

# LINKING BACKGROUND INFORMATION

INTERLINK D4 Appendix 4, Michel Böhms (TNO)

With input from EU V-CON and bSI LDWG



- › Basic Linking
- › More Background Info on L1/L2/L3 semantic levels
- › Advanced Linking Issues



- › For Link Sets we distinguish:
  - › ‘Linking Ontologies (LO)’ containing ontology/class-level links
    - › Alignment Ontologies (AO) for Data Linking (**‘Basic Linking’**)
    - › Conversion Rule Sets (CRS) for Data Conversion (‘Advanced Linking’ involving more complex rules’)
  - › ‘Linking Data Sets (LDS)’ containing data set/individual-level links only (**‘Basic Linking’**)
- › Only Basic Linking is discussed here further as expressed in RDF/RDFS/OWL(subset)
- › In case SHACL is used in future over OWL
  - › owl:Class becomes rdfs:Class
  - › owl:DatatypeProperty/ObjectProperty both become rdf:Property
  - › owl:sameAs can become alternative like skos:exactMatch

# BASIC LINKING: FOUR RELEVANT DIMENSIONS



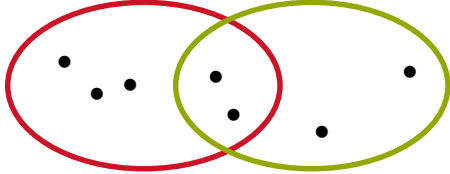
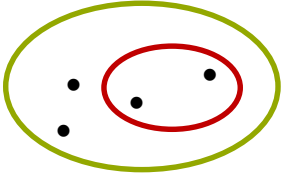
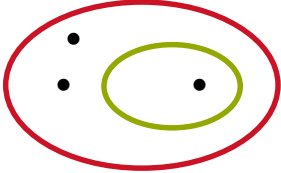
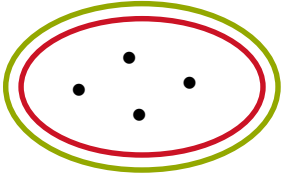
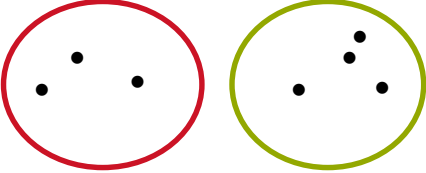
1. Three key Meta concepts:
  - Classes (owl:Class)
  - Properties (owl:DatatypeProperty or owl:ObjectProperty)
  - Individuals (owl:NamedIndividual)
2. 12 “Venn situations” representing intended semantics
  - 5 for Classes, same 5 for Properties, 2 others involving individuals
3. Three Semantic Levels for classes
4. Asserted versus Inferred data

# BASIC LINKING: ASSUME ...

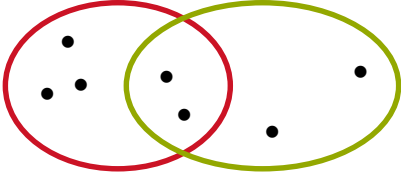


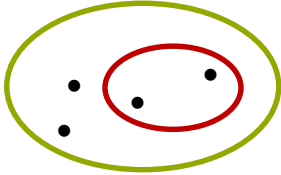
- › 2 ontologies: x and y
  - › Where x, y are prefixes for their name space URIs
  
- › asserted or inferred
  - › 1 Class A in ontology x                      > x:A
  - › 1 Class B in ontology y                      > y:B
  - › 1 Property in ontology x                      > x:c
  - › 1 Property in ontology y                      > y:d
  - › 1 Individual in ontology x                      > x:a of type x:A
  - › 1 Individual in ontology y                      > y:b of type y:B
  
- › Classes represented by circles, Individuals represented by dots in the next three figures

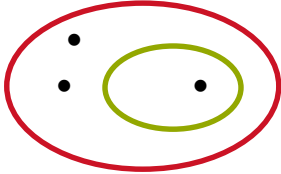
# BASIC LINKING: 5 VENN SITUATIONS FOR CLASSES

