Graph-based RDF Data Managment

Lei Zou

Peking University
Institute of Computer Science and Technology

Joint work with M. Tamer Özsu (UW), Lei Chen (HKUST), Jeffery Xu Yu(CUHK), Haixun Wang (Google), Dongyan Zhao(PKU) and students (Xuchuan Shen,Youhuan Li, Ruizhe Huang, Shuo Han)

RDF and Semantic Web

- RDF is a language for the conceptual modeling of information about web resources
- A building block of semantic web
  - Facilitates exchange of information
  - Search engines can retrieve more relevant information
  - Facilitates data integration (mashes)
- Machine understandable
  - Understand the information on the web and the interrelationships among them

What's Sematic Web: A Simple Example (RDFa)

The traditional Web (HTML) only considers the display of the content.

How is the page displayed, such as which font and the format of the pictures ?

```
<html>
    <font size="3" color="red"> Lei Zou </font>
    </br>
    Email:<a href= "mailto: zoulei@pku.edu.cn">
zoulei@pku.edu.cn  </a>
    <p>
       <font size="3" color="black">Publications: </font>
    </p>
   <div>
        Lei Zou, Jinhui Mo, Lei Chen, M. Tamer Ozsu,
Dongyan Zhao, gStore: Answering SPARQL Queries Via
Subgraph Matching, VLDB, 2011
    </div>
</html>
```

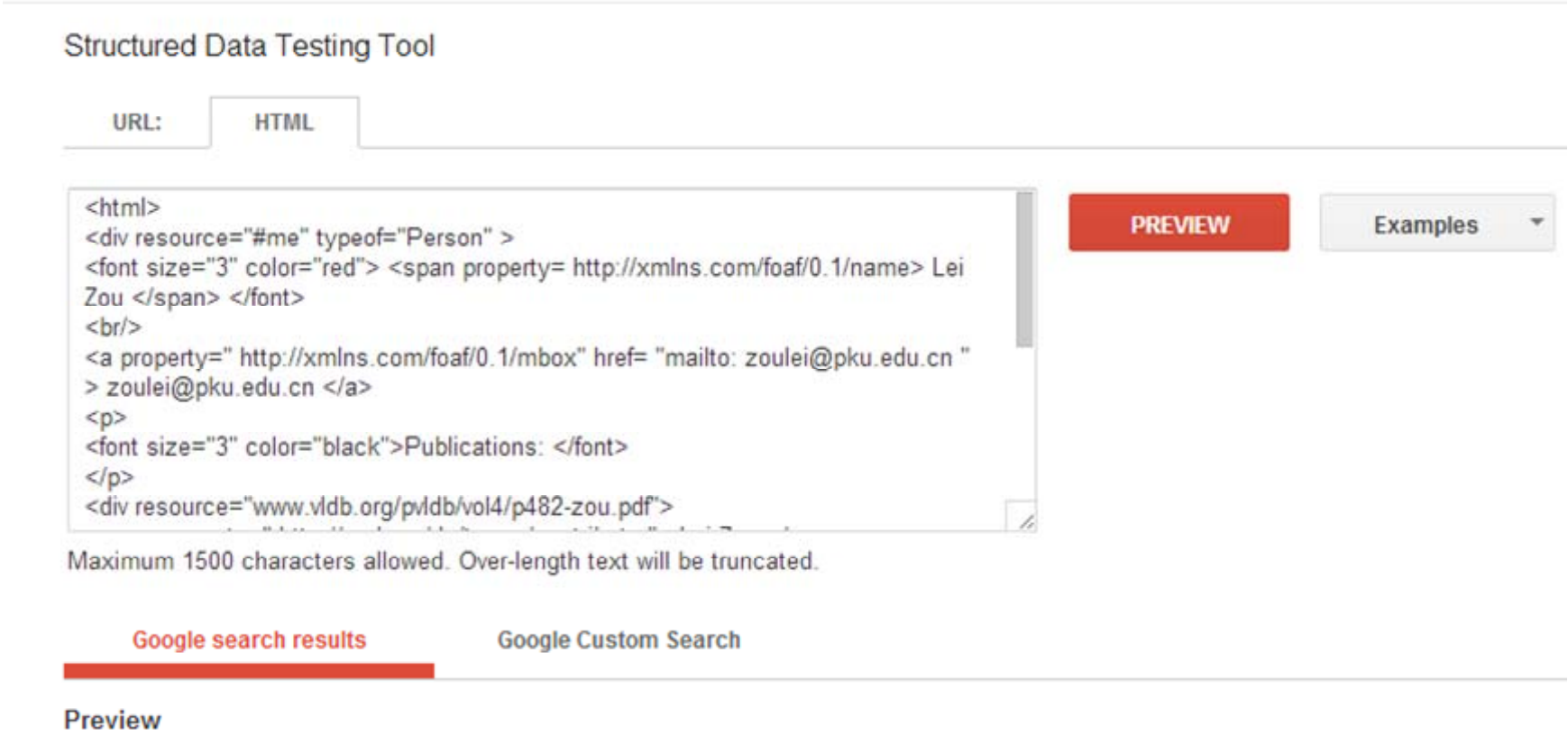
What's Sematic Web: A Simple Example (RDFa)

Sematic Web considers the sematics of the content.

What does the content in the page mean? e.g., What are the mean of "zoulei@pku.edu.cn" and "VLDB" ?

```
<html>
<div resource="#me" typeof="Person" >
<font size="3" color="red"> <span property= http://xmlns.com/foaf/0.1/name> Lei
Zou </span> </font>
<br/>
<a property=" http://xmlns.com/foaf/0.1/mbox" href= "mailto: zoulei@pku.edu.cn "
> zoulei@pku.edu.cn </a>
<p>
<font size="3" color="black">Publications: </font>
</p>
<div resource="www.vldb.org/pvldb/vol4/p482-zou.pdf">
    <span property=" http://purl.org/dc/terms/contributor"> Lei Zou </span>,
    <span property="http://purl.org/dc/terms/contributor"> Jinghui Mo </span>,
    <span property=" http://purl.org/dc/terms/contributor"> Lei Chen </span>,
    <span property=" http://purl.org/dc/terms/contributor"> M. Tamer Özsu</span>,
    <span property=" http://purl.org/dc/terms/contributor"> Dongyan Zhao</span>,
    <span property=" http://purl.org/dc/terms/title"> gStore: Answering SPARQL
Queries Via Subgraph Matching </span>,
    <span property=" http://purl.org/dc/terms/Publisher"> VLDB </span>
    <span property=" http://purl.org/dc/terms/Date">2011</span>
</div>
</html>
```

What's Sematic Web: Google Snippet



What's Sematic Web: Google Snippet

Extracted structured data

rdfa-node

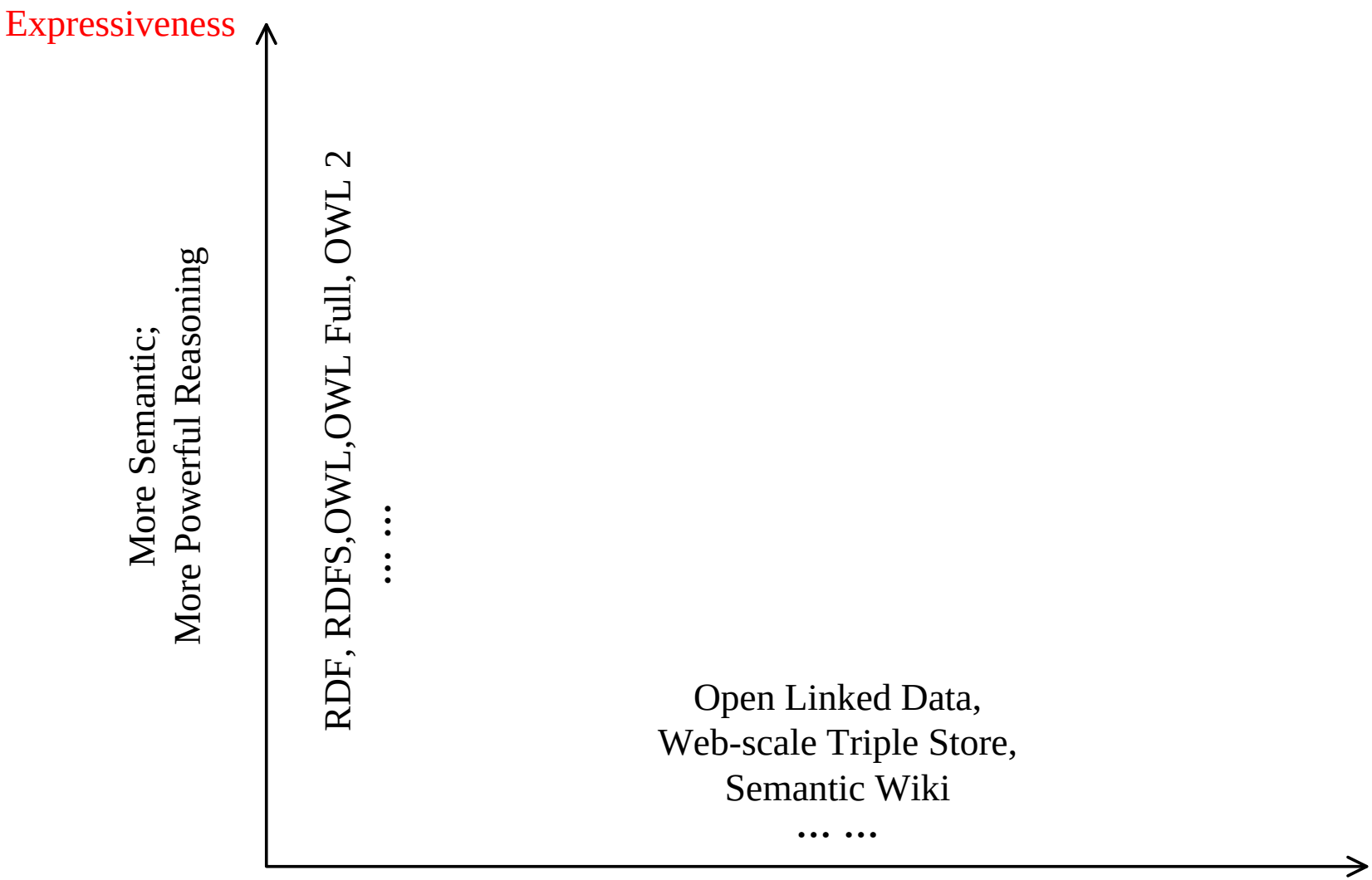
relationship:

name:	mbox
value:	zoulei@pku.edu.cn
href:	mailto:%20zoulei@pku.edu.cn

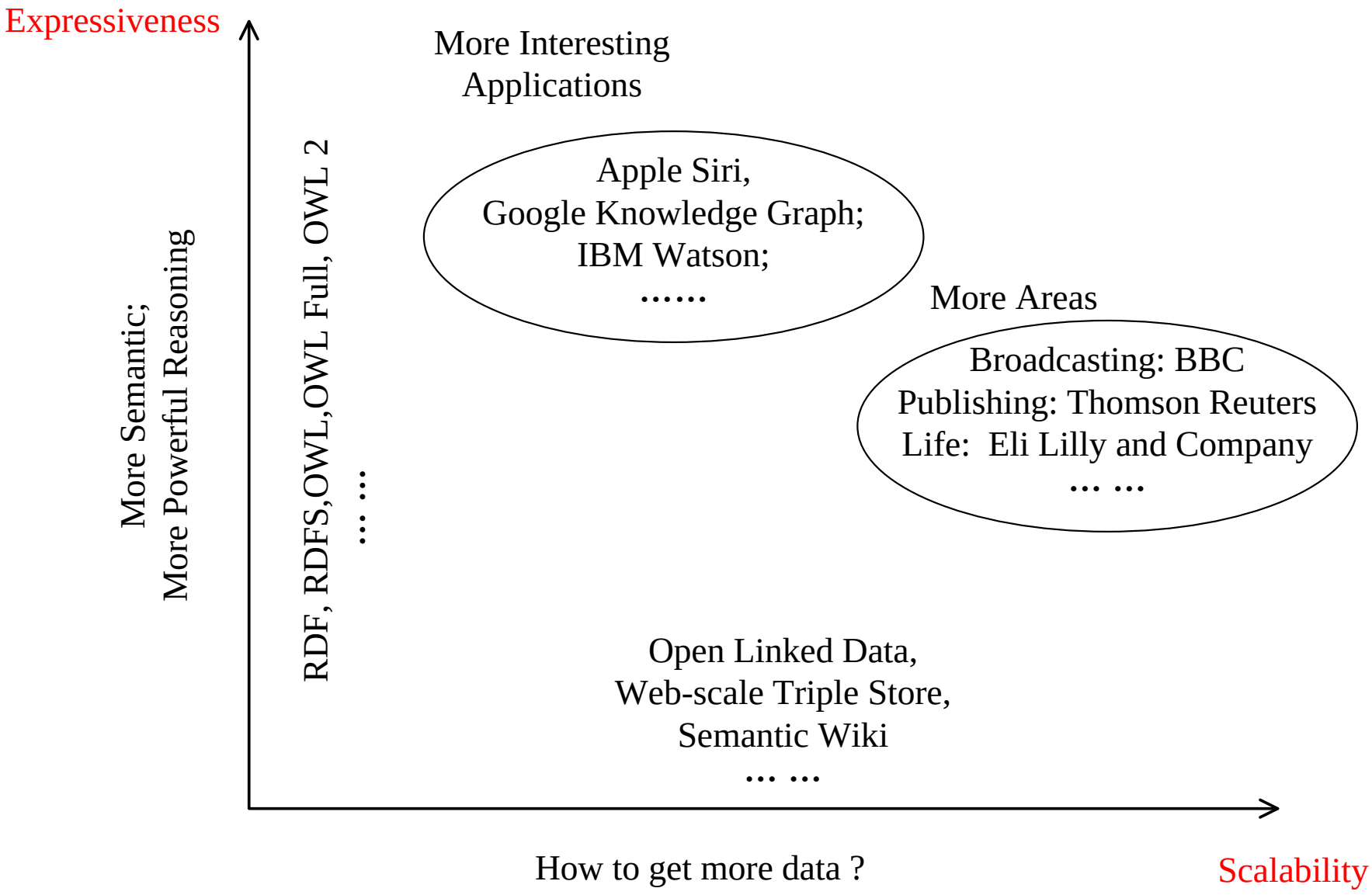
property:

name:	Lei Zou
contributor:	Lei Zou
contributor:	Jinghui Mo
contributor:	Lei Chen
contributor:	M. Tamer Özsu
contributor:	Dongyan Zhao
title:	gStore: Answering SPARQL Queries Via Subgraph Matching
Publisher:	VLDB
Date:	2011

What's Sematic Web: From Two Perspectives



What's Sematic Web: From Two Perspectives

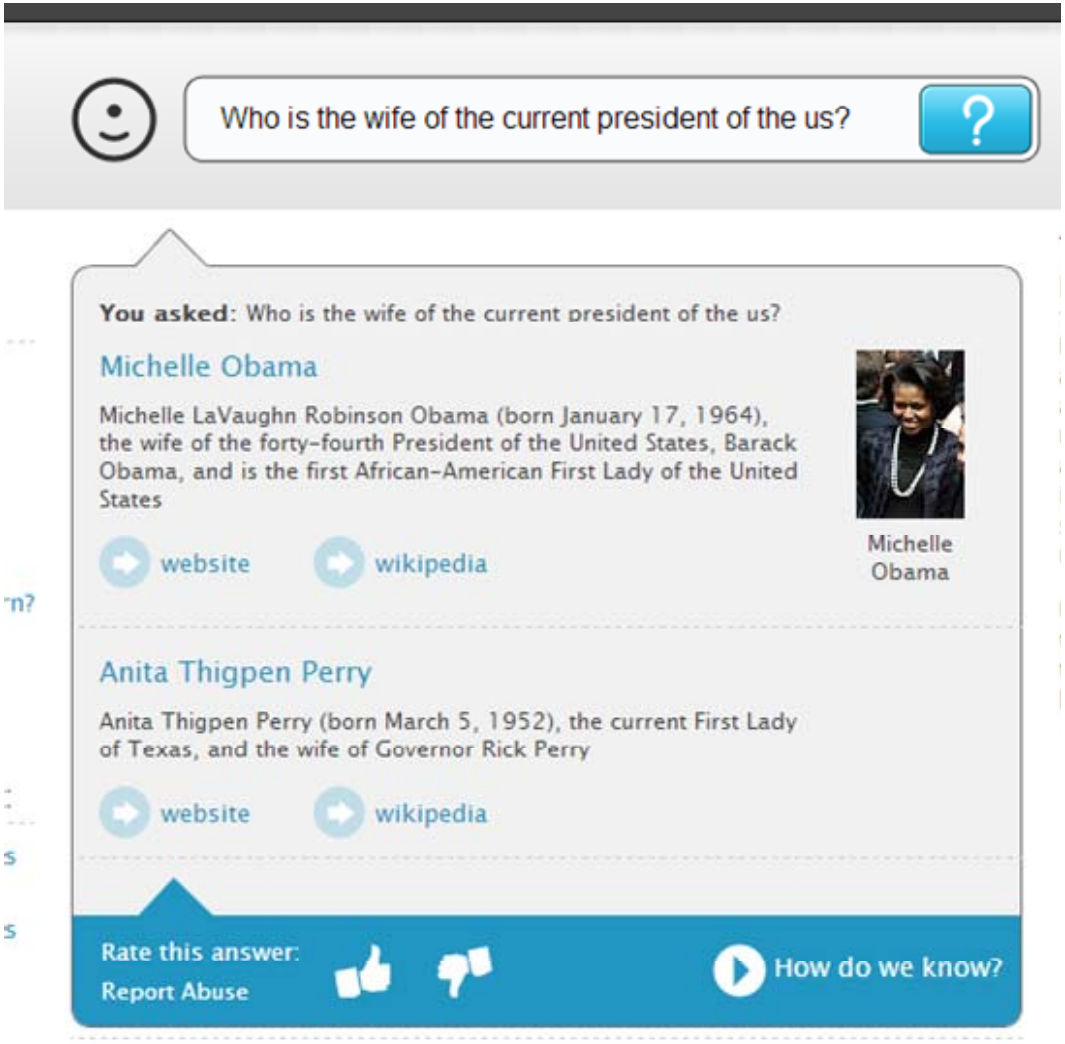


Some Interesting Products

IBM Watson

Some Interesting Products

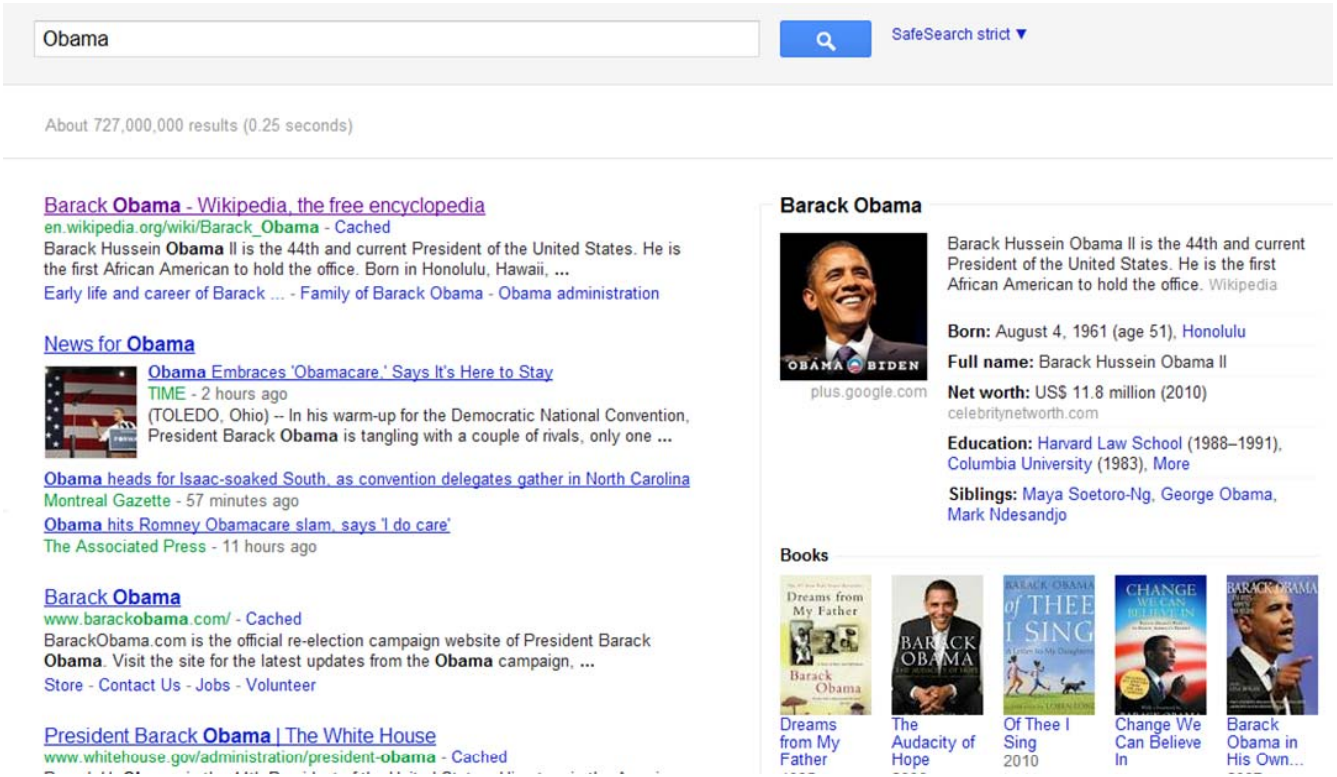
Evi



William Tunstall-Pedoe: True Knowledge: Open-Domain Question Answering Using Structured Knowledge and Inference. AI Magazine 31(3): 80-92 (2010)

Some Interesting Products

Google Knowledge Graph



RDF Uses

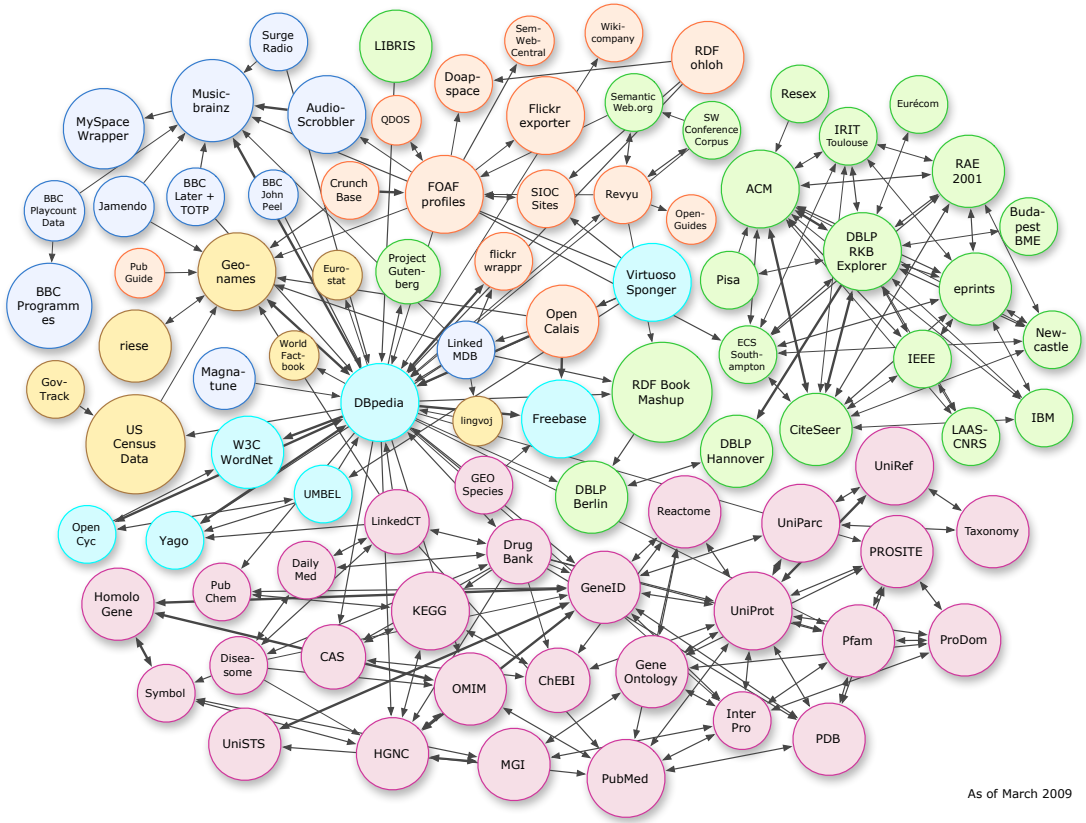
- Yago and DBPedia extract facts from Wikipedia & represent as RDF $\rightarrow$ structural queries
- Communities build RDF data
  - E.g., biologists: Bio2RDF and Uniprot RDF
- Web data integration
  - Linked Data Cloud
- ...

RDF Data Volumes . . .

- . . . are growing – and fast
  - Linked data cloud currently consists of 325 datasets with $>$25B triples
  - Size almost doubling every year

RDF Data Volumes . . .

