Query Execution Optimization for Clients of Triple Patterns Fragments

Reducing HTTP traffic for
scalable linked data consumption

Joachim Van Herwegen, Ruben Verborgh, Erik Mannens, Rik Van De Walle

Accessing linked data

SPARQL endpoints, data dumps, simple interfaces, …

Still looking for the **ultimate** linked data solution

Full SPARQL support

High scalability

Fast response time

Low server & client load

…

Not found yet, so we focused on improving the **response time** for clients using simple interfaces (*Triple Pattern Fragments*).

Query Execution Optimization for Clients of Triple Patterns Fragments

Accessing Linked Data

Problem statement

Improved join tree

Optimizing local joins

Bringing it all together

Query Execution Optimization for Clients of Triple Patterns Fragments

Accessing Linked Data

Problem statement

Improved join tree

Optimizing local joins

Bringing it all together

Linked Data access extremes

SPARQL protocol

Live data

Full SPARQL support

High server load

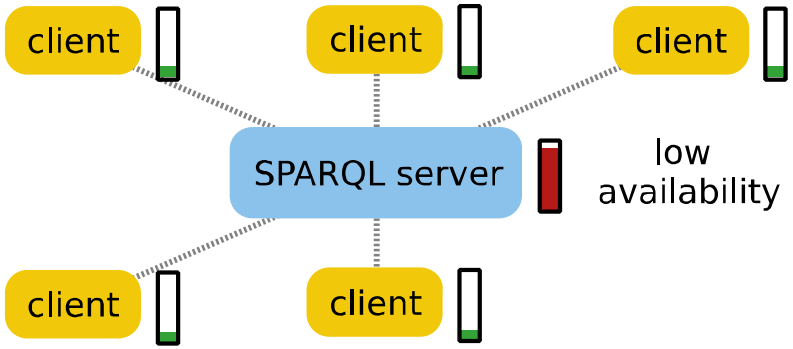
Data dump

Static data

Remote: 1 query

Local: full queries

High client load



Linked Data Fragments

Generic way to describe how linked data can be accessed

Data	Results when accessing a selector
Metadata	Description of the fragment
Controls	Links to other fragments

Verborgh et al. – Web-scale querying through Linked Data Fragments

SPARQL endpoint

Accessing data through a SPARQL endpoint

Data	Bindings matching a SPARQL query
Metadata	{ } (*data contains everything needed*)
Controls	{ } (*interface can answer everything*)

Triple Pattern Fragments

Accessing data through Triple Pattern Fragments

Data	Triples matching a triple pattern
Metadata	Count estimate, page size, etc.
Controls	First page, next page, root fragment

Triple Pattern Fragments

Data Portal @ linkeddatafragments.org

#LD
Linked Data Fragments

DBpedia 2014

Query DBpedia 2014 by triple pattern

subject:
predicate: http://dbpedia.org/ontology/birthPlace
object:

query

Find matching triples

Matches in DBpedia 2014 for *{ ?s <http://dbpedia.org/ontology/birthPlace> ?o }*

Showing triples 1 to 100 of ±625,811 with 100 triples per page. next

metadata/controls

```
!PAUS3   birthPlace   Odessa.
!PAUS3   birthPlace   Ukraine.
%22Bassy%22_Bob_Brockmann   birthPlace   Los_Angeles.
%22Bassy%22_Bob_Brockmann   birthPlace   New_Orleans.
%22Dr._Death%22_Steve_Williams   birthPlace   Lakewood,_Colorado.
%22Dr._Death%22_Steve_Williams   birthPlace   Oklahoma.
%22Freeway%22_Rick_Ross   birthPlace   Troup,_Texas.
%22Freeway%22_Rick_Ross   birthPlace   United_States.
```

