28/11/2022

SHACL

Shapes Constraint Language

Dr Eleni Tsalapati

Marie Curie Fellow

Why SHACL

- Ontologies & KGs have gained attention also in more "closed" domains than Semantic Web
  - Manufacturing
  - Enterprises
  - Banking sector
  - etc

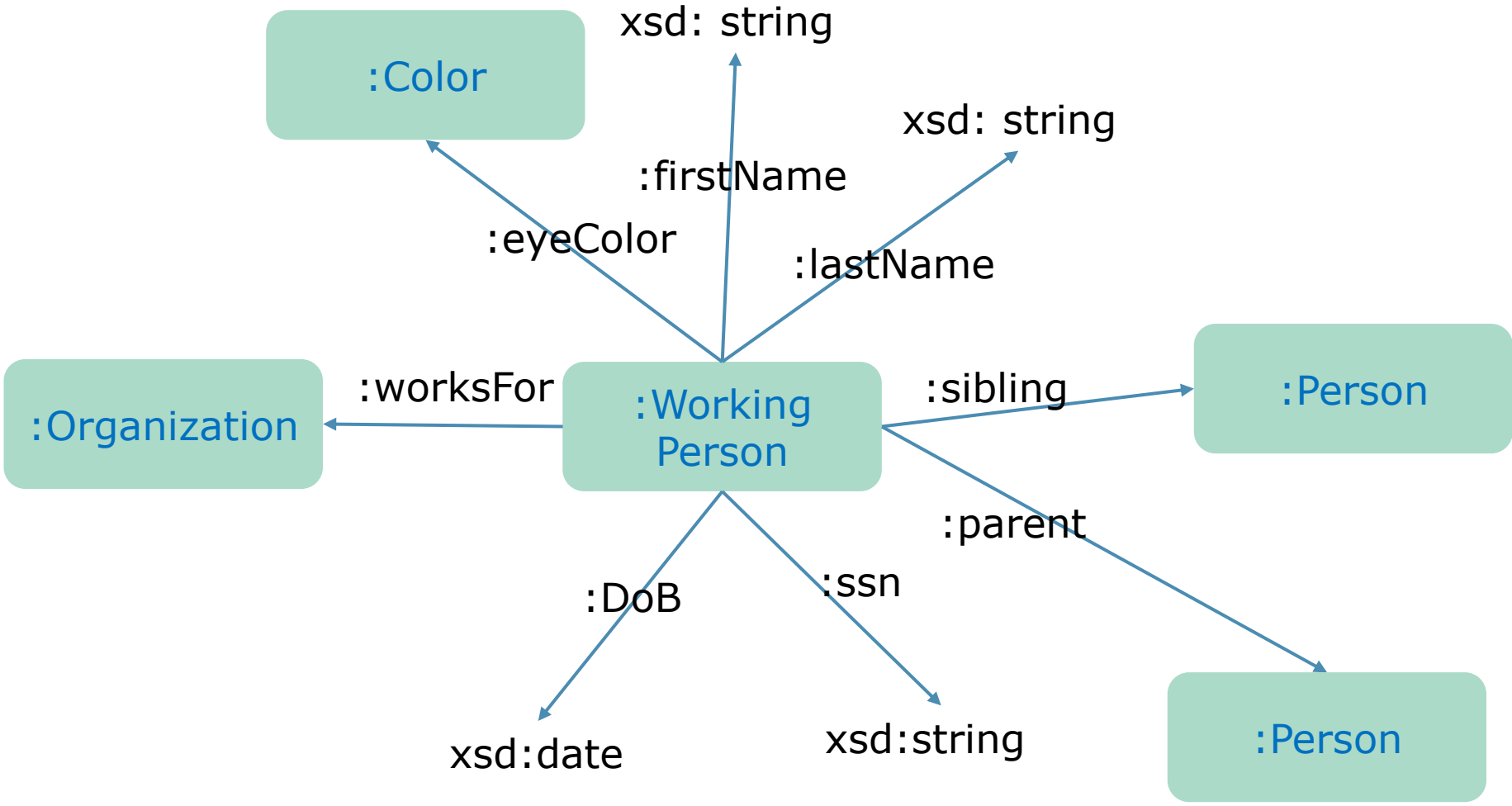
Why SHACL

- For these cases OWA is not always a good idea:
  - Manufacturing:
    - e.g., the deployment of a system
    - must be ensured that nothing is left out
  - Banking & Enterprise:
    - Different clients have different rights & benefits
- These worlds must be “closed”
- We need something more restrictive

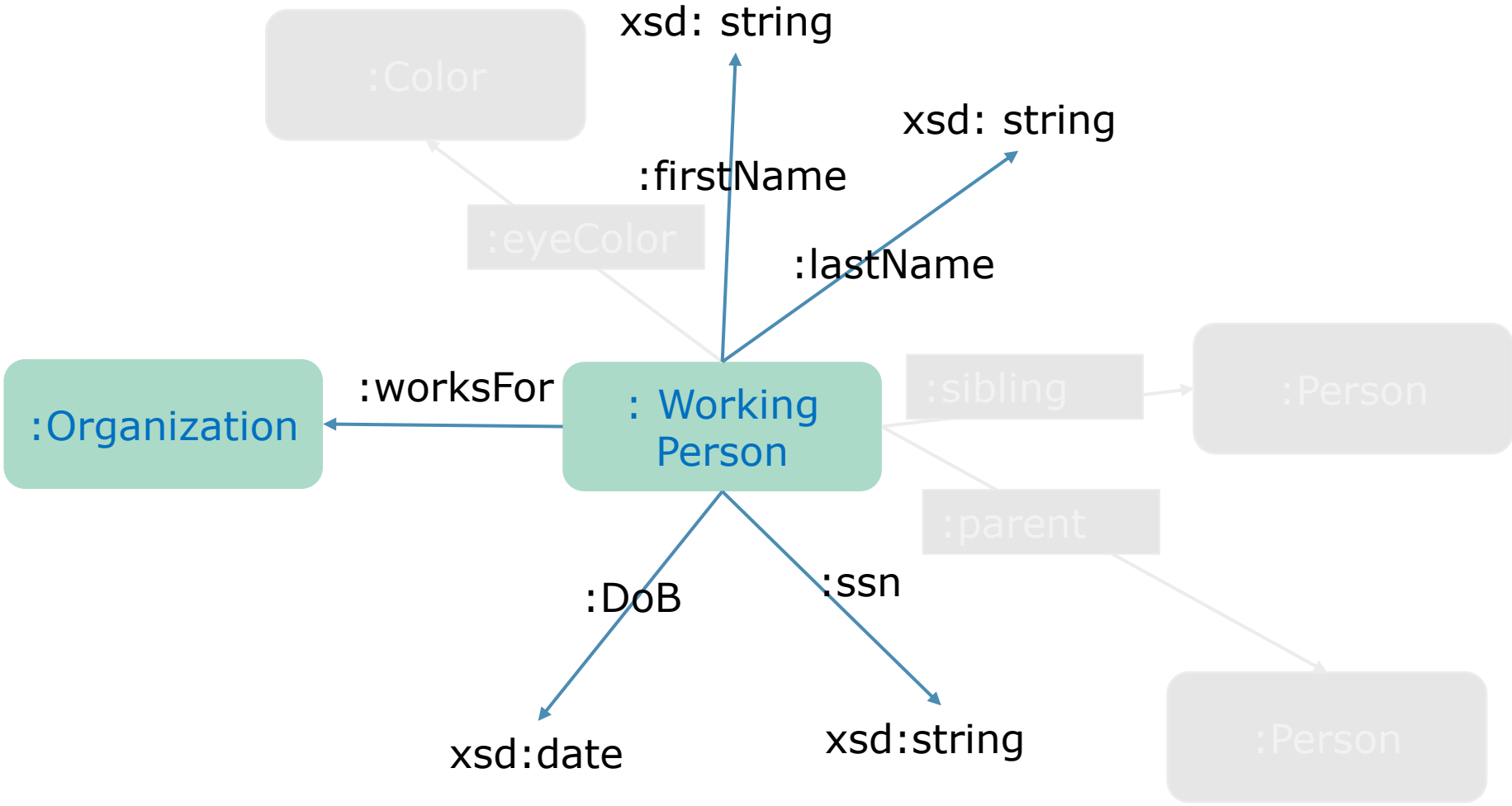
SHACL-Shapes Constraint Language

- An RDF-based language for **describing** and **validating** RDF graphs
- **Standardized** as a W3C Recommendation in July 2017
- Enables the ontology **reuse** through different **shapes** over the data

