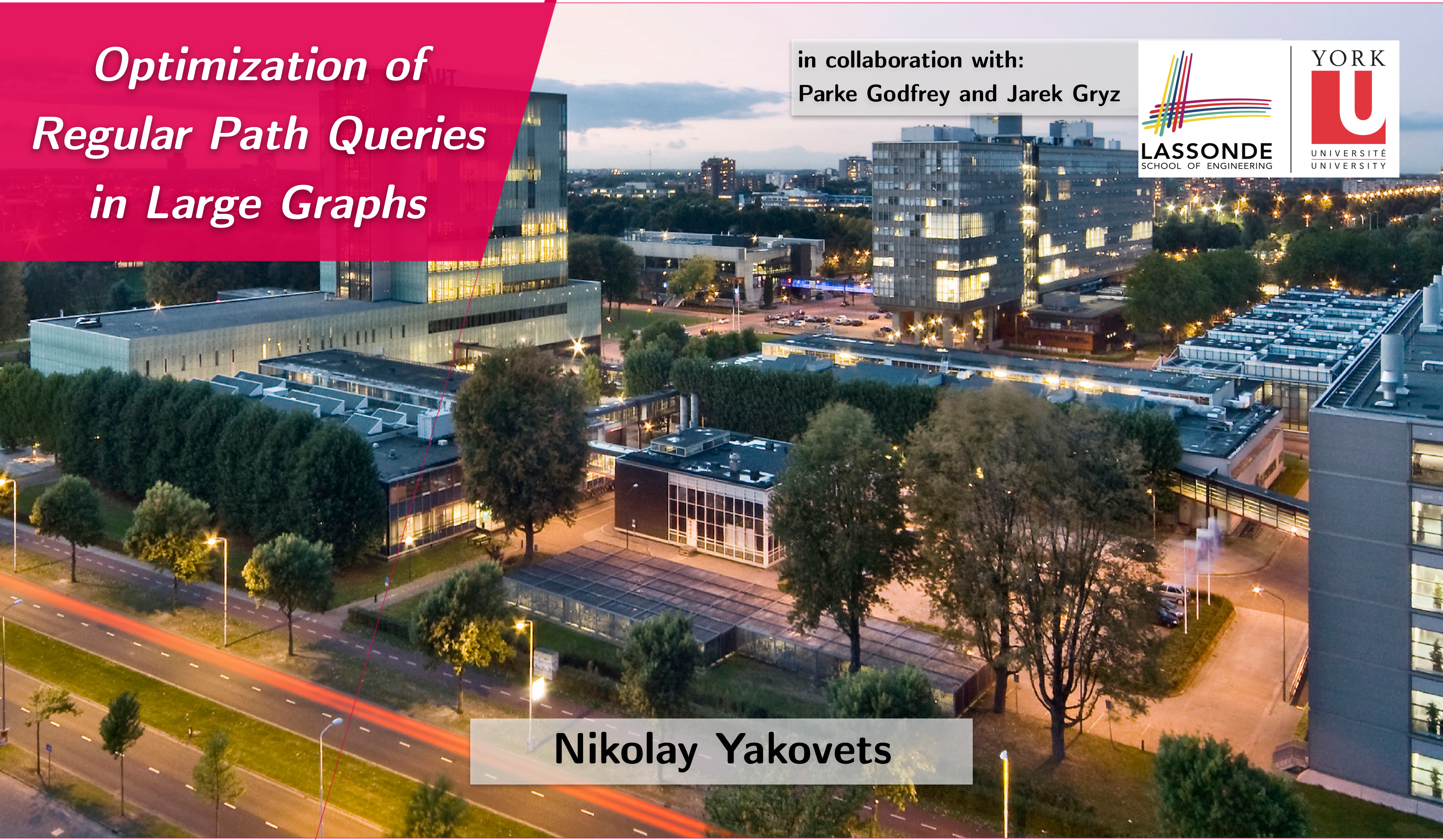
Technische Universiteit **Eindhoven** University of Technology

Optimization of Regular Path Queries in Large Graphs



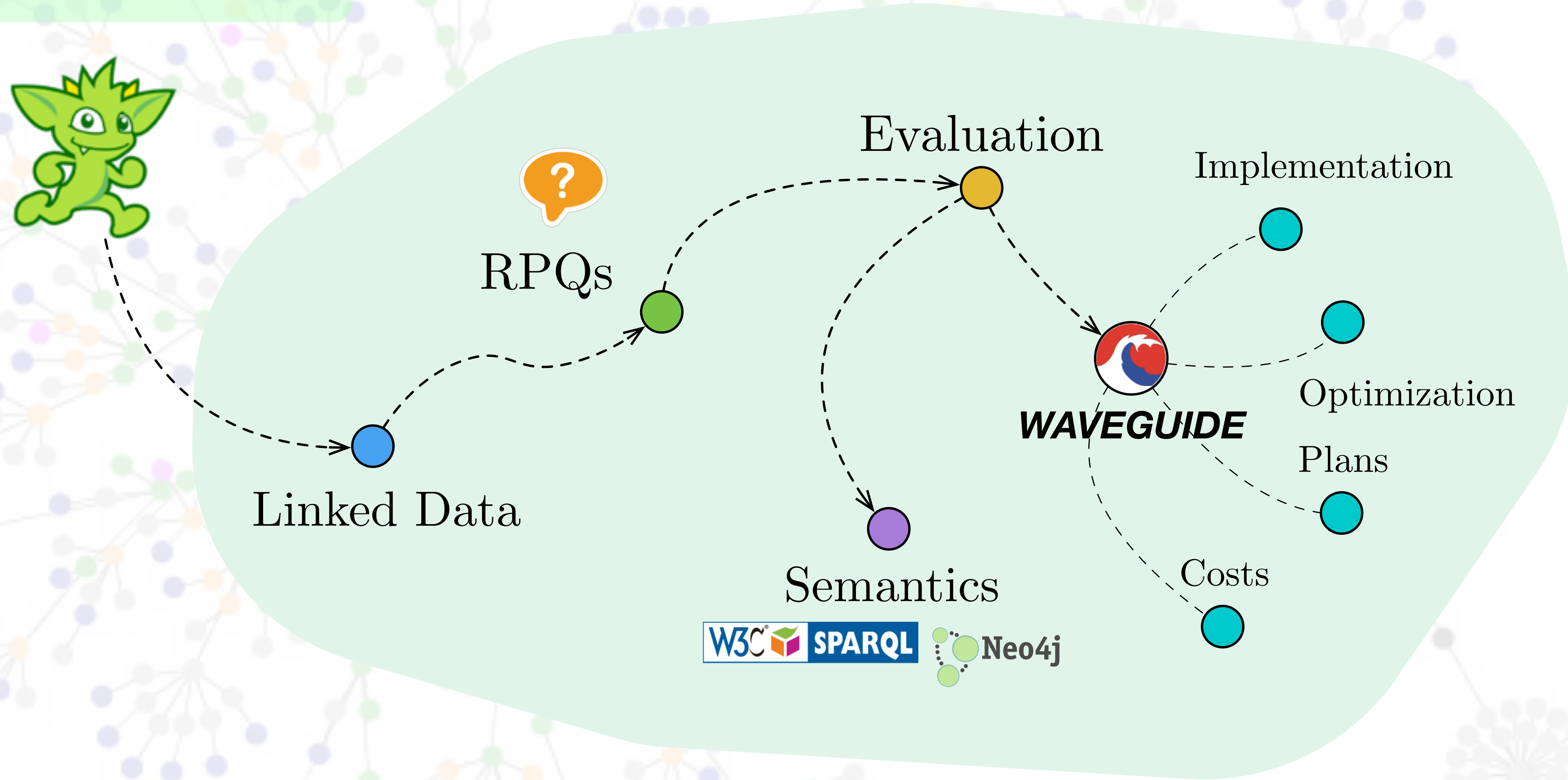
in collaboration with:
Parke Godfrey and Jarek Gryz

LASSONDE SCHOOL OF ENGINEERING
YORK UNIVERSITÉ UNIVERSITY

Nikolay Yakovets

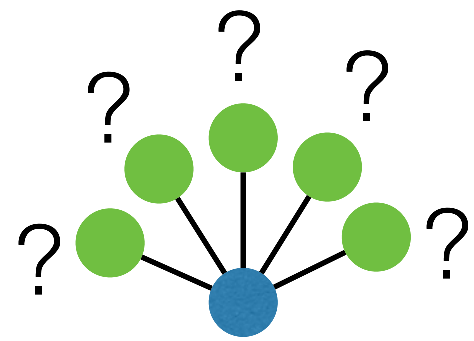
Optimization of RPQs

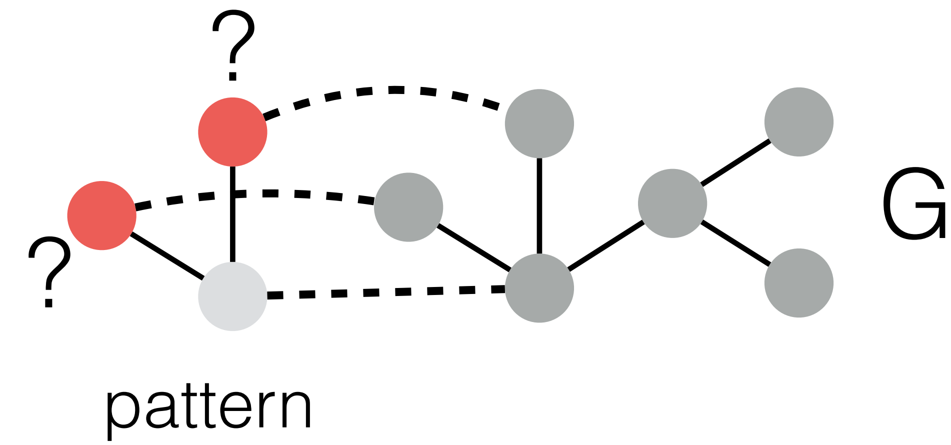
Scalable & efficient evaluation of **regular path queries**



Graph Query Languages

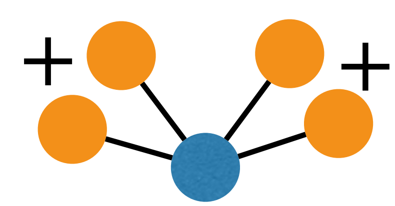
* **Adjacency** Query
  list all neighbours, find k-neighbourhood of a node



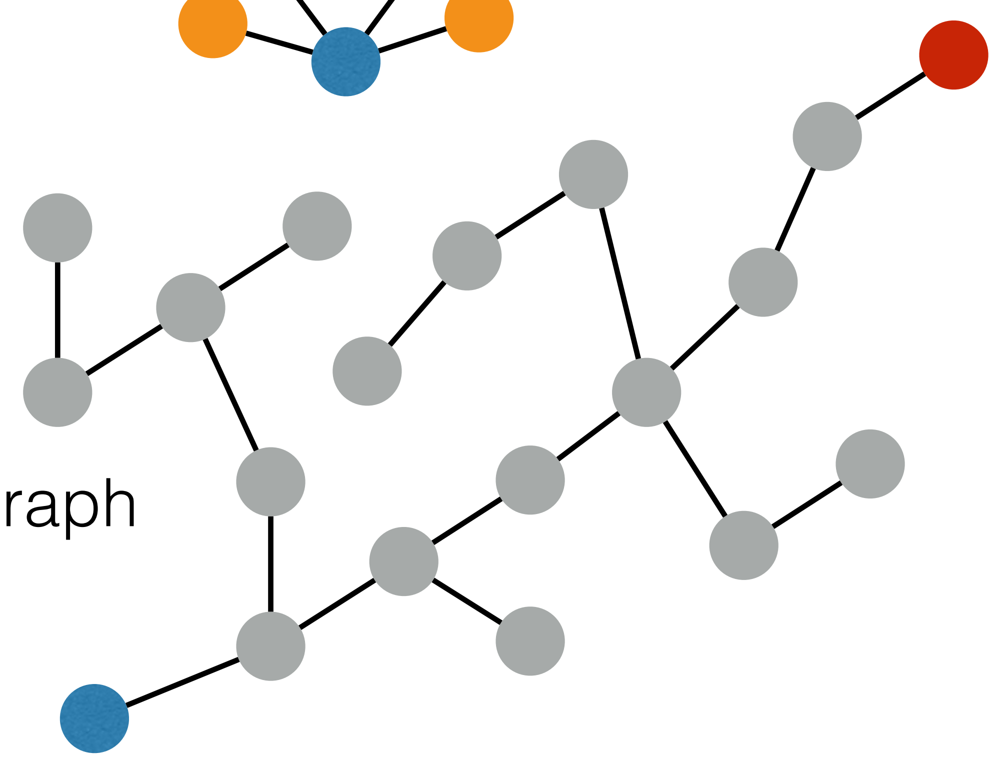


* **Pattern Matching** Query
  find all sub-graphs in a database that are isomorphic to a given query pattern graph

* **Summarization** Query
  summarize or operate on query results
  e.g. aggregation; avg(), min(), max(), etc



* **Reachability/Path** Query
  navigational query
  deals with paths in a graph
  test whether nodes are reachable in a graph
  paths of fixed or arbitrary lengths



