Property Graph Transformations in Action: From Data Integration to Causal Analysis

Angela Bonifati
Lyon 1 University, CNRS Liris & IUF (France)
Data Conference 2026 - Porto, 16-18 July 2026

Outline

- Property graphs in graph databases
- Graph transformations as primitives for:
  - data integration
  - metadata management
  - data cleaning
- Graph transformations as a backbone for:
  - causal analysis
  - causal model extraction and identification
  - counterfactual analysis

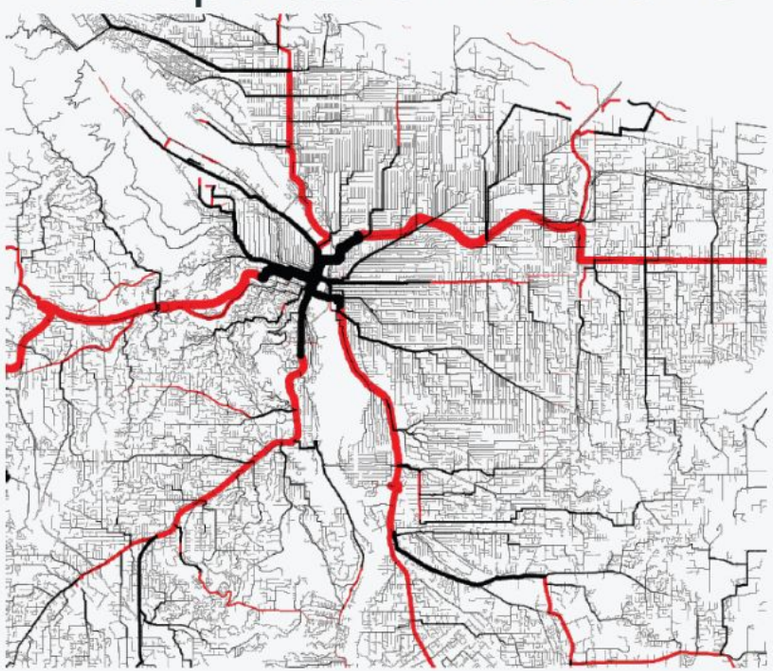
Outline

- **Property graphs in graph databases**
- Graph transformations as primitives for:
  - data integration
  - metadata management
  - data cleaning
- Graph transformations as a backbone for:
  - causal analysis
  - causal model extraction and identification
  - counterfactual analysis

Graphs are everywhere!

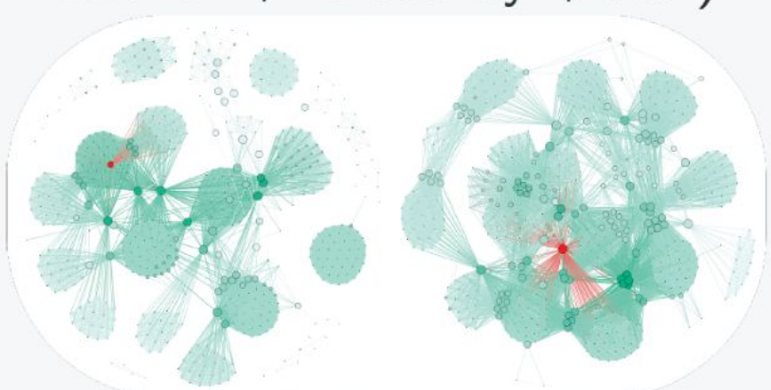
They provide a universal and simple blueprint for managing data

Transportation Networks

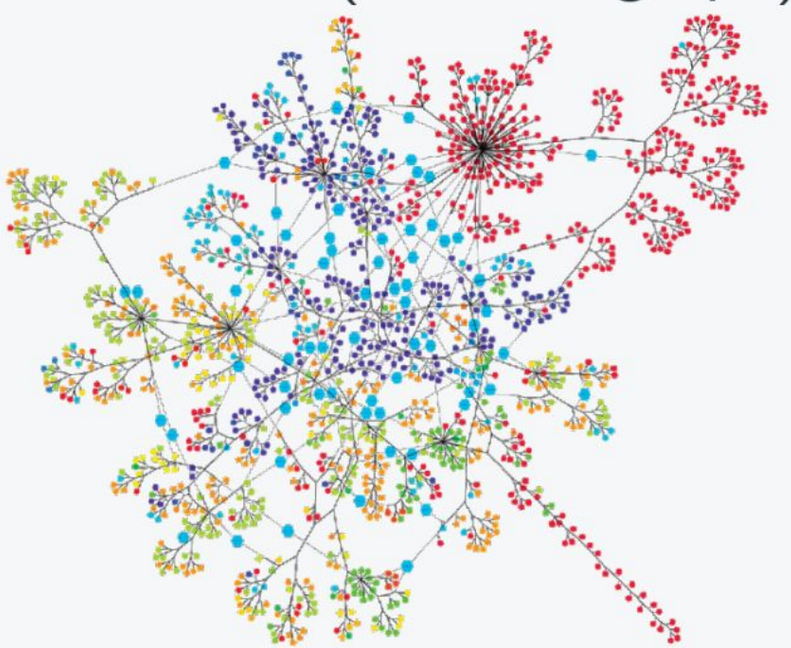


Scientific Publications

(with relationships “same author”, “cited by”, etc.)



Health Care (Covid19 graph)



Graph Technology Landscape

Property graphs: *Neo4j, Amazon Neptune,Memgraph,TigerGraph, JanusGraph,Rocketgraph, NebulaGraph,DGraph,Ultipa, FalkorDB*

Graph Data Integration: *Graph.Build, Apache Hop, Apache Airflow, Kafka, Neo4j ETL Tool, Graph in Microsoft Fabric, Pentaho*

https://linkurious.com/blog/introduction-graph-technology-landscape/

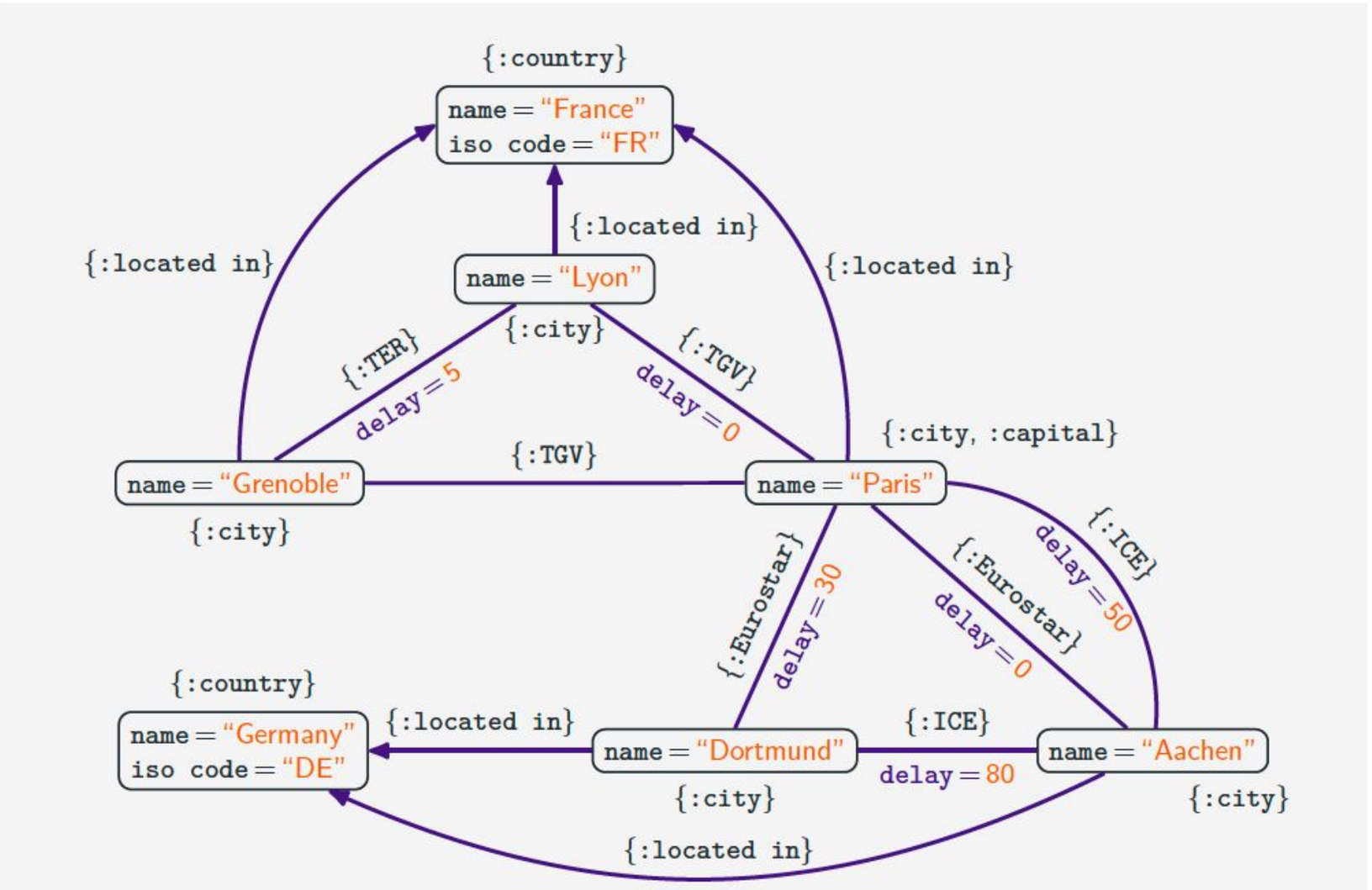
Graphs as unifying abstractions

- Graphs are **natural abstractions** for representing interconnected objects when encoding, explaining and predicting real-world and digital-world phenomena.
- Graphs are underpinning several **data management ecosystems**, in societal, scientific, RDF, product and digital domains.
- There is **no unique killer application** for graphs, but several exist
- Nevertheless, the **data models, query languages and system requirements** needed for graphs are constantly evolving.

The Ubiquity of Large Graphs and Surprising Challenges of Graph Processing: Extended Survey

Siddhartha Sahu · Amine Mhedhbi · Semih Salihoglu · Jimmy Lin · M. Tamer Özsu

A Property Graph by example

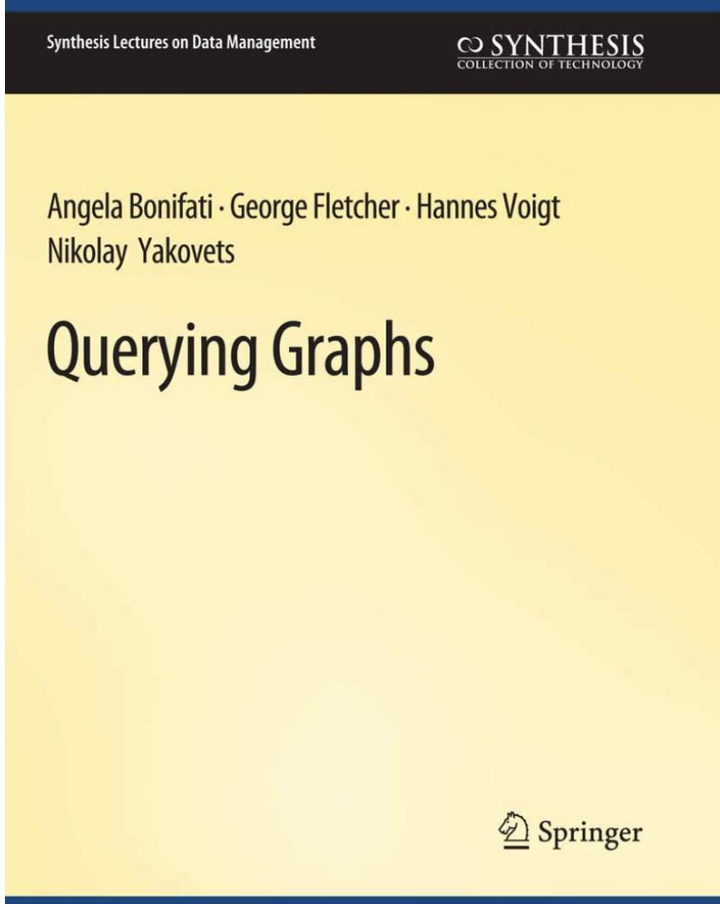


Property graphs formally

Assume pairwise-disjoint sets of :
$\mathcal{O}$ (objects), $\mathcal{L}$ (labels), $\mathcal{K}$ (property keys), and $\mathcal{N}$ (values)

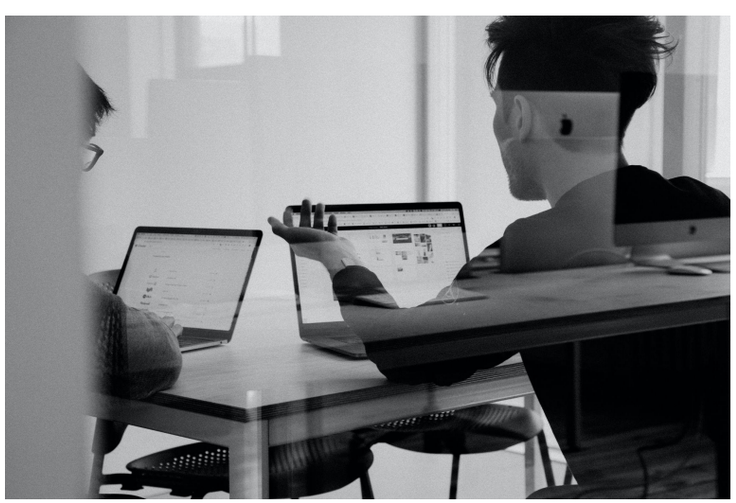
A property graph is a structure V, E where

