SQL -> Graql

Created by: Brandon Fergerson

Agenda

- **Setup Grakn**

- **What is a graph?**

- **Why use graphs?**

- **Are hypergraphs useful?**

- **How do I model a graph?**

- **Introducing Grakn**

  - Ontology & Knowledge Model

  - Hierarchies & Relations

  - Modeling Tips

- **Introducing Graql**

  - Defining Schemas

  - Writing Data

  - Reading Data

  - Deleting/Modifying Data

- **Exercises**

Setup Grakn

Setup Grakn & Grakn Workbase

- Download grakn-core-all-* from https://github.com/graknlabs/grakn/releases/tag/1.6.2
- Extract grakn-core-all, open terminal/CMD, and CD into extracted directory
- Run ./grakn server start (or .\grakn.bat server start on Windows)

- Download grakn-workbase-* from https://github.com/graknlabs/workbase/releases/tag/1.2.7
- Extract grakn-workbase, open terminal, and CD into extracted directory
  - On Windows, double-click the .exe to install
- Run ./grakn-workbase
  - On Windows, run installed Grakn Workbase app

Download/Clone Training Data

https://github.com/BFergerson/sql-to-graql

Executing Graql Files

- On Linux/mac:
  - ./grakn console -f /path/to/file.gql

- On Windows:
  - .\grakn.bat console -f C:\path\to\file.gql

Notice:
- Files under "graql" are standalone and can be executed separately or together in any order
- Files under "answers/exercise" are standalone and can be executed separately or together in any order
- Files under "answers/fill-in-the-blank" must be executed in the order they appear in this presentation
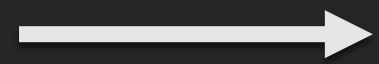
What is a graph?

What is a graph?

An abstract representation of a set of objects where some pairs are connected by links.

Vertex (Node, Point, Instance)

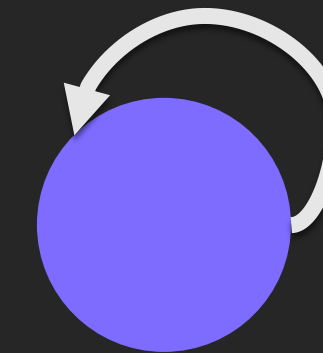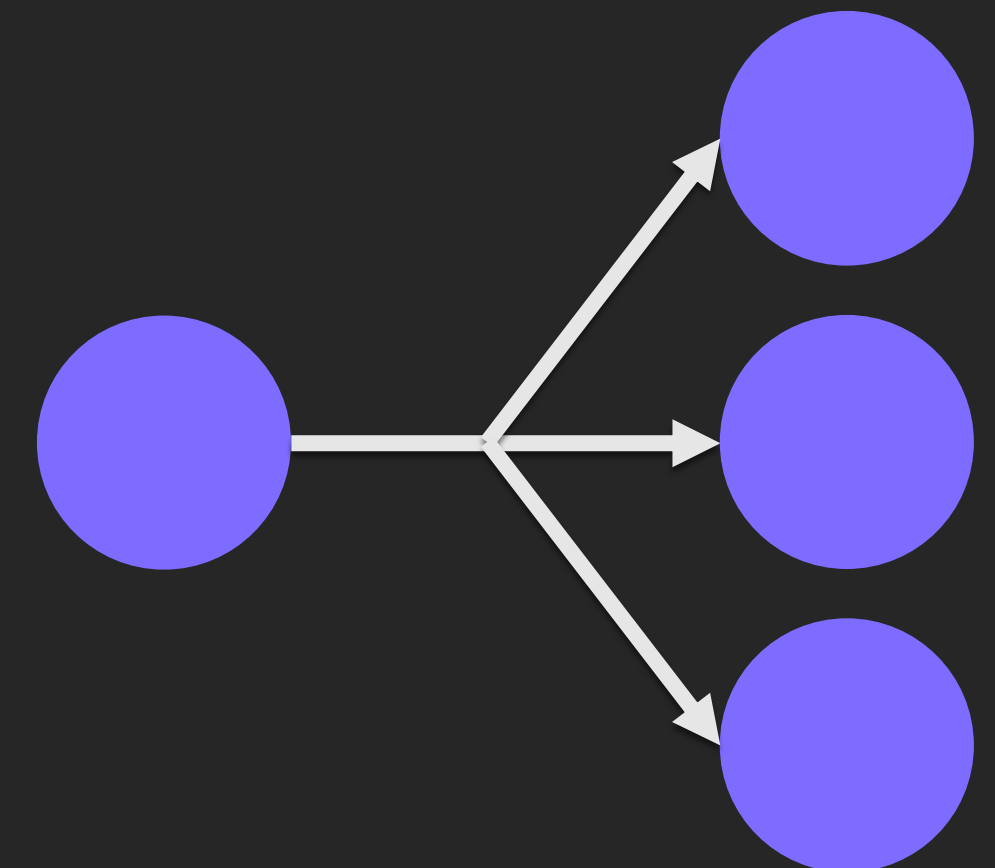Edge (Link, Line, Arc, Relation)

Undirected

Directed
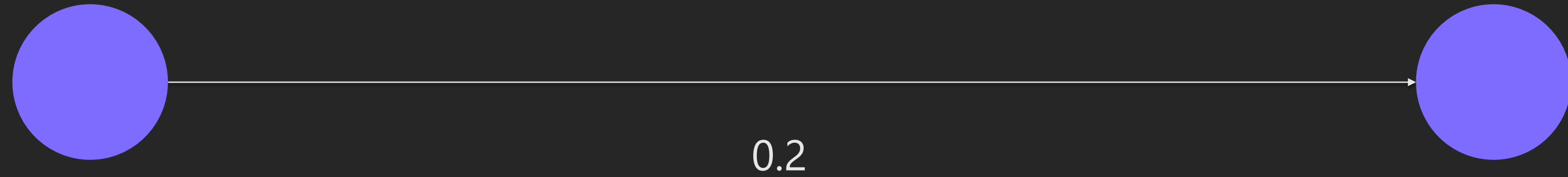
Pseudo

Multi

Hyper

What is a graph?

Weighted

0.2

Labeled

Max → Bob

knows

Property

Max → Bob

Type: has_brother
Since: 12/18/1980

Name: Max
Age: 32

Name: Bob
Age: 31

What is a graph?

Users	
...	...
2	Brandon
...	...

Checkouts	
...	...
2	90
2	99
...	...

Books	
90	Ender's Game
...	...
99	The Selfish Gene

What is a graph?

2	Brandon

2	90
2	99

90	Ender's Game

99	The Selfish Gene

What is a graph?

user   **Vertices**   book

book

Ender's Game

Brandon

The Selfish Gene

checkout   **Edges**   checkout

What is a graph?

1:1 Relationship

Id	Name	Address
1	Bob	3
2	Max	5
3	Jeffrey	10

Id	Location
3	Tampa
4	Liverpool
5	Vancouver
10	Dubai

1:n Relationship

Id	Name
1	Bob
2	Max
3	Jeffrey

Id	Customer	Location
3	1	Tampa
5	2	Vancouver
8	2	New York
10	3	Dubai

m:n Relationship

Id	Name
1	Bob
2	Max
3	Jeffrey

Cid	Aid
1	3
2	5
2	8
3	10

Id	Location
3	Tampa
5	Vancouver
8	New York
10	Dubai
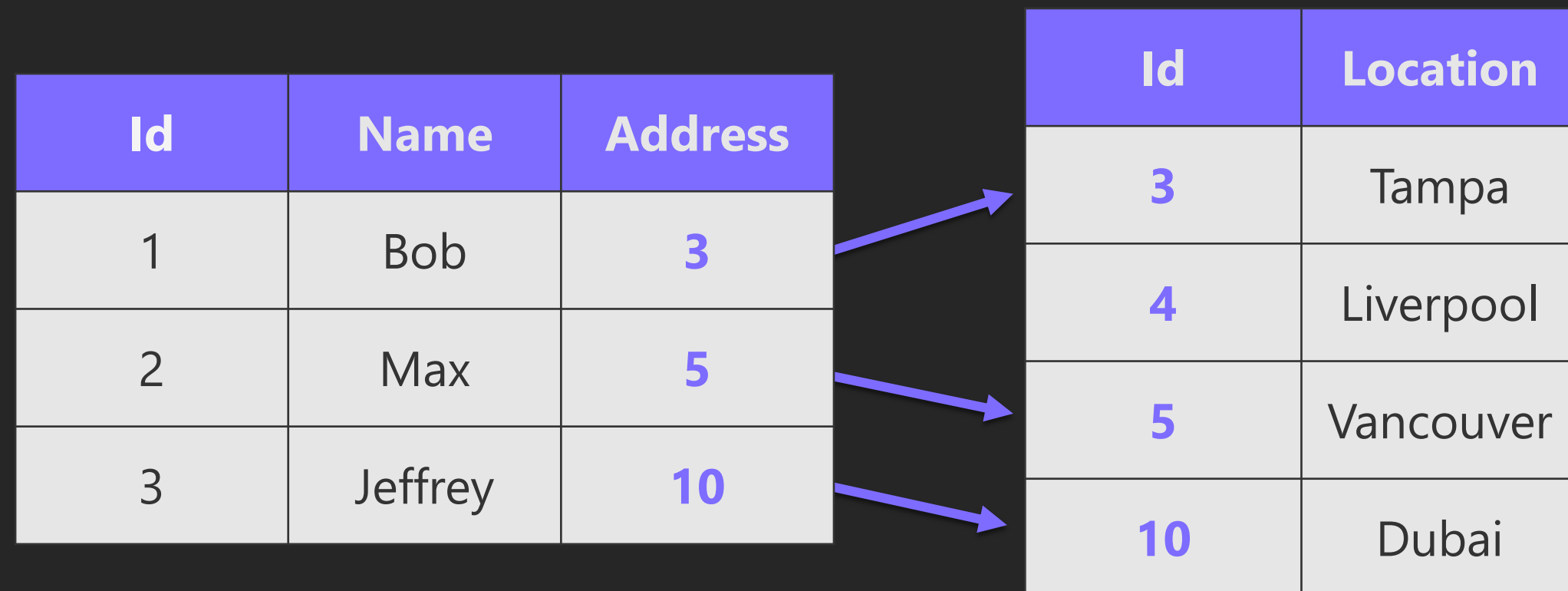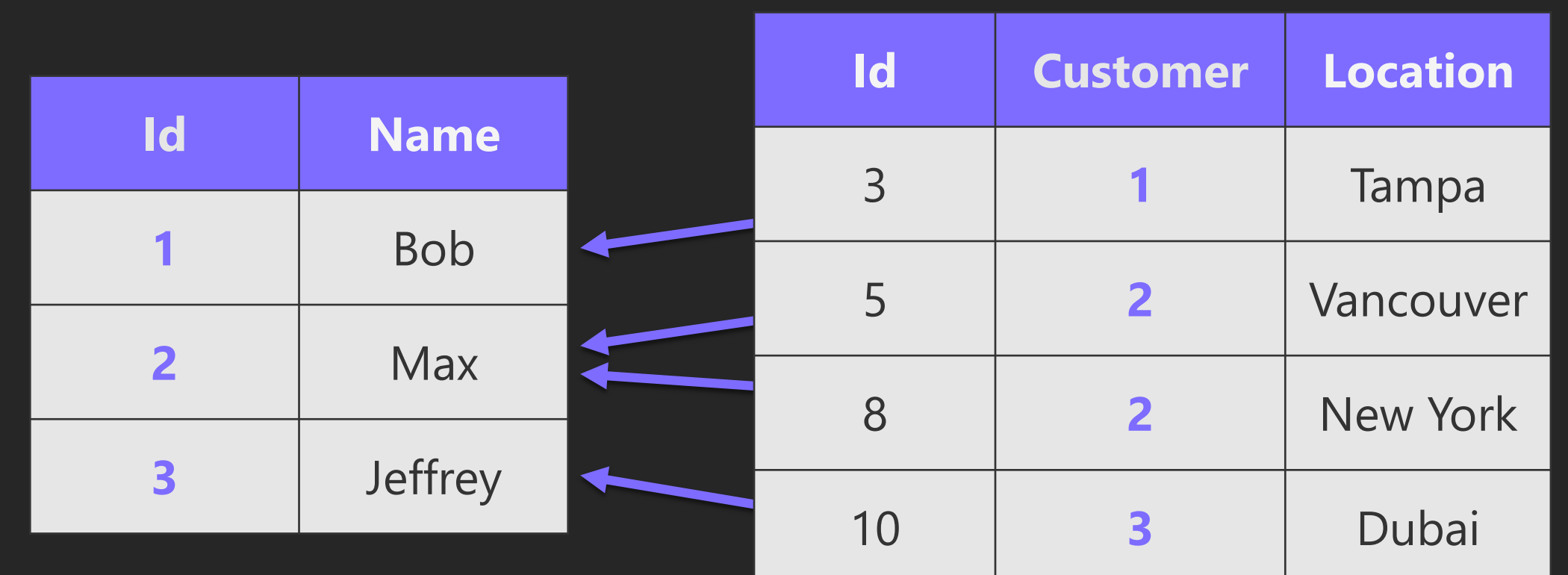
Why use graphs?

Why use graphs?

Optimized for aggregate data

Table A
A1
A2
A3

Table B
B1
B2
B3
B4
B5
B6
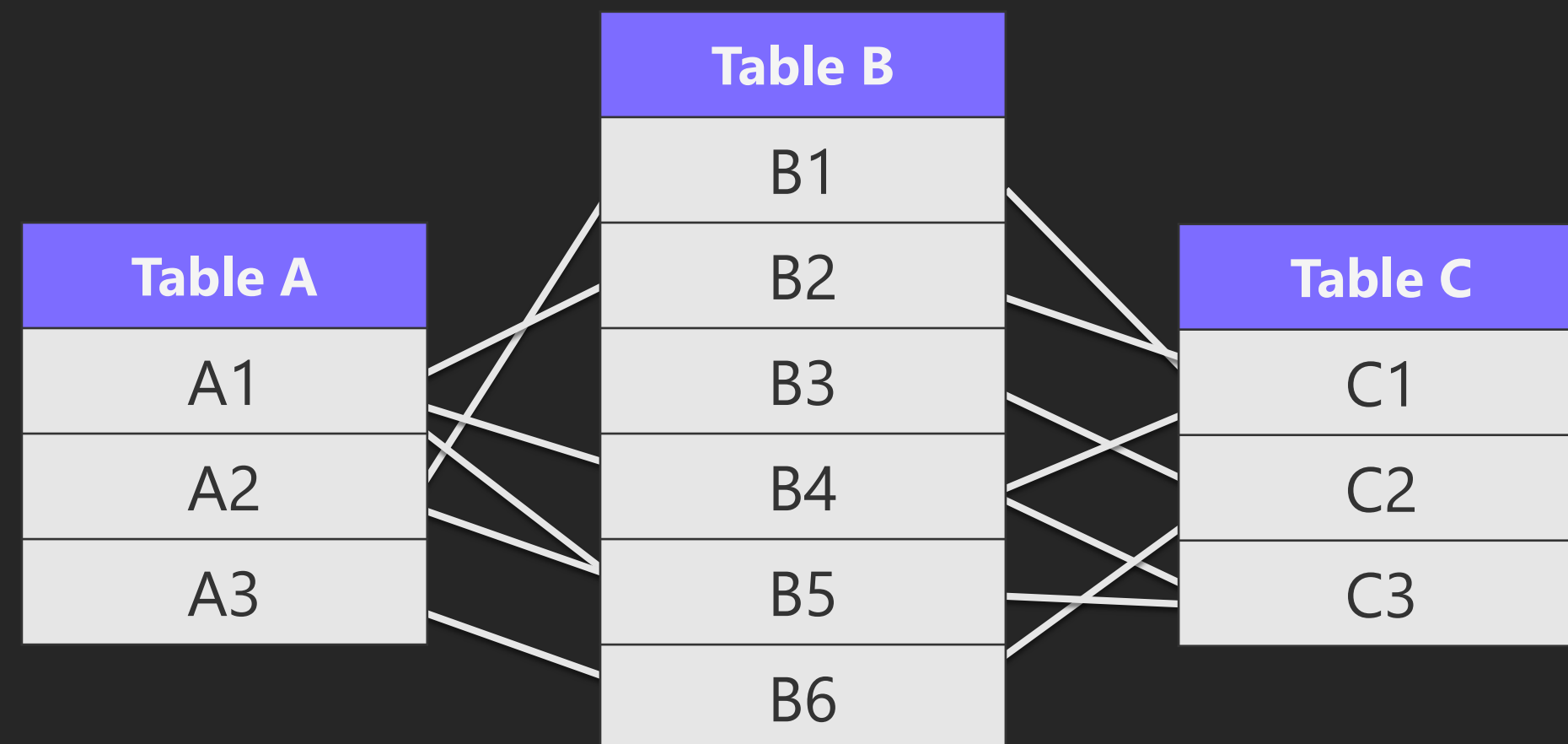
Table C
C1
C2
C3

Optimized for connected data

B6 — C2

C3 — A3 — B5

A2

B2

C1

B1

B4

B3

A1
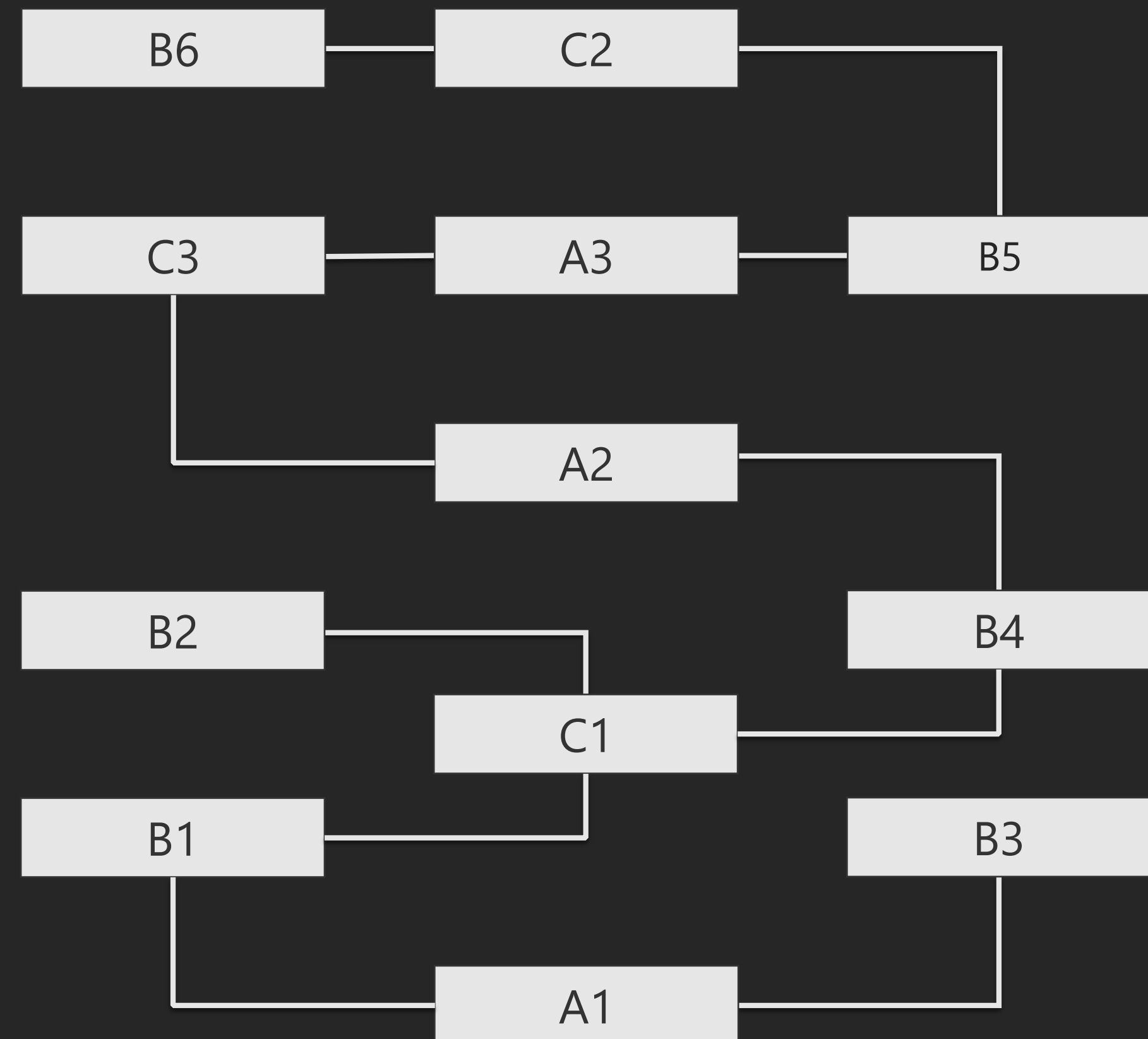
Why use graphs?

Use graphs when:
- Shape of the domain is naturally a graph
- Problems with join performance
- Joining more than 7 tables together
- Majority of your tables are junction tables
- Written stored procedures with multiple recursive self and inner joins
- Continuously evolving data set (often means creating wide and sparse tables)
- Constantly changing schema
- Require combing multiple schemas

Graphs are good at:
- Path finding (how do people know each other)
- Link analysis (fraud detection)
- Highly connected data (social networks)
- Recommendations (e-commerce)
- A* (least-cost path analysis)

Graphs are designed to:
- Store interconnected data
- Make it easy to make sense of data connections
- Make it easy to evolve the database
- Enable high-performance on operations for:
  - Discovery of connected data patterns
  - Relatedness queries > depth 1

Why use graphs?

Depth

Size

Graphs

RDBMS

Document

Column Family

Key-Value
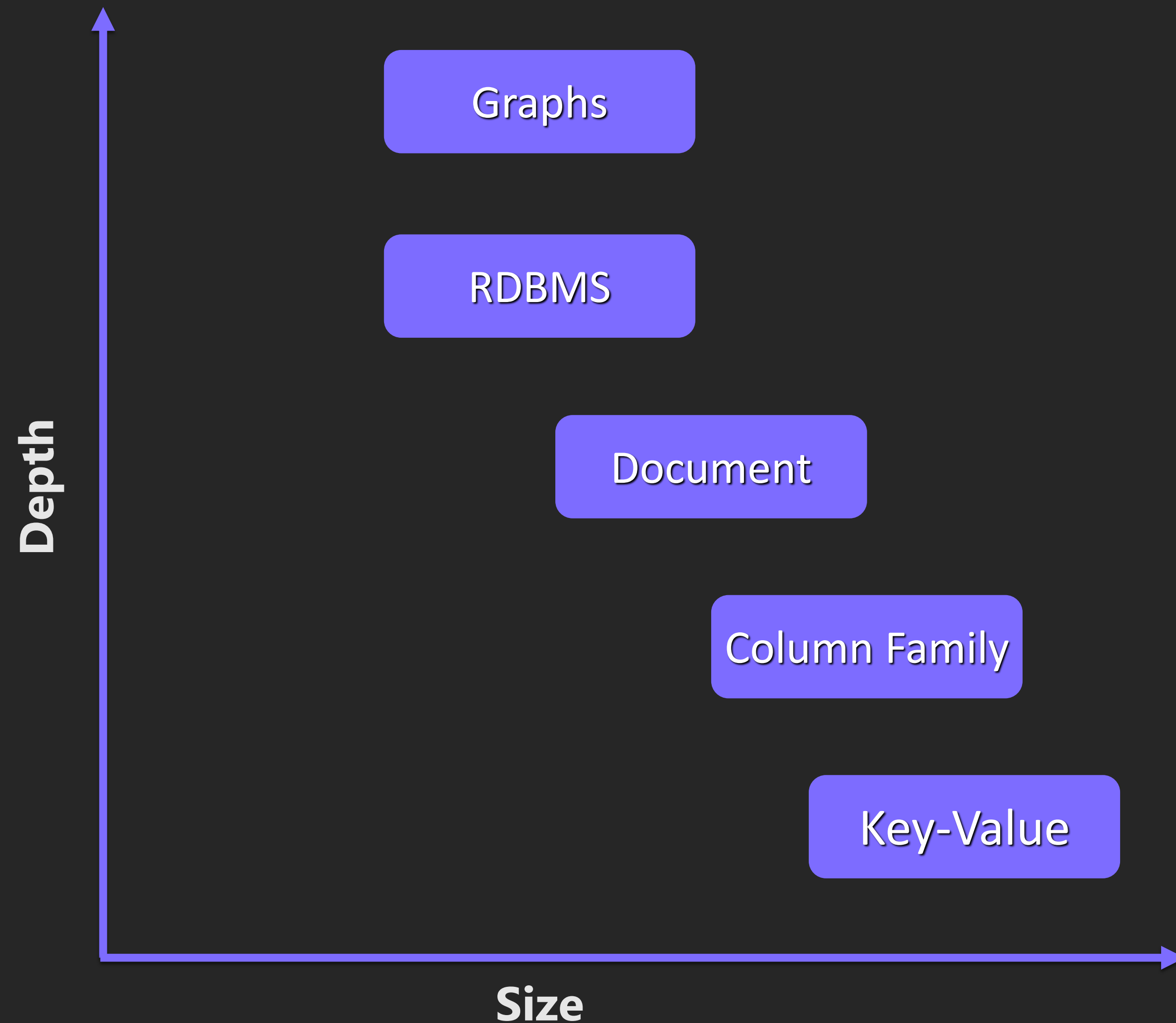
Friends-of-friends
1,000,000 people
~50 friends each

Depth	RDBMS execution time (s)	Neo4j execution time (s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110k
4	1543.505	1.359	~600k
5	Unfinished	2.132	~800k

Friends-of-friends
~50 friends each
Depths of 4

Database	# persons	Query time
MySQL	1,000	2 sec
Neo4j	1,000	2 ms
Neo4j	1,000,000	2 ms

Why use graphs?