results

Query Execution Optimization for Clients of Triple Patterns Fragments

Accessing Linked Data

Problem statement

Improved join tree

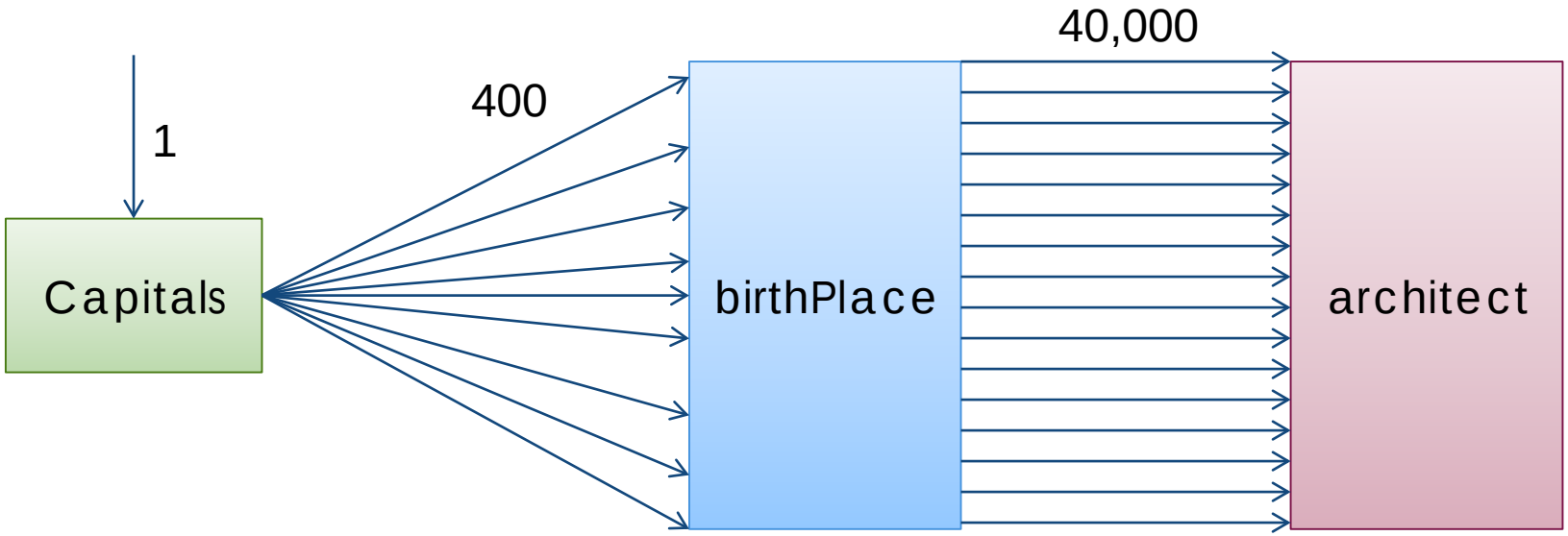
Optimizing local joins

Bringing it all together

Greedy algorithm

```
SELECT ?person ?city WHERE {
   ?person a db:Architect.                          1200 triples
   ?person db:birthPlace ?city.                     430,000 triples
   ?city dc:subject db:Category:Capitals_in_Europe. 60 triples
}
```

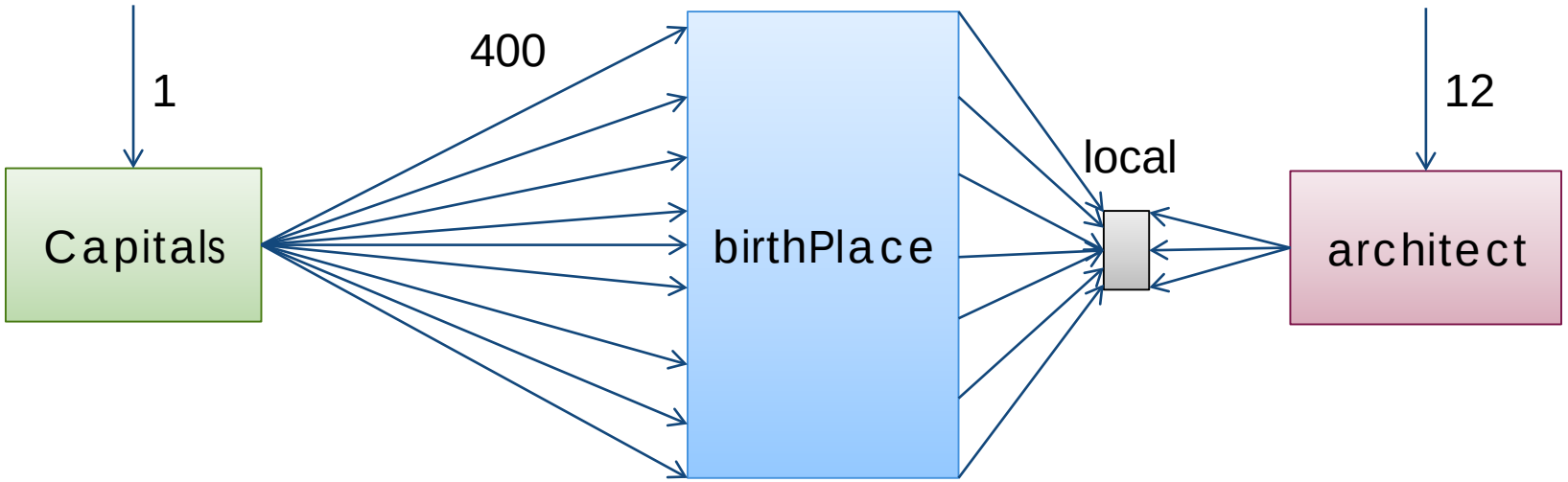
Start from the smallest pattern, apply bindings and do recursion



