Linked Data Query Processing

Tutorial at the 22nd International World Wide Web Conference (WWW 2013)

May 14, 2013

http://db.uwaterloo.ca/LDQTut2013/

5. Query Planning and Optimization

Olaf Hartig
University of Waterloo

Query Plan Selection

- **Possible assessment criteria:**
  - Benefit (size of *computed* query result)
  - Cost (overall query execution time)
  - Response time (time for returning *k* solutions)
- **To select from candidate plans, criteria must be estimated**
- **For index-based source selection: estimation may be based on information recorded in the index** [HHK+10]
- **For (pure) live exploration: estimation impossible**
  - No a-priori information available
  - Use heuristics instead

Outline

- **Heuristics-Based Planning**
- **Optimizing Link Traversing Iterators**
  - Prefetching
  - Postponing
- **Source Ranking**
  - Harth et al. [HHK+10, UHK+11]
  - Ladwig and Tran [LT10]

Heuristics-Based Plan Selection [Har11a]

- **Four rules:**
  - DEPENDENCY RULE
  - SEED RULE
  - INSTANCE SEED RULE
  - FILTER RULE

- **Tailored to LTBQE implemented by link traversing iterators**
- **Assumptions about queries:**
  - Query pattern refers to instance data
  - URIs mentioned in the query pattern are the seed URIs

Dependency Rule

Use a *dependency respecting* query plan

- **Dependency: a variable from each triple pattern already occurs in one of the preceding triple patterns**

tp_1 = (*?p* , ex:affiliated_with , <http://.../orgaX>) I_1

tp_2 = (*?p* , ex:interested_in , *?b*) I_2

tp_3 = (*?b* , rdf:type , <http://.../Book>) I_3

Query

?p ex:affiliated_with <http://.../orgaX>
?p ex:interested_in *?b*
?b rdf:type <http://.../Book>

DEPENDENCY RULE

Use a *dependency respecting* query plan

- **Dependency: a variable from each triple pattern already occurs in one of the preceding triple patterns**

tp_1 = (*?p* , ex:affiliated_with , <http://.../orgaX>) I_1

tp_2 = (*?p* , ex:interested_in , *?b*) I_2

tp_3 = (*?b* , rdf:type , <http://.../Book>) I_3

Query

?p ex:affiliated_with <http://.../orgaX>
?p ex:interested_in *?b*
?b rdf:type <http://.../Book>

DEPENDENCY RULE

Use a *dependency respecting* query plan

- **Dependency: a variable from each triple pattern already occurs in one of the preceding triple patterns**
- **Rationale: Avoid cartesian products**

tp_1 = (*?p* , ex:affiliated_with , <http://.../orgaX>) I_1

tp_2 = (*?b* , rdf:type , <http://.../Book>) I_2

tp_3 = (*?p* , ex:interested_in , *?b*) I_3

Query

?p ex:affiliated_with <http://.../orgaX>
?p ex:interested_in *?b*
?b rdf:type <http://.../Book>

SEED RULE

Use a plan with a *seed triple pattern*

- **Seed triple pattern of a plan**

  … is the first triple pattern in the plan, and

  … contains at least one HTTP URI

- **Rationale:**
  Good starting point

Recall assumption:
seed URIs = URIs in the query

Query

?p ex:affiliated_with <http://.../orgaX> √
?p ex:interested_in *?b* √
?b rdf:type <http://.../Book> √

INSTANCE SEED RULE

Avoid a seed triple pattern with vocabulary terms

- **Patterns to avoid:**
  - ✗ *?s* ex:any_property *?o*
  - ✗ *?s* rdf:type ex:any_class

- **Rationale: URIs for vocabulary terms usually resolve to vocabulary definitions with little instance data**

Query

?p ex:affiliated_with <http://.../orgaX> √

?p ex:interested_in *?b* ✗

?b rdf:type <http://.../Book> ✗

FILTER RULE

Use a plan where all *filtering triple patterns* are as close to the first triple pattern as possible

- **Filtering triple pattern:** ***each*** **variable already occurs in one of the preceding triple patterns**

- **For each valuation consumed as input a filtering TP can only report 1 or 0 valuations as output**

- **Rationale: Reduce cost**

tp_1 = (*?p* , ex:affiliated_with , <http://.../orgaX>) I_1

{ *?p* = <http://.../alice> }

tp_2 = (*?p* , ex:interested_in , *?b*)
tp_2' = (<http://.../alice> , ex:interested_in , *?b*) I_2