Query Response Time
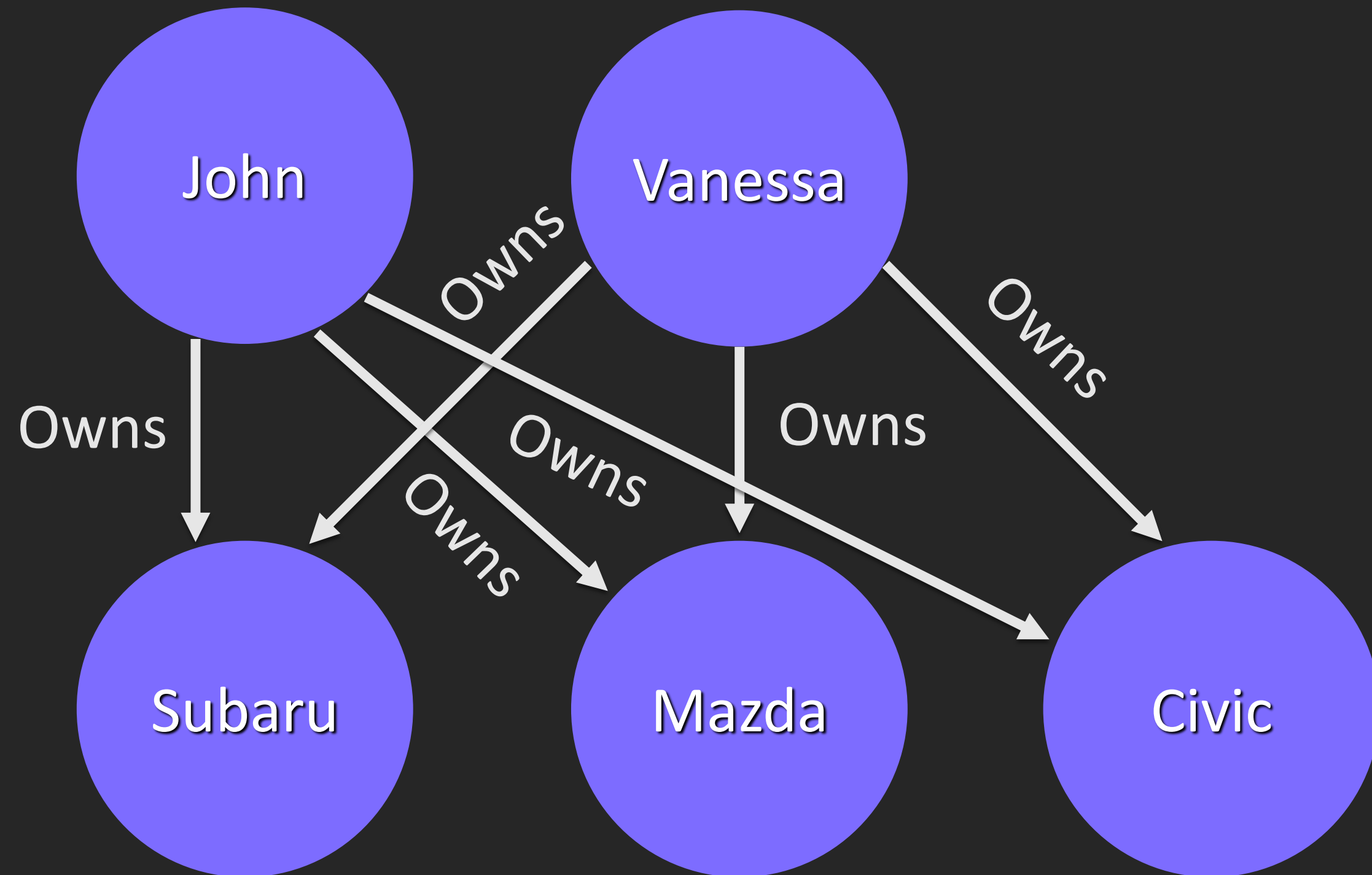   = **f**(graph density, graph size, query degree)

- Graph density (avg # rel's / node)
- Graph size (total # of nodes in the graph)
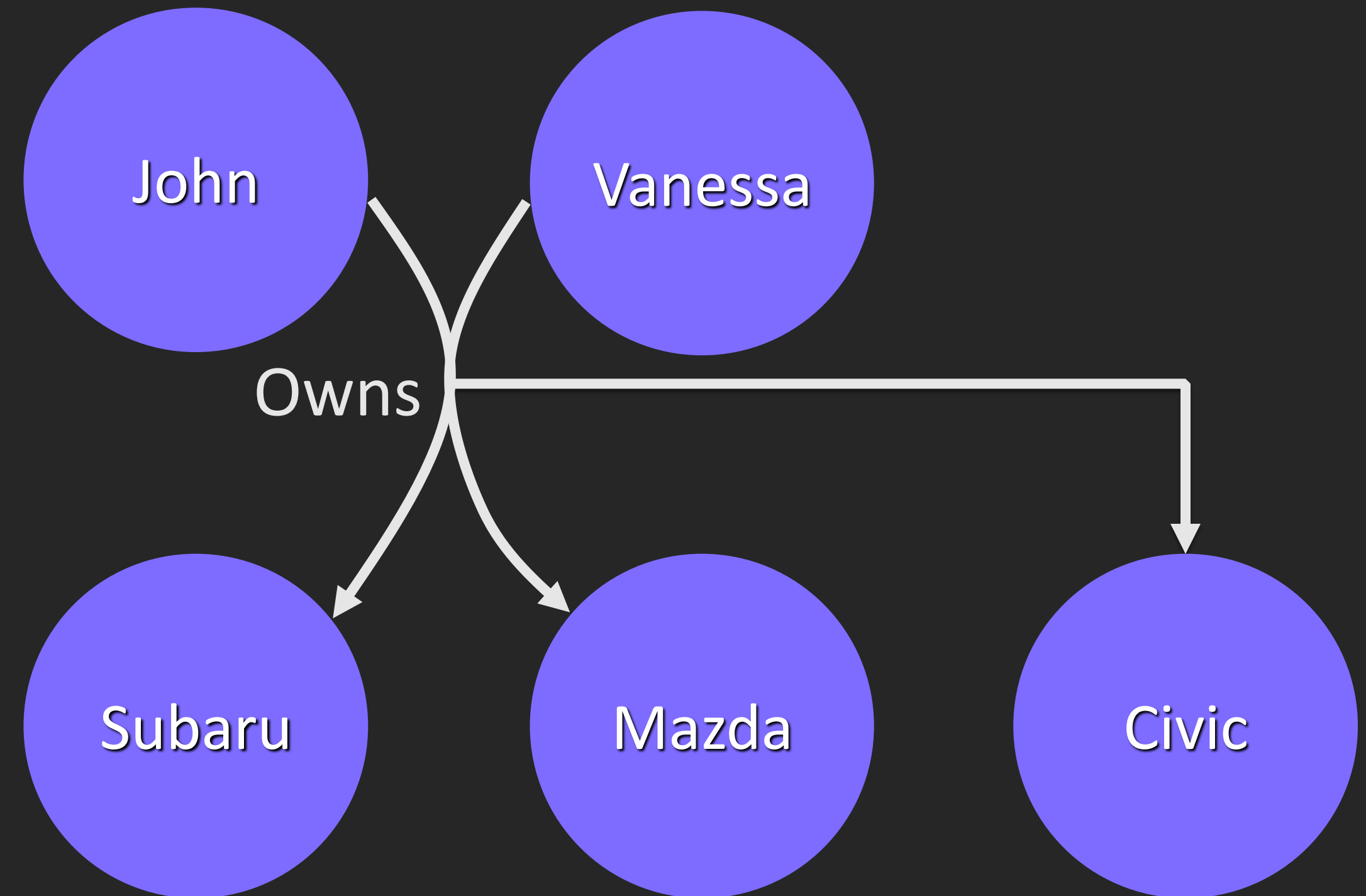- Query degree (# of hops in one's query)
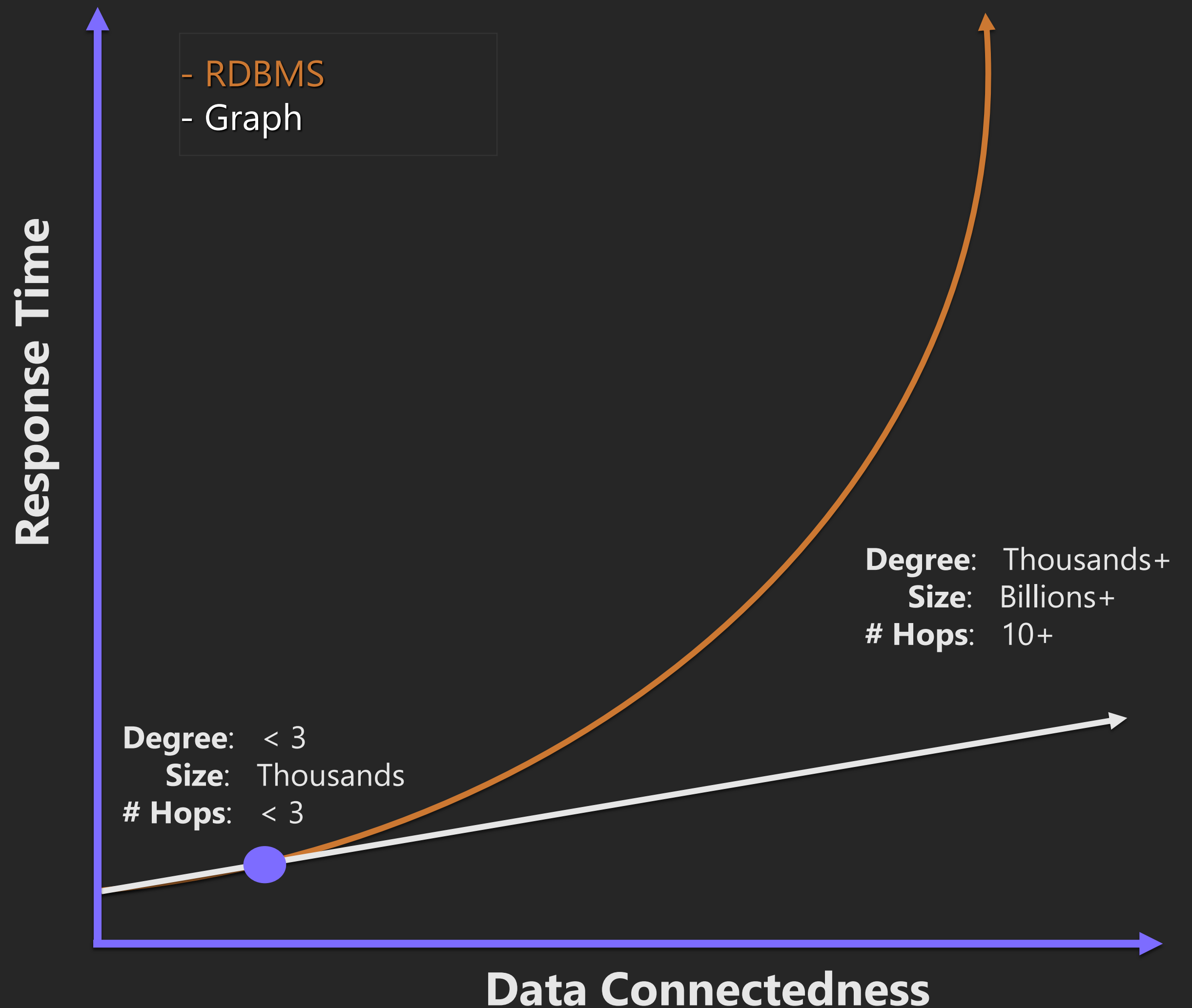
Are hypergraphs useful?

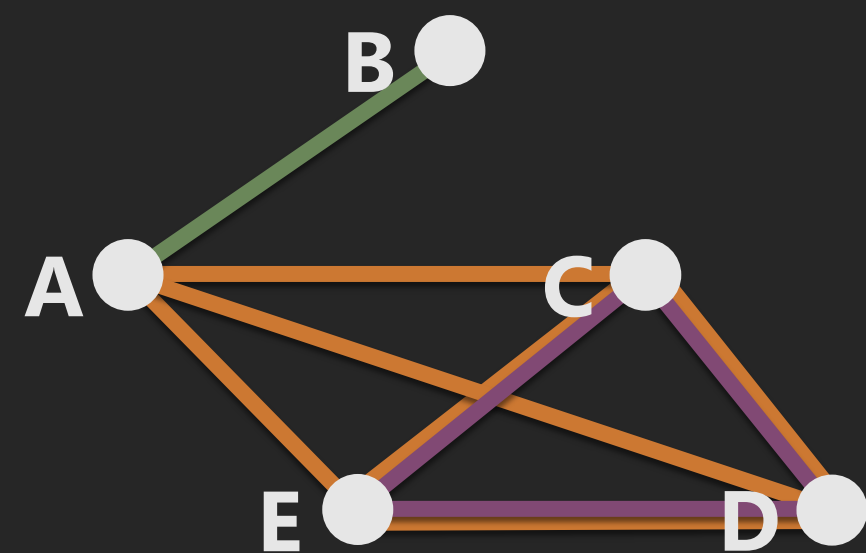# Are hypergraphs useful?

Directed Graph
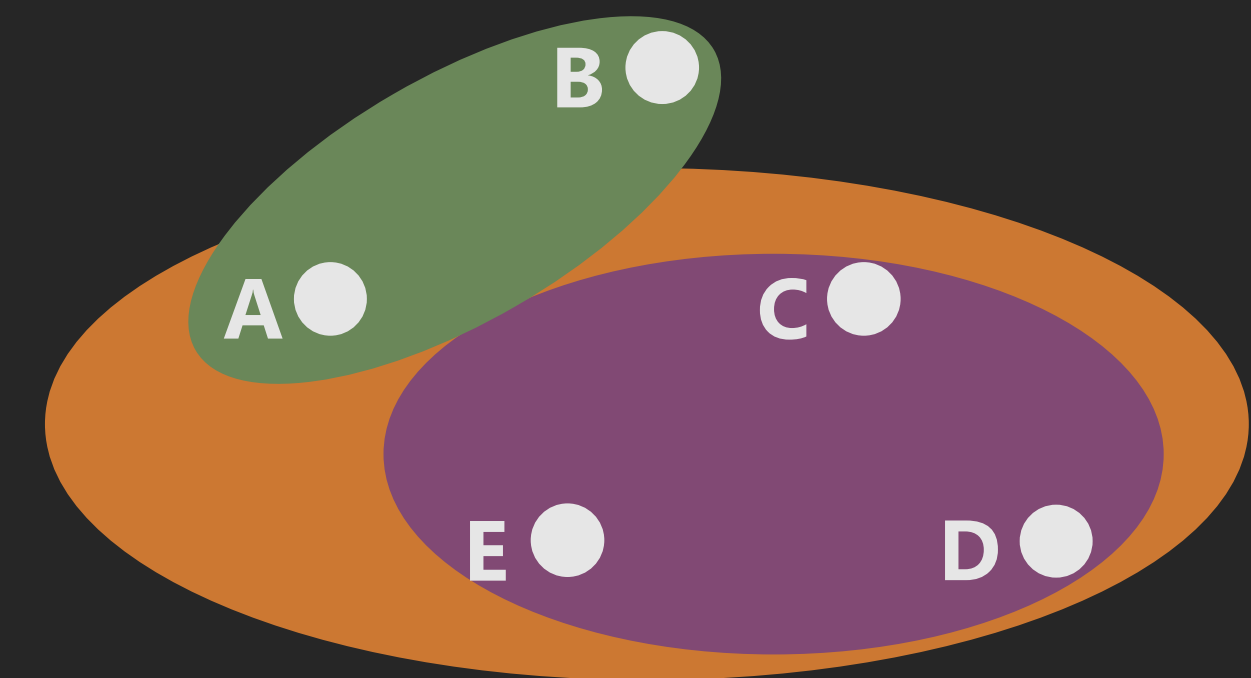
Hypergraph

Are hypergraphs useful?

Graph

	1	2	3	4	5	6	7	8	9	10
A	X				X	X	X			
B	X									
C		X	X		X			X	X	
D		X		X		X		X		X
E			X	X			X		X	X

Hypergraph

	1	2	3
A	X		X
B	X		
C		X	X
D		X	X
E		X	X

Hypergraphs generalize the common notion of graphs by relaxing the definition of edges
Graph: Edge = pair of vertices
Hypergraph: Hyperedge = set of vertices

Scenario – Traditional Marriage

Marriages

husband_id	husband_id	wife_id
h1	m1	w1

Husbands

husband_id	name
h1	John

Wives

wife_id	name
w1	Vanessa

Relational

John — married → Vanessa

Directed

Scenario – Polygamous Marriage

Marriages

husband_id	husband_id	wife_id
h1	m1	w1

Husbands

husband_id	name
h1	John

Wives

wife_id	name
w1	Vanessa

~~Relational~~

John — married → Olivia

John — married → Vanessa

John — married → ...
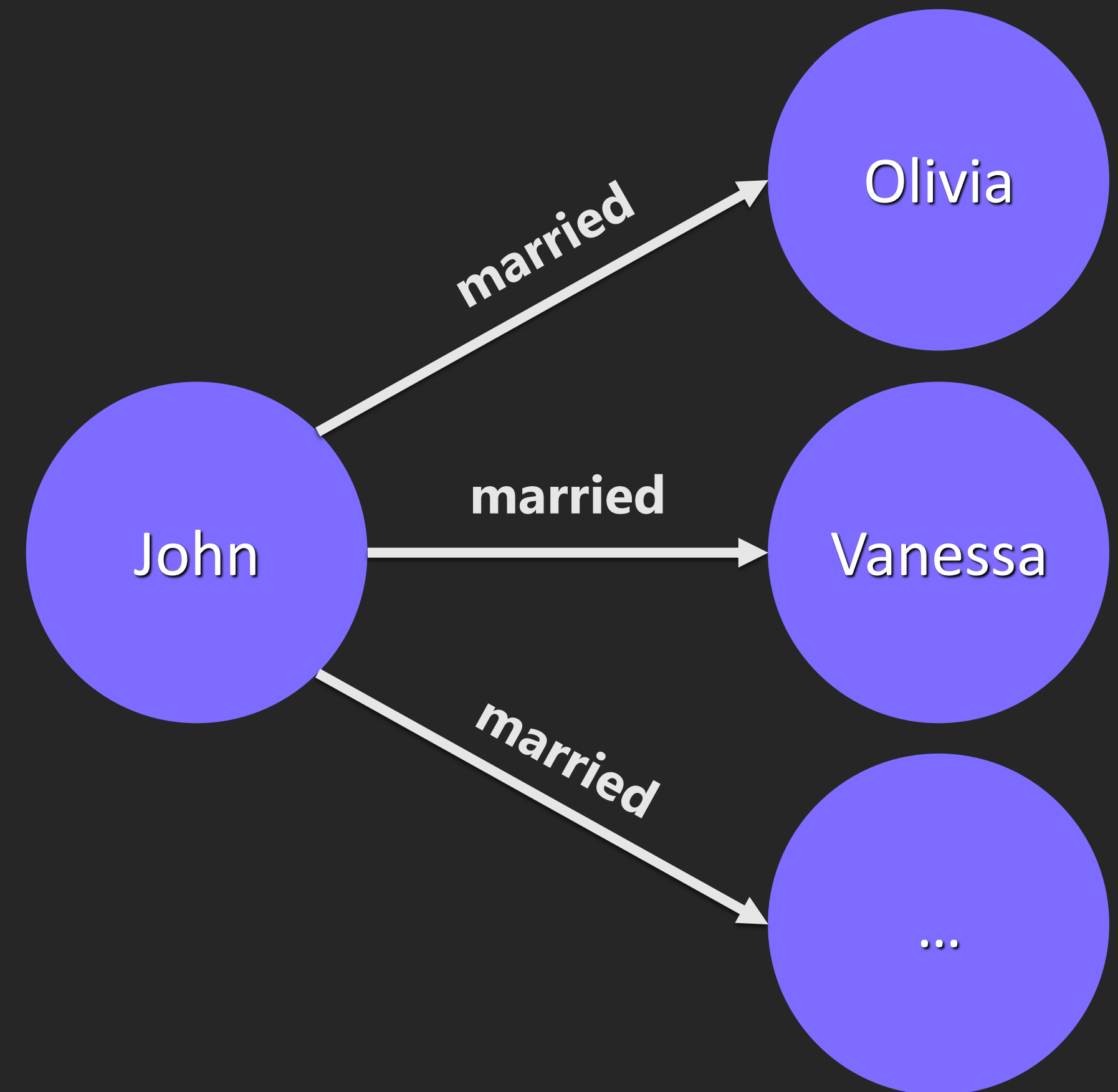
Directed

Scenario – Divorce Filing
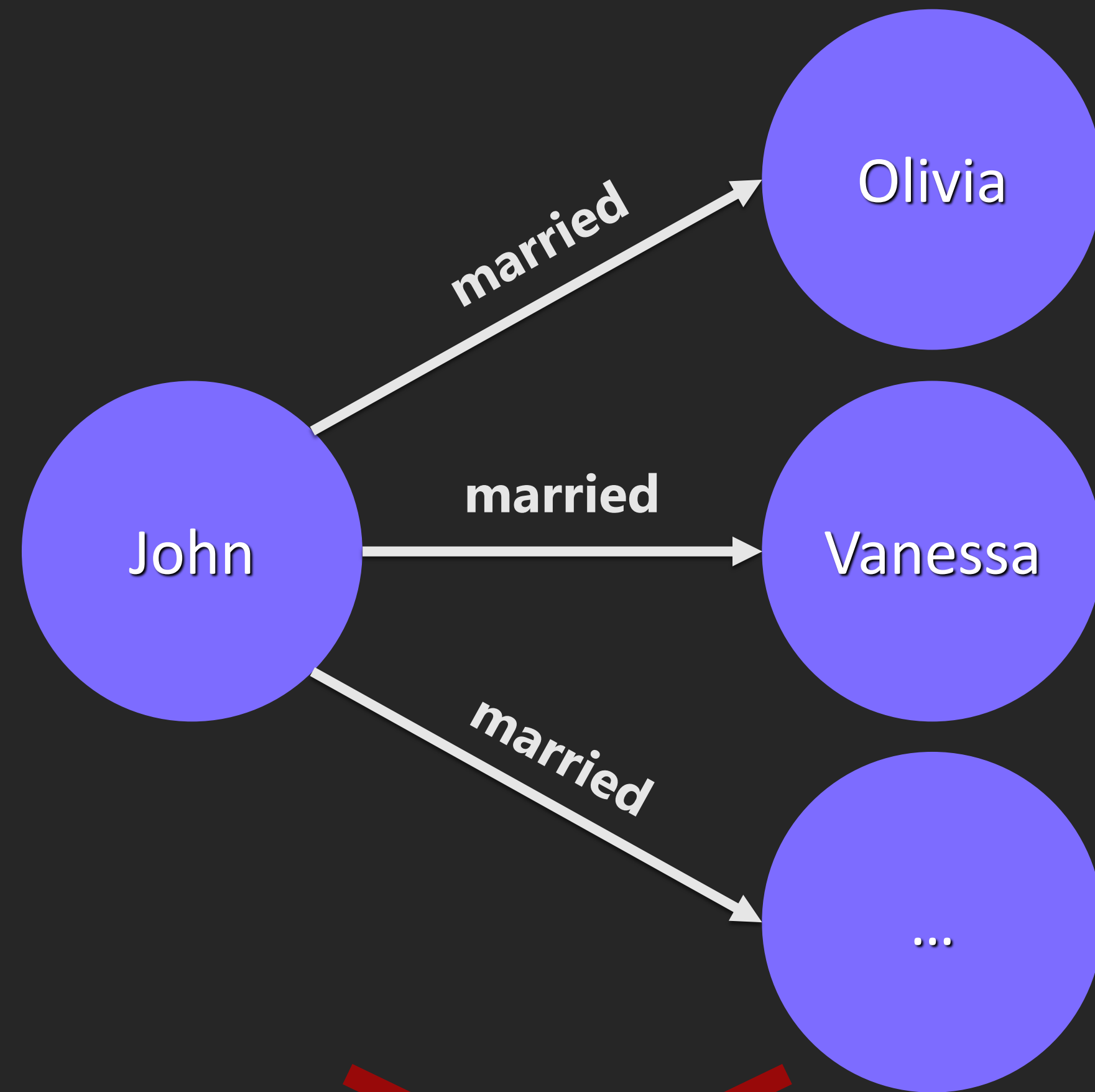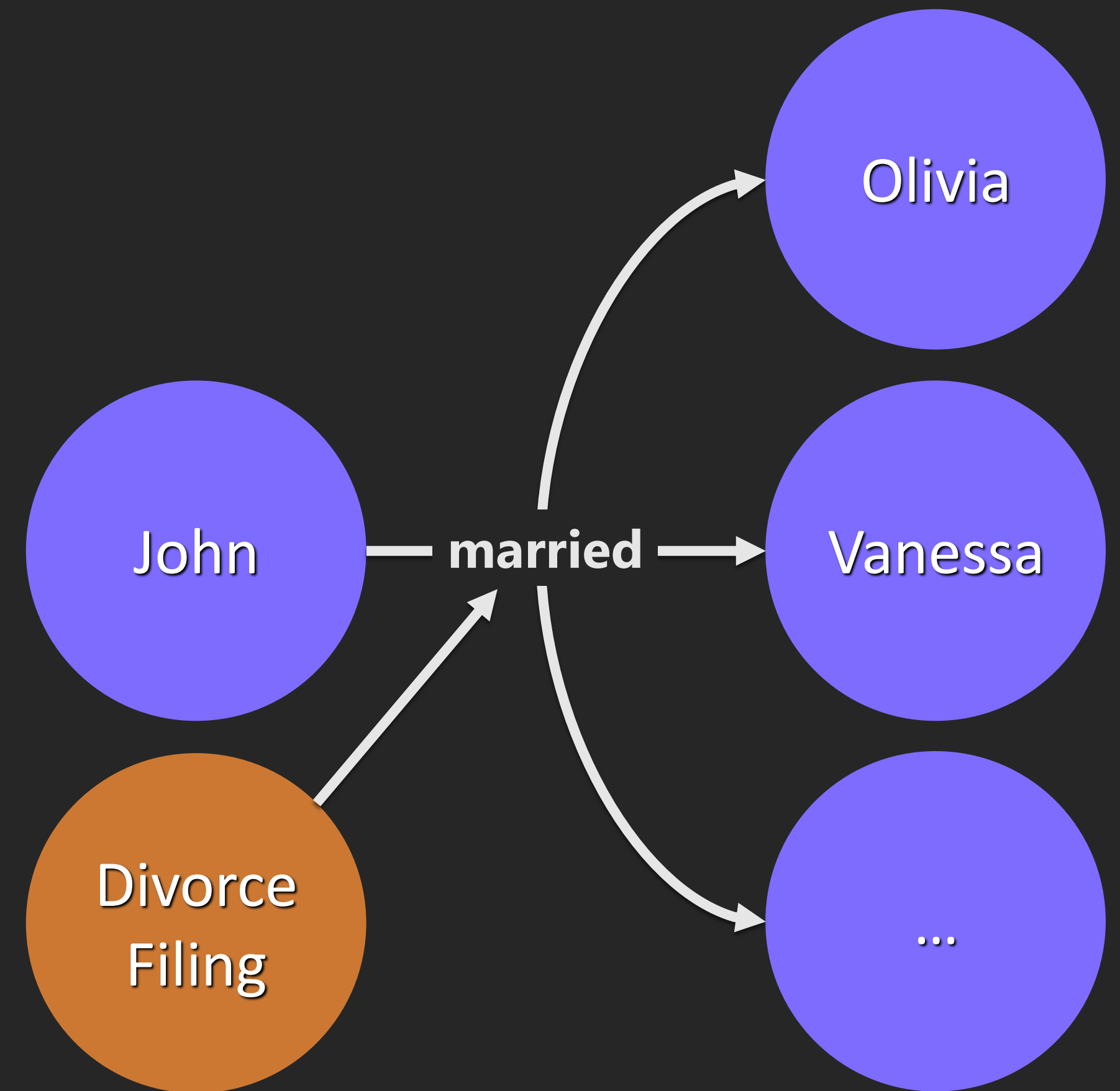
How do I model a graph?

