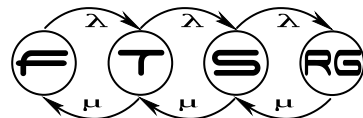


Writing a Cypher Engine in Clojure

Dávid Szakállas (BME)

Gábor Szárnyas (BME, MTA)



Magyar Tudományos
Akadémia



McGill

AGENDA

Gábor:

- Graph databases
- Cypher language
- Query evaluation

Dávid:

- Graph exploration strategies
- Optimization
- How to compile patterns

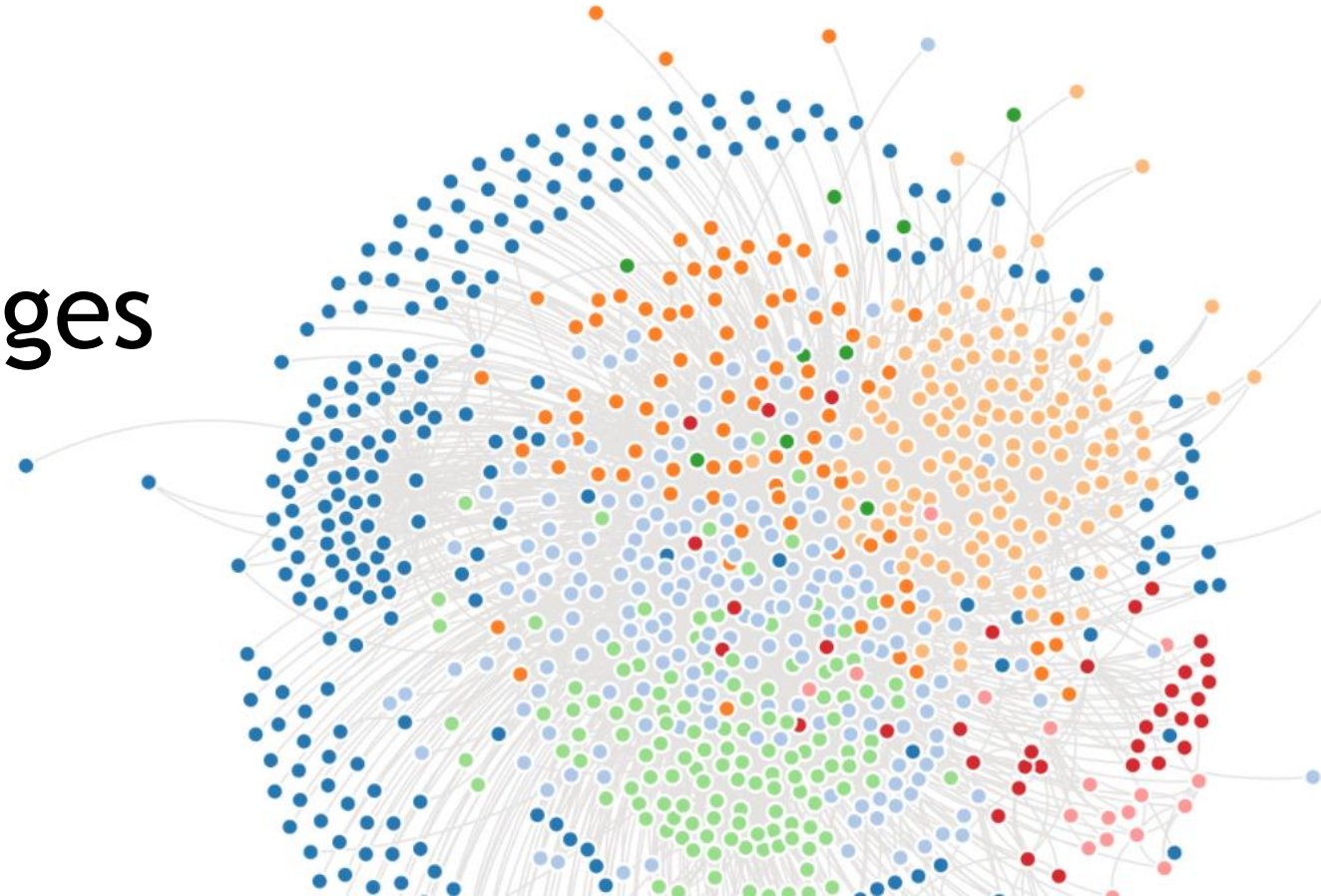
GRAPHS

Textbook definition

$$G = (V, E) \quad E \subseteq V \times V$$

Flavours:

- directed/undirected edges
- labels
- **properties**








GRAPH DATABASES


NoSQL family

Data model: *property graphs* = vertices, edges, and properties

pattern matching



analytics



GraphX



APACHE GIRAPH



GraphLab

*(6)

Person(2)

Project(1)

Institution(2)

Presentation(1)

*(9)

ADVISOR_OF(1)

DEVELOPS(1)

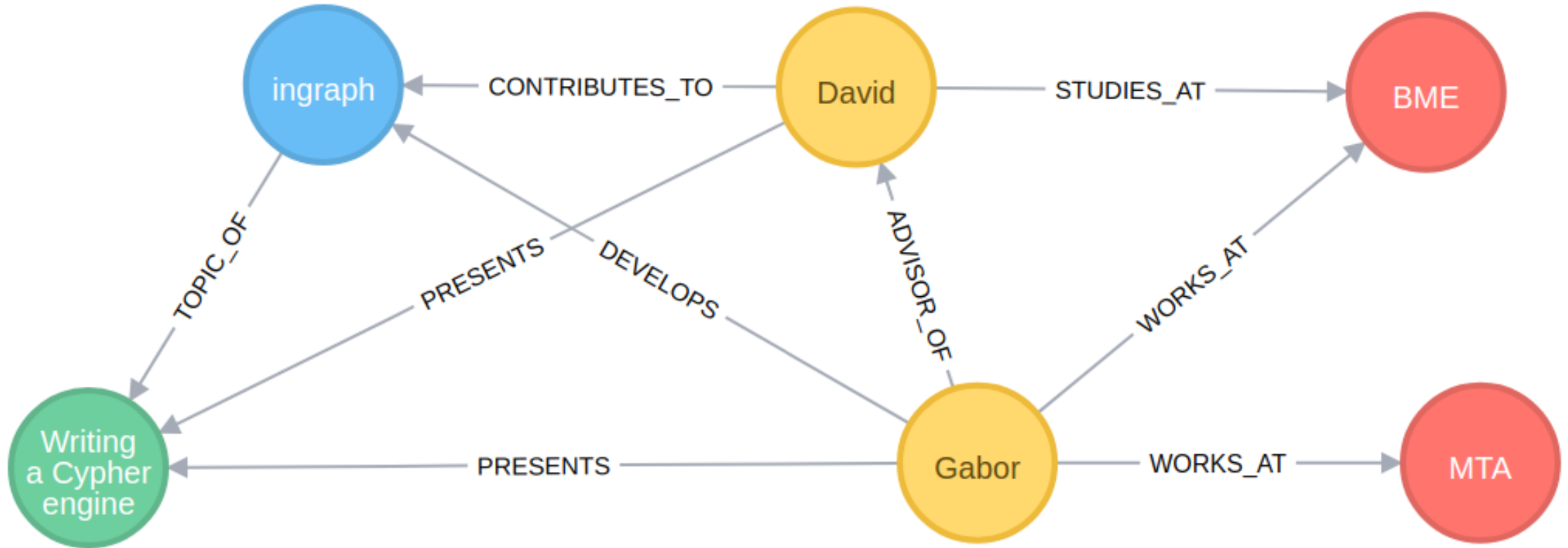
WORKS_AT(2)

PRESENTS(2)

CONTRIBUTES_TO(1)

STUDIES_AT(1)