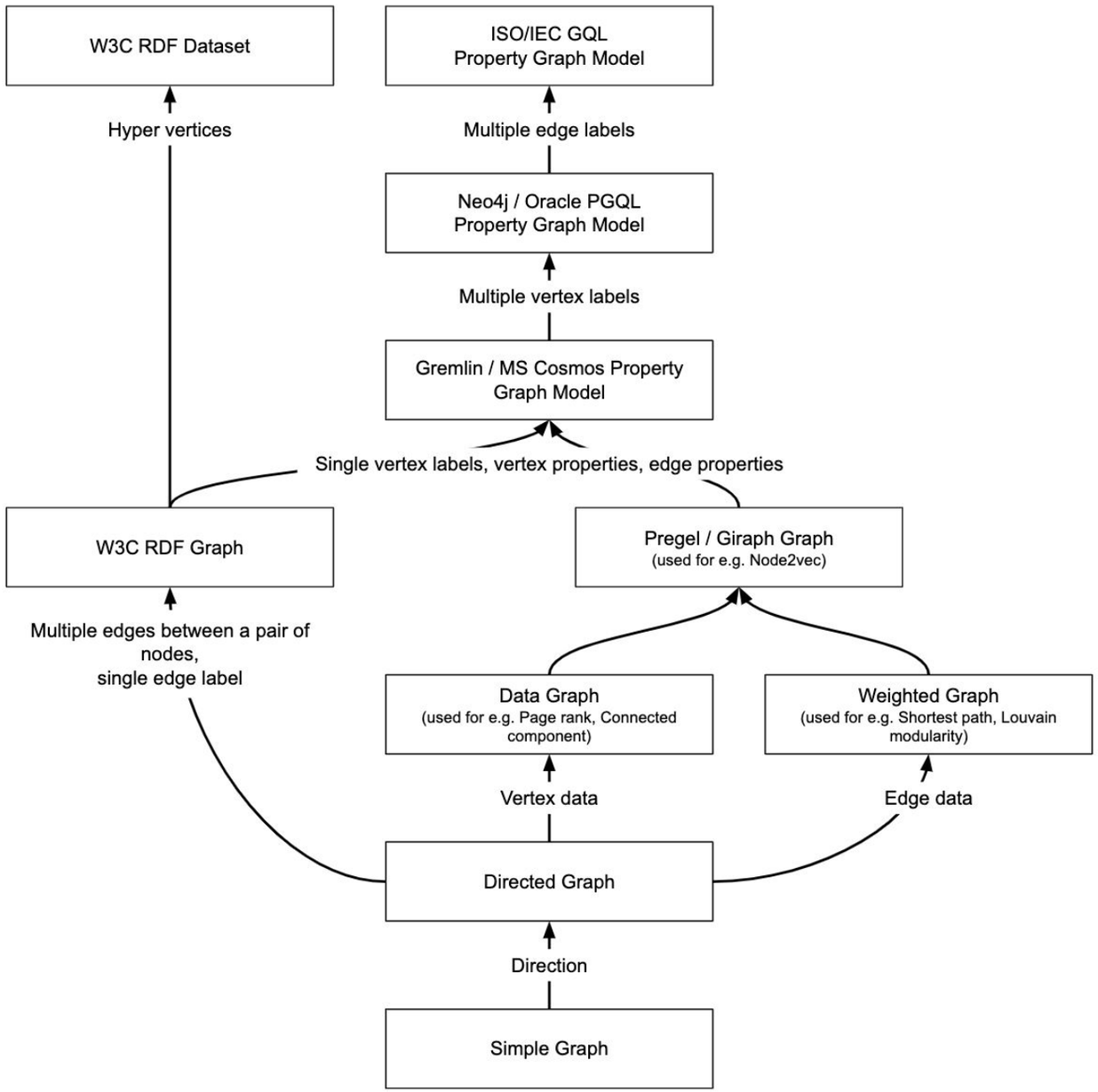
- $\mathcal{V} \subseteq \mathcal{O}$: finite set of objects (vertices)
- $E \subseteq \mathcal{O}$: finite set of objects (edges)
- $\eta : E \to V \times V$:
  assignment of an ordered pair of vertices to each edge
- $\lambda : V \cup E \to \mathcal{P}(\mathcal{L})$:
  assignment of a finite set of labels to each object
- $\nu : (V \cup E) \times \mathcal{K} \to \mathcal{N}$:
  partial assignment of values for properties to objects



Transformations across data models

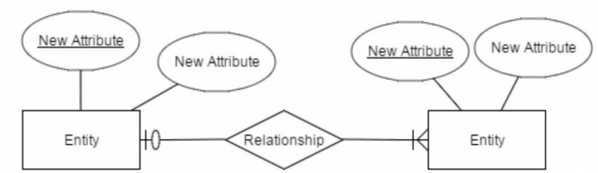
- Need of making different data models **interoperable** via mappings or direct graph transformations
- Need of designing graph transformations within the same data model and across data models (w/out **schema**)





Schemas for Graphs: a Fragmented Landscape

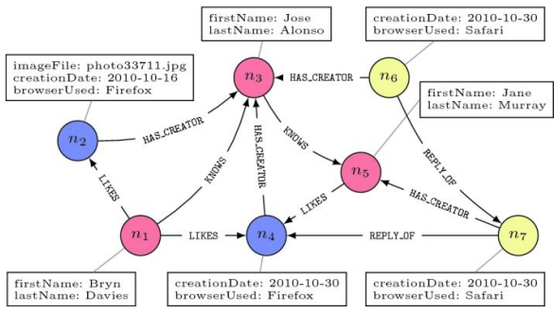
(1) ER Models:
- (a) Chen ER
- (b) Extended ER
- (c) Enhanced ER
- (d) ORM2
- (e) UML Class Diagram



(2) RDF Schemas:
- (a) RDFS
- (b) OWL
- (c) SHACL
- (d) ShEx

(3) Tree-shaped Schemas:
- (a) DTD/XML Schema
- (b) JSON Schema
- (c) RELAX NG

(4) Graph Schemas
- (a) GraphQL
- (b) openCypher
- (c) SQL/PGQ



(5) (Limited) Schemas in Graph DBs
- (d) AgensGraph
- (e) ArangoDB
- (f) DataStax
- (g) JanusGraph
- (h) Nebula Graph/nGQL
- (i) Neo4j
- (j) Oracle/PGQL
- (k) OrientDB/SQL
- (l) Sparksee
- (m) TigerGraph/GSQL
- (n) TypeDB/TypeQL

Property Graph Standards

```
CREATE GRAPH TYPE   fraudGraphType STRICT {
  (personType: Person {name STRING}),
  (customerType: personType & Customer {id INT32}),
  (creditCardType: CreditCard {num STRING}),
  (transactionType: Transaction {num STRING}),
  (accountType: Account {id INT32}),
  (:customerType)
    -[ownsType: owns]->
  (:accountType),
  (:customerType)
    -[usesType: uses]->
  (:creditCardType),
  (:transactionType)
    -[chargesType: charges {amount DOUBLE}]->
  (:creditCardType),
  (:transactionType)
    -[activityType: deposits|withdraws]->
  (:accountType)
}
```

Fig. 2. PG-Schema of a fraud graph schema.

PG-Schema: Schemas for Property Graphs

RENZO ANGLES, Faculty of Engineering, Universidad de Talca, Chile
ANGELA BONIFATI, Lyon 1 University & Liris CNRS, France
& SAMOVAR - Institut Polytechnique de Paris, France

Industrial Track Paper
SIGMOD '21, June 20–25, 2021, Virtual Event,

PG-Keys: Keys for Property Graphs

Renzo Angles
Universidad de Talca, IMFD Chile

Angela Bonifati
Lyon 1 Univ., Liris CNRS & INRIA

Stefania Dumbrava
ENSIIE & Inst. Polytechnique de Par

George Fletcher
Eindhoven Univ. of Technology

Keith W. Hare
JCC Consulting Inc., Neo4j

Jan Hidders
Birkbeck, Univ. of London

Victor E. Lee
TigerGraph

Bei Li
Google LLC

Leonid Libkin
U. of Edinburgh, ENS-Paris/PSL, Neo4

Wim Martens
University of Bayreuth

Filip Murlak
University of Warsaw

Josh Perryman
Interos Inc.

Ognjen Savković
Free Univ. of Bozen-Bolzano

Michael Schmidt
Amazon Web Services

Juan Sequeda
data.world

Sławek Staworko
U. Lille, INRIA LINKS, CRIStAL CNRS

Dominik Tomaszuk
Inst. of Comp. Sci., U. of Bialystok

ABSTRACT

We report on a community effort between industry and academia to shape the future of property graph constraints. The standardi
for a property graph query

KEYWORDS

property graphs,

Towards GQL-compliant Property Graph Schemas

- Limited Support of Schemas in Contemporary Graph Databases
  - 11 graph databases are reviewed in the paper [PGS23]
  - **AgensGraph**
  - **ArangoDB**
  - **DataStax**
  - **JanusGraph**
  - **Nebula Graph/nGQL**
  - **Neo4j**
  - **Oracle/PGQL**
  - **OrientDB/SQL**
  - **Sparksee**
  - **TigerGraph/GSQL**
  - **TypeDB/TypeQL**

- Need of a consensus on design requirements of PG Schemas
- Ongoing standardization of PG schemas (already implemented in SQL/PGQ)

[PGS23] R. Angles et al. PG-Schemas: Schemas for Property Graphs. SIGMOD Conference 2023: 198:1-198:25

Descriptive and Prescriptive PG Schemas

- *Schemas are one of the key metadata pillars for tackling the ambiguity of generative and agentic AI*
- Three possible scenarios in PG-Schemas:
  - Schema-First (or **Prescriptive**)
    - schema provided during setup
  - Flexible Schema (or **Descriptive**)
    - users can use schema as description of what is in the data
  - **Partial** Schema (Prescriptive and Descriptive co-exist)
    - both prescriptive and descriptive are allowed for the same property graph

Key Constraints for Property Graphs

Keys are one of the key components of PG-Schemas … key in data management for identifying, referencing and constraining objects.

For example, `Person` nodes

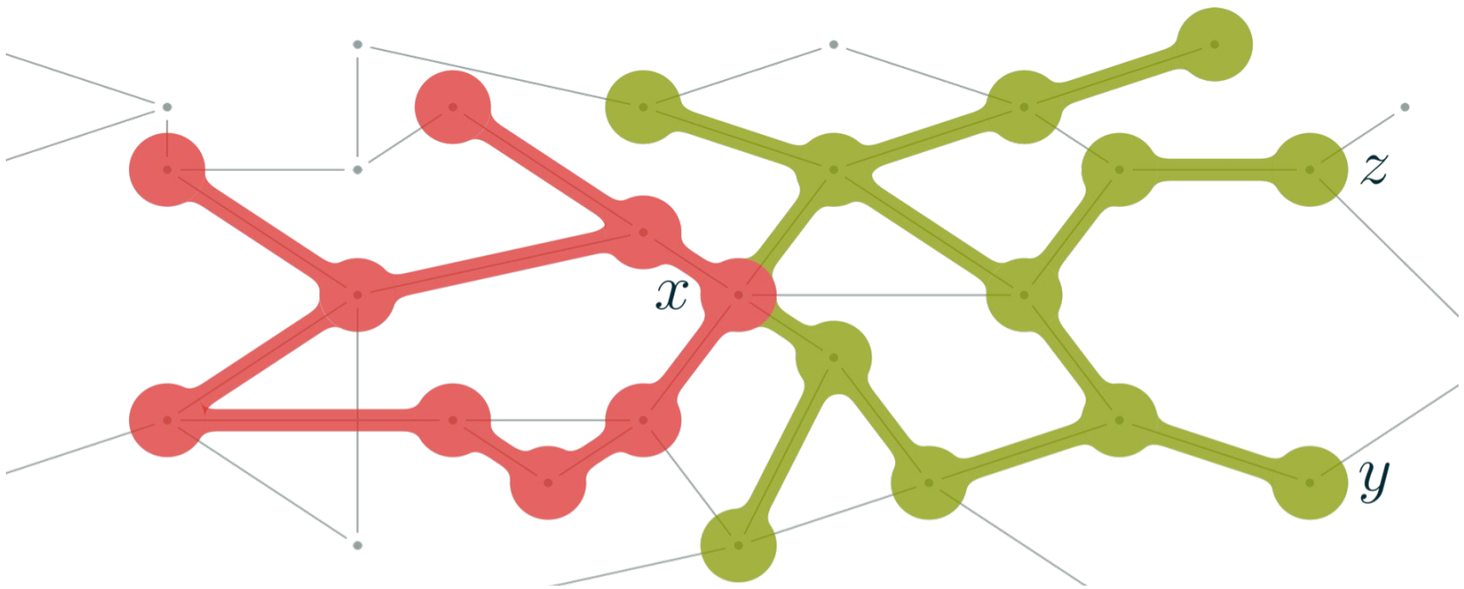
- are uniquely identified by their login ID
- can be referenced using one of their email addresses (and it is mandatory that each person has at least one email), of which at most one can be the preferred email.
- have zero or more aliases which are exclusive (i.e., no two people can share an alias)

and discussion `Forum` nodes are identified by the forum's name and the person who moderates the forum

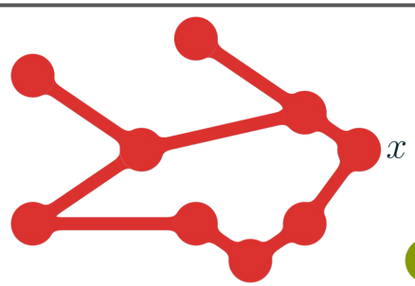
- `(:Person)<-[:hasModerator]-(:Forum)`

[PGK21] R. Angles et al. PG-Keys: Keys for Property Graphs. SIGMOD Conference 2021: 2423-2436

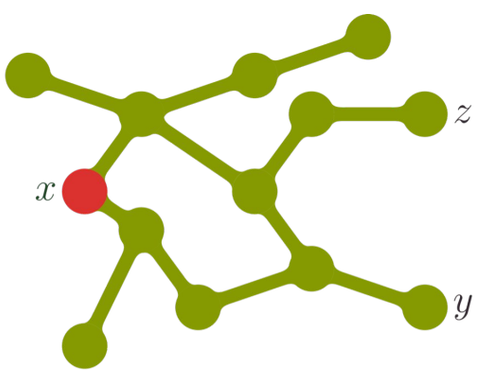
PG-Keys: Keys for Property Graphs




```
FOR x WITHIN
```




```
IDENTIFIER y, z WITHIN
```



Design requirements

1. Flexible choice of key scope and descriptor of key values.
2. Keys for nodes, edges, and properties.
3. Identify, reference, and constrain objects.
4. Easy to validate.

