Previously on Introduction to Linked Data…

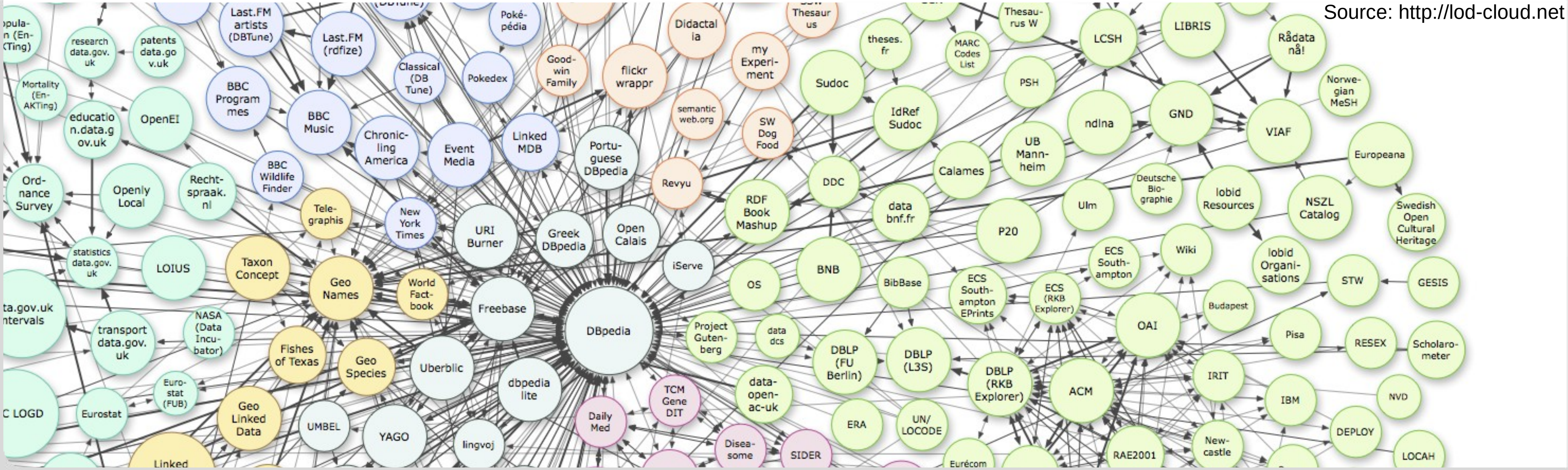
- You learned the benefits of a graph-structured data model and outline different serialization syntaxes for RDF graphs.
- You applied RDF lists in both the Turtle syntax shortcut and the triple representation, and reification in modelling.
- You have learned when two RDF graphs are subgraphs of each other.
- You understand whether one graph is an instance of another graph
- You are able to construct an RDF dataset from multiple RDF graphs.

C04 Querying RDF Datasets with SPARQL
How to access and query descriptions of things?

Version 2022-05-12

Lecturer: Prof. Dr. Andreas Harth

CHAIR OF TECHNICAL INFORMATION SYSTEMS



Source: http://lod-cloud.net

CC - Creative Commons Licensing

- This set of slides is part of the lecture „Semantic Web Technologies“ held at Karlsruhe Institute of Technology
- The content of the lecture was prepared by PD Dr. Andreas Harth based on his book „Introduction to Linked Data“
- The initial slides were prepared by Lars Heling with major modifications by Maribel Acosta

Agenda

1. **Introduction**
2. Structure of SPARQL Queries
3. Basic Graph Patterns
4. Group Graph Patterns
5. Filters, Functions and Modifiers
6. Querying Multiple (Named) RDF Graphs

Example Question

“What are the boroughs of Berlin?”

- How can we answer this question over RDF data?

Retrieving Data from a Dataset

- How to retrieve data from a dataset?
  - Queries are used in order to retrieve *relevant* data from a dataset

- Relational databases:
  - A set of tuples is stored in a table (Relation)
  - **S**tructured **Q**uery **L**anguage (SQL)

Relation: Cities

Name	Population	BoroughOf
Oststadt	21 091	Karlsruhe
Pankow	384 367	Berlin
…	…	…

```
SELECT Name
FROM Cities
WHERE BoroughOf = “Berlin” ;
```

- Graph databases:
  - What is a dataset in RDF?
  - How can we query data represented in RDF?

RDF Datasets

- A collection of graphs is called an RDF dataset.

