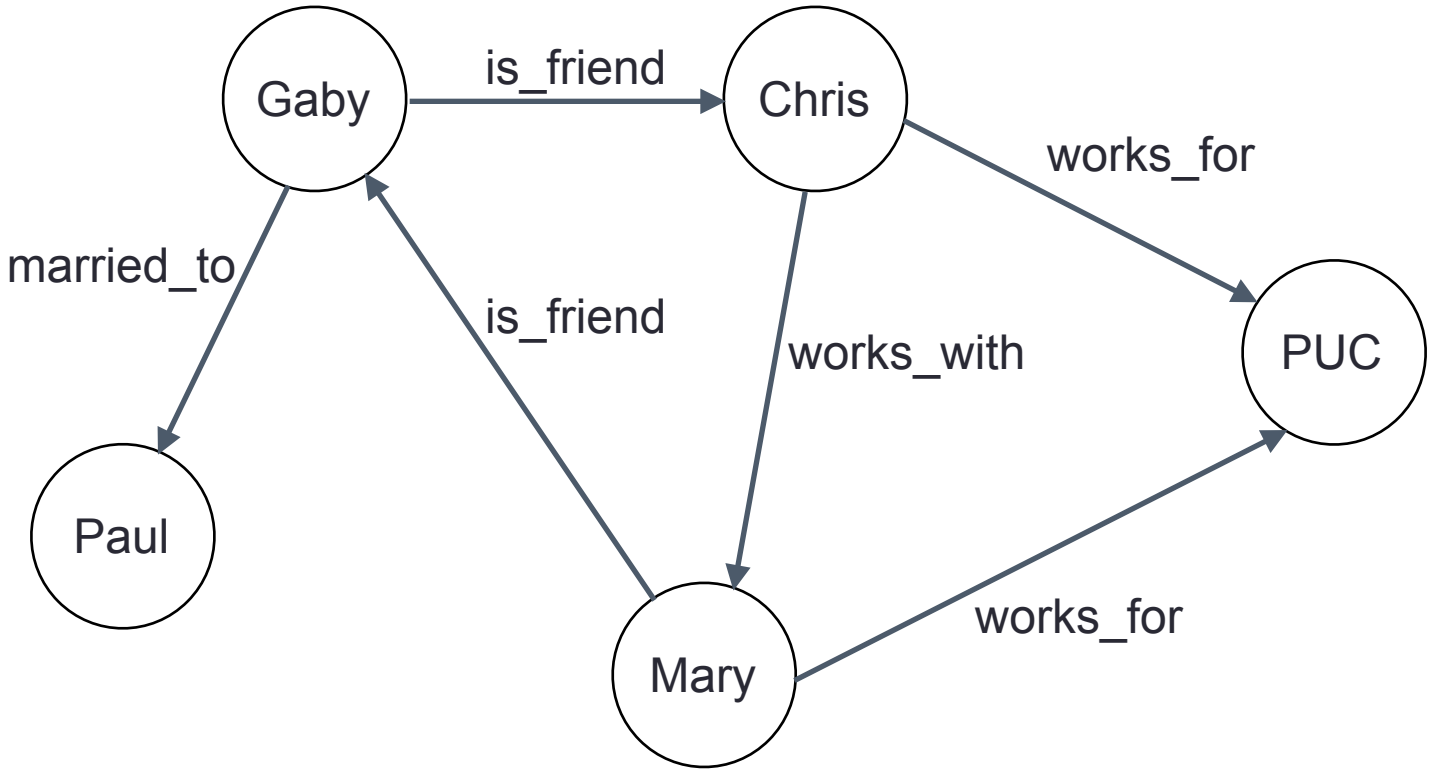
Navigational and Rule-Based Languages for Graph Databases

Juan L. Reutter

PUC Chile

Center for Semantic Web Research

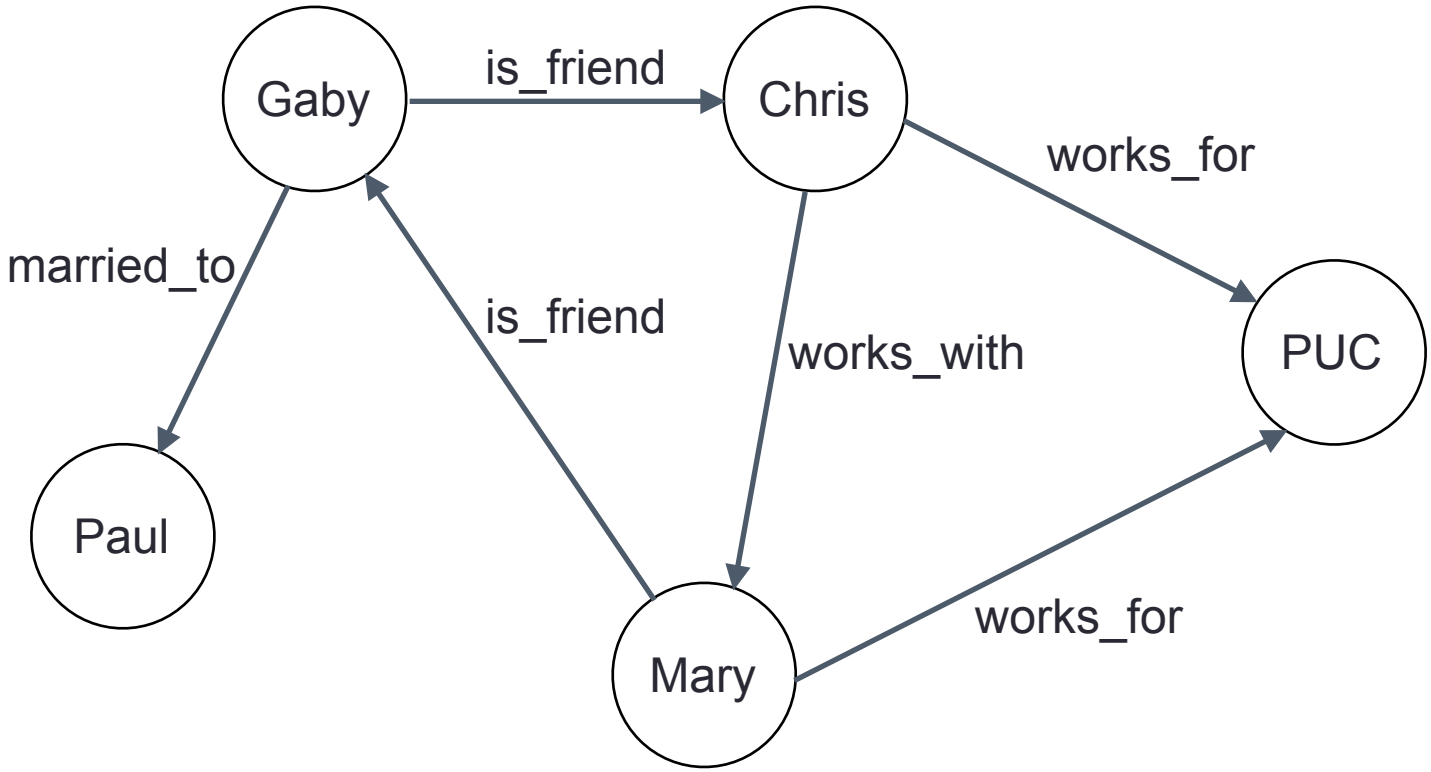
Graph Databases



Graph Databases

- Social Networks
- Biological DBs
- Geographic Models

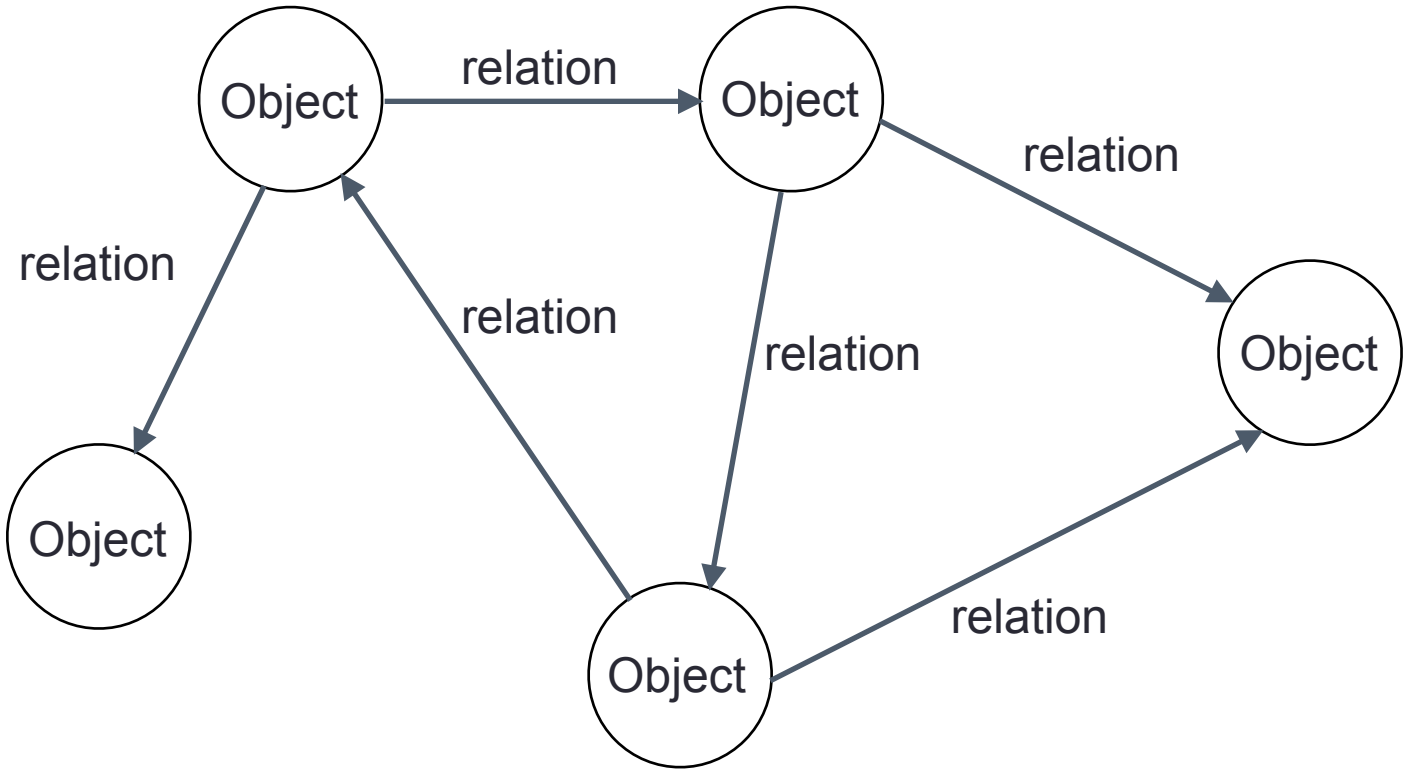
- RDF



Graph Databases

- Nodes represent objects

- Edges represent relations



Graph Databases

- Nodes represent objects

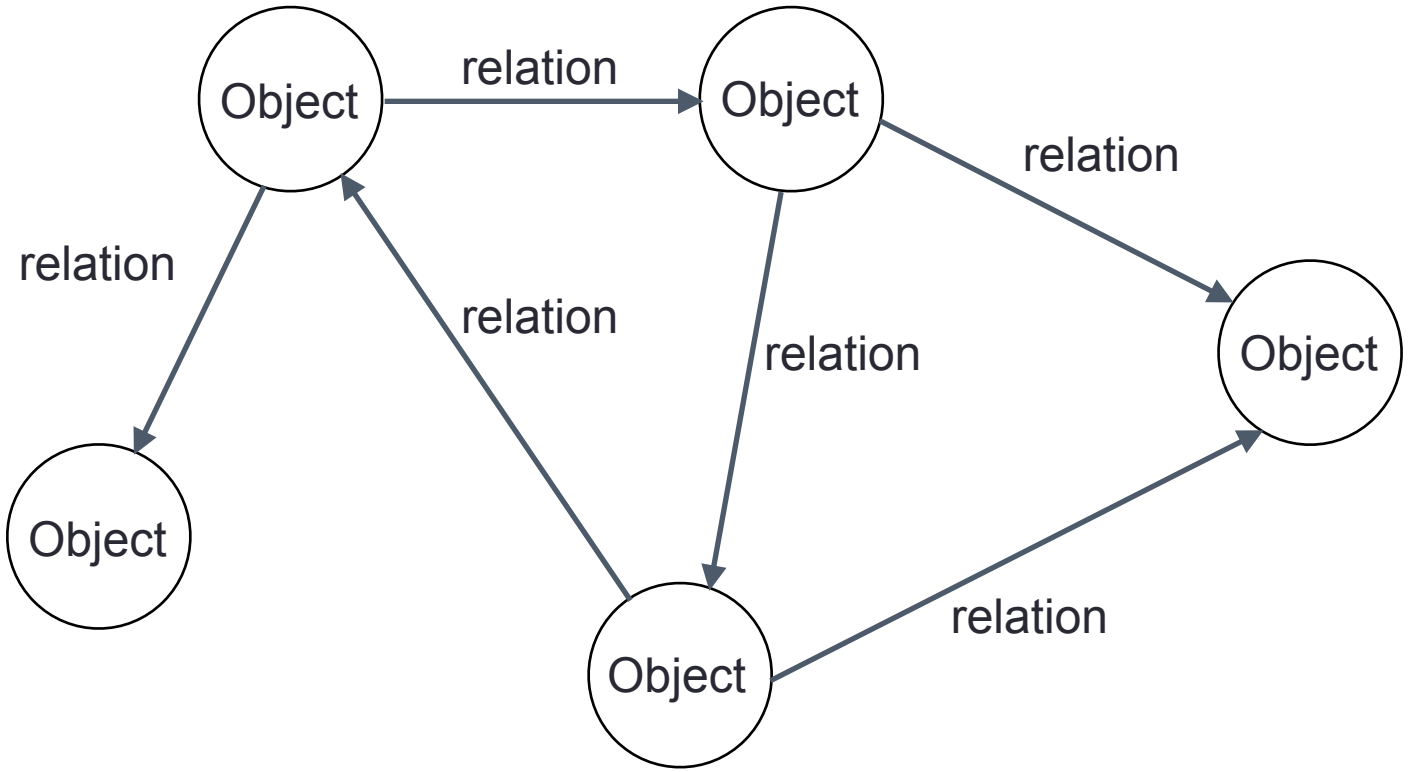
- Edges represent relations

$G = (V, E)$

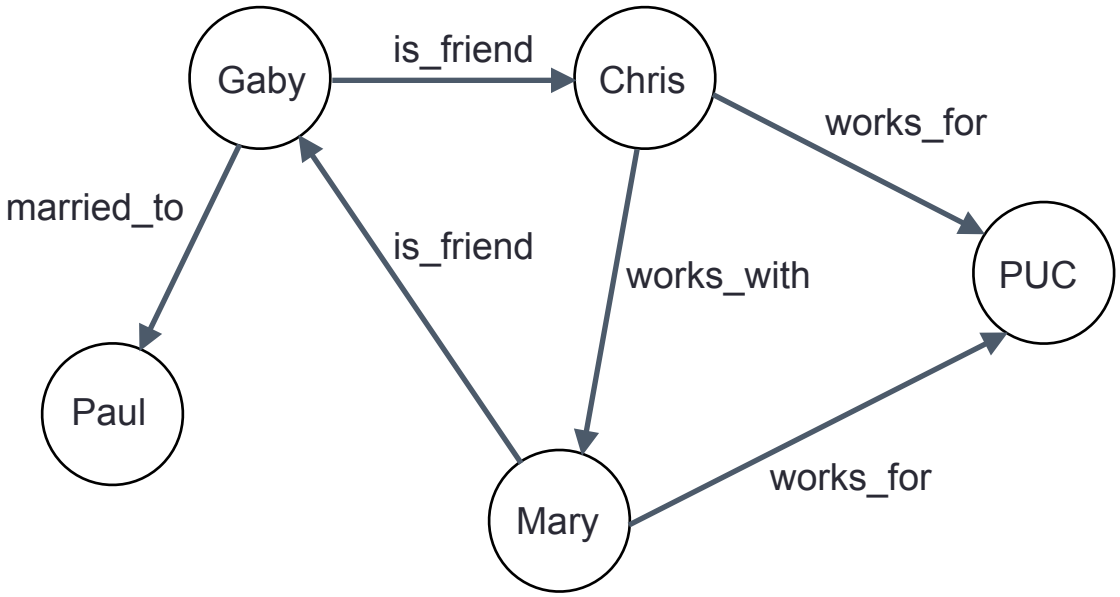
V is set of nodes

$E \subseteq V \times \Sigma \times V$ is set of edges

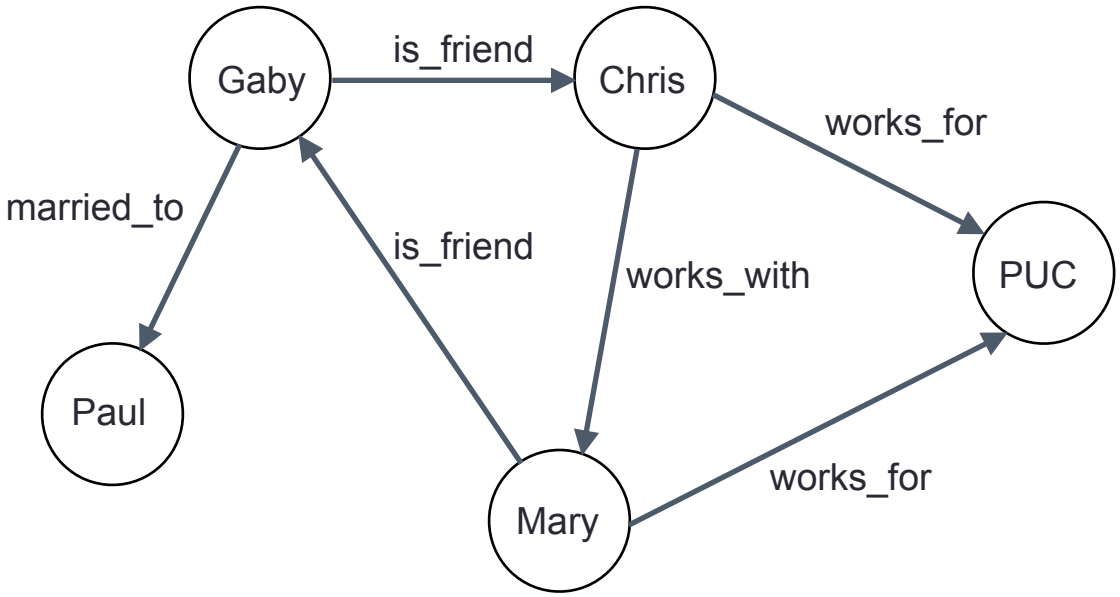
Σ set of labels (finite)



Graph as Relational Databases



Graph as Relational Databases



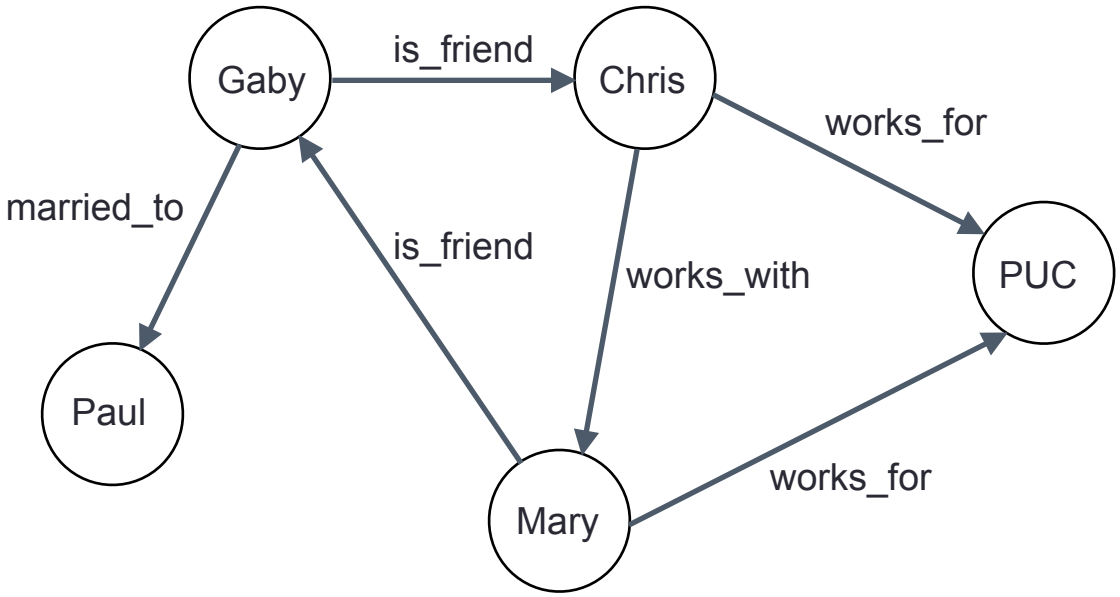
works_for	
Chris	PUC
Mary	PUC

works_with	
Chris	Mary

married_to	
Gaby	Paul

is_friend	
Gaby	Chris
Mary	Gaby

Graph as Relational Databases



works_for	
Chris	PUC
Mary	PUC

works_with	
Chris	Mary

married_to	
Gaby	Paul

is_friend	
Gaby	Chris
Mary	Gaby

So really graphs are a special type of relational DB

Graph queries are fundamentally different

(this is one of the reasons we study them)

