which may include some normative corrections.

A color-coded version of this document showing changes made since the previous version is also available.

Formal Underpinnings: Description Logics

Name	Syntax	OWL key-term	DL
		Concept Definitions	
Atomic Concept	A	`owl:Class`	$\mathcal{ALC}$
Top Concept	$\top$	`owl:Thing`	$\mathcal{ALC}$
Bottom Concept	$\bot$	`owl:Nothing`	$\mathcal{ALC}$
Concept Negation	$\neg C$	`owl:complementOf`	$\mathcal{ALC}$
Concept Intersection	$C \sqcap D$	`owl:intersectionOf`	$\mathcal{ALC}$
Concept Union	$C \sqcup D$	`owl:unionOf`	$\mathcal{ALC}$
Nominal	$\{a_1, ..., a_n\}$	`owl:oneOf`	$\mathcal{O}$
Existential Restriction	$\exists R.C$	`owl:someValuesFrom`	$\mathcal{ALC}$
Universal Restriction	$\forall R.C$	`owl:allValuesFrom`	$\mathcal{ALC}$
Self Restriction	$\exists R.\mathsf{Self}$	`owl:hasSelf`	$\mathcal{R}$
Number Restriction	$\leq n\, R, \geq n\, R, = n\, R$	`owl:*cardinality`	$\mathcal{N}$
Qualified Number Restriction	$\leq n\, R.C, \geq n\, R.C, = n\, R.C$	`owl:*qualifiedCardinality`	$\mathcal{Q}$
		Concept Axioms (T-Box)	
Concept Inclusion	$C \sqsubseteq D$	`rdfs:subClassOf`	$\mathcal{ALC}$
		Role Definitions	
Role	R	`owl:*Property`	$\mathcal{ALC}$
Inverse Role	R^-	`owl:inverseOf`	$\mathcal{I}$
Universal Role	U	`owl:top*Property`	$\mathcal{R}$
		Role Axioms (R-Box)	
Role Inclusion	$R \sqsubseteq S$	`rdfs:subPropertyOf`	$\mathcal{H}$
Complex Role Inclusion	$R_1 \circ ... \circ R_n \sqsubseteq S$	`owl:propertyChainAxiom`	$\mathcal{R}$
Transitive Roles	$\mathsf{Trans}(R)$	`owl:TransitiveProperty`	$\mathcal{S}$
Functional Roles	$\mathsf{Func}(R)$	`owl:FunctionalProperty`	$\mathcal{F}$
Reflexive Roles	$\mathsf{Ref}(R)$	`owl:ReflexiveProperty`	$\mathcal{R}$
Irreflexive Roles	$\mathsf{Irref}(R)$	`owl:IrreflexiveProperty`	$\mathcal{R}$
Symmetric Roles	$\mathsf{Sym}(R)$	`owl:SymmetricProperty`	$\mathcal{I}$
Asymmetric Roles	$\mathsf{Asym}(R)$	`owl:AsymmetricProperty`	$\mathcal{R}$
Disjoint Roles	$\mathsf{Disj}(R, S)$	`owl:disjointPropertyWith`	$\mathcal{R}$
		Assertional Definitions	
(Named) Individual	a	(*RDF IRI or Literal*)	$\mathcal{ALC}$
		Assertional Axioms (A-Box)	
Role Assertion	$R(a, b)$	(*RDF triple*)	$\mathcal{ALC}$
Negative Role Assertion	$\neg R(a, b)$	`owl:NegativePropertyAssertion`	$\mathcal{ALC}$
Concept Assertion	$C(a)$	`rdf:type`	$\mathcal{ALC}$
Equality	$a = b$	`owl:sameAs`	$\mathcal{ALC}$
Inequality	$a \neq b$	`owl:differentFrom`	$\mathcal{ALC}$

For today: A running example



:Vito

:Carmela

:Sonny

:Connie

:Fredo

:Michael

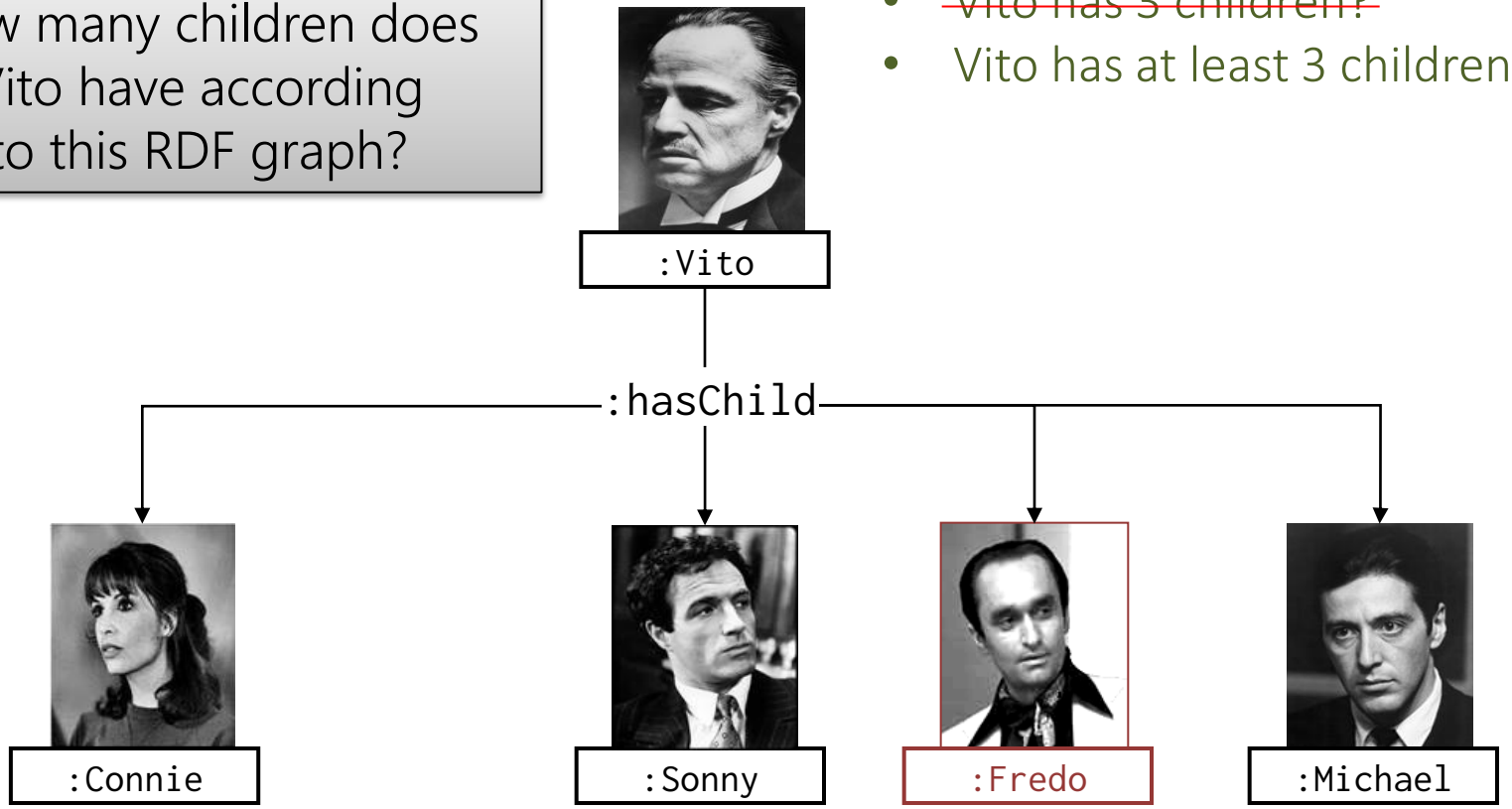
:Vincent

:Mary

Logical Assumptions

Open World Assumption (OWA)

How many children does Vito have according to this RDF graph?



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

```
:Vito :hasChild :Connie , :Sonny , :Michael .
:Vito :hasChild :Fredo .
... ?
```

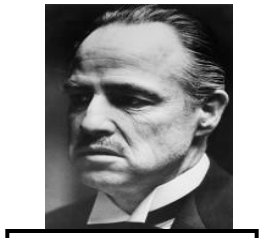
Open World Assumption

- RDF(S) and OWL:
  - Take an Open World Assumption (OWA):
    - Anything not known is not assumed to be false, simply unknown
    - Without further information, Vito may have children that we don't know about!

Why might this assumption be important for the Web?
OWA: Assuming Web data to be complete a Bad Idea®.

No Unique Name Assumption (No UNA)

How many children does Vito have according to this RDF graph?



:Vito

:hasChild

:Connie

:Sonny

:Fredo

:Michael

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

```
:Vito :hasChild :Connie , :Sonny , :Michael .
:Vito :hasChild :Fredo .
... ?
```

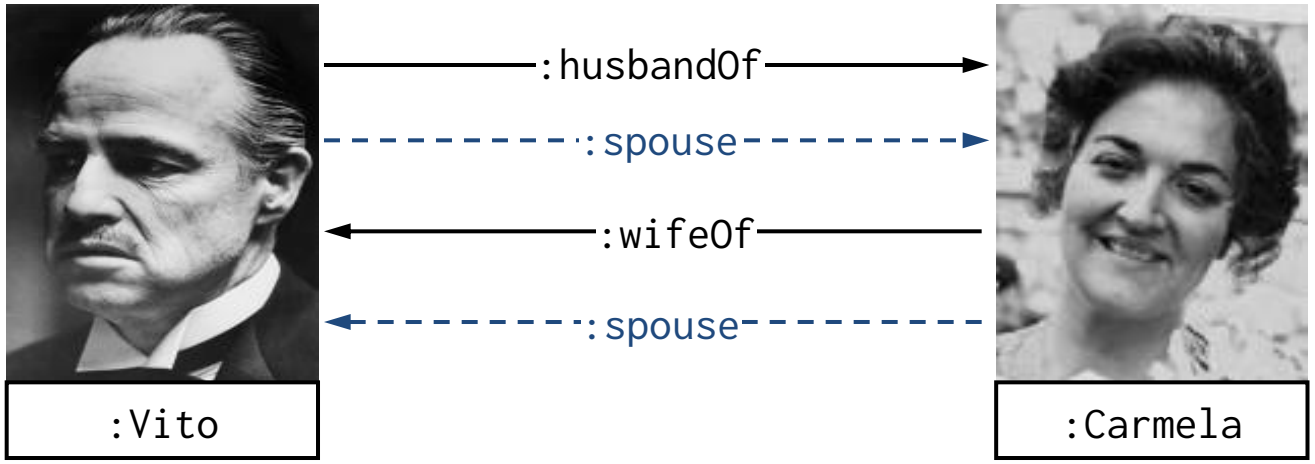
No Unique Name Assumption (No UNA)

- RDF(S) and OWL:
  - 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 real-world thing!

Why might this assumption be important for the Web?
No UNA: Assuming strict naming agreement on the Web a Bad Idea®.

Let's start with some RDFS ...

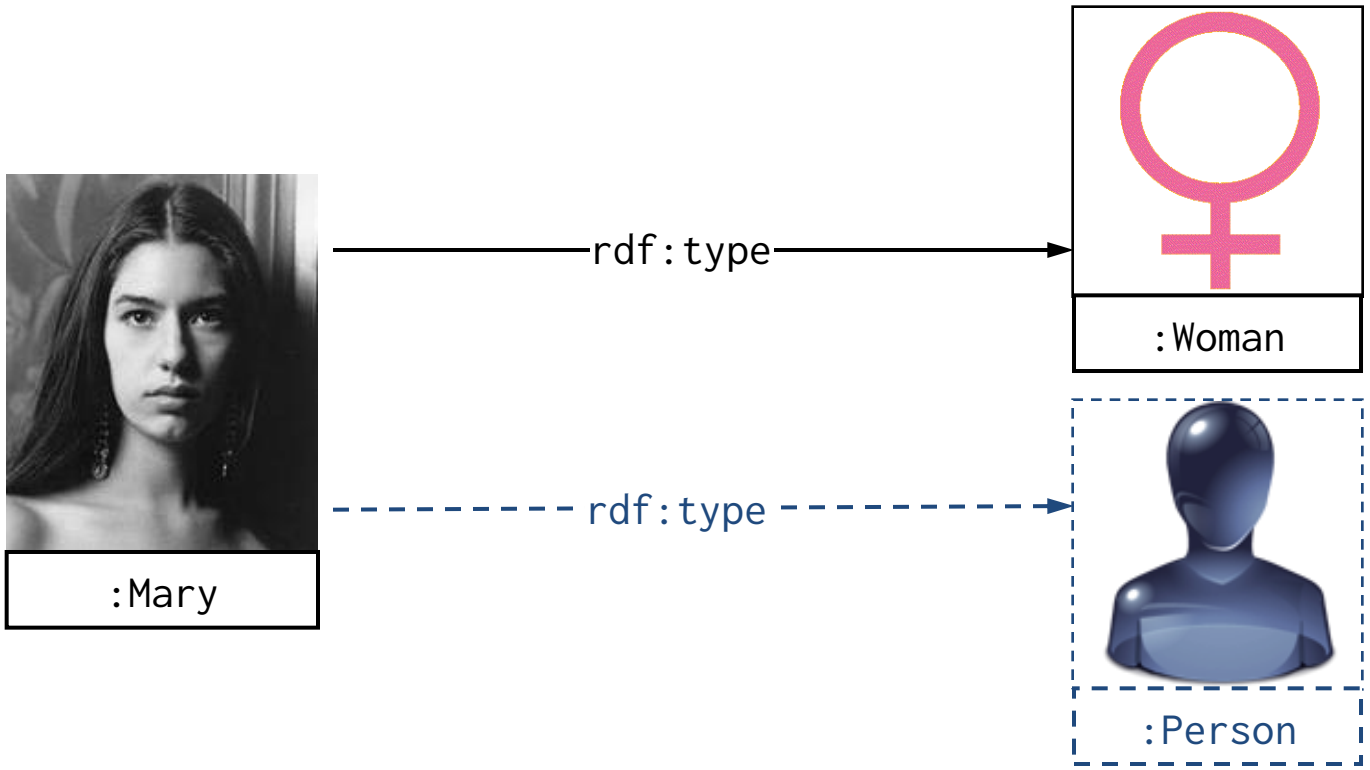
rdfs:subPropertyOf




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

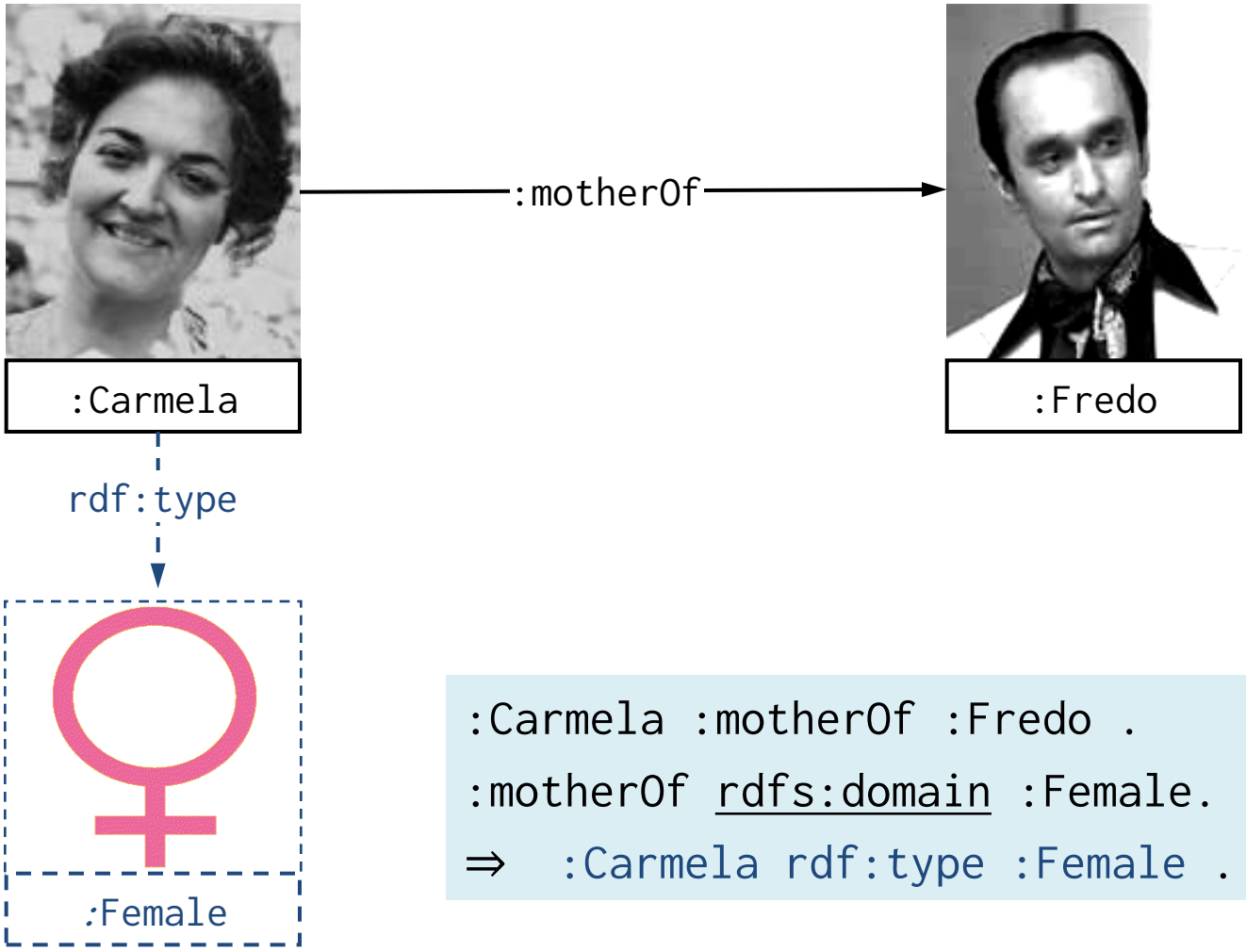
```
:Carmela :wifeOf :Vito .
:wifeOf rdfs:subPropertyOf :spouse .
⇒  :Carmela :spouse :Vito .
```

