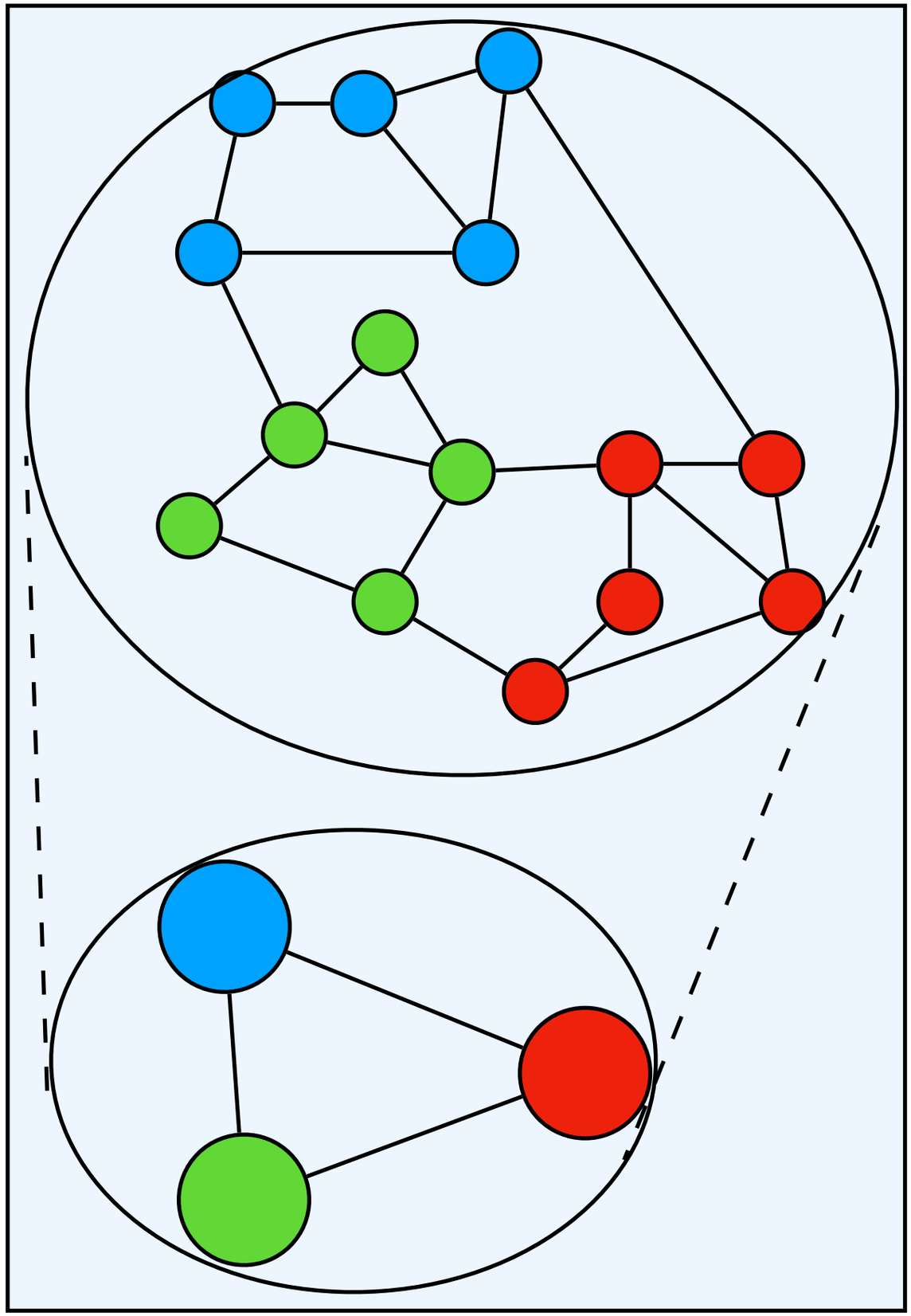
Flexible and Expressive Typed Path Patterns for GQL

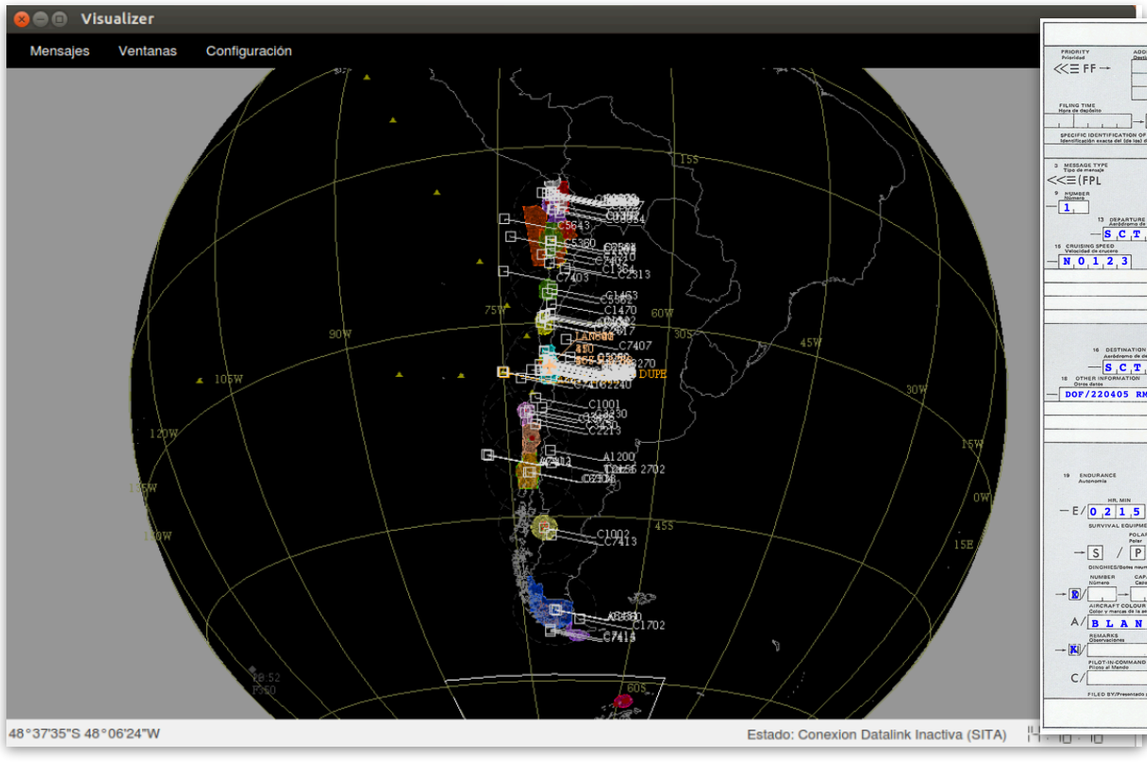
(Helping the AI make less mistakes)

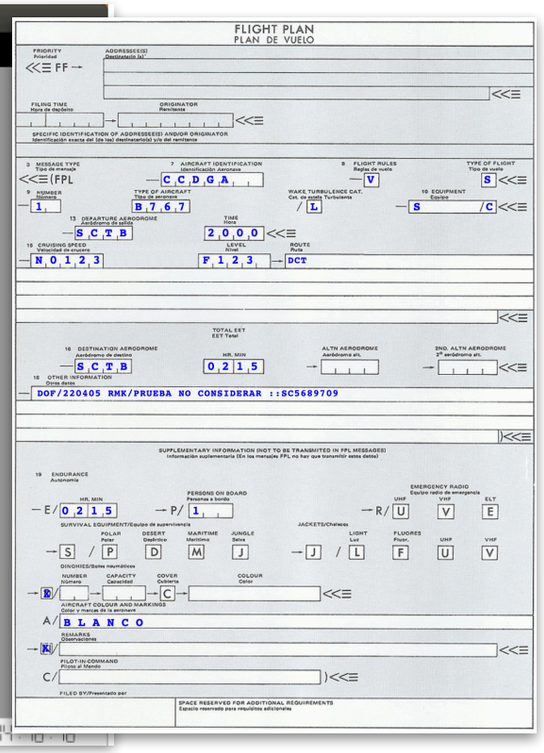
Matías Toro
University of Chile

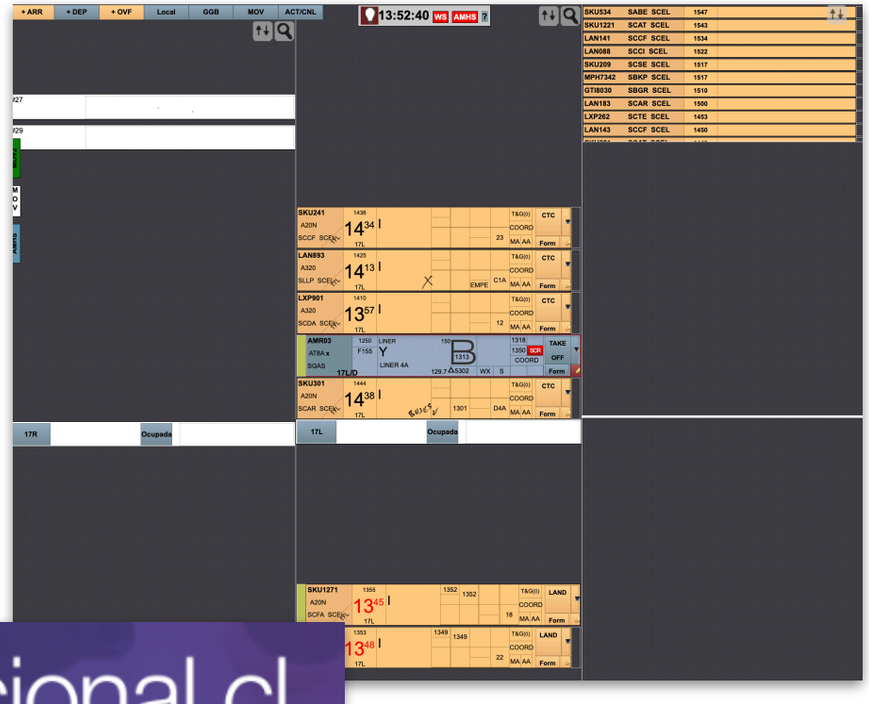


Who I am

My name is **Matías Toro**. Before I become an assistant professor, I have worked for **14 years** in the aeronautical and educational fields designing and developing software.





