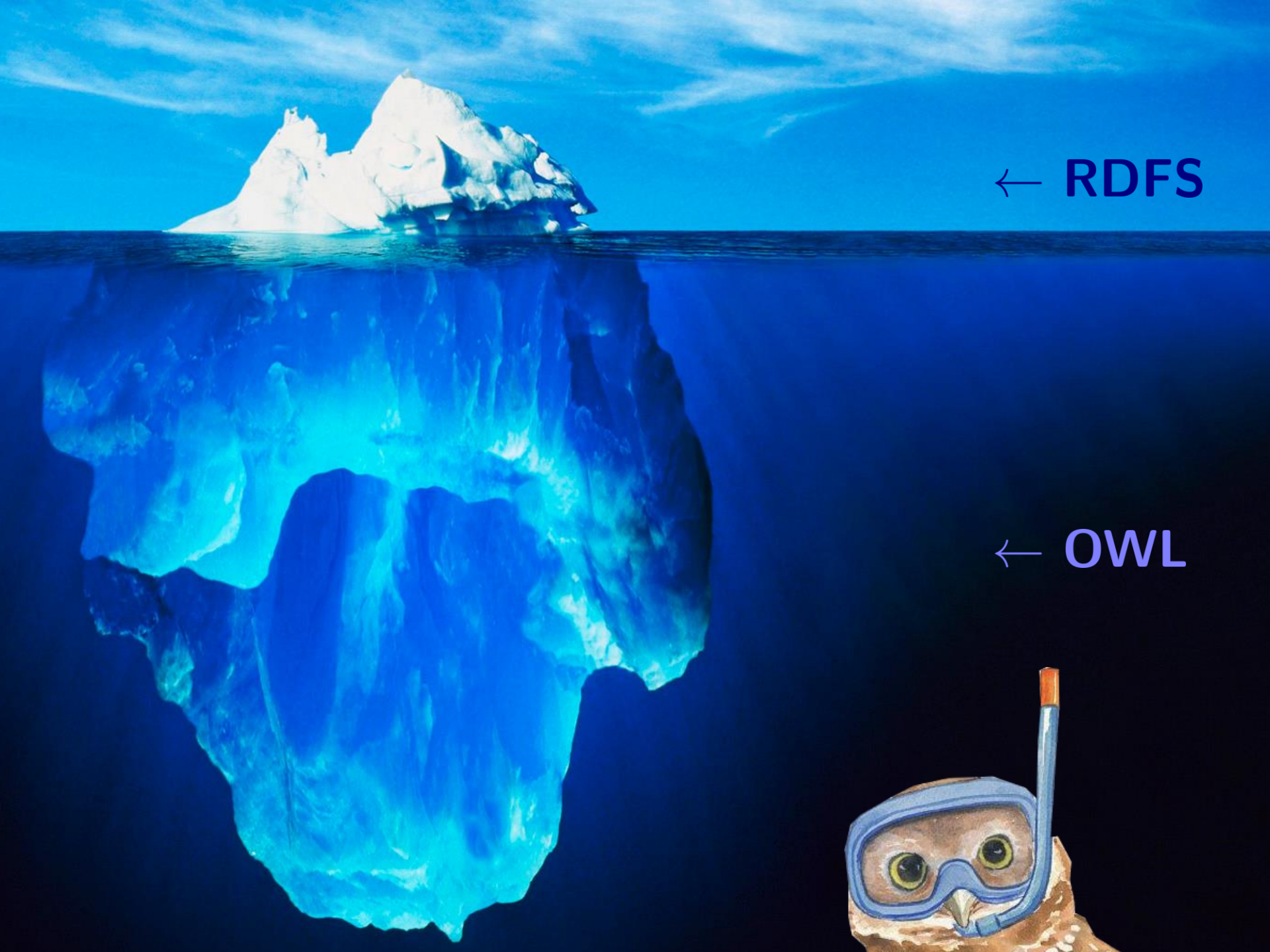
CC7220-1
La Web de Datos
Primavera 2018

Lecture 7: SPARQL [1.0]

Aidan Hogan
aidhog@gmail.com

Last time ...


← RDFS
← OWL

* More or less

Semantic Web: Data → Rules → Query → Output*

Data:

Ireland

(Ireland,partOf,Europe)
(Ireland,isA,Country)
(Ireland,capital,Dublin)

Dublin

(Ireland,capital,Dublin)
(Dublin,population,1000000)

Rules: “$(b$,capital,$a)$ → $(a$,partOf,$b)$”
“$(a$,partOf,$b)$, $(b$,partOf,$c)$ → $(a$,partOf,$c)$”

Query: “$(x$,partOf,$y)$?”

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

Today's Topic

* More or less

Semantic Web: Data → Rules → Query → Output*

Data:

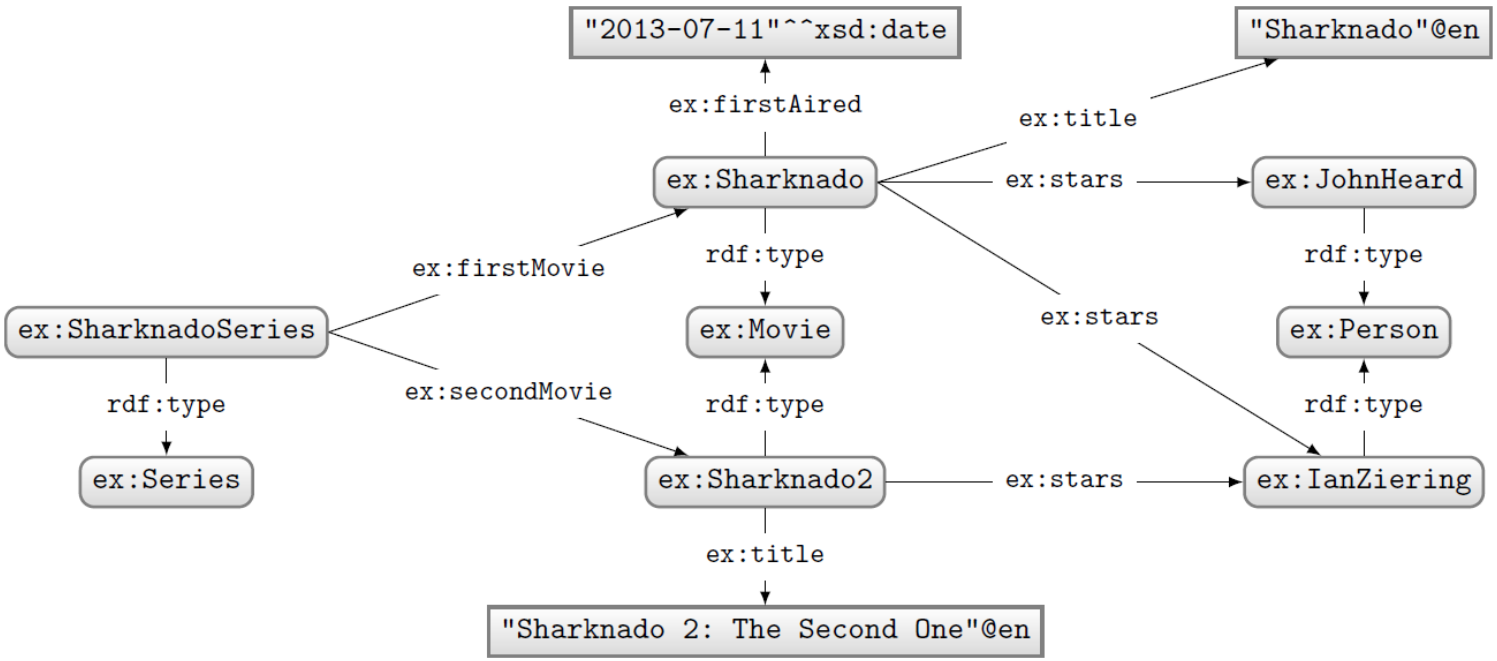
Rules: “$(b$,capital,$a) \rightarrow (a$,partOf,$b)$”
“$(a$,partOf,$b)$, $(b$,partOf,$c) \rightarrow (a$,partOf,$c)$”

Query: “$(x$,partOf,$y)$?”

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

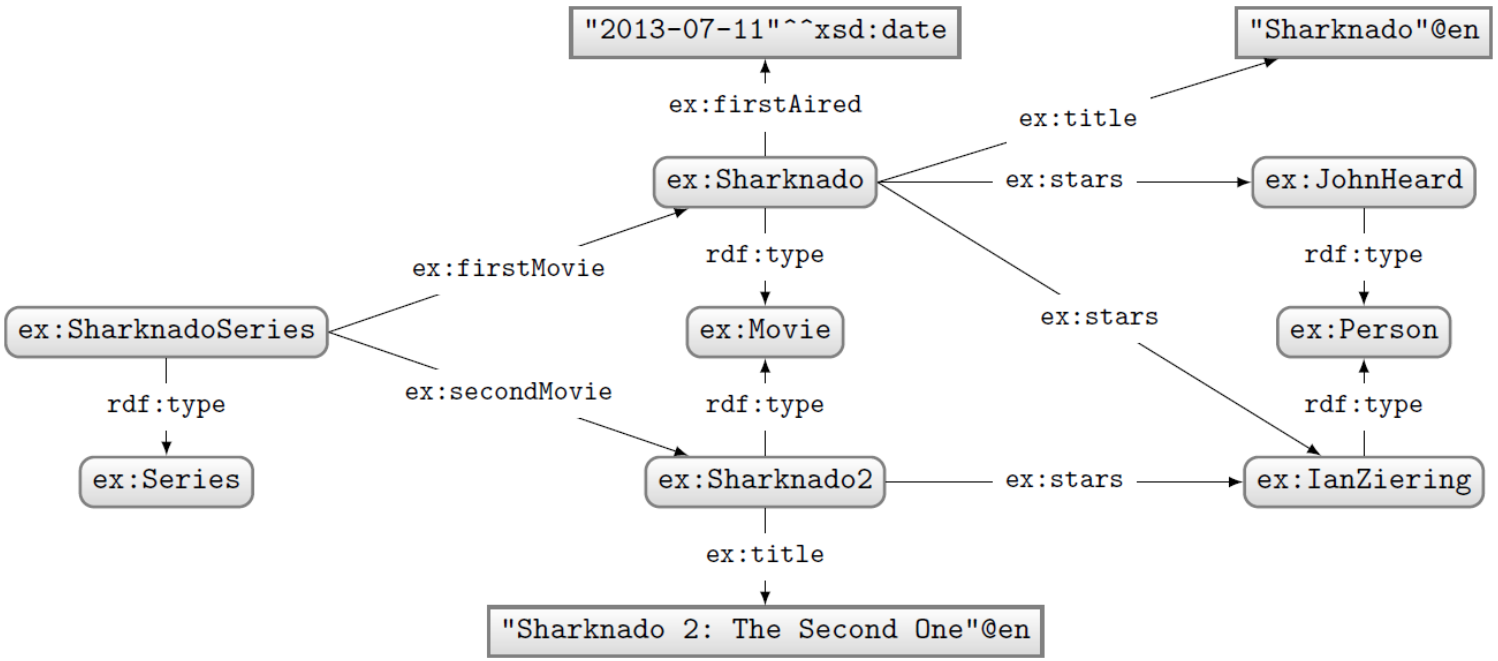
First SPARQL (1.0)
Then SPARQL 1.1

SPARQL: Query Language for RDF



How to ask: “Who stars in ‘Sharknado’?”

SPARQL: Query Language for RDF



Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
  ex:Sharknado ex:stars ?star .
}
```

Solutions:

?star
ex:JohnHeard
ex:IanZiering

SPARQL: Prefix declarations

SPARQL: PREFIX DECLARATIONS

- Shortcuts for IRIs (exactly like in Turtle)

```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
  ex:Sharknado ex:stars ?star .
}
```

SPARQL: WHERE CLAUSE

SPARQL: WHERE CLAUSE

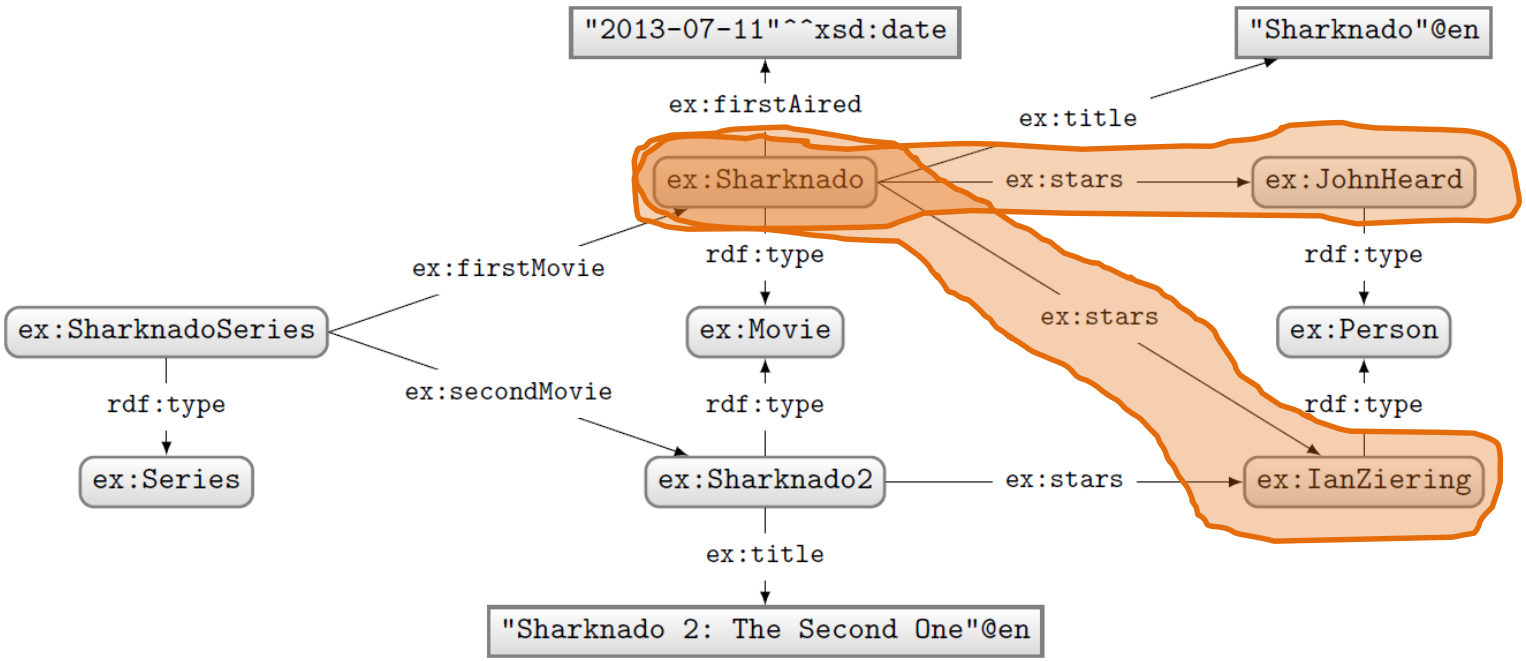
- Specifies what to match in the data


```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
  ex:Sharknado ex:stars ?star .
}
```

"Triple pattern"

(a triple with variables)

SPARQL: WHERE CLAUSE



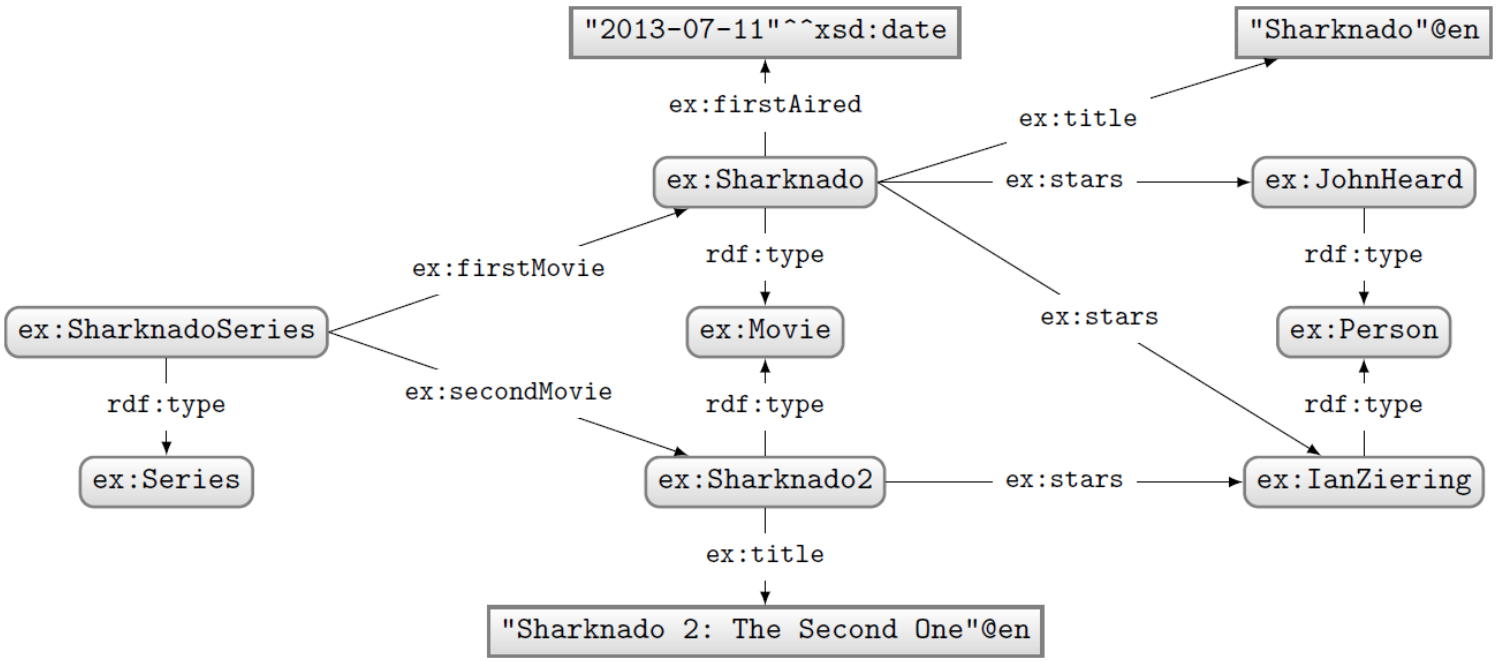
Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
  ex:Sharknado ex:stars ?star .
}
```

Solutions:

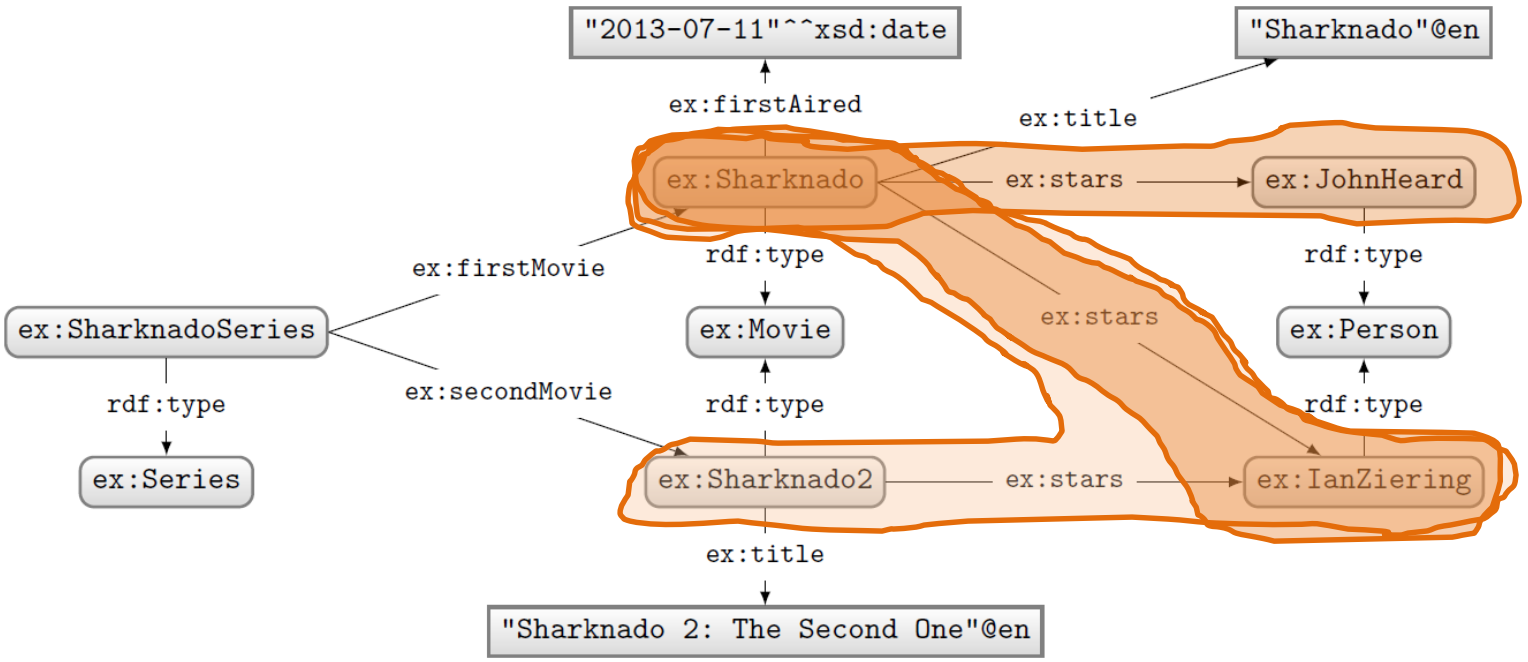
?star
ex:JohnHeard
ex:IanZiering

SPARQL: **WHERE** CLAUSE



How to ask: “What movies did the stars of ‘Sharknado’ also star in?”

SPARQL: Basic Graph Patterns



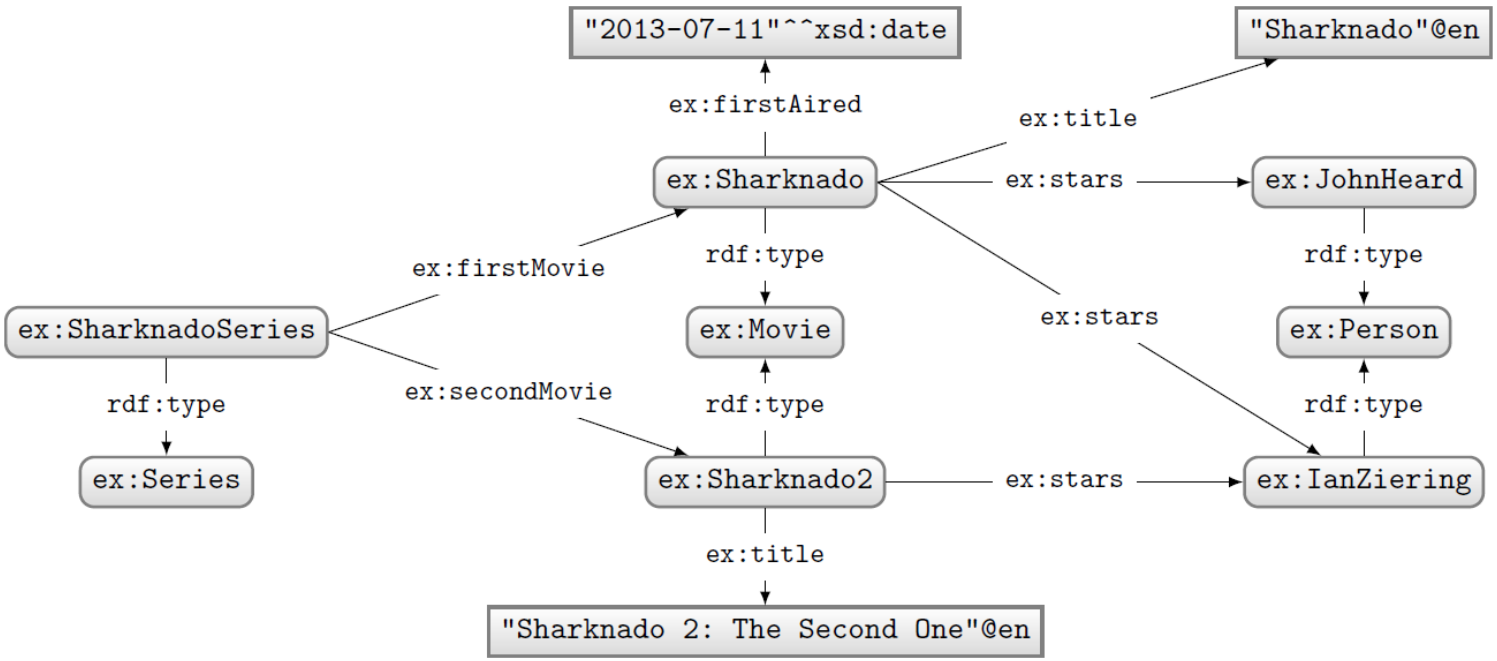
Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
  ex:Sharknado ex:stars ?star .
  ?movie ex:stars ?star .
}
```

Solutions:

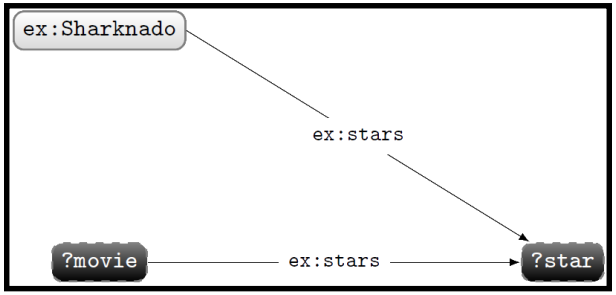
?star	?movie
ex:IanZiering	ex:Sharknado2
ex:IanZiering	ex:Sharknado
ex:JohnHeard	ex:Sharknado

SPARQL: Basic Graph Patterns



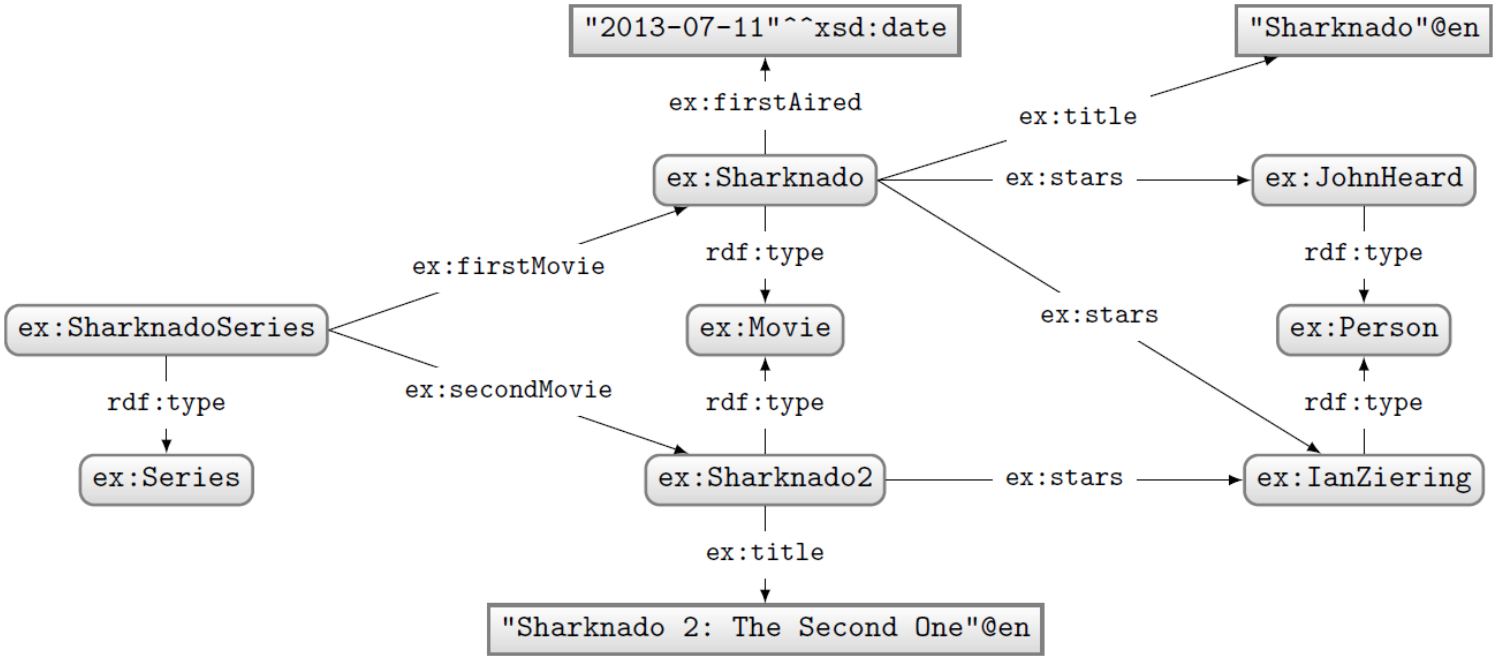
Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
  ex:Sharknado ex:stars ?star .
  ?movie ex:stars ?star .
}
```



