Matteo Palmonari

matteo.palmonari@disco.unimib.it

2.4
Querying RDF Knowledge Graphs
SPARQL

INSID&S Lab
***In**teraction and **S**emantics for*
***I**nnovation with **D**ata & **S**ervices*

SPARQL

Query RDF data

01 – Introduction to SPARQL

Scope of this Presentation and Additional Online Resources

- Scope of this presentation
  - Explanation of key principles and features of SPARQL (shortened and modified version of EUCLID slides – see below)
- More details on SPARQL (online):
  - EUCLID Module 2 - SPARQL: Querying Linked Data
    - Available at: https://www.slideshare.net/EUCLIDproject/querying-linked-data)
    - EUCLID project: training for semantic technologies, 6 modules, slides available with CC license at http://euclid-project.eu/
  - SPARQL specification:
    - https://www.w3.org/TR/sparql11-overview/ (overview of SPARQL)
    - https://www.w3.org/TR/sparql11-query/ (language specification)

How to Get Familiar with SPARQL

- Get familiar with RDF data by browsing DBpedia online
  - E.g., search for “Elton John dbpedia” (or a singer you like better), jump to its page, follow the links and explore more data; check in particular properties and types (for the moment, types are values of the property rdf:type; we will further discuss types in the next lessons)
- Understand the key principles underpinning SPARQL
  - This presentation
- Make exercises
  - SPARQL hands-on session in this course (next lesson)
  - Make queries to the DBpedia SPARQL portal at http://dbpedia.org/sparql
  - While making queries use the SPARQL specification as a reference:
    - https://www.w3.org/TR/sparql11-query/ (language specification)

Outline of These Lessons (2)

- Introduction
  - What is SPARQL
- Basic Query Patterns (Tabular Results)
  - Idea
  - Semantics
- Graph Traversal Queries (Graph Queries)
  - Idea
  - Semantics
- Query Forms, Modifiers, Reasoning, etc.

1st lesson

1st lesson

2nd lesson

2nd lesson

SPARQL Query: Example

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX mo: <http://purl.org/ontology/mo/>

SELECT ?album
FROM <http://musicbrainz.org/20130302>
WHERE {
    dbpedia:The_Beatles foaf:made ?album .
    ?album a mo:Record ; dc:title ?title
}
ORDER BY ?title
```

Agenda

EUCLID
EdUcational Curriculum for the usage of LInked Data

1. Introduction to SPARQL
2. Querying Linked Data with SPARQL
3. Updating Linked Data with SPARQL 1.1
4. SPARQL Protocol
5. Reasoning over Linked Data

Agenda

EUCLID
EdUcational Curriculum for the usage of LInked Data

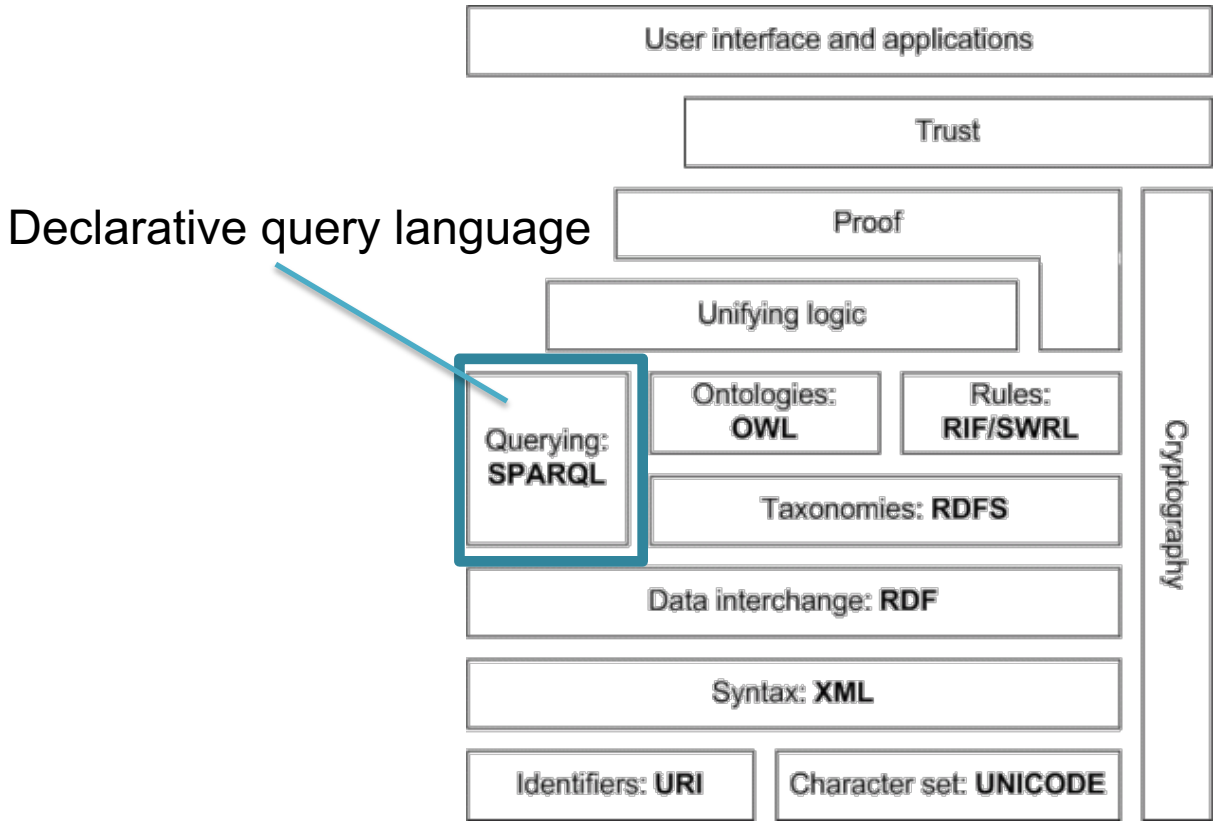
1. **Introduction to SPARQL**
2. **Querying Linked Data with SPARQL**
3. Updating Linked Data with SPARQL 1.1
4. SPARQL Protocol
5. Reasoning over Linked Data

More details at: http://euclid-project.eu/modules/chapter2.html

SPARQL

EUCLID
EdUcational Curriculum for the usage of LInked Data

* Protocol and RDF Query Language



Semantic Web Stack
Berners-Lee (2006)

SPARQL

EUCLID
EdUcational Curriculum for the usage of LInked Data

- **SPARQL Query**
  - **Declarative query language for RDF data**
  - **http://www.w3.org/TR/rdf-sparql-query/**
- SPARQL Algebra
  - Mapping from queries to relational algebra for result manipulation
  - http://www.w3.org/2001/sw/DataAccess/rq23/rq24-algebra.html
- SPARQL Update
  - Declarative manipulation language for RDF data
  - http://www.w3.org/TR/sparql11-update/
- SPARQL Protocol
  - Standard for communication between SPARQL services and clients
  - http://www.w3.org/TR/sparql11-protocol/

SPARQL Query 1.1

- **SPARQL 1.0** only allows accessing the data (**query**)
- **SPARQL 1.1** introduces:

Query extensions

- Aggregates, Subqueries, Negation, Expressions in the SELECT clause, **Property paths**, assignment, short form for CONSTRUCT, expanded set of functions and operators

Updates

- **Data management**: Insert, Delete, Delete/Insert
- **Graph management**: Create, Load, Clear, Drop, Copy, Move, Add

Federation extension

- Service, values, service variables (informative)

SPARQL

Query RDF data

02 –SPARQL Basics

SPARQL Basics

EUCLID
EdUcational Curriculum for the usage of LInked Data

- **RDF graph:** Set of RDF assertions, manipulated as a labeled directed graph.
- **RDF data set:** set of RDF triples. It is comprised of:
  - One default graph
  - Zero or more named graphs
- **SPARQL protocol client:** HTTP client that sends requests for SPARQL Protocol operations (queries or updates)
- **SPARQL protocol service:** HTTP server that services requests for SPARQL Protocol operations
- **SPARQL endpoint:** The URI at which a SPARQL Protocol service listens for requests from SPARQL clients

SPARQL Basics / KGBook

EUCLID
EdUcational Curriculum for the usage of LInked Data

RDF data set:

- One default graph
- Zero or more named graphs

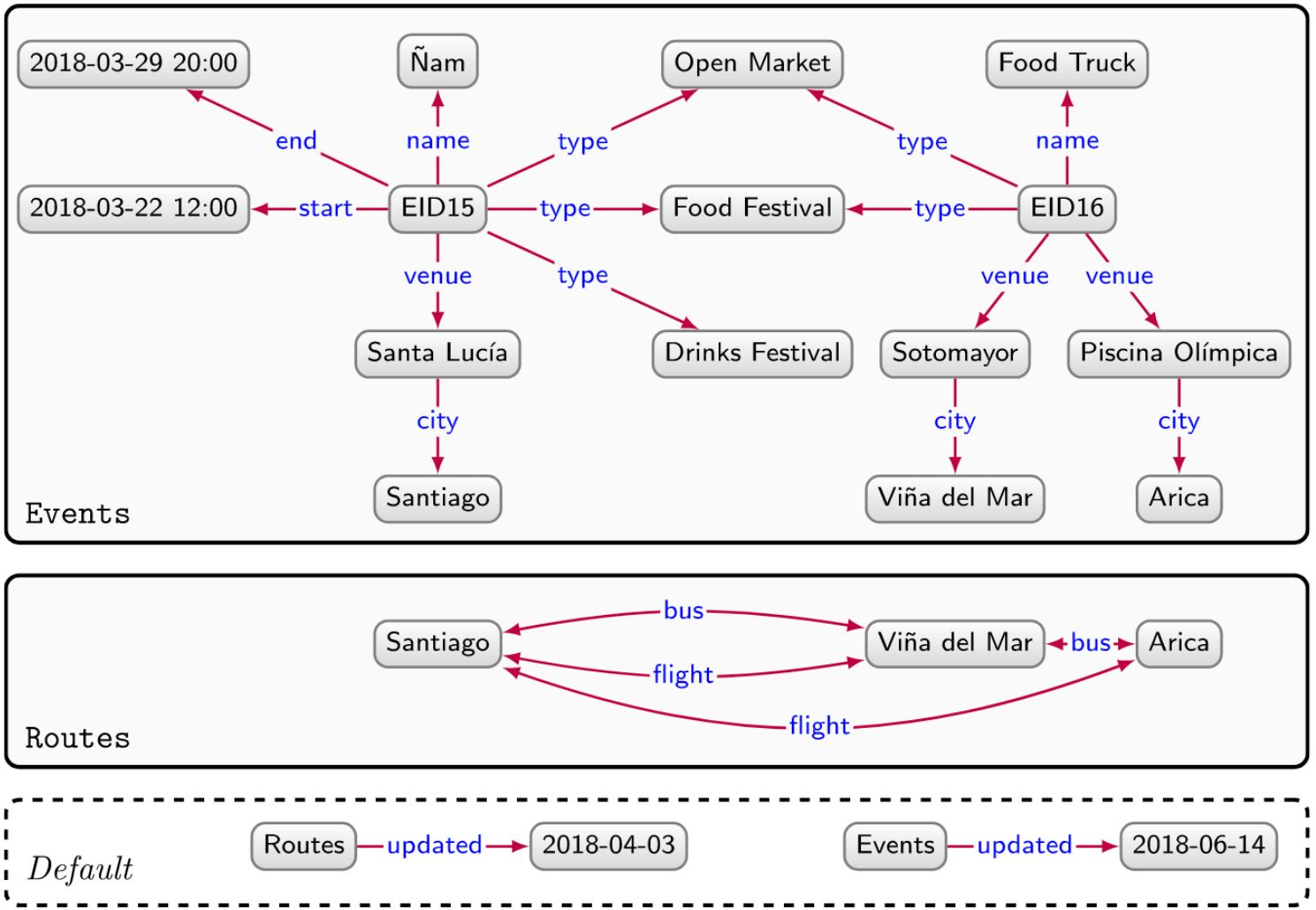


Figure 2.4: Graph dataset based on directed edge-labelled graphs with two named graphs and a default graph describing events and routes

SPARQL Basics

EUCLID
EdUcational Curriculum for the usage of LInked Data

- **RDF graph:** Set of RDF assertions, manipulated as a labeled directed graph.
- **RDF data set:** set of RDF triples. It is comprised of:
  - One default graph
  - Zero or more named graphs
- **SPARQL protocol client:** HTTP client that sends requests for SPARQL Protocol operations (queries or updates)
- **SPARQL protocol service:** HTTP server that services requests for SPARQL Protocol operations
- **SPARQL endpoint:** The URI at which a SPARQL Protocol service listens for requests from SPARQL clients

SPARQL Basics

EUCLID
EdUcational Curriculum for the usage of LInked Data

- **RDF triple:** Basic building block, of the form subject, predicate, object. Example:

```
dbpedia:The_Beatles foaf:name "The Beatles" .
```

- **RDF triple pattern:** Contains one or more variables. Examples:

```
dbpedia:The_Beatles foaf:made ?album.
?album mo:track ?track .
?album ?p ?o .
```

- **RDF quad pattern:** Contains graph name: URI or variable. Examples:

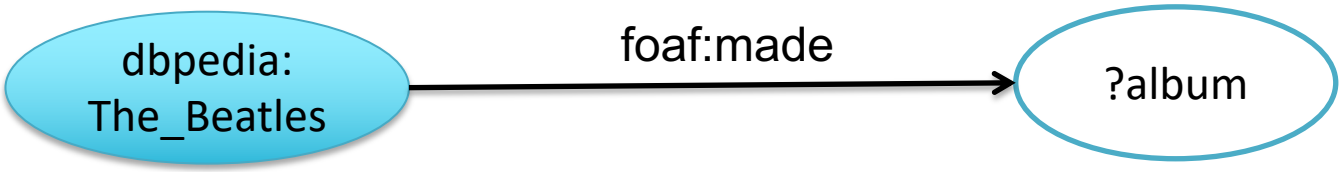
```
GRAPH <:g> {:s :p :o .}
GRAPH ?g {dbpedia:The_Beatles foaf:name ?o.}
```

SPARQL Query

EUCLID
EdUcational Curriculum for the usage of LInked Data

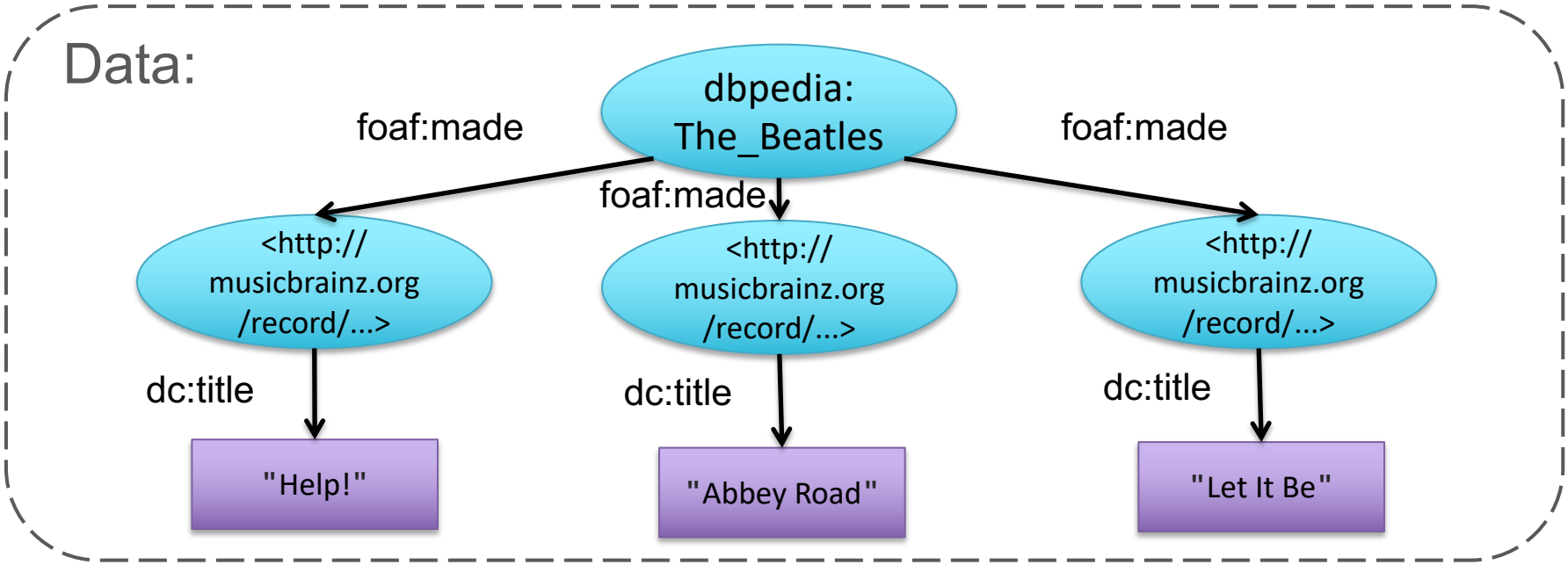
Main idea: **Pattern matching**

- Queries describe sub-graphs of the queried graph
- **Graph patterns** are RDF graphs specified in Turtle syntax, which contain variables (prefixed by either “?” or “$”)



- Sub-graphs that match the graph patterns yield a **result**

SPARQL Query

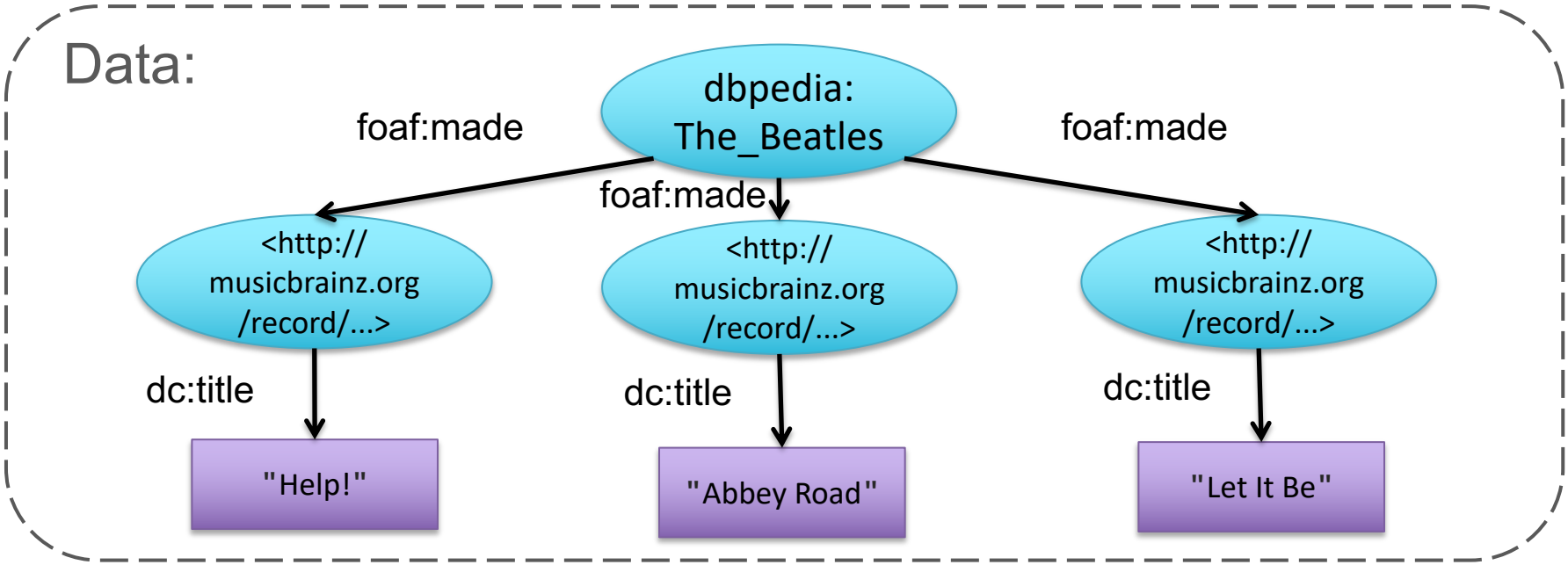


Graph pattern:

Results:

?album

SPARQL Query

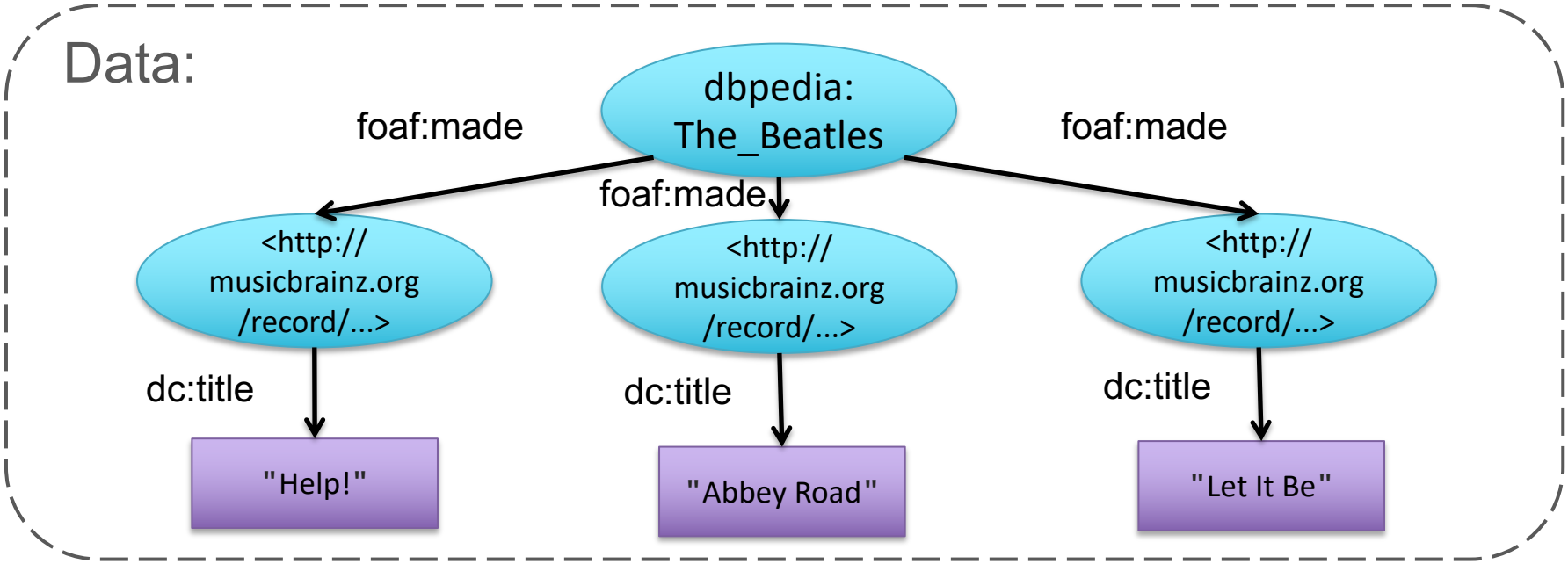


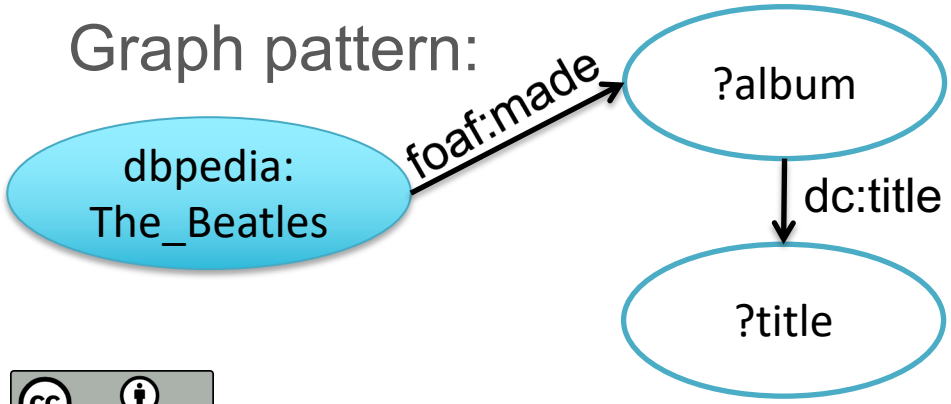
Graph pattern:

Results:

?album
<http://musicbrainz.org...>
<http://musicbrainz.org...>
<http://musicbrainz.org...>

SPARQL Query

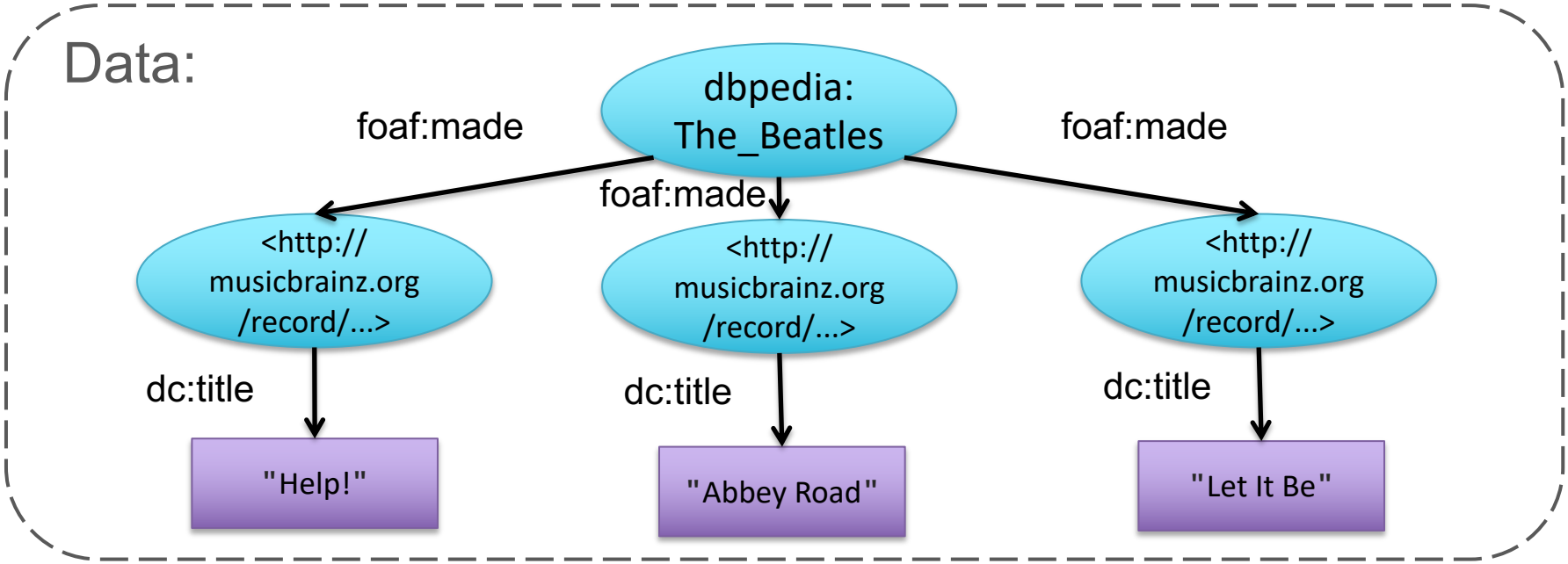


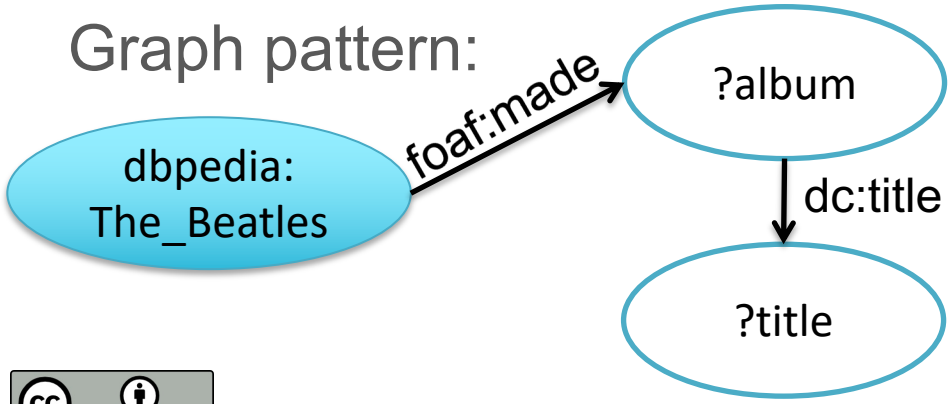


Results:

?album	?title

SPARQL Query

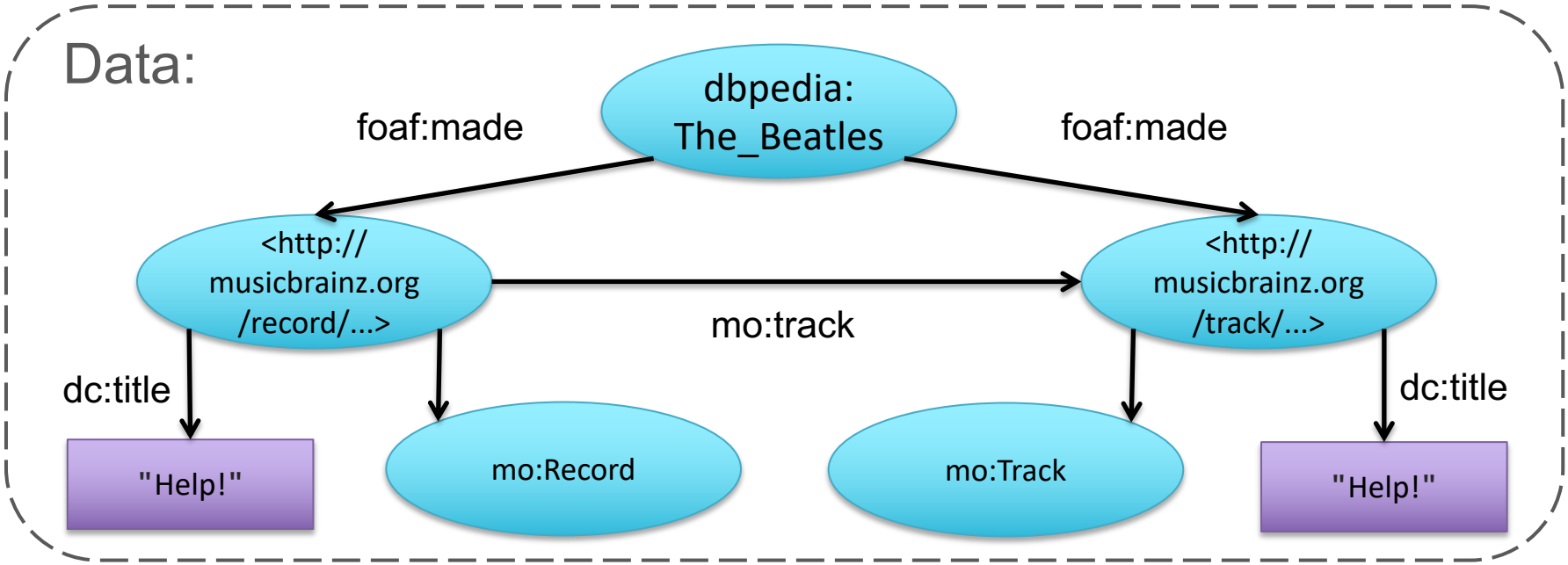


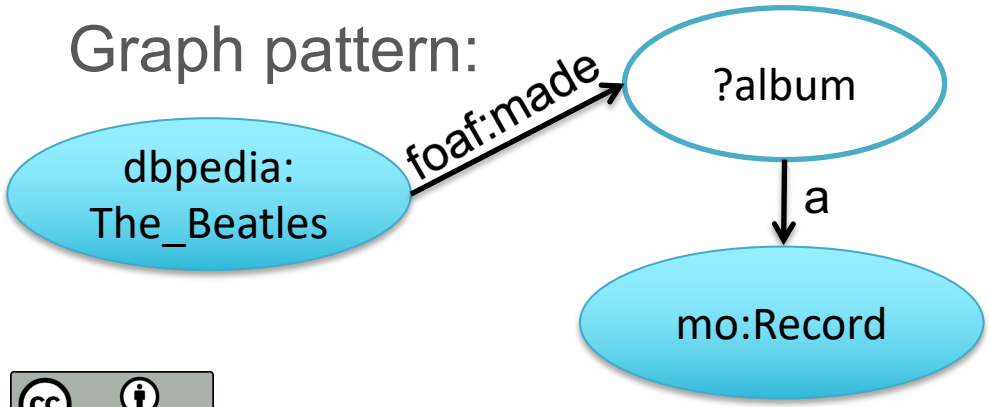


Results:

?album	?title
<http://...>	"Help!"
<http://...>	"Abbey Road"
<http://...>	"Let It Be"

SPARQL Query

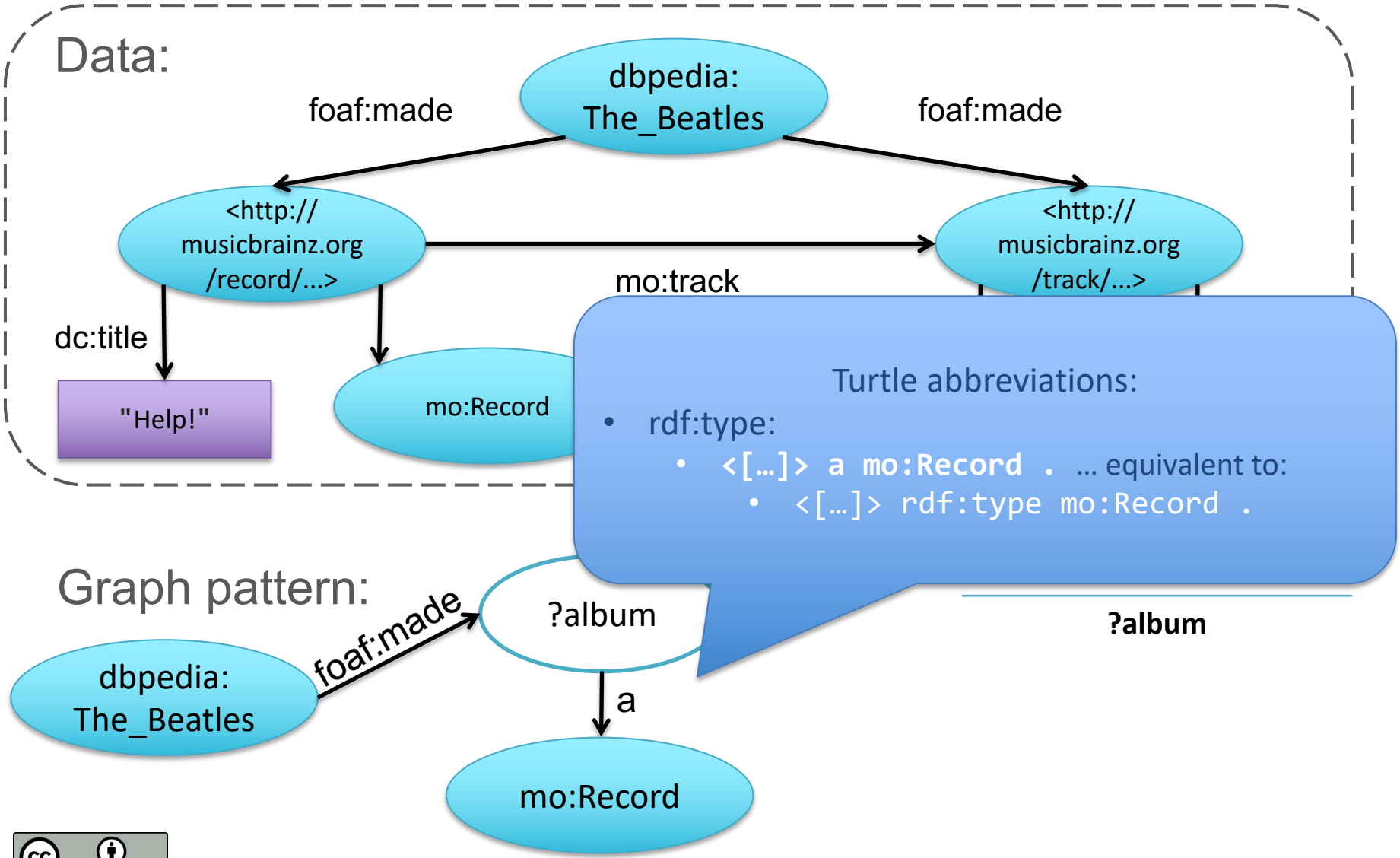




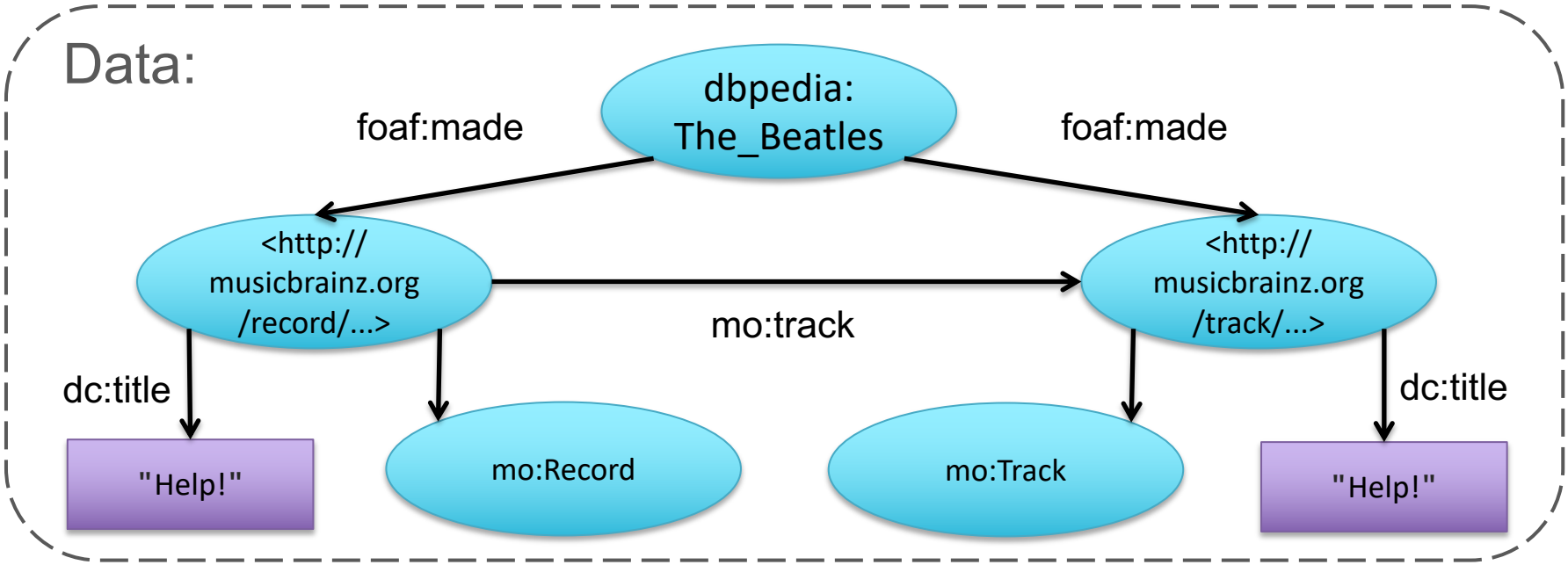
Results:

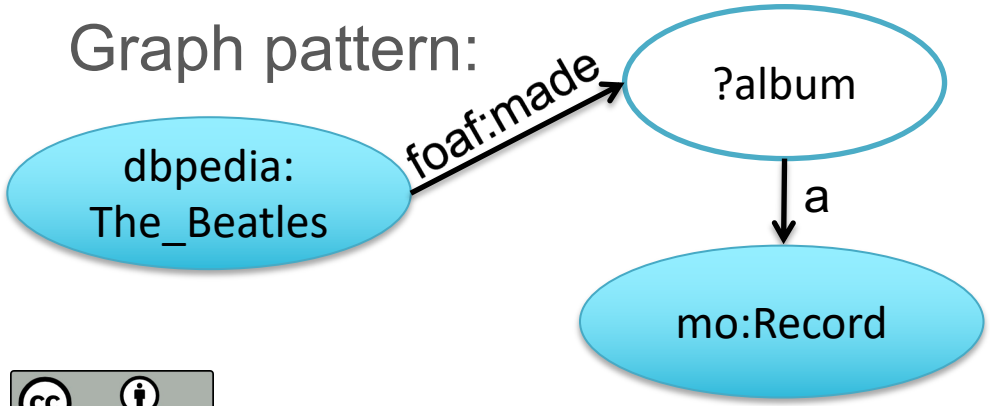
?album

SPARQL Query



SPARQL Query





Results:

?album
<http://musicbrainz.org...>

Graphs vs Graph Patterns

- **Con** = countably infinite set of constants
- **Var** = countably infinite set of variables
- **Term** = **Con**∪ **Var**
- **Con** ∩ **Var** = {}

All definitions* here from KGBook: https://kgbook.org/#ssec-querying

* Some definition has been recently updated, e.g., Directed edge-labeled graph; I keep the previous simpler definitions as they cover properly what we need

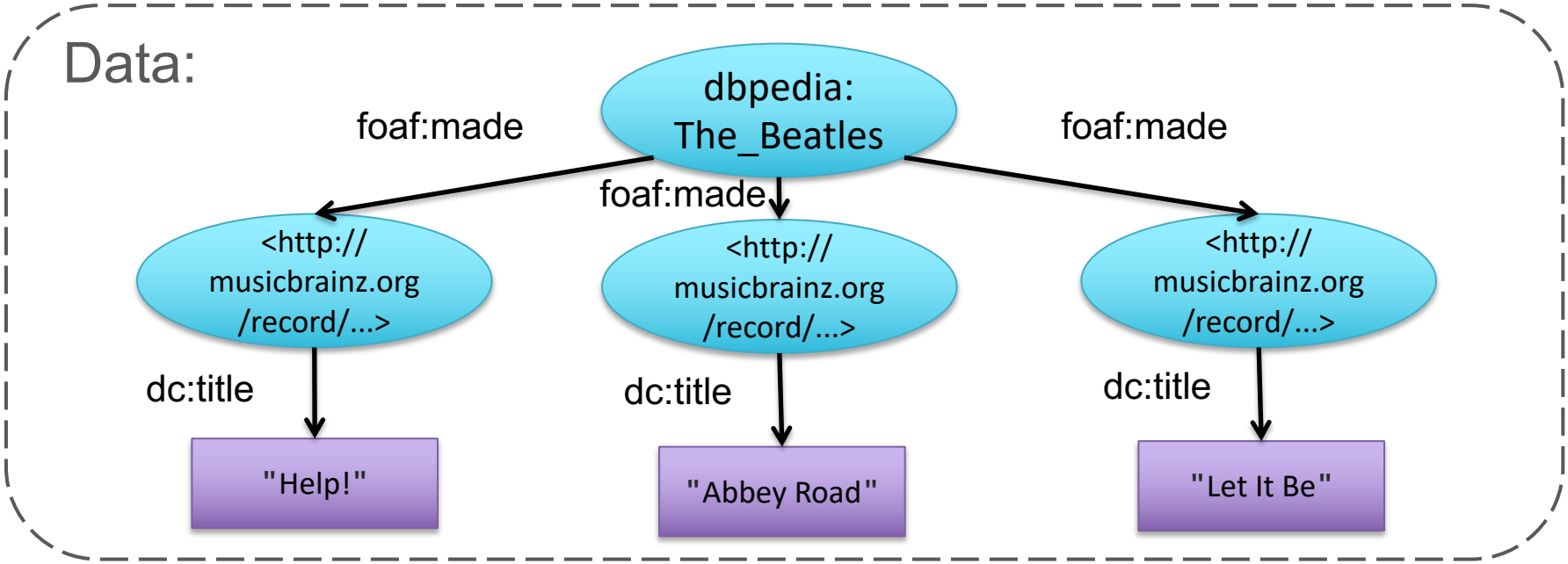
Definition 2.1 (Directed edge-labelled graph)

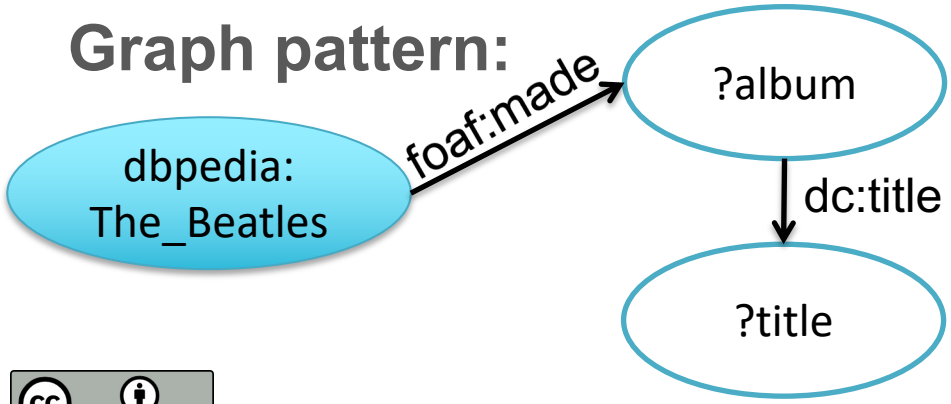
A directed edge-labelled graph is a tuple $G = (V, E, L)$*, where* $V \subseteq \mathbf{Con}$ *is a set of nodes,* $L \subseteq \mathbf{Con}$ *is a set of edge labels, and* $E \subseteq V \times L \times V$ *is a set of edges.*

Definition 2.5 (Basic directed edge-labelled graph pattern)

We define a basic directed edge-labelled graph pattern as a tuple $Q = (V, E, L)$*, where* $V \subseteq \mathbf{Term}$ *is a set of node terms,* $L \subseteq \mathbf{Term}$ *is a set of edge terms, and* $E \subseteq V \times L \times V$ *is a set of edges (triple patterns).*

SPARQL Query

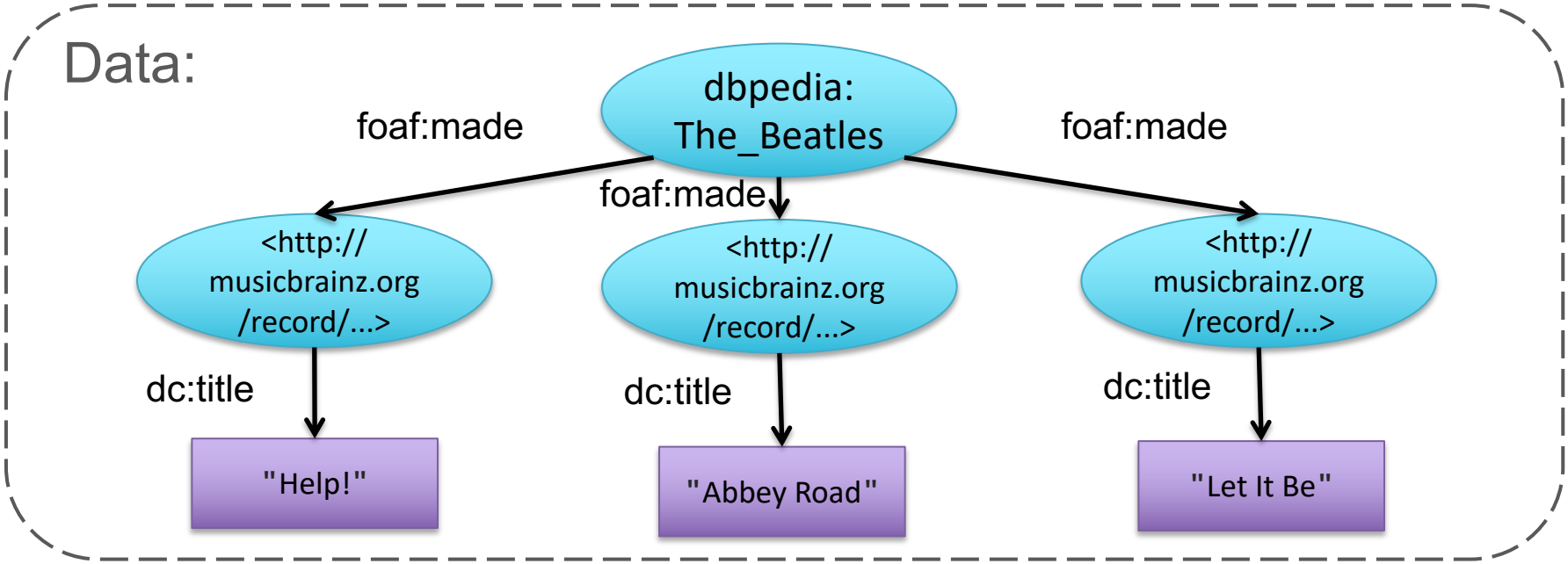




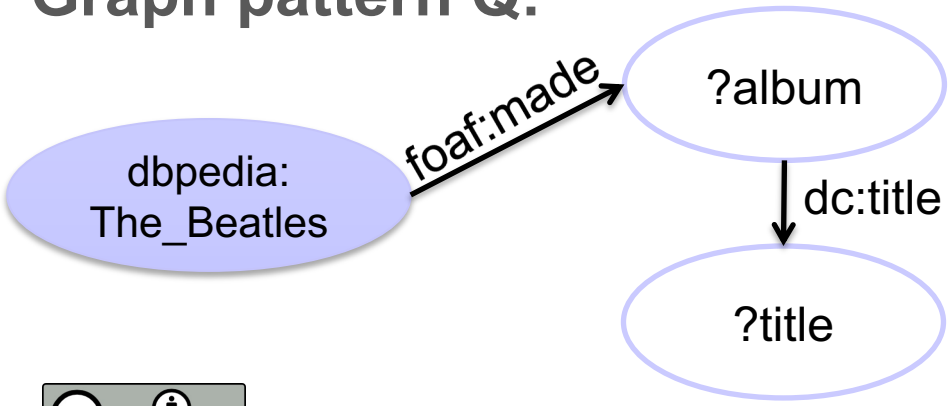
Results:

?album	?title
<http://...>	"Help!"
<http://...>	"Abbey Road"
<http://...>	"Let It Be"

SPARQL Query



Graph pattern Q:



Var(Q)= {?album,?title}