- An RDF dataset has one default graph without a name,

and

- zero or more graphs with a name (a URI)

SPARQL

- Acronym:
  - **S**PARQL **P**rotocol **A**nd **R**DF **Q**uery **L**anguage

- Specified by W3C[1]
  - Current version: SPARQL 1.1 (March 2013)

- There are eleven SPARQL Recommendations, covering:
  - Syntax and semantics of queries over RDF
  - Protocol to pose queries against a SPARQL endpoint and to retrieve results
  - Various serialisations of query results
  - Entailment regimes
  - Update language
  - Federated query
  - …

[1] http://www.w3.org/TR/sparql11-overview/

Agenda

1. Introduction
2. **Structure of SPARQL Queries**
3. Basic Graph Patterns
4. Group Graph Patterns
5. Filters, Functions and Modifiers
6. Querying Multiple (Named) RDF Graphs

Back to Our Question

“What are the boroughs of Berlin?”

```
PREFIX ex: <http://example.org/cities.ttl#>

SELECT ?borough
FROM <http://example.org/cities.ttl>
WHERE {
      (Some conditions)
}
```

Components of SPARQL Queries (1)

```
PREFIX ex: <http://example.org/cities.ttl#>

SELECT ?borough
FROM <http://example.org/cities.ttl>
WHERE {
      (Some conditions)
}
```

Prefix definitions:

- PREFIX keyword to introduce CURIEs
- Subtly different from Turtle syntax
  - The final period is not used
  - No “@” at the beginning

Components of SPARQL Queries (2)

```
PREFIX ex: <http://example.org/cities.ttl#>

SELECT ?borough
FROM <http://example.org/cities.ttl>
WHERE {
      (Some conditions)
}
```

Query form:

- `ASK, SELECT, DESCRIBE,` or `CONSTRUCT`
- Details in a bit…

Components of SPARQL Queries (3)

```
PREFIX ex: <http://example.org/cities.ttl#>

SELECT ?borough
FROM <http://example.org/cities.ttl>
WHERE {
      (Some conditions)
}
```

Variable projection:

- Variables are “placeholders” for RDF terms
- Variables are prefixed using “?” or “$”
- To select all variables contained in a query: “`SELECT * `“

Components of SPARQL Queries (4)

```
PREFIX ex: <http://example.org/cities.ttl#>

SELECT ?borough
FROM <http://example.org/cities.ttl>
WHERE {
      (Some conditions)
}
```

Dataset selection:

- `FROM` or `FROM NAMED` keyword to specify the RDF dataset
- Indicates the sources for the data against which to find matches

Components of SPARQL Queries (5)

```
PREFIX ex: <http://example.org/cities.ttl#>

SELECT ?borough
FROM <http://example.org/cities.ttl>
WHERE {
      (Some condition)
}
```

Query pattern:

- Specifies *what* we want to query
- Contains graph patterns that are matched against RDF data

Components of SPARQL Queries (6)

```
PREFIX ex: <http://example.org/cities.ttl#>

SELECT ?borough
FROM <http://example.org/cities.ttl>
WHERE {
      (Some condition)
} ORDER BY ?borough
```

Sequence modifiers:

- Modify the result set (query answers)
- `ORDER BY` changes the order of the result set
- `LIMIT, OFFSET` selects chunks of the result set
- `DISTINCT` (after `SELECT`), removes duplicate answers

Query Forms

- There are four different query forms that SPARQL supports:
  - `SELECT`

    Return all or a subset of the solution mappings
  - `CONSTRUCT`

    Return a set of triples/a graph, where the mappings are filled into a specific graph pattern template
  - `ASK`

    Return true or false, depending on whether there is a solution mapping or graph pattern
  - `DESCRIBE`

    Return a set of triples / a graph that describes a certain resource (URI)

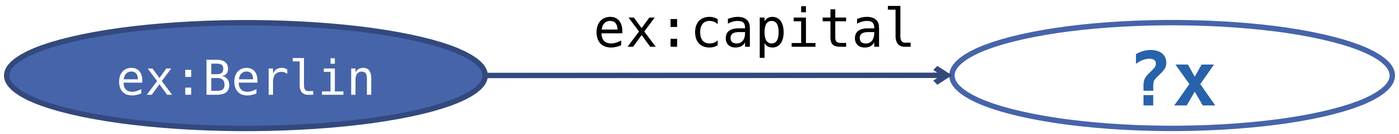
Agenda

1. Introduction
2. Structure of SPARQL Queries
3. **Basic Graph Patterns**
4. Group Graph Patterns
5. Filters, Functions and Modifiers
6. Querying Multiple (Named) RDF Graphs

Triple Patterns

- Building block of SPARQL queries: **triple patterns.**
  - Similar to RDF triples but with variables (specified with ? or $).

- **Example:** Berlin is the capital of ____________.