"Basic Graph Pattern"
(a set of triple patterns)

SPARQL: Join Variables



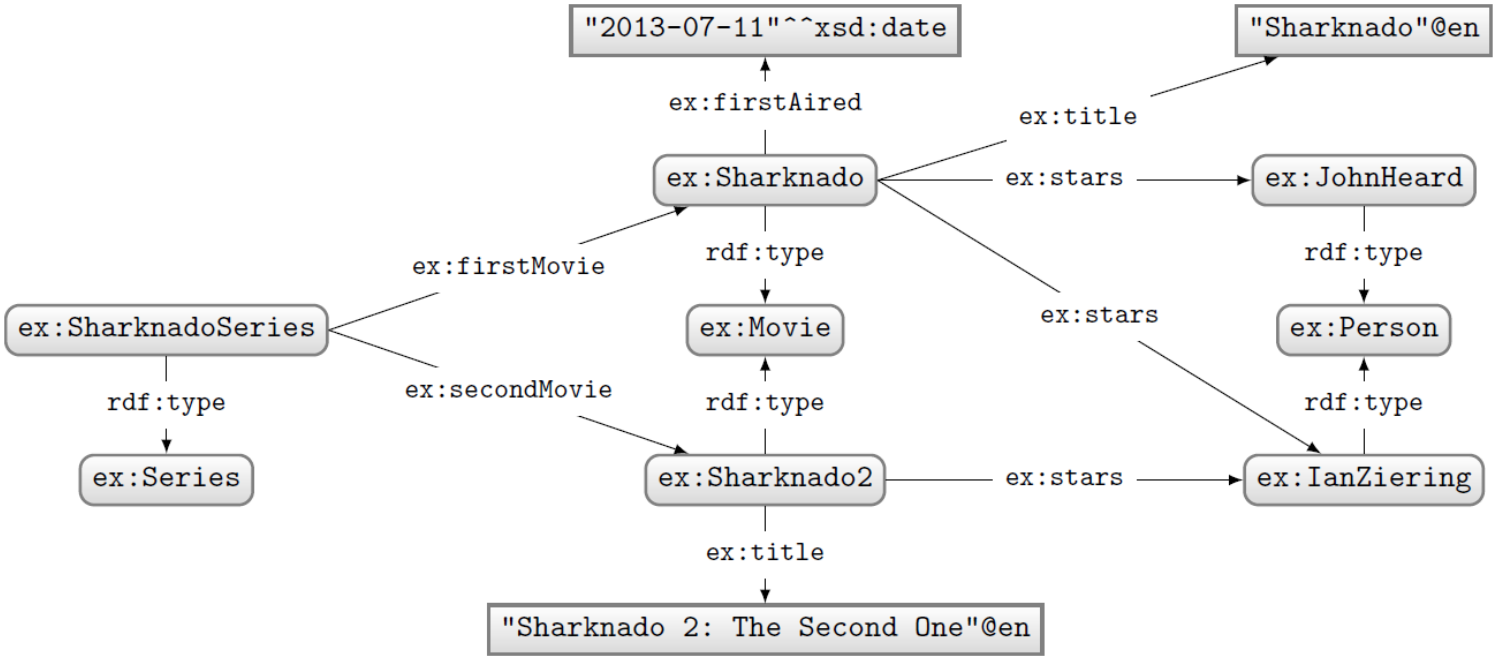
Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
  ex:Sharknado ex:stars ?star .
  ?movie ex:stars ?star .
}
```

"Join Variable"

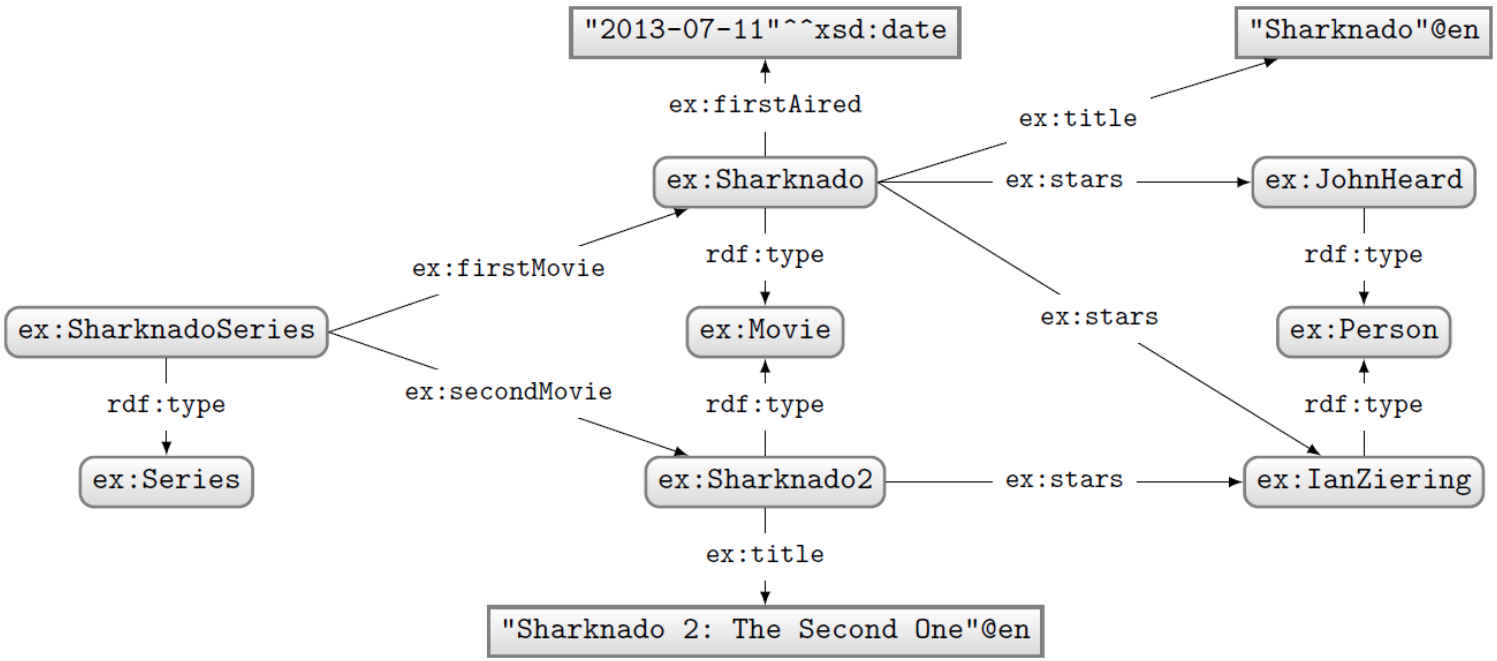
(a variable appearing multiple times)

SPARQL: Disjunction



How to ask: "What are the titles of the (first two) movies in the Sharknado series?"

SPARQL: Disjunction (UNION)



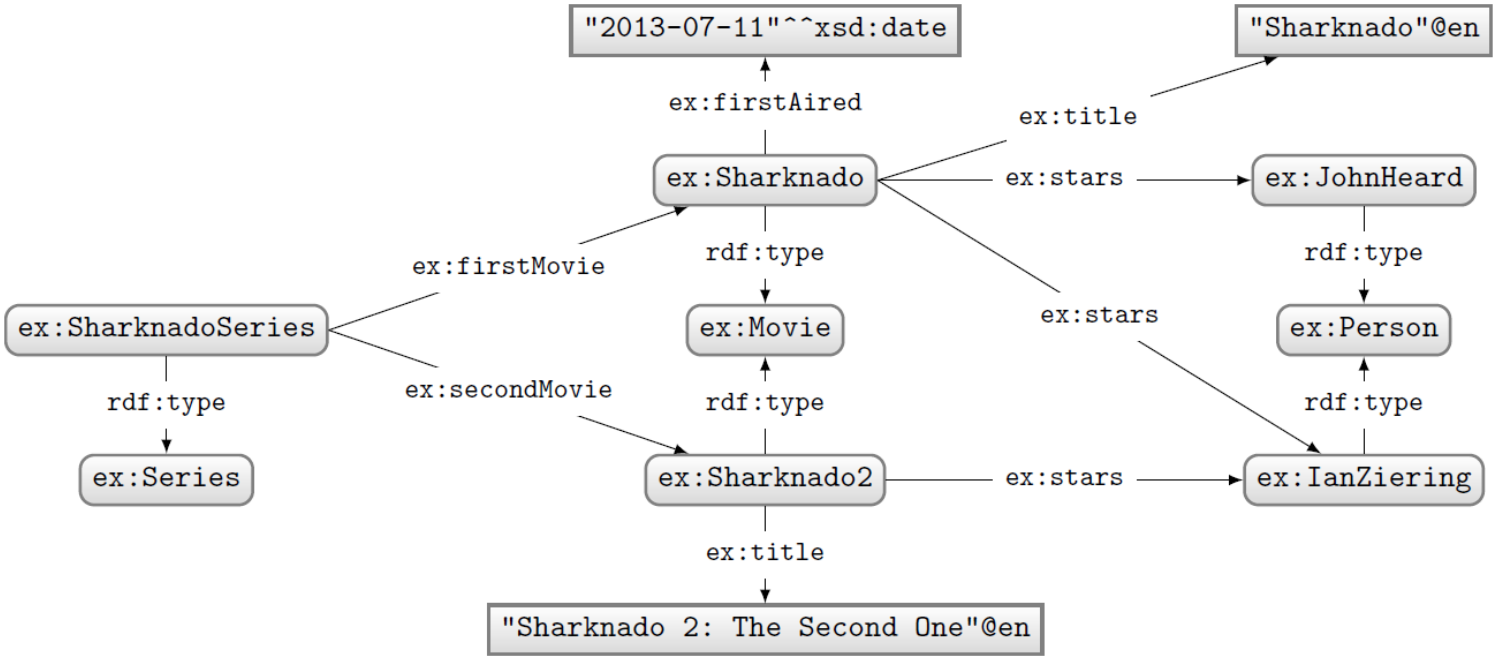
Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
  { ex:SharknadoSeries ex:firstMovie ?movie . }
  UNION
  { ex:SharknadoSeries ex:secondMovie ?movie . }
  ?movie ex:title ?title .
}
```

Solutions:

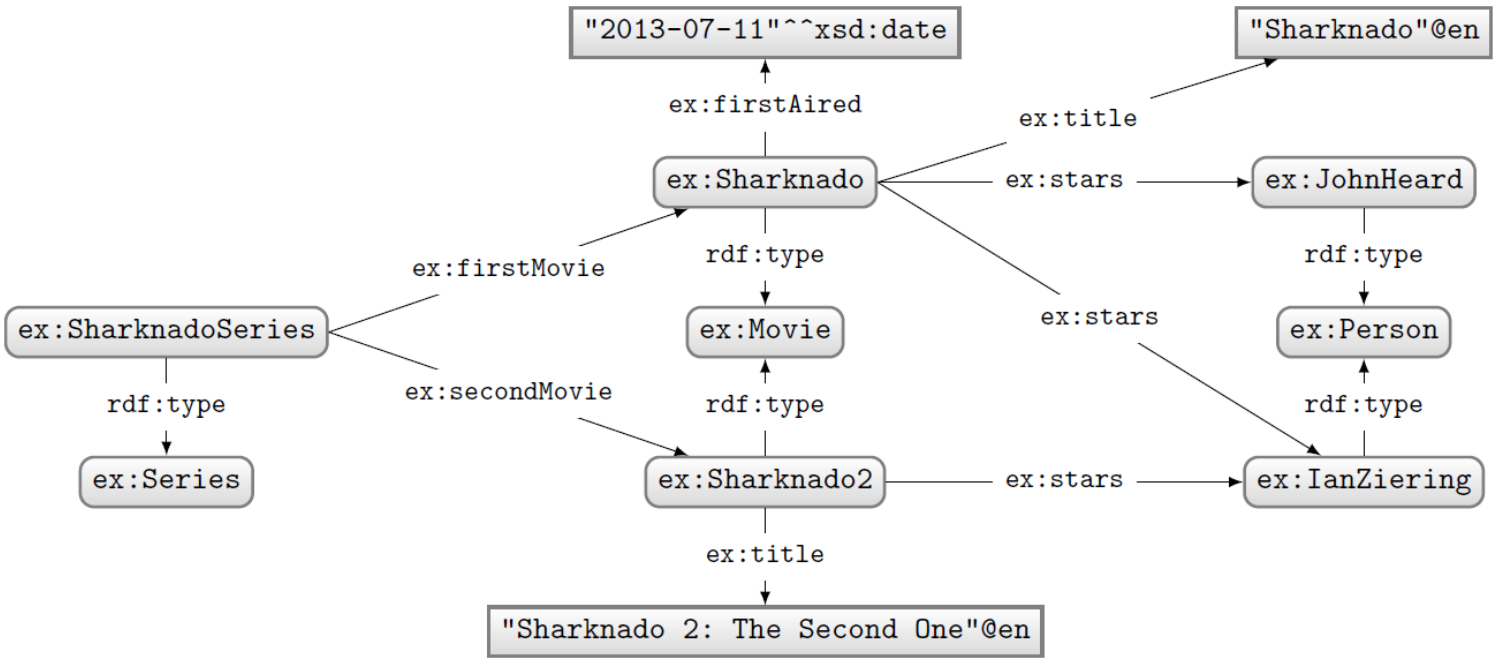
?movie	?title
ex:Sharknado	"Sharknado"@en
ex:Sharknado2	"Sharknado 2: The Second One"@en

SPARQL: Left-join



How to ask: "Give me the titles of all movies and, if available, their first-aired date?"

SPARQL: Left-join (OPTIONAL)



Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
  ?movie a ex:Movie ; ex:title ?title .
  OPTIONAL { ?movie ex:firstAired ?date }
}
```

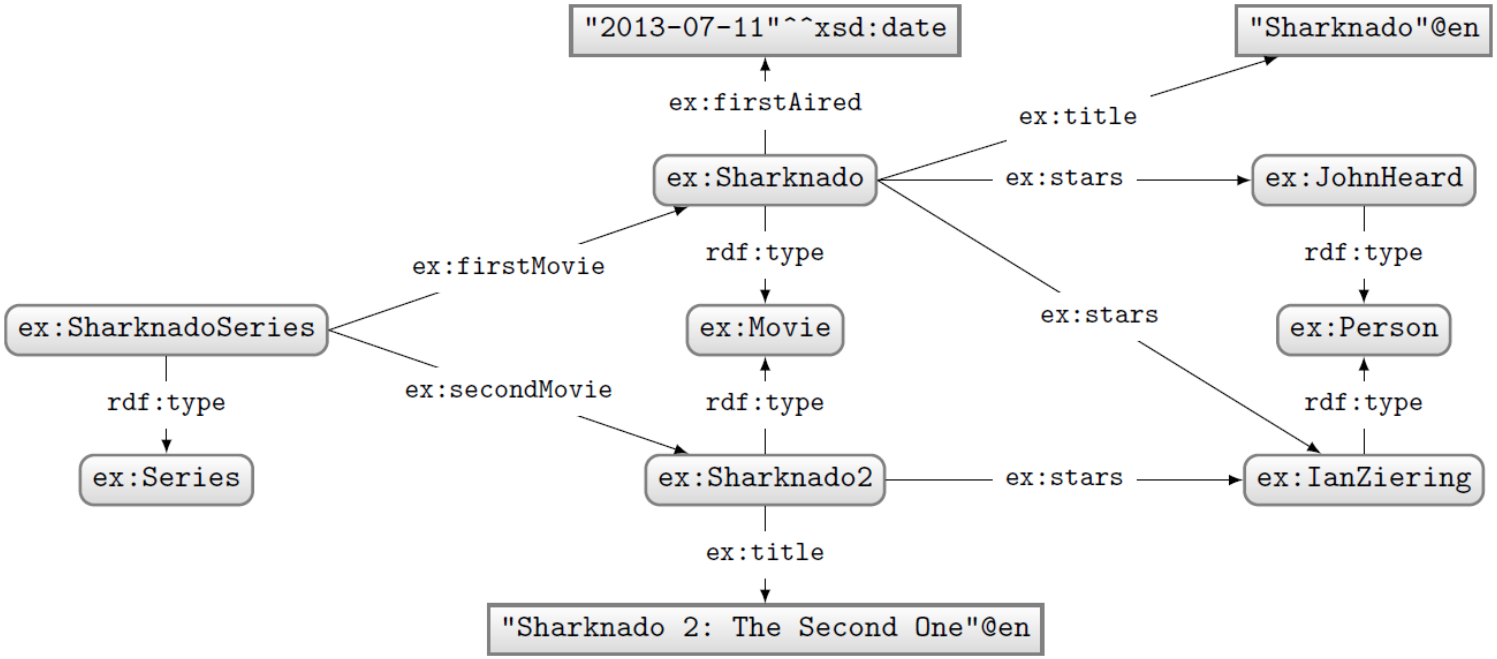
Solutions:

?movie	?title	?date
ex:Sharknado	"Sharknado"@en	"2013-07-11"^^xsd:date
ex:Sharknado2	"Sharknado 2: The Second One"@en	

"UNBOUND Variable"

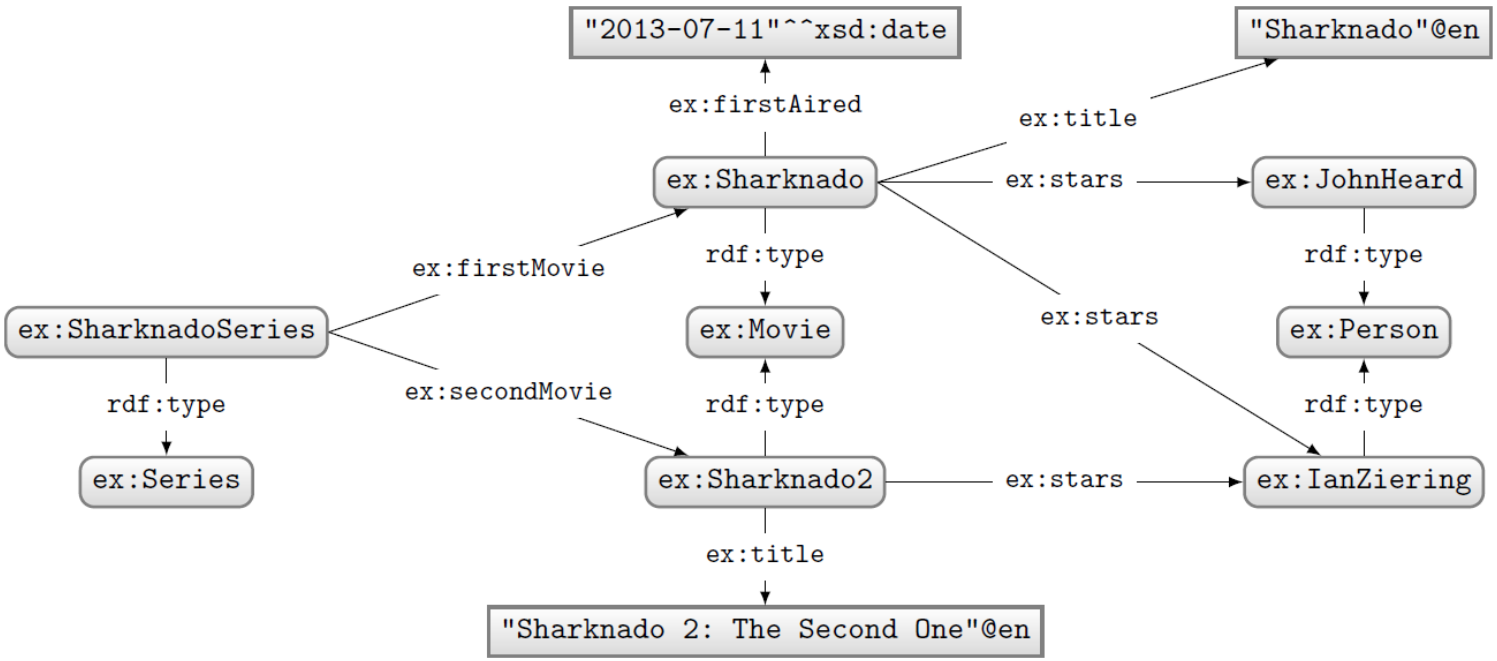
(a variable without a binding in a solution)

SPARQL: Filtering results



How to ask: “What movies were first aired in 2014?”

SPARQL: FILTER



Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
  ?movie a ex:Movie ; ex:firstAired ?date .
  FILTER(?date > "2013-12-31"^^xsd:date
        && ?date <="2014-12-31"^^xsd:date)
}
```

Solutions:

?movie	?date

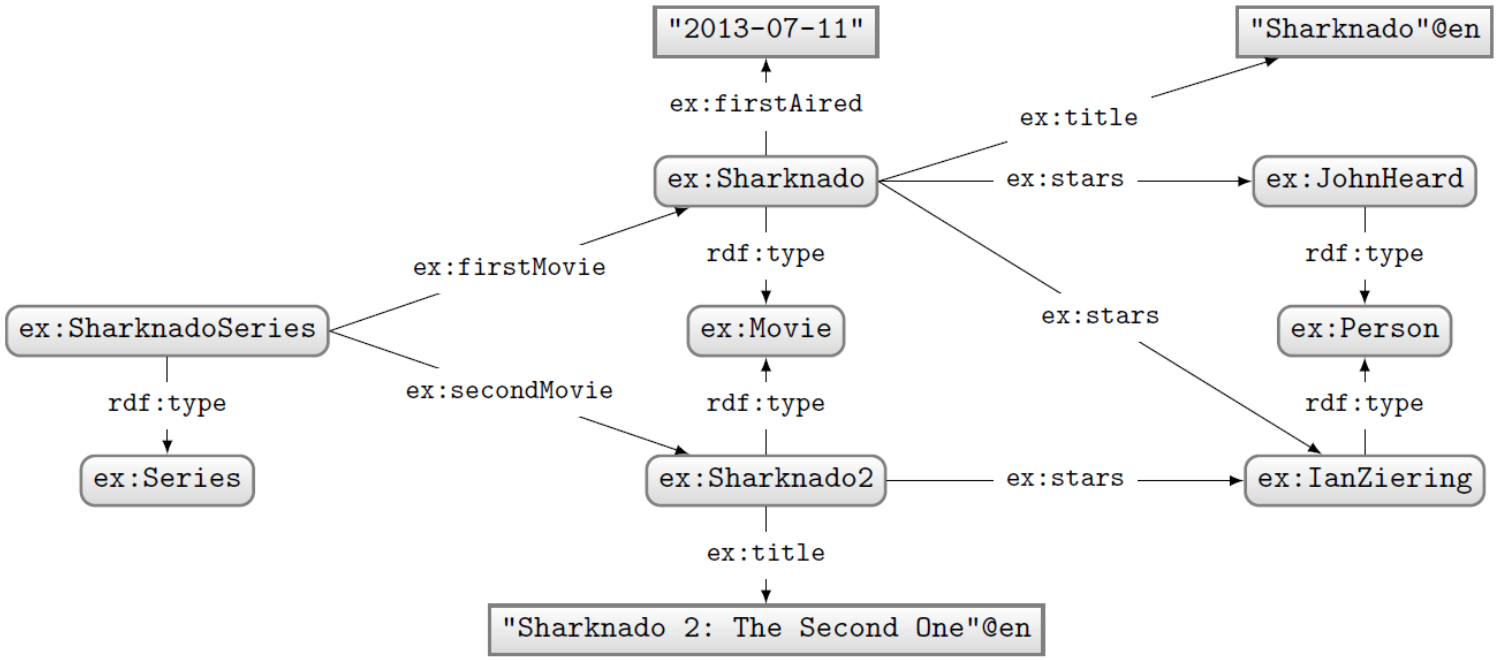
"Empty Results"

... also missing the xsd: prefix ;)

Any problem here?