Optimized algorithm

```
SELECT ?person ?city WHERE {
   ?person a db:Architect.                          1200 triples
   ?person db:birthPlace ?city.                     430,000 triples
   ?city dc:subject db:Category:Capitals_in_Europe. 60 triples
}
```

Find optimal solution for every pattern



Query Execution Optimization for Clients of Triple Patterns Fragments

Accessing Linked Data

Problem statement

Improved join tree

Optimizing local joins

Bringing it all together

Optimized algorithm

Goal

Minimize HTTP calls required to solve BGP query

Solution

2 possible *roles* for every pattern in query:

Download pattern completely

or

Bind variable and download resulting patterns

Estimate best option for every pattern

Extended example

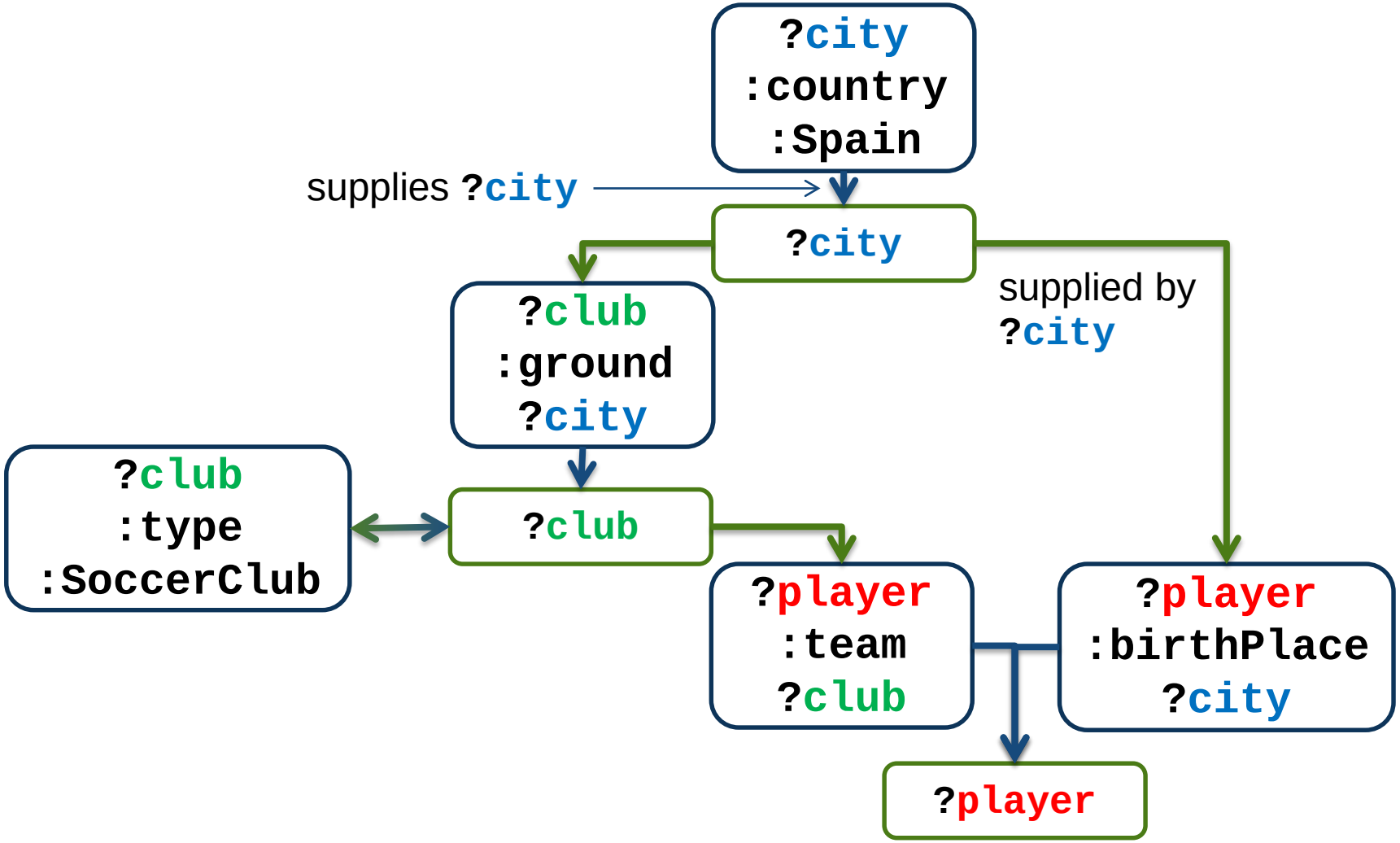
`?player :team ?club`	365,000 triples
`?club :type :SoccerClub`	16,000 triples
`?club :ground ?city`	15,000 triples
`?city :country :Spain`	**7,000 triples**
`?player :birthPlace ?city`	430,000 triples