- . . . are growing – and fast
  - Linked data cloud currently consists of 325 datasets with >25B triples
  - Size almost doubling every year

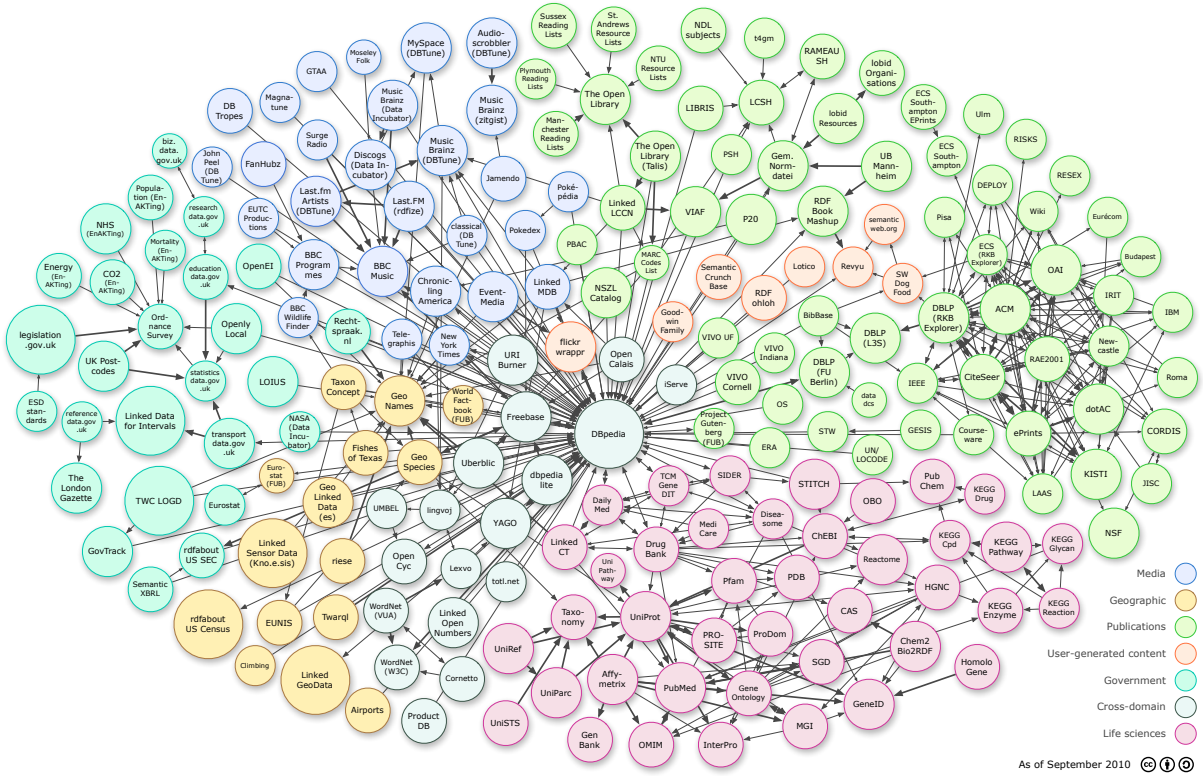


March '09:
89 datasets

Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch.
http://lod-cloud.net/

RDF Data Volumes . . .

- . . . are growing – and fast
  - Linked data cloud currently consists of 325 datasets with >25B triples
  - Size almost doubling every year

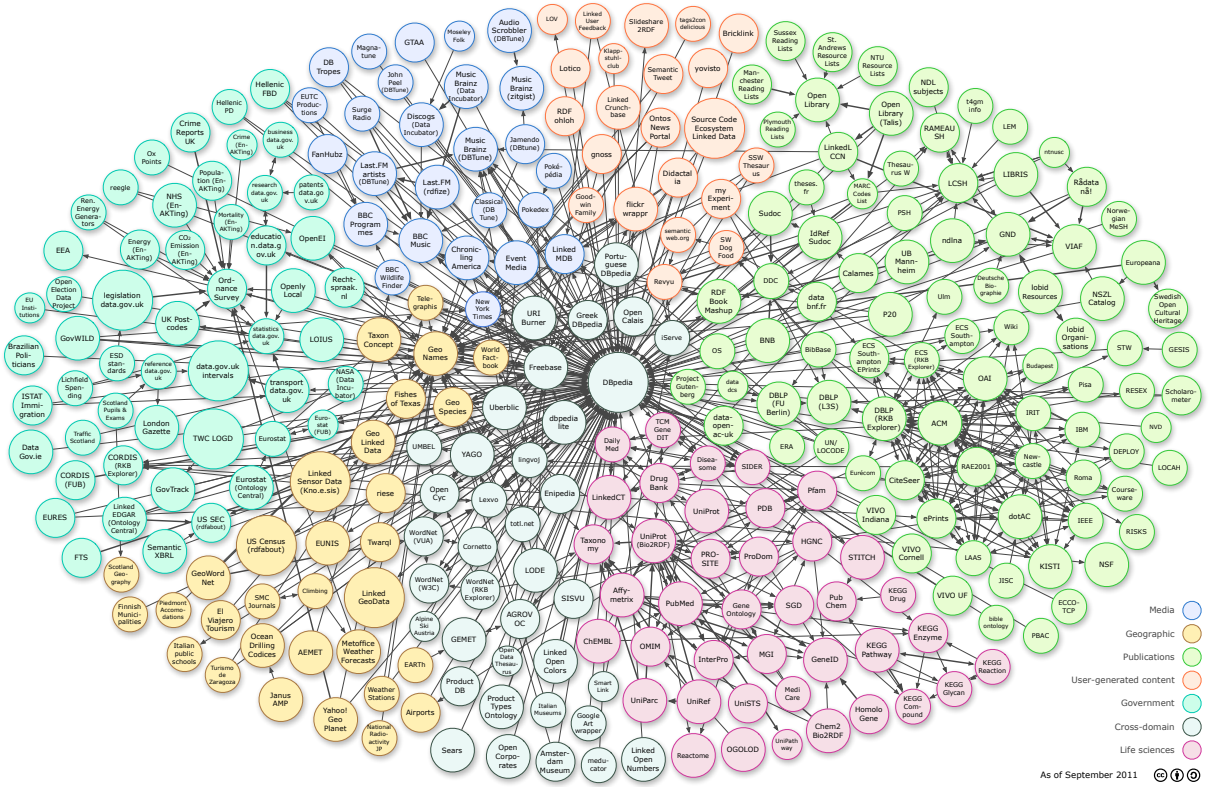


September '10:
203 datasets

Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch.
http://lod-cloud.net/

RDF Data Volumes . . .

- . . . are growing – and fast
  - Linked data cloud currently consists of 325 datasets with >25B triples
  - Size almost doubling every year

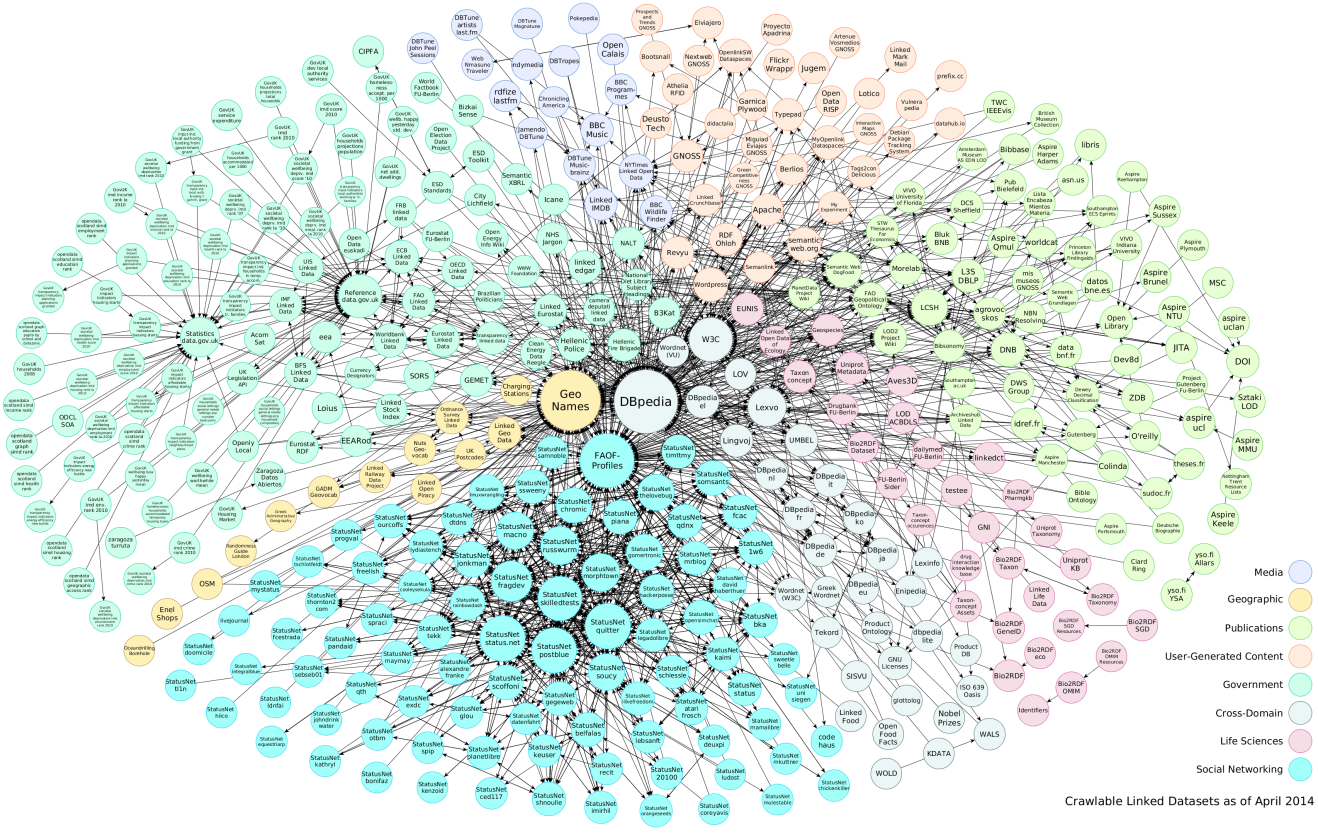


September '11:
295 datasets

Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch.
http://lod-cloud.net/

RDF Data Volumes . . .

- . . . are growing – and fast
  - Linked data cloud currently consists of 325 datasets with >25B triples
  - Size almost doubling every year



April '14:
1091 datasets, ??? triples

Max Schmachtenberg, Christian Bizer, and Heiko Paulheim: Adoption of Linked Data Best Practices in Different Topical Domains. In *Proc. ISWC*, 2014.

Outline

RDF Introduction

gStore: a graph-based SPARQL query engine

Answering SPARQL queries using graph pattern matching [Zou et al., PVLDB 2011, VLDB J 2014]

gAnswer: Natural Language Question Answering over RDF

A Graph Data Driven Approach [Zou et al., SIGMOD 2014]

Outline

RDF Introduction

gStore: a graph-based SPARQL query engine

Answering SPARQL queries using graph pattern matching [Zou et al., PVLDB 2011, VLDB J 2014]

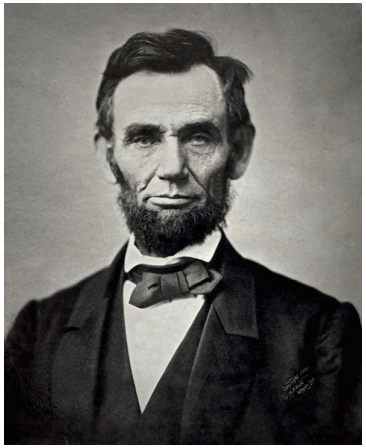
gAnswer: Natural Language Question Answering over RDF

A Graph Data Driven Approach [Zou et al., SIGMOD 2014]

RDF Introduction

- Everything is an uniquely named resource

http://en.wikipedia.org/wiki/Abraham_Lincoln

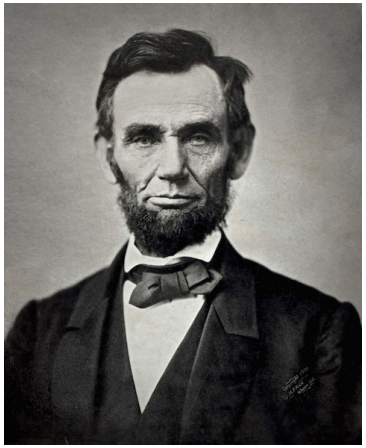


RDF Introduction

- Everything is an uniquely named resource
- Namespaces can be used to scope the names

xmlns:y=http://en.wikipedia.org/wiki

y:Abraham_Lincoln

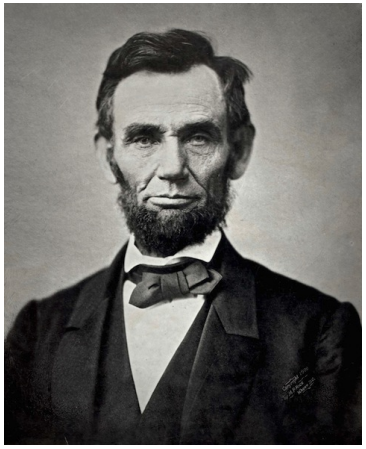


RDF Introduction

- Everything is an uniquely named resource
- Namespaces can be used to scope the names
- Properties of resources can be defined

xmlns:y=http://en.wikipedia.org/wiki

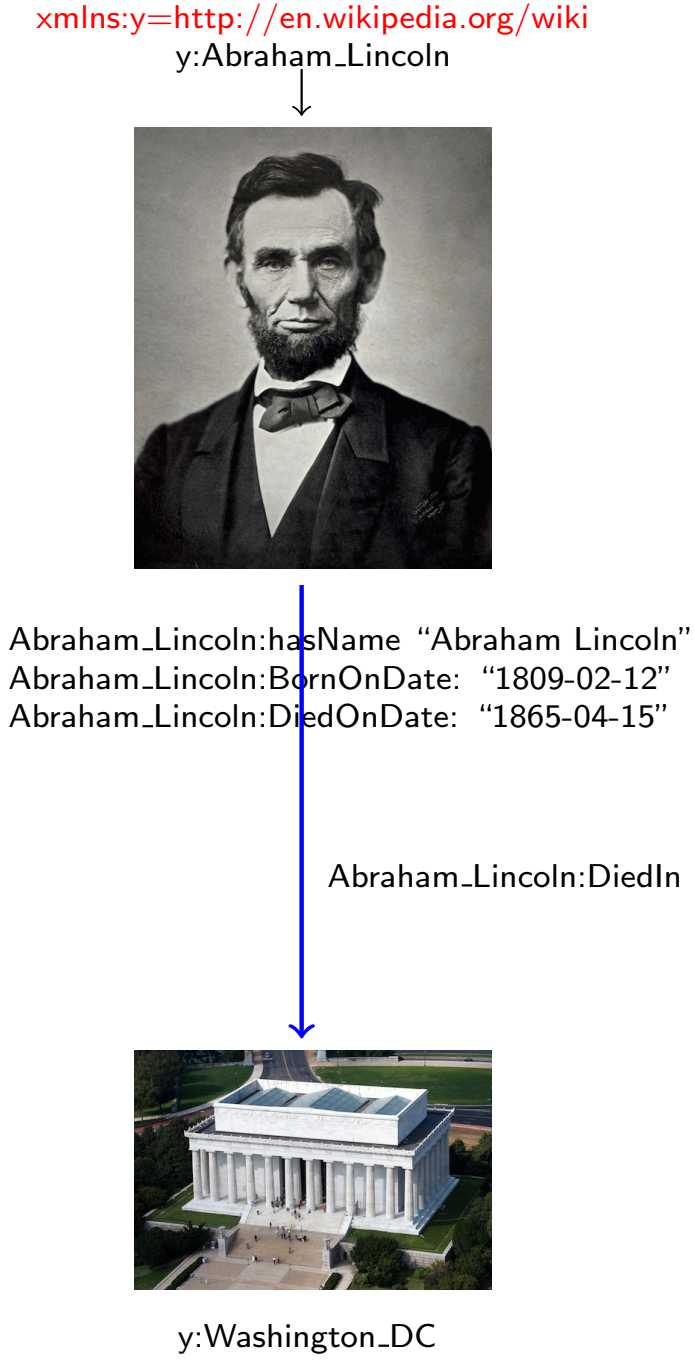
y:Abraham_Lincoln



Abraham_Lincoln:hasName "Abraham Lincoln"
Abraham_Lincoln:BornOnDate: "1809-02-12"
Abraham_Lincoln:DiedOnDate: "1865-04-15"

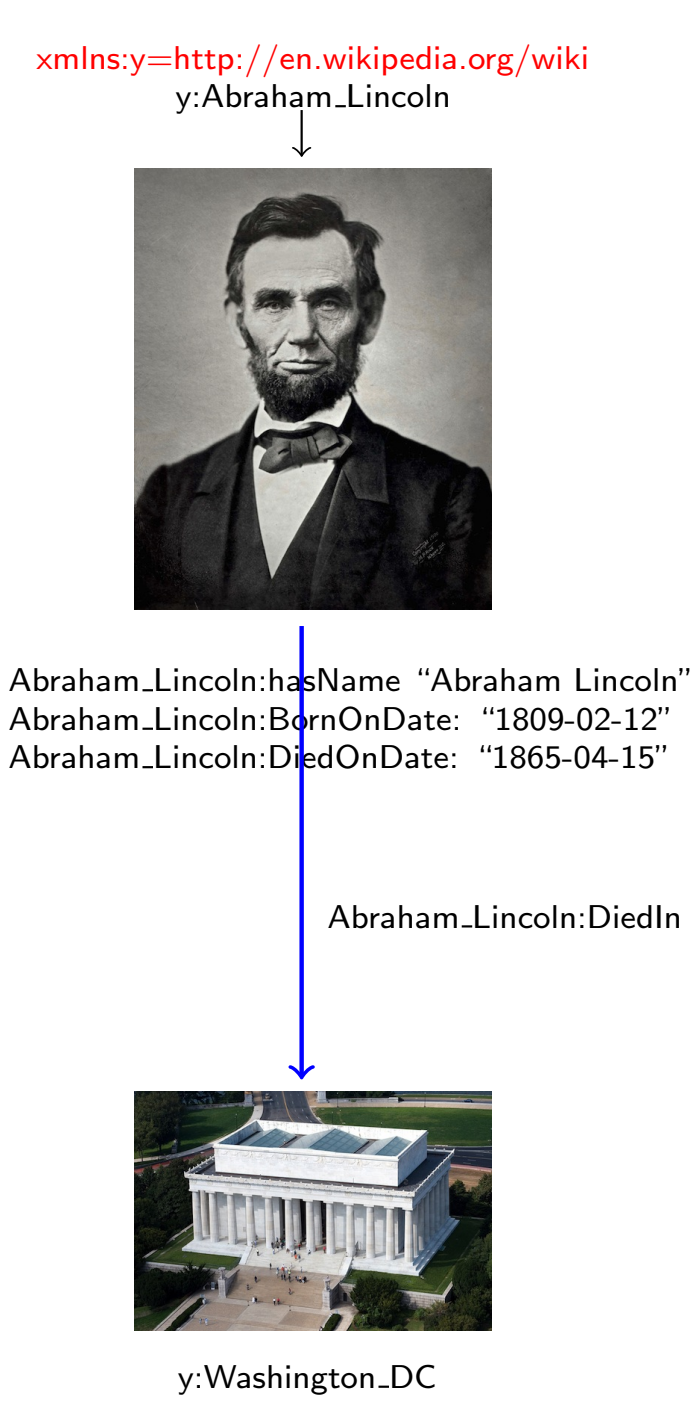
RDF Introduction

- Everything is an uniquely named resource
- Namespaces can be used to scope the names
- Properties of resources can be defined
- Relationships with other resources can be defined



RDF Introduction

- Everything is an uniquely named resource
- Namespaces can be used to scope the names
- Properties of resources can be defined
- Relationships with other resources can be defined
- Resources can be contributed by different people/groups and can be located anywhere in the web
  - Integrated web “database”



RDF Data Model

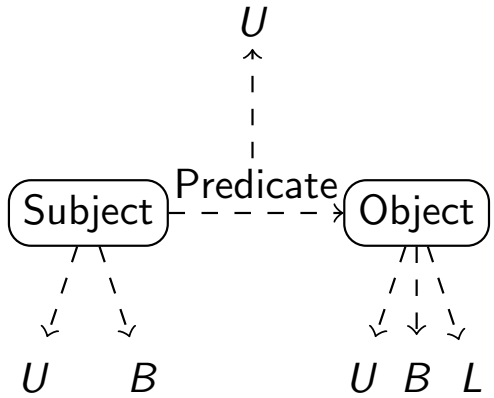
- Triple: Subject, Predicate (Property), Object (s, p, o)

  Subject: the entity that is described (URI or blank node)

  Predicate: a feature of the entity (URI)

  Object: value of the feature (URI, blank node or literal)

- $(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$
- Set of RDF triples is called an RDF graph



U: set of URIs
B: set of blank nodes
L: set of literals

Subject	Predicate	Object
Abraham_Lincoln	hasName	"Abraham Lincoln"
Abraham_Lincoln	BornOnDate	"1809-02-12"
Abraham_Lincoln	DiedOnDate	"1865-04-15"

RDF Example Instance

Prefix: y=http://en.wikipedia.org/wiki

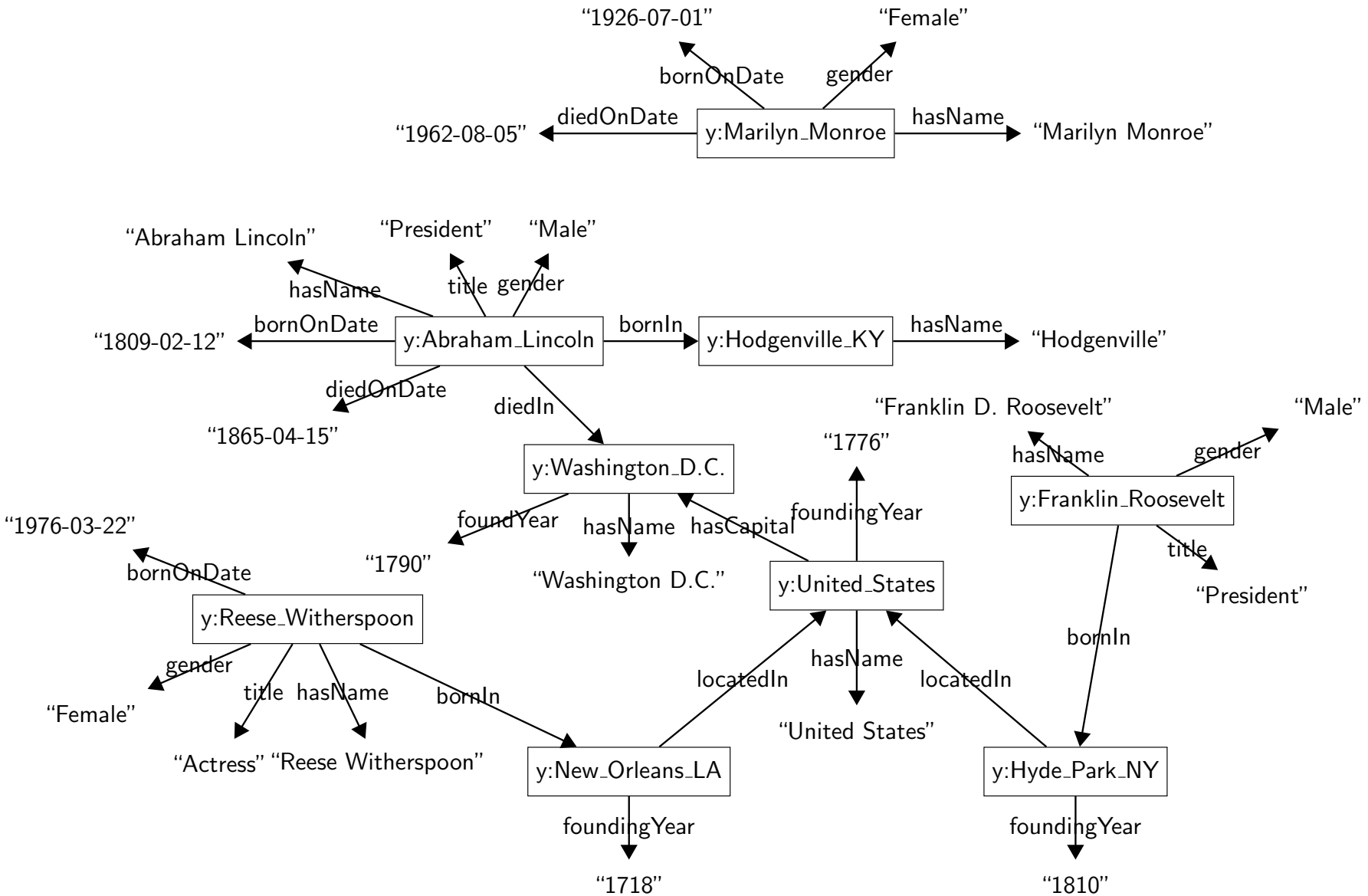
Subject	Predicate	Object
y: Abraham_Lincoln	hasName	"Abraham Lincoln"
y: Abraham_Lincoln	BornOnDate	"1809-02-12"'
y: Abraham_Lincoln	DiedOnDate	"1865-04-15"
y:Abraham_Lincoln	bornIn	y:Hodgenville_KY
y: Abraham_Lincoln	DiedIn	y: Washington_DC
y:Abraham_Lincoln	title	"President"
y:Abraham_Lincoln	gender	"Male"
y: Washington_DC	hasName	"Washington D.C."
y:Washington_DC	foundingYear	"1790"
y:Hodgenville_KY	hasName	"Hodgenville"
y:United_States	hasName	"United States"
y:United_States	hasCapital	y:Washington_DC
y:United_States	foundingYear	"1776"
y:Reese_Witherspoon	bornOnDate	"1976-03-22"
y:Reese_Witherspoon	bornIn	y:New_Orleans_LA
y:Reese_Witherspoon	hasName	"Reese Witherspoon"
y:Reese_Witherspoon	gender	"Female"
y:Reese_Witherspoon	title	"Actress"
y:New_Orleans_LA	foundingYear	"1718"
y:New_Orleans_LA	locatedIn	y:United_States
y:Franklin_Roosevelt	hasName	"Franklin D. Roosevelt"
y:Franklin_Roosevelt	bornIn	y:Hyde_Park_NY
y:Franklin_Roosevelt	title	"President"
y:Franklin_Roosevelt	gender	"Male"
y:Hyde_Park_NY	foundingYear	"1810"
y:Hyde_Park_NY	locatedIn	y:United_States
y:Marilyn_Monroe	gender	"Female"
y:Marilyn_Monroe	hasName	"Marilyn Monroe"
y:Marilyn_Monroe	bornOnDate	"1926-07-01"
y:Marilyn_Monroe	diedOnDate	"1962-08-05"

