Graph Pattern Matching

- Cypher today
- Coming up in Cypher: quantified path patterns

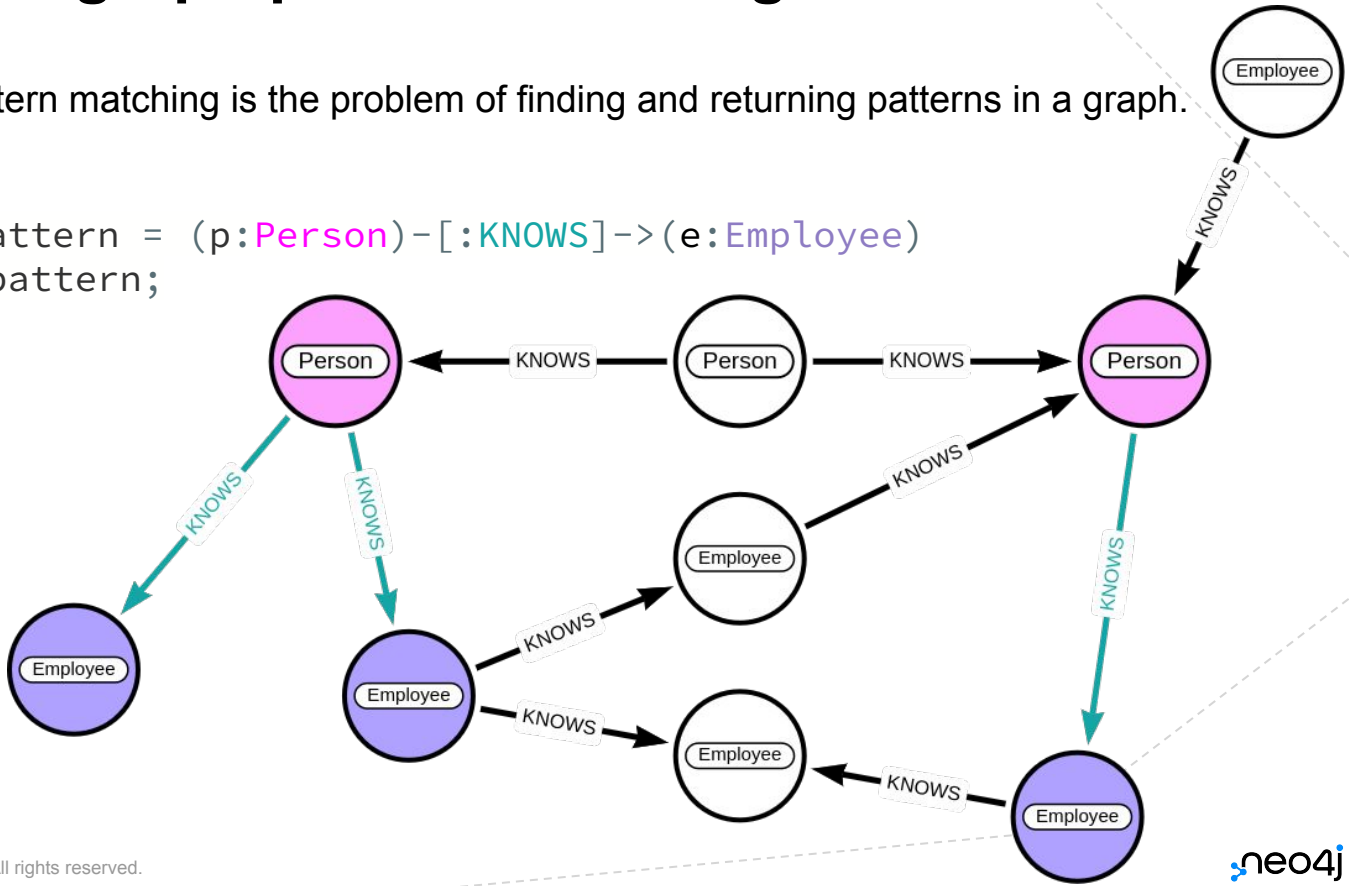
16 November, 2022

Nadja Müller and Petra Selmer

What is graph pattern matching?

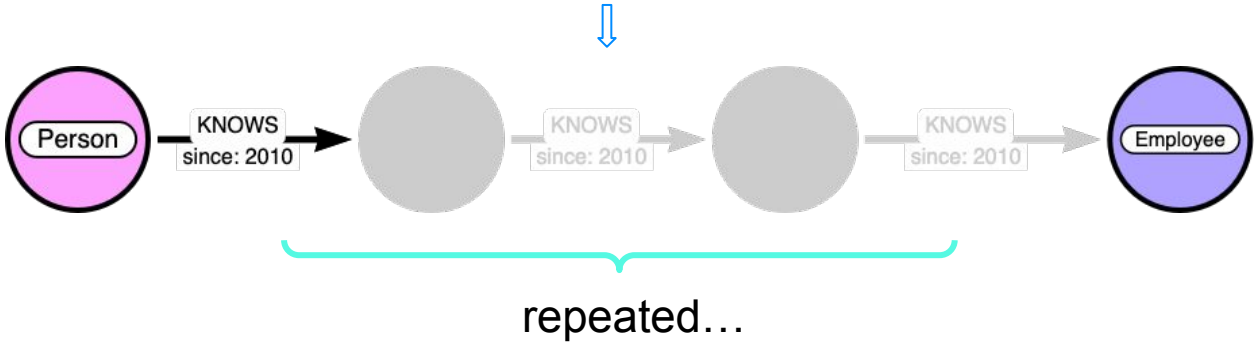
Graph pattern matching is the problem of finding and returning patterns in a graph.

```
MATCH pattern = (p:Person)-[:KNOWS]->(e:Employee)
RETURN pattern;
```



