SPARQL

Query Language for RDF

Motivation

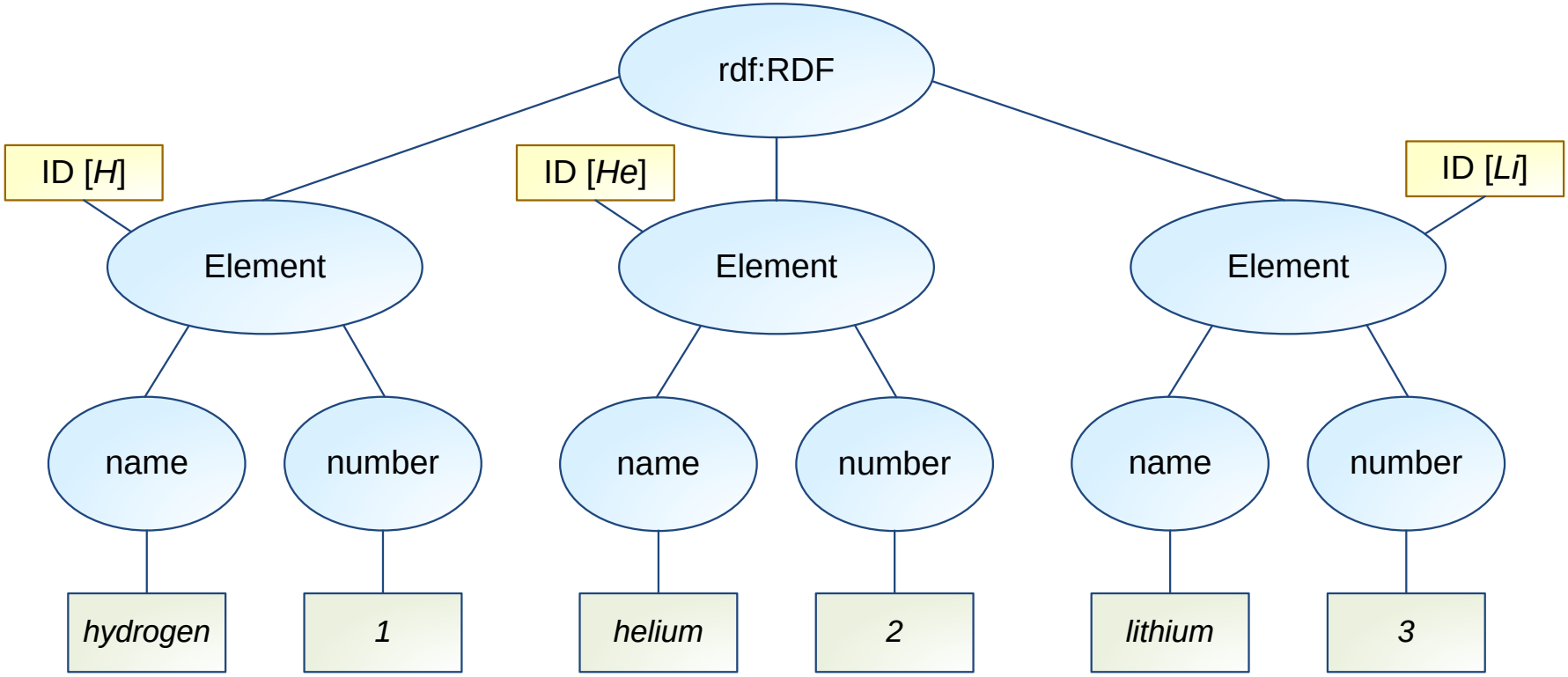
- RDF, RDF Schema, OWL provide data and meta-data
  - meta-data is important
  - graph data is important
  - inference is important
- How to process this data?
  - SQL, XQuery etc. not a good match to process graphs
  - OWL/Inference tackles different problem
  - Need for another language

