SPARQL

IE650 Knowledge Graphs

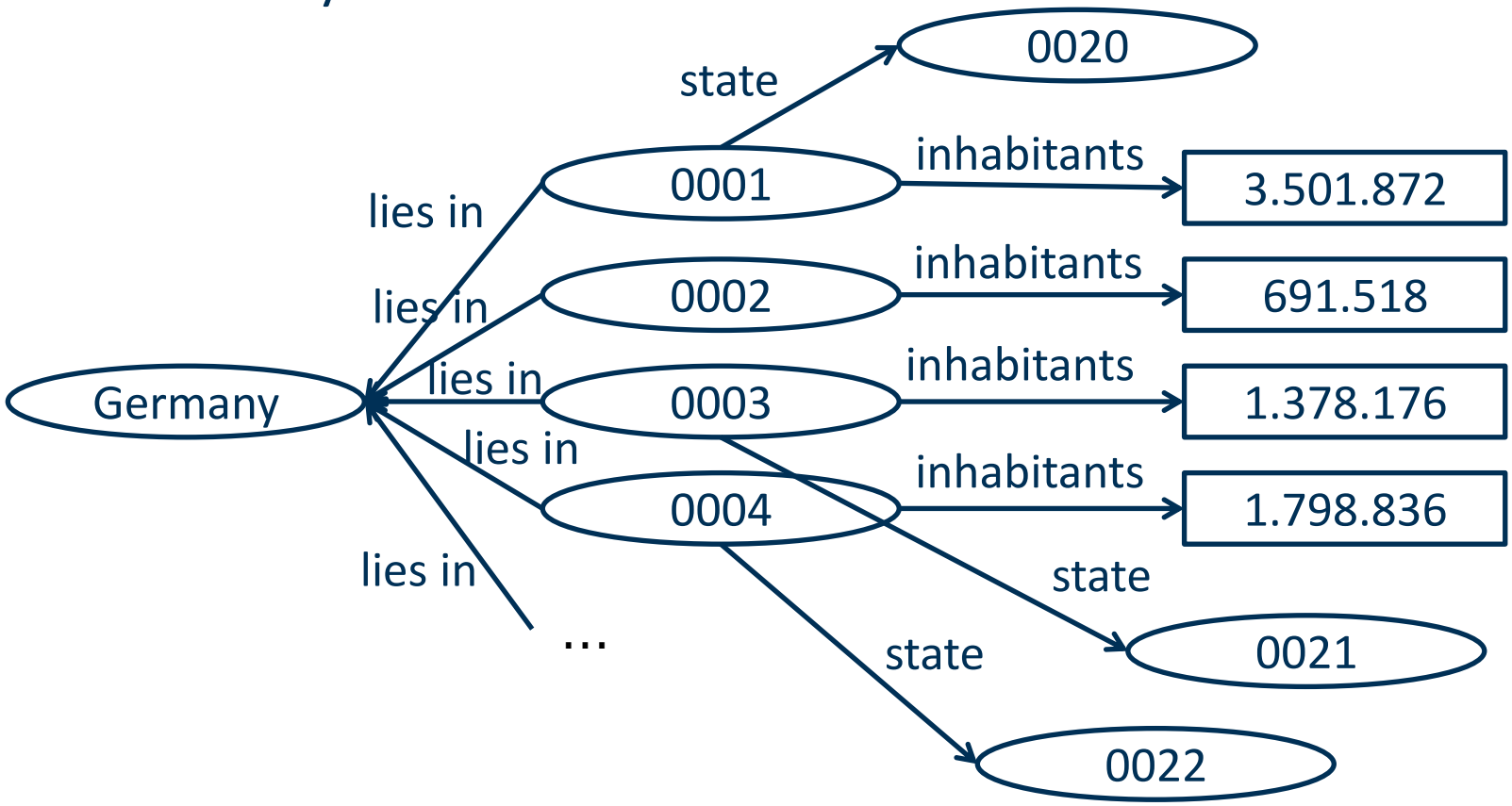
Previously on “Knowledge Graphs”

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- We have got to know
  - The RDF and RDFS languages
  - The Linked Open Data paradigm
- We have accessed KGs and Linked Open Data
  - With browsers and via programming frameworks
  - Jumping from node to node in the graph
- ...let us have a closer look!

An Example RDF Graph

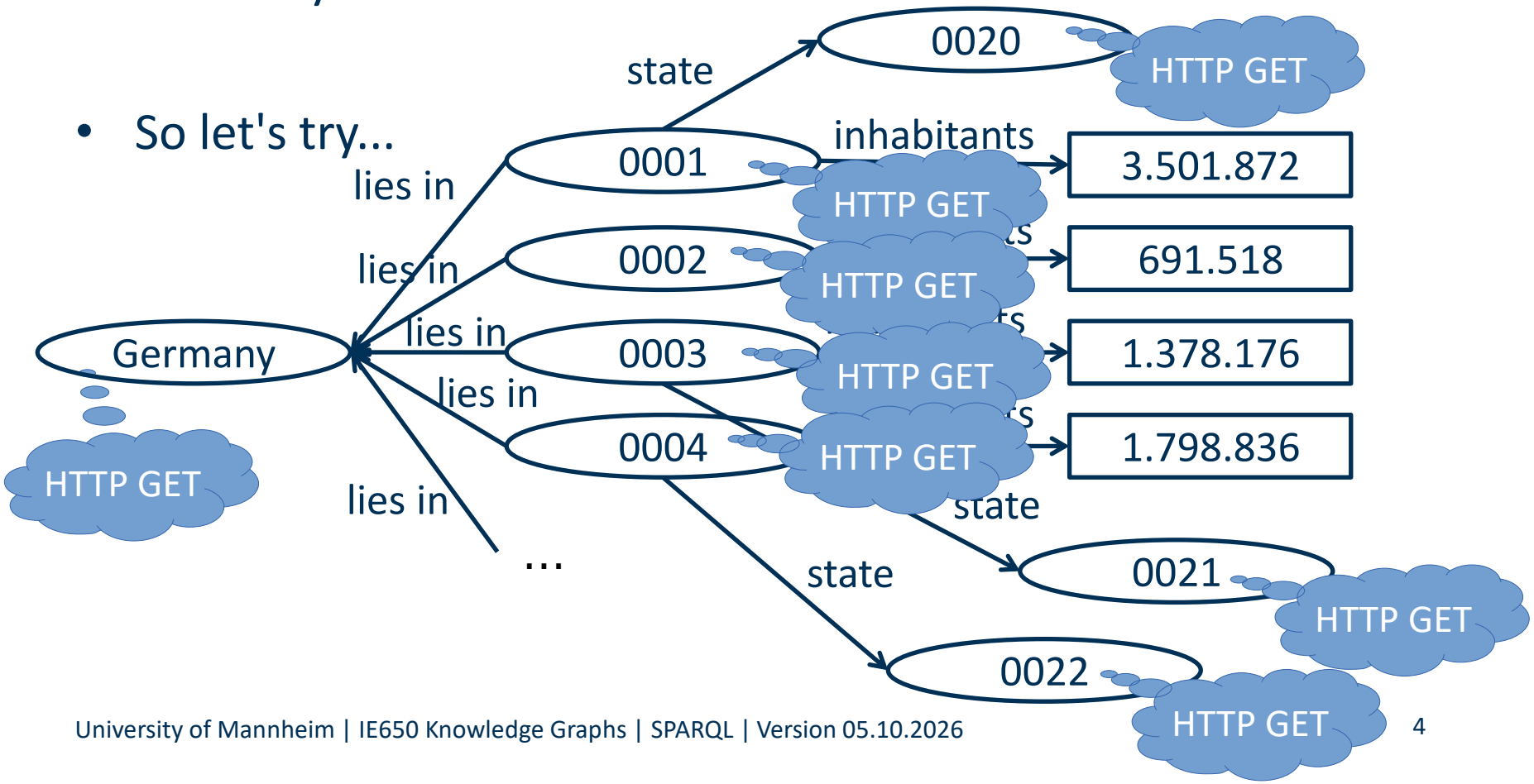
- Question: In which states are the five biggest cities of Germany located?



Information Retrieval on Linked Open Data

- Question: in which states are the five biggest cities of Germany located?

- So let's try...



Information Retrieval on Linked Open Data

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Observations
  - Navigation across derefencable URIs ultimately leads to a goal
  - But it is tedious
  - A lot of useless data is potentially retrieved

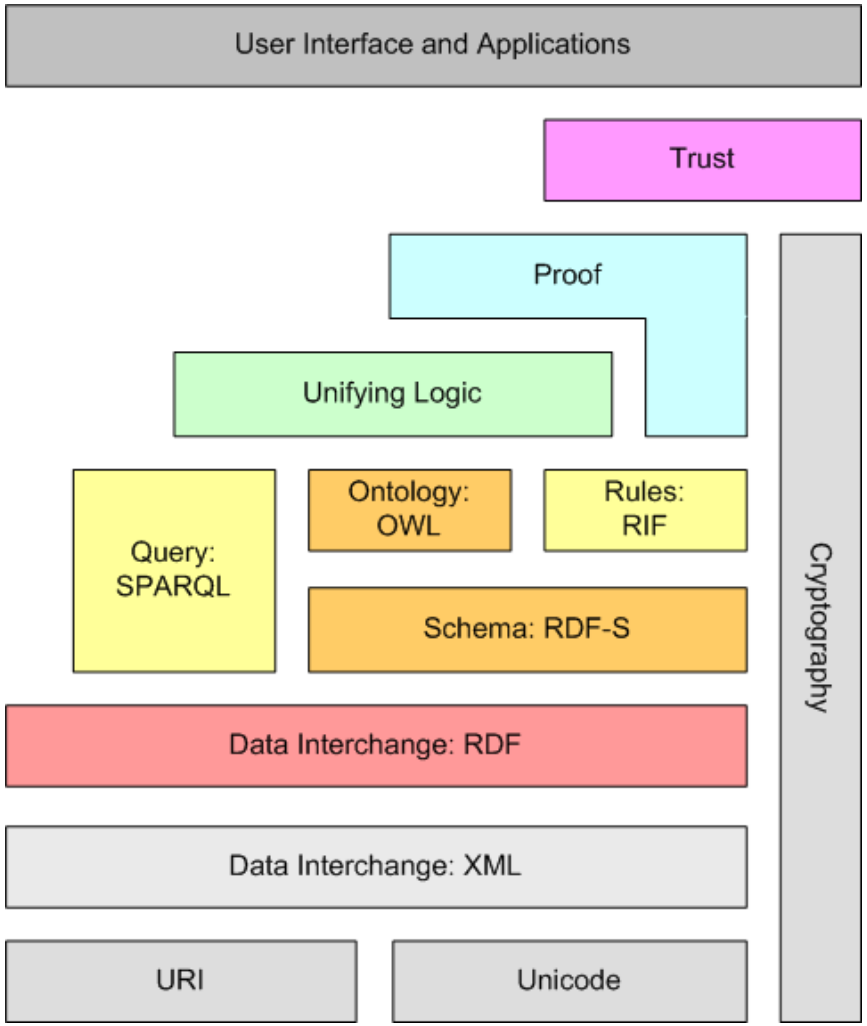
- Different information needs
  - Good for retrieving simple facts
  - Less efficient for more complex questions

Technology Stack

here be dragons...

