Query languages for property graphs

From RPQs to Cypher

NoSQL and New SQL course

M2 LID, Université Gustave-Eiffel

2025-2026

version 6.1

Introduction

About this PDF

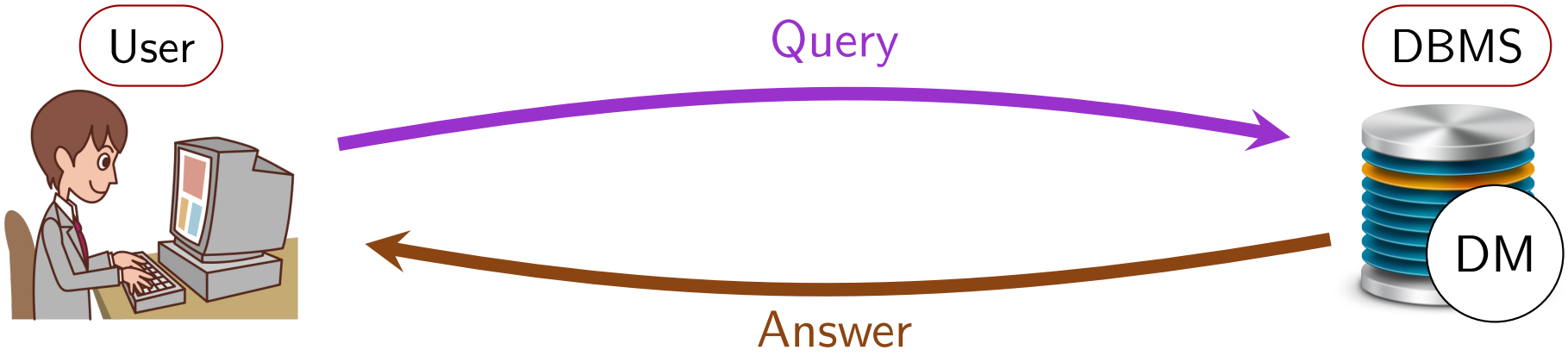
Navigation

From any frame, the page number is a link to the navigable outline.

Term translations

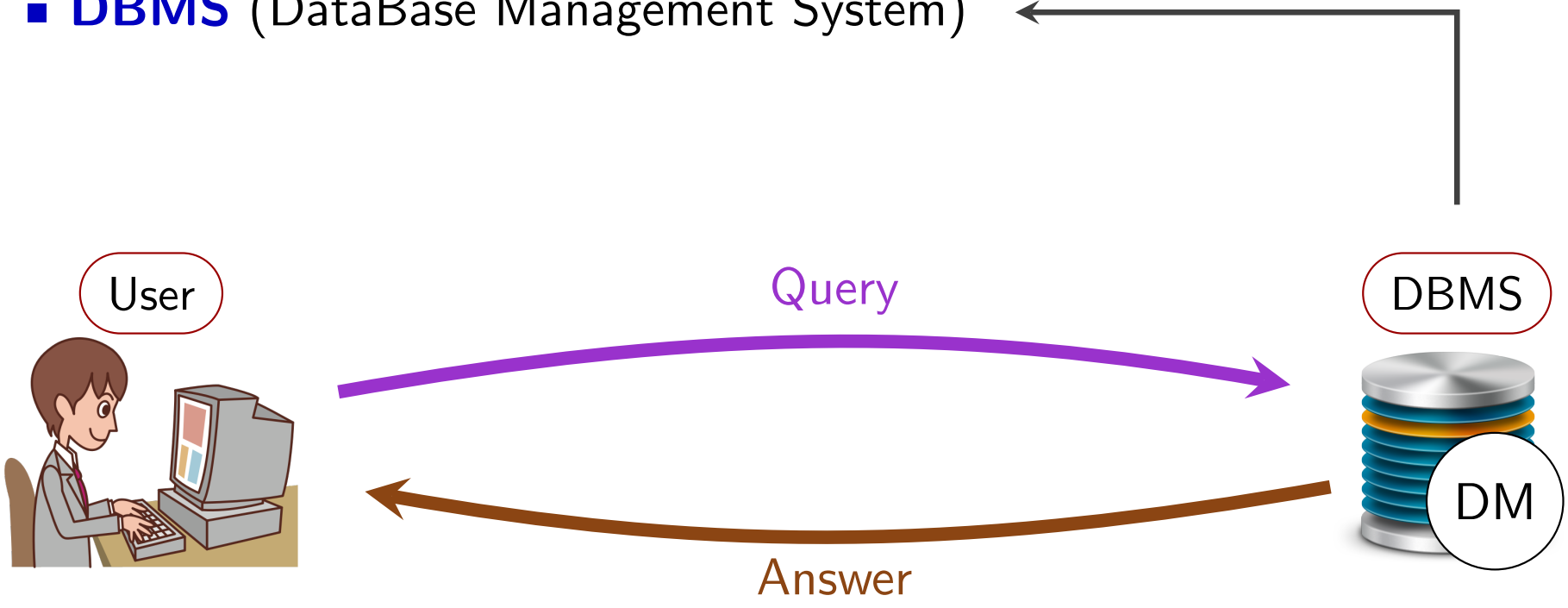
There is a French/English lexicon at the end.

Overview of query answering



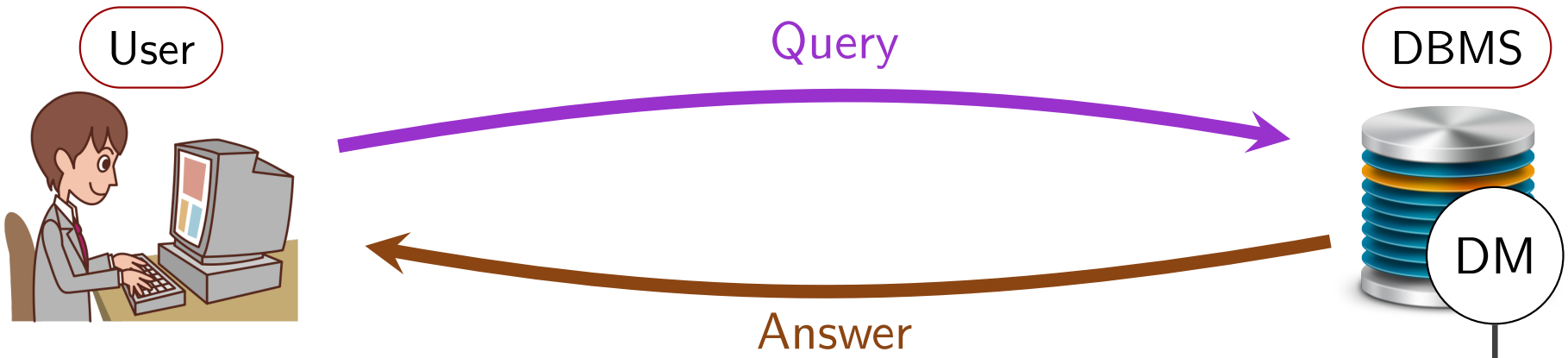
Overview of query answering

- **DBMS** (DataBase Management System)



Overview of query answering

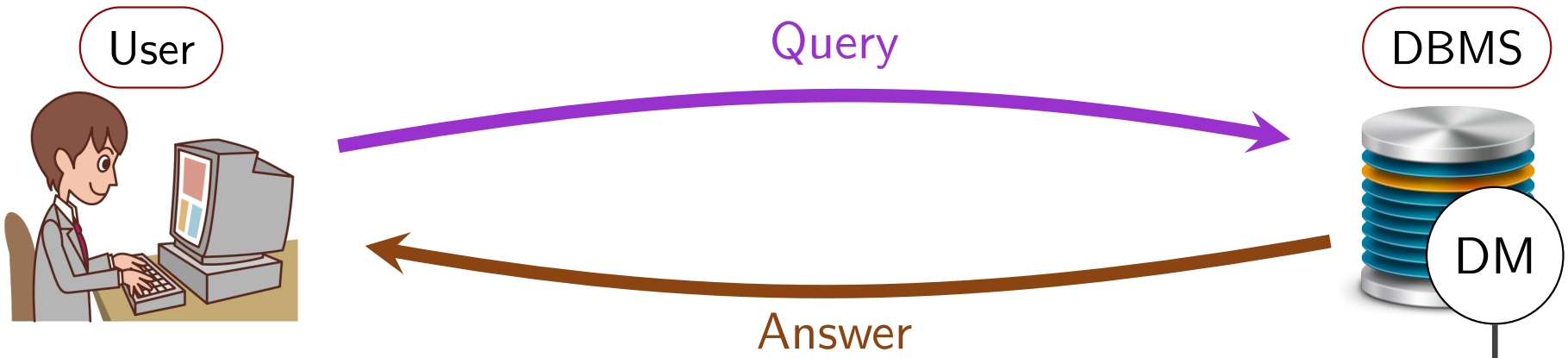
- **DBMS** (DataBase Management System)



- **DM** (**Data Model**)
  - "*How is the data structured?*" "*What data is representable?*"
  - Ex: Relations (SQL), Trees (XML, JSON), Graphs (PGs, RDF),

Overview of query answering

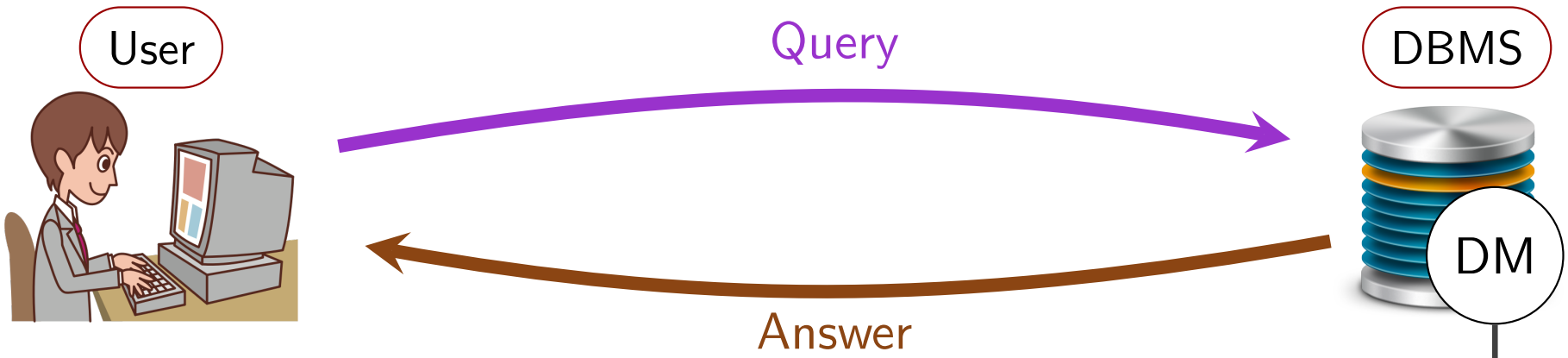
- **DBMS** (DataBase Management System)
- **Query language**
  - "*What can the user write?*"



- **DM** (**Data Model**)
  - "*How is the data structured?*" "*What data is representable?*"
  - Ex: Relations (SQL), Trees (XML, JSON), Graphs (PGs, RDF),

Overview of query answering

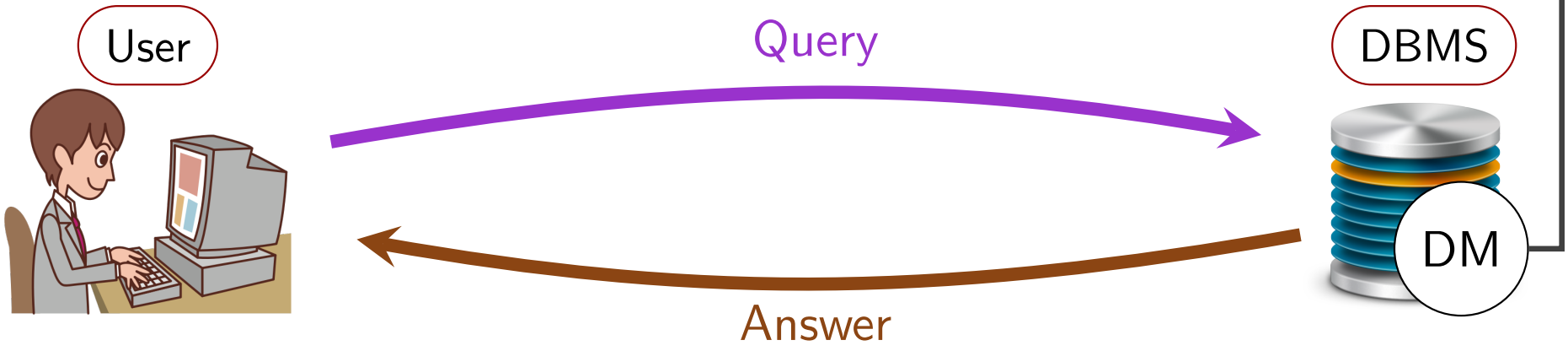
- **DBMS** (DataBase Management System)
- **Query language**
  - "*What can the user write?*"



- **Semantics**
  - "*What does the query mean?*" "*What is the correct answer?*"
  - Ex: Set semantics (duplicate elimination)
- **DM** (**Data Model**)
  - "*How is the data structured?*" "*What data is representable?*"
  - Ex: Relations (SQL), Trees (XML, JSON), Graphs (PGs, RDF),

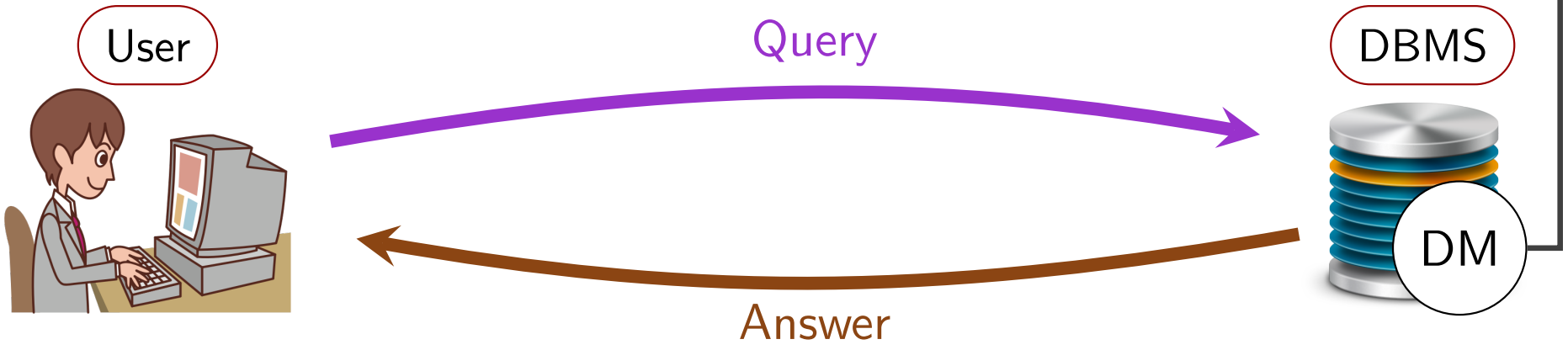
This segment is about **query languages for property graphs**

- The **data model** is **Property Graph (PG)**



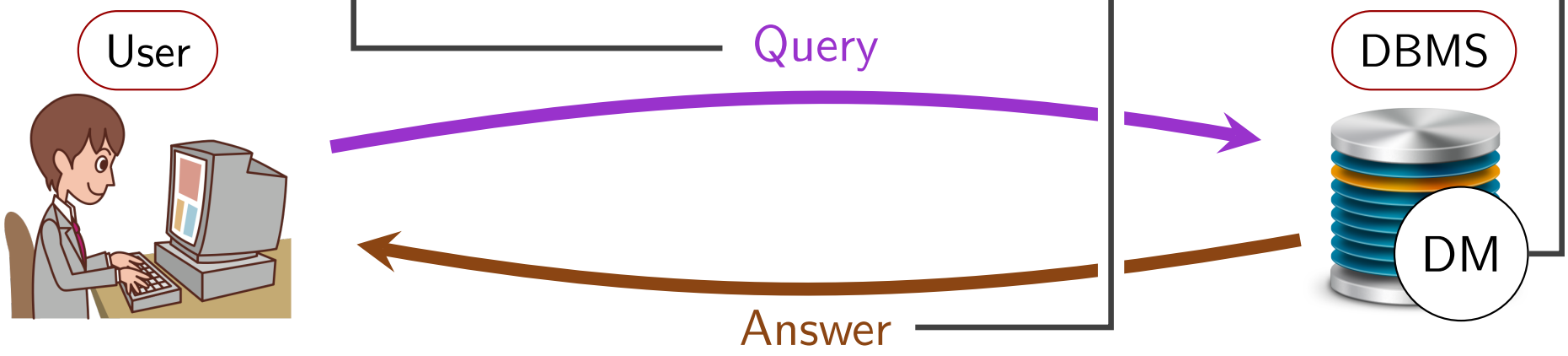
This segment is about **query languages for property graphs**

- The **data model** is **Property Graph (PG)**
- The **DBMS** we will use (**Neo4j**) implements this DM



This segment is about **query languages for property graphs**

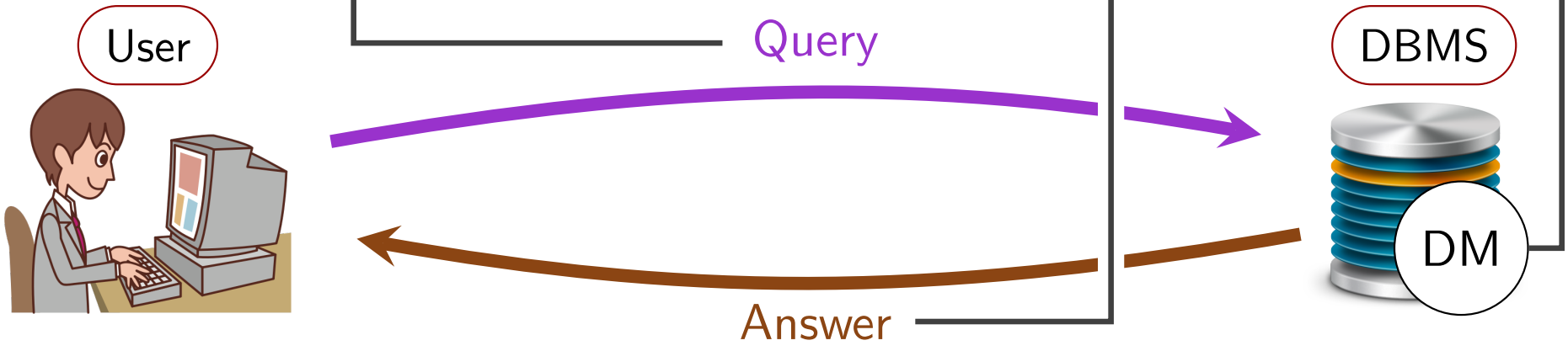
- The **data model** is **Property Graph (PG)**
- The **DBMS** we will use (**Neo4j**) implements this DM
- The **query languages** we consider (**Cypher**, GQL, etc.) need this DM to have **any meaning**



This segment is about **query languages for property graphs**

In part **II**:

- The **data model** is **Property Graph (PG)**
- The **DBMS** we will use (**Neo4j**) implements this DM
- The **query languages** we consider (**Cypher**, GQL, etc.) need this DM to have **any meaning**



Popularity of Graph DBMS's (1)

Vast majority of DMBS's are relational, not graph

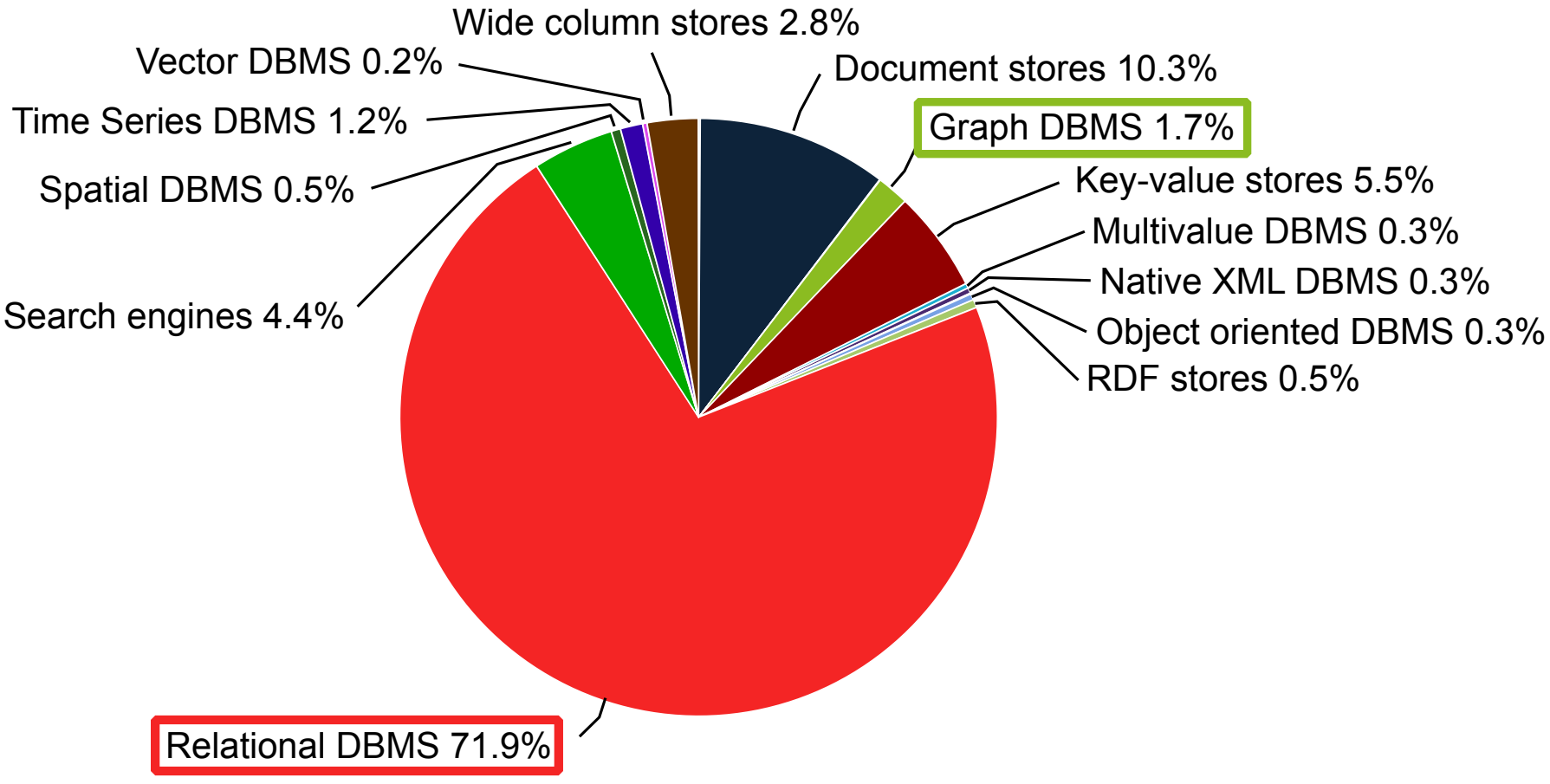


Figure and data from `db-engines.com`, August 2023

Popularity of Graph DBMS's (2)

Graph DBMS's has grown in popularity for ten years

Relational DBMS's continued their slow decline

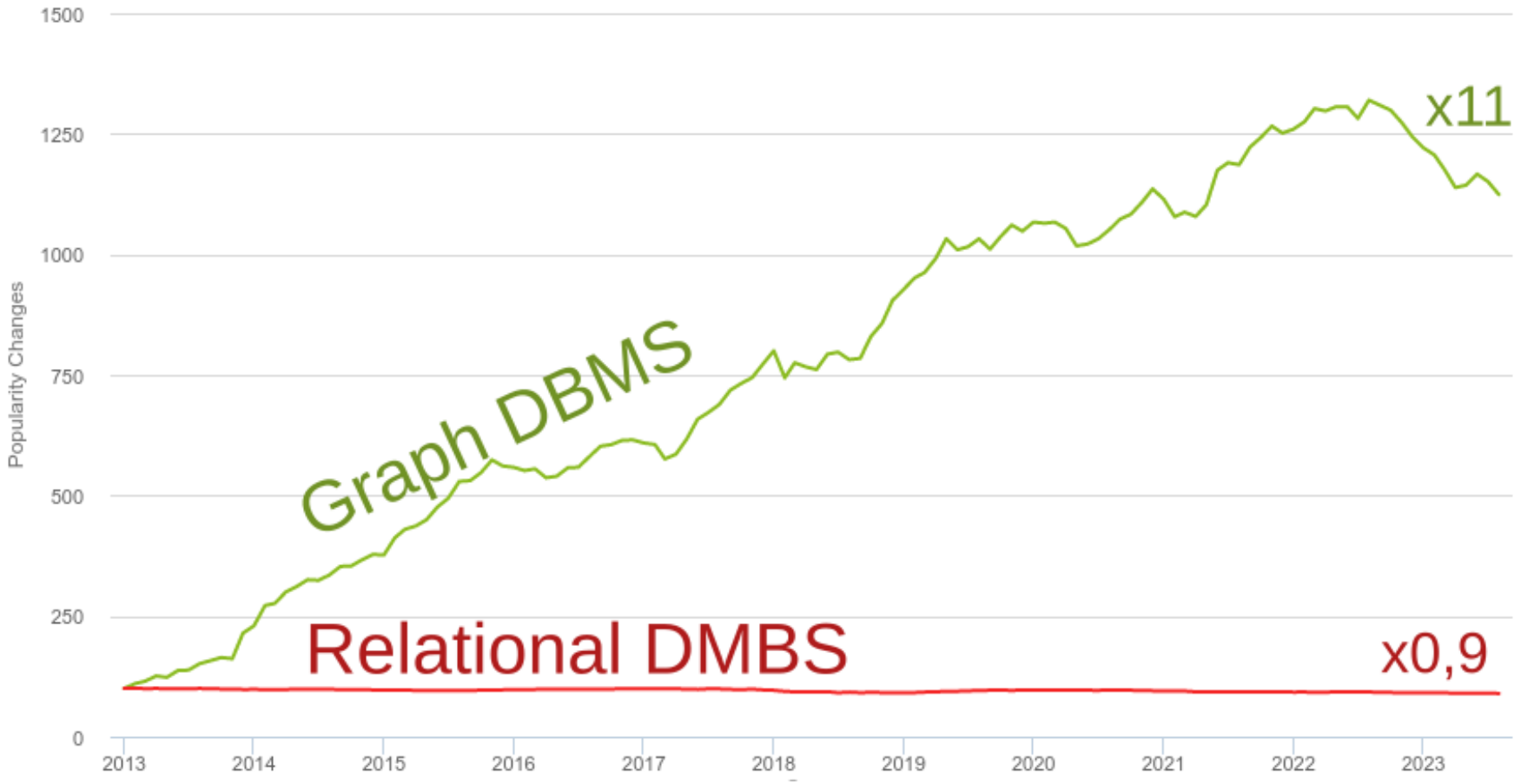
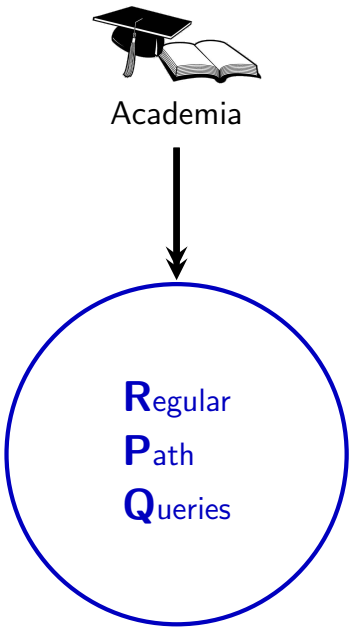


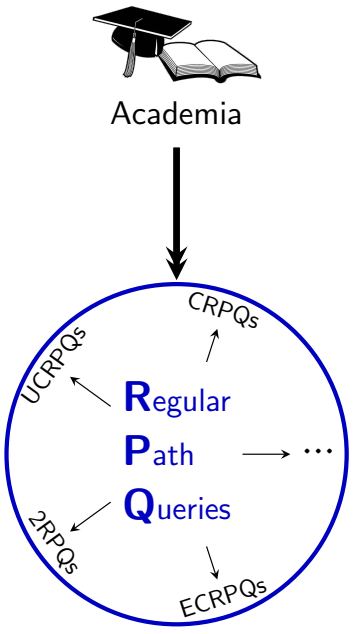
Figure and data from `db-engines.com`, August 2023

A bit of history



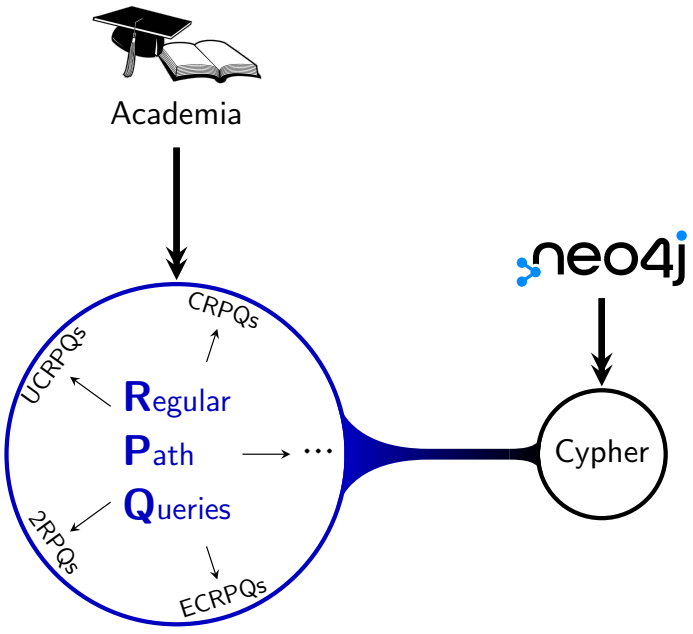
Late 1980's – RPQs are invented

A bit of history



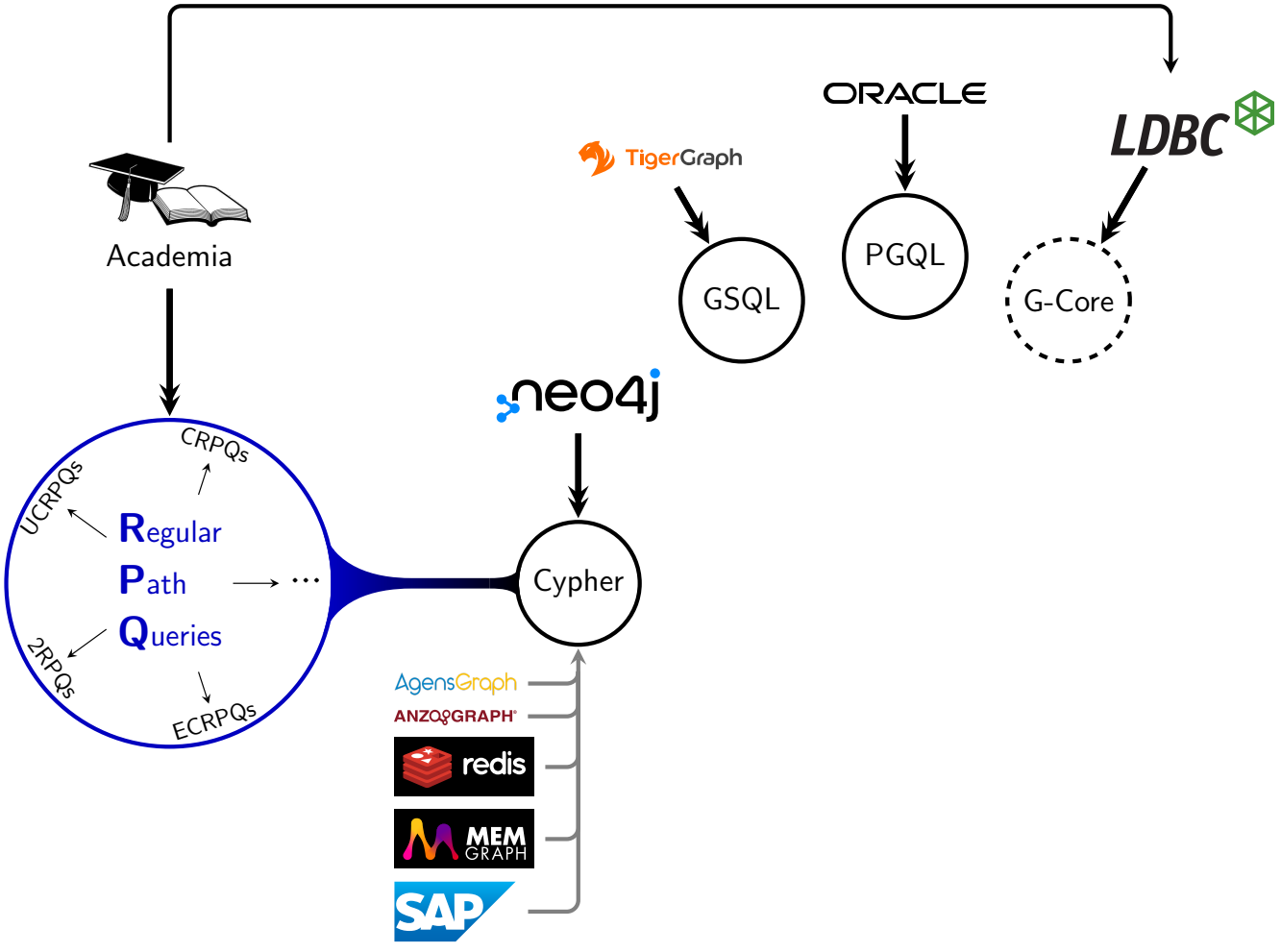
Since 1990's – RPQs are studied and extended in academia

A bit of history



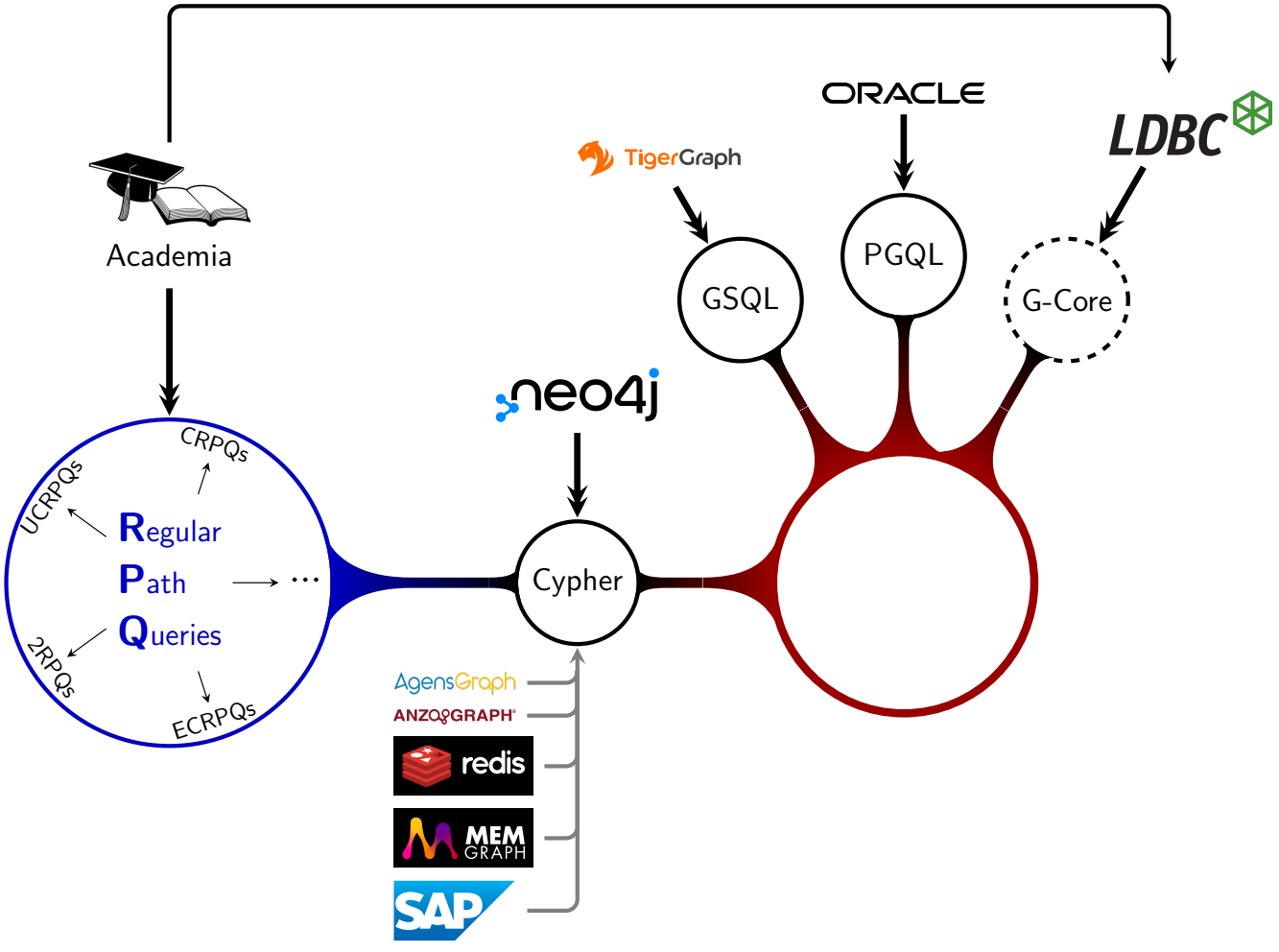
2011 – The query language Cypher is released with the DBMS Neo4j

A bit of history



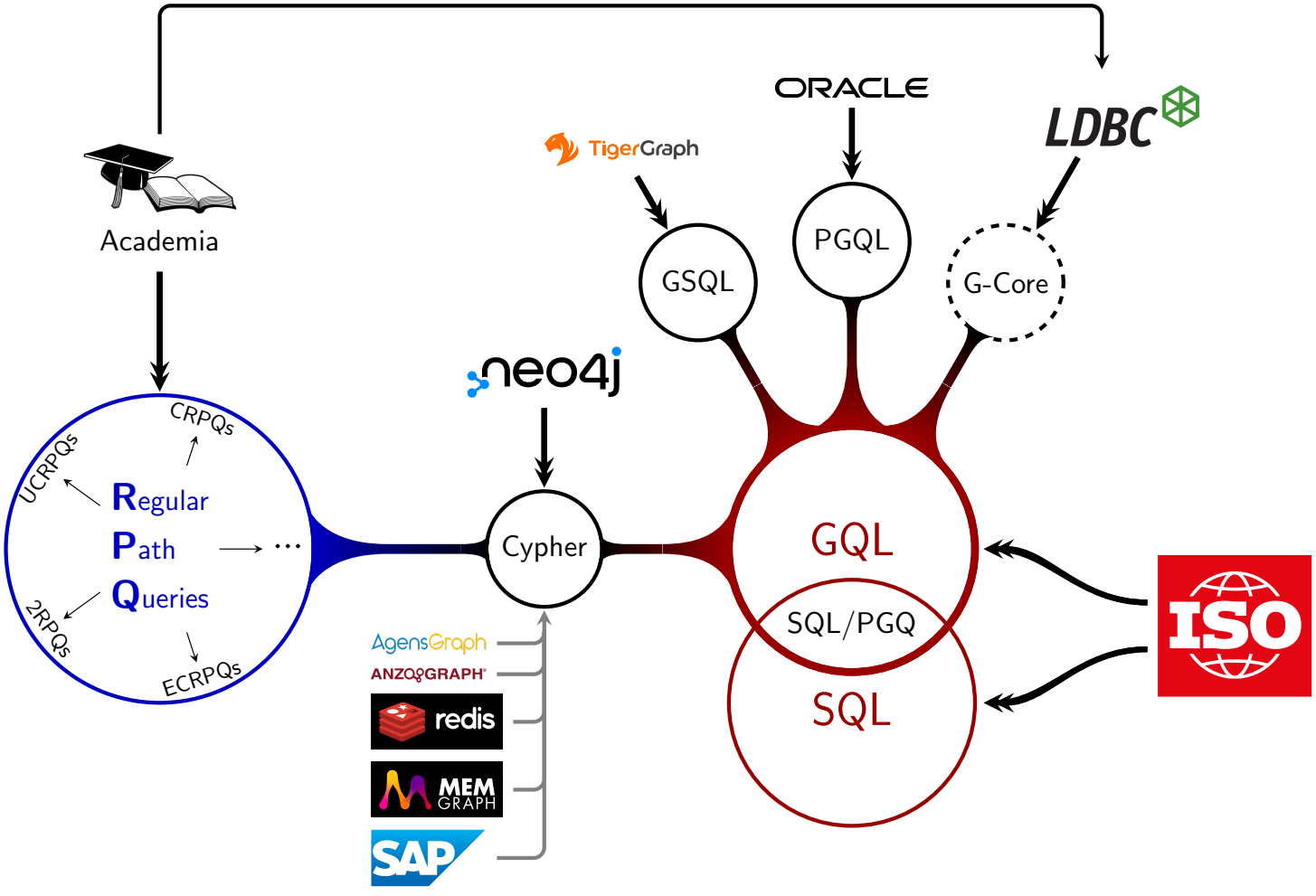
Mid 2010's – Cypher is successful and new graph DBMS's appear. Some use Cypher, some come with their own query language.

A bit of history



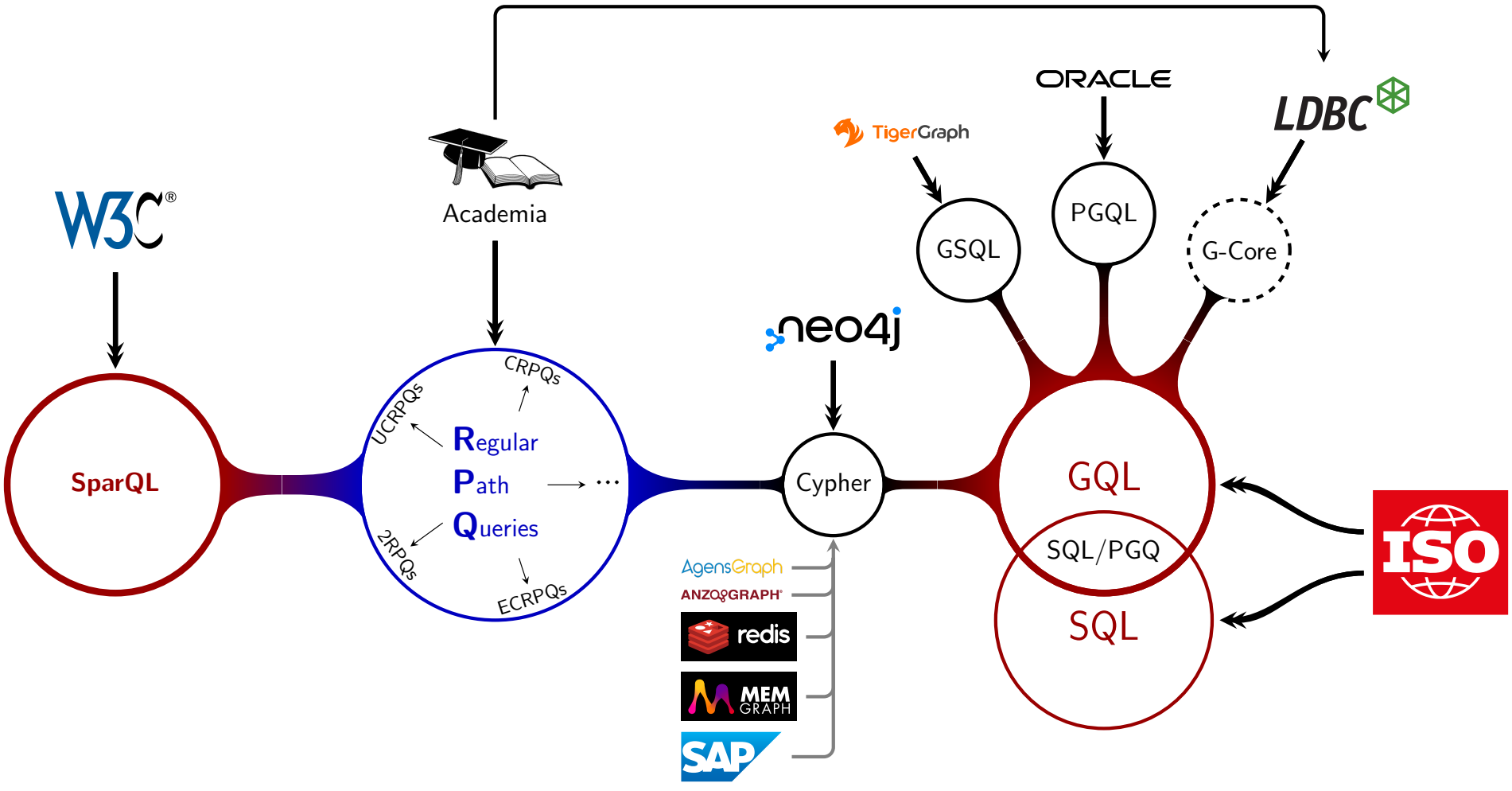
Late 2010's – Idea to merge existing languages for interoperability

A bit of history



2023 – SQL/PGQ support for querying PG's in SQL
2024 – GQL, standard query language for PG's

A bit of history



Side note: In SPARQL, the standard language for the RDF DM, features *Property paths* which are also based on RPQ's.

Outline

Course I: Theoretical Foundations

- Data model: Labeled Graphs
- Query language: RPQs

Course II & III: A practical application

- Data model: Property graphs
- Query language: Cypher

Part I: Theoretical foundations

Part I: Theoretical foundations

1. Data model: labeled graphs

Our data model : (Labeled) graphs (1)

Example

A graph consists of ...

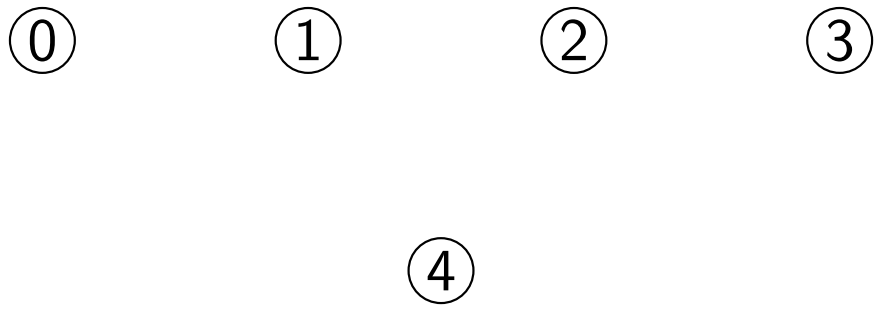
- Vertices
- Edges
- Edge labels

Our data model : (Labeled) graphs (1)

Example

A graph consists of ...

- Vertices
- Edges
- Edge labels

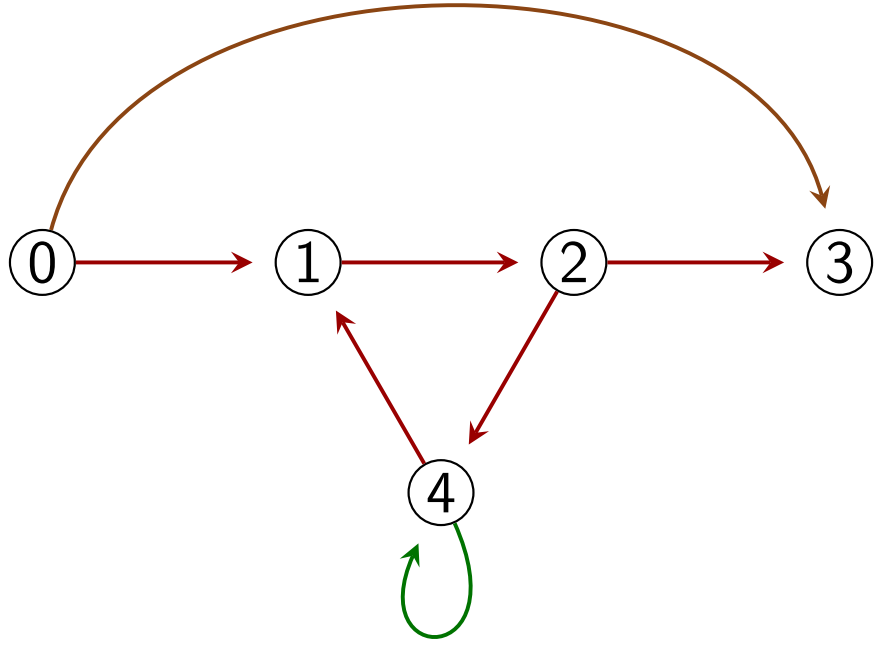


Our data model : (Labeled) graphs (1)

Example

A graph consists of ...

- Vertices
- Edges
- Edge labels

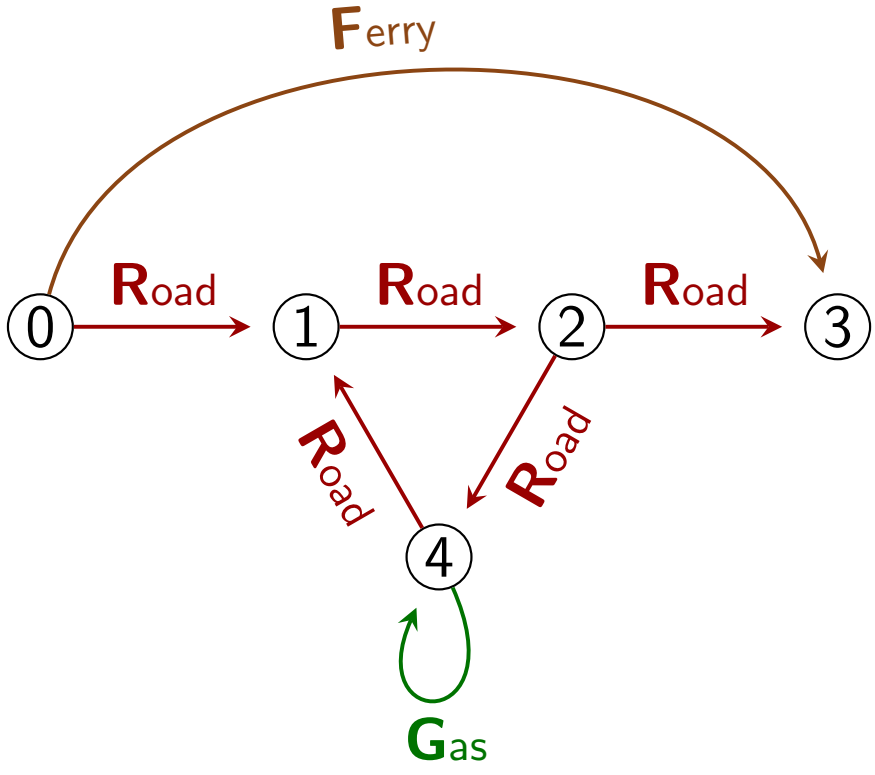


Our data model : (Labeled) graphs (1)

Example

A graph consists of ...

- Vertices
- Edges
- Edge labels



Our data model : (Labeled) graphs (2)

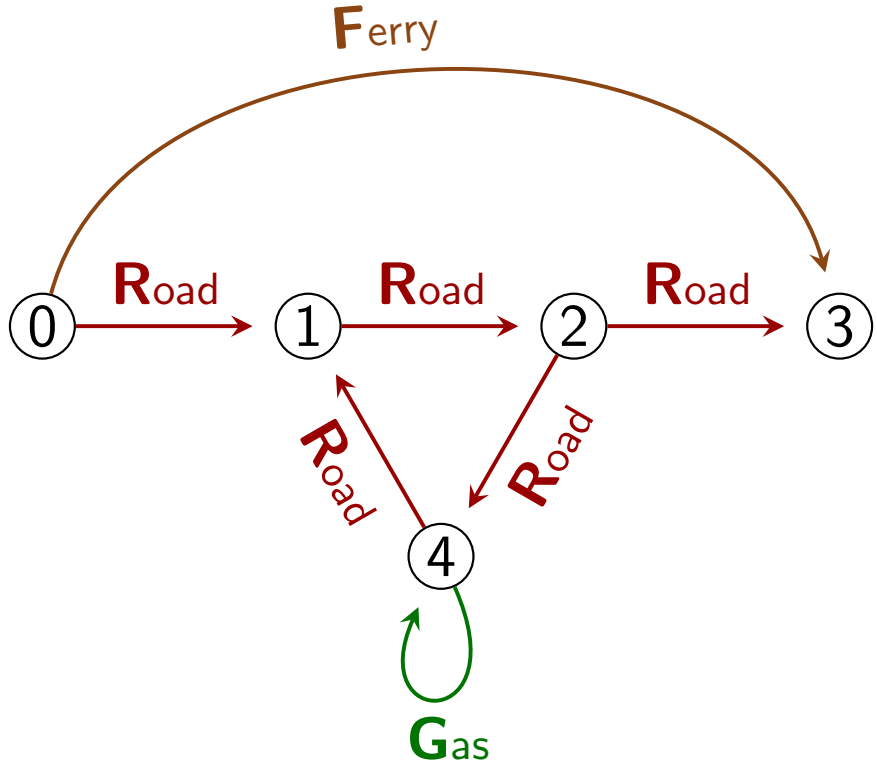
Definition

A labeled graph consists of:

- V is a finite set of **vertices**
- L is a finite set of **labels**
- $E \subseteq V \times L \times V$ is a finite set of **edges**

Formal representation of G

- $V = \{0, 1, 2, 3, 4\}$
- $L = \{\mathbf{R}, \mathbf{F}, \mathbf{G}\}$
- $E = \{ (0, \mathbf{R}, 1), (1, \mathbf{R}, 2), (2, \mathbf{R}, 3), (2, \mathbf{R}, 4), (4, \mathbf{R}, 1), (0, \mathbf{F}, 3), (4, \mathbf{G}, 4) \}$



Example graph G

Our data model : (Labeled) graphs (2)

Formalisation

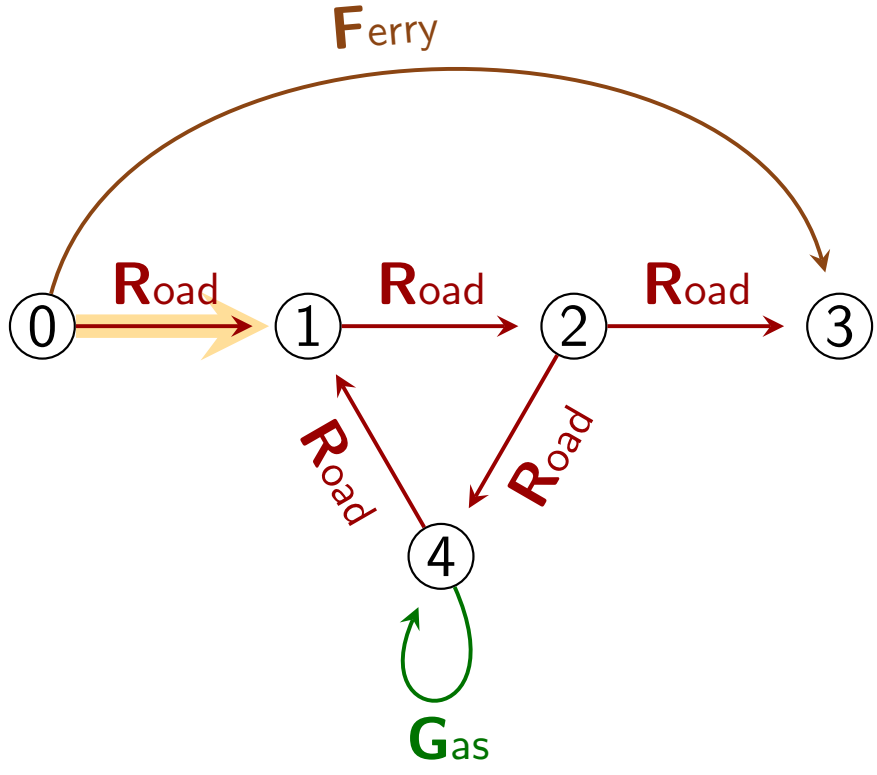
Definition

A labeled graph consists of:

- V is a finite set of **vertices**
- L is a finite set of **labels**
- $E \subseteq V \times L \times V$ is a finite set of **edges**

Formal representation of G

- $V = \{0, 1, 2, 3, 4\}$
- $L = \{\mathbf{R}, \mathbf{F}, \mathbf{G}\}$
- $E = \{ (0, \mathbf{R}, 1), (1, \mathbf{R}, 2), (2, \mathbf{R}, 3), (2, \mathbf{R}, 4), (4, \mathbf{R}, 1), (0, \mathbf{F}, 3), (4, \mathbf{G}, 4) \}$



Example graph G

Our data model : (Labeled) graphs (2)

Formalisation

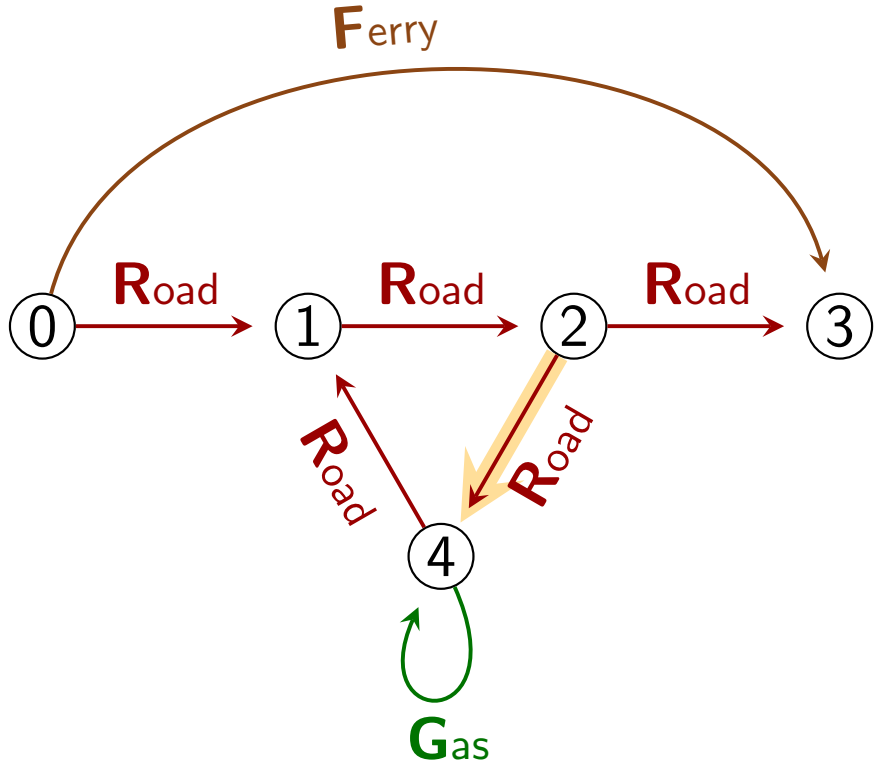
Definition

A labeled graph consists of:

- V is a finite set of **vertices**
- L is a finite set of **labels**
- $E \subseteq V \times L \times V$ is a finite set of **edges**

Formal representation of G

- $V = \{0, 1, 2, 3, 4\}$
- $L = \{\mathbf{R}, \mathbf{F}, \mathbf{G}\}$
- $E = \{ (0, \mathbf{R}, 1), (1, \mathbf{R}, 2), (2, \mathbf{R}, 3), (2, \mathbf{R}, 4), (4, \mathbf{R}, 1), (0, \mathbf{F}, 3), (4, \mathbf{G}, 4) \}$



Example graph G

Our data model : (Labeled) graphs (2)

Formalisation

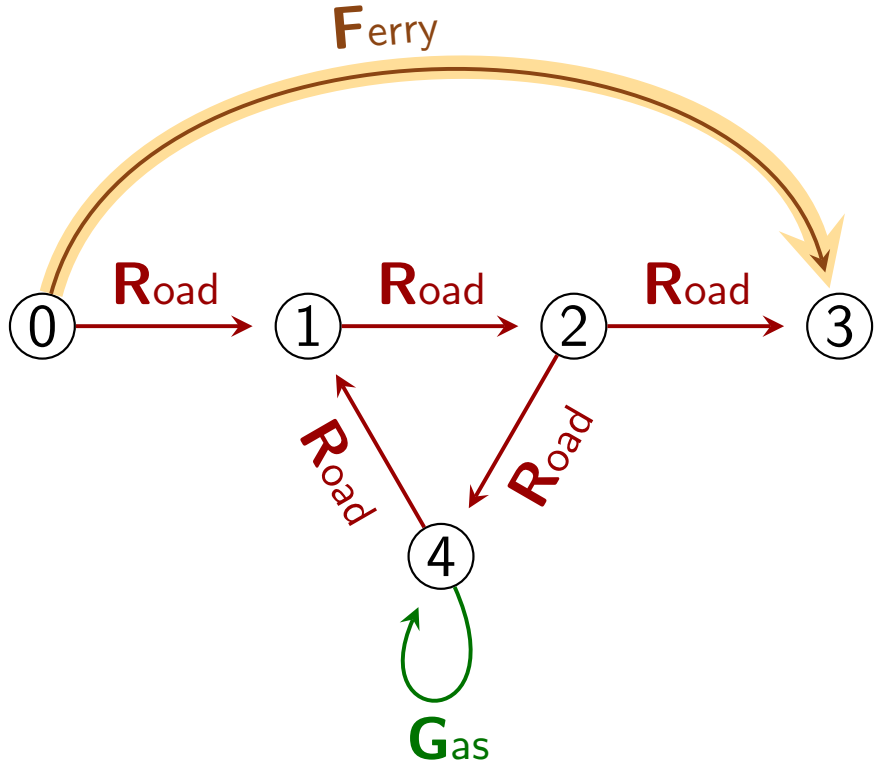
Definition

A labeled graph consists of:

- V is a finite set of **vertices**
- L is a finite set of **labels**
- $E \subseteq V \times L \times V$ is a finite set of **edges**

Formal representation of G

- $V = \{0, 1, 2, 3, 4\}$
- $L = \{\mathbf{R}, \mathbf{F}, \mathbf{G}\}$
- $E = \{ (0, \mathbf{R}, 1), (1, \mathbf{R}, 2), (2, \mathbf{R}, 3), (2, \mathbf{R}, 4), (4, \mathbf{R}, 1), (0, \mathbf{F}, 3), (4, \mathbf{G}, 4) \}$



Example graph G

Our data model : (Labeled) graphs (2)

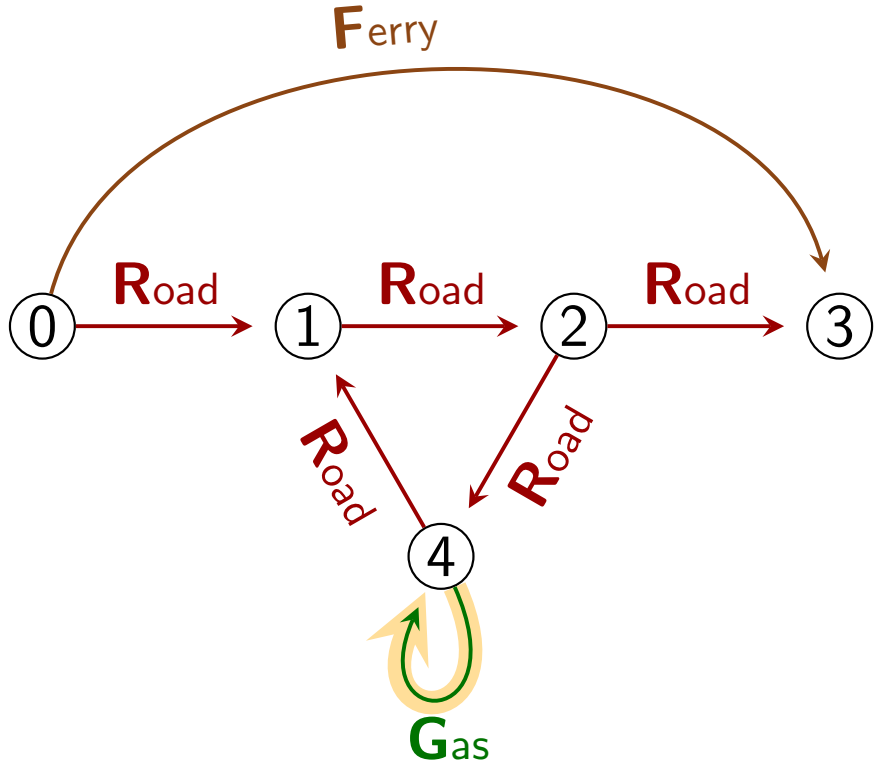
Definition

A labeled graph consists of:

- V is a finite set of **vertices**
- L is a finite set of **labels**
- $E \subseteq V \times L \times V$ is a finite set of **edges**

Formal representation of G

- $V = \{0, 1, 2, 3, 4\}$
- $L = \{\mathbf{R}, \mathbf{F}, \mathbf{G}\}$
- $E = \{ (0, \mathbf{R}, 1), (1, \mathbf{R}, 2), (2, \mathbf{R}, 3), (2, \mathbf{R}, 4), (4, \mathbf{R}, 1), (0, \mathbf{F}, 3), (4, \mathbf{G}, 4) \}$

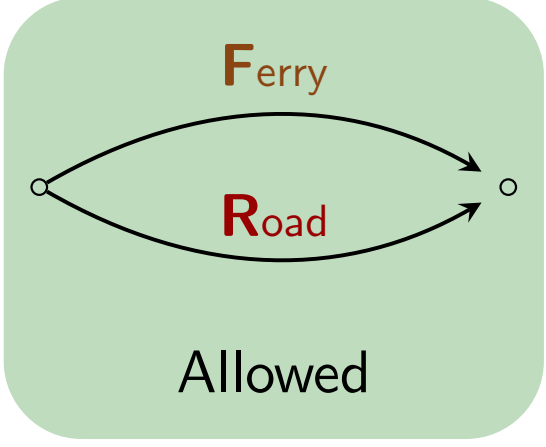


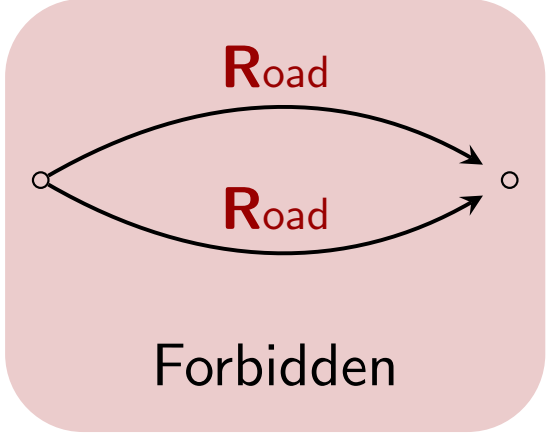
Example graph G

Limits to the graph data model (1)

Our graphs are single-labeled and single-edge

- Each edge has exactly one label.
- There cannot be two identical edges.





Limits to the graph data model (2)

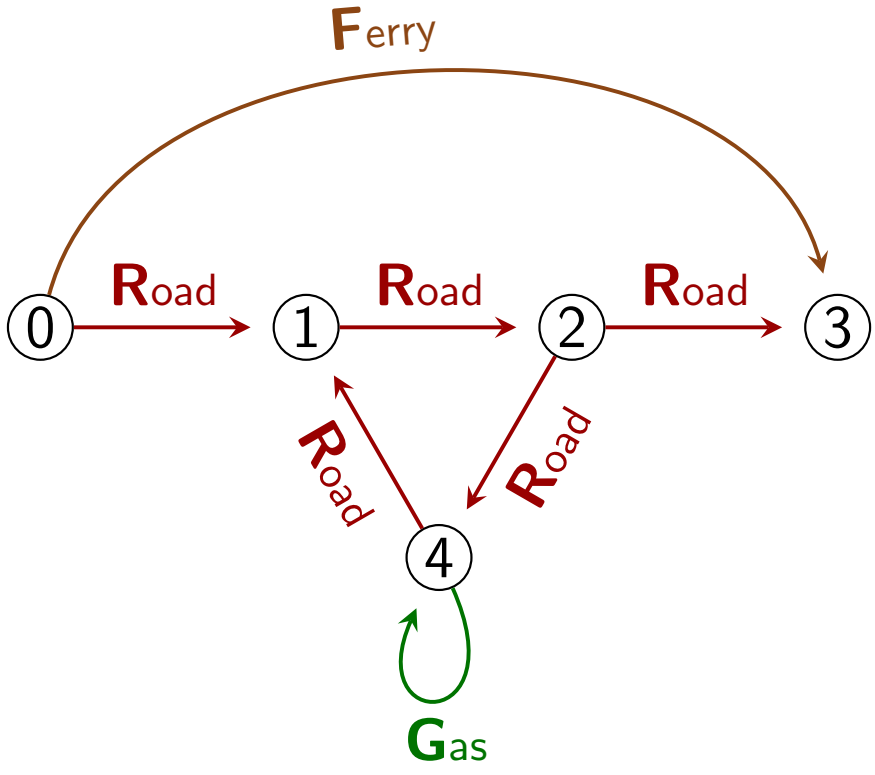
The graph DM is about topology, not data

- We encode the existence of entities and of relations between entities
  Ex: cities, roads
- We don't encode specific data of an entity or relation
  Ex: names, distances

Examples

Our model **cannot** encode that

- some road from is 2km long
- the gas price is 2€ in some vertex



Part I: Theoretical foundations

2. Regular Path Queries

What is an RPQ?

Regular **P**ath **Q**ueries offer a pattern-matching mechanism for walks.

An RPQ

- is a regular expression
- sent to a (labeled) graph
- to match walks.

Terminology reminders

A **letter** is a symbol coming from a finite set, the **alphabet**.

In our case, the alphabet is the label-set of the graph.

Examples:

- $\{\mathbf{R}, \mathbf{F}, \mathbf{G}\}$ is an alphabet
- **R** and **G** are letters

Terminology reminders

A **letter** is a symbol coming from a finite set, the **alphabet**.

In our case, the alphabet is the label-set of the graph.

Examples:

- $\{\mathbf{R}, \mathbf{F}, \mathbf{G}\}$ is an alphabet
- **R** and **G** are letters

A **word** is a finite sequence of letters

Examples words:

- **RGRR**
- **R**
- ε, the **empty word**

Terminology reminders

A **letter** is a symbol coming from a finite set, the **alphabet**.

In our case, the alphabet is the label-set of the graph.

Examples:

- $\{\mathbf{R}, \mathbf{F}, \mathbf{G}\}$ is an alphabet
- **R** and **G** are letters

A **word** is a finite sequence of letters

Examples words:

- **RGRR**
- **R**
- ε, the **empty word**

A **language** is a finite or infinite set of words

Example languages:

1. $\{\mathbf{R}, \mathbf{RG}\}$
2. $\{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \ldots\}$
3. Words with one **G**
4. Words of even length
5. Palindromes (e.g. kayak)
6. Words with more **R** than **G**

Terminology reminders

A **letter** is a symbol coming from a finite set, the **alphabet**.

In our case, the alphabet is the label-set of the graph.

Examples:

- {**R**, **F**, **G**} is an alphabet
- **R** and **G** are letters

A **word** is a finite sequence of letters

Examples words:

- **RGRR**
- **R**
- ε, the **empty word**

A **language** is a finite or infinite set of words

Example languages:

1. {**R**, **RG**}
2. {**R**, **RR**, **RRR**, ...}
3. Words with one **G**
4. Words of even length
5. Palindromes (e.g. kayak)
6. Words with more **R** than **G**

A language is **regular** if it is

- accepted by an automaton; or
- denoted by a regexp.

(Both are equivalent.)

Example:

Terminology reminders

A **letter** is a symbol coming from a finite set, the **alphabet**.

In our case, the alphabet is the label-set of the graph.

Examples:

- {**R**, **F**, **G**} is an alphabet
- **R** and **G** are letters

A **word** is a finite sequence of letters

Examples words:

- **RGRR**
- **R**
- ε, the **empty word**

A **language** is a finite or infinite set of words

Example languages:

1. {**R**, **RG**}
2. {**R**, **RR**, **RRR**, . . .}
3. Words with one **G**
4. Words of even length
5. Palindromes (e.g. kayak)
6. Words with more **R** than **G**

A language is **regular** if it is

- accepted by an automaton; or
- denoted by a regexp.

(Both are equivalent.)

Example: 1–4 are regular

The four ways to build a regexp

Atoms

- Each letter is a regexp
- ε is a regexp

Ex: ε, **R** and **F** are regexps

The four ways to build a regexp

Atoms

- Each letter is a regexp
- ε is a regexp

Ex: ε, **R** and **F** are regexps

Concatenation $\cdot$

If Q_1 and Q_2 are regexps
Then $Q_1 \cdot Q_2$ is a regexp

Ex: $\mathbf{R} \cdot \mathbf{R}$ and $\mathbf{G} \cdot \mathbf{F}$ are regexps
$(\mathbf{R} \cdot \mathbf{R}) \cdot (\mathbf{G} \cdot \mathbf{F})$ is a regexp

The four ways to build a regexp

Atoms

- Each letter is a regexp
- ε is a regexp

Ex: ε, **R** and **F** are regexps

Concatenation $\cdot$

If Q_1 and Q_2 are regexps
Then $Q_1 \cdot Q_2$ is a regexp

Ex: $\mathbf{R} \cdot \mathbf{R}$ and $\mathbf{G} \cdot \mathbf{F}$ are regexps
$(\mathbf{R} \cdot \mathbf{R}) \cdot (\mathbf{G} \cdot \mathbf{F})$ is a regexp

Disjunction +

If Q_1 and Q_2 are regexps
Then $Q_1 + Q_2$ is a regexp

Ex: $\mathbf{R} + \mathbf{R}$ and $\mathbf{G} + \mathbf{F}$ are regexps
$(\mathbf{R} \cdot \mathbf{R}) + (\mathbf{G} \cdot \mathbf{F})$ is a regexp

The four ways to build a regexp

Atoms

- Each letter is a regexp
- ε is a regexp

Ex: ε, **R** and **F** are regexps

Concatenation $\cdot$

If Q_1 and Q_2 are regexps
Then $Q_1 \cdot Q_2$ is a regexp

Ex: $\mathbf{R} \cdot \mathbf{R}$ and $\mathbf{G} \cdot \mathbf{F}$ are regexps
$(\mathbf{R} \cdot \mathbf{R}) \cdot (\mathbf{G} \cdot \mathbf{F})$ is a regexp

Disjunction +

If Q_1 and Q_2 are regexps
Then $Q_1 + Q_2$ is a regexp

Ex: $\mathbf{R} + \mathbf{R}$ and $\mathbf{G} + \mathbf{F}$ are regexps
$(\mathbf{R} \cdot \mathbf{R}) + (\mathbf{G} \cdot \mathbf{F})$ is a regexp

Kleene star *

If Q is a regexp
Then Q^* is a regexp

Ex: $\mathbf{R}^*$ and $\mathbf{G}^*$ are regexps
$((\mathbf{R}^* \cdot \mathbf{G}) + \mathbf{F})^*$ is a regexp

A regexp describes a language $L(Q)$

Examples:

1. **R** describes $\{\mathbf{R}\}$

A regexp describes a language $L(Q)$

Examples:

1. **R** describes $\{\mathbf{R}\}$
2. $\mathbf{R} \cdot \mathbf{F} \cdot \mathbf{G}$ describes $\{\mathbf{RFG}\}$

A regexp describes a language $L(Q)$

Examples:

1. **R** describes $\{\mathbf{R}\}$
2. $\mathbf{R} \cdot \mathbf{F} \cdot \mathbf{G}$ describes $\{\mathbf{RFG}\}$
3. $\mathbf{R} + \mathbf{G}$ describes $\{\mathbf{R}, \mathbf{G}\}$

A regexp describes a language $L(Q)$

Examples:

1. $\mathbf{R}$ describes $\{\mathbf{R}\}$
2. $\mathbf{R} \cdot \mathbf{F} \cdot \mathbf{G}$ describes $\{\mathbf{RFG}\}$
3. $\mathbf{R} + \mathbf{G}$ describes $\{\mathbf{R}, \mathbf{G}\}$
4. $(\mathbf{R} + \mathbf{G}) \cdot \mathbf{R}$ describes $\{\mathbf{RR}, \mathbf{GR}\}$

A regexp describes a language $L(Q)$

Examples:

1. **R** describes $\{\mathbf{R}\}$
2. $\mathbf{R} \cdot \mathbf{F} \cdot \mathbf{G}$ describes $\{\mathbf{RFG}\}$
3. $\mathbf{R} + \mathbf{G}$ describes $\{\mathbf{R}, \mathbf{G}\}$
4. $(\mathbf{R} + \mathbf{G}) \cdot \mathbf{R}$ describes $\{\mathbf{RR}, \mathbf{GR}\}$
5. $\mathbf{R}^*$ describes $\{\varepsilon, \mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \ldots\}$

A regexp describes a language $L(Q)$

Examples:

1. **R** describes $\{\mathbf{R}\}$
2. $\mathbf{R} \cdot \mathbf{F} \cdot \mathbf{G}$ describes $\{\mathbf{RFG}\}$
3. $\mathbf{R} + \mathbf{G}$ describes $\{\mathbf{R}, \mathbf{G}\}$
4. $(\mathbf{R} + \mathbf{G}) \cdot \mathbf{R}$ describes $\{\mathbf{RR}, \mathbf{GR}\}$
5. $\mathbf{R}^*$ describes $\{\varepsilon, \mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \ldots\}$

A regexp describes a language $L(Q)$

Examples:

1. $\mathbf{R}$ describes $\{\mathbf{R}\}$
2. $\mathbf{R} \cdot \mathbf{F} \cdot \mathbf{G}$ describes $\{\mathbf{RFG}\}$
3. $\mathbf{R} + \mathbf{G}$ describes $\{\mathbf{R}, \mathbf{G}\}$
4. $(\mathbf{R} + \mathbf{G}) \cdot \mathbf{R}$ describes $\{\mathbf{RR}, \mathbf{GR}\}$
5. $\mathbf{R}^*$ describes $\{\varepsilon, \mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \ldots\}$

Exercices:

6. $(\mathbf{R} + \mathbf{G})^*$ describes
7. $(\mathbf{R} \cdot \mathbf{R})^*$ describes

8. $\mathbf{R}^* \cdot \mathbf{G}^*$ describes

9. $\mathbf{R}^* \cdot \mathbf{G} \cdot \mathbf{R}^*$ describes

A regexp describes a language $L(Q)$

Examples:

1. $\mathbf{R}$ describes $\{\mathbf{R}\}$
2. $\mathbf{R} \cdot \mathbf{F} \cdot \mathbf{G}$ describes $\{\mathbf{RFG}\}$
3. $\mathbf{R} + \mathbf{G}$ describes $\{\mathbf{R}, \mathbf{G}\}$
4. $(\mathbf{R} + \mathbf{G}) \cdot \mathbf{R}$ describes $\{\mathbf{RR}, \mathbf{GR}\}$
5. $\mathbf{R}^*$ describes $\{\varepsilon, \mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \ldots\}$

Exercices:

6. $(\mathbf{R} + \mathbf{G})^*$ describes $\{\varepsilon, \mathbf{R}, \mathbf{G}, \mathbf{RR}, \mathbf{RG}, \mathbf{GG}, \ldots\}$
7. $(\mathbf{R} \cdot \mathbf{R})^*$ describes

8. $\mathbf{R}^* \cdot \mathbf{G}^*$ describes

9. $\mathbf{R}^* \cdot \mathbf{G} \cdot \mathbf{R}^*$ describes

A regexp describes a language $L(Q)$

Examples:

1. $\mathbf{R}$ describes $\{\mathbf{R}\}$
2. $\mathbf{R} \cdot \mathbf{F} \cdot \mathbf{G}$ describes $\{\mathbf{RFG}\}$
3. $\mathbf{R} + \mathbf{G}$ describes $\{\mathbf{R}, \mathbf{G}\}$
4. $(\mathbf{R} + \mathbf{G}) \cdot \mathbf{R}$ describes $\{\mathbf{RR}, \mathbf{GR}\}$
5. $\mathbf{R}^*$ describes $\{\varepsilon, \mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \ldots\}$

Exercices:

6. $(\mathbf{R} + \mathbf{G})^*$ describes $\{\varepsilon, \mathbf{R}, \mathbf{G}, \mathbf{RR}, \mathbf{RG}, \mathbf{GG}, \ldots\}$
7. $(\mathbf{R} \cdot \mathbf{R})^*$ describes $\{\varepsilon, \mathbf{RR}, \mathbf{RRRR}, \mathbf{RRRRRR}, \ldots\}$
   "words of even length"
8. $\mathbf{R}^* \cdot \mathbf{G}^*$ describes

9. $\mathbf{R}^* \cdot \mathbf{G} \cdot \mathbf{R}^*$ describes

A regexp describes a language $L(Q)$

Examples:

1. $\mathbf{R}$ describes $\{\mathbf{R}\}$
2. $\mathbf{R} \cdot \mathbf{F} \cdot \mathbf{G}$ describes $\{\mathbf{RFG}\}$
3. $\mathbf{R} + \mathbf{G}$ describes $\{\mathbf{R}, \mathbf{G}\}$
4. $(\mathbf{R} + \mathbf{G}) \cdot \mathbf{R}$ describes $\{\mathbf{RR}, \mathbf{GR}\}$
5. $\mathbf{R}^*$ describes $\{\varepsilon, \mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \ldots\}$

Exercices:

6. $(\mathbf{R} + \mathbf{G})^*$ describes $\{\varepsilon, \mathbf{R}, \mathbf{G}, \mathbf{RR}, \mathbf{RG}, \mathbf{GG}, \ldots\}$
7. $(\mathbf{R} \cdot \mathbf{R})^*$ describes $\{\varepsilon, \mathbf{RR}, \mathbf{RRRR}, \mathbf{RRRRRR}, \ldots\}$
   "words of even length"
8. $\mathbf{R}^* \cdot \mathbf{G}^*$ describes $\{\varepsilon, \mathbf{R}, \mathbf{G}, \mathbf{RR}, \mathbf{RG}, \mathbf{GG}, \mathbf{RRR}, \mathbf{RRG}, \mathbf{RGG}, \ldots\}$
   "any number of $\mathbf{R}$ *followed by any number of* $\mathbf{G}$*"*
9. $\mathbf{R}^* \cdot \mathbf{G} \cdot \mathbf{R}^*$ describes

A regexp describes a language $L(Q)$

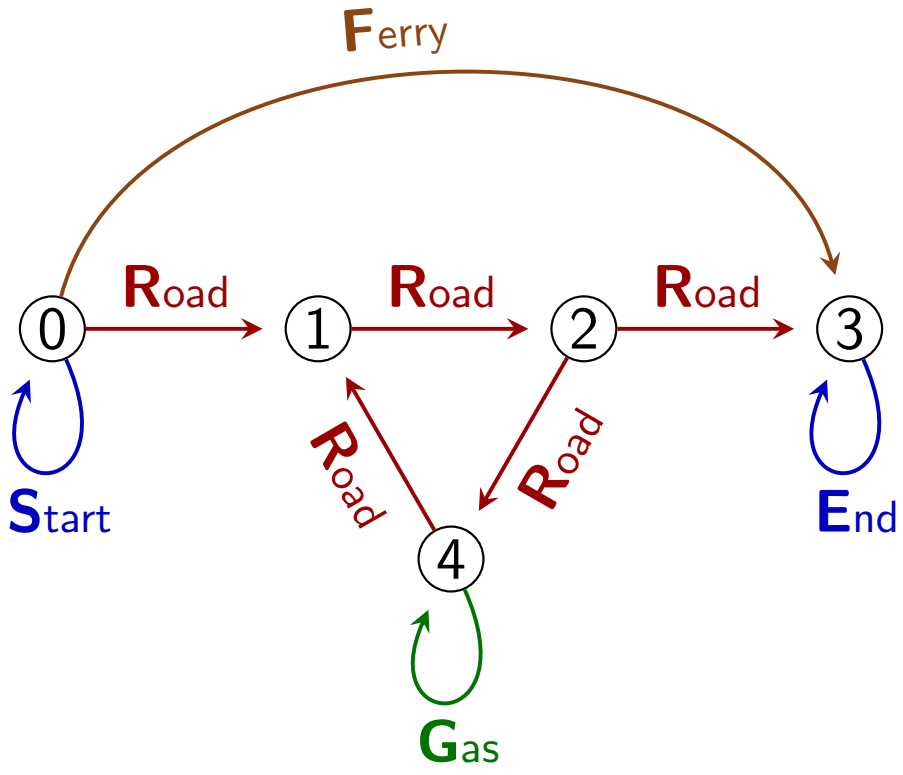
Examples:

1. $\mathbf{R}$ describes $\{\mathbf{R}\}$
2. $\mathbf{R} \cdot \mathbf{F} \cdot \mathbf{G}$ describes $\{\mathbf{RFG}\}$
3. $\mathbf{R} + \mathbf{G}$ describes $\{\mathbf{R}, \mathbf{G}\}$
4. $(\mathbf{R} + \mathbf{G}) \cdot \mathbf{R}$ describes $\{\mathbf{RR}, \mathbf{GR}\}$
5. $\mathbf{R}^*$ describes $\{\varepsilon, \mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \ldots\}$

Exercices:

6. $(\mathbf{R} + \mathbf{G})^*$ describes $\{\varepsilon, \mathbf{R}, \mathbf{G}, \mathbf{RR}, \mathbf{RG}, \mathbf{GG}, \ldots\}$
7. $(\mathbf{R} \cdot \mathbf{R})^*$ describes $\{\varepsilon, \mathbf{RR}, \mathbf{RRRR}, \mathbf{RRRRRR}, \ldots\}$
   "words of even length"
8. $\mathbf{R}^* \cdot \mathbf{G}^*$ describes $\{\varepsilon, \mathbf{R}, \mathbf{G}, \mathbf{RR}, \mathbf{RG}, \mathbf{GG}, \mathbf{RRR}, \mathbf{RRG}, \mathbf{RGG}, \ldots\}$
   "any number of $\mathbf{R}$ *followed by any number of* $\mathbf{G}$*"*
9. $\mathbf{R}^* \cdot \mathbf{G} \cdot \mathbf{R}^*$ describes $\{\mathbf{G}, \mathbf{RG}, \mathbf{GR}, \mathbf{RGR}, \mathbf{RRG}, \ldots\}$
   "words over $\{\mathbf{G}, \mathbf{R}\}$ *with exactly one* $\mathbf{G}$*"*

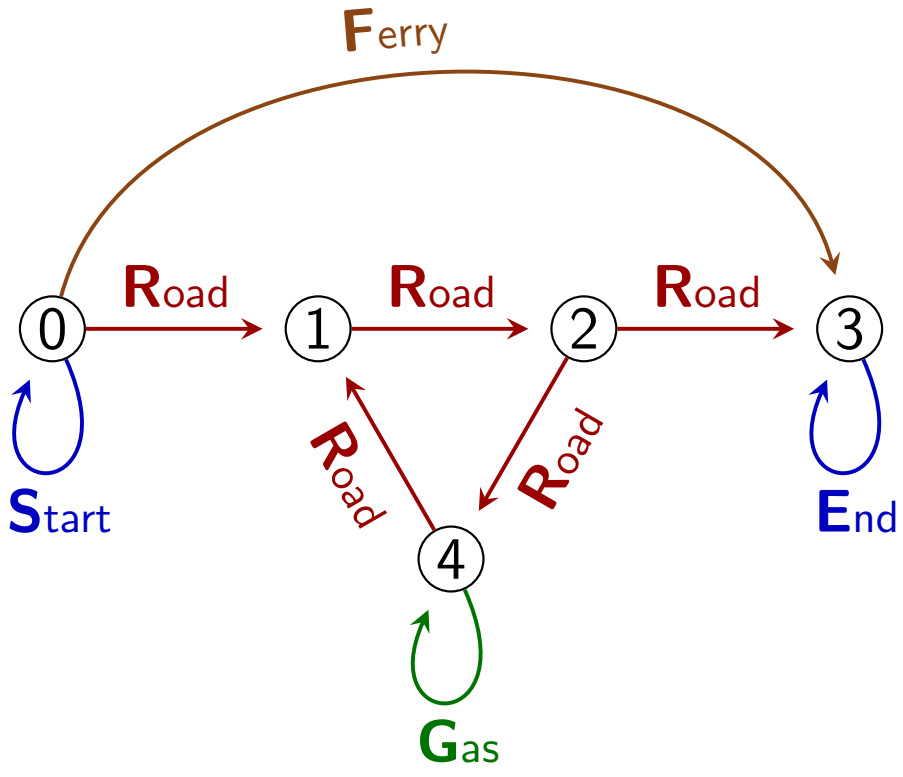
From walks to words



From walks to words

A **walk** in $\mathcal{D}$ is a consistent sequence of edges in $\mathcal{D}$.

The **label of a walk** is the **word** formed by the label of its edges.

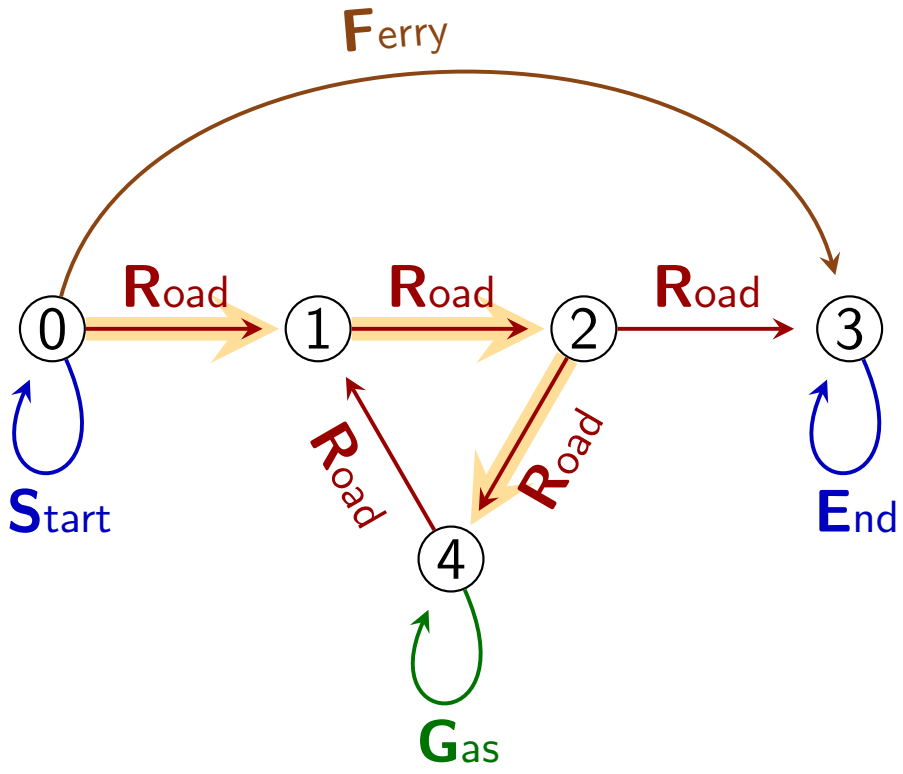


From walks to words

A **walk** in $\mathcal{D}$ is a consistent sequence of edges in $\mathcal{D}$.

The **label of a walk** is the **word** formed by the label of its edges.

Example walk	Label
$0 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 4$	**RRR**
$0 \xrightarrow{S} 0 \xrightarrow{F} 3$	**SF**
$0 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 4 \xrightarrow{G}$ $4 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 3$	**RRRGRRR**

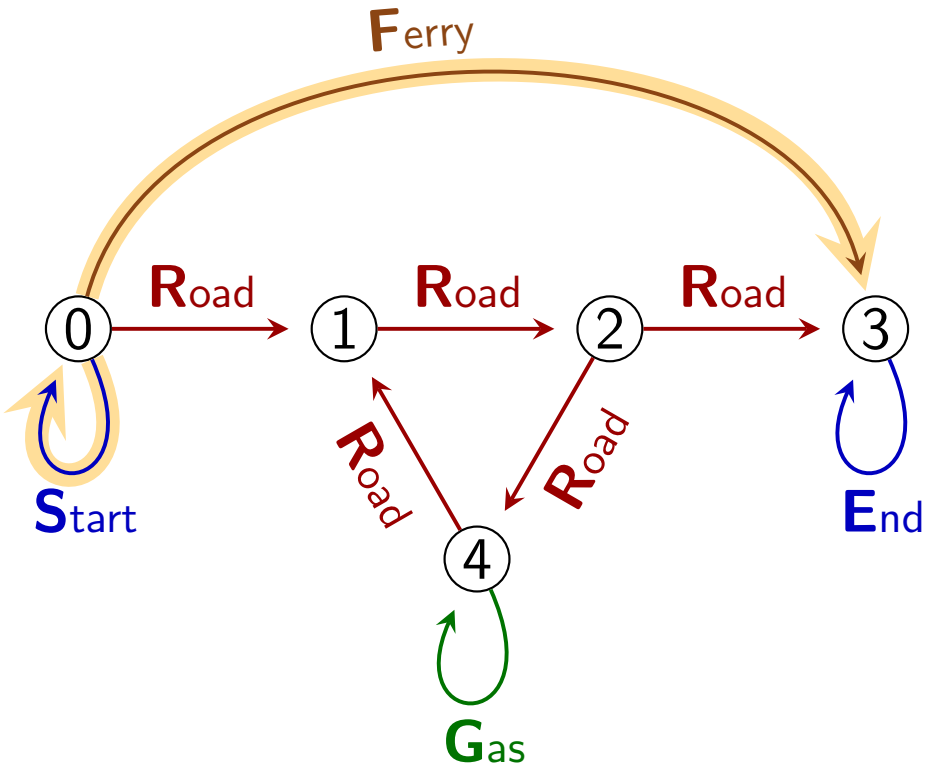


From walks to words

A **walk** in $\mathcal{D}$ is a consistent sequence of edges in $\mathcal{D}$.

The **label of a walk** is the **word** formed by the label of its edges.

Example walk	Label
$0 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 4$	**RRR**
$0 \xrightarrow{S} 0 \xrightarrow{F} 3$	**SF**
$0 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 4 \xrightarrow{G}$ $4 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 3$	**RRRGRRR**

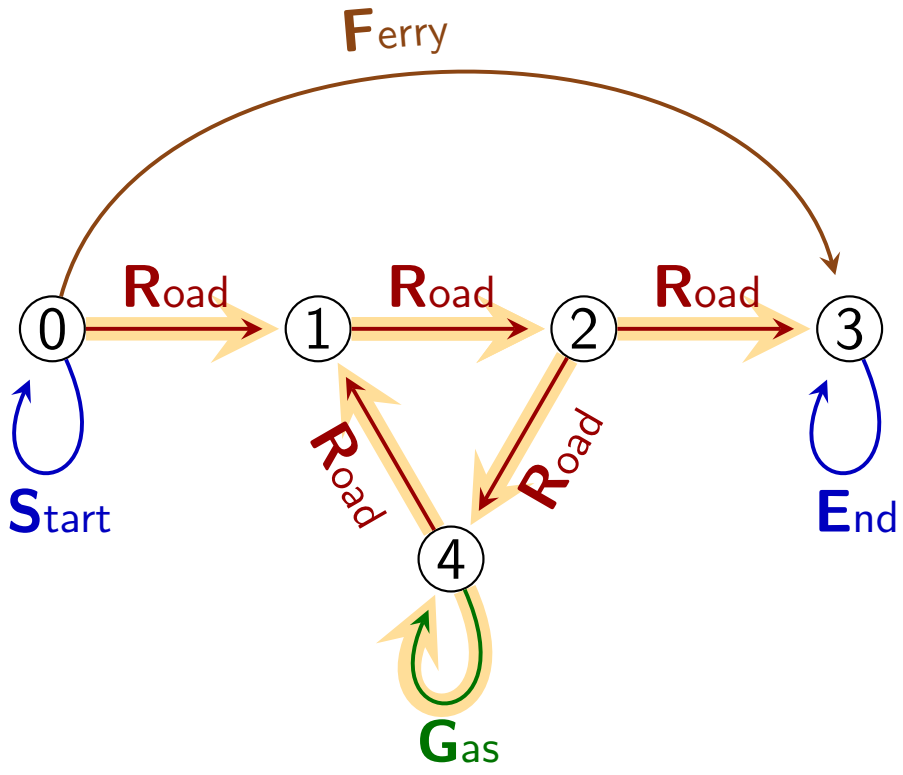


From walks to words

A **walk** in $\mathcal{D}$ is a consistent sequence of edges in $\mathcal{D}$.

The **label of a walk** is the **word** formed by the label of its edges.

Example walk	Label
$0 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 4$	**RRR**
$0 \xrightarrow{S} 0 \xrightarrow{F} 3$	**SF**
$0 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 4 \xrightarrow{G}$ $4 \xrightarrow{R} 1 \xrightarrow{R} 2 \xrightarrow{R} 3$	**RRRGRRR**

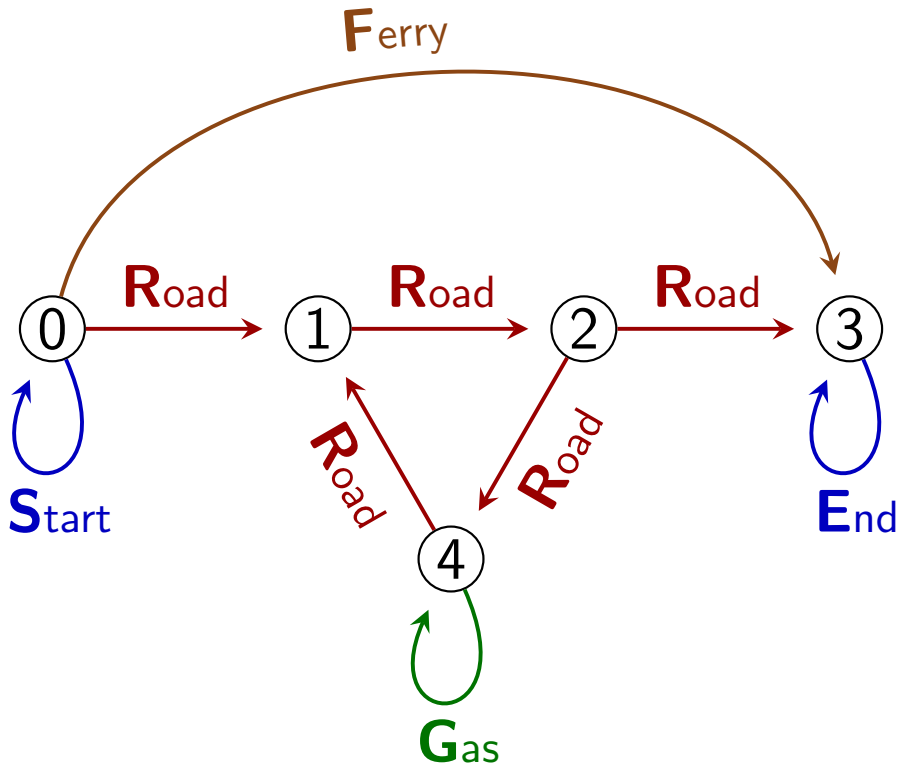


From walks to words

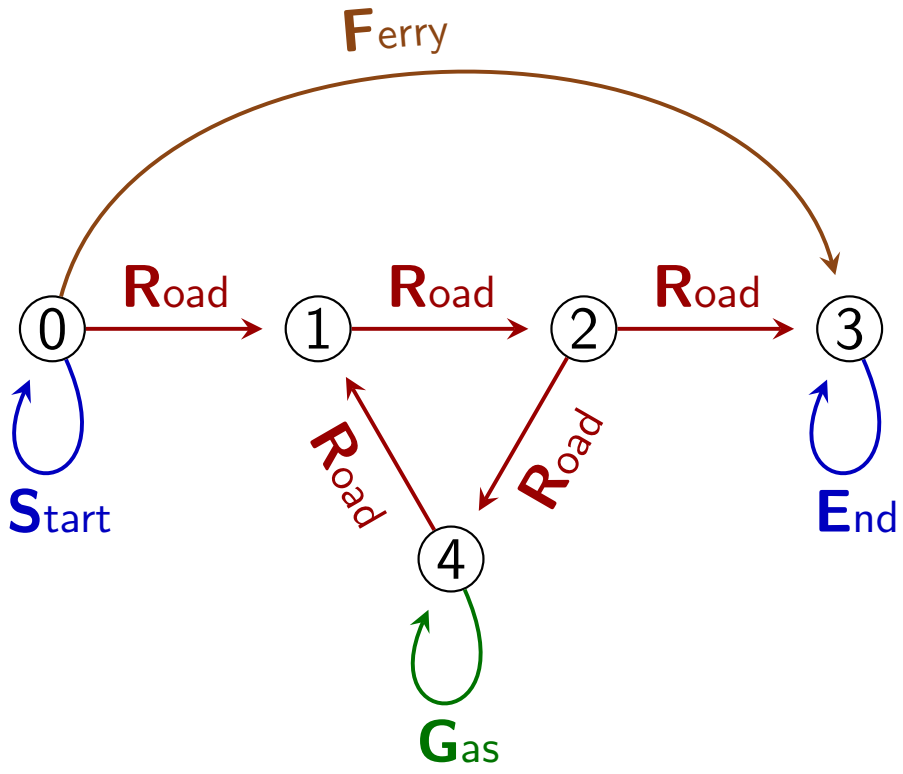
A **walk** in $\mathcal{D}$ is a consistent sequence of edges in $\mathcal{D}$.

The **label of a walk** is the **word** formed by the label of its edges.

Example walk	Label
$0 \xrightarrow{\mathbf{R}} 1 \xrightarrow{\mathbf{R}} 2 \xrightarrow{\mathbf{R}} 4$	**RRR**
$0 \xrightarrow{\mathbf{S}} 0 \xrightarrow{\mathbf{F}} 3$	**SF**
$0 \xrightarrow{\mathbf{R}} 1 \xrightarrow{\mathbf{R}} 2 \xrightarrow{\mathbf{R}} 4 \xrightarrow{\mathbf{G}} 4 \xrightarrow{\mathbf{R}} 1 \xrightarrow{\mathbf{R}} 2 \xrightarrow{\mathbf{R}} 3$	**RRRGRRR**



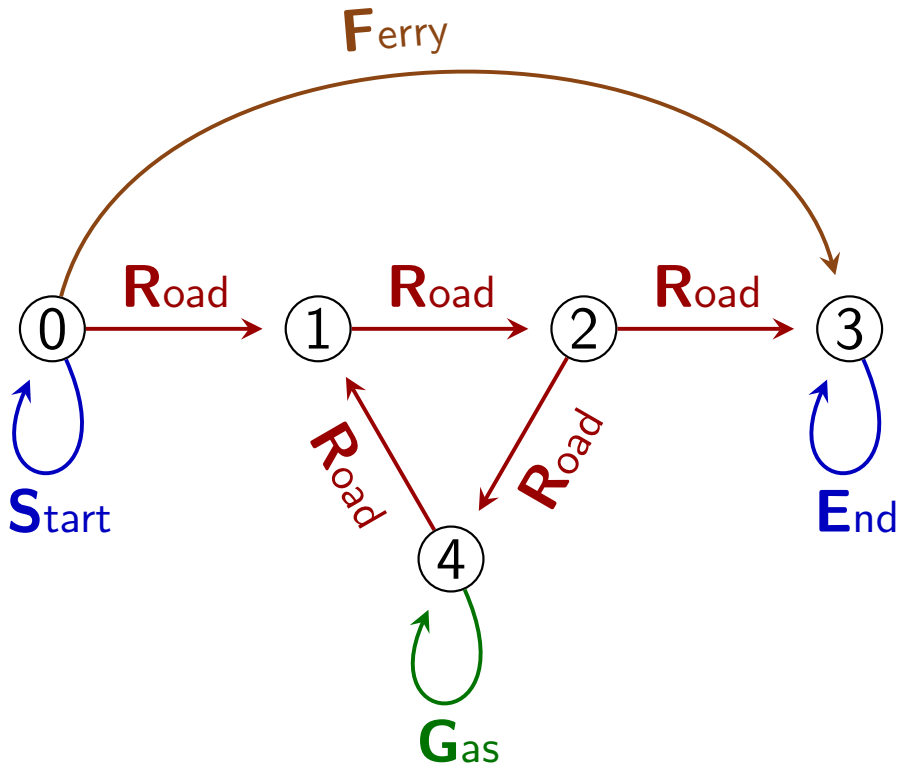
Matching walks with RPQs



Matching walks with RPQs

A **Regular Path Query (RPQ)**

- queries a graph $\mathcal{D} = (V, L, E)$
- is a **regexp** over L
- **matches** a set of **walks** in $\mathcal{D}$

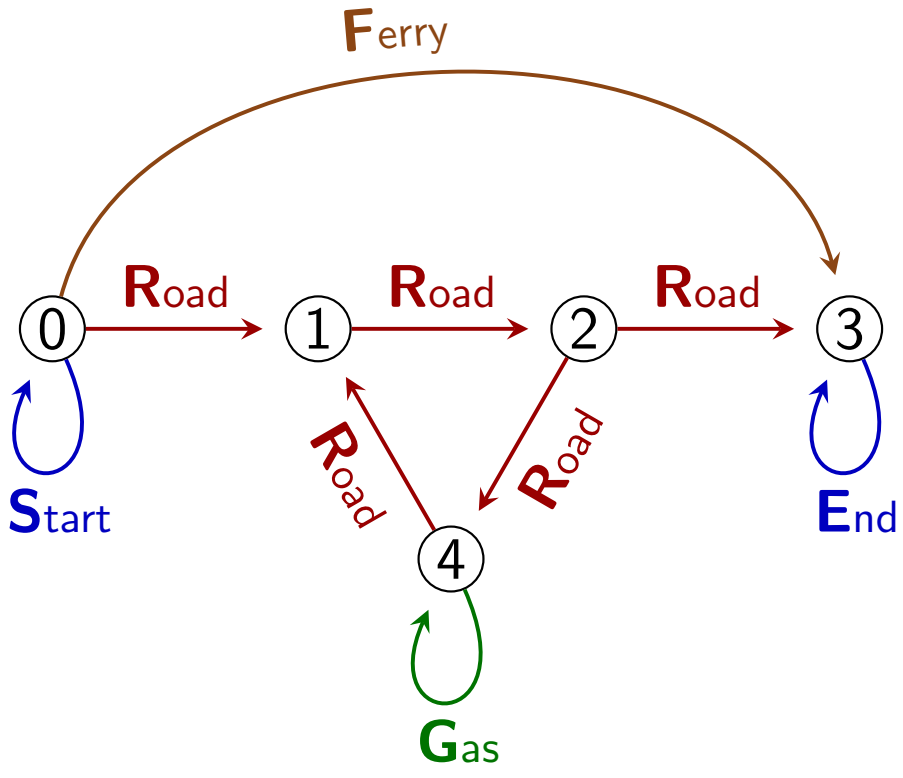


Matching walks with RPQs

A **Regular Path Query (RPQ)**

- queries a graph $\mathcal{D} = (V, L, E)$
- is a **regexp** over L
- **matches** a set of **walks** in $\mathcal{D}$

Given an **RPQ** Q,
a **walk** w is a **match** to Q
if the **label** of w conforms to Q.

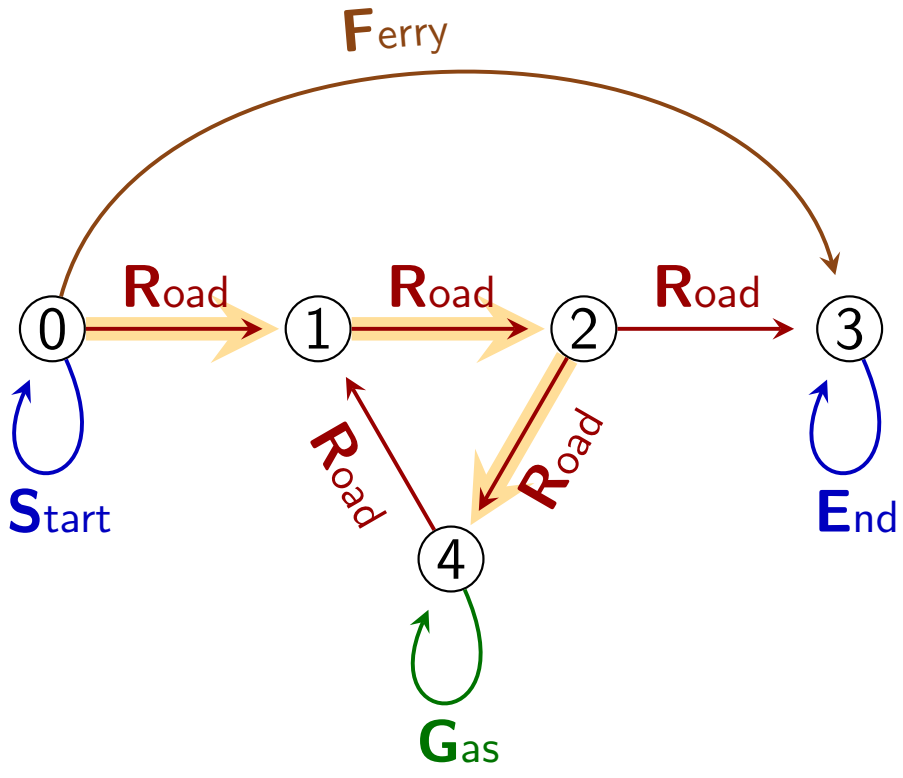


Matching walks with RPQs

A **Regular Path Query (RPQ)**
- queries a graph $\mathcal{D} = (V, L, E)$
- is a **regexp** over L
- **matches** a set of **walks** in $\mathcal{D}$

Given an **RPQ** Q,
a **walk** w is a **match** to Q
if the **label** of w conforms to Q.

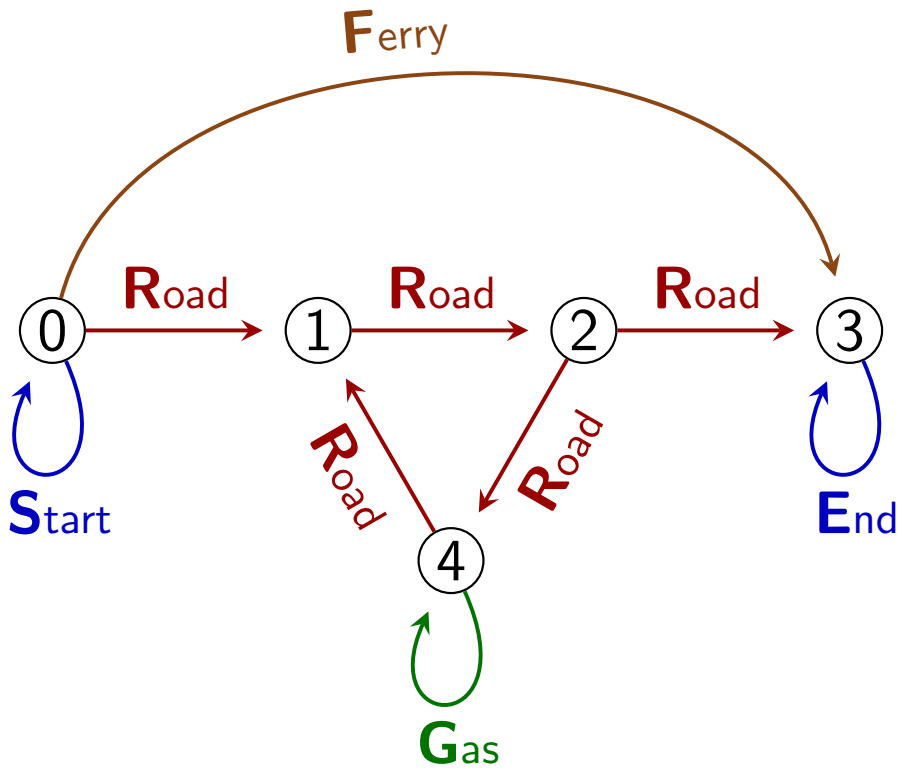
First example:
- $Q = \mathbf{R}^*$
- $w = 0 \to 1 \to 2 \to 4$
- $\ell = \mathbf{RRR}$



Matching atoms

Matching query Q_1 = **R**

$L(Q_1) = \{\mathbf{R}\}$

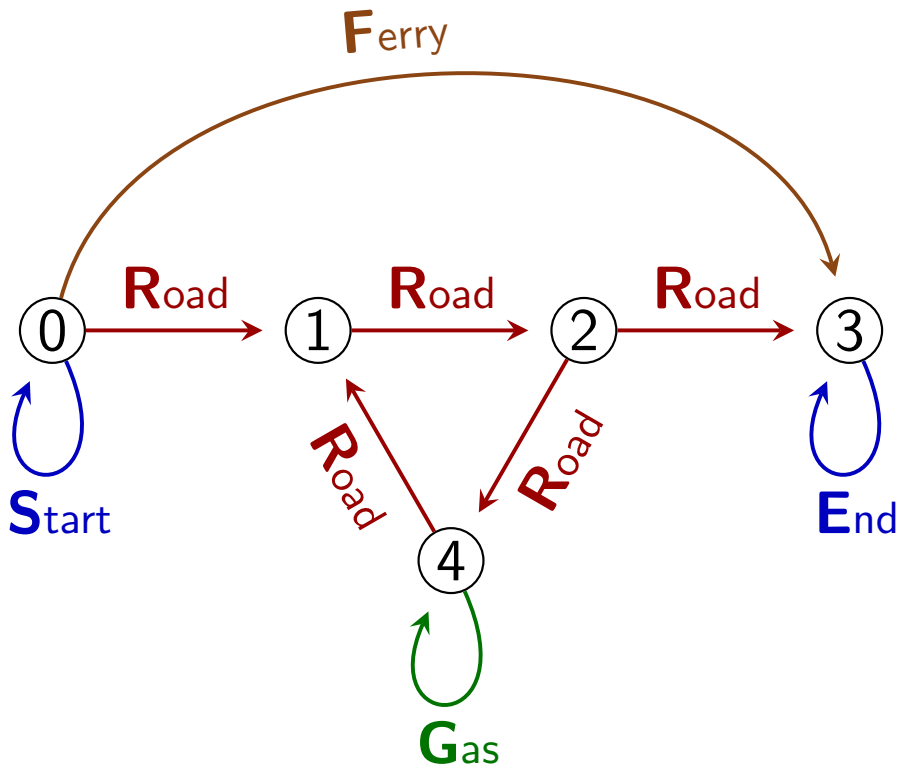


Matching atoms

Matching query Q_1 = **R**

$L(Q_1) = \{\mathbf{R}\}$

The matches to Q_1 are the walks labeled by some word in $L(Q_1)$, that is labeled by **R**.



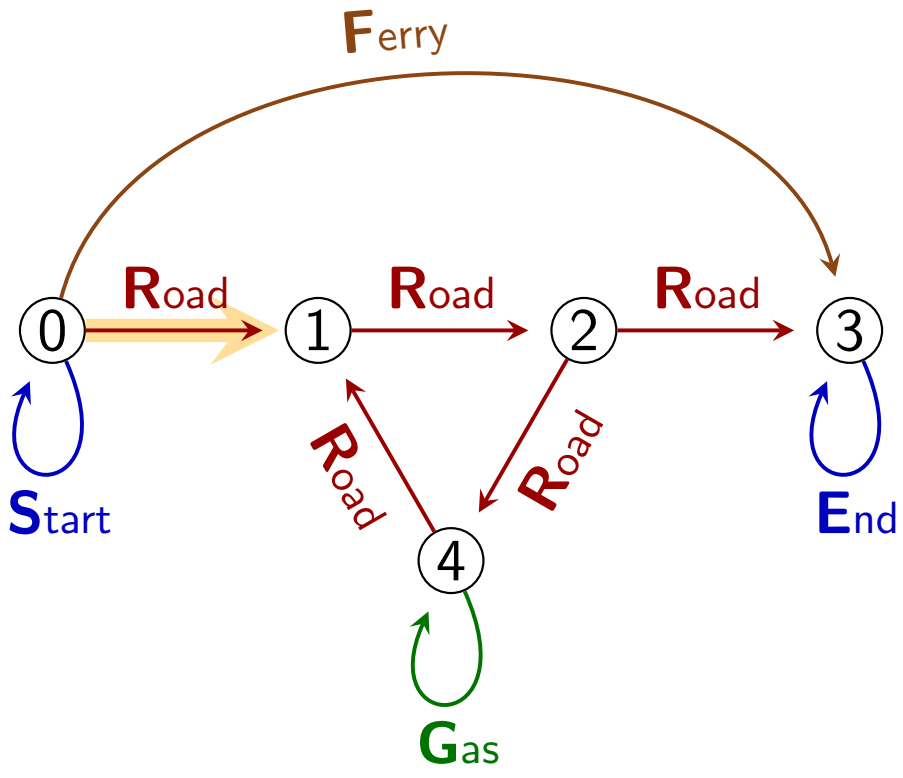
Matching atoms

Matching query Q_1 = **R**

$L(Q_1) = \{\mathbf{R}\}$

The matches to Q_1 are the walks labeled by some word in $L(Q_1)$, that is labeled by **R**.

Match for Q_1	Label
$0 \to 1$	**R**



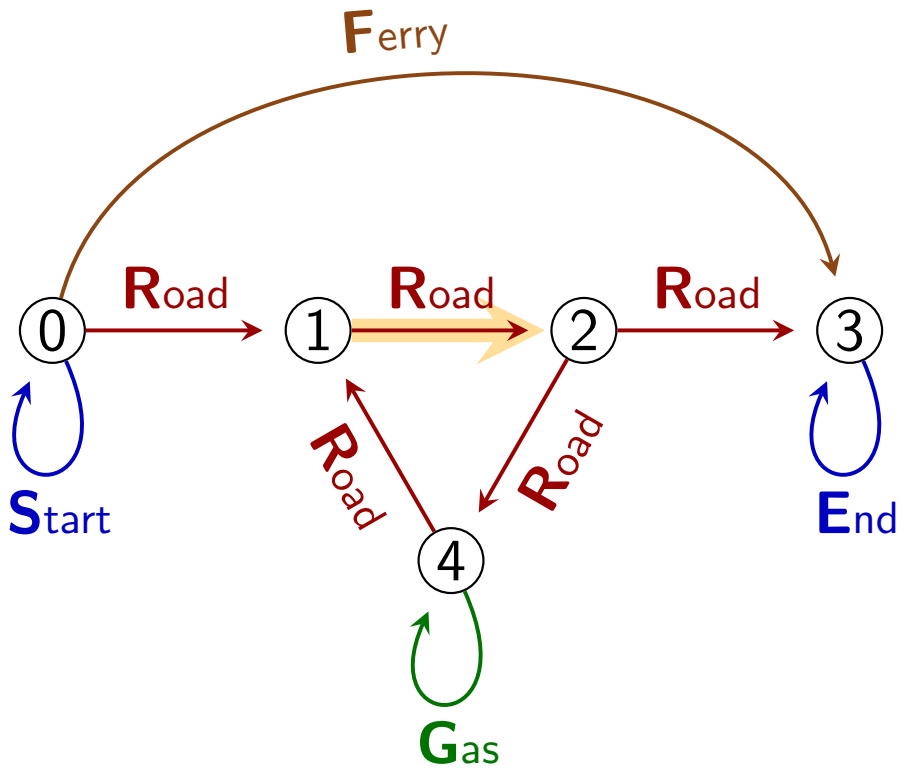
Matching atoms

Matching query Q_1 = **R**

$L(Q_1) = \{\mathbf{R}\}$

The matches to Q_1 are the walks labeled by some word in $L(Q_1)$, that is labeled by **R**.

Match for Q_1	Label
$0 \to 1$	**R**
$1 \to 2$	**R**



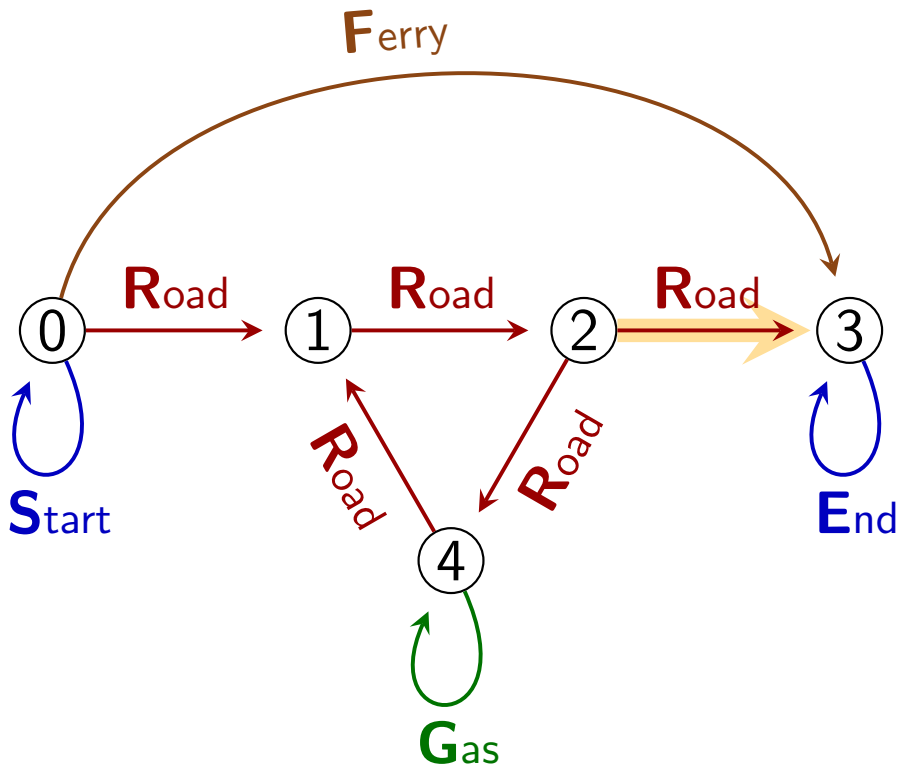
Matching atoms

Matching query Q_1 = **R**

$L(Q_1) = \{\mathbf{R}\}$

The matches to Q_1 are the walks labeled by some word in $L(Q_1)$, that is labeled by **R**.

Match for Q_1	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**



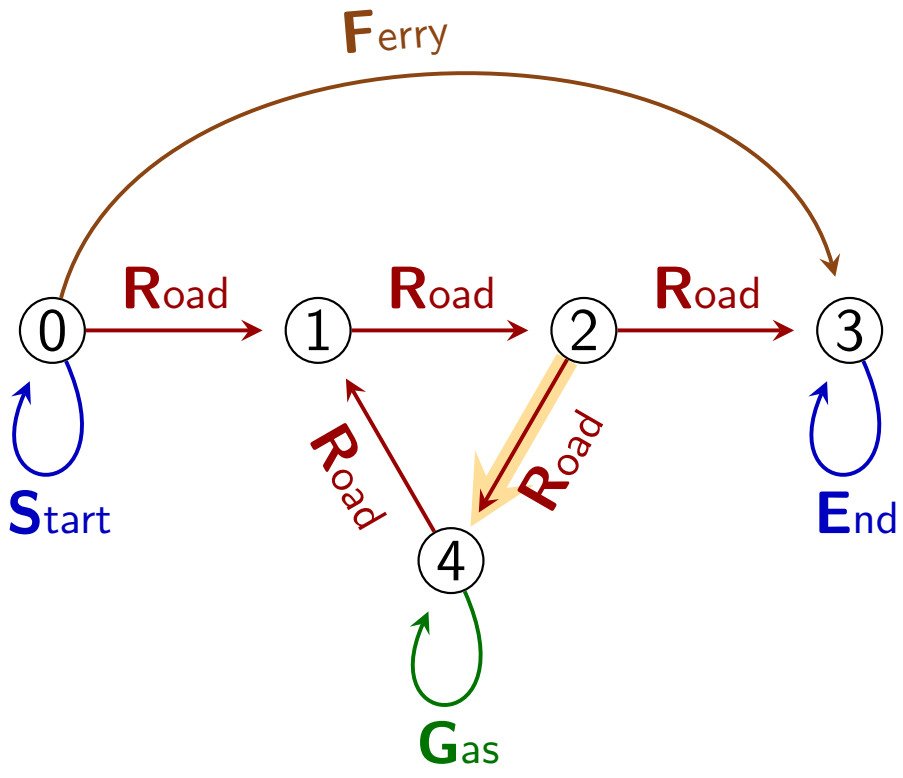
Matching atoms

Matching query Q_1 = **R**

$L(Q_1) = \{\mathbf{R}\}$

The matches to Q_1 are the walks labeled by some word in $L(Q_1)$, that is labeled by **R**.

Match for Q_1	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**



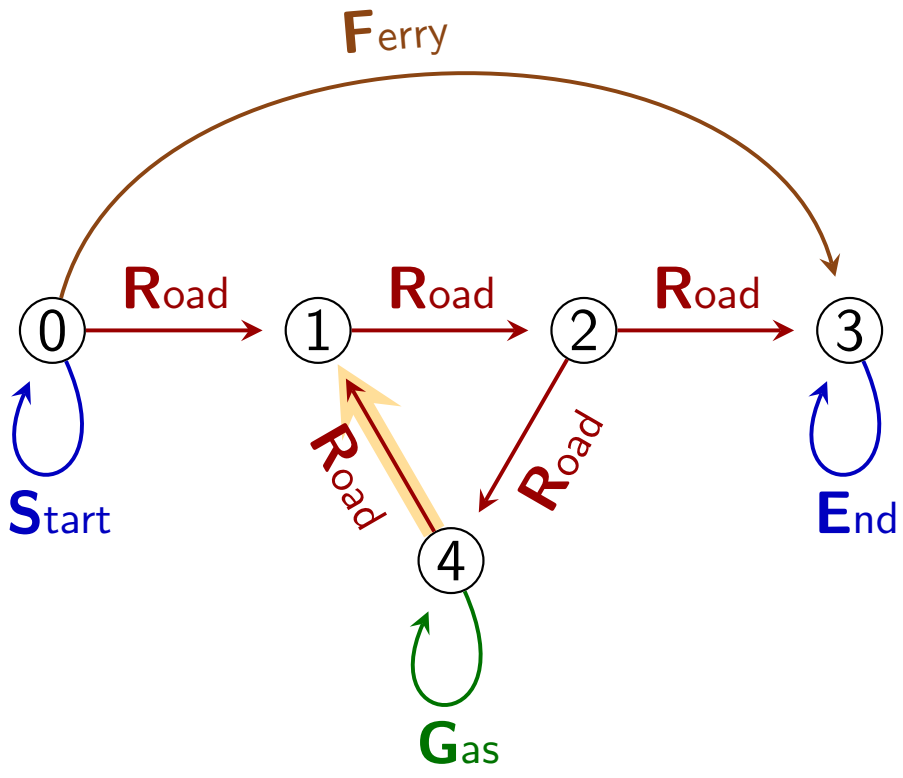
Matching atoms

Matching query Q_1 = **R**

$L(Q_1) = \{\mathbf{R}\}$

The matches to Q_1 are the walks labeled by some word in $L(Q_1)$, that is labeled by **R**.

Match for Q_1	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**
$4 \to 1$	**R**



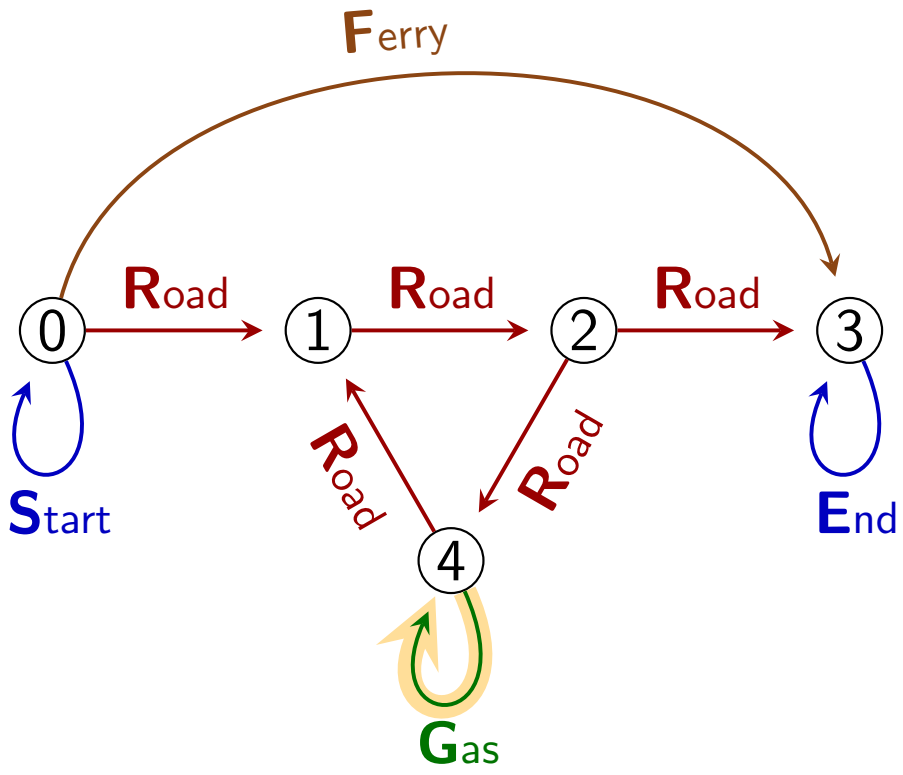
Matching atoms

Matching query Q_1 = **R**

$L(Q_1) = \{\mathbf{R}\}$

The matches to Q_1 are the walks labeled by some word in $L(Q_1)$, that is labeled by **R**.

