Relational Knowledge Graph Management Systems

relationalAI

Martin Bravenboer, VP Engineering

RelationalAI

- RAI brings together experts from DB, PL, ML and business domains
- RAI develops a **Relational KGMS**

~~Database~~

Relational Knowledge Graph Management System (KGMS)

- Why Relational?
- How do we define knowledge?
- What does it mean to manage a knowledge graph?

Rel - A Relational Language

- RAI designs **Rel** - A declarative language for modern data applications
- Modern data applications are knowledge graphs

The **relational model** represents all data using first-order relations

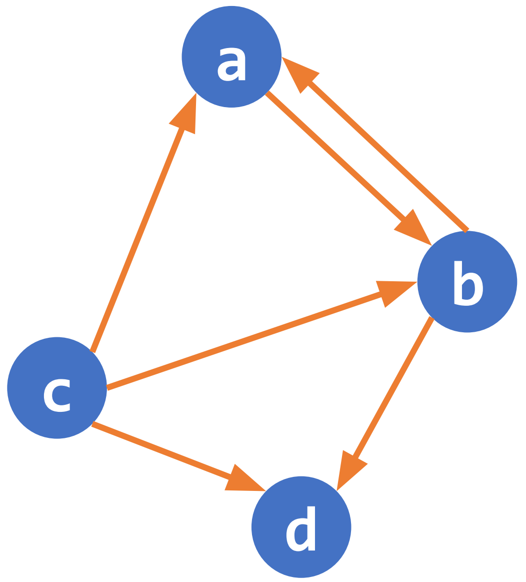
The purpose is independence of application logic
from changes in data representation:

data independence

SQL is not the relational model

(inadequacy of SQL is often used as motivation for new data models)

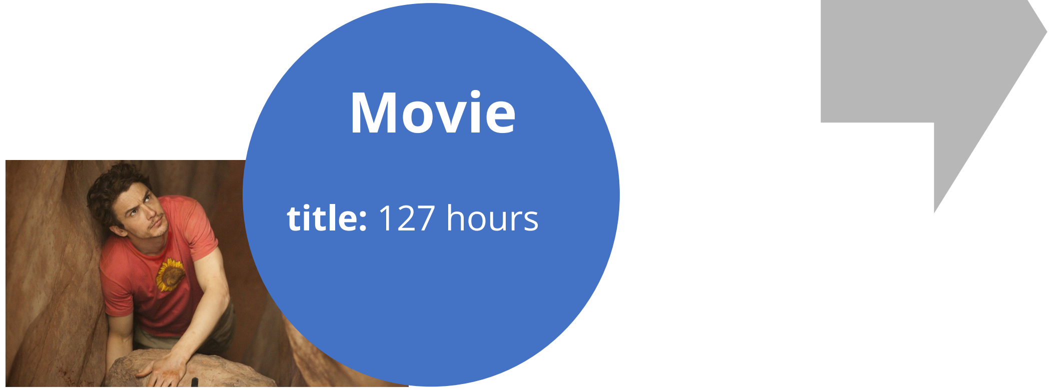
Directed graphs can be represented with relations