... be careful comparing dates without time-zones!

SPARQL: FILTER



Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
  ?movie a ex:Movie ; ex:firstAired ?date .
  FILTER(?date > "2013-12-31"^^xsd:date
        && ?date <="2014-12-31"^^xsd:date)
}
```

What happens in this case where ?date bound in data to a string?

FILTERs (and other functions we see later) expect certain types. If not given, a type error is given.

SPARQL: Boolean `FILTER` operators

- `FILTER`s evaluate as `true`, `false` or `error`
- Only results evaluating as `true` are returned
- Can apply AND (`&&`) or OR (`||`)
- Can also apply NOT (`!`)
  - `!E` → `E`

A	B	A || B	A && B
T	T	T	T
T	F	T	F
F	T	T	F
F	F	F	F
T	E	T	E
E	T	T	E
F	E	E	F
E	F	E	F
E	E	E	E

SPARQL Operators

A	**Op**	B	**Return type and value**	
	`!`	BOOL b	BOOL	`true` if $I_L(b)$ is false; `false` otherwise
BOOL b_1	`||`	BOOL b_2	BOOL	`true` if $I_L(b_1)$ or $I_L(b_2)$; `false` otherwise
BOOL b_1	`&&`	BOOL b_2	BOOL	`true` if $I_L(b_1)$ and $I_L(b_2)$; `false` otherwise
TERM* t_1	`=`	TERM* t_2	BOOL	`true` if t_1 same term as t_2; `false` otherwise
TERM* t_1	`!=`	TERM* t_2	BOOL	`true` if t_1 not same term as t_2; `false` otherwise
COM v_1	`=`	COM v_2	BOOL	`true` if $I_L(v_1) = I_L(v_2)$; `false` otherwise
COM v_1	`!=`	COM v_2	BOOL	`true` if $I_L(v_1) \neq I_L(v_2)$; `false` otherwise
COM v_1	`<`	COM v_2	BOOL	`true` if $I_L(v_1) < I_L(v_2)$; `false` otherwise
COM v_1	`>`	COM v_2	BOOL	`true` if $I_L(v_1) > I_L(v_2)$; `false` otherwise
COM v_1	`<=`	COM v_2	BOOL	`true` if $I_L(v_1) \leq I_L(v_2)$; `false` otherwise
COM v_1	`>=`	COM v_2	BOOL	`true` if $I_L(v_1) \geq I_L(v_2)$; `false` otherwise
	`+`	NUM n	NUM	n
	`-`	NUM n	NUM	$-n$
NUM n_1	`+`	NUM n_2	NUM	$I_L(v_1) + I_L(v_2)$
NUM n_1	`-`	NUM n_2	NUM	$I_L(v_1) + I_L(v_2)$
NUM n_1	`*`	NUM n_2	NUM	$I_L(v_1) \times I_L(v_2)$
NUM n_1	`/`	NUM n_2	NUM	$\frac{I_L(v_1)}{I_L(v_2)}$

- COM: a comparable literal value
- TERM*: a non-comparable RDF term
- $I_L(\cdot)$: the value (e.g., 2 not "2")

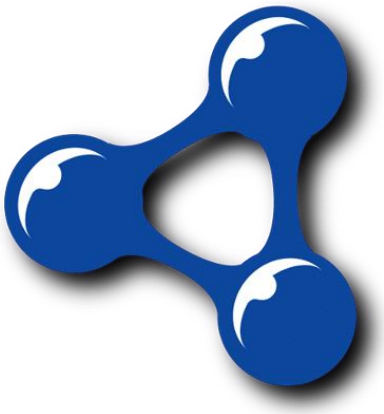
SPARQL Functions: Existence, equality, if ...

Function	Return type and value	
`bound`(TERM t)	BOOL	`true` if t is bound; `false` if unbound
`if`(BOOL b,TERM t_1,TERM t_2)	TERM	t_1 if b is `true`; t_2 otherwise
`coalesce`(TERM t_1,...,t_n)	TERM	first t_i $(1 \leq i \leq n)$ that is not an error or unbound
`not exists`(SUB Q)	BOOL	`true` if Q has any solution; `false` otherwise
`exists`(SUB Q)	BOOL	`true` if Q has no solution; `false` otherwise
`sameTerm`(TERM t_1, TERM t_2)	BOOL	`true` if t_1 same term as t_2; `false` otherwise
TERM t `in`(TERM t_1,...,t_n)	BOOL	`true` if $t = t_i$ for any $t_i \in \{t_1, ..., t_n\}$; `false` otherwise
TERM t `not in`(TERM t_1,...,t_n)	BOOL	`true` if $t \neq t_i$ for all $t_i \in \{t_1, ..., t_n\}$; `false` otherwise

- SUB: a sub-query

SPARQL Functions: Terms

Function	Return type and value	
`isIRI`(TERM t)	BOOL	`true` if t is an IRI; `false` otherwise
`isBlank`(TERM t)	BOOL	`true` if t is a blank node; `false` otherwise
`isLiteral`(TERM t)	BOOL	`true` if t is a literal; `false` otherwise
`isNumeric`(TERM t)	BOOL	`true` if t is a numeric value; `false` otherwise
`str`(LIT l|IRI i)	STR	lexical value of l | string of i
`lang`(LIT l)	STR	language tag string of l
`datatype`(LIT l)	IRI	datatype IRI of l
`iri`(STR s|IRI i)	IRI	s resolved against the in-scope base IRI | i
`bnode`([STR s])	BNODE	fresh blank node [unique to s]
`strdt`(STR s,IRI i)	LIT	"s"^^<i>
`strlang`(STR s,STR l)	LIT	"s"@l
`uuid`()	IRI	fresh IRI (from UUID URN scheme)
`struuid`()	STR	fresh string (from UUID URN scheme)



- $a|b$ indicates a or b
- $[a]$ indicates a optional

SPARQL Functions: Strings

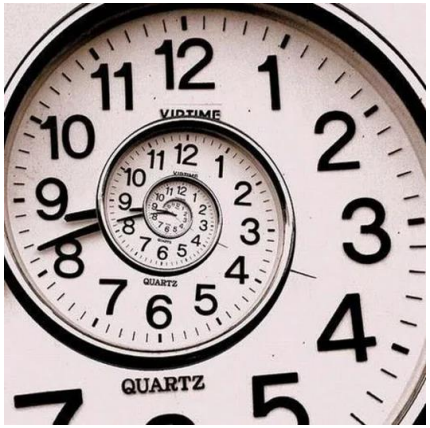
Function	Return type and value	
`strlen`(STR s)	INT	length of string s
`substr`(STR s,INT b,[INT l])	STR	substring of s from index b [of length l]
`ucase`(STR s)	STR	uppercase s
`lcase`(STR s)	STR	lowercase s
`strstarts`(STR s,STR p)	BOOL	`true` if s starts with p; `false` otherwise
`strends`(STR s,STR p)	BOOL	`true` if s ends with p; `false` otherwise
`strbefore`(STR s,STR p)	STR	string before first match for p in s
`strafter`(STR s,STR p)	STR	string after first match for p in s
`encode_for_iri`(STR s)	STR	s percent-encoded
`concat`(STR s_1,...,s_n)	STR	$s_1, ..., s_n$ concatenated
`langMatches`(STR s,STR l)	BOOL	`true` if s a language tag matching l; `false` otherwise
`regex`(STR s,STR p[,STR f])	BOOL	`true` if s matches regex p [with flags f]; `false` otherwise
`replace`(STR s,STR p,STR r[,STR f])	STR	s with matches for regex p [with flags f] replaced by r

SPARQL Functions: Numerics

Function	Return type and value	
`abs`(NUM n)	NUM	absolute value of n
`round`(NUM n)	NUM	round to nearest whole number (towards $+\infty$ for *.5)
`ceil`(NUM n)	NUM	round up (towards $+\infty$) to nearest whole number
`floor`(NUM n)	NUM	round down (towards $-\infty$) to nearest whole number
`rand`(NUM n)	NUM	random double between 0 (inclusive) and 1 (exclusive)

SPARQL Functions: Temporal

Function	Return type and value	
`now`()	DT	current date-time
`year`(DT d)	INT	year of d (as an integer)
`month`(DT d)	INT	month of d (as an integer)
`day`(DT d)	INT	day of d (as an integer)
`hours`(DT d)	INT	hours of d (as an integer)
`minutes`(DT d)	INT	minutes of d (as an integer)
`seconds`(DT d)	INT	seconds of d (as an integer)
`timezone`(DT d)	DTD	timezone of d (as day-time-duration)
`tz`(DT d)	STR	timezone of d (as a string)



- DT: date–time
- DTD: day–time–duration

SPARQL Functions: Hashing

Function	Return type and value	
md5(STR s)	STR	MD5 hash of s
sha1(STR s)	STR	SHA1 hash of s
sha256(STR s)	STR	SHA256 hash of s
sha384(STR s)	STR	SHA384 hash of s
sha512(STR s)	STR	SHA512 hash of s

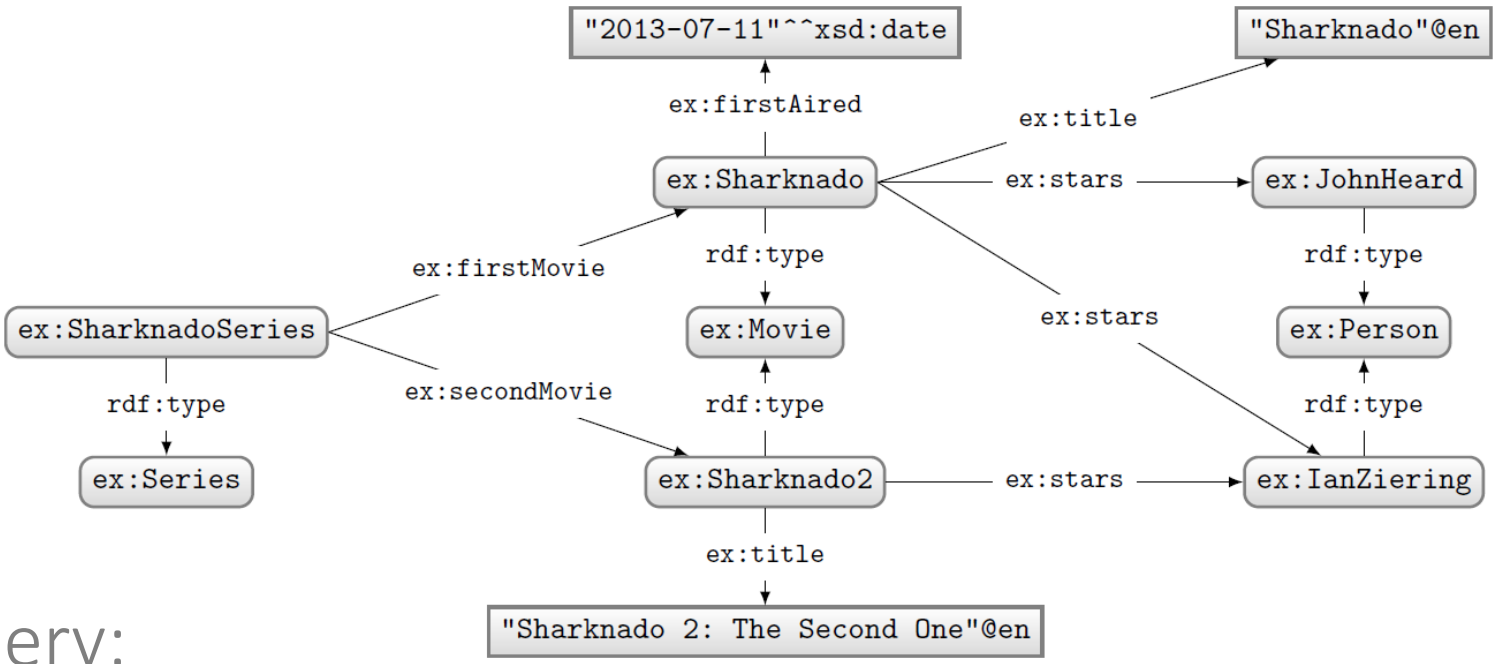
SPARQL: Casting between types

- Y: always allowed
- N: never allowed
- M: depends on value
  - e.g., "2"^^xsd:string can be mapped to xsd:int but "P"^^xsd:string cannot

From \ To	str	flt	dbl	dec	int	dT	bool
str	Y	M	M	M	M	M	M
flt	Y	Y	Y	M	M	N	Y
dbl	Y	Y	Y	M	M	N	Y
dec	Y	Y	Y	Y	Y	N	Y
int	Y	Y	Y	Y	Y	N	Y
dT	Y	N	N	N	N	Y	N
bool	Y	Y	Y	Y	Y	N	Y
IRI	Y	N	N	N	N	N	N
ltrl	Y	M	M	M	M	M	M

bool = xsd:boolean
dbl = xsd:double
flt = xsd:float
dec = xsd:decimal
int = xsd:integer
dT = xsd:dateTime
str = xsd:string
IRI = IRI
ltrl = simple literal

SPARQL: WHERE CLAUSE EXAMPLE (I)



Query:

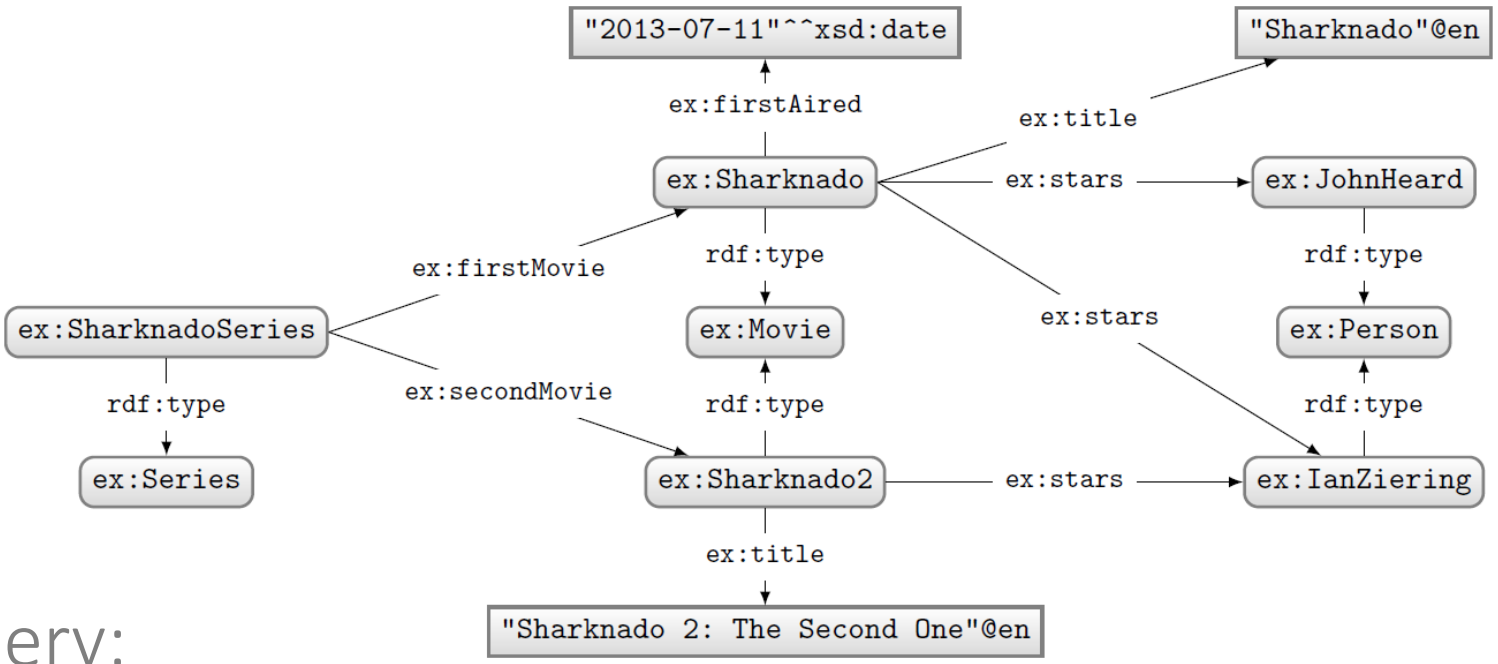
```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
  { ex:SharknadoSeries ex:firstMovie ?movie . }
  UNION
  { ex:SharknadoSeries ex:secondMovie ?movie . }
  OPTIONAL
  { ?movie ex:firstAired ?date . }
  ?movie ex:title ?title .
  FILTER(REGEX(STR(?title),"*[0-9]*"))
}
```

What solutions would this query return?

Solutions:

?movie	?title	?date
ex:Sharknado2	"Sharknado 2: The Second One"@en	

SPARQL: WHERE CLAUSE EXAMPLE (II)



Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
  ?movie a ex:Movie .
  OPTIONAL
  { ?movie ex:firstAired ?date . }
  FILTER(!BOUND(?date))
}
```

What solutions would this query return?

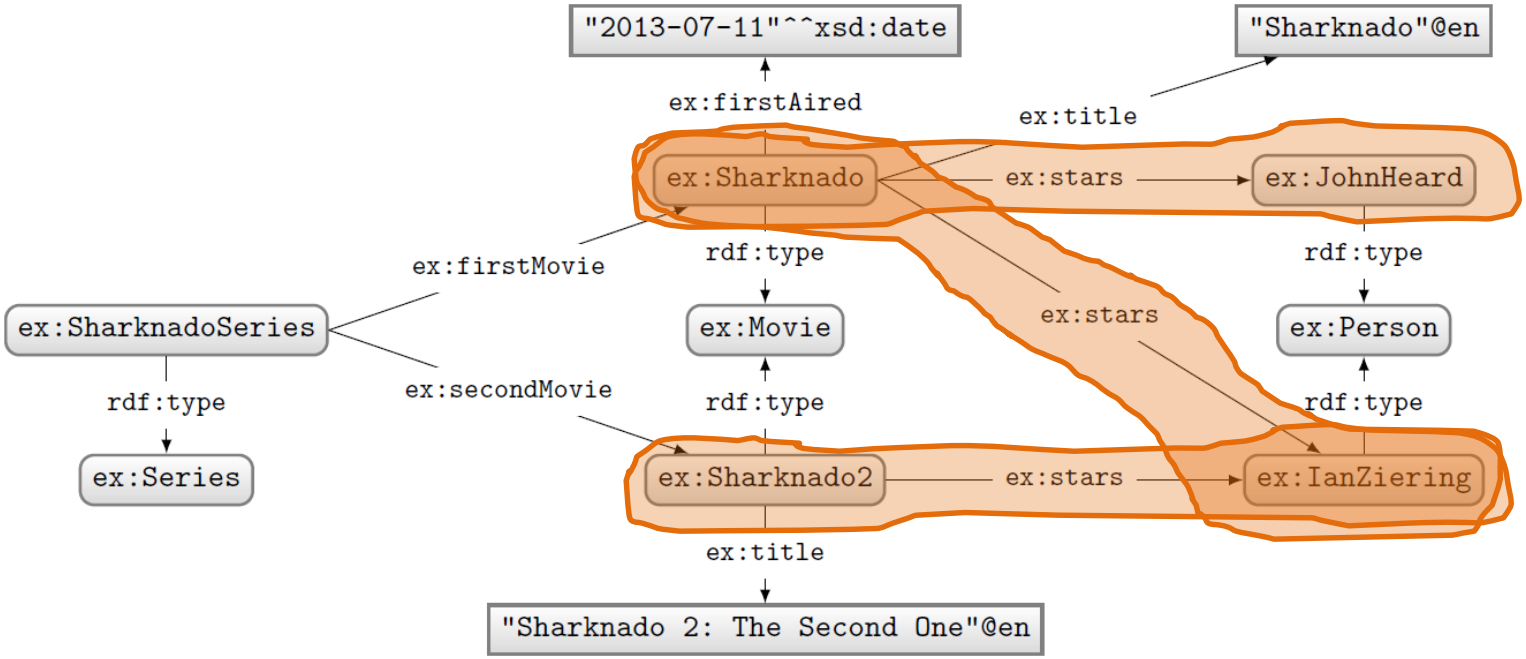
Solutions:

?movie	?date
ex:Sharknado2	

Can do a closed-world style of negation!

SPARQL: Query Types

SPARQL: SELECT WITH *



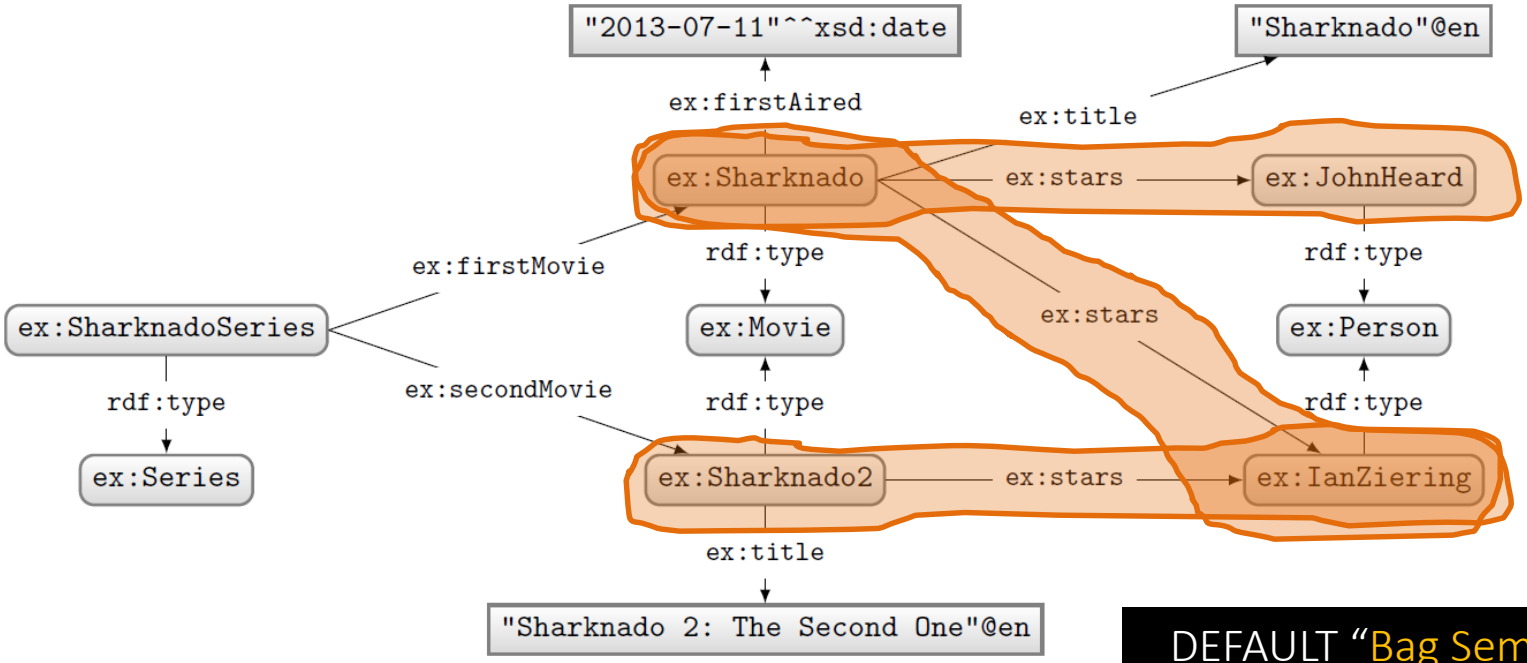
Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
  ?movie a ex:Movie.
  ?movie ex:stars ?star .
}
```

Solutions:

?movie	?star
ex:Sharknado	ex:JohnHeard
ex:Sharknado	ex:IanZiering
ex:Sharknado2	ex:IanZiering

SPARQL: SELECT WITH PROJECTION



DEFAULT "Bag Semantics"

(number of results returned must correspond to number of matches in data)

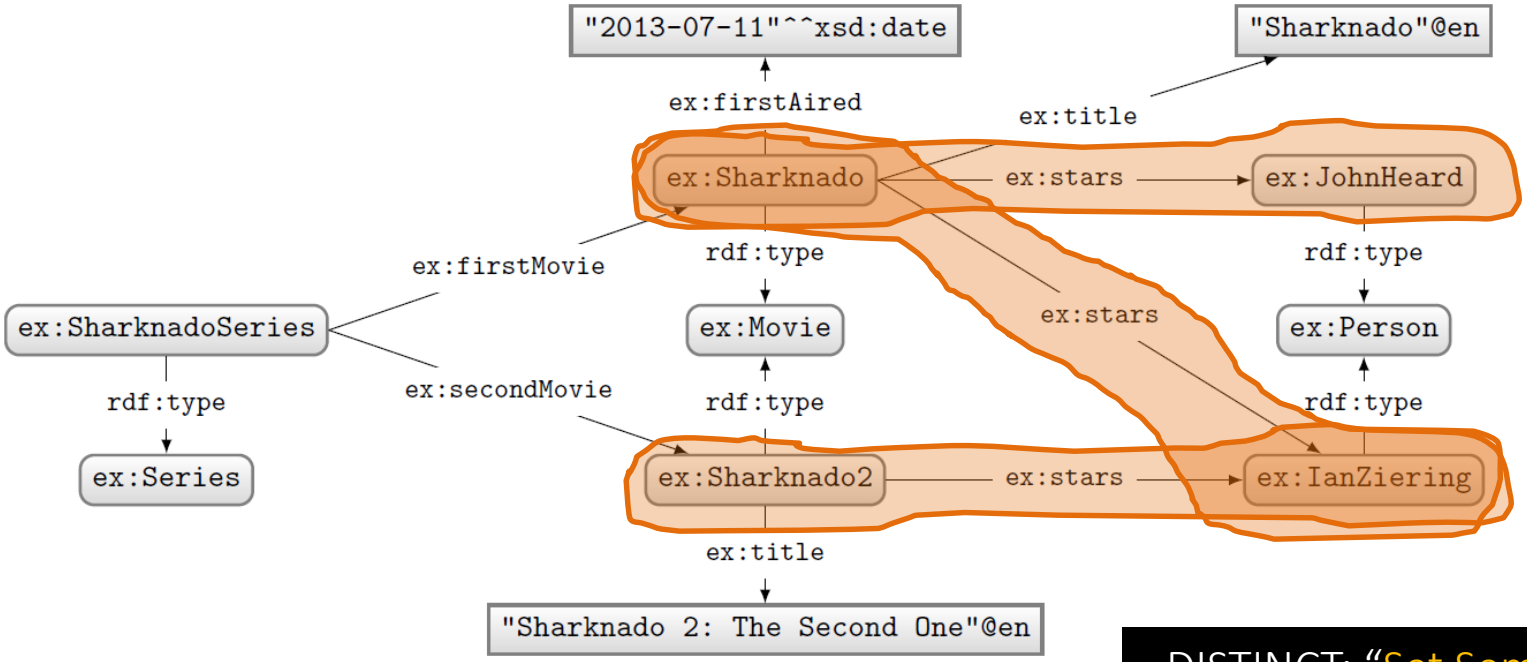
Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT ?star
WHERE {
  ?movie a ex:Movie.
  ?movie ex:stars ?star .
}
```