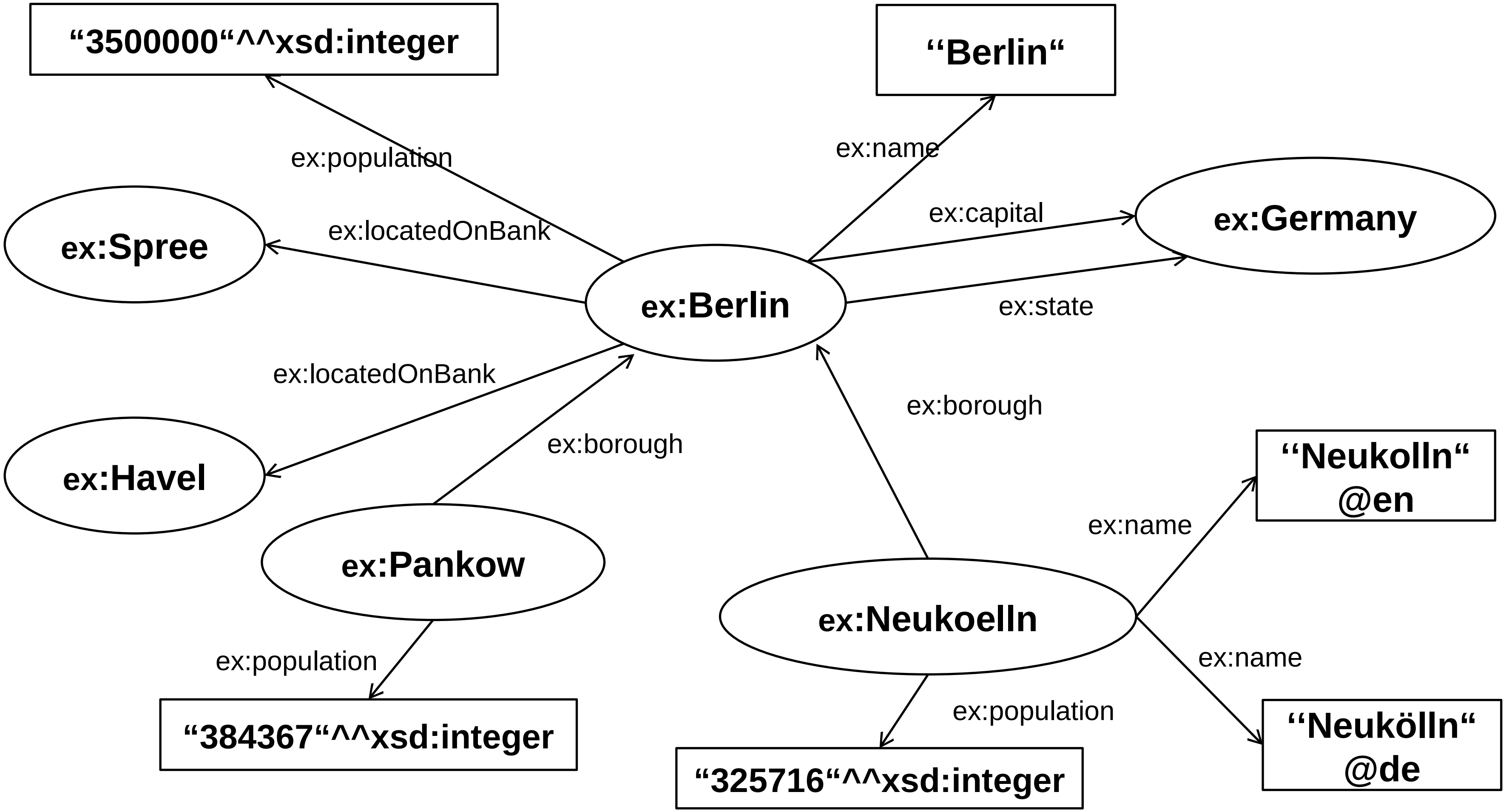


Or:

```
ex:Berlin ex:capital ?x .
```

http://example.org/cities.ttl

“What are the boroughs of Berlin?”



http://example.org/cities.ttl

“What are the boroughs of Berlin?”

```
{
  ?berlin ex:name ”Berlin“ .
  ?borough ex:borough ?berlin .
}
```

Basic Graph Pattern (1)

- Basic Graph Pattern (BGP) contains several triple patterns.
- BGPs represent *conjunction* of triple patterns.

- **Example:** The following BGP obtains the boroughs of `ex:Berlin` **and** the population of the boroughs

```
{
      ?borough ex:borough    ex:Berlin .
      ?borough ex:population ?population .
}
```

A variable may be used on the subject, predicate or object position

Basic Graph Pattern (2)

- BGPs can be specified using Turtle syntax
  - Example:

```
{ ?borough ex:borough      ?berlin ;
           ex:population   ?population .
  ?berlin  ex:name         “Berlin” . }
```

- In BGPs blank nodes are treated similar to variables.
  - Example:

```
{  _:bn1   ex:name   ?name .
   _:bn1   ex:population ?population . }
```

  - But: blank nodes may only appear on subject and object position of a triple pattern.