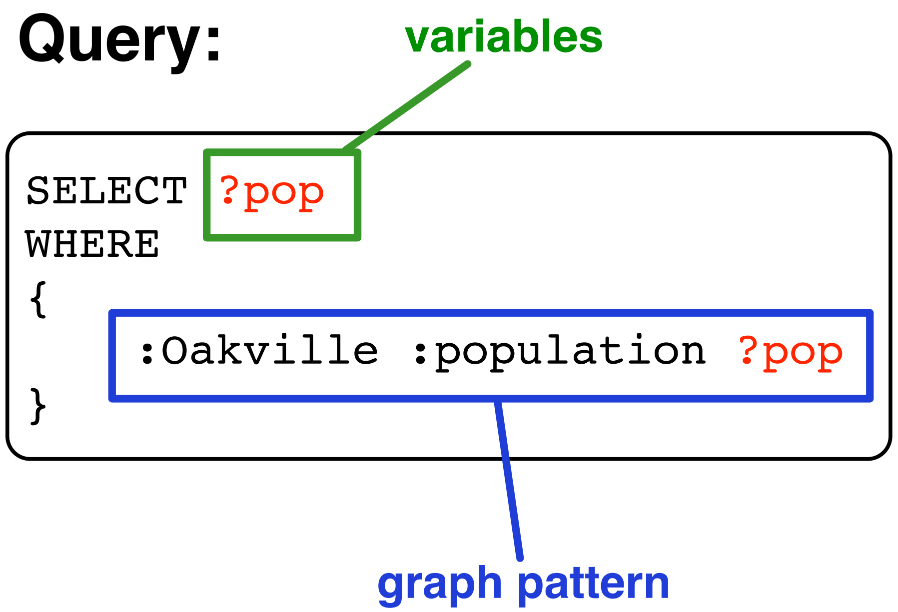
SPARQL - Query Language

✓ adjacency ✓ pattern matching ✓ summarization

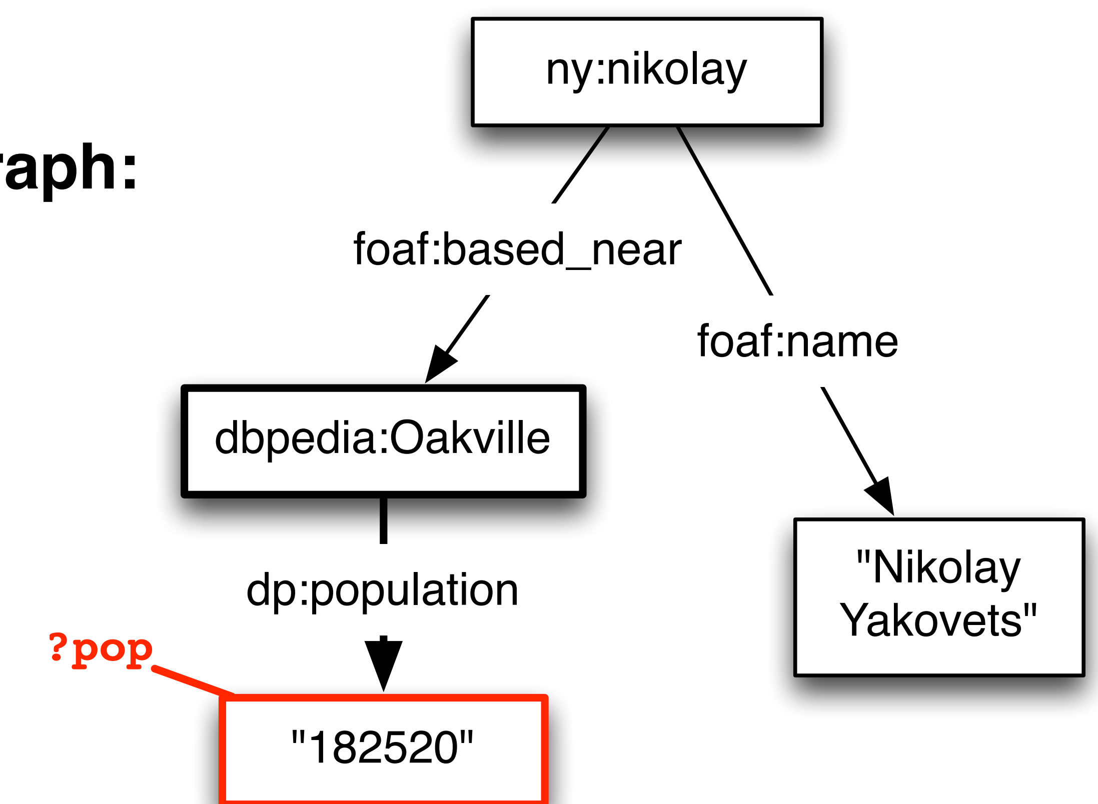
SPARQL Protocol and RDF Query Language (SPARQL)

- declarative, based on **pattern matching**
- **graph patterns** describe subgraphs of the queried RDF graphs
- those subgraphs that match a description yield a result

Query:



Graph:



SPARQL Property Paths

- Part of SPARQL 1.1 W3C recommendation
- Allow **regular expressions** to describe paths between nodes:

$p_1|p_2$ disjunction

p_1/p_2 concatenation

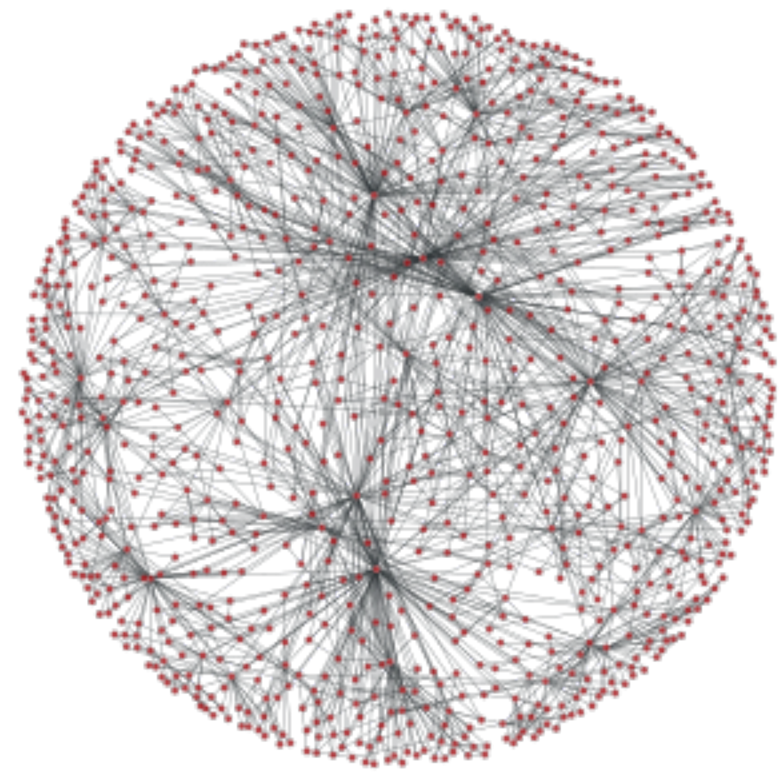
$p?$ zero or one

$\hat{}p$ inverted

$!iri$ negated

$p*$ Kleene star

$p+$ one or more

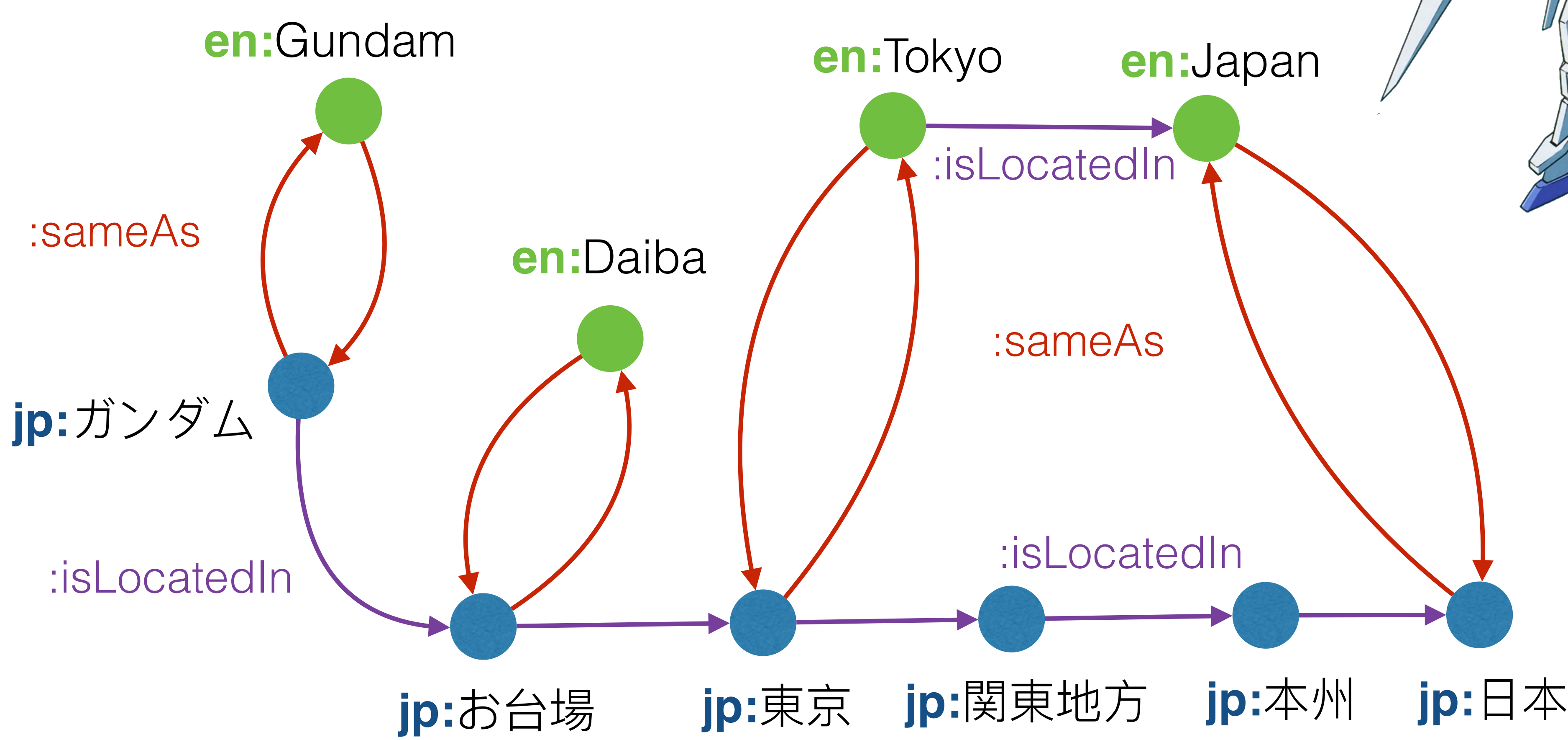


- **Useful** in many application domains: **social networks**, **biological**, **encyclopedic**
- Convenient declarative mechanism to answer queries **without prior knowledge of underlying data paths**

SPARQL Property Paths

- **Example:** DBPedia snippet, part of a LOD dataset
- Two datasets **English** and **Japanese interlinked** with OWL terms

G:



Q: select **?place**
{ **en:**Gundam (:sameAs*/:isLocatedIn)+/sameAs* **?place** .}

- **Query:** Where is Gundam statue located?
- **Solution:** Need to resolve equivalent data entities (**:sameAs**) and traverse spacial hierarchy (**:isLocatedIn**) to fully utilize richer spacial information in Japanese dataset

Formal Evaluation

- **Property Paths** in SPARQL are essentially **Regular Path Queries** (RPQs)
- RPQs have been well-studied before the advent of RDF and SPARQL

- **Formal def.:**

regular language

$$Q = (x, L(r), y)$$

free variables

- **Semantics of Evaluation:**

$[[Q]]_G$ - an evaluation of Q over graph database G

a collection (s, t) such that
$\exists$ a path p in G between s and t
such that p conforms to regex r

a **bag** (allow duplicates) $\forall$
aka. *solution counting*

a **set** (discard duplicates) $\exists$
aka. *existential semantics*

path-induced string $\lambda(p) \in L(r)$
path is *simple* or *arbitrary*

Paths in SPARQL

simple $\forall$ | **simple** $\exists$ | **regular** $\forall$

- Evaluation of **simple** paths is *NP-complete* on general graphs (Mendelzon et al., **1987**)
  - Tractable on DAGs, or restricted *compatible* regex
- **Counting** procedures are *#P-complete* on general graphs (Arenas et al., Losemann et al., **2013**)
  - Tractable on DAGs, or restricted *compatible* regex

regular $\exists$

SPARQL (W3C proposal for RDF query language)

- support of RPQs through SPARQL1.1 property paths

RPQ Evaluation

$[[Q]]_G$ - an evaluation of Q over graph database G

+

considering **existential** semantics on **regular** paths

FA-based

- Use **finite state machines** in evaluation
- *Mendelzon et al., 1987*

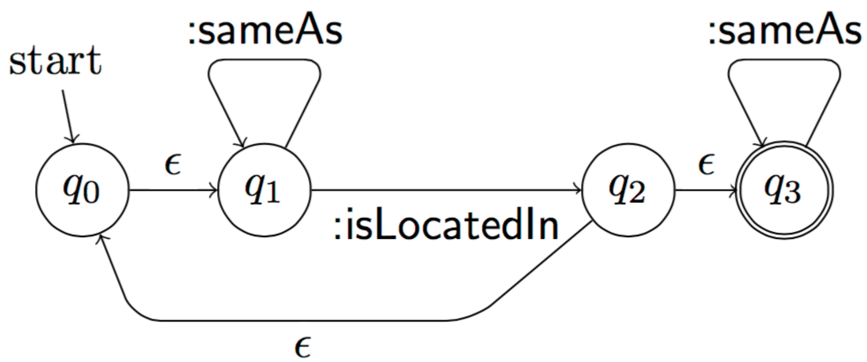
α-RA-based

- Use **relational algebra** extended with **alpha-operator** which computes transitive closure
- *Losemann et al., 2013*

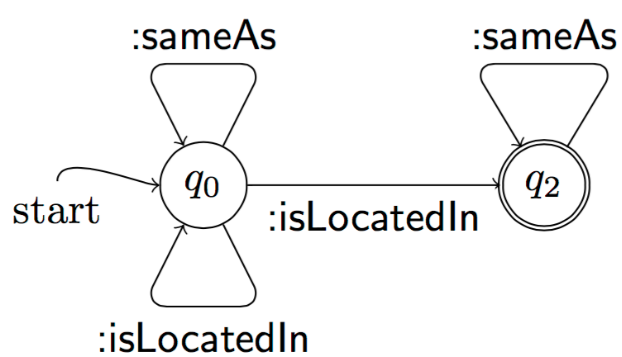
FA-based Evaluation

Q: select **?place**
{ **en:**Gundam (:sameAs*/:isLocatedIn)+/sameAs* **?place** .}