Match for Q_1	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**
$4 \to 1$	**R**



Matching Q_2 = **G**

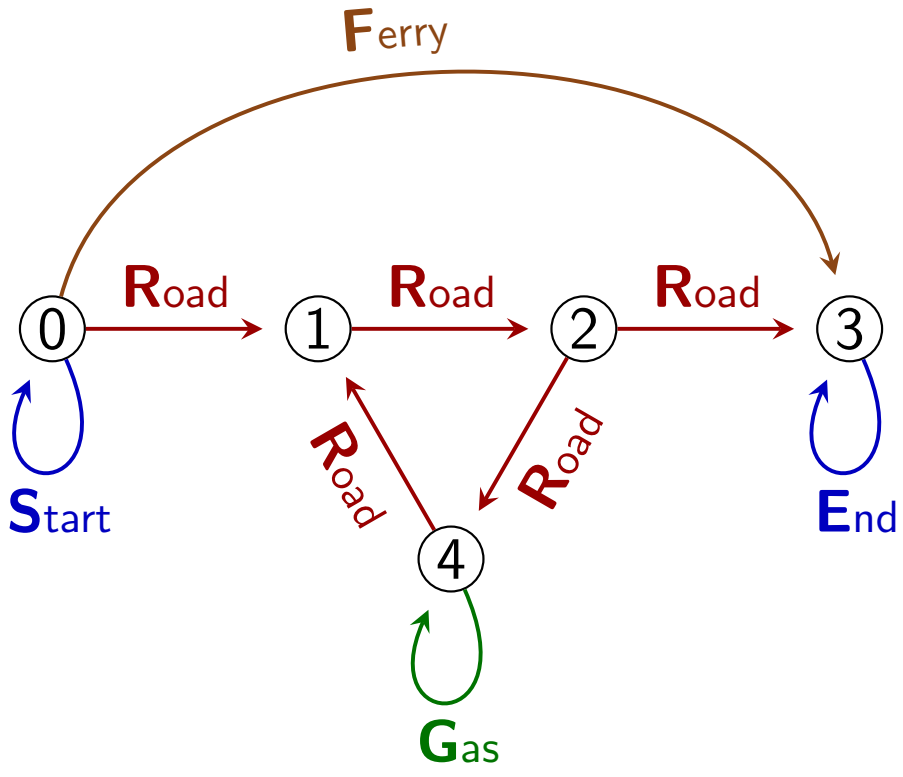
$L(Q_2) = \{\mathbf{G}\}$

Match for Q_2	Label
$4 \to 4$	**G**

Disjunction

$Q_3 = \mathbf{R} + \mathbf{F}$

$L(Q_3) = \{\mathbf{R}, \mathbf{F}\}$

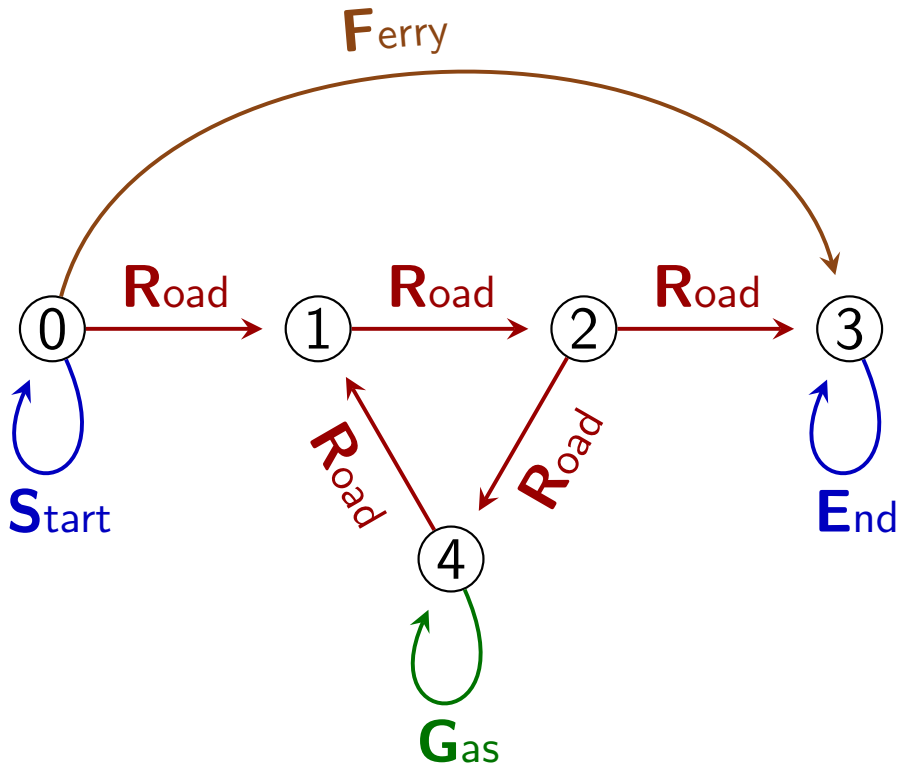


Disjunction

$Q_3 = \mathbf{R}+\mathbf{F}$

$L(Q_3) = \{\mathbf{R}, \mathbf{F}\}$

The matches to Q_3 are the walks labeled by some word in $L(Q_3)$, that is labeled by **R** or by **F**.



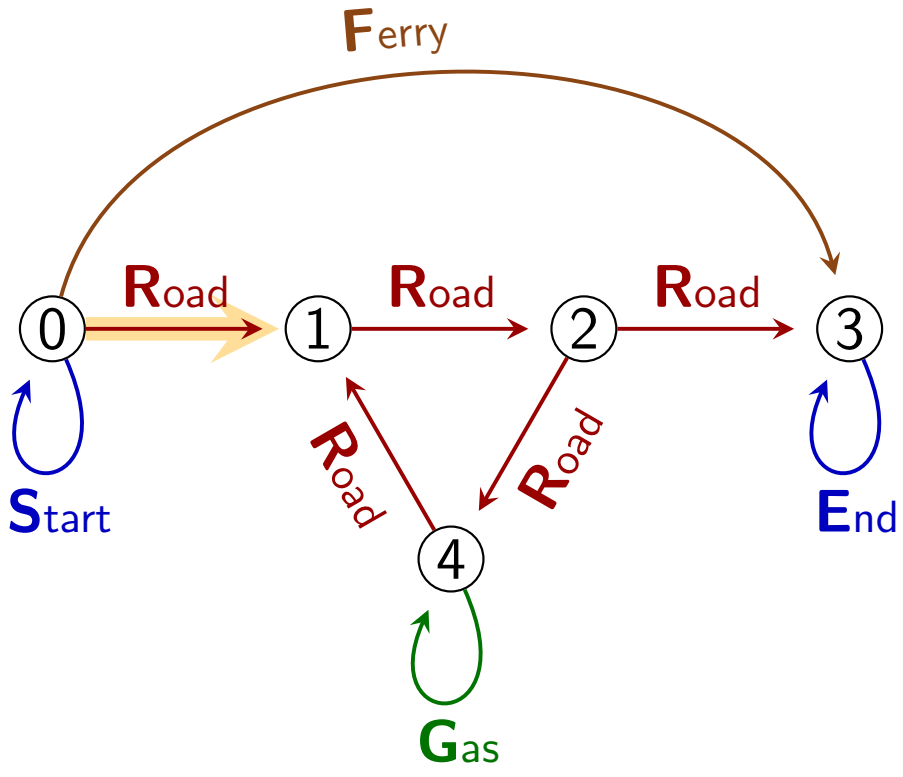
Disjunction

$Q_3 = \mathbf{R} + \mathbf{F}$

$L(Q_3) = \{\mathbf{R}, \mathbf{F}\}$

The matches to Q_3 are the walks labeled by some word in $L(Q_3)$, that is labeled by **R** or by **F**.

Match for Q_3	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**
$4 \to 1$	**R**
$0 \to 3$	**F**



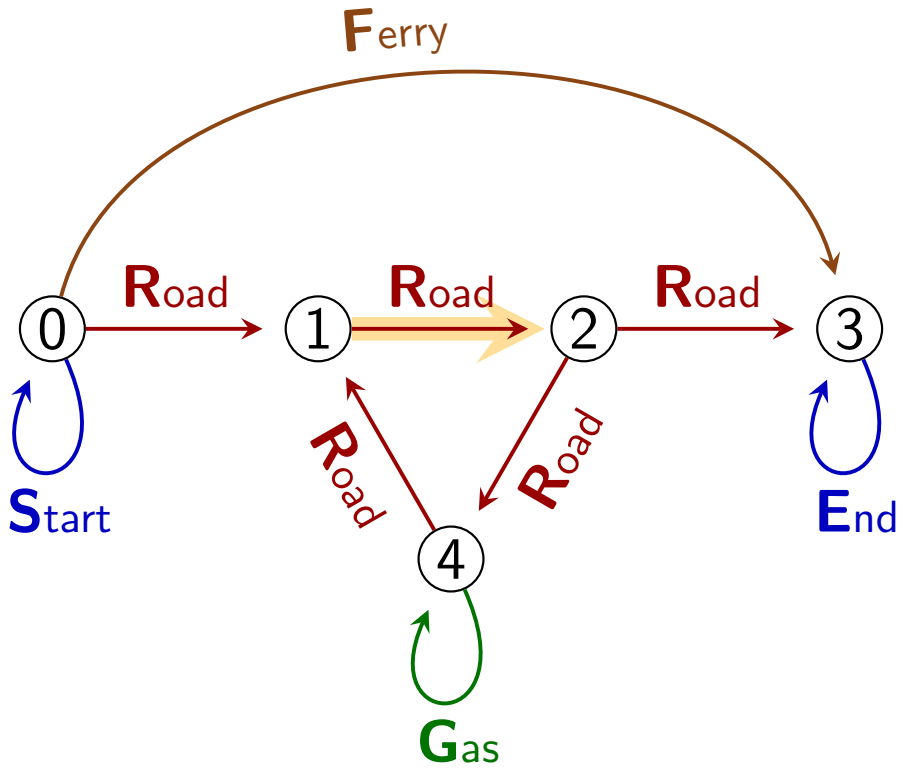
Disjunction

$Q_3 = \mathbf{R} + \mathbf{F}$

$L(Q_3) = \{\mathbf{R}, \mathbf{F}\}$

The matches to Q_3 are the walks labeled by some word in $L(Q_3)$, that is labeled by **R** or by **F**.

Match for Q_3	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**
$4 \to 1$	**R**
$0 \to 3$	**F**



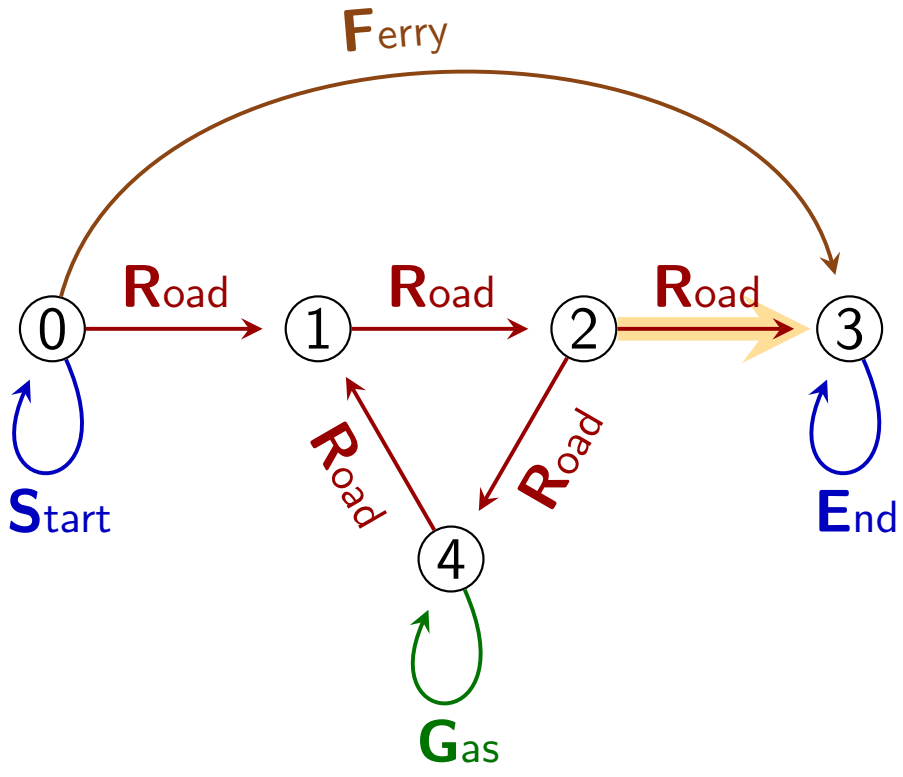
Disjunction

$Q_3 = \mathbf{R} + \mathbf{F}$

$L(Q_3) = \{\mathbf{R}, \mathbf{F}\}$

The matches to Q_3 are the walks labeled by some word in $L(Q_3)$, that is labeled by **R** or by **F**.

Match for Q_3	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**
$4 \to 1$	**R**
$0 \to 3$	**F**



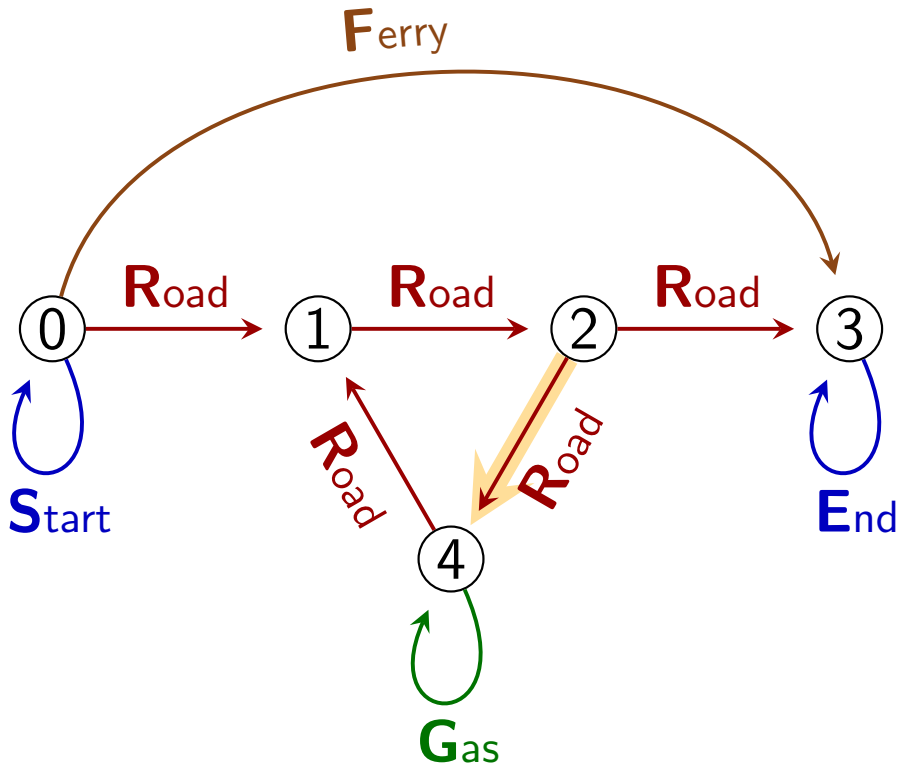
Disjunction

$Q_3 = \mathbf{R}+\mathbf{F}$

$L(Q_3) = \{\mathbf{R}, \mathbf{F}\}$

The matches to Q_3 are the walks labeled by some word in $L(Q_3)$, that is labeled by **R** or by **F**.

Match for Q_3	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**
$4 \to 1$	**R**
$0 \to 3$	**F**



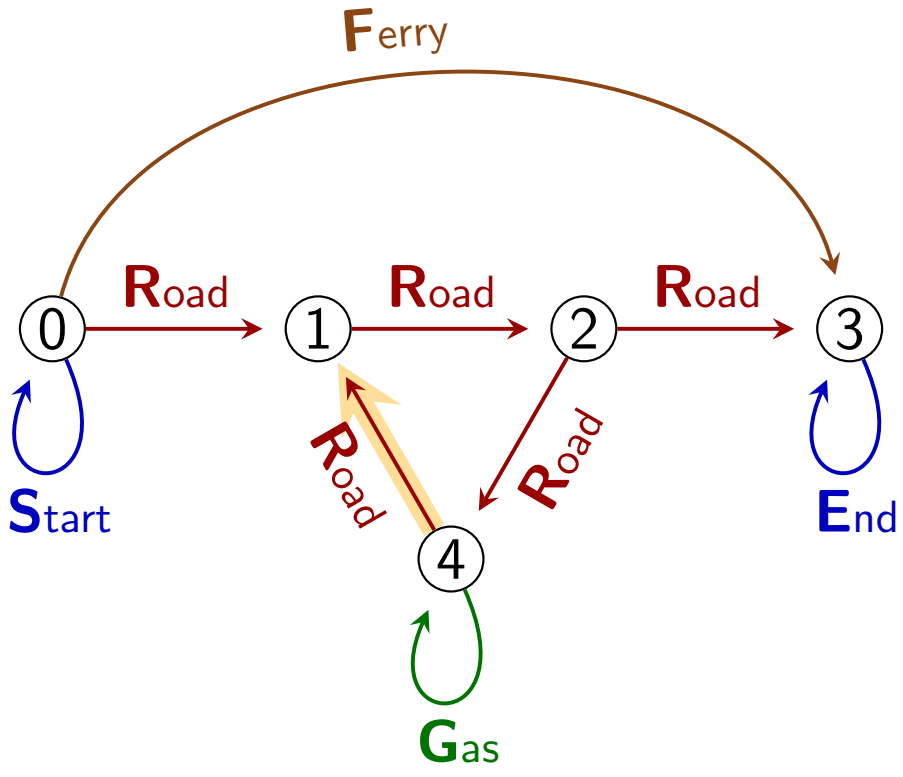
Disjunction

$Q_3 = \mathbf{R}+\mathbf{F}$

$L(Q_3) = \{\mathbf{R}, \mathbf{F}\}$

The matches to Q_3 are the walks labeled by some word in $L(Q_3)$, that is labeled by **R** or by **F**.

Match for Q_3	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**
$4 \to 1$	**R**
$0 \to 3$	**F**



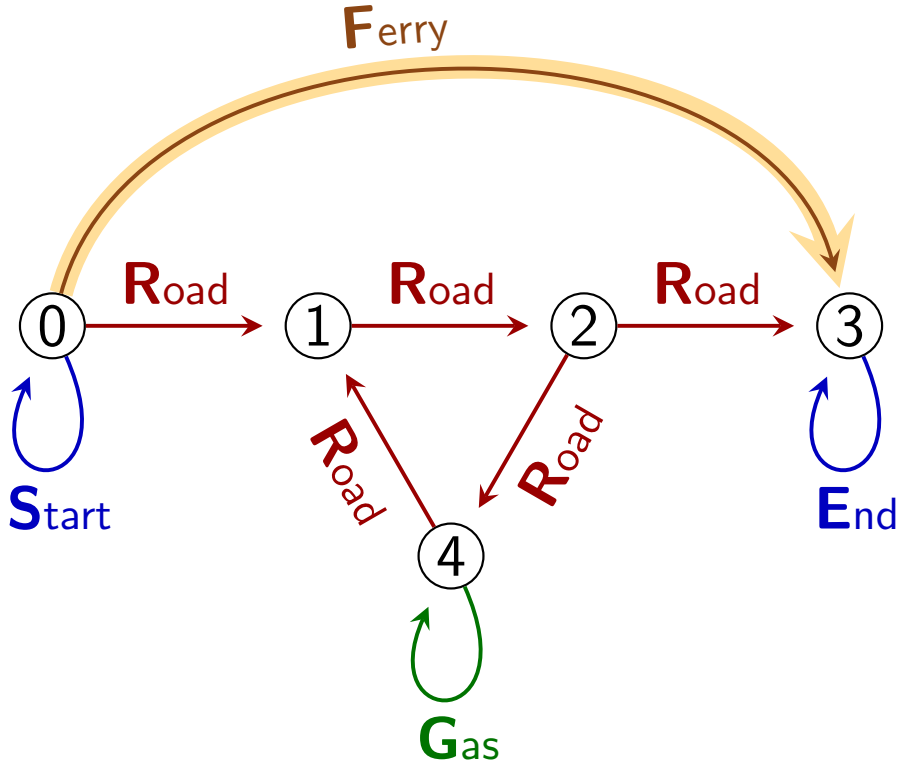
Disjunction

$Q_3 = \mathbf{R}+\mathbf{F}$

$L(Q_3) = \{\mathbf{R}, \mathbf{F}\}$

The matches to Q_3 are the walks labeled by some word in $L(Q_3)$, that is labeled by **R** or by **F**.

Match for Q_3	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**
$4 \to 1$	**R**
$0 \to 3$	**F**



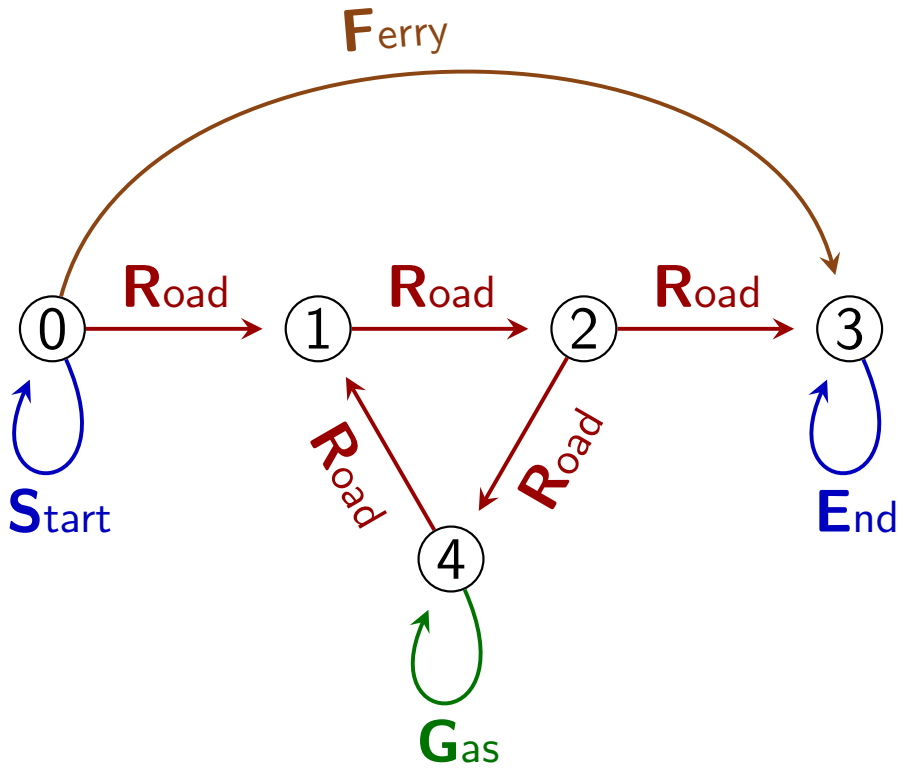
Disjunction

$Q_3 = \mathbf{R}+\mathbf{F}$

$L(Q_3) = \{\mathbf{R}, \mathbf{F}\}$

The matches to Q_3 are the walks labeled by some word in $L(Q_3)$, that is labeled by **R** or by **F**.

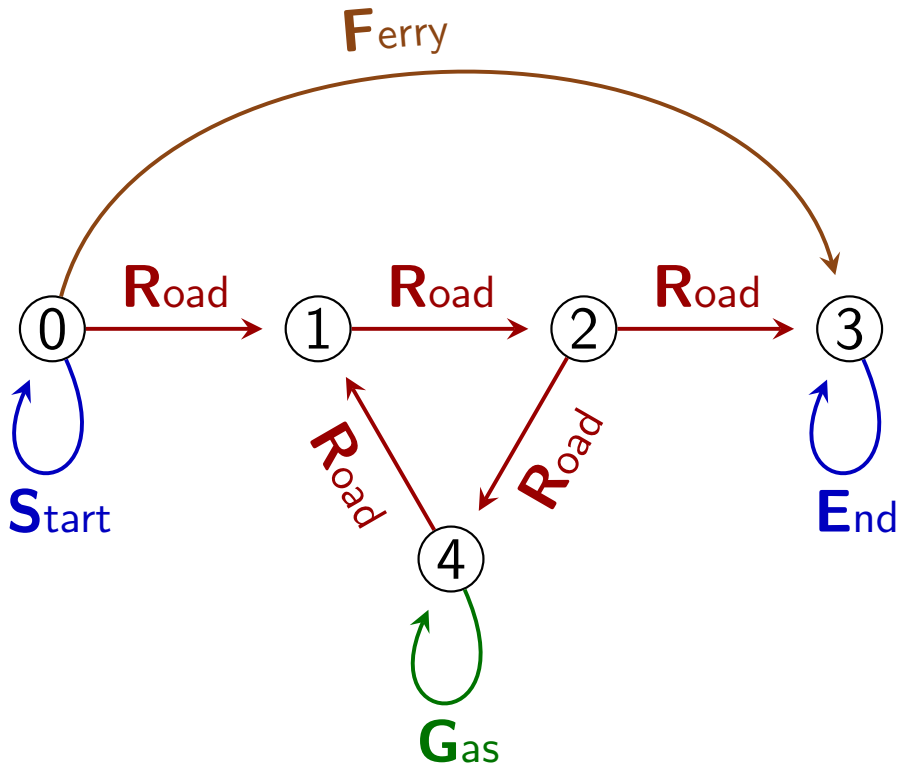
Match for Q_3	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$2 \to 3$	**R**
$2 \to 4$	**R**
$4 \to 1$	**R**
$0 \to 3$	**F**



Concatenation

$Q_4 = \mathbf{R} \cdot \mathbf{R}$

$L(Q_4) = \{\mathbf{RR}\}$

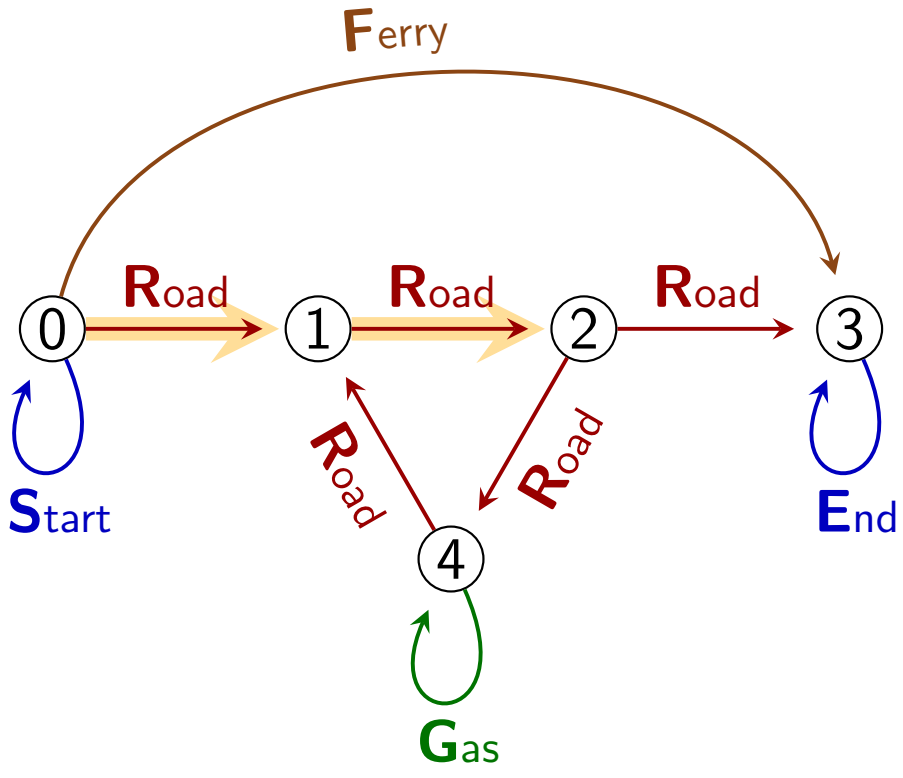


Concatenation

$Q_4 = \mathbf{R} \cdot \mathbf{R}$

$L(Q_4) = \{\mathbf{RR}\}$

Match for Q_4	Label
$0 \to 1 \to 2$	**RR**

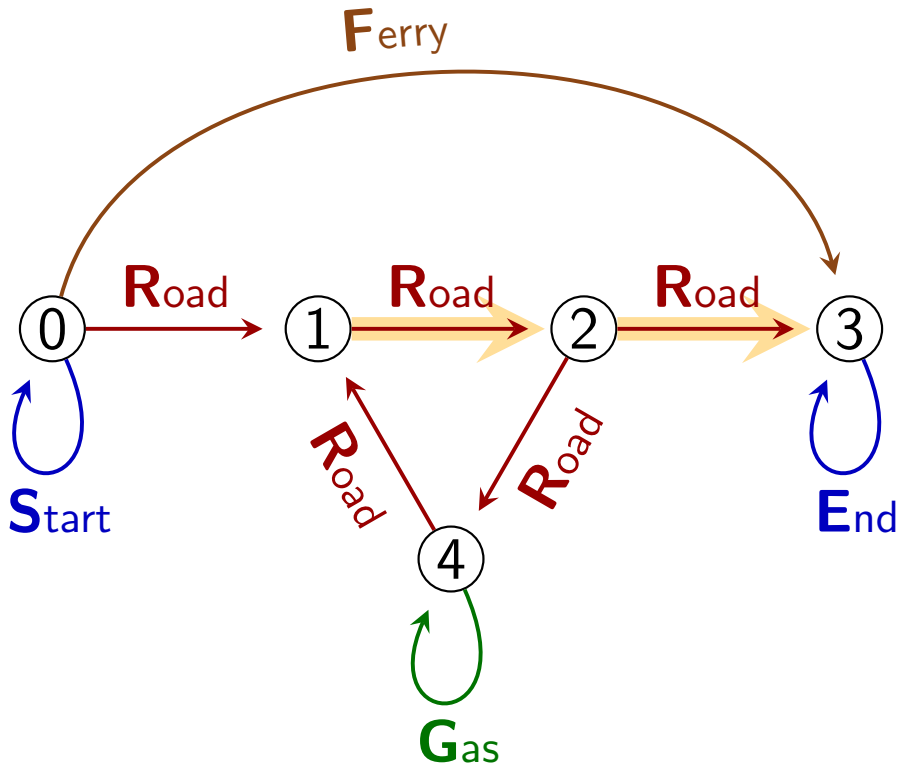


Concatenation

$Q_4 = \mathbf{R} \cdot \mathbf{R}$

$L(Q_4) = \{\mathbf{RR}\}$

Match for Q_4	Label
$0 \to 1 \to 2$	**RR**
$1 \to 2 \to 3$	**RR**

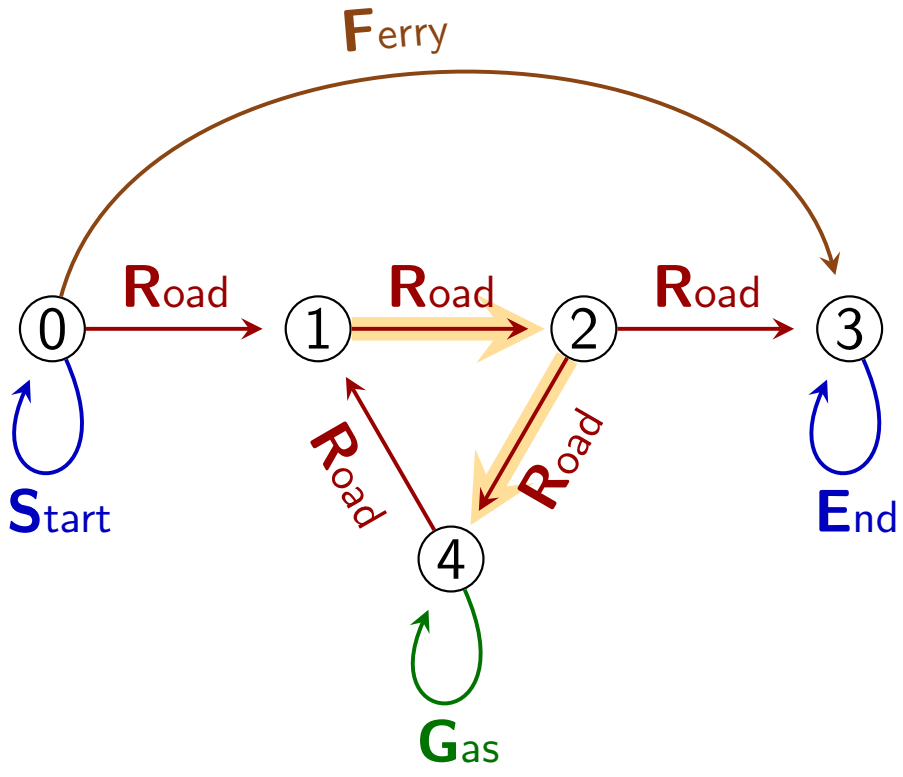


Concatenation

$Q_4 = \mathbf{R} \cdot \mathbf{R}$

$L(Q_4) = \{\mathbf{RR}\}$

Match for Q_4	Label
$0 \to 1 \to 2$	**RR**
$1 \to 2 \to 3$	**RR**
$1 \to 2 \to 4$	**RR**

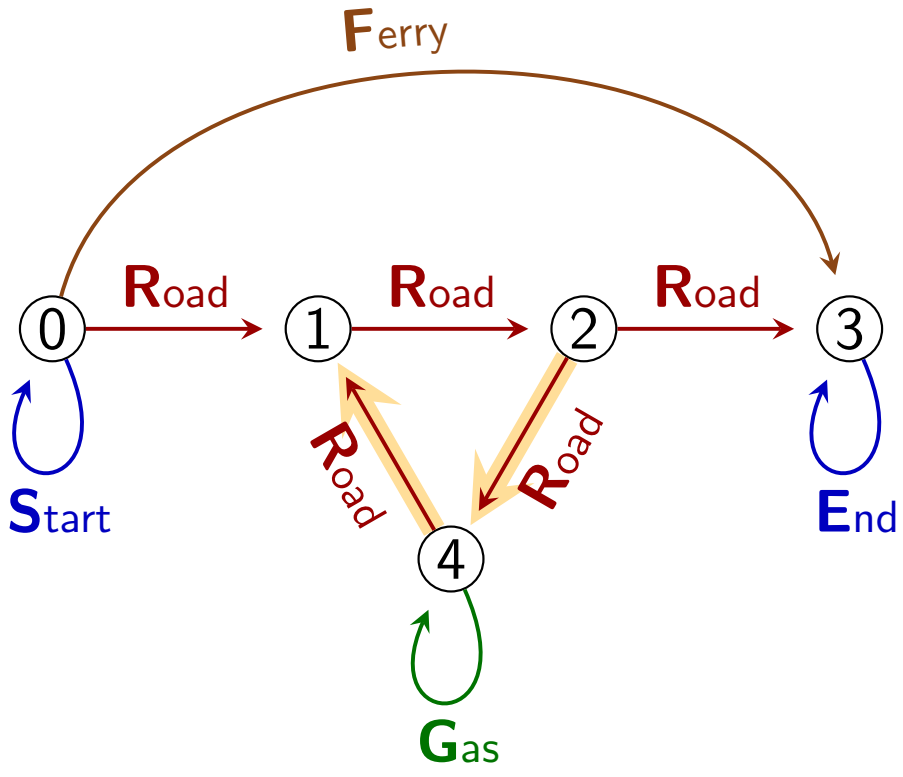


Concatenation

$Q_4 = \mathbf{R} \cdot \mathbf{R}$

$L(Q_4) = \{\mathbf{RR}\}$

Match for Q_4	Label
$0 \to 1 \to 2$	**RR**
$1 \to 2 \to 3$	**RR**
$1 \to 2 \to 4$	**RR**
$2 \to 4 \to 1$	**RR**

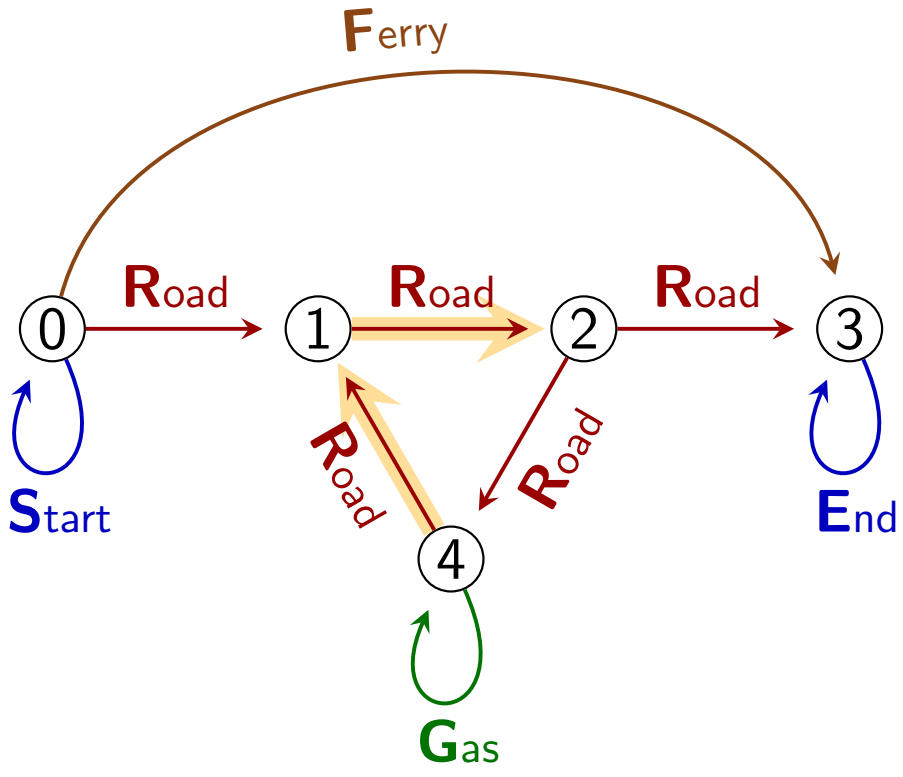


Concatenation

$Q_4 = \mathbf{R} \cdot \mathbf{R}$

$L(Q_4) = \{\mathbf{RR}\}$

Match for Q_4	Label
$0 \to 1 \to 2$	**RR**
$1 \to 2 \to 3$	**RR**
$1 \to 2 \to 4$	**RR**
$2 \to 4 \to 1$	**RR**
$4 \to 1 \to 2$	**RR**

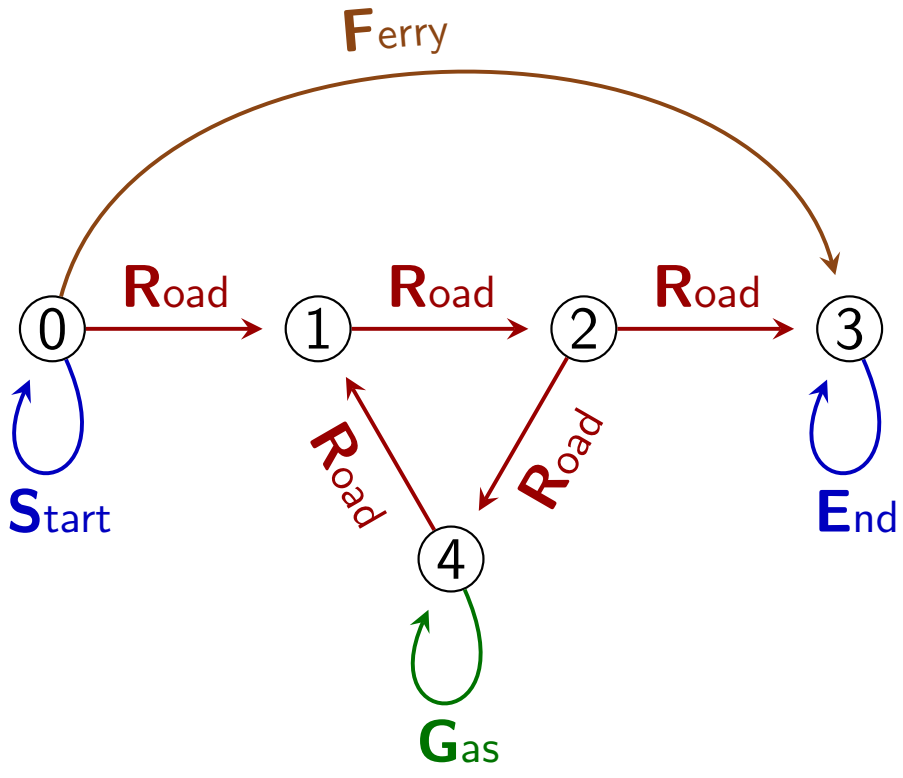


Concatenation

$Q_4 = \mathbf{R} \cdot \mathbf{R}$

$L(Q_4) = \{\mathbf{RR}\}$

Match for Q_4	Label
$0 \to 1 \to 2$	**RR**
$1 \to 2 \to 3$	**RR**
$1 \to 2 \to 4$	**RR**
$2 \to 4 \to 1$	**RR**
$4 \to 1 \to 2$	**RR**



Matches for $Q_5 = \mathbf{S} \cdot \mathbf{R} \cdot \mathbf{R} \cdot \mathbf{R}$

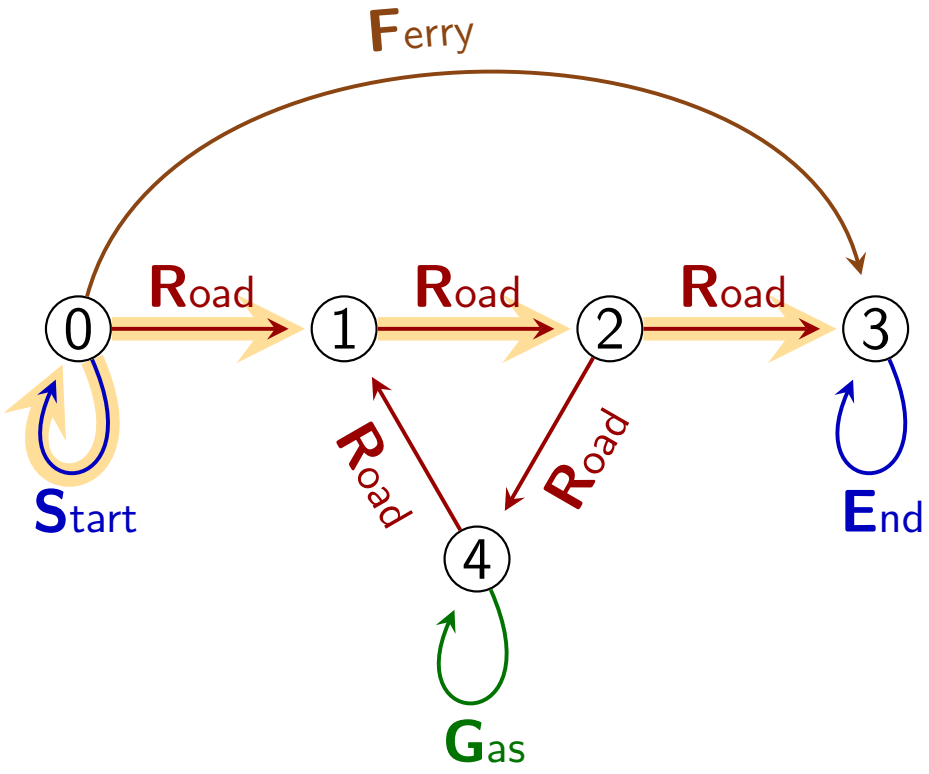
$L(Q_5) = \{\mathbf{SRRR}\}$

Concatenation

$Q_4 = \mathbf{R} \cdot \mathbf{R}$

$L(Q_4) = \{\mathbf{RR}\}$

Match for Q_4	Label
$0 \to 1 \to 2$	**RR**
$1 \to 2 \to 3$	**RR**
$1 \to 2 \to 4$	**RR**
$2 \to 4 \to 1$	**RR**
$4 \to 1 \to 2$	**RR**



Matches for $Q_5 = \mathbf{S} \cdot \mathbf{R} \cdot \mathbf{R} \cdot \mathbf{R}$

$L(Q_5) = \{\mathbf{SRRR}\}$

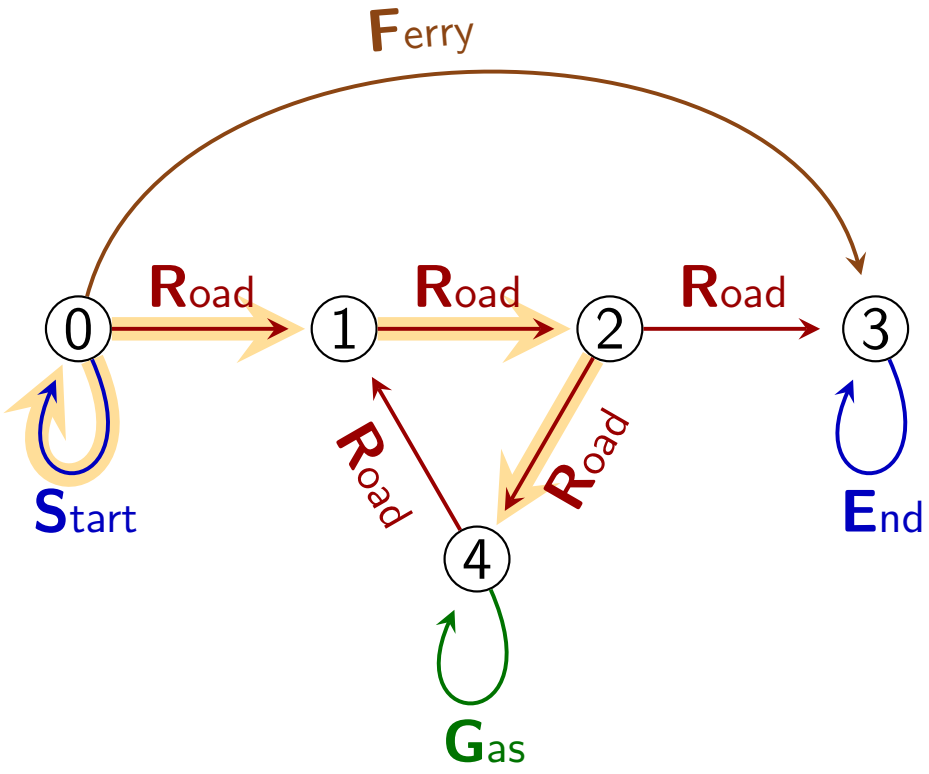
$0 \to 0 \to 1 \to 2 \to 3$	**SRRR**
$0 \to 0 \to 1 \to 2 \to 4$	**SRRR**

Concatenation

$Q_4 = \mathbf{R} \cdot \mathbf{R}$

$L(Q_4) = \{\mathbf{RR}\}$

Match for Q_4	Label
$0 \to 1 \to 2$	**RR**
$1 \to 2 \to 3$	**RR**
$1 \to 2 \to 4$	**RR**
$2 \to 4 \to 1$	**RR**
$4 \to 1 \to 2$	**RR**



Matches for $Q_5 = \mathbf{S} \cdot \mathbf{R} \cdot \mathbf{R} \cdot \mathbf{R}$

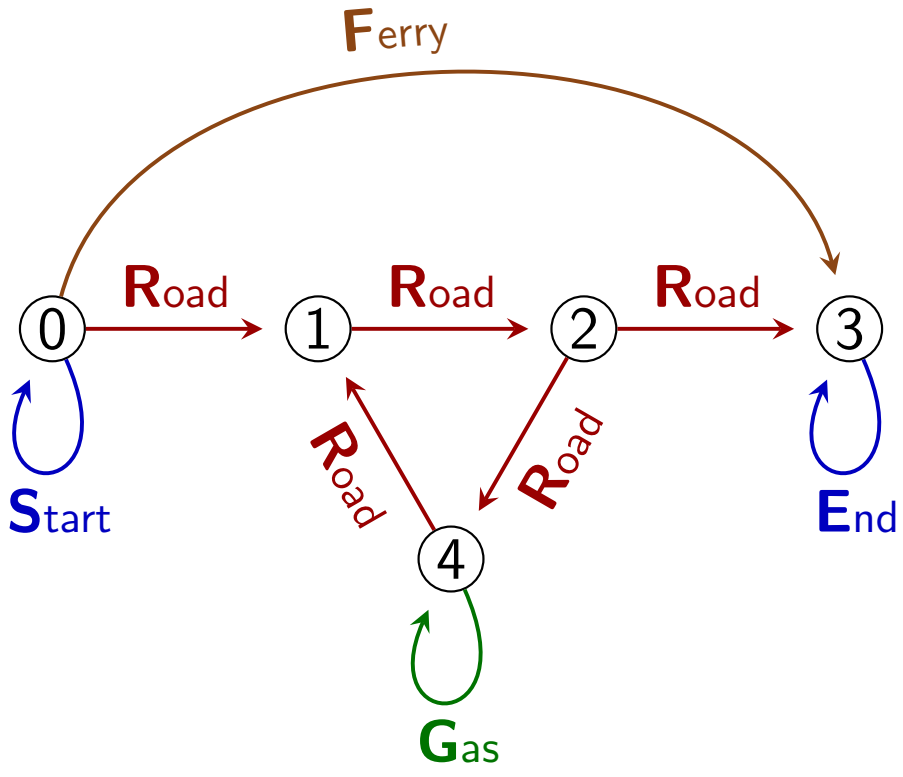
$L(Q_5) = \{\mathbf{SRRR}\}$

$0 \to 0 \to 1 \to 2 \to 3$	**SRRR**
$0 \to 0 \to 1 \to 2 \to 4$	**SRRR**

Concatenation + disjunction

$Q_6 = \mathbf{S} \cdot (\mathbf{R} + \mathbf{F})$

$L(Q_6) = \{\mathbf{SR}, \mathbf{SF}\}$

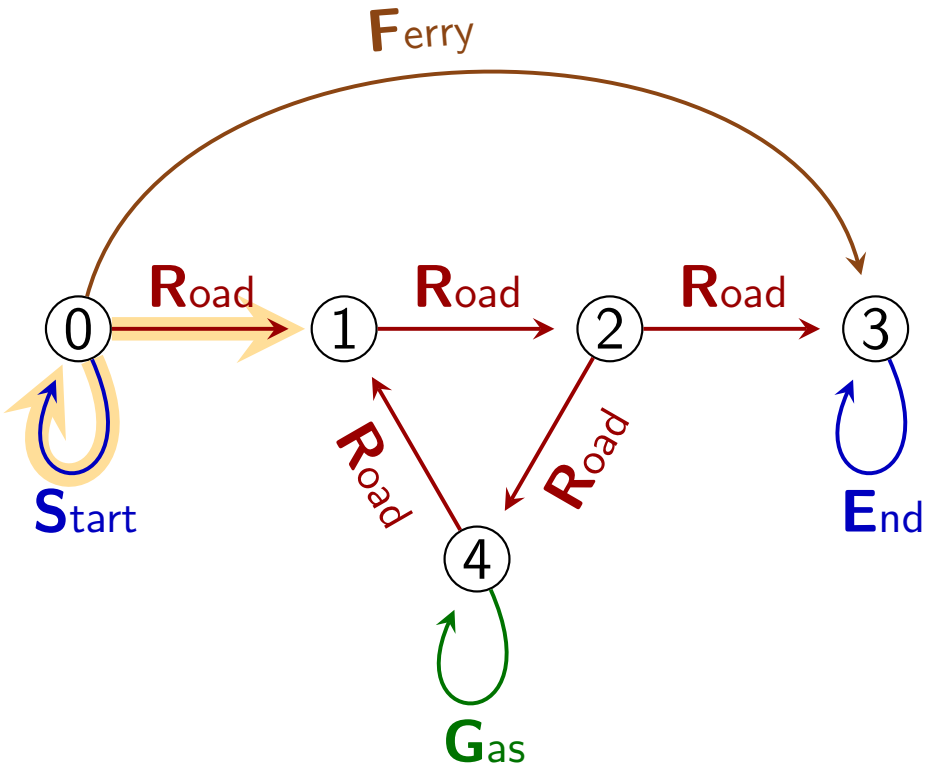


Concatenation + disjunction

$Q_6 = \mathbf{S} \cdot (\mathbf{R} + \mathbf{F})$

$L(Q_6) = \{\mathbf{SR}, \mathbf{SF}\}$

Match for Q_6	Label
$0 \to 0 \to 1$	**SR**
$0 \to 0 \to 3$	**SF**

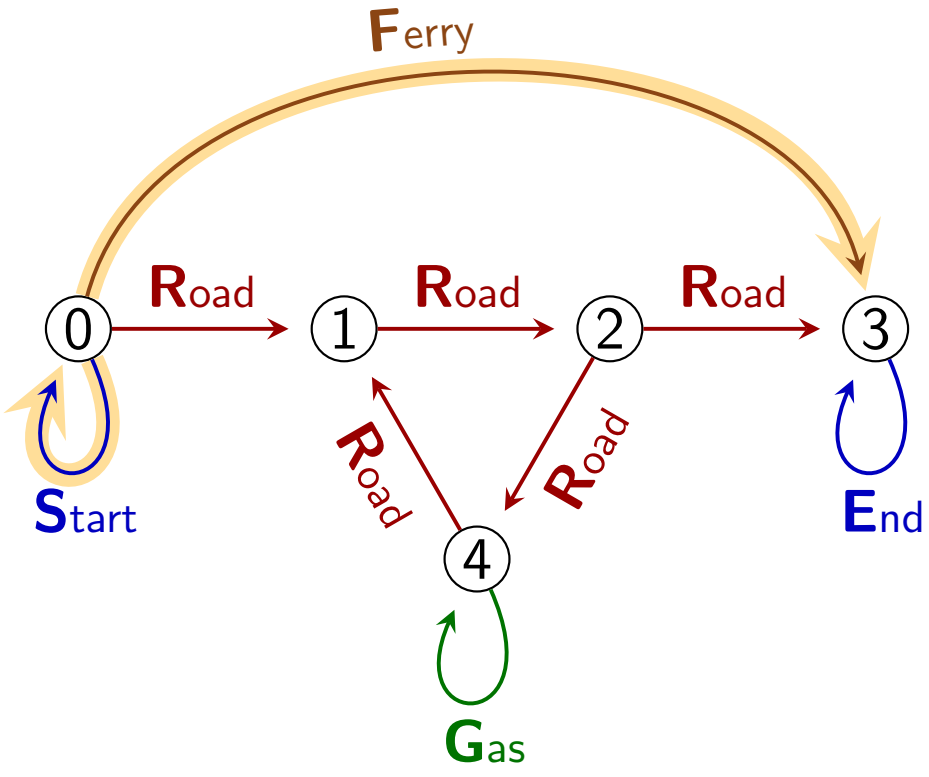


Concatenation + disjunction

$Q_6 = \mathbf{S} \cdot (\mathbf{R} + \mathbf{F})$

$L(Q_6) = \{\mathbf{SR}, \mathbf{SF}\}$

Match for Q_6	Label
$0 \to 0 \to 1$	**SR**
$0 \to 0 \to 3$	**SF**



Concatenation + disjunction

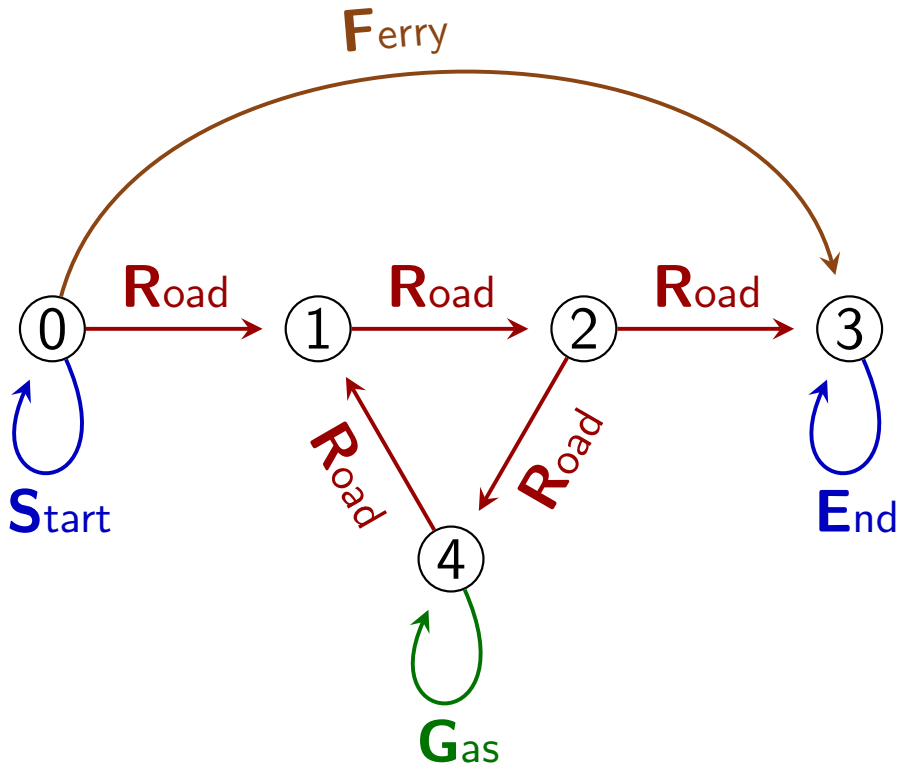
$Q_6 = \mathbf{S} \cdot (\mathbf{R} + \mathbf{F})$

$L(Q_6) = \{\mathbf{SR}, \mathbf{SF}\}$

Match for Q_6	Label
$0 \to 0 \to 1$	**SR**
$0 \to 0 \to 3$	**SF**

$Q_7 = (\mathbf{S} + \mathbf{R})(\mathbf{F} + \mathbf{G})(\mathbf{E} + \mathbf{R})$

$L(Q_7) =$



Concatenation + disjunction

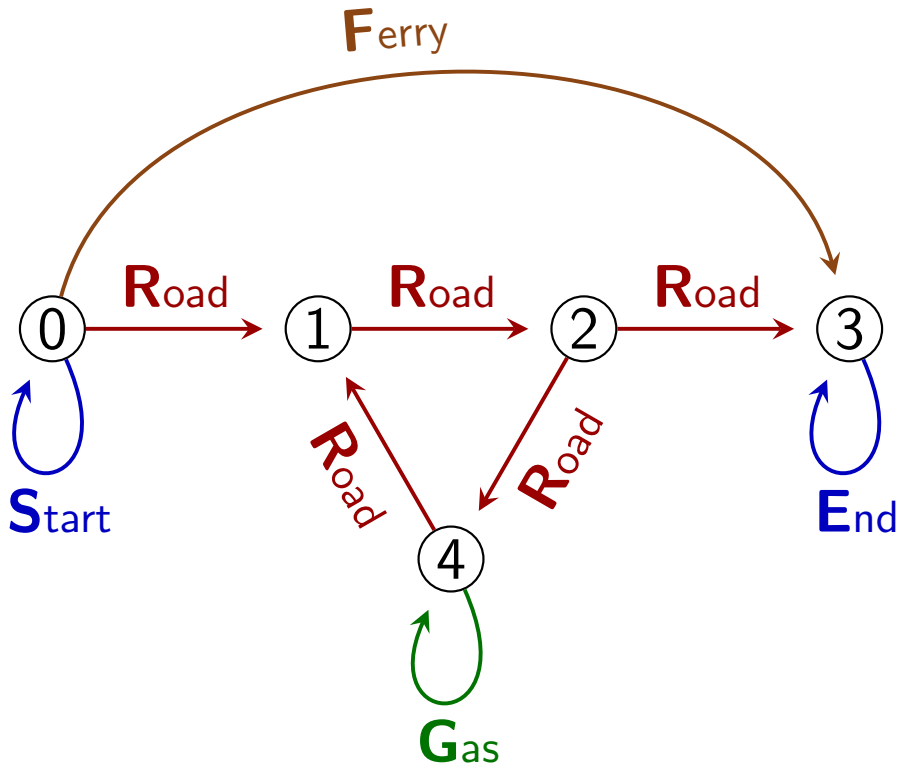
$Q_6 = \mathbf{S} \cdot (\mathbf{R}+\mathbf{F})$

$L(Q_6) = \{\mathbf{SR}, \mathbf{SF}\}$

Match for Q_6	Label
$0 \to 0 \to 1$	**SR**
$0 \to 0 \to 3$	**SF**

$Q_7 = (\mathbf{S}+\mathbf{R})(\mathbf{F}+\mathbf{G})(\mathbf{E}+\mathbf{R})$

$L(Q_7) = \{\mathbf{SFE}, \mathbf{SFR}, \mathbf{SGE}, \mathbf{SGR}, \mathbf{RFE}, \mathbf{RFR}, \mathbf{RGE}, \mathbf{RGR}\}$



Concatenation + disjunction

$Q_6 = \mathbf{S} \cdot (\mathbf{R}+\mathbf{F})$

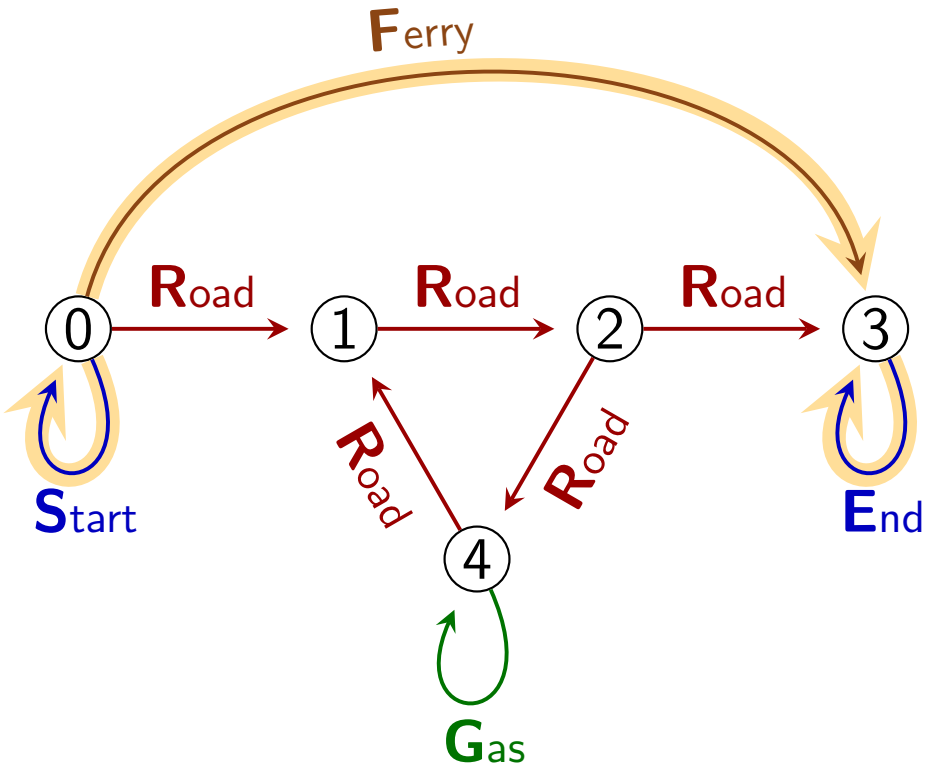
$L(Q_6) = \{\mathbf{SR}, \mathbf{SF}\}$

Match for Q_6	Label
$0 \to 0 \to 1$	**SR**
$0 \to 0 \to 3$	**SF**

$Q_7 = (\mathbf{S}+\mathbf{R})(\mathbf{F}+\mathbf{G})(\mathbf{E}+\mathbf{R})$

$L(Q_7) = \{\mathbf{SFE}, \mathbf{SFR}, \mathbf{SGE}, \mathbf{SGR}, \mathbf{RFE}, \mathbf{RFR}, \mathbf{RGE}, \mathbf{RGR}\}$

Match for Q_7	Label
$0 \to 0 \to 3 \to 3$	**SFE**
$2 \to 4 \to 4 \to 1$	**RGR**



Concatenation + disjunction

$Q_6 = \mathbf{S} \cdot (\mathbf{R} + \mathbf{F})$

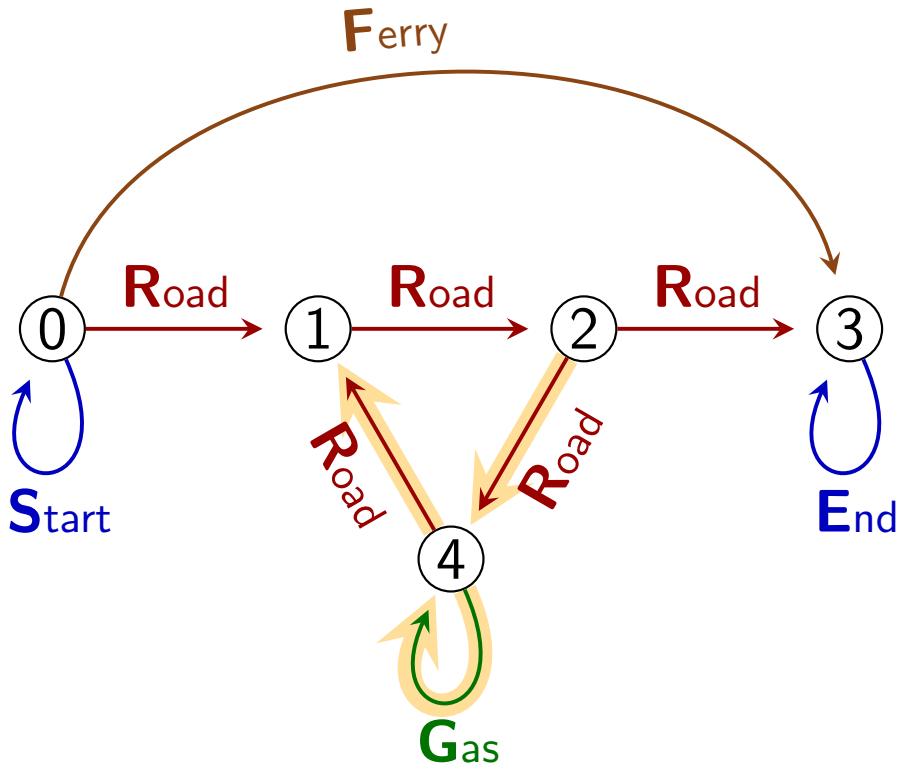
$L(Q_6) = \{\mathbf{SR}, \mathbf{SF}\}$

Match for Q_6	Label
$0 \to 0 \to 1$	**SR**
$0 \to 0 \to 3$	**SF**

$Q_7 = (\mathbf{S} + \mathbf{R})(\mathbf{F} + \mathbf{G})(\mathbf{E} + \mathbf{R})$

$L(Q_7) = \{\mathbf{SFE}, \mathbf{SFR}, \mathbf{SGE}, \mathbf{SGR}, \mathbf{RFE}, \mathbf{RFR}, \mathbf{RGE}, \mathbf{RGR}\}$

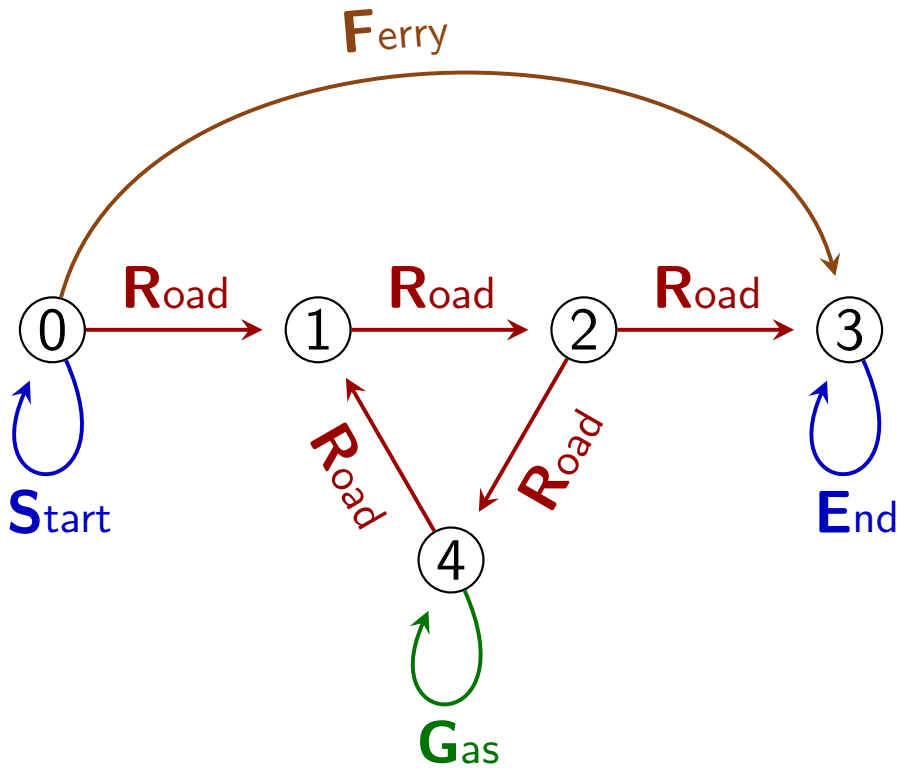
Match for Q_7	Label
$0 \to 0 \to 3 \to 3$	**SFE**
$2 \to 4 \to 4 \to 1$	**RGR**



Kleene star

$Q_8 = \mathbf{R}^*$

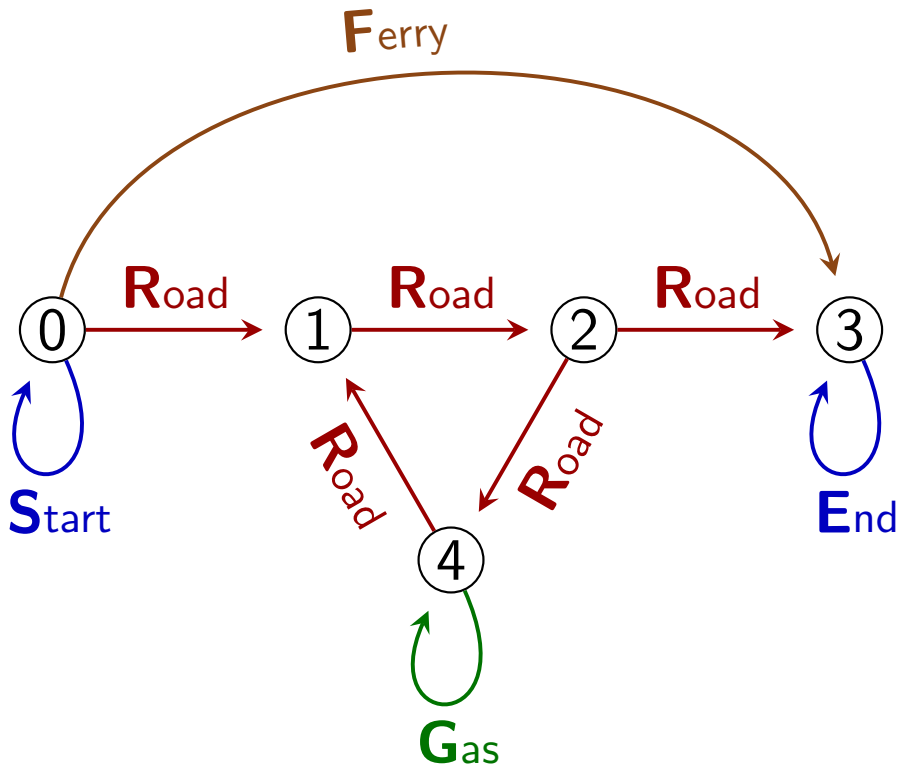
$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$



Kleene star

$Q_8 = \mathbf{R}^*$

$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$



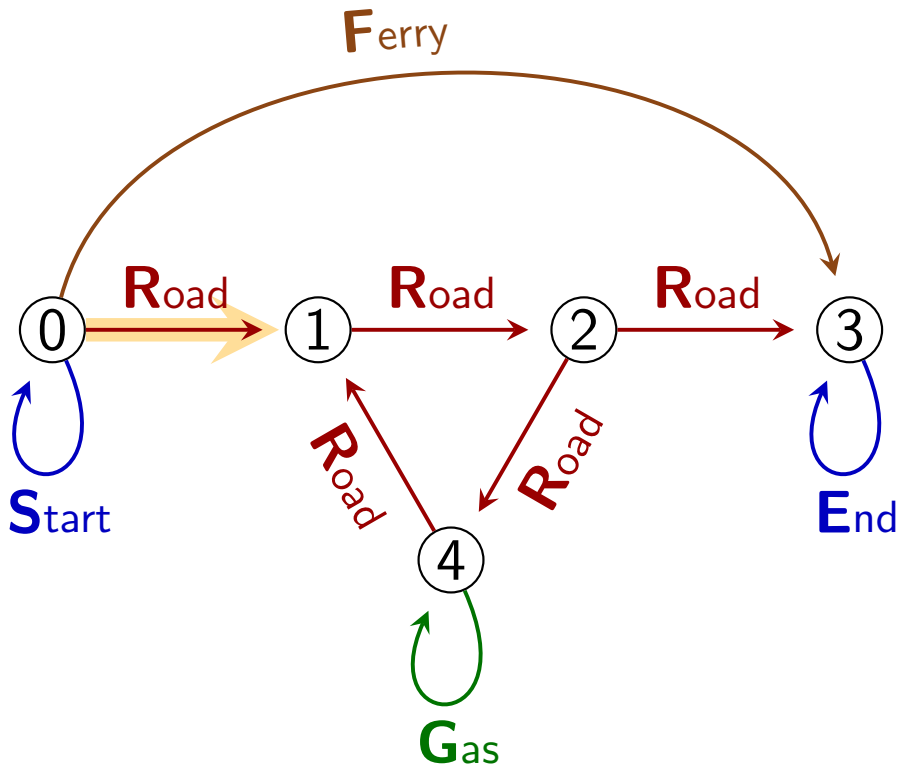
⚠ $L(Q_8)$ is infinite ⚠

Kleene star

$Q_8 = \mathbf{R}^*$

$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$

Match for Q_8	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$\vdots$	
$2 \to 4 \to 1$	**RR**
$\vdots$	
$1 \to 2 \to 4 \to 1$	**RRR**
$\vdots$	
$1 \to 2 \to 4 \to$ $1 \to 2 \to 4 \to 1$	**RRRRRR**
$\vdots$	



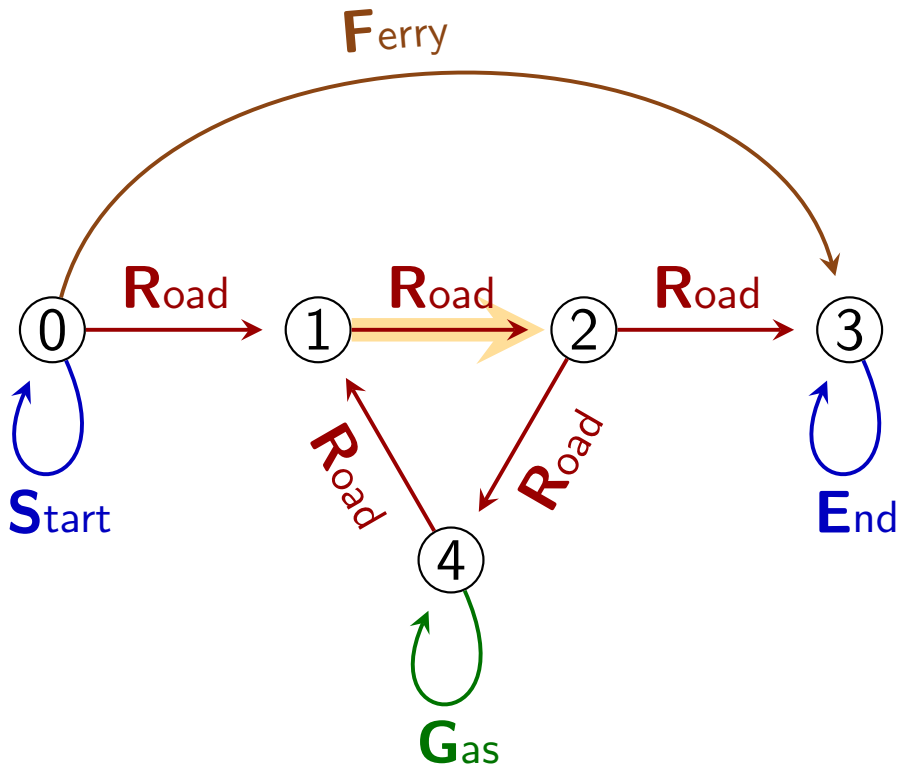
⚠ $L(Q_8)$ is infinite ⚠

Kleene star

$Q_8 = \mathbf{R}^*$

$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$

Match for Q_8	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$\vdots$	
$2 \to 4 \to 1$	**RR**
$\vdots$	
$1 \to 2 \to 4 \to 1$	**RRR**
$\vdots$	
$1 \to 2 \to 4 \to$ $1 \to 2 \to 4 \to 1$	**RRRRRR**
$\vdots$	



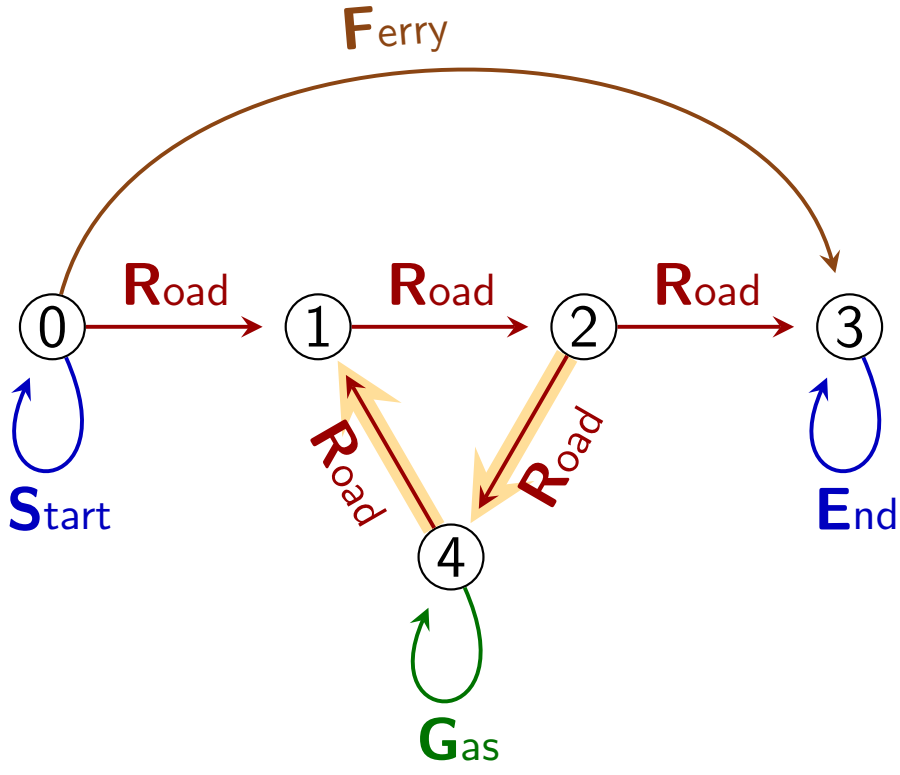
⚠ $L(Q_8)$ is infinite ⚠

Kleene star

$Q_8 = \mathbf{R}^*$

$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$

Match for Q_8	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$\vdots$	
$2 \to 4 \to 1$	**RR**
$\vdots$	
$1 \to 2 \to 4 \to 1$	**RRR**
$\vdots$	
$1 \to 2 \to 4 \to$ $1 \to 2 \to 4 \to 1$	**RRRRRR**
$\vdots$	



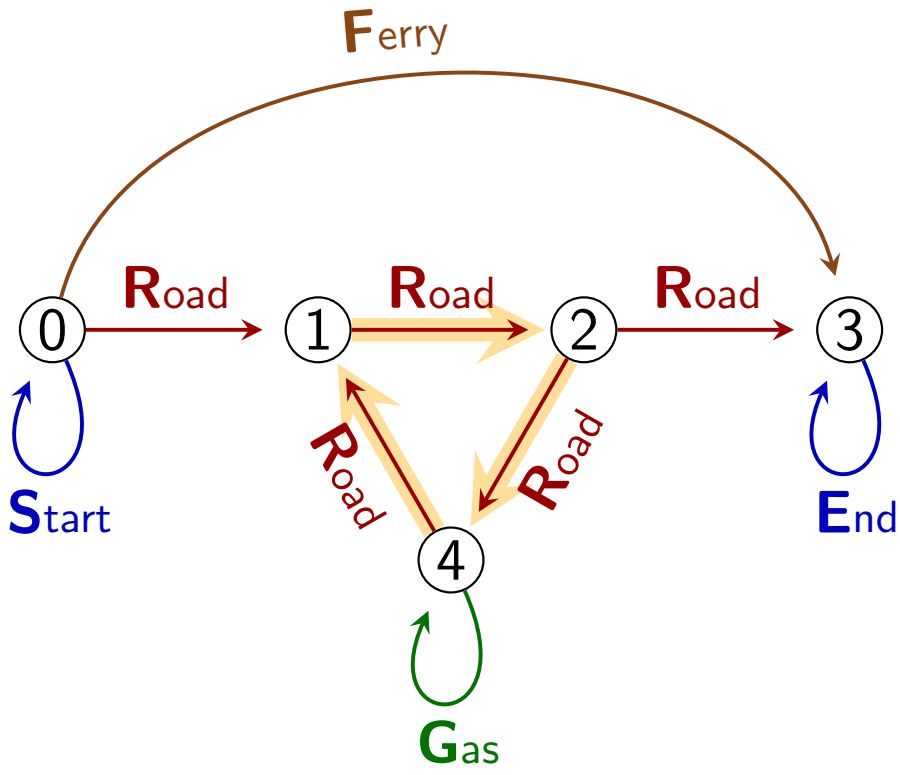
⚠ $L(Q_8)$ is infinite ⚠

Kleene star

$Q_8 = \mathbf{R}^*$

$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$

Match for Q_8	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$\vdots$	
$2 \to 4 \to 1$	**RR**
$\vdots$	
$1 \to 2 \to 4 \to 1$	**RRR**
$\vdots$	
$1 \to 2 \to 4 \to 1 \to 2 \to 4 \to 1$	**RRRRRR**
$\vdots$	



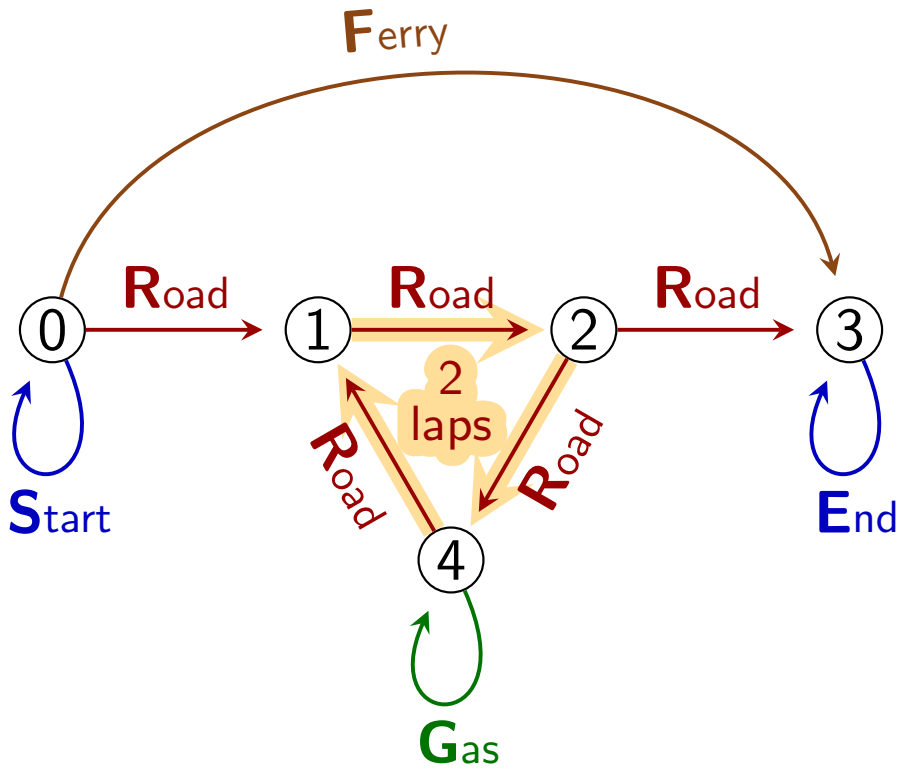
⚠ $L(Q_8)$ is infinite ⚠

Kleene star

$Q_8 = \mathbf{R}^*$

$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$

Match for Q_8	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$\vdots$	
$2 \to 4 \to 1$	**RR**
$\vdots$	
$1 \to 2 \to 4 \to 1$	**RRR**
$\vdots$	
$1 \to 2 \to 4 \to$	**RRRRRR**
$1 \to 2 \to 4 \to 1$	
$\vdots$	



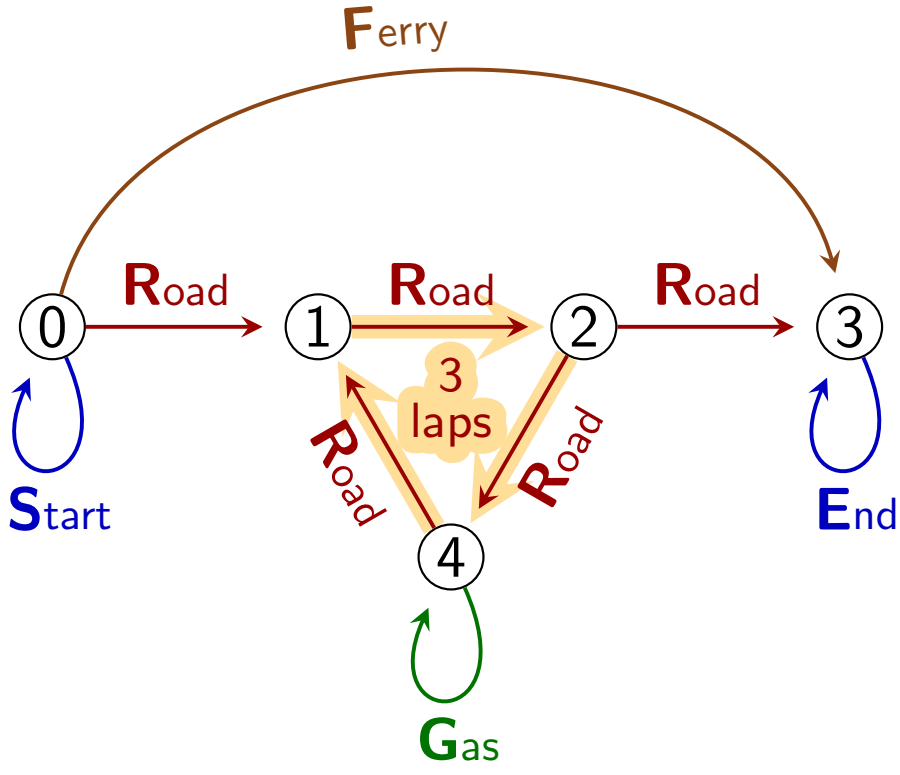
⚠ $L(Q_8)$ is infinite ⚠

Kleene star

$Q_8 = \mathbf{R}^*$

$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$

Match for Q_8	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$\vdots$	
$2 \to 4 \to 1$	**RR**
$\vdots$	
$1 \to 2 \to 4 \to 1$	**RRR**
$\vdots$	
$1 \to 2 \to 4 \to 1 \to 2 \to 4 \to 1$	**RRRRRR**
$\vdots$	



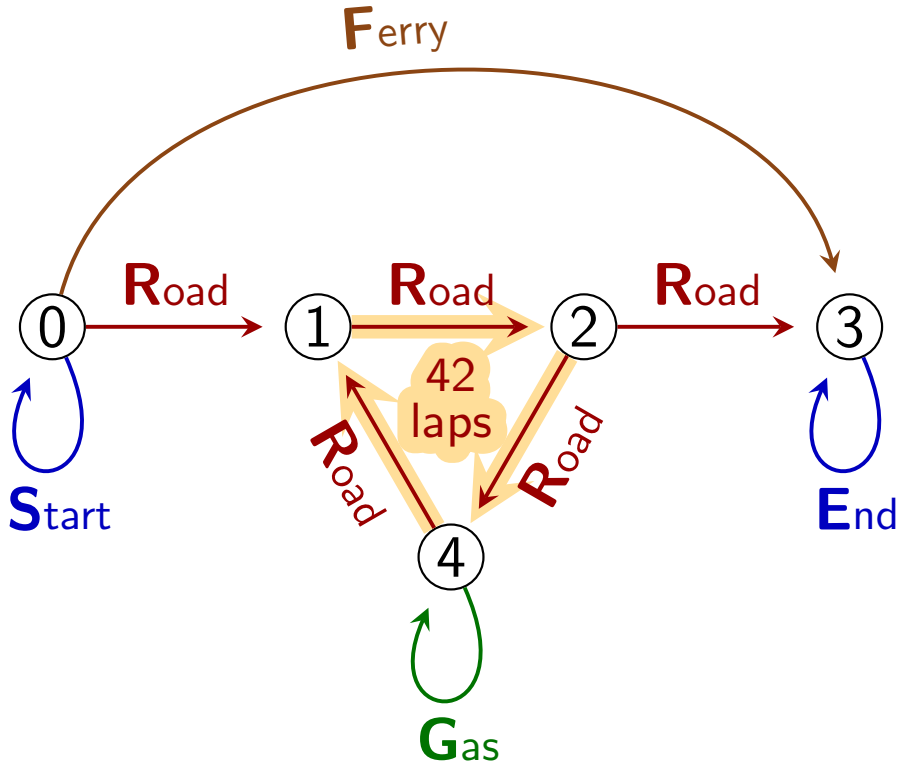
⚠ $L(Q_8)$ is infinite ⚠

Kleene star

$Q_8 = \mathbf{R}^*$

$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$

Match for Q_8	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$\vdots$	
$2 \to 4 \to 1$	**RR**
$\vdots$	
$1 \to 2 \to 4 \to 1$	**RRR**
$\vdots$	
$1 \to 2 \to 4 \to 1 \to 2 \to 4 \to 1$	**RRRRRR**
$\vdots$	



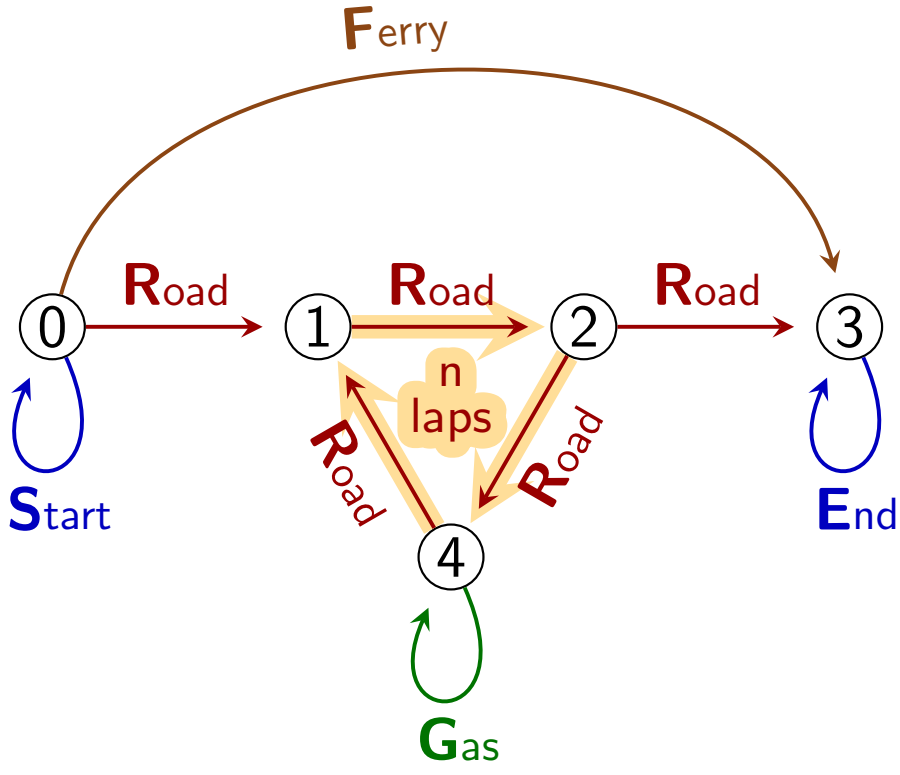
⚠ $L(Q_8)$ is infinite ⚠

Kleene star

$Q_8 = \mathbf{R}^*$

$L(Q_8) = \{\mathbf{R}, \mathbf{RR}, \mathbf{RRR}, \mathbf{RRRR}, \mathbf{RRRRR}, \mathbf{RRRRRR}, \ldots\}$

Match for Q_8	Label
$0 \to 1$	**R**
$1 \to 2$	**R**
$\vdots$	
$2 \to 4 \to 1$	**RR**
$\vdots$	
$1 \to 2 \to 4 \to 1$	**RRR**
$\vdots$	
$1 \to 2 \to 4 \to 1 \to 2 \to 4 \to 1$	**RRRRRR**
$\vdots$	



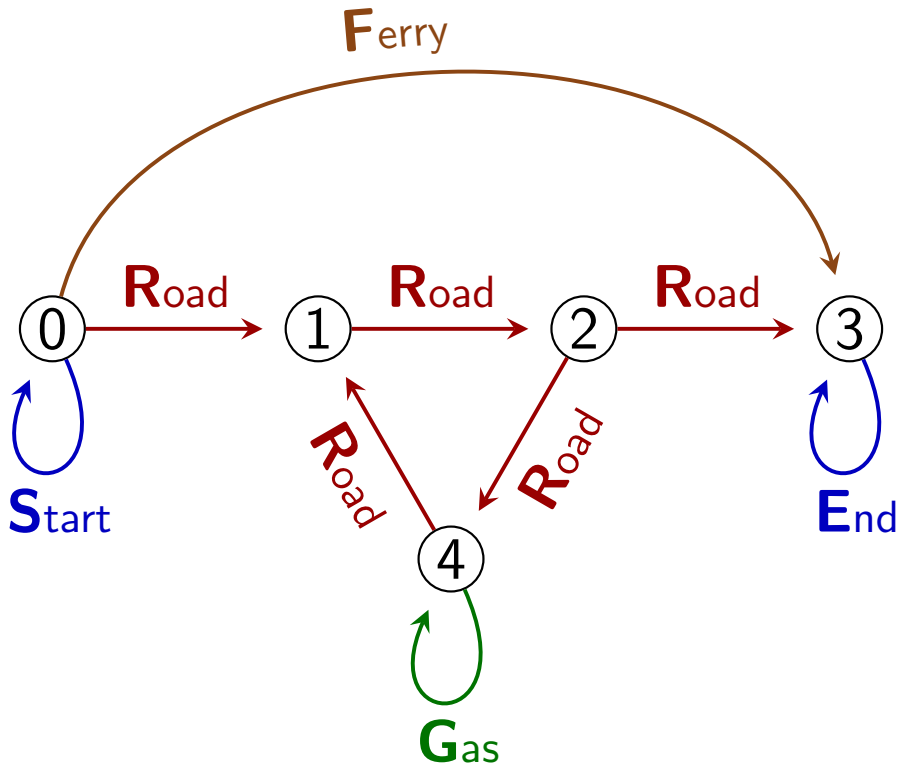
⚠ $L(Q_8)$ is infinite ⚠

⚠ Infinitely many matches ⚠

Computing matches (1)

Exercice

Find a finite representation of the matches to $Q_9 = \mathbf{S}(\mathbf{R} + \mathbf{F})^* \mathbf{E}$.



Computing matches (1)

Exercice

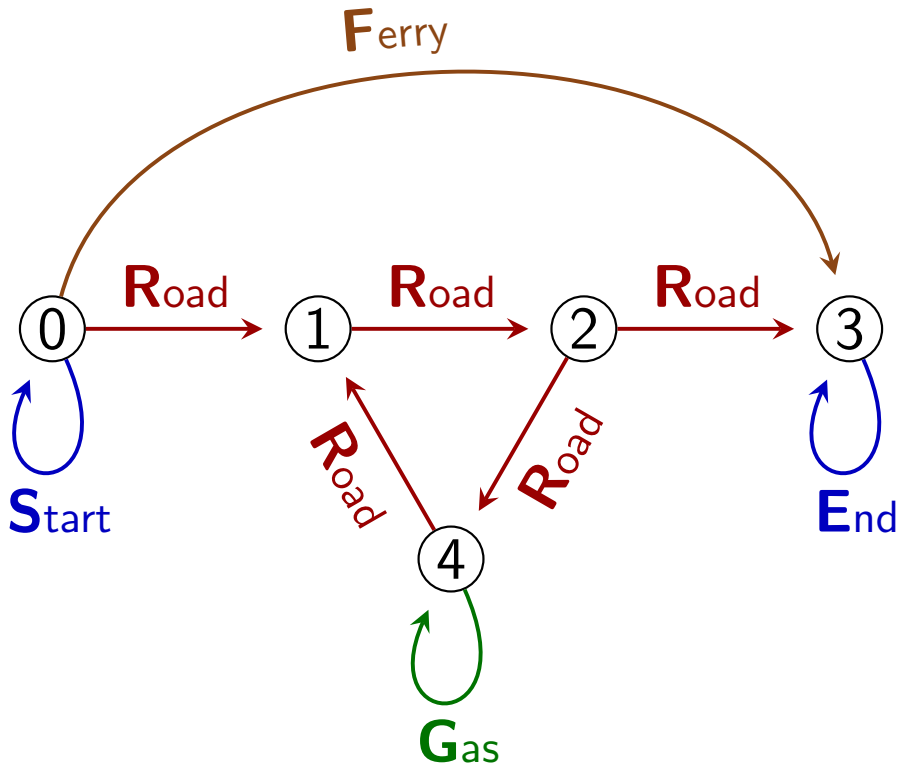
Find a finite representation of the matches to $Q_9 = \mathbf{S}(\mathbf{R} + \mathbf{F})^* \mathbf{E}$.

Answer

$$0 \xrightarrow{\mathbf{S}} 0 \xrightarrow{\mathbf{R}} 1 \xrightarrow{\mathbf{R}} 2$$

$$\left(\xrightarrow{\mathbf{R}} 4 \xrightarrow{\mathbf{R}} 1 \xrightarrow{\mathbf{R}} 2 \right)^*$$

$$\xrightarrow{\mathbf{R}} 3 \xrightarrow{\mathbf{E}} 3$$



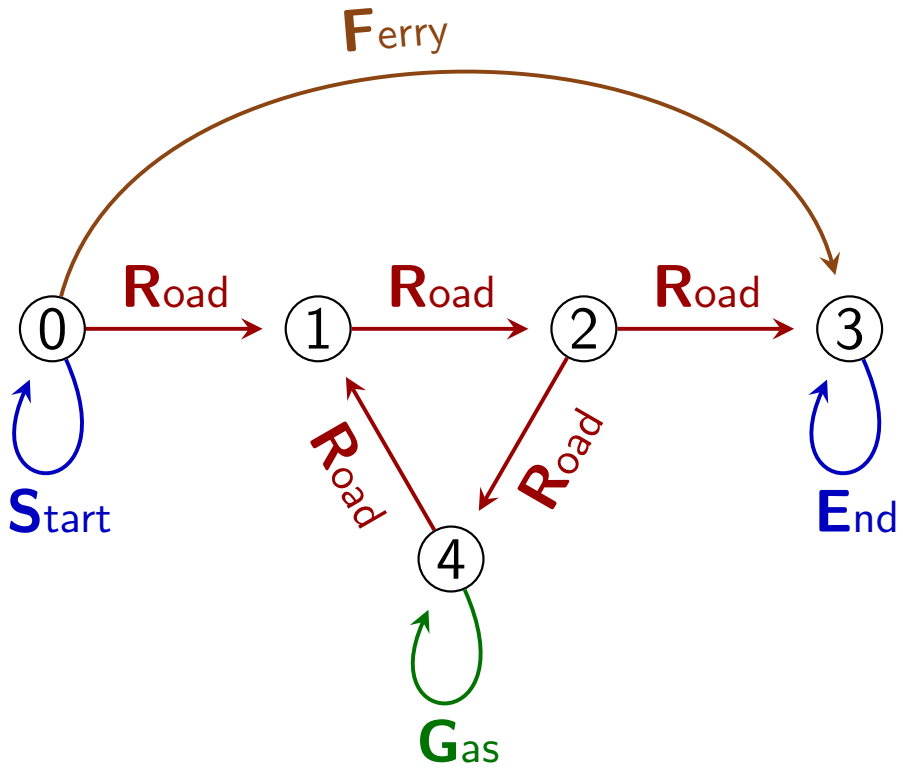
Computing matches (1)

Exercice

Find a finite representation of the matches to $Q_9 = \mathbf{S}(\mathbf{R} + \mathbf{F})^* \mathbf{E}$.

Answer

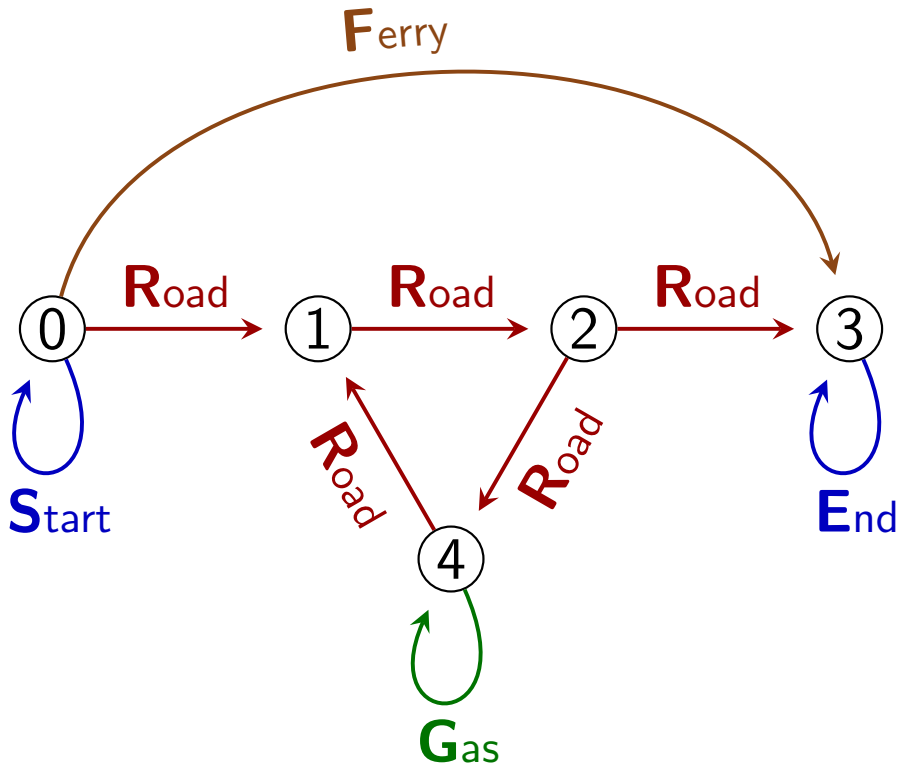
$$\Big(0 \xrightarrow{\mathbf{S}} 0 \xrightarrow{\mathbf{R}} 1 \xrightarrow{\mathbf{R}} 2 \Big(\xrightarrow{\mathbf{R}} 4 \xrightarrow{\mathbf{R}} 1 \xrightarrow{\mathbf{R}} 2\Big)^* \xrightarrow{\mathbf{R}} 3 \xrightarrow{\mathbf{E}} 3\Big) + (0 \xrightarrow{\mathbf{S}} 0 \xrightarrow{\mathbf{F}} 3 \xrightarrow{\mathbf{E}} 3)$$



Computing matches (2)

Exercice

Find a finite repr. of the matches to $Q_{10} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$.



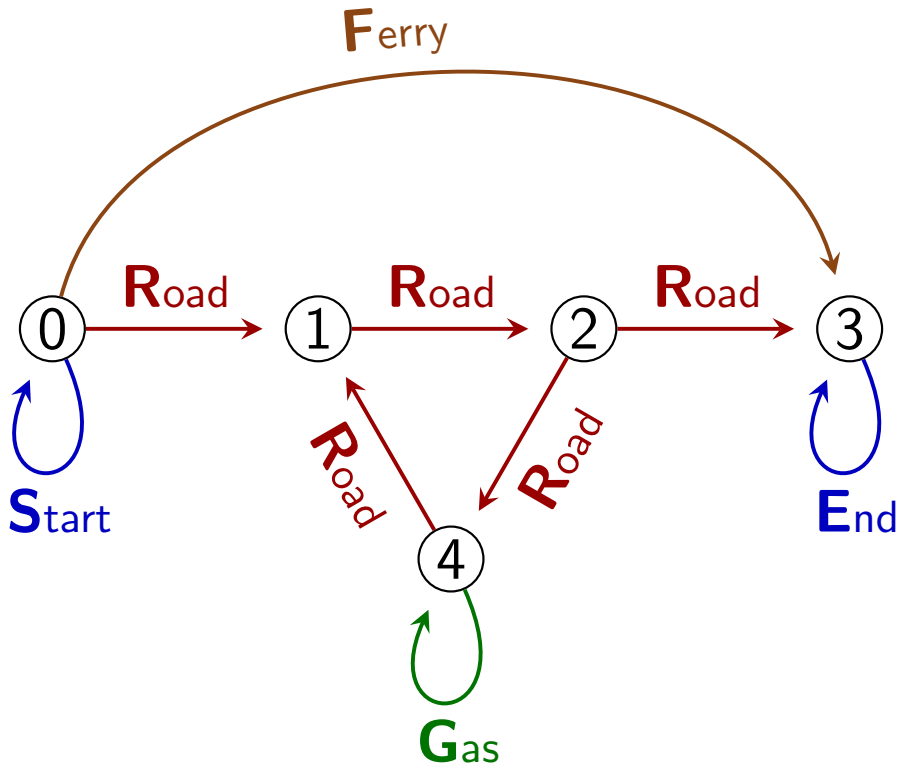
Computing matches (2)

Exercice

Find a finite repr. of the matches to $Q_{10} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$.

Answer

$$0 \xrightarrow{\mathbf{S}} 0 \xrightarrow{\mathbf{R}} 1 \xrightarrow{\mathbf{R}} 2 \xrightarrow{\mathbf{R}} 4 \left(\xrightarrow{\mathbf{R}} 1 \xrightarrow{\mathbf{R}} 2 \xrightarrow{\mathbf{R}} 4\right)^* \xrightarrow{\mathbf{G}} 4 \left(\xrightarrow{\mathbf{R}} 1 \xrightarrow{\mathbf{R}} 2 \xrightarrow{\mathbf{R}} 4\right)^* \xrightarrow{\mathbf{R}} 1 \xrightarrow{\mathbf{R}} 2 \xrightarrow{\mathbf{R}} 3 \xrightarrow{\mathbf{E}} 3$$



Any idea an how to compute matches in general?

Regexps may be transformed into a finite automaton

For instance: Glushkov Construction (aka. position automaton, Berry-Sethi)

Input Regexp Q
Output Nondeterministic Automaton $\mathcal{A}$

- $L(\mathcal{A}) = L(Q)$
- $\mathcal{A}$ is small: $O(\text{size}(Q))$ states
- $\mathcal{A}$ is computed efficiently: $O(\text{size}(Q)^2)$

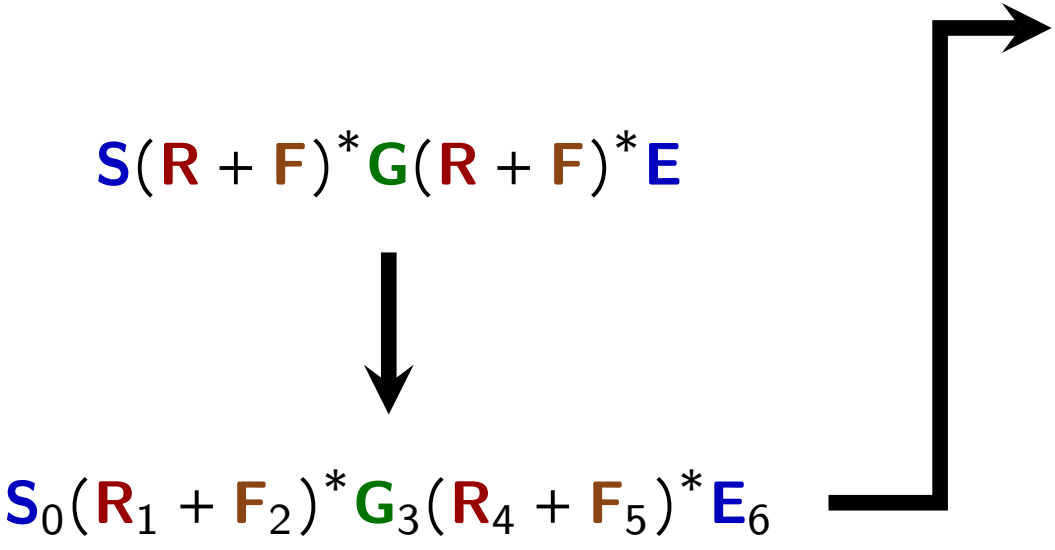
$$\mathbf{S}(\mathbf{R} + \mathbf{F})^* \mathbf{G} (\mathbf{R} + \mathbf{F})^* \mathbf{E}$$

Regexps may be transformed into a finite automaton

For instance: Glushkov Construction (aka. position automaton, Berry-Sethi)

Input Regexp Q
Output Nondeterministic Automaton $\mathcal{A}$

- $L(\mathcal{A}) = L(Q)$
- $\mathcal{A}$ is small: $O(\text{size}(Q))$ states
- $\mathcal{A}$ is computed efficiently: $O(\text{size}(Q)^2)$



Regexps may be transformed into a finite automaton

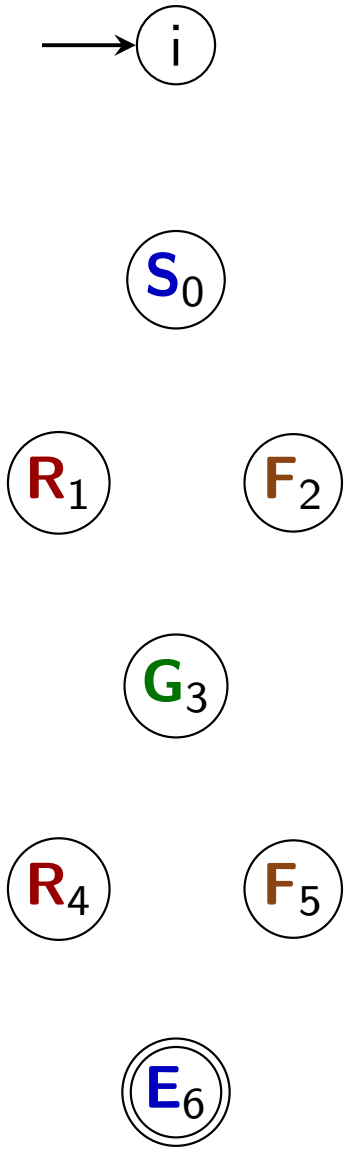
For instance: Glushkov Construction (aka. position automaton, Berry-Sethi)

Input Regexp Q
Output Nondeterministic Automaton $\mathcal{A}$

- $L(\mathcal{A}) = L(Q)$
- $\mathcal{A}$ is small: $O(\text{size}(Q))$ states
- $\mathcal{A}$ is computed efficiently: $O(\text{size}(Q)^2)$

$$\mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$$

$$\mathbf{S}_0(\mathbf{R}_1+\mathbf{F}_2)^*\mathbf{G}_3(\mathbf{R}_4+\mathbf{F}_5)^*\mathbf{E}_6$$



Regexps may be transformed into a finite automaton

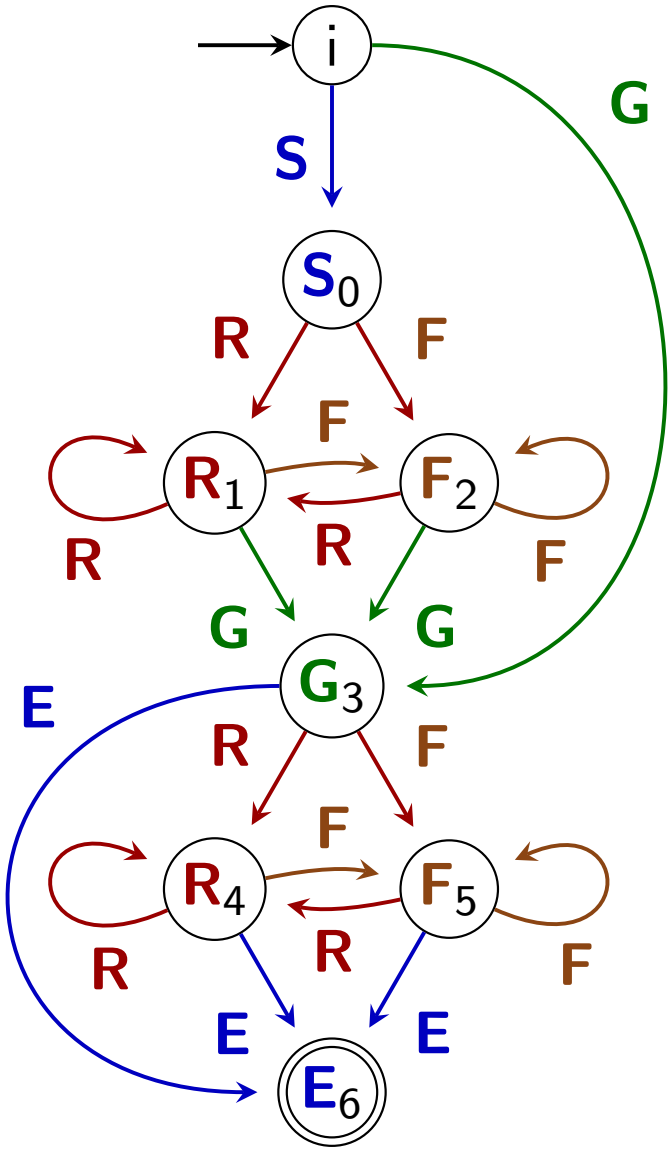
For instance: Glushkov Construction (aka. position automaton, Berry-Sethi)

Input Regexp Q
Output Nondeterministic Automaton $\mathcal{A}$

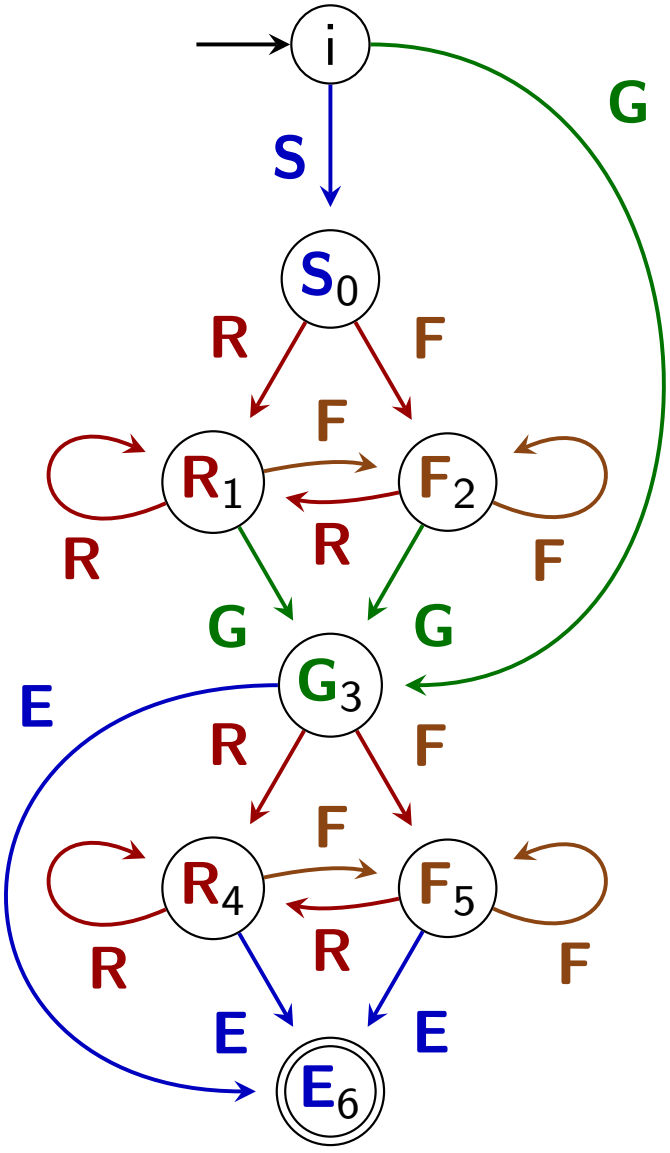
- $L(\mathcal{A}) = L(Q)$
- $\mathcal{A}$ is small: $O(\text{size}(Q))$ states
- $\mathcal{A}$ is computed efficiently: $O(\text{size}(Q)^2)$

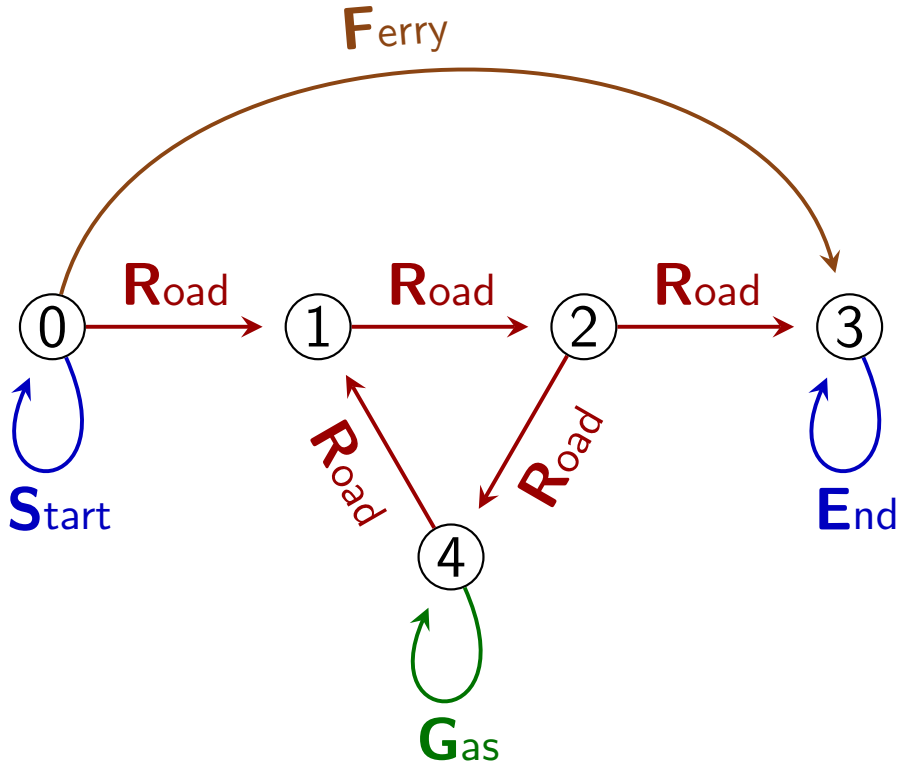
$$\mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$$

$$\mathbf{S}_0(\mathbf{R}_1+\mathbf{F}_2)^*\mathbf{G}_3(\mathbf{R}_4+\mathbf{F}_5)^*\mathbf{E}_6$$



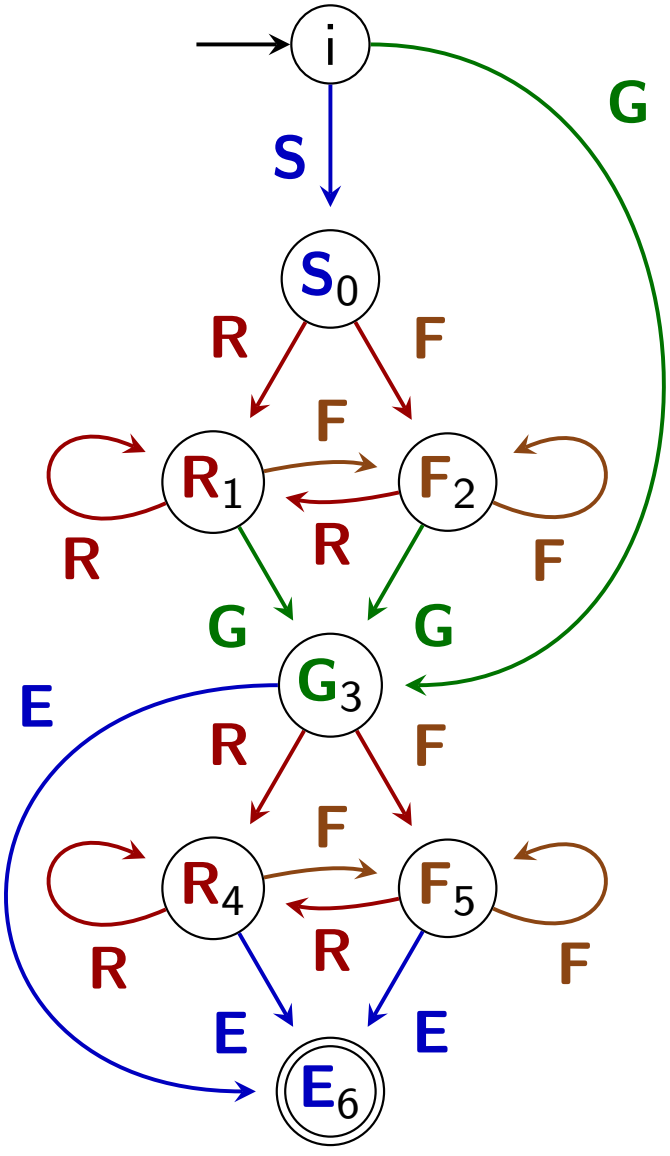
A graph is essentially an automaton

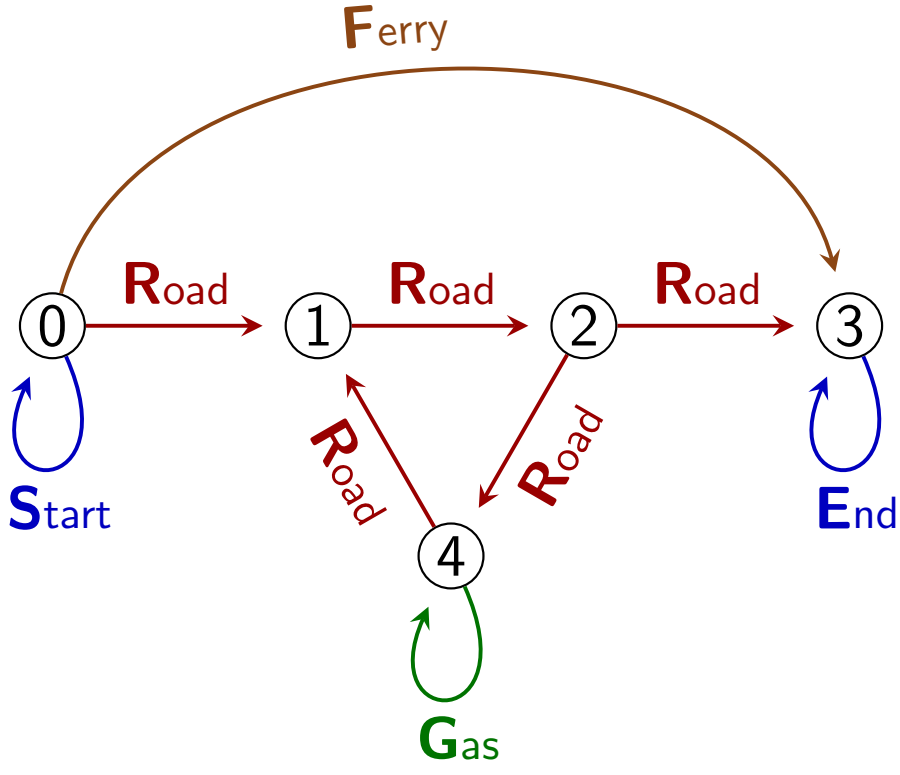




A graph is essentially an automaton

Exercice: compute the product graph×query

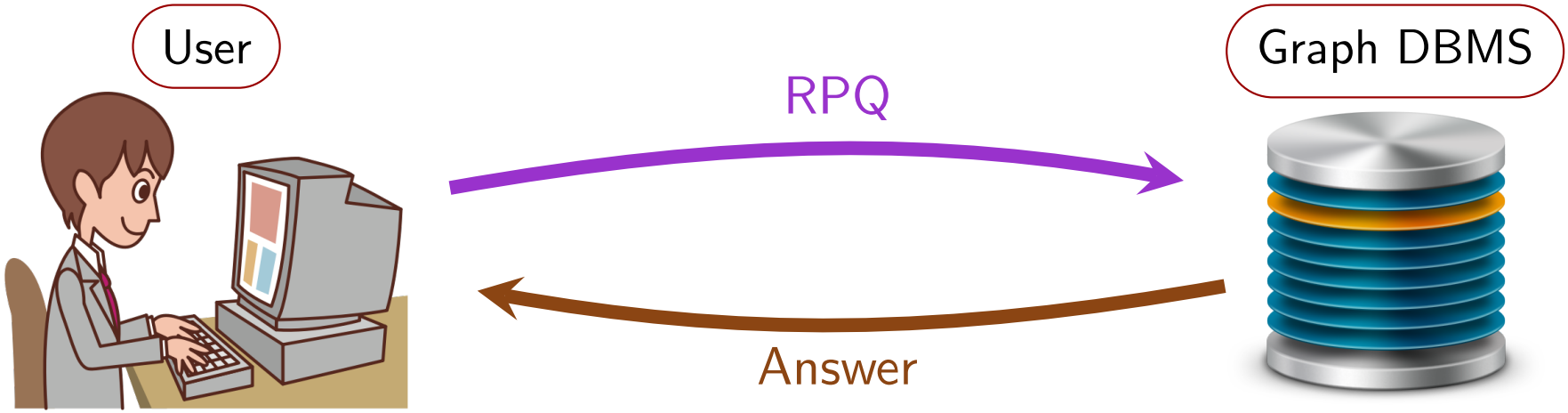




Part I: Theoretical foundations

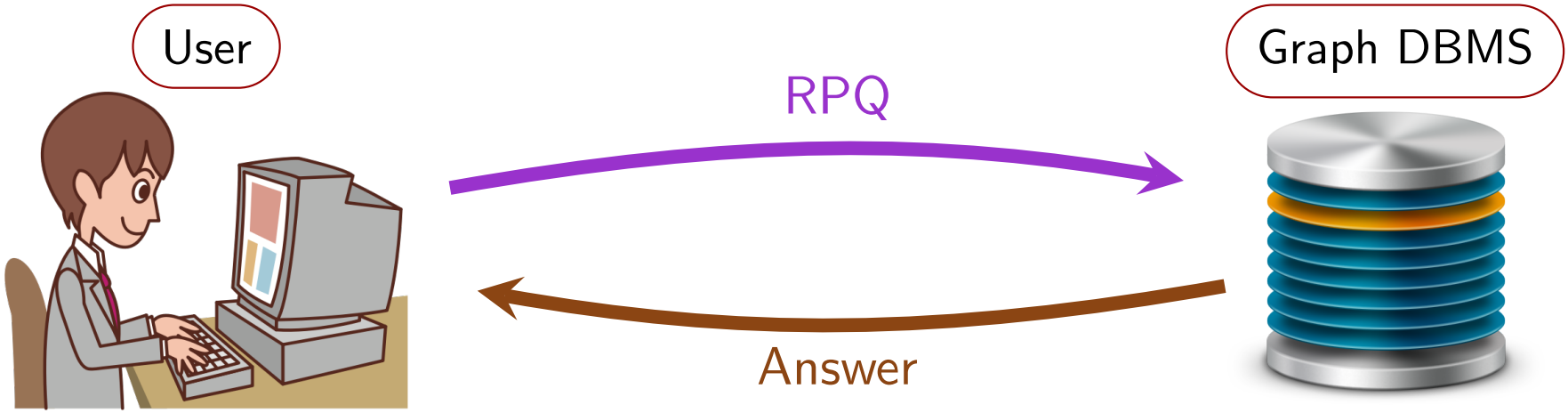
3. RPQ semantics

Computing a finite answer



⚠ **Infinitely** many matches but the user expects **finite** answer ⚠

Computing a finite answer



⚠ **Infinitely** many matches but the user expects **finite** answer ⚠

- A **RPQ semantics** = a way to interpret RPQs
- The semantics defines the **correct answer**
  ⇒ The same query has different answers under different semantics
- Goal of an RPQ semantics: ensure the answer to be **finite**, while remaining **meaningful** and **easy to compute**.

