Schema Discovery

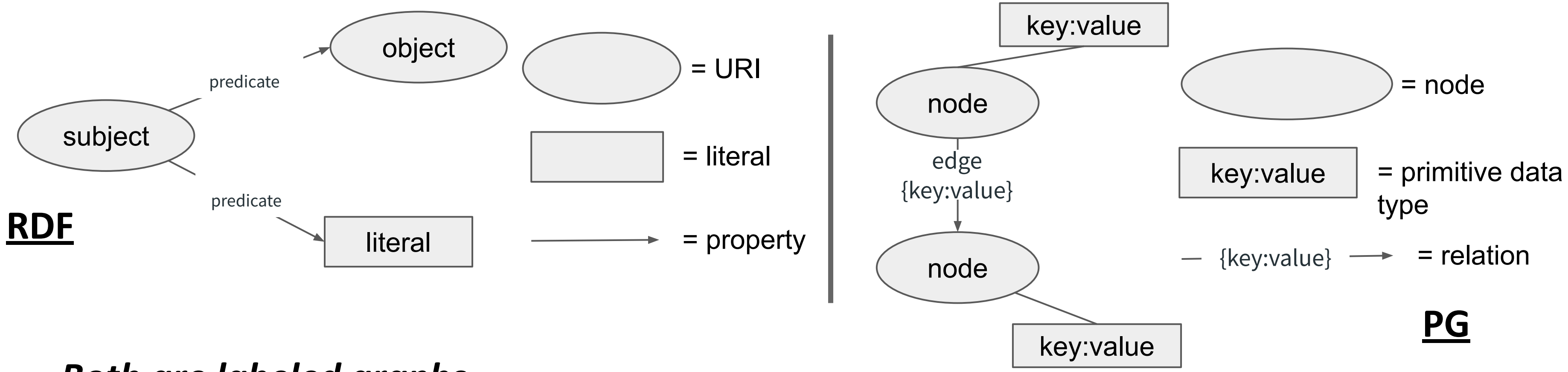
CS-562 Lab 5

Sophia Sideri

Preliminaries

RDF & Property Graphs:

- **RDF**: subject–predicate–object triples, semantic interoperability via ontologies.
- **PGs**: nodes/edges with labels and key–value properties; flexible, used in industry.



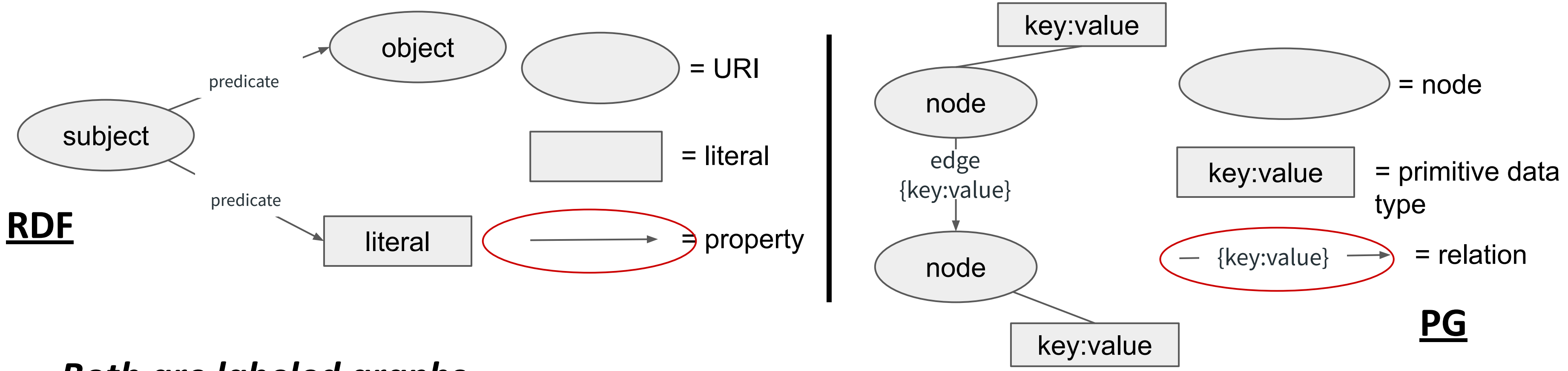
Both are labeled graphs.

Key difference: RDF edges (triples) can't hold properties; PG edges can.

Preliminaries

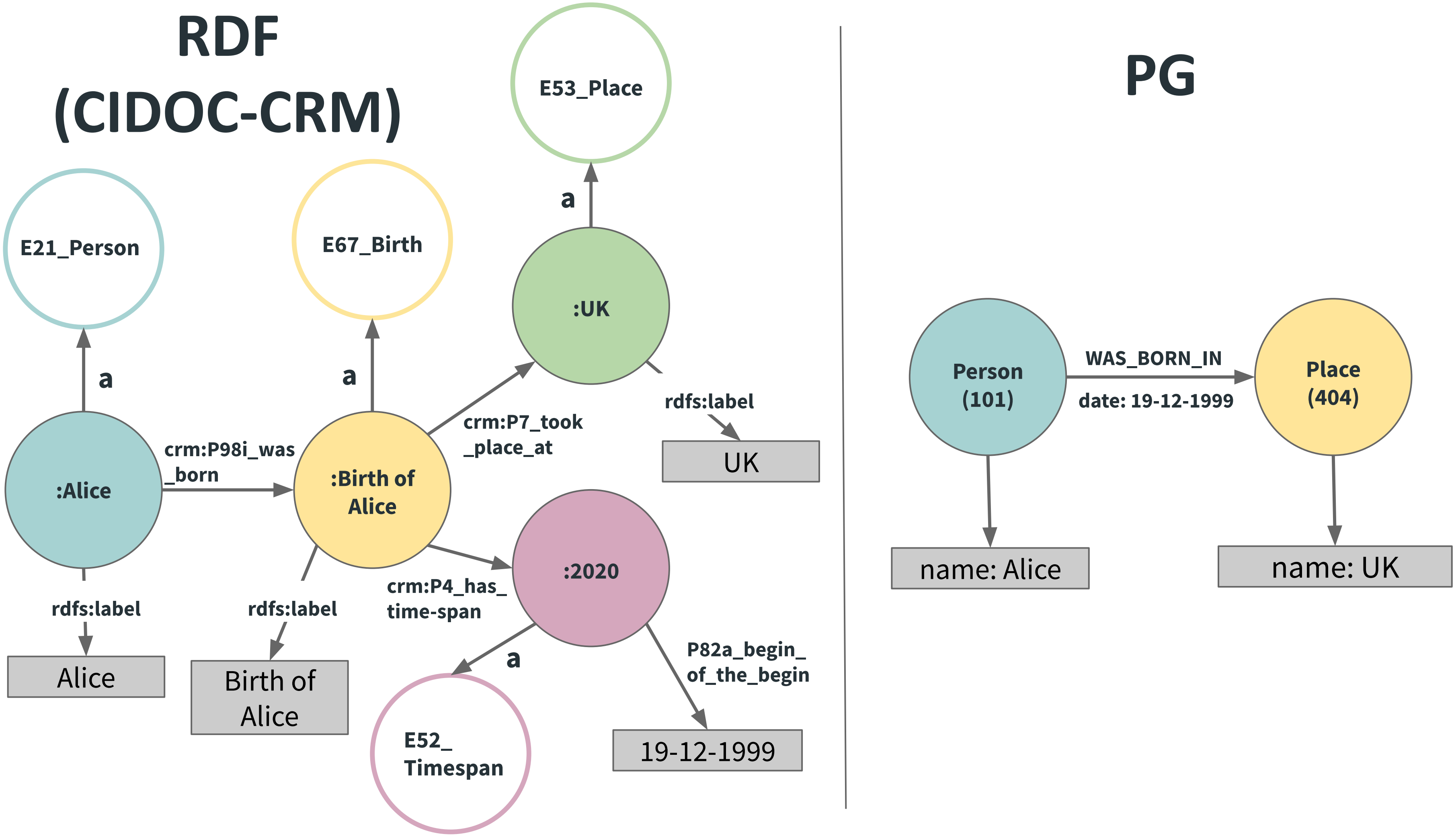
RDF & Property Graphs:

- **RDF**: subject–predicate–object triples, semantic interoperability via ontologies.
- **PGs**: nodes/edges with labels and key–value properties; flexible, used in industry.



Both are labeled graphs.

Key difference: RDF edges (triples) can't hold properties; PG edges can.



RDF
(CIDOC-CRM)
E53_Place
a
E21_Person
E67_Birth
:UK
a
a
rdfs:label
crm:P7_took
_place_at
UK
crm:P98i_was
_born
:Alice
:Birth of
Alice
:2020
crm:P4_has_
time-span
rdfs:label
rdfs:label
Alice
Birth of
Alice
a
P82a_begin_
of_the_begin
E52_
Timespan
19-12-1999
PG
Person
(101)
WAS_BORN_IN
date: 19-12-1999
Place
(404)
name: Alice
name: UK

Schema Discovery Problem in PGs

*"Given a property graph of **arbitrary size and structure**, with **missing type** information, **heterogeneous properties**, and **frequent updates**, infer the schema graph efficiently and accurately."*

Challenges:

Label heterogeneity and ambiguity	Evolving datasets
Efficiency	Schema constraint level

Ambiguities in Schema Discovery in PGs

In real datasets we might have several difficult cases to identify types:

Case 1: Many labels assigned to each instance

Case 2: Different labels referring to the semantically same entity

Case 3: Missing labels to some instances

Case 4: No labels to all instances and inconsistent properties

Ambiguities in Schema Discovery in PGs

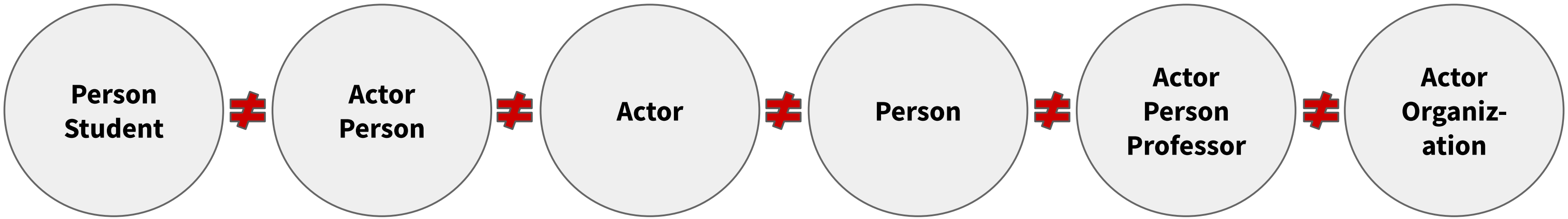
In real datasets we might have several difficult cases to identify types:

Case 1: Many labels assigned to each instance

Case 2: Different labels referring to the semantically same entity

Case 3: Missing labels to some instances

Case 4: No labels to all instances and inconsistent properties



We cannot assume that these are the same type

Ambiguities in Schema Discovery in PGs

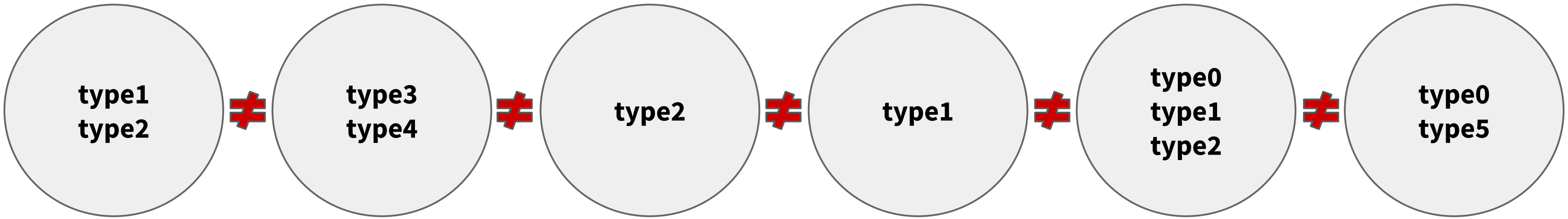
In real datasets we might have several difficult cases to identify types:

Case 1: Many labels assigned to each instance

Case 2: Different labels referring to the semantically same entity

Case 3: Missing labels to some instances

Case 4: No labels to all instances and inconsistent properties



We cannot assume that these are the same type

Ambiguities in Schema Discovery in PGs

In real datasets we might have several difficult cases to identify types:

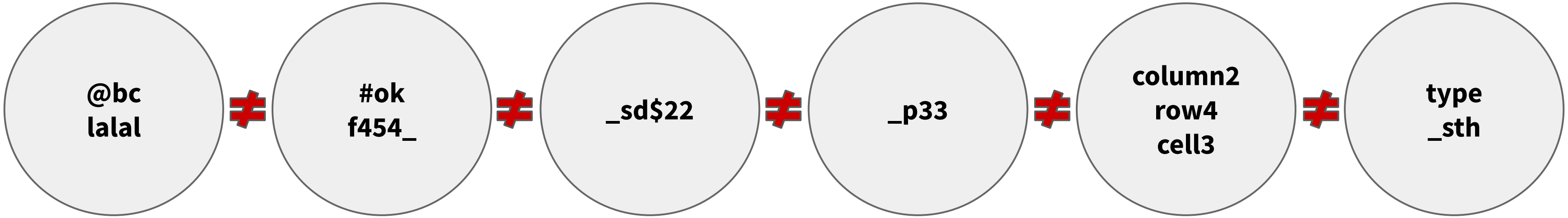
Case 1: Many labels assigned to each instance