URI

Literal

URI

RDF Graph


"1926-07-01"
"Female"
bornOnDate
gender
diedOnDate
y:Marilyn_Monroe
hasName
"1962-08-05"
"Marilyn Monroe"
"Abraham Lincoln"
"President"
"Male"
hasName
title
gender
"1809-02-12"
bornOnDate
y:Abraham_Lincoln
bornIn
y:Hodgenville_KY
hasName
"Hodgenville"
diedOnDate
diedIn
"1865-04-15"
"Franklin D. Roosevelt"
"Male"
"1776"
y:Washington_D.C.
hasName
gender
foundYear
hasName
hasCapital
foundingYear
y:Franklin_Roosevelt
"1976-03-22"
"1790"
"Washington D.C."
title
"President"
bornOnDate
y:United_States
y:Reese_Witherspoon
bornIn
gender
hasName
locatedIn
locatedIn
"Female"
title
hasName
bornIn
"United States"
"Actress"
"Reese Witherspoon"
y:New_Orleans_LA
y:Hyde_Park_NY
foundingYear
foundingYear
"1718"
"1810"

RDF Query Model

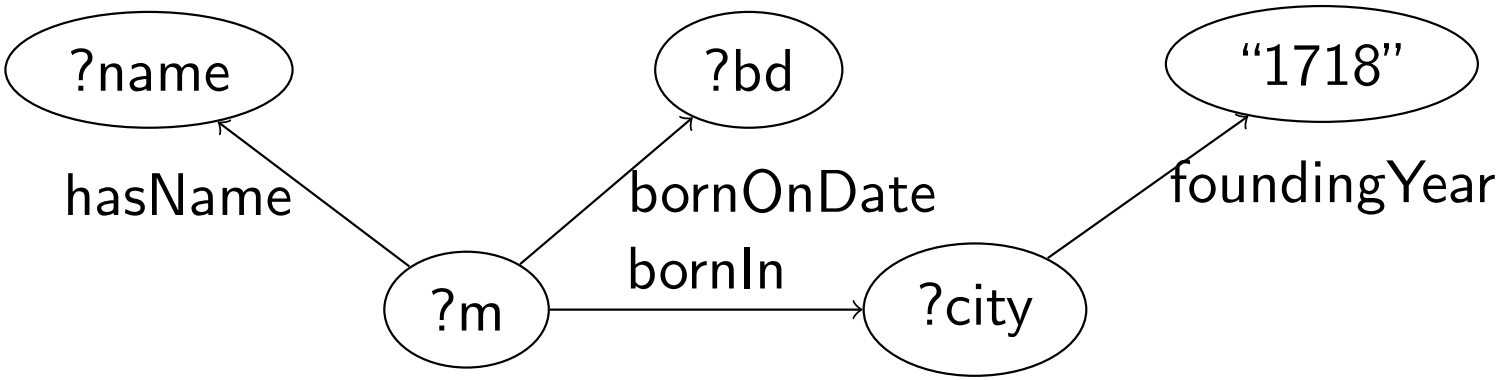
- Query Model - **S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage
- Given U (set of URIs), L (set of literals), and V (set of variables), a SPARQL expression is defined recursively:
  - an atomic triple pattern, which is an element of
    $$(U \cup V) \times (U \cup V) \times (U \cup V \cup L)$$
    - ?x hasName "Abraham Lincoln"
  - P FILTER R, where P is a graph pattern expression and R is a built-in SPARQL condition (i.e., analogous to a SQL predicate)
    - ?x price ?p FILTER(?p $<$ 30)
  - $P1$ AND/OPT/UNION $P2$, where $P1$ and $P2$ are graph pattern expressions
- Example:

```
SELECT ?name
WHERE {
   ?m <bornIn> ?city. ?m <hasName> ?name.
   ?m<bornOnDate> ?bd. ?city <foundingYear> ''1718''.
   FILTER(regex(str(?bd),''1976''))
}
```

SPARQL Queries

```
SELECT ?name
WHERE {
   ?m <bornIn> ?city. ?m <hasName> ?name.
   ?m<bornOnDate> ?bd. ?city <foundingYear> ''1718''.
   FILTER(regex(str(?bd),''1976''))
}
```

FILTER(*regex*(*str*(?bd),"1976"))



Naïve Triple Store Design

```
SELECT ?name
WHERE {
   ?m <bornIn> ?city. ?m <hasName> ?name.
   ?m<bornOnDate> ?bd. ?city <foundingYear> ''1718''.
   FILTER(regex(str(?bd),''1976''))
}
```

Subject	Property	Object
y:Abraham_Lincoln	hasName	"Abraham Lincoln"
y:Abraham_Lincoln	bornOnDate	"1809-02-12"
y:Abraham_Lincoln	diedOnDate	"1865-04-15"
y:Abraham_Lincoln	bornIn	y:Hodgenville_KY
y:Abraham_Lincoln	diedIn	y:Washington_DC
y:Abraham_Lincoln	title	"President"
y:Abraham_Lincoln	gender	"Male"
y:Washington_DC	hasName	"Washington D.C."
y:Washington_DC	foundingYear	"1790"
y:Hodgenville_KY	hasName	"Hodgenville"
y:United_States	hasName	"United States"
y:United_States	hasCapital	y:Washington_DC
y:United_States	foundingYear	"1776"
y:Reese_Witherspoon	bornOnDate	"1976-03-22"
y:Reese_Witherspoon	bornIn	y:New_Orleans_LA
y:Reese_Witherspoon	hasName	"Reese Witherspoon"
y:Reese_Witherspoon	gender	"Female"
y:Reese_Witherspoon	title	"Actress"
y:New_Orleans_LA	foundingYear	"1718"
y:New_Orleans_LA	locatedIn	y:United_States
y:Franklin_Roosevelt	hasName	"Franklin D. Roosevelt"
y:Franklin_Roosevelt	bornIn	y:Hyde_Park_NY
y:Franklin_Roosevelt	title	"President"
y:Franklin_Roosevelt	gender	"Male"
y:Hyde_Park_NY	foundingYear	"1810"
y:Hyde_Park_NY	locatedIn	y:United_States
y:Marilyn_Monroe	gender	"Female"
y:Marilyn_Monroe	hasName	"Marilyn Monroe"
y:Marilyn_Monroe	bornOnDate	"1926-07-01"
y:Marilyn_Monroe	diedOnDate	"1962-08-05"

Naïve Triple Store Design

```
SELECT ?name
WHERE {
   ?m <bornIn> ?city. ?m <hasName> ?name.
   ?m<bornOnDate> ?bd. ?city <foundingYear>''1718''.
   FILTER(regex(str(?bd),''1976''))
}
```

Subject	Property	Object
y:Abraham_Lincoln	hasName	"Abraham Lincoln"
y:Abraham_Lincoln	bornOnDate	"1809-02-12"
y:Abraham_Lincoln	diedOnDate	"1865-04-15"
y:Abraham_Lincoln	bornIn	y:Hodgenville_KY
y:Abraham_Lincoln	diedIn	y:Washington_DC
y:Abraham_Lincoln	title	"President"
y:Abraham_Lincoln	gender	"Male"
y:Washington_DC	hasName	"Washington D.C."
y:Washington_DC	foundingYear	"1790"
y:Hodgenville_KY	hasName	"Hodgenville"
y:United_States	hasName	"United States"
y:United_States	hasCapital	y:Washington_DC
y:United_States	foundingYear	"1776"
y:Reese_Witherspoon	bornOnDate	"1976-03-22"
y:Reese_Witherspoon	bornIn	y:New_Orleans_LA
y:Reese_Witherspoon	hasName	"Reese Witherspoon"
y:Reese_Witherspoon	gender	"Female"
y:Reese_Witherspoon	title	"Actress"
y:New_Orleans_LA	foundingYear	"1718"
y:New_Orleans_LA	locatedIn	y:United_States
y:Franklin_Roosevelt	hasName	"Franklin D. Roosevelt"
y:Franklin_Roosevelt	bornIn	y:Hyde_Park_NY
y:Franklin_Roosevelt	title	"President"
y:Franklin_Roosevelt	gender	"Male"
y:Hyde_Park_NY	foundingYear	"1810"
y:Hyde_Park_NY	locatedIn	y:United_States
y:Marilyn_Monroe	gender	"Female"
y:Marilyn_Monroe	hasName	"Marilyn Monroe"
y:Marilyn_Monroe	bornOnDate	"1926-07-01"
y:Marilyn_Monroe	diedOnDate	"1962-08-05"

```
SELECT T2.object
FROM   T as T1, T as T2, T as T3,
       T as T4
WHERE T1.property="bornIn"
AND T2.property="hasName"
AND T3.property="bornOnDate"
AND T1.subject=T2.subject
AND T2.subject=T3.subject
AND T4.propety="foundingYear"
AND T1.object=T4.subject
AND T4.object="1718"
AND T3.object LIKE '%1976%'
```

Naïve Triple Store Design

```
SELECT ?name
WHERE {
   ?m <bornIn> ?city. ?m <hasName> ?name.
   ?m<bornOnDate> ?bd. ?city <foundingYear>
   FILTER(regex(str(?bd),''1976''))
}
```

Subject	Property	Object
y:Abraham_Lincoln	hasName	"Abraham Lincoln"
y:Abraham_Lincoln	bornOnDate	"1809-02-12"
y:Abraham_Lincoln	diedOnDate	"1865-04-15"
y:Abraham_Lincoln	bornIn	y:Hodgenville_KY
y:Abraham_Lincoln	diedIn	y:Washington_DC
y:Abraham_Lincoln	title	"President"
y:Abraham_Lincoln	gender	"Male"
y:Washington_DC	hasName	"Washington D.C."
y:Washington_DC	foundingYear	"1790"
y:Hodgenville_KY	hasName	"Hodgenville"
y:United_States	hasName	"United States"
y:United_States	hasCapital	y:Washington_DC
y:United_States	foundingYear	"1776"
y:Reese_Witherspoon	bornOnDate	"1976-03-22"
y:Reese_Witherspoon	bornIn	y:New_Orleans_LA
y:Reese_Witherspoon	hasName	"Reese Witherspoon"
y:Reese_Witherspoon	gender	"Female"
y:Reese_Witherspoon	title	"Actress"
y:New_Orleans_LA	foundingYear	"1718"
y:New_Orleans_LA	locatedIn	y:United_States
y:Franklin_Roosevelt	hasName	"Franklin D. Roosevelt"
y:Franklin_Roosevelt	bornIn	y:Hyde_Park_NY
y:Franklin_Roosevelt	title	"President"
y:Franklin_Roosevelt	gender	"Male"
y:Hyde_Park_NY	foundingYear	"1810"
y:Hyde_Park_NY	locatedIn	y:United_States
y:Marilyn_Monroe	gender	"Female"
y:Marilyn_Monroe	hasName	"Marilyn Monroe"
y:Marilyn_Monroe	bornOnDate	"1926-07-01"
y:Marilyn_Monroe	diedOnDate	"1962-08-05"


```
SELECT T2.object
FROM   T as T1, T as T2, T as T3,
       T as T4
WHERE T1.property="bornIn"
AND T2.property="hasName"
AND T3.property="bornOnDate"
AND T1.subject=T2.subject
AND T2.subject=T3.subject
AND T4.propety="foundingYear"
AND T1.object=T4.subject
AND T4.object="1718"
AND T3.object LIKE '%1976%'
```

Existing Solutions

1. Property table
   - Each class of objects go to a different table $\Rightarrow$ similar to normalized relations
   - Eliminates some of the joins
2. Vertically partitioned tables
   - For each property, build a two-column table, containing both subject and object, ordered by subjects
   - Can use merge join (faster)
   - Good for subject-subject joins but does not help with subject-object joins
3. Exhaustive indexing
   - Create indexes for each permutation of the three columns
   - Query components become range queries over individual relations with merge-join to combine
   - Excessive space usage

Property Tables

- Grouping by entities; Jena [Wilkinson et al., SWDB 03] ,FlexTable [Wang et al., DASFAA 10] , DB2-RDF [Bornea et al., SIGMOD 13]
- *Clustered property table*: group together the properties that tend to occur in the same (or similar) subjects
- *Property-class table*: cluster the subjects with the same type of property into one property table

Subject	**Property**	**Object**
mdb:film/2014	rdfs:label	"The Shining"
mdb:film/2014	movie:director	mdb:director/8476
mdb:film/2685	movie:director	mdb:director/8476
mdb:film/2685	rdfs:label	"A Clockwork Orange"
mdb:actor/29704	movie:actor_name	"Jack Nicholson"
...	...	...

Subject	**refs:label**	**movie:director**
mob:film/2014	"The Shining"	mob:director/8476
mob:film/2685	"The Clockwork Orange"	mob:director/8476

Subject	**movie:actor_name**
mdb:actor	"Jack Nicholson"

Property Tables

- Grouping by entities; Jena [Wilkinson et al., SWDB 03] ,FlexTable [Wang et al., DASFAA 10] , DB2-RDF [Bornea et al., SIGMOD 13]
- *Clustered property table*: group together the properties that tend to occur in the same (or similar) subjects

Advantages

- Fewer joins
- If the data is structured, we have a relational system – similar to normalized relations

mdb:film/2685	movie:director	mdb:director/8476
mdb:film/2685	rdfs:label	"A Clockwork Orange"
mdb:actor/29704	movie:actor_name	"Jack Nicholson"
...	...	...

Subject	**refs:label**	**movie:director**
mob:film/2014	"The Shining"	mob:director/8476
mob:film/2685	"The Clockwork Orange"	mob:director/8476

Subject	**movie:actor_name**
mdb:actor	"Jack Nicholson"

Property Tables

- Grouping by entities; Jena [Wilkinson et al., SWDB 03] ,FlexTable [Wang et al., DASFAA 10] , DB2-RDF [Bornea et al., SIGMOD 13]
- *Clustered property table*: group together the properties that tend to occur in the same (or similar) subjects

Advantages

- Fewer joins
- If the data is structured, we have a relational system – similar to normalized relations

Disadvantages

- Potentially a lot of NULLs
- Clustering is not trivial
- Multi-valued properties are complicated

mob:film/2014	"The Shining"	mob:director/8476
mob:film/2685	"The Clockwork Orange"	mob:director/8476

mdb:actor	"Jack Nicholson"

Binary Tables

- Grouping by properties: For each property, build a two-column table, containing both subject and object, ordered by subjects [Abadi et al., VLDB 07]
- Also called vertical partitioned tables
- n two column tables (n is the number of unique properties in the data)

Subject	Property	Object
mdb:film/2014	rdfs:label	"The Shining"
mdb:film/2014	movie:director	mdb:director/8476
mdb:film/2685	movie:director	mdb:director/8476
mdb:film/2685	rdfs:label	"A Clockwork Orange"
mdb:actor/29704	movie:actor_name	"Jack Nicholson"
...	...	...

refs:label

Subject	Object
mob:film/2014	"The Shining"
mob:film/2685	"The Clockwork Orange"

movie:director

Subject	Object
mdb:film/2014	mdb:director/8476
mdb:film/2685	mdb:director/8476

movie:actor_name

Subject	Object
mdb:actor/29704	"Jack Nicholson"

Binary Tables

- Grouping by properties: For each property, build a two-column table, containing both subject and object, ordered by subjects

Advantages

- Supports multi-valued properties
- No NULLs
- No clustering
- Read only needed attributes (i.e. less I/O)
- Good performance for subject-subject joins

mdb:film/2685	rdfs:label	"A Clockwork Orange"
mdb:actor/29704	movie:actor_name	"Jack Nicholson"
...	...	...

refs:label

Subject	Object
mob:film/2014	"The Shining"
mob:film/2685	"The Clockwork Orange"

movie:director

Subject	Object
mdb:film/2014	mdb:director/8476
mdb:film/2685	mdb:director/8476

movie:actor_name

Subject	Object
mdb:actor/29704	"Jack Nicholson"

Binary Tables

- Grouping by properties: For each property, build a two-column table, containing both subject and object, ordered by subjects

Advantages

- Supports multi-valued properties
- No NULLs
- No clustering
- Read only needed attributes (i.e. less I/O)
- Good performance for subject-subject joins

Disadvantages

- Not useful for subject-object joins
- Expensive inserts

Subject	Object
mob:film/2014	"The Shining"
mob:film/2685	"The Clockwork Orange"

Subject	Object
mdb:film/2014	mdb:director/8476
mdb:film/2685	mdb:director/8476

Subject	Object
mdb:actor/29704	"Jack Nicholson"

Exhaustive Indexing

- RDF-3X [Neumann and Weikum, PVLDB 08] , Hexastore [Weiss et al., PVLDB 08]
- Strings are mapped to ids using a mapping table

Original triple table

Subject	Property	Object
mdb: film/2014	rdfs:label	"The Shining"
mdb:film/2014	movie:initial_release_date	"1980-05-23"
mdb:director/8476	movie:director_name	"Stanley Kubrick"
mdb:film/2685	movie:director	mdb:director/8476

Encoded triple table

Subject	Property	Object
0	1	2
0	3	4
5	6	7
8	9	5

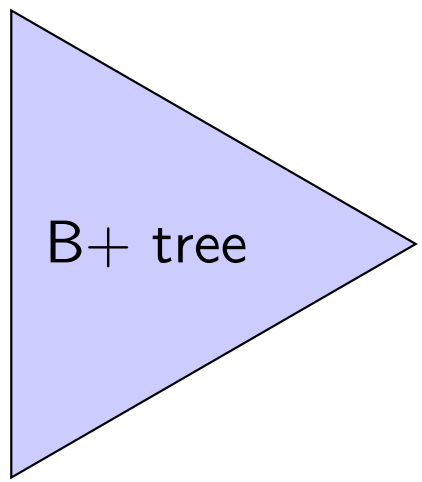
Mapping table

ID	Value
0	mdb: film/2014
1	rdfs:label
2	"The Shining"
3	movie:initial_release_date
4	"1980-05-23"
5	mdb:director/8476
6	movie:director_name
7	"Stanley Kubrick"
8	mdb:film/2685
9	movie:director

Exhaustive Indexing

- RDF-3X [Neumann and Weikum, PVLDB 08] , Hexastore [Weiss et al., PVLDB 08]
- Strings are mapped to ids using a mapping table
- Triples are indexed in a clustered B+ tree in lexicographic order

Subject	Property	Object
0	1	2
0	3	4
5	6	7
8	9	5
⋮	⋮	⋮

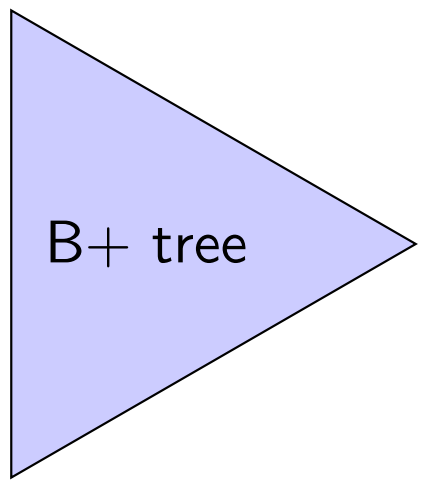


Easy querying through mapping table

Exhaustive Indexing

- RDF-3X [Neumann and Weikum, PVLDB 08] , Hexastore [Weiss et al., PVLDB 08]
- Strings are mapped to ids using a mapping table
- Triples are indexed in a clustered B+ tree in lexicographic order
- Create indexes for permutations of the three columns: SPO, SOP, PSO, POS, OPS, OSP

Subject	Property	Object
0	1	2
0	3	4
5	6	7
8	9	5
⋮	⋮	⋮



Easy querying through mapping table

Exhaustive Indexing–Query Execution

- Each triple pattern can be answered by a range query
- Joins between triple patterns computed using merge join
- Join order is easy due to extensive indexing

Subject	Property	Object
0	1	2
0	3	4
5	6	7
8	9	5
⋮	⋮	⋮

ID	Value
0	mdb: film/2014
1	rdfs:label
2	"The Shining"
3	movie:initial_release_date
4	"1980-05-23"
5	mdb:director/8476
6	movie:director_name
7	"Stanley Kubrick"
8	mdb:film/2685
9	movie:director

Exhaustive Indexing–Query Execution

- Each triple pattern can be answered by a range query
- Joins between triple patterns computed using merge join

Advantages

- Eliminates some of the joins – they become range queries
- Merge join is easy and fast

5	6	7
8	9	5
⋮	⋮	⋮

2	"The Shining"
3	movie:initial_release_date
4	"1980-05-23"
5	mdb:director/8476
6	movie:director_name
7	"Stanley Kubrick"
8	mdb:film/2685
9	movie:director

Exhaustive Indexing–Query Execution

- Each triple pattern can be answered by a range query
- Joins between triple patterns computed using merge join

Advantages

- Eliminates some of the joins – they become range queries
- Merge join is easy and fast

Disadvantages

- Space usage

5	mdb:director/8476
6	movie:director_name
7	"Stanley Kubrick"
8	mdb:film/2685
9	movie:director

Outline

RDF Introduction

gStore: a graph-based SPARQL query engine

Answering SPARQL queries using graph pattern matching [Zou et al., PVLDB 2011, VLDB J 2014]

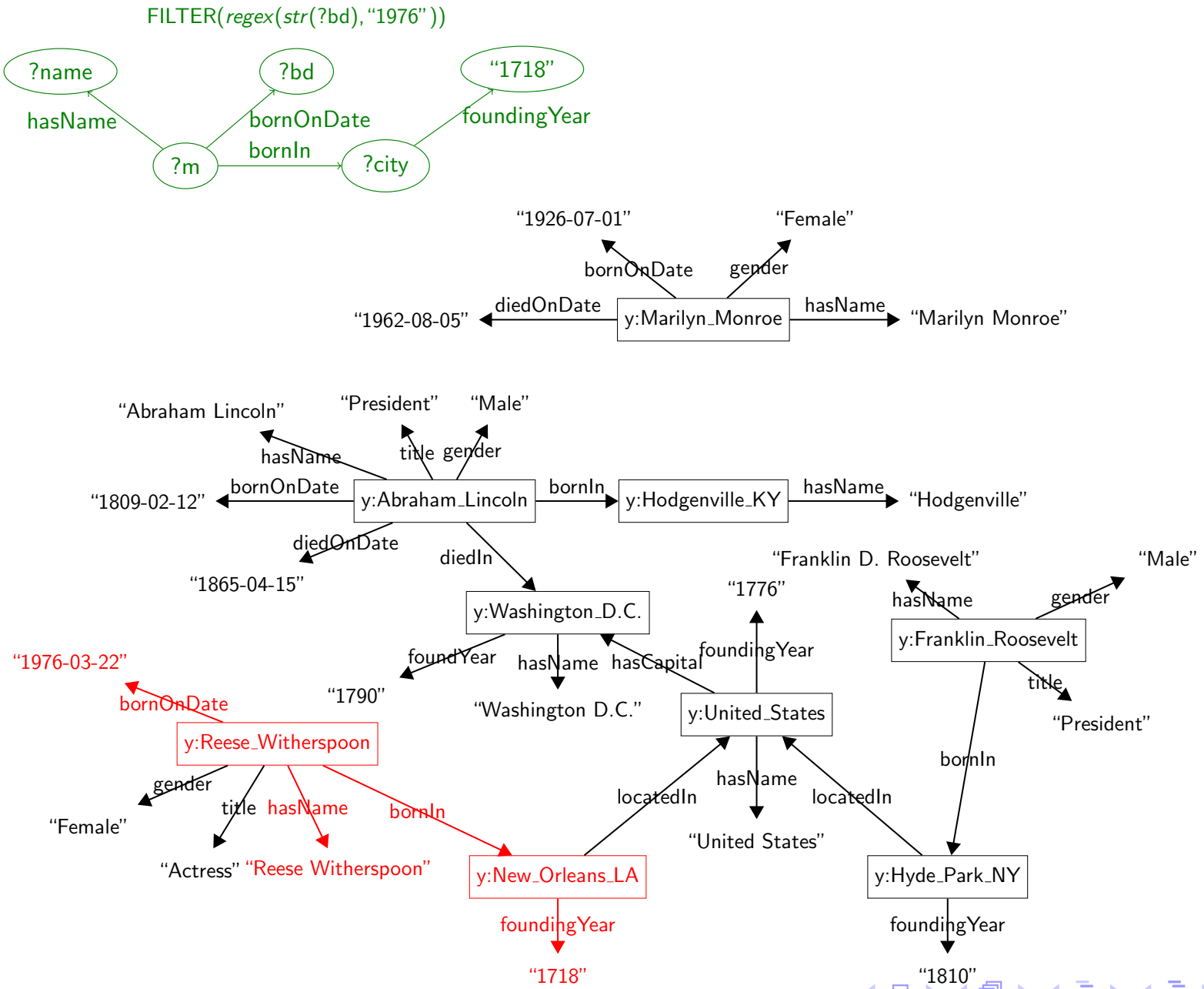
gAnswer: Natural Language Question Answering over RDF

A Graph Data Driven Approach [Zou et al., SIGMOD 2014]

gStore – General Idea

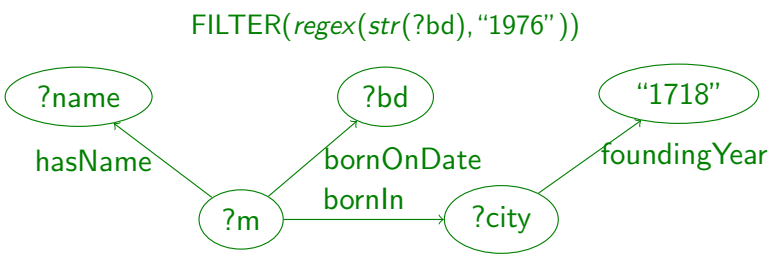
- We work directly on the RDF graph and the SPARQL query graph
  - Answering SPARQL query $\equiv$ subgraph matching
  - Subgraph matching is computationally expensive
- Use a signature-based encoding of each entity and class vertex to speed up matching
- Filter-and-evaluate
  - Use a false positive algorithm to prune nodes and obtain a set of candidates; then do more detailed evaluation on those
- We develop an index (VS*-tree) over the data signature graph (has light maintenance load) for efficient pruning