Graph queries are fundamentally different

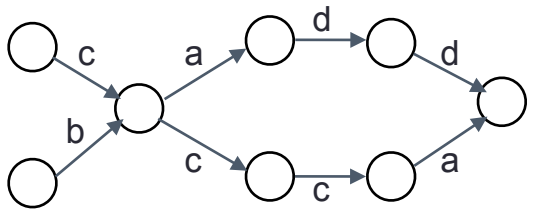
(this is one of the reasons we study them)

Patterns

Navigation

Graph queries are fundamentally different: Patterns

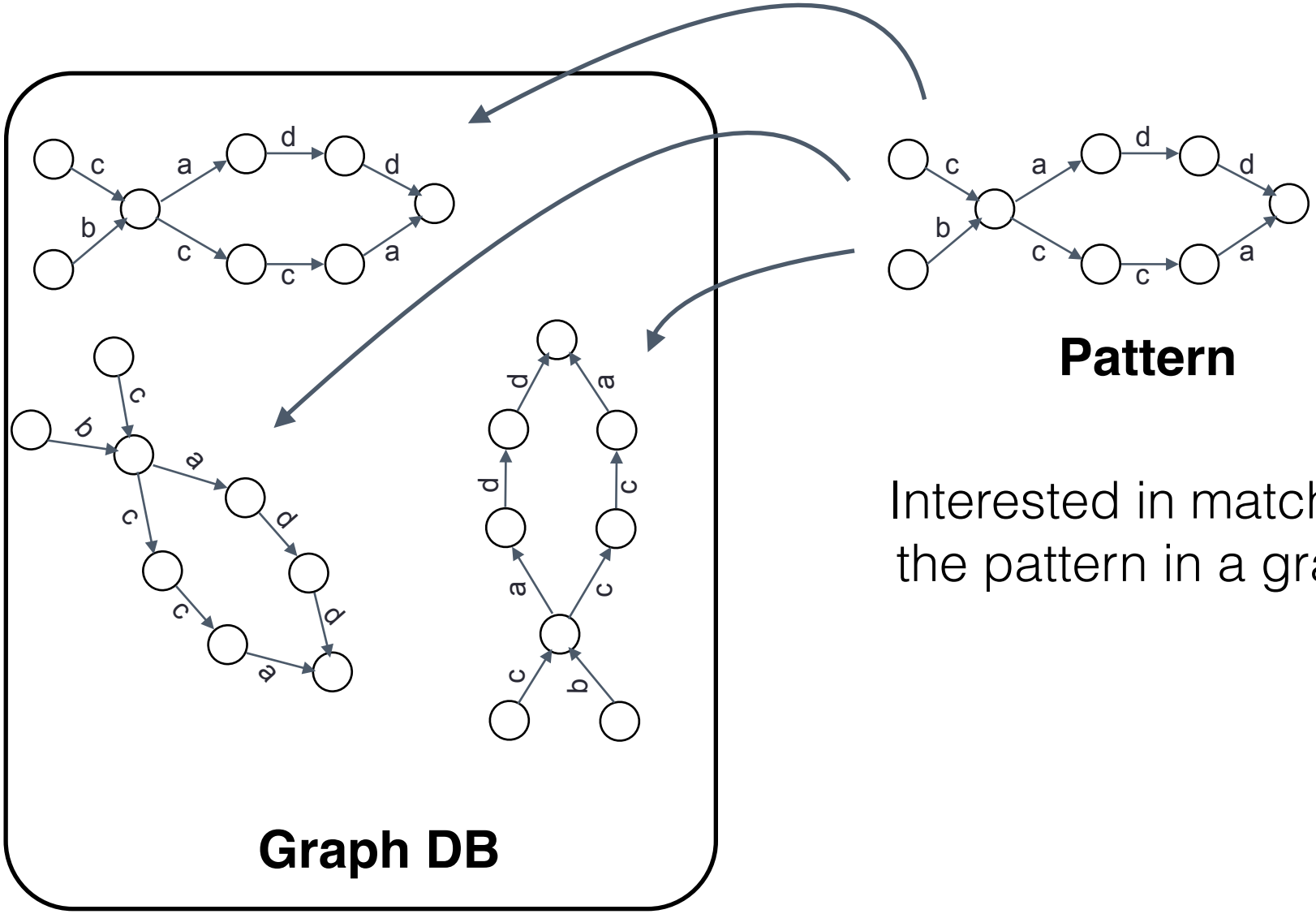
Graph DB



Pattern

Interested in matching the pattern in a graph

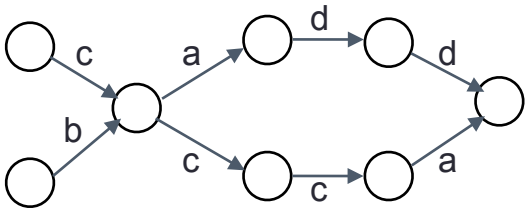
Graph queries are fundamentally different: Patterns



Pattern

Interested in matching the pattern in a graph

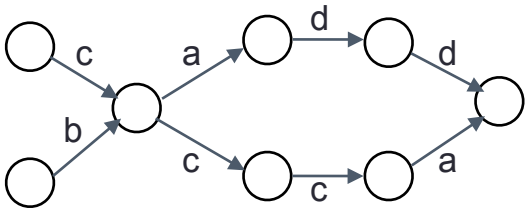
Graph queries are fundamentally different: Patterns



Pattern

Interested in matching the pattern in a graph

Graph queries are fundamentally different: Patterns

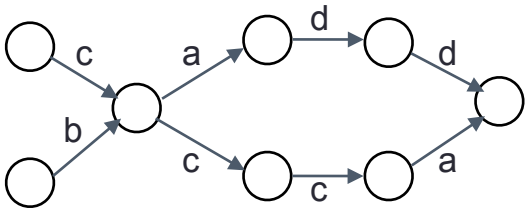


Pattern

Nightmare query for relational DBs!

Interested in matching the pattern in a graph

Graph queries are fundamentally different: Patterns



Pattern

Nightmare query for relational DBs!

$$c \bowtie a \bowtie d \bowtie d \bowtie b \bowtie c \bowtie c \bowtie a$$

Interested in matching the pattern in a graph

$$\exists z_1 \exists z_2 \exists z_3 \exists z_4 \exists z_5 \exists z_6 \exists z_7 \; c(x, z_1) \wedge a(z_1, z_2) \wedge d(z_2, z_3) \wedge$$
$$d(z_3, z_4) \wedge b(y, z_5) \wedge c(z_5, z_6) \wedge c(z_6, z_7) \wedge a(z_7, z_4)$$

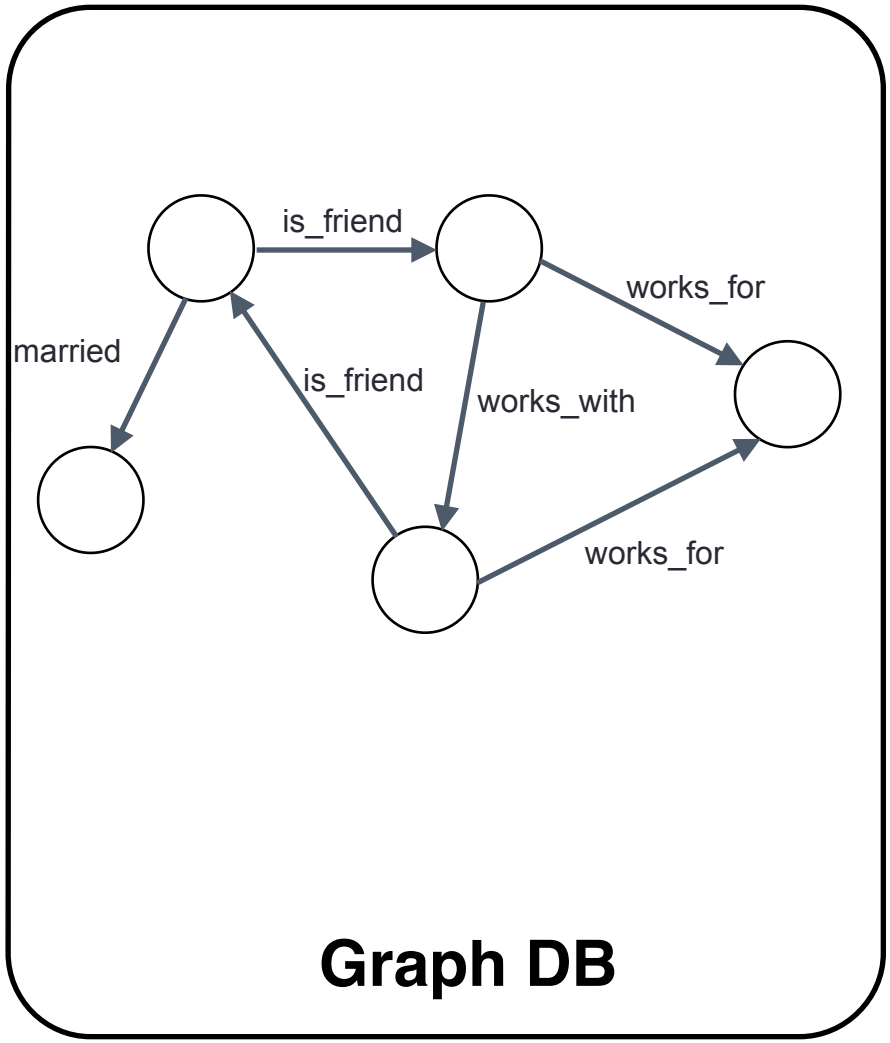
Graph queries are fundamentally different

(this is one of the reasons we study them)

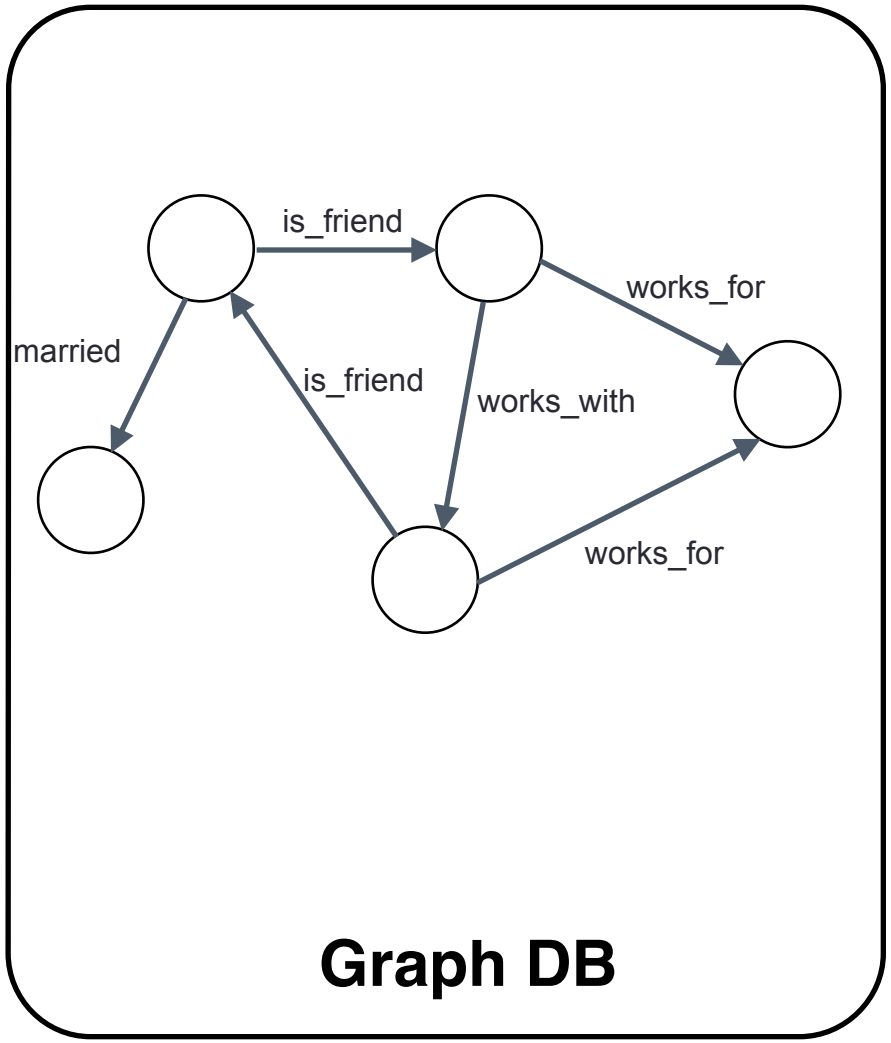
Patterns: Graphs must be optimised for pattern matching

Navigation

Graph queries are fundamentally different: Navigation

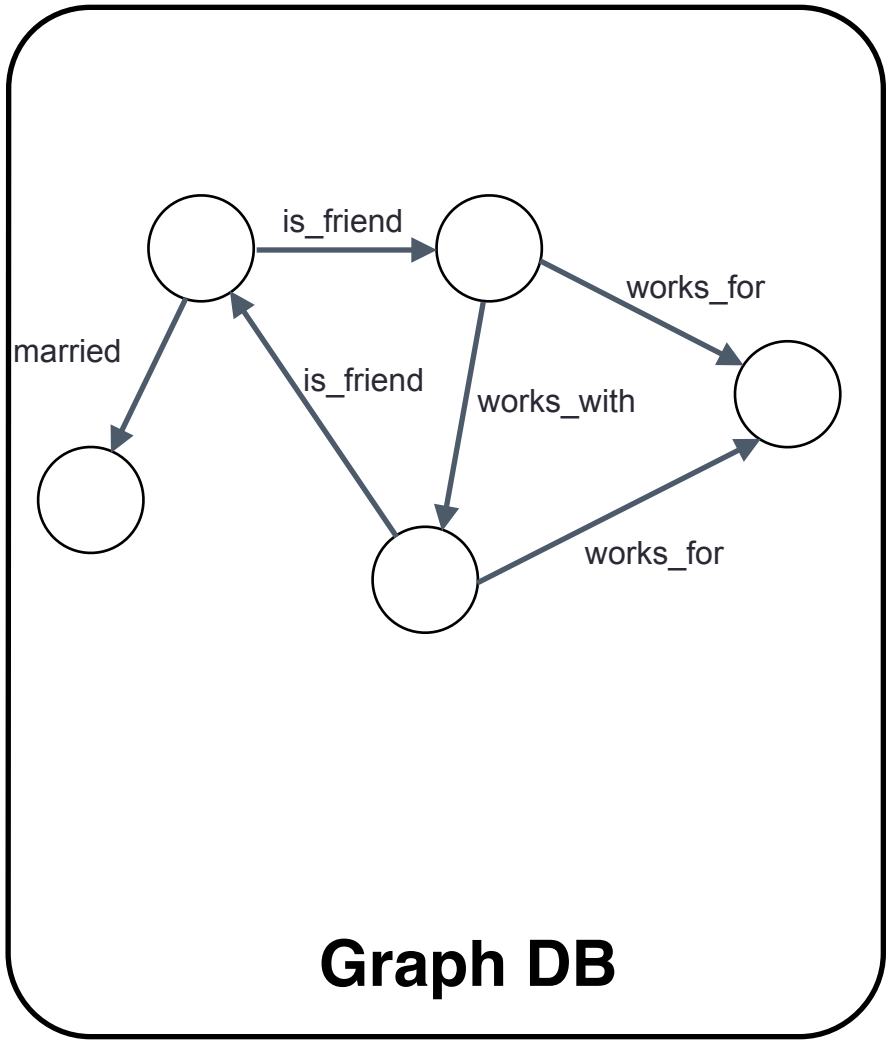


Graph queries are fundamentally different: Navigation



Do I have a friend
that works at Google?

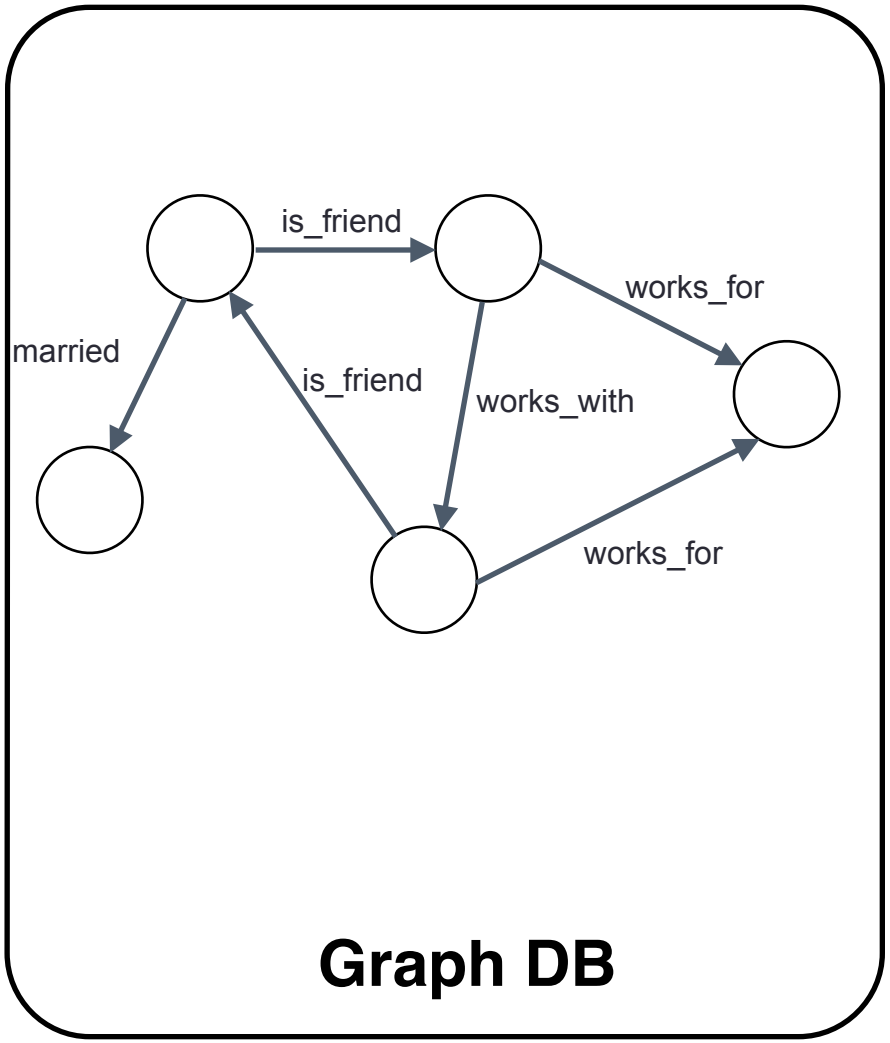
Graph queries are fundamentally different: Navigation



Do I have a friend that works at Google?

Can I reach a Googler via a chain of friends?

Graph queries are fundamentally different: Navigation



Do I have a friend
that works at Google?

Can I reach a Googler
via a chain of friends?

Can I reach a Googler
via a chain of
friends and / or coworkers?

Graph queries are fundamentally different: Patterns

Do I have a friend
that works at Google?

Can I reach a Googler
via a chain of friends?

Can I reach a Googler
via a chain of
friends and / or coworkers?

Graph queries are fundamentally different: Patterns

Nightmare query for relational DBs!

Do I have a friend that works at Google?

Can I reach a Googler via a chain of friends?

Can I reach a Googler via a chain of friends and / or coworkers?

Graph queries are fundamentally different: Patterns

Nightmare query for relational DBs!

Do I have a friend that works at Google?

Can I reach a Googler via a chain of friends?

Can I reach a Googler via a chain of friends and / or coworkers?

