Querying Graph Data
Where Are We? Where Should We Go?

Leonid Libkin
RelationalAI, IRIF,
University of Edinburgh

Filip Murlak
University of Warsaw

Wim Martens
University of Bayreuth
RelationalAI

Domagoj Vrgoč
PUC Chile

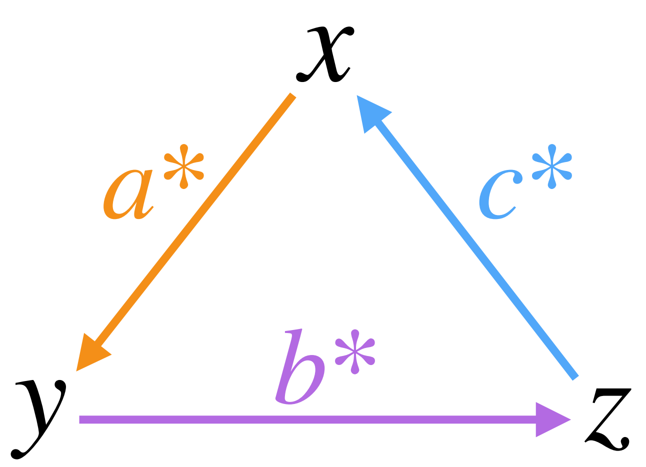
Liat Peterfreund
Hebrew University

Gems Talk
PODS 2025

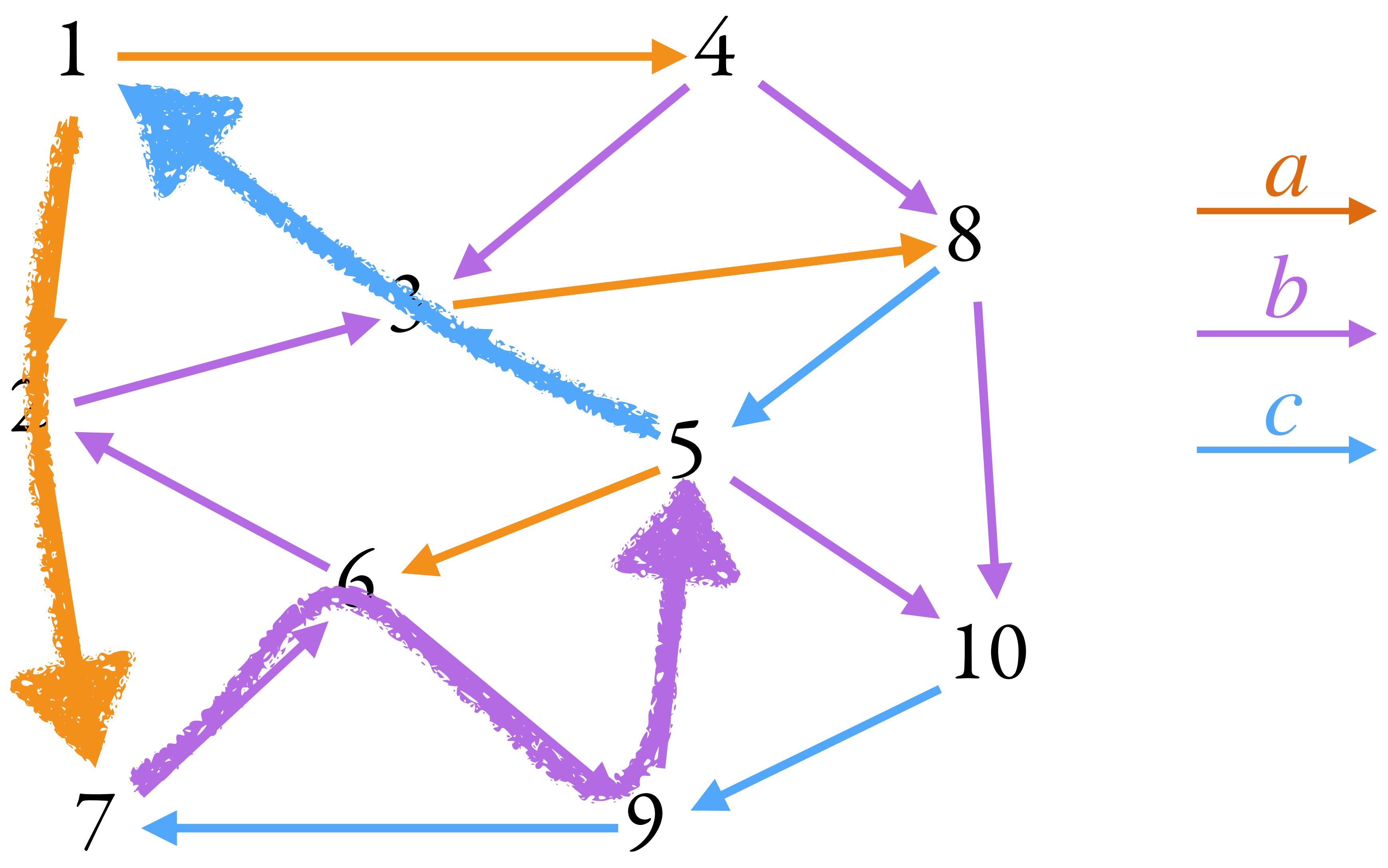
Paper:

Conjunctive Regular Path Queries (CRPQ)

Query



Graph Database



Paper:

Conjunctive Regular Path Queries 2.0

What Do Cypher, SQL/PGQ, GQL Patterns add to CRPQs?

(a) handling of nodes and edges

(b) path and list variables ⇝ we'll get to this

(c) path modes ⇝ simple, trail,...

(d) data filters ⇝ data value comparisons

The "Four Features"

How do we cleanly define these? so that we can study them

How do we cleanly design these in a real-world language? so that they can be used

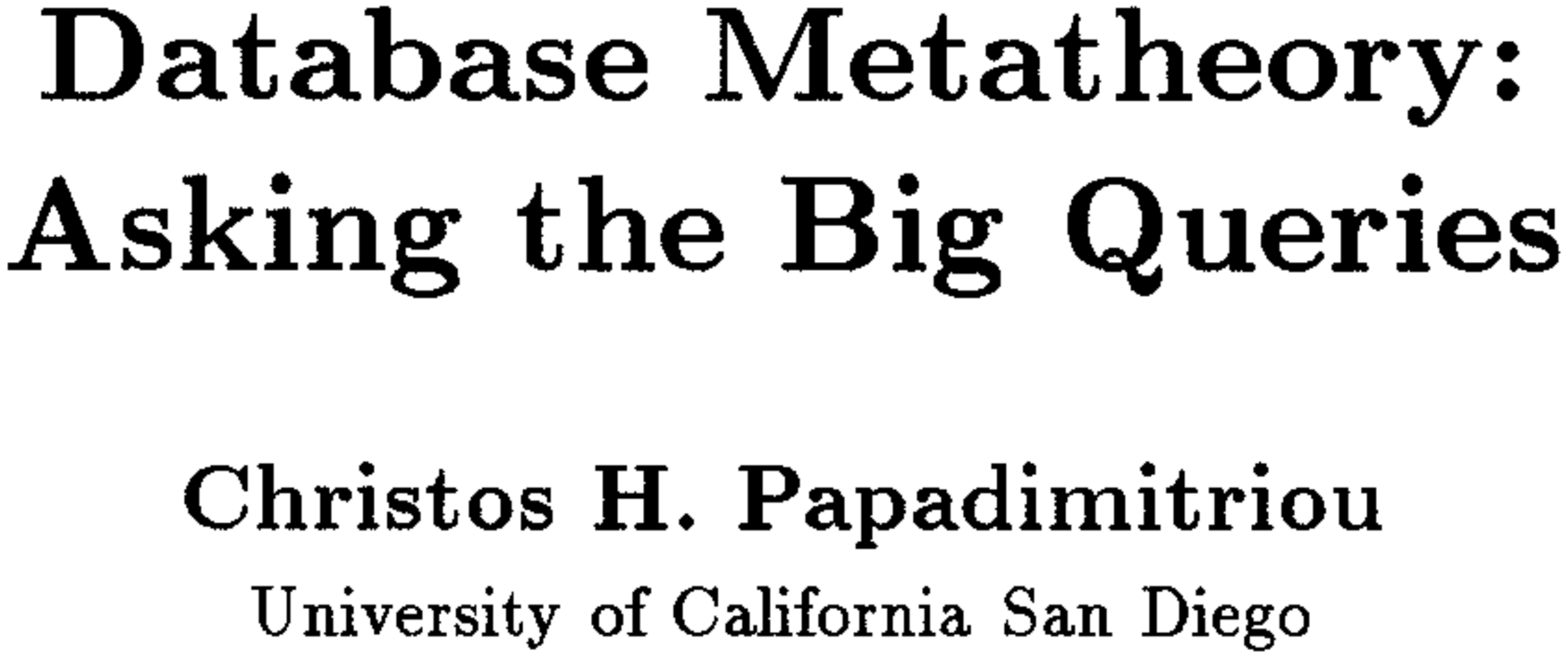
Query Language Design! Nice!

(c) [Bagan, Bonifati, Groz PODS'13] [M., Trautner ICDT'18] [M., Niewerth, Trautner STACS'20] [M., Popp PODS'22]

(d) [Neven, Schwentick, Vianu TOCL'04] [Bojanczyk et al. PODS'06, LICS'06] [Libkin, M., Vrgoc ICDT'13]

The Broader Context

Big Questions

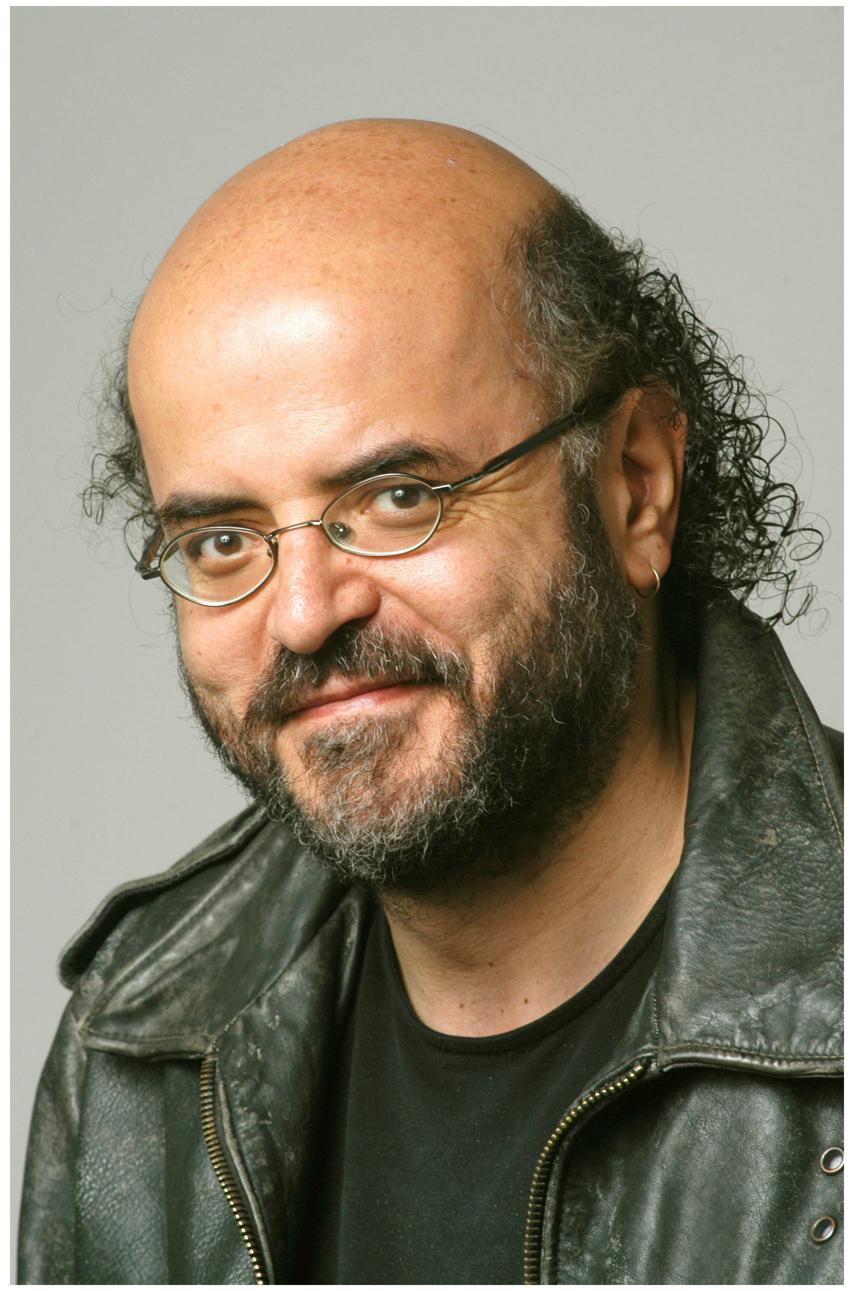


Database Metatheory: Asking the Big Queries

Christos H. Papadimitriou

University of California San Diego

PODS 1995



Big Questions

Quiz

What is the most important open research question for database theory?

My humble opinion

Can we make set semantics perform in practice?

Why is this question important?

Database Research Landscape

Theory

Sets

divide

why?

Systems

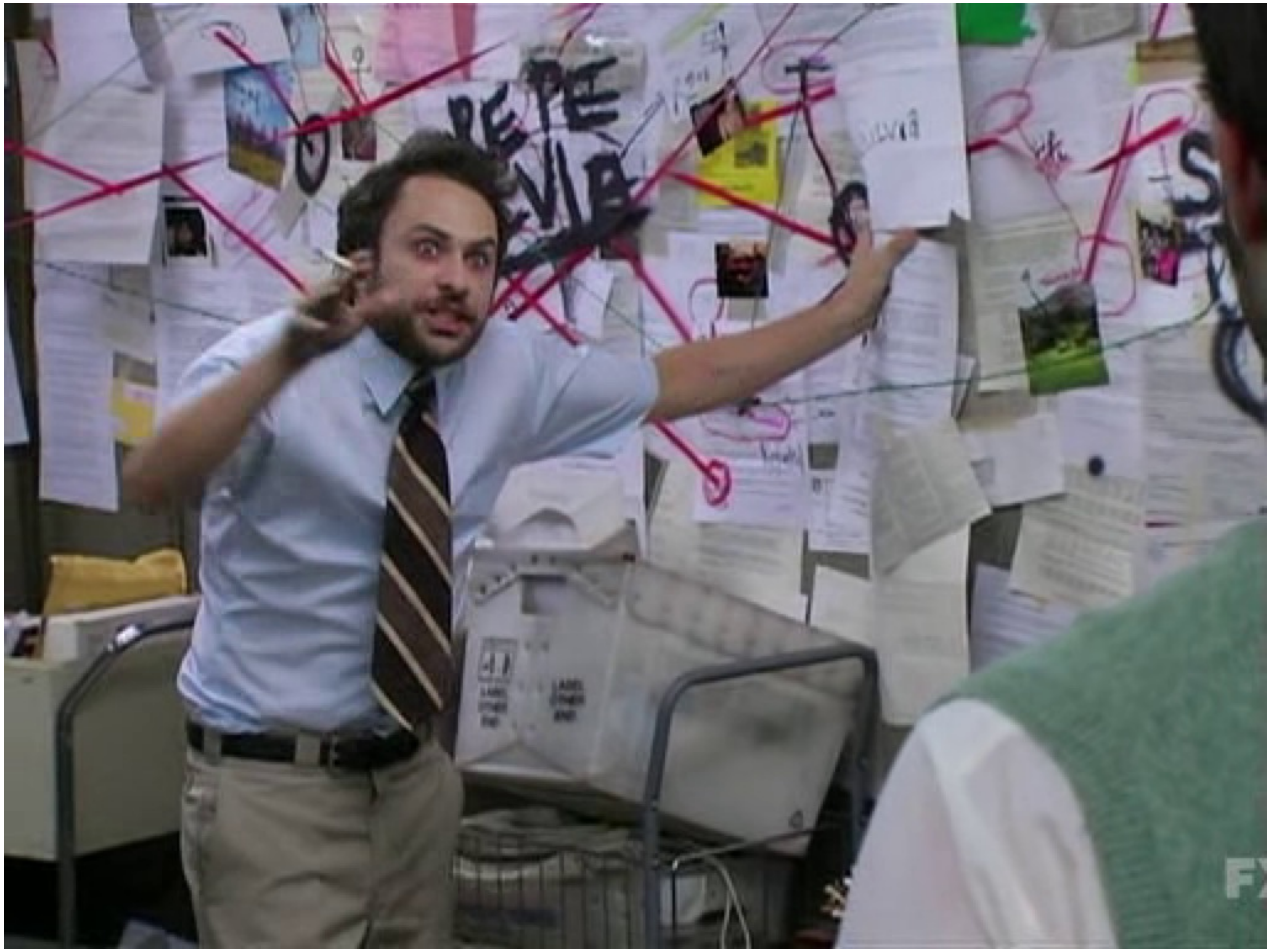
Bags
(Multisets)

Con sets: more costly union & project
⇝ more data structure overhead
Pro sets: more room for query optimization
Pro sets: …

(https://www.geologyin.com/2024/05/rift-valleys.html)

Why Am I Talking About Graphs & Sets & Bags?

...because everything is connected



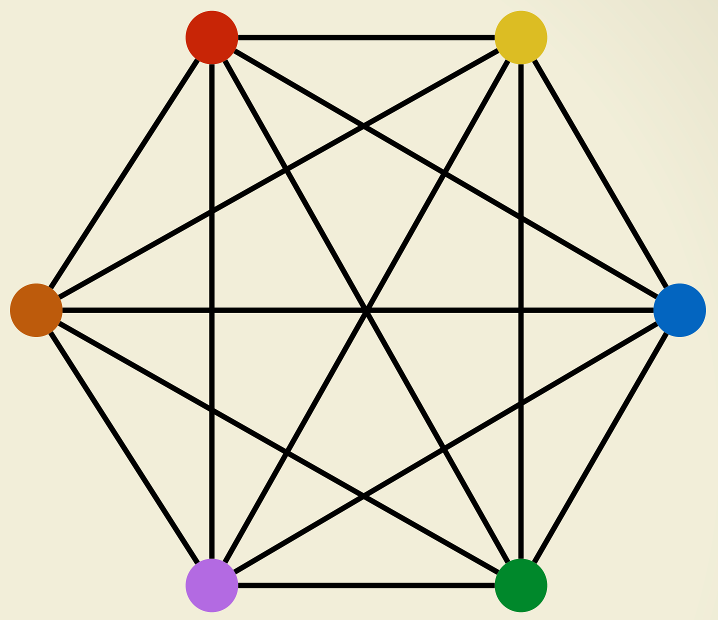
"Everything is connected!"

(https://imgflip.com/memegenerator/268702234/Charliethinkingits-all-connectedconnect-the-dot)

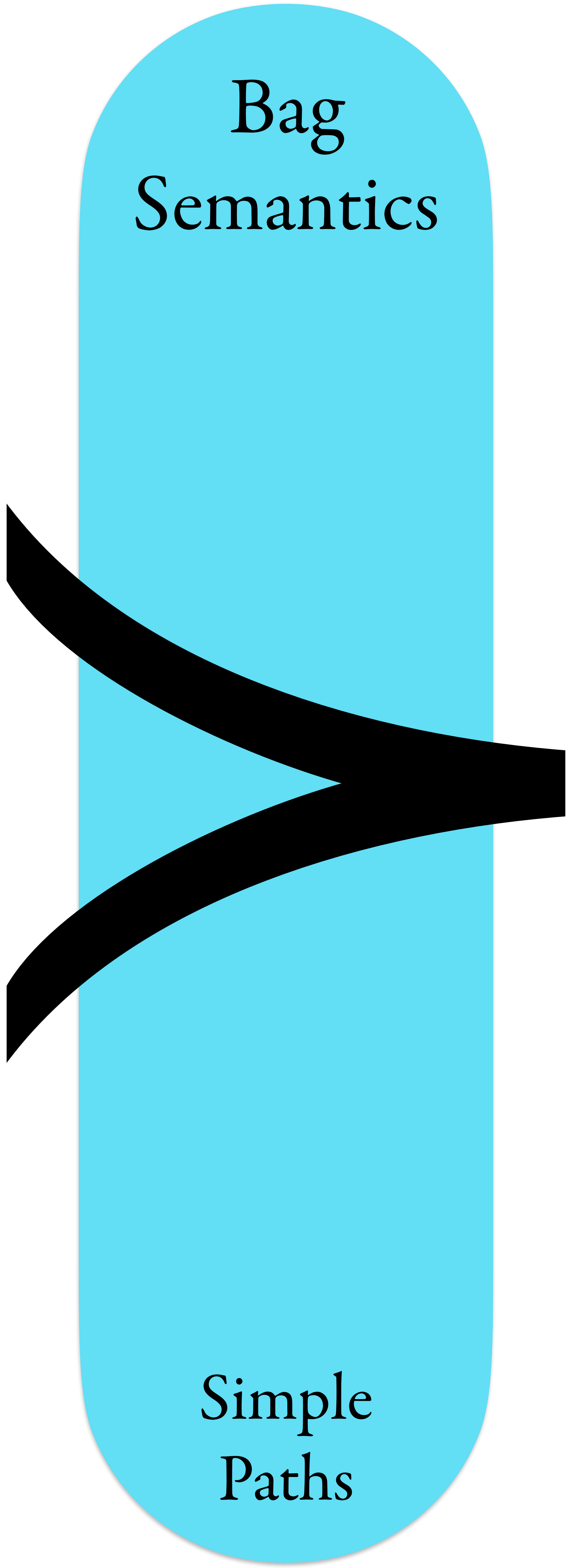
Bags & Recursion: Boom!

Query $((((a^*)^*)^*)^*$

Data



6-clique
every edge labeled *a*



Bag
Semantics

Simple
Paths

Every answer is returned 10^{269} times

Smoking gun
for bags?