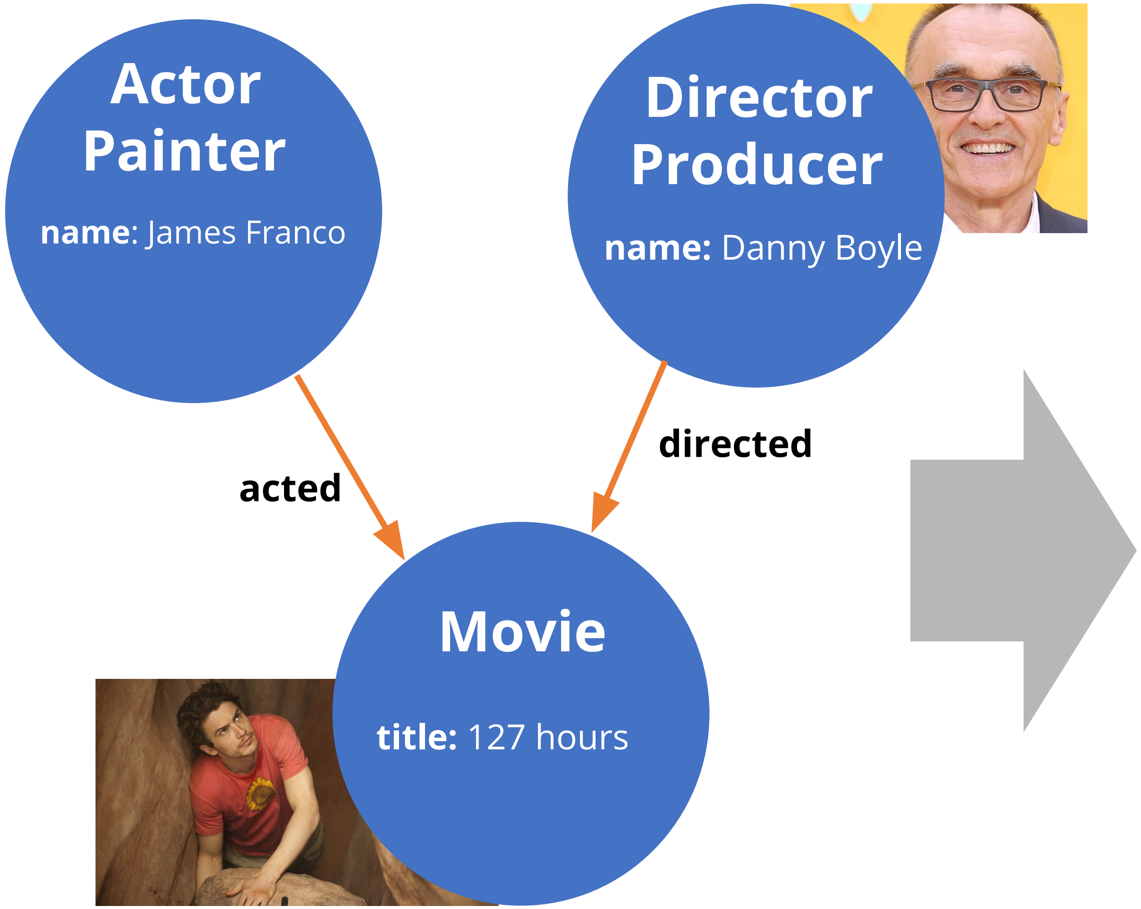
```
edge(a, b)

edge(b, a)
edge(b, d)

edge(c, a)
edge(c, b)
edge(c, d)
```

Property graphs can be represented with relations

```
movie(m)
title(m, "127 Hours")
```