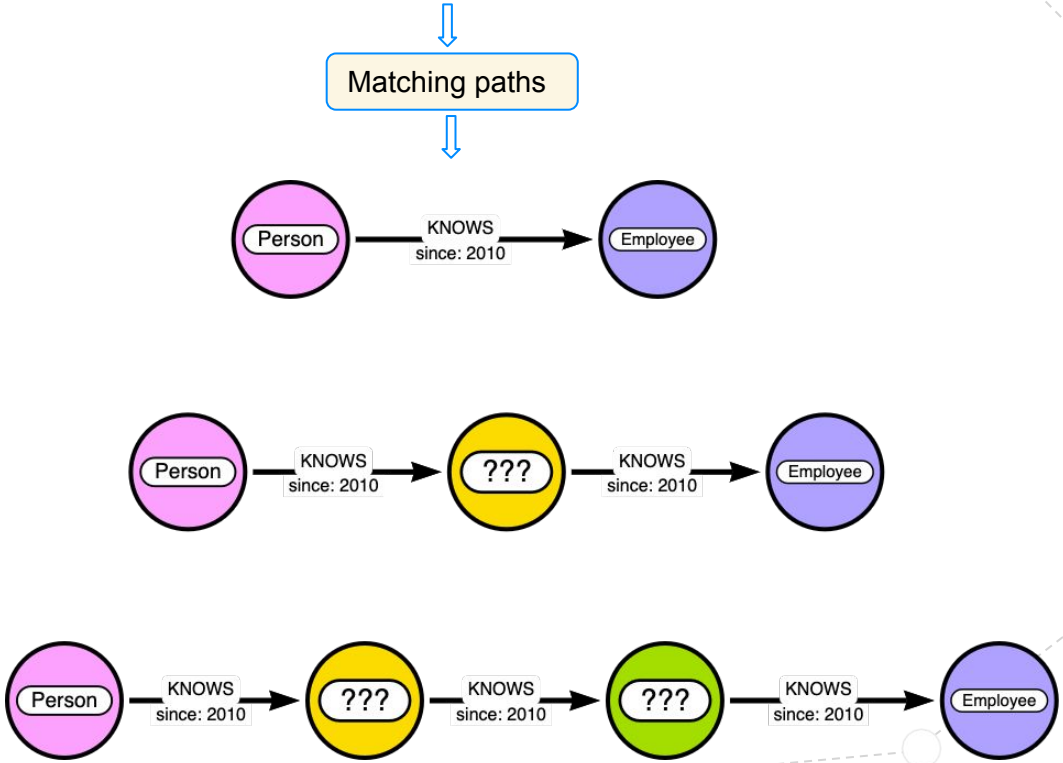
Variable length pattern matching

```
MATCH (p:Person)-[:KNOWS* {since: 2010}]->(e:Employee)
```

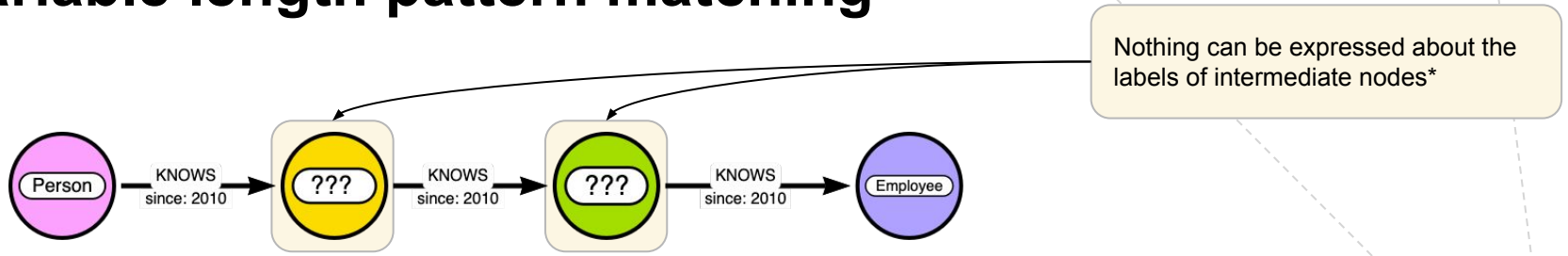


Variable length pattern matching

```
MATCH (p:Person)-[:KNOWS* {since: 2010}]->(e:Employee)
```

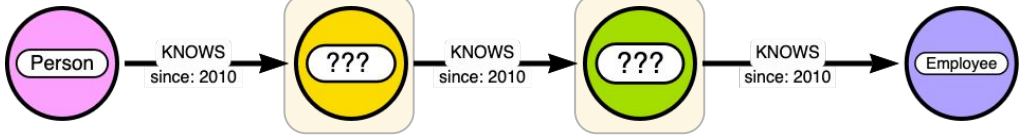


Variable length pattern matching

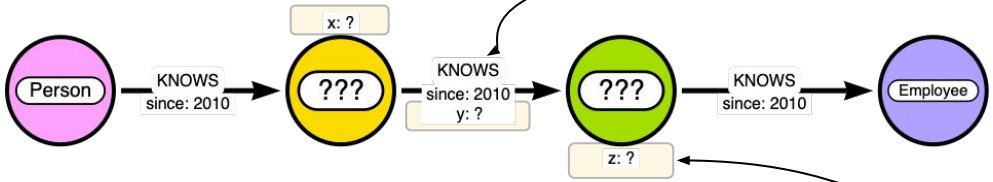


*Predicate functions can be used in some cases, but these are limited and can get quite convoluted

Variable length pattern matching



Nothing can be expressed about the labels of intermediate nodes*

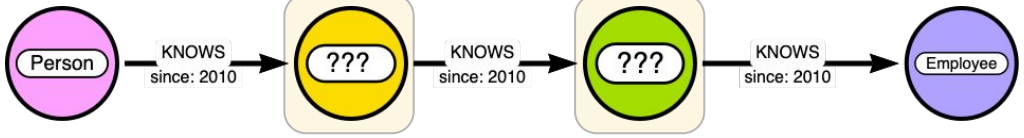


Only equality predicates can be expressed about the properties of relationships*

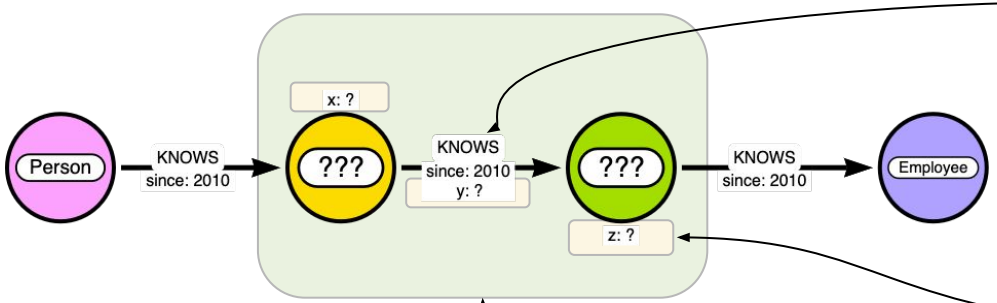
Nothing can be expressed about the properties of intermediate nodes*