TOPIC_OF(1)



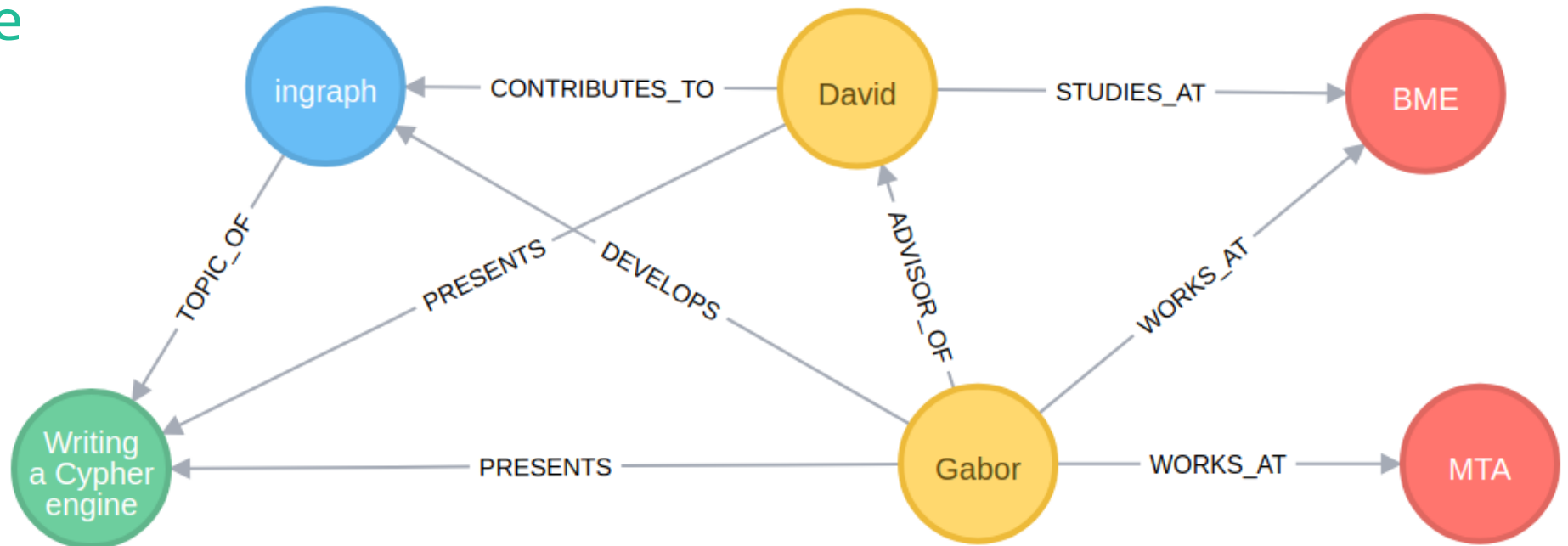
Project <id>: 62 languages: Scala,Clojure name: ingraph url: https://github.com/FTSRG/ingraph

CYPHER

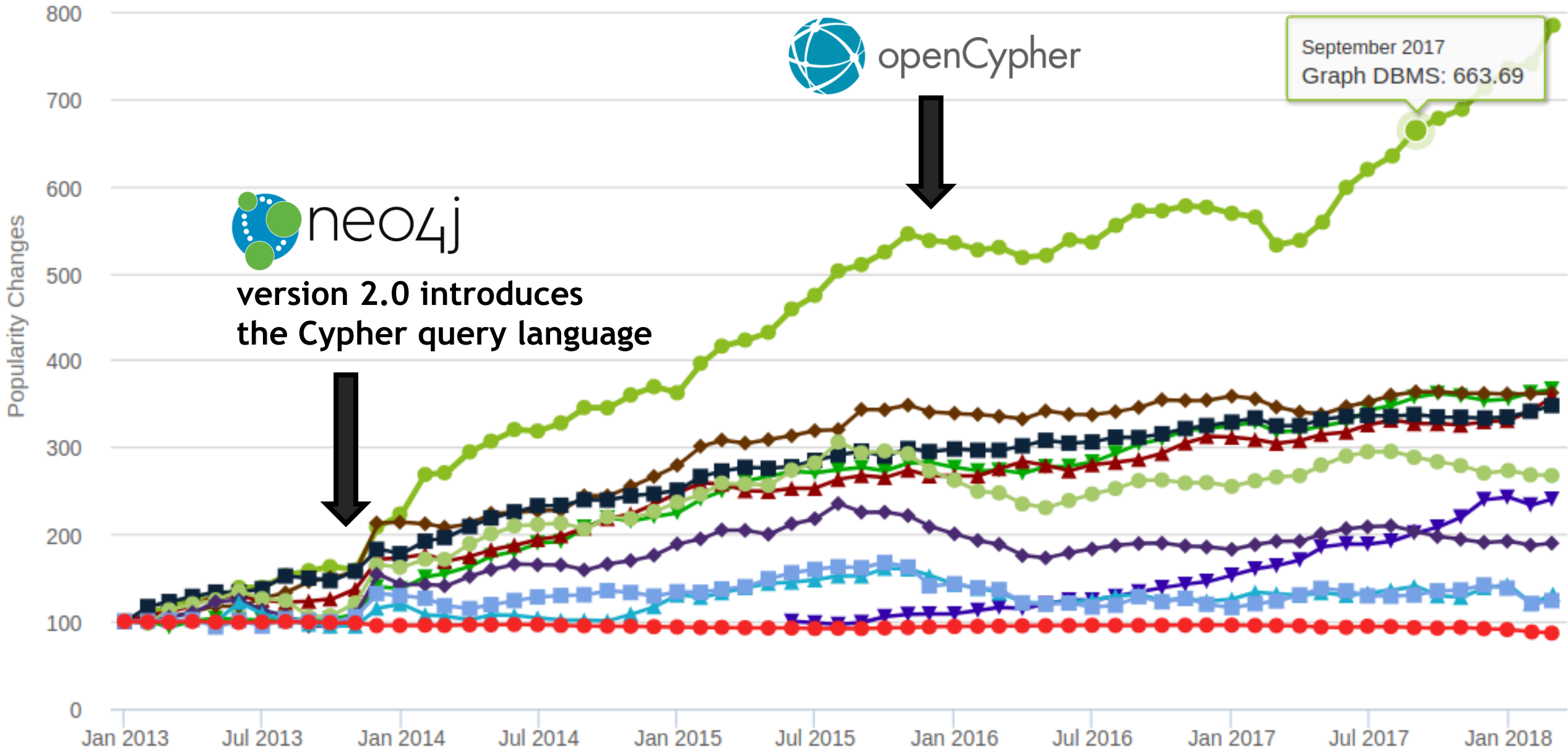
„Cypher is a declarative, SQL-inspired language for describing patterns in graphs visually using an ascii-art syntax.”


```
MATCH (pers:Person)-[:PRESENTERS]->
      (:Presentation)<-[:TOPIC_OF]->(proj:Project)
WHERE proj.name = 'ingraph'
RETURN pers.name
```


