SPARQLing Kleene – Fast Property Paths in RDF-3X

Andrey Gubichev, TU Munich
Stephan Seufert, MPI
Srikanta Bedathur, IIIT-Delhi

June 23, 2013

Motivation

- RDF data is a graph
- SPARQL 1.1 has introduced the property paths
- `select * where {Munich yago:isLocatedIn* ?place }`
- What entities are reached from `Munich` via `yago:isLocatedIn`?

Motivation

- RDF data is a graph
- SPARQL 1.1 has introduced the property paths
- `select * where {Munich yago:isLocatedIn* ?place }`
- What entities are reached from `Munich` via `yago:isLocatedIn`?
- We could use joins and unions over the triple store to answer it
- Can we do better with a bit of indexing?

Semantics of Property Paths

- Originally, one could also count the number of paths between start and end point
- However, this semantics leads to #P-hard problems (M.Arenas, WWW'12)
- Now, W3C standard only allows to check for reachability, not counting paths

Previous Work: RDF-3X

- a triple store
- extensive indexing
- join ordering with Dynamic Programming
- accurate cardinality estimation for common types of queries
- T.Neumann et al, SIGMOD 2009

Previous Work: Reachability Index FERRARI

- FERRARI index: based on tree interval labeling, assigns exact and approximate labels to nodes (ICDE'2013)
- Runtime: use index plus limited DFS
- FERRARI:
  - indexes 100 Mln triples of YAGO in 90 sec
  - takes 210 Mb
  - answers a reachability query for (start,end) in microseconds
- (all the numbers: off-the-shelf laptop)

Our Contribution

How to use FERRARI in RDF-3X

- Query optimization
- Runtime technique to speed up query execution

QO: Getting the Logical Operator

Property path triple may correspond to:

- a filter (if one of subject or object is constant)
  - `select * where {Munich yago:isLocatedIn* ?place }`
- a scan, if one of subject of object is not bound
  - `select * where {?city yago:isLocatedIn* ?place }`
- a join, otherwise
  - Reachability Join: similar to Hash Join (build and probe part)
  - `select * where {?city yago:isLocatedIn* ?place.`
    `?city hasName "Munich".`
    `?place type ?type. }`

In the last case, there is one more join opportunity (reflected in the Query Graph)

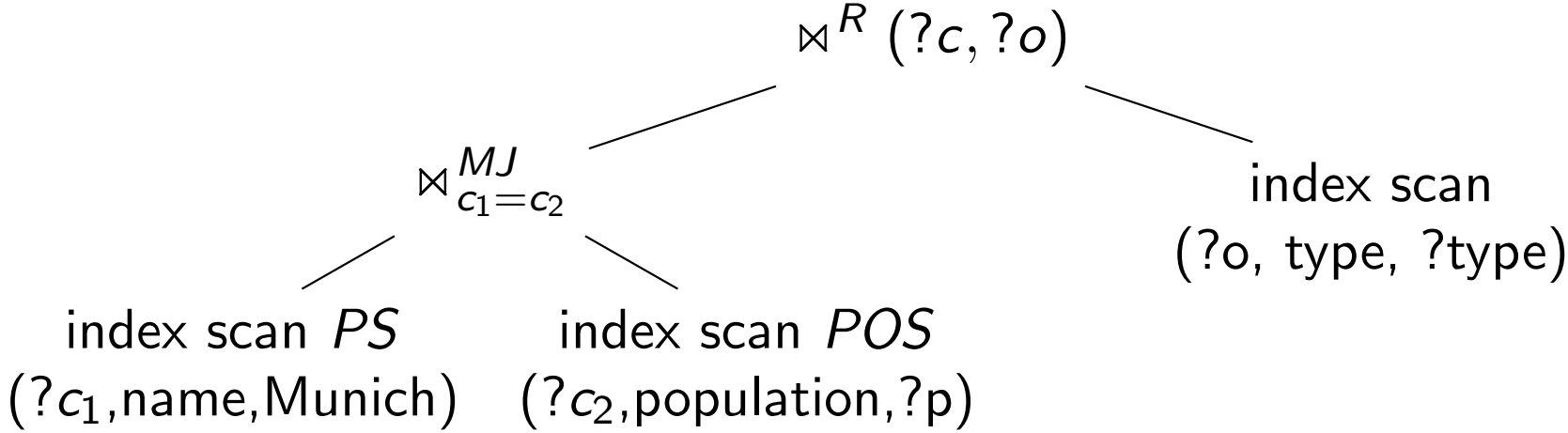
QO: Plan generation

In order to use Dynamic Programming, we extend the cost model

- Estimated cardinality of the scan is provided by the index immideately
- Cardinality estimation for the join: independence assumption + index information