Data Modeling

Conceptual Model

Logical Model

Physical Model

Technical Expertise

Abstraction

Conceptual Model

Entities
- Attribute value comprises a **complex value** type (e.g. address)
- Values with conceptual identities
- Value requires qualification via relation

- Example:
    - Find all recent orders delivered to the **same delivery address** (complex value type)

Relations
 - Specify **weight**, **strength**, or some other **quality** about the relationship

- Example:
    - Find all my colleagues who are **level 2 or above** (relationship quality) in a **skill** (attribute value) we have in common

Attributes
- There's no need to qualify the relationship and consists of a **simple value** type (e.g. color)
- Have no conceptual identity (metadata)

- Example:
    - Find those projects written by contributors to my projects that use the same **language** (attribute value) as my projects

Determine Entities

Which library users have books currently lent which are over-due?

Determine Entities

Which library users have books currently lent which are over-due?

Determine Relations

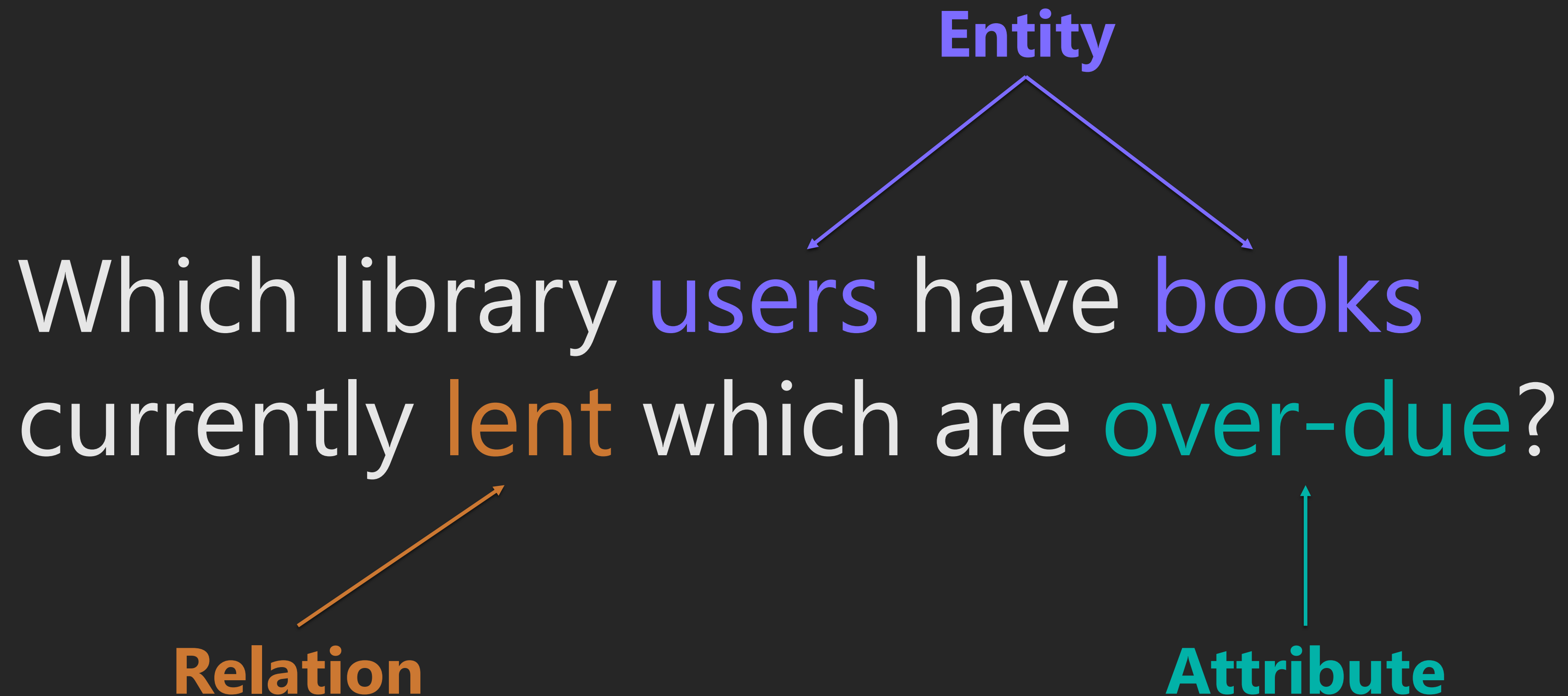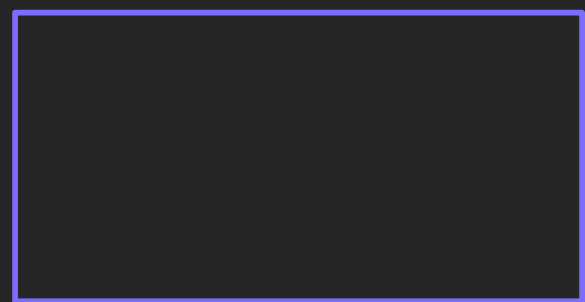Which library users have books currently lent which are over-due?

Determine Relations

Which library users have books currently lent which are over-due?

Determine Attributes

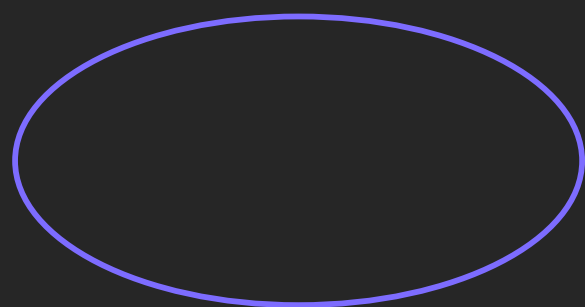Which library users have books currently lent which are over-due?

Determine Attributes

Which library users have books currently lent which are over-due?

Conceptual Model

Entity

Which library users have books currently lent which are over-due?

Relation

Attribute

Entity–Relationship Model

Symbol	Meaning	Example
▭	Entity	Employee
⬭	Attribute	Name
◇	Relationship	Knows
—	Link	Knows — Employee — Name
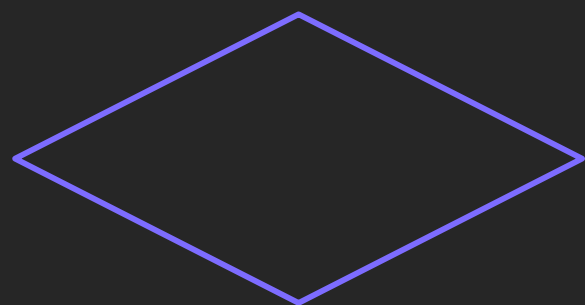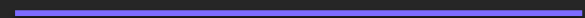
Draw Conceptual Model

Objective
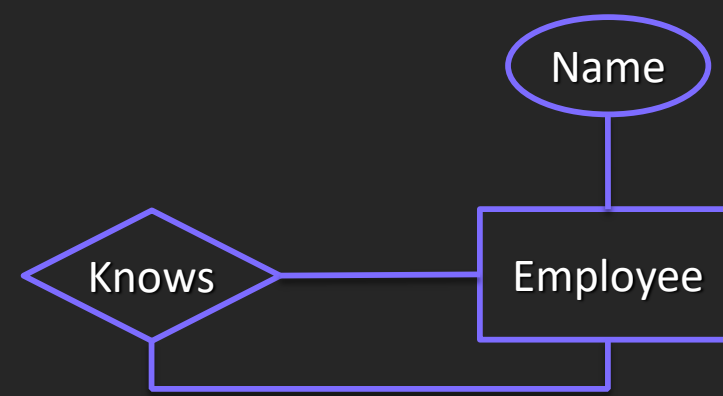
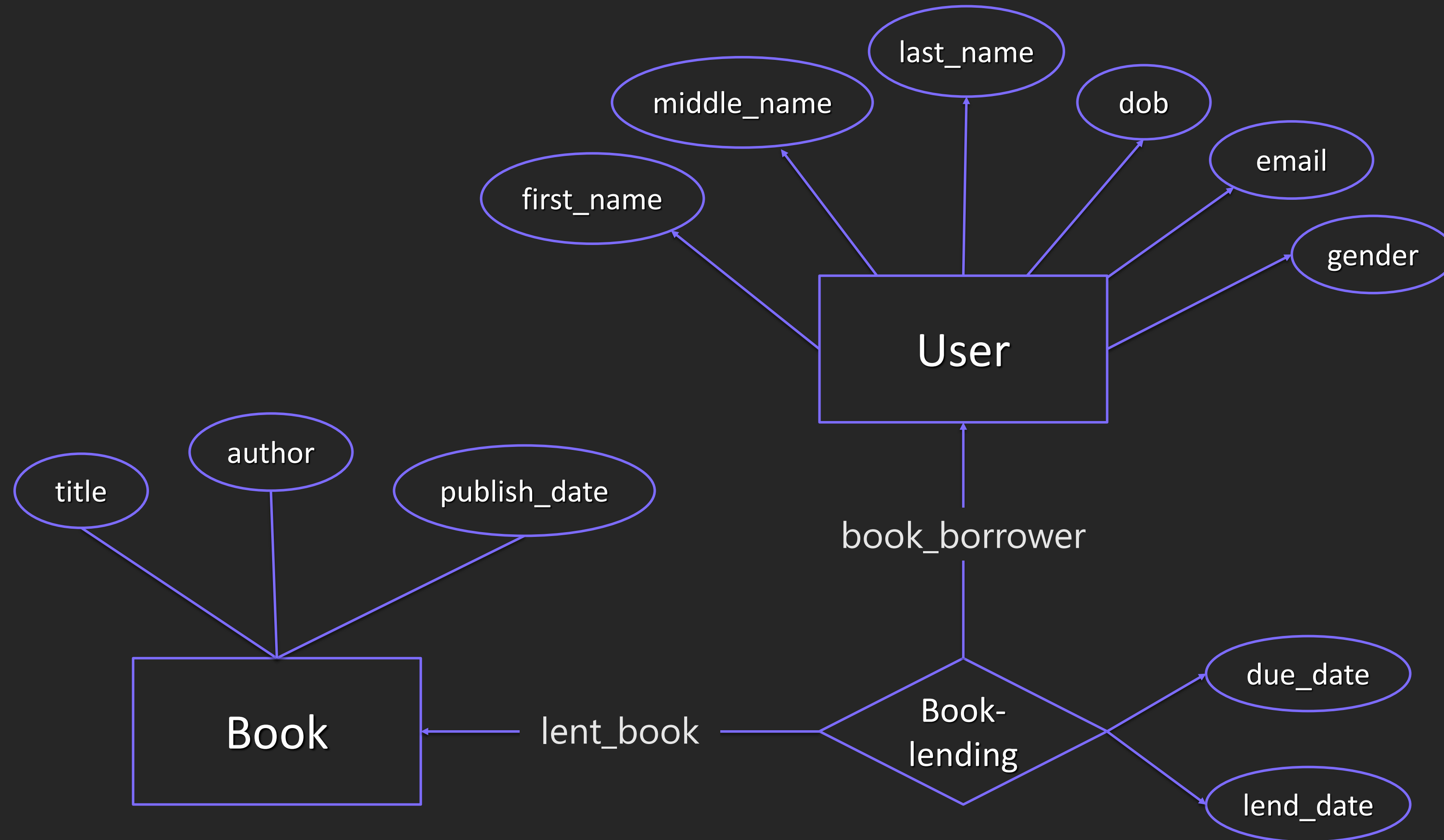Which library users have books currently lent which are over-due?

Data Available

Book
- Author
- Title
- Publish date
- Lend date
- Due date

User
- First/middle/last name
- DOB
- Email
- Gender

Draw Conceptual Model

Conceptual Model

Data Modeling

Data Modeling

Normalization

Conceptual Model

Logical Model

Physical Model

Data Modeling
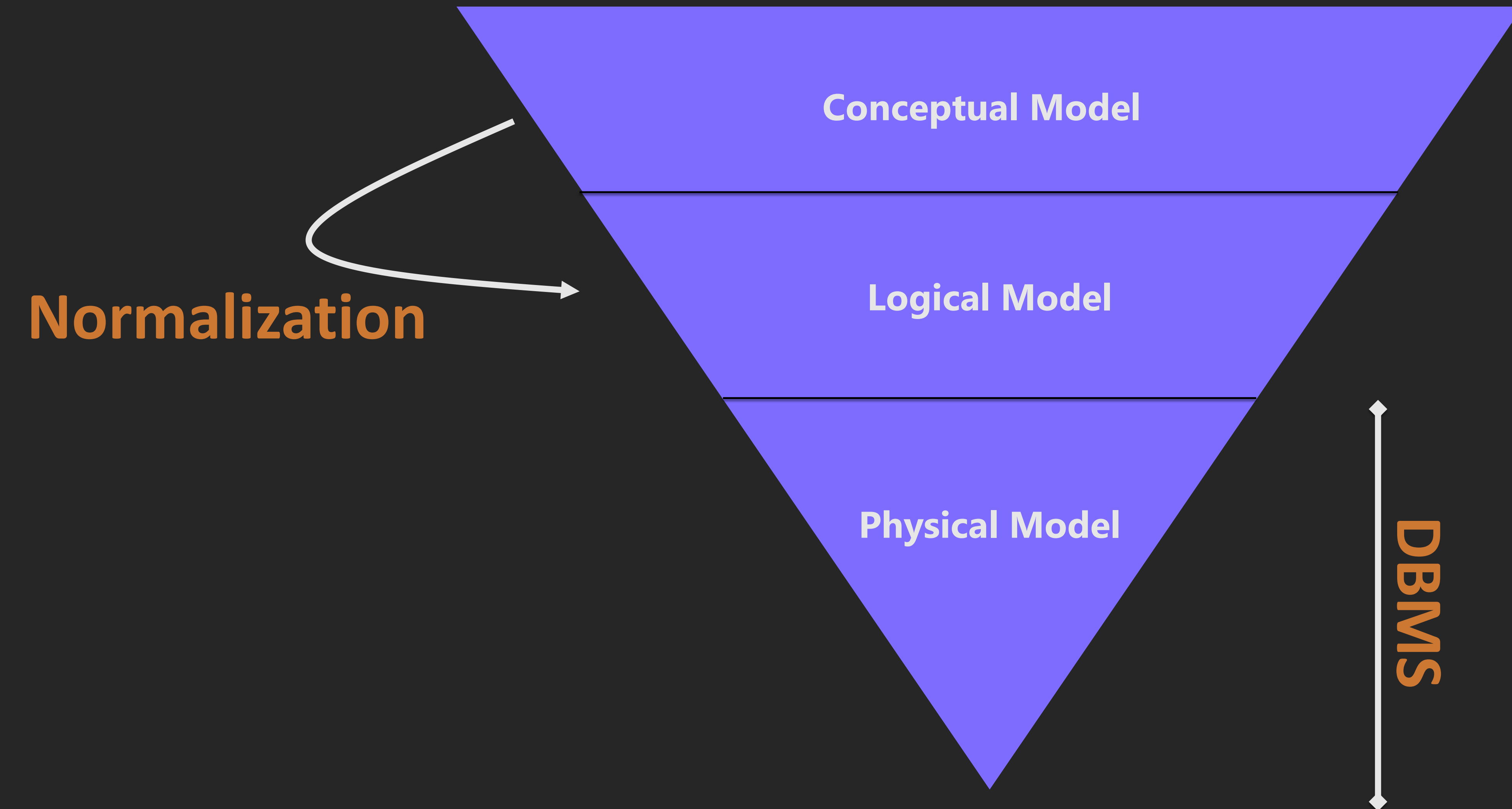
Users

user_id
first_name
middle_name
last_name
dob
email
gender

BookLendings

user_id
book_id
lend_date
due_date

Books

book_id
title
author
publish_date

Data Modeling

Introducing Grakn
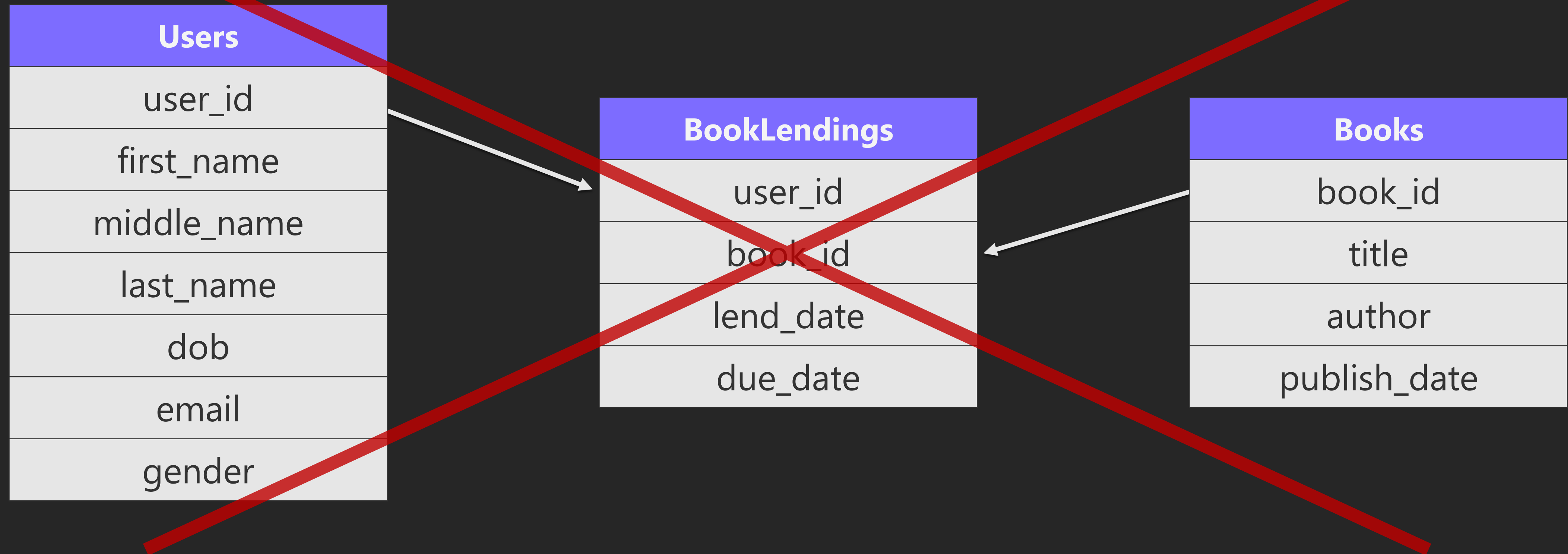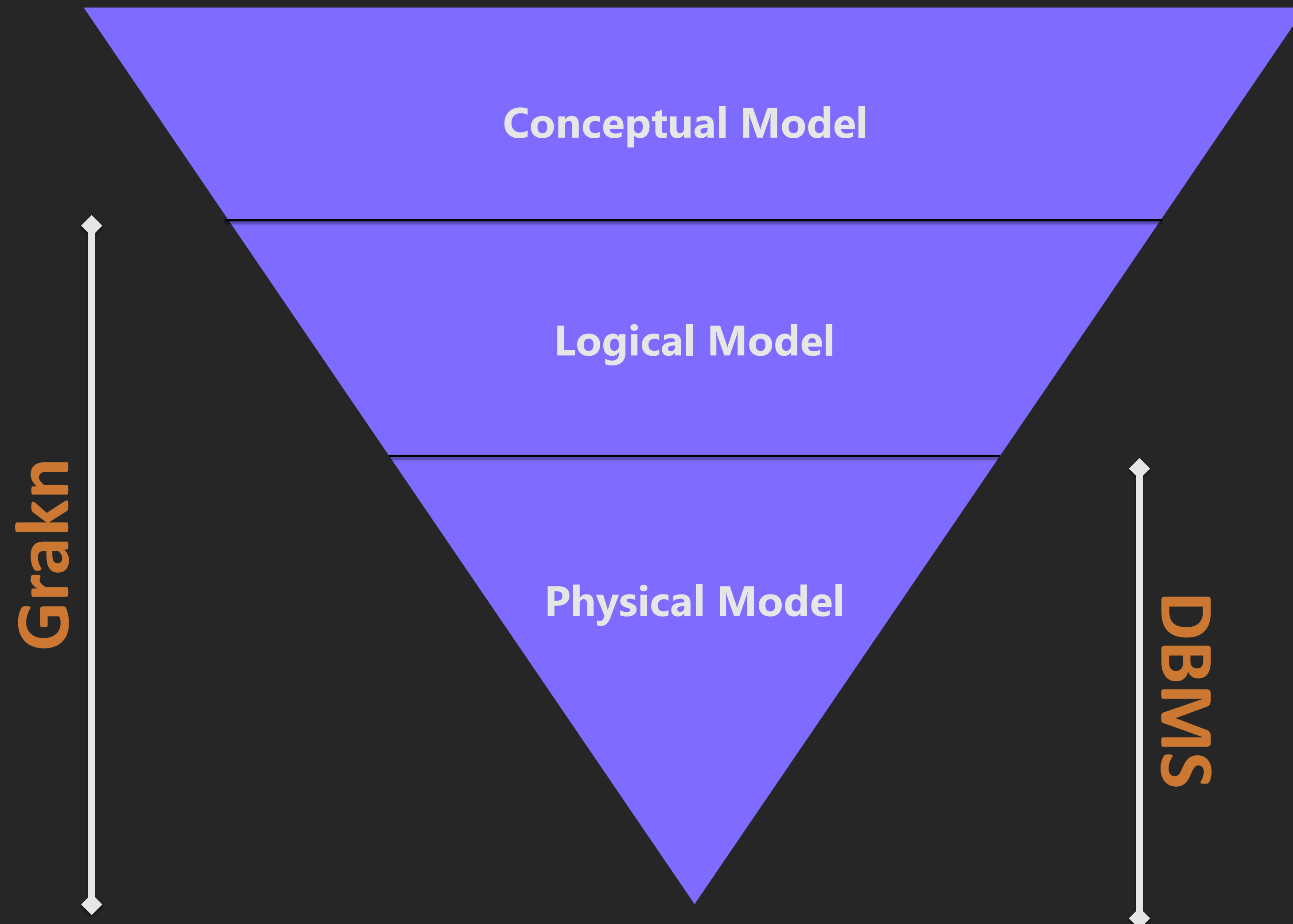
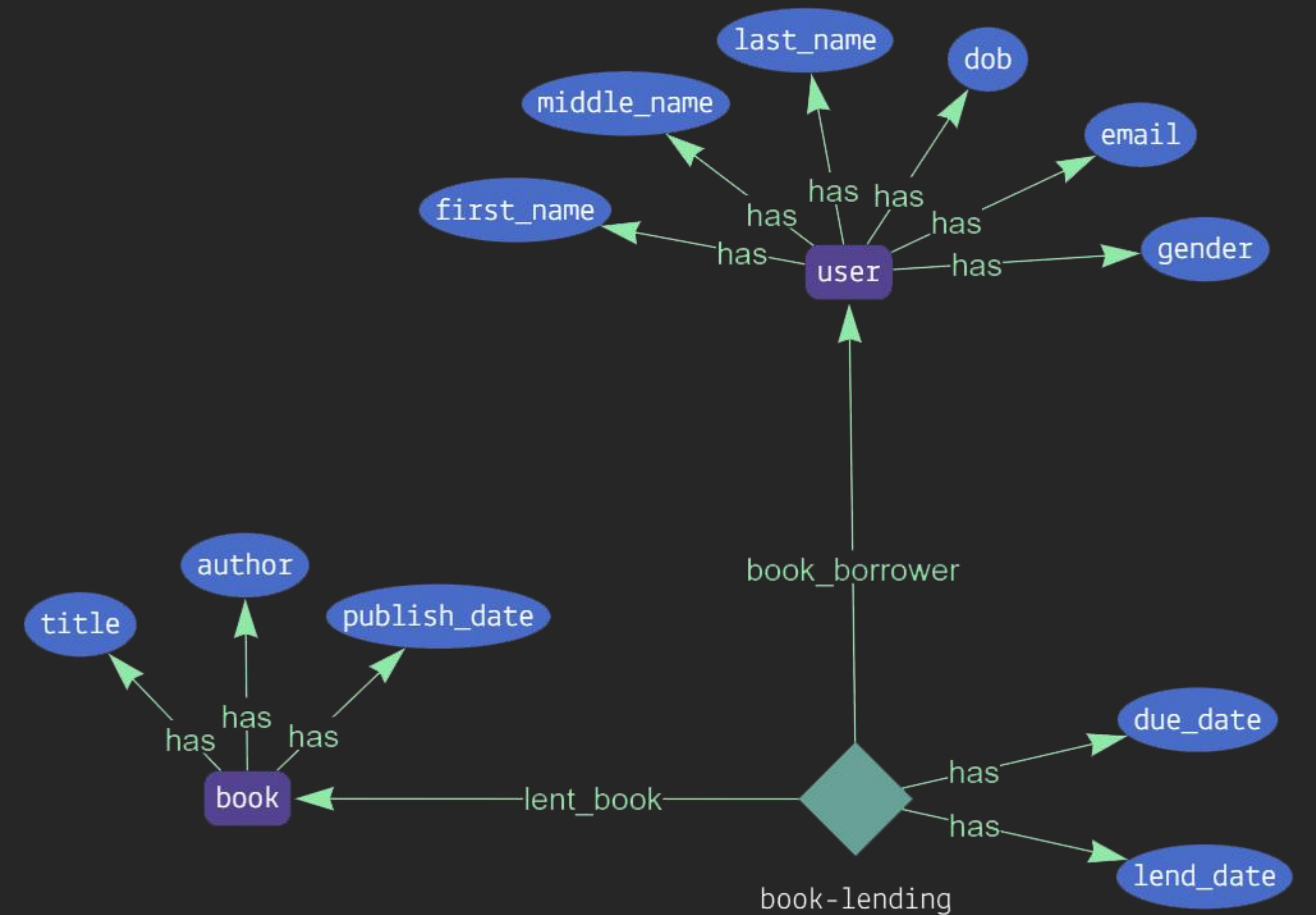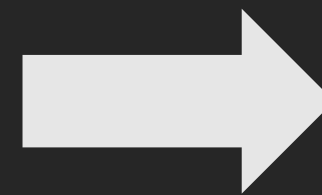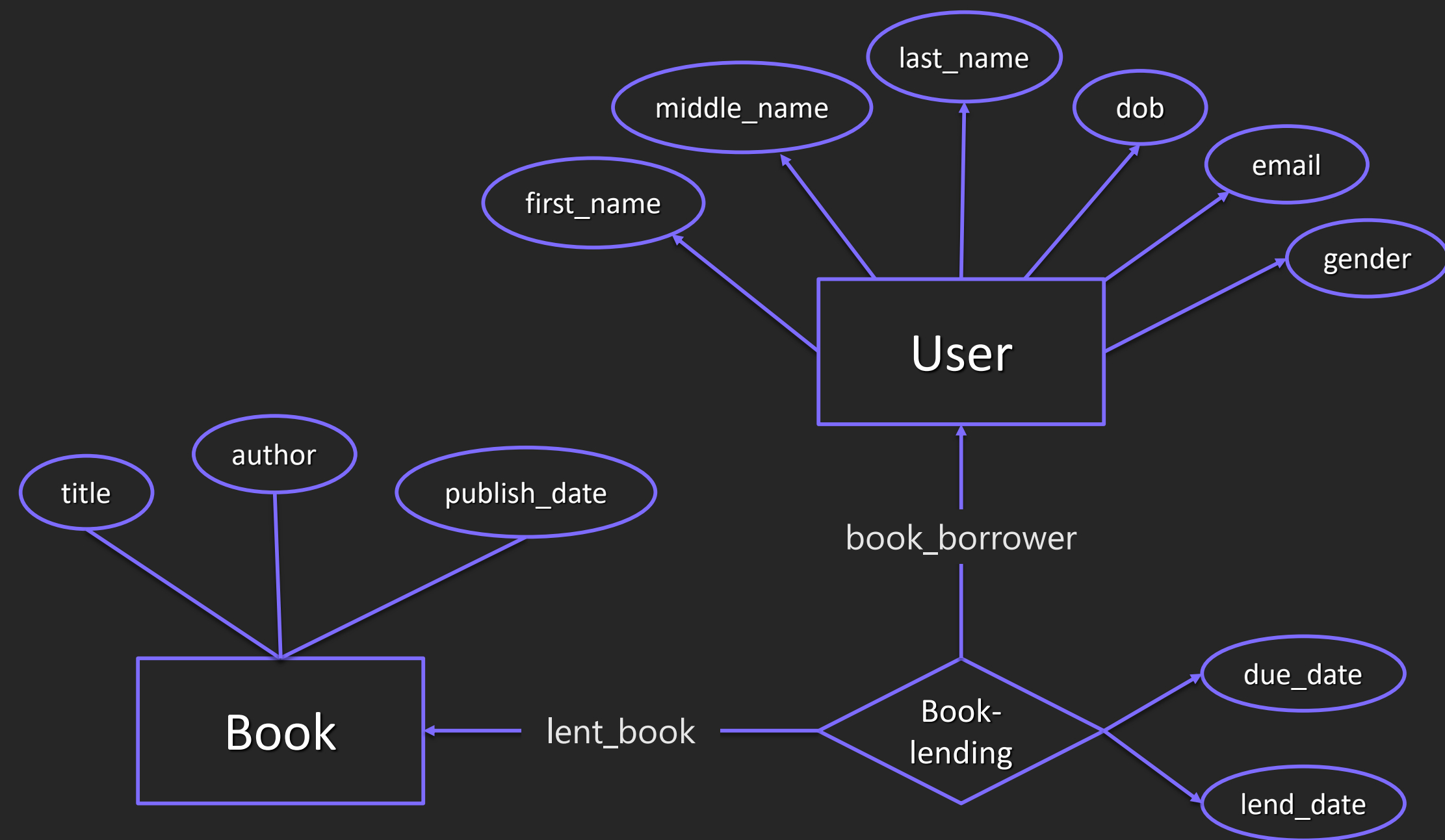# Data Modeling

