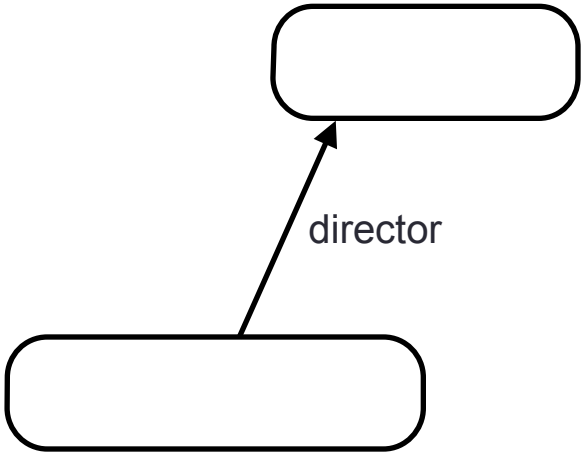
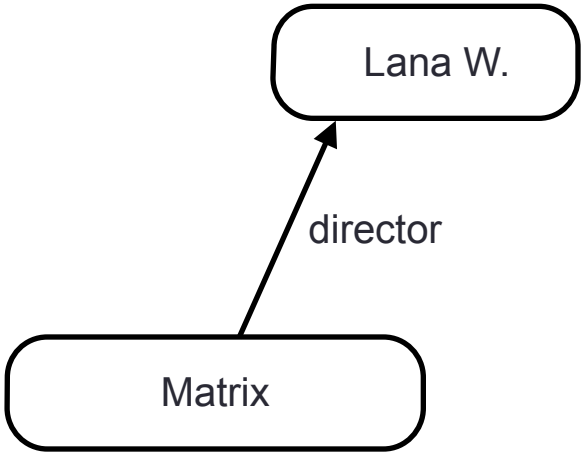
# Unstructured Data:
## advantages of going without schema

# Unstructured Data:
## advantages of going without schema

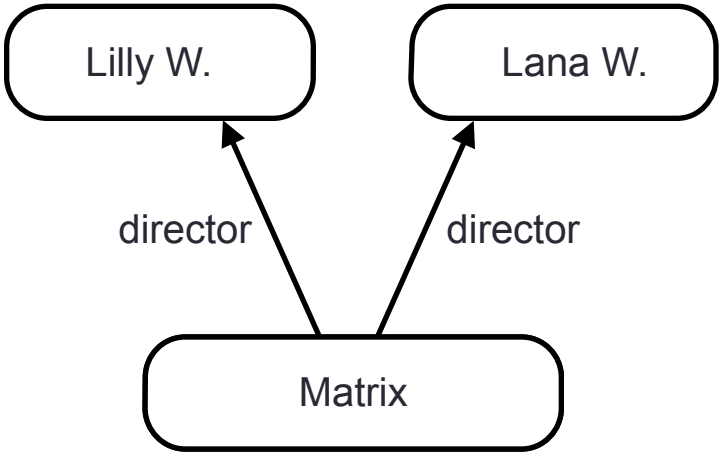
# Unstructured Data:
## advantages of going without schema

| *name* | *director* |
| --- | --- |
| ***Matrix*** | ***Lana Wachowski*** |

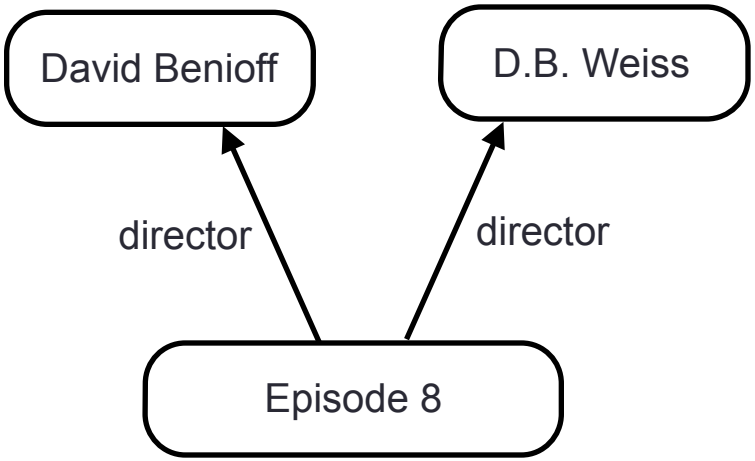


# Unstructured Data:
## advantages of going without schema

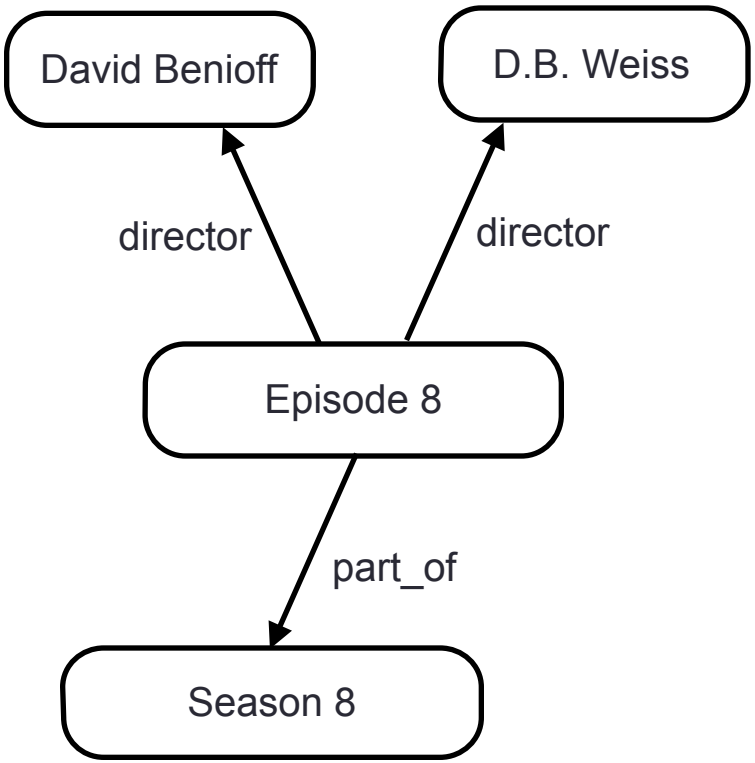
| *name* | *director* |
|---|---|
| ***Matrix*** | ***Lana Wachowski*** |
| ***Matrix*** | ***Lilly Wachowski*** |



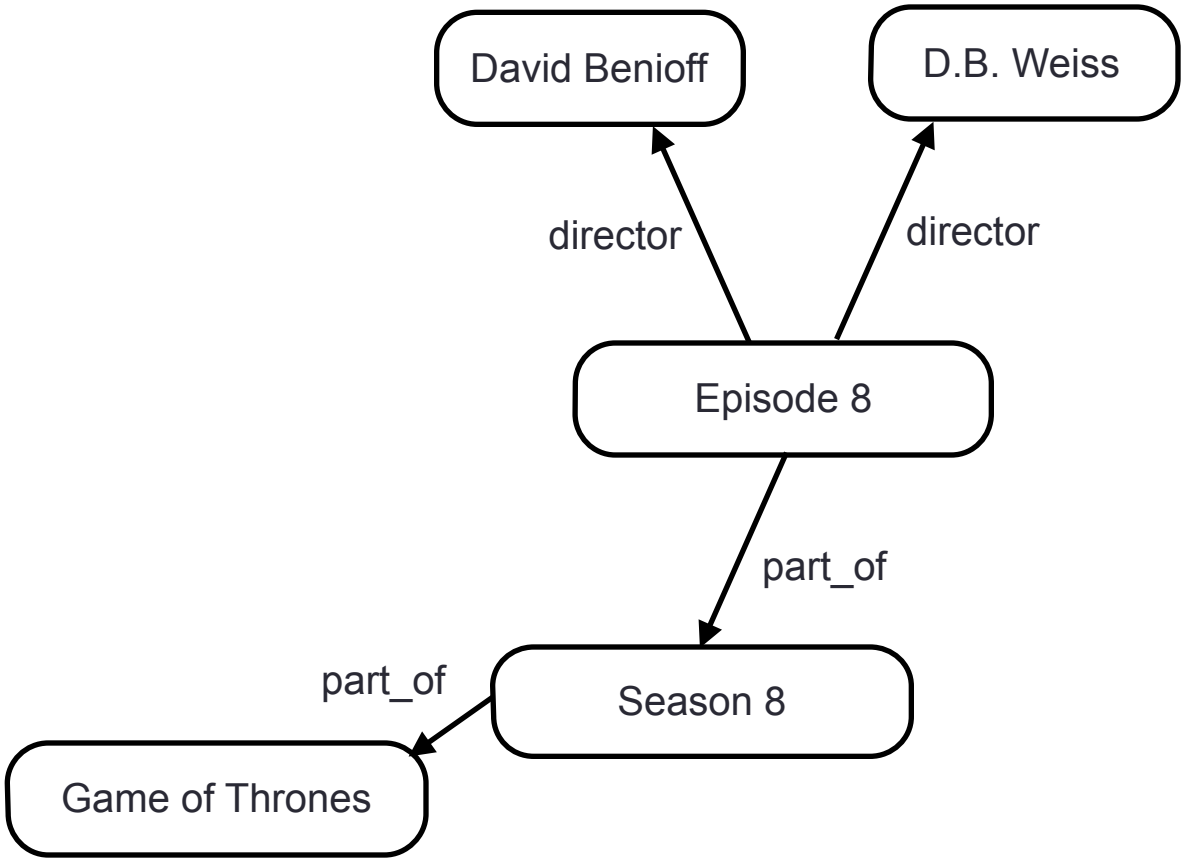
# Unstructured Data:
## advantages of going without schema



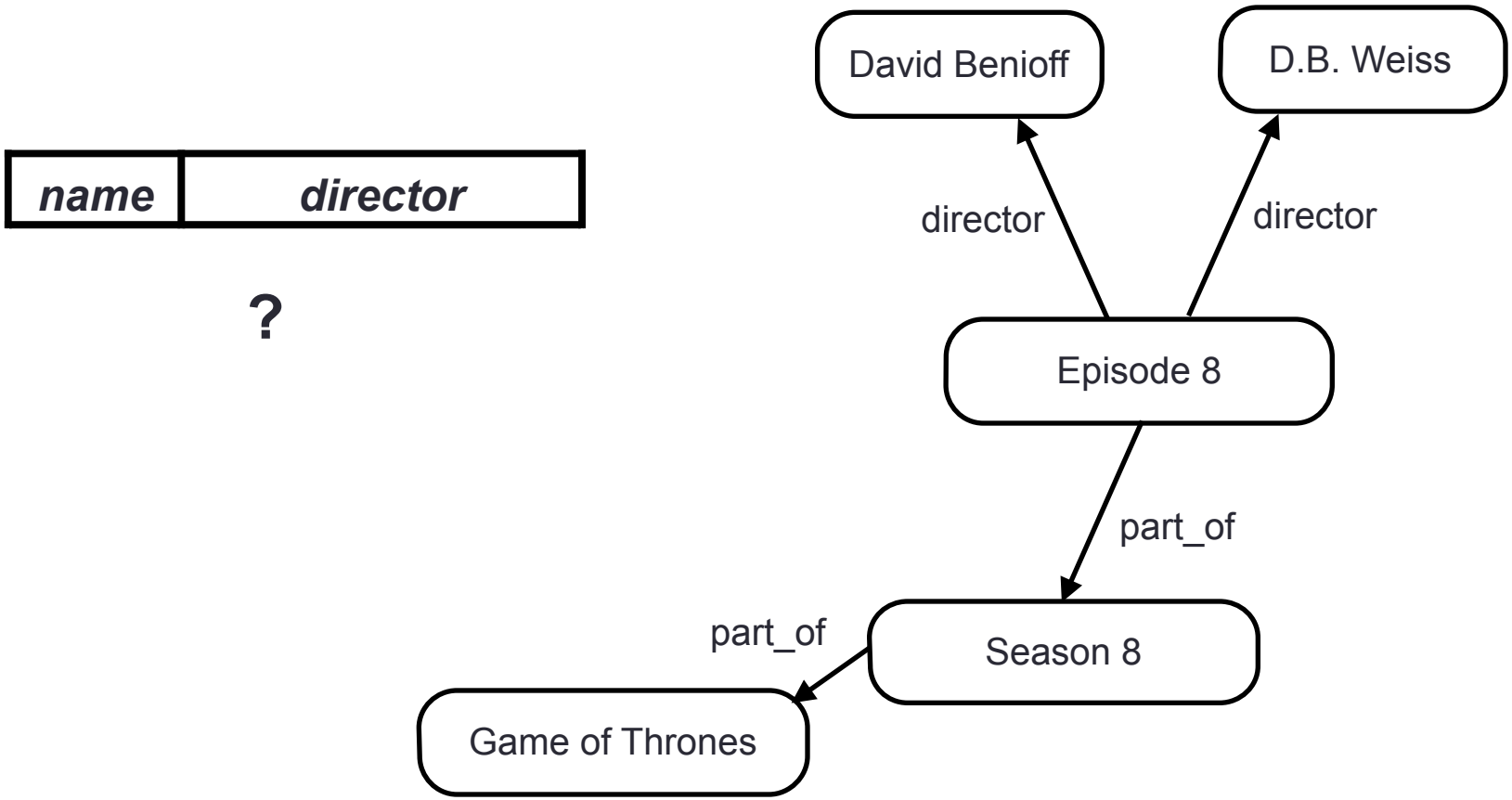
# Unstructured Data:
## advantages of going without schema



# Unstructured Data:
## advantages of going without schema



# Unstructured Data:
## advantages of going without schema



# Unstructured Data:
## advantages of going without schema

```
{
    “name”: “Crazy, Stupid Love”
    “director: “Glenn Ficarra”
}
```