*Predicate functions can be used in some cases, but these are limited and can get quite convoluted

Variable length pattern matching



Nothing can be expressed about the labels of intermediate nodes*



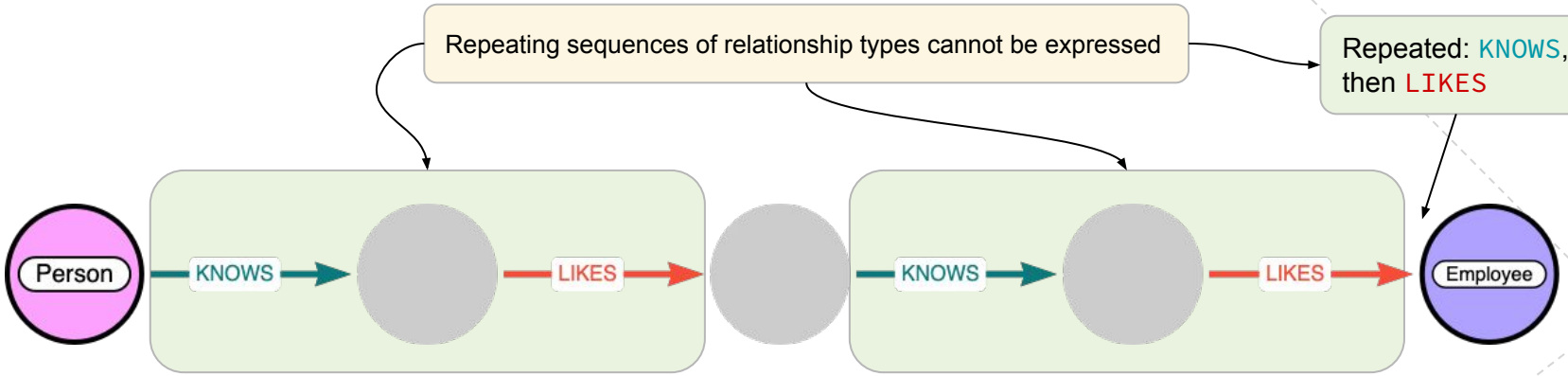
Only equality predicates can be expressed about the properties of relationships*

Nothing can be expressed about the properties of intermediate nodes*

Nothing can be expressed about the properties of intermediate nodes and relationships in relation to each other

*Predicate functions can be used in some cases, but these are limited and can get quite convoluted

Variable length pattern matching



Graph pattern matching alternatives

```
MATCH (p:Person)
CALL apoc.path.expandConfig(p, {
relationshipFilter: "KNOWS>,LIKES>",
labelFilter: ">Employee"
})
YIELD path
RETURN path
```

apoc & Cypher (Traversal API)

Why we are improving Graph Pattern Matching

```
MATCH (p:Person)
CALL apoc.path.expandConfig(p, {
relationshipFilter: "KNOWS>,LIKES>",
labelFilter: ">Employee"
})
YIELD path
RETURN path
```

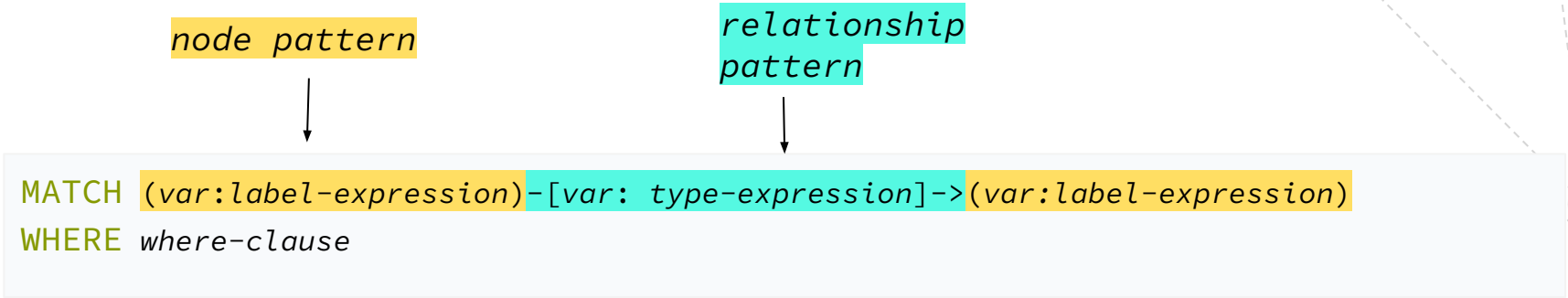
apoc & Cypher (Traversal API)

```
MATCH path =(p:Person)(()-[:KNOWS]->()-[:LIKES]->())+(e:Employee)
RETURN path
```

Cypher with advanced Graph Pattern Matching
+ readability
+ memory tracking
+ maintainability

What we have been working on

Terminology



WHERE clause in node patterns

4.4

```
MATCH (p:Person WHERE p.age > 18)
RETURN *
```

not supported in var length and shortest path patterns

WHERE clause in relationship patterns

5.0

```
MATCH ()-[k:KNOWS { related: false } WHERE k.since < 2020]-()
RETURN *
```

not supported in var length and shortest path patterns

Additional operators in label and type expressions

5.0

```
MATCH (a:!(A|B))-[r:!R&!S]->(:%)
RETURN *
```

&	And
|	Or
!	Not
( )	Grouping
%	Wildcard

not supported in var length and shortest path patterns

QUIZ: How do you find nodes without any labels?

&	And
|	Or
!	Not
( )	Grouping
%	Wildcard

QUIZ: How do you find nodes without any labels?

4.x

```
MATCH (n) WHERE size(labels(n))=0
RETURN *
```

&	And
|	Or
!	Not
( )	Grouping
%	Wildcard

QUIZ: How do you find nodes without any labels?