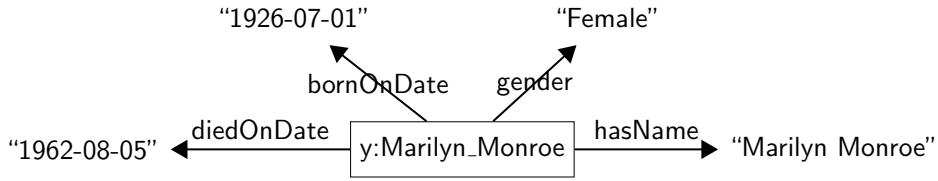
0. Start with RDF Graph *G*

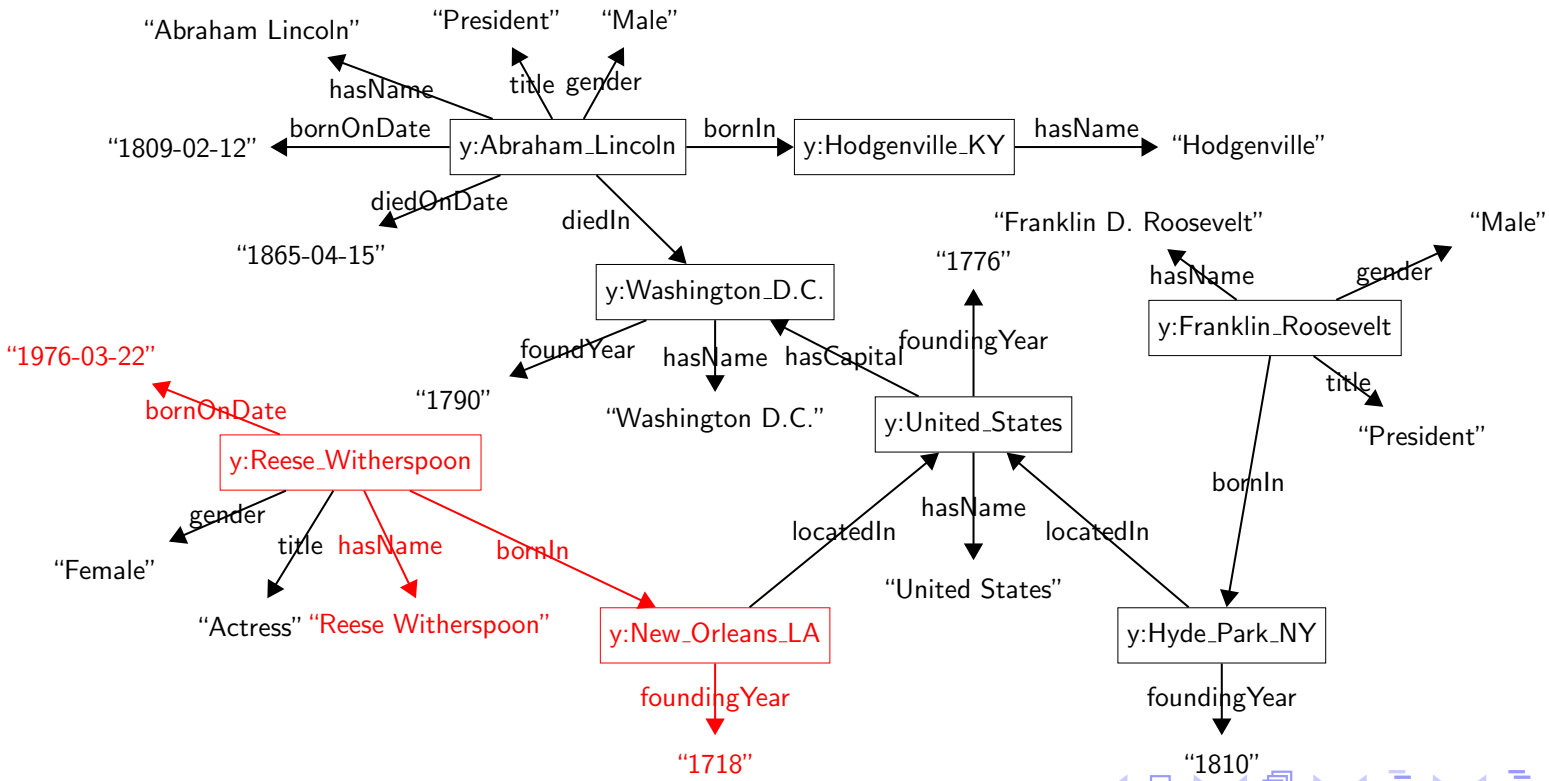


0. Start with RDF Graph G

Finding matches over a large graph is not a trivial task!







1. Represent G as Adjacency List and Encode Each Vertex

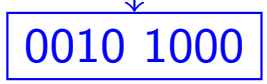
Prefix: y=http://en.wikipedia.org/wiki/

Label	adjList
⋮	⋮
y:Reese_Witherspoon	(BornOnDate, "1976-03-22"), (gender, "Female"), (title, "Actress"), (hasName, "Reese Witherspoon"), (BornIn,New_Orleans_LA)
⋮	⋮

1. Represent G as Adjacency List and Encode Each Vertex

Prefix: y=http://en.wikipedia.org/wiki/

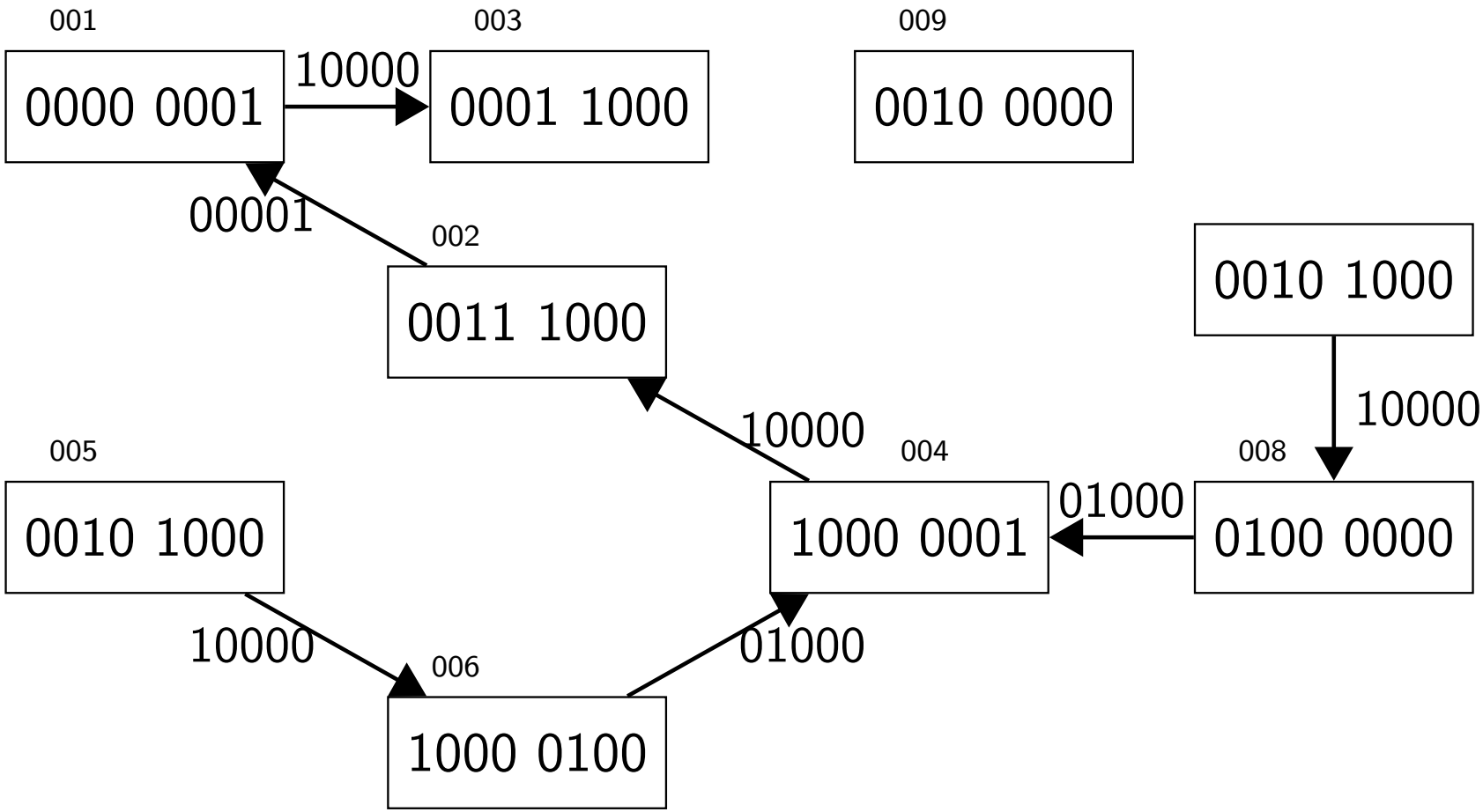
Label	adjList
⋮	⋮
y:Reese_Witherspoon	(BornOnDate, "1976-03-22"), (gender, "Female"), (title, "Actress"), (hasName, "Reese Witherspoon"), (BornIn,New_Orleans_LA)
⋮	⋮



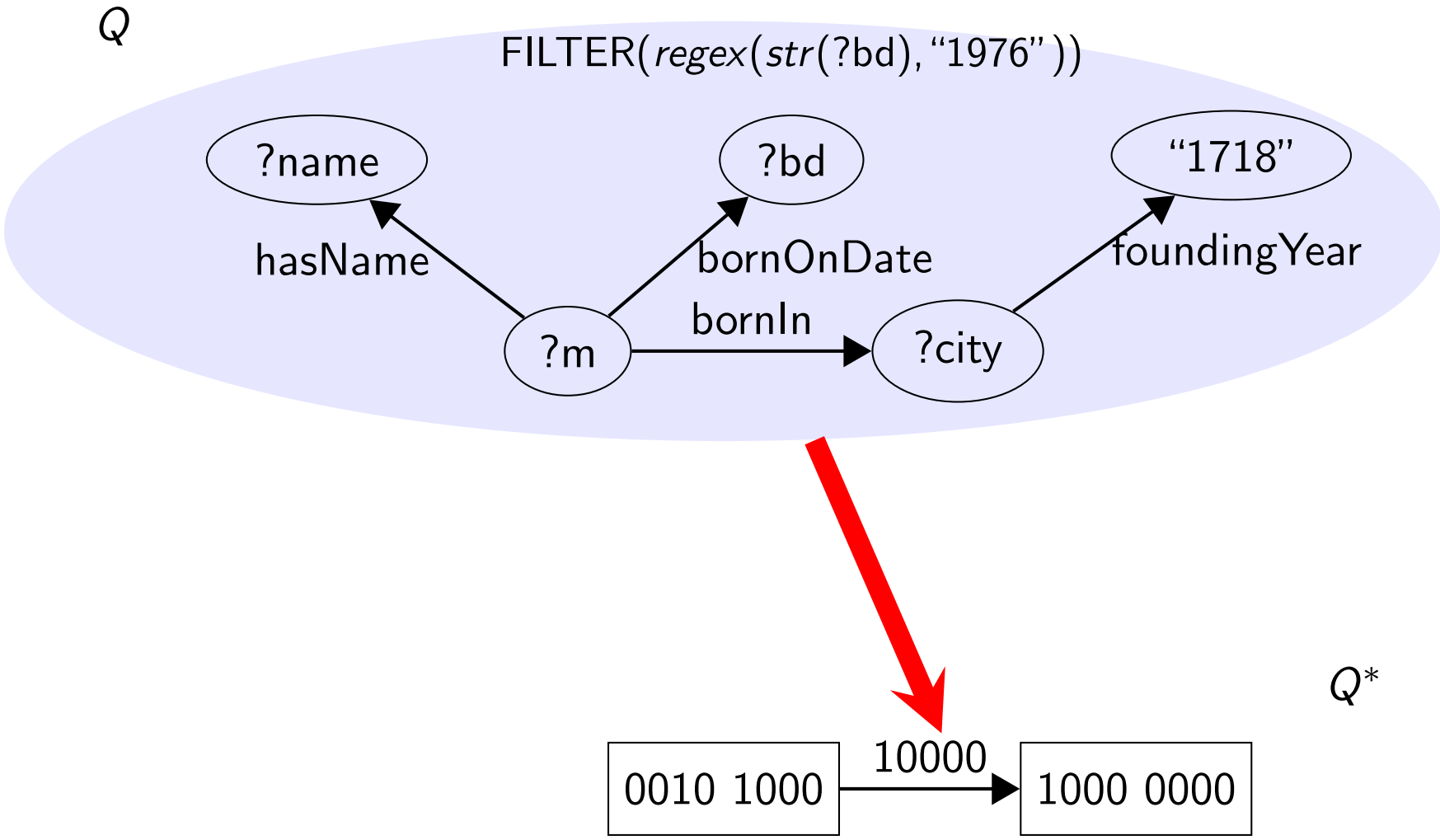
Encode all neighbors of a vertex into a bit string

vertex signature

2. Construct Data Signature Graph G^*

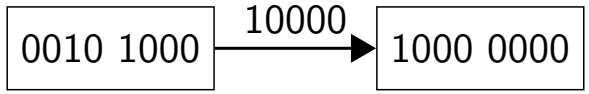


3. Encode Q to Get Signature Graph Q^*

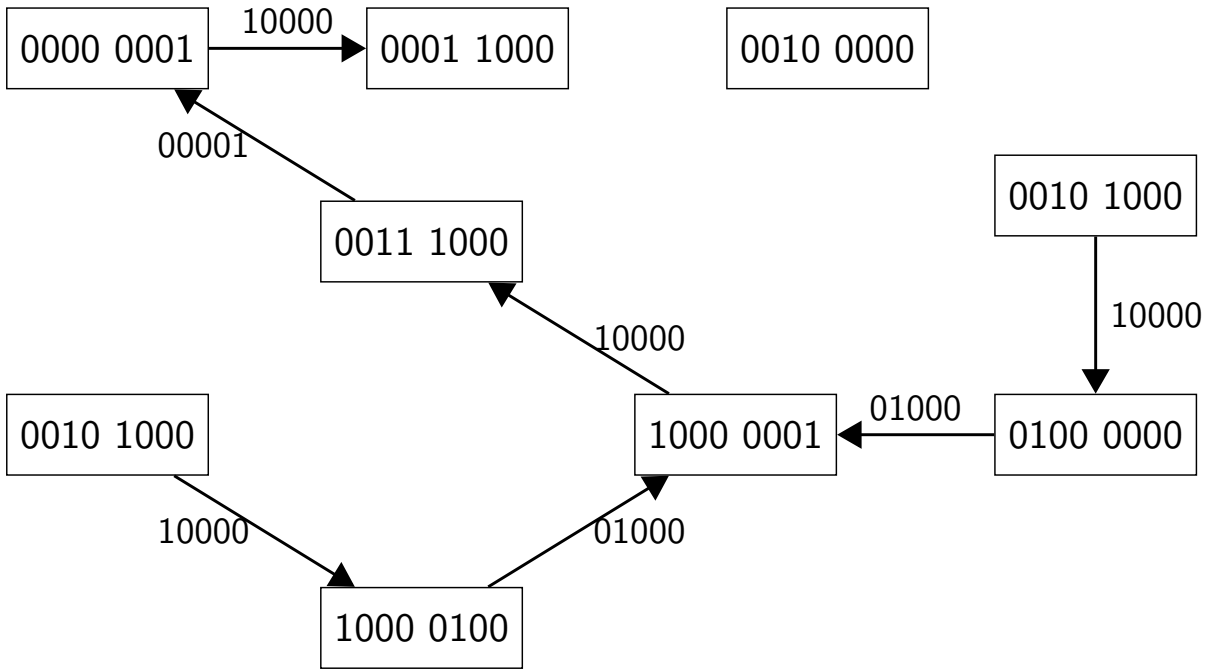


4. Filter-and-Evaluate

Query signature graph Q^*



Data signature graph G^*



Find matches of Q^* over signature graph G^* → Verify each match in RDF graph G

Outline

RDF Introduction

gStore: a graph-based SPARQL query engine

Answering SPARQL queries using graph pattern matching [Zou et al., PVLDB 2011, VLDB J 2014]

gAnswer: Natural Language Question Answering over RDF

A Graph Data Driven Approach [Zou et al., SIGMOD 2014]

gAnswer: Natural Language Question Answering Over Knowledge Graph–A Graph Data Driven Approach

- An Easy-to-Use Interface to Access Knowledge Graph
  - It is interesting to both academia and industry.
  - Interdisciplinary research between database and NLP (natural language processing) communities.

gAnswer: Natural Language Question Answering Over Knowledge Graph–A Graph Data Driven Approach

- An Easy-to-Use Interface to Access Knowledge Graph
  - It is interesting to both academia and industry.
  - Interdisciplinary research between database and NLP (natural language processing) communities.

Running Example

Question: Who was married to an actor that play in Philadelphia ?

Subject	**Property**	**Object**
Antonio_Banderas	type	actor
Antonio_Banderas	spouse	Melanie_Griffith
Antonio_Banderas	starring	Philadelphia_(film)
Philadelphia_(film)	type	film
Jonathan_Demme	director	film
Philadelphia	type	city
Aaron_McKie	bornIn	Philadelphia
James_Anderson	playForTeam	Philadelphia_76ers
Constantin_Stanislavski	create	An_Actor_Prepares
Philadelphia_76ers	type	Basketball_team
An_Actor_Prepares	type	Book

Running Example

Question: Who was married to an actor that play in Philadelphia ?

Subject	Property	Object
Antonio_Banderas	type	actor
Antonio_Banderas	spouse	Melanie_Griffith
Antonio_Banderas	starring	Philadelphia_(film)
Philadelphia_(film)	type	film
Jonathan_Demme	director	film
Philadelphia	type	city
Aaron_McKie	bornIn	Philadelphia
James_Anderson	playForTeam	Philadelphia_7
Constantin_Stanislavski	create	An_Actor_Prepa
Philadelphia_76ers	type	Basketball_team
An_Actor_Prepares	type	Book

Existing Solutions

Who was married to an actor that play in Philadelphia ?

Translate NL Question to structured queries

```
SELECT ?y
WHERE {
?x starring Philadelphia_(film).
?x type actor.
?x spouse ?y . }
```

Query Processing

Melanie Griffith

Existing Solutions

Who was married to an actor that play in Philadelphia ?

```
SELECT ?y
WHERE {
?x starring Philadelphia_(film).
?x type actor.
?x spouse ?y . }
```

Translate NL Question to structured queries

Philadelpha

↓

Philadelpha_(film)

↓

Philadelpha_76ers

Query Processing

Melanie Griffith

Existing Solutions

Who was married to an actor that play in Philadelphia?

```
SELECT ?y
WHERE {
?x starring Philadelphia_(film).
?x type actor.
?x spouse ?y . }
```

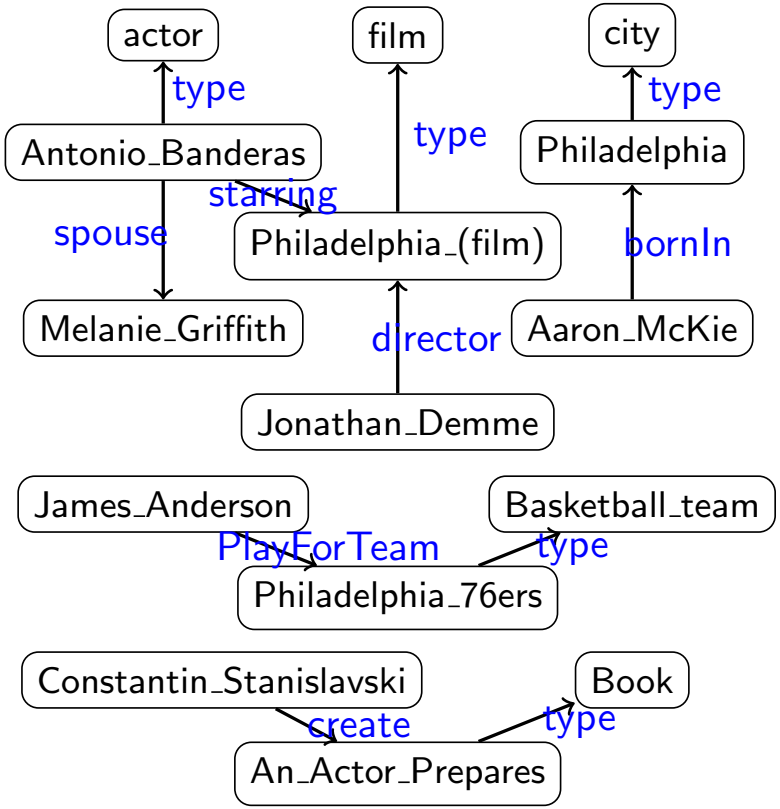
Translate NL Question to structured queries

playForTeam
↓
starring
↓
director

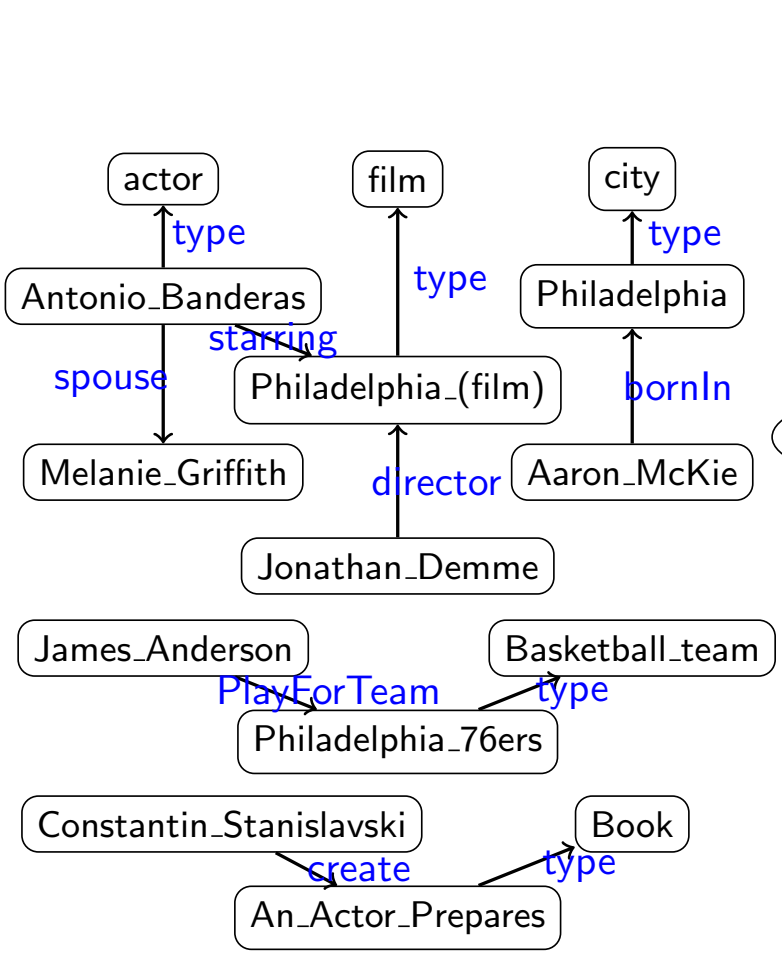
Query Processing

Melanie Griffith

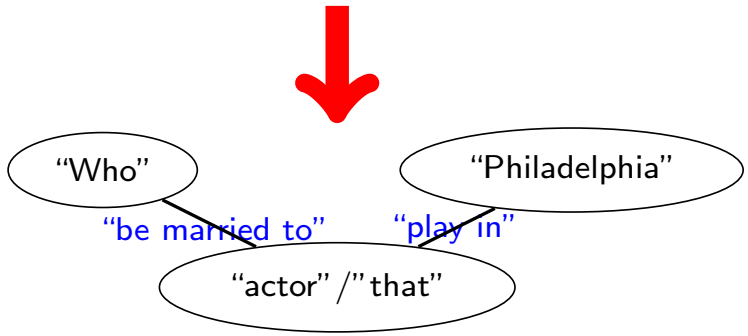
Our Method: Motivation–Data Driven



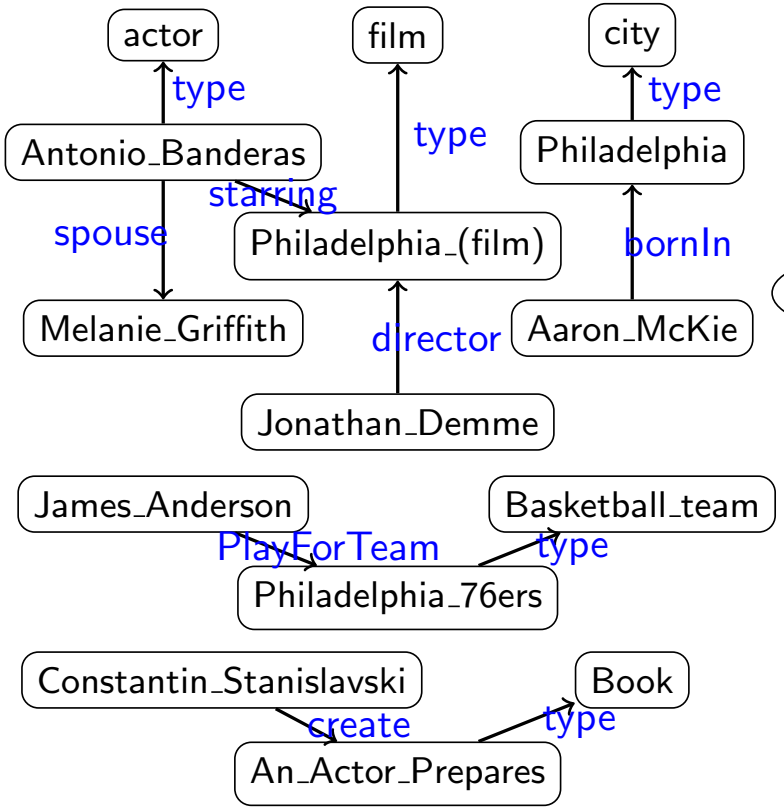
Our Method: Motivation–Data Driven

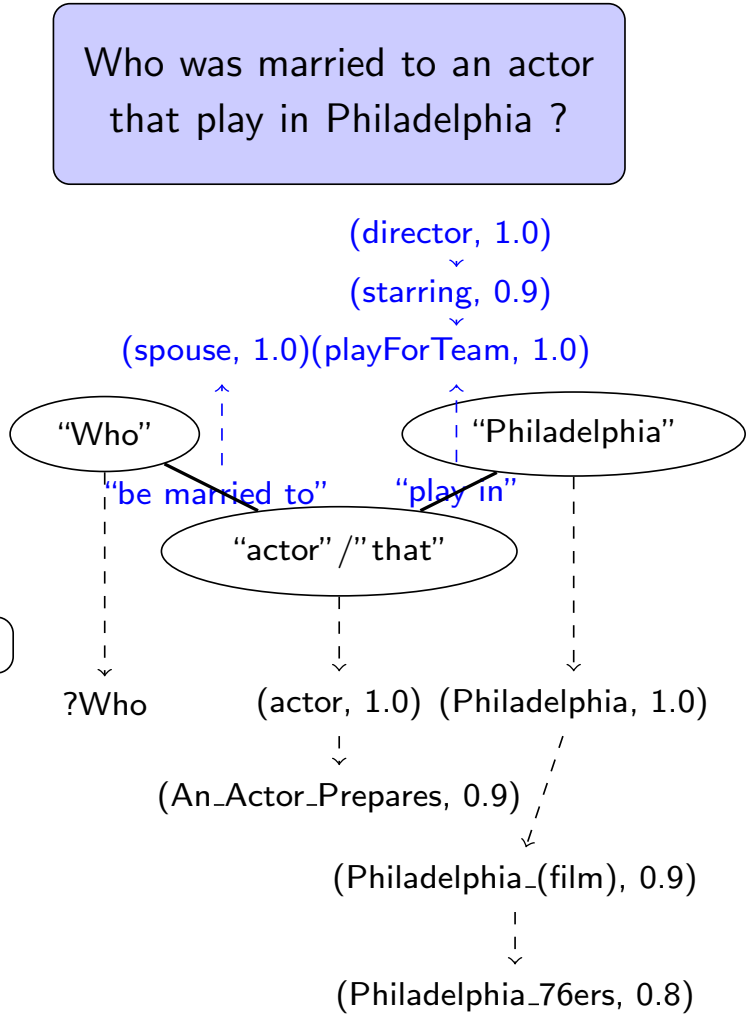


Who was married to an actor that play in Philadelphia ?