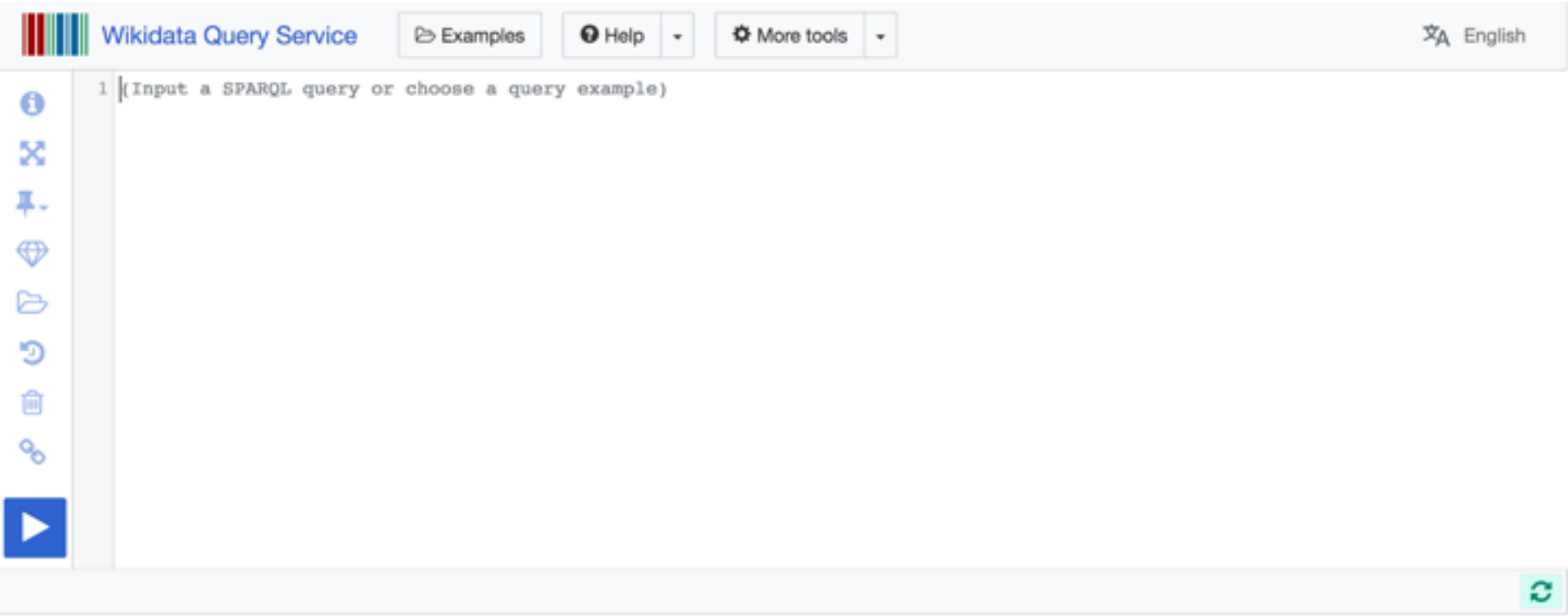
# Unstructured Data:
## advantages of going without schema

```
{
    “name”: “Crazy, Stupid Love”
    “director: “Glenn Ficarra”
},
{
    “name”: “Matrix”
    “director”: [“Lana W.”, “Lilly W.” ]
}
```

As databases grow,
we need a way to understand what they have
and how to query them

As databases grow,
we need a way to understand what they have
and how to query them

As databases grow,
we need a way to understand what they have
and how to query them



So, it appears we do need schemas

- for JSON data
- for graphs
- even for tabular or text data!

This talk:

SHACL (Shapes Constraint Language)
ShEx (Shape Expressions)

JSON Schema

Schemas for graphs and other forms of semi-structured data

Why these? Why now?

SHACL: W3C recommendation (Mid 2017)
ShEx Group still working (last update 4’2019)
JSON Schema: Working Group in IETF

Also working group for schemas in property graphs

Millennium Institute
Foundational
Research on Data

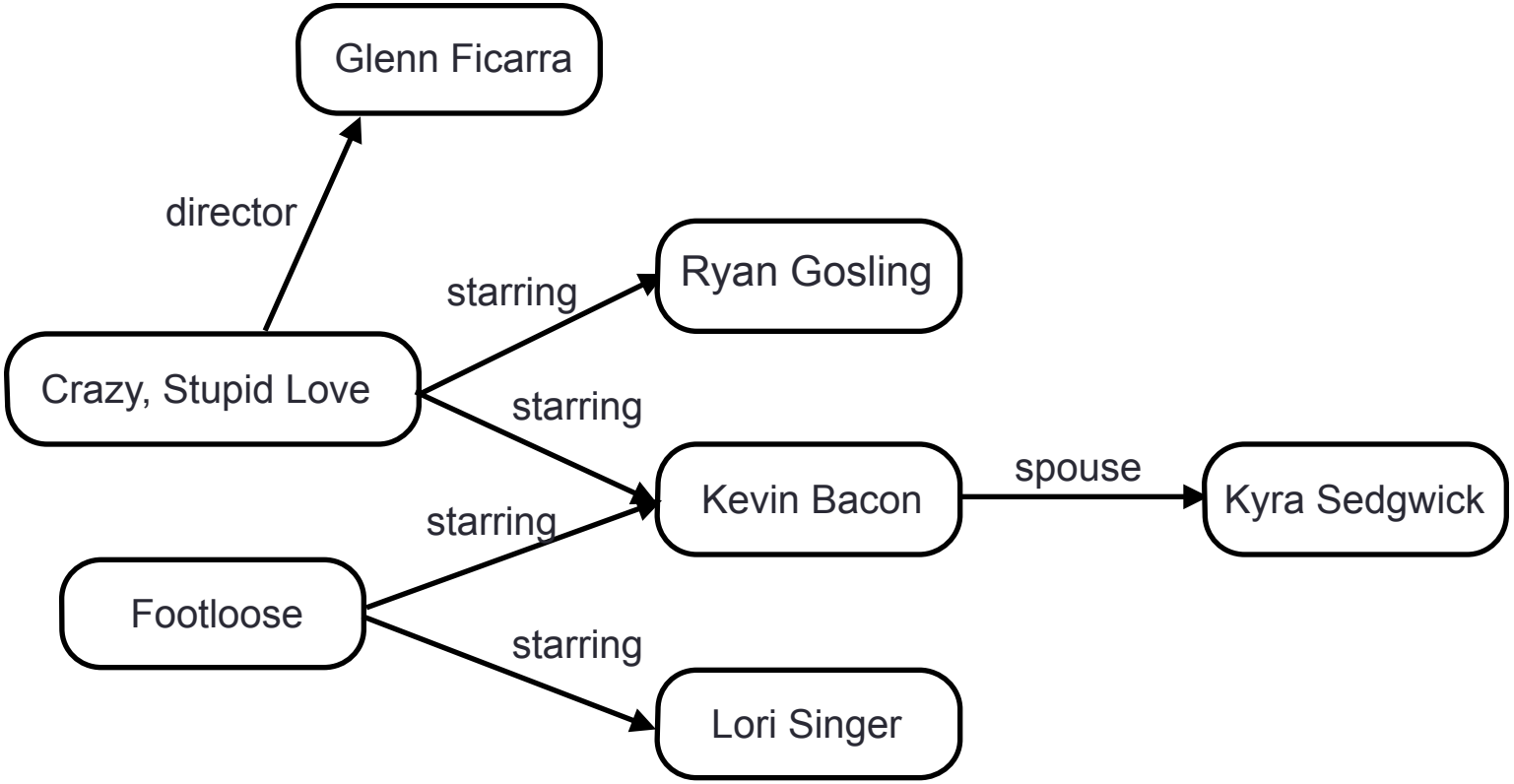
# Schemas for graphs and other forms of semi-structured data


Juan L. Reutter

PUC Chile

Felipe Pezoa, Domagoj Vrgoč, Martín Ugarte, Fernando Suarez, Pierre Bouhris, Ognjen Savkovic, Julien Corman, Fernando Florenzano, Iovka Boneva, Sławek Staworko

# RDF Graphs




In this talk: Set V of nodes
Edges over V x V, labelled with a string

# JSON

```
{
    “name”: “Crazy, Stupid Love”
    “director: “Glenn Ficarra”
},
{
    “name”: “Matrix”
    “director”: [Lana W., Lilly W. ]
}
```

Data is always about resources,
and linking resources with other resources.

Schema imposes conditions on some of them.

# Shape-based Schemas - general form

$\mathcal{L}_{\text{type}}$

language to express shapes

$\mathcal{L}_{\text{const}}$

language to express constraints

# Shape-based Schemas - general form

$\mathcal{L}_{\text{type}}$ language to express shapes

$\mathcal{L}_{\text{const}}$ language to express constraints

$T(x)$ Answers of this query must be of a shape

$\varphi(x)$ Nodes of the shape must satisfy this query

# Shape-based Schemas - general form

$\mathcal{L}_{\text{type}}$ language to express shapes

$\mathcal{L}_{\text{const}}$ language to express constraints

$T(x)$ Answers of this query must be of a shape

$\varphi(x)$ Nodes of the shape must satisfy this query

$$T(x) \rightarrow \varphi(x)$$

# JSON Schema

```
{
  “name”: “Aconcagua”,
  “elevation”: 6960,
  “country”: “Argentina”,
  “first_ascender”: {
    “name”: “Matthias”,
    “surname”: “Zurbriggen”
  }
}
```

```
{
  "type": "object",
  "properties": {
    "name": {"type": "string"},
    "elevation": {"type": "integer"},
    "country": {"type": "string"},
    "first_ascender": {
          ...
    }
  }
}
```

# JSON Schema

$\mathcal{L}_{\text{type}}$ root shape must conform root JSON Schema

$\mathcal{L}_{\text{const}}$ There must be a name (string),
there must be a country (string),…

If there is a first ascender, then

# JSON Schema

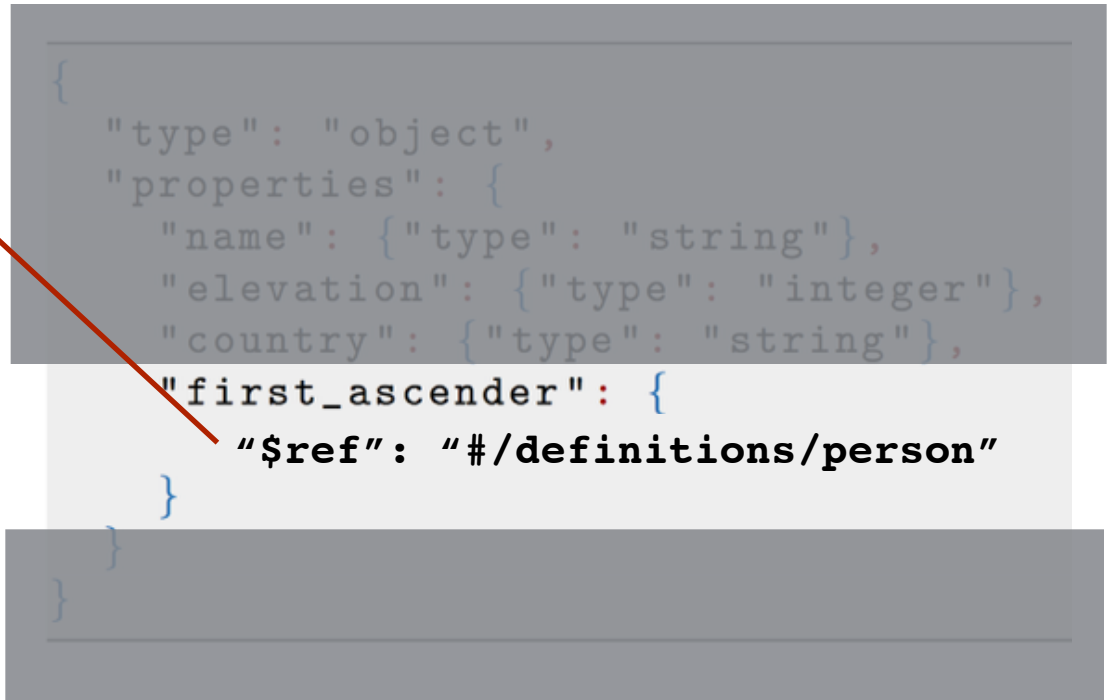
```
{
  “name”: “Aconcagua”,
  “elevation”: 6960,
  “country”: “Argentina”,
  “first_ascender”: {
    “name”: “Matthias”,
    “surname”: “Zurbriggen”
  }
}
```

```
{
  "type": "object",
  "properties": {
    "name": {"type": "string"},
    "elevation": {"type": "integer"},
    "country": {"type": "string"},
    "first_ascender": {
          ...
    }
  }
}
```

# JSON Schema

```
{
  “name”: “Aconcagua”,
  “elevation”: 6960,
  “country”: “Argentina”,
  “first_ascender”: {
    “name”: “Matthias”,
    “surname”: “Zurbriggen”
  }
}
```

```
“definitions”: {
  “person”: {
     “type”: “object”,
     “properties”: {
          “name”: {“type”: “string”},
          “surname”: {“type”: “string”}
          }
     }
}
```