Data Modeling

Data Modeling
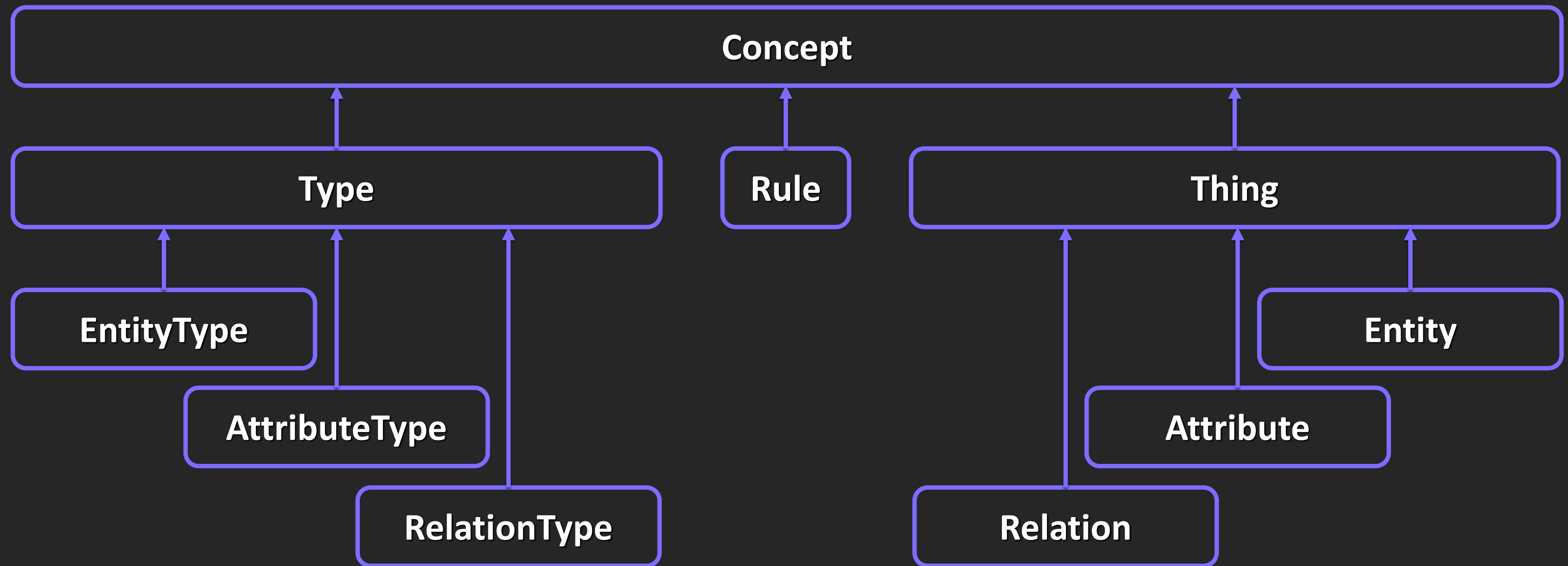
Knowledge Model

A schema which can represent:
type hierarchies, hyper-relations and rules
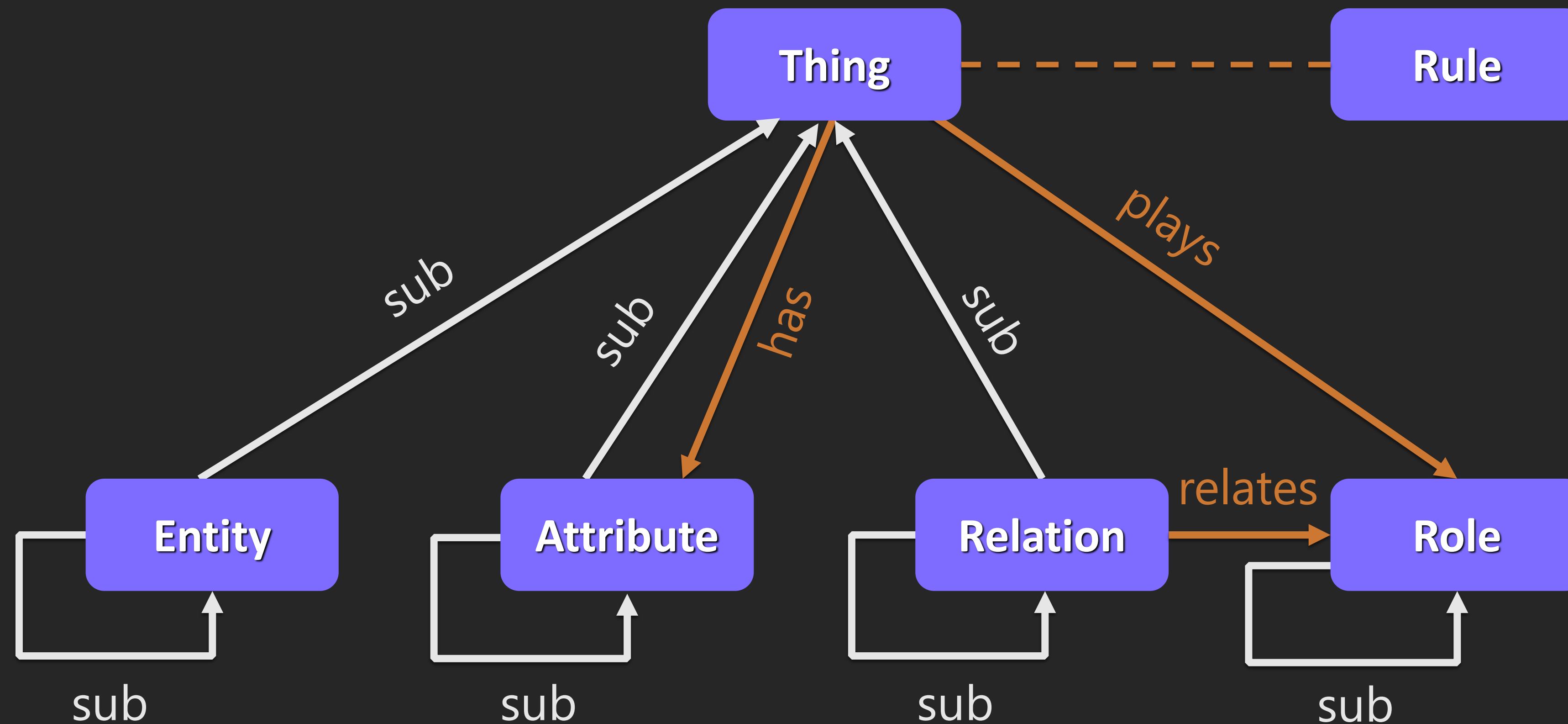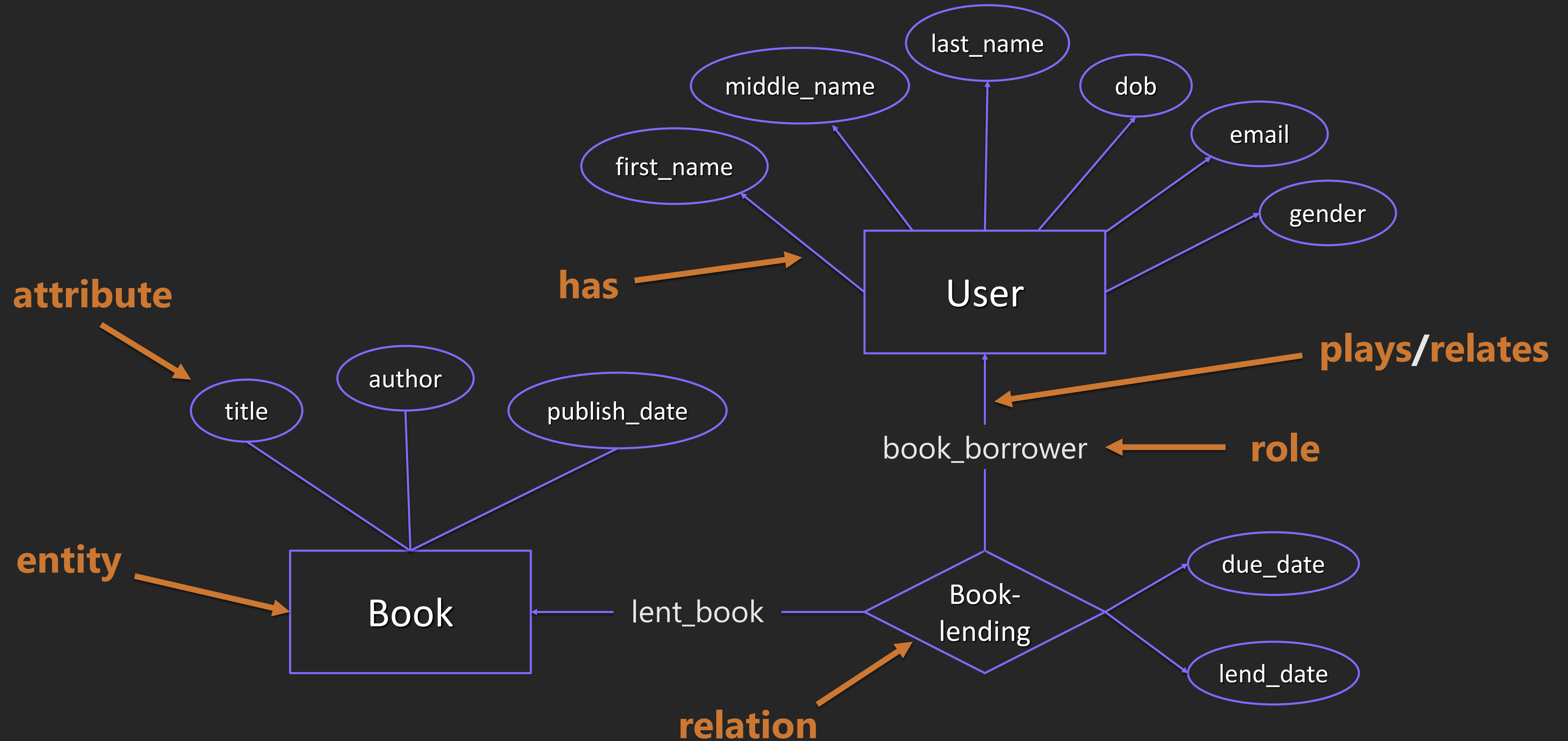
Grakn Modeling – Hierarchies

Entity Type Hierarchies
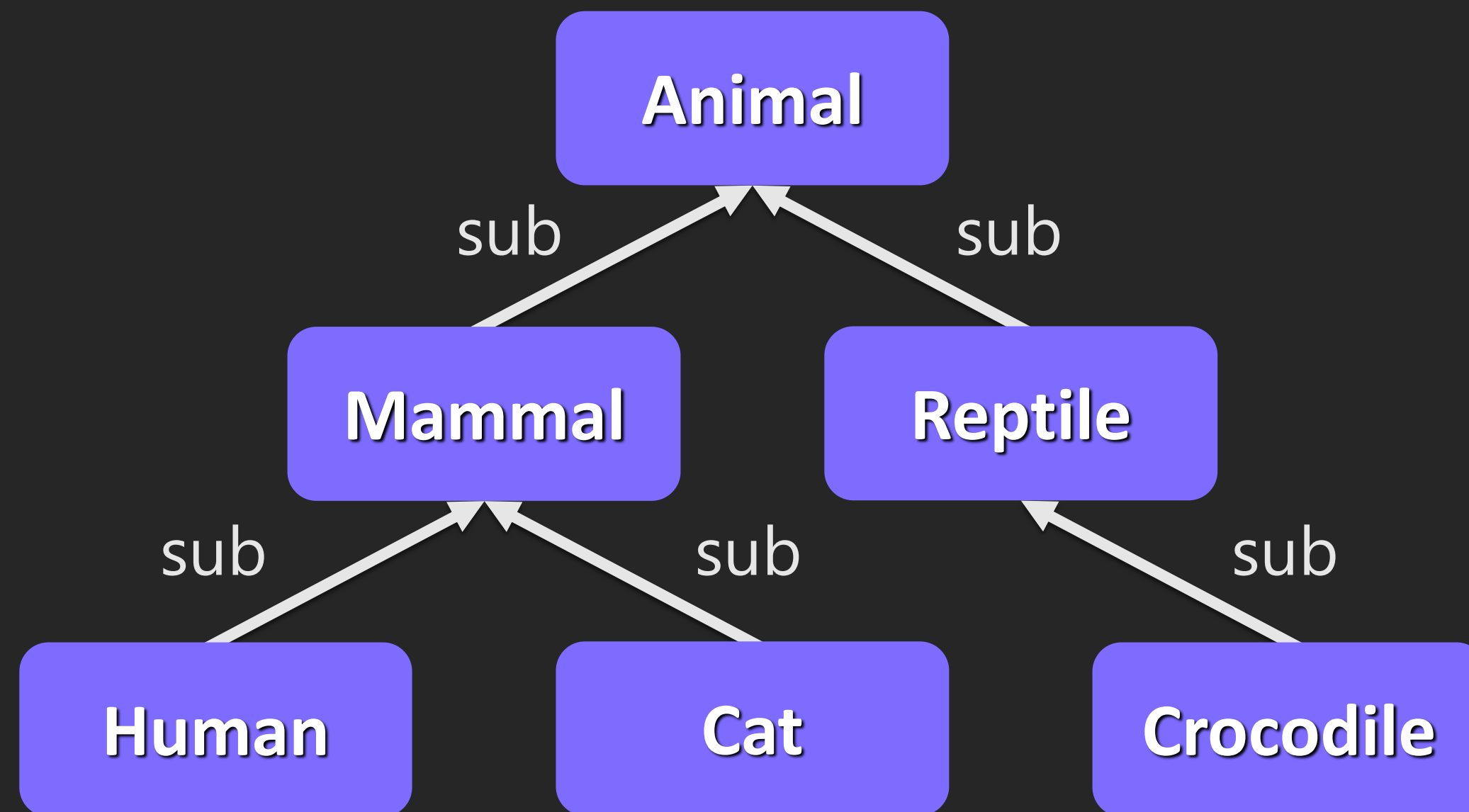
Relation Type Hierarchies
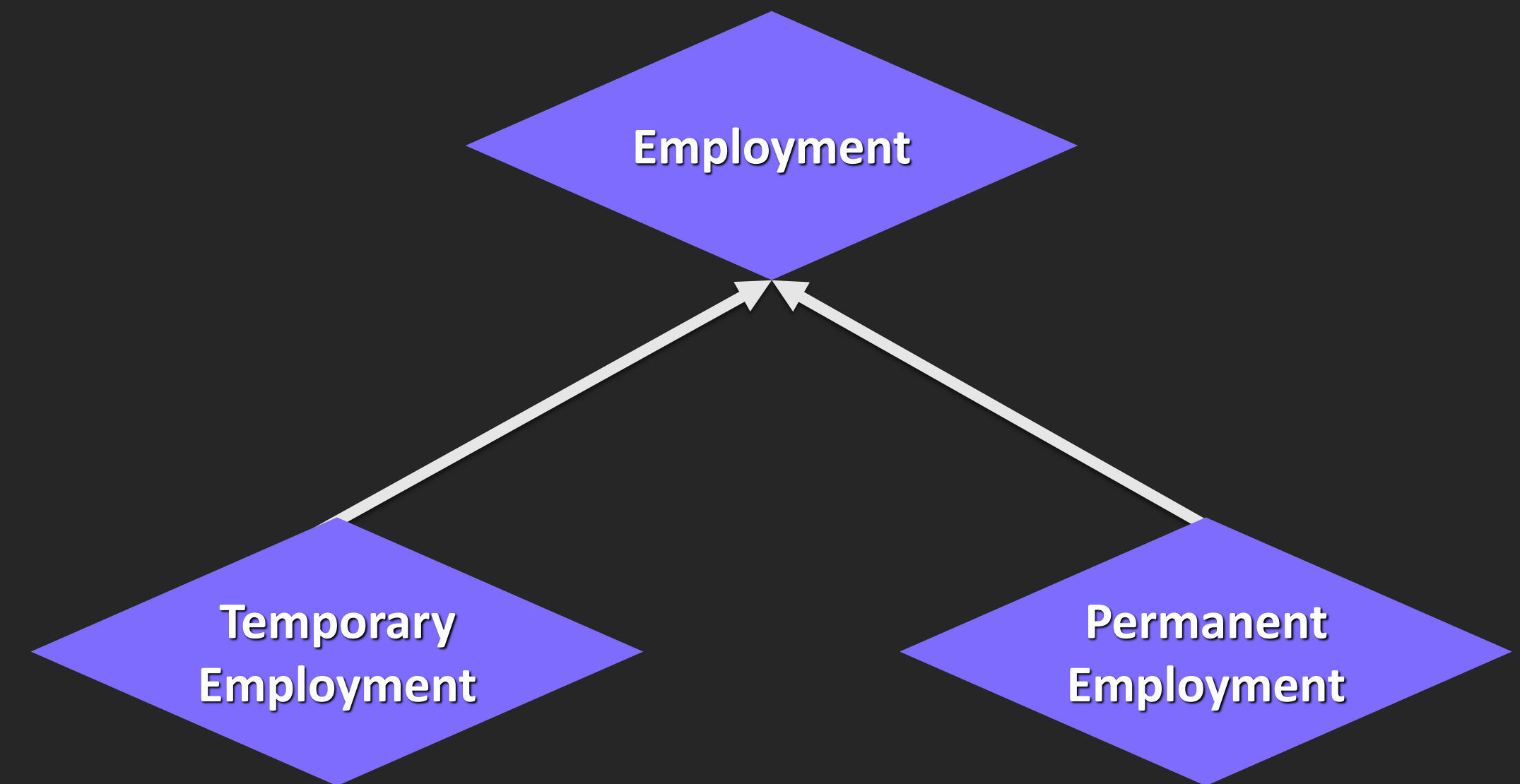
Grakn Modeling – Relations