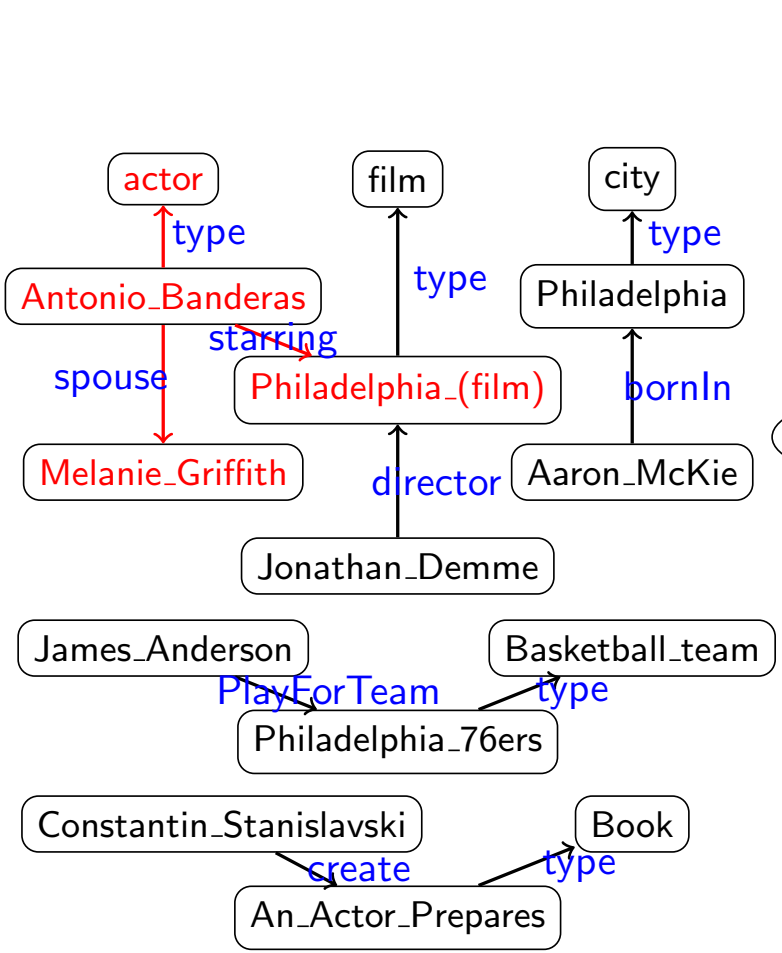


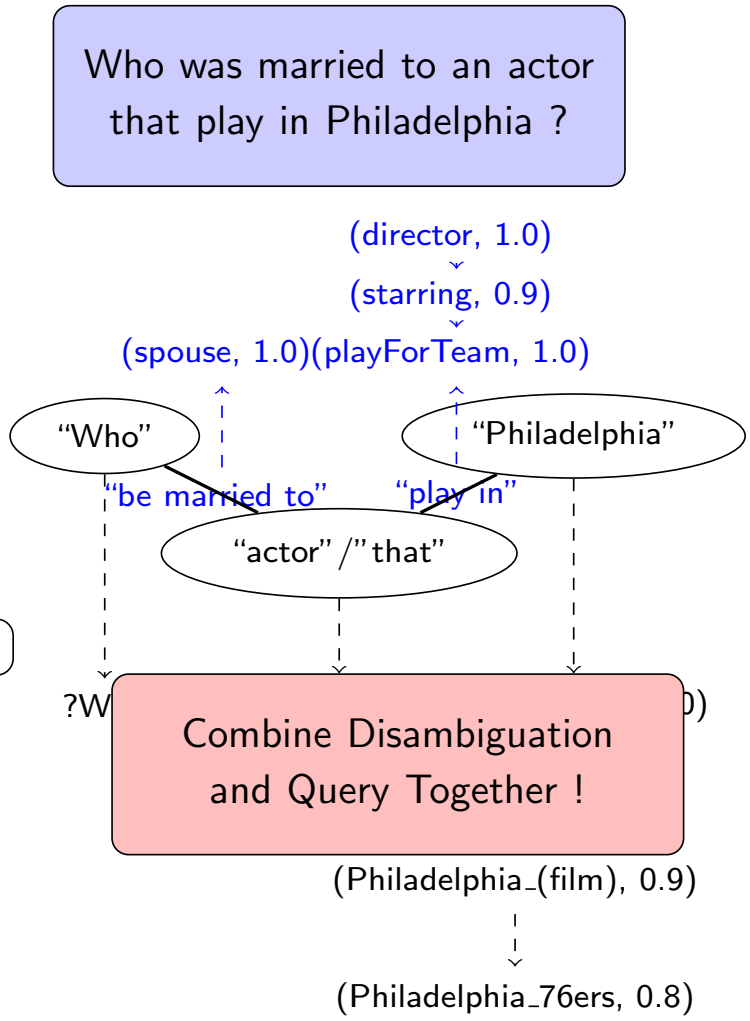
Our Method: Motivation–Data Driven



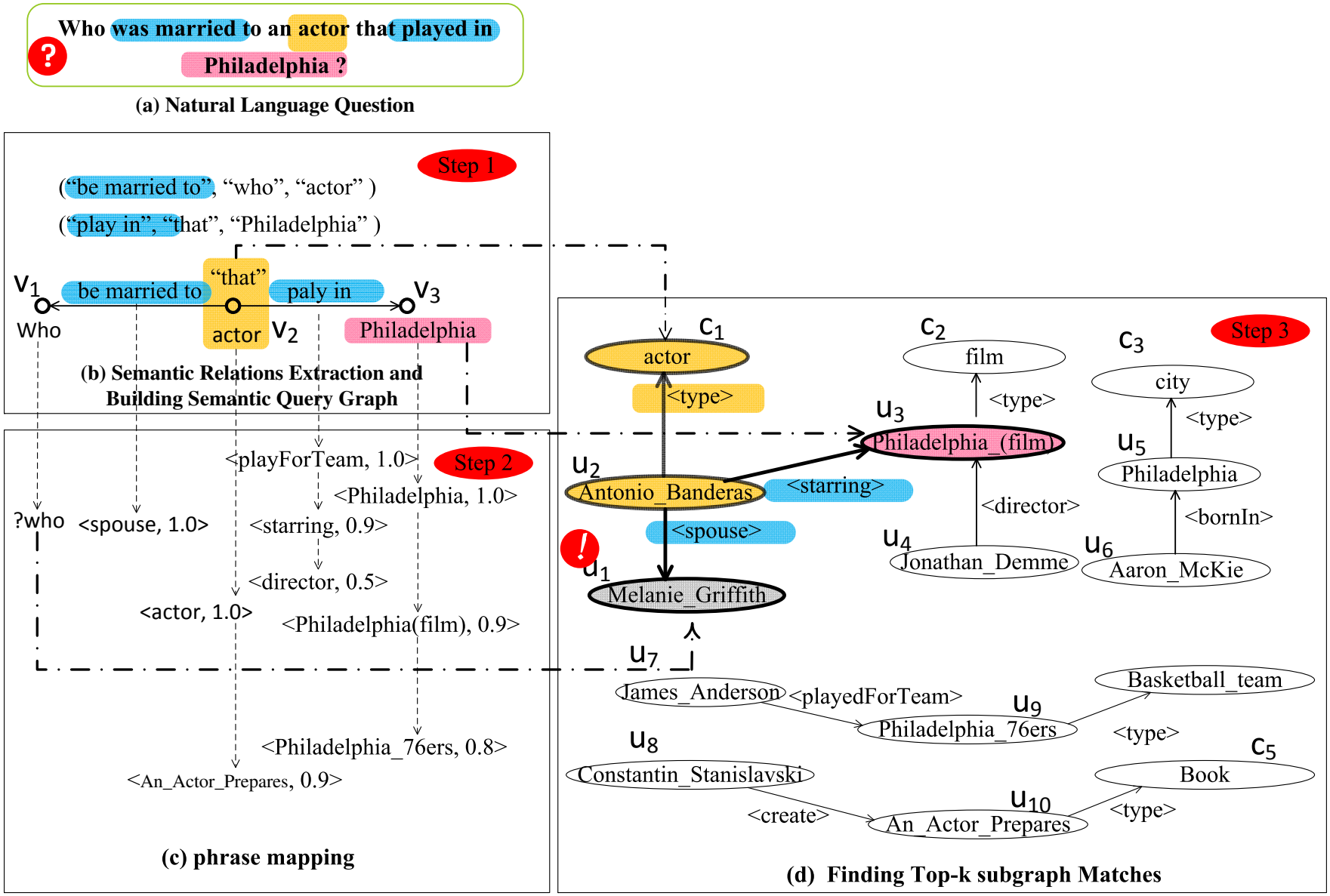


Our Method: Motivation–Data Driven



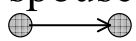
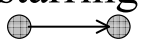
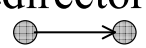


Our Method: Framework



Our Method: Offline

- **TASK**: Building a paraphrase dictionary to map relation phrases to predicates in RDF graph.
- **METHOD**: Data Driven Approach

Relation Phrases	Predicates or Predicate Paths	Confidence Probability
"be married to"	<spouse> 	1.0
"play in"	<starring> 	0.9
"play in"	<director> 	0.5
"uncle of"	<hasChild> <hasChild> 	0.8
… …	… …	… …

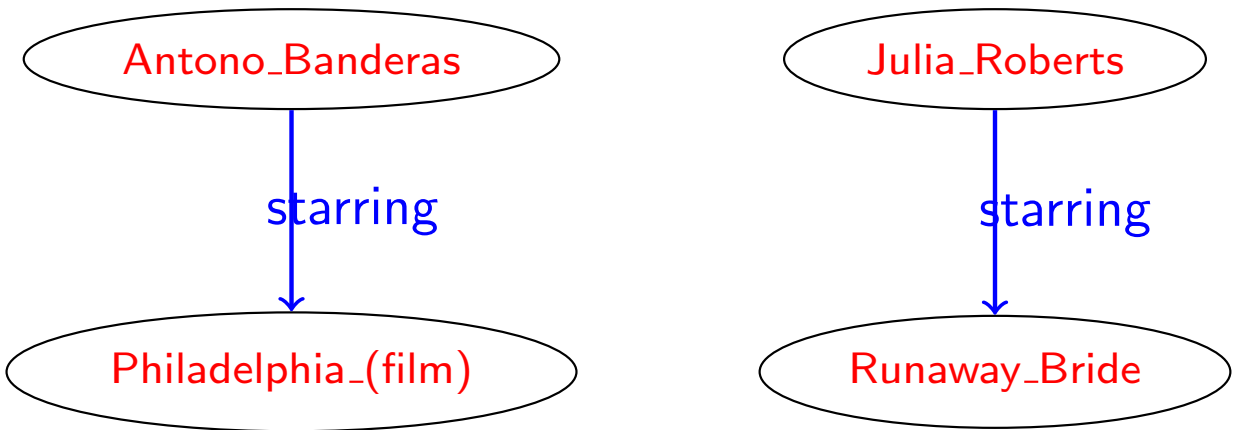
Our Method: Offline

- **INPUT**: Relation Phrases and Supporting Entity Pairs.
- **OUTPUT**: Possible Mapping Predicates / Predicate Paths

Relation Phrase	**Supporting Entity Pairs**
"play in"	(⟨ Antonio_Banderas ⟩, ⟨ Philadelphia(film) ⟩), (⟨ Julia_Roberts ⟩, ⟨ Runaway_Bride ⟩),.....
"uncle of"	(⟨ Ted_Kennedy ⟩, ⟨ John_F._Kennedy,_Jr. ⟩) (⟨ Peter_Corr ⟩, ⟨ Jim_Corr ⟩),.....

Our Method: Offline

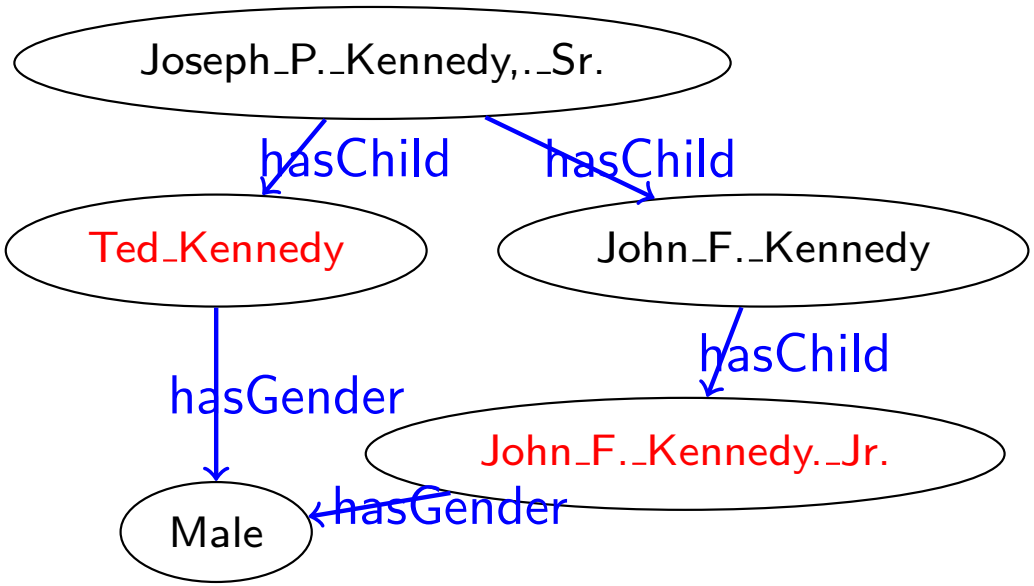
- **INPUT**: Relation Phrases and Supporting Entity Pairs.
- **OUTPUT**: Possible Mapping Predicates / Predicate Paths



Relation Phrase	Supporting Entity Pairs
"play in"	(⟨ Antonio_Banderas ⟩, ⟨ Philadelphia(film) ⟩) , (⟨ Julia_Roberts ⟩, ⟨ Runaway_Bride ⟩),.....
"uncle of"	(⟨ Ted_Kennedy ⟩, ⟨ John_F._Kennedy,_Jr. ⟩) (⟨ Peter_Corr ⟩, ⟨ Jim_Corr ⟩),.....

Our Method: Offline

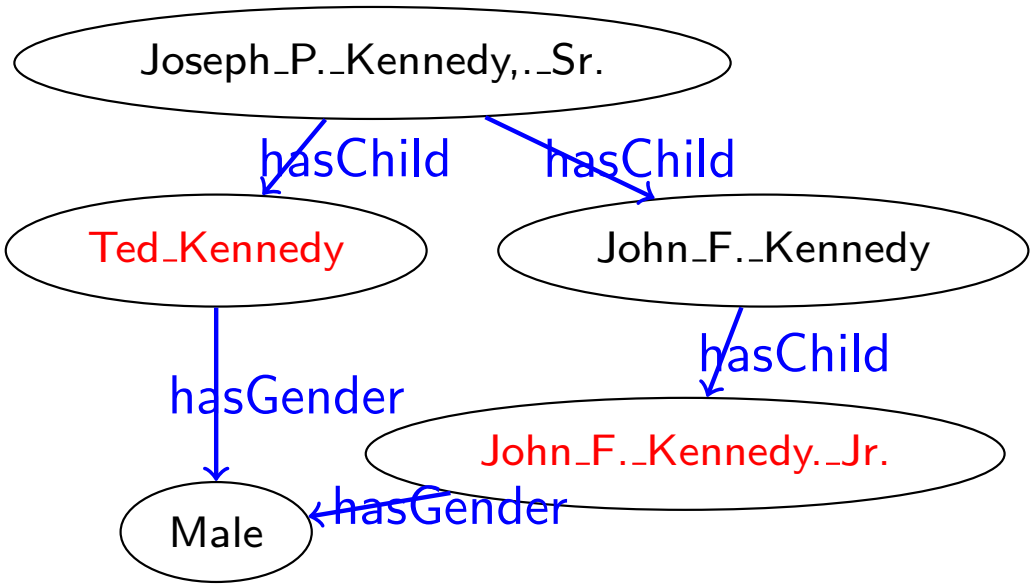
- **INPUT**: Relation Phrases and Supporting Entity Pairs.
- **OUTPUT**: Possible Mapping Predicates / Predicate Paths

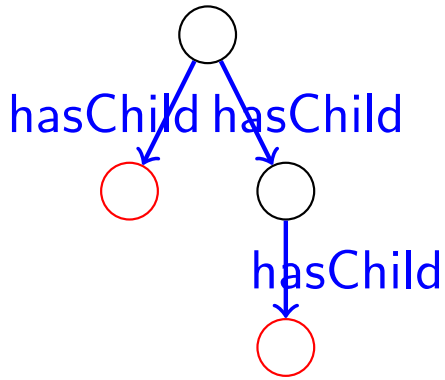


Relation Phrase	Supporting Entity Pairs
"play in"	(⟨ Antonio_Banderas ⟩, ⟨ Philadelphia(film) ⟩) , (⟨ Julia_Roberts ⟩, ⟨ Runaway_Bride ⟩),.....
"uncle of"	(⟨ Ted_Kennedy ⟩, ⟨ John_F._Kennedy,_Jr. ⟩) (⟨ Peter_Corr ⟩, ⟨ Jim_Corr ⟩),.....

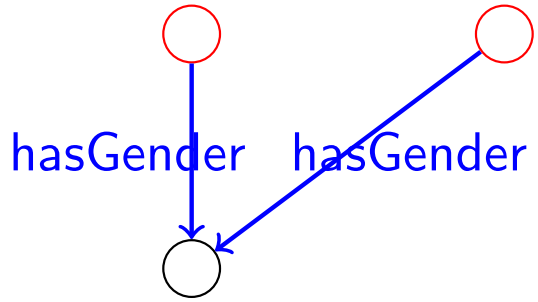
Our Method: Offline

- **INPUT**: Relation Phrases and Supporting Entity Pairs.
- **OUTPUT**: Possible Mapping Predicates / Predicate Paths



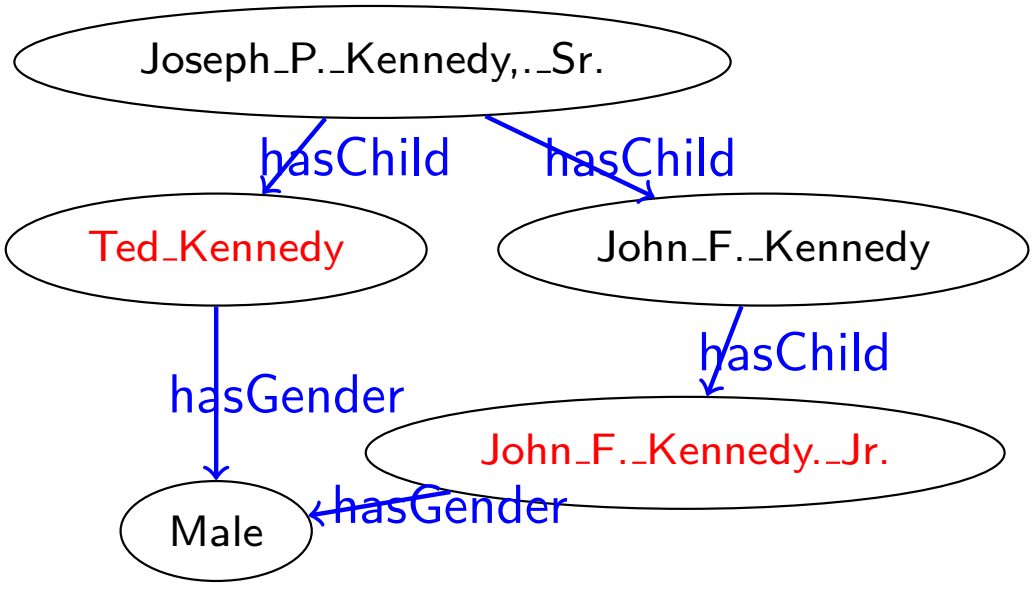


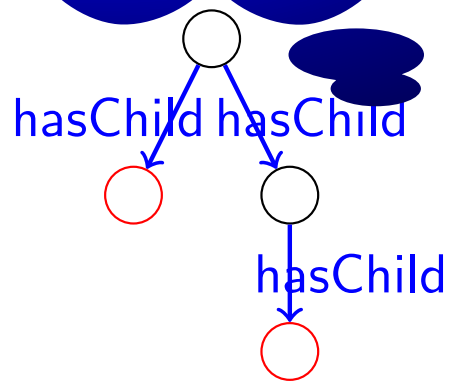
Relation Phrase	Supporting Entity Pairs
"play in"	(⟨ Antonio_Banderas ⟩, ⟨ Philadelphia(film) ⟩) , (⟨ Julia_Roberts ⟩, ⟨ Runaway_Bride ⟩),.....
"uncle of"	(⟨ Ted_Kennedy ⟩, ⟨ John_F._Kennedy,_Jr. ⟩) (⟨ Peter_Corr ⟩, ⟨ Jim_Corr ⟩),.....



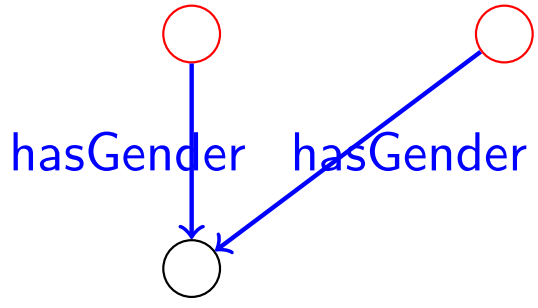
Our Method: Offline

- **INPUT**: Relation Phrases and Supp
- **OUTPUT**: Possible Mapping Predica





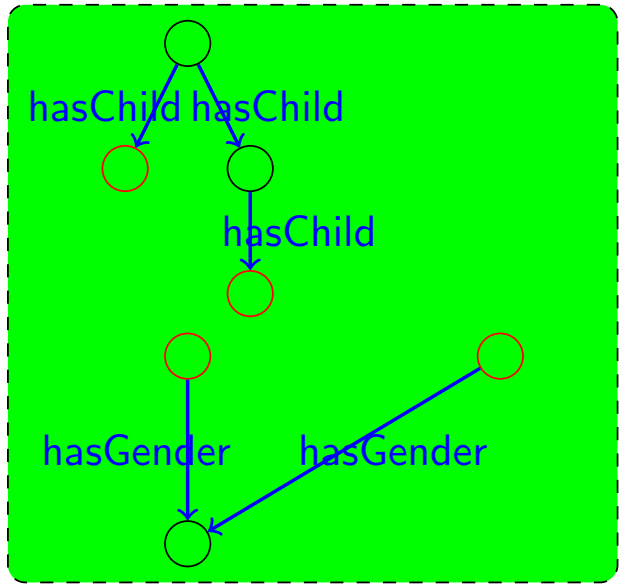
Relation Phrase	Supporting Entity Pairs
"play in"	(⟨ Antonio_Banderas ⟩, ⟨ Philadelphia(film) ⟩) , (⟨ Julia_Roberts ⟩, ⟨ Runaway_Bride ⟩),.....
"uncle of"	(⟨ Ted_Kennedy ⟩, ⟨ John_F._Kennedy,_Jr. ⟩) (⟨ Peter_Corr ⟩, ⟨ Jim_Corr ⟩),.....