Solutions:

?star
ex:JohnHeard
ex:IanZiering
ex:IanZiering

SPARQL: SELECT WITH DISTINCT



DISTINCT: "Set Semantics"

(each result row must be unique)

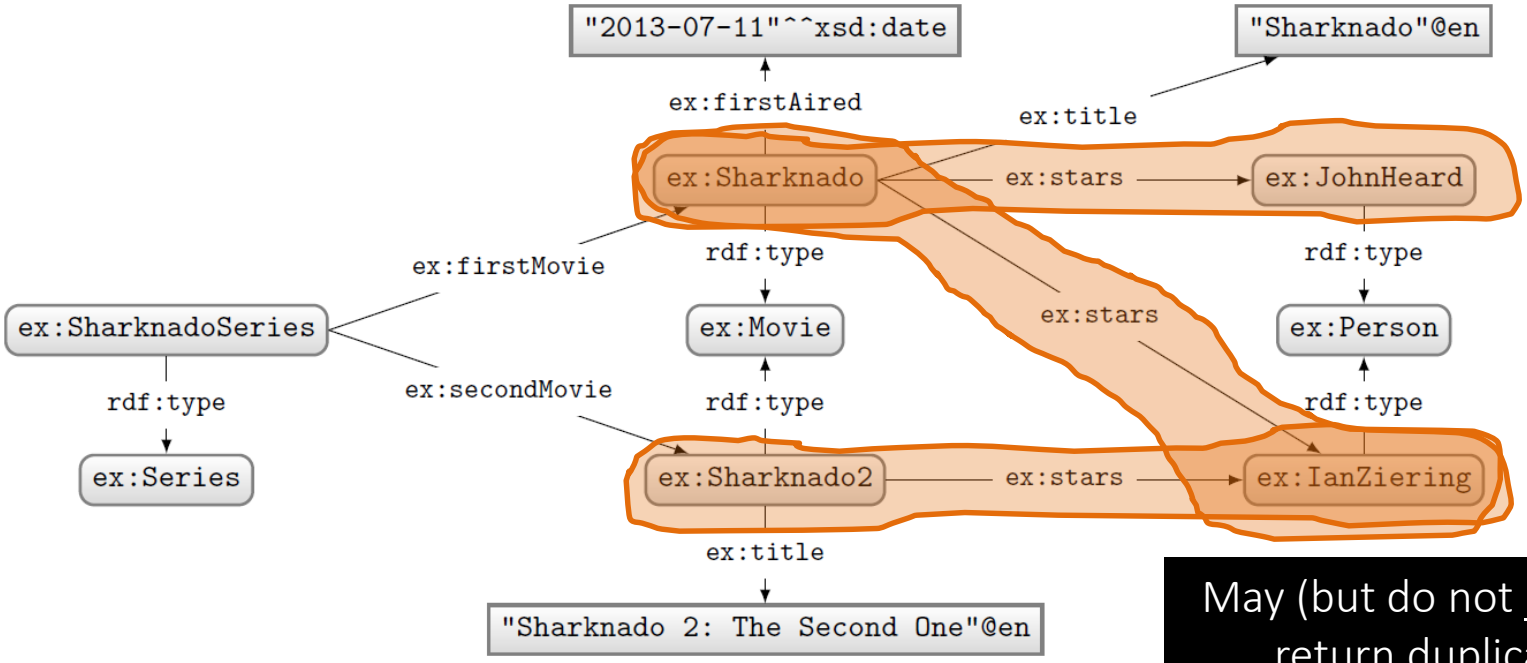
Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT DISTINCT ?star
WHERE {
  ?movie a ex:Movie.
  ?movie ex:stars ?star .
}
```

Solutions:

?star
ex:JohnHeard
ex:IanZiering

SPARQL: SELECT WITH REDUCED



May (but do not need to) return duplicates.
(This allows the engine do whatever is most efficient.)

Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT REDUCED ?star
WHERE {
  ?movie a ex:Movie.
  ?movie ex:stars ?star .
}
```

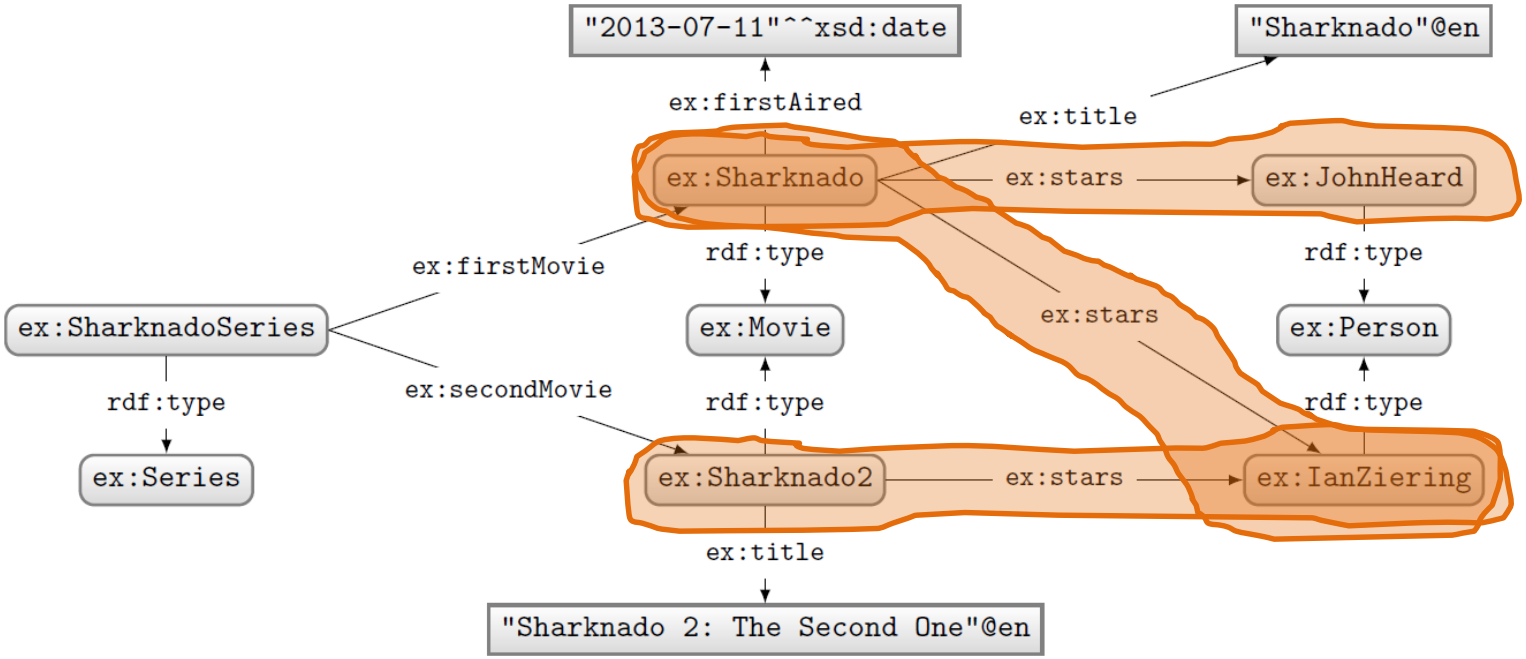
Solutions:

OR

?star
ex:JohnHeard
ex:IanZiering

?star
ex:JohnHeard
ex:IanZiering
ex:IanZiering

SPARQL: ASK



Query:

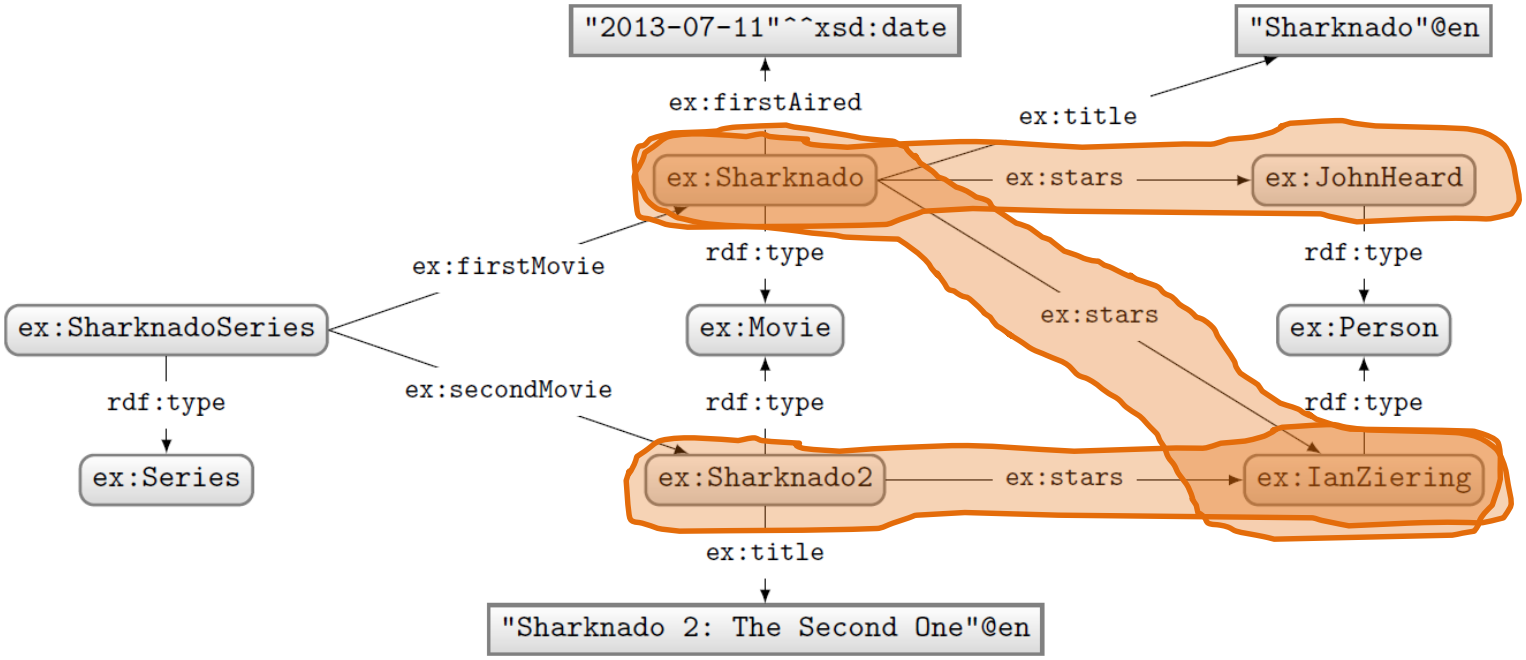
```
PREFIX ex: <http://ex.org/voc#>
ASK
WHERE {
  ?movie a ex:Movie.
  ?movie ex:stars ?star .
}
```

Solutions:

`true`

Returns `true` if there is a match, `false` otherwise.

SPARQL: CONSTRUCT



Query:

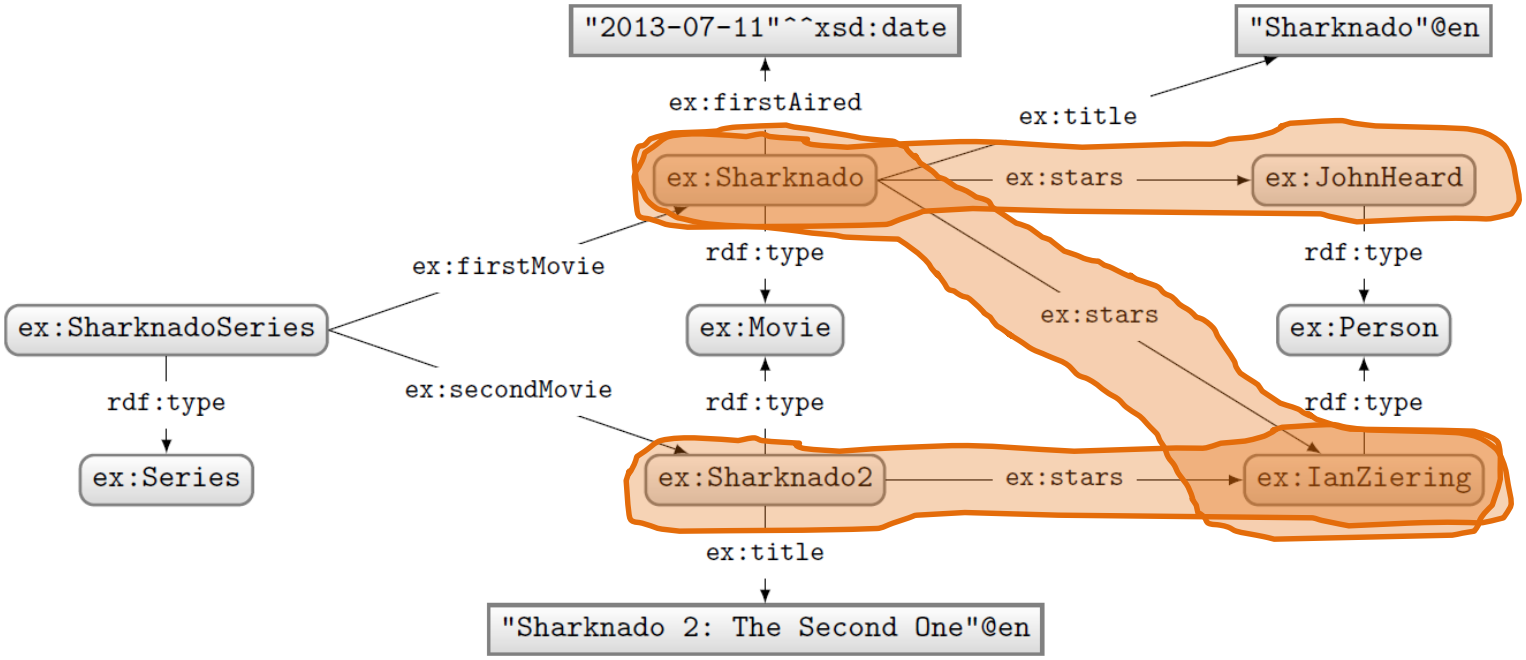
```
PREFIX ex: <http://ex.org/voc#>
CONSTRUCT { ?star ex:job ex:Actor }
WHERE {
  ?movie a ex:Movie.
  ?movie ex:stars ?star .
}
```

Solutions:

```
@prefix ex: <http://ex.org/voc#> .
ex:JohnHeard ex:job ex:Actor .
ex:IanZiering ex:job ex:Actor .
```

Returns an RDF graph based on the matching CONSTRUCT clause.

SPARQL: DESCRIBE (NON-NORMATIVE FEATURE)



Query:

```
PREFIX ex: <http://ex.org/voc#>
DESCRIBE ?star
WHERE {
  ?movie a ex:Movie.
  ?movie ex:stars ?star .
}
```

Solutions:

```
@prefix ex: <http://ex.org/voc#> .
ex:JohnHeard a ex:Person .
ex:IanZiering a ex:Person .
```

Returns an RDF graph “describing” the returned results. This is an non-normative feature. What should be returned is left open.

SPARQL: Solution Modifiers

Solution modifiers

- `ORDER BY (DESC)`
  - Can be used to order results
  - By default ascending (`ASC`), can specify descending (`DESC`)
  - Can order lexicographically on multiple items
- `LIMIT` *n*
  - Return only *n* results
- `OFFSET` *n*
  - Skip the first *n* results

Strictly speaking, by default, no ordering is applied. Hence OFFSET means nothing without ORDER BY. However, some engines support a default ordering (e.g., the order of computation of results).

How might we ask for the second and third most recently released movies?

```
PREFIX ex: <http://ex.org/voc#>
SELECT ?movie
WHERE { ?movie ex:firstAired ?date . }
ORDER BY DESC(?date)
LIMIT 2
OFFSET 1
```

Solution modifiers

The order of execution is always:
ORDER→ OFFSET → LIMIT
Changing the order of LIMIT/OFFSET makes no difference to the query solutions.

ORDER BY must come before LIMIT/OFFSET according to the query syntax

How might we ask for the second and third most recently released movies?

```
PREFIX ex: <http://ex.org/voc#>
SELECT ?movie
WHERE { ?movie ex:firstAired ?date . }
ORDER BY DESC(?date)
LIMIT 2
OFFSET 1
```

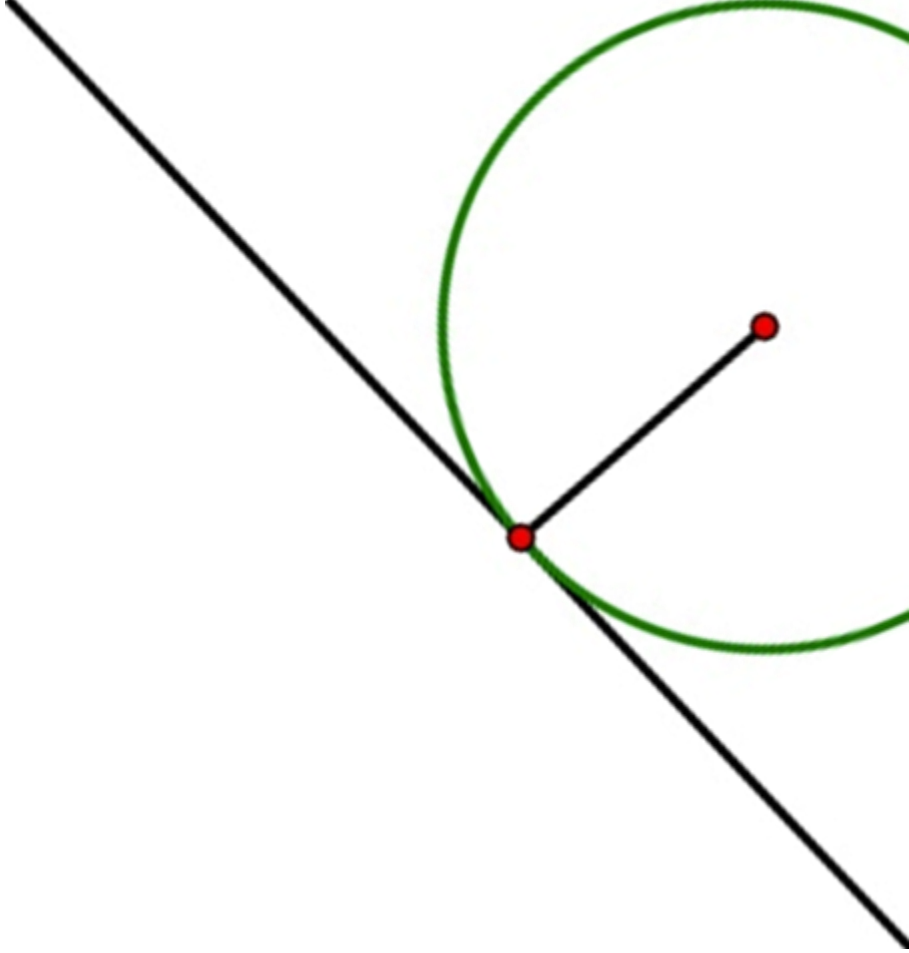
≡

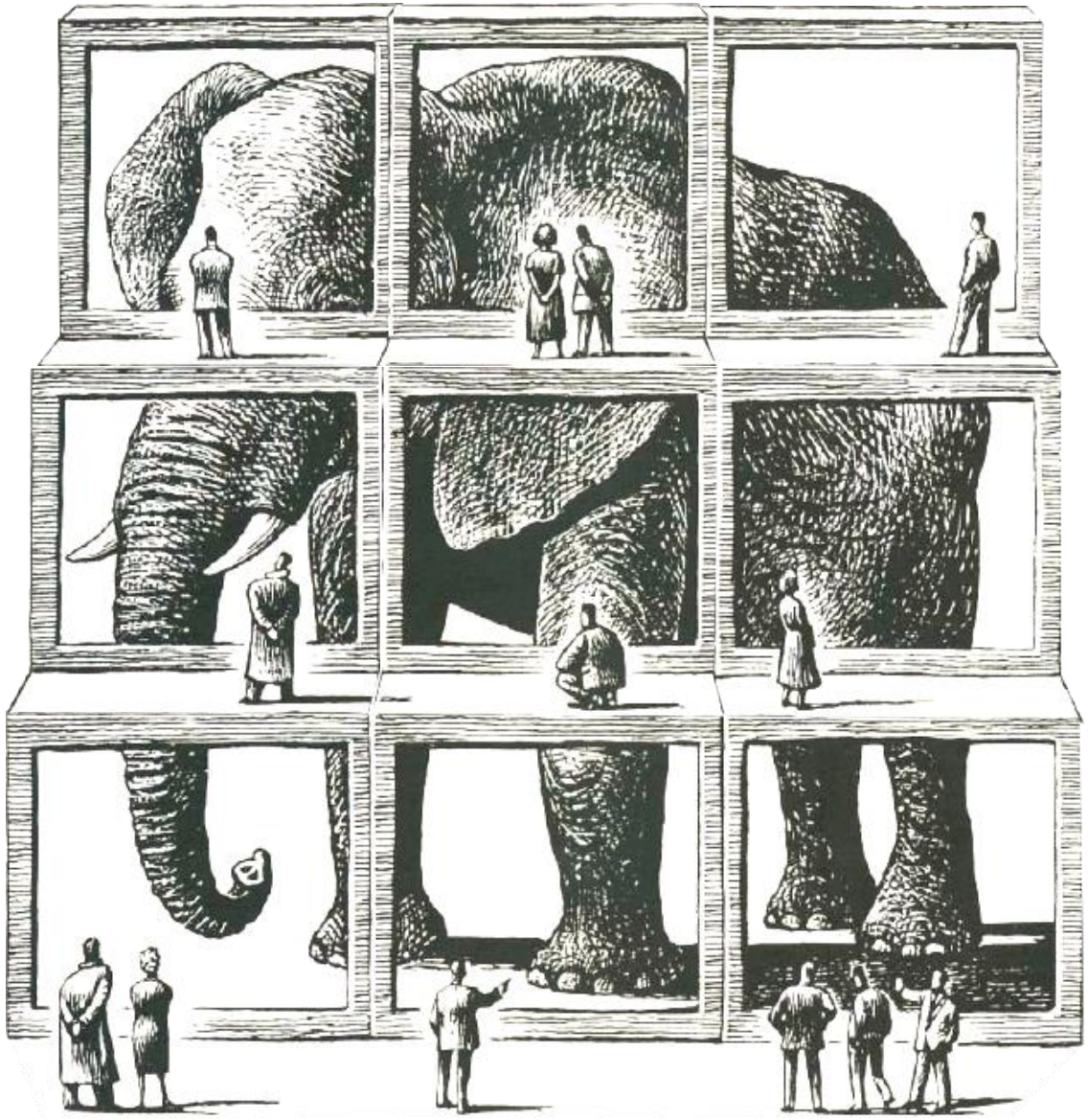
```
PREFIX ex: <http://ex.org/voc#>
SELECT ?movie
WHERE { ?movie ex:firstAired ?date . }
ORDER BY DESC(?date)
OFFSET 1
LIMIT 2
```

SPARQL: Named Graphs

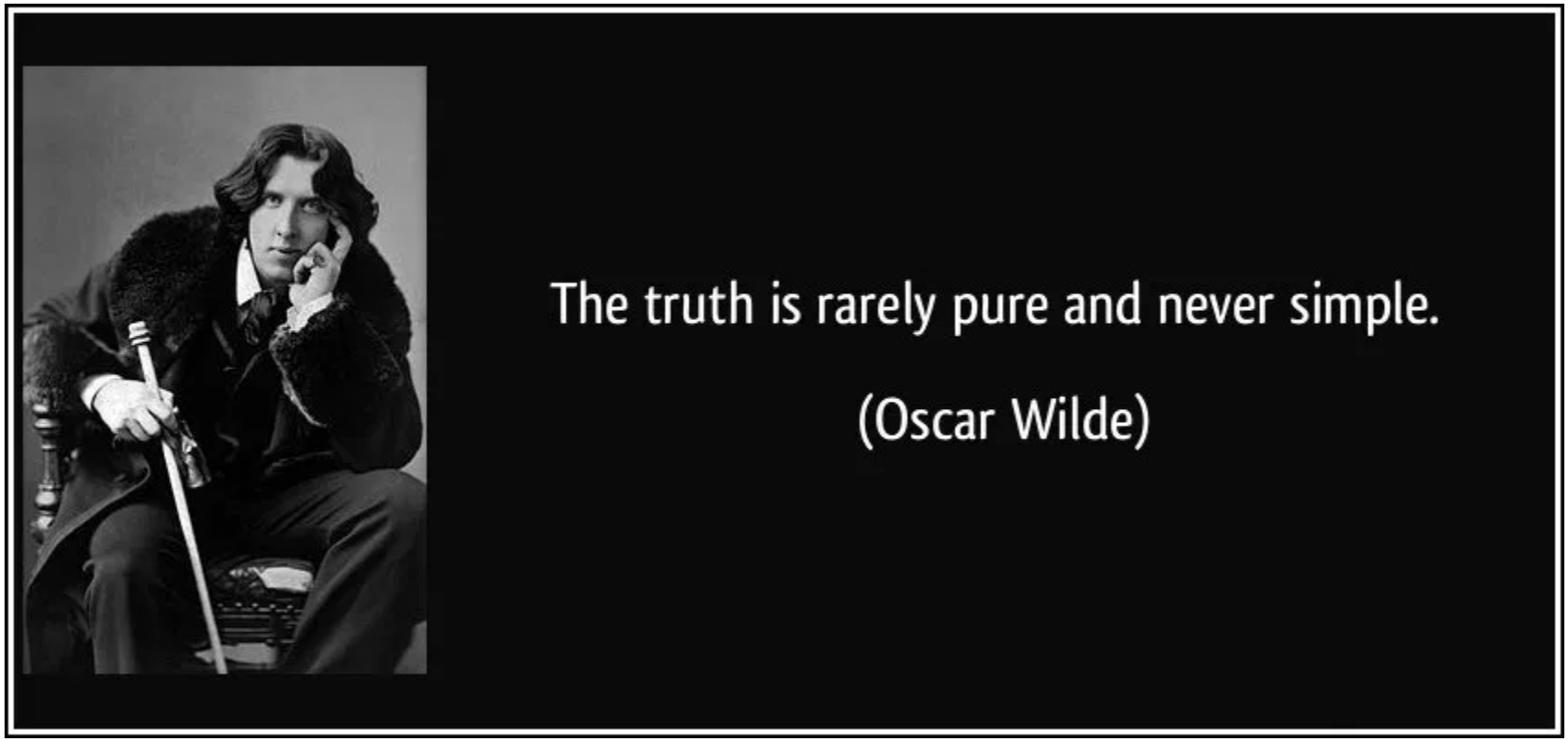
BUT FIRST: CONTEXT

Context ...





