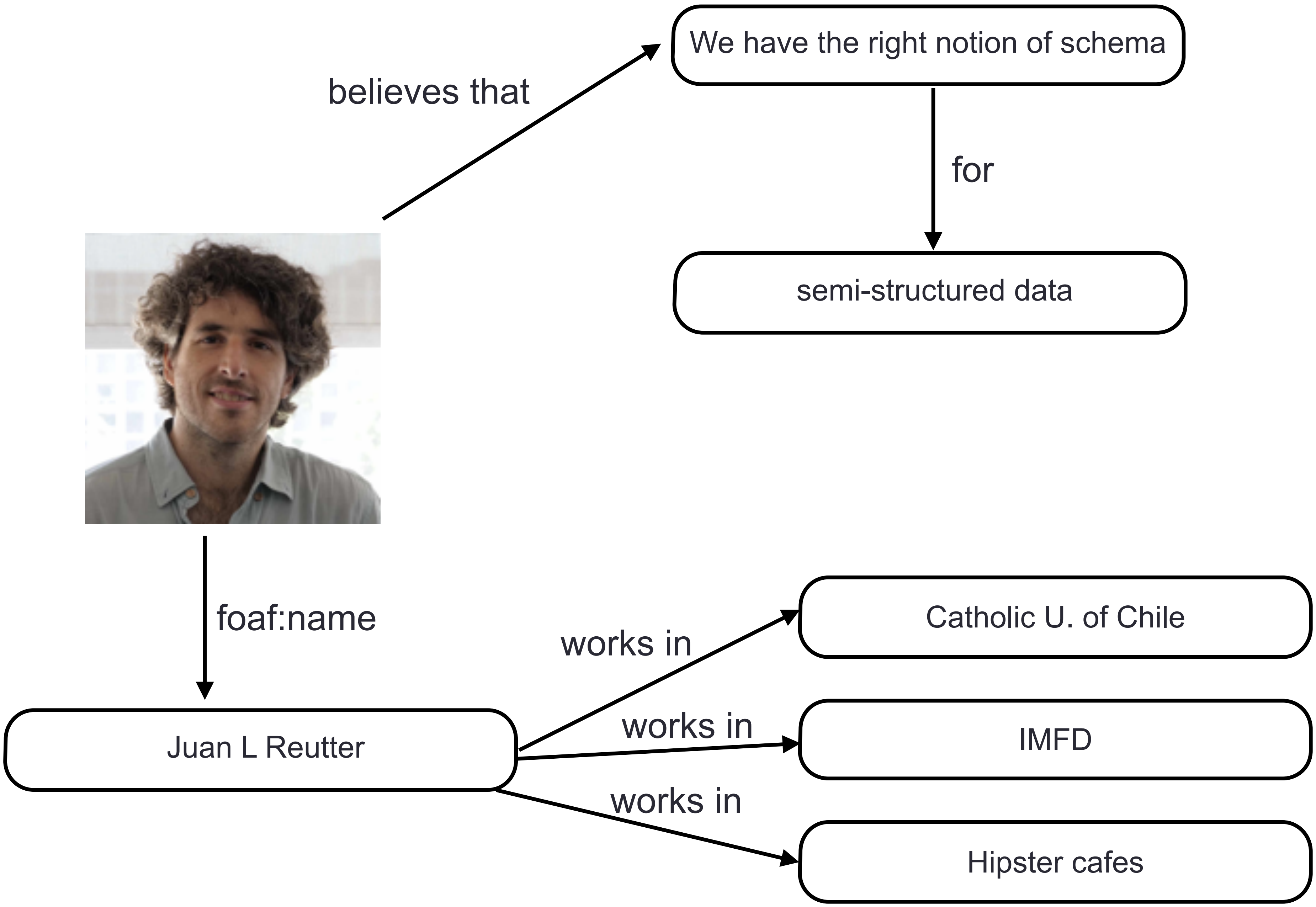
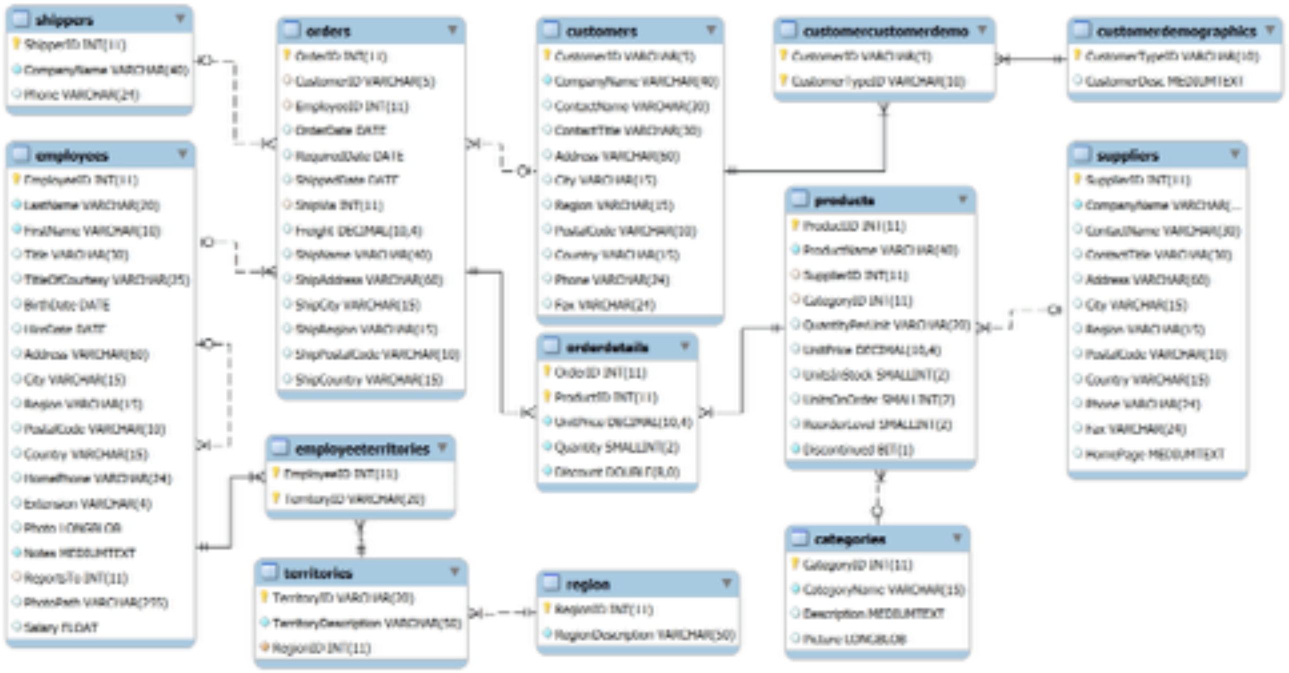

believes that
We have the right notion of schema
for
semi-structured data
foaf:name
Juan L Reutter
works in
Catholic U. of Chile
works in
IMFD
works in
Hipster cafes

Schema

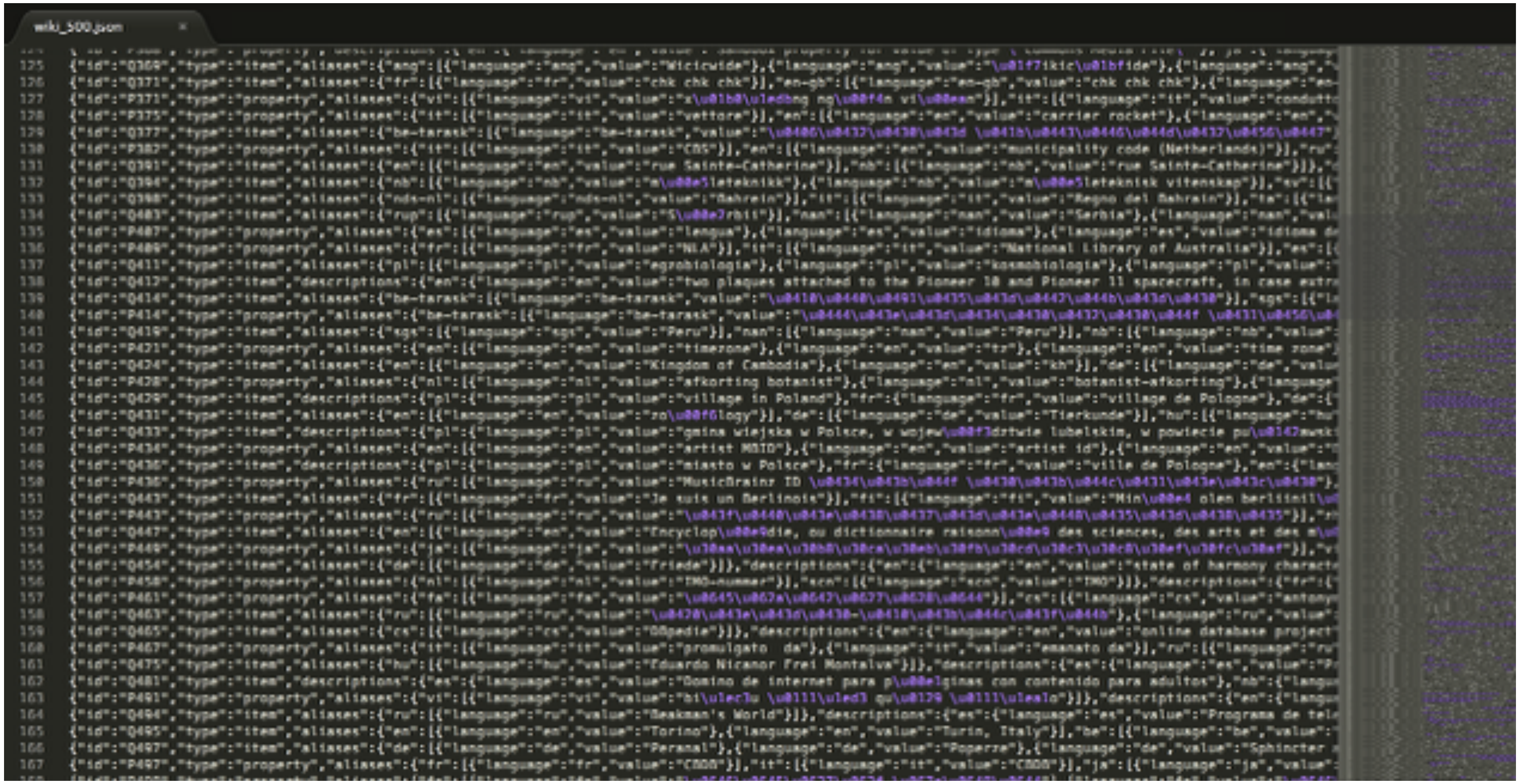
Schema

“the home where the data lives in”