It implies using recursion in SQL, not always supported, poorly optimised

Graph queries are fundamentally different

(this is one of the reasons we study them)

Patterns: Graphs must be optimised for pattern matching

Navigation: Graphs must be optimised for navigation

Graph queries are fundamentally different

(this is one of the reasons we study them)

Patterns: Graphs must be optimised for pattern matching

Navigation: Graphs must be optimised for navigation

This talk:
Query languages mixing patterns + navigation

Navigational and Rule-Based Languages for Graph Databases

Juan L. Reutter

PUC Chile

Center for Semantic Web Research

Outline

Navigation and Patterns

Rule-based languages

Moving to RDF

Outline

Navigation and Patterns

Path Queries (definition, evaluation)

Conjunctions of Path Queries (definition, evaluation)

Beyond Patterns

Outline

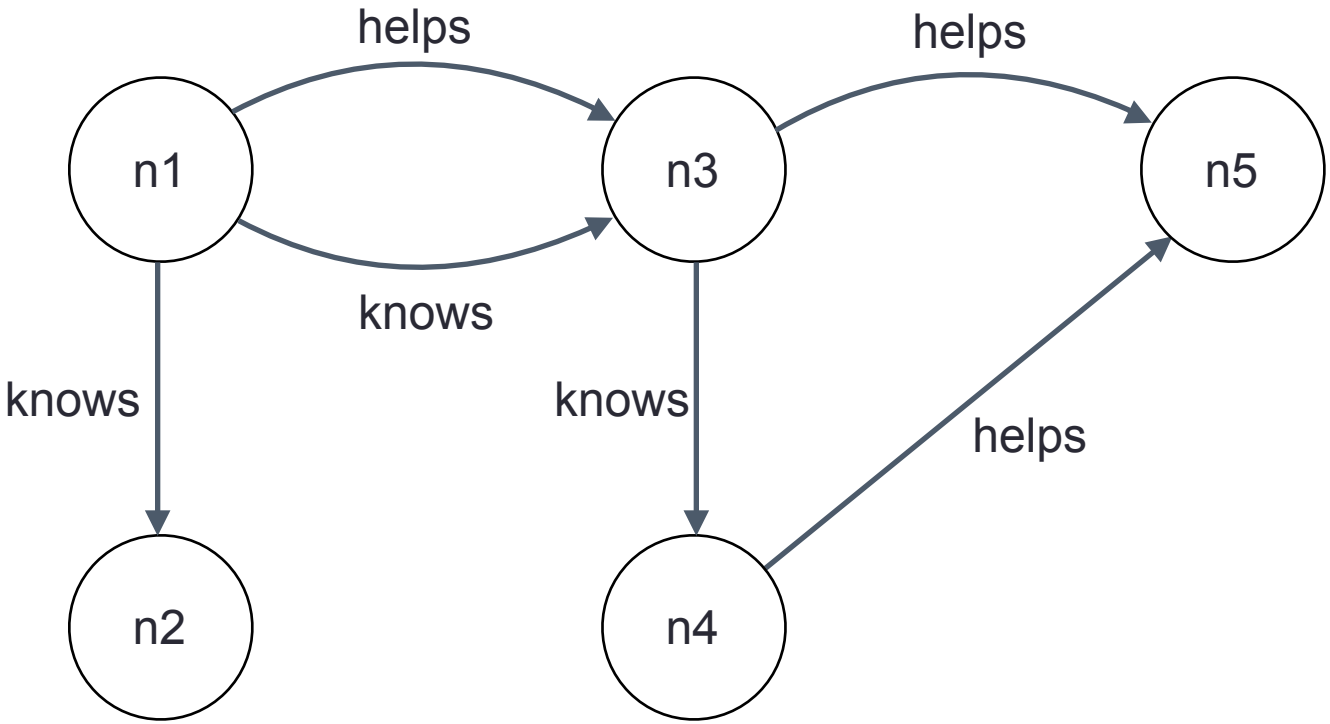
Path Queries

Navigation and Patterns

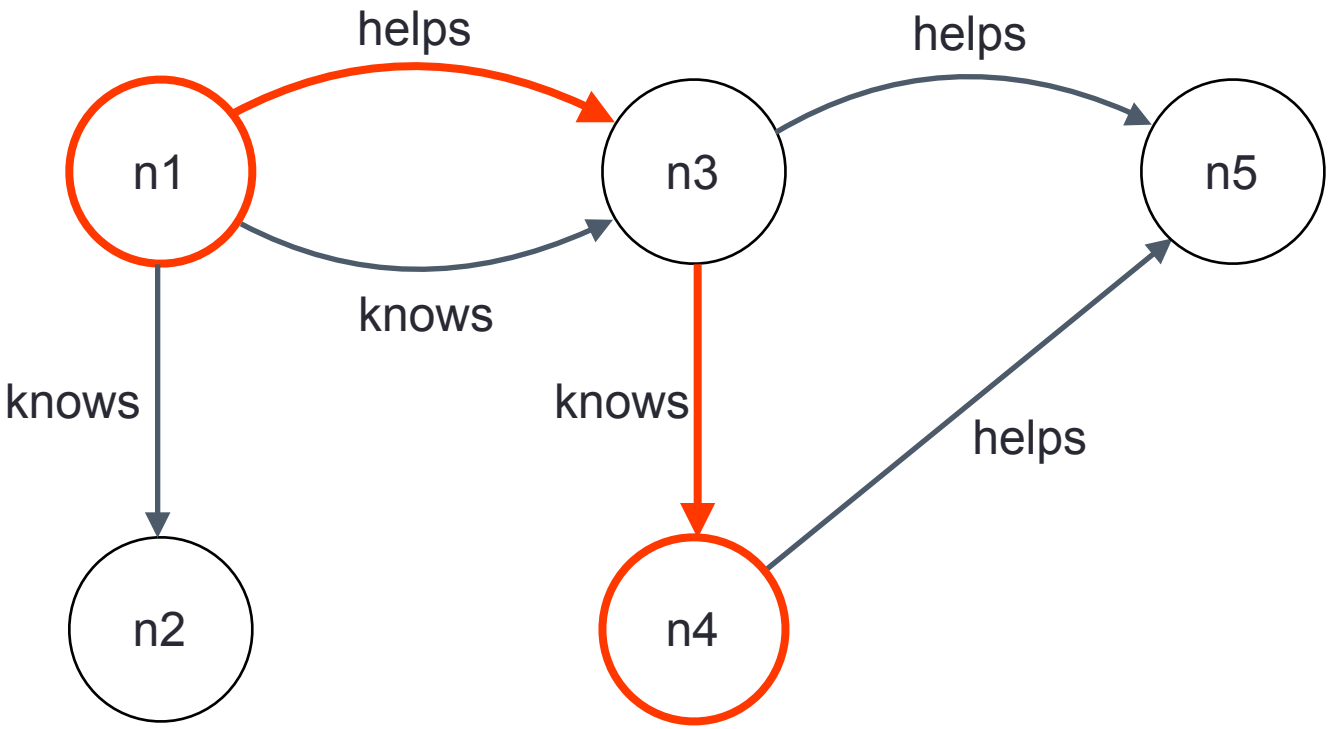
Conjunctions of Path Queries

Beyond Patterns

Notation

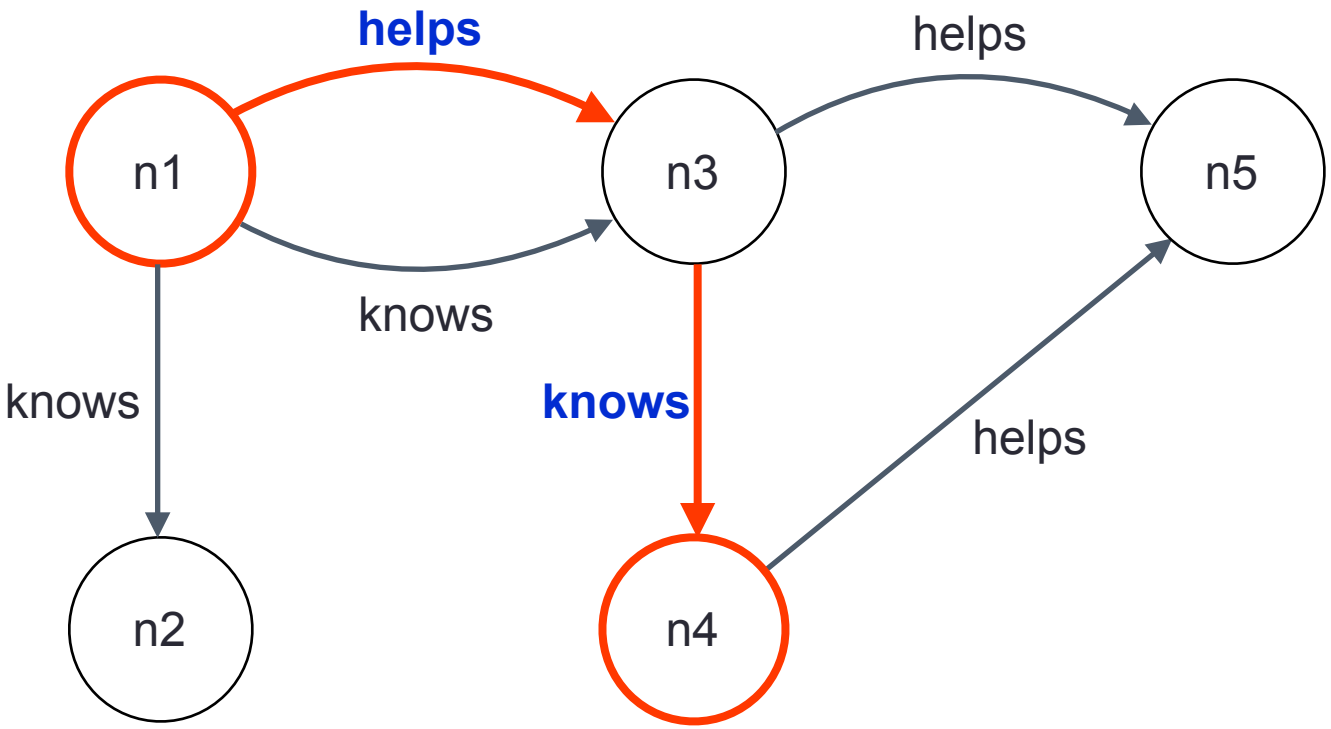


Notation



This is a path
between n1 and n4

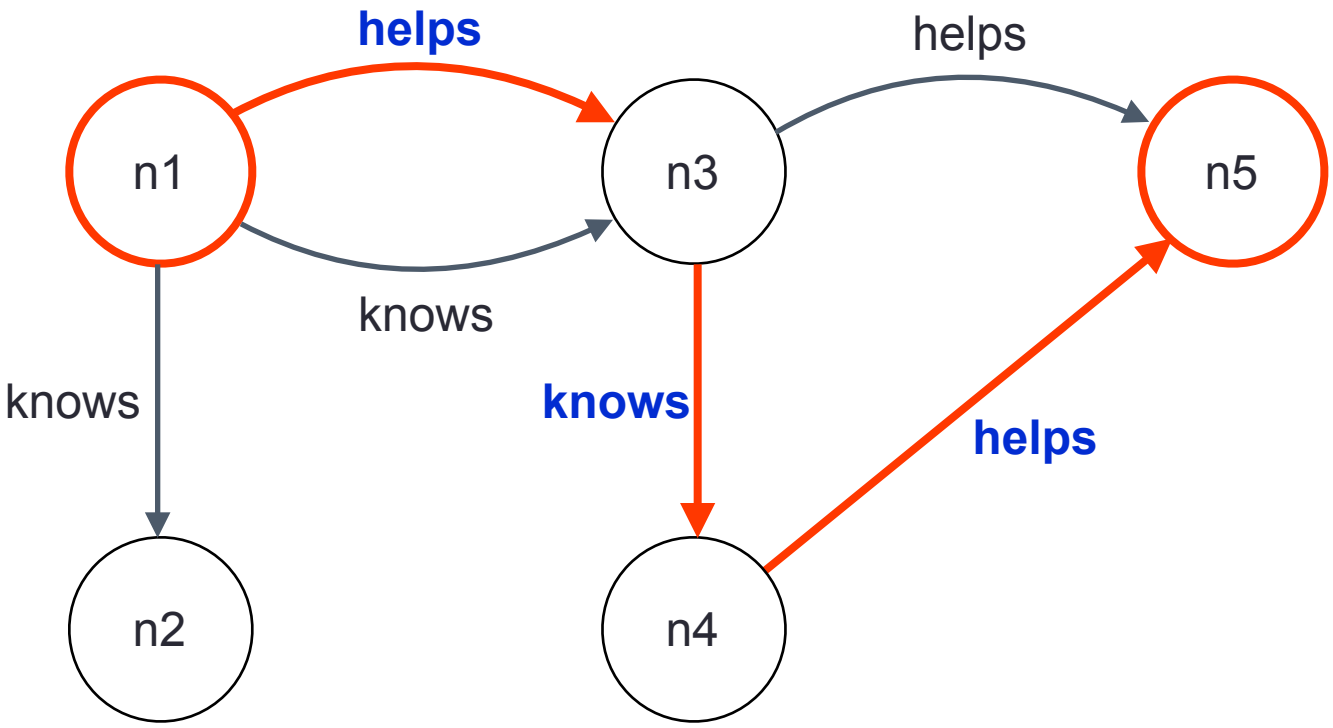
Notation



This is a path
between n1 and n4

The label of this path is
helps knows

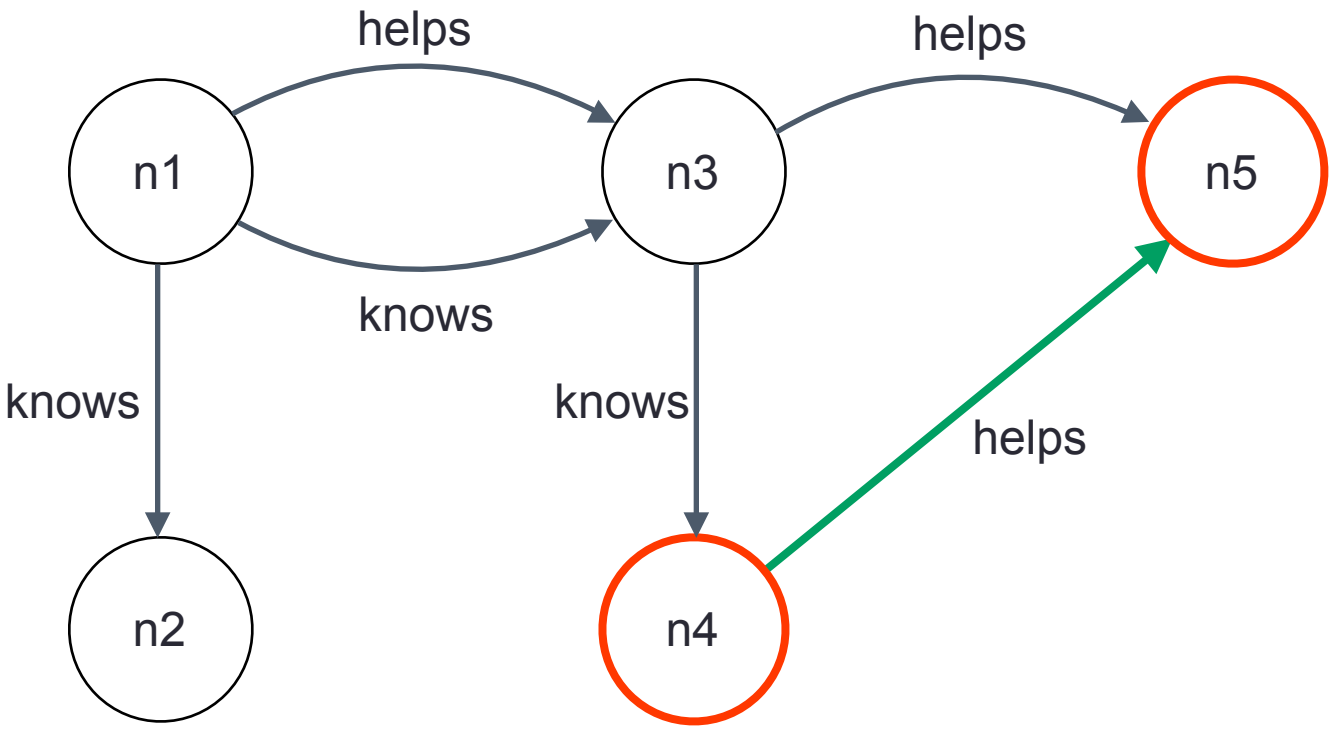
Notation



This is a path
between n1 and n5

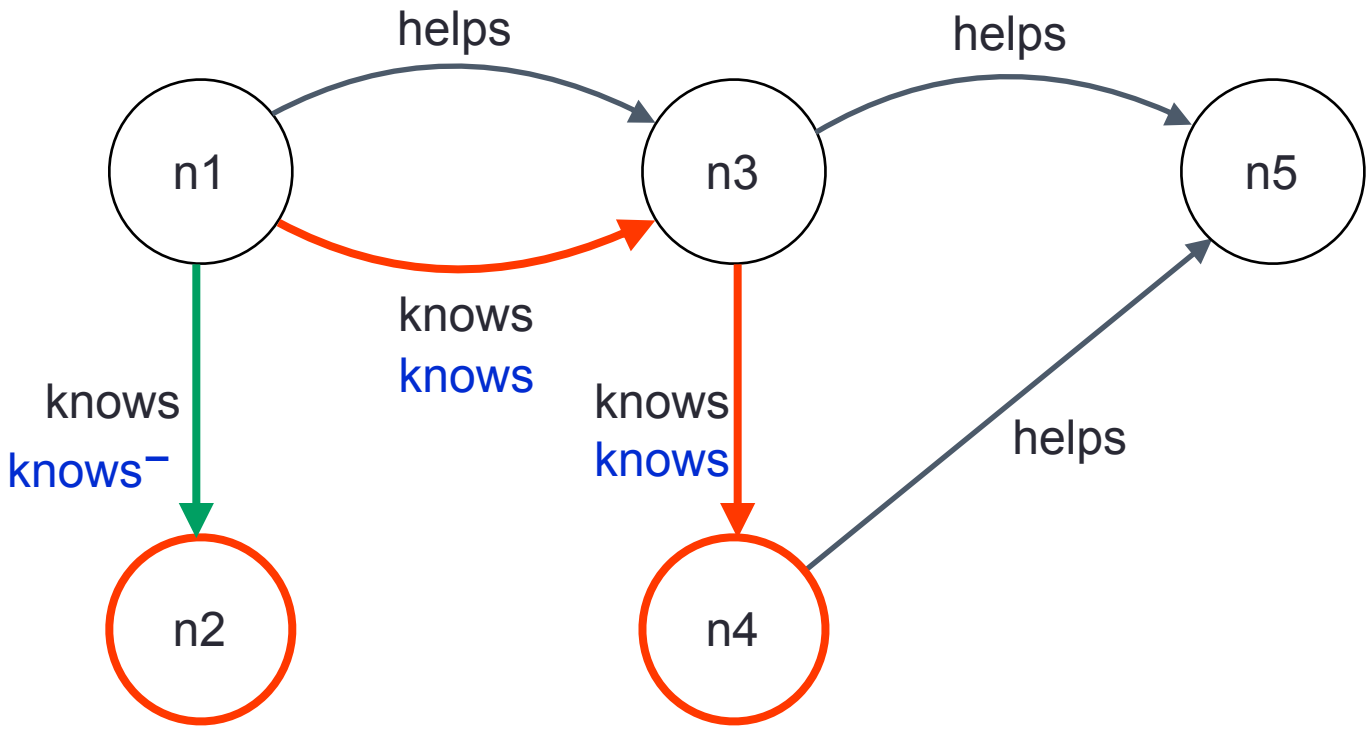
The label of this path is
helps knows helps

Notation



n5 is connected to n4
via an inverse edge, **helps⁻**

Notation



The label of this path is

knows⁻ knows knows

Navigational Primitives:

Regular Path Queries (RPQs):

Select pairs of nodes connected by a path,
whose label conforms to a regular expression

Navigational Primitives:

Regular Path Queries (RPQs):

Select pairs of nodes connected by a path,
whose label conforms to a regular expression

Two-way Regular Path Queries (2RPQs):

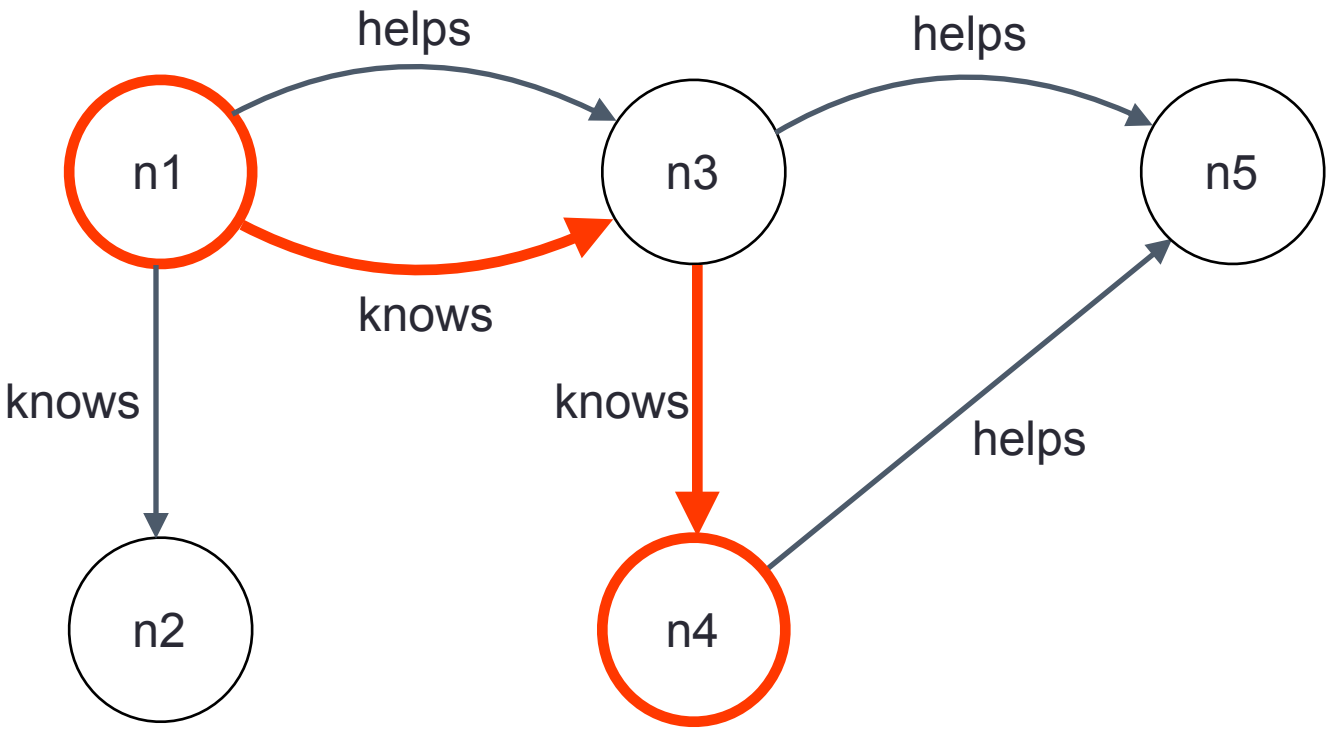
Select pairs of nodes connected by a path,
whose label conforms to a regular expression with inverses