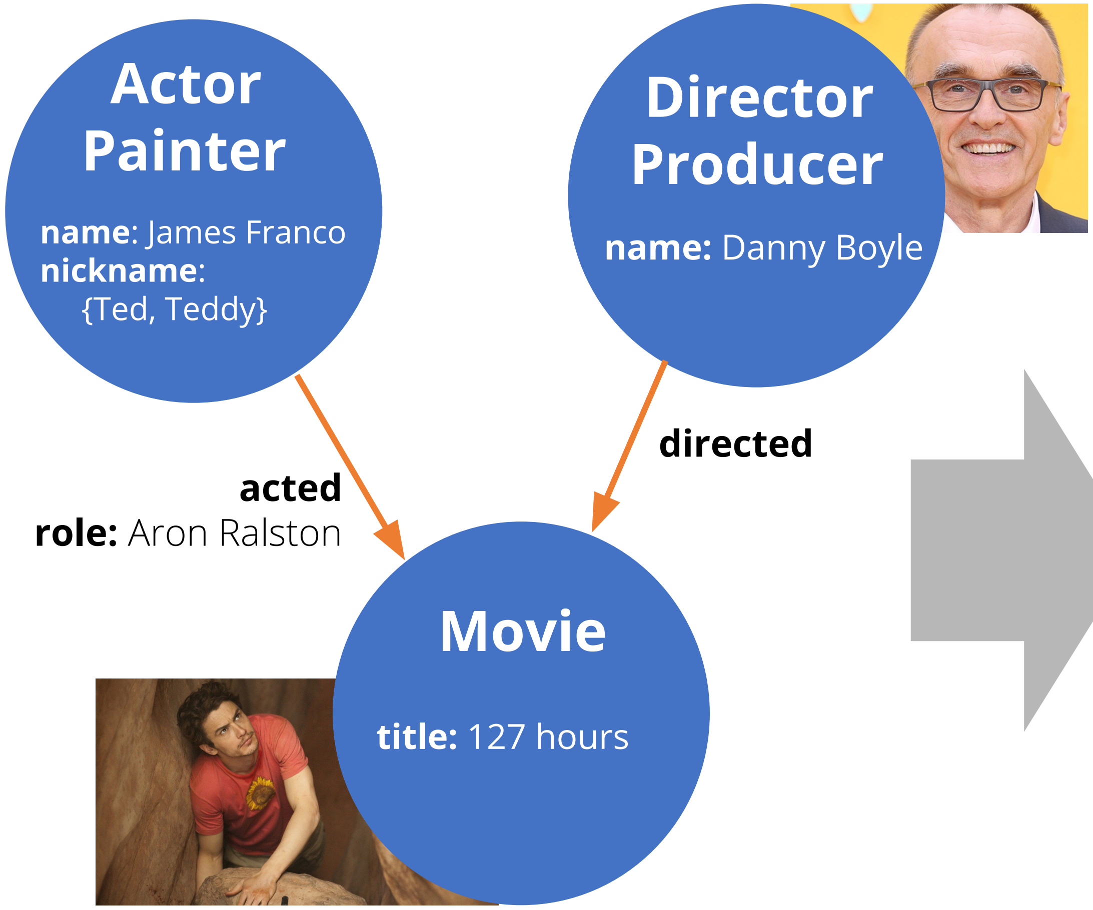


Property graphs can be represented with relations



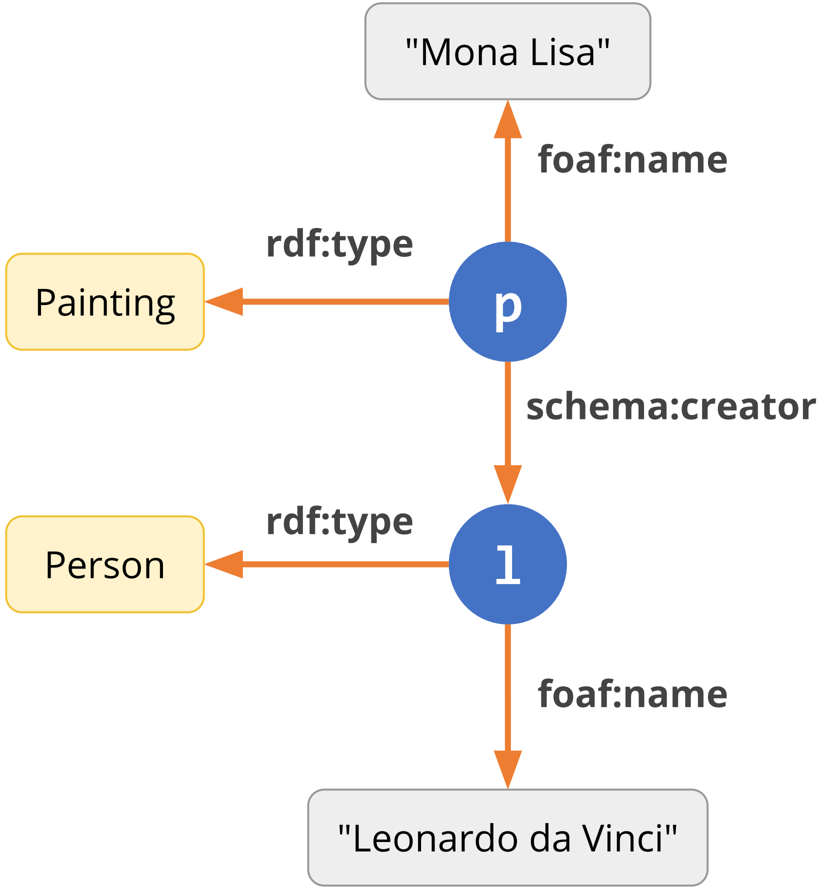
```
movie(m)
title(m, "127 Hours")

director(d)
producer(d)
name(d, "Danny Boyle")

directed(d, m)

actor(j)
painter(j)
name(j, "James Franco")



acted(j, m)
```

Property graphs can be represented with relations



```
movie(m)
title(m, "127 Hours")

director(d)
producer(d)
name(d, "Danny Boyle")

directed(d, m)

actor(j)
painter(j)
name(j, "James Franco")
nickname(j, "Ted")
nickname(j, "Teddy")

acted(j, m)
role(j, m, "Aron Ralston")
```