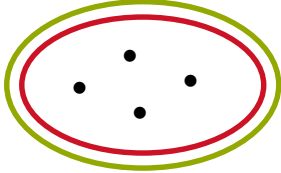
1.  The “default”, unconstrained situation.  
Independent classes, no linking rules relevant
2.   $x:A \text{ rdfs:subClassOf } y:B$
3.   $y:B \text{ rdfs:subClassOf } x:A$
4.  2. AND 3.  
 $x:A \text{ owl:EquivalentClass } y:B$  (symmetric)
5.   $x:A \text{ owl:DisjointWith } y:B$  (symmetric)

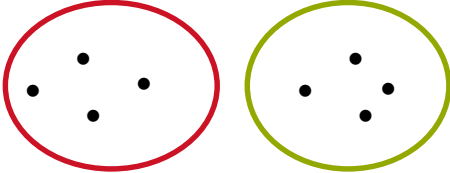
# BASIC LINKING: 5 VENN SITUATIONS FOR PROPERTIES

1. 

The “default”, unconstrained.  
Independent properties, no mapping rule  
(any area CAN also be empty)
2. 

`x:c rdfs:subPropertyOf y:d`
3. 

`y:d rdfs:subPropertyOf x:c`
4. 

2. AND 3.  
`x:A owl:EquivalentProperty y:B (symmetric)`
5. 

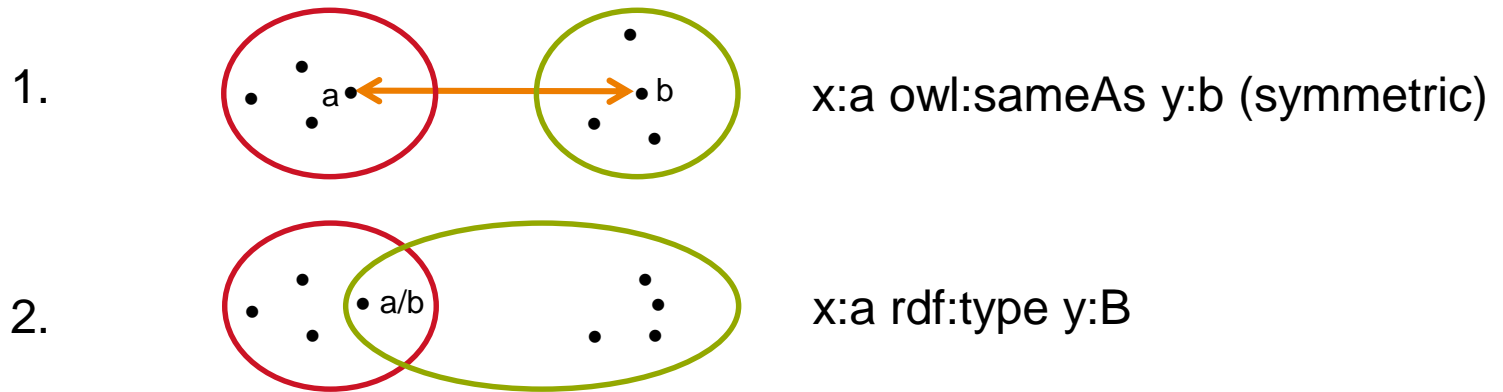
`x:c owl:propertyDisjointWith y:d (symmetric)`



- › Declared Classes (`rdf:type owl:Class`), or
- › Restriction Classes (`rdf:type owl:RestrictionClass`)
  - › Cardinalities (qualified or not)
  - › `hasValue`
  - › `someValuesFrom` / `allValuesFrom`



# BASIC LINKING: 2 VENN SITUATIONS FOR INDIVIDUALS



# BASIC LINKING: LINK TYPES REPRESENTED IN RDF/RDFS/OWL



## › **Class/Property level (in Alignment Ontology)**

- › `x:A rdfs:subClassOf y:B`                      (`== y:B rdfs:subClassOf x:A`)
- › `x:A owl:EquivalentClass y:B`              (syntactic sugar really)
- › `x:c rdfs:subPropertyOf y:d`                  (`==y:d rdfs:subPropertyOf x:c`)
- › `x:A owl:EquivalentProperty y:B`          (syntactic sugar really)
- › `x:A owl:disjointWith y:B`
- › `x:c owl:propertyDisjointWith y:d`