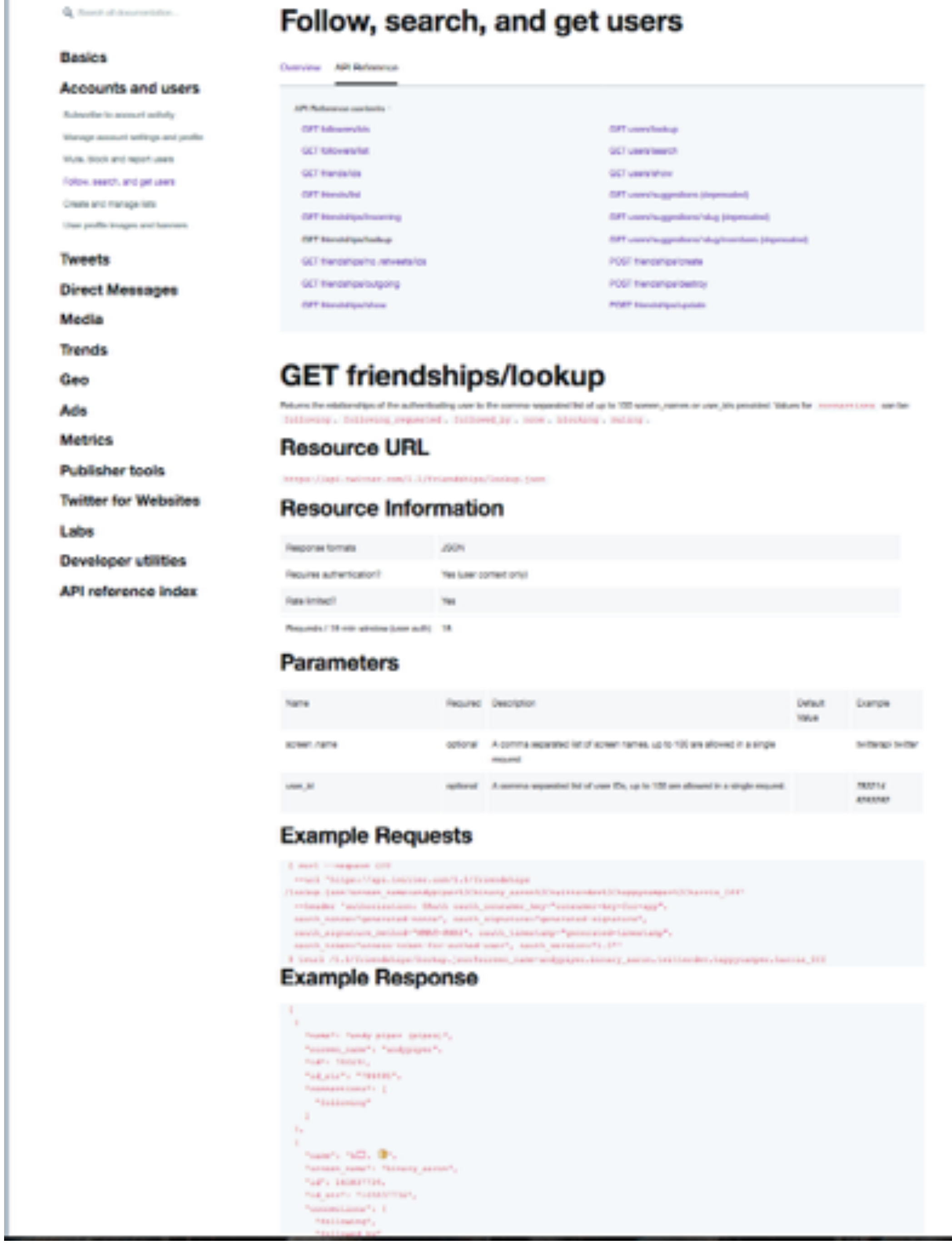


Semistructured data

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



```
{"id":"Q369","type":"item","aliases":{"ang":[{"language":"ang","value":"Wicicwide"},{"language":"ang","value":"\u01f7ikic\u01bfide"},{"language":"ang","
{"id":"Q371","type":"item","aliases":{"fr":[{"language":"fr","value":"chk chk chk"}],"en-gb":[{"language":"en-gb","value":"chk chk chk"},{"language":"en-
{"id":"P371","type":"property","aliases":{"vi":[{"language":"vi","value":"x\u01b0\u1edbng ng\u00f4n vi\u00ean"}],"it":[{"language":"it","value":"condutt
{"id":"P375","type":"property","aliases":{"it":[{"language":"it","value":"vettore"}],"en":[{"language":"en","value":"carrier rocket"},{"language":"en","
{"id":"Q377","type":"item","aliases":{"be-tarask":[{"language":"be-tarask","value":"\u0406\u0432\u0430\u043d \u041b\u0443\u0446\u044d\u0432\u0456\u0447"
{"id":"P382","type":"property","aliases":{"it":[{"language":"it","value":"CBS"}],"en":[{"language":"en","value":"municipality code (Netherlands)"}],"ru"
{"id":"Q391","type":"item","aliases":{"en":[{"language":"en","value":"rue Sainte-Catherine"}],"nb":[{"language":"nb","value":"rue Sainte-Catherine"}]},"
{"id":"Q394","type":"item","aliases":{"nb":[{"language":"nb","value":"m\u00e5leteknikk"},{"language":"nb","value":"m\u00e5leteknisk vitenskap"}],"sv":[{
{"id":"Q398","type":"item","aliases":{"nds-nl":[{"language":"nds-nl","value":"Bahrein"}],"it":[{"language":"it","value":"Regno del Bahrain"}],"fa":[{"la
{"id":"Q403","type":"item","aliases":{"rup":[{"language":"rup","value":"S\u00e2rbii"}],"nan":[{"language":"nan","value":"Serbia"},{"language":"nan","val
{"id":"P407","type":"property","aliases":{"es":[{"language":"es","value":"lengua"},{"language":"es","value":"idioma"},{"language":"es","value":"idioma d
{"id":"P409","type":"property","aliases":{"fr":[{"language":"fr","value":"NLA"}],"it":[{"language":"it","value":"National Library of Australia"}],"es":[{
{"id":"Q411","type":"item","aliases":{"pl":[{"language":"pl","value":"egzobiologia"},{"language":"pl","value":"kosmobiologia"},{"language":"pl","value":
{"id":"Q412","type":"item","descriptions":{"en":{"language":"en","value":"two plaques attached to the Pioneer 10 and Pioneer 11 spacecraft, in case extr
{"id":"Q414","type":"item","aliases":{"be-tarask":[{"language":"be-tarask","value":"\u0410\u0440\u0433\u0435\u043d\u0442\u044b\u043d\u0430"}],"sgs":[{"l
{"id":"P414","type":"property","aliases":{"be-tarask":[{"language":"be-tarask","value":"\u0444\u043e\u043d\u0434\u0430\u0432\u0430\u044f \u0431\u0456\u0
{"id":"Q419","type":"item","aliases":{"sgs":[{"language":"sgs","value":"Peru"}],"nan":[{"language":"nan","value":"Peru"}],"nb":[{"language":"nb","value":
{"id":"P421","type":"property","aliases":{"en":[{"language":"en","value":"timezone"},{"language":"en","value":"tz"},{"language":"en","value":"time zone"}
{"id":"Q424","type":"item","aliases":{"en":[{"language":"en","value":"Kingdom of Cambodia"},{"language":"en","value":"kh"}],"de":[{"language":"de","valu
{"id":"P428","type":"property","aliases":{"nl":[{"language":"nl","value":"afkorting botanist"},{"language":"nl","value":"botanist-afkorting"},{"language
{"id":"Q429","type":"item","descriptions":{"pl":{"language":"pl","value":"village in Poland"},"fr":{"language":"fr","value":"village de Pologne"},"de":{"
{"id":"Q431","type":"item","aliases":{"en":[{"language":"en","value":"zo\u00f6logy"}],"de":[{"language":"de","value":"Tierkunde"}],"hu":[{"language":"hu
{"id":"Q433","type":"item","descriptions":{"pl":{"language":"pl","value":"gmina wiejska w Polsce, w wojew\u00f3dztwie lubelskim, w powiecie pu\u0142awsk
{"id":"P434","type":"property","aliases":{"en":[{"language":"en","value":"artist MBID"},{"language":"en","value":"artist id"},{"language":"en","value":"
{"id":"Q436","type":"item","descriptions":{"pl":{"language":"pl","value":"miasto w Polsce"},"fr":{"language":"fr","value":"ville de Pologne"},"en":{"lang
{"id":"P436","type":"property","aliases":{"ru":[{"language":"ru","value":"MusicBrainz ID \u0434\u043b\u044f \u0430\u043b\u044c\u0431\u043e\u043c\u0430"},
{"id":"Q443","type":"item","aliases":{"fr":[{"language":"fr","value":"Je suis un Berlinois"}],"fi":[{"language":"fi","value":"Min\u00e4 olen berliinil\u0
{"id":"P443","type":"property","aliases":{"ru":[{"language":"ru","value":"\u043f\u0440\u043e\u0438\u0437\u043d\u043e\u0448\u0435\u043d\u0438\u0435"}],"r
{"id":"Q447","type":"item","aliases":{"en":[{"language":"en","value":"Encyclop\u00e9die, ou dictionnaire raisonn\u00e9 des sciences, des arts et des m\u0
{"id":"P449","type":"property","aliases":{"ja":[{"language":"ja","value":"\u30aa\u30ea\u30b8\u30ca\u30eb\u30fb\u30cd\u30c3\u30c8\u30ef\u30fc\u30af"}],"v
{"id":"Q454","type":"item","aliases":{"de":[{"language":"de","value":"Friede"}]},"descriptions":{"en":{"language":"en","value":"state of harmony charact
{"id":"P458","type":"property","aliases":{"nl":[{"language":"nl","value":"IMO-nummer"}],"scn":[{"language":"scn","value":"IMO"}]},"descriptions":{"fr":{
{"id":"P461","type":"property","aliases":{"fa":[{"language":"fa","value":"\u0645\u062a\u0642\u0627\u0628\u0644"}],"cs":[{"language":"cs","value":"antony
{"id":"Q463","type":"item","aliases":{"ru":[{"language":"ru","value":"\u0420\u043e\u043d\u0430-\u0410\u043b\u044c\u043f\u044b"},{"language":"ru","value":
{"id":"Q465","type":"item","aliases":{"cs":[{"language":"cs","value":"DBpedie"}]},"descriptions":{"en":{"language":"en","value":"online database project
{"id":"P467","type":"property","aliases":{"it":[{"language":"it","value":"promulgato  da"},{"language":"it","value":"emanato da"}],"ru":[{"language":"ru
{"id":"Q475","type":"item","aliases":{"hu":[{"language":"hu","value":"Eduardo Nicanor Frei Montalva"}]},"descriptions":{"es":{"language":"es","value":"P
{"id":"Q481","type":"item","descriptions":{"es":{"language":"es","value":"Dominio de internet para p\u00e1ginas con contenido para adultos"},"nb":{"langu
{"id":"P491","type":"property","aliases":{"vi":[{"language":"vi","value":"bi\u1ec3u \u0111\u1ed3 qu\u0129 \u0111\u1ea1o"}]},"descriptions":{"en":{"langu
{"id":"Q494","type":"item","aliases":{"ru":[{"language":"ru","value":"Beakman's World"}]},"descriptions":{"es":{"language":"es","value":"Programa de tel
{"id":"Q495","type":"item","aliases":{"en":[{"language":"en","value":"Torino"},{"language":"en","value":"Turin, Italy"}],"be":[{"language":"be","value":
{"id":"Q497","type":"item","aliases":{"de":[{"language":"de","value":"Peranal"},{"language":"de","value":"Poperze"},{"language":"de","value":"Sphincter 
{"id":"P497","type":"property","aliases":{"fr":[{"language":"fr","value":"CBDB"}],"it":[{"language":"it","value":"CBDB"}],"ja":[{"language":"ja","value":
```