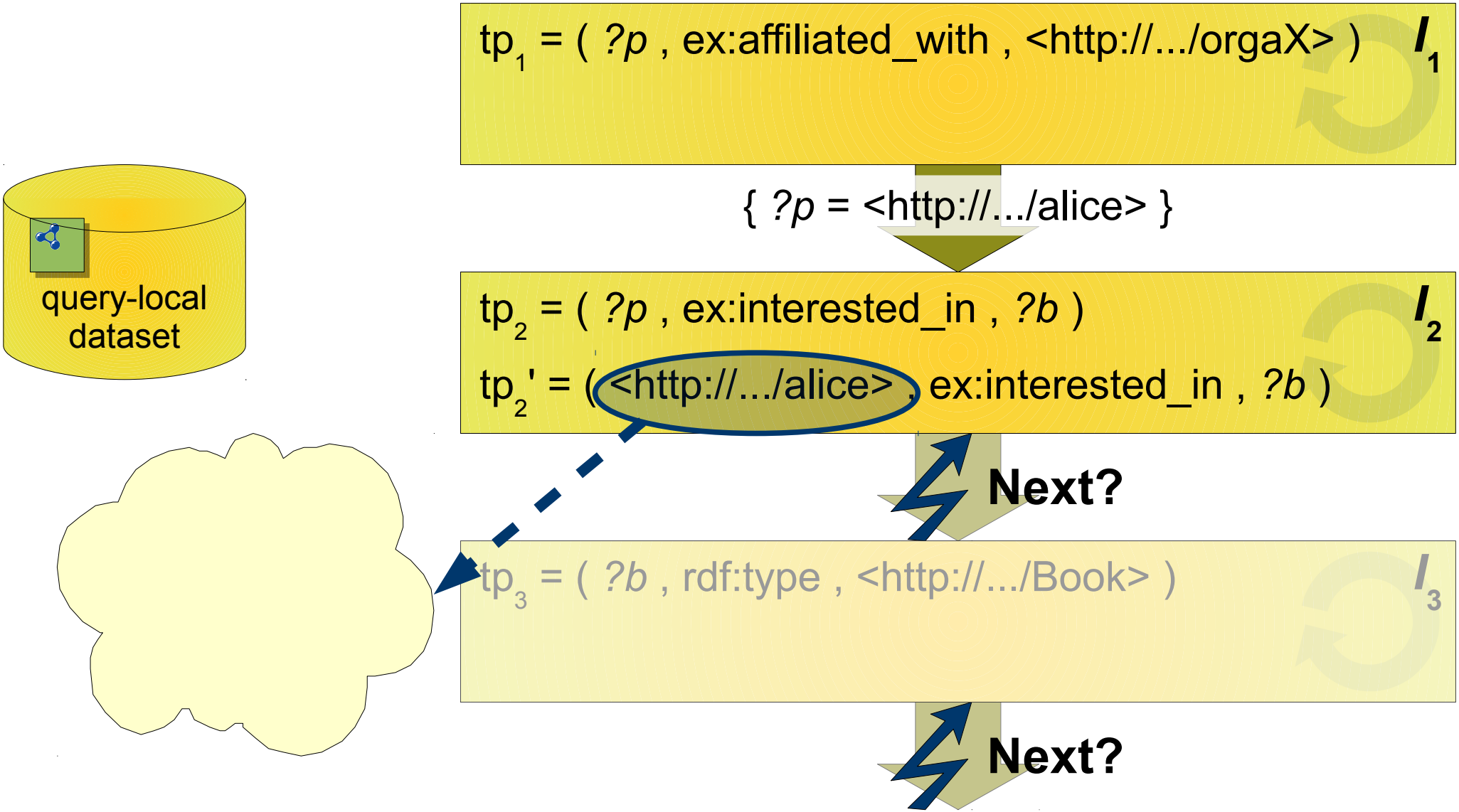
{ *?p* = <http://.../alice> , *?b* = <http://.../b1> }

tp_3 = (*?b* , rdf:type , <http://.../Book>)
tp_3' = (<http://.../b1> , rdf:type , <http://.../Book>) I_3