Outline

- Property graphs in graph databases
- **Graph transformations as primitives for:**
  - data integration
  - metadata management
  - data cleaning
- Graph transformations as a backbone for:
  - causal analysis
  - causal model extraction and identification
  - counterfactual analysis

Data Integration on Property Graphs with PG transformations

- The flexibility of the PG model requires performing **complex** transformations:
  - Nodes may be transformed into edges (and vice-versa).
  - A new piece of target data may represent a complex pattern over the source data:
    - Defining an appropriate notion of identity is crucial (w/out schemas)
  - Primitives for handling data contents:
    - Data values may contribute to the identity of new elements.
    - Similarly, labels may contribute to the identity of new elements.
  - New nodes/edges may aggregate the content of existing ones.

State of affairs for Property Graph Transformations

- Typical semantics of standard graph queries (GQL, SQL/PGQ) outputs tuples
  …hence, it is not compositional
- Existing solutions and current approaches:
  - In practical graph dbms such as Neo4j:
    - Handcrafted openCypher scripts
    - APOC (Awesome Procedures on Cypher)
  - In the research literature:
    - Data exchange for graph databases (Barcélo et al. 2013, Francis et al. 2017)
- Declarative specifications have long been recognized as pivotal for solving data programmability problems (Bernstein et al., 2007).

Ingredients of Property Graph Transformations

- Transforming Property Graphs:
  - Syntax and Semantics
  - Static analysis: Consistency checking and satisfiability
  - Efficiently implementation of PG transformations in openCypher

- DTGraph: Declarative Transformations of Property Graphs:
  - Integrating the framework into openCypher: the GENERATE clause

[BMR24a] A. Bonifati, F. Murlak, Y. Ramusat. Transforming Property Graphs. In PVLDB Vol. 17 (2024)
[BMR24b] A. Bonifati, F. Murlak, Y. Ramusat, Amela Fejza, Rachid Echahed. DTGraph: Declarative Transformations of Property Graphs. In PVLDB Vol. 17 (2024)

Rule syntax

$$(u : \text{User}), (a : \text{Address}), (\ell : \text{Location}) \underset{\langle u.address{=}a.aid,\ u.address{=}\ell.aid\rangle}{\Longrightarrow} ((u) : \text{Person}) \xrightarrow{:\text{HasLocation}} ((\ell.countryName) : \text{Country}) \quad \langle name{=}u.name\rangle \qquad \langle name{=}\ell.countryName,\ code{=}\ell.countryCode\rangle$$

- The left-hand-side is a GPC (graph pattern calculus) query.
- We only export Node and Edge variables.
- The right-hand-side is either a node or an edge constructor,

$$C_s(a,u,\ell) := \{ \quad \text{Id: } (u) \quad \text{Labels: } \{\text{Person}\} \quad \text{Properties: } \langle name = u.name\rangle \},$$

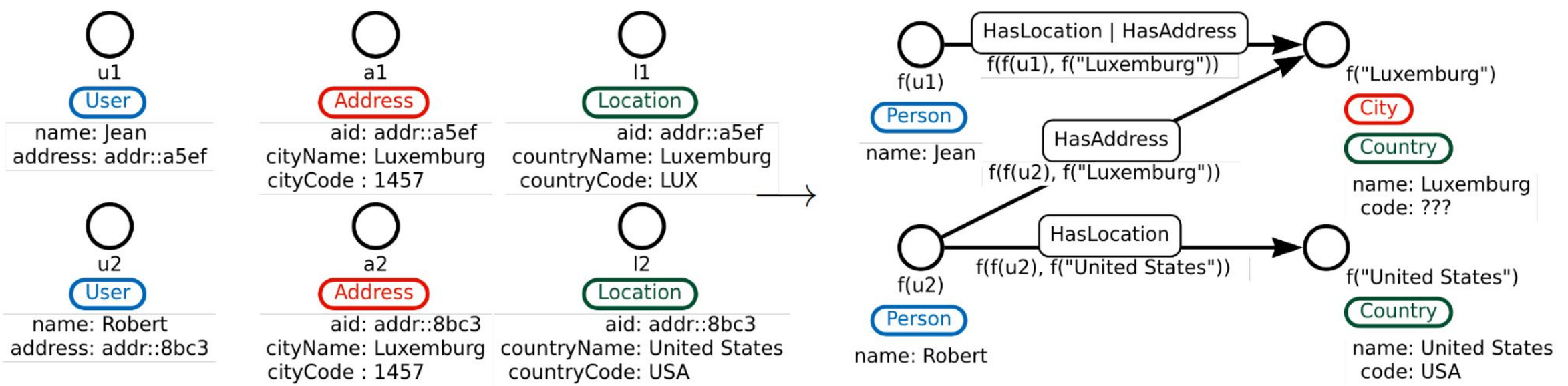
$$C(a,u,\ell) := \{ \quad \text{Id: } () \quad \text{Labels: } \{\text{HasLocation}\} \quad \text{Properties: } \langle\rangle \},$$

$$C_t(a,u,\ell) := \{ \quad \text{Id: } (\ell.countryName) \quad \text{Labels: } \{\text{Country}\} \quad \text{Properties: } \langle name = \ell.countryName,\ code = \ell.countryCode\rangle \}.$$

Rule syntax (example)

$$(u : \text{User}), (a : \text{Address}), (\ell : \text{Location}) \;\underset{\langle u.address{=}a.aid,\ u.address{=}\ell.aid\rangle}{\Longrightarrow}\; \underset{\langle name{=}u.name\rangle}{((u) : \text{Person})} \xrightarrow{: \text{HasLocation}} \underset{\langle name{=}\ell.countryName,\ code{=}\ell.countryCode\rangle}{((\ell.countryName) : \text{Country})}$$

$$(u : \text{User}), (a : \text{Address}), (\ell : \text{Location}) \;\underset{\langle u.address{=}a.aid,\ u.address{=}\ell.aid\rangle}{\Longrightarrow}\; \underset{\langle name{=}u.name\rangle}{((u) : \text{Person})} \xrightarrow{: \text{HasAddress}} \underset{\langle name{=}a.cityName,\ code{=}a.cityCode\rangle}{((a.cityName) : \text{City})}$$



Consistency of PG Transformations

A *conflict* is when there are two or more concurrent values for an attribute.

Definition (Consistency – For a transformation)

A transformation *T* is *consistent* if for every input property graph *G* , its output has no conflict.

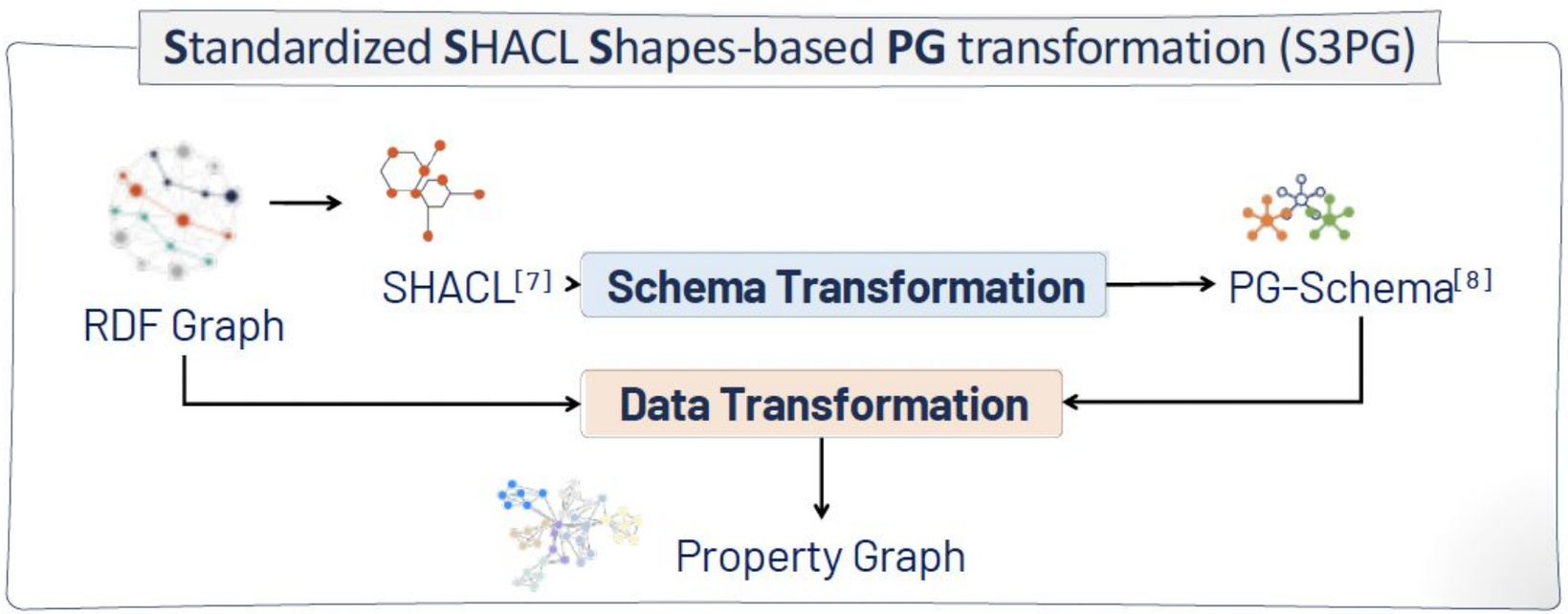
Definition (Satisfiability – For a GPC query (resp. GPC+))

Given a GPC (resp. GPC+) query *P*, check if *P* is *satisfiable*.

* *Complexity results on PTIME reducibility are in the paper*

Graph transformations using standardized schemas

- **Problem:** Interoperability between RDF & PG data models; **Goal: Lossless**, **schema-aware** transformation of RDF graph data into property graph data; **Challenges:** Structural mismatch, heterogenous, property types, integrity constraints, scalability, evolving data



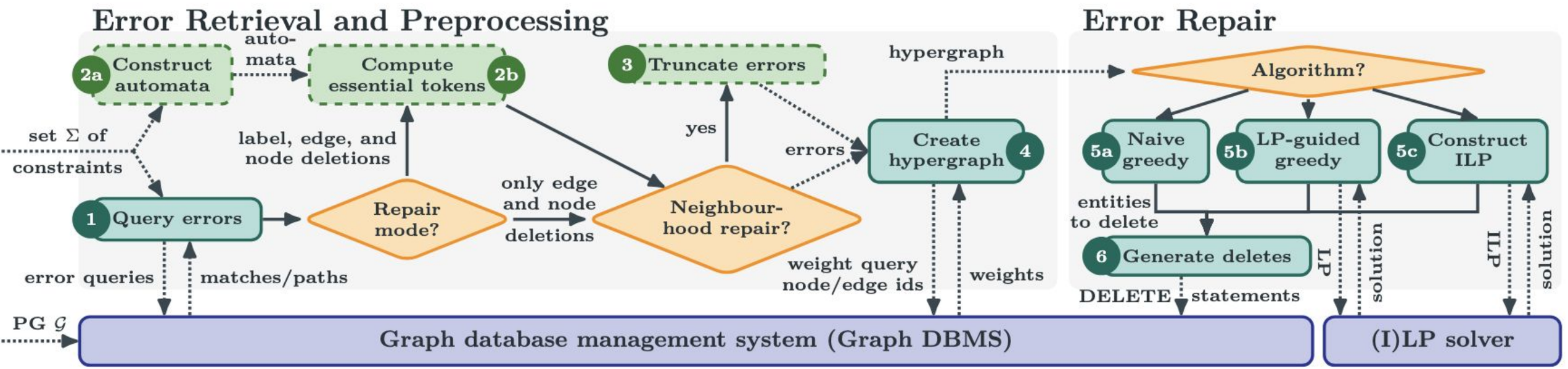
[RDF-PG24] K. Rabbani, M. Lissandrini, A. Bonifati, K. Hose: Transforming RDF Graphs to Property Graphs using Standardized Schemas. Proc. ACM Manag. Data 2(6): 242:1-242:25 (2024)

The S3PG system: schema <-> data

- **Schema Transformation** Phase (SHACL ⇒PG-Schema)
  - Each NodeShape becomes a PG-Schema node type with the same label(s) and inheritance hierarchy, while every PropertyShape is translated into either
    - (i) a property key on that node type (for single-valued literals), or
    - (ii) an edge type that records source/target node types, allowed datatypes, and min/max cardinalities.
- **Data-Transformation Phase** (RDF Triples ⇒Property Graph)
  - S3PG reads through the RDF triples only once.
  - For every resource, it creates a single node, aggregating all rdf:type triples into multi-labels.
    - if the object is another resource, an edge of the predicate's label is added
    - if the object is a literal, S3PG either stores it as an inline property or, when the schema allows heterogeneous or multi-valued data, as a dedicated value node linked by an edge.
- Inserts or deletes can be replayed **incrementally**, ensuring the property graph remains a monotonic, lossless reflection of its evolving RDF source.