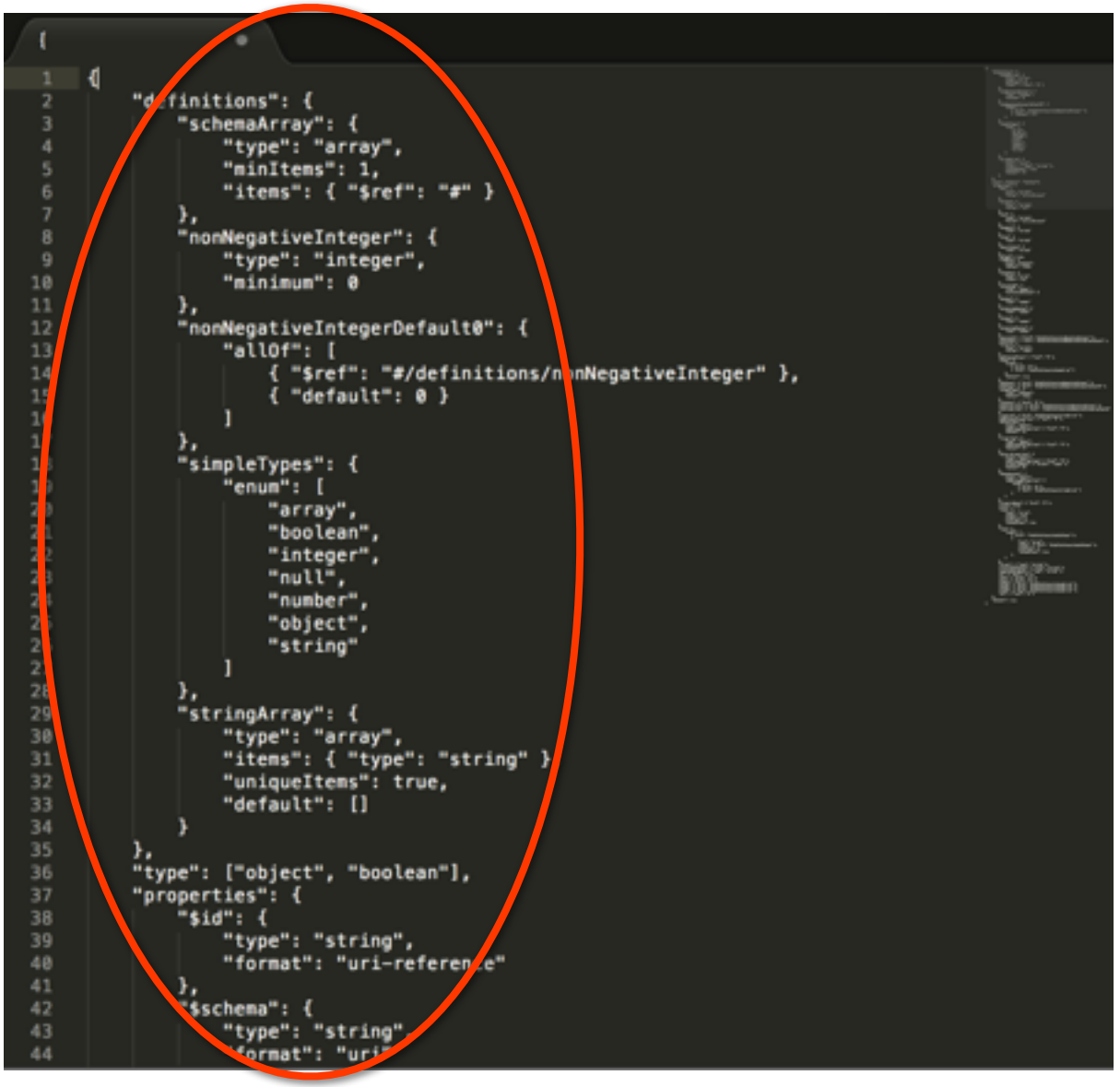
# JSON Schema

$\mathcal{L}_{\text{type}}$ root shape must conform root JSON Schema

$\mathcal{L}_{\text{const}}$ There must be a name (string),
there must be a country (string),…

If there is a first ascender, then it satisfies shape person

# Real JSON schemas use a lot of shapes



# Shape-based Schemas - general form

$\mathcal{L}_{\text{type}}$

language to express shapes

$\mathcal{L}_{\text{const}}$

language to express constraints

$\mathcal{S}$ Set of shapes (person, address, mountain, etc…)

$T_S(x)$ Answers of this query must be of shape S

$\varphi_S(x)$ Nodes of shape S must satisfy this query.
Query can use shape names!

# SHACL

```
:movieShape
     a   sh:NodeShape ;
     sh:targetClass  :movie ;
     sh:property [
          sh:path  :starring ;
          sh:node  :personShape
     ] ;
     sh:property [
          sh:path  :director ;
          sh:minCount  1 ;
          sh:node  :personShape
     ] ;
```

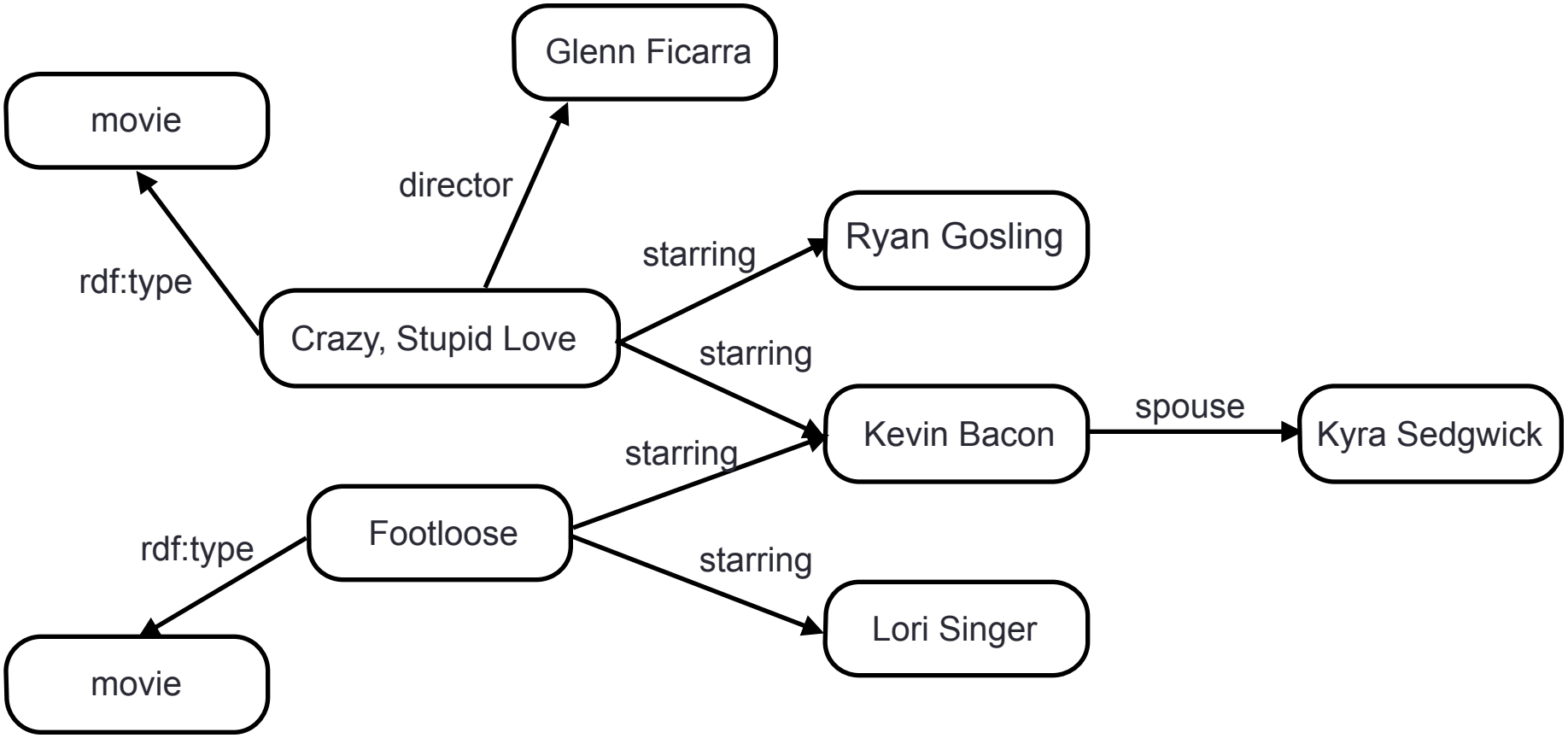
```
:personShape
     a   sh:NodeShape ;
     sh:property [
     sh:path  :spouse ;
     sh:node  :personShape
     ] .
```

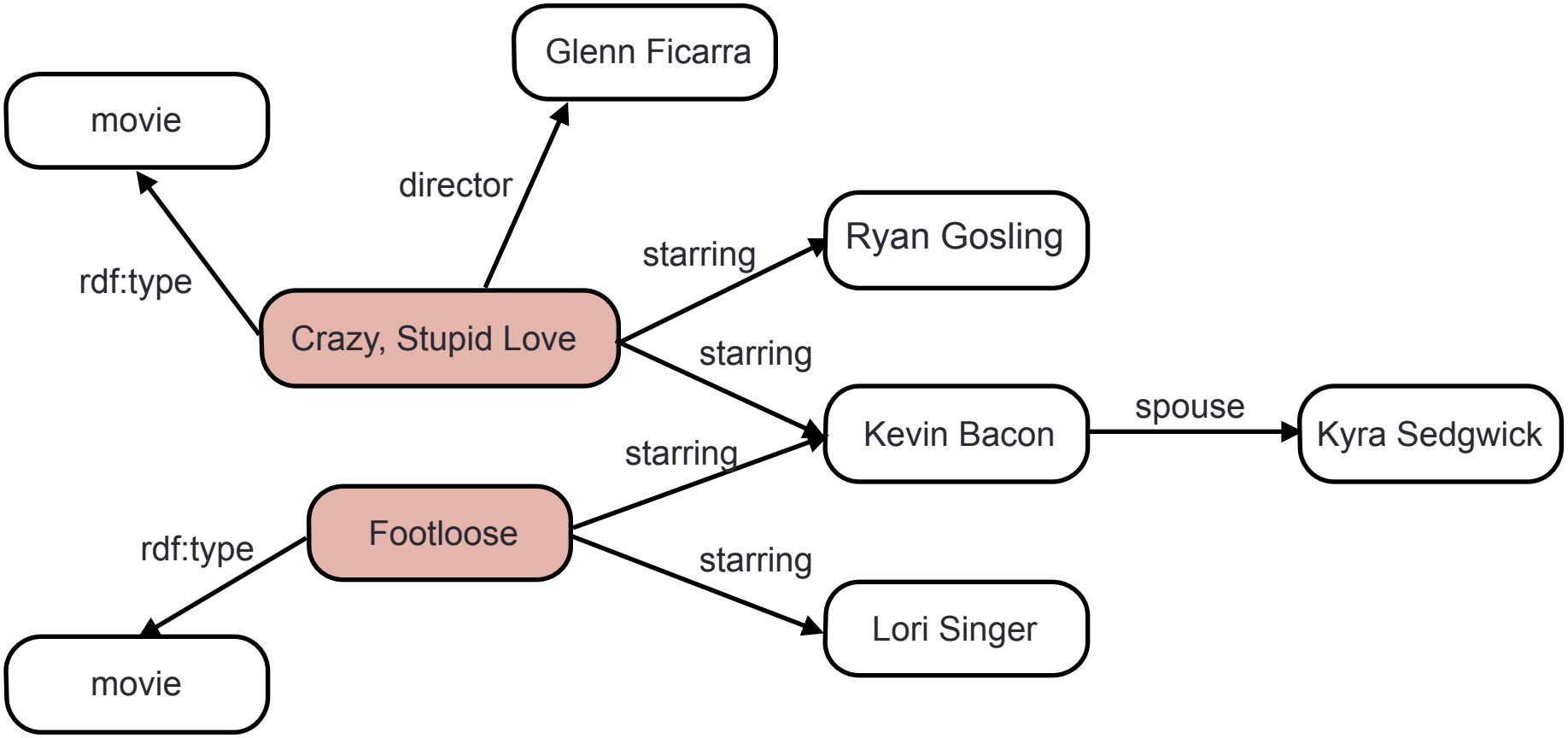
# SHACL

```
:movieShape
    a   sh:NodeShape ;
    sh:targetClass  :movie ;
    sh:property [
        sh:path  :starring ;
        sh:node  :personShape
    ] ;
    sh:property [
        sh:path  :director ;
        sh:minCount  1 ;
        sh:node  :personShape
    ] ;
```

```
:personShape
    a   sh:NodeShape ;
    sh:property [
    sh:path  :spouse ;
    sh:node  :personShape
    ] .
```

All nodes of type :movie must conform to :movieShape


Glenn Ficarra
movie
director
rdf:type
Crazy, Stupid Love
starring
Ryan Gosling
starring
Kevin Bacon
spouse
Kyra Sedgwick
starring
Footloose
rdf:type
starring
Lori Singer
movie



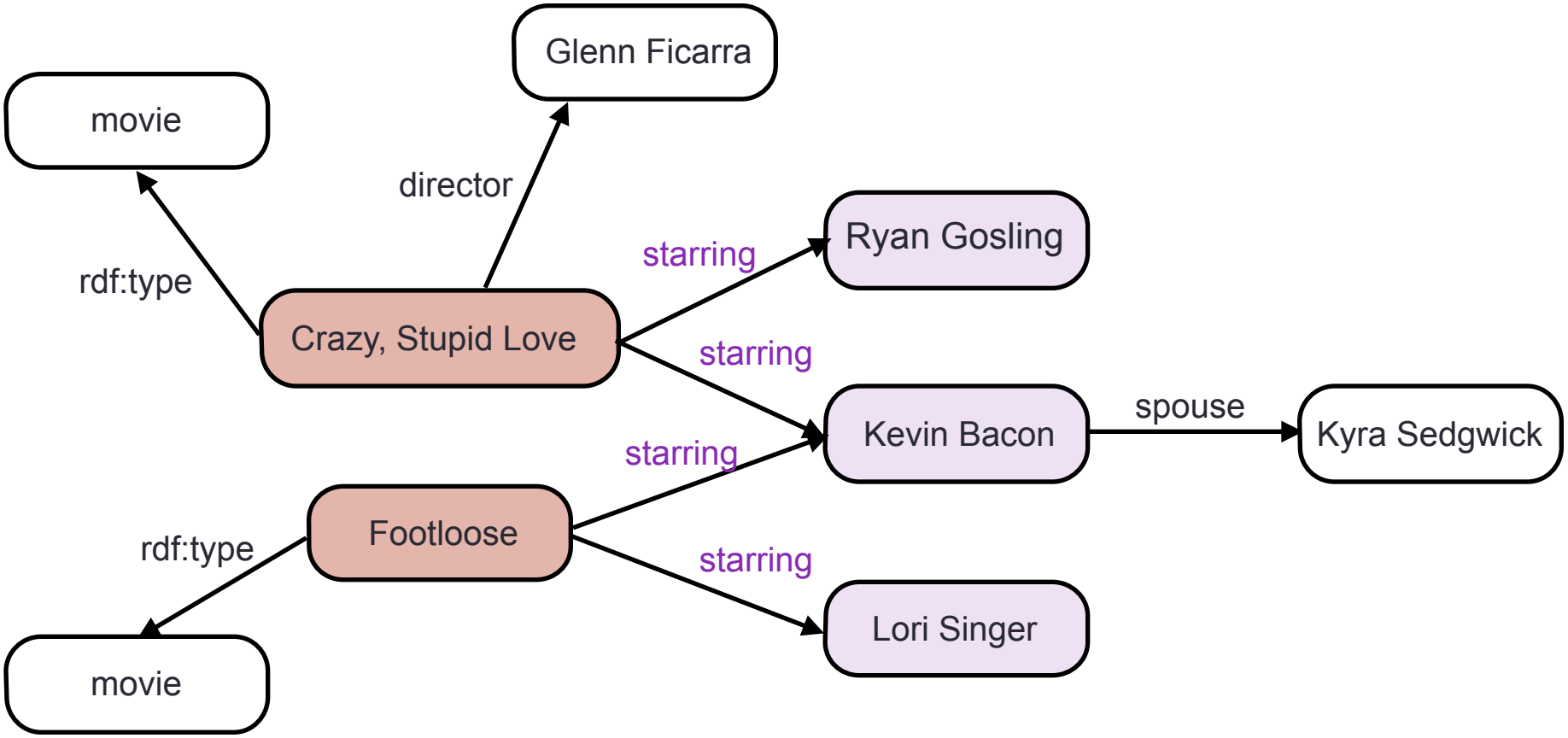

these nodes must conform to :movieShape

# SHACL

```
:movieShape
    a   sh:NodeShape ;
    sh:targetClass  :movie ;
    sh:property [
        sh:path  :starring ;
        sh:node  :personShape
    ] ;
    sh:property [
        sh:path  :director ;
        sh:minCount  1 ;
        sh:node  :personShape
    ] ;
```

```
:personShape
    a   sh:NodeShape ;
    sh:property [
    sh:path  :spouse ;
    sh:node  :personShape
    ] .
```