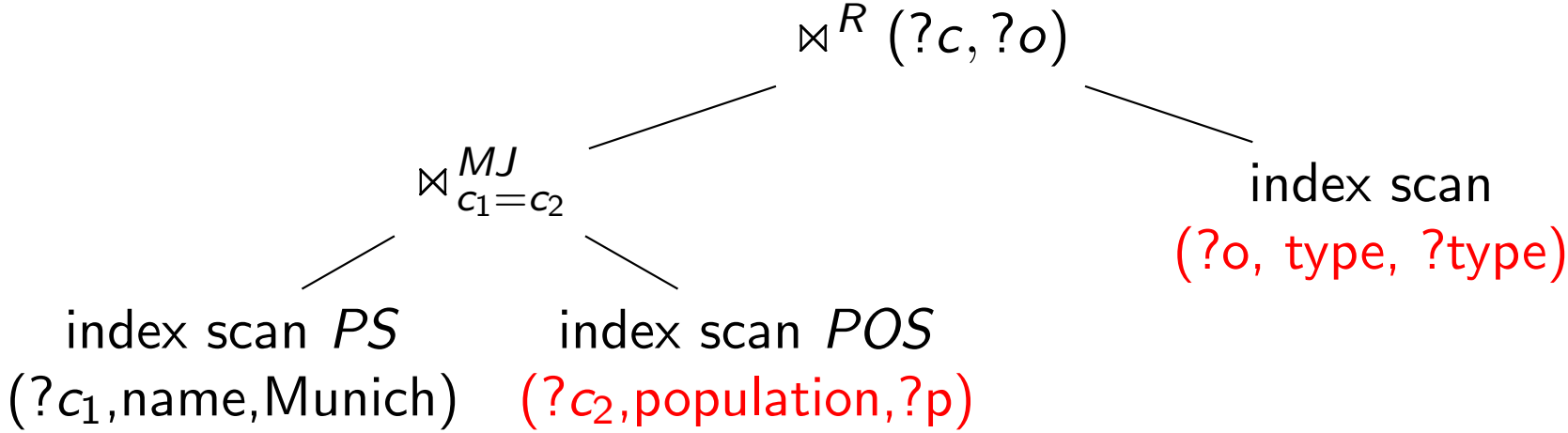
Runtime: A typical execution plan

```
select ?city ?p ?type where { ?city hasName "Munich".
?city hasPopulation ?p.  ?city locatedIn*/type ?type.  }
```



Runtime: A typical execution plan

```
select ?city ?p ?type where { ?city hasName "Munich".
?city hasPopulation ?p.  ?city locatedIn*/type ?type.  }
```



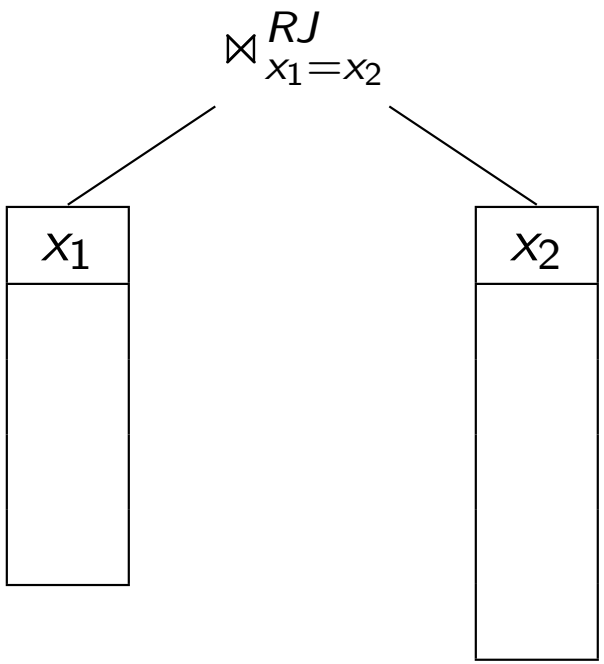
- Individual triple patterns are very unselective
- We can pass gap information between different index scans, so that most part of the data can be skipped (indirectly)
- (With some restrictions) this idea extends to Reachability Joins