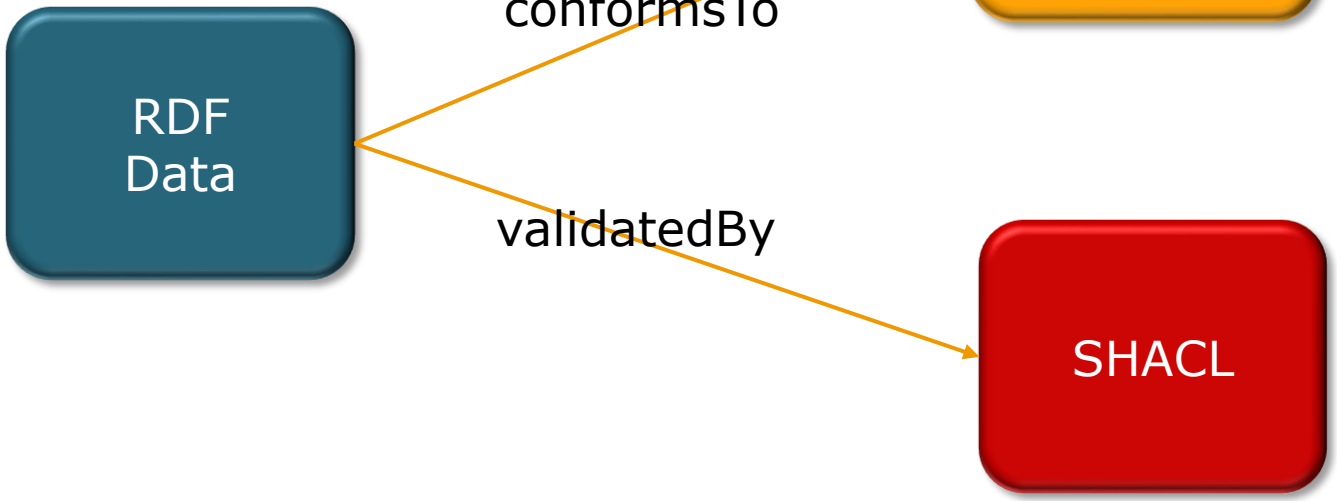
SHACL – Ontology Reuse



SHACL – Ontology Reuse



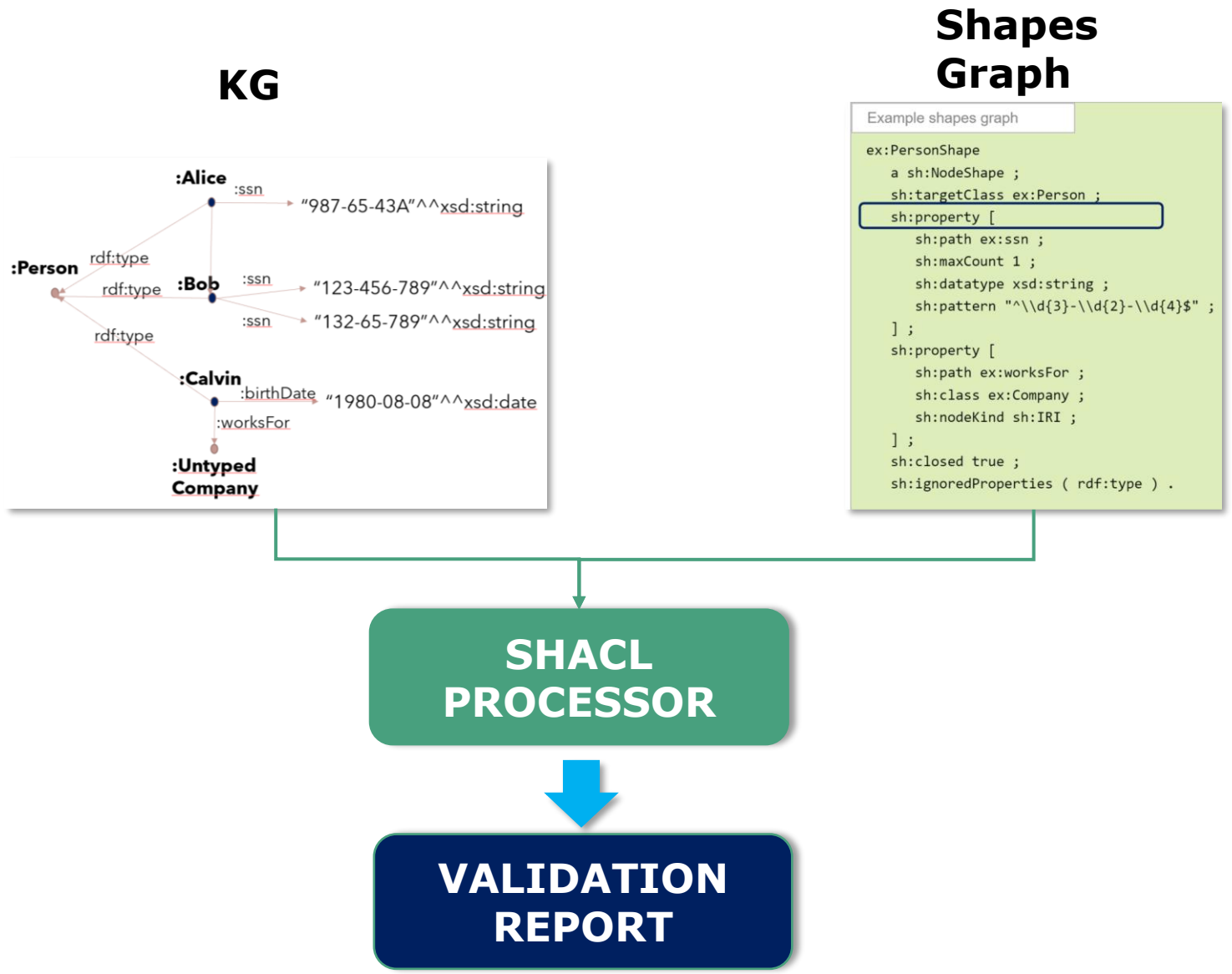
KG – OWL -SHACL



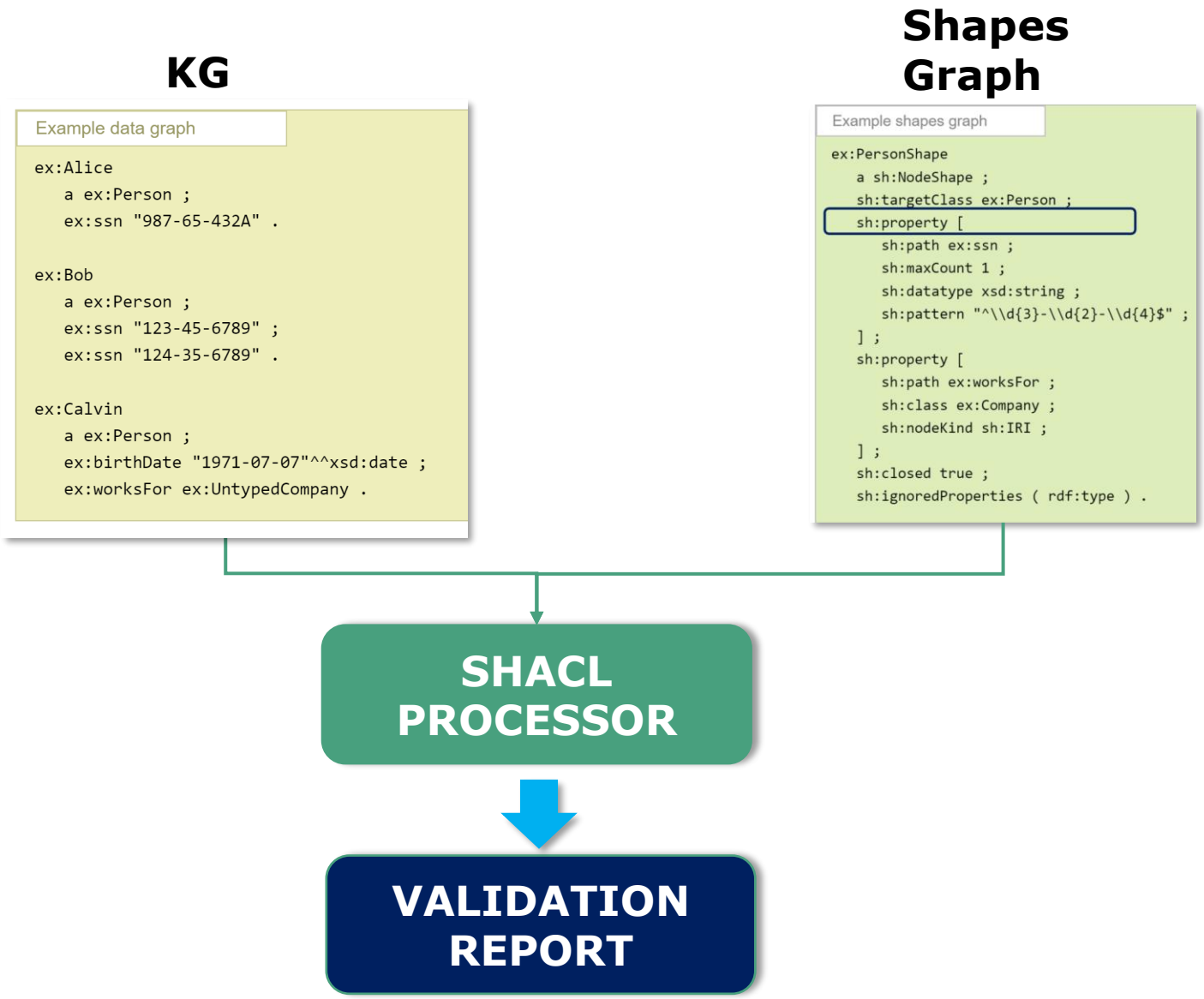
data integration
data interoperation
reuse
implied knowledge