Importance of context



Without context

A proposition φ is TRUE.

- `president(Clinton,US)` is TRUE.
- `Illegal(Bitcoin)` is TRUE.
- `bornIn(Obama,Kenya)` is TRUE.

With context

A proposition φ is TRUE *in context* c.

- `president(Clinton,US)` is TRUE.
- `Illegal(Bitcoin)` is TRUE.
- `bornIn(Obama,Kenya)` is TRUE.

With context

A proposition φ is TRUE *in context* c.

- `president(Clinton,US)` is TRUE *in context* `[1993,2001]` (*temporal*).
- `Illegal(Bitcoin)` is TRUE.
- `bornIn(Obama,Kenya)` is TRUE.

With context

A proposition φ is TRUE *in context* c.

- `president(Clinton,US)` is TRUE *in context* `[1993,2001]` (*temporal*).
- `Illegal(Bitcoin)` is TRUE *in context* `Bolivia` (*geographical*).
- `bornIn(Obama,Kenya)` is TRUE.

With context

A proposition φ is TRUE *in context* c.

- `president(Clinton,US)` is TRUE *in context* `[1993,2001]` (*temporal*).
- `Illegal(Bitcoin)` is TRUE *in context* `Bolivia` (*geographical*).
- `bornIn(Obama,Kenya)` is TRUE *in context* `Breitbart` (*provenance*).

With context

A proposition φ is TRUE *in context* c.

- `president(Clinton,US)` is TRUE *in context* `[1993,2001]` (*temporal*).
- `Illegal(Bitcoin)` is TRUE *in context* `Bolivia` (*geographical*).
- `bornIn(Obama,Kenya)` is TRUE *in context* `Breitbart` (*provenance*).
- ...

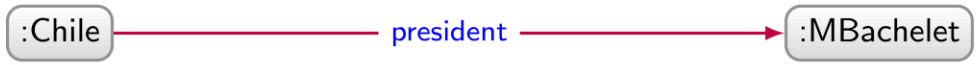
With context

A proposition φ is TRUE *in context* c.

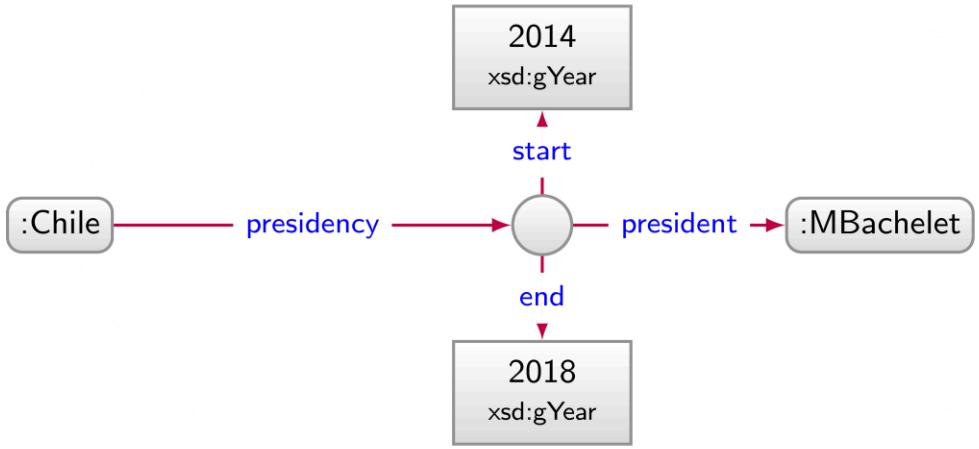
- `president(Clinton,US)` is TRUE *in context* `[1993,2001]` (*temporal*).
- `Illegal(Bitcoin)` is TRUE *in context* `Bolivia` (*geographical*).
- `bornIn(Obama,Kenya)` is TRUE *in context* `Breitbart` (*provenance*).
- ...

Data needs "context"?

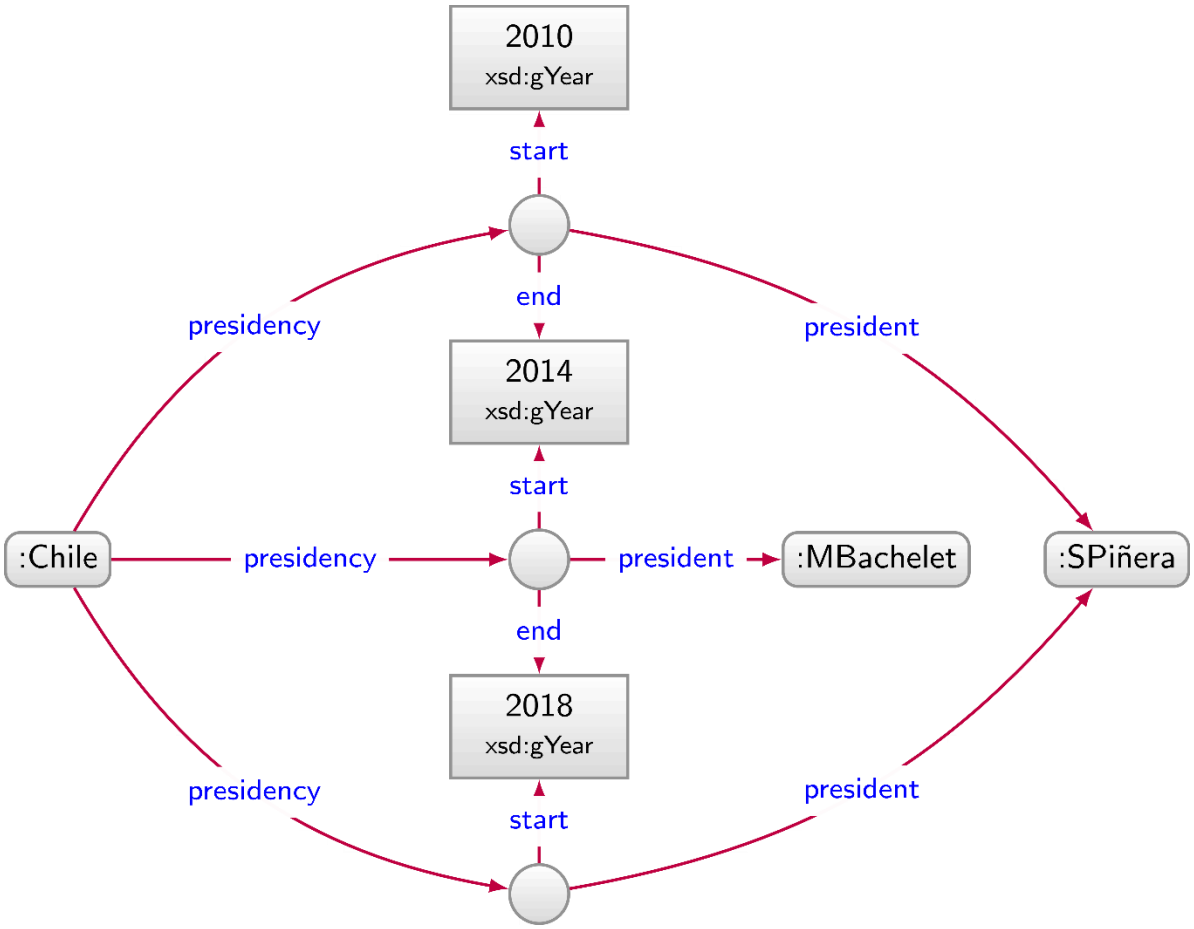
Data needs context?



Data needs context?



Data needs context?



Is this context? Or is this data?

DATACONTEXT

With context

A proposition φ is TRUE *in context* c.

- `president(Clinton,US)` is TRUE *in context* `[1993,2001]` (*temporal*).
- `Illegal(Bitcoin)` is TRUE *in context* `Bolivia` (*geographical*).
- `bornIn(Obama,Kenya)` is TRUE *in context* `Breitbart` (*provenance*).

Without context

A proposition φ is TRUE *in context* c.

- `president(Clinton,US)` is TRUE *in context* `[1993,2001]` (*temporal*).
- `Illegal(Bitcoin)` is TRUE *in context* `Bolivia` (*geographical*).
- `bornIn(Obama,Kenya,Breitbart)` is TRUE.

Without context

A proposition φ is TRUE *in context* c.

- `president(Clinton,US)` is TRUE *in context* `[1993,2001]` (*temporal*).
- `Illegal(Bitcoin,Bolivia)` is TRUE.
- `bornIn(Obama,Kenya,Breitbart)` is TRUE.

Without context

A proposition φ is TRUE *in context* c.

- `president(Clinton,US,1993,2001)` is TRUE.
- `Illegal(Bitcoin,Bolivia)` is TRUE.
- `bornIn(Obama,Kenya,Breitbart)` is TRUE.

Without context

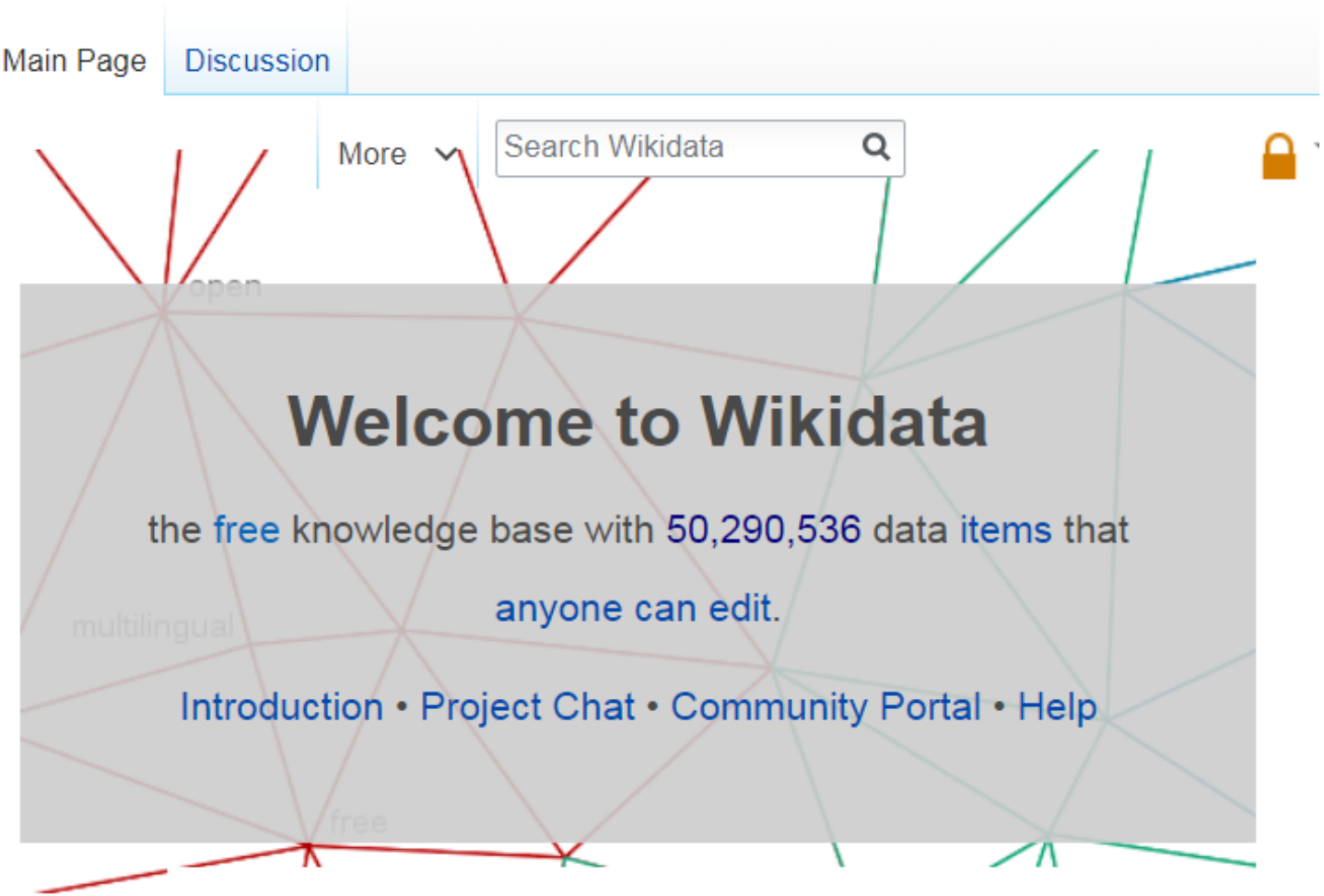
A proposition φ is TRUE *in context* c.

- `president(Clinton,US,1993,2001)` is TRUE.
- `Illegal(Bitcoin,Bolivia)` is TRUE.
- `bornIn(Obama,Kenya,Breitbart)` is TRUE.

But how to represent as a graph?

Context as graph data: Representation

Wikidata


Main Page
Discussion
More
Search Wikidata
Welcome to Wikidata
the free knowledge base with 50,290,536 data items that
anyone can edit.
Introduction • Project Chat • Community Portal • Help

Wikidata: Qualifiers

Sebastian Piñera (Q306)

35th & 37th President of Chile

Miguel Juan Sebastián Piñera Echenique | Sebastián Piñera Echenique

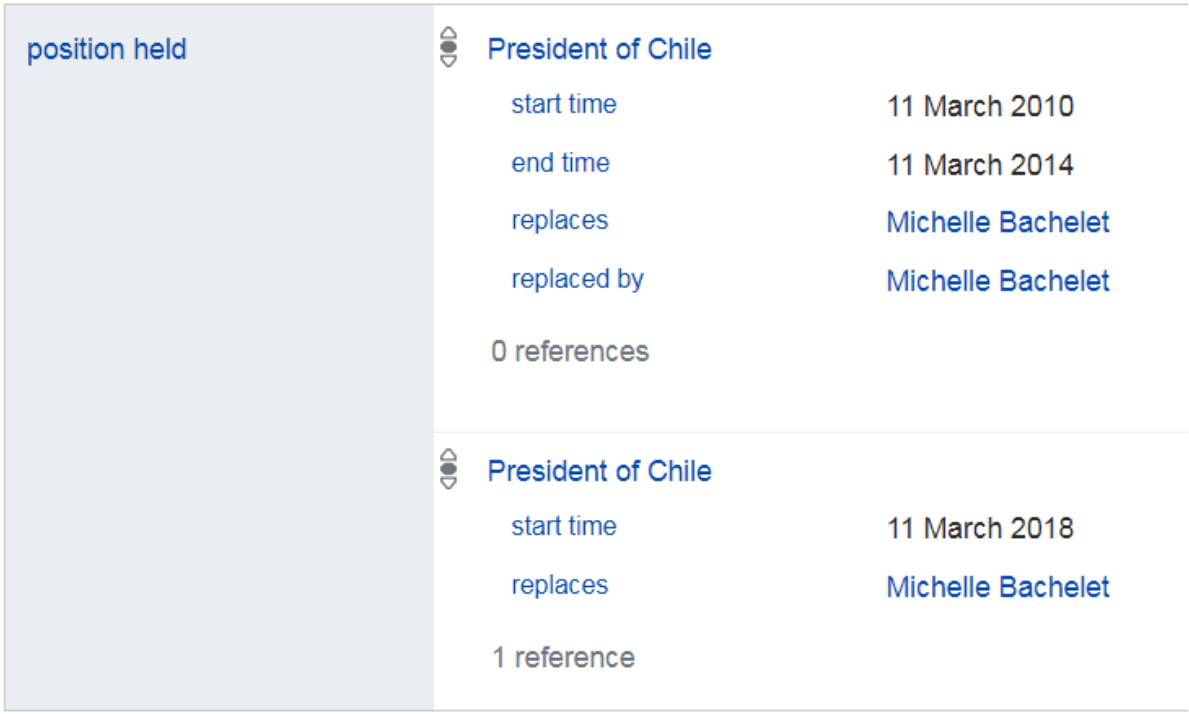
position held	President of Chile	
	start time	11 March 2010
	end time	11 March 2014
	replaces	Michelle Bachelet
	replaced by	Michelle Bachelet
	0 references	
	President of Chile	
	start time	11 March 2018
	replaces	Michelle Bachelet
	1 reference	

Qualified Statements: Quins

S	P	O	Q	V
:SPiñera	:president	:Chile	:start	"2010"^^xsd:gYear
:SPiñera	:president	:Chile	:end	"2014"^^xsd:gYear
:SPiñera	:president	:Chile	:replaces	:MBachelet
:SPiñera	:president	:Chile	:replacedBy	:MBachelet

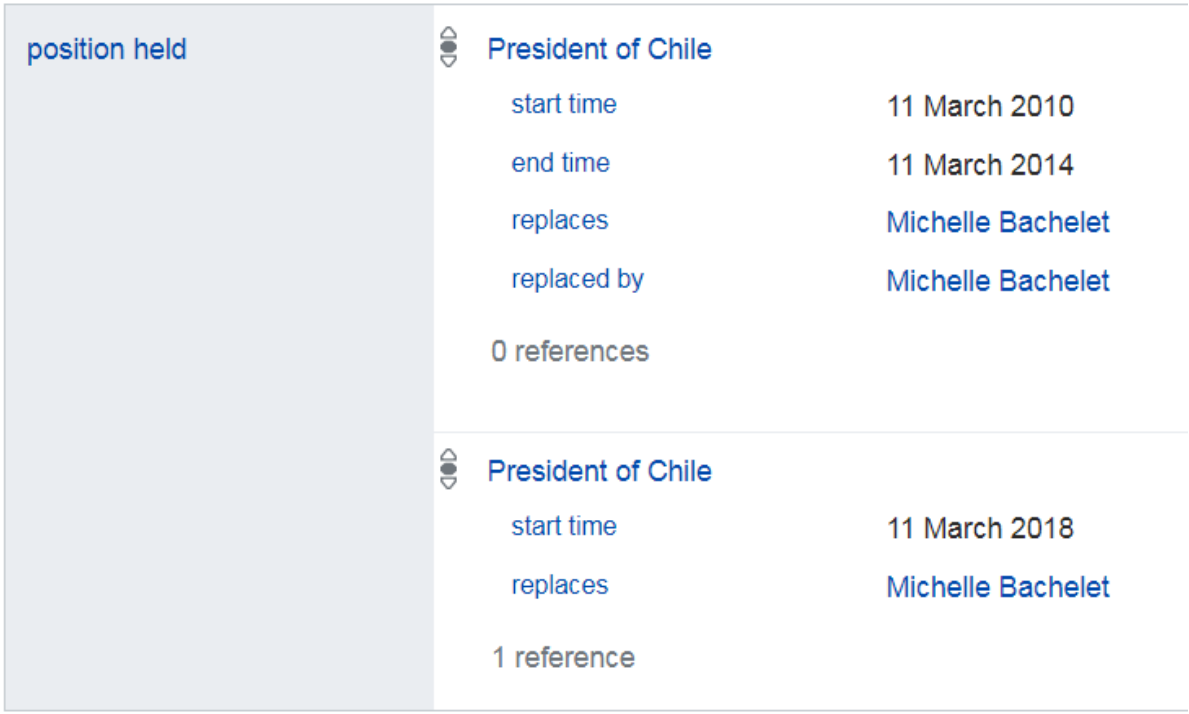
Qualified Statements: Quins

S	P	O	Q	V
:SPiñera	:president	:Chile	:start	"2010"^^xsd:gYear
:SPiñera	:president	:Chile	:end	"2014"^^xsd:gYear
:SPiñera	:president	:Chile	:replaces	:MBachelet
:SPiñera	:president	:Chile	:replacedBy	:MBachelet
:SPiñera	:president	:Chile	:replaces	:MBachelet
:SPiñera	:president	:Chile	:start	"2018"^^xsd:gYear



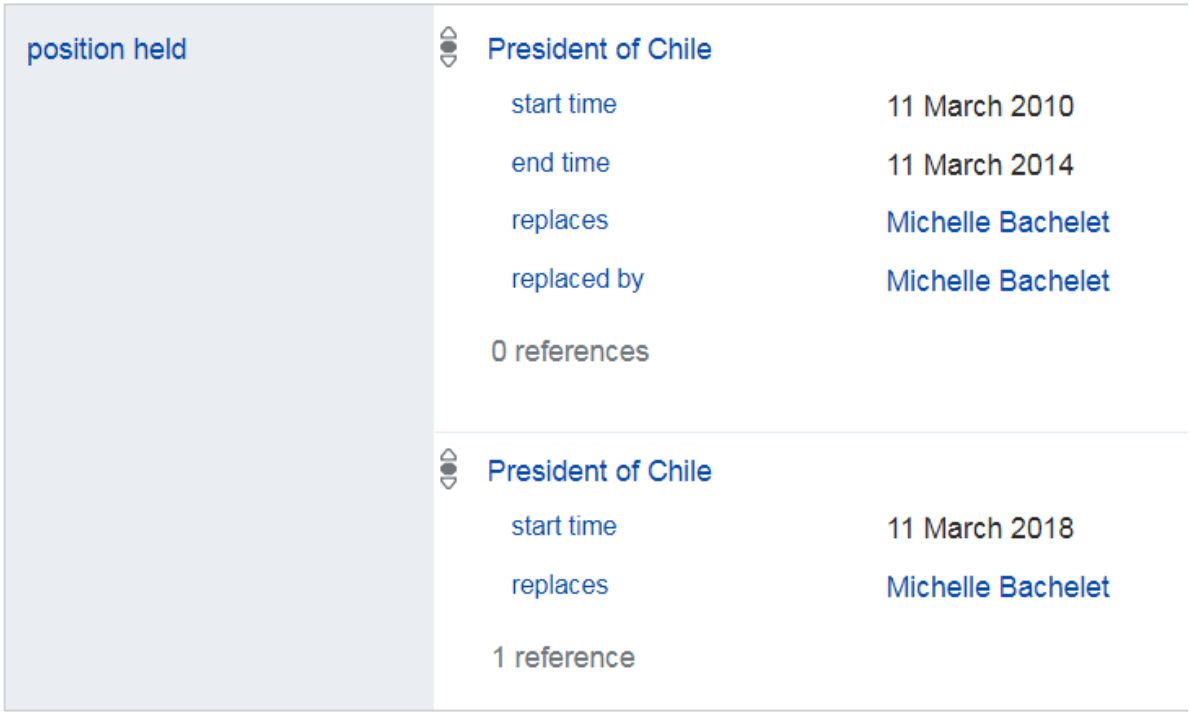
Qualified Statements: ~~Quins~~

S	P	O	Q	V
:SPiñera	:president	:Chile	:start	"2010"^^xsd:gYear
:SPiñera	:president	:Chile	:end	"2014"^^xsd:gYear
:SPiñera	:president	:Chile	:replaces	:MBachelet
:SPiñera	:president	:Chile	:replacedBy	:MBachelet
:SPiñera	:president	:Chile	:replaces	:MBachelet
:SPiñera	:president	:Chile	:start	"2018"^^xsd:gYear