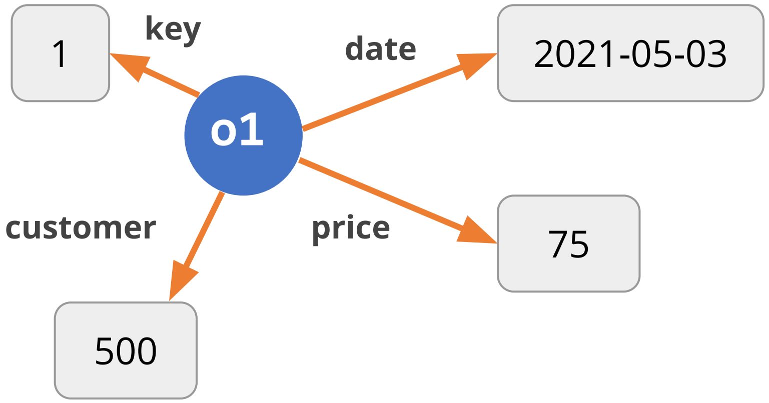
RDF can be represented with relations



```
type(p, :Painting)
name(p, "Mona Lisa")


type(l, :Person)
name(l, "Leonardo da Vinci")

creator(p, l)
```

SQL tables can be represented with relations

order			
key	customer	date	price
1	500	2021-05-03	75
2	...	...	...



```
order(1, ~~500, 2021-05-03,~~ 75)
```

```
key(o1, 1)
customer(o1, 500)
date(o1, 2021-05-03)
price(o1, 75)
```

```
order:key(o1, 1)
order:customer(o1, 500)
order:date(o1, 2021-05-03)
order:price(o1, 75)
```

Relations can be used for many, if not all, data models and are particularly well-suited for graphs.

Next:

How do we define knowledge?

Relational KGMS needs expressive reasoning

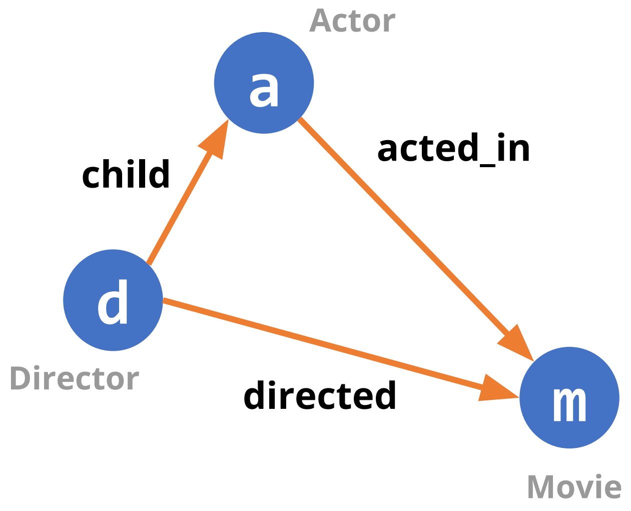
Reasoning: derive ***edges, labels and nodes*** in the knowledge graph.

We use a triangle query in a movie graph as a recurring example.

Triangle Query in Rel

```
def employs_child(d, a, m) =
    directed(d, m) and
    child(d, a) and
    acted_in(a, m)
```

Triangle Graph Pattern



Relational KGMS needs expressive reasoning

Make spouse symmetric

```
def spouse(a, b) = spouse(b, a)
```

Label nodes for the Netherlands and Dutch citizens

```
def netherlands(c) = country_code(c, "NL")
def dutch(p) = exists(c: citizen(p, c) and netherlands(c))
```

Count the number of edges in the graph

```
def edge_count = count[edge]
```

Count the number of outgoing edges for every node x

```
def outdegree[x in vertex] = count[edge[x]]
```

For every team t, sum the salary of all team members p

```
def salaries_by_team[t in team] = sum[salary[p] for p in member[t]]
```

Relational KGMS needs expressive reasoning

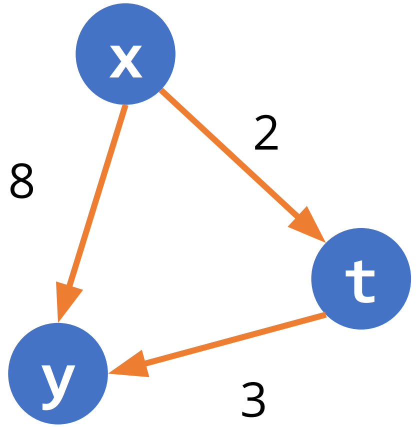
Reasoning can involve recursion. Recursion in Rel is user-defined.

```
def reachable(a, b) = edge(a, b)
def reachable(a, b) = exists(t: edge(a, t) and reachable(t, b))
```

Rel supports mutual recursion.

Recursion can involve aggregation:

```
def shortest_path[x, y] =
    Min[length[x, y]
        ∪
        shortest_path[x, t] +length[t, y] from t]
```



Cached computations need to be incrementally evaluated wrt changes in inputs (**dynamic graphs**). Our approach is based on differential dataflow.