Ternary Relations

Product

Purchase

Buyer

Seller

Infinitary Relations

Venture

Joint Venture

Owner

Owner

Owner

Owner

…

Grakn Modeling – Relations
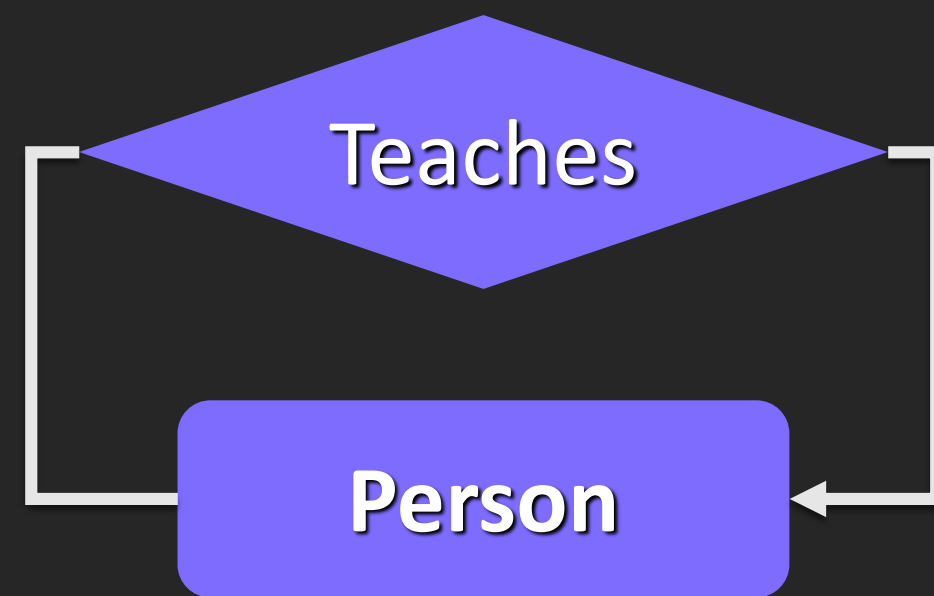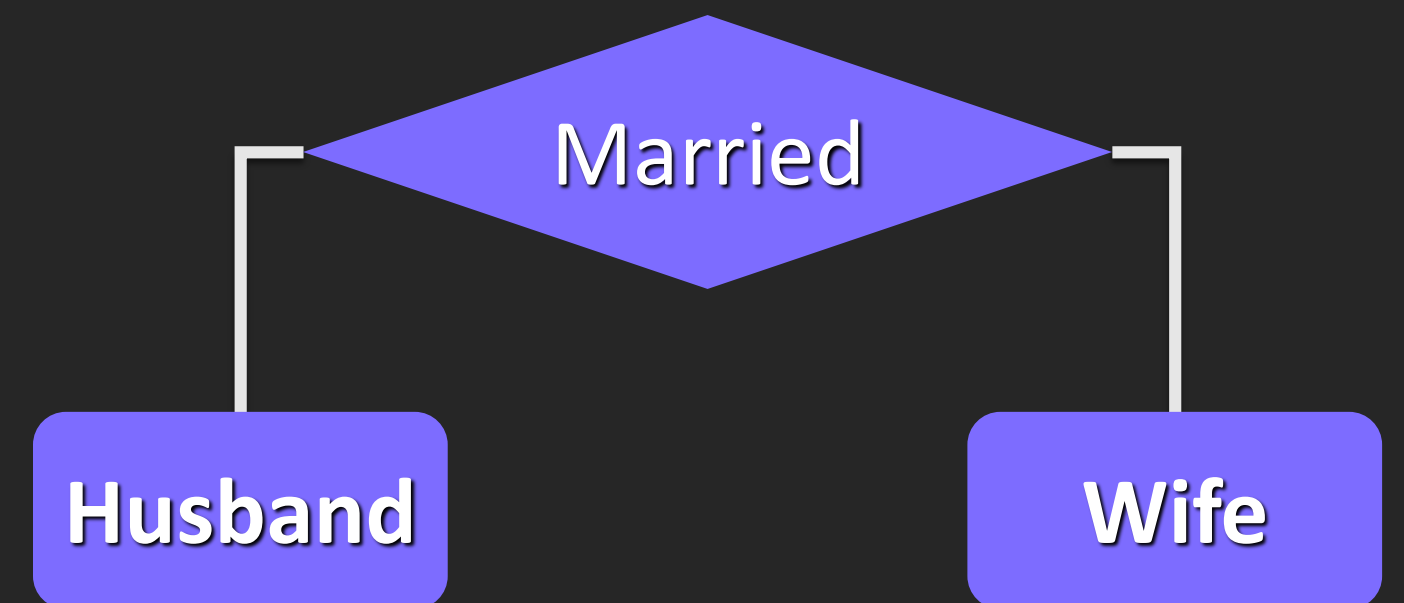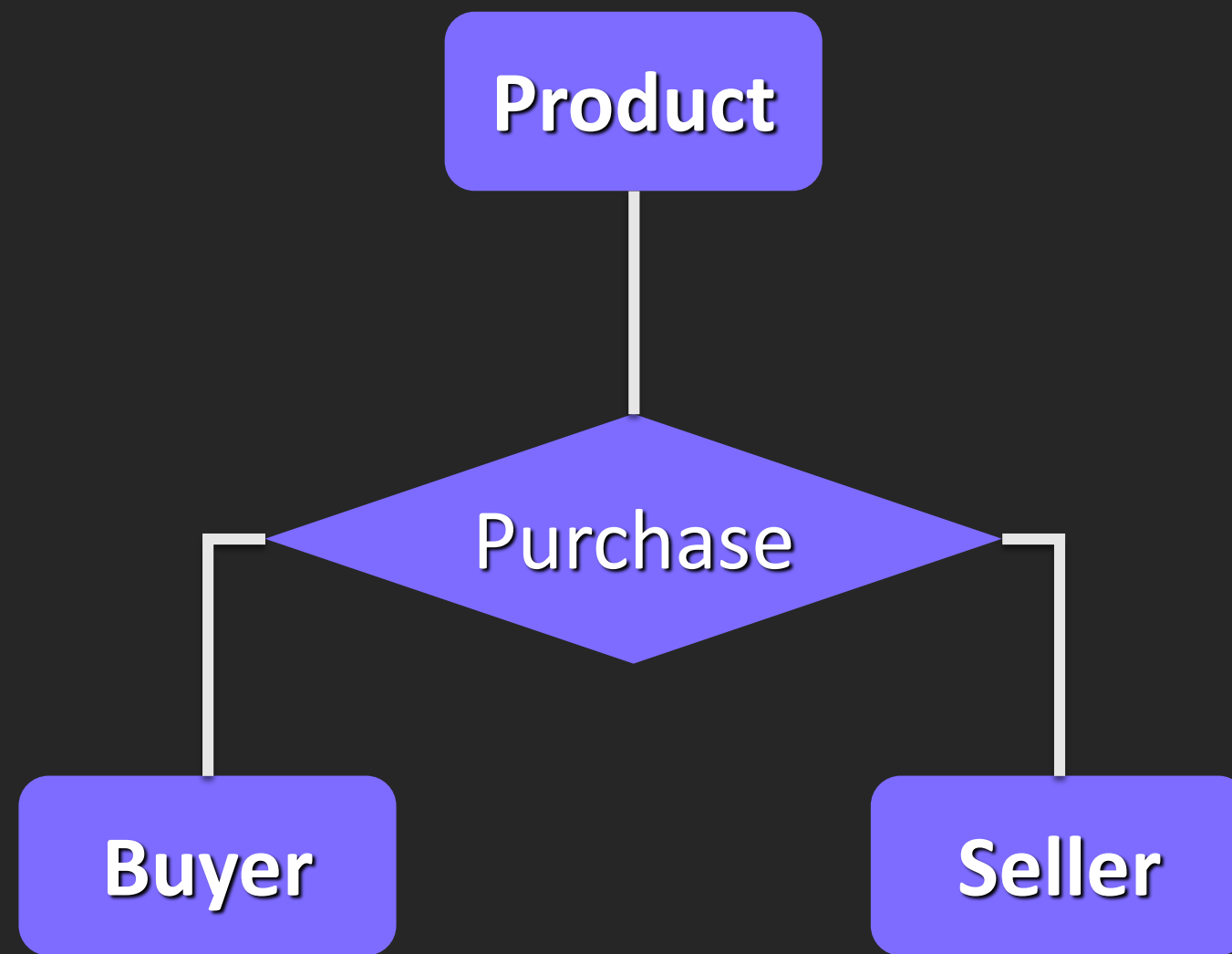
Nested Relations
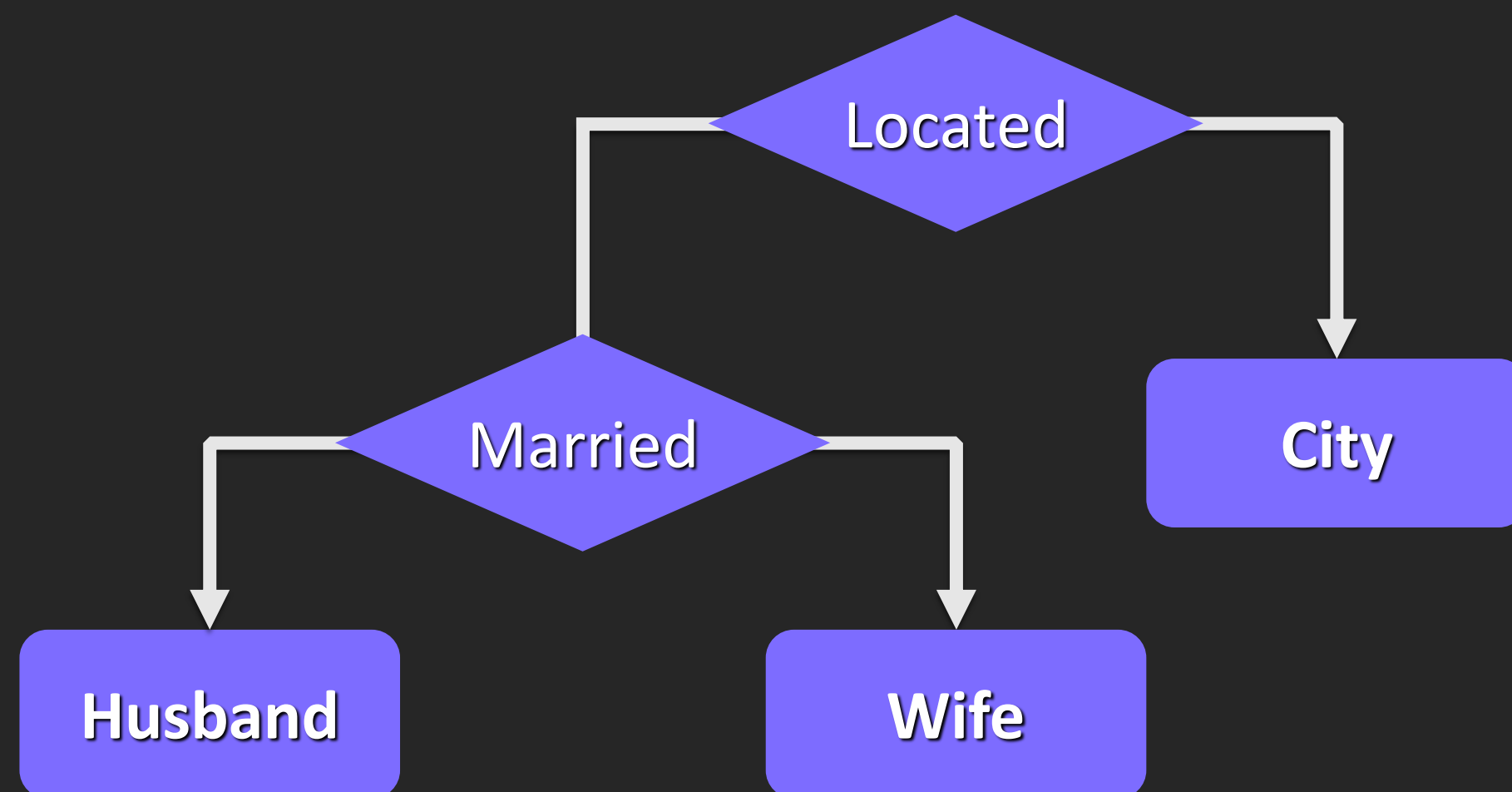
Equivalent Relations
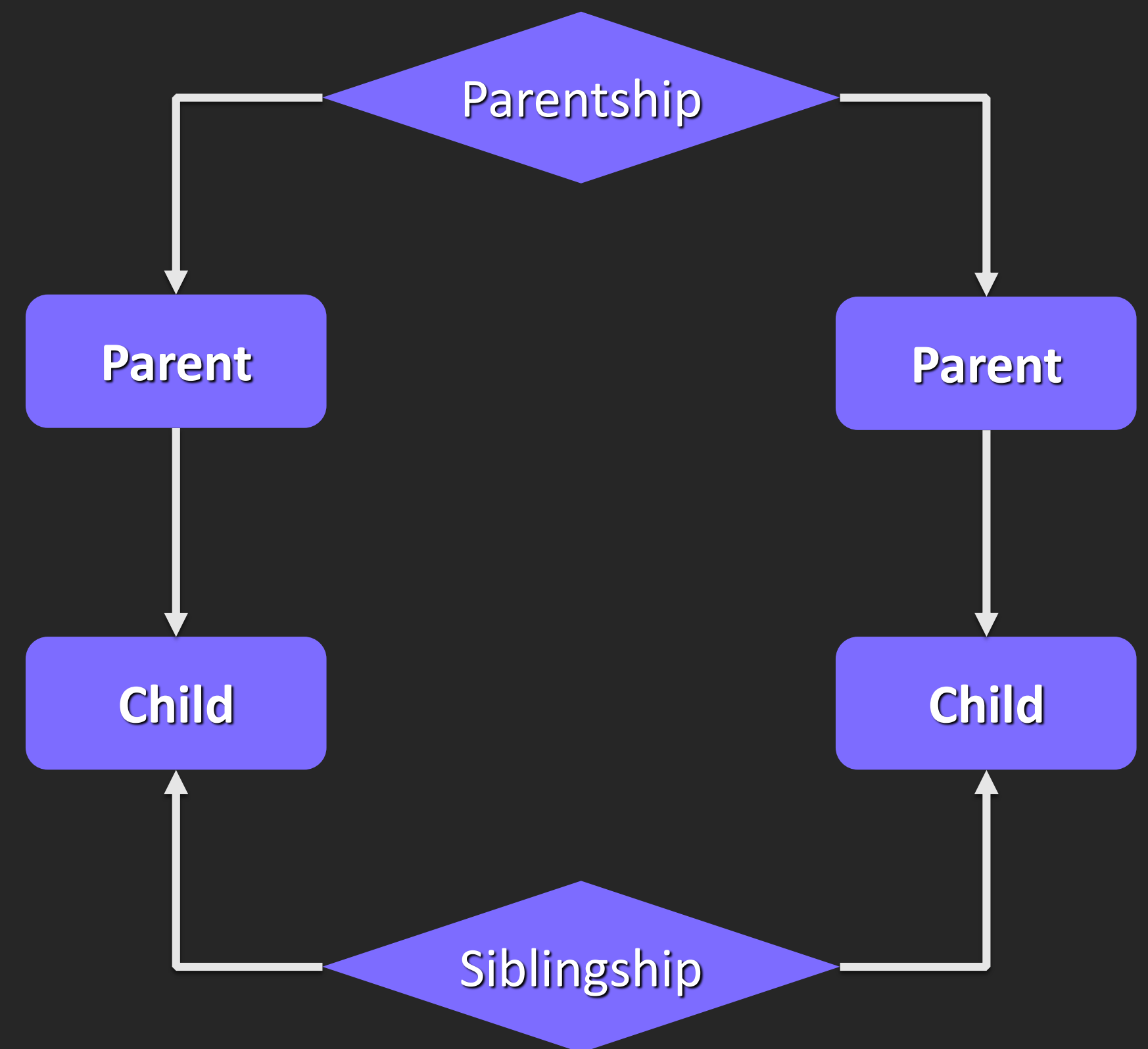
Grakn Modeling – Tips

Model incrementally
- Start with questions you want answered now

Be as specific as possible
- father > parent

Relations are first-class citizens
- No foreign keys!

Generally: Tables are entities, rows are instances, columns are attributes
- Associative/junction tables are relations

Unconnected data is hard to reach
- Relations provide easy access to connected data

No such thing as nulls
- The absence of value indicates the lack of value

Prefer ingesting non-changing data (facts) over syncing volatile data
- Represent new data (new facts) with additional values

Introducing Graql

Commands

CREATE/ALTER TABLE   ->   define

SELECT+WHERE   ->   match+get

INSERT INTO   ->   insert

DELETE   ->   delete

DROP TABLE   ->   undefine

UPDATE   ->   match+delete+insert

Data Types

NUMERIC   ->   long

DECIMAL   ->   double

TEXT   ->   string

BOOLEN   ->   boolean

DATE   ->   date

Defining Schemas

Products
ProductId
ProductName
CategoryId
UnitPrice

➡️

```sql
CREATE TABLE products (
    product_id smallint NOT NULL PRIMARY KEY,
    product_name character varying(40) NOT NULL,
    category_id smallint,
    unit_price float,
    FOREIGN KEY (category_id) REFERENCES categories
);
```

Defining Schemas

```
CREATE TABLE products (
    product_id smallint NOT NULL PRIMARY KEY,
    product_name character varying(40) NOT NULL,
    category_id smallint,
    unit_price float,
    FOREIGN KEY (category_id) REFERENCES categories
);
```

Defining Schemas

```sql
CREATE TABLE products (
    product_id smallint NOT NULL PRIMARY KEY,
    product_name character varying(40) NOT NULL,
    category_id smallint,
    unit_price float,
    FOREIGN KEY (category_id) REFERENCES categories
);
```

Defining Schemas

```
CREATE TABLE products (
    product_id smallint NOT NULL PRIMARY KEY,
    product_name character varying(40) NOT NULL,
    category_id smallint,
    unit_price float,
    FOREIGN KEY (category_id) REFERENCES categories
);
```

SQL	Graql

`CREATE TABLE products (`
`    product_id smallint NOT NULL PRIMARY KEY,`
`    product_name character varying(40) NOT NULL,`
`    category_id smallint,`
`    unit_price float,`
`    FOREIGN KEY (category_id) REFERENCES categories`
`);`
 	
`CREATE TABLE categories (`
`    category_id smallint NOT NULL PRIMARY KEY,`
`    category_name character varying(40)`
`);`
	`define`
	`product sub entity,`
	`    key product_name,`
	`    has unit_price,`
	`    plays product-assignment;`
	`category sub entity,`
	`    has category_name,`
	`    plays assigned-category;`
	
	`product_name sub attribute, datatype string;`
	`unit_price sub attribute, datatype double;`
	`category_name sub attribute, datatype string;`
	
	`category-assignment sub relation,`
	`    relates assigned-category,`
	`    relates product-assignment;`

Defining Schemas

SQL

```
CREATE TABLE albums (
    album_id SMALLINT NOT NULL PRIMARY KEY,
    album_name CHARACTER VARYING(40) NOT NULL,
    release_date DATE NOT NULL,
    artist_id SMALLINT NOT NULL,
    FOREIGN KEY (artist_id) REFERENCES artists
);


CREATE TABLE artists (
    artist_id smallint NOT NULL PRIMARY KEY,
    artist_name character varying(40)
);
```

Graql

```
_____
album sub _____,
    key album_name,
    has release_date,
    _____ released_album;
artist sub entity,
    ___ artist_name,
    plays releasing_artist;


album-release sub relation,
    relates released_album,
    relates releasing_artist;


album_name sub _____, datatype string;
release_date sub attribute, datatype date;
artist_name sub attribute, datatype _____;
```

Defining Schemas

SQL

category_id	category_name

product_id	product_name
100	Candy

```
INSERT INTO categories (category_id, category_name)
VALUES (100, 'Candy');
```

Graql
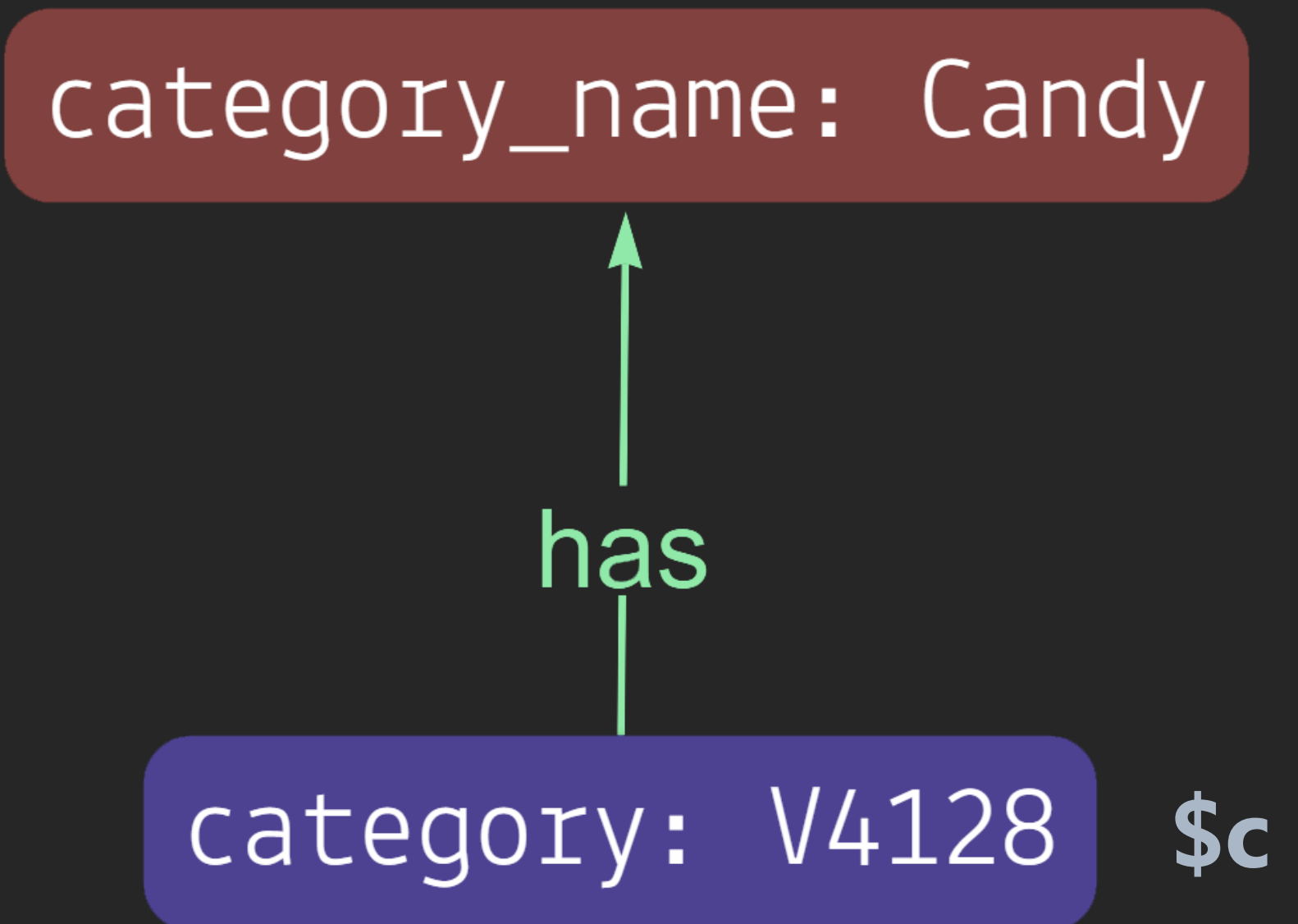

```
insert $c isa category,
    has category_name "Candy";
```

SQL

Graql

INSERT INTO artists (artist_id, artist_name)
VALUES (1, 'Michael Jackson');

_____ $a isa _____,
    has _____ "Michael Jackson";

Writing Data – Insert

SQL

category_id	category_name
100	Candy

product_id	product_name	category_id	unit_price
99	Skittles	**100**	1.10

```sql
INSERT INTO products
    (product_id, product_name, category_id, unit_price)
SELECT 99, 'Skittles', category_id, 1.10
FROM categories
WHERE category_name = 'Candy';
```

Graql

category_name: Candy

product_name: Skittles   unit_price: 1.1

has

category: V4128 **$c**

has   has

product: V4240 **$p**

assigned-category   product-assignment
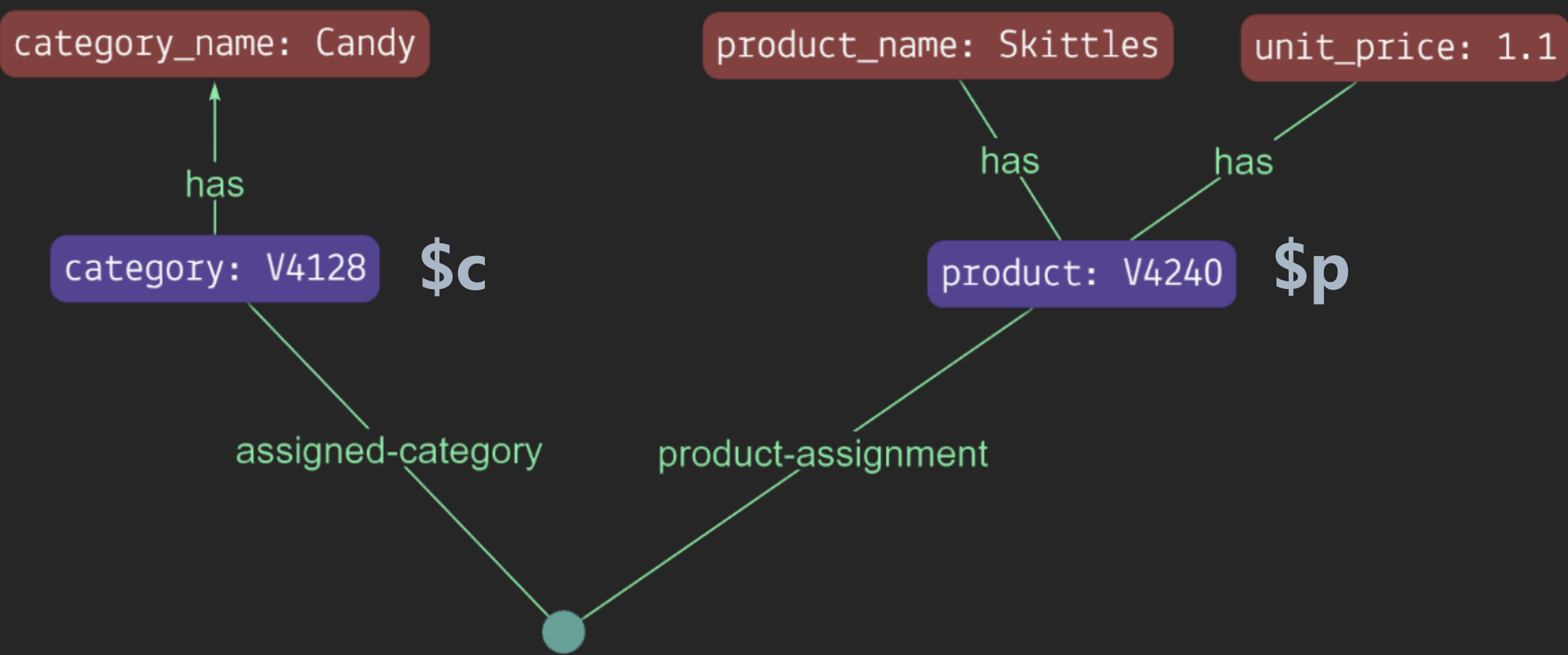
```
match
$c isa category, has category_name "Candy";
insert
$p isa product, has product_name "Skittles", has unit_price 1.10;
(product-assignment: $p, assigned-category: $c)
    isa category-assignment;
```

Writing Data – Associate

SQL

Graql

```
INSERT INTO albums
    (album_id, album_name, release_date, artist_id)
SELECT 1, 'Bad', to_date('1987-08-31','YYYY-MM-DD'), artist_id
FROM artists
WHERE artist_name = 'Michael Jackson';
```

```
_____
____ isa artist, ____ artist_name "Michael Jackson";
insert
____ isa album,
    has album_name "Bad",
    has release_date 1987-08-31;
(released_album: $al, releasing_artist: $ar) ____ album-release;
```

Writing Data – Associate

SQL

product_id	product_name	category_id	unit_price

```
SELECT product_name, unit_price
FROM products;
```
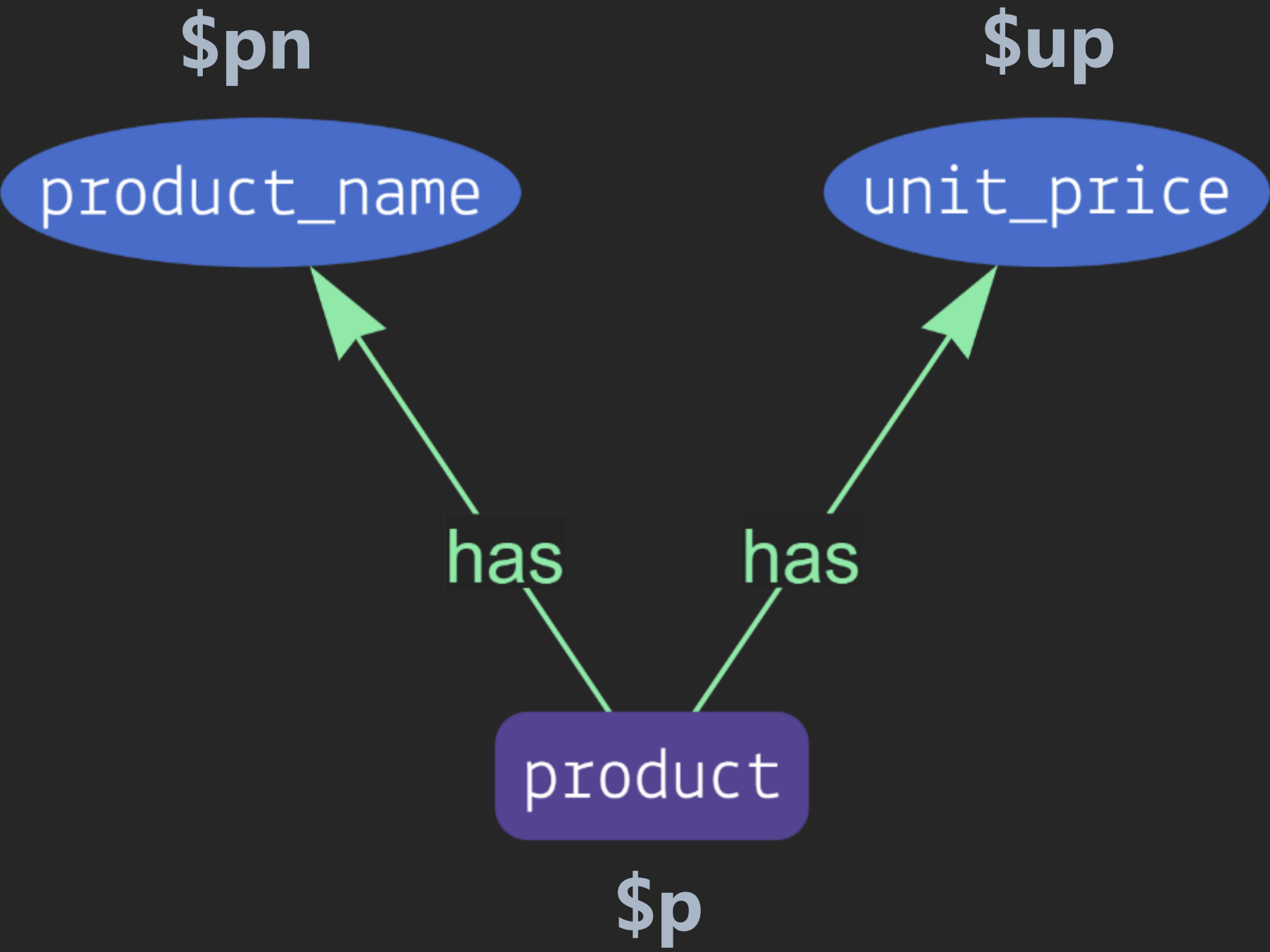
Graql


```
match
$p isa product,
    has product_name $pn,
    has unit_price $up;
get $pn, $up;
```