Knowledge Graph Technologies
(This lecture)

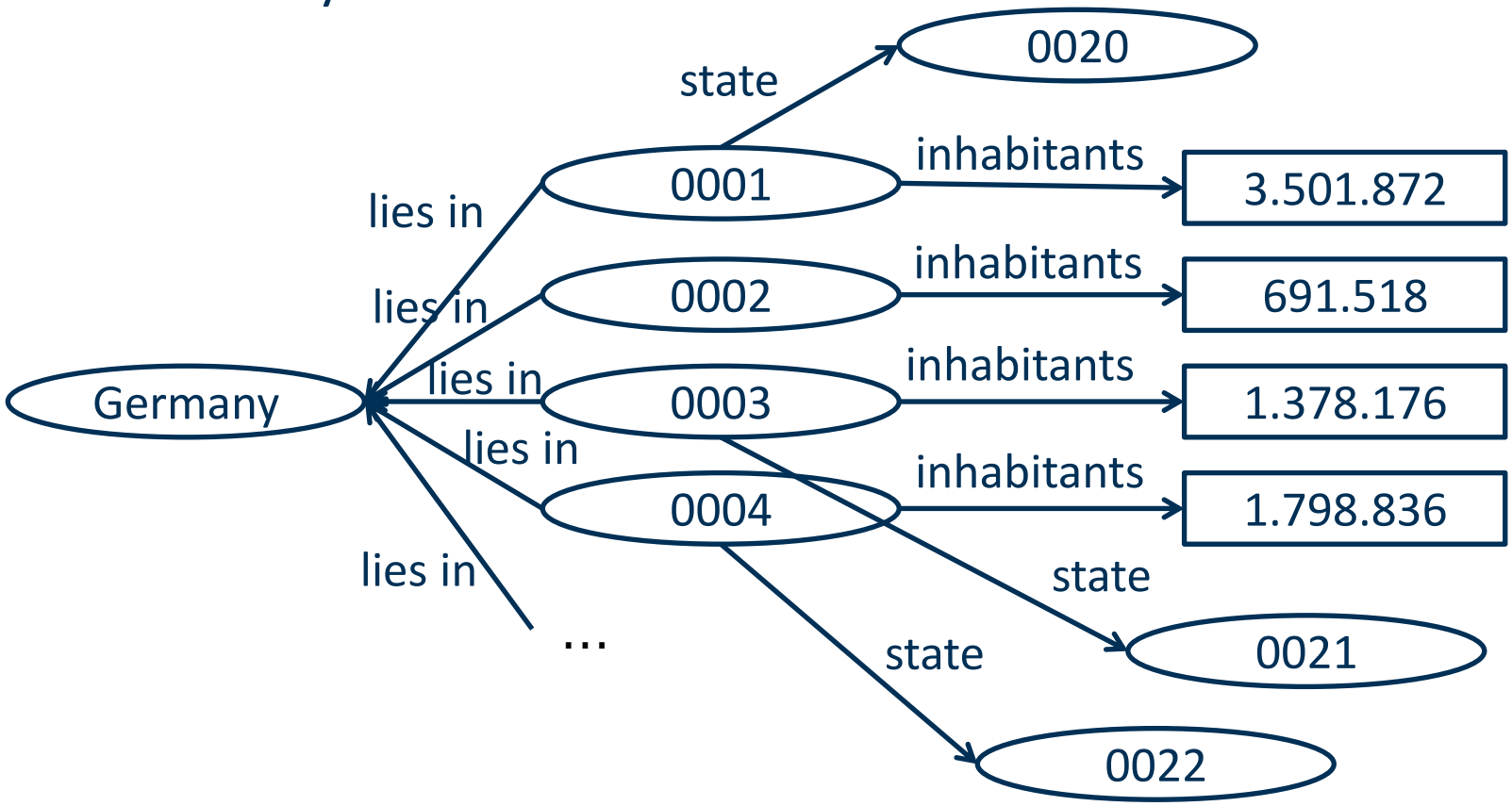
Technical
Foundations



Berners-Lee (2009): *Semantic Web and Linked Data* http://www.w3.org/2009/Talks/0120-campus-party-tbl/

What Would We Like to Have?

- Question: in which states are the five biggest cities of Germany located?



Wanted: A Query Language for the KGs

- ...just like SQL is for relational databases

```
SELECT   name, birthdate FROM customers
WHERE    id = '00423789'
```

id	name	birthdate
00183283	Stephen Smith	23.08.1975
00423782	Julia Meyer	05.09.1982
00789534	Sam Shepherd	31.03.1953
00423789	Herbert King	02.04.1960
...	...	...

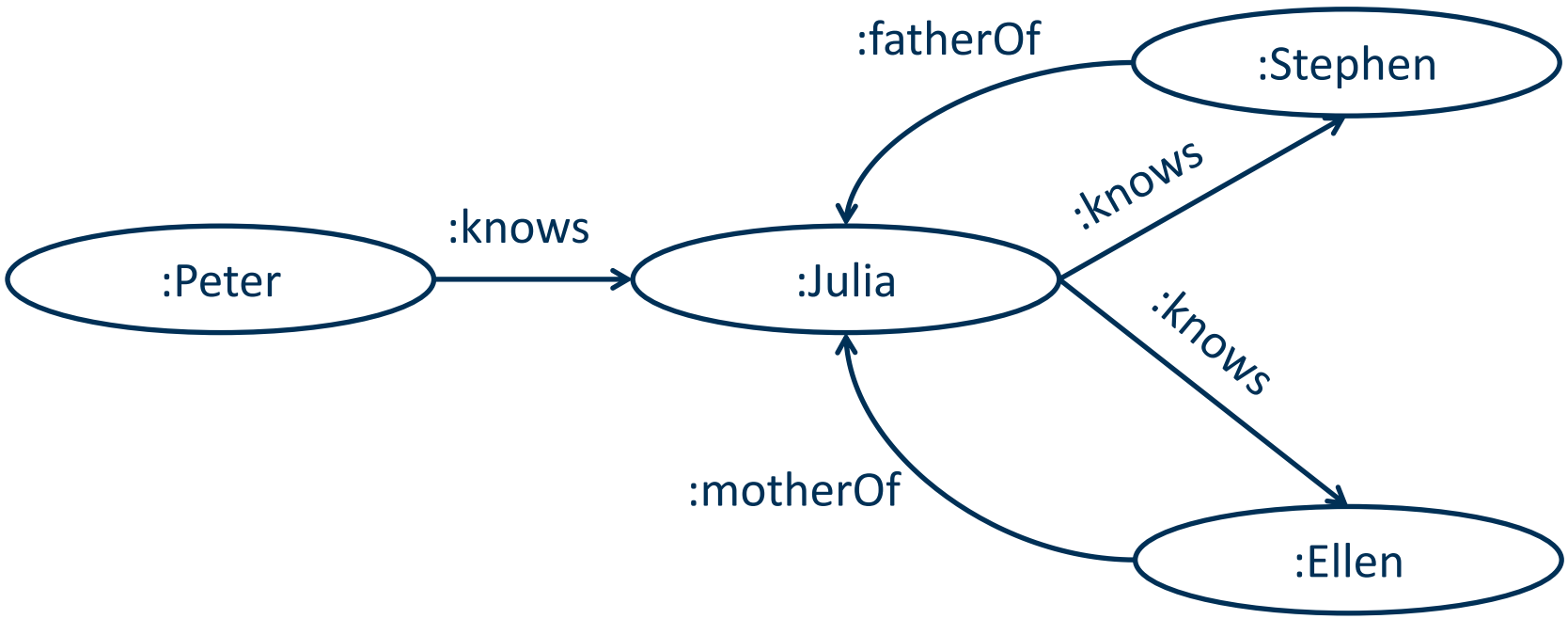
Wanted: A Query Language for the KGs

- SPARQL: "SPARQL Query Language for RDF"
  - a recursive acronym

- A W3C Standard since 2008
- Allows for querying RDF graphs

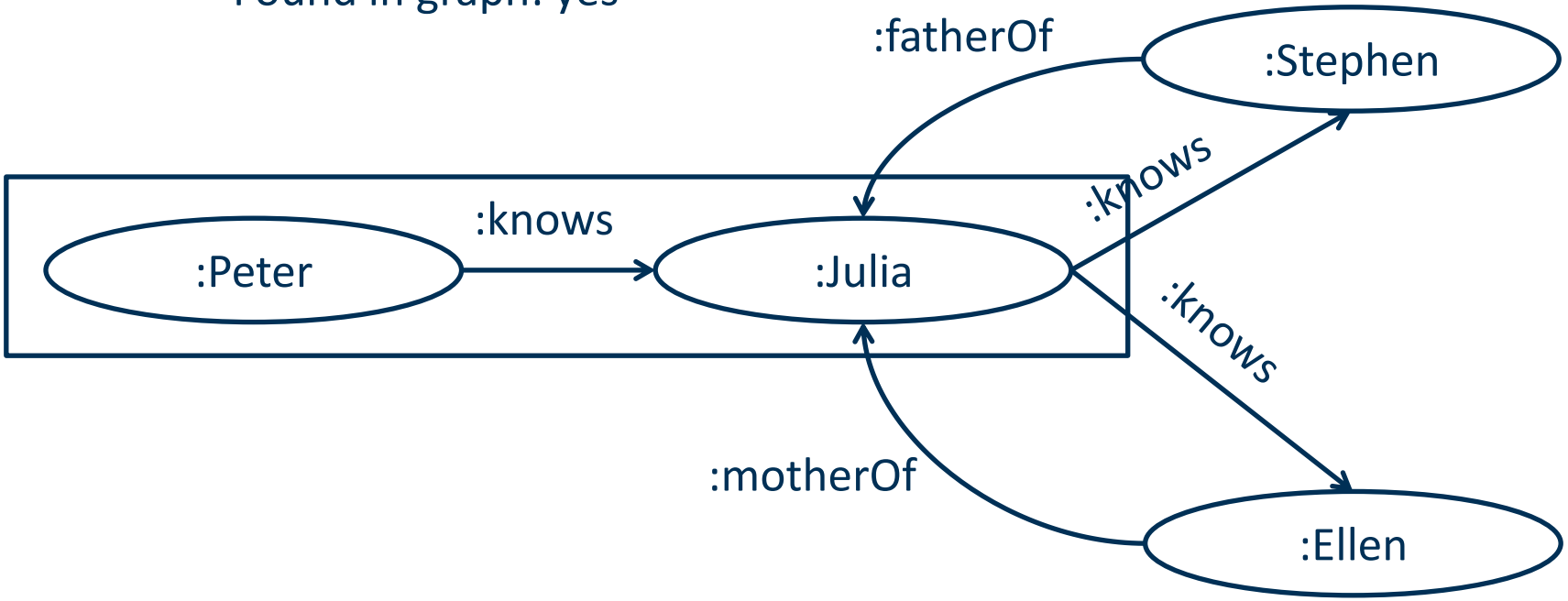
SPARQL: General Idea

- Look for specific subgraphs in a knowledge graph
  - Subgraphs may contain variables