pers.name
David
Gabor



DBMS POPULARITY BY DATA MODEL



 neo4j
version 2.0 introduces
the Cypher query language

 openCypher

September 2017
Graph DBMS: 663.69

OPENCYIPHER SYSTEMS

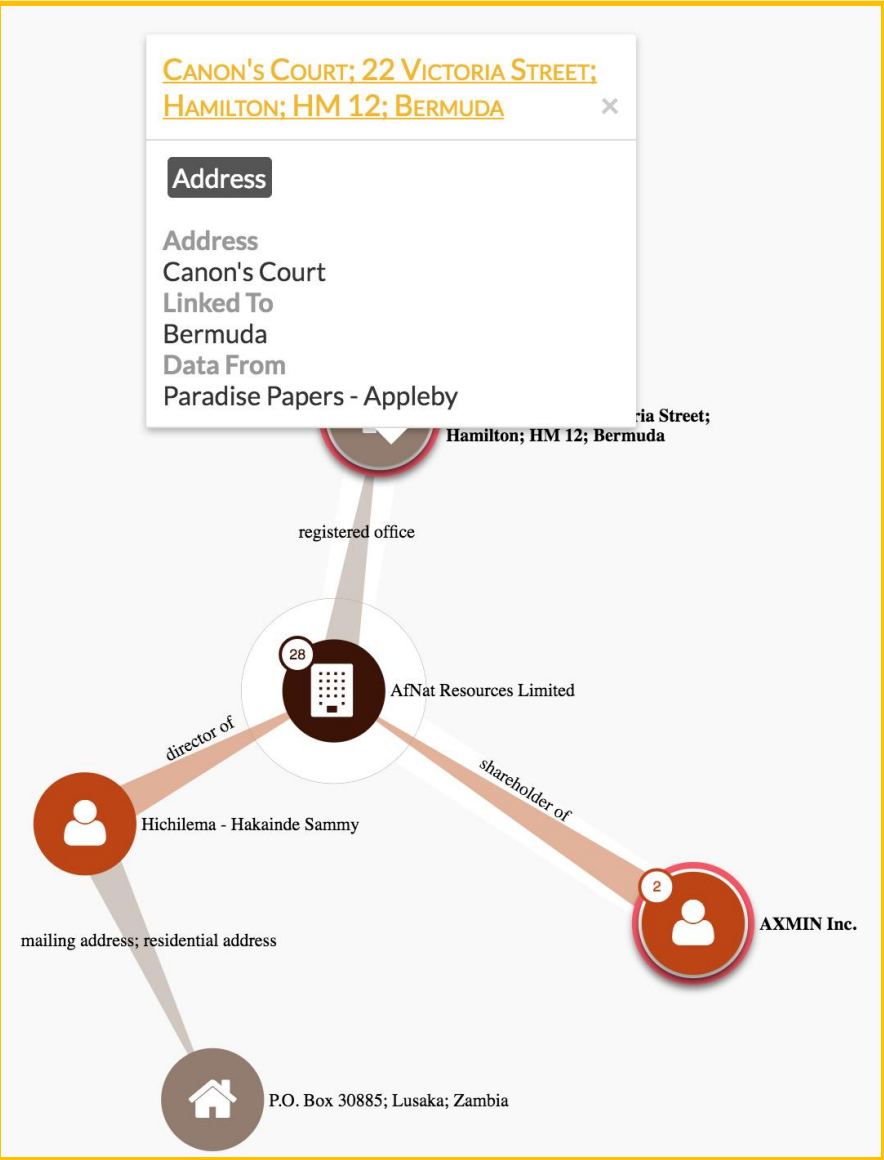
- Goal: deliver a full and open specification of Cypher
- Relational databases:
 - SAP HANA
 - AGENS Graph
- Research prototypes:
 - Graphflow (University of Waterloo)
 - ingraph (incremental graph engine)



(Source: Keynote talk @ GraphConnect NYC 2017)

USE CASES

- Analytics
 - IT security
 - Investigative journalism
- “Real-time” execution
 - Fraud detection
 - Recommendation systems



Walmart

Walmart uses Neo4j to optimize customer experience with personal recommendations

INGRAPH

- MTA-BME project
- PoC query engine for openCypher
- Primarily written in Scala
- Goals:
 - Provide continuous evaluation (*incremental view maintenance*)
 - Cover standard openCypher constructs
 - Does not work efficiently for one-time query execution



Szárnyas, G. et al.,

IncQuery-D: A distributed incremental model query framework in the cloud.

MODELS, 2014,

https://link.springer.com/chapter/10.1007/978-3-319-11653-2_40

Result and subresult operations. Rules for RETURN also apply to WITH .		
$\llbracket r \rrbracket$ RETURN $\langle x1 \rangle$ AS $\langle y1 \rangle$, ...	$\pi_{x1 \rightarrow y1, \dots}(r)$	(15)
$\llbracket r \rrbracket$ RETURN DISTINCT $\langle x1 \rangle$ AS $\langle y1 \rangle$, ...	$\delta(\pi_{x1 \rightarrow y1, \dots}(r))$	(16)
$\llbracket r \rrbracket$ RETURN $\langle x1 \rangle$, $\langle \text{aggr} \rangle(\langle x2 \rangle)$	$\gamma_{x1, \text{aggr}(x2)}^{x1}(r)$ (see Sec. 3.1)	(17)
$\llbracket r \rrbracket$ WITH $\langle x1 \rangle$ $\llbracket s \rrbracket$ RETURN $\langle x2 \rangle$	$\pi_{x2}\left(\left(\pi_{x1}(r)\right) \bowtie s\right)$	(18)
Unwinding and list operations		
$\llbracket r \rrbracket$ UNWIND $\langle xs \rangle$ AS $\langle x \rangle$	$\omega_{xs \rightarrow x}(r)$	(19)
$\llbracket r \rrbracket$ ORDER BY $\langle x1 \rangle$ ASC , $\langle x2 \rangle$ DESC , ...	$\tau_{\uparrow x1, \downarrow x2, \dots}(r)$	(20)
$\llbracket r \rrbracket$ SKIP $\langle s \rangle$ LIMIT $\langle l \rangle$	$\lambda_l^s(r)$	(21)
Combining results		
$\llbracket r \rrbracket$ UNION $\llbracket s \rrbracket$	$r \cup s$	(22)
$\llbracket r \rrbracket$ UNION ALL $\llbracket s \rrbracket$	$r \uplus s$	(23)

Table 2: Mapping from openCypher constructs to relational algebra. Variables, labels, types and literals are typeset as $\langle v \rangle$. The notation $\llbracket p \rrbracket$ represents patterns resulting in a relation p , while $\llbracket r \rrbracket$ denotes previous query parts resulting in a relation r . To avoid confusion with the “..” language construct (used for ranges), we use ... to denote omitted query parts.

Language construct	Relational algebra expression	
Vertices and patterns. (p) denotes a pattern that contains a vertex $\langle v \rangle$.		
$\langle v \rangle$	$\bigcirc_{(v)}$	①
$\langle v \rangle : \langle l_1 \rangle : \dots : \langle l_n \rangle$	$\bigcirc_{(v: l_1 \wedge \dots \wedge l_n)}$	②
$(p) - [\langle e \rangle : \langle t_1 \rangle \dots \langle t_k \rangle] \rightarrow \langle w \rangle$	$\uparrow_{(v)}^{(w)} [e: t_1 \vee \dots \vee t_k] (p)$, where e is an edge	③
$(p) \leftarrow [\langle e \rangle : \langle t_1 \rangle \dots \langle t_k \rangle] - \langle w \rangle$	$\downarrow_{(v)}^{(w)} [e: t_1 \vee \dots \vee t_k] (p)$, where e is an edge	④
$(p) \leftarrow [\langle e \rangle : \langle t_1 \rangle \dots \langle t_k \rangle] \rightarrow \langle w \rangle$	$\updownarrow_{(v)}^{(w)} [e: t_1 \vee \dots \vee t_k] (p)$, where e is an edge	⑤
$(p) - [\langle e \rangle * \langle \min \rangle .. \langle \max \rangle] \rightarrow \langle w \rangle$	$\uparrow_{(v)}^{(w)} [e *_{\min}^{\max}] (p)$, where e is a list of edges	⑥
Combining and filtering pattern matches		
MATCH (p_1) , (p_2) , ...	$\neq_{\text{edges of } p_1, p_2, \dots} (p_1 \bowtie p_2 \bowtie \dots)$	⑦
MATCH (p_1) MATCH (p_2)	$\neq_{\text{edges of } p_1} (p_1) \bowtie \neq_{\text{edges of } p_2} (p_2)$	⑧
OPTIONAL MATCH (p)	$\{\langle \rangle\} \bowtie \neq_{\text{edges of } p} (p)$	⑨
OPTIONAL MATCH (p) WHERE (condition)	$\{\langle \rangle\} \bowtie_{\text{condition}} \neq_{\text{edges of } p} (p)$	⑩
[[r]] OPTIONAL MATCH (p)	$\neq_{\text{edges of } r} (r) \bowtie \neq_{\text{edges of } p} (p)$	⑪
[[r]] WHERE $\langle \text{condition} \rangle$	$\sigma_{\text{condition}}(r)$	⑫
[[r]] WHERE $\langle v \rangle : \langle l_1 \rangle : \dots : \langle l_n \rangle$	$\sigma_{v.l=l_1 \wedge \dots \wedge n.l=l_n}(r)$	⑬
[[r]] WHERE (p)	$r \bowtie p$	⑭