Term(Q) = {?album,?title, dbpedia: The_Beatles, foaf:made, dc:title}

Mappings and Evaluations (Basic Graph Patterns)

Towards defining the results of evaluating a basic graph pattern over a data graph (following the same model), we first define a partial mapping $\mu : \mathbf{Var} \to \mathbf{Con}$ from variables to constants, whose *domain* (the set of variables for which it is defined) is denoted by $\mathbf{dom}(\mu)$. Given a basic graph pattern Q, let $\mathbf{Var}(Q)$ denote the set of all variables appearing in (some recursively nested element of) Q. We further denote by $\mu(Q)$ the image of Q under μ, meaning that any variable $v \in \mathbf{Var}(Q) \cap \mathbf{dom}(\mu)$ is replaced in Q by $\mu(v)$. Observe that when $\mathbf{Var}(Q) \subseteq \mathbf{dom}(\mu)$, then $\mu(Q)$ is a data graph (in the corresponding model of Q).

Next, we define the notion of containment between data graphs. For two directed edge-labelled graphs $G_1 = (V_1, E_1, L_1)$ and $G_2 = (V_2, E_2, L_2)$, we say that G_1 is a *sub-graph* of G_2, denoted $G_1 \subseteq G_2$, if and only if $V_1 \subseteq V_2$, $E_1 \subseteq E_2$, and $L_1 \subseteq L_2$.[3]

Definition 2.7 (Evaluation of a basic graph pattern)

Let Q be a basic graph pattern and let G be a data graph (in the same model). We then define the evaluation of the basic graph pattern Q over the data graph G, denoted $Q(G)$, to be the set of mappings $Q(G) = \{\mu \mid \mu(Q) \subseteq G \text{ and } \mathbf{dom}(\mu) = \mathbf{Var}(Q)\}$.

Mappings and Evaluations (Basic Graph Patterns)

Towards defining the results of evaluating a basic graph pattern over a data graph (following the same model), we first define a partial mapping $\mu : \mathbf{Var} \rightarrow \mathbf{Con}$ from variables to constants, whose *domain* (the set [illegible] by $\mathbf{dom}(\mu)$. Given a basic graph pattern Q, let [illegible] some recursively nested element of) Q. We [illegible] aning that any variable $v \in \mathbf{Var}(Q) \cap \mathbf{dom}($[illegible] $\mathbf{Var}(Q) \subseteq \mathbf{dom}(\mu)$, then $\mu(Q)$ is a data graph[illegible]

In practice:
- Compute **substitutions** of variables: mappings from variables in Q to constants in G
- Check which substitions are the results: substitutions that make the graph pattern a subset of G

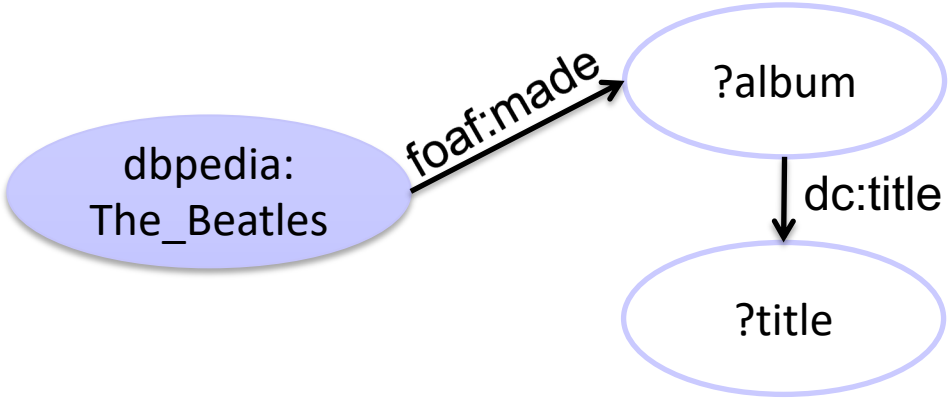
Next, we define the [illegible] directed edge-labelled graphs $G_1 = (V_1, E_1, L_1)$ and $G_2 = (V_2, E_2, L_2)$, we say that G_1 is a *sub-graph* of G_2, denoted $G_1 \subseteq G_2$, if and only if $V_1 \subseteq V_2$, $E_1 \subseteq E_2$, and $L_1 \subseteq L_2$.[3]

Definition 2.7 (Evaluation of a basic graph pattern)

Let Q be a basic graph pattern and let G be a data graph (in the same model). We then define the evaluation of the basic graph pattern Q over the data graph G, denoted $Q(G)$, to be the set of mappings $Q(G) = \{\mu \mid \mu(Q) \subseteq G \text{ and } \mathbf{dom}(\mu) = \mathbf{Var}(Q)\}$.