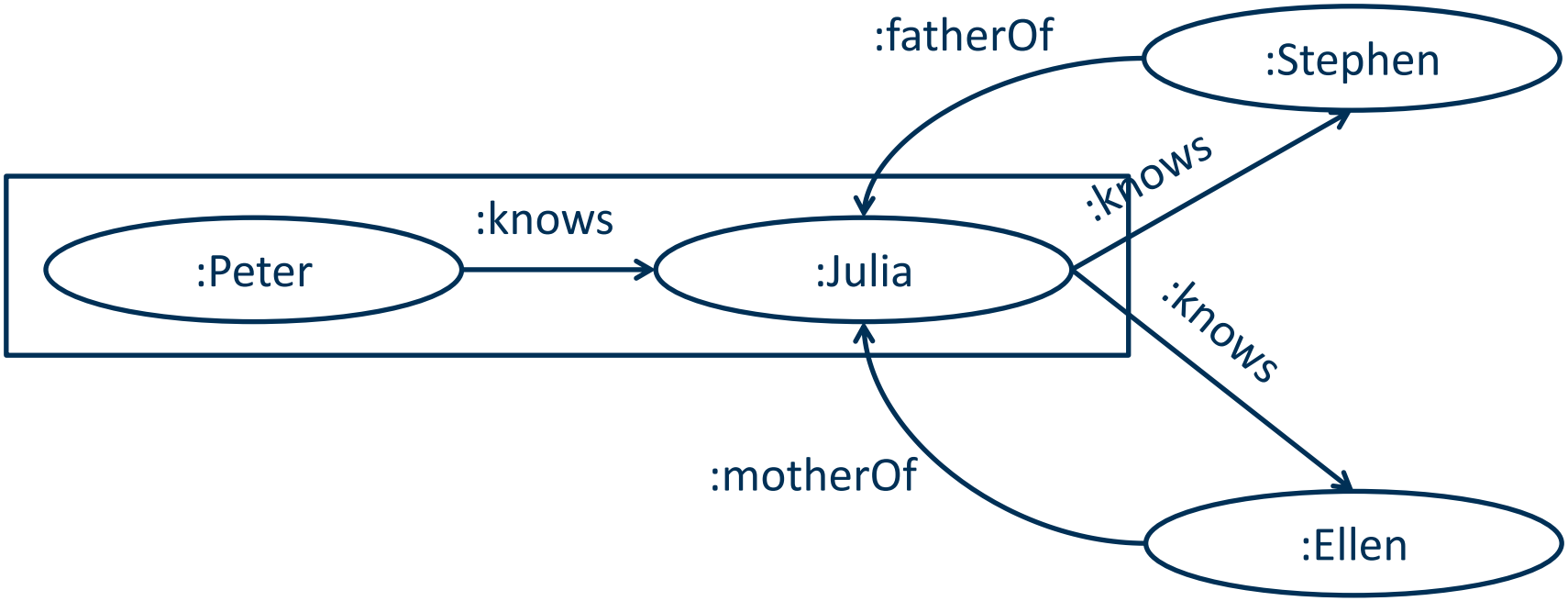
SPARQL Basics: Graph Pattern Matching

- Subgraph: :Peter :knows :Stephen
  - Found in graph: no
- Subgraph: :Peter :knows :Julia .
  - Found in graph: yes