rdfs:subClassOf



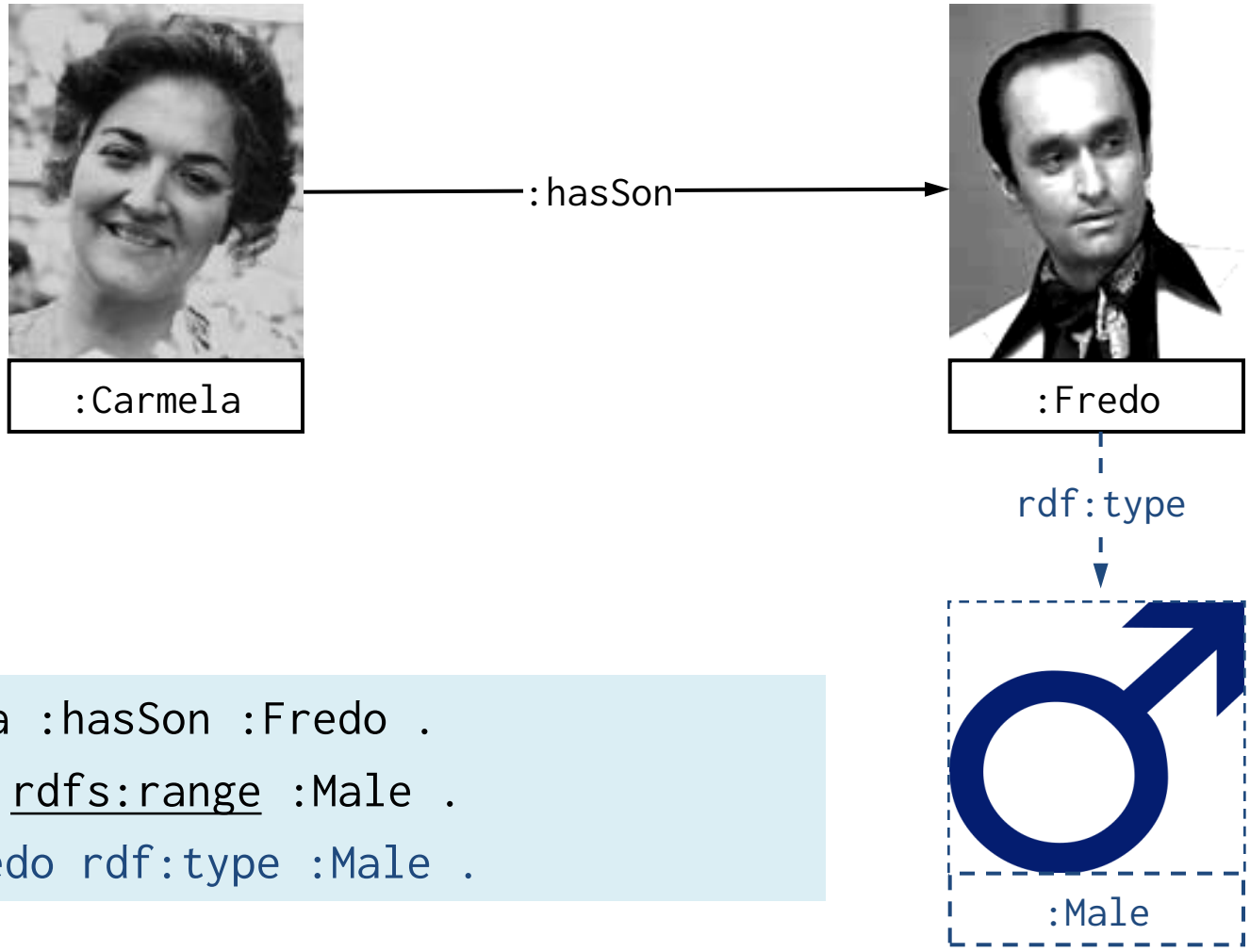

```
:Mary rdf:type :Woman .
:Woman rdfs:subClassOf :Person .
⇒   :Mary rdf:type :Person .
```

rdfs:domain




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

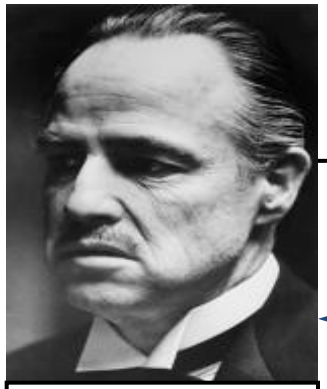
rdfs:range



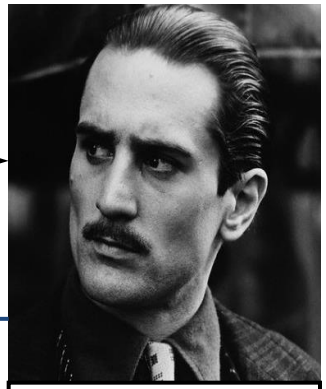

```
:Carmela :hasSon :Fredo .
:hasSon rdfs:range :Male .
⇒  :Fredo rdf:type :Male .
```

(In)Equality in OWL …

owl:sameAs



owl:sameAs



owl:sameAs

:VitoOld

:VitoOld
:VitoYoung

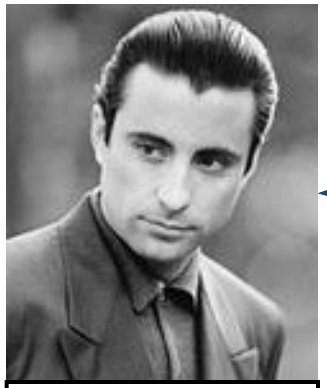
:VitoYoung

:hasGrandson

:hasGrandson

:granddaughterOf

:granddaughterOf



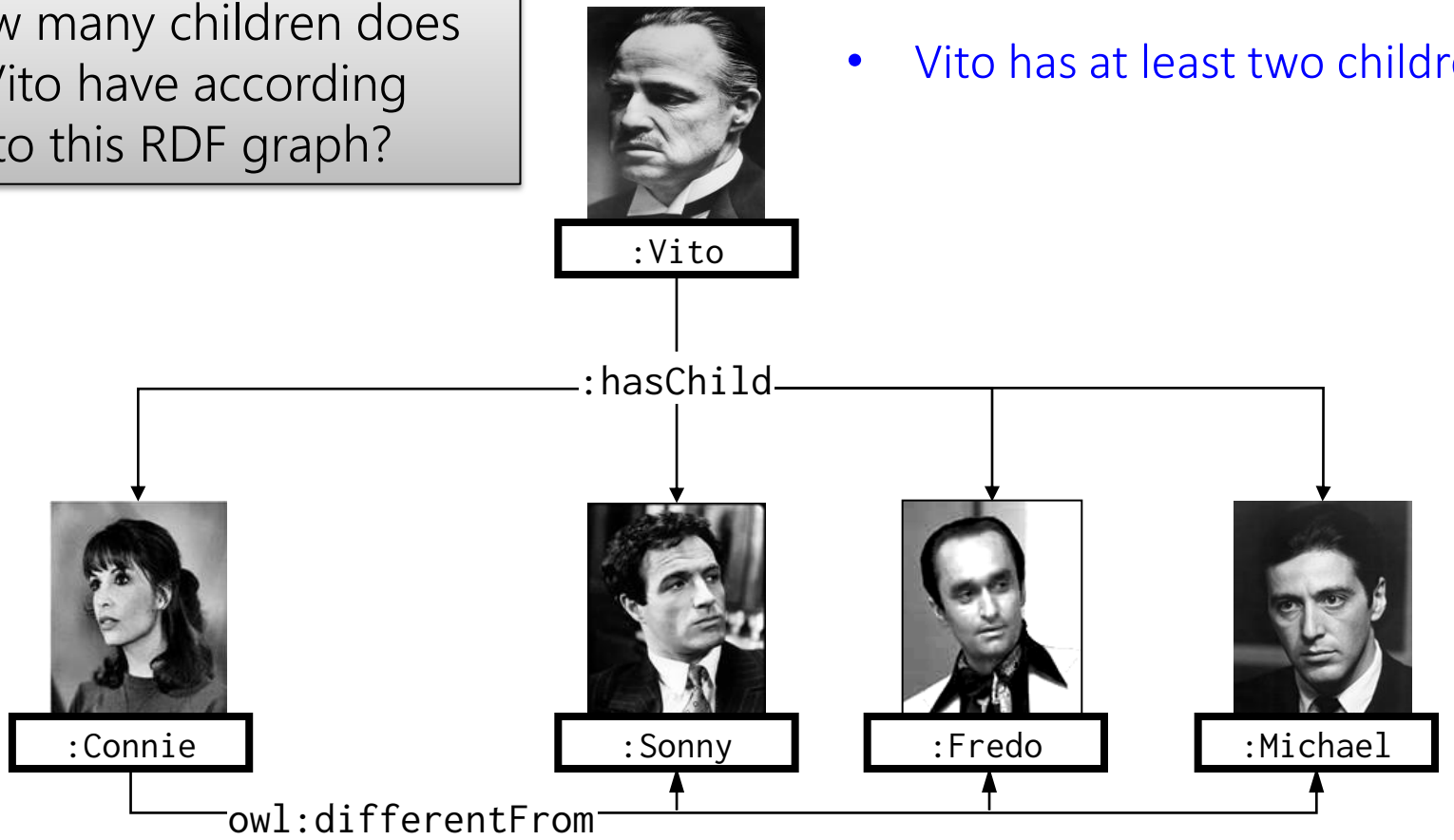
:Vincent

:Mary

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

owl:differentFrom

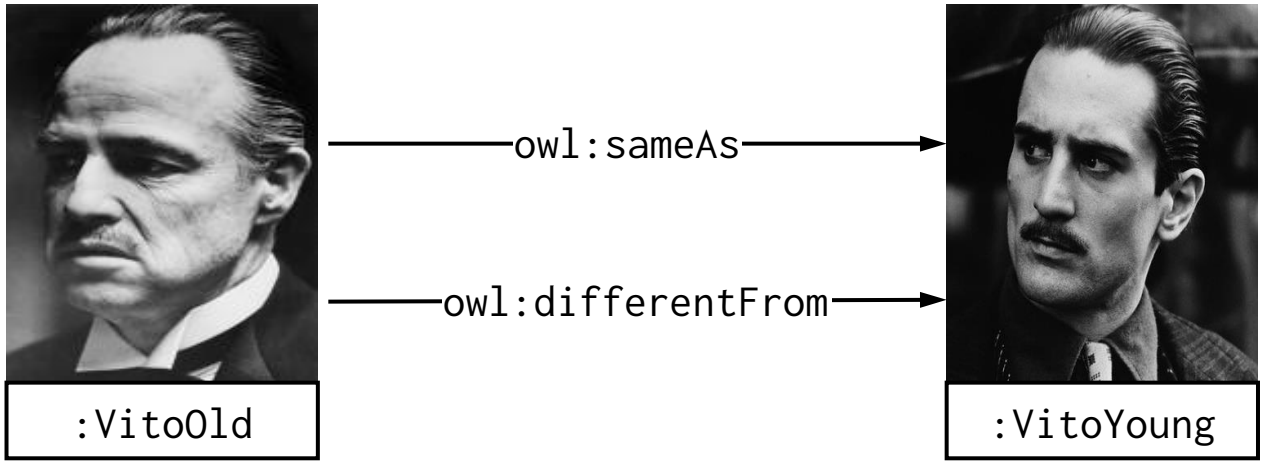
How many children does Vito have according to this RDF graph?



- Vito has at least two children!