Search all documentation...

Basics

Accounts and users

- Subscribe to account activity
- Manage account settings and profile
- Mute, block and report users
- Follow, search, and get users
- Create and manage lists
- User profile images and banners

Tweets

Direct Messages

Media

Trends

Geo

Ads

Metrics

Publisher tools

Twitter for Websites

Labs

Developer utilities

API reference index

Follow, search, and get users

Overview API Reference

API Reference contents

GET followers/ids	GET users/lookup
GET followers/list	GET users/search
GET friends/ids	GET users/show
GET friends/list	GET users/suggestions (deprecated)
GET friendships/incoming	GET users/suggestions/:slug (deprecated)
GET friendships/lookup	GET users/suggestions/:slug/members (deprecated)
GET friendships/no_retweets/ids	POST friendships/create
GET friendships/outgoing	POST friendships/destroy
GET friendships/show	POST friendships/update

GET friendships/lookup

Returns the relationships of the authenticating user to the comma-separated list of up to 100 screen_names or user_ids provided. Values for `connections` can be: `following`, `following_requested`, `followed_by`, `none`, `blocking`, `muting`.

Resource URL

`https://api.twitter.com/1.1/friendships/lookup.json`

Resource Information

Response formats	JSON
Requires authentication?	Yes (user context only)
Rate limited?	Yes
Requests / 15-min window (user auth)	15

Parameters

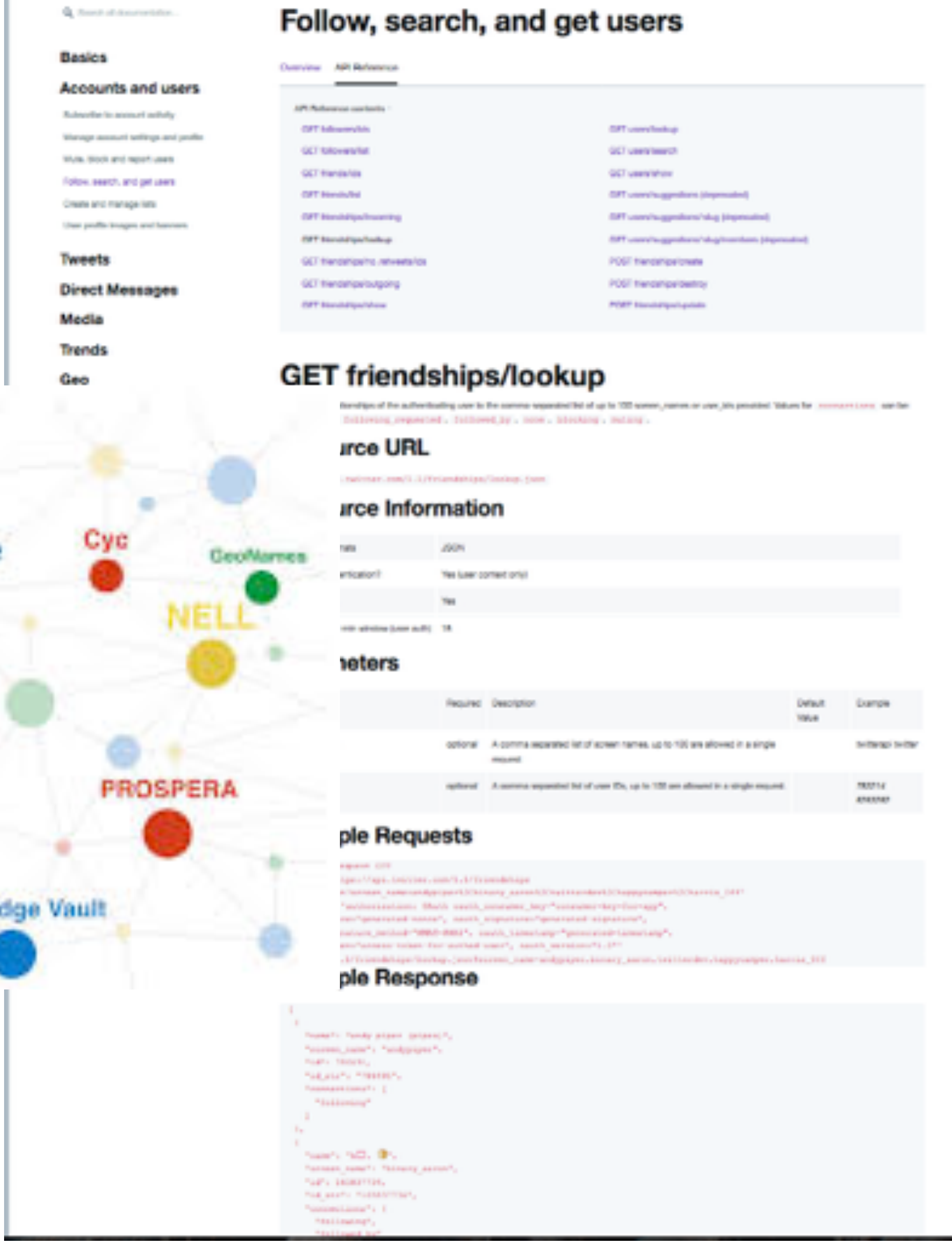
Name	Required	Description	Default Value	Example
screen_name	optional	A comma separated list of screen names, up to 100 are allowed in a single request.		twitterapi,twitter
user_id	optional	A comma separated list of user IDs, up to 100 are allowed in a single request.		783214,6253282

Example Requests

```
$ curl --request GET
  --url 'https://api.twitter.com/1.1/friendships
/lookup.json?screen_name=andypiper%2Cbinary_aaron%2Ctwitterdev%2Chappycamper%2Ctwitter_ptr'
  --header 'authorization: OAuth oauth_consumer_key="consumer-key-for-app",
  oauth_nonce="generated-nonce", oauth_signature="generated-signature",
  oauth_signature_method="HMAC-SHA1", oauth_timestamp="generated-timestamp",
  oauth_token="access-token-for-authed-user", oauth_version="1.0"'
$ twurl /1.1/friendships/lookup.json?screen_name=andypiper,binary_aaron,twitterdev,happycamper,twitter_ptr
```

Example Response

```
[
 {
   "name": "andy piper (pipes)",
   "screen_name": "andypiper",
   "id": 786491,
   "id_str": "786491",
   "connections": [
     "following"
   ]
 },
 {
   "name": "[illegible]",
   "screen_name": "binary_aaron",
   "id": 165837734,
   "id_str": "165837734",
   "connections": [
     "following",
     "followed_by"
```



Search all documentation...