SPARQL Basics: Graph Pattern Matching

- Subgraph with variable: ?x :knows :Julia .
  - Result: { ?x = :Peter }



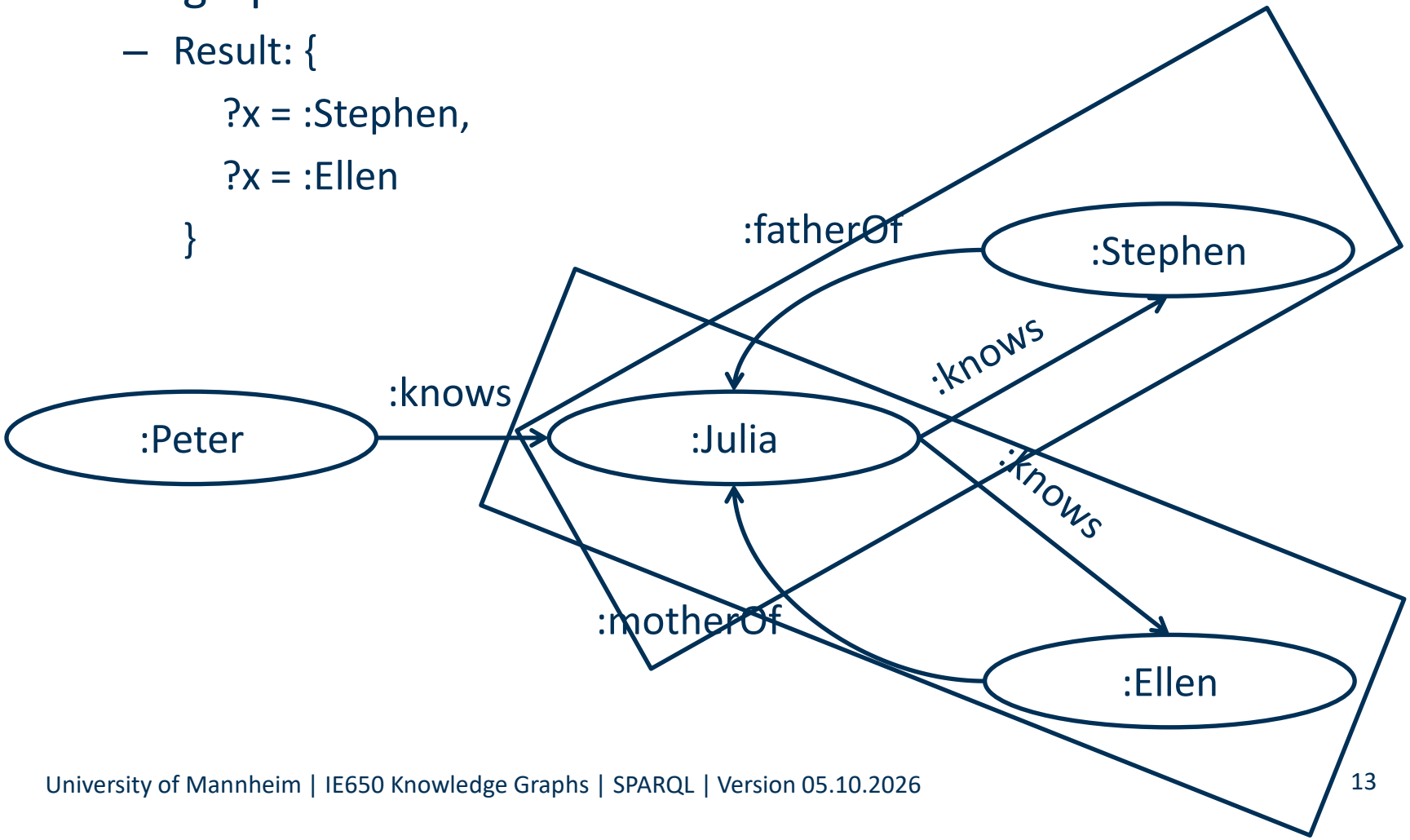
SPARQL Basics: Graph Pattern Matching

- Subgraph with variable: :Julia :knows ?x .
  - Result: {

    ?x = :Stephen,

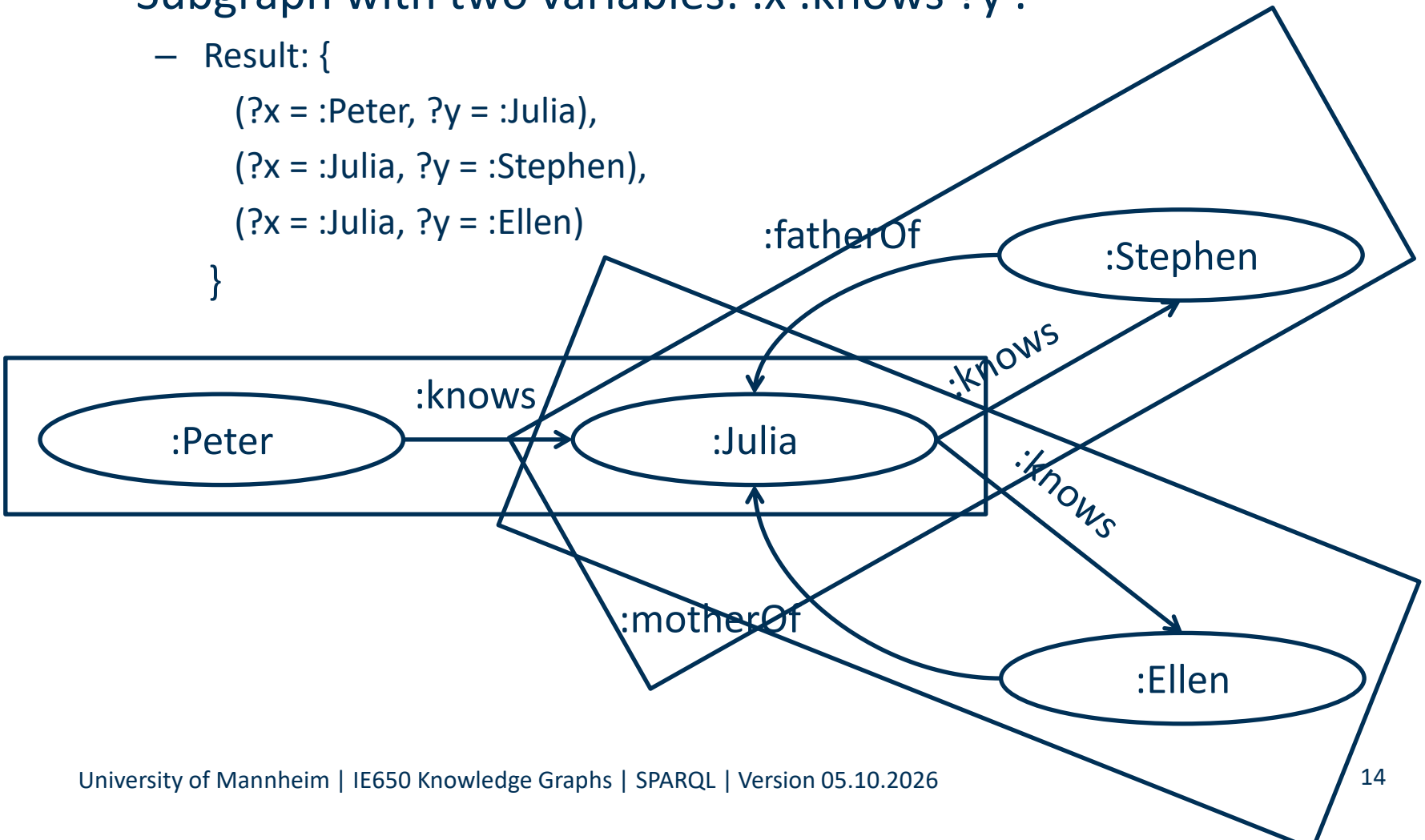
    ?x = :Ellen

    }



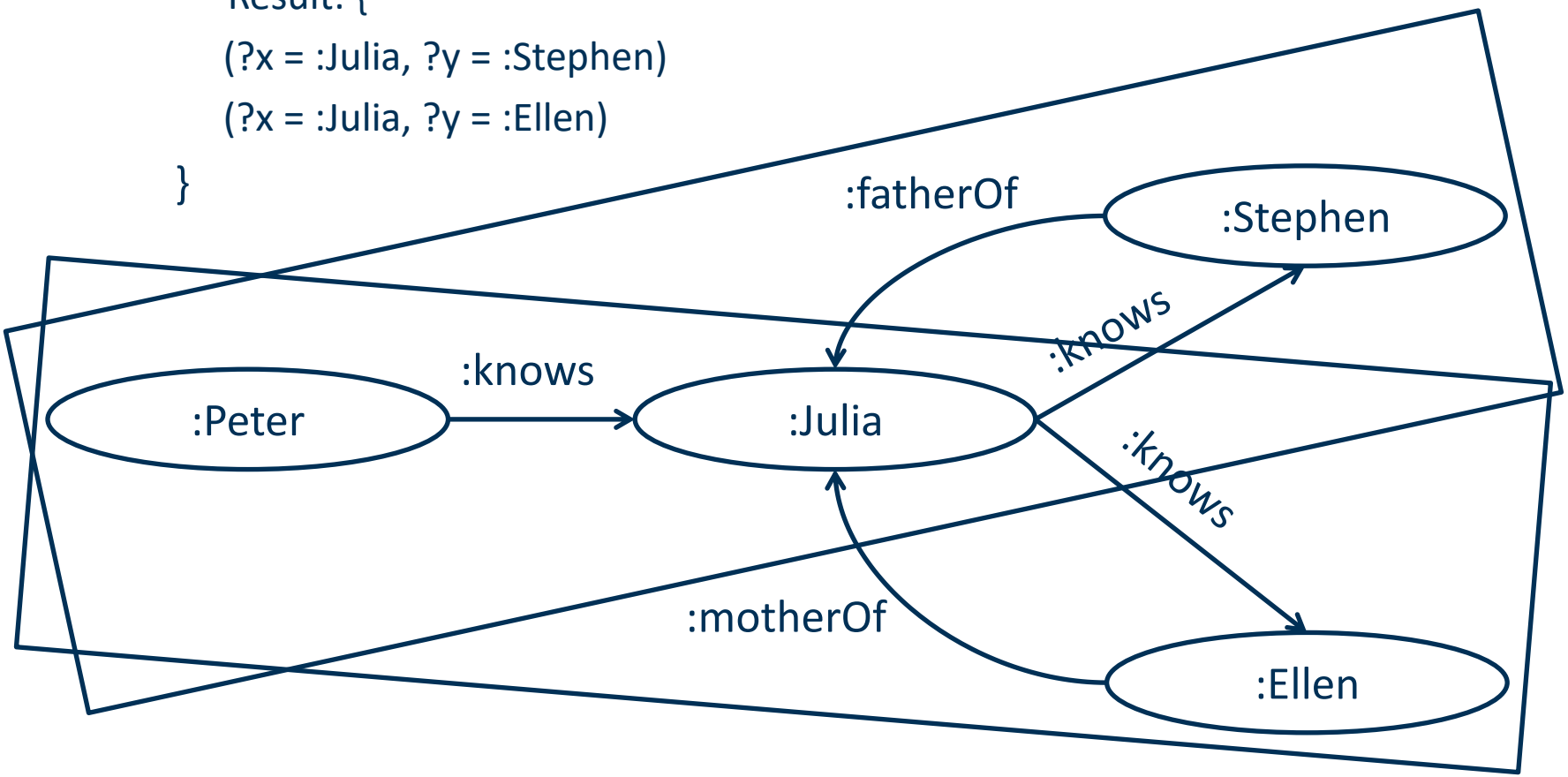
SPARQL Basics: Graph Pattern Matching

- Subgraph with two variables: :x :knows ?y .
  - Result: {
    (?x = :Peter, ?y = :Julia),
    (?x = :Julia, ?y = :Stephen),
    (?x = :Julia, ?y = :Ellen)
    }



SPARQL Basics: Graph Pattern Matching

- Complex subgraph: :Peter :knows ?x . ?x knows ?y .
  - Result: {
    (?x = :Julia, ?y = :Stephen)
    (?x = :Julia, ?y = :Ellen)
    }



Hello SPARQL!

- Example:

```
SELECT ?child
WHERE { :Stephen :fatherOf ?child }
```

Expressions with ? denote variables

SPARQL Syntax

- Basic structure:

```
SELECT <list of variables>
WHERE { <pattern> }
```

- Variables denoted with ?

- Prefixes as in RDF/N3:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?person ?name
WHERE { ?person foaf:name ?name }
```

SPARQL Syntax

- Basic structure:

```
SELECT <list of variables>
WHERE { <pattern> }
```

- The <pattern> in the WHERE clause is like N3
  - with variables

```
{?p foaf:name ?n }
{?p foaf:name ?n; foaf:homepage ?hp }
{?p foaf:knows ?p1, ?p2 }
```

SPARQL: Pattern Matching on RDF Graphs

- A person who has a daughter and a son:

```
{ ?p :hasDaughter ?d ; :hasSon ?s . }
```

- A person knowing two persons who know each other:

```
{ ?p :knows ?p1 , ?p2 . ?p1 :knows ?p2 . }
```

- A person who has two children:

```
{ ?p :hasChild ?c1, ?c2 . }
```

SPARQL: Pattern Matching on RDF Graphs

- ~~A person who has two children:~~

  ~~{ ?p :hasChild ?c1, ?c2 . }~~

Observation: different variables are not necessarily bound to different resources!

- ResultSet:
  - ?p=:Stephen, ?c1=:Julia, ?c2=:Julia

SPARQL: Blank Nodes

- WHERE clause: an RDF graph with variables

```
SELECT ?person1 ?person2 ?otherPerson
WHERE {  ?person1 :knows ?otherPerson .
         ?otherPerson :fatherOf ?person2 . }
```

- Result:
  - ?person1 = :Peter,
  - ?person2 = :Julia;
  - ?otherPerson = _:x1
- Note: Blank Node IDs are only unique within one result set!

SPARQL: Matching Literals

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Strings

```
{ ?country :name "Germany" . }
```

- Watch out for language tags!

```
{ ?country :name "Germany"@en . }
```

→ The Strings `"Germany"` and `"Germany"@en` are different!

- Numbers:

```
{ ?person :age "42"^^xsd:int .}
```

- Short hand notation:

```
{ ?person :age 42 . }
```

SPARQL: Filters

- Used for further restricting results

```
{?person :age ?age . FILTER(?age < 42) }
```

- Operators for comparisons:

```
=   !=   <   >   <=   >=
```

- Logical operations:

```
&&   ||   !
```

SPARQL: Filters

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Persons with younger siblings

```
{ ?p1 :siblingOf ?p2 .
  ?p1 :age ?a1 .
  ?p2 :age ?a2 .
  FILTER(?a2 < ?a1)}
```

- Persons that have both younger and older siblings

```
{ ?p1 :siblingOf ?p2, ?p3 .
  ?p1 :age ?a1 .
  ?p2 :age ?a2 .
  ?p3 :age ?a3 .
  FILTER(?a2 < ?a1 && ?a3 > ?a1)}
```

Question: why do we get different persons for p2 and p3?

SPARQL: Filters

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Second try: a person with two children

```
{ ?p :hasChild ?c1, ?c2 . FILTER( ?c1 != ?c2) }
```

- A slight improvement
  → Variables are now bound to different resources
- But: we still have the Non-Unique Naming Assumption
  → i.e., given that
  :Peter :hasChild :Julia .
  :Peter :hasChild :Stefan .
  we still cannot conclude that Peter has two children!
- Furthermore, there is still the Open World Assumption
  → i.e., Peter could also have more children

Filters for Strings

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Searching in Strings: using regular expressions

- People called “Ann”

```
{?person :name ?n . FILTER(regex(?n,"^Ann$")) }
{?person :name ?n . FILTER(regex(?n,"Ann")) }
```

→ the second variant would also find, e.g., “Mary-Ann”

- `str`: URIs and Literals as strings
- allows for, e.g., searching for literals across languages

```
{?country :name ?n . FILTER(str(?n) = "Tyskland") }
```

Further Built-In Features

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Querying the type of a resource
  - `isURI`
  - `isBLANK`
  - `isLITERAL`
- Querying for the data type and language tags of literals
  - `DATATYPE(?v)`
  - `LANG(?v)`
- Comparing the language of two literals
  - `langMATCHES(?v1,?v2)`
  - Caution: given `?v1 = "Januar"@DE`, `?v2 = "Jänner"@DE-at`

    `LANG(?v1) = LANG(?v2)` → false

    `langMATCHES(?v1,?v2)` → true

Combining Patterns

- Find the private and work phone number