Relational KGMS needs integrity and knowledge

Two uses of constraints:

1. Constraints that derive new nodes and edges (reasoning)
2. Constraints for data integrity

Instead of repairing problems, data integrity can be enforced with integrity constraints:

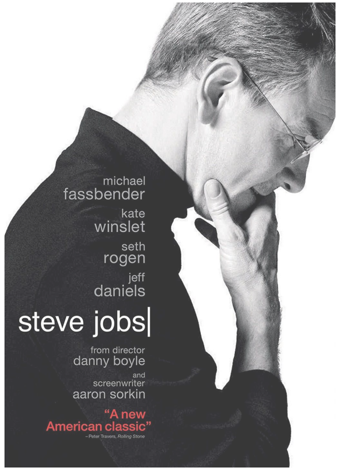
```
ic forall(x: actor(x) implies person(x))
ic actor ⊆ person

ic forall(x, y: parent(x, y) implies person(x) and person(y))

ic function(birthdate)
```

Deeper knowledge:

```
ic symmetric(spouse)
ic transitive(located_in)
```

Relational KGMS needs schema-level features

Key difference of **SPARQL, Cypher, Graql, GQL, GSQL** vs **SQL** is the intrinsic ability to write queries over **schema as well as data**. This is critical for knowledge graph applications.



SPARQL

```
SELECT ?rel1 ?x ?rel2
WHERE {
    <http://dbpedia.org/resource/Danny_Boyle> ?rel1 ?x .
    ?x ?rel2 <http://dbpedia.org/resource/Aaron_Sorkin> .
}
```

Generic graph algorithms critically depend on this capability as well.

Relational KGMS needs schema-level features

Reasoning with specific as well as generic edge types:

```
with movie_graph use ...

def connections(a, rel1, b, rel2, c) =
    name(a, "Danny Boyle") and
    name(c, "Aaron Sorkin") and
    movie_graph(rel1, a, b) and
    movie_graph(rel2, b, c)
```

Reuse a generic algorithm over the movie graph:

```
def movie_pagerank   = pagerank[movie_graph[_]]
def movie_similarity = cosine_similarity[movie_graph[_]]
```

Generic reasoning used to define `dutch(person), british(person)` etc

```
def movie_graph(relname, person) =
    movie_graph:citizen(person, country) and
    movie_graph:citizenship(country, relname)
```

Relations can be used for many, if not all, data models and are particularly well-suited for graphs.

To define knowledge, essential features are expressive reasoning and integrity constraints, including recursion and schema-level features.

Next:

Why are new join algorithms needed for knowledge graphs?

Relational KGMS needs better join algorithms

Join algorithms used in SQL-based relational databases are **binary join algorithms**. For knowledge graphs intermediate results are too large. Example:

```
directed(d, m) and child(d, a) and acted_in(a, m)
```

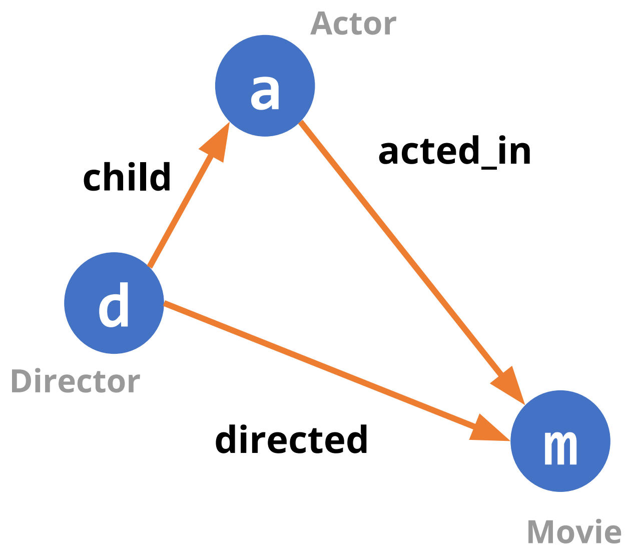
Binary join options:

```
d, m, a: directed(d, m) and child(d, a)
```
not selective: most directors have children!

```
d, m, a: directed(d, m) and acted_in(a, m)
```
not selective: every movie has a director and actors!