Reading Data – Project

SQL

Graql

SELECT album_name, release_date
FROM albums;

```
match
$a ___ album,
    has album_name $n,
    has release_date $r;
get __, __;
```

Reading Data – Project

SQL

product_id	product_name	category_id	unit_price
			>= 1.25
			>= 1.25
			>= 1.25
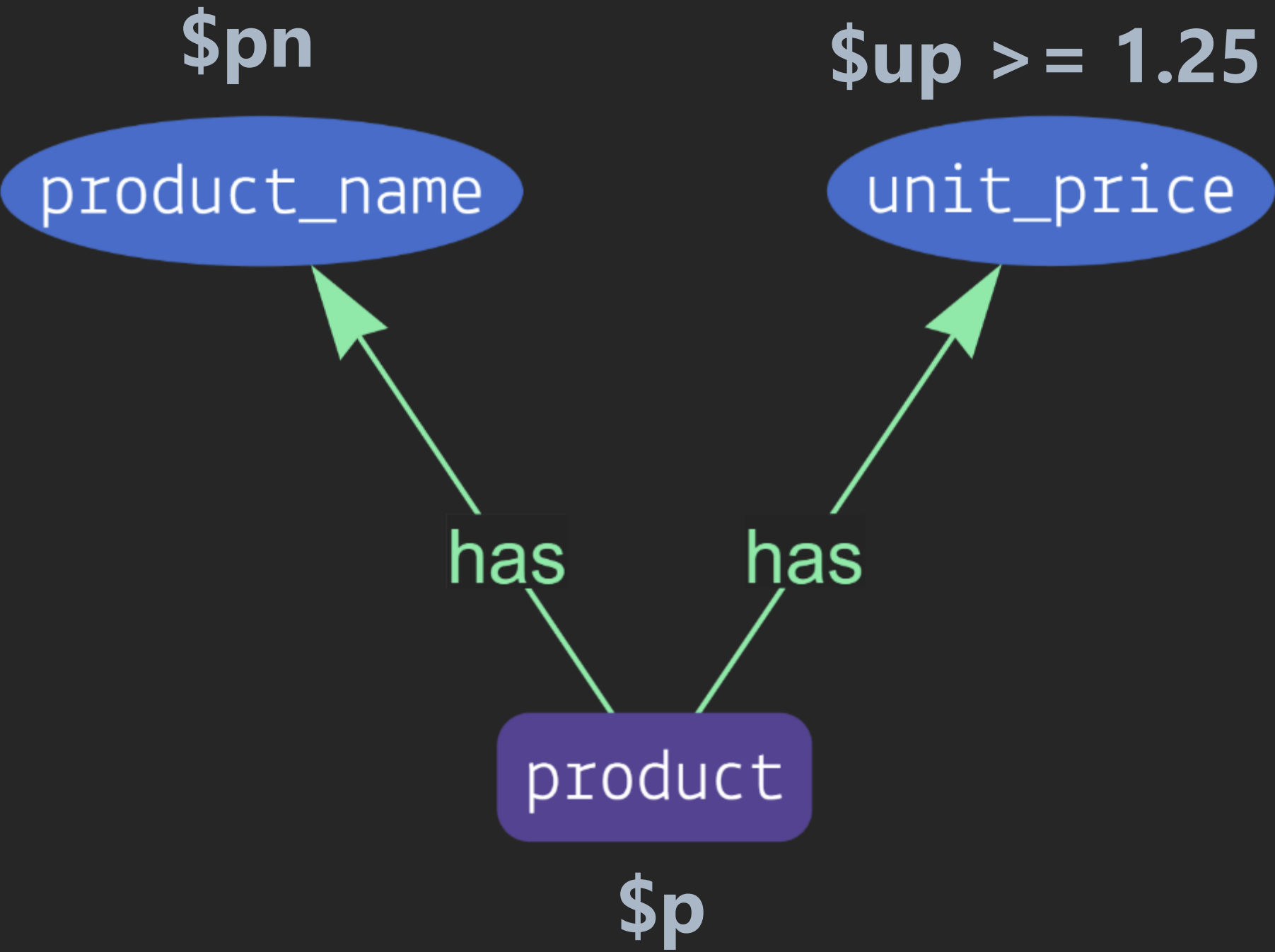
```sql
SELECT product_name, unit_price
FROM products
WHERE unit_price >= 1.25;
```

Graql

$pn

$up >= 1.25

product_name   unit_price

      has   has

         product

          $p

```
match
$p isa product,
    has product_name $pn,
    has unit_price $up;
$up >= 1.25;
get $pn, $up;
```

Reading Data – Restrict

SQL

Graql

```
match
$a isa album,
    has album_name $n,
    has release_date $r;
$r < 1990-01-01;
___ __, __;
```

```
SELECT album_name, release_date
FROM albums
WHERE release_date < to_date('1990-01-01','YYYY-MM-DD');
```

Reading Data – Restrict

SQL

Graql

CategoryName	...

ProductName	...

CategoryName	ProductName

$cn

category_name

has

$c category

assigned-category

$pn

product_name

has

product $p

product-assignment

category-assignment
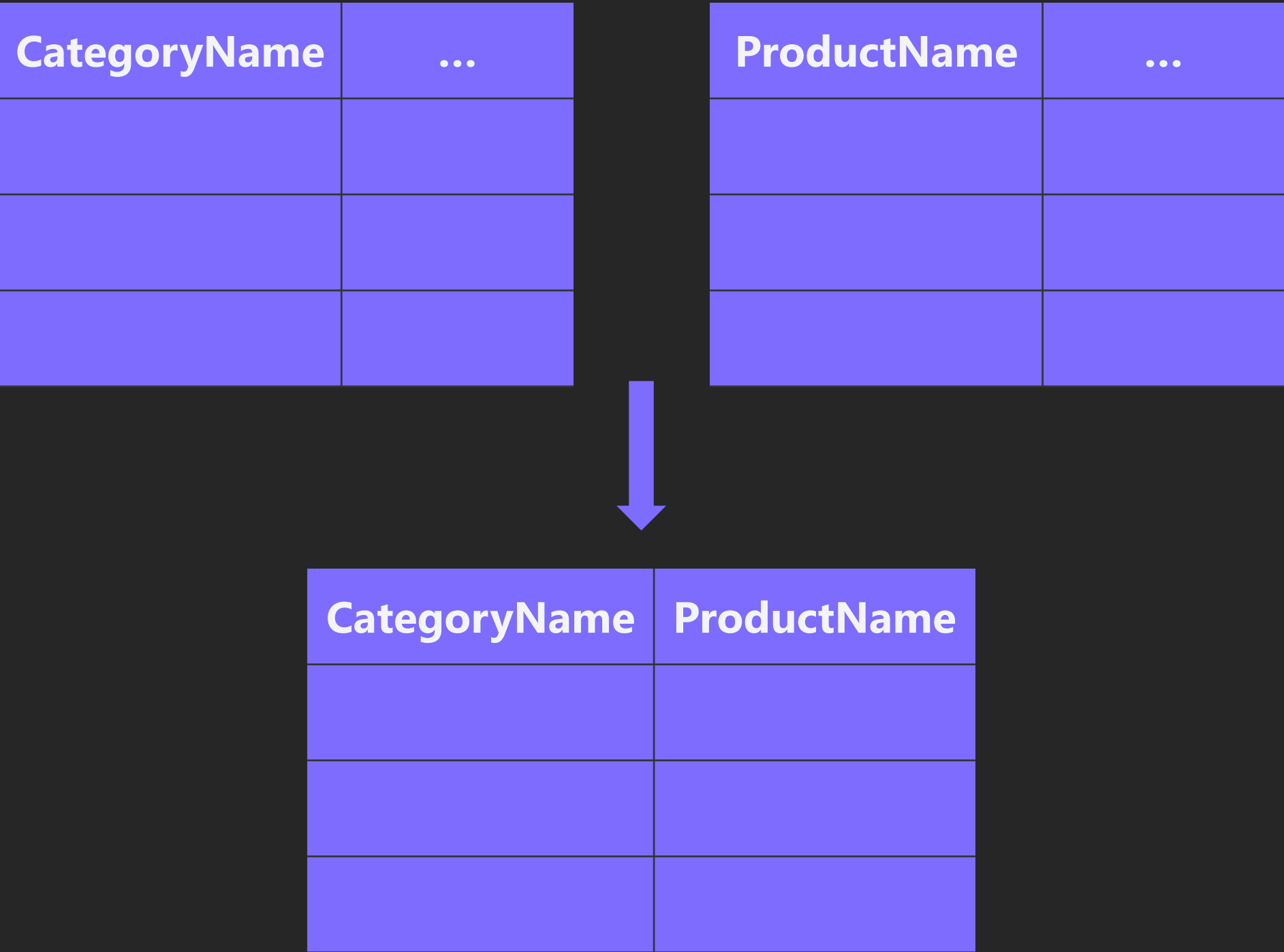
SELECT category_name, product_name
FROM categories
INNER JOIN products
    ON categories.category_id = products.category_id;
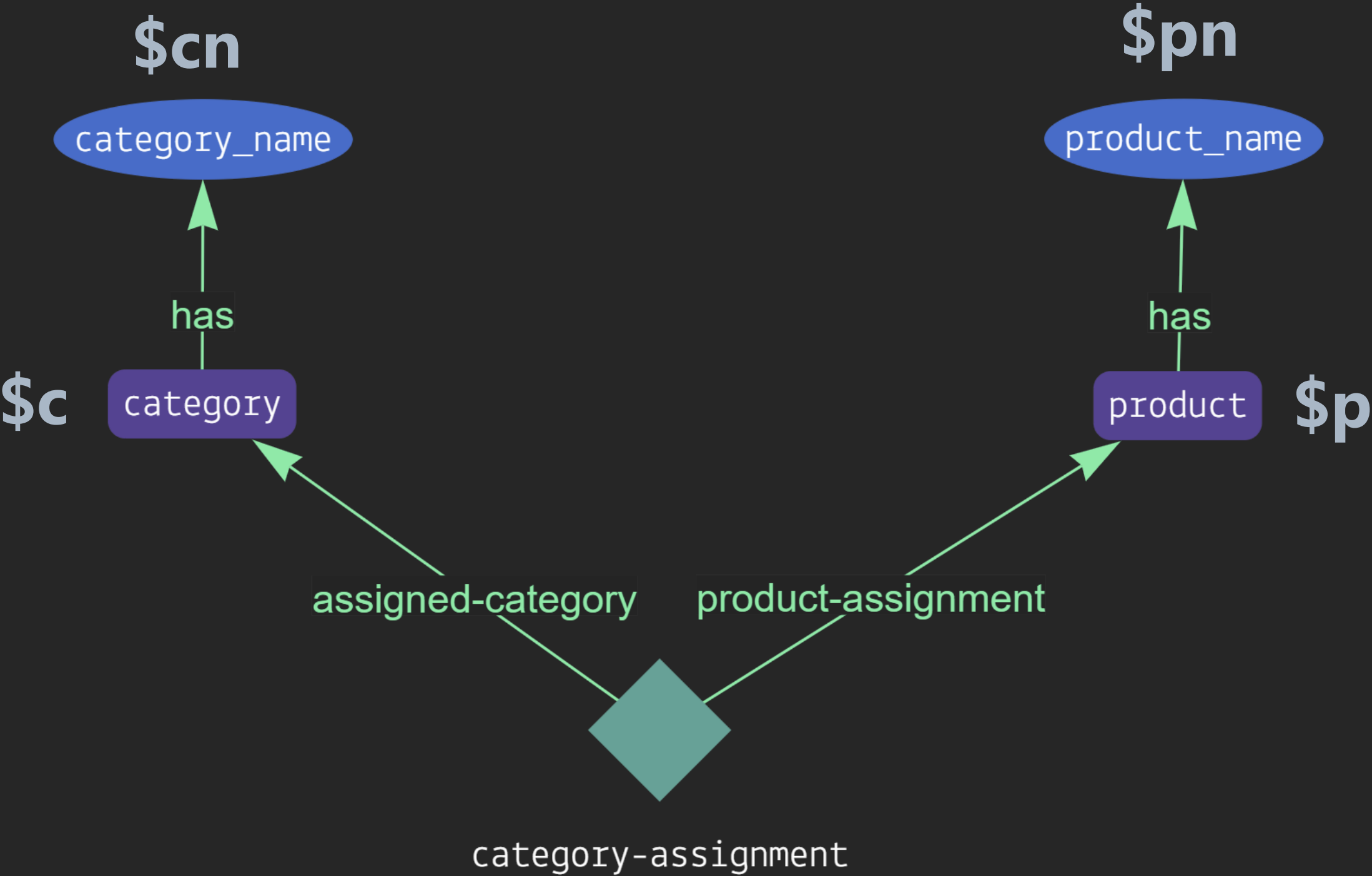
match
$c isa category, has category_name $cn;
$p isa product, has product_name $pn;
($c, $p) isa category-assignment;
get $cn, $pn;

Reading Data – Inner Join

SQL

Graql

SELECT artist_name, album_name
FROM artists
INNER JOIN albums
    ON artists.artist_id = albums.artist_id;

```
match
$ar isa artist,
    has artist_name $ar_name;
$al isa album,
    has album_name $al_name;
(____, ____) isa album-release;
get $ar_name, $al_name;
```

Reading Data – Inner Join

SQL

Suppliers

			city_name	

Customers

		city_name			

```
SELECT city_name
FROM customers
INTERSECT
SELECT city_name
FROM suppliers;
```
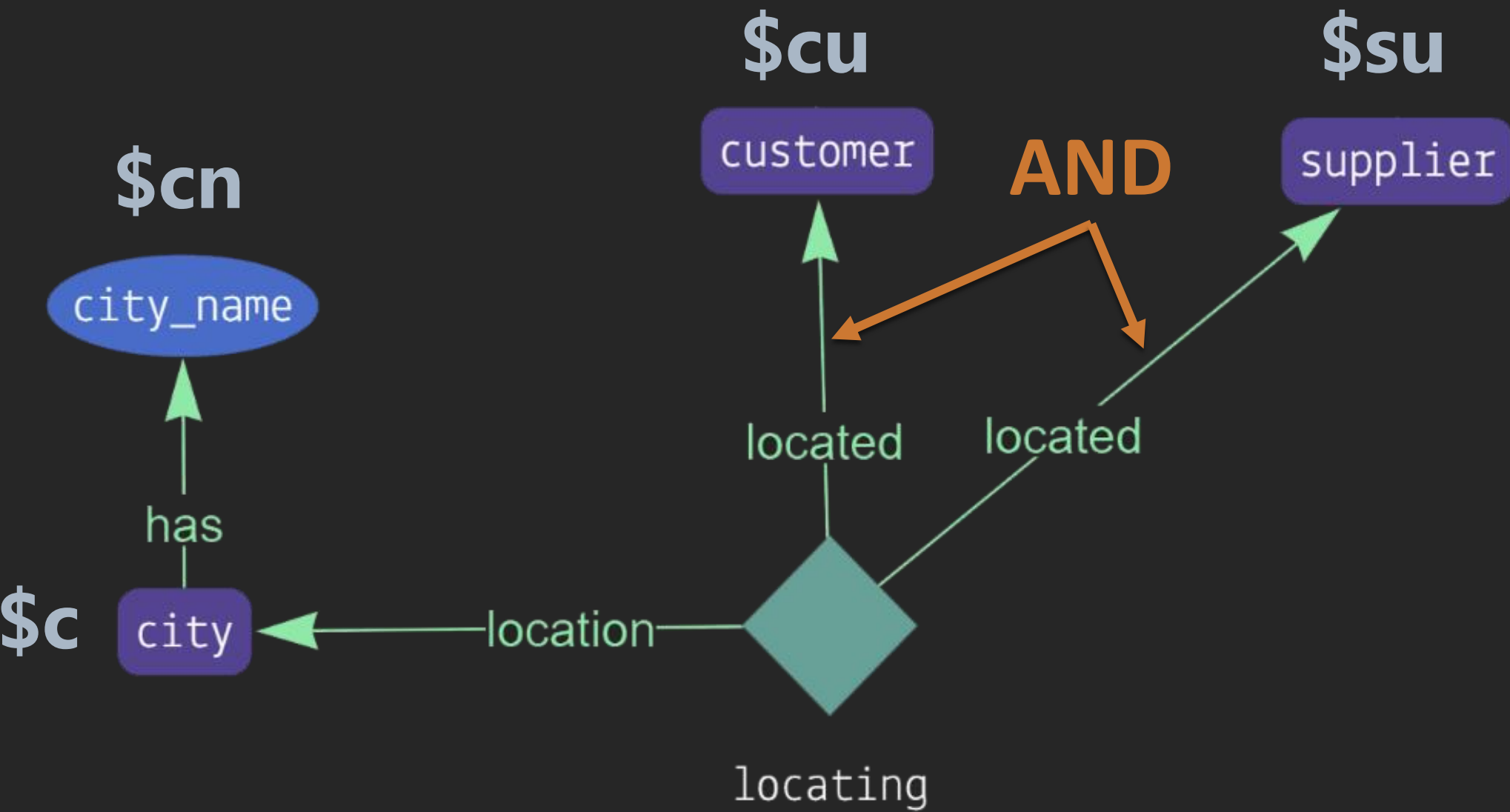
Graql


```
match
$c isa city, has city_name $cn;
$cu isa customer; (location: $c, $cu);
$su isa supplier; (location: $c, $su);
get $cn;
```

Reading Data – Intersect

SQL

Graql

```
SELECT album_name
FROM albums
INNER JOIN artists on albums.artist_id = artists.artist_id
WHERE artist_name = 'Michael Jackson'
INTERSECT
SELECT album_name
FROM albums
WHERE release_date >= to_date('2000-01-01','YYYY-MM-DD');
```

```
_____
$al ___ album, has album_name __;
$ar isa artist, ___ artist_name "Michael Jackson";
(released_album: ___, releasing_artist: $ar) isa album-release;
$al has release_date >= _____;
get $n;
```

Reading Data – Intersect

SQL

Suppliers

			city_name	

Customers

		city_name		

```
SELECT city_name
FROM customers
UNION
SELECT city_name
FROM suppliers;
```
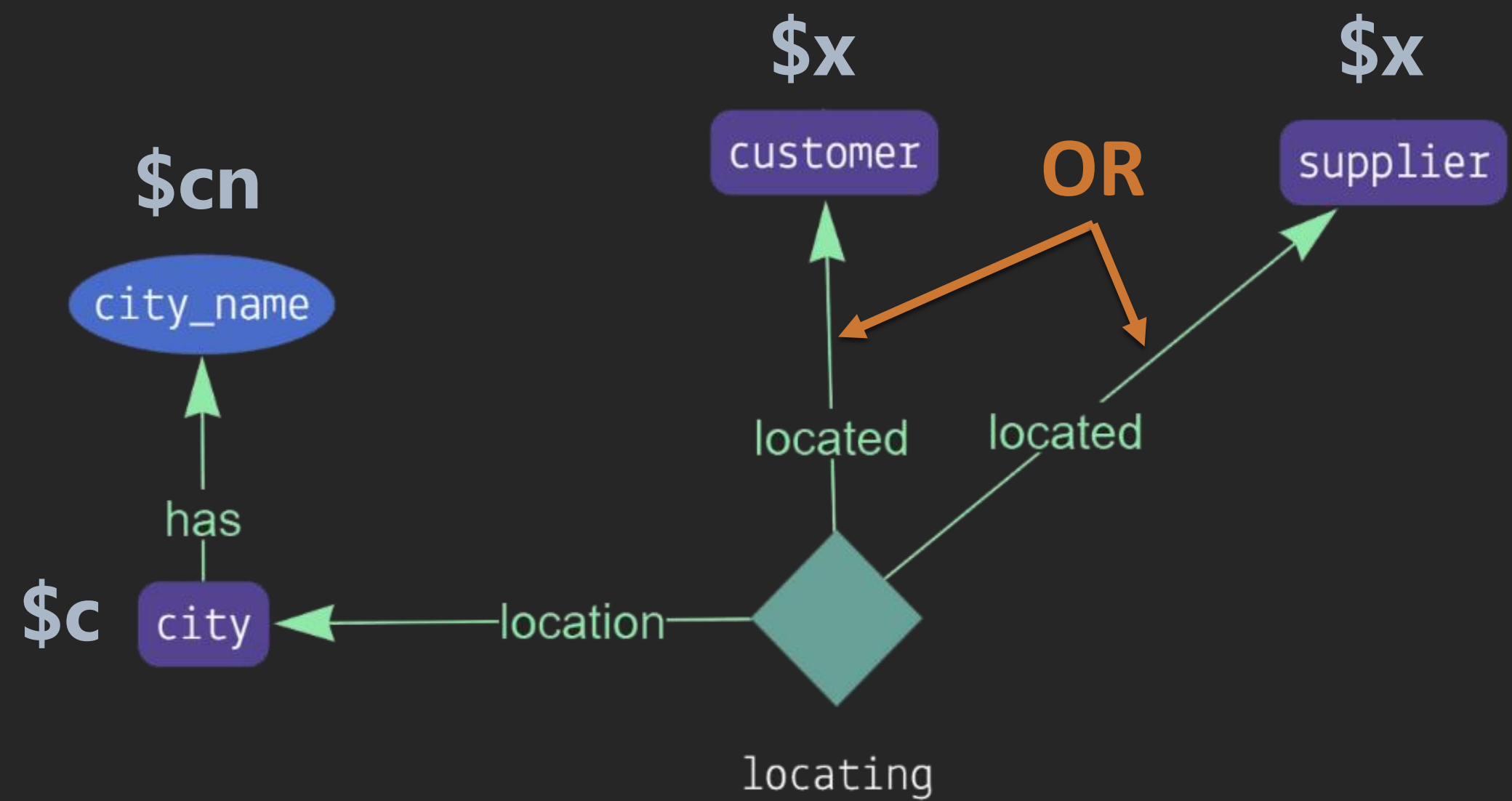
Graql


```
match
$c isa city, has city_name $cn;
{$x isa supplier;} or {$x isa customer;};
(location: $c, $x);
get $cn;
```

Reading Data – Union

SQL

```
SELECT album_name
FROM albums
INNER JOIN artists on albums.artist_id = artists.artist_id
WHERE artist_name = 'Michael Jackson'
UNION
SELECT album_name
FROM albums
WHERE release_date >= to_date('2000-01-01','YYYY-MM-DD');
```

Graql

```
match
____ ____ _____, ____ _____ $n;
{
    $ar isa artist, has artist_name "Michael Jackson";
    (released_album: $al, releasing_artist: $ar) isa album-release;
} __ {
    $al has release_date >= 2000-01-01;
};
get $n;
```
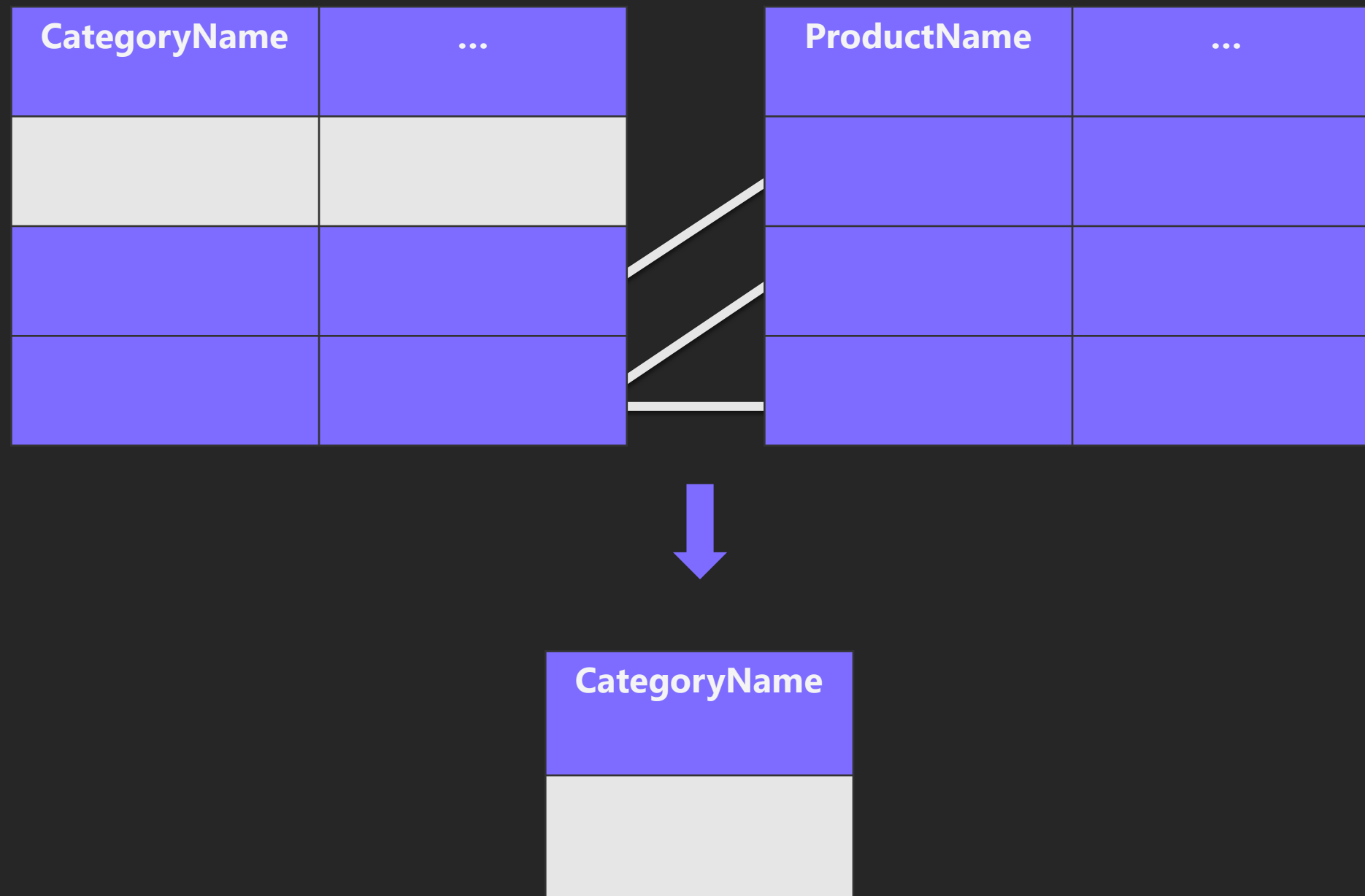
Reading Data – Union

SQL

CategoryName	...