1. From a parse tree, construct a query **ε-NFA**:

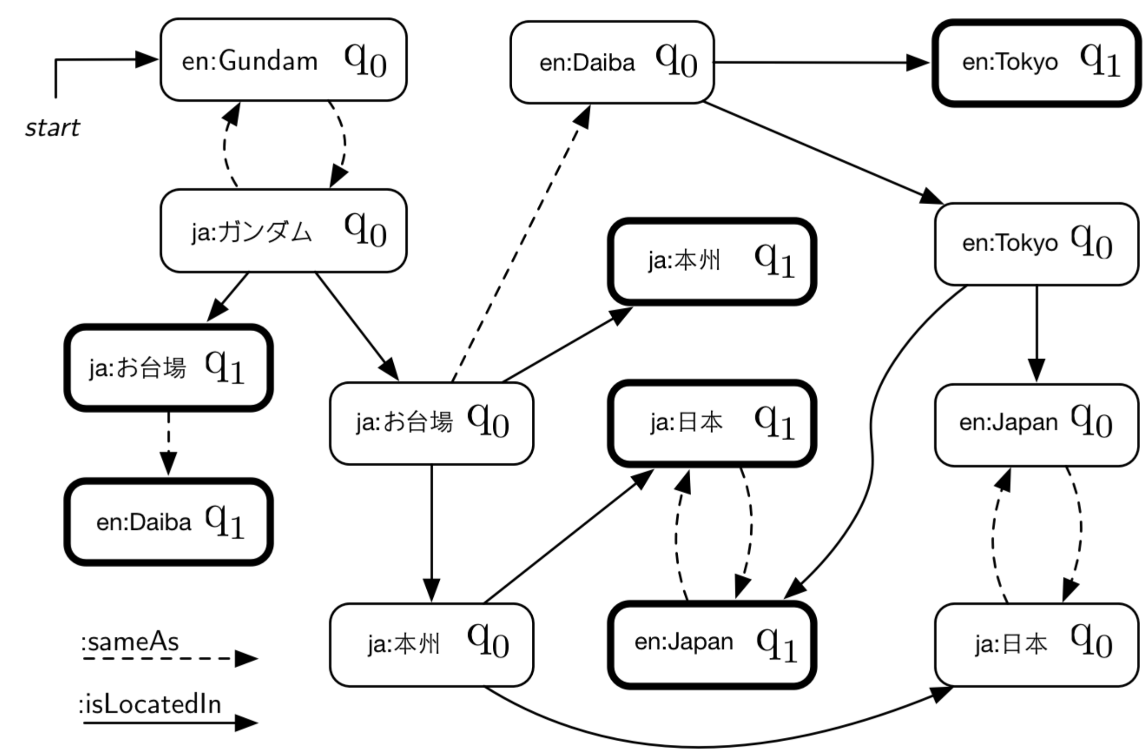


2. **Minimize** the query automaton, if necessary :



3. Construct a product ***P*** of query and graph automata.

4. Check ***P*** for reachable accepting states to produce an answer to a query



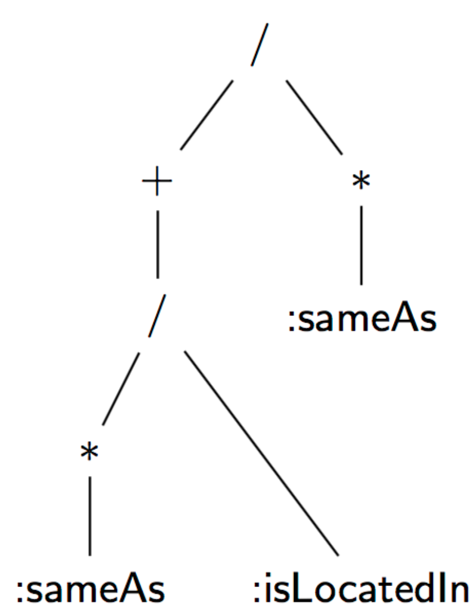
α-RA-based Evaluation

Q: select **?place**
{ **en:**Gundam (:sameAs*/:isLocatedIn)+/sameAs* **?place** .}

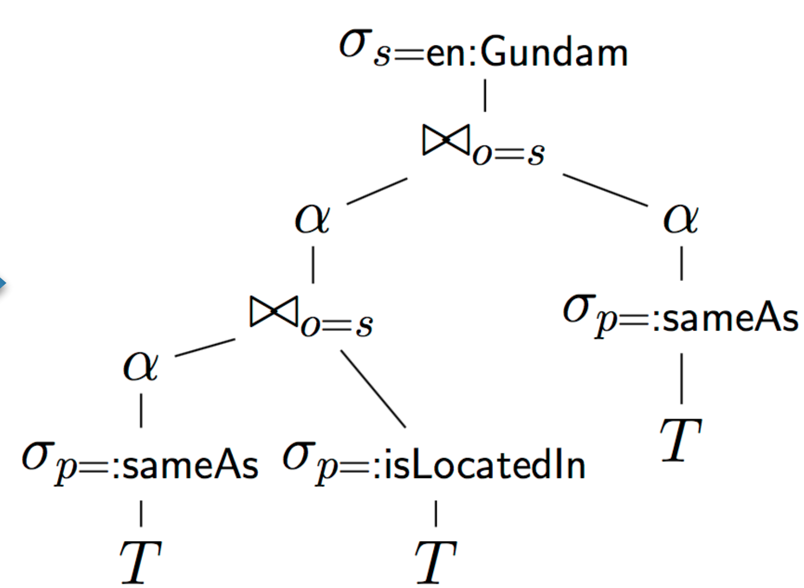
$$T = \pi_{1,3}(G)$$

- Have **SPRJU-RA** extended with **α**
- **α** computes the least-fixpoint: $T^+ = T \cup \pi_{1,3}(T^+ \bowtie_{T^+.o=T.s} T)$
- **α** computes the **transitive closure** of a given relation

1. From a parse tree, construct an RA tree:



Q parse tree



Q RA tree

favourite **RDBMS**

Comparing Approaches

Th: FA and are α-RA *incomparable*

Pf.:

- translation into Datalog
- examine induced sequence of joins

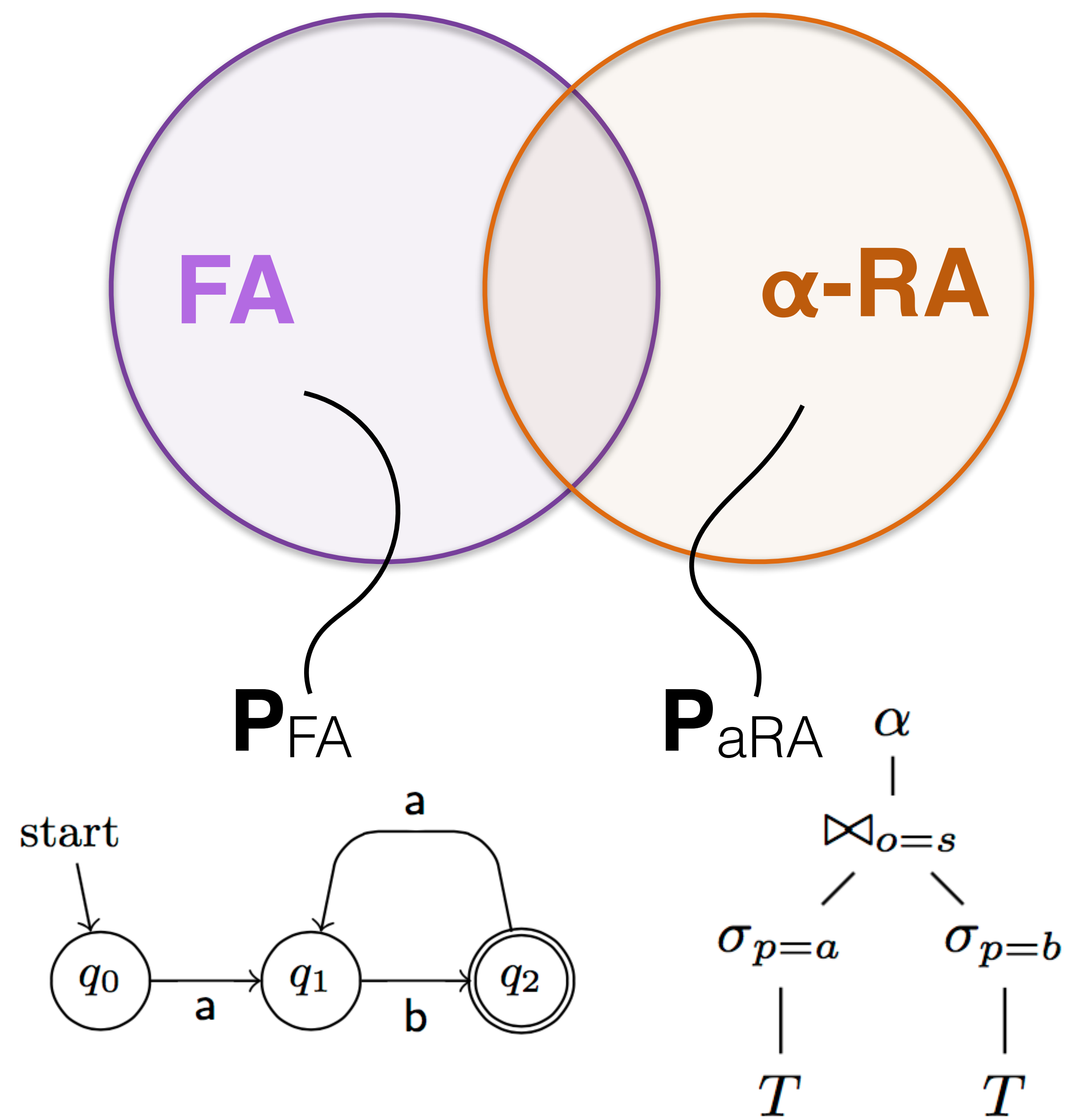
e.g. **(?x, (a/b)+, ?y)**

- **P**FA =((((a⋈b)⋈a)⋈b)⋈a)..
- **P**aRA =(a⋈b)⋈(a⋈b)⋈(a⋈b)..

PaRA ∉ **FA** ⇓ **α-RA** ⊄ **FA**

PFA ∉ **α-RA** ⇓ **FA** ⊄ **α-RA**

plan spaces



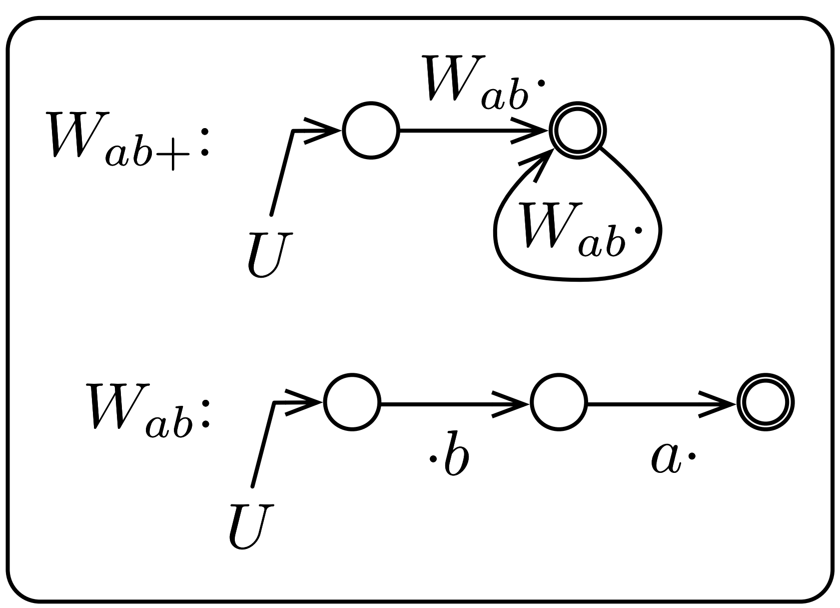
WAVEGUIDE

Goal: Need to consider both FA and α-RA plan spaces

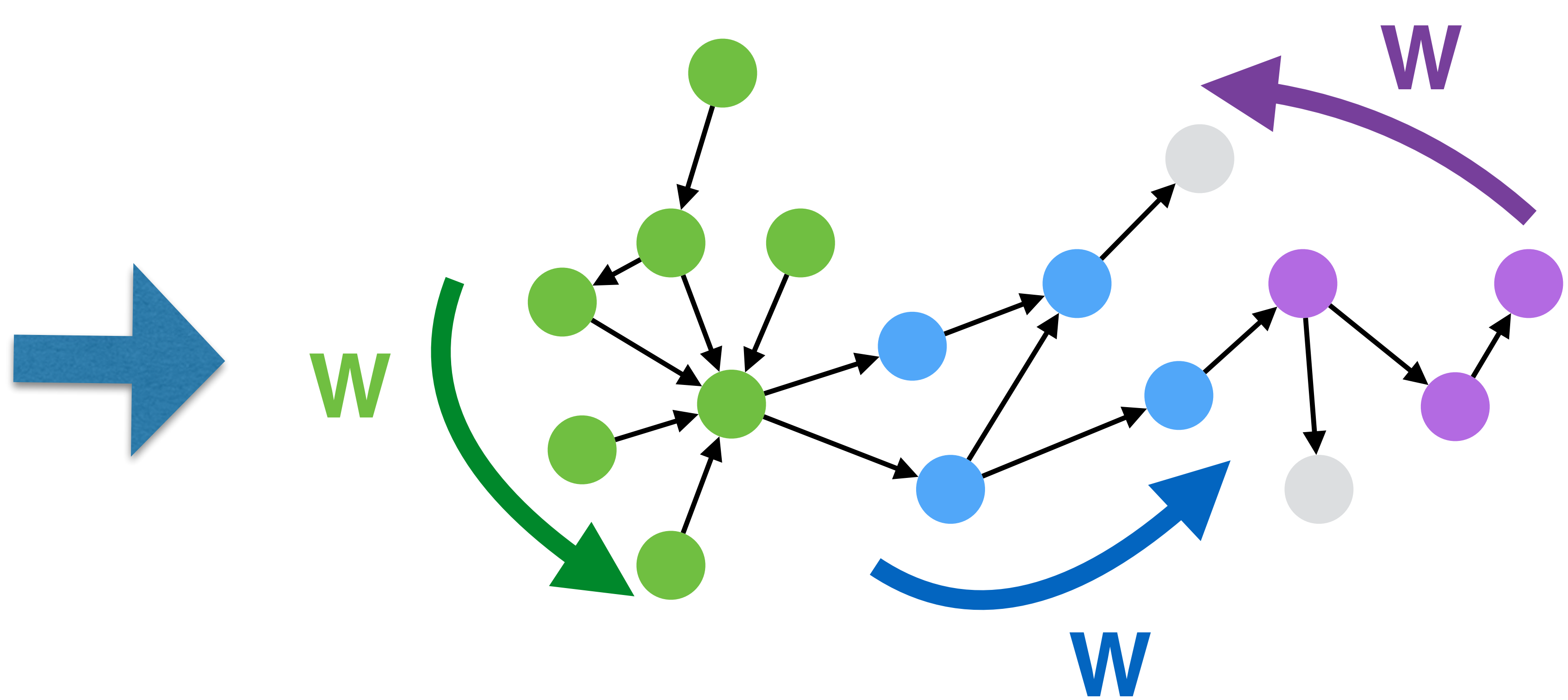
- Search driven by a **waveplan** which **guides** a number of **wavefronts** which **iteratively explore the graph**

waveplan

P_{ab+}



guided iterative graph search



search wavefronts

a **wavefront** W_l

- an expanding search unit
- guided by a **wavefront automaton**
- **labeled** with regex it evaluates
- **seeded** with S