Sideways Information Passing for Property Paths

Build phase: construct **domain filters** for observed attribute values, using approx intervals from FERRARI:

min	max	Bloom filter (1024 bytes)

Probe phase: pass the bloome filter to the right index scan; it can skip values

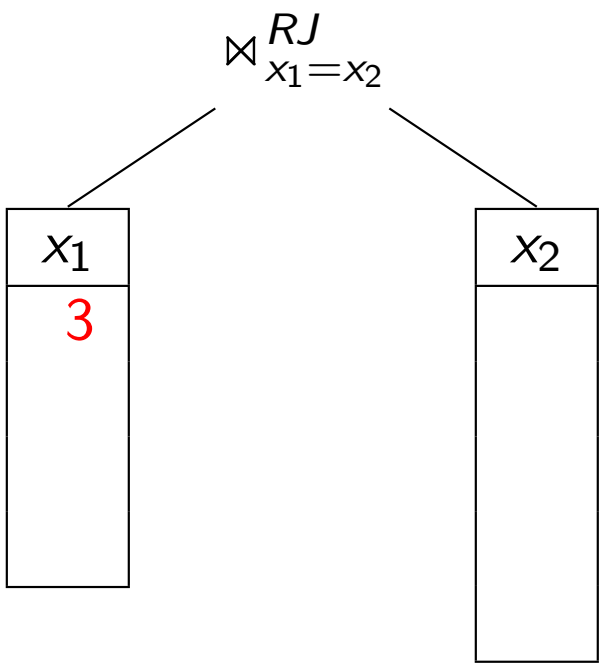


Sideways Information Passing for Property Paths

Build phase: construct **domain filters** for observed attribute values, using approx intervals from FERRARI:

min	max	Bloom filter (1024 bytes)

Probe phase: pass the bloome filter to the right index scan; it can skip values

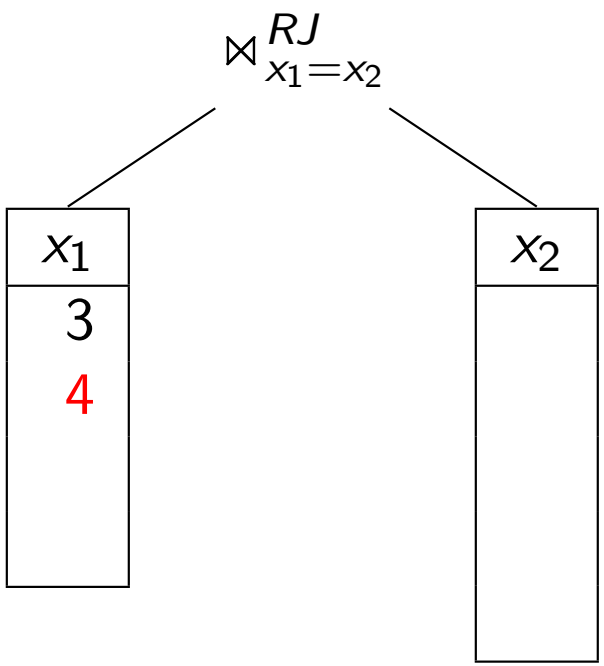


Sideways Information Passing for Property Paths

Build phase: construct **domain filters** for observed attribute values, using approx intervals from FERRARI:

min	max	Bloom filter (1024 bytes)

Probe phase: pass the bloome filter to the right index scan; it can skip values



FERRARI Index

ID	Intervals
3	[1, 1]
4	[8, 8], [9, 9]