Repairing Property Graphs seen as Transformations

- **Problem:** Repairing is the process of transforming a given property graph G into a property graph G' that satisfies all constraints; **Goal:** Covering complex PG constraints with recursive graph patterns; **Challenges:** Repair each error by deleting edges/renaming labels and minimize the number of deletions



[SBE-26] Christopher Spinrath, Angela Bonifati, Rachid Echahed: Repairing Property Graphs under PG-Constraints. Proc. VLDB Endow. 19(6): 1212-1225 (2026)

Repairing Errors generated by PG-Constraints (incl. PG-Keys)

PG-Constraints

- ✔ Designed for property graphs
- ● Expressive and extensive
  - ▶ since they are based on the standardized query language GQL for property graphs
- X No formal results

Important Features (of our fragment)

- ▶ Recursive patterns (also nested)
  `-[:references]->+` (One or more edges)
- ▶ Can match/filter w.r.t. **labels and properties** of **nodes and edges**
- ▶ Complex label expressions
  `(z:document & !important)`

Example

If a document references another, important document directly or indirectly (`MATCH` + `FILTER`) then the other document has a strictly lower access level. (`MANDATORY`)

```
MATCH (x:document)-[:references]->+(y:document & important)
FILTER x <> y
MANDATORY y.access_level < x.access_level
```

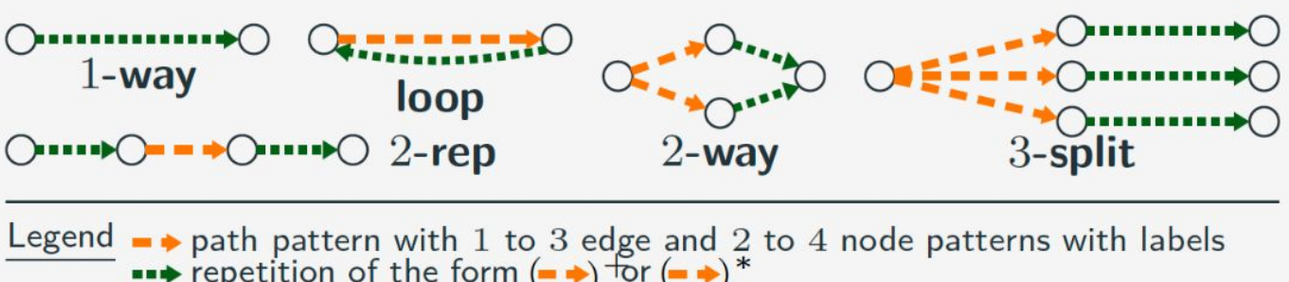
Highlights of experimental results

Datasets

Name	#Nodes	#Edges
ICIJ Property Graph	2 016 523	3 339 267
Italian Legislative Graph	411 787	1 117 528
Coreutils Code Property Graph	206 506	387 284
LDBC, scaling factor 0.3	940 322	5 003 201

Constraints

5 to 188 constraints per dataset/common pattern shape



Outcome Extensions

- Label deletions
  - Up to 59% less deletions
  - Up to $5\times$ runtime
- Neighbourhood repairs
  - Up to 62% faster
  - Competitive quality for short patterns
- Approximations
  - Up to 89% faster
  - Often good quality

Outline

- Property graphs in graph databases
- Graph transformations as primitives for:
  - data integration
  - metadata management
  - data cleaning
- **Graph transformations as a backbone for:**
  - causal analysis
  - causal model extraction and identification
  - counterfactual analysis

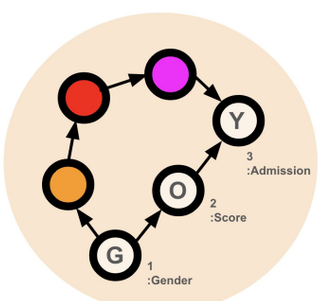
Graph transformations for causal analysis

- **Graphs + Causality are powerful but disconnected**.
  - Graphs model relationships and enable complex tasks, while causal DAGs capture cause–effect but they are not handled in graph systems.
- **Current gap: two isolated worlds**.
  - Causal inference relies on ad-hoc scripts, while graph databases focus on declarative querying — with no unified framework.
- **GO-Y vision: causality-driven graph databases**.
  - Integrate causal graphs into graph data systems with support for causal models, queries, integration, and versioning.

European Research Council

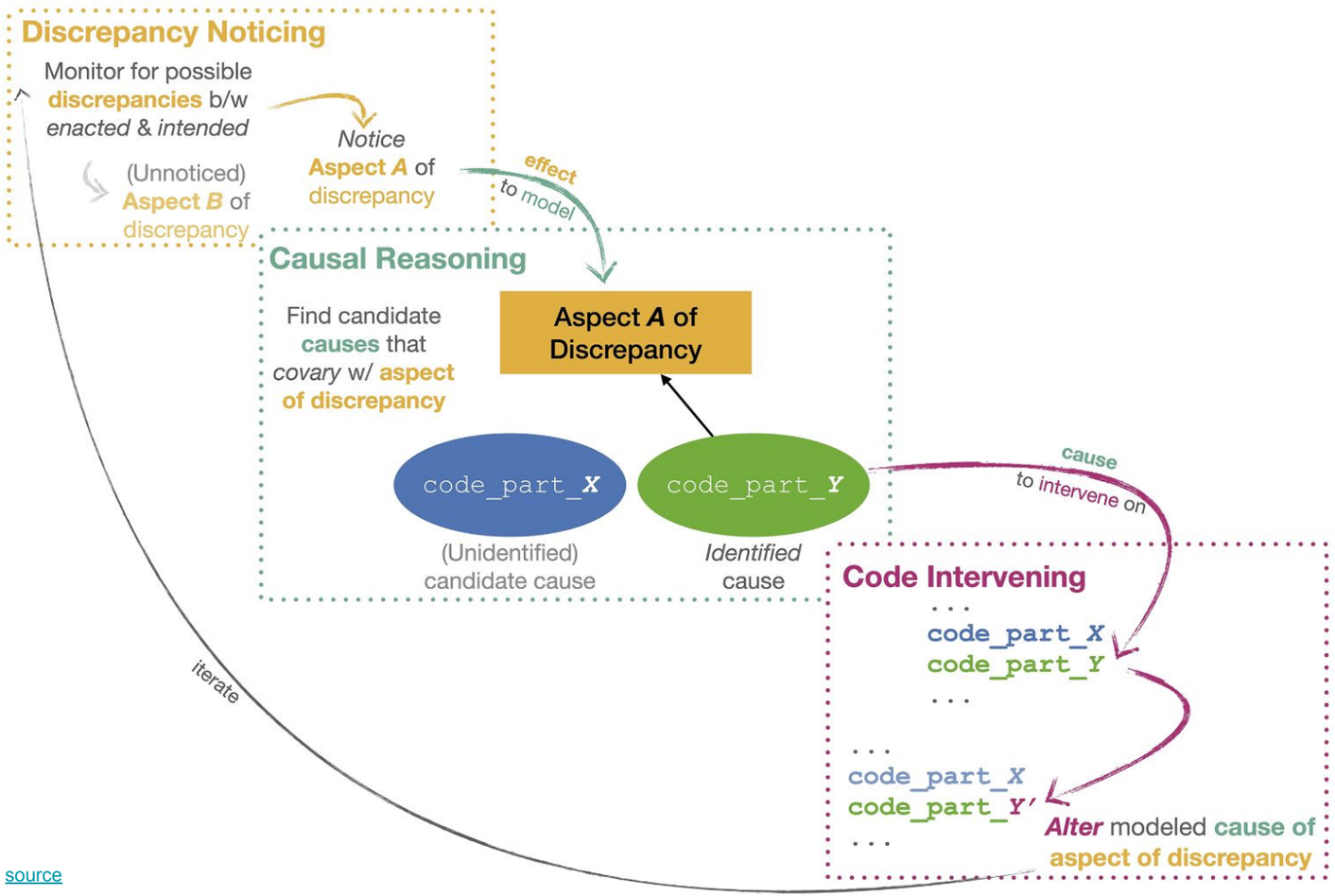
Established by the European Commission

ERC AdG GO-Y: Unifying Graph Databases and Causal Models



Causality in Real-World Applications

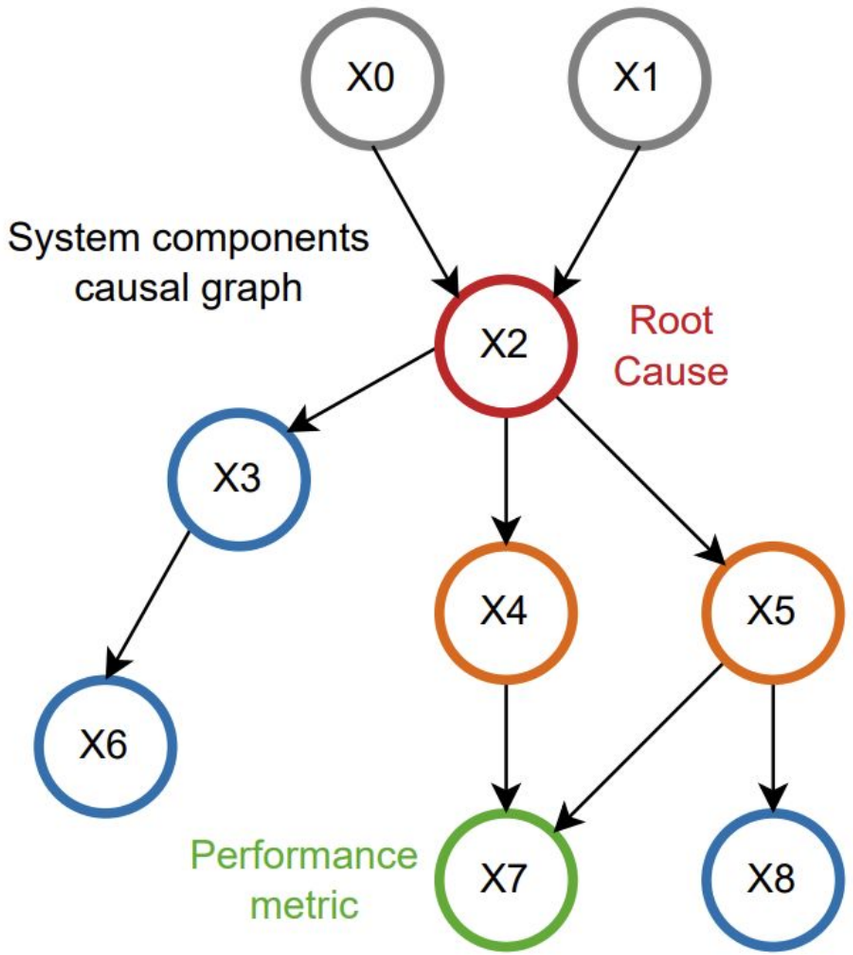
- **Debugging query results**: why is this answer returned?

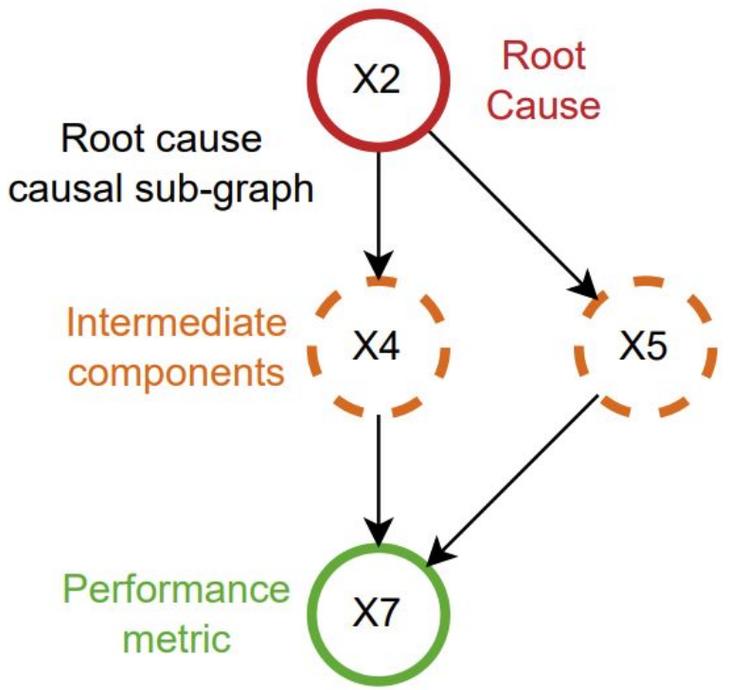


source

Causality in Real-World Applications

- **System diagnosis**: what is the root cause of failure?