different views of data
reuse these views
validate them

SHACL Processing

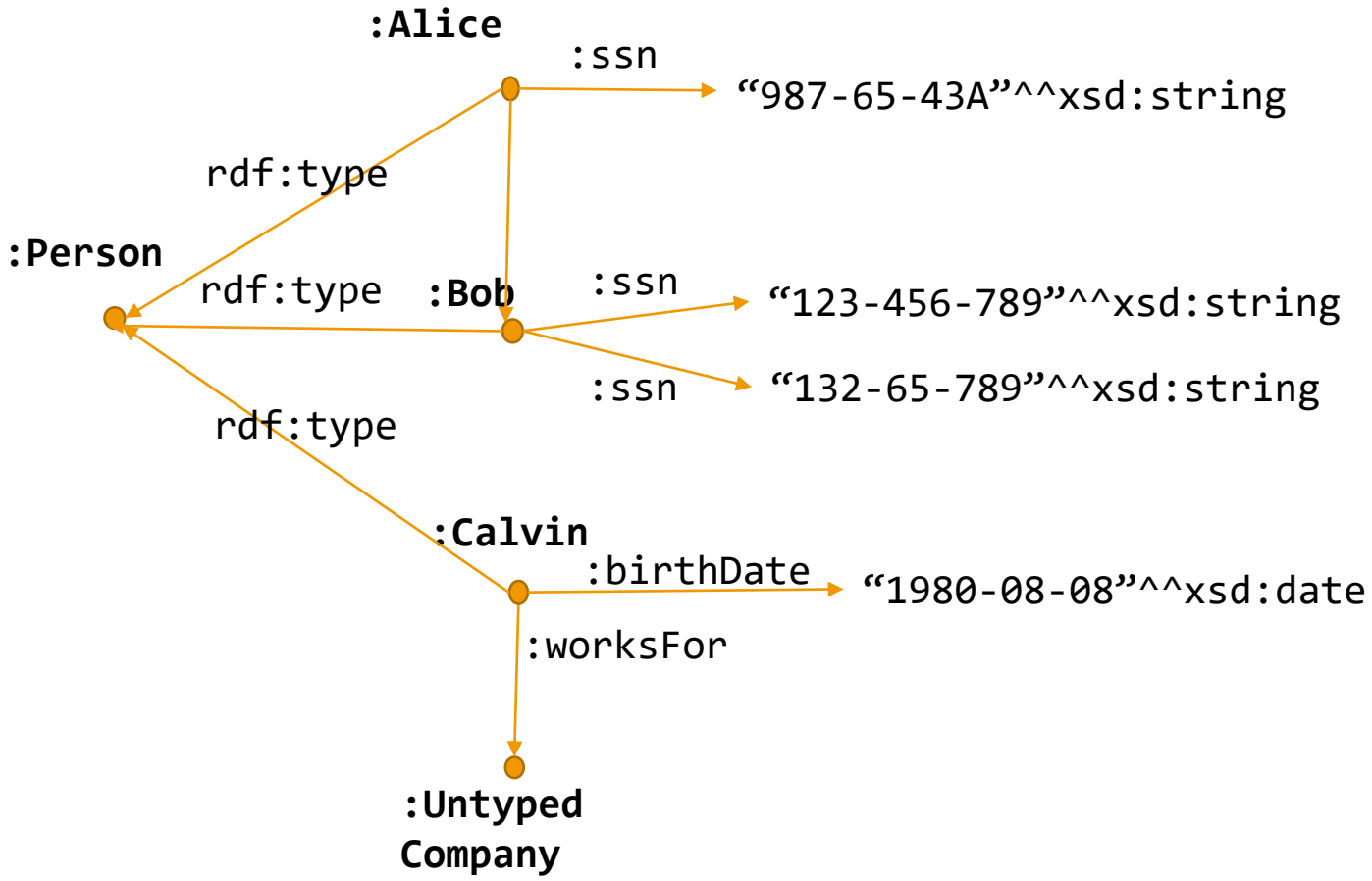


SHACL Processing



Example

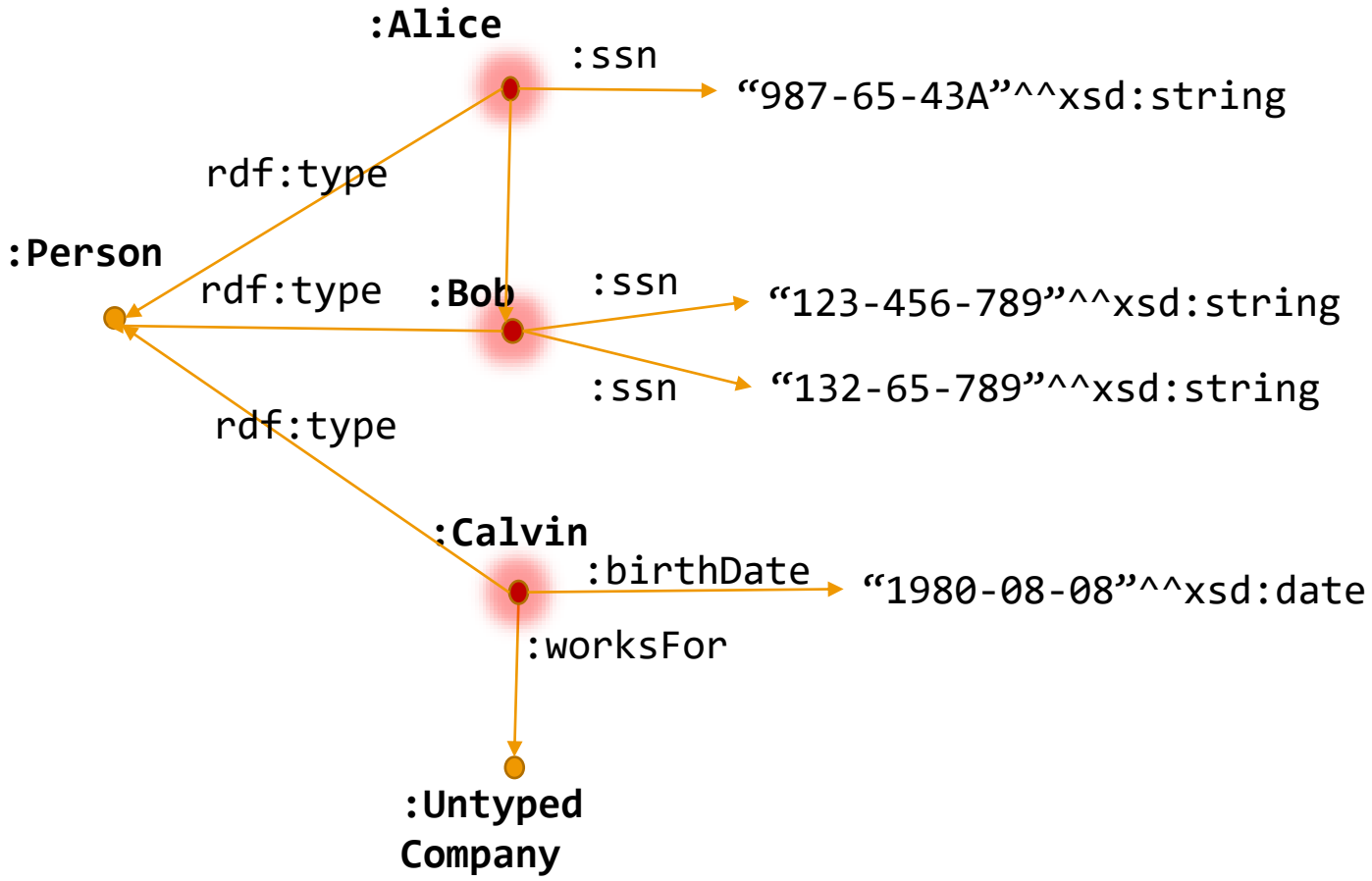
- **Shapes graph**: an RDF graph that contains shapes



Example shapes graph

```
ex:PersonShape
    a sh:NodeShape ;
    sh:targetClass ex:Person ;
    sh:property [
        sh:path ex:ssn ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
        sh:pattern "^\\d{3}-\\d{2}-\\d{4}$" ;
    ] ;
    sh:property [
        sh:path ex:worksFor ;
        sh:class ex:Company ;
        sh:nodeKind sh:IRI ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .
```