Always download smallest pattern

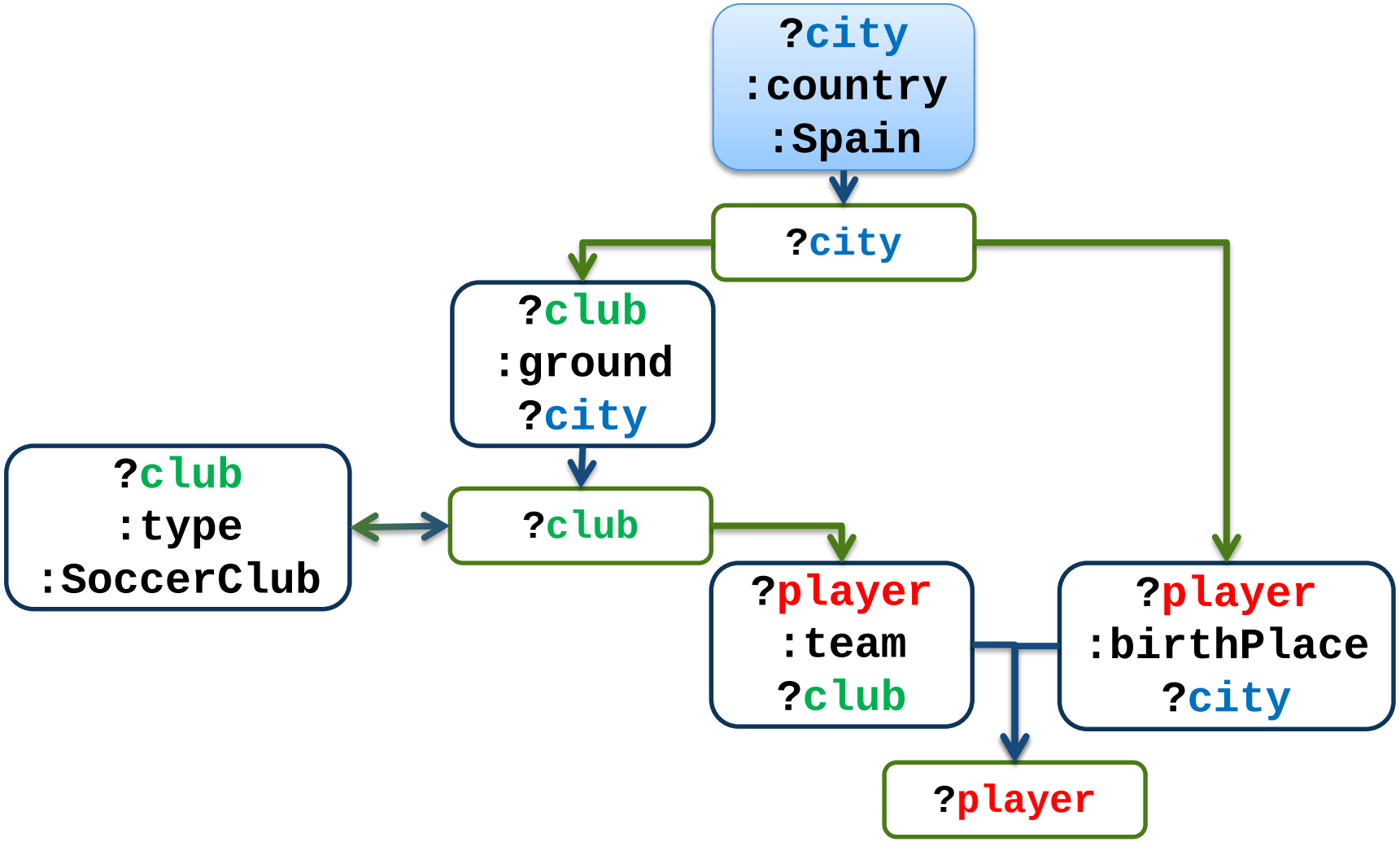
Determine others on shared variables and results so far