Case 2: Different labels referring to the semantically same entity

Case 3: Missing labels to some instances

Case 4: No labels to all instances and inconsistent properties

These are usually the types you are going to find in ***real noisy*** datasets



We cannot assume that these are the same type

Ambiguities in Schema Discovery in PGs

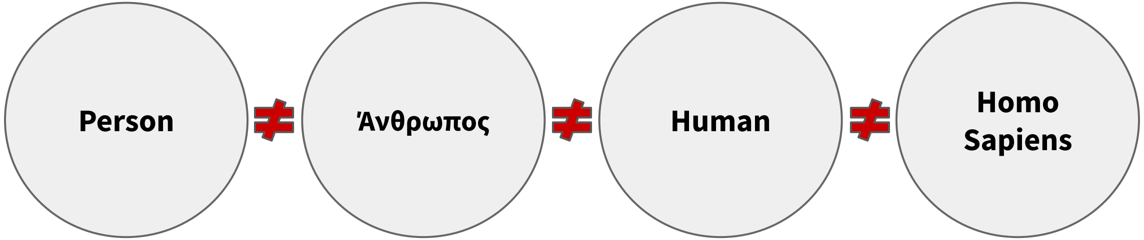
In real datasets we might have several difficult cases to identify types:

Case 1: Many labels assigned to each instance

Case 2: Different labels referring to the semantically same entity

Case 3: Missing labels to some instances

Case 4: No labels to all instances and inconsistent properties



We cannot assume that these are the same type

Ambiguities in Schema Discovery in PGs

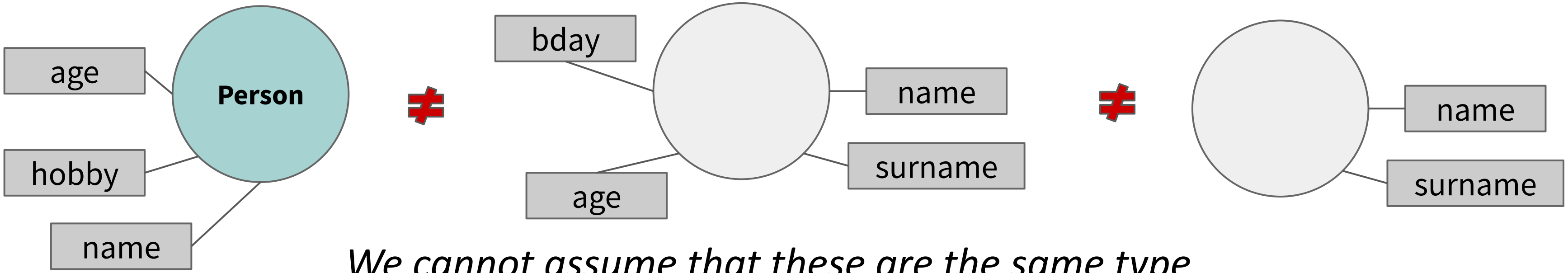
In real datasets we might have several difficult cases to identify types:

Case 1: Many labels assigned to each instance

Case 2: Different labels referring to the semantically same entity

Case 3: Missing labels to some instances

Case 4: No labels to all instances and inconsistent properties



We cannot assume that these are the same type

Ambiguities in Schema Discovery in PGs

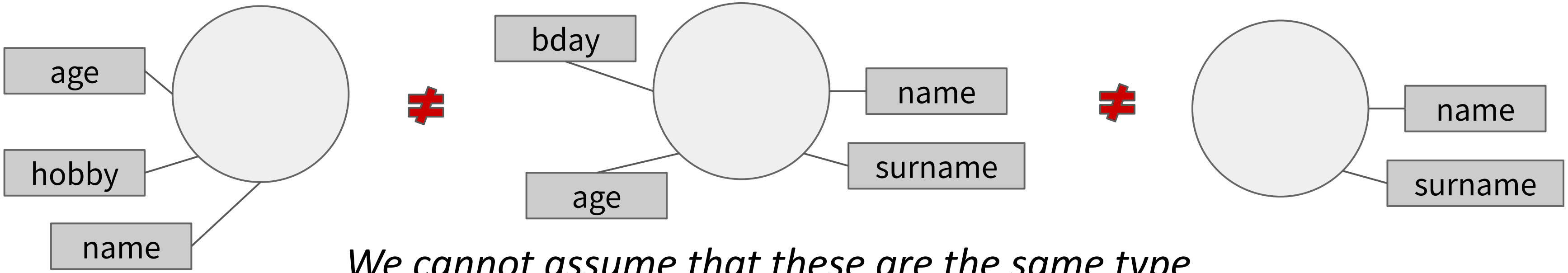
In real datasets we might have several difficult cases to identify types:

Case 1: Many labels assigned to each instance

Case 2: Different labels referring to the semantically same entity

Case 3: Missing labels to some instances

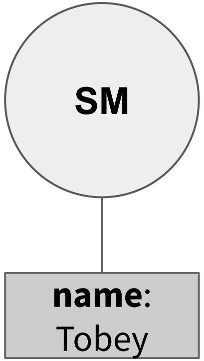
Case 4: No labels to all instances and inconsistent properties



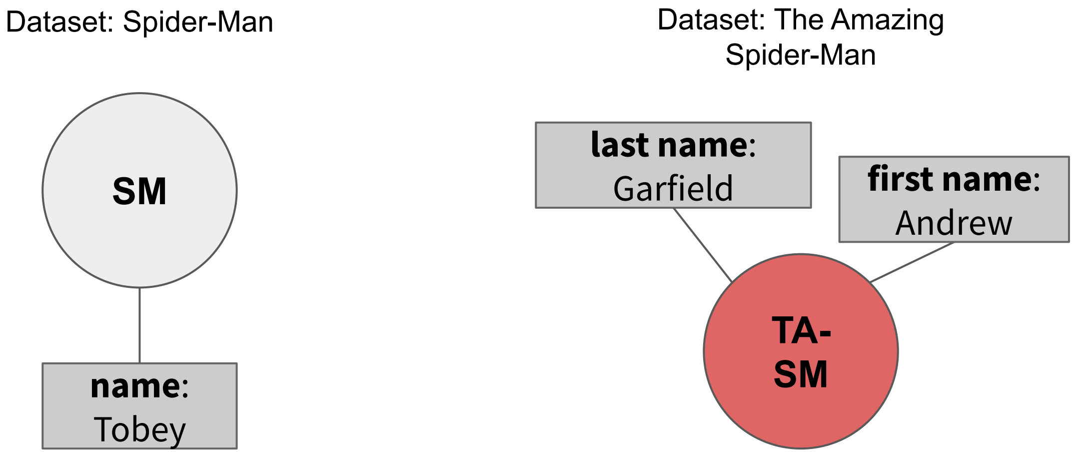
We cannot assume that these are the same type

Ambiguities in Integrated Datasets

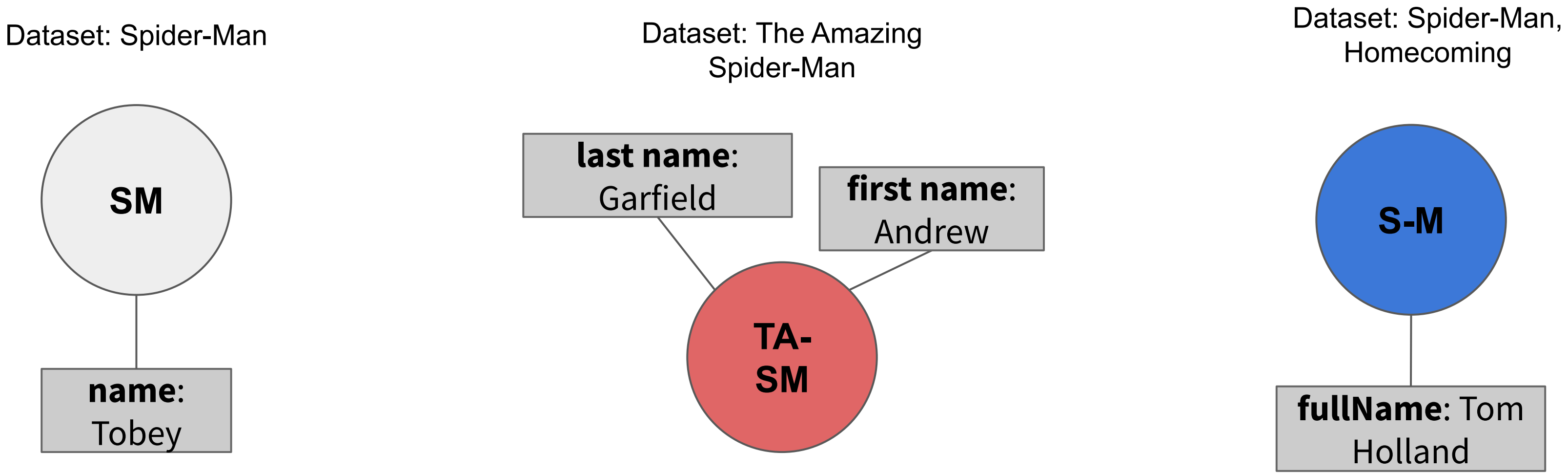
Dataset: Spider-Man



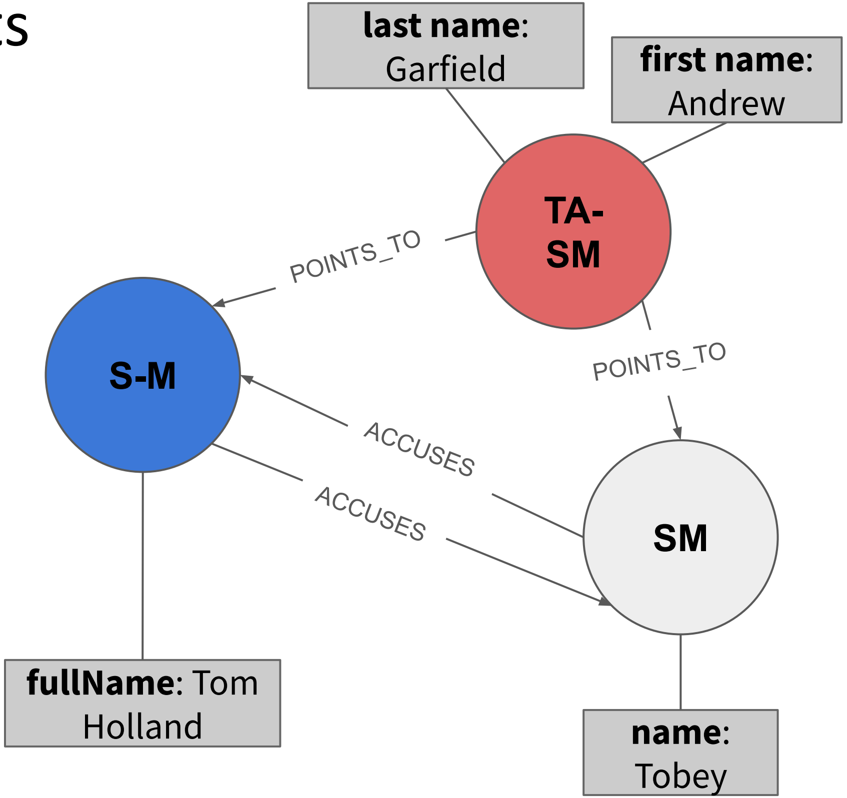
Ambiguities in Integrated Datasets



Ambiguities in Integrated Datasets



Ambiguities in Integrated Datasets



Is anyone the imposter? 👀

Ambiguities in Integrated Datasets

Spider-Man: No Way Home (2021)

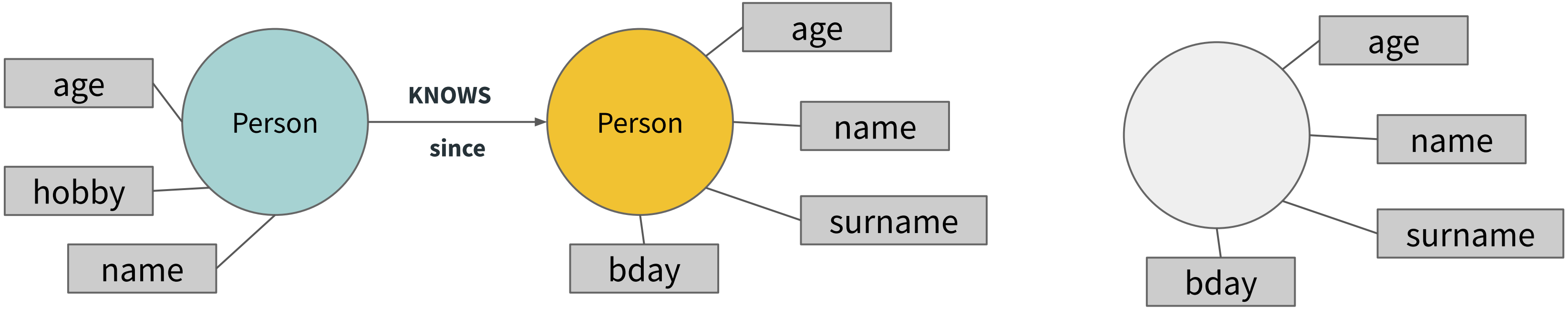
No one was an imposter, but how could we know? 🤔

The data **had different type labels, and different properties.**

What is a *type*?