```
        { ?p :privatePhone ?nr }
UNION   { ?p :workPhone ?nr }
```

- UNION creates a set union

  ?p = :peter, ?nr = 123;
  ?p = :john, ?nr = 234;
  ?p = :john, ?nr = 345;
  ...

That happens if John has both a private and a work phone

Interlude: A Real-World Example

Deutschland

Ab in den Warenkorb

Kriminalität Das Drogengeschäft verändert sich: Dealer verkaufen ihre Ware im frei zugänglichen Internet – abgerechnet wird in Bitcoins.

Chemical Love heißt die Website, und sie wirbt mit einem passenden Logo: eine Herzsilhouette, kombiniert mit dem Erlenmeyerkolben, Symbol der Chemiker. Was es dort zu kaufen gibt, hat wenig mit Liebe zu tun: „Downers", „Crystal Meth", „Kokain", „XTC", „Medikamente". Es ist ein Onlineshop mit Warenkorb, Nutzerrezensionen und dem Versprechen an Stammkunden: „Sie erfahren zuerst von den neuesten Produkten und kriegen Testpakete zugeschickt."

Derzeit allerdings ruht das Geschäft, die Betreiber des Webshops sitzen in Untersuchungshaft. Bei Polizeirazzien in Baden-Württemberg und Rheinland-Pfalz stellten die Fahnder kürzlich eine große Menge Drogen sicher: 54 Kilo Amphetamin, 4 Kilo Heroin, 1,3 Kilo Kokain und rund 25 000 Ecstasy-Tabletten.

Von „Deutschlands größtem illegalen Webshop für Betäubungsmittel" spricht die Staatsanwaltschaft Verden (Aller), deren Zentralstelle für Internetkriminalität die Ermittlungen angestoßen hatte. Bis zur Polizeiaktion brummte das Geschäft – bis zu 50 Päckchen mit Drogen verschickten die Dealer pro Tag per Post.

Bereits voriges Jahr war ein Leipziger Lehrling aufgeflogen, Spitzname „Shiny Flakes", der von seinem Kinderzimmer aus einen Drogenversand betrieb (SPIEGEL 34/2015). Aber anders als S., der im verschlüsselten Darknet agierte, verkauften die Betreiber von Chemical Love vorwiegend im offenen Internet. Die Domains waren auf Tonga und den australischen Kokosinseln registriert.

Fahnder sehen einen neuen Trend zum leicht auffindbaren Drogen-Webshop. Ein Dealer erklärt das auf einer einschlägigen Website so: „Ich denke, Clearnet wird DW (*Darkweb* –Red.) übertreffen, da viele User schlichtweg zu faul sind, sich einzuarbeiten, und es vielen auch zu kompliziert ist." Denn auf der dunklen Seite des Internets benötigt man zum Surfen eine spezielle Verschlüsselungssoftware, die den Eintritt in das anonyme Netzwerk („Tor-Netzwerk") ermöglicht.

Ganz ohne Verschleierung lief das Geschäft aber auch bei Chemical Love nicht. Die Betreiber ließen sich mit dem virtuellen Zahlungsmittel „Bitcoin" entlohnen, eine im Internet erzeugte und gehandelte Währung, die schnelle und anonyme Transaktionen in unbegrenzter Höhe zulässt.

Das verschlüsselte Zahlungssystem habe sich bei illegalen Geschäften im Netz durchgesetzt, bestätigt Staatsanwalt Benjamin Krause, Spezialist für die Bekämpfung der Internetkriminalität bei der Frankfurter Generalstaatsanwaltschaft.

Als die Frankfurter Zentralstelle 2010 ihre Arbeit aufnahm, wurden Rechnungen für krumme Deals im Netz noch häufig in bar beglichen: mit Geldscheinen etwa, die an eine Packstation des Händlers geschickt wurden. Bei Geschäften mit Portalen aus Übersee kamen auch Prepaid-Karten wie „Paysafe" zum Einsatz, die man an Tankstellen oder in Postfilialen kaufen kann.

Bitcoins machten die Sache für Kunden und Anbieter jedoch bequemer und unauffälliger, so Krause. Auch deshalb wandere ein zunehmender Teil des Drogenhandels mittlerweile ins Netz ab. „Wir haben es dort auch mit einer anderen Täterstruktur zu tun als im klassischen Drogenmilieu auf der Straße", sagt der Staatsanwalt. Am Computer müsse man sich nicht mit internationalen Banden oder Kiezgrößen um [illegible]

Auch hinter Chemical Love standen keine professionellen Drogendealer. Zu den Verdächtigen gehört Walter K., ein ehemaliger Fußballprofi aus Stuttgart. K. spielte zeitweise sogar in der deutschen Nationalmannschaft, nach seiner Karriere betrieb er für die Allianz ein Versicherungsbüro. Nachdem er ausgesagt hatte, wurde sein Haftbefehl außer Vollzug gesetzt.

Kopf der Bande ist nach den noch laufenden Ermittlungen Nikolas K., der Sohn des Fußballers. Der wegen Handels mit Crystal Meth vorbestrafte 29-Jährige sitzt in der rheinland-pfälzischen JVA Rohrbach ein. Davor bezog er mit Vorliebe Suiten von Luxushotels. Nikolas K. war es auch, der die Bitcoins umtauschte – gegen eine Überweisung vom Laptop aus gab es von Bitcoin-Händlern Cash in Plastiktüten.

K. verfüge über keine besonderen Computerkenntnisse, betont sein Anwalt Achim Bächle aus Stuttgart. Mithilfe eines einfachen Buchungssystems verwaltete K. die Bestellungen der Kunden und reichte sie an zwei Komplizen weiter.

Der eine, Denis T., 30, beschaffte Ware in den Niederlanden und gab die Pakete auf. Die Drogen soll sein Bruder Rene L., 31, ein Schichtarbeiter, im Keller seines Hauses in der Pfalz gelagert haben.

Dort griff die Polizei zu, als die Gruppe aus den Niederlanden zurückkam – Nikolas K. hatte für die Fahrt einen Jaguar XF gemietet. Bei K.s Papieren fand sich auch der Fahrzeugschein für einen Maserati.

Neben ihrem Gangster-Geprotze wurde der Gruppe ein Logistikproblem zum Verhängnis, das auch Shiny Flakes nicht gelöst hatte: Virtuell lassen sich die Drogen kaum an den Konsumenten bringen.

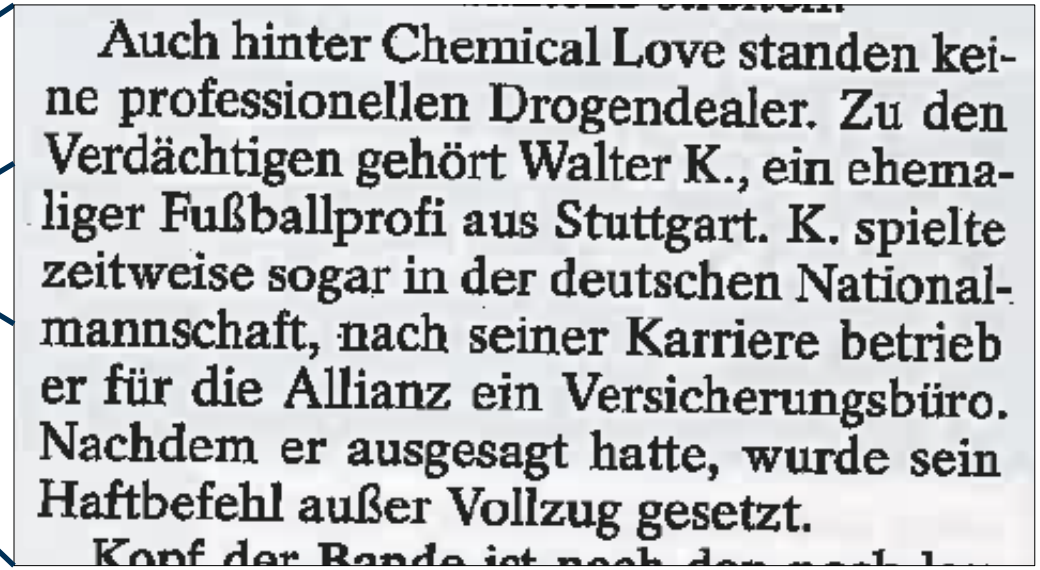
So konnten Ermittler die Pakete bis zu einer Pforzheimer Postfiliale zurückverfolgen. Das Bild einer Überwachungskamera zeigt T. in einer Situation, die auch Kunden mit weniger heikler Fracht gut kennen: in der Schlange vor dem Schalter.

Matthias Bartsch, Jan Friedmann

CHEMICAL LOVE

Angebot im Drogen-Webshop Chemical Love: Testpakete für Stammkunden

52 DER SPIEGEL 27/2016

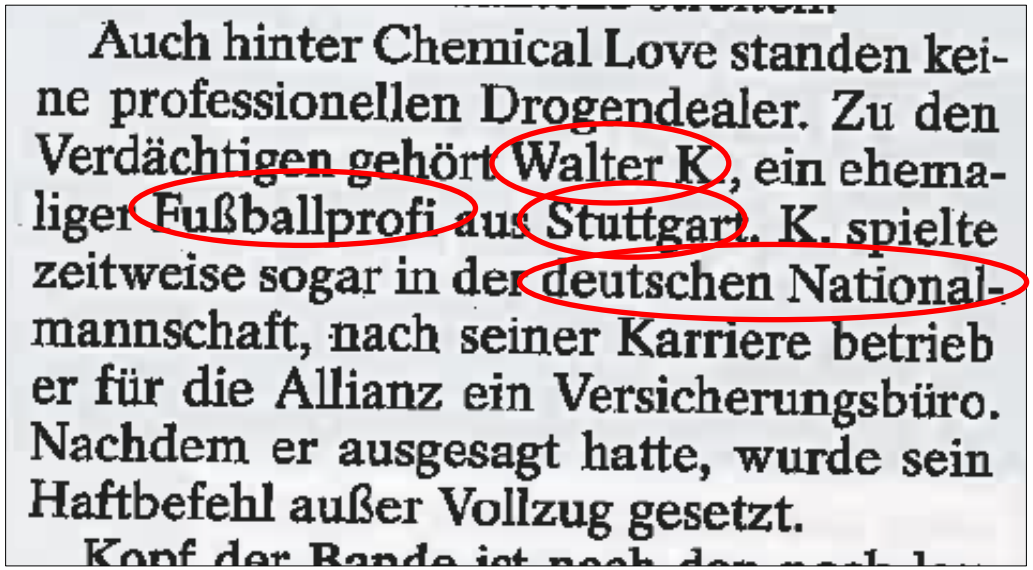


Auch hinter Chemical Love standen keine professionellen Drogendealer. Zu den Verdächtigen gehört Walter K., ein ehemaliger Fußballprofi aus Stuttgart. K. spielte zeitweise sogar in der deutschen Nationalmannschaft, nach seiner Karriere betrieb er für die Allianz ein Versicherungsbüro. Nachdem er ausgesagt hatte, wurde sein Haftbefehl außer Vollzug gesetzt.

Kopf der Bande ist nach den noch

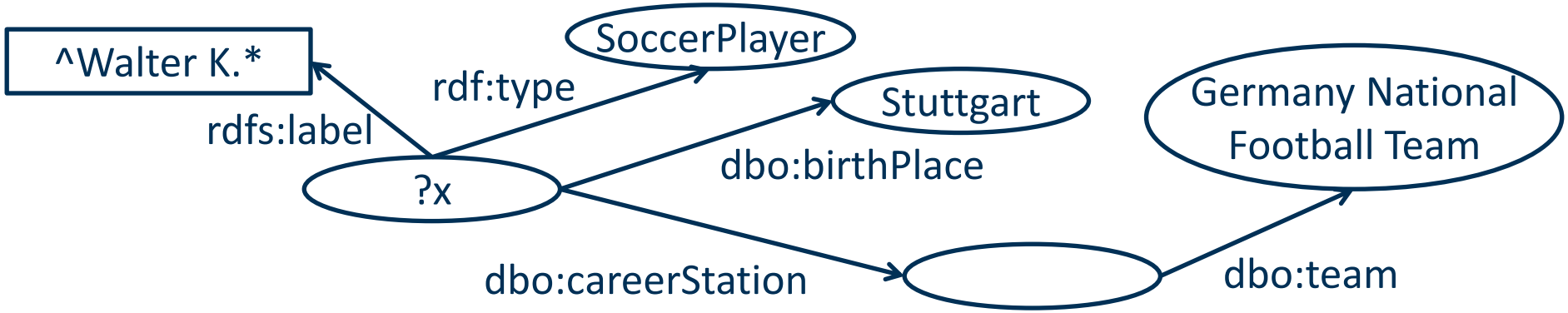
Der SPIEGEL, 27/2016, p. 52

Interlude: A Real-World Example

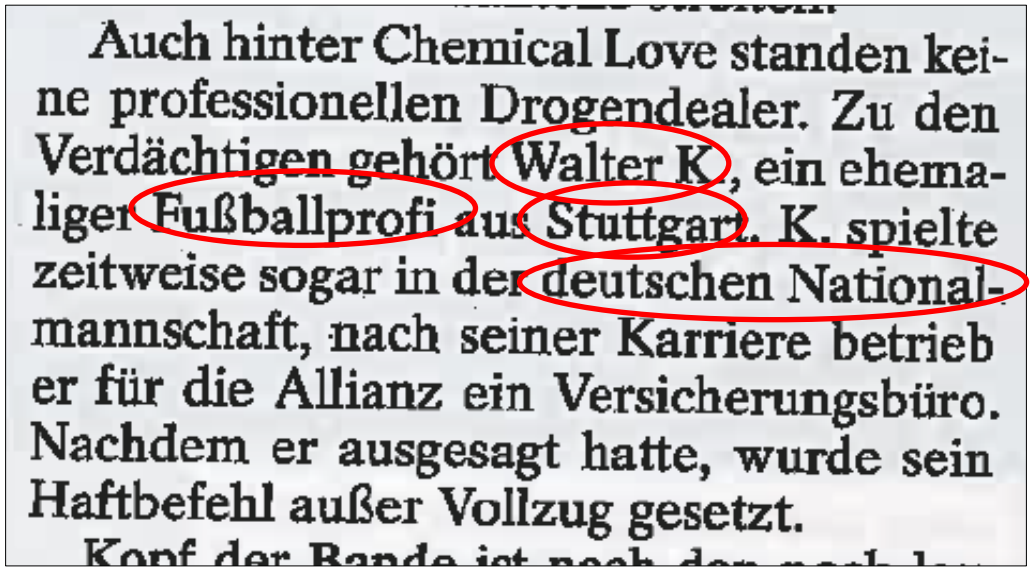


Auch hinter Chemical Love standen keine professionellen Drogendealer. Zu den Verdächtigen gehört Walter K., ein ehemaliger Fußballprofi aus Stuttgart. K. spielte zeitweise sogar in der deutschen Nationalmannschaft, nach seiner Karriere betrieb er für die Allianz ein Versicherungsbüro. Nachdem er ausgesagt hatte, wurde sein Haftbefehl außer Vollzug gesetzt.

Who is this Walter K.?



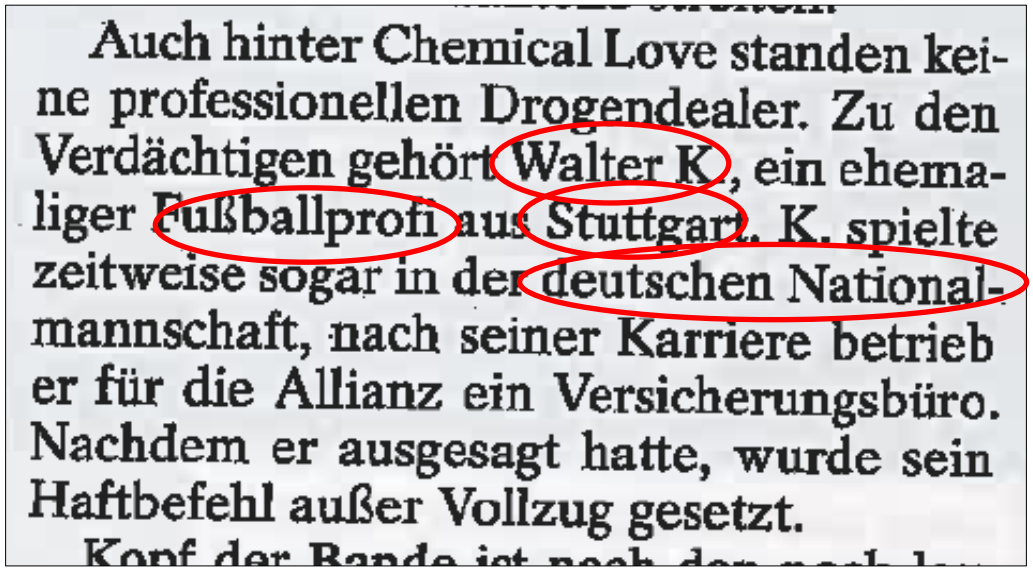
Interlude: A Real-World Example


Auch hinter Chemical Love standen keine professionellen Drogendealer. Zu den Verdächtigen gehört Walter K., ein ehemaliger Fußballprofi aus Stuttgart. K. spielte zeitweise sogar in der deutschen Nationalmannschaft, nach seiner Karriere betrieb er für die Allianz ein Versicherungsbüro. Nachdem er ausgesagt hatte, wurde sein Haftbefehl außer Vollzug gesetzt.

Who is this Walter K.?