Can change during runtime

Extended example

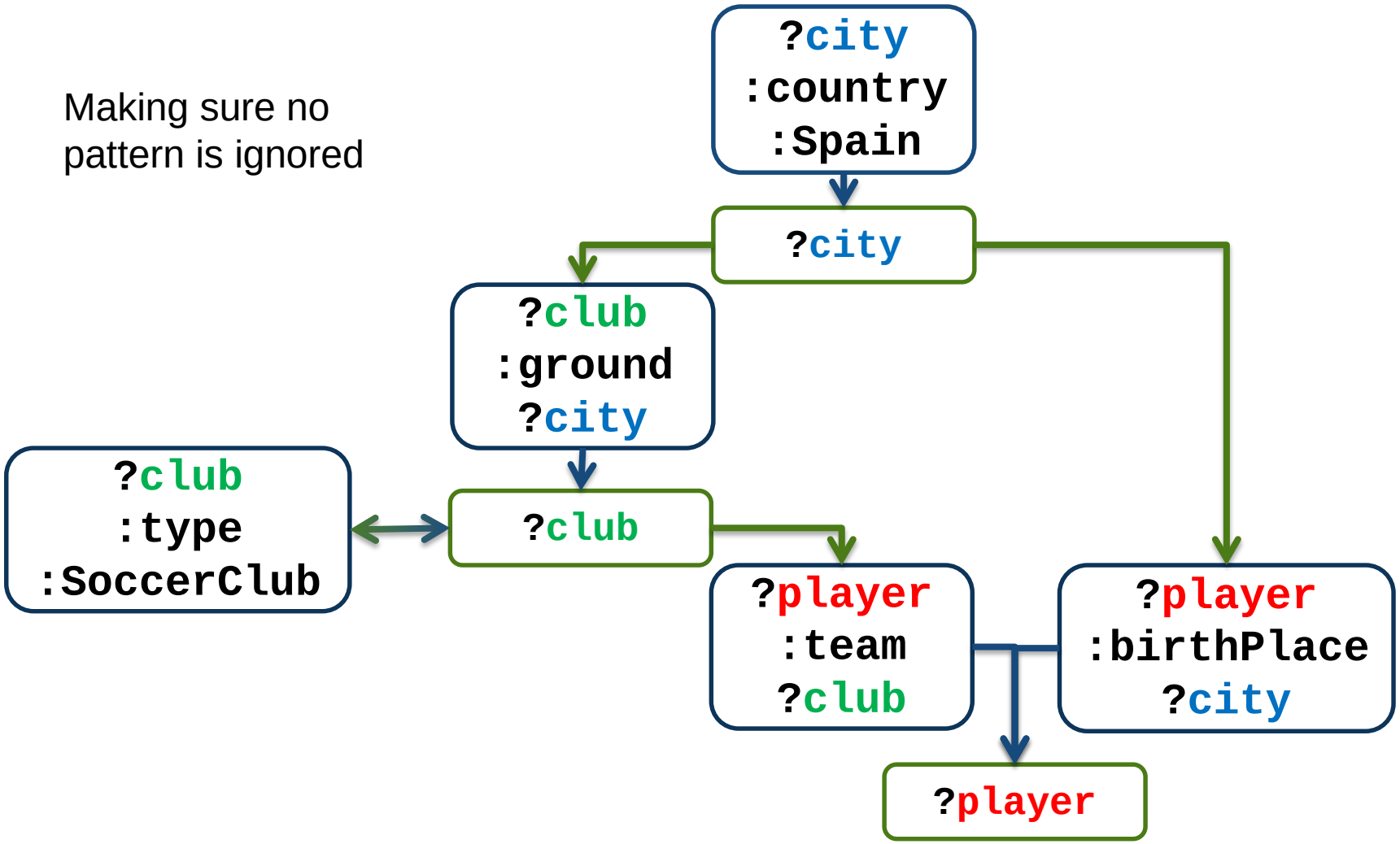


First iteration



Further iterations

Making sure no pattern is ignored



Updating pattern roles

Estimate which option requires least HTTP calls.