4.x

```
MATCH (n) WHERE size(labels(n))=0
RETURN *
```

5.0

```
MATCH (n:!%)
RETURN *
```

&	And
|	Or
!	Not
( )	Grouping
%	Wildcard

New directions for graph pattern matching

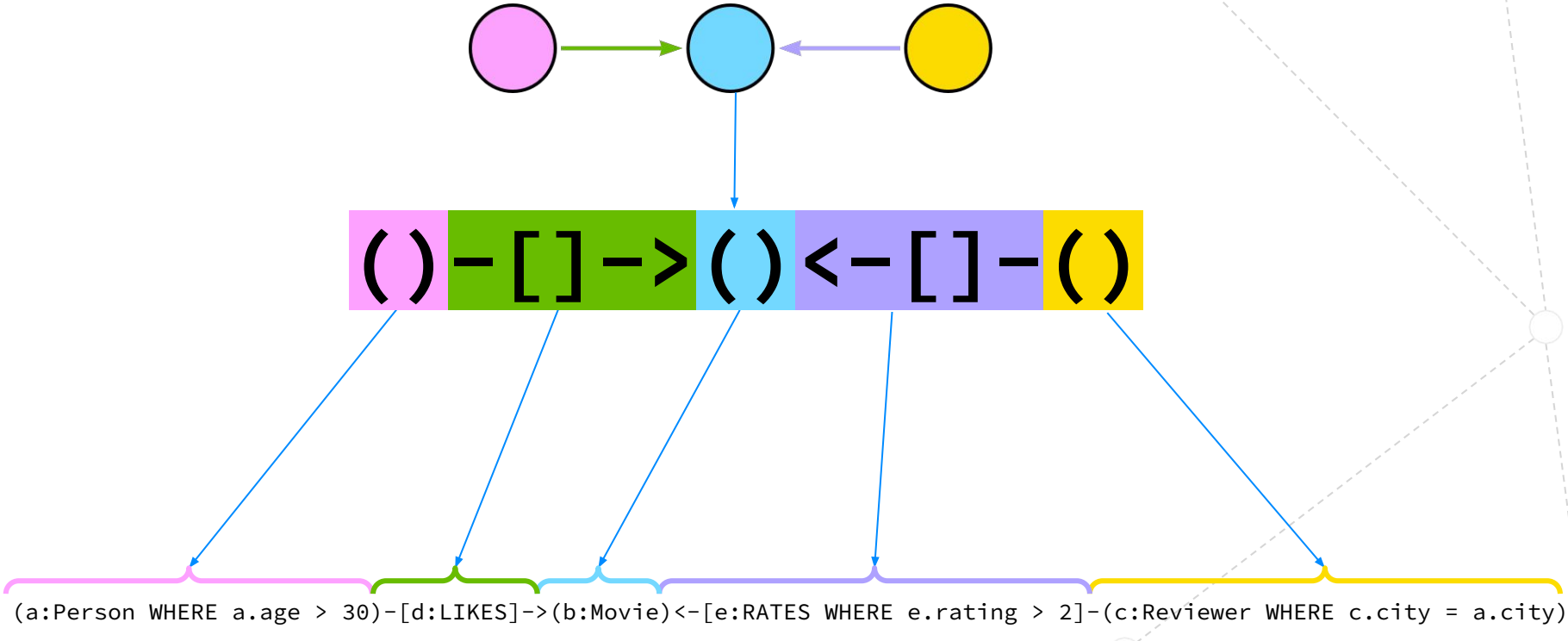
- Part of the upcoming GQL standard
- Adds power and expressivity to the `MATCH` clause
- **Quantified path patterns** are the first and most important of these
  - Prerequisite sub-features already in (inline predicates and complex label expressions)
  - We are currently working on this - will span a number of releases (5.x)
- Subsequent graph pattern-matching features will follow

- These features are for advanced use cases
- No existing graph pattern-matching features are being removed

The next set of slides introduce brand-new concepts

Quantified path patterns -
the future of variable-length path matching

Path pattern



Introducing quantified path patterns

```
MATCH (path pattern){min, max}
```

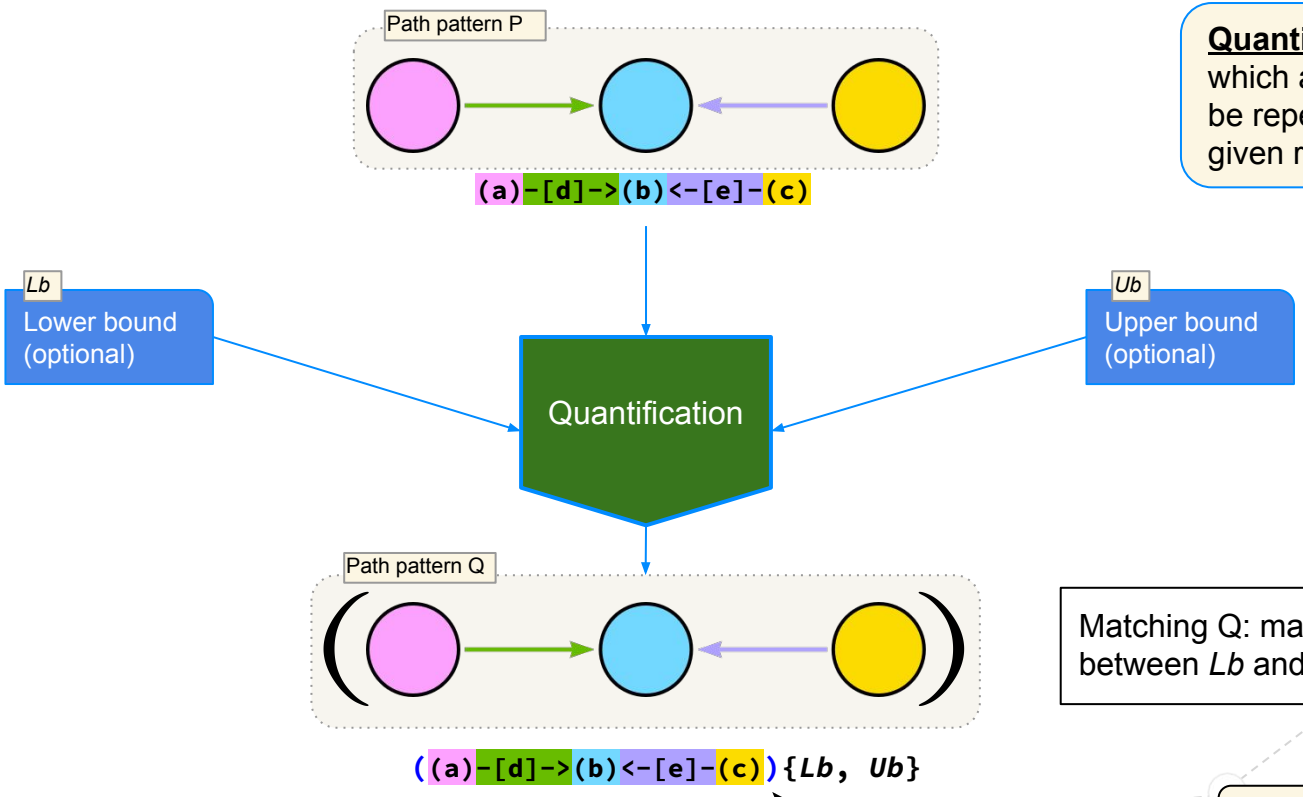
Quantifier

```
MATCH ((a)-[b]->(c)){1,}
```