GRAPH QUERY PROCESSING APPROACHES

- Relational: projection, selection, join
 - SAP HANA
 - Agens Graph (PostgreSQL + Cypher)
 - ingraph
- Relational + expand
 - Neo4j
- Constraint satisfaction
 - VIATRA (BME project since 2002): “local search” engine
 - Our proposed openCypher engine (this talk)

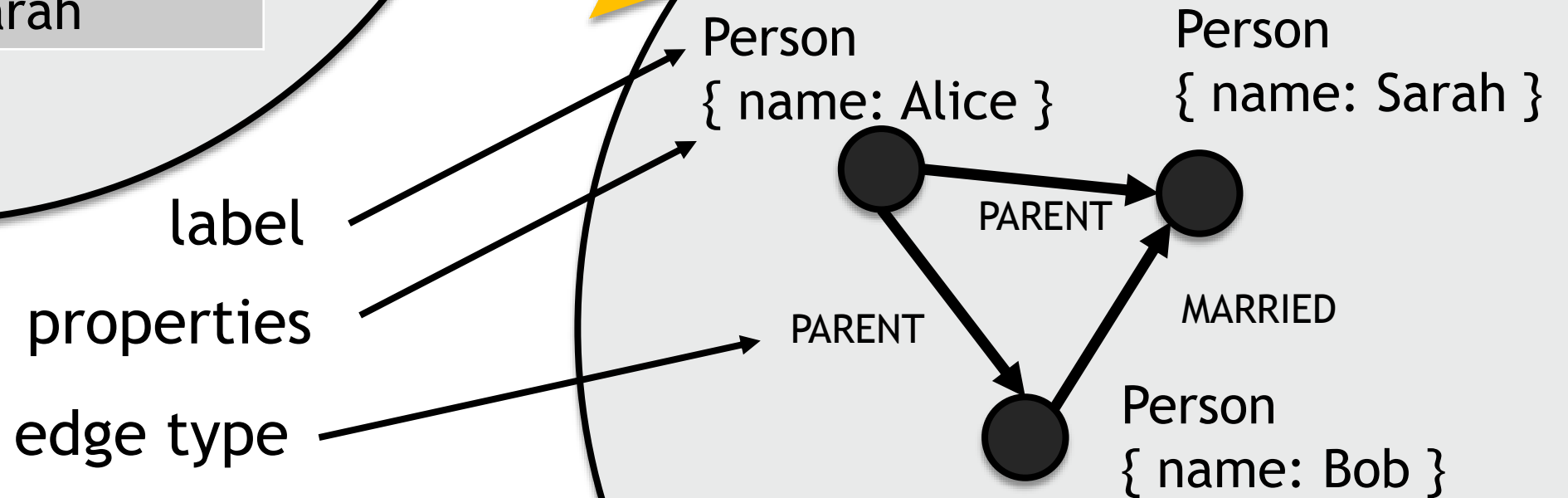
RELATIONAL MODEL

Parent	
person	parent
Alice	Sarah
Alice	Bob

Married	
a	b
Bob	Sarah

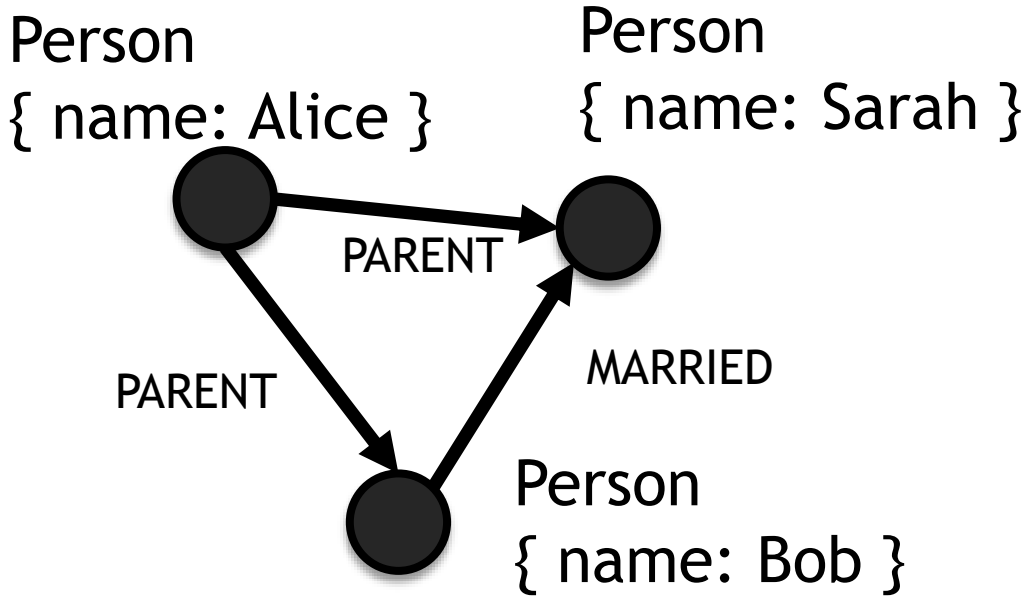


PROPERTY GRAPH



EXPLORATION BASED METHODS

- given this graph

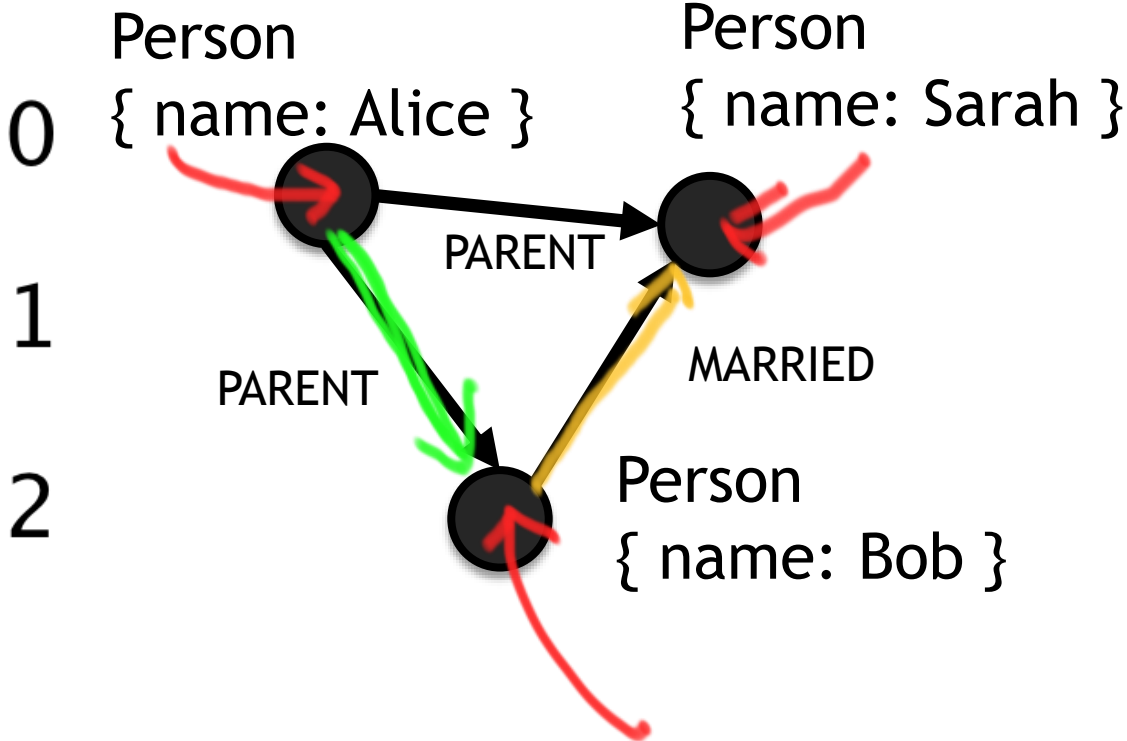
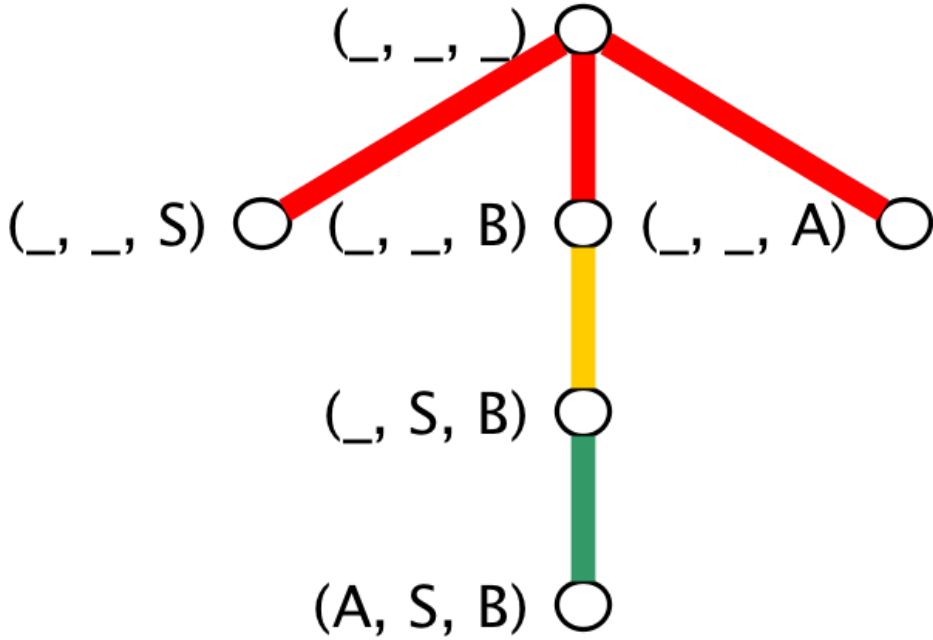


- and query

```
MATCH (u:Person)
      -[:PARENT]->
      (f:Person)
      -[m:MARRIED]->
      (m:Person)
RETURN m, f
```

find an execution plan

EXPLORATION BASED METHODS



- DFT -> small memory footprint
- Small state-space -> less steps