Neighbours of nodes assigned :movieShape,
connected by :starring,
must satisfy :personShape




these nodes must conform to :personShape

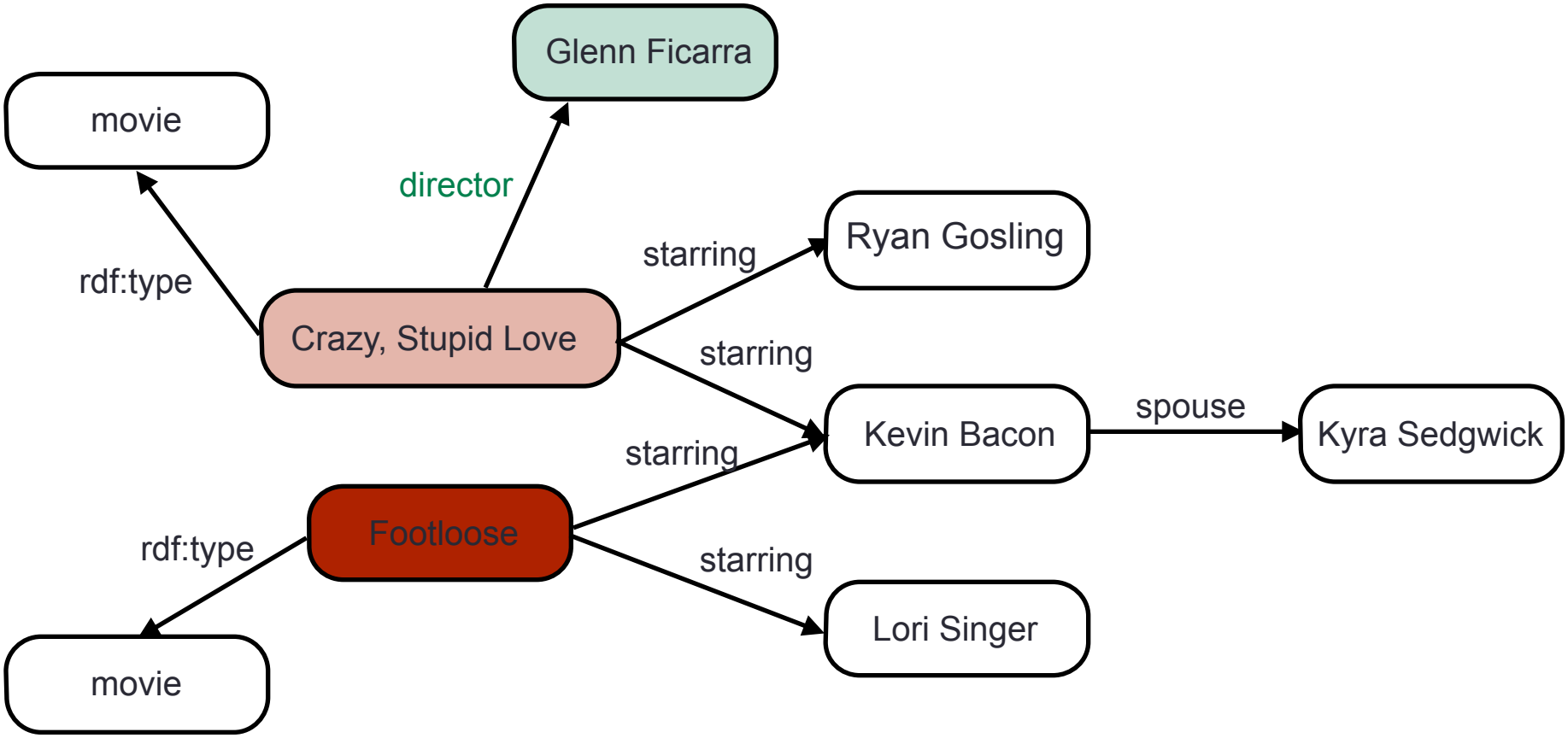
# SHACL

```
:movieShape
    a   sh:NodeShape ;
    sh:targetClass  :movie ;
    sh:property [
        sh:path  :starring ;
        sh:node  :personShape
    ] ;
    sh:property [
        sh:path  :director ;
        sh:minCount  1 ;
        sh:node  :personShape
    ] ;
```

```
:personShape
    a   sh:NodeShape ;
    sh:property [
    sh:path  :spouse ;
    sh:node  :personShape
    ] .
```

Neighbours of nodes assigned :movieShape,
connected by :director,
must satisfy :personShape,
we need at least 1

this node must conform to :personShape



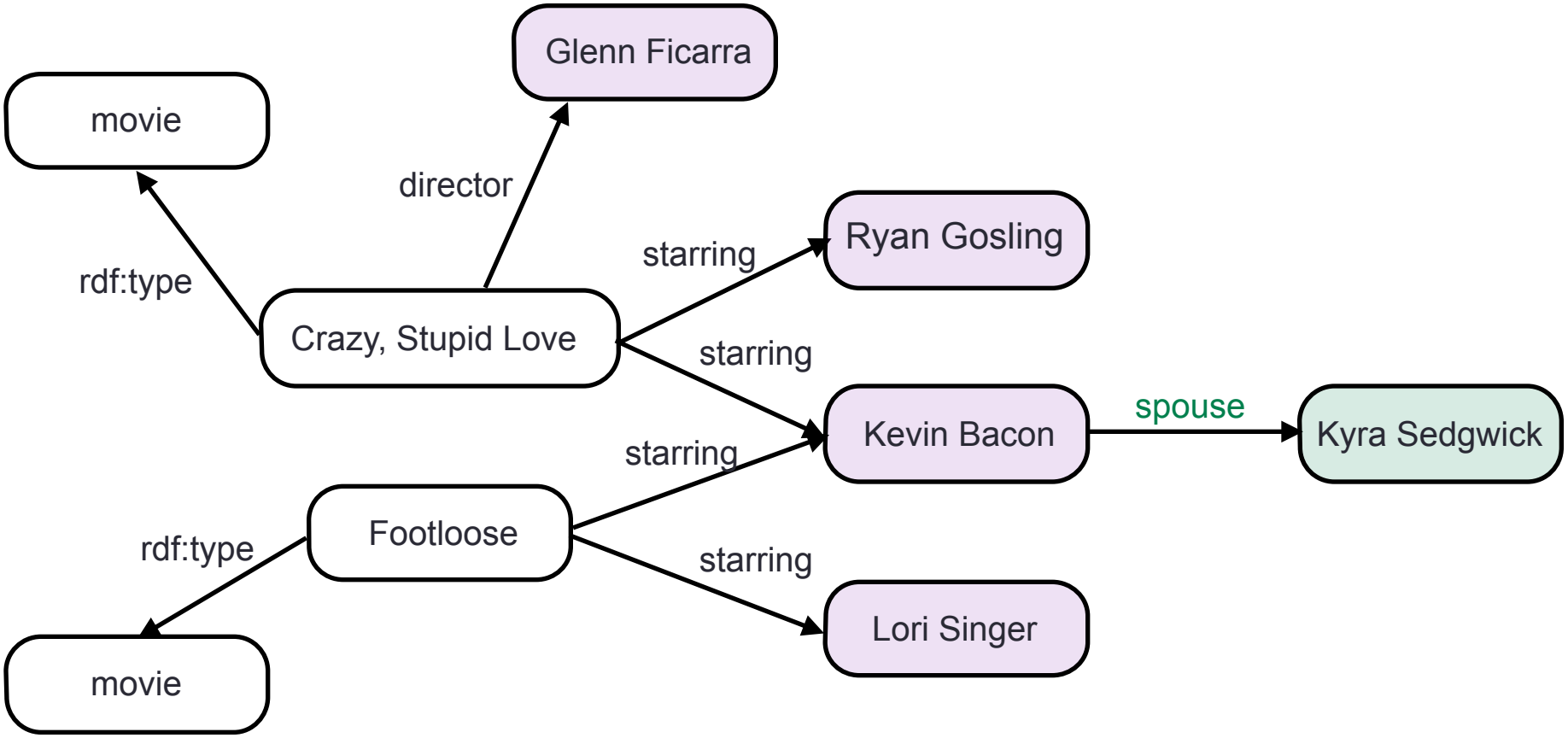

violation: every movie needs at least one director

# SHACL

```
:movieShape
    a   sh:NodeShape ;
    sh:targetClass  :movie ;
    sh:property [
        sh:path  :starring ;
        sh:node  :personShape
    ] ;
    sh:property [
        sh:path  :director ;
        sh:minCount  1 ;
        sh:node  :personShape
    ] ;
```

```
:personShape
    a   sh:NodeShape ;
    sh:property [
    sh:path  :spouse ;
    sh:node  :personShape
    ] .
```

Neighbours of nodes assigned :personShape,
connected by :spouse,
must satisfy :personShape




these nodes must conform to :personShape

# Shape-based Schemas - general form

$\mathcal{L}_{\text{type}}$

language to express shapes

$\mathcal{L}_{\text{const}}$

language to express constraints

$\mathcal{S}$ Set of shapes (person, address, mountain, etc…)

$T_S(x)$ Answers of this query must be of shape S

$\varphi_S(x)$ Nodes of shape S must satisfy this query.
Query can use shape names!

# SHACL

$\mathcal{L}_{\text{type}}$ Individual nodes
Answers of query {?x rdf:type U}

$\mathcal{L}_{\text{const}}$
- Is a string, is a number, …
- # of neighbours connected by a path
- what my neighbours satisfy (these can be other shapes)

$\mathcal{S}$

$T_S(x)$

$\varphi_S(x)$

# SHACL

$\mathcal{L}_{\text{type}}$

Individual nodes
Answers of query {?x rdf:type U}

$\mathcal{L}_{\text{const}}$

- Is a string, is a number, …
- # of neighbours connected by a path
- what my neighbours satisfy (these can be other shapes)

~ FO with 2 variables + counting + paths
(modal logic with counting)

$\mathcal{S}$

$T_S(x)$

$\varphi_S(x)$

# SHACL

$\mathcal{L}_{\text{type}}$

Individual nodes
Answers of query {?x rdf:type U}

$\mathcal{L}_{\text{const}}$

- Is a string, is a number, …
- # of neighbours connected by a path
- what my neighbours satisfy (these can be other shapes)

~ FO with 2 variables + counting + paths
(modal logic with counting)

$\mathcal{S}$
$T_S(x)$
$\varphi_S(x)$

semantics is not trivial!

# ShEx

$\mathcal{L}_{\text{type}}$ allows any pattern of the form {?x p U} or {U p ?x}

$\mathcal{L}_{\text{const}}$ very similar to SHACL (will return to this this)

# Why do we study Shape-based Schemas?

All these languages are specifications or established drafts

Need for formal specification.

Understand best way of defining things

How to solve tasks:  validation, satisfiability, …

# Need for formal specification?

# Need for formal specification?

JSON Schema was quite messy when we started (2015)

| | V1 | V2 | V3 | V4 | V5 |
|---|---|---|---|---|---|
| T1: | N | Y | Y | N | Y |
| T2: | Y | N | Y | N | Y |
| T3: | N | Y | N | N | N |
| T4: | – | – | N | – | – |

| | |
|---|---|
| Y | valid |
| N | invalid |
| – | unsupported |

Each test T1-T4: validating a document against a schema
V1-V5: first 5 validators in google search
(circa 10/2015)

# Need for formal specification?

SHACL official W3C recommendation

The validation with recursive shapes is not defined in SHACL and is left to SHACL processor implementations. For example, SHACL processors may support recursion scenarios or produce a failure when they detect recursion.

# Need for formal specification?

SHACL official W3C recommendation

The validation with recursive shapes is not defined in SHACL and is left to SHACL processor implementations. For example, SHACL processors may support recursion scenarios or produce a failure when they detect recursion.

ShEx report late 2018

This is an editor's draft of the Shape Expressions specification. ShEx 2.x differs significantly from the W3C ShEx Submission. The July 2017 publication included a definition of validation which implied infinite recursion. This version explicitly includes recursion checks. No tests changed as a result of this and no implementations or applications are known to have been affected.

# Why do we study Shape-based Schemas?

All these languages are specifications or established drafts

Need for formal specification.

Understand best way of defining things

How to solve tasks:  validation, satisfiability, …

# Why do we study Shape-based Schemas?

All these languages are specifications or established drafts

Need for formal specification.