```
SELECT DISTINCT(?x) WHERE {
        ?x dbo:birthPlace dbr:Stuttgart .
        ?x a dbo:SoccerPlayer .
        ?x dbo:careerStation ?s.
             ?s dbo:team dbr:Germany_national_football_team.
        ?x rdfs:label ?l . FILTER(REGEX(?l,"^Walter K.*"))
}
```

Interlude: A Real-World Example



Auch hinter Chemical Love standen keine professionellen Drogendealer. Zu den Verdächtigen gehört Walter K., ein ehemaliger Fußballprofi aus Stuttgart. K. spielte zeitweise sogar in der deutschen Nationalmannschaft, nach seiner Karriere betrieb er für die Allianz ein Versicherungsbüro. Nachdem er ausgesagt hatte, wurde sein Haftbefehl außer Vollzug gesetzt.

Who is this Walter K.?

We get one result:

<http://dbpedia.org/resource/Walter_Kelsch>

Interlude: A Real-World Example

UNIVERSITY OF MANNHEIM
Data and Web Science Group

Dienstag, 03. Mai 2016

Auf schiefe Bahn geraten

Ex-Nationalspieler Kelsch sitzt in U-Haft

Walter Kelsch war ein torgefährlicher Fußballer - sieben Jahre lang spielte er für den VfB Stuttgart, vier Mal trug er das DFB-Trikot. Nun ist der 60-jährige Schwabe mit dem Gesetz in Konflikt geraten.

Teilen Twittern Teilen

Walter Kelsch
(Foto: imago/Pressefoto Baumann)

Der ehemalige Fußball-Nationalspieler Walter Kelsch sitzt wegen Drogenhandels im Internet in Stuttgart-Stammheim in Untersuchungshaft. Dies bestätigte die Staatsanwaltschaft im niedersächsischen Verden.

Insgesamt waren fünf Personen bei einer Razzia am 14. April festgenommen worden. Dabei sei "Deutschlands größter illegaler Webshop für Betäubungsmittel" zerschlagen worden, teilte die Staatsanwaltschaft mit.

Man werfe den Tatverdächtigen vor, "als Gruppierung 'Chemical-Love' über ein vorwiegend deutschsprachiges Dark-Market Forum sowie über einen eigenen Webshop Kokain und diverse synthetische Drogen vertrieben zu haben", hieß es in einer Pressemitteilung. Insgesamt seien 54 kg Amphetamin, etwa 4 kg Heroin, rund 1,3 kg Kokain und etwa 25.000 Ecstasy-Tabletten sichergestellt worden.

Der heute 60-jährige Kelsch hatte von 1977 bis 1984 beim VfB Stuttgart gespielt und in 202 Bundesligaspielen 51 Tore erzielt. 1984 wurde er mit den Schwaben deutscher Meister. In vier Länderspielen erzielte der Offensivspieler drei Tore. Bis 2011 war Kelsch sogar Präsidiumsmitglied bei den Stuttgarter Kickers gewesen.

n-tv.de, wne/sid

Now you

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- What is the the federal state and founding year of Mannheim?
- Hints:
  - Sparql endpoint of DBpedia:
    - https://dbpedia.org/sparql
  - Page representing Mannheim in Dbpedia (for humans)
    - http://dbpedia.org/page/Mannheim
  - URI of Mannheim:
    - http://dbpedia.org/resource/Mannheim
    - dbr:Mannheim
  - Prefixes already defined:
    - dbr -> http://dbpedia.org/resource/
    - dbo -> http://dbpedia.org/ontology/ (including classes and properties)
    - dbp -> http://dbpedia.org/property/ (for properties that are not manually mapped)

Now you

- What is the the federal state and founding year of Mannheim?
- Hints:
  - Sparql endpoint of DBpedia:
    - https://dbpedia.org/sparql
  - URI of Mannheim: dbr:Mannheim

```
SELECT ?state ?year
WHERE {


}
```

Now you

- What is the the federal state and founding year of Mannheim?
- Hints:
  - Sparql endpoint of DBpedia:
    - https://dbpedia.org/sparql
  - URI of Mannheim: dbr:Mannheim

```
SELECT ?state ?year
WHERE {
      dbr:Mannheim dbo:federalState ?state;
                   dbo:foundingYear ?year.
}
```

Now you

- What happens if you do it for Schwetzingen?