SPARQL: A Query Language for RDF

- Name is a recursive acronym
  - SPARQL = SPARQL Protocol and RDF Query Language.
- Available as W3C Recommendation since 2008
- Query language for RDF instances
- Several aspects:
  - Query Language (discussed here)
  - Result Format: Representing Results as XML/…
  - Protocol: Transferring Queries and Results over the network
- SPARQL 1.0 currently stable, 1.1 under development
- Relatively simple/restricted language,
- 1.1 overcome some obvious limitations

Sample RDF Graph

The Periodic System of Elements



Sample SPARQL Query

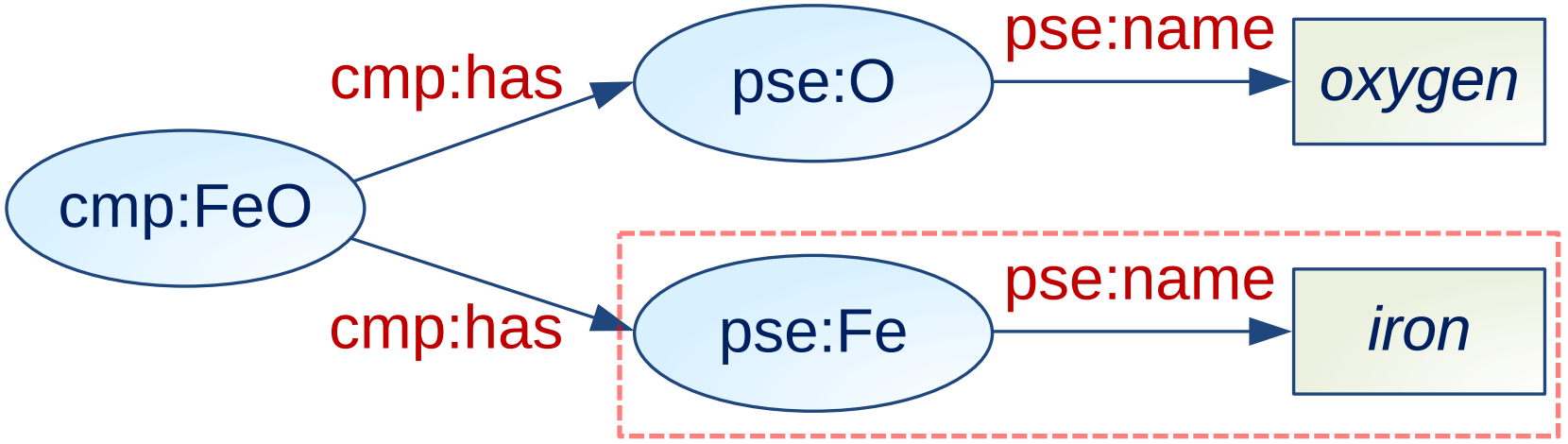
```
SELECT ?number
WHERE {
   ?element chemistry:name ”iron”.
   ?element chemistry:number ?number.
}
```

Give back the number of each element that has the name iron

N.B. This is a SELECT query – there are also other types

SPARQL: Matching triples (1)

RDF



SPARQL Basic Query (1)

Get the ID of all elements which have the name “iron”

SPARQL: Matching Triples (2)

- **Matching triples fundamental operation in SPARQL**
- **At each part, either provide constant or bind variable**
- **Turtle syntax, prefixes/shorthands allowed**

SPARQL code

```
PREFIX pse: <http://www.daml.org/2003/01/pse#>

SELECT ?element
WHERE {
  ?element pse:name "iron".
}
```

„Get the ID of all elements which have the name “iron“ “

More Details on Matching

How to deal with data types and languages for literals?

```
ex:bsp1 ex:p "test" .
ex:bsp2 ex:p "test"^^xsd:string .
ex:bsp3 ex:p "test"@de .
```

What is the result of

```
{ ?subject <http://example.org/p> "test" . }
```

Explicit type/language checks using the same syntax

```
{?subject <http://example.org/p> "test"^^xsd:string.}
```

Implicit typing with string (see above) and numbers

```
{ ?subject <http://example.org/p> 42 . }
```