List of quantifiers

Quantifier	Minimum	Maximum	Semantic Equivalence
Canonical forms			
`{m, n}`	`m`	`n`	
`{m}`	`m`	`m`	`{m, m}`
`{m,}`	`m`	`∞`	`{m, ∞}`
`{,n}`	`0`	`n`	`{0, m}`
Shorthand forms			
`+` *(Kleene +)*	`1`	`∞`	`{1, ∞}`
`*` *(Kleene *)*	`0`	`∞`	`{0, ∞}`

Quantification (or repetition)

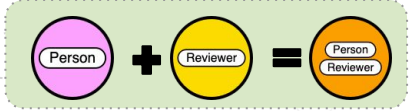


Quantification: the operation by which a path pattern is specified to be repeated a number of times in a given range

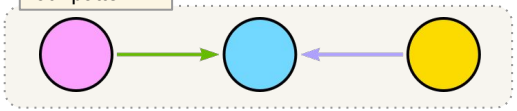
Matching Q: match P repeated between *Lb* and *Ub* times

`()` delimit the scope of the quantifier

```
((a)-[d]->(b)<-[e]-(c)){1, 2}
```

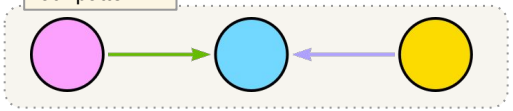


Path pattern P

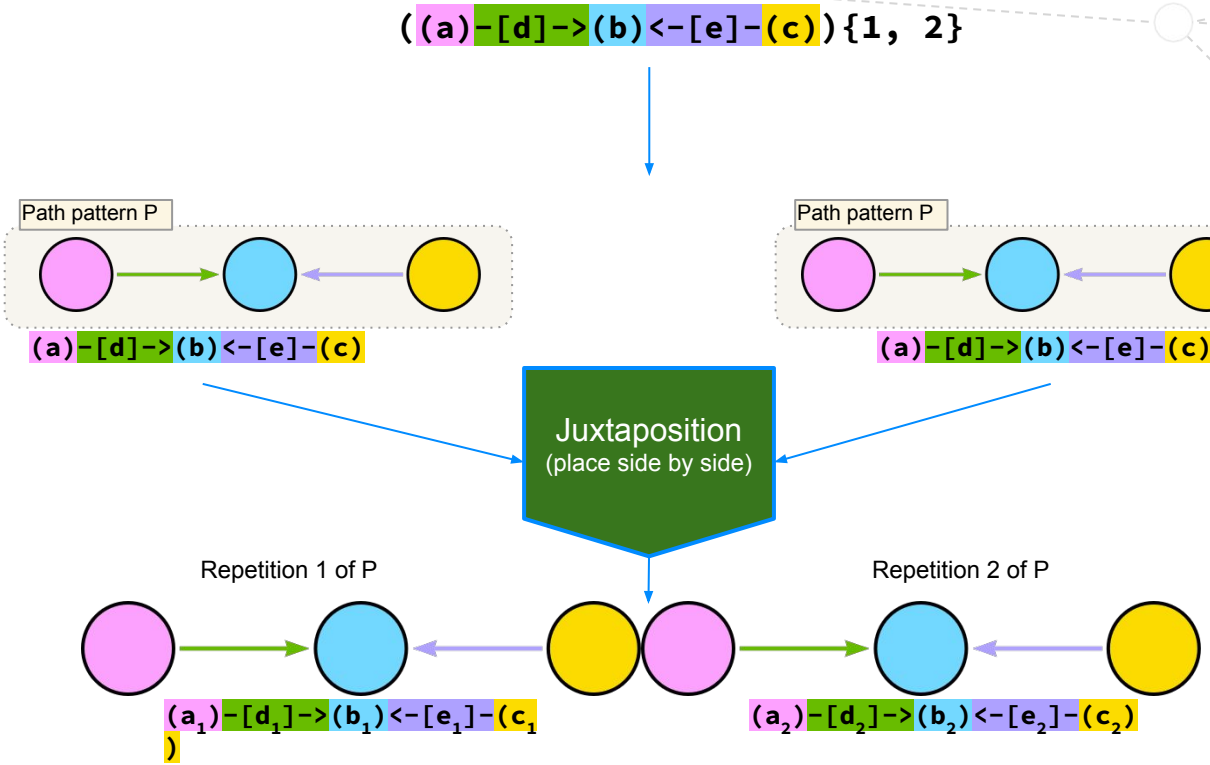


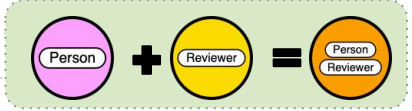
```
(a)-[d]->(b)<-[e]-(c)
```

Path pattern P



```
(a)-[d]->(b)<-[e]-(c)
```

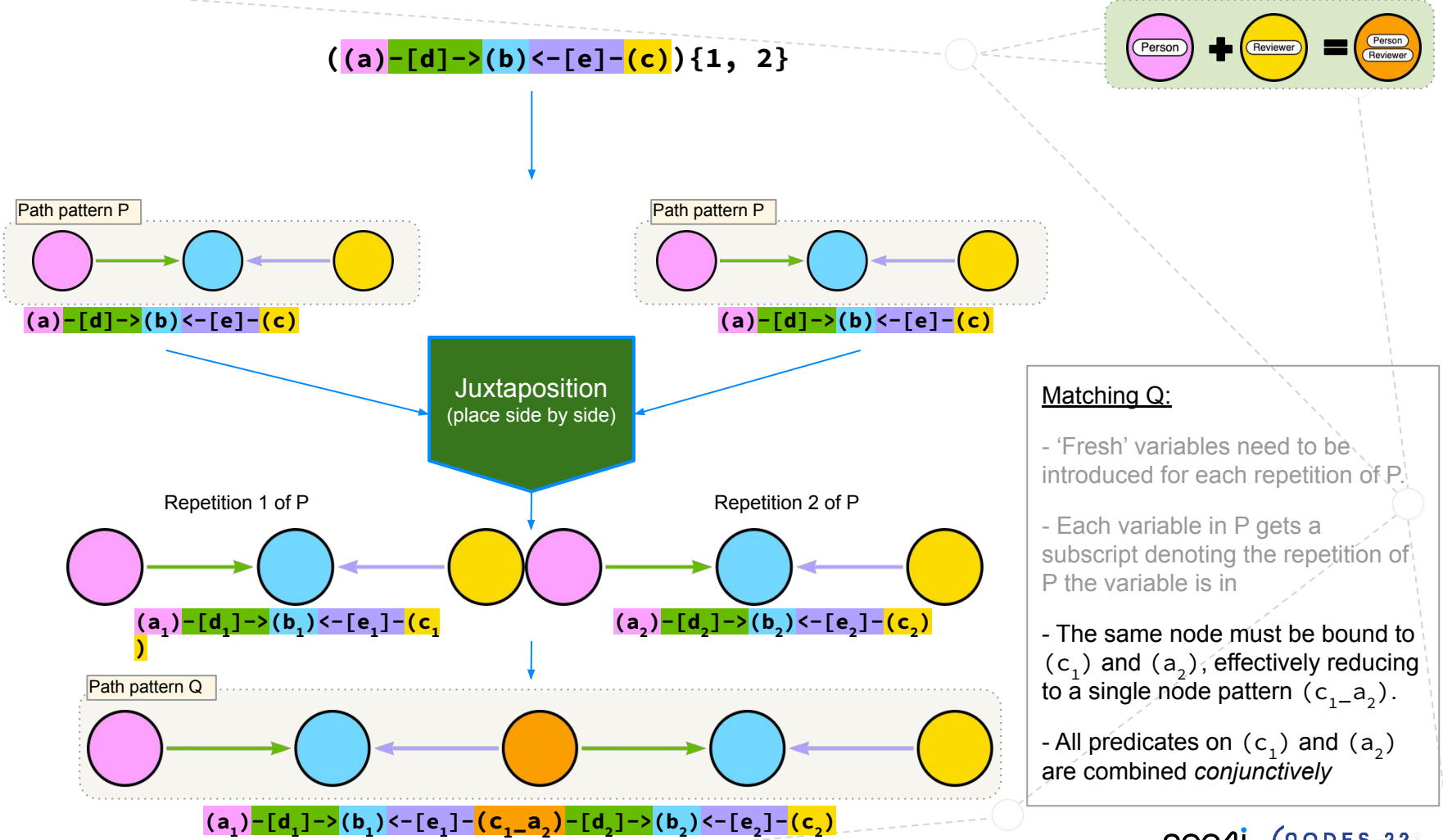