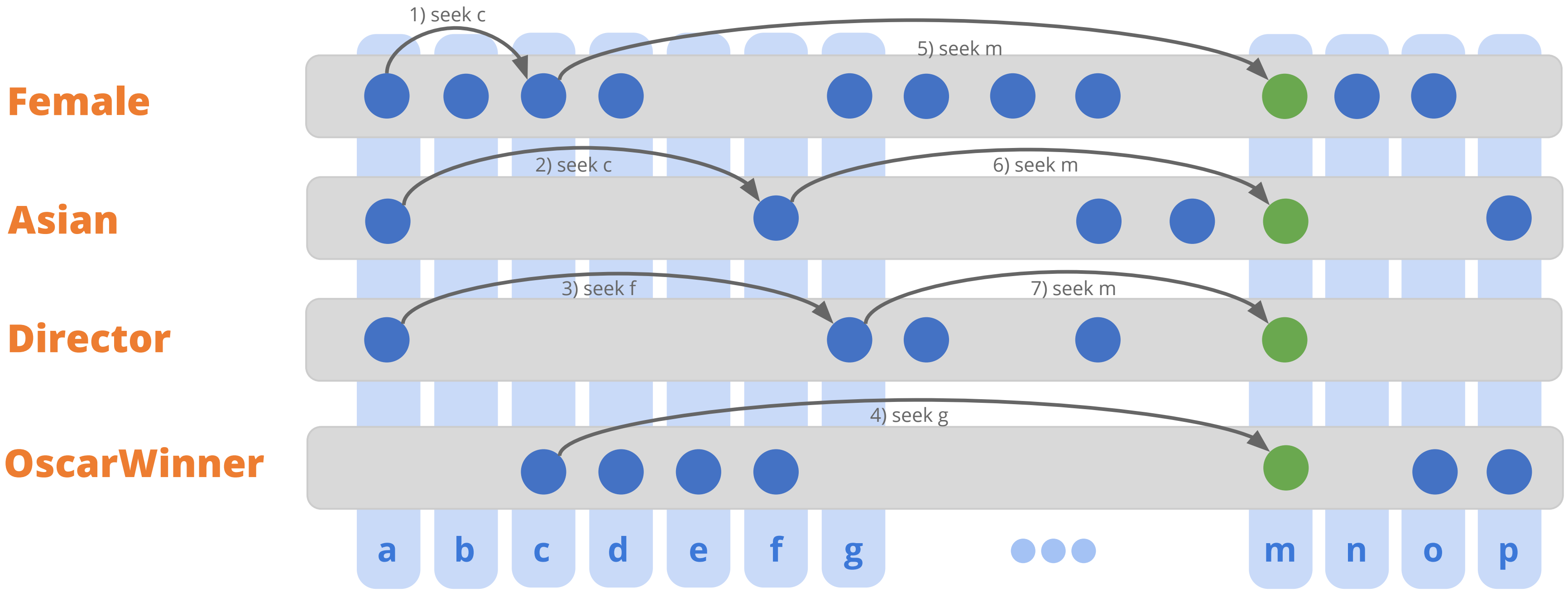
```
d, m, a: child(d, a) and acted_in(a, m)
```
not selective: every actor has parents!

This is one reason for the stigma 'joins are bad'.



Triangle Graph Pattern

Relational KGMS needs better join algorithms



Worst-case optimal join (WCOJ) algorithms use the sparsity of all relations to narrow down the search.

Leapfrog Triejoin (LFTJ), GenericJoin and Dovetail Join are WCOJ algorithms.

Chloé Zhao

Relational KGMS needs better join algorithms

Multi-way joins are used **continuously** during reasoning, not just for unary joins

```
child(d, a) and directed(d, m) and acted_in(a, m)
```

Given a variable ordering of d, a, m (determined by query optimizer)

```
child(d, _)
directed(d, _)
```

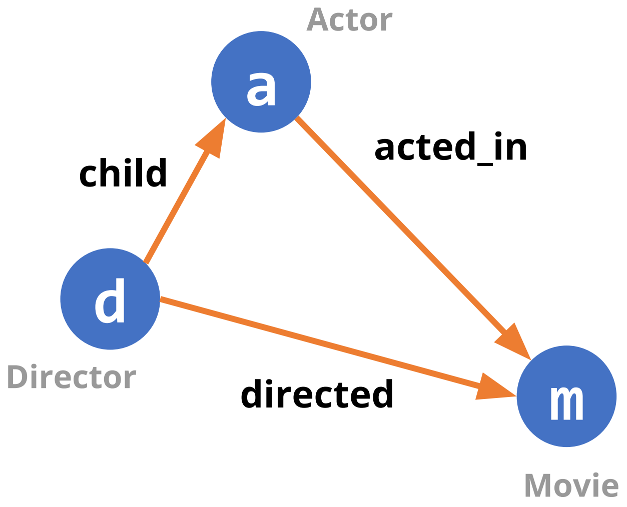
find directors **d** who `directed` some movie and have *some* `child`

```
child[d](a)
acted_in(a, _)
```

find children **a** of director **d** who `acted_in` *some movie*

```
directed[d](m)
acted_in[a](m)
```

find movies **m** `directed` by **d** and `acted_in` by actor **a** (intersection)



RAI KGMS uses a JIT-compiled WCOJ algorithm called Dovetail join.