Endpoint semantics (1)

Used by SparQL (RDF) and arguably GQL with keyword `ANY WALK`

Endpoint semantics (1)

Used by SparQL (RDF) and arguably GQL with keyword `ANY WALK`

Principles

- Returns a set of pairs of vertices (and not walks)
- Precisely, returns the endpoints (first and last vertex) of the matches

Example

Matching walks	Projection on endpoints
$1 \to 0 \to 2 \to 2 \to 3$	(1,3)
$2 \to 2$	(2,2)
$0 \to 0 \to 2 \to 3 \to 0 \to 3$	(0,3)
$1 \to 0 \to 3$	(1,3)

Full answer is: $\{(1,3),(2,2),(0,3)\}$

Endpoint semantics (1)

Used by SparQL (RDF) and arguably GQL with keyword `ANY WALK`

Principles

- Returns a set of pairs of vertices (and not walks)
- Precisely, returns the endpoints (first and last vertex) of the matches

Example

Matching walks	Projection on endpoints
$1 \to 0 \to 2 \to 2 \to 3$	(1,3)
$2 \to 2$	(2,2)
$0 \to 0 \to 2 \to 3 \to 0 \to 3$	(0,3)
$1 \to 0 \to 3$	(1,3)

Full answer is: $\{(1,3),(2,2),(0,3)\}$

Endpoint semantics (1)

Used by SparQL (RDF) and arguably GQL with keyword `ANY WALK`

Principles

- Returns a set of pairs of vertices (and not walks)
- Precisely, returns the endpoints (first and last vertex) of the matches

Example

Matching walks	Projection on endpoints
$1 \to 0 \to 2 \to 2 \to 3$	(1,3)
$2 \to 2$	(2,2)
$0 \to 0 \to 2 \to 3 \to 0 \to 3$	(0,3)
$1 \to 0 \to 3$	(1,3)

Full answer is: $\{(1,3), (2,2), (0,3)\}$

Endpoint semantics (1)

Used by SparQL (RDF) and arguably GQL with keyword `ANY WALK`

Principles

- Returns a set of pairs of vertices (and not walks)
- Precisely, returns the endpoints (first and last vertex) of the matches

Example

Matching walks	Projection on endpoints
$1 \to 0 \to 2 \to 2 \to 3$	(1,3)
$2 \to 2$	(2,2)
$0 \to 0 \to 2 \to 3 \to 0 \to 3$	(0,3)
$1 \to 0 \to 3$	(1,3)

Full answer is: $\{(1,3),(2,2),(0,3)\}$

Endpoint semantics (1)

Used by SparQL (RDF) and arguably GQL with keyword `ANY WALK`

Principles

- Returns a set of pairs of vertices (and not walks)
- Precisely, returns the endpoints (first and last vertex) of the matches

Example

Matching walks	Projection on endpoints
$1 \to 0 \to 2 \to 2 \to 3$	(1,3)
$2 \to 2$	(2,2)
$0 \to 0 \to 2 \to 3 \to 0 \to 3$	(0,3)
$1 \to 0 \to 3$	(1,3)

Full answer is: $\{(1,3),(2,2),(0,3)\}$

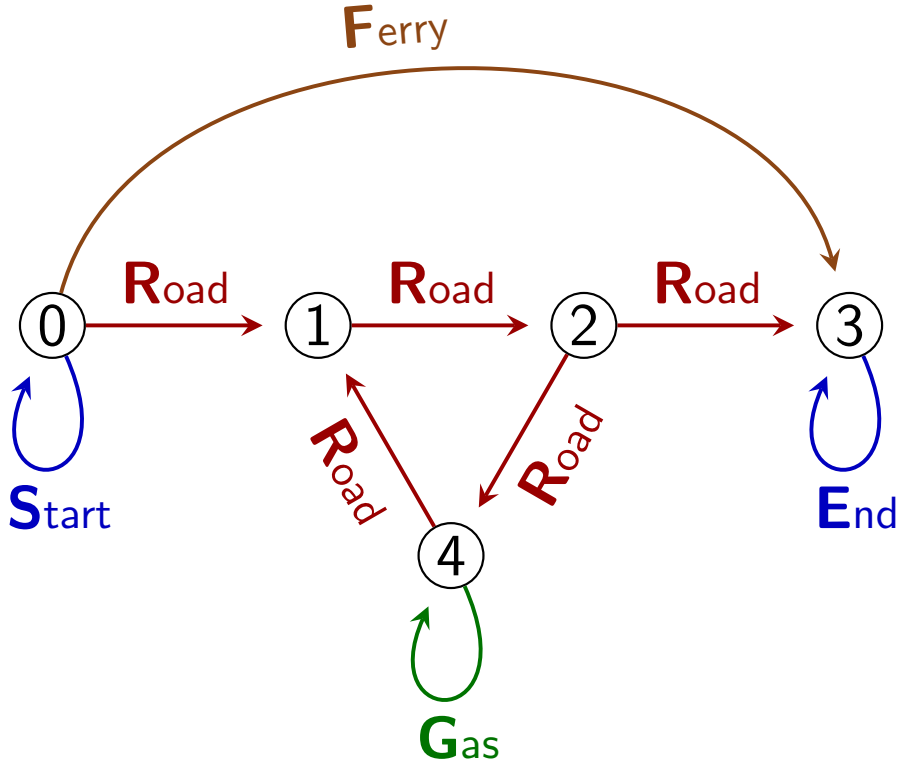
Endpoint semantics (2)

Evaluating a reachability query

$Q_{11} = \mathbf{GR}^*$

Match	Endpoints
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)
⋮	⋮
$4 \to 4 \to 1 \to 2$	
$\to 4 \to 1 \to 2$	
$\to 3$	(4,3)
⋮	⋮

Other matches do not add new pairs to the answer



Answer to Q_{11} under endpoint sem.: $\{(4,4),(4,1),(4,2),(4,3)\}$

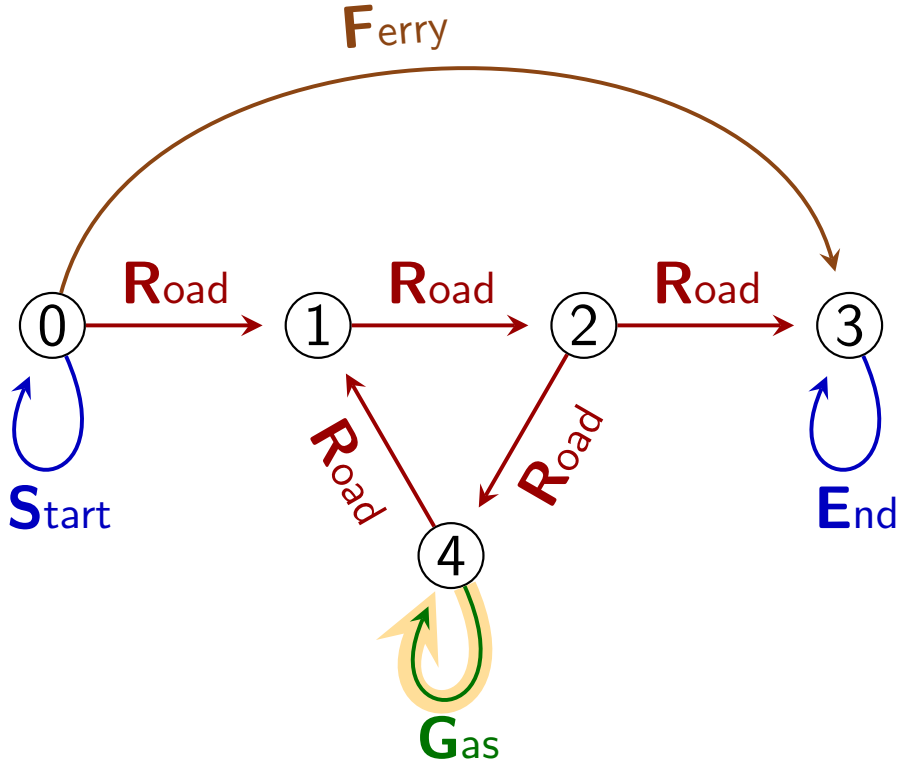
Endpoint semantics (2)

Evaluating a reachability query

$Q_{11} = \mathbf{GR}^*$

Match	Endpoints
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)
⋮	⋮
$4 \to 4 \to 1 \to 2$	
$\to 4 \to 1 \to 2$	
$\to 3$	(4,3)
⋮	⋮

Other matches do not add new pairs to the answer



Answer to Q_{11} under endpoint sem.: $\{(4,4),(4,1),(4,2),(4,3)\}$

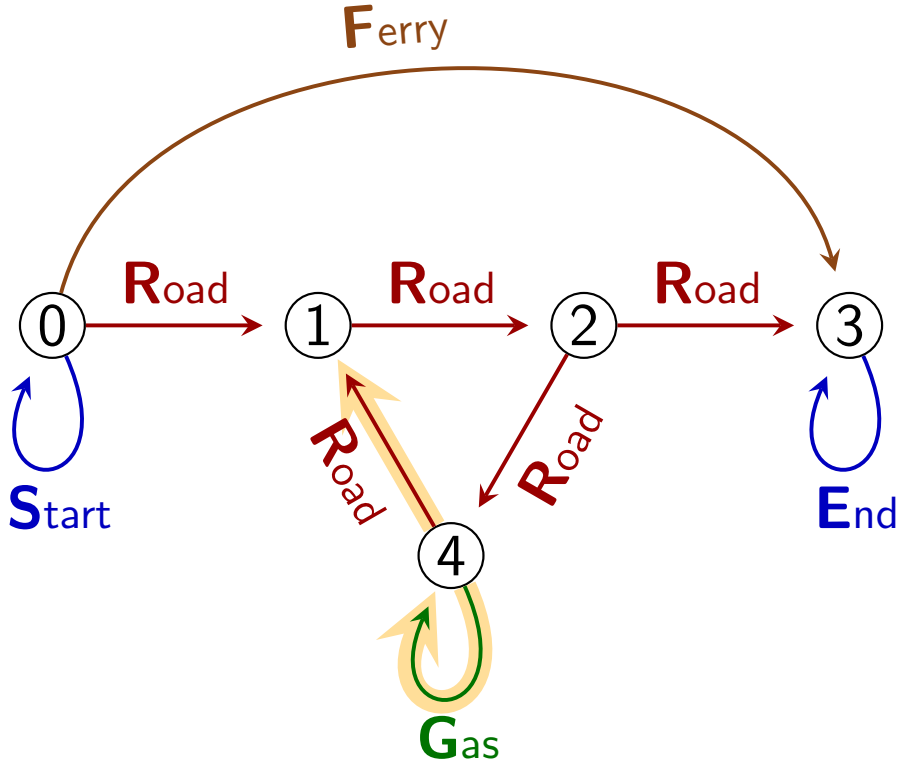
Endpoint semantics (2)

Evaluating a reachability query

$Q_{11} = \mathbf{G}\mathbf{R}^*$

Match	Endpoints
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)
⋮	⋮
$4 \to 4 \to 1 \to 2$	
$\to 4 \to 1 \to 2$	
$\to 3$	(4,3)
⋮	⋮

Other matches do not add new pairs to the answer



Answer to Q_{11} under endpoint sem.: $\{(4,4),(4,1),(4,2),(4,3)\}$

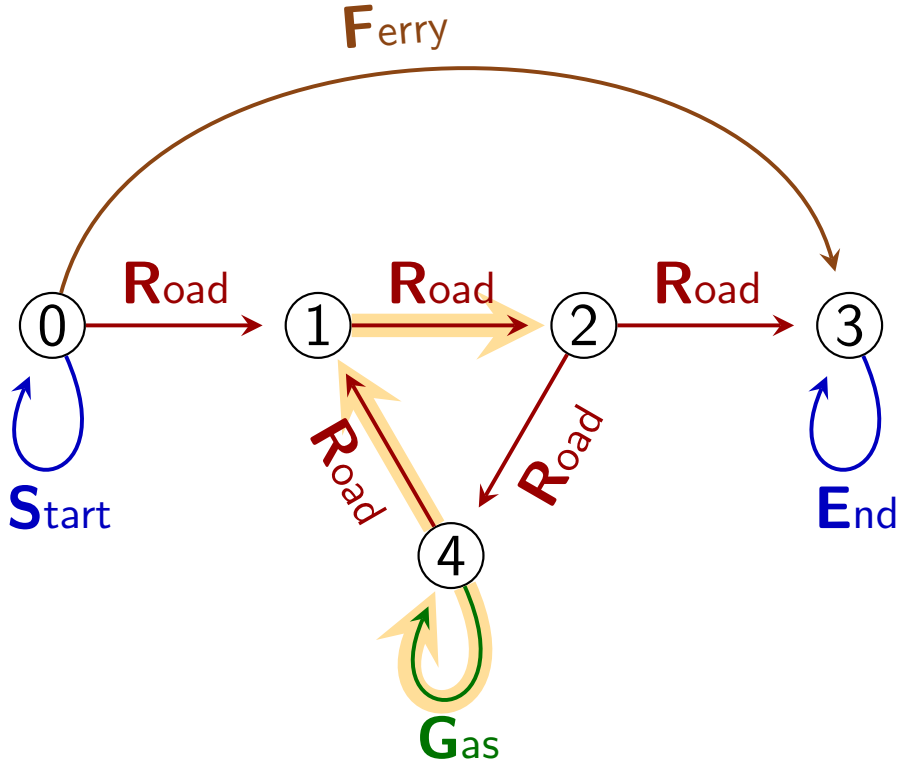
Endpoint semantics (2)

Evaluating a reachability query

$Q_{11} = \mathbf{G}\mathbf{R}^*$

Match	Endpoints
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)
⋮	⋮
$4 \to 4 \to 1 \to 2$	
$\to 4 \to 1 \to 2$	
$\to 3$	(4,3)
⋮	⋮

Other matches do not add new pairs to the answer



Answer to Q_{11} under endpoint sem.: $\{(4,4),(4,1),(4,2),(4,3)\}$

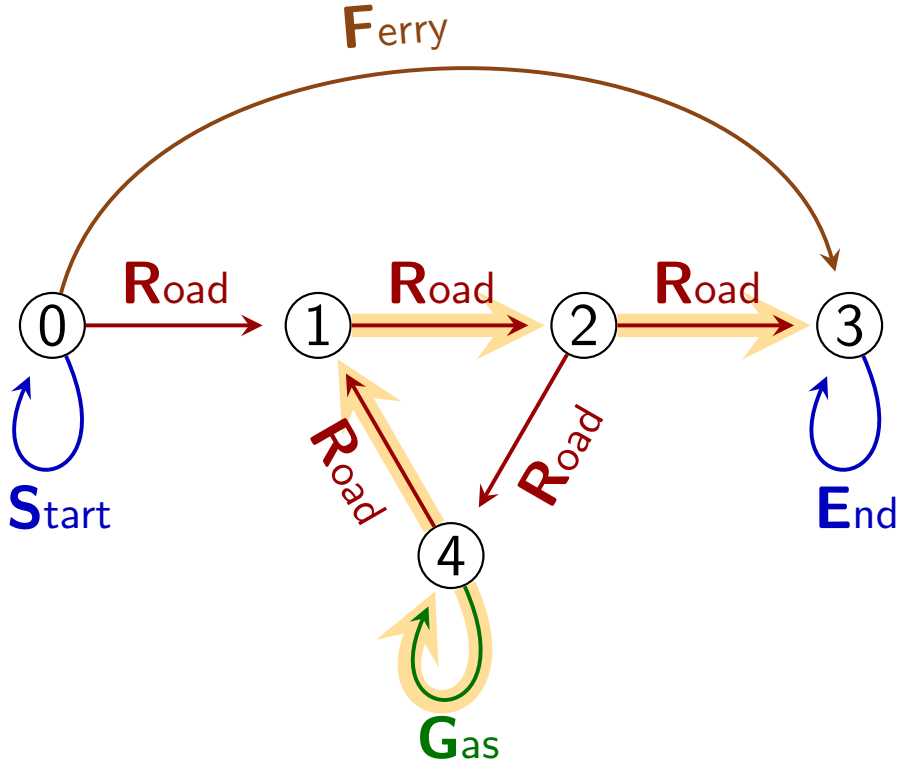
Endpoint semantics (2)

Evaluating a reachability query

$Q_{11} = \mathbf{GR}^*$

Match	Endpoints
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)
⋮	⋮
$4 \to 4 \to 1 \to 2$	
$\to 4 \to 1 \to 2$	
$\to 3$	(4,3)
⋮	⋮

Other matches do not add new pairs to the answer



Answer to Q_{11} under endpoint sem.: $\{(4,4),(4,1),(4,2),(4,3)\}$

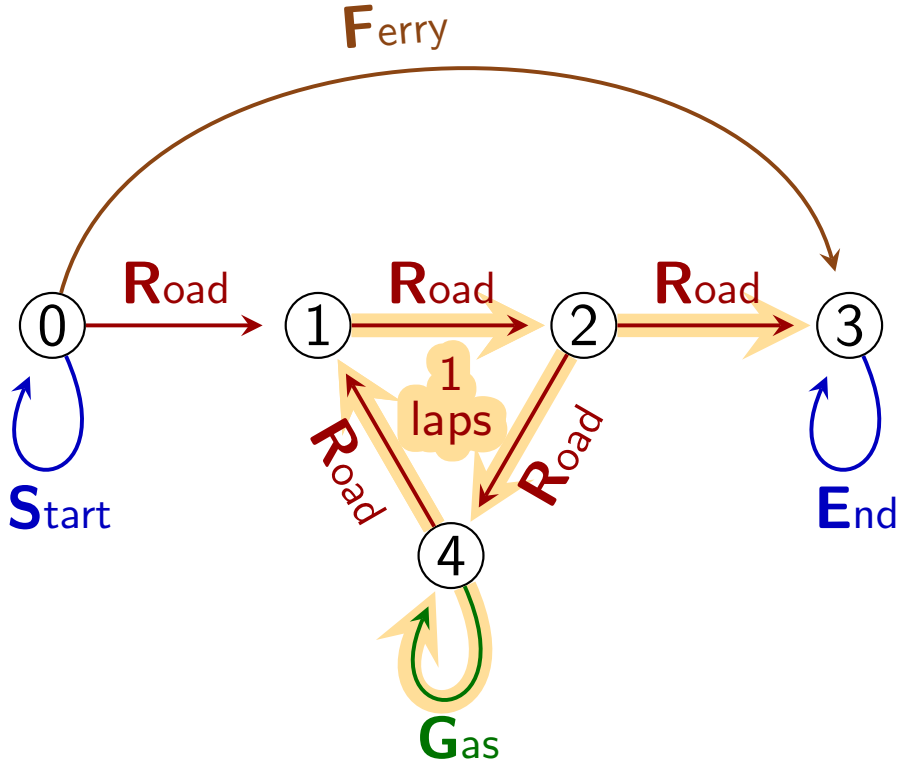
Endpoint semantics (2)

Evaluating a reachability query

$Q_{11} = \mathbf{GR}^*$

Match	Endpoints
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)
⋮	⋮
$4 \to 4 \to 1 \to 2$	
$\to 4 \to 1 \to 2$	
$\to 3$	(4,3)
⋮	⋮

Other matches do not add new pairs to the answer



Answer to Q_{11} under endpoint sem.: $\{(4,4),(4,1),(4,2),(4,3)\}$

Endpoint semantics (3)

Pros and cons

Pros

- Efficient algorithms
- Output is always small
- Well grounded theory

Endpoint semantics (3)

Pros

- Efficient algorithms
- Output is always small
- Well grounded theory

Cons

- Very limited information in the answer
  - User: *"I want to go from Paris to Lyon by car"*
  - Database: *"Yes you can"*

Shortest semantics (1)

Used in GSQL (TigerGraph), PGQL (Oracle) and GQL with `ALL SHORTEST`

Shortest semantics (1)

Used in GSQL (TigerGraph), PGQL (Oracle) and GQL with `ALL SHORTEST`

Principles

- Return walks
- For each endpoints (s, t), return the "best" match from s to t
- Best = shortest = least number of edges

Example

Match	Endpoints	Length
$1 \to 0 \to 2 \to 3$	$(1, 3)$	3
$1 \to 0 \to 2 \to 2 \to 3$	$(1, 3)$	4
$0 \to 2 \to 2 \to 3$	$(0, 3)$	3
$0 \to 2 \to 3$	$(0, 3)$	2
$0 \to 0 \to 3$	$(0, 3)$	2

Full answer: $\{1 \to 0 \to 2 \to 3, \quad 0 \to 2 \to 3, \quad 0 \to 0 \to 3\}$

Shortest semantics (1)

Used in GSQL (TigerGraph), PGQL (Oracle) and GQL with `ALL SHORTEST`

Principles

- Return walks
- For each endpoints (s, t), return the "best" match from s to t
- Best = shortest = least number of edges

Example

Match	Endpoints	Length
$1 \to 0 \to 2 \to 3$	$(1, 3)$	3
$1 \to 0 \to 2 \to 2 \to 3$	$(1, 3)$	4
$0 \to 2 \to 2 \to 3$	$(0, 3)$	3
$0 \to 2 \to 3$	$(0, 3)$	2
$0 \to 0 \to 3$	$(0, 3)$	2

Full answer: $\{1 \to 0 \to 2 \to 3, \quad 0 \to 2 \to 3, \quad 0 \to 0 \to 3\}$

Shortest semantics (1)

Used in GSQL (TigerGraph), PGQL (Oracle) and GQL with `ALL SHORTEST`

Principles

- Return walks
- For each endpoints (s, t), return the "best" match from s to t
- Best = shortest = least number of edges

Example

Match	Endpoints	Length	
$1 \to 0 \to 2 \to 3$	$(1, 3)$	3	Shortest for $(1, 3)$
$1 \to 0 \to 2 \to 2 \to 3$	$(1, 3)$	4	Not shortest for $(1, 3)$
$0 \to 2 \to 2 \to 3$	$(0, 3)$	3	
$0 \to 2 \to 3$	$(0, 3)$	2	
$0 \to 0 \to 3$	$(0, 3)$	2	

Full answer: $\{1 \to 0 \to 2 \to 3, \quad 0 \to 2 \to 3, \quad 0 \to 0 \to 3\}$

Shortest semantics (1)

Used in GSQL (TigerGraph), PGQL (Oracle) and GQL with `ALL SHORTEST`

Principles

- Return walks
- For each endpoints (s, t), return the "best" match from s to t
- Best = shortest = least number of edges

Example

Match	Endpoints	Length	
$1 \to 0 \to 2 \to 3$	$(1, 3)$	3	Shortest for $(1, 3)$
$1 \to 0 \to 2 \to 2 \to 3$	$(1, 3)$	4	Not shortest for $(1, 3)$
$0 \to 2 \to 2 \to 3$	$(0, 3)$	3	
$0 \to 2 \to 3$	$(0, 3)$	2	
$0 \to 0 \to 3$	$(0, 3)$	2	

Full answer: $\{1 \to 0 \to 2 \to 3, \quad 0 \to 2 \to 3, \quad 0 \to 0 \to 3\}$

Shortest semantics (1)

Used in GSQL (TigerGraph), PGQL (Oracle) and GQL with `ALL SHORTEST`

Principles

- Return walks
- For each endpoints (s, t), return the "best" match from s to t
- Best = shortest = least number of edges

Example

Match	Endpoints	Length	
$1 \to 0 \to 2 \to 3$	$(1, 3)$	3	Shortest for $(1, 3)$
$1 \to 0 \to 2 \to 2 \to 3$	$(1, 3)$	4	Not shortest for $(1, 3)$
$0 \to 2 \to 2 \to 3$	$(0, 3)$	3	Not shortest for $(0, 3)$
$0 \to 2 \to 3$	$(0, 3)$	2	Tied shortest for $(0, 3)$
$0 \to 0 \to 3$	$(0, 3)$	2	Tied shortest for $(0, 3)$

Full answer: $\{1 \to 0 \to 2 \to 3,\quad 0 \to 2 \to 3,\quad 0 \to 0 \to 3\}$

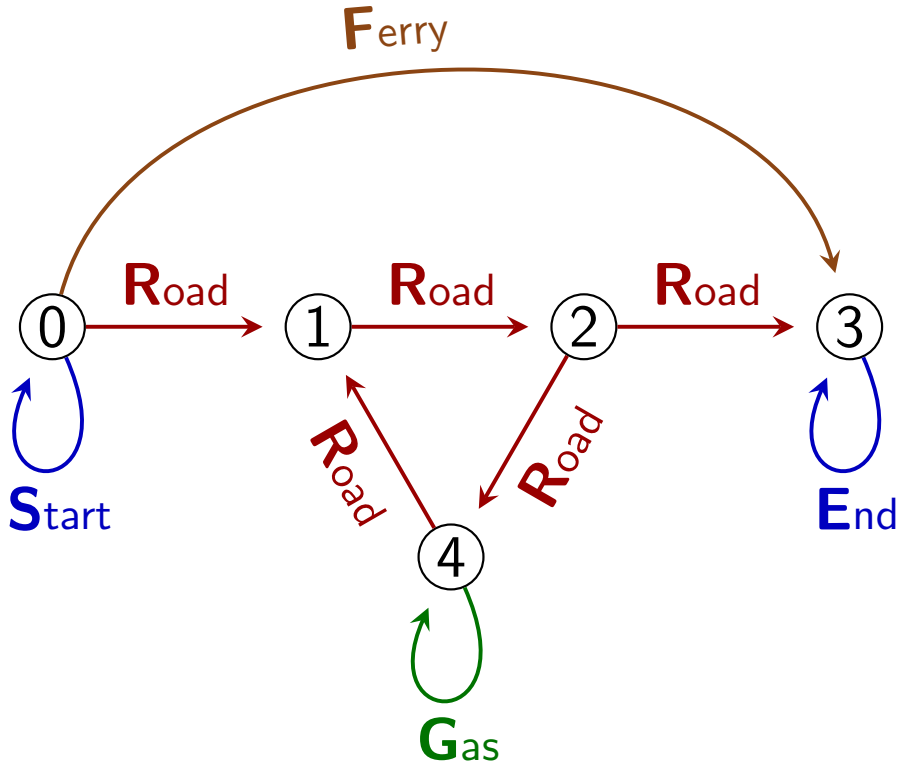
Shortest semantics (2)

Evaluating a reachability query

$Q_{12} = \mathbf{GR}^*$

Answer under shortest sem.

Walk	Shortest for
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)



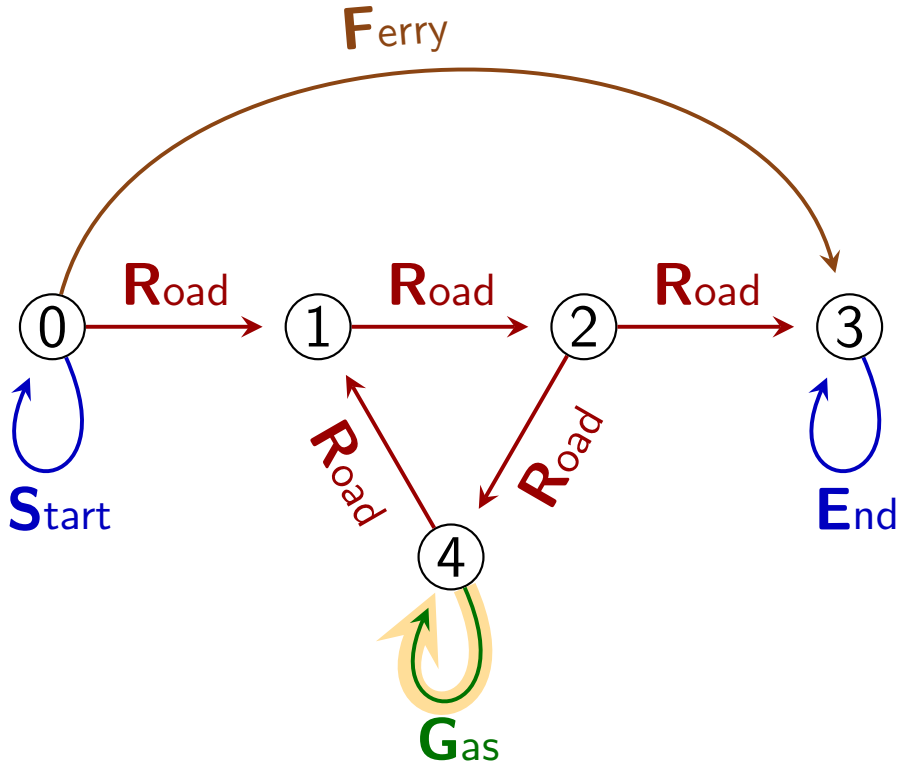
Shortest semantics (2)

Evaluating a reachability query

$Q_{12} = \mathbf{GR}^*$

Answer under shortest sem.

Walk	Shortest for
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)



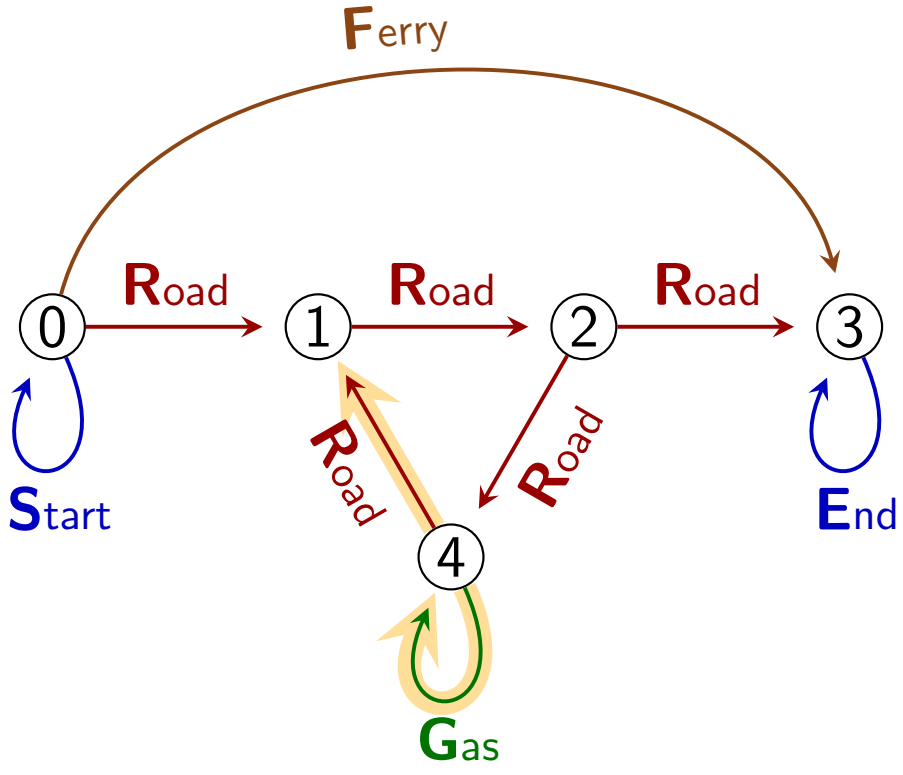
Shortest semantics (2)

Evaluating a reachability query

$Q_{12} = \mathbf{GR}^*$

Answer under shortest sem.

Walk	Shortest for
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)



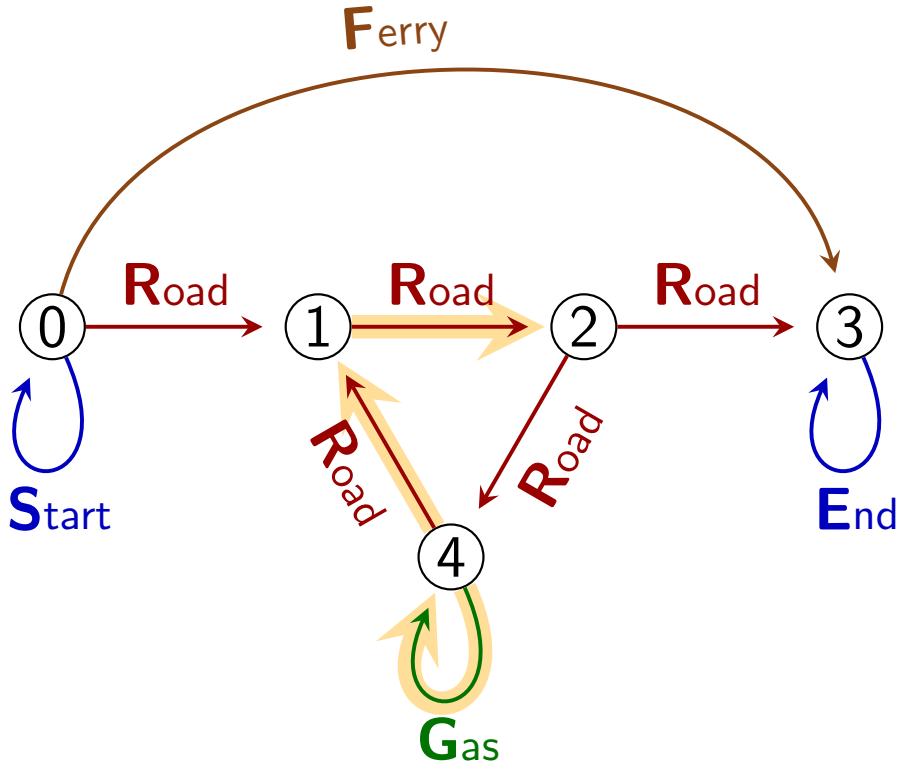
Shortest semantics (2)

Evaluating a reachability query

$Q_{12} = \mathbf{GR}^*$

Answer under shortest sem.

Walk	Shortest for
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)



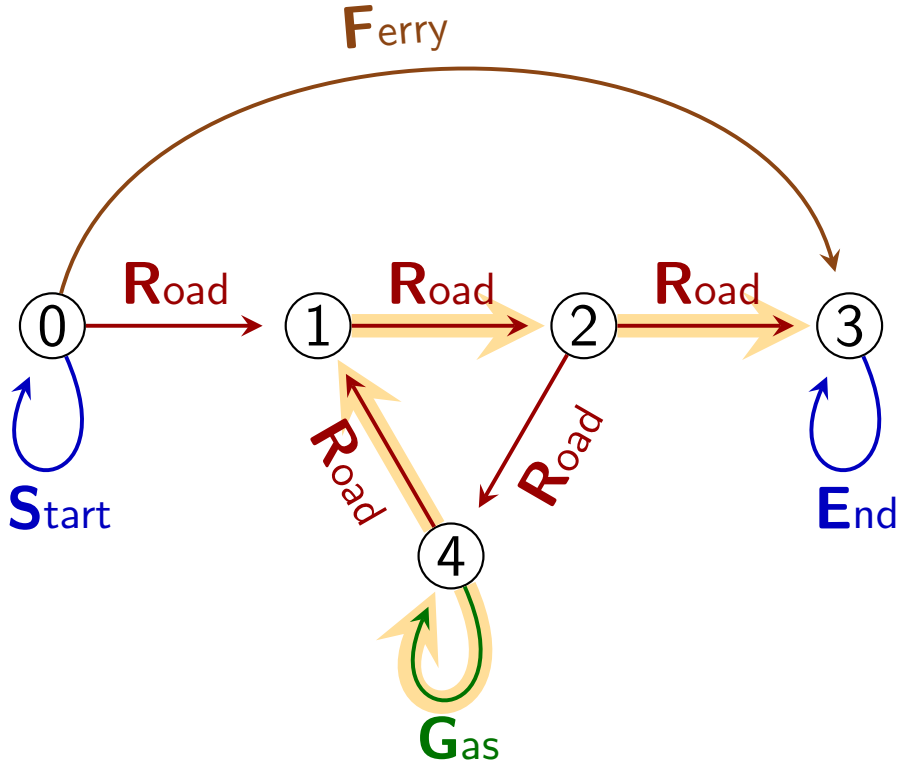
Shortest semantics (2)

Evaluating a reachability query

$Q_{12} = \mathbf{GR}^*$

Answer under shortest sem.

Walk	Shortest for
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)



Shortest semantics (2)

Evaluating a reachability query

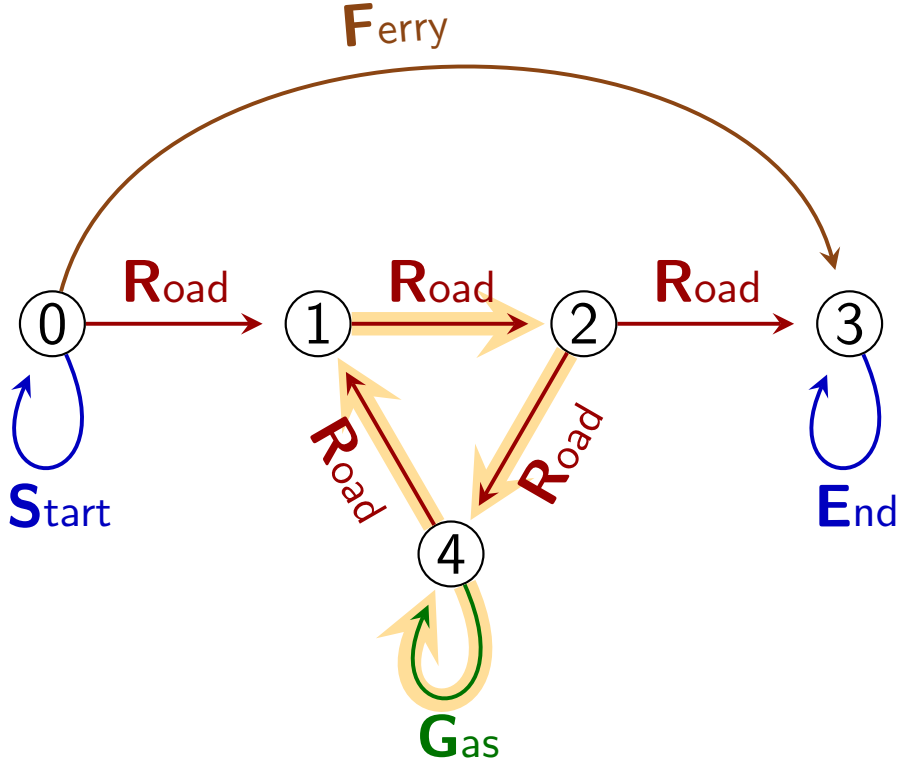
Q_{12} = **GR***

Answer under shortest sem.

Walk	Shortest for
$4 \to 4$	(4,4)
$4 \to 4 \to 1$	(4,1)
$4 \to 4 \to 1 \to 2$	(4,2)
$4 \to 4 \to 1 \to 2 \to 3$	(4,3)

Example of discarded match

$4 \to 4 \to 1 \to 2 \to 4$ is not in the answer because it is longer than $4 \to 4$



Shortest semantics (3)

Exercice: evaluating some queries

$Q_{13} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

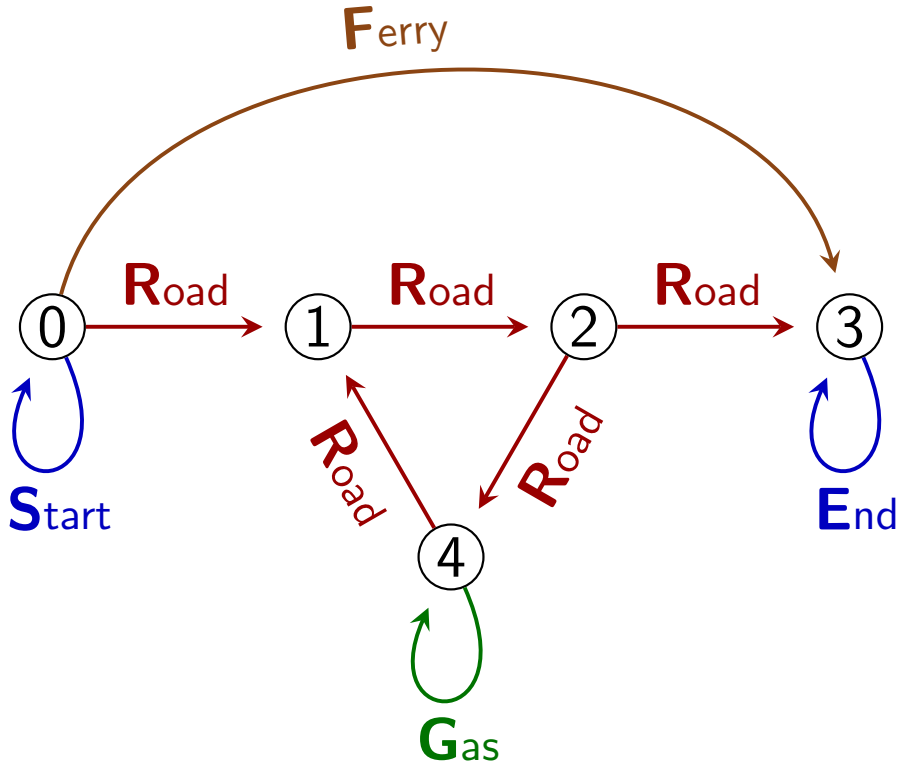
Answer to Q_{13}:

?

$Q_{14} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{14}:

?



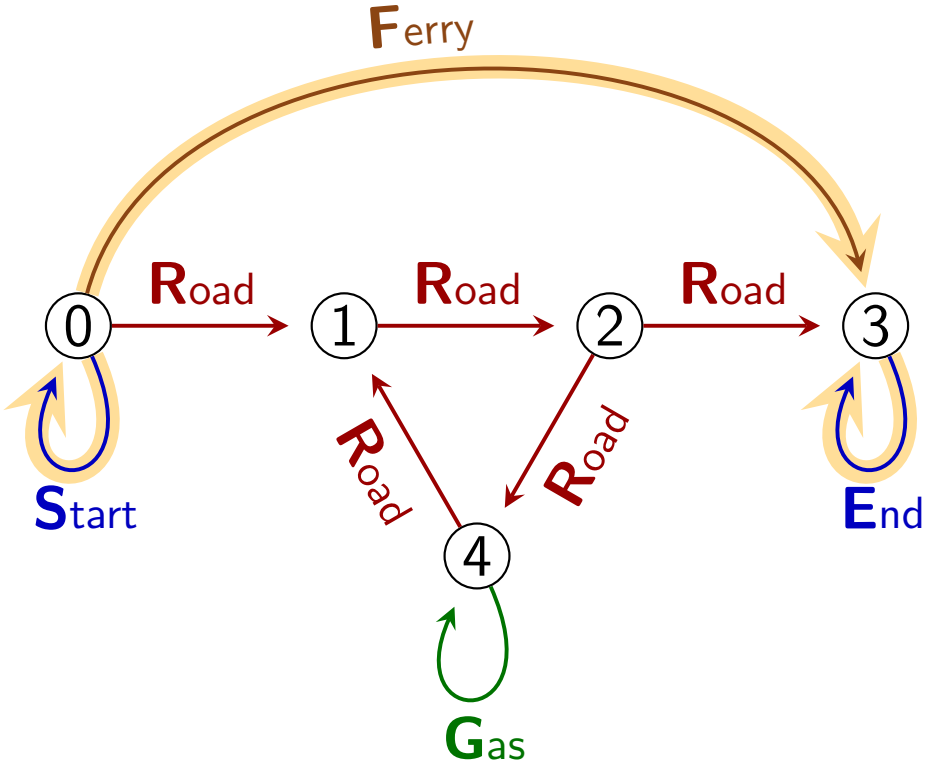
Shortest semantics (3)

Exercice: evaluating some queries

$Q_{13} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{13}:

$\{\ 0 \to 0 \to 3 \to 3\ \}$



$Q_{14} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{14}:

?

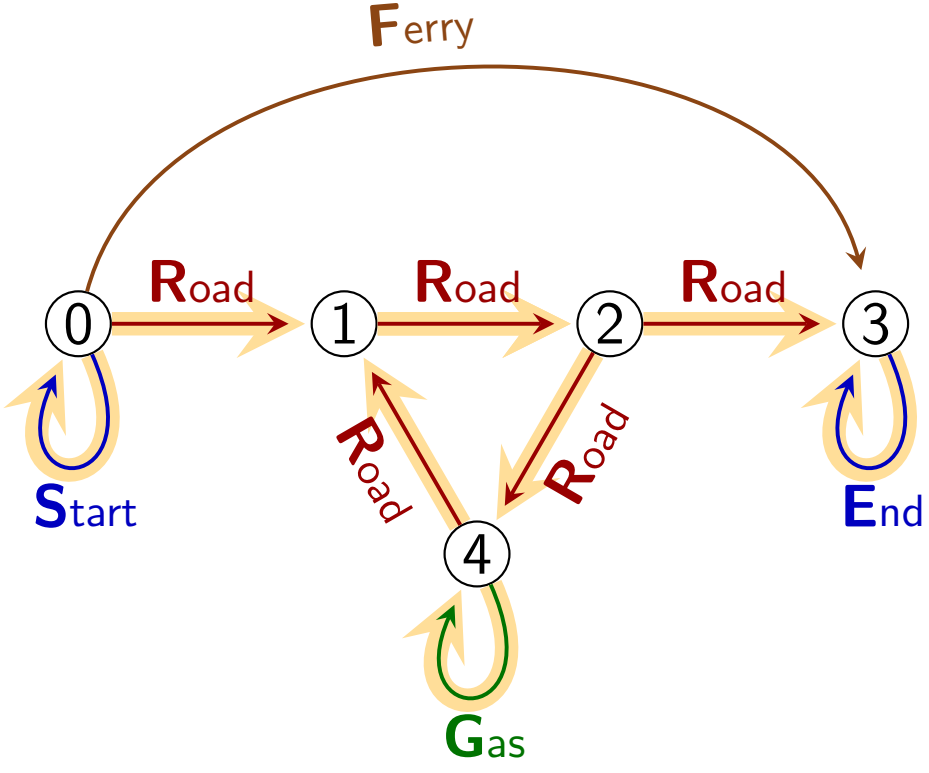
Shortest semantics (3)

Exercice: evaluating some queries

$Q_{13} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{13}:

$\{\ 0 \to 0 \to 3 \to 3\ \}$



$Q_{14} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{14}:

$\{\ 0 \to 0 \to 1 \to 2 \to 4 \to 4 \to 1 \to 2 \to 3 \to 3\ \}$

Shortest semantics (4)

Pros

- Returns walks
- Efficient algorithms (BFS in the product graph×query)
- If there are matches from s to t, at least one of them is in the answer

Shortest semantics (4)

Pros and con

Pros

- Returns walks
- Efficient algorithms (BFS in the product graph×query)
- If there are matches from s to t, at least one of them is in the answer

Cons

- The shortest walk is not always the "best"
  - *"Do we always want to take the ferry over the direct road?"*
  - (Real query languages allow to assign costs to edges/atoms)
- No vertical post-processing
  - Vertical = accross the walks with the same endpoints
  - *"What is the average time?"*
  - *"What is the connectedness level?"*

Trail semantics (1)

Used by Cypher (Neo4j) and GQL with keyword `ALL TRAIL`

Trail semantics (1)

Used by Cypher (Neo4j) and GQL with keyword `ALL TRAIL`

Principle

- Return a set of walks
- Apply a filter on the set of matching walks
- The filter is: each walk that repeats an edge is filtered out

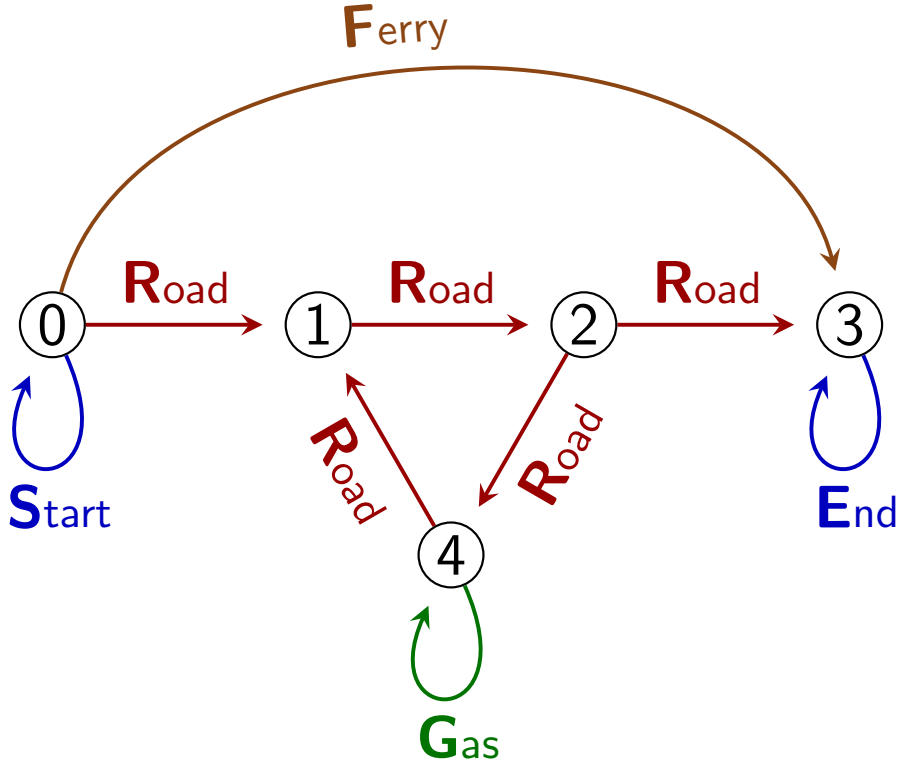
Examples

Match	Decision
$1 \to 0 \to 2 \to 2 \to 3$	No repetition $\Rightarrow$ Kept in the answer
$1 \to 0 \to 2 \to 3 \to 0 \to 2$	Repeated edges $\Rightarrow$ Filtered out

Trail semantics (2)

Evaluating Q_{15}

$Q_{15} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$



Applying the filter

Matches	Keep?
The ferry walk	
The straight road	
The road with 1 lap	
The road with 2 laps	
⋮	

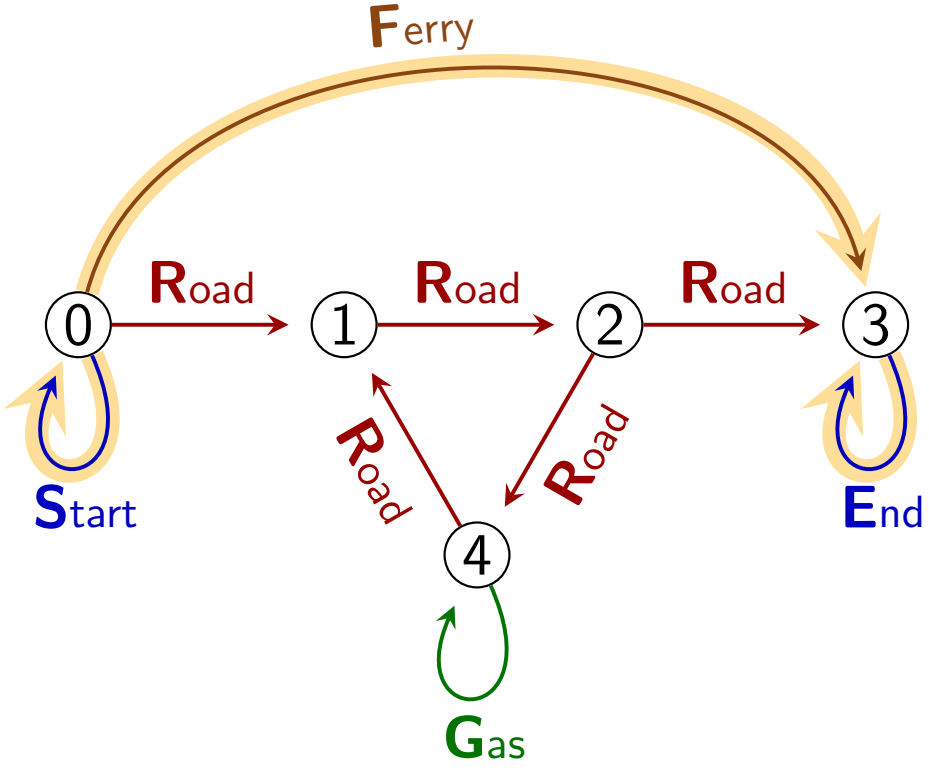
Answer of Q_{15} under trail semantics:

{ }

Trail semantics (2)

Evaluating Q_{15}

$Q_{15} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$



Applying the filter

Matches	Keep?
The ferry walk	
The straight road	
The road with 1 lap	
The road with 2 laps	
⋮	

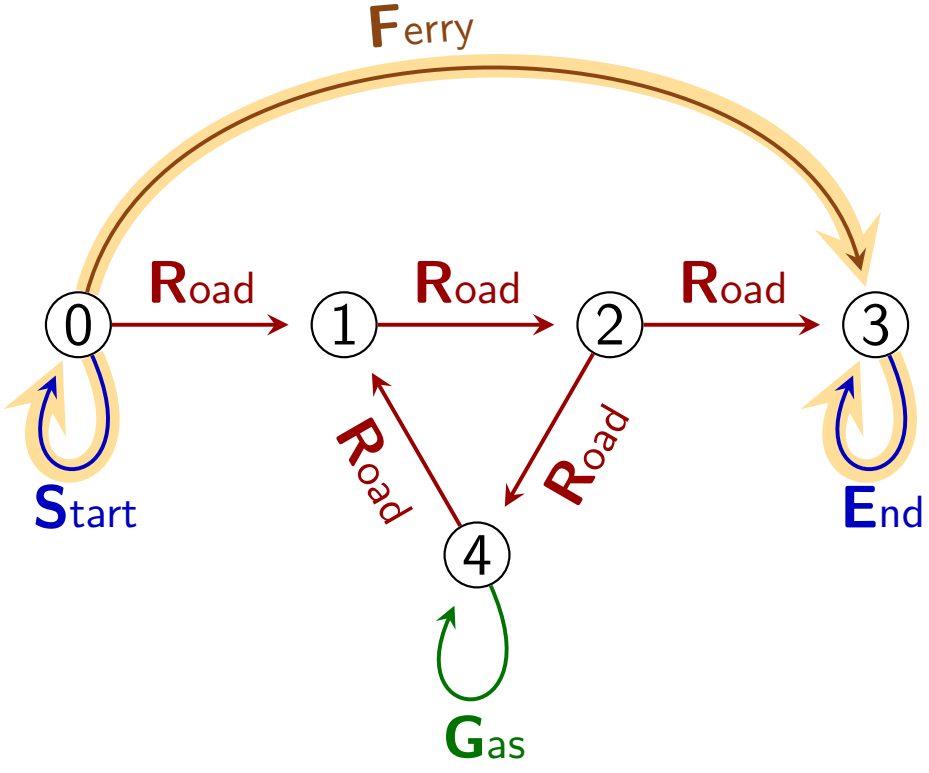
Answer of Q_{15} under trail semantics:

{ }

Trail semantics (2)

Evaluating Q_{15}

$$Q_{15} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$$



Applying the filter

Matches	Keep?
The ferry walk	Yes
The straight road	
The road with 1 lap	
The road with 2 laps	
⋮	

Answer of Q_{15} under trail semantics:

{ $0 \to 0 \to 3 \to 3$ }

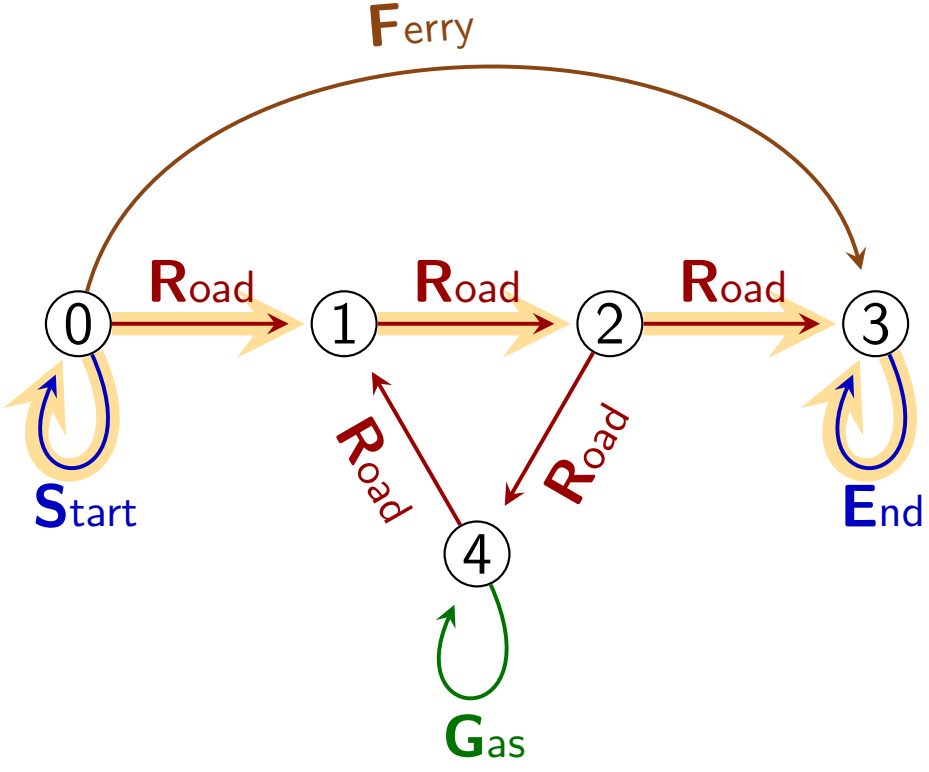
Trail semantics (2)

Evaluating Q_{15}

$$Q_{15} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$$

Applying the filter

Matches	Keep?
The ferry walk	Yes
The straight road	
The road with 1 lap	
The road with 2 laps	
⋮	



Answer of Q_{15} under trail semantics:

$\{ \quad 0 \to 0 \to 3 \to 3 \quad \}$

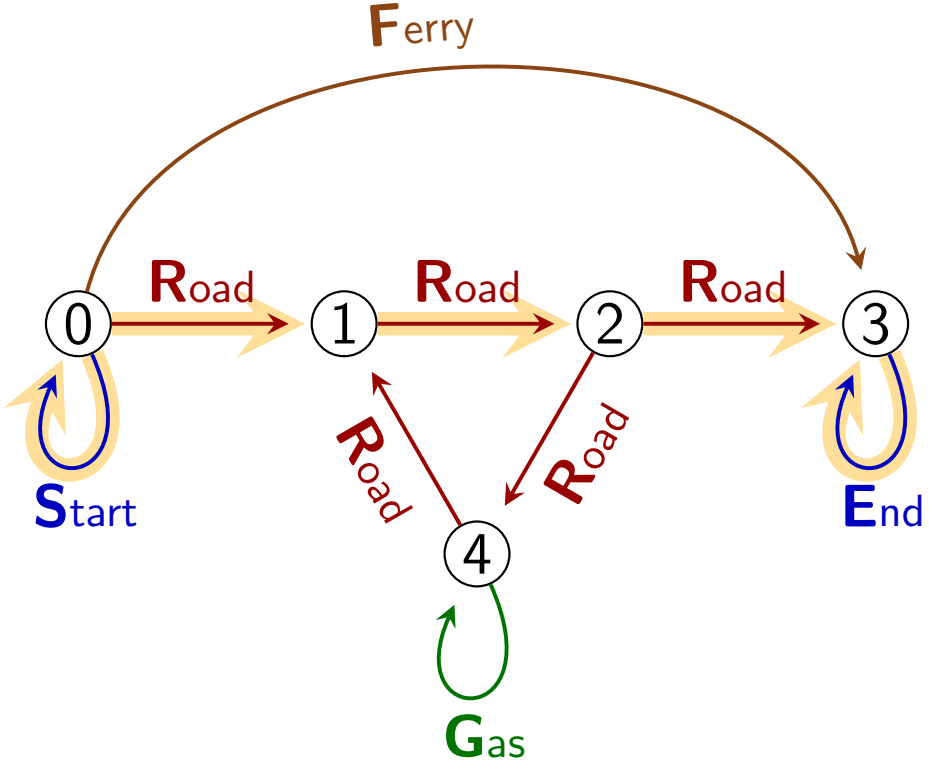
Trail semantics (2)

Evaluating Q_{15}

$$Q_{15} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$$

Applying the filter

Matches	Keep?
The ferry walk	Yes
The straight road	Yes
The road with 1 lap	
The road with 2 laps	
⋮	



Answer of Q_{15} under trail semantics:

$\{ \quad 0 \to 0 \to 3 \to 3, \quad 0 \to 0 \to 1 \to 2 \to 3 \to 3 \quad \}$

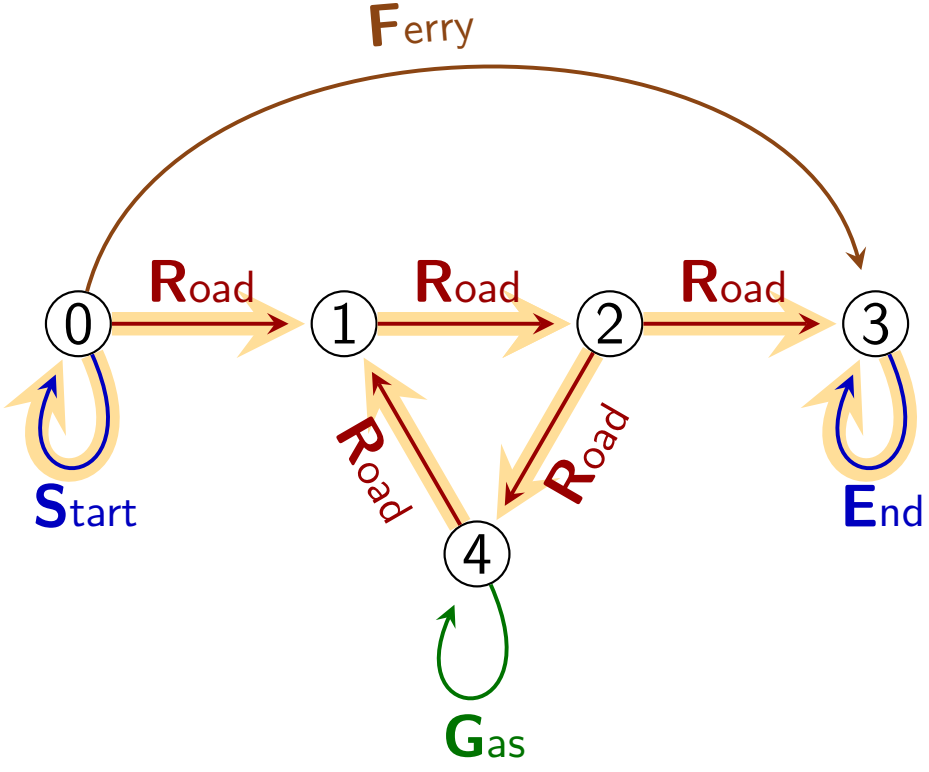
Trail semantics (2)

Evaluating Q_{15}

$Q_{15} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Applying the filter

Matches	Keep?
The ferry walk	Yes
The straight road	Yes
The road with 1 lap	
The road with 2 laps	
⋮	



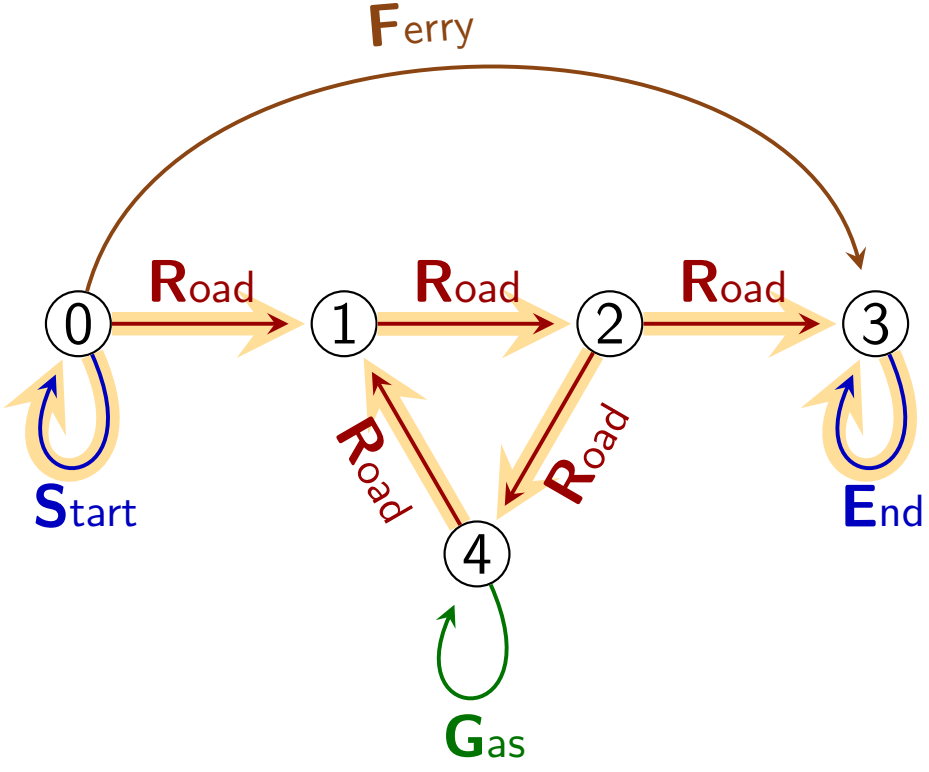
Answer of Q_{15} under trail semantics:

$\{ \quad 0 \to 0 \to 3 \to 3, \quad 0 \to 0 \to 1 \to 2 \to 3 \to 3 \quad \}$

Trail semantics (2)

Evaluating Q_{15}

$Q_{15} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$



Applying the filter

Matches	Keep?
The ferry walk	Yes
The straight road	Yes
The road with 1 lap	No
The road with 2 laps	
⋮	

Answer of Q_{15} under trail semantics:

$\{ \quad 0 \to 0 \to 3 \to 3, \quad 0 \to 0 \to 1 \to 2 \to 3 \to 3 \quad \}$

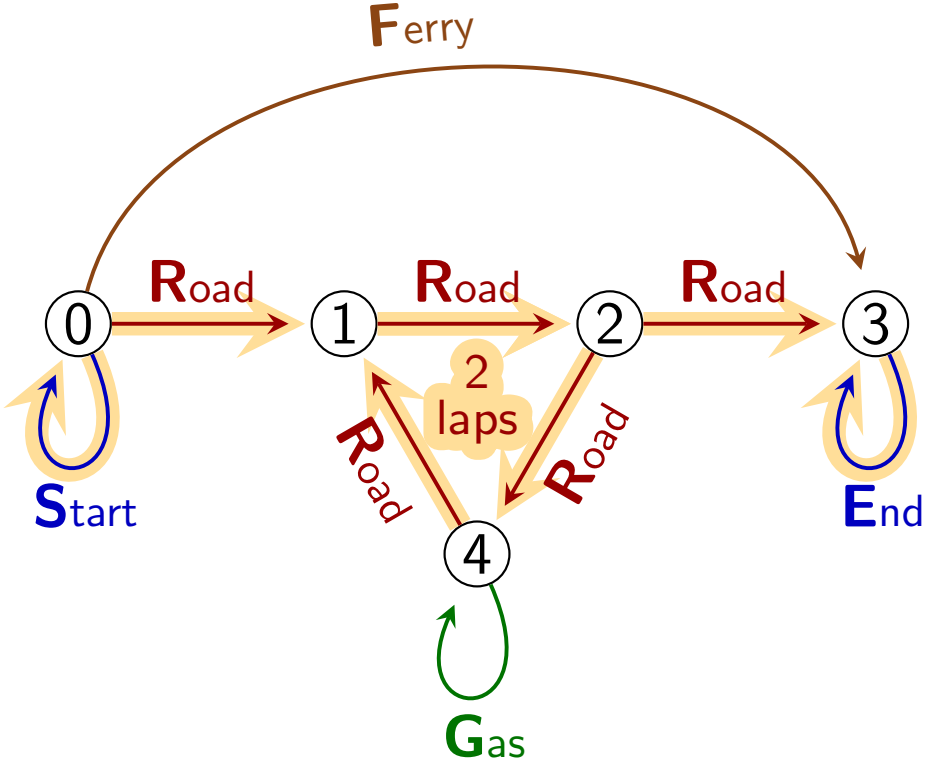
Trail semantics (2)

Evaluating Q_{15}

$$Q_{15} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$$

Applying the filter

Matches	Keep?
The ferry walk	Yes
The straight road	Yes
The road with 1 lap	No
The road with 2 laps	
⋮	



Answer of Q_{15} under trail semantics:

$$\{\quad 0 \to 0 \to 3 \to 3,\quad 0 \to 0 \to 1 \to 2 \to 3 \to 3 \quad\}$$

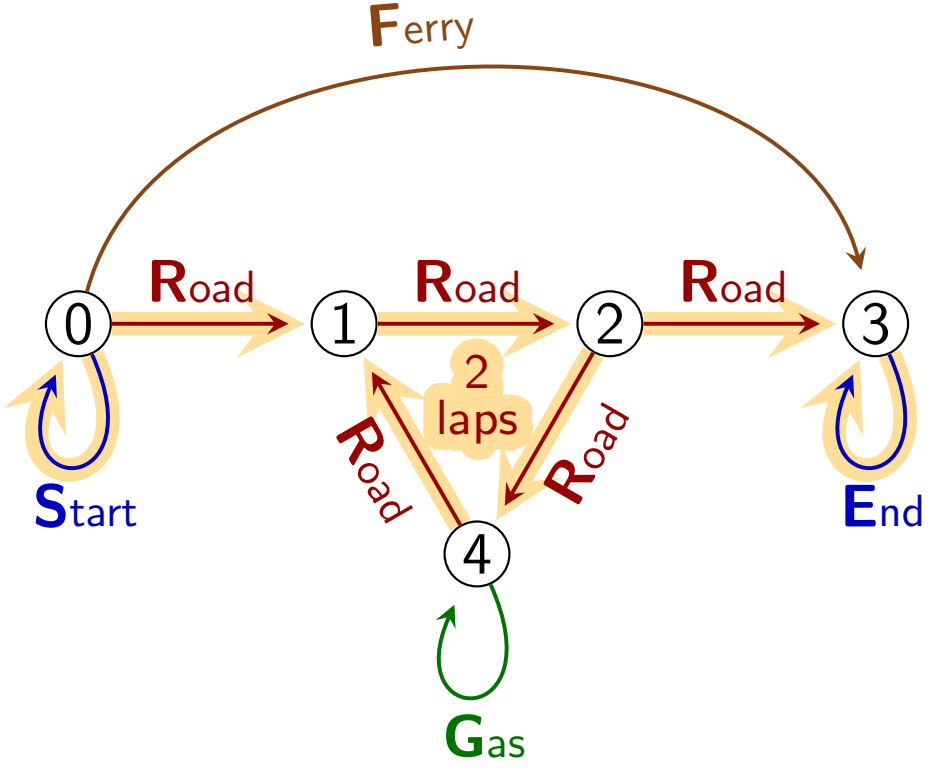
Trail semantics (2)

Evaluating Q_{15}

$$Q_{15} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$$

Applying the filter

Matches	Keep?
The ferry walk	Yes
The straight road	Yes
The road with 1 lap	No
The road with 2 laps	No
⋮	



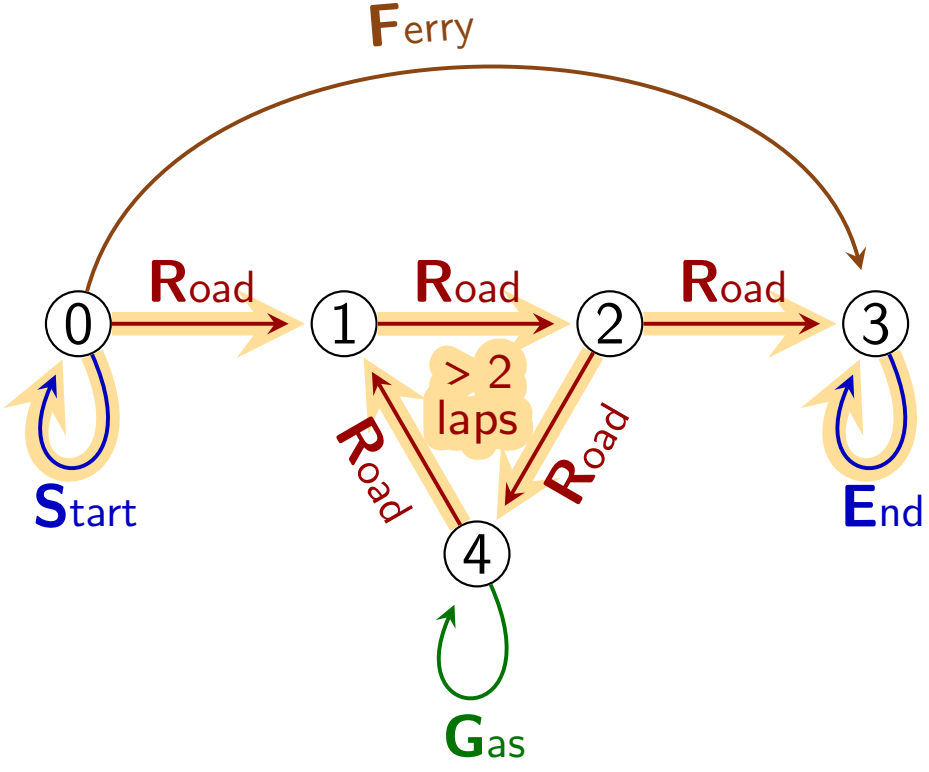
Answer of Q_{15} under trail semantics:

$$\{\quad 0 \to 0 \to 3 \to 3,\quad 0 \to 0 \to 1 \to 2 \to 3 \to 3 \quad\}$$

Trail semantics (2)

Evaluating Q_{15}

$$Q_{15} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$$



Applying the filter

Matches	Keep?
The ferry walk	Yes
The straight road	Yes
The road with 1 lap	No
The road with 2 laps	No
⋮	No

Answer of Q_{15} under trail semantics:

$$\{\quad 0 \to 0 \to 3 \to 3,\quad 0 \to 0 \to 1 \to 2 \to 3 \to 3 \quad\}$$

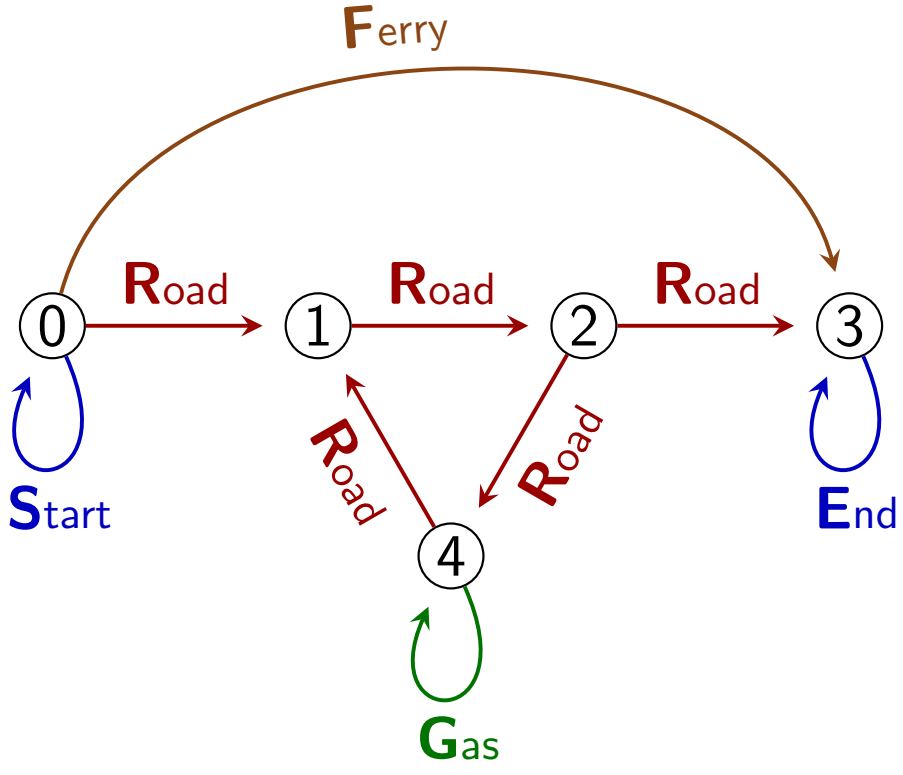
Trail semantics (3)

Exercice: evaluating some queries

$Q_{16} = \mathbf{G}\mathbf{R}^*$

Answer to Q_{16}:

?



$Q_{17} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{17}:

?

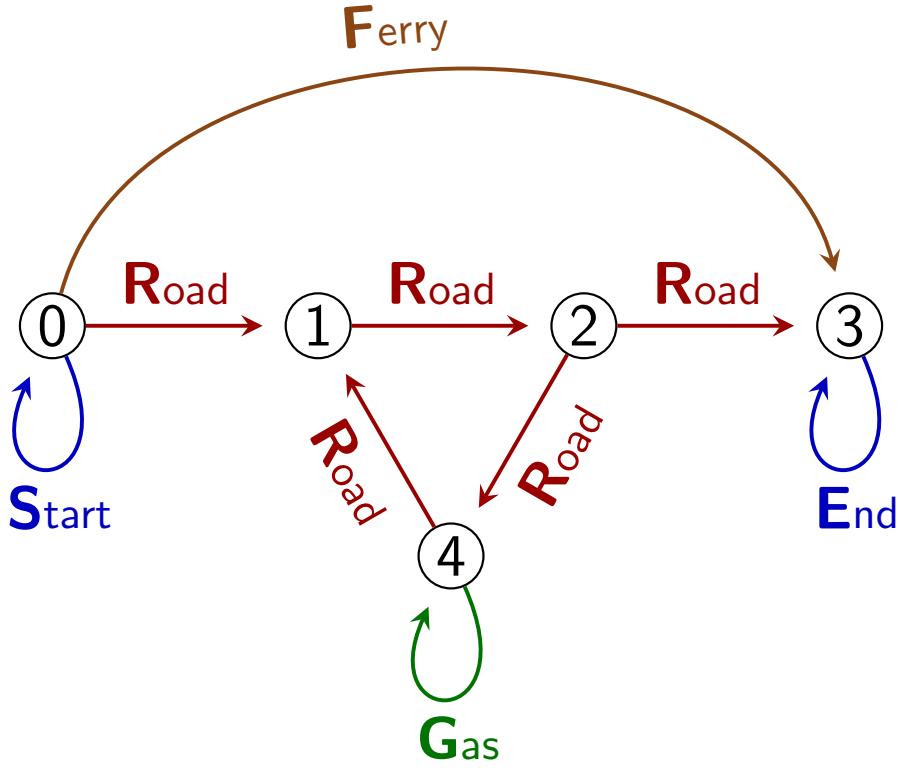
Trail semantics (3)

Exercice: evaluating some queries

$Q_{16} = \mathbf{G}\mathbf{R}^*$

Answer to Q_{16}:

$$\{\ 4 \to 4\ ,\ 4 \to 4 \to 1\ ,\ 4 \to 4 \to 1 \to 2\ ,\ 4 \to 4 \to 1 \to 2 \to 3\ ,\ 4 \to 4 \to 1 \to 2 \to 4\ \}$$



$Q_{17} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{17}:

?

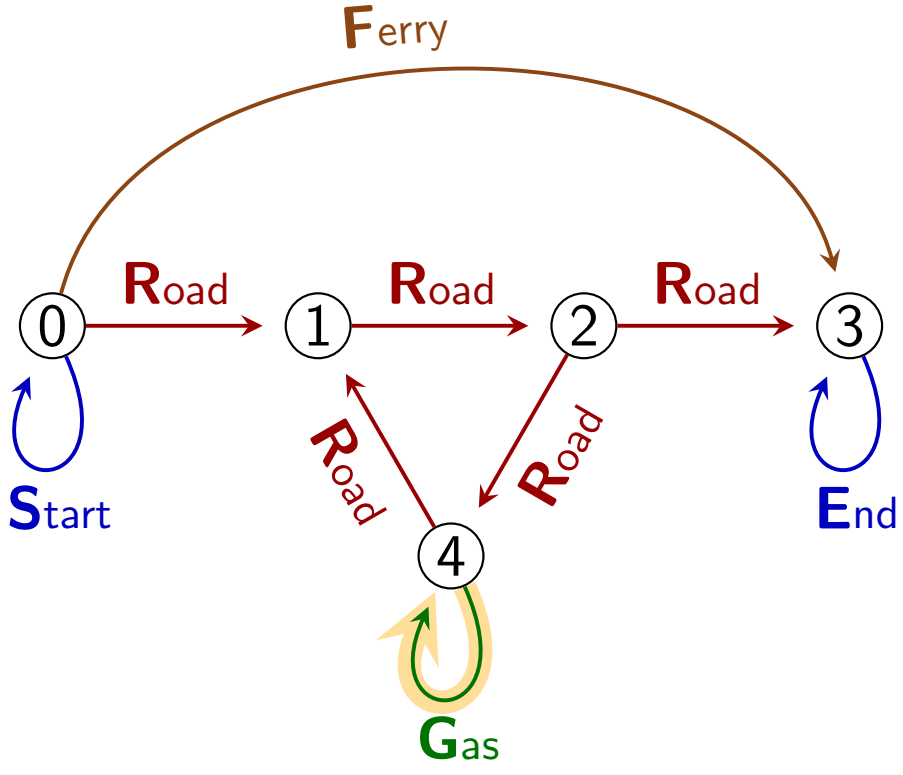
Trail semantics (3)

Exercice: evaluating some queries

$Q_{16} = \mathbf{G}\mathbf{R}^*$

Answer to Q_{16}:

$\{ \; 4 \to 4 \;,$
$4 \to 4 \to 1 \;,$
$4 \to 4 \to 1 \to 2 \;,$
$4 \to 4 \to 1 \to 2 \to 3 \;,$
$4 \to 4 \to 1 \to 2 \to 4 \quad \}$



$Q_{17} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{17}:

?

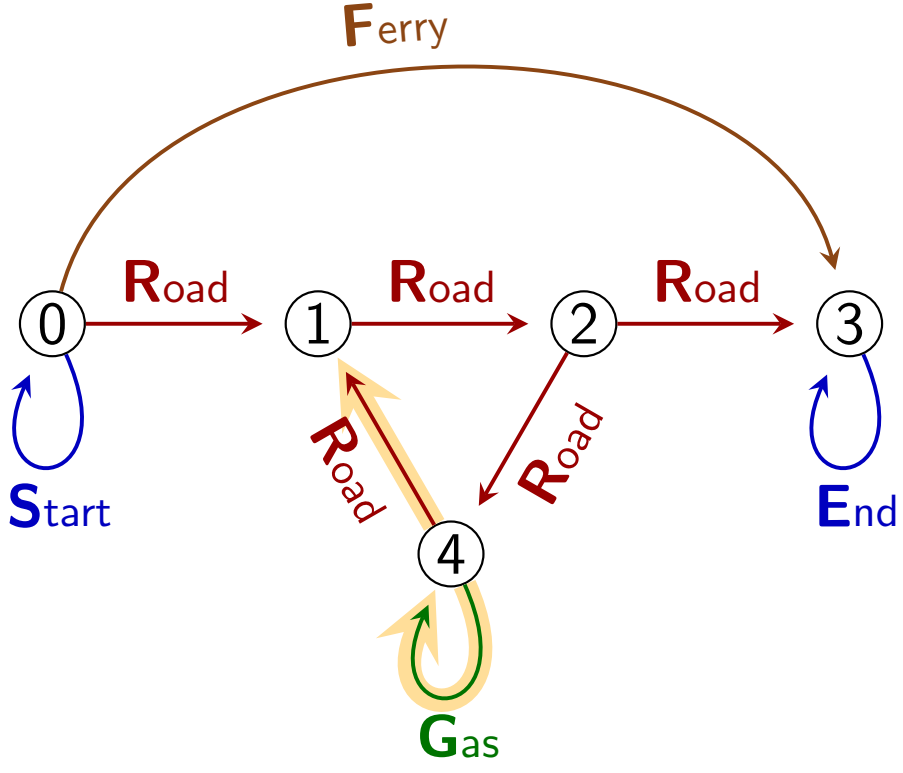
Trail semantics (3)

Exercice: evaluating some queries

$Q_{16} = \mathbf{G}\mathbf{R}^*$

Answer to Q_{16}:

$\{ \; 4 \to 4 \,,$
$4 \to 4 \to 1 \,,$
$4 \to 4 \to 1 \to 2 \,,$
$4 \to 4 \to 1 \to 2 \to 3 \,,$
$4 \to 4 \to 1 \to 2 \to 4 \quad \}$



$Q_{17} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{17}:

?

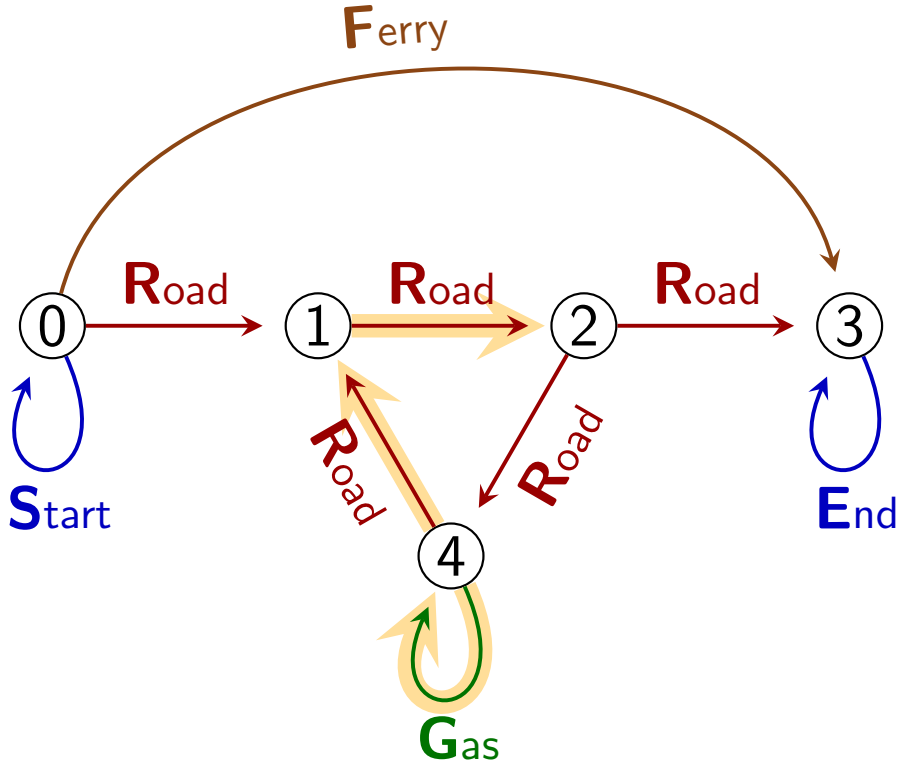
Trail semantics (3)

Exercice: evaluating some queries

$Q_{16} = \mathbf{G}\mathbf{R}^*$

Answer to Q_{16}:

$$\{\ 4 \to 4\ ,\ 4 \to 4 \to 1\ ,\ 4 \to 4 \to 1 \to 2\ ,\ 4 \to 4 \to 1 \to 2 \to 3\ ,\ 4 \to 4 \to 1 \to 2 \to 4\ \}$$



$Q_{17} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{17}:

?

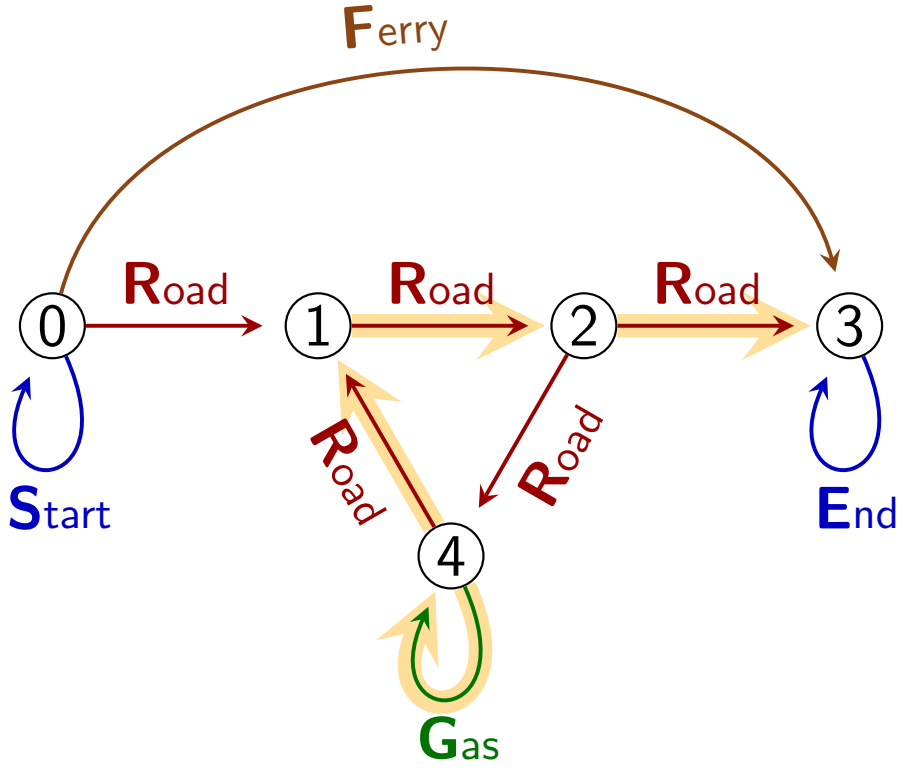
Trail semantics (3)

Exercice: evaluating some queries

$Q_{16} = \mathbf{G}\mathbf{R}^*$

Answer to Q_{16}:

$$\{\ 4 \to 4\ ,\ 4 \to 4 \to 1\ ,\ 4 \to 4 \to 1 \to 2\ ,\ 4 \to 4 \to 1 \to 2 \to 3\ ,\ 4 \to 4 \to 1 \to 2 \to 4\ \}$$



$Q_{17} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{17}:

?

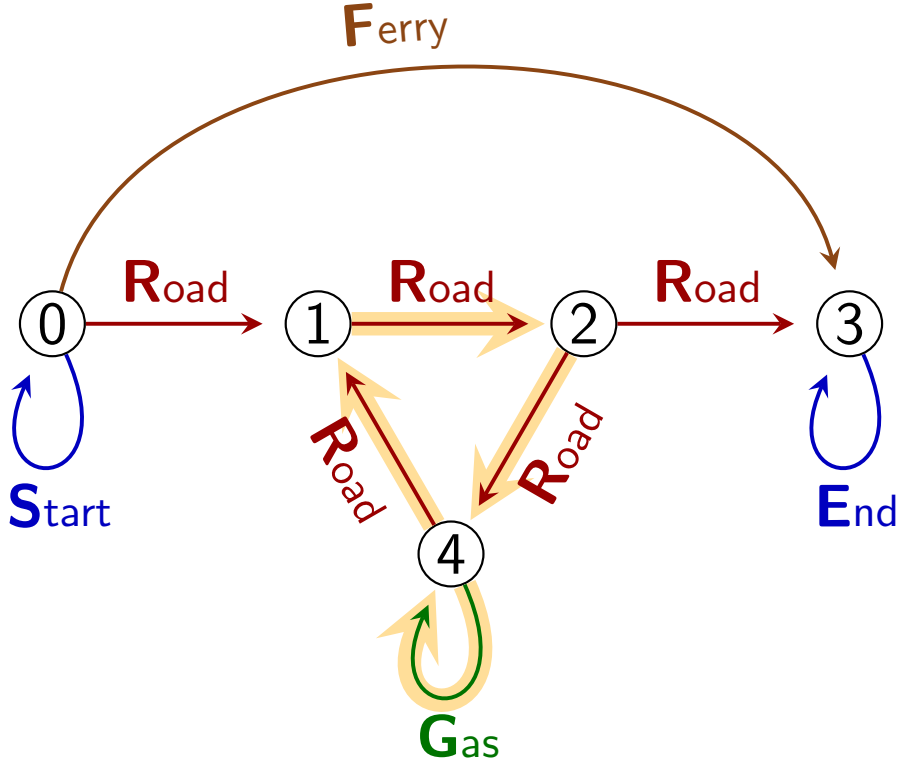
Trail semantics (3)

Exercice: evaluating some queries

$Q_{16} = \mathbf{G}\mathbf{R}^*$

Answer to Q_{16}:

$\{ \; 4 \to 4 \,,$
$4 \to 4 \to 1 \,,$
$4 \to 4 \to 1 \to 2 \,,$
$4 \to 4 \to 1 \to 2 \to 3 \,,$
$4 \to 4 \to 1 \to 2 \to 4 \quad \}$



$Q_{17} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{17}:

?

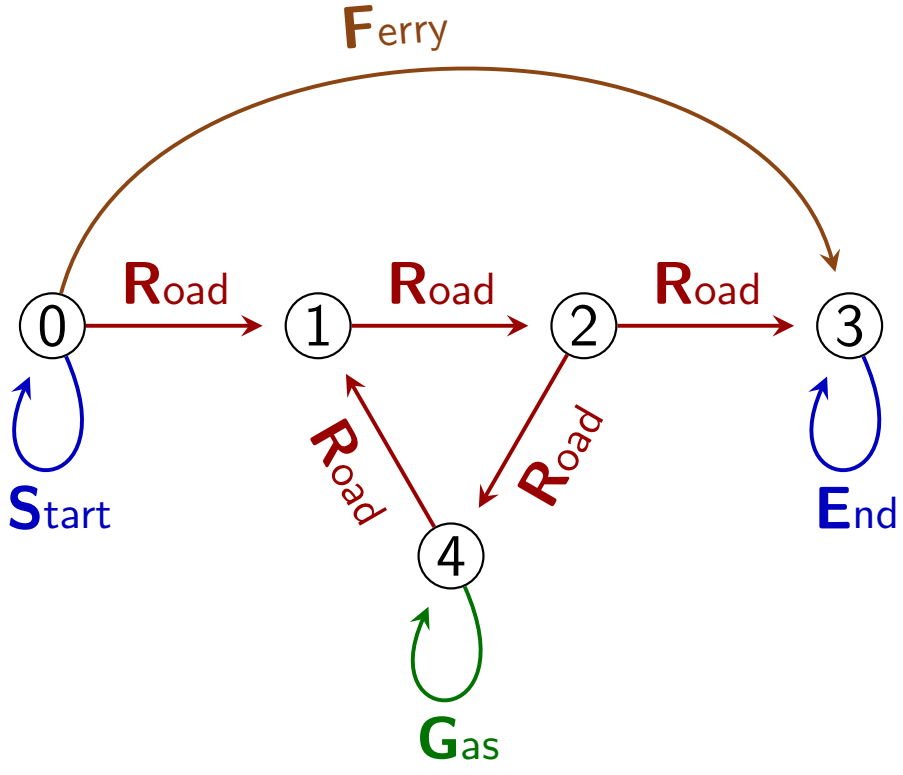
Trail semantics (3)

Exercice: evaluating some queries

$Q_{16} = \mathbf{G}\mathbf{R}^*$

Answer to Q_{16}:

$$\{\ 4 \to 4\ ,\ 4 \to 4 \to 1\ ,\ 4 \to 4 \to 1 \to 2\ ,\ 4 \to 4 \to 1 \to 2 \to 3\ ,\ 4 \to 4 \to 1 \to 2 \to 4\ \}$$



$Q_{17} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{17}:

?

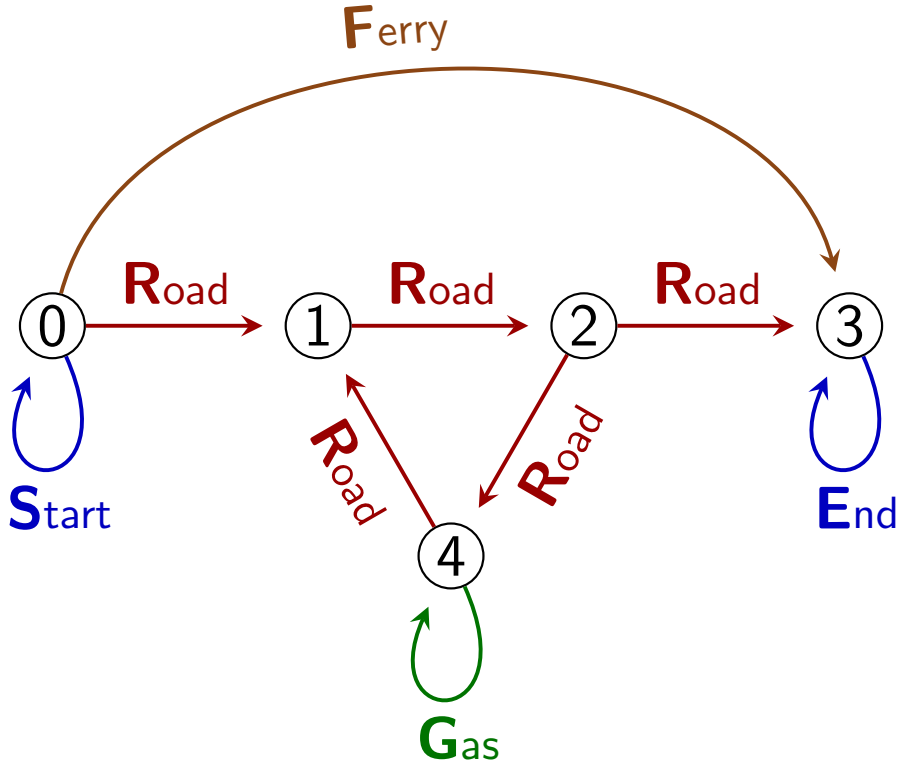
Trail semantics (3)

Exercice: evaluating some queries

$Q_{16} = \mathbf{G}\mathbf{R}^*$

Answer to Q_{16}:

$$\{\ 4 \to 4\ ,\ 4 \to 4 \to 1\ ,\ 4 \to 4 \to 1 \to 2\ ,\ 4 \to 4 \to 1 \to 2 \to 3\ ,\ 4 \to 4 \to 1 \to 2 \to 4\ \}$$



$Q_{17} = \mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$

Answer to Q_{17}:

$\varnothing$

Trail semantics (4)

Pros

- Returns walks
- Easy to explain
- Enable vertical post-processing
  - Vertical = accross the walks with the same endpoints
  - *"What is the average time?"*
  - *"What is the connectedness level?"*

Trail semantics (4)

Pros and cons

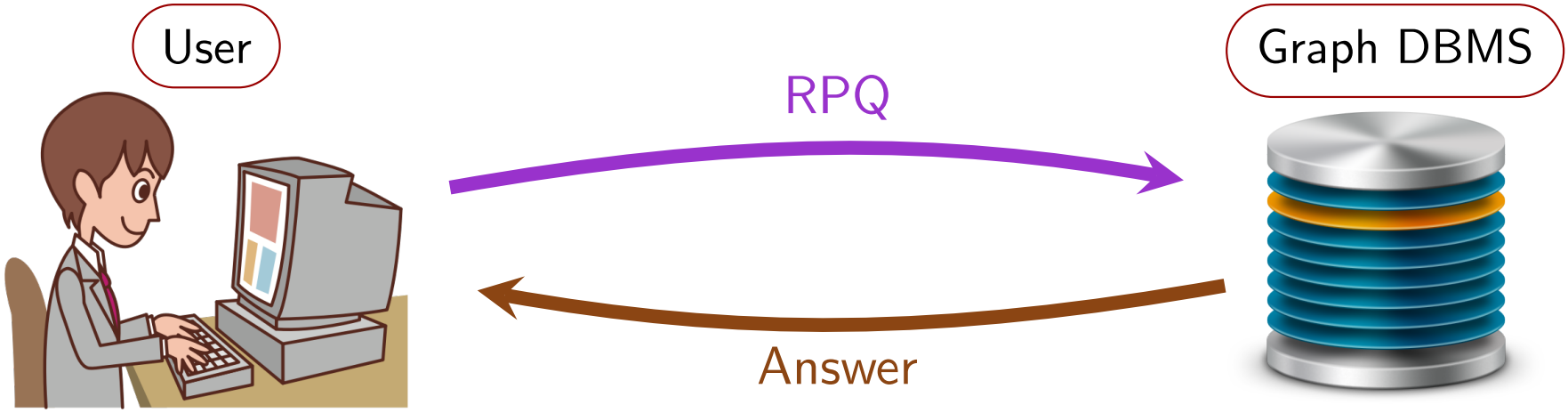
Pros

- Returns walks
- Easy to explain
- Enable vertical post-processing
  - Vertical = accross the walks with the same endpoints
  - *"What is the average time?"*
  - *"What is the connectedness level?"*

Cons

- **Inefficient** in bad cases.
  Ex: checking whether $\mathbf{R}^*\mathbf{G}\mathbf{R}^*$ returns anything is NP-hard
- "No repeated edge" is a filter that is sometimes **counterintuitive**
  Ex: $\mathbf{S}(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*\mathbf{E}$ had matches but the answer is empty

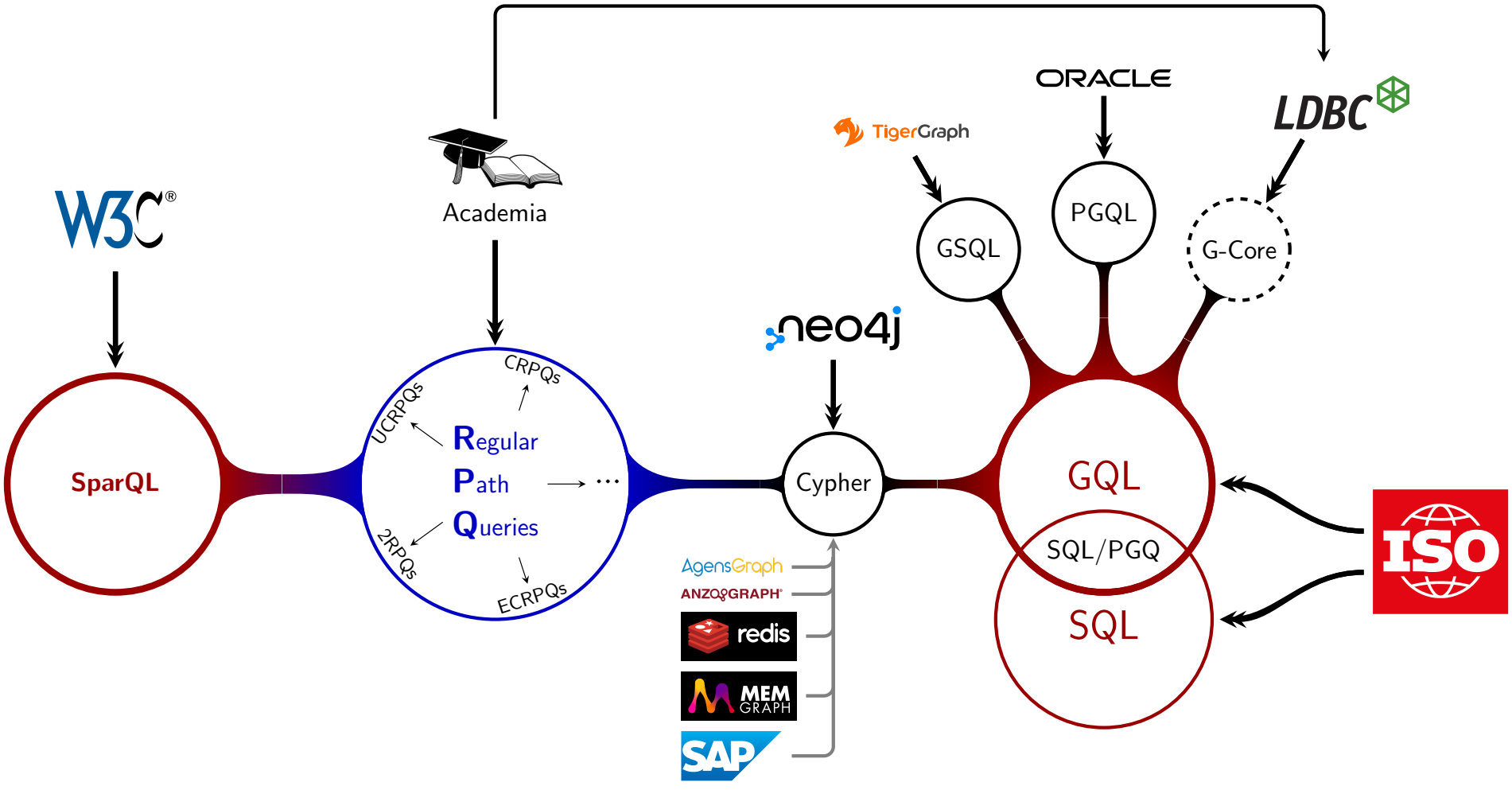
Computing a finite answer



⚠ **Infinitely** many matches but the user expects **finite** answer ⚠

- Endpoint → Filters out all navigational information
- Shortest → No vertical postprocessing and arbitrary metrics
- Trail → Inefficient and sometimes discard meaningful matches

$\Longrightarrow$ No RPQ semantics is clearly superior



- SPARQL and most academic work on RPQs use endpoint semantics
- Cypher uses trail semantics
- GSQL, PGQL and G-Core uses shortest semantics (and variants)
- GQL and SQL/PGQ allow to switch between many RPQ semantics

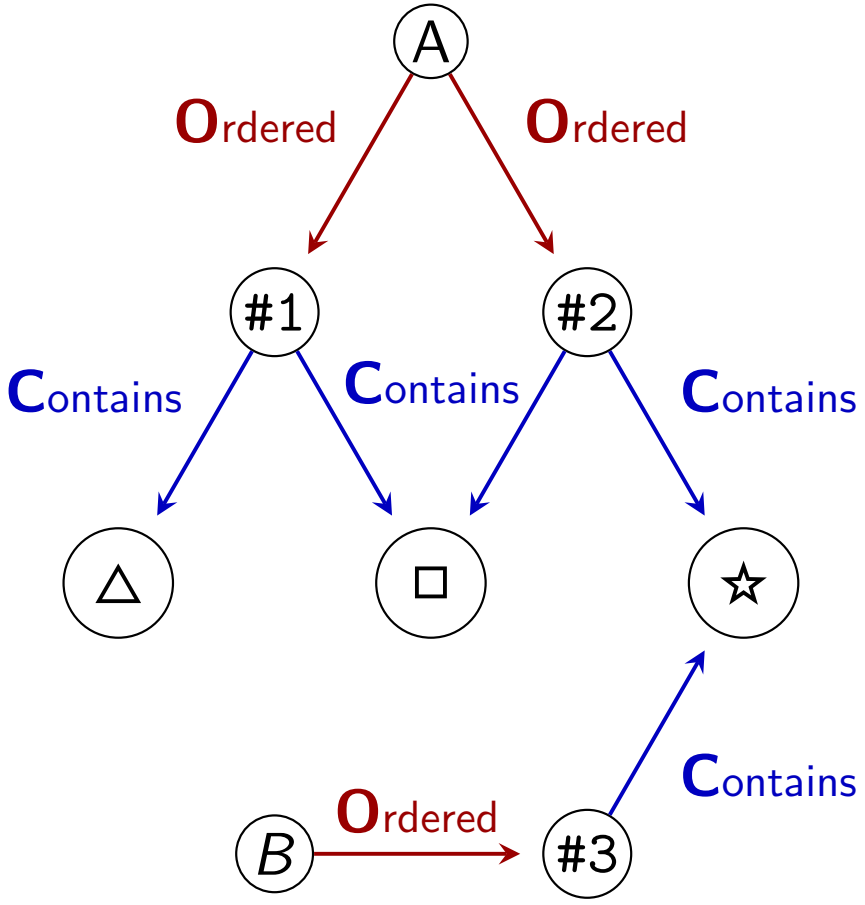
Part I: Theoretical foundations

4. Common extensions to RPQs

Two user requests

Consider the graph with

- clients (A,B)
- orders (#1,#2,#3)
- products ($\triangle, \square, \star$)



Two user requests

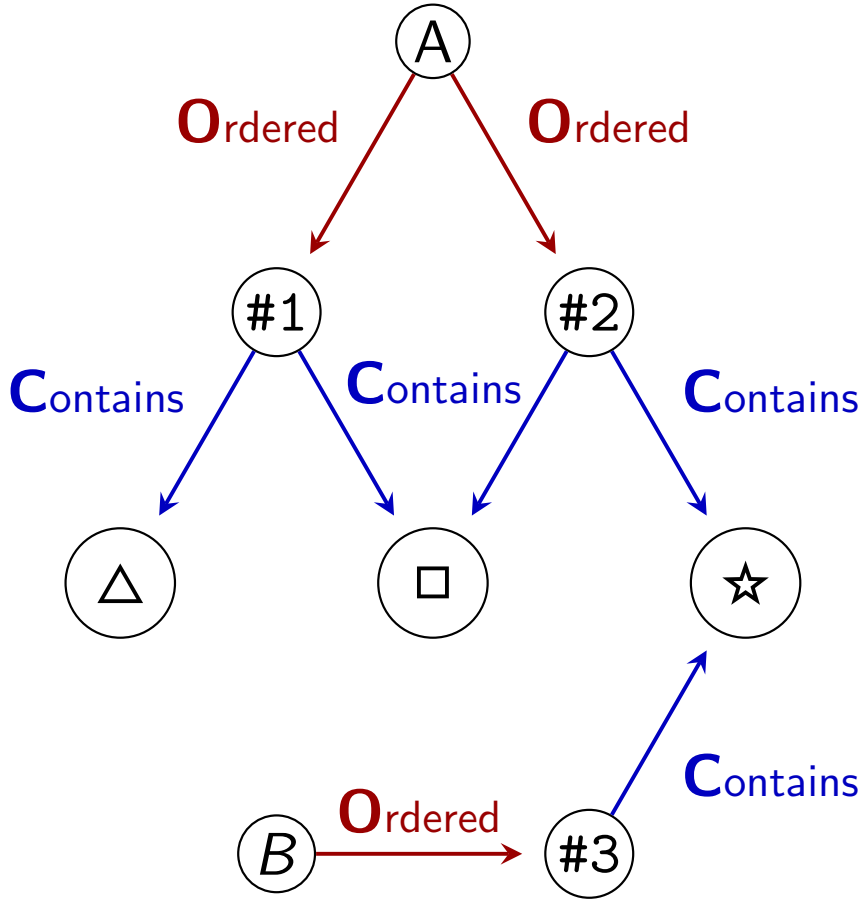
Consider the graph with

- clients (A, B)
- orders (#1,#2,#3)
- products $(\triangle, \square, \star)$

Write RPQs to extract

1. Products that were ordered twice (that is ☆ and □).

2. Triples (x, y, z) such that x ordered y and z in the same order. Ex: $(A, \triangle, \square)$.



Two user requests

Consider the graph with

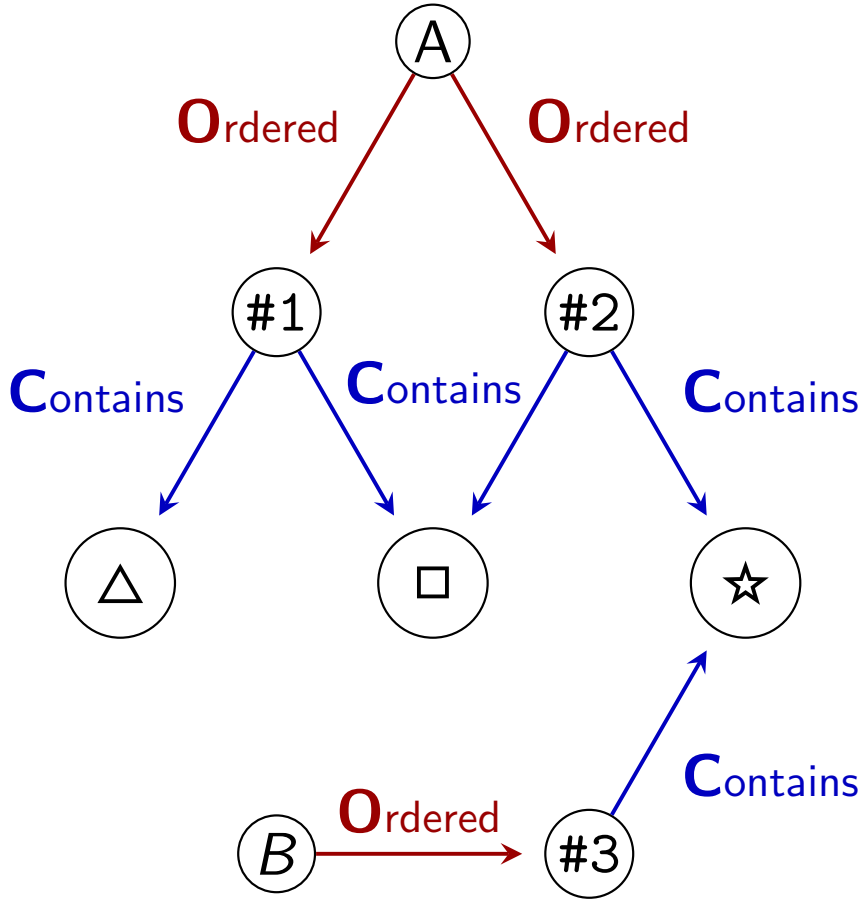
- clients (A, B)
- orders (#1,#2,#3)
- products $(\triangle, \square, \star)$

Write RPQs to extract

1. Products that were ordered twice (that is ☆ and □).

   ⚠ Impossible ⚠

2. Triples (x, y, z) such that x ordered y and z in the same order. Ex: $(A, \triangle, \square)$.



Two user requests

Consider the graph with

- clients (A, B)
- orders (#1,#2,#3)
- products $(\triangle, \square, \star)$

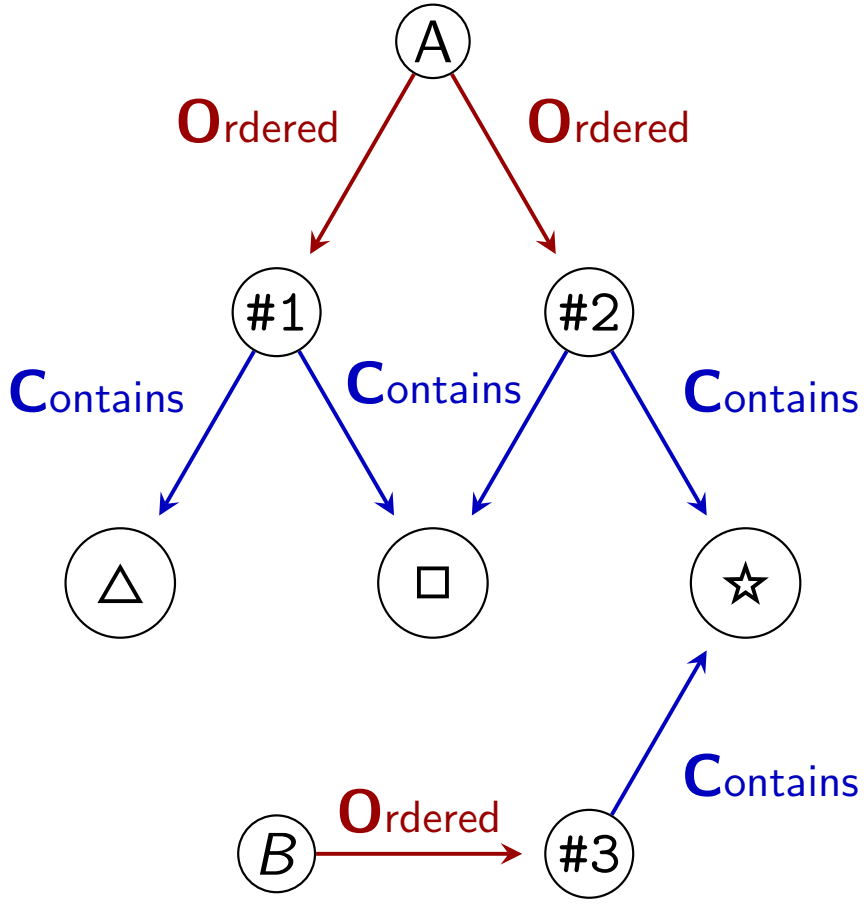
Write RPQs to extract

1. Products that were ordered twice (that is ☆ and □).

   ⚠ Impossible ⚠

2. Triples (x, y, z) such that x ordered y and z in the same order. Ex: $(A, \triangle, \square)$.

   ⚠ Impossible ⚠



The four ways to build a regexp

Atoms

- Each letter is a regexp
- ε is a regexp

Ex: ε, **R** and **F** are regexps

Concatenation $\cdot$

If Q_1 and Q_2 are regexps
Then $Q_1 \cdot Q_2$ is a regexp

Ex: $\mathbf{R} \cdot \mathbf{R}$ and $\mathbf{G} \cdot \mathbf{F}$ are regexps
$(\mathbf{R} \cdot \mathbf{R}) \cdot (\mathbf{G} \cdot \mathbf{F})$ is a regexp

Disjunction +

If Q_1 and Q_2 are regexps
Then $Q_1 + Q_2$ is a regexp

Ex: $\mathbf{R} + \mathbf{R}$ and $\mathbf{G} + \mathbf{F}$ are regexps
$(\mathbf{R} \cdot \mathbf{R}) + (\mathbf{G} \cdot \mathbf{F})$ is a regexp

Kleene star *

If Q is a regexp
Then Q^* is a regexp

Ex: $\mathbf{R}^*$ and $\mathbf{G}^*$ are regexps
$((\mathbf{R}^* \cdot \mathbf{G}) + \mathbf{F})^*$ is a regexp

The four ways to build a 2-way regexp

Atoms

- Each forward or backward letter is a regexp
- ε is a regexp

Ex: ε, $\mathbf{R}$, $\overline{\mathbf{R}}$, $\overline{\mathbf{G}}$ and $\mathbf{F}$ are regexps

Disjunction +

If Q_1 and Q_2 are regexps
Then $Q_1 + Q_2$ is a regexp

Ex: $\mathbf{R} + \overline{\mathbf{R}}$ and $\mathbf{G} + \mathbf{F}$ are regexps
$(\mathbf{R} \cdot \mathbf{R}) + (\mathbf{G} \cdot \mathbf{F})$ is a regexp

Concatenation ·

If Q_1 and Q_2 are regexps
Then $Q_1 \cdot Q_2$ is a regexp

Ex: $\mathbf{R} \cdot \mathbf{R}$ and $\mathbf{G} \cdot \mathbf{F}$ are regexps
$(\mathbf{R} \cdot \mathbf{R}) \cdot (\overline{\mathbf{G}} \cdot \overline{\mathbf{F}})$ is a regexp

Kleene star *

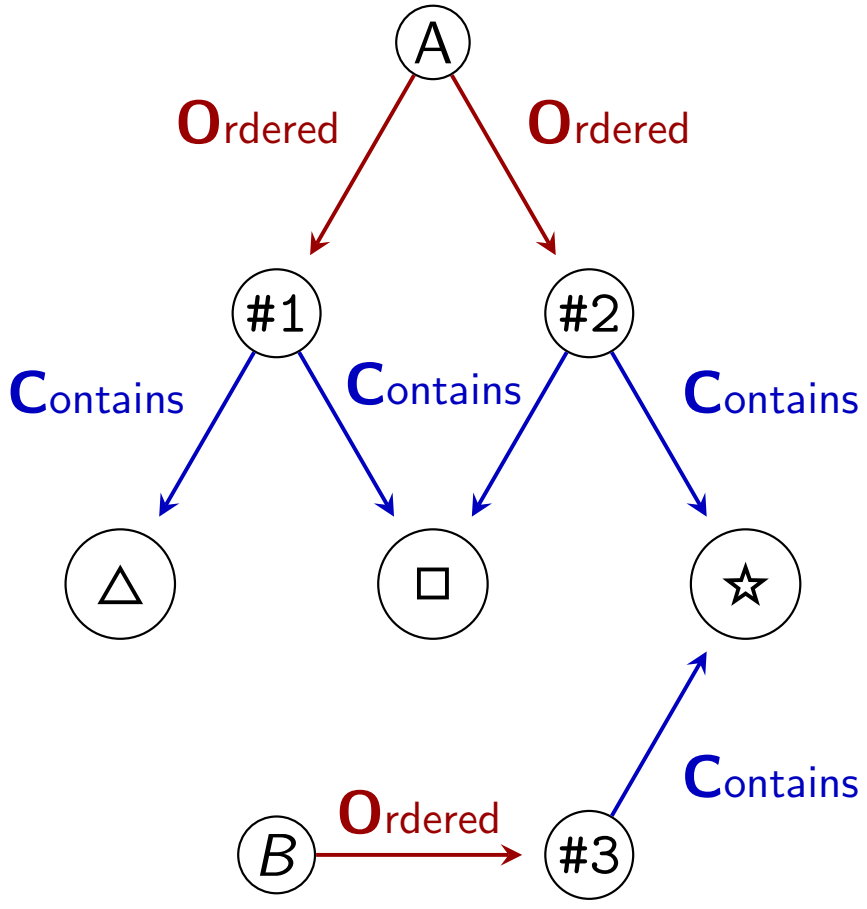
If Q is a regexp
Then Q^* is a regexp

Ex: $\mathbf{R}^*$ and $\mathbf{G}^*$ are regexps
$((\mathbf{R}^* \cdot \overline{\mathbf{G}}) + \mathbf{F})^*$ is a regexp

2RPQ for request 1

Write a 2RPQ to "extract"

1. Products that were ordered twice (that is ☆ and □).

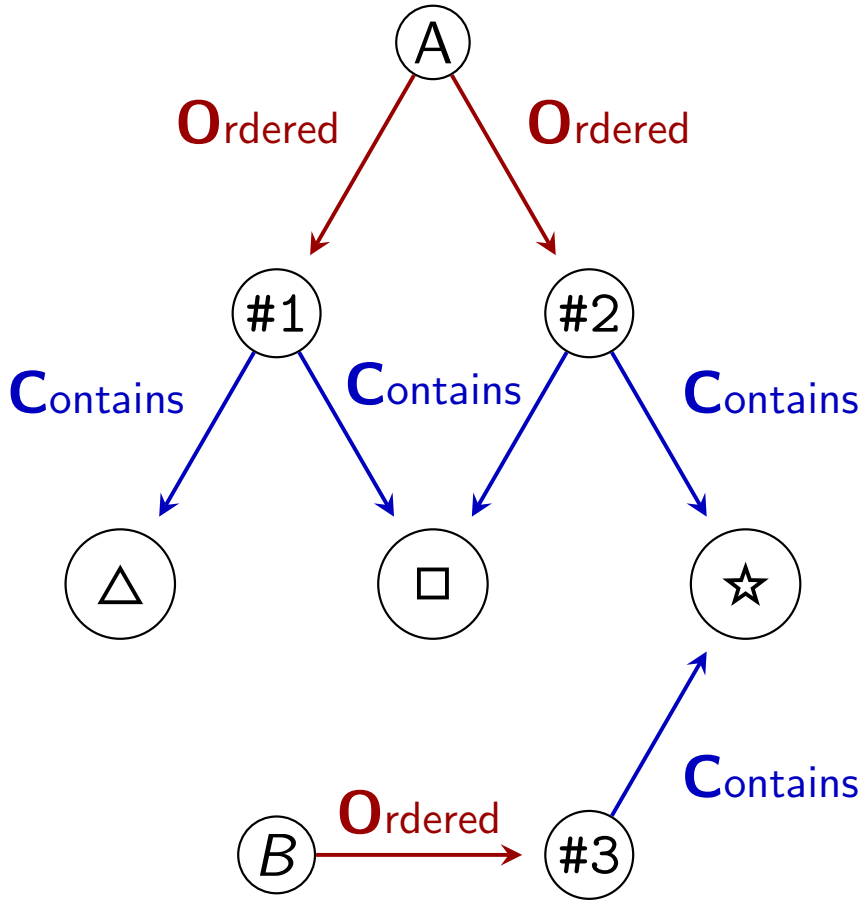


2RPQ for request 1

Write a 2RPQ to "extract"

1. Products that were ordered twice (that is ☆ and □).

Answer: $Q_{18} = \mathbf{C} \cdot \bar{\mathbf{C}}$



2RPQ for request 1

Write a 2RPQ to "extract"

1. Products that were ordered twice (that is ☆ and □).

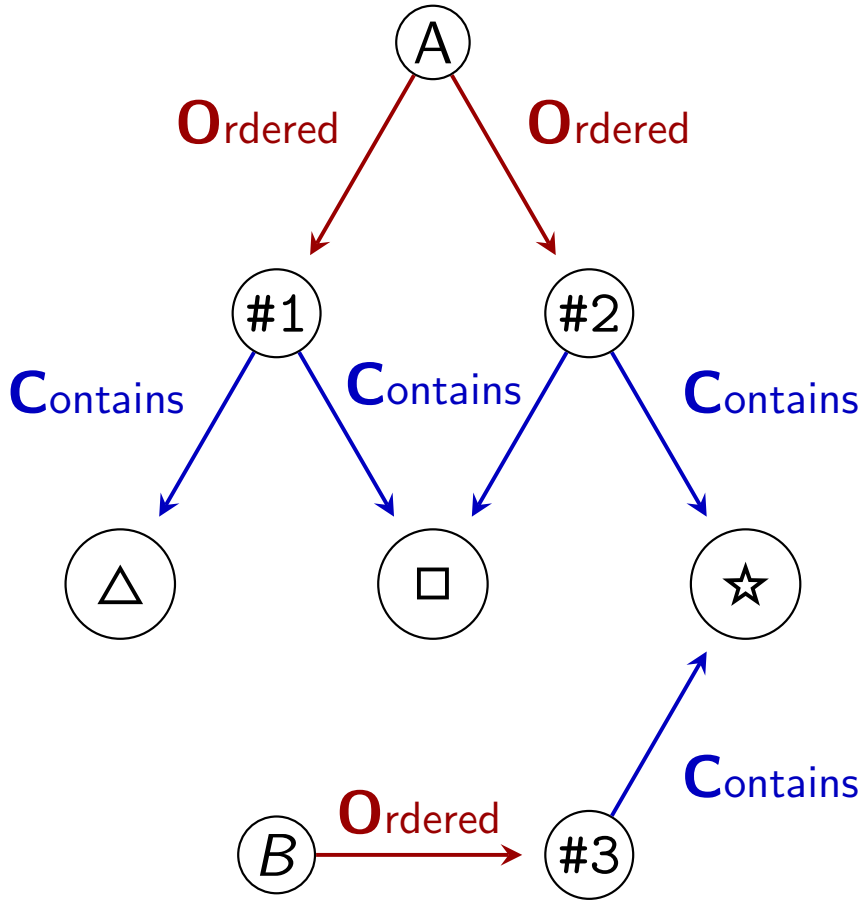
Answer: $Q_{18} = \mathbf{C} \cdot \bar{\mathbf{C}}$

- Walks and matches now may contain backward edges

Matches to Q_{18}:

#1 → □ ← #2, #3 → ☆ ← #2

#1 → △ ← #1, etc.



2RPQ for request 1

Write a 2RPQ to "extract"

1. Products that were ordered twice (that is ☆ and □).

Answer: $Q_{18} = \mathbf{C} \cdot \bar{\mathbf{C}}$

- Walks and matches now may contain backward edges

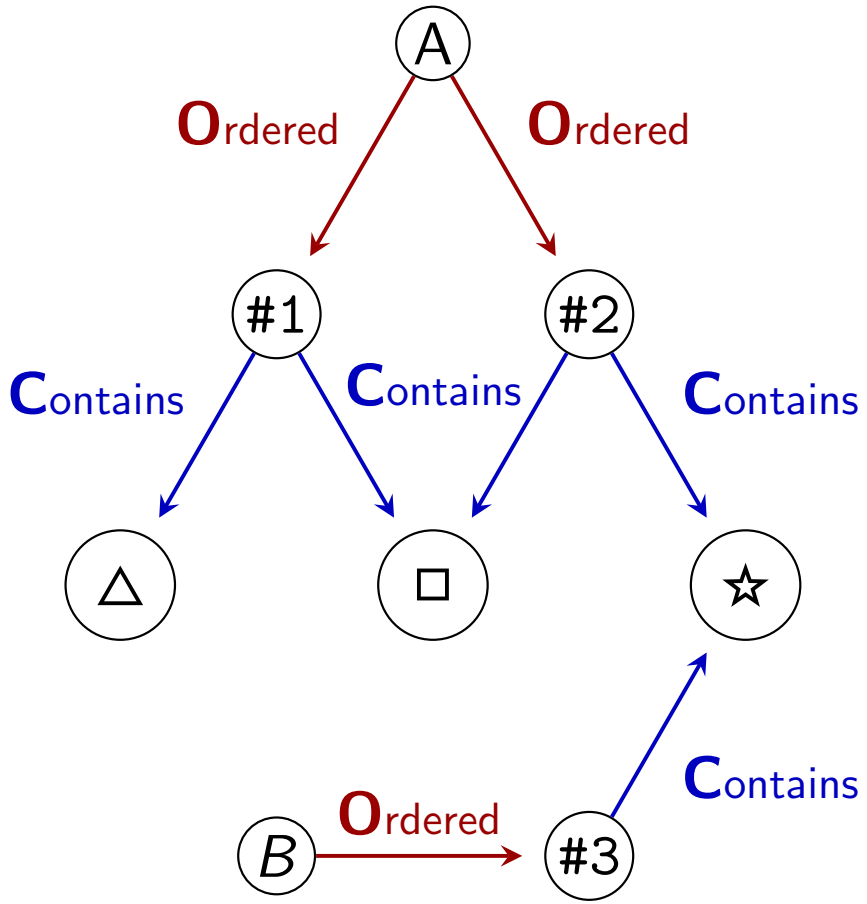
Matches to Q_{18}:

#1 → □ ← #2, #3 → ☆ ← #2

#1 → △ ← #1, etc.

- Trail forbids using the same edge backward and forward

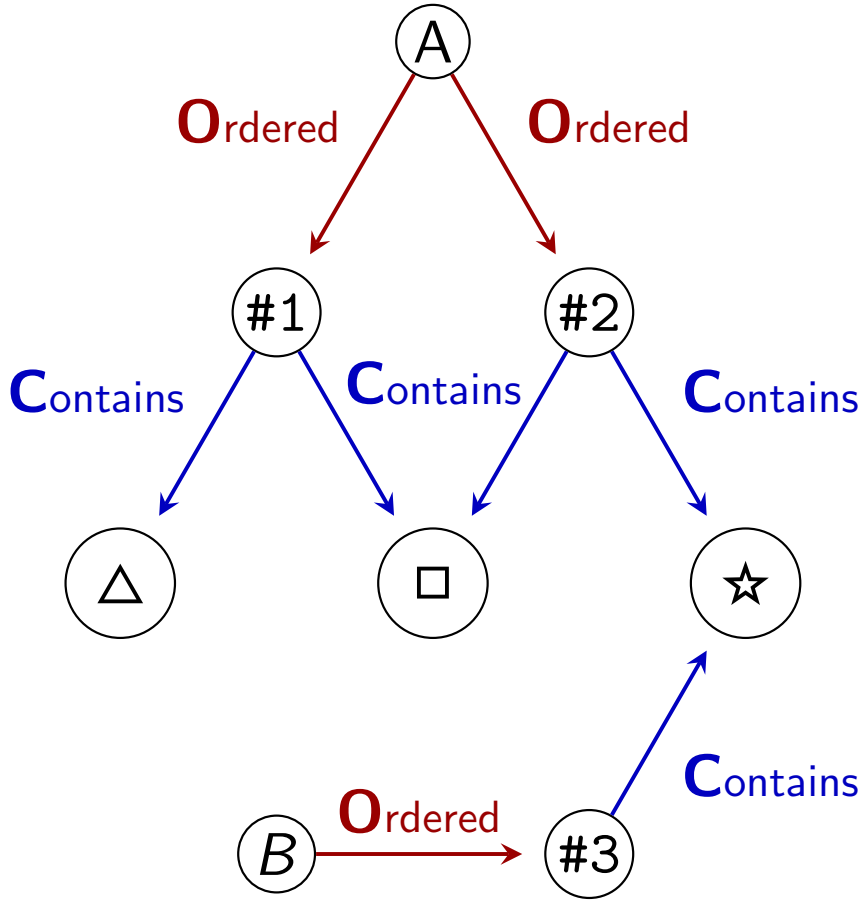
Under trail, Q_{18} returns walks with ☆ and □ as the middle vertex.