Patterns & Types

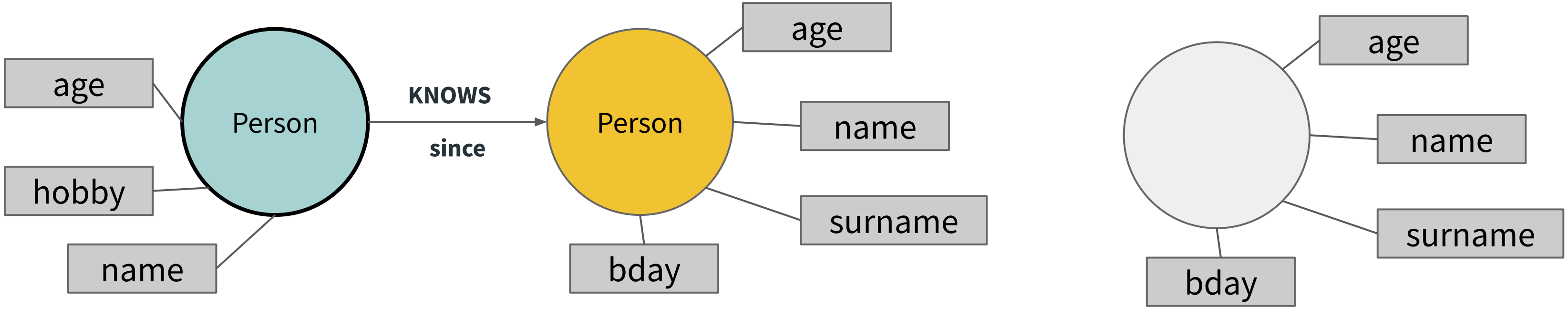
We identify patterns in the data in order to extract our types.



Pattern 1: {Person}, {age, hobby, name}
Pattern 2: {Person}, {age, name, surname, bday}
Pattern 3: {∅}, {age, name, surname, bday}
Pattern 4: {KNOWS}, {Person}, {Person}, {since}

Patterns & Types

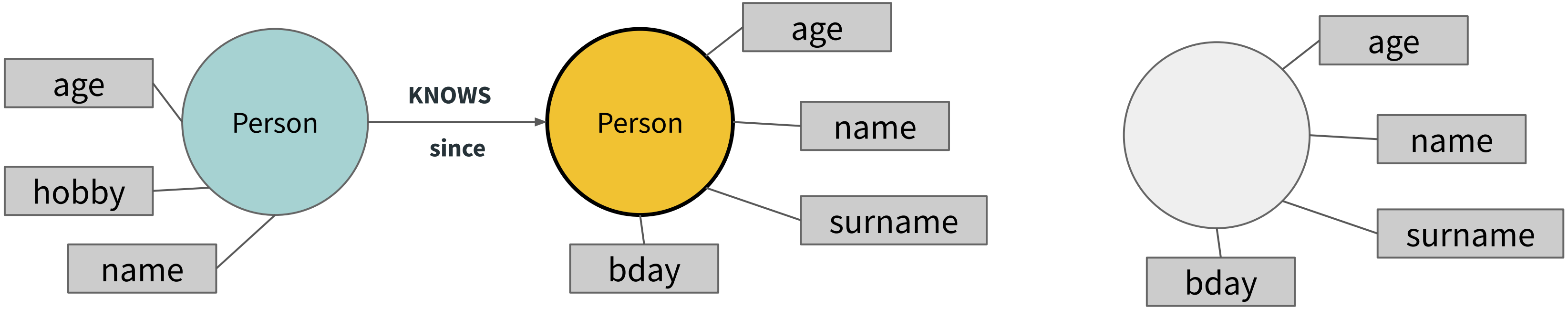
We identify patterns in the data in order to extract our types.



Pattern 1: {Person}, {age, hobby, name}
Pattern 2: {Person}, {age, name, surname, bday}
Pattern 3: {∅}, {age, name, surname, bday}
Pattern 4: {KNOWS}, {Person}, {Person}, {since}

Patterns & Types

We identify patterns in the data in order to extract our types.



Pattern 1: {Person}, {age, hobby, name}

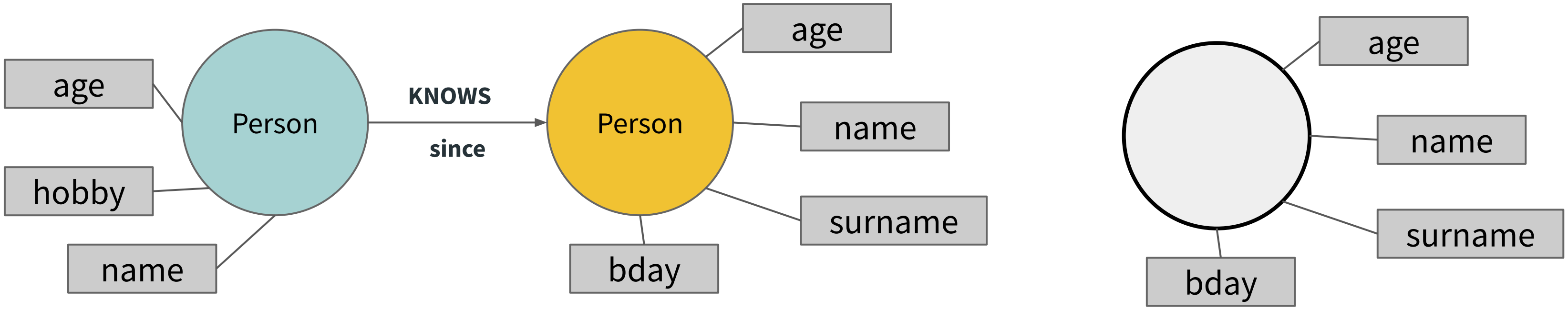
Pattern 2: {Person}, {age, name, surname, bday}

Pattern 3: {∅}, {age, name, surname, bday}

Pattern 4: {KNOWS}, {Person}, {Person}, {since}

Patterns & Types

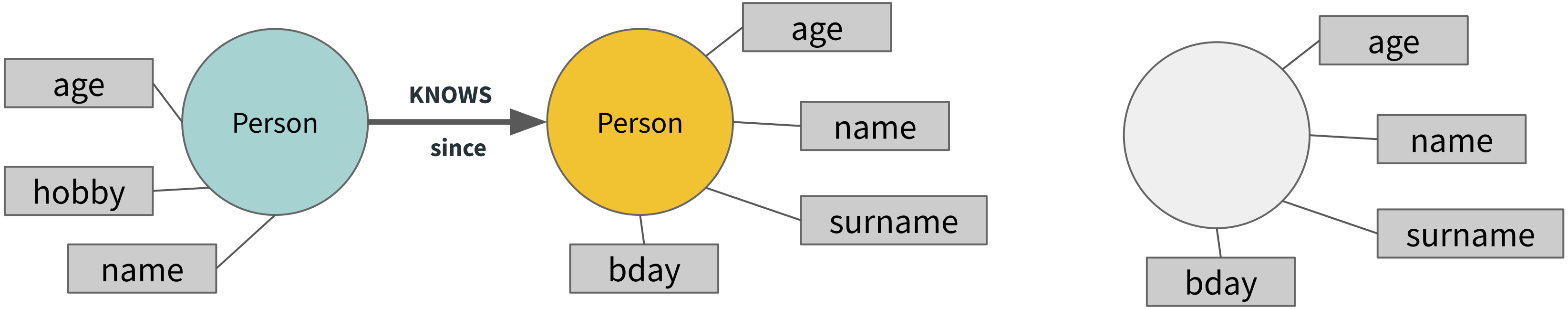
We identify patterns in the data in order to extract our types.



Pattern 1: {Person}, {age, hobby, name}
Pattern 2: {Person}, {age, name, surname, bday}
Pattern 3: {∅}, {age, name, surname, bday}
Pattern 4: {KNOWS}, {Person}, {Person}, {since}

Patterns & Types

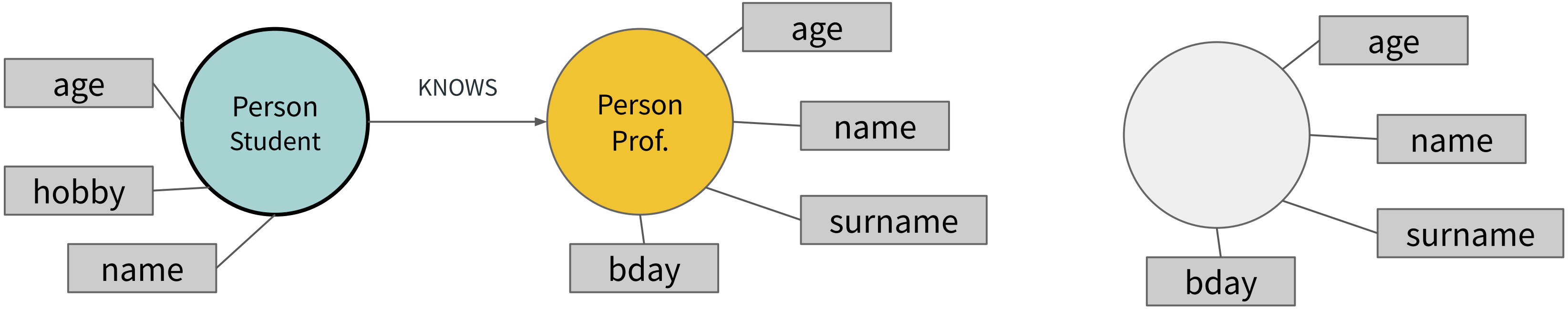
We identify patterns in the data in order to extract our types.



Pattern 1: {Person}, {age, hobby, name}
Pattern 2: {Person}, {age, name, surname, bday}
Pattern 3: {∅}, {age, name, surname, bday}
Pattern 4: {KNOWS}, {Person}, {Person}, {since}

Patterns & Types

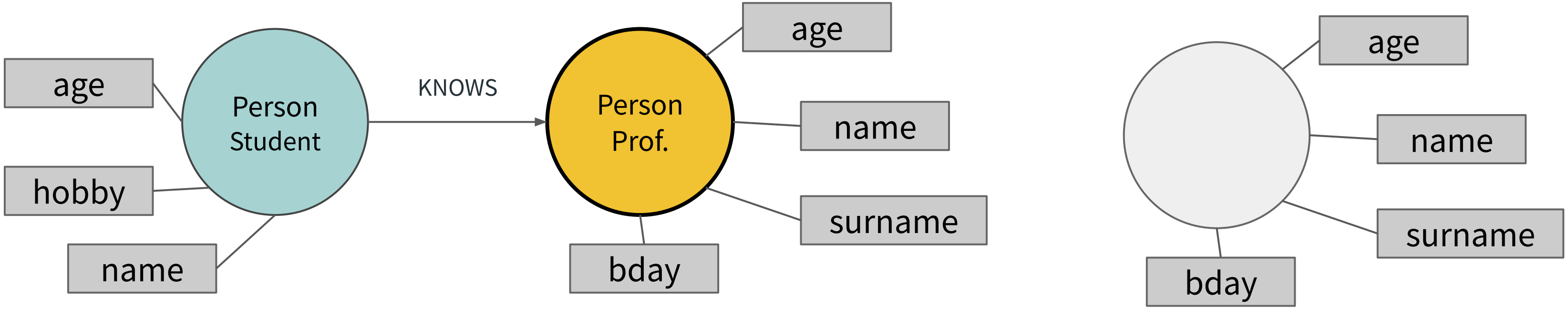
We identify patterns in the data in order to extract our types.



Pattern 5: {Person,Student}, {age, hobby, name}
Pattern 6: {Person,Prof.}, {age, name, surname, bday}
Pattern 7: {KNOWS}, {Person}, {Person}, {∅}

Patterns & Types

We identify patterns in the data in order to extract our types.