Example of an RPQ

knows+

Find nodes connected by a sequence of knows edges

Example of an RPQ



knows+

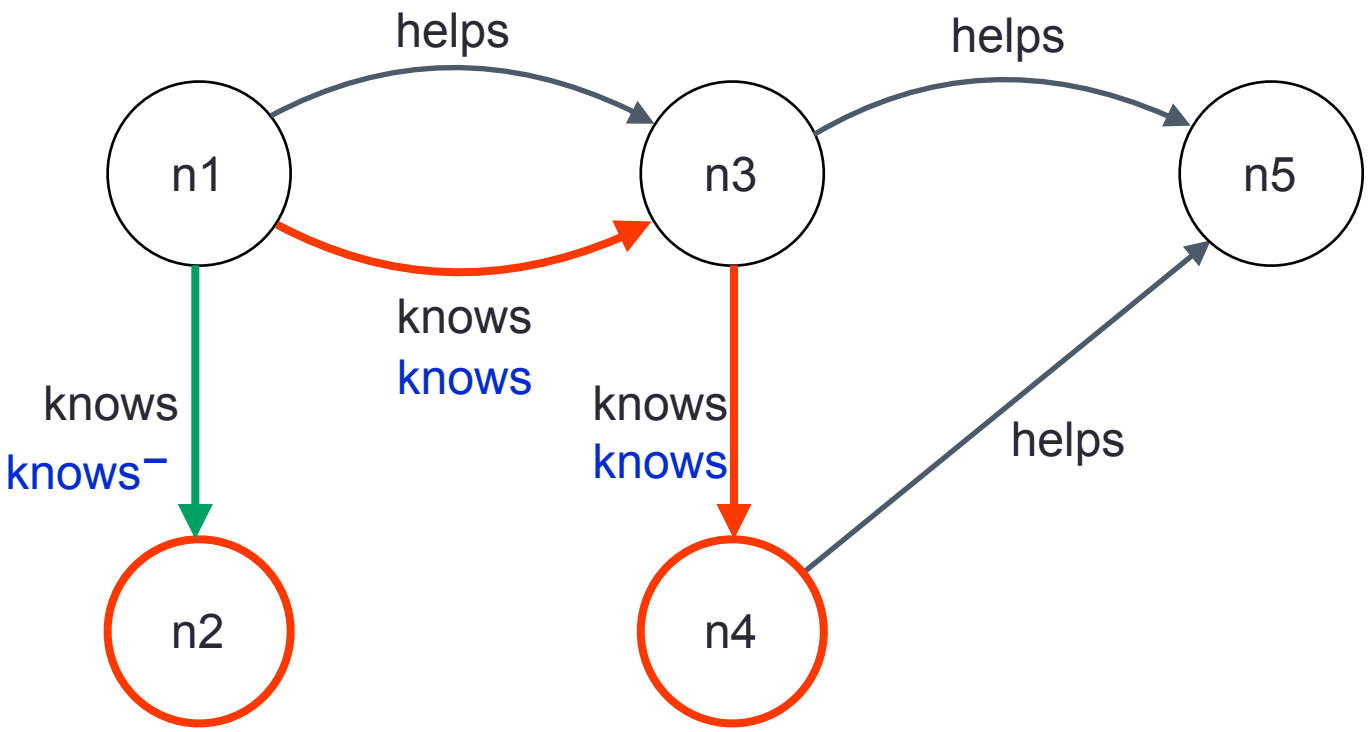
Find nodes connected by a sequence of knows edges

Example of a 2RPQ

$(\text{knows} \mid \text{knows}^{-})^{+}$

Find nodes connected by a sequence of knows edges,
(forward or backward)

Example of a 2RPQ



(knows | knows⁻)⁺

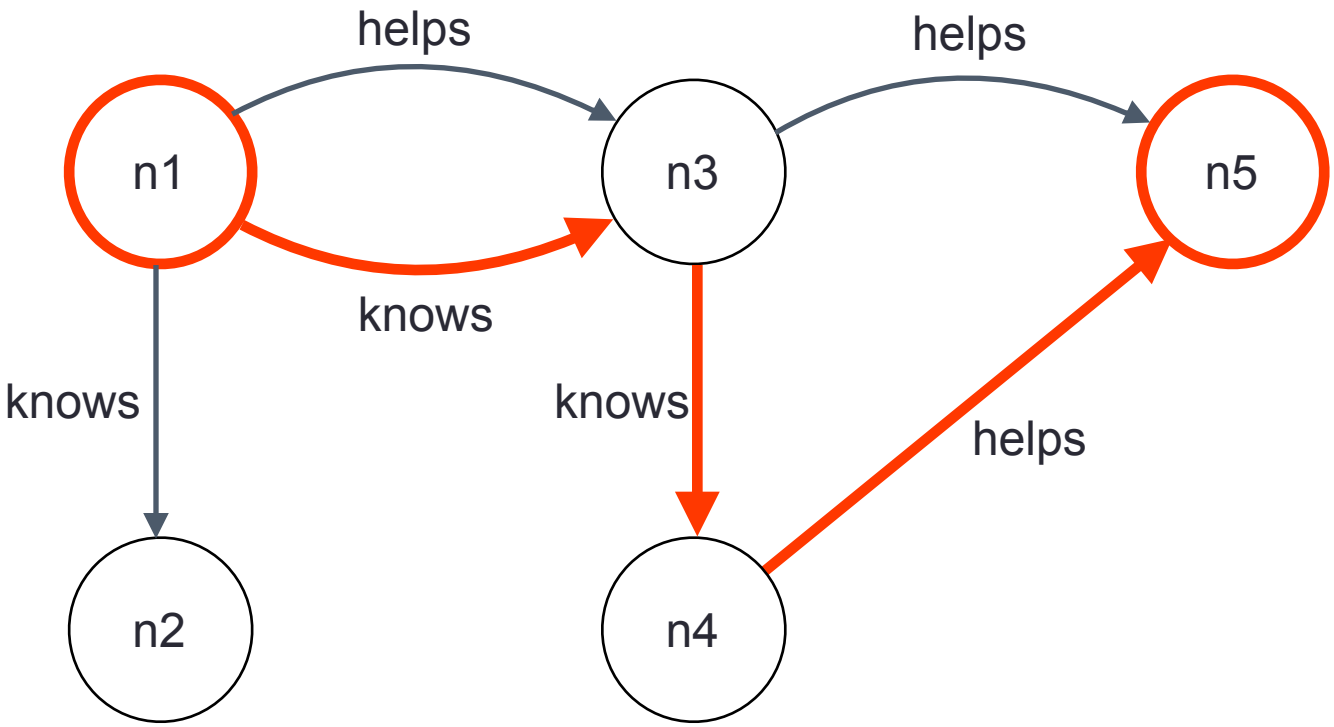
Find nodes connected by a sequence of knows edges,
(forward or backward)

Another 2RPQ (this query is also an RPQ)

(knows l helps)+

Find nodes connected by a sequence of knows edges or helps edges

Another 2RPQ (this query is also an RPQ)



(knows | helps)+

Find nodes connected by a sequence of knows edges or helps edges

Navigational Primitives:

Regular Path Queries (RPQs):

Select pairs of nodes connected by a path,
whose label conforms to a regular expression

Two-way Regular Path Queries (2RPQs):

Select pairs of nodes connected by a path,
whose label conforms to a regular expression with inverses

Navigational Primitives continued:

Regular Path Queries (RPQs):

Select pairs of nodes connected by a path,
whose label conforms to a regular expression

Two-way Regular Path Queries (2RPQs):

Select pairs of nodes connected by a path,
whose label conforms to a regular expression with inverses

Nested Path Queries (NPQs):

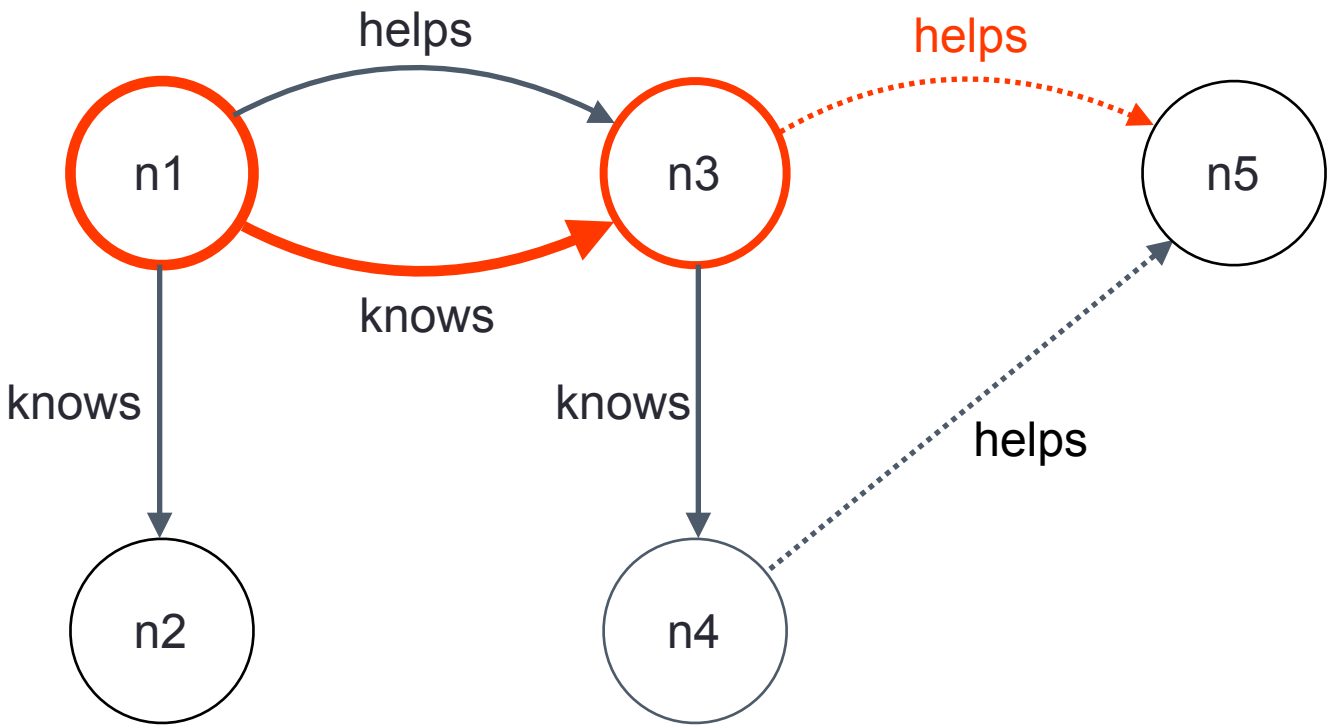
Adds existential test [] to 2RPQs

Example of an NPQ

(knows[helps])+

Find nodes connected by a sequence of knows edges, where each node in the path also helps someone

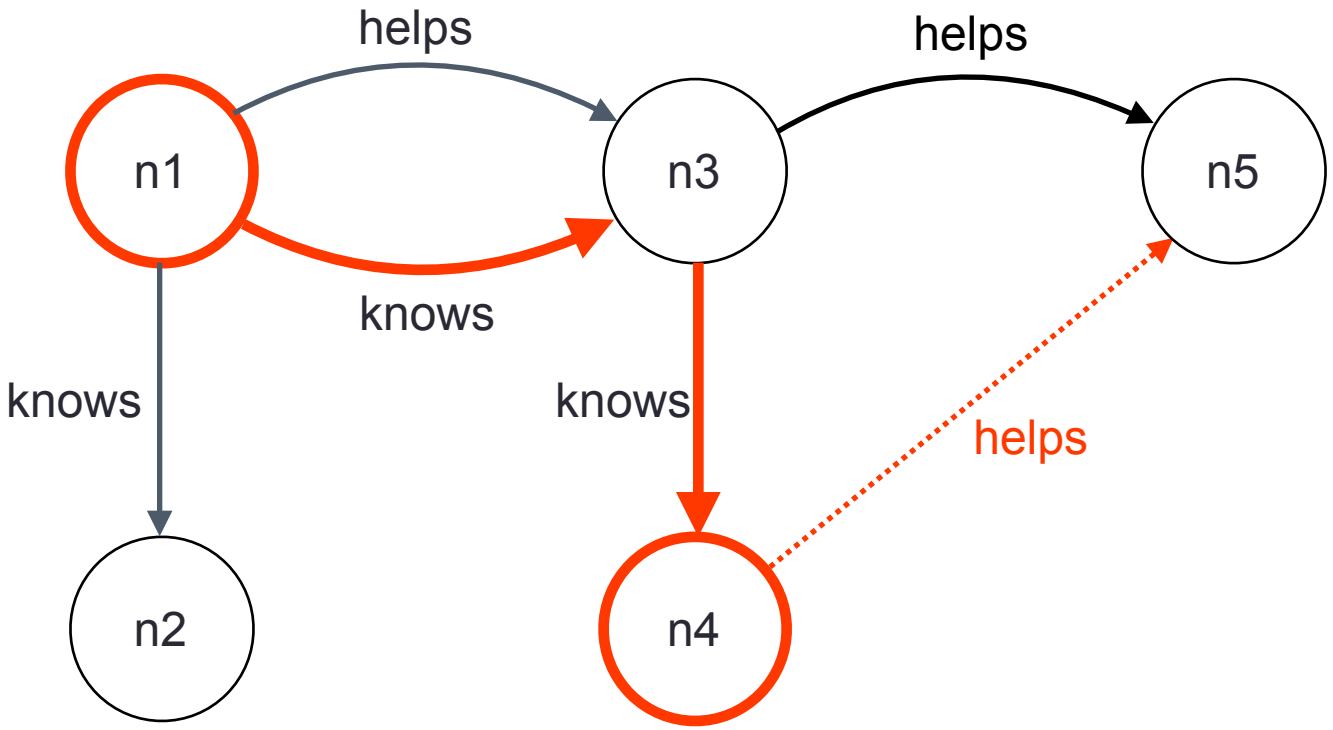
Example of an NPQ



(knows[helps])+

Find nodes connected by a sequence of knows edges, where each node in the path also helps someone

Example of an NPQ



(knows[helps])+

Find nodes connected by a sequence of knows edges, where each node in the path also helps someone

Evaluation

The evaluation of an Path Query Q over a graph G is

all pairs (u,v) of nodes such that

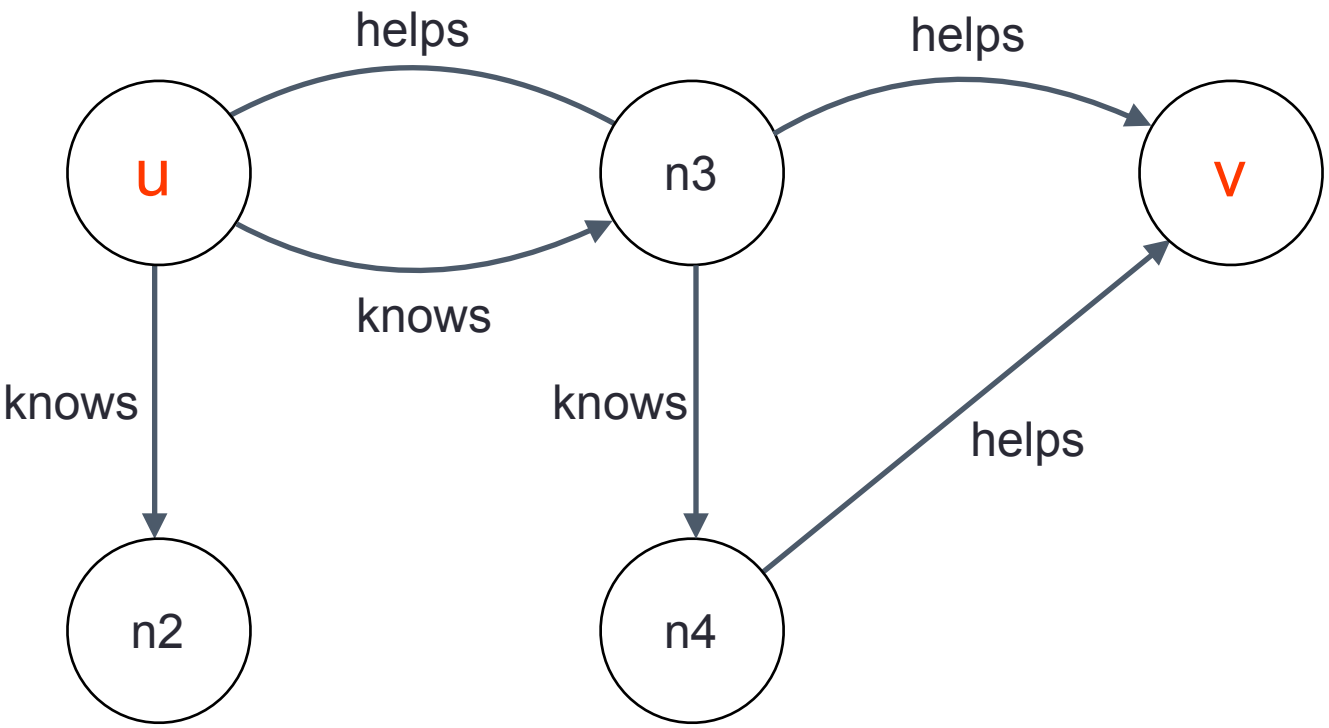
- u is connected to v by a path satisfying Q

Evaluation problem (RPQs):

Given graph G, nodes u,v from G, path query Q.
Is (u,v) in the evaluation of Q over G?

Evaluation problem (RPQs):

Given graph G, nodes u,v from G, path query Q.
Is (u,v) in the evaluation of Q over G?



Graph G, two nodes u and v

knows+

Path Query Q

Evaluation problem (RPQs):

Given graph G, nodes u,v from G, path query Q.
Is (u,v) in the evaluation of Q over G?

Algorithm:

1.- Transform query Q into an automata
2.- Graph G also an automata, u is the starting state,
v is the final state

Evaluation problem (RPQs):

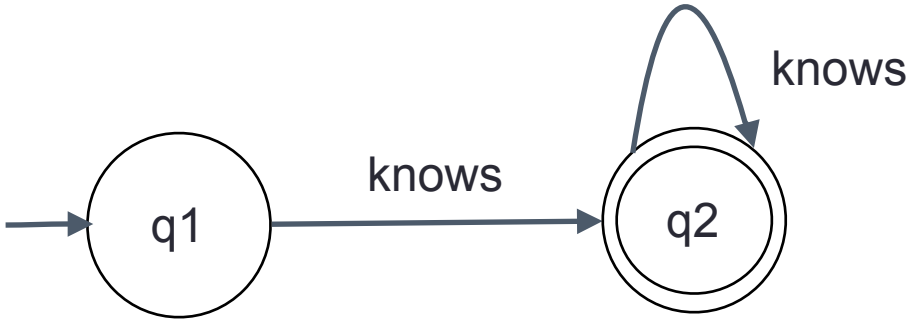
Given graph G, nodes u,v from G, path query Q.
Is (u,v) in the evaluation of Q over G?