Basic Graph Pattern

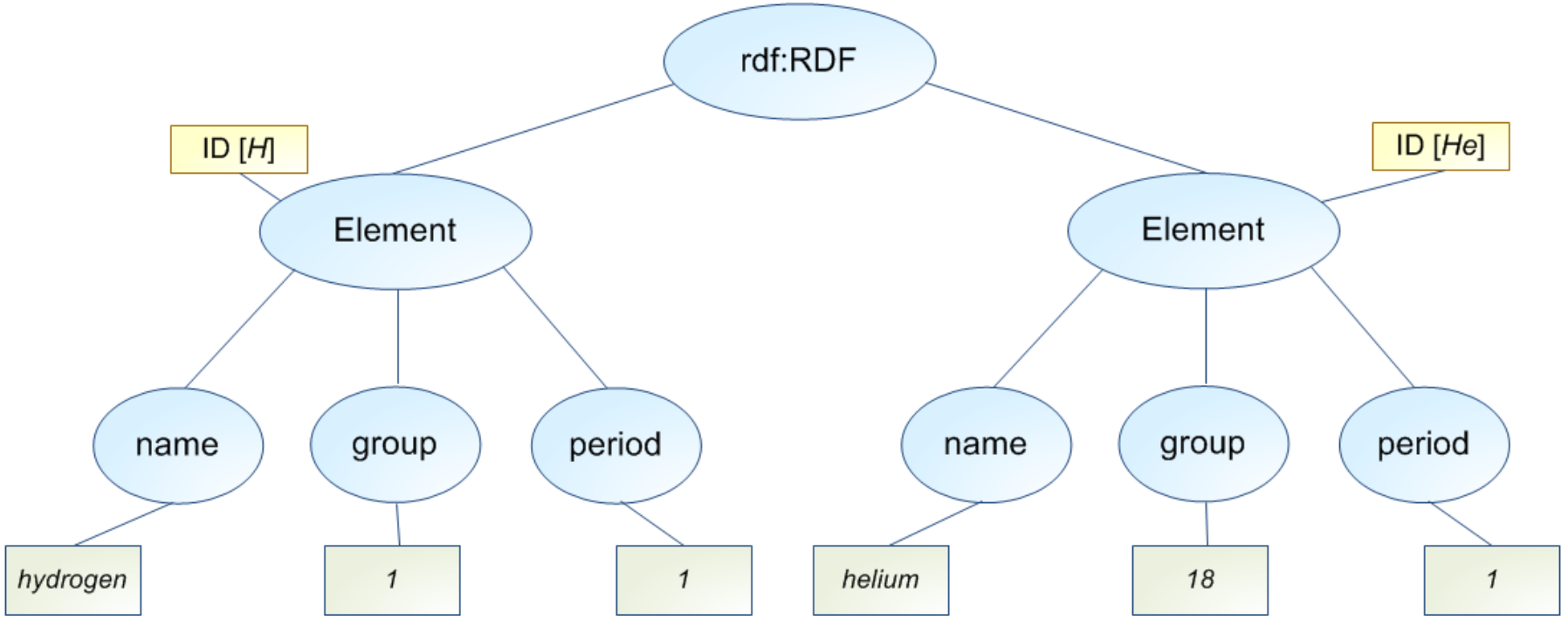
- contains a set of triple patterns
- each triple consists of subject, predicate and object
- Variables possible
- several triples in one basic graph pattern combined by conjunction