## › **Individual level (in Alignment Ontology (reference individuals!) or Linking Data Set)**

- › `x:a owl:sameAs y:b`

## › **Individual <> class level**

- › `x:a rdf:type y:B` (`== y:B rdf:type x:A`)

- › `==` means: same type of link

# BASIC LINKING: INFERENCE POTENTIAL



- › Depending on the semantic level of the classes in ontologies more or less can be inferred from assertions (available in ontologies, LOs, LDSs and/or (individual) data sets)
- › Standard OWL2 inferences (‘entailments regimes’ or ‘meta-rules’) apply
- › Example meta-rule
  - › **IF** (x:a rdf:type x:A) AND (x:A rdfs:subClassOf y:B)
  - › **THEN** (x:a rdf:type y:B)
- › Where “x:A rdfs:subClassOf y:B” would be an actual “Link”

# BASIC LINKING: TWO MORE META-RULE EXAMPLES



- › **IF** (x:a rdf:type x:A) AND  
(x:A rdf:type owl:Class AND  
rdfs:subClassOf [ a owl:Restriction ;  
owl:hasValue ? ;  
owl:onProperty x:c  
] .)
- › **THEN** (x:a x:c ?)
- › **IF** (x:a rdf:type x:A) AND (y:b rdf:type y:B) AND  
(x:a owl:sameAs y:b) AND (x:a x:c x:z)
- › **THEN** (y:b x:c x:z)

# BASIC LINKING: 3 SEMANTIC LEVELS FOR CLASSES



- › L1: Class without restrictions (the class is just ‘declared’)
  - › L2: Class with only “necessary” Restrictions
  - › L3: Class with “necessary & sufficient” (n&s) Restrictions
- 
- › An ontology can have a mix of 1/2/3-type classes
  - › An ontology can be “semantically complete”, having only L3 classes
  - › In practice an ontology is typically a mix of many L1s, some L2s and often no L3s, i.e. “semantically not that strong”

# BASIC LINKING: NINE RESULTING (BI-DIRECTIONAL) LINKING SITUATIONS



› We distinguish the following 9 situations for classes in ontology x and y:

x/y	L1	L2	L3
L1	Situation 1	Situation 2	Situation 5
L2	Situation 3	Situation 4	Situation 7
L3	Situation 6	Situation 8	Situation 9

- Situation 1: L1/L1: worst case (but often encountered in practice)
- Situation 9: L3/L3: best case (but very unlikely)
- L3 often not needed/relevant
  - `rdf:type`'s are often asserted, so no automatic classification is required
  - Links are often asserted, so automatic linking not required
  - So we can simplify to ...

# BASIC LINKING: FOUR MOST RELEVANT LINKING SITUATIONS

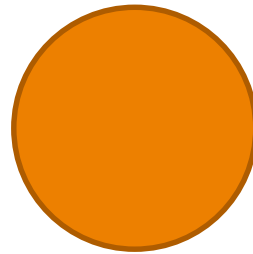


x/y	L1	L2
L1	Situation 1	Situation 2
L2	Situation 3	Situation 4

- › We will now analyse a simple example and show for each of the four situations what can be inferred (for a given intended linking)