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

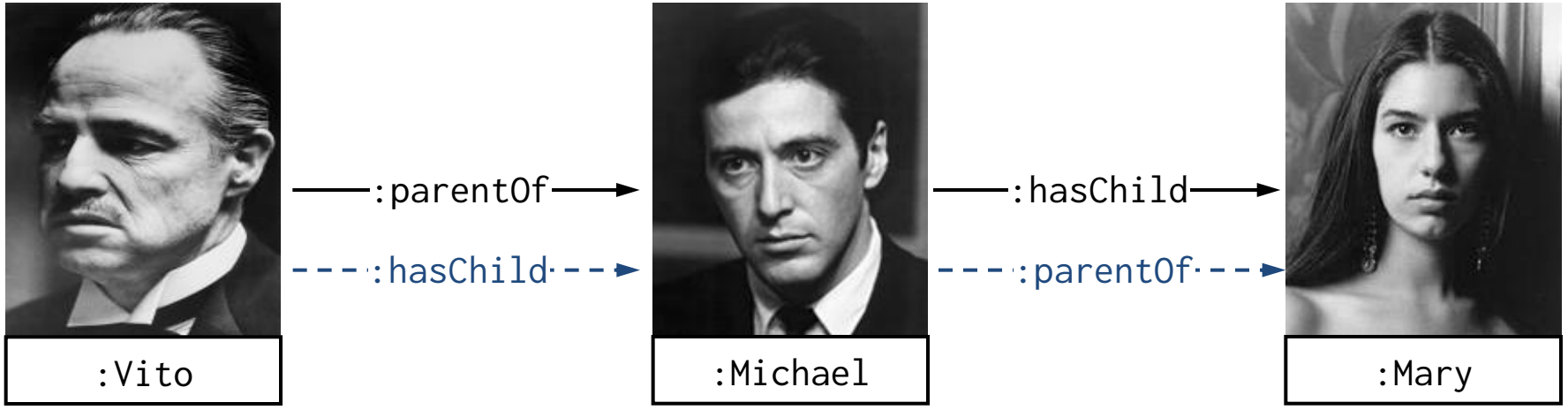
Inconsistency in OWL …




```
:VitoOld owl:sameAs :VitoYoung .
:VitoOld owl:differentFrom :VitoYoung .
⇒ FALSE
```

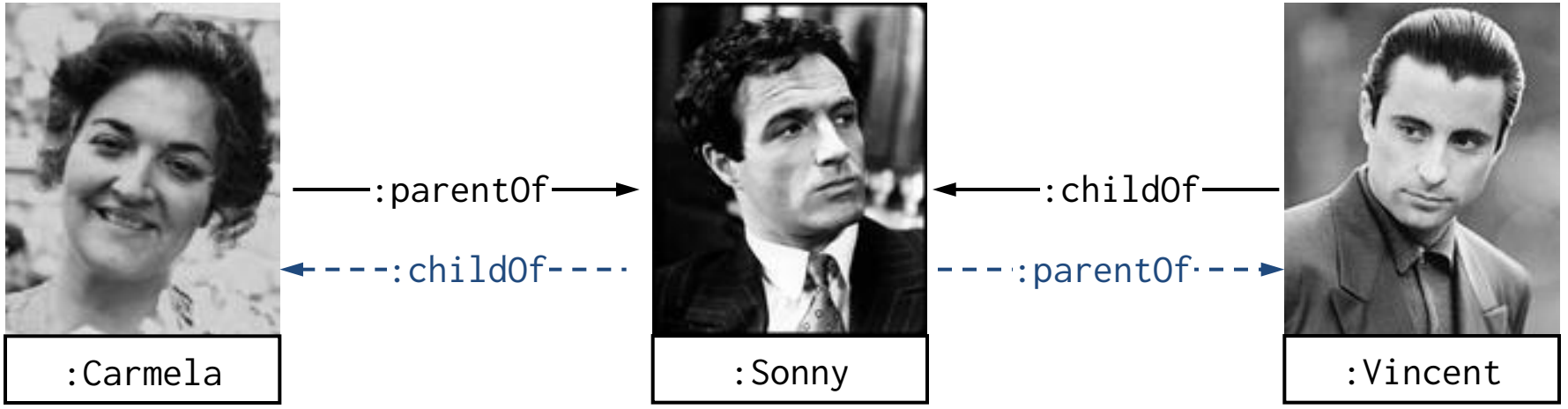
Property Axioms in OWL …

owl:equivalentProperty



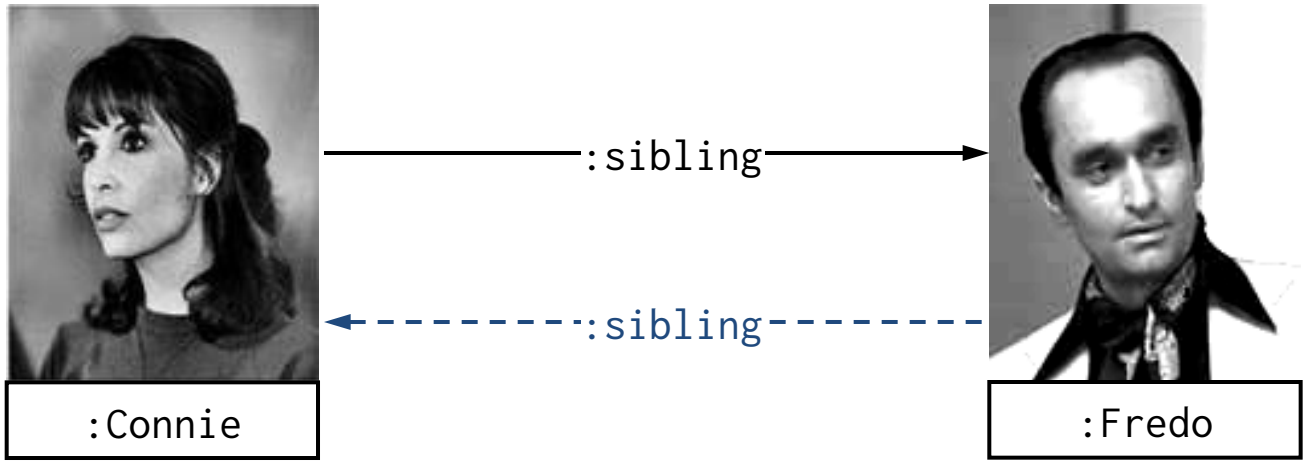

```
:Vito :parentOf :Michael .
:Michael :hasChild :Mary .
:parentOf owl:equivalentProperty :hasChild .
⇒  :Vito :hasChild :Michael .
⇒  :Michael :parentOf :Mary .
```

owl:inverseOf



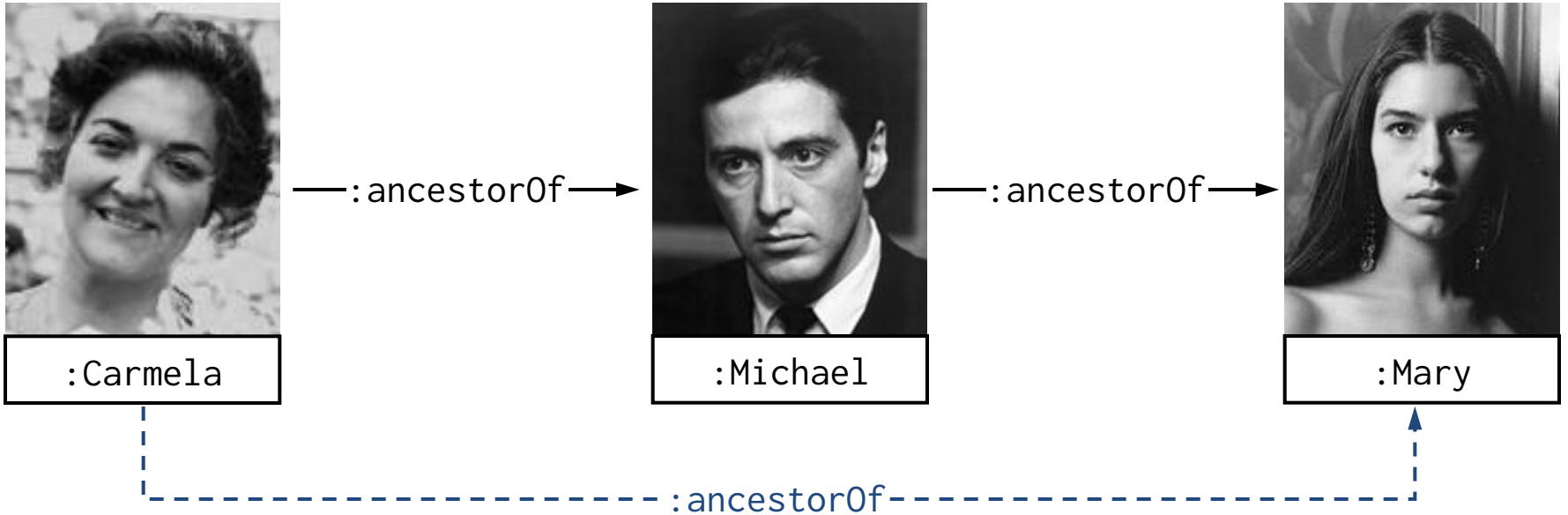

```
:Carmela :parentOf :Sonny .
:Vincent :childOf :Sonny .
:parentOf owl:inverseOf :childOf .
⇒  :Sonny :parentOf :Vincent .
⇒  :Sonny :childOf :Carmela .
```

owl:SymmetricProperty




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

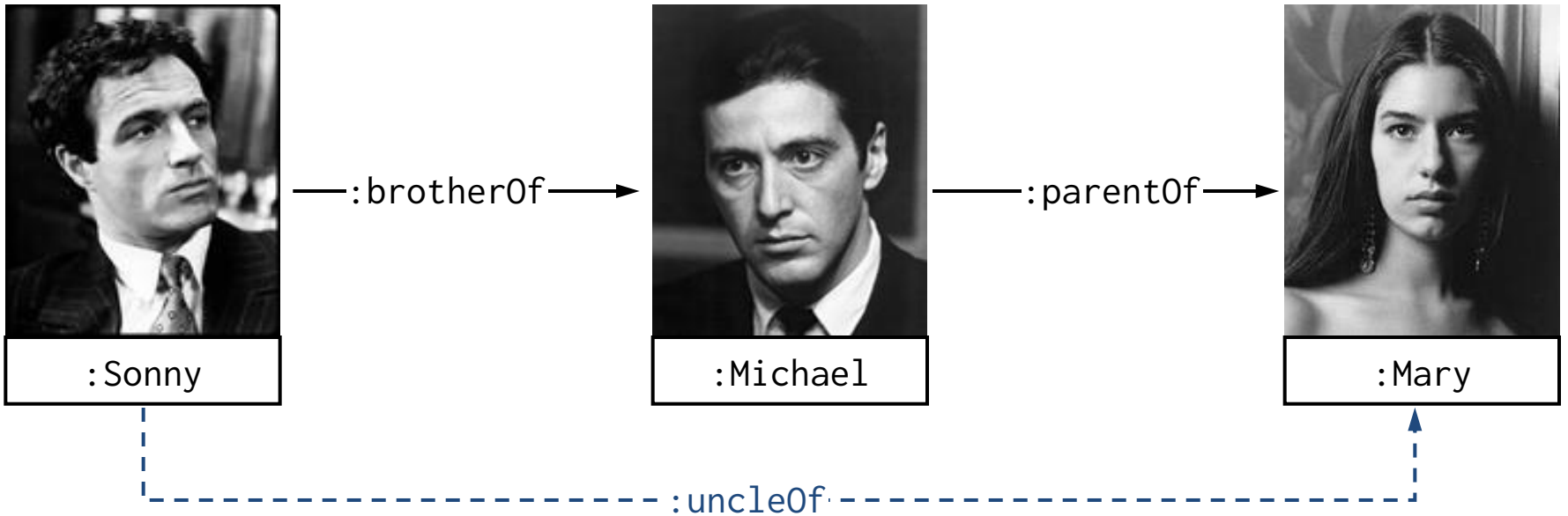
owl:TransitiveProperty




```
:Carmela :ancestorOf :Michael .
:Michael :ancestorOf :Mary .
:ancestorOf rdf:type owl:TransitiveProperty .
⇒  :Carmela :ancestorOf :Mary .
```

owl:propertyChainAxiom

Means new to OWL version 2.0!




```
:Sonny :brotherOf :Michael .
:Michael :parentOf :Mary .
:uncleOf owl:propertyChainAxiom (:brotherOf :parentOf) .
⇒  :Sonny :uncleOf :Mary .
```

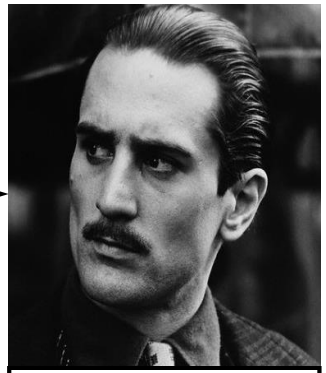
owl:ReflexiveProperty


```
:similarTo rdf:type owl:ReflexiveProperty .
⇒ :Connie :similarTo :Connie .
  :Fredo :similarTo :Fredo .
  # everything :similarTo itself
```

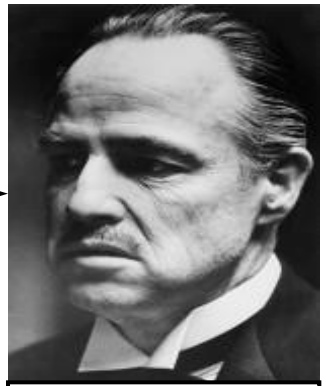
owl:FunctionalProperty

:Fredo

:hasBioFather



:VitoOld
:VitoYoung



:VitoOld

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

Aside …

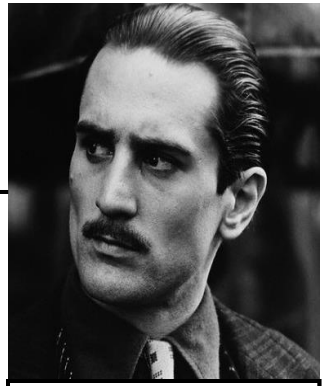
What if we said `:hasFather` was functional?

- Tom Hagen, the adopted son of Vito
  - Maybe he has two fathers?

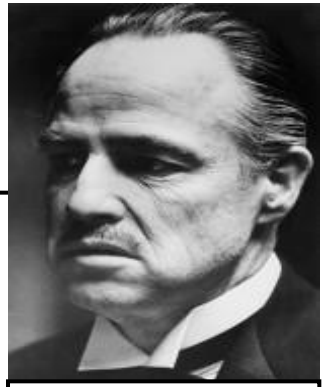
owl:InverseFunctionalProperty

:Connie

:bioFatherOf



:VitoOld
:VitoYoung



:VitoOld

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

owl:hasKey

:Connie

:hasBioMother

:Carmela

:Singleton

rdf:type

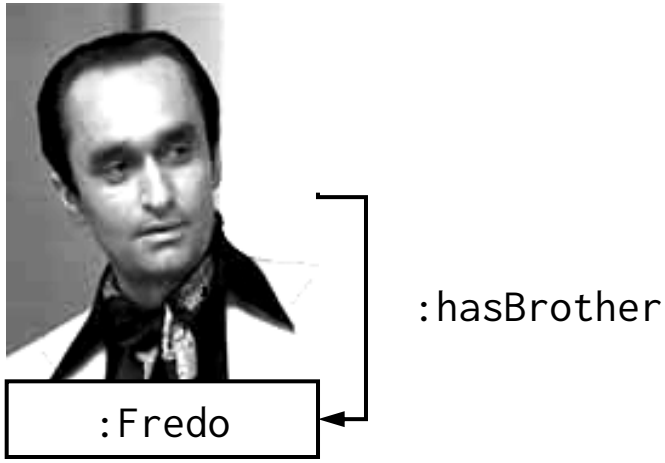
:Constanza

:born

1922-04-16

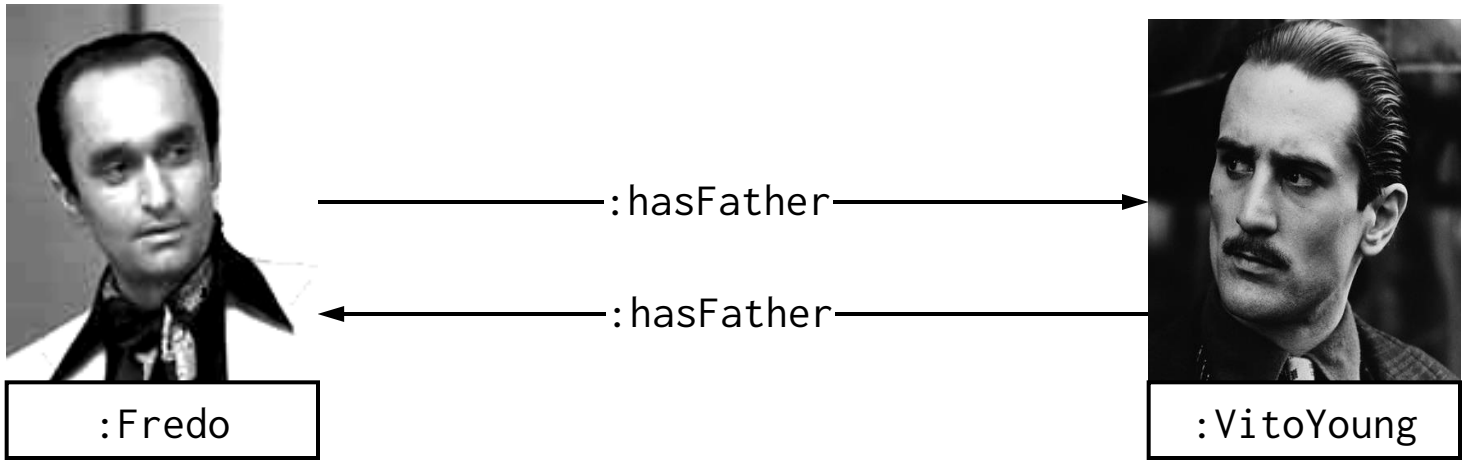
```
:Connie a :Singleton ; :hasBioMother :Carmela ; :born “1922-04-16”^^xsd:date .
:Constanza a :Singleton ; :hasBioMother :Carmela ; :born “1922-04-16”^^xsd:date .
:Singleton owl:hasKey ( :hasBioMother :born ) .
⇒ :Connie owl:sameAs :Constanza .
```