```
SELECT ?state ?year
WHERE {
      dbr:Schwetzingen dbo:federalState ?state;
                       dbo:foundingYear ?year.
}
```

dbo:country	• dbr:http://dbpedia.org/resource/Germany
dbo:district	• dbr:Rhein-Neckar-Kreis
dbo:elevation	• 100.000000 (xsd:double)
dbo:federalState	• dbr:Baden-Württemberg
dbo:leaderTitle	• Lord Mayor (en)
dbo:licenceNumber	• HD
dbo:postalCode	• 68701–68723

foundingYear

Optional Patterns

- Find a person's phone number and fax number, **if existing**

```
          { ?p :phone ?tel }
OPTIONAL { ?p :fax ?fax }
```

- OPTIONAL also creates unbound variables

  ?p = :peter, ?tel = 123, ?fax = 456;

  ?p = :john, ?tel = 234, ?fax = ;

  ?p = :julia, ?nr = 978; ?fax = 349;

  ...

Unbound variable:
John does not have a fax number (as far as we know)

Unbound Variables

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Variables can remain unbound
- We can test this with BOUND

- Everybody who has a phone or a fax (or both):

```
OPTIONAL {?p :phone ?tel . }
OPTIONAL {?p :fax ?fax . }
FILTER ( BOUND(?tel) || BOUND(?fax) )
```

Negation

- This is a common question w.r.t. SPARQL
- How do I do this:
  - "Find all persons who do not have siblings."

- This is left out of SPARQL intentionally!
- Why?

- Open World Assumption
  - We cannot know!
- For the same reason, there is no `COUNT`
  - At least not in standard SPARQL

Negation – Hacking SPARQL

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- However, there is a possibility
  - try with caution!

- Using `OPTIONAL` and `BOUND`
- Find all persons without siblings

```
OPTIONAL {?p :hasSibling ?s . }
FILTER ( !BOUND(?s) )
```

- This works
- However, you should know what you are doing
  - ...and how to interpret the results!

Negation – Hacking SPARQL

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- How does that work?
- Results before FILTER:

```
OPTIONAL {?p :hasSibling ?s . }

  ?p = :peter, ?s = :julia
  ?p = :julia, ?s = :peter
  ?p = :mary, ?s =
  ?p = :paul, ?s =
```

Unbound variables

- Applying the FILTER
  - `FILTER(!BOUND(?s))`

```
  ?p = :mary, ?s =
  ?p = :paul, ?s =
```

Sorting and Paging Results

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Sorting: `ORDER BY ?name`
- Limitations: `LIMIT 100`
- Lower Bounds: `OFFSET 200`

- Example: persons 101-200, ordered by name
  - `ORDER BY ?name LIMIT 100 OFFSET 100`

- `LIMIT/OFFSET` without `ORDER BY`:
  - Result orderings are not deterministic
  - There is no default ordering

Sorting and Paging Results

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Application scenarios:
  - Some SPARQL services limit their result set sizes
  - Pre-loading in applications
- Application example:
  - let the user browse cities
  - it is more likely that users want to see the big cities
  - display 100 biggest cities on one page, show more on demand
-
```
SELECT ?city ?population
WHERE {?city hasPopulation ?population}
ORDER BY DESC(?population)
LIMIT 100
```

Filtering Duplicates

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- ```
  SELECT DISTINCT ?person
  WHERE { ?person :privatePhone ?nr }
  UNION { ?person :workPhone ?nr }
  ```
- This means: all results with identical variable bindings are filtered
- This does not mean: persons identified by *?person* are actually different
- Why?
  - Non-unique naming assumption
  ```

Custom Built-Ins

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Some SPARQL engines allow special constructs
- also known as *Custom Built-Ins*
- Example: geographic processing
  - Knowledge Graph: DBpedia
  - Runs on Virtuoso

Custom Built-Ins

- Querying for coordinates

```
SELECT ?x
WHERE {
    ?x geo:long ?long;
       geo:lat ?lat .
    FILTER (?long>8.46 &&
            ?long<8.47 &&
            ?lat>49.48 &&
            ?lat<49.49)
}
```

SPARQL | HTML5 table

x
http://dbpedia.org/resource/Action_at_Mannheim_(1795)
http://dbpedia.org/resource/GESIS_–_Leibniz_Institute_for_the_Social_Sciences
http://dbpedia.org/resource/Reiss_Engelhorn_Museum
http://dbpedia.org/resource/University_of_Mannheim__University_of_Mannheim__1
http://dbpedia.org/resource/University_of_Mannheim
http://dbpedia.org/resource/University_of_Mannheim_Business_School
http://dbpedia.org/resource/Jesuit_Church,_Mannheim
http://dbpedia.org/resource/Battle_of_Mannheim_(1799)
http://dbpedia.org/resource/Battle_of_Seckenheim
http://dbpedia.org/resource/Klapsmühl'_am_Rathaus
http://dbpedia.org/resource/Mannheim
http://dbpedia.org/resource/Mannheim_Business_School
http://dbpedia.org/resource/Mannheim_University_Library
http://dbpedia.org/resource/Zentrum_für_Europäische_Wirtschaftsforschung
http://dbpedia.org/resource/Mannheim_Observatory
http://dbpedia.org/resource/Palais_Bretzenheim
http://dbpedia.org/resource/Angel_of_Peace_(Mannheim)
http://dbpedia.org/resource/Graduate_School_of_Economic_and_Social_Sciences
http://dbpedia.org/resource/Mannheim_Palace
http://dbpedia.org/resource/Mannheim_Palace_Church

Custom Built-Ins

- More complex queries
  - All museums within a 10km radius of a given point

```
SELECT ?x
WHERE { ?x rdf:type dbo:Museum;
           geo:long ?long;
           geo:lat ?lat
        FILTER (bif:st_intersects(
            bif:st_point(?long, ?lat),
            bif:st_point(8.466, 49.488), 10))
}
```

x
http://dbpedia.org/resource/Automuseum_Dr._Carl_Benz
http://dbpedia.org/resource/Reiss_Engelhorn_Museum
http://dbpedia.org/resource/Kunsthalle_Mannheim
http://dbpedia.org/resource/Technoseum

Further Query Types: ASK

- So far, we have only looked at SELECT queries
- ASK allows for yes/no queries:
  - e.g., are there persons with siblings?

```
ASK {?p :hasSibling ?s . }
```

- Often faster than SELECT queries

- The answer is true or false
  - *false* means: there are no matching sub graphs
  - do not misinterpret (Open World Assumption!)

Further Query Types: DESCRIBE

- All properties of a resource

```
DESCRIBE <http://dbpedia.org/resource/Berlin>
```

- Can be combined with a WHERE clause

```
DESCRIBE ?city WHERE { :Peter :livesIn ?city . }
```

- Allows for exploration of a dataset with unknown structure
- Caution: types of results are not standardized, results vary from implementation to implementation!

Further Query Types: CONSTRUCT

- Creates a new RDF graph

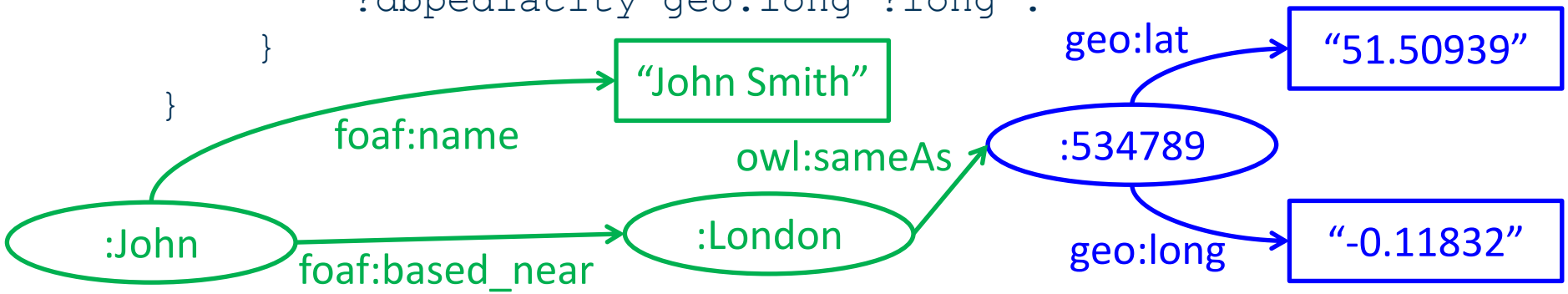
```
CONSTRUCT {
    ?x rdfs:seeAlso <http://dbpedia.org/resource/Berlin> .
}
WHERE {
    <http://dbpedia.org/resource/Berlin> ?y ?x .
    FILTER (isURI(?x))
}
```

- CONSTRUCT returns complete RDF graphs
  - e.g., for further processing

Query Federation

- Queries can be answered over *multiple* SPARQL endpoints
- Example

```
SELECT ?name ?lat ?long WHERE {
    ?x a foaf:Person .
    ?x foaf:name ?name .
    ?x foaf:based_near ?city .
    ?city owl:sameAs ?dbpediacity .
    SERVICE <http://dbpedia.org/sparql> {
        ?dbpediacity geo:lat ?lat .
        ?dbpediacity geo:long ?long .
    }
}
```



SPARQL: Wrap-Up

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- SPARQL is a query language for the semantic web
- Basic principle: pattern matching on graphs
- SPARQL allows for directed search for information instead of navigating the graph from node to node

- Results follow the semantic principles of RDF!
  - Open World Assumption
  - Non-unique naming assumption

Example: Jena + SPARQL

- Querying models with SPARQL

```
String queryString = "SELECT ?x ...";
Query query = QueryFactory.create(queryString);
QueryExecution qe =
  QueryExecutionFactory.create(query, model);
ResultSet results = qe.execSelect();
while(results.hasNext()) {
   QuerySolution sol = results.next();
   String s = sol.get("x").toString();
   ...
}
```

Example: RDFlib + SPARQL

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Querying models with SPARQL

```python
from rdflib import Graph, Namespace

# Create a graph
g = Graph()

# Load data about Mannheim directly from DBpedia (RDF/XML format)
g.parse("http://dbpedia.org/data/Mannheim.rdf")

# Define prefixes
dbr = Namespace("http://dbpedia.org/resource/")
dbo = Namespace("http://dbpedia.org/ontology/")

# Example query: get federal state and founding year
q = """
SELECT ?state ?year
WHERE {
    dbr:Mannheim dbo:federalState ?state ;
                 dbo:foundingYear ?year .
}
"""

# Run query
for row in g.query(q, initNs={"dbr": dbr, "dbo": dbo}):
    print(f"State: {row.state}, Year: {row.year}")
```

SPARQL and Reasoning

- Important note:
  - By default, SPARQL does **not** use any reasoning
- Given the following T-box