Our Method: Offline

relation phrase: rel_i

"uncle of"

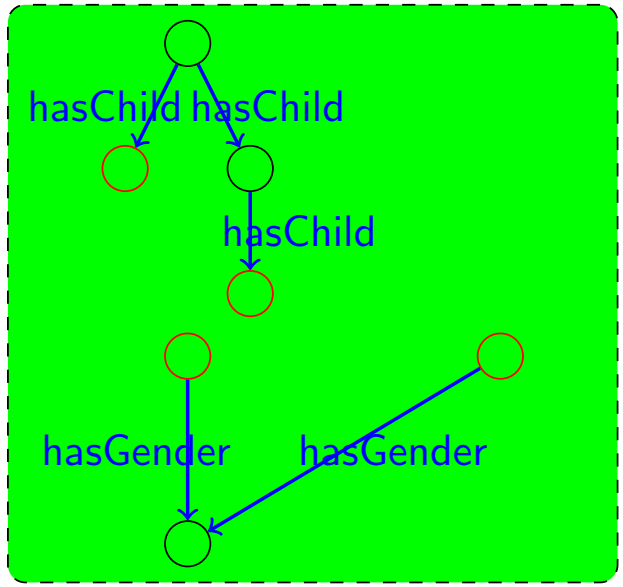


virtual document

Our Method: Offline

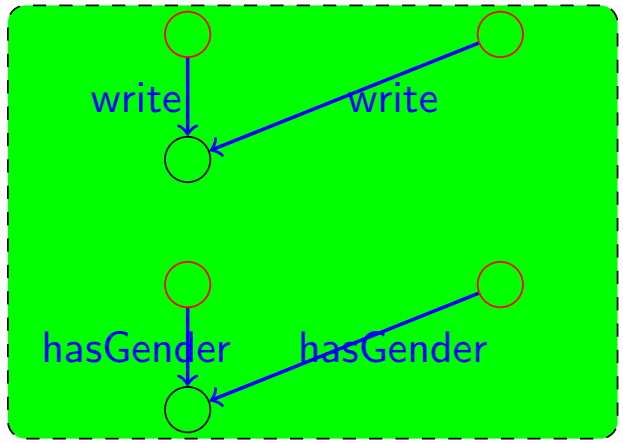
relation phrase: rel_i

"uncle of"



virtual document

"co-author with"

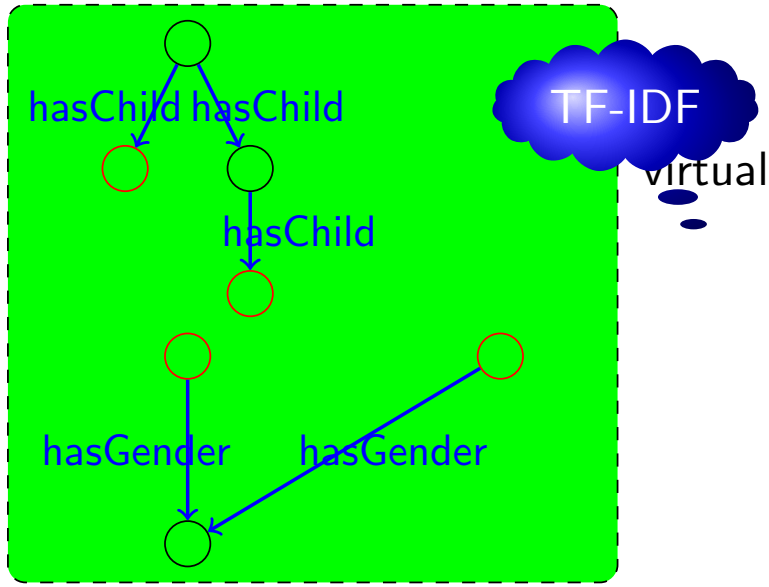


virtual document

Our Method: Offline

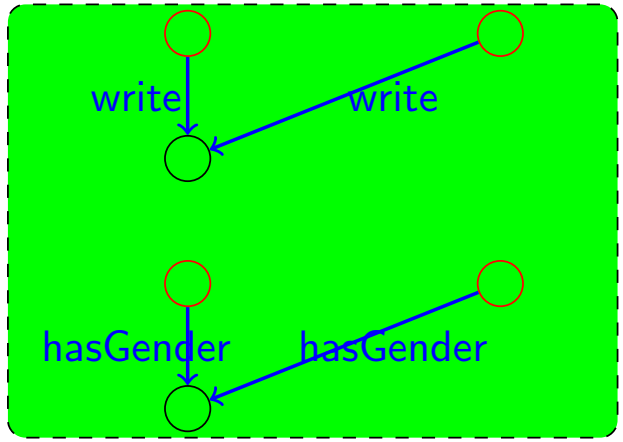
relation phrase: rel_i

"uncle of"



virtual document

"co-author with"



virtual document

Our Method: Offline

- **INPUT**: Relation Phrases and Supporting Entity Pairs.
- **OUTPUT**: Possible Mapping Predicates / Predicate Paths

Definition

Given a predicate path L , the *tf-value* of L in $PS(rel_i)$ is defined as follows:

$$tf(L, PS(rel_i)) = |\{Path(v_i^j, v_i'^j) | L \in Path(v_i^j, v_i'^j)\}|$$

The *idf-value* of L over the whole relation phrase dictionary $T = \{rel_1, ..., rel_n\}$ is defined as follows:

$$idf(L, T) = \log \frac{|T|}{|\{rel_i \in T | L \in PS(rel_i)\}| + 1}$$

The *tf-idf value* of L is defined as follows:

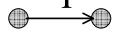
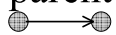
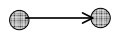
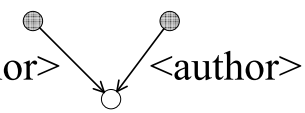
$$\text{tf}-\text{idf}(L, PS(rel_i), T) = tf(L, PS(rel_i)) \times idf(L, T)$$

We define the confidence probability of mapping relation phrase *rel* to predicate or predicate path L as follows.

$$\delta(rel, L) = \text{tf}-\text{idf}(L, PS(rel_i), T) \quad (1)$$

Our Method: Offline

- **INPUT**: Relation Phrases and Supporting Entity Pairs.
- **OUTPUT**: Possible Mapping Predicates / Predicate Paths

Relation Phrases	**Predicates \Predicate Paths**	**Confidence Probability**
"was married to"	<spouse>	1.00
"was born in"	<birthplace> 	1.00
"mother of"	<parent> 	0.95
"are located in"	<locatedInArea> 	0.98
"is fed by"	<inflow> 	1.00
"open in"	<locationCity> 	1.00
"is coauthor of"	<author> 	1.00

Our Method: Offline

- **INPUT**: Relation Phrases and Supporting Entity Pairs.
- **OUTPUT**: Possible Mapping Predicates / Predicate Paths

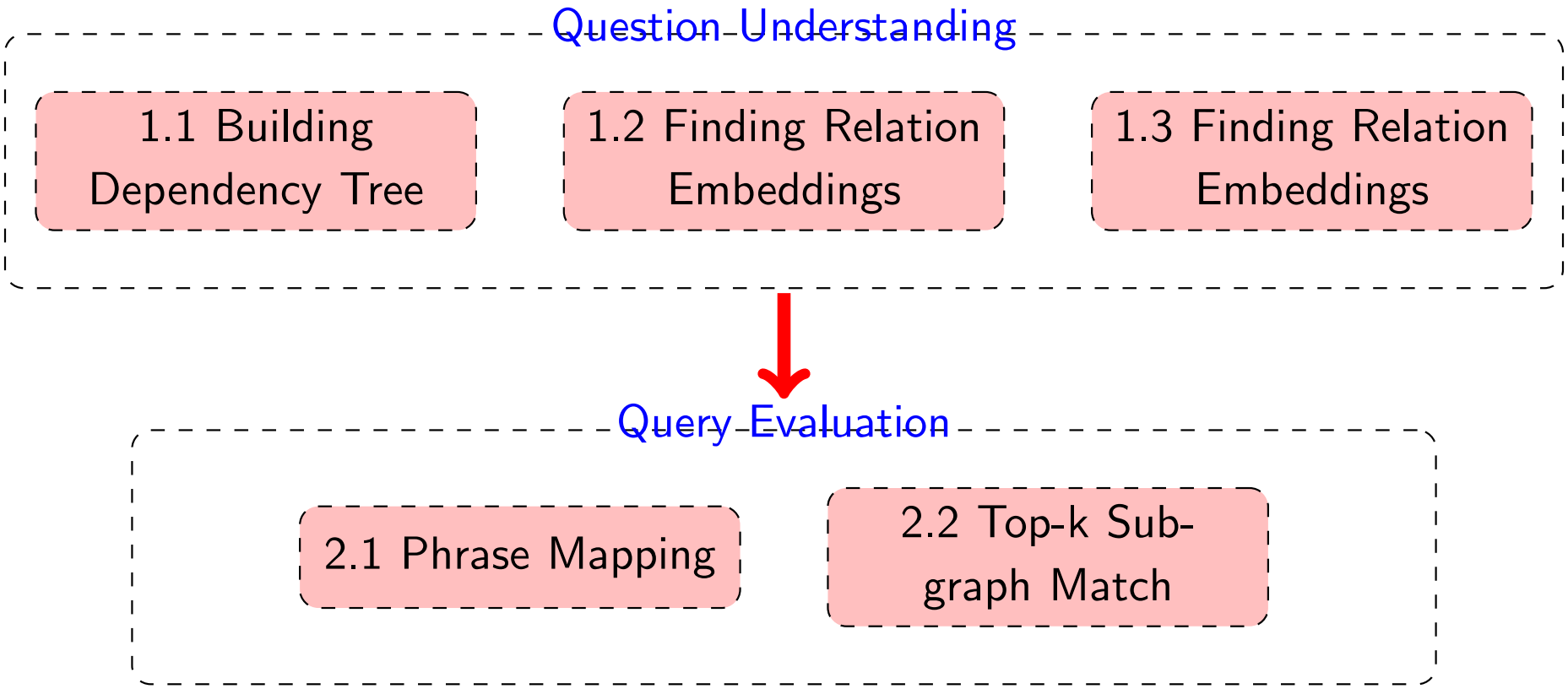
Precision@1	**Length-1**	**Length-2**	**Length-3**	**Length-4**
wordnet-wikipedia	61.10%	24.31%	11.18%	7.32%
freebase-wikipedia	58.56%	19.82%	9.23%	8.17%
Precision@3				
wordnet-wikipedia	72.35%	27.87%	16.73%	10.26%
freebase-wikipedia	67.72%	21.56%	10.37%	11.35%

Our Method: Offline

- **INPUT**: Relation Phrases and Supporting Entity Pairs.
- **OUTPUT**: Possible Mapping Predicates / Predicate Paths

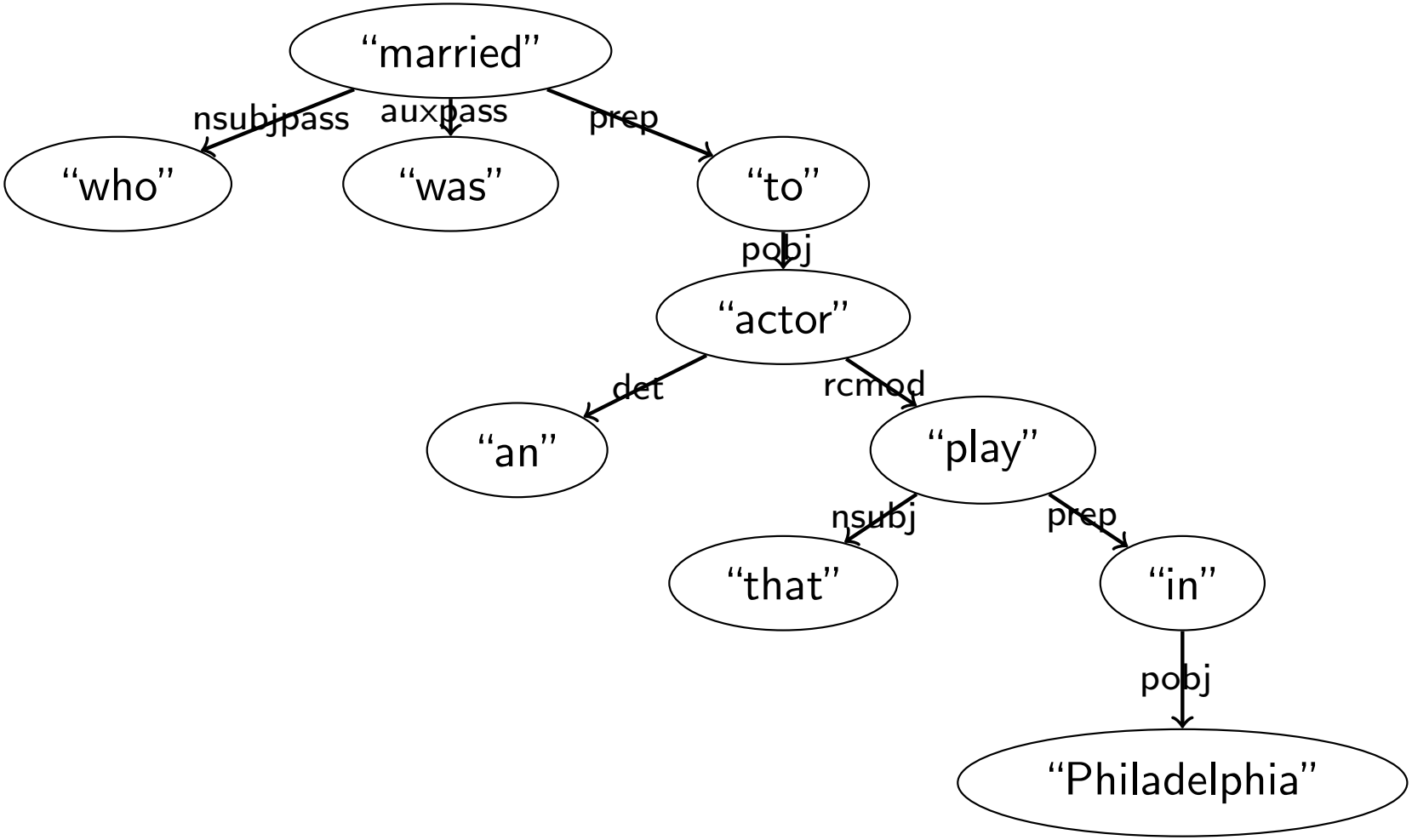
Precision@1	**Length-1**	**Length-2**	**Length-3**	**Length-4**
wordnet-wikipedia	61.10%	24.31%	11.18%	7.32%
freebase-wikipedia	58.56%	19.82%	9.23%	8.17%
Precision@3				
wordnet-wikipedia	72.35%	27.87%		
freebase-wikipedia	67.72%	21.56%		

Online: Framework

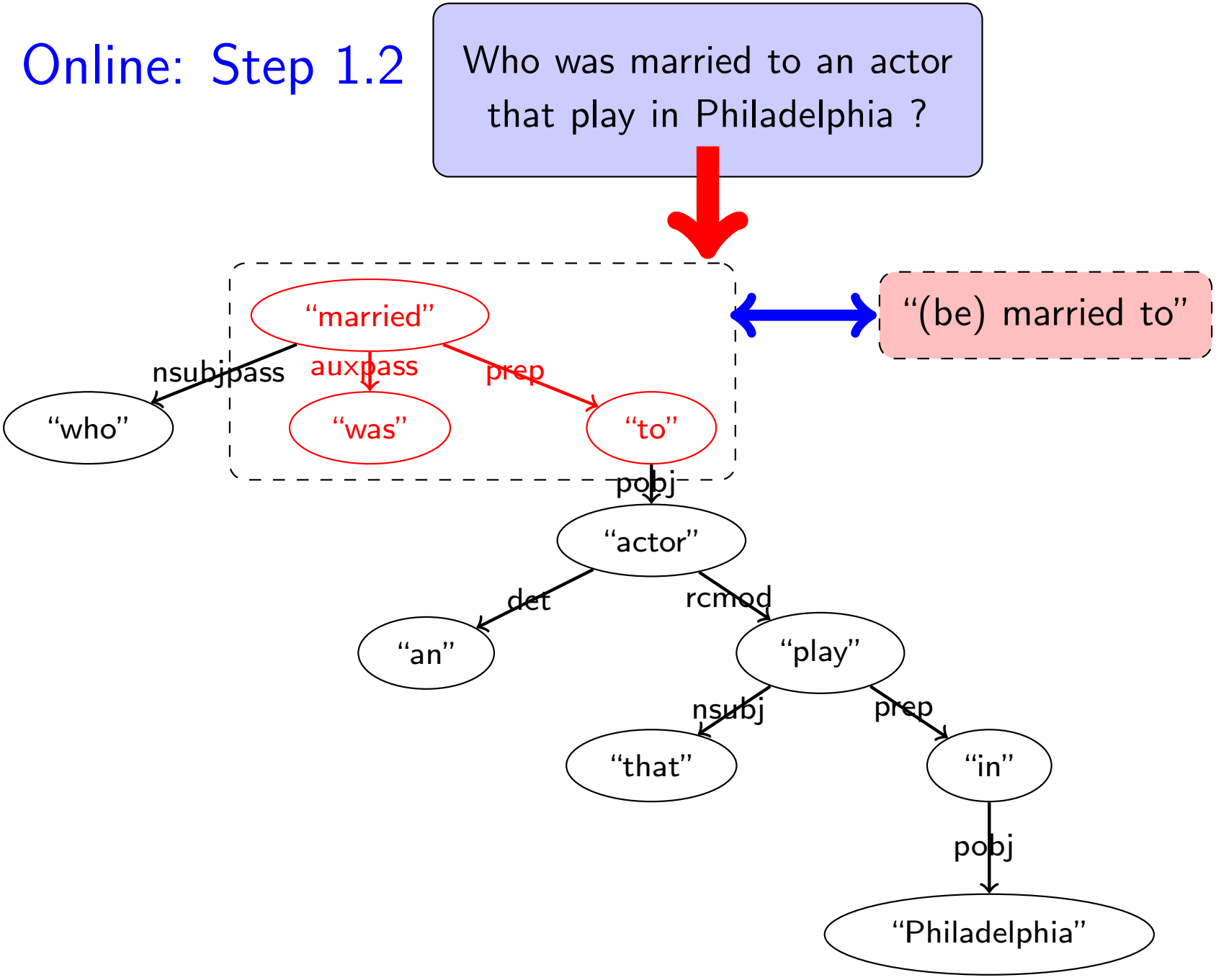


Online: Step 1.1

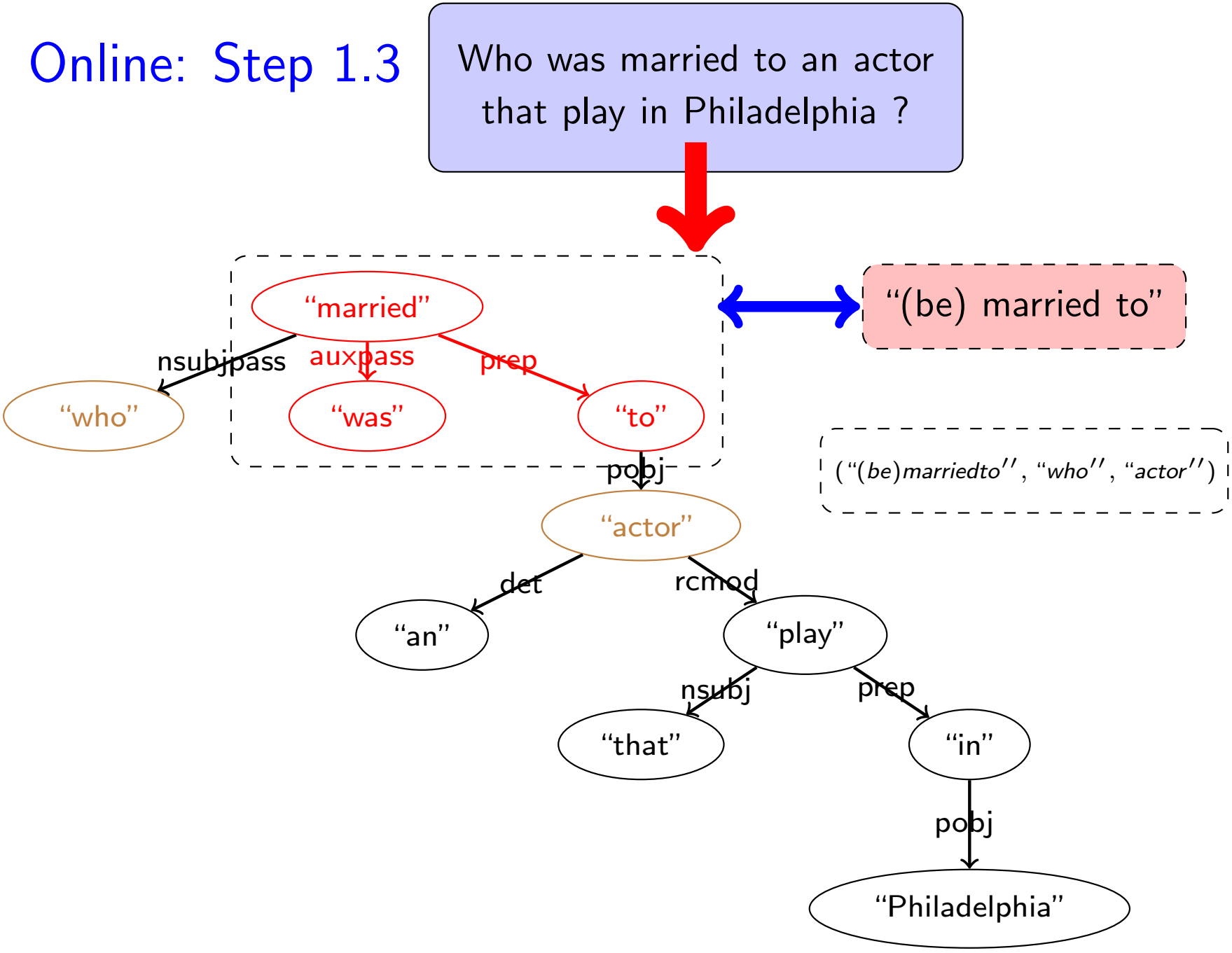
Who was married to an actor that play in Philadelphia ?



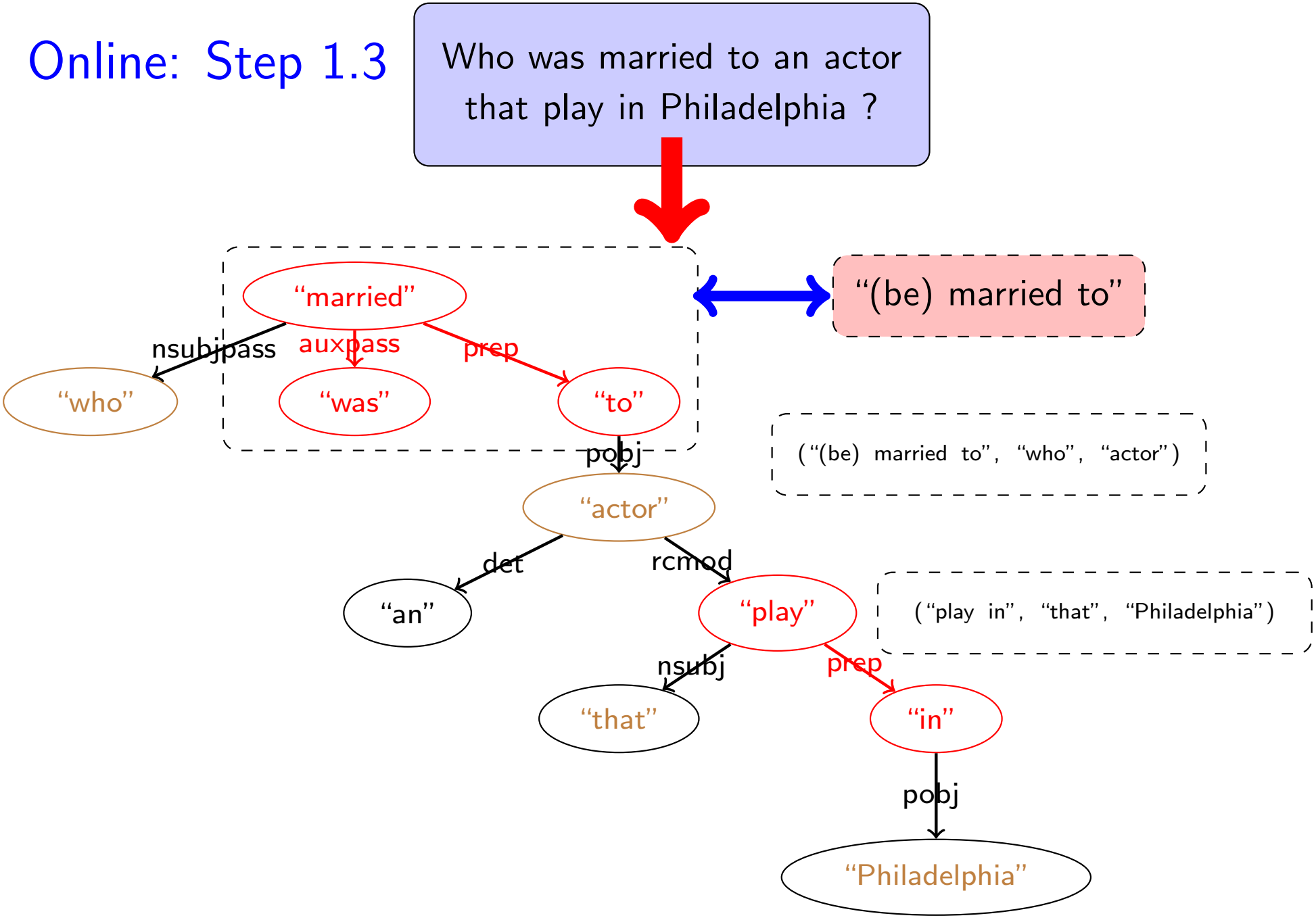
Online: Step 1.2



Online: Step 1.3

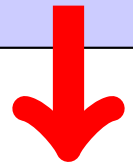


Online: Step 1.3



Online: Step 1.4

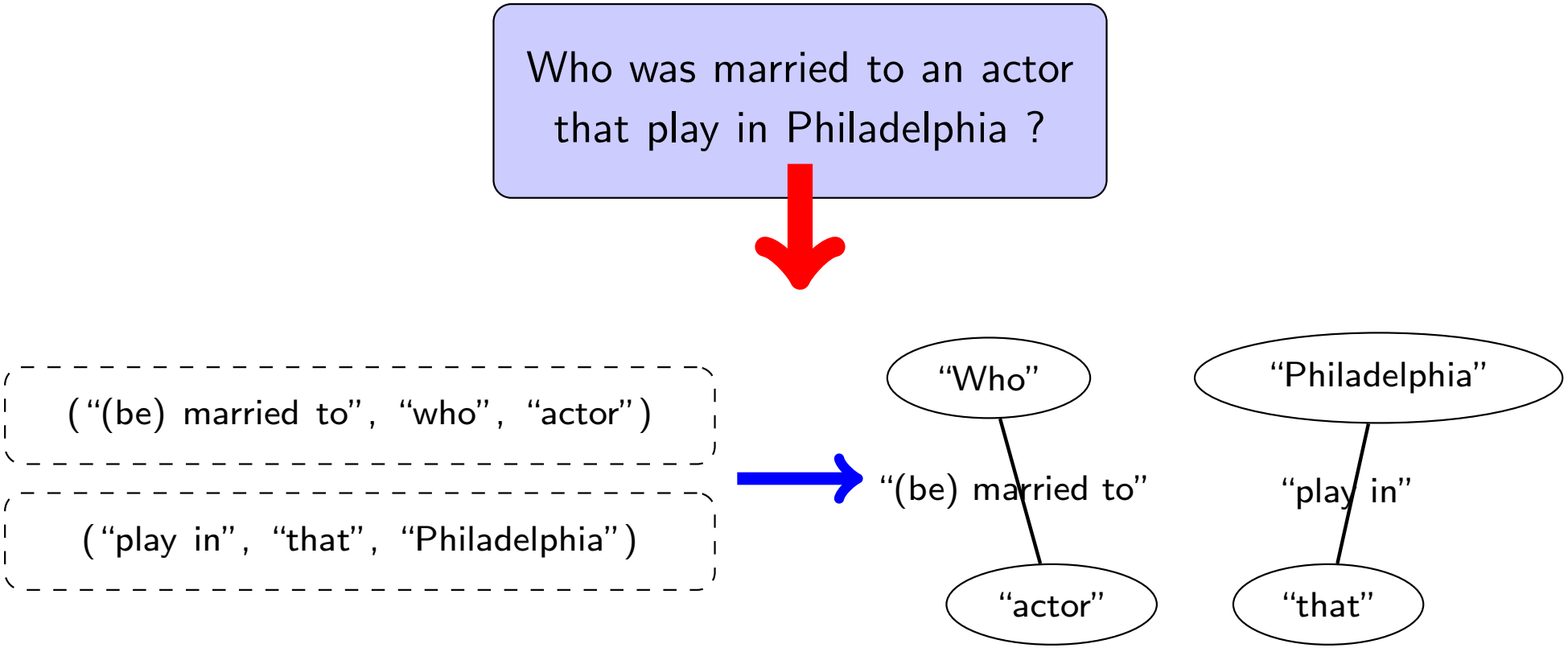
Who was married to an actor that play in Philadelphia ?