Understand best way of defining things

Graphs

How to solve tasks: validation, satisfiability, …

# SHACL/ShEx

Best way of defining things

- syntax
- semantics

Tasks: validation, satisfiability, …

# Defining Shape-based Schemas

$\mathcal{L}_{\text{type}} + \mathcal{L}_{\text{const}} + \text{Semantics}$

# Defining Shape-based Schemas

$\mathcal{L}_{\text{type}}$ Way of selecting nodes that must be of shape S

- must select particular node
- Specs use very simple queries {?x p U} or {U p ?x}

# Defining Shape-based Schemas

$\mathcal{L}_{\text{type}}$ Way of selecting nodes that must be of shape S

- must select particular node
- Specs use very simple queries {?x p U} or {U p ?x}

Any unary query would do

if $\mathcal{L}_{\text{type}} \subseteq \mathcal{L}_{\text{const}}$

then most likely this does not affect the expressive power

# Defining Shape-based Schemas

$\mathcal{L}_{\text{const}}$ What nodes of shape S must satisfy

# Defining Shape-based Schemas: SHACL

$\mathcal{L}_{\text{const}}$ What nodes of shape S must satisfy

- unary tests (is a string, is this node, etc)
- shape tests (node is assigned a shape)

# Defining Shape-based Schemas: SHACL

$\mathcal{L}_{\text{const}}$ What nodes of shape S must satisfy

- unary tests (is a string, is this node, etc)
- shape tests (node is assigned a shape)
- counting neighbours:

$\geq_n p.\ \varphi$

$\leq_n p.\ \varphi$

min/max # of p-neighbours satisfying $\varphi$

# Defining Shape-based Schemas: SHACL

$\mathcal{L}_{\text{const}}$ What nodes of shape S must satisfy

- unary tests (is a string, is this node, etc)
- shape tests (node is assigned a shape)
- counting neighbours:

$\geq_n p.\ \varphi$

$\leq_n p.\ \varphi$

min/max # of p-neighbours satisfying $\varphi$

- comparing paths:

$EQ(p_1, p_2)$ set of $p_1$-neighbours = set of $p_2$-neighbours

Paths are defined using RPQs/property paths

# SHACL

```
:movieShape
     a   sh:NodeShape ;
     sh:targetClass  :movie ;
     sh:property [
          sh:path  :starring ;
          sh:node  :personShape
     ] ;
     sh:property [
          sh:path  :director ;
          sh:minCount  1 ;
          sh:node  :personShape
     ] ;
```

$\leq_0$ :starring.(¬:personShape)

# SHACL

```
:movieShape
     a   sh:NodeShape ;
     sh:targetClass  :movie ;
     sh:property [
          sh:path  :starring ;
          sh:node  :personShape
     ] ;
     sh:property [
          sh:path  :director ;
          sh:minCount  1 ;
          sh:node  :personShape
     ] ;
```

$\geq_1$ :director.(:personShape)

# SHACL

```
:movieShape
     a   sh:NodeShape ;
     sh:targetClass  :movie ;
     sh:property [
          sh:path  :starring ;
          sh:node  :personShape
     ] ;
     sh:property [
          sh:path  :director ;
          sh:minCount  1 ;
          sh:node  :personShape
     ] ;
```

$$T_{\text{:movieShape}} = \{?x\ rdf{:}type\ \ :movie\}$$

# SHACL

```
:movieShape
     a   sh:NodeShape ;
     sh:targetClass  :movie ;
     sh:property [
          sh:path  :starring ;
          sh:node  :personShape
     ] ;
     sh:property [
          sh:path  :director ;
          sh:minCount  1 ;
          sh:node  :personShape
     ] ;
```

$T_{\text{:movieShape}} = \{?x \text{ rdf:type :movie}\}$

$\varphi_{\text{:movieShape}} = \leq_0 \text{:starring}.(\neg\text{:personShape}) \wedge \geq_1 \text{:director}.(\text{:personShape})$

# Defining Shape-based Schemas: ShEx

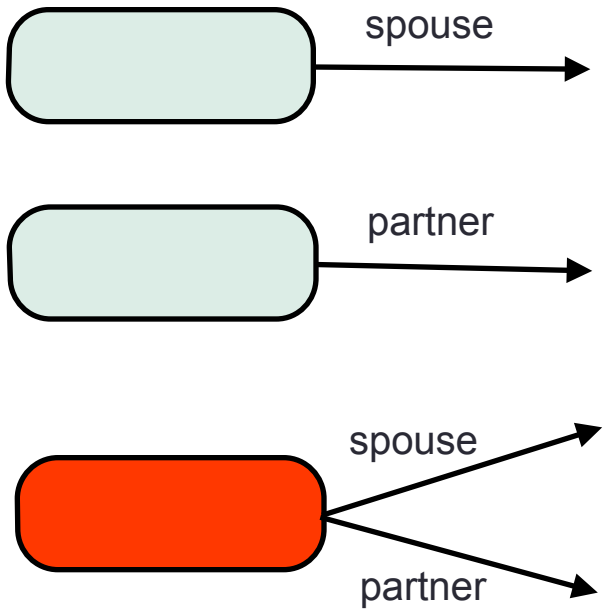
$\mathcal{L}_{\text{const}}$ What nodes of shape S must satisfy

- unary tests (is a string, is this node, etc)
- shape tests (node is assigned a shape)

# Defining Shape-based Schemas: ShEx

$\mathcal{L}_{const}$ What nodes of shape S must satisfy

- unary tests (is a string, is this node, etc)
- shape tests (node is assigned a shape)

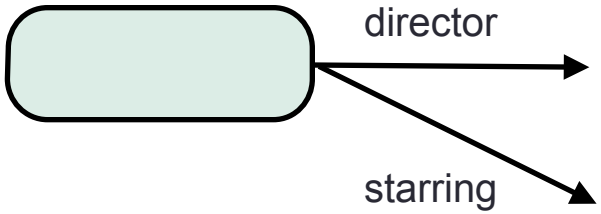



spouse @:personShape ;
partner @:personShape

# Defining Shape-based Schemas: ShEx

$\mathcal{L}_{\text{const}}$ What nodes of shape S must satisfy

- unary tests (is a string, is this node, etc)
- shape tests (node is assigned a shape)




.
.
.

$$\text{director } @\text{:personShape } ;$$
$$(\text{starring } @\text{:personShape})[0, *]$$

# Defining Shape-based Schemas: ShEx

$\mathcal{L}_{\text{const}}$ What nodes of shape S must satisfy

- unary tests (is a string, is this node, etc)
- shape tests (node is assigned a shape)
- regular bag expressions over p @S
  interpreted over bag of neighbours

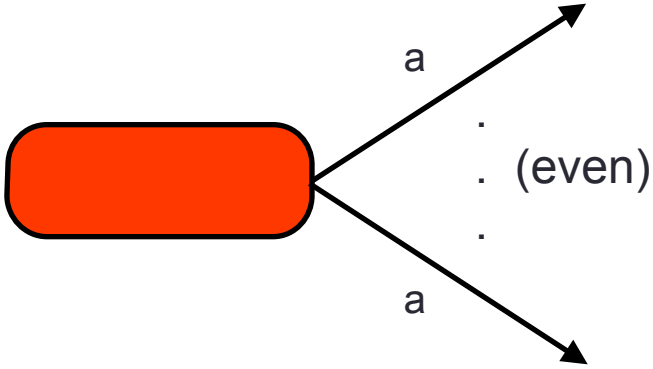
$$\text{exp} = p\ @s \mid \varepsilon \mid \text{exp}|\text{exp} \mid \text{exp};\text{exp} \mid \text{exp}[m, n]$$

# So which one is better?

Word is still open for consideration.

Both formalisms are incomparable:

$$(a\ @S; a\ @S)[0, *]$$

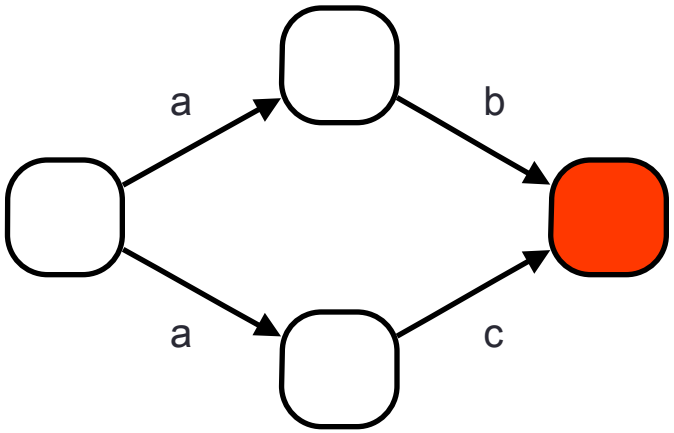


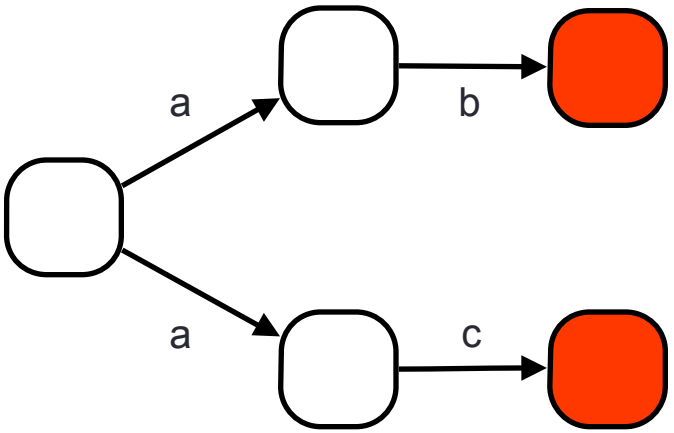
# So which one is better?

Word is still open for consideration.

Both formalisms are incomparable:

$$EQ(ab, ac)$$



# So which one is better?

Word is still open for consideration.

Both formalisms are incomparable.

Complexity issues (data complexity):

- checking if a SHACL constraint holds in a node is tractable
- checking if a ShEx constraint holds is not

# So which one is better?

Word is still open for consideration.

Both formalisms are incomparable

Complexity issues (data complexity):

- checking if a SHACL constraint holds in a node is tractable
- checking if a ShEx constraint holds is not

(usually one restricts to ShEx where the * is not nested)

# So which one is better?

Word is still open for consideration.

Both formalisms are incomparable

Complexity issues.

Expressive power / ease to write

# SHACL/ShEx

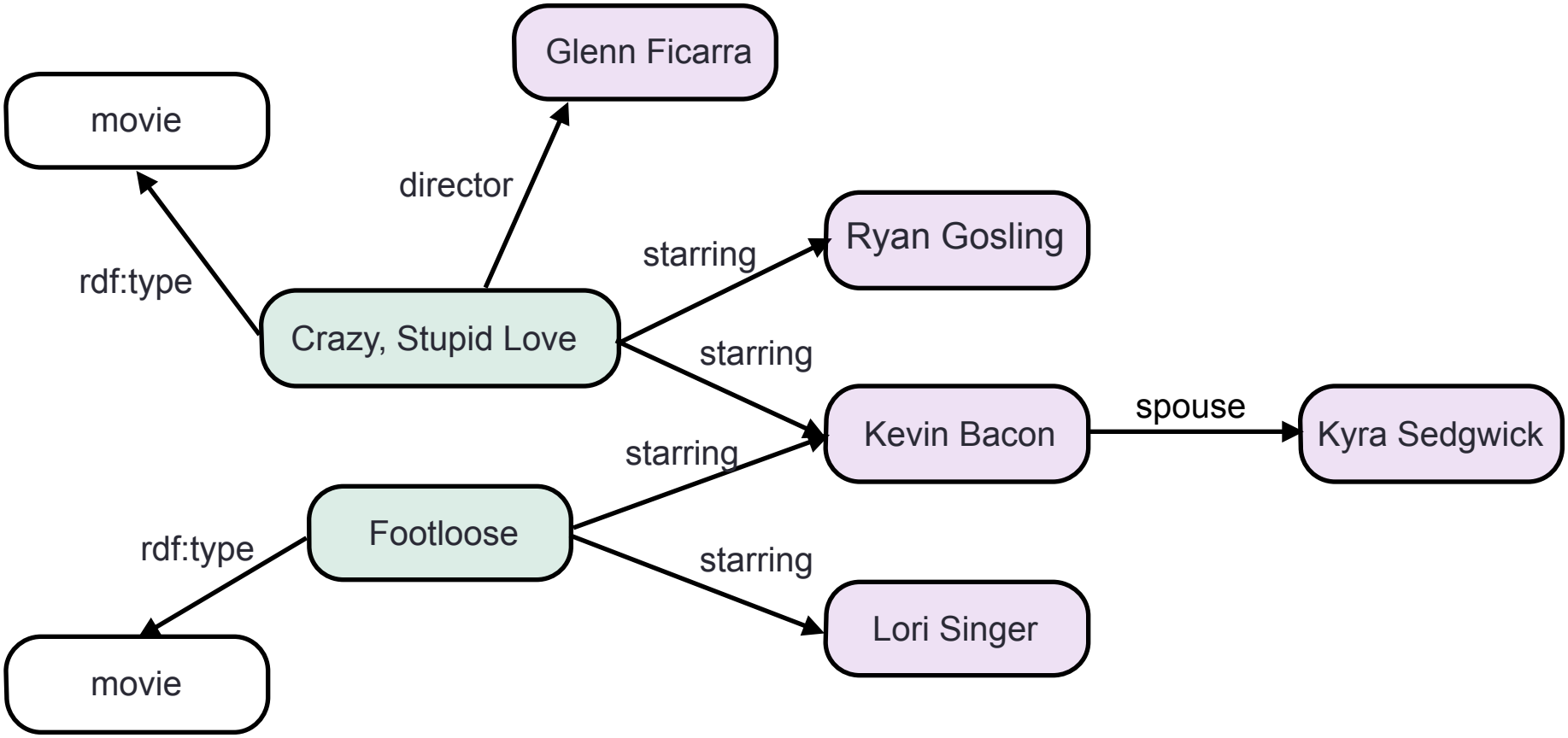
Best way of defining things

- syntax
- semantics

Tasks: validation, satisfiability, …

# Semantics

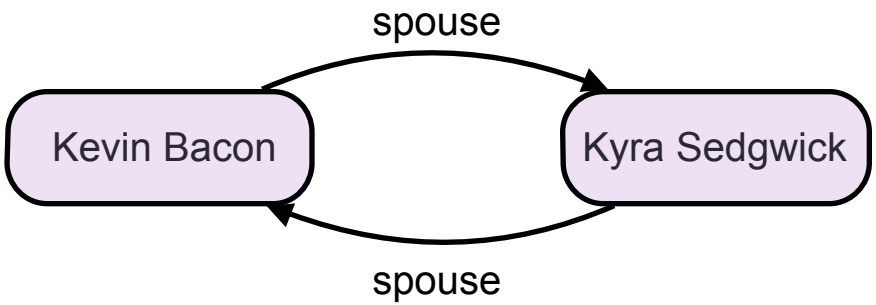
these nodes must conform to :personShape




these nodes must conform to :MovieShape

Semantics:
iteratively assign shapes when needed?