- In contrast to variables, one may not specify blank nodes in the query form (e.g., `SELECT`)

Think-Pair-Share

Write a SPARQL query against the following RDF graph to *retrieve the country where `dbr:Barack_Obama` was born.* Assume the graph is available at `http://example.org/dbpedia`.

```
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.org/cities.ttl#> .
dbr:Barack_Obama foaf:name "Barack Obama"@en ;
                 dbo:spouse dbr:Michelle_Obama ;
                 dbo:birthPlace dbr:Honolulu .
dbr:Hasso_Plattner dbo:birthPlace ex:Berlin .
dbr:Honolulu dbo:country dbr:United_States .
ex:Berlin dbo:country dbr:Germany .
```

Agenda

1. Introduction
2. Structure of SPARQL Queries
3. Basic Graph Patterns
4. **Group Graph Patterns**
5. Filters, Functions and Modifiers
6. Querying Multiple (Named) RDF Graphs

Group Graph Patterns

- Graph patterns can be grouped using `“{ }”`
- Group Graph Patterns are used to specify more elaborate queries
- SPARQL features many complex graph pattern constructs
- We start with `UNION` and `OPTIONAL`
- And later consider `GRAPH` graph patterns

Optionals and Alternatives in Group Graph Patterns

- Optional triple patterns can be specified using the **`OPTIONAL`** keyword.
  - **Example:** *Retrieve the capital of `ex:Germany` and, if available, the total population of the capital.*

```
?x ex:capital ex:Germany .
OPTIONAL { ?x ex:population ?y . }
```

- Disjunctions of triple patterns can be specified using the **`UNION`** keyword.
  - **Example:**

```
{ ex:Neukoelln ex:population ?y . }
UNION
{ ex:Pankow ex:population ?y . }
```

Agenda

1. Introduction
2. Structure of SPARQL Queries
3. Basic Graph Patterns
4. Group Graph Patterns
5. **Filters, Functions and Modifiers**
6. RDF Datasets and Named Graphs

Filters - Introduction

- Filters are used to check conditions
- These conditions are specified in the `WHERE` clause
- Specified using the `FILTER` keyword

- Example: *Retrieve all boroughs of `ex:Berlin` with more than 350.000 inhabitants.*

```
{
   ?borough ex:borough     ex:Berlin ;
            ex:population  ?population .

   FILTER(?population > 350000)

   }
```

Components of a Filter

- Result of a filter:
  - True/false
  - Error

- Operators for filter expressions:
  - <, =, >, <=, >=, !=
  - Usable on numeric types, strings, `xsd:dateTime` and `xsd:boolean`

- Arithmetic operators:
  - +, -, *, /

- Filter expressions can be combined using:
  - AND (&&), OR (| |), NOT (!)

Comparing Literals in Filters vs. BGP Matching

- Filter conditions take into account the datatype of compared literals (similar to *D*-entailment).

- But Basic Graph Pattern matching (the expression in the WHERE clause) does not take into account the datatypes of literals (similar to simple entailment)

Using Functions in SPARQL

- Functions can be used within the filter expression
- Examples:

Function	Return value	Description
STR(L∪U)	simple literal	Returns the lexical form of a literal or the codepoint representation of an URI
LANG(L)	simple literal	Returns the language tag of a literal. Return “” if literal has no language tag
DATATYPE(L)	URI	Returns the datatype URI of a literal

- Further examples include functions on numbers (xsd:integer, xsd:decimal…), e.g., ABS(), which takes the absolute value of a number.

Note, that this is just a selection of frequently used functions.
For a complete list please refer to the W3C SPARQL Recommendation.[1]

[1] http://www.w3.org/TR/2013/REC-sparql11-query-20130321/#func-rdfTerms

How to Use Functions

- We want only want to retrieve boroughs of `ex:Berlin` where there is an English name for them
- Using a filter and a function:

```
{
   ?borough  ex:borough ex:Berlin ;
                ex:name    ?name .

   FILTER(lang(?name) = “en”)
   }
```

- In the given example graph only `:Neukoelln` would be retrieved, since it is the only resource which has a `:name` with an en-language tag

Assigning Values to Variables

- To be able to assign values to new variables we can use the `BIND` keyword
- Values can be calculated using basic arithmetic operations or can be the result of applying a function

- Example:
  - Retrieving the difference in population of two boroughs

```
{
   ex:Pankow       ex:population ?pPop .
   ex:Neukoelln  ex:population ?nPop .

   BIND ( abs(?pPop - ?nPop) AS ?diffPop )
   }
```

Modifying the Result Set

- Modifiers are used in order to edit the solution mapping
- There are four different modifiers that SPARQL supports:
  - `ORDER BY`

    Sort according to the order specified in `FILTER` comparison operator.
    Possible to order by ASC (default) or DESC

  - `LIMIT`

    Specify a limited number of results to be returned.

  - `OFFSET`

    Solutions start after a specified number of solution mappings,
    i.e., `OFFSET x` discards the first x solution mappings.
    Note: An offset of ‘0’ has no effect.