seed, label, edge labels, wavefront labels, accepting states

$$W_l = (l, S, q_0, Q, \delta, E, L, F)$$

starting state, set of states, transition function

a **transition function** δ

- appending and prepending transitions
- transitions over graphs and views

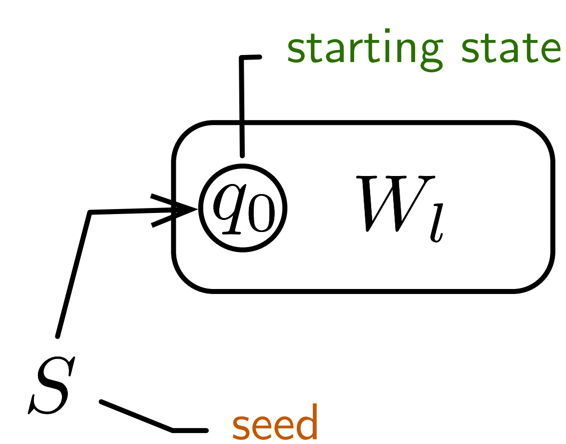
appending or prepending

$$\delta : Q \times ((E \cup L) \times \{\cdot\_, \_\cdot\} \cup \{\varepsilon\}) \to 2^Q$$

graph edges or wavefront labels, pipeline

a **seed** S

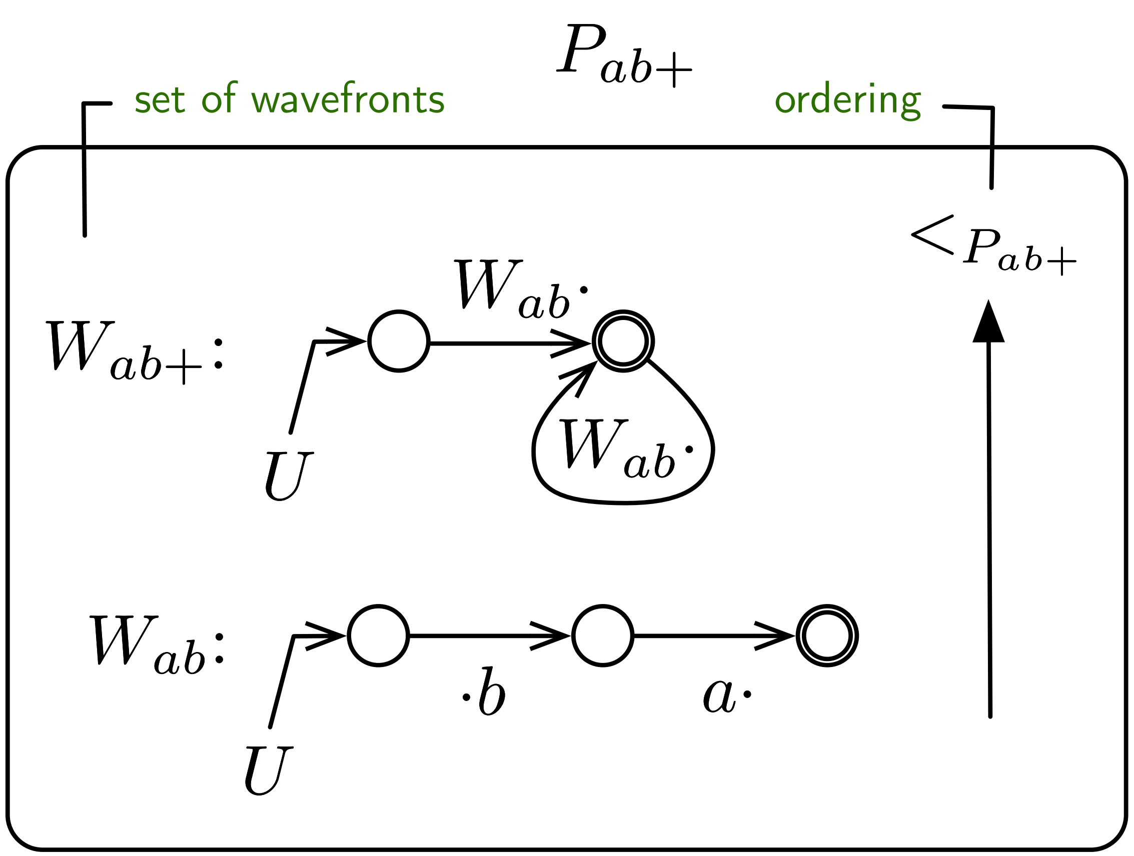
- edge incoming into accepting state in W_l
- defined with an RPQ, a wavefront or by construction
- can be **universal**, any node in a graph



a waveplan

a **waveplan** P_Q

- produces an answer to a given query Q
- an **ordered** set of wavefront automata
- order defines **which labels** can be used in the seed and transitions over a view
- **higher wavefronts** can use lower wavefronts as their labels and seeds, but not vice-versa
- **query answered by the highest wavefront**



e.g., query **(?x, (a/b)+, ?y)**

- W_{ab} produces an answer for (a/b) regex
- W_{ab+} uses W_{ab} as a view to compute (a/b)+

WAVEGUIDE - iterative search

- Exploration procedure based on **semi-naive evaluation**
- **Intermediate** search results kept in the **search cache**
- **cache** keeps track of end-nodes and corresponding states in a plan

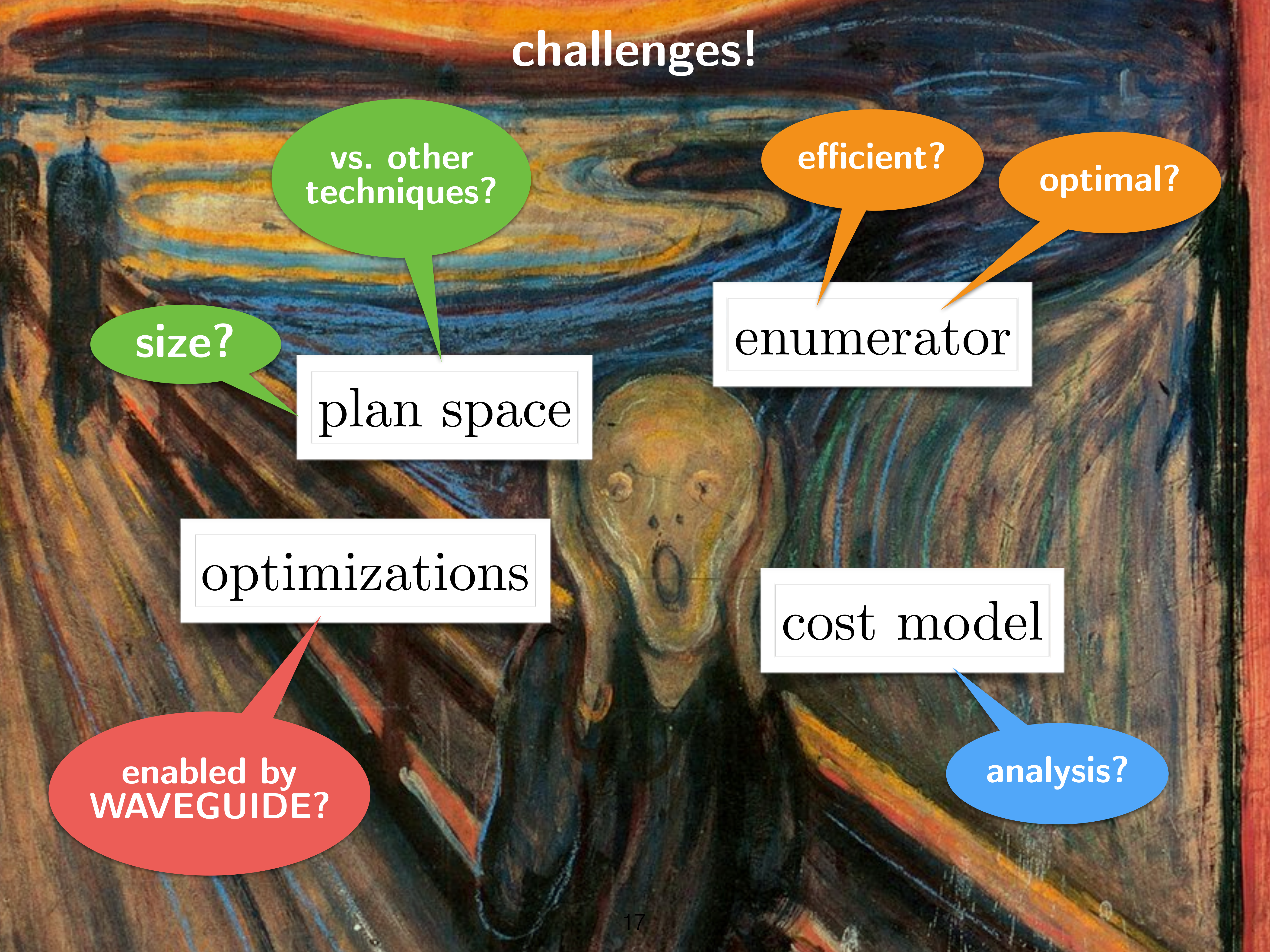
$\Delta_0^R \leftarrow$ seed(G);
$i \leftarrow 0$;
while $|\Delta_i^R| \geq 0$ do
  $\Delta_{i+1}^S \leftarrow$ seed(Δ_i^R);
  $\Delta_{i+1}^C \leftarrow$ crank$(\Delta_{i+1}^S, \Delta_i^R, G, C_i, A_Q)$;
  $\Delta_{i+1}^R \leftarrow$ reduce$(\Delta_{i+1}^C, \Delta_i^R, C_i)$;
  $C_{i+1} \leftarrow$ cache(Δ_{i+1}^R, C_i);
  $i \leftarrow i + 1$;
done;
return extract(C_i);

- **seed** specifies node pairs to start from

loop while discover new tuples

- **crank** *advances* simultaneously in a graph and automaton
- **reduce** *prunes* the delta, handles unbounded computation
- **cache** materializes according to the specified strategy
- **extract** produces answers

challenges!

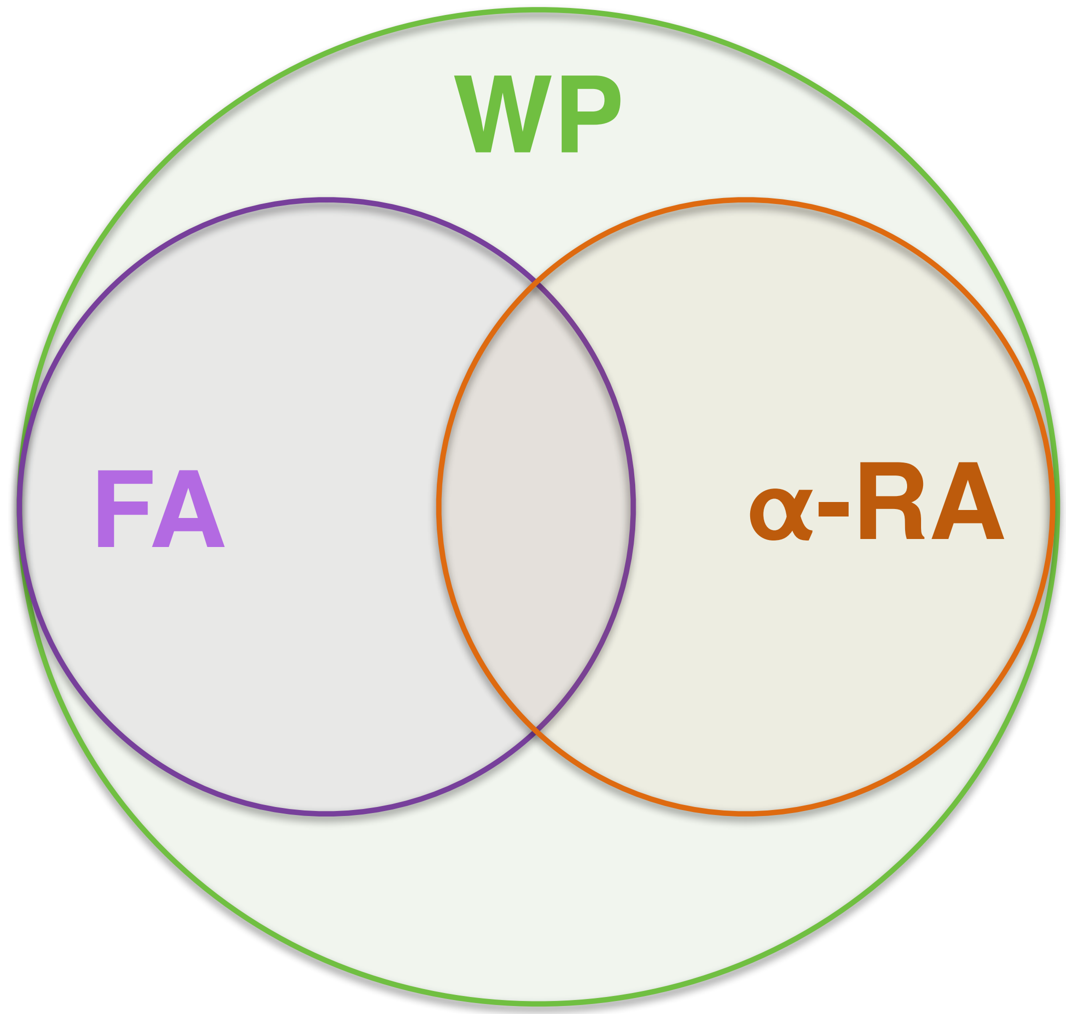

vs. other techniques?
efficient?
optimal?
size?
plan space
enumerator
optimizations
cost model
enabled by WAVEGUIDE?
analysis?