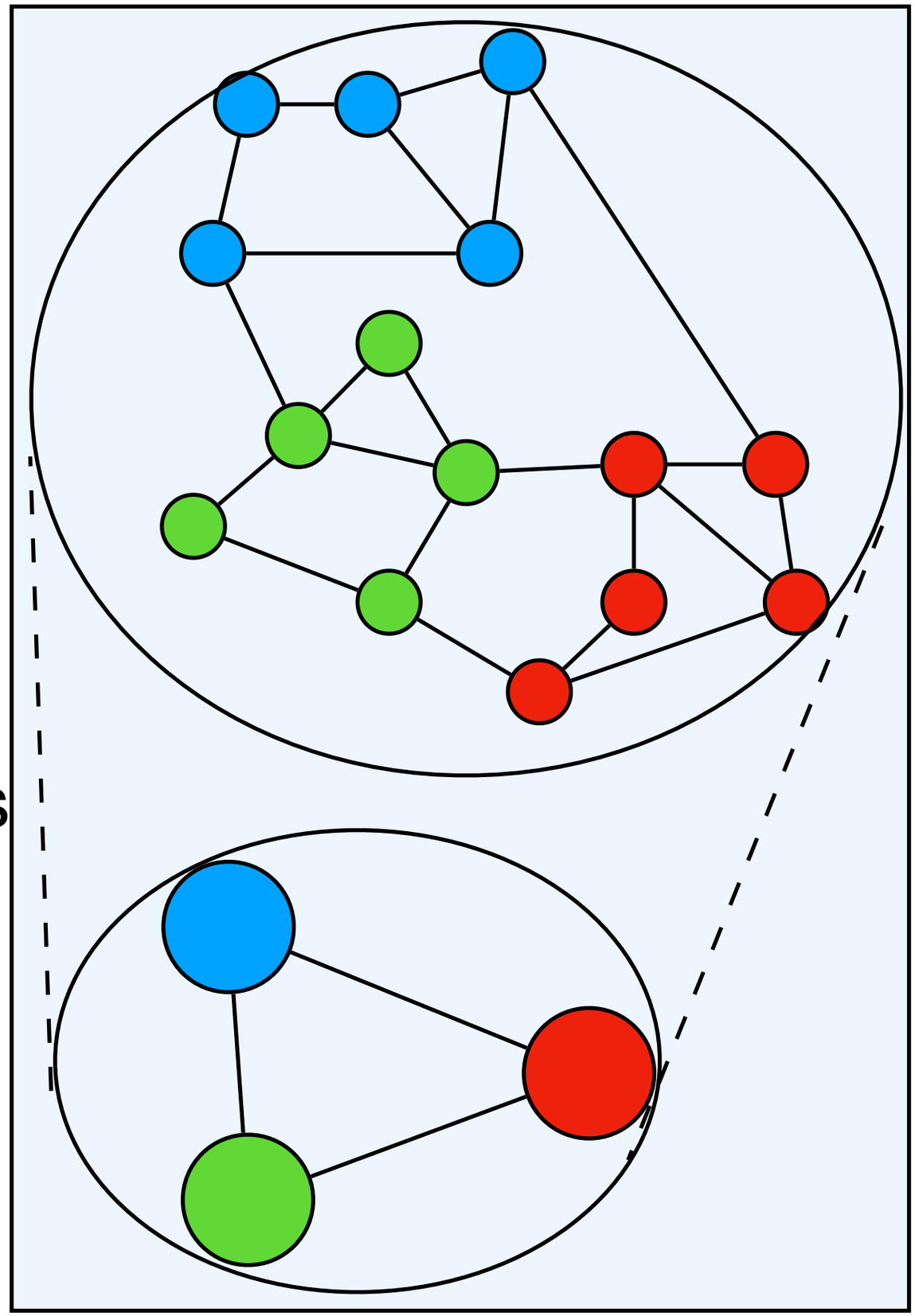


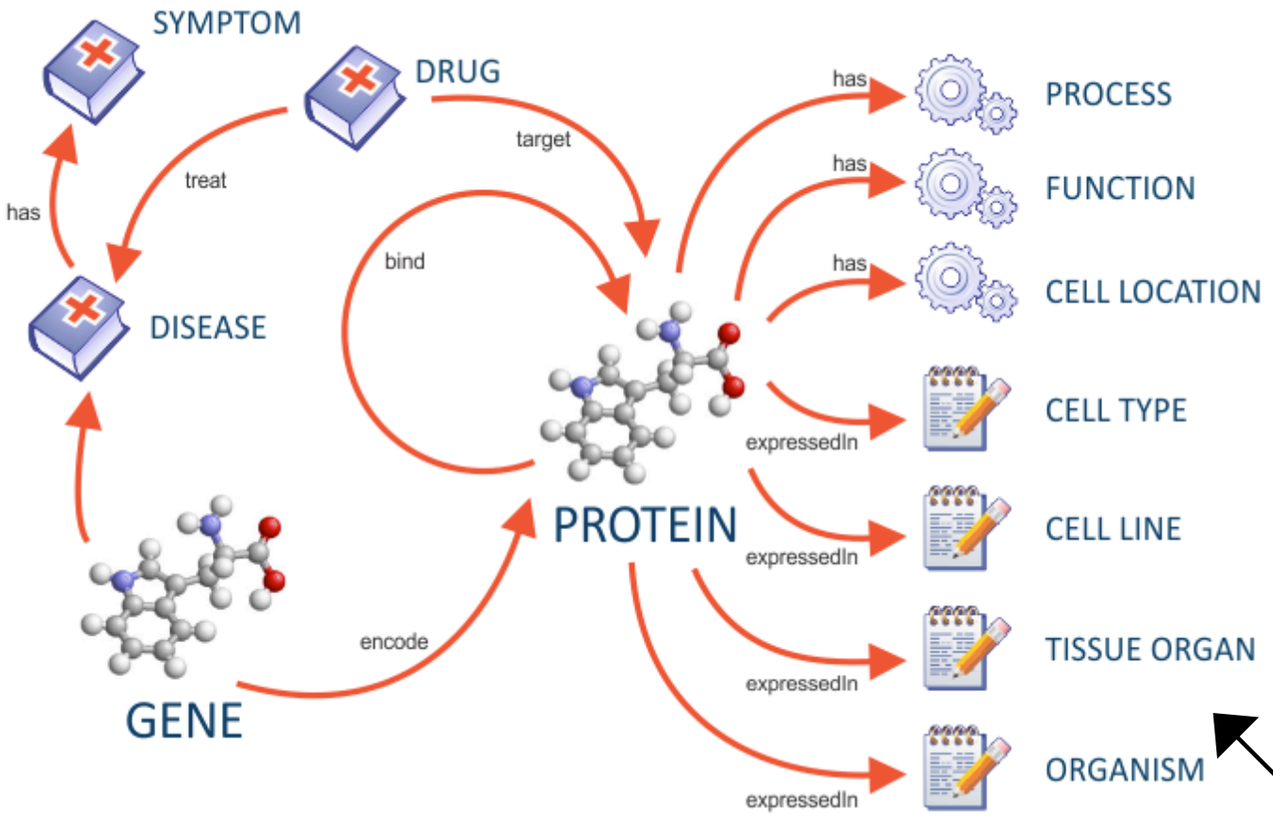
What I do

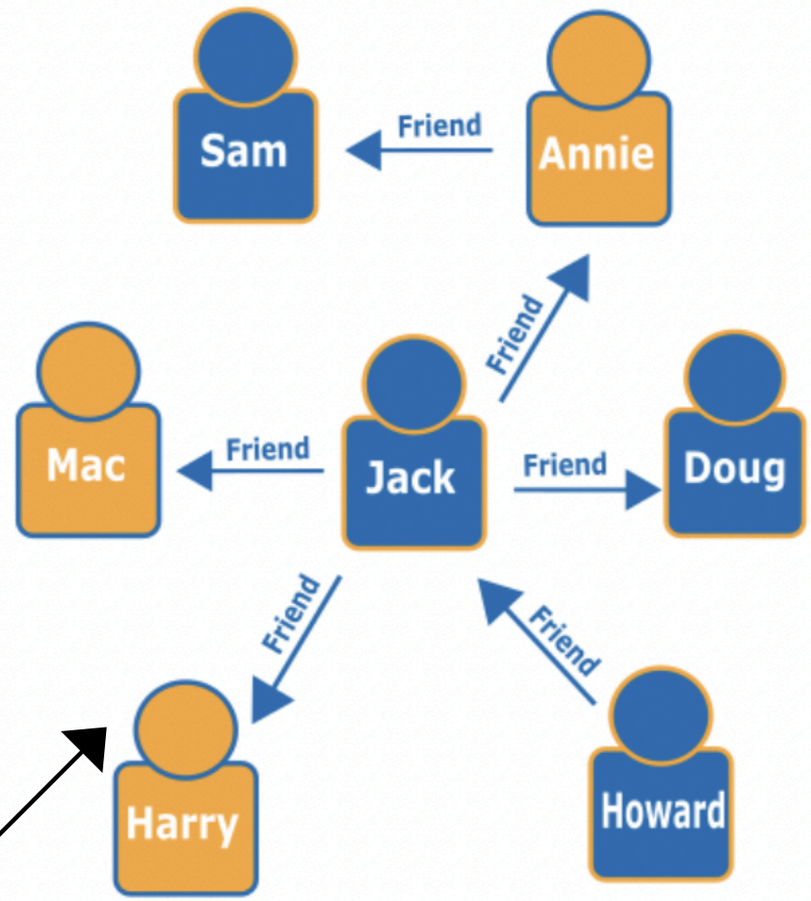
My research sits at the intersection of **gradual typing** and language design for **safety guarantees** (confidentiality/ differential privacy). More recently, I've also been exploring how to bring type system foundations to database query languages.

Flexible and Expressive Typed Path Patterns for GQL

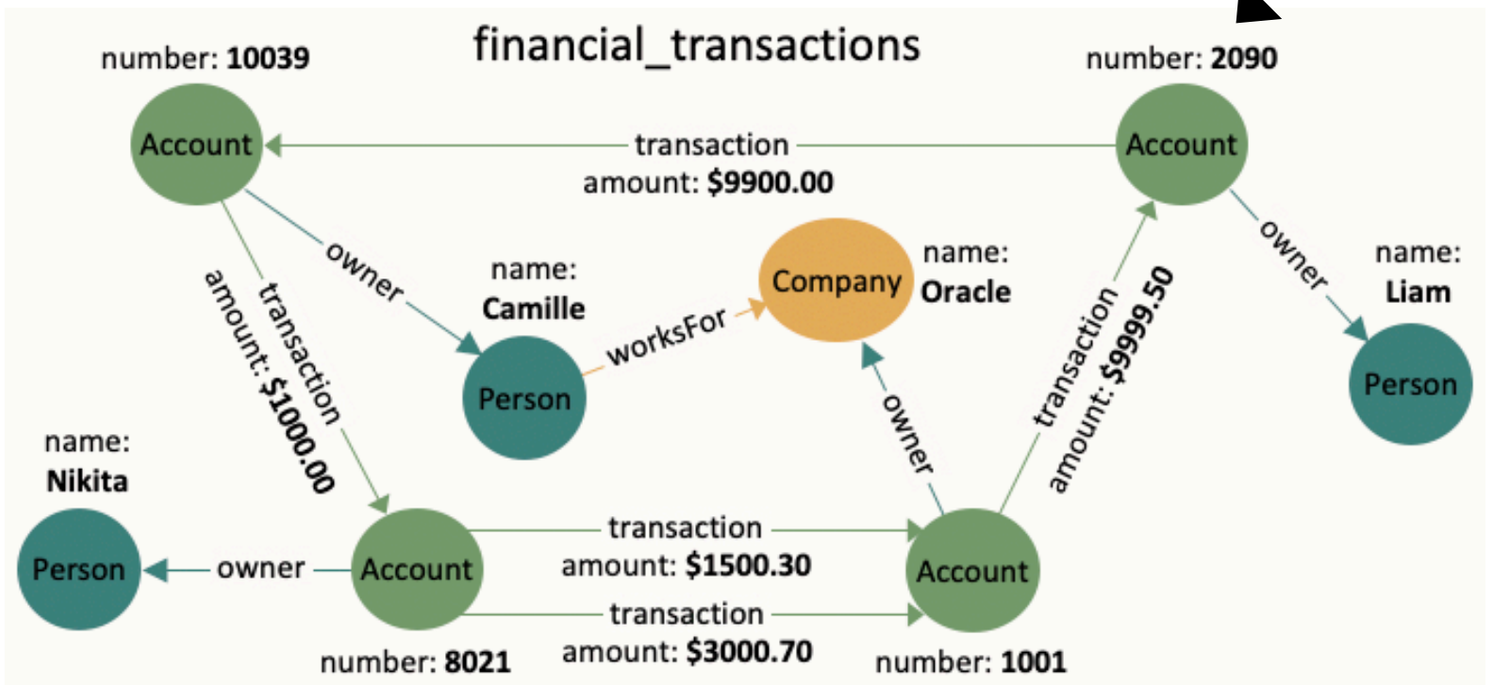
Wenjia Ye, **Matías Toro**, Tomás Díaz, Bruno C.d.S. Oliveira, Manuel Rigger, Claudio Gutierrez, Domagoj Vrgoč