    If there are several solutions with the same terms, the solution is only returned once.

  - `SELECT DISTINCT`

Sorting Results

- Variables can bind to arbitrary RDF terms (URIs, literals, blank nodes)
- The sort order for RDF terms (lowest to highest):
  - No value assigned to variable or expression in this solution
  - Blank Nodes
  - URIs
  - Literals

Modifiers - Example

- Example query using modifiers:

```
PREFIX : <http://example.org/cities.ttl#>

SELECT DISTINCT ?borough
FROM <http://example.org/cities.ttl>
WHERE {
       ?borough ex:borough        ex:Berlin ;
                ex:name    ?name .
}
ORDER BY ?name

LIMIT 10

OFFSET 2
```

A solution mapping is only returned once

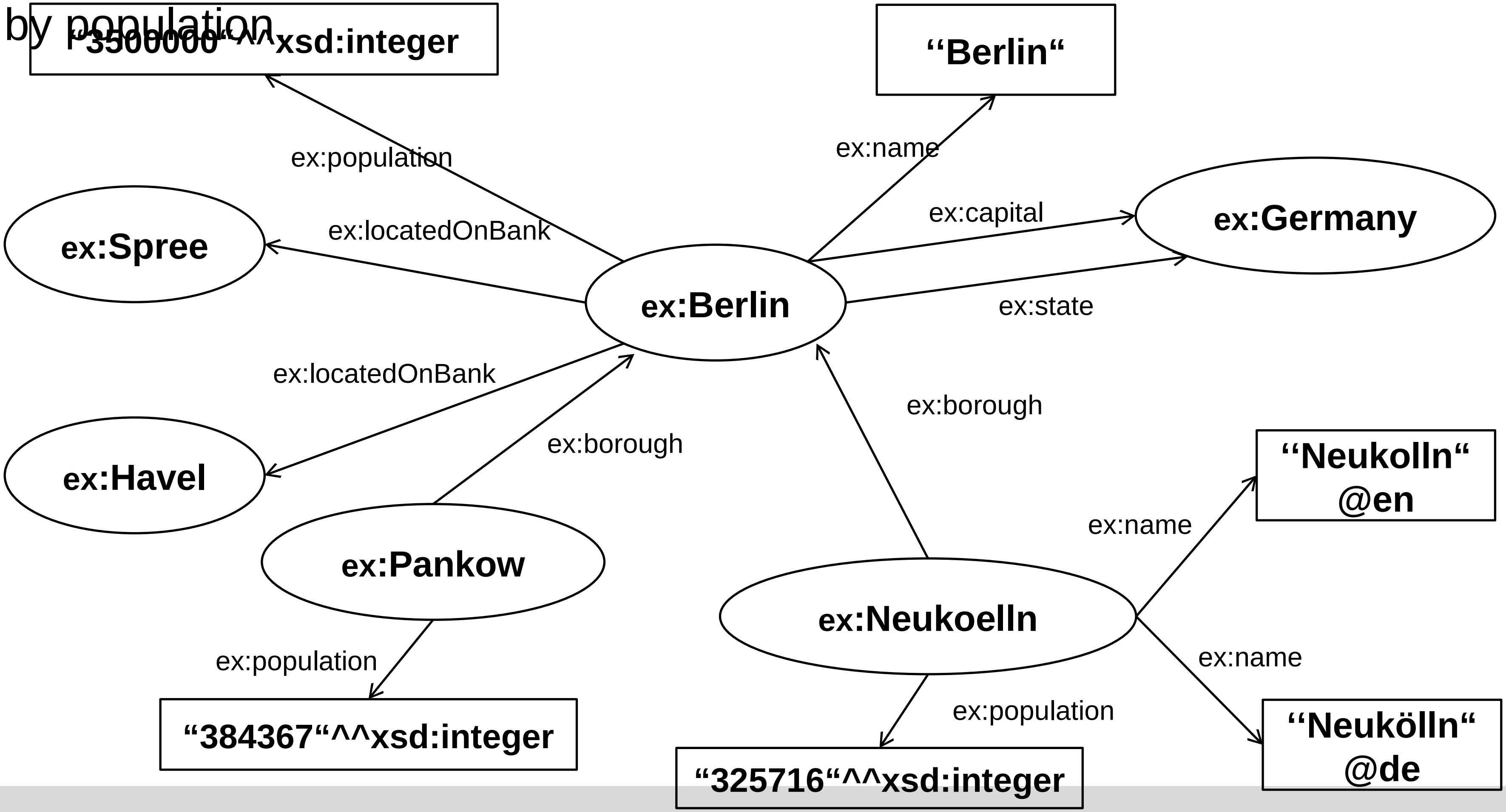
Order (ascending) by `?name`

Maximum of 10 results

Discard the first 2 results

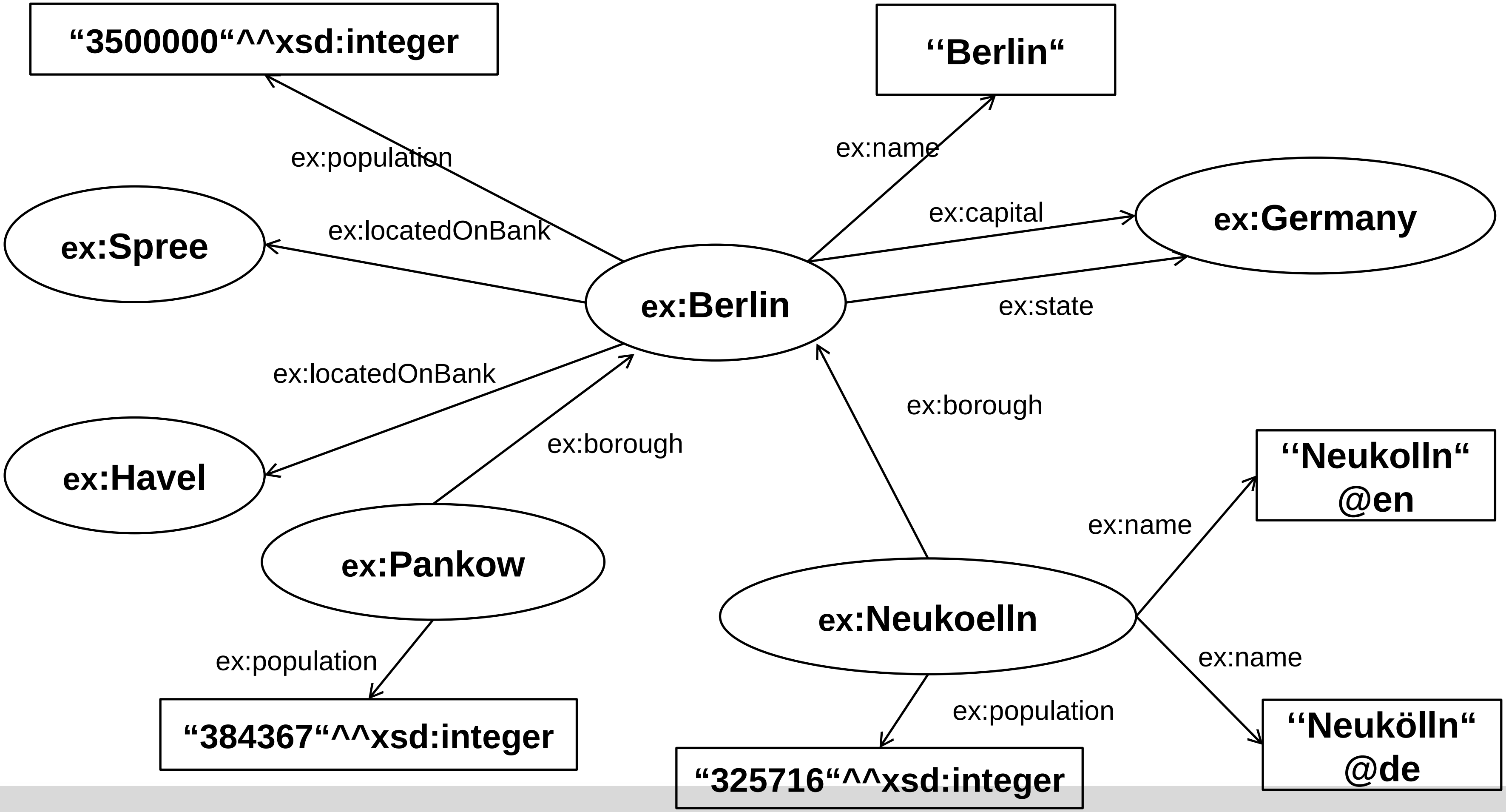
Think-Pair-Share

Write a SPARQL query against the dataset `http://example.org/cities.ttl` to retrieve boroughs of `ex:Berlin` whose population is less or equal than `350000` and ordered (descending) by population



Think-Pair-Share

Write a SPARQL query against the dataset `http://example.org/cities.ttl` to retrieve the borough of `:Berlin` with the second highest population.