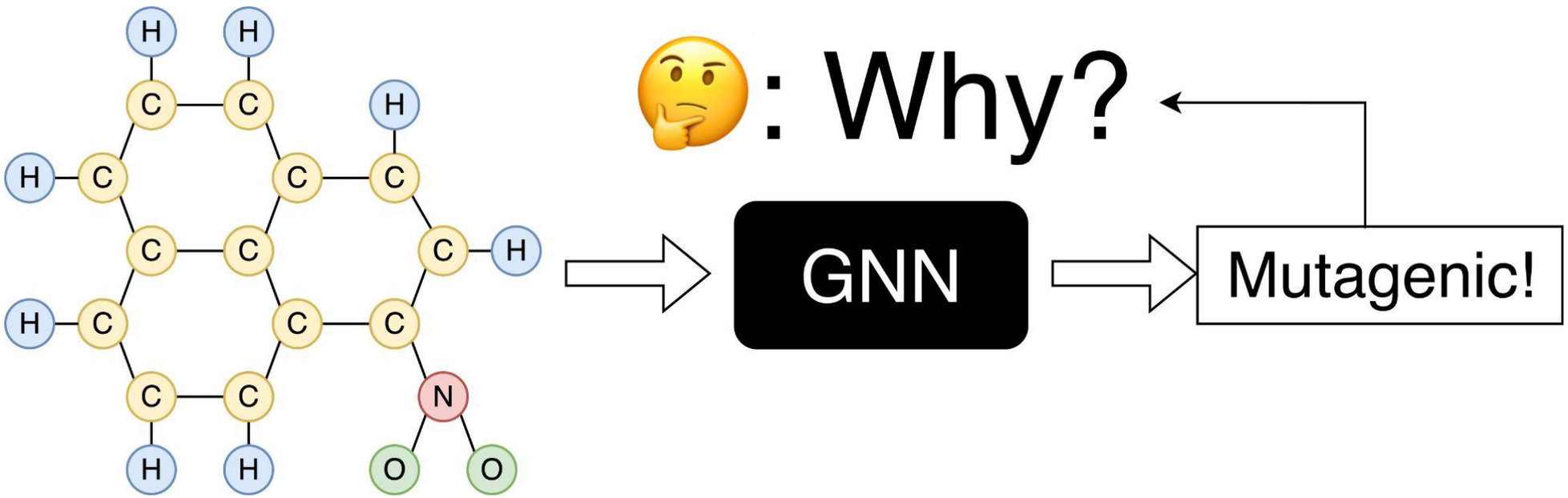




source

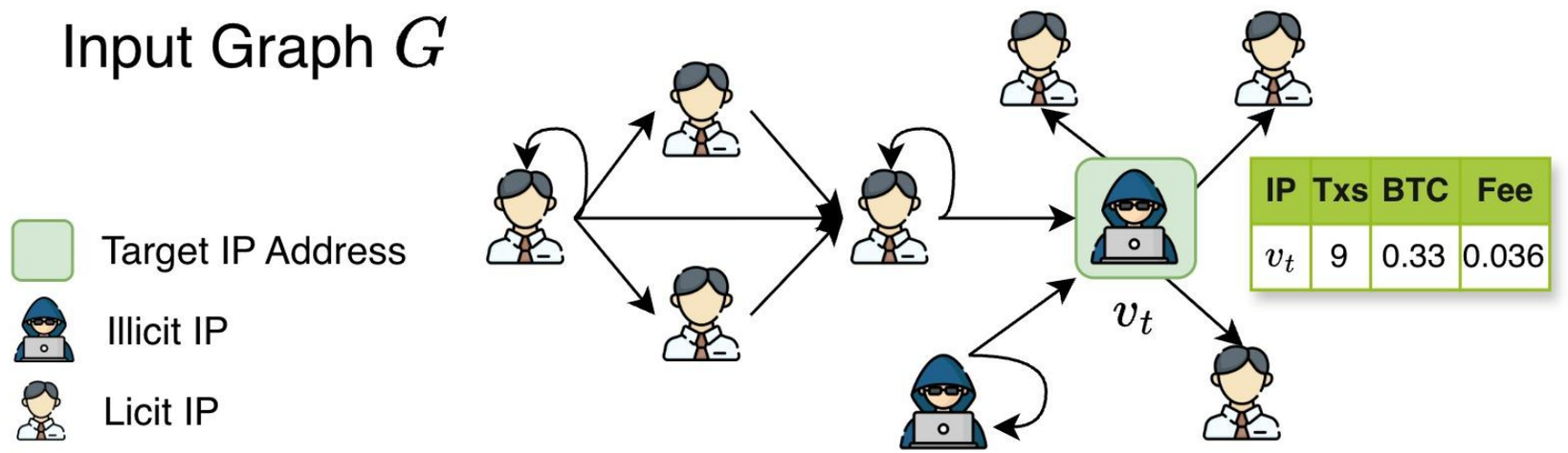
Causality in Real-World Applications

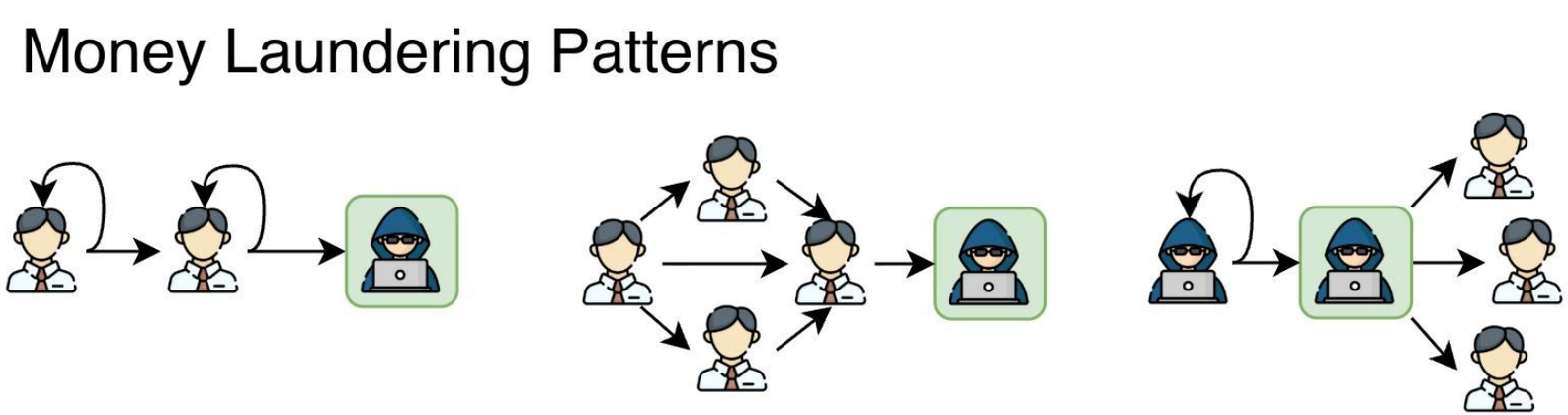
- **Explainable AI**: which factors caused the prediction?



Causality Analysis for Graph Data

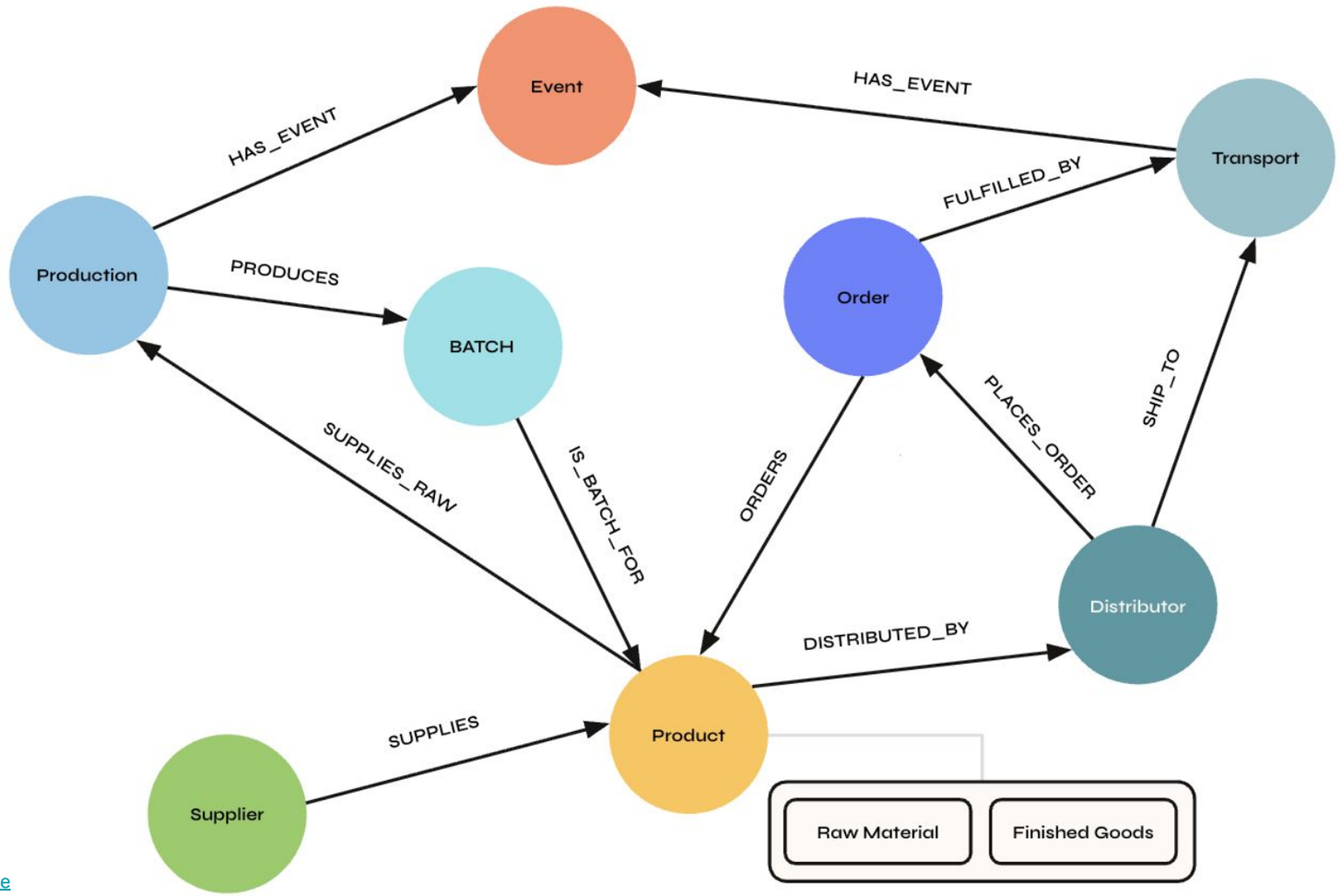
- **Fraud graphs**: which transfer edge caused the suspicious pattern?





Causality Analysis for Graph Data

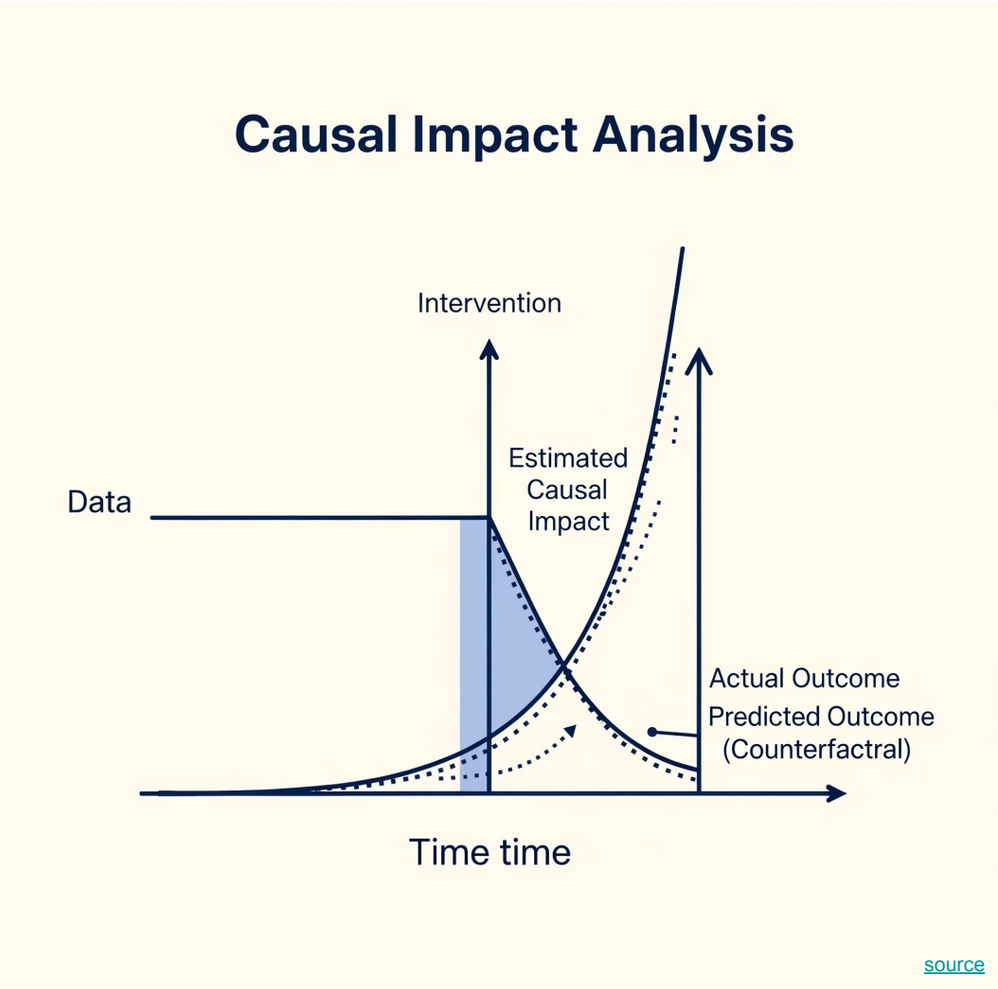
- **Supply-chain graphs**: which supplier/link causes disruption to spread?