Example –Focus nodes

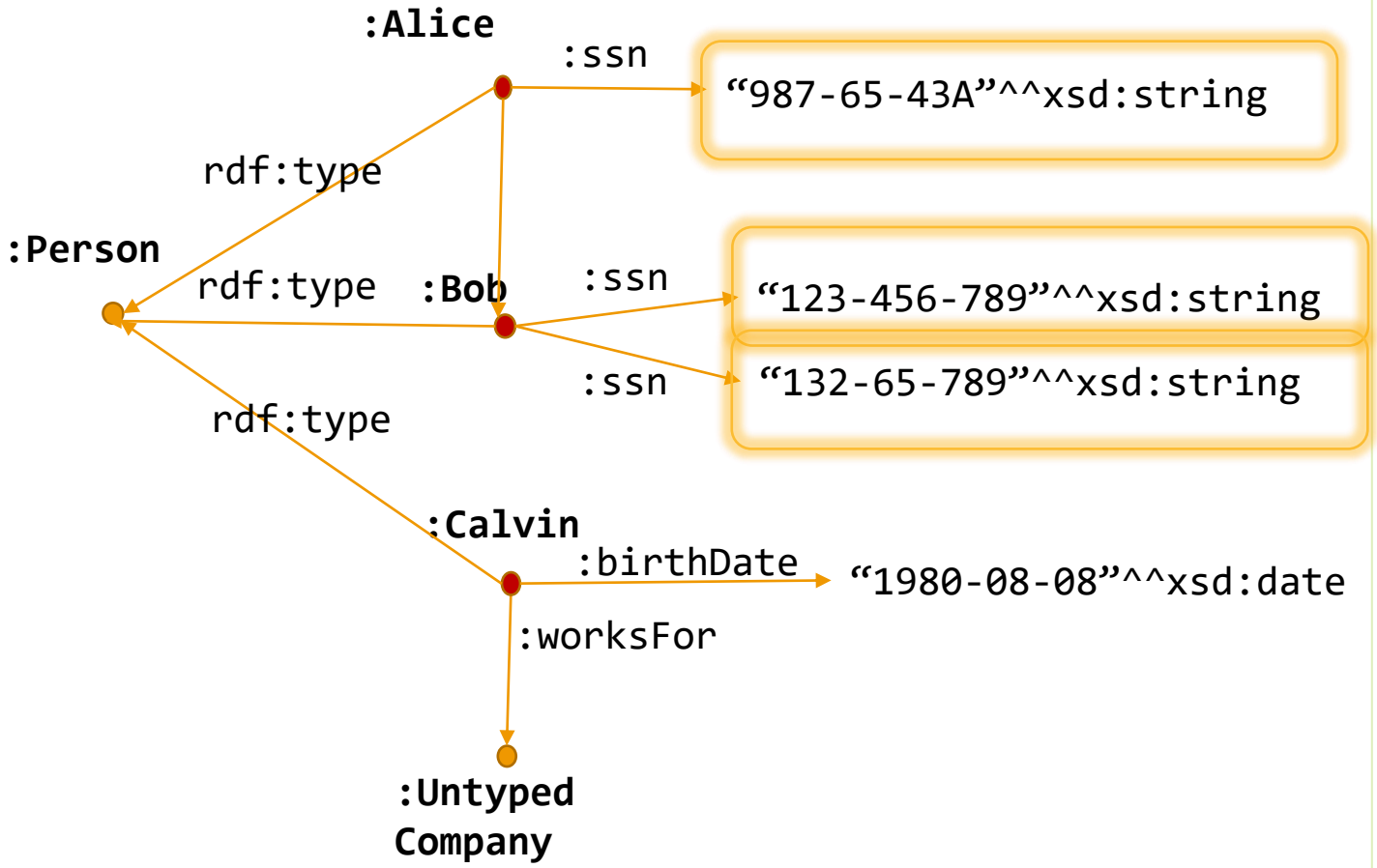


Example shapes graph

```
ex:PersonShape
    a sh:NodeShape ;
    sh:targetClass ex:Person ;
    sh:property [
        sh:path ex:ssn ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
        sh:pattern "^\\d{3}-\\d{2}-\\d{4}$" ;
    ] ;
    sh:property [
        sh:path ex:worksFor ;
        sh:class ex:Company ;
        sh:nodeKind sh:IRI ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .
```

During the validation, the instances of the target class (ex:Person) become the **focus nodes** of the shape

Example –Focus nodes

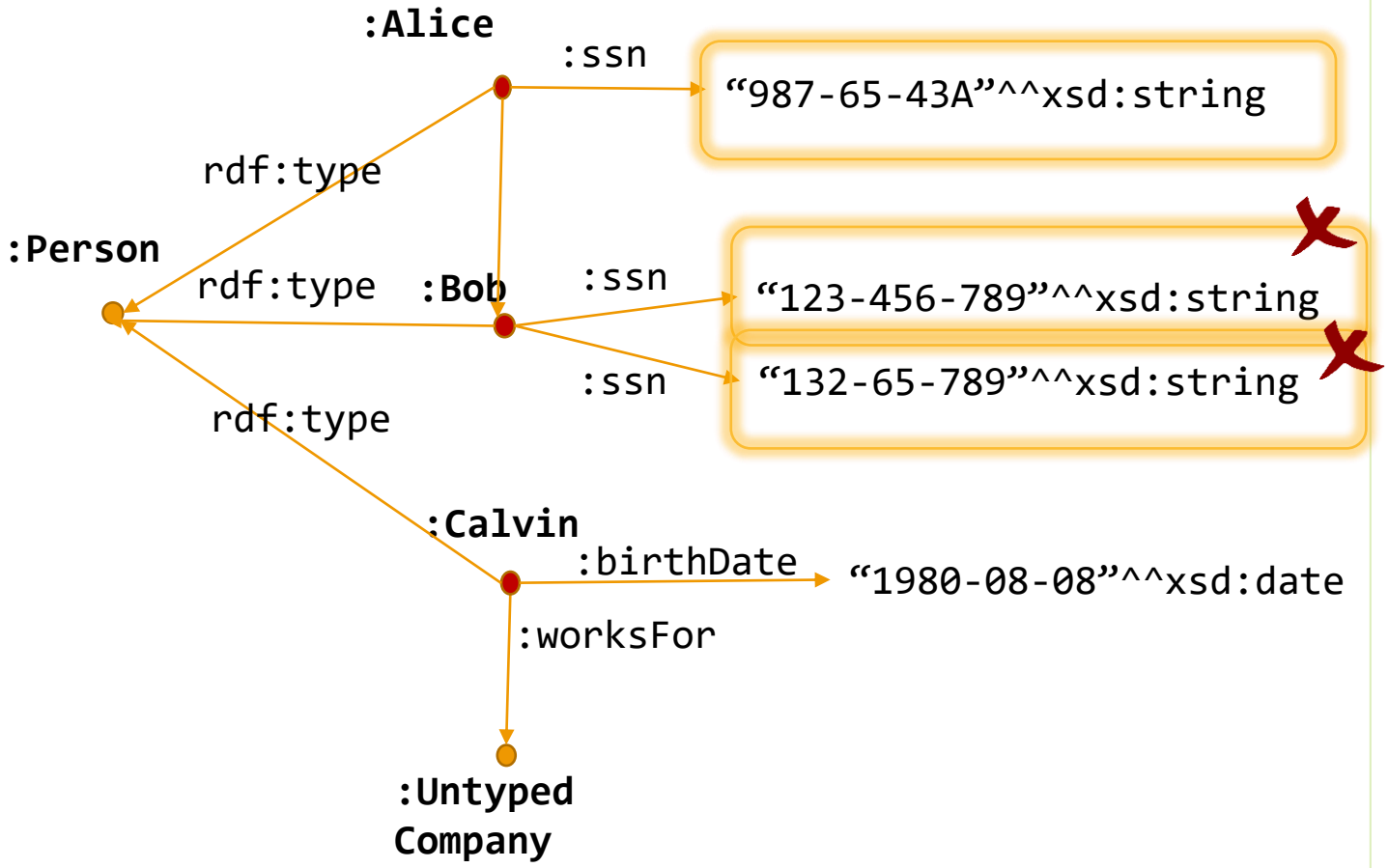


Example shapes graph

```
ex:PersonShape
    a sh:NodeShape ;
    sh:targetClass ex:Person ;
    sh:property [
        sh:path ex:ssn ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
        sh:pattern "^\\d{3}-\\d{2}-\\d{4}$" ;
    ] ;
    sh:property [
        sh:path ex:worksFor ;
        sh:class ex:Company ;
        sh:nodeKind sh:IRI ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .
```

Property shape: constraints on the set of nodes that can be reached from the focus node through the sh:path value (i.e., the value nodes)