owl:IrreflexiveProperty



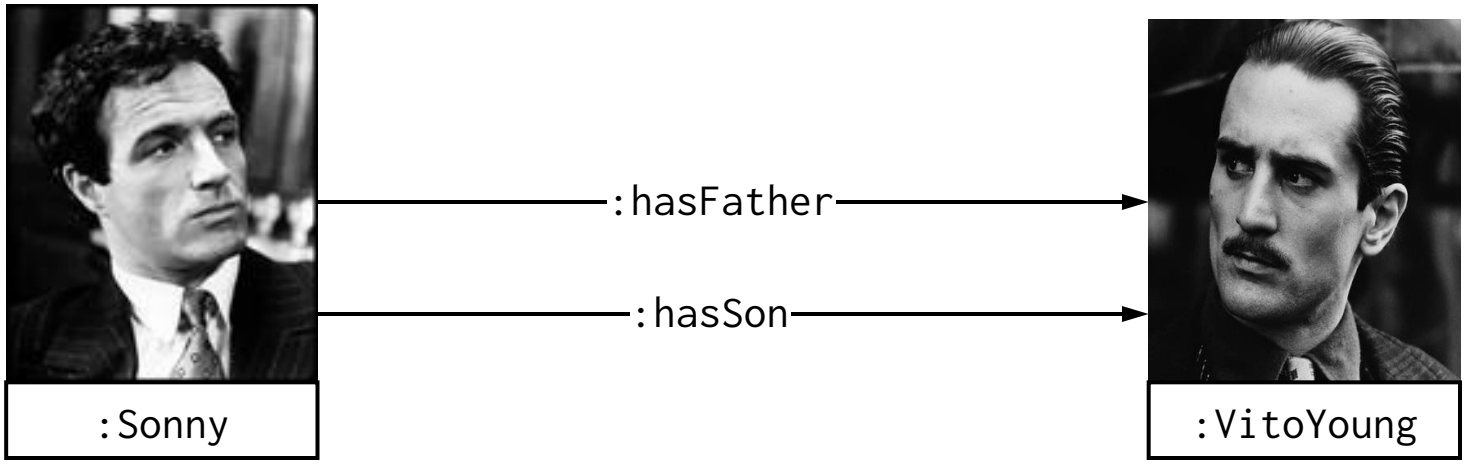

```
:Fredo :hasBrother :Fredo .
:hasBrother rdf:type owl:IrreflexiveProperty .
⇒ FALSE
```

owl:AsymmetricProperty



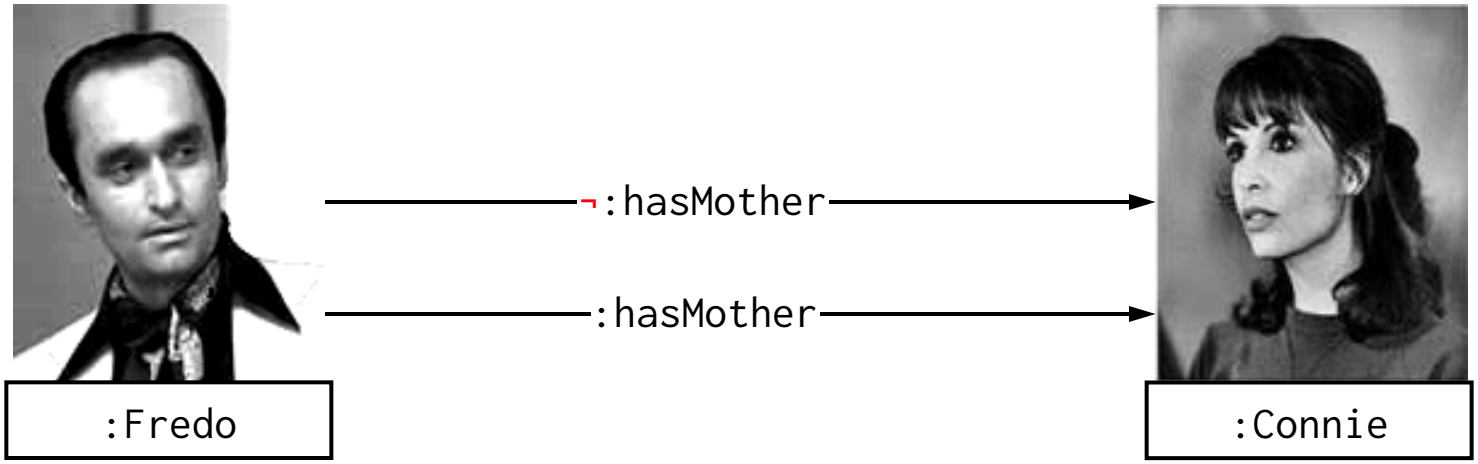

```
:Fredo :hasFather :VitoYoung .
:VitoYoung :hasFather :Fredo .
:hasFather rdf:type owl:AsymmetricProperty .
⇒ FALSE
```

owl:propertyDisjointWith




```
:Sonny :hasFather :VitoYoung .
:Sonny :hasSon :VitoYoung .
:hasSon owl:propertyDisjointWith :hasFather .
⇒ FALSE
```

Negative property assertions

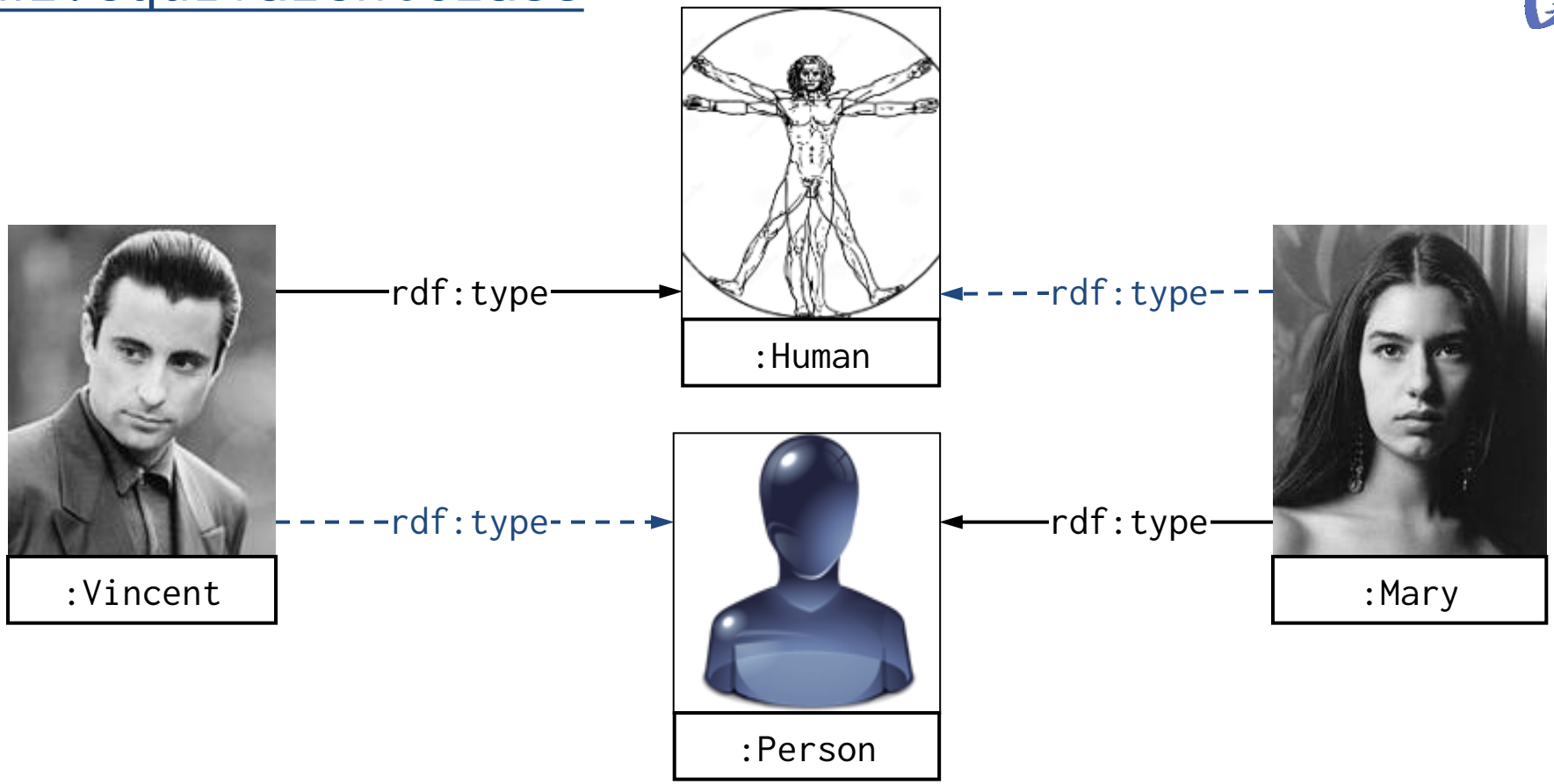



```
[] owl:sourceIndividual :Fredo ;
   owl:assertionProperty :hasMother ;
   owl:targetIndividual :Connie .
:Fredo :hasMother :Connie .
```

⇒ FALSE

Class Axioms in OWL

owl:equivalentClass




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

owl:disjointWith

:MafiaBoss

:Lawful

rdf:type

:Vincent

rdf:type

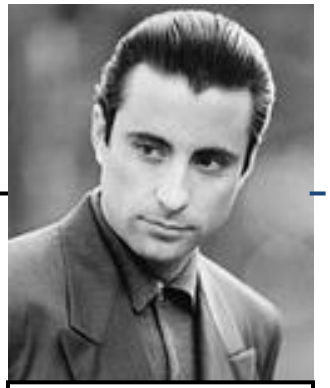
```
:Vincent rdf:type :MafiaBoss , :Lawful .
:MafiaBoss owl:disjointWith :Lawful .
⇒ FALSE
```

owl:disjointWith (ii)

:MafiaBoss

:Lawful

rdf:type



:Vincent

----owl:differentFrom ->

:Mary

rdf:type

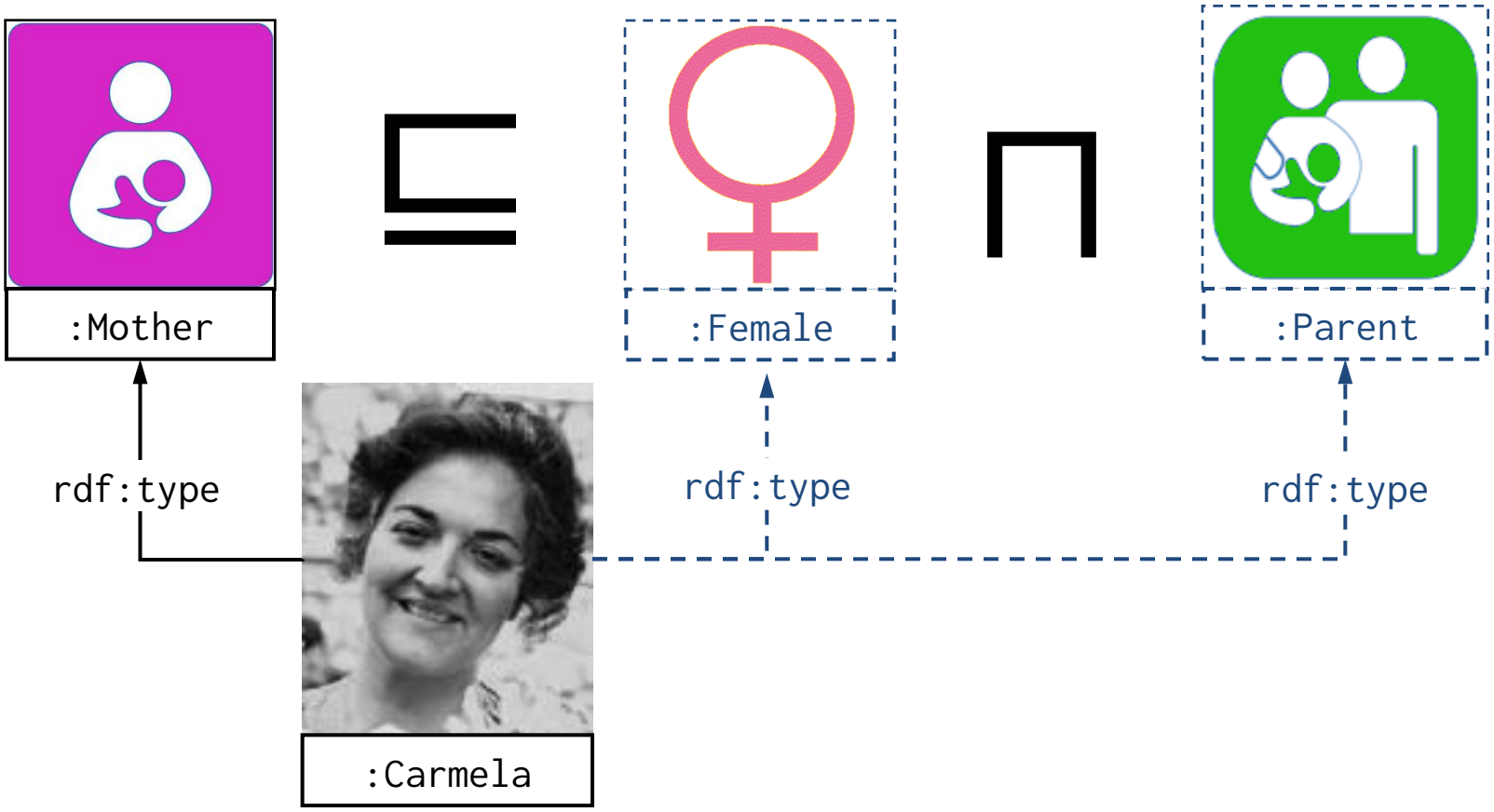
```
:Vincent rdf:type :MafiaBoss .
:Mary rdf:type :Lawful .
:MafiaBoss owl:disjointWith :Lawful .
⇒ :Vincent owl:differentFrom :Mary
```