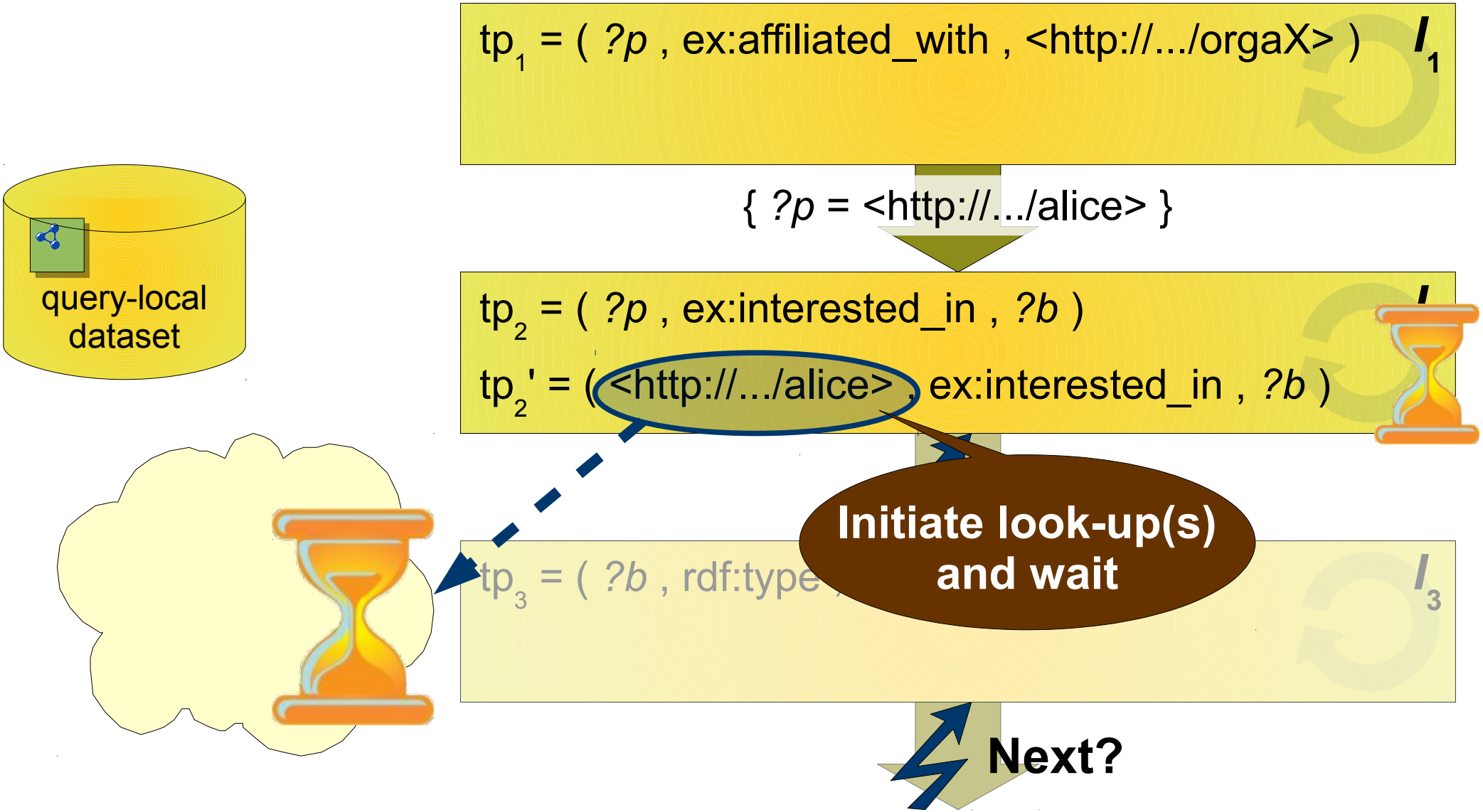
Outline

- **Heuristics-Based Planning** √
- **Optimizing Link Traversing Iterators**
  - Prefetching
  - Postponing
- **Source Ranking**
  - Harth et al. [HHK+10, UHK+11]
  - Ladwig and Tran [LT10]