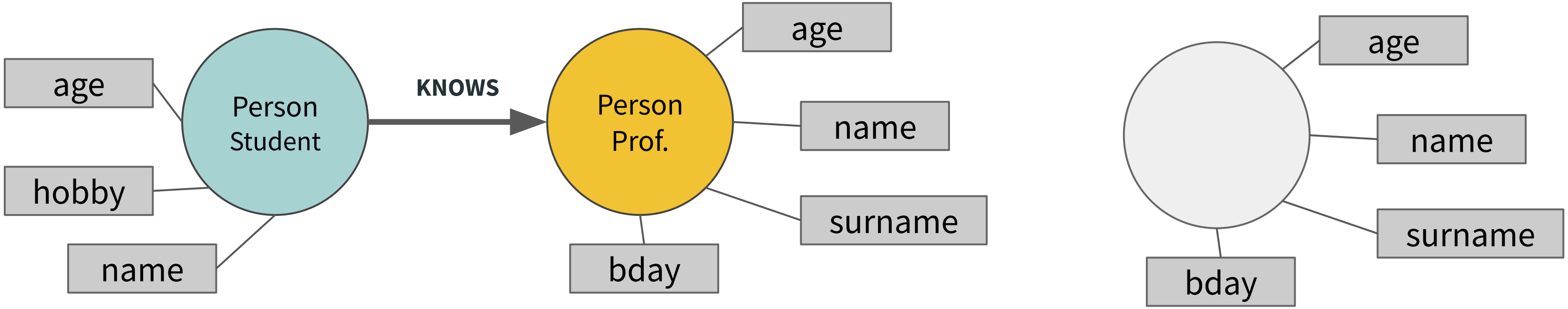
Pattern 5: {Person,Student}, {age, hobby, name}

Pattern 6: {Person,Prof.}, {age, name, surname, bday}

Pattern 7: {KNOWS}, {Person}, {Person}, {∅}

Patterns & Types

We identify patterns in the data in order to extract our types.



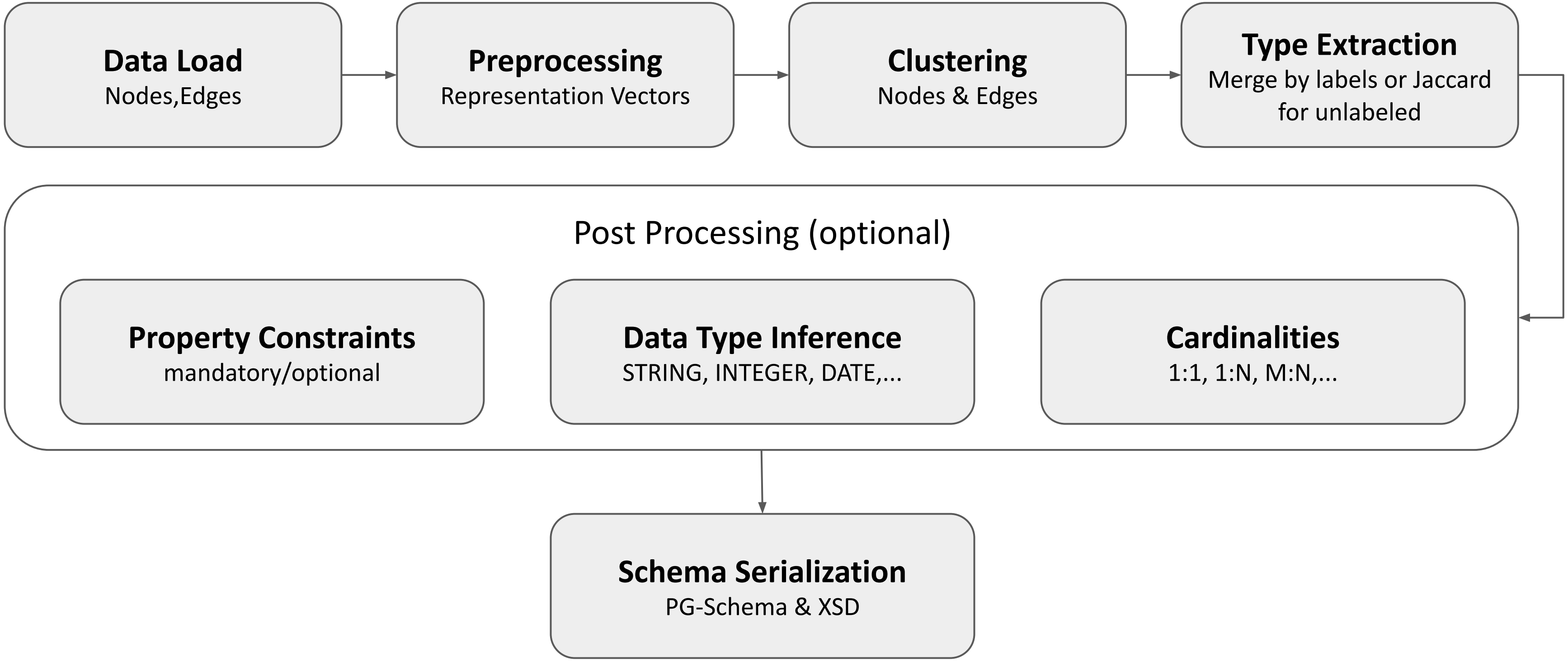
Pattern 5: {Person,Student}, {age, hobby, name}
Pattern 6: {Person,Prof.}, {age, name, surname, bday}
Pattern 7: {KNOWS}, {Person}, {Person}, {∅}

PG-HIVE

PG-HIVE: Hybrid Incremental Schema Discovery for PGs

HYBRID	**INCREMENTAL**
Labels	**Batch processing**
Properties	**Merging schemas**
Graph characteristics	

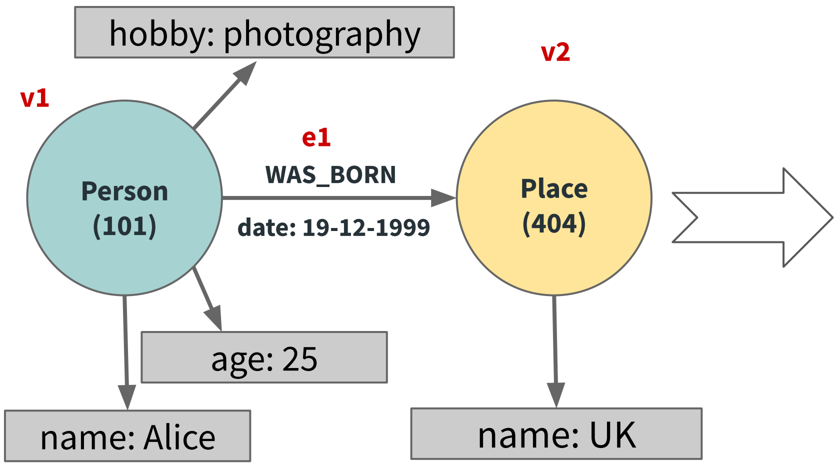
PG-HIVE: Hybrid Incremental Schema Discovery for PGs



PG-HIVE: Data Loading

We load data from PG storage (Neo4j) and transform them into vectors:

vi={{Word2Vec_label},{one-hot encoded properties}}
ei={{Word2Vec_src},{Word2Vec_tgt},{Word2Vec_label},{one-hot encoded properties}}

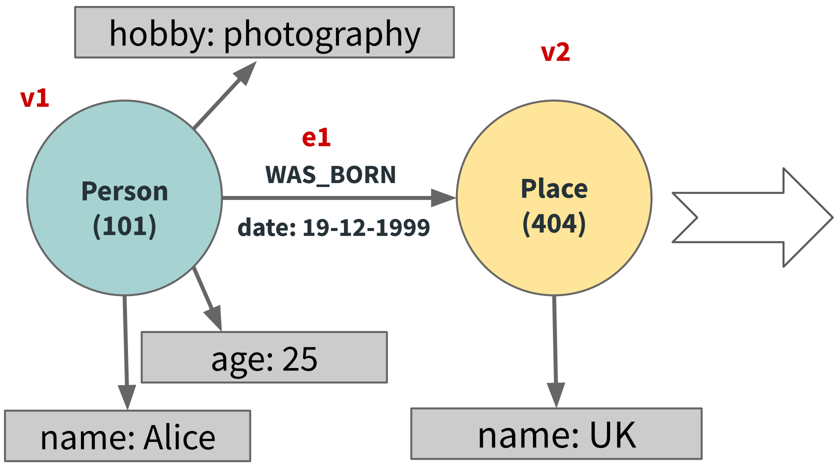


v1={w2v{Person},{0-1{name,age,hobby}}}
v2={w2v{Place},{0-1{name,age,hobby}}}
e1={w2v{Person},w2v{Place},w2v{WAS_BORN},
{0-1{name}}}

PG-HIVE: Data Loading

We load data from PG storage (Neo4j) and transform them into vectors:

vi={{Word2Vec_label},{one-hot encoded properties}}
ei={{Word2Vec_src},{Word2Vec_tgt},{Word2Vec_label},{one-hot encoded properties}}

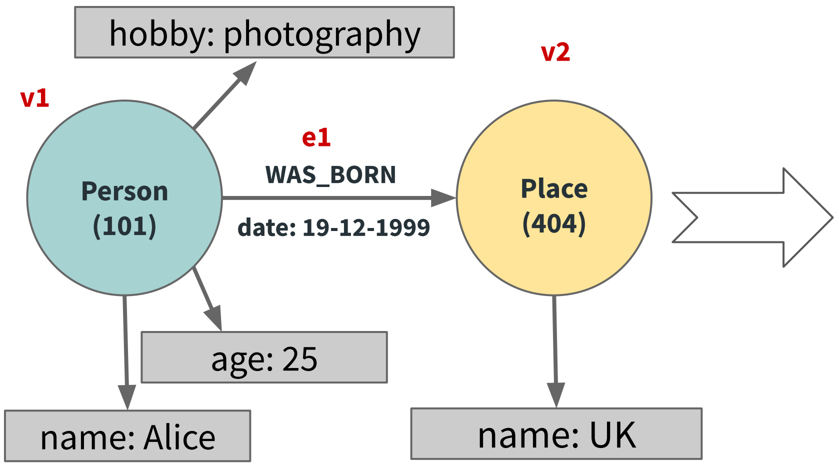


v1={w2v{Person},{0-1{name,age,hobby}}}
v2={w2v{Place},{0-1{name,age,hobby}}}
e1={w2v{Person},w2v{Place},w2v{WAS_BORN},
{0-1{name}}}

PG-HIVE: Data Loading

We load data from PG storage (Neo4j) and transform them into vectors:

vi={{Word2Vec_label},{one-hot encoded properties}}
ei={{Word2Vec_src},{Word2Vec_tgt},{Word2Vec_label},{one-hot encoded properties}}



v1={w2v{2,1,4},{0-1{1,1,1}}}
v2={w2v{1,6,3},{0-1{1,0,0}}}
e1={w2v{2,1,4},w2v{1,6,3},w2v{1,1,1},{0-1{1}}}

PG-HIVE: Clustering

We use two clustering algorithms:

ELSH

Euclidean distance

Parameters: ***T, b***

T: number of hash tables

b: bucket length

Performs better in *feature vectors*

MinHash LSH

Jaccard Similarity

Parameter: ***T***

T: number of hash tables

Performs better in *set-like* data

PG-HIVE: Clustering

We use two clustering algorithms:

ELSH

Euclidean distance

Parameters: ***T, b***

T: number of hash tables

b: bucket length

Performs better in *feature vectors*

MinHash LSH

Jaccard Similarity

Parameter: ***T***

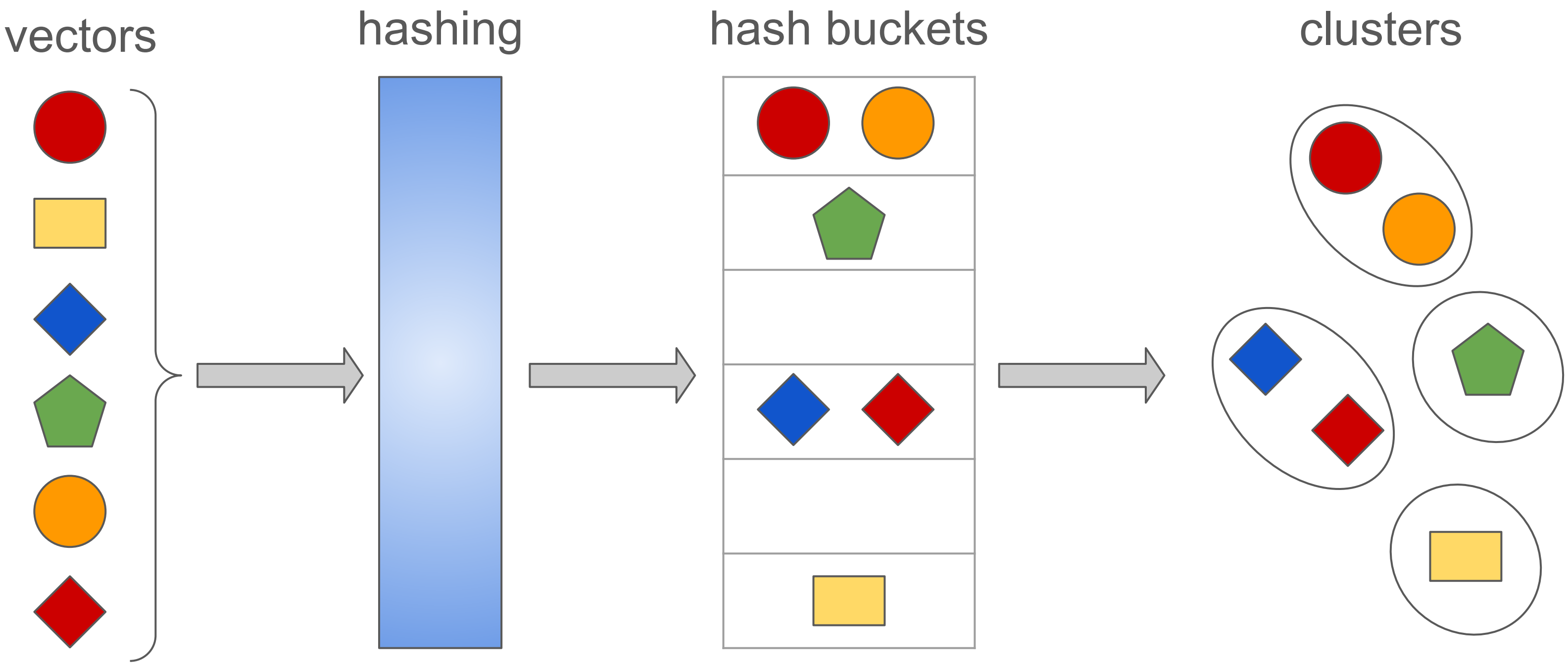
T: number of hash tables

Performs better in *set-like* data

How to choose the best parameters? 🤔

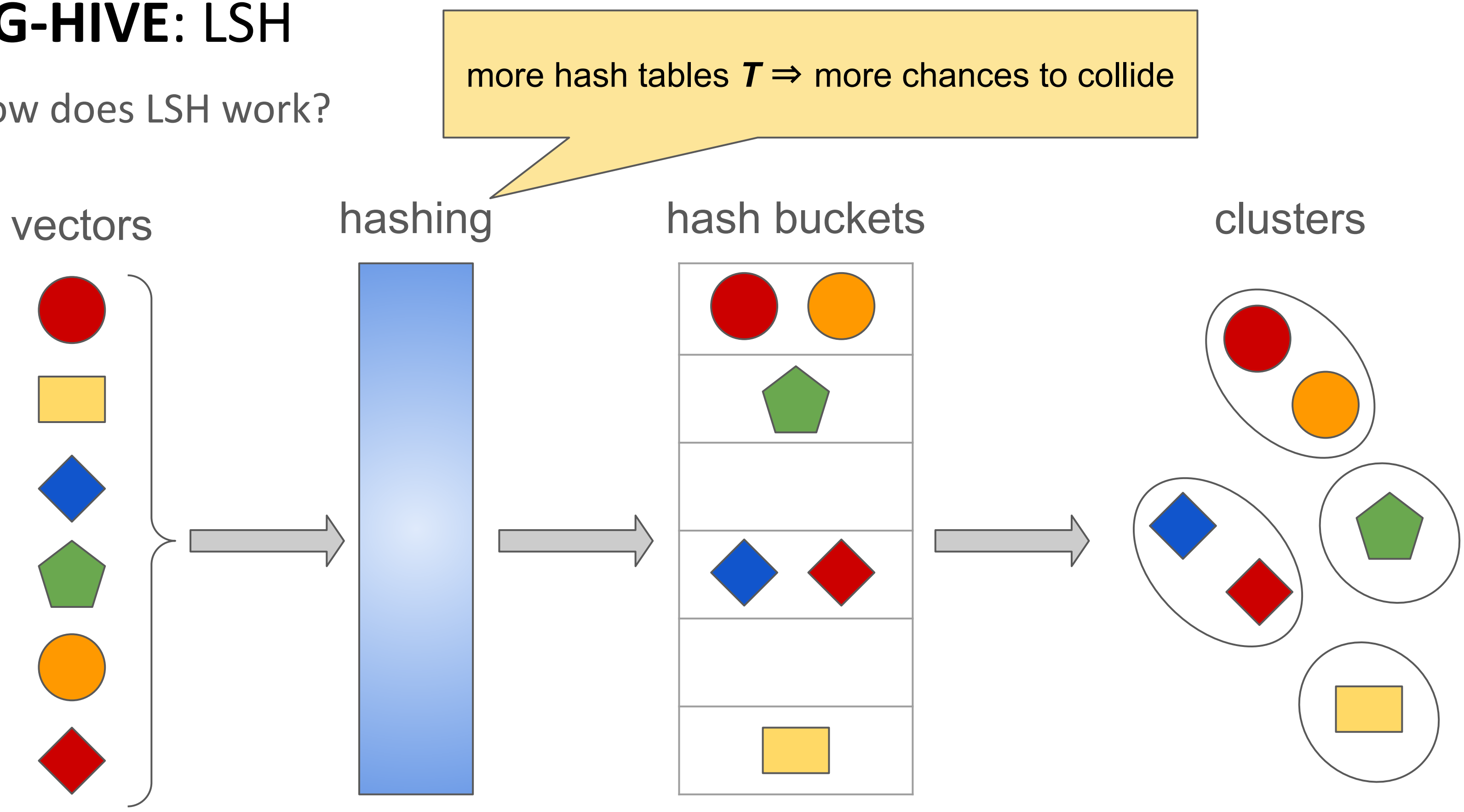
PG-HIVE: LSH

How does LSH work?



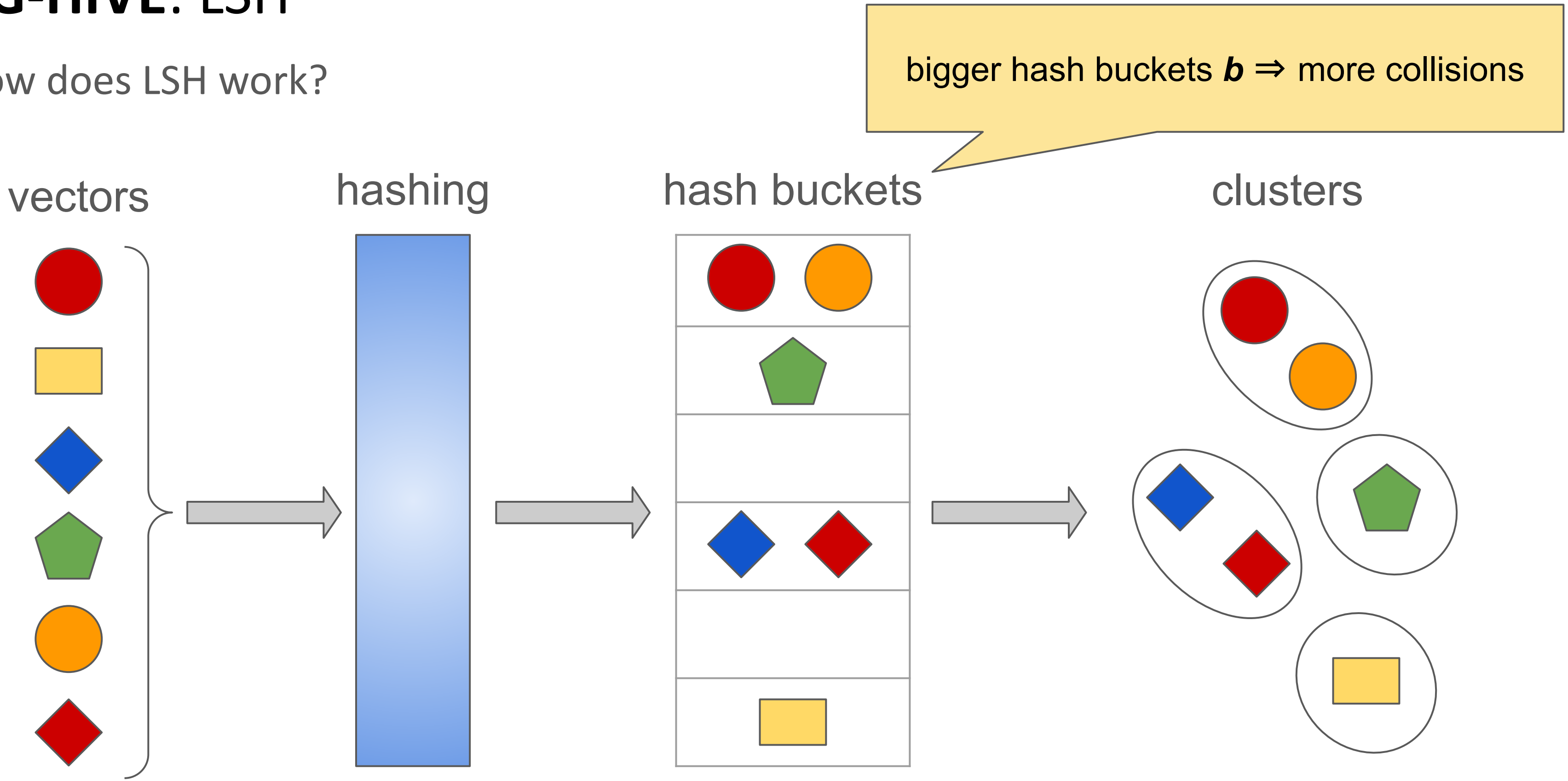
PG-HIVE: LSH

How does LSH work?



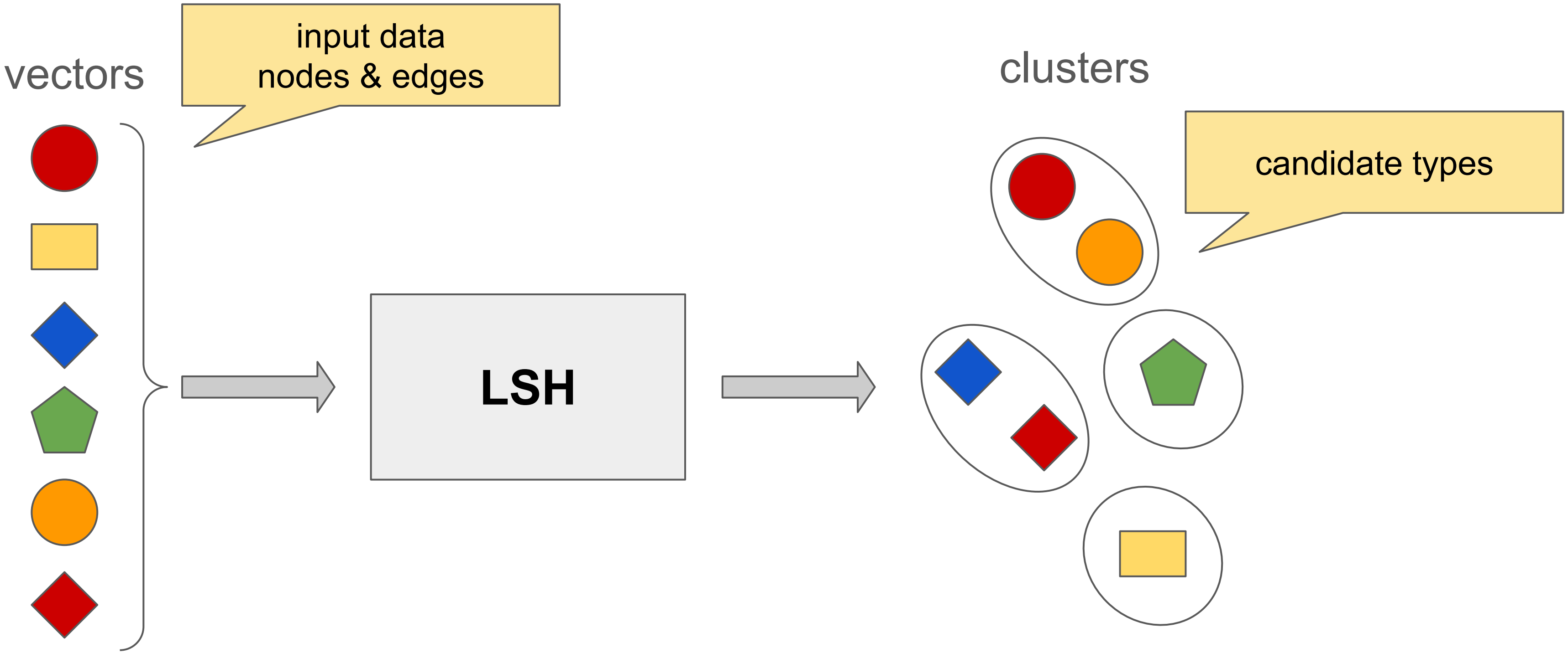
PG-HIVE: LSH

How does LSH work?