Class Definitions in OWL

Description Logics

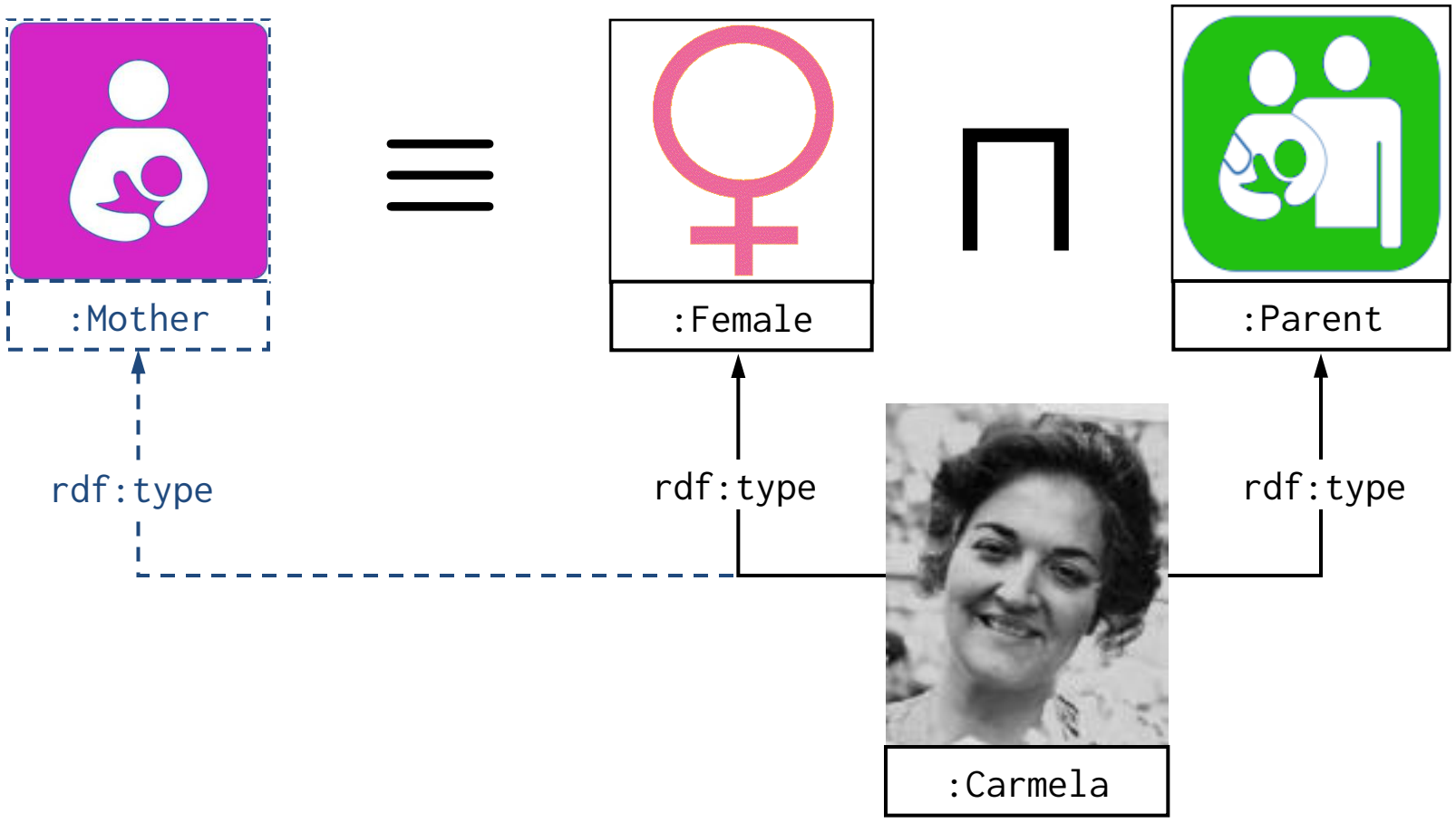
Name	Syntax	OWL key-term	DL
	Concept Definitions		
Atomic Concept	A	`owl:Class`	$\mathcal{ALC}$
Top Concept	$\top$	`owl:Thing`	$\mathcal{ALC}$
Bottom Concept	$\bot$	`owl:Nothing`	$\mathcal{ALC}$
Concept Negation	$\neg C$	`owl:complementOf`	$\mathcal{ALC}$
Concept Intersection	$C \sqcap D$	`owl:intersectionOf`	$\mathcal{ALC}$
Concept Union	$C \sqcup D$	`owl:unionOf`	$\mathcal{ALC}$
Nominal	$\{a_1, ..., a_n\}$	`owl:oneOf`	$\mathcal{O}$
Existential Restriction	$\exists R.C$	`owl:someValuesFrom`	$\mathcal{ALC}$
Universal Restriction	$\forall R.C$	`owl:allValuesFrom`	$\mathcal{ALC}$
Self Restriction	$\exists R.\mathsf{Self}$	`owl:hasSelf`	$\mathcal{R}$
Number Restriction	$\leq n\, R, \geq n\, R, = n\, R$	`owl:*cardinality`	$\mathcal{N}$
Qualified Number Restriction	$\leq n\, R.C, \geq n\, R.C, = n\, R.C$	`owl:*qualifiedCardinality`	$\mathcal{Q}$
	Concept Axioms (T-Box)		
Concept Inclusion	$C \sqsubseteq D$	`rdfs:subClassOf`	$\mathcal{ALC}$
	Role Definitions		
Role	R	`owl:*Property`	$\mathcal{ALC}$
Inverse Role	R^-	`owl:inverseOf`	$\mathcal{I}$
Universal Role	U	`owl:top*Property`	$\mathcal{R}$
	Role Axioms (R-Box)		
Role Inclusion	$R \sqsubseteq S$	`rdfs:subPropertyOf`	$\mathcal{H}$
Complex Role Inclusion	$R_1 \circ ... \circ R_n \sqsubseteq S$	`owl:propertyChainAxiom`	$\mathcal{R}$
Transitive Roles	$\mathsf{Trans}(R)$	`owl:TransitiveProperty`	$\mathcal{S}$
Functional Roles	$\mathsf{Func}(R)$	`owl:FunctionalProperty`	$\mathcal{F}$
Reflexive Roles	$\mathsf{Ref}(R)$	`owl:ReflexiveProperty`	$\mathcal{R}$
Irreflexive Roles	$\mathsf{Irref}(R)$	`owl:IrreflexiveProperty`	$\mathcal{R}$
Symmetric Roles	$\mathsf{Sym}(R)$	`owl:SymmetricProperty`	$\mathcal{I}$
Asymmetric Roles	$\mathsf{Asym}(R)$	`owl:AsymmetricProperty`	$\mathcal{R}$
Disjoint Roles	$\mathsf{Disj}(R, S)$	`owl:disjointPropertyWith`	$\mathcal{R}$
	Assertional Definitions		
(Named) Individual	a	(*RDF IRI or Literal*)	$\mathcal{ALC}$
	Assertional Axioms (A-Box)		
Role Assertion	$R(a, b)$	(*RDF triple*)	$\mathcal{ALC}$
Negative Role Assertion	$\neg R(a, b)$	`owl:NegativePropertyAssertion`	$\mathcal{ALC}$
Concept Assertion	$C(a)$	`rdf:type`	$\mathcal{ALC}$
Equality	$a = b$	`owl:sameAs`	$\mathcal{ALC}$
Inequality	$a \neq b$	`owl:differentFrom`	$\mathcal{ALC}$

owl:intersectionOf (⊓) [i]



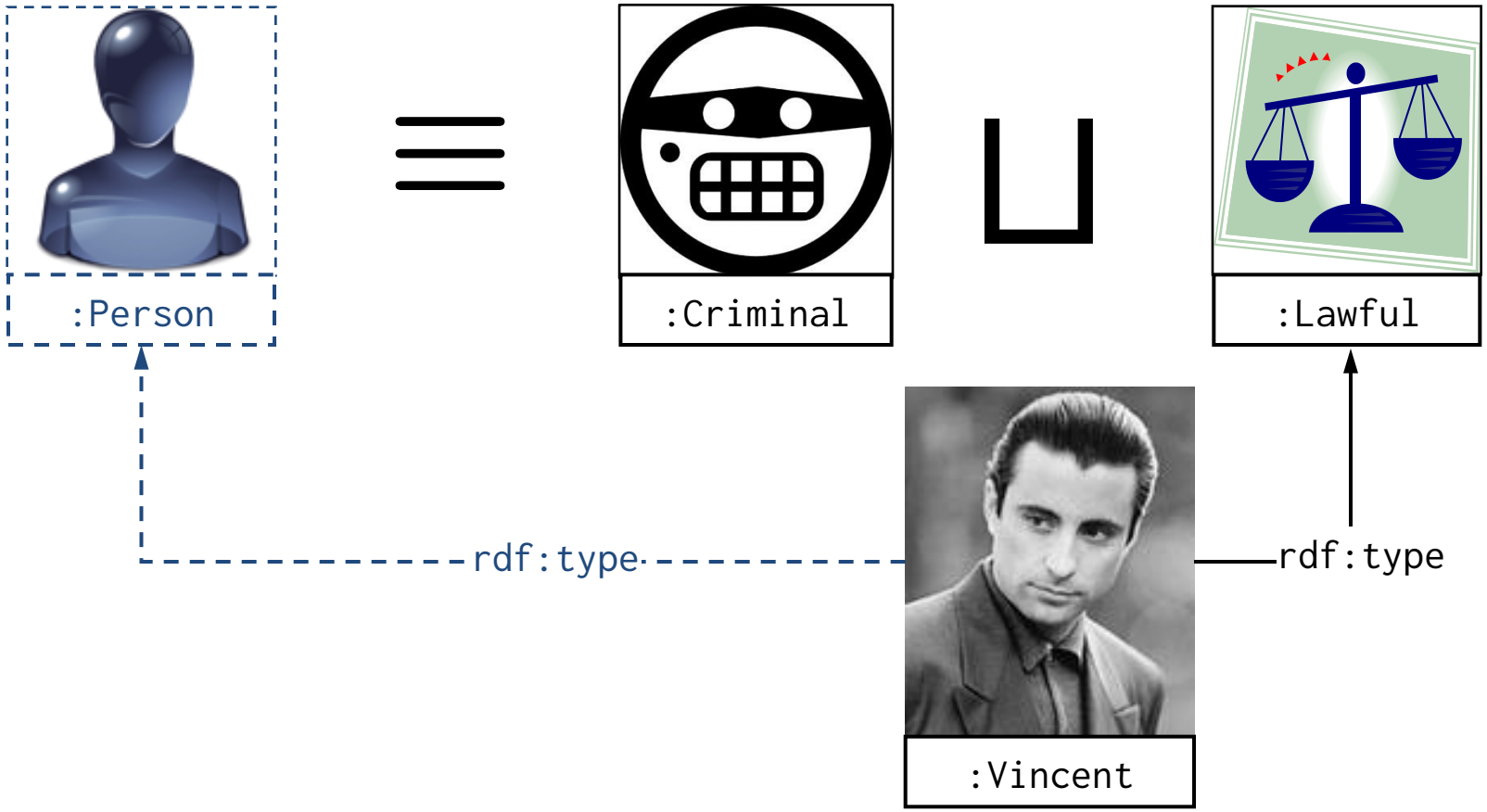

```
:Carmela rdf:type :Mother .
:Mother rdfs:subClassOf [ owl:intersectionOf ( :Female :Parent ) ]
⇒ :Carmela rdf:type :Female , :Parent .
```

owl:intersectionOf (⊓) [ii]



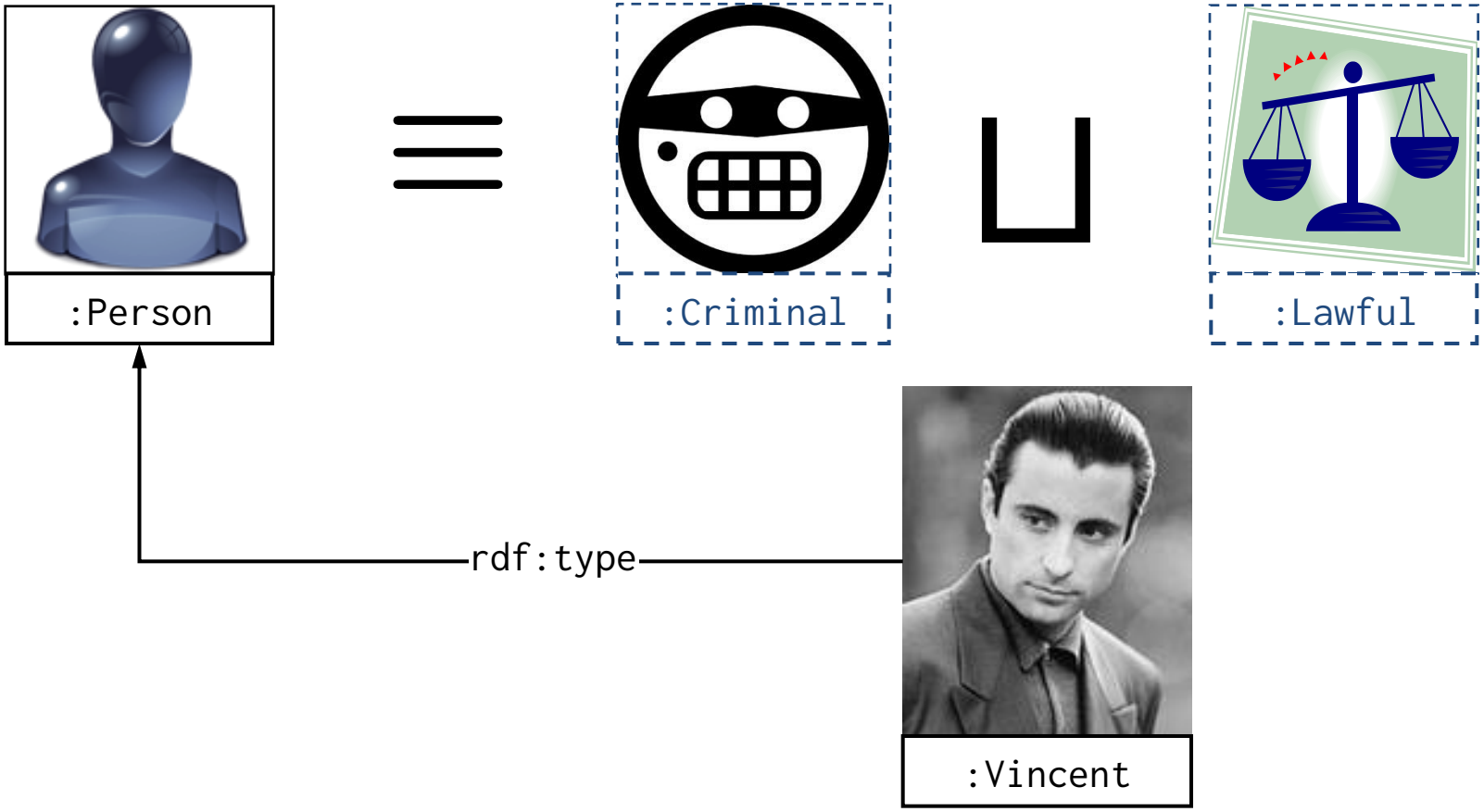

```
:Carmela rdf:type :Female , :Parent .
:Mother owl:equivalentClass [ owl:intersectionOf ( :Female :Parent ) ]
⇒ :Carmela rdf:type :Mother .
```

owl:unionOf (⊔) [i]