EXPLORATION BASED METHODS

Goal: minimize state-space generated for a query.

- Cost based optimizations
- Constraint satisfaction problems
- SAT is an NP-full problem
- Greedy algorithm unsatisfactory

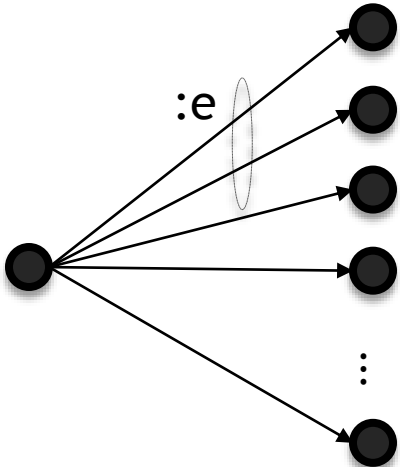
EXPLORATION BASED METHODS

Goal: minimize state-space generated for a query.

- Iterative Dynamic Programming (IDP):
 - Generalization of DP and greedy algorithms
 - Polynomial complexity
 - Multiple variants
 - Widely researched in the RDMS area

-> use iterative dynamic programming

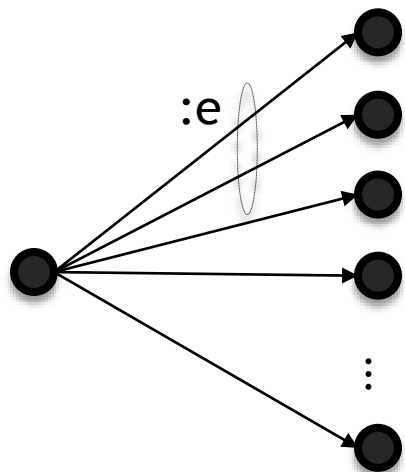
FULL VS GREEDY VS IDP



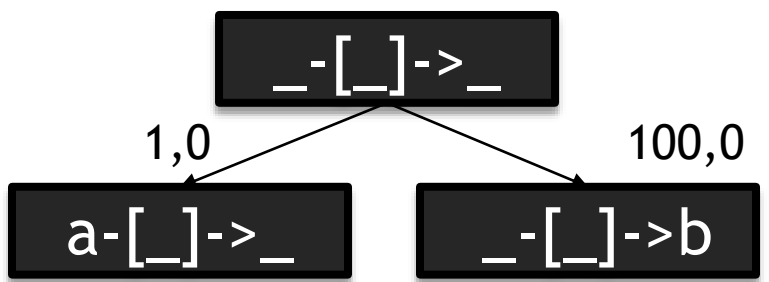
`_ -[_]-> _`

```
MATCH (a) -[:e]->(b)  
RETURN a, b
```

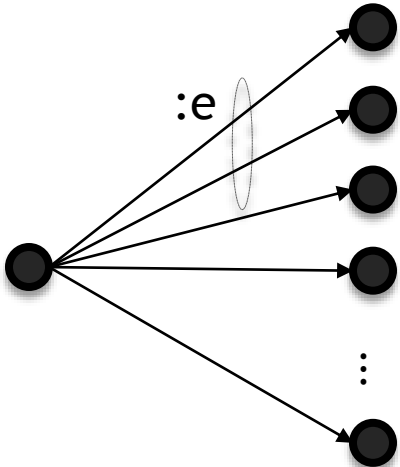
FULL VS GREEDY VS IDP



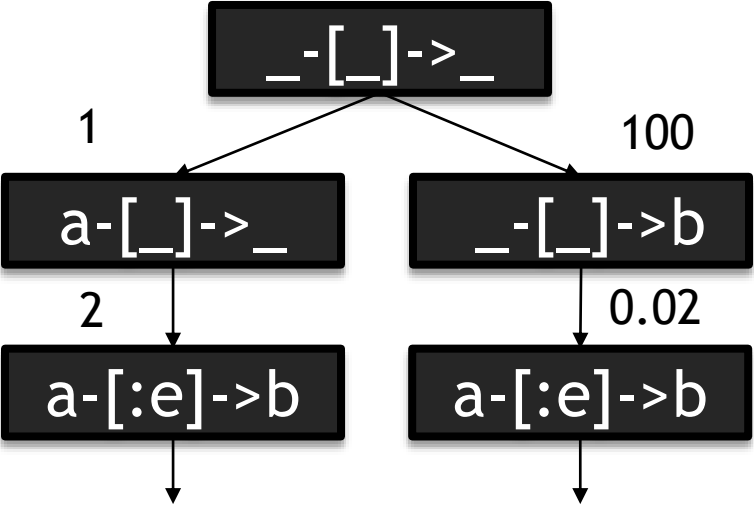
```
MATCH (a)-[:e]->(b)  
RETURN a, b
```