Mappings and Evaluations (Basic Graph Patterns)

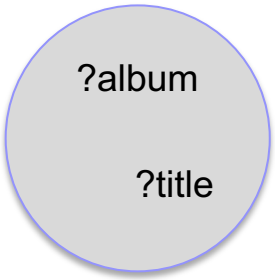
Graph pattern Q



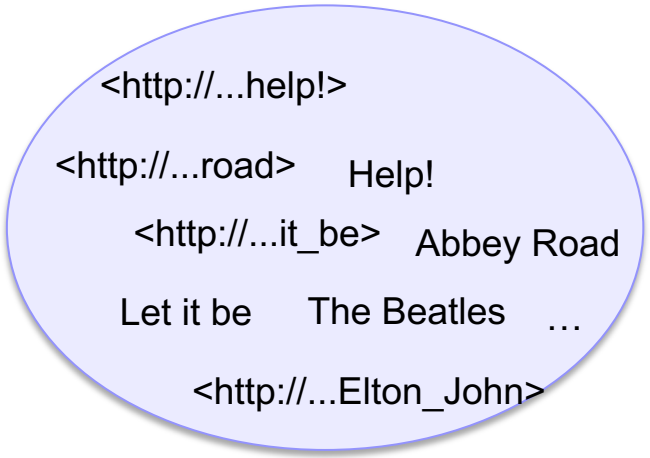
Var(Q)= {?album,?title}
Term(Q) = {?album,?title, dbpedia: The_Beatles, foaf:made, dc:title}

Mappings (functions)

Var(Q)

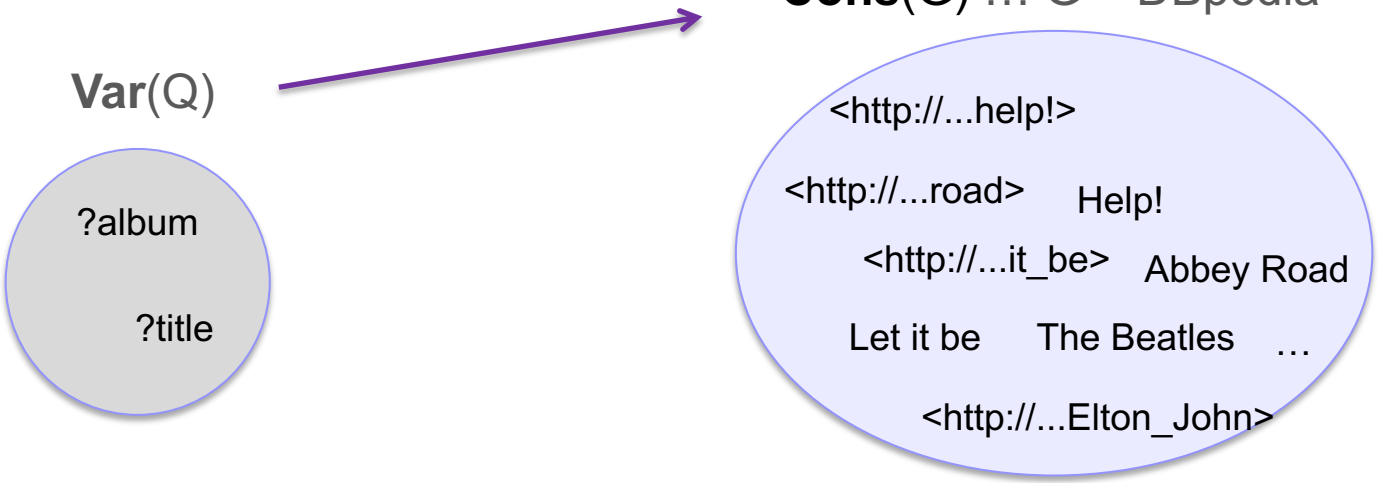


Cons(G) … G = DBpedia



Mappings and Evaluations (Basic Graph Patterns)

Mappings (functions)



All the possible mappings for Q:

μ_1

?album → <http://...help!>

?title → Help!

μ_2

?album → <http://...Elton_John>

?title

μ_3

?album → <http://...help!>

?title → Let it be

μ_4

?album → <http://...it_be>

?title → Let it be

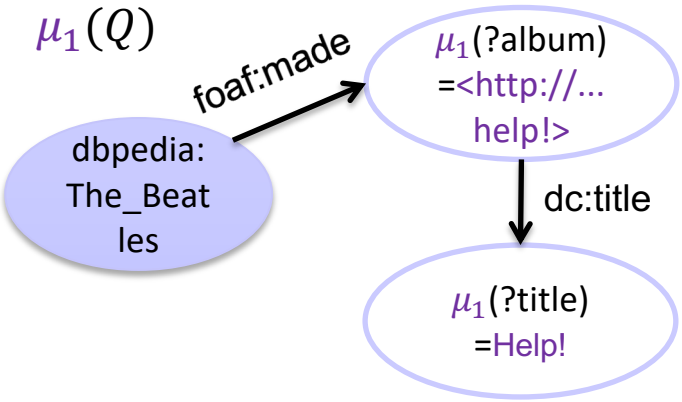
$\mu_{...}$

?album → …

?title → …

Mappings and Evaluations (Basic Graph Patterns)

Replacement



All the possible mappings for Q:

μ_1

?album → <http://...help!>

?title → Help!

μ_2

?album → <http://...Elton_John>

?title

μ_3

?album → <http://...help!>

?title → Let it be

μ_4

?album → <http://...it_be>

?title → Let it be

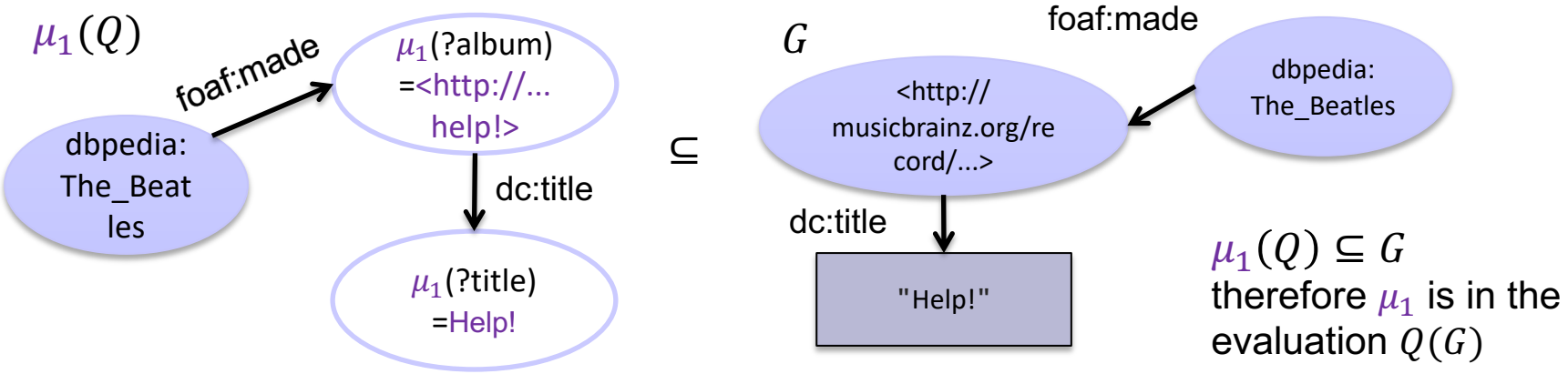
$\mu_{...}$

?album → …

?title → …

Mappings and Evaluations (Basic Graph Patterns)

Replacement and evaluation



$\mu_1(Q) \subseteq G$
therefore μ_1 is in the evaluation $Q(G)$

All the possible mappings for Q:

μ_1
?album → <http://...help!>
?title → Help!

μ_2
?album → <http://...Elton_John>
?title

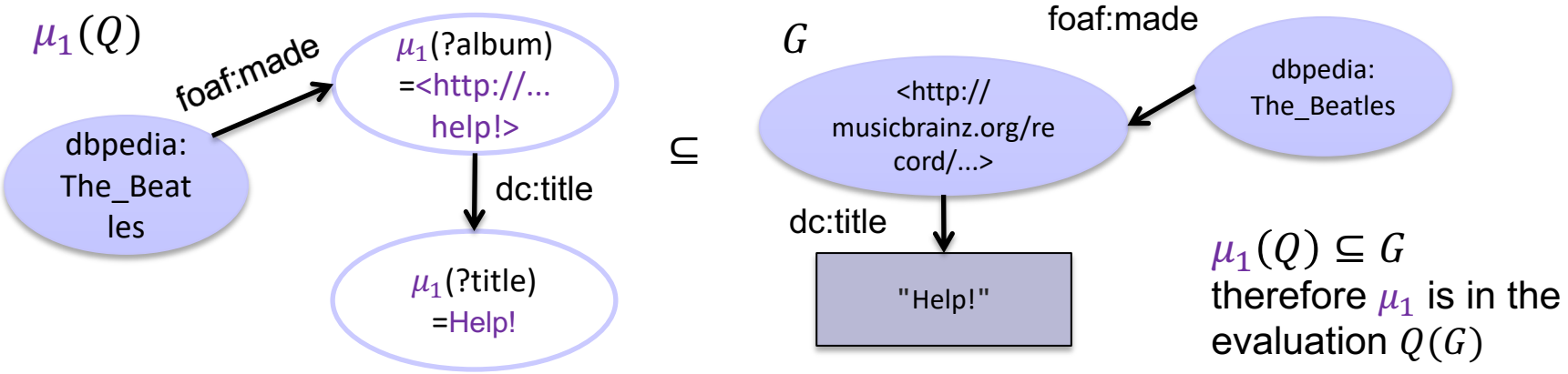
μ_3
?album → <http://...help!>
?title → Let it be

μ_4
?album → <http://...it_be>
?title → Let it be

$\mu_{...}$
?album → …
?title → …

Mappings and Evaluations (Basic Graph Patterns)

Replacement and evaluation



$\mu_1(Q) \subseteq G$
therefore μ_1 is in the evaluation $Q(G)$

μ_1

?album → <http://...help!>

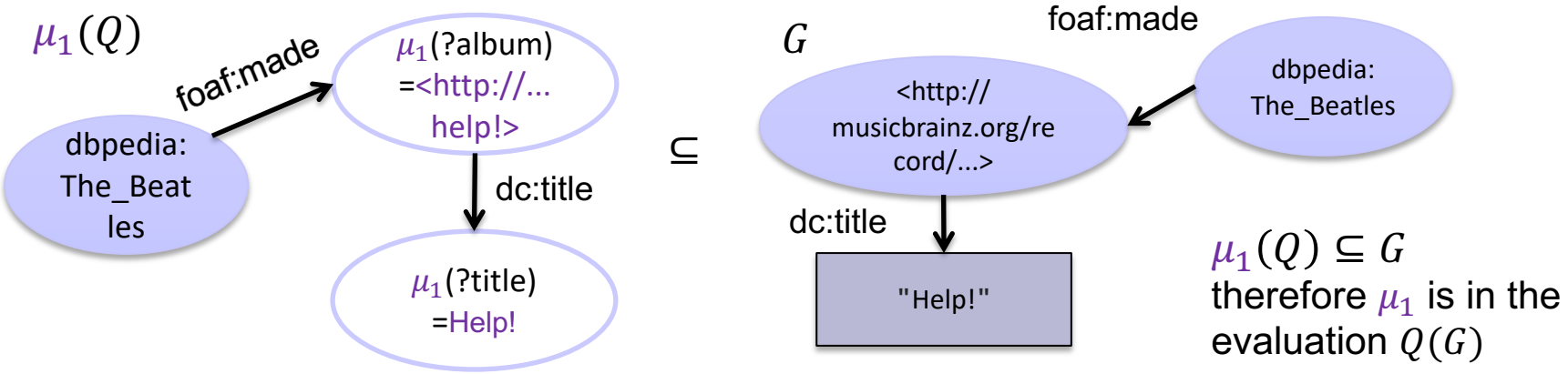
?title → Help!

Mappings as rows in a table

	?album	?title
$\mu_1(Q) \subseteq G$	<http://...>	"Help!"
$\mu_i(Q) \subseteq G$	<http://...>	"Abbey Road"
$\mu_j(Q) \subseteq G$	<http://...>	"Let It Be"

Mappings and Evaluations (Basic Graph Patterns)

Replacement and evaluation



$\mu_1(Q) \subseteq G$
therefore μ_1 is in the evaluation $Q(G)$

Mappings as **rows** in a table

An evaluation returns a **table** (!)

$\mu_1(Q) \subseteq G$
$\mu_i(Q) \subseteq G$
$\mu_j(Q) \subseteq G$

?album	?title
<http://...>	"Help!"
<http://...>	"Abbey Road"
<http://...>	"Let It Be"

From Basic to Complex Graph Patterns

A (basic) graph pattern transforms an input graph into a table of results (as shown in Figure 2.5). We may then consider using the relational algebra to combine and/or transform such tables, thus forming more complex queries from one or more graph patterns. Recall that the relational algebra consists of unary operators that accept one input table, and binary operators that accept two input tables. Unary operators include projection (π) to