SPARQL code

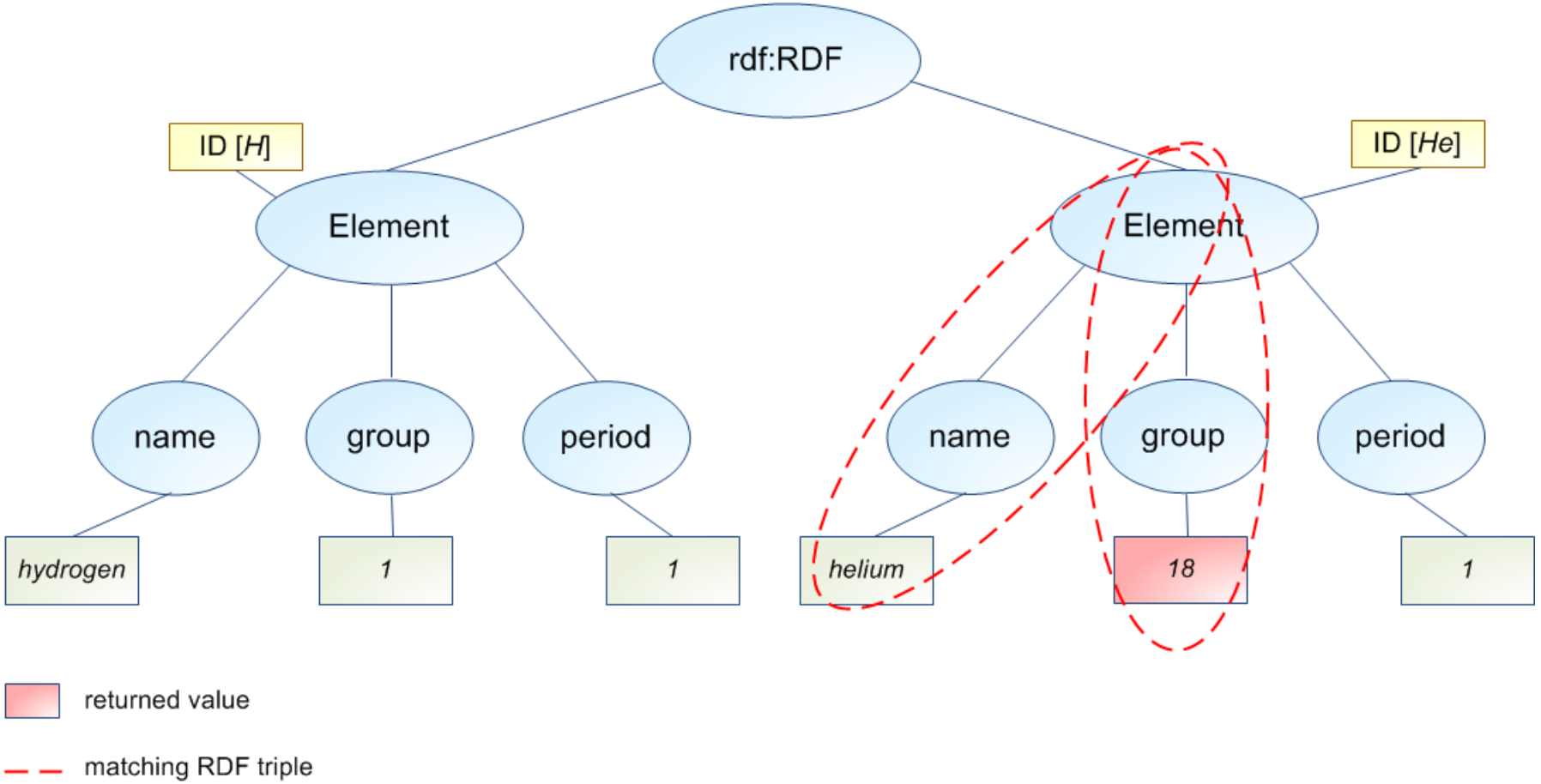
```
{
  ?element chemistry:name ?name.
  ?element chemistry:group 18.
}
```

„all chemical elements which have a name and are in group 18

Basic Graph Pattern: Example



Basic Graph Pattern: Example



„all chemical elements which have a name and are in group 18

Variable Bindings and Solutions

- Bindings of variables over triples generate a „solution“

?name	?group	?color
hydrogen	1	
helium	18	
iron		grey
iron	8	grey

- Not ordered
- May contain duplicates
- May contain variables without a value/binding

Blank nodes

- Recall blank nodes in RDF:
  - nodes without a resource ID
  - Local identifier
  - Describe existence of a node, but not its details
- Blank nodes in patterns
  - Can be specified as subject or object
  - Arbitrary, but distinct ids
  - Like variables which cannot be output
- Blank nodes in SPARQL results
  - Placeholder for unknown elements
  - IDs again arbitrary, only valid within query result