Matching Q:

- 'Fresh' variables need to be introduced for each repetition of P.

- Each variable in P gets a subscript denoting the repetition of P the variable is in



Matching Q:

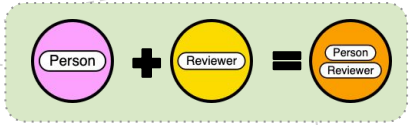
- 'Fresh' variables need to be introduced for each repetition of P.

- Each variable in P gets a subscript denoting the repetition of P the variable is in

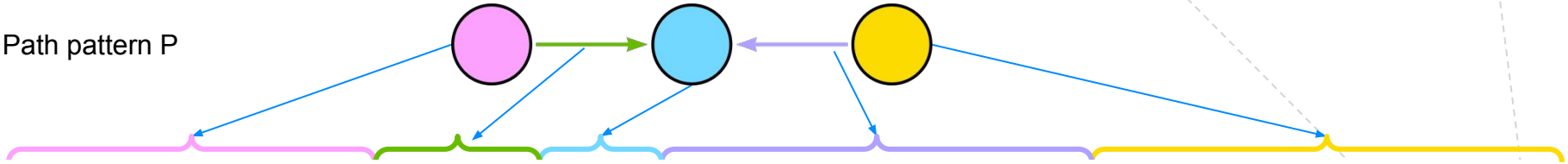
- The same node must be bound to (c_1) and (a_2), effectively reducing to a single node pattern $(c_1{=}a_2)$.

- All predicates on (c_1) and (a_2) are combined *conjunctively*

Repeating a path pattern

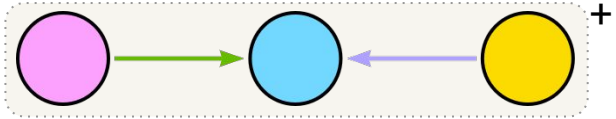


Path pattern P



```
(a:Person WHERE a.age > 30)-[d:LIKES]->(b:Movie)<-[e:RATES WHERE e.rating > 2]-(c:Reviewer WHERE c.city = a.city)
```

Path pattern Q = P+ ⇨



Union operator

P_1

P_1P_2

$P_1P_2P_3$

...(etc)