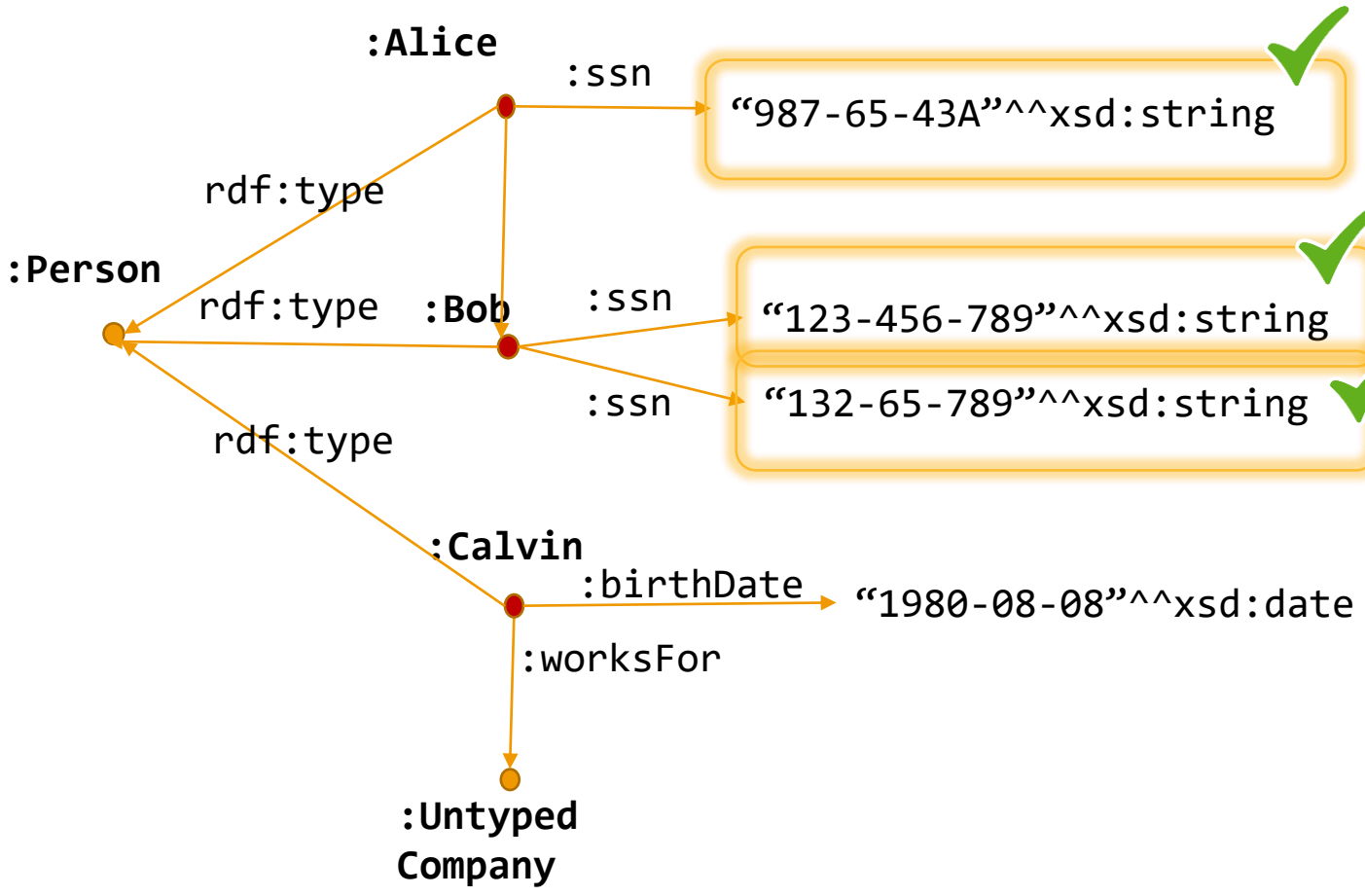
Example –Focus nodes



Example shapes graph

```
ex:PersonShape
    a sh:NodeShape ;
    sh:targetClass ex:Person ;
    sh:property [
        sh:path ex:ssn ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
        sh:pattern "^\\d{3}-\\d{2}-\\d{4}$" ;
    ] ;
    sh:property [
        sh:path ex:worksFor ;
        sh:class ex:Company ;
        sh:nodeKind sh:IRI ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .
```

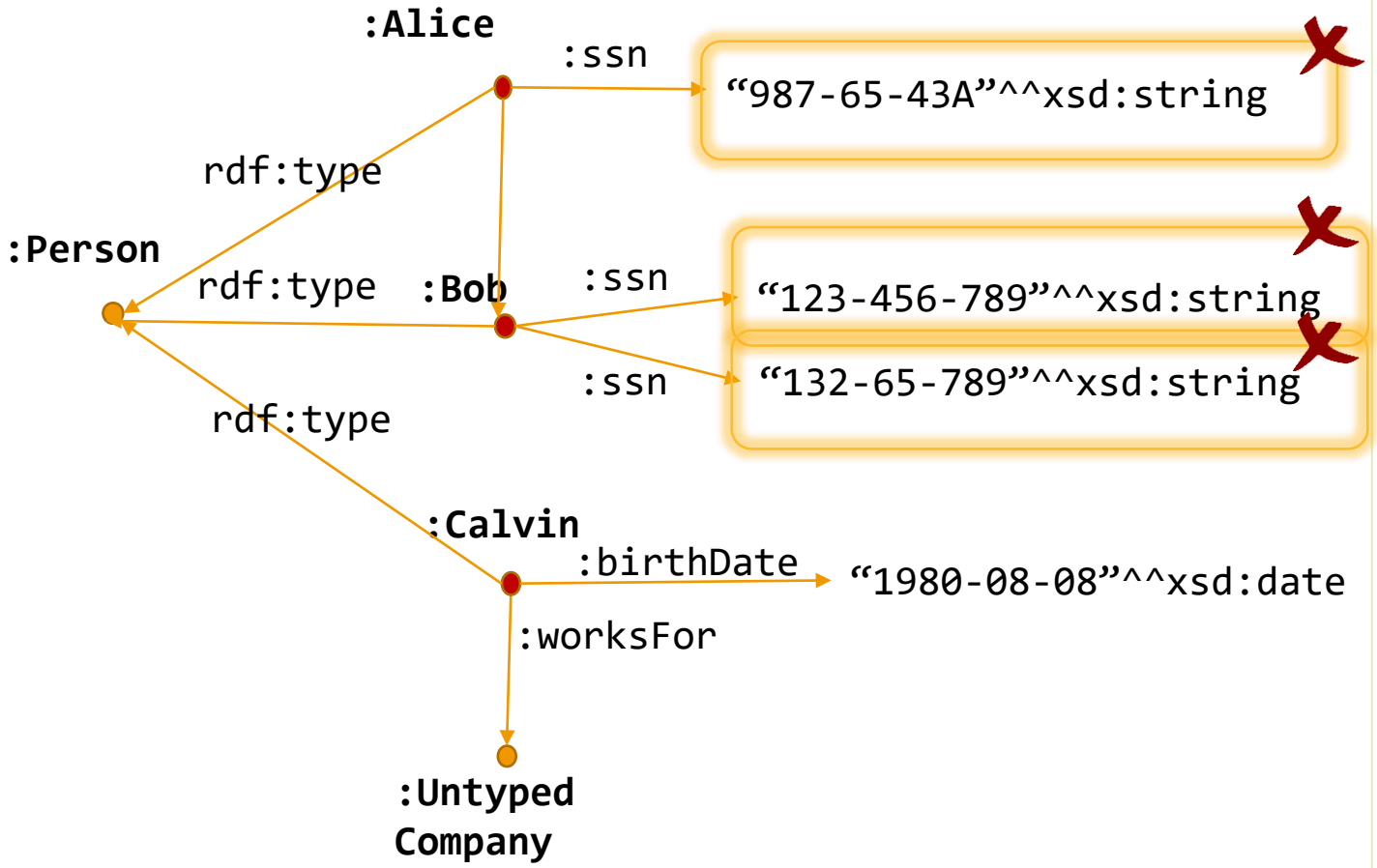
Example –Focus nodes



Example shapes graph

```
ex:PersonShape
    a sh:NodeShape ;
    sh:targetClass ex:Person ;
    sh:property [
        sh:path ex:ssn ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
        sh:pattern "^\\d{3}-\\d{2}-\\d{4}$" ;
    ] ;
    sh:property [
        sh:path ex:worksFor ;
        sh:class ex:Company ;
        sh:nodeKind sh:IRI ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .
```