Algorithm:

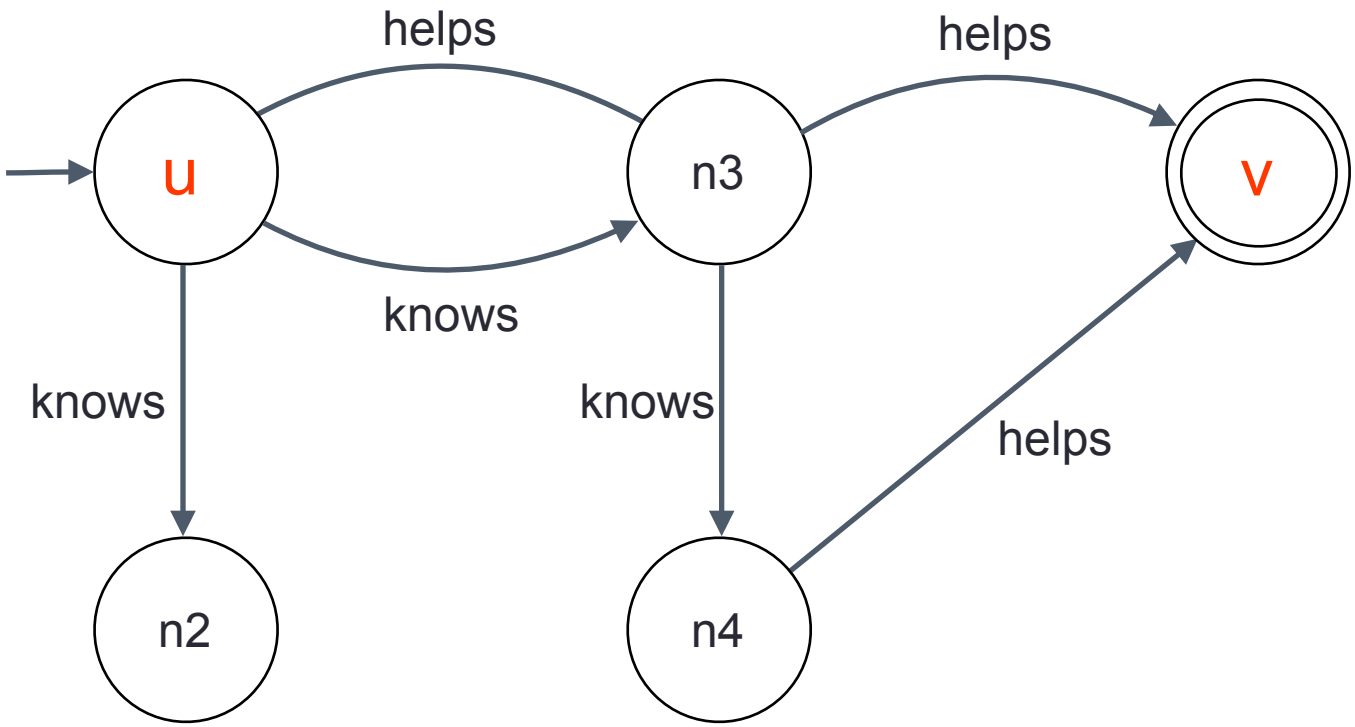
1.- Transform query Q into an automata
2.- Graph G also an automata, u is the starting state,
v is the final state
3.- Compute the cross product:
4.- If nonempty, (u,v) is in the evaluation of Q over G

Transform query and graph into automata

Query Q
(as NFA)

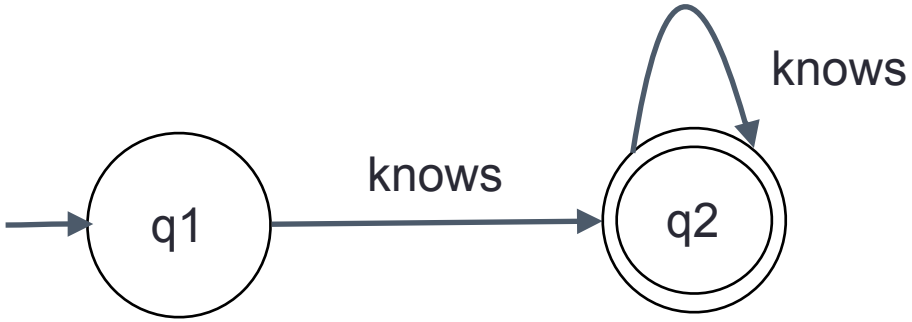


Graph G
(as NFA)
(u is initial state)
(v is final state)



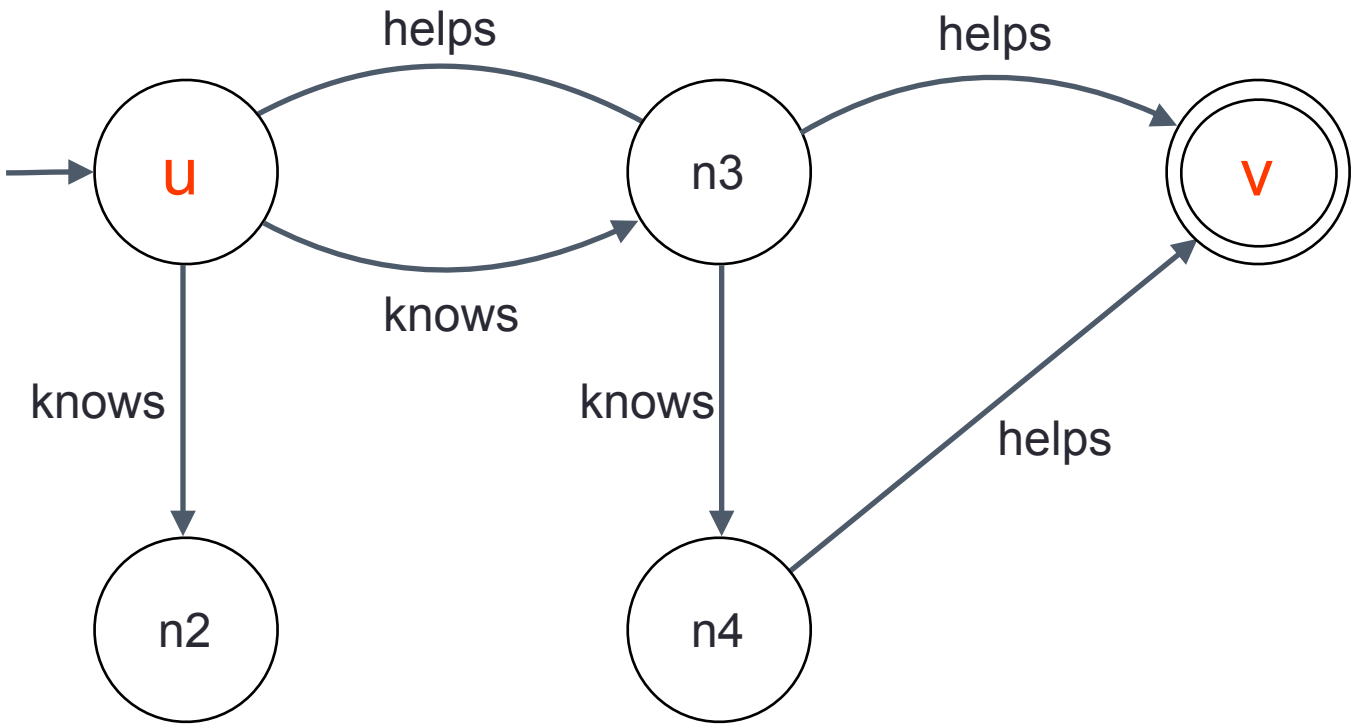
Compute cross product

Query Q
(as NFA)

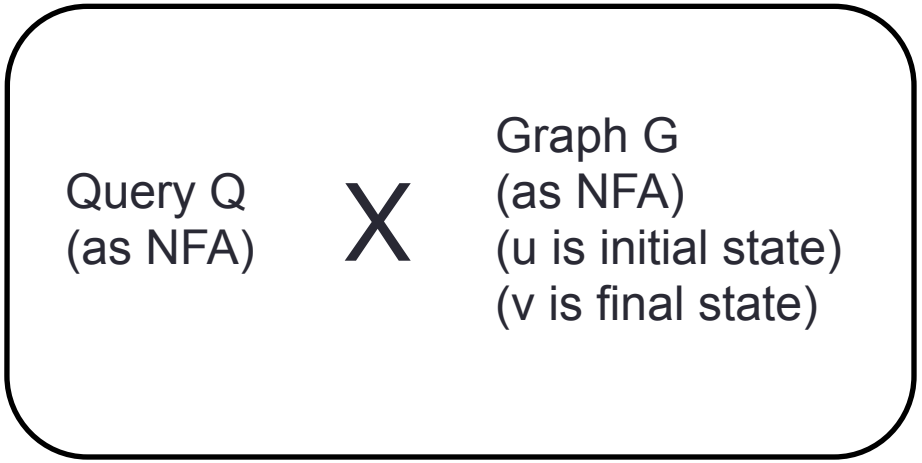


X

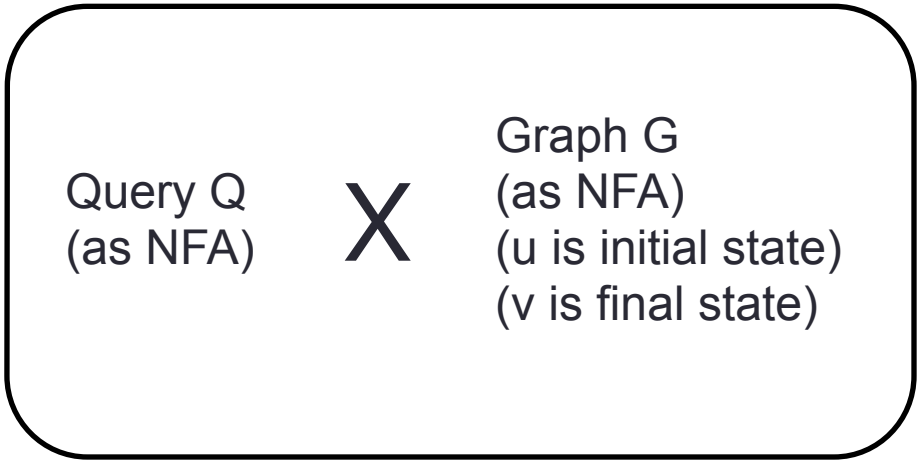
Graph G
(as NFA)
(u is initial state)
(v is final state)



If cross product accepts a word,
(u,v) belongs to the evaluation of Q



If cross product accepts a word, (u,v) belongs to the evaluation of Q



This is also an automaton.

(the words of this automaton are the paths between u and v that satisfy Q)

Evaluation problem:

Given graph G, nodes u,v from G, path query Q.
Is (u,v) in the evaluation of Q over G?

Evaluation of RPQs is: [Cruz, Mendelzon, Wood 87]

- NLogSpace-complete
- Linear in the graph, if the query is fixed (data complexity)

Evaluation problem:

Given graph G, nodes u,v from G, path query Q.
Is (u,v) in the evaluation of Q over G?

Evaluation of RPQs is: [Cruz, Mendelzon, Wood 87]

- NLogSpace-complete
- Linear in the graph, if the query is fixed (data complexity)

Same holds for 2RPQs ([Calvanese, De Giacomo, Lenzerini, Vardi 00]):
algorithm: add first all inverses to the graph.
each 2RPQ is now an RPQ over this new graph

Evaluation problem:

Given graph G, nodes u,v from G, path query Q.
Is (u,v) in the evaluation of Q over G?

Evaluation of NPQs is: [Perez, Arenas, Gutierrez 10]

- Linear in the graph, if the query is fixed (data complexity)
- NLogSpace-complete if the number of nested []'s is fixed
- in PTIME otherwise

Evaluation problem:

Given graph G, nodes u,v from G, path query Q.
Is (u,v) in the evaluation of Q over G?

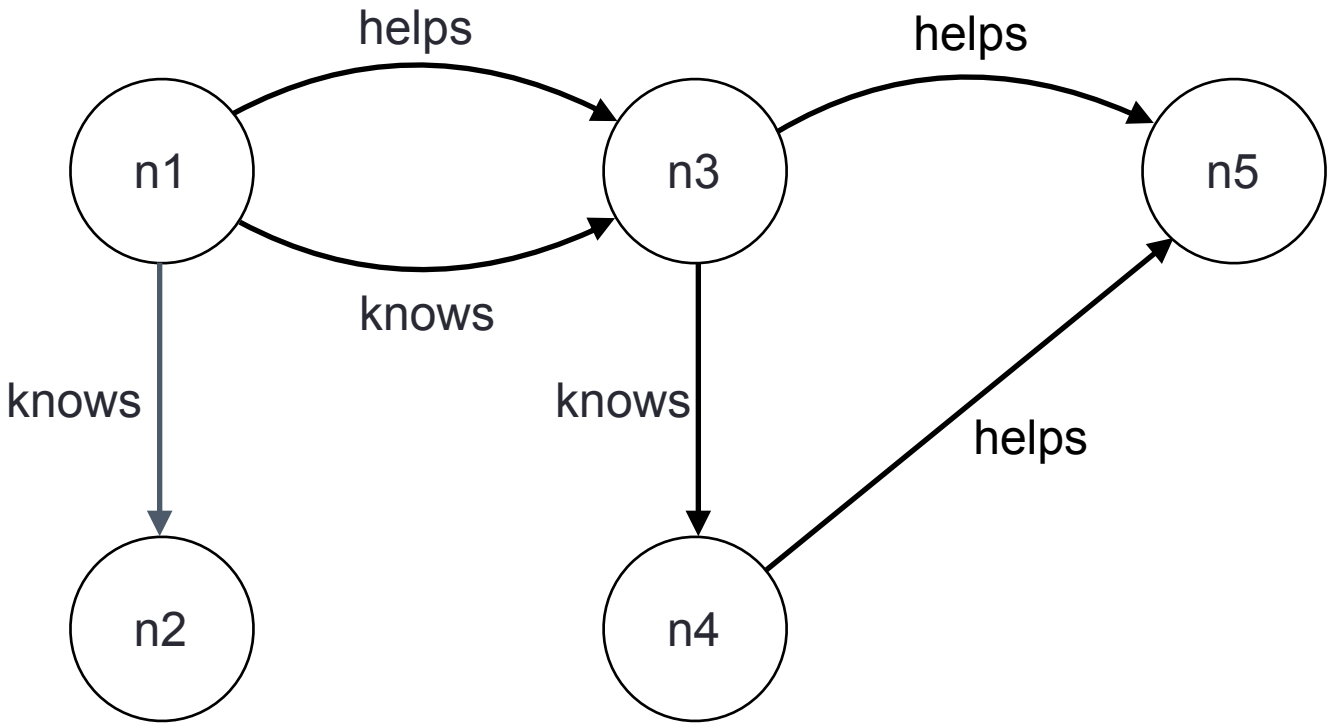
Evaluation of NPQs is: [Perez, Arenas, Gutierrez 10]

- Linear in the graph, if the query is fixed (data complexity)
- NLogSpace-complete if the number of nested []'s is fixed
- in PTIME otherwise

Idea: if Q is u[e]w, then first evaluate e.
add new label e to the Graph (query is now a 2RPQ)

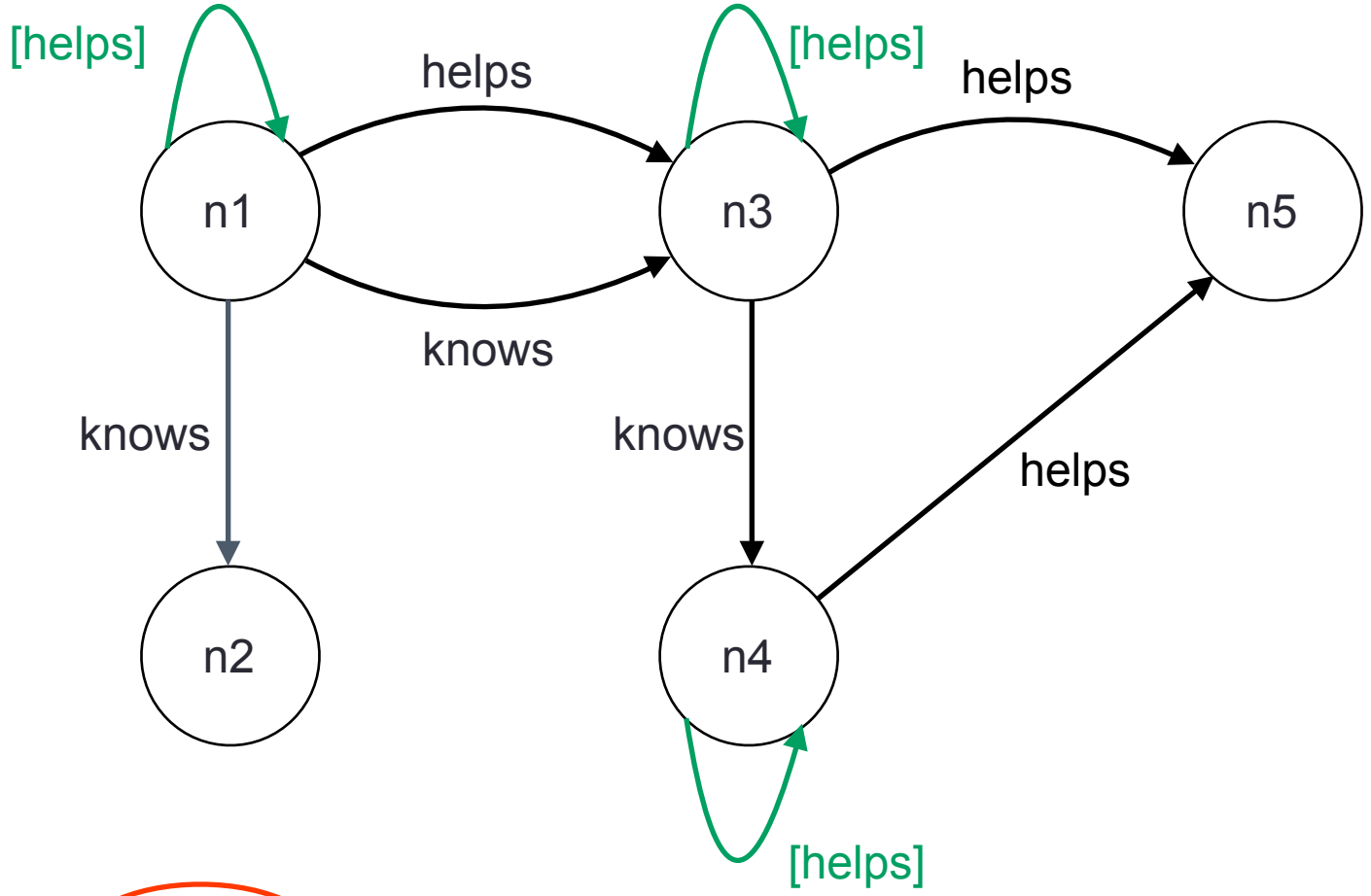
Repeat for nested []s

Evaluation algorithm for NPQs (example)



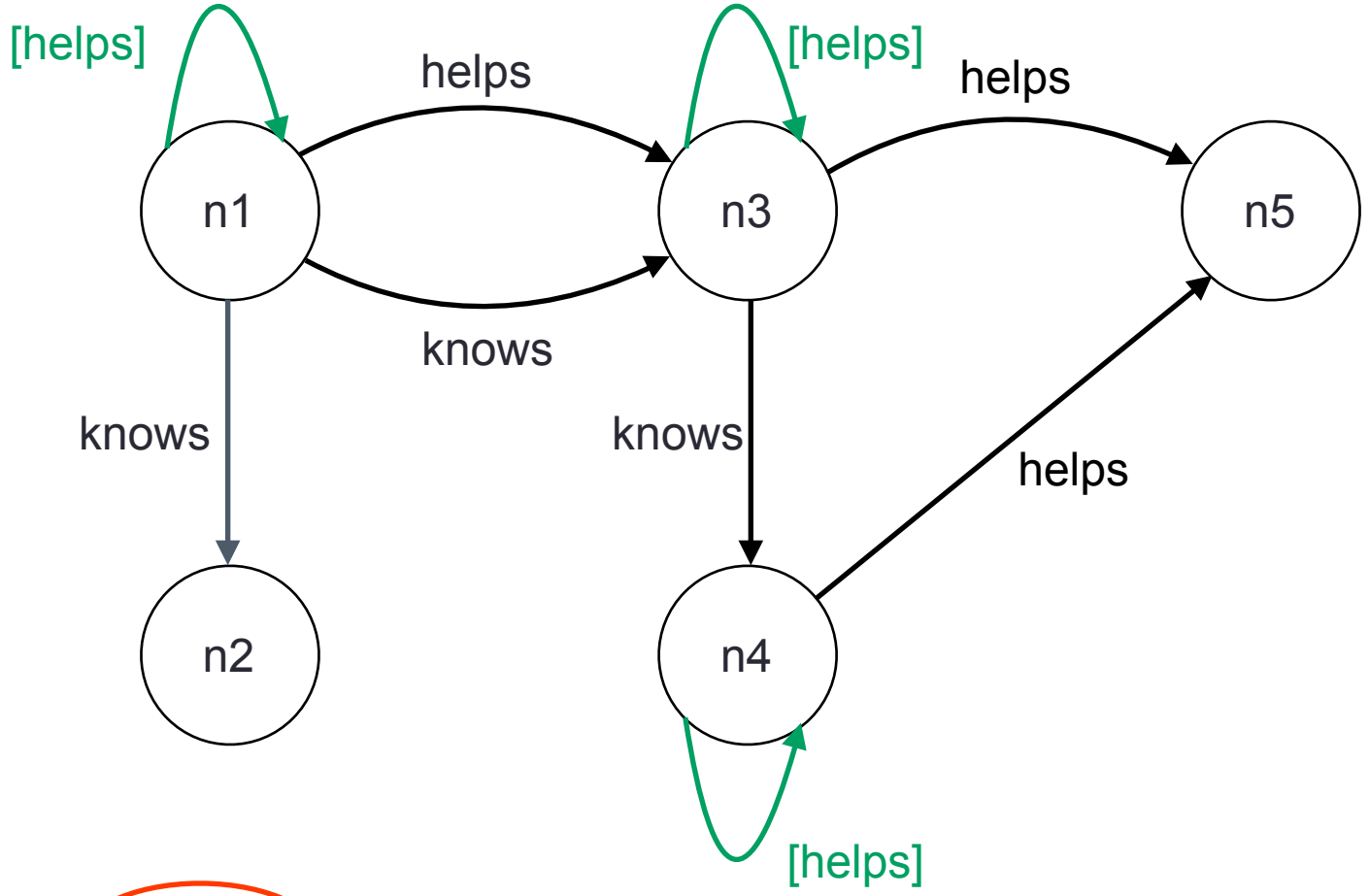
(knows[helps])+

Evaluation algorithm for NPQs (example)



(knows[helps])+

Evaluation algorithm for NPQs (example)

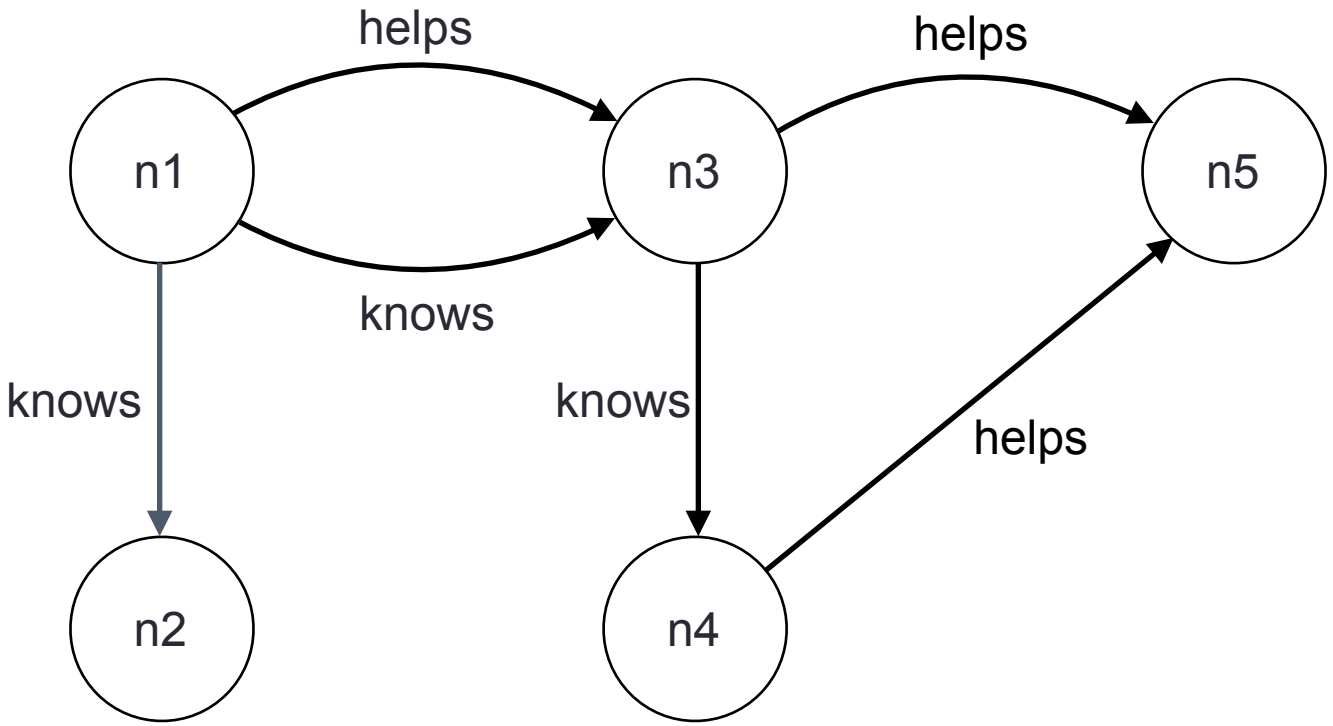


(knows[helps])+

This is now an RPQ!