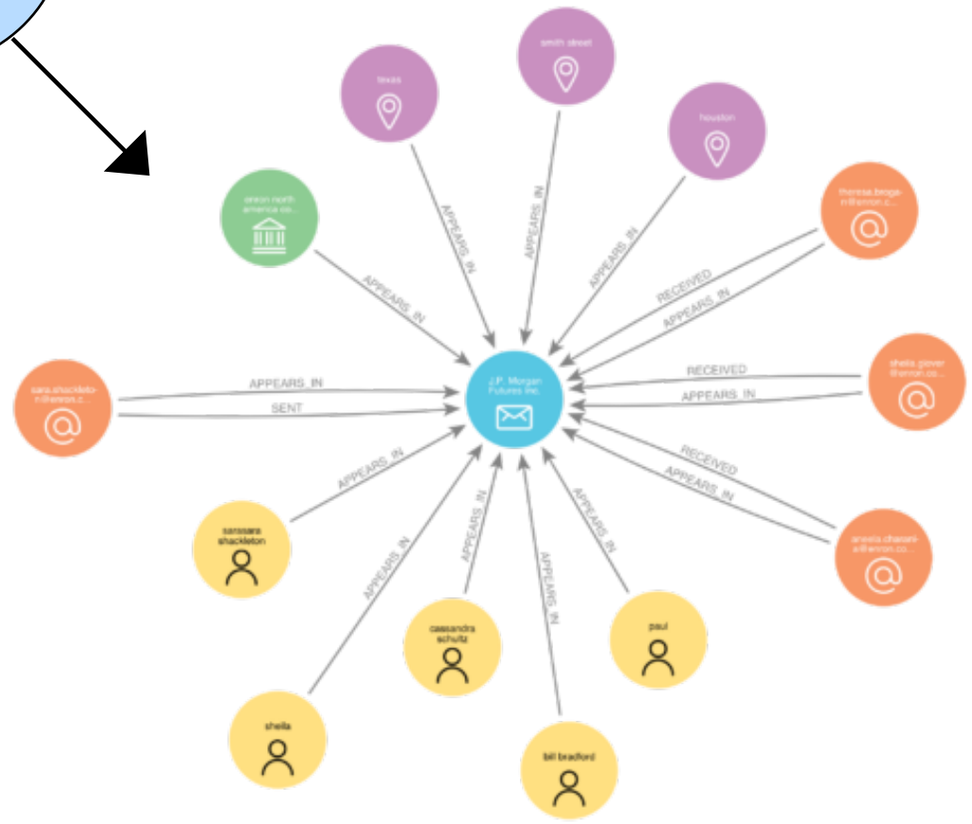






Graph Database



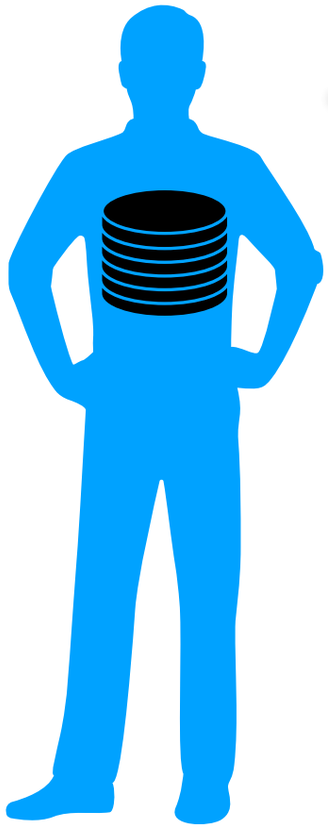


GQL

GQL (Graph Query Language) is a **new (2024) ISO standard** for **graph query languages** that is being developed to be the counterpart of SQL for graph databases.

GPC

Denotational semantics



What do you think about the type system?

Mmmh… (laughs in spanish)

GPC: A Pattern Calculus for Property Graphs

Nadime Francis
LIGM, Université Gustave Eiffel, CNRS
Champs-sur-Marne, France
nadime.francis@univ-eiffel.fr

Amélie Gheerbrant
Université Paris Cité, CNRS, IRIF
Paris, France
amelie@irif.fr

Paolo Guagliardo
University of Edinburgh
Edinburgh, UK
paolo.guagliardo@ed.ac.uk

d Libkin*
of Edinburgh
urgh, UK
tionalAI
s, France
libk.in

Victor Marsault
LIGM, Université Gustave Eiffel, CNRS
Champs-sur-Marne, France
victor.marsault@univ-eiffel.fr

Wim Martens
University of Bayreuth
Bayreuth, Germany
wim.martens@uni-bayreuth.de

Murlak
University of Warsaw
Warsaw, Poland
f.murlak@uw.edu.pl

Liat Peterfreund
LIGM, Université Gustave Eiffel, CNRS
Champs-sur-Marne, France
liat.peterfreund@univ-eiffel.fr

Alexandra Rogova†
Université Paris Cité, CNRS, IRIF
Paris, France
rogova@irif.fr

Domagoj Vrgoč
PUC Chile & IMFD
Santiago de Chile, Chile
vrdomagoj@uc.cl

ABSTRACT

The development of practical query languages for graph databases runs well ahead of the underlying theory. The ISO committee in charge of database query languages is currently developing a new standard called *Graph Query Language* (GQL) as well as an extension of the SQL Standard for querying property graphs represented by a schema, called SQL/PGQ. The main component of both is standards. similar r years. rd speci-

CCS CONCEPTS

• **Information systems** → **Graph-based database models**; **Query languages**.

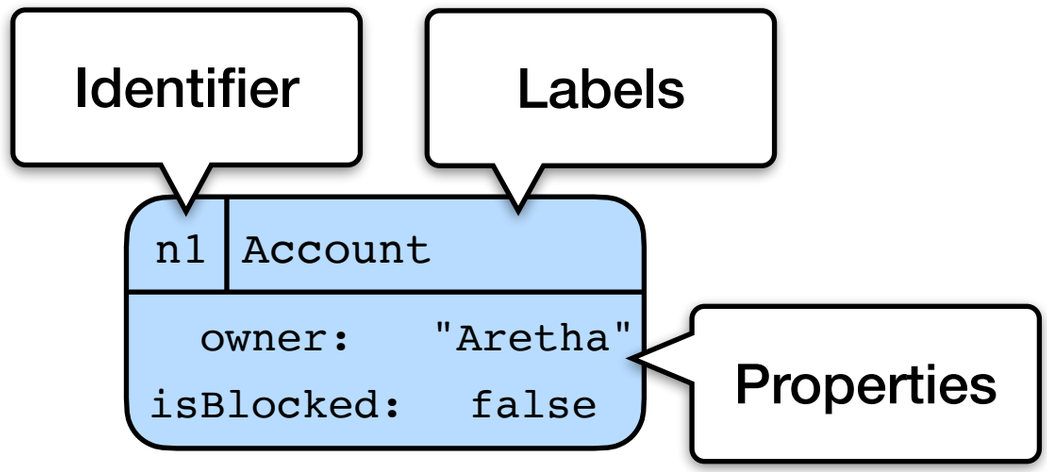
KEYWORDS

graph databases, graph query languages, GQL, SQL/PGQ, pattern matching, syntax and semantics, expressive power, complexity, type systems

ACM Reference Format:

Nadime Francis, Amélie Gheerbrant, Paolo Guagliardo, Leonid Libkin, Victor

Nodes



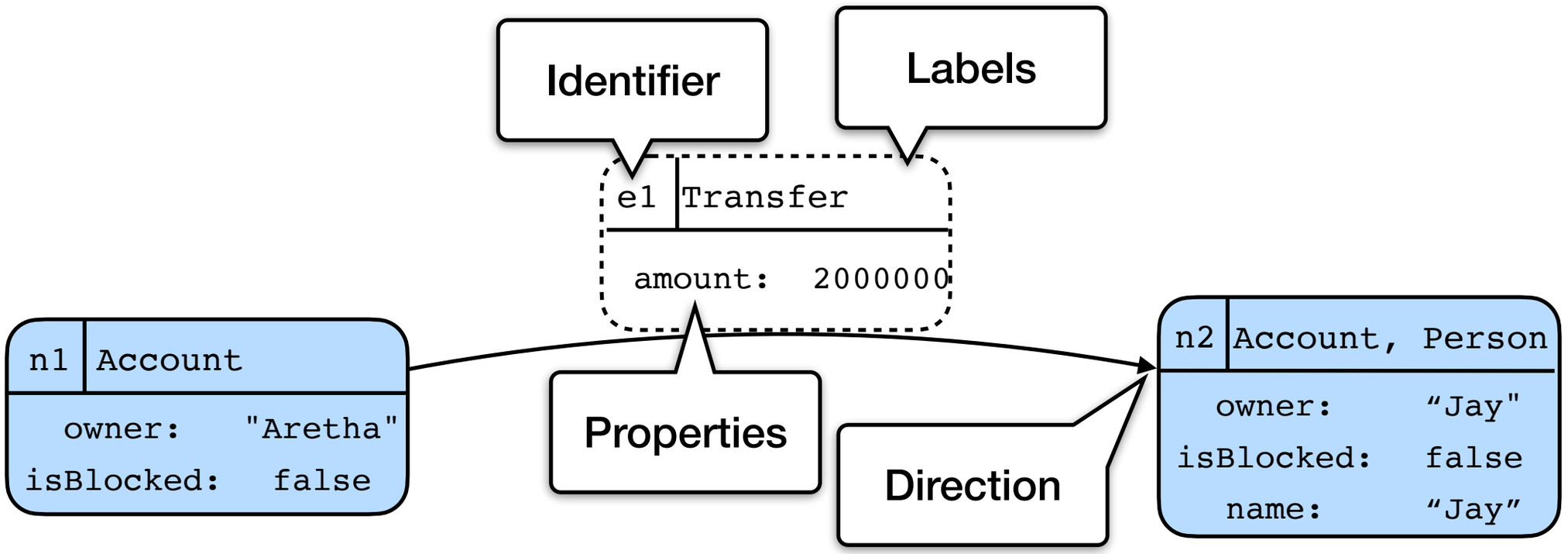
Data model G

$$\mathcal{N}_{id} = \{n_1\}$$

$$\mathcal{L} = \{n_1 : \{\texttt{Account}\}\}$$

$$\delta = \{n_1 : \{\texttt{owner} : \text{"Aretha"}, \texttt{isBlocked} : \texttt{false}\}\}$$