("(be) married to", "who", "actor")

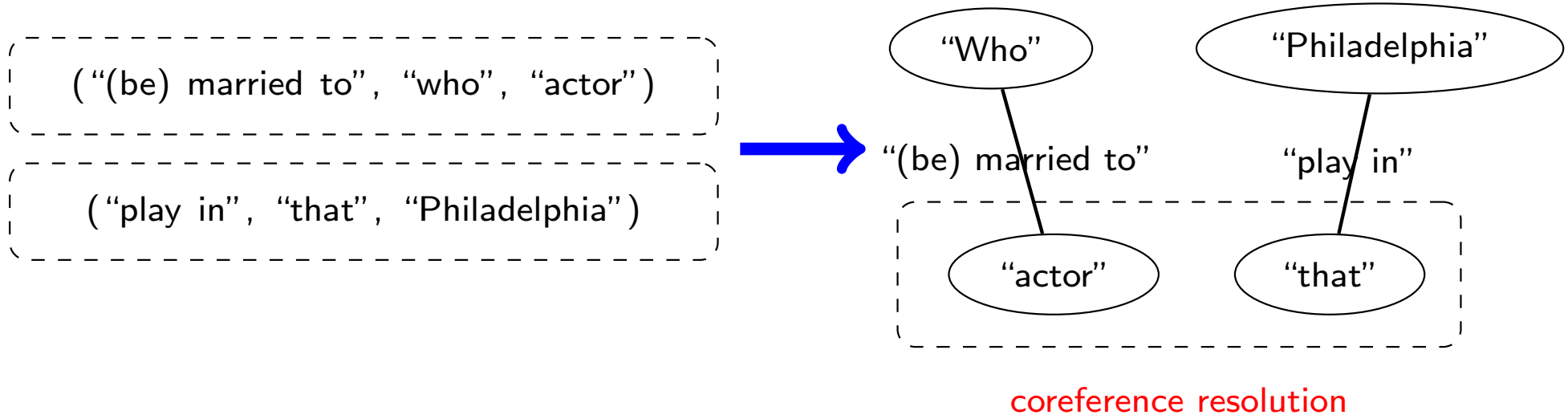
("play in", "that", "Philadelphia")

Online: Step 1.4



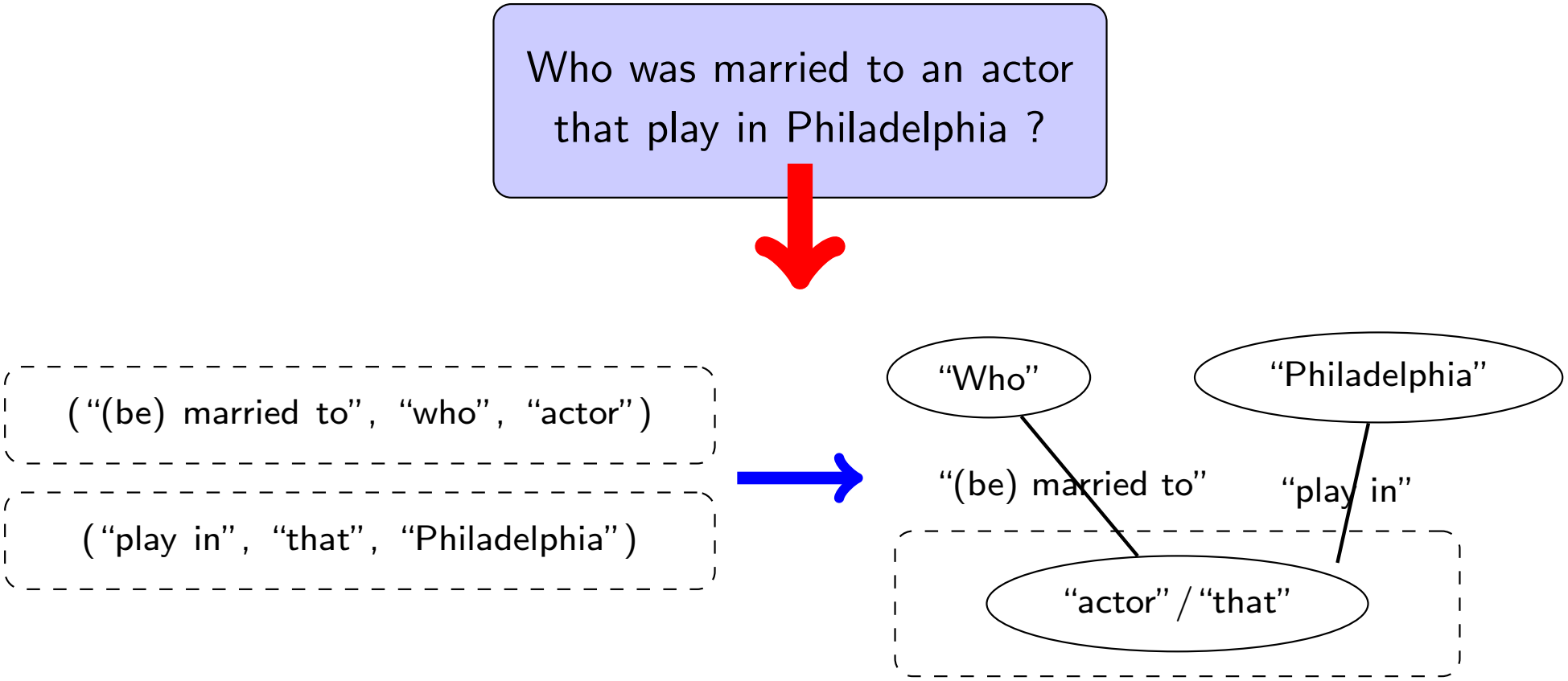
Online: Step 1.4

Who was married to an actor that play in Philadelphia ?