Relational KGMS needs better join algorithms

RAI KGMS - Rel

```
def triangle_count[E] =
    count[a, b, c:
        E(a, b) and
        E(a, c) and
        E(b, c)]
```

Neo4J - Cypher

```
CALL gds.triangleCount.stream(
  graphName: String,
  configuration: Map
)
YIELD
  nodeId: Integer,
  triangleCount: Integer
```

Java™

TigerGraph - GSQL

```
CREATE QUERY tri_count_fast*EXT*() FOR GRAPH *graph* {
        SumAccum<int> @@cnt, @outdegree;
        SetAccum<int> @neighbors;
        all = {*vertex-types*};
        all = SELECT s
              FROM all:s
              ACCUM s.@outdegree += *s_outdegrees*;
        tmp = SELECT s
              FROM all:s -(*edge-types*) -:t
              ACCUM IF s == t THEN
                      s.@outdegree += -1
              END;
        tmp = SELECT t
              FROM all:s-(*edge-types*)-> :t
              WHERE s.@outdegree > t.@outdegree OR
                  (s.@outdegree == t.@outdegree AND
                   getvid(s) > getvid(t))
              ACCUM t.@neighbors += getvid(s);
        tmp = SELECT t
              FROM all:s-(*edge-types*)-> :t
              WHERE s != t
              ACCUM @@cnt += COUNT(
                 s.@neighbors INTERSECT t.@neighbors);
        PRINT @@cnt/2 AS num_triangles;
}
```

Relational KGMS needs better join algorithms

- **Worst-case Optimal Join Algorithms**
  Hung Q. Ngo, Ely Porat, Christopher Ré, Atri Rudra.
  PODS 2012 (Best Paper Award).

- **Leapfrog Triejoin: A Simple, Worst-Case Optimal Join Algorithm**
  Todd L. Veldhuizen
  ICDT 2014 (Best Newcomer Award)

- **A Worst-case Optimal Join Algorithm for SPARQL**
  Aidan Hogan, Cristian Riveros, Carlos Rojas, Adrián Soto
  ISWC 2019

- **Worst-Case Optimal Graph Joins in Almost No Space**
  Diego Arroyuelo, Aidan Hogan, Gonzalo Navarro, Juan L. Reutter, Javiel Rojas-Ledesma and Adrián Soto
  SIGMOD 2021

Relations can be used for many, if not all, data models and are particularly suited for knowledge graphs.

To define knowledge, essential features are expressive reasoning and integrity constraints, including recursion and schema-level features.

WCOJ algorithms are key for graph queries and reasoning in knowledge graph systems.

Next:
How do we leverage knowledge in the system?

Relational KGMS needs semantic optimization

Goal: users write high-level declarative specifications, **not algorithms**

RAI KGMS uses **knowledge** (semantics) **to optimize** the program.

General mathematical **knowledge**

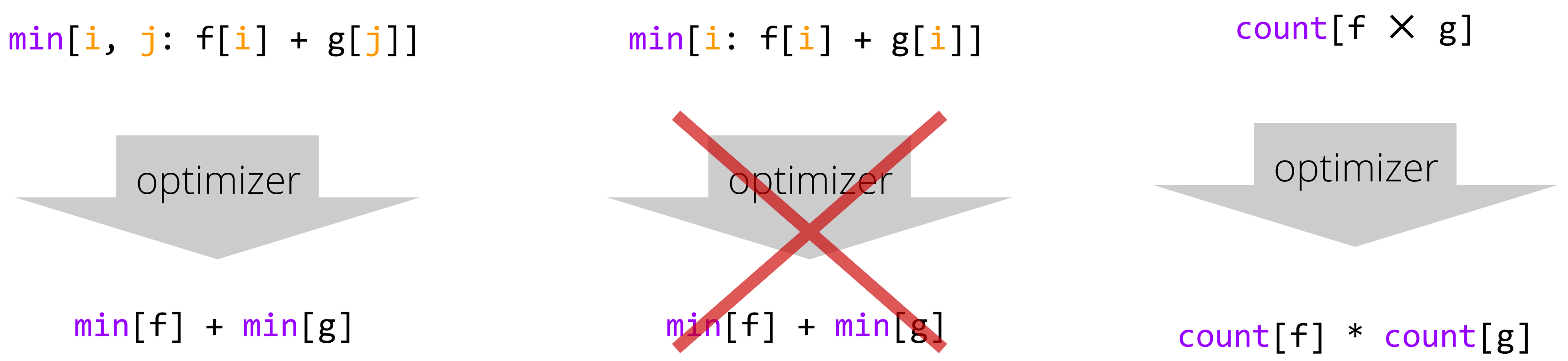
commutativity, associativity, semiring, ...