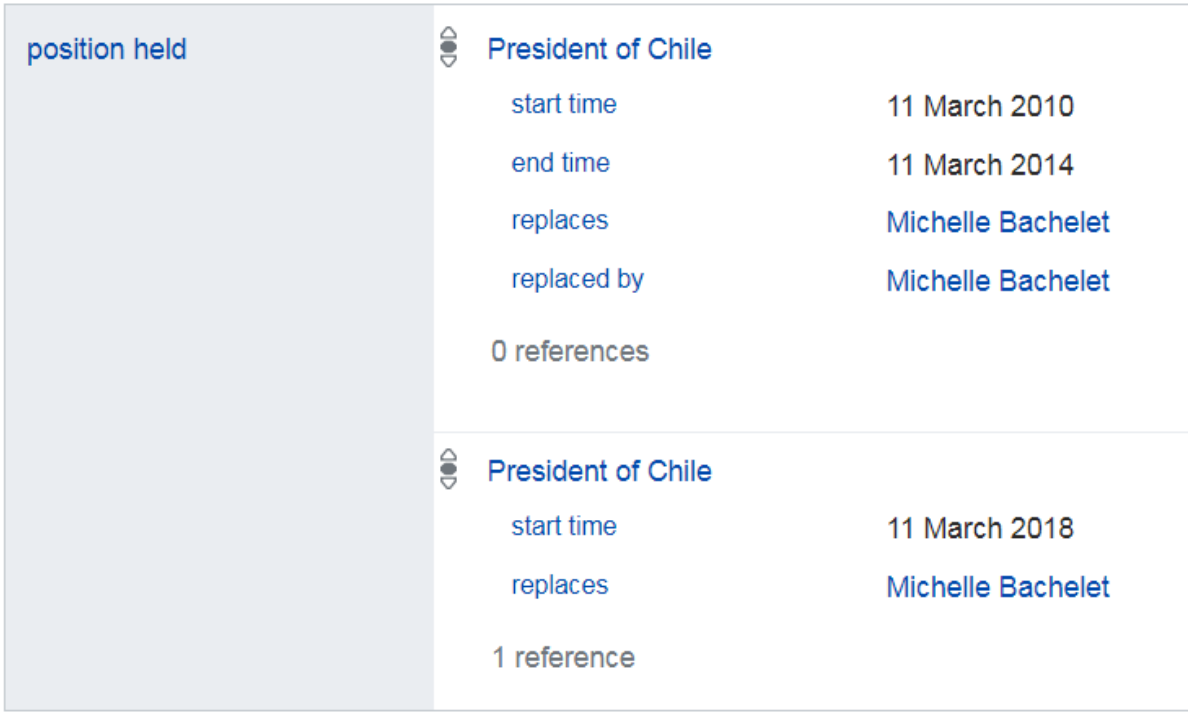
Qualified Statements: Sextuples

S	P	O	Q	V	E
:SPiñera	:president	:Chile	:start	"2010"^^xsd:gYear	:E1
:SPiñera	:president	:Chile	:end	"2014"^^xsd:gYear	:E1
:SPiñera	:president	:Chile	:replaces	:MBachelet	:E1
:SPiñera	:president	:Chile	:replacedBy	:MBachelet	:E1
:SPiñera	:president	:Chile	:replaces	:MBachelet	:E2
:SPiñera	:president	:Chile	:start	"2018"^^xsd:gYear	:E2



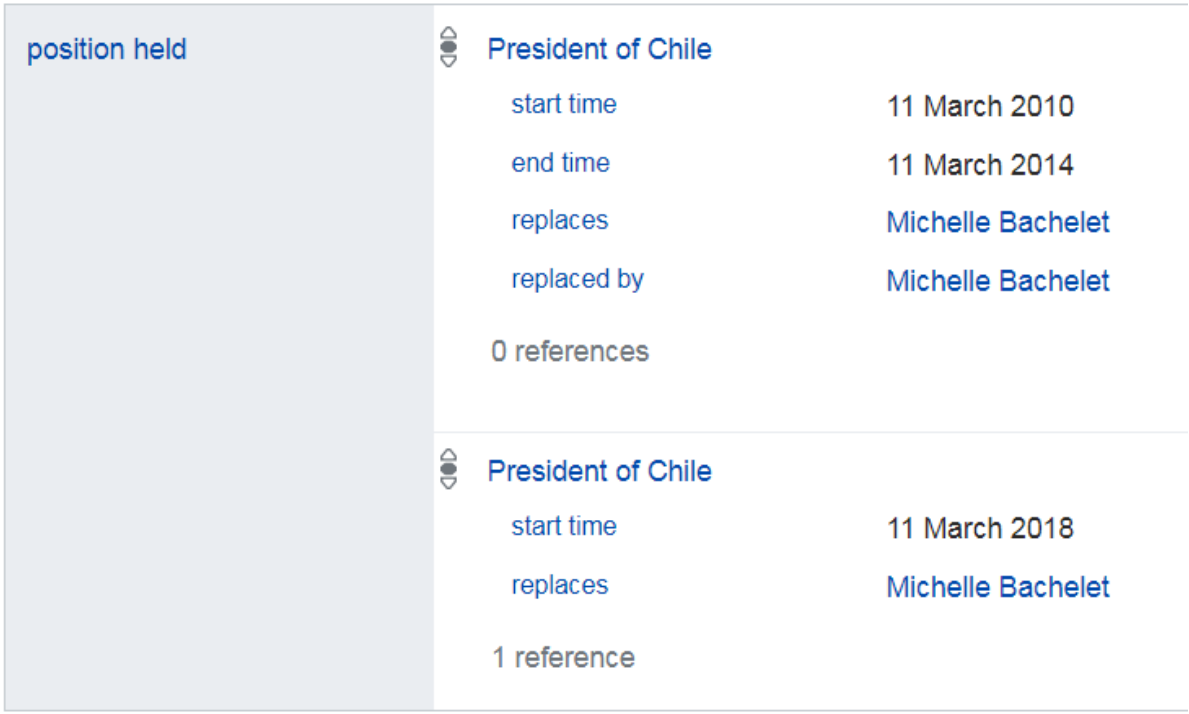
Qualified Statements: Sextuples

S	P	O	Q	V	E
:SPiñera	:president	:Chile	:start	"2010"^^xsd:gYear	:E1
:SPiñera	:president	:Chile	:end	"2014"^^xsd:gYear	:E1
:SPiñera	:president	:Chile	:replaces	:MBachelet	:E1
:SPiñera	:president	:Chile	:replacedBy	:MBachelet	:E1
:SPiñera	:president	:Chile	:replaces	:MBachelet	:E2
:SPiñera	:president	:Chile	:start	"2018"^^xsd:gYear	:E2



Qualified Statements: Sextuples

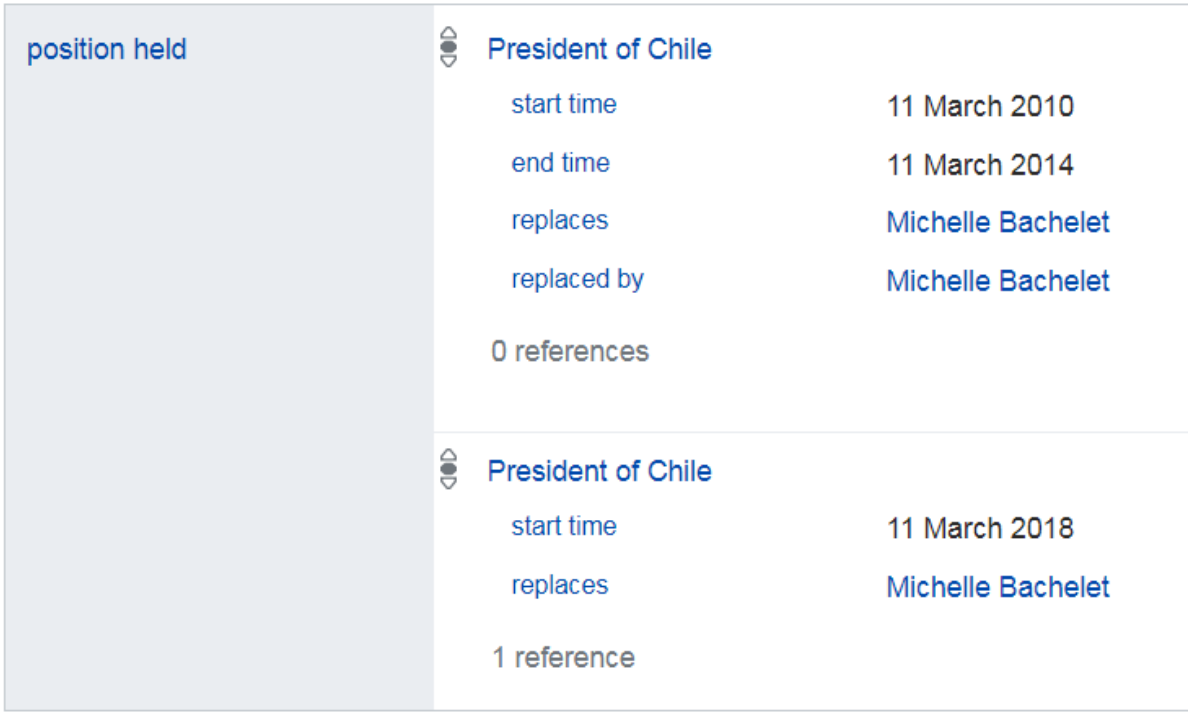
S	P	O	Q	V	E
:SPiñera	:president	:Chile	:start	"2010"^^xsd:gYear	:E1
:SPiñera	:president	:Chile	:end	"2014"^^xsd:gYear	:E1
:SPiñera	:president	:Chile	:replaces	:MBachelet	:E1
:SPiñera	:president	:Chile	:replacedBy	:MBachelet	:E1
:SPiñera	:president	:Chile	:replaces	:MBachelet	:E2
:SPiñera	:president	:Chile	:start	"2018"^^xsd:gYear	:E2



Qualified Statements: Quads + Triples

S	P	O	E
:SPiñera	:president	:Chile	:E1
:SPiñera	:president	:Chile	:E2

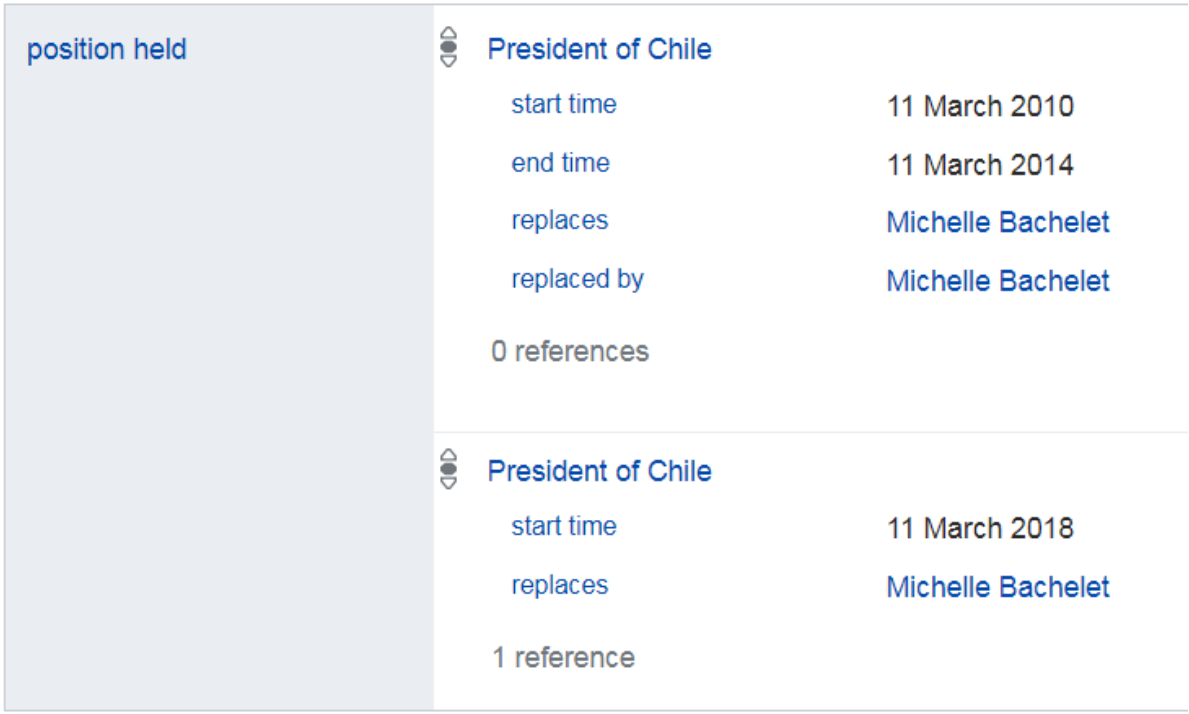
Q	V	E
:start	"2010"^^xsd:gYear	:E1
:end	"2014"^^xsd:gYear	:E1
:replaces	:MBachelet	:E1
:replacedBy	:MBachelet	:E1
:replaces	:MBachelet	:E2
:start	"2018"^^xsd:gYear	:E2



Qualified Statements: Quads + Triples

S	P	O	E
:SPiñera	:president	:Chile	:E1
:SPiñera	:president	:Chile	:E2

E	Q	V
:E1	:start	"2010"^^xsd:gYear
:E1	:end	"2014"^^xsd:gYear
:E1	:replaces	:MBachelet
:E1	:replacedBy	:MBachelet
:E2	:replaces	:MBachelet
:E2	:start	"2018"^^xsd:gYear



Qualified Statements: Quads + Triples

S	P	O	E
:SPiñera	:president	:Chile	:E1
:SPiñera	:president	:Chile	:E2

How to represent this as a graph?

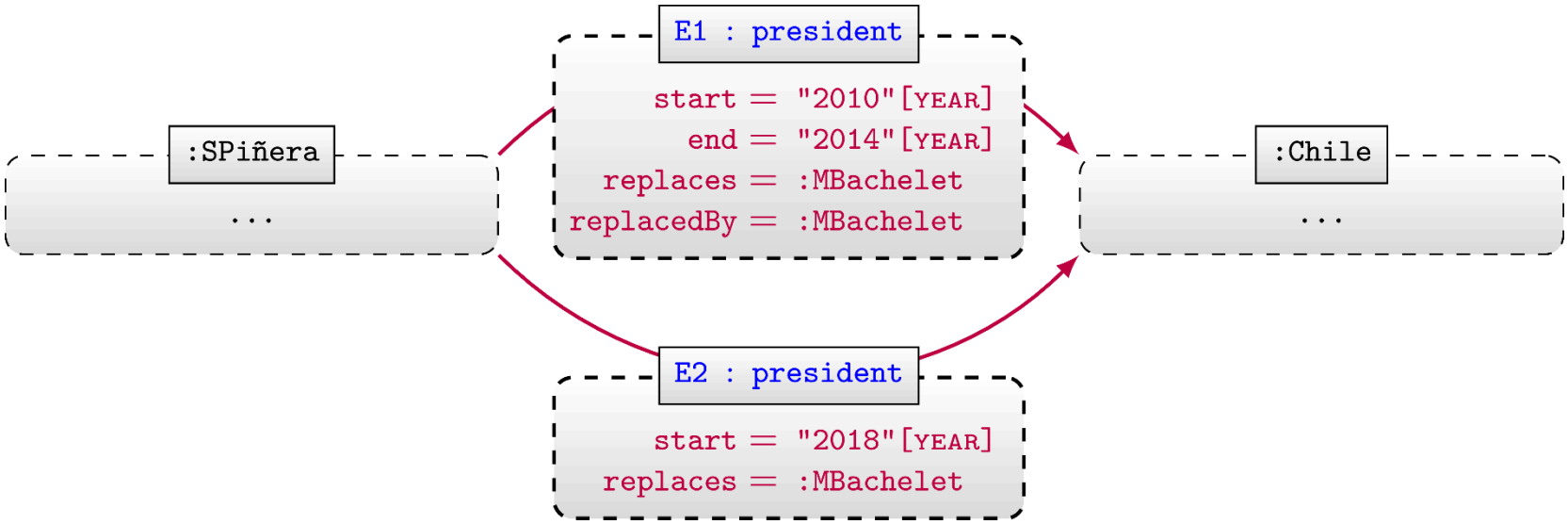
E	Q	V
:E1	:start	"2010"^^xsd:gYear
:E1	:end	"2014"^^xsd:gYear
:E1	:replaces	:MBachelet
:E1	:replacedBy	:MBachelet
:E2	:replaces	:MBachelet
:E2	:start	"2018"^^xsd:gYear

Property Graph

S	P	O	E
:SPiñera	:president	:Chile	:E1
:SPiñera	:president	:Chile	:E2

E	Q	V
:E1	:start	"2010"^^xsd:gYear
:E1	:end	"2014"^^xsd:gYear
:E1	:replaces	:MBachelet
:E1	:replacedBy	:MBachelet
:E2	:replaces	:MBachelet
:E2	:start	"2018"^^xsd:gYear

How to represent this as a graph?

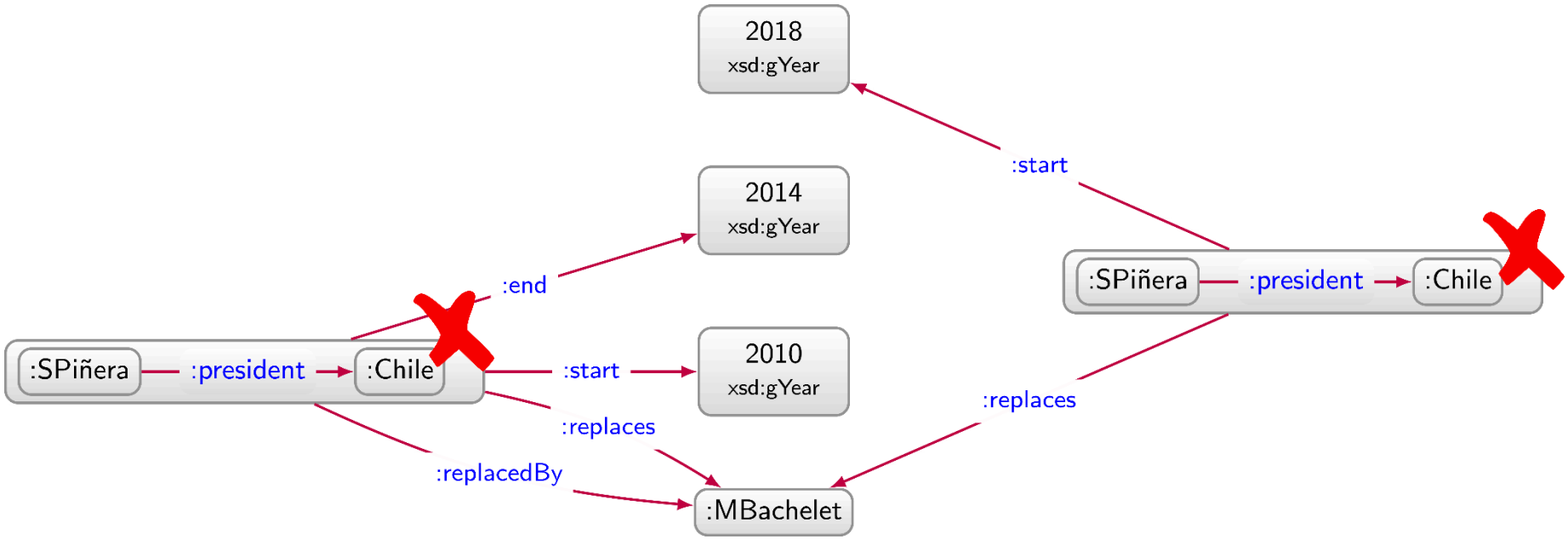


RDF*

S	P	O	E
:SPiñera	:president	:Chile	:E1
:SPiñera	:president	:Chile	:E2

How to represent this as a graph?

E	Q	V
:E1	:start	"2010"^^xsd:gYear
:E1	:end	"2014"^^xsd:gYear
:E1	:replaces	:MBachelet
:E1	:replacedBy	:MBachelet
:E2	:replaces	:MBachelet
:E2	:start	"2018"^^xsd:gYear



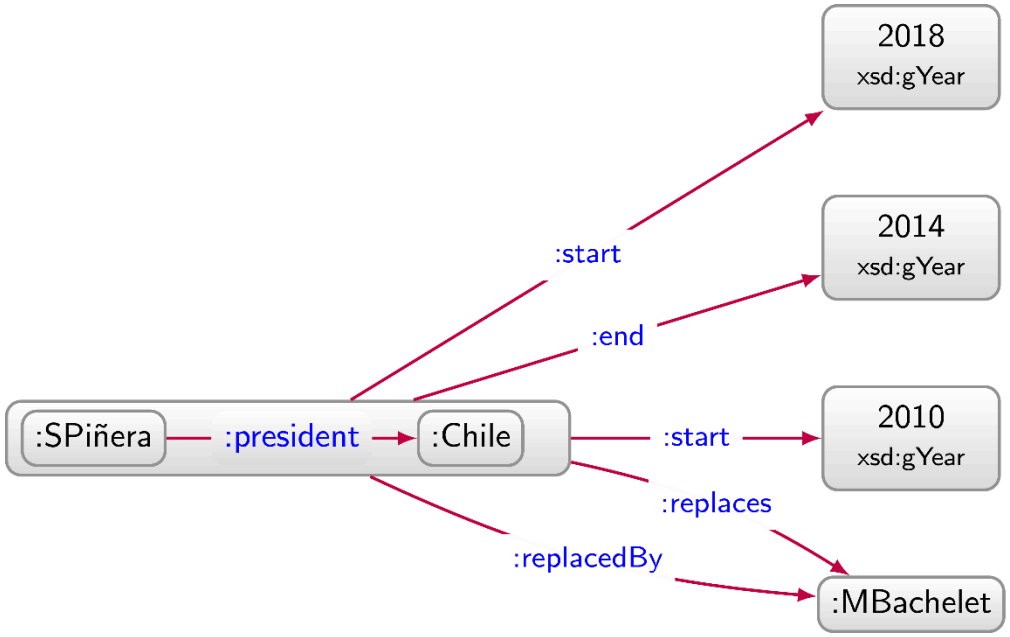
[Hartig 17]

RDF*

S	P	O	E
:SPiñera	:president	:Chile	:E1
:SPiñera	:president	:Chile	:E2

How to represent this as a graph?

E	Q	V
:E1	:start	"2010"^^xsd:gYear
:E1	:end	"2014"^^xsd:gYear
:E1	:replaces	:MBachelet
:E1	:replacedBy	:MBachelet
:E2	:replaces	:MBachelet
:E2	:start	"2018"^^xsd:gYear



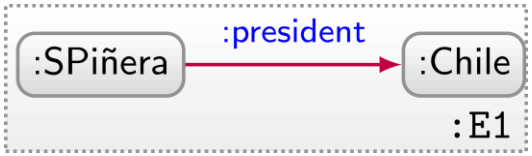
[Hartig 17]

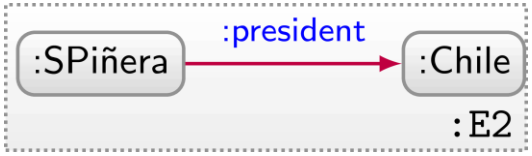
Named Graphs

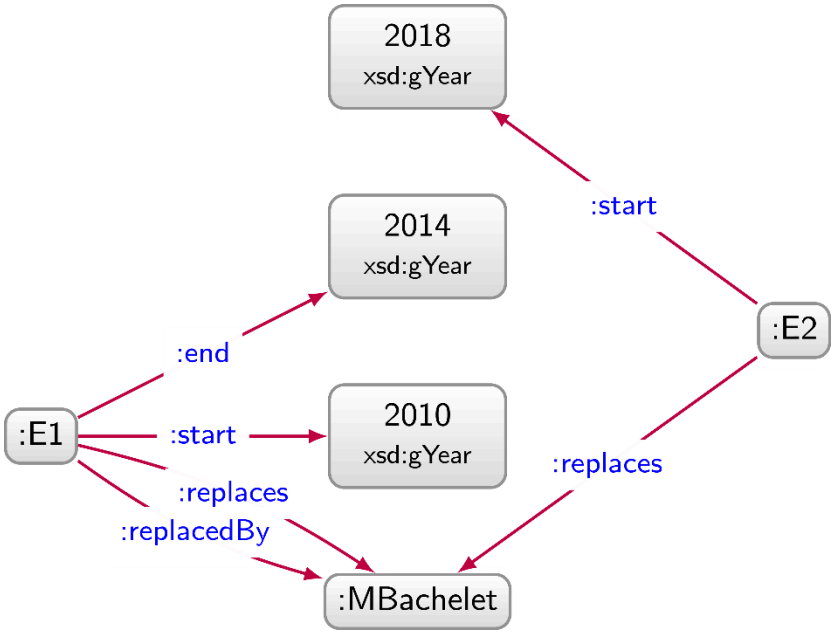
S	P	O	E
:SPiñera	:president	:Chile	:E1
:SPiñera	:president	:Chile	:E2

How to represent this as a graph?