SPARQL: Complex Patterns

- Build more complex combinations of triple patterns
  - Groups
  - Optional
  - Union
  - Named Graph
- Pattern combinations are left-associative
- A Pattern B Pattern C = (A Pattern B) Pattern C

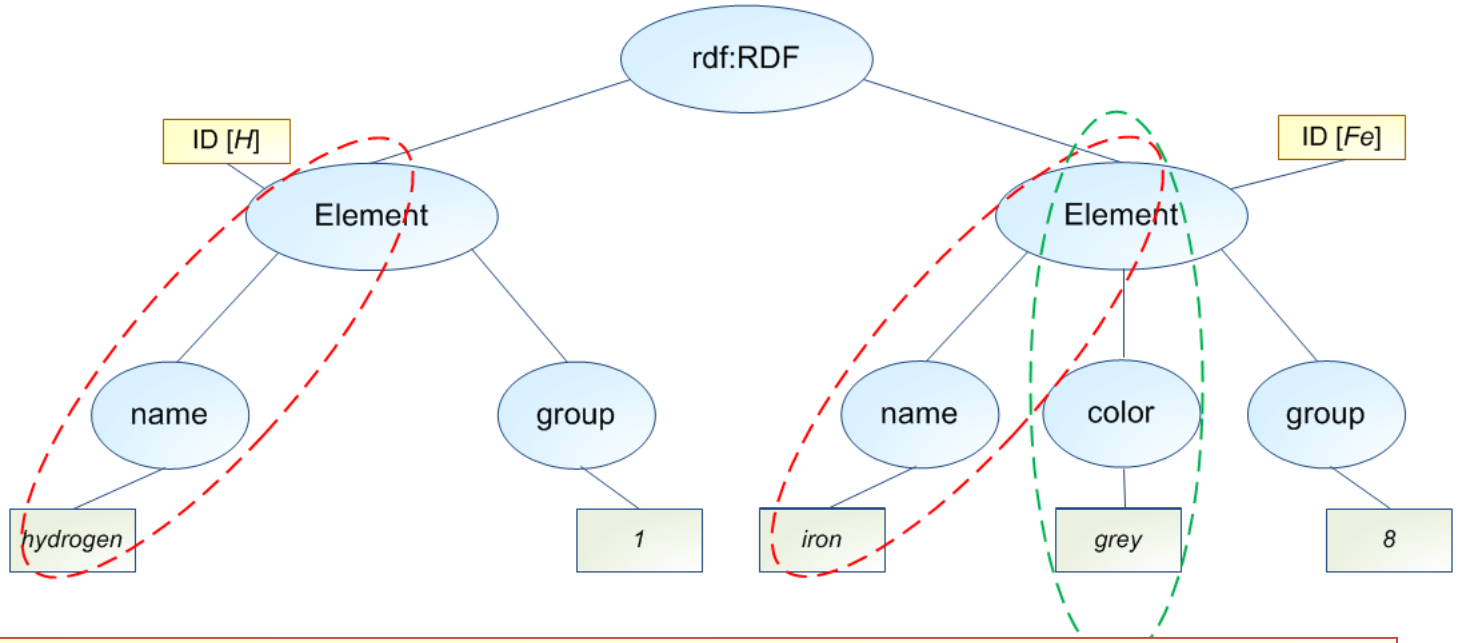
Optional Pattern

- Goal: supplement the solution with additional information
- Bind variables within OPTIONAL clause to one or many solutions
- Variable is unbound (=empty) if OPTIONAL clause does not match

SPARQL code

```
{
  pattern
  OPTIONAL { pattern }
  OPTIONAL { pattern }
  ...
}
```

Optional Pattern: Example



SPARQL code

```
{
  ?element chemistry:name ?name.
  OPTIONAL {
    ?element chemistry:color ?color.
   }
}
```