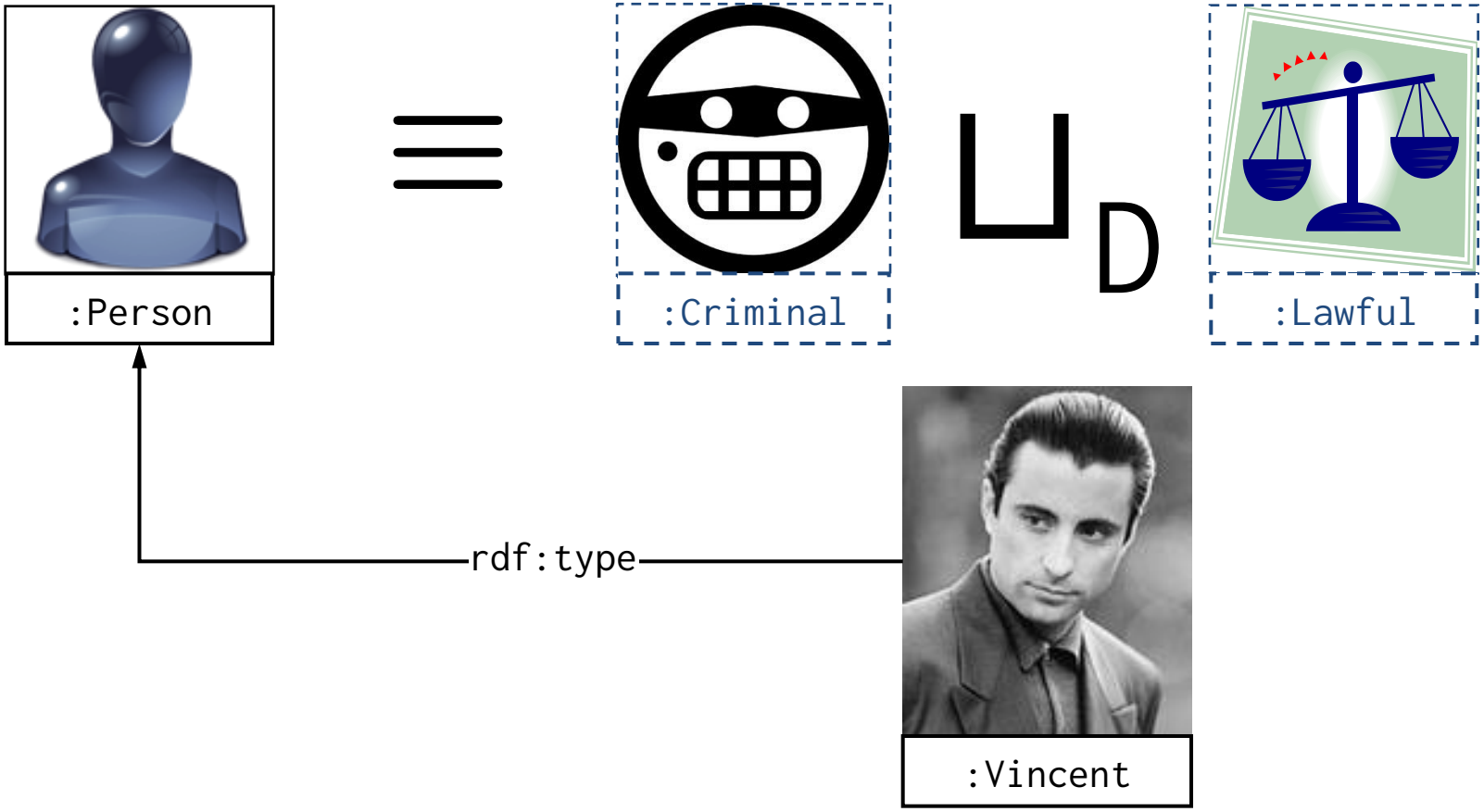

```
:Vincent rdf:type :Lawful .
:Person owl:equivalentClass [ owl:unionOf ( :Criminal :Lawful ) ]
⇒  :Vincent rdf:type :Person .
```

owl:unionOf (⊔) [ii]



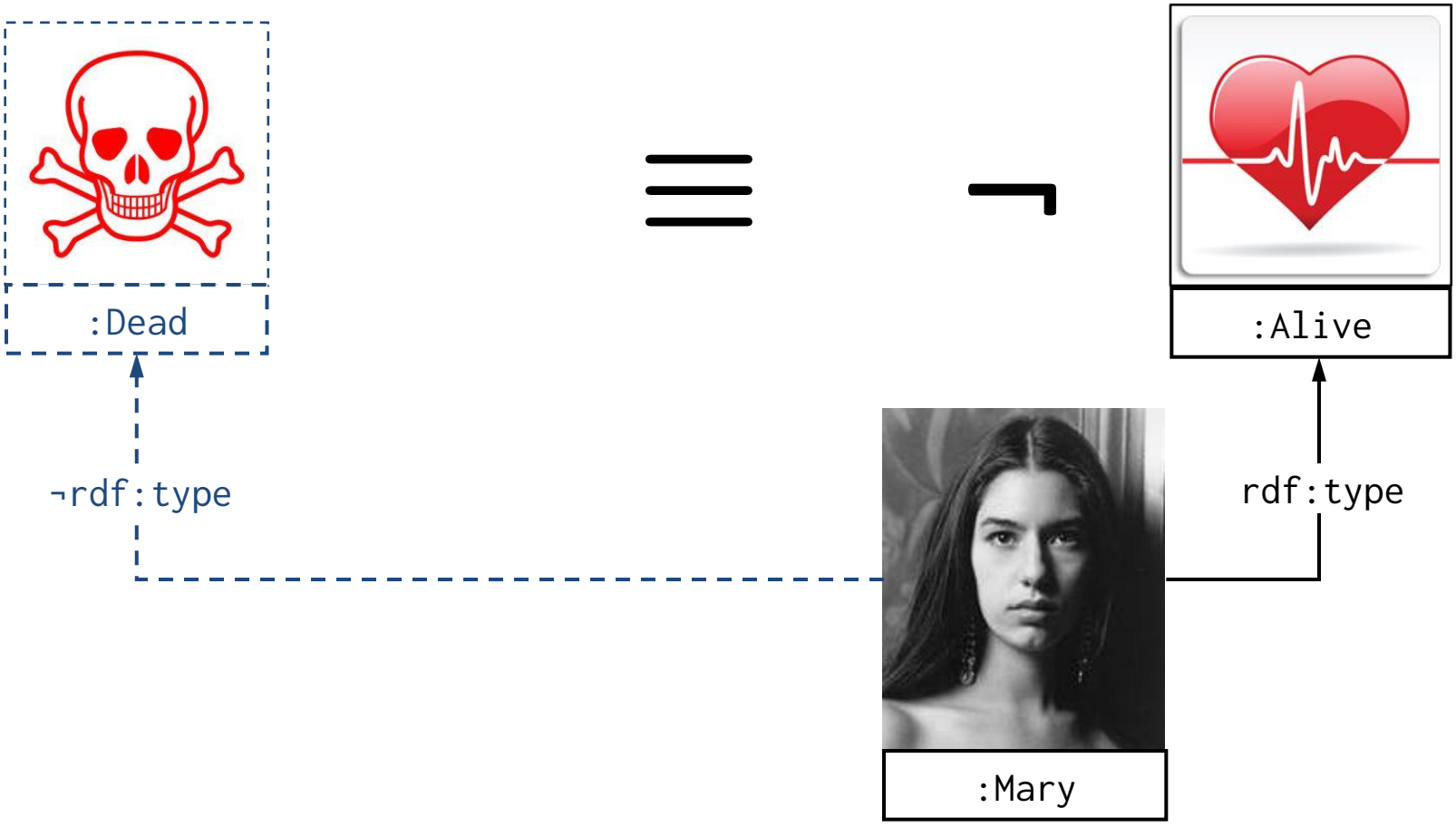

```
:Vincent rdf:type :Person .
:Person owl:equivalentClass [ owl:unionOf ( :Criminal :Lawful ) ]
⇒ # :Vincent must be either :Lawful or :Criminal (or both)
```

owl:disjointUnionOf ($\sqcup_D$)




```
:Vincent rdf:type :Person .
:Person owl:equivalentClass
     [ owl:disjointUnionOf ( :Criminal :Lawful ) ]
⇒   # :Vincent must be either :Lawful or :Criminal (not both)
```

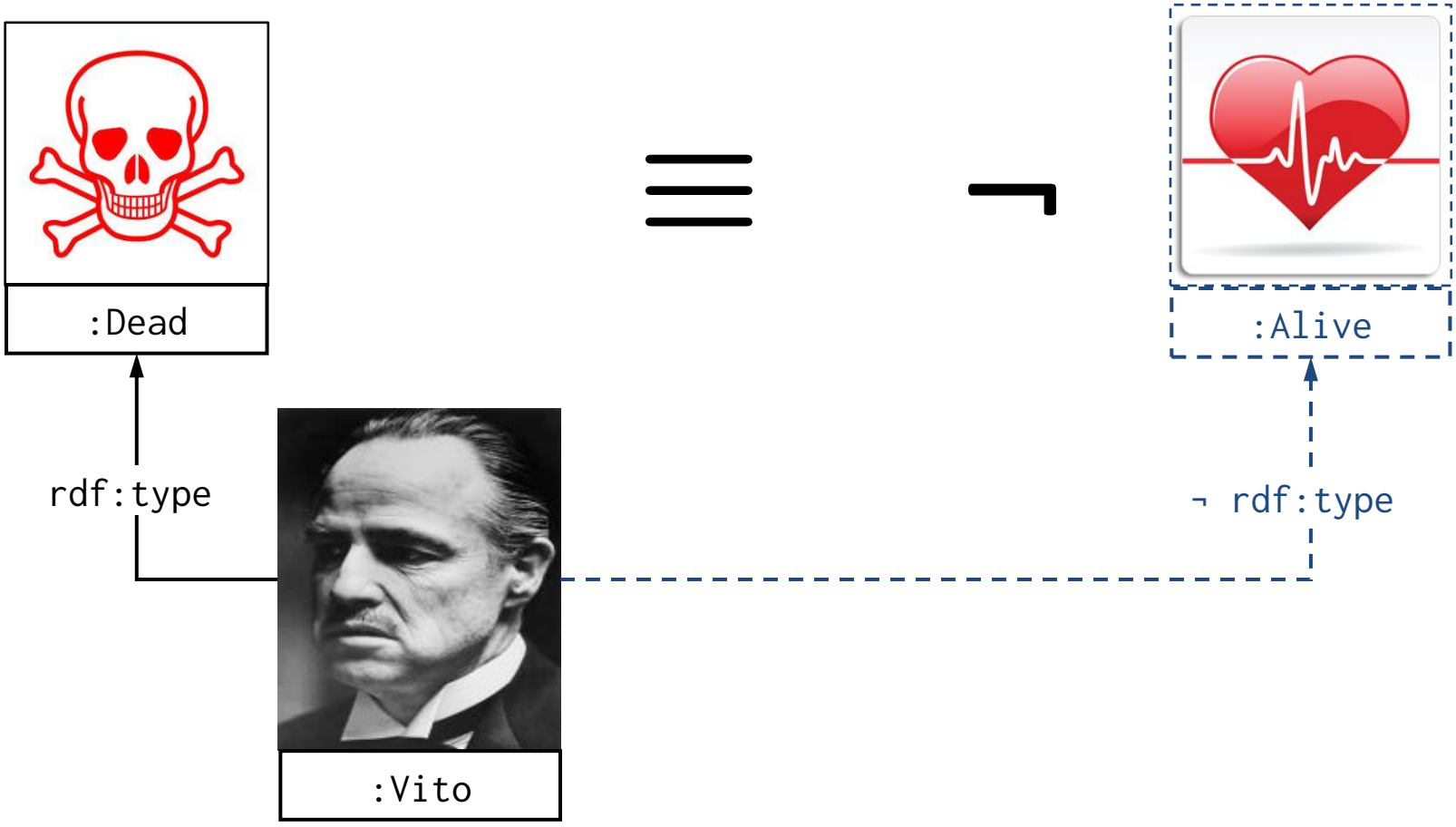
owl:complementOf (¬) [i]




```
:Mary rdf:type :Alive .
:Dead owl:equivalentClass [ owl:complementOf :Alive ]
⇒ [] owl:sourceIndividual :Mary ; owl:targetProperty rdf:type ;
       owl:targetIndividual :Dead .
```


WARNING
SPOILER
ALERT

owl:complementOf (¬) [ii]

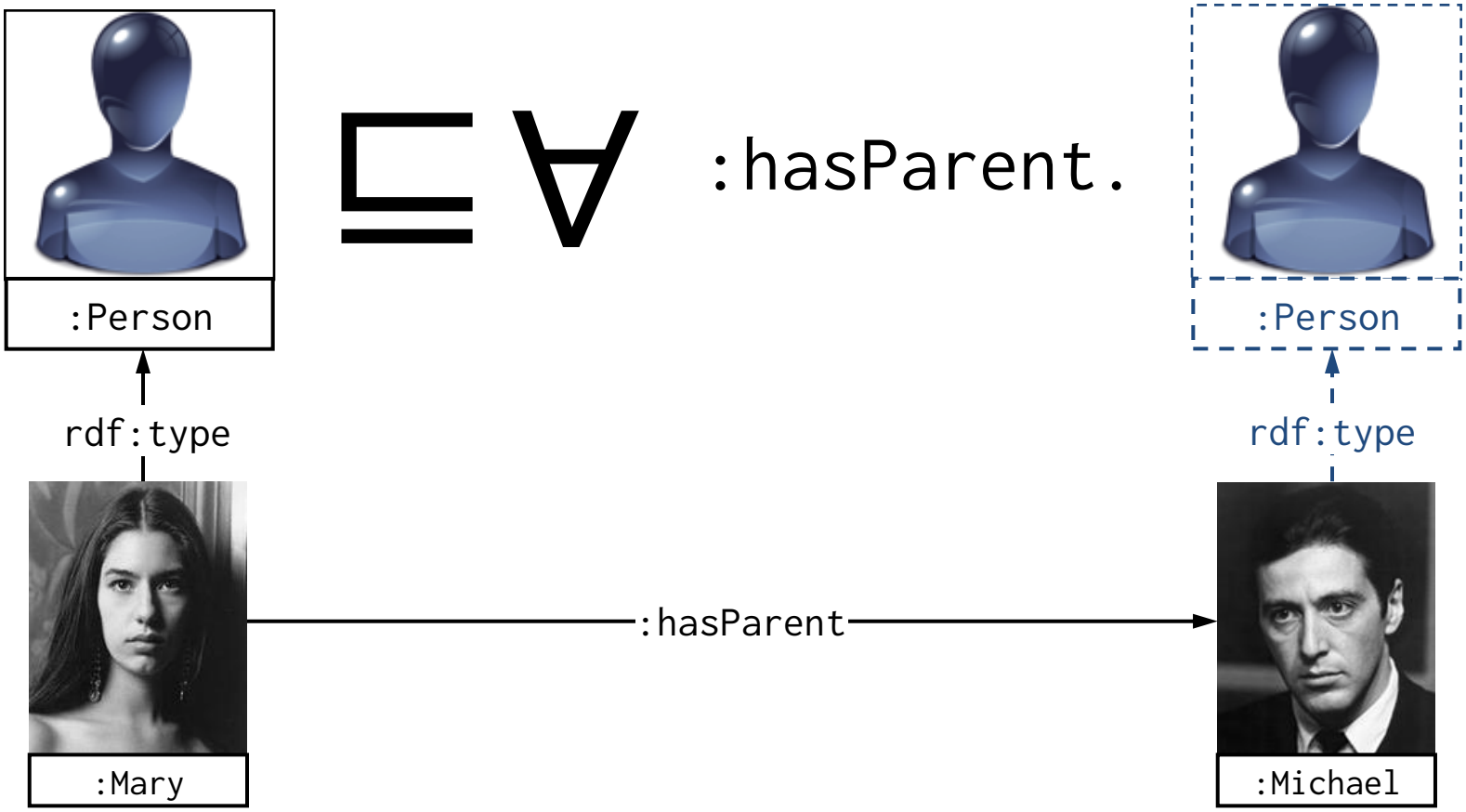



```
:Vito rdf:type :Dead .
:Dead owl:equivalentClass [ owl:complementOf :Alive ]
⇒ [] owl:sourceIndividual :Vito ; owl:targetProperty rdf:type ;
       owl:targetIndividual :Alive .
```

owl:oneOf ({})