E	Q	V
:E1	:start	"2010"^^xsd:gYear
:E1	:end	"2014"^^xsd:gYear
:E1	:replaces	:MBachelet
:E1	:replacedBy	:MBachelet
:E2	:replaces	:MBachelet
:E2	:start	"2018"^^xsd:gYear



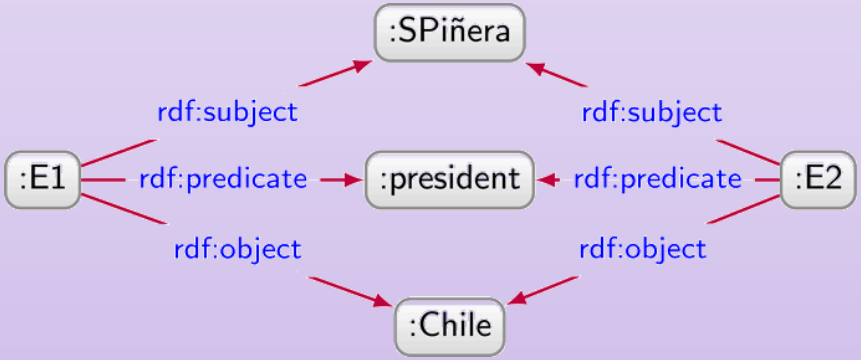




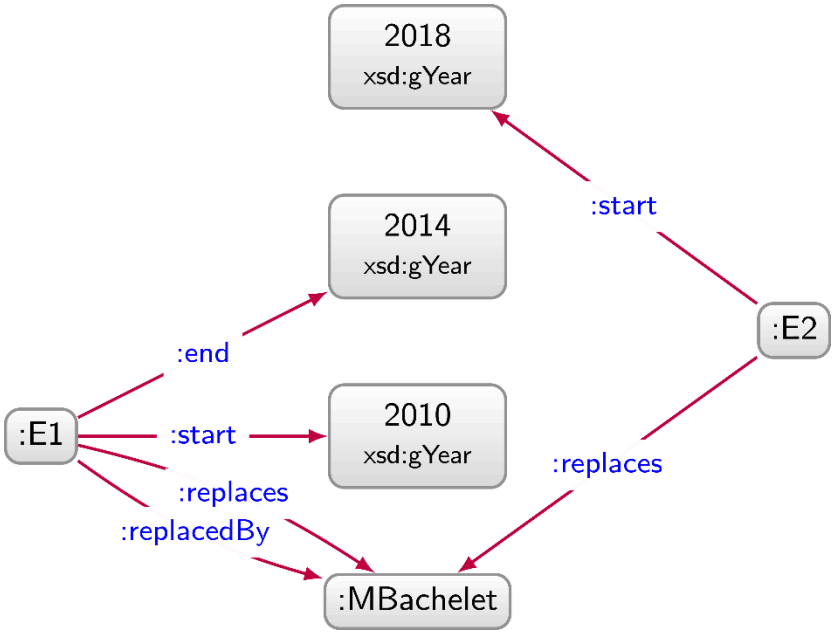
RDF Reification

S	P	O	E
:SPiñera	:president	:Chile	:E1
:SPiñera	:president	:Chile	:E2

How to represent this as a graph?



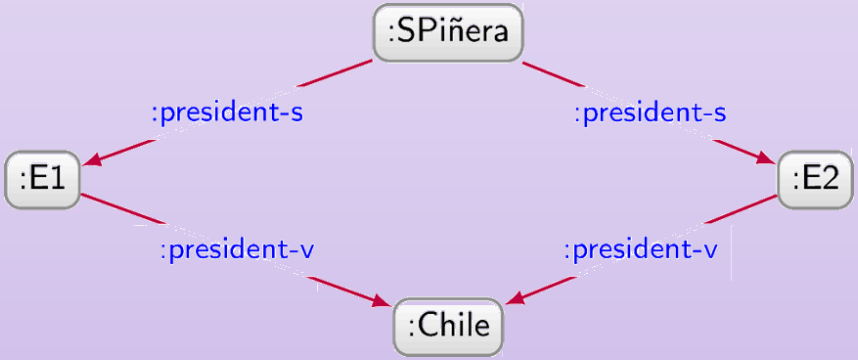
E	Q	V
:E1	:start	"2010"^^xsd:gYear
:E1	:end	"2014"^^xsd:gYear
:E1	:replaces	:MBachelet
:E1	:replacedBy	:MBachelet
:E2	:replaces	:MBachelet
:E2	:start	"2018"^^xsd:gYear

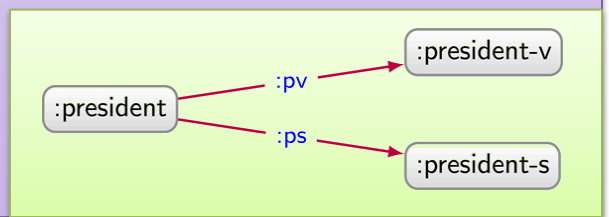


n-ary Relations

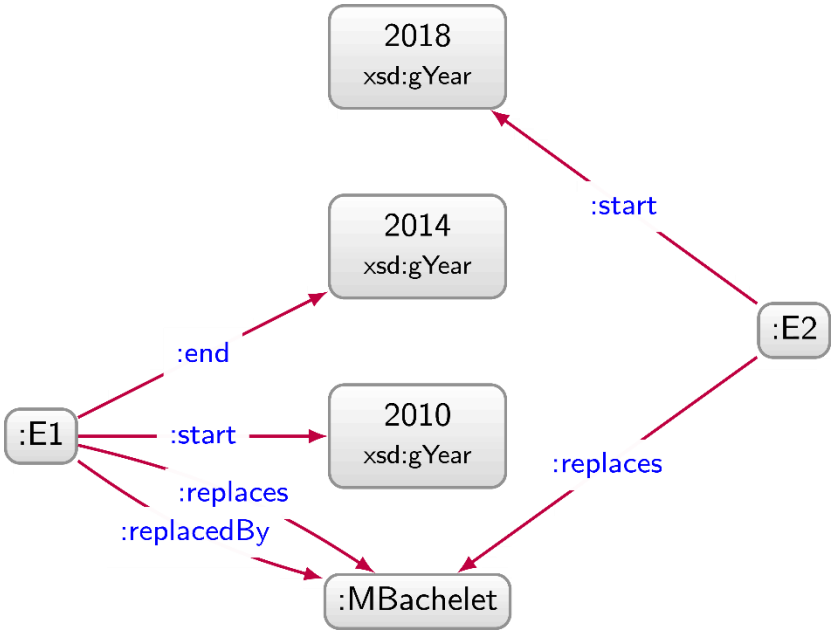
S	P	O	E
:SPiñera	:president	:Chile	:E1
:SPiñera	:president	:Chile	:E2

How to represent this as a graph?





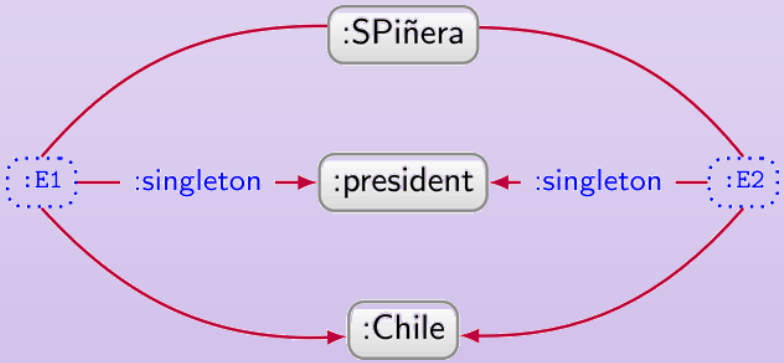
E	Q	V
:E1	:start	"2010"^^xsd:gYear
:E1	:end	"2014"^^xsd:gYear
:E1	:replaces	:MBachelet
:E1	:replacedBy	:MBachelet
:E2	:replaces	:MBachelet
:E2	:start	"2018"^^xsd:gYear



Singleton Properties

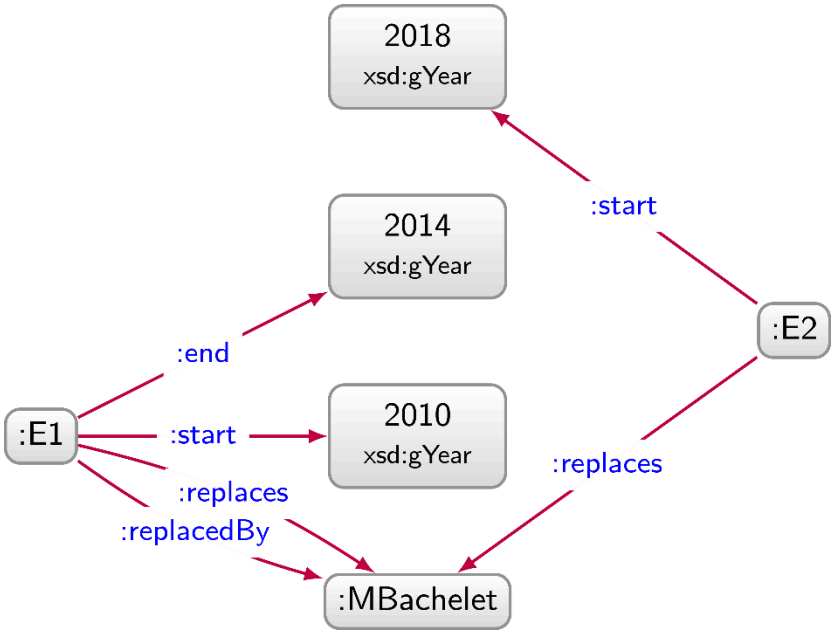
S	P	O	E
:SPiñera	:president	:Chile	:E1
:SPiñera	:president	:Chile	:E2

How to represent this as a graph?



[Nguyen et al. 14]

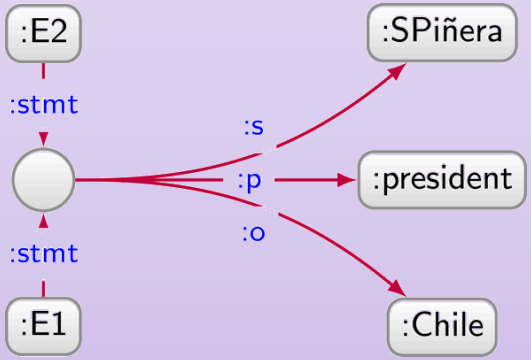
E	Q	V
:E1	:start	"2010"^^xsd:gYear
:E1	:end	"2014"^^xsd:gYear
:E1	:replaces	:MBachelet
:E1	:replacedBy	:MBachelet
:E2	:replaces	:MBachelet
:E2	:start	"2018"^^xsd:gYear



Hoganification

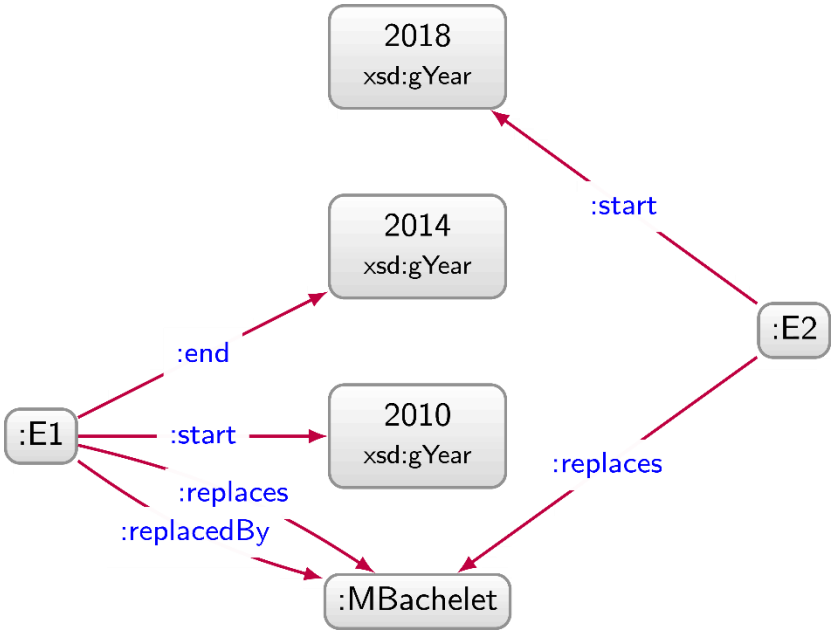
S	P	O	E
:SPiñera	:president	:Chile	:E1
:SPiñera	:president	:Chile	:E2

How to represent this as a graph?



[Hogan 18]

E	Q	V
:E1	:start	"2010"^^xsd:gYear
:E1	:end	"2014"^^xsd:gYear
:E1	:replaces	:MBachelet
:E1	:replacedBy	:MBachelet
:E2	:replaces	:MBachelet
:E2	:start	"2018"^^xsd:gYear



Qualified Statements (quads + triples)

- Beyond RDF graphs
  - Property graphs
  - RDF*
  - Named Graphs
- Good ol' RDF graphs
  - Reification
  - *n*-ary relations
  - Singleton properties
  - Hoganification

Qualified Statements (quads + triples)

- Beyond RDF graphs
  - Property graphs
  - RDF*
  - Named Graphs
- Good ol' RDF graphs
  - Reification
  - *n*-ary relations
  - Singleton properties
  - Hoganification

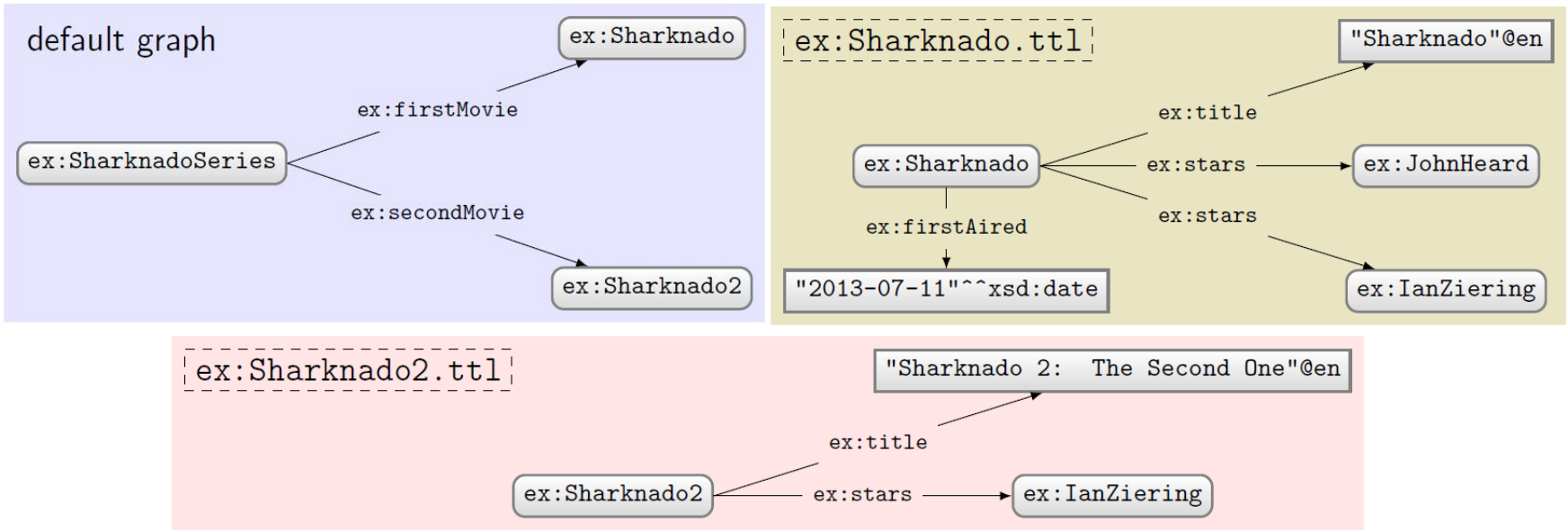
SPARQL: Named Graphs

SPARQL defined over a Dataset

- A dataset $D = \{G, (G_1, n_1), \ldots, (G_k, n_k)\}$
- $G, G_1, \ldots, G_k$ are RDF graphs
- $n_1, \ldots, n_k$ are IRIs
- G is called the **default graph**
- each (G_i, n_i) is a **named graph** $(1 \leq i \leq n)$

Core idea: SPARQL can support multiple RDF graphs, not just one.
When using SPARQL, you can partition your data into multiple graphs.
The default graph is chosen if you don't specify a graph.
Otherwise you can explicitly select a named graph using it's IRI name.

An example dataset



Creating a dataset for a query

- Say an index has dataset $D = \{G, (G_1, n_1), \ldots, (G_k, n_k)\}$
- A query can pick an active dataset from the named graphs

- `FROM`
  - Used to define a default graph for the query using graph names
  - If multiple graphs are specified, they are RDF-merged
- `FROM NAMED`
  - Used to select the active named graphs to be used for the query

Using either feature clears the index dataset

Creating a dataset for a query

- Indexed dataset: $D = \{G, (G_1, n_1), \ldots, (G_k, n_k)\}$
- Query dataset (no `FROM`/`FROM NAMED`): D
- Query dataset D' (example 1):

```
FROM n1
FROM n2
FROM NAMED n3
FROM NAMED n4
```

$\rightarrow D' = \{G_1 \uplus G_2, (G_3, n_3), (G_4, n_4)\}$

- Query dataset D' (example 2):

```
FROM n1
FROM n2
```

$\rightarrow D' = \{G_1 \uplus G_2\}$

($\uplus$ indicates RDF merge)

Creating a dataset for a query

- Indexed dataset: $D = \{G, (G_1, n_1), \ldots, (G_k, n_k)\}$
- Query dataset (no `FROM`/`FROM NAMED`): D
- Query dataset D' (example 1):

```
FROM n1
FROM n2
FROM NAMED n3
FROM NAMED n4
```

$$\rightarrow D' = \{G_1 \uplus G_2, (G_3, n_3), (G_4, n_4)\}$$

- Query dataset D' (example 2):

```
FROM n1
FROM n2
```

$$\rightarrow D' = \{G_1 \uplus G_2\}$$

($\uplus$ indicates RDF merge)

Creating a dataset for a query

- Indexed dataset: $D = \{G, (G_1, n_1), \ldots, (G_k, n_k)\}$
- Query dataset (no `FROM`/`FROM NAMED`): D
- Query dataset D' (example 1):

```
FROM n1
FROM n2
FROM NAMED n3
FROM NAMED n4
```

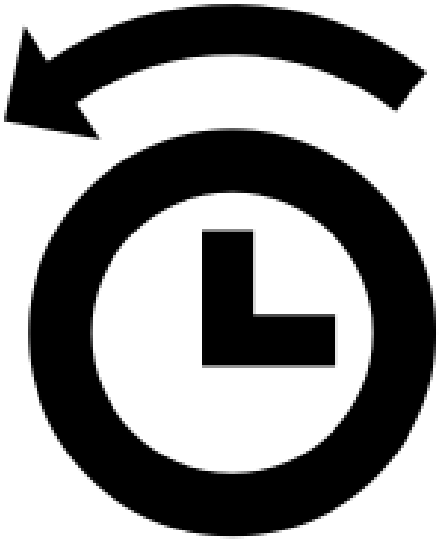
$\rightarrow D' = \{G_1 \uplus G_2, (G_3, n_3), (G_4, n_4)\}$

- Query dataset D' (example 2):

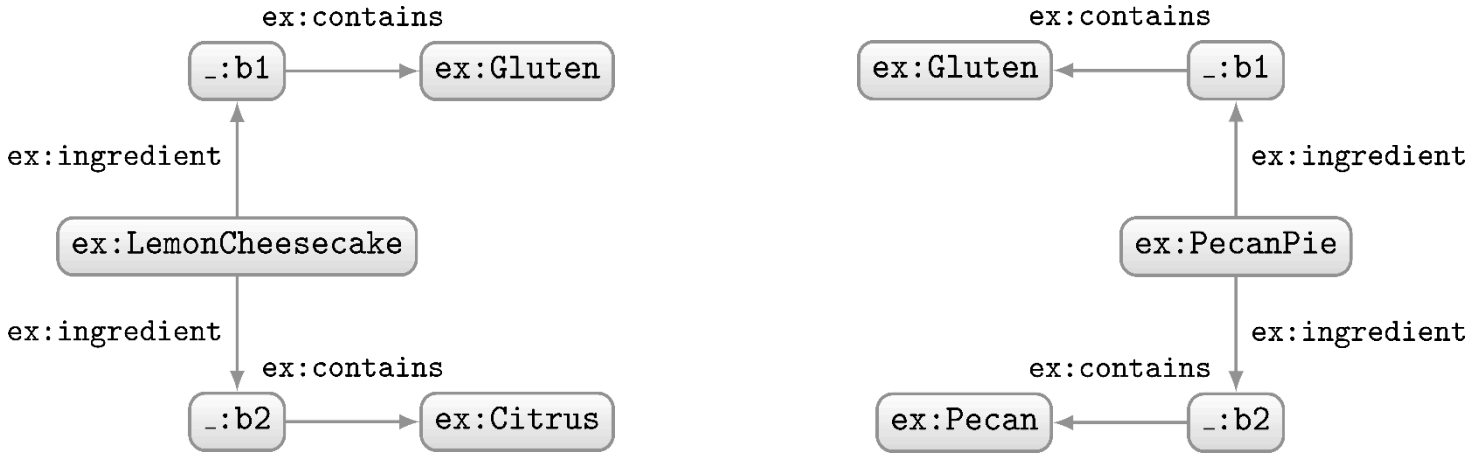
```
FROM n1
FROM n2
```

$\rightarrow D' = \{G_1 \uplus G_2\}$

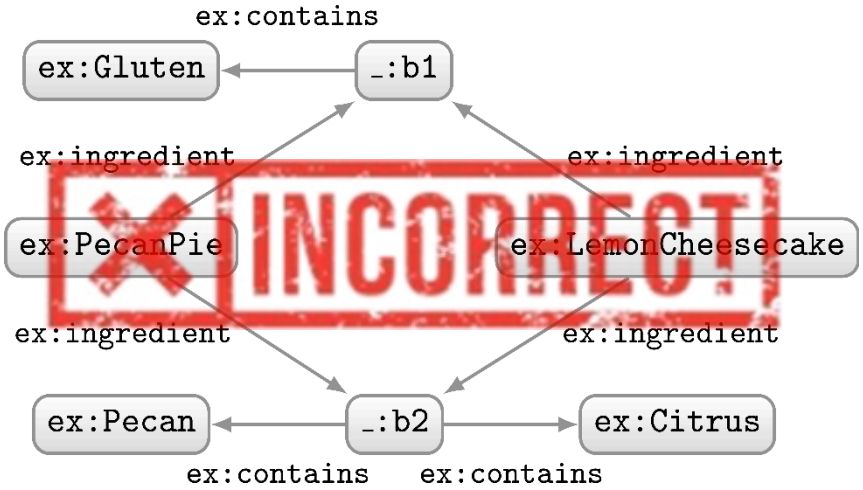
($\uplus$ indicates RDF merge)



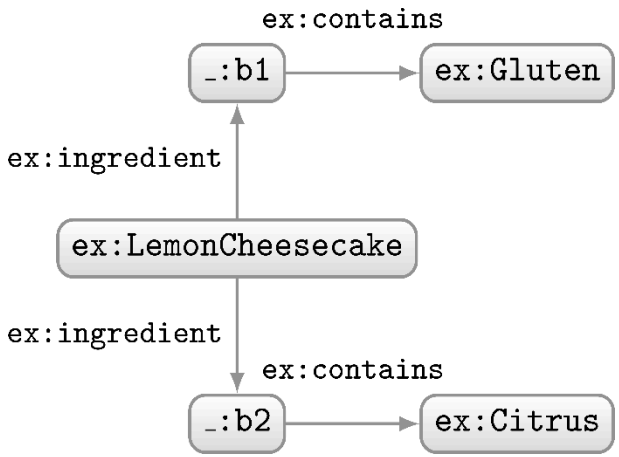
RDF merge: A quick reminder

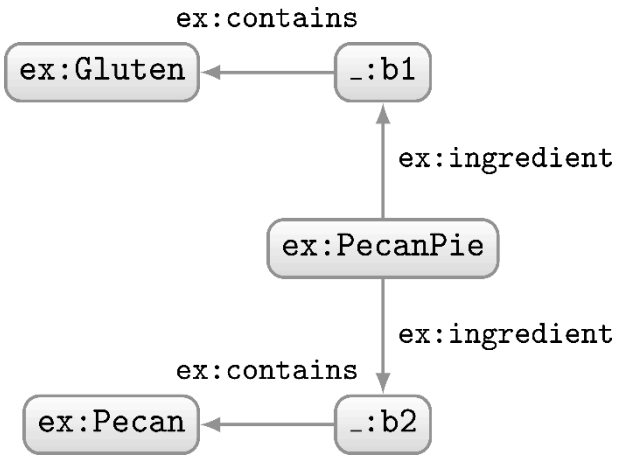


How should we combine these two RDF graphs?