Basics

Accounts and users

Subscribe to account activity

Manage account settings and profile

Mute, block and report users

Follow, search, and get users

Create and manage lists

User profile images and banners

Tweets

Direct Messages

Media

Trends

Geo

Follow, search, and get users

Overview API Reference

API Reference contents

GET followers/ids	GET users/lookup
GET followers/list	GET users/search
GET friends/ids	GET users/show
GET friends/list	GET users/suggestions (deprecated)
GET friendships/incoming	GET users/suggestions/:slug (deprecated)
GET friendships/lookup	GET users/suggestions/:slug/members (deprecated)
GET friendships/no_retweets/ids	POST friendships/create
GET friendships/outgoing	POST friendships/destroy
GET friendships/show	POST friendships/update

GET friendships/lookup

[illegible] of the authenticating user to the comma-separated list of up to 100 screen_names or user_ids provided. Values for connections can be: following_requested, followed_by, none, blocking, muting.

urce URL

.twitter.com/1.1/friendships/lookup.json

urce Information

mats	JSON
entication?	Yes (user context only)
	Yes
min window (user auth)	15

neters

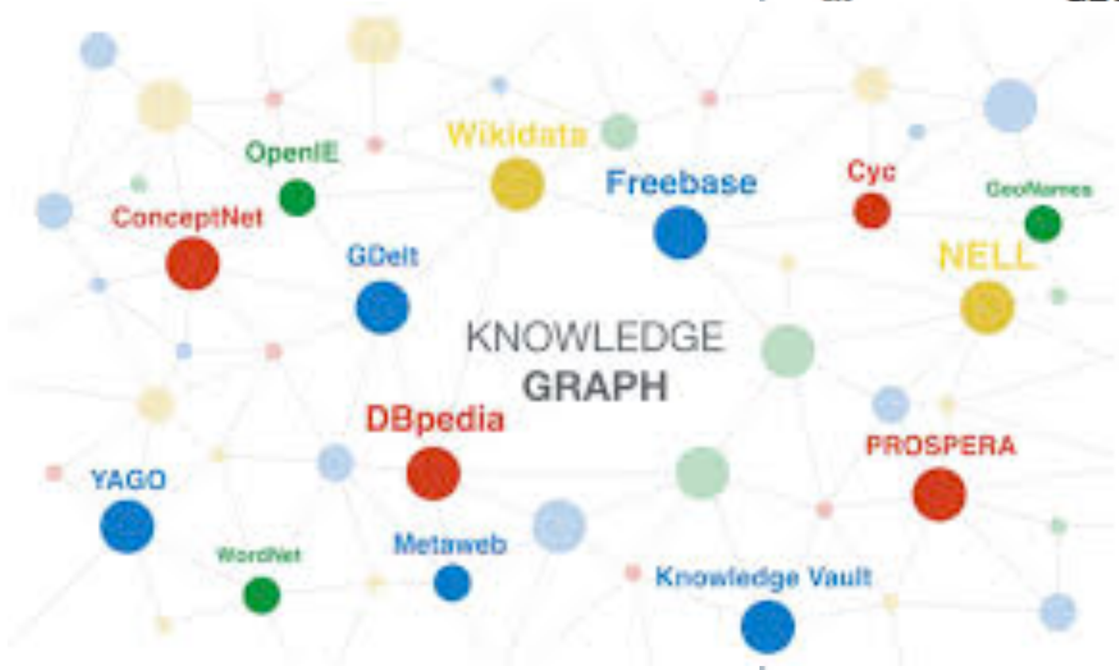
	Required	Description	Default Value	Example
	optional	A comma separated list of screen names, up to 100 are allowed in a single request.		twitterapi,twitter
	optional	A comma separated list of user IDs, up to 100 are allowed in a single request.		783214,6253282

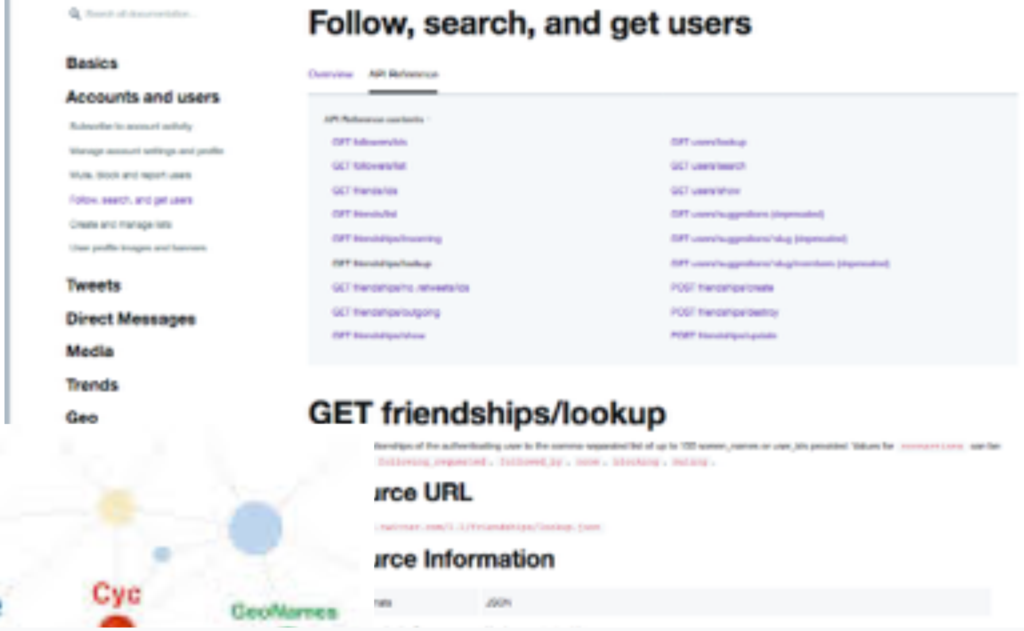
ple Requests

```
[illegible]
```

ple Response

```
[illegible]
```





Basics

Accounts and users

Subscribe to account activity

Manage account settings and profile

Mute, block and report users

Follow, search, and get users

Create and manage lists

User profile images and banners

Tweets

Direct Messages

Media

Trends

Geo

Follow, search, and get users

Overview | API Reference

API Reference contents

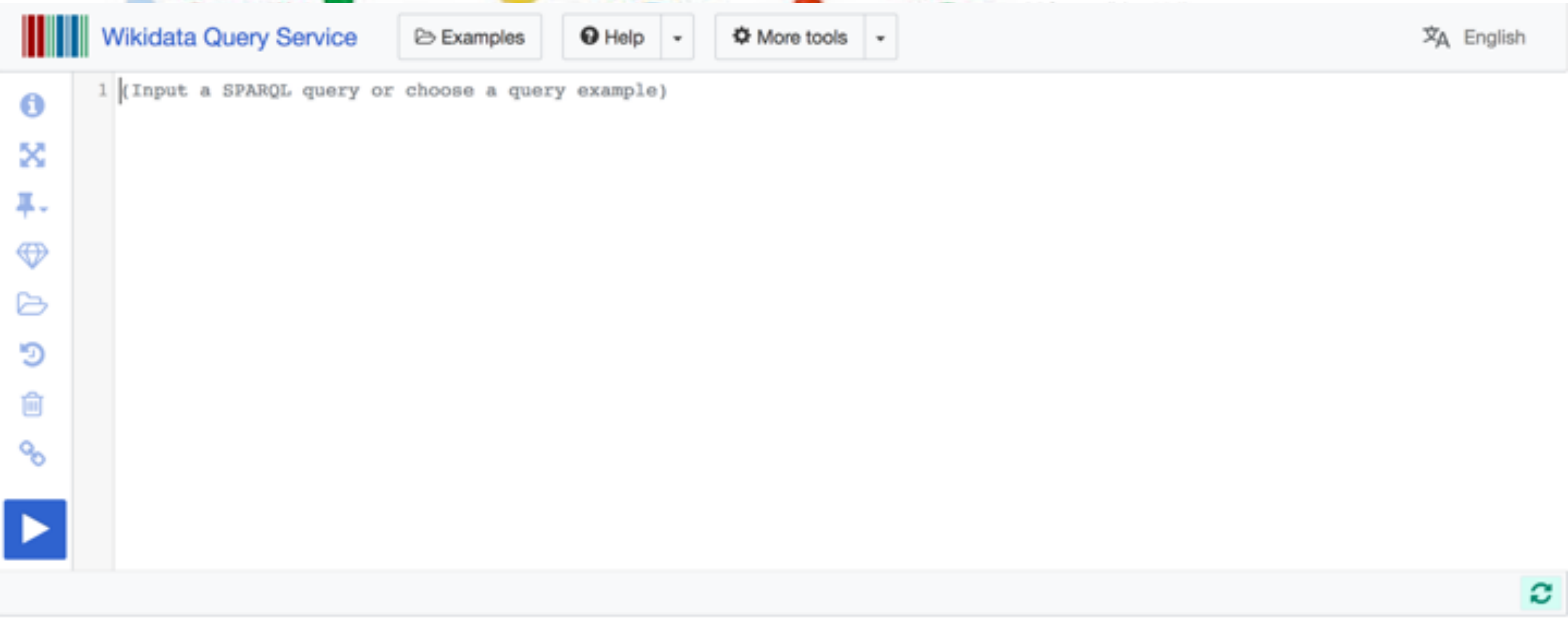
GET followers/ids	GET users/lookup
GET followers/list	GET users/search
GET friends/ids	GET users/show
GET friends/list	GET users/suggestions (deprecated)
GET friendships/incoming	GET users/suggestions/:slug (deprecated)
GET friendships/lookup	GET users/suggestions/:slug/members (deprecated)
GET friendships/no_retweets/ids	POST friendships/create
GET friendships/outgoing	POST friendships/destroy
GET friendships/show	POST friendships/update

GET friendships/lookup

[illegible]

urce URL

urce Information



```
(Input a SPARQL query or choose a query example)
```

Schema in semistructured data

Information about:

what is in the data
how to query it
systems can use it

This talk => Shape-based schemas

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

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

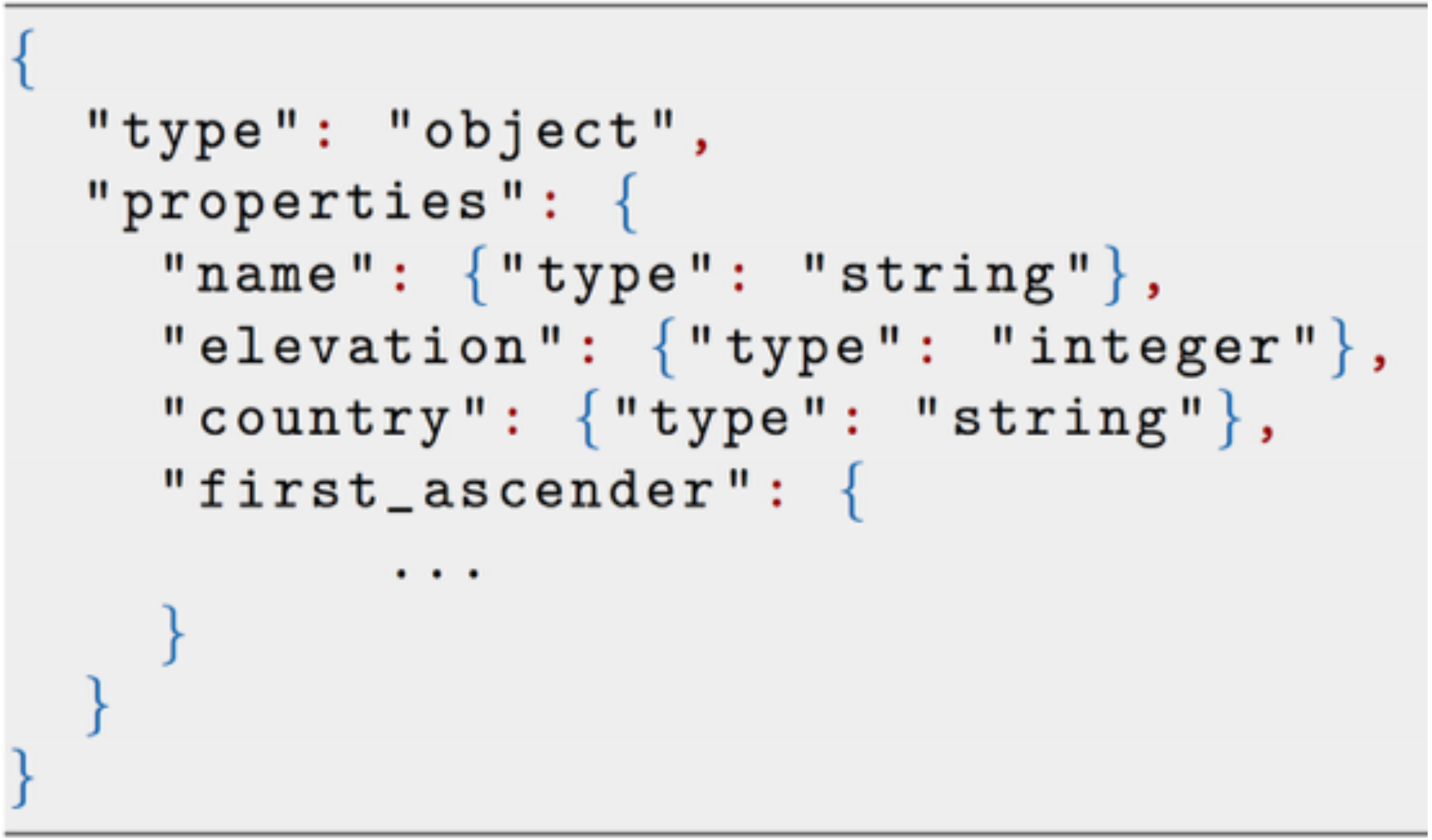
$T(x)$ Answers of this query must be of a shape

$\varphi(x)$ 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

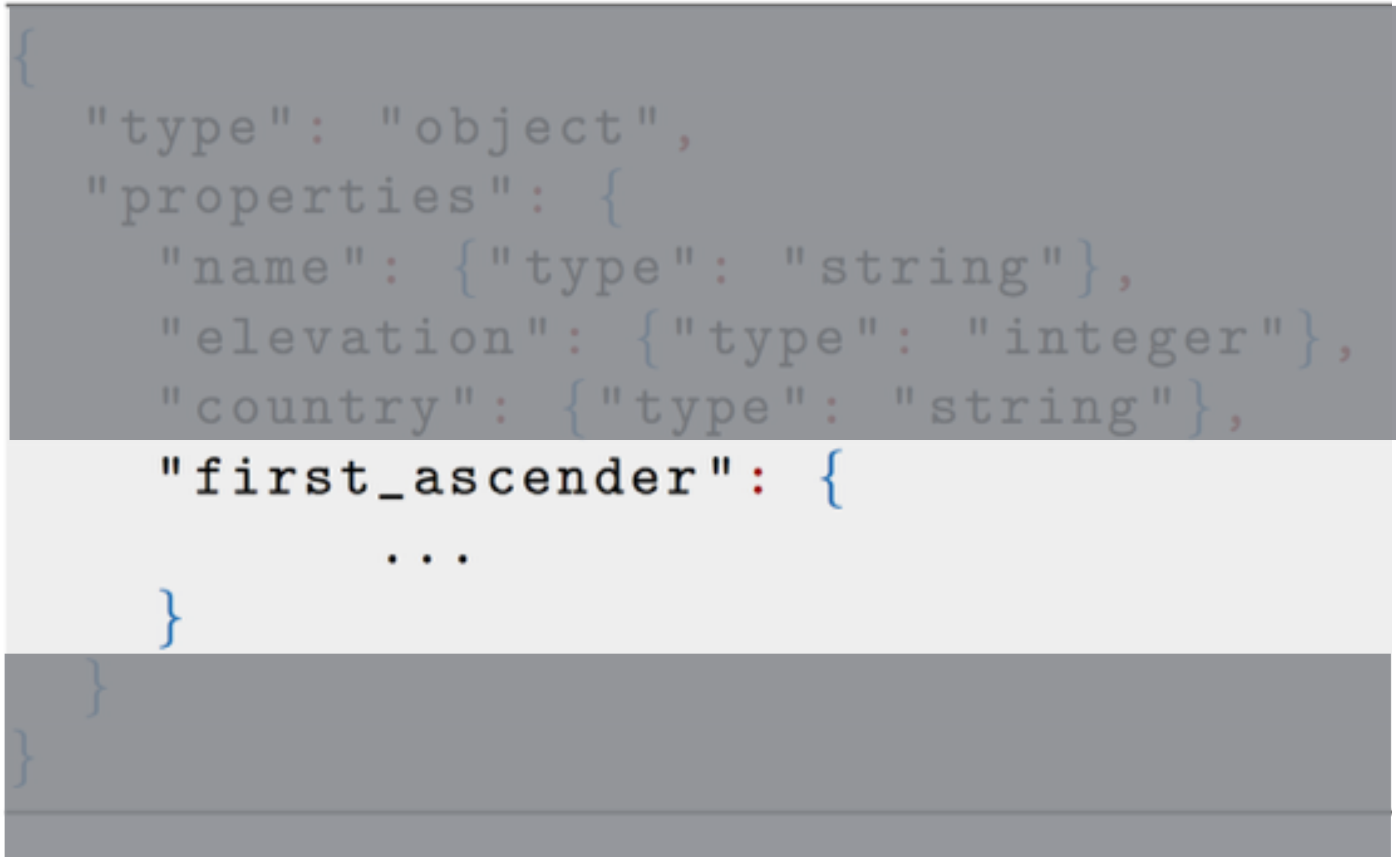
$\mathcal{L}_{\text{type}}$ root shape must conform root JSON Schema

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

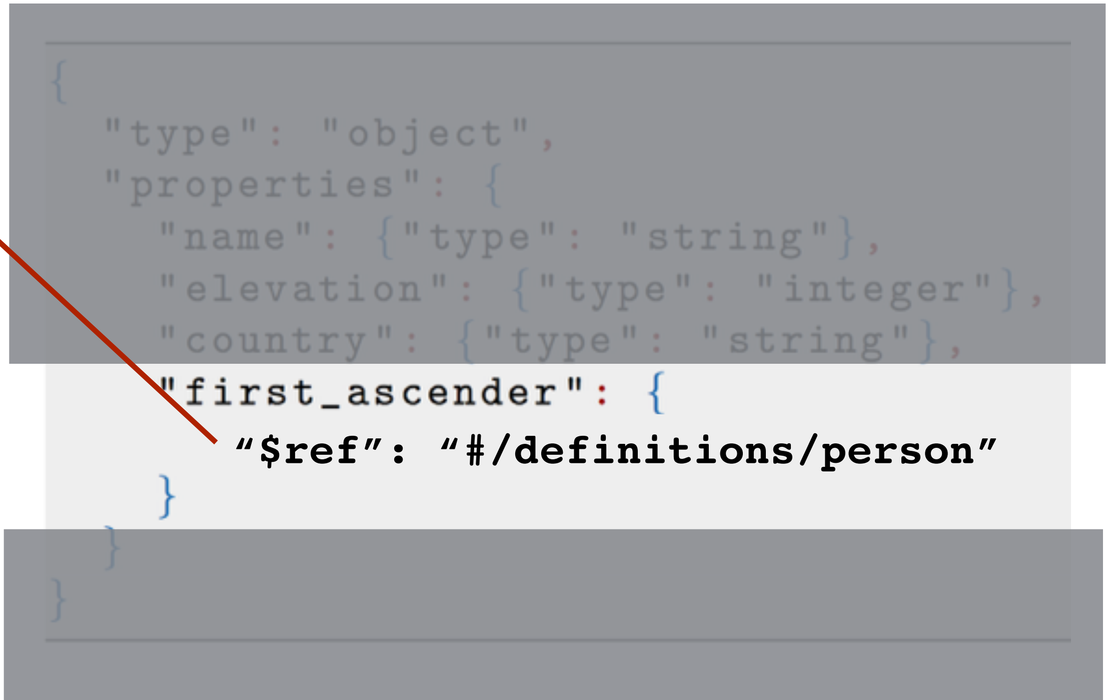


```
{
  "type": "object",
  "properties": {
    "name": {"type": "string"},
    "elevation": {"type": "integer"},
    "country": {"type": "string"},
    "first_ascender": {
         ...
    }
  }
}
```