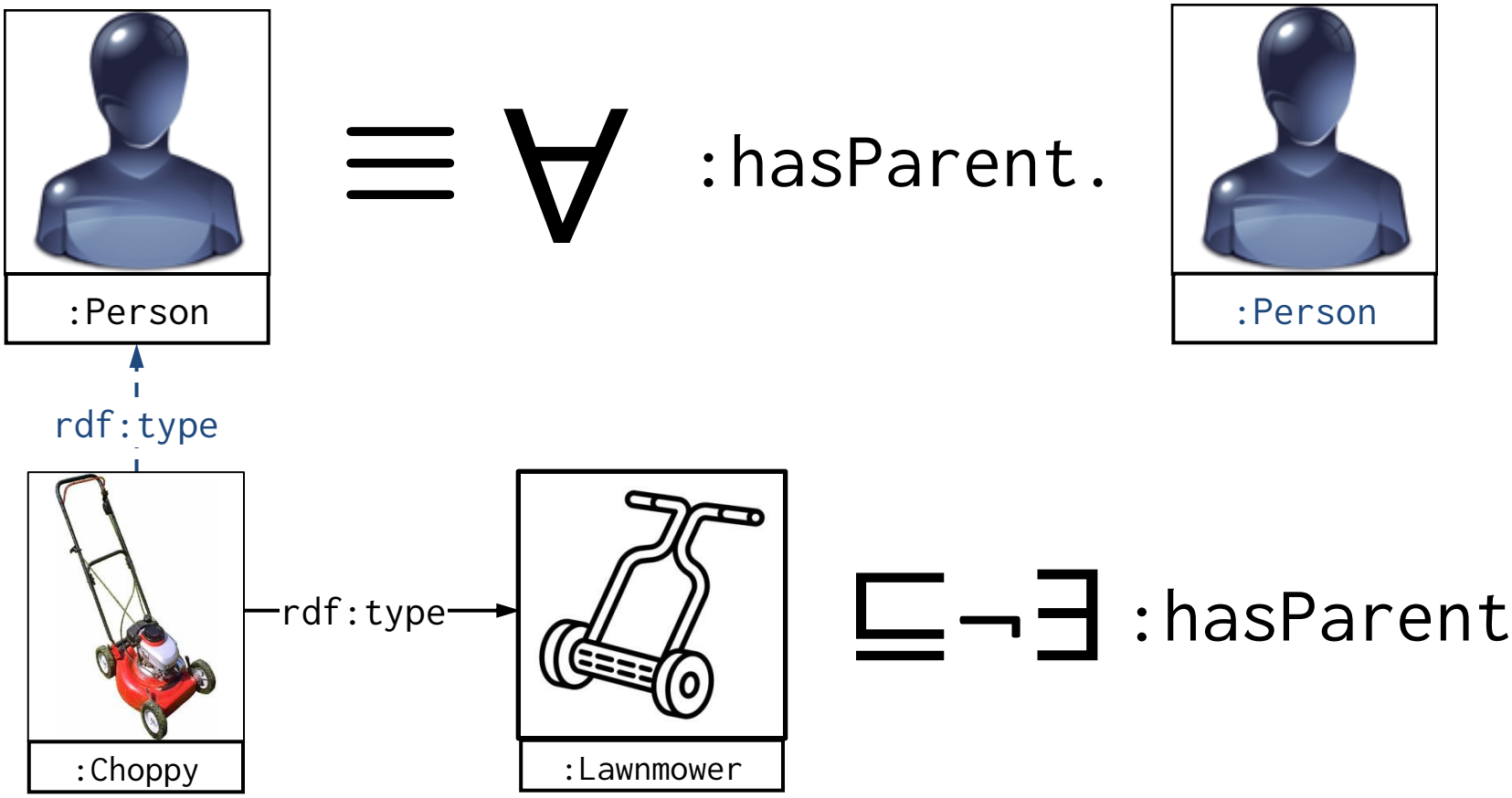

```
:Godfather owl:equivalentClass
   [ owl:oneOf (:Vito :Michael :Vincent) ]
⇒ :Vito rdf:type :Godfather .
⇒ :Michael rdf:type :Godfather .
⇒ :Vincent rdf:type :Godfather .
```

owl:allValuesFrom (∀) [i]




```
:Mary rdf:type :Person ; :hasParent :Michael .
:Person rdfs:subClassOf
  [ owl:allValuesFrom :Person ; owl:onProperty :hasParent ]
⇒  :Michael rdf:type :Person .
```

owl:allValuesFrom (∀) [ii]



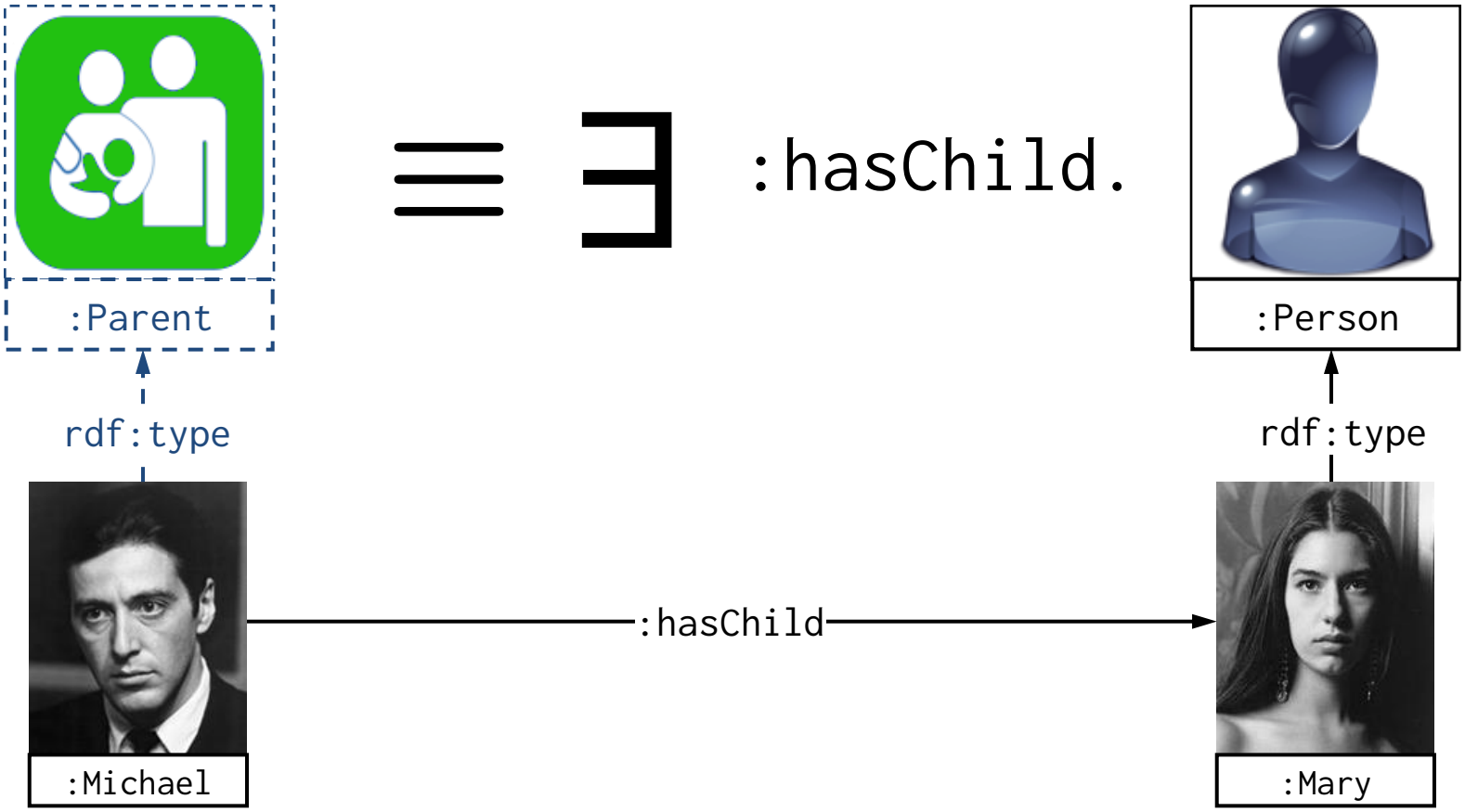
Need to be careful using ≡ with owl:allValuesFrom!

If all of X's parents are :Persons, X is a :Person.

But :Choppy does not have parents, so all its parents are trivially anything!

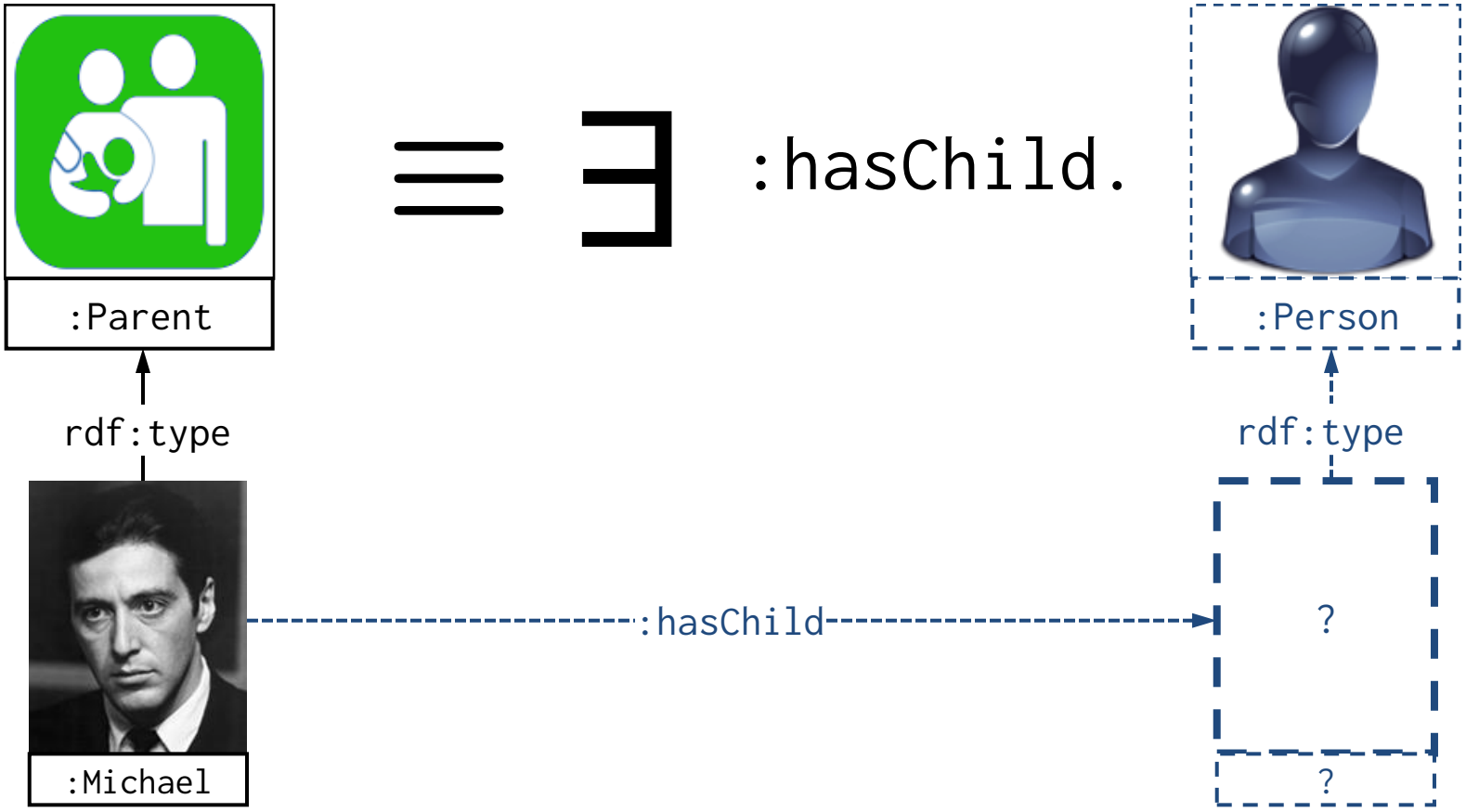
Thus :Choppy must be a :Person!

owl:someValuesFrom (∃) [i]



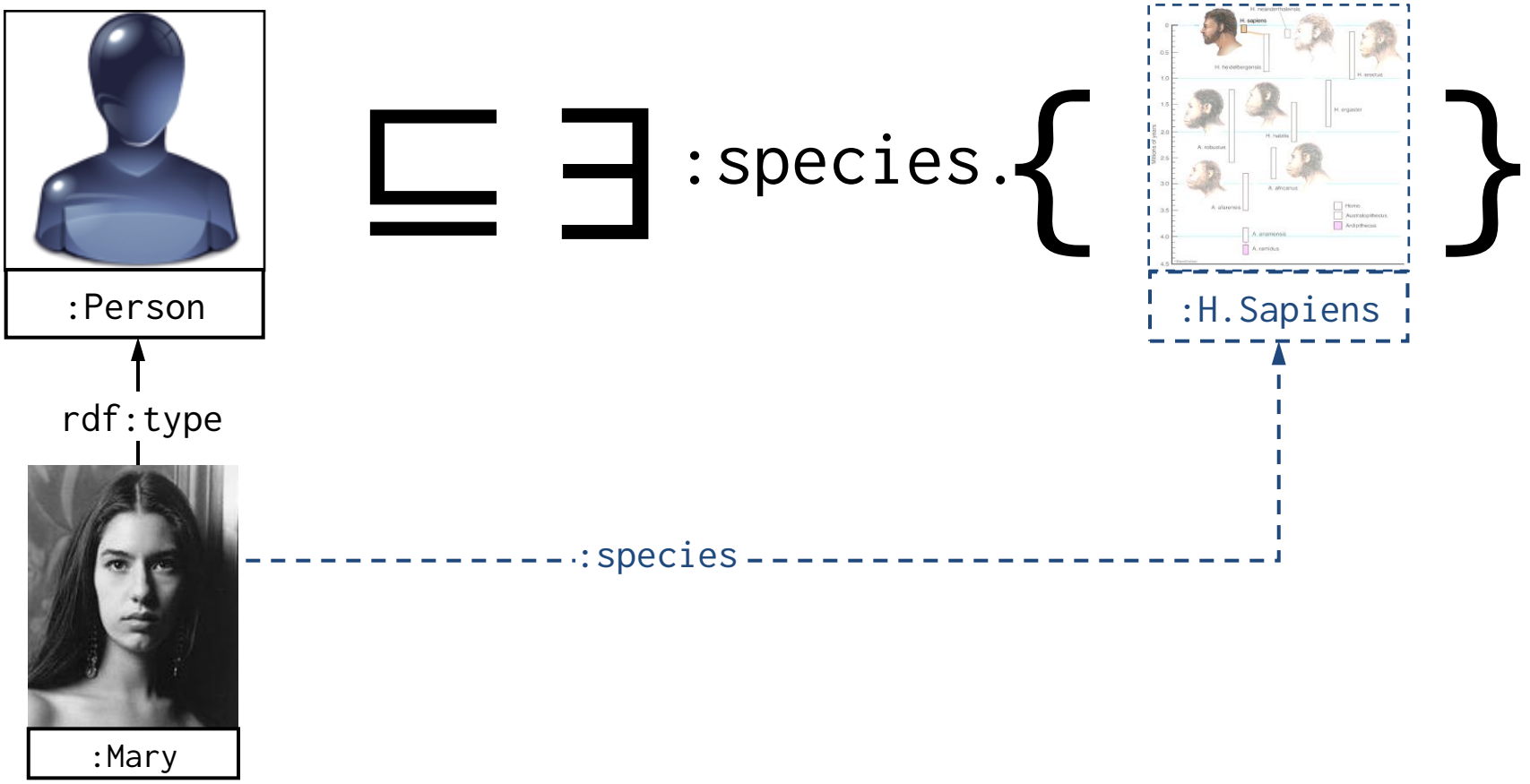

```
:Michael :hasChild :Mary . :Mary rdf:type :Person .
:Parent owl:equivalentClass
  [ owl:someValuesFrom :Person ; owl:onProperty :hasChild ]
⇒  :Michael rdf:type :Parent .
```

owl:someValuesFrom (∃) [ii]