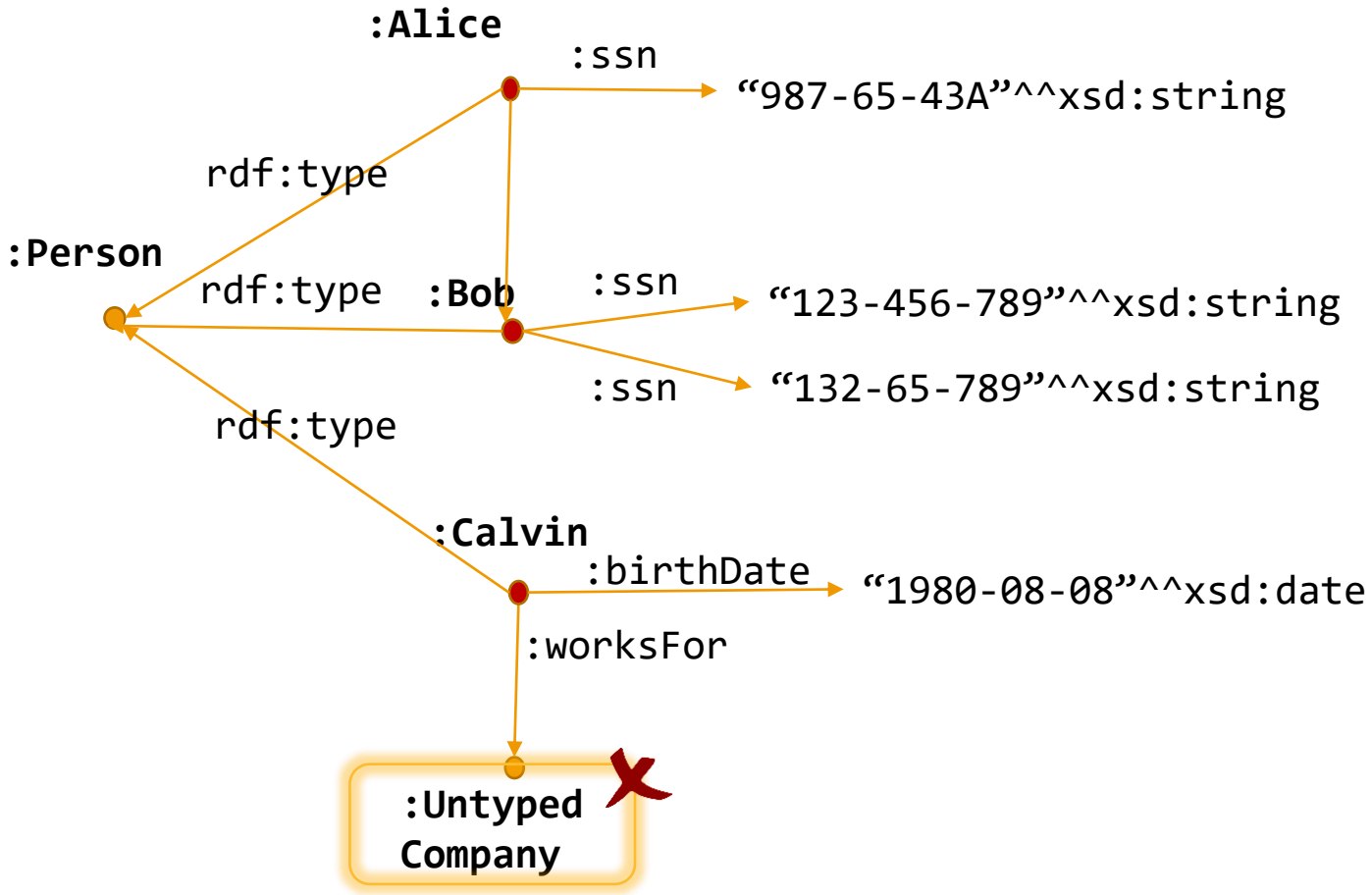
Example –Focus nodes



Example shapes graph

```
ex:PersonShape
    a sh:NodeShape ;
    sh:targetClass ex:Person ;
    sh:property [
        sh:path ex:ssn ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
        sh:pattern "^\\d{3}-\\d{2}-\\d{4}$" ;
    ] ;
    sh:property [
        sh:path ex:worksFor ;
        sh:class ex:Company ;
        sh:nodeKind sh:IRI ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .
```

Example –Focus nodes

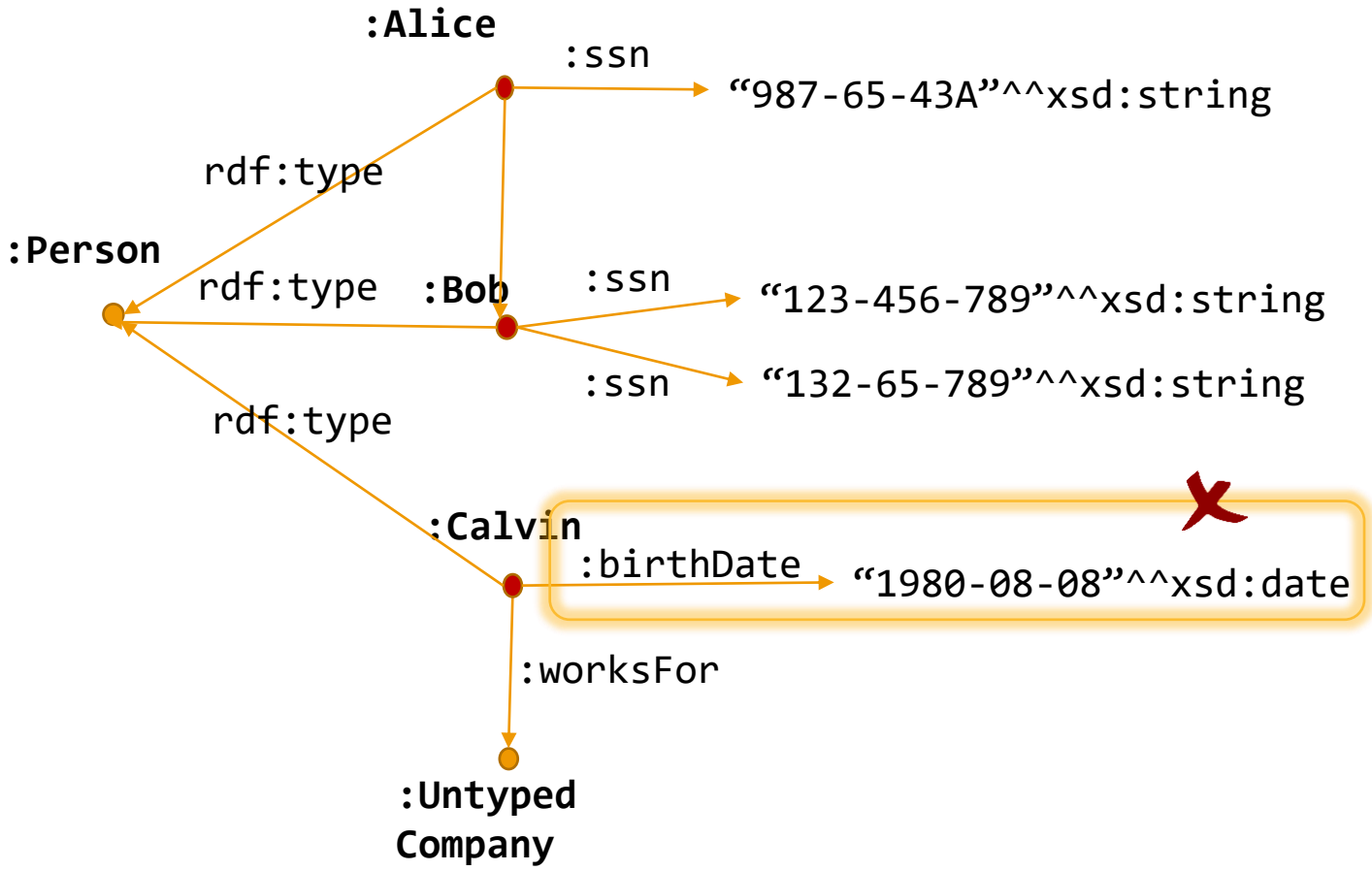


Example shapes graph

```
ex:PersonShape
    a sh:NodeShape ;
    sh:targetClass ex:Person ;
    sh:property [
        sh:path ex:ssn ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
        sh:pattern "^\\d{3}-\\d{2}-\\d{4}$" ;
    ] ;
    sh:property [
        sh:path ex:worksFor ;
        sh:class ex:Company ;
        sh:nodeKind sh:IRI ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .
```