PG-HIVE: LSH

In our case



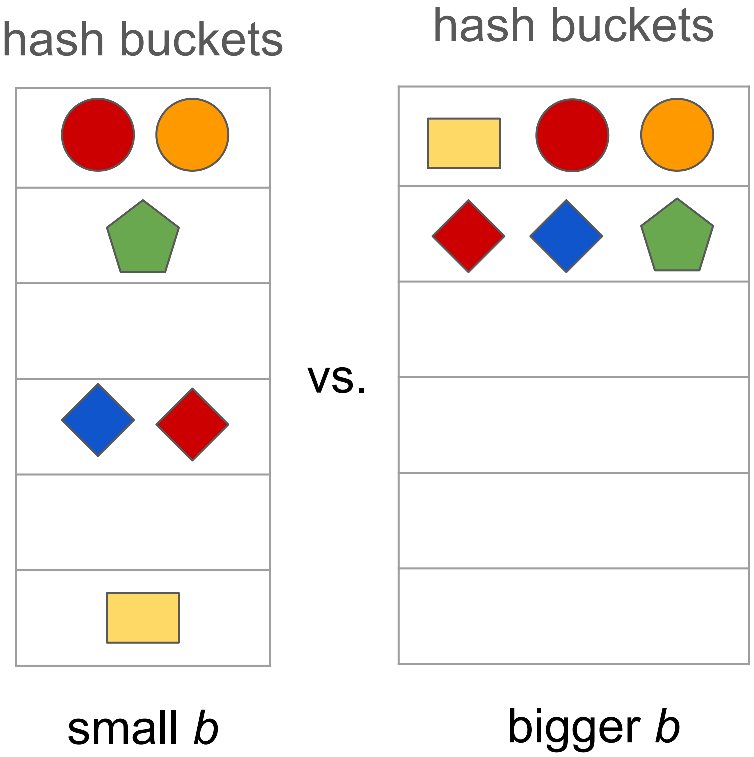
PG-HIVE: Adaptive parameterization

To avoid manual tuning of LSH, we introduce an adaptive selection of b and T.

For sparse graph we increase the bucket length and hash tables

For less sparse graphs we decrease them

increasing $T \Rightarrow$ more chances to collide
increasing $b \Rightarrow$ more collisions, higher recall but lower precision



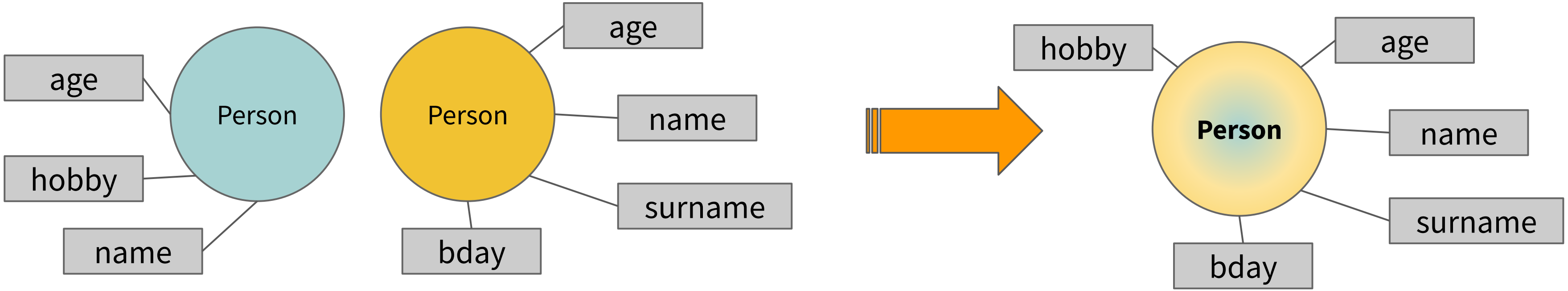
PG-HIVE: Extracting Types

After the clustering step, we refine the candidate node/edge types by merging the ones that correspond to the same schema type

Step 1: Same label

Step 2: Unlabeled clusters with Labeled ones (*iff* Jaccard > 0.9)

Step 3: Unlabeled clusters with the remaining unlabeled (*iff* Jaccard > 0.9)



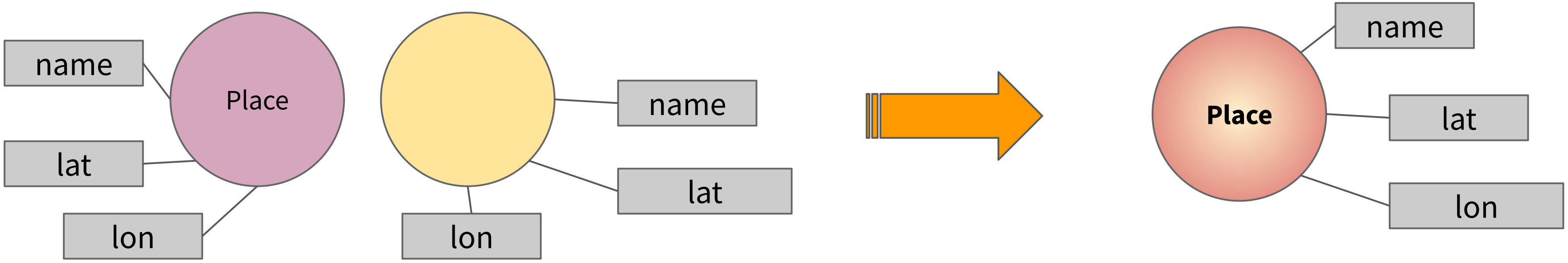
PG-HIVE: Extracting Types

After the clustering step, we refine the candidate node/edge types by merging the ones that correspond to the same schema type

Step 1: Same label

Step 2: Unlabeled clusters with Labeled ones (*iff* Jaccard > 0.9)

Step 3: Unlabeled clusters with the remaining unlabeled (*iff* Jaccard > 0.9)



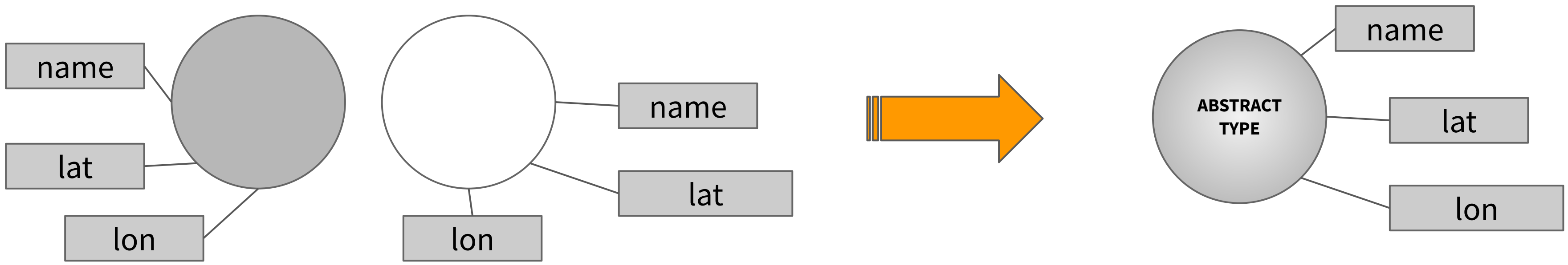
PG-HIVE: Extracting Types

After the clustering step, we refine the candidate node/edge types by merging the ones that correspond to the same schema type

Step 1: Same label

Step 2: Unlabeled clusters with Labeled ones (*iff* Jaccard > 0.9)

Step 3: Unlabeled clusters with the remaining unlabeled (*iff* Jaccard > 0.9)



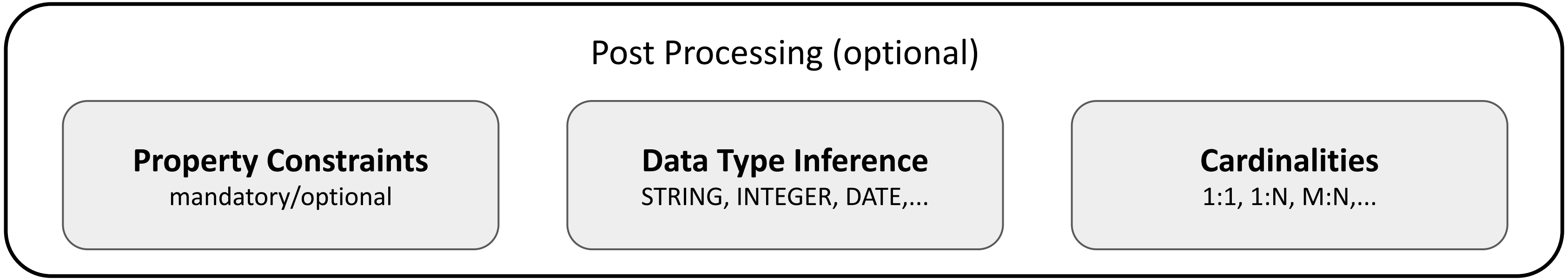
PG-HIVE: Post Processing

The post processing step is optional infers:

(i) property constraints

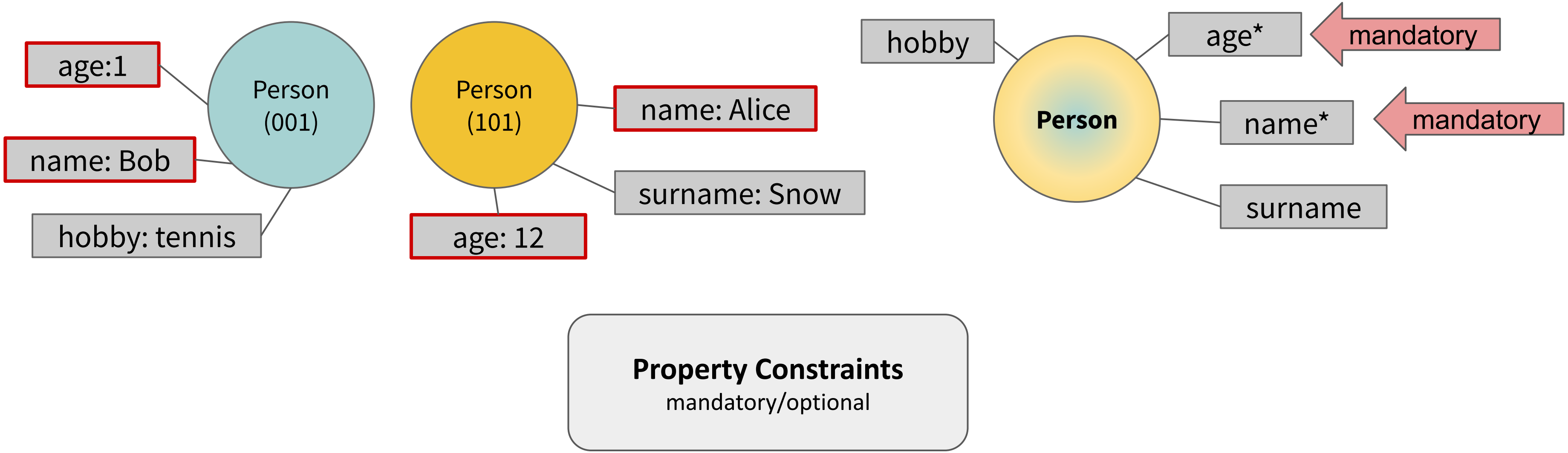
(ii) data type inference

(iii) cardinalities



PG-HIVE: Post Processing - Property Constraints

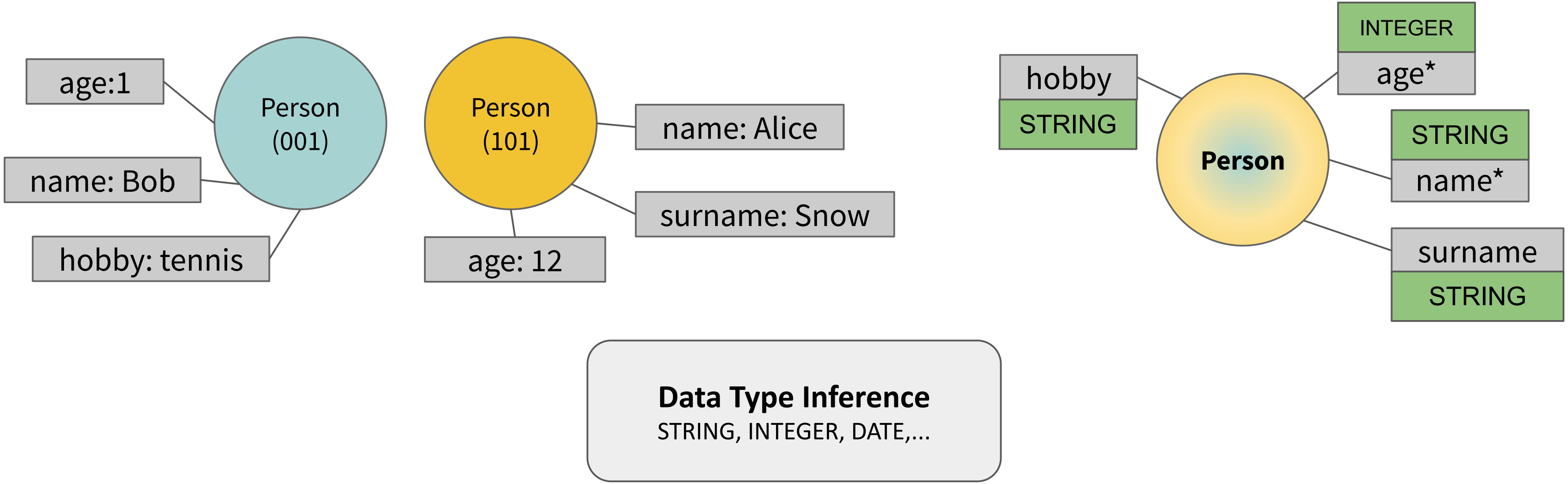
A property is characterized as **mandatory** for a given type if it appears in every instance of that type, otherwise, it is considered **optional**.