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”}
          }
     }
}
```



```
{
  "type": "object",
  "properties": {
    "name": {"type": "string"},
    "elevation": {"type": "integer"},
    "country": {"type": "string"},
    "first_ascender": {
       “$ref”: “#/definitions/person”
    }
  }
}
```

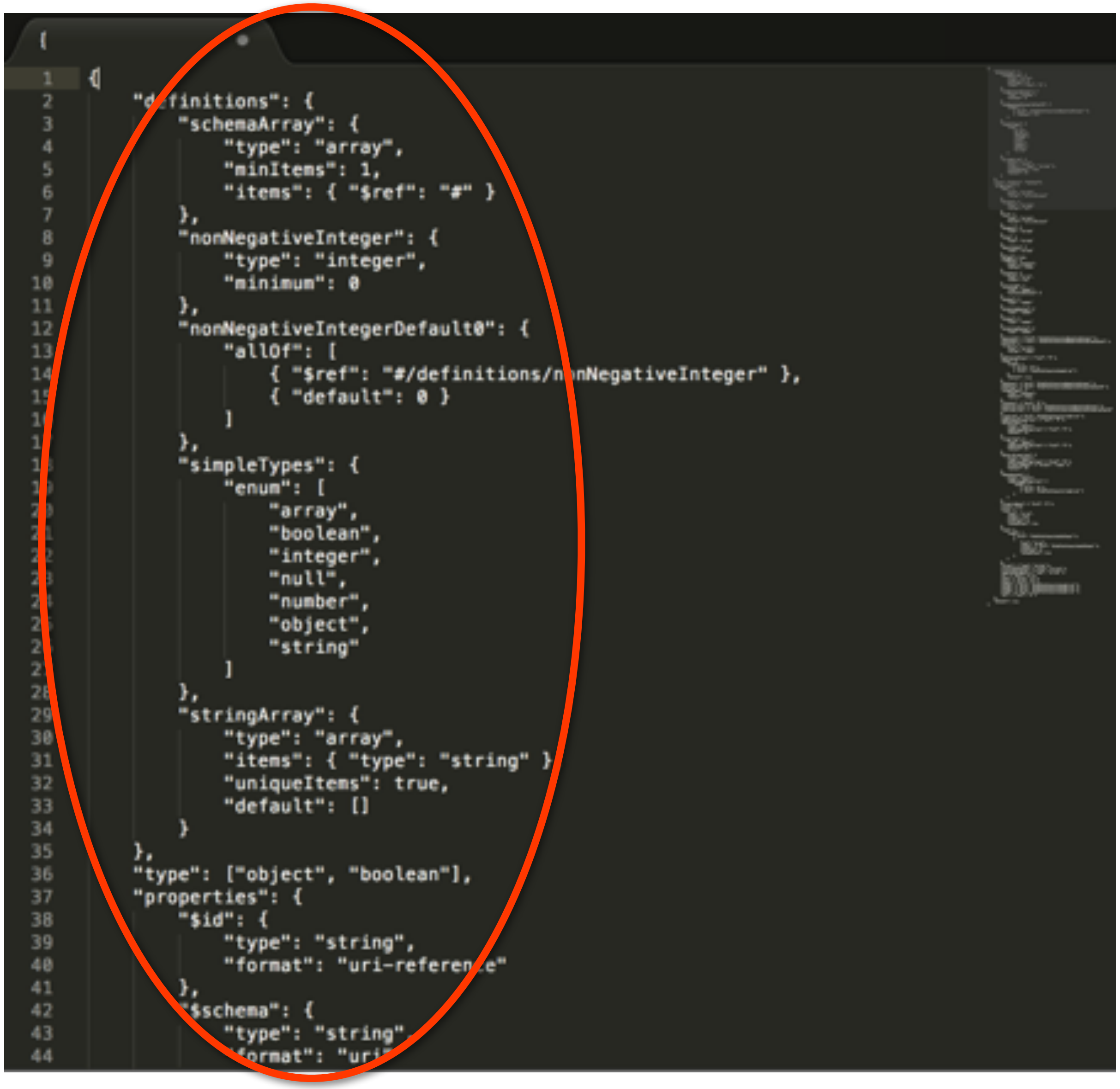
JSON Schema

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



```
{
    "definitions": {
        "schemaArray": {
            "type": "array",
            "minItems": 1,
            "items": { "$ref": "#" }
        },
        "nonNegativeInteger": {
            "type": "integer",
            "minimum": 0
        },
        "nonNegativeIntegerDefault0": {
            "allOf": [
                { "$ref": "#/definitions/nonNegativeInteger" },
                { "default": 0 }
            ]
        },
        "simpleTypes": {
            "enum": [
                "array",
                "boolean",
                "integer",
                "null",
                "number",
                "object",
                "string"
            ]
        },
        "stringArray": {
            "type": "array",
            "items": { "type": "string" },
            "uniqueItems": true,
            "default": []
        }
    },
    "type": ["object", "boolean"],
    "properties": {
        "$id": {
            "type": "string",
            "format": "uri-reference"
        },
        "$schema": {
            "type": "string",
            "format": "uri
```

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
     ] ;
```

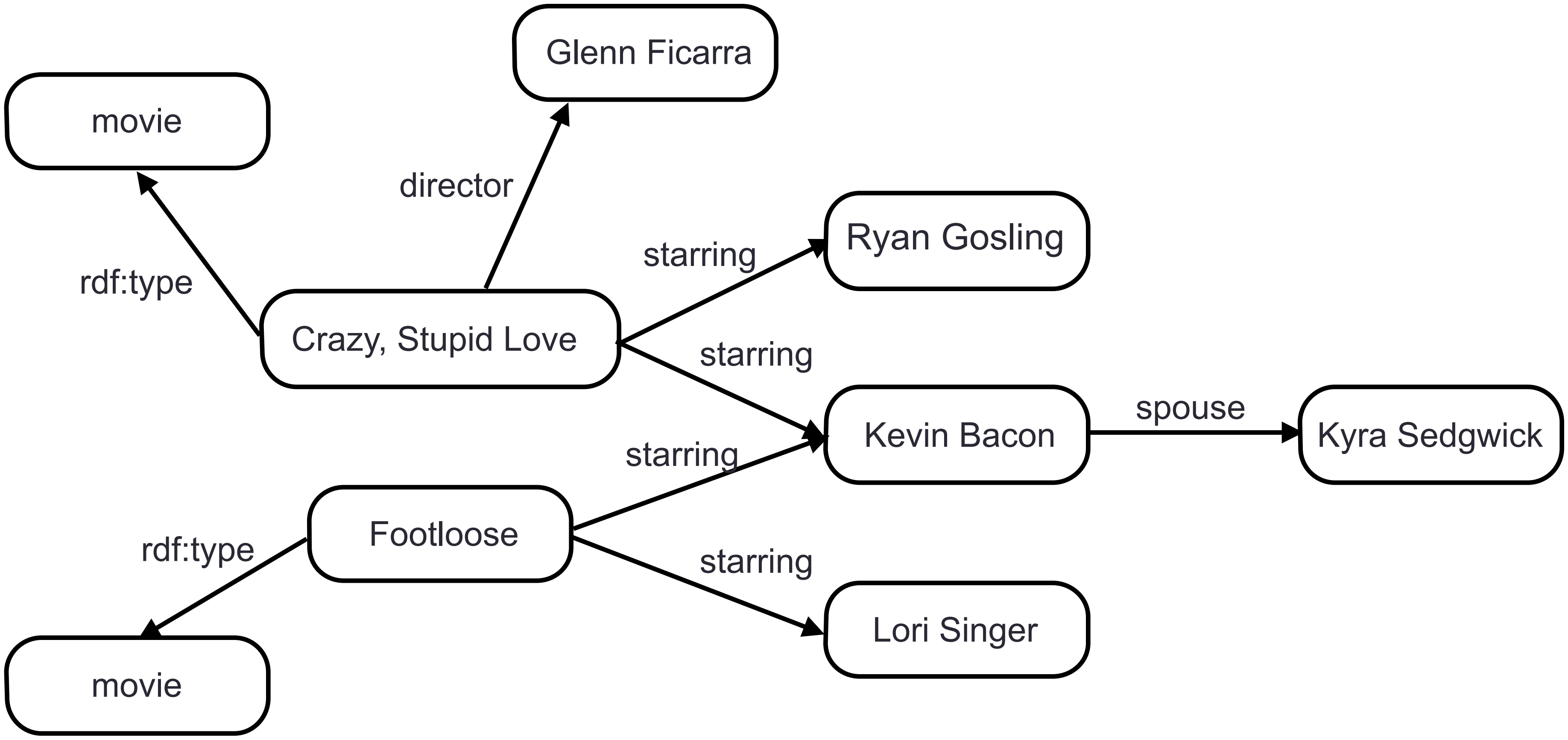
```
:personShape
     a   sh:NodeShape ;
     sh:property [
     sh:path  :spouse ;
     sh:node  :personShape
     ] .
```