2RPQ for request 2

Write a 2RPQ to extract

2 Triples (x, y, z) such that x ordered y and z in the same order. Ex: $(A, \triangle, \square)$.

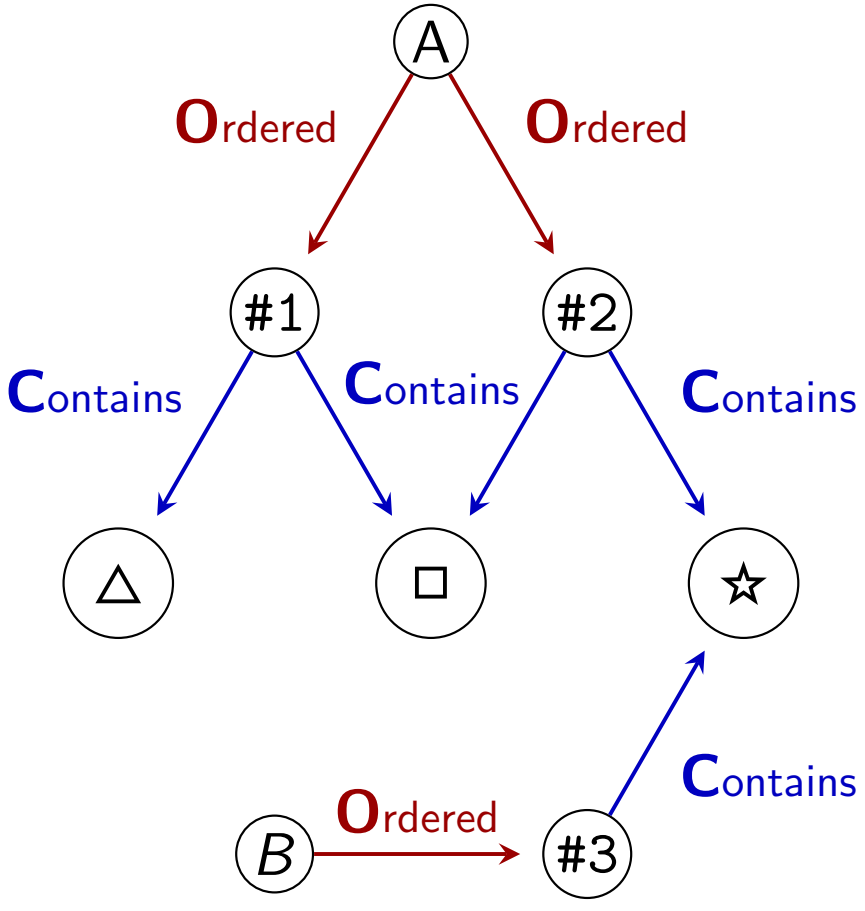


2RPQ for request 2

Write a 2RPQ to extract

2 Triples (x, y, z) such that x ordered y and z in the same order. Ex: $(A, \triangle, \square)$.

⚠ Still impossible ⚠



Conjunction of Regular Path Queries (CRPQs)

Definition

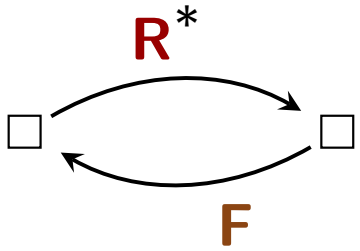
CRPQ = graph pattern matching
that is, a graph where each edge bears an RPQ

Conjunction of Regular Path Queries (CRPQs)

Definition

CRPQ = graph pattern matching
that is, a graph where each edge bears an RPQ

Use-case 1: cycles

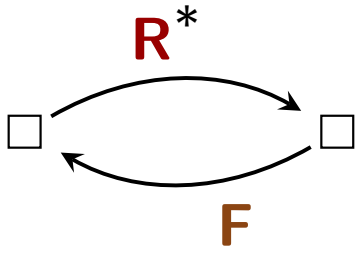


Conjunction of Regular Path Queries (CRPQs)

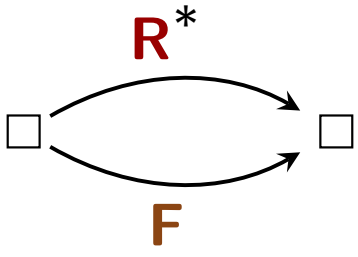
Definition

CRPQ = graph pattern matching
that is, a graph where each edge bears an RPQ

Use-case 1: cycles



Use-case 2:
Multi-way

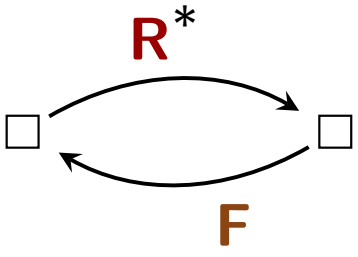


Conjunction of Regular Path Queries (CRPQs)

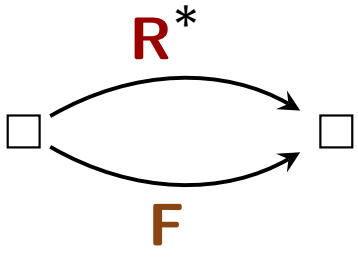
Definition

CRPQ = graph pattern matching
that is, a graph where each edge bears an RPQ

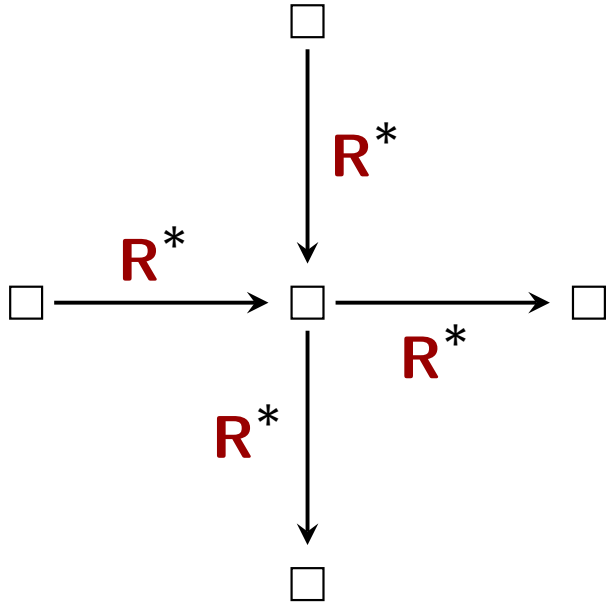
Use-case 1: cycles



Use-case 2: Multi-way



Use-case 3: Cross



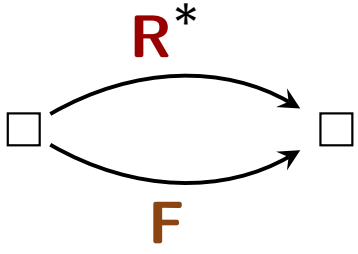
Matches to CRPQs

Definition

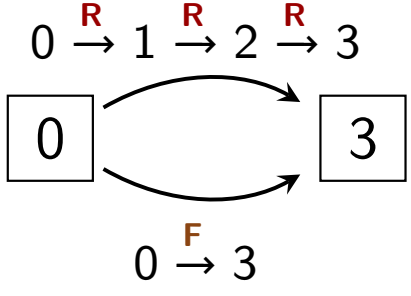
A **match** in graph G to a CRPQ Q consists of

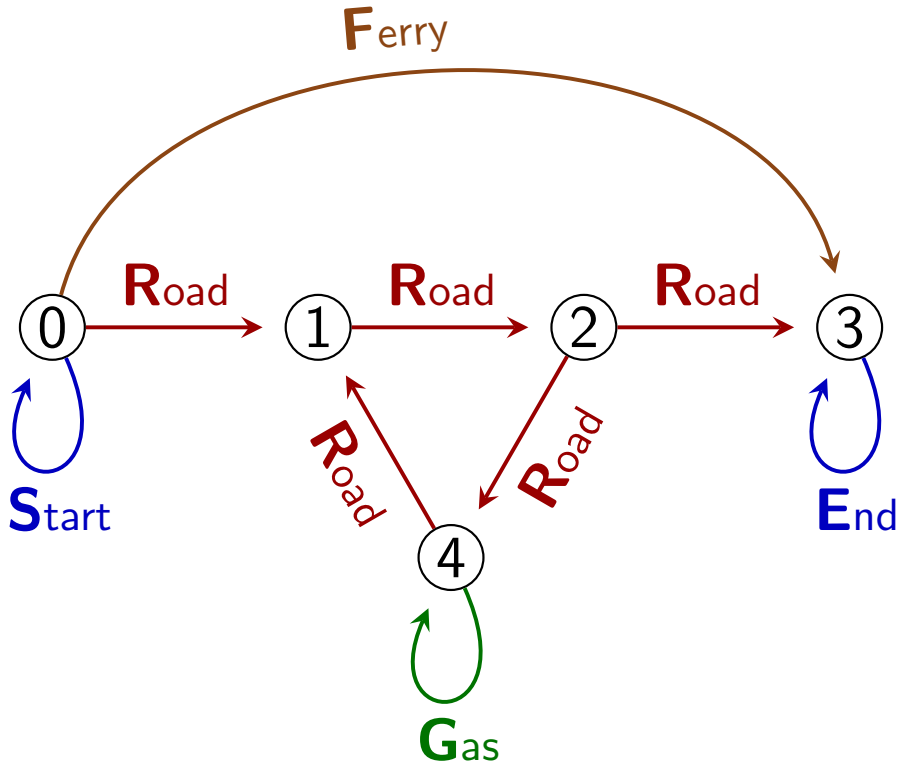
- a map: $\text{Vertex}(Q) \rightarrow \text{Vertex}(G)$
- a map: $\text{Edge}(Q) \rightarrow \text{Walks}(G)$

Query:



A match:





Evaluating CRPQs

Endpoint semantics

- Return the vertex map only

Shortest semantics

Two possibilities

- Shortest for each RPQ
  Ex: GQL, Tigergraph, etc.
- Return the global minimum
  Ex: None?

Evaluating CRPQs

Endpoint semantics

- Return the vertex map only

Shortest semantics

Two possibilities

- Shortest for each RPQ
  Ex: GQL, Tigergraph, etc.
- Return the global minimum
  Ex: None?

Trail semantics

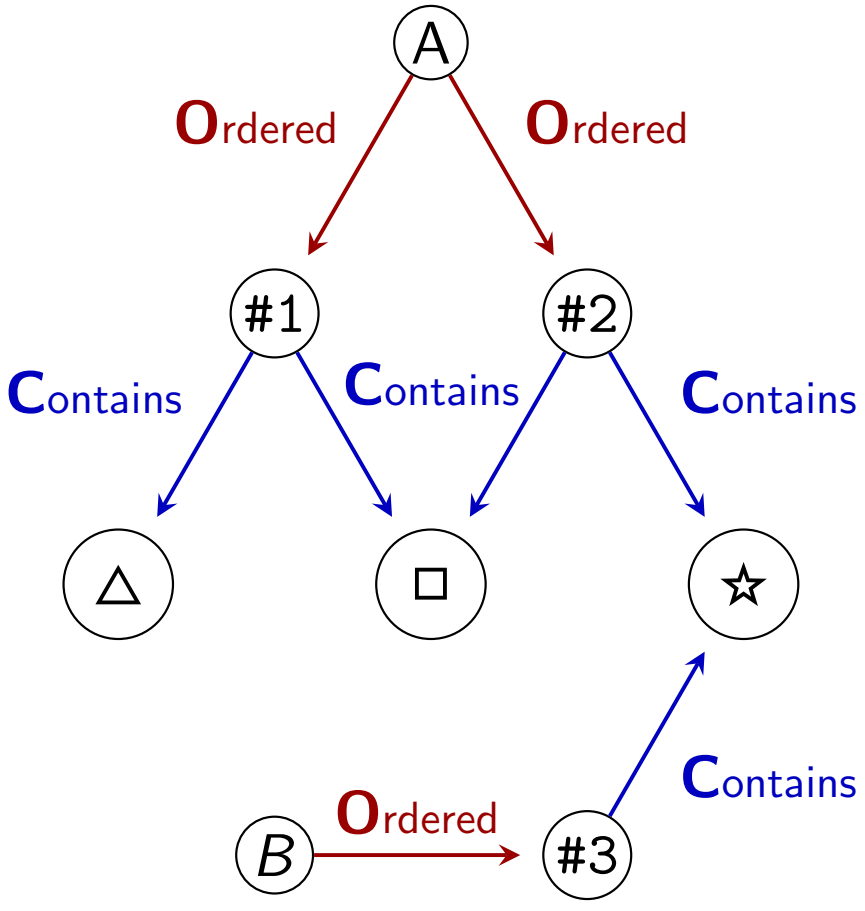
Two possibilities:

- No edge repetition for each RPQ
  Ex: GQL
- No edge repetition overall
  Ex: Cypher, GQL

CRPQ for request 2

Write a 2RPQ to extract

2 Triples (x, y, z) such that x ordered y and z in the same order. Ex: $(A, \triangle, \square)$.

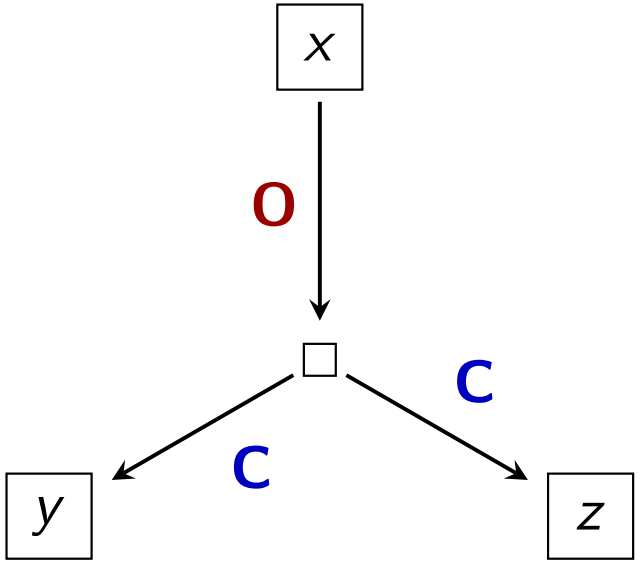


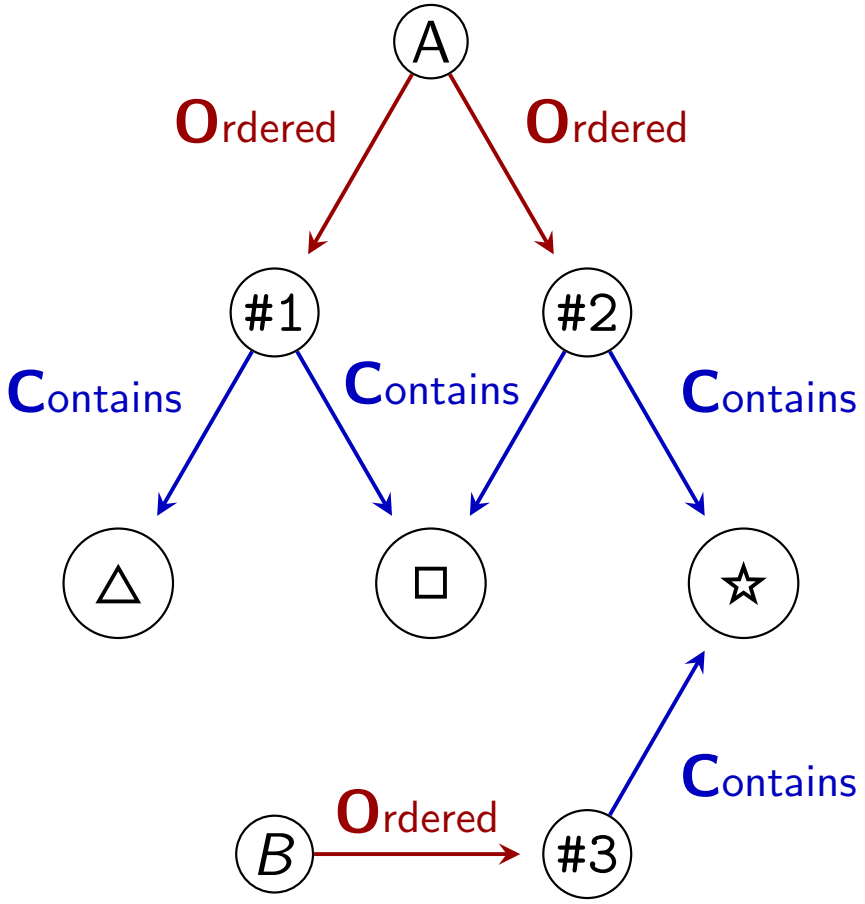
CRPQ for request 2

Write a 2RPQ to extract

2 Triples (x, y, z) such that x ordered y and z in the same order. Ex: $(A, \triangle, \square)$.

Answer:





Which semantics?

Essential knowledge in part I

- *Labeled Graph* data model.
- Definition and language denoted by a regexp (and a 2-way regexp).
- Writing abstract and concrete RPQs, 2RPQ, CRPQs.
- Computing matches for RPQs, 2RPQ, CRPQs.
- Concept of product graph.
- What is an RPQ semantics and why we need one.
- The three common RPQ semantics:
  - Definition
  - Evaluating RPQs, 2RPQs under each semantics
  - Usage
  - Difference

Part II: Property Graphs

Part II: Property Graphs

1. Data model

Components of a property graph

A **node** (≈**vertex**) encodes a complex values.
It bears **labels** for grouping.

Ex: t carries `Teacher`, `Person`
c carries `Course`

Components of a property graph

A **node** (≈**vertex**) encodes a complex values.
It bears **labels** for grouping.

Ex: t carries `Teacher`, `Person`
c carries `Course`

A **Relation** (≈**edge**) connects **nodes**.
It bears one **type** (≈**label**) provides the nature of the relation.

Ex: $e = t \xrightarrow{\texttt{TEACHES}} c$

Components of a property graph

A **node** (≈**vertex**) encodes a complex values.
It bears **labels** for grouping.

Ex: t carries `Teacher`, `Person`
c carries `Course`

A **Relation** (≈**edge**) connects **nodes**.
It bears one **type** (≈**label**) provides the nature of the relation.

Ex: $e = t \xrightarrow{\texttt{TEACHES}} c$

A **property** describes an aspect of a **node** or an **relation**.
It maps

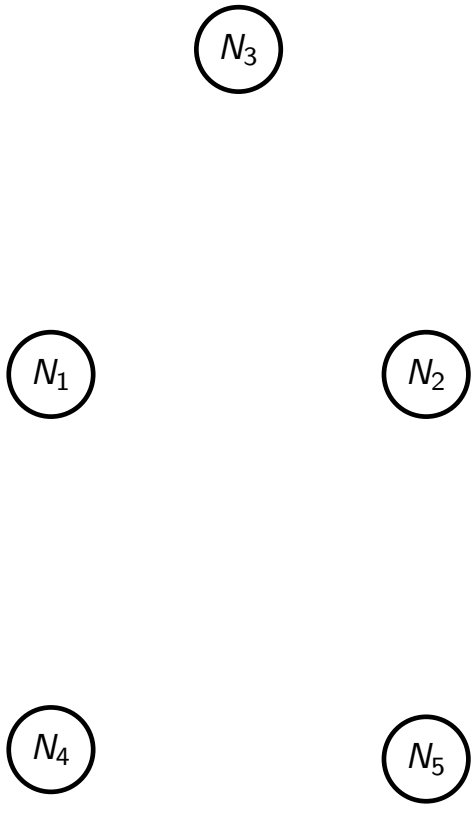
- a **key** (described aspect)
- to a **pure value** (description)

Ex: t has `name`: `"Victor"`
e has `since`: 2023

A **pure value** (int, string, etc.) contains all the information about itself.

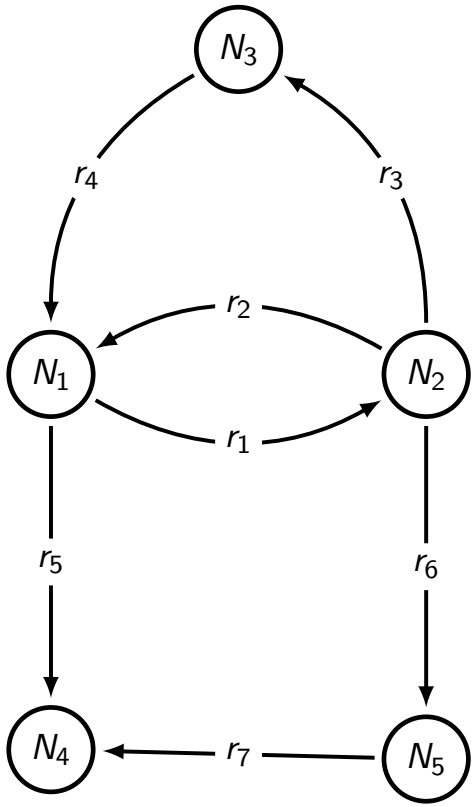
Ex: `"Victor"` has 6 letters

First example of a property graph



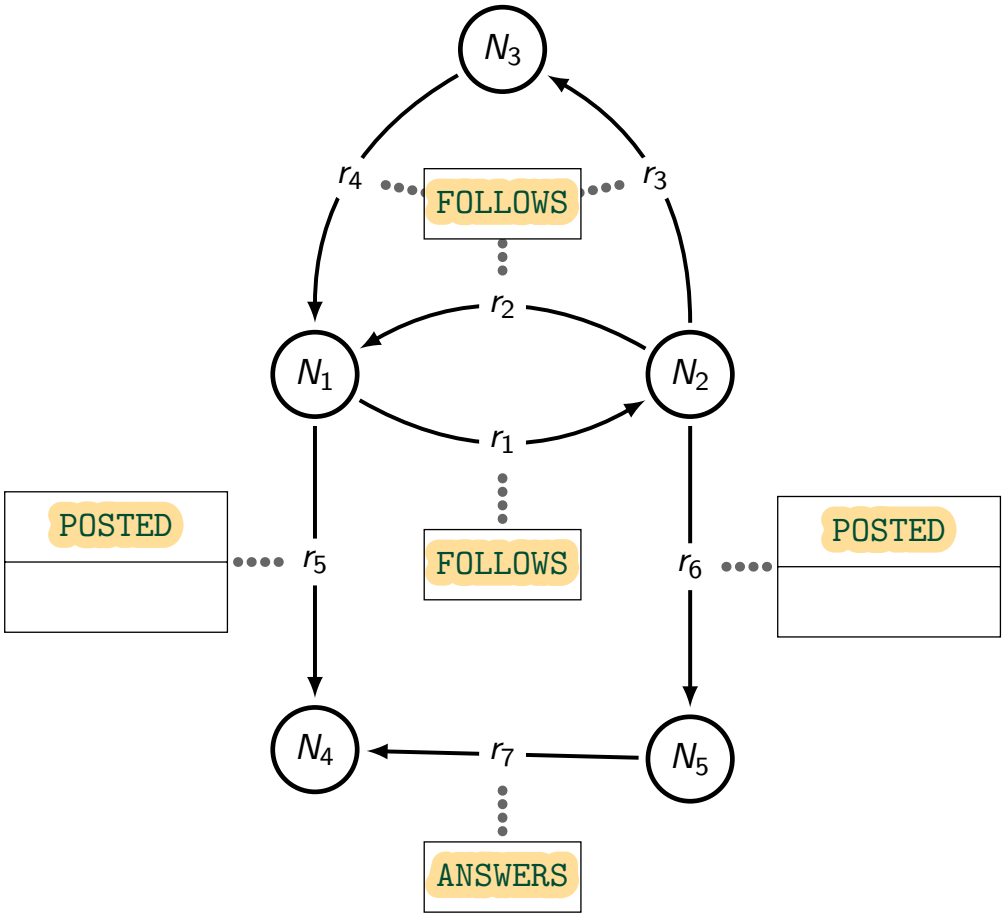
- Nodes : $N_1, N_2, \cdots, N_5$

First example of a property graph



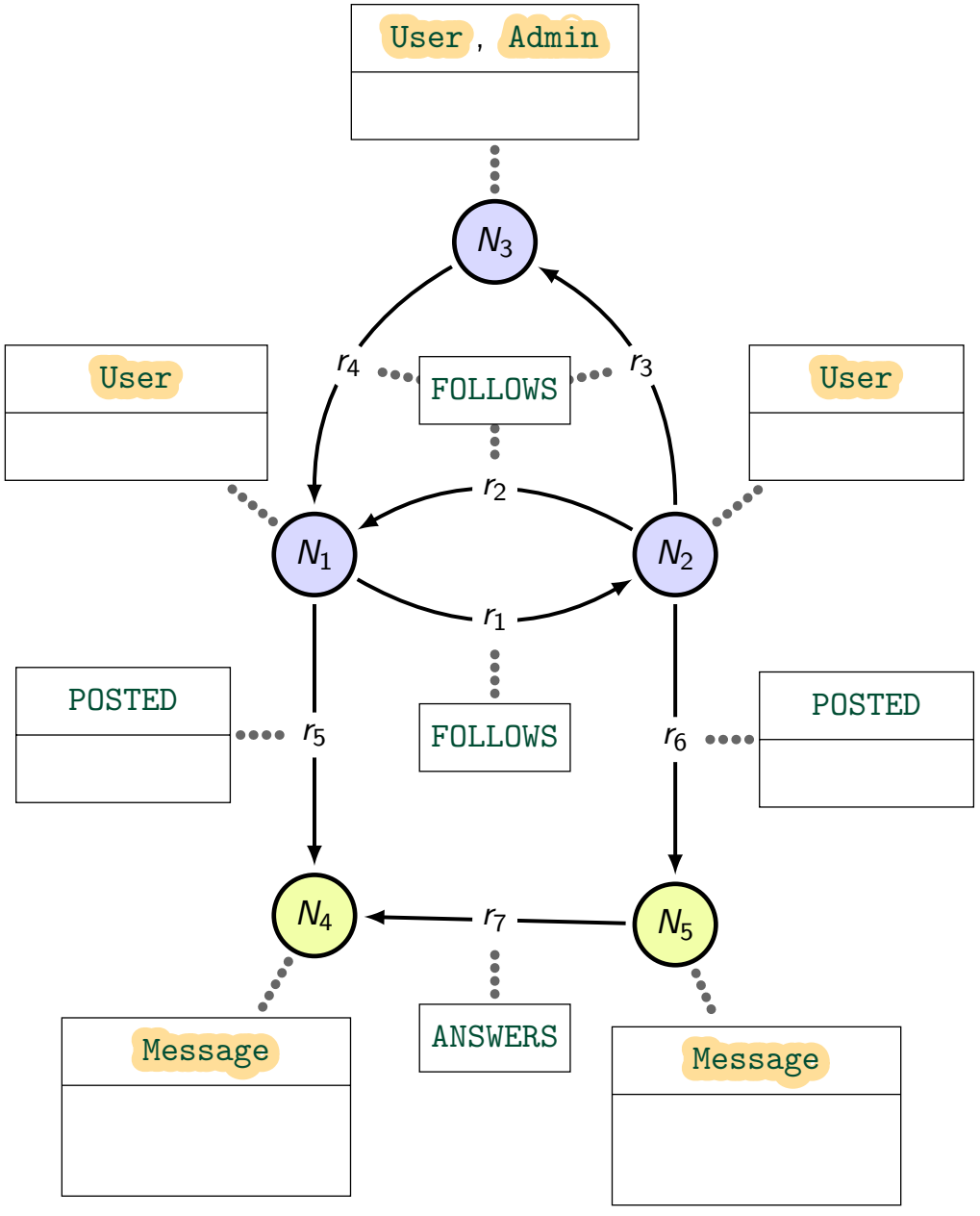
- Nodes : $N_1, N_2, \cdots, N_5$
- Relations : $r_1, r_2, \cdots, r_7$

First example of a property graph



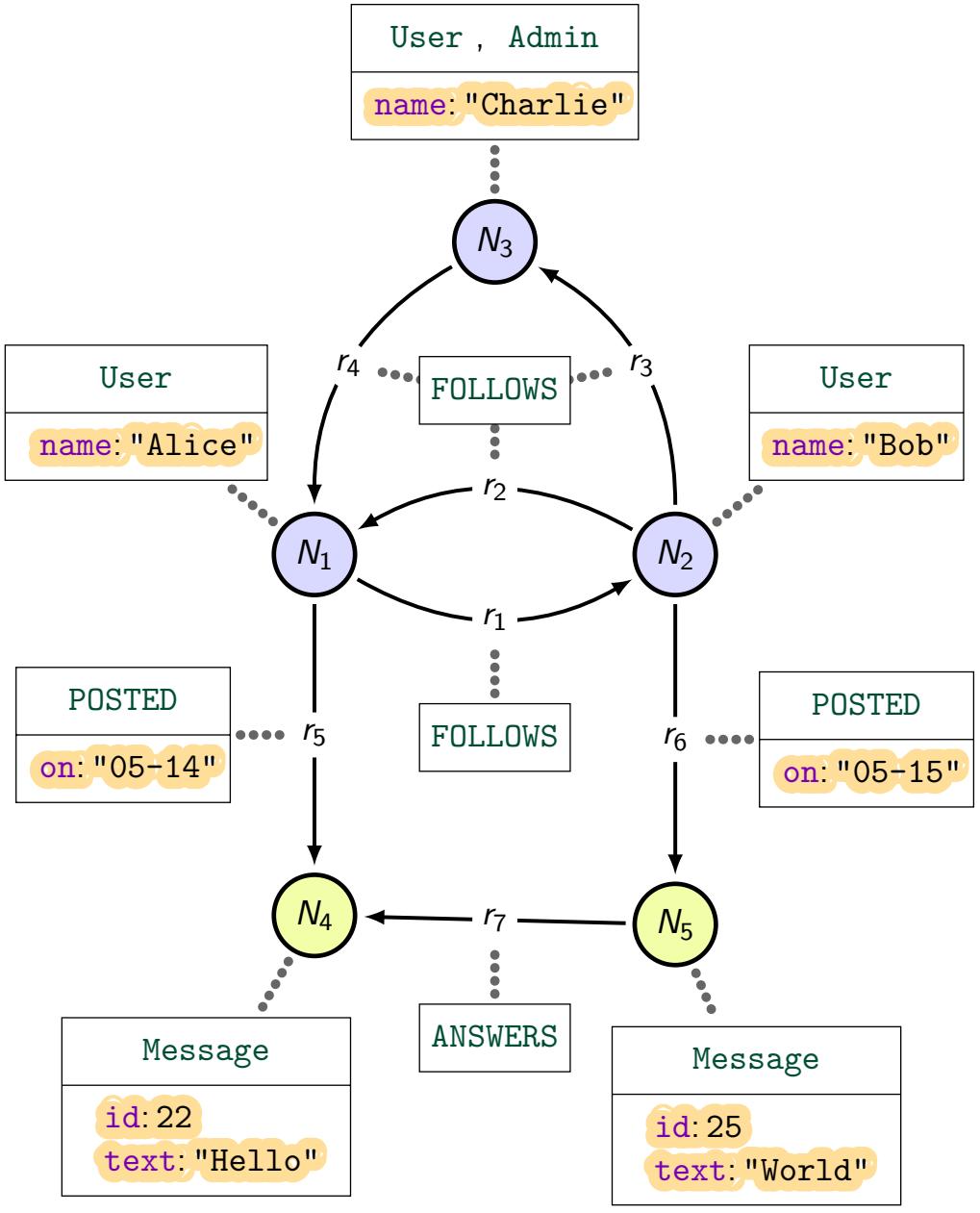
- Nodes : $N_1, N_2, \cdots, N_5$
- Relations : $r_1, r_2, \cdots, r_7$
- Types: `FOLLOWS`, `POSTED`, `ANSWERS`

First example of a property graph



- Nodes : $N_1, N_2, \cdots, N_5$
- Relations : $r_1, r_2, \cdots, r_7$
- Types: `FOLLOWS`, `POSTED`, `ANSWERS`
- Labels: `User`, `Admin`, `Message`

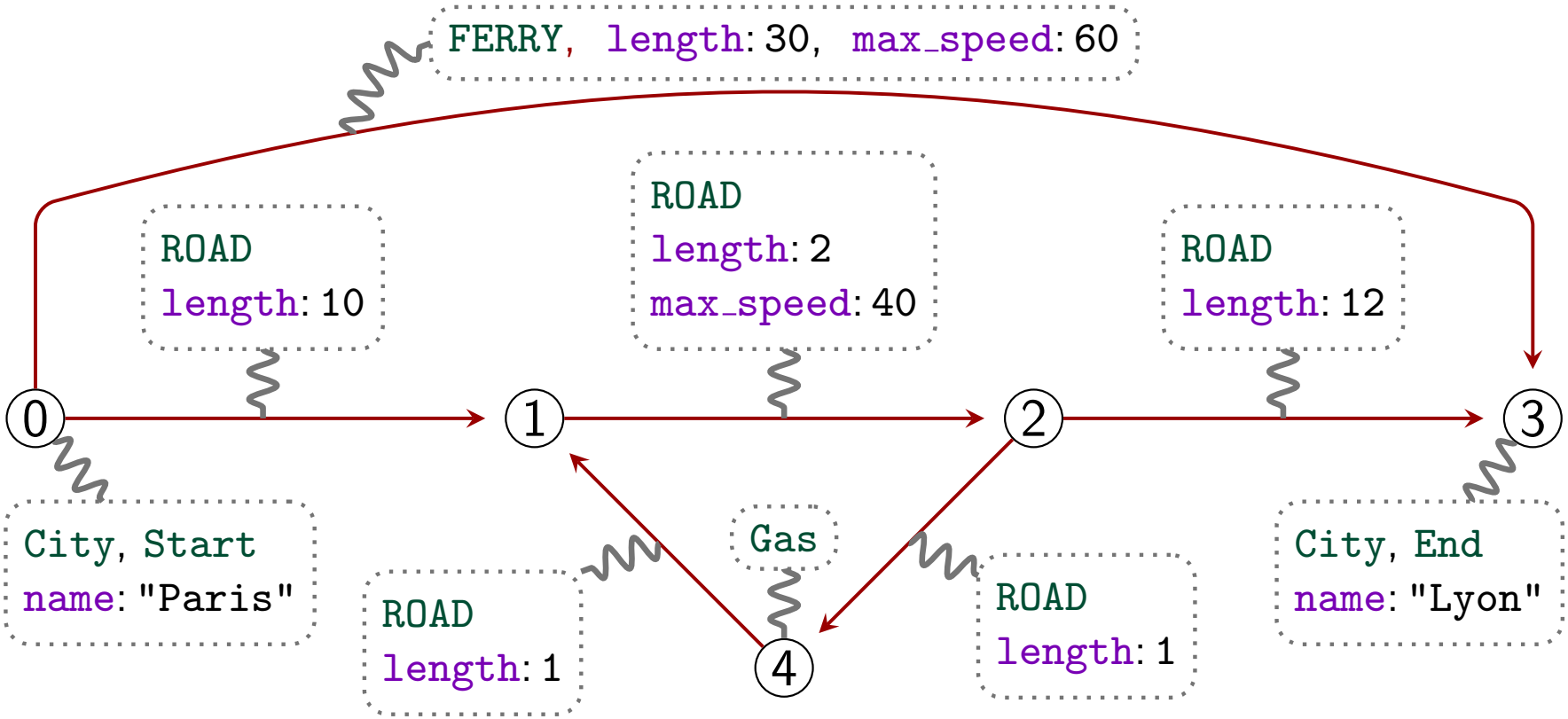
First example of a property graph



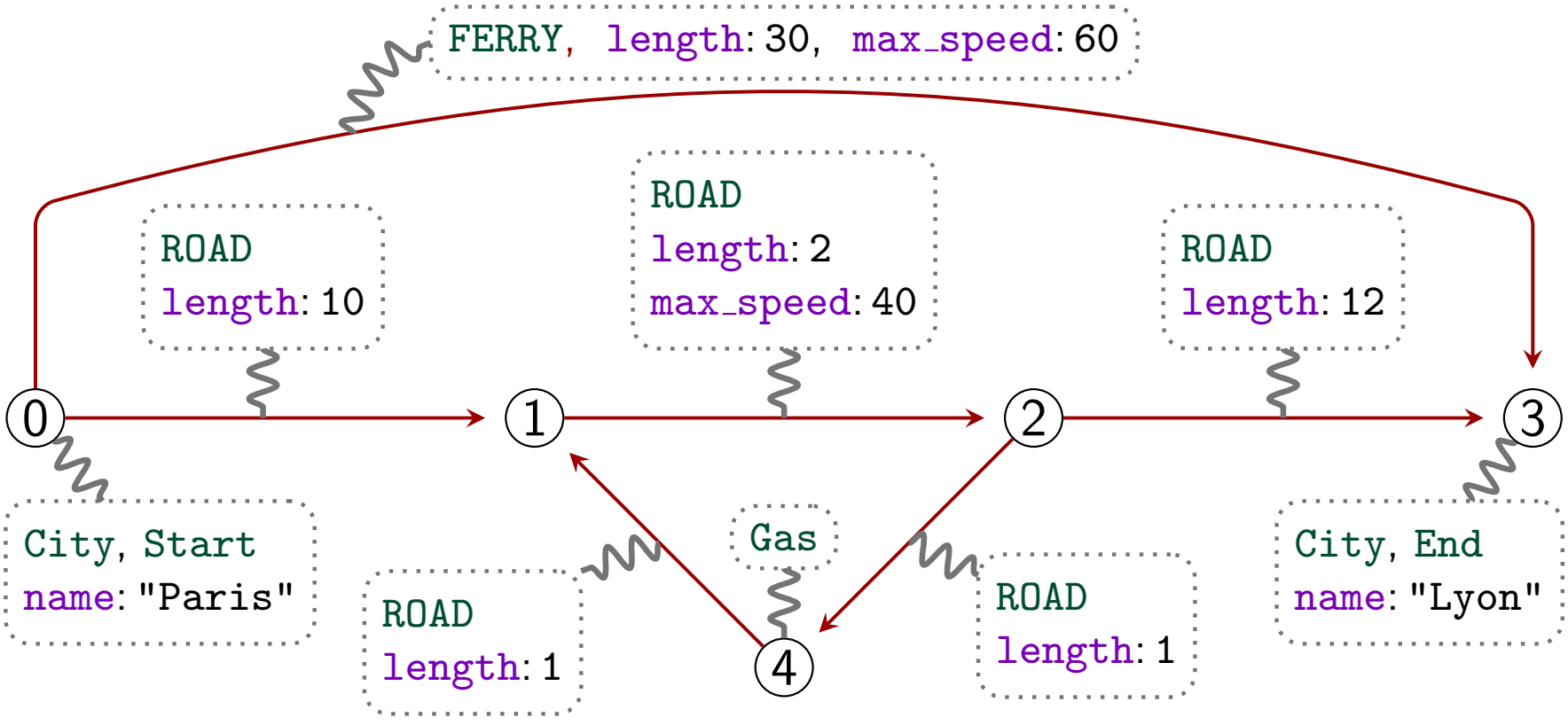
- Nodes : $N_1, N_2, \cdots, N_5$
- Relations : $r_1, r_2, \cdots, r_7$
- Types: `FOLLOWS`, `POSTED`, `ANSWERS`
- Labels: `User`, `Admin`, `Message`
- Properties, that is Key-Value pairs:
  - `name`: "Alice"
  - `id`: 22
  - `text`: "Hello"

  etc.

Second example of a property graph



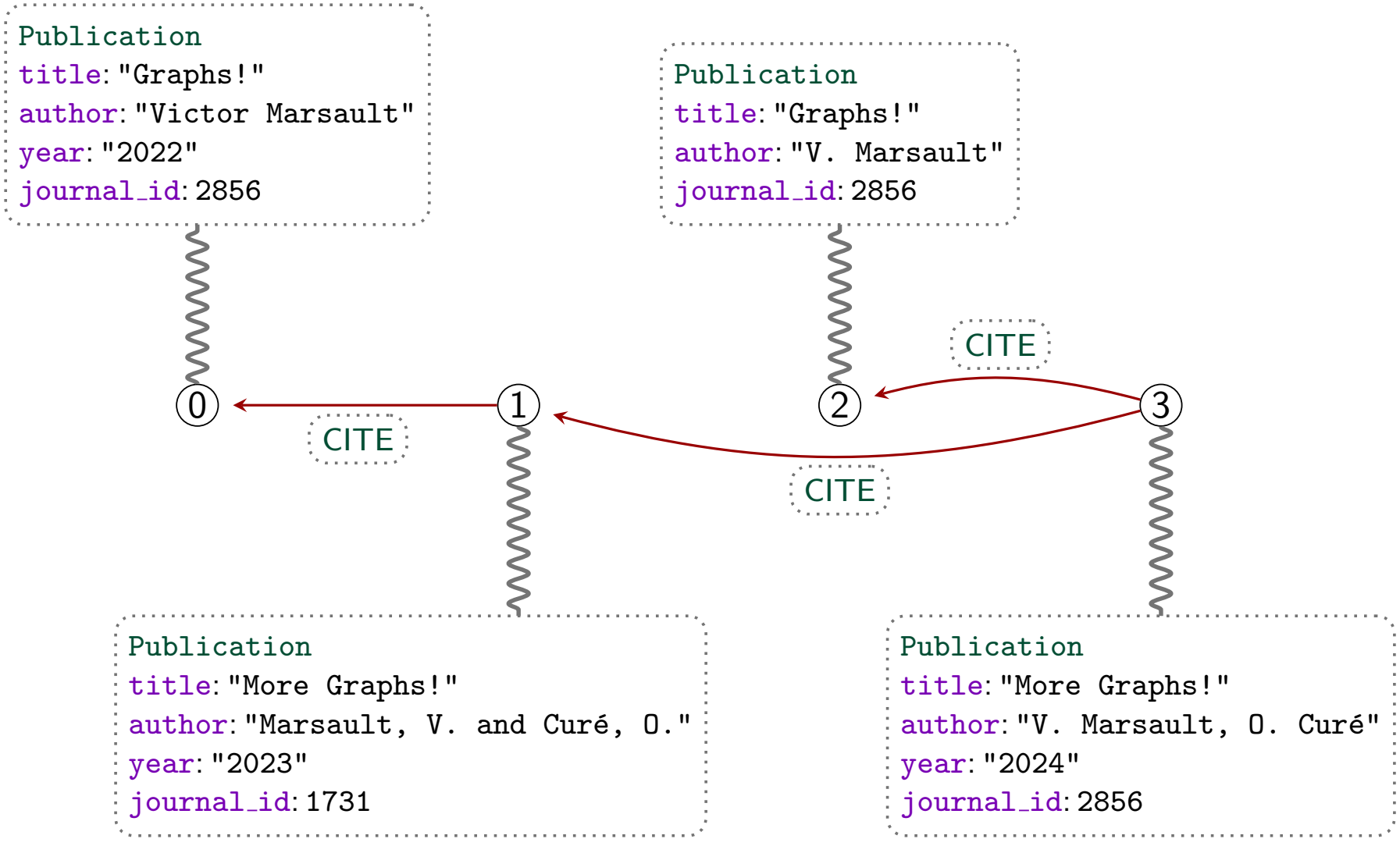
Property graphs are very flexible



- Relations with the same type may have different property keys
- Nodes may have any number of labels and property keys

Third example of a property graph

Exercice: What's wrong with this property graph ? Fix it !



Part II: Property Graphs

2. Translation: Graph → Tables

Preliminary poll: PG vs Relational ?

1. The property graphs data model is more expressive.
2. The relational data model (that is, tables) is stronger.
3. They are both as expressive.

Preliminary poll: PG vs Relational ?

1. The property graphs data model is more expressive.
2. The relational data model (that is, tables) is stronger.
3. They are both as expressive.

 Counterintuitively, tables are way more general 

Preliminary poll: PG vs Relational ?

1. The property graphs data model is more expressive.
2. The relational data model (that is, tables) is stronger.
3. They are both as expressive.

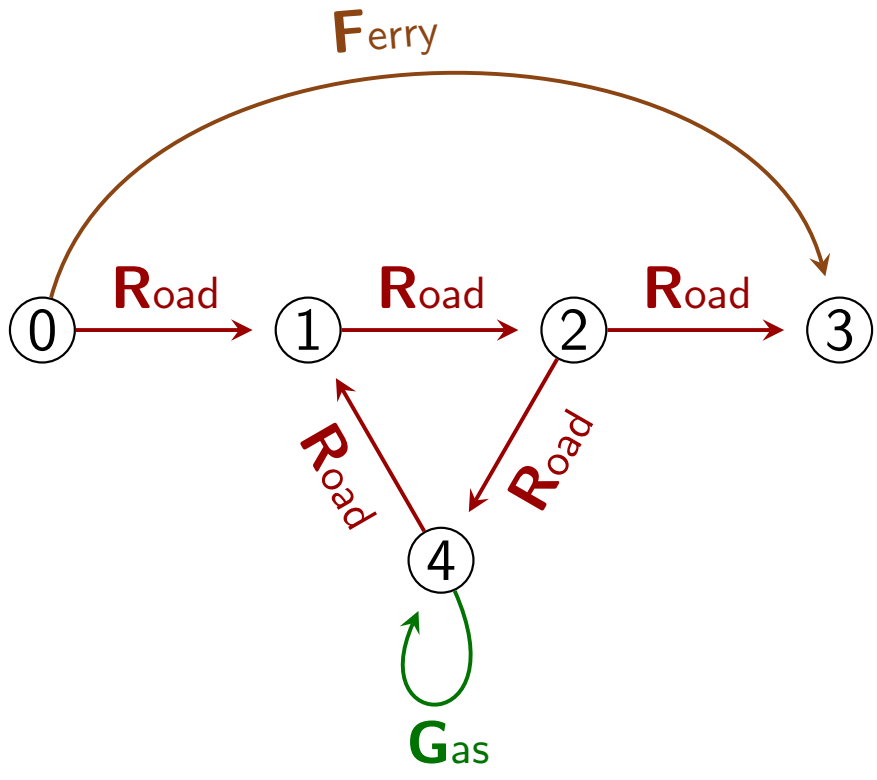
 Counterintuitively, tables are way more general 

Consequences:

- Translating PGs→Tables is **easy**
- Translating Tables→PGs is **not always directly possible**

Translation: Graph to Tables (1)

Can a graph be stored in tables?



Translation: Graph to Tables (1)

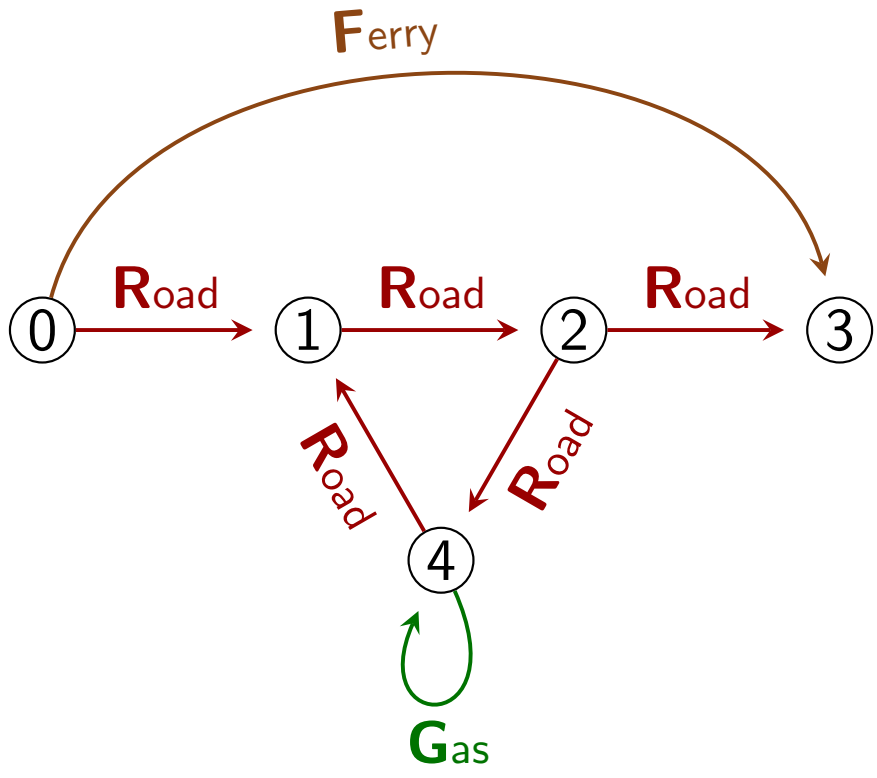
Example – One **Vertex** table with one row per vertex in the graph

Vertex

id
0
1
2
3
4

Road

#src	#tgt
0	1
1	2
2	3
3	2
4	4
4	1



Ferry

#src	#tgt
0	3

Gas

#src	#tgt
4	4

Translation: Graph to Tables (1)

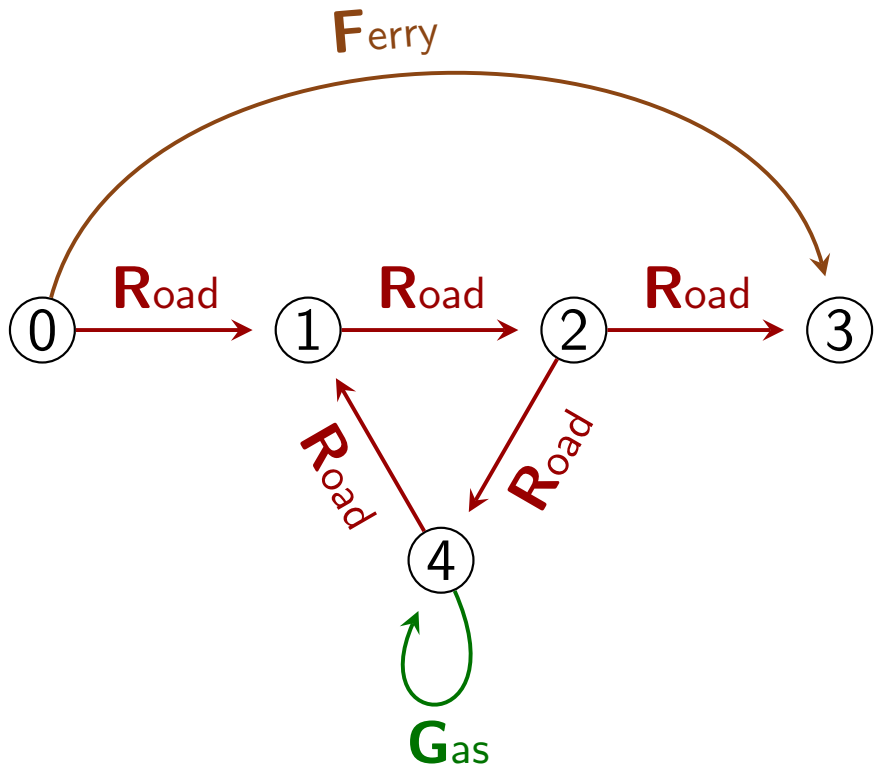
Example – One table for each different label in the graph

Vertex

id
0
1
2
3
4

Road

#src	#tgt
0	1
1	2
2	3
3	4
4	1



Ferry

#src	#tgt
0	3

Gas

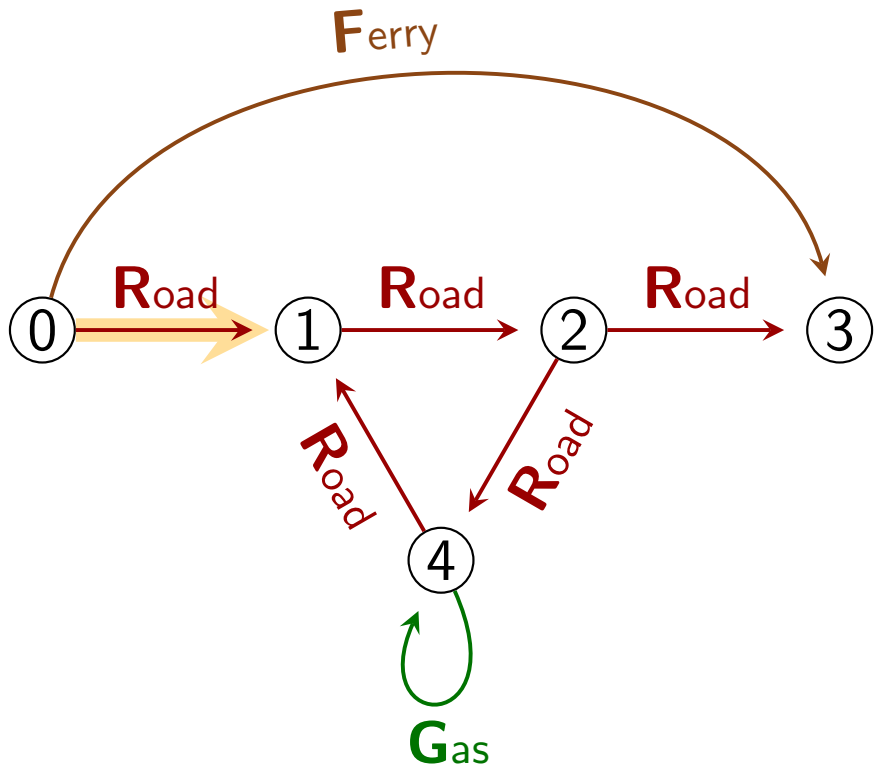
#src	#tgt
4	4

Translation: Graph to Tables (1)

Example – For each edge (i, ℓ, j) in the graph add row (i, j) in table ℓ

Vertex
id
0
1
2
3
4

Road	
#src	#tgt
0	1
1	2
2	3
2	4
4	1



Ferry	
#src	#tgt
0	3

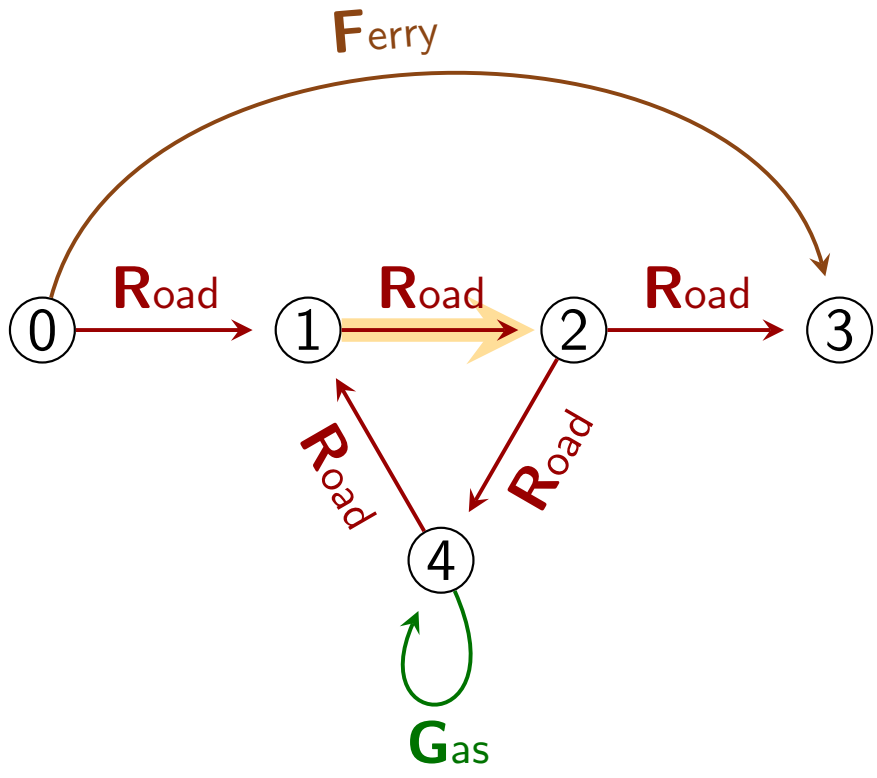
Gas	
#src	#tgt
4	4

Translation: Graph to Tables (1)

Example – For each edge (i, ℓ, j) in the graph add row (i, j) in table ℓ

Vertex
id
0
1
2
3
4

Road	
#src	#tgt
0	1
1	2
2	3
2	4
4	1



Ferry	
#src	#tgt
0	3

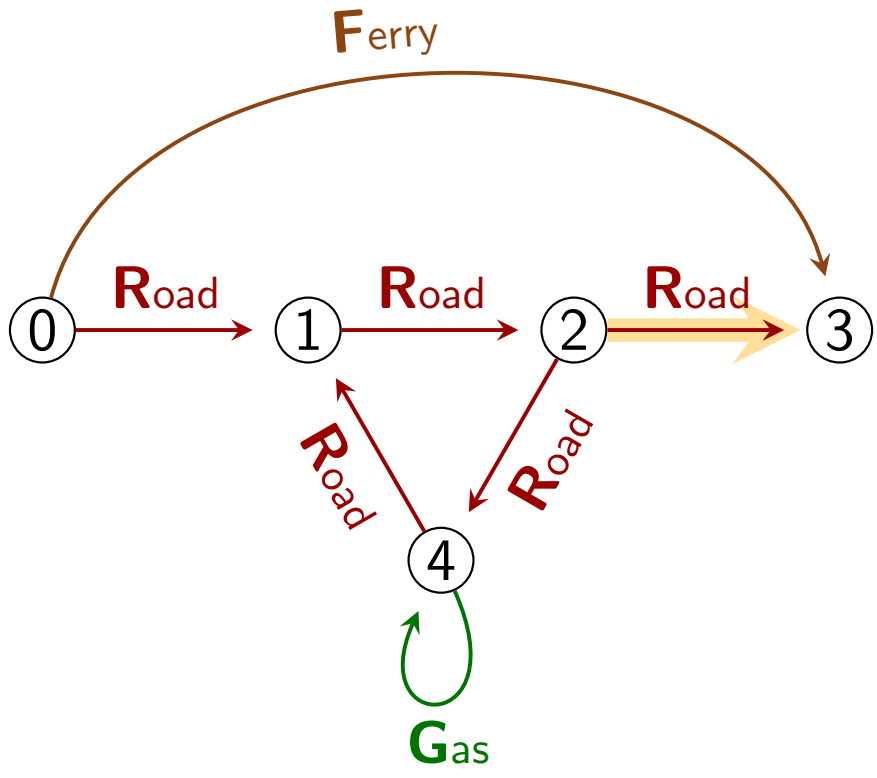
Gas	
#src	#tgt
4	4

Translation: Graph to Tables (1)

Example – For each edge (i, ℓ, j) in the graph add row (i, j) in table ℓ

Vertex
id
0
1
2
3
4

Road	
#src	#tgt
0	1
1	2
2	3
2	4
4	1



Ferry	
#src	#tgt
0	3

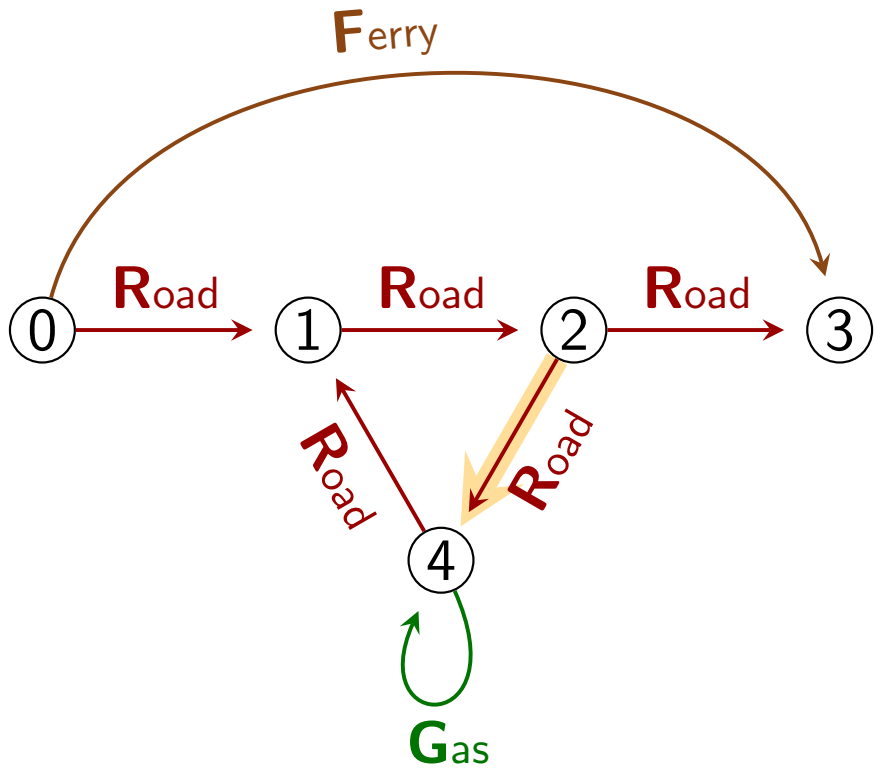
Gas	
#src	#tgt
4	4

Translation: Graph to Tables (1)

Example – For each edge (i, ℓ, j) in the graph add row (i, j) in table ℓ

Vertex
id
0
1
2
3
4

Road	
#src	#tgt
0	1
1	2
2	3
2	4
4	1



Ferry	
#src	#tgt
0	3

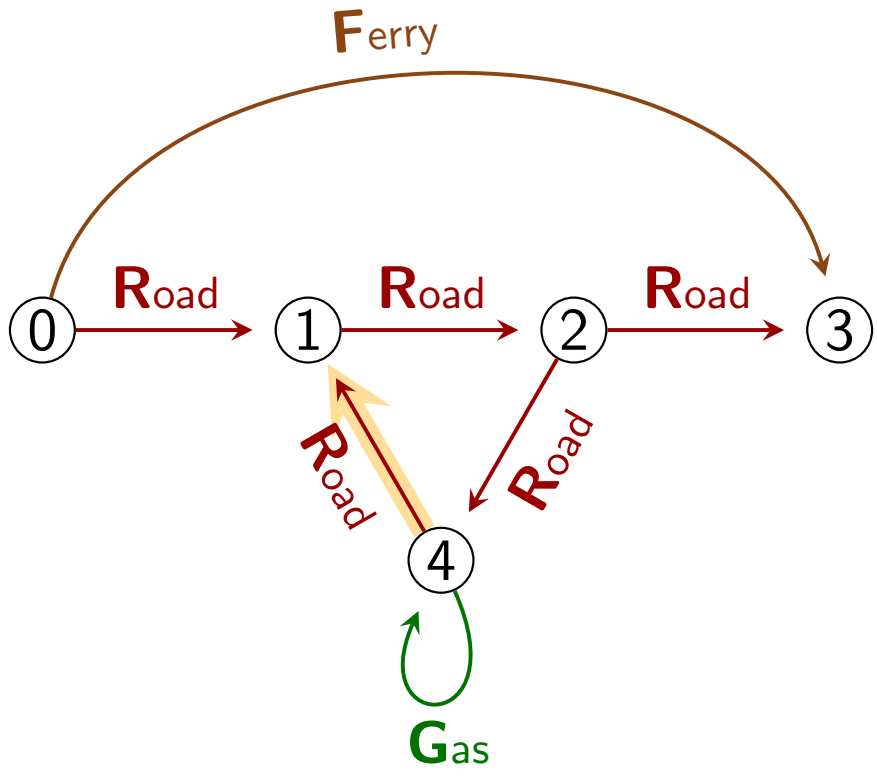
Gas	
#src	#tgt
4	4

Translation: Graph to Tables (1)

Example – For each edge (i, ℓ, j) in the graph add row (i, j) in table ℓ

Vertex
id
0
1
2
3
4

Road	
#src	#tgt
0	1
1	2
2	3
2	4
4	1



Ferry	
#src	#tgt
0	3

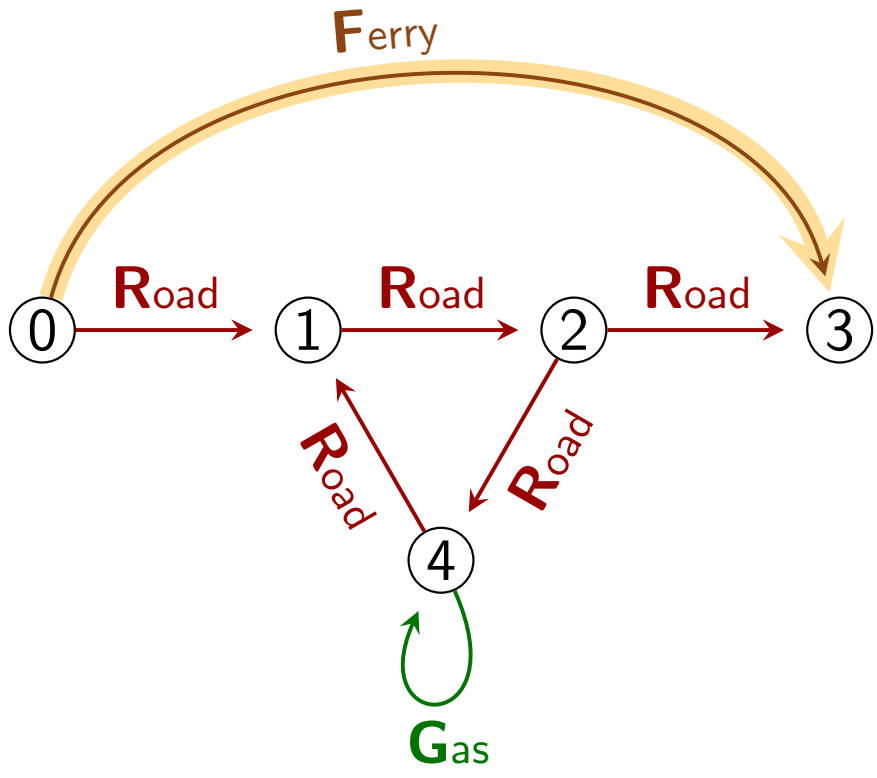
Gas	
#src	#tgt
4	4

Translation: Graph to Tables (1)

Example – For each edge (i, ℓ, j) in the graph add row (i, j) in table ℓ

Vertex
id
0
1
2
3
4

Road	
#src	#tgt
0	1
1	2
2	3
2	4
4	1



Ferry	
#src	#tgt
0	3

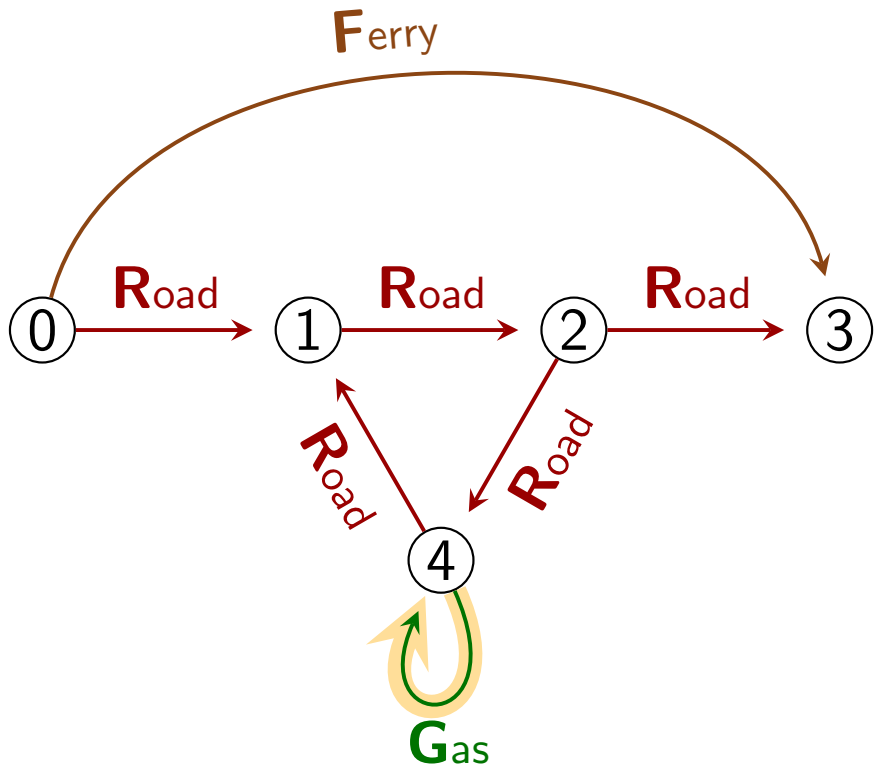
Gas	
#src	#tgt
4	4

Translation: Graph to Tables (1)

Example – For each edge (i, ℓ, j) in the graph add row (i, j) in table ℓ

Vertex
id
0
1
2
3
4

Road	
#src	#tgt
0	1
1	2
2	3
2	4
4	1



Ferry	
#src	#tgt
0	3

Gas	
#src	#tgt
4	4

Translation: Graph to Tables (2)

Principles of the translation

We start from a graph (V, L, E)
Since V is finite we may enumerate it: $V = \{v_1, \ldots, v_n\}$

One table for vertices

Vertex
id
0
1
⋮
n

One table per label ℓ in L

ℓ	
#src	#tgt
⋮	⋮
i	j
⋮	⋮

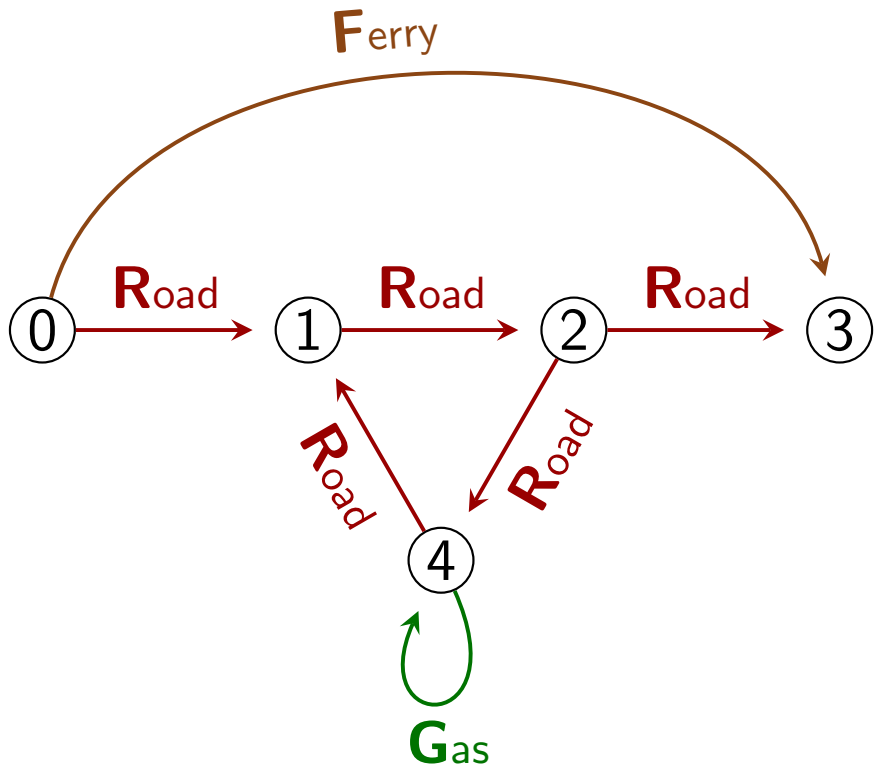
Table ℓ contains (i, j)
$\iff (v_i, \ell, v_j) \in E$

Translation: RPQ to SQL

Exercice: translate **RRRGRRR** and $(\mathbf{R}+\mathbf{F})^*\mathbf{G}(\mathbf{R}+\mathbf{F})^*$ in SQL

Vertex
id
0
1
2
3
4

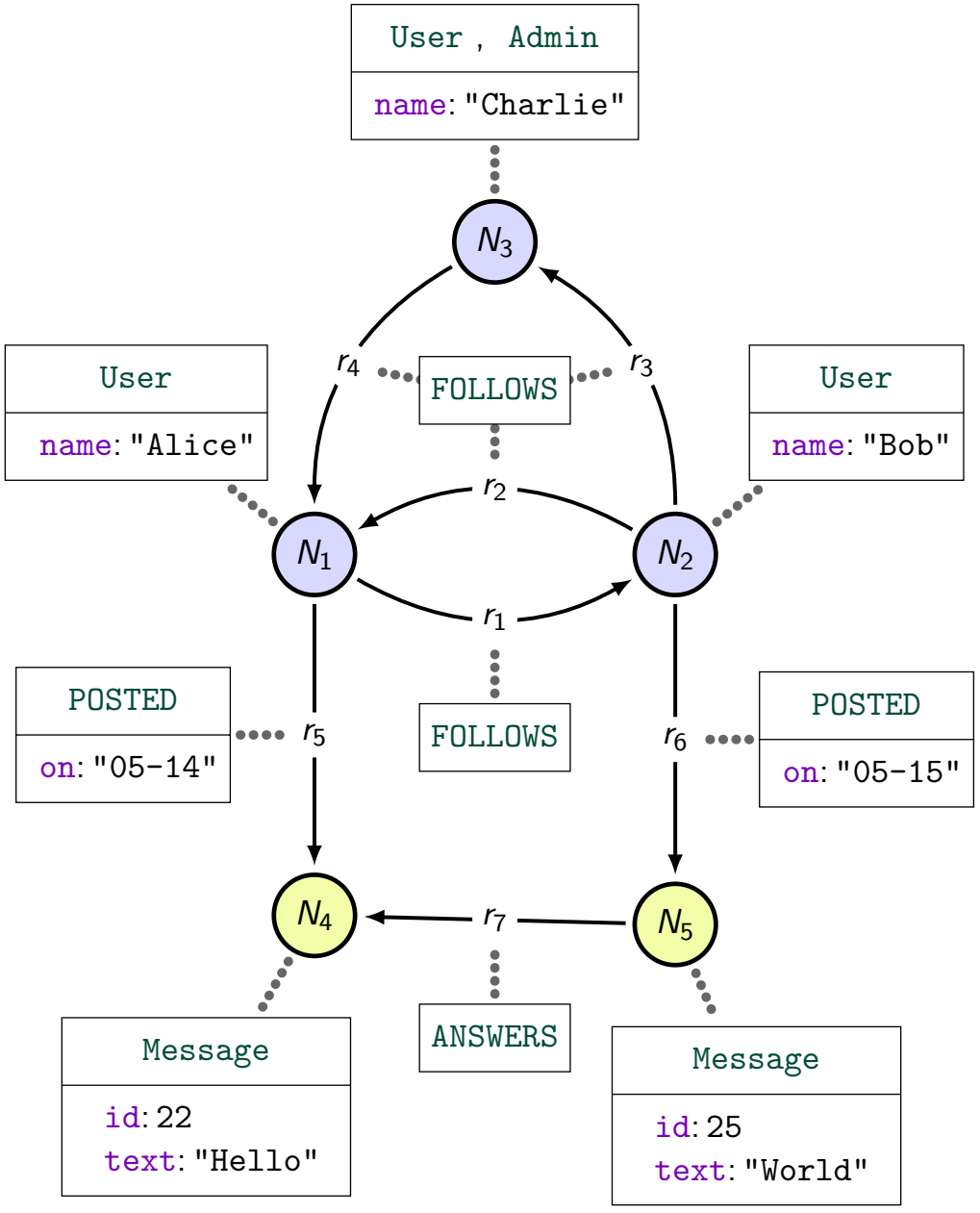
Road	
#src	#tgt
0	1
1	2
2	3
2	4
4	1



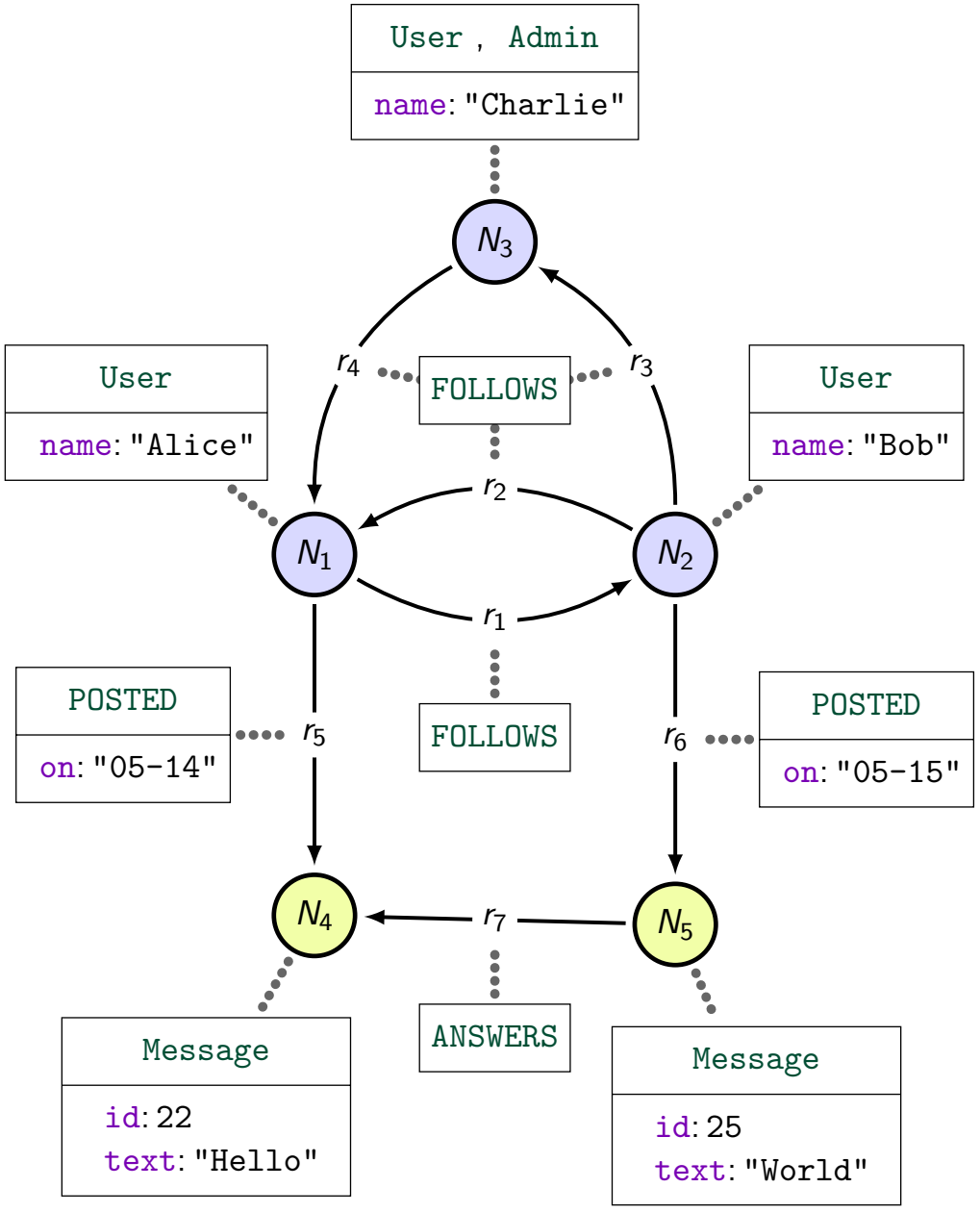
Ferry	
#src	#tgt
0	3

Gas	
#src	#tgt
4	4

Exercice: Storing graph 1 in tables

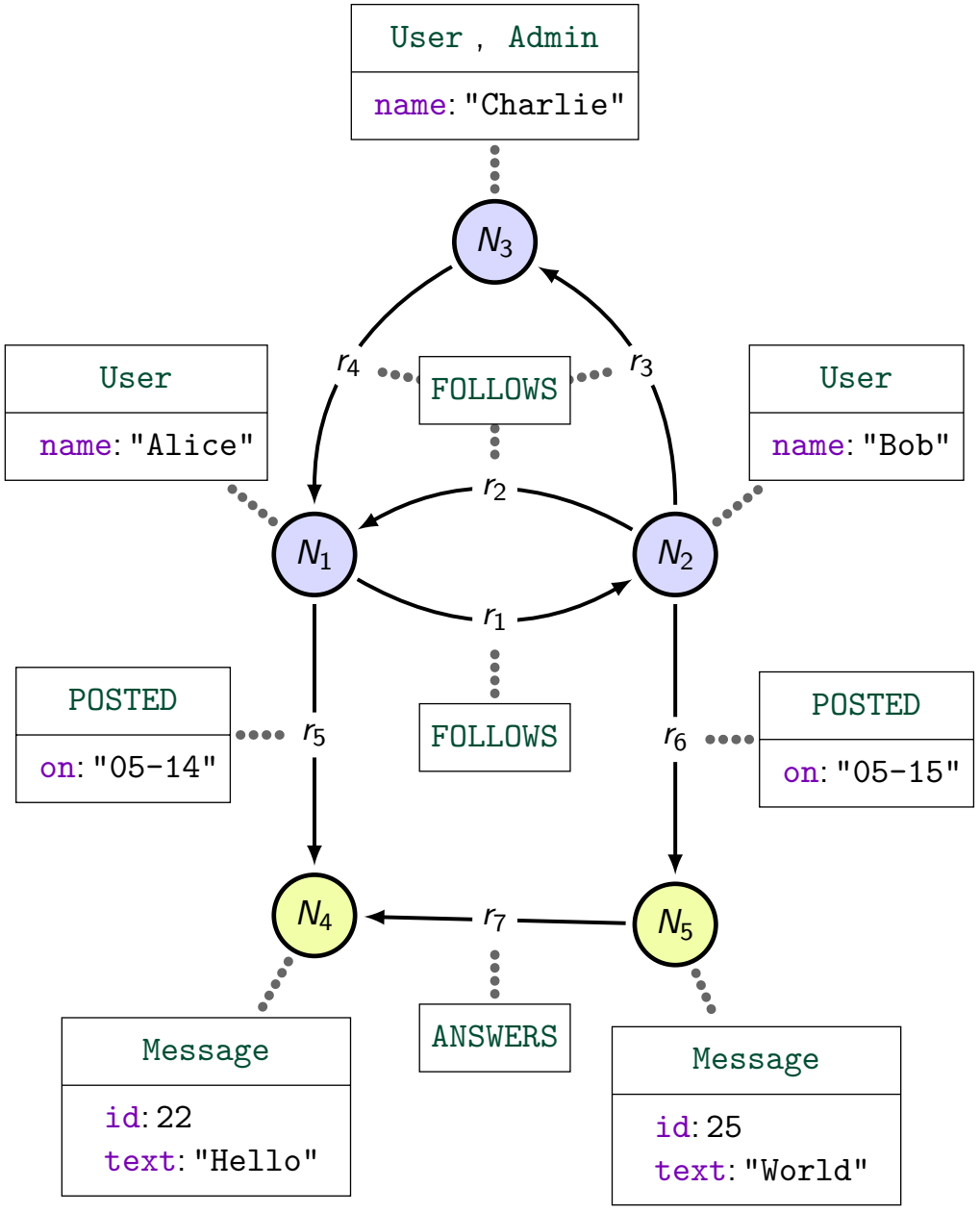


Exercice: Storing graph 1 in tables



Node	**Relation**		
id	id	#src	#tgt
1	1	1	2
2	2	2	1
⋮	⋮	⋮	⋮

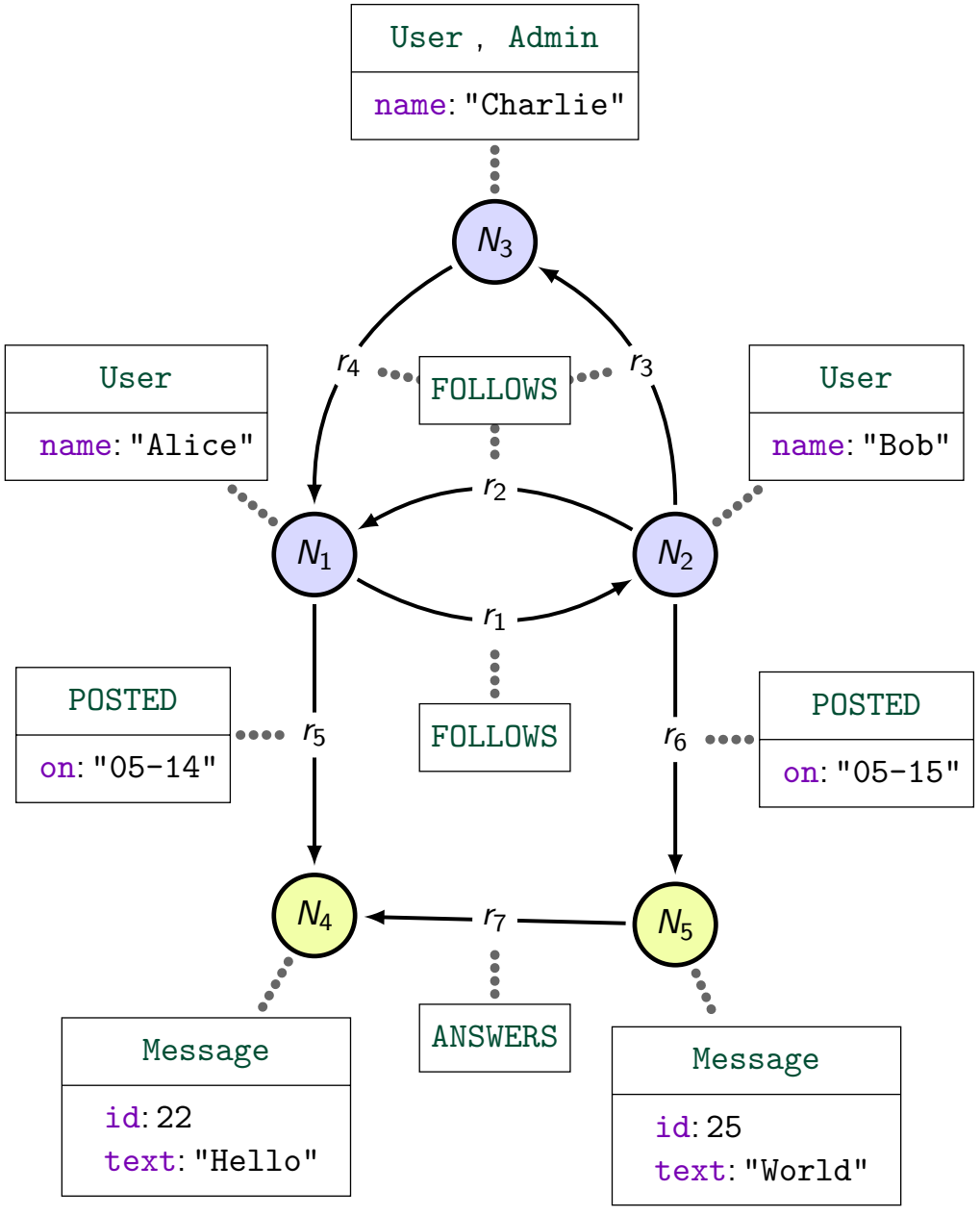
Exercice: Storing graph 1 in tables



Node	**Relation**		
id	id	#src	#tgt
1	1	1	2
2	2	2	1
⋮	⋮	⋮	⋮

Posted	**Message**
#eid	#vid
5	4
6	5

Why so many tables?



Node	Relation		
id	id	#src	#tgt
1	1	1	2
2	2	2	1
⋮	⋮	⋮	⋮

Posted	Message
#eid	#vid
5	4
6	5

On		Id	
#eid	val	#vid	val
5	"05-14"	4	22
6	"05-15"	5	25

Part II: Property Graphs

3. Translations: Tables → Graph

Preliminary poll: PG vs Relational ?

1. The property graphs data model is more expressive.
2. The relational data model (that is, tables) is stronger.
3. They are both as expressive.

 Counterintuitively, tables are way more general 

Consequences:

- Translating PGs→Tables is **easy**
- Translating Tables→PGs is **not always directly possible**

Translating some tables into a Property graph (1)

A relational database that we want to encode in a graph

Client

login	address
"Alice"	"Wonderland"
"Bob"	"124 Conch St."
"Eve"	null

Product

name	price
"Watch"	42
"Rabbit"	0
"Pants"	8
"Broom&Bucket"	4

___ : part of primary key

Translating some tables into a Property graph (1)

A relational database that we want to encode in a graph

Client

login	address
"Alice"	"Wonderland"
"Bob"	"124 Conch St."
"Eve"	null

Order

id	#buyer	date
0	"Alice"	01-11-1865
1	"Bob"	07-07-2022
2	"Bob"	07-11-2023

→Client.login

Product

name	price
"Watch"	42
"Rabbit"	0
"Pants"	8
"Broom&Bucket"	4

___ : part of primary key
: foreign keys

Translating some tables into a Property graph (1)

A relational database that we want to encode in a graph

Client

login	address
"Alice"	"Wonderland"
"Bob"	"124 Conch St."
"Eve"	null

Order

id	#buyer	date
0	"Alice"	01-11-1865
1	"Bob"	07-07-2022
2	"Bob"	07-11-2023

→Client.login

Product

name	price
"Watch"	42
"Rabbit"	0
"Pants"	8
"Broom&Bucket"	4

Contains

#order	#product	quant
0	"Rabbit"	1
0	"Watch"	1
1	"Pants"	7
2	"Pants"	14

→Order.id →Product.name

___ : part of primary key
: foreign keys

Translating some tables into a Property graph (1)

A relational database that we want to encode in a graph

Client

login	address
"Alice"	"Wonderland"
"Bob"	"124 Conch St."
"Eve"	null

Order

id	#buyer	date
0	"Alice"	01-11-1865
1	"Bob"	07-07-2022
2	"Bob"	07-11-2023

→Client.login

Product

name	price
"Watch"	42
"Rabbit"	0
"Pants"	8
"Broom&Bucket"	4

Contains

#order	#product	quant
0	"Rabbit"	1
0	"Watch"	1
1	"Pants"	7
2	"Pants"	14

→Order.id →Product.name

___ : part of primary key
\# : foreign keys

Exercice: Translate these to a graph!

Translating some tables into a Property graph (2)

Who is a node and who is an edge?

Client	
login	**address**

Order		
id	**#buyer**	**date**
	→Client.login	

Product	
name	**price**

Contains		
#order	**#product**	**quant**
→Order.id	→Product.name	

___ : part of primary key
\# : foreign keys

Translating some tables into a Property graph (2)

Who is a node and who is an edge? → Reverse engineer ER Diagram

Client	
login	**address**

Order		
id	**#buyer**	**date**
	→Client.login	

Product	
name	**price**

Contains		
#order	**#product**	**quant**
→Order.id	→Product.name	

___ : part of primary key
\# : foreign keys

Translating some tables into a Property graph (2)

Who is a node and who is an edge? → Reverse engineer ER Diagram

Client

login	address

Order

id	# buyer	date

→Client.login

Product

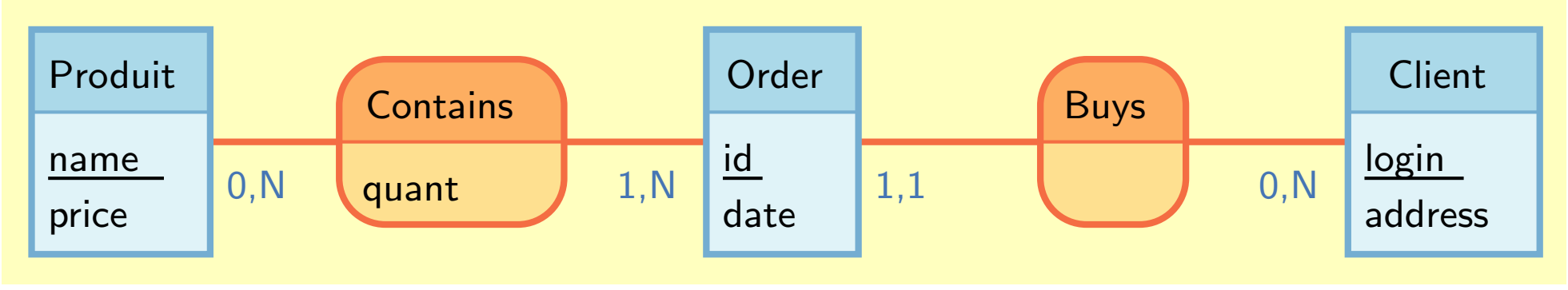
name	price

Contains

# order	# product	quant

→Order.id →Product.name

___ : part of primary key
\# : foreign keys



Conditions for the translation to be straightforward

Relational DB consists of tables $T_1, \ldots, T_k$.

Each table T_i

- has a primary key, consisting of several columns
- has columns that are foreign keys

Foreign keys can be part of the primary key.

Conditions for the translation to be straightforward

Relational DB consists of tables $T_1, \ldots, T_k$.

Each table T_i
- has a primary key, consisting of several columns
- has columns that are foreign keys


Foreign keys can be part of the primary key.

Conditions for the database to be encodable in a graph

Each table T_i satisfies one of the following.

0. Zero foreign key is part of the primary key of T_i. (~Vertex)
1. One foreign key is part of the primary key of T_i.
2. Two foreign keys are part of the primary key of T_i. (~Edges)

Translating some tables into a Property graph (4)

A relational database that we want to encode in a graph

Client

login	address
"Alice"	"Wonderland"
"Bob"	"124 Conch St."
"Eve"	null

Order

id	#buyer	date
0	"Alice"	01-11-1865
1	"Bob"	07-07-2022
2	"Bob"	07-11-2023

→Client.login

Product

name	price
"Watch"	42
"Rabbit"	0
"Pants"	8
"Broom&Bucket"	4

Contains

#order	#product	quant
0	"Rabbit"	1
0	"Watch"	1
1	"Pants"	7
2	"Pants"	14

→Order.id →Product.name

___ : part of primary key
\# : foreign keys

- **Client**, **Product** and **Order** satisfy 0
- **Contains** satisfies 2

Translating some tables into a Property graph (5)

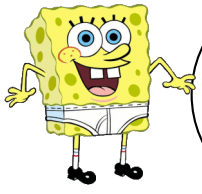
One vertex per row in table satisfying **0** or **1**

Client row 1

Order row 1

Product row 1

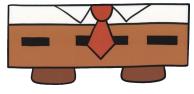
Product row 2



Client row 2

Order row 2

Product row 3



Client row 3

Order row 3

Product row 4

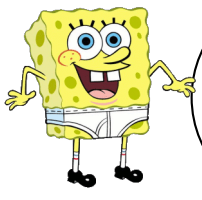
Translating some tables into a Property graph (5)

One edge per row and per foreign-key column in each table satisfying **0** or **1**

Client row 1

Buyer

Order row 1



Client row 2

Buyer

Order row 2

Buyer

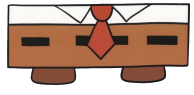
Client row 3

Order row 3

Product row 1

Product row 2

Product row 3

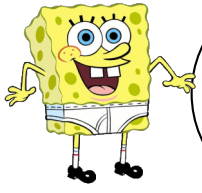
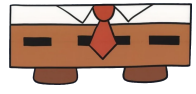


Product row 4

Translating some tables into a Property graph (5)

One edge per row of tables satisfying 2

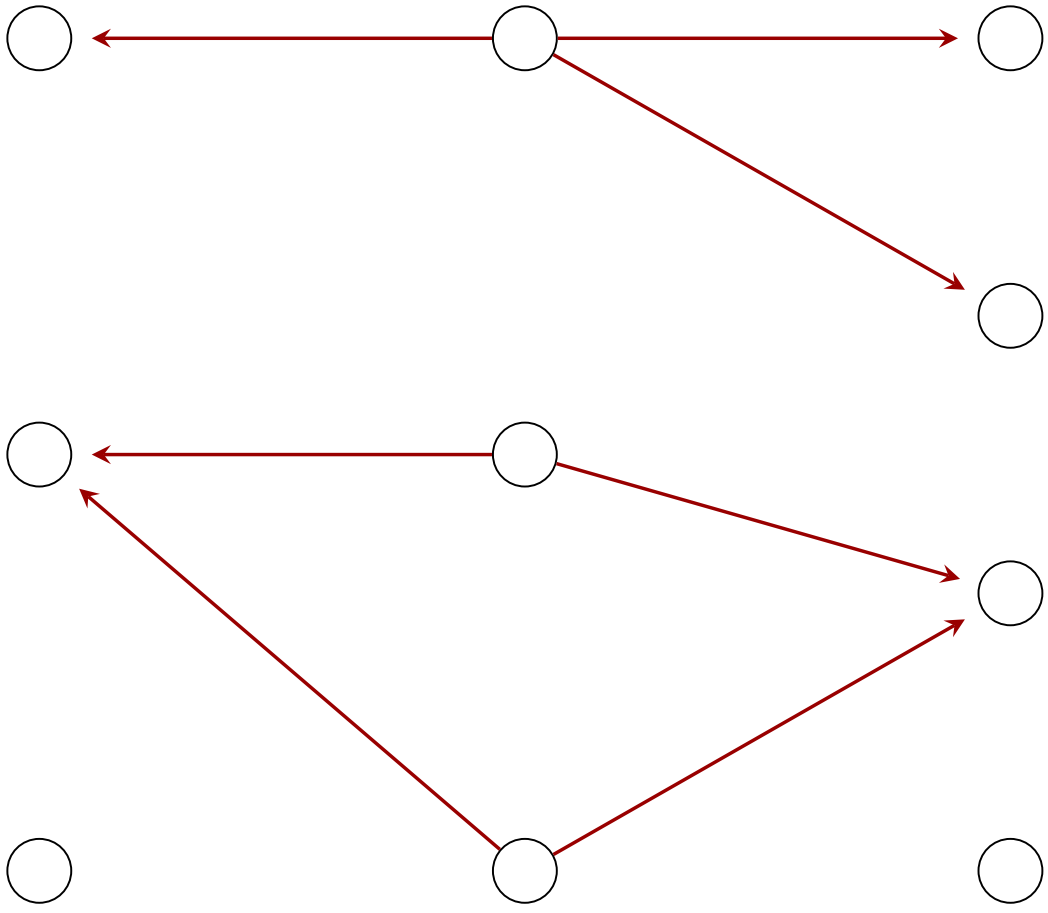

Client row 1
Buyer
Order row 1
Contains
Product row 1
Contains
Product row 2


Client row 2
Buyer
Order row 2
Contains
Product row 3

Buyer
Contains

Client row 3
Order row 3
Product row 4

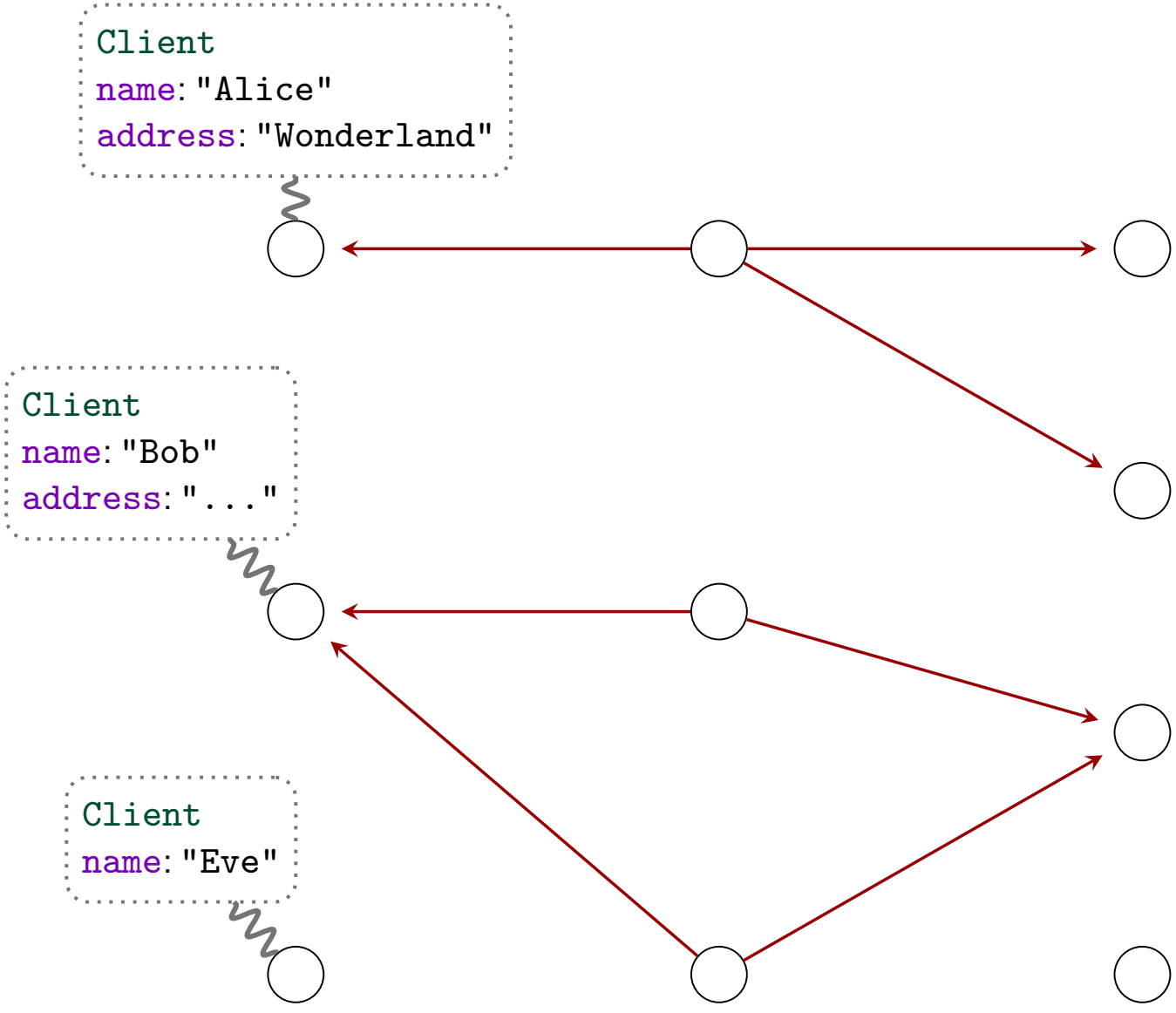

Translating some tables into a Property graph (5)

Then, we add properties



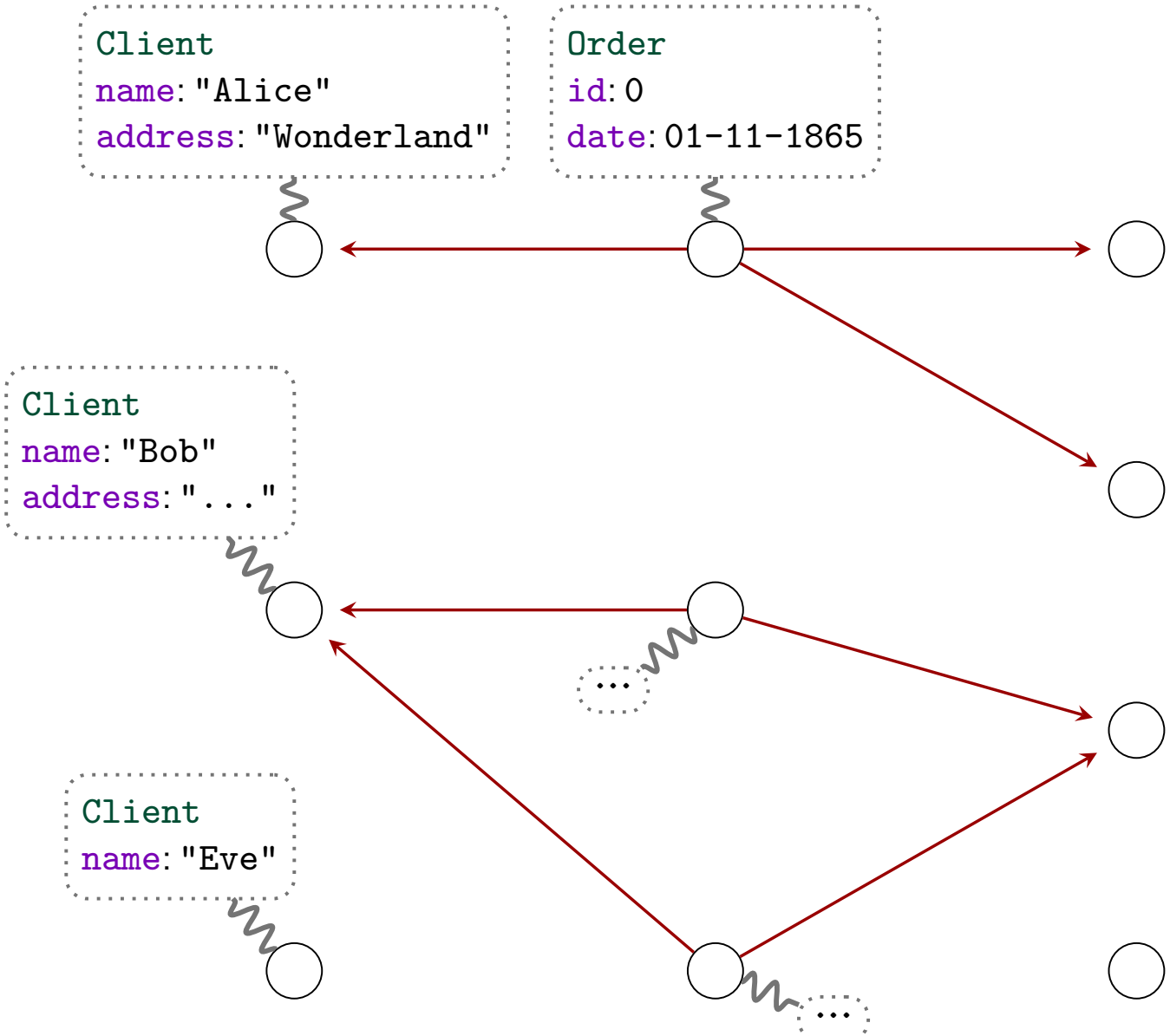
Translating some tables into a Property graph (5)

Then, we add properties



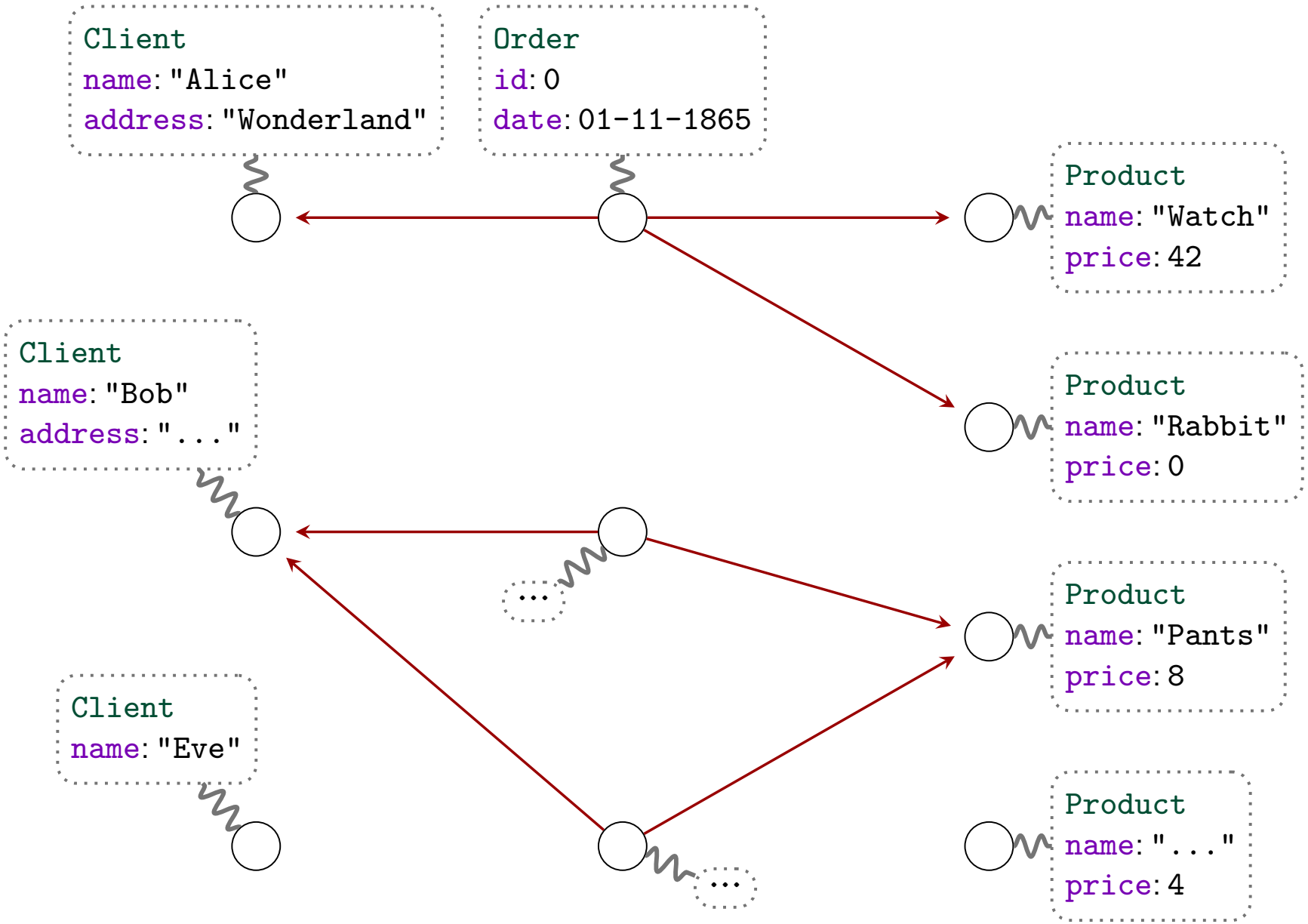
Translating some tables into a Property graph (5)

Then, we add properties



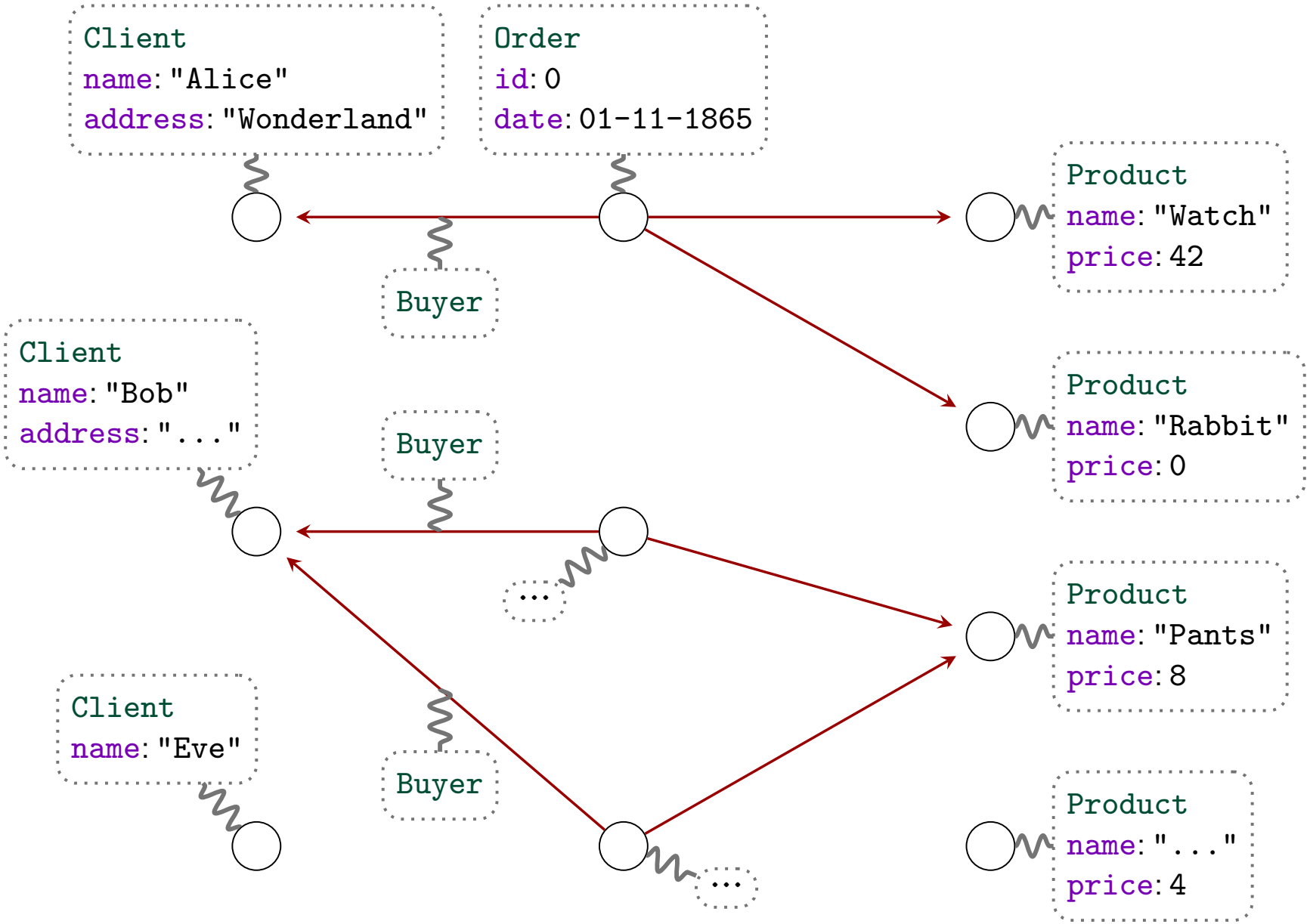
Translating some tables into a Property graph (5)

Then, we add properties



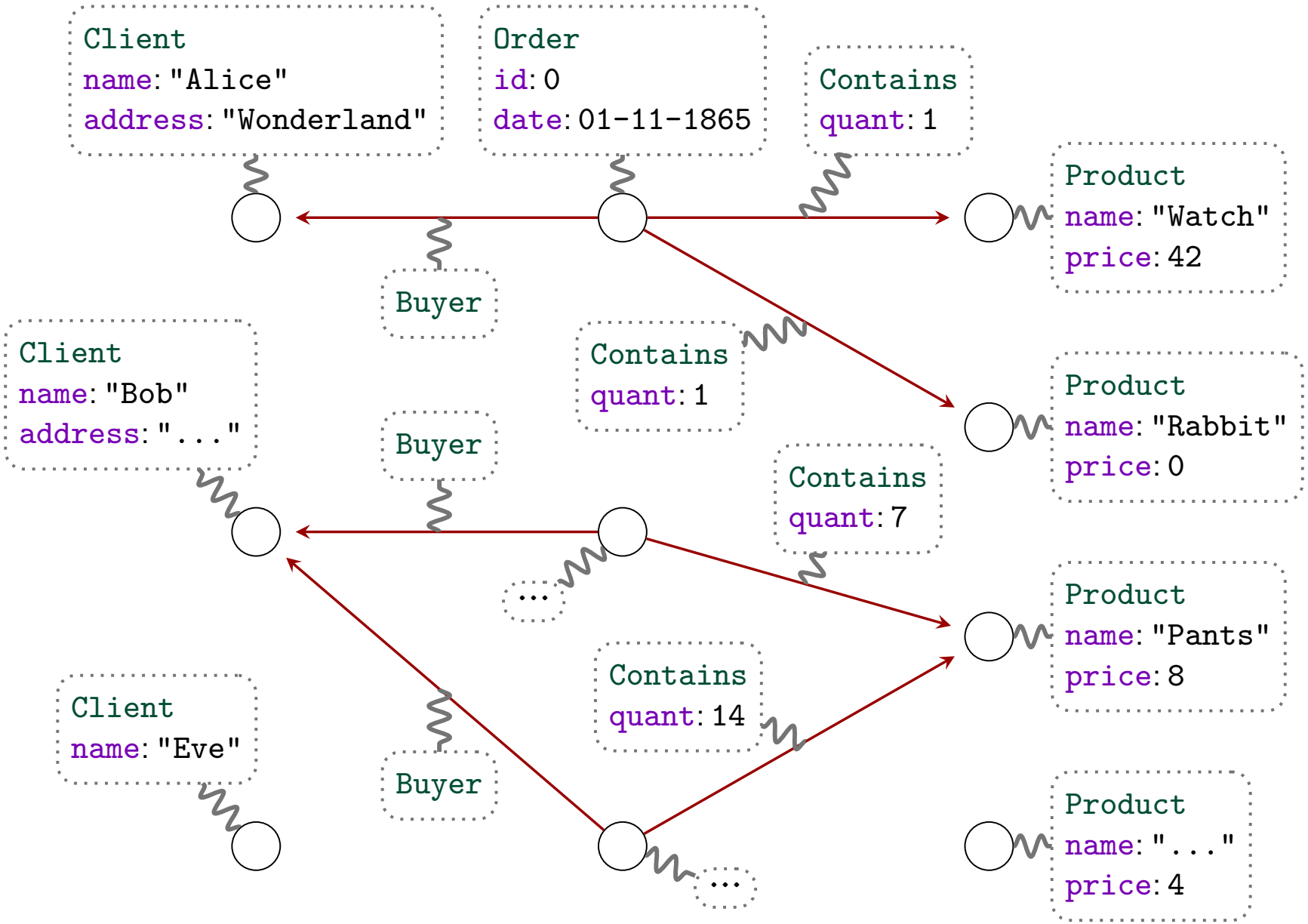
Translating some tables into a Property graph (5)

Then, we add properties



Translating some tables into a Property graph (5)

Then, we add properties



Translating some tables into a Property graph (5)

Which representation do you prefer?

Client

login	address
"Alice"	"Wonderland"
"Bob"	"124 Conch St."
"Eve"	null

Order

id	#buyer	date
0	"Alice"	01-11-1865
1	"Bob"	07-07-2022
2	"Bob"	07-11-2023

→Client.login

Product

name	price
"Watch"	42
"Rabbit"	0
"Pants"	8
"Broom&Bucket"	4

Contains

#order	#product	quant
0	"Rabbit"	1
0	"Watch"	1
1	"Pants"	7
2	"Pants"	14

→Order.id →Product.name

___ : part of primary key
\# : foreign keys

- **Client**, **Product** and **Order** satisfy 0
- **Contains** satisfies 2

Translating some tables into a Property graph (6)

Conditions for the database to be encodable in a graph

Each table T_i satisfies one of the following.

0. Zero foreign key is part of the primary key of T_i. (~Vertex)
1. One foreign key is part of the primary key of T_i.
2. Two foreign keys are part of the primary key of T_i. (~Edges)

Translating some tables into a Property graph (6)

Conditions for the database to be encodable in a graph

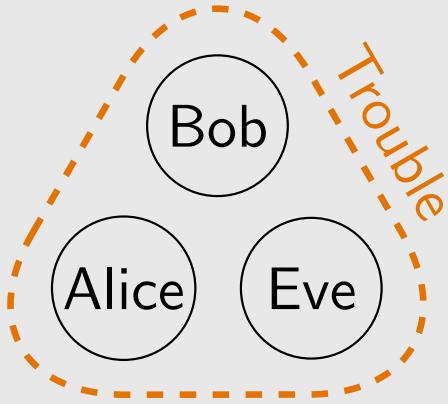
Each table T_i satisfies one of the following.

0. Zero foreign key is part of the primary key of T_i. (~Vertex)
1. One foreign key is part of the primary key of T_i.
2. Two foreign keys are part of the primary key of T_i. (~Edges)

3. Three foreign keys are part of the primary key of $T_i \Longrightarrow$ Trouble

Trouble

#person1	#person2	#person3
Alice	Bob	Eve
⋮	⋮	⋮



Encoding non-binary relations in graphs (1)

Question: how would you do it?

Trouble

#pers1	#pers2	#pers3
Alice	Bob	Eve
Alice	Carl	Dave

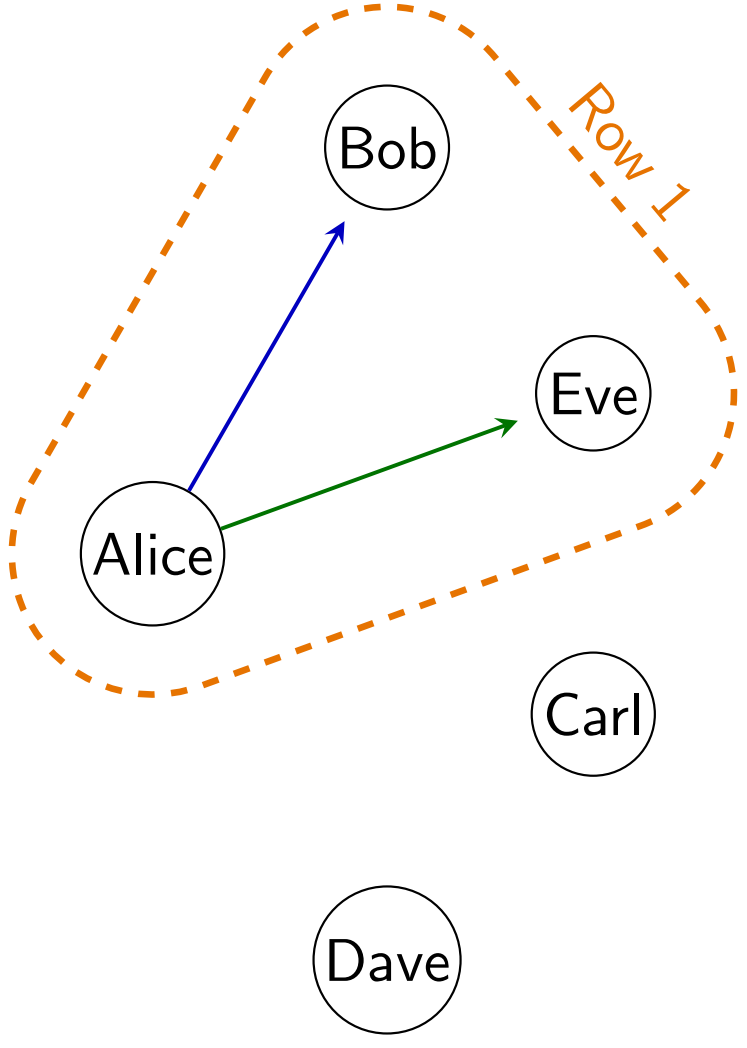
Encoding non-binary relations in graphs (2)

The **wrong** way: adding more edges

Trouble

#pers1	#pers2	#pers3
Alice	Bob	Eve
Alice	Carl	Dave

Let us try to add two edges per row of table **Trouble**.



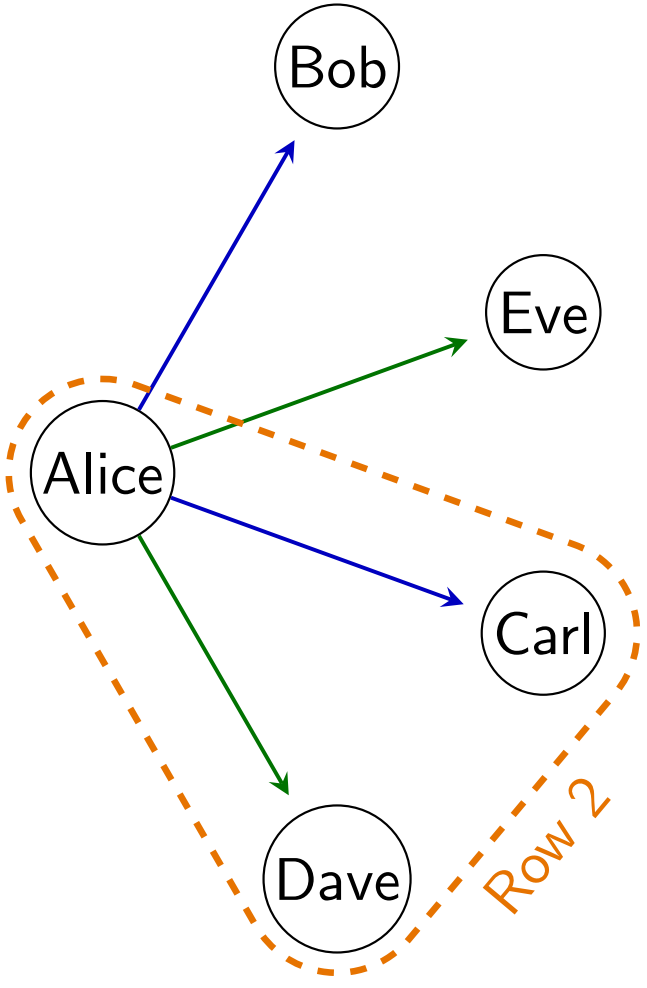
Encoding non-binary relations in graphs (2)

The **wrong** way: adding more edges

Trouble

#pers1	#pers2	#pers3
Alice	Bob	Eve
Alice	Carl	Dave

Let us try to add two edges per row of table **Trouble**.



Encoding non-binary relations in graphs (2)

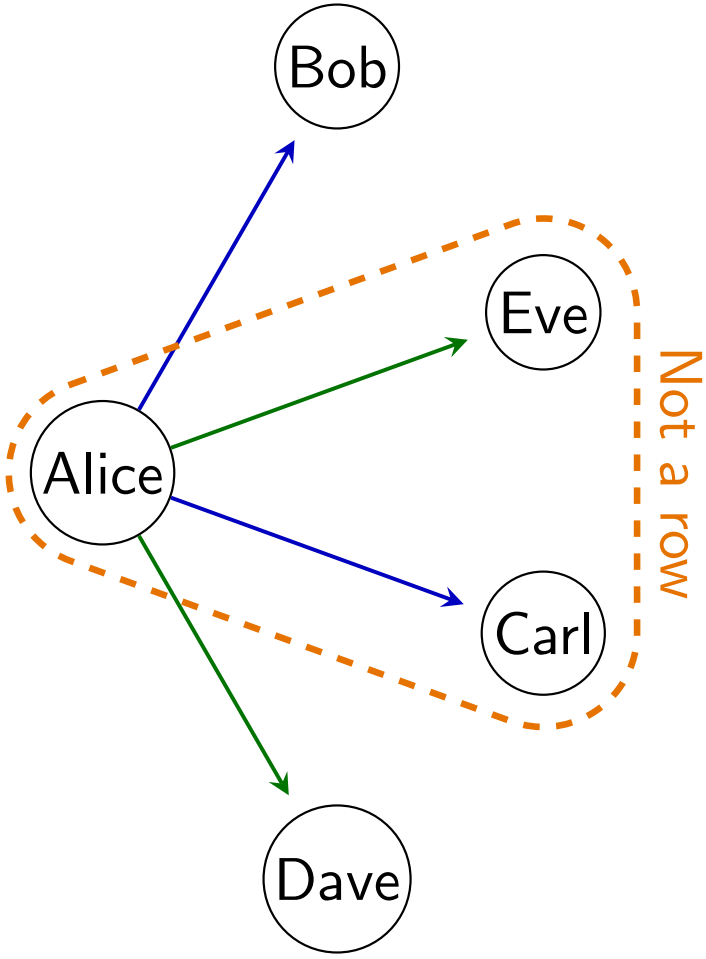
The **wrong** way: adding more edges

Trouble

#pers1	#pers2	#pers3
Alice	Bob	Eve
Alice	Carl	Dave

Let us try to add two edges per row of table **Trouble**.