Download: $\left\lceil \frac{\#triples}{pagesize} \right\rceil$

Bind: $\left\lceil \frac{\overbrace{avg\ triples/binding}^{\text{avg pages per binding}}}{pagesize} \right\rceil \cdot \max \left\{ \frac{\overbrace{bindings\ found}^{\text{avg bindings per triple}}}{triples\ downloaded} \cdot \#triples \right\}$ for all suppliers

Swap when necessary, taking into account work done so far

Query Execution Optimization for Clients of Triple Patterns Fragments

Accessing Linked Data

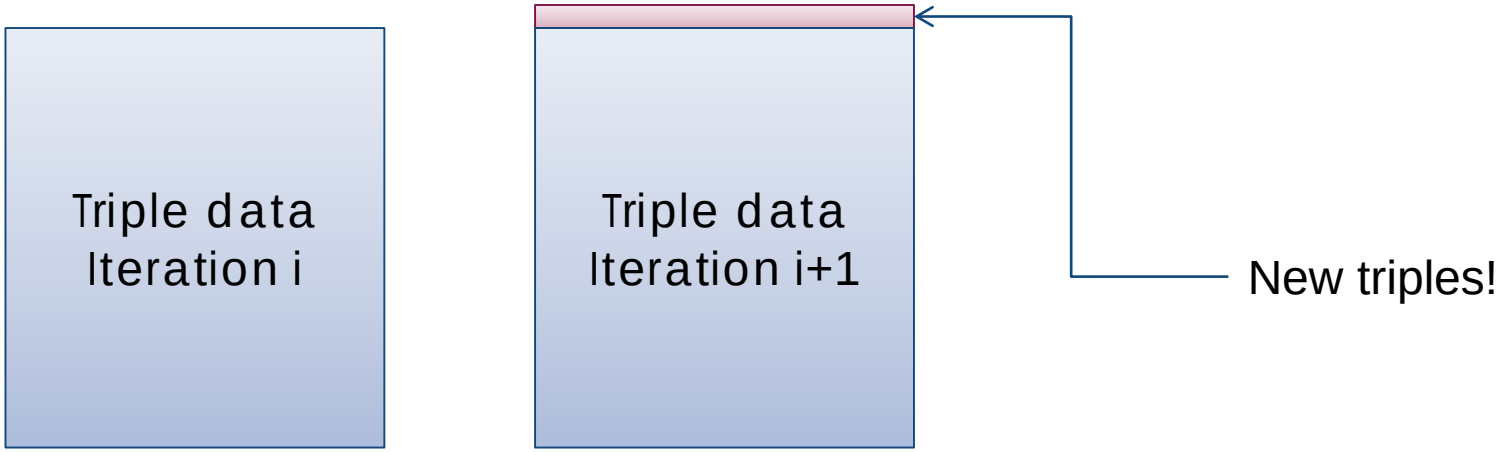
Problem statement

Improved join tree

Optimizing local joins

Bringing it all together

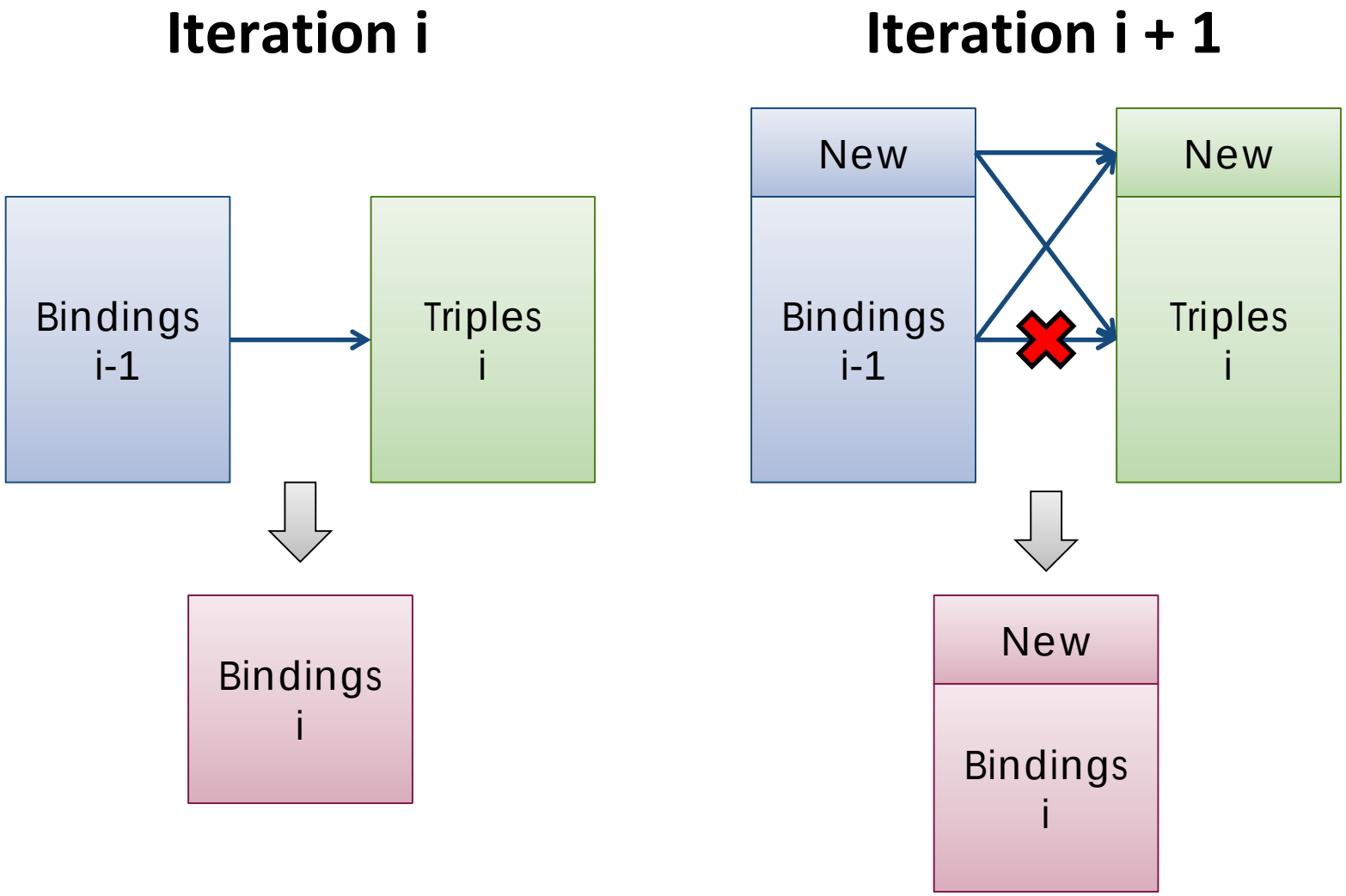
Local joining



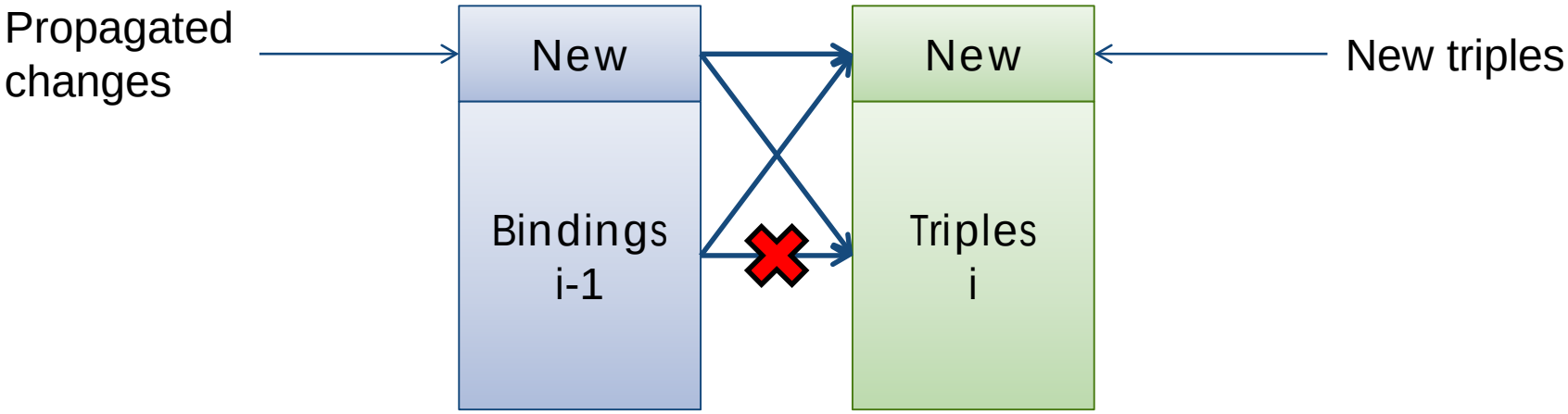
Most join data from previous iterations can be reused.