Domain for $?o$

min	max	Bloom
1	9	011000

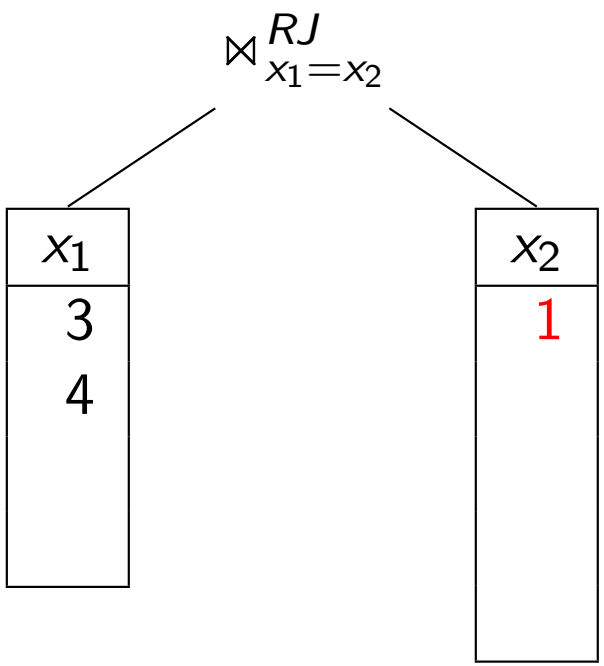
hash function: $v \mod 7$

Sideways Information Passing for Property Paths

Build phase: construct **domain filters** for observed attribute values, using approx intervals from FERRARI:

min	max	Bloom filter (1024 bytes)

Probe phase: pass the bloome filter to the right index scan; it can skip values



FERRARI Index

ID	Intervals
3	[1, 1]
4	[8, 8], [9, 9]

Domain for $?o$

min	max	Bloom
1	9	011000

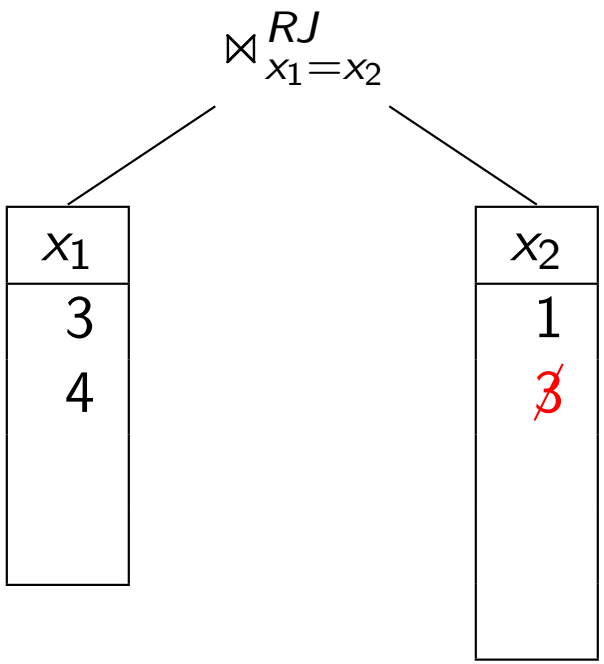
hash function: $v \mod 7$

Sideways Information Passing for Property Paths

Build phase: construct **domain filters** for observed attribute values, using approx intervals from FERRARI:

min	max	Bloom filter (1024 bytes)

Probe phase: pass the bloome filter to the right index scan; it can skip values



FERRARI Index

ID	Intervals
3	[1, 1]
4	[8, 8], [9, 9]

Domain for $?o$

min	max	Bloom
1	9	011000

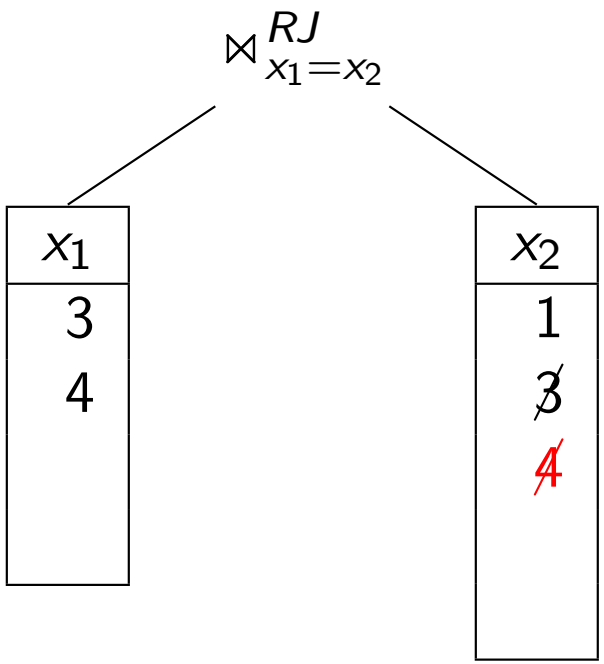
hash function: $v \mod 7$

Sideways Information Passing for Property Paths

Build phase: construct **domain filters** for observed attribute values, using approx intervals from FERRARI:

min	max	Bloom filter (1024 bytes)