output a subset of columns, selection (σ) to output a subset of rows matching a given condition, and renaming of columns (ρ). Binary operators include union ($\cup$) to merge the rows of two tables into one table, difference ($-$) to remove the rows from the first table present in the second table, and joins ($\bowtie$) to extend the rows of one table with rows from the other table that satisfy a join condition. Selection and join conditions typically include equalities ($=$), inequalities ($\leq$), negation ($\neg$), disjunction ($\vee$), etc. From these operators, we can further define other (syntactic) operators, such as intersection ($\cap$) to output rows in both tables, anti-join ($\triangleright$, aka *minus*) to output rows from the first table for which there are no join-compatible rows in the second table, left-join ($\bowtie$, aka *optional*) to perform a join but keeping rows from the first table without a compatible row in the second table, etc.

Complex Graph Patterns

Definition 2.8 (Complex graph pattern)

Complex graph patterns are defined recursively, as follows:

- *If Q is a basic graph pattern, then Q is a complex graph pattern.*
- *If Q is a complex graph pattern, and $\mathcal{V} \subseteq \mathbf{Var}(Q)$, then $\pi_{\mathcal{V}}(Q)$ is a complex graph pattern.*
- *If Q is a complex graph pattern, and R is a selection condition with Boolean and equality connectives ($\wedge$, $\vee$, $\neg$, $=$) , then $\sigma_R(Q)$ is a complex graph pattern.*
- *If both Q_1 and Q_2 are complex graph patterns, then $Q_1 \bowtie Q_2$, $Q_1 \cup Q_2$, $Q_1 - Q_2$ and $Q_1 \triangleright Q_2$ are also complex graph patterns.*

Definition 2.9 (Complex graph pattern evaluation)

Given a complex graph pattern Q, if Q is a basic graph pattern, then $Q(G)$ is defined per Definition 2.7. Otherwise, $Q(G)$ is defined as follows:

$$\pi_{\mathcal{V}}(Q)(G) = \{\mu[\mathcal{V}] \mid \mu \in Q(G)\}$$
$$\sigma_R(Q)(G) = \{\mu \mid \mu \in Q(G) \text{ and } \mu \models R\}$$
$$Q_1 \bowtie Q_2(G) = \{\mu_1 \cup \mu_2 \mid \mu_1 \in Q_2(G), \mu_2 \in Q_1(G) \text{ and } \mu_1 \sim \mu_2\}$$
$$Q_1 \cup Q_2(G) = \{\mu \mid \mu \in Q_1(G) \text{ or } \mu \in Q_2(G)\}$$
$$Q_1 - Q_2(G) = \{\mu \mid \mu \in Q_1(G) \text{ and } \mu \notin Q_2(G)\}$$
$$Q_1 \triangleright Q_2(G) = \{\mu \mid \mu \in Q_1(G) \text{ and } \nexists \mu_2 \in Q_2(G) \text{ such that } \mu \sim \mu_2\}$$

Projection: keep only a subset of columns

Selection: keep only a subset of rows that satisfy some conditions

Joins, unions, …

The mapping satisfy the condition

Mappings are compatible, i.e., they map common variables to the same constants

Path Queries #1

- Check: https://www.w3.org/TR/sparql11-query/#propertypaths

9.2 Examples

Variables cannot be used as part of the path itself
Vertexes are returned when the paths are evaluated

Alternatives: Match one or both possibilities