„elements which have a name and optionally a color “

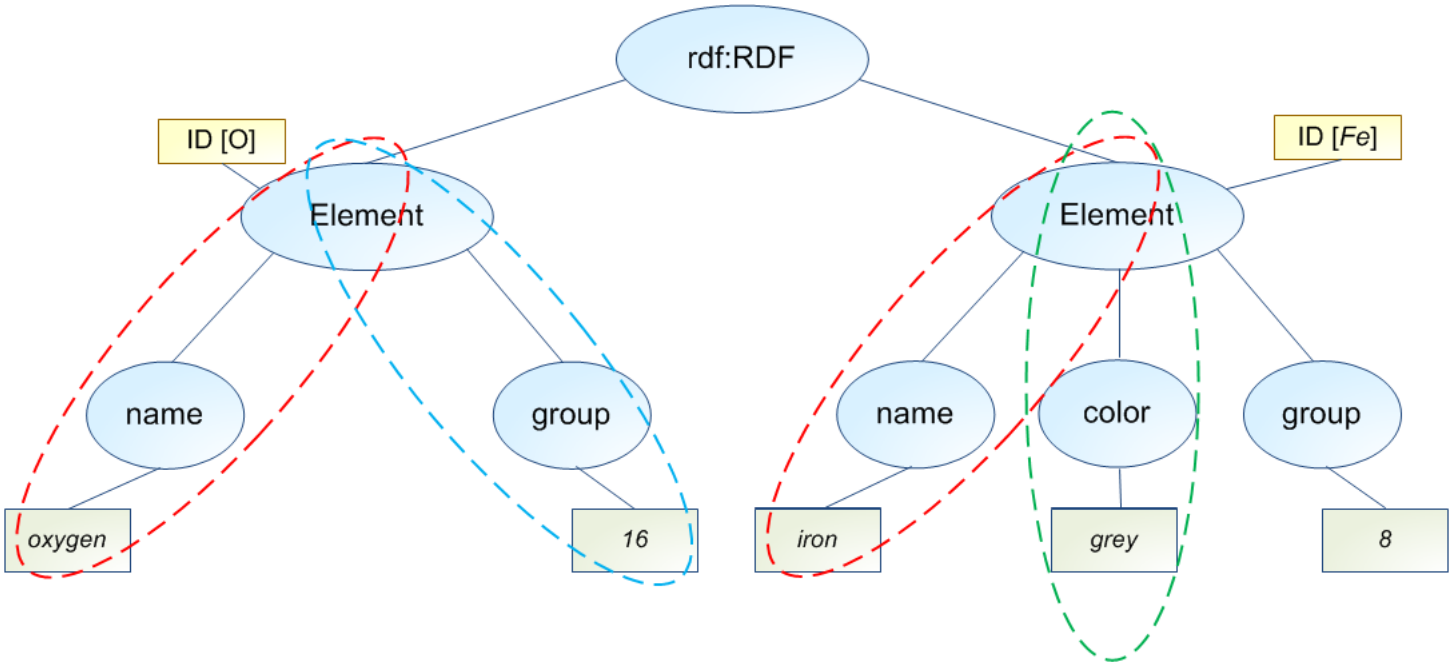
Alternative Pattern

- Combination of all solutions
- Total pattern matches if one or several pattern matches
- If more than one alternative found, return all solutions

SPARQL code

```
{ pattern }
UNION { pattern }
UNION { pattern }
...
```

Alternative Pattern: Example



SPARQL code

```
{
  ?element chemistry:group 16.
}
UNION {
  ?element chemistry:color ?color.
}
```

„elements which have a color or are in group 16“

SPARQL: Patterns

Group Graph Pattern

- In a Group Graph Pattern all patterns must match
- Used to provide additional structure among patterns
- Also allow empty groups {}

SPARQL code

```
{ pattern }
{ pattern }
{ pattern }
...
```

Named Graphs

- Adding additional RDF documents
- Name of the graph may again a variable
- Each query must define a default graph which is active when no named graph is in scope

SPARQL code

```
GRAPH ?src
{
  ?compound comp:element ?element.
  ?compound comp:name ?compoundName.
}
```