WAVEGUIDE Plan Space

- **subsumes** both FA and α-RA
- adds **exclusive** new plans

$$\alpha\text{-RA} \cup \text{FA} \subset \text{WP}$$

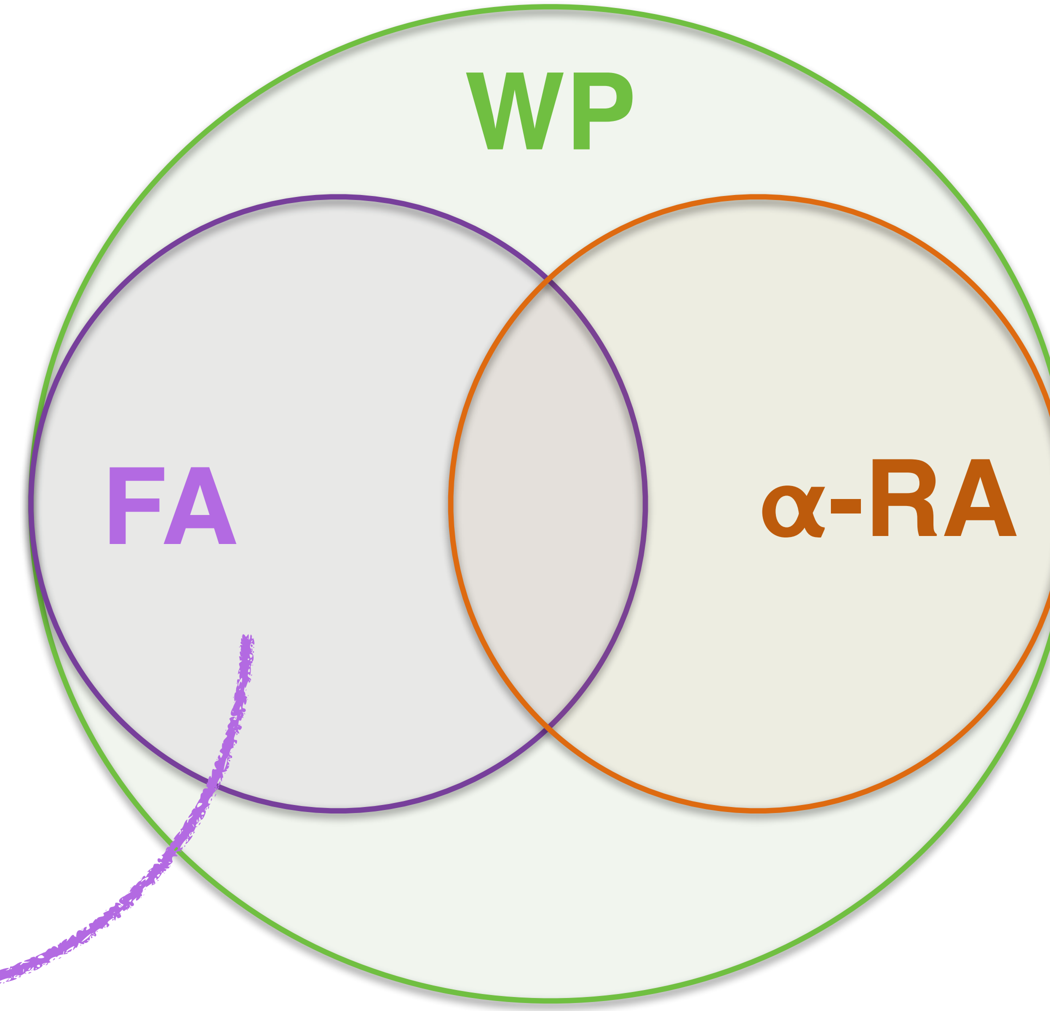
- **e.g.,** (?x, (a/b/c)+, ?y)



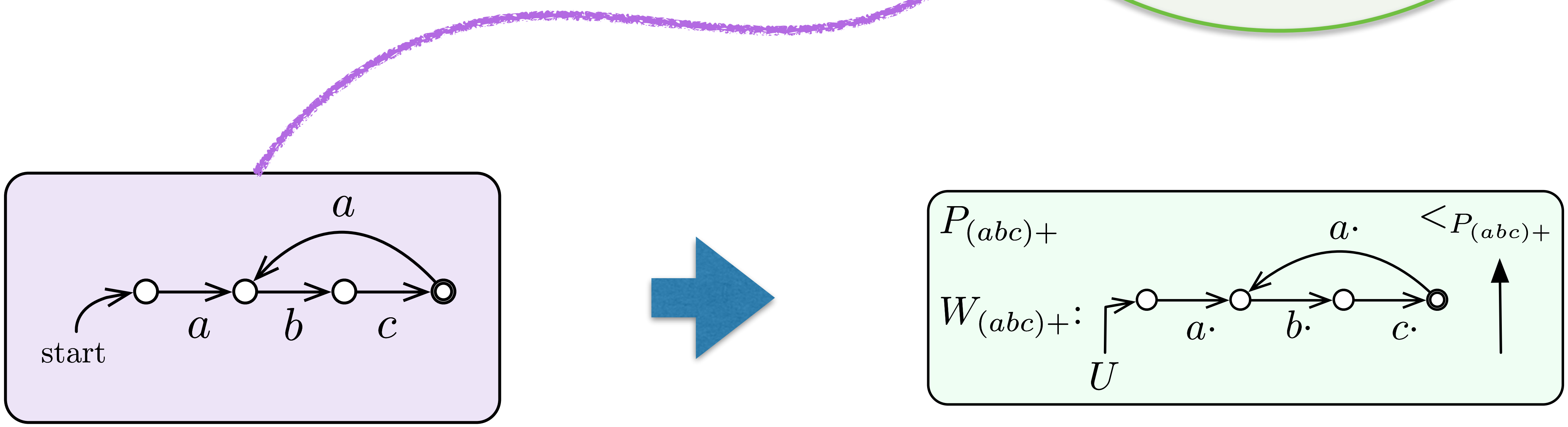
WAVEGUIDE Plan Space

- **subsumes** both FA and α-RA
- adds **exclusive** new plans

α-RA ∪ FA ⊂ WP



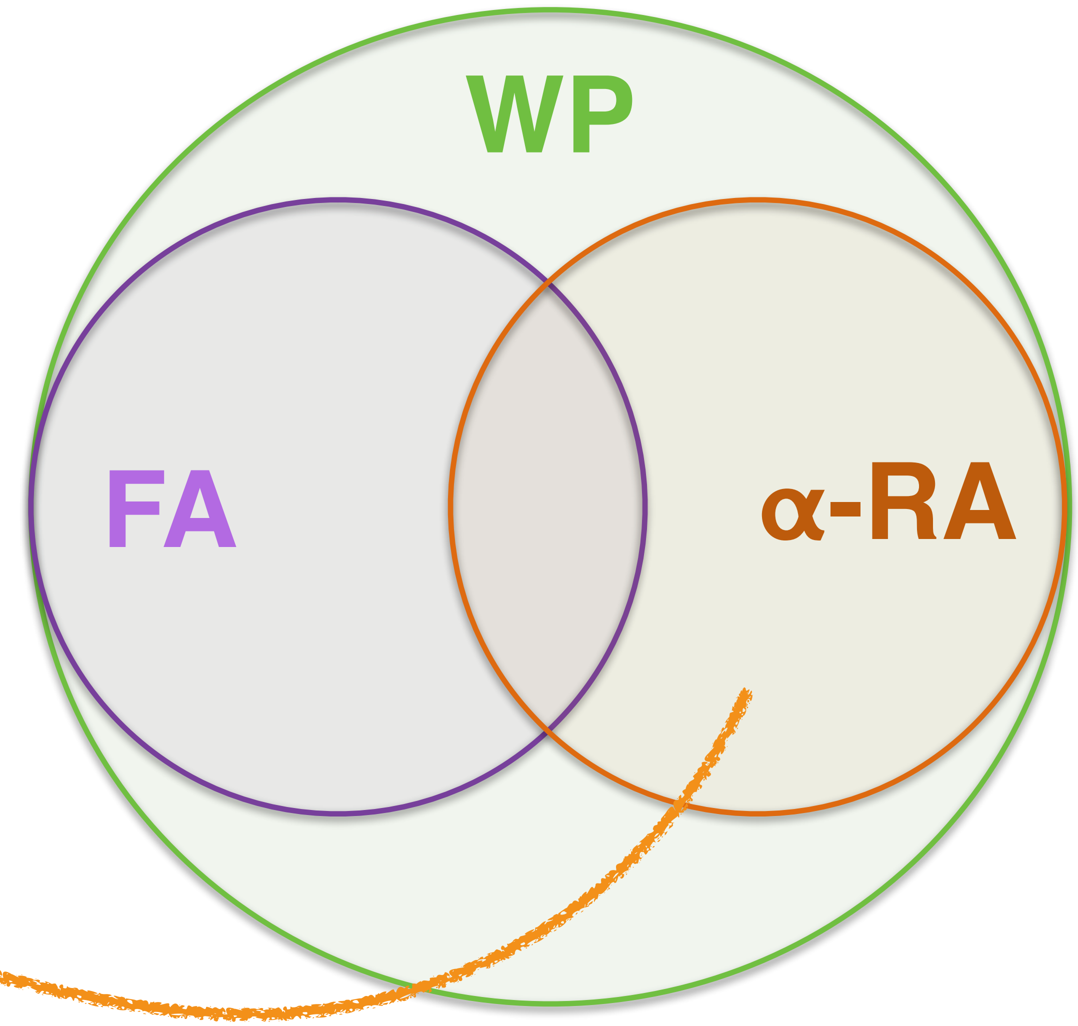
- **e.g.,** (?x, (a/b/c)+, ?y)



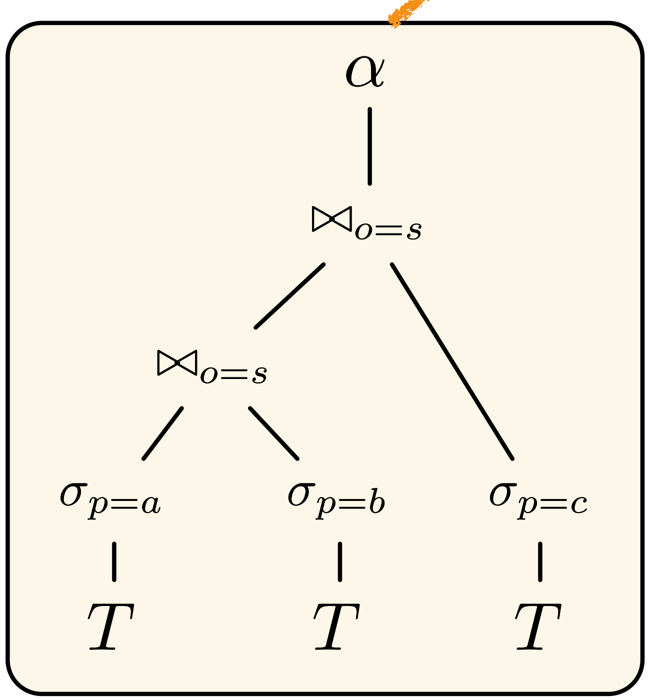
WAVEGUIDE Plan Space

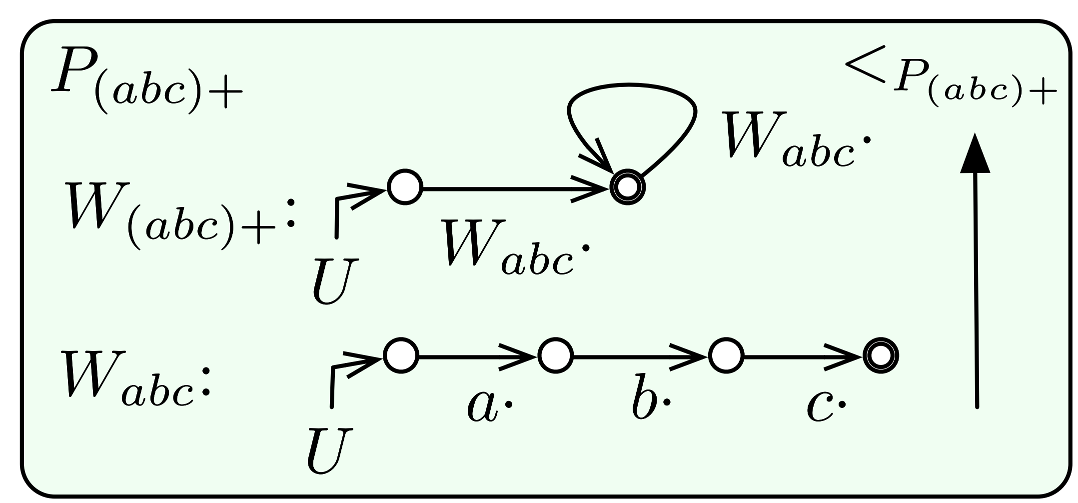
- **subsumes** both FA and α-RA
- adds **exclusive** new plans

α-RA ∪ FA ⊂ WP



- **e.g.,** (?x, (a/b/c)+, ?y)