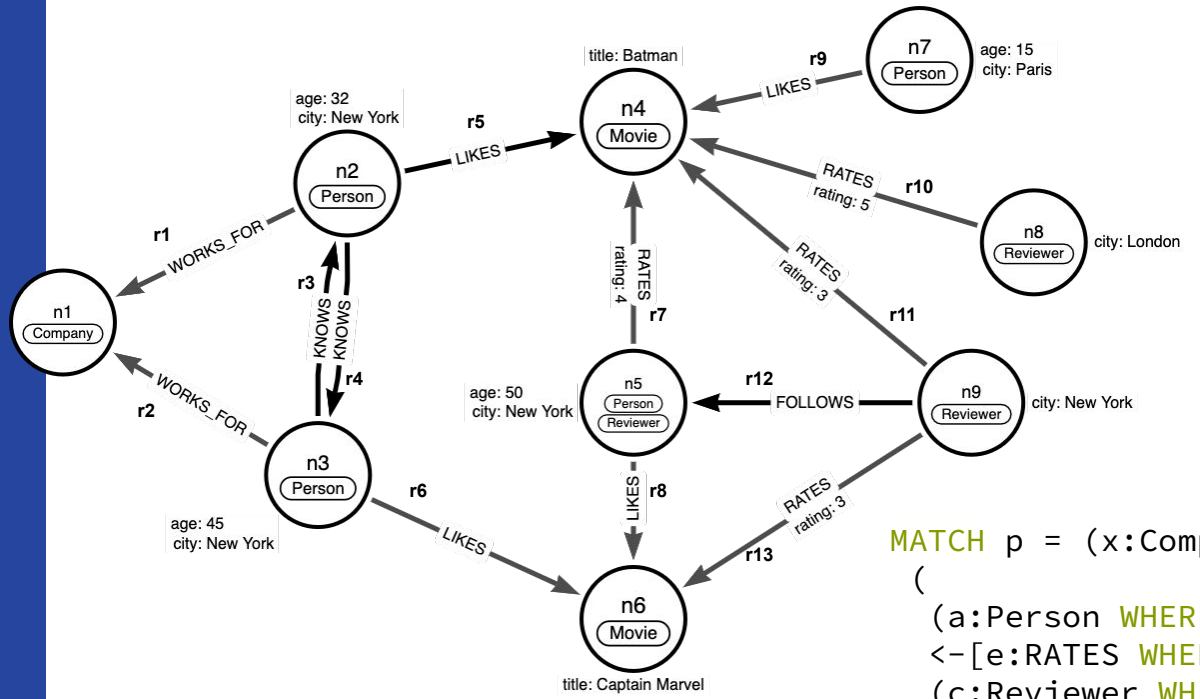
neo4j NODES 22

Constructing longer path patterns

A path pattern can be composed from any number of quantified sub-path patterns and fixed-length sub-path patterns.
Adjacent nodes – where the sub-path patterns connect – are juxtaposed.

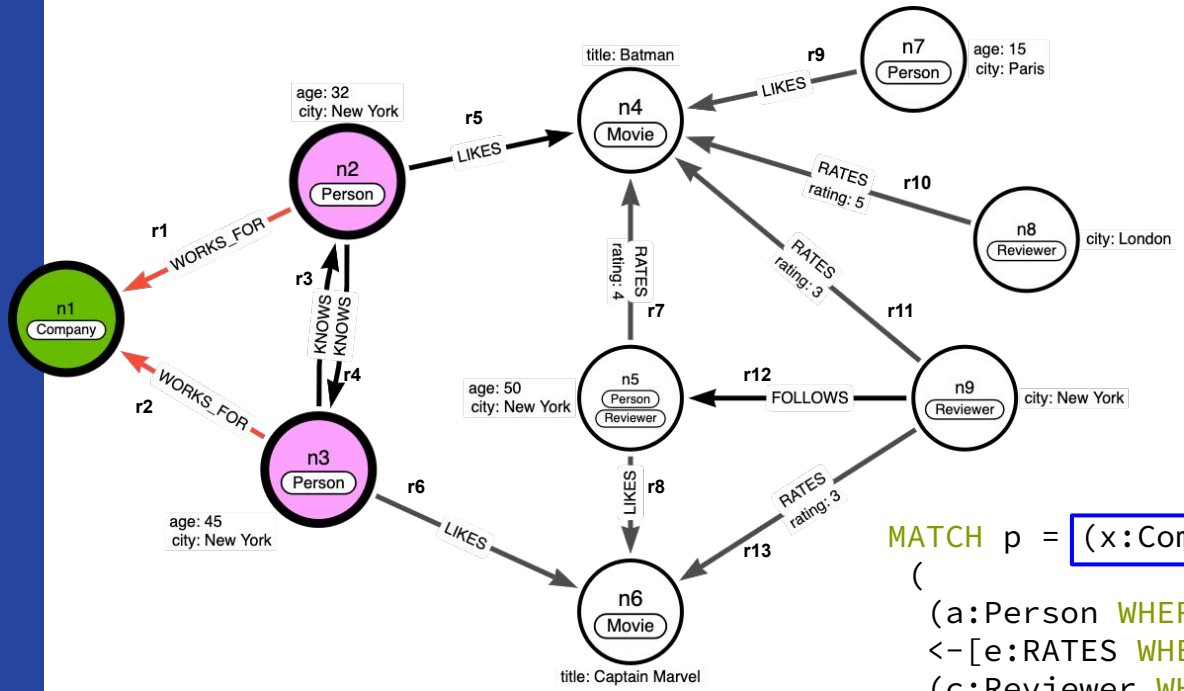
```
MATCH p = (x:Company)<-[y:WORKS_FOR]-(z)((a:Person WHERE a.age > 30)
            -[d:LIKES]->(b:Movie)
            <-[e:RATES WHERE e.rating > 2]-
            (c:Reviewer WHERE c.city = a.city)
          )+
```

Example



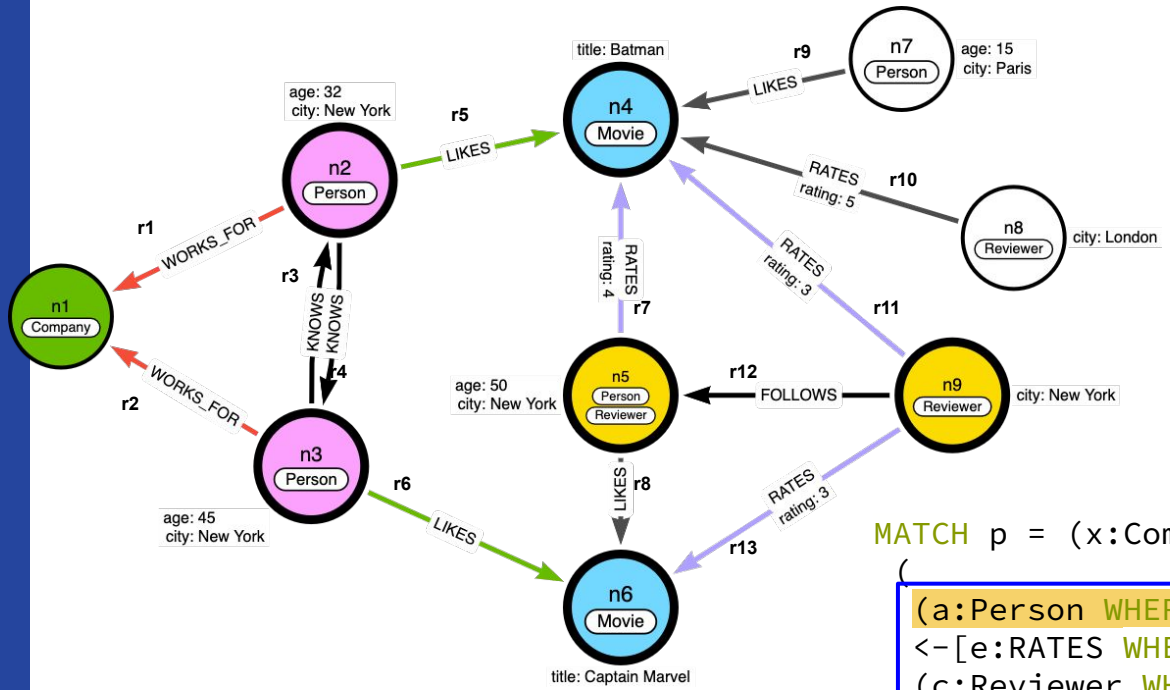

```
MATCH p = (x:Company)<-[y:WORKS_FOR]-(z)
 (
  (a:Person WHERE a.age > 30)-[d:LIKES]->(b:Movie)
  <-[e:RATES WHERE e.rating > 2]-
  (c:Reviewer WHERE c.city = a.city)
  )+
```