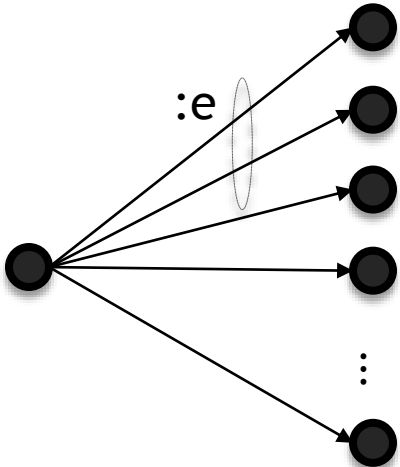
FULL VS GREEDY VS IDP



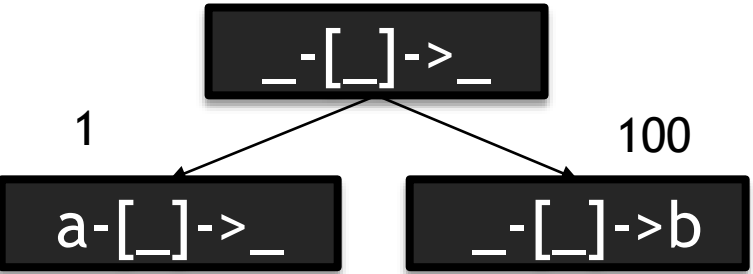
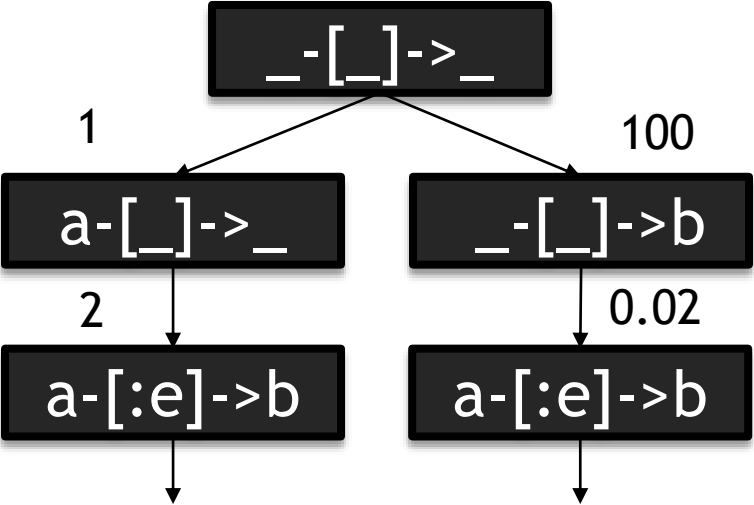
```
MATCH (a)-[:e]->(b)  
RETURN a, b
```



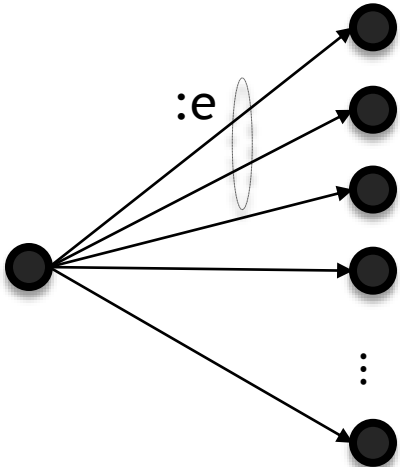
FULL VS GREEDY VS IDP



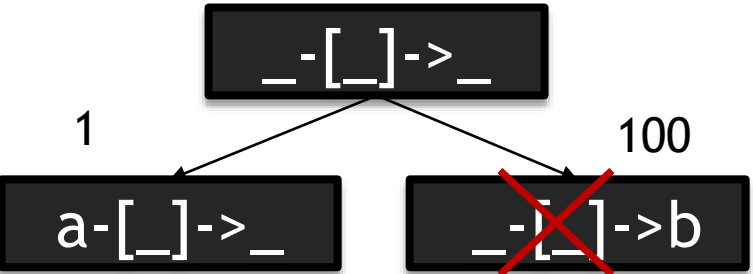
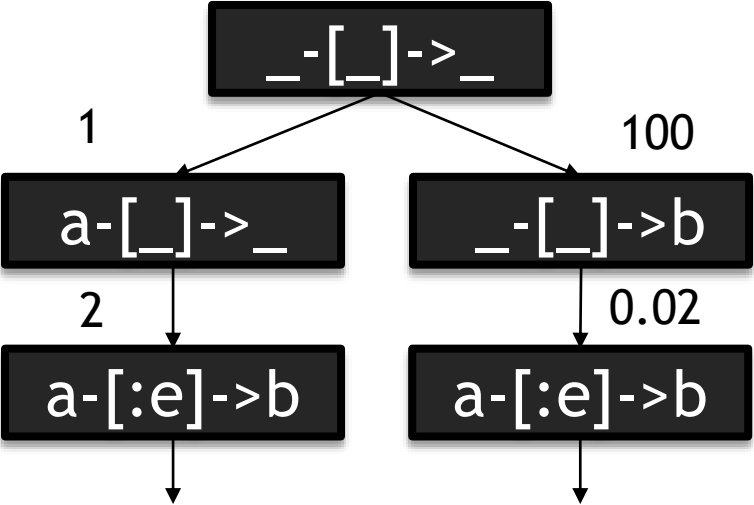
```
MATCH (a)-[:e]->(b)  
RETURN a, b
```