(knows[helps])+

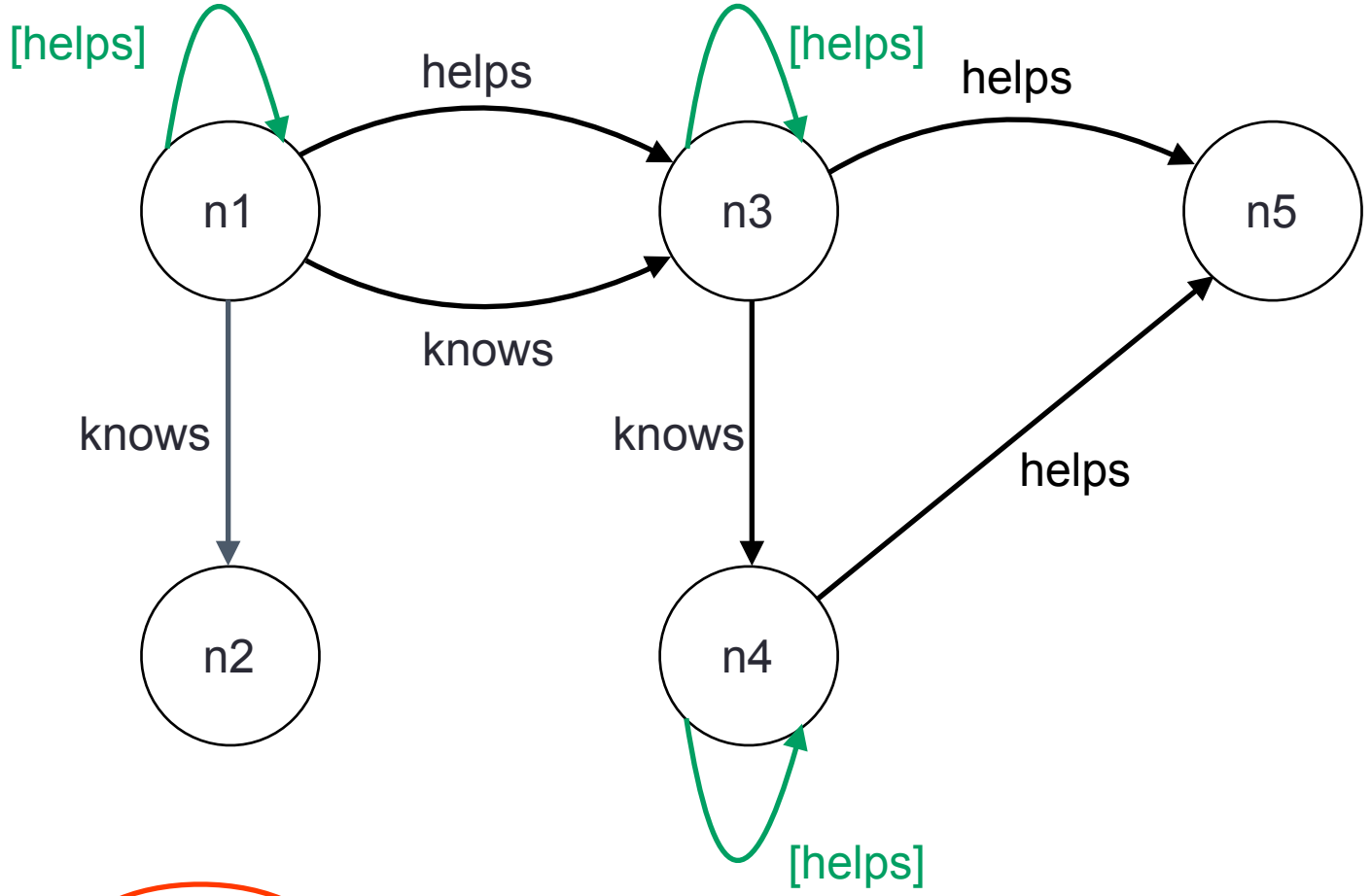
Evaluation algorithm for NPQs (example)



(knows[helps])+

Need to do this incrementally, inside out

Evaluation algorithm for NPQs (example)



(knows[helps])+

Need to do this incrementally, inside out

Outline

Path Queries (definition, evaluation)

Navigation and Patterns

Conjunctions of Path Queries (definition, evaluation)

Beyond Patterns

As a query language, path queries are not enough,

need to mix navigation with patterns

First we see how to represent patterns with logic

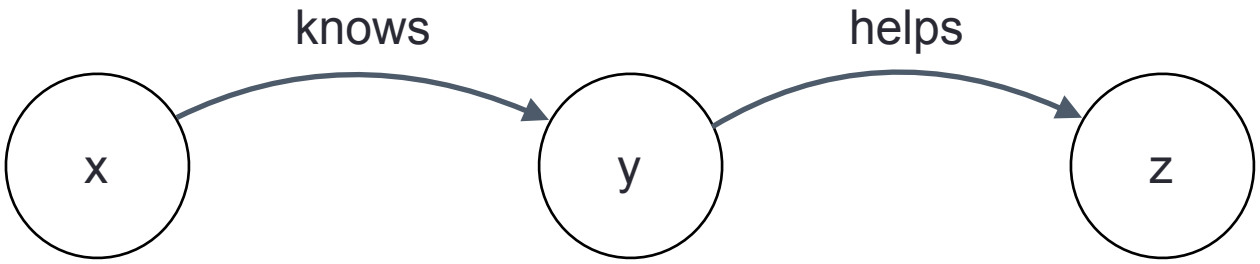
Logical Representation of patterns

Patterns correspond to Conjunctive Queries,
over the vocabulary of all labels and their inverses

(Conjunctive Queries: conjunctions of atoms,
some variables existentially quantified)

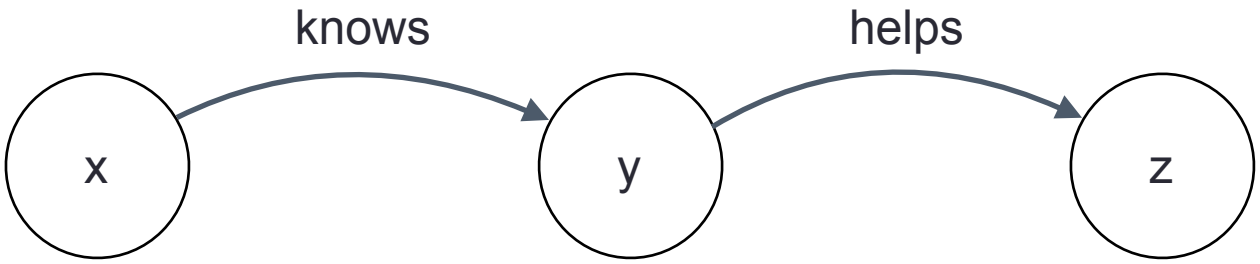
Logical Representation of patterns

Find all people that know a helper



Logical Representation of patterns

Find all people that know a helper

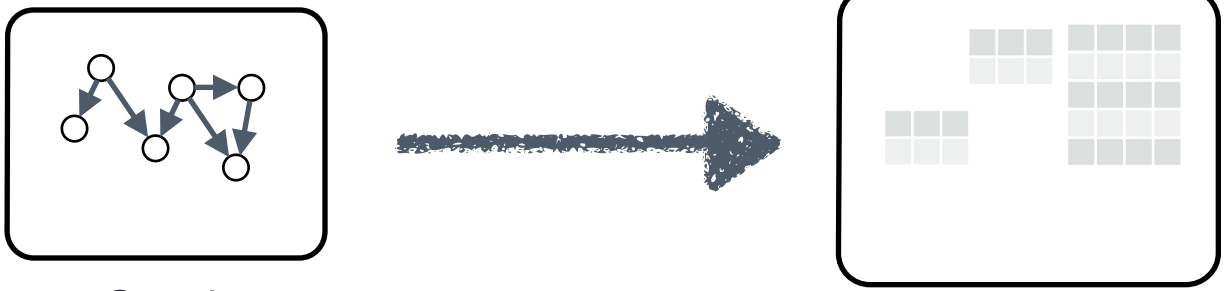


$$Q(x) \;=\; \exists y \exists z \big(\text{knows}(x, y) \wedge \text{helps}(y, z) \big)$$

Logical Representation of patterns

Patterns correspond to Conjunctive Queries,
over the vocabulary of all labels and their inverses

Pattern → **Conjunctive Queries**

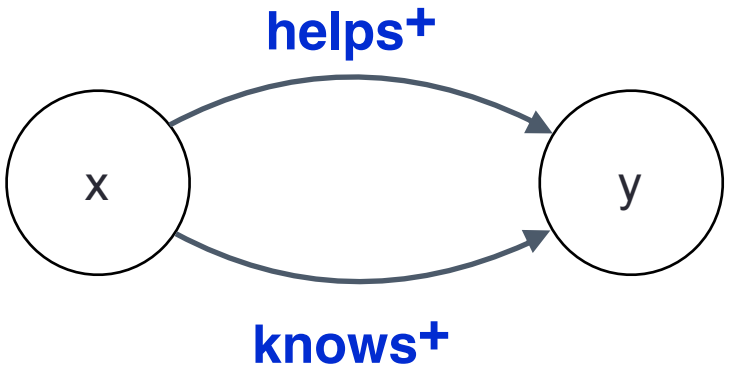


Graph

Relacional representation

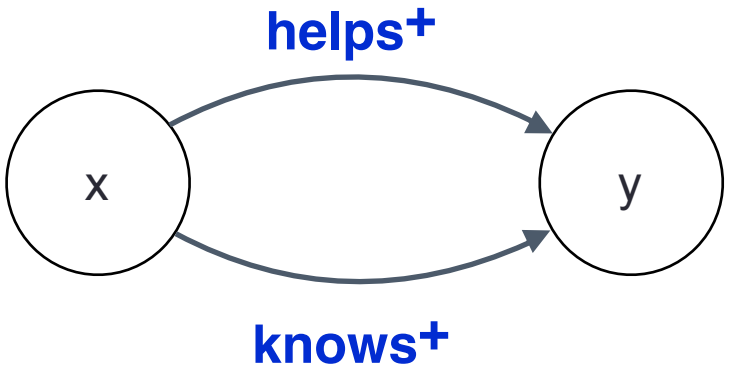
Patterns + Navigation

Look for nodes x and y connected by
- a path of knows, and
- a path of helps



Patterns + Navigation

Look for nodes x and y connected by
- a path of knows, and
- a path of helps



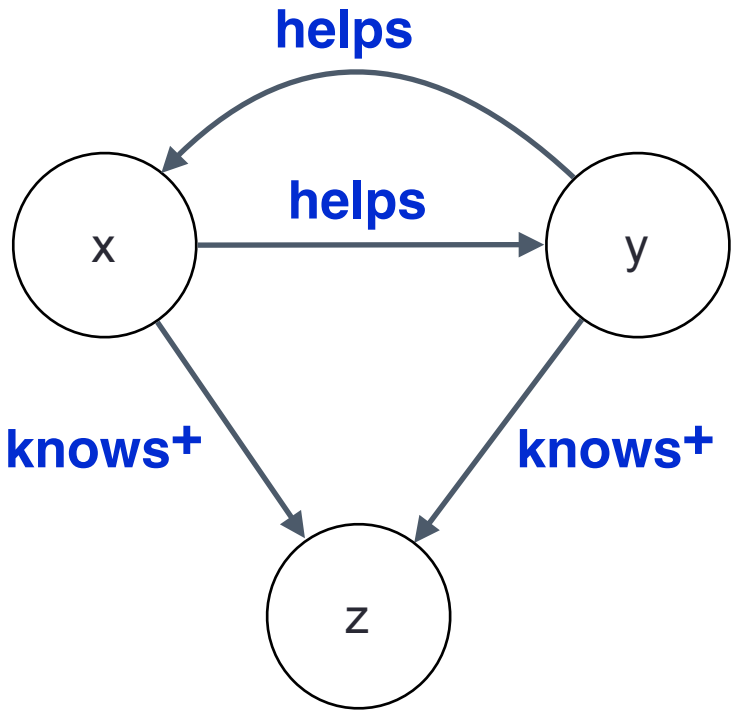
$$Q(x, y) \;=\; \left(\text{knows}^{+}(x, y) \wedge \text{helps}^{+}(x, y)\right)$$

We represent navigational patterns as CQs over the vocabulary of:

- all labels and their inverses
- any path query

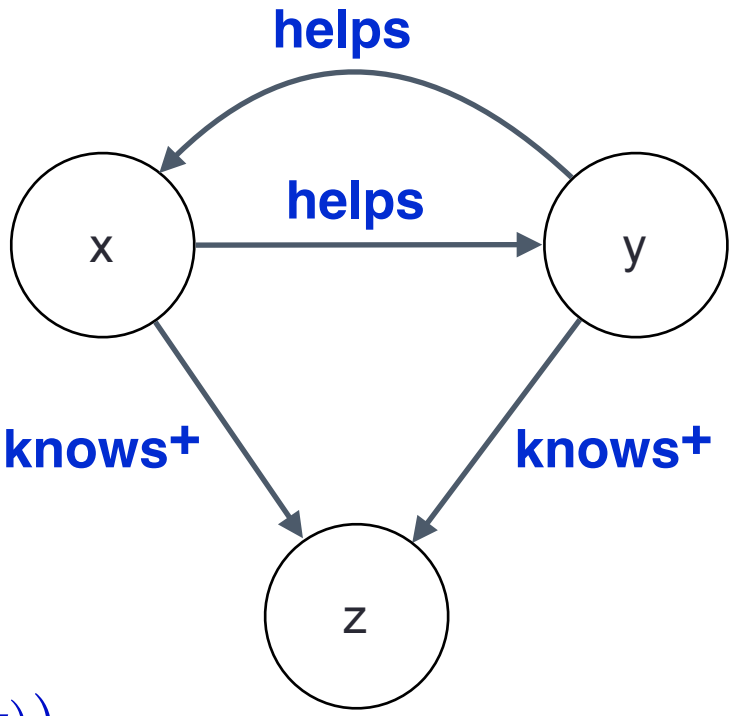
Example of Navigational Pattern

All nodes x and y that have helped each other an have indirect common friend



Example of Navigational Pattern

All nodes x and y that have helped each other
an have indirect common friend



$$Q(x,y) \;=\; \exists z\big(\text{helps}(x,y) \wedge \text{helps}(y,x) \wedge \text{knows}^{+}(x,z) \wedge \text{knows}^{+}(y,z)\big)$$

Conjunctive Regular Path Queries (CRPQs)

$$Q(\bar{z}) \;=\; L_1(x_1, y_1) \;\wedge\; \cdots \;\wedge\; L_n(x_n, y_n)$$

$\bar{z}$ is a tuple of variables in $\{x_1, \ldots, x_n, y_1, \ldots, y_n\}$

$L_1, \ldots, L_n$ are Regular Path Queries (RPQs)

Conjunctive 2-way Regular Path Queries (C2RPQs)

$$Q(\bar{z}) \;=\; L_1(x_1, y_1) \;\wedge\; \cdots \;\wedge\; L_n(x_n, y_n)$$

$\bar{z}$ is a tuple of variables in $\{x_1, \ldots, x_n, y_1, \ldots, y_n\}$

$L_1, \ldots, L_n$ are 2-Way Regular Path Queries (2RPQs)

Conjunctive Nested Path Queries (CNPQs)

$$Q(\bar{z}) \;=\; L_1(x_1, y_1) \;\wedge\; \cdots \;\wedge\; L_n(x_n, y_n)$$

$\bar{z}$ is a tuple of variables in $\{x_1, \ldots, x_n, y_1, \ldots, y_n\}$

$L_1, \ldots, L_n$ are Nested Path Queries (NPQs)

Different primitives give rise to different navigational queries

CRPQs - C2RPQs - CNPQs

All based on the same idea:

edges of patterns can be connected by path queries

Evaluation

Query $Q(\bar{z}) \;=\; L_1(x_1, y_1) \;\wedge\; \cdots \;\wedge\; L_n(x_n, y_n)$

Evaluation

Query $Q(\bar{z}) \;=\; L_1(x_1, y_1) \;\wedge\; \cdots \;\wedge\; L_n(x_n, y_n)$

Tuple $\bar{a}$ is in the evaluation of Q over G

Evaluation

Query $Q(\bar{z}) = L_1(x_1, y_1) \wedge \cdots \wedge L_n(x_n, y_n)$

Tuple $\bar{a}$ is in the evaluation of Q over G if there is mapping

σ: $\{x_1, \ldots, x_n\}$ $\{y_1, \ldots, y_n\}$ → nodes V of G , $\sigma(\bar{z}) = \bar{a}$