Edges



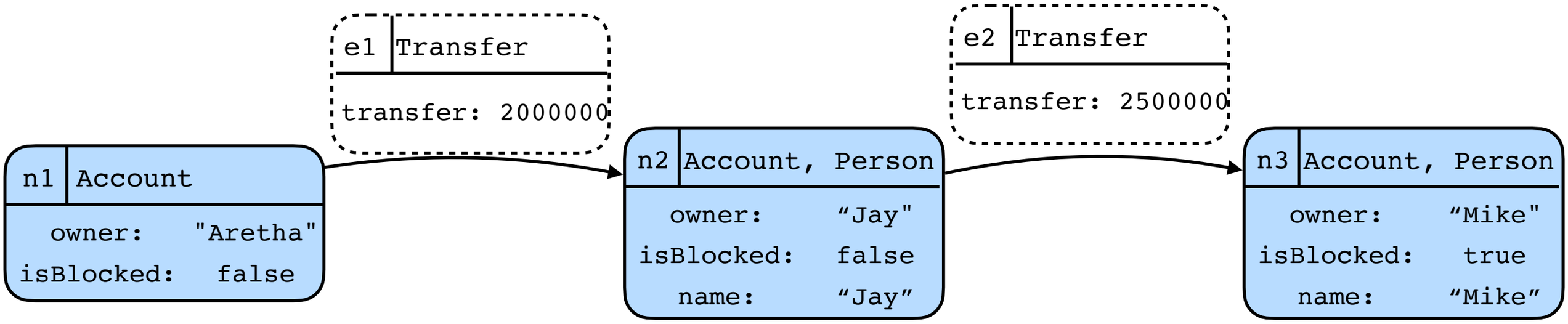
Data model G

$\mathcal{N}_{id} = \{n_1, n_2\}$ $\mathcal{E}_d = \{e_1\}$

$\mathcal{L} = \{n_1 : \{\texttt{Account}\}, n_2 : \{\texttt{Account}, \texttt{Person}\}, e_1 : \{\texttt{Transfer}\}\}$

$\delta = \{n_1 : \{\texttt{owner} : \text{"Aretha"}, \texttt{isBlocked} : \texttt{false}\}, \ldots\}$

Path Queries

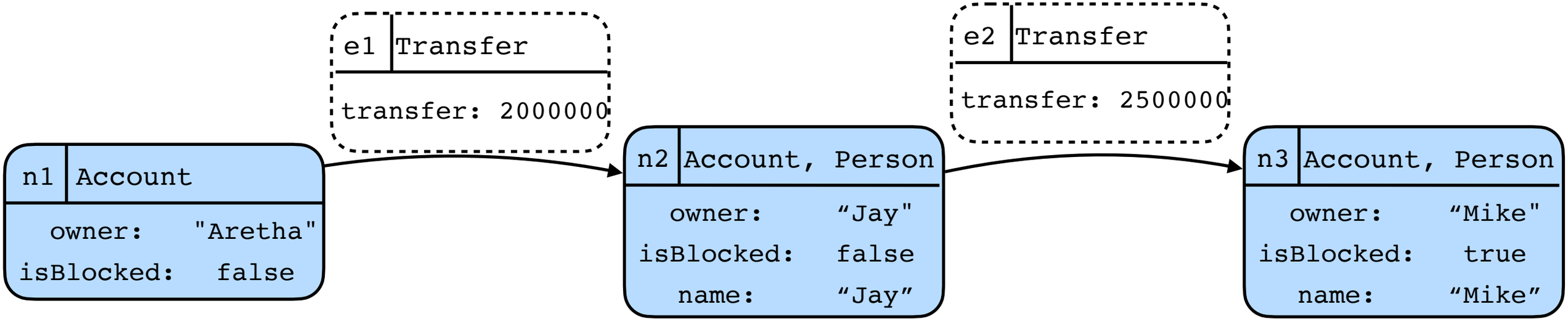



```
(x) -[z:Transfer WHERE z.amount>1000000]-> (y WHERE y.isBlocked=true)
```

$$[\![\pi]\!]_G = \{\overline{(p, \mu)}\}$$

path	x	y	z
…	…	…	…
…	…	…	…

Path Queries

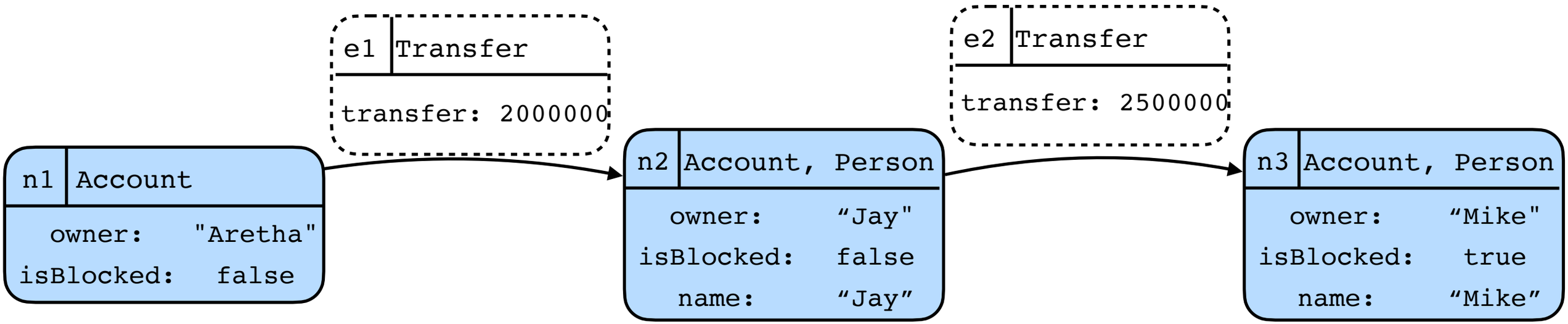


Match any node.
Store the result in `x`

```
(x)-[z:Transfer WHERE z.amount>1000000]-> (y WHERE y.isBlocked=true)
```

path	x
n1	n1
n2	n2
n3	n3

Path Queries



Match any node.
Store the result in `x`

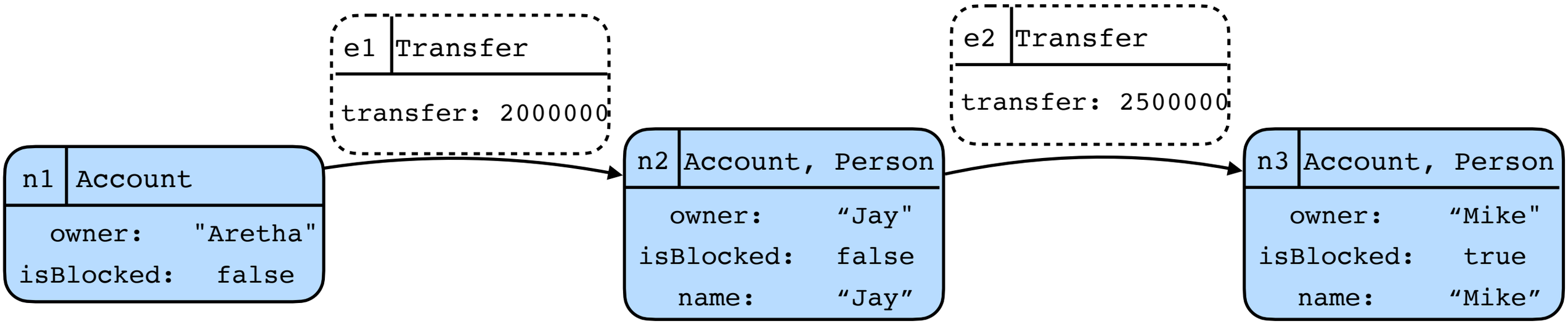
Match directional edges that has label `Transfer` and the property `amount` is bigger than 1M.
Store the result in `z`.

```
(x) -[z:Transfer WHERE z.amount>1000000]-> (y WHERE y.isBlocked=true)
```

path	x
n1	n1
n2	n2
n3	n3

path	z
n1 e1 n2	e1
n2 e2 n3	e2

Path Queries



Match any node.
Store the result in `x`

Match directional edges that has label `Transfer` and the property `amount` is bigger than 1M.
Store the result in `z`.

Match any node such that the property `isBlocked` is `true`.
Store the result in `y`

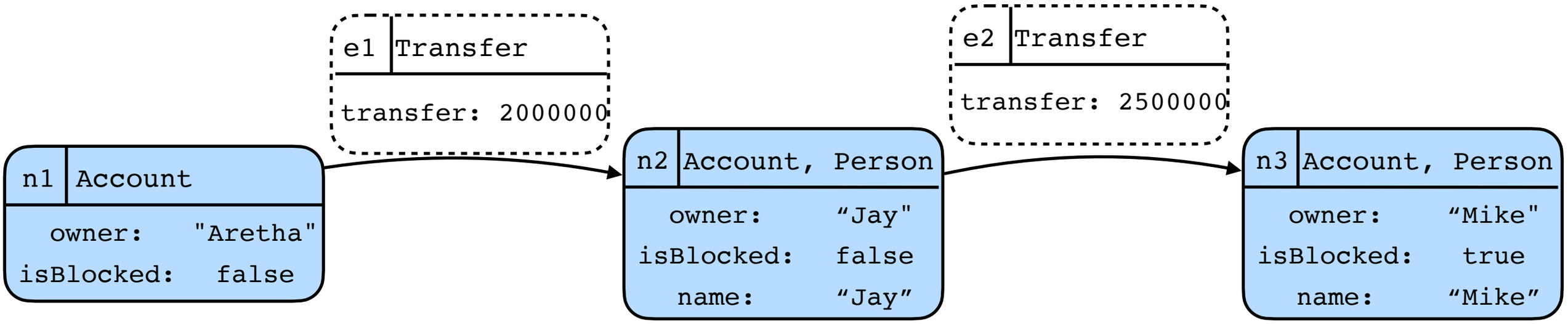
```
(x)-[z:Transfer WHERE z.amount>1000000]->(y WHERE y.isBlocked=true)
```

path	x
n1	n1
n2	n2
n3	n3

path	z
n1 e1 n2	e1
n2 e2 n3	e2

path	y
n3	n3

Path Queries



Match any node.
Store the result in `x`

Match directional edges that has label `Transfer` and the property `amount` is bigger than 1M.
Store the result in `z`.