⚠ (Alice, Carl, Eve) is not a row of table **Trouble**



Encoding non-binary relations in graphs (3)

The **right** way : Reification

Reification

- Literally, make into an object
- For us, transform into a node

Encoding non-binary relations in graphs (3)

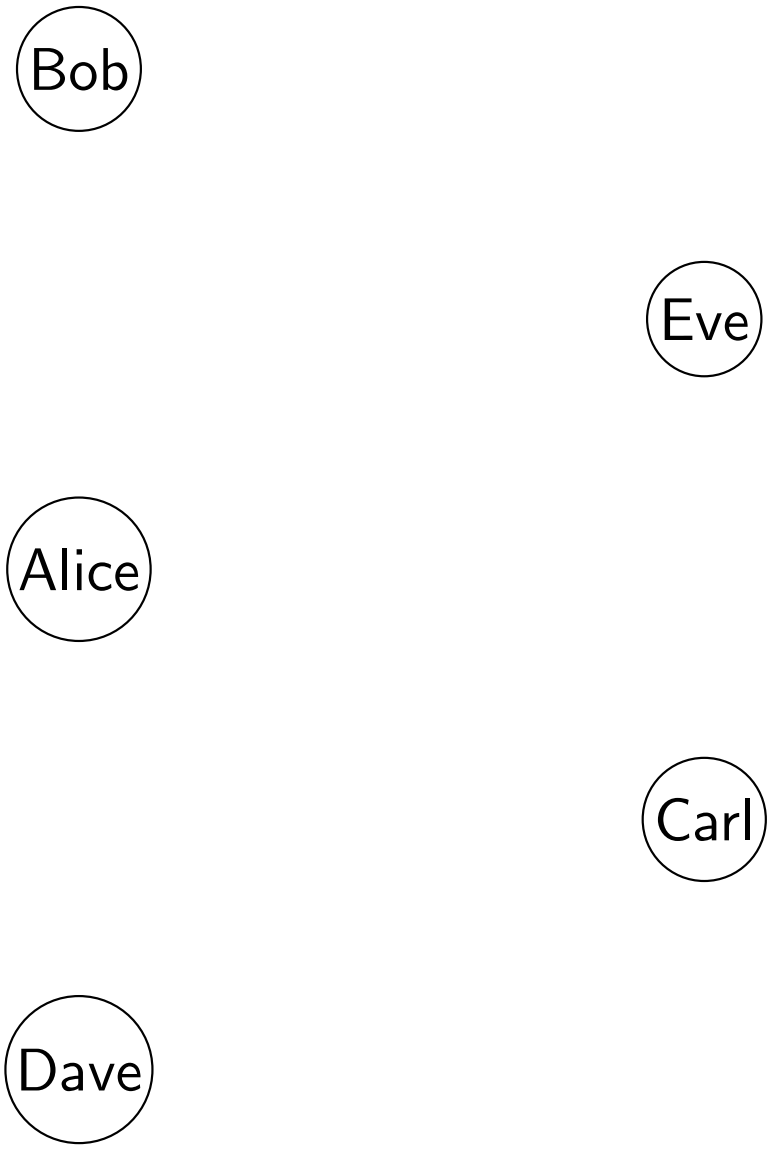
The **right** way : Reification

Reification

- Literally, make into an object
- For us, transform into a node

Trouble

#pers1	#pers2	#pers3
Alice	Bob	Eve
Alice	Carl	Dave



Encoding non-binary relations in graphs (3)

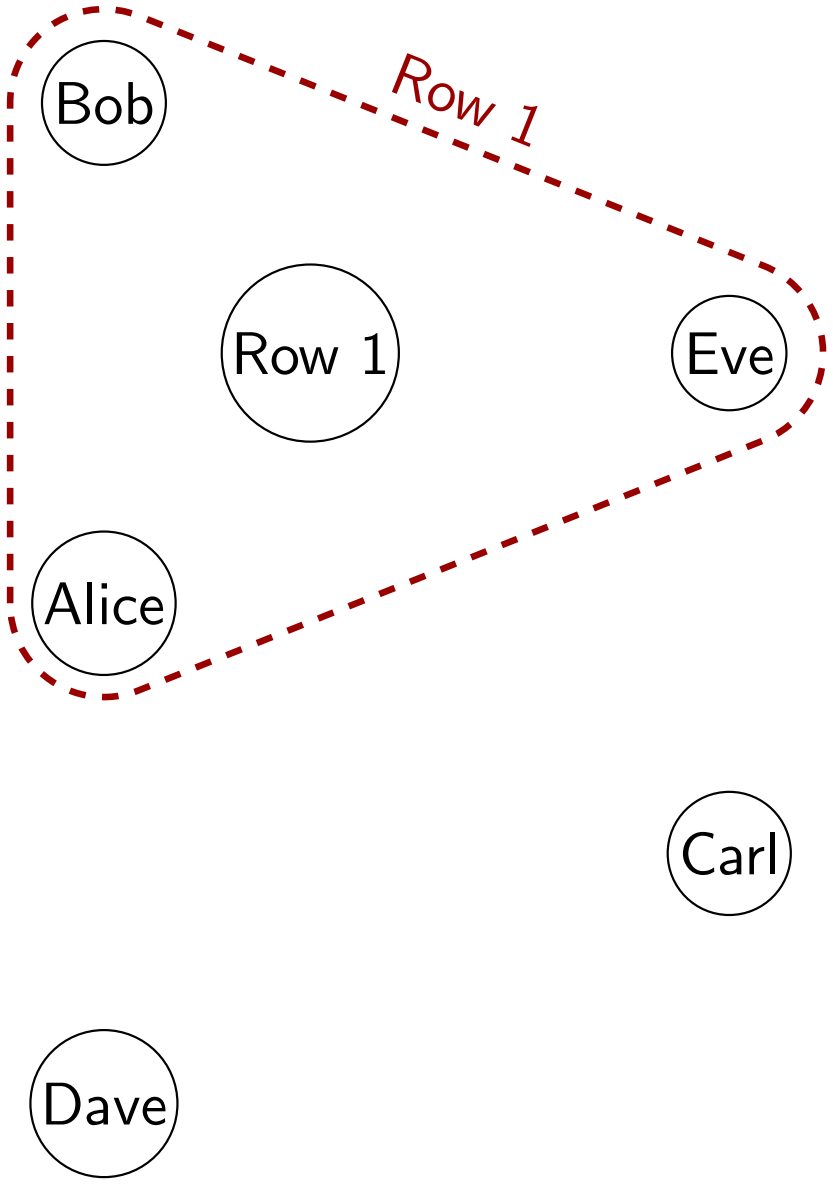
The **right** way : Reification

Reification

- Literally, make into an object
- For us, transform into a node

Trouble

#pers1	#pers2	#pers3
Alice	Bob	Eve
Alice	Carl	Dave

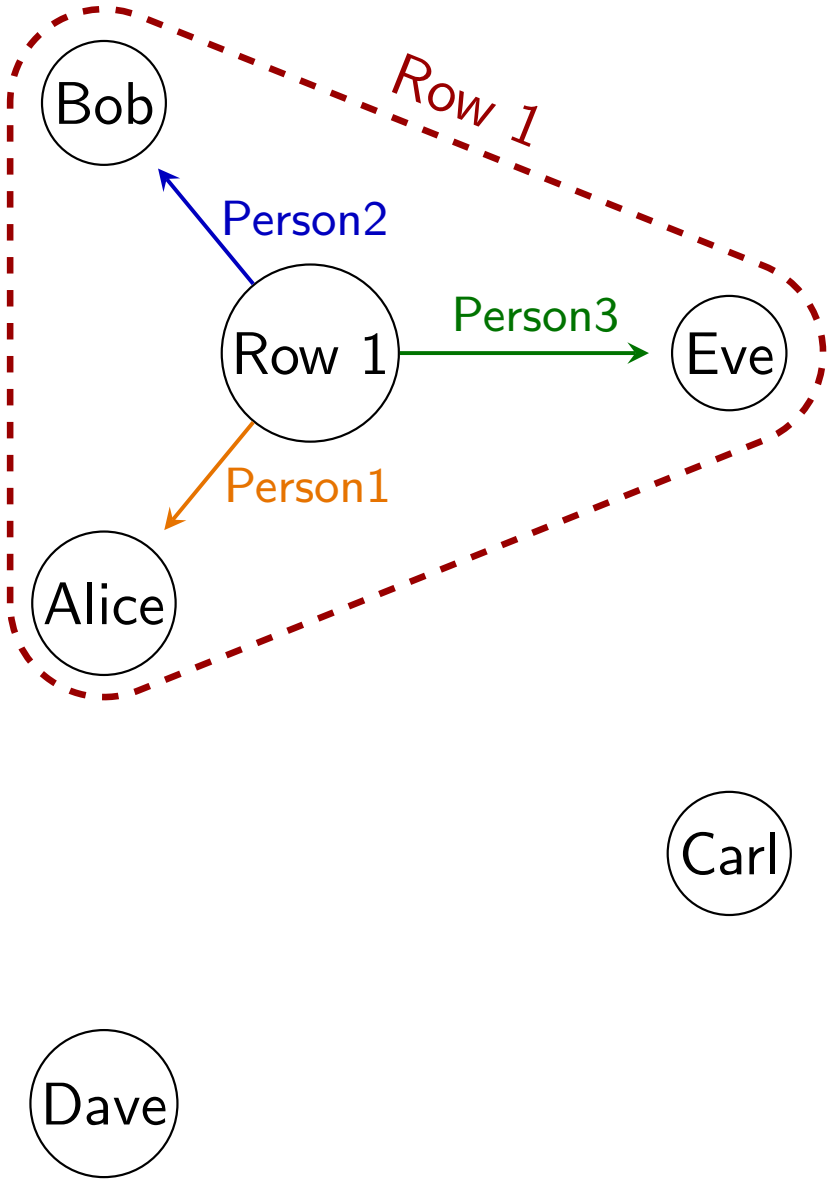


Encoding non-binary relations in graphs (3)

The **right** way : Reification

Reification

- Literally, make into an object
- For us, transform into a node



Trouble

#pers1	#pers2	#pers3
Alice	Bob	Eve
Alice	Carl	Dave

Encoding non-binary relations in graphs (3)

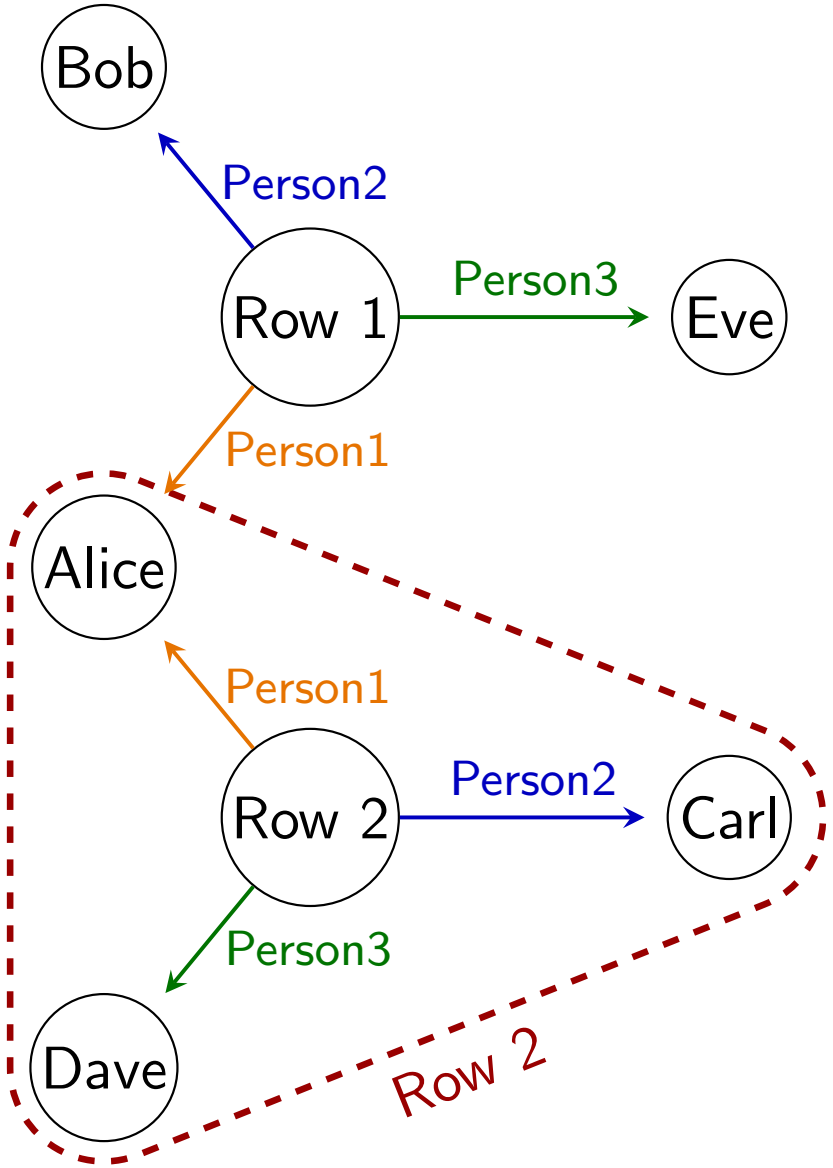
The **right** way : Reification

Reification

- Literally, make into an object
- For us, transform into a node

Trouble

#pers1	#pers2	#pers3
Alice	Bob	Eve
Alice	Carl	Dave



Encoding non-binary relations in graphs (3)

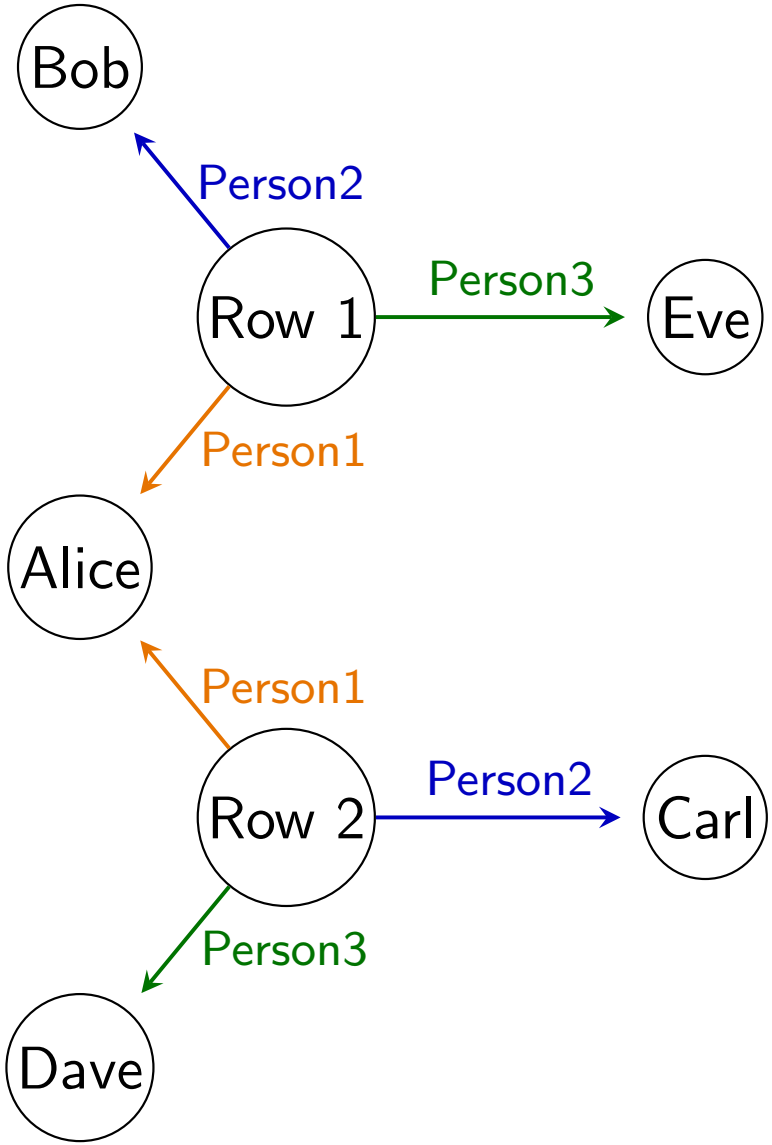
The **right** way : Reification

Reification

- Literally, make into an object
- For us, transform into a node

Trouble

#pers1	#pers2	#pers3
Alice	Bob	Eve
Alice	Carl	Dave



Encoding non-binary relations in graphs (4)

Reification is no miracle solution

Reification works...

- Reversible (one may reconstruct the **Trouble** table)
- Easy to generalize to any arity

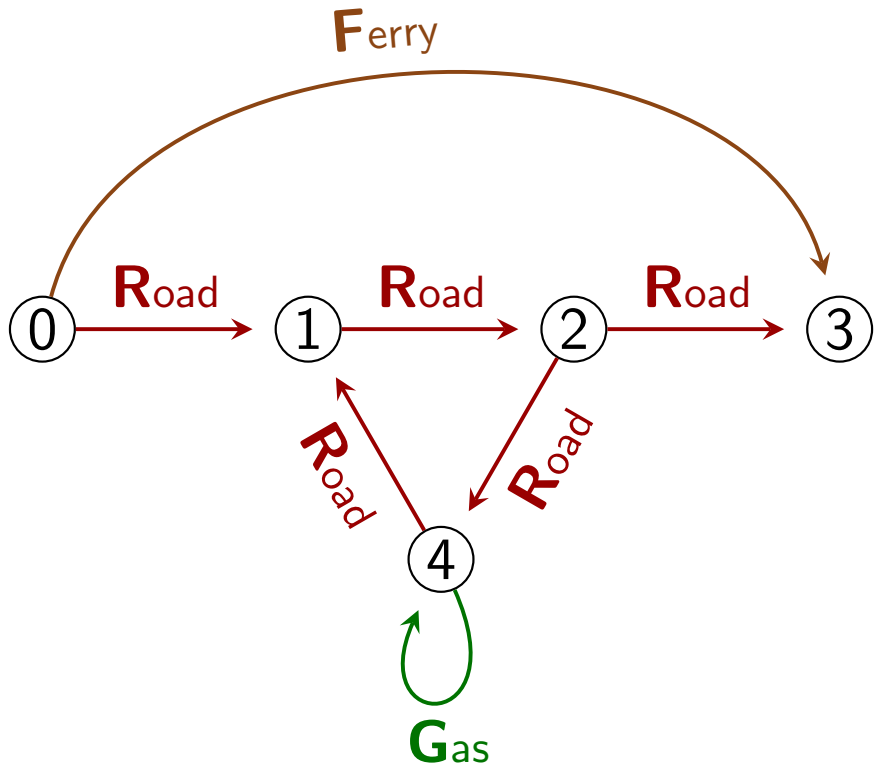
...but, it is contrary to the spirit of graphs:

- The graph requires extra knowledge and maintenance:
  - Special vertices/edges/labels
  - Implicitly linked labels/edges (Person1/Person2/Person3)
  - Integrity constraints
- Query languages for graphs are based on walks, reification is fundamentally branching

Part II: Property Graphs

4. Storage matters

Elementary operations on graphs

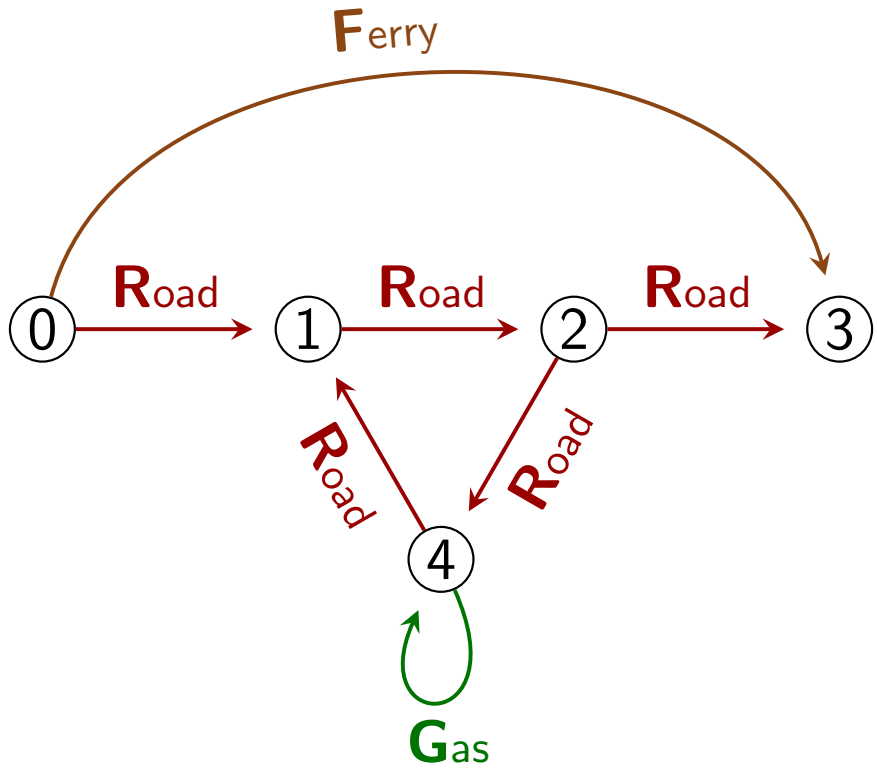


Edge test

Given s, ℓ, t, does $s \xrightarrow{\ell} t$ exist?

Ex: Is there an edge 0 $\xrightarrow{\text{Road}}$ 4 ?

Answer: no



Elementary operations on graphs

Edge test

Given s, ℓ, t, does $s \xrightarrow{\ell} t$ exist?

Ex: Is there an edge 0 $\xrightarrow{\text{Road}}$ 4 ?

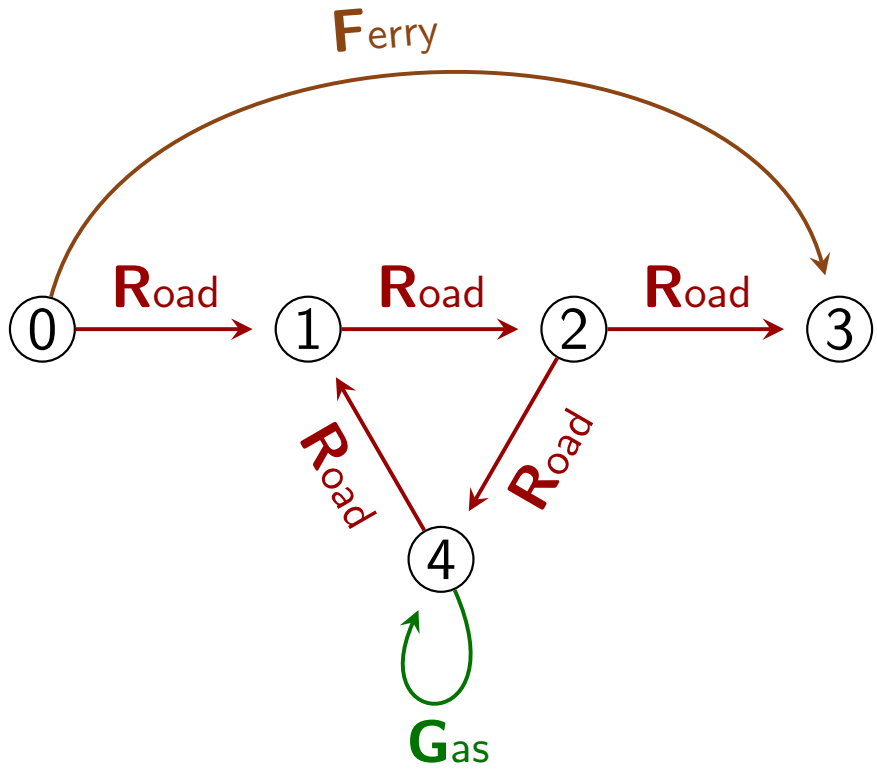
Answer: no

Successor

Given s, ℓ, compute all t such that $s \xrightarrow{\ell} t$ exists.

Ex: Which nodes are reachable from 2 by a Road edge.

Answer: 3 and 4.



Elementary operations on graphs

Edge test

Given s, ℓ, t, does $s \xrightarrow{\ell} t$ exist?

Ex: Is there an edge $0 \xrightarrow{\text{Road}} 4$?

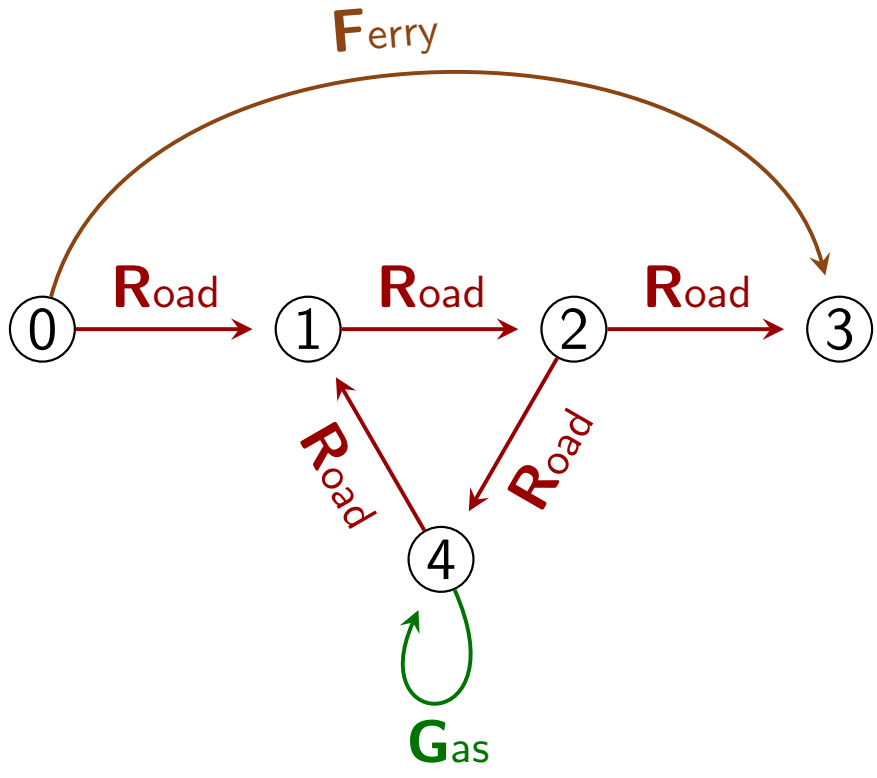
Answer: no

Successor

Given s, ℓ, compute all t such that $s \xrightarrow{\ell} t$ exists.

Ex: Which nodes are reachable from 2 by a Road edge.

Answer: 3 and 4.



Cost of these operations?

Elementary operations on graphs

Edge test

Given s, ℓ, t, does $s \xrightarrow{\ell} t$ exist?

Ex: Is there an edge $0 \xrightarrow{\text{Road}} 4$?

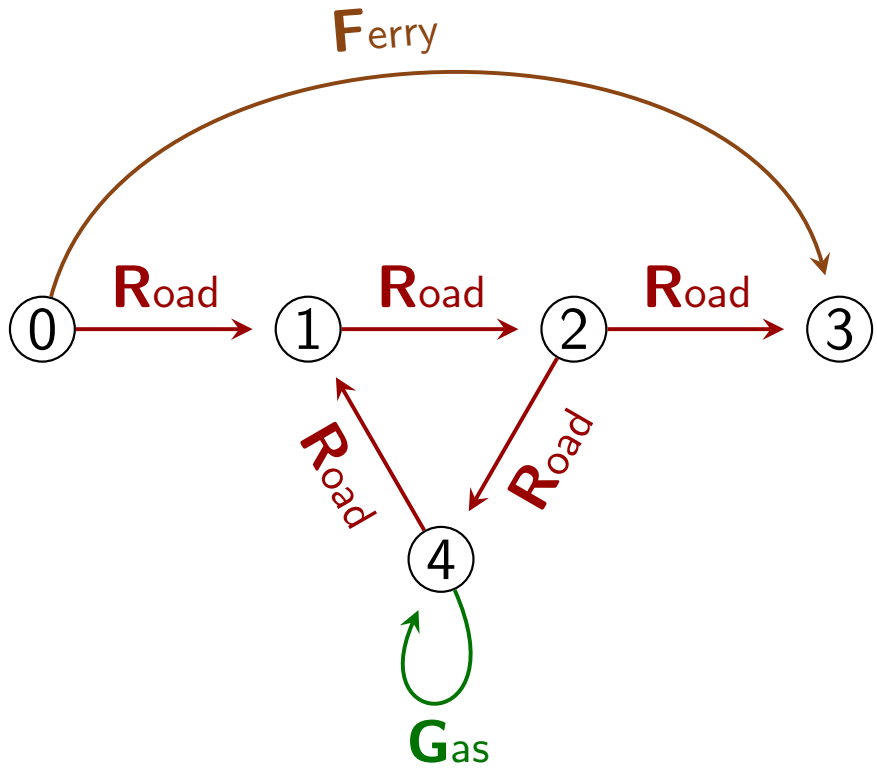
Answer: no

Successor

Given s, ℓ, compute all t such that $s \xrightarrow{\ell} t$ exists.

Ex: Which nodes are reachable from 2 by a Road edge.

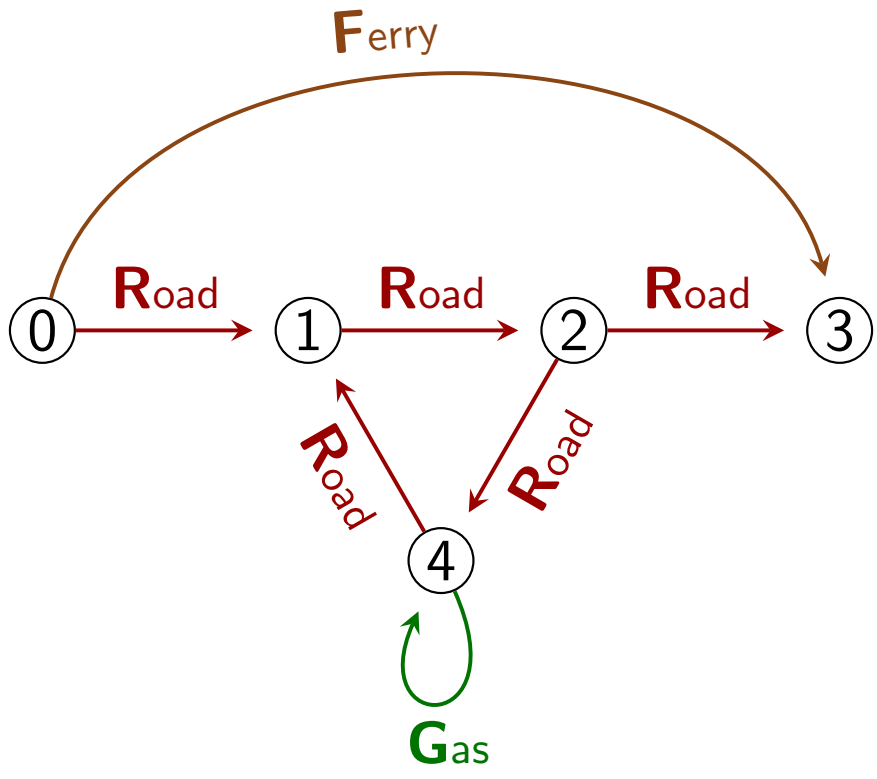
Answer: 3 and 4.



Cost of these operations?

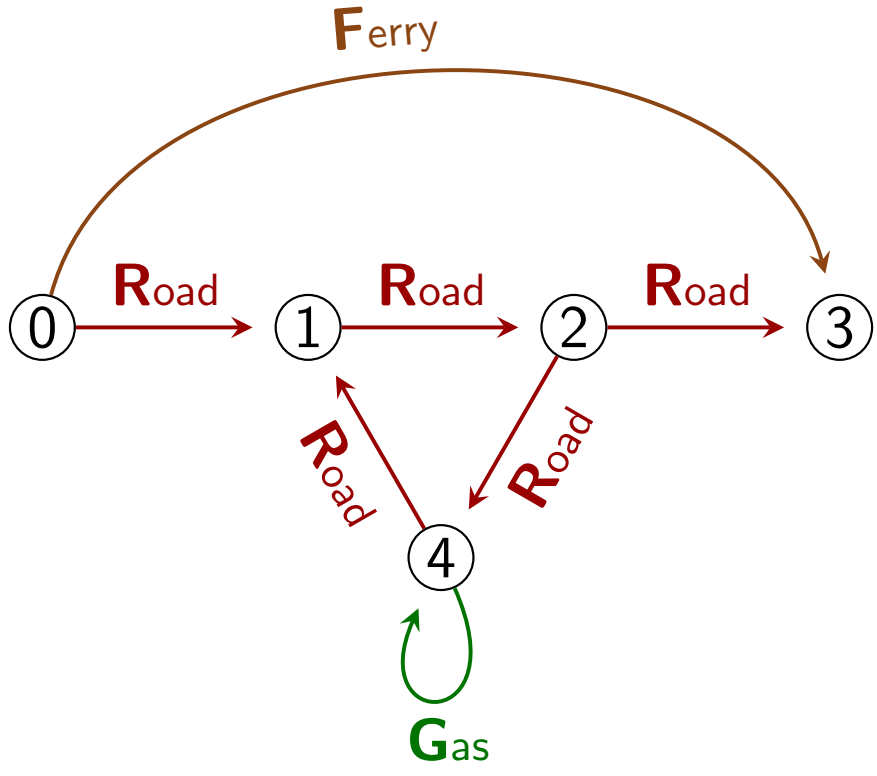
It depends on storage

Storing adjacency lists ("Native graph storage", ex: Neo4j)



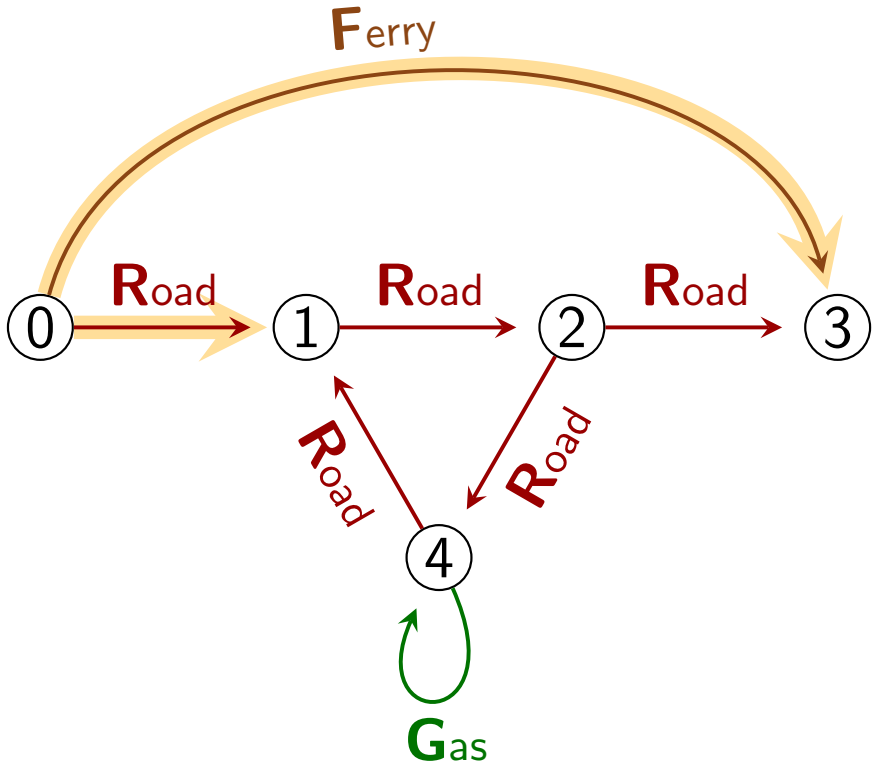
Storing adjacency lists ("Native graph storage", ex: Neo4j)

- A memory zone for each vertex and edge
- Each edge stores refs to source, label, target
- Each vertex stores refs to adjacent edges (usually indexed by label)



Storing adjacency lists ("Native graph storage", ex: Neo4j)

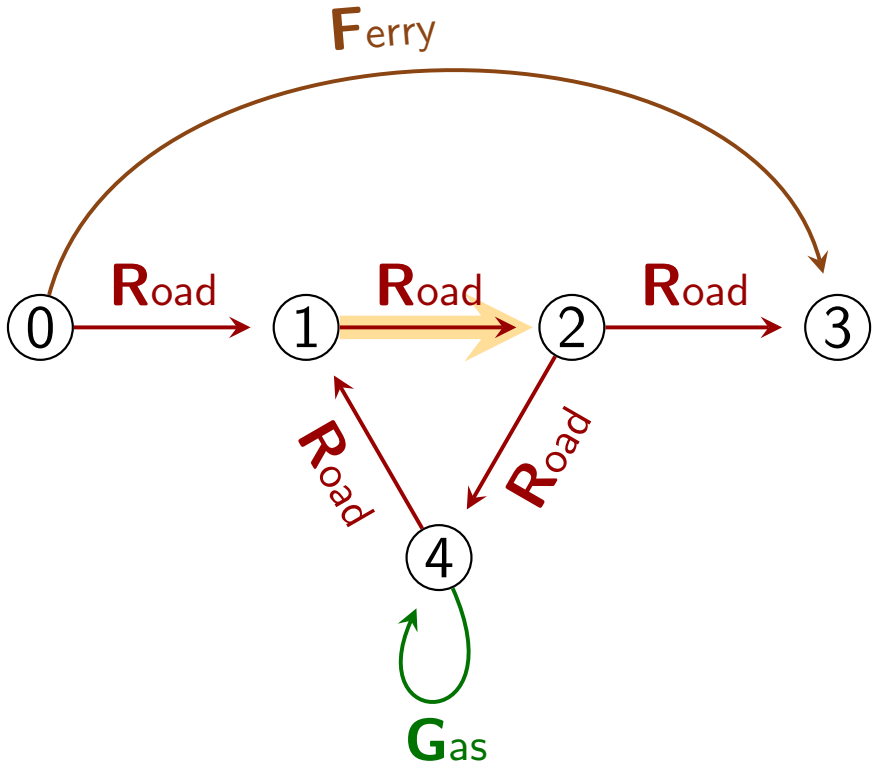
- A memory zone for each vertex and edge
- Each edge stores refs to source, label, target
- Each vertex stores refs to adjacent edges (usually indexed by label)



	Road	Ferry	Gas
0:	[1]	[3]	[]
1:			
2:			
3:			
4:			

Storing adjacency lists ("Native graph storage", ex: Neo4j)

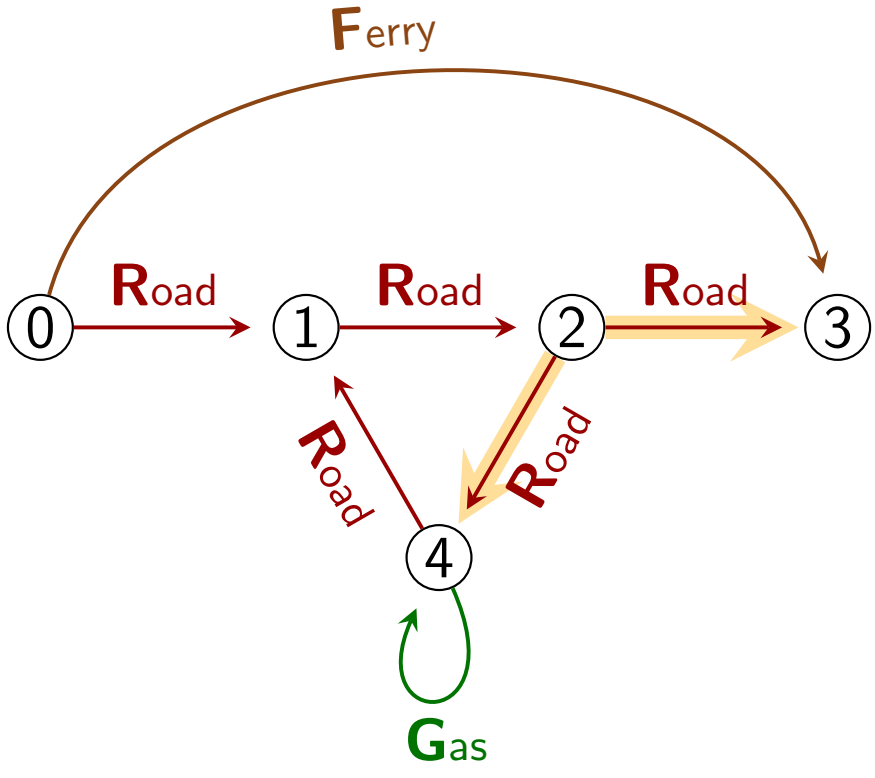
- A memory zone for each vertex and edge
- Each edge stores refs to source, label, target
- Each vertex stores refs to adjacent edges (usually indexed by label)



	Road	Ferry	Gas
0:	[1]	[3]	[]
1:	[2]	[]	[]
2:			
3:			
4:			

Storing adjacency lists ("Native graph storage", ex: Neo4j)

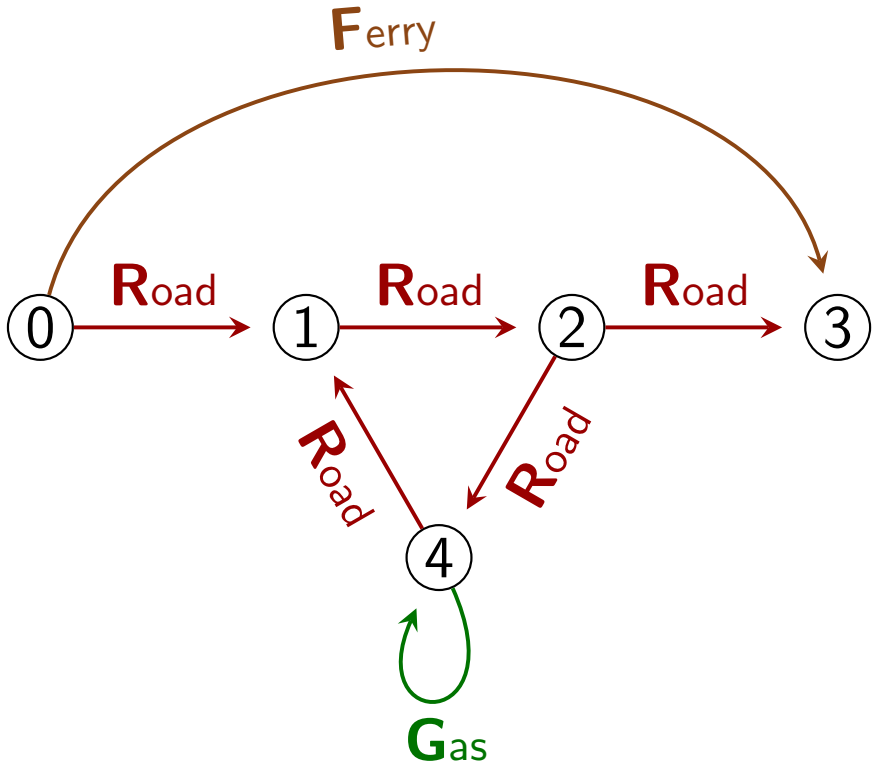
- A memory zone for each vertex and edge
- Each edge stores refs to source, label, target
- Each vertex stores refs to adjacent edges (usually indexed by label)



	Road	Ferry	Gas
0:	[1]	[3]	[]
1:	[2]	[]	[]
2:	[3,4]	[]	[]
3:			
4:			

Storing adjacency lists ("Native graph storage", ex: Neo4j)

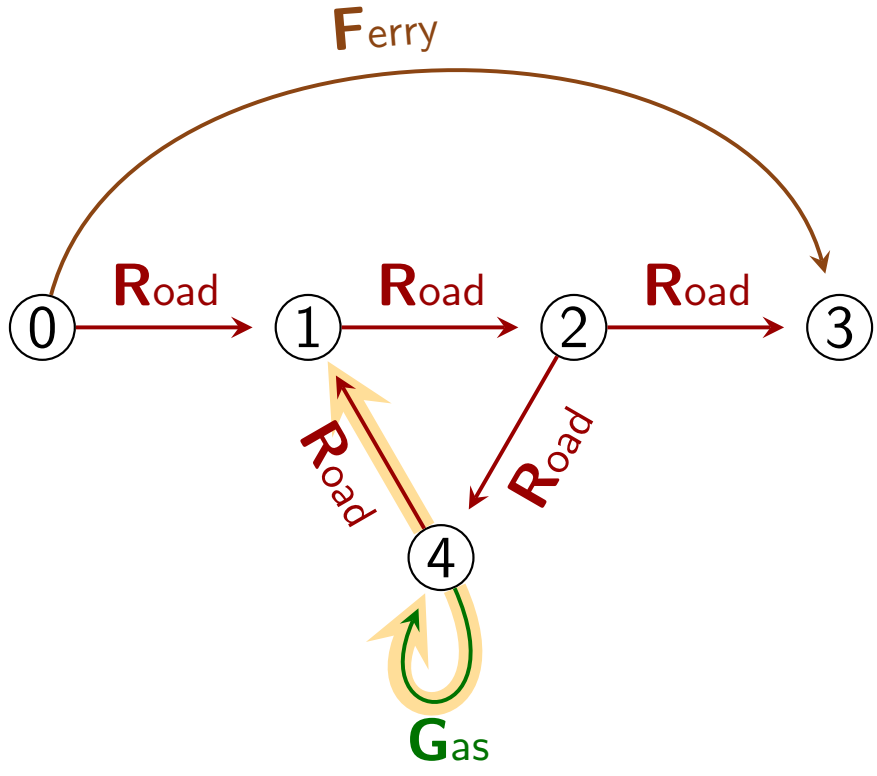
- A memory zone for each vertex and edge
- Each edge stores refs to source, label, target
- Each vertex stores refs to adjacent edges (usually indexed by label)



	Road	Ferry	Gas
0:	[1]	[3]	[]
1:	[2]	[]	[]
2:	[3,4]	[]	[]
3:	[]	[]	[]
4:			

Storing adjacency lists ("Native graph storage", ex: Neo4j)

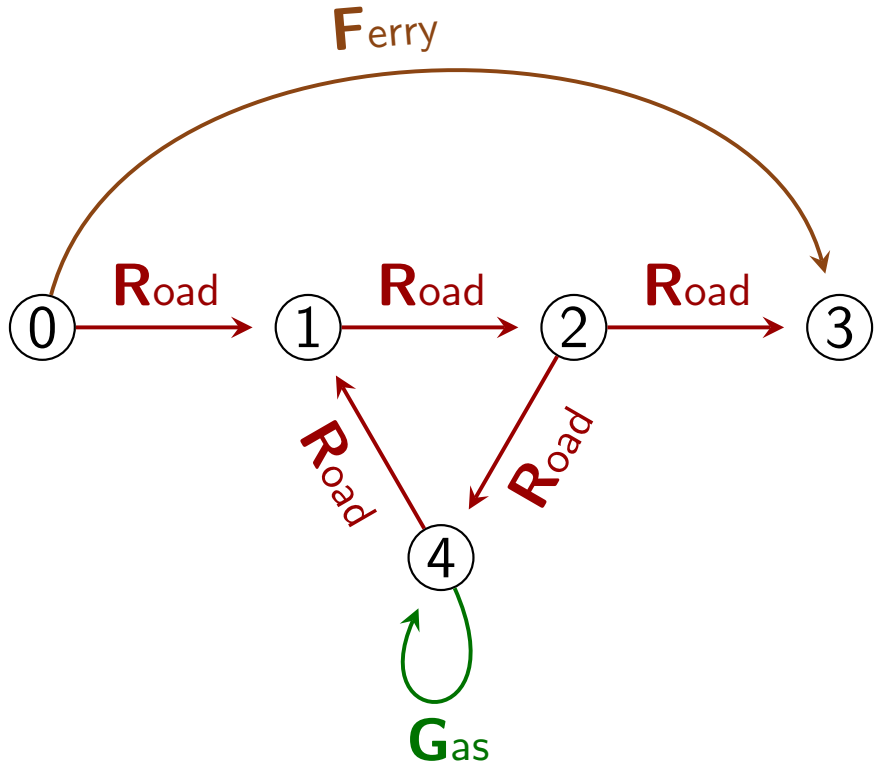
- A memory zone for each vertex and edge
- Each edge stores refs to source, label, target
- Each vertex stores refs to adjacent edges (usually indexed by label)



	Road	Ferry	Gas
0:	[1]	[3]	[]
1:	[2]	[]	[]
2:	[3,4]	[]	[]
3:	[]	[]	[]
4:	[1]	[]	[4]

Storing adjacency lists ("Native graph storage", ex: Neo4j)

- A memory zone for each vertex and edge
- Each edge stores refs to source, label, target
- Each vertex stores refs to adjacent edges (usually indexed by label)

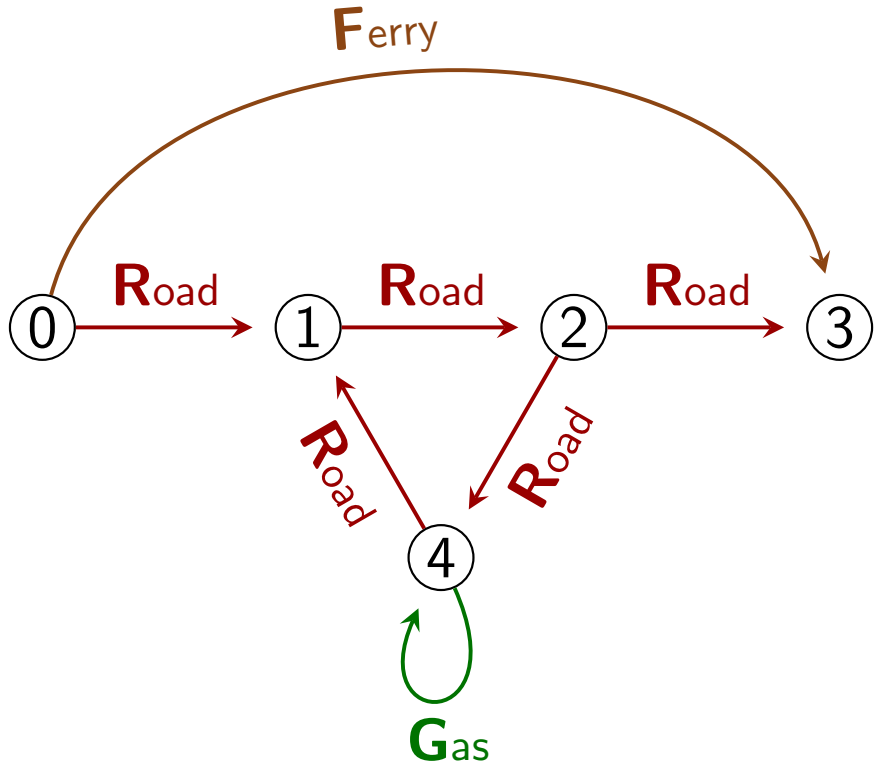


	Road	Ferry	Gas
0:	[1]	[3]	[]
1:	[2]	[]	[]
2:	[3,4]	[]	[]
3:	[]	[]	[]
4:	[1]	[]	[4]

- Edge test:
- Successors:

Storing adjacency lists ("Native graph storage", ex: Neo4j)

- A memory zone for each vertex and edge
- Each edge stores refs to source, label, target
- Each vertex stores refs to adjacent edges (usually indexed by label)

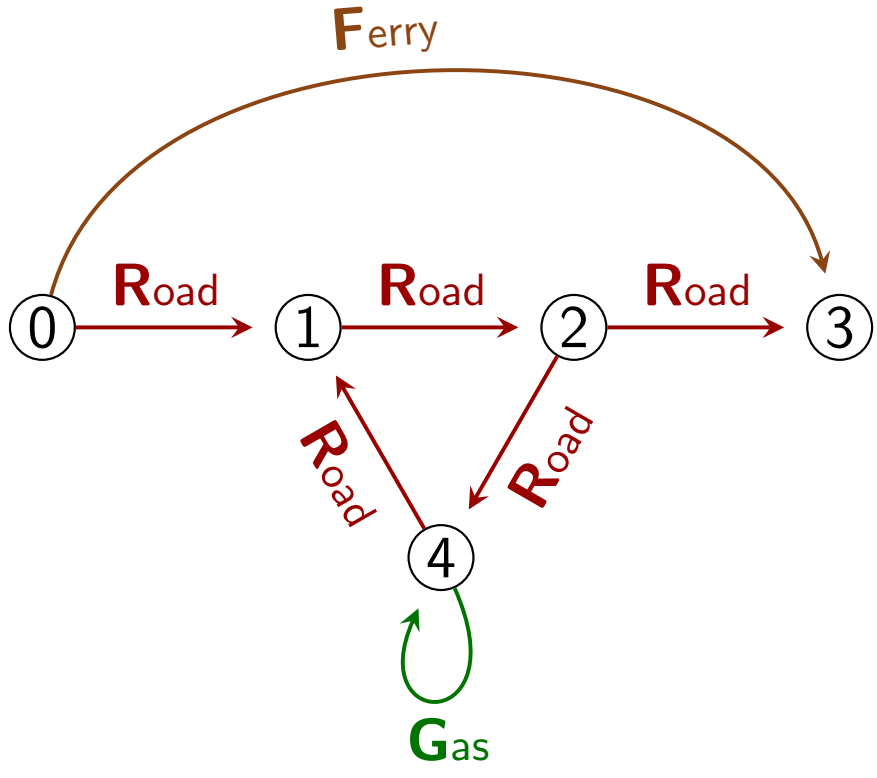


	Road	Ferry	Gas
0:	[1]	[3]	[]
1:	[2]	[]	[]
2:	[3,4]	[]	[]
3:	[]	[]	[]
4:	[1]	[]	[4]

- Edge test: $O(\#Successors)$
- Successors:

Storing adjacency lists ("Native graph storage", ex: Neo4j)

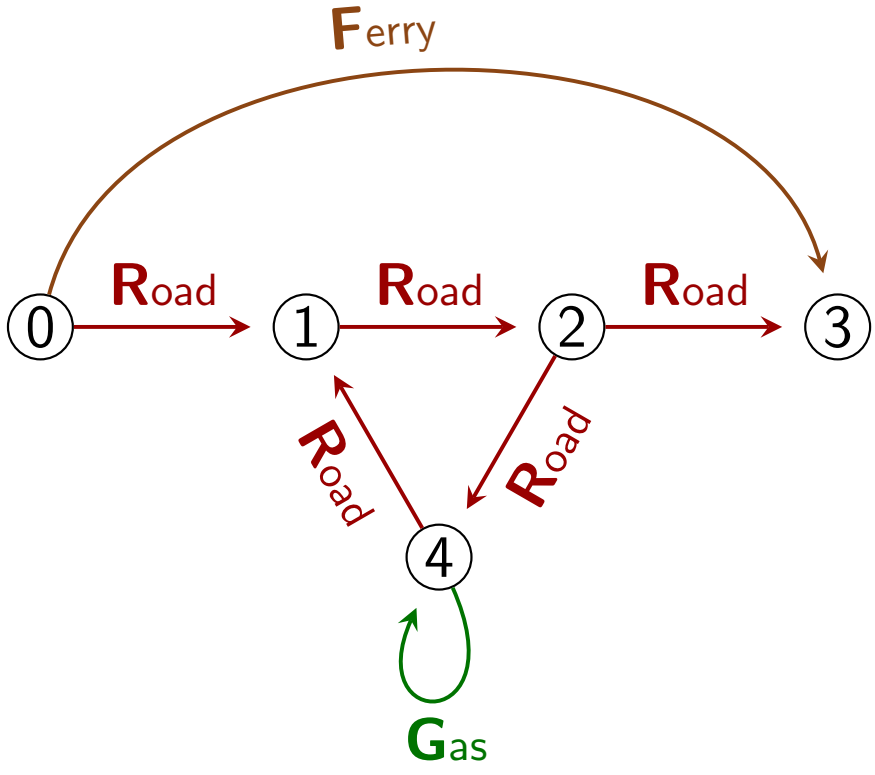
- A memory zone for each vertex and edge
- Each edge stores refs to source, label, target
- Each vertex stores refs to adjacent edges (usually indexed by label)



	Road	Ferry	Gas
0:	[1]	[3]	[]
1:	[2]	[]	[]
2:	[3,4]	[]	[]
3:	[]	[]	[]
4:	[1]	[]	[4]

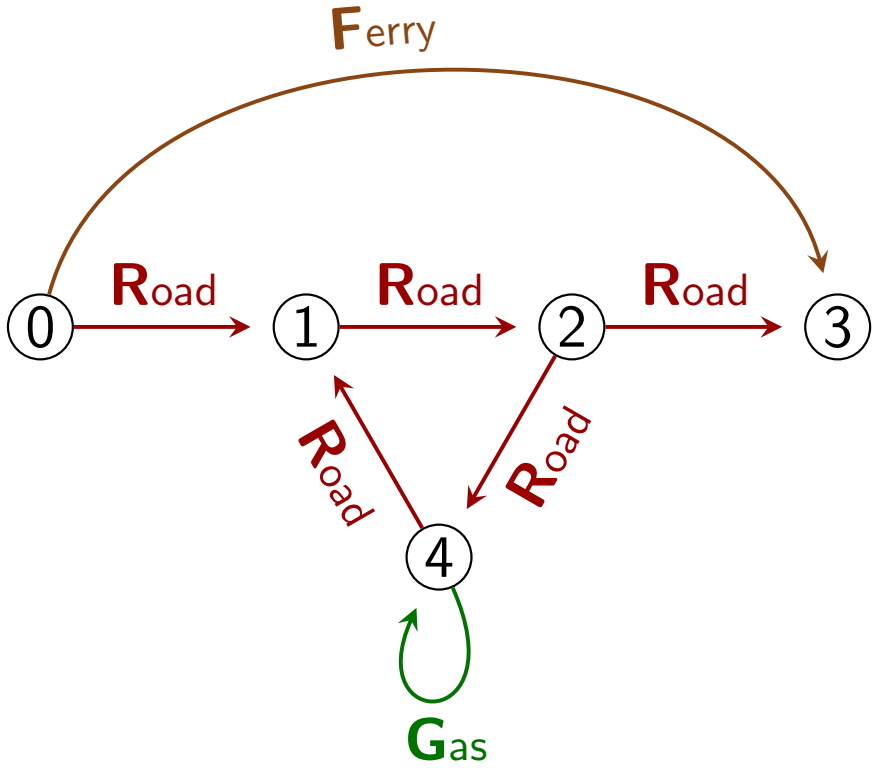
- Edge test: $O(\#Successors)$
- Successors: $O(\#Successors)$

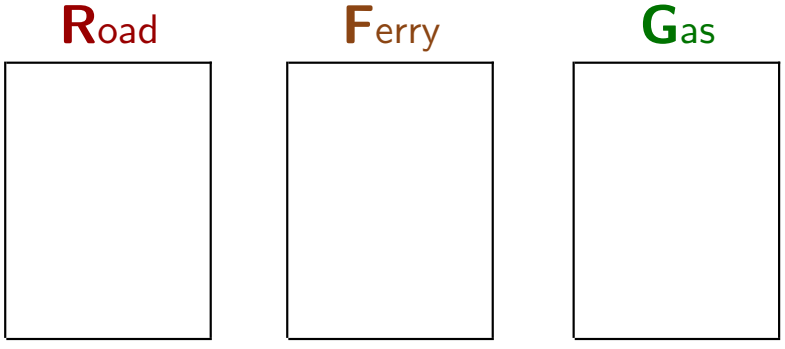
Storing adjacency matrices (ex: RedisGraph)



Storing adjacency matrices (ex: RedisGraph)

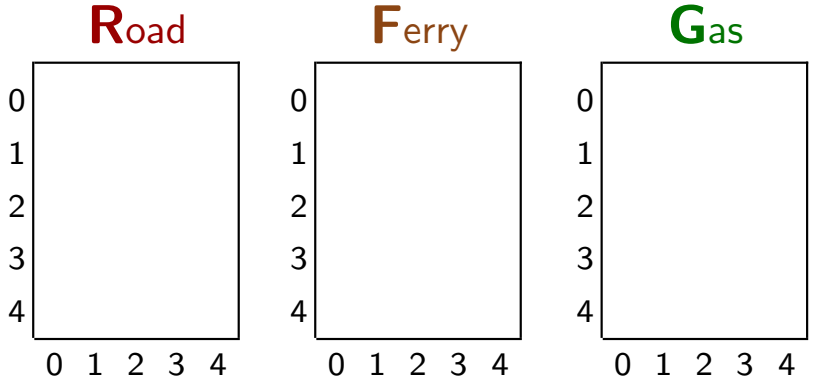
- One matrix per label

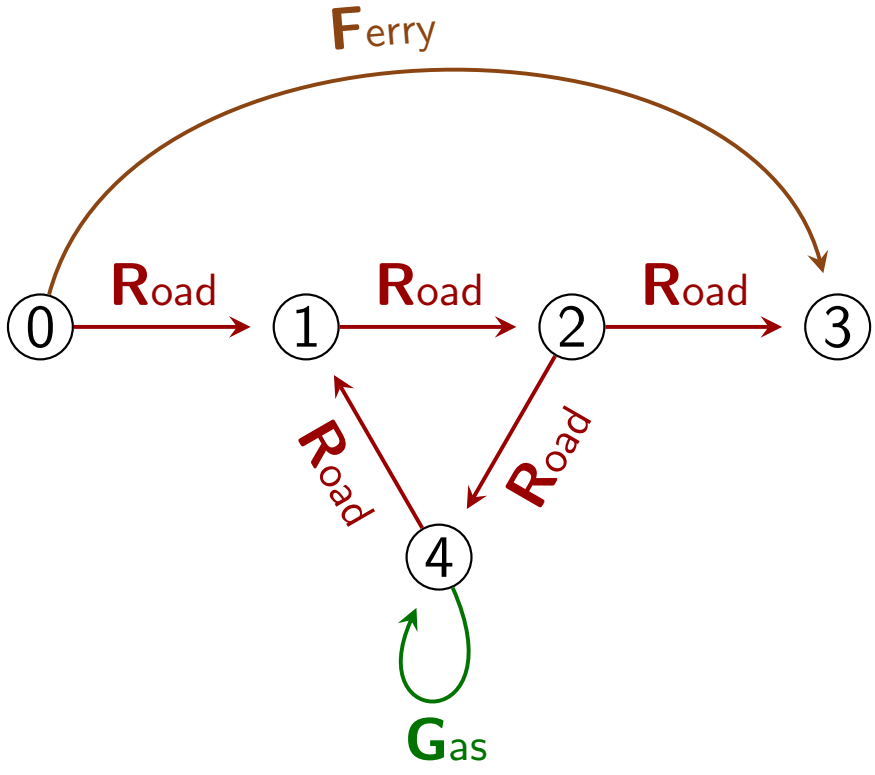




Storing adjacency matrices (ex: RedisGraph)

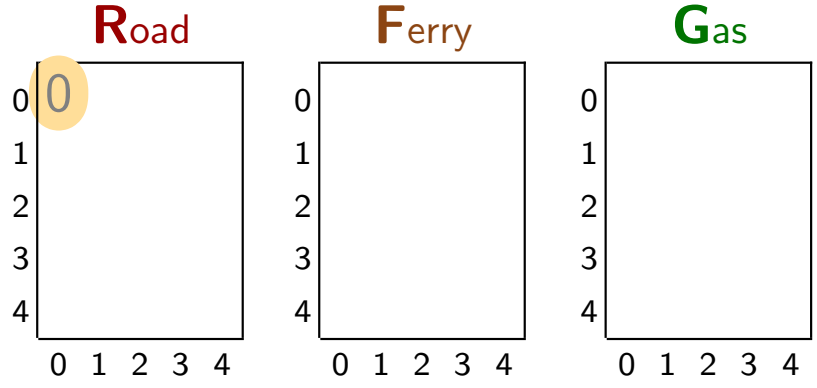
- One matrix per label
- One line per vertex
- One column per vertex



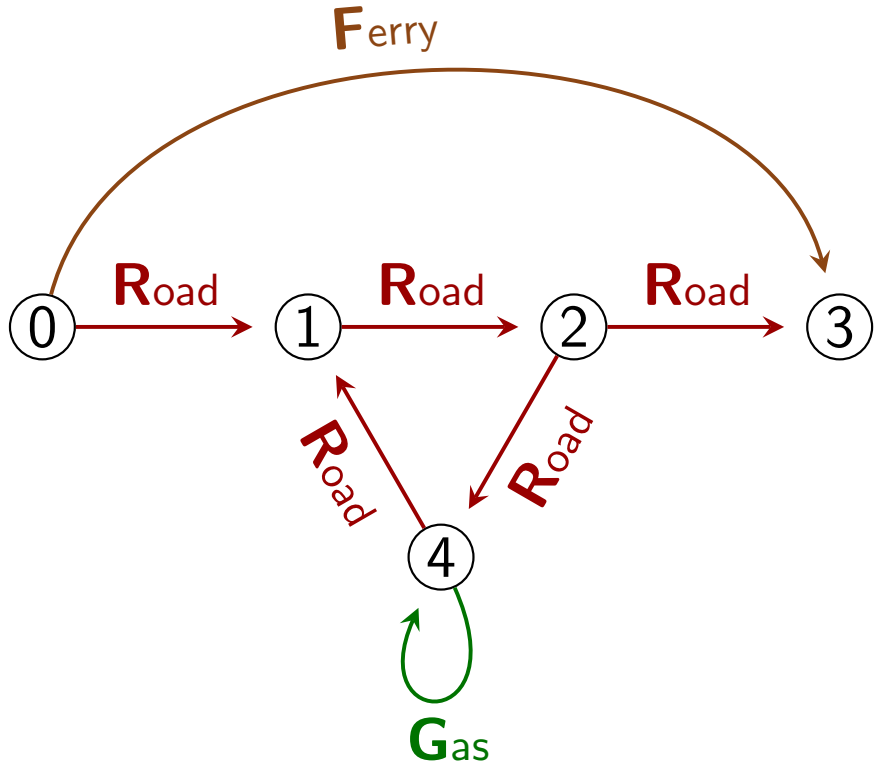


Storing adjacency matrices (ex: RedisGraph)

- One matrix per label
- One line per vertex
- One column per vertex
- Cell (i, j) in $L \iff i \xrightarrow{L} j$

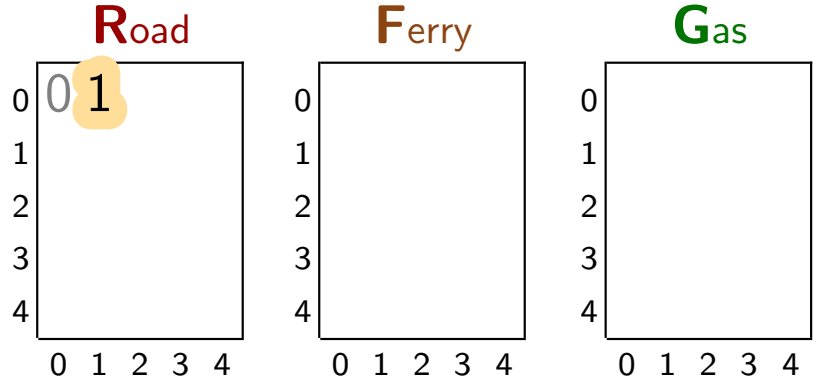


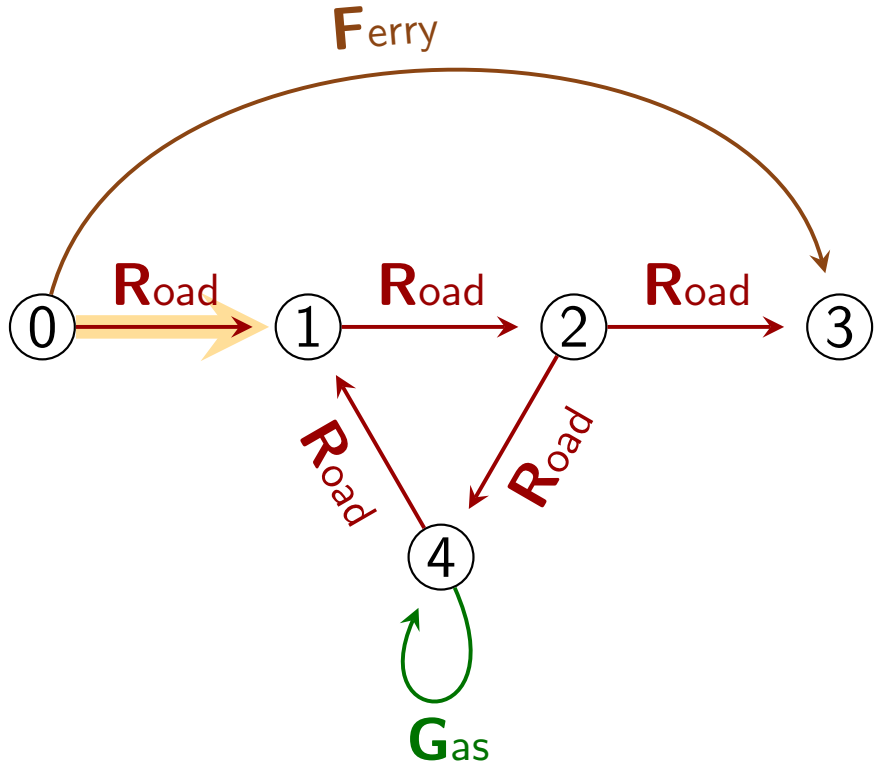
No edge $0 \xrightarrow{\text{Road}} 0$



Storing adjacency matrices (ex: RedisGraph)

- One matrix per label
- One line per vertex
- One column per vertex
- Cell (i, j) in $L \iff i \xrightarrow{L} j$

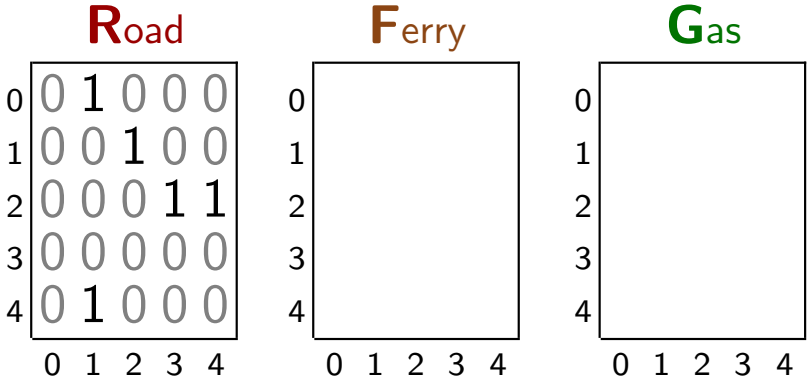


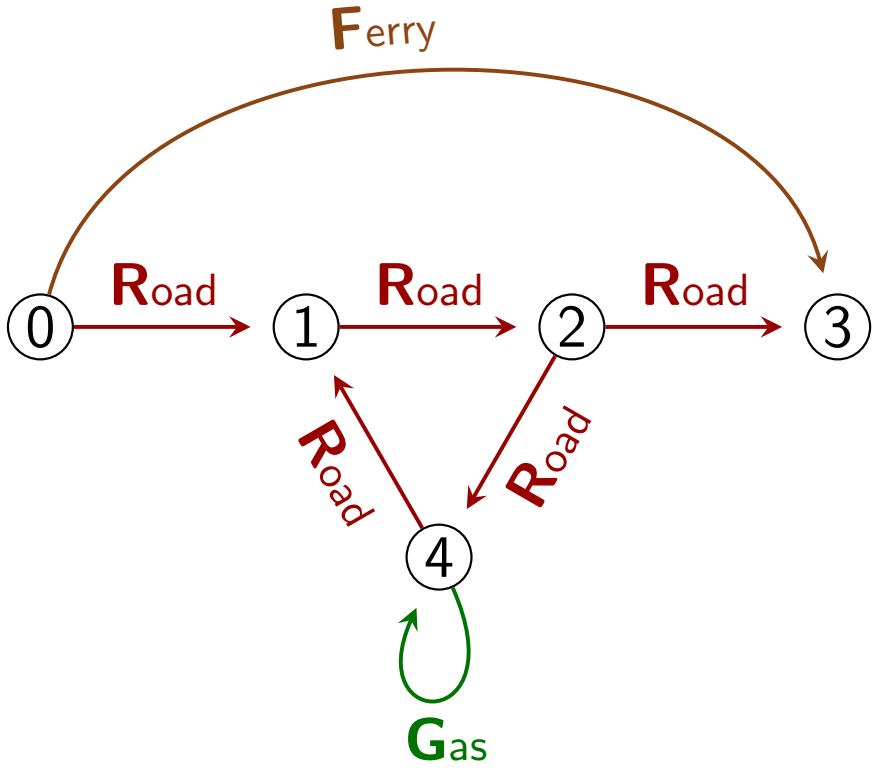


There is an edge $0 \xrightarrow{\text{Road}} 1$

Storing adjacency matrices (ex: RedisGraph)

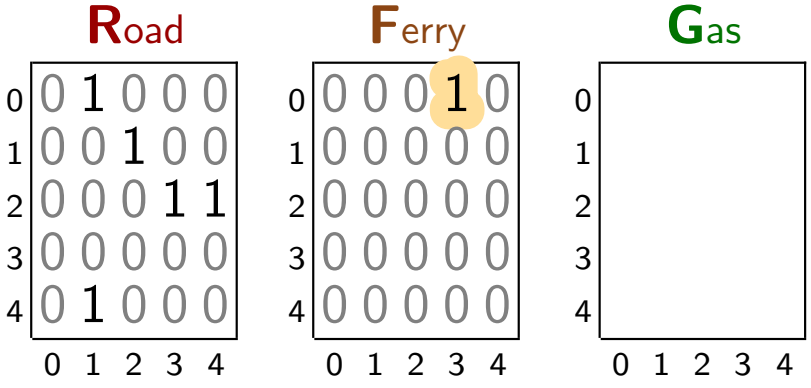
- One matrix per label
- One line per vertex
- One column per vertex
- Cell (i, j) in $L \iff i \xrightarrow{L} j$

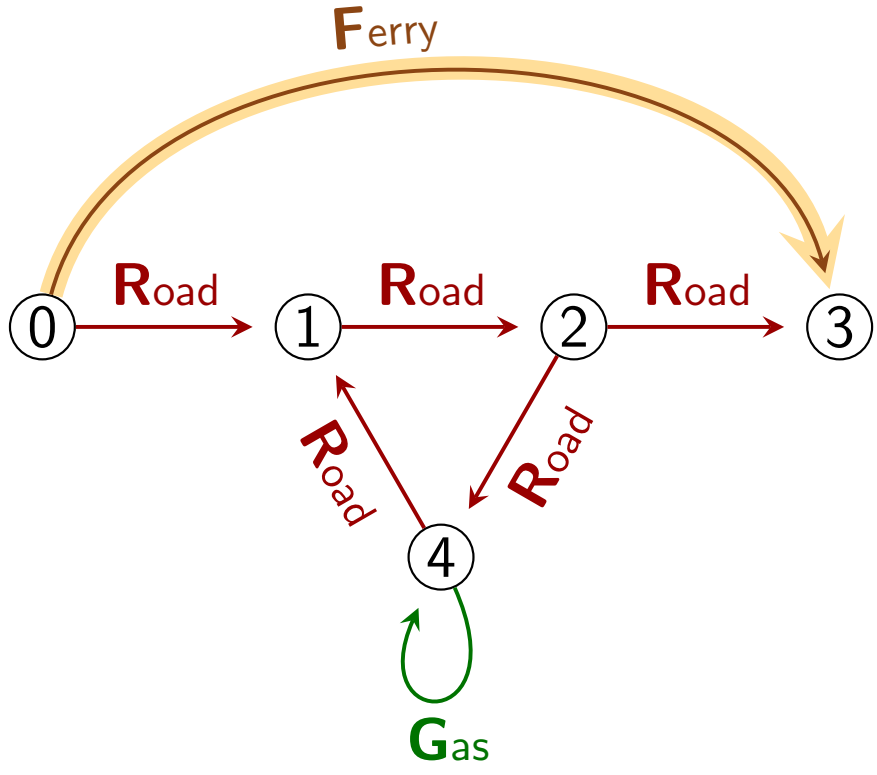




Storing adjacency matrices (ex: RedisGraph)

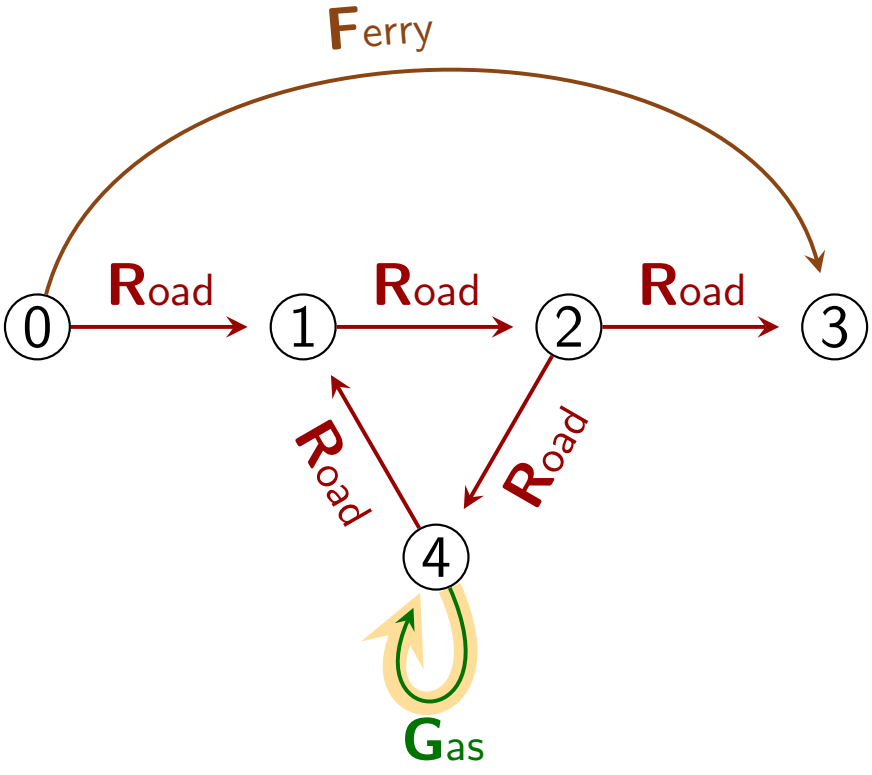
- One matrix per label
- One line per vertex
- One column per vertex
- Cell (i, j) in $L \iff i \xrightarrow{L} j$





Storing adjacency matrices (ex: RedisGraph)

- One matrix per label
- One line per vertex
- One column per vertex
- Cell (i, j) in $L \iff i \xrightarrow{L} j$



Road

	0	1	2	3	4
0	0	1	0	0	0
1	0	0	1	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0

Ferry

	0	1	2	3	4
0	0	0	0	1	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

Gas

	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	1

Storing adjacency matrices (ex: RedisGraph)

- One matrix per label
- One line per vertex
- One column per vertex
- Cell (i, j) in $L \iff i \xrightarrow{L} j$

Road

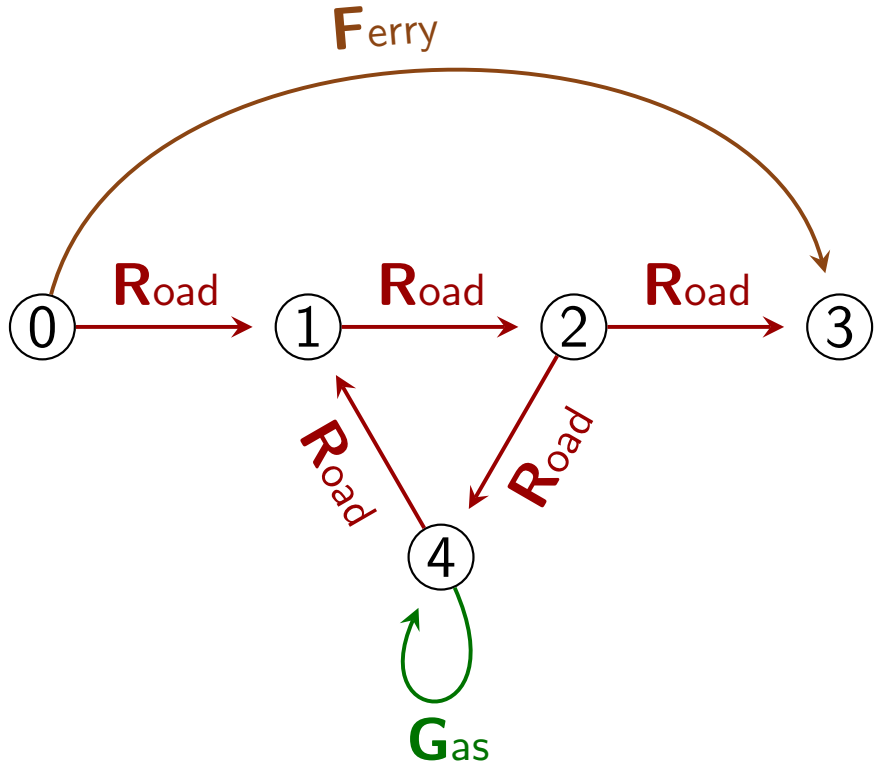
	0	1	2	3	4
0	0	1	0	0	0
1	0	0	1	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0

Ferry

	0	1	2	3	4
0	0	0	0	1	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

Gas

	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	1



- Edge test:
- Successors:

Storing adjacency matrices (ex: RedisGraph)

- One matrix per label
- One line per vertex
- One column per vertex
- Cell (i, j) in $L \iff i \xrightarrow{L} j$

Road

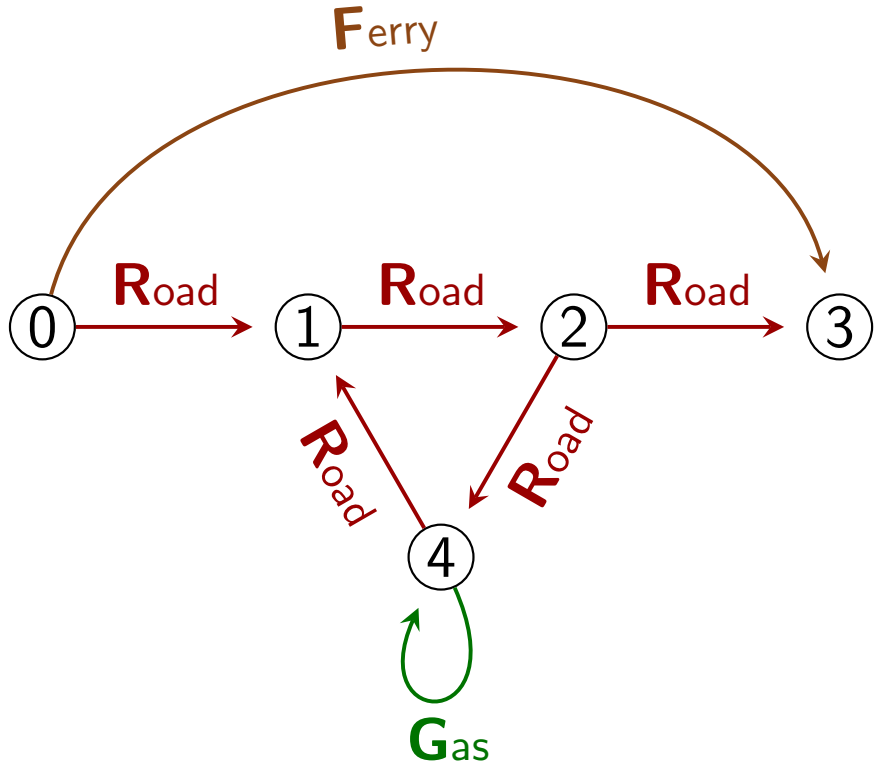
	0	1	2	3	4
0	0	1	0	0	0
1	0	0	1	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0

Ferry

	0	1	2	3	4
0	0	0	0	1	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

Gas

	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	1



- Edge test: O(1)
- Successors:

Storing adjacency matrices (ex: RedisGraph)

- One matrix per label
- One line per vertex
- One column per vertex
- Cell (i, j) in $L \iff i \xrightarrow{L} j$

Road

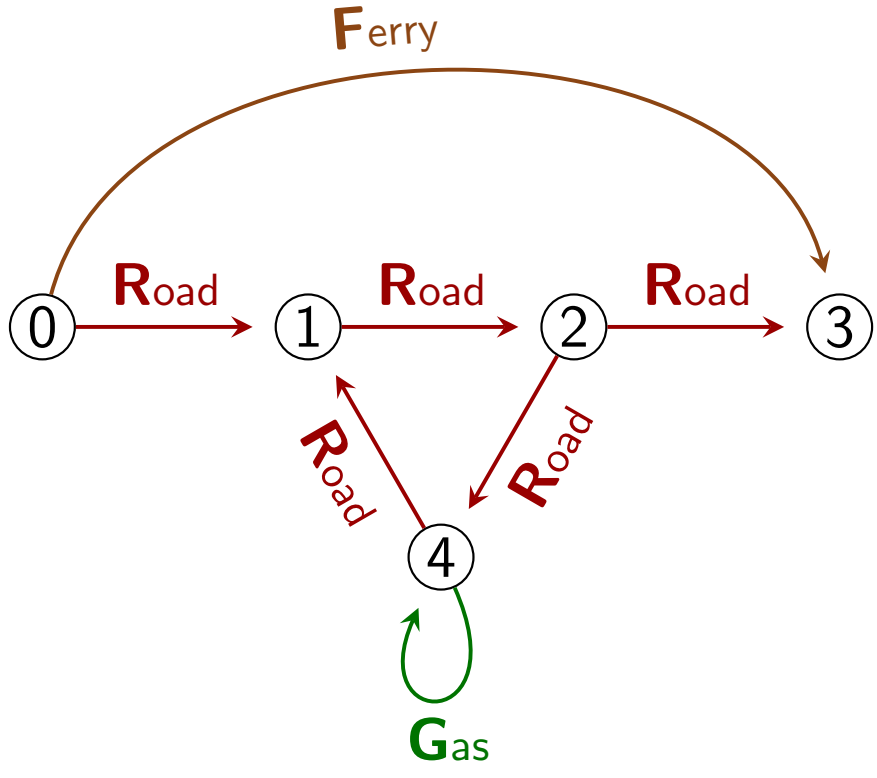
	0	1	2	3	4
0	0	1	0	0	0
1	0	0	1	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0

Ferry

	0	1	2	3	4
0	0	0	0	1	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

Gas

	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	1



- Edge test: O(1)
- Successors: (#Vertices)

Storing adjacency matrices (ex: RedisGraph)

- One matrix per label
- One line per vertex
- One column per vertex
- Cell (i, j) in $L \iff i \xrightarrow{L} j$

Road

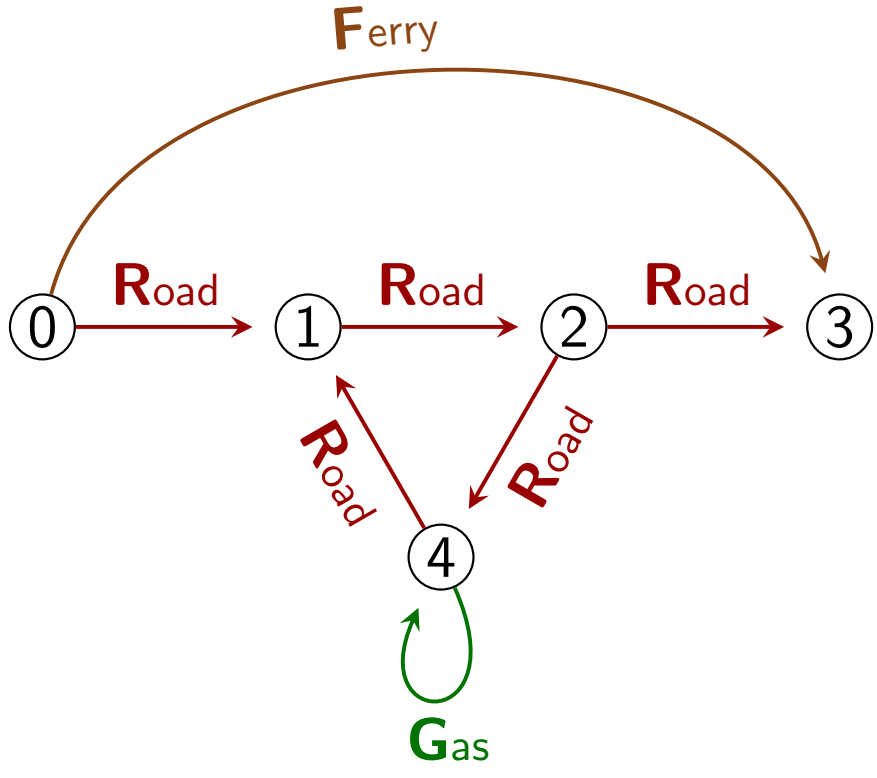
	0	1	2	3	4
0	0	1	0	0	0
1	0	0	1	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	1	0	0	0

Ferry

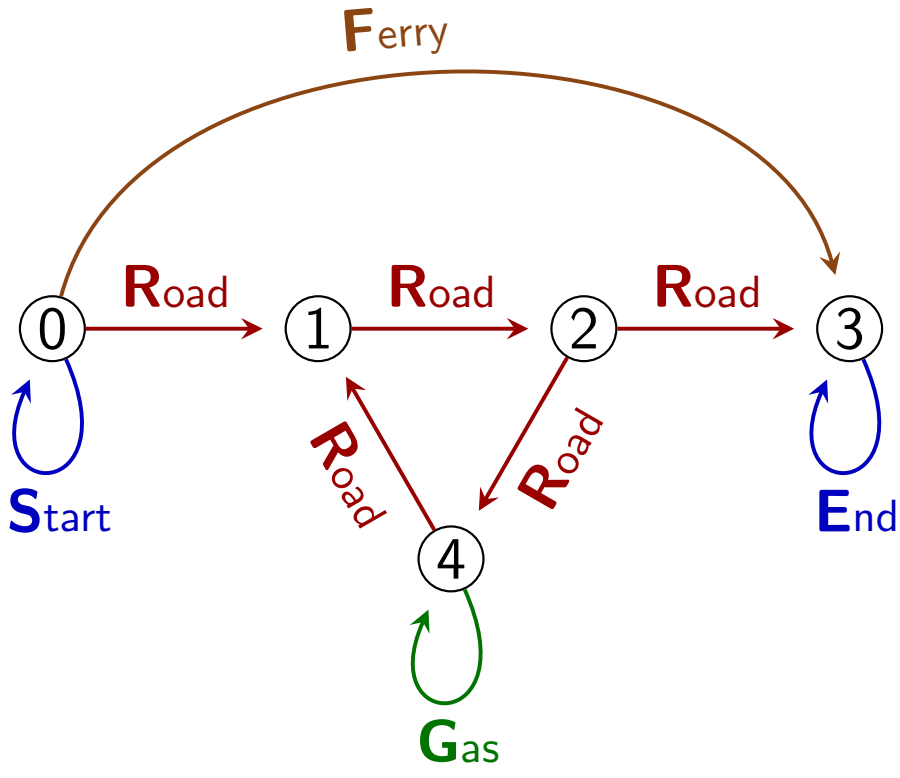
	0	1	2	3	4
0	0	0	0	1	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0

Gas

	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	1

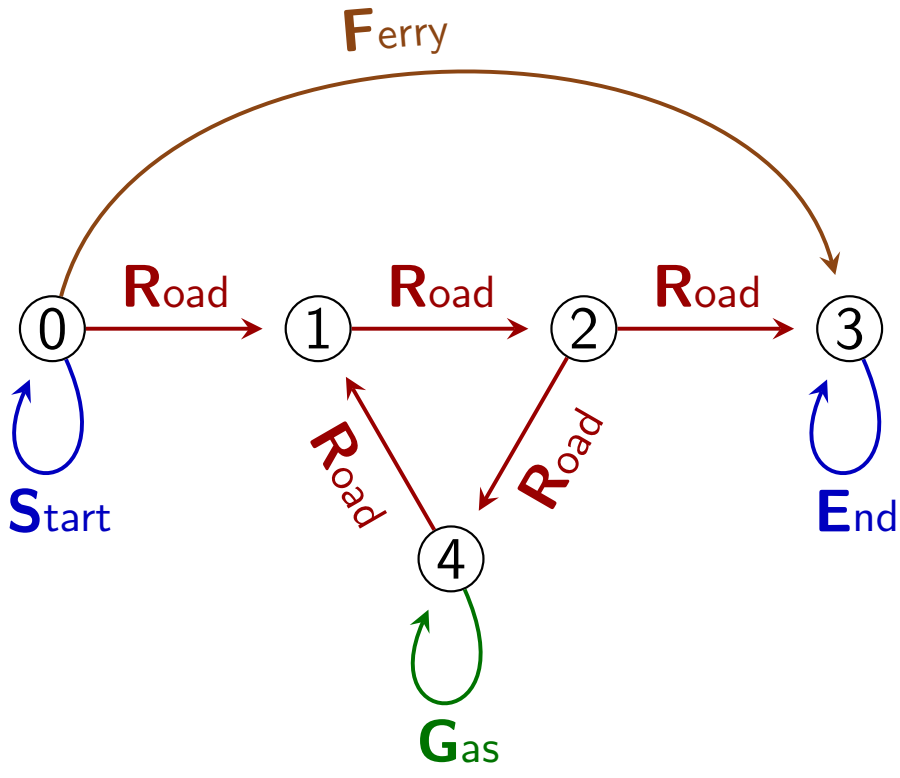


- Edge test: O(1)
- Successors: (#Vertices)
- Memory: $O((\#\text{Vertices})^2)$


Ferry
Road
Road
Road
0
1
2
3
Start
Road
Road
End
4
Gas

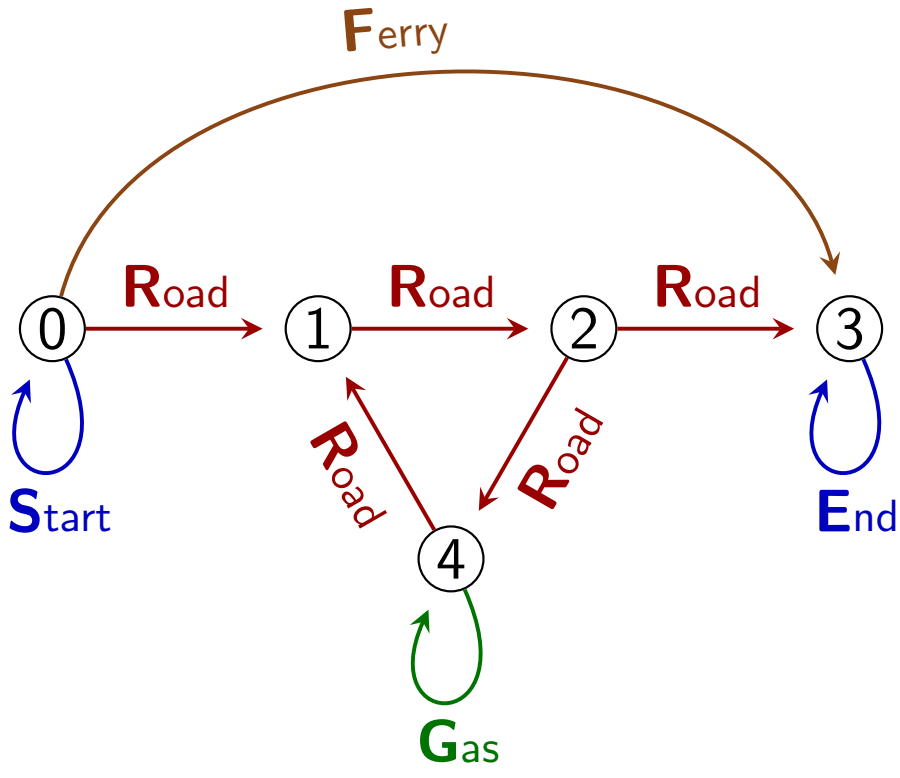
Storing edge tree sets (ex: Relational databases)

- One tree-set (table) for each edge type



Storing edge tree sets (ex: Relational databases)

- One tree-set (table) for each edge type



Road: $\{(0,1);(1,2);(2,3);$
$(2,4);(4,1)\}$

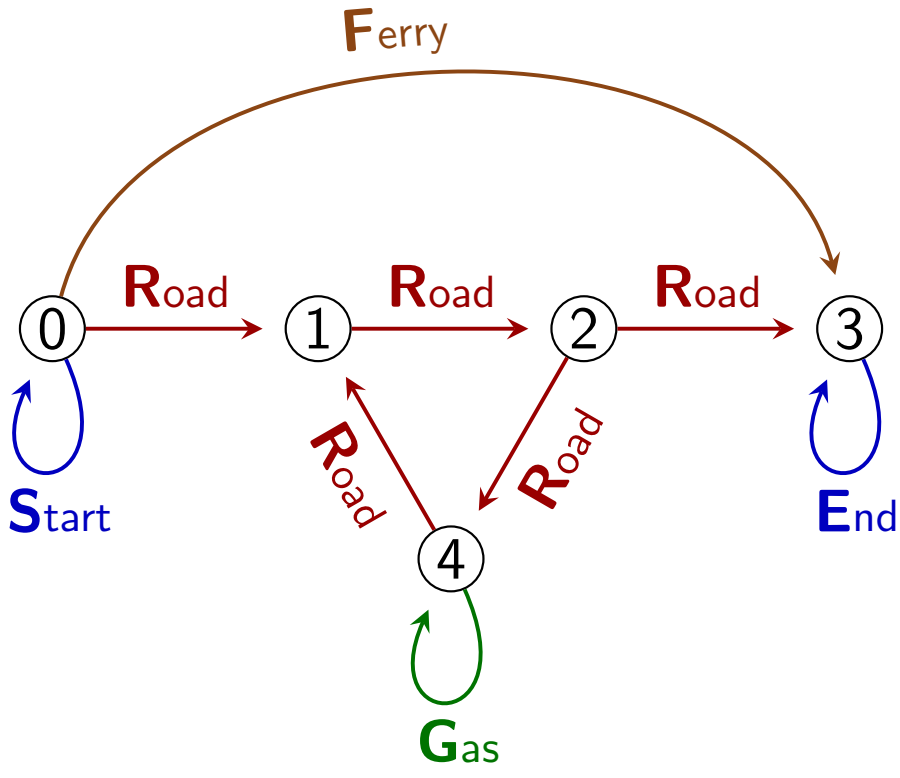
Ferry: $\{(0,3)\}$

Gas: $\{(4,4)\}$

- Edge test:
- Successors:

Storing edge tree sets (ex: Relational databases)

- One tree-set (table) for each edge type



Road: $\{(0,1);(1,2);(2,3);$
$(2,4);(4,1)\}$

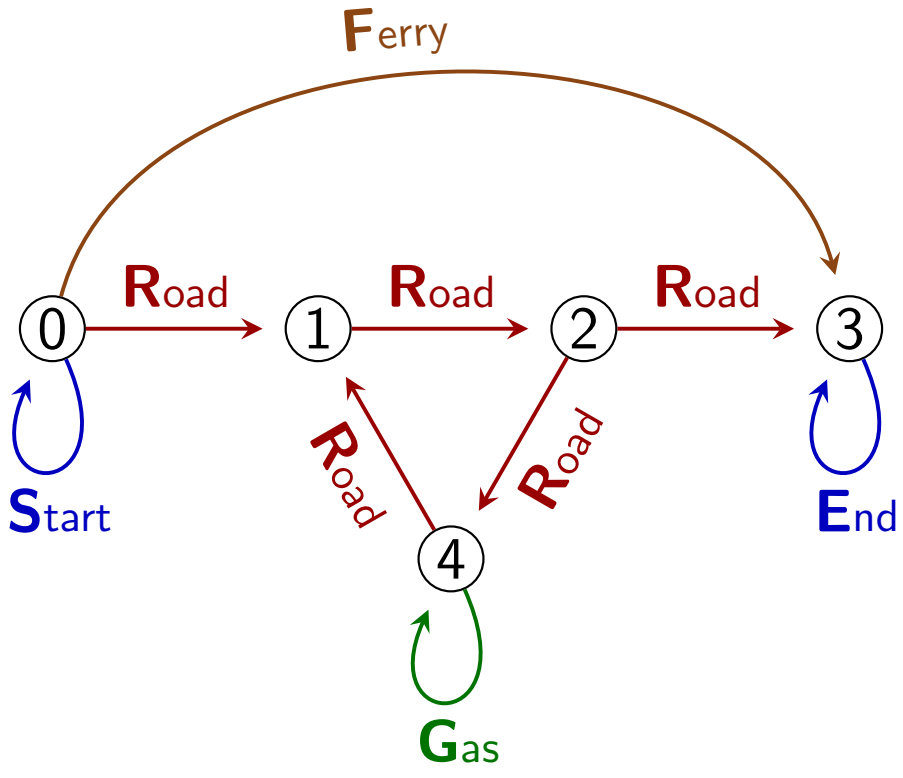
Ferry: $\{(0,3)\}$

Gas: $\{(4,4)\}$

- Edge test: $O(\log(\#Edges))$
- Successors:

Storing edge tree sets (ex: Relational databases)

- One tree-set (table) for each edge type



Road: $\{(0,1);(1,2);(2,3);$
$(2,4);(4,1)\}$

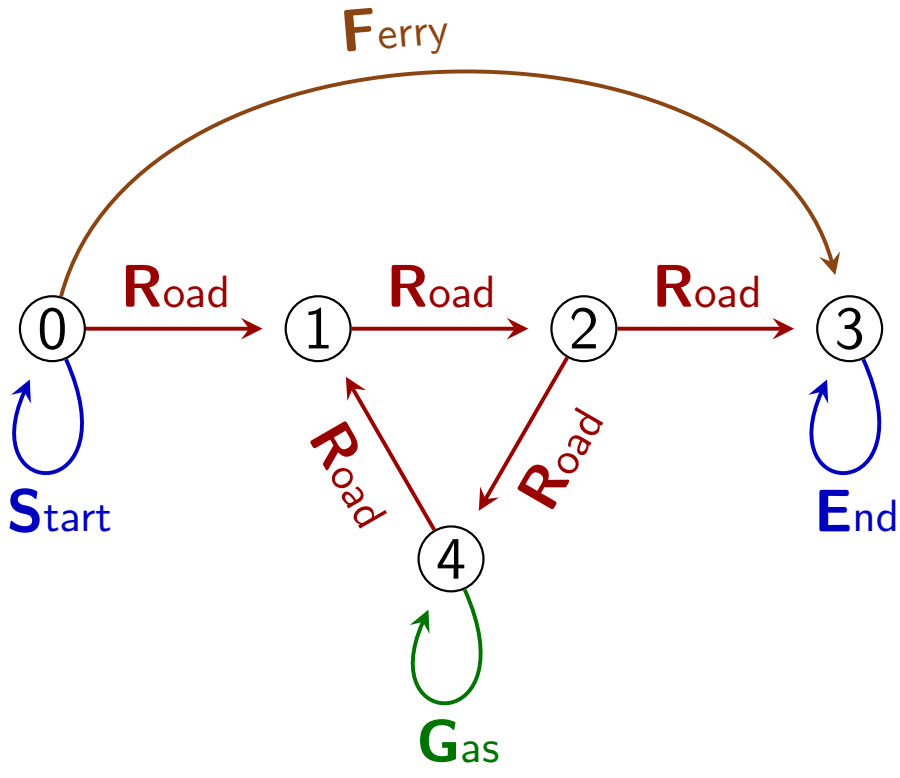
Ferry: $\{(0,3)\}$

Gas: $\{(4,4)\}$

- Edge test: $O(\log(\#Edges))$
- Successors: $\#Edges$

Storing edge tree sets (ex: Relational databases)

- One tree-set (table) for each edge type



Road: $\{(0,1);(1,2);(2,3);$
$(2,4);(4,1)\}$

Ferry: $\{(0,3)\}$

Gas: $\{(4,4)\}$

- Edge test: $O(\log(\#Edges))$
- Successors: $\#Edges$
  or $O(\#\log(Edges))$ if index

Recap of different storage options

	Edge test	Successors
Adjacency list	$O(\#Succ)$	$O(\#Succ)$
Adjacency matrix	$O(1)$	$O(\#Vert)$
Edge tree set (†)	$O(log(\#Edge))$	$O(log(\#Edge))$

(†) with proper indexing

Goal

Finding walks (e.g. matching an RPQ)

Which one seems better?

Adjacency list

- Memory zones contains property maps

Adjacency list

- Memory zones contains property maps

Adjacency matrix

- Cells in the matrix contains reference to edge content
- Property maps for edges and nodes

Storing properties

Adjacency list

- Memory zones contains property maps

Adjacency matrix

- Cells in the matrix contains reference to edge content
- Property maps for edges and nodes

Tree sets

- Properties are stored in other tables (see translation)

Part II: Property Graphs

5. Conclusion

Strength of property graph DBMS (1)

Native storage

- Elementary graph operations are efficient
- Access to property is efficient
- Query answering is based on graph algorithms and not on joins
  Ex: $\mathbf{S}(\mathbf{R}+\mathbf{F})^2$, $\mathbf{S}(\mathbf{R}+\mathbf{F})^3$, $\mathbf{S}(\mathbf{R}+\mathbf{F})^*$
- Allows flexible schemas or a schema-less approach

Strength of property graph DBMS (1)

Native storage

- Elementary graph operations are efficient
- Access to property is efficient
- Query answering is based on graph algorithms and not on joins
  Ex: $\mathbf{S}(\mathbf{R}+\mathbf{F})^2$, $\mathbf{S}(\mathbf{R}+\mathbf{F})^3$, $\mathbf{S}(\mathbf{R}+\mathbf{F})^*$
- Allows flexible schemas or a schema-less approach


Some PG DBMS's do not use native storage


Strength of property graph DBMS (2)

Specialized algorithms and languages

Restriction on the DM increases the liberty in the query language.

"We never have to treat the case of non-binary relations"

- Graph notions in the core of the language (path as values)
- Graph algorithms directly available

Strength of property graph DBMS (3)

Easier to grasp for humans

- Easier modeling
  "The data looks like the ER diagram"
- Direct data visualization
  "One may navigate in the data"

Strength of property graph DBMS (3)

Easier to grasp for humans

- Easier modeling
  "The data looks like the ER diagram"
- Direct data visualization
  "One may navigate in the data"
- Visualization of query result$^{(\dagger)}$
  ($\dagger$) Arguable, see part **III**.

Strength of property graph DBMS (3)

Easier to grasp for humans

- Easier modeling
  "The data looks like the ER diagram"
- Direct data visualization
  "One may navigate in the data"
- Visualization of query result$^{(\dagger)}$
  ($\dagger$) Arguable, see part **III**.
- Query languages can be made user-friendly
  "What you write looks like what you search for"

Strength of property graph DBMS (3)

Easier to grasp for humans

- Easier modeling
  "The data looks like the ER diagram"
- Direct data visualization
  "One may navigate in the data"
- Visualization of query result$^{(\dagger)}$
  ($\dagger$) Arguable, see part **III**.
- Query languages can be made user-friendly
  "What you write looks like what you search for"

$\Longrightarrow$ Property graphs are usable by non-experts

Ex: Panama papers

Weaknesses of property graph DBMS (1)

Efficiency

- Efficiency gain from native graph storage can be mitigated
  - Proper indexing
  - Worst-case optimal join
  - Highly structured data cannot be leveraged in PG's

  Ex: RDF engines usually do not use native graph storage

Weaknesses of property graph DBMS (1)

Efficiency

- Efficiency gain from native graph storage can be mitigated
  - Proper indexing
  - Worst-case optimal join
  - Highly structured data cannot be leveraged in PG's

  Ex: RDF engines usually do not use native graph storage
- Efficiency falls off if the need is outside the scope

Weaknesses of property graph DBMS (1)

Efficiency

- Efficiency gain from native graph storage can be mitigated
  - Proper indexing
  - Worst-case optimal join
  - Highly structured data cannot be leveraged in PG's

  Ex: RDF engines usually do not use native graph storage
- Efficiency falls off if the need is outside the scope
  - Non-navigational queries

Weaknesses of property graph DBMS (1)

Efficiency

- Efficiency gain from native graph storage can be mitigated
  - Proper indexing
  - Worst-case optimal join
  - Highly structured data cannot be leveraged in PG's

  Ex: RDF engines usually do not use native graph storage
- Efficiency falls off if the need is outside the scope
  - Non-navigational queries
  - When walks are not needed

Weaknesses of property graph DBMS (1)

Efficiency

- Efficiency gain from native graph storage can be mitigated
  - Proper indexing
  - Worst-case optimal join
  - Highly structured data cannot be leveraged in PG's

  Ex: RDF engines usually do not use native graph storage
- Efficiency falls off if the need is outside the scope
  - Non-navigational queries
  - When walks are not needed
  - Analytics (even graph analytics)

Weaknesses of property graph DBMS (2)

Too specialized?

- PG does not handle well some data

  Ex: ternary relations, extremely large values, disconnected data

Weaknesses of property graph DBMS (2)

Too specialized?

- PG does not handle well some data

  Ex: ternary relations, extremely large values, disconnected data

- Way less PG DBMS experts than Rel DBMS experts

Weaknesses of property graph DBMS (2)

Too specialized?

- PG does not handle well some data

  Ex: ternary relations, extremely large values, disconnected data
- Way less PG DBMS experts than Rel DBMS experts
- Relational databases may simulate property graphs

Weaknesses of property graph DBMS (2)

Too specialized?

- PG does not handle well some data

  Ex: ternary relations, extremely large values, disconnected data
- Way less PG DBMS experts than Rel DBMS experts
- Relational databases may simulate property graphs
  - User-friendly visualization and modeling tools can be made

Weaknesses of property graph DBMS (2)

Too specialized?

- PG does not handle well some data

  Ex: ternary relations, extremely large values, disconnected data
- Way less PG DBMS experts than Rel DBMS experts
- Relational databases may simulate property graphs
  - User-friendly visualization and modeling tools can be made
  - Graph-views can be made on top of Relational DBMS

Weaknesses of property graph DBMS (2)

Too specialized?

- PG does not handle well some data

  Ex: ternary relations, extremely large values, disconnected data

- Way less PG DBMS experts than Rel DBMS experts
- Relational databases may simulate property graphs
  - User-friendly visualization and modeling tools can be made
  - Graph-views can be made on top of Relational DBMS
  - Cypher is hard to translate in SQL...

Weaknesses of property graph DBMS (2)

Too specialized?

- PG does not handle well some data

  Ex: ternary relations, extremely large values, disconnected data

- Way less PG DBMS experts than Rel DBMS experts
- Relational databases may simulate property graphs
  - User-friendly visualization and modeling tools can be made
  - Graph-views can be made on top of Relational DBMS
  - Cypher is hard to translate in SQL...

    ...but SQL/PGQ brings it into SQL

Essential knowledge in part II

- Property graph data model:
  - Definition
  - Bad modeling
  - Different storage options
  - Strengths and Weaknesses of those
- Data model translations:
  - Property Graph $\leftrightarrow$ Tables (easy)
  - Tables $\leftrightarrow$ Property Graph (harder)
  - Reification

Part III: Cypher

Part III: Cypher

1. General presentation

A Cypher query

- queries a property graph
- returns a table

$\Rightarrow$**heterogeneous**.

Example of Cypher query:

```
MATCH (u1)-[p1:POSTED]->(m1)
  WHERE p1.id = 22
WITH u1.name AS uname,
     p1.on AS date,
     m1.text AS msg
RETURN *
```

Example of table:

uname	mdate	msg
"Alice"	"05-14"	"Hello"
"Bob"	"05-15"	"World"

A Cypher query

- queries a property graph
- returns a table

⇒**heterogeneous**.

A Cypher query

- Is a sequence of **clauses** and **sub-clauses**
  - `MATCH`, `WITH`, `RETURN`; `WHERE`
  - last clause is always `RETURN`

Example of Cypher query:

```
MATCH (u1)-[p1:POSTED]->(m1)
  WHERE p1.id = 22
WITH u1.name AS uname,
     p1.on AS date,
     m1.text AS msg
RETURN *
```

Example of table:

uname	mdate	msg
"Alice"	"05-14"	"Hello"
"Bob"	"05-15"	"World"

A Cypher query

- queries a property graph
- returns a table

⇒**heterogeneous**.

A Cypher query

- Is a sequence of **clauses** and **sub-clauses**
  - `MATCH`, `WITH`, `RETURN`; `WHERE`
  - last clause is always `RETURN`

A Cypher query

- manipulates intermediary tables
- uses **variables** → column names
  - `u1`, `p1`, `uname`, `mdate`, etc.

Example of Cypher query:

```
MATCH (u1)-[p1:POSTED]->(m1)
  WHERE p1.id = 22
WITH u1.name AS uname,
     p1.on AS date,
     m1.text AS msg
RETURN *
```

Example of table:

uname	mdate	msg
"Alice"	"05-14"	"Hello"
"Bob"	"05-15"	"World"

Values in Cypher

- **Values** are the elements that may appear in tables
- **Pure values** are the values with no reference to the graph
- **Property** is a key to pure values mapping

Values are

- Base values Ex: `true`, `42`, `"NoSQL"`
- Graph elements Ex: nodes, relations
- Paths (alternate lists of nodes and relations)
- List of values Ex: `[1,"Hello",true,"World",`N_1`]`
- Property dictionary Ex: `{name:"Victor", age:35}`

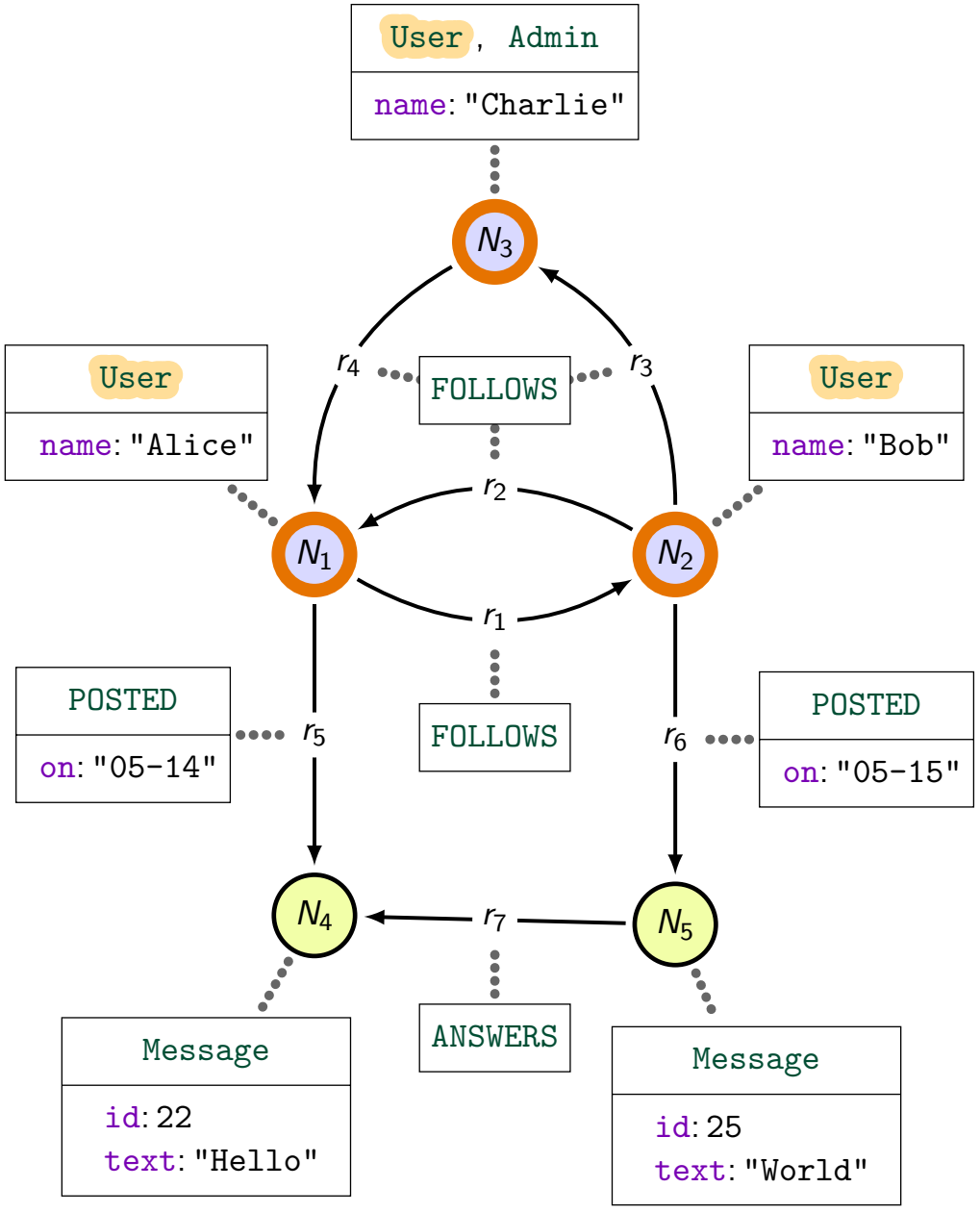
Warning

- The `RETURN` clause is mandatory
- For the sake of simplicity, we **omit it** in the following
- You will need to add one in the TP about Cypher
- Note that: `RETURN *` is a noop
- Add `RETURN *` to my queries to make them syntactically correct

Part III: Cypher

2. Pattern matching with `MATCH`

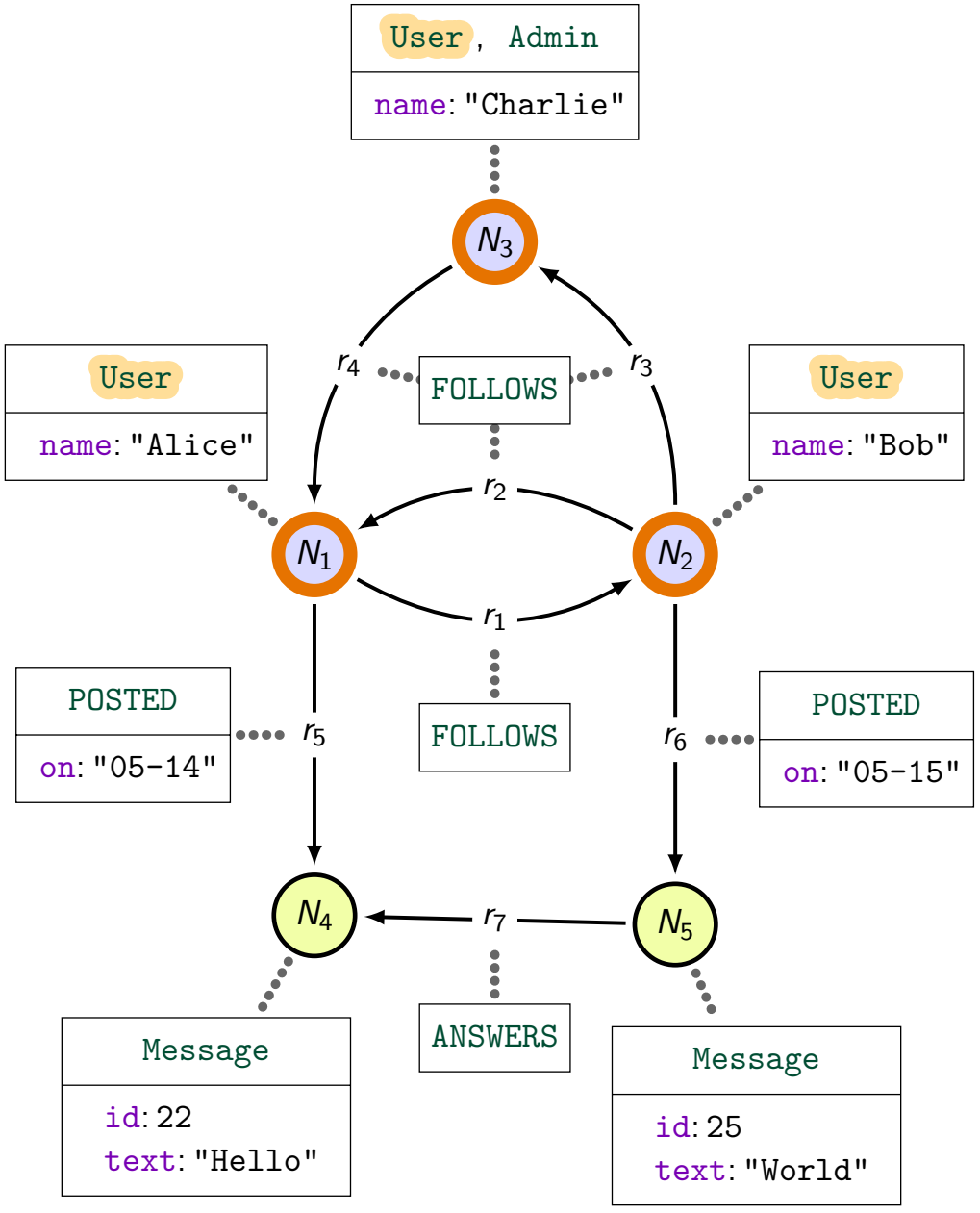
Matching nodes (1)



Query:

```
MATCH (u1:User)
```

Matching nodes (1)



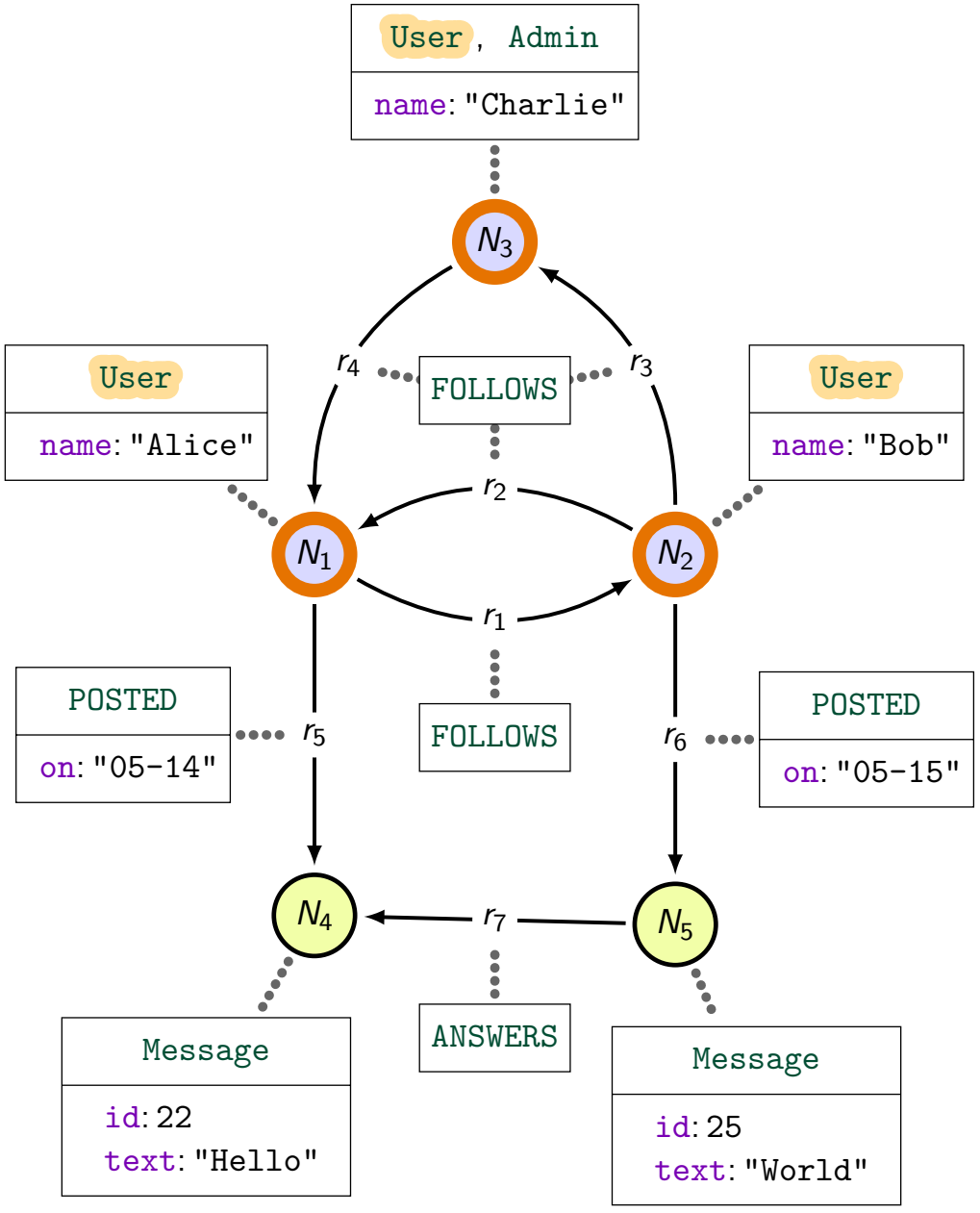
Query:

```
MATCH (u1:User)
```

Result:

u1
N_1
N_2
N_3

Matching nodes (1)



Query:

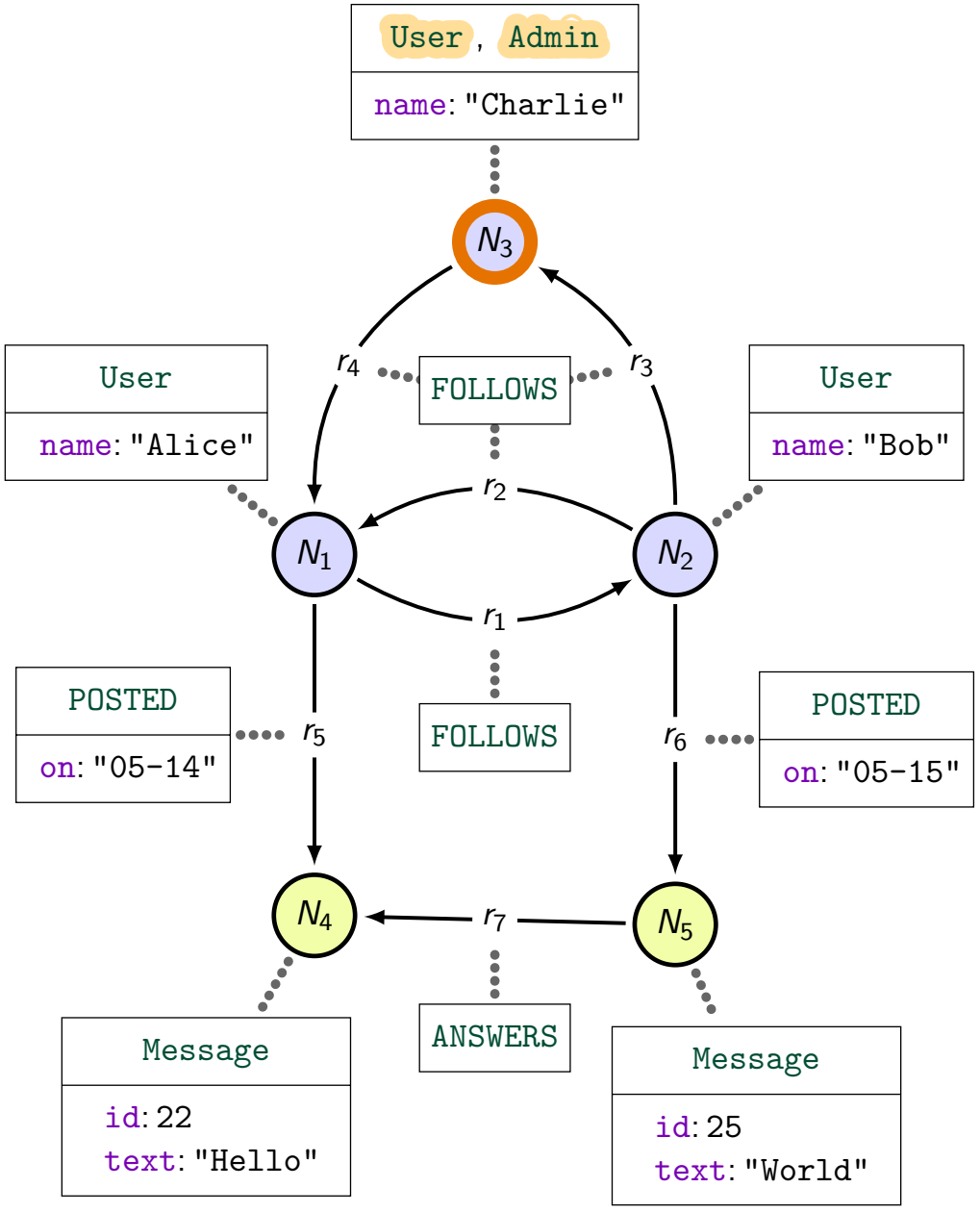
```
MATCH (u1:User)
```

Result:

u1
N_1
N_2
N_3

- Node patterns are between ( )
- Variable: first element
- Labels are prefixed by :

Matching nodes (2)



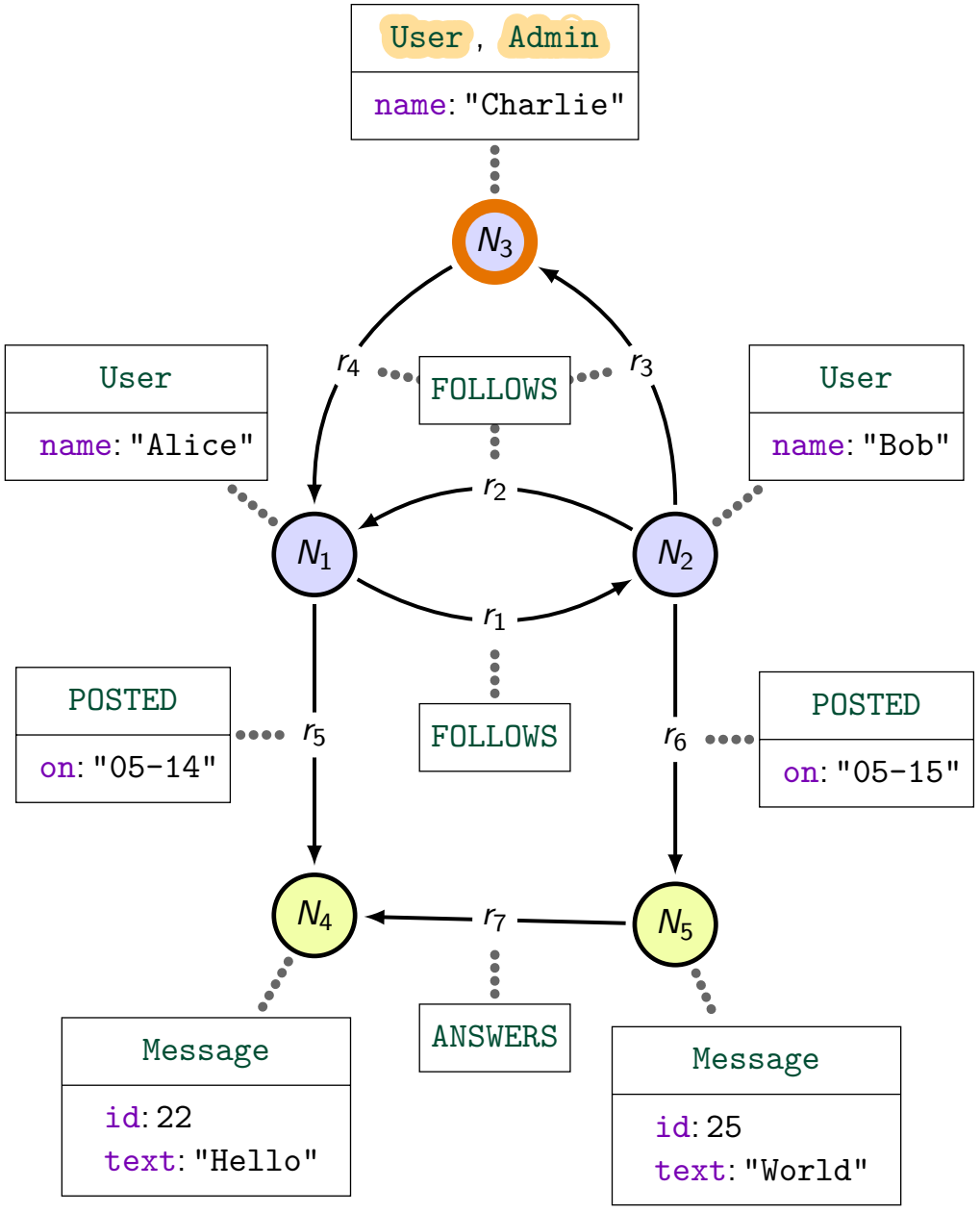
Query:

```
MATCH (u1:User:Admin)
```

Result:

u1
N_3

Matching nodes (2)



Query:

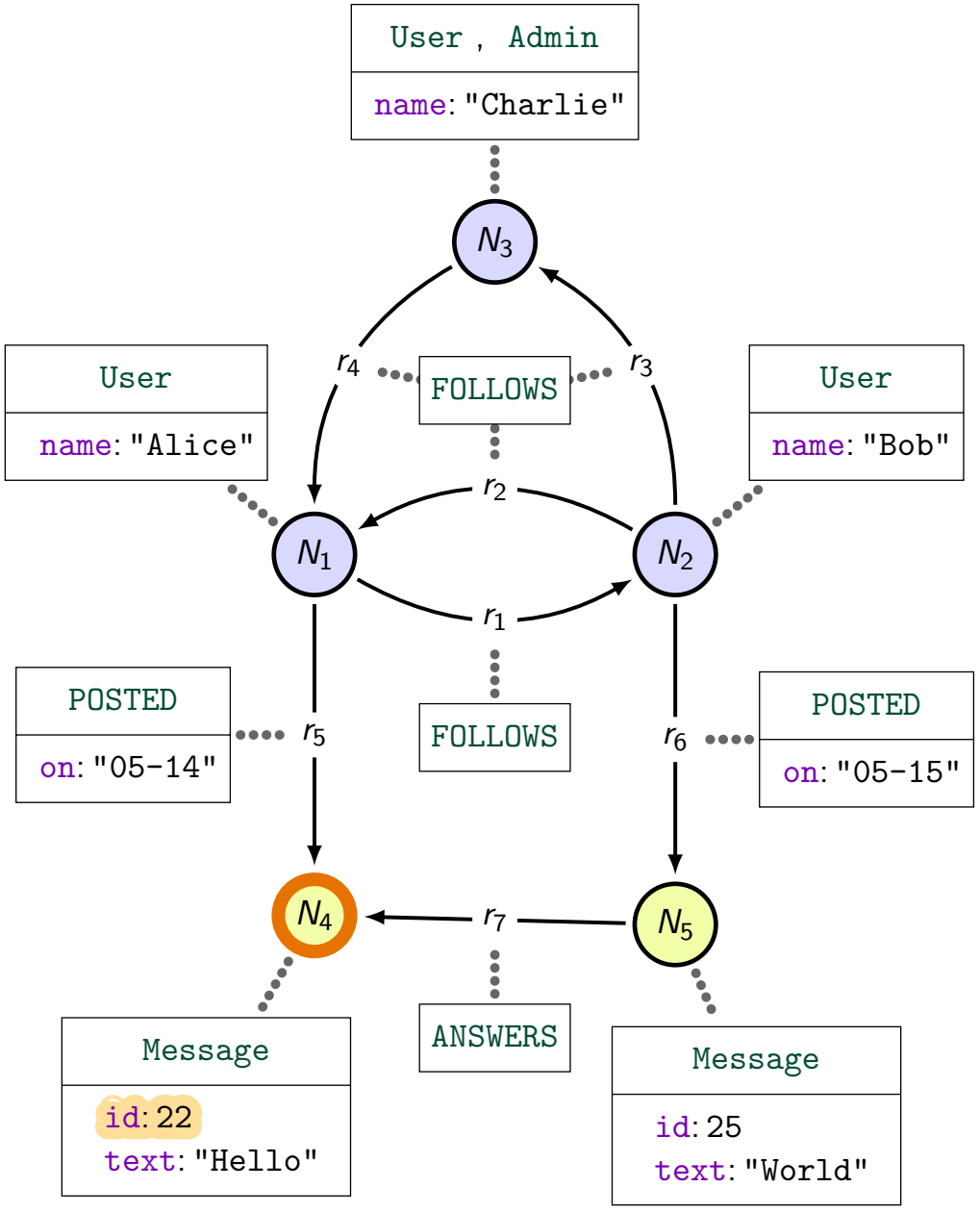
```
MATCH (u1:User:Admin)
```

Result:

u1
N_3

- Multiple labels: conjunction

Matching nodes (3)