PG-HIVE: Post Processing - Data Types

We use heuristics to find the data type of each property:

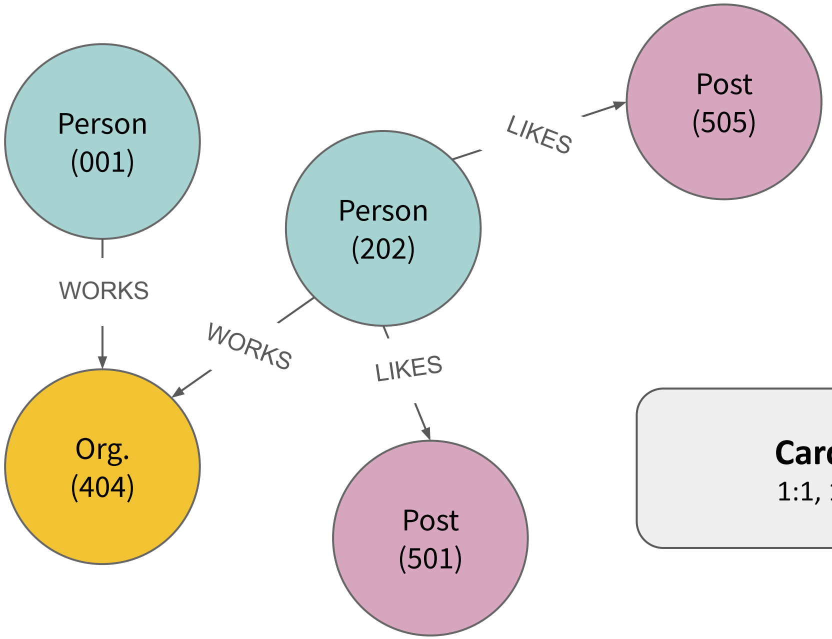
INTEGER ⟶ FLOAT ⟶ BOOLEAN ⟶ DATE ⟶ STRING (fallback)



**using sampling*

PG-HIVE: Post Processing - Cardinalities

To find the cardinalities of edge types, we query how many distinct targets each source connects to, for each edge type, and vice versa.

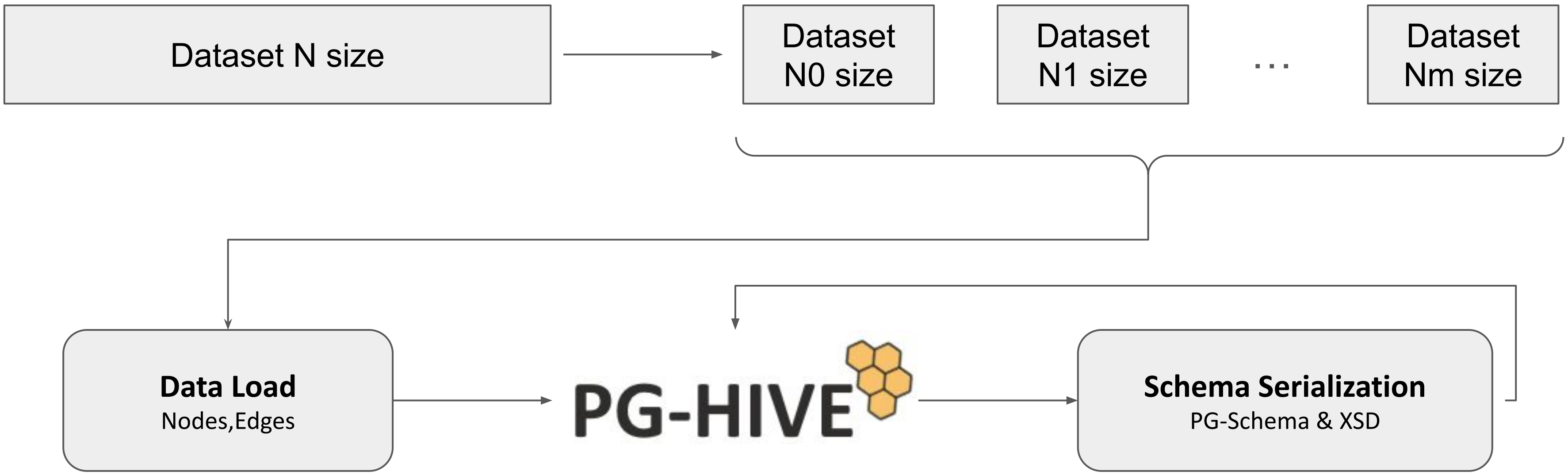


Cardinalities
1:1, 1:N, M:N,...

Person:Org ⇒N,1
Person:Post ⇒ 0,N

PG-HIVE: Incremental Module

To process big datasets in different settings we have also an incremental module



PG-HIVE: Serialization - XSD

```xml
<xs:schema name="NewGraphSchema" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="Place">
    <xs:sequence>
      <xs:element name="type" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="name" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="id" type="xs:double" minOccurs="1" maxOccurs="1"/>
      <xs:element name="url" type="xs:string" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID" use="required"/>
    <xs:attribute name="label" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="Person">
    <xs:sequence>
      <xs:element name="lastName" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="firstName" type="xs:string" minOccurs="1" maxOccurs="1"/>
      <xs:element name="birthday" type="xs:date" minOccurs="1" maxOccurs="1"/>
    </xs:sequence>
    <xs:attribute name="id" type="xs:ID" use="required"/>
    <xs:attribute name="label" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="IS_LOCATED_IN">
    <xs:sequence>
      <xs:element name="source" type="Person"/>
      <xs:element name="target" type="Place"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

PG-HIVE: Serialization - PG-Schema

```
CREATE GRAPH TYPE NewGraphSchema LOOSE {
  (PlaceType: Place {type STRING, name STRING, id DOUBLE, url STRING}),
  (PersonType: Person {lastName STRING, firstName STRING, birthday DATE}),
  (EmailType: Email {address STRING}),
  (:PersonType)-[HAS_EMAILType: HAS_EMAIL]->(:EmailType),
  (:PersonType)-[IS_LOCATED_INType: IS_LOCATED_IN]->(:PlaceType),
}
```

LOOSE

```
CREATE NODE TYPE PlaceType : Place {type STRING, name STRING, id DOUBLE, url STRING};
CREATE NODE TYPE PersonType : Person {lastName STRING, firstName STRING, birthday DATE};
CREATE NODE TYPE EmailType : Email {address STRING};

CREATE EDGE TYPE has_emailType : HAS_EMAIL;
CREATE EDGE TYPE is_located_inType : IS_LOCATED_IN;

CREATE GRAPH TYPE NewGraphSchema STRICT {
  (PlaceType),
  (PersonType),
  (EmailType),
  (:PersonType)-[has_emailType]->(:EmailType),
  (:PersonType)-[is_located_inType]->(:PlaceType),