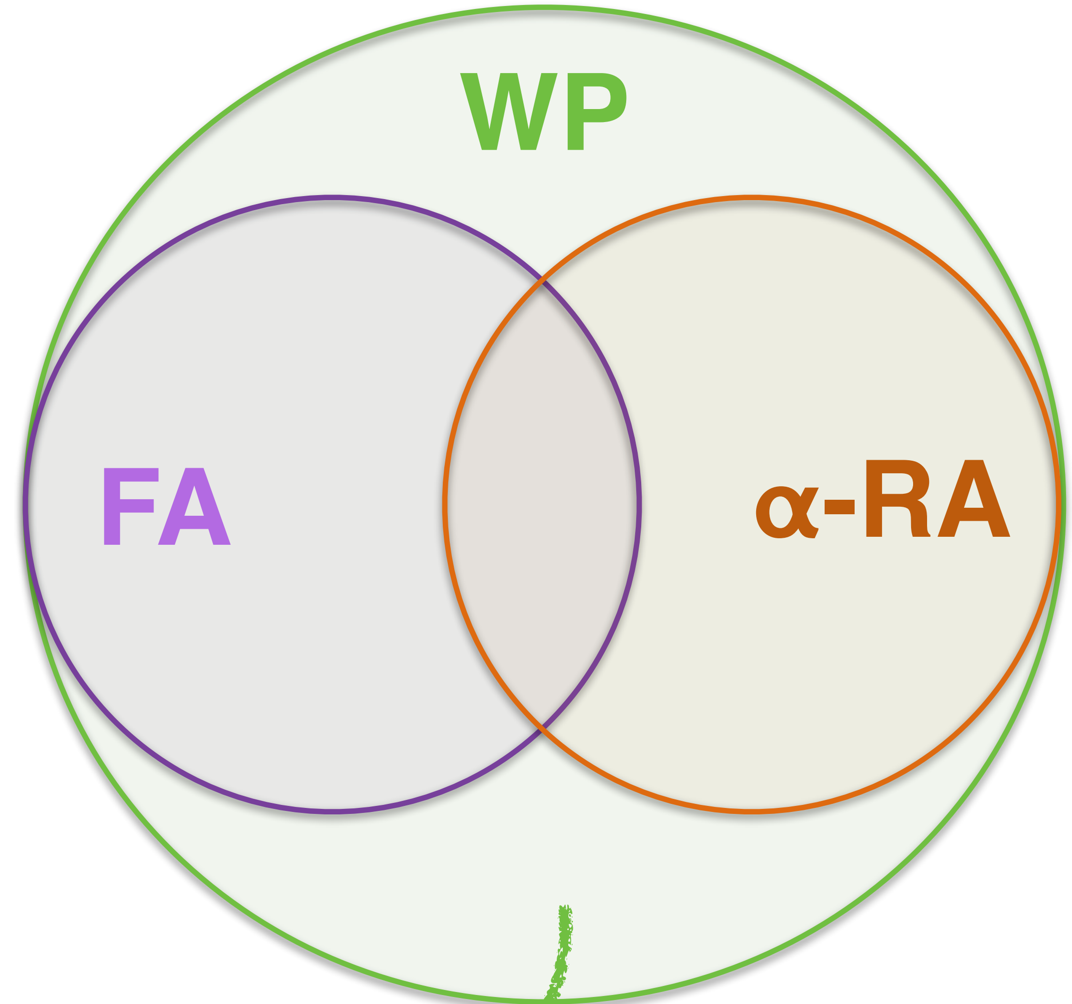


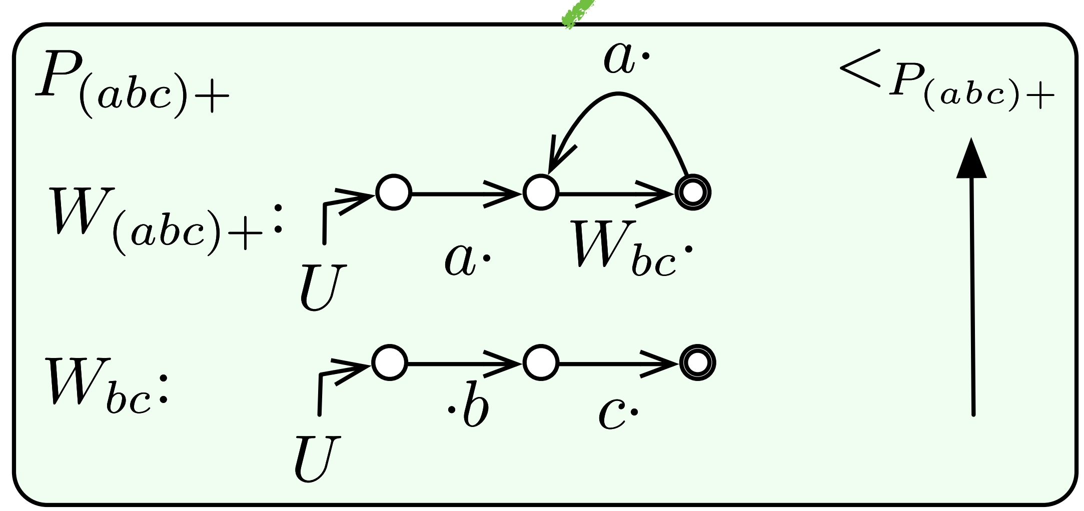
WAVEGUIDE Plan Space

- **subsumes** both FA and α-RA
- adds **exclusive** new plans

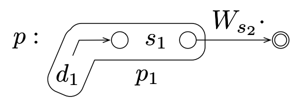
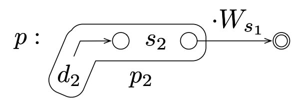
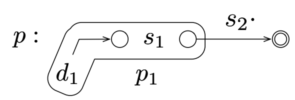
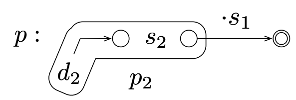
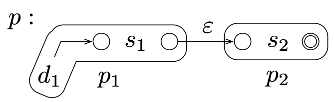
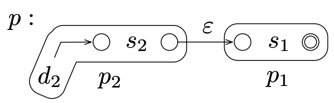
α-RA ∪ FA ⊂ WP

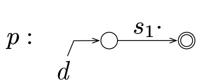
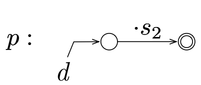
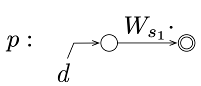
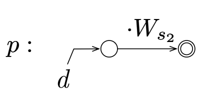
- **e.g., (?x, (a/b/c)+, ?y)**

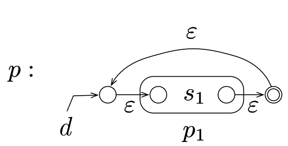
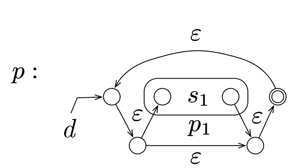




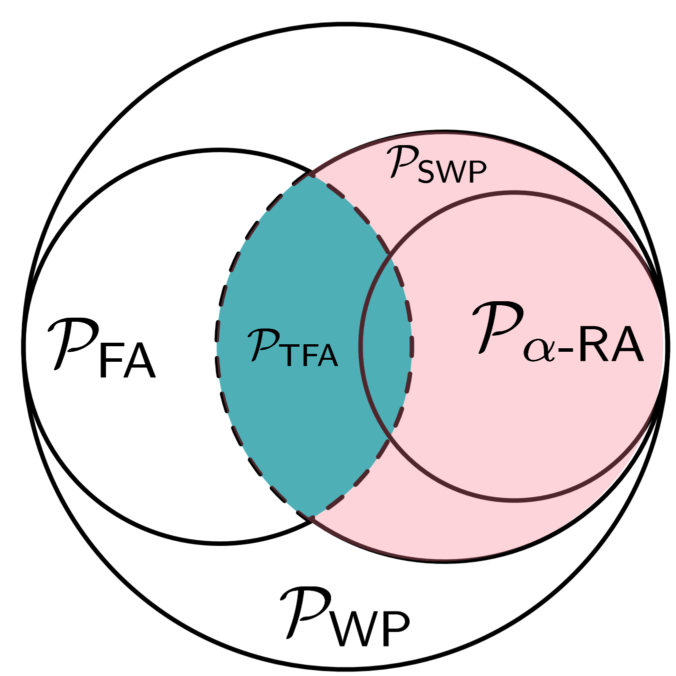
enumerator

rule		waveplan	precondition			
id	*description*		s_1	s_2	*op*	*seed*
CC	concat compound		$\lvert s_1\rvert > 1$	$\lvert s_2\rvert > 1$ $d_2 = U$	/	null
CCF	concat compound flip		$\lvert s_1\rvert > 1$ $d_1 = U$	$\lvert s_2\rvert > 1$	/	null
CP	concat pipe		$\lvert s_1\rvert > 0$	$\lvert s_2\rvert = 1$	/	null
CPF	concat pipe flip		$\lvert s_1\rvert = 1$	$\lvert s_2\rvert > 0$	/	null
DP	direct pipeline		$\lvert s_1\rvert > 0$	$d_2 = s_1$	/	null
DP	inverse pipeline		$d_1 = s_2$	$\lvert s_2\rvert > 0$	/	null

rule		waveplan	precondition			
id	*description*		s_1	s_2	*op*	*seed* (*seed passing*)
ASDP	absorb seed direct pipe		$\lvert s_1\rvert = 1$	null	null	d
ASIP	absorb seed inverse pipe		null	$\lvert s_2\rvert = 1$	null	d
ASDC	absorb seed direct compound		$\lvert s_1\rvert > 1$ $d_1 = U$	null	null	d
ASIC	absorb seed inverse compound		null	$\lvert s_2\rvert > 1$ $d_2 = U$	null	d

rule		waveplan	precondition			
id	*description*		s_1	s_2	*op*	*seed*
KP	kleene plus		$d_1 = d/(s_1)+$ $d_1 = (s_1)+/d$	null	+	null
KS	kleene star		$d_1 = d/(s_1)*$ $d_1 = (s_1)*/d$	null	*	null

- **enumeration** algorithm to walk the sub-space of **standard** plans $\mathcal{P}_{\text{SWP}}$
- bottom-up DP
- **polynomial** in the size of the query
- generates *legal* plans
- guarantees **optimal substructure** wrt. the cost model



High-level Cost Model

WaveguideSearch$(G,\ A_Q)$

$\Delta_0^R \leftarrow$ seed(G);
$i \leftarrow 0$;
while $|\Delta_i^R| \geq 0$ do
  $\Delta_{i+1}^S \leftarrow$ seed(Δ_i^R);
  $\Delta_{i+1}^C \leftarrow$ crank$(\Delta_{i+1}^S,\ \Delta_i^R,\ G,\ C_i,\ A_Q)$;
  $\Delta_{i+1}^R \leftarrow$ reduce$(\Delta_{i+1}^C,\ \Delta_i^R,\ C_i)$;
  $C_{i+1} \leftarrow$ cache$(\Delta_{i+1}^R,\ C_i)$;
  $i \leftarrow i+1$;
done;
return extract(C_i);

$$\mathbf{C}_{\text{crank}} = \sum_{i=0}^{n} f_1(|\Delta_i|)$$

$$\mathbf{C}_{\text{reduce}} = \sum_{i=0}^{n} (f_2(|\Delta_i|) + f_3(|C_i|))$$

$$\mathbf{C}_{\text{cache}} = \sum_{i=0}^{n} f_4(|C_i|)$$

* Costs of **crank-reduce-cache** operations

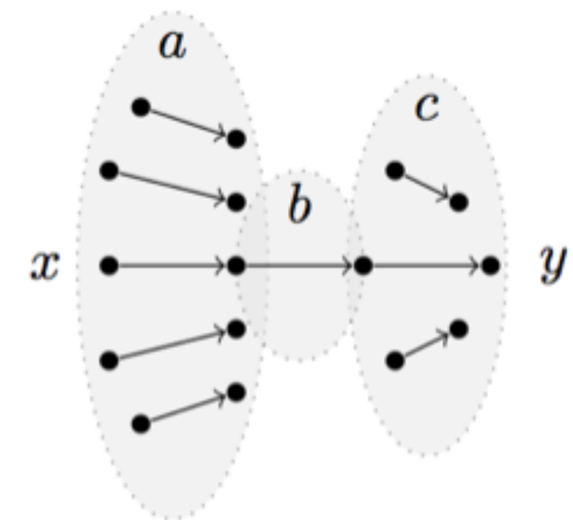
C_{crank} + C_{reduce} + C_{cache}

C_{crank}
- Total number of *edge walks* during the search
- Roughly the sum of sizes of all deltas (**search space**)

C_{reduce}
- Duplicate removal within a delta (**search space**)
- Duplicate removal against the cache (**materialized cache size**)

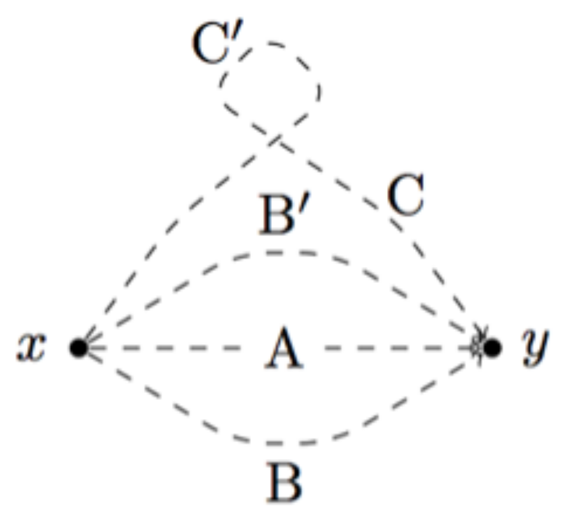
C_{cache}
- Cache maintenance (indexing, etc.)

Cost Factors



Search cardinality

- *Number* of wavefronts, *starting* points, *directions*
- similar to *join ordering* in relational databases
- use *graph statistics* such as **joint label frequencies - synopsis**

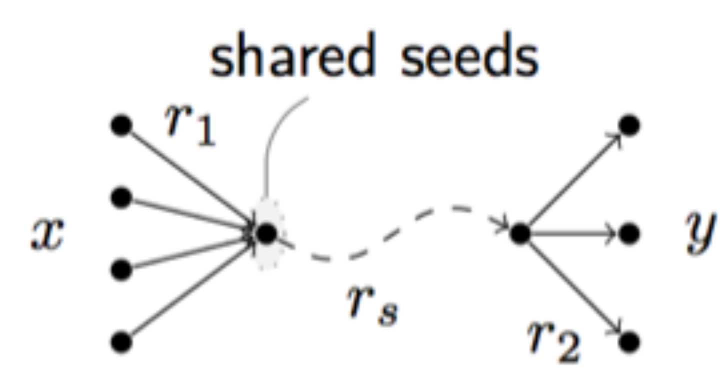


Solution redundancy

- due to existential semantics of RPQ evaluation
- need only one solution per satisfying node pair
- nodes re-discovered by following different conforming paths
- nodes rediscovered by following cycles
- **different redundancy for different plans!**

Sub-path redundancy

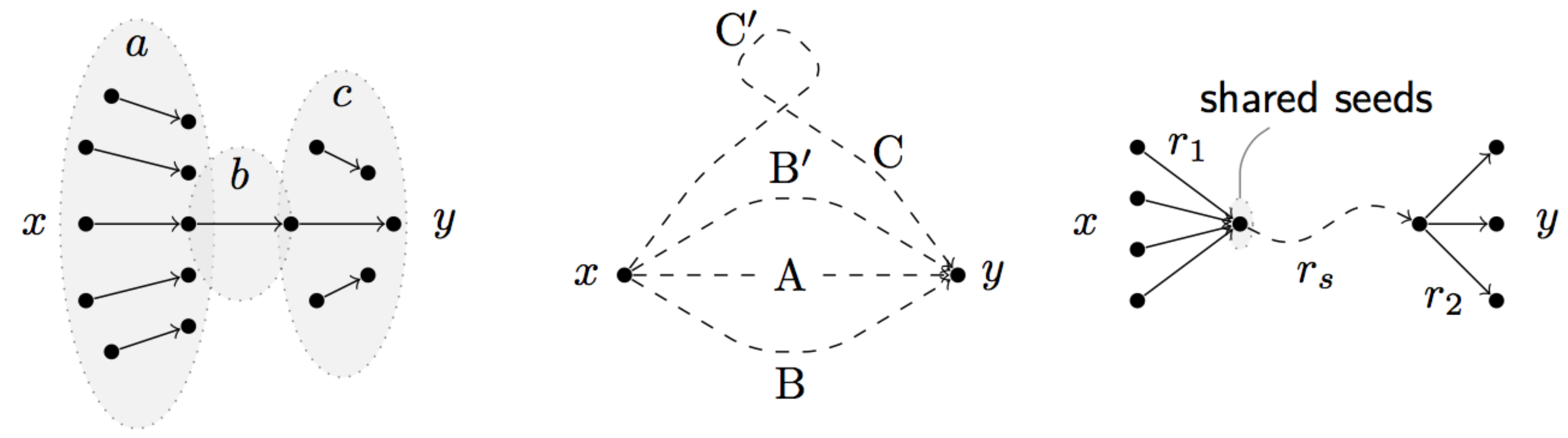
- common in dense graphs with *hierarchical* structures
- answer pairs may *share* significant sub-paths
- efficient to evaluate *separately*