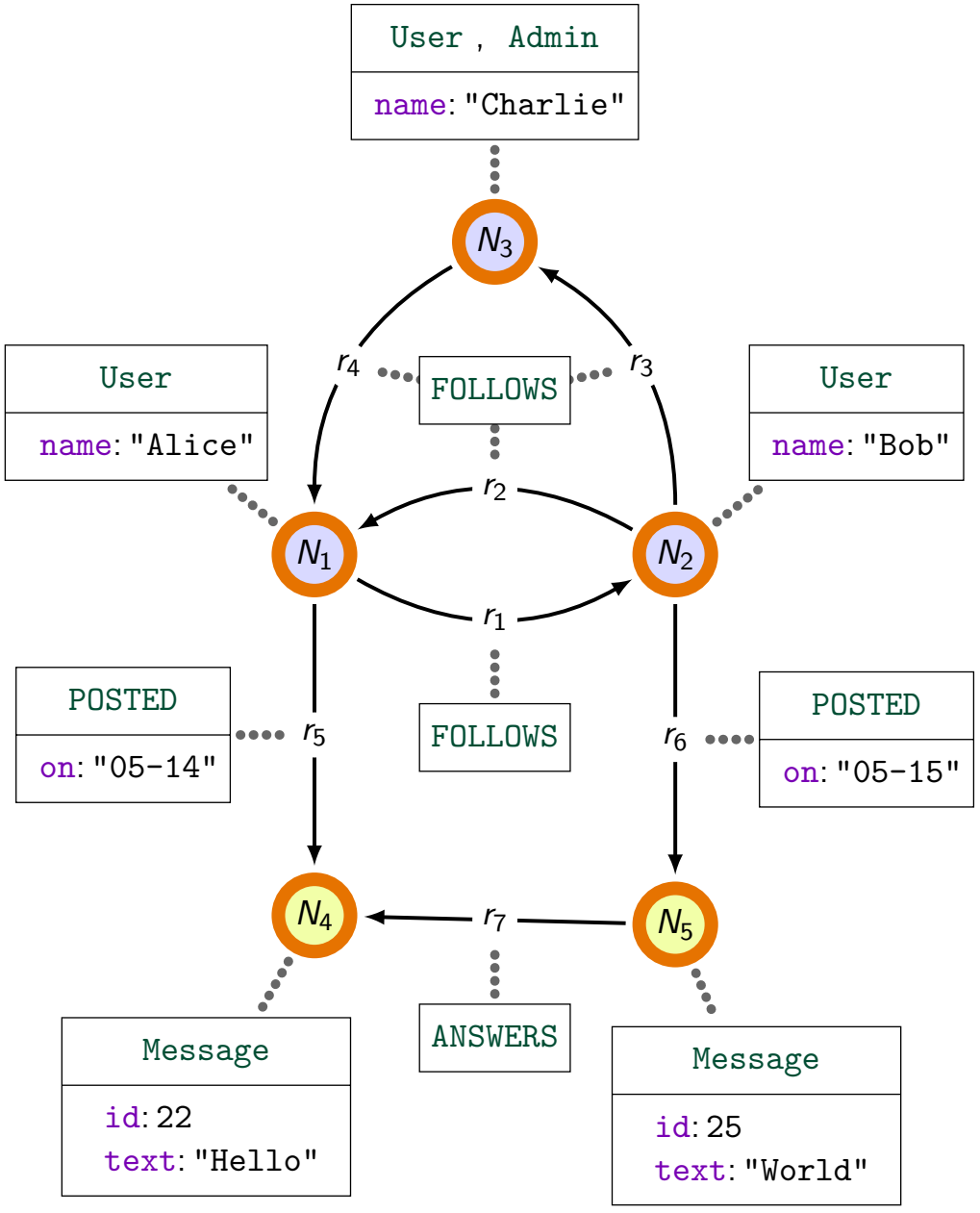
Query:

```
MATCH (u1{id:22})
```

Result:

u1
N_4

Matching nodes (4)

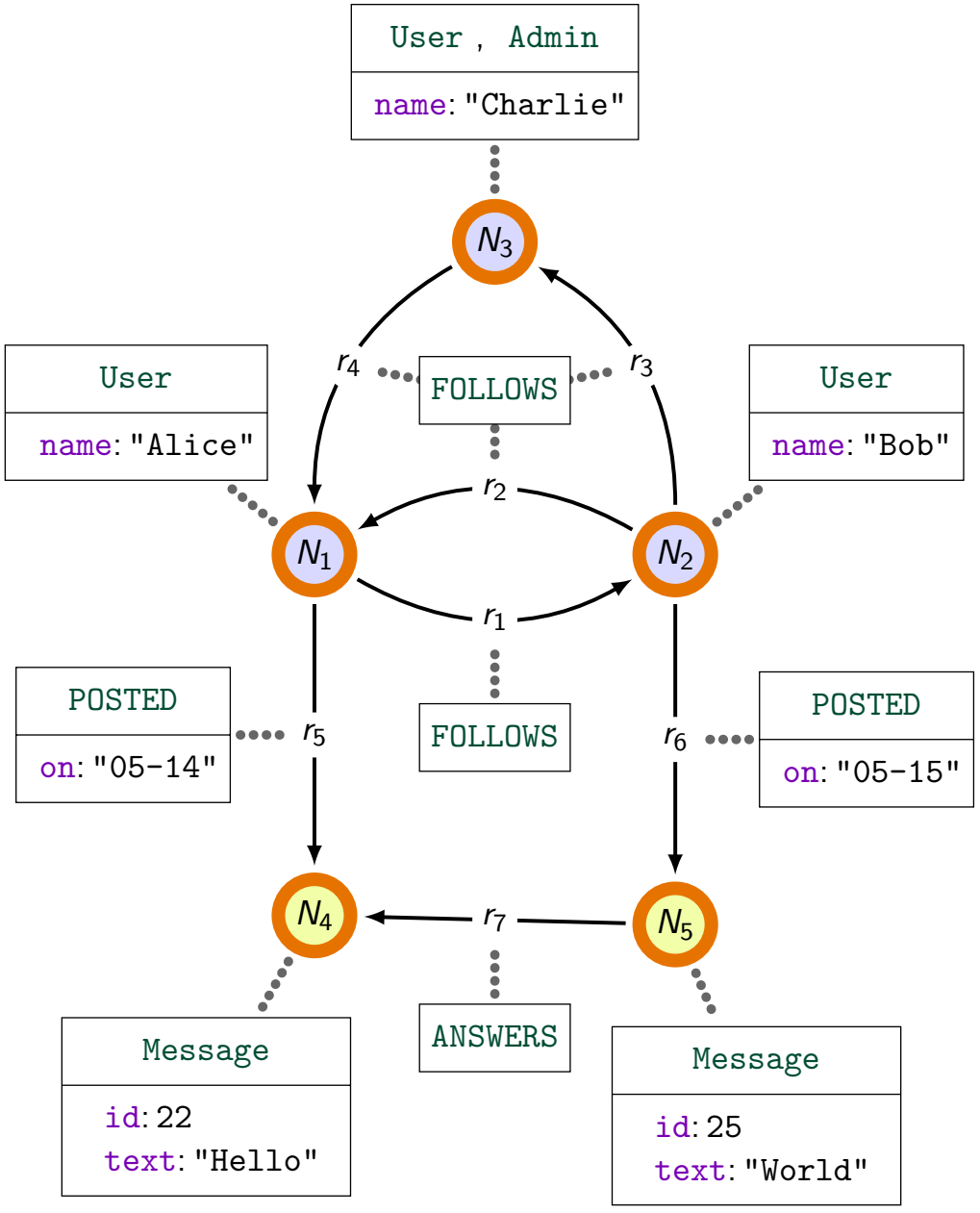


Query:

```
MATCH (u1)
```

Result:

Matching nodes (4)



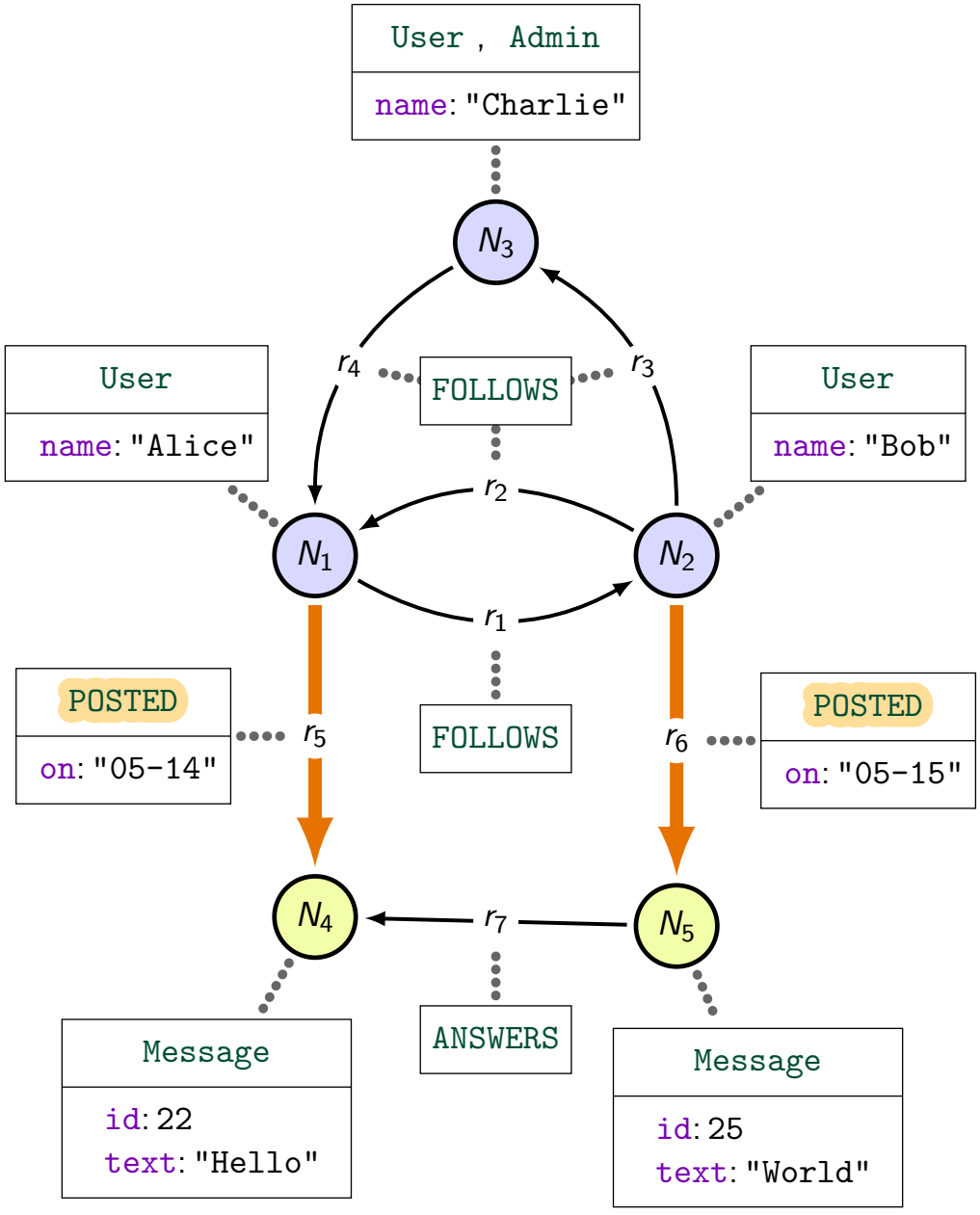
Query:

```
MATCH (u1)
```

Result:

u1
N_1
N_2
N_3
N_4
N_5

Matching relations (1)



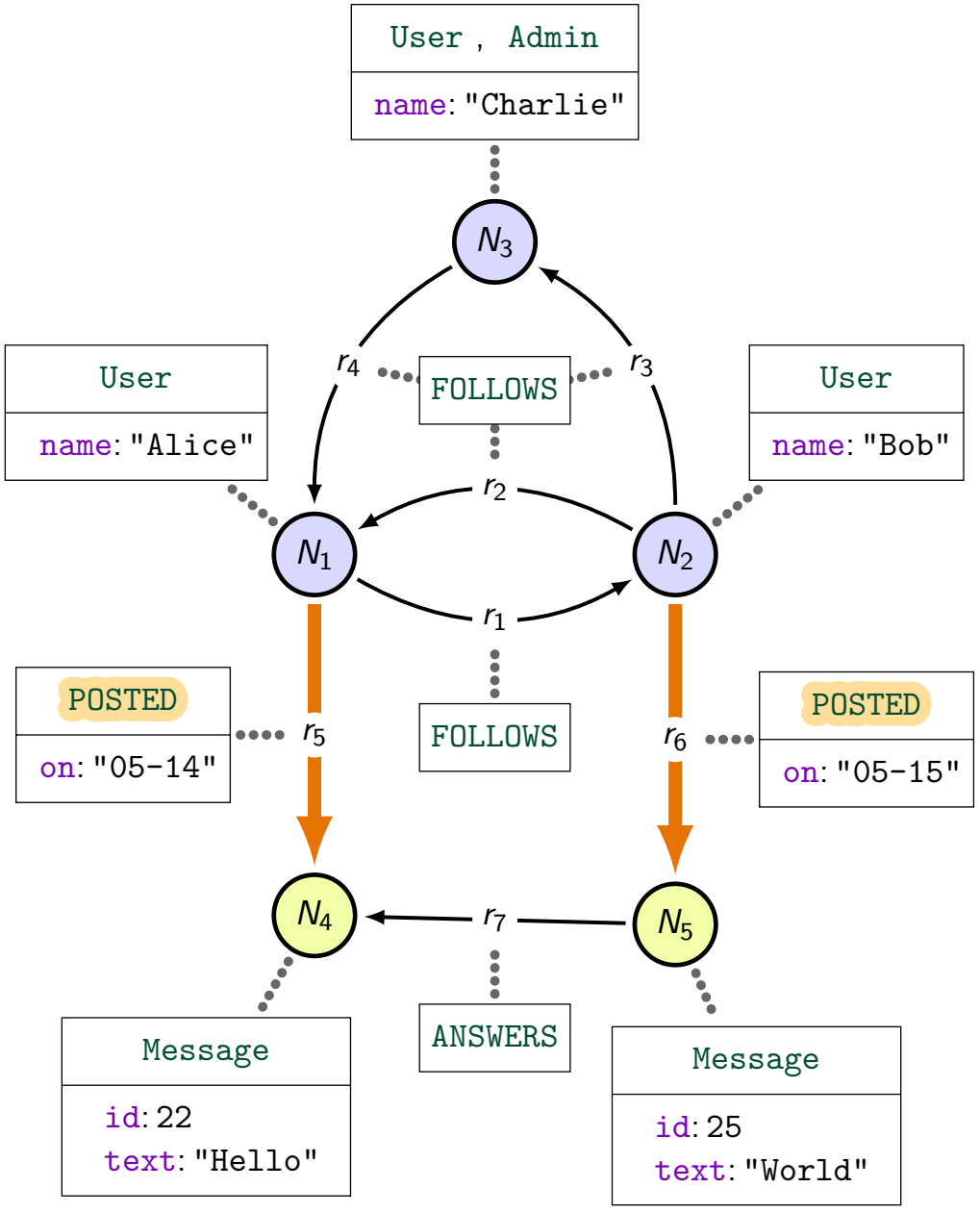
Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
```

Result:

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Matching relations (1)



Query:

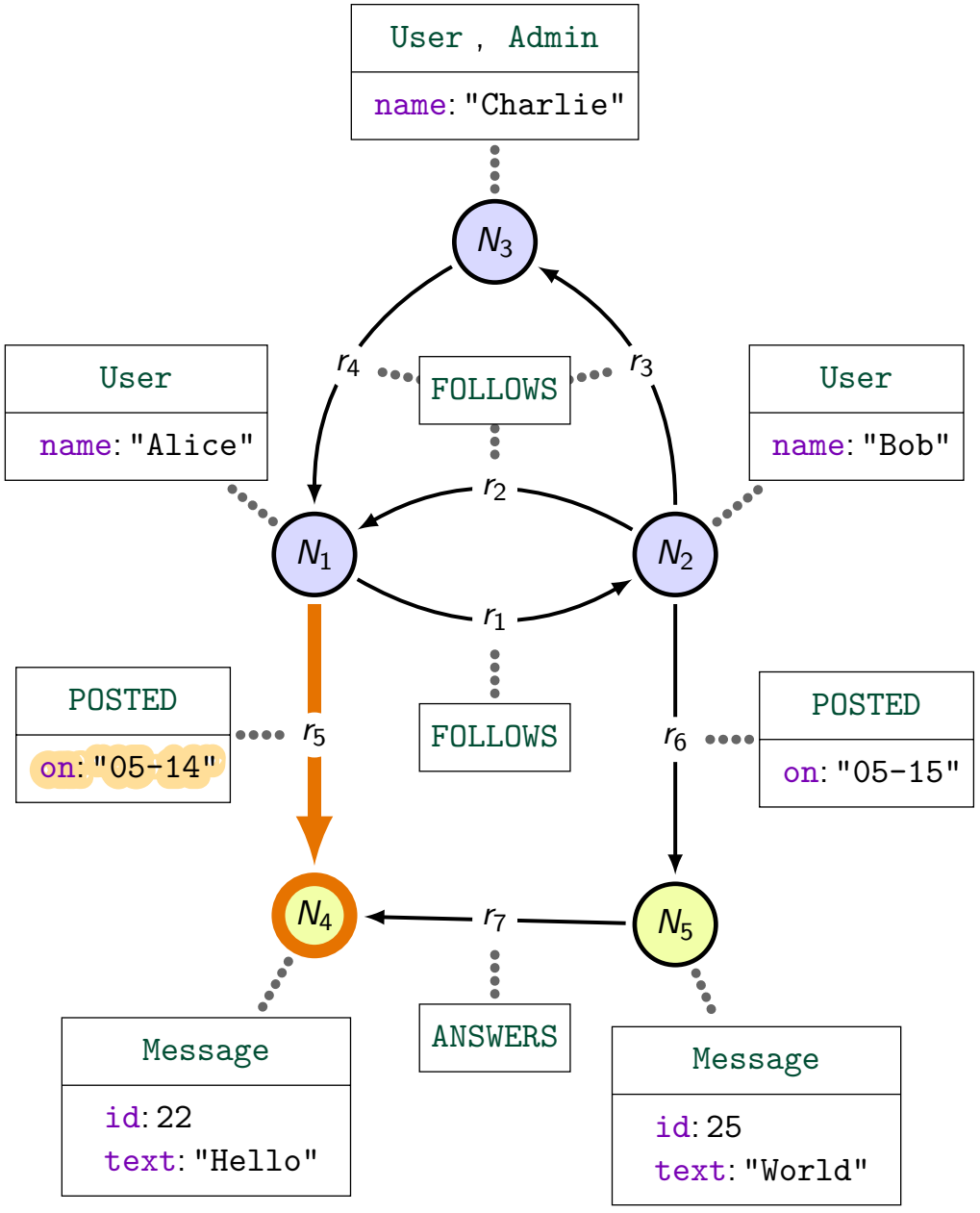
```
MATCH (u1)-[p1:POSTED]->(m1)
```

Result:

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

- Edge patterns are between `[ ]`
- Direction is indicated by `-[ ]->` or `<-[ ]-` or `-[ ]-`
- Edge type is prefixed by `:`

Matching relations (2)



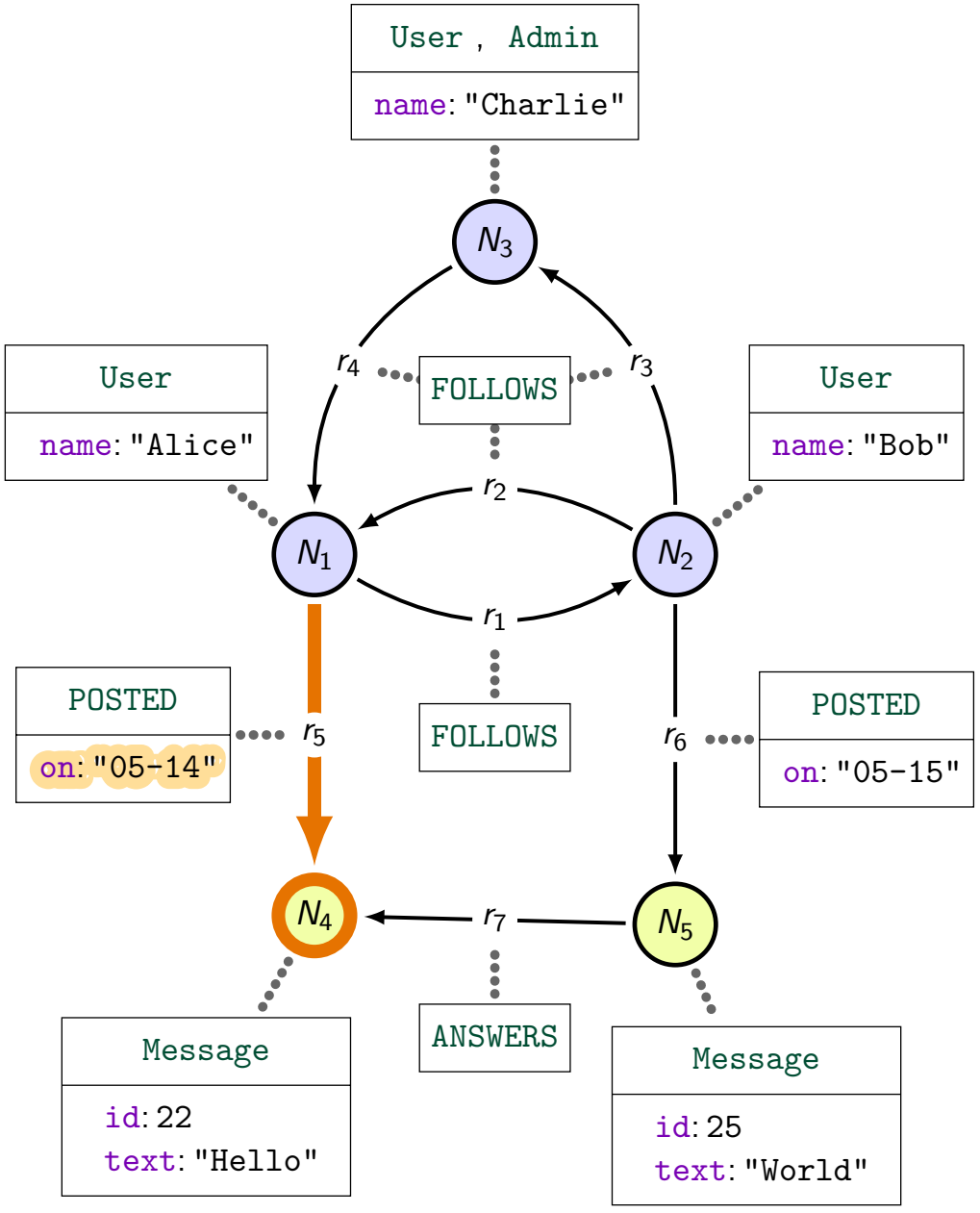
Query:

```
MATCH (m1)<-[p1{on:"05-14"}]-()
```

Result:

m1	p1
N_4	r_5

Matching relations (2)



Query:

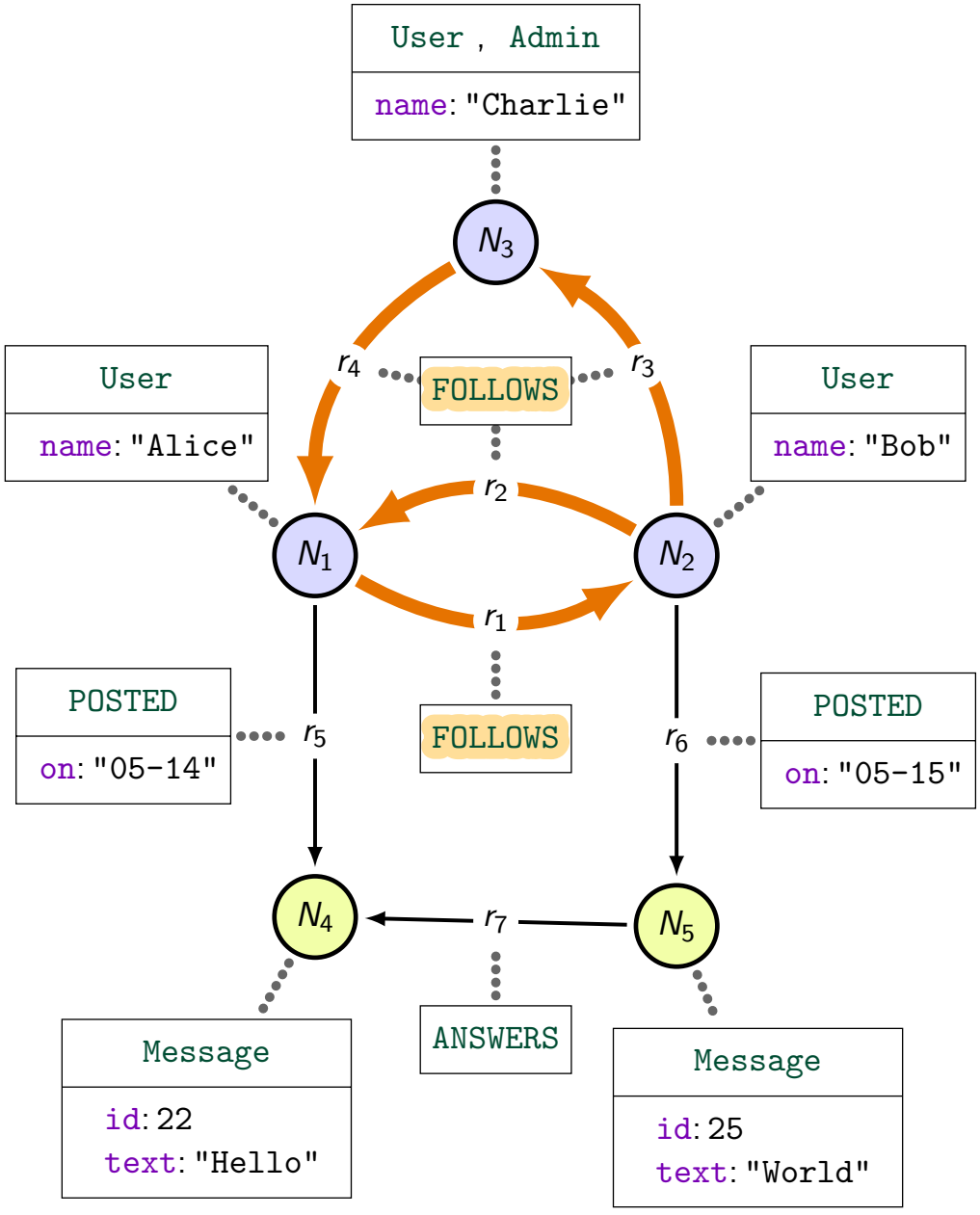
```
MATCH (m1)<-[p1{on:"05-14"}]-()
```

Result:

m1	p1
N_4	r_5

- Backward edge: `<-[ ]-`
- Properties are between `{ }`
- All variables are optional

Matching relations (3)

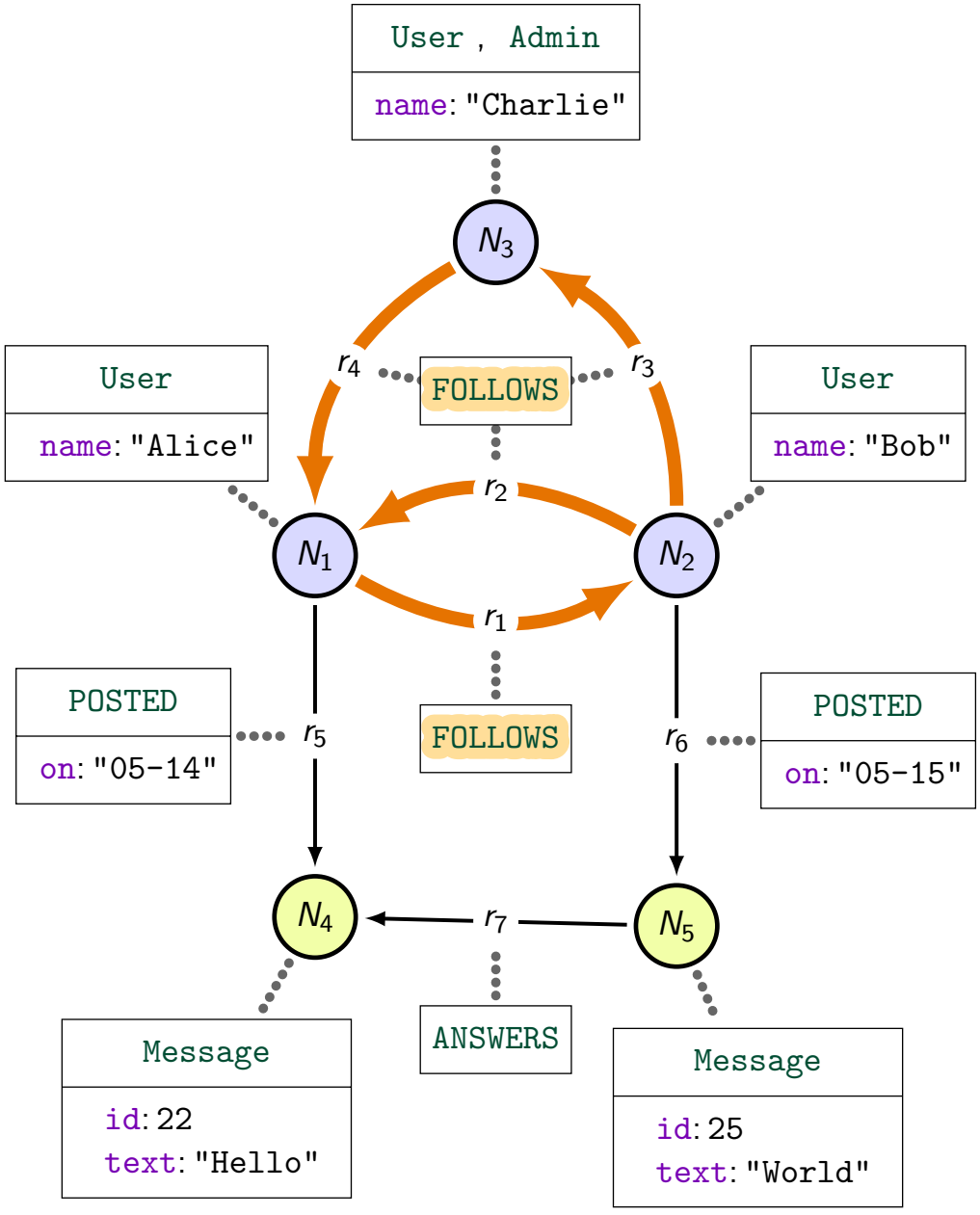


Query:

```
MATCH (u1)-[:FOLLOWS]->()
```

Result:

Matching relations (3)



Query:

```
MATCH (u1)-[:FOLLOWS]->()
```

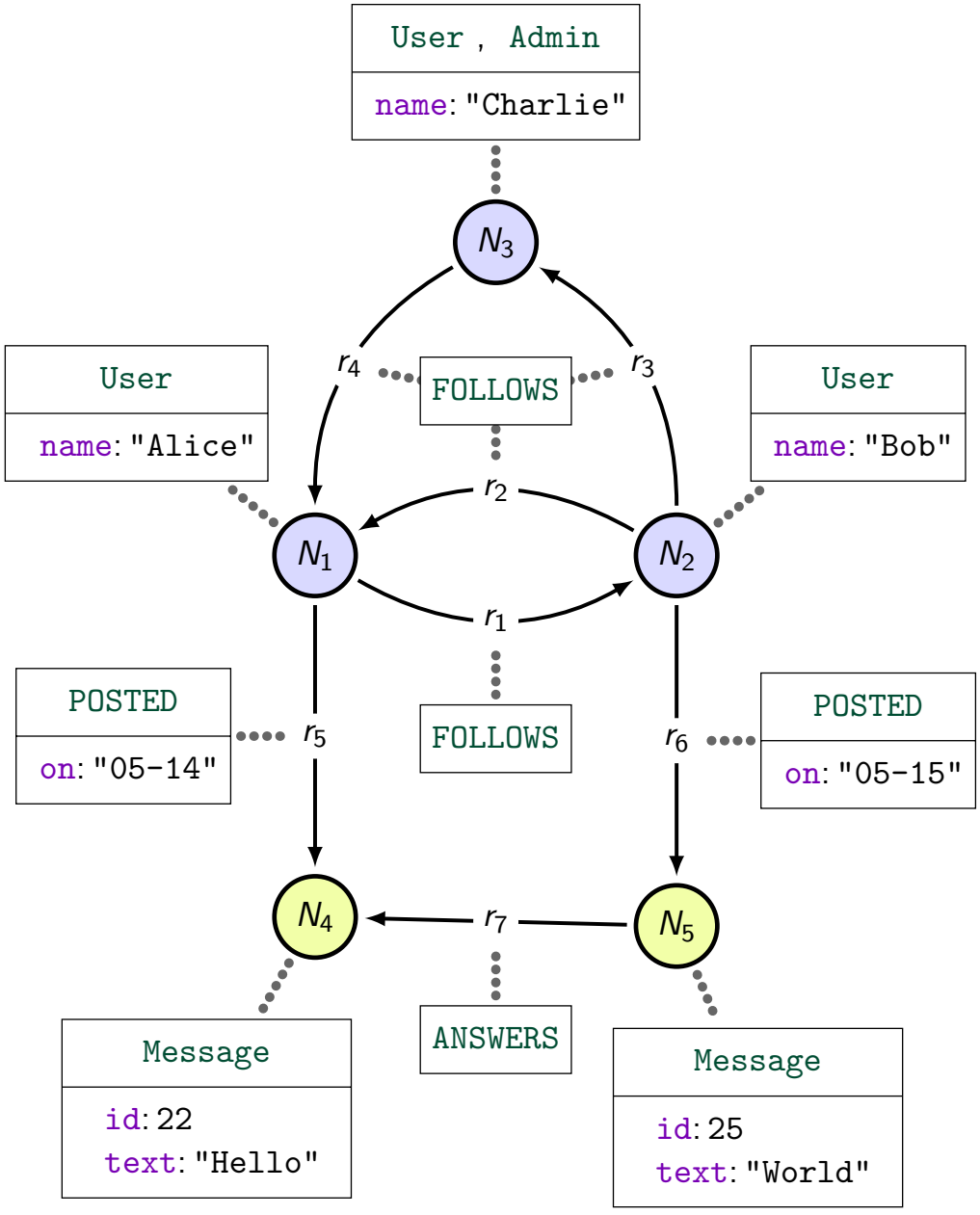
Result:

u1
N_1
N_2
N_2
N_3

Cypher has bag semantics:

N_2 has two outgoing `follows` relations $\Rightarrow$ two lines N_2

Matching chained relations



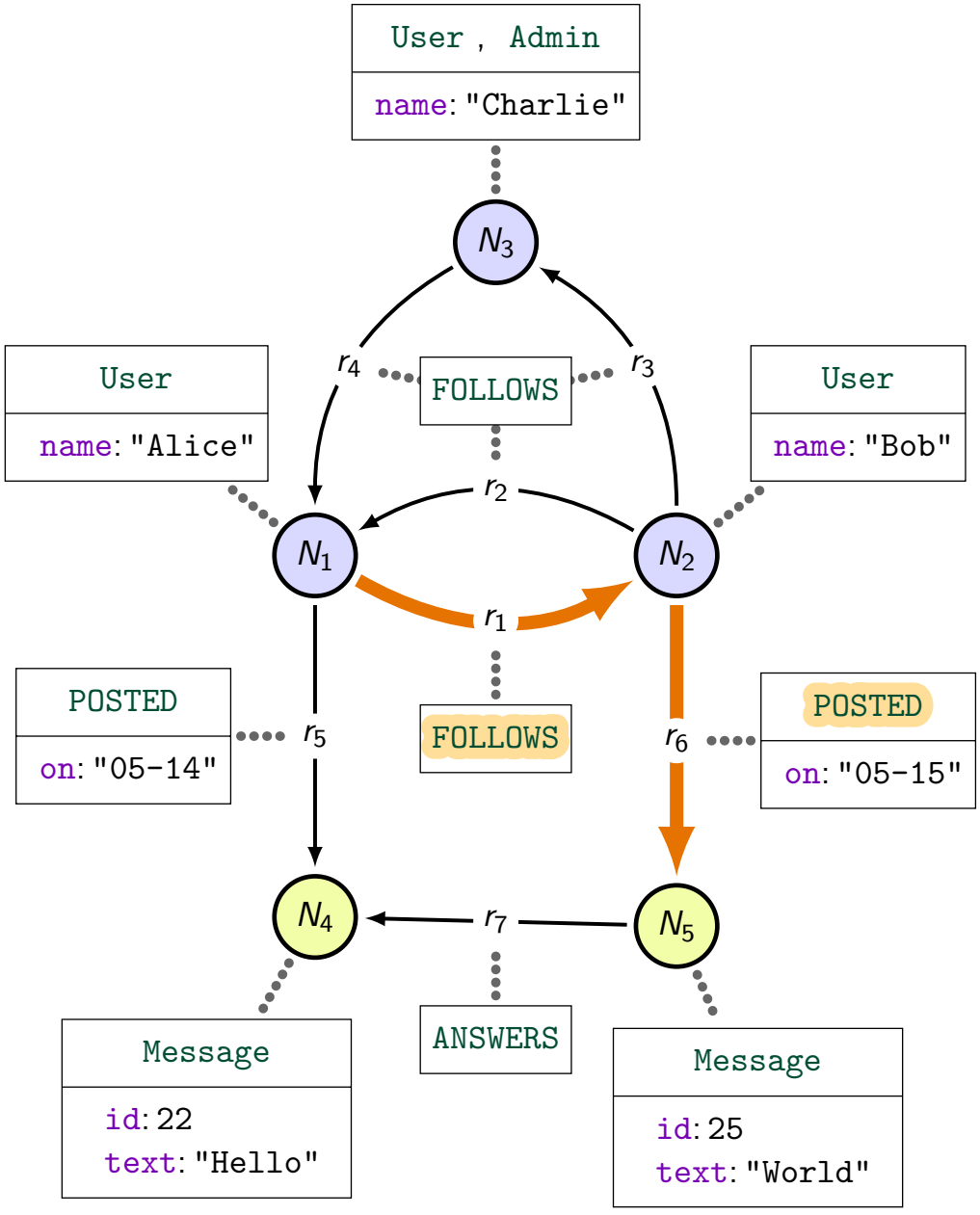
Query:

```
MATCH (u1)-[:FOLLOWS]->()
                -[:POSTED]->(m1)
```

Result:

u1	m1
N_1	N_5
N_2	N_4
N_3	N_4

Matching chained relations



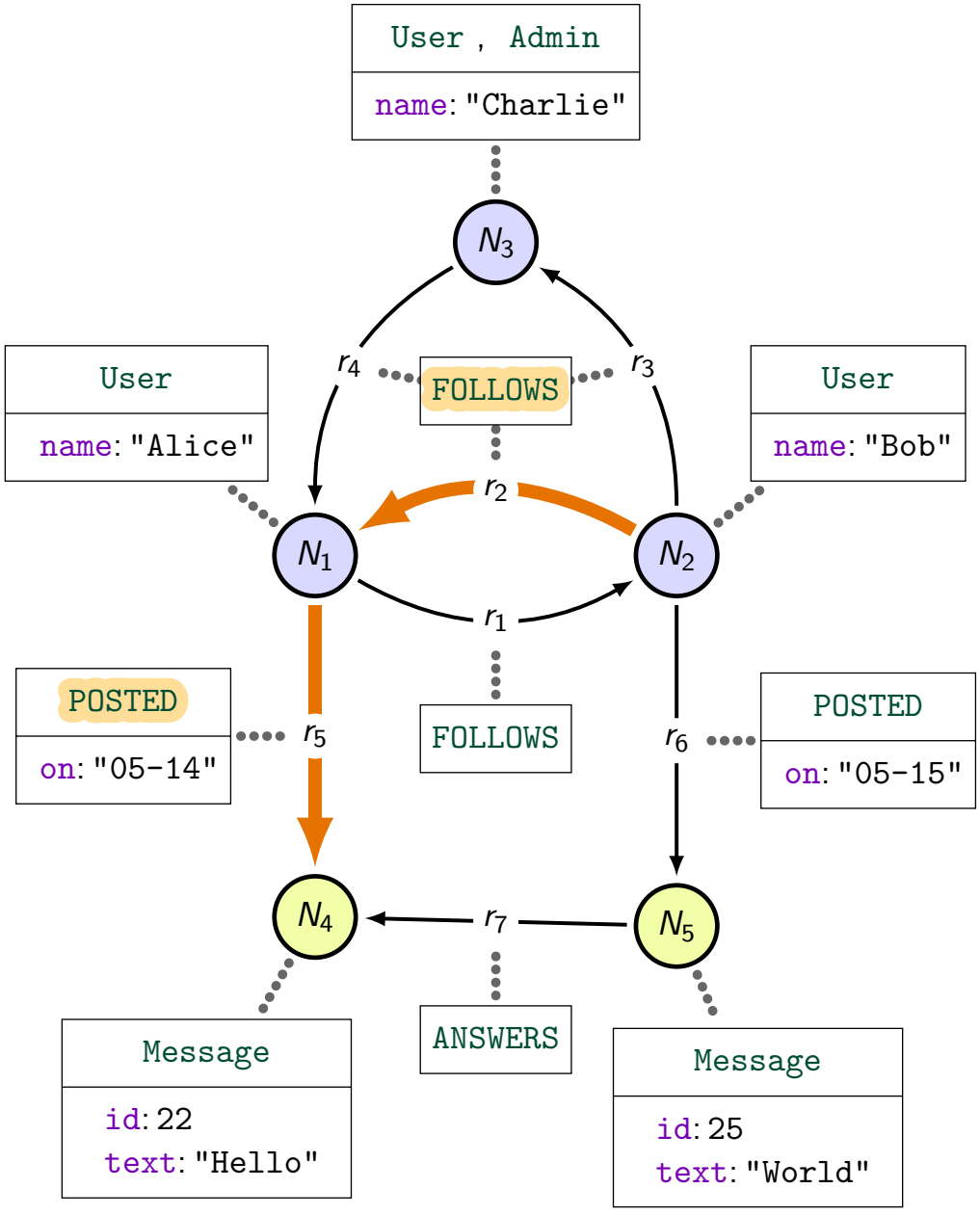
Query:

```
MATCH (u1)-[:FOLLOWS]->()
                -[:POSTED]->(m1)
```

Result:

u1	m1
N_1	N_5
N_2	N_4
N_3	N_4

Matching chained relations



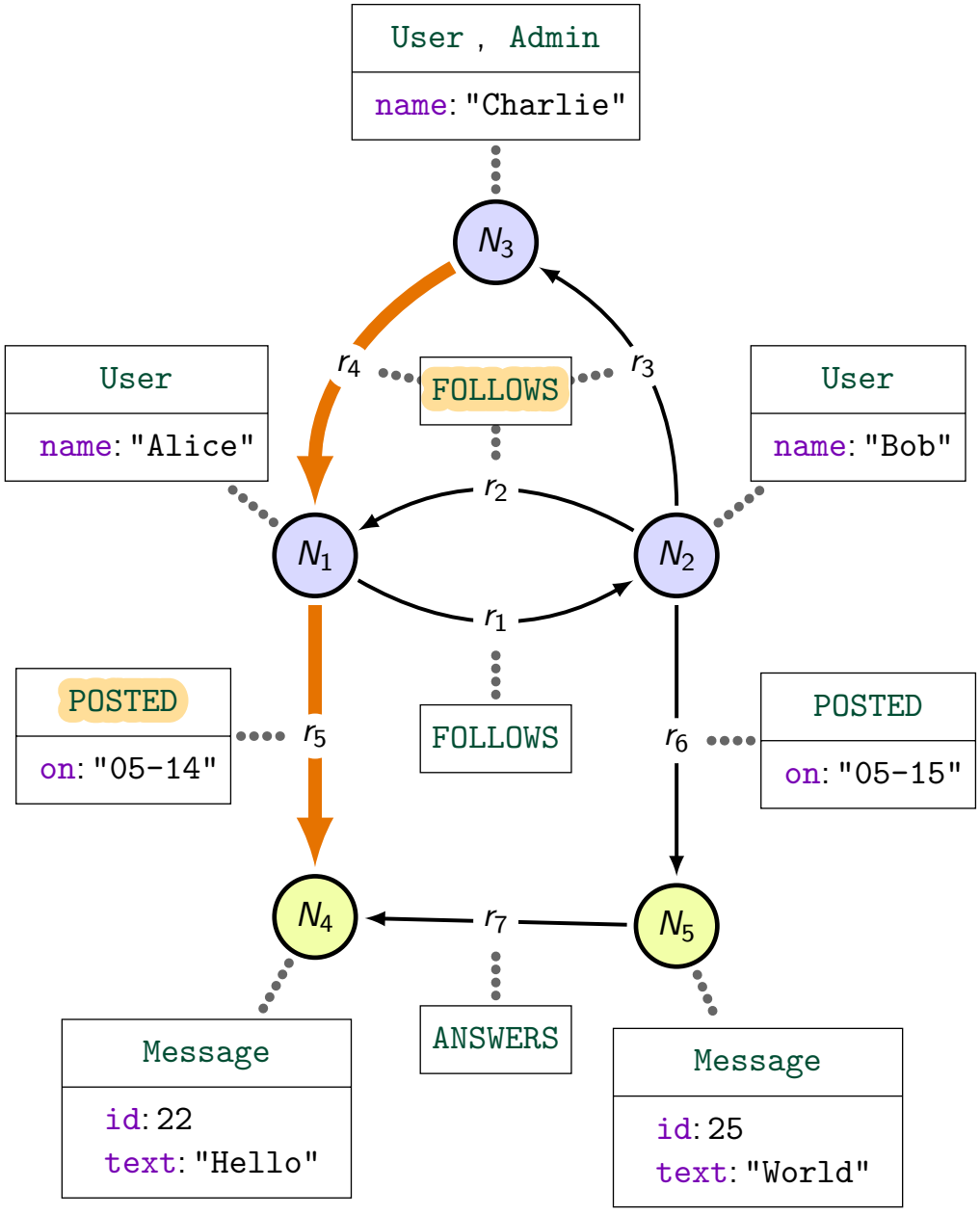
Query:

```
MATCH (u1)-[:FOLLOWS]->()
                -[:POSTED]->(m1)
```

Result:

u1	m1
N_1	N_5
N_2	N_4
N_3	N_4

Matching chained relations



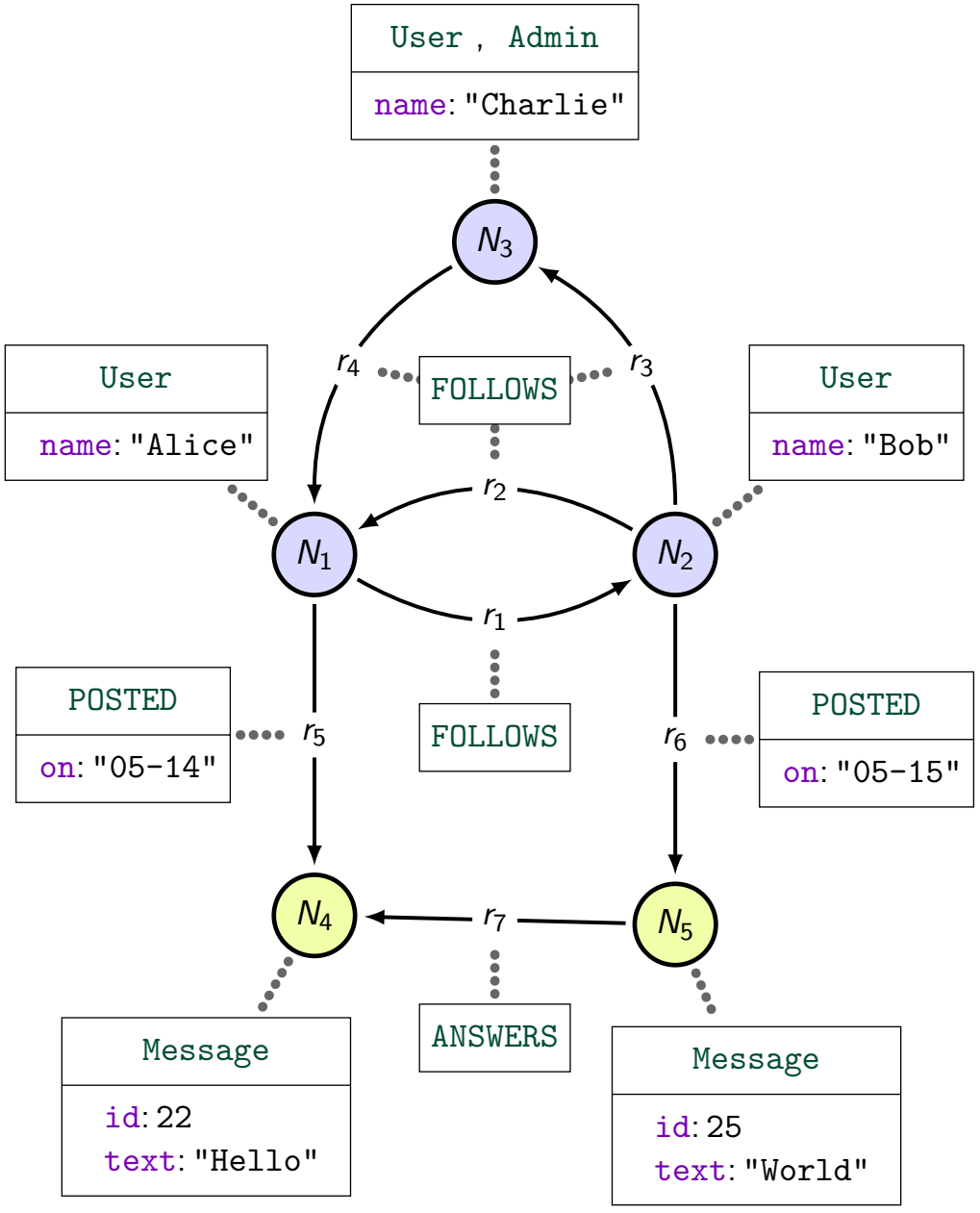
Query:

```
MATCH (u1)-[:FOLLOWS]->()
                -[:POSTED]->(m1)
```

Result:

u1	m1
N_1	N_5
N_2	N_4
N_3	N_4

More chaining



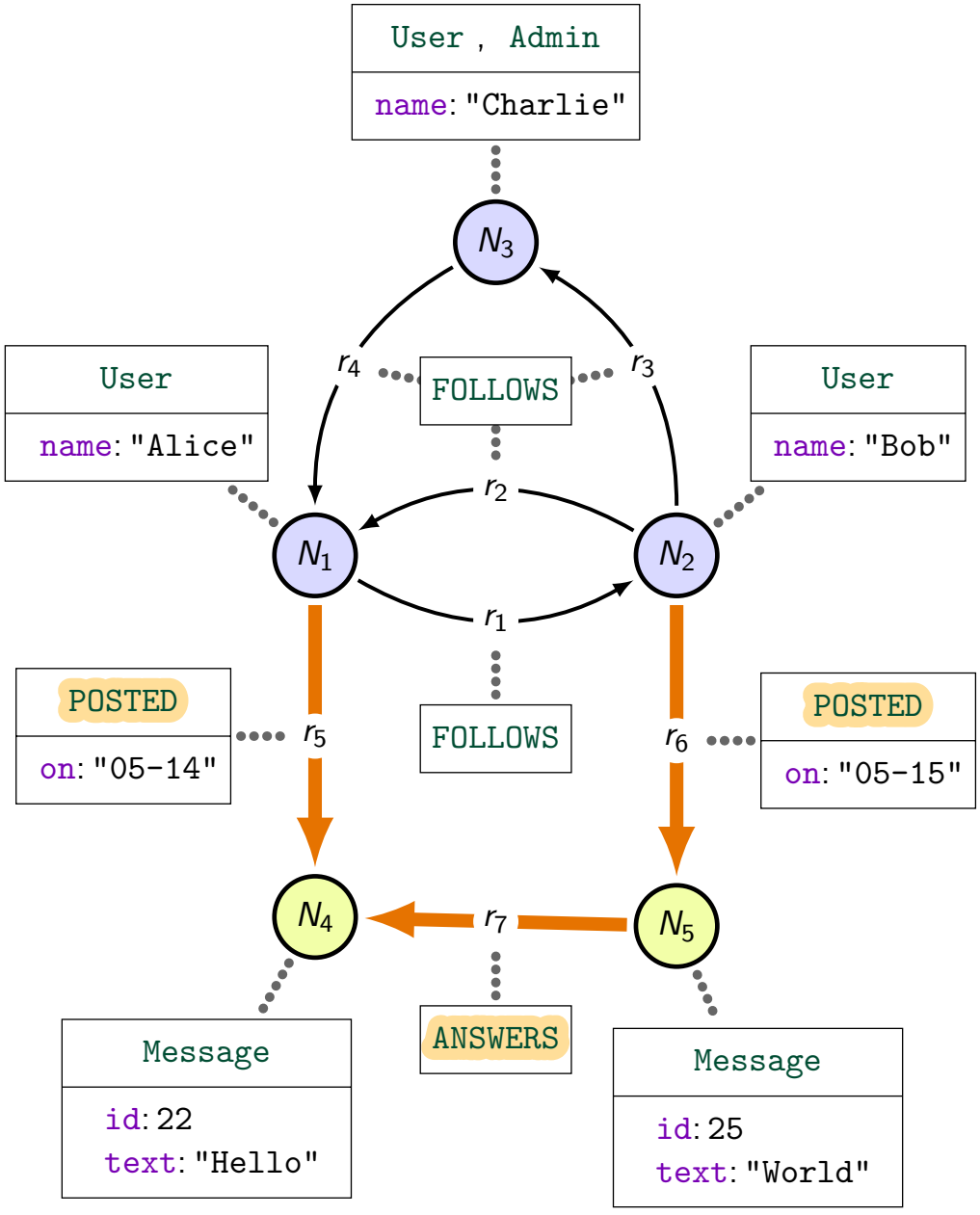
Query:

```
MATCH (u1)-[:POSTED]->()
          <-[:ANSWERS]-(m2)
          <-[:POSTED]-(u2)
```

Result:

u1	m2	u2
N_1	N_5	N_2

More chaining



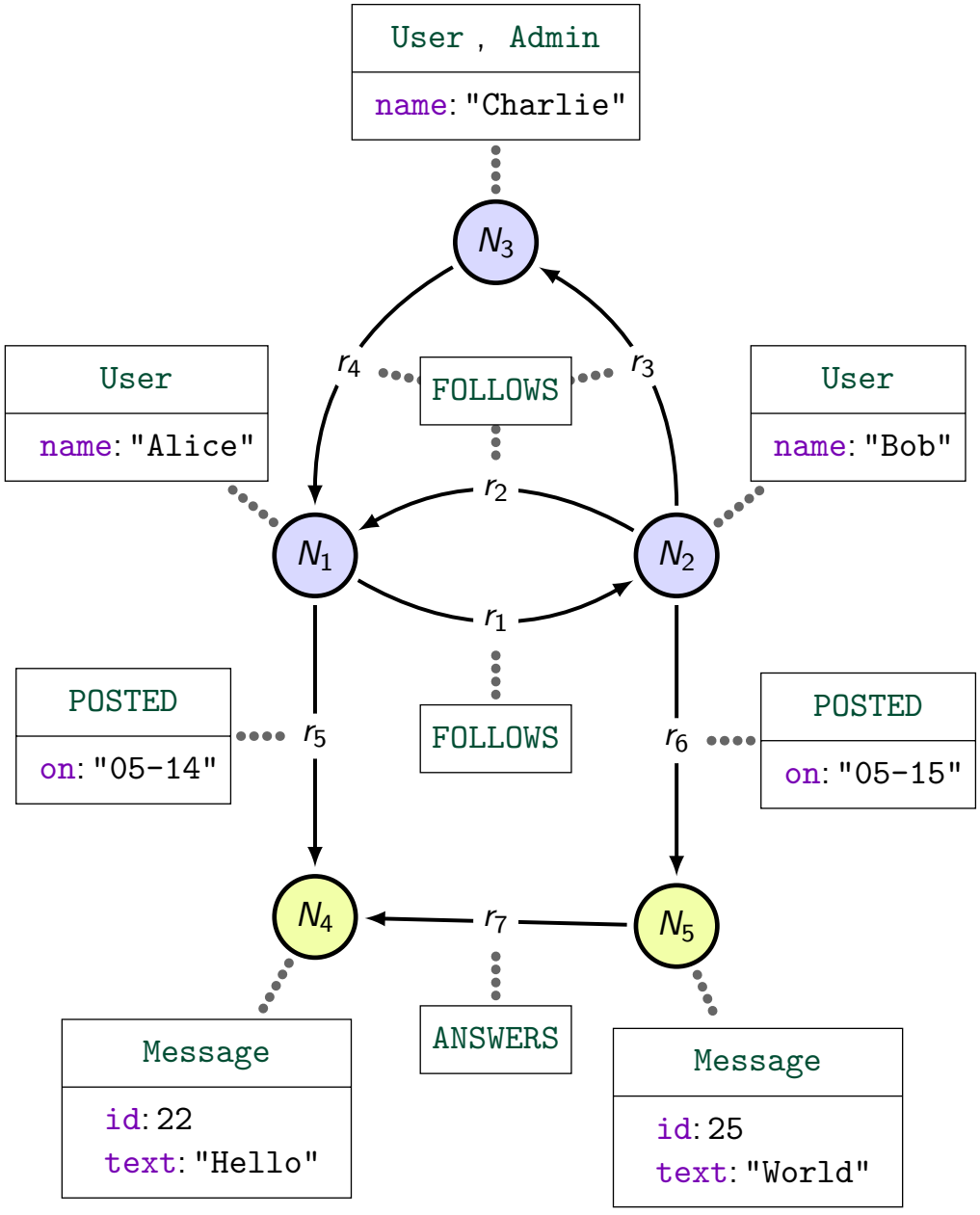
Query:

```
MATCH (u1)-[:POSTED]->()
          <-[:ANSWERS]-(m2)
          <-[:POSTED]-(u2)
```

Result:

u1	m2	u2
N_1	N_5	N_2

Any-directed relation pattern



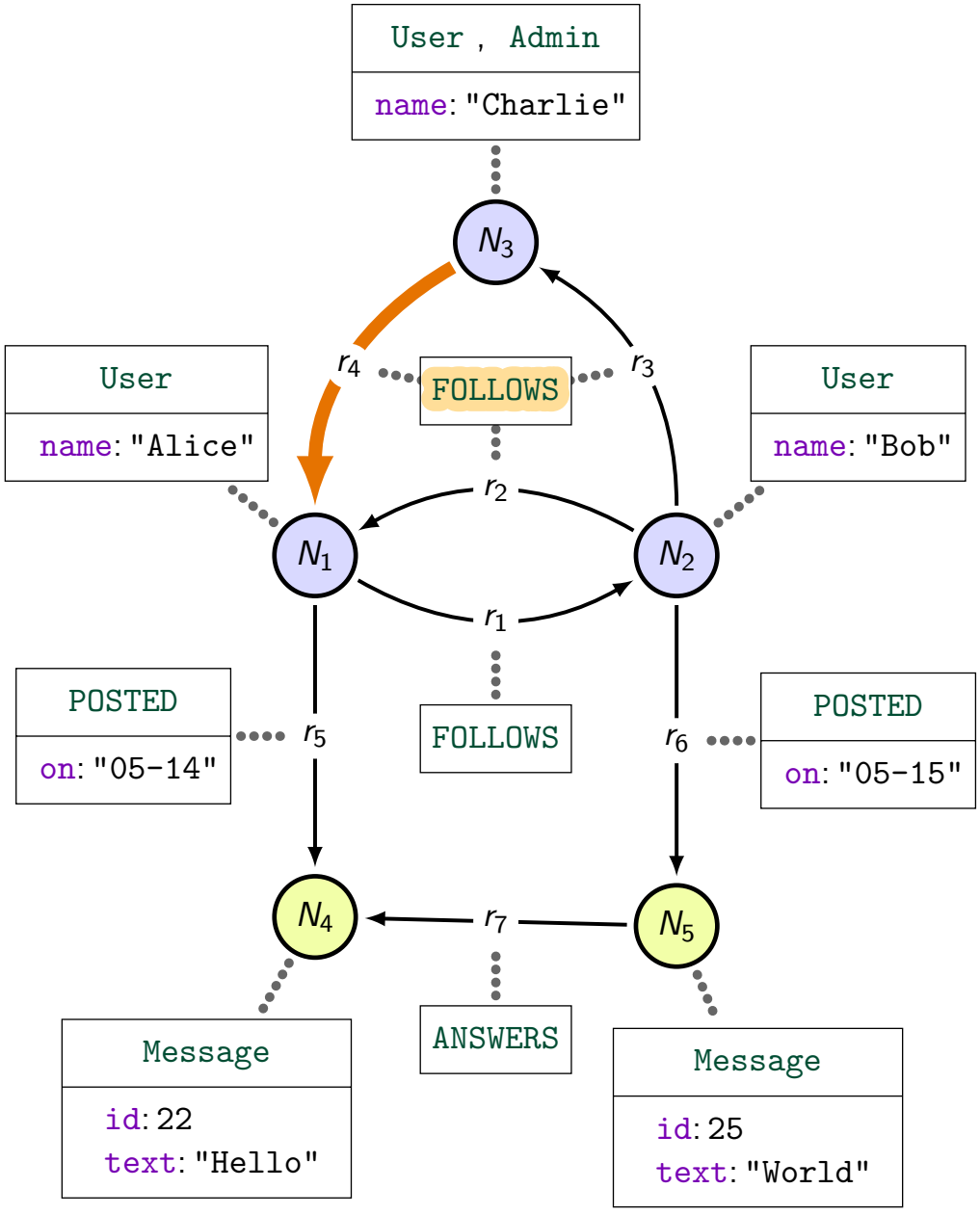
Query:

```
MATCH (u1:Admin)
                -[:FOLLOWS]-(u2)
```

Result:

u1	u2
N_3	N_1
N_3	N_2

Any-directed relation pattern



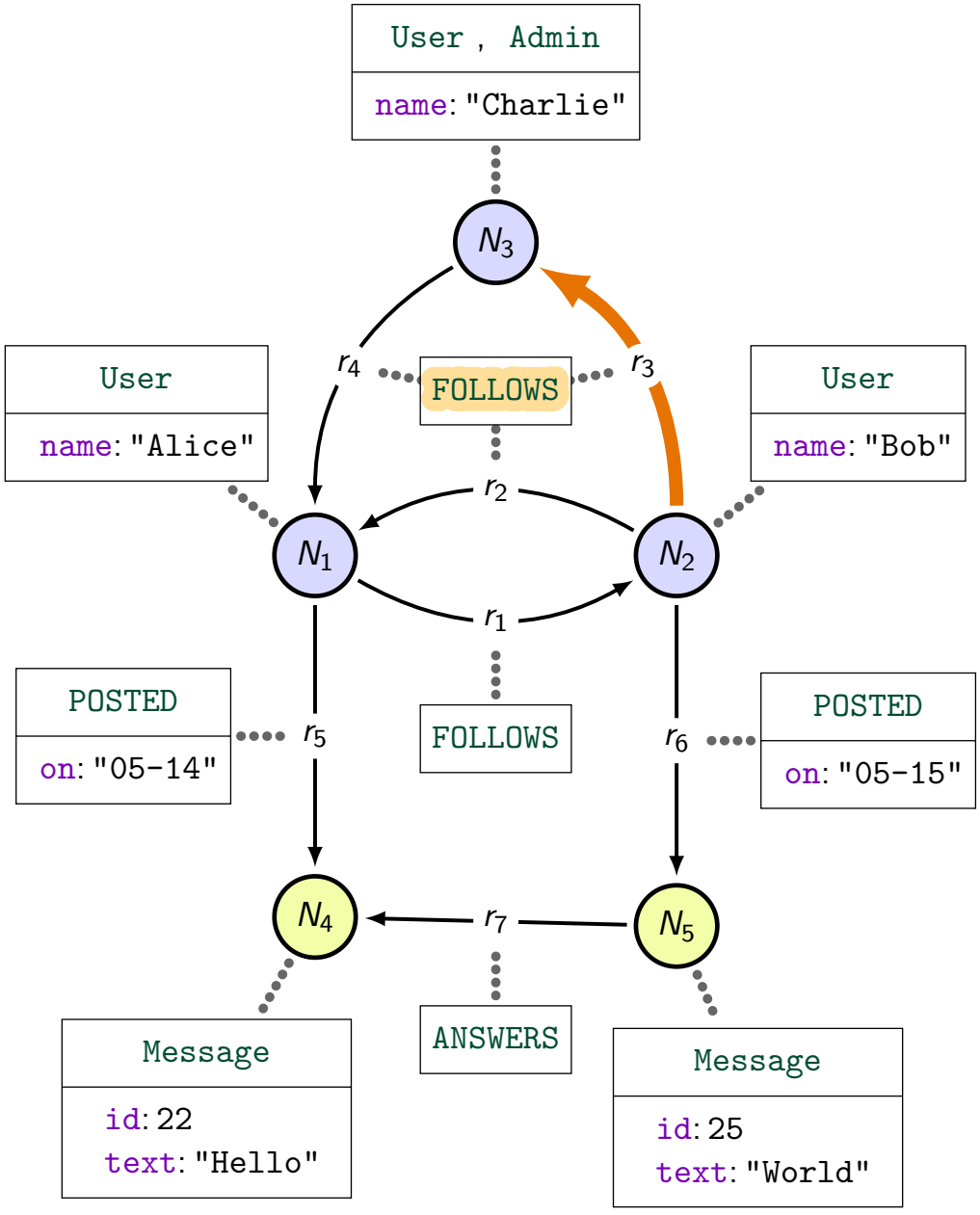
Query:

```
MATCH (u1:Admin)
                -[:FOLLOWS]-(u2)
```

Result:

u1	u2
N_3	N_1
N_3	N_2

Any-directed relation pattern



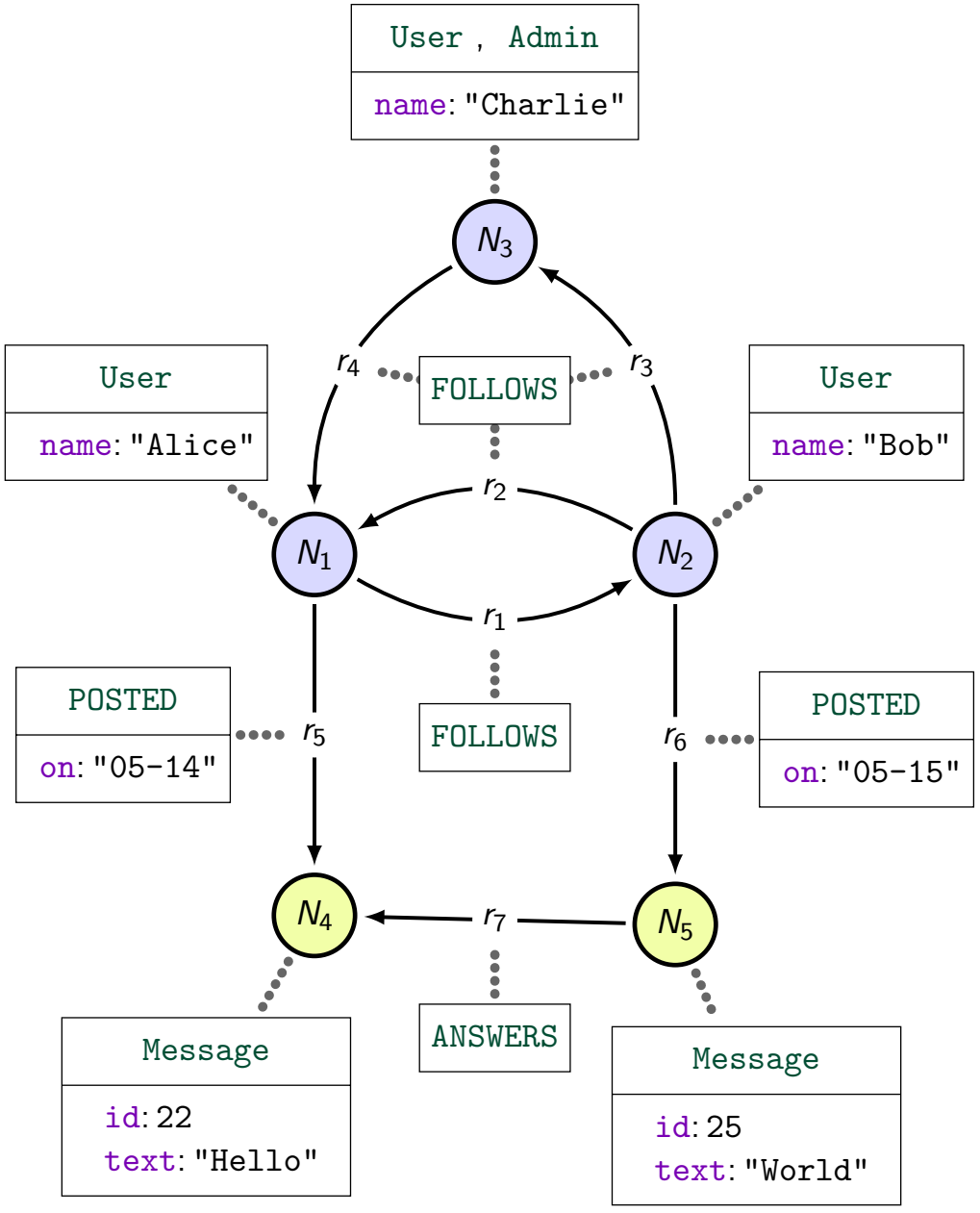
Query:

```
MATCH (u1:Admin)
                -[:FOLLOWS]-(u2)
```

Result:

u1	u2
N_3	N_1
N_3	N_2

Implicit equijoin on variables



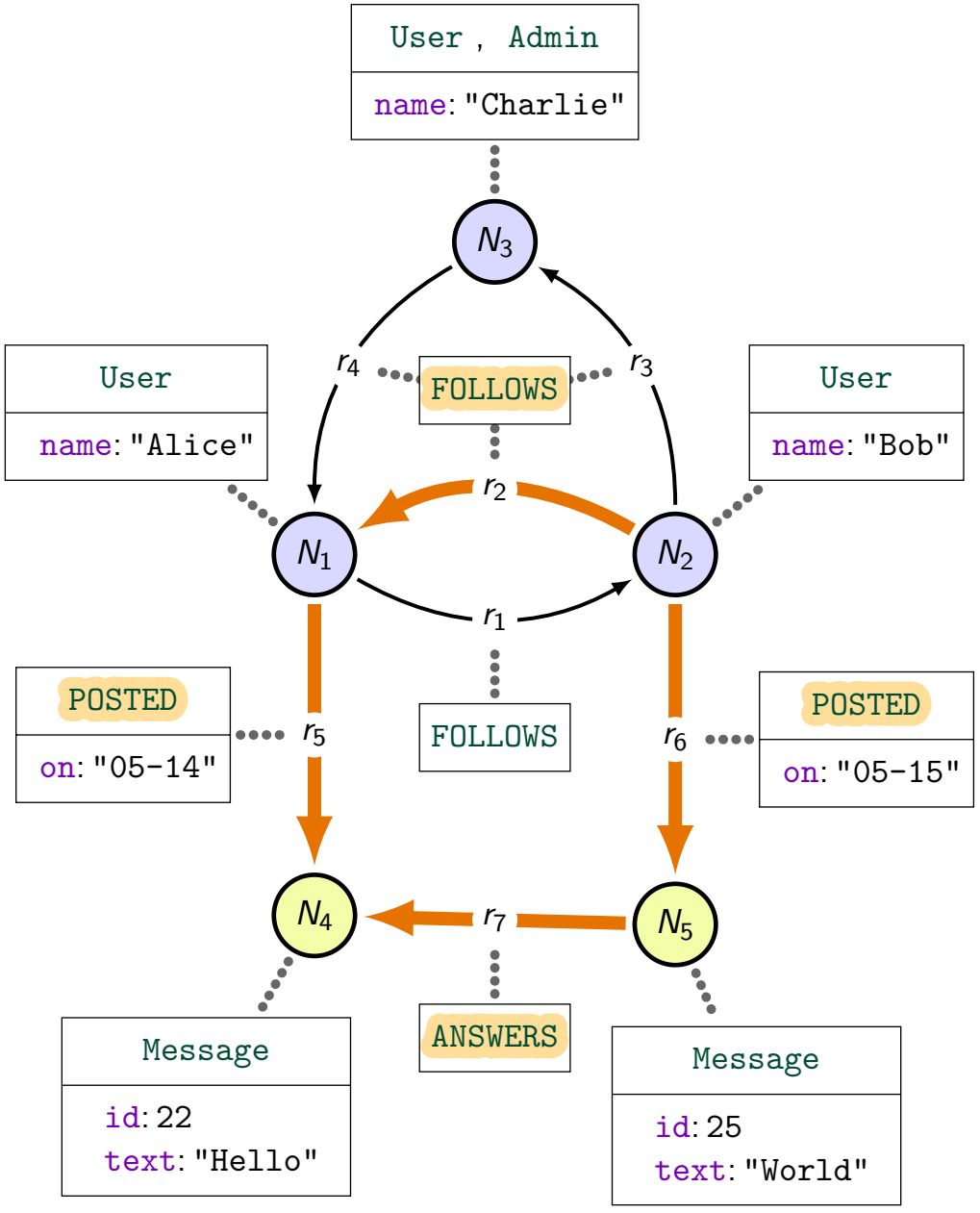
Query:

```
MATCH (u1)-[:POSTED]->()
          <-[:ANSWERS]-(m2)
          <-[:POSTED]-(u2)
          -[:FOLLOWS]->(u1)
```

Result:

u1	m2	u2
N_1	N_5	N_2

Implicit equijoin on variables



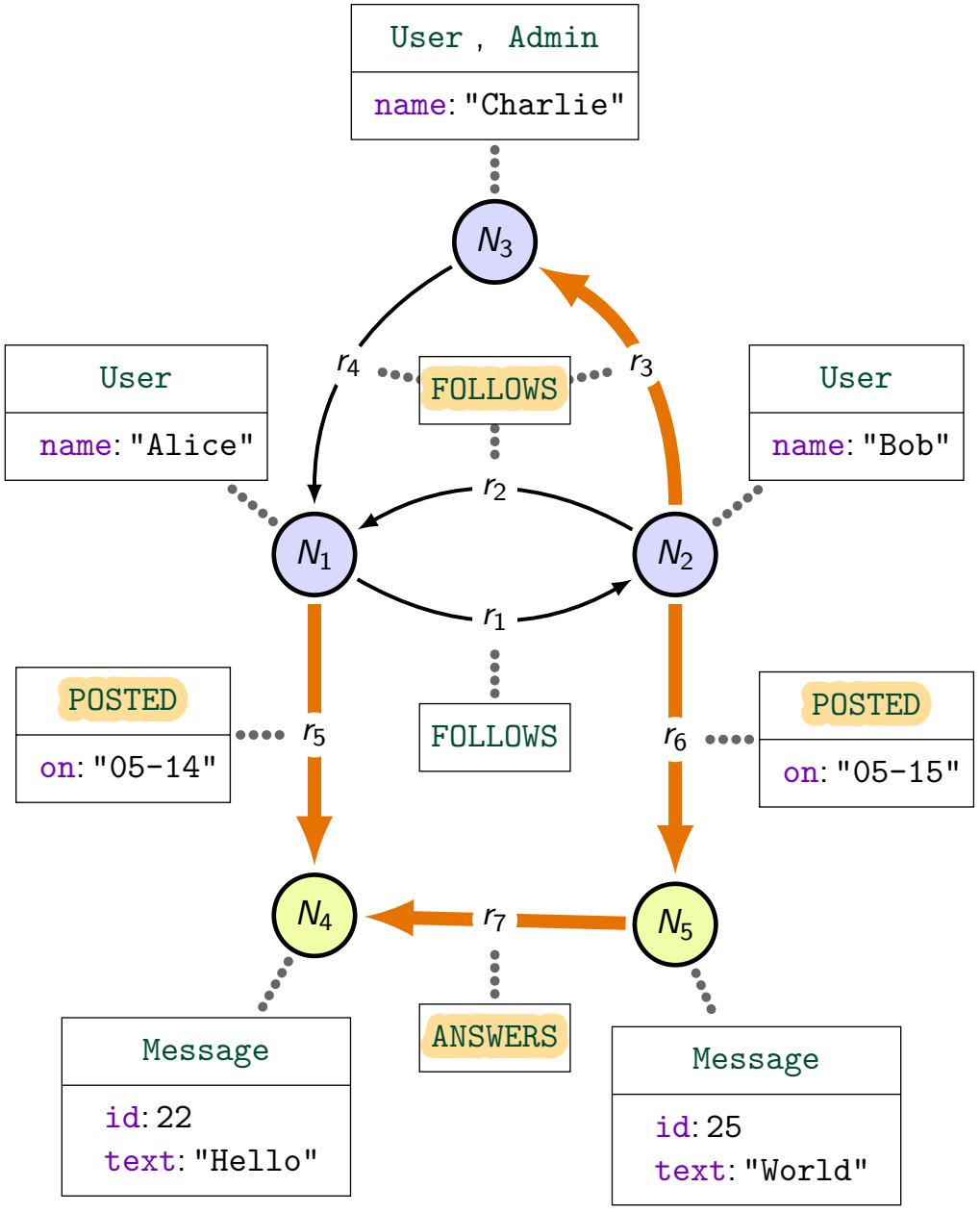
Query:

```
MATCH (u1)-[:POSTED]->()
            <-[:ANSWERS]-(m2)
            <-[:POSTED]-(u2)
            -[:FOLLOWS]->(u1)
```

Result:

u1	m2	u2
N_1	N_5	N_2

Implicit equijoin on variables



Query:

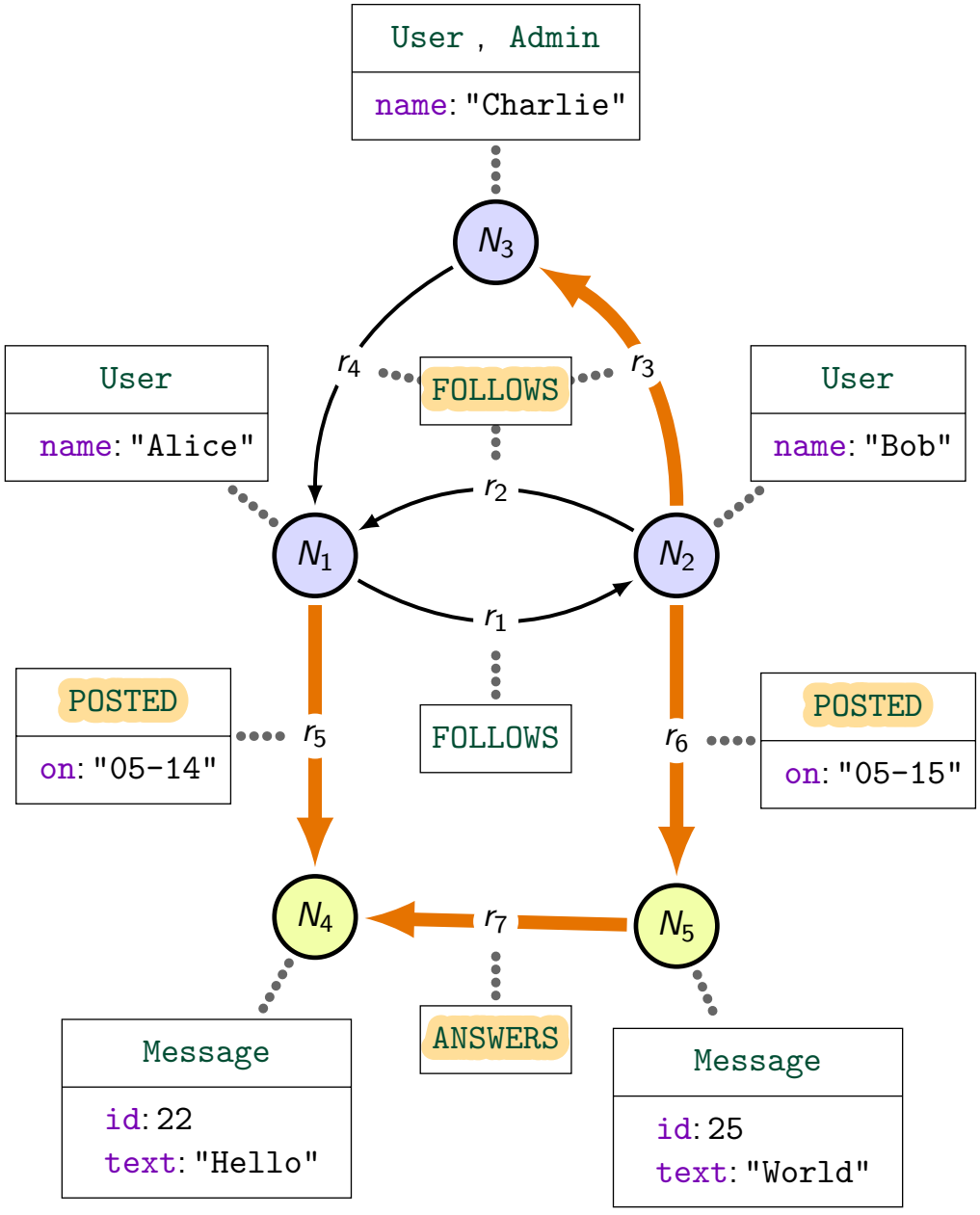
```
MATCH (u1)-[:POSTED]->()
      <-[:ANSWERS]-(m2)
      <-[:POSTED]-(u2)
      -[:FOLLOWS]->(u1)
```

Result:

u1	m2	u2
N_1	N_5	N_2

Variable reuse $\Longrightarrow$ equality

Implicit equijoin on variables



Query:

```
MATCH (u1)-[:POSTED]->()
      <-[:ANSWERS]-(m2)
      <-[:POSTED]-(u2)
      -[:FOLLOWS]->(u1)
```

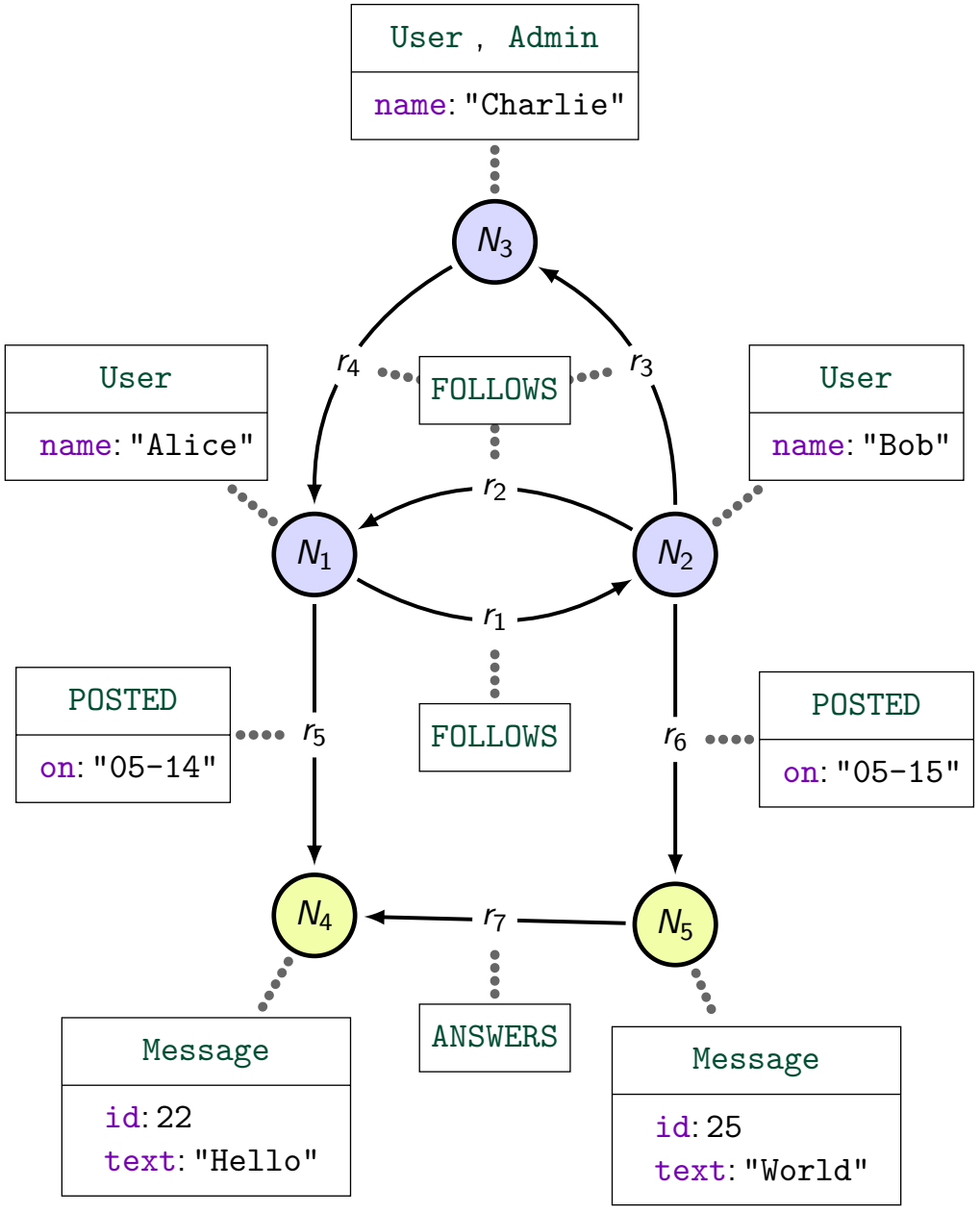
Result:

u1	m2	u2
N_1	N_5	N_2

Variable reuse $\Longrightarrow$ equality

The orange path is invalid: two different nodes for u1.

Matching paths (1)



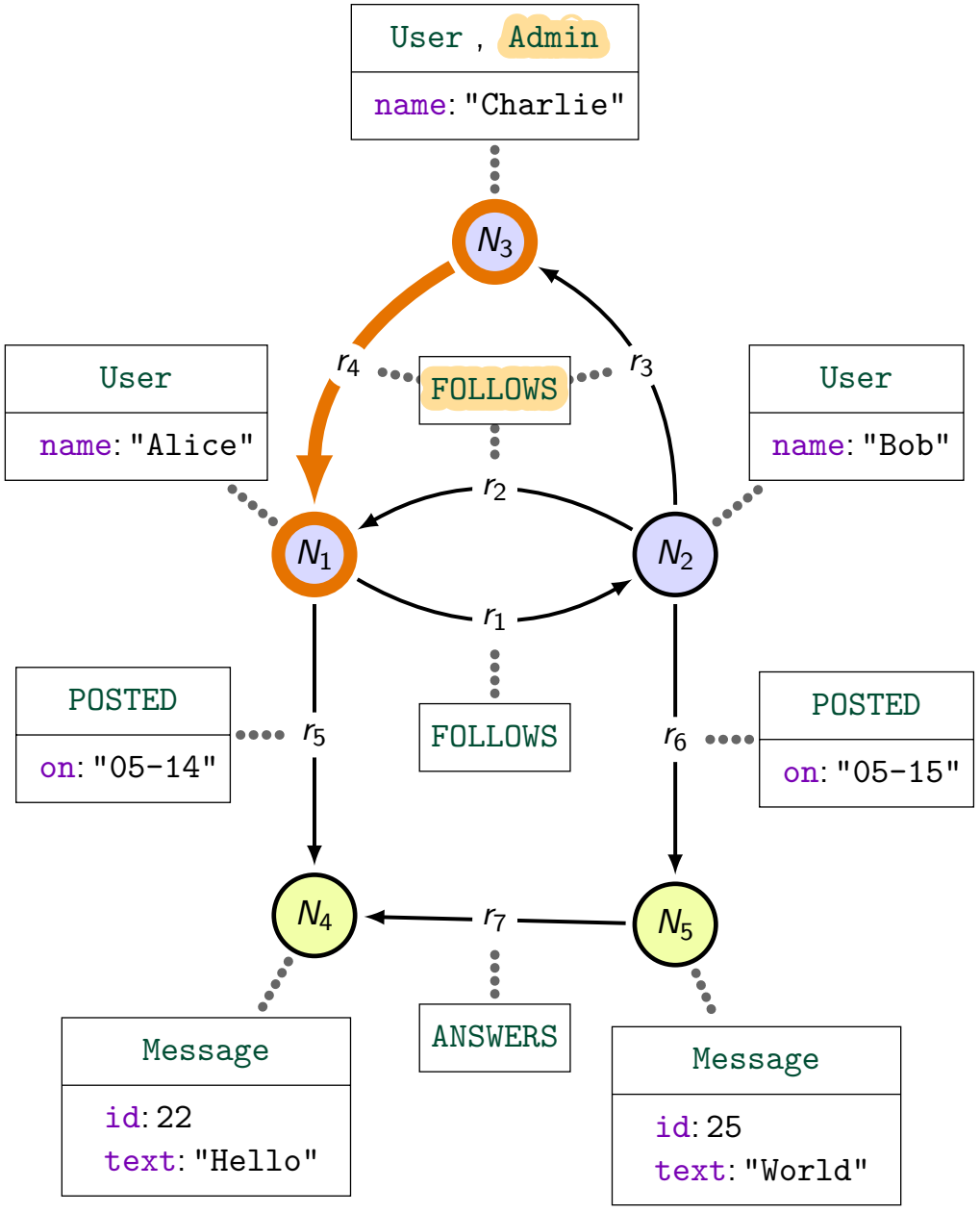
Query:

```
MATCH (u1:Admin)
          -[l1:FOLLOWS*]->(m1)
```

Result:

u1	l1	m1
N_3	$[r_4]$	N_1
N_3	$[r_4, r_1]$	N_2
N_3	$[r_4, r_1, r_2]$	N_1
N_3	$[r_4, r_1, r_3]$	N_3

Matching paths (1)



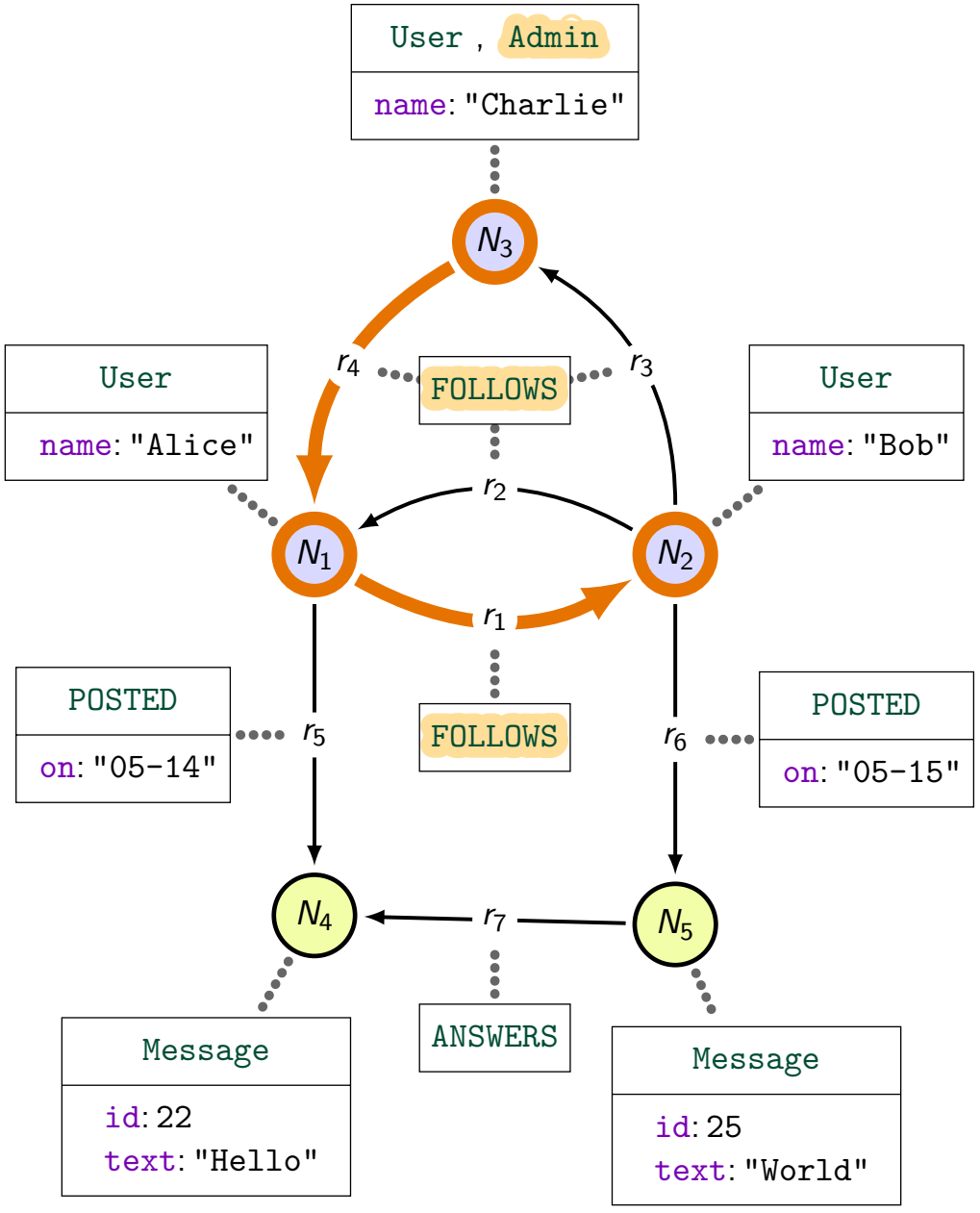
Query:

```
MATCH (u1:Admin)
          -[l1:FOLLOWS*]->(m1)
```

Result:

u1	l1	m1
N_3	$[r_4]$	N_1
N_3	$[r_4, r_1]$	N_2
N_3	$[r_4, r_1, r_2]$	N_1
N_3	$[r_4, r_1, r_3]$	N_3

Matching paths (1)



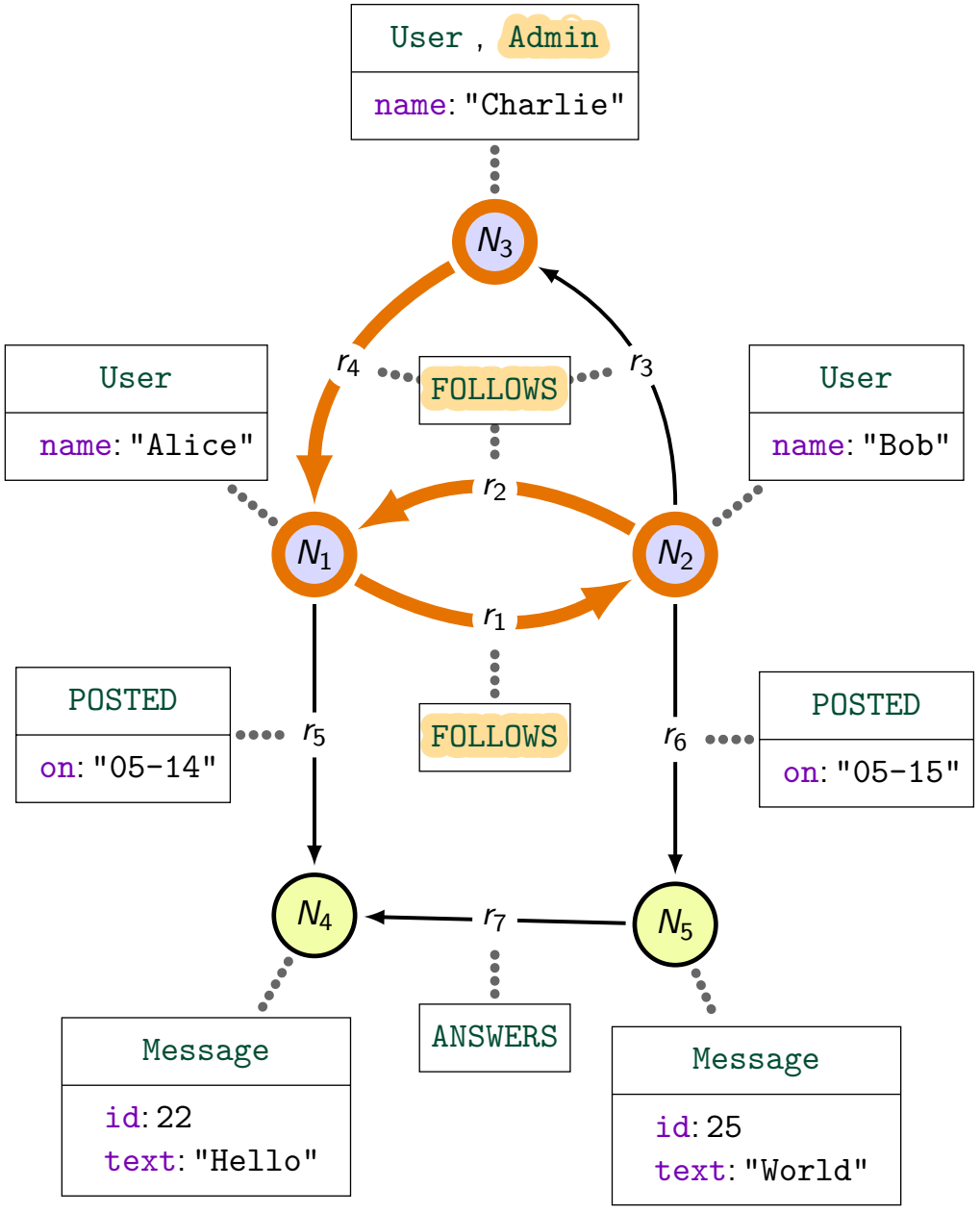
Query:

```
MATCH (u1:Admin)
          -[l1:FOLLOWS*]->(m1)
```

Result:

u1	l1	m1
N_3	$[r_4]$	N_1
N_3	$[r_4, r_1]$	N_2
N_3	$[r_4, r_1, r_2]$	N_1
N_3	$[r_4, r_1, r_3]$	N_3

Matching paths (1)



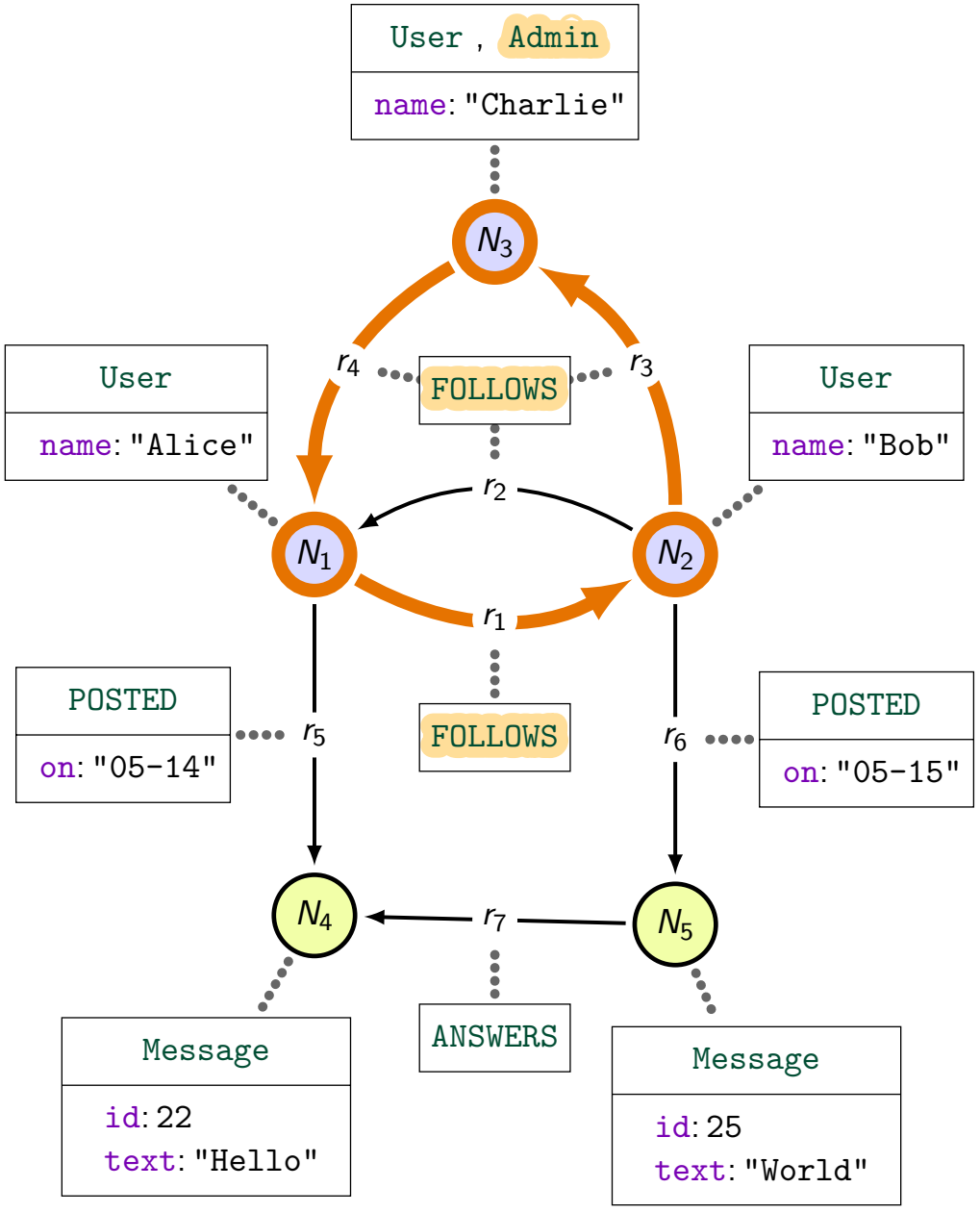
Query:

```
MATCH (u1:Admin)
          -[l1:FOLLOWS*]->(m1)
```

Result:

u1	l1	m1
N_3	$[r_4]$	N_1
N_3	$[r_4, r_1]$	N_2
N_3	$[r_4, r_1, r_2]$	N_1
N_3	$[r_4, r_1, r_3]$	N_3

Matching paths (1)



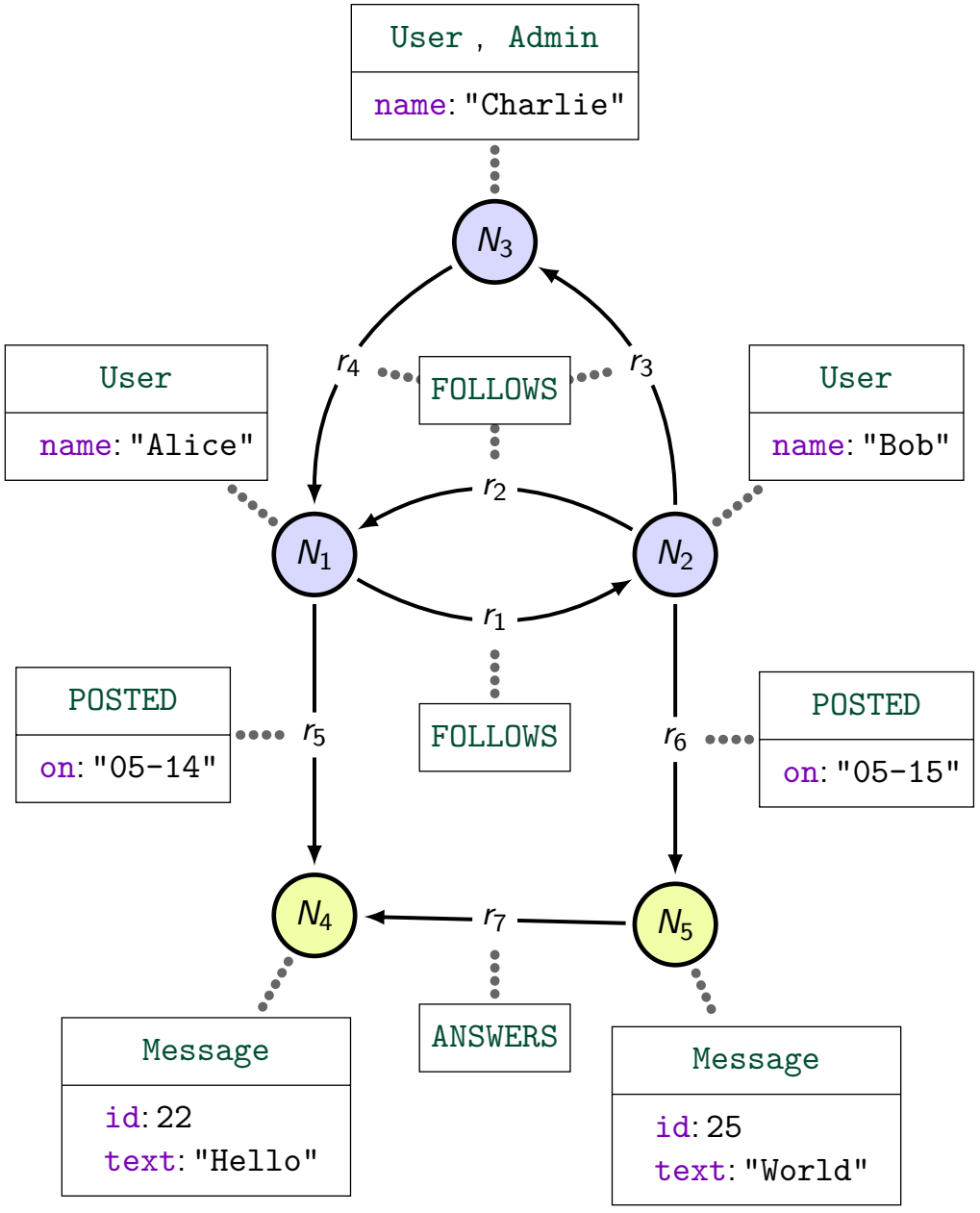
Query:

```
MATCH (u1:Admin)
          -[l1:FOLLOWS*]->(m1)
```

Result:

u1	l1	m1
N_3	$[r_4]$	N_1
N_3	$[r_4, r_1]$	N_2
N_3	$[r_4, r_1, r_2]$	N_1
N_3	$[r_4, r_1, r_3]$	N_3

Matching paths (1)



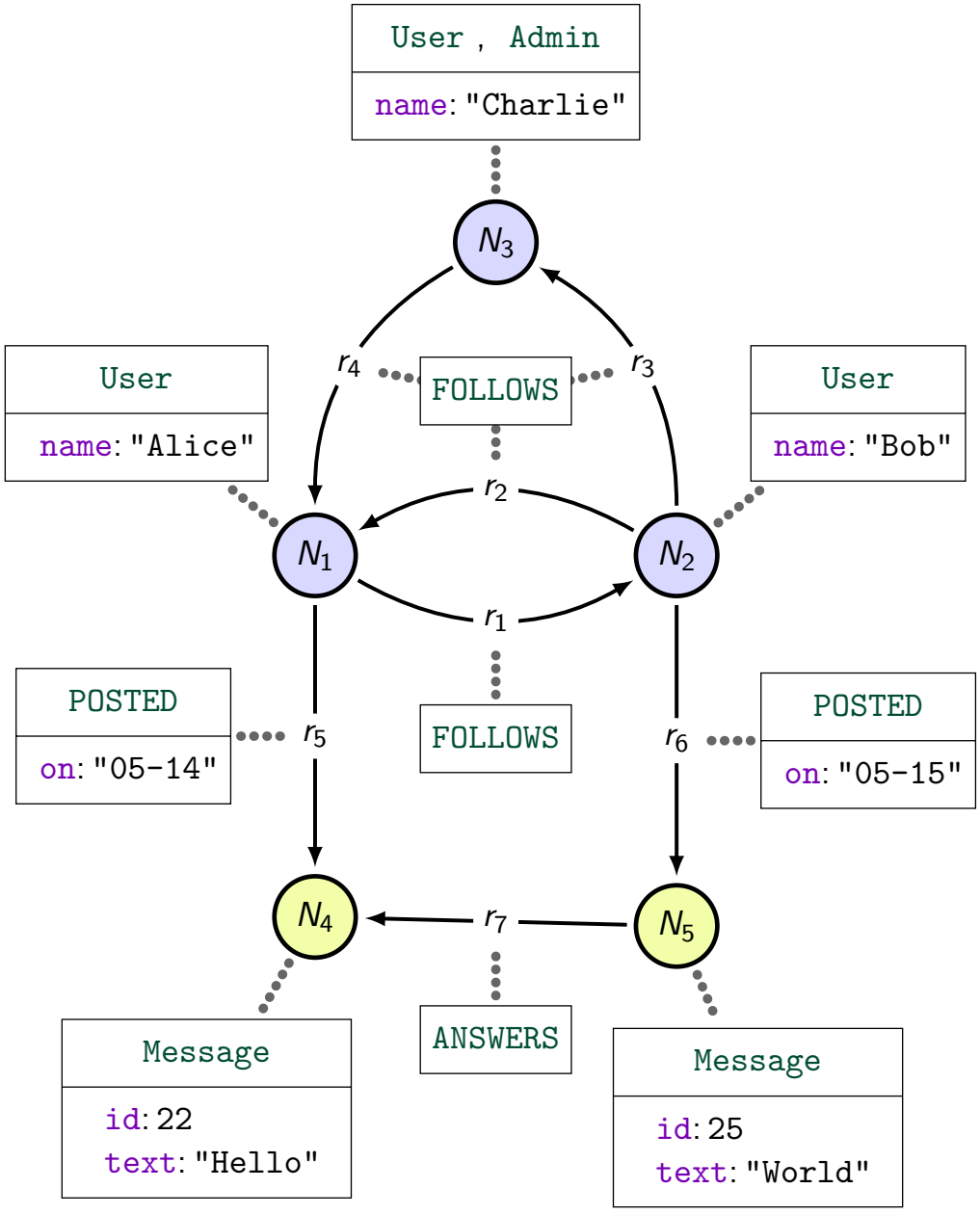
Query:

```
MATCH (u1:Admin)
          -[l1:FOLLOWS*]->(m1)
```

Result:

u1	l1	m1
N_3	$[r_4]$	N_1
N_3	$[r_4, r_1]$	N_2
N_3	$[r_4, r_1, r_2]$	N_1
N_3	$[r_4, r_1, r_3]$	N_3

Matching paths (1)



Query:

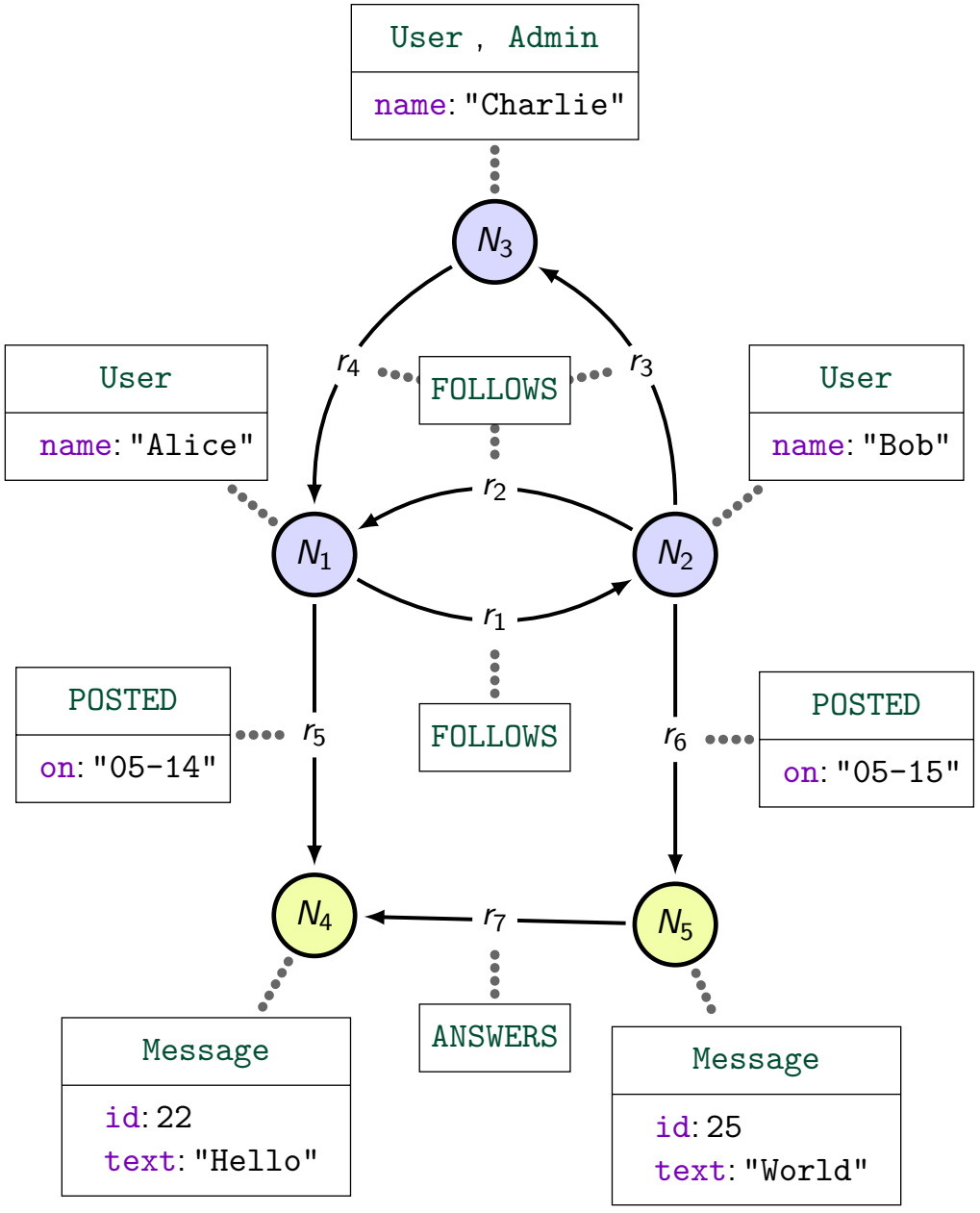
```
MATCH (u1:Admin)
          -[l1:FOLLOWS*]->(m1)
```

Result:

u1	l1	m1
N_3	$[r_4]$	N_1
N_3	$[r_4, r_1]$	N_2
N_3	$[r_4, r_1, r_2]$	N_1
N_3	$[r_4, r_1, r_3]$	N_3

- Cypher uses **trail semantics**
- In Cypher $*$ means **one or more**

Matching paths (2)



Query:

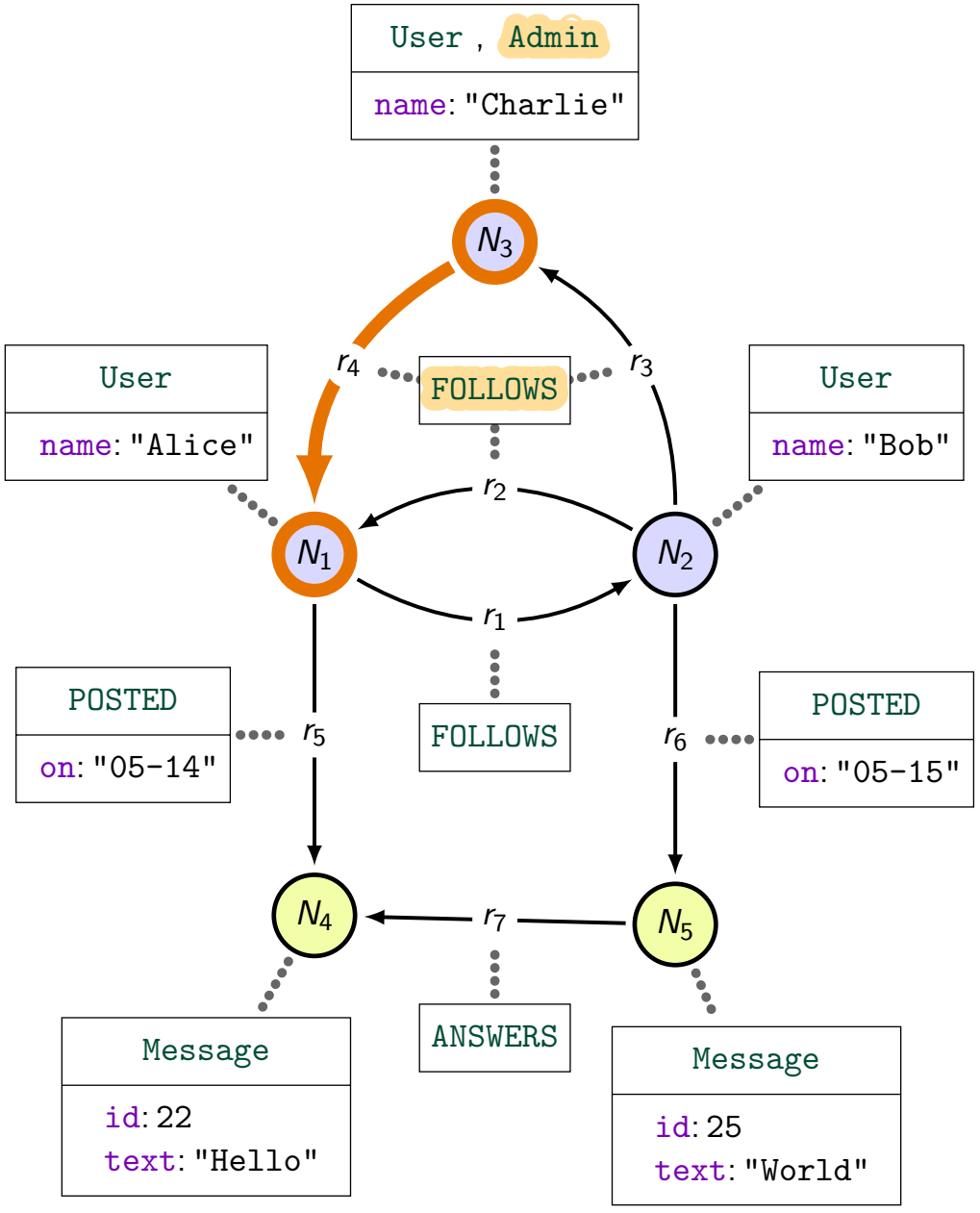
```
MATCH p= (:Admin)
         -[:FOLLOWS*]->(m1)
```

Result:

p1
$N_3 \xrightarrow{r_4} N_1$
$N_3 \xrightarrow{r_4} N_1 \xrightarrow{r_1} N_2$
$N_3 \xrightarrow{r_4} N_1 \xrightarrow{r_1} N_2 \xrightarrow{r_2} N_1$
$N_3 \xrightarrow{r_4} N_1 \xrightarrow{r_1} N_2 \xrightarrow{r_3} N_3$

- Path variables can be used =

Matching paths (2)



Query:

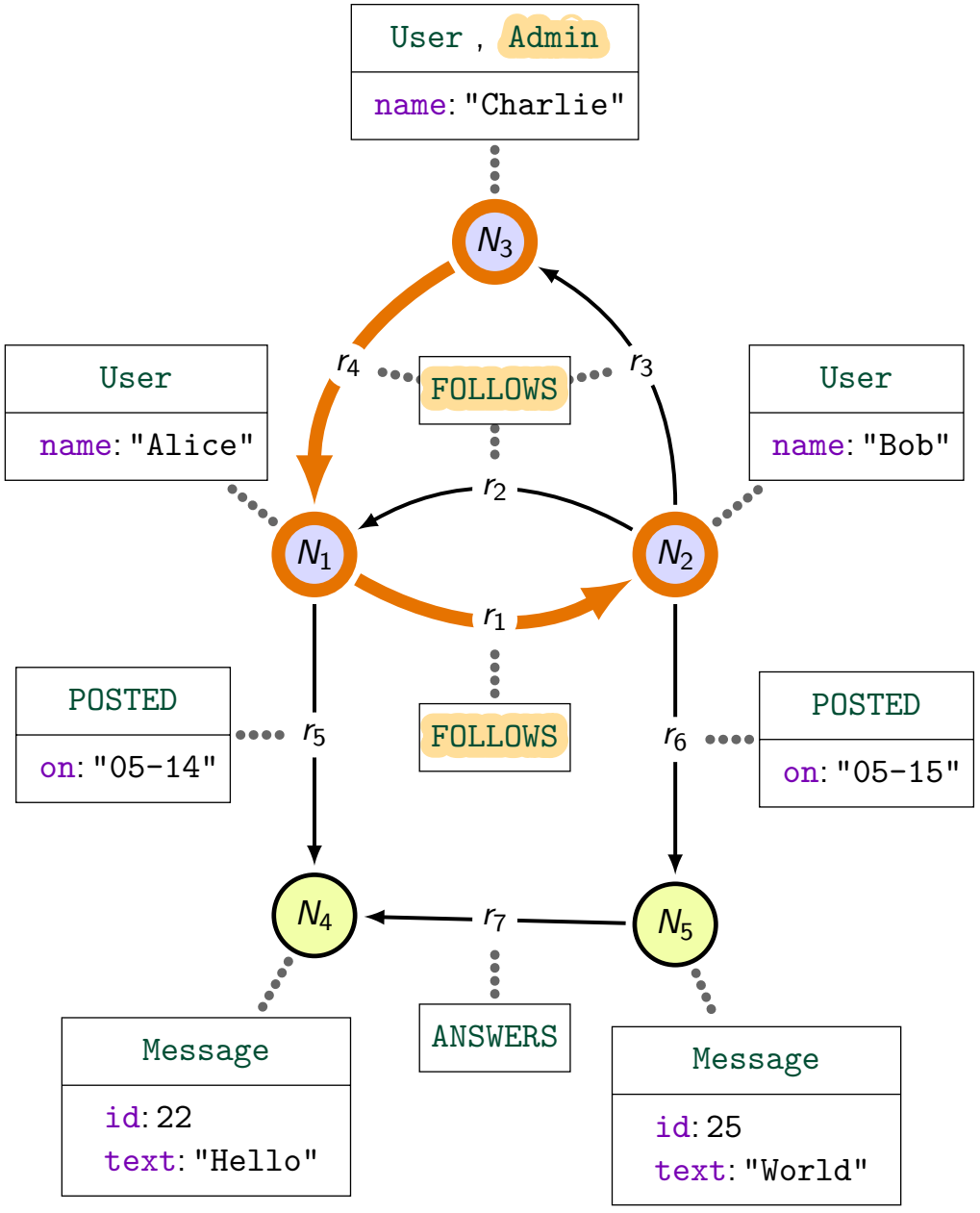
```
MATCH p= (:Admin)
         -[:FOLLOWS*]->(m1)
```

Result:

p1
$N_3 \xrightarrow{r_4} N_1$
$N_3 \xrightarrow{r_4} N_1 \xrightarrow{r_1} N_2$
$N_3 \xrightarrow{r_4} N_1 \xrightarrow{r_1} N_2 \xrightarrow{r_2} N_1$
$N_3 \xrightarrow{r_4} N_1 \xrightarrow{r_1} N_2 \xrightarrow{r_3} N_3$

- Path variables can be used =

Matching paths (2)



Query:

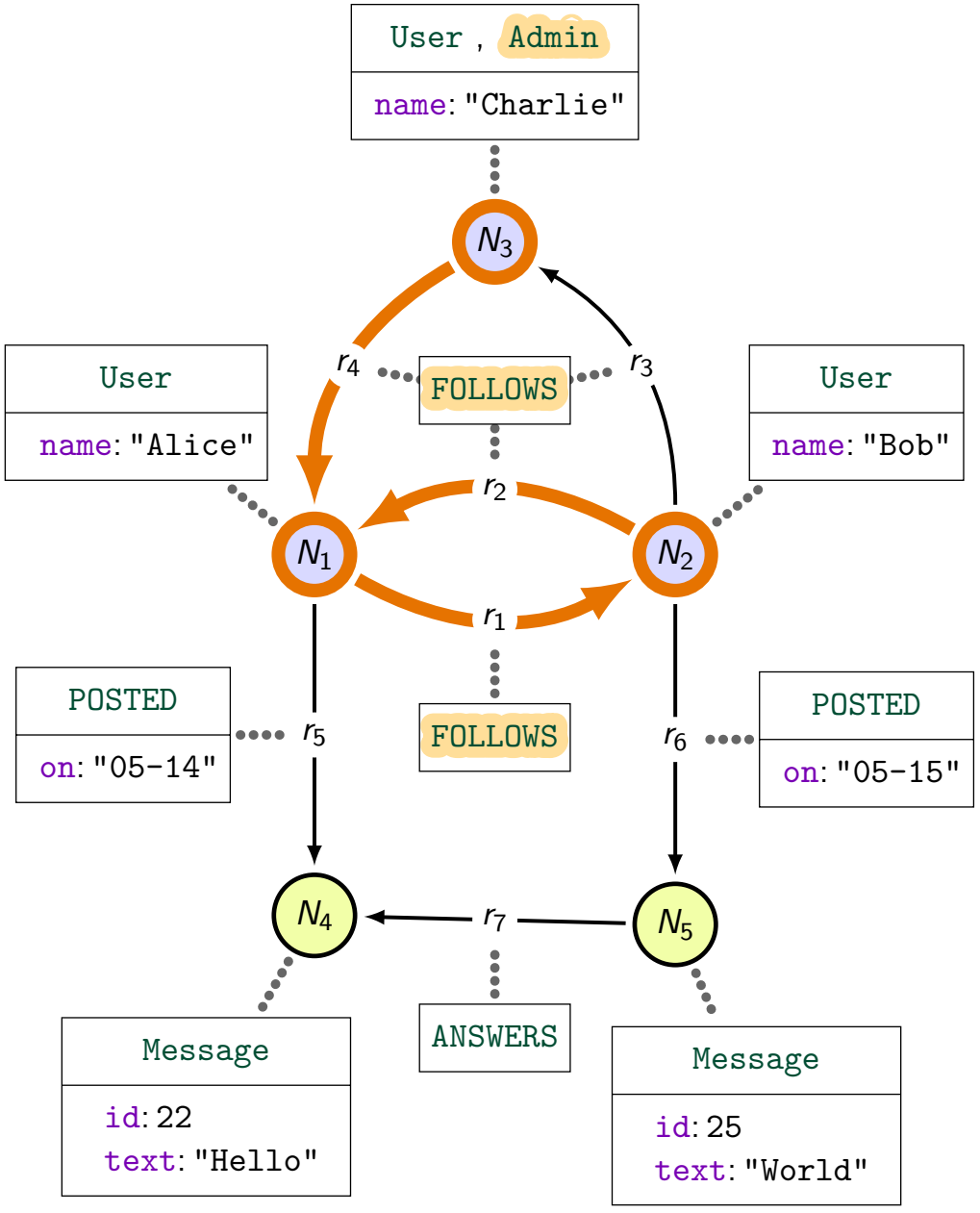
```
MATCH p= (:Admin)
         -[:FOLLOWS*]->(m1)
```

Result:

p1
$N_3 \xrightarrow{r_4} N_1$
$N_3 \xrightarrow{r_4} N_1 \xrightarrow{r_1} N_2$
$N_3 \xrightarrow{r_4} N_1 \xrightarrow{r_1} N_2 \xrightarrow{r_2} N_1$
$N_3 \xrightarrow{r_4} N_1 \xrightarrow{r_1} N_2 \xrightarrow{r_3} N_3$

- Path variables can be used =

Matching paths (2)



Query:

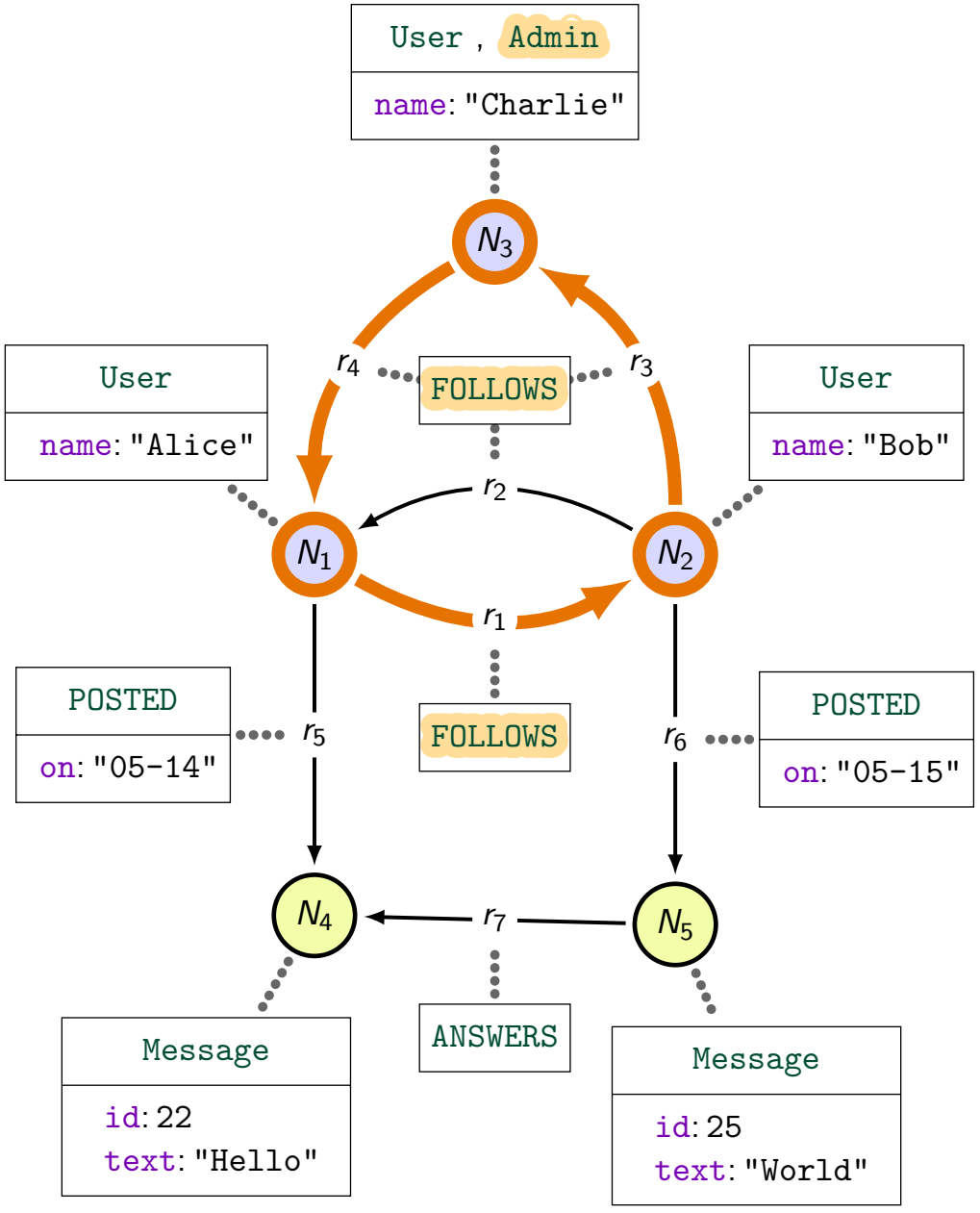
```
MATCH p= (:Admin)
         -[:FOLLOWS*]->(m1)
```

Result:

p1
$N_3 \xrightarrow{r_4} N_1$
$N_3 \xrightarrow{r_4} N_1 \xrightarrow{r_1} N_2$
$N_3 \xrightarrow{r_4} N_1 \xrightarrow{r_1} N_2 \xrightarrow{r_2} N_1$
$N_3 \xrightarrow{r_4} N_1 \xrightarrow{r_1} N_2 \xrightarrow{r_3} N_3$

- Path variables can be used =

Matching paths (2)



Query:

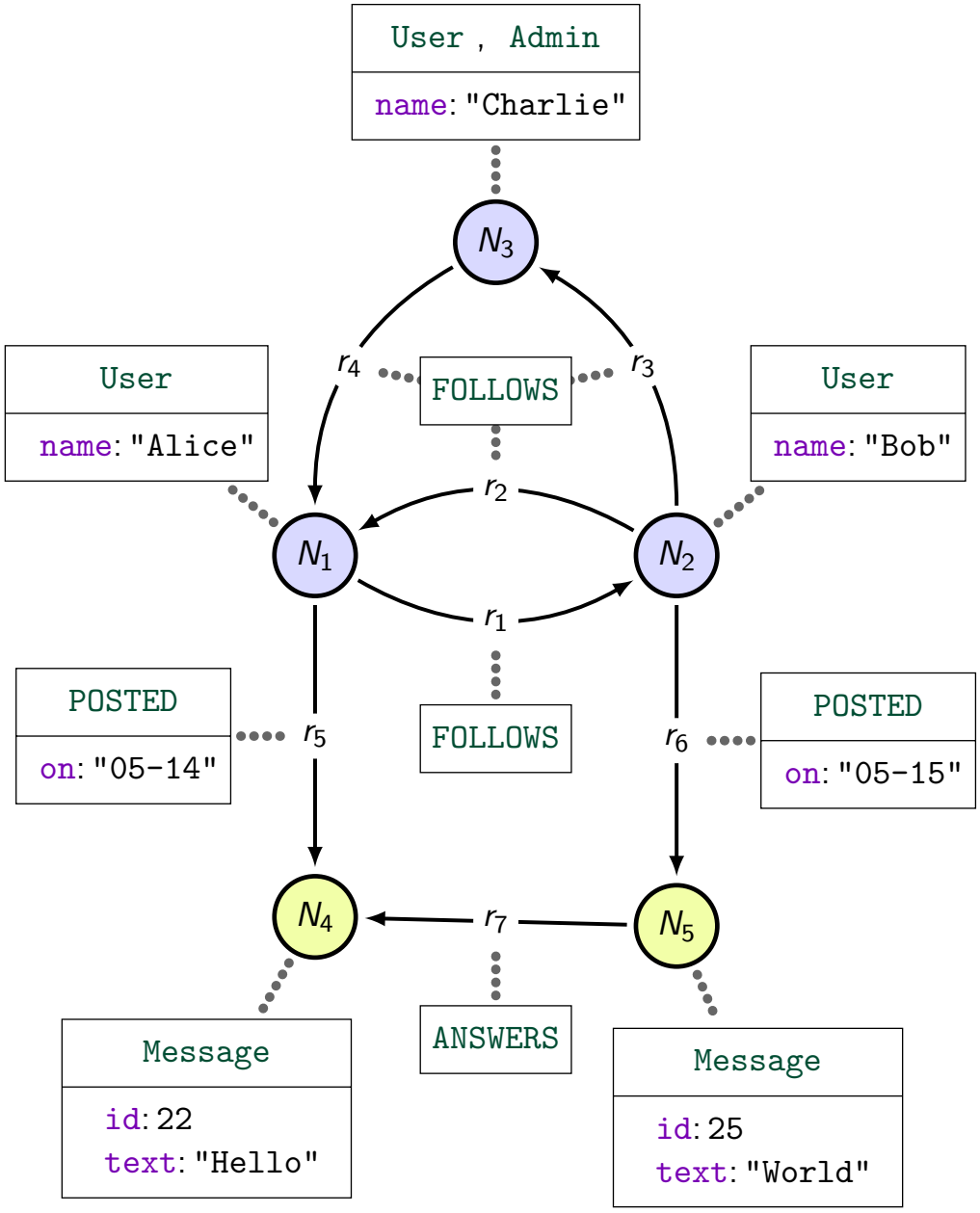
```
MATCH p= (:Admin)
         -[:FOLLOWS*]->(m1)
```

Result:

p1
$N_3 \xrightarrow{r_4} N_1$
$N_3 \xrightarrow{r_4} N_1 \xrightarrow{r_1} N_2$
$N_3 \xrightarrow{r_4} N_1 \xrightarrow{r_1} N_2 \xrightarrow{r_2} N_1$
$N_3 \xrightarrow{r_4} N_1 \xrightarrow{r_1} N_2 \xrightarrow{r_3} N_3$

- Path variables can be used =

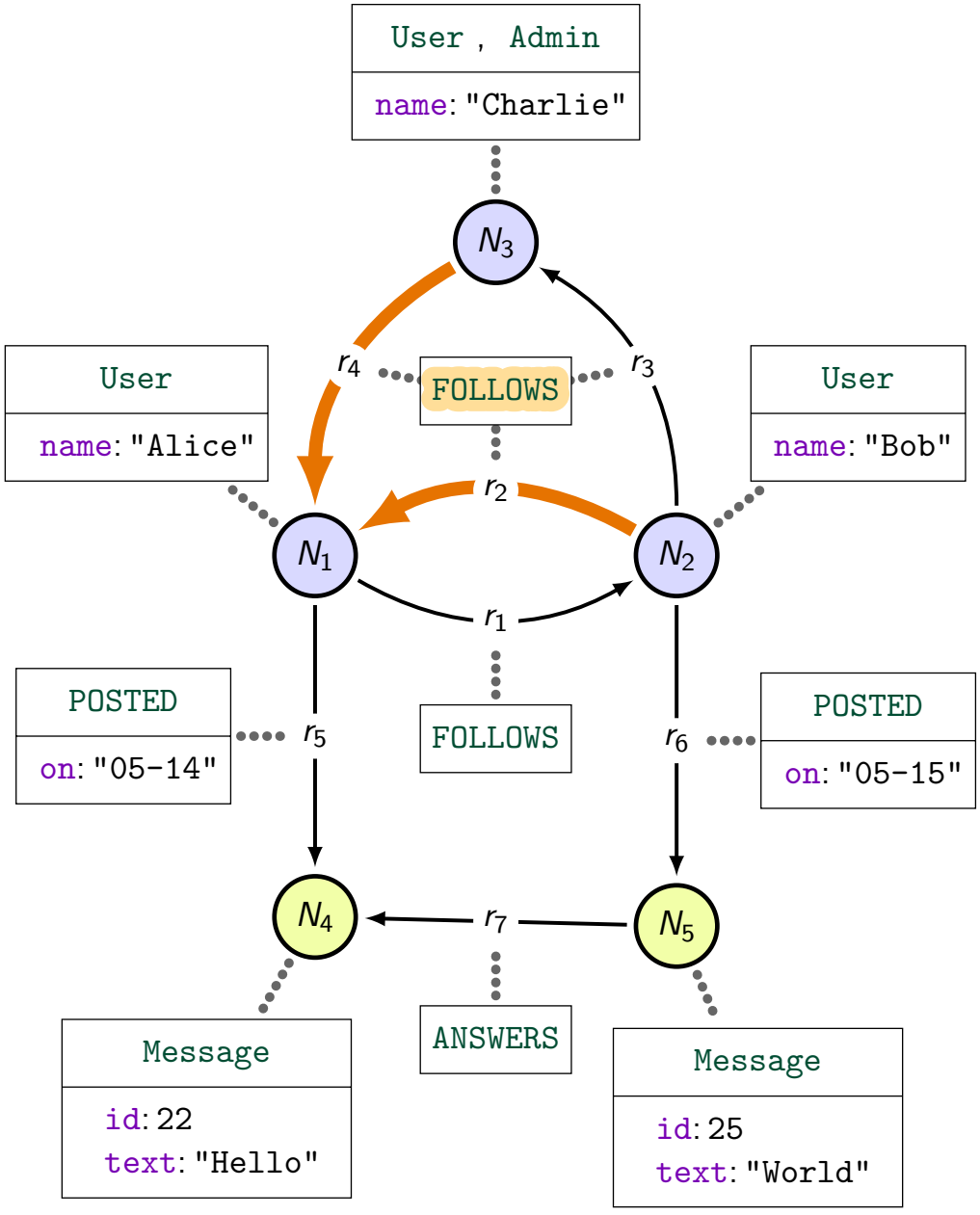
An interesting query



Query:

```
MATCH (u2)-[:FOLLOWS]->
        (u1)<-[:FOLLOWS]-(u3)
```

An interesting query



Query:

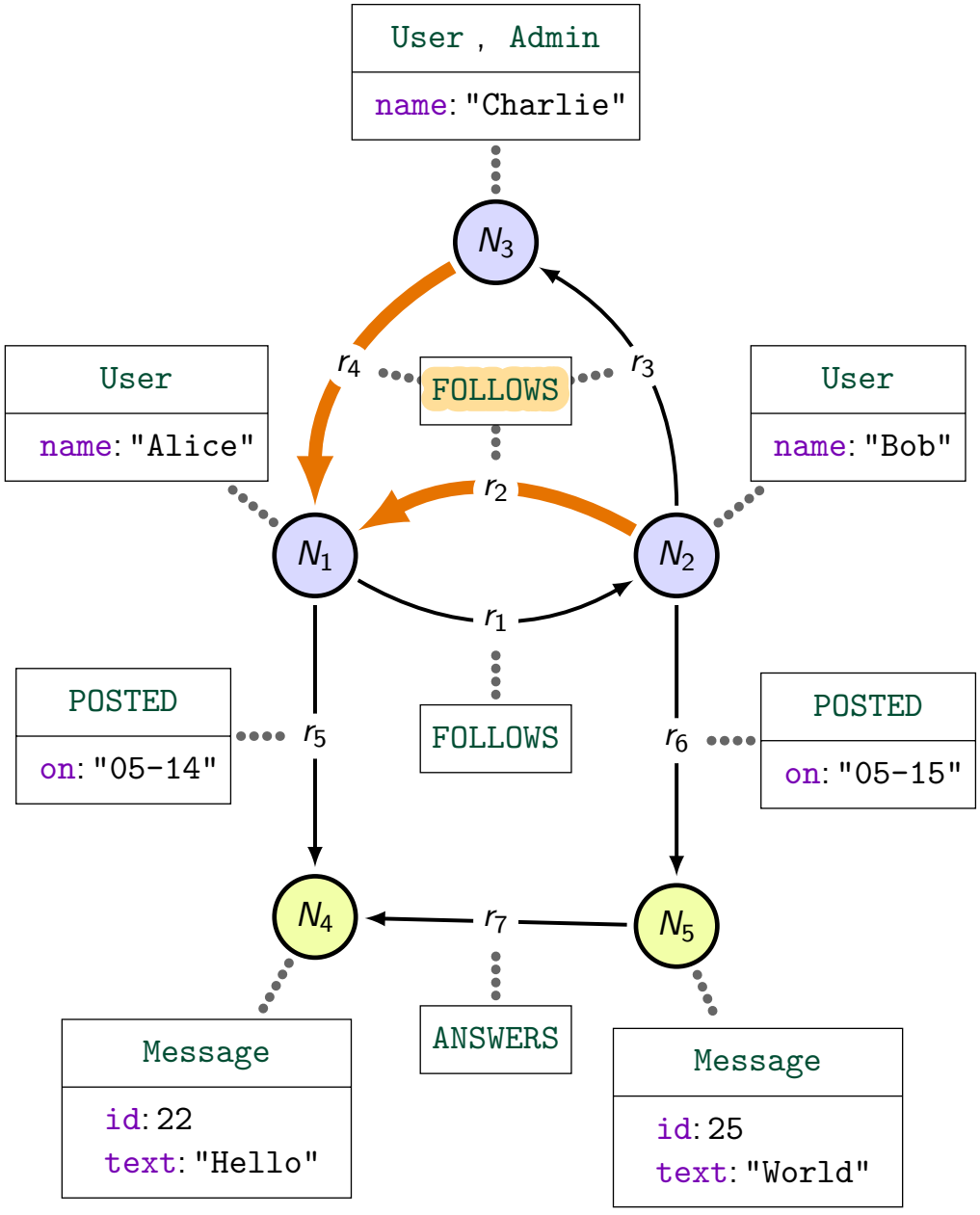
```
MATCH (u2)-[:FOLLOWS]->
        (u1)<-[:FOLLOWS]-(u3)
```

Result:

u2	u1	u3
N_3	N_1	N_2
N_2	N_1	N_3

- Line 1: $N_3 \xrightarrow{r_4} N_1 \xleftarrow{r_2} N_2$
- Line 2: $N_2 \xrightarrow{r_2} N_1 \xleftarrow{r_4} N_3$
- No (N_3, N_1, N_3) due to trail semantics

An interesting query



Query:

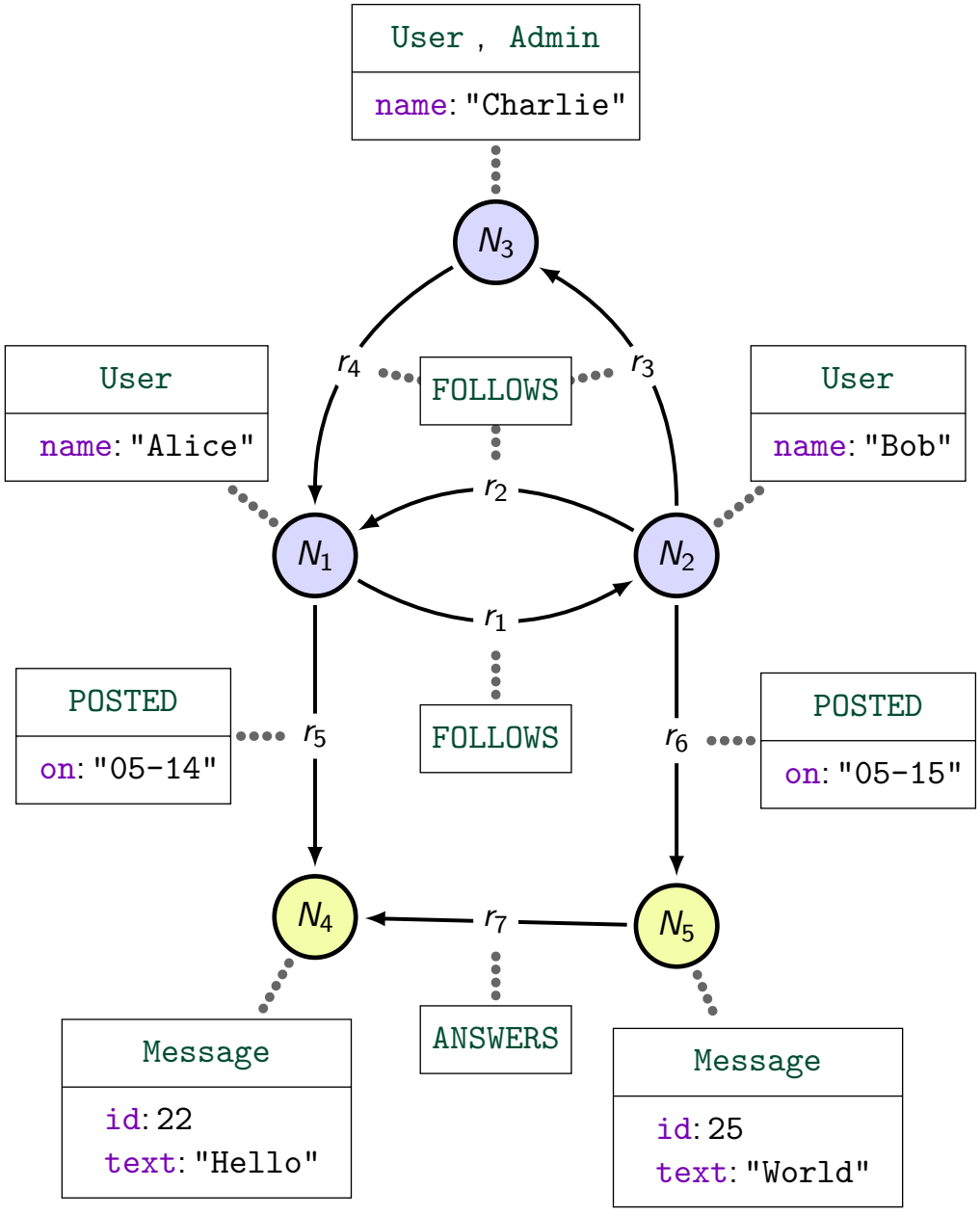
```
MATCH (u2)-[:FOLLOWS]->
      (u1)<-[:FOLLOWS]-(u3)
```

Result:

u2	u1	u3
N_3	N_1	N_2
N_2	N_1	N_3

- Line 1: $N_3 \xrightarrow{r_4} N_1 \xleftarrow{r_2} N_2$
- Line 2: $N_2 \xrightarrow{r_2} N_1 \xleftarrow{r_4} N_3$
- No (N_3, N_1, N_3) due to trail semantics

Matching paths (3)



Query:

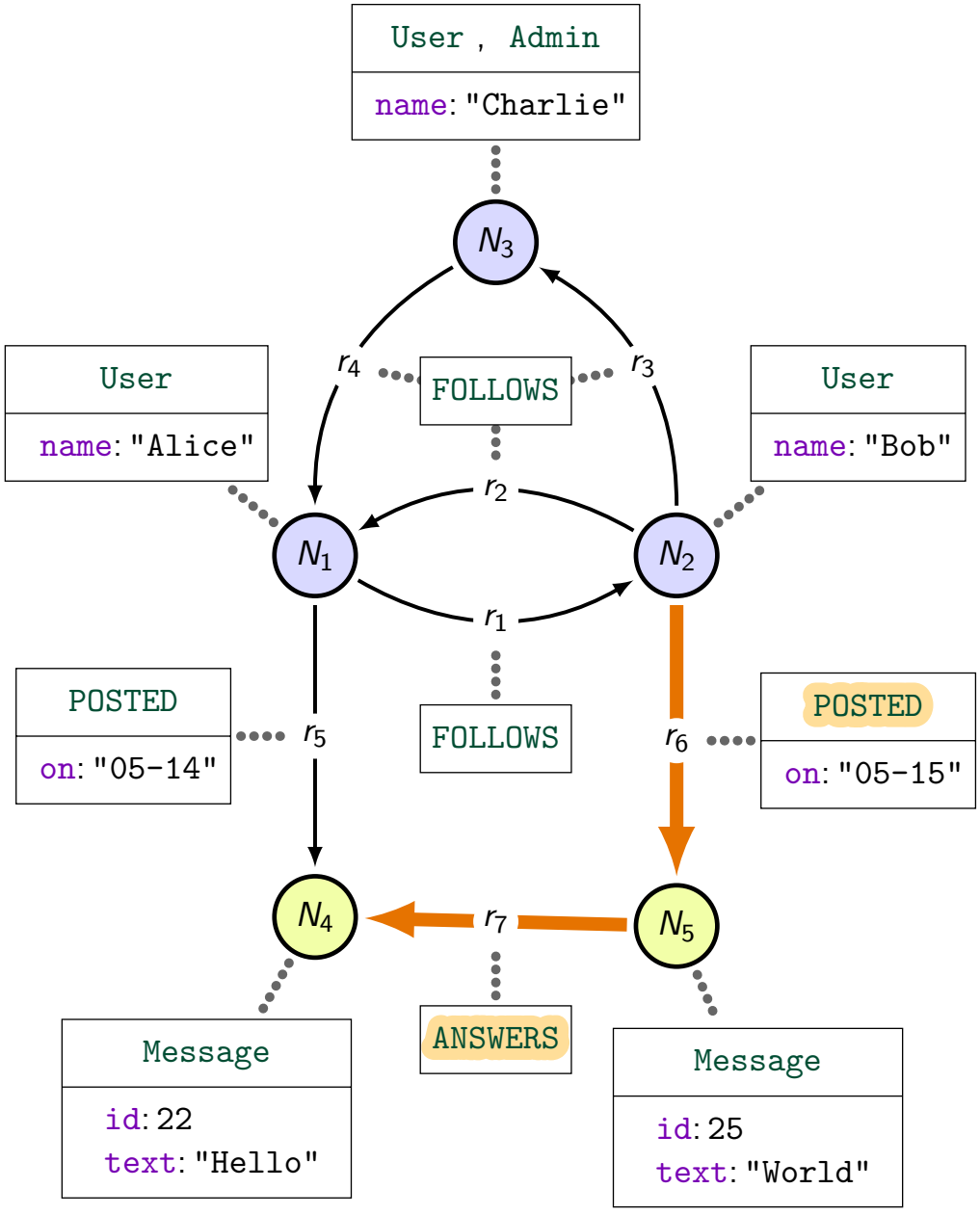
```
MATCH (u1)
  -[l1:POSTED|ANSWERS *]->(m1)
```

Result:

u1	l1	m1
N_2	$[r_6, r_7]$	N_4
N_5	$[r_7]$	N_4
N_2	$[r_6]$	N_5
N_1	$[r_5]$	N_4

- In edge patterns, disjunction of labels uses `:` and `|`

Matching paths (3)



Query:

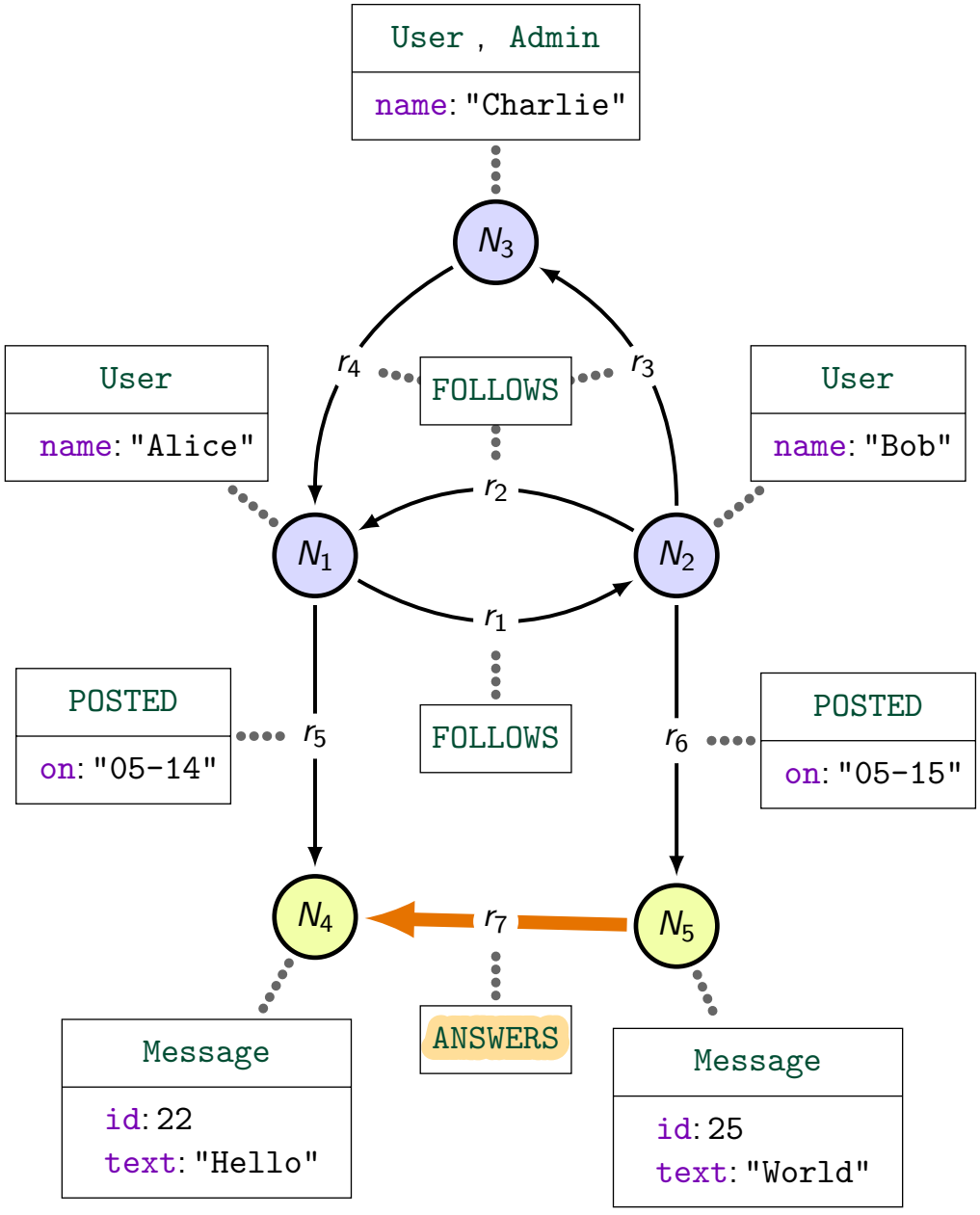
```
MATCH (u1)
  -[l1:POSTED|ANSWERS *]->(m1)
```

Result:

u1	l1	m1
N_2	$[r_6, r_7]$	N_4
N_5	$[r_7]$	N_4
N_2	$[r_6]$	N_5
N_1	$[r_5]$	N_4

- In edge patterns, disjunction of labels uses `:` and `|`

Matching paths (3)



Query:

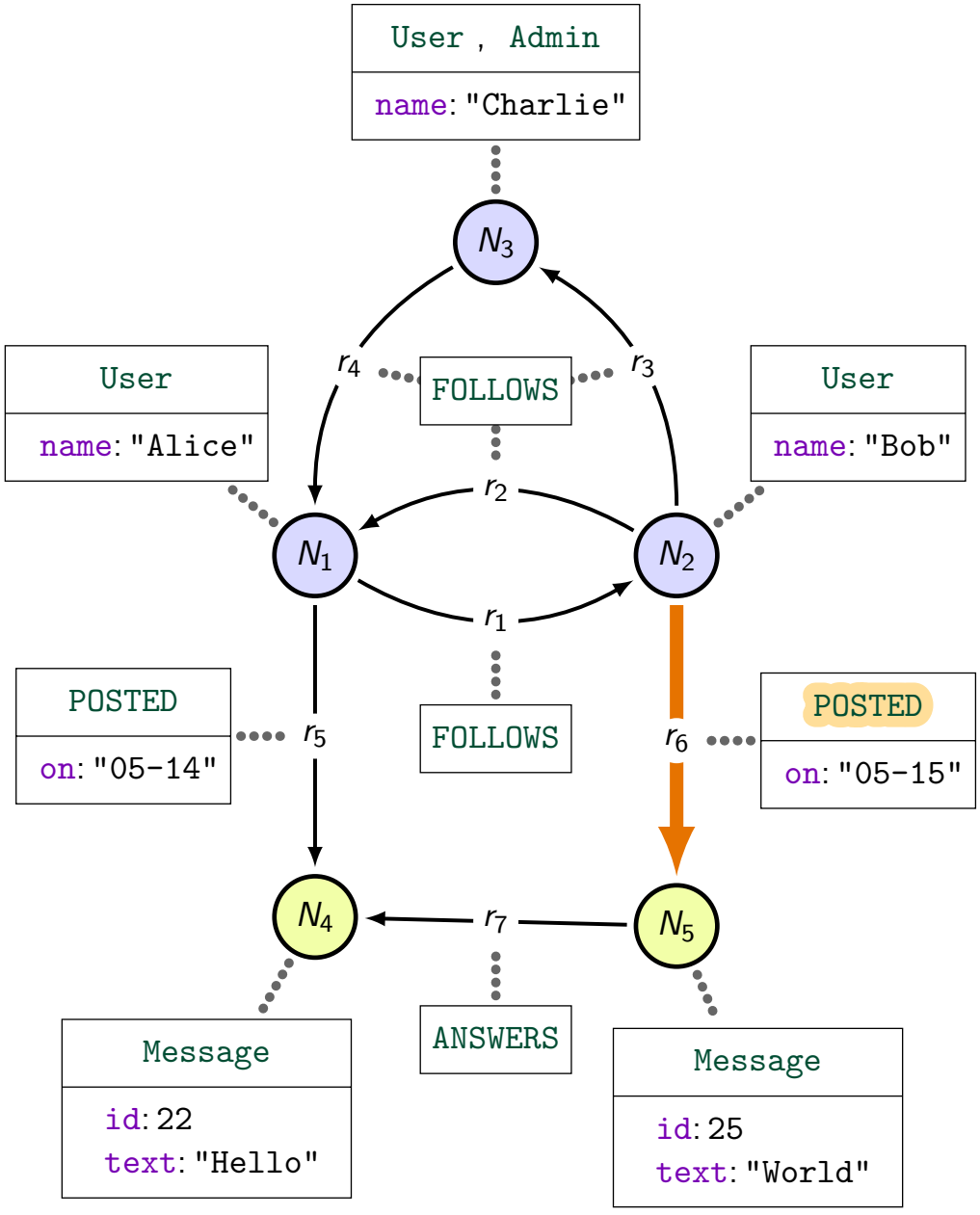
```
MATCH (u1)
  -[l1:POSTED|ANSWERS *]->(m1)
```

Result:

u1	l1	m1
N_2	$[r_6, r_7]$	N_4
N_5	$[r_7]$	N_4
N_2	$[r_6]$	N_5
N_1	$[r_5]$	N_4

- In edge patterns, disjunction of labels uses `:` and `|`

Matching paths (3)



Query:

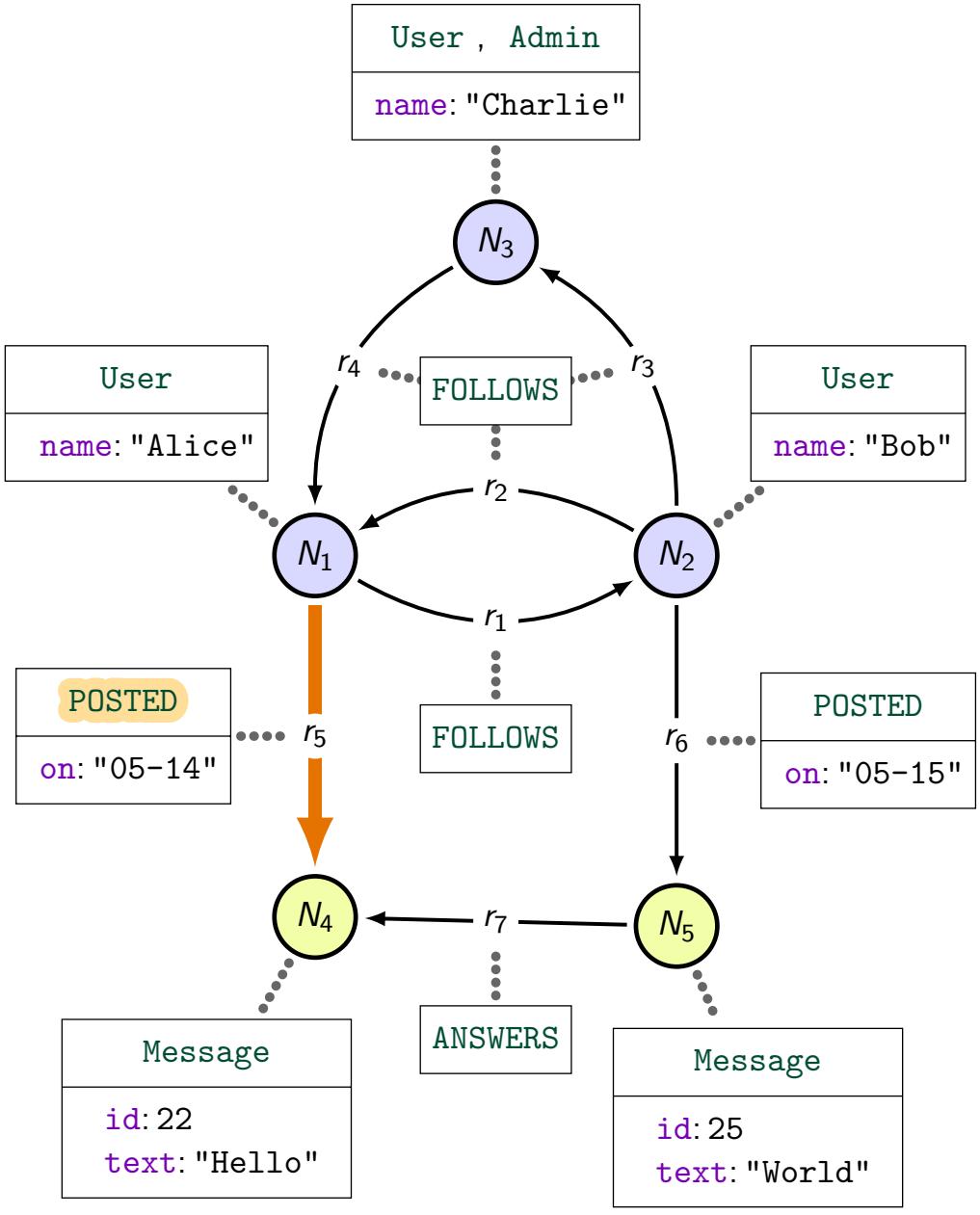
```
MATCH (u1)
  -[l1:POSTED|ANSWERS *]->(m1)
```

Result:

u1	l1	m1
N_2	$[r_6, r_7]$	N_4
N_5	$[r_7]$	N_4
N_2	$[r_6]$	N_5
N_1	$[r_5]$	N_4

- In edge patterns, disjunction of labels uses `:` and `|`

Matching paths (3)



Query:

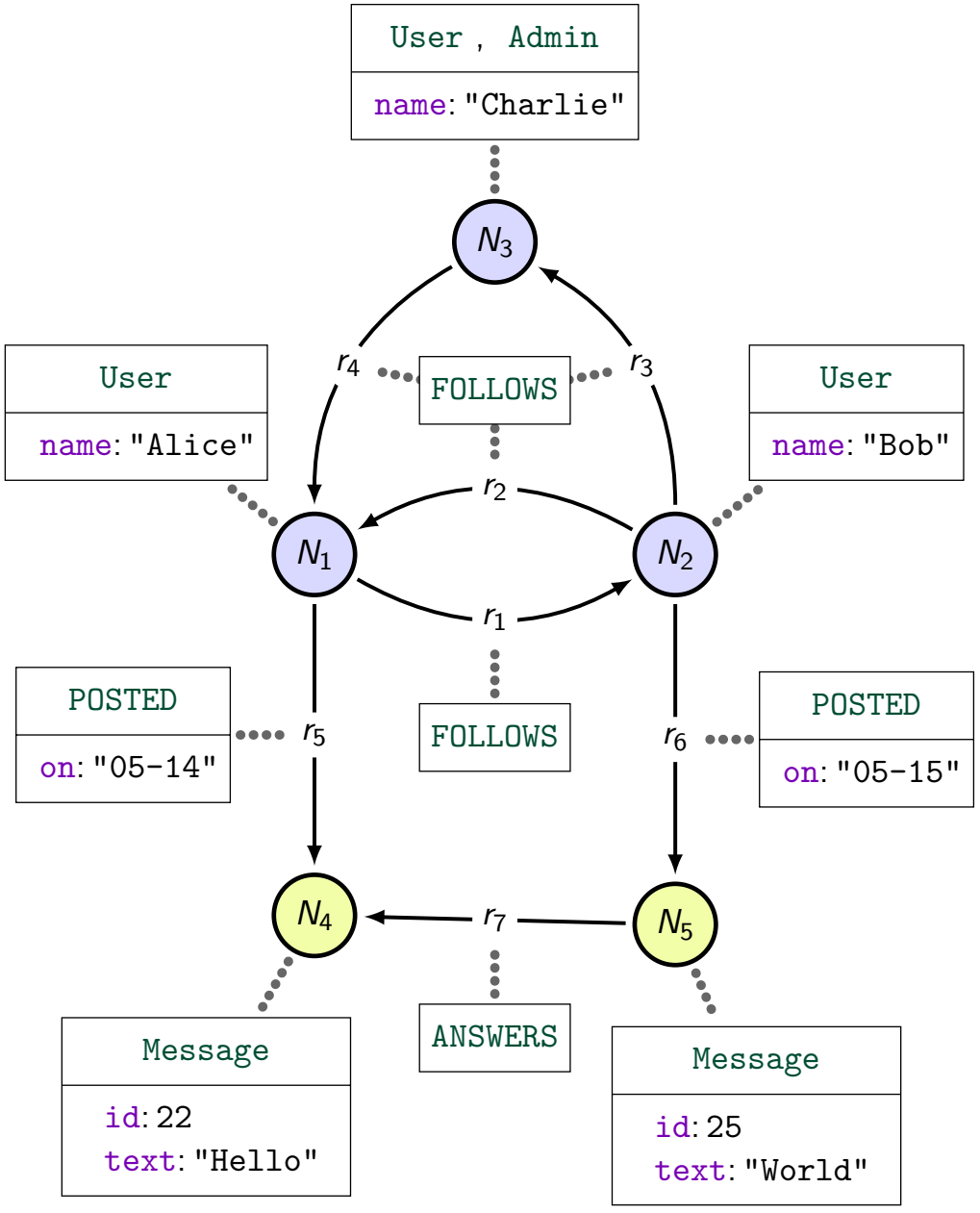
```
MATCH (u1)
  -[l1:POSTED|ANSWERS *]->(m1)
```

Result:

u1	l1	m1
N_2	$[r_6, r_7]$	N_4
N_5	$[r_7]$	N_4
N_2	$[r_6]$	N_5
N_1	$[r_5]$	N_4

- In edge patterns, disjunction of labels uses `:` and `|`

Matching subgraphs



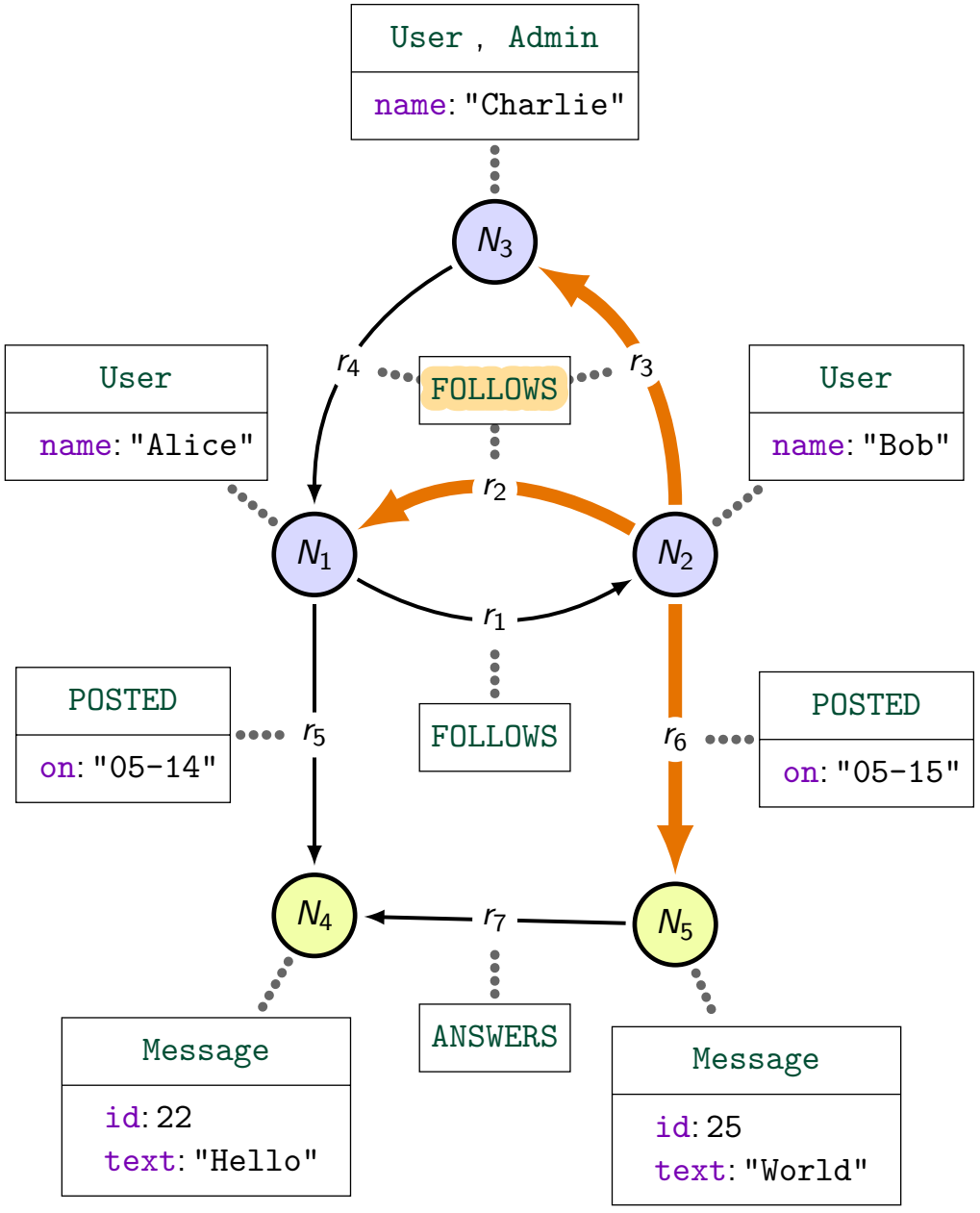
Query:

```
MATCH (u1)-[:FOLLOWS]->(u2),
      (u1)-[:FOLLOWS]->(u3),
      (u1)-[:POSTED]->(m1)
```

Result:

u1	u2	u3	m1
N_2	N_1	N_3	N_5
N_2	N_3	N_1	N_5

Matching subgraphs



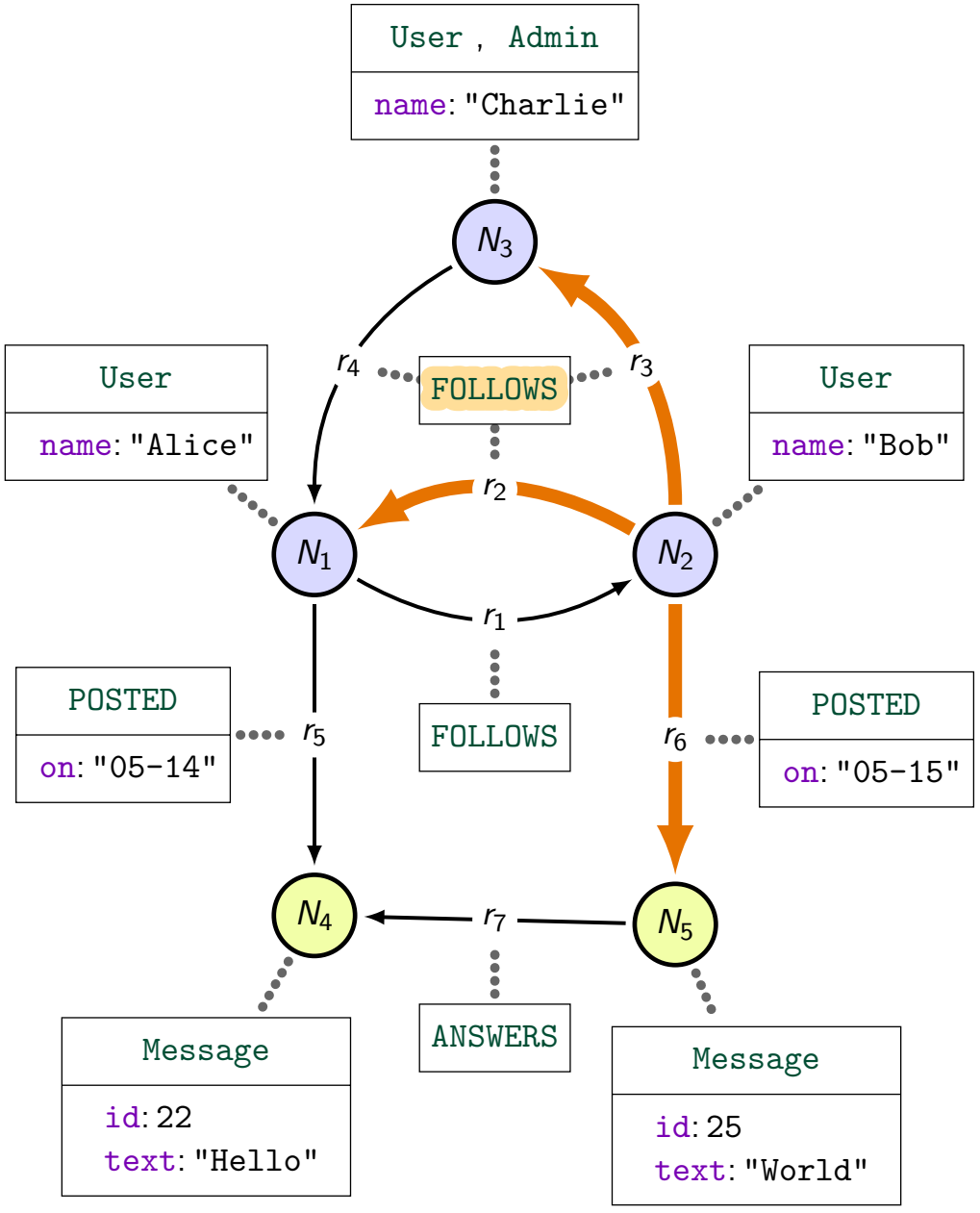
Query:

```
MATCH (u1)-[:FOLLOWS]->(u2),
      (u1)-[:FOLLOWS]->(u3),
      (u1)-[:POSTED]->(m1)
```

Result:

u1	u2	u3	m1
N_2	N_1	N_3	N_5
N_2	N_3	N_1	N_5

Matching subgraphs



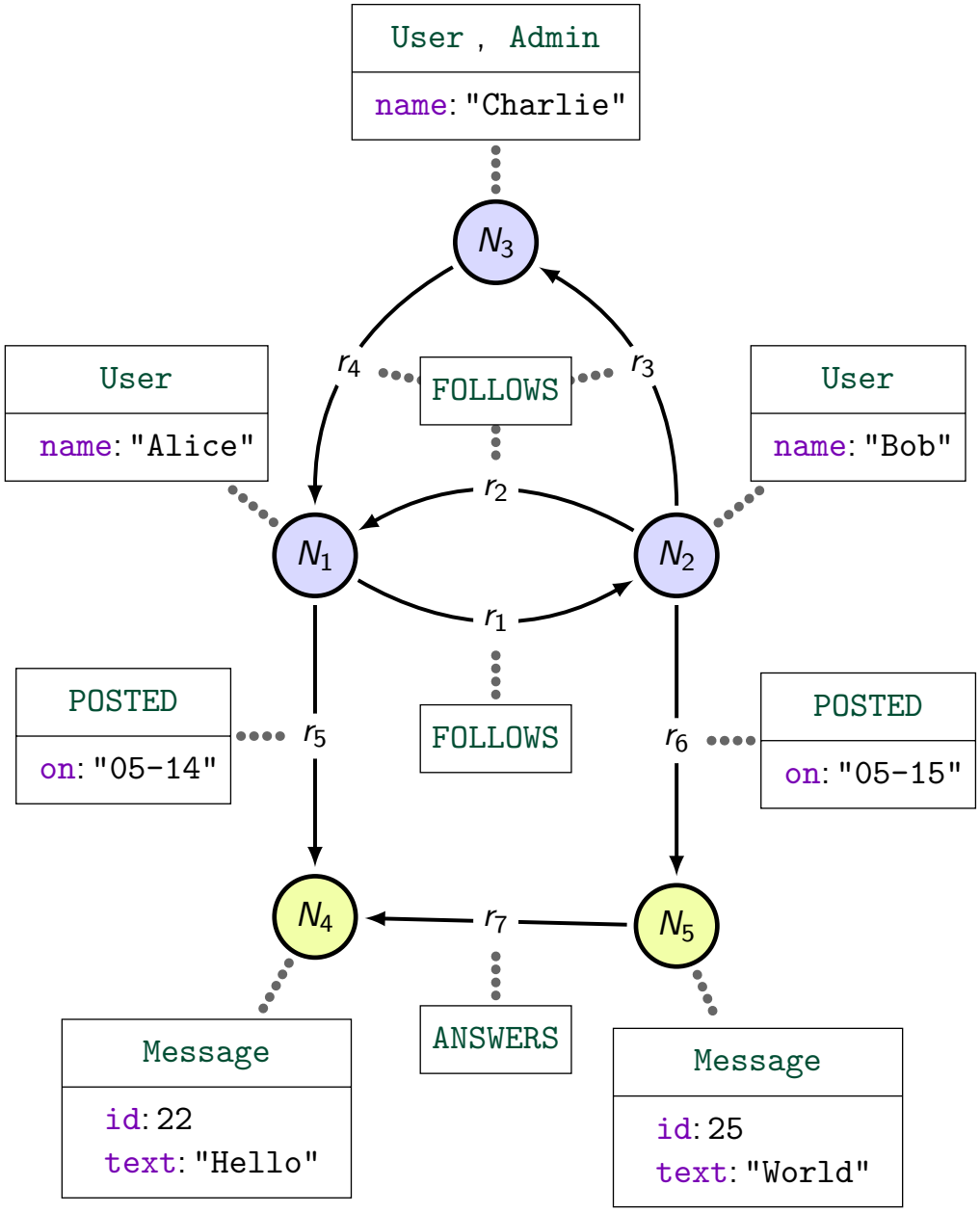
Query:

```
MATCH (u1)-[:FOLLOWS]->(u2),
      (u1)-[:FOLLOWS]->(u3),
      (u1)-[:POSTED]->(m1)
```

Result:

u1	u2	u3	m1
N_2	N_1	N_3	N_5
N_2	N_3	N_1	N_5

Matching subgraphs



Query:

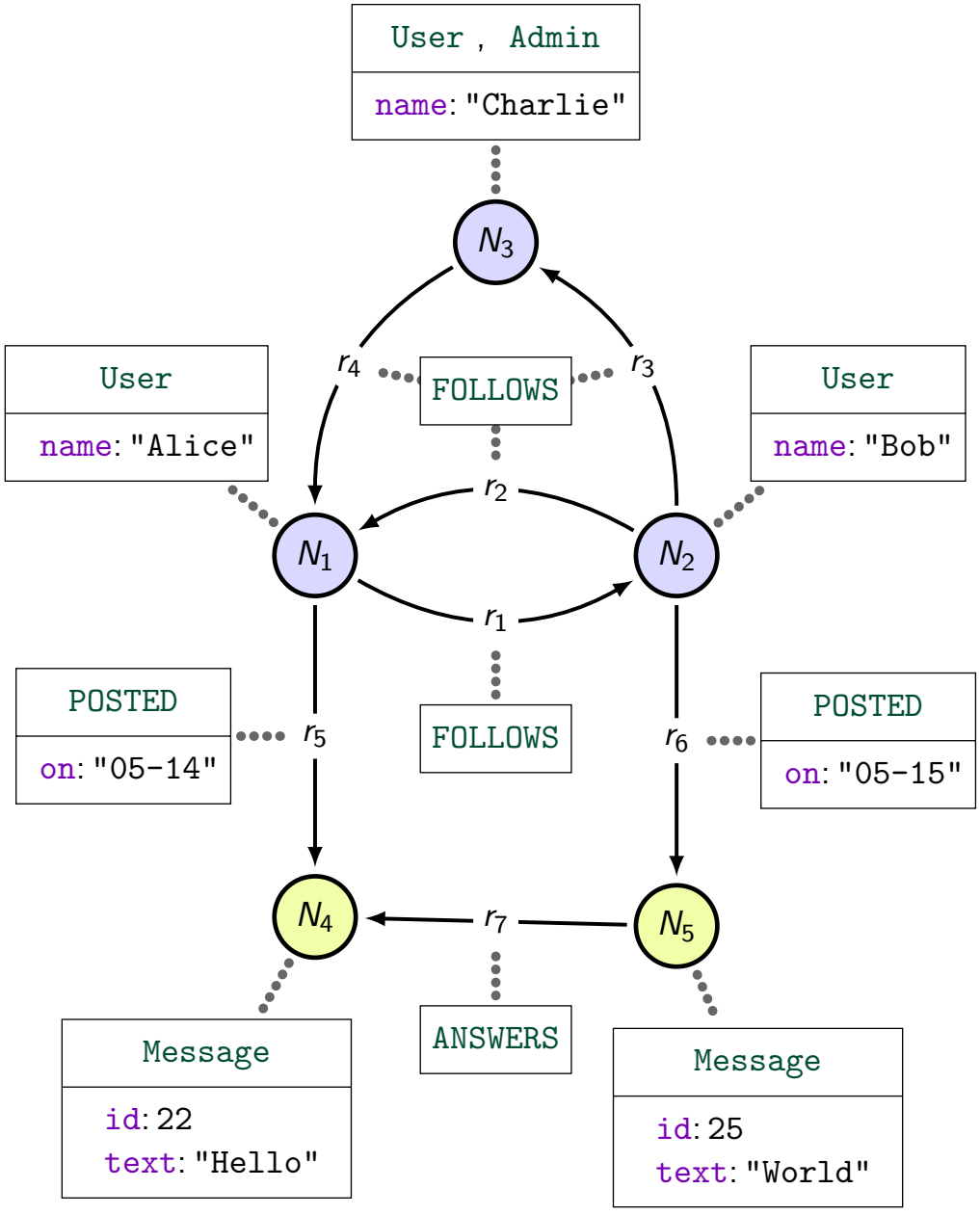
```
MATCH (u1)-[:FOLLOWS]->(u2),
      (u1)-[:FOLLOWS]->(u3),
      (u1)-[:POSTED]->(m1)
```

Result:

u1	u2	u3	m1
N_2	N_1	N_3	N_5
N_2	N_3	N_1	N_5

⚠ CRPQ ⚠

Matching subgraphs



Query:

```
MATCH (u1)-[:FOLLOWS]->(u2),
      (u1)-[:FOLLOWS]->(u3),
      (u1)-[:POSTED]->(m1)
```

Result:

u1	u2	u3	m1
N_2	N_1	N_3	N_5
N_2	N_3	N_1	N_5

⚠ CRPQ ⚠

⚠ Cartesian product ⚠

Recap of MATCH

- C2RPQ-like pattern-matching
  - Equi-join on variables that are used multiple times
- Trail semantics (no repeated edge in the same MATCH)
- Computation of the resulting table:
  - C2RPQ-like evaluations → tuples of walks
  - project on variables
  - return a table: variable as column names, one line per tuple of walks

`MATCH` allows C2RPQ-like pattern-matching

- Letters are put between brackets
  - $\mathbf{A} \rightsquigarrow$ `()-[:A]->()`
  - $\overline{\mathbf{B}} \rightsquigarrow$ `()<-[:B]-()`

`MATCH` allows C2RPQ-like pattern-matching

- Letters are put between brackets
  A $\rightsquigarrow$ `()-[:A]->()`
  $\overline{\mathbf{B}}$ $\rightsquigarrow$ `()<-[:B]-()`
- Repetitions follows a `*` in brackets
  $\mathbf{A}^+$ $\rightsquigarrow$ `()-[:A *]->()`
  $\mathbf{A}^*$ $\rightsquigarrow$ `()-[:A *0..]->()`

`MATCH` allows C2RPQ-like pattern-matching

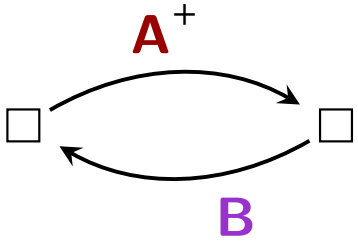
- Letters are put between brackets
  $\mathbf{A} \rightsquigarrow$ `()-[:A]->()`
  $\overline{\mathbf{B}} \rightsquigarrow$ `()<-[:B]-()`
- Repetitions follows a `*` in brackets
  $\mathbf{A}^{+} \rightsquigarrow$ `()-[:A *]->()`
  $\mathbf{A}^{*} \rightsquigarrow$ `()-[:A *0..]->()`
- Concatenation is done by direct chaining
  $\mathbf{A} \cdot \mathbf{B}^{+} \cdot \mathbf{C} \rightsquigarrow$ `()->[:A]->()-[:B*]->()-[:C]->()`

`MATCH` allows C2RPQ-like pattern-matching

- Letters are put between brackets
  $\mathbf{A} \rightsquigarrow$ `()-[:A]->()`
  $\overline{\mathbf{B}} \rightsquigarrow$ `()<-[:B]-()`
- Repetitions follows a `*` in brackets
  $\mathbf{A}^{+} \rightsquigarrow$ `()-[:A *]->()`
  $\mathbf{A}^{*} \rightsquigarrow$ `()-[:A *0..]->()`
- Concatenation is done by direct chaining
  $\mathbf{A} \cdot \mathbf{B}^{+} \cdot \mathbf{C} \rightsquigarrow$ `()->[:A]->()-[:B*]->()-[:C]->()`
- Union is simulated by `|` in bracket or any-directed edge patterns
  $\mathbf{A} + \mathbf{B} \rightsquigarrow$ `()-[:A|B]->()`
  $\mathbf{C} + \overline{\mathbf{C}} \rightsquigarrow$ `()-[:C]-()`

MATCH allows C2RPQ-like pattern-matching

- Letters are put between brackets
  A $\rightsquigarrow$ `()-[:A]->()`
  $\overline{\mathbf{B}}$ $\rightsquigarrow$ `()<-[:B]-()`
- Repetitions follows a `*` in brackets
  $\mathbf{A}^+$ $\rightsquigarrow$ `()-[:A *]->()`
  $\mathbf{A}^*$ $\rightsquigarrow$ `()-[:A *0..]->()`
- Concatenation is done by direct chaining
  $\mathbf{A} \cdot \mathbf{B}^+ \cdot \mathbf{C}$ $\rightsquigarrow$ `()->[:A]->()-[:B*]->()-[:C]->()`
- Union is simulated by `|` in bracket or any-directed edge patterns
  $\mathbf{A} + \mathbf{B}$ $\rightsquigarrow$ `()-[:A|B]->()`
  $\mathbf{C} + \overline{\mathbf{C}}$ $\rightsquigarrow$ `()-[:C]-()`
- CRPQs are simulated with commas



$\rightsquigarrow$ `(a)-[:A*]->(b), (b)-[:B]->(a)`

MATCH allows C2RPQ-like pattern-matching

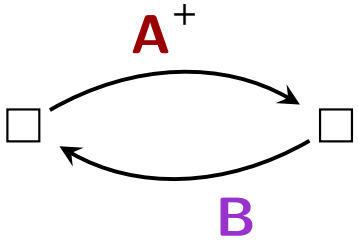
- Letters are put between brackets
  $A \rightsquigarrow$ `()-[:A]->()`
  $\overline{B} \rightsquigarrow$ `()<-[:B]-()`

- Repetitions follows a `*` in brackets
  $A^+ \rightsquigarrow$ `()-[:A *]->()`
  $A^* \rightsquigarrow$ `()-[:A *0..]->()`

- Concatenation is done by direct chaining
  $A \cdot B^+ \cdot C \rightsquigarrow$ `()->[:A]->()-[:B*]->()-[:C]->()`

- Union is simulated by `|` in bracket or any-directed edge patterns
  $A + B \rightsquigarrow$ `()-[:A|B]->()`
  $C + \overline{C} \rightsquigarrow$ `()-[:C]-()`

- CRPQs are simulated with commas



$\rightsquigarrow$ `(a)-[:A*]->(b), (b)-[:B]->(a)`

Exercice: find RPQs, 2RPQs and 2CRPQs that are not expressible with MATCH

MATCH cannot express all C2RPQs

MATCH cannot express all C2RPQs

RPQs

- Only atoms can be unionized — $\mathbf{AA} + \mathbf{BB}$
- No nested stars — $(\mathbf{A}^* \mathbf{B})^*$
- No concatenation under star — $(\mathbf{A} \cdot \mathbf{B})^*$

MATCH cannot express all C2RPQs

RPQs

- Only atoms can be unionized — $\mathbf{AA} + \mathbf{BB}$
- No nested stars — $(\mathbf{A}^*\mathbf{B})^*$
- No concatenation under star — $(\mathbf{A} \cdot \mathbf{B})^*$

2RPQs

- Unions of atoms with inconsistent directions — $\mathbf{A} + \overline{\mathbf{B}}$

NB: $\mathbf{A} + \overline{\mathbf{A}} + \mathbf{B} + \overline{\mathbf{B}}$ is expressible with `()-[:A|B]-()`

MATCH cannot express all C2RPQs

RPQs

- Only atoms can be unionized — $\mathbf{AA} + \mathbf{BB}$
- No nested stars — $(\mathbf{A}^*\mathbf{B})^*$
- No concatenation under star — $(\mathbf{A} \cdot \mathbf{B})^*$

2RPQs

- Unions of atoms with inconsistent directions — $\mathbf{A} + \overline{\mathbf{B}}$

NB: $\mathbf{A} + \overline{\mathbf{A}} + \mathbf{B} + \overline{\mathbf{B}}$ is expressible with `()-[:A|B]-()`

C2RPQs

- No further restrictions

MATCH goes beyond C2RPQs

- Testing properties

```
MATCH ()-[{date:"22-12"}]->()
```

MATCH goes beyond C2RPQs

- Testing properties

```
MATCH ()-[{date:"22-12"}]->()
```

- Testing labels and properties **on nodes**

```
MATCH (:Admin)
MATCH ({id:21})
```

MATCH goes beyond C2RPQs

- Testing properties
  ```
  MATCH ()-[{date:"22-12"}]->()
  ```
- Testing labels and properties **on nodes**
  ```
  MATCH (:Admin)
  MATCH ({id:21})
  ```
- Returning part of the matched walks thanks to variable
  `MATCH (a)-[:Road*]->()` ⇝ source nodes
  `MATCH ()-[b:Road*]->()` ⇝ edge lists
  `MATCH ()-[:Road*]->(c:Gas)-[:Road*]->()` ⇝ middle nodes

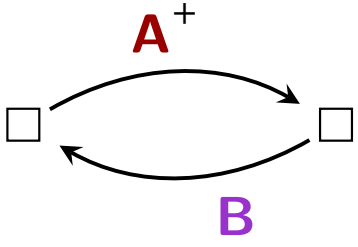
MATCH goes beyond C2RPQs

- Testing properties
  `MATCH ()-[{date:"22-12"}]->()`

- Testing labels and properties **on nodes**
  `MATCH (:Admin)`
  `MATCH ({id:21})`

- Returning part of the matched walks thanks to variable
  `MATCH (a)-[:Road*]->()` ⇝ source nodes
  `MATCH ()-[b:Road*]->()` ⇝ edge lists
  `MATCH ()-[:Road*]->(c:Gas)-[:Road*]->()` ⇝ middle nodes

- Variable reuse allows lightweitht C2RPQ without commas

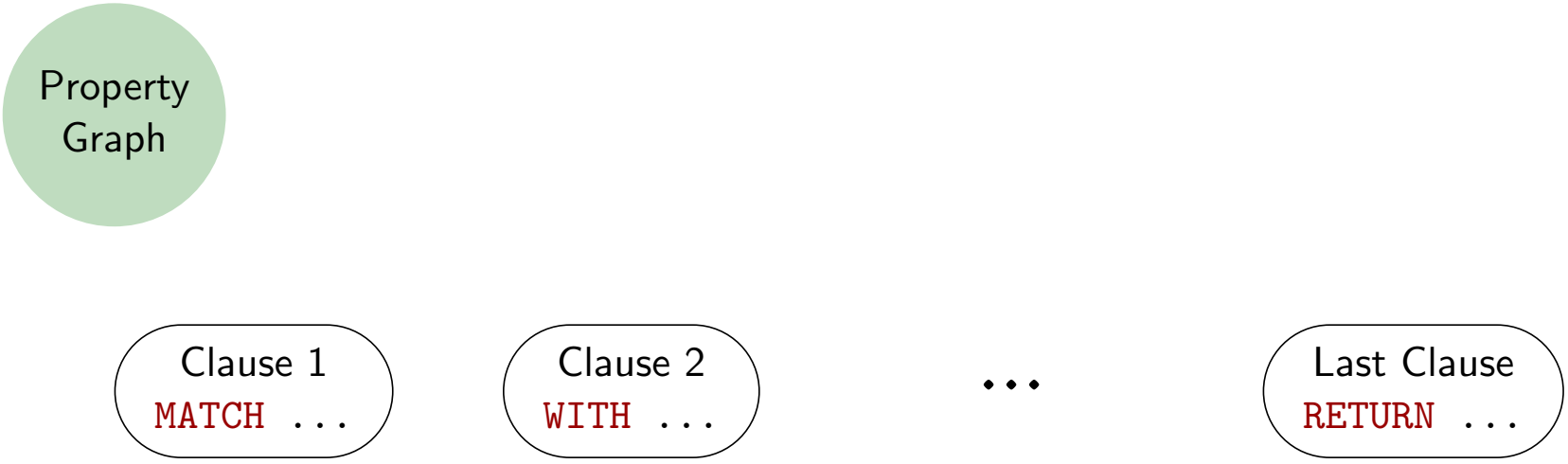


⇝ `(a)-[:A*]->()-[:B]->(a)`

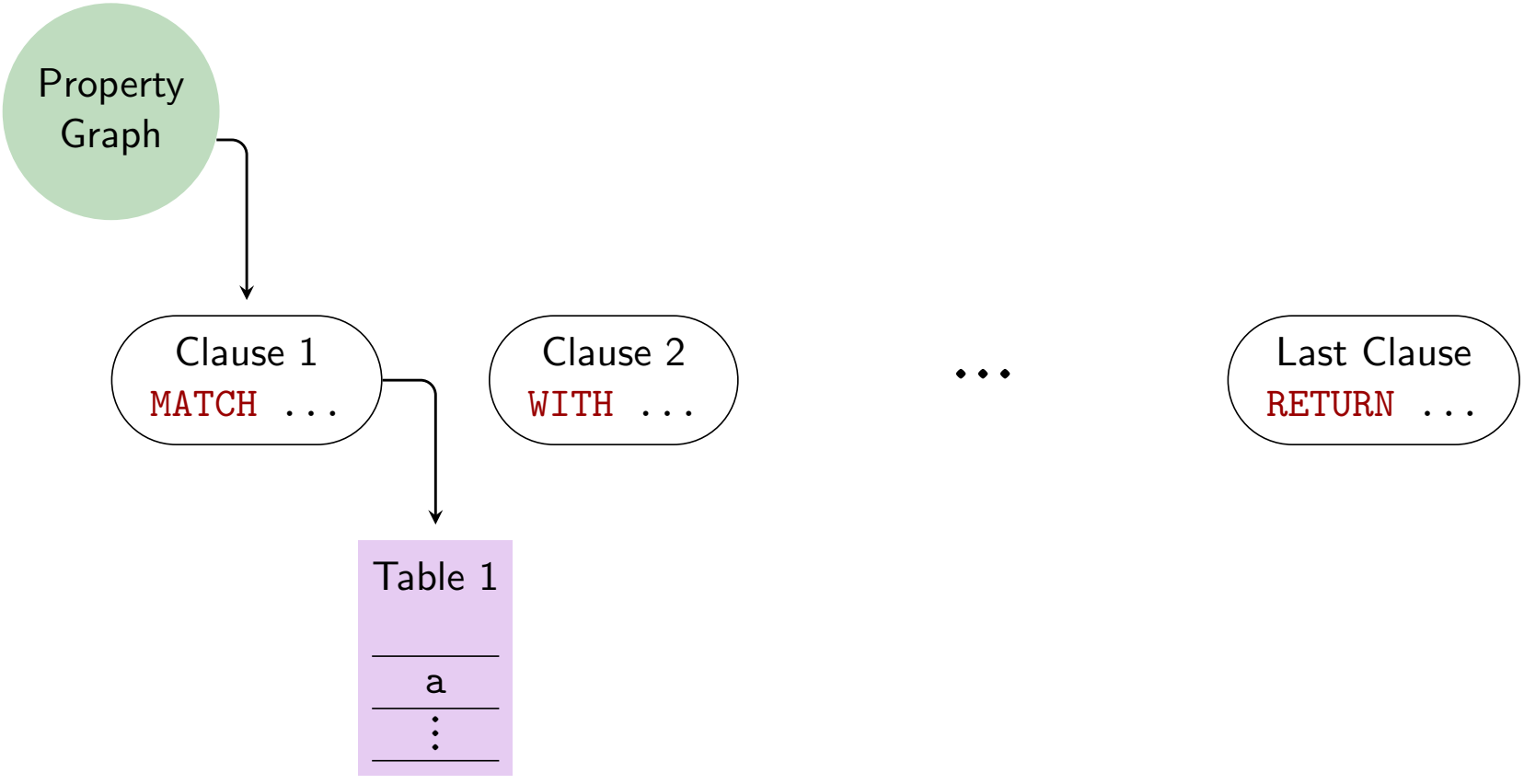
Part III: Cypher

3. Sequence of clauses

How evaluation works

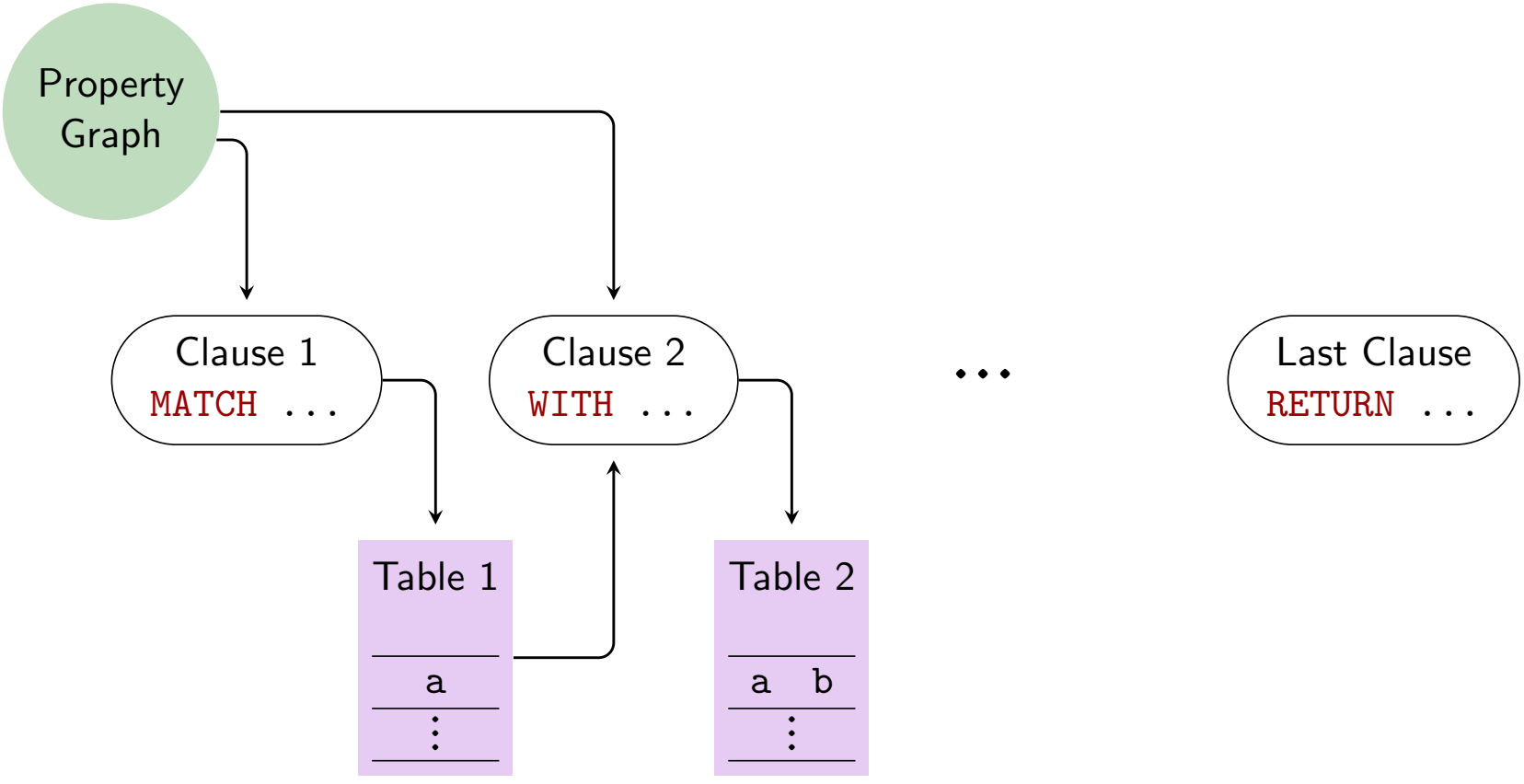


How evaluation works



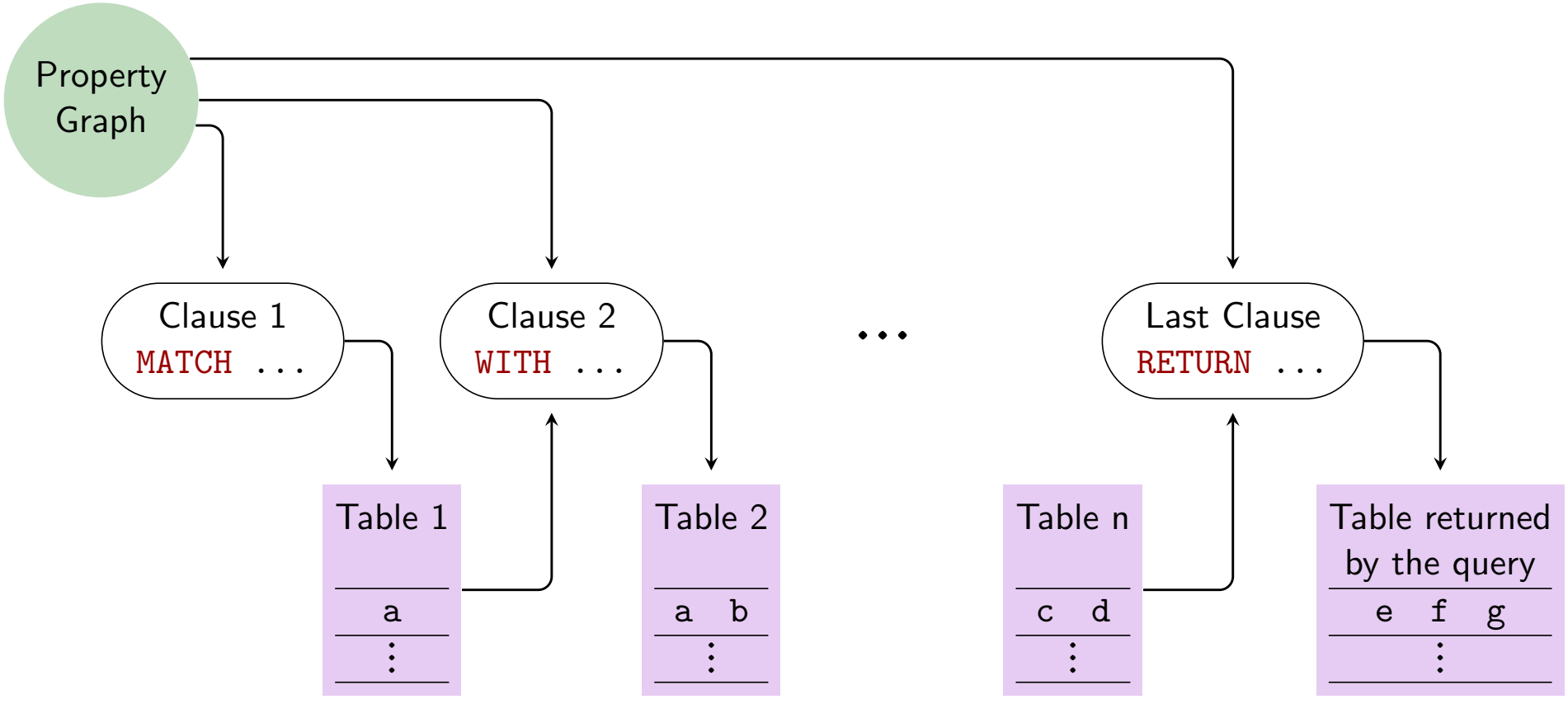
- The first clause produces a table from the property graph

How evaluation works



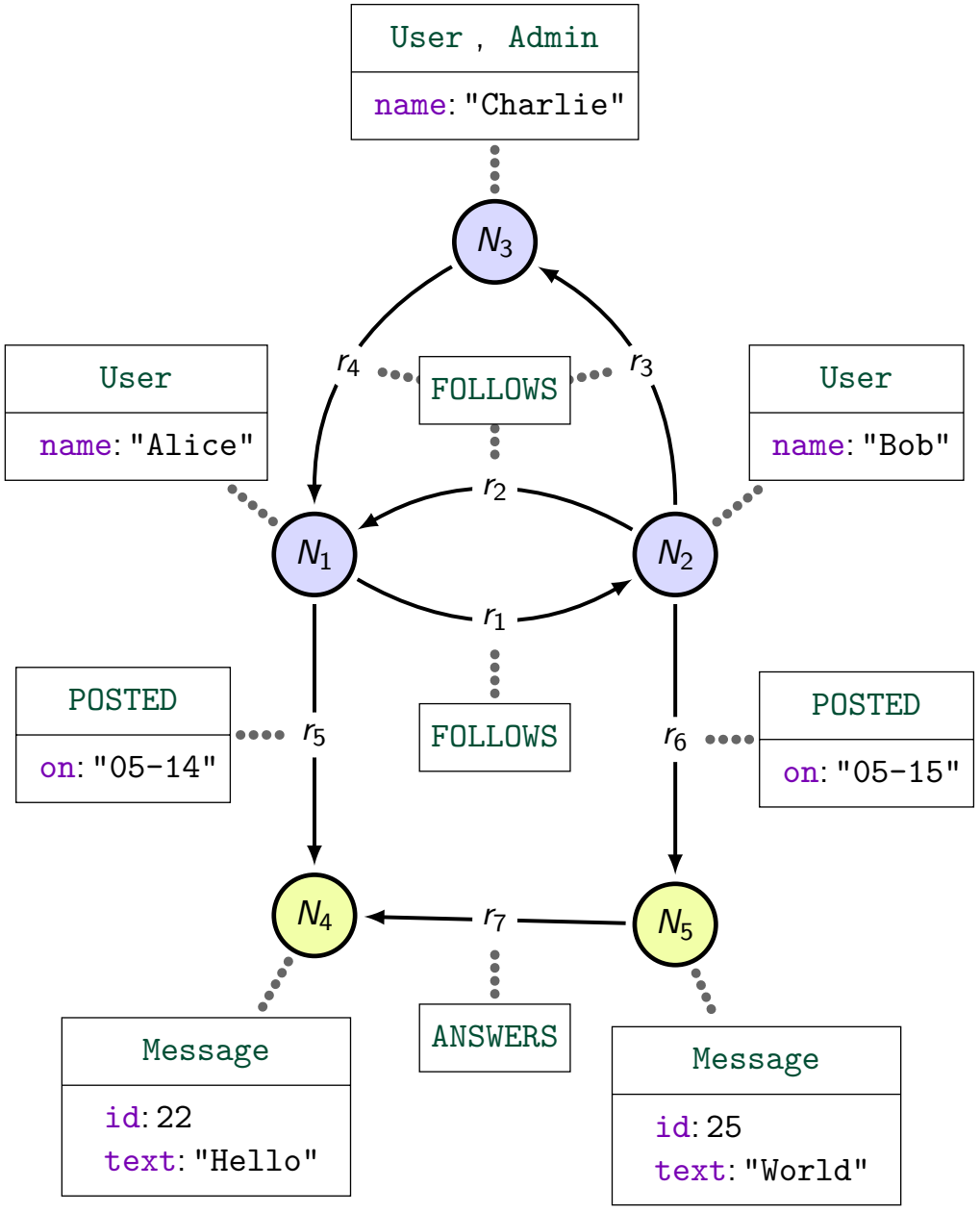
- The first clause produces a table from the property graph
- Subsequent clauses produces a new table from the property graph **and the prior table**

How evaluation works



- The first clause produces a table from the property graph
- Subsequent clauses produces a new table from the property graph **and the prior table**
- Until we reach the last clause, which produces the table to return

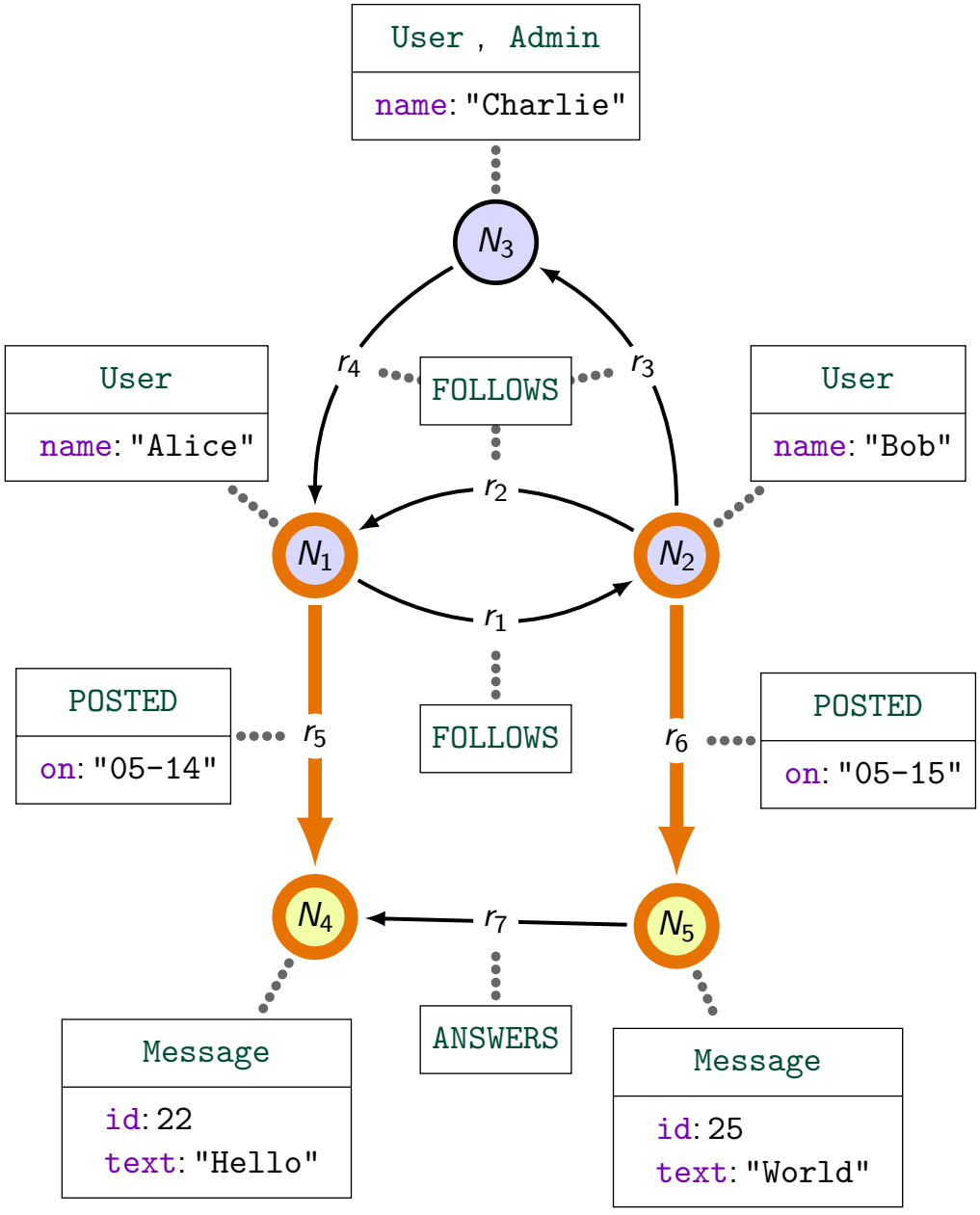
Sequence of MATCH clauses



Query:

```
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
                -[:FOLLOWS]->(u3)
```

Sequence of MATCH clauses



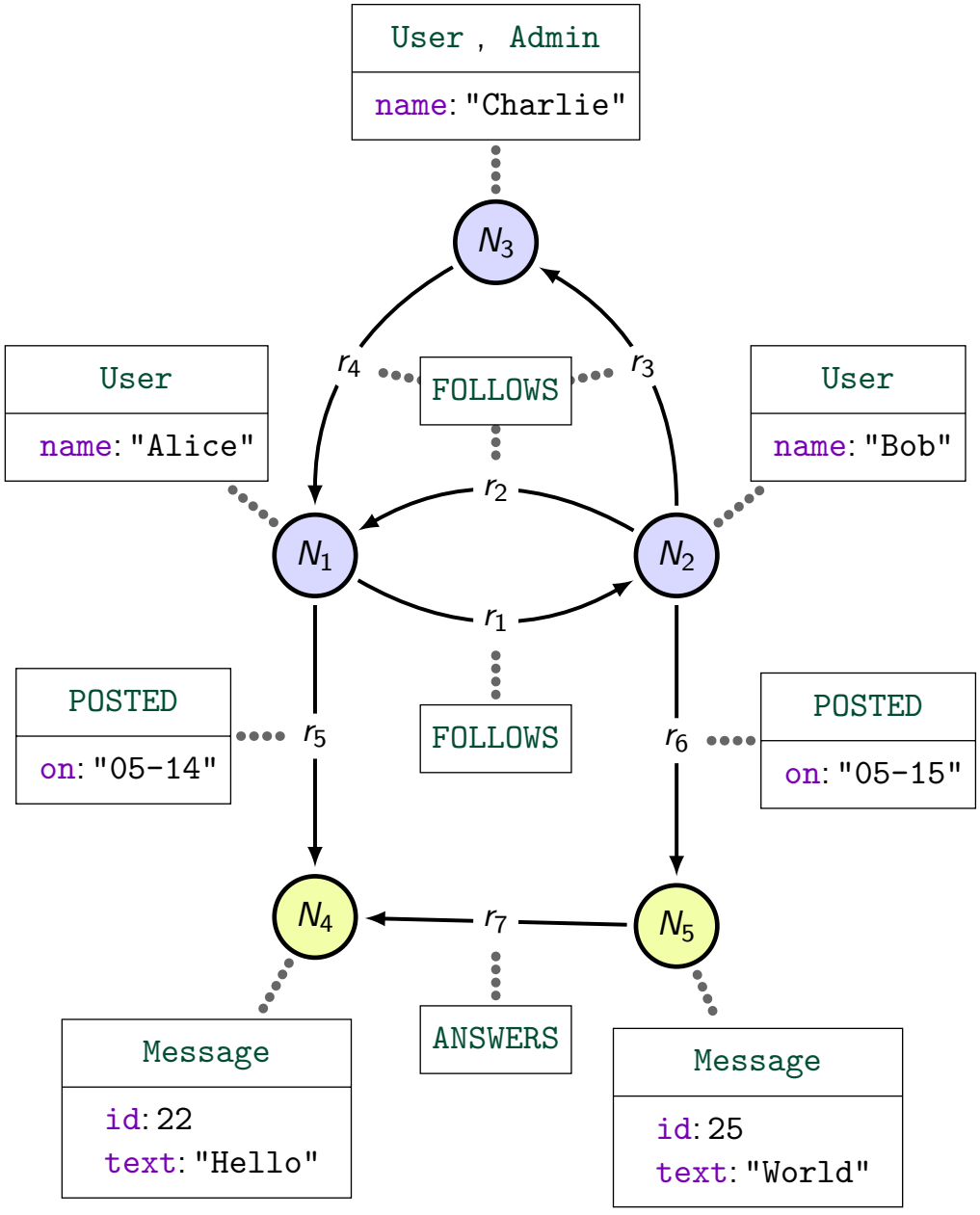
Query:

```
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
                -[:FOLLOWS]->(u3)
```

Table after first MATCH:

u1	m1
N_1	N_4
N_2	N_5

Sequence of MATCH clauses



Query:

```
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
                -[:FOLLOWS]->(u3)
```

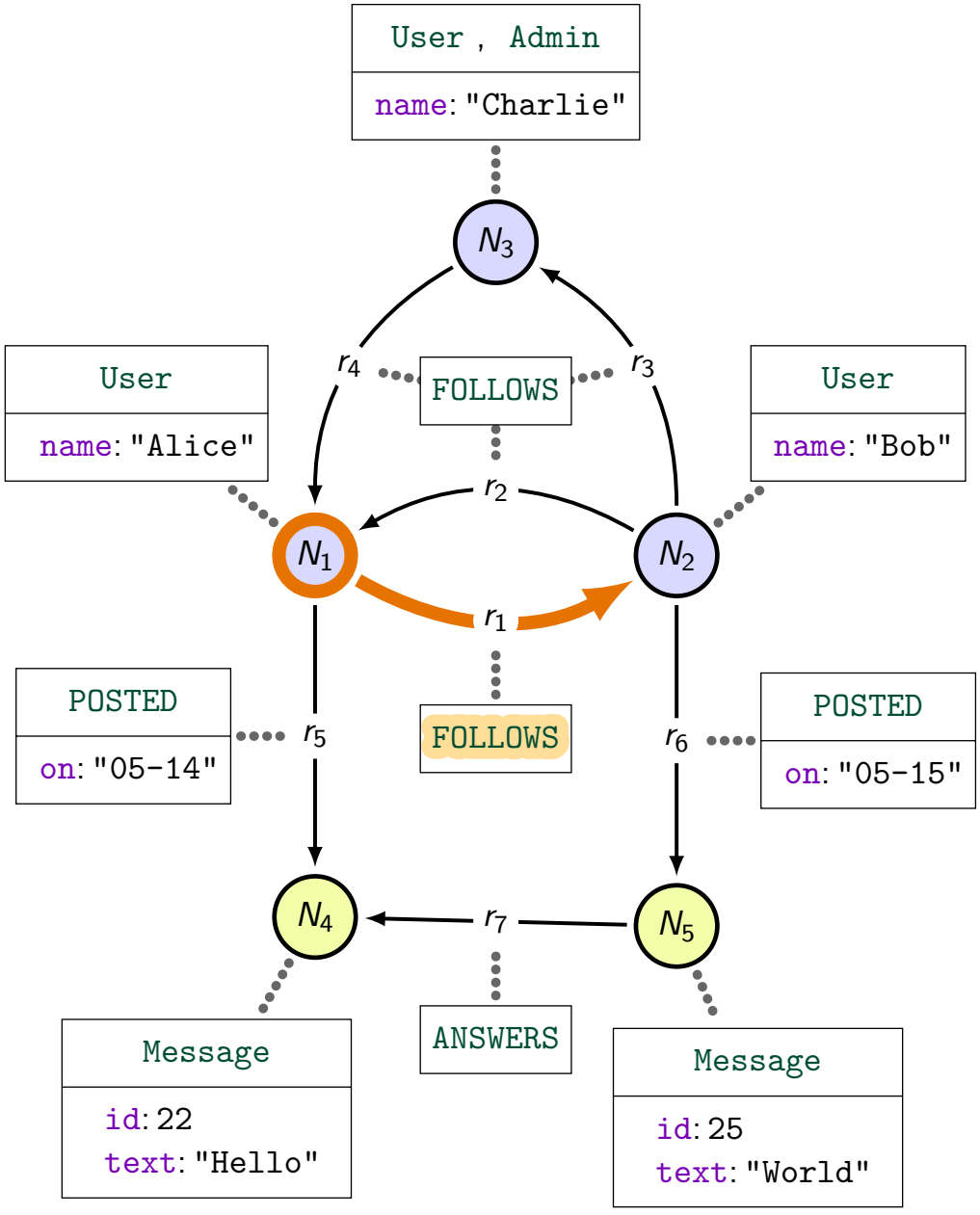
Table after first MATCH:

u1	m1
N_1	N_4
N_2	N_5

Table after second MATCH:

u1	m1	u2	u3
N_1	N_4	·	·
N_2	N_5	·	·

Sequence of MATCH clauses



Query:

```
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
                -[:FOLLOWS]->(u3)
```

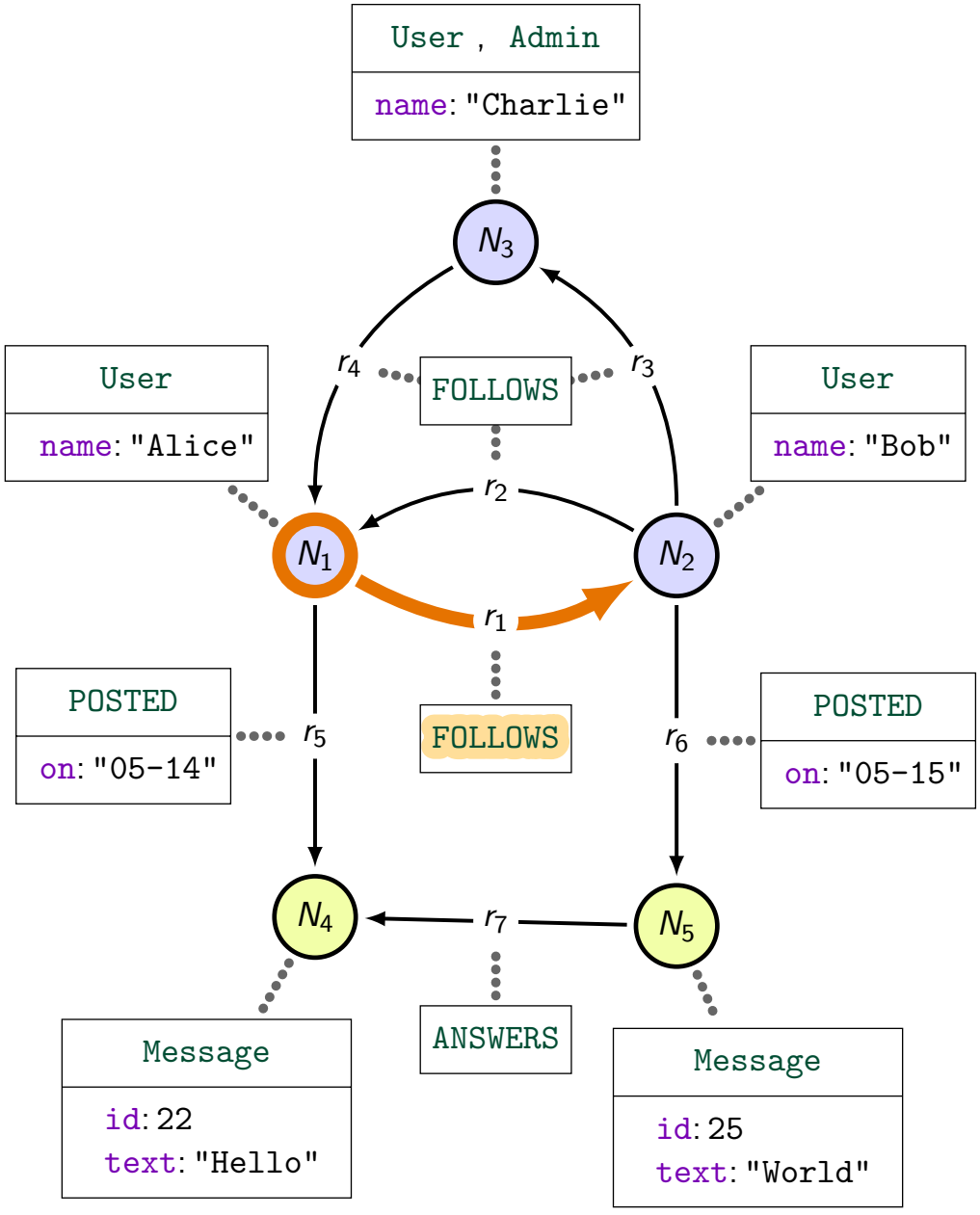
Table after first MATCH:

u1	m1
N_1	N_4
N_2	N_5

Table after second MATCH:

u1	m1	u2	u3
N_1	N_4	·	·
N_2	N_5	·	·

Sequence of MATCH clauses



Query:

```
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
                -[:FOLLOWS]->(u3)
```

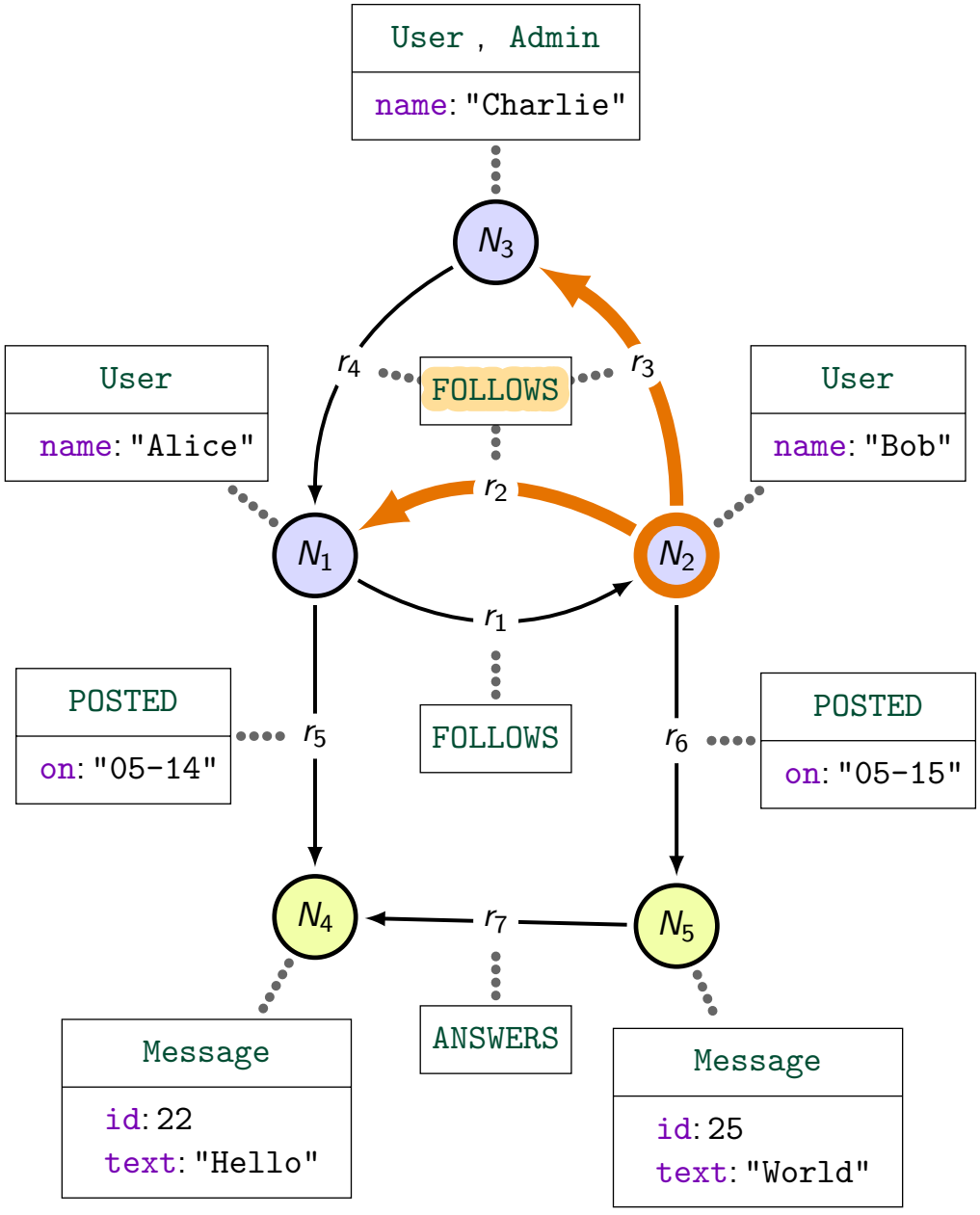
Table after first MATCH:

u1	m1
N_1	N_4
N_2	N_5

Table after second MATCH:

u1	m1	u2	u3
~~N_1~~	~~N_4~~	~~·~~	~~·~~
N_2	N_5	·	·

Sequence of MATCH clauses



Query:

```
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
      -[:FOLLOWS]->(u3)
```

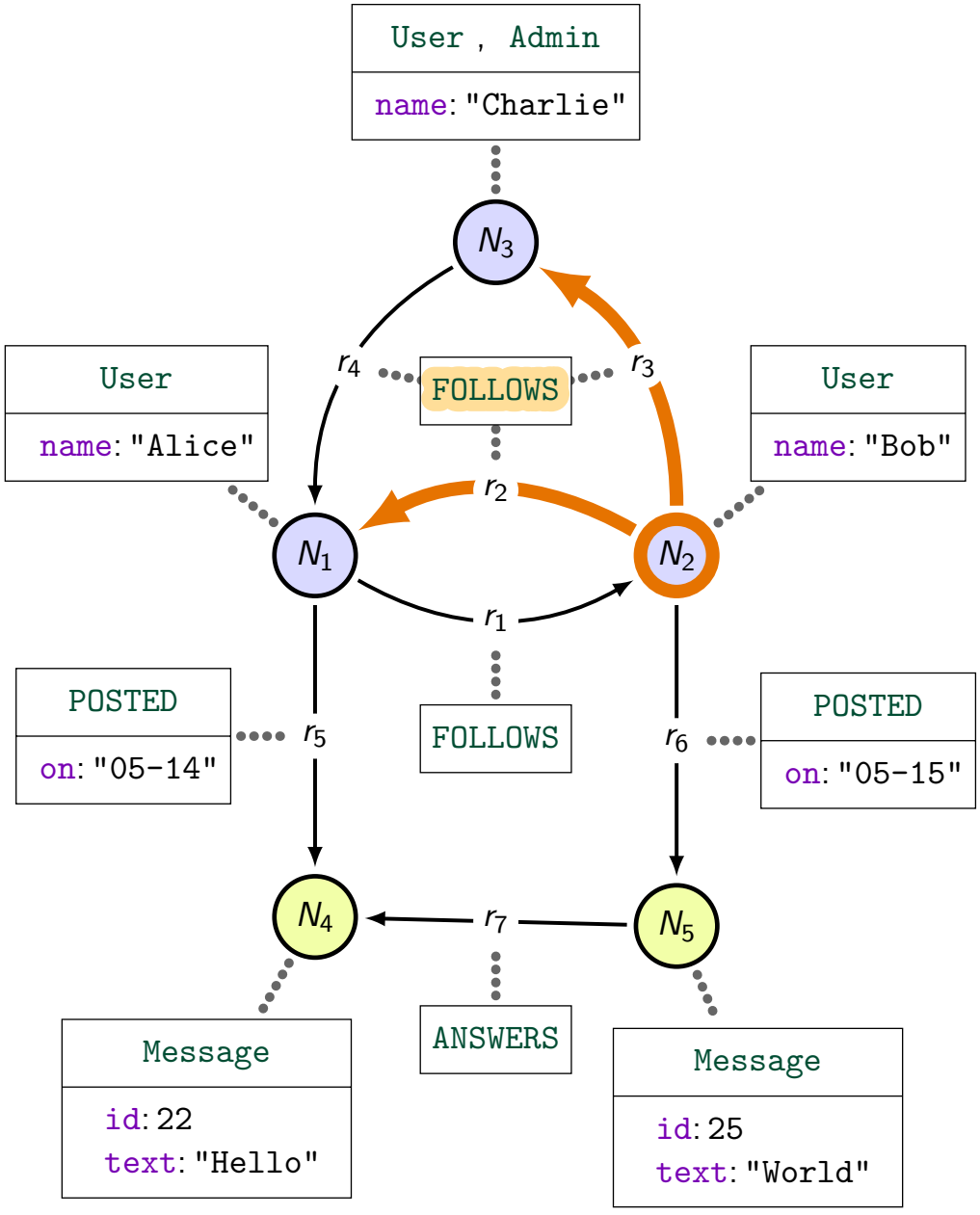
Table after first MATCH:

u1	m1
N_1	N_4
N_2	N_5

Table after second MATCH:

u1	m1	u2	u3
~~N_1~~	~~N_4~~	~~·~~	~~·~~
N_2	N_5	·	·

Sequence of MATCH clauses



Query:

```
MATCH (u1)-[:POSTED]->(m1)
MATCH (u2)<-[:FOLLOWS]-(u1)
                -[:FOLLOWS]->(u3)
```

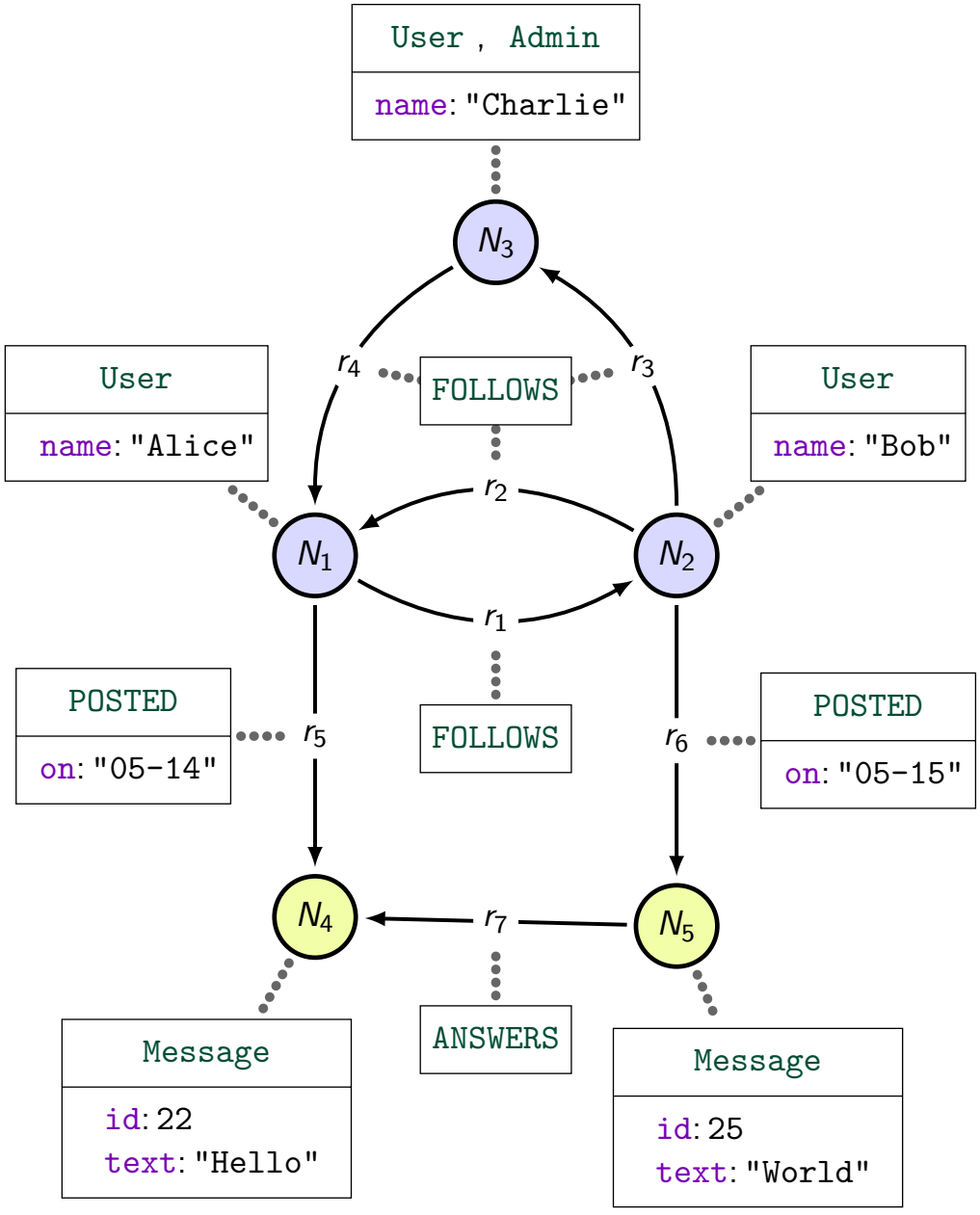
Table after first MATCH:

u1	m1
N_1	N_4
N_2	N_5

Table after second MATCH:

u1	m1	u2	u3
N_2	N_5	N_1	N_3
N_2	N_5	N_3	N_1

Exercice



The two following queries compute similar thing:

```
MATCH (a)⟨pat1⟩(b)⟨pat2⟩(c)
```

```
MATCH (a)⟨pat1⟩(b)
MATCH (b)⟨pat2⟩(c)
```

1 Compute their answer for

$\langle pat_1 \rangle$ = `-[:FOLLOWS]->`

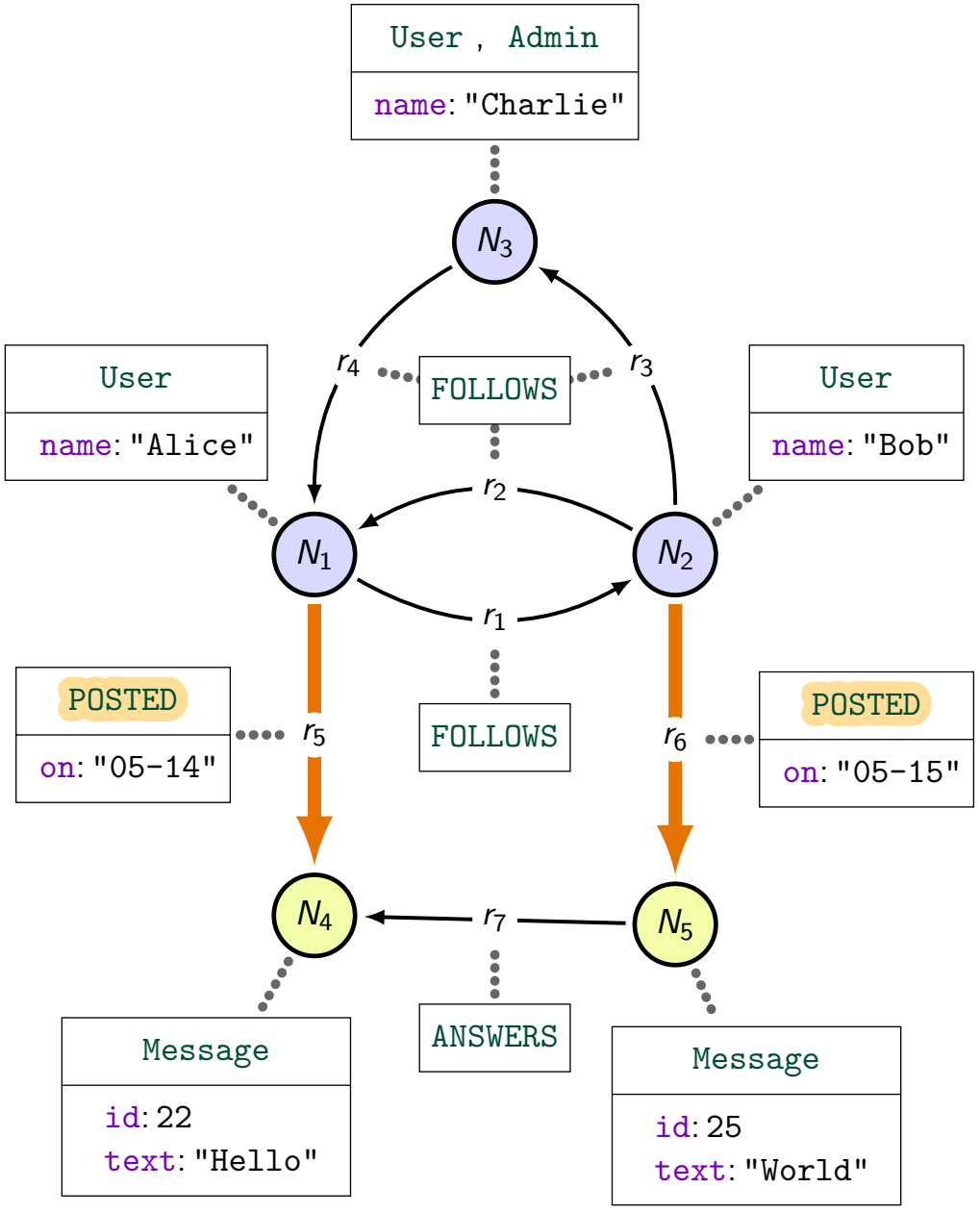
$\langle pat_2 \rangle$ = `-[:POSTED]->`

2 Can you find patterns $\langle pat_1 \rangle$ and $\langle pat_2 \rangle$ for which their answer is different?

Part III: Cypher

4. Usage of `WITH` (or `RETURN`)

Column manipulation



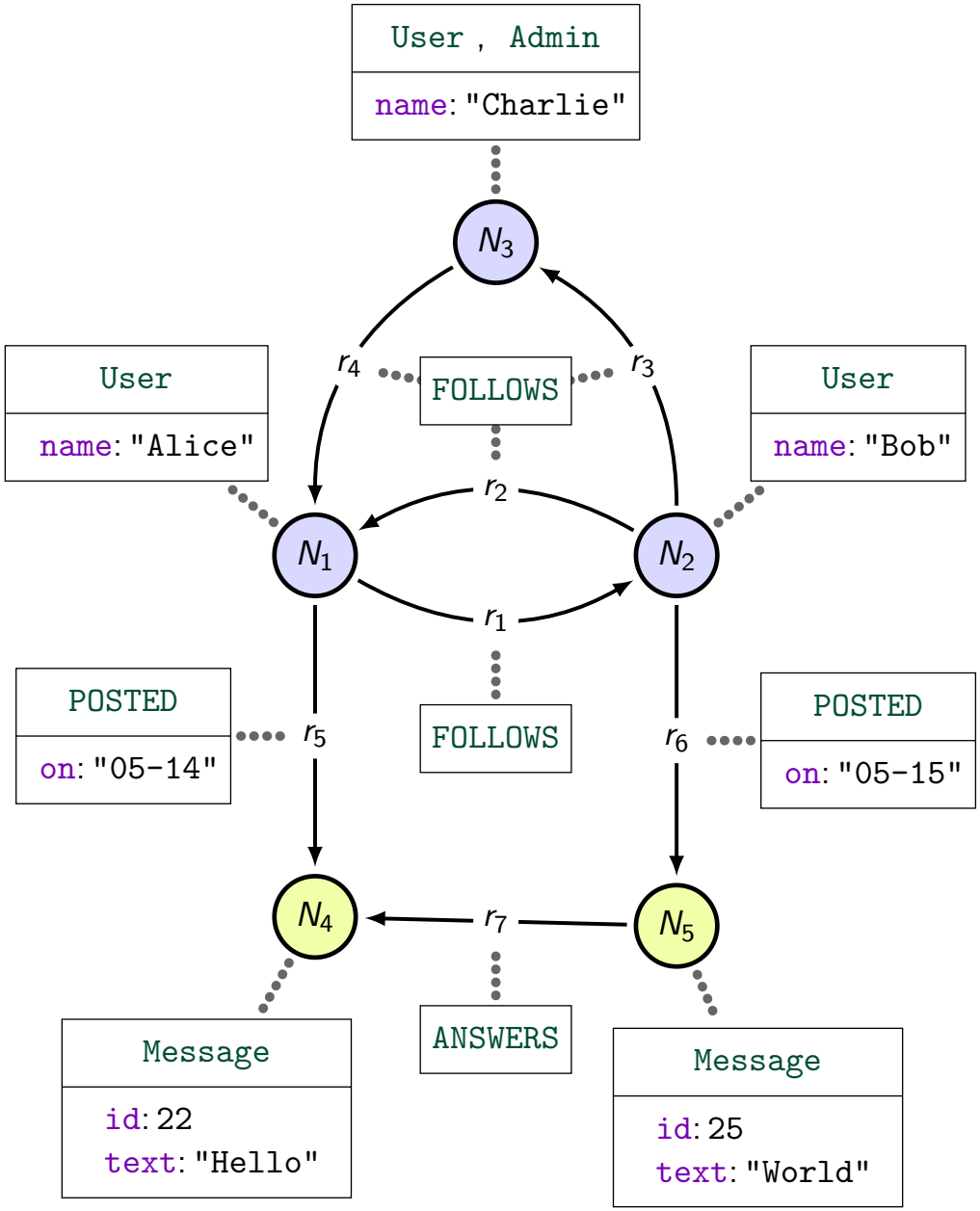
Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1
```

After the MATCH clause

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Column manipulation



Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1
```

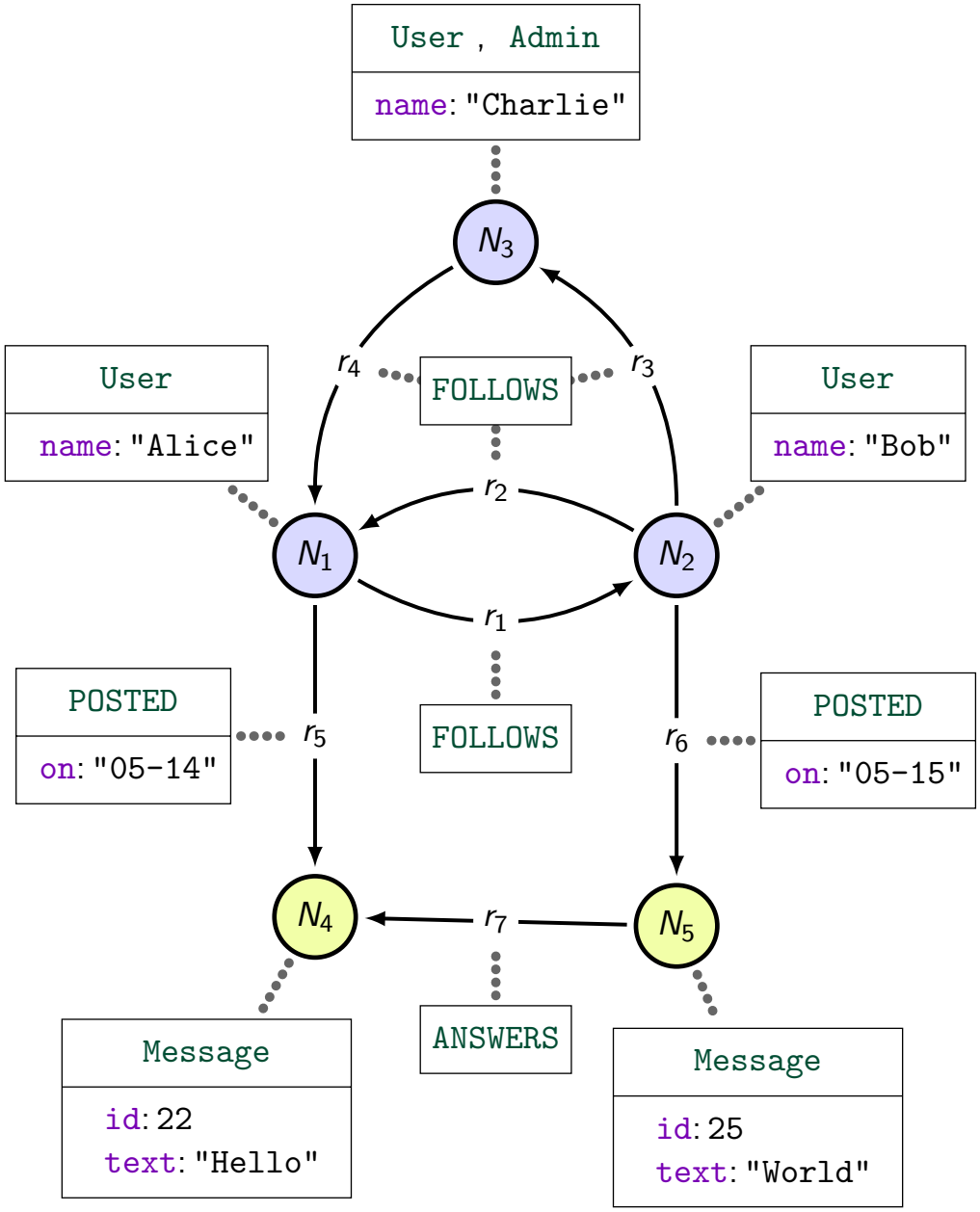
After the MATCH clause

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Execution of the WITH clause

u1	p1	t1
N_1	r_5	
N_2	r_6	

Column manipulation



Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1
```

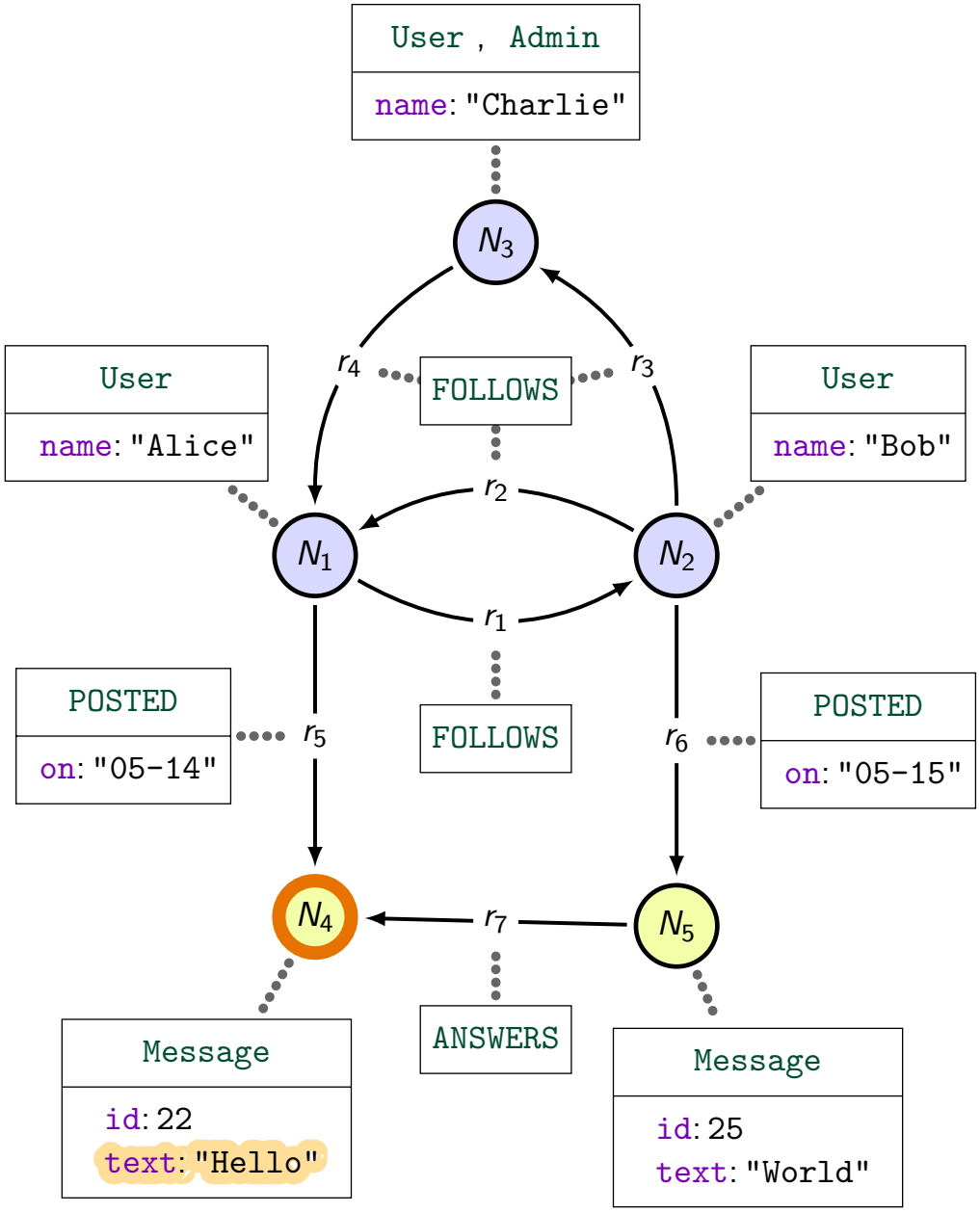
After the MATCH clause

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Execution of the WITH clause

u1	p1	t1
N_1	r_5	
N_2	r_6	

Column manipulation



Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1
```

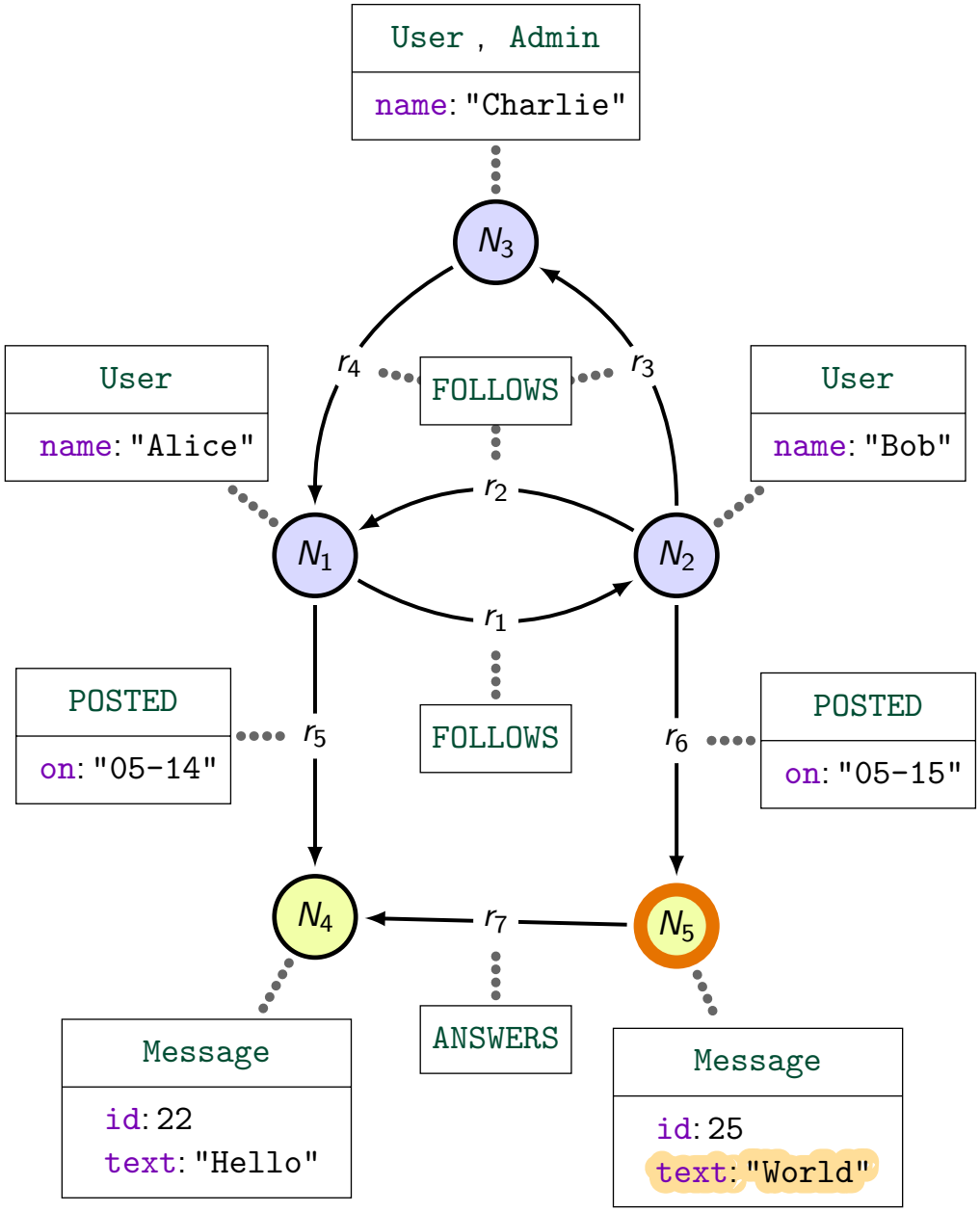
After the MATCH clause

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Execution of the WITH clause

u1	p1	t1
N_1	r_5	"Hello"
N_2	r_6	

Column manipulation



Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1
```