Evaluation

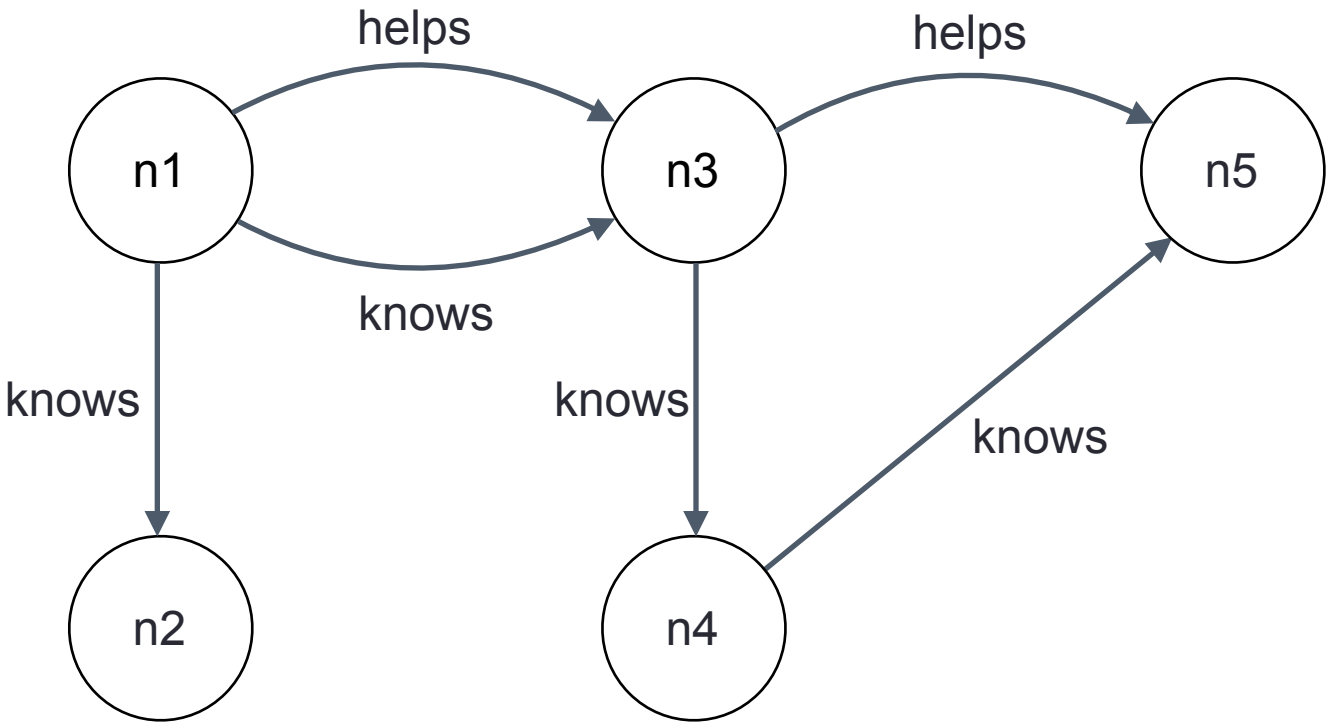
Query $Q(\bar{z}) \;=\; L_1(x_1, y_1) \;\wedge\; \cdots \;\wedge\; L_n(x_n, y_n)$

Tuple $\bar{a}$ is in the evaluation of Q over G if there is mapping

$$\sigma: \begin{matrix}\{x_1, \ldots, x_n\} \\ \{y_1, \ldots, y_n\}\end{matrix} \rightarrow \text{nodes V of G}, \quad \sigma(\bar{z}) = \bar{a}$$

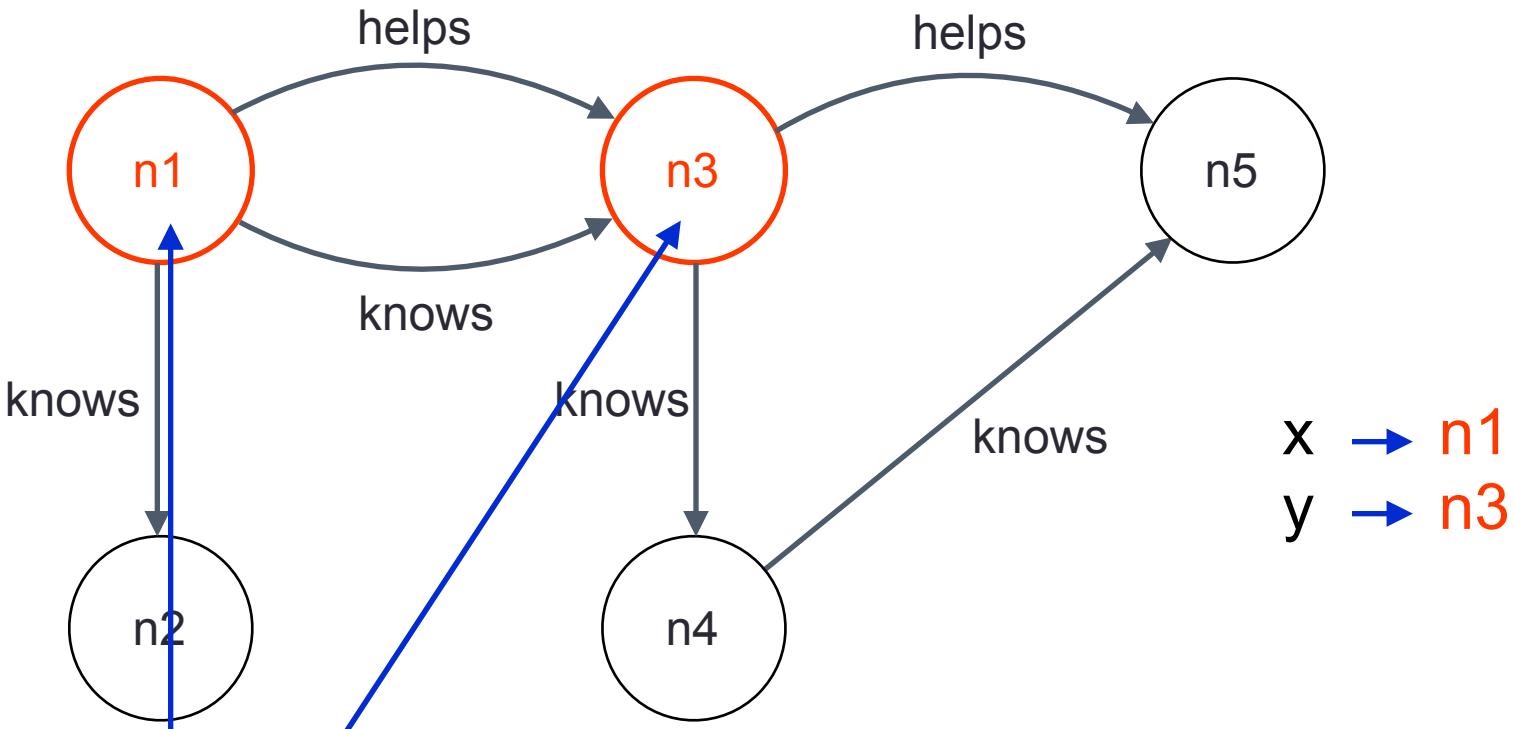
such that each $\sigma(x_i)$ is connected to $\sigma(y_i)$ by means of L_i

Evaluation example



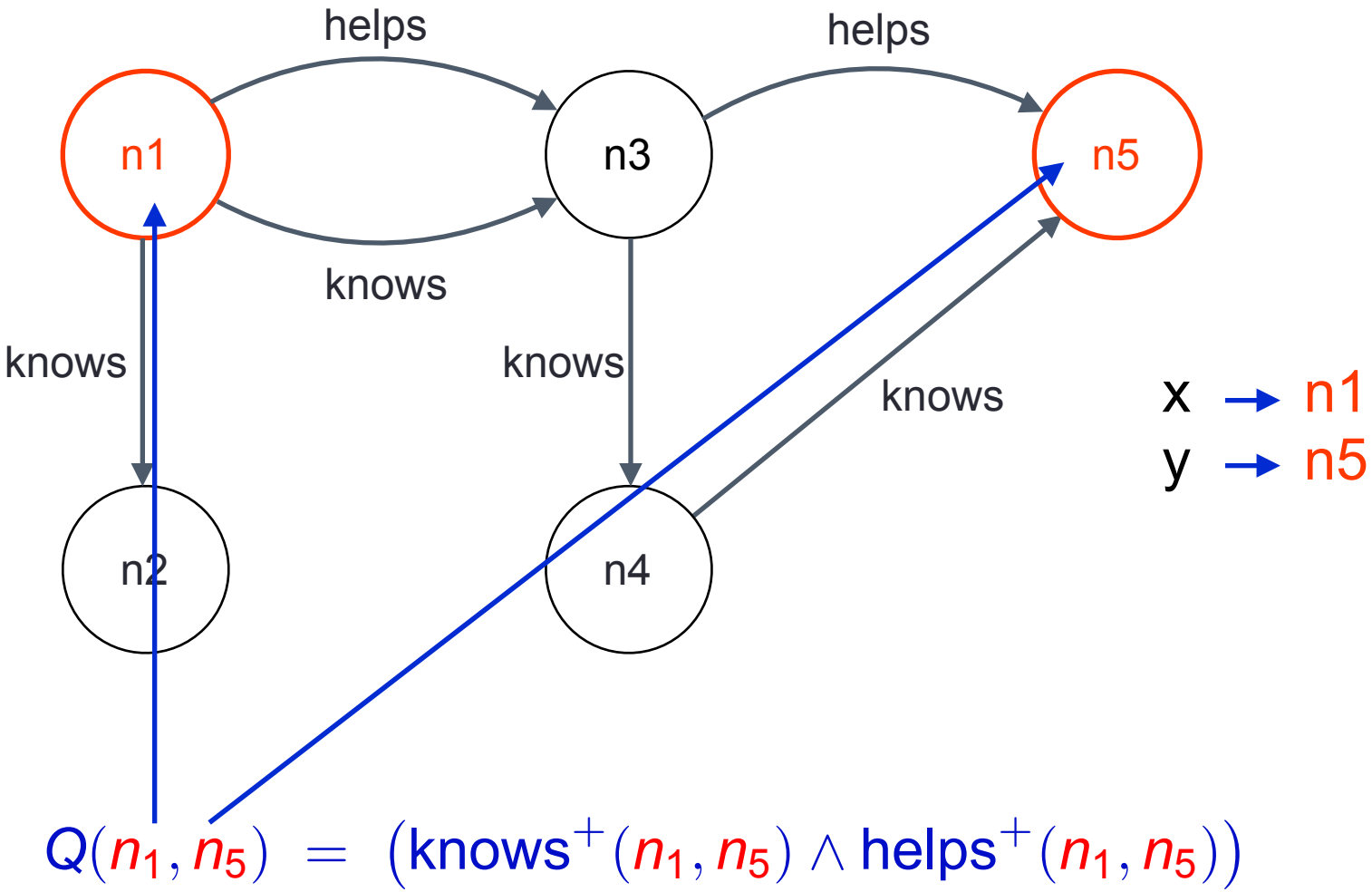
$$Q(x, y) \;=\; \left(\text{knows}^{+}(x, y) \wedge \text{helps}^{+}(x, y)\right)$$

Evaluation example



$$Q(n_1, n_3) = \left(\text{knows}^+(n_1, n_3) \wedge \text{helps}^+(n_1, n_3)\right)$$

Evaluation example



Complexity of Evaluation Problem

(checking if a tuple belongs to the answer of a navigational graph pattern)

Complexity of Evaluation Problem

For CQs already NP-complete:

- guess a mapping, check in PTIME

Complexity of Evaluation Problem

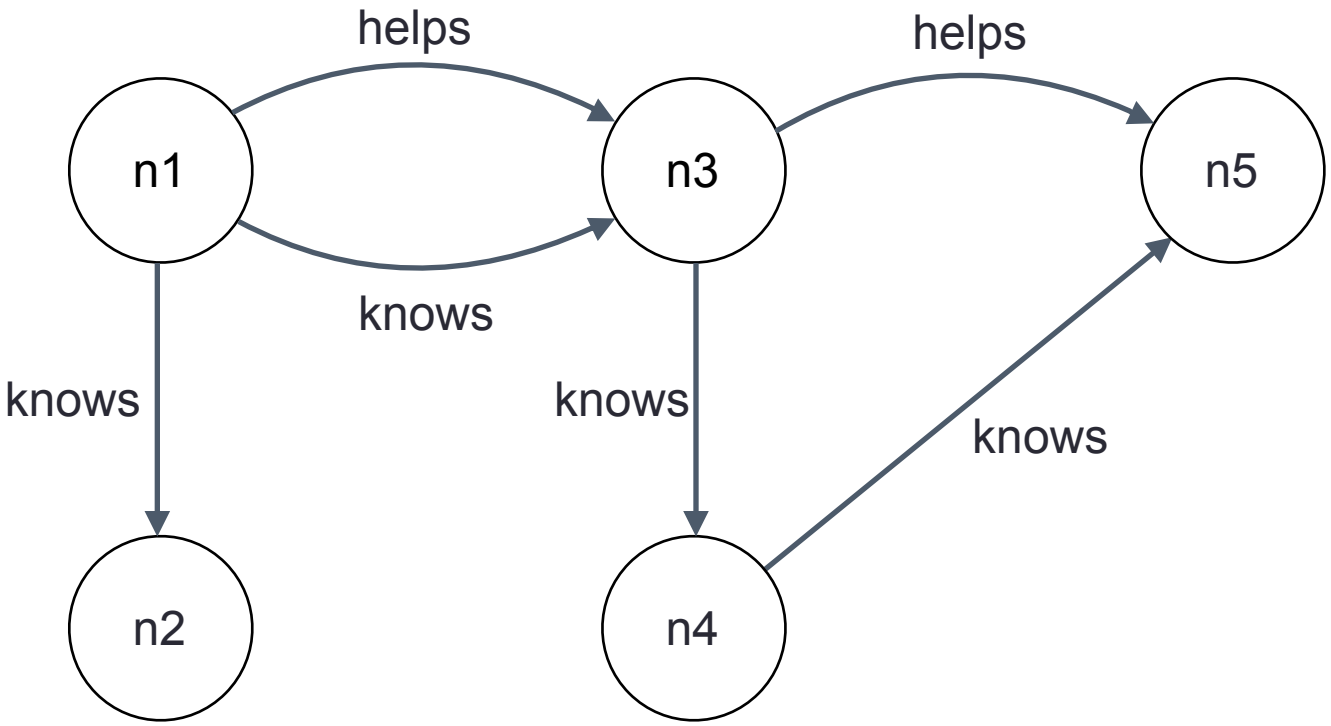
For CQs already NP-complete:
- guess a mapping, check in PTIME

Still NP-complete for CNPQs

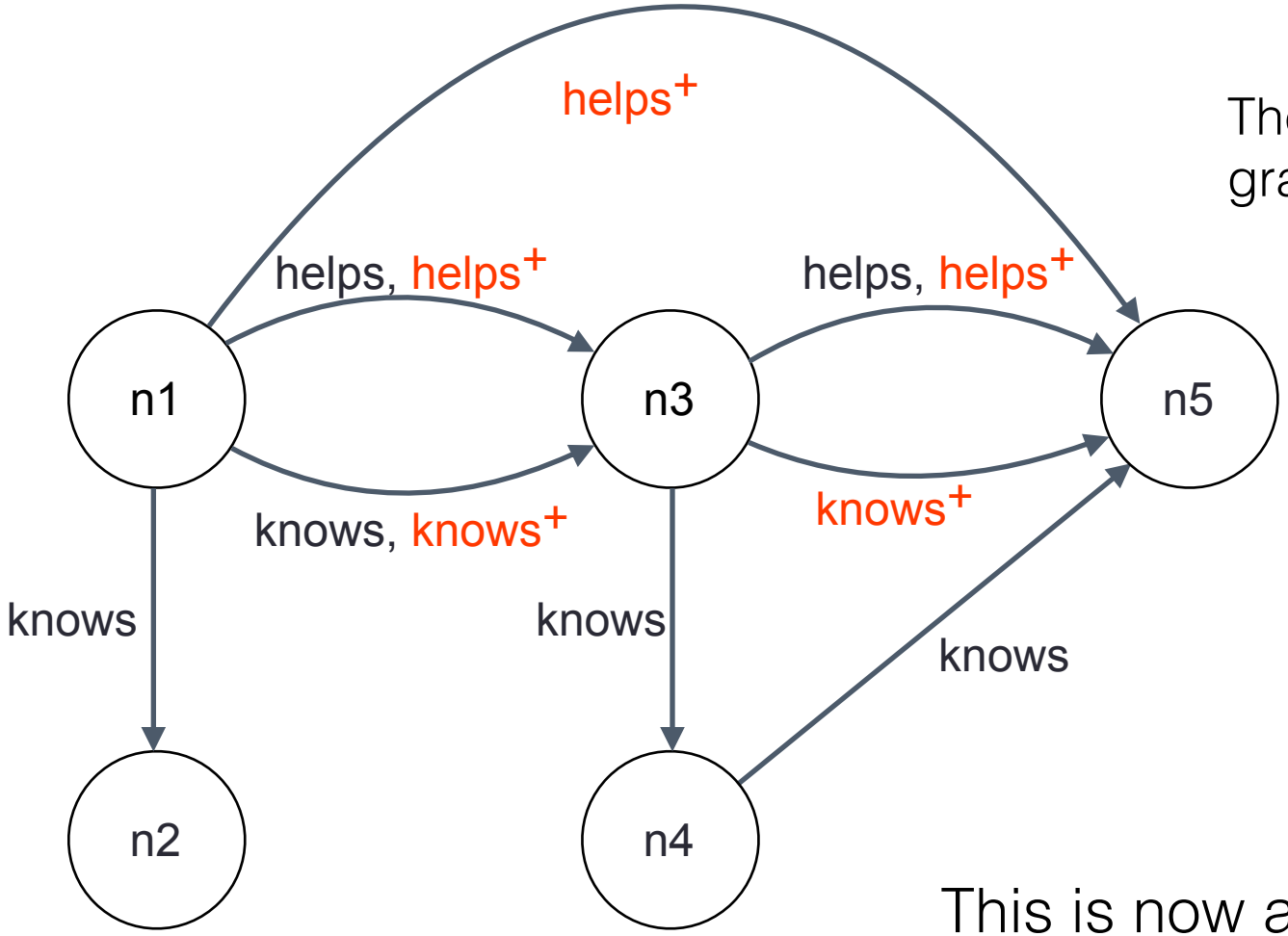
Membership:
- evaluate all NPQs first (in ptime), add them as labels
- then the CNPQ is just a CQ over extended voacbulary

Hardness already for CQs

Evaluation example



$$Q(x, y) \;=\; \left(\text{knows}^{+}(x, y) \wedge \text{helps}^{+}(x, y)\right)$$



The augmented graph is quadratic …

This is now a CQ over helps, knows, helps+, knows+

$$Q(x, y) \;=\; \left(\text{knows}^{+}(x, y) \wedge \text{helps}^{+}(x, y)\right)$$

Data Complexity of Evaluation Problem

(checking if a tuple is in the answer of a fixed navigational graph pattern)

For CQs in LogSpace (and even lower)

NLogSpace-complete for CNPQs

Membership:
- compose algorithm for CQs + NPQs

Hardness already for NPQs (even RPQs)

Recap

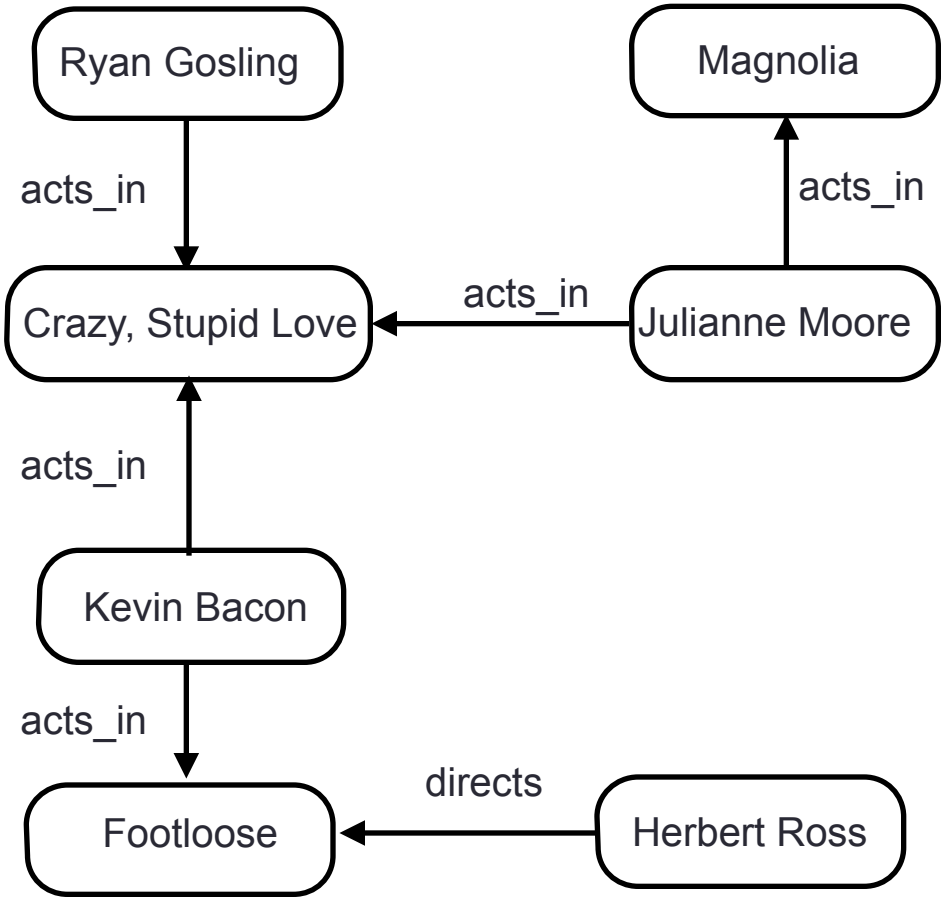
Recap

Most used navigational primitives:

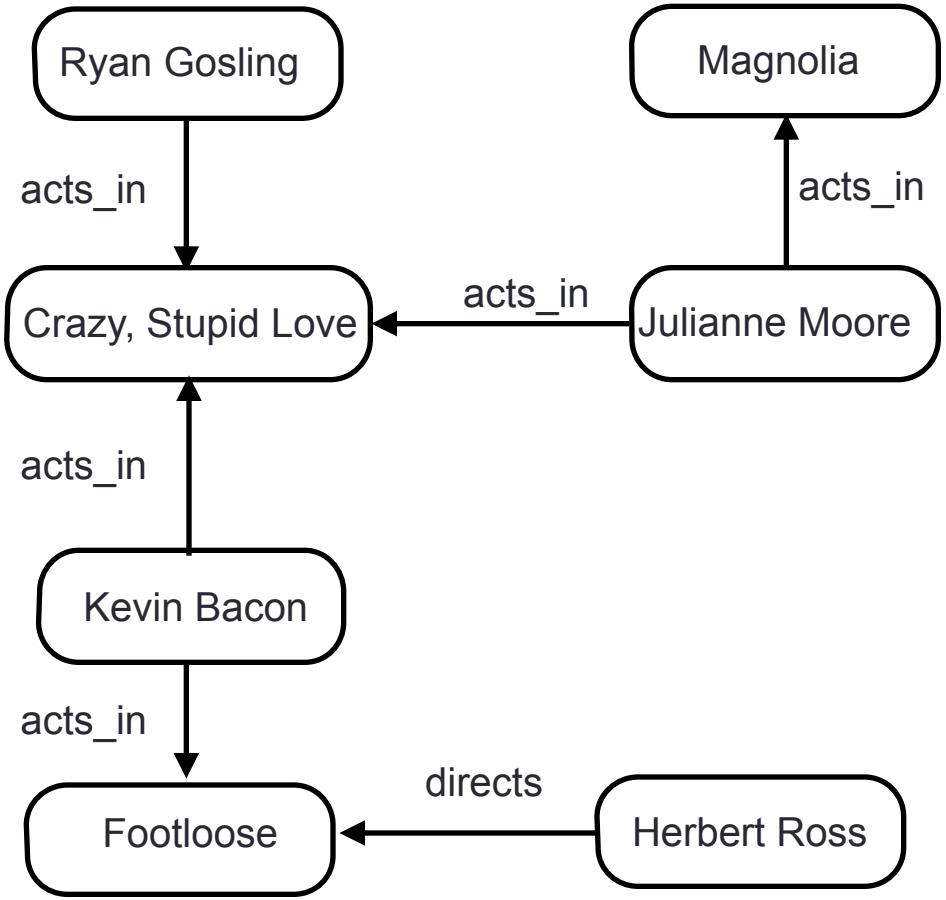
2RPQs: regular expressions over labels and their inverses

NPQs: adds [] operator

More examples of path queries



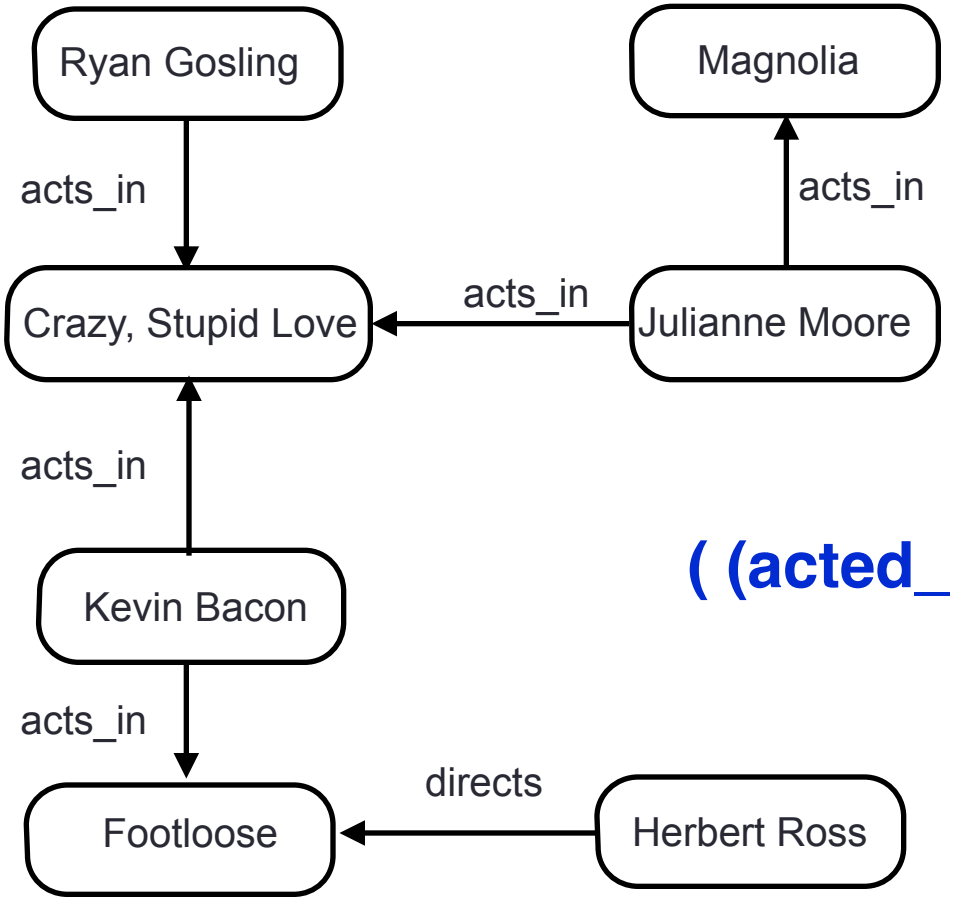
More examples of path queries



(acted_in | acted_in⁻)⁺

Look for all co-actors:
Bacon number query
(a 2RPQ)

More examples of path queries



((acted_in ∣ acted_in⁻)[directs]) +

Look for all co-actors,
but each actor is also a director
(an NPQ)

More examples of navigational patterns

More examples of navigational patterns

$$Q(x) \;=\; \exists p \exists m \big(\mathsf{knows}^*(x, p) \wedge \mathsf{acted\_in}(p, m) \wedge \mathsf{directs}(p, m) \big)$$

Computes all x that knows a person p
that acted and directed a movie m

More examples of navigational patterns

$$Q(x) \;=\; \exists a \exists r \big((\text{acted\_in} | \text{acted\_in}^{-})^{+}(x, a) \wedge \text{coauthor}^{+}(x, r) \big)$$

Computes all x connected to an actor a (via acting)
and researcher r (via coauthor)

Erdos - Bacon number query

Expressive power

adds $\wedge, \exists$ CRPQ	C2RPQ	CNPQ
RPQ	2RPQ adds a^-	NPQ adds $[\ ]$

Expressive power

Implemented in SPARQL (RDF databases)

adds $\wedge, \exists$ CRPQ C2RPQ CNPQ

RPQ 2RPQ NPQ

adds a^-

adds $[\,]$

Expressive power

(Partially) Implemented in CYPHER (Neo4j graph database)

adds $\wedge, \exists$ CRPQ C2RPQ CNPQ

RPQ 2RPQ NPQ

adds a^-

adds $[\,]$

Outline

Path Queries

Navigation and Patterns

Conjunctions of Path Queries

Beyond Patterns

What can’t navigational patterns do

Two issues:

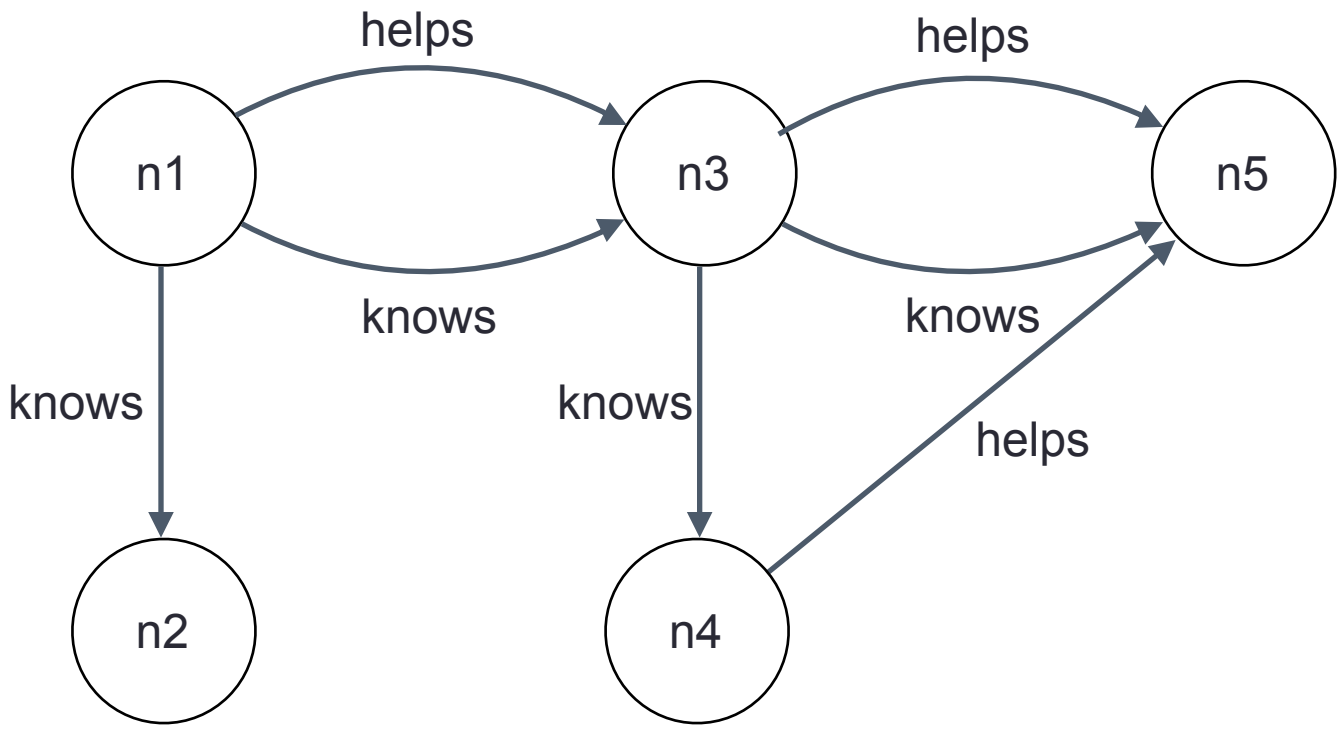
- Sometimes we need more expressive power
- Navigational patterns are not algebraically closed

People have recognised the need
for more expressive graph queries

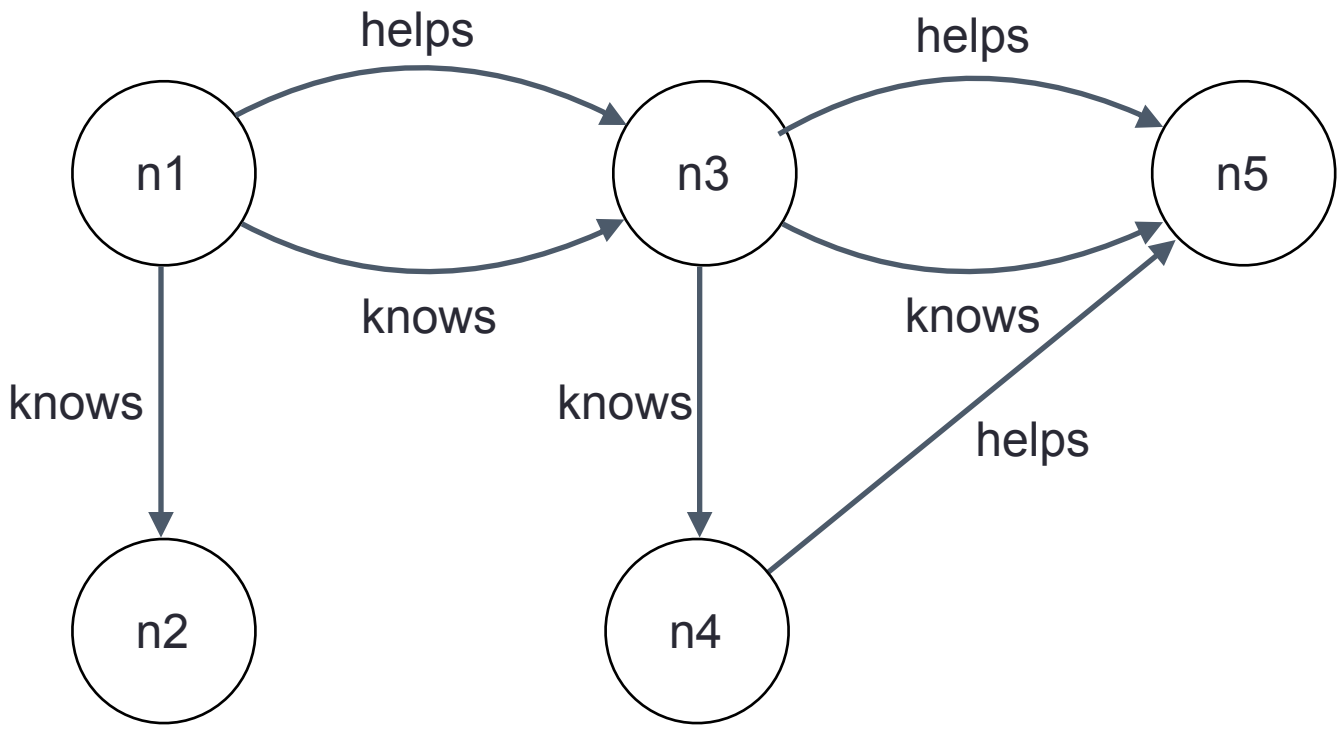
People have recognised the need
for more expressive graph queries

more interplay
between pattern matching and navigation

Beyond patterns

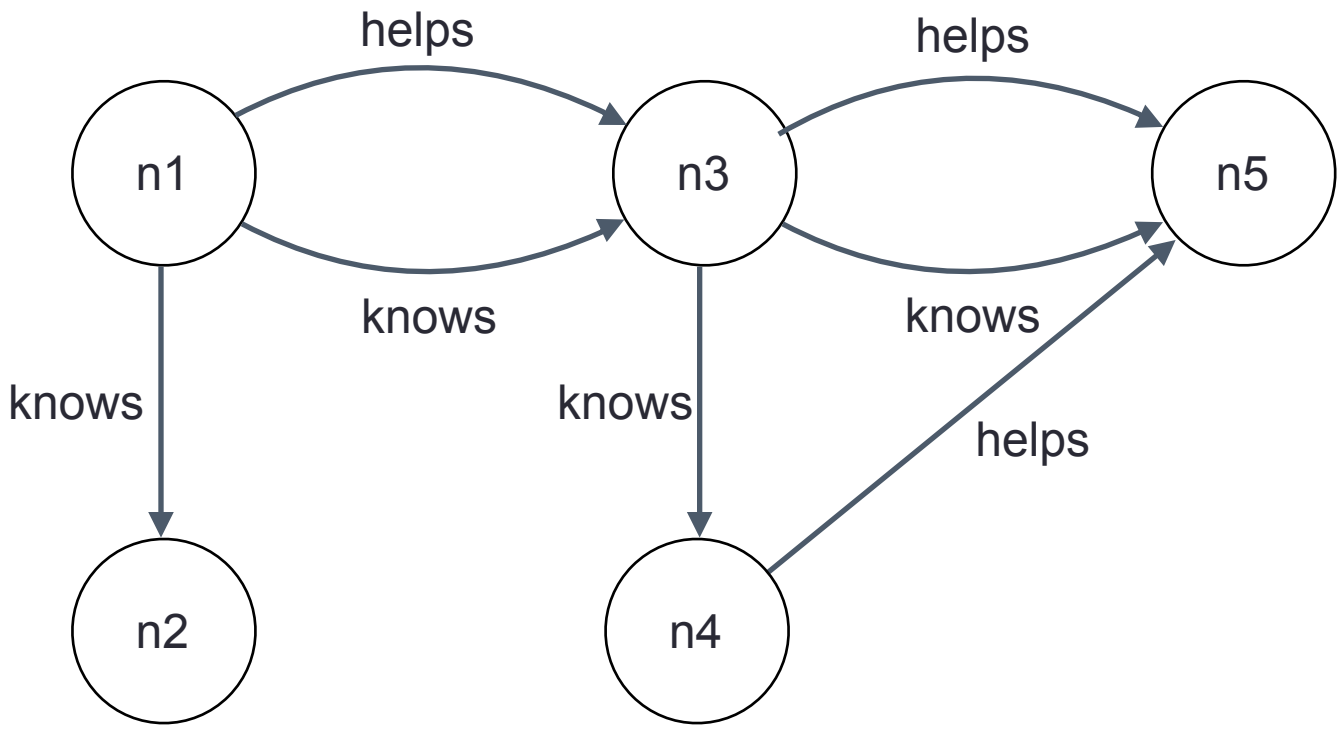


Beyond patterns



Say u is a friend of v if
u knows v and u helps v

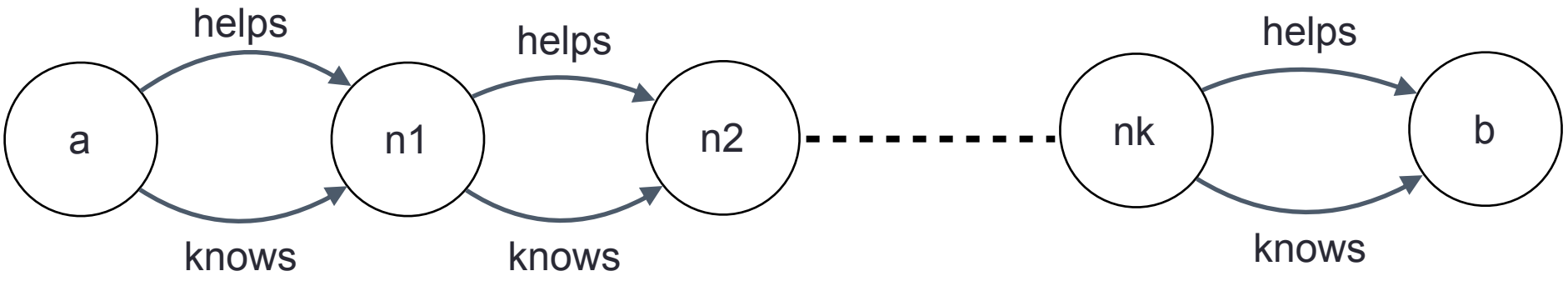
Beyond patterns



Say u is a friend of v if
u knows v and u helps v

Find all nodes connected by a chain of friends

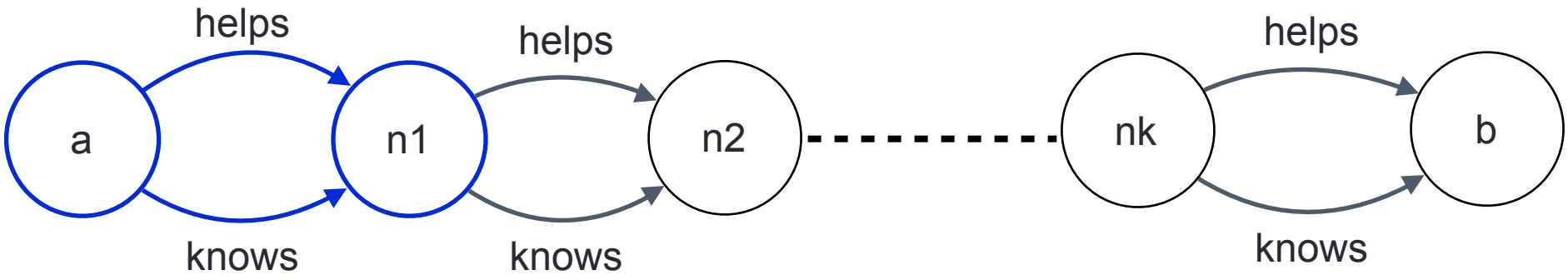
Beyond patterns

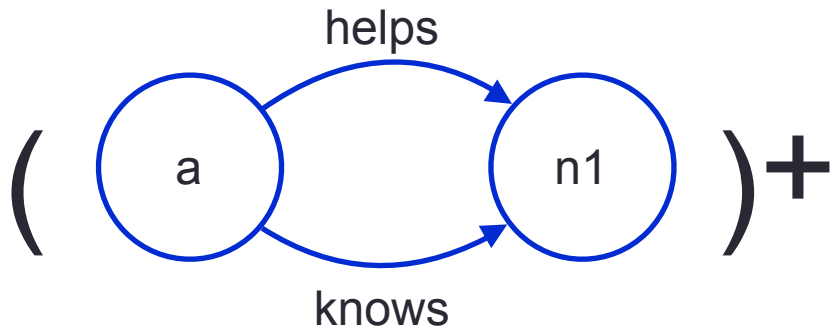


Say u is a friend of v if
u knows v and u helps v

Find all nodes connected by a chain of friends

Beyond patterns





Say u is a friend of v if
u knows v and u helps v

Find all nodes connected by a chain of friends

Navigational Patterns are not algebraically closed

A language is algebraically closed if one stays in the same language when applying operators

Algebraically closed languages:

- Relational Algebra
- Conjunctive Queries
- Path Queries

Navigational Patterns are not algebraically closed

A language is algebraically closed if one stays in the same language when applying operators

CRPQs, C2RPQs and CNPQs are not algebraically closed:

What is the kleene star of a pattern?

One problem of queries not algebraically closed is that implementations end up with two engines:

- One for pattern matching (the main DB engine)
- One for path queries

One problem of queries not algebraically closed is that implementations end up with two engines:

- One for pattern matching (the main DB engine)
- One for path queries

Query planning and query optimisation is harder!

Even though these queries are implemented,
their performance is not optimal.

In the lasts years we have seen renewed efforts
to find expressive graph languages
that are algebraically closed

The majority are based on Datalog