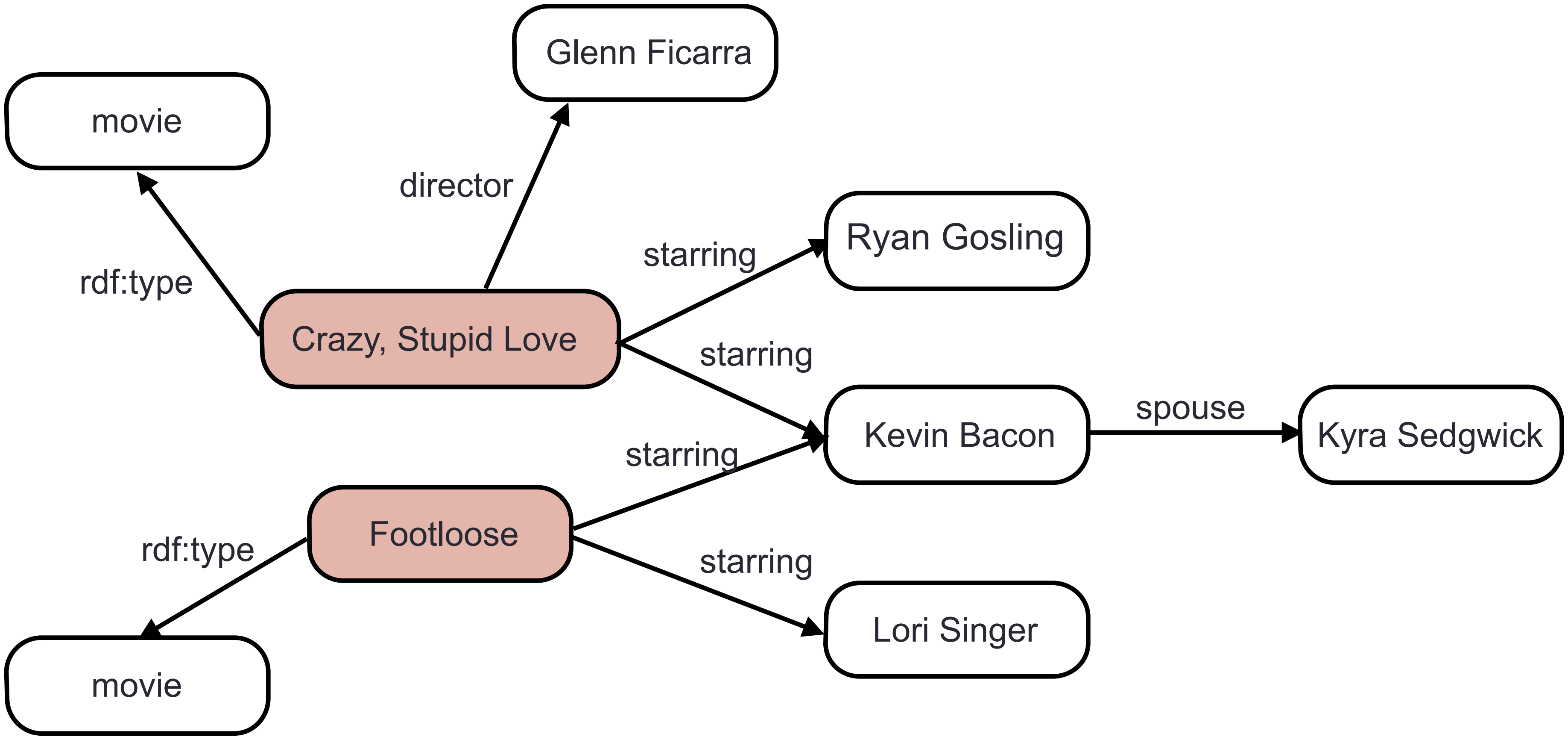
SHACL

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



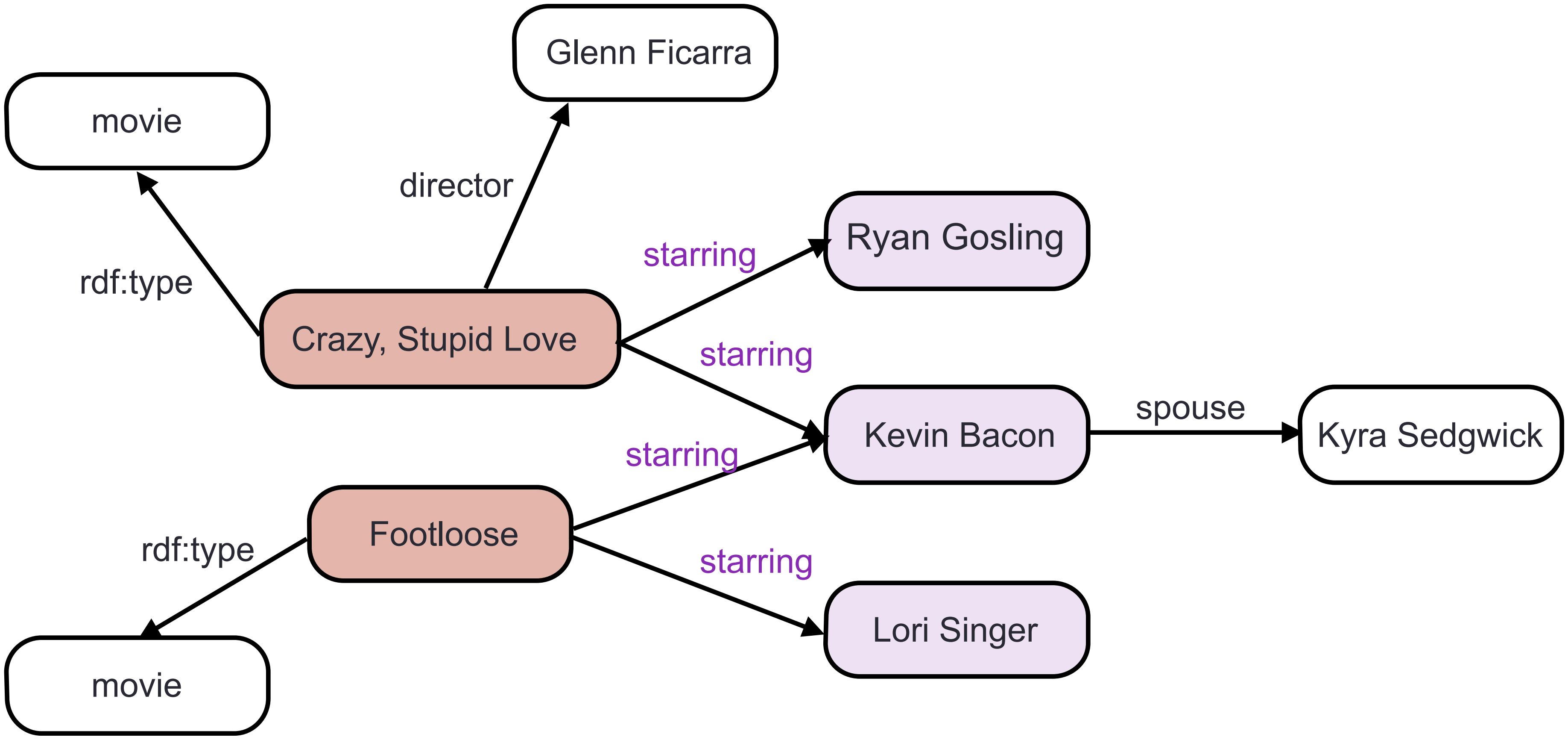
these nodes must conform to :movieShape

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

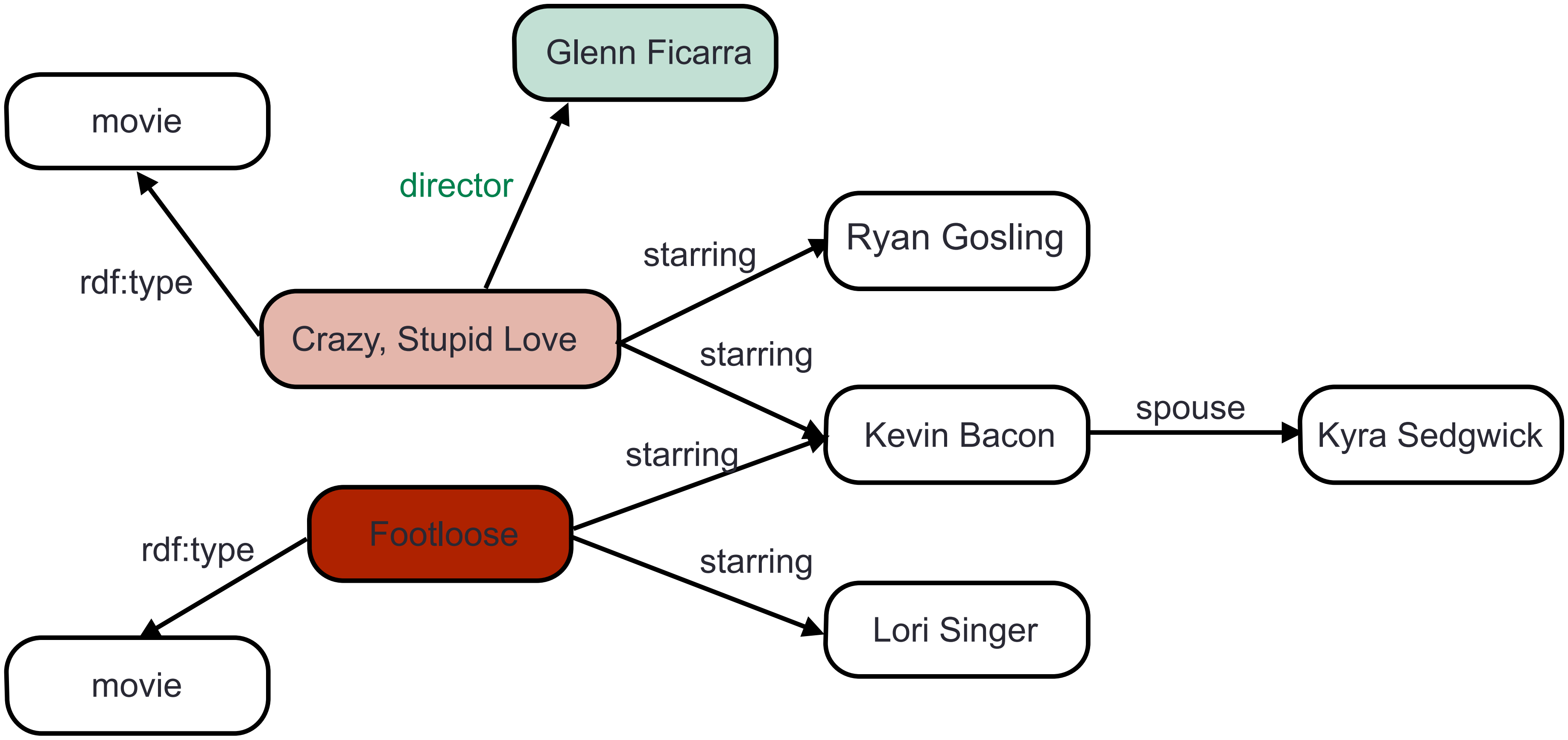
SHACL

```
: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



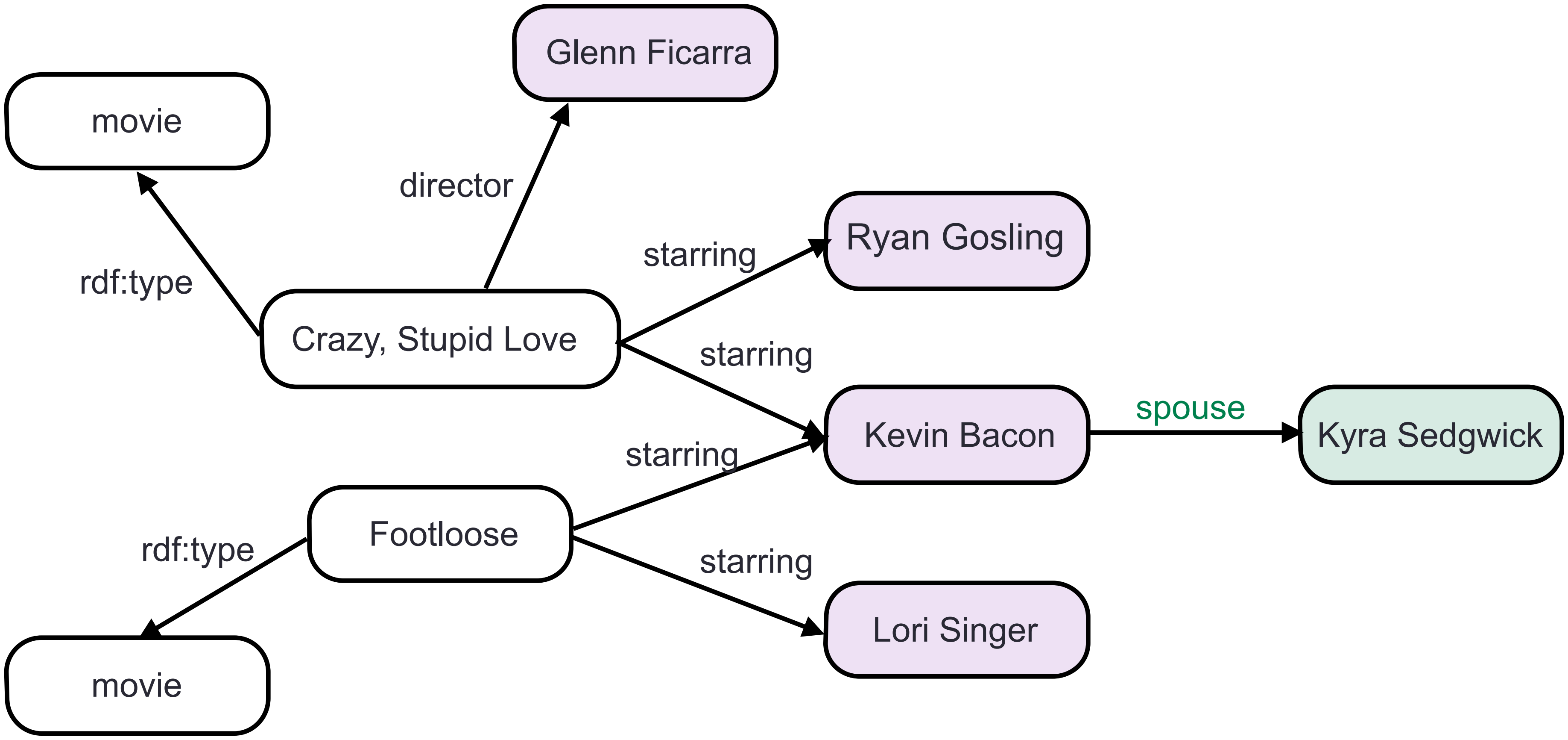
violation: every movie needs at least one director

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!

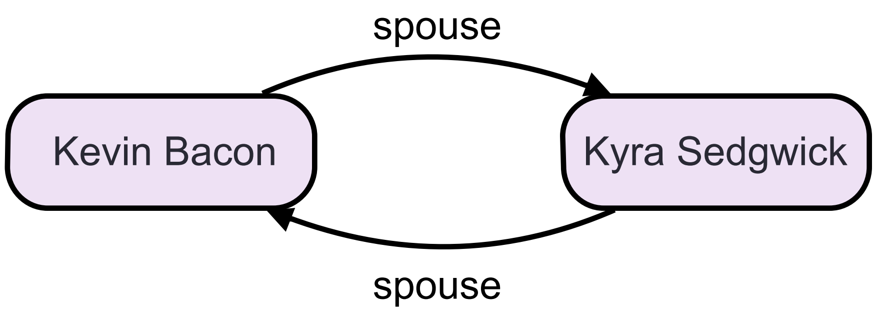
What have we done

JSON Schema
SHACL (Shapes Constraint Language)

Helping with specification
Semantics (specs never provide this)
Validation
Learning

Remark: semantics

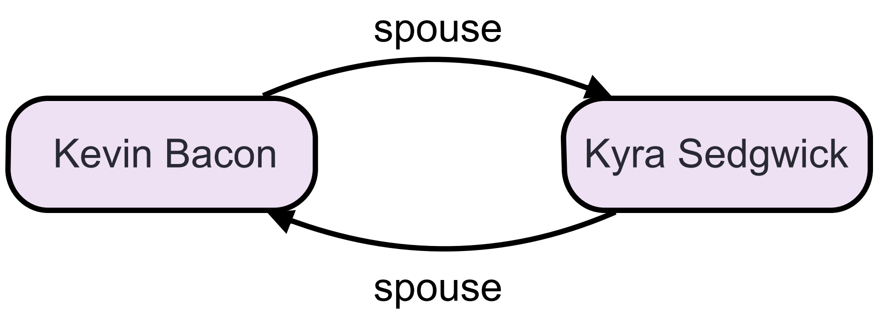
Remark: semantics



“Spouses of persons are persons”

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

Remark: semantics

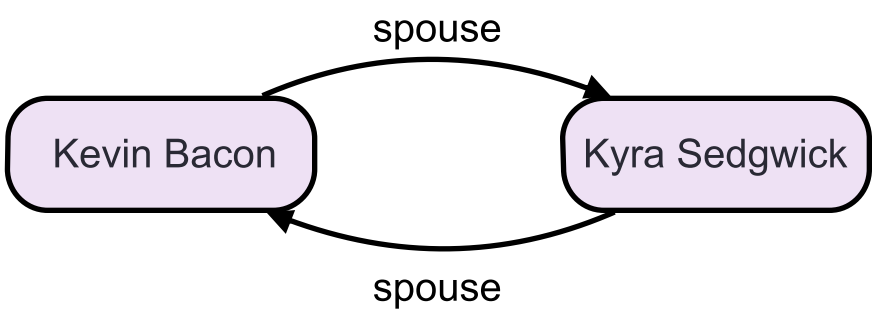


“Spouses of persons are persons”

SAT semantics:

graph satisfy schema if
there is an assignment of shapes
that satisfy the constraints