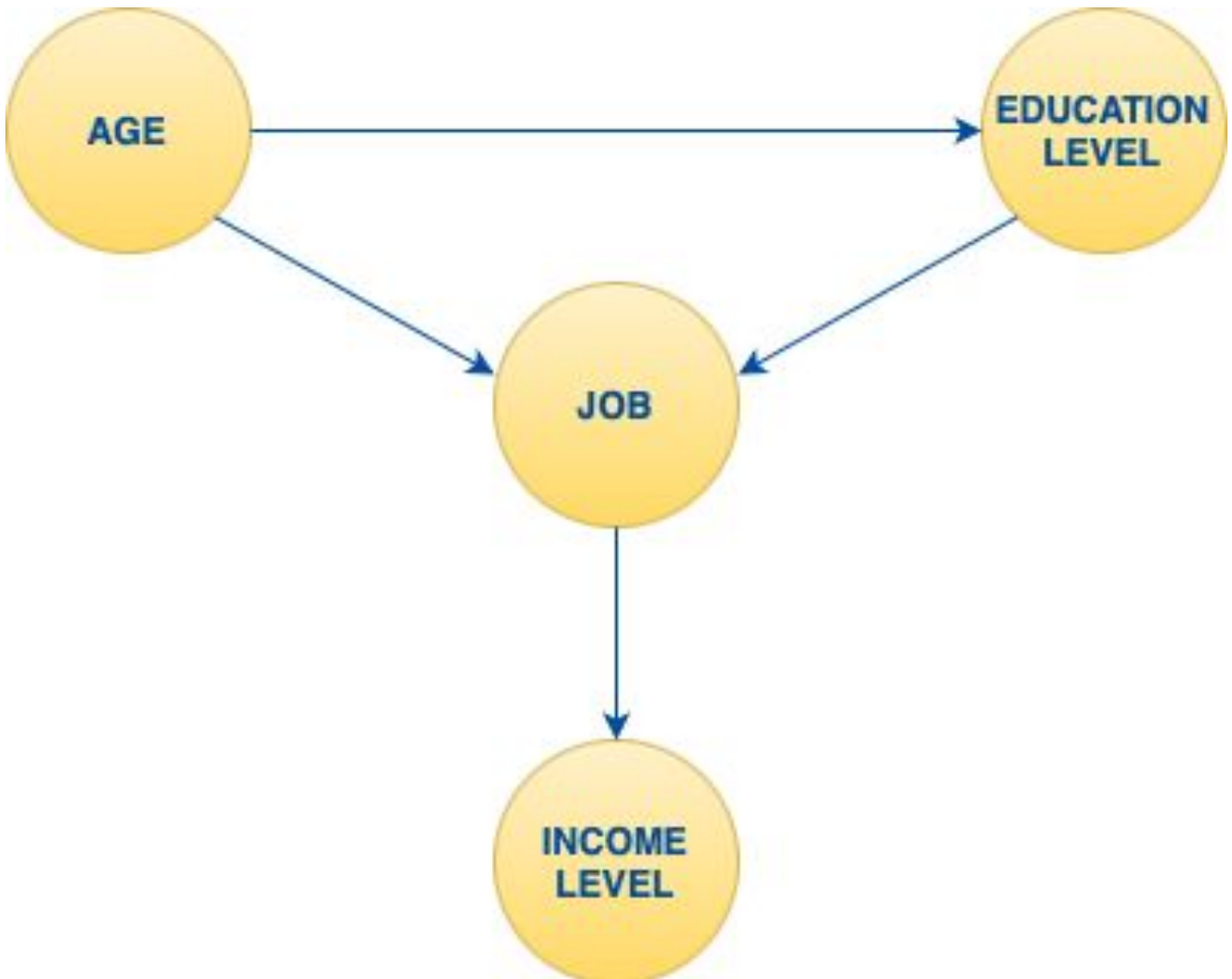
source

Causality in Real-World Applications

- **What-if counterfactual analysis**: how does the result change after an intervention?



Structural Causal Models and Causal Graphs



The first component of a SCM is the causal graph.

Typically, a causal graph is a DAG (directed acyclic graph).

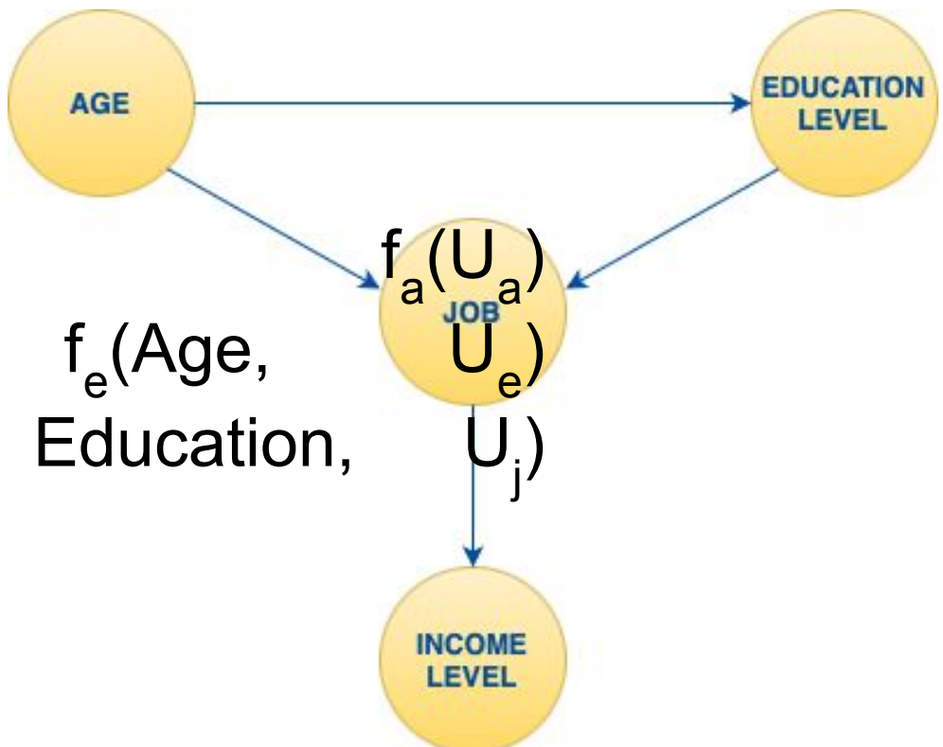
A generic edge $x \rightarrow y$ in the causal DAG identifies a direct causal relationship between the exposure x and the outcome y.

Structural Causal Models and Structural Equations

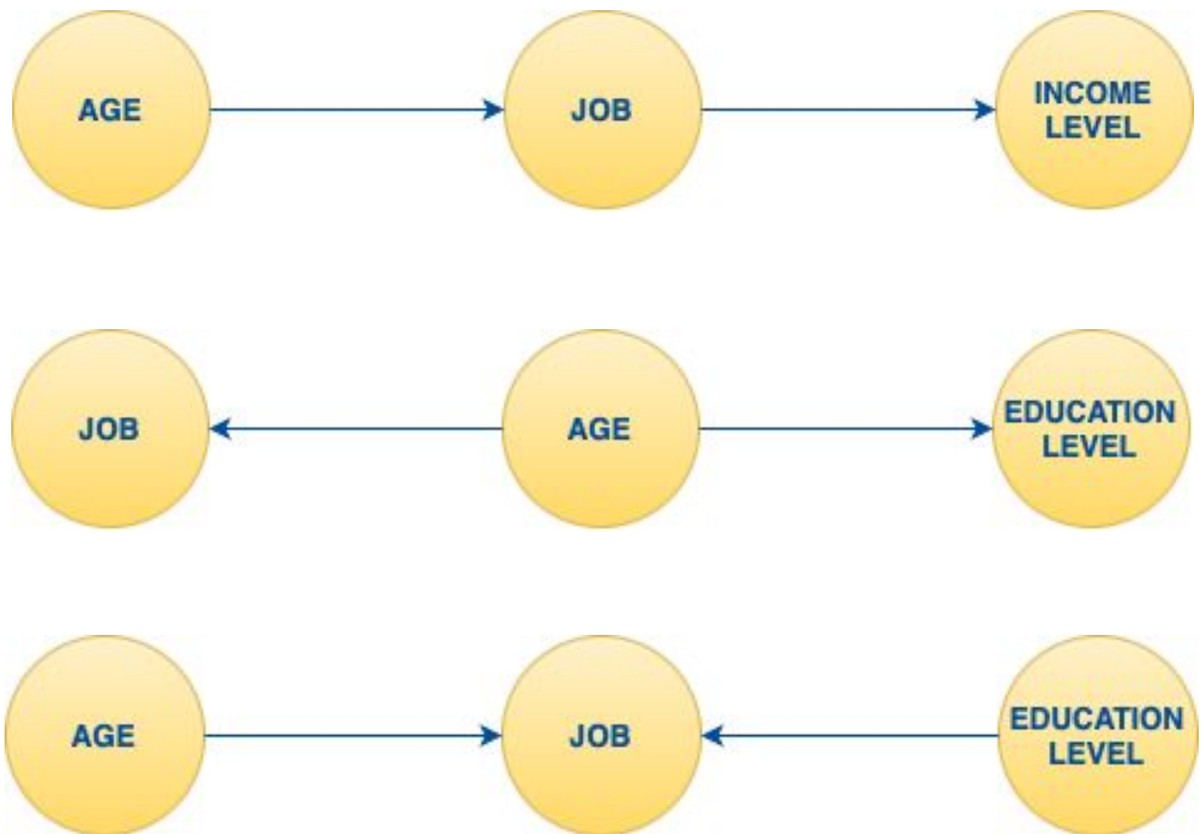
The second element of a SCM is its structural equations (SEs).

The SEs specify, for each variable, the identity of its causes and how they are functionally related.

$$\text{Age} = f_a(U_a)$$
$$\text{Education} = f_e(\text{Age}, U_e)$$
$$\text{Job} = f_j(\text{Age}, \text{Education}, U_j)$$
$$\text{Income} = f_i(\text{Job}, U_i)$$



Mediators, confounders and colliders



A **mediator** is an *intermediate node* through which the exposure affects the outcome.

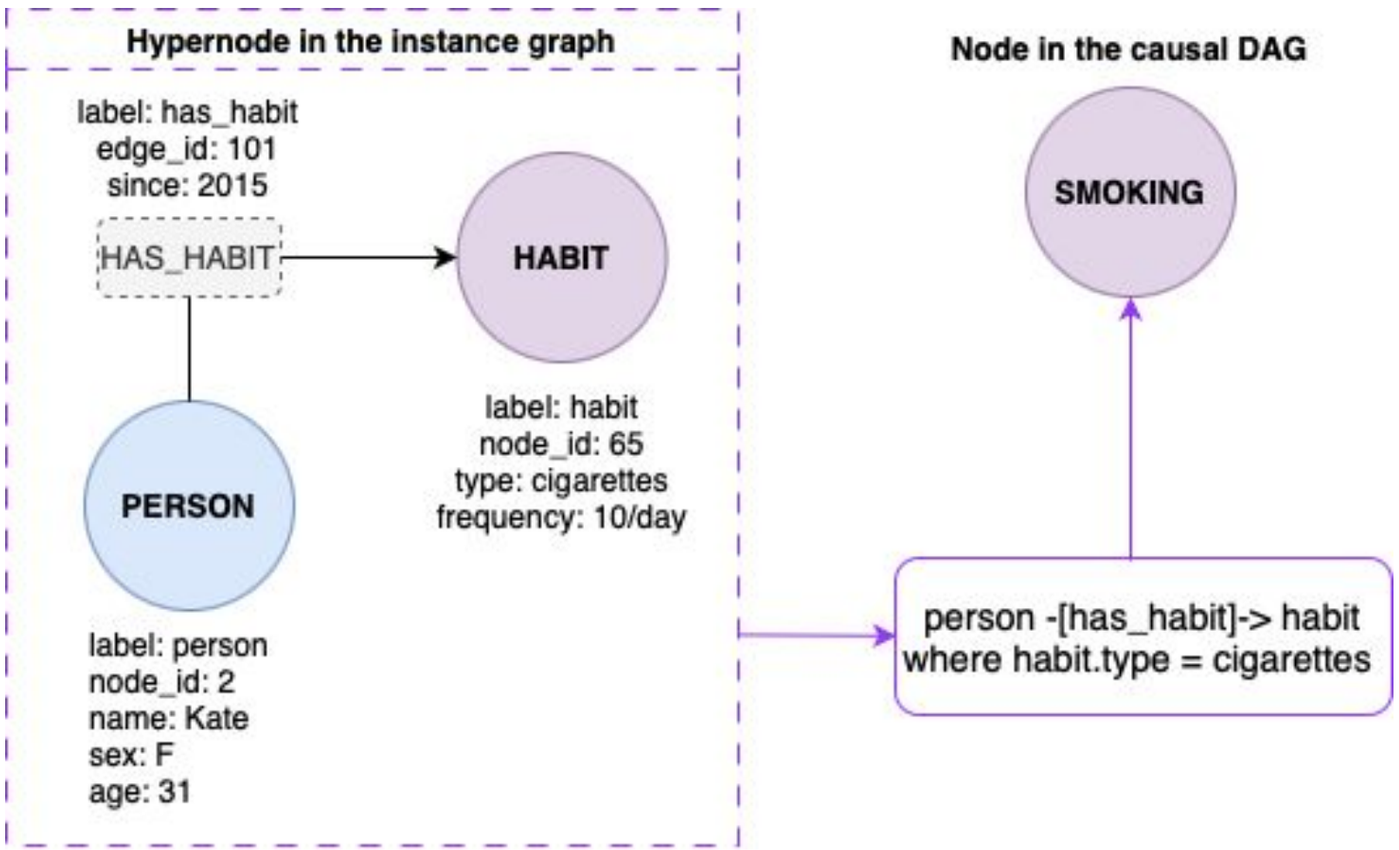
A **confounder** is a *common cause* of two variables.