# Semantics:
# iteratively assign shapes when needed?

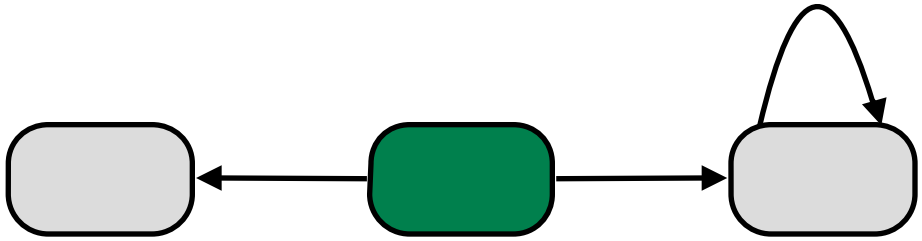


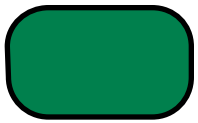

"Spouses of persons are persons"

```
:personShape
     a   sh:NodeShape ;
     sh:property [
     sh:path  :spouse ;
     sh:node  :personShape
     ] .
```

Semantics:
guess a good assignment?

# Semantics:
# guess a good assignment?

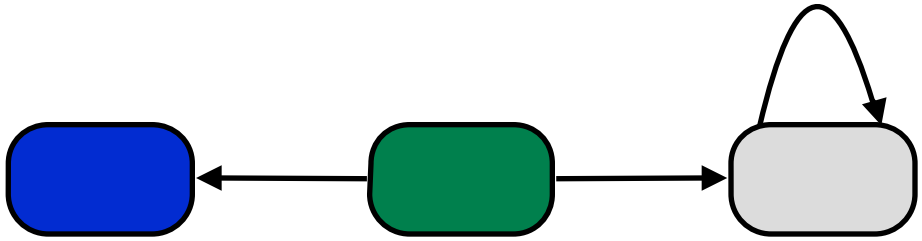


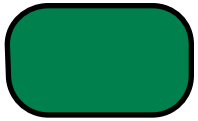


“I have a blue neighbour”

“My neighbours are not blue”

# Semantics:
# guess a good assignment?

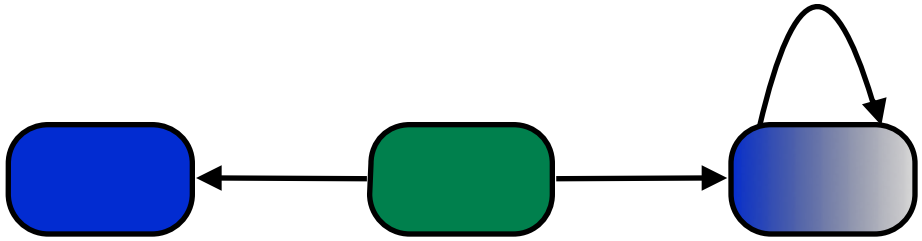


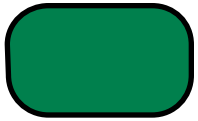


“I have a blue neighbour”

“My neighbours are not blue”

# Semantics:
# guess a good assignment?



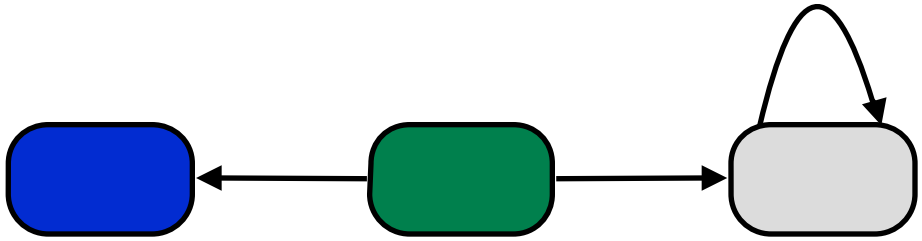


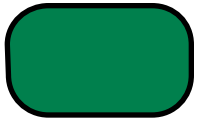
“I have a blue neighbour”

“My neighbours are not blue”

Semantics:
guess a partial good assignment

# Semantics:
# guess a partial good assignment

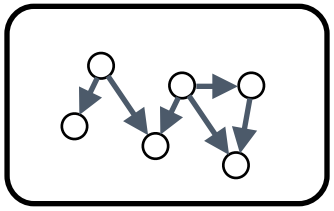




“I have a blue neighbour”

“My neighbours are not blue”

# Graph validating a schema



graph

$s_1, \ldots, s_n$

shapes

$T_{s_1}, \ldots, T_{s_n}$

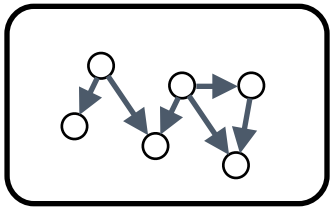
some nodes are assigned shapes

$\varphi_{s_1}, \ldots, \varphi_{s_n}$

nodes in a shape must satisfy these

Can I assign shapes and satisfy all constraints?

# Graph validating a schema



graph

$s_1, \ldots, s_n$

shapes

$T_{s_1}, \ldots, T_{s_n}$

some nodes are assigned shapes

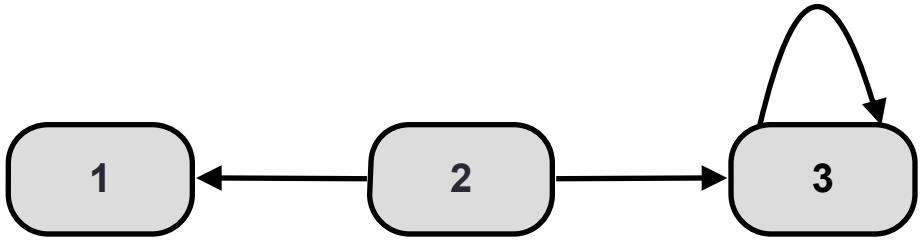
$\varphi_{s_1}, \ldots, \varphi_{s_n}$

nodes in a shape must satisfy these

Can I assign shapes and satisfy all constraints?

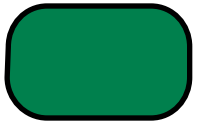
- Assignment respects $T_{s_1}, \ldots, T_{s_n}$
- Assignment agrees with $\varphi_{s_1}, \ldots, \varphi_{s_n}$
- Every node is assigned a shape, its negation, or nothing

# Graph validating a schema




**green**

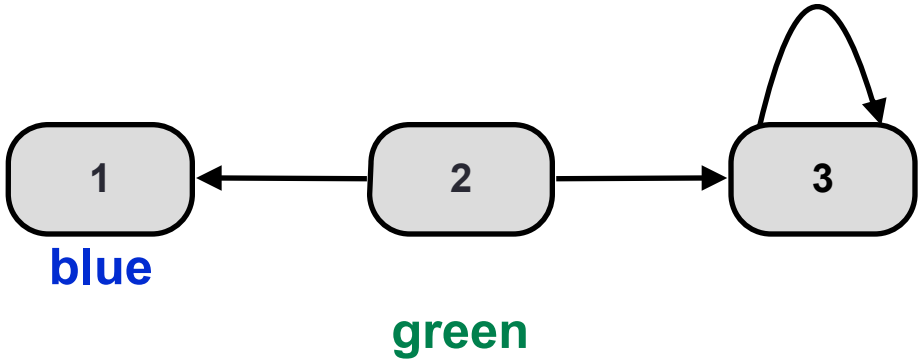
node **2** must be green



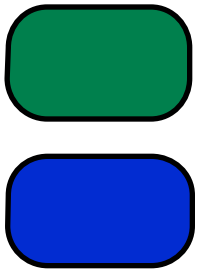
“I have a blue neighbour”

“My neighbours are not blue”

# Graph validating a schema



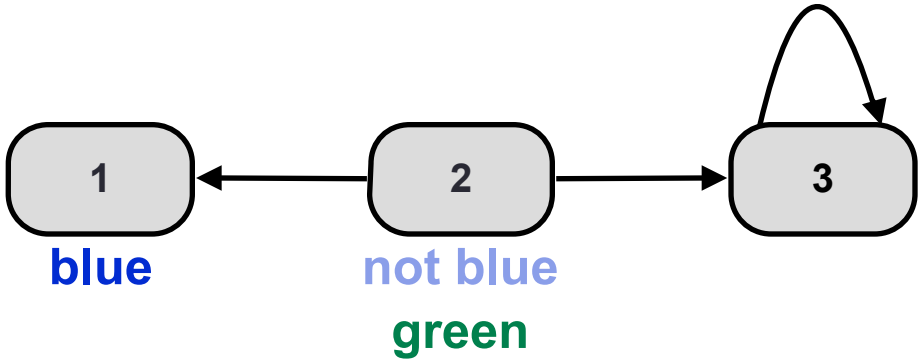

node **2** must be green



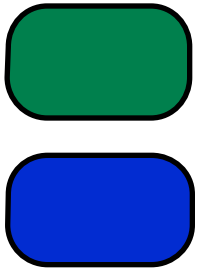
“I have a blue neighbour”

“My neighbours are not blue”

# Graph validating a schema



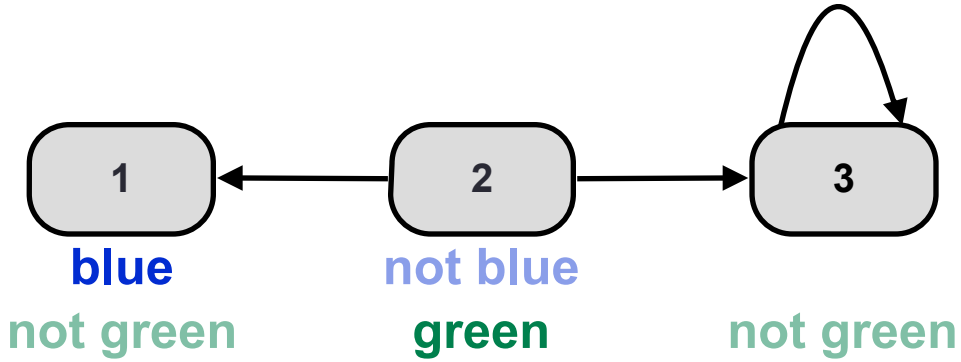

node **2** must be green



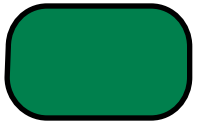
“I have a blue neighbour”

“My neighbours are not blue”

# Graph validating a schema




node **2** must be green

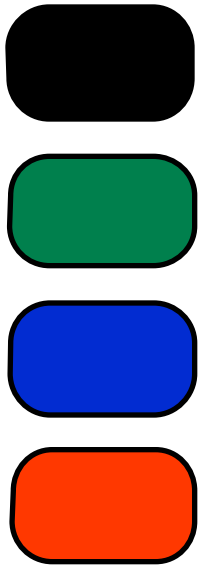


“I have a blue neighbour”

“My neighbours are not blue”

# Graph validating a schema

all nodes must be **black**



“I’m green or blue or red”

“My neighbours are not green”

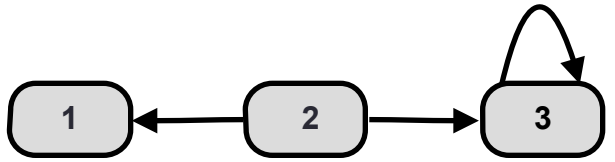
“My neighbours are not blue”

“My neighbours are not red”

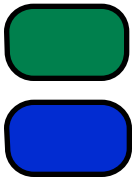
Deciding if a graph validates a schema is NP-complete

# Graph validating a schema: restrictions / aproximations

- Consider only complete assignments




node **2** must be green

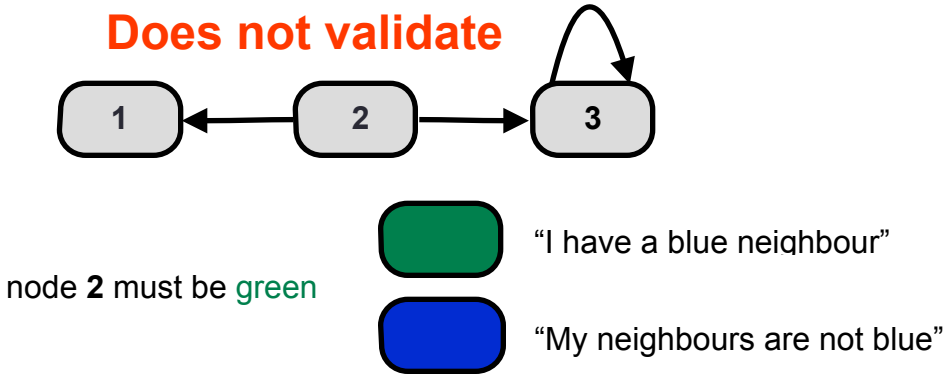


"I have a blue neighbour"

"My neighbours are not blue"

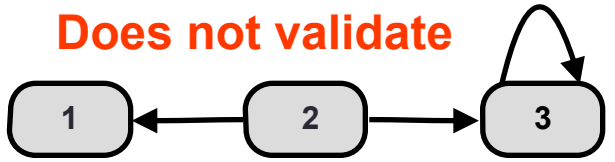
# Graph validating a schema: restrictions / aproximations

- Consider only complete assignments



# Graph validating a schema: restrictions / aproximations

- Consider only complete assignments




node **2** must be green

“I have a blue neighbour”