WAVEGUIDE Optimization Methods

Choice of wavefronts

- starting points, directions with direct/inverse and graph/view transitions



a) search cardinality b) solution redundancy c) sub-path sharing

Threading

- seeded sub-automata
- use results via named sets (views)

Reduce

- counter duplicates both *re-discovered* and *cyclic*
- first-path pruning (**FPP**)

Partial materialization

- often materialization not necessary
- identify *pipelining* cases

Loop caching

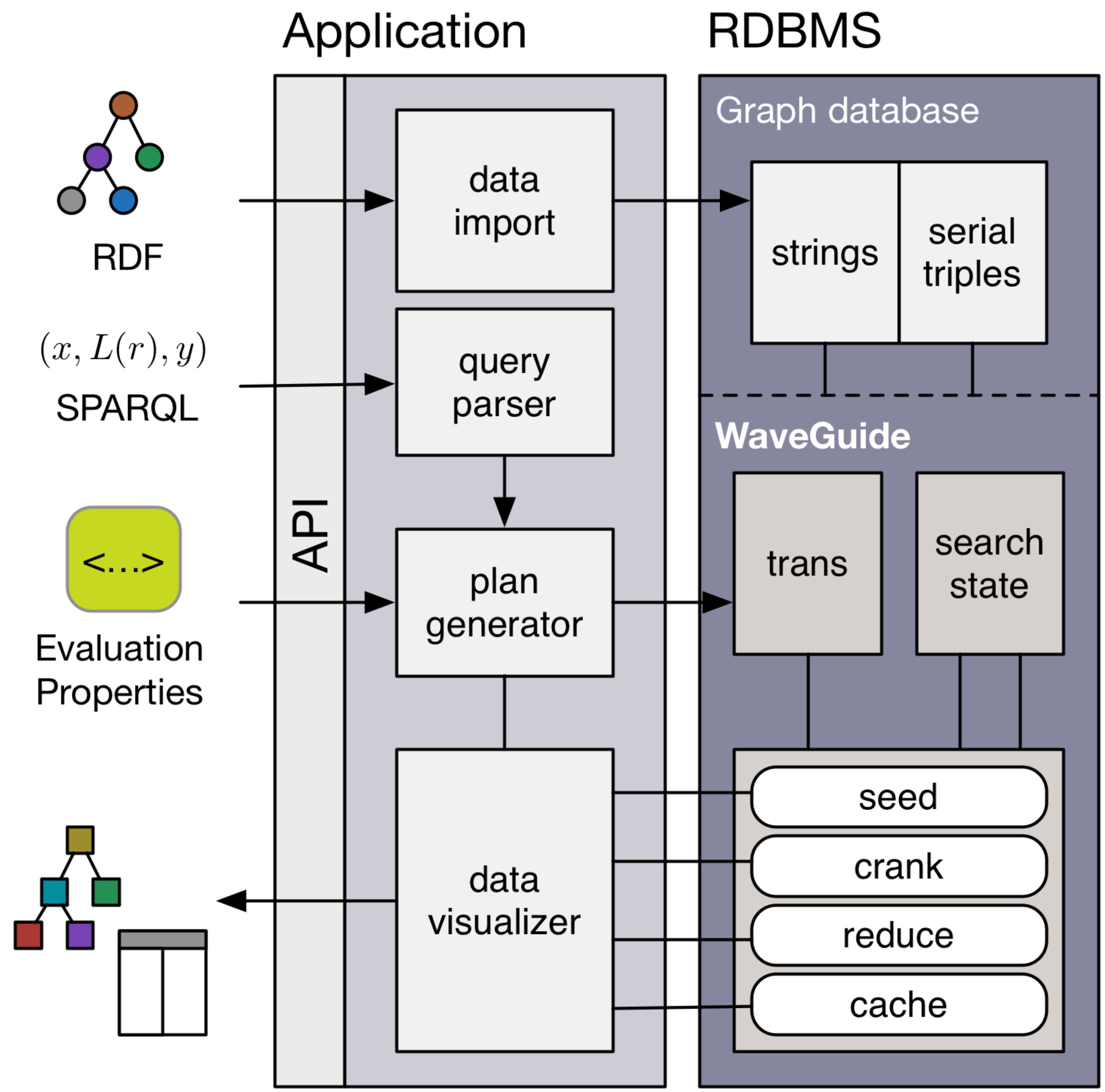
- pre-computing parts of the automata within a loop

Implementation

* **Waveguide in the context of SPARQL**
  - case study of SPARQL property path query optimization on large RDF datasets

* **Guided search as procedural SQL**
  - implemented in PostgreSQL

* **Illustration**
  - query plan designer
  - runtime visualizer
  - profiler



Performance

* **Various domains**
  - *social* (LDBC social network intelligence benchmark)
  - *life sciences* (UNIPROT)
  - *encyclopedic* (Yago2s, DBPedia)

* **Queries**
  - mining for specified RPQ pattern templates
  - a set of realistic queries

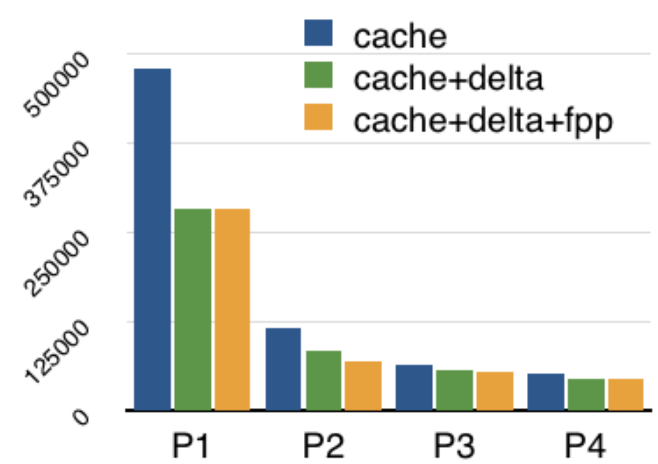
Plan Performance

- Example query on Yago2s dataset:

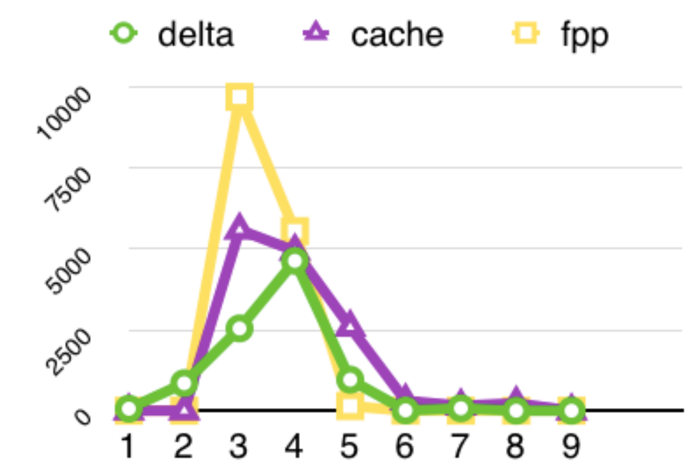
$$Q = \text{?p :marriedTo/:diedIn/:locatedIn+/:dealsWith+ USA}$$

- Sample waveplans:

P_1: single wavefront USA → ?p.
P_2: single wavefront ?p → USA.
P_3: two wavefronts
?p → :locatedIn+/:dealsWith ← USA.
P_4: P_2 but with a threaded sub-path
:locatedIn+/:dealsWith+ USA.

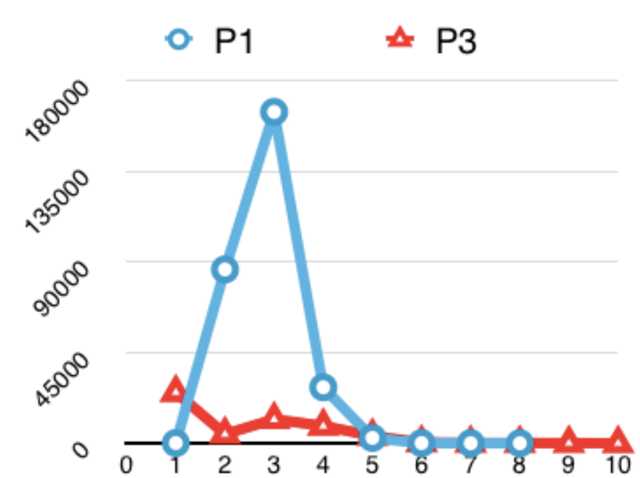


a) Search size for different plans and pruning types

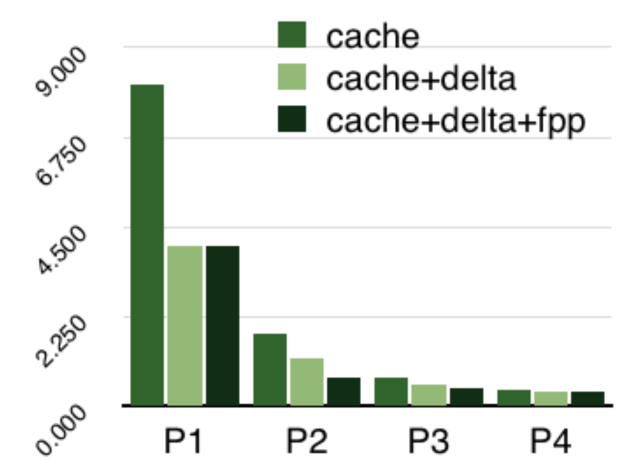


b) Redundancy pruning (by type) over iterations of P2

- **Observations**
  - can achieve **orders of magnitude improvement** even for simple queries
  - different **redundancy pruning profiles** depending on tape
  - want to **constrain** delta sizes over iterations