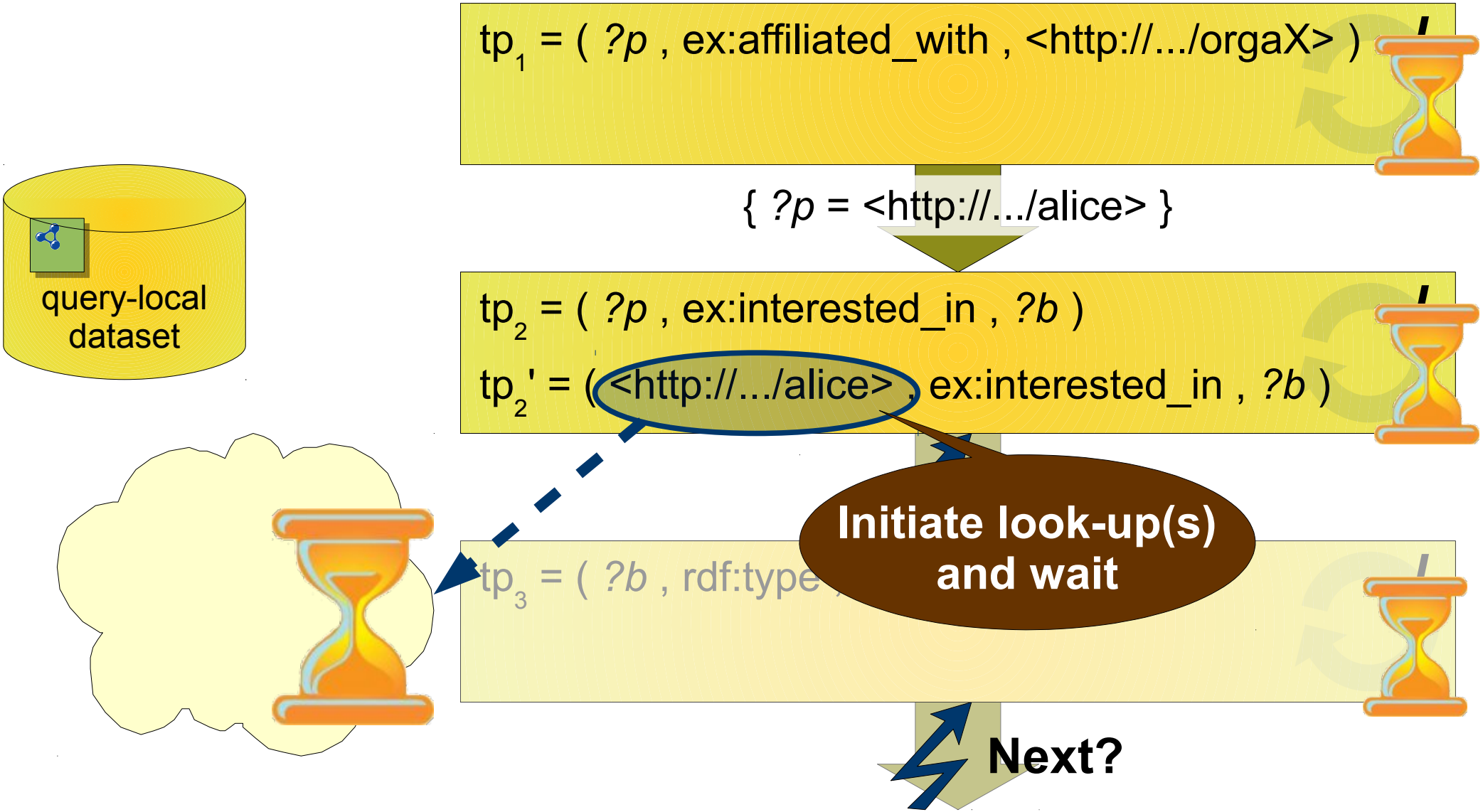
Link Traversing Iterators May Block!