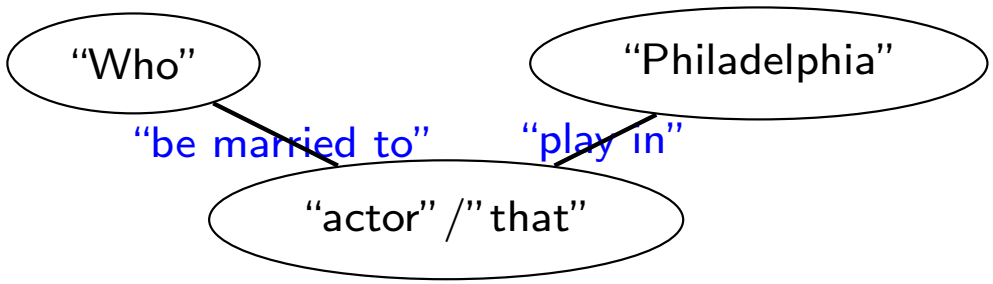


coreference resolution

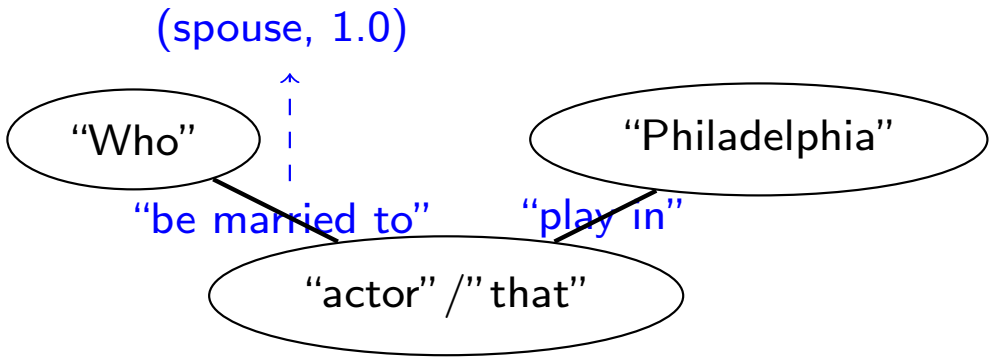
Online: Step 1.4



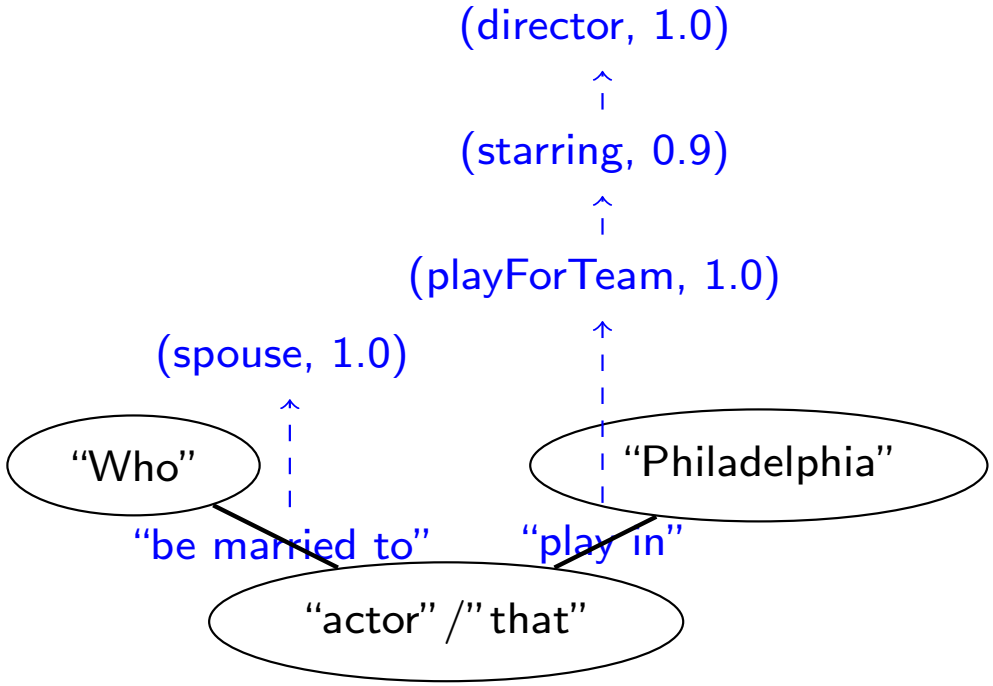
Online: Step 2.1 Mapping Edge



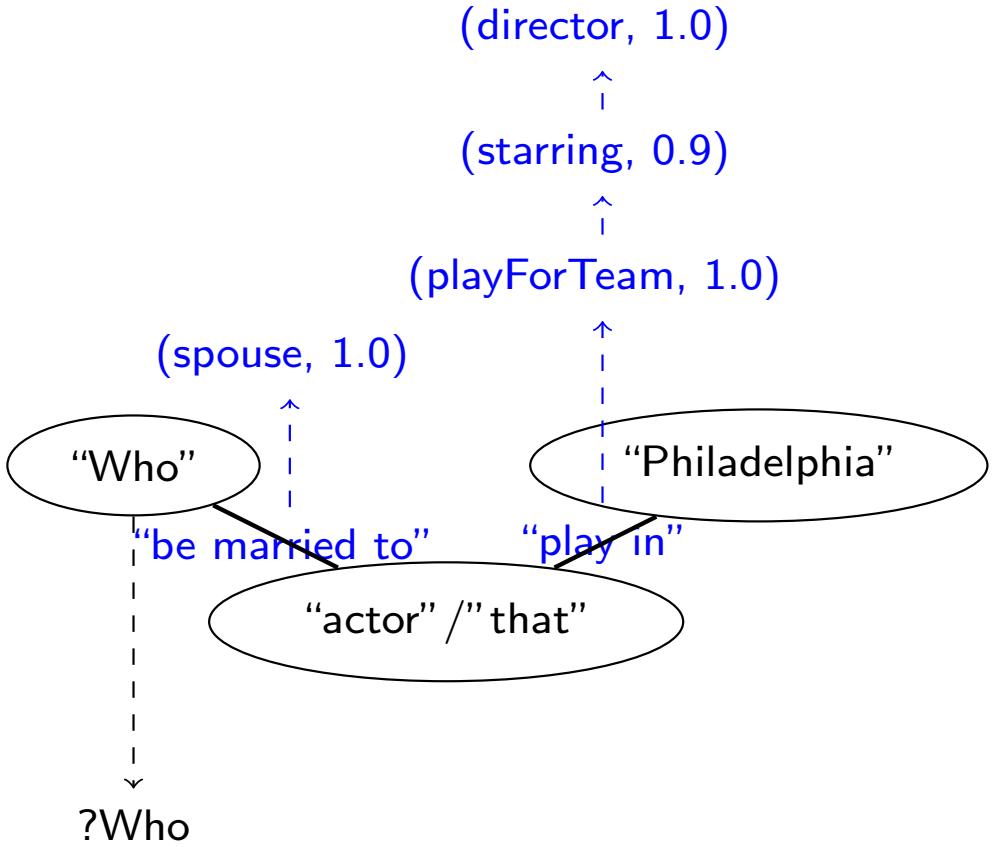
Online: Step 2.1 Mapping Edge



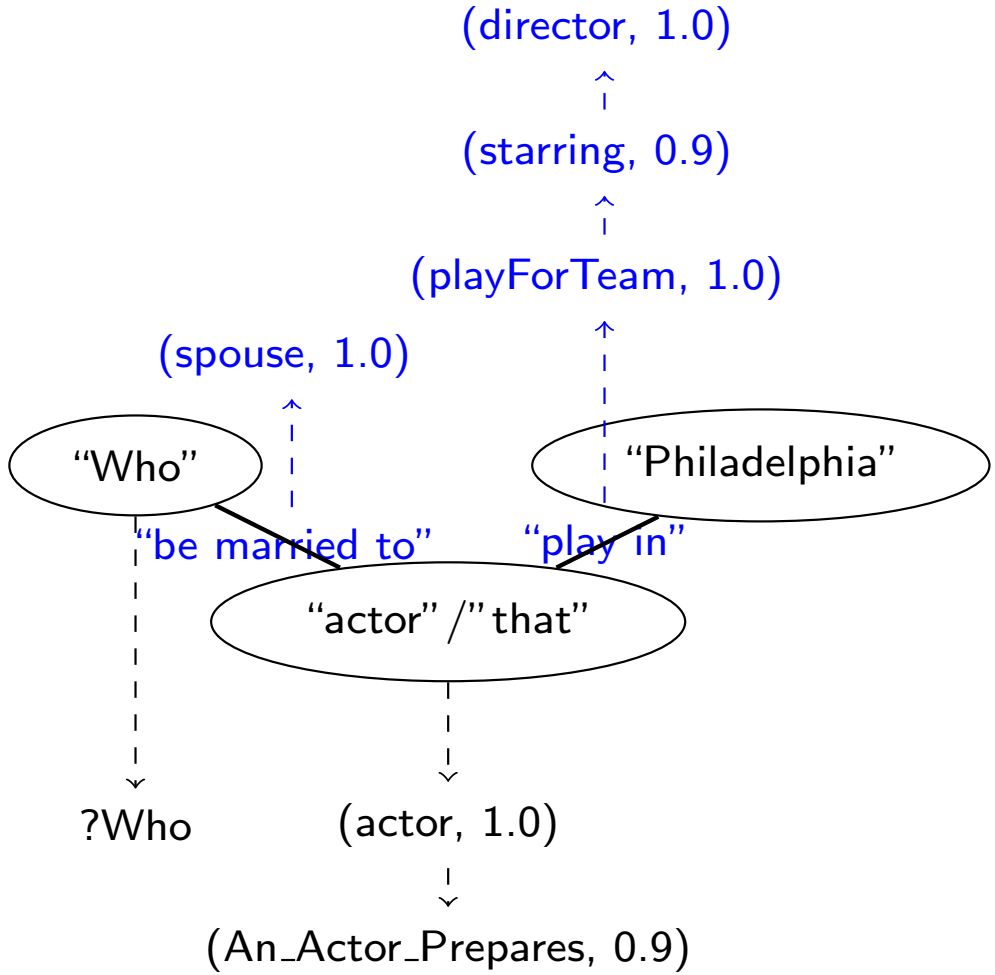
Online: Step 2.1 Mapping Edge



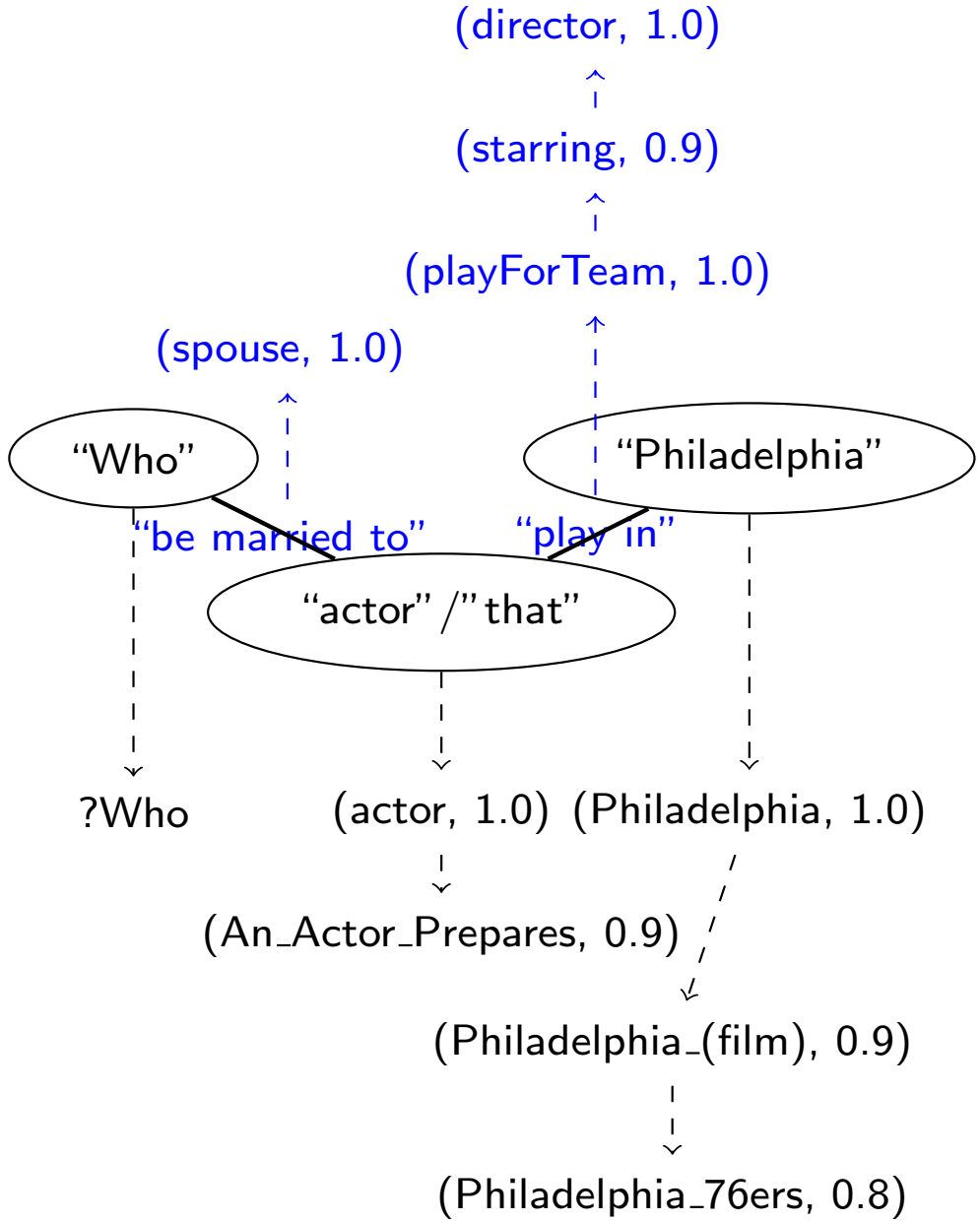
Online: Step 2.2 Mapping Vertices



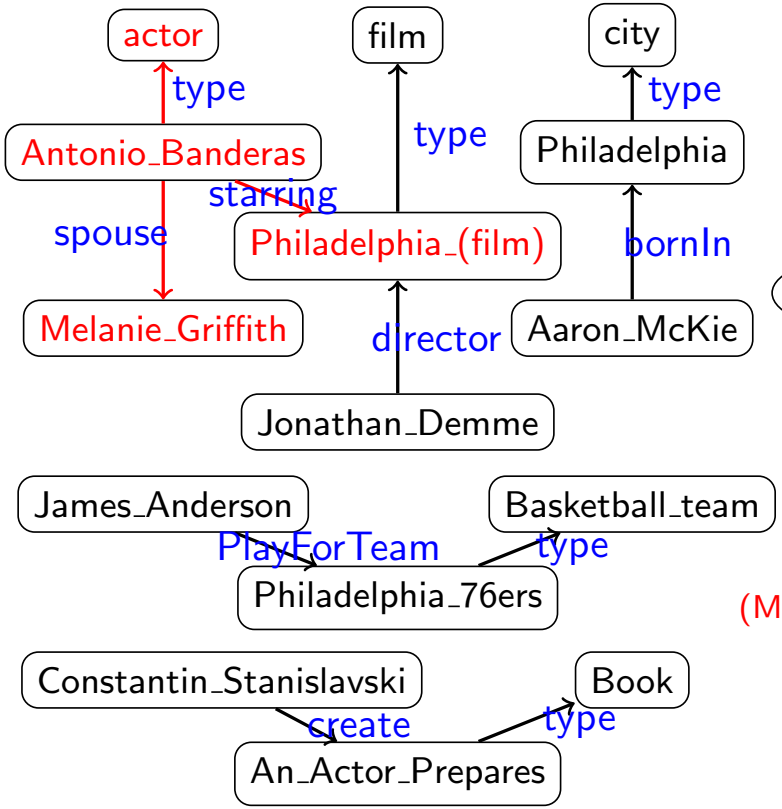
Online: Step 2.2 Mapping Vertices

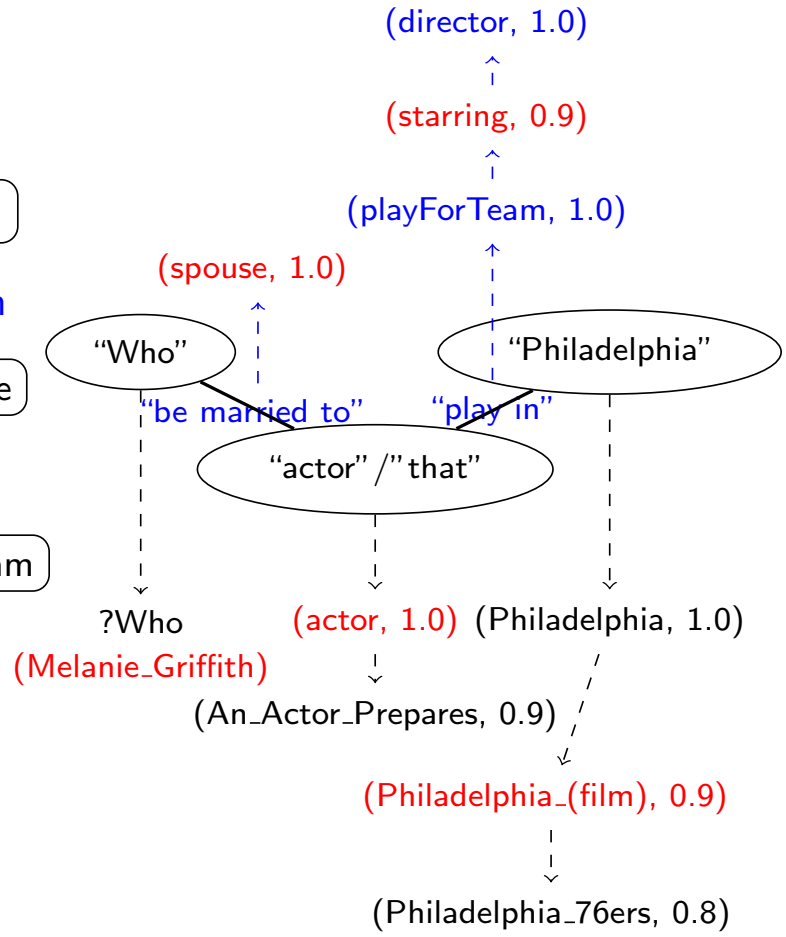


Online: Step 2.2 Mapping Vertices



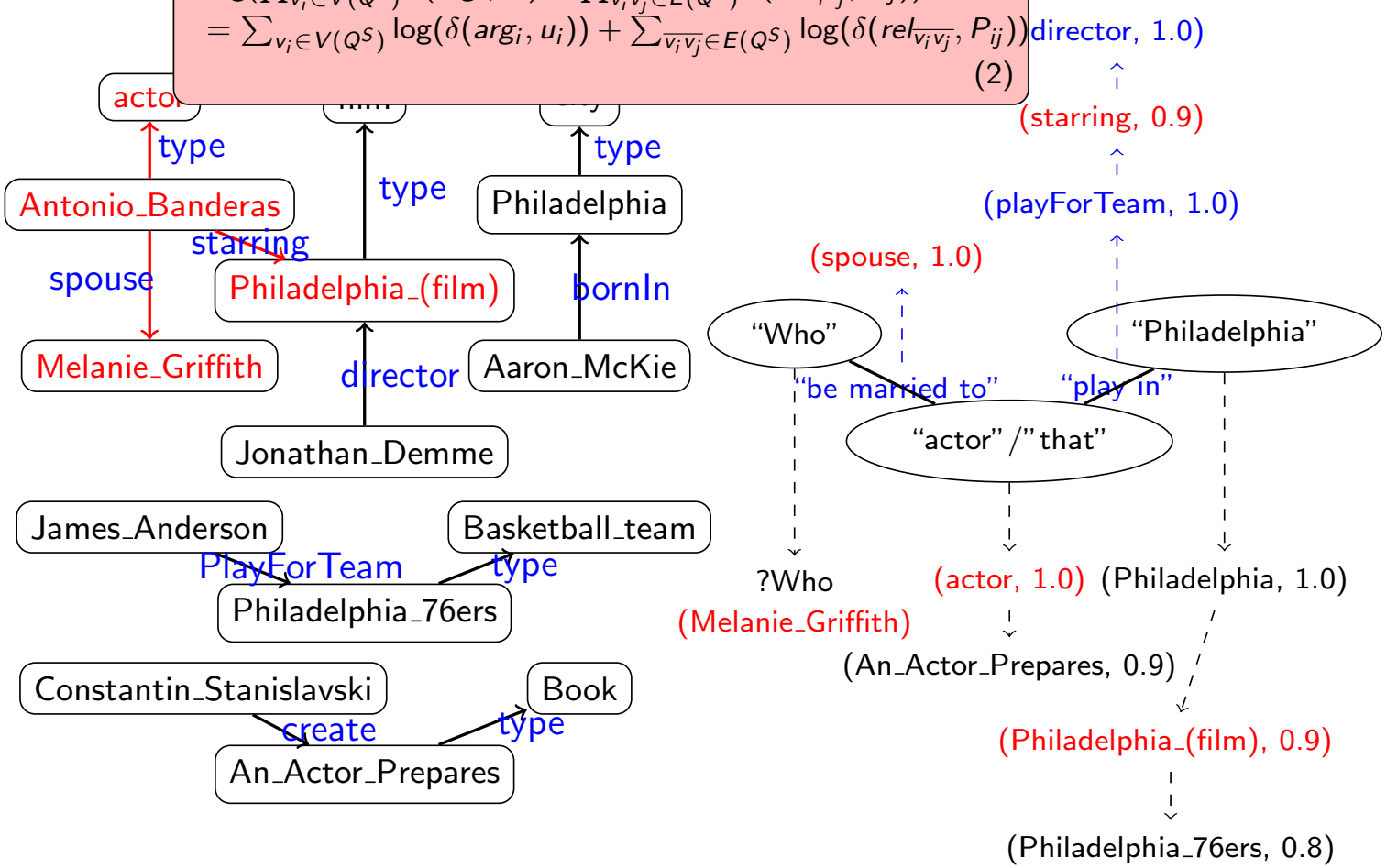
Online: Step 2.3 Finding Top-k Matches





Online: Step 2.3 Finding Top-k Matches

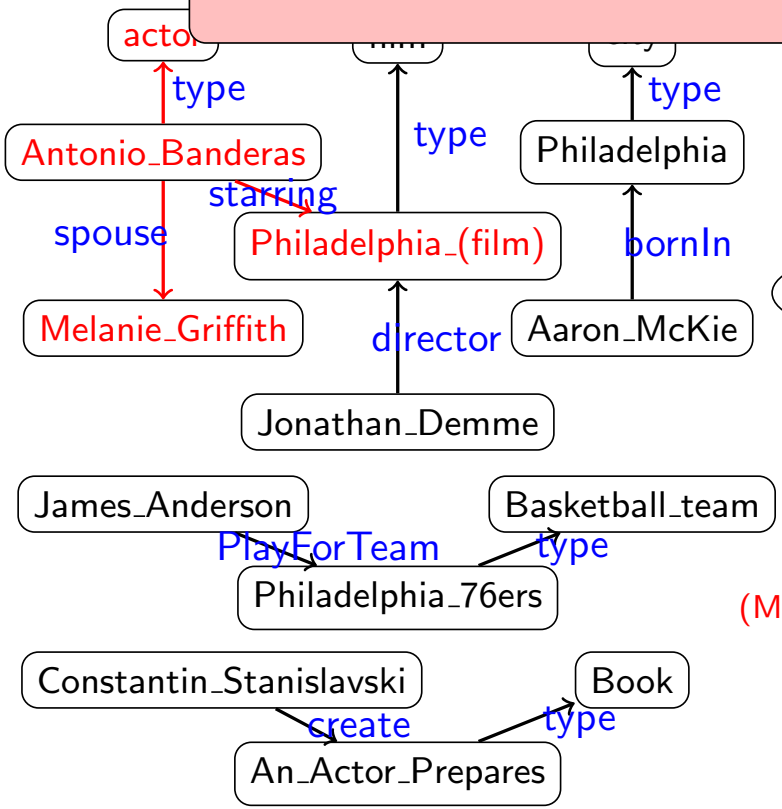
$$Score(M) = \log(\prod_{v_i \in V(Q^S)} \delta(arg_i, u_i) \times \prod_{\overline{v_i v_j} \in E(Q^S)} \delta(rel_{\overline{v_i v_j}}, P_{ij})) = \sum_{v_i \in V(Q^S)} \log(\delta(arg_i, u_i)) + \sum_{\overline{v_i v_j} \in E(Q^S)} \log(\delta(rel_{\overline{v_i v_j}}, P_{ij})) \quad (2)$$

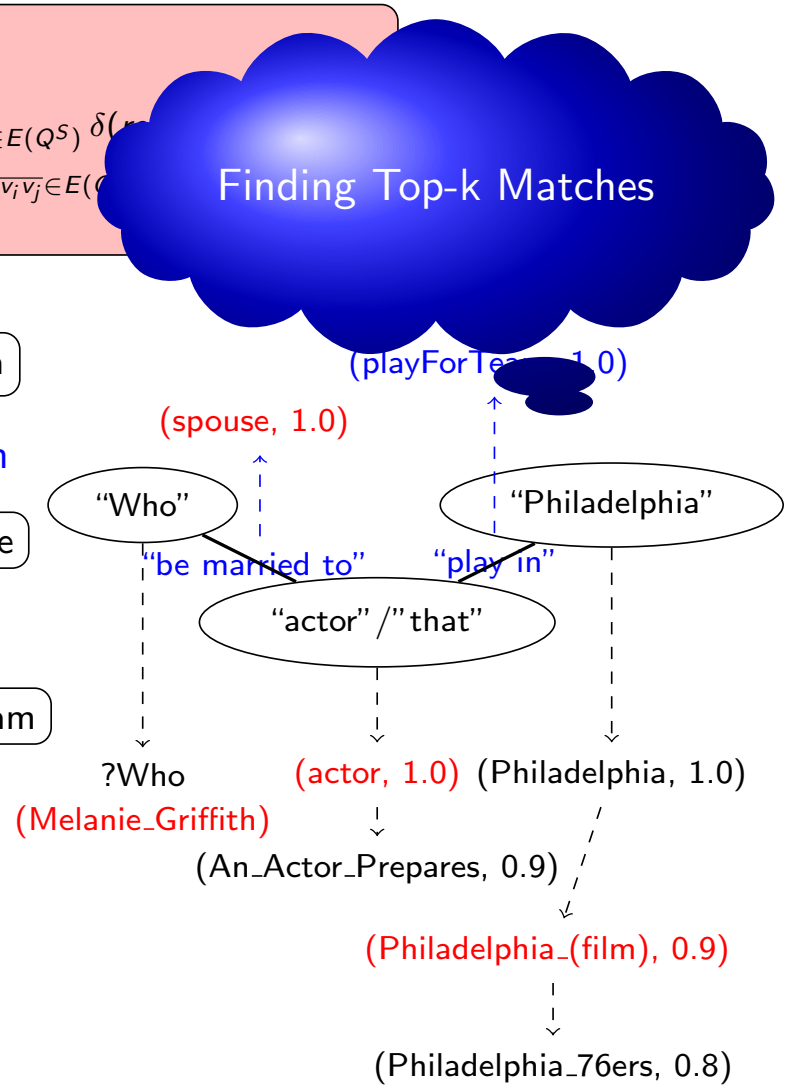


Online: Step 2.3 Finding Top-k Matches

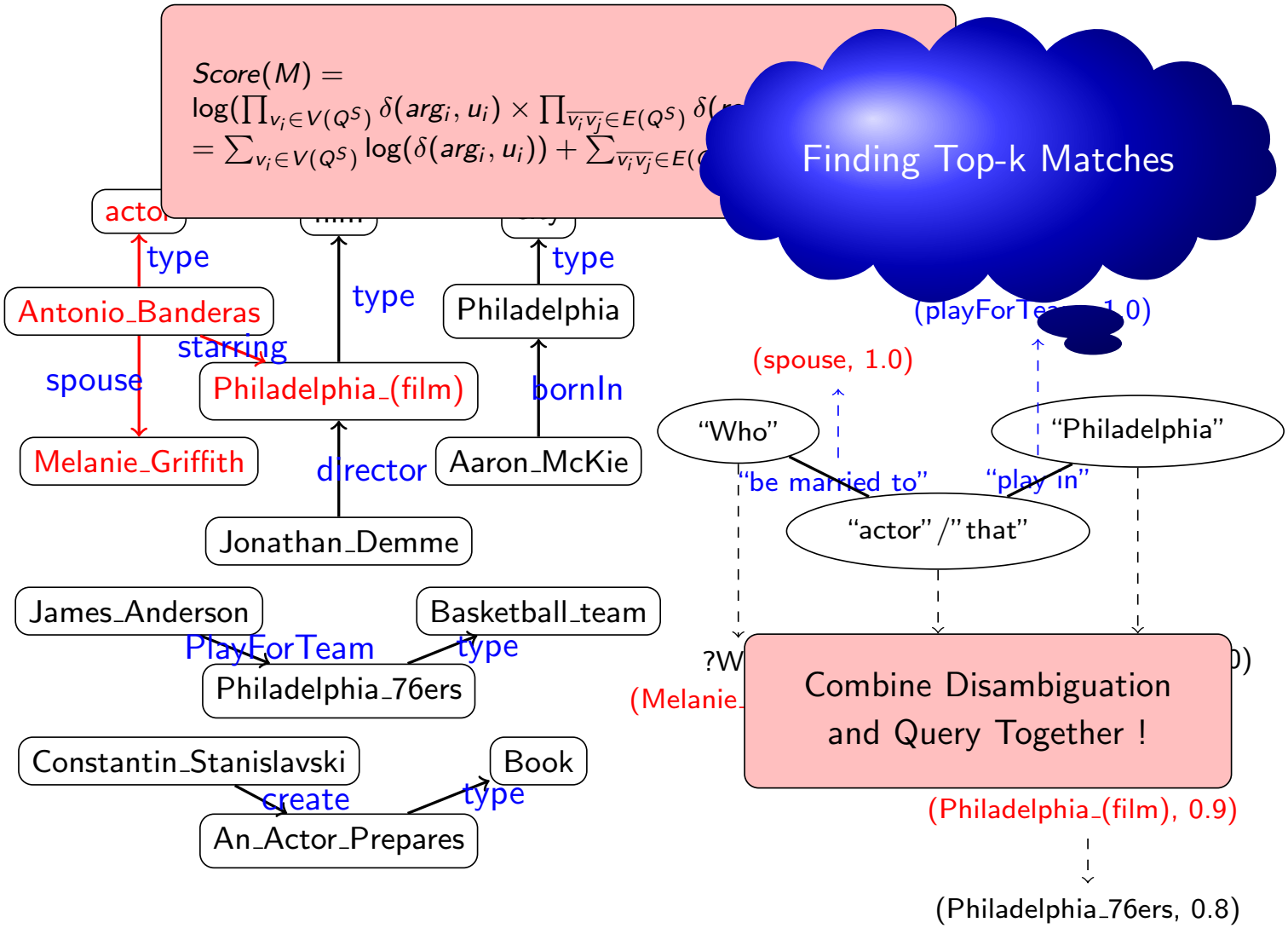
$Score(M) =$
$\log(\prod_{v_i \in V(Q^S)} \delta(arg_i, u_i) \times \prod_{\overline{v_i v_j} \in E(Q^S)} \delta($
$= \sum_{v_i \in V(Q^S)} \log(\delta(arg_i, u_i)) + \sum_{\overline{v_i v_j} \in E($

Finding Top-k Matches





Online: Step 2.3 Finding Top-k Matches



Online: Step 2.3 Finding Top-k Matches

Theorem

Finding Top-k subgraph matches of Q^S over RDF graph G is an NP-hard problem.

Online: Step 2.3 Finding Top-k Matches

Theorem

Finding Top-k subgraph matches of Q^S over RDF graph G is an NP-hard problem.

Require: **Input**: A semantic query graph Q^S and a RDF G. **Output**: Top-k SPARQL Queries, i.e., the top-k matches from Q^S to G.

1: **for** each candidate list L_{r_i}, $i = 1, \ldots\ldots, |E(Q^S)|$ **do**
2: Sorting all candidate relations in L_{r_i} in a non-ascending order
3: **for** each candidate list L_{arg_j}, $j = 1, \ldots\ldots, |V(Q^S)|$ **do**
4: Sorting all candidate entities/classes (i.e., vertices in G') in L_{arg_j} in a non-ascending order.
5: Set cursor c_i to the head of L_{r_i} and cursor c_j to the head of L_{arg_j}, respectively.
6: Set the upper bound $Upbound(Q)$ and the threshold $\theta = -\infty$
7: **while** true **do**
8: **for** each cursor c_j in list L_{arg_j}, $j = 1, \ldots\ldots, |V(Q^S)|$ **do**
9: Performance a exploration based subgraph isomorphism algorithm from cursor c_j, such as VF2, to find subgraph matches (of Q^S over G), which contains c_j.
10: Update the threshold θ to be the top-k match sore so far.
11: Move all cursors c_i and c_j by one step forward in each list.
12: Update the upper bound $Upbound(Q)$ according to Equation **??**.
13: **if** $\theta \geq Upbound(Q)$ **then**
14: Break // TA-style stopping strategy

Online: Step 2.3 Finding Top-k Matches

Theorem

Finding Top-k subgraph matches of Q^S NP-hard problem.

TA-style Top-k Algorithm !

Require: **Input**: A semantic query graph Q^S and a RDF G. **Output** matches from Q^S to G.

1: **for** each candidate list L_{r_i}, $i = 1, \ldots\ldots, |E(Q^S)|$ **do**
2: Sorting all candidate relations in L_{r_i} in a non-ascending order
3: **for** each candidate list L_{arg_j}, $j = 1, \ldots\ldots, |V(Q^S)|$ **do**
4: Sorting all candidate entities/classes (i.e., vertices in G') in L_{arg_j} in a non-ascending order.
5: Set cursor c_i to the head of L_{r_i} and cursor c_j to the head of L_{arg_j}, respectively.
6: Set the upper bound $Upbound(Q)$ and the threshold $\theta = -\infty$
7: **while** true **do**
8: **for** each cursor c_j in list L_{arg_j}, $j = 1, \ldots\ldots, |V(Q^S)|$ **do**
9: Performance a exploration based subgraph isomorphism algorithm from cursor c_j, such as VF2, to find subgraph matches (of Q^S over G), which contains c_j.
10: Update the threshold θ to be the top-k match sore so far.
11: Move all cursors c_i and c_j by one step forward in each list.
12: Update the upper bound $Upbound(Q)$ according to Equation **??**.
13: **if** $\theta \geq Upbound(Q)$ **then**
14: Break // TA-style stopping strategy

Experiments: Datasets

- RDF repository: DBPedia

Table : Statistics of RDF Graph

	DBpedia
Number of Entities	5.2 million
Number of Triples	60 million
Number of Predicates	1643
Size of RDF Graphs (in GB)	6.1

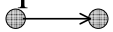
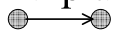
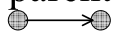
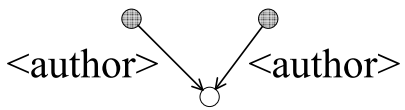
- Relation Phrase Dictionary: Patty

Table : Statistics of Relation Phrase Dataset

	wordnet-wikipedia	**freebase-wikipedia**
Number of Textual Patterns	350,568	1,631,530
Number of Entity Pairs	3,862,304	15,802,947
Average Entity Pair Number For Each Pattern	11	9

Experiments: Offline

Table : A Sample of Textual Patterns and Predicates/Predicate Paths in DBpedia

Relation Phrases	**Predicates \Predicate Paths**	**Confidence Probability**
"was married to"	<spouse> 	1.00
"was born in"	<birthplace> 	1.00
"mother of"	<parent> 	0.95
"are located in"	<locatedInArea> 	0.98
"is fed by"	<inflow> 	1.00
"open in"	<locationCity> 	1.00
"is coauthor of"		1.00

Experiments: Offline

Table : Precisions of Mapping Relation Phrases to Predicates/Predicate Paths

Precision@1	**Length-1**	**Length-2**	**Length-3**	**Length-4**
wordnet-wikipedia	61.10%	24.31%	11.18%	7.32%
freebase-wikipedia	58.56%	19.82%	9.23%	8.17%
Precision@3				
wordnet-wikipedia	72.35%	27.87%	16.73%	10.26%
freebase-wikipedia	67.72%	21.56%	10.37%	11.35%

Table : Running Time of Offline Processing

	$\theta = 2$	$\theta = 4$
wordnet-wikipedia	17 mins	3.88 hrs
freebase-wikipedia	119 mins	30.33 hrs

Experiments: Online

Benchmark: QALD-3, 99 Natural Language Questions

Table : Evaluating QALD-3 Testing Questions (on DBpedia)

	Processed	Right	Partially	Recall	Precision	F-1
Our Method	76	32	11	0.40	0.40	0.40
squall2sparql	96	77	13	0.85	0.89	0.87
CASIA	52	29	8	0.36	0.35	0.36
Scalewelis	70	1	38	0.33	0.33	0.33
RTV	55	30	4	0.34	0.32	0.33
Intui2	99	28	4	0.32	0.32	0.32
SWIP	21	14	2	0.15	0.16	0.16
DEANNA	27	21	0	0.21	0.21	0.21

Experiments: Online

ID	Questions	Response Time (in ms)
Q2	Who was the successor of John F. Kennedy?	1699
Q3	Who is the mayor of Berlin?	677
Q14	Give me all members of Prodigy?	811
Q17	Give me all cars that are produced in Germany ?	297
Q19	Give me all people that were born in Vienna and died in Berlin ?	2557
Q20	How tall is Michael Jordan ?	942
Q21	What is the capital of Canada ?	1342
Q22	Who is the governor of Wyoming ?	796
Q24	Who was the father of Queen Elizabeth II?	538
Q27	Sean Parnell is the governor of which U.S. state ?	1210
Q28	Give me all movies directed by Francis Ford Coppola.	577
Q30	What is the birth name of Angela Merkel ?	250
Q35	Who developed Minecraft ?.	2565
Q39	Give me all companies in Munich.	1312
Q41	Who founded Intel?	1105
Q42	Who is the husband of Amanda Palmer ?	1418
Q44	Which cities does the Weser flow through ?	1139
Q45	Which countries are connected by the Rhine ?	736
Q54	What are the nicknames of San Francisco ?	321
Q58	What is the time zone of Salt Lake City ?	316
Q63	Give me all Argentine films.	427
Q70	Is Michelle Obama the wife of Barack Obama ?	316
Q74	When did Michael Jackson die ?	258
Q76	List the children of Margaret Thatcher.	1139
Q77	Who was called Scarface?	719
Q81	Which books by Kerouac were published by Viking Press ?	796
Q83	How high is the Mount Everest ?	635
Q84	Who created the comic Captain America ?	589
Q86	What is the largest city in Australia ?	1419
Q89	In which city was the former Dutch queen Juliana buried ?	1700
Q98	Which country does the creator of Miffy come from ?	2121
Q100	Who produces Orangina ?	367

Experiments: Online

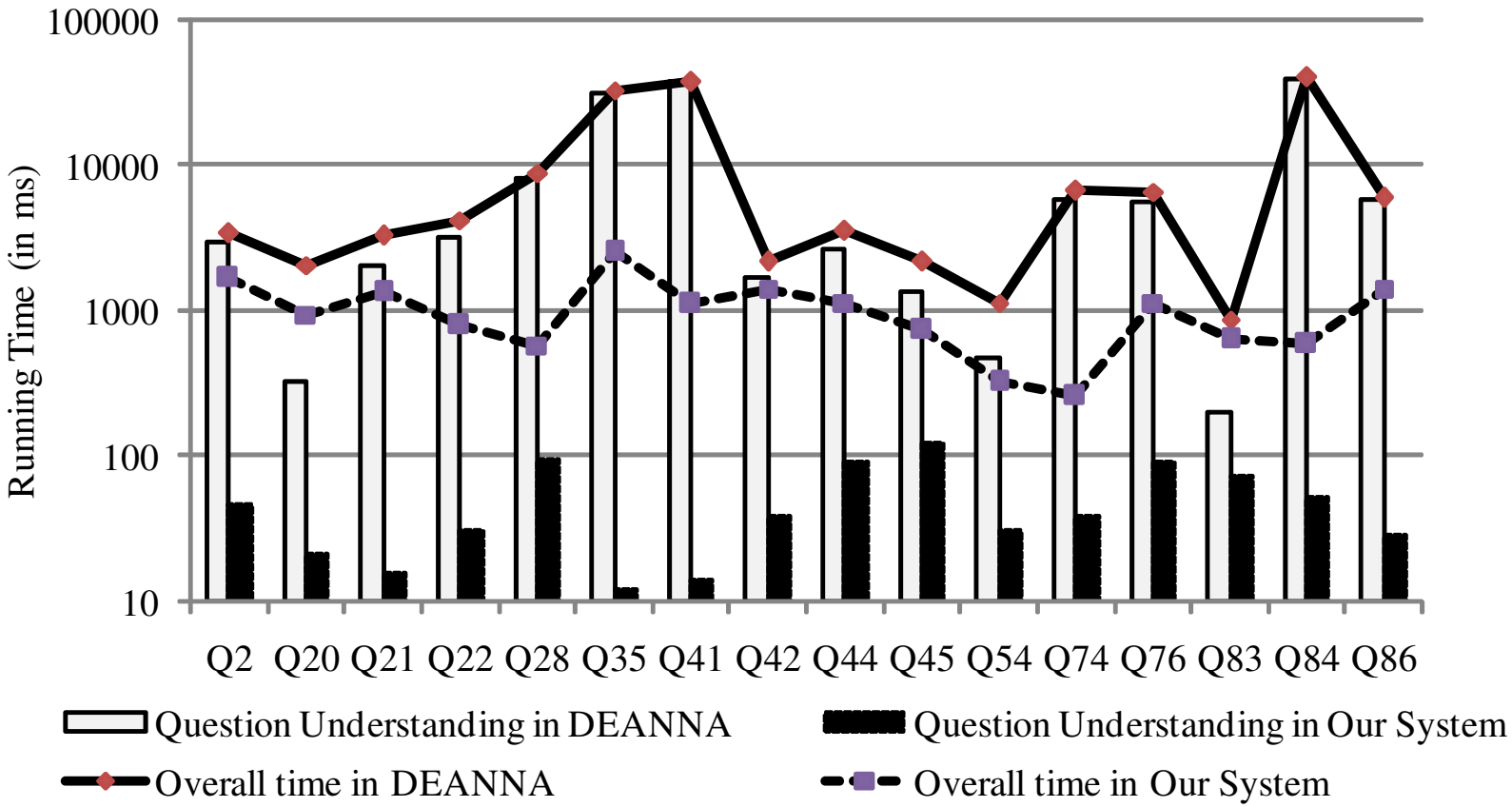


Figure : Online Running Time Comparison

Experiments: Online

QUESTION ANSWERING OVER LINKED DATA

QALD-4: Evaluation results

Workshop website: http://www.sc.cit-ec.uni-bielefeld.de/qald

Results[1] *for Task 1: Multilingual question answering over DBpedia*

	Total	Processed	Right	Partially	Recall	Precision	F-measure
Xser	50	40	34	6	0.71	0.72	**0.72**
gAnswer	50	25	16	4	0.37	0.37	**0.37**
CASIA	50	26	15	4	0.40	0.32	**0.36**
Intui3	50	33	10	4	0.25	0.23	**0.24**
ISOFT	50	28	10	3	0.26	0.21	**0.23**
RO_FII	50	50	6	0	0.12	0.12	**0.12**

Figure : QALD-4 Results

Conclusions

- Graph Database is a Possible Way for RDF Knowledge Base Management.

Conclusions

- Graph Database is a Possible Way for RDF Knowledge Base Management.
- Subgraph Matching is a Strong Tool.

Conclusions

- Graph Database is a Possible Way for RDF Knowledge Base Management.
- Subgraph Matching is a Strong Tool.
- Using RDF repository, how to Provide Knowledge Services for Applications and Common Users?

Thank you!