  FOR (x:EmailType) SINGLETON x WITHIN (:PersonType)-[y: has_emailType]->(x)
  FOR (x:PersonType) SINGLETON y WITHIN (x)-[y: is_located_inType]->(:PlaceType)
}
```

STRICT

Part 3:

Evaluation

Previous Work on Schema Discovery for PGs

	SchemI[1]	GMMSchema[2]	DiscoPG[3]	PG-HIVE (ours)
Label Independent	❌	❌	❌	✅
Multi Labeled Elements	❌	✅	✅	✅
Schema Elements	Nodes & Edges	Nodes only	Nodes & queries associated Edges	Nodes & Edges & constraints
Constraints	❌	❌	❌	✅
Incremental	❌	❌	✅	✅
Automation	✅	✅	✅ + UI	✅
Notes	Cannot handle missing labels	GMM clustering, can not handle missing labels	Demo of GMMSchema	LSH + fine tuning

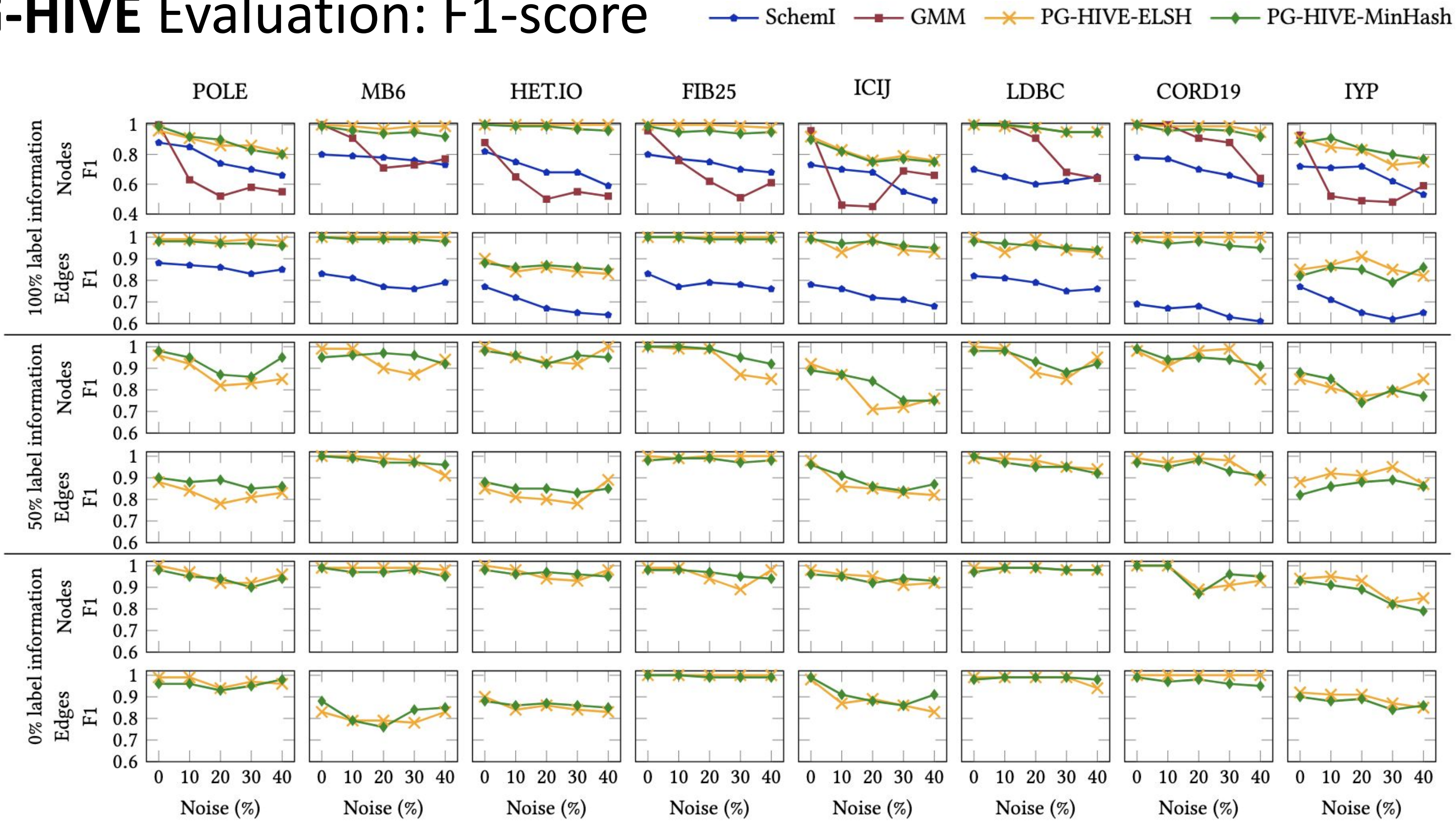
[1]SchemI:Lbath, H., Bonifati, A., Harmer, R.: Schema inference for property graphs.EDBT 2021
[2]GMMSchema: Bonifati, A., Dumbrava, S., Mir, N.: Hierarchical clustering for property graph schema discovery. EDBT 2022
[3]DiscoPG:Bonifati, A., Dumbrava, S., Martinez, E., Ghasemi, F., Jaffré, M., Luton, P., Pickles, T.: Discopg: Property graph schema discovery and exploration. Proc. VLDB 2022

PG-HIVE Evaluation: Datasets

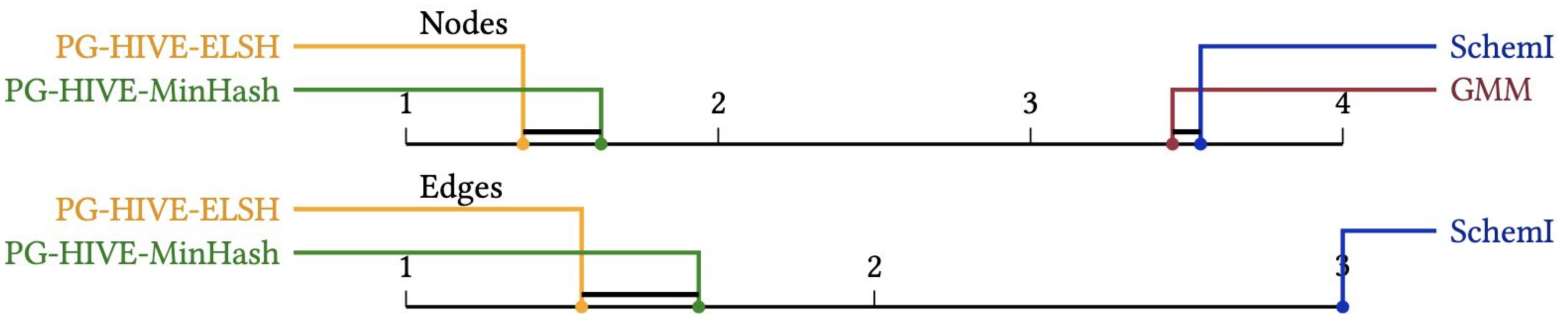
Dataset	Nodes	Edges	Node Types	Edge Types	Node Labels	Edge Labels	Node Pat.	Edge Pat.	Real Synth.
POLE	61,521	105,840	11	17	11	16	11	16	S
MB6	486,267	961,571	4	5	10	3	52	9	S
HET.IO	47,031	2,250,197	11	24	12	24	14	38	R
FIB25	802,473	1,625,428	4	5	10	3	31	9	S
ICIJ	2,016,523	3,339,267	5	14	6	14	2,263	88	R
LDBC	3,181,724	17,256,038	7	17	8	15	9	23	S
CORD19	36,025,729	59,768,373	6	5	6	5	89	6	R
IYP	44,539,999	251,432,812	32	24	33	24	25,137	790	R

Environment: Spark cluster (4 nodes × 38 cores), Scala 2.12.10, Neo4j 4.4.0

PG-HIVE Evaluation: F1-score

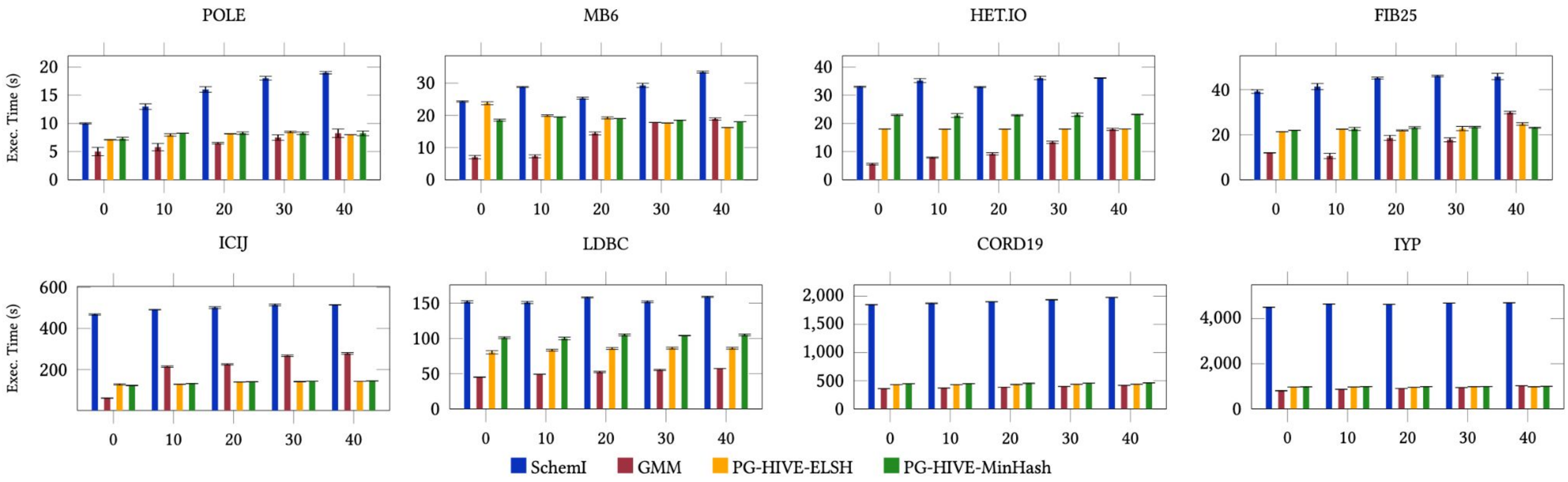


PG-HIVE Evaluation: Statistical Significance



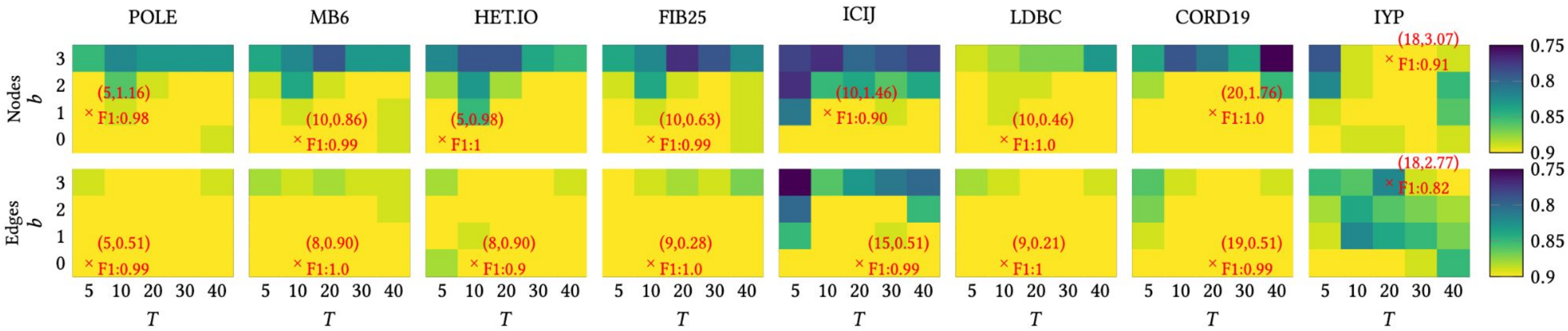
Statistical significance analysis of F1-scores across datasets for **nodes (top)** and **edges (bottom)** --GMM does not produce edge types.

PG-HIVE Evaluation: Execution Time



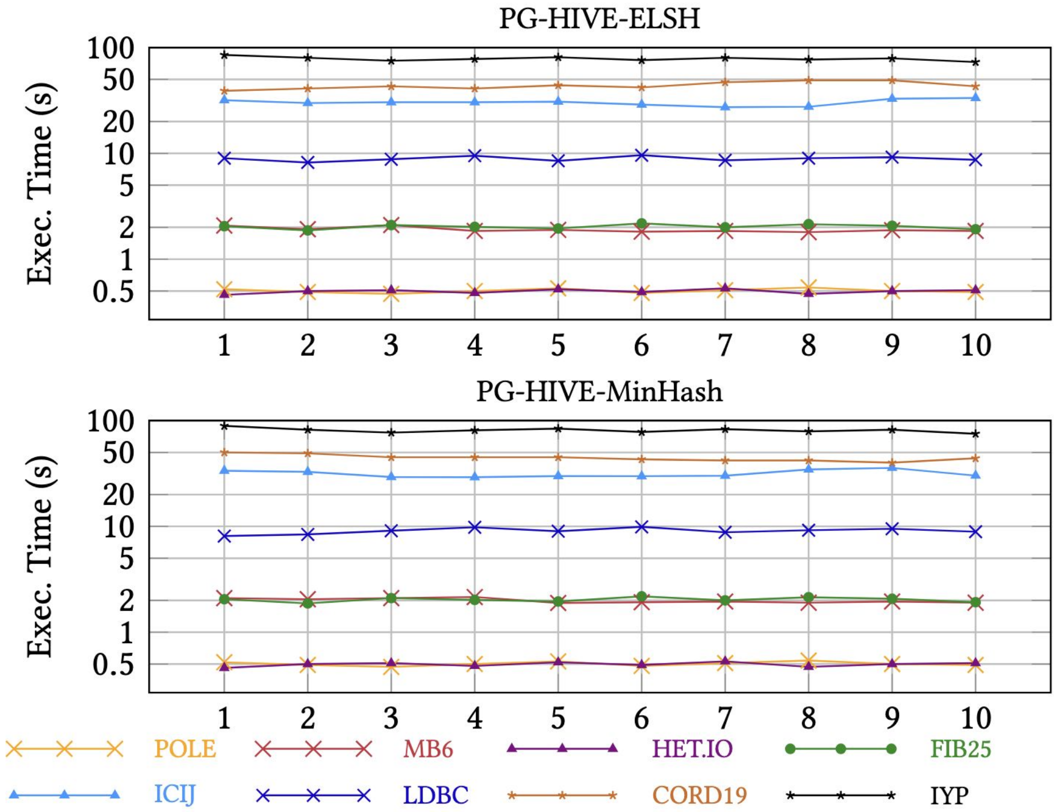
Execution time until type discovery on each dataset across different noise percentages (0% - 40%)

PG-HIVE Evaluation: Adaptive Parameterization



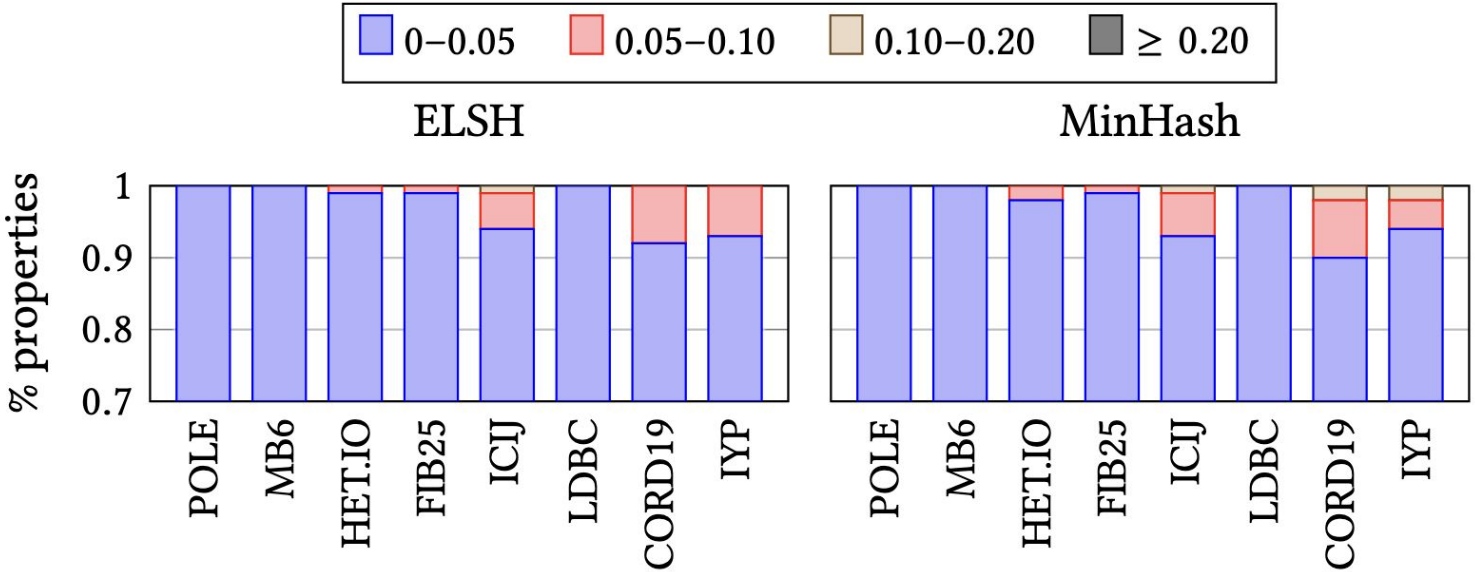
Heatmaps of F1-scores across datasets (100% label availability & 0% noise) for nodes (top) and edges (bottom) with varying T and b; red × (T , b) denotes the adaptive choice for ELSH.

PG-HIVE Evaluation: Incremental Execution



Incremental execution time per iteration. We separate each dataset in 10 equal parts.

PG-HIVE Evaluation: Data Type Inference Sampling Error



Distribution of data type inference errors using sampling, across datasets for ELSH (left) and MinHash (right).

Future Work

Schema Discovery:

- Handle unlabeled and highly sparse graphs
- Use LLMs to align and normalize labels across datasets/languages
- Support provenance, versioning, and schema evolution
- Use association rule mining to discover subtypes

Thank you

PG-HIVE github repo