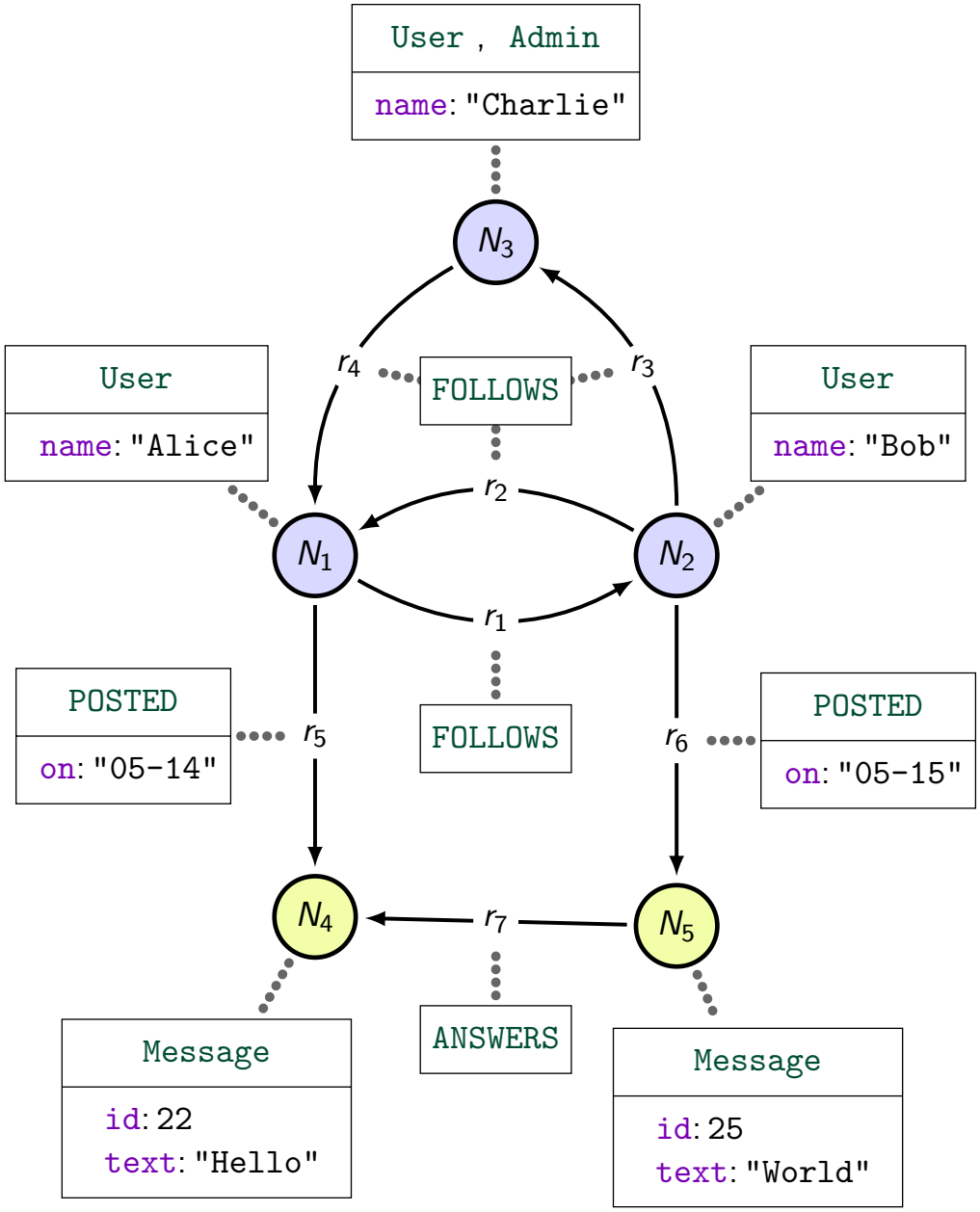
After the MATCH clause

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Execution of the WITH clause

u1	p1	t1
N_1	r_5	"Hello"
N_2	r_6	"World"

Column manipulation



Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WITH u1, p1, m1.text AS t1
```

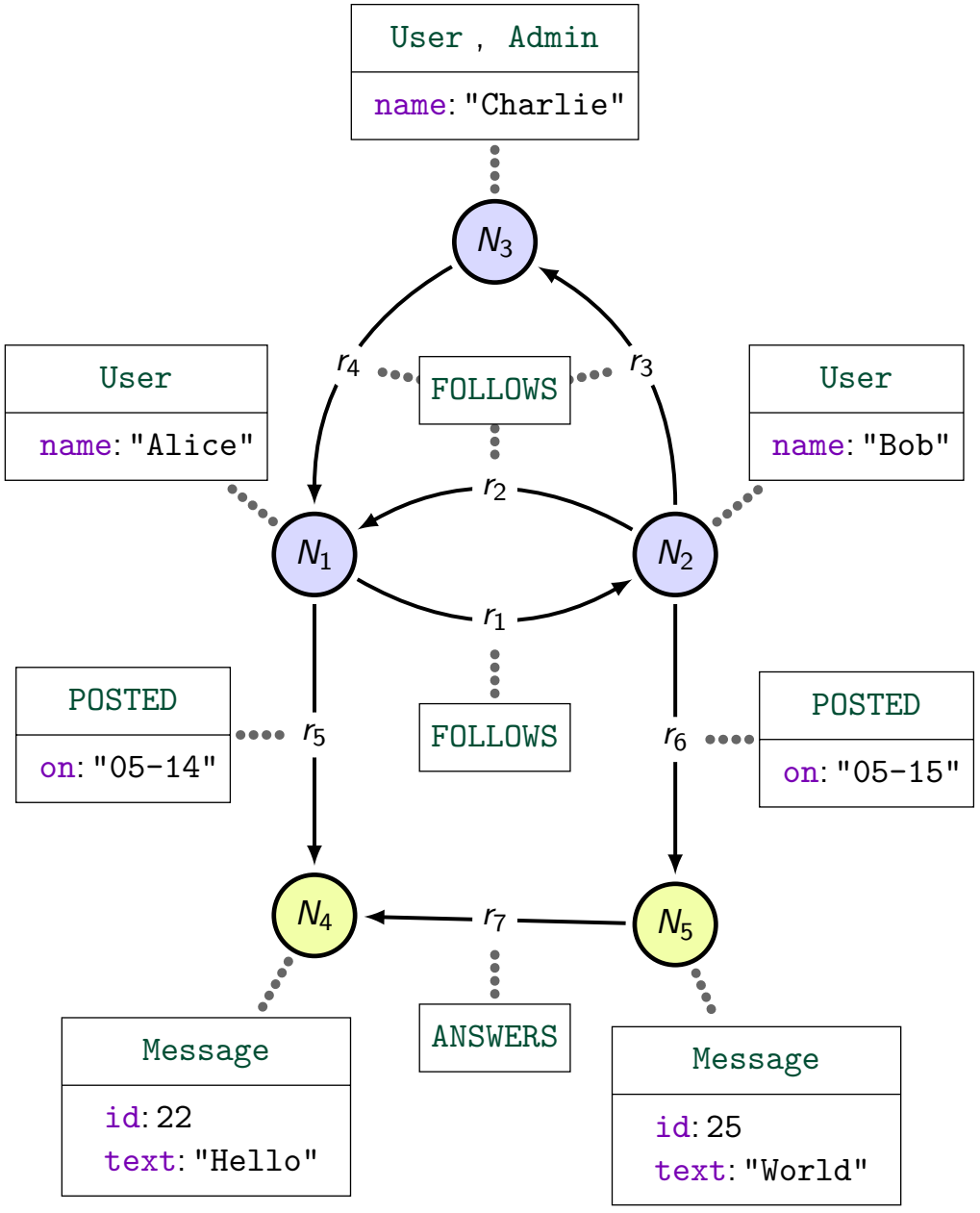
After the MATCH clause

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

After WITH:

u1	p1	t1
N_1	r_5	"Hello"
N_2	r_6	"World"

Elimination of duplicate rows



Query:

```
MATCH (u1)-[:FOLLOWS]->()
WITH DISTINCT u1
```

After MATCH:

u1
N_1
N_2
N_2
N_3

After WITH:

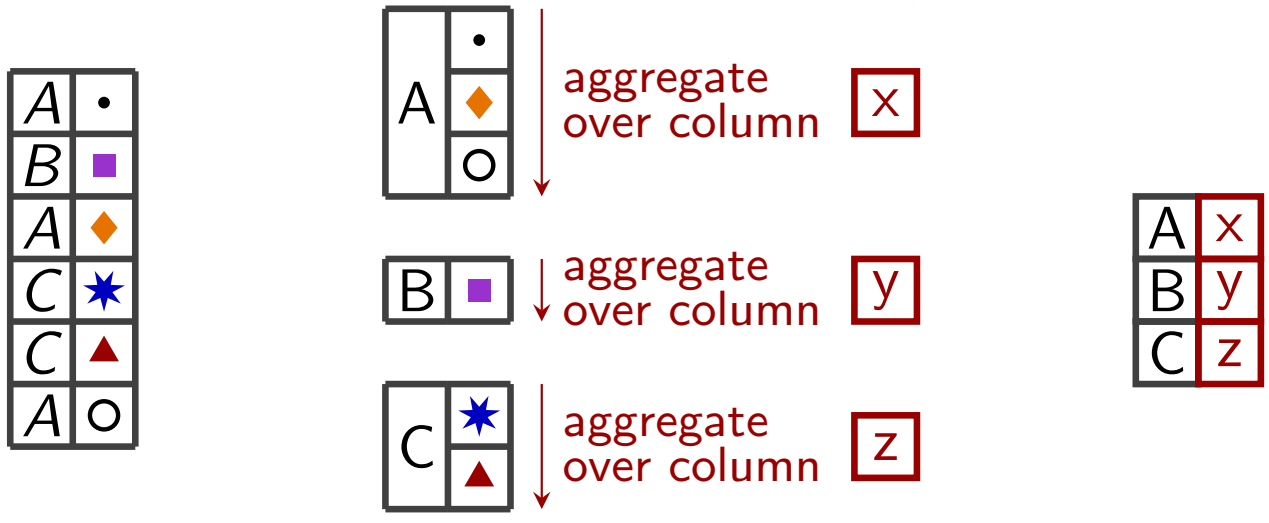
u1
N_1
N_2
N_3

Aggregation: horizontal versus vertical

- Aggregation = Compute one value from a list/set of value
  Ex: sum, count, max, collect

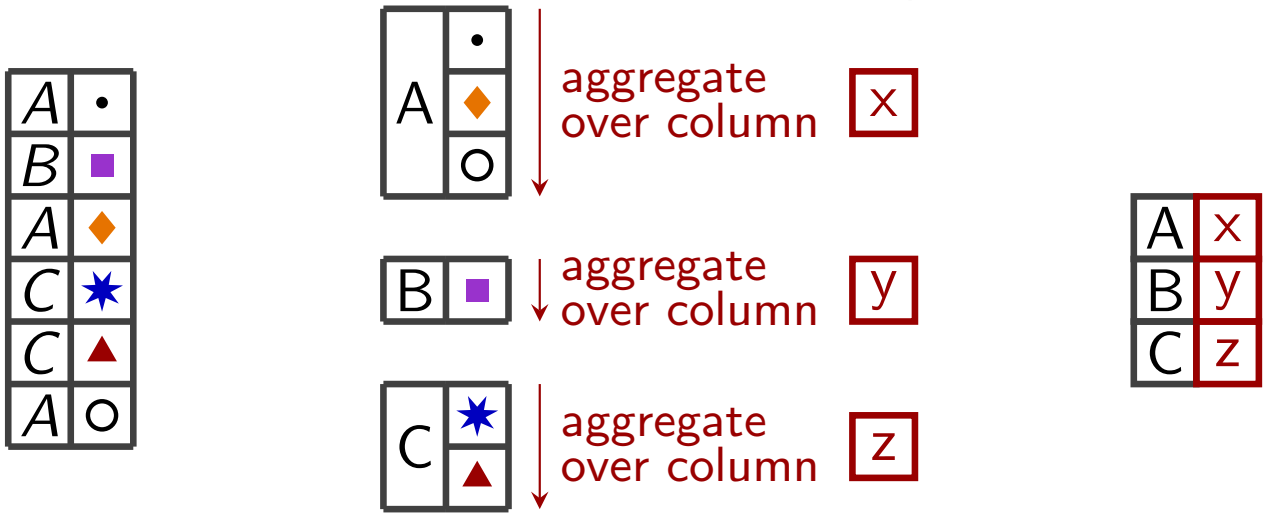
Aggregation: horizontal versus vertical

- Aggregation = Compute one value from a list/set of value
  Ex: sum, count, max, collect
- Vertical aggregation = usual aggregation (`GROUP BY` in SQL)

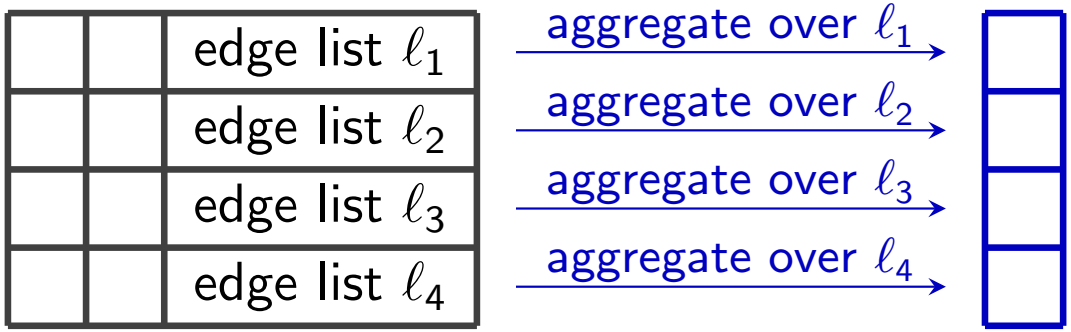


Aggregation: horizontal versus vertical

- Aggregation = Compute one value from a list/set of value
  Ex: sum, count, max, collect
- Vertical aggregation = usual aggregation (`GROUP BY` in SQL)



- Horizontal aggregation = aggregate over each matched paths



Vertical Aggregation

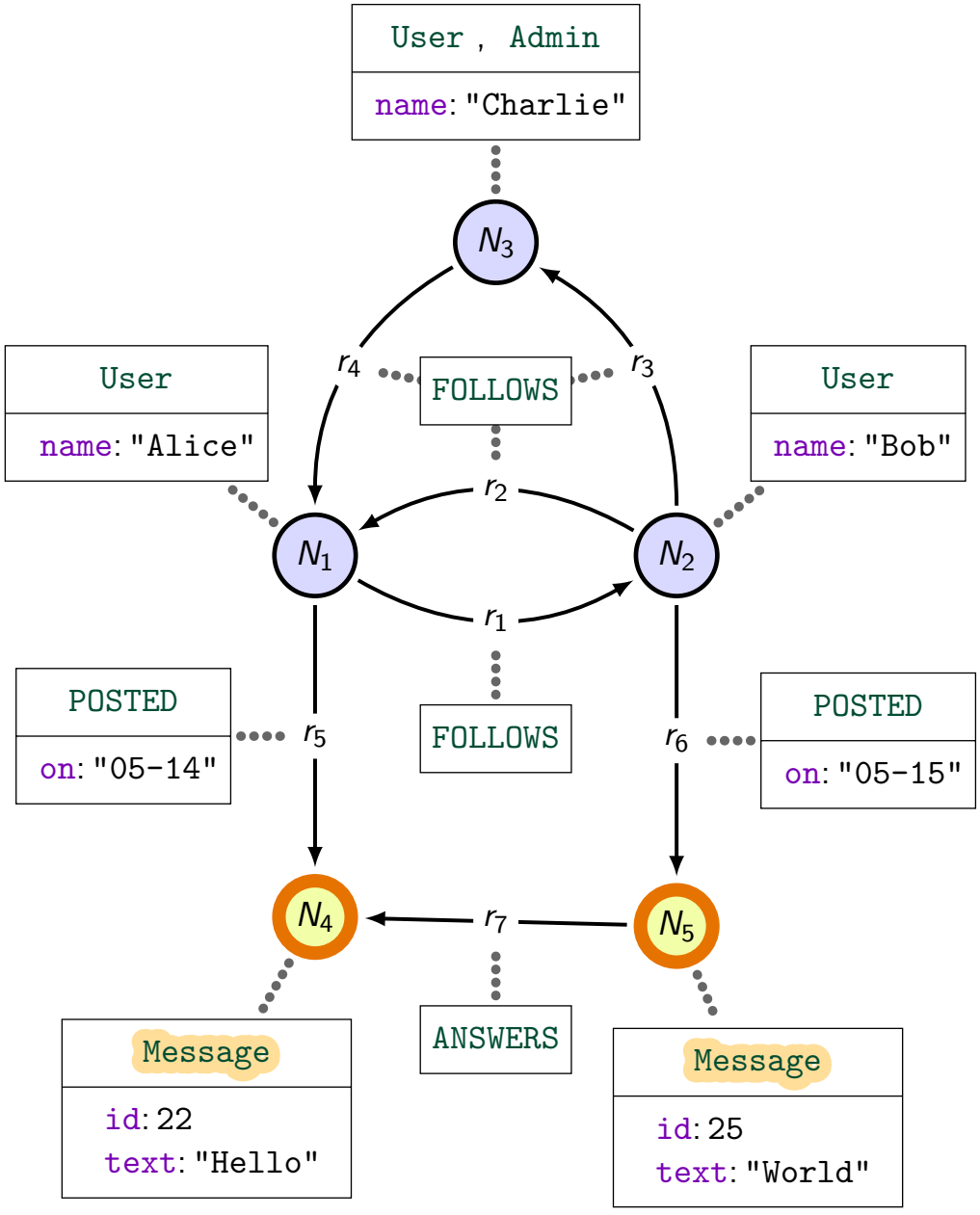
`WITH` ⟨*columns*⟩, ⟨*aggr*⟩(⟨*expr*⟩)

Grouping is implicit: every variable used in ⟨*columns*⟩ is used for grouping

⟨*aggr*⟩ is a built-in **aggregation function**, that is, a function from list to a single value.

Example: `count`, `sum`, `min`, `collect`, etc.

Counting the Message nodes



Query:

```
MATCH (m1:Message)
WITH count(m1) AS c
```

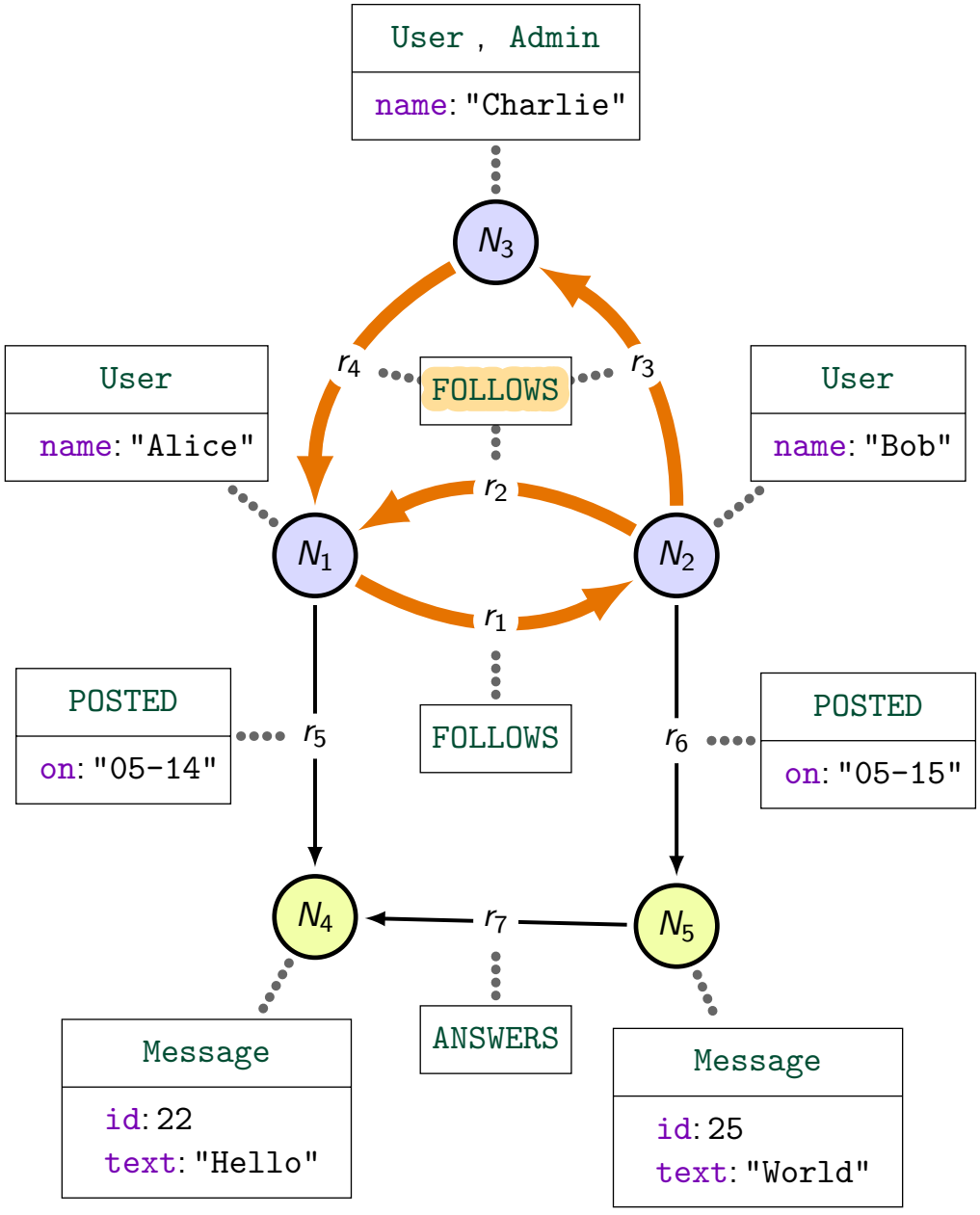
After MATCH:

m1
N_4
N_5

After WITH:

c
2

Collecting names of followers



Query:

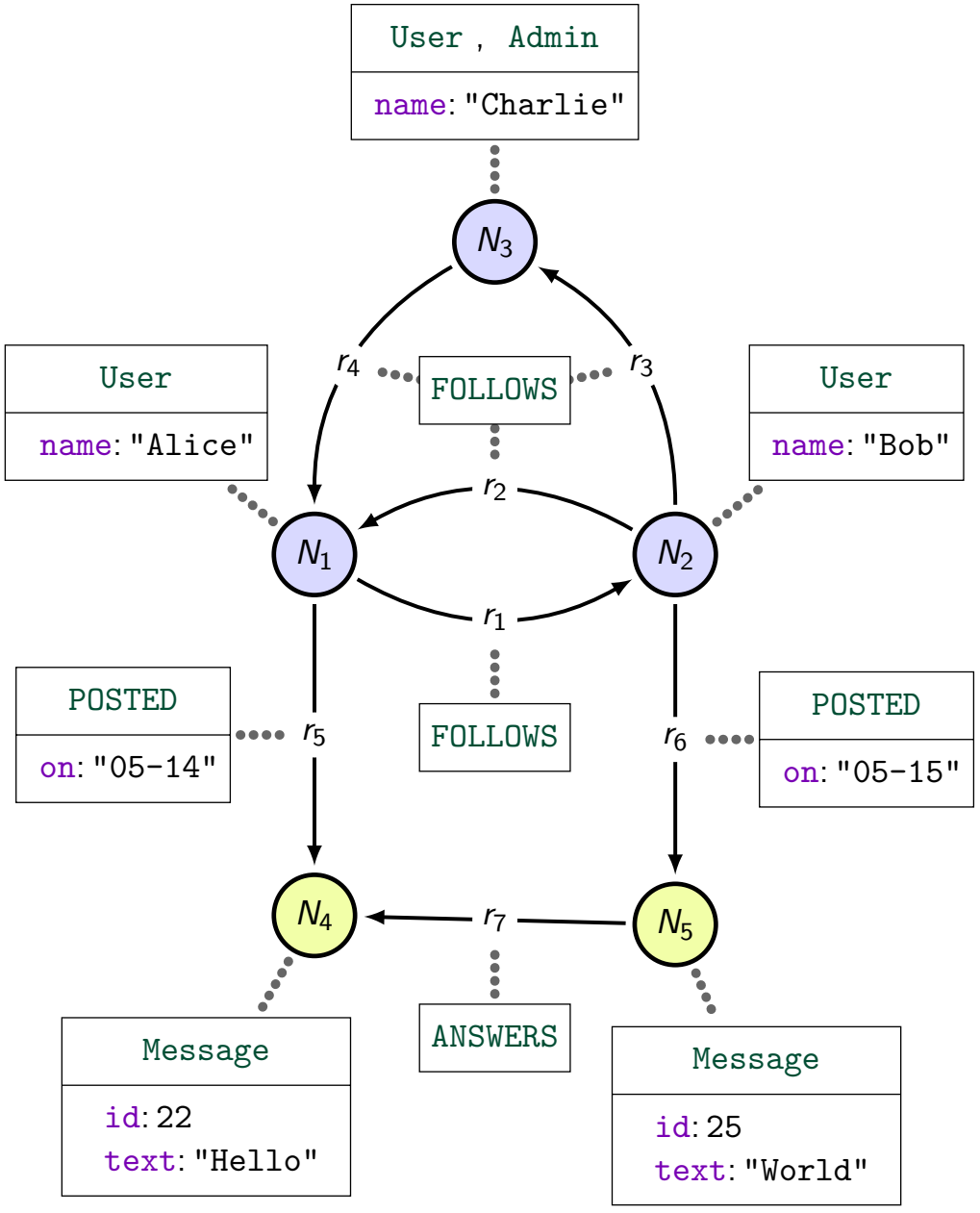
```
MATCH (u1)<-[:FOLLOWS]-(u2)
WITH u1, collect(u2.name) AS n
```

Result after WITH:

u1	n
N_1	["Bob","Charlie"]
N_2	["Alice"]
N_3	["Bob"]

⚠ Grouping by u1 ⚠

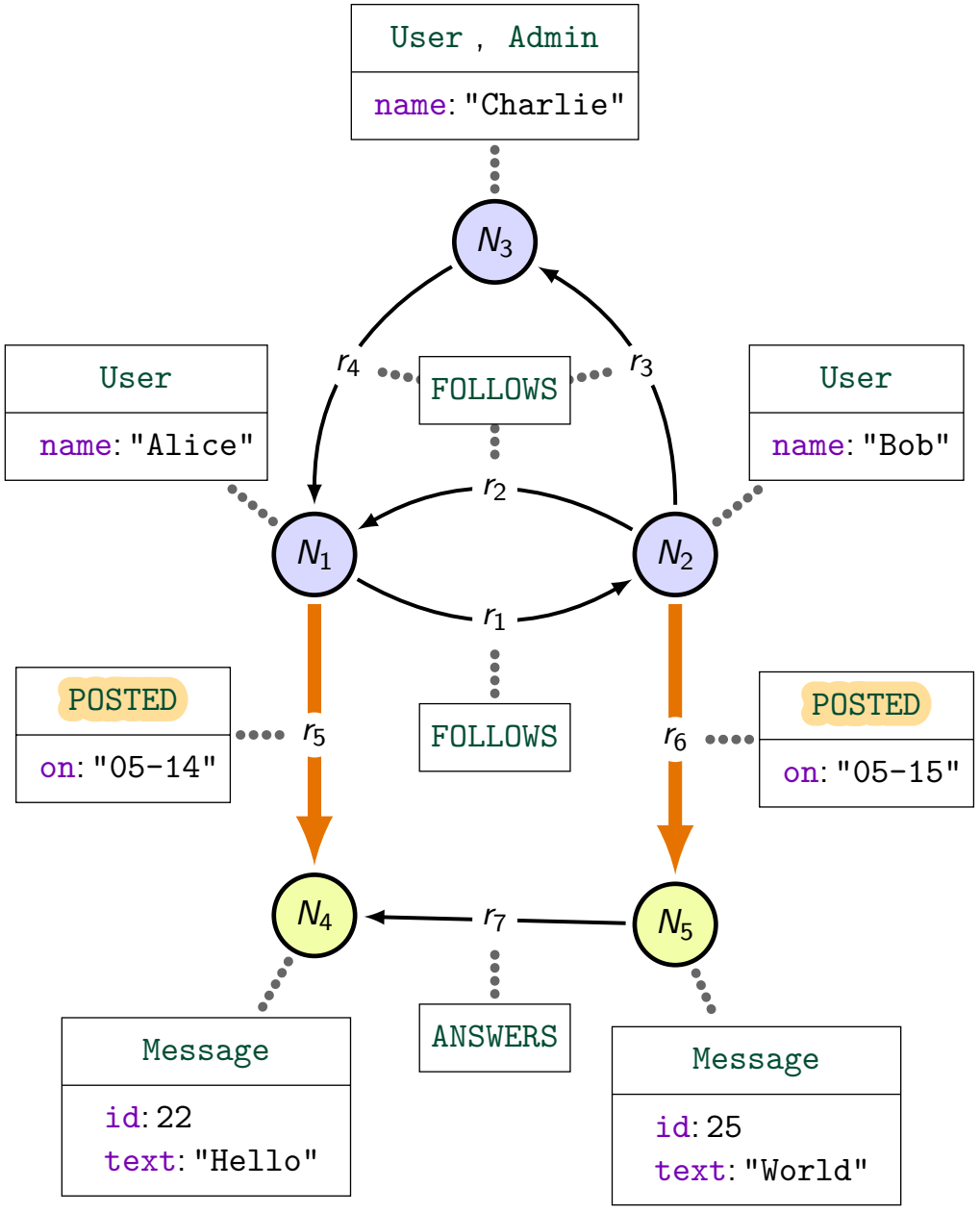
Exercice: what does this compute?



Query:

```
MATCH ()-[e:POSTED]->()
WITH max(e.on) AS d
MATCH ()-[:POSTED
                    {on:d}]->(m1)
WITH m1.text as txt
```

Exercice: what does this compute?

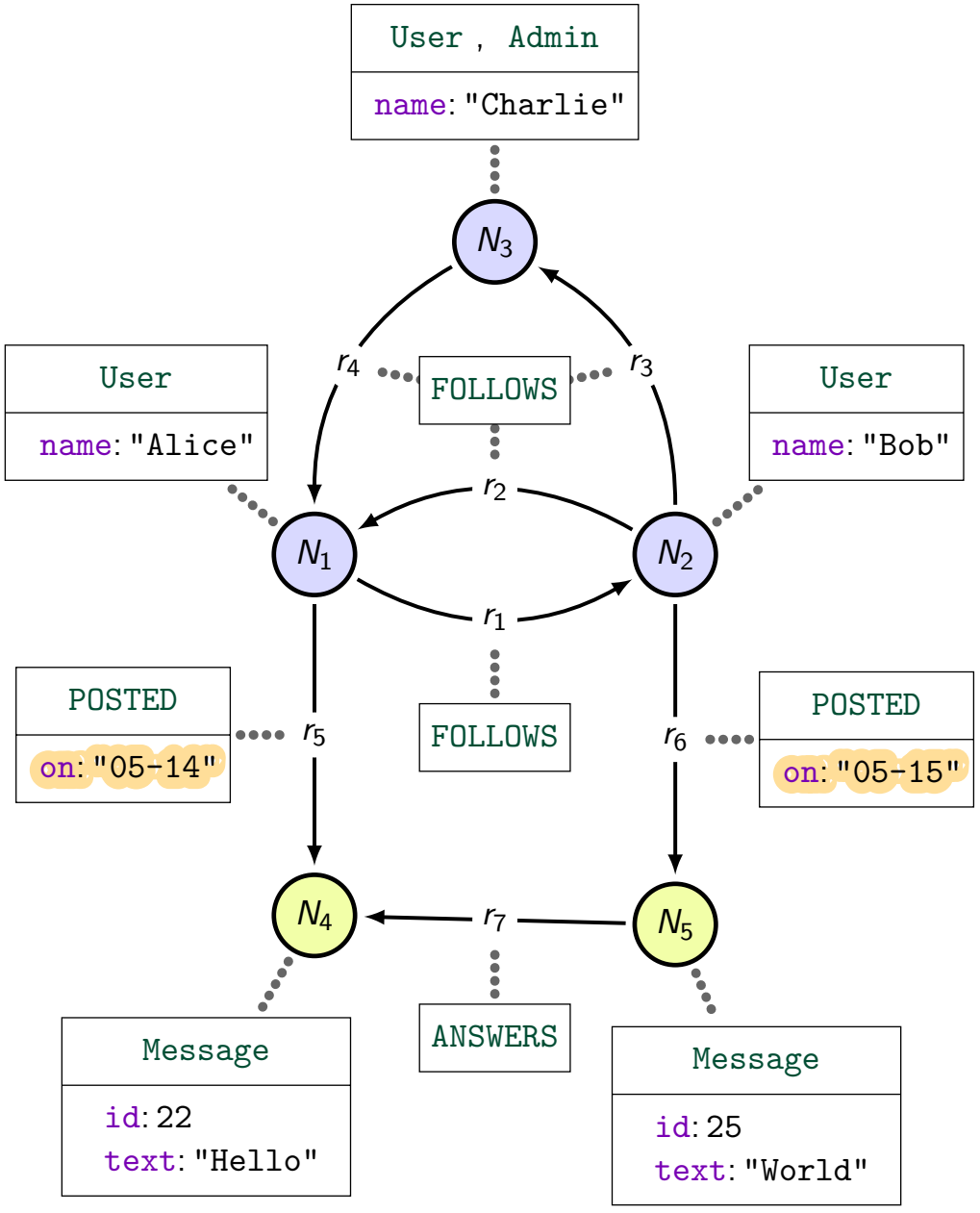


Query:

```
MATCH ()-[e:POSTED]->()
WITH max(e.on) AS d
MATCH ()-[:POSTED
                    {on:d}]->(m1)
WITH m1.text as txt
```

e
r_5
r_6

Exercice: what does this compute?



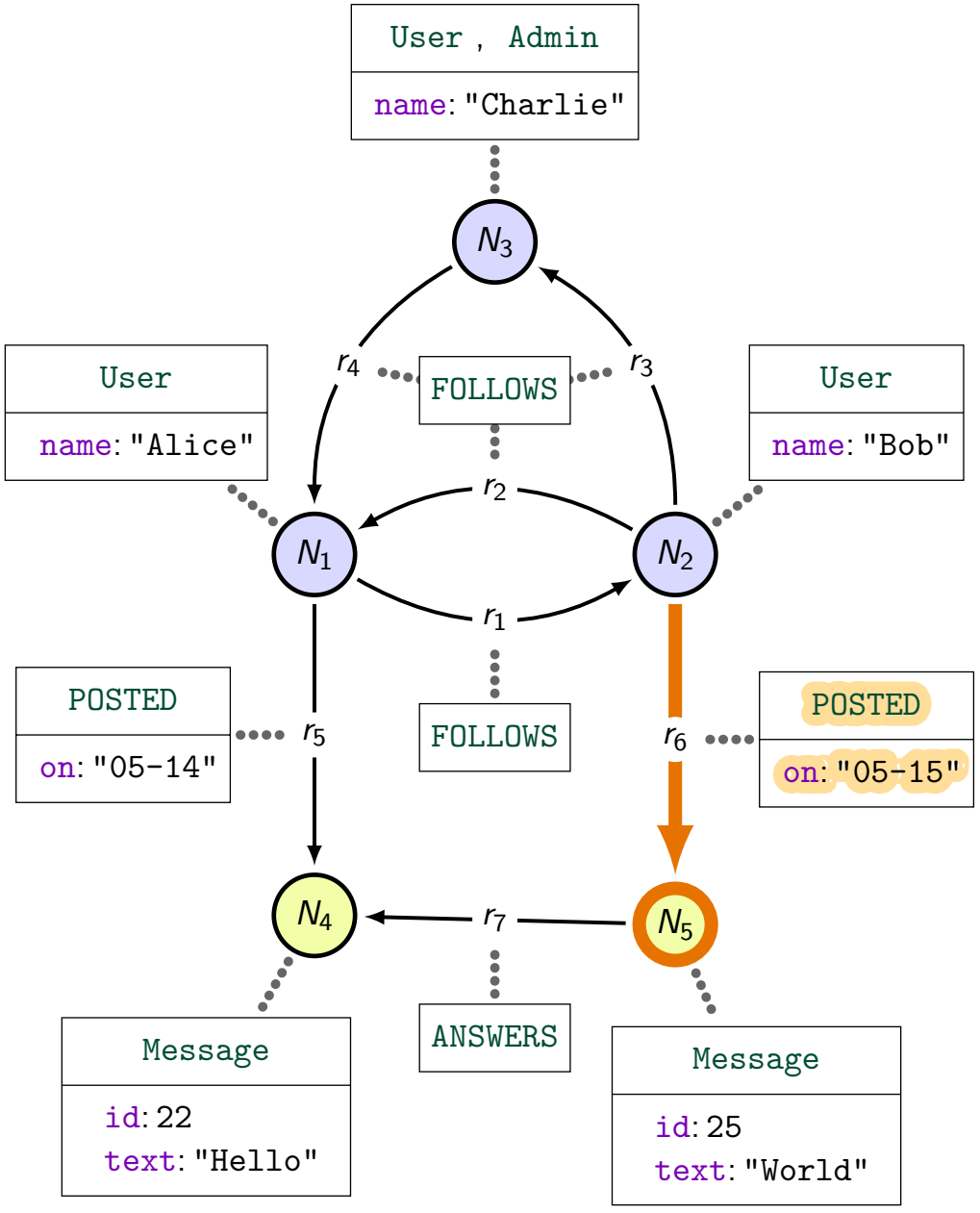
Query:

```
MATCH ()-[e:POSTED]->()
WITH max(e.on) AS d
MATCH ()-[:POSTED
                    {on:d}]->(m1)
WITH m1.text as txt
```

e
r_5
r_6

d
"05-15"

Exercice: what does this compute?



Query:

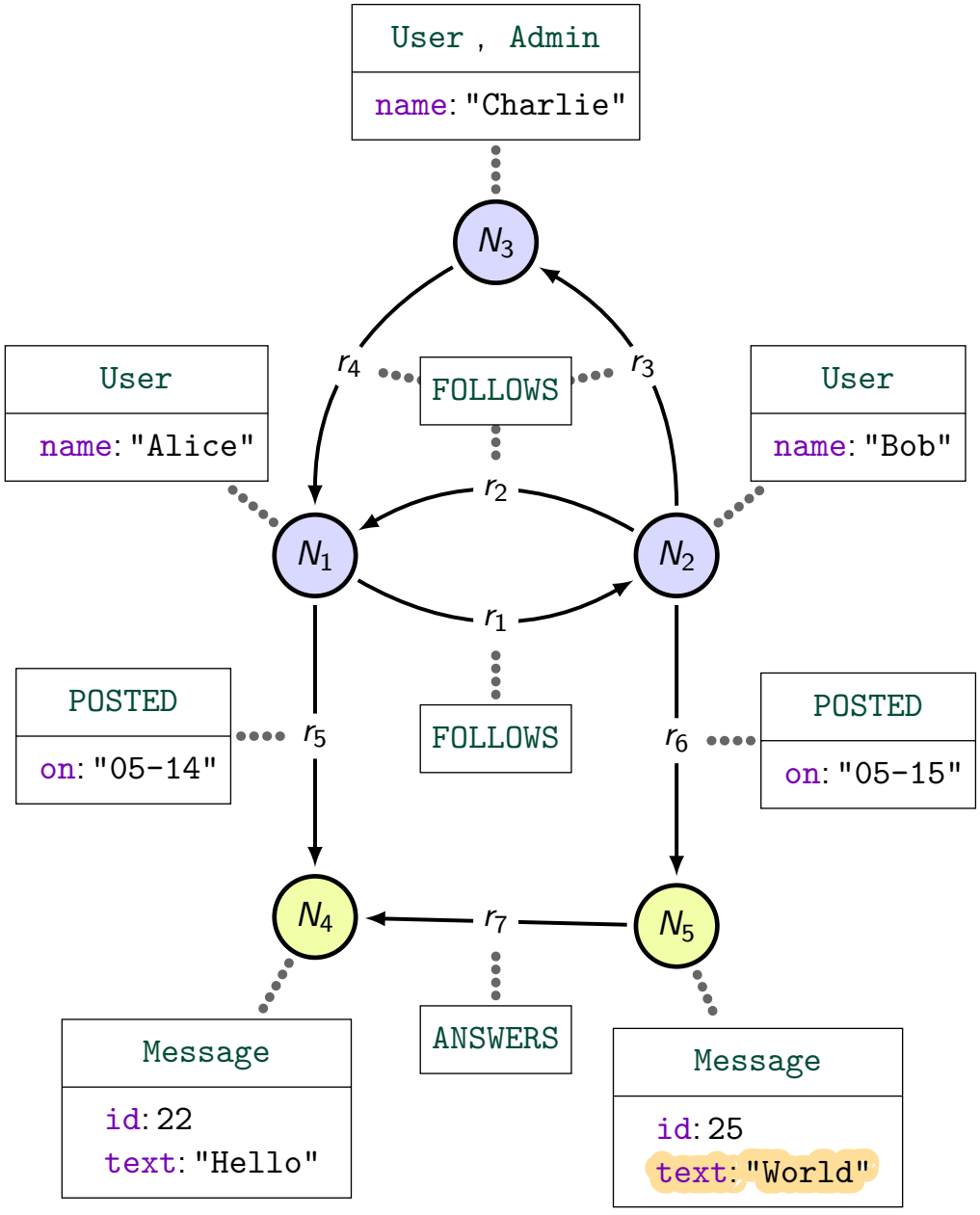
```
MATCH ()-[e:POSTED]->()
WITH max(e.on) AS d
MATCH ()-[:POSTED
                {on:d}]->(m1)
WITH m1.text as txt
```

e
r_5
r_6

d
"05-15"

d	m1
"05-15"	N_5

Exercice: what does this compute?



Query:

```
MATCH ()-[e:POSTED]->()
WITH max(e.on) AS d
MATCH ()-[:POSTED
                    {on:d}]->(m1)
WITH m1.text as txt
```

e
r_5
r_6

d
"05-15"

d	m1
"05-15"	N_5

txt
"World"

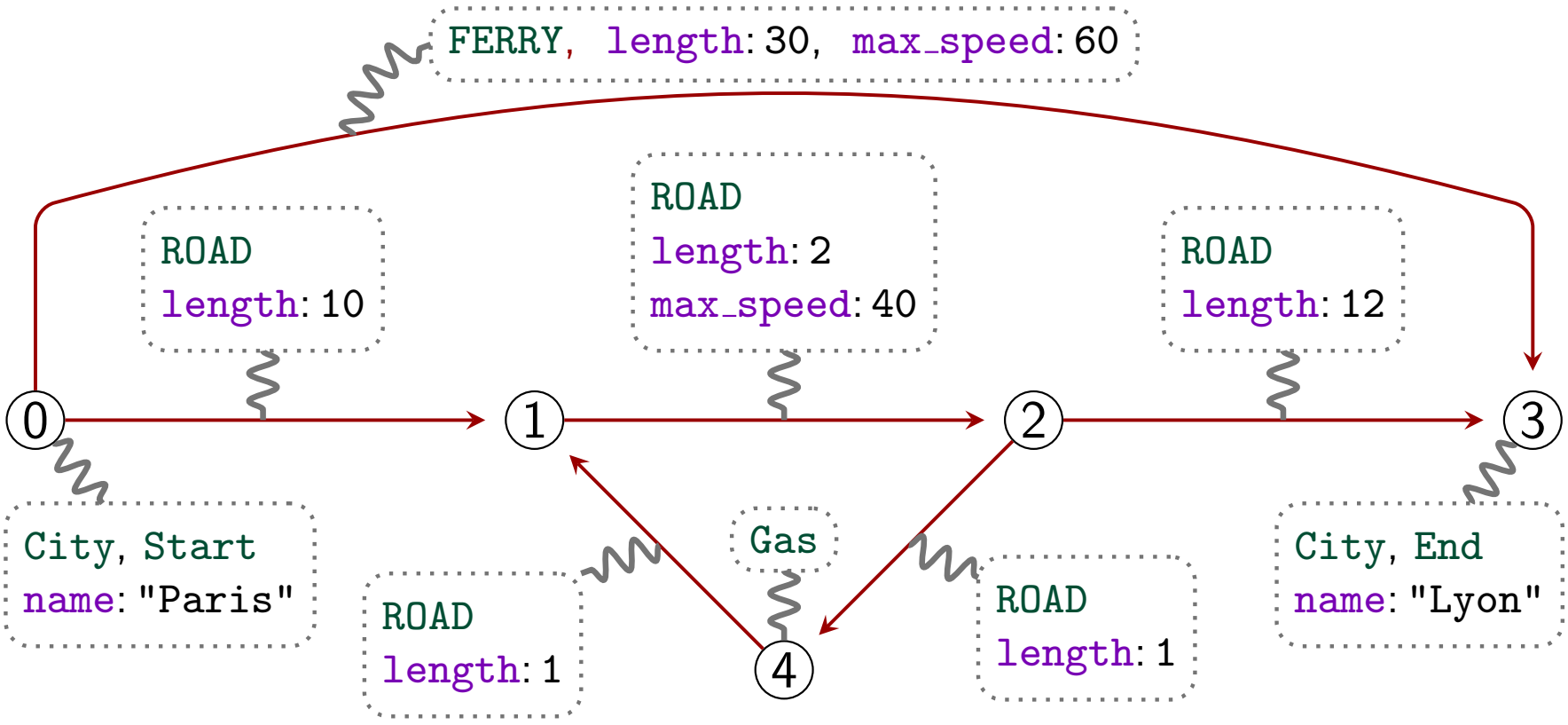
Syntax

`reduce(`⟨*acc*⟩ `=` ⟨*init*⟩`,` ⟨*var*⟩ `IN` ⟨*list*⟩ `|` ⟨*update*⟩`)`

Equivalent to the following pseudo code

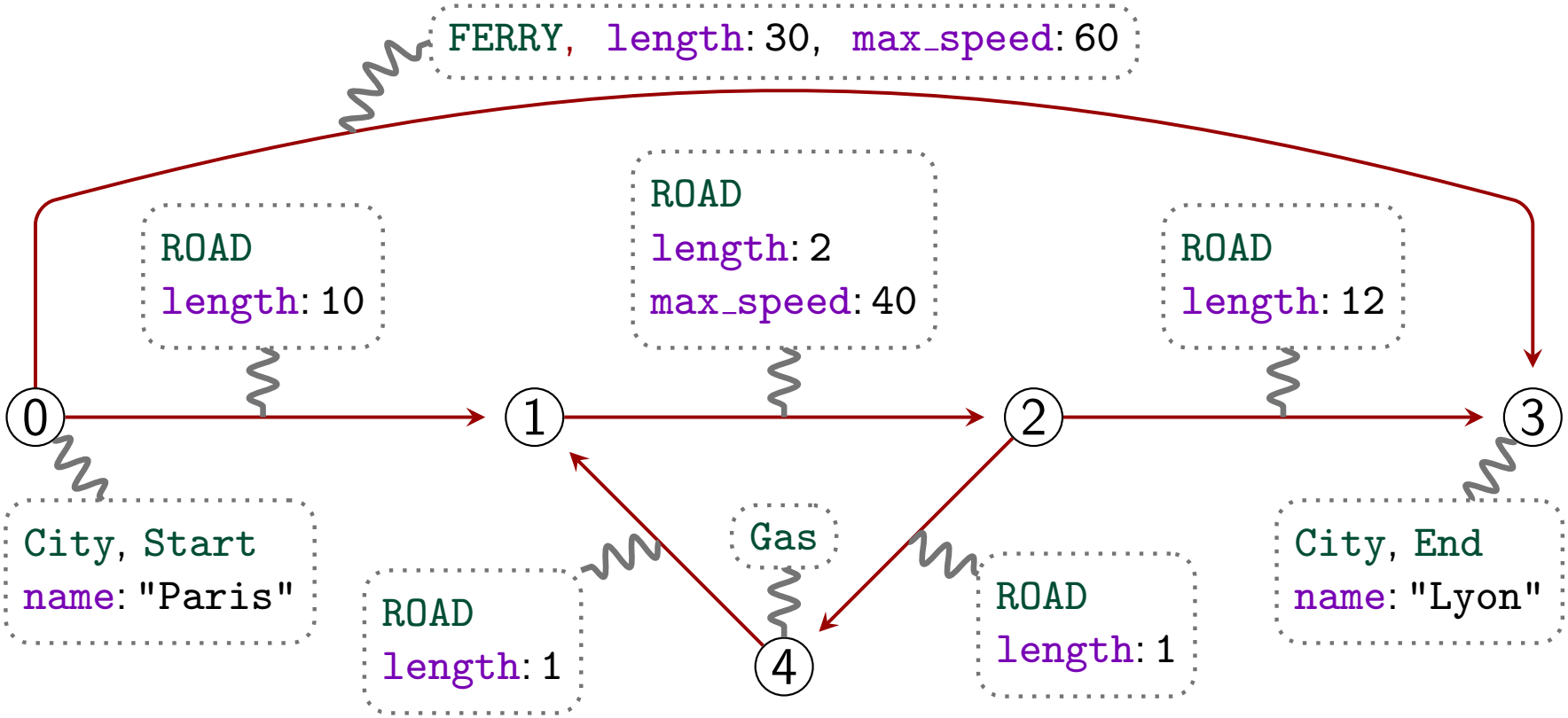
⟨*acc*⟩ `:=` ⟨*init*⟩
`for` ⟨*var*⟩ `in` ⟨*list*⟩`:`
    ⟨*acc*⟩ `:=` ⟨*update*⟩

Computing the length of a path




```
MATCH (:Start)-[e:ROAD|FERRY*]->(:End)
WITH reduce(acc=0, x IN e | acc+x.length) AS l
```

Computing the duration of a path




```
MATCH (:Start)-[e:ROAD|FERRY*]->(:End)
WITH reduce(acc = 0, x IN e
                | acc + x.length/coalesce(x.max_speed,80)) AS d
```

Part III: Cypher

5. Subclauses of `MATCH` and/or `WITH`

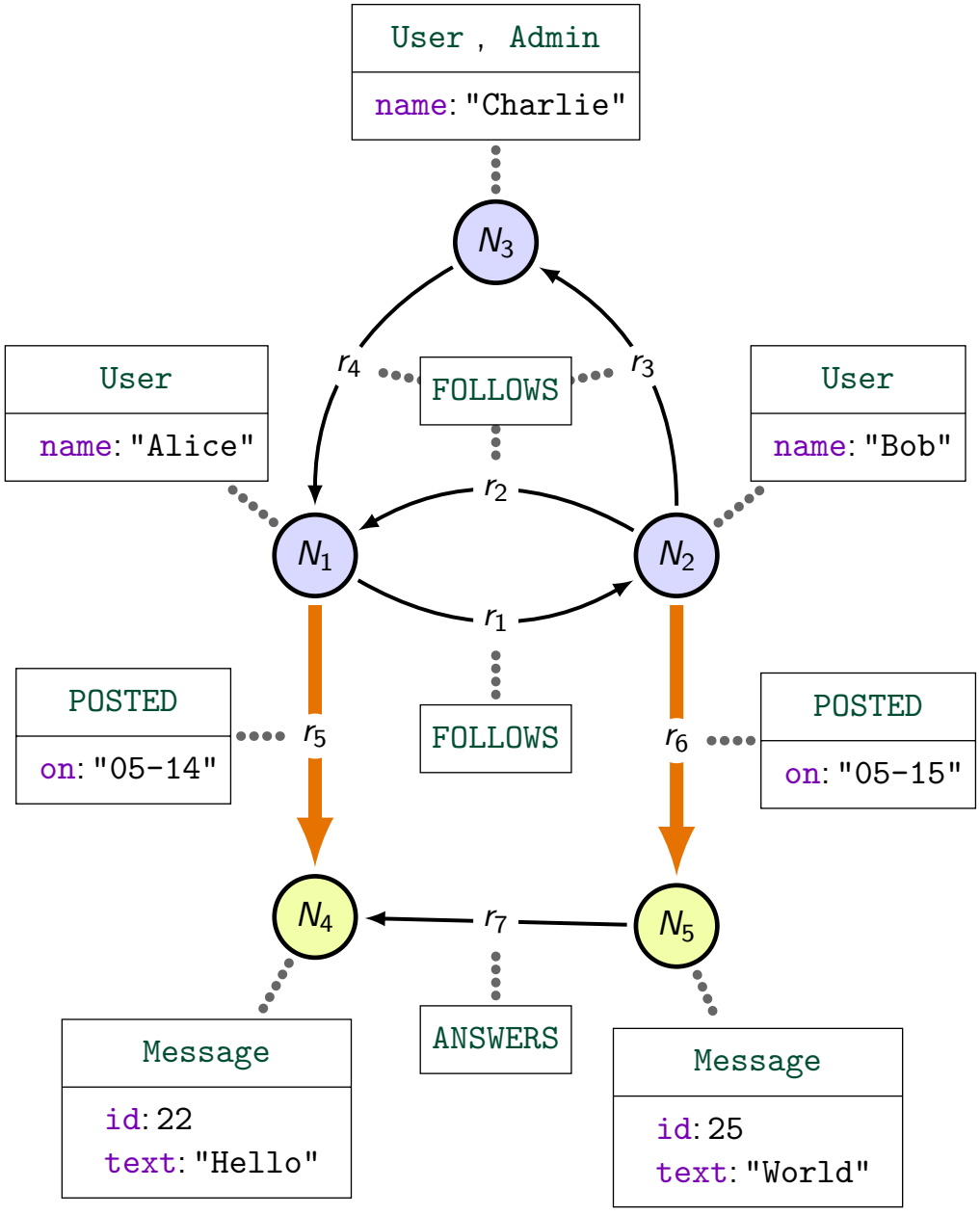
Filtering rows with WHERE (1)

Syntax

`MATCH` ... `WHERE` ⟨*condition*⟩
or
`WITH` ... `WHERE` ⟨*condition*⟩

Remove from the table computed by `MATCH` or `WHERE` the row that make ⟨*condition*⟩ false

Filtering rows with WHERE (2)



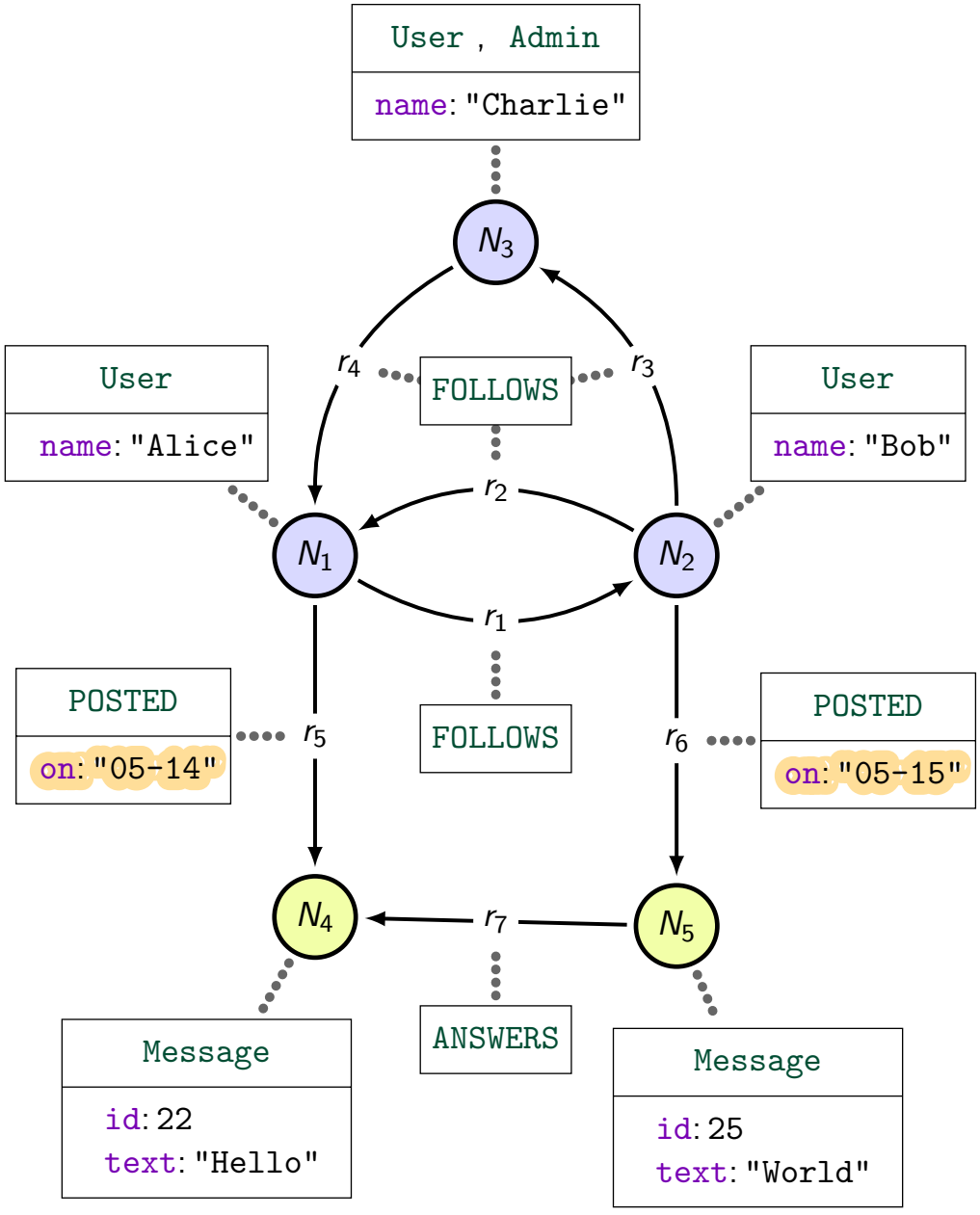
Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WHERE p1.on > "05-14"
```

After the MATCH clause

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Filtering rows with WHERE (2)



Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WHERE p1.on > "05-14"
```

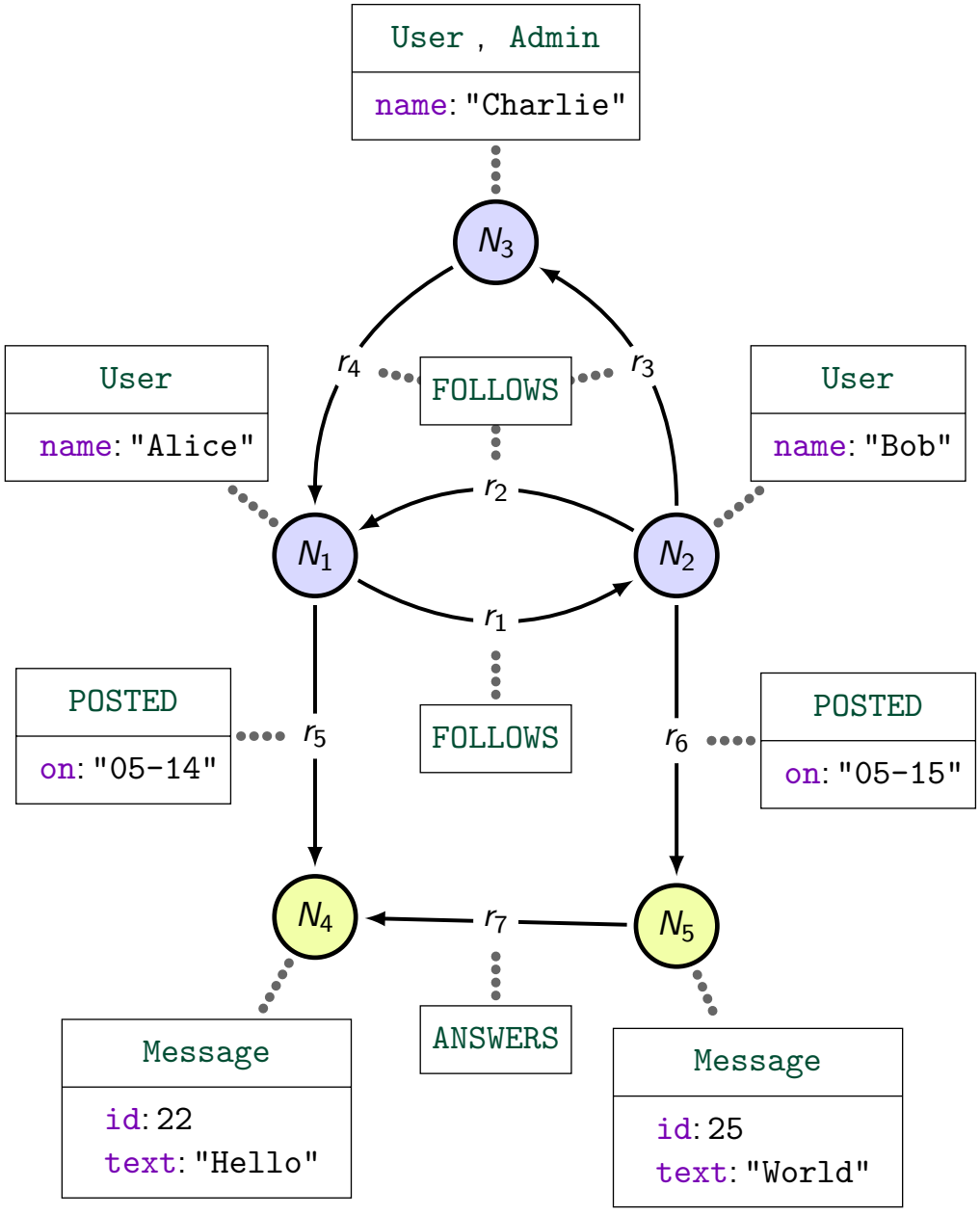
After the MATCH clause

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Execution of the WHERE clause

u1	p1	m1
~~N_1~~	~~r_5~~	~~N_4~~
N_2	r_6	N_5

Filtering rows with WHERE (2)



Query:

```
MATCH (u1)-[p1:POSTED]->(m1)
WHERE p1.on > "05-14"
```

After the MATCH clause

u1	p1	m1
N_1	r_5	N_4
N_2	r_6	N_5

Final result

u1	p1	m1
N_2	r_6	N_5

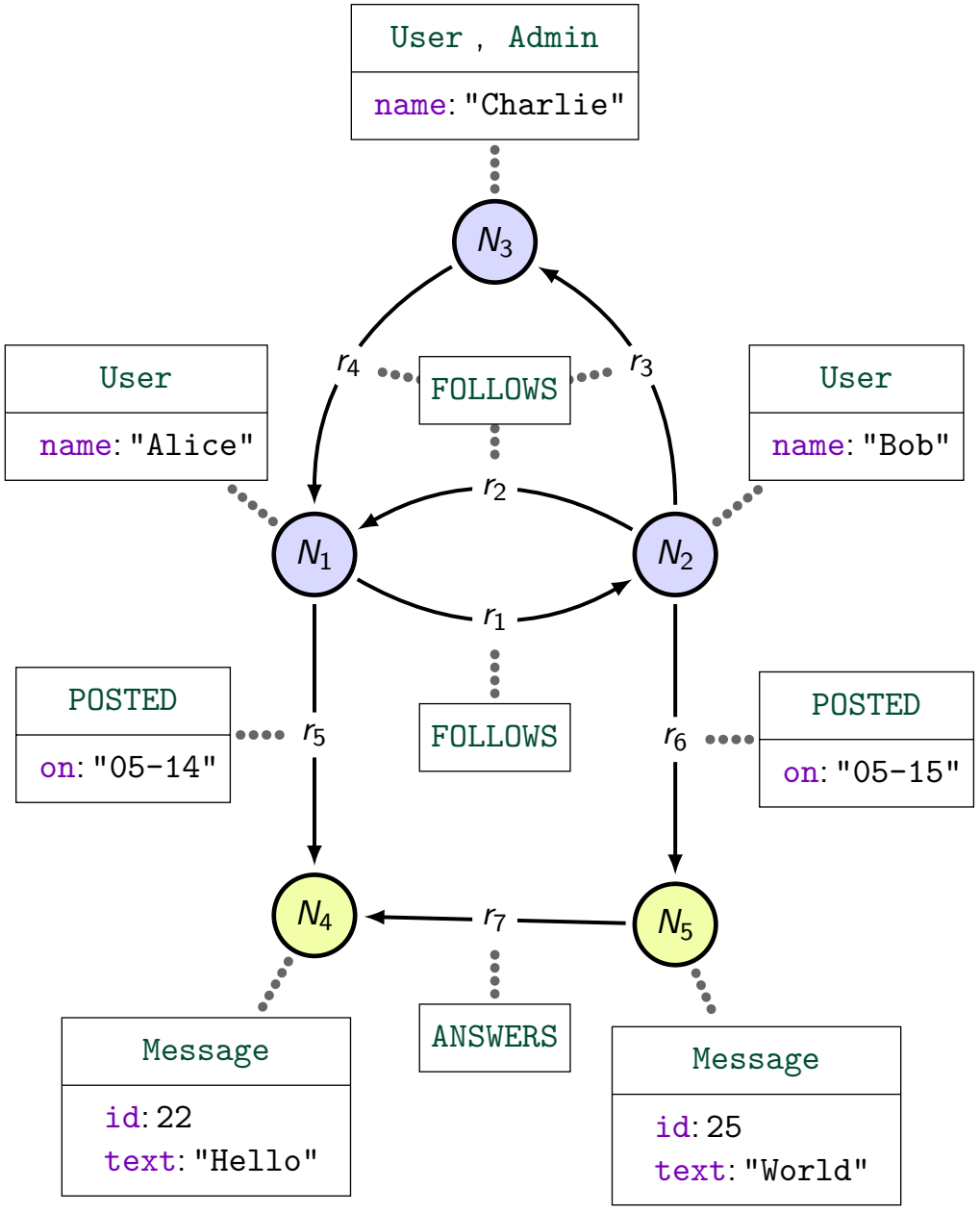
ORDER BY, LIMIT and SKIP

Syntax

$$\texttt{WITH} \ldots \underbrace{\texttt{ORDER BY} \langle oexpr_1 \rangle \overbrace{\texttt{DESC}}^{\text{optional}} , \ldots}_{\text{optional}} \underbrace{\texttt{SKIP} \langle sexpr \rangle}_{\text{optional}} \underbrace{\texttt{LIMIT} \langle lexpr \rangle}_{\text{optional}}$$

- Order the table by $\langle oexpr_1 \rangle$
  - Ties are broken by the value of $\langle oexpr_2 \rangle$, remaining ties are broken by $\langle oexpr_3 \rangle$, etc
  - `DESC` means the order is descending.
  - ⚠ We might end up with ties → Nondeterminism
- Then, remove the first $\langle sexpr \rangle$ rows
- Then, keep the first $\langle lexpr \rangle$ rows, at most

Compute the User with the most followers

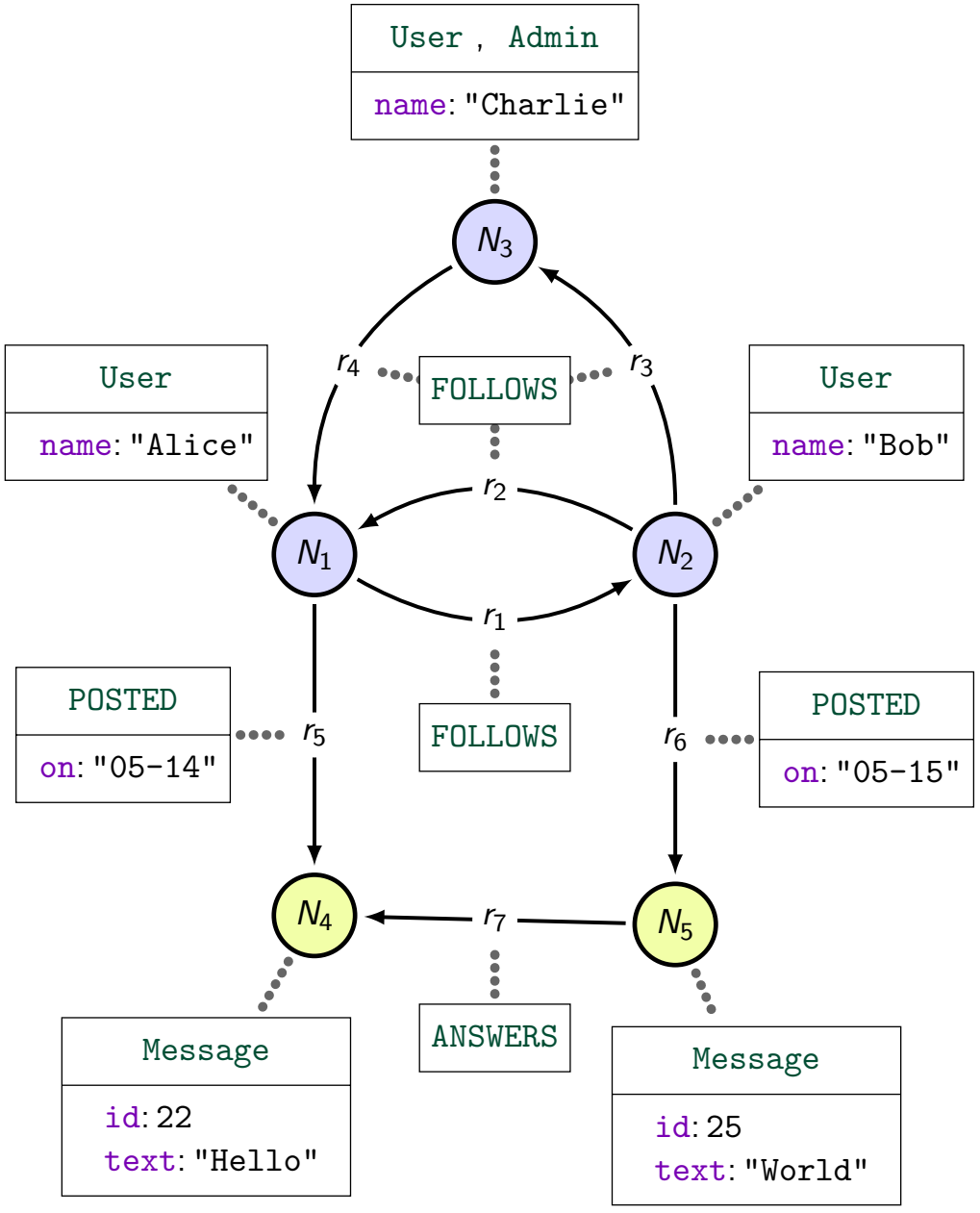


Query:

```
MATCH (u1)<-[:FOLLOWS]-(u2)
WITH u1, count(u2) AS c
  ORDER BY c DESC
  LIMIT 1
```

u1	c
N_1	2

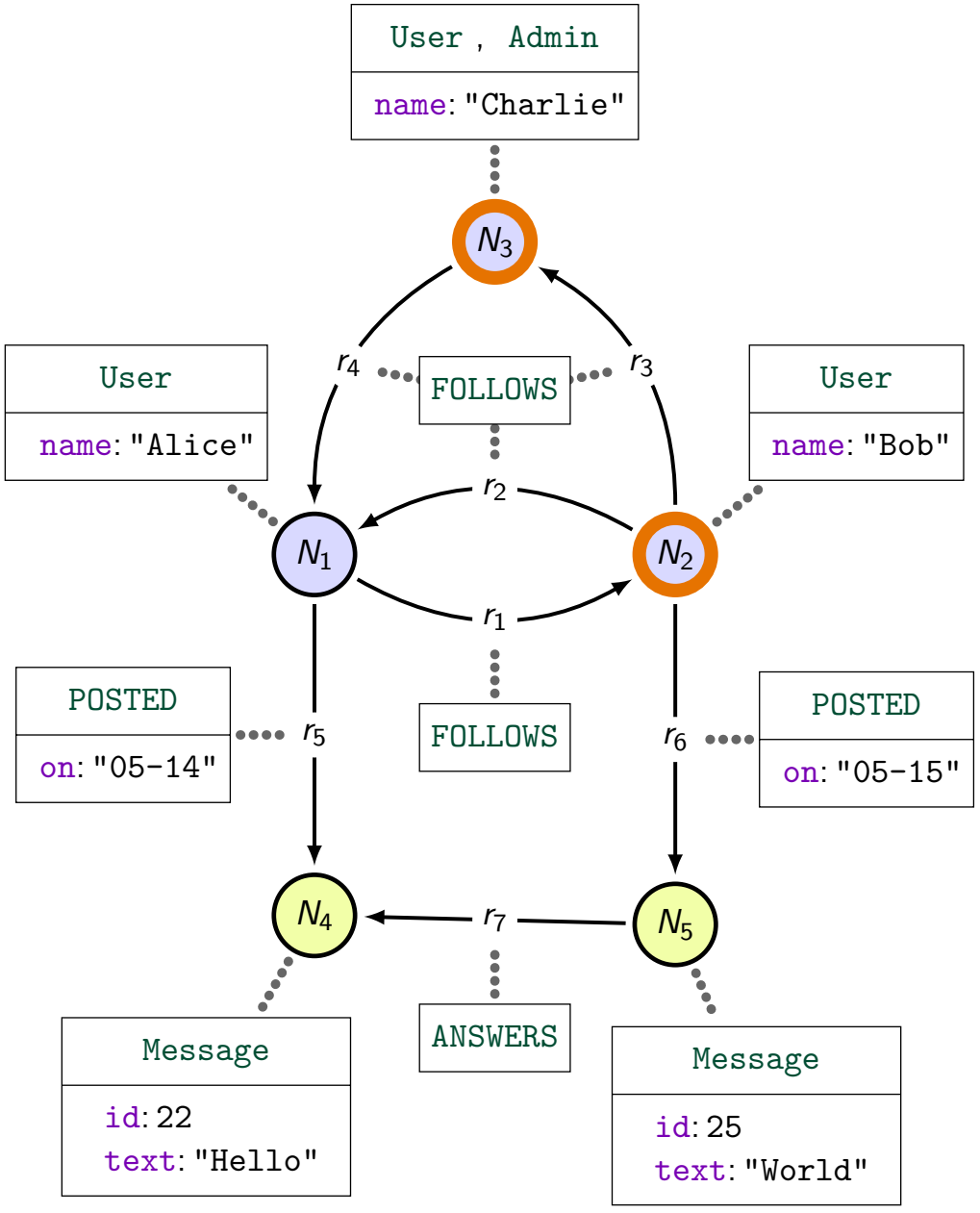
Compute the two User with the most followers



Query:

```
MATCH (u1)<-[:FOLLOWS]-(u2)
WITH u1, count(u2) AS c
  ORDER BY c DESC
  LIMIT 2
```

Compute the two User with the most followers



Query:

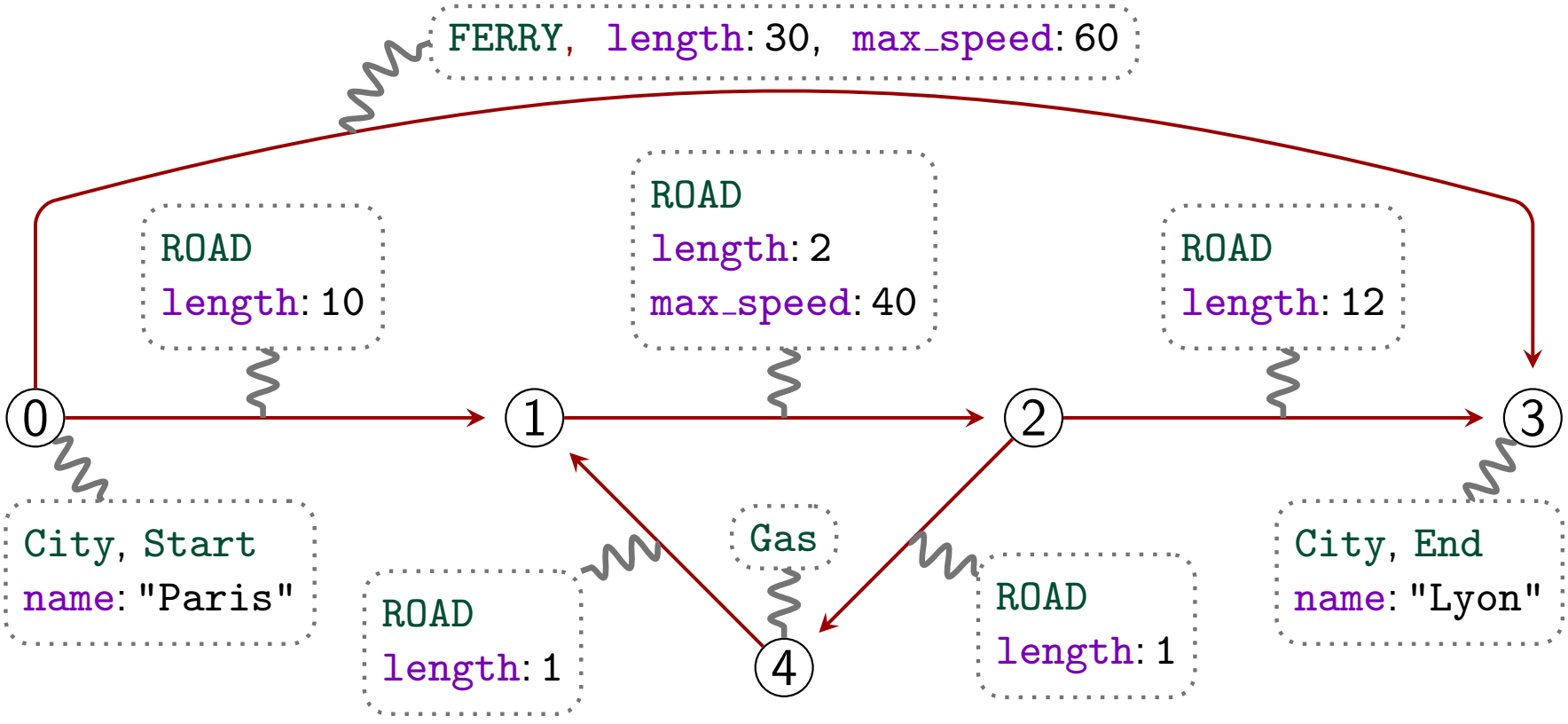
```
MATCH (u1)<-[:FOLLOWS]-(u2)
WITH u1, count(u2) AS c
  ORDER BY c DESC
  LIMIT 2
```

Since Charlie and Bob both have 1 follower, the final table is either:

u1	c
N_1	2
N_2	1

u1	c
N_1	2
N_3	1

Exercice: what does this compute?

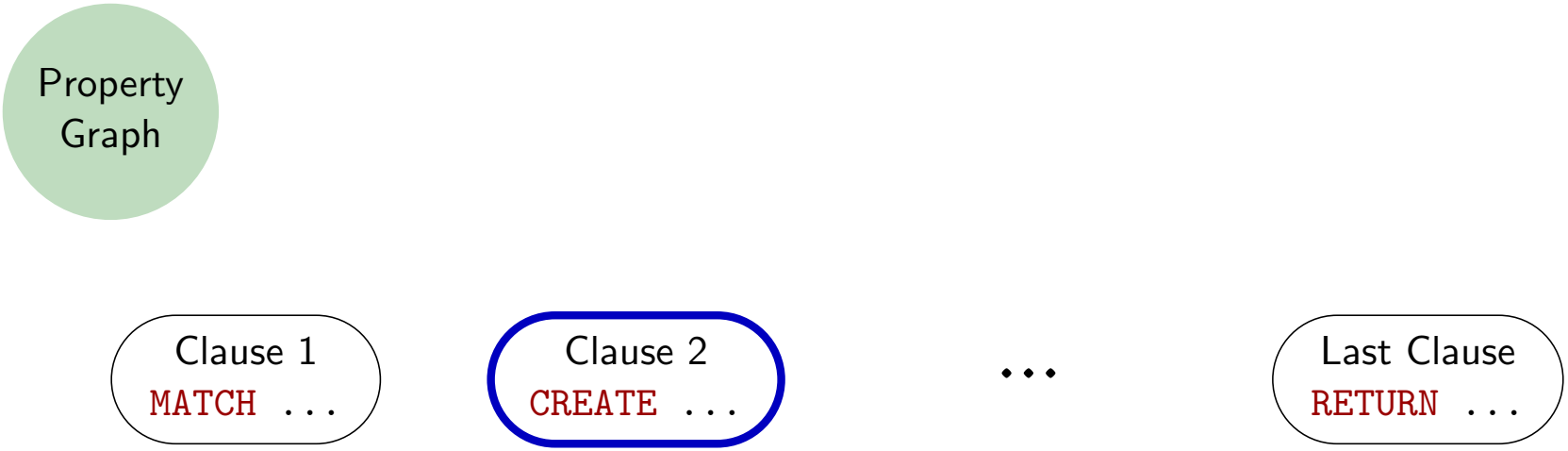



```
MATCH (:Start)-[e:ROAD*]->(:Gas)-[f:ROAD*]->(:End)
WITH reduce(acc=0, x IN e | acc+x.length) AS l,
     reduce(acc=0, x IN f | acc+x.length) AS m
  ORDER BY l+m ASC
  LIMIT 1
```

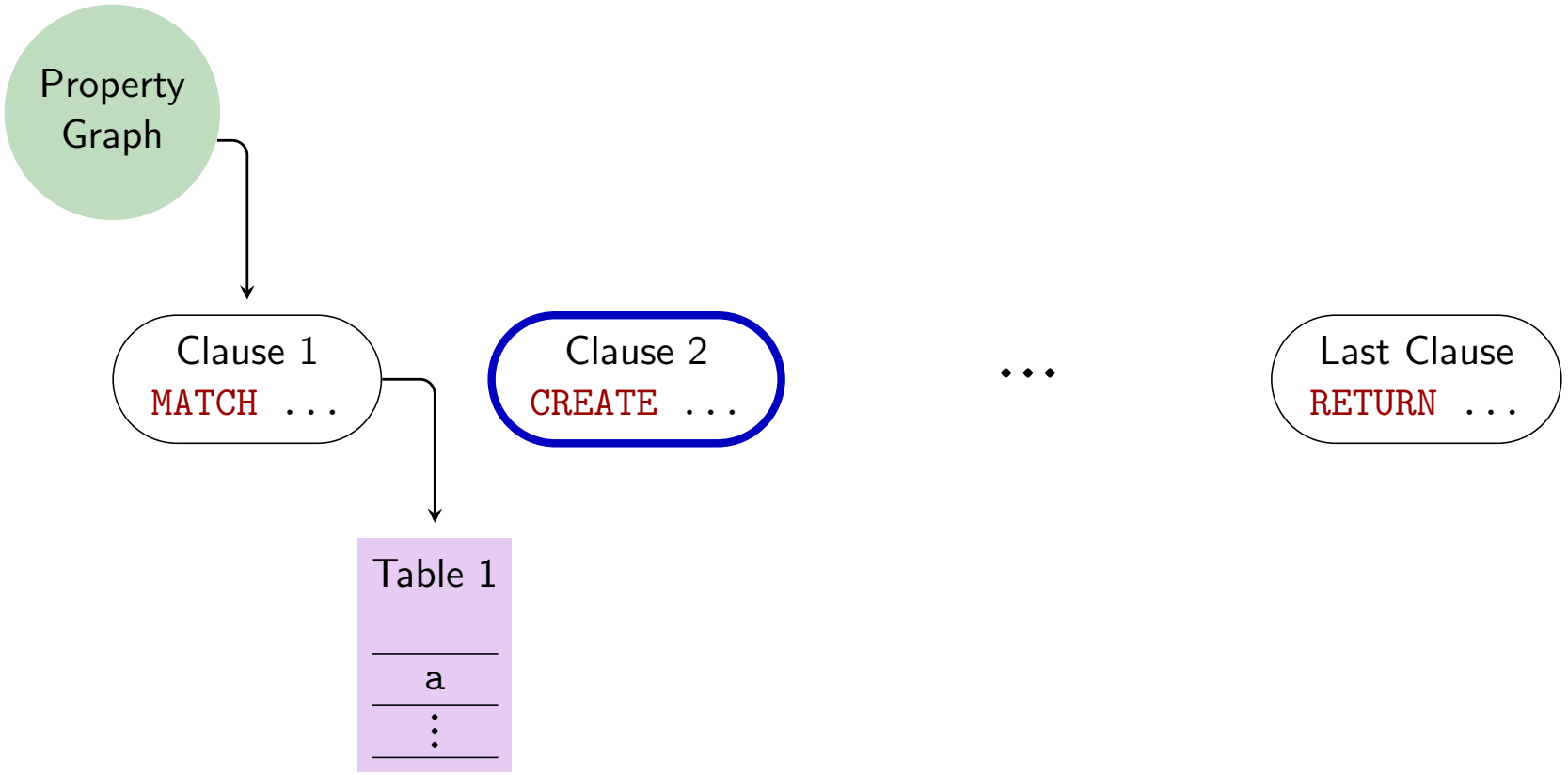
Part III: Cypher

6. Updates

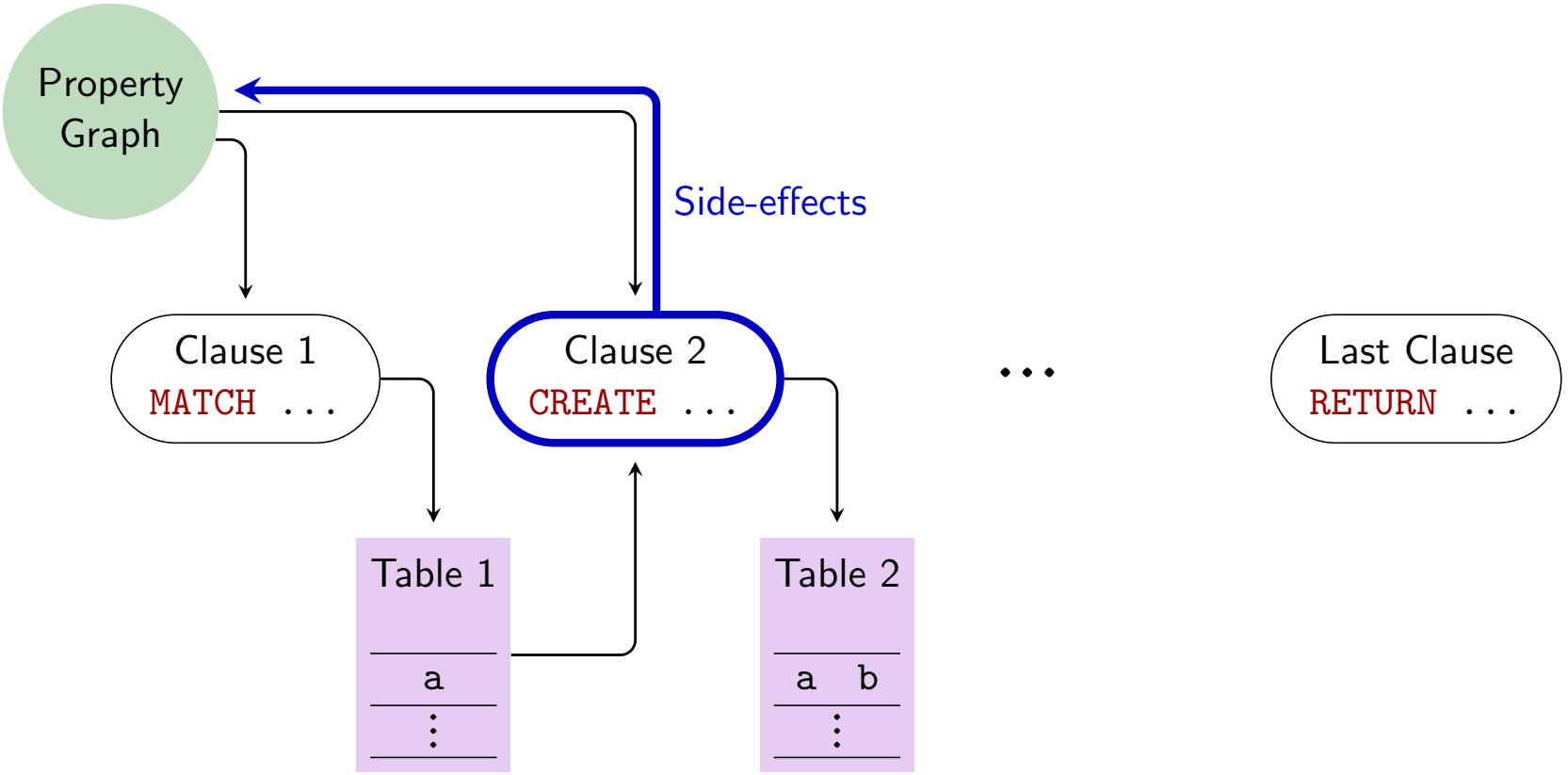
How evaluation works with update clauses



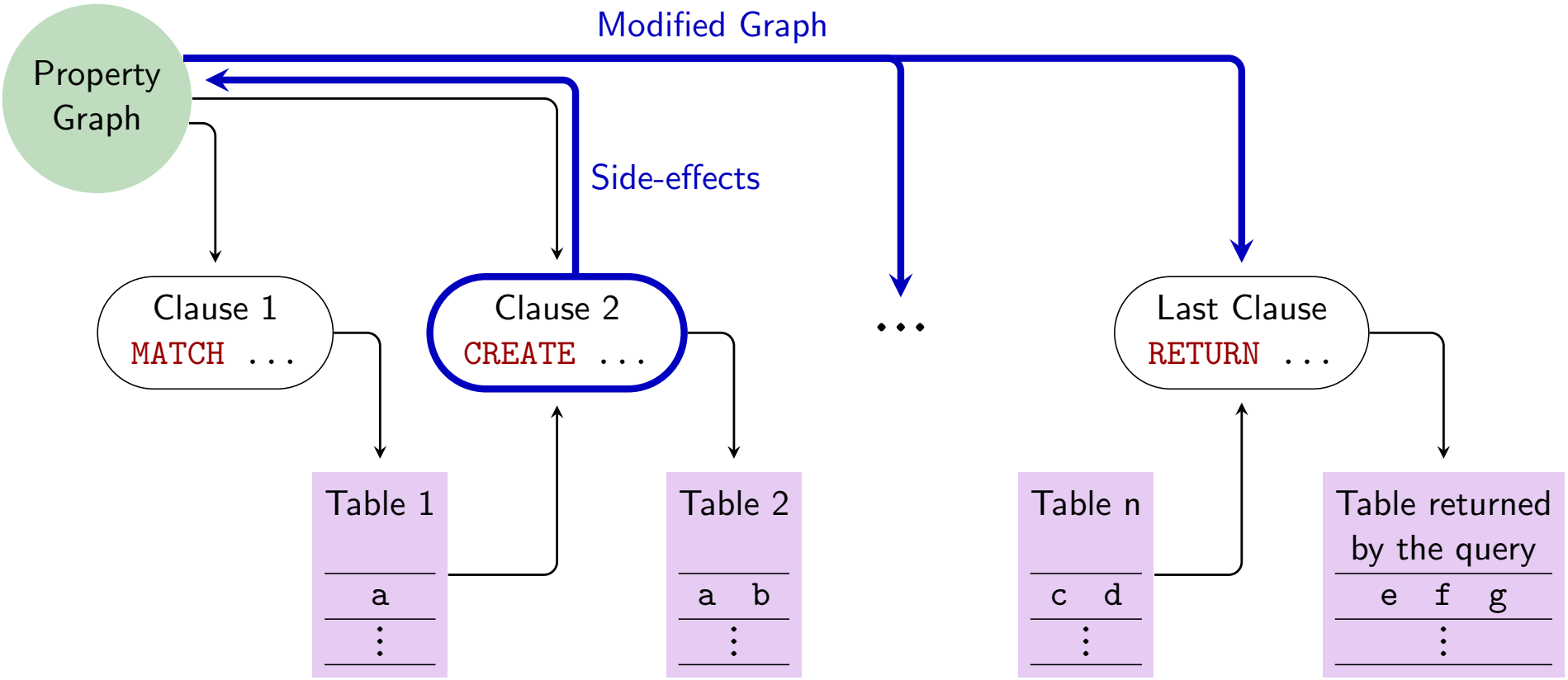
How evaluation works with update clauses



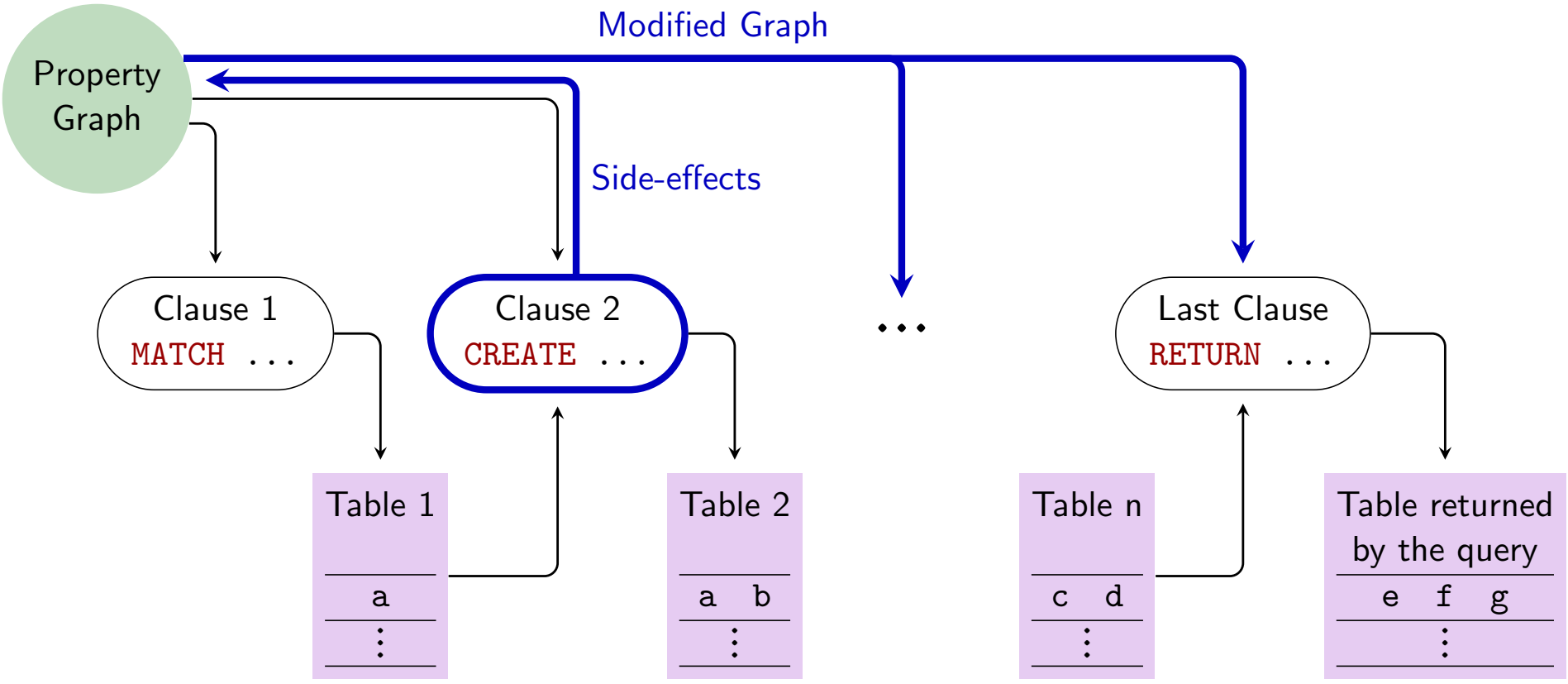
How evaluation works with update clauses



How evaluation works with update clauses



How evaluation works with update clauses



Neo4j complies to ACID

A $\Longrightarrow$ Modifications are **undone** if evaluation fails

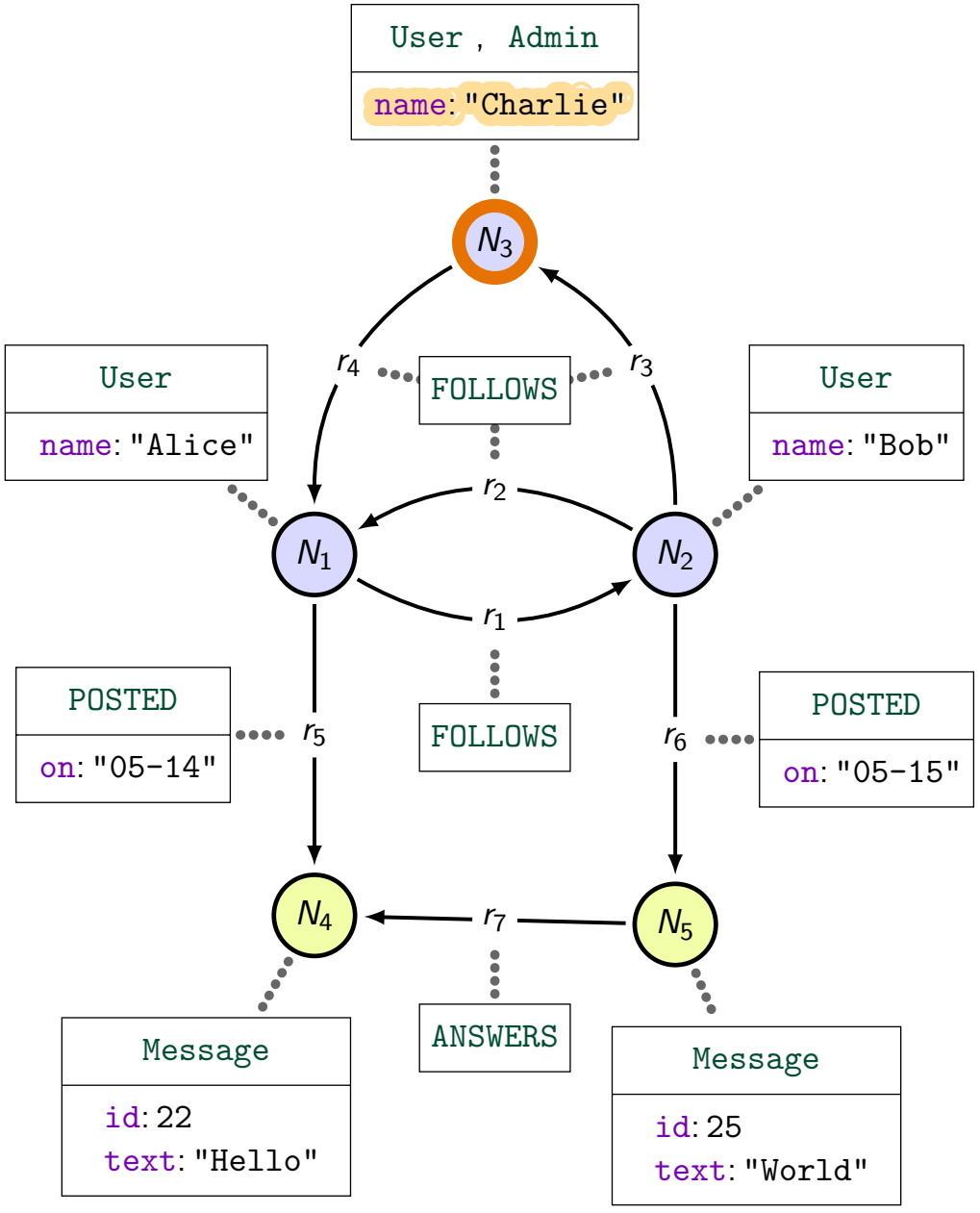
C $\Longrightarrow$ The PG must complies to IC **at the end** of evaluation only

I $\Longrightarrow$ Modifications are **invisible** to concurrent queries

Create nodes and relations (1)

- `CREATE (a:User {name:"Alice"})`
  - Creates a new node
  - Stores it in column `a`
- `CREATE (a)-[e:POSTED {on:"12-07"}]->(b)`
  - Creates a new relation from `a` to `b`
  - If `a` the input table has no column named `a`, creates a new node
  - Idem for `b`
  - Stores the new relation in column `e`

Create nodes and relations (2)



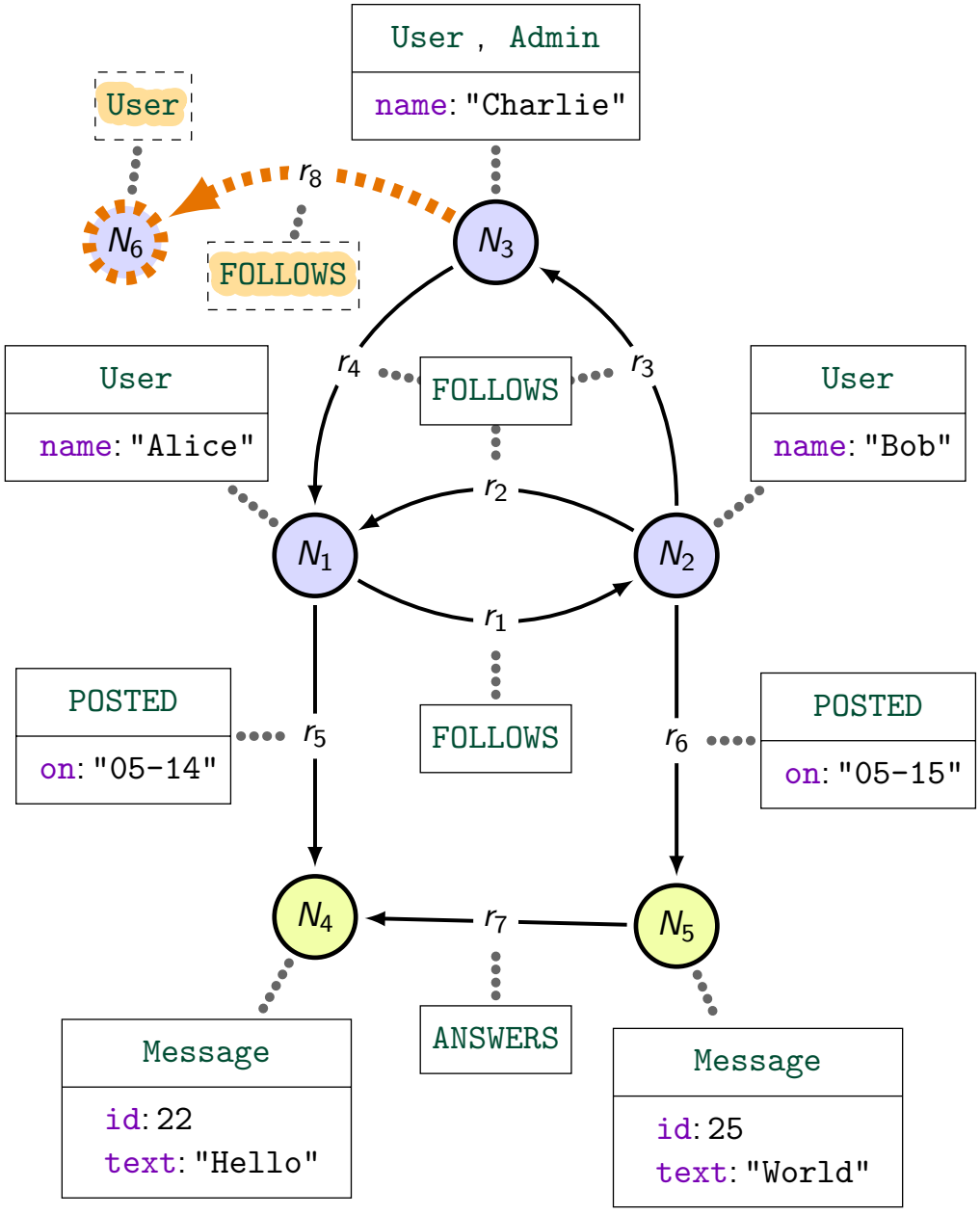
Query:

```
MATCH (a {name:"Charlie"})
CREATE (a)-[:FOLLOWS]->
                    (b:User)
```

Table after MATCH clause:

a
N_3

Create nodes and relations (2)



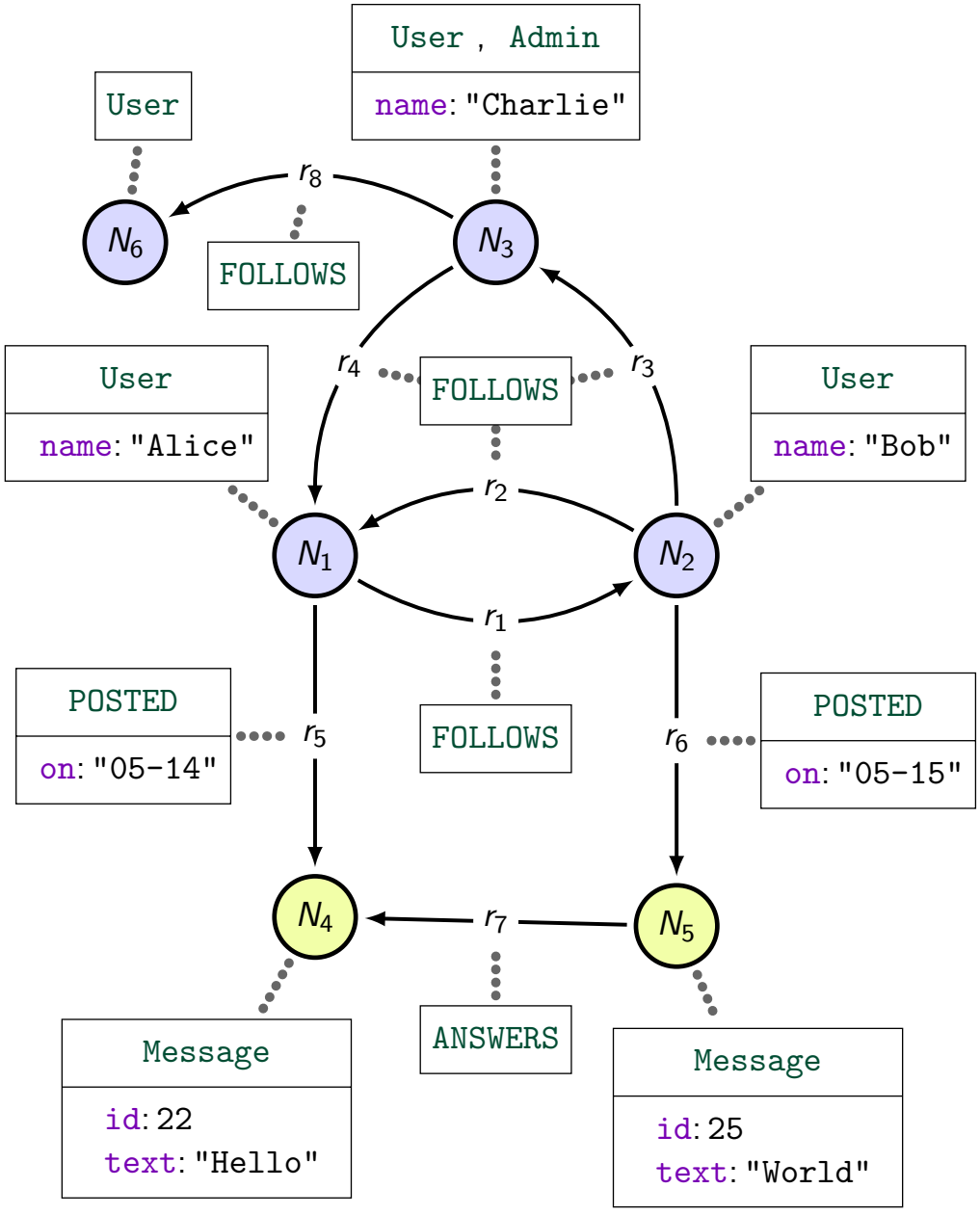
Query:

```
MATCH (a {name:"Charlie"})
CREATE (a)-[:FOLLOWS]->
                        (b:User)
```

Table after MATCH clause:

a
N_3

Create nodes and relations (2)



Query:

```
MATCH (a {name:"Charlie"})
CREATE (a)-[:FOLLOWS]->
                    (b:User)
```

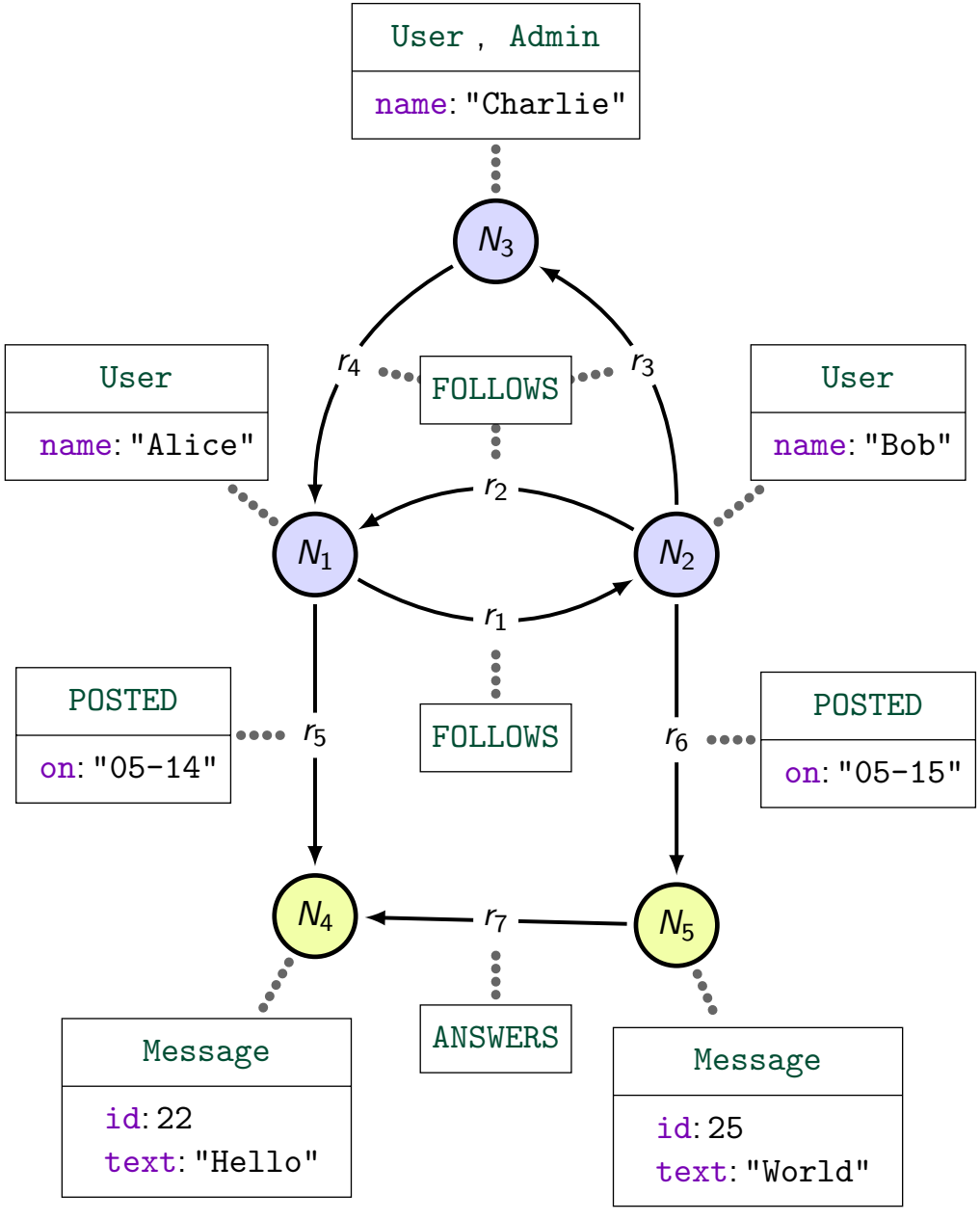
Table after MATCH clause:

a
N_3

Table after CREATE clause:

a	b
N_3	N_6

The example graph stored as CREATE clauses



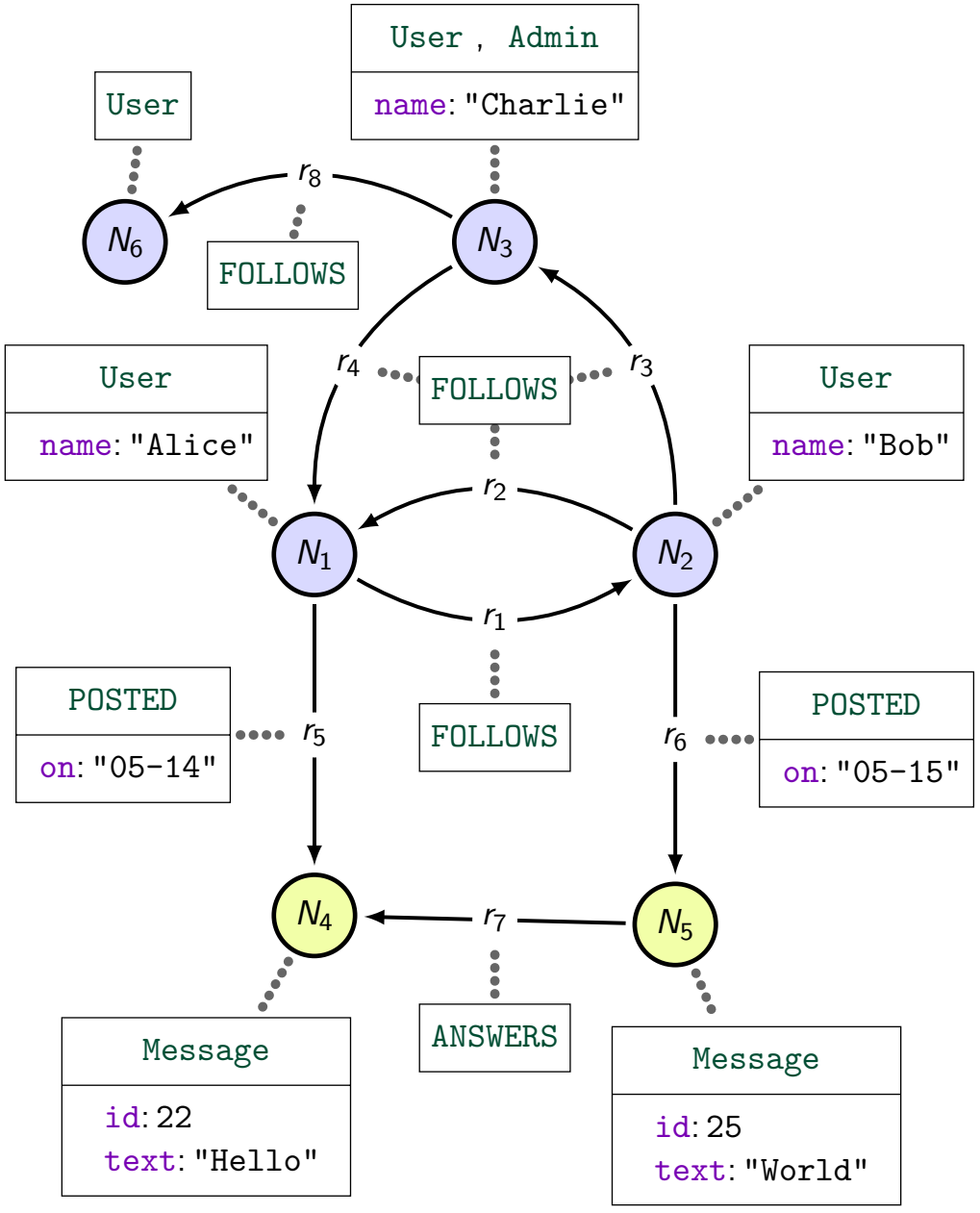
Query:

```
CREATE
  (n1:User{name:"Alice"}),
  (n2:User{name:"Bob"}),
  (n3:User:Admin
            {name:"Charlie"}),
  (n4:Message {id:22,
               text:"Hello"}),
  (n5:Message {id:25,
               text:"World"})
CREATE
  (n1)-[:FOLLOWS]->(n2),
  (n1)-[:POSTED
         {on:"05-04"}]->(n4),
  (n2)-[:FOLLOWS]->(n1),
  (n2)-[:FOLLOWS]->(n3),
  (n2)-[:POSTED
         {on:"05-04"}]->(n5),
  (n3)-[:FOLLOWS]->(n1),
  (n5)-[:ANSWERS]->(n4),
```

Delete nodes and relations

- `DELETE` a
  - If column a contains relations, delete them
  - If column a contains node:
    - if none of them has adjacent relation, delete them
    - otherwise the query fails.
- `DETACH DELETE` a
  - If column a contains relations, delete them
  - If column a contains nodes, delete them as well as every adjacent relations.

Modifying labels and properties (1)



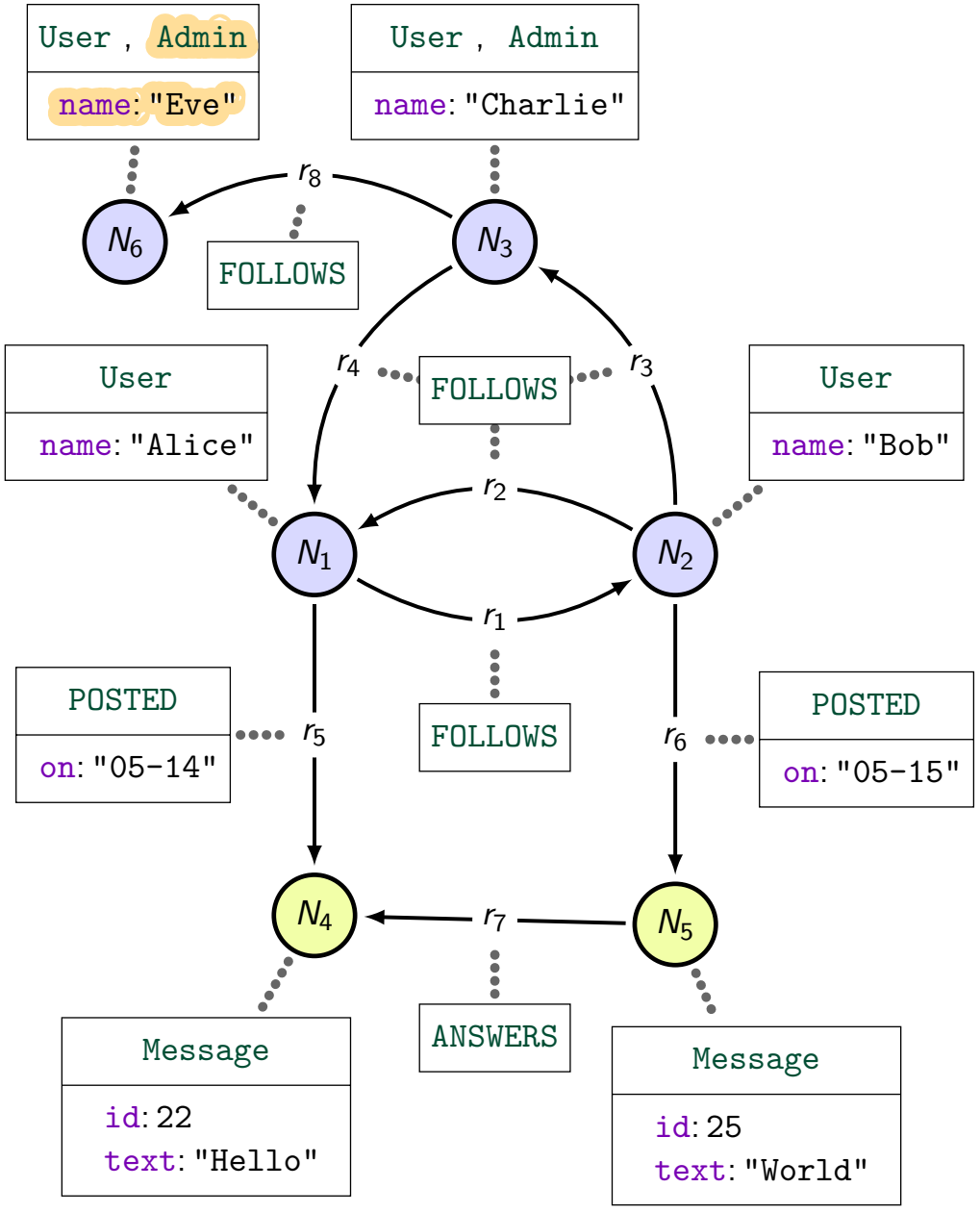
Query:

```
MATCH (a{name:"Charlie"})
CREATE (a)-[:FOLLOWS]->
                        (b:User)
SET b:Admin, b.name="Eve"
```

Table after CREATE clause:

a	b
N_3	N_6

Modifying labels and properties (1)



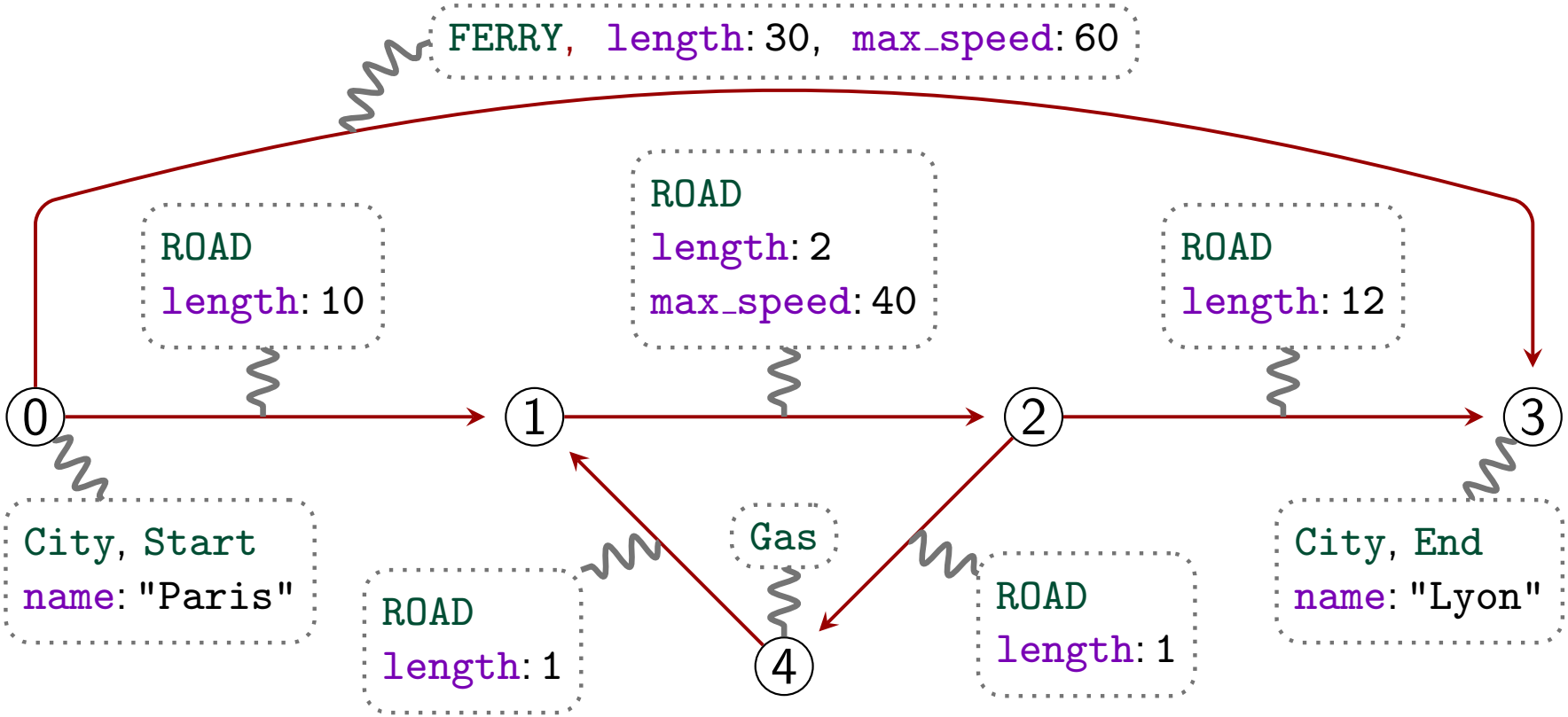
Query:

```
MATCH (a{name:"Charlie"})
CREATE (a)-[:FOLLOWS]->
                        (b:User)
SET b:Admin, b.name="Eve"
```

Table after CREATE clause:

a	b
N_3	N_6

Bulk updates




```
MATCH ()-[e:ROAD]->()
    WHERE e.max_speed IS NULL
SET e.max_speed=80
```

$\Longrightarrow$ Adds the property `max_speed`:80 to all `ROAD` that do not have one.

Part III: Cypher

7. Summary

Overview of read-only clauses

`MATCH` is for pattern matching

- C2RPQ-like
- Trail semantics
- Projects paths into a table
- Inner join with the input table
  - `OPTIONAL MATCH` does an outer join instead

Overview of read-only clauses

`MATCH` is for pattern matching

- C2RPQ-like
- Trail semantics
- Projects paths into a table
- Inner join with the input table
  - `OPTIONAL MATCH` does an outer join instead

`WITH` is for:

- Column manipulation (add, remove, rename, etc.)
- Aggregation
  - Vertical
  - Horizontal (`reduce`)
- Order and limit output size (`ORDER BY`, `SKIP` and `LIMIT`)

Overview of read-only clauses

`MATCH` is for pattern matching

- C2RPQ-like
- Trail semantics
- Projects paths into a table
- Inner join with the input table
  - `OPTIONAL MATCH` does an outer join instead

`WHERE` filters rows

- Subclause of `MATCH`, `WITH` and `RETURN`

`WITH` is for:

- Column manipulation (add, remove, rename, etc.)
- Aggregation
  - Vertical
  - Horizontal (`reduce`)
- Order and limit output size (`ORDER BY`, `SKIP` and `LIMIT`)

Overview of read-only clauses

`MATCH` is for pattern matching

- C2RPQ-like
- Trail semantics
- Projects paths into a table
- Inner join with the input table
  - `OPTIONAL MATCH` does an outer join instead

`WHERE` filters rows

- Subclause of `MATCH`, `WITH` and `RETURN`

`WITH` is for:

- Column manipulation (add, remove, rename, etc.)
- Aggregation
  - Vertical
  - Horizontal (`reduce`)
- Order and limit output size (`ORDER BY`, `SKIP` and `LIMIT`)

`RETURN` is a mandatory `WITH` at the end of the query

Overview of read-only clauses

`MATCH` is for pattern matching

- C2RPQ-like
- Trail semantics
- Projects paths into a table
- Inner join with the input table
  - `OPTIONAL MATCH` does an outer join instead

`WHERE` filters rows

- Subclause of `MATCH`, `WITH` and `RETURN`

`UNWIND` splits rows for each element in a list

`WITH` is for:

- Column manipulation (add, remove, rename, etc.)
- Aggregation
  - Vertical
  - Horizontal (`reduce`)
- Order and limit output size (`ORDER BY`, `SKIP` and `LIMIT`)

`RETURN` is a mandatory `WITH` at the end of the query

Overview of read-only clauses

`MATCH` is for pattern matching

- C2RPQ-like
- Trail semantics
- Projects paths into a table
- Inner join with the input table
  - `OPTIONAL MATCH` does an outer join instead

`WHERE` filters rows

- Subclause of `MATCH`, `WITH` and `RETURN`

`UNWIND` splits rows for each element in a list

`WITH` is for:

- Column manipulation (add, remove, rename, etc.)
- Aggregation
  - Vertical
  - Horizontal (`reduce`)
- Order and limit output size (`ORDER BY`, `SKIP` and `LIMIT`)

`RETURN` is a mandatory `WITH` at the end of the query

`UNION` and `UNION ALL` are for set and bag union.

Overview of read-write clauses

`CREATE` creates new elements, and sets their properties

- `MATCH`-like patterns (no *, |)
- Variables bound in input table
  - allows to link new elements to existing elements
  - hence allows bulk creation
- Unbound variables
  - induces creation of new nodes and relations
  - are put in the output table for further updates

Overview of read-write clauses

`CREATE` creates new elements, and sets their properties

- `MATCH`-like patterns (no *, |)
- Variables bound in input table
  - allows to link new elements to existing elements
  - hence allows bulk creation
- Unbound variables
  - induces creation of new nodes and relations
  - are put in the output table for further updates

(`DETACH`) `DELETE` deletes nodes and/or edges.

Overview of read-write clauses

`CREATE` creates new elements, and sets their properties

- `MATCH`-like patterns (no *, |)
- Variables bound in input table
  - allows to link new elements to existing elements
  - hence allows bulk creation
- Unbound variables
  - induces creation of new nodes and relations
  - are put in the output table for further updates

(`DETACH`) `DELETE` deletes nodes and/or edges.

`SET` modifies labels, types and properties

Overview of read-write clauses

`CREATE` creates new elements, and sets their properties

- `MATCH`-like patterns (no *, |)
- Variables bound in input table
  - allows to link new elements to existing elements
  - hence allows bulk creation
- Unbound variables
  - induces creation of new nodes and relations
  - are put in the output table for further updates

(`DETACH`) `DELETE` deletes nodes and/or edges.

`SET` modifies labels, types and properties

`MERGE` is a "`MATCH` else `CREATE`" mecanism:

- `MATCH`-like patterns (no *, |)
- If a line in the input table is not matched, new elements are created
- "Similar" elements are created only once
- Allows to integrate relational data (csv)
- Nondeterminism issues

Essential knowledge in part III

- General scheme of evaluating a Cypher query.
- Writing Cypher queries with `MATCH`, `WITH`, `WHERE`, `RETURN` clauses.
- Writing Cypher queries with several clauses.
- The two kinds of aggregation and how to use them with Cypher.
- Writing Cypher queries to update the database (`CREATE`, `DELETE`, `SET`.)
- Translations: Cypher $\leftrightarrow$ C2RPQs

Appendix

Navigable outline (1)

Navigable outline (2)

Part II: Property Graphs

Navigable outline (3)

Part III: Cypher

English-French translation I

English	French
Acyclic	Acyclique, Acircuitique
Bag, multiset	Multi-ensemble
Data model (DM)	Modèle de données
Edge	Arête, Arc
Endpoints	Extrémités
Endpoint semantics	Sémantique d'extrémité
Entity-Relationship diagram (ER diagram)	Schéma Entité-Association (EA)
Key	Clef
Label	Etiquette
Match	
Pattern matching	Recherche de motif

English-French translation II

Property, Attribute	Propriété, Attribut
Property Graph (PG)	Graphe à propriétés, Graphe de propriété, Graphe attribué
Regular Path Query (RPQ)	
Semantics	Semantique
Set	Ensemble
Shortest semantics	Sémantique de plus-court-chemin
Source	Source
Target	Destination
Trail	
Trail semantics	Sémantique sans-répétition-d'arête
Type	Type
Value	Valeur
Vertex, Node	Sommet, Noeud

English-French translation III

Walk, Path

Chemin, Marche