Domain-specific **knowledge**

from constraints for reasoning and integrity (functional dependencies)

Relational KGMS needs semantic optimization

Using mathematical knowledge in semantic optimization



```
min[i, j: f[i] + g[j]]
```

```
min[f] + min[g]
```

```
min[i: f[i] + g[i]]
```

~~`min[f] + min[g]`~~

```
count[f × g]
```

```
count[f] * count[g]
```

Relational KGMS needs semantic optimization

Push min aggregation into a recursive path definition to derive Dijkstra's algorithm

```
def path[x, y] = length[x, y]
def path[x, y] = path[x, t] + length[t, y] from t

def shortest_path[x, y] = min[path[x, y]]
```

optimizer

```
def shortest_path[x, y] =
    min[length[x, y]
        ∪
        shortest_path[x, t] + length[t, y] from t]
```

Relational KGMS needs semantic optimization

Instead of tying **shortest_path** to a specific relation **length**, we want a reusable abstraction

```
def path[x, y] = length[x, y]
def path[x, y] = path[x, t] + length[t, y] from t

def shortest_path[x, y] = min[path[x, y]]
```

Introduce a higher-order parameter **W** for the weighted edge relation:

```
def path[W, x, y] = W[x, y]
def path[W, x, y] = path[x, t] + W[t, y] from t

def shortest_path[W, x, y] = min[path[W, x, y]]
```

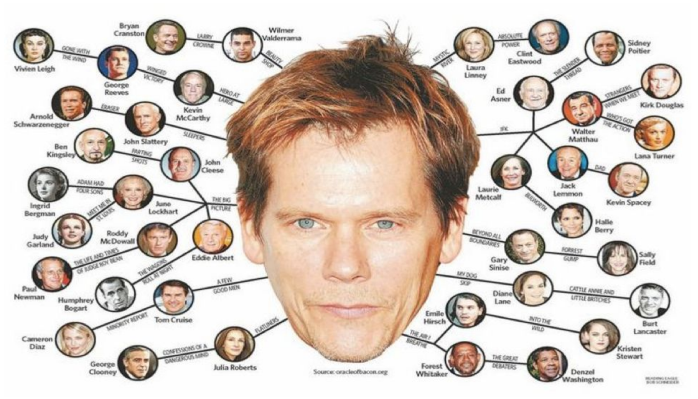
Relational KGMS needs semantic optimization

Optimize **all-pairs** shortest path to **single-source** shortest path using **demand transformation**

```
def bacon_number[p] =
    shortest_path[co_star × 1, KevinBacon, p]
```

optimizer

```
def bacon_number[p] =
    Min[num:
        co_star(KevinBacon, p) and num = 1
        or exists(t: co_star(t, p) and num = bacon_number[t] + 1)
    ]
```



Relational KGMS needs semantic optimization

- **FAQ: Questions Asked Frequently**
  Mahmoud Abo Khamis, Hung Q. Ngo, Atri Rudra
  PODS 2016 (Best Paper Award)

- **What Do Shannon-type Inequalities, Submodular Width, and Disjunctive Datalog Have to Do with One Another?**
  Mahmoud Abo Khamis, Hung Q. Ngo, Dan Suciu
  PODS 2017

- **Precise complexity analysis for efficient Datalog queries**
  K Tuncay Tekle, Yanhong A Liu
  PPDP 2010

Relations can be used for many, if not all, data models and are particularly suited for knowledge graphs.

To define knowledge, essential features are expressive reasoning and integrity constraints, including recursion and schema-level features.

WCOJ algorithms are key for graph queries and reasoning in knowledge graph systems.

Deep knowledge of the application logic is used to derive efficient algorithms from declarative specifications.

Next:
How do we incorporate machine learning?

Relational Machine Learning

The Rel library includes **feature engineering** capabilities