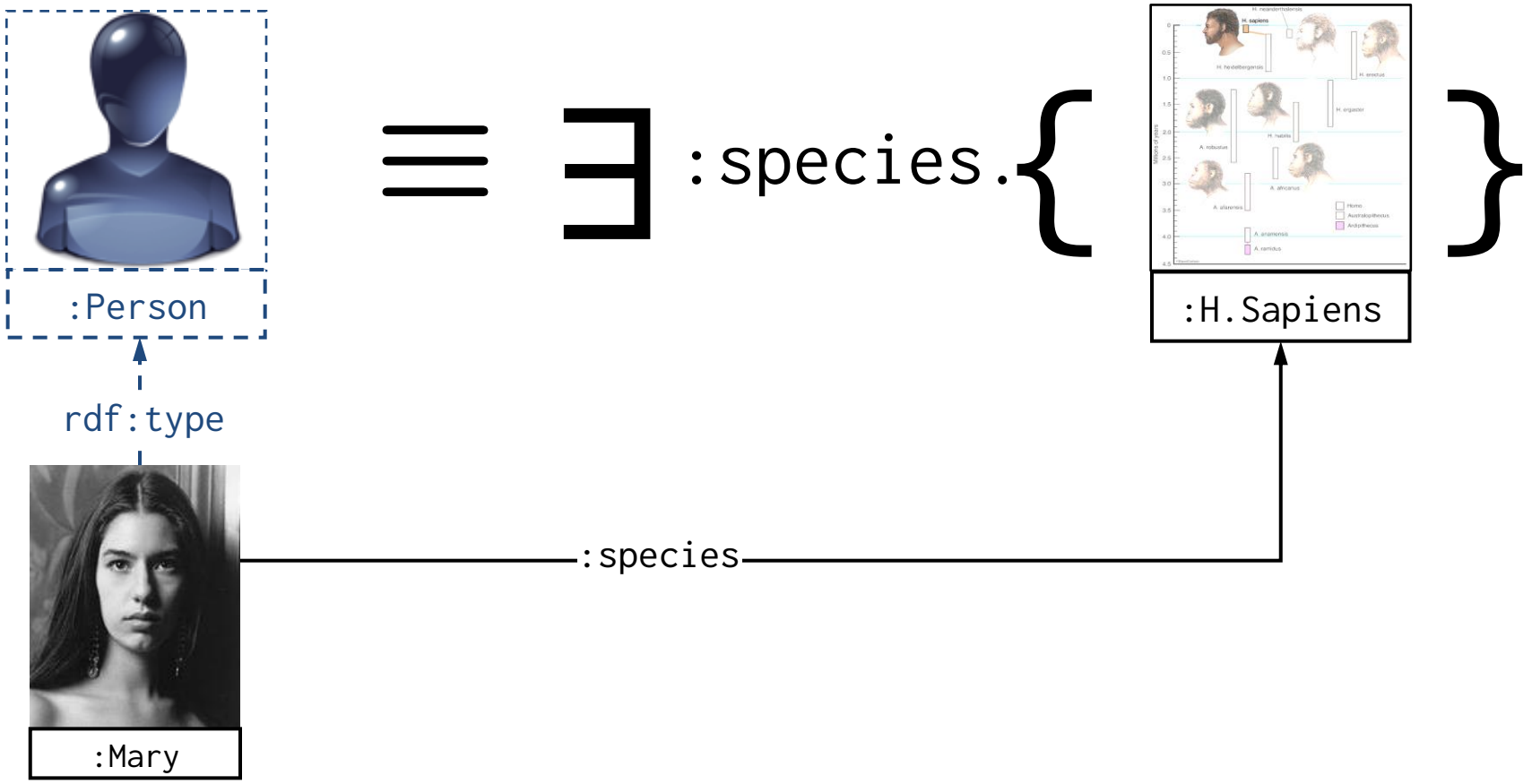

```
:Michael rdf:type :Parent .
:Parent owl:equivalentClass
  [ owl:someValuesFrom :Person ; owl:onProperty :hasChild ]
⇒  :Michael :hasChild _:someone . _:someone rdf:type :Person .
```

owl:hasValue (∃P.{x}) [i]



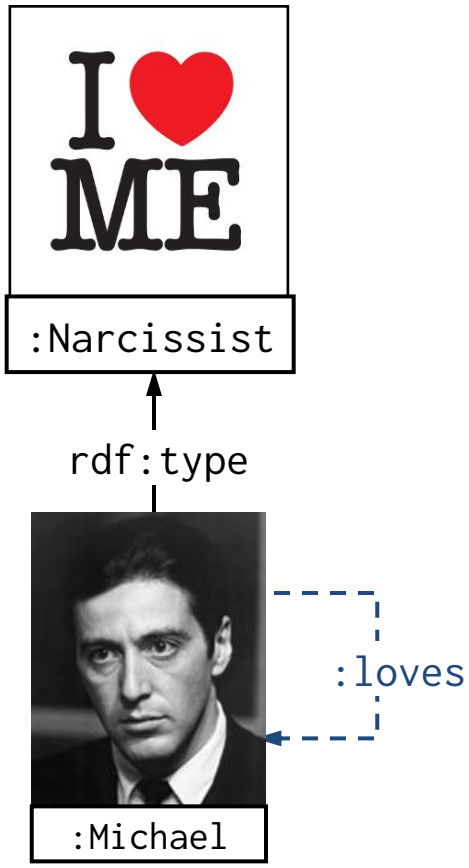

```
:Mary rdf:type :Person .
:Person rdfs:subClassOf
  [ owl:hasValue :H.Sapiens ; owl:onProperty :species ]
⇒   :Mary :species :H.Sapiens .
```

owl:hasValue (∃P.{x}) [i]




```
:Mary :species :H.Sapiens .
:Person owl:equivalentClass
  [ owl:hasValue :H.Sapiens ; owl:onProperty :species ]
⇒  :Mary rdf:type :Person .
```

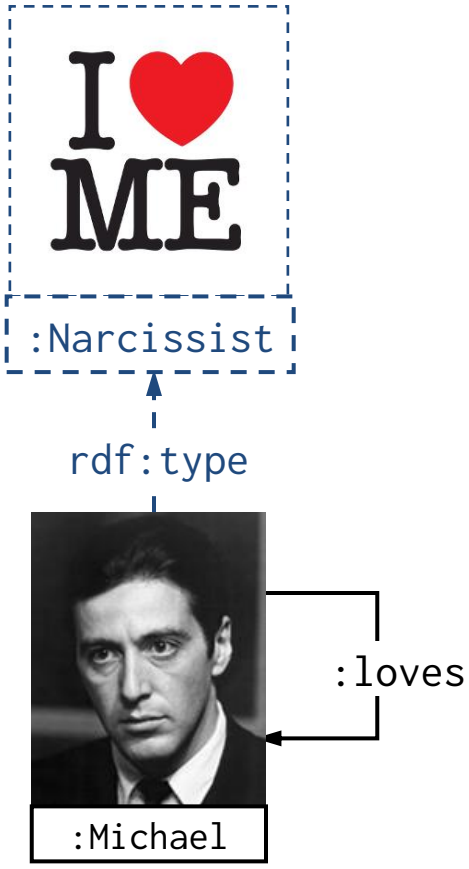
owl:hasSelf (Self) [i]



$\sqsubseteq$ Self(*:loves*)

```
:Michael rdf:type :Narcissist .
:Narcissist rdfs:subClassOf
  [ owl:hasSelf true ; owl:onProperty :loves ]
⇒  :Michael :loves :Michael .
```

owl:hasSelf (Self) [ii]



≡ Self(:loves)

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

Cardinality restrictions (≥, ≤,=)

- Define a class with a given number of values for a property:
  - Exact: `:Person ⊑ = 2 (:hasBioParent)`

```
:Person rdfs:subClassOf [ owl:cardinality 2 ;
                          owl:onProperty :hasBioParent ] .
```

  - Max: `:Monogamist ⊑ ≤ 1 (:hasCurrentSpouse)`

```
:Monogamist rdfs:subClassOf [ owl:maxCardinality 1 ;
                              owl:onProperty :hasCurrentSpouse ] .
```

  - Min: `:Parent ≡ ≥ 1 (:hasChild)`

```
:Parent owl:equivalentClass [ owl:minCardinality 1 ;
                              owl:onProperty :hasChild ] .
```

Qualified cardinality restrictions (≥, ≤,=)

- Define a class with a given number of values from a given class for a property:
  - Exact: `:Biped ≡ =2 (:hasLimb.Leg)`

```
:Biped owl:equivalentClass [ owl:qualifiedCardinality 2 ;
                             owl:onProperty :hasLimb ;
                             owl:onClass :Leg ] .
```

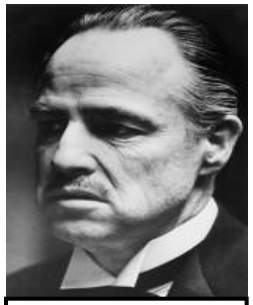
  - Max: `:Mafia ⊑ ≤ 1 (:hasCurrentMember.Godfather)`

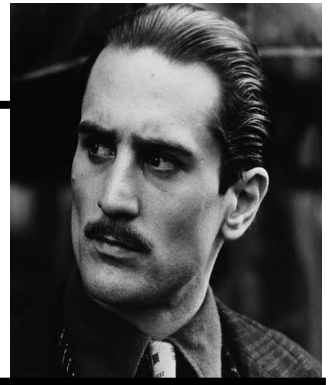
```
:Mafia rdfs:subClassOf [ owl:maxQualifiedCardinality 1 ;
                         owl:onProperty :hasCurrentMember ;
                         owl:onClass :Godfather ] .
```

  - Min: `:Twin ⊑ ≥ 1 (:sibling.Twin)`

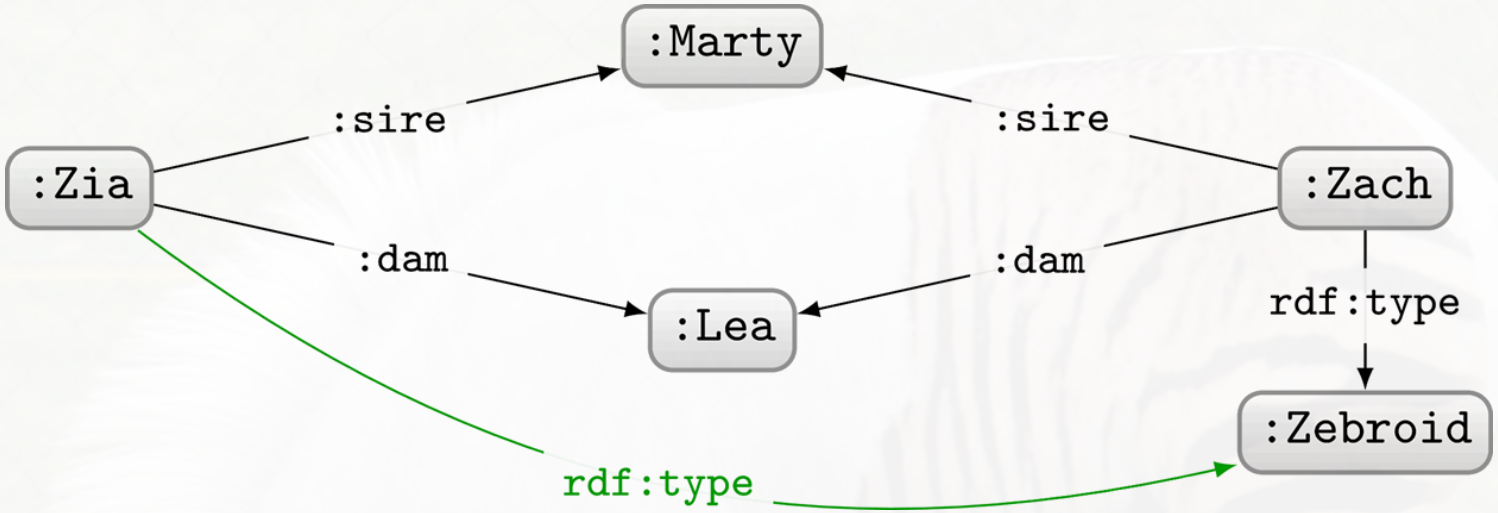
```
:Twin rdfs:subClassOf [ owl:minQualifiedCardinality 1 ;
                        owl:onProperty :sibling ;
                        owl:onClass :Twin ] .
```

Slides are examples, not definitions

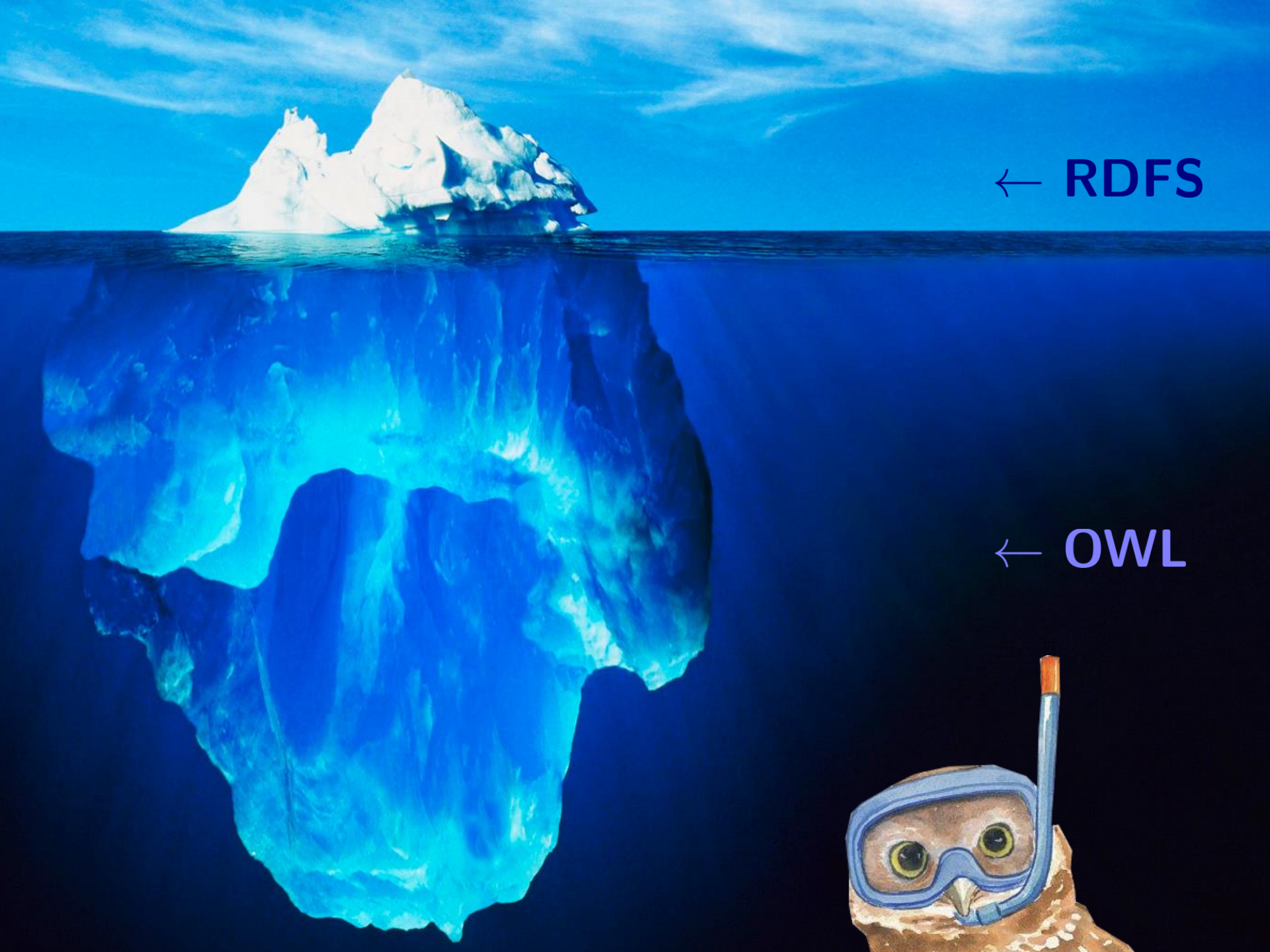
:Godfather ≡ {  :Vito ,  :Michael ,  :Vincent }

:Godfather ← rdf:type —  :VitoYoung

⇒ :VitoYoung must be owl:sameAs :Vito or :Michael or :Vincent



- `sire` is a sub-property of `parent`
- `dam` is a sub-property of `parent`
- A `Zebroid` has exactly one `parent` a `Zebra`
- A `Zebroid` has exactly one `parent` a (¬`Zebra` and a `Equine`)
- A `Zebroid` is a sub-class of `Equine`
- An `Equine` has exactly two `parents`
- Two things cannot be related by `sire` and `dam` at the same time


← RDFS
← OWL

Questions?