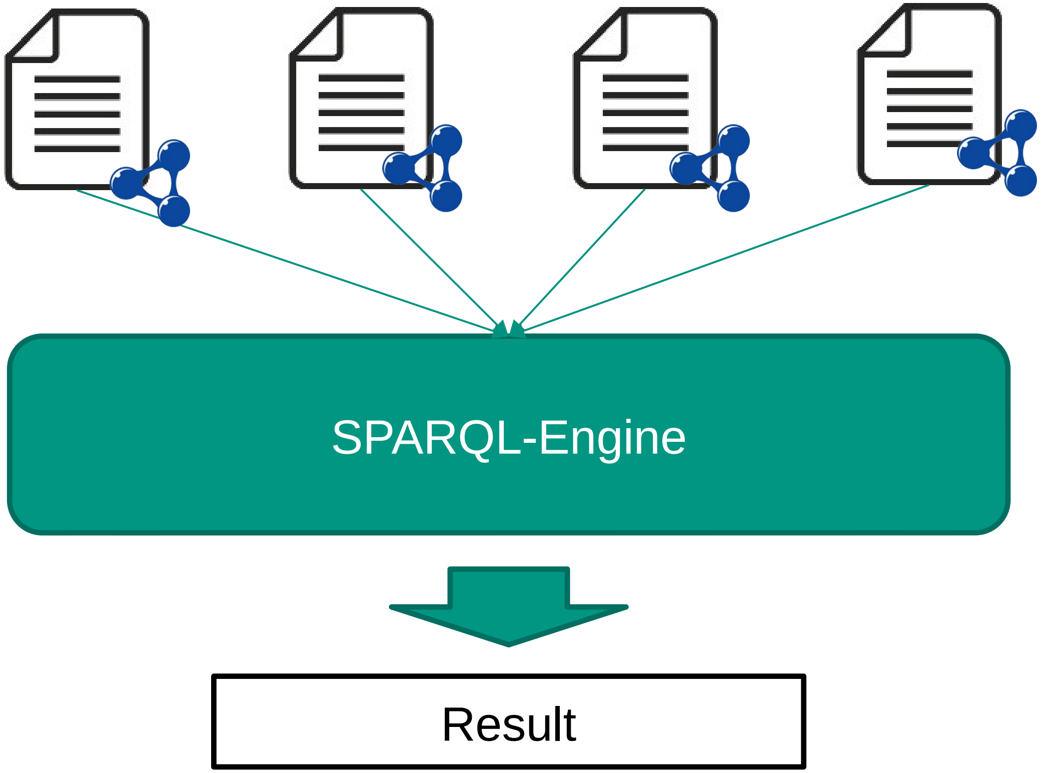


Agenda

1. Introduction
2. Structure of SPARQL Queries
3. Basic Graph Patterns
4. Group Graph Patterns
5. Filters, Functions and Modifiers
6. **Querying Multiple (Named) RDF Graphs**

Multiple Graphs

- Information may be spread over several documents
- Therefore, several documents should be addressable in a query