Match any node such that the property `isBlocked` is `true`.
Store the result in `y`

```
(x)-[z:Transfer WHERE z.amount>1000000]->(y WHERE y.isBlocked=true)
```

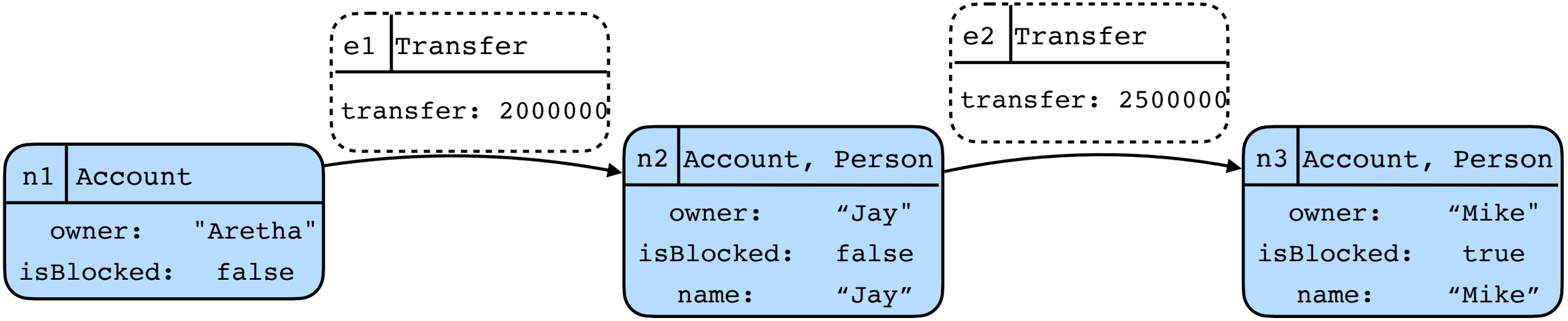
Join the results (by paths) if possible!

path	x
n1	n1
n2	n2
n3	n3

path	z
n1 e1 n2	e1
n2 e2 n3	e2

path	y
n3	n3

Path Queries




```
(x) -[z:Transfer WHERE z.amount>1000000]-> (y WHERE y.isBlocked=true)
```

Result

path	x	y	z
n2 e2 n3	n2	n3	e2

Path patterns

Edge pattern

Concatenation pattern

Repeated pattern

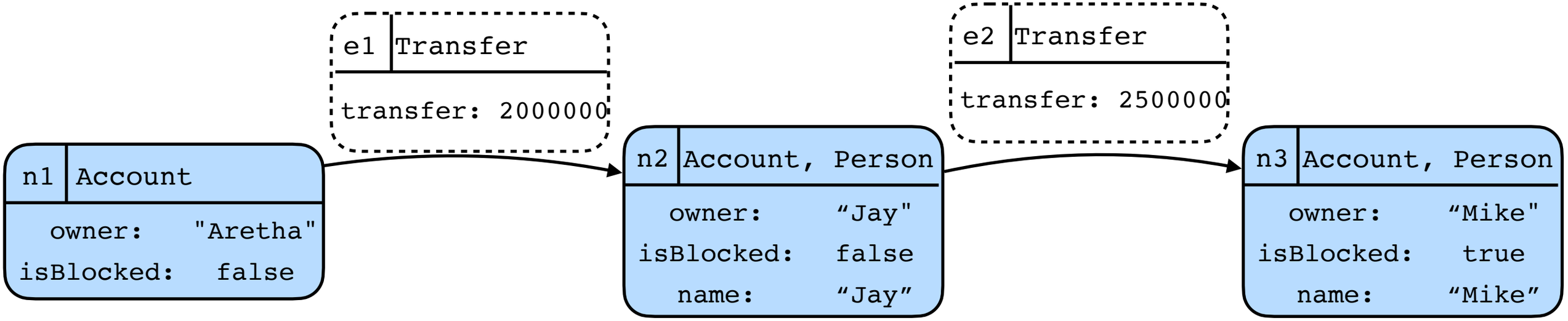
$$\pi ::= (d) \mid \overset{d}{\Longleftrightarrow} \mid \pi + \pi \mid \pi\,\pi \mid \pi_{\langle t \rangle} \mid \pi^{l..u}$$

Node pattern

Union pattern

Conditioned pattern

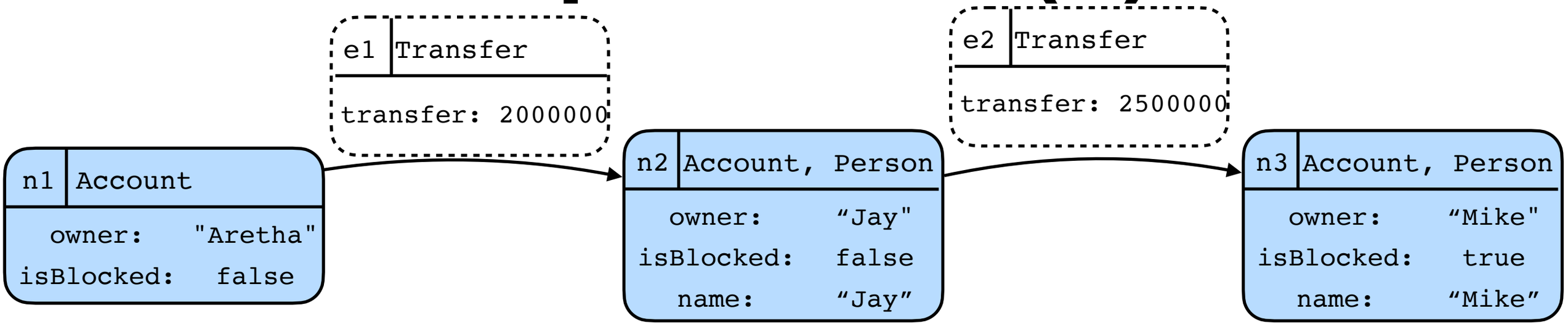
Union



$$(x : \text{Account}) + (y : \text{Person})$$

PATH	X	Y
n1	n1	Nothing
n2	n2	Nothing
n3	n3	Nothing
n2	Nothing	n2
n3	Nothing	n3

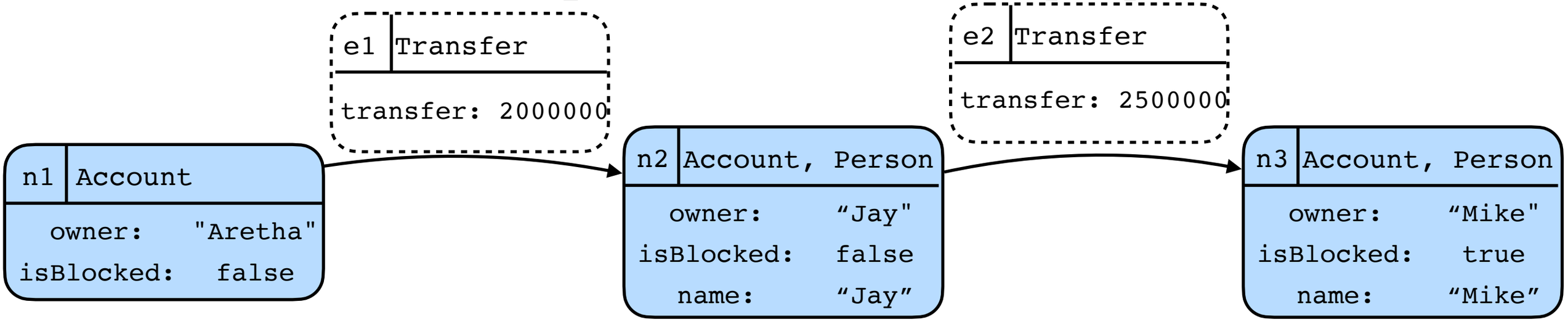
Repetition (1)



$$x \xrightarrow{1..2}$$

PATH	X
n1 e2 n3	[e2]
n3 e3 n2	[e3]
n1 e2 n3 e3 n2	[e2, e3]

Repetition (2)



$$\left(\overset{x}{\rightarrow}^{1..2} \right)^{1..2}$$

PATH	X
n1 e2 n3	[[e2]]
n3 e3 n2	[[e3]]
n1 e2 n3 e3 n2	[[e2, e3]]
n1 e2 n3 e3 n2	[[e2], [e3]]

Observations

- A query can ”go wrong”.
- GQL ISO standard defines types for data, but not formal type system for validating queries.
- GQL ISO standard is written in a mix of prose and pseudocode, marking it hard to interpret and apply in a rigorous, research driven manner.

What can go wrong?

What can go wrong?

Incompatible shapes

```
(x) -[x]->
```

According to the ISO this is syntactically invalid

Validated by GPC!

What can go wrong?

Incompatible shapes

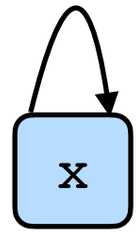
```
(x) -[x]->
```

According to the ISO this is syntactically invalid

ISO: this is “fine”, but the result will be always empty!

Incompatible properties

```
(x WHERE x.isBlocked is bool) -> (x WHERE x.isBlocked is str)
```



What can go wrong?

Incompatible shapes

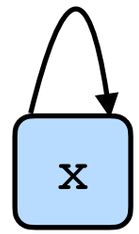
```
(x) -[x]->
```

According to the ISO this is syntactically invalid

ISO: this is “fine”, but the result will be always empty!

Incompatible properties

```
(x WHERE x.isBlocked is bool) -> (x WHERE x.isBlocked is str)
```



Nonexistent attributes

```
(x WHERE x.amout>0)
```

ISO: it depends. It might crash or return empty result.

What can go wrong?

Incompatible shapes

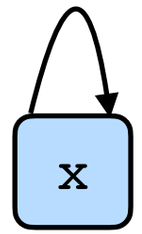
```
(x) -[x]->
```

According to the ISO this is syntactically invalid

Incompatible properties

```
(x WHERE x.isBlocked is bool) -> (x WHERE x.isBlocked is str)
```

ISO: this is “fine”, but the result will be always empty!



Nonexistent attributes

```
(x WHERE x.amout>0)
```

ISO: it depends. It might crash or return empty result.

Wrong usage of attributes

```
(x WHERE x.isBlocked>0)
```

ISO: it depends. It might crash or return empty result.

What can go wrong?

Incompatible shapes

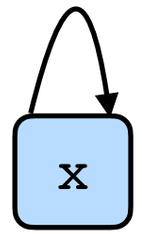
```
(x) -[x]->
```

According to the ISO this is syntactically invalid

Incompatible properties

```
(x WHERE x.isBlocked is bool) -> (x WHERE x.isBlocked is str)
```

ISO: this is “fine”, but the result will be always empty!



Nonexistent attributes

```
(x WHERE x.amout>0)
```

ISO: it depends. It might crash or return empty result.