„retrieves elements whose name end in ‘ium’ “

Filters

- So far, we have performed exact matches on triples
- Need more complex predicates on solutions
- FILTER eliminates results if the effective boolean conditions false or an errors
- Borrows from XQuery/XPath functions and operators

SPARQL code

```
{
  ?element chemistry:name ?name.
  FILTER regex(?name, "ium$")
}
```

„retrieve elements whose name end in ‘ium’ “

Filter: Comparison

- Usual comparison operators:
  <, =, >, <=, >=, !=
- !=, = for all data types
- Other operators for numeric, string, literals, dateTime, boolean (1 > 0)
- No comparison of incompatible types

Filter: Arithmetics

- Again, usual operators: +, - , *, /
- Work on numeric data

Filter: Logical Operations and Errors

- A && B, A || B, !A
- Invoke *effective boolean value* for A and B
- Three-valued logic: True, False, Error
  - A || B: T, E => T; F, E => E
  - A && B: T, E => E; F, E => F
- FILTER: E => False

Filter: RDF/SPARQL-specific functions

- BOUND (Variable)
- isIRI/isURI
- isBLANK
- isLITERAL
- STR(literal), STR(IRI)
- LANG(literal)
- DATATYPE(typed literal), DATATYPE(simple literal)
- sameTERM
- langMATCHES
- REGEX

Solution Modifiers

- Solution as generated by patterns
  - Does not have an order
  - May contain duplicates
  - …

$\Rightarrow$ Solution Sequence Modifiers

  - ORDER BY
  - Projection: Choose a subset of variables
  - LIMIT, OFFSET
  - DISTINCT, REDUCED

ORDER BY

```
SELECT ?name
WHERE {
   ?element chemistry:name ?name.
   ?element chemistry:number ?number.
} ORDER BY ?number
```

- Order like in FILTER comparisons
- URIs in alphabetical order
- Not bound < blank node < URI < Literal

LIMIT, OFFSET, DISTINCT

- Restrict result set:
  - LIMIT: restrict maximum number of results
  - OFFSET: position of first delivered result
  - SELECT DISTINCT: remove duplicate values
  - REDUCED: allow removal of some duplicate values

```
SELECT DISTINCT ?name
WHERE {
   ?element chemistry:name ?name.
   ?element chemistry:number ?number.
} ORDER BY ?number OFFSET 10 LIMIT 5
```

Application order of modifiers

1. Sorting
2. Projection
3. Duplicate Elimination
4. Offset
5. Limit

Why?

SPARQL: Query Types

Different ways to present results - **Query Forms:**

- **`SELECT:`** return the value of variables which may be bound by a matching query pattern
- **`ASK:`** return true if a given query matches and false if not
- **`CONSTRUCT:`** return an RDF graph by substituting the values in given templates
- **`DESCRIBE:`** return an RDF graph which defines the matching resource

SPARQL: ASK Queries

Back to Introductory Example

- **ASK: Test if a query pattern has a solution**

SPARQL code

```
PREFIX pse: <http://www.daml.org/2003/01/pse#>

ASK {
  ?element pse:name "iron".
}
```

„Is there an element which have the name “iron“ “

SPARQL: CONSTRUCT Queries

Returning an RDF Graph

- **CONSTRUCT: Graph specified by a graph template**

SPARQL code

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>
CONSTRUCT {
  <http://example.org/person#Alice> vcard:FN ?name
}
WHERE {
  ?x foaf:name ?name
}
```

SPARQL: DESCRIBE Queries

Data about Resources

- **DESCRIBE: Returning an RDF graph with data about a resource**

SPARQL code

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
DESCRIBE ?x
WHERE {
  ?x foaf:name "Alice"
}
```

SPARQL 1.0 – Evaluation

- Relatively small language
- Provides basic triple matching and filtering operations
- Limited expressive power
- SQL-Style Syntax, limited graph operations and filters
- Semantics sometimes underspecified (see next lectures)
- SPARQL 1.1 overcomes many limitations

SPARQL 1.0 limitations

- Limited graphs operations: How to compute connectedness?
- No updates
- No aggregates
- No explicit negation
- No subqueries
- …