Challenge: reuse as much data as possible.

Join tree step



Minimizing joins



Start with the largest unchanged, connected set of patterns.

Estimate remainder of join order based on pattern size and connectivity.

Query Execution Optimization for Clients of Triple Patterns Fragments

Accessing Linked Data

Problem statement

Improved join tree

Optimizing local joins

Bringing it all together

Summary

Prevent local optima

Join tree instead of join path

Reuse local join data

Test setup

Single machine

Intel Core i5-3230M CPU @ 2.60GHz

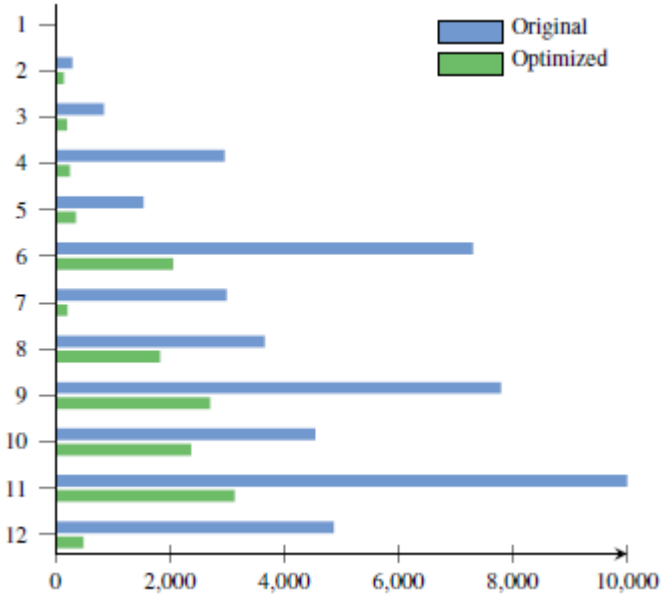
8 GB RAM

Both client and server

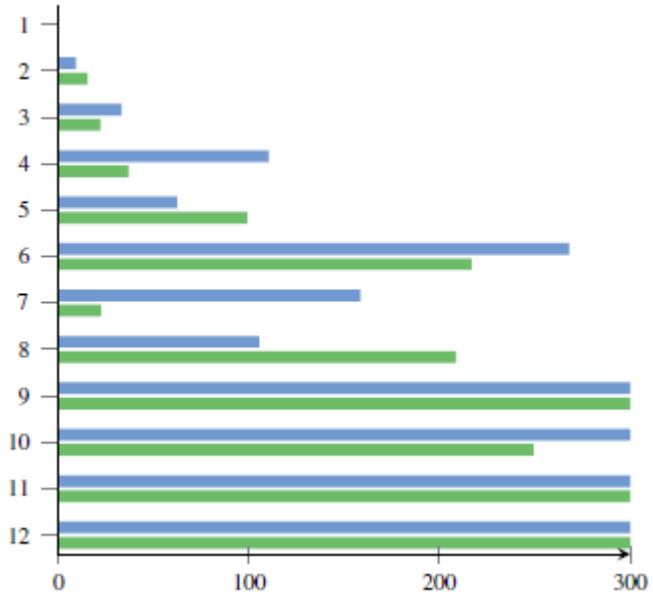
Artificial delay of 100ms on server to simulate network delay

Results

WatDiv benchmark queries, 100ms delay on server



Median # HTTP calls



Median time (s)

Conclusion

Less HTTP calls with more client-side processing

Ideal for slow connection situations

Still room for improvements

- *No parallelism*
- *Focus on BGPs*
- *More work per HTTP call*
- *Not guaranteed to be better*

Questions?

Thank you!

Come see demo #13 on thursday