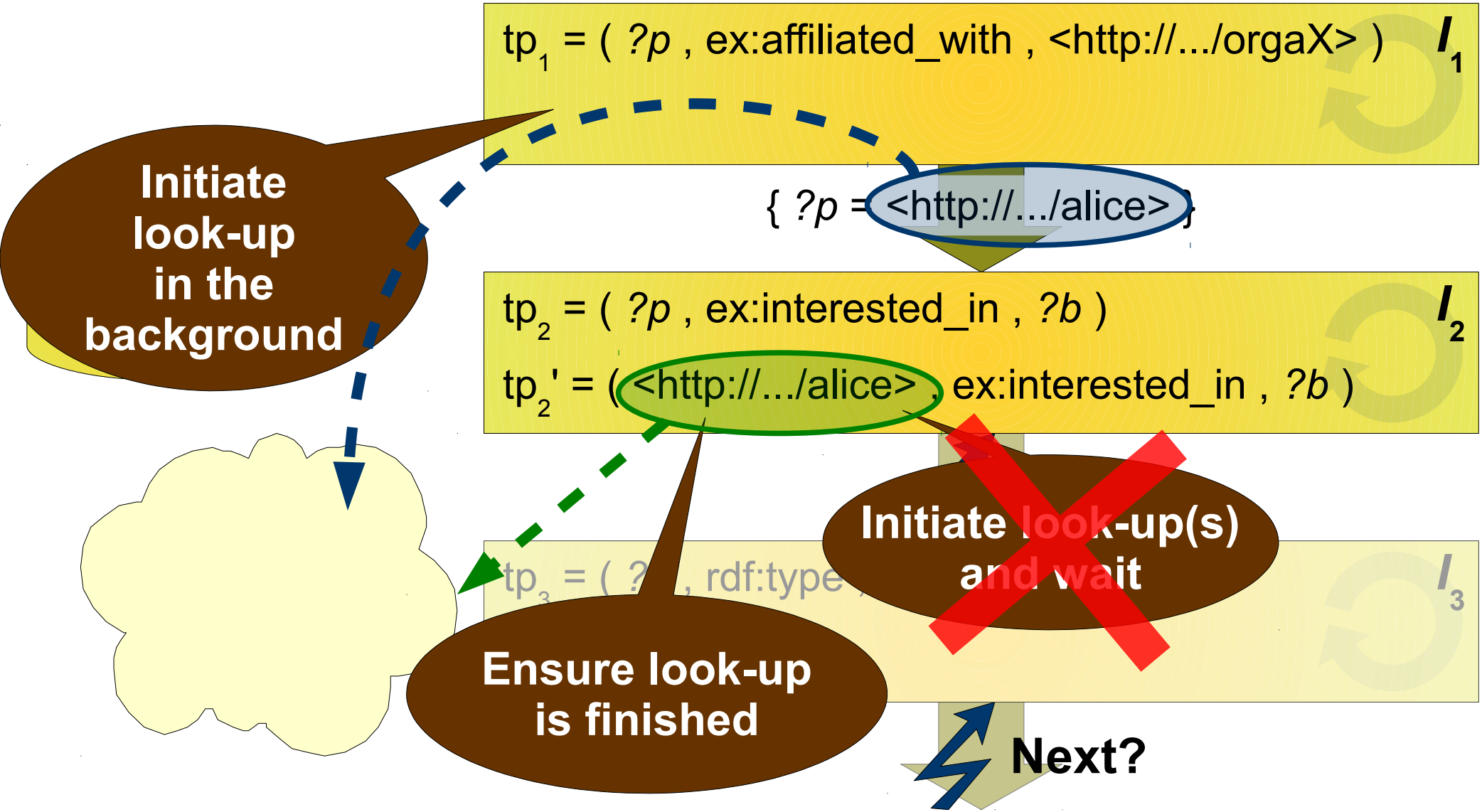
Link Traversing Iterators May Block!