Probe phase: pass the bloome filter to the right index scan; it can skip values



FERRARI Index

ID	Intervals
3	[1, 1]
4	[8, 8], [9, 9]

Domain for $?o$

min	max	Bloom
1	9	011000

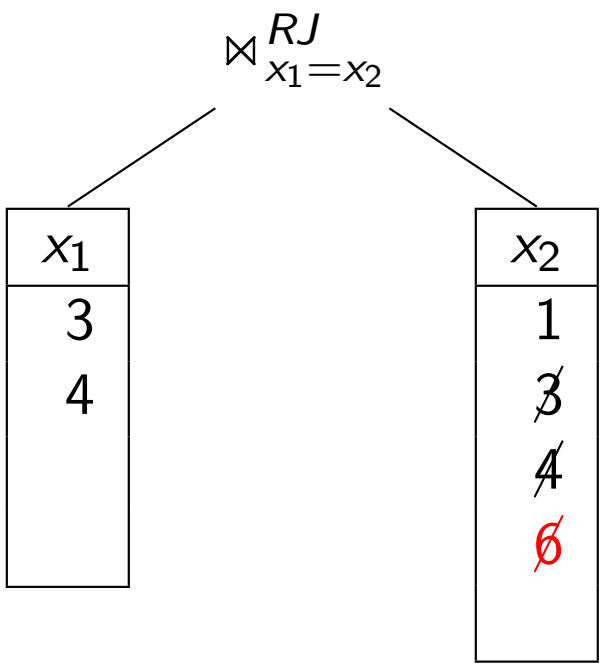
hash function: $v \mod 7$

Sideways Information Passing for Property Paths

Build phase: construct **domain filters** for observed attribute values, using approx intervals from FERRARI:

min	max	Bloom filter (1024 bytes)

Probe phase: pass the bloome filter to the right index scan; it can skip values



FERRARI Index

ID	Intervals
3	[1, 1]
4	[8, 8], [9, 9]

Domain for $?o$

min	max	Bloom
1	9	011000

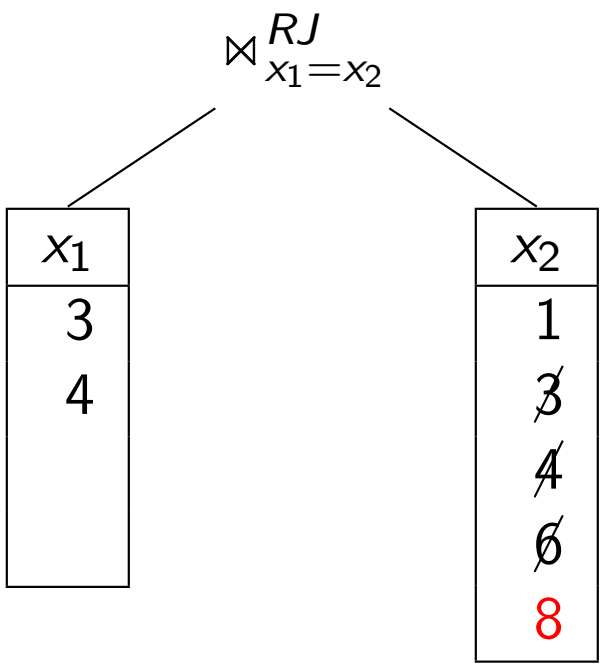
hash function: $v \mod 7$

Sideways Information Passing for Property Paths

Build phase: construct **domain filters** for observed attribute values, using approx intervals from FERRARI:

min	max	Bloom filter (1024 bytes)

Probe phase: pass the bloome filter to the right index scan; it can skip values



FERRARI Index

ID	Intervals
3	[1, 1]
4	[8, 8], [9, 9]

Domain for $?o$

min	max	Bloom
1	9	011000

hash function: $v \mod 7$

Choke points

How to formulate interesting queries to test property path support? What are the hard things?

- Choosing the right build part
- Compare cardinalities of different property paths
- Compare cardinalities of property paths vs index scans

We suggested some queries and evaluated our solution (against Virtuoso)

Conclusions

We have:

- Support for property paths in RDF-3X
- Full-fledged system: query optimization, sideways information passing
- Choke points and queries and evaluation

Future Work:

- Updates