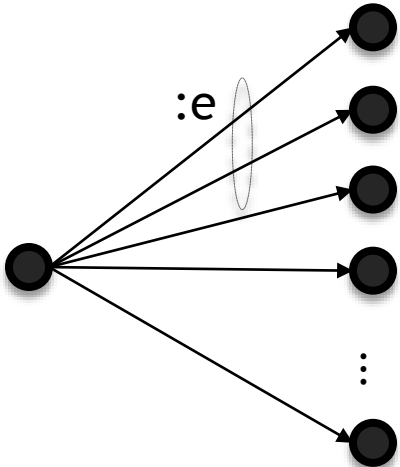
FULL VS GREEDY VS IDP



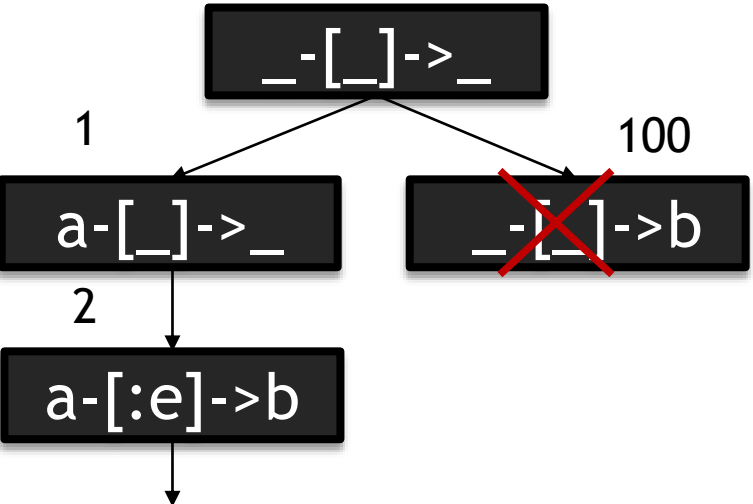
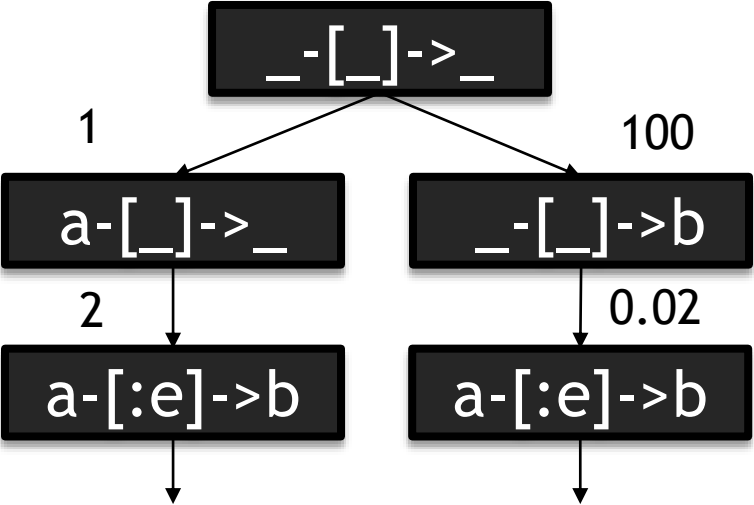
```
MATCH (a)-[:e]->(b)  
RETURN a, b
```



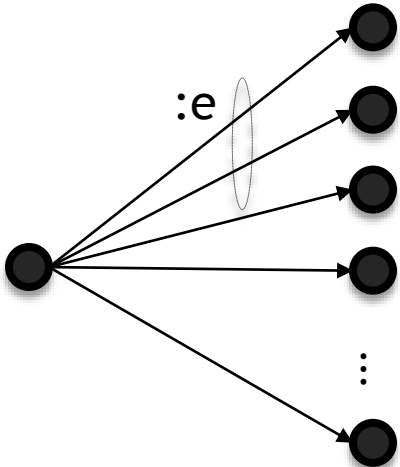
FULL VS GREEDY VS IDP



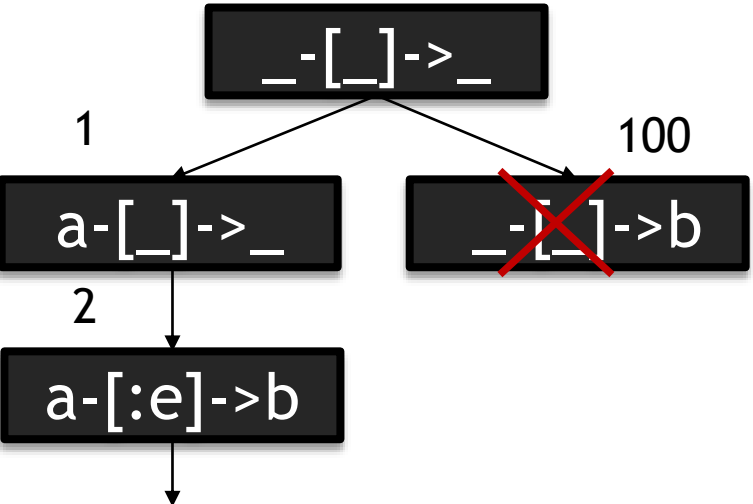
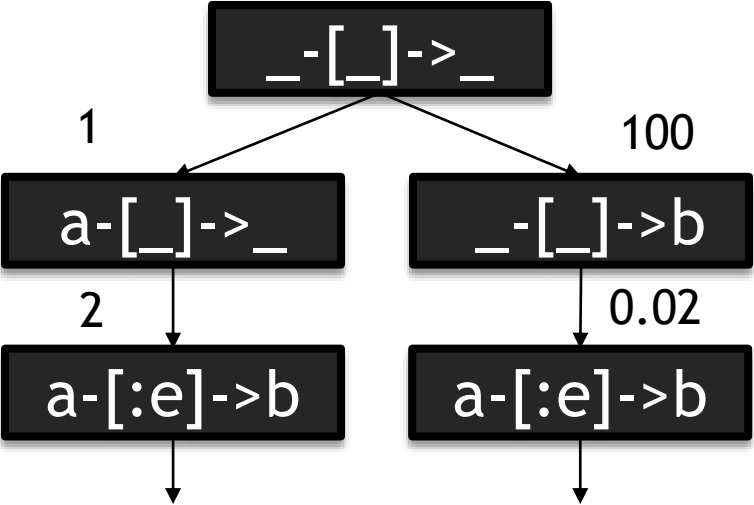
```
MATCH (a)-[:e]->(b)  
RETURN a, b
```



FULL VS GREEDY VS IDP



MATCH (a)-[:e]->(b)
RETURN a, b



IDP: Something in between...

MODEL-SENSITIVE APPROACH



G. Varró, F. Deckwerth, M. Wieber, A. Schürr,
*An algorithm for generating model-sensitive search plans for pattern
matching on EMF models,*
Software and Systems Modeling, 2013



- Derives cost based on the graph elements
- Object-oriented approach
- Different goal: model validation

Idea: adapt to property graphs.

ADAPTING IT

Requirements

- Variables for edges and vertices
- Estimate based on the graph
- Map to specialized indexer operations

Difficulties

- No type information
- Original algorithm oblivious to edges with properties

CONSTRAINT DEFINITIONS

- Constraints and operations are separate
- Constraints can imply other constraints

```
(defconstraint Known [known])
(defconstraint Element [element] < Known [element])
(defconstraint Edge [edge] < Element [edge])
(defconstraint Vertex [vertex] < Element [vertex])
(defconstraint HasLabels [vertex labels]
  < Vertex [vertex] Known [labels])
(defconstraint HasType [edge type] <
  Edge [edge] Known [type])
(defconstraint Property [element key value]
  < Element [element] Known [key] Known [value])
(defconstraint DirectedEdge [source edge target]
  < Vertex [source] Edge [edge] Vertex [target])
(defconstraint GenUnaryAssertion [x cond]
  < Known [x] Known [cond])
(defconstraint GenBinaryAssertion [x y cond]
  < Known [x] Known [y] Known [cond])
(defconstraint Constant [x value] < Known [x])
```

OPERATIONS - CONSTRAINTS

```
(defop MyOp [& vars] preconditions* -> postconditions*)
```

```
(defop  
  GetEdges [source edge target] -> DirectedEdge [source edge target])
```

```
(defop  
  GetEdgesByType [source edge target type]  
  Known [type] -> DirectedEdge [source edge target]  
  HasType [edge type])
```

```
(defop  
  AccessPropertyByKey [element key val]  
  Element [element] Known [key] -> Property [element key val]  
  :opts {:immediate true})
```

```
(defop  
  ExtendOut [source edge target]  
  Vertex [source] -> DirectedEdge [source edge target])
```


OPERATIONS - CONSTRAINTS

```
(defop MyOp [& vars] preconditions* -> postconditions*)
```

```
(defop  
  GetEdges [source edge target] -> DirectedEdge [source edge target])
```

```
(defop  
  GetEdgesByType [source edge target type]  
  Known [type] -> DirectedEdge [source edge target]  
  HasType [edge type])
```

```
(defop  
  AccessPropertyByKey [element key val]  
  Element [element] Known [key] -> Property [element key val]  
  :opts {:immediate true})
```

```
(defop  
  ExtendOut [source edge target]  
  Vertex [source] -> DirectedEdge [source edge target])
```

IMMEDIATE

OPERATIONS - COSTS

```
(defweight op args estimator-body)
```

- indexer support (context sensitive)
- Based on variables referenced in the constraints
- Given a (HasLabels a t) and a (Vertex a) constraints we can get the number of vertices for that label