```
{ :book1 dc:title|rdfs:label ?displayString }
```

which could have writen:

```
{ :book1 <http://purl.org/dc/elements/1.1/title> | <http://www.w3.org/2000/01/rdf-schema#label> ?displayString }
```

Sequence: Find the name of any people that Alice knows.

```
{
  ?x foaf:mbox <mailto:alice@example> .
  ?x foaf:knows/foaf:name ?name .
}
```

Sequence: Find the names of people 2 "`foaf:knows`" links away.

```
{
  ?x foaf:mbox <mailto:alice@example> .
  ?x foaf:knows/foaf:knows/foaf:name ?name .
}
```

Path Queries #2

Variables cannot be used as part of the path itself
Vertexes are returned when the paths are evaluated

Inverse Property Paths: These two are the same query: the second is just reversing the property direction which swaps the roles of subject and object.

```
{ ?x foaf:mbox <mailto:alice@example> }
```

```
{ <mailto:alice@example> ^foaf:mbox ?x }
```

Inverse Path Sequence: Find all the people who know someone `?x` knows.

```
{
  ?x foaf:knows/^foaf:knows ?y .
  FILTER(?x != ?y)
}
```

Arbitrary length match: Find the names of all the people that can be reached from Alice by `foaf:knows`:

```
{
  ?x foaf:mbox <mailto:alice@example> .
  ?x foaf:knows+/foaf:name ?name .
}
```

Paths & Evaluation

Definition 2.10 (Path expression)

A constant (edge label) c is a path expression. Furthermore, if r, r_1 and r_2 are path expressions, then:

- *r^- (inverse) and r^* (Kleene star) are path expressions.*
- *$r_1 \cdot r_2$ (concatenation) and $r_1 \mid r_2$ (disjunction) are path expressions.*

r* = zero or more paths
Why "zero or": next slide!

We now define the evaluation of a path expression on a directed-edge labelled graph under the SPARQL 1.1-style semantics whereby the endpoints (pairs of start and end nodes) of the path are returned [Harris et al., 2013].

Definition 2.11 (Path evaluation (directed edge-labelled graph))

Given a directed edge-labelled graph $G = (V, E, L)$ and a path expression r, we define the evaluation of r over G, denoted $r[G]$, as follows:

$$
\begin{aligned}
r[G] &= \{(u,v) \mid (u,r,v) \in E\} \text{ (for } r \in \mathbf{Con}) \\
r^-[G] &= \{(u,v) \mid (v,u) \in r[G]\} \\
r_1 \mid r_2[G] &= r_1[G] \cup r_2[G] \\
r_1 \cdot r_2[G] &= \{(u,v) \mid \exists w \in V : (u,w) \in r_1[G] \text{ and } (w,v) \in r_2[G]\} \\
r^*[G] &= V \cup \bigcup_{n \in \mathbb{N}^+} r^n[G]
\end{aligned}
$$

{(u,u) | u ∈ V} U […]

where by r^n we denote the n^{th}-concatenation of r (e.g., $r^3 = r \cdot r \cdot r$).

Path Queries & Evaluation

Definition 2.12 (Regular path query)

A regular path query is a triple (x, r, y) *where* $x, y \in \mathbf{Con} \cup \mathbf{Var}$ *and* r *is a path expression.*

Definition 2.13 (Regular path query evaluation)

Let G *denote a directed edge-labelled graph,* c, c_1, $c_2 \in \mathbf{Con}$ *denote constants and* z, z_1, $z_2 \in \mathbf{Var}$ *denote variables. Then the evaluation of a regular path query is defined as follows:*

$$
\begin{aligned}
(c_1, r, c_2)(G) &= \{\mu_\emptyset \mid (c_1, c_2) \in r[G]\} \\
(c, r, z)(G) &= \{\mu \mid \mathbf{dom}(\mu) = \{z\} \text{ and } (c, \mu(z)) \in r[G]\} \\
(z, r, c)(G) &= \{\mu \mid \mathbf{dom}(\mu) = \{z\} \text{ and } (\mu(z), c) \in r[G]\} \\
(z_1, r, z_2)(G) &= \{\mu \mid \mathbf{dom}(\mu) = \{z_1, z_2\} \text{ and } (\mu(z_1), \mu(z_2)) \in r[G]\}
\end{aligned}
$$

where $\mu_\emptyset$ *denotes the empty mapping such that* $\mathbf{dom}(\mu) = \emptyset$ *(the join identity).*

Definition 2.14 (Navigational graph pattern)

If Q *is a basic graph pattern, then* Q *is a navigational graph pattern. If* Q *is a navigational graph pattern and* (x, r, y) *is a regular path query, then* $Q \bowtie (x, r, y)$ *is a navigational graph pattern.*

Paths & Evaluation

r* = zero or more paths
Why "**zero** or more" instead of "one or more"

it does seem a bit silly to return these (*u*,*u*) pairs, but in a query like:

```
?s rdf:type/rdfs:subClassOf* ex:Building .
```

it's quite nice as it returns those things that are directly instances of buildings, because we will have (`ex:Building`,`ex:Building`) in the evaluation of `rdfs:subClassOf*`.

So often the zero case is already "bound" to something that is copied to the output, rather than returning (*u*,*u*) for all u in V. But in the base case we define there, it's not bound to anything yet.

Definition 2.11 (Path evaluation (directed edge-labelled graph))

Given a directed edge-labelled graph $G = (V, E, L)$ and a path expression r, we define the evaluation of r over G, denoted $r[G]$, as follows:

$$
\begin{aligned}
r[G] &= \{(u,v) \mid (u,r,v) \in E\} \text{ (for } r \in \mathbf{Con}) \\
r^{-}[G] &= \{(u,v) \mid (v,u) \in r[G]\} \\
r_1 \mid r_2[G] &= r_1[G] \cup r_2[G] \\
r_1 \cdot r_2[G] &= \{(u,v) \mid \exists w \in V : (u,w) \in r_1[G] \text{ and } (w,v) \in r_2[G]\} \\
r^{*}[G] &= \cancel{V} \cup \bigcup_{n \in \mathbb{N}^{+}} r^{n}[G]
\end{aligned}
$$

{(*u*,*u*) | *u* ∈ V} U […]

where by r^n we denote the n^{th}-concatenation of r (e.g., $r^3 = r \cdot r \cdot r$).

SPARQL Query: Components

EUCLID
EdUcational Curriculum for the usage of LInked Data

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX mo: <http://purl.org/ontology/mo/>

SELECT ?album
FROM <http://musicbrainz.org/20130302>
WHERE {
    dbpedia:The_Beatles foaf:made ?album .
    ?album a mo:Record ; dc:title ?title
}
ORDER BY ?title
```