ProductName	...

CategoryName

```
SELECT category_name
FROM categories
LEFT JOIN products
    ON categories.category_id = products.category_id
WHERE products.category_id IS NULL;
```

Graql

$cn

category_name

has

$c category

assigned-category

category-assignment

product-assignment

product_name

has

product **$x**

Missing relation

```
match
$c isa category,
    has category_name $cn;
not { ($c, $x) isa category-assignment; };
get $cn;
```

Reading Data – Left Join

SQL

Graql

SELECT artist_name
FROM artists
LEFT JOIN albums
   ON artists.artist_id = albums.artist_id
WHERE albums.artist_id IS NULL;

```
_____
$ar isa artist,
   has artist_name $ar_name;
___ {
   (___, $x) ___ album-release;
};
___ $ar_name;
```

Reading Data – Left Join

SQL

category_name	...

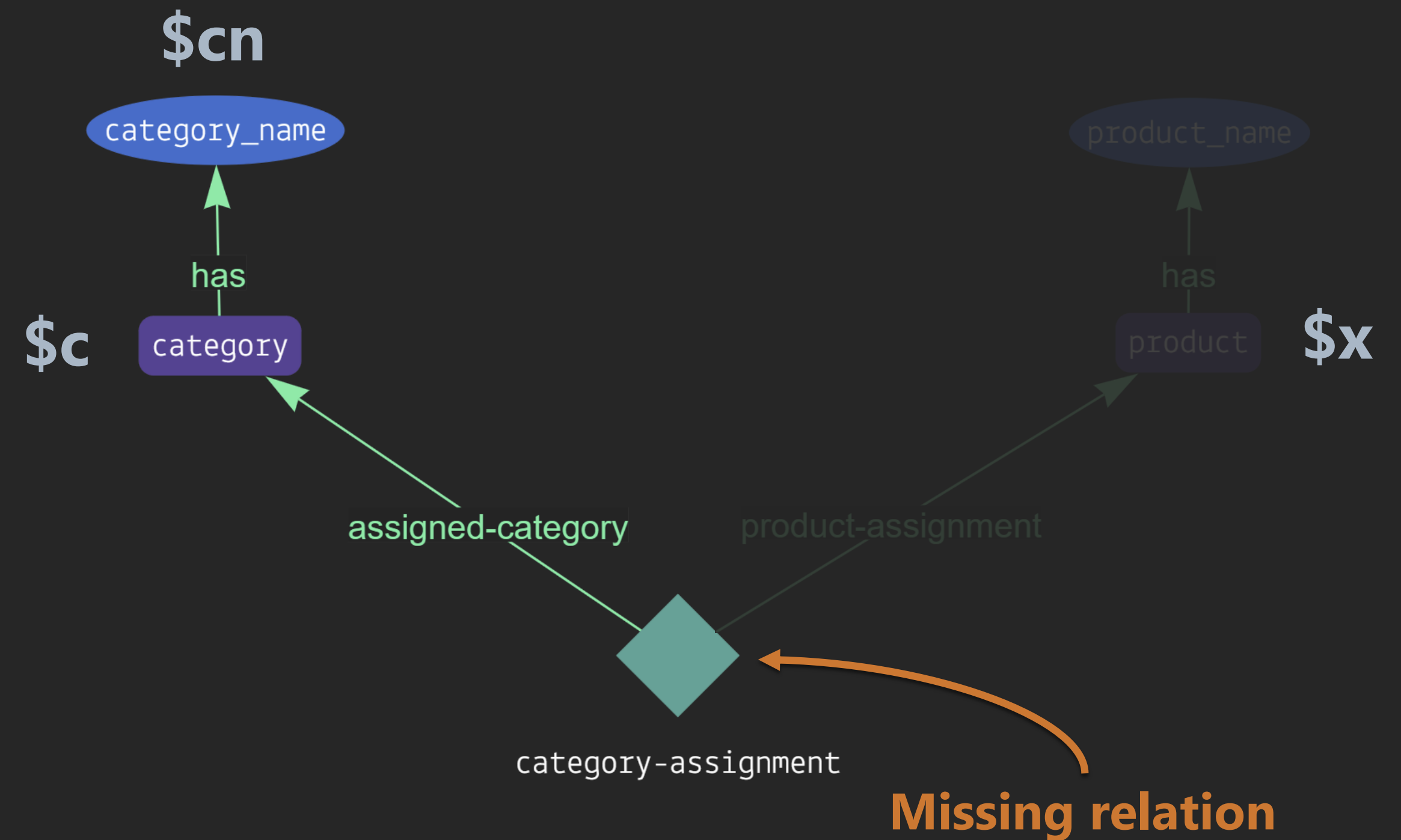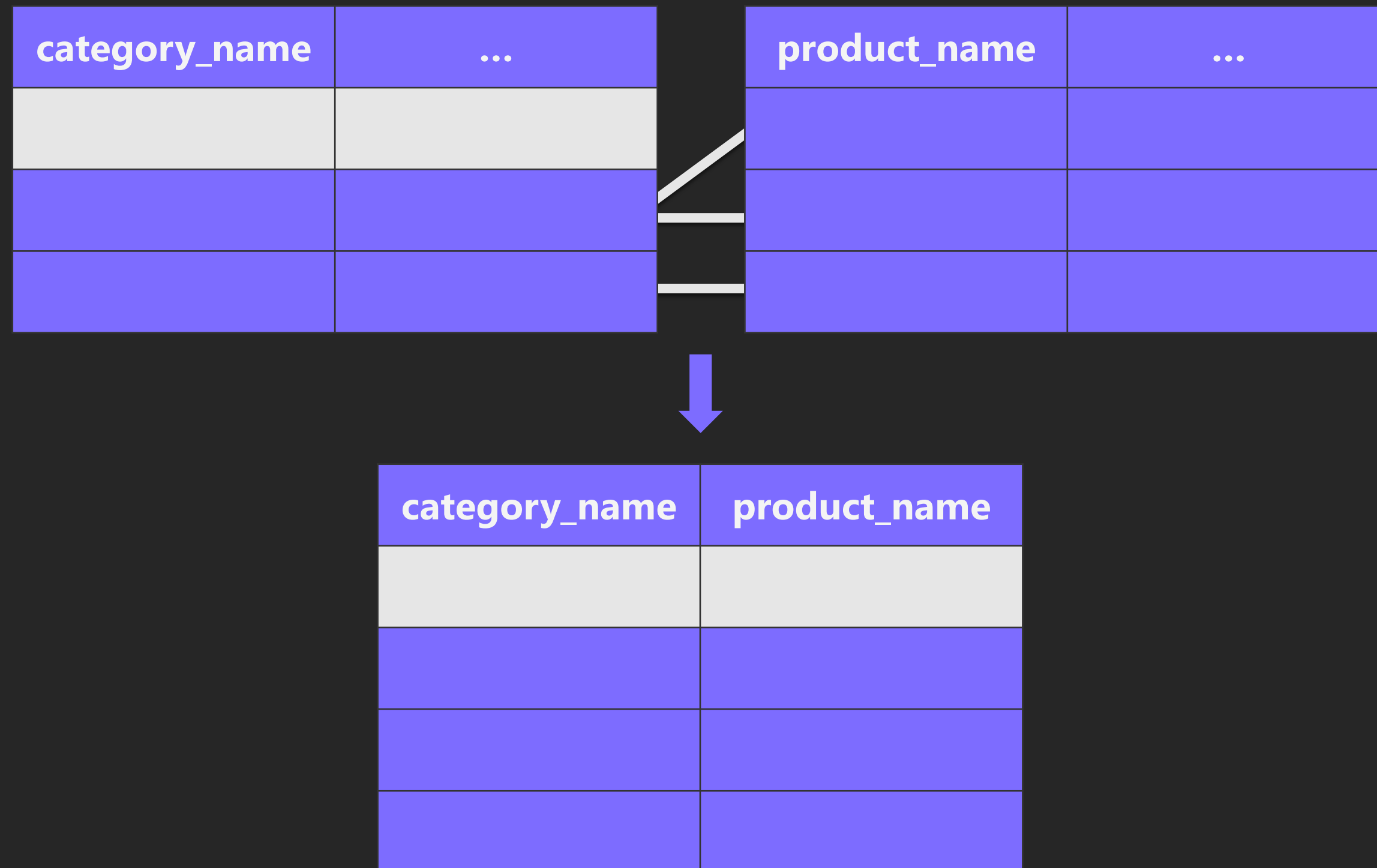| product_name	...

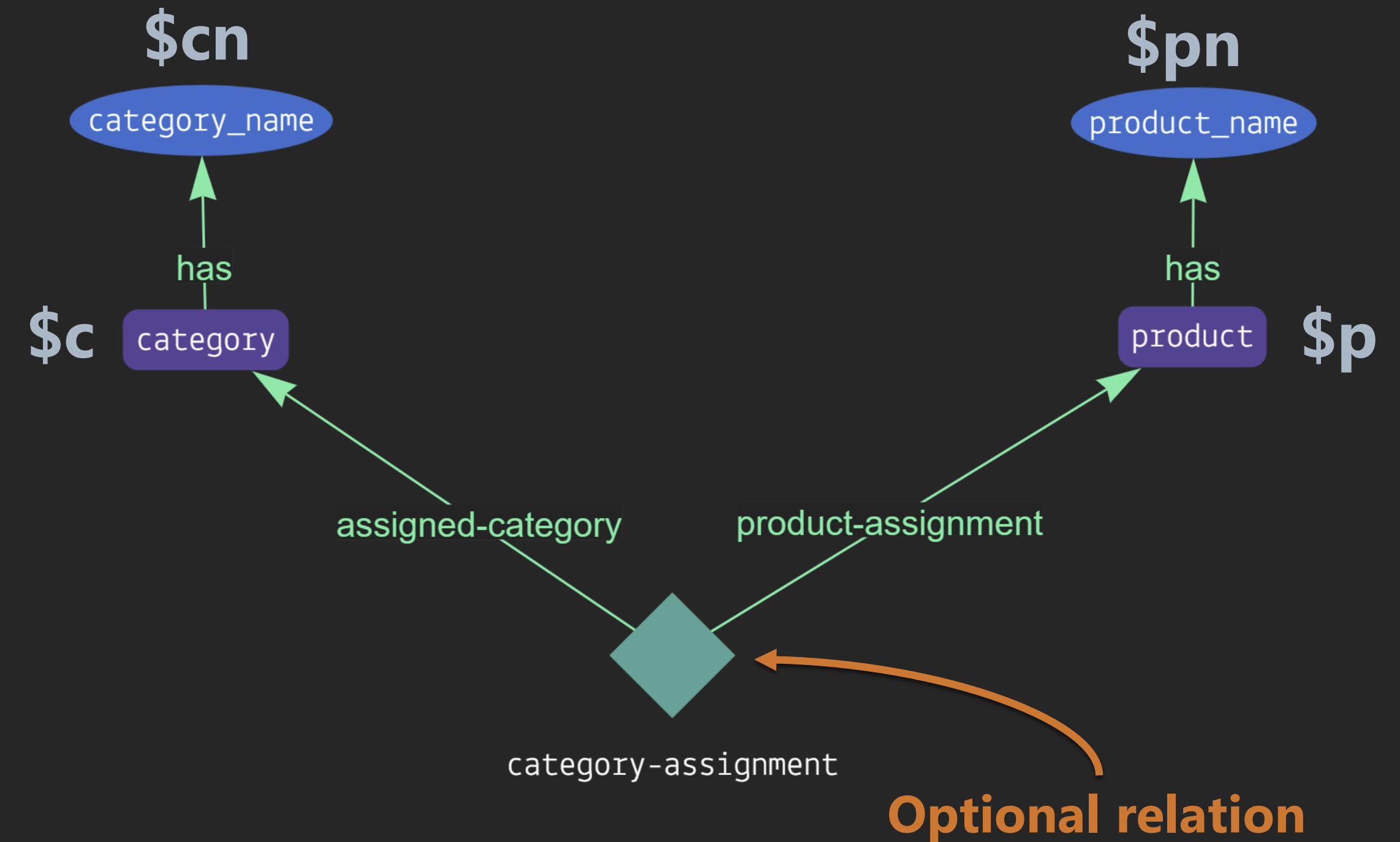| category_name	product_name

SELECT category_name
FROM categories
LEFT JOIN products
   ON categories.category_id = products.category_id
WHERE products.category_id IS NULL;

Graql

$cn
category_name

$pn
product_name

has

has

$c category

product $p

assigned-category

product-assignment

category-assignment

Optional relation

match
$c isa category, has category_name $cn;
$p isa product, has product_name $pn;
{ ($c, $p) isa category-assignment; }
or { not { ($c, $p) isa category-assignment; }; };
get $cn, $pn;

Reading Data – Full Outer Join

SQL

Graql

```
match
$ar isa artist, has artist_name $ar_name;
$al ___ album, has album_name $al_name;
{
    ($ar, $al) isa album-release;
} __ {
    ___ {
        ($ar, $al) isa _____;
    };
};
get $ar_name, _____;
```

```
SELECT artist_name, album_name
FROM artists
FULL OUTER JOIN albums
    ON artists.artist_id = albums.artist_id;
```

Reading Data – Full Outer Join

Reading Data – Inference

- All men are mortal
- Socrates is a man
- Therefore, Socrates is mortal
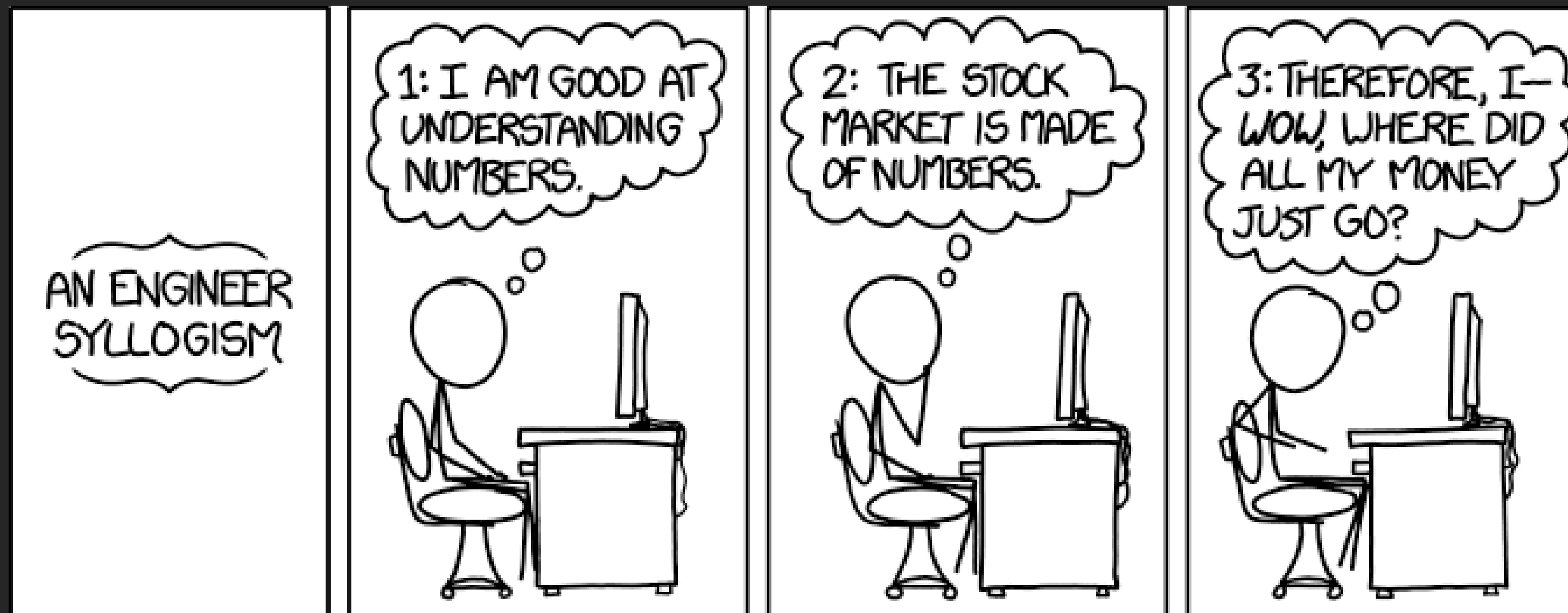
```
men-are-mortal sub rule,
when {
    $man isa man;
},
then {
    $man isa mortal;
};

insert $m isa man, has name "Socrates";
match $mortal isa mortal; get;
```

Reading Data – Inference

https://xkcd.com/1570/

- All women are mortal
- Socrates is a man
- Therefore, Socrates is mortal

Reading Data – Inference

- <u>When</u>
  - Texas *($b)* is located in USA *($a)*
  - Dallas *($c)* is located in Texas *($b)*

- <u>Then</u>
  - Dallas *($c)* is located in USA *($a)*

```
transitive-location sub rule,
when {
    (location: $a, located: $b) isa locating;
    (location: $b, located: $c) isa locating;
}, then {
    (location: $a, located: $c) isa locating;
};
```

SQL

product_id	product_name
14	**Pepsi**
15	Coca-Cola
16	Fanta

product_id	product_name
15	Coca-Cola
16	Fanta

Graql

product_name: Pepsi

has

product: V16576

$p

```
DELETE FROM products
WHERE product_name = 'Pepsi';
```

```
match
$p isa product, has product_name "Pepsi";
delete $p;
```

Deleting Data

SQL

Graql

DELETE
FROM albums
USING artists
WHERE albums.artist_id = artists.artist_id
AND artists.artist_name = 'Michael Jackson'
AND albums.album_name = 'Bad';

———
$ar isa artist, has artist_name "Michael Jackson";
$al isa album, has album_name "Bad";
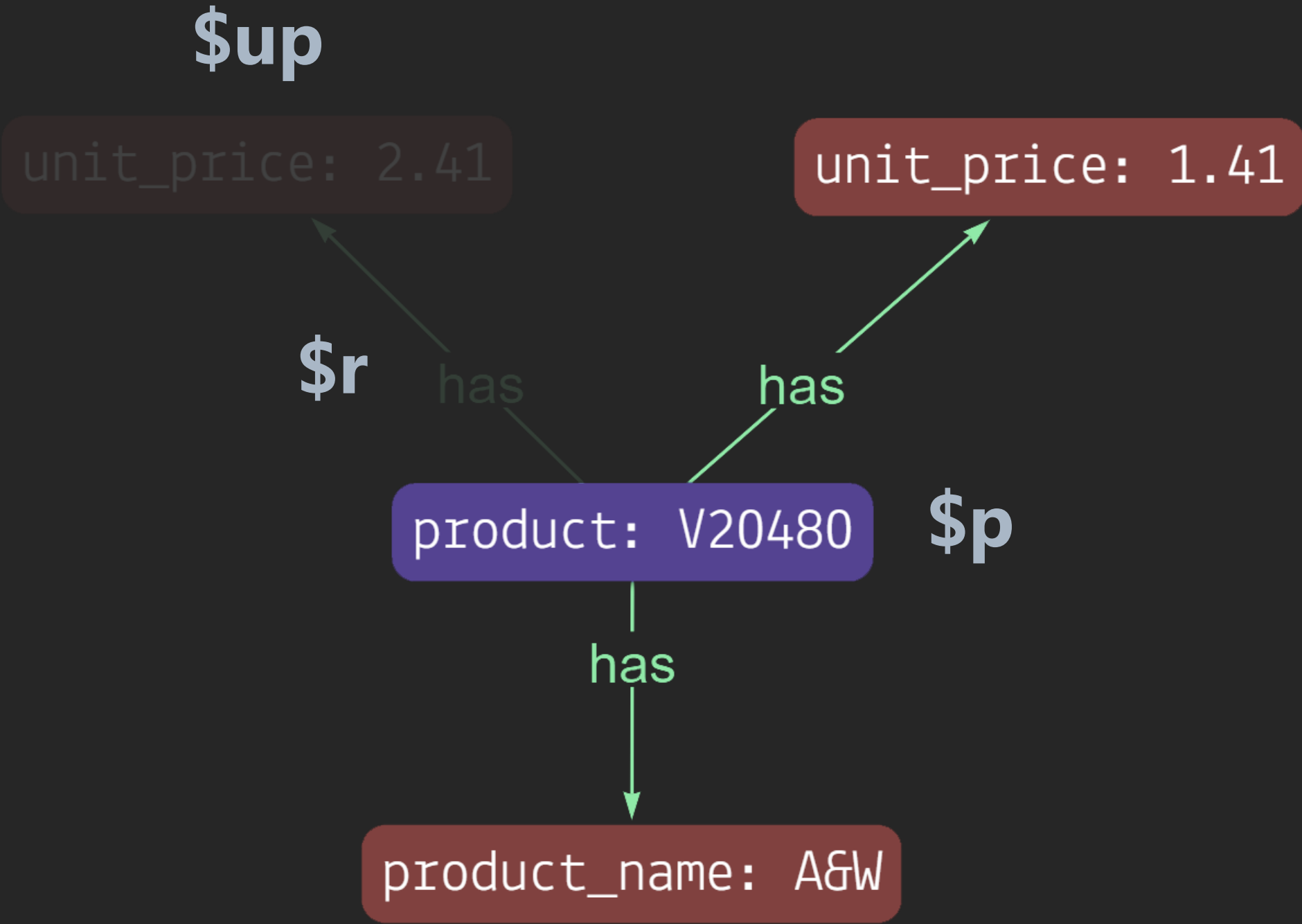($ar, $al) isa album-release;
——— ——;

Deleting Data

SQL

product_id	product_name	unit_price
17	Sprite	$1.55
18	7 UP	$1.68
19	A&W	**$2.41**

product_id	product_name	unit_price
17	Sprite	$1.55
18	7 UP	$1.68
19	A&W	**$1.41**

```
UPDATE products
SET unit_price = 1.41
WHERE product_name = 'A&W';
```

Graql

$up

unit_price: 2.41

unit_price: 1.41

$r   has   has

product: V20480   **$p**

has

product_name: A&W

```
match $p isa product,
    has product_name "A&W", has unit_price $up via $r;
delete $r;

match $p isa product, has product_name "A&W";
insert $p has unit_price 1.41;
```