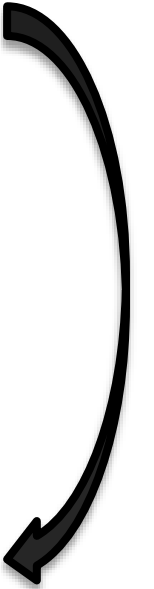
```
def getAverageNumberOfLabelsPerVertices(): Float  
def getNumberOfVerticesWithLabel(label: String): Int  
def getNumberOfEdgesWithType(`type`: String): Int
```

SIMPLE PATTERNS

Conjunction of positive terms

```
MATCH (segment:Segment)
WHERE segment.length <= 0
RETURN
    segment,
    segment.length AS length
```

```
(pattern/pattern [:segment :length] [])
  [(bind proto-1/Vertex [:segment])
   (bind proto-1/HasLabels [:segment :-segment-label])
   (bind proto-1/Constant [-segment-label "Segment"])
   (bind proto-1/Property [:segment :-length-key :length])
   (bind proto-1/Constant [-length-key "length"])
   (bind proto-1/GenBinaryAssertion [:length :-0 :-<=])
   (bind proto-1/Constant [-0 0])
   (bind proto-1/Constant [-<= <=]))]
```



NEGATIVE PATTERNS

Negation needs at least one bound variable

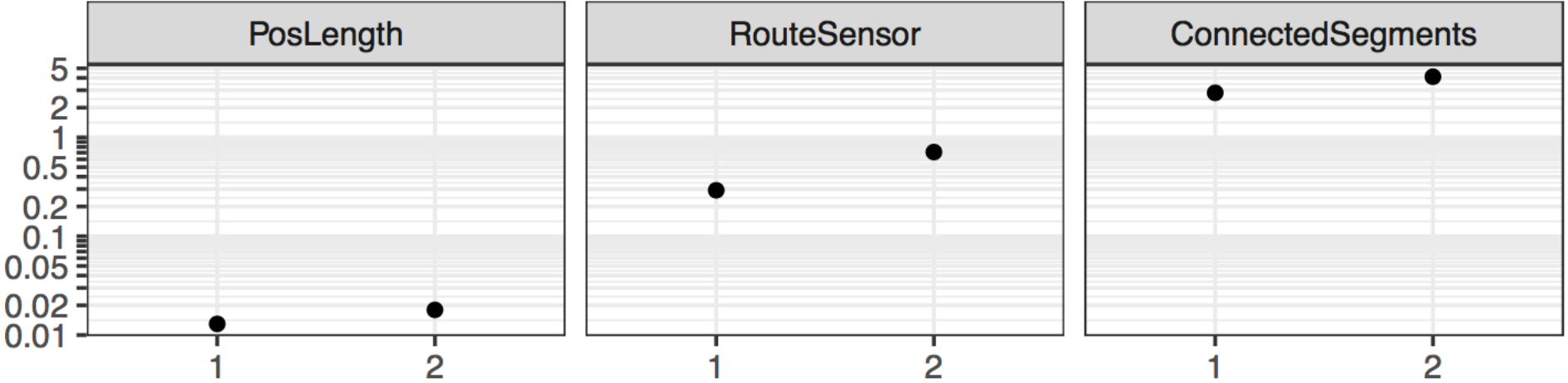
```
MATCH (r:Route)-[:follows]->(swP:SwitchPosition),  
      (swP:SwitchPosition)-[:target]->(sw:Switch),  
      (sw:Switch)-[:monitoredBy]->(sensor:Sensor)  
WHERE NOT (r)-[:requires]->(sensor)  
RETURN r, sensor, swP, sw
```

Compiles to a single constraint with inner constraints.

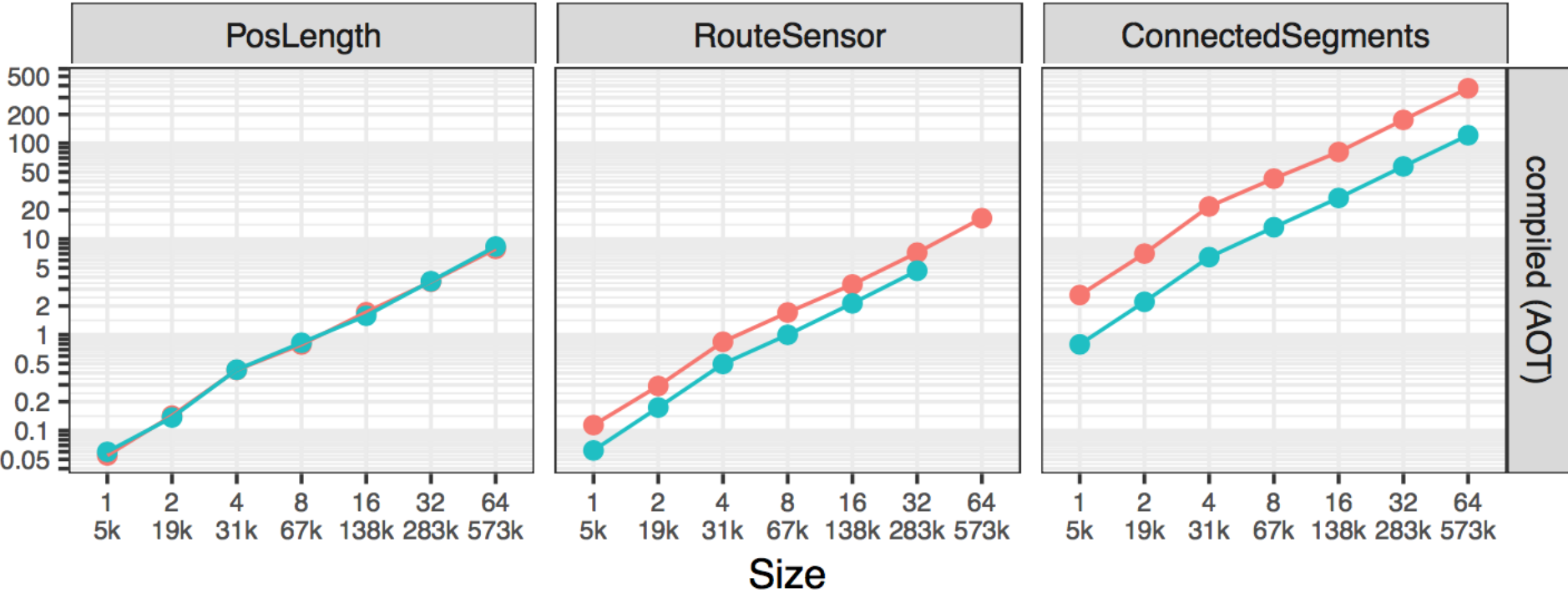
```
(bind (pattern/not  
      (pattern/pattern [:route :sensor]  
        [(bind proto-1/Vertex [:route])  
         (bind proto-1/Vertex [:sensor])])  
        [(bind proto-1/DirectedEdge [:route :-requires :sensor])  
         (bind proto-1/HasType [:-requires :-requires-type])  
         (bind proto-1/Constant [:-requires-type "requires"])])))  
[:route :sensor])
```

BENCHMARKING

design



execution



CONCLUSION

- Difficult to adapt this OO algorithm to property graphs
- Clojure is well fitted to the problem
- Subpar performance from the initial implementation



Dávid Szakállas: *Evaluation of openCypher Graph Queries with a Local Search-Based Algorithm*. Master's thesis, Budapest University of Technology and Economics, 2017.



FTSRG/ingraph

- Looking for maintainers
- Interested? Contact us: [@szarnyasg](#) [@szdavid92](#)