“My neighbours are not blue”

complete assignment => partial assignment

# Graph validating a schema: restrictions / aproximations

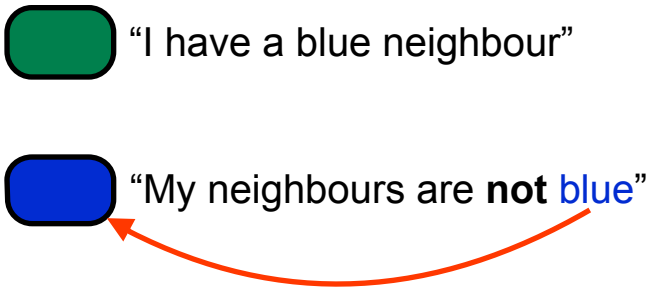
- Consider only complete assignments

- Restrict schemas using stratified negation

# Graph validating a schema: restrictions / aproximations

- Consider only complete assignments

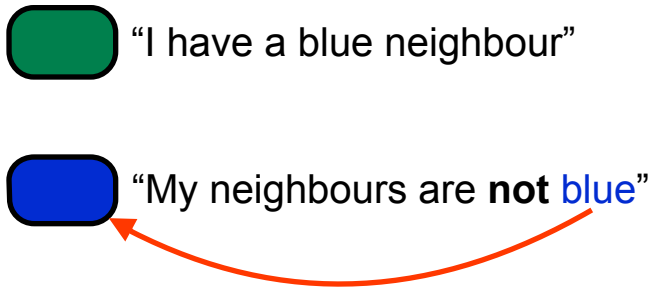
- Restrict schemas using stratified negation



# Graph validating a schema: restrictions / aproximations

- Consider only complete assignments

- Restrict schemas using stratified negation




- still NP-hard
- only need complete assignments

# Graph validating a schema: restrictions / aproximations

- Consider only complete assignments

- Restrict schemas using stratified negation

+

- Only care about assignments that can be built iteratively

  - easy to compute
  - misses assignments: may not validate reasonable graphs

# Graph validating a schema: everything is pretty when non-recursive

- All notions of assignment are equivalent

- Problem in PTIME if checking constraints is in PTIME

- Can even be transformed into SPARQL queries

# Graph validating a schema: everything is pretty when non-recursive

- All notions of assignment are equivalent

- Problem in PTIME if checking constraints is in PTIME

- Can even be transformed into SPARQL queries

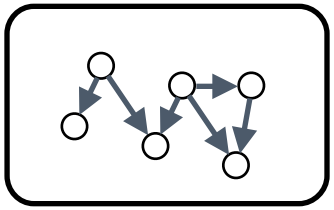
```
:personShape
     a   sh:NodeShape ;
     sh:property [
     sh:path  :spouse ;
     sh:node  :personShape
     ] .
```

# SHACL/ShEx

Best way of defining things

Tasks: validation, satisfiability, …

# Validation



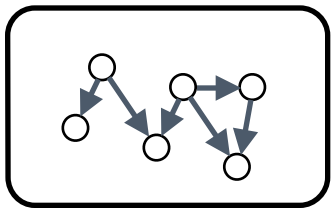
graph

$$s_1, \ldots, s_n$$
$$T_{s_1}, \ldots, T_{s_n}$$
$$\varphi_{s_1}, \ldots, \varphi_{s_n}$$

Is this valid?

# Validation … as in ASP

$s_1, \ldots, s_n$
$T_{s_1}, \ldots, T_{s_n}$
$\varphi_{s_1}, \ldots, \varphi_{s_n}$

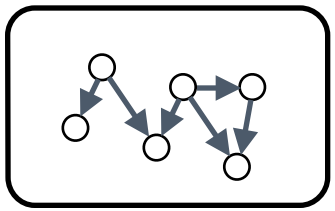
graph

- Compute SPARQL query

# Validation … as in ASP

$s_1, \ldots, s_n$
$T_{s_1}, \ldots, T_{s_n}$
$\varphi_{s_1}, \ldots, \varphi_{s_n}$

"evaluate"



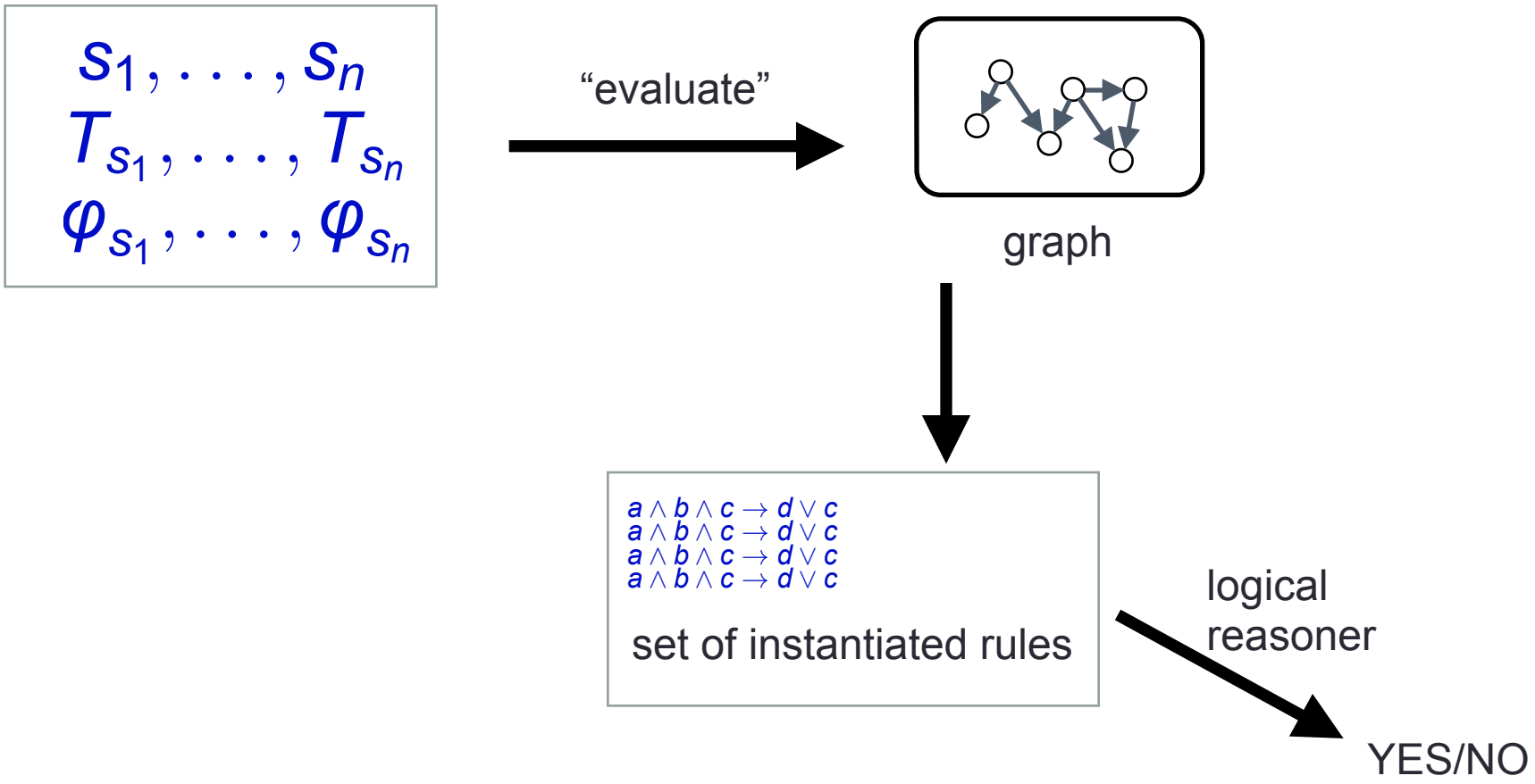
graph

- Compute SPARQL query
- Build rules from answers of this query

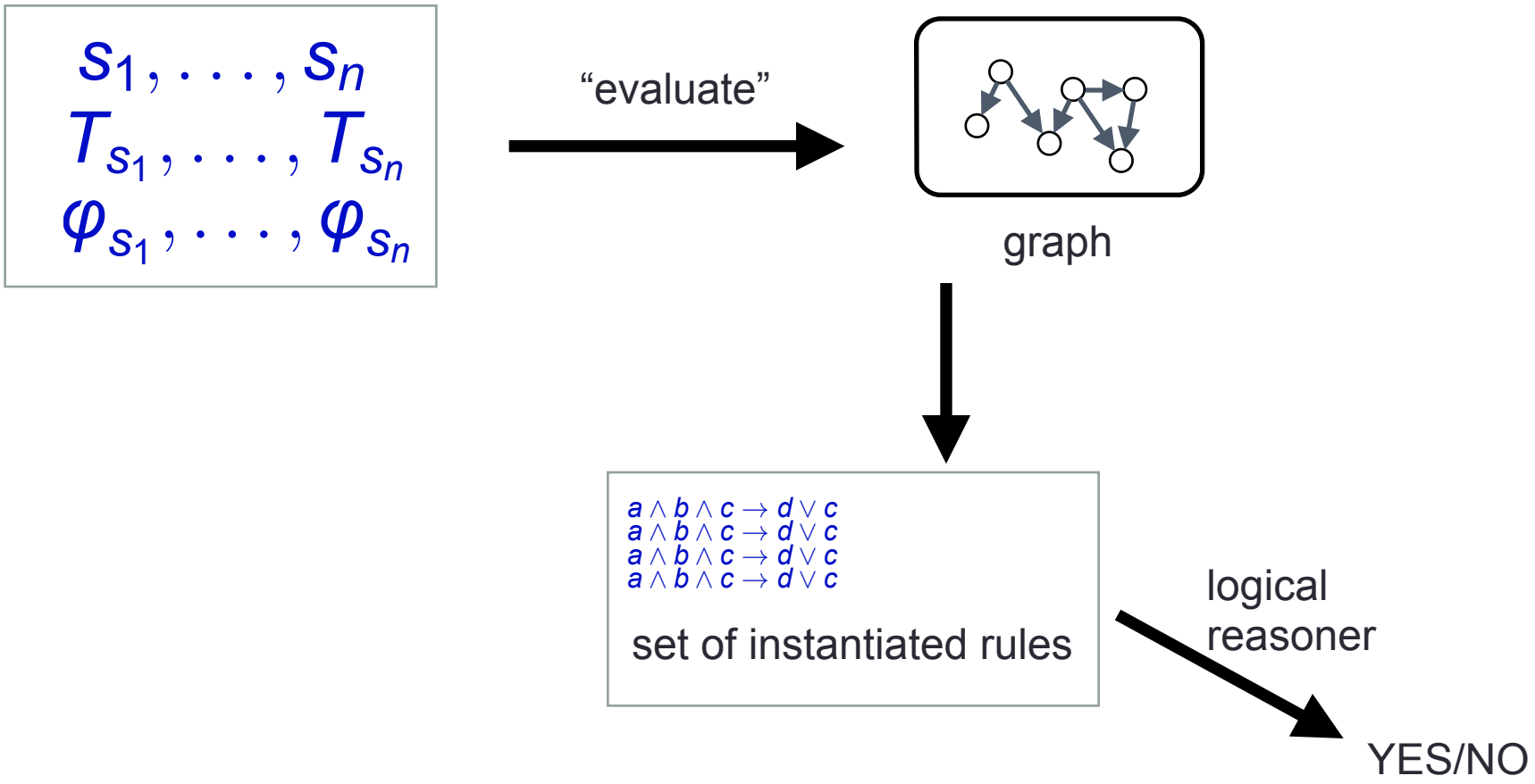
$a \wedge b \wedge c \rightarrow d \vee c$
$a \wedge b \wedge c \rightarrow d \vee c$
$a \wedge b \wedge c \rightarrow d \vee c$
$a \wedge b \wedge c \rightarrow d \vee c$