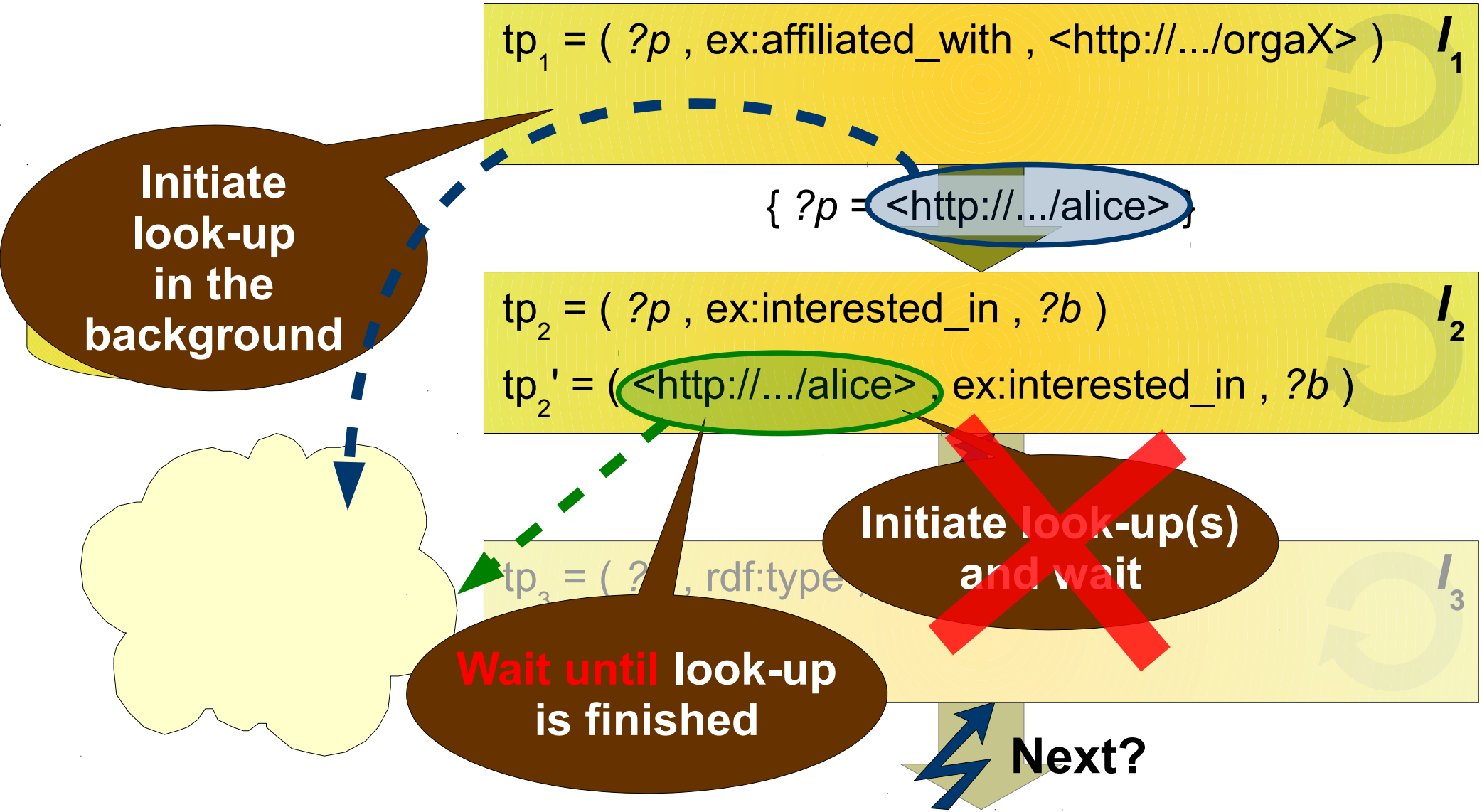
Link Traversing Iterators May Block!



Prefetching of URIs [HBF09]



Prefetching of URIs [HBF09]



Postponing Iterator [HBF09]

- **Idea: temporarily reject an input solution if processing it would cause blocking**

- **Enabled by an extension of the iterator paradigm:**
  - New function POSTPONE: treat the element most recently reported by GETNEXT as if it has not yet been reported (i.e., "take back" this element)
  - Adjusted GETNEXT: either return a (new) next element or return a formerly postponed element

Outline

- **Heuristics-Based Planning** √
- **Optimizing Link Traversing Iterators**
  - Prefetching
  - Postponing √
- **Source Ranking**
  - Harth et al. [HHK+10, UHK+11]
  - Ladwig and Tran [LT10]