Prologue:

- Prefix definitions
- Subtly different from Turtle syntax - the final period is not used

SPARQL Query: Components

EUCLID
EdUcational Curriculum for the usage of LInked Data

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX mo: <http://purl.org/ontology/mo/>

SELECT ?album
FROM <http://musicbrainz.org/20130302>
WHERE {
    dbpedia:The_Beatles foaf:made ?album .
    ?album a mo:Record ; dc:title ?title
}
ORDER BY ?title
```

Query form:

- ASK, SELECT, DESCRIBE or CONSTRUCT
- SELECT retrieves variables and their bindings as a table

SPARQL Query: Components

EUCLID
EdUcational Curriculum for the usage of LInked Data

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX mo: <http://purl.org/ontology/mo/>

SELECT ?album
FROM <http://musicbrainz.org/20130302>
WHERE {
    dbpedia:The_Beatles foaf:made ?album .
    ?album a mo:Record ; dc:title ?title
}
ORDER BY ?title
```

Data set specification:

- This clause is optional
- FROM or FROM NAMED
- Indicates the sources for the data against which to find matches

SPARQL Query: Components

EUCLID
EdUcational Curriculum for the usage of LInked Data

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX mo: <http://purl.org/ontology/mo/>

SELECT ?album
FROM <http://musicbrainz.org/20130302>
WHERE {
```

Graph names

Premise: instead of having one large graph, we can have many smaller graphs and exploit natural joins over RDF data - via URIs; how to name graphs:

- **Name of the graph**, i.e., the id specified in the beginning of the RDF file:
  - http://musicbrainz.org/20130302
- **N-Quads format** (https://www.w3.org/TR/n-quads/) instead of triples
  - `<subject> <predicate> <object> <b><context></b> .`
  - `<subject> <predicate> <object> <b><graphname></b> .`
    - We use the <context> field to specify the graph name

SPARQL Query: Components

EUCLID
EdUcational Curriculum for the usage of LInked Data

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX mo: <http://purl.org/ontology/mo/>

SELECT ?album
FROM <http://musicbrainz.org/20130302>
WHERE {
    dbpedia:The_Beatles foaf:made ?album .
    ?album a mo:Record ; dc:title ?title
}
ORDER BY ?title
```

Data set specification:

- This clause is optional
- FROM or FROM NAMED
- Indicates the sources for the data against which to find matches

SPARQL Query: Components

EUCLID
EdUcational Curriculum for the usage of LInked Data

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX mo: <http://purl.org/ontology/mo/>

SELECT ?album
FROM <http://musicbrainz.org/20130302>
WHERE {
    dbpedia:The_Beatles foaf:made ?album .
    ?album a mo:Record ; dc:title ?title
}
ORDER BY ?title
```

Query pattern:

- Defines patterns to match against the data
- Generalises Turtle with variables and keywords – N.B. final period optional

Turtle abbreviations

- Same subject:
  - `<[…]> a mo:Record ; dc:title “Help!” .` … equivalent to:
    - `<[…]> a mo:Record .`
    - `<[…]> dc:title “Help” .`
- Same subject and predicate:
  - `<[…]> dc:title “Help!@En” , “Aiuto!@It” .` … equivalent to:
    - `<[…]> dc:title “Help!@En” .`
    - `<[…]> dc:title “Aiuto!@It” .`

```
WHERE {
    dbpedia:The_Beatles foaf:made ?album .
    ?album a mo:Record ; dc:title ?title
}
ORDER BY ?title
```

Query pattern:

- Defines patterns to match against the data
- Generalises Turtle with variables and keywords – N.B. final period optional

SPARQL Query: Components

EUCLID
EdUcational Curriculum for the usage of LInked Data

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX mo: <http://purl.org/ontology/mo/>

SELECT ?album
FROM <http://musicbrainz.org/20130302>
WHERE {
    dbpedia:The_Beatles foaf:made ?album .
    ?album a mo:Record ; dc:title ?title
}
ORDER BY ?title
```

Query pattern:

- Defines patterns to match against the data
- Generalises Turtle with variables and keywords – N.B. final period optional

SPARQL Query: Components

EUCLID
EdUcational Curriculum for the usage of LInked Data

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX mo: <http://purl.org/ontology/mo/>

SELECT ?album
FROM <http://musicbrainz.org/20130302>
WHERE {
    dbpedia:The_Beatles foaf:made ?album .
    ?album a mo:Record ; dc:title ?title
}
ORDER BY ?title
```

Check SPARQL specifications when you need to use these advanced features. Once you understood WHERE clauses they are quite straightforward

Solution modifier:

- Modify the result set
- `ORDER BY`, `LIMIT` or `OFFSET` re-organise rows;
- `GROUP BY` combines them

SPARQL

Query RDF data

03 –SPARQL Query Forms

Query Forms

EUCLID
EdUcational Curriculum for the usage of LInked Data

SPARQL supports different query forms:

- **ASK** tests whether or not a query pattern has a solution. Returns yes/no
- **SELECT** returns variables and their bindings directly
- **CONSTRUCT** returns a single RDF graph specified by a graph template
- **DESCRIBE** returns a single RDF graph containing RDF data about resource

Query Form: ASK

- Namespaces are added with the 'PREFIX' directive
- Statement patterns that make up the graph are specified between brackets ("{}")

Query: *Is Paul McCartney member of 'The Beatles'?*

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpedia-ont: <http://dbpedia.org/ontology/>
PREFIX mo: http://purl.org/ontology/mo/
ASK WHERE { dbpedia:The_Beatles mo:member
                        dbpedia:Paul_McCartney.}
```

Results:

true

Query: *Is Elvis Presley member of 'The Beatles'?*

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpedia-ont: <http://dbpedia.org/ontology/>
PREFIX mo: http://purl.org/ontology/mo/
ASK WHERE { dbpedia:The_Beatles mo:member
                        dbpedia:Elvis_Presley.}
```

Results:

false

Query Form: SELECT

EUCLID
EdUcational Curriculum for the usage of LInked Data

- The solution modifier **projection** nominates which components of the matches should be returned
- "*" means all components should be returned

Query: *What albums and tracks did 'The Beatles' make?*

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX mo: <http://purl.org/ontology/mo/>
SELECT ?album_name ?track_title
WHERE {
    dbpedia:The_Beatles foaf:made ?album .
    ?album dc:title ?album_name ;
           mo:track ?track .
    ?track dc:title ?track_title .}
```

Query Form: SELECT (2)

EUCLID
EdUcational Curriculum for the usage of LInked Data

Filter expressions

- Different types of filters and functions may be used

Filter: Comparison and logical operators

Query: *Retrieve the albums and tracks recorded by 'The Beatles', where the duration of the song is more than 300 secs. and no longer than 400 secs.*

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX mo: <http://purl.org/ontology/mo/>
SELECT ?album_name ?track_title ?duration
WHERE {
    dbpedia:The_Beatles foaf:made ?album .
    ?album dc:title ?album_name ;
           mo:track ?track .
    ?track dc:title ?track_title ;
               mo:duration ?duration;
    FILTER (?duration>300000 && ?duration<400000) }
```

Query Form: CONSTRUCT

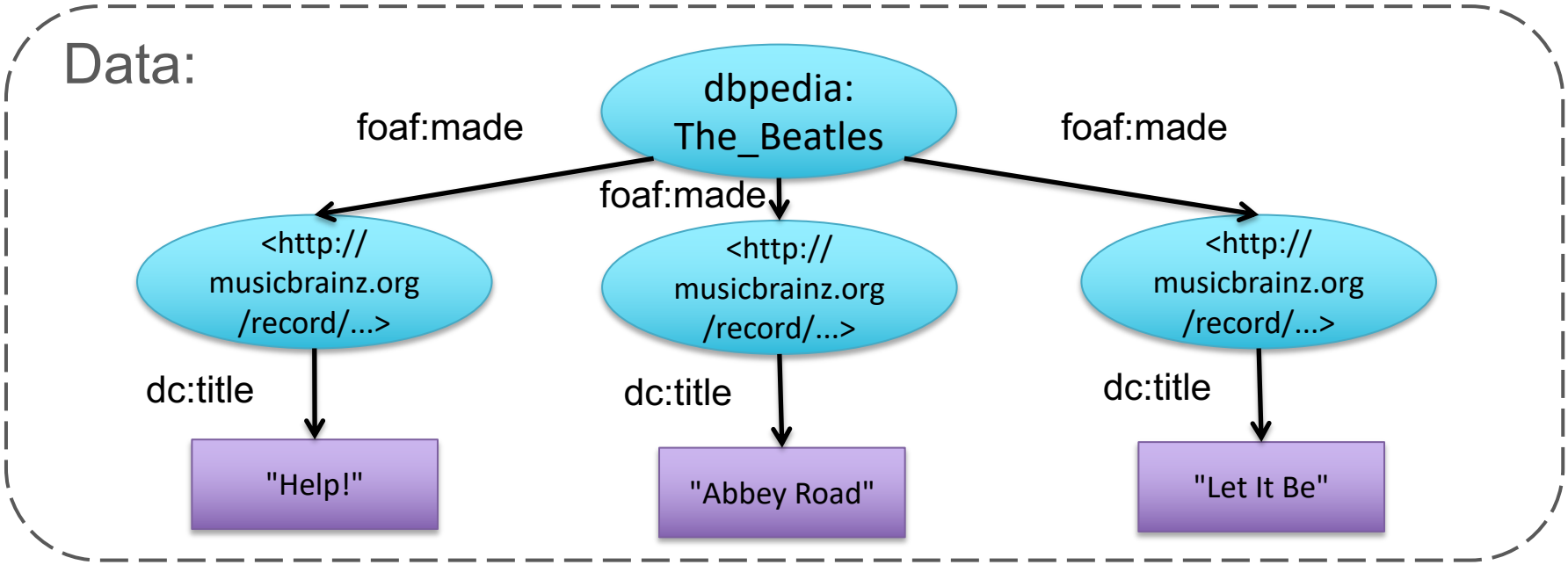
EUCLID
EdUcational Curriculum for the usage of LInked Data

- **CONSTRUCT WHERE:** In order to query for a subgraph, without change, it is no longer necessary to repeat the graph pattern in the template

Example:

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
CONSTRUCT WHERE {
  dbpedia:The_Beatles foaf:made ?album .
  ?album mo:track ?track .}
```

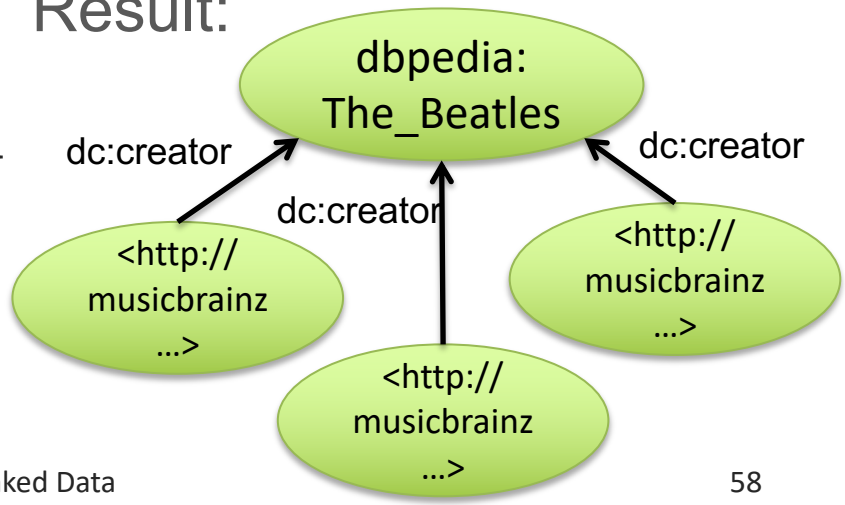
Query Form: CONSTRUCT (2a)



Query:

```
CONSTRUCT {
  ?album dc:creator dbpedia:The_Beatles .}
WHERE {
  dbpedia:The_Beatles foaf:made ?album .}
```

Result:



SPARQL Query Formulation Exercises

- You can query DBpedia through a web interface at: http://dbpedia.org/sparql
- How to Know What to Query?
  - After this lesson:
    - Understand the vocabulary (properties and classes) by exploring DBpedia online (see previous slide) – [after this lesson]
    - Use explorative queries (e.g., search for all properties used with subjects of type dbo:City)
  - After the lessons on ontologies:
    - Look at DBpedia ontology using Protegè