```
:Museum rdfs:subClassOf :Building .
```

- and A-box

```
:Technoseum rdf:type :Museum .
```

- The result of

```
SELECT ?x WHERE {?x rdf:type :Building}
```

is empty!

SPARQL and Reasoning

- Important note:
  - By default, SPARQL does **not** use any reasoning
- Given the following T-box

```
:Museum rdfs:subClassOf :Building .
```

- and A-box

```
:Technoseum rdf:type :Museum .
```

- We would have to use something like

```
SELECT ?x WHERE {
    {?x rdf:type :Building } UNION
    {?x rdf:type ?t .
     ?t rdfs:subClassOf :Building }
}
```

Recap: Reasoning with Jena

- Given: a schema and some data

```
Model schemaModel = ModelFactory.createDefaultModel();
InputStream IS = new
  FileInputStream("data/example_schema.rdf");
schemaModel.read(IS);

Model dataModel = ModelFactory.createDefaultModel();
IS = new FileInputStream("data/example_data.rdf");
dataModel.read(IS);

Model reasoningModel =
    ModelFactory.createRDFSModel(schemaModel, dataModel);
```

- Now, `reasoningModel` contains all derived facts

Example: Jena + SPARQL + Reasoning

- Derived facts can also be queries with SPARQL
- Given the `reasoningModel`

```
Query q = QueryFactory.create(
     "SELECT ?t WHERE
     { <http://example.org/Madrid>
       <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
       ?t .}" );
   QueryExecution qexec =
     QueryExecutionFactory.create(q, reasoningModel);
   ResultSet rs = qexec.execSelect();
   while(rs.hasNext())
     String type = rs.next().get("t");
```

- Here, the query produces two solutions
  - `http://example.org/City`
  - `http://www.w3.org/2000/01/rdf-schema#Resource`

Accessing Public SPARQL Endpoints (Jena)

- SPARQL Endpoints are an important building block of the Semantic Web tool stack

- Access using Jena:

```
String query = "SELECT ...";
String endpoint = "http://dbpedia.org/sparql";
Query q = QueryFactory.create(strQuery);
QueryExecution qexec =
    QueryExecutionFactory.sparqlService(endpoint, q);
ResultSet RS = qexec.executeSelect();
```

Accessing Public SPARQL Endpoints (SPARQLWrapper)

- Querying the endpoint directly
  - Usually **SPARQLWrapper** is used

```python
from SPARQLWrapper import SPARQLWrapper, JSON

# Set up the endpoint
sparql = SPARQLWrapper("https://dbpedia.org/sparql")

# Define the query
query = """
SELECT ?state ?year
WHERE {
    dbr:Mannheim dbo:federalState ?state ;
                 dbo:foundingYear ?year .
}
"""

# Configure the request
sparql.setQuery(query)
sparql.setReturnFormat(JSON)

# Execute and fetch results
results = sparql.query().convert()

# Print results
for result in results["results"]["bindings"]:
    state = result["state"]["value"]
    year = result["year"]["value"]
    print(f"State: {state}, Year: {year}")
```

Triple Pattern Fragments (TPFs)

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Observation:
  - Operating SPARQL endpoints is costly
    - Hence, there are often downtimes
    - Maintenance often ends as project funding runs out
  - Accessing data via dumps or dereferencing is time consuming
    - See initial experiment
- Triple Pattern Fragments provide a middle ground solution

http://linkeddatafragments.org

Triple Pattern Fragments (TPFs)

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Only allow simple restrictions
  - i.e., only {?s ?p ?o} (each can be a URI or a variable)
- Provide results in a paged fashion
  - Estimated count
  - Links to further pages

Data Portal @ linkeddatafragments.org

#LD Linked Data Fragments

DBpedia 2014

Query DBpedia 2014 by triple pattern

subject:
predicate: http://www.w3.org/1999/02/22-rdf-syntax-ns#type
object: http://dbpedia.org/ontology/Restaurant

Find matching triples

Matches in DBpedia 2014 for *{ ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://...*

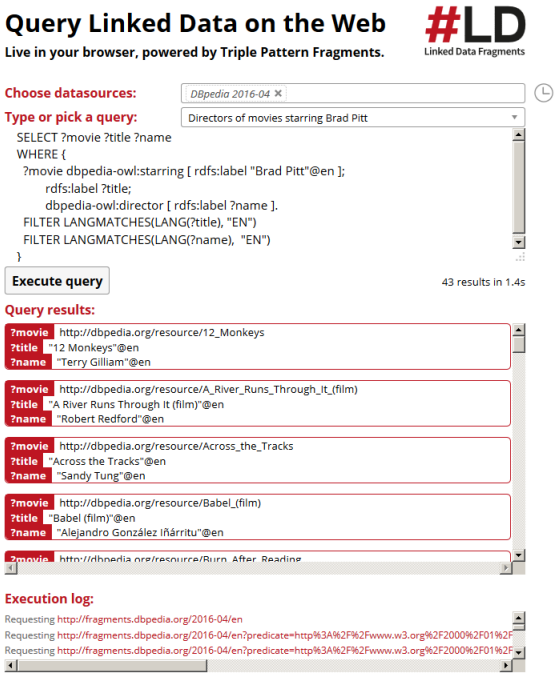
Showing triples 1 to 100 of ±1,213 with 100 triples per page. **next**

```
&samhoud_places  type  Restaurant.
't_Brouwerskolkje  type  Restaurant.
't_Fornuis  type  Restaurant.
't_Ganzenest  type  Restaurant.
't_Koetshuis  type  Restaurant.
't_Misverstant  type  Restaurant.
't_Nonnetje  type  Restaurant.
't_Schulten_Hues  type  Restaurant.
't_Veerhuis  type  Restaurant.
1321_Downtown_Taproom_Bistro  type  Restaurant.
21_Club  type  Restaurant.
36_on_the_Quay  type  Restaurant.
5_North_St  type  Restaurant.
68-86_Bar_and_Restaurant  type  Restaurant.
Aan_de_Poel  type  Restaurant.
Ad_Hoc_(restaurant)  type  Restaurant.
Akelare  type  Restaurant.
Alain_Ducasse_at_the_Dorchester  type  Restaurant.
Albannach_(restaurant)  type  Restaurant.
```


http://linkeddatafragments.org

Triple Pattern Fragments (TPFs)

- Most SPARQL queries can be solved by iteratively retrieving TPFs
  - Successively issuing new *selectors*
  - More targeted, i.e., less calls, than derefencing individual URIs



http://linkeddatafragments.org

Triple Pattern Fragments (TPFs)

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Example: astronauts born in capital countries

```
Select ?x ?y ?z where {
    ?x a dbpedia-owl:Astronaut .
    ?x dbpedia-owl:birthPlace ?y .
    ?z dbpedia-owl:capital ?y .
    ?z a dbpedia-owl:Country
}
```

+/- 651

±1,137,061

±5,034

±13,779

- Algorithm:
  - retrieve pattern: `?x a dbpedia-owl:Astronaut .` 653
  - for each result ?x retrieve: `?x dbpedia-owl:birthPlace ?y .` 1,209
  - for each result ?y retrieve: `?z dbpedia-owl:capital ?y .` 637
  - for each result ?z check: `?z a dbpedia-owl:Country .`

Triple Pattern Fragments vs. SPARQL

UNIVERSITY OF MANNHEIM
Data and Web Science Group

- Middle ground between
  - setting up a SPARQL server (costly for the server)
  - providing a full RDF dump (costly for the client)
- In our example, a SPARQL query was broken down into ~3k HTTP GET requests
  - Using clever index structures, this might still be faster
  - Results may also be streamed – allows for early stopping

Triple Pattern Fragments vs. SPARQL

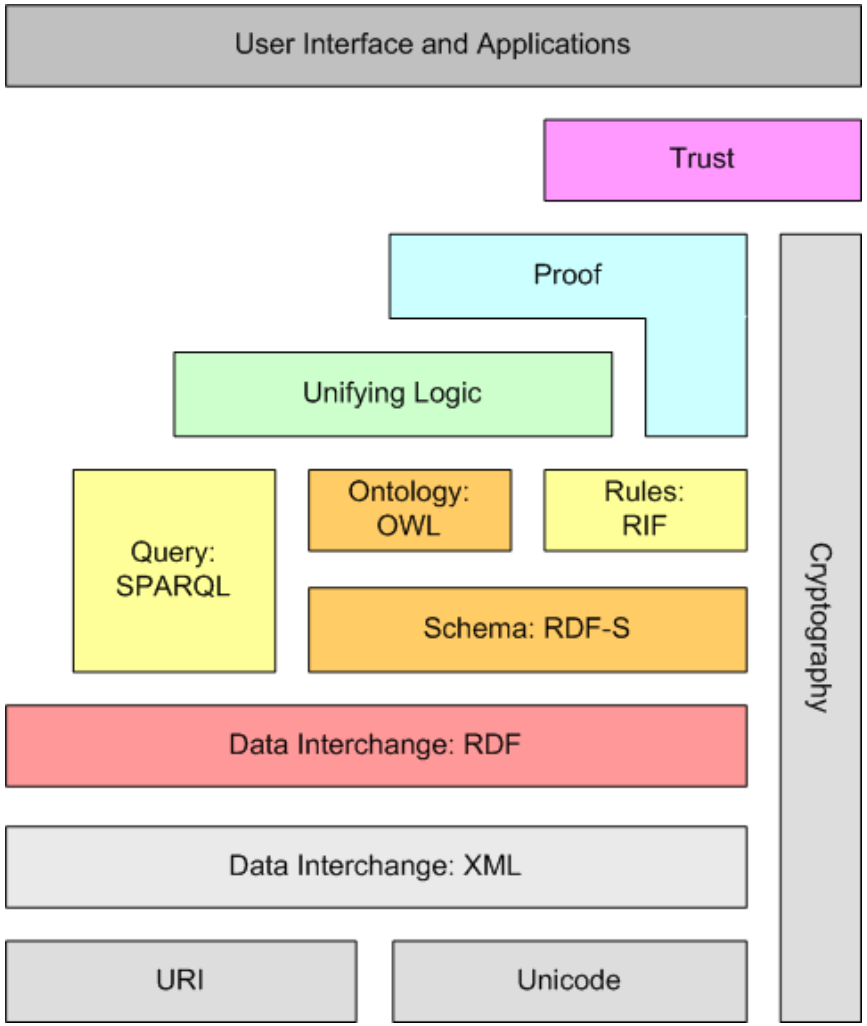
- All SPARQL constructs can be translated to a TPF query plan
- Some are quite fast
  - e.g., typical star-shaped queries
- Some are rather slow
  - e.g., regex queries for labels

Technology Stack

here be dragons...

Knowledge Graph Technologies
(This lecture)

Technical
Foundations



Berners-Lee (2009): *Semantic Web and Linked Data* http://www.w3.org/2009/Talks/0120-campus-party-tbl/

Questions?