Example




```
MATCH p = (x:Company)<-[y:WORKS_FOR]-(z)
 (
  (a:Person WHERE a.age > 30)-[d:LIKES]->(b:Movie)
  <-[e:RATES WHERE e.rating > 2]-
  (c:Reviewer WHERE c.city = a.city)
  )+
```

Example

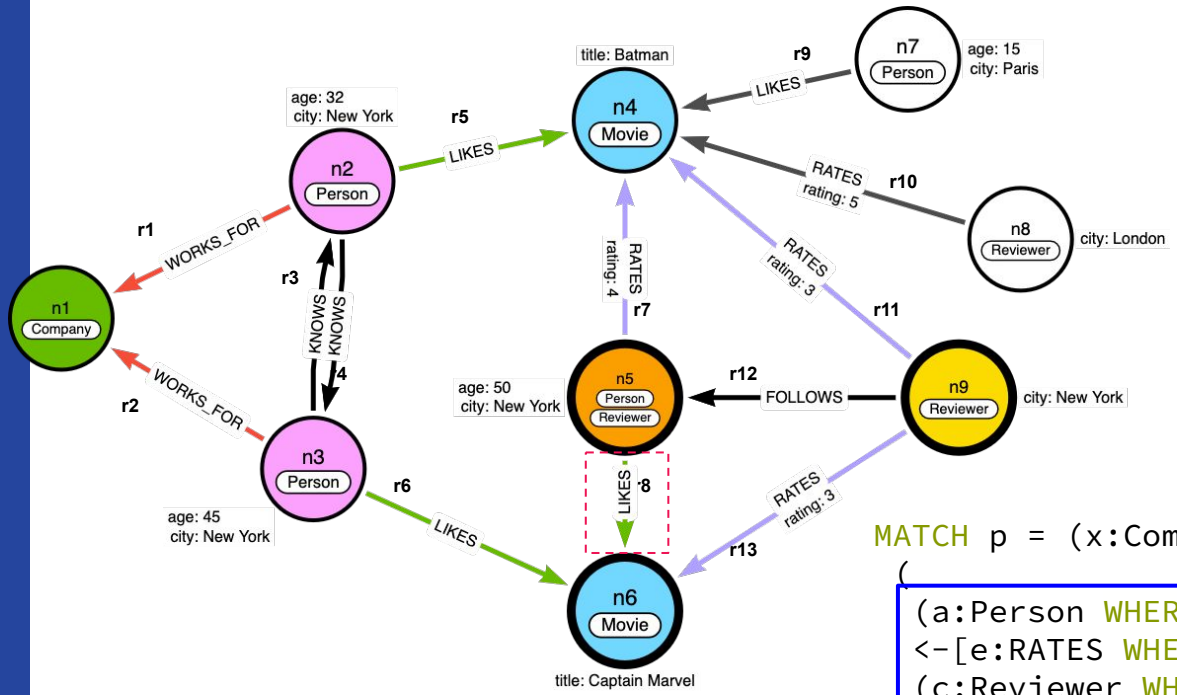


The node patterns **`(z)`** and **`(a:Person)`** are juxtaposed here, and so are "merged": the same node must be bound to `z` and `a`, and all predicates are combined conjunctively

P_1

```
MATCH p = (x:Company)<-[y:WORKS_FOR]-(z)
 (
  (a:Person WHERE a.age > 30)-[d:LIKES]->(b:Movie)
  <-[e:RATES WHERE e.rating > 2]-
  (c:Reviewer WHERE c.city = a.city)
 )+
```

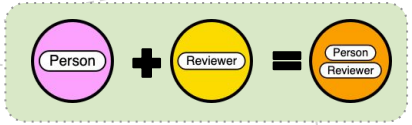
Example



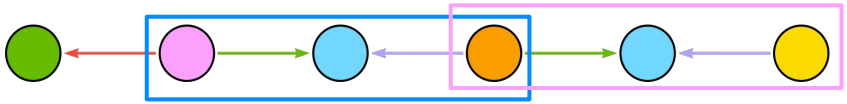
P_2

```
MATCH p = (x:Company)<-[y:WORKS_FOR]-(z)
 (
  (a:Person WHERE a.age > 30)-[d:LIKES]->(b:Movie)
  <-[e:RATES WHERE e.rating > 2]-
  (c:Reviewer WHERE c.city = a.city)
 )+
```

Solutions



p	(x)	<-y-	(z)	(a)	-d->	(b)	<-e-	(c)
[n1, r1, n2, r5, n4, r11, n9]	n1	r1	n2	[n2]	[r5]	[n4]	[r11]	[n9]
[n1, r1, n2, r5, n4, r7, n5]	n1	r1	n2	[n2]	[r5]	[n4]	[r7]	[n5]
[n1, r2, n3, r6, n6, r13, n9]	n1	r2	n3	[n3]	[r6]	[n6]	[r13]	[n9]



p	(x)	<-y-	(z)	(a)	-d->	(b)	<-e-	(c)
[n1, r1, n2, r5, n4, r7, n5, r8, n6, r13, n9]	n1	r1	n2	[**n2**, **n5**]	[r5, r8]	[n4, n6]	[r7, r13]	[**n5**, **n9**]

To conclude…

- Limitations of variable-length patterns in Cypher today
- WHERE clause in node and relationship patterns
- New operators in node label and relationship type expressions
- Quantified path patterns

This is just the beginning - there is plenty more to come!