Query
and/or
Reasoning

04 –SPARQL & Reasoning

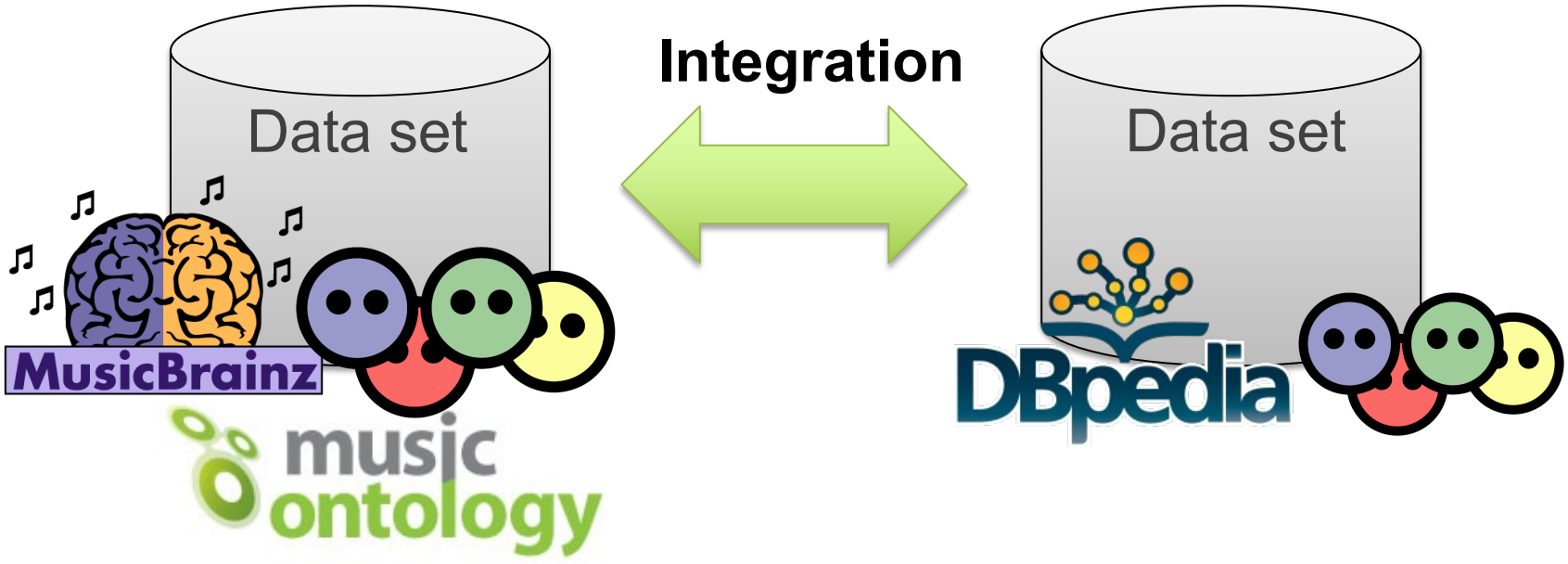
SPARQL Evolution

- It used to be only a query language
- We will discuss inference in the next slides; here, just a simple example:
  - owl:sameAs: predicate to state that two entities are the same

Reasoning for Linked Data Integration

EUCLID
EdUcational Curriculum for the usage of LInked Data

- **Example:** Integration of the **MusicBrainz** data set and the **DBpedia** data set

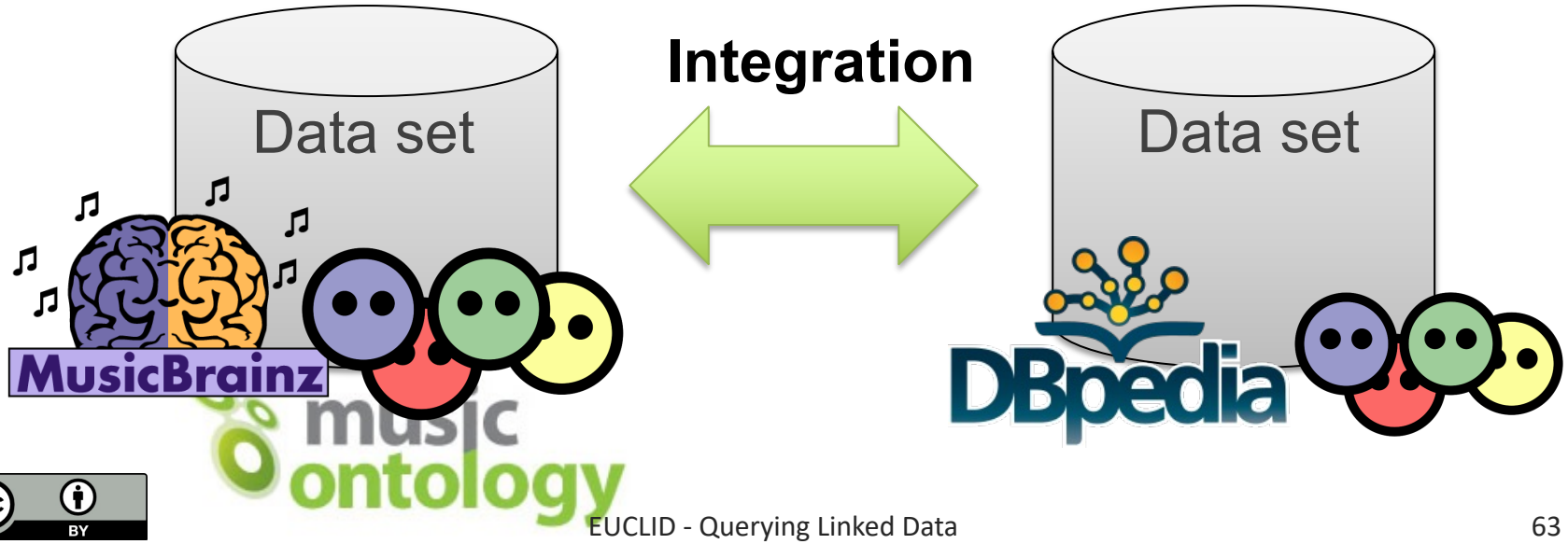


Reasoning for Linked Data Integration

EUCLID
EdUcational Curriculum for the usage of LInked Data

same

```
mo:b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d ⟺ dbpedia:The_Beatles
    foaf:name The Beatles;                    dbpedia-ont:origin dbpedia:Liverpool;
    mo:member                                 dbpedia-ont:genre dbpedia:Rock_music;
     mo:ba550d0e-adac-4864-b88b-407cab5e76af; foaf:depiction <image> .
    mo:member
     mo:4d5447d7-c61c-4120-ba1b-d7f471d385b9;
    mo:member
     mo:42a8f507-8412-4611-854f-926571049fa0;
    mo:member
     mo:300c4c73-33ac-4255-9d57-4e32627f5e13.
```



SPARQL Evolution

- It used to be only a query language
- We will discuss inference in the next slides; here, just a simple example:
  - owl:sameAs: predicate to state that two entities are the same
    - `<mo:b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d, owl:sameAs, dbpedia:The_Beatles>` *interpreted as*
    - `mo:b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d, = dbpedia:The_Beatles`
    - ... intuitively, everything we state for the first entity is also true for the second entity ... or, better ... everything we state for the first entity should also be **inferred** to be true for the second entity

Reasoning for Linked Data Integration

EUCLID
EdUcational Curriculum for the usage of LInked Data

owl:sameAs

```
mo:b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d
    foaf:name The Beatles;
    mo:member
     mo:ba550d0e-adac-4864-b88b-407cab5e76af;
    mo:member
     mo:4d5447d7-c61c-4120-ba1b-d7f471d385b9;
    mo:member
     mo:42a8f507-8412-4611-854f-926571049fa0;
    mo:member
     mo:300c4c73-33ac-4255-9d57-4e32627f5e13 .
```

⟺

```
dbpedia:The_Beatles
    dbpedia-ont:origin dbpedia:Liverpool;
    dbpedia-ont:genre dbpedia:Rock_music;
    foaf:depiction  .
```

Query:

```
SELECT ?m ?g
WHERE {
 dbpedia:The_Beatles
    mo:member ?m;
    dbpedia-ont:genre ?g.}
```

Expected result set (with inference)

?m	?g
mo:ba550d0e-adac-4864-b88b-407cab5e76af	dbpedia:Rock_music
mo:4d5447d7-c61c-4120-ba1b-d7f471d385b9	dbpedia:Rock_music
mo42a8f507-8412-4611-854f-926571049fa0;	dbpedia:Rock_music
mo300c4c73-33ac-4255-9d57-4e32627f5e13	dbpedia:Rock_music

Reasoning for Linked Data Integration

EUCLID
EdUcational Curriculum for the usage of LInked Data

owl:sameAs

```
mo:b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d
    foaf:name The Beatles;
    mo:member
     mo:ba550d0e-adac-4864-b88b-407cab5e76af;
    mo:member
     mo:4d5447d7-c61c-4120-ba1b-d7f471d385b9;
    mo:member
     mo:42a8f507-8412-4611-854f-926571049fa0;
    mo:member
     mo:300c4c73-33ac-4255-9d57-4e32627f5e13 .
```

⟺

```
dbpedia:The_Beatles
    dbpedia-ont:origin dbpedia:Liverpool;
    dbpedia-ont:genre dbpedia:Rock_music;
    foaf:depiction
```

.

Query:

```
SELECT ?m ?g
WHERE {
 dbpedia:The_Beatles
   mo:member ?m;
   dbpedia-ont:genre ?g.}
```

Returned result set (w/out inference)

?m	?g

No mo:member-labeled edge from dbpedia:The_Beatles

Reasoning for Linked Data Integration

EUCLID
EdUcational Curriculum for the usage of LInked Data

owl:sameAs

```
mo:b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d
    foaf:name The Beatles;
    mo:member
     mo:ba550d0e-adac-4864-b88b-407cab5e76af;
    mo:member
     mo:4d5447d7-c61c-4120-ba1b-d7f471d385b9;
    mo:member
     mo:42a8f507-8412-4611-854f-926571049fa0;
    mo:member
     mo:300c4c73-33ac-4255-9d57-4e32627f5e13 .
```

⟺

```
dbpedia:The_Beatles
    dbpedia-ont:origin dbpedia:Liverpool;
    dbpedia-ont:genre dbpedia:Rock_music;
    foaf:depiction  .
    owl:sameAs mo:b10bbbfc-cf9e-42e0[...]
```

Modified query:

```
SELECT ?m ?g
WHERE {
 dbpedia:The_Beatles
    owl:sameAs ?x;
    dbpedia-ont:genre ?g.
 ?x mo:member ?m.}
```

Returned result set (w/out inference)

?m	?g
mo:ba550d0e-adac-4864-b88b-407cab5e76af	dbpedia:Rock_music
mo:4d5447d7-c61c-4120-ba1b-d7f471d385b9	dbpedia:Rock_music
mo42a8f507-8412-4611-854f-926571049fa0;	dbpedia:Rock_music
mo300c4c73-33ac-4255-9d57-4e32627f5e13	dbpedia:Rock_music

Reasoning for Linked Data Integration

EUCLID
EdUcational Curriculum for the usage of LInked Data

owl:sameAs

```
mo:b10bbbfc-cf9e-42e0-be17-e2c3e1d2600d
    foaf:name The Beatles;
    mo:member
     mo:ba550d0e-adac-4864-b88b-407cab5e76af;
    mo:member
     mo:4d5447d7-c61c-4120-ba1b-d7f471d385b9;
    mo:member
     mo:42a8f507-8412-4611-854f-926571049fa0;
    mo:member
     mo:300c4c73-33ac-4255-9d57-4e32627f5e13 .
```

⟺

```
dbpedia:The_Beatles
    dbpedia-ont:origin dbpedia:Liverpool;
    dbpedia-ont:genre dbpedia:Rock_music;
    foaf:depiction  .
    owl:sameAs mo:b10bbbfc-cf9e-42e0[...]
```

Modified query:

```
SELECT ?m ?g
WHERE {
 dbpedia:The_Beatles
    owl:sameAs ?x;
    dbpedia-ont:genre ?g.
 ?x mo:member ?m.}
```

Returned result set (w/out inference)

?m	?g
mo:ba550d0e-adac-4864-b88b-407cab5e76af	dbpedia:Rock_music

Now we have a path from dbpedia:The_Beatles to its members through owl:sameAs and mo:member-labeled edges

ALERT: often it is not possible to modify the query to get what we'd like to infer (see next lessons)

SPARQL 1.1: Entailment Regimes

EUCLID
EdUcational Curriculum for the usage of LInked Data

- SPARQL 1.0 was defined only for simple entailment (pattern matching)

- SPARQL 1.1 is extended with entailment regimes other than simple entailment:
  - RDF entailment
  - **RDFS entailment**
  - D-Entailment
  - OWL RL entailment
  - OWL Full entailment
  - OWL 2 DL, EL, and QL entailment
  - RIF entailment

Entailment: ~ "implication", ~ inference
REMARK: the triple store serving the SPARQL endpoint must support the selected entailment regime (cont'd: different stores support different entailment regimes)

Source: http://www.w3.org/TR/rdf-mt/#RDFSRules