Wrong usage of attributes

```
(x WHERE x.isBlocked>0)
```

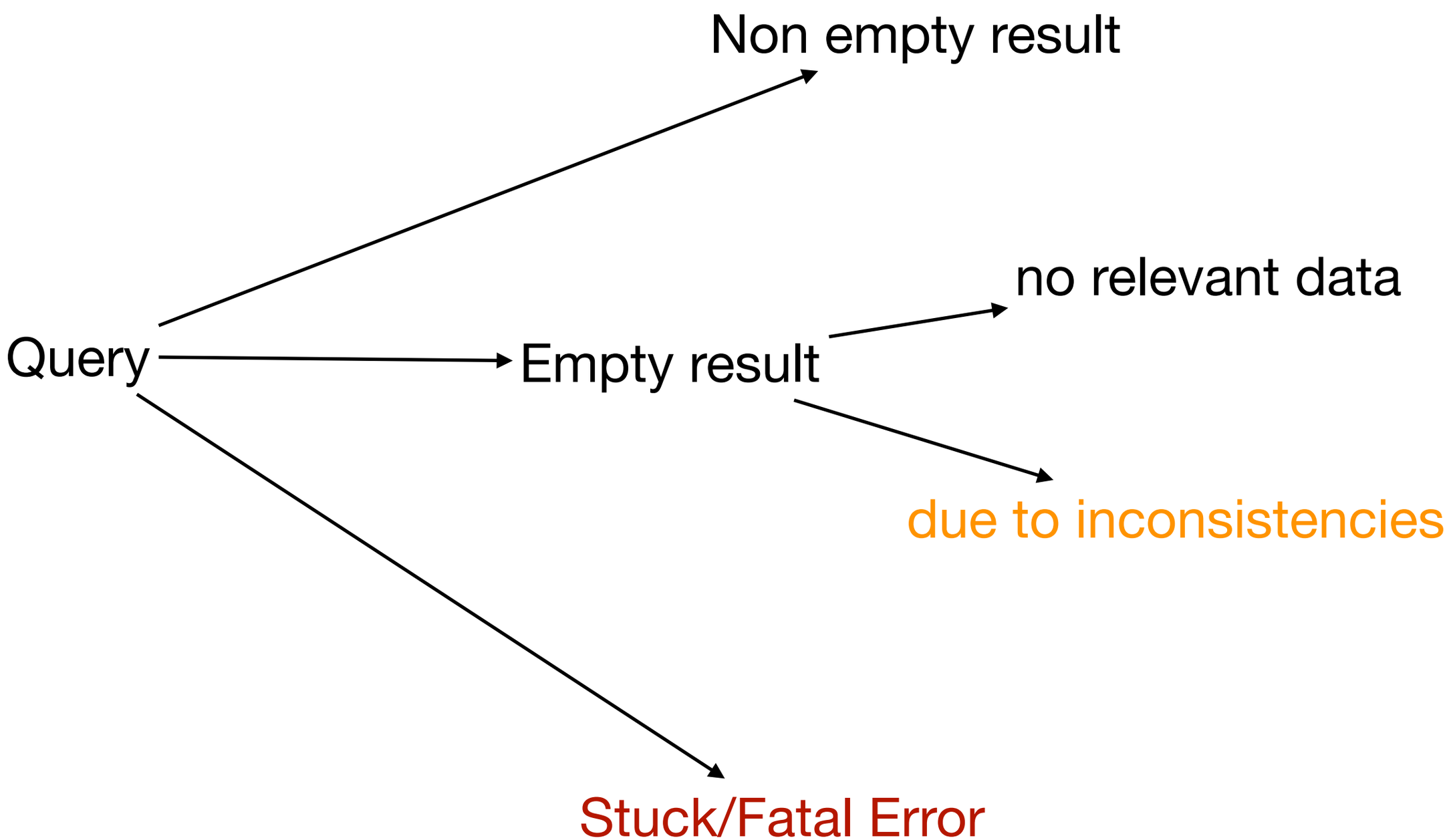
ISO: it depends. It might crash or return empty result.

Unbound variables

```
(y WHERE x.isBlocked=true)
```

Fatal error/stuck execution!

The execution of a query



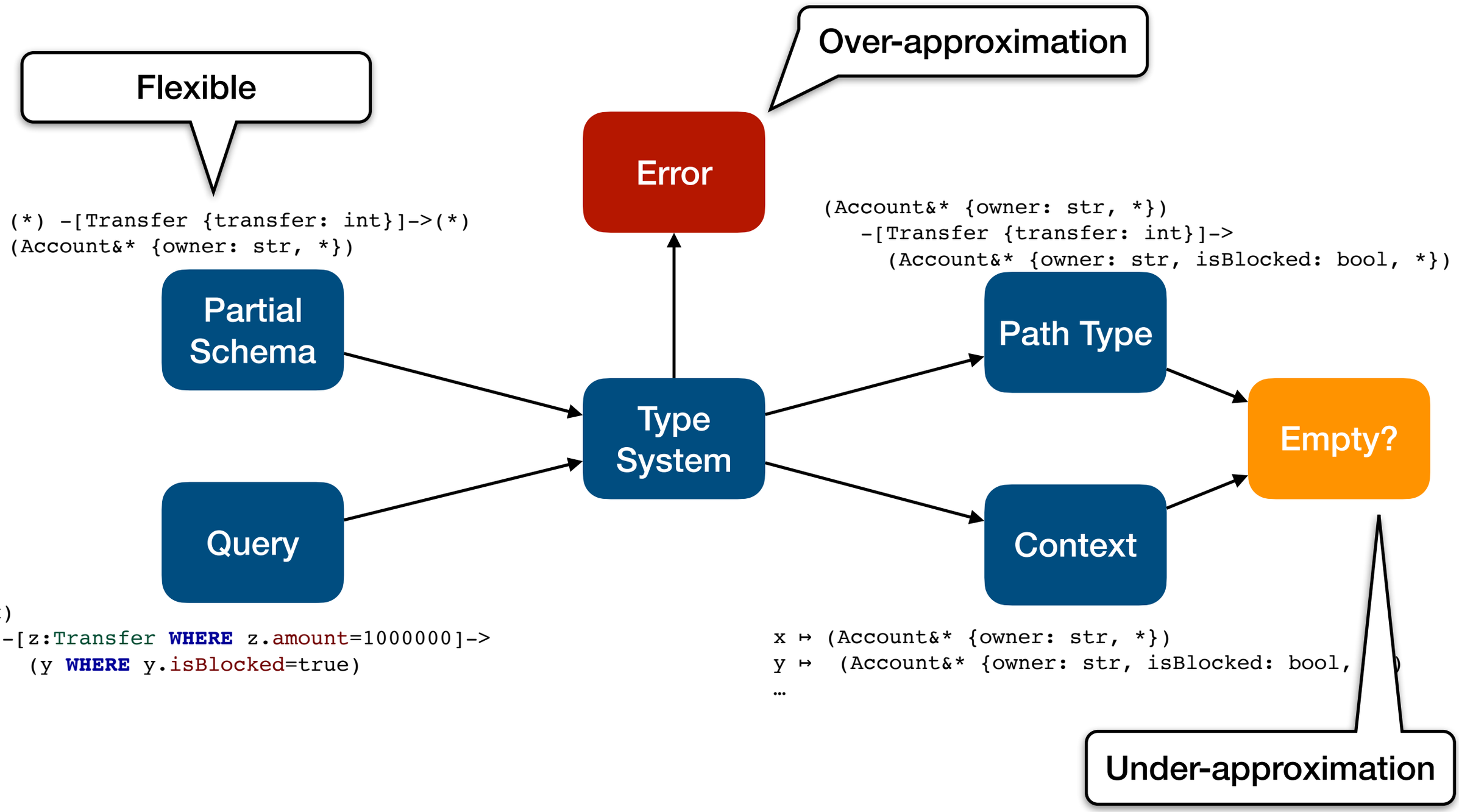
Goals

- Provide a static type system for GQL that:
  - **Under-approximate definitive empty results**. Issues warnings for programs that will **definitively** return an empty result.
  - **Over-approximate stuck evaluations**. Rejects programs that **may potentially** get stuck or abort during query execution.
  - **Flexible**. Starts with a schema that has no types and allows the *gradual* addition of types, progressively increasing the precision of the type system.
  - (And typing must be **faster** than running the actual query)

Our Approach

- Flexible Path Pattern Calculus (FPPC)
  - a formal model of the path pattern language of GQL.
  - extending ISO GQL with new syntax for filtering by property types.
  - use of gradual typing [Siek and Taha 2006]: both the schema and path pattern language with imprecise type information by introducing an unknown type (★).

Overview: FPPC



FPPC in action

Query

Typecheck | Run | Ctrl+Enter to run

```
1 (x : Teacher)
2   -[: Likes ]-> (: Comment)
3   -[: Author ]-> (: Student )
4   -[y: Knows WHERE y.since < 2021]- (x)
```

Type Check OK

Path Type: ⟪((Person & Teacher) {name:str,status:str}⟫ - ⟪(Comment {content:str,status:bool}⟫ - ⟪((Person & Student) {name:str,status:int}⟫ - ⟪((Person & Teacher) {name:str,status:str}⟫

Type Environment:

y ↦ ⟪((Person & Teacher) {name:str,status:str}⟫-[Knows {since:int}]-⟪((Person & Student) {name:str,status:int}⟫

x ↦ ⟪((Person & Teacher) {name:str,status:str}⟫

Results (1 rows)

PATH	X	Y
n1 e2 n3 e3 n2 e1 n1	n1	e1

FPPC in action

Query

Typecheck | Run | Ctrl+Enter to run

```
1 (x: {status: bool} WHERE x.status > 0) -[:Author WHERE y.foo]->
```

Type Check FAIL

Path Type: ⊥

Type Environment:

```
x ↦ ⟪Comment {content:str,status:bool}⟫
```

Warnings

```
File "...", line 202 in check_expr:
  Binop > between types bool and int is not defined
..at check_expr: ⊢ (x.status > 0) : ???
.at check_path_pattern: ⊢ ⟪x: * {status:bool,*}⟫ WHERE (x.status > 0)
: ???
at check_path_pattern: ⊢ ⟪x: * {status:bool,*}⟫ WHERE (x.status > 0) -
[Author {*}]→ WHERE y.foo : ???
```

Errors

```
File "...", line 173 in check_expr:
  Variable y not found in context
..at check_expr: ⊢ y.foo : ???
.at check_path_pattern: ⊢ -[Author {*}]→ WHERE y.foo : ???
at check_path_pattern: ⊢ ⟪x: * {status:bool,*}⟫ WHERE (x.status > 0) -
[Author {*}]→ WHERE y.foo : ???
```

Types

$$\tau ::= \mathbb{Z} \mid \ldots \mid \star \mid \tau \oplus \tau \mid \bot$$

Simple type

$$R ::= \{\overline{a_i : \tau_i}\} \mid \{\overline{a_i : \tau_i}, \star\} \mid \bot$$

Property type

$$\ell ::= \mathtt{1} \mid \star \mid \ell \,\&\, \ell \mid \ell \oplus \ell \mid \top$$

Label type

$$L ::= \ell \; R$$

Descriptor type

Matching by properties

Previously:

$$[\![(x? : \ell)]\!]_G = \{\ (n, x? \mapsto n) \mid n \in N_{id},\ \vdash \ell \in \mathscr{L}(n)\}$$

A single label

Now:

$$[\![(x? : L)]\!]_G = \{\ (n, x? \mapsto n) \mid n \in N_{id}, \vdash \mathscr{L}(n)\ \mathscr{P}(n) : L', L' <: L\}$$

Now we include matching by label and properties

We infer labels and properties of the node

We test by using subtyping instead

Type Judgment

$$S \vdash \pi : P; \Gamma$$

Partial schema

Path query

Path type

Type environment

$$N ::= (L)$$

Node type

$$E ::= N \xrightarrow{L} N \mid N \overset{L}{\sim} N$$

Edge type

$$S ::= \cdot \mid S, N \mid S, E$$

Partial schema

Type Judgment

$$S \vdash \pi : P; \Gamma$$

Partial schema

Path query

Path type

Type environment

$$P ::= N \mid P - N \mid P + P \mid \bot$$

Path type

$$T ::= N \mid E \mid T + T \mid [T]^{l} \mid \bot \mid \text{Null}$$

Variable type

$$\Gamma ::= \cdot \mid \Gamma, x \mapsto T$$

Partial schema

FPPC

👍 FPPC detects all errors!

Incompatible shapes

```
(x) -[x]->
```

ill-typed! meet between a node type and an edge type is not defined.

empty warning! meet between bool and string is ⊥.

Incompatible properties

```
(x WHERE x.isBlocked is bool) -> (x WHERE x.isBlocked is str)
```

Nonexistent attributes

```
(x WHERE x.amout>0)
```

empty warning! amout can not be find in the schema.

Wrong usage of attributes

```
(x WHERE x.isBlocked>0)
```

empty warning! type error in the conditional term is propagated to the pattern

Unbound variables

```
(y WHERE x.isBlocked=true)
```

ill-typed! variable x is not present in the type environment

Properties

Soundness

- Well-typed path patterns do not get stuck during execution.
- their evaluation always produces “well-typed result”, i.e. subtype of the original path pattern type (but only for the extremes nodes due to repetition).