General Idea of Source Ranking

Rank the URIs resulting from source selection

such that

the ranking represents a priority for lookup

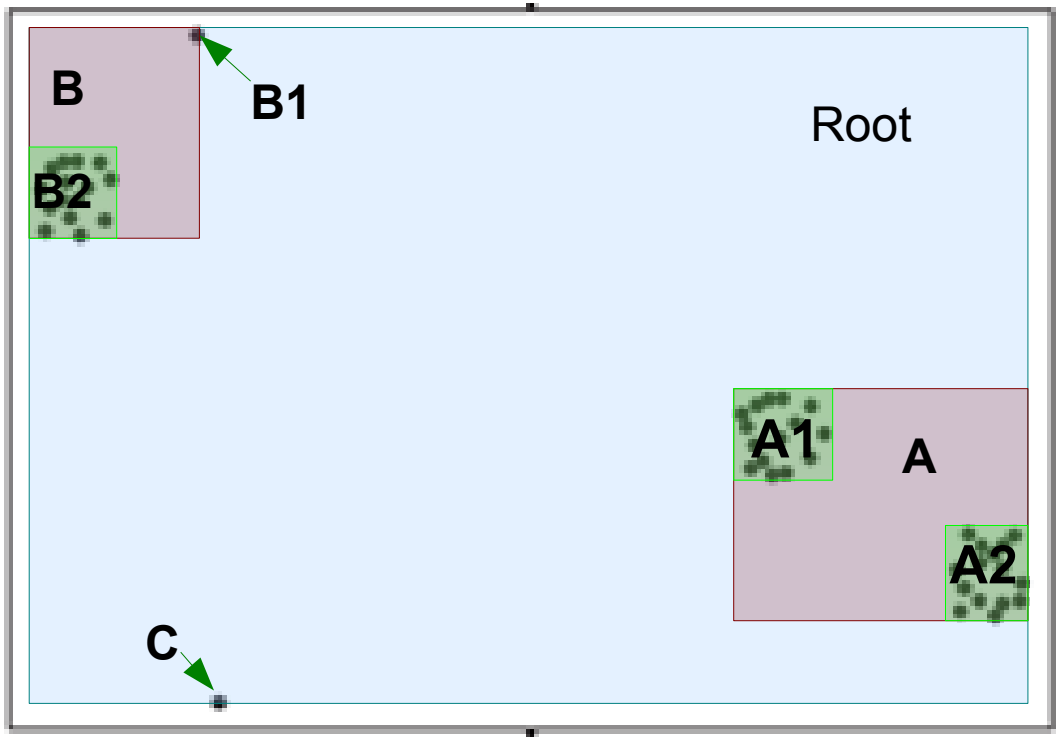
- **Possible objectives:**
  - Report first solutions as early as possible
  - Minimize time for computing the first *k* solutions
  - Maximize the number of solutions computed in a given amount of time

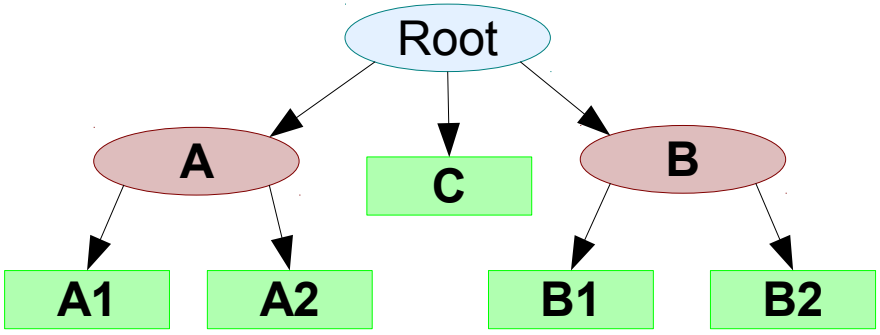
Harth et al. [HHK+10, UHK+11]

For any URI u (selected by the QTree-based approach), let:

$$rank(u) := \text{estimated number of solutions that } u \text{ contributes to}$$

- **For triple patterns this number is directly available:**
  - Recall, each QTree bucket stores a set of (URI,count)-pairs
  - All query-relevant buckets are known after source selection





Harth et al. [HHK+10, UHK+11]

For any URI u (selected by the QTree-based approach), let:

$$rank(u) := \text{estimated number of solutions that } u \text{ contributes to}$$

- **For triple patterns this number is directly available:**
  - Recall, each QTree bucket stores a set of (URI,count)-pairs
  - All query-relevant buckets are known after source selection
- **For BGPs, estimate the number recursively:**
  - Recursively determine regions of join-able data (based on overlapping QTree buckets for each triple pattern)
  - For each of these regions, recursively estimate number of triples the URI contributes to the region
  - Factor in the estimated join result cardinality of these regions (estimated based on overlap between contributing buckets)

Ladwig and Tran [LT10]

- **Multiple scores**
  - Triple pattern cardinality
  - Triple frequency – inverse source frequency (TF–ISF)
  - (URI-specific) join pattern cardinality
  - Incoming links
- **Assumption: pre-populated index that stores triple pattern cardinalities and join pattern cardinalities for each URI**
- **Aggregation of the scores to obtain ranks**
  - For indexed URIs: weighted summation of all scores
  - For non-indexed URIs: weighting of (currently known) in-links
- **Ranking is refined at run-time**

Metric: Triple Pattern Cardinality [LT10]

For a selected URI *u*, and a triple pattern *tp* (from the query), let:

$$card(u, tp) := \text{number of triples in the data of } u \text{ that match } tp$$

- **Rationale:** data that contains many matching triples is likely to contribute to many solutions
- **Requirement:** pre-populated index that stores the cardinalities
- **Caveat:** some triple patterns have a high cardinality for almost all URIs
  - Example: (*?x*, rdf:type, *?y*)
  - These patterns do not discriminate URIs