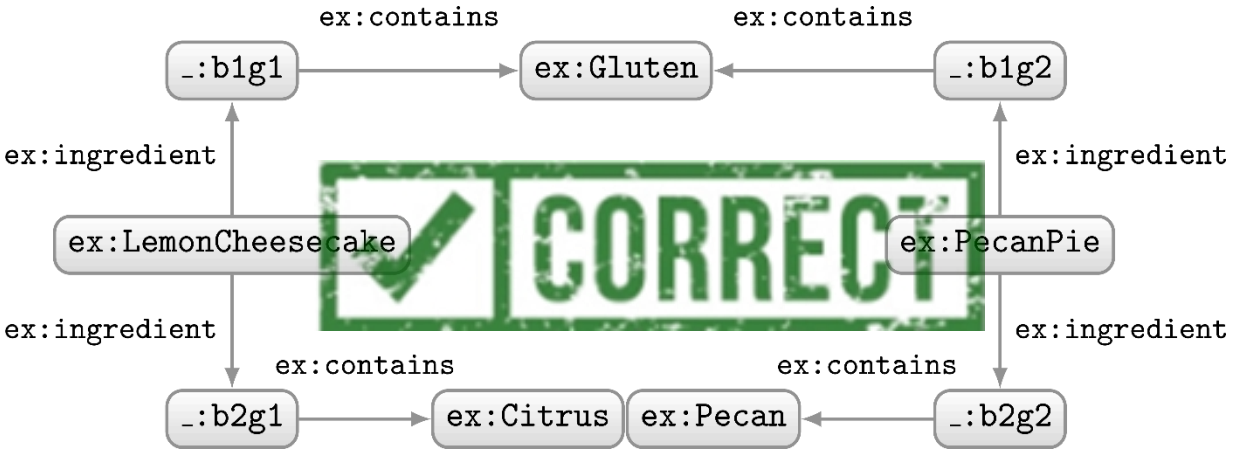


RDF Merge: A quick reminder





How should we combine these two RDF graphs?



Creating a dataset for a query

- Indexed dataset: $D = \{G, (G_1, n_1), \ldots, (G_k, n_k)\}$
- Query dataset (no `FROM`/`FROM NAMED`): D
- Query dataset D' (example 1):

```
FROM n1
FROM n2
FROM NAMED n3
FROM NAMED n4
```

$\rightarrow D' = \{G_1 \uplus G_2, (G_3, n_3), (G_4, n_4)\}$

- Query dataset D' (example 2):

```
FROM n1
FROM n2
```

$\rightarrow D' = \{G_1 \uplus G_2\}$

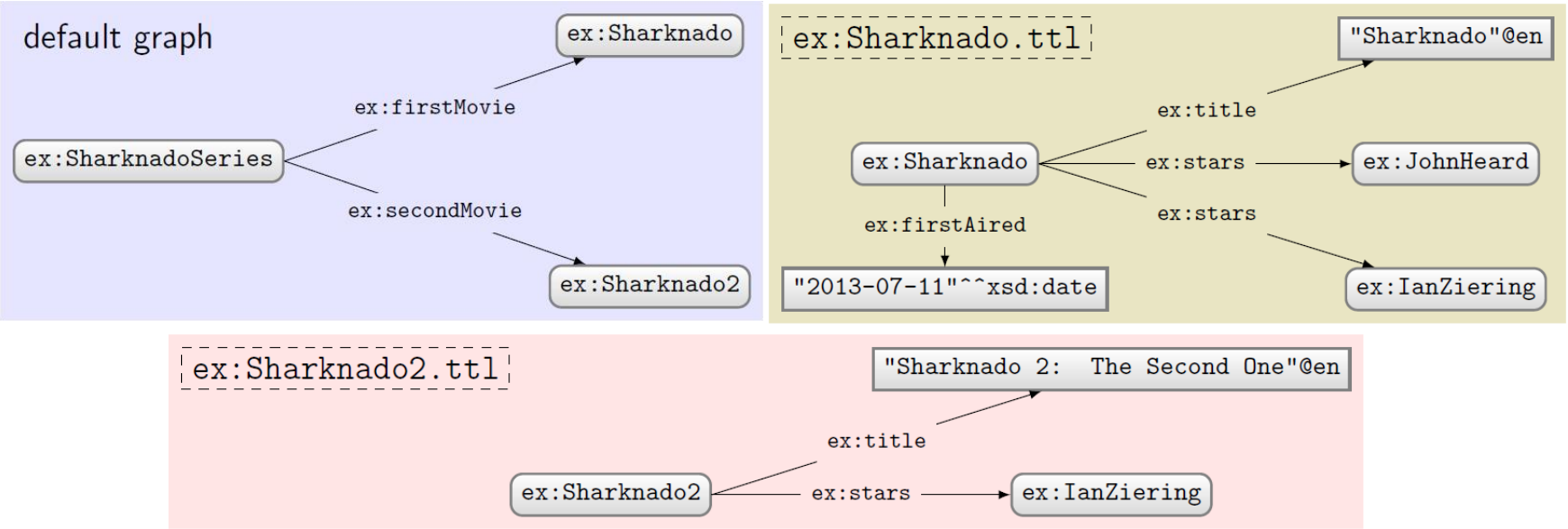
($\uplus$ indicates RDF merge)

Querying the named graphs in a dataset

- We can query parts of the dataset using …

- GRAPH: specifies the URI of a named graph from which results or a variable that ranges over all named graphs
  - Does not access the default graph!
  - If not specified, default graph is accessed

An example query



Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT DISTINCT ?s
WHERE { ?s ?p ?o }
```

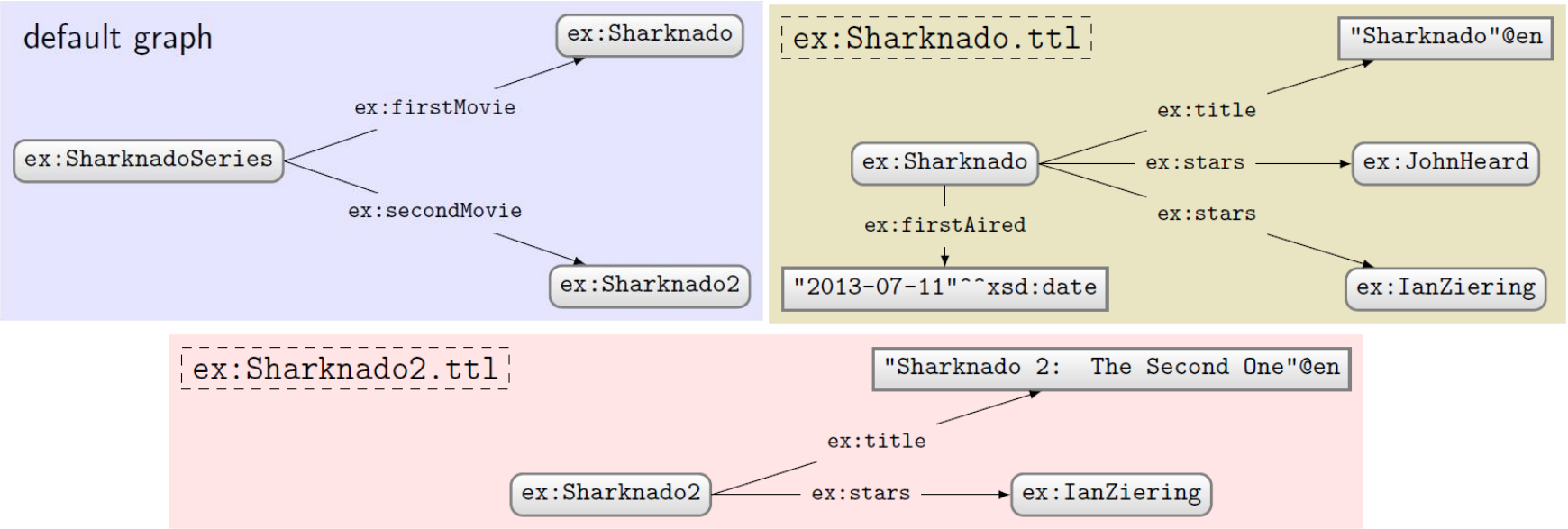
What solutions would this query return?

Solutions:

?s
ex:SharknadoSeries

No GRAPH clause so answers come from default graph only

Using FROM



Query:

```
PREFIX ex: <http://ex.org/voc#>
FROM ex:Sharknado.ttl
FROM ex:Sharknado2.ttl
SELECT DISTINCT ?s
WHERE { ?s ?p ?o }
```

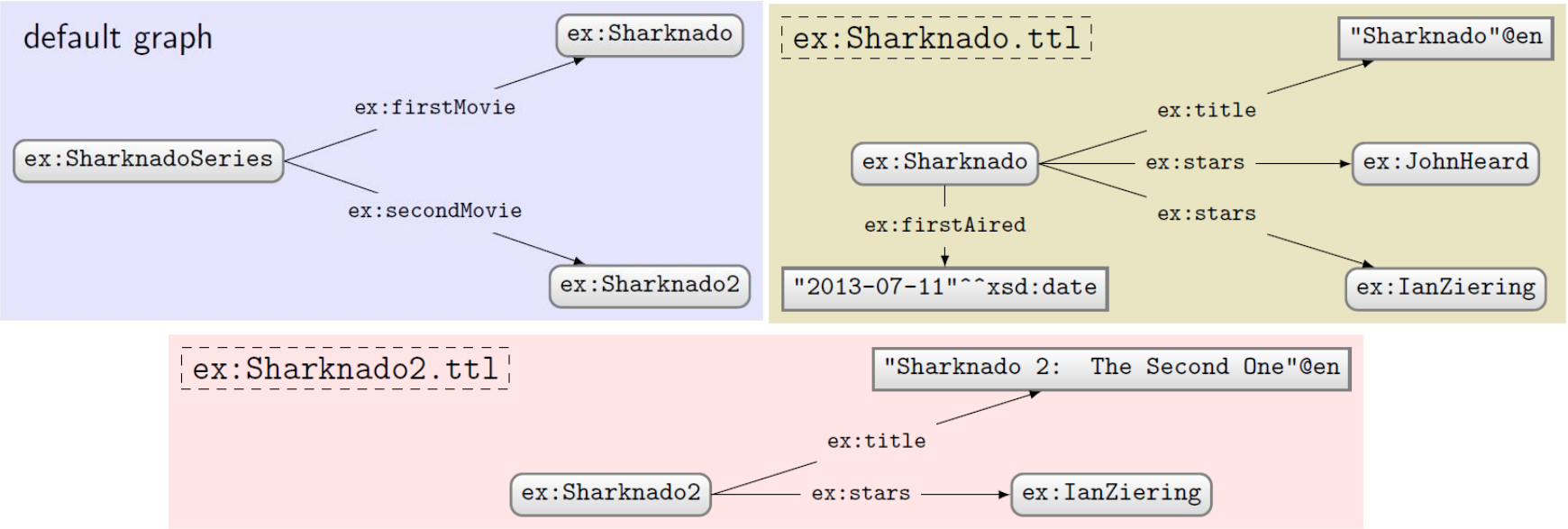
No GRAPH clause so answers come from default graph defined by FROM (old default graph cleared)

What solutions would this query return?

Solutions:

?s
ex:Sharknado
ex:Sharknado2

Using FROM NAMED



Query:

```
PREFIX ex: <http://ex.org/voc#>
FROM NAMED ex:Sharknado.ttl
SELECT DISTINCT ?s
WHERE { ?s ?p ?o }
```

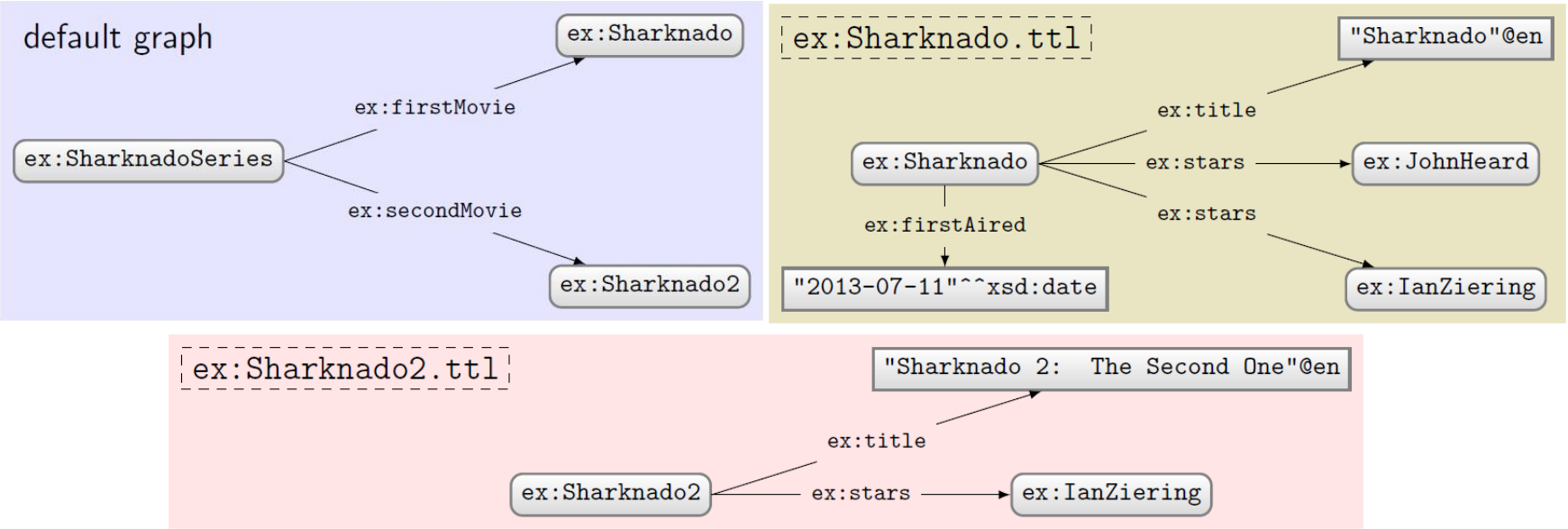
What solutions would this query return?

Solutions:

?s

No GRAPH clause so answers come from default graph, which is empty (since old default graph cleared)!

Using GRAPH with a variable



Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT DISTINCT ?s ?g
WHERE { GRAPH ?g { ?s ?p ?o } }
```

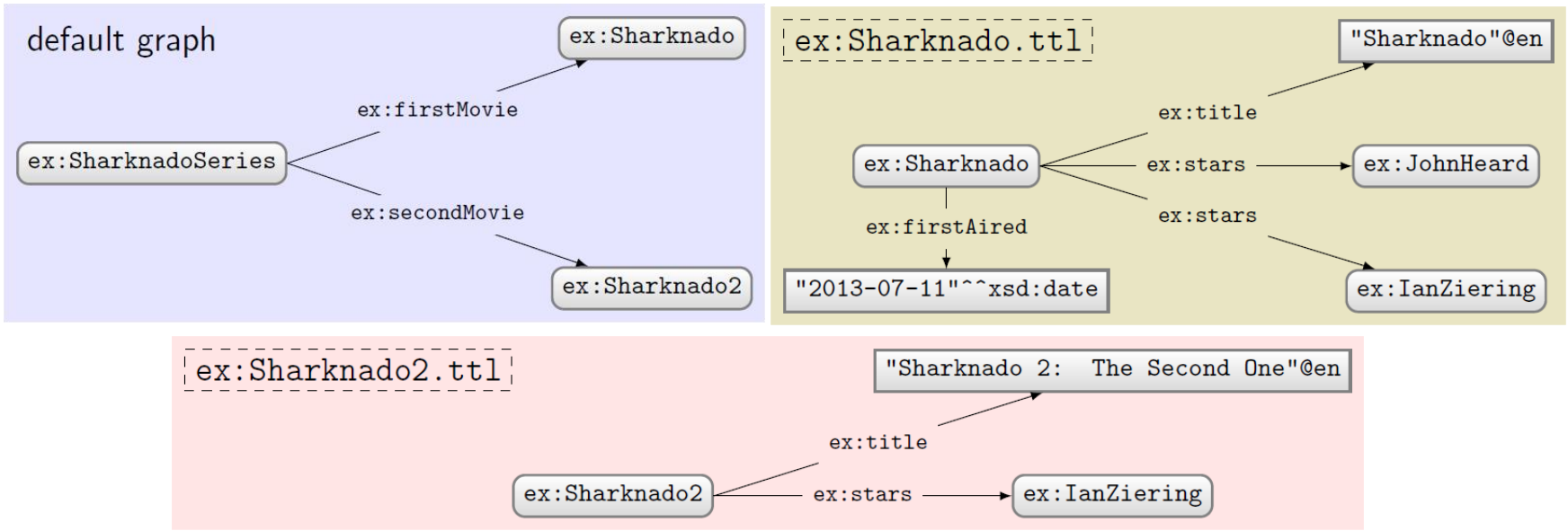
What solutions would this query return?

Solutions:

?s	?g
ex:Sharknado	ex:Sharnado.ttl
ex:Sharknado2	ex:Sharnado2.ttl

GRAPH clause only ranges over the named graphs.

Using GRAPH with a name



Query:

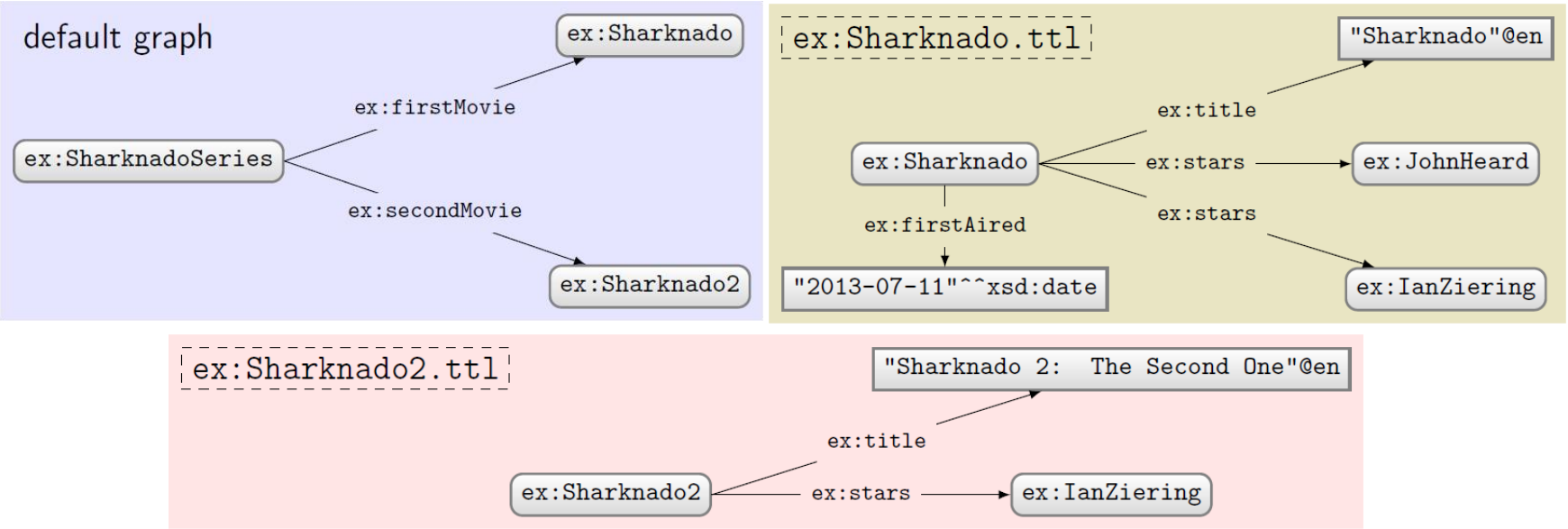
```
PREFIX ex: <http://ex.org/voc#>
SELECT DISTINCT ?s
WHERE {
  GRAPH ex:Sharknado.ttl { ?s ?p ?o }
}
```

What solutions would this query return?

Solutions:

?s
ex:Sharknado

Using GRAPH with FROM



Query:

```
PREFIX ex: <http://ex.org/voc#>
FROM ex:Sharknado.ttl
SELECT DISTINCT ?s ?g
WHERE {
  GRAPH ?g { ?s ?p ?o }
}
```

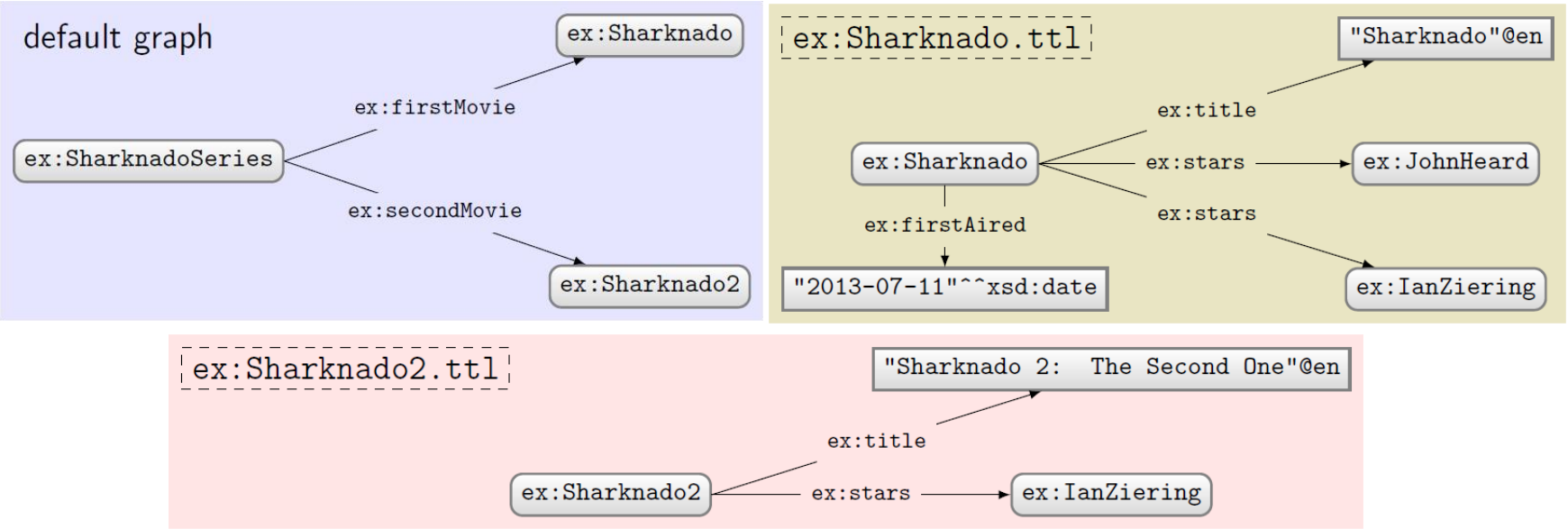
What solutions would this query return?

Solutions:

?s	?g

No named graphs specified!

Using GRAPH with FROM NAMED



Query:

```
PREFIX ex: <http://ex.org/voc#>
FROM NAMED ex:Sharknado.ttl
SELECT DISTINCT ?s ?g
WHERE {
  GRAPH ?g { ?s ?p ?o }
}
```

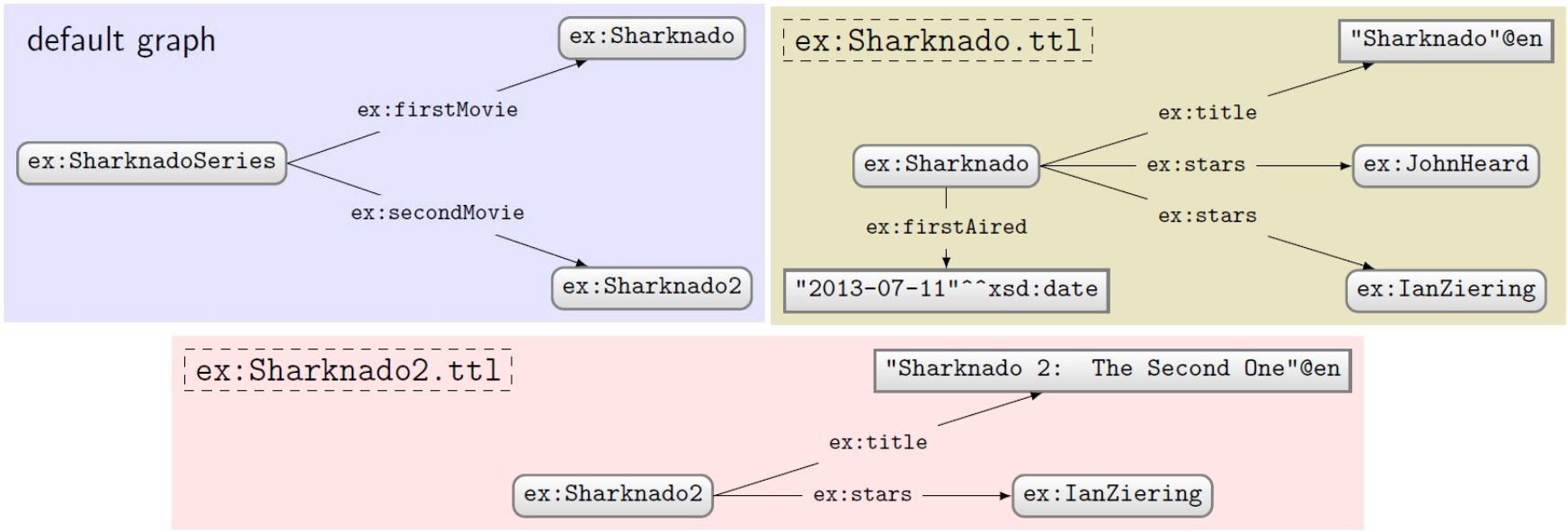
What solutions would this query return?

Solutions:

?s	?g
ex:Sharknado	ex:Sharnado.ttl

GRAPH accesses the one and only named graph

Using GRAPH with FROM and FROM NAMED



Query:

```
PREFIX ex: <http://ex.org/voc#>
FROM ex:Sharknado2.ttl
FROM NAMED ex:Sharknado.ttl
SELECT DISTINCT ?x ?q
WHERE {
  GRAPH ?g { ?s ?p ?o }
  ?x ?q ?o .
}
```

What solutions would this query return?

Solutions:

?x	?q
ex:Sharknado2	ex:stars

Questions?