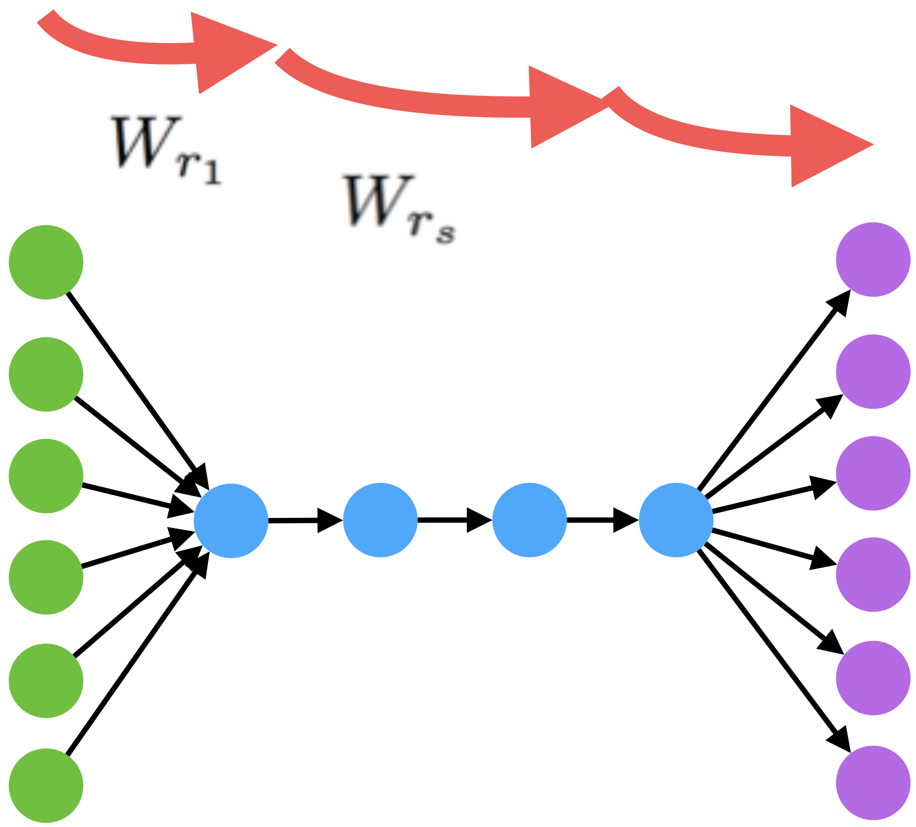


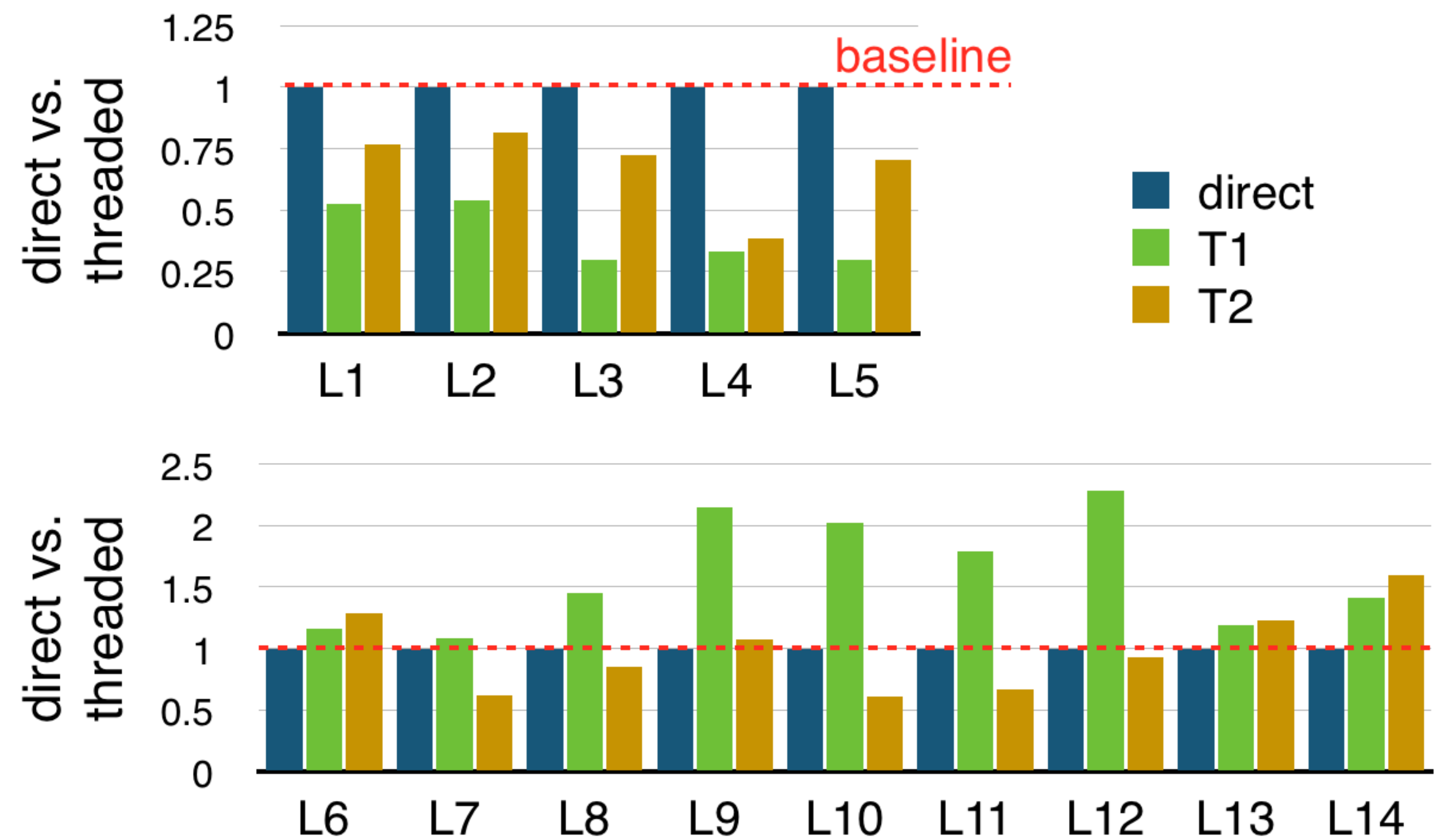
c) Delta sizes over iterations



d) Total query time

Threading Performance

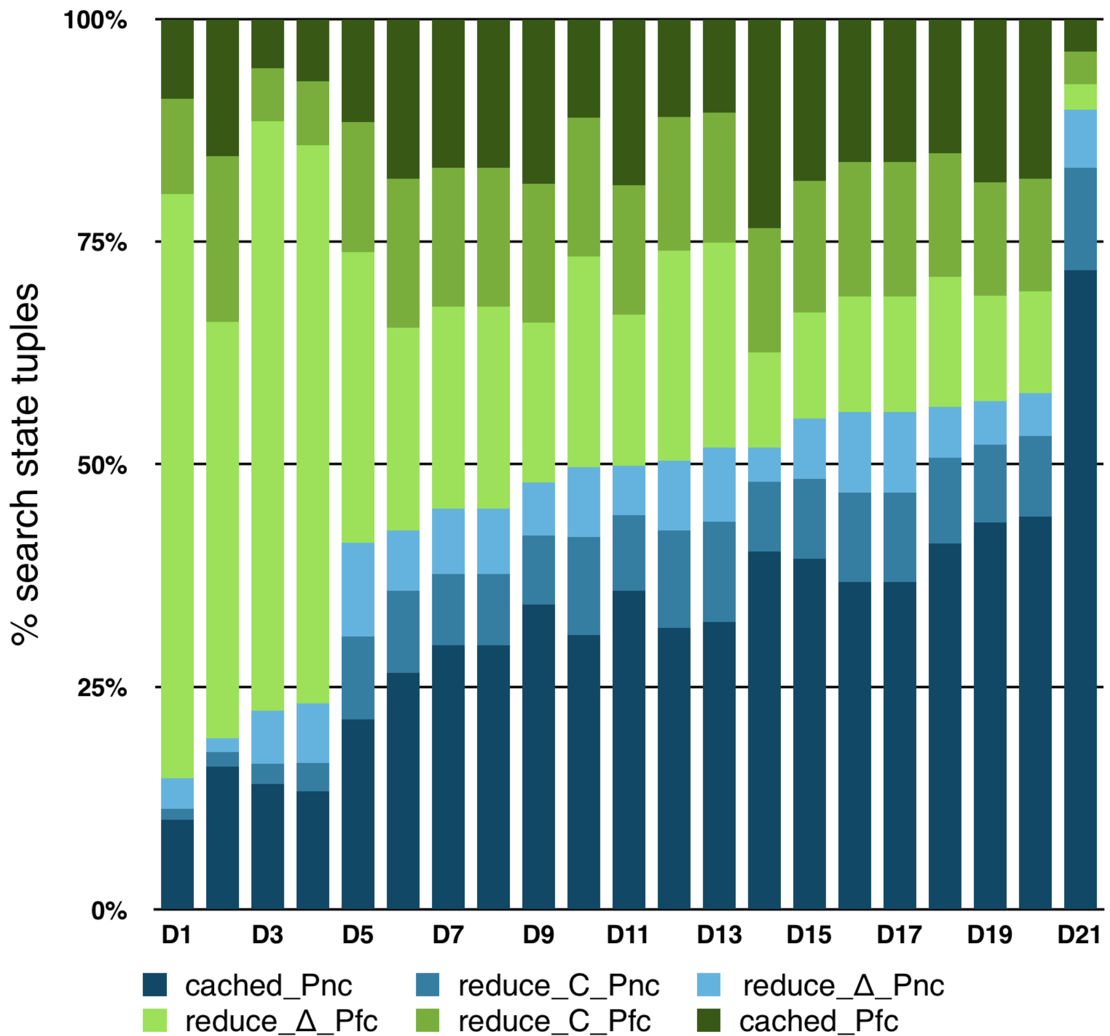


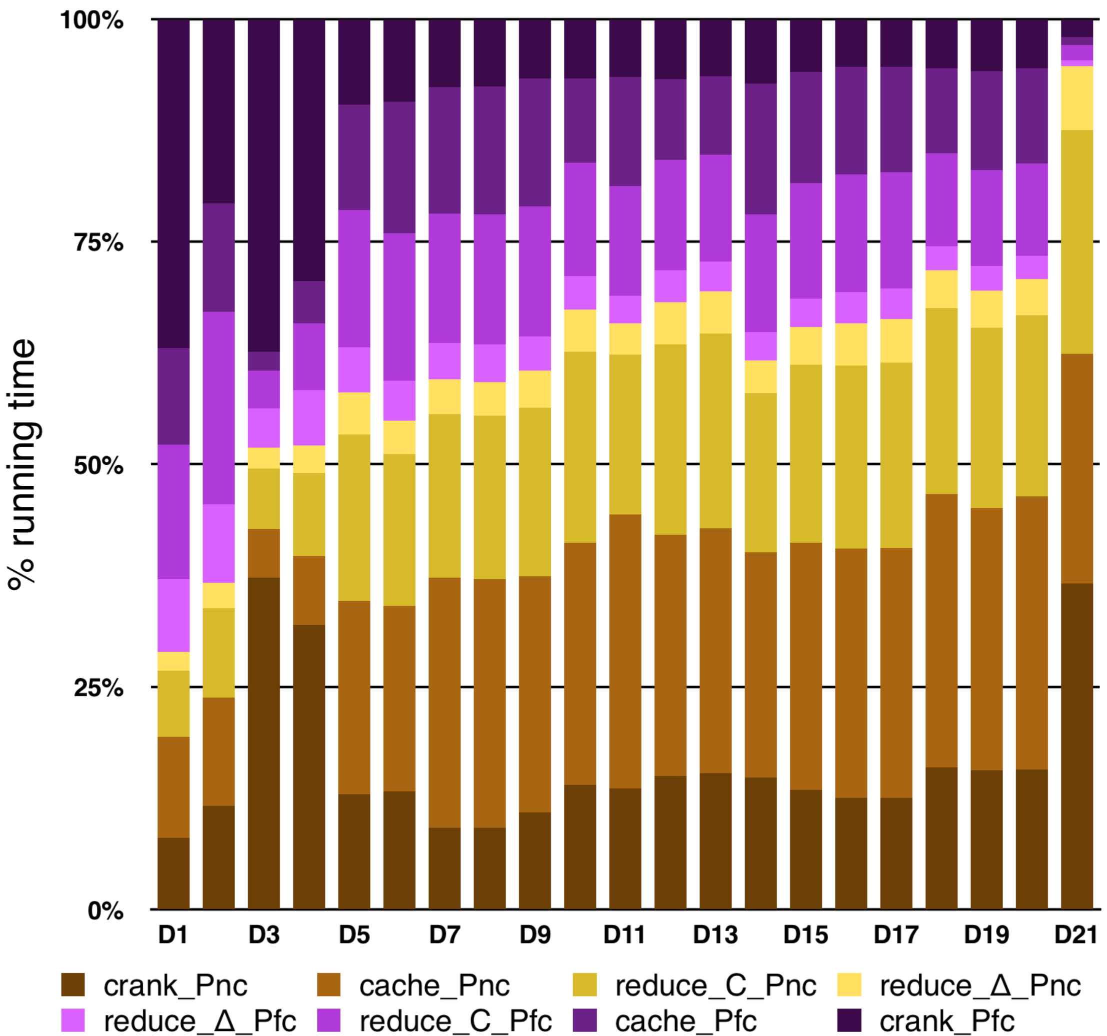


- DBPedia dataset
- Different threading points and different labels
- **Where to thread**?
  - hierarchy vs. length of potential shared path
- Can be **harmful** if threading chosen poorly
- Need to **cost**

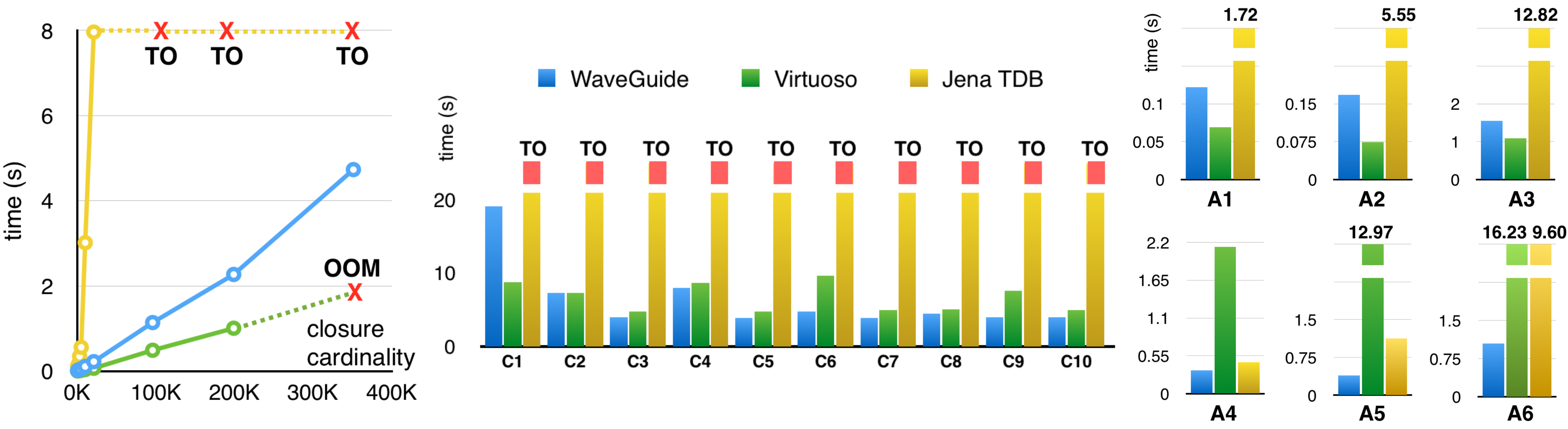
Loop-caching Performance

* DBPedia dataset: mining 21 queries of type **?x (a/b) ?y**
* evaluating pipelined and full loop caching: **is rich WG plan space useful?**
* need to cost, as the type of edge walks performed is different depending on a **plan** *and* **shape of the graph**





vs. others



* **mining** RPQ patterns and set of **realistic** queries over YAGO2s and DBPedia
* benchmarking:
  * transitive closure
  * query planning
* despite slower transitive closure, WG gains significant improvement due to richer plan space

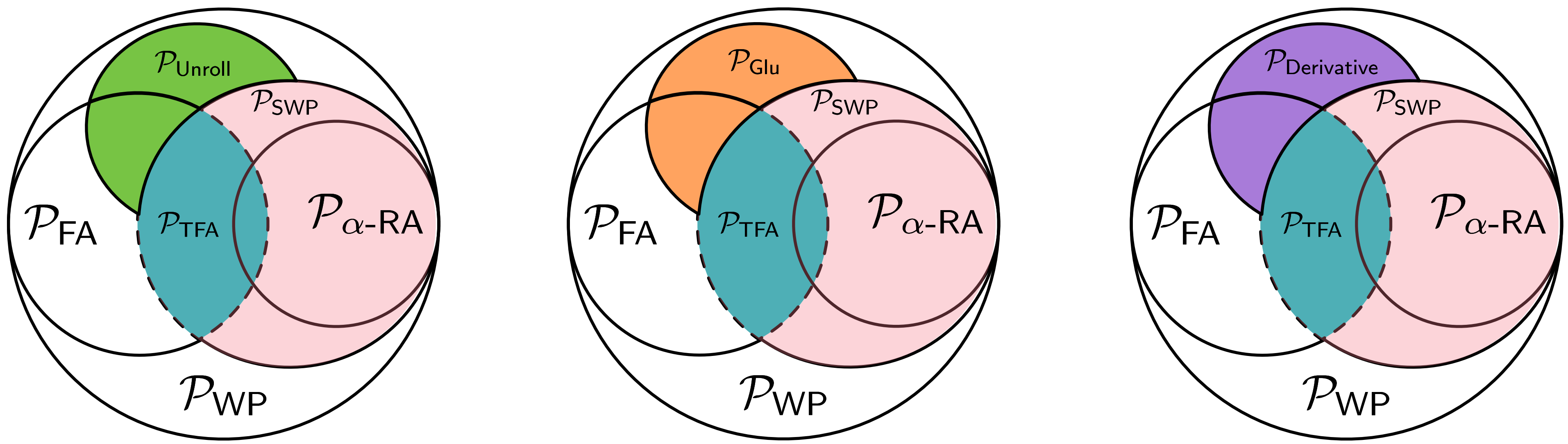
WAVEGUIDE

* **Devise** *WAVEGUIDE* (WG) framework for planning and evaluation of RPQs (SPARQL property paths)
* **Demonstrate** that it subsumes existing techniques and extends well beyond them
* **Analyze** WG's plan space and provide an efficient way to enumerate through subspace of plans
* **Model** the cost factors that determine the efficiency of the plans
* **Present** and **prototype** powerful optimizations offered by WG plans

BEYOND WAVEGUIDE

- **Multiple and Conjunctive RPQs**
  - ***extend*** from single-path property-path queries (RPQs)
  - how to utilize ***common subexpressions*** to find global optimal plans?
- **Richer Enumerator**
  - go beyond ***Thompson-like*** construction of waveplans
  - explore ***k-unrolling*** for Kleene expressions
  - other ***automata minimization/construction*** techniques
- **Better Cardinality Estimation**
  - overcome ***uniformity assumption*** with ***extended synopsis with binning***
  - estimate ***correlations across joins*** to overcome independence assumption

richer plan space



- have efficient enumeration for a subspace of standard waveplans $\mathcal{P}_{\text{SWP}}$

can we do better?

- analyze if using:
  - ***k-unrolling*** - to (partially) unroll Kleene expressions
  - ***Glushkov*** automata
  - ***Derivative*** automata

Thank You!