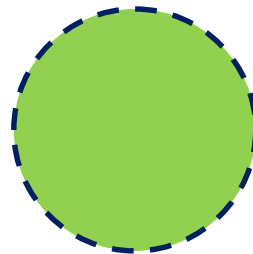
# EXAMPLE: LEGENDA



**Class**

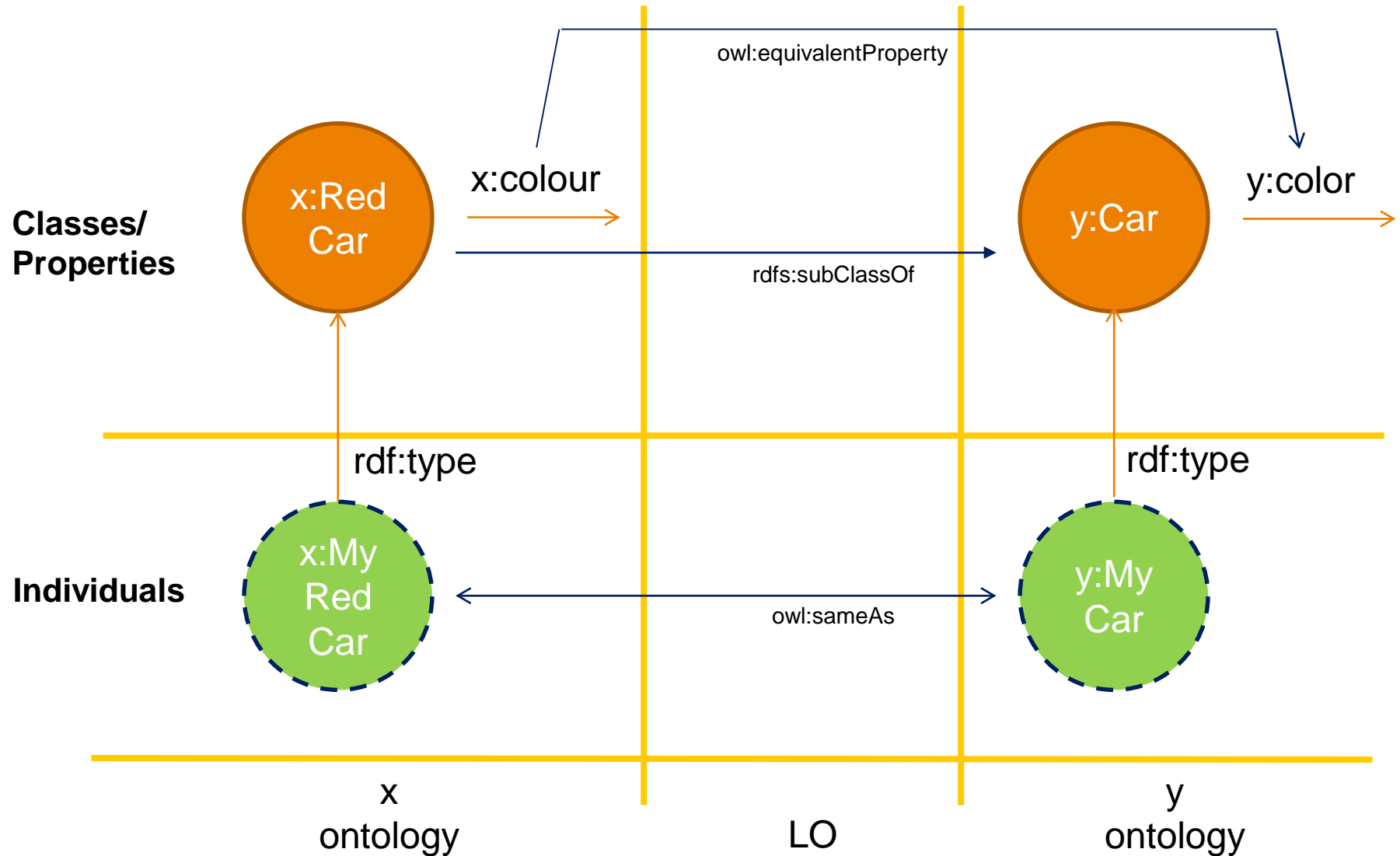


**Property**

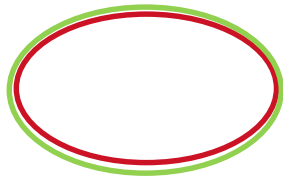
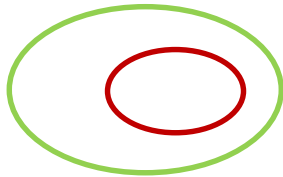


**Individual**

# EXAMPLE: GRAPHICALLY (BOTH CLASSES ARE L1)



# EXAMPLE: INTENDED LINKING BETWEEN X AND Y CLASSES/PROPERTIES IN VENN- DIAGRAM



x:RedCar is a subclass of y:Car .

x:colour is equivalent property for y:color .