A **collider** is a *common effect* of two variables.

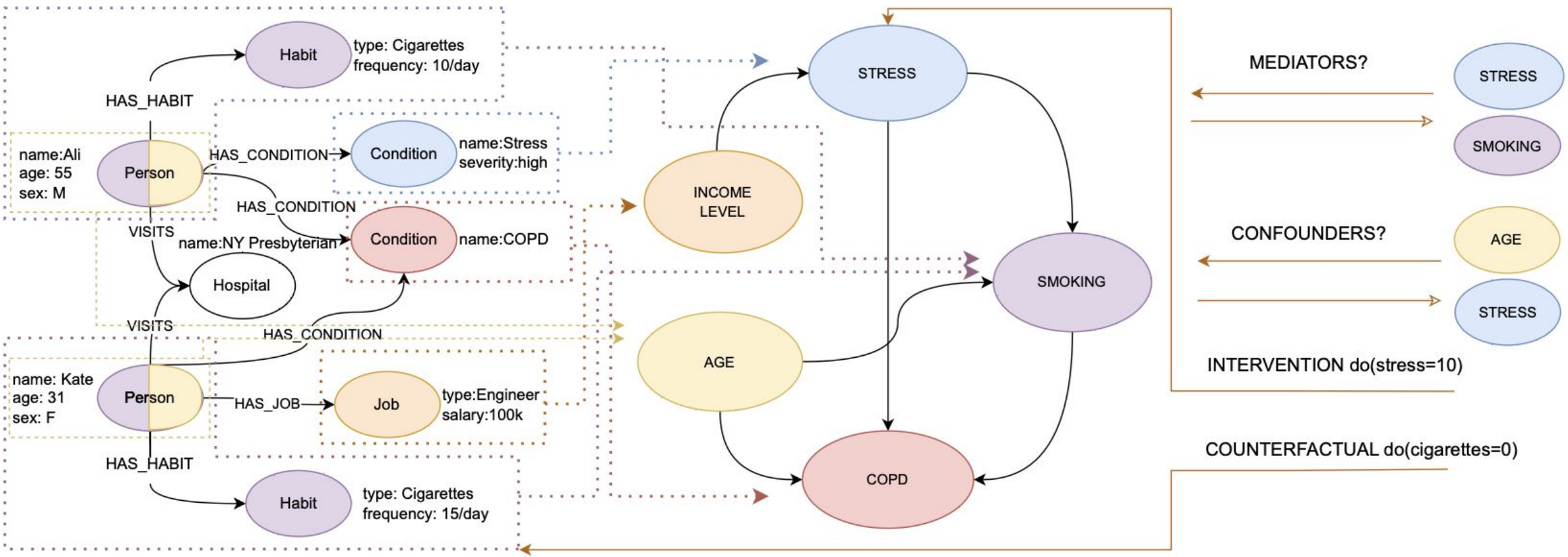
Mapping DAGs to the underlying graph database

- One of the current **limitations** of causal models are their limited connection to the observed data

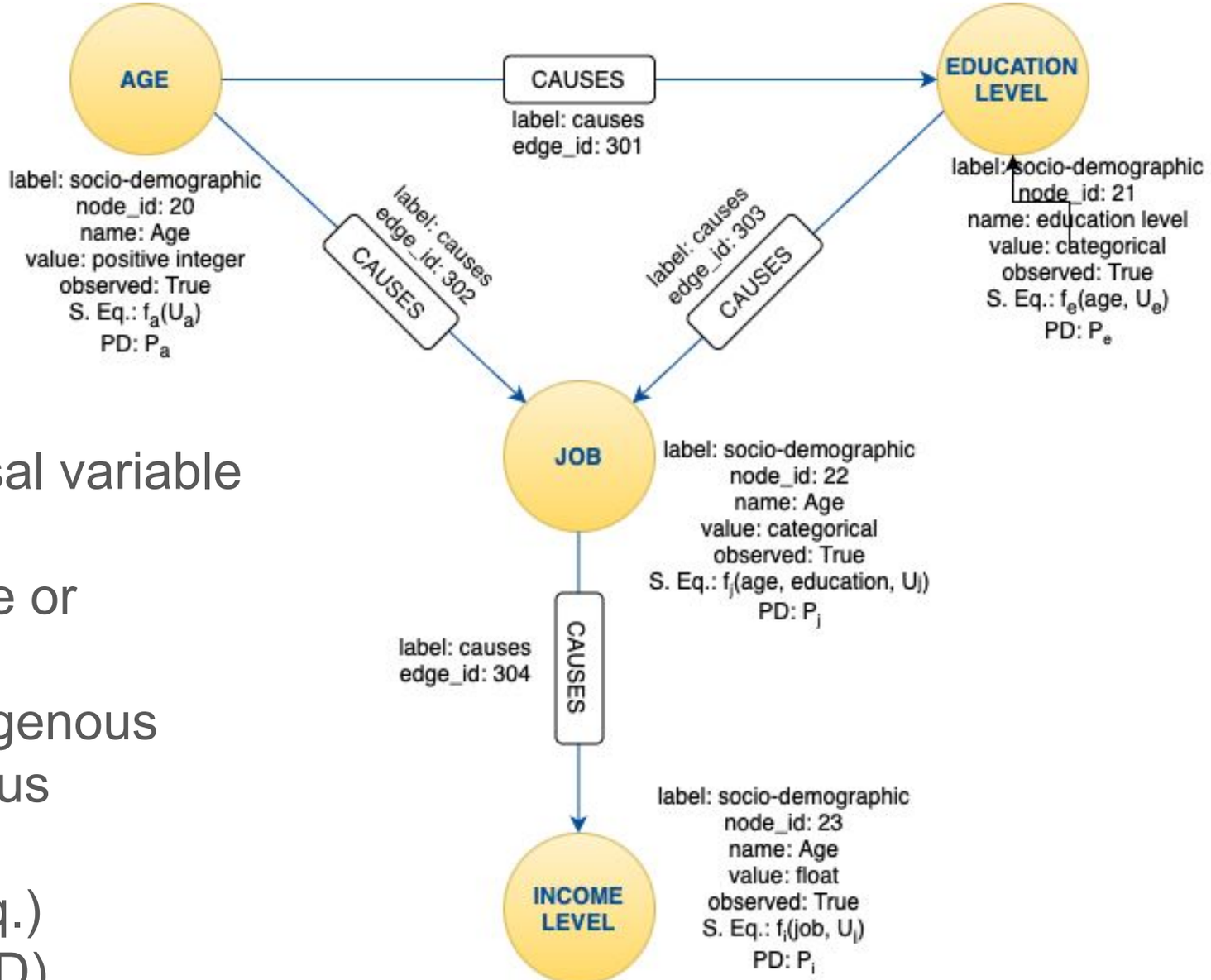
- A set of **graph patterns** with causal variables is linked to one or more subgraphs in the observed property graph, modeled as **hypernodes**.



Mappings between property graph and causal DAG



The causal DAG modeled as a property graph



In each node representing a causal variable we can store as properties:

- the variable type (quantitative or qualitative)
- whether the variable is endogenous (observed=True) or exogenous (observed=False)
- the structural equation (S. Eq.)
- the probability distribution (PD)

Current limitations of PGs and query languages

Simple analyses can be performed using existing openCypher/GQL constructs.

Complex causal tasks that require navigating between the causal DAG and the observational data necessitate additional constructs that are currently not present in current languages.

RESEARCH QUESTION:

How can we empower graph databases, property graph models and queries with causal analysis capabilities?

Causal Analysis with Graph Databases

Main contributions:

- Formalization of a causal property graph model augmented with hypernodes.
- Envision extensions of current graph query languages with causal operators.
- Prototype implementation.

*The vision will allow to **make graph databases aware of causal knowledge** and pave the way to data-driven personalized decision-making in several scientific fields.*

[PPA25] Amedeo Pachera, Mattia Palmiotto, Angela Bonifati, Andrea Mauri *What if: Causal Analysis with Graph Databases*, Proc. VLDB Endow. 18 (2025)

What if system vs. related approaches and libraries

Task	PyWhy	CausalAI	Graph DBMS	WhatIf
Causal Discovery	✓	✓	✗	✓
Causal DAG Mining	✗	✗	✗	✓
Causal Path Finding	~	~	✓	✓
Causal DAG Manipulation	✗	✗	✗	✓
Causal DAG Maintenance	✗	✗	✗	✓
Causal DAG Transportability	✗	✗	✗	✓
Intervention	✓	✓	~	✓
Counterfactual	✓	✓	~	✓

Table 1: Comparison of tools on causal inference tasks.

PG transformation for causal variable extraction

The Cypher query on the left creates a node for each matched path: a node (SMOKING) is extracted twice from the property graph instance.

The EXTRACT operator on the right would allow to easily express the causal variable extraction by abstracting out the details of hypernodes and graph transformations.

```
MATCH (p:Person)-[:HAS_HABIT]->(h:Habit),
(p)-[:HAS_CONDITION]->(c:Condition)
WHERE c.name = "COPD" AND h.type="Cigarettes"
WITH h, c
MERGE (h)-[:BELONGS]->(x:SMOKING)-->(y:COPD)<-[:BELONGS]-(c)
```

```
MATCH (p:Person)-[:HAS_HABIT]->(h:Habit),
(p)-[:HAS_CONDITION]->(c:Condition)
WHERE c.name = "COPD" AND h.type="Cigarettes"
EXTRACT (x:SMOKING)-->(y:COPD)
```

Causal PG queries for mediator analysis

The advantages of relying on PGs come from the possibility of accessing directly to the instances for further analysis, since the DAG and the instances belong to the same data model.
The following query retrieves the graph instance corresponding to the mediator between the variable INCOME LEVEL and COPD in the case of Kate:

```
MATCH (p)--(j:Job)-[:BELONGS]->(a:INCOME LEVEL)-->(m)-->(c:COPD)
WHERE p.name="Kate" WITH m
MATCH ALL instance=(m)<-[:BELONGS]-(n1)-[]-{*}(n2)-[:BELONGS]->(m)
RETURN instance
```

Computes all paths including mediator m [ABG25]

[ABG25] R. Angles, A. Bonifati, R. García, D. Vrgoc: Path-based Algebraic Foundations of Graph Query Languages. EDBT 2025: 783-795

Conditional probability query

We could retrieve and store as node property the conditional probability P(Y|X) by identifying the instances in the underlying graph linked to the variable X and those linked to the variable Y, and pass them as argument to a PROBABILITY() operator extending GQL/Cypher:

```
MATCH (a:X)-[r]->(b:Y)
WITH a, b
MATCH ALL instancesX=(a)<-[BELONGS]-(n1)-[]-{*}(n2)-[BELONGS]->(a)
MATCH ALL instancesY=(b)<-[BELONGS]-(n1)-[]-{*}(n2)-[BELONGS]->(b)
RETURN PROBABILITY(instanceY,instanceX)
```

Causal path queries

The query on the left identifies all the chains (patterns like **(a)->(m)->(b)**) in the graph and returns their variables.

The query on the right only returns those chains such that there is no confounder (patterns like **(a)<-(c)->(b)**) between any two variables in the chain.

```
MATCH (a:X)-->(m)-->(b:Y)
RETURN a as exposure, m as Mediator,
       labels(m) as MediatorVariable, b as outcome
```

```
MATCH (a:X)-->(m)-->(b:Y)
WHERE NOT EXISTS { MATCH (a)<--(c)-->(b) }
AND NOT EXISTS { MATCH (a)<--(c)-->(m) }
AND NOT EXISTS { MATCH (m)<--(c)-->(b) }
RETURN a as Exposure, m as Mediator,
       b as Outcome
```

Causal PG queries and transformations

Interventional queries focus on predicting the effects of specific interventions.

For example: *"What will be the risk of contracting COPD from smoking 10 cigarettes a day?"* This question can be answered by retrieving the structural equations from the node properties:

```
MATCH SHORTEST 1 (a:SMOKING)-->{*}(m)-->(b:COPD)
RETURN a.s_eq as s_s, m.s_eq as s_m, b.s_eq as s_l
```

We need to:

- extend the query with a new operator DO-CALCULUS([values],[equations]) to the variable SMOKING leading to a PG transformation
- evaluate the impact of this intervention on the variable COPD.