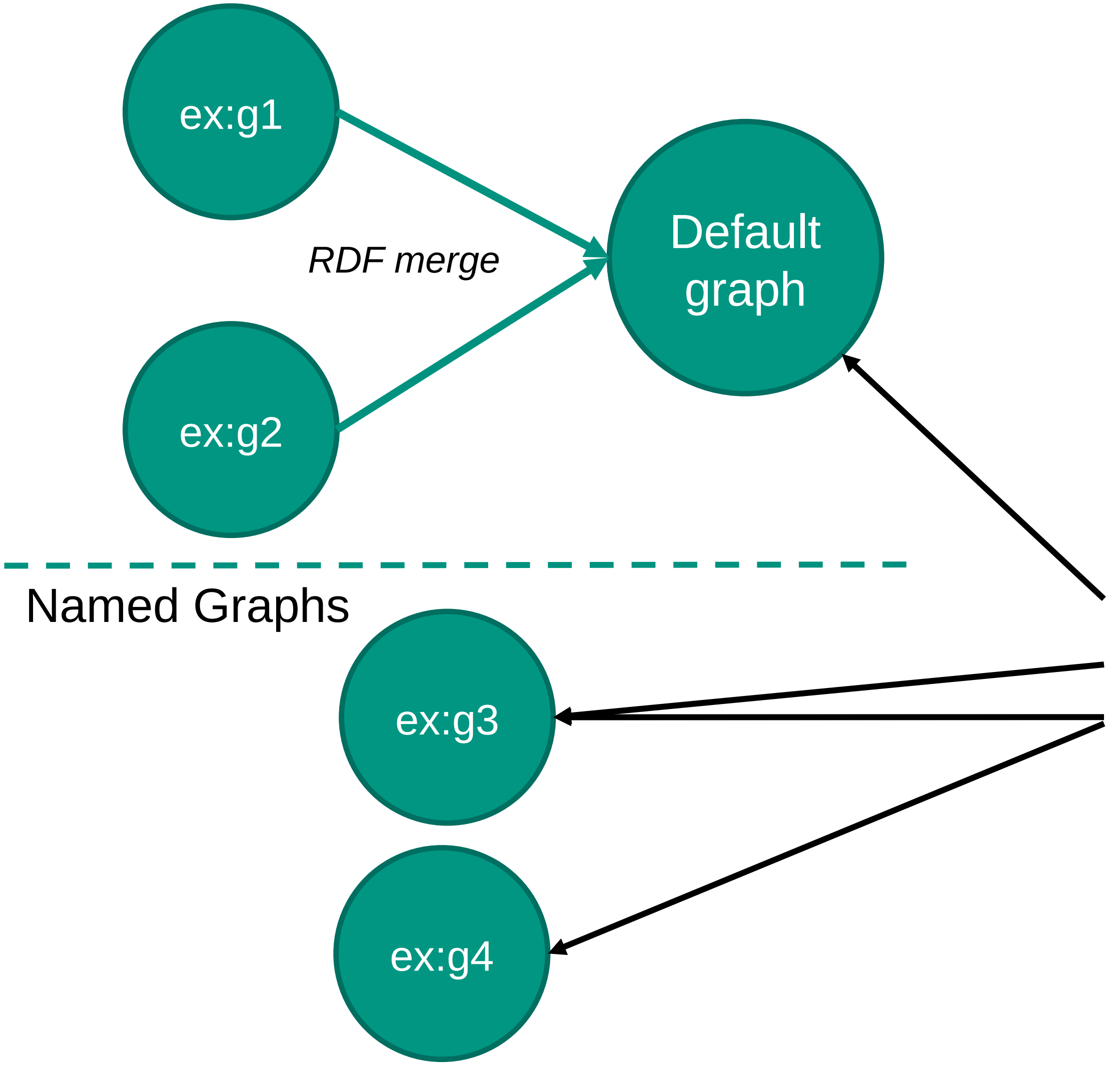
Multiple Graphs

- SPARQL supports handling multiple graphs:
  - These graphs may be different data sources
  - Graphs can be added using the `FROM` keyword
  - All graphs specified in the `FROM` clause are combined to a default graph

- SPARQL supports handling of multiple **named** graphs:
  - Using the `FROM NAMED` keyword
  - These graphs can be accessed using the `GRAPH` keyword
  - Used to query data from specific graphs

➡ To identify the triples belonging to a graph data we extend the triple model to quadruples, to be able to hold information on the context (name of the graph).

Multiple Graphs - Example



```
PREFIX ex : <...>
SELECT *
FROM ex:g1
FROM ex:g2
FROM NAMED ex:g3
FROM NAMED ex:g4
WHERE
{
…
?s    ?p    ?o.
GRAPH ex:g3 { ... }
GRAPH ?graph { ... }
}
```

Think-Pair-Share

Given the RDF graphs available at `http://example.org/cities.ttl` and `http://example.org/dbpedia`.

1. Write a SPARQL query to *retrieve the boroughs of the city where `dbr:Hasso_Plattner` was born*.

2. Write a SPARQL query to *retrieve the country where ex:Berlin is located and the URI of the graph that contains that data.*

SPARQL Query Processors vs. SPARQL Endpoints

Query Processor

- Acts as user agent
- Graphs are retrieved via HTTP during query processing
- Default graph is empty, so queries require FROM/FROM NAMED clauses

Endpoint

- Acts as server
- Graphs are indexed and stored on disk during installation (like a database)
- Default graph is configured, so no FROM/FROM NAMED clauses needed

Summary of Core SPARQL Features

- Basic concepts: Triple patterns
- SPARQL Query structure:
  - Prefix declarations: `PREFIX`
  - Query forms: `ASK, SELECT, DESCRIBE, CONSTRUCT`
  - Variable projection: Subset of variables that we want to return
  - Dataset selection: `FROM, FROM NAMED`
  - Query patterns
    - Basic Graph Patterns (BGP)
    - Graph Patterns (`UNION, OPTIONAL, GRAPH`)
    - Functions (`FILTER, BIND AS`)
  - Sequence modifiers: `ORDER BY, LIMIT, OFFSET, DISTINCT`

Learning Goals

- G 4.1 Write BGP queries in SPARQL with query forms (SELECT, CONSTRUCT, ASK and DESCRIBE).
- G 4.2 Use FROM, FROM NAMED and GRAPH in queries in conjunction with RDF datasets.
- G 4.3 Correctly apply UNION and OPTIONAL in queries.
- G 4.4 Use FILTER and BIND ... AS in conjunction with expressions involving functions.
- G 4.5 Describe the handling of typed literals in SPARQL graph pattern matching and filter expressions.
- G 4.6 Apply ORDER BY, LIMIT and OFFSET in queries.

Outlook – Chapter 5

- We have seen how to write SPARQL queries.
- In the next lecture, we learn how to evaluate SPARQL queries.

- We are still only concerned with one (out of the eleven) SPARQL Recommendations, covering:
  - Syntax and semantics of queries over RDF
  - Protocol to pose queries against a SPARQL endpoint and to retrieve results
  - Various serialisations of query results
  - Entailment regimes
  - Update language
  - Federated query
  - …