set of instantiated rules

# Validation … as in ASP



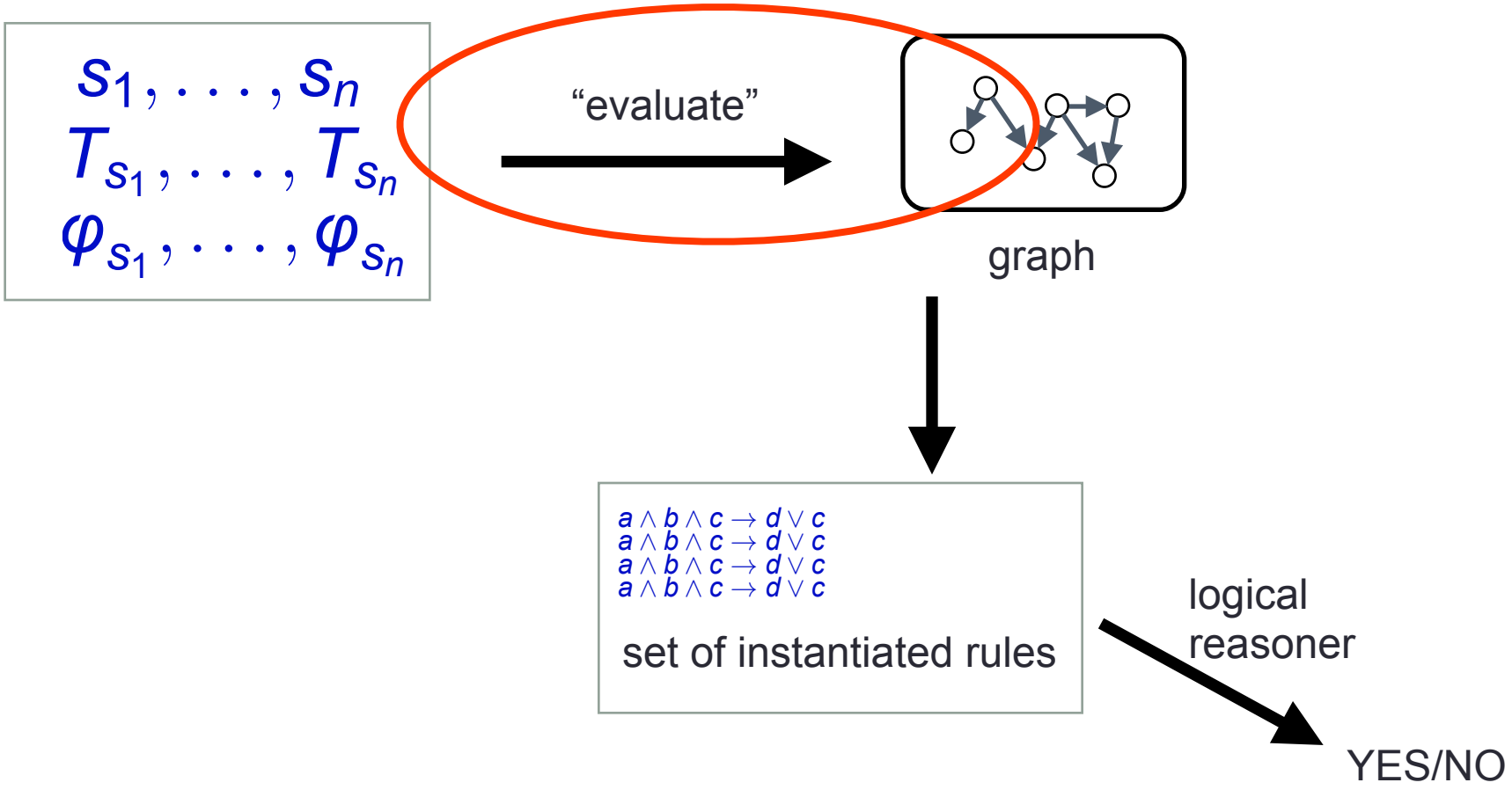
# Validation … as in ASP




For classes of schemas
we know validation is in PTIME

This runs in PTIME

# Approach introduces design considerations



# Approach introduces design considerations

$\mathcal{L}_{\text{type}}$

language to express shapes

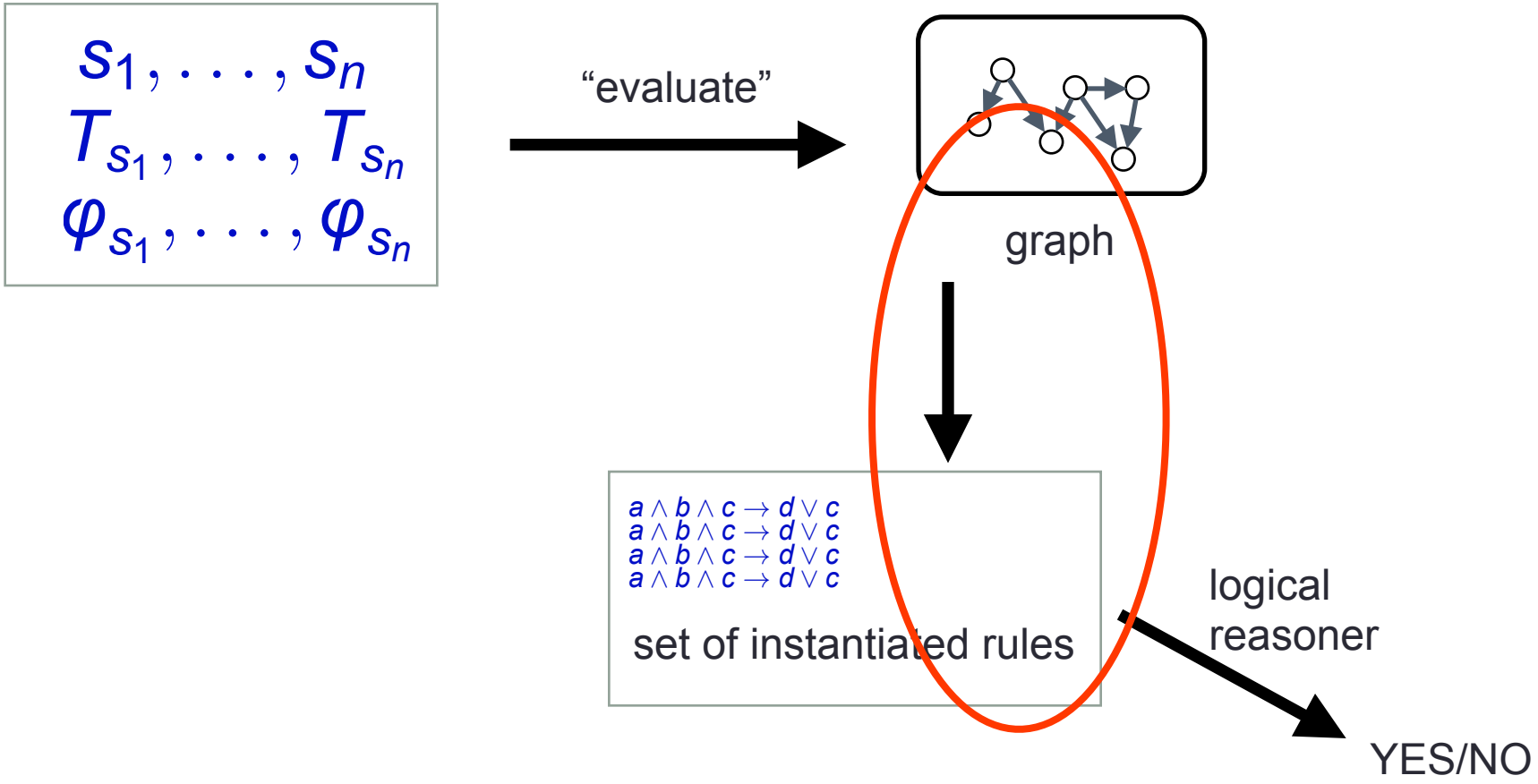
fast queries!

$\mathcal{L}_{\text{const}}$

language to express constraints

fast queries!

# Approach introduces design considerations



# Approach introduces design considerations

$\mathcal{L}_{\text{type}}$

language to express shapes

fast queries!

$\mathcal{L}_{\text{const}}$

language to express constraints

fast queries!

queries without many answers

# Approach introduces design considerations

$\mathcal{L}_{type}$

language to express shapes

fast queries!

$\mathcal{L}_{const}$

language to express constraints

fast queries!

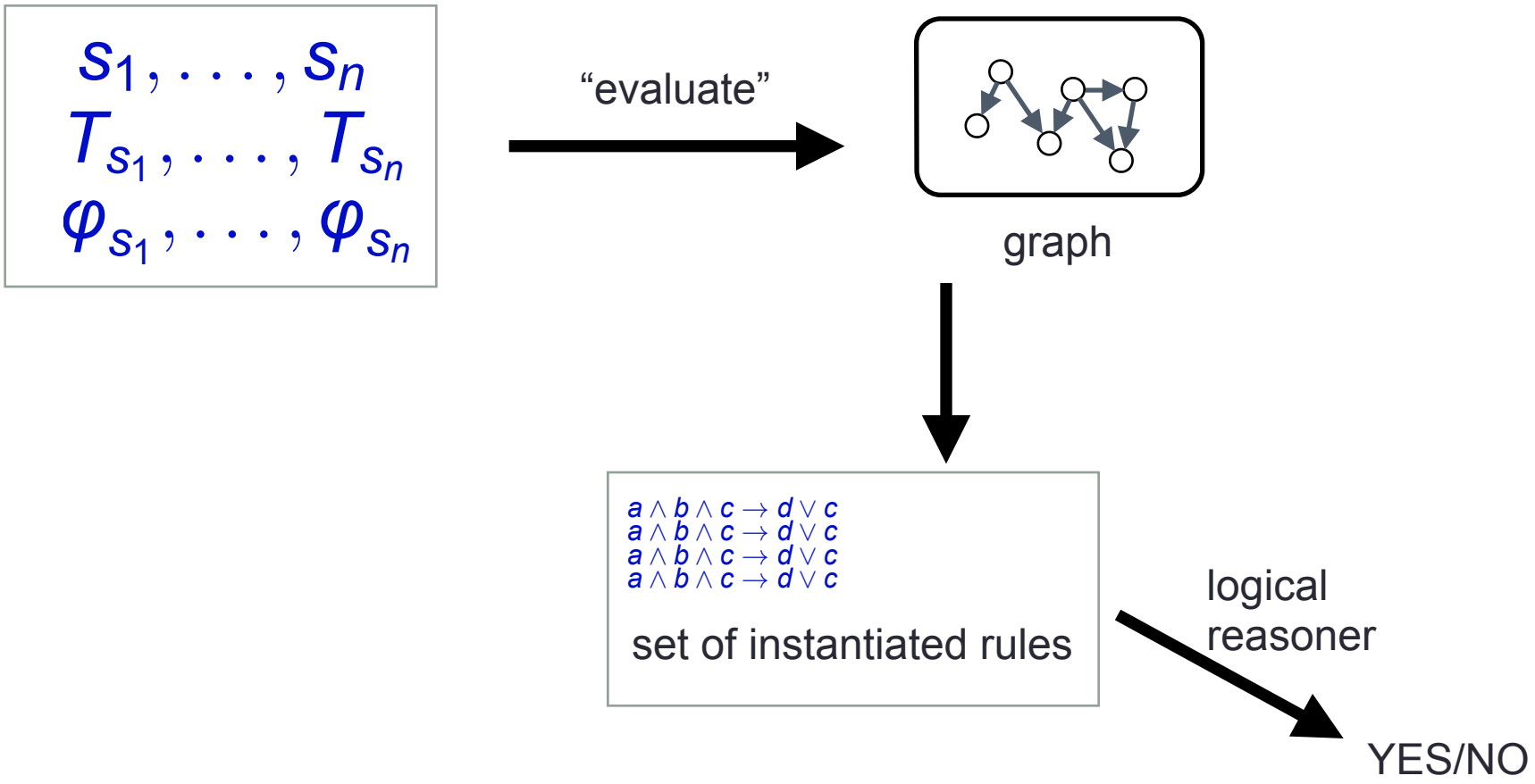
queries without many answers

true in SHACL
ShEx (no nesting of *)

# Further / Ongoing Work

Understand what is fast / what is not

# Further / Ongoing Work

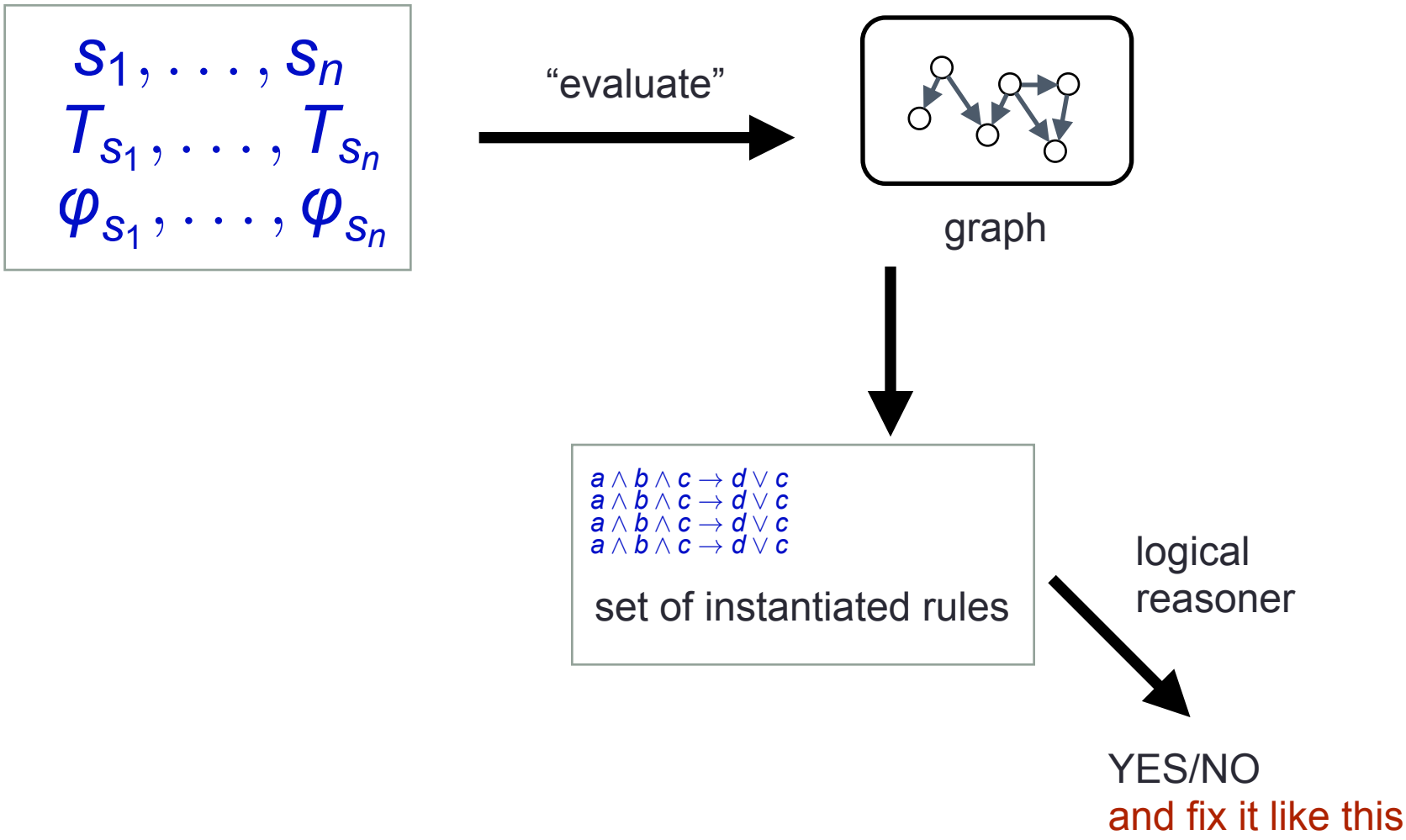


# Further / Ongoing Work

Understand what is fast / what is not

Explanations

# Further / Ongoing Work



# Further / Ongoing Work

Understand what is fast / what is not

Explanations

Property Graphs? Text? CSV?

# Further / Ongoing Work

Understand what is fast / what is not

More theory (satisfiability, data exchange, …)

Explanations

Property Graphs? Text? CSV?

# Further / Ongoing Work

Understand what is fast / what is not

More theory (satisfiability, data exchange, …)

Explanations

Schema design

Property Graphs? Text? CSV?

# Further / Ongoing Work

Understand what is fast / what is not

More theory (satisfiability, data exchange, …)

Explanations

Query Optimisation

Schema design

Property Graphs? Text? CSV?

Learn schemas

Millennium Institute
Foundational
Research on Data

# Schemas for graphs and other forms of semi-structured data

Juan L. Reutter

PUC Chile

. S. Staworko, I. Boneva, J. E. Labra Gayo, S. Hym, E. G. Prud'hommeaux, and H. Solbrig. Complexity and Expressiveness of ShEx for RDF. In ICDT, 2015

. I. Boneva, J. E. L. Gayo, and E. G. Prud'hommeaux. Semantics and Validation of Shapes Schemas for RDF. In ISWC, 2017.

. J. Corman, J. L. Reutter, and O. Savkovic. A tractable notion of stratification for SHACL. In ISWC, 2018.

Felipe Pezoa, Juan L. Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of JSON schema. In WWW, 2016.