The “rdf:type sh:Company” for ex:UntypedCompany is missing

Example –Focus nodes

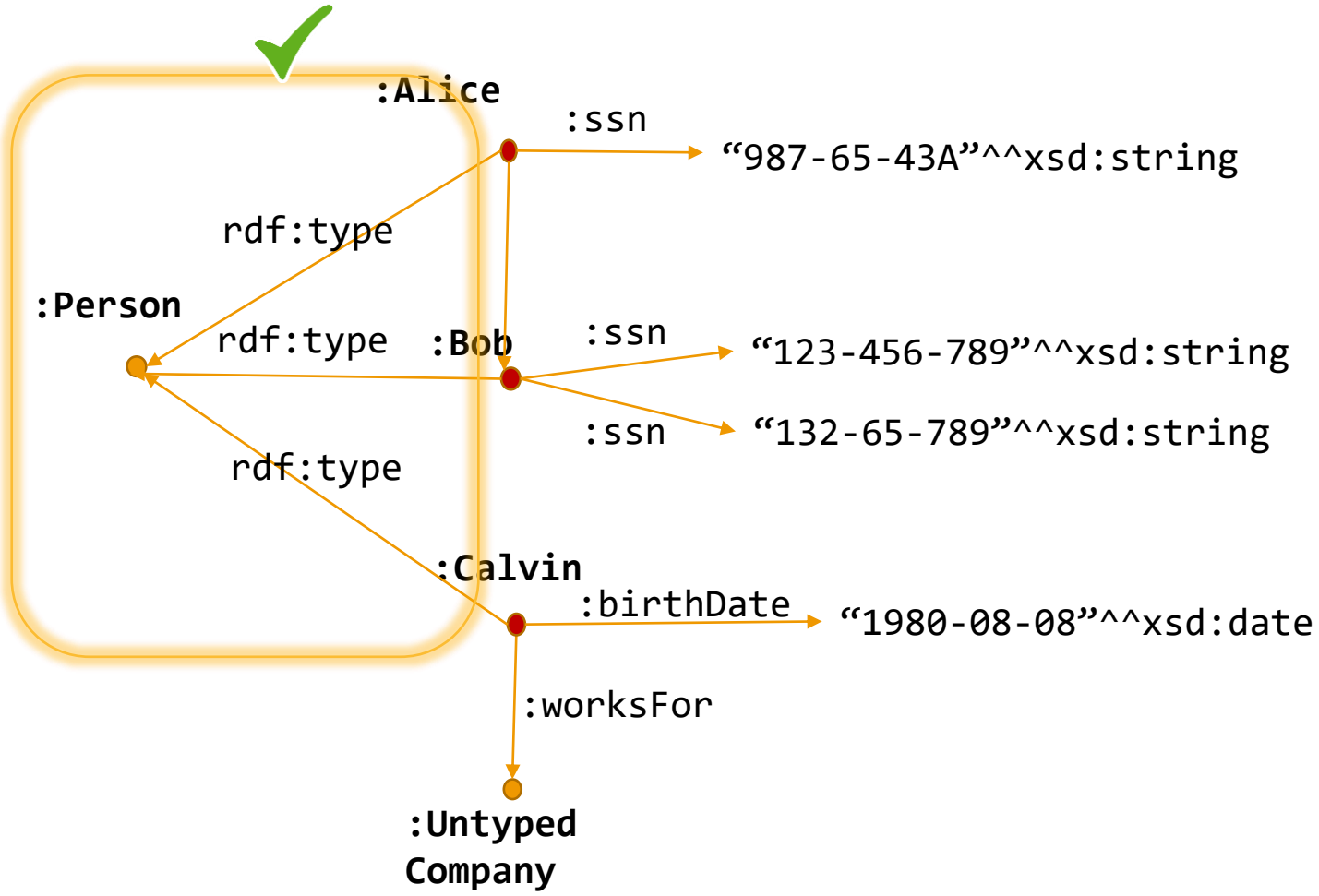


Example shapes graph

```
ex:PersonShape
    a sh:NodeShape ;
    sh:targetClass ex:Person ;
    sh:property [
        sh:path ex:ssn ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
        sh:pattern "^\\d{3}-\\d{2}-\\d{4}$" ;
    ] ;
    sh:property [
        sh:path ex:worksFor ;
        sh:class ex:Company ;
        sh:nodeKind sh:IRI ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .
```

Each focus node has values **only** for those properties that have been **explicitly enumerated via the property shapes** specified for the shape via sh:property.

Example –Focus nodes



Example shapes graph

```
ex:PersonShape
    a sh:NodeShape ;
    sh:targetClass ex:Person ;
    sh:property [
        sh:path ex:ssn ;
        sh:maxCount 1 ;
        sh:datatype xsd:string ;
        sh:pattern "^\\d{3}-\\d{2}-\\d{4}$" ;
    ] ;
    sh:property [
        sh:path ex:worksFor ;
        sh:class ex:Company ;
        sh:nodeKind sh:IRI ;
    ] ;
    sh:closed true ;
    sh:ignoredProperties ( rdf:type ) .
```

…**except for** the properties in sh:ignoredProperties, i.e. except for rdf:type!

Example validation results

```
[  a sh:ValidationReport ;
   sh:conforms false ;
   sh:result
   [  a sh:ValidationResult ;
      sh:resultSeverity sh:Violation ;
      sh:focusNode ex:Alice ;
      sh:resultPath ex:ssn ;
      sh:value "987-65-432A" ;
      sh:sourceConstraintComponent sh:RegexConstraintComponent ;
      sh:sourceShape ... blank node _:b1 on ex:ssn above ... ;
   ] ,
   [  a sh:ValidationResult ;
      sh:resultSeverity sh:Violation ;
      sh:focusNode ex:Bob ;
      sh:resultPath ex:ssn ;
      sh:sourceConstraintComponent sh:MaxCountConstraintComponent ;
      sh:sourceShape ... blank node _:b1 on ex:ssn above ... ;
   ] ,
   [  a sh:ValidationResult ;
      sh:resultSeverity sh:Violation ;
      sh:focusNode ex:Calvin ;
      sh:resultPath ex:worksFor ;
      sh:value ex:UntypedCompany ;
      sh:sourceConstraintComponent sh:ClassConstraintComponent ;
      sh:sourceShape ... blank node _:b2 on ex:worksFor above ... ;
   ] ,
   [  a sh:ValidationResult ;
      sh:resultSeverity sh:Violation ;
      sh:focusNode ex:Calvin ;
      sh:resultPath ex:birthDate ;
      sh:value "1971-07-07"^^xsd:date ;
      sh:sourceConstraintComponent sh:ClosedConstraintComponent ;
      sh:sourceShape sh:PersonShape ;
   ]
```

If sh:resultSeverity is not declared in the shape, then the default severity is sh:Violation

Different from ^\\d{3}-\\d{2}-\\d{4}$

More than one values for ex:ssn

The “rdf:type sh:Company” for ex:UntypedCompany is missing

The predicate ex:birthDate is different from the allowed predicates and the ignored predicates.

Shapes

- **Shape**: A conjunction of constraints that a **target** must satisfy. A target can be:
  - Class (sh:targetClass)
  - Node (sh:targetNode)
  - Objects of a property (sh:targetObje
  - Subjects of a property (sh:targetSub

All instances of the target class

```
:MyShape a sh:NodeShape;
    sh:targetClass :Person;
```

Shapes

- **Shape**: A conjunction of constraints that a **target** must satisfy. A target can be :
  - Class (sh:targetClass)
  - Node (sh:targetNode)
  - Objects of a property (sh:targetObje
  - Subjects of a property (sh:targetSub

Only to specific instances

```
:MyShape a sh:NodeShape;
    sh:targetNode :alice, :bob;
```

Shapes

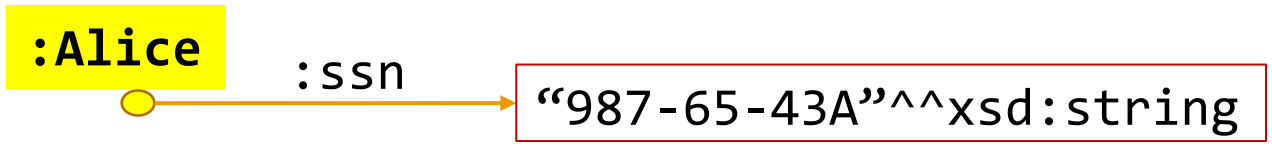
- **Shape**: A conjunction of constraints t
satisfy. A target can be :
  - Class (sh:targetClass)
  - Node (sh:targetNode)
  - Objects of a property (sh:targetObjectsOf)
  - Subjects of a property (sh:targetSubjectsOf)

```
:MyShape a sh:NodeShape;
    sh:targetObjectsOf :ssn;
```

Shapes

- **Shape**: A conjunction of constraints [illegible] satisfy. A target can be :
  - Class (sh:targetClass)
  - Node (sh:targetNode)
  - Objects of a property (sh:targetObjectsOf)
  - Subjects of a property (sh:targetSubjectsOf)

```
:MyShape a sh:NodeShape;
    sh: targetSubjectsOf :ssn;
```



Node Shape

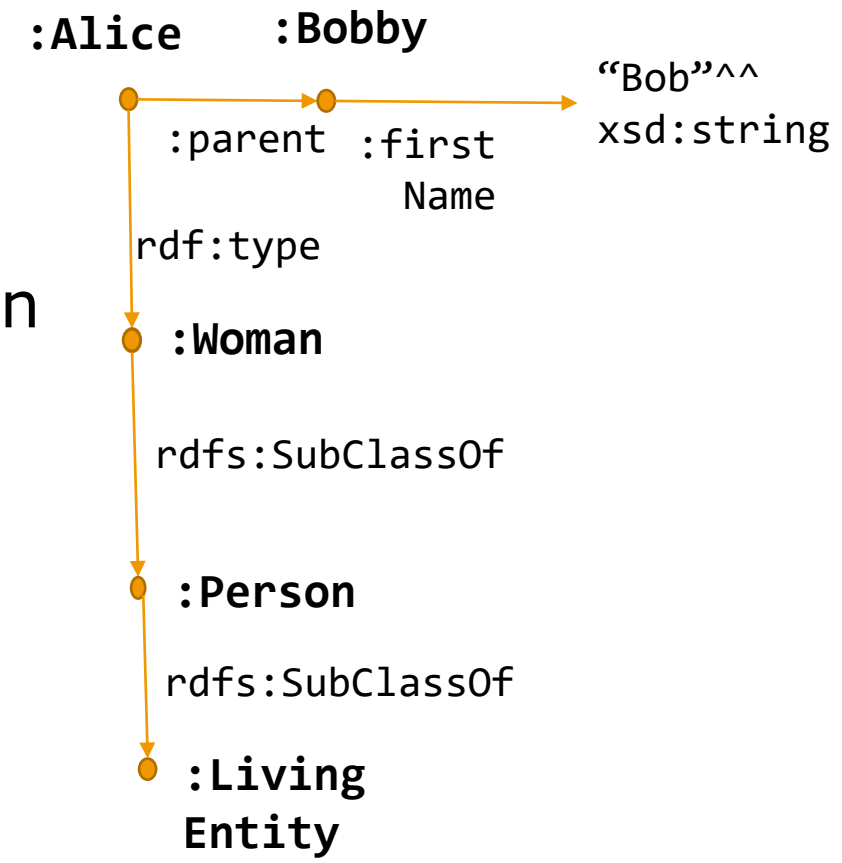
- **Node Shape**: constraints directly on a node (e.g. Customer must be an IRI)
  - e.g., `sh:not`, `sh:closed`, `sh:or`, `sh:property`

Example shapes graph

```
ex:OrConstraintExampleShape
   a sh:NodeShape ;
   sh:targetNode ex:Bob ;
   sh:or (
      [
         sh:path ex:firstName ;
         sh:minCount 1 ;
      ]
      [
         sh:path ex:givenName ;
         sh:minCount 1 ;
      ]
   ) .
```

Property Shape

- **Property Shape**: constraints on the property that is connected to the node via a path (e.g. every Customer *must* have an ID). The path can be:
  - A predicate (`ex:parent`)
  - Sequence of paths (`(ex:parent ex:firstName)`)
  - Inverse of a path (`[sh:inversePath ex: parent]`)
  - (`rdf:type [sh: zeroOrMorePath rdfs:SubClassOf]`)
  - `[sh:alternativePath (ex:father ex:mother)]`



Property shape's properties

- Constraint parameters:
  - sh:minCount, sh:maxCount, sh:class, sh:datatype,sh:disjoint, sh:equals, **sh:node**

The node shape that all value nodes need to conform to.