Metric: TF–ISF [LT10]

- **Idea: adopt TF-IDF concept to weight triple patterns**
- **Triple Frequency – Inverse Source Frequency (TF–ISF)**

For a selected URI *u*, a triple pattern *tp*, and a set of all known URIs U_{known}, let:

$$tf.isf(u, tp) := card(u, tp) * \log\left(\frac{|U_{known}|}{\{r \in U_{known} \mid card(r, tp) > 0\}}\right)$$

- **Rationale:**
  - Importance positively correlates to the number of matching triples that occur in the data for a URI
  - Importance negatively correlates to how often matching triples occur for all known URIs (i.e., all indexed URIs)

Metric: Join Pattern Cardinality [LT10]

For a selected URI u, two triple pattern tp_i and tp_j, and query variable v, let:

$$card(u, tp_i, tp_j, v) := \text{number of solutions produced by joining } tp_i \text{ and } tp_j \text{ on variable } v \text{ using only the data from } u$$

- **Rationale:** data that matches pairs of (joined) triple patterns is highly relevant, because it matches a larger part of the query
- **Requirement:** these join cardinalities are also pre-computed and stored in a pre-populated index

Ladwig and Tran [LT10]

- **Multiple scores**
  - Triple pattern cardinality
  - Triple frequency – inverse source frequency (TF–ISF)
  - (URI-specific) join pattern cardinality
  - Incoming links
- **Assumption: pre-populated index that stores triple pattern cardinalities and join pattern cardinalities for each URI**
- **Aggregation of the scores to obtain ranks**
  - For indexed URIs: weighted summation of all scores
  - For non-indexed URIs: weighting of (currently known) in-links
- **Ranking is refined at run-time**

Refinement at Run-Time [LT10]

- **During query execution information becomes available**

  (1) intermediate join results (2) more incoming links

- **Use it to adjust scores & ranking (for integrated execution)**
  - Re-estimate join pattern cardinalities based on samples of intermediate results (available from hash tables in SHJ)

- **Parameters for influencing behavior of ranking process:**
  - Invalid score threshold: re-rank when the number of URIs with invalid scores passes this threshold
  - Sample size: larger samples give better estimates, but make the process more costly
  - Re-sampling threshold: reuse cached estimates unless the hash table of join operators grows past this threshold

Outline

- **Heuristics-Based Planning** √
- **Optimizing Link Traversing Iterators**
  - Prefetching
  - Postponing √
- **Source Ranking**
  - Harth et al. [HHK+10, UHK+11]
  - Ladwig and Tran [LT10] √

Tutorial Outline

(1) Introduction

(2) Theoretical Foundations

(3) Source Selection Strategies

(4) Execution Process

(5) Query Planning and Optimization

… Thanks!

These slides have been created by
Olaf Hartig
for the
WWW 2013 tutorial on
Link Data Query Processing

Tutorial Website: http://db.uwaterloo.ca/LDQTut2013/

(Some of the slides in this slide set have been inspired by slides from Günter Ladwig [LT10] – Thanks!)

These slides have been created by
Olaf Hartig
for the
WWW 2013 tutorial on
Link Data Query Processing

Tutorial Website: http://db.uwaterloo.ca/LDQTut2013/

(Slides 24 - 26, 33, and 34 are inspired by slides
from Günter Ladwig [LT10] – Thanks!)

Backup Slides

Metric: Links to Results [LT10]

The "links to results" of a selected URI *u* is defined by:

$$links(u) := \{ l \in links(u, u_{processed}) \mid u_{processed} \in U_{processed} \}$$

where $U_{processed}$ is the set of URIs whose data has already been processed and $links(u_1, u_2)$ are the links to URI u_1 mentioned in the data from URI u_2.

- **Rationale:** a URI is more relevant if data from many relevant URIs mention it
- **Links are only discovered at run-time**

Metric: Retrieval Cost [LT10]

The retrieval cost of a selected URI u is defined by:

$$cost(\ u\ ) := \mathrm{Agg}(\ size(u)\ ,\ bandwidth(u)\ )$$

where $size(u)$ is the of the data from u, and $bandwidth(u)$ is the bandwidth of the Web server that hosts u.

- **Rationale: URIs are more relevant the faster their data can be retrieved**
- **Size is available in the pre-populated index**
- **Bandwidth for any particular host can be approximated based on past experience or average performance recorded during the query execution process**