Remark: semantics



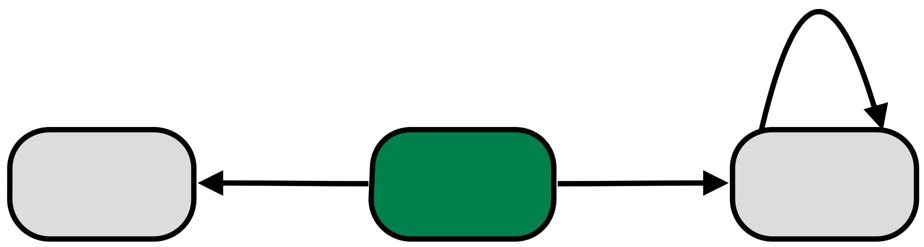
"Spouses of persons are persons"

Stable Model Semantics:

graph satisfy schema if
there is an assignment of shapes
that satisfy the constraints
and where each assignment is justified

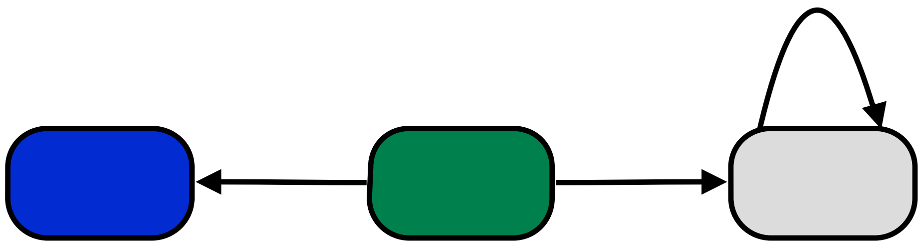
Remark: semantics

Guess a good assignment?



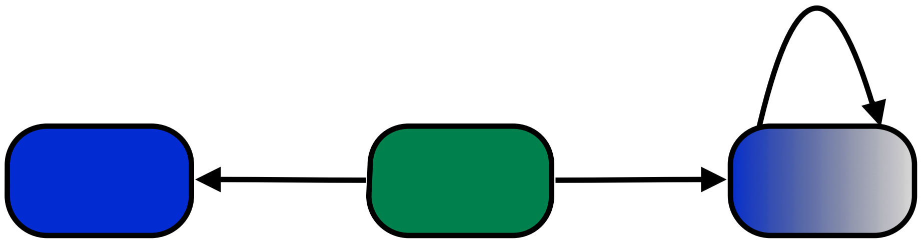
🟩	“I have a blue neighbour”
🟦	“My neighbours are not blue”

Remark: semantics
Guess a good assignment?



🟩	"I have a blue neighbour"
🟦	"My neighbours are not blue"

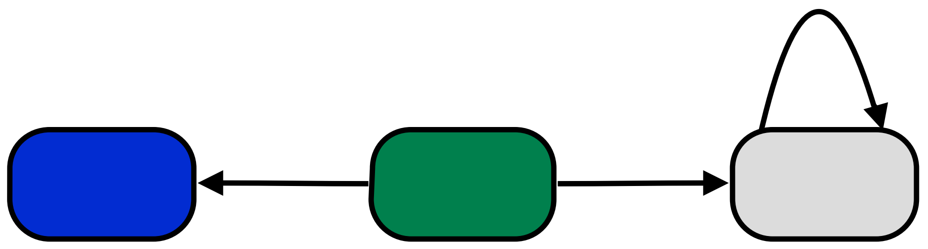
Remark: semantics
Guess a good assignment?



🟩 “I have a blue neighbour”

🟦 “My neighbours are not blue”

Remark: semantics
Guess a (partial) good assignment

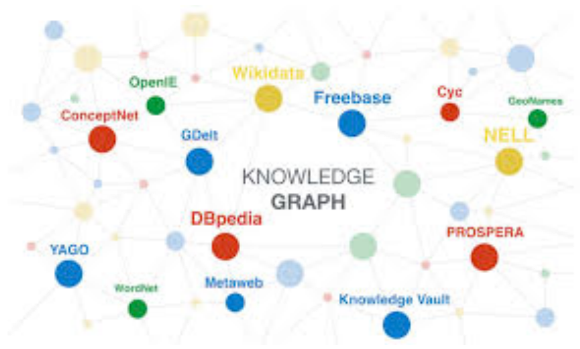


🟩	“I have a blue neighbour”
🟦	“My neighbours are not blue”

Where should we go from here?

Where should we go from here?

OPEN API
INITIATIVE



everyone wants schemas
difficult to write

need to learn all these schemas

Where should we go from here?

Querying!

Use the schema to speed things up