[Arenas, Conca, Pérez WWW'12]

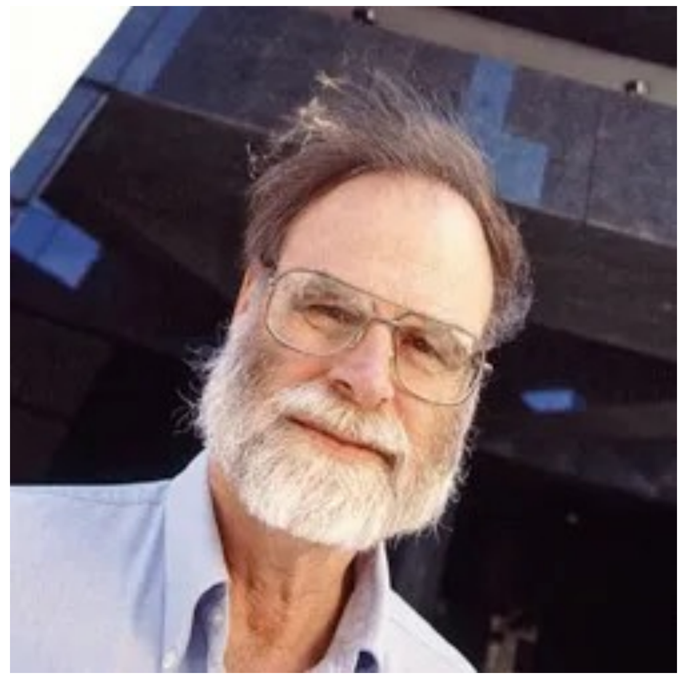
What Are We Doing in (Query) Language Design?

Language Design: The Goal?

Useful to give us direction

Automatic Programming

- Allowing human programmers to code at a significantly higher level
- Focusing on the problem that needs solving, not on its “administrative aspects”



Jim Gray

We, the DB community (theory & systems) are **great** at this

- Declarative query languages
- Automatic optimization, automatic out-of-core computation, etc.

The only thing is that we’re doing it just for DB queries

Why not think bigger?

Can we target more general-purpose programming?

If so, then our languages better be well-designed!

We need the right principles!

Molham Aref

Thursday 17:00
SIGMOD Industry 6
Be there
I kid you not

Humanity

Will we survive?

Science

Abundant clean energy

Computer Science

Automatic Programming

Is $P \neq NP$?

Databases

Efficiency, Efficiency, Efficiency

Proper data independence

Database Theory

Are sets the right model?

CQ bag containment

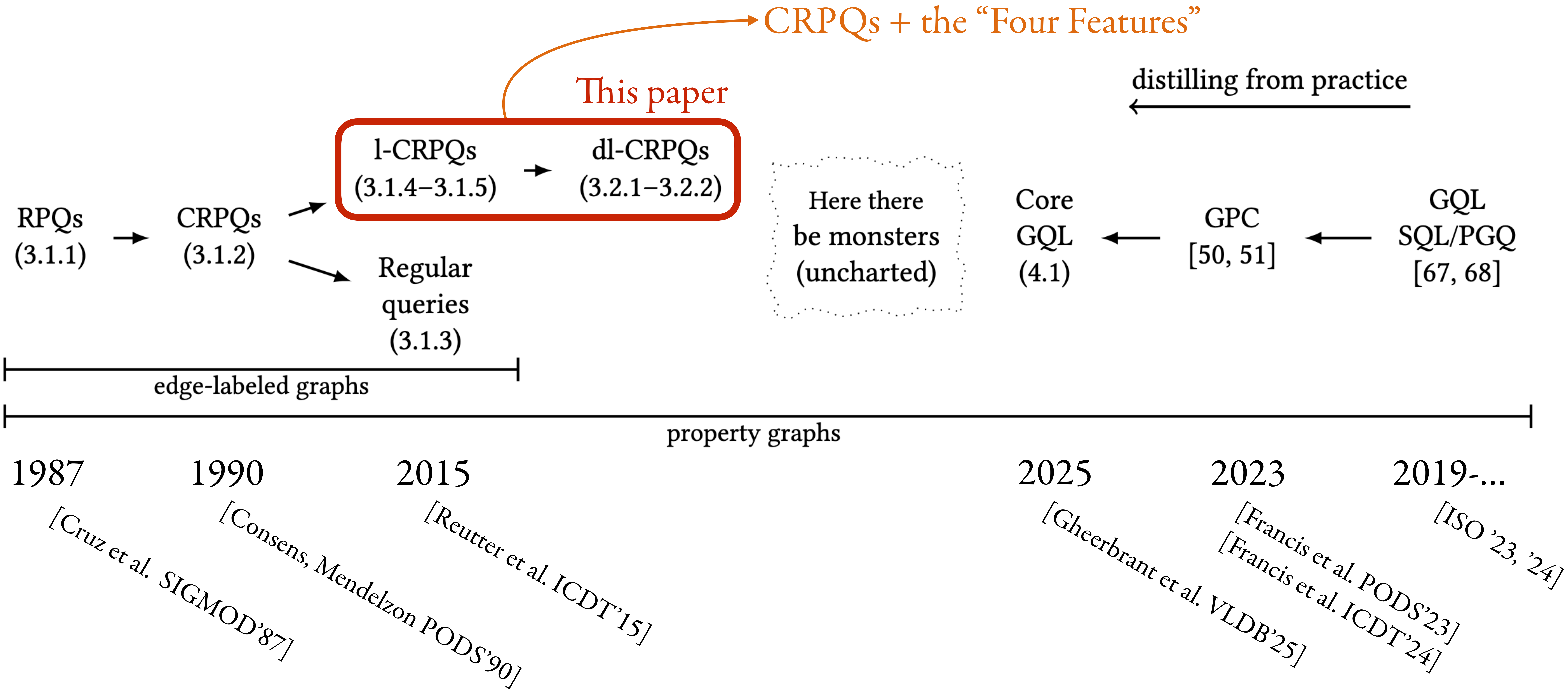
Graph Database Theory

We go here again

Where We Are

We wanted to do query language design in graph pattern matching

Graph Pattern Matching Landscape



CRPQs + the "Four Features"

This paper

distilling from practice

l-CRPQs (3.1.4–3.1.5) → dl-CRPQs (3.2.1–3.2.2)

RPQs (3.1.1) → CRPQs (3.1.2) → Regular queries (3.1.3)

Here there be monsters (uncharted)

Core GQL (4.1) ← GPC [50, 51] ← GQL SQL/PGQ [67, 68]

edge-labeled graphs

property graphs

1987 [Cruz et al. SIGMOD'87]

1990 [Consens, Mendelzon PODS'90]

2015 [Reutter et al. ICDT'15]

2025 [Gheerbrant et al. VLDB'25]

2023 [Francis et al. PODS'23] [Francis et al. ICDT'24]

2019-... [ISO '23, '24]

Why Theory Is Needed

Example 1

```
(x) ( ()-[ :Follows]->() ){2} (y)
```

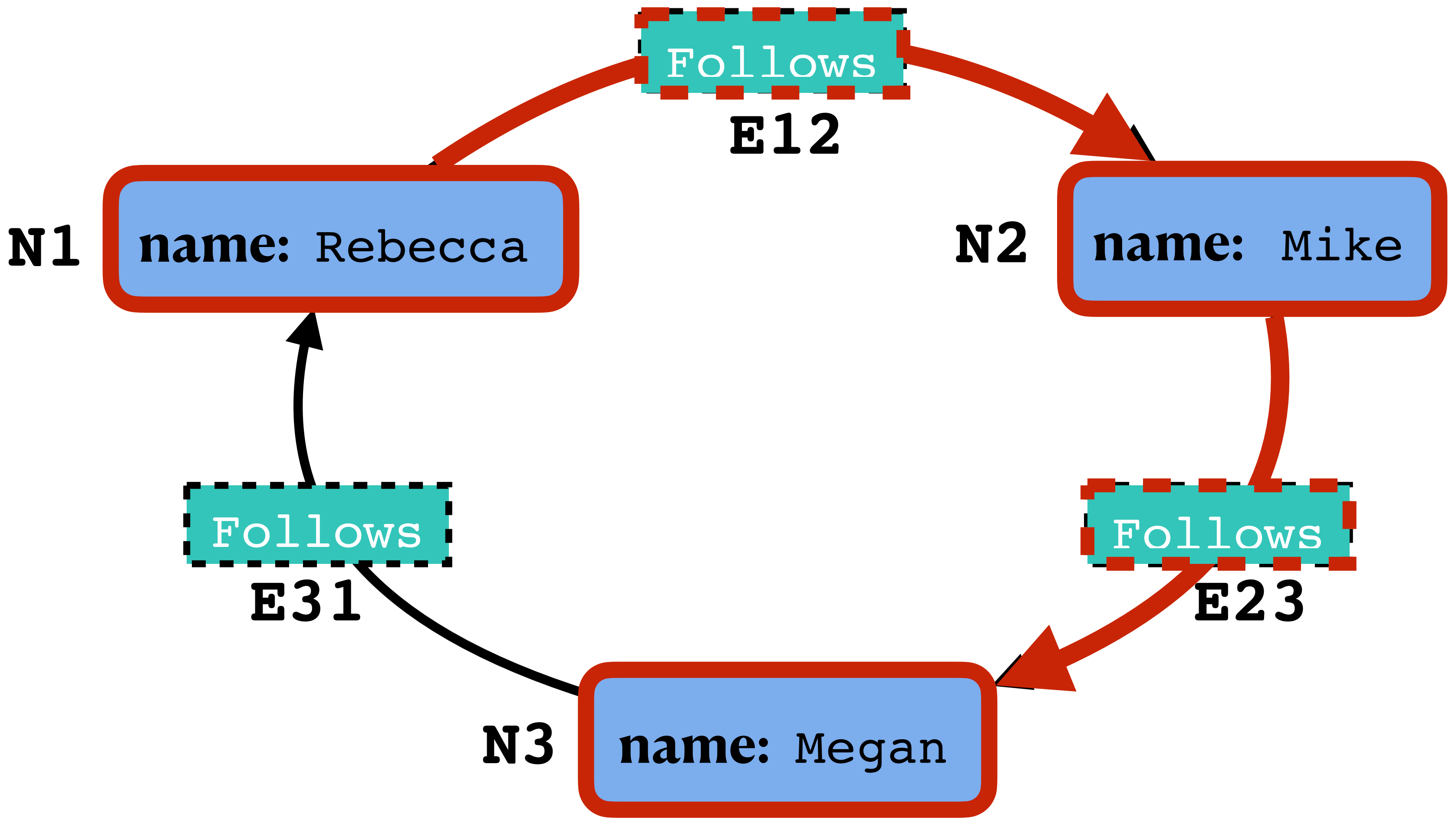
(…) node

[…] edge

→ paths from x to y

→ edge labeled “Follows”

→ repeat the edge 2 times



Output:

x	y
N1	N3
N2	N1
N3	N2

Equivalent to

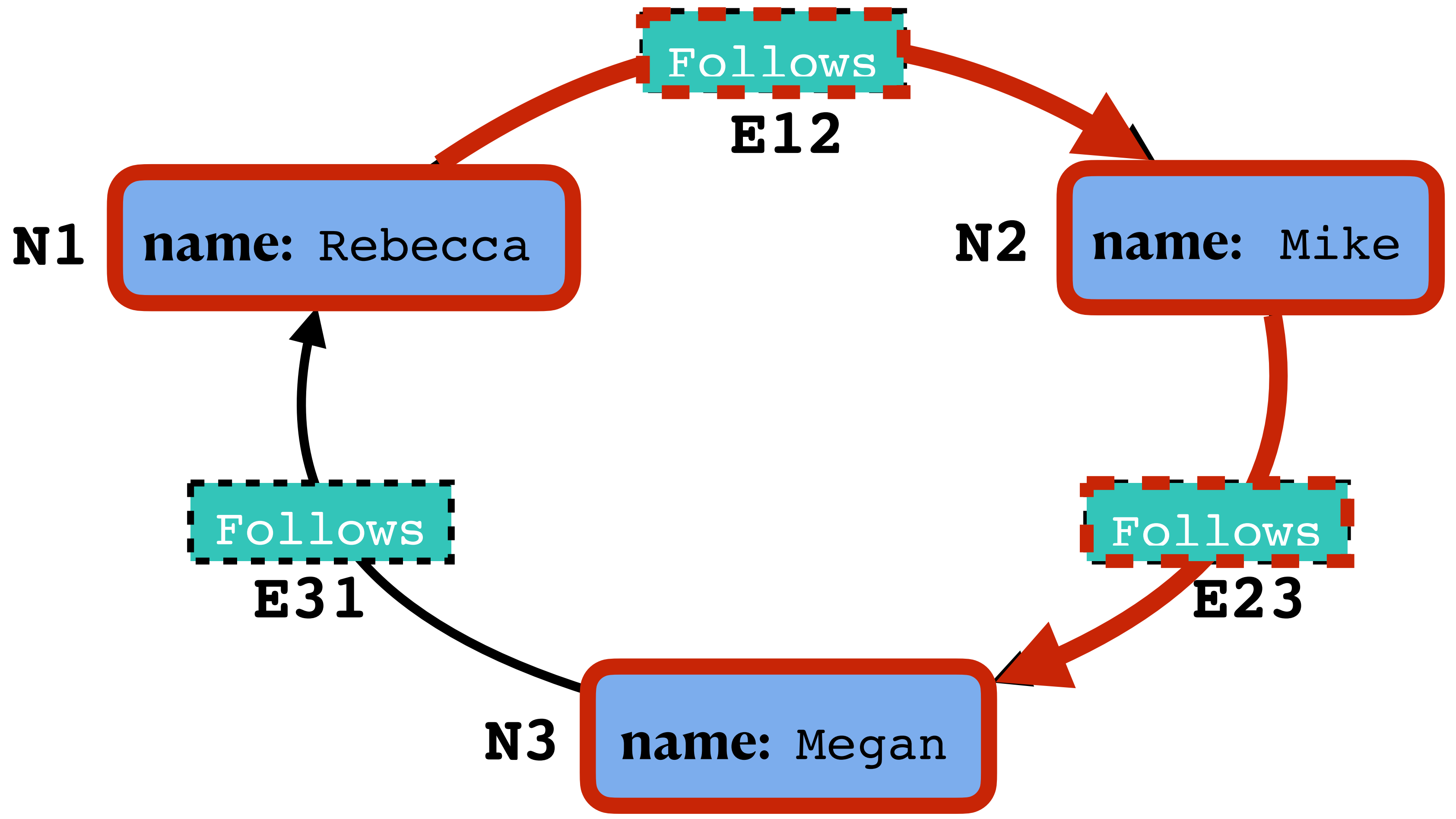
```
(x) ( ()-[:Follows]->() ) ( ()-[:Follows]->() ) (y)
(x) ( ()-[:Follows]->()-[:Follows]->() ) (y)
```

Why Theory Is Needed

Example 1

```
(x) ( ()-[z:Follows]->() ){2} (y)
```

- paths from x to y
- edge labeled “Follows”, in variable z
- repeat the edge 2 times



Output:

x	y	z
N1	N3	[E12, E23]
N2	N1	[E23, E31]
N3	N2	[E31, E12]

Not equivalent to any of

```
(x) ( ()-[z:Follows]->() ) ( ()-[z:Follows]->() ) (y)
(x) ( ()-[z:Follows]->()-[z:Follows]->() ) (y)
(x) ( ()-[z1:Follows]->()-[z2:Follows]->() ) (y)
```

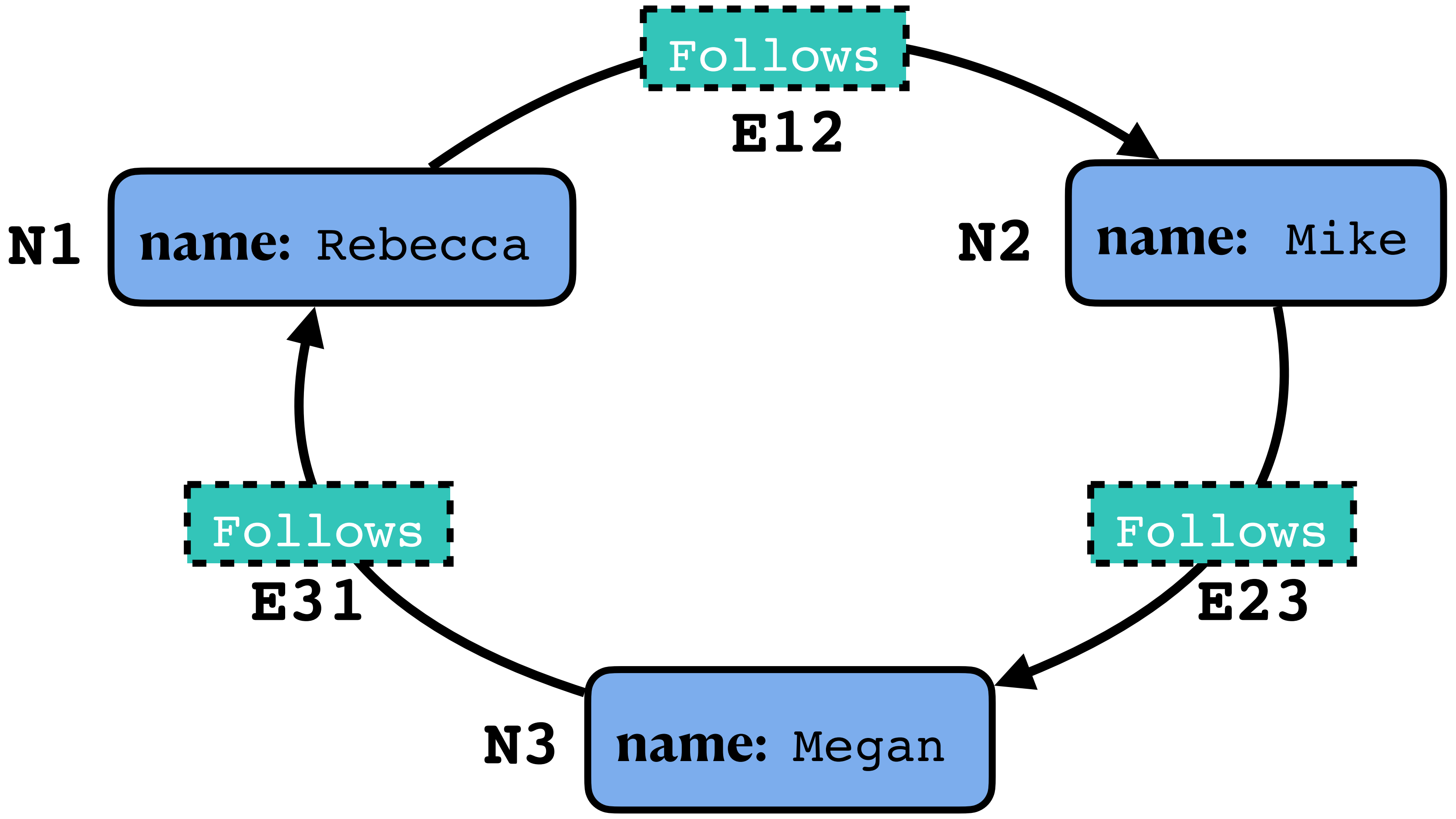
Why Theory Is Needed

Example 2

```
(x) ( ( )-[:Follows]->( )-[:Follows]->( ) )+ (y)
```

paths of even length

edges labeled “Follows”



Output:
(trails only)

x	y
N1	N3
N2	N1
N3	N2

Why Theory Is Needed

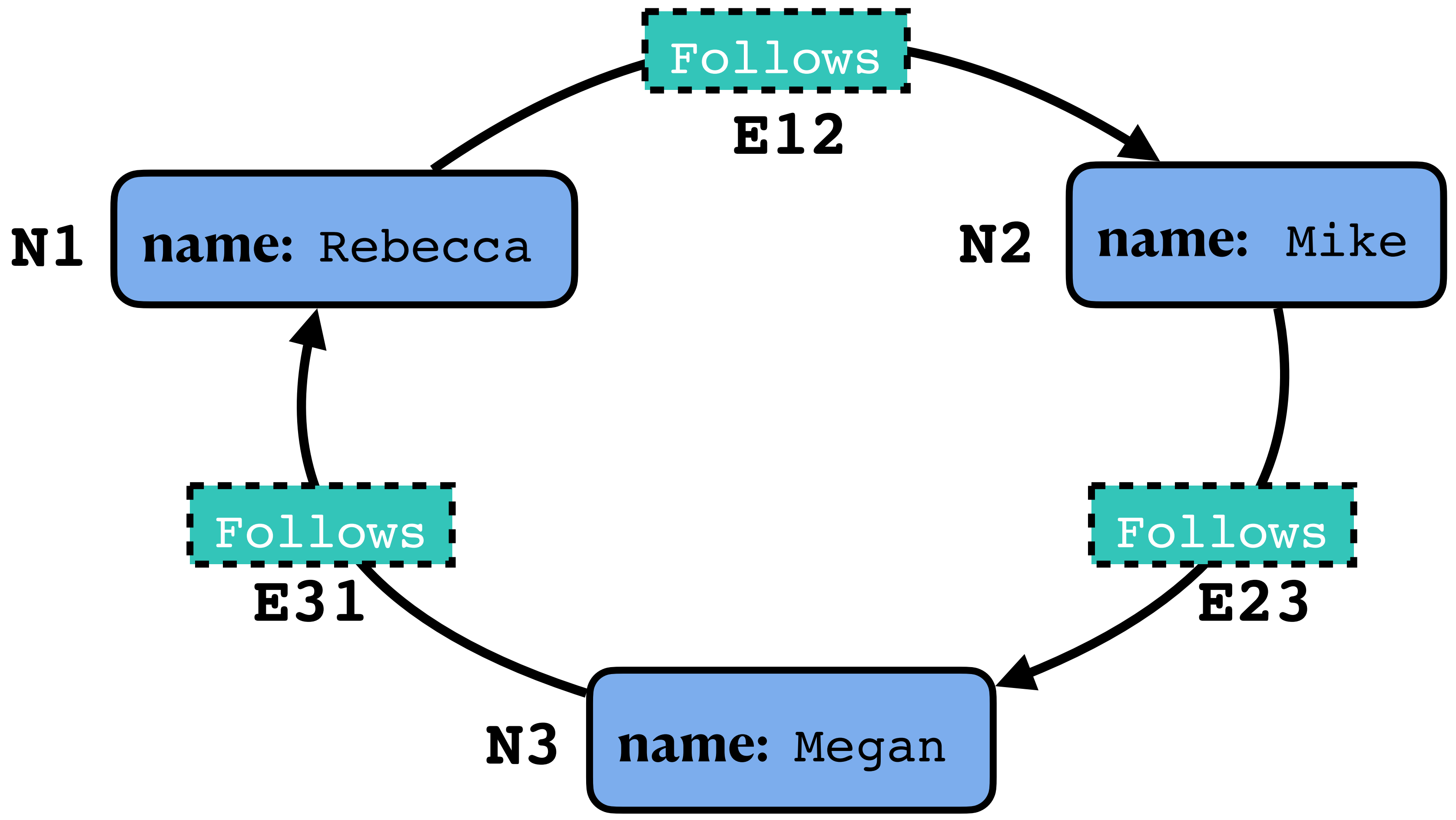
Example 2

```
(x) ( (z)-[:Follows]->(z)-[:Follows]->( ) )+ (y)
```

paths of even length

edges labeled “Follows”

using variable z



Output: Nothing!

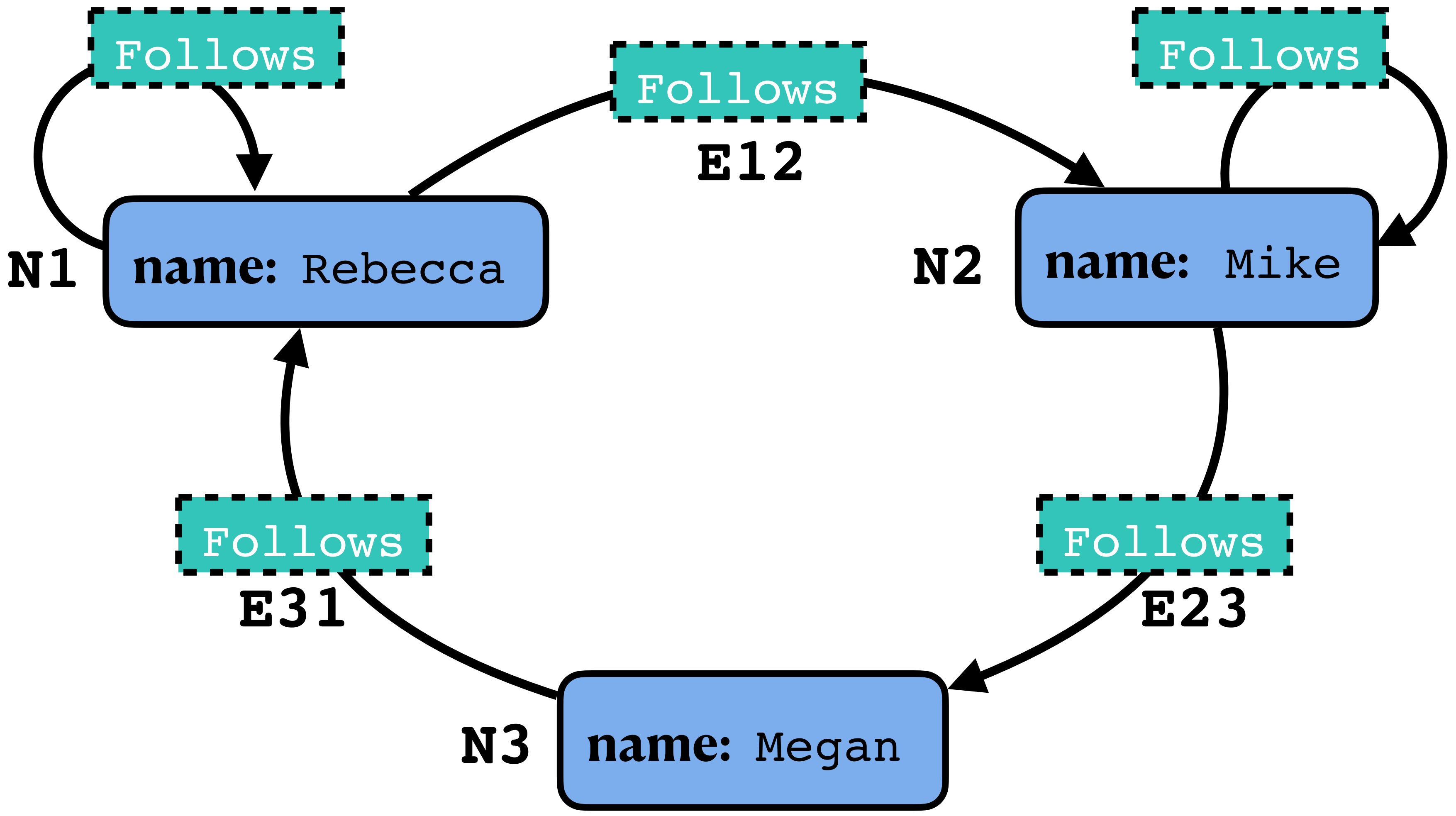
Why?

Why Theory Is Needed

Example 2

```
(x) ( (z)-[:Follows]->(z)-[:Follows]->( ) )+ (y)
```

- paths of even length
- edges labeled “Follows”
- using variable z



Output:

x	y	z
N1	N3	[N1, N2]

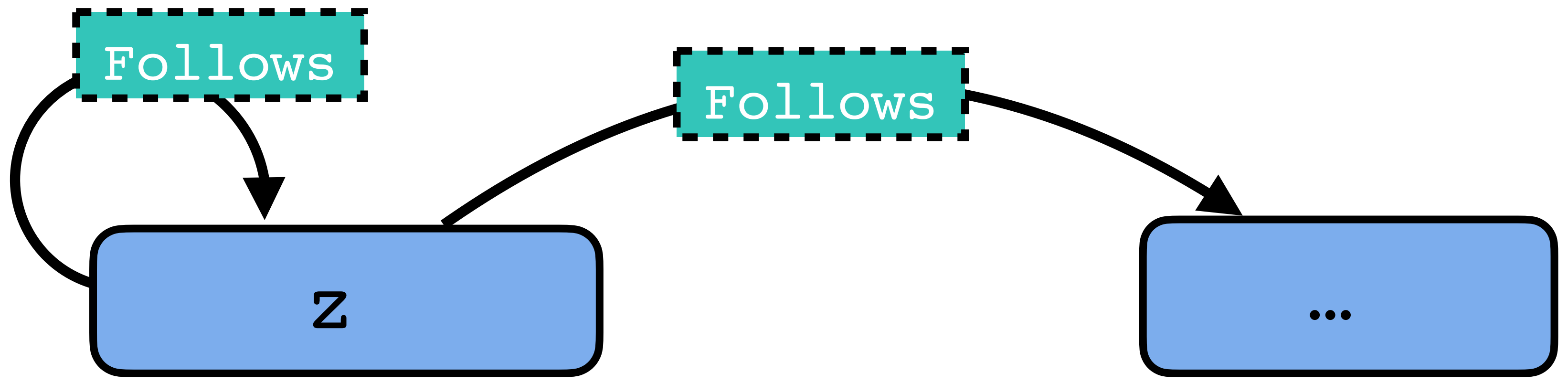
...and others

Again, why?

Why Theory Is Needed

Example 2

```
(z)-[:Follows]->(z)-[:Follows]->( )
```

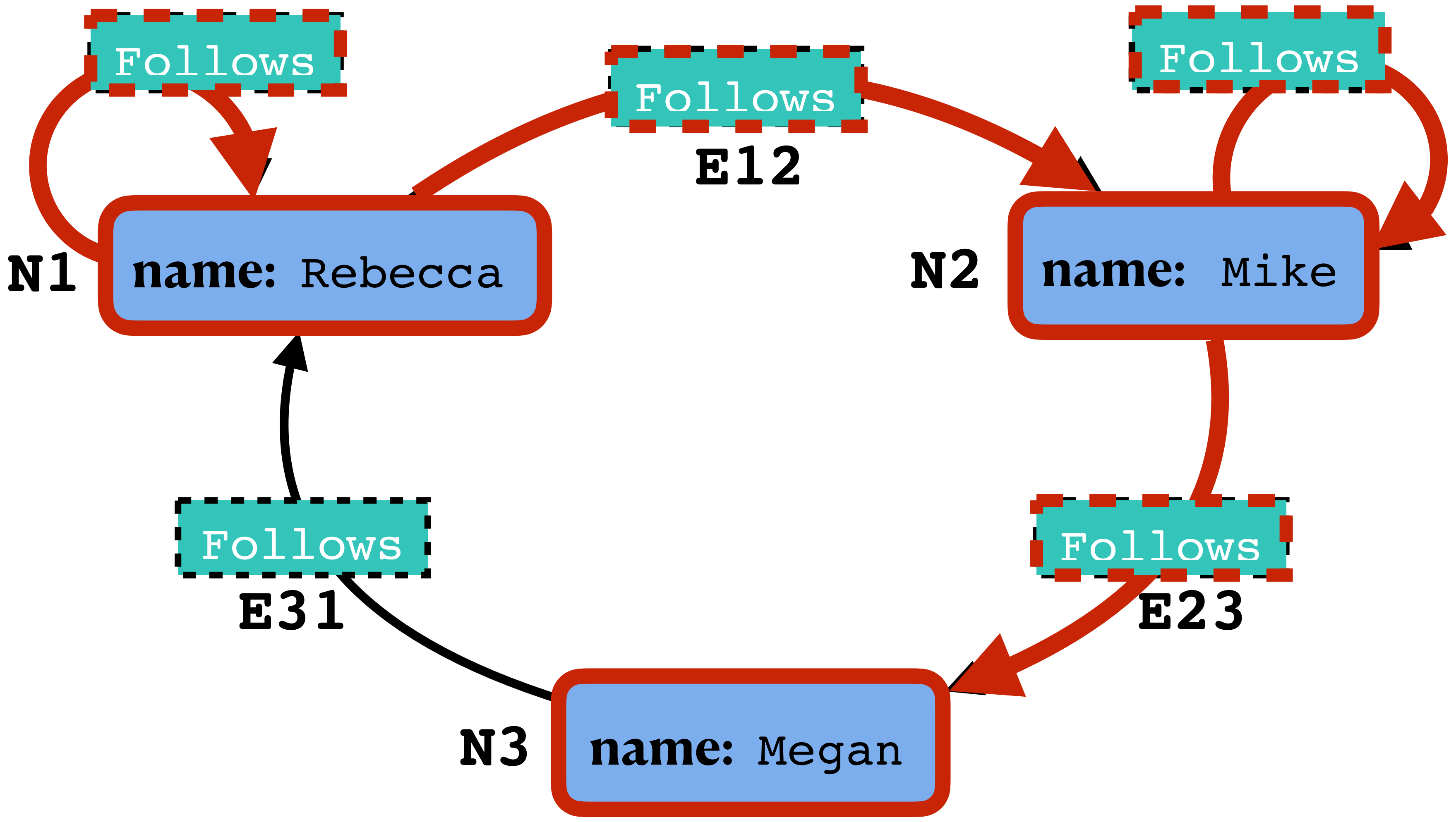


Syntax-driven design!

Why Theory Is Needed

Example 2

```
(x) ( (z)-[:Follows]->(z)-[:Follows]->( ) )+ (y)
```



Output:

x	y	z
N1	N3	[N1, N2]

Syntax-driven design!

Need to separate concerns? (Joins ⟷ Lists)

If you think about it as CRPQs, then

- the RPQs are about matching paths
- the CRPQ vars are about joining

Why Theory Is Needed

Example 3

```
p = (x) ( (u)-[ ]->(v) WHERE u.date < v.date)* (y)
```

→ Well done!
Cypher
SQL/PGQ
GQL

Paths from x to y, where dates increase on nodes

Intermezzo: in SQL, you'd need to start with...

```
WITH good_edge(S,T) AS
  SELECT Source.N_ID AS S, Target.N_ID AS T
  FROM Source, Target, Dates D1, Dates D2
  WHERE Source.E_ID = Target.E_ID
    AND D1.N_ID=Source.N_ID AND D2.N_ID = Target.N_ID
    AND D1.date < D2.date
RECURSIVE path_n(S, T) AS
  SELECT * FROM good_edge
  UNION
  SELECT good_edge.S, path_n.T
  FROM good_edge, path_n
  WHERE good_edge.T=path_n.S
SELECT * FROM path_n
```

Why Theory Is Needed

Example 3

```
p = (x) ( (u)-[ ]->(v) WHERE u.date < v.date)* (y)
```

Paths from x to y, where dates increase on nodes

How do you do paths from x to y, where dates increase on edges?
Umm....

```
p = (x) (  (u) -[ ]->(v)      WHERE u.date < v.date)* (y)
```

increasing on nodes ✓

```
p = (x) ( -[u]->( ) -[v]->    WHERE u.date < v.date)* (y)
```

not increasing on edges ✗

```
p = (x) ( -[u]->( ) -[v]->() WHERE u.date < v.date)* (y)
```

not increasing on edges ✗

⇝ match 1 3 2 4

It Looks Like There's Work To Be Done Here

Wait, Weren’t You People on ISO?

Wait, Weren’t You People on ISO?

You don’t have total control

Apple is like a ship,
with a hole in the bottom,
leaking water
and my job is
to get the ship
pointed
in the right direction

—Steve Jobs

Shout out to

Nadime Francis

Victor Marsault

Made sense of

- constant stream of proposals
- each with dozens of pages
- in standardese

Paper:

What Now ?

Where to go ?

This is the starting point of the paper — we've only just gone through the intro

We're going to propose something. Let's first argue why we are proposing it.

Research Agenda

Query Languages in Theory

Sets

Declarative

Automata

Prove advantages of our principles!

Make them work in practice!

Systems

Bags

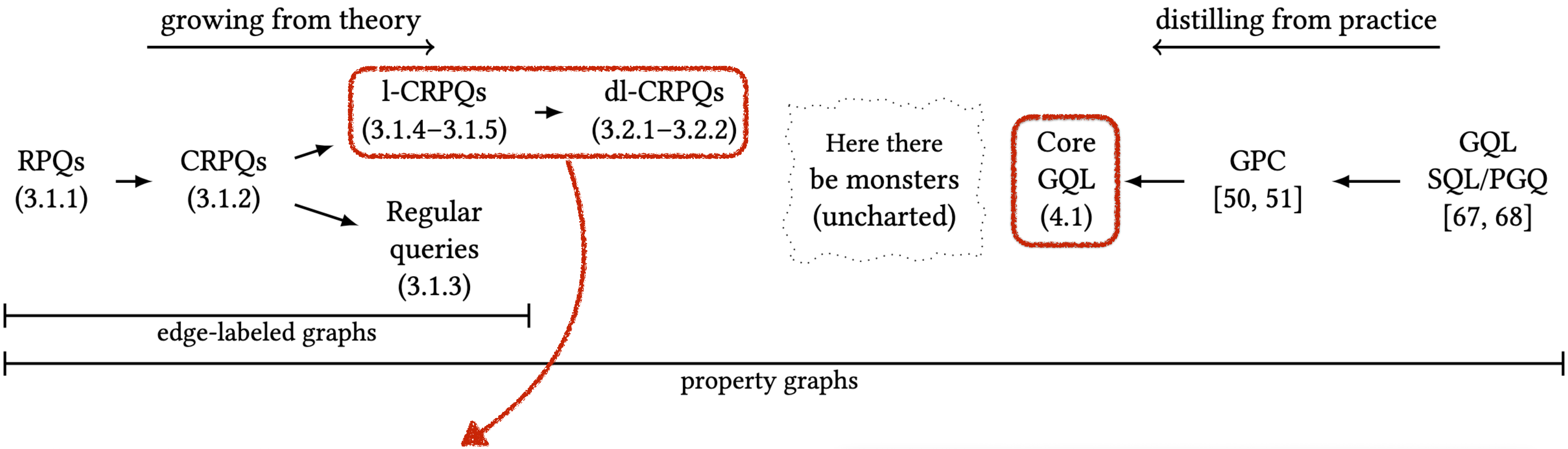
Syntax-Driven

Efficient

Actually work in practice

(https://www.geologyin.com/2024/05/rift-valleys.html)

What’s Next in the Talk?



growing from theory

distilling from practice

RPQs (3.1.1)

CRPQs (3.1.2)

l-CRPQs (3.1.4–3.1.5)

dl-CRPQs (3.2.1–3.2.2)

Regular queries (3.1.3)

Here there be monsters (uncharted)

Core GQL (4.1)

GPC [50, 51]

GQL SQL/PGQ [67, 68]

edge-labeled graphs

property graphs

We choose to follow

(1) compatibility with automata

(2) set semantics

(3) symmetric treatment of nodes and edges

These principles seem to work very well

- definitions become elegant
- fewer semantic issues
- a lot of potential for query optimization

(language becomes “more declarative”)

Growing From Theory

CRPQs with List Variables

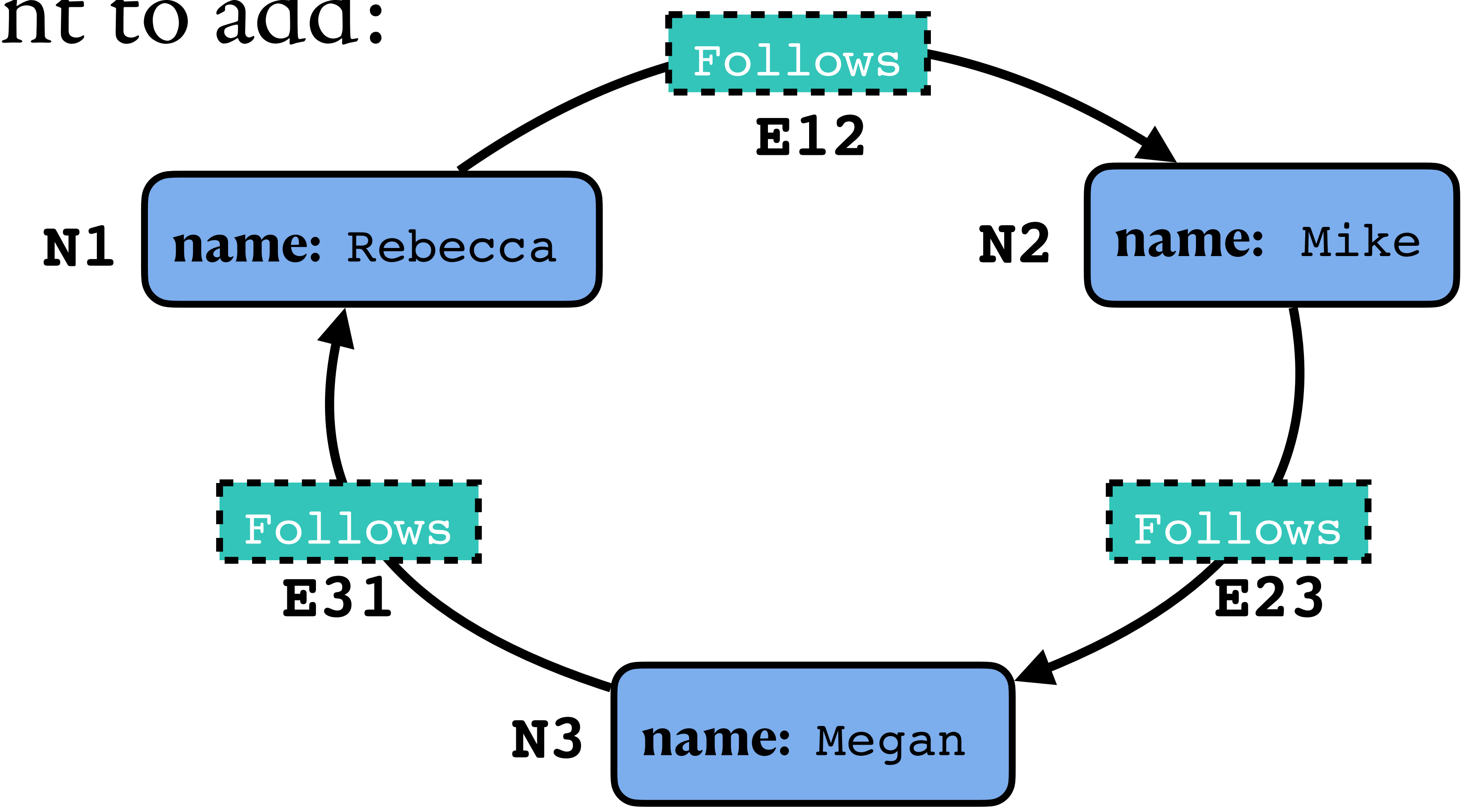
Standard CRPQs

$$q(x_1, \ldots, x_k) \;=\; \bigwedge_{i=1}^{n} y_i \xrightarrow{r_i} z_i$$

Ingredients

- Output variables: $x_1, \ldots, x_k$
- Join variables: $y_1, z_1, \ldots, y_n, z_n$
- Regular expressions : $r_1, \ldots, r_n$

We want to add:



capability to return elements
along matched paths

Current GQL

```
  ()-[z:Follows]->()-[z:Follows]->()
( ()-[z:Follows]->()-[z:Follows]->() )+
```

$\leadsto$ **z** is a join variable
$\leadsto$ **z** is a “list” variable
(called group variable in GQL)

CRPQs with List Variables

Standard CRPQs

$$q(x_1, \ldots, x_k) \;=\; \bigwedge_{i=1}^{n} y_i \xrightarrow{r_i} z_i$$

Ingredients

- Output variables: $x_1, \ldots, x_k$
- Join variables: $y_1, z_1, \ldots, y_n, z_n$
- Regular expressions : $r_1, \ldots, r_n$

CRPQs with List Variables

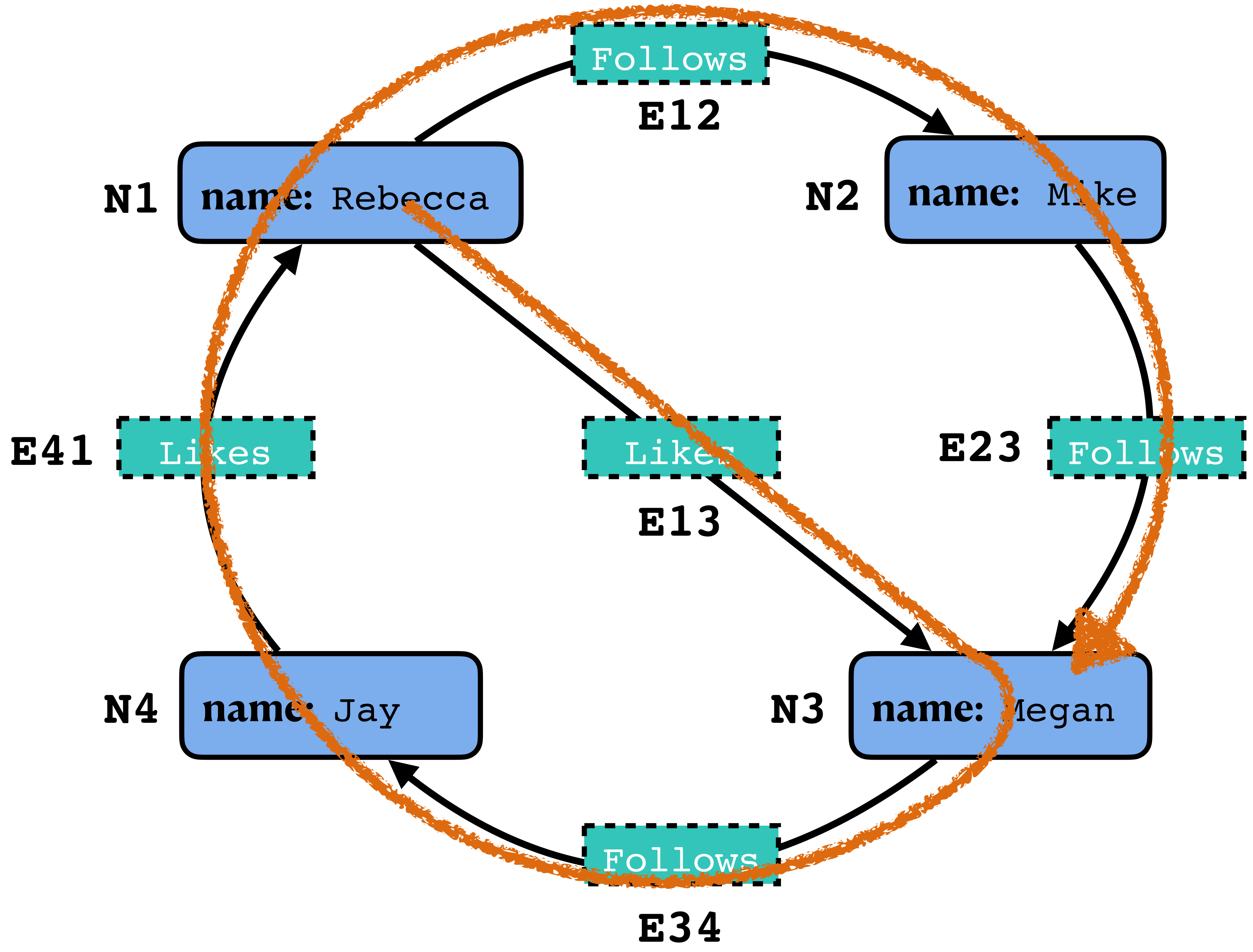
$$q(x_1, \ldots, x_k) \;=\; \bigwedge_{i=1}^{n} y_i \xrightarrow{r_i} z_i$$

Ingredients

- Output variables: $x_1, \ldots, x_k$
- **Join variables**: $y_1, z_1, \ldots, y_n, z_n$
- Regular expressions with **list variables**: $r_1, \ldots, r_n$

Keep **join variables** & **list variables** separated
We'll soon see why

REs with List Variables



(Follows^z + Likes)*

- Paths in which every edge is labeled Follows or Likes
- Annotate the Follows edges with variable z

Path: $\text{E13} — \underset{z}{\text{E34}} — \text{E41} — \underset{z}{\text{E12}} — \underset{z}{\text{E23}}$

⇝ z binds to [E34, E12, E23]

Why a Design Like This?

Such REs

(1) are highly compatible with automata

(2) allow a product graph construction

Again, Why?

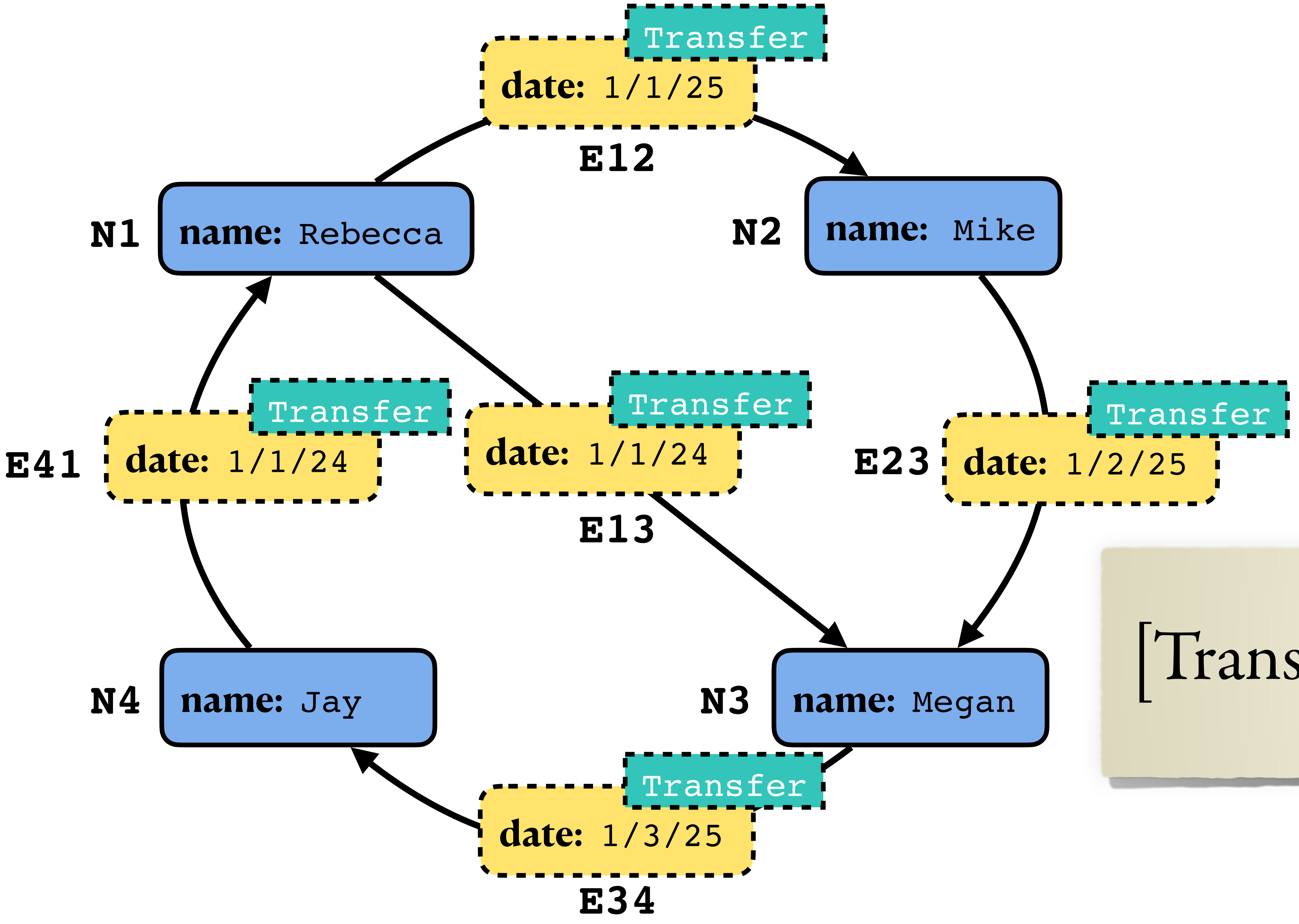
(1) ⇝ Enables more query optimization

(2) ⇝ Enables factorization for matching paths

[Fagin, Kimelfeld, Reiss, Vansummeren PODS'13]
[Riveros, Van Sint Jan, Vrgoc VLDB'23]
[Doleschal, Kimelfeld, M., Nahshon, Neven PODS'19]

[M., Niewerth, Popp, Rojas, Vansummeren, Vrgoc VLDB'23]
[Farias, M., Rojas, Vrgoc ISWC'24]

CRPQs with Data & List Variables



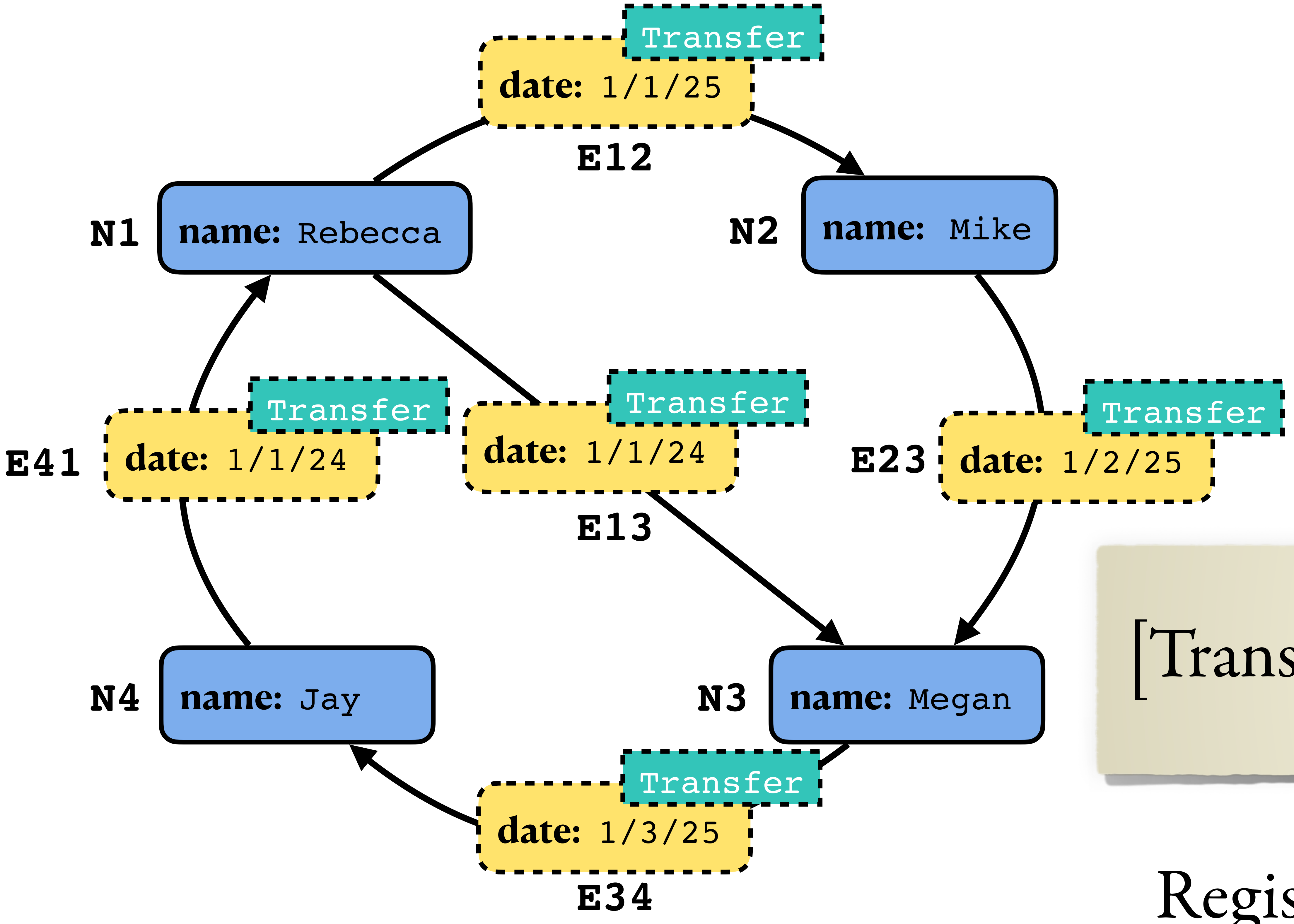
We're Going to Add

- node & edge treatment
- data filters

Increasing date values on edges

$$[\text{Transfer}^{z}]\,[x := \text{date}]\Big((\_)\,[\text{Transfer}^{z}]\,[\text{date} > x]\,[x := \text{date}]\Big)^{*}$$

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

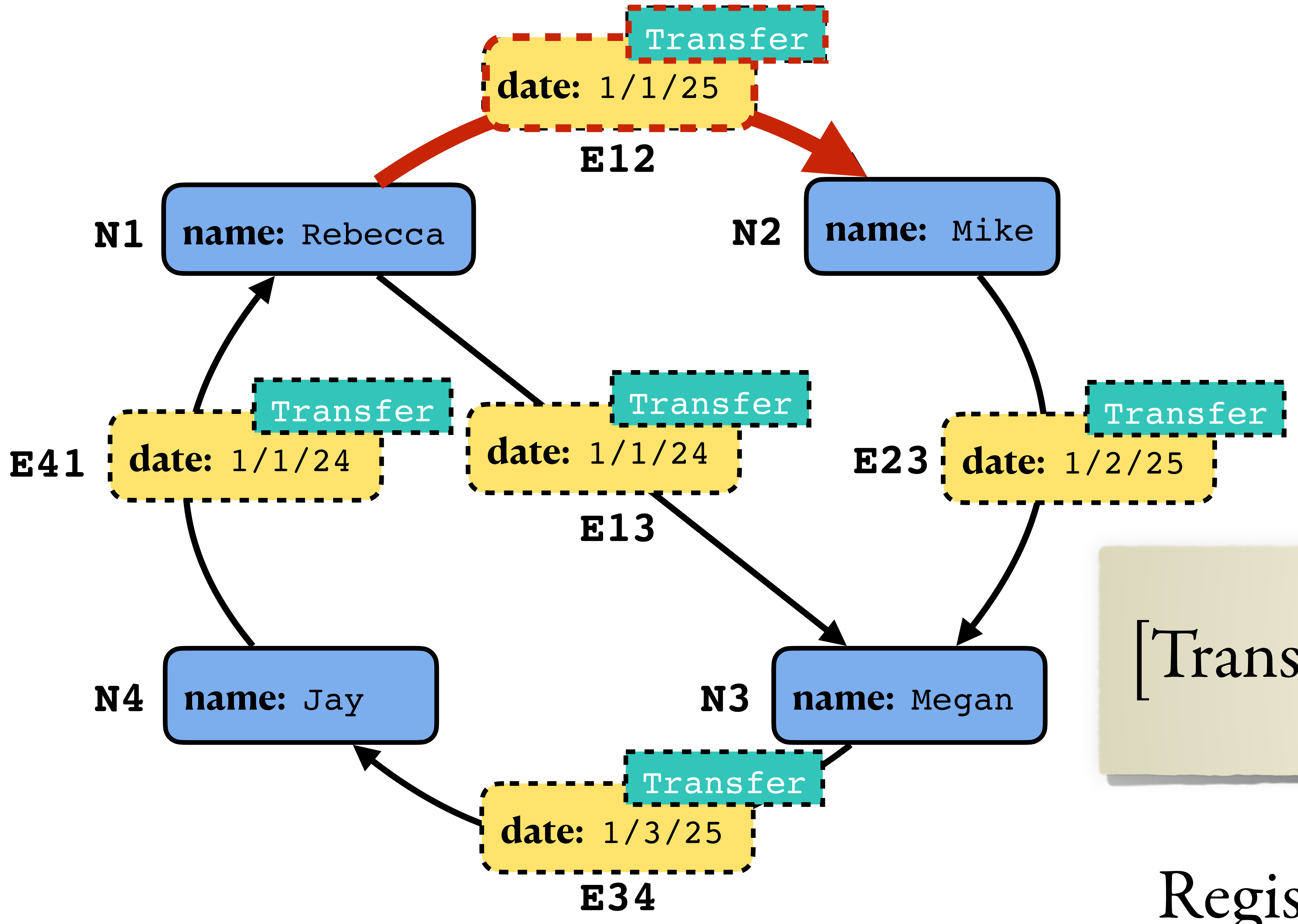
Increasing date values on edges

$[\text{Transfer}^z]$

Register

Considered path:

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

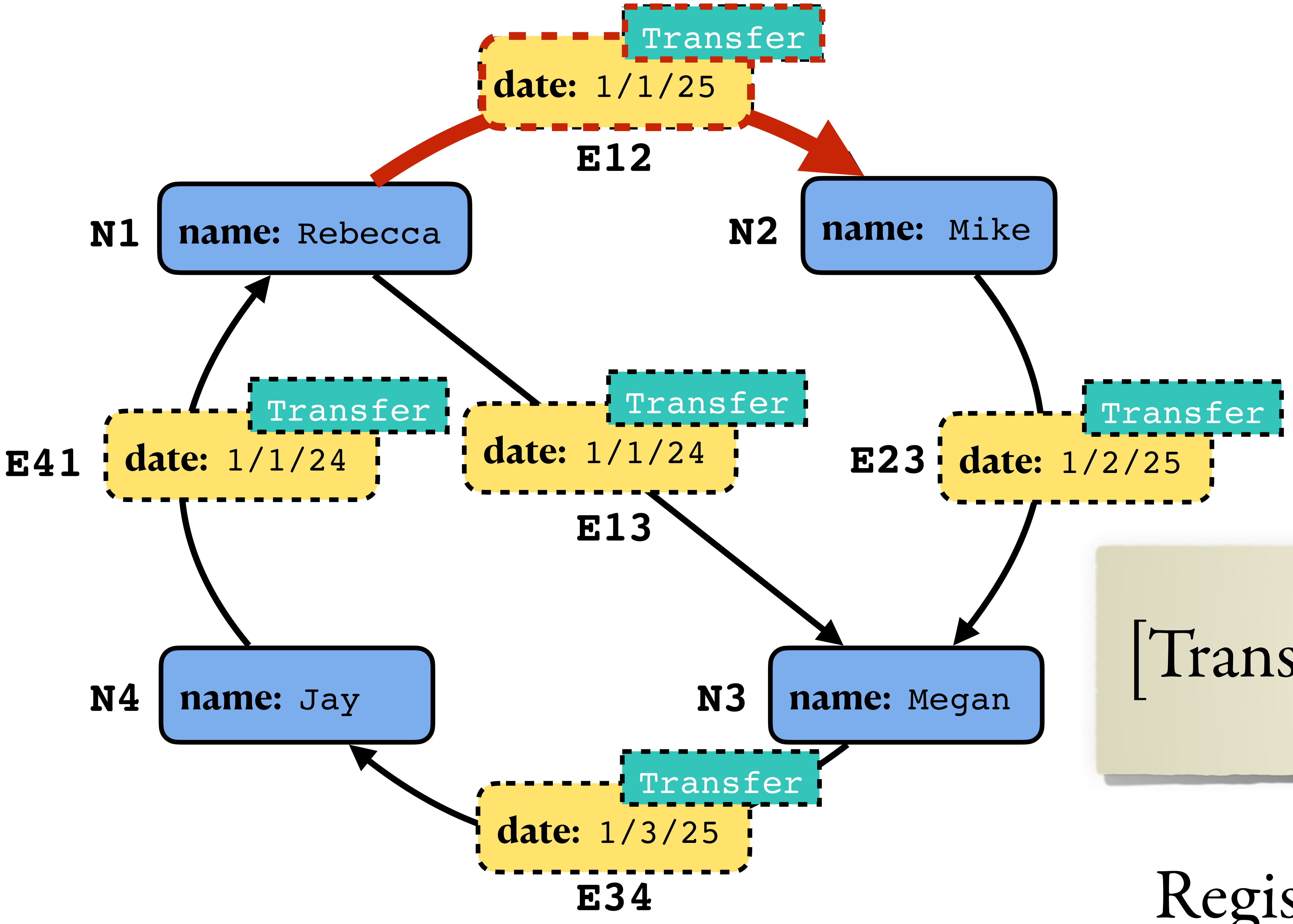
Increasing date values on edges

$[\text{Transfer}^{z}]$

Register

Considered path:

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

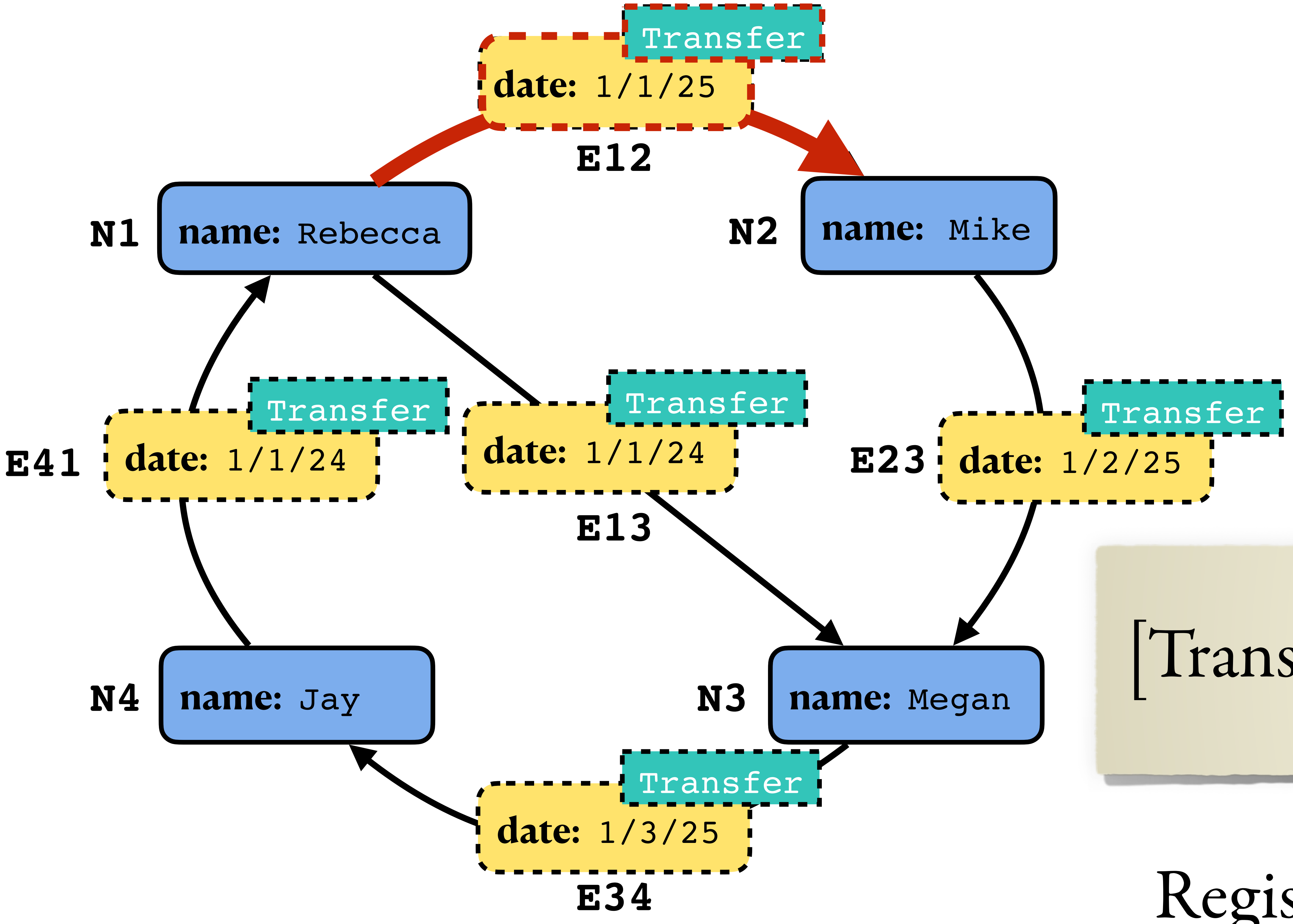
Increasing date values on edges

$[\mathrm{Transfer}^{z}]$

Register

Considered path: $\underset{z}{\mathrm{E12}}$

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

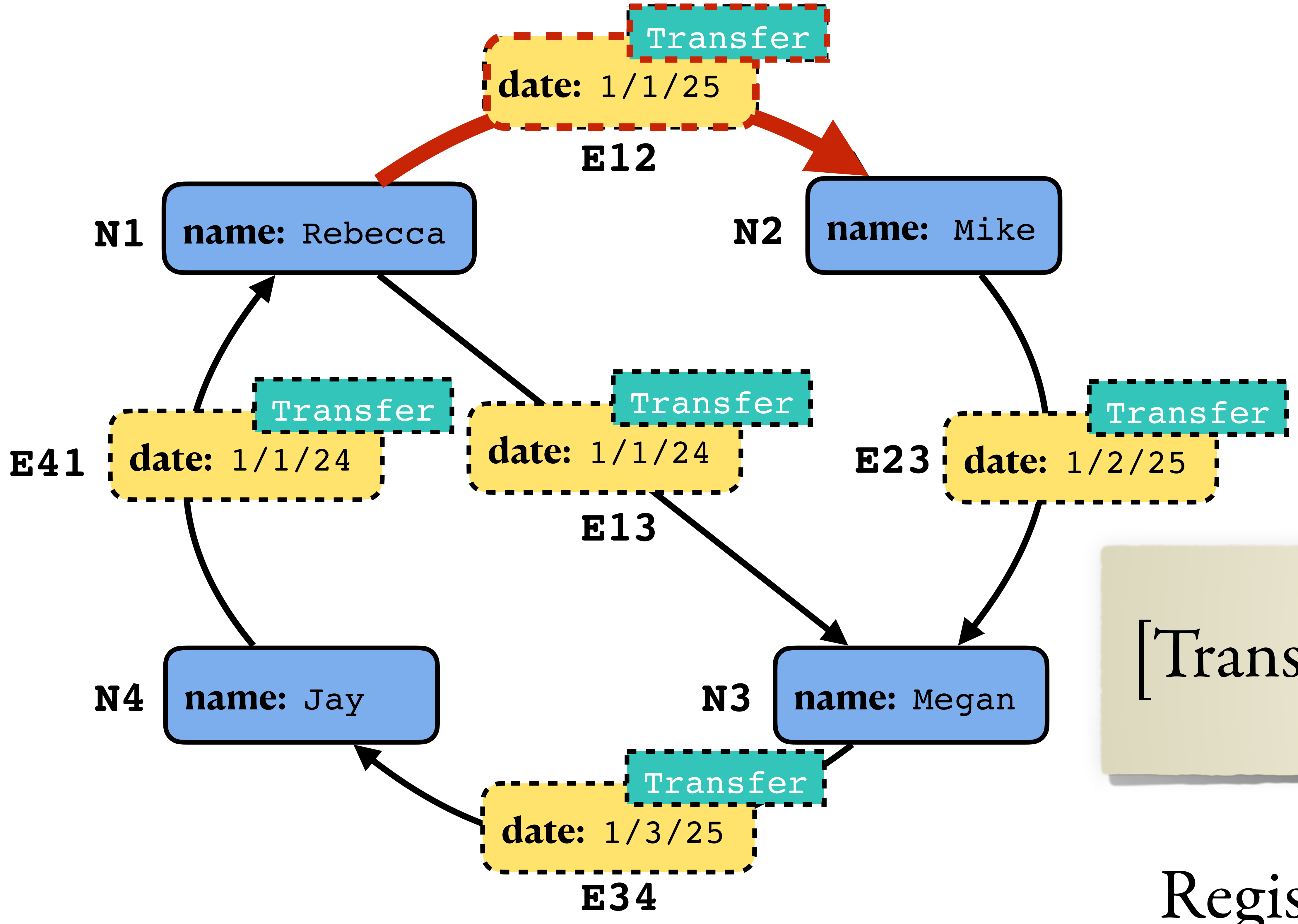
Increasing date values on edges

$$[\text{Transfer}^{z}]\,[x := \text{date}]$$

Register

Considered path: $\underset{z}{\text{E12}}$

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

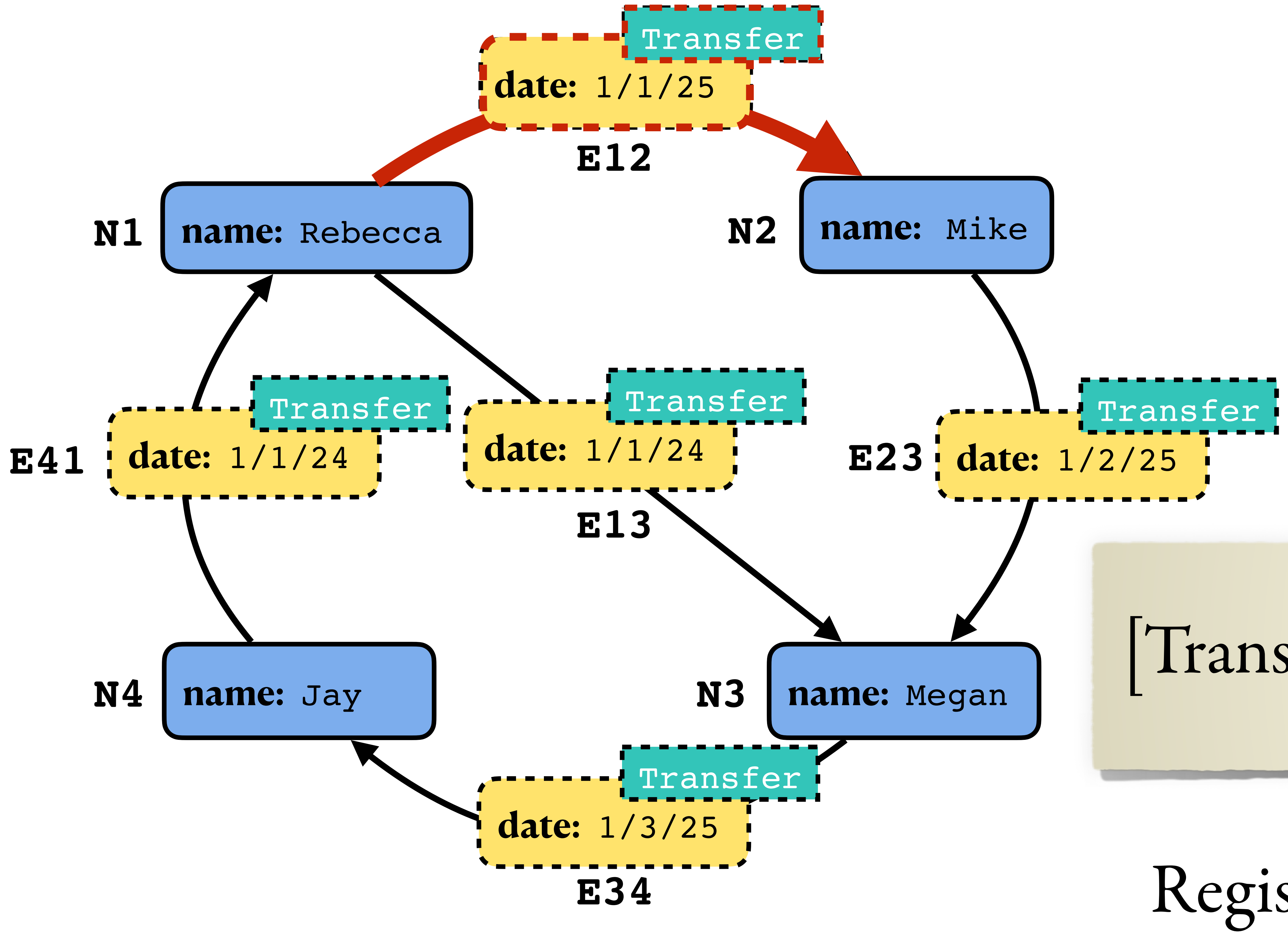
Increasing date values on edges

$[\text{Transfer}^{z}]\,[x := \text{date}]$

Register $x = 1/1/25$

Considered path: $\underset{z}{\text{E12}}$

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

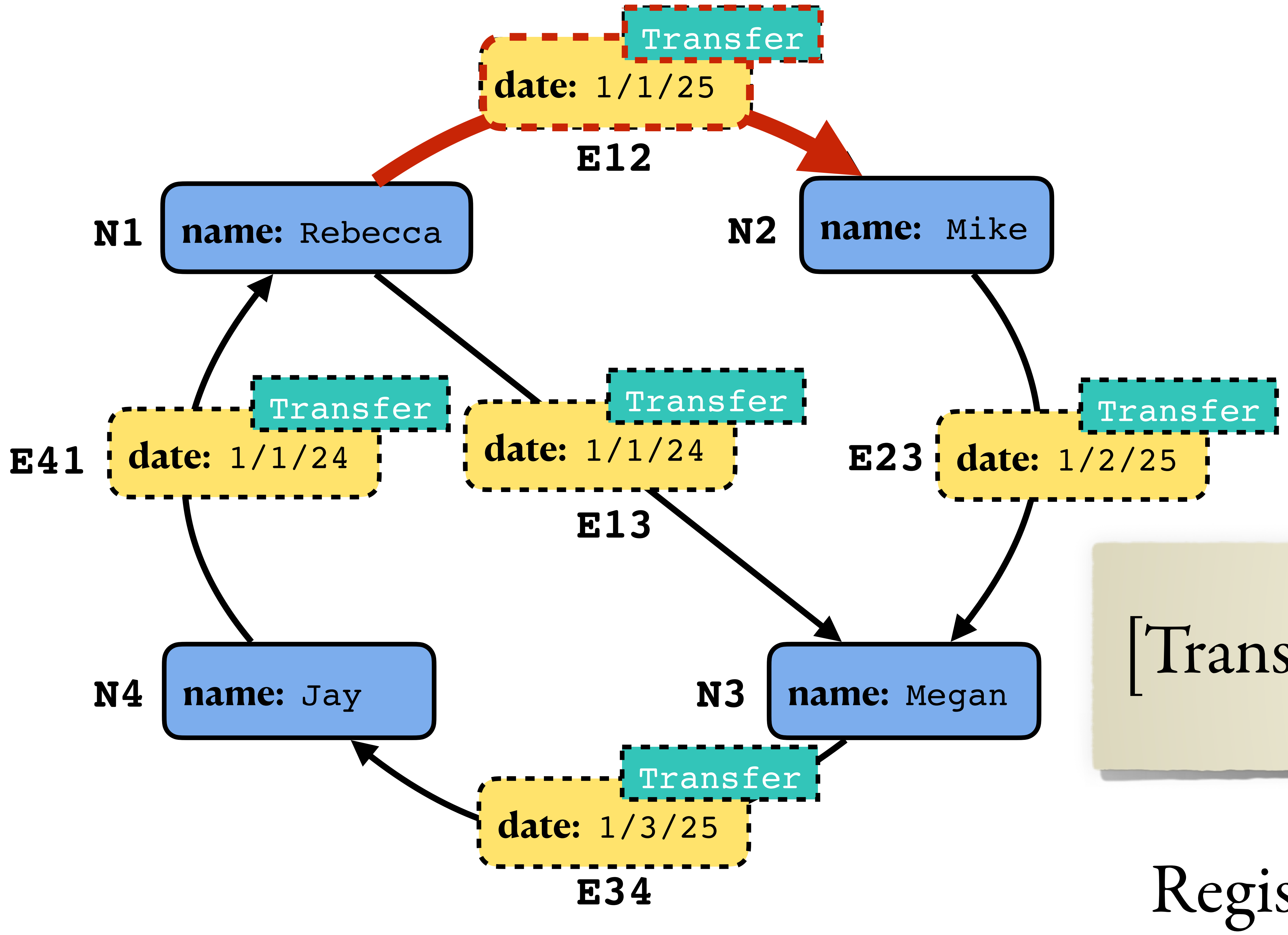
Increasing date values on edges

$$[\text{Transfer}^{z}]\,[x := \text{date}]\Big(\qquad\qquad\qquad\qquad\qquad\qquad\Big)^{*}$$

Register $x = 1/1/25$

Considered path: $\underset{z}{\text{E12}}$

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

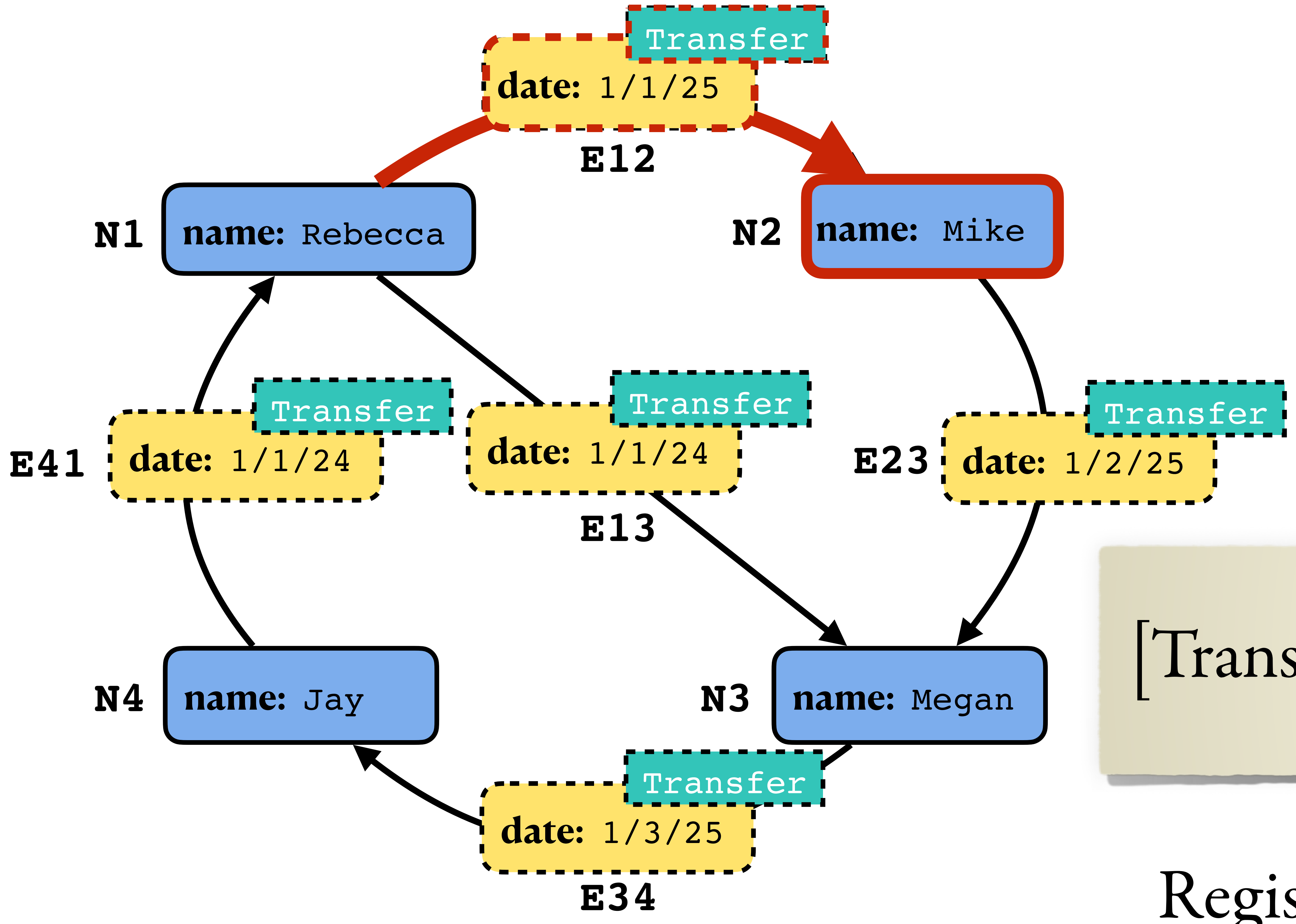
Increasing date values on edges

$$[\mathrm{Transfer}^{z}]\,[x := \mathrm{date}]\Big((\_) \qquad\qquad \Big)^{*}$$

Register $x = 1/1/25$

Considered path: $\underset{z}{\mathrm{E12}}$

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

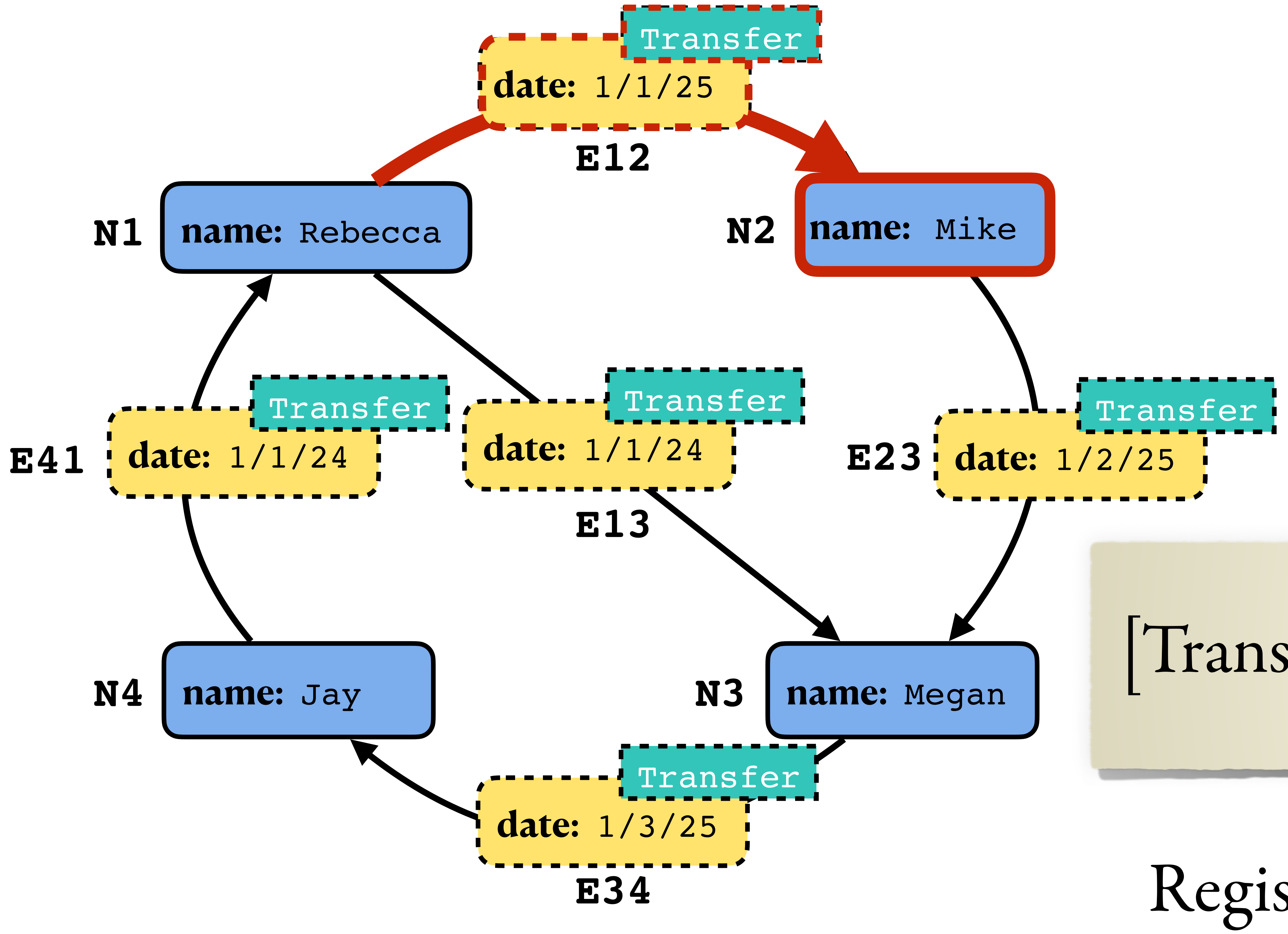
Increasing date values on edges

$$[\text{Transfer}^{z}]\,[x := \text{date}]\Big((\_) \qquad\qquad\qquad\qquad \Big)^{*}$$

Register $x = 1/1/25$

Considered path: $\underset{z}{\text{E12}} \longrightarrow \text{N2}$

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

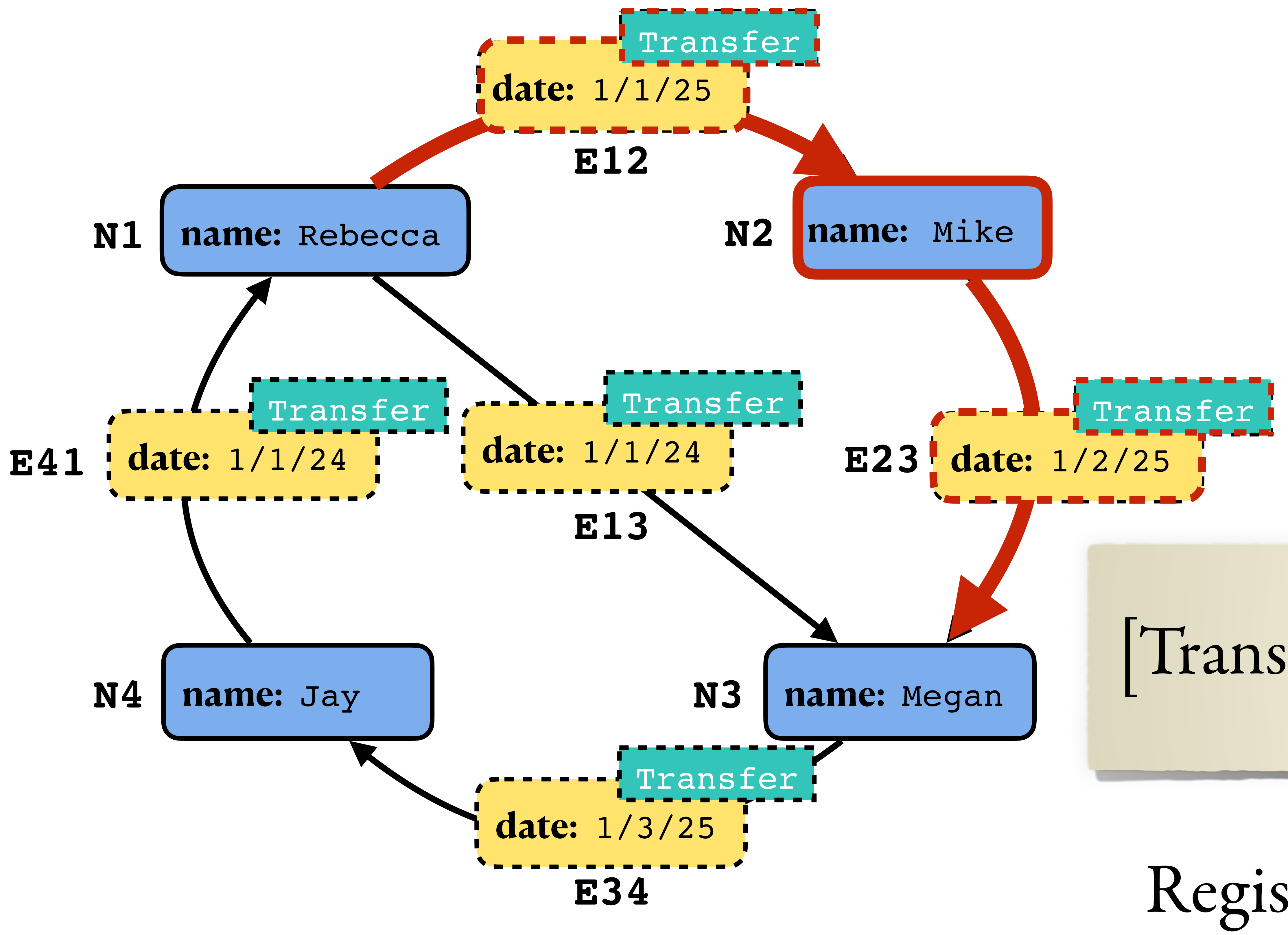
Increasing date values on edges

$$[\mathrm{Transfer}^{z}]\,[x := \mathrm{date}]\Big(\big(\_\big)\,[\mathrm{Transfer}^{z}]\qquad\qquad\Big)^{*}$$

Register $x = 1/1/25$

Considered path: $\underset{z}{\mathrm{E12}} \longrightarrow \mathrm{N2}$

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

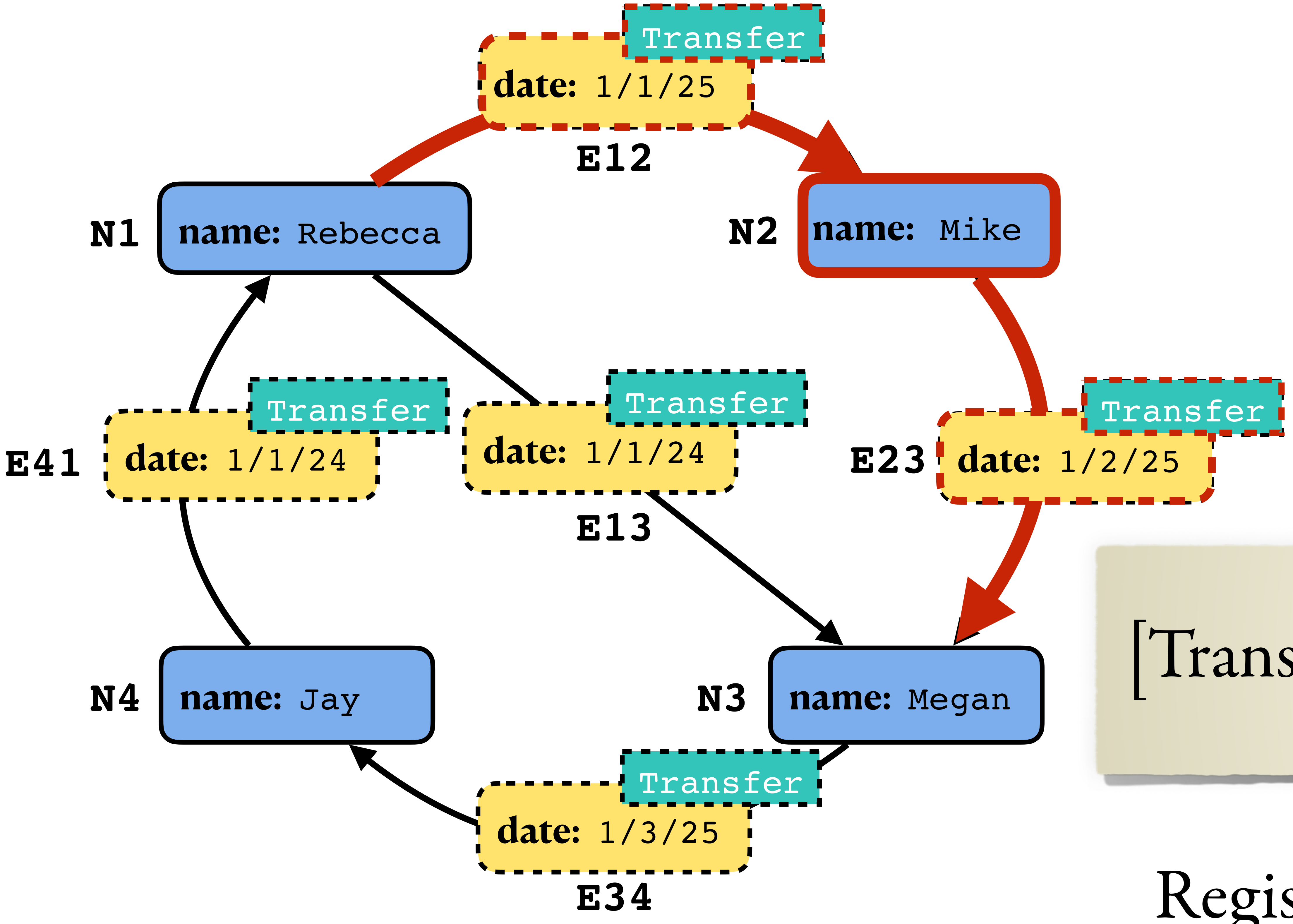
Increasing date values on edges

$$[\text{Transfer}^{z}]\,[x := \text{date}]\Big(\big(\_\big)\,[\text{Transfer}^{z}]\qquad\qquad\Big)^{*}$$

Register $x = 1/1/25$

Considered path: $\underset{z}{\text{E12}} \longrightarrow \text{N2} \longrightarrow \underset{z}{\text{E23}}$

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

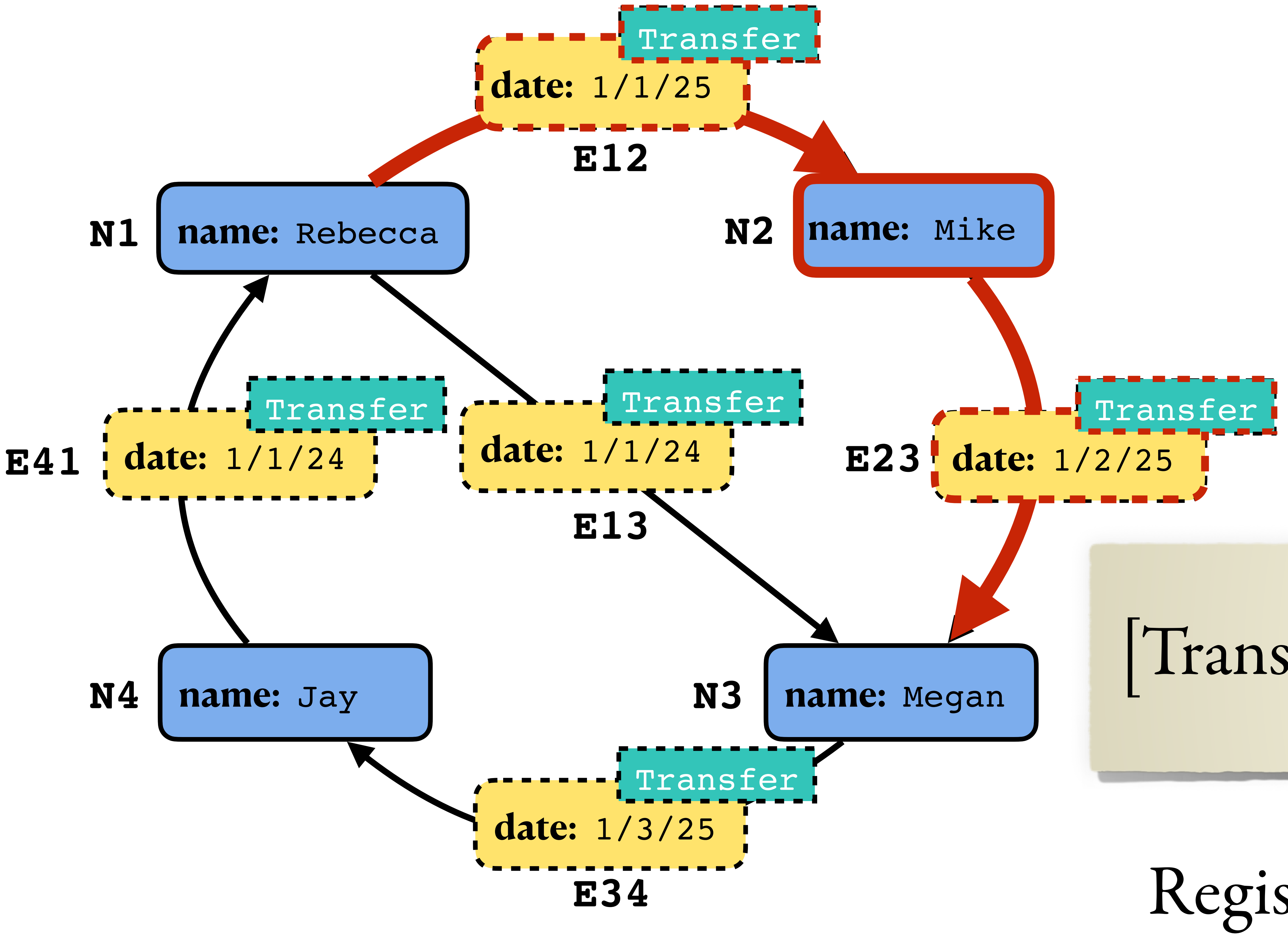
Increasing date values on edges

$$[\text{Transfer}^{z}]\,[x := \text{date}]\Big(\big(\_\big)\,[\text{Transfer}^{z}]\,[\text{date} > x]\Big)^{*}$$

Register $x = 1/1/25$

Considered path: $\underset{z}{\text{E12}} \longrightarrow \text{N2} \longrightarrow \underset{z}{\text{E23}}$

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

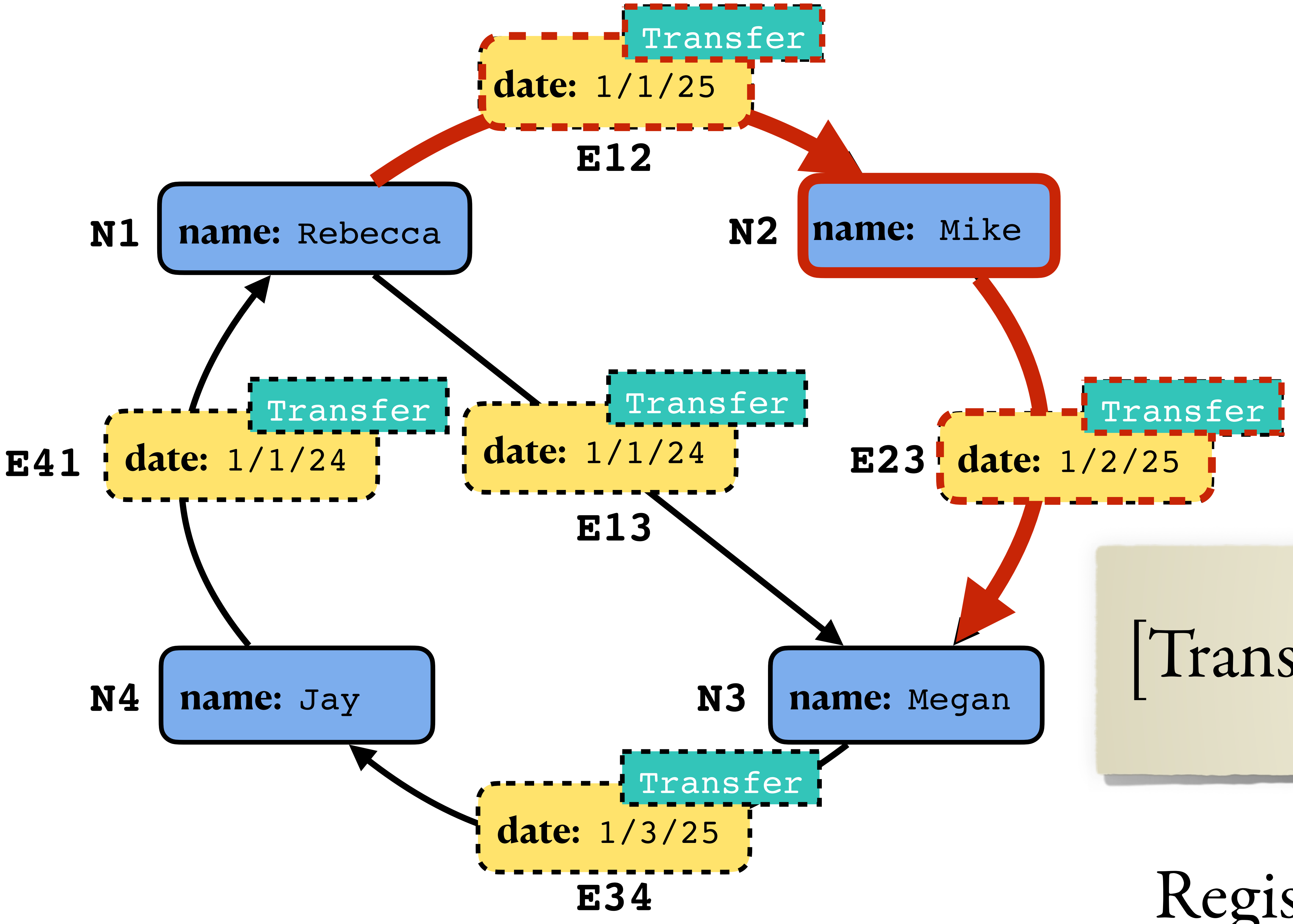
Increasing date values on edges

$$[\text{Transfer}^{z}]\,[x := \text{date}]\Big(\big(\_\big)\,[\text{Transfer}^{z}]\,[\text{date} > x]\qquad\Big)^{*}$$

Register $x = 1/1/25$ date = 1/2/25 ✓

Considered path: $\underset{z}{\text{E12}} \longrightarrow \text{N2} — \underset{z}{\text{E23}}$

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

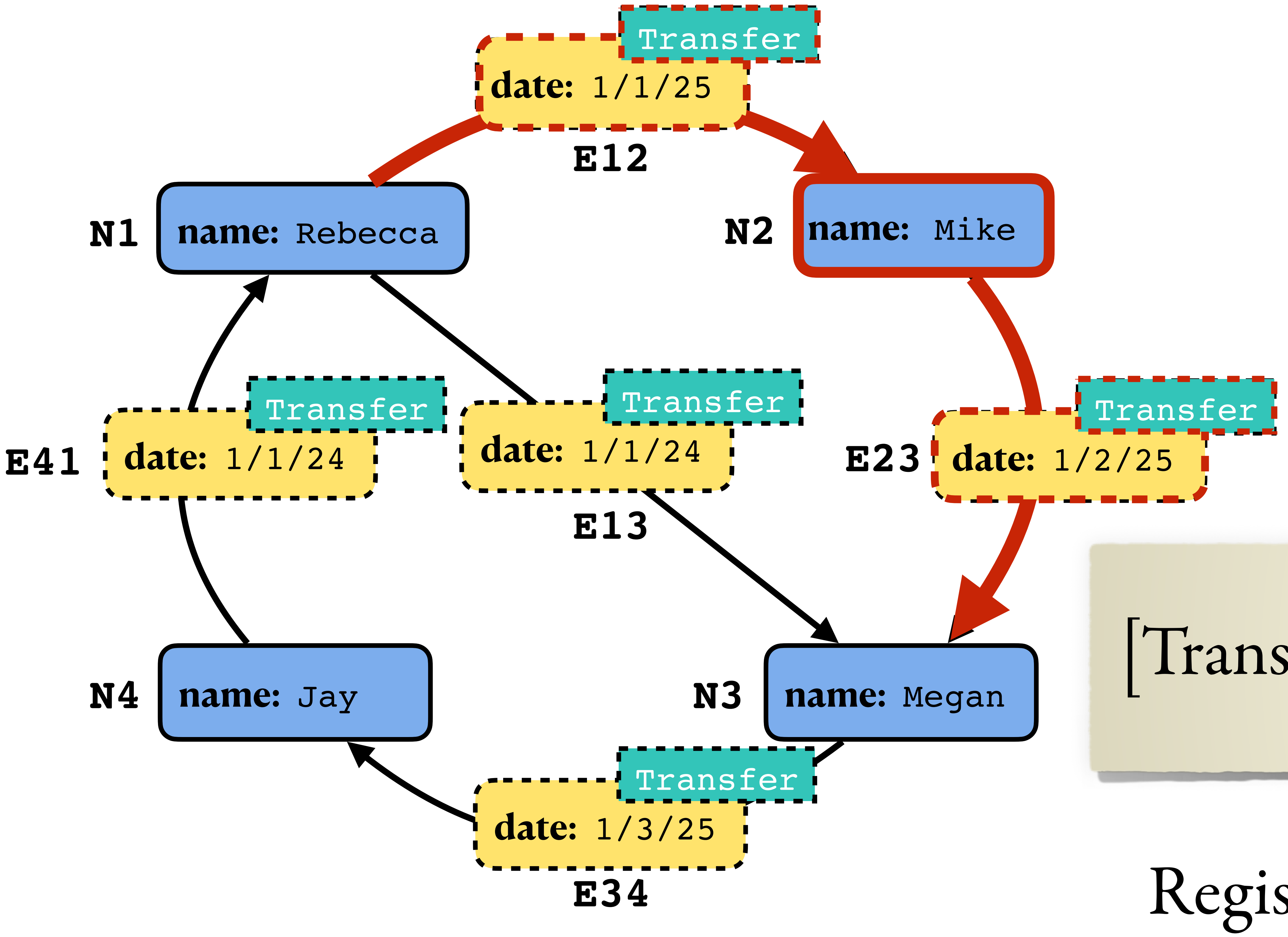
Increasing date values on edges

$$[\text{Transfer}^{z}]\,[x := \text{date}]\Big(\big(\_\big)\,[\text{Transfer}^{z}]\,[\text{date} > x]\Big)^{*}$$

Register $x = 1/1/25$

Considered path: $\underset{z}{\text{E12}} \longrightarrow \text{N2} \text{——} \underset{z}{\text{E23}}$

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

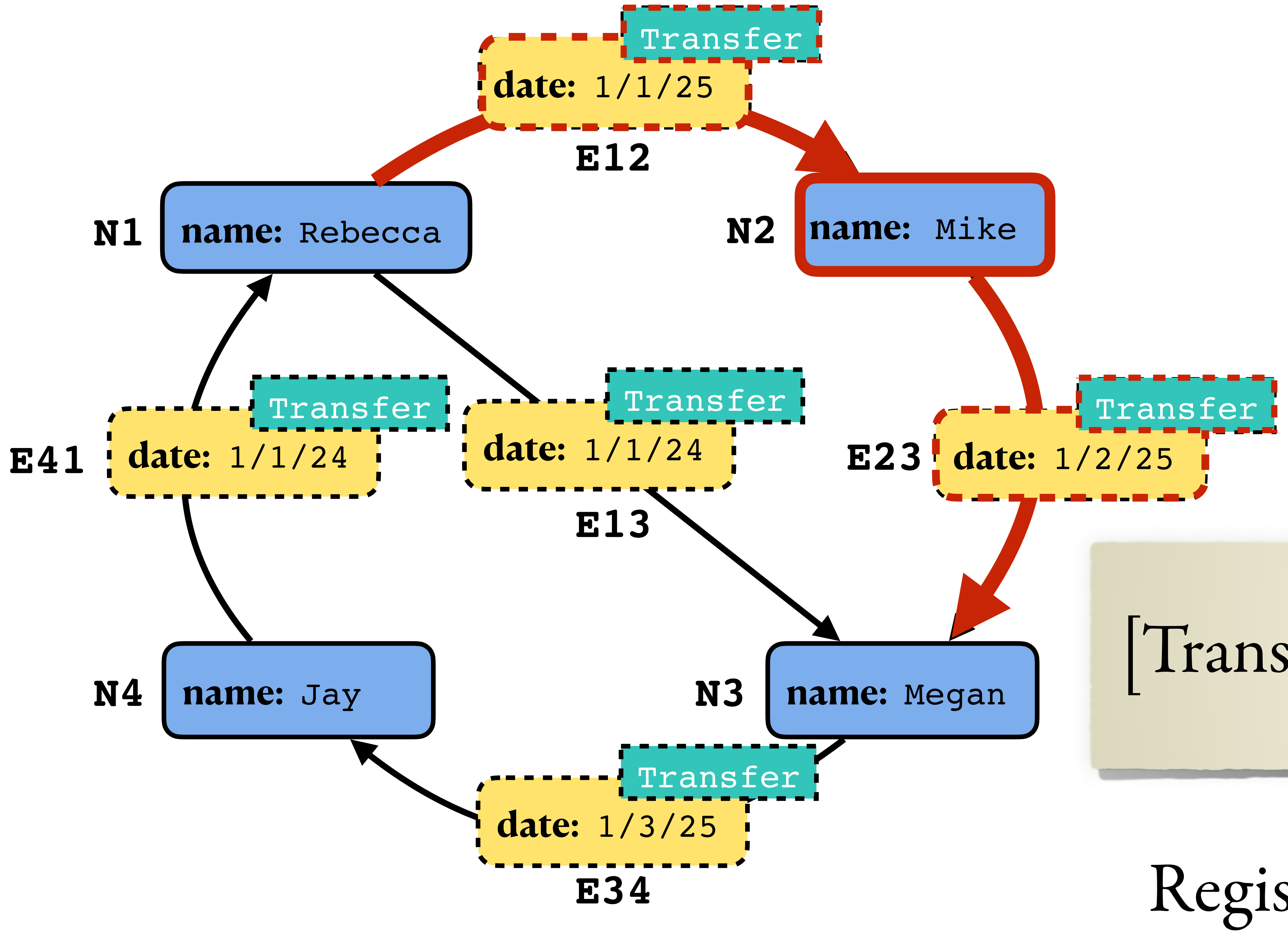
Increasing date values on edges

$$[\text{Transfer}^{z}]\,[x := \text{date}]\Big(\big(\_\big)\,[\text{Transfer}^{z}]\,[\text{date} > x]\,[x := \text{date}]\Big)^{*}$$

Register $x = 1/1/25$

Considered path: $\underset{z}{\text{E12}} \longrightarrow \text{N2} —— \underset{z}{\text{E23}}$

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

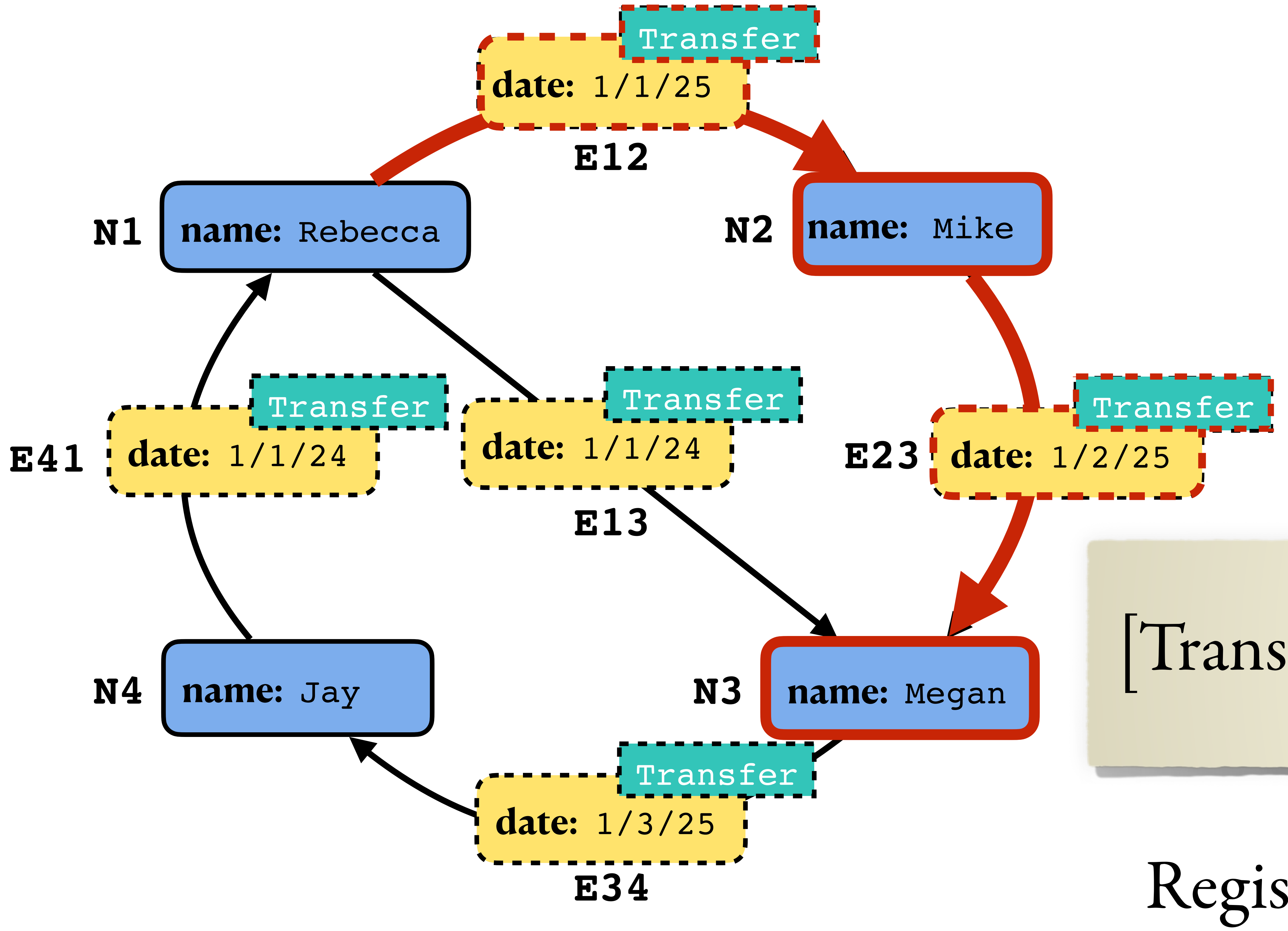
Increasing date values on edges

$$[\text{Transfer}^{z}]\,[x := \text{date}]\Big((\_)\,[\text{Transfer}^{z}]\,[\text{date} > x]\,[x := \text{date}]\Big)^{*}$$

Register $x = 1/2/25$

Considered path: $\underset{z}{\text{E12}} \longrightarrow \text{N2} \text{——} \underset{z}{\text{E23}}$

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

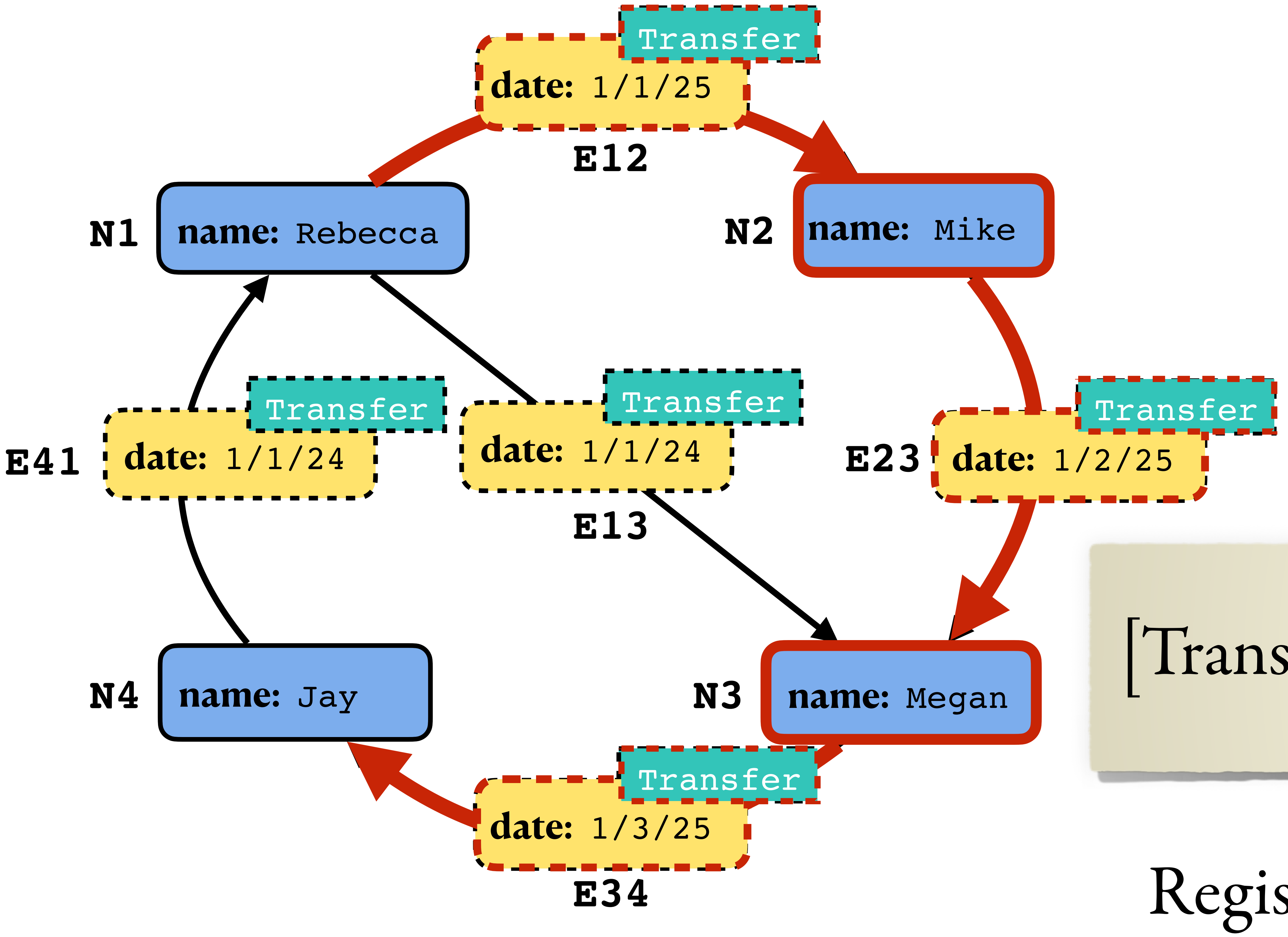
Increasing date values on edges

$$[\mathrm{Transfer}^{z}]\,[x := \mathrm{date}]\Big((\_)\,[\mathrm{Transfer}^{z}]\,[\mathrm{date} > x]\,[x := \mathrm{date}]\Big)^{*}$$

Register $x = 1/2/25$

Considered path: $\underset{z}{\mathrm{E12}} \longrightarrow \mathrm{N2} \longrightarrow \underset{z}{\mathrm{E23}} \longrightarrow \mathrm{N3}$

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

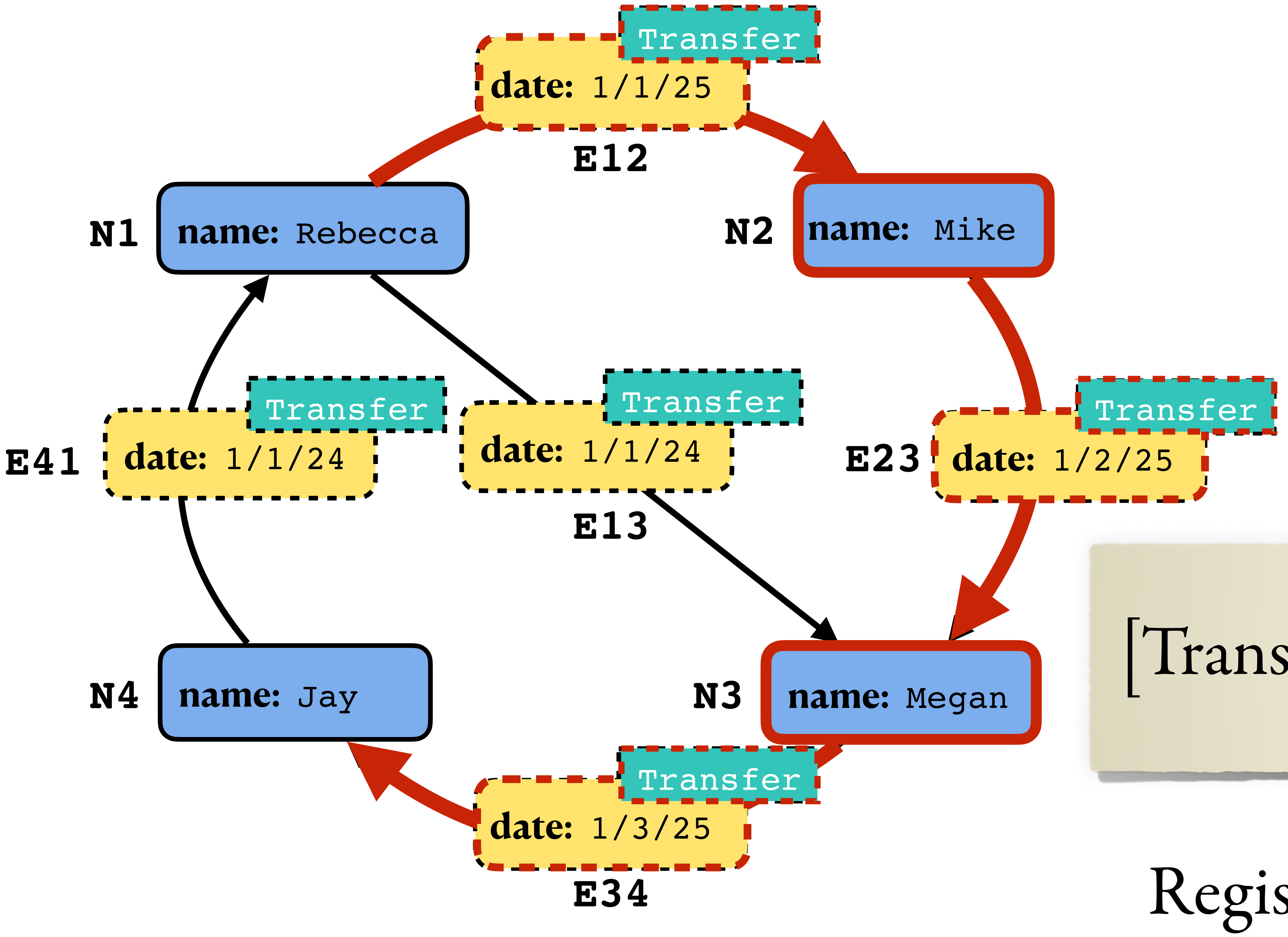
Increasing date values on edges

$$[\mathrm{Transfer}^{z}]\,[x := \mathrm{date}]\Big(\big(\_\big)\,[\mathrm{Transfer}^{z}]\,[\mathrm{date} > x]\,[x := \mathrm{date}]\Big)^{*}$$

Register $x = 1/2/25$

Considered path: $\underset{z}{\text{E12}} \longrightarrow \text{N2} \text{ — } \underset{z}{\text{E23}} \longrightarrow \text{N3} \text{ — } \underset{z}{\text{E34}}$

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

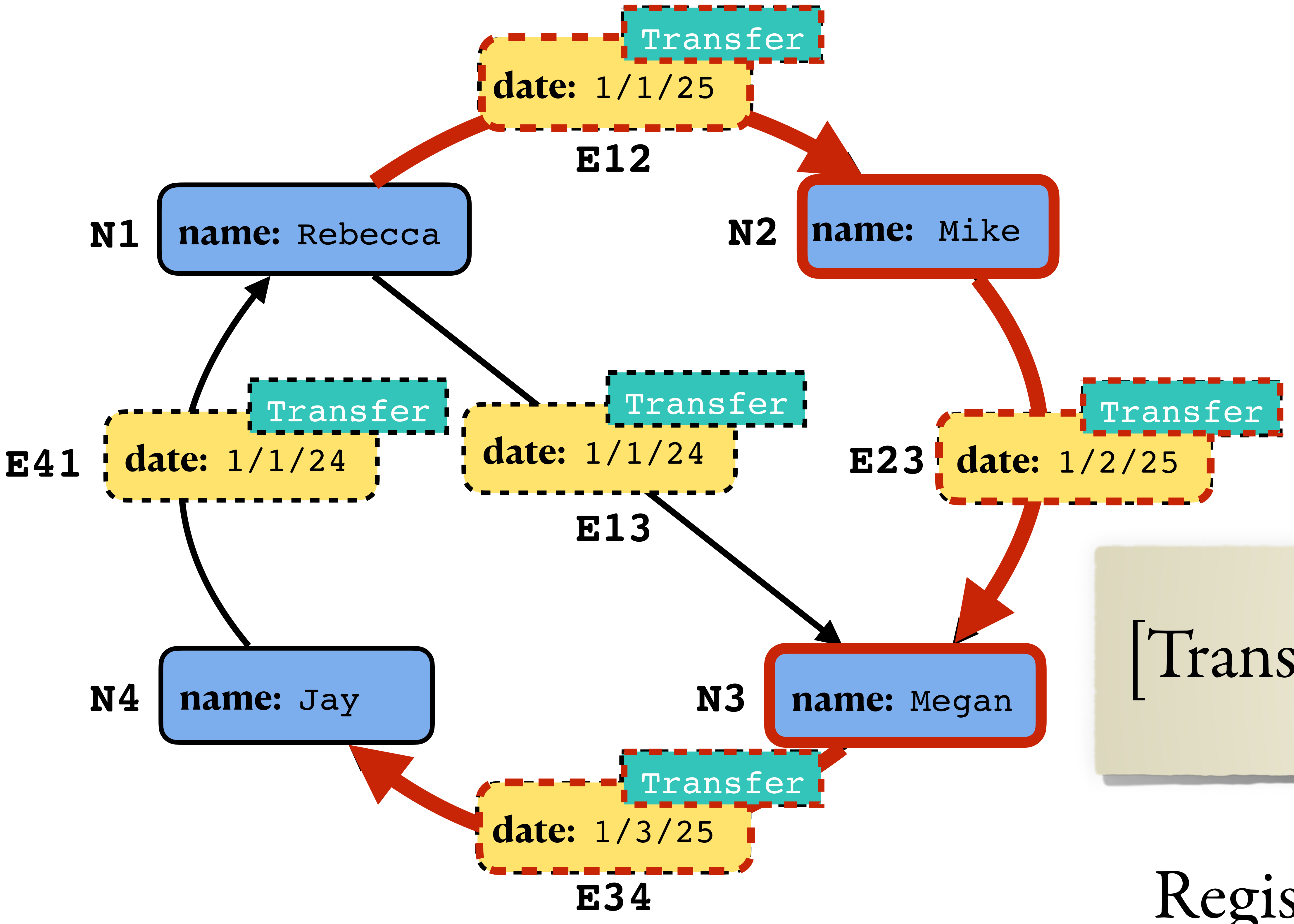
Increasing date values on edges

$$[\text{Transfer}^{z}]\,[x := \text{date}]\Big(\big(\_\big)\,[\text{Transfer}^{z}]\;[\text{date} > x]\;[x := \text{date}]\Big)^{*}$$

Register $x = 1/3/25$

Considered path: $\underset{z}{\text{E12}} \longrightarrow \text{N2} \text{ —— } \underset{z}{\text{E23}} \longrightarrow \text{N3} \text{ —— } \underset{z}{\text{E34}}$

CRPQs with Data & List Variables



We're Going to Add

- node & edge treatment
- data filters

Increasing date values on edges

$$[\text{Transfer}^{z}]\,[x := \text{date}]\Big((\_)\,[\text{Transfer}^{z}]\,[\text{date} > x]\,[x := \text{date}]\Big)^{*}$$

Register $x = 1/3/25$

paths don't need to be node-to-node (as in GQL)

Considered path: $\underset{z}{\text{E12}} \longrightarrow \text{N2} \text{---} \underset{z}{\text{E23}} \longrightarrow \text{N3} \text{---} \underset{z}{\text{E34}}$

CRPQs with Data & List Variables

Increasing date values on **nodes**

$$(\mathrm{Transfer}^{z})\ (x := \mathrm{date})\ \Big([\_]\ (\mathrm{Transfer}^{z})\ (\mathrm{date} > x)\ (x := \mathrm{date}) \Big)^{*}$$

node-to-node paths

Symmetry!

$$[\mathrm{Transfer}^{z}]\ [x := \mathrm{date}]\ \Big((\_)\ [\mathrm{Transfer}^{z}]\ [\mathrm{date} > x]\ [x := \mathrm{date}] \Big)^{*}$$

edge-to-edge paths

Increasing date values on **edges**

CRPQs with Data & List Variables

https://www.raullara.net/wp-content/uploads/2023/07/Reflective-Symmetry-art-1024x576.jpg

CRPQs with Data & List Variables

Increasing date values on **nodes**

$$(\mathrm{Transfer}^{z})\ (x := \mathrm{date})\ \Big([\_]\ (\mathrm{Transfer}^{z})\ (\mathrm{date} > x)\ (x := \mathrm{date}) \Big)^{*}$$

node-to-node
paths

Symmetry!

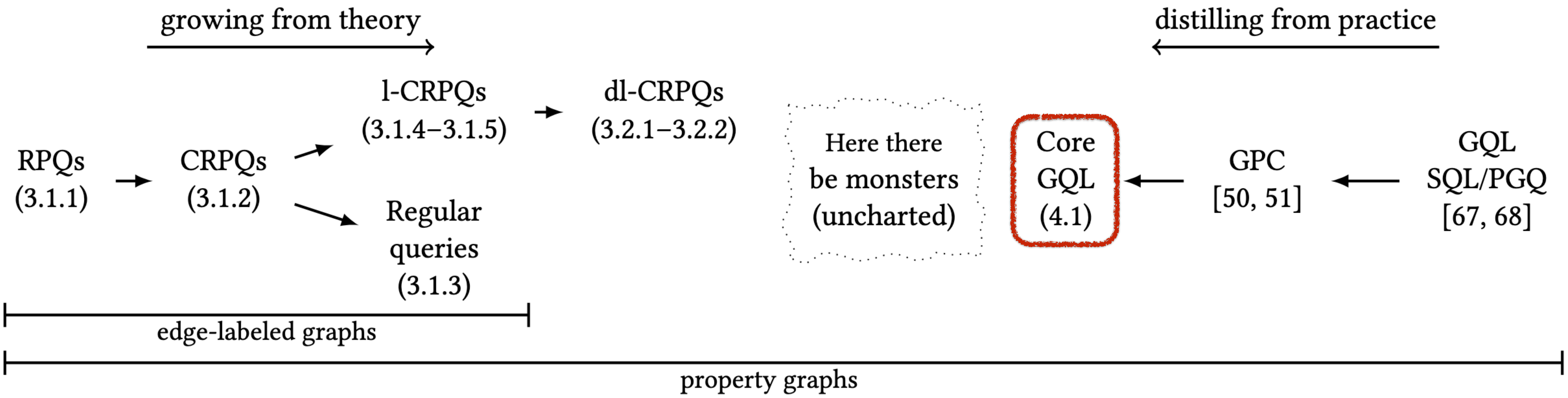
$$[\mathrm{Transfer}^{z}]\ [x := \mathrm{date}]\ \Big((\_)\ [\mathrm{Transfer}^{z}]\ [\mathrm{date} > x]\ [x := \mathrm{date}] \Big)^{*}$$

edge-to-edge
paths

Increasing date values on **edges**

Distilling From Practice

Distilling From Practice



growing from theory

distilling from practice

RPQs (3.1.1)

CRPQs (3.1.2)

l-CRPQs (3.1.4–3.1.5)

Regular queries (3.1.3)

dl-CRPQs (3.2.1–3.2.2)

Here there be monsters (uncharted)

Core GQL (4.1)

GPC [50, 51]

GQL SQL/PGQ [67, 68]

edge-labeled graphs

property graphs

CoreGQL

$$
\begin{array}{lcll}
\pi & := & (x) & \text{node} \\
 & \mid & \xrightarrow{x} & \text{edge} \\
 & \mid & \pi_1\ \pi_2 & \text{concatenation} \\
 & \mid & \pi_1 + \pi_2 & \text{disjunction} \\
 & \mid & \pi^{n..m} & \text{repetition} \\
 & \mid & \pi\langle\theta\rangle & \text{condition}
\end{array}
$$

$$\theta, \theta' \ := \ x.k = x'.k' \mid x.k < x'.k' \mid \ell(x) \mid \theta \vee \theta' \mid \theta \wedge \theta' \mid \neg\theta$$

More? ⇝ Leonid @ GRADES/NDA
Friday 2PM

Results About Distilled Models

Theorem [Gheerbrant, Libkin, Peterfreund, Rogova ICDT’25]

The RPQ $(aa)^*$ is not expressible using Cypher patterns

Theorem [Gheerbrant, Libkin, Peterfreund, Rogova PVLDB’25]

“Increasing values on edges” cannot be expressed without repeating variables

…

Research Agenda

for Graph Query Languages

Research Agenda for Graph Query Languages

Growing from Theory

- Design and study elegant models
- Understand expressiveness
- Understand complexity
- Use our knowledge of logics, automata,…
- …

Distilling from Practice

- Identify what's good
- Find opportunities for improvement
- Inexpressibility results
- Complexity lower bounds
- …

Research Agenda for Graph Query Languages

Growing from Theory

- Design and study elegant models
- Understand expressiveness
- Understand complexity
- Use our knowledge of logics, automata,…
- …

Distilling from Practice

- Identify what's good
- Find opportunities for improvement
- Inexpressibility results
- Complexity lower bounds
- …

Light side

(https://www.reddit.com/r/cats/comments/a1ofm6/kungfu_cat/)

Dark side

(https://de.pinterest.com/pin/724024077609835047/)

Research Agenda for Graph Query Languages

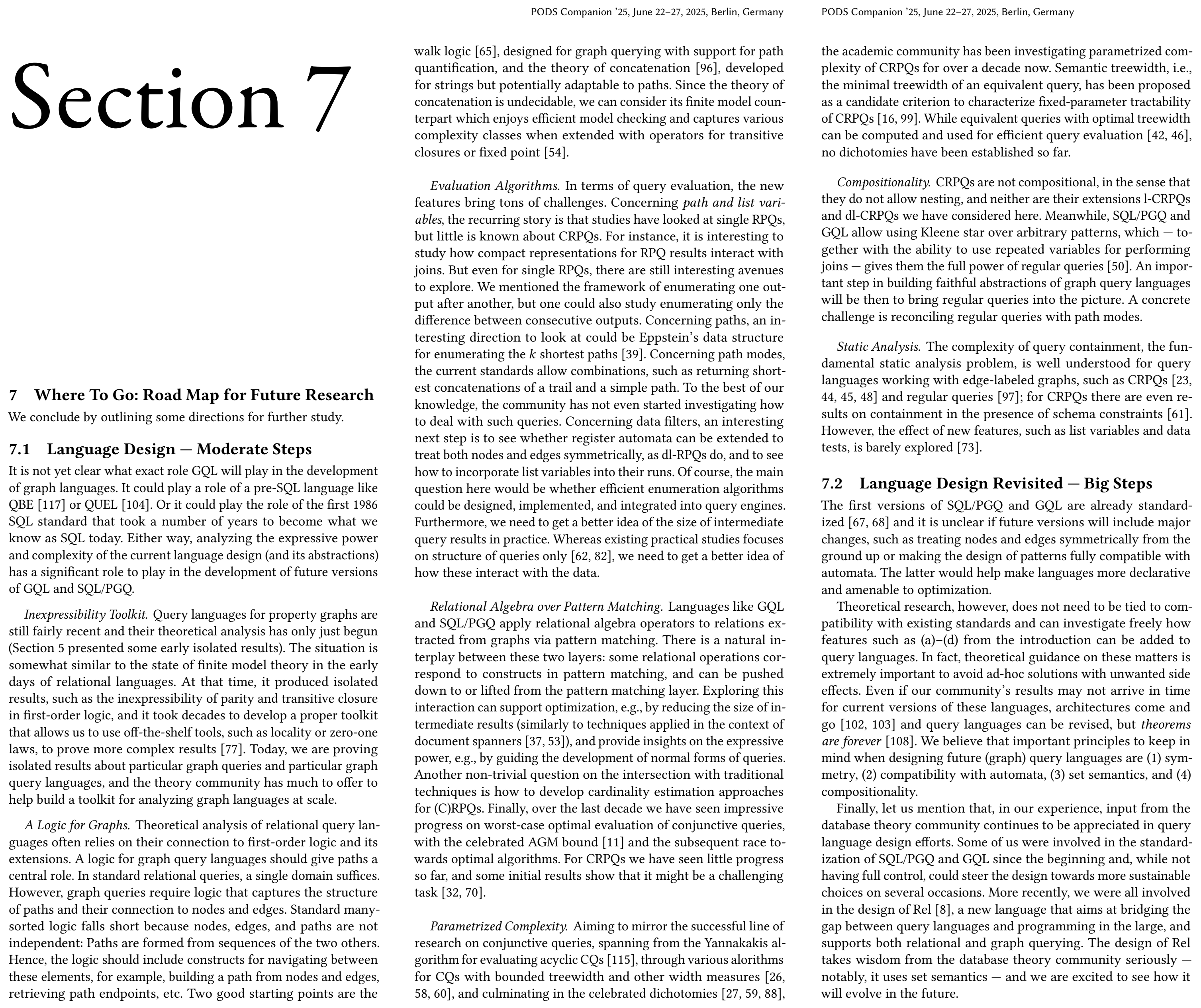
Growing from Theory

- Design and study elegant models
- Understand expressiveness
- Understand complexity
- Use our knowledge of logics, automata,…
- …

Distilling from Practice

- Identify what's good
- Find opportunities for improvement
- Inexpressibility results
- Complexity lower bounds
- …

Section 7



7 Where To Go: Road Map for Future Research

We conclude by outlining some directions for further study.

7.1 Language Design — Moderate Steps

It is not yet clear what exact role GQL will play in the development of graph languages. It could play a role of a pre-SQL language like QBE [117] or QUEL [104]. Or it could play the role of the first 1986 SQL standard that took a number of years to become what we know as SQL today. Either way, analyzing the expressive power and complexity of the current language design (and its abstractions) has a significant role to play in the development of future versions of GQL and SQL/PGQ.

Inexpressibility Toolkit. Query languages for property graphs are still fairly recent and their theoretical analysis has only just begun (Section 5 presented some early isolated results). The situation is somewhat similar to the state of finite model theory in the early days of relational languages. At that time, it produced isolated results, such as the inexpressibility of parity and transitive closure in first-order logic, and it took decades to develop a proper toolkit that allows us to use off-the-shelf tools, such as locality or zero-one laws, to prove more complex results [77]. Today, we are proving isolated results about particular graph queries and particular graph query languages, and the theory community has much to offer to help build a toolkit for analyzing graph languages at scale.

A Logic for Graphs. Theoretical analysis of relational query languages often relies on their connection to first-order logic and its extensions. A logic for graph query languages should give paths a central role. In standard relational queries, a single domain suffices. However, graph queries require logic that captures the structure of paths and their connection to nodes and edges. Standard many-sorted logic falls short because nodes, edges, and paths are not independent: Paths are formed from sequences of the two others. Hence, the logic should include constructs for navigating between these elements, for example, building a path from nodes and edges, retrieving path endpoints, etc. Two good starting points are the walk logic [65], designed for graph querying with support for path quantification, and the theory of concatenation [96], developed for strings but potentially adaptable to paths. Since the theory of concatenation is undecidable, we can consider its finite model counterpart which enjoys efficient model checking and captures various complexity classes when extended with operators for transitive closures or fixed point [54].

Evaluation Algorithms. In terms of query evaluation, the new features bring tons of challenges. Concerning *path and list variables*, the recurring story is that studies have looked at single RPQs, but little is known about CRPQs. For instance, it is interesting to study how compact representations for RPQ results interact with joins. But even for single RPQs, there are still interesting avenues to explore. We mentioned the framework of enumerating one output after another, but one could also study enumerating only the difference between consecutive outputs. Concerning paths, an interesting direction to look at could be Eppstein's data structure for enumerating the k shortest paths [39]. Concerning path modes, the current standards allow combinations, such as returning shortest concatenations of a trail and a simple path. To the best of our knowledge, the community has not even started investigating how to deal with such queries. Concerning data filters, an interesting next step is to see whether register automata can be extended to treat both nodes and edges symmetrically, as dl-RPQs do, and to see how to incorporate list variables into their runs. Of course, the main question here would be whether efficient enumeration algorithms could be designed, implemented, and integrated into query engines. Furthermore, we need to get a better idea of the size of intermediate query results in practice. Whereas existing practical studies focuses on structure of queries only [62, 82], we need to get a better idea of how these interact with the data.

Relational Algebra over Pattern Matching. Languages like GQL and SQL/PGQ apply relational algebra operators to relations extracted from graphs via pattern matching. There is a natural interplay between these two layers: some relational operations correspond to constructs in pattern matching, and can be pushed down to or lifted from the pattern matching layer. Exploring this interaction can support optimization, e.g., by reducing the size of intermediate results (similarly to techniques applied in the context of document spanners [37, 53]), and provide insights on the expressive power, e.g., by guiding the development of normal forms of queries. Another non-trivial question on the intersection with traditional techniques is how to develop cardinality estimation approaches for (C)RPQs. Finally, over the last decade we have seen impressive progress on worst-case optimal evaluation of conjunctive queries, with the celebrated AGM bound [11] and the subsequent race towards optimal algorithms. For CRPQs we have seen little progress so far, and some initial results show that it might be a challenging task [32, 70].

Parametrized Complexity. Aiming to mirror the successful line of research on conjunctive queries, spanning from the Yannakakis algorithm for evaluating acyclic CQs [115], through various alorithms for CQs with bounded treewidth and other width measures [26, 58, 60], and culminating in the celebrated dichotomies [27, 59, 88], the academic community has been investigating parametrized complexity of CRPQs for over a decade now. Semantic treewidth, i.e., the minimal treewidth of an equivalent query, has been proposed as a candidate criterion to characterize fixed-parameter tractability of CRPQs [16, 99]. While equivalent queries with optimal treewidth can be computed and used for efficient query evaluation [42, 46], no dichotomies have been established so far.

Compositionality. CRPQs are not compositional, in the sense that they do not allow nesting, and neither are their extensions l-CRPQs and dl-CRPQs we have considered here. Meanwhile, SQL/PGQ and GQL allow using Kleene star over arbitrary patterns, which — together with the ability to use repeated variables for performing joins — gives them the full power of regular queries [50]. An important step in building faithful abstractions of graph query languages will be then to bring regular queries into the picture. A concrete challenge is reconciling regular queries with path modes.

Static Analysis. The complexity of query containment, the fundamental static analysis problem, is well understood for query languages working with edge-labeled graphs, such as CRPQs [23, 44, 45, 48] and regular queries [97]; for CRPQs there are even results on containment in the presence of schema constraints [61]. However, the effect of new features, such as list variables and data tests, is barely explored [73].

7.2 Language Design Revisited — Big Steps

The first versions of SQL/PGQ and GQL are already standardized [67, 68] and it is unclear if future versions will include major changes, such as treating nodes and edges symmetrically from the ground up or making the design of patterns fully compatible with automata. The latter would help make languages more declarative and amenable to optimization.

Theoretical research, however, does not need to be tied to compatibility with existing standards and can investigate freely how features such as (a)–(d) from the introduction can be added to query languages. In fact, theoretical guidance on these matters is extremely important to avoid ad-hoc solutions with unwanted side effects. Even if our community's results may not arrive in time for current versions of these languages, architectures come and go [102, 103] and query languages can be revised, but *theorems are forever* [108]. We believe that important principles to keep in mind when designing future (graph) query languages are (1) symmetry, (2) compatibility with automata, (3) set semantics, and (4) compositionality.

Finally, let us mention that, in our experience, input from the database theory community continues to be appreciated in query language design efforts. Some of us were involved in the standardization of SQL/PGQ and GQL since the beginning and, while not having full control, could steer the design towards more sustainable choices on several occasions. More recently, we were all involved in the design of Rel [8], a new language that aims at bridging the gap between query languages and programming in the large, and supports both relational and graph querying. The design of Rel takes wisdom from the database theory community seriously — notably, it uses set semantics — and we are excited to see how it will evolve in the future.

Research Agenda for Graph Query Languages

Growing from Theory

- Design and study elegant models
- Understand expressiveness
- Understand complexity
- Use our knowledge of logics, automata,…
- …

Distilling from Practice

- Identify what's good
- Find opportunities for improvement
- Inexpressibility results
- Complexity lower bounds
- …

Build Systems

- Implement our ideas
- Make them work efficiently
- Or learn why this is impossible

[M., Niewerth, Popp, Rojas, Vansummeren, Vrgoc VLDB'23]
[Farias, M., Rojas, Vrgoc ISWC'24]
[…]

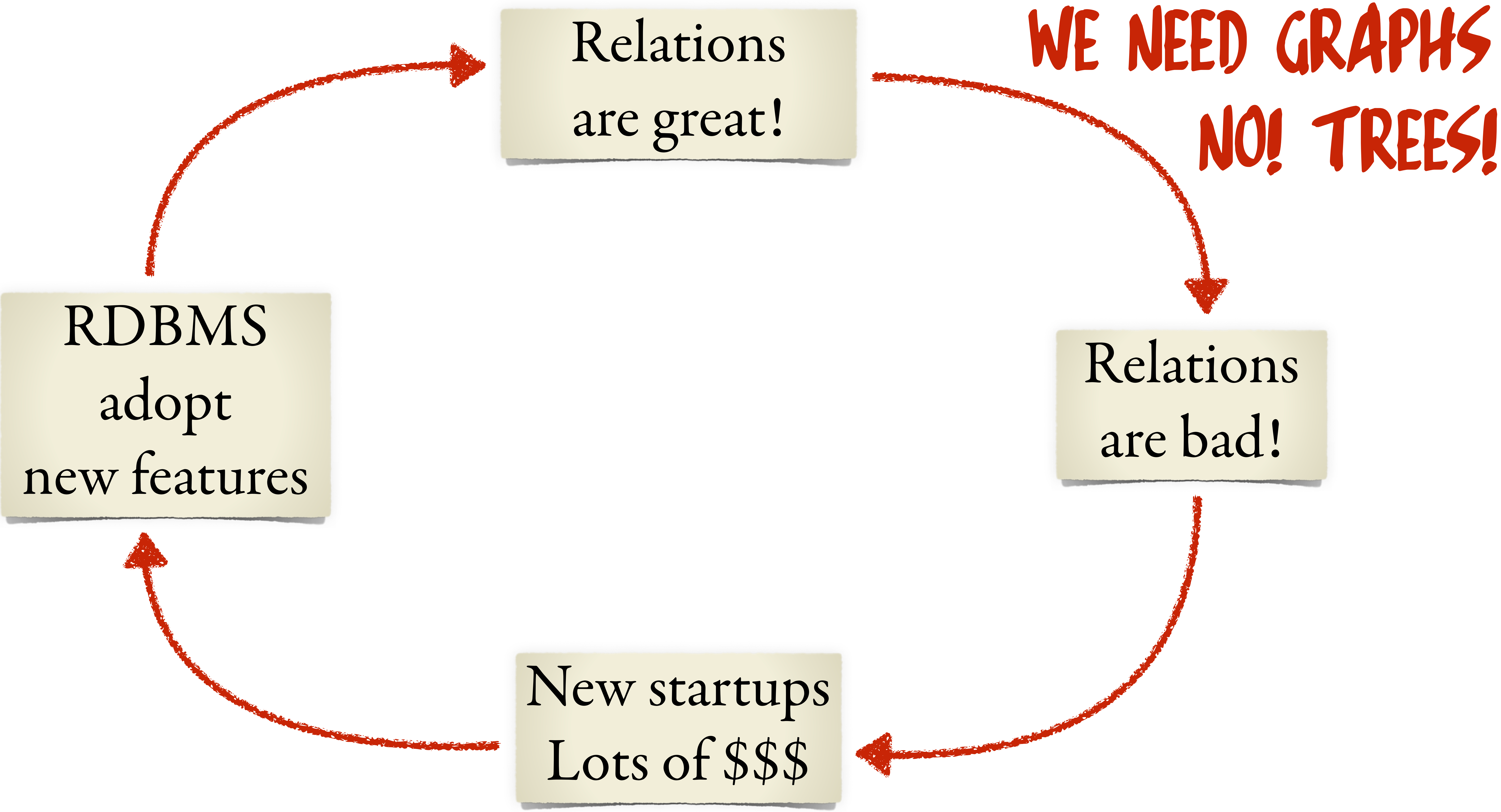
Graphs vs Relations

What Goes Around Comes Around?

Big Questions Again?

Sets? Bags?

What Goes Around Comes Around



Relations are great!

WE NEED GRAPHS

NO! TREES!

Relations are bad!

New startups Lots of $$$

RDBMS adopt new features

Slide copied & adapted from Andy Pavlo

Graph Query Languages

Observation

For the query language,
the underlying DB architecture
(graph, relational,...)
is irrelevant

Cypher / GQL	⇝ implemented in	neo4j	graph native
	⇝ implemented in	kùzu	graph native
SQL/PGQ	⇝ implemented in	ORACLE	relational
	⇝ implemented in	DuckDB	relational
↳ can be translated into Rel	⇝ implemented in	RelationalAI	relational + set semantics!

Wrapping Up

Wrapping Up

New graph query languages add

(a) handling of nodes and edges
(b) path and list variables
(c) path modes
(d) data filters
to Conjunctive Regular Path Queries

Their design in the standard(s) isn't smooth yet
⇝ we have work to do

How would we design features like this in a graph query language?

Our proposal:

- l-CRPQs
- dl-CRPQs

(see paper)

What can you do?

- Study these new CRPQs
- Come up with your own design?
- Study GQL & SQL/PGQ
- Do RPQs in Datalog
- Prove that sets are better than bags
- Prove that bags are better than sets
- Solve Automatic Programming
- ...

Thank you!

Happy to talk to you in the break!