Counterfactual queries and transformations

They explore “what if” scenarios about hypothetical alternatives.
In addition to forcing a variable to take a specific value, observed evidence is taken into account to determine the consequence of the counterfactual.

```
MATCH (h:Habit)<-[HAS_HABIT]-(p:Person)-[HAS_CONDITION]->(c:Condition)
WHERE p.name="Ali" AND h.type="Cigarettes" AND c.name="COPD"
RETURN h.frequency as f, c.severity as s
WITH f,s
MATCH SHORTEST 1 path=(a:SMOKING)-->(b:COPD)
RETURN DO-CALCULUS([SMOKING=f,COPD=s],[b.seq]) as noise
WITH noise
MATCH SHORTEST 1 path=(a:SMOKING)-->(b:COPD)
RETURN DO-CALCULUS([SMOKING=0,ϵ_COPD=noise],[b.seq])
```

Causal Hypergraphs performances

#Nodes (#C.Var.)	Extract	Mediators	Confound.	Colliders	Interv.	Counterf.
5k (5)	0.780	**0.003** (0.648)	**0.004** (0.656)	**0.003** (2.218)	0.194 (0.113)	**0.016** (0.017)
25k (25)	1.236	**0.005** (3.660)	**0.004** (8.246)	**0.012** (24.003)	0.801 (**0.104**)	0.055 (**0.020**)
50k (50)	4.074	**0.009** (25.855)	**0.013** (14.095)	**0.004** (39.596)	10.01 (**2.954**)	0.926 (**0.289**)
50k (5)	1.032	**0.010** (2.917)	**0.008** (4.891)	**0.004** (11.358)	**0.109** (0.757)	**0.023** (0.048)
250k (25)	1.277	**0.009** (37.431)	**0.009** (82.466)	**0.012** (230.949)	**0.838** (1.161)	0.056 (**0.044**)
500k (50)	3.974	**0.011** (367.681)	**0.011** (771.086)	**0.00**4 (2274.338)	9.951 (**3.262**)	0.935 (**0.293**)
500k (5)	1.028	**0.008** (20.081)	**0.007** (33.072)	**0.004** (64.904)	**0.106** (4.233)	**0.020** (0.472)
2500k (25)	1.207	**0.006** (93.703)	**0.006** (537.99)	**0.003** (961.93)	0.834 (**0.314**)	0.053 (**0.042**)
5000k (50)	3.989	**0.007** (162.502)	**0.005** (1242.618)	**0.004** (1842.14)	9.831 (**0.653**)	0.892 (**0.104**)

Table 2: Runtimes (s) by query type and DAG size; bracketed values show runtimes without the causal model, and best runtimes are in **bold**.

d-separation is a key step of many causal AI tasks

Confounder identification: used to determine which variables should be conditioned on to block backdoor paths and obtain an unbiased estimate of a causal effect.

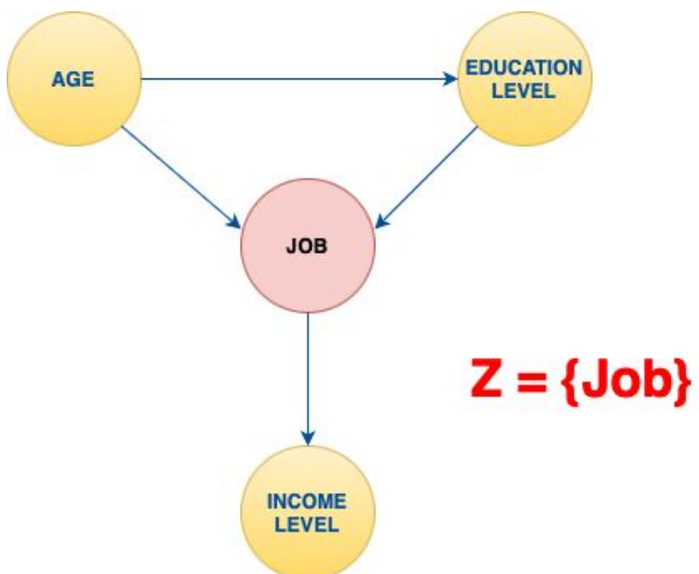
Conditional independence testing: used to identify which conditional independence relationships should hold in a causal graph, allowing researchers to test whether the graph is consistent with observed data.

Causal effect identification: used to verify whether a set of variables satisfies criteria such as the backdoor or frontdoor criterion, enabling valid estimation of causal effects from observational data.

d-separation in Causal Analysis

The variables ***X*** and ***Y*** are **d-separated** conditional on a set of variables ***Z*** if they are independent when conditioning on ***Z***.
To check d-separation, we must compute *all paths* between ***X*** and ***Y*** and determine whether they are blocked or not. If all paths are blocked, ***X*** and ***Y*** are d-separated. By contrary, they are said **d-connected**.

When conditioning on Job, Age is d-separated from Income and d-connected to Education.



d-separation as a graph transformation in graph DBs

The following Cypher query answers the question:

*"Are **Age** and **Income** d-separated conditional on **Job**?"*

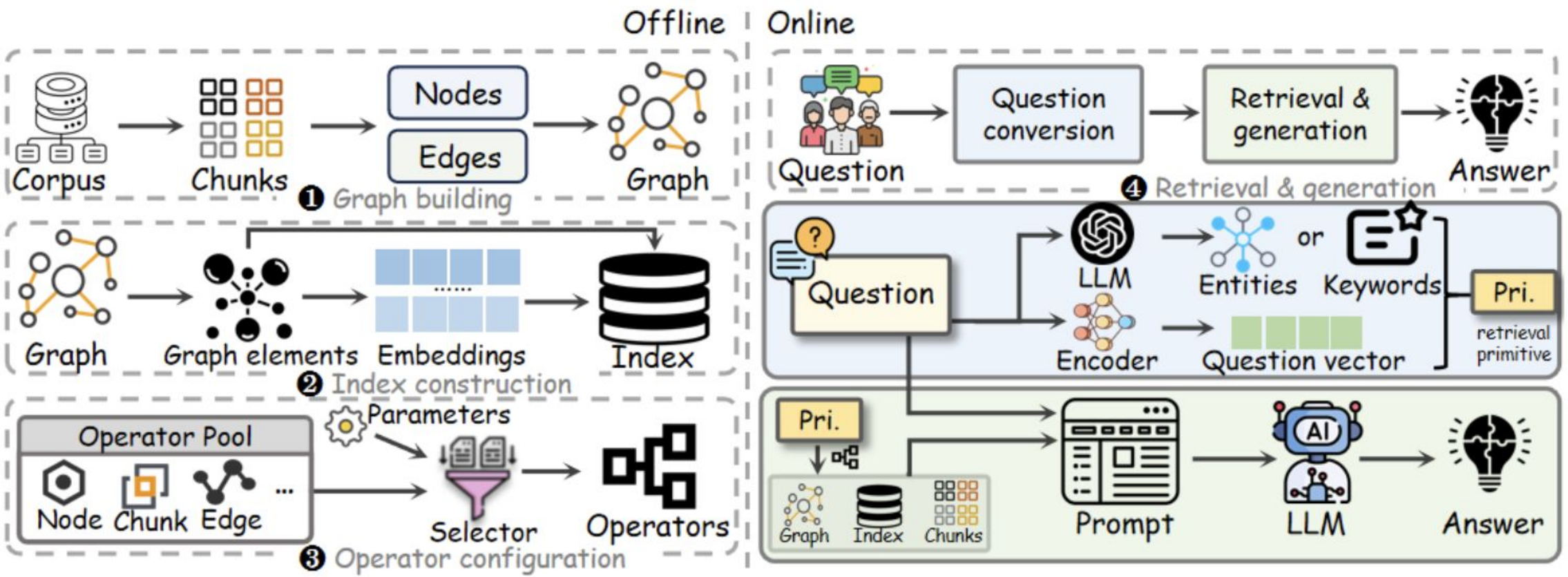
```
WITH ["Job"] AS Z_names
MATCH (a)-[r1]-(b)-[r2]-(c)
WHERE
EXISTS { MATCH (a)-[r1]->(b)-[r2]->(c) WHERE b.name IN Z_names }
OR EXISTS { MATCH (a)<-[r1]-(b)<-[r2]-(c) WHERE b.nam :e IN Z_names }
OR EXISTS { MATCH (a)<-[r1]-(b)-[r2]->(c) WHERE b.name IN Z_names }
OR EXISTS { MATCH (a)-[r1]->(b)<-[r2]-(c) WHERE NOT EXISTS  {MATCH (b)-[]->*(d)
WHERE d.name IN Z_names }}
WITH COLLECT([r1, r2]) AS B
MATCH p = (X {name: "Age"})-[]-+(Y {name: "Income"})
AND NONE(n IN nodes(p) WHERE size([m IN nodes(p) WHERE m = n]) > 1)
WITH p, relationships(p) AS rels, B
WHERE ALL(i IN RANGE(0, size(rels)-2) WHERE NOT [rels[i], rels[i+1]] IN B)
RETURN COUNT(p) = 0 AS d_separated
```

The query:

- *finds all the three-node patterns (a)-(b)-(c) in the DAG which block a path;*
- *finds all the paths between Age and Income;*
- *discards all the blocked paths;*
- *if no path is left, returns True, else False.*

[PBM26] M. Palmiotto, A. Bonifati, A. Mauri: Computing d-separation on Causal Property Graphs (under submission)

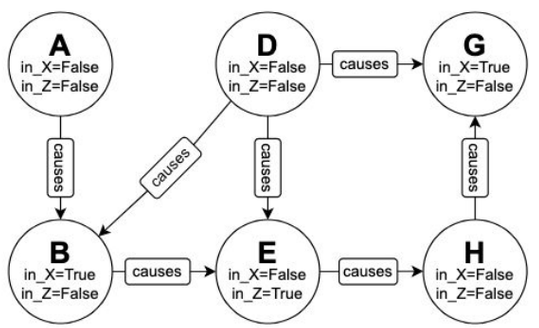
Graph Retrieval-Augmented Generation (RAG)



[BFWZ26] A. Bonifati, Y. Fang, S. Wang, Y. Zhou. Graph-Based Retrieval-Augmented Generation: Applications,Challenges, Solutions, and Opportunities. Tutorial at VLDB 2026

Concluding Remarks

- Property graphs are the backbones of graph databases
- Despite standardization, query languages miss primitives for property graph transformations
- Property graph transformations enable graph data integration tasks while guaranteeing efficiency and scalability
- They also serve as key tools for causal analysis
  - For intervention operations on graph databases
  - For What-if counterfactual graph queries
  - For query and AI model explainability



Thank you for your attention

Angela Bonifati

Angela.Bonifati@liris.cnrs.fr

European Research Council

Established by the European Commission

Prior Work on relational causal queries

- Previous work [SPK20] has studied how to specify causal queries using Datalog-like rules.
- Other studies [GGR22,SHG23] have extended this approach to include counterfactual relational queries and hypothetical reasoning.
- Another line of research [YCS23] has investigated data completeness in causal DAGs, considering information derived from multiple tuples within relational tables.

[SPK20] Babak Salimi, Harsh Parikh, Moe Kayali, Lise Getoor, Sudeepa Roy, and Dan Suciu. 2020. *Causal Relational Learning*. In Proceedings of the 2020 ACM SIGMOD
[GGR22] Sainyam Galhotra, Amir Gilad, Sudeepa Roy, and Babak Salimi. 2022. *HypeR: Hypothetical Reasoning With What-If and How-To Queries Using a Probabilistic Causal Approach*. In SIGMOD '22: International Conference on Management of Data.
[SHG23] Fangzhu Shen, Kayvon Heravi, Oscar Gomez, Sainyam Galhotra, Amir Gilad, Sudeepa Roy, and Babak Salimi. 2023. *Causal What-If and How-To Analysis Using HypeR*. In 39th IEEE International Conference on Data Engineering, ICDE 2023.
[YCS23]Brit Youngmann, Michael J. Cafarella, Babak Salimi, and Anna Zeng. 2023. *Causal Data Integration*. Proc. VLDB Endow. 16, 10 (2023), 2659–2665.

Prior Work on causal explanations for queries and repairs

- Other works [MGM10,YCG24] have applied causal reasoning for query explainability to identify the underlying factors behind specific query outcomes.

- Certain studies [BB17a, BB17b] have demonstrated that causal explanations for query answers are connected to database repairs, consistency-based diagnosis, Datalog abduction, view-update problems

[MGM10] Alexandra Meliou, Wolfgang Gatterbauer, Katherine F. Moore, and Dan Suciu. 2010. *The Complexity of Causality and Responsibility for Query Answers and non-Answers*. Proc. VLDB Endow. 4, 1 (2010), 34–45.
[YCG24] Brit Youngmann, Michael Cafarella, Amir Gilad, and Sudeepa Roy. 2024. *Summarized Causal Explanations For Aggregate Views*. Proc. ACM Manag. Data 2, 1, Article 71 (March 2024), 27 pages.
[BB17a] Leopoldo E. Bertossi and Babak Salimi. 2017. *Causes for query answers from databases: Datalog abduction, view-updates, and integrity constraints*. Int. J.Approx. Reason. 90 (2017), 226–252.
[BB17b] Leopoldo E. Bertossi and Babak Salimi. 2017. *From Causes for Database Queries to Repairs and Model-Based Diagnosis and Back*. Theory Comput. Syst. 61, 1 (2017), 191–232.