Modifying Data

SQL

Graql

```
UPDATE artists
SET artist_name = 'Michael Joseph Jackson'
WHERE artist_name = 'Michael Jackson';
```

```
_____ $ar isa artist, has artist_name "Michael Jackson";
_____ $ar has artist_name "UMichael Joseph Jackson";

_____ $ar isa artist, has artist_name "Michael Jackson" via $r;
_____ $r;
```
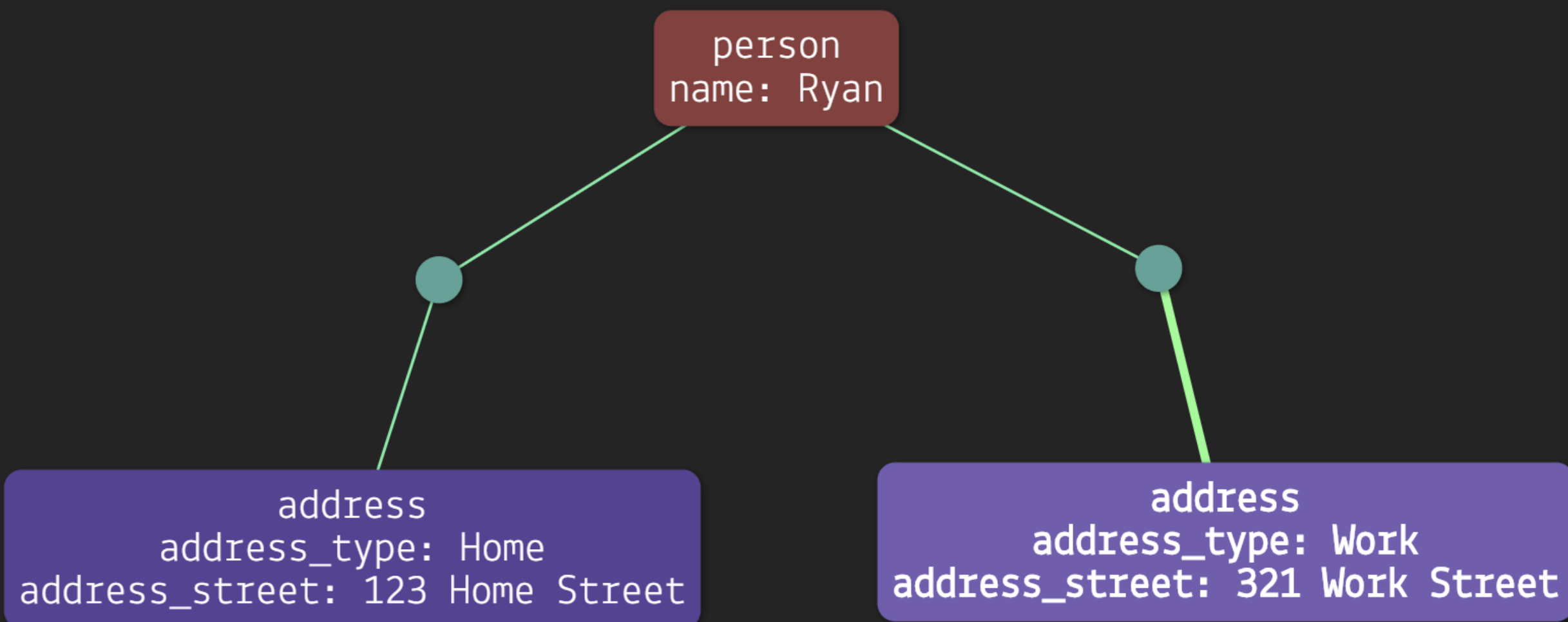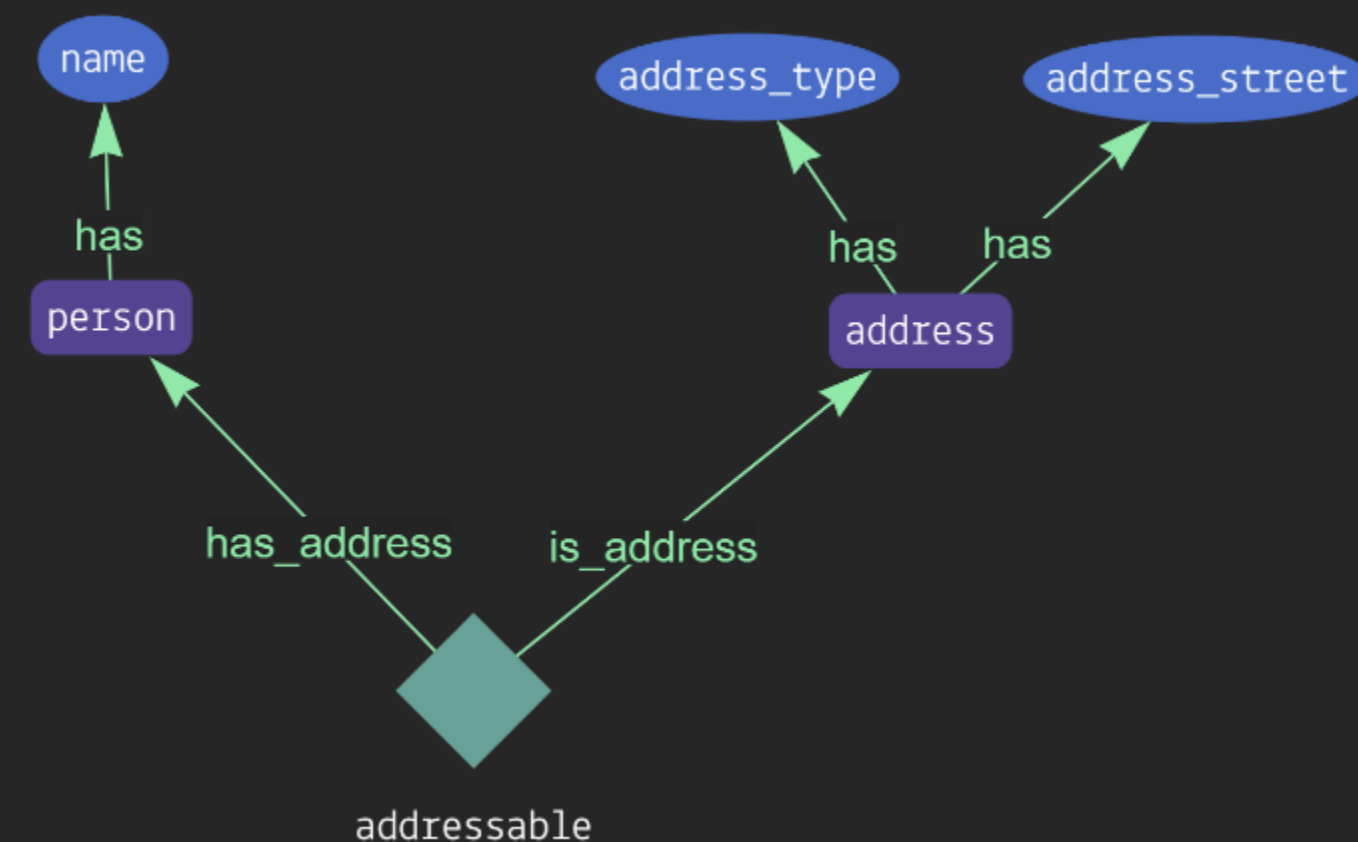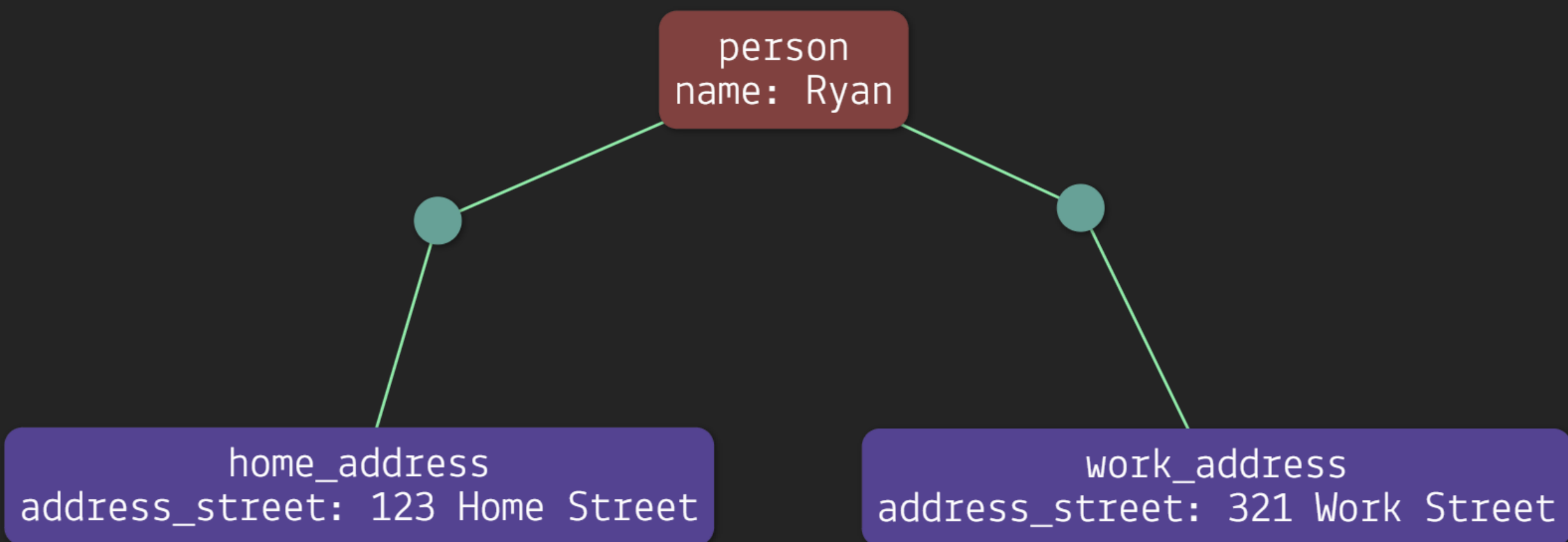
Modifying Data

Exercises

Exercise – Which is Better?

Option A

Option B
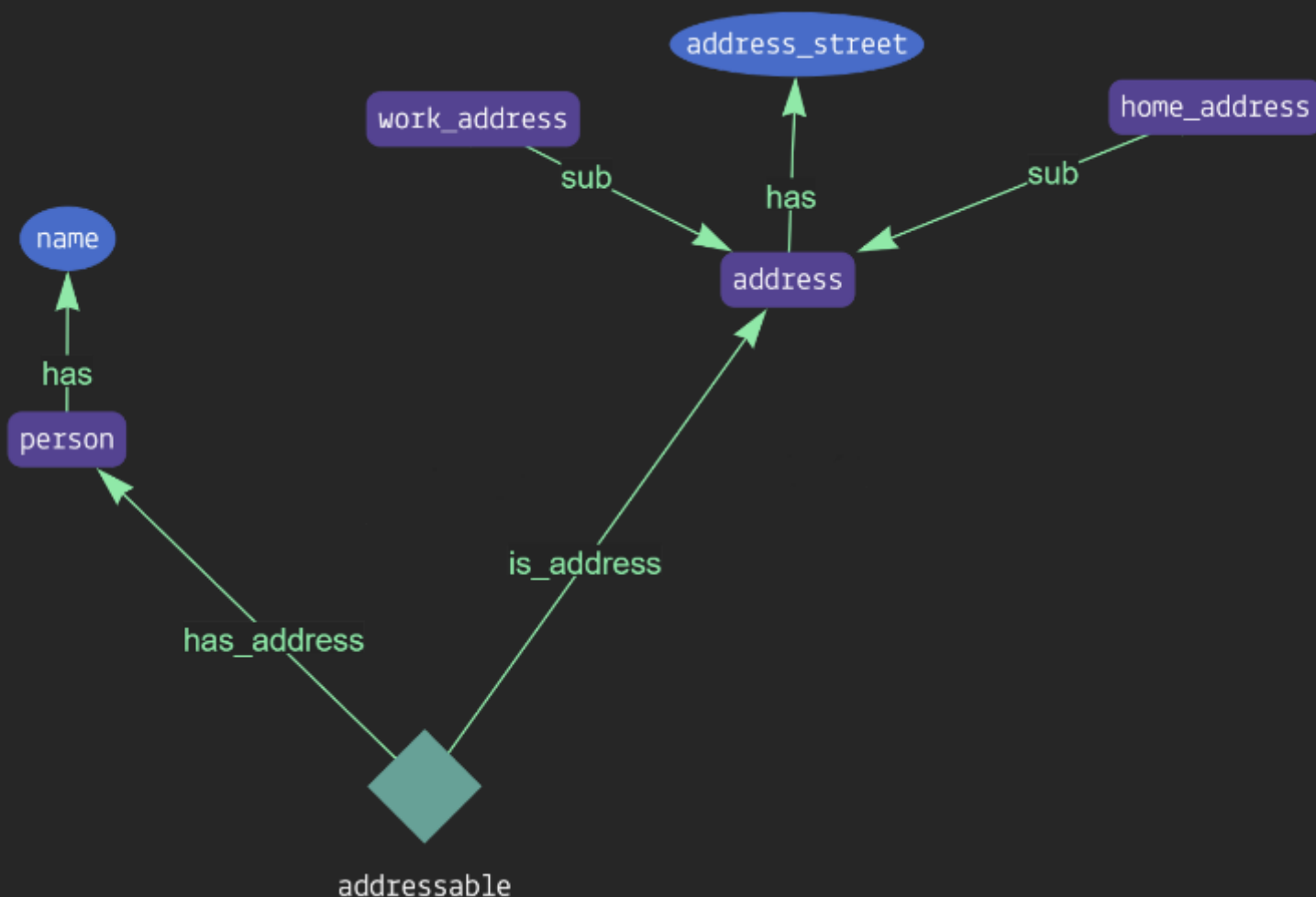
Exercise – Will This Work?

```
############################## SCHEMA ##############################
define
product sub entity,
    key product_name;
city sub entity,
    has city_name,
    plays location,
    plays located;
locating sub relation,
    relates location,
    relates located;
product_name sub attribute, datatype string;
city_name sub attribute, datatype string;


############################## DATA ##############################
insert
$bacon isa product, has product_name "Bacon";
$toronto isa city, has city_name "Toronto";
(location: $bacon, located: $toronto) isa locating;
```

Exercise – Convert Tables

Objectives

1. Convert the following tables to an equivalent Grakn schema
2. Populate the knowledge base with the equivalent data
3. Write a query to get the primary language of Argentina

Countries

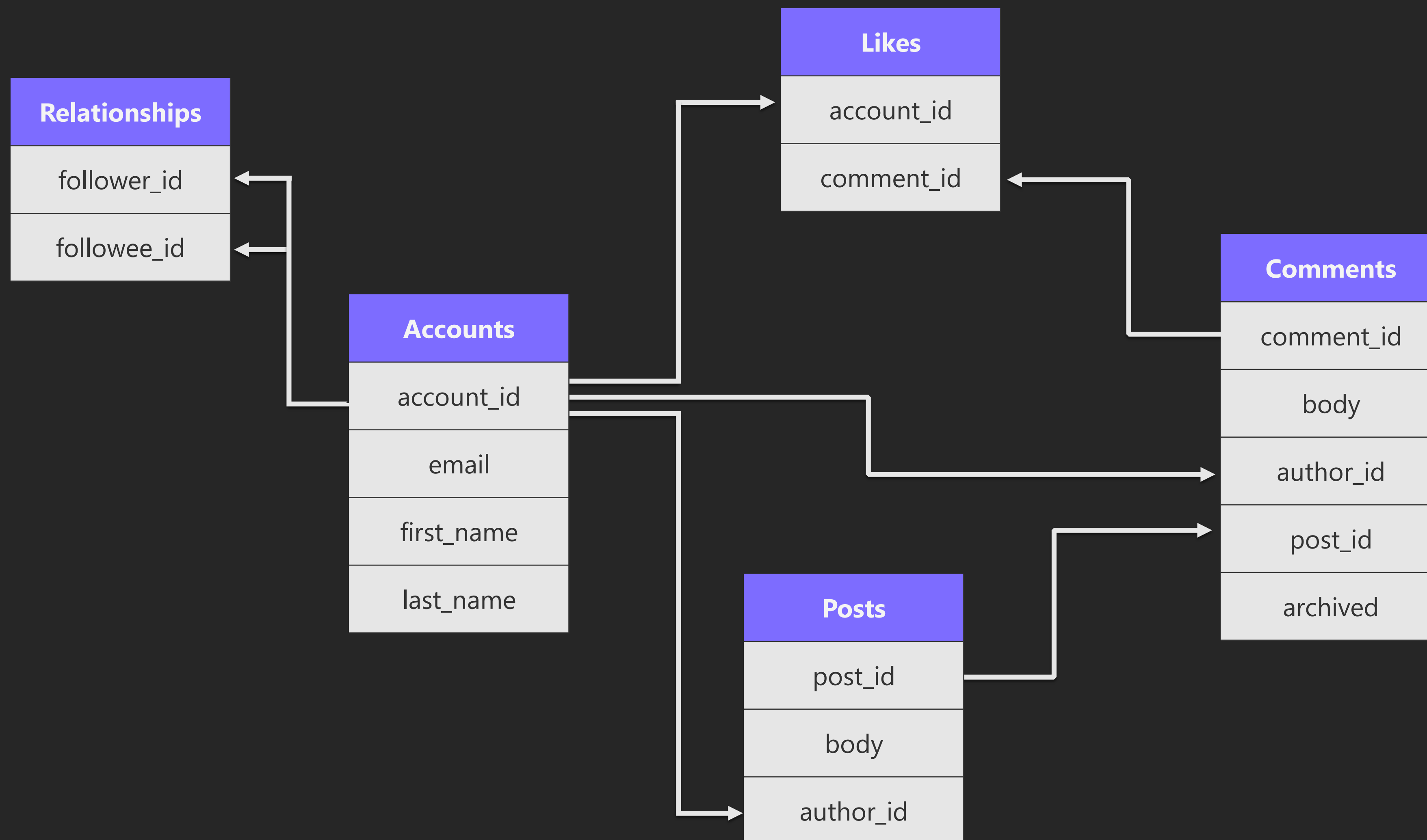| country_id	country_name	population
1	Argentina	45,000,000

CountryLanguages

language_id	country_id	primary
1	1	True
2	1	False

Languages

language_id	language_name	word_count
1	Spanish	150,000
2	English	600,000

Exercise – Convert Schema
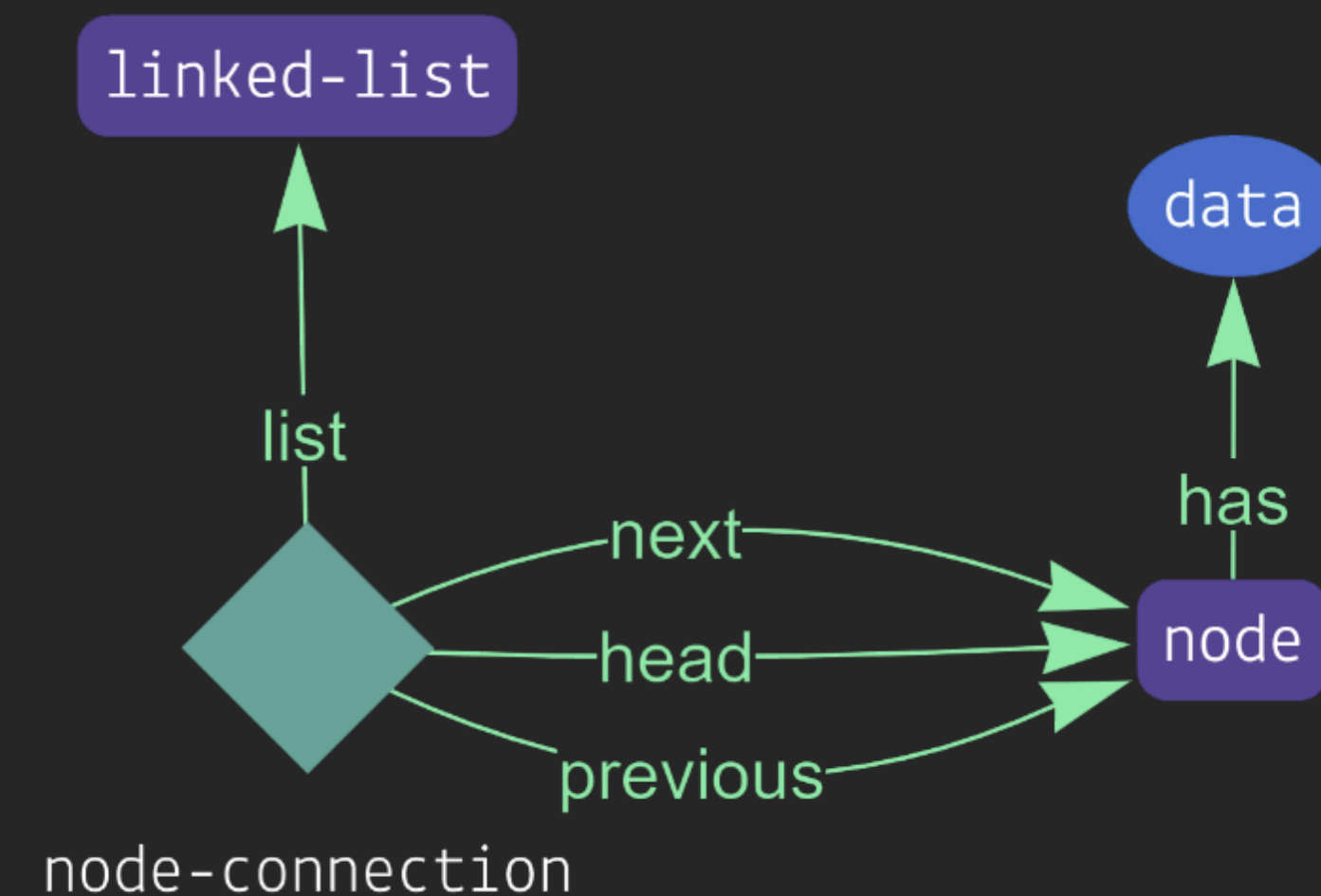
Exercise – Linked List

Objectives

1. Implement the given schema
2. Populate the knowledge base with the given data
3. Write queries to answer following questions:
   1. Who is in queue before Alexis?
   2. Who is in queue after the person after Bob?
   3. Who is the last person in the queue?

Schema

Data

Bob –> Sam -> Jessica -> Eric -> Alexis

Exercise – Find Friends-of-Friends

Objectives

1. Implement the given schema
2. Populate the knowledge base with the given data
3. Write a query to find the friends of Brandon's friends
4. Create a rule to infer two people may be acquaintances based on mutual friends

Data

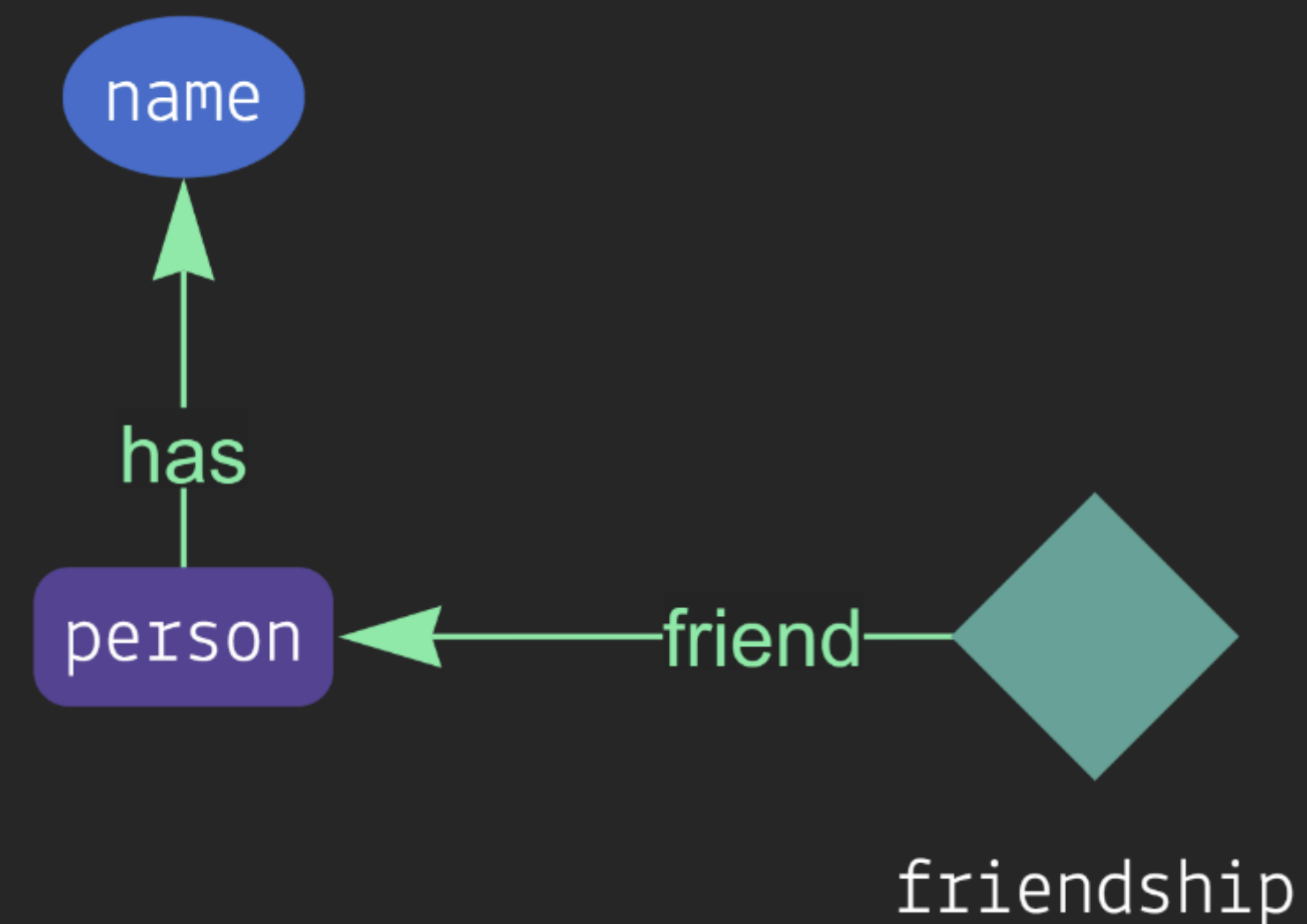Brandon's Friends:
- Travis
- Mark
- Bob
- Melissa

Mark's Friends:
- Bob

Melissa's Friends:
- Louise
- Alice

Schema

Exercise – Find Skilled Colleagues

Objectives

1. Implement the given schema
2. Populate the knowledge base with the given data
3. Write a query to find everyone who is good at Kotlin and Java
4. Write a query to find everyone who has no experience with C#
5. Extend the graph to add the ability to assign skill level (weight)
   1. Update Eric to have a skill level of 3 in current skills
   2. Update Larry to have a skill level of 2 in current skills
   3. Update Sergey to have a skill level of 4 in current skills
   4. Write a query to find everyone with a skill level greater than 2 in C#

Data

Company: Google
    Employees:
        Eric
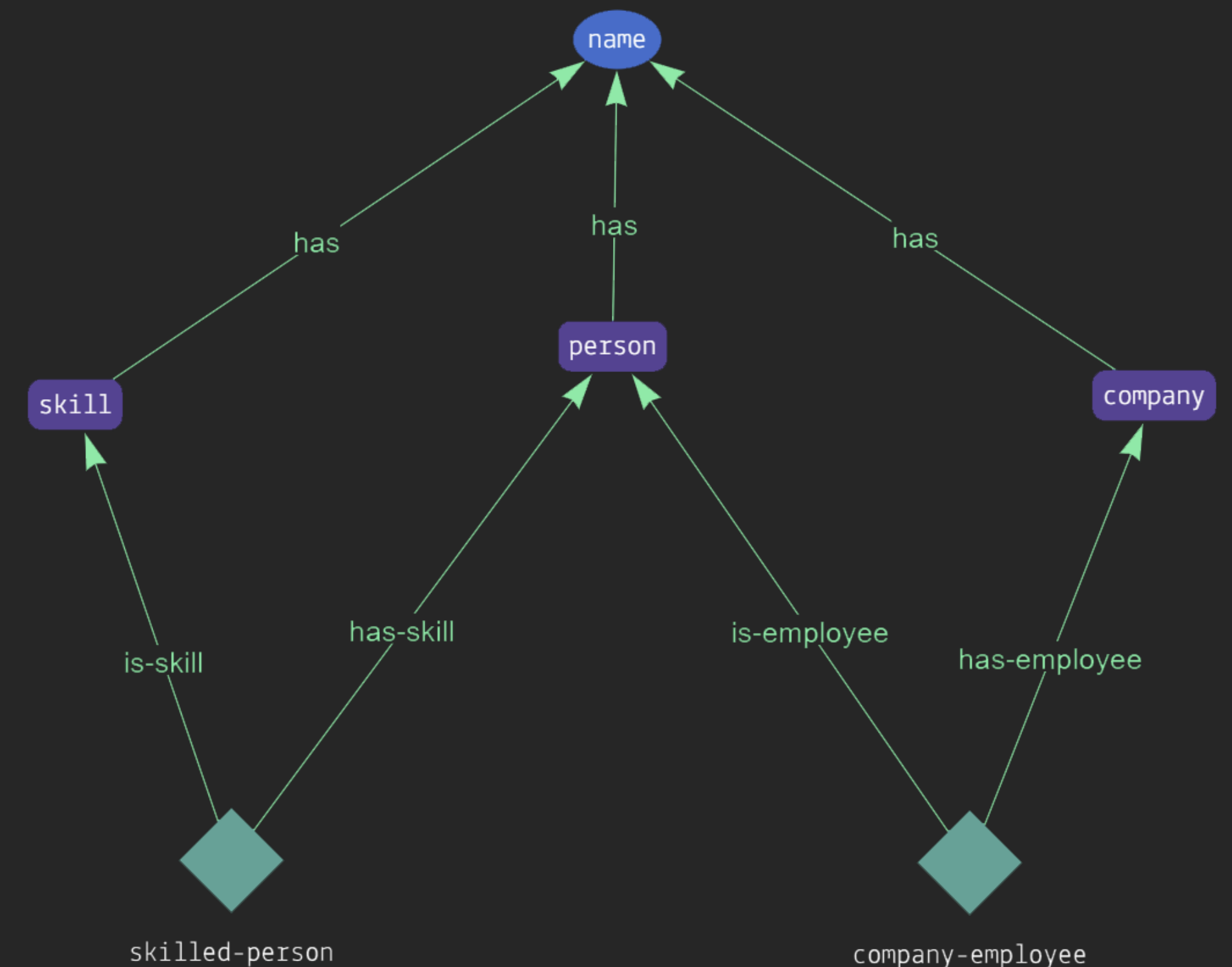   Skills: Python, C#
        Larry
            Skills: C#, Kotlin, Java
        Sergey
            Skills: Kotlin, Java

Schema

Questions/Comments?

Resources:

- https://grakn.ai

- https://neo4j.com

codebrig