This knowledge becomes part of the LO

# EXAMPLE: SITUATION 1: L1/L1 (FULL CORRESPONDENCE TO DIAGRAM)



## › Ontology x asserts:

```
x:RedCar rdf:type owl:Class .  
x:colour rdf:type owl:DatatypeProperty ;  
    rdfs:range xsd:string .
```

## › Ontology y asserts:

```
y:Car rdf:type owl:Class .  
y:color rdf:type owl:DatatypeProperty ;  
    rdfs:range xsd:string .
```

## › LO asserts:

```
x:RedCar rdfs:subClassOf y:Car .  
x:colour owl:equivalentProperty y:color .
```

# EXAMPLE: SITUATION 1: L1/L1

## WHAT CAN BE INFERRED?



### › In case we first extra assert

- › `x:MyRedCar rdf:type x:RedCar .`
- › we can infer:
  - › `x:MyRedCar rdf:type y:Car .`

### › In case we next extra assert

- › `x:MyRedCar x:colour "red"^^xsd:string .`
- › we can extra infer:
  - › `x:MyRedCar y:color "red"^^xsd:string .`

### › In case we next extra assert:

- › `y:MyCar rd:type y:Car .`
- › `x:MyRedCar owl:sameAs y:MyCar . *`
- › we can extra infer:
  - › `y:MyCar x:colour "red"^^xsd:string .`
  - › `y:MyCar y:color "red"^^xsd:string .`

### › In case we first extra assert

- › `y:MyCar rdf:type y:Car .`
- › (we cannot infer:
  - › `y:MyCar rdf:type x:RedCar .`)

### › In case we next extra assert

- › `y:MyCar y:color "red"^^xsd:string .`
- › we can extra infer:
  - › `x:MyCar x:colour "red"^^xsd:string .`

### › In case we next extra assert:

- › `x:MyRedCar rdf:type x:RedCar .`
- › `y:MyCar owl:sameAs x:MyRedCar . *`
- › we can extra infer:
  - › `x:MyRedCar y:color "red"^^xsd:string .`
  - › `x:MyRedCar x:colour "red"^^xsd:string .`

\* In  
Linking  
Data Set

# EXAMPLE: SITUATION 2: L1/L2 – ONTOLOGY Y GETS SMARTER (L2)



- › Since y:Car has no restriction wrt the relevant property y:color this class is already L2 wrt this property
- › I.e. ontology y cannot be made 'smarter' wrt this property
- › So the situation is the same as Situation 1
- › Same inferences as in Situation 1

# EXAMPLE: SITUATION 3: L2/L1 - ONTOLOGY X GETS SMARTER (L2)



- › Ontology x asserts as extra in **red**:
  - › `x:RedCar rdf:type owl:Class ;`  
`rdfs:subClassOf [ a owl:Restriction ;`  
`owl:hasValue "red"^^xsd:string ;`  
`owl:onProperty x:colour`  
`].`
- › Ontology y stays the same (i.e. stays level L1)



# EXAMPLE: SITUATION 3: L2/L1

## WHAT CAN BE INFERRED?

### › In case we first extra assert

- › `x:MyRedCar rdf:type x:RedCar .`
- › we can infer:
  - › `x:MyRedCar rdf:type y:Car .`
  - › `x:MyRedCar x:colour "red"^^xsd:string .`
  - › `x:MyRedCar y:color "red"^^xsd:string .`

### › In case we next extra assert:

- › `y:MyCar rd:type y:Car .`
- › `x:MyRedCar owl:sameAs y:MyCar . *`
- › we can extra infer:
  - › `y:MyCar x:colour "red"^^xsd:string .`
  - › `y:MyCar y:color "red"^^xsd:string .`

Which is consistent with the restriction...

\* In  
Linking  
Data Set

### › In case we first extra assert

- › `y:MyCar rdf:type y:Car .`
- › )we can~~not~~ infer:
  - › `y:MyCar rdf:type x:RedCar .`

### › In case we next extra assert

- › `y:MyCar y:color "red"^^xsd:string .`
- › we can extra infer:
  - › `x:MyCar x:colour "red"^^xsd:string .`

### › In case we next extra assert:

- › `x:MyRedCar rdf:type x:RedCar .`
- › `y:MyCar owl:sameAs x:MyRedCar . *`
- › we can extra infer:
  - › `x:MyRedCar y:color "red"^^xsd:string .`
  - › `x:MyRedCar x:colour "red"^^xsd:string .`



# EXAMPLE: SITUATION 4: L2/L2 BOTH X AND Y GET SMARTER



- › Again, since `y:Car` has no restriction wrt the relevant property `y:color` this class is already L2 wrt this property
- › I.e. ontology `y` cannot be made 'smarter' wrt this property
- › So the situation is the same as Situation 3
- › Same inferences as in Situation 3

- › L1 level classes are most likely for existing/practical specifications
- › When in control we can have the ambition/manage to make them L2 or even L3
- › On individual-level we can use owl:sameAs to compensate missing ‘L2-ness’
- › Class-level L2 gives more inferences than L1 in case the property has constrained values for that class (i.e. reflects a ‘necessary condition’)
- › We can only exploit ‘L2-ness’ if the relevant properties are linked too (preferable via property equivalence, like in the example)
- › We can define L2 variants of L1 classes in the Linking Ontology and make them equivalent to the originating L1 variants. This way we do not change the original ontology (typically under foreign authority) but provide our own interpretation/‘more precise variant’ of it.

# EXAMPLE OF ADDING KNOWLEDGE TO THE LO



- › Ontology x asserts:  
x:RedCar rdf:type owl:Class .  
x:colour rdf:type owl:DatatypeProperty ;  
rdfs:range xsd:string .
  
- › Ontology y asserts:  
y:Car rdf:type owl:Class .  
y:color rdf:type owl:DatatypeProperty ;  
rdfs:range xsd:string .
  
- › LO assert:  
I:RedCar rdf:type owl:Class ;  
I:RedCar owl:EquivalentClass x:RedCar ;  
rdfs:subClassOf [ a owl:Restriction ;  
owl:hasValue "red"^^xsd:string ;  
owl:onProperty x:colour  
] .  
I:RedCar rdfs:subClassOf y:Car .  
x:colour owl:equivalentProperty y:color .

# MORE BACKGROUND INFO ON L1/L2/L3 LEVELS /1



- › **L1:** The class is just declared
- › **L2:** The class is declared and restrictions are added that indicate “necessary conditions” for one or more properties (can be a specific value or a range or complex combination, etc.)
  - › Example (having 2 such conditions):

```
RoadSegment rdf:type owl:Class ;  
  rdfs:subClassOf [ a owl:Restriction ;  
                    owl:hasValue "transport"^^xsd:string ;  
                    owl:onProperty :functionPerformed  
                  ] .  
  rdfs:subClassOf [ a owl:Restriction ;  
                    owl:hasValue "vehicles"^^xsd:string ;  
                    owl:onProperty :applicationArea  
                  ] .
```
- › **L3:** Like L2 but now so many restrictions that this set of restrictions is sufficient to fully define the class. In other words: the class and the set of combined restrictions are equivalent. So when you encounter an individual in the real world and you can see/know that it has all the values for those properties (in general: ‘it satisfies all restrictions’) it is a member of that set defined by the restrictions and hence, because of the equivalence, a member of the class. In short: automatic classification becomes possible, you don’t have to assert anymore that `MyRoadSegment rdf:type :RoadSegment`.

- › In practice we see the need for more complex constraints/rules
- › In general: all of the those supported by the OWL-RL variant, involving
  - › Intersections, unions
  - › Restriction classes (cardinalities, hasValue, all/someValuesFrom etc.)
- › It seems handy to distinguish two Kind of LO
  - › Alignment Ontologies (AOs), and
  - › Conversion Rule Sets (CRSs), involving OWL-RL and beyond (OWL-DL, SHACL...)
    - › Even involving ‘calculations’ (via SPARQL or even coding etc.)
- › CRSs might import AO (as basic CRS)
  - › Some Tools (like TBC) can convert OWL-RL based AO to SHACL

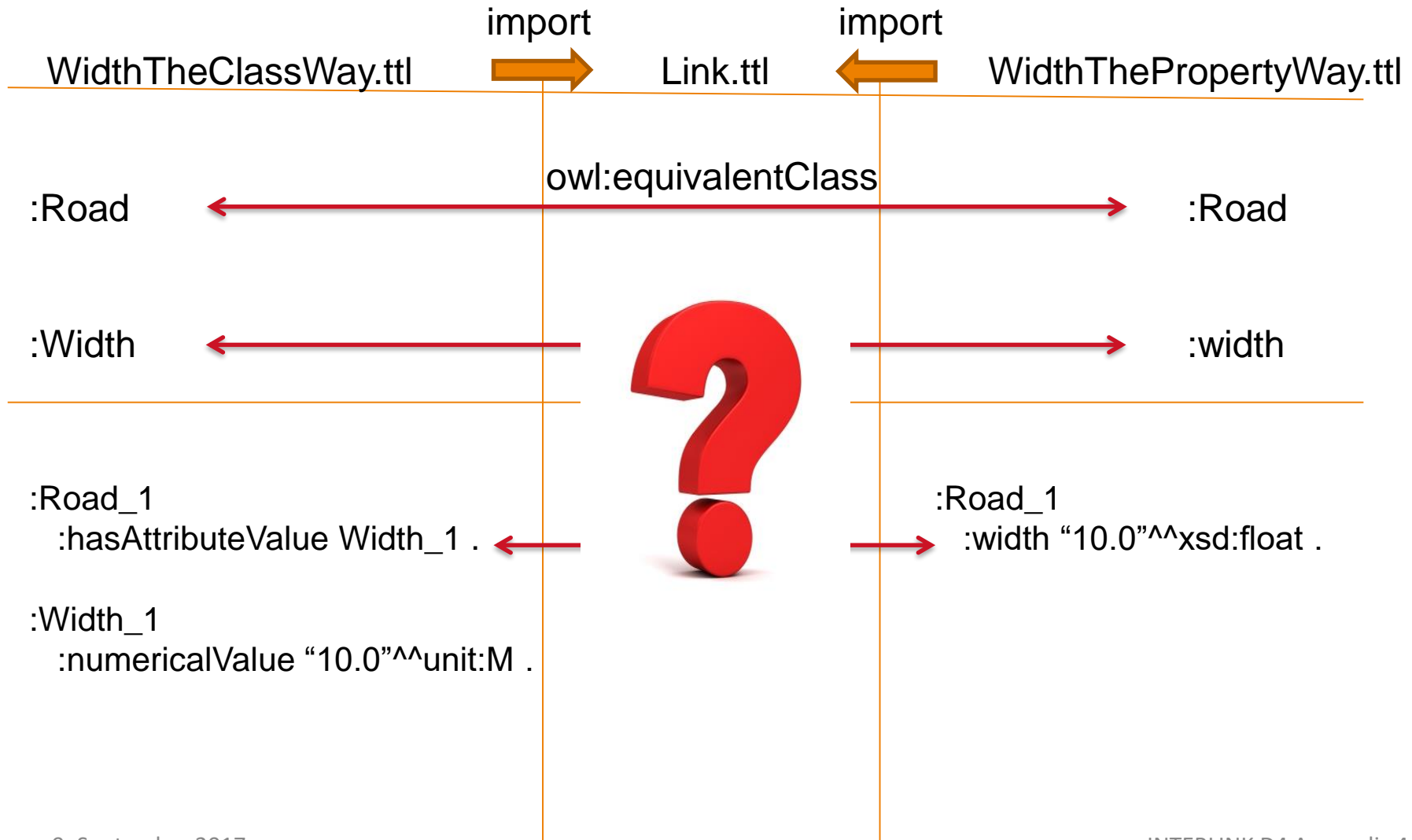
# SPECIAL ISSUE: CLASSES <> PROPERTIES <> INDIVIDUALS



- › Depending on the 'Modelling Style', attributes and relationships can be modelled in different ways: as classes (COINS, CB-NL, INTERLINK Powerful Modelling Style, ...) or as properties (INTERLINK Simple Modelling Style, ...)
- › Since we also want to interrelate/map such ontologies (for conversion and/or linking data) the issue arises of mapping "classes" to/from "properties"
- › A small example is depicted in the next sheet



# Special Issue: Example: Classes <> Properties



› **THANK YOU FOR YOUR ATTENTION**