Example shapes graph

```
ex:AddressShape
   a sh:NodeShape ;
   sh:property [
      sh:path ex:postalCode ;
      sh:datatype xsd:string ;
      sh:maxCount 1 ;
   ] .

ex:PersonShape
   a sh:NodeShape ;
   sh:targetClass ex:Person ;
   sh:property [   # _:b1
      sh:path ex:address ;
      sh:minCount 1 ;
      sh:node ex:AddressShape ;
   ] .
```

Property shape’s properties

- Constraint parameters:
  - sh:minCount, sh:maxCount, sh:class, sh:datatype,sh:disjoint, sh:equals, sh:node, sh:qualifiedValueShape, sh:QualifiedMinCount, sh:QualifiedMaxCount,etc

The node shape that all value nodes need to conform to.

```
ex:QualifiedValueShapeExampleShape
   a sh:NodeShape ;
   sh:targetNode ex:QualifiedValueShapeExampleValidResource ;
   sh:property [
      sh:path ex:parent ;
      sh:minCount 2 ;
      sh:maxCount 2 ;
      sh:qualifiedValueShape [
         sh:path ex:gender ;
         sh:hasValue ex:female ;
      ] ;
      sh:qualifiedMinCount 1 ;
   ] .
```

Shapes’ properties

- We can exclude shapes that do not apply to a specific case (sh:deactivate)
- We can present messages to the users (sh:message)
- We can declare the severity of the violation (sh:Info, sh:Warning, sh:Violation)

Syntax OWL vs SHACL

```
ex:Person
   a owl:Class ;
   rdfs:subClassOf [
      a owl:Restriction ;
      owl:onProperty ex:hasFather ;
      owl:maxCardinality 1 ;
   ] ;
   rdfs:subClassOf [
      a owl:Restriction ;
      owl:onProperty ex:hasFather ;
      owl:allValuesFrom ex:Person ;
   ] .
```

```
ex:Person
   a owl:Class, sh:NodeShape ;
   sh:property [
      sh:path ex:hasFather ;
      sh:maxCount 1 ;
      sh:class ex:Person ;
   ] .
```

OR

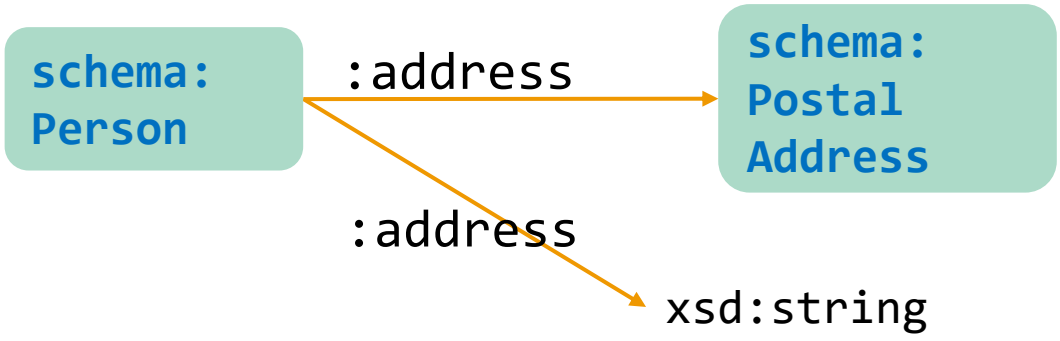
```
ex:Person
   a owl:Class .

ex:PersonShape
   a sh:NodeShape ;
   sh:targetClass ex:Person ;
   sh:property [
      sh:path ex:hasFather ;
      sh:maxCount 1 ;
      sh:class ex:Person ;
   ] .
```

We can reuse it now

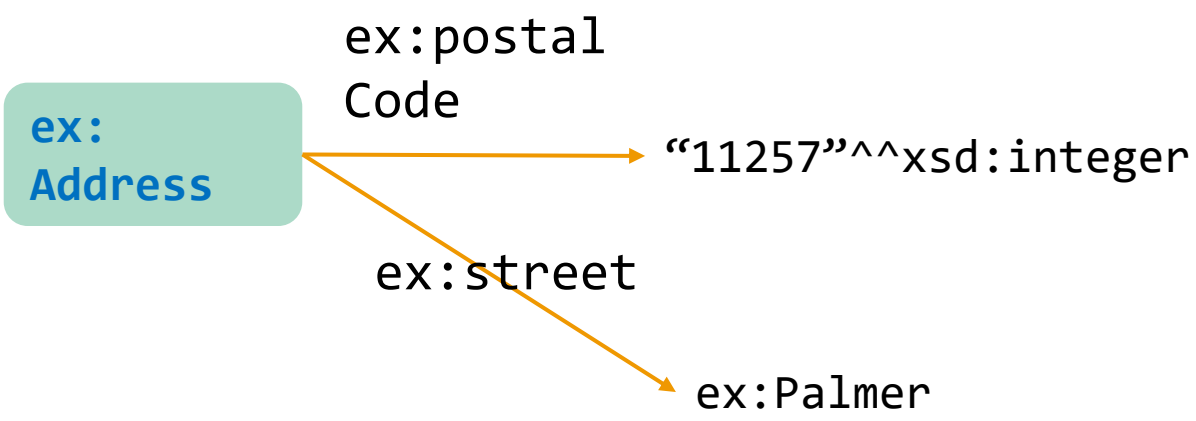
Syntax OWL vs SHACL

- Not always straightforward though




```
schema:Person
    a owl:Class, sh:NodeShape ;
    sh:property [
        sh:path schema:address ;
        sh:or (
            [ sh:class schema:PostalAddress ]
            [ sh:datatype xsd:string ]
        )
    ] .
```

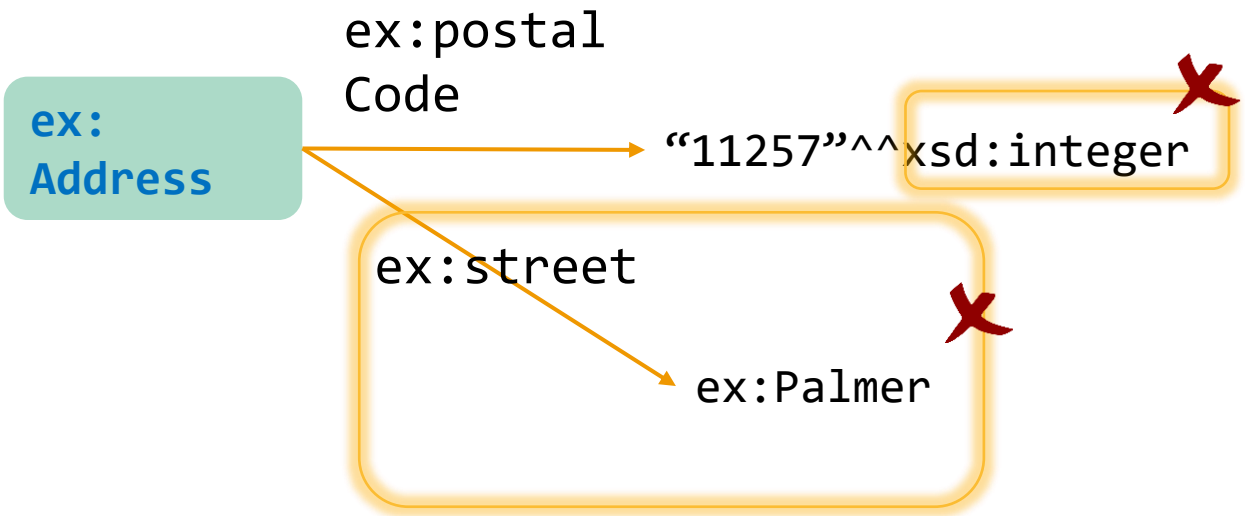
EXERCISE




```
sh:AddressShape  a    sh:NodeShape;
        sh:targetClass ex:Address [
        sh:property ex:postalCode;
        sh:maxCount 1;
        sh:datatype xsd:string;
        ]
        sh:closed true.
```

Indicate the violations (if any).

EXERCISE




```
sh:AddressShape  a    sh:NodeShape;
        sh:targetClass ex:Address [
        sh:property ex:postalCode;
        sh:maxCount 1;
        sh:datatype xsd:string;
        ];
        sh:closed true .
```

Resources

- https://www.w3.org/TR/shacl/
- “The many shapes of SHACL”: https://youtu.be/ccs-KhnWR1U (TopBraid)
- OWL vs SHACL: https://spinrdf.org/shacl-and-owl.html
- A nice example from the banking sector: https://youtu.be/apG5K3zc4V0?t=1151