If $S \vdash \pi : P; \Gamma$ and $S \vdash G$ then
$\exists \gamma \,.\, [\![\pi]\!]_G = \gamma,\ \forall (p, \mu) \in \gamma,\ G \vdash \mu : \Gamma,\ G \vdash p : (N_1, N_2)$
and $(N_1, N_2) \leq (\text{first}(P), \text{last}(P))$

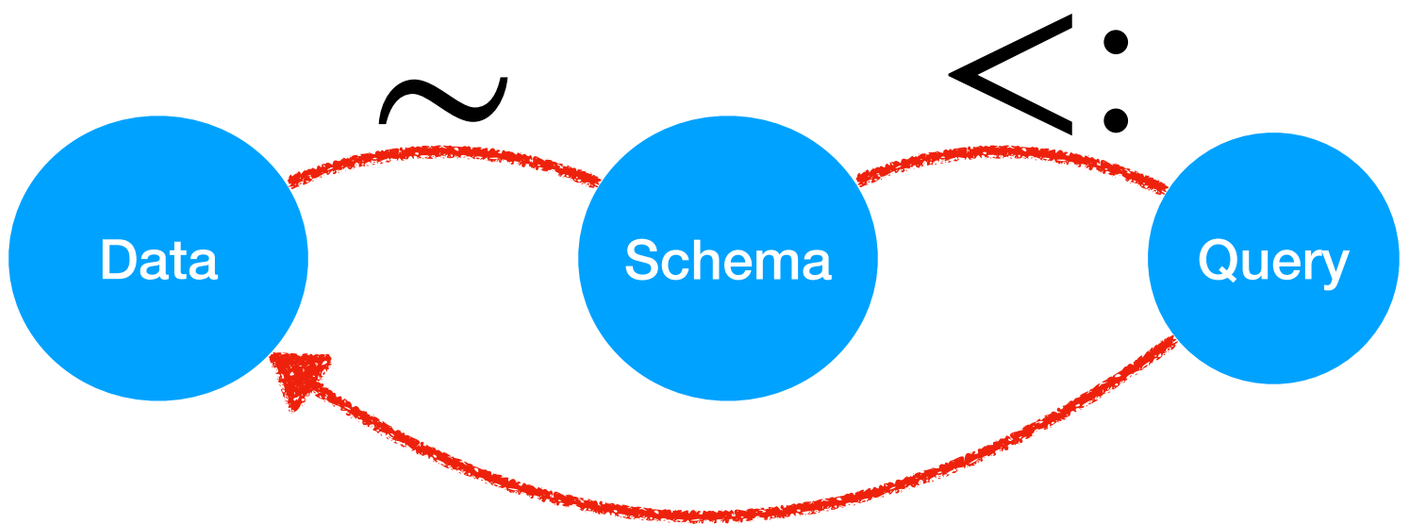
Emptiness

- a well-typed path pattern, if either the path type is empty or the type environment is empty,
- then the result is definitively empty.

If $S \vdash \pi : P; \Gamma,\ S \vdash G$ **and** $(\text{empty}(P)$ **or** $\text{empty}(\Gamma))$ **then** $[\![\pi]\!]_G = \{\,\}$

Restriction on Emptiness

- The Schema S must be *consistently equal* to the actual data, not a consistent supertype:
  - Consider: $\mathscr{L}_{id} = \{n_1 = \{A\}, n_2 = \{A, B\}\}$
- If Schema $S = (\ \top\ \{\ \star\ \})$, then path query $(x : A)$ would be deemed empty, even though the query result would be not empty.



[Siek et al.2015]

Gradual Guarantee

- Static Gradual Guarantee (SGG) ensures that typing remains monotonic with respect to precision.

If $S_1 \sqsubseteq S_2,\ \pi_1 \sqsubseteq \pi_2$ **and** $S_1 \vdash \pi_1 : P_1; \Gamma_1$
then $S_2 \vdash \pi_2 : P_2; \Gamma_2,\ P_1 \sqsubseteq P_2$ **and** $\Gamma_1 \sqsubseteq \Gamma_2$

- Dynamic Gradual Guarantee (DGG) guarantees that reduction preserves monotonicity with respect to precision

If $S_1 \sqsubseteq S_2,\ S_1 \vdash G,\ S_2 \vdash G,$
$S_1 \vdash \pi_1 : P_1; \Gamma_1,\ S_2 \vdash \pi_2 : P_2; \Gamma_2$ **and** $\pi_1 \sqsubseteq^* \pi_2$
then $[\![\pi_1]\!]_G \sqsubseteq [\![\pi_2]\!]_G$

Does it work in practice?

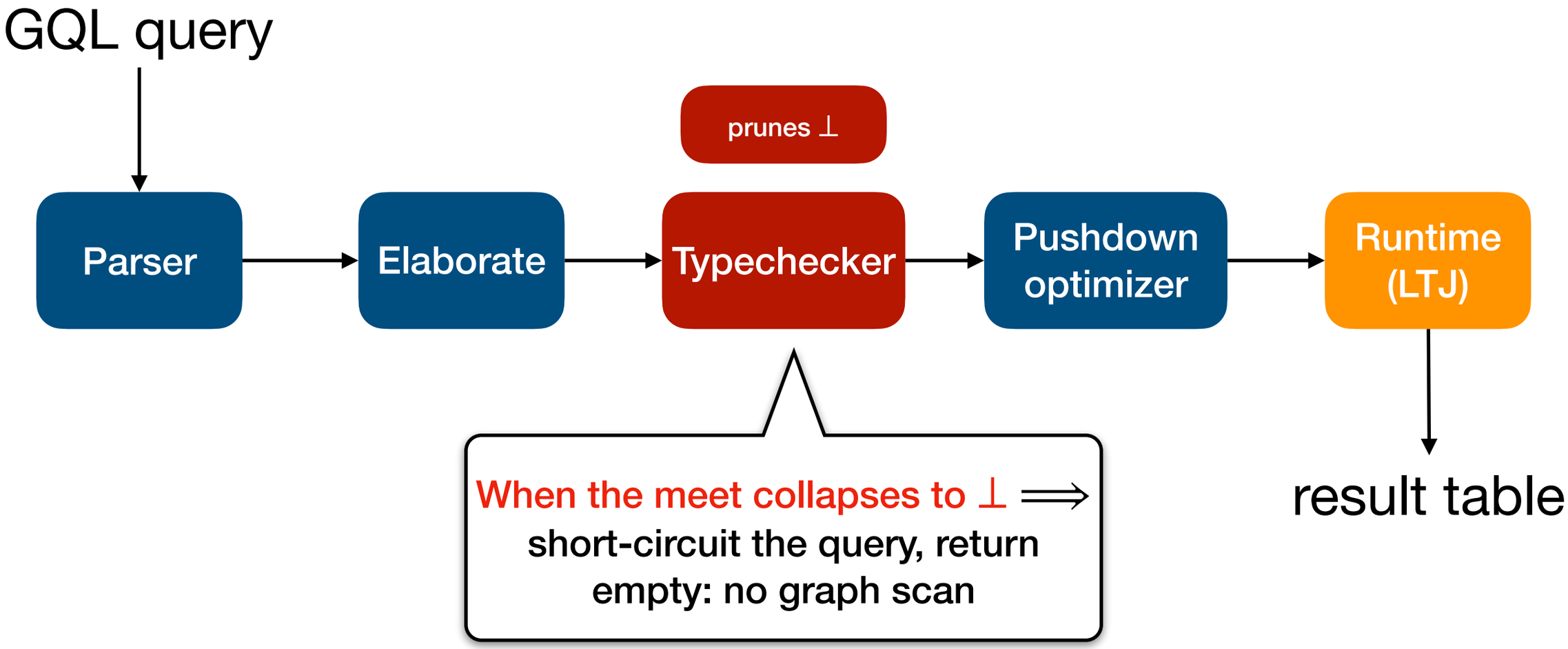
Evaluation

- “Typing must be faster than running the query”

- **froGQL**: A single-file ISO GQL property-graph engine in Rust. MIT-licensed, on PyPI and NPM.

froGQL

The same type system now lives inside a real engine, and runs before any graph access.



The typechecker is (almost) free

< 6%

type checker overhead

on 29 of 36 valid-query cells (<1% whenever the runtime exceeds 1ms)

146 μs

to reject a 4-hop contradiction

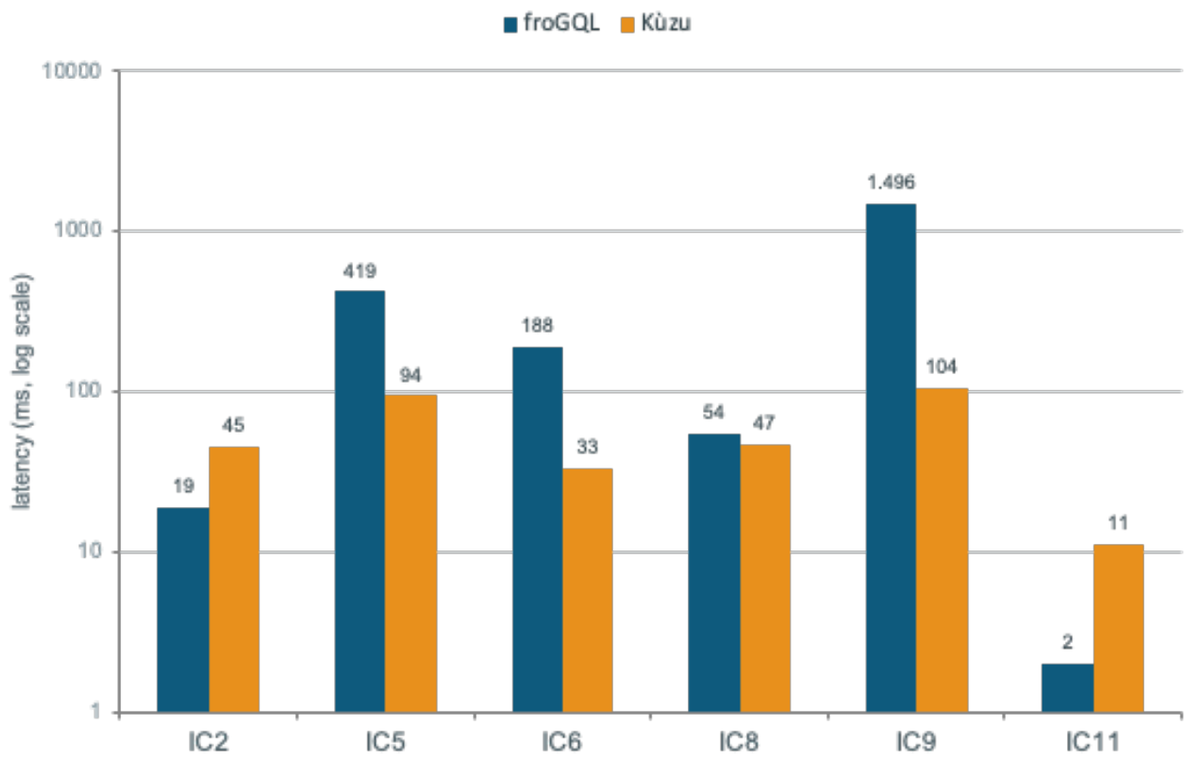
that the unchecked runtime spend 38s enumerating

2,280,000x

peak speed-up

on the worst chain enumeration measured to completion

Where LTJ wins, and where work remains



Limitations

- No formal support for negation on label:
  - What is the meaning of $(x : A \ \& \ \neg \star)$?
  - How to implemented this fast?
- Potential explosion of types due multiple unions + long schema.

Future work

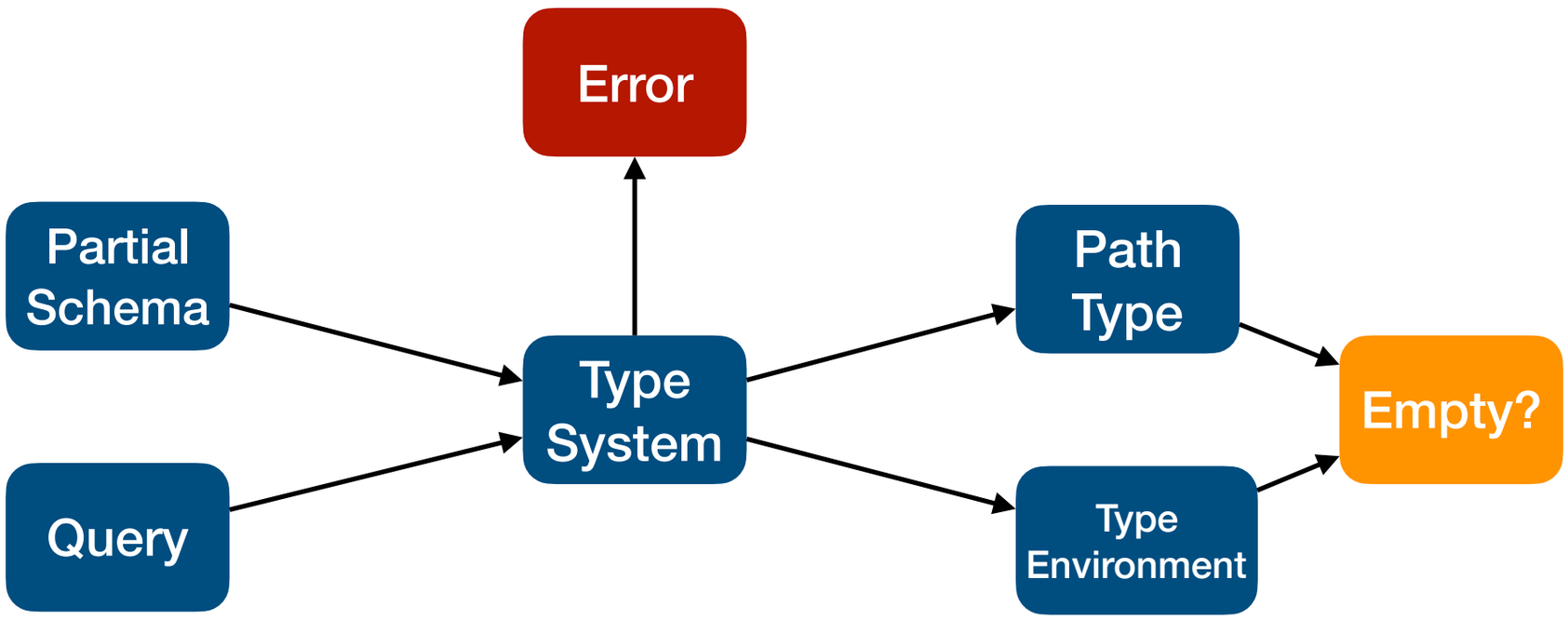
- Not only path patterns but to formally support full range of GQL queries.

- Formally supporting aggregation and negation in label predicates.

- Exploring other optimizations for query evaluation under these typing enhancements.

- Further refinements to the interaction between schema evolution and gradual typing in graph databases (Spark/Shacl).

Thank you


Error
Partial Schema
Query
Type System
Path Type
Type Environment
Empty?

$$\boxed{[\![\pi]\!]_G = \{\overline{(p,\mu)}\}}$$

$$[\![(x? : L)]\!]_G = \left\{(n, x?\widetilde{\mapsto}n) \;\middle|\; n \in N_{id}, \vdash \mathcal{L}(n)\ \mathcal{P}(n) : L', L' \lesssim L\right\} \tag{R()}$$

$$[\![\xrightarrow{x?:L}]\!]_G = \left\{(n_1 e n_2, x?\widetilde{\mapsto}e) \;\middle|\; e \in E_d, n_1 = \mathsf{src}(e), n_2 = \mathsf{tgt}(e), \vdash \mathcal{L}(e)\ \mathcal{P}(e) : L', L' \lesssim L\right\} \tag{R⟶}$$

$$[\![\xleftarrow{x?:L}]\!]_G = \left\{(n_2 e n_1, x?\widetilde{\mapsto}e) \;\middle|\; e \in E_d, n_1 = \mathsf{src}(e), n_2 = \mathsf{tgt}(e), \vdash \mathcal{L}(e)\ \mathcal{P}(e) : L', L' \lesssim L\right\} \tag{R⟵}$$

$$[\![\overset{x? : L}{\sim}]\!]_G = \left\{(n_1 e n_2, x?\widetilde{\mapsto}e) \;\middle|\; e \in E_u, \mathsf{endpoints}(e) = \{n_1, n_2\}, \vdash \mathcal{L}(e)\ \mathcal{P}(e) : L', L' \lesssim L\right\} \tag{R~}$$

$$[\![\underline{x?:L}]\!]_G = [\![\overset{x? : L}{\sim}]\!]_G \cup [\![\xrightarrow{x?:L}]\!]_G \cup [\![\xleftarrow{x?:L}]\!]_G \tag{R–}$$

$$[\![\pi_1\ \pi_2]\!]_G = \left\{(p_1 \cdot p_2, \mu_1 \cdot \mu_2) \;\middle|\; (p_i, \mu_i) \in [\![\pi_i]\!]_G, p_1 \text{ and } p_2 \text{ concatenate}, \mu_1 \text{ and } \mu_2 \text{ concatenate}\right\} \tag{R·}$$

$$[\![\pi_1 + \pi_2]\!]_G = \left\{(p, \mathsf{fillNulls}(\mu, \mathsf{freevar}(\pi_1 + \pi_2))) \;\middle|\; (p, \mu) \in [\![\pi_1]\!]_G \cup [\![\pi_2]\!]_G\right\} \tag{R+}$$

$$[\![\pi_{\langle t \rangle}]\!]_G = \left\{(p, \mu) \in [\![\pi]\!]_G \;\middle|\; [\![t]\!]_{\mu,\delta} = \mathsf{true}\right\} \tag{Rθ}$$

$$[\![\pi^{l..u}]\!]_G = \bigcup_{i=l}^{u} [\![\pi]\!]_G^i \tag{Rl..u}$$

$$[\![c]\!]_{\mu,\delta} = \mathbf{ok}\ c \tag{Rc}$$

$$[\![x.a]\!]_{\mu,\delta} = \begin{cases} \mathbf{stuck} & \text{if } x \notin dom(\mu) \\ \mathbf{ok}\ \delta(\mu(x))(a) & \text{if } a \in dom(\delta(\mu(x))) \\ \mathbf{ok}\ \mathsf{null} & \text{otherwise} \end{cases} \tag{Ra}$$

$$[\![t\ \mathsf{is}\ \tau]\!]_{\mu,\delta} = \mathsf{for}\ (v \leftarrow [\![t]\!]_{\mu,\delta})\ \mathsf{yield}\ (ty(v) \lesssim \tau) \tag{Ris}$$

$$[\![t\ \mathsf{as}\ \tau]\!]_{\mu,\delta} = \mathsf{for}\ (v \leftarrow [\![t]\!]_{\mu,\delta})\ \mathsf{yield}\ (v \Rightarrow \tau) \tag{Ras}$$

$$[\![t_1\ \mathsf{bop}\ t_2]\!]_{\mu,\delta} = \begin{cases} \mathbf{stuck} \quad \text{if } \triangle(\mathsf{bop}) \text{ is not defined or } [\![\mathsf{bop}]\!] \text{ is not defined} \\ \mathsf{for}\ (v_1 \leftarrow [\![t_1\ \mathsf{as}\ \tau_1]\!]_{\mu,\delta};\ v_2 \leftarrow [\![t_2\ \mathsf{as}\ \tau_2]\!]_{\mu,\delta}) \\ \qquad \mathsf{yield}\ (v_1\ [\![\mathsf{bop}]\!]\ v_2) \quad \text{if } \triangle(\mathsf{bop}) = \tau_1 \times \tau_2 \to \tau_3 \end{cases} \tag{Rbop}$$

$$[\![\mathsf{uop}\ t]\!]_{\mu,\delta} = \begin{cases} \mathbf{stuck} \quad \text{if } \triangle(\mathsf{uop}) \text{ is not defined or } [\![\mathsf{uop}]\!] \text{ is not defined} \\ \mathsf{for}\ (v \leftarrow [\![t\ \mathsf{as}\ \tau_1]\!]_{\mu,\delta})\ \mathsf{yield}\ ([\![\mathsf{uop}]\!]\ v) \quad \text{if } \triangle(\mathsf{uop}) = \tau_1 \to \tau_2 \end{cases} \tag{Ruop}$$

$$\boxed{S \vdash \pi : P;\ \Gamma}$$

$$\text{(Tnode)}\ \frac{S \vdash (x? : L) \triangleright T}{S \vdash (x? : L) : \lfloor T \rfloor^{-};\ x?\ \widetilde{\mapsto}\ T}$$

$$\text{(Tedge)}\ \frac{S \vdash \overset{x?:L}{\Longleftrightarrow} \triangleright T}{S \vdash \overset{x?:L}{\Longleftrightarrow} : \lfloor T \rfloor^{\Leftrightarrow};\ x?\ \widetilde{\mapsto}\ T}$$

$$\text{(Tcat)}\ \frac{S \vdash \pi_1 : P_1;\ \Gamma_1 \qquad S \vdash \pi_2 : P_2;\ \Gamma_2}{S \vdash \pi_1\ \pi_2 : P;\ \Gamma}\ \frac{S \vdash P_1 \sqcap P_2 \triangleright P \qquad S \vdash \Gamma_1 \sqcap \Gamma_2 \triangleright \Gamma}{}$$

$$\text{(Tu)}\ \frac{S \vdash \pi_1 : P_1;\ \Gamma_1 \qquad S \vdash \pi_2 : P_2;\ \Gamma_2}{S \vdash \pi_1 + \pi_2 : P_1 \sqcup P_2;\ (\Gamma_1 \sqcup \Gamma_2)}$$

$$\text{(Tcon)}\ \frac{S \vdash \pi : P;\ \Gamma \qquad \Gamma \vdash t : \tau \qquad \tau \sqcap \mathbb{B} \neq \bot}{S \vdash \pi_{\langle t \rangle} : P;\ \Gamma}$$

$$\text{(Tfail)}\ \frac{S \vdash \pi : P;\ \Gamma \qquad \Gamma \vdash t : \tau \qquad \tau \sqcap \mathbb{B} = \bot}{S \vdash \pi_{\langle t \rangle} : \bot;\ \Gamma}$$

$$\text{(Trep)}\ \frac{S \vdash \pi : P;\ \Gamma \qquad len(P) > 0 \qquad 0 \leq l \leq u \qquad S \vdash P^{\min(l,2)} \triangleright P'}{S \vdash \pi^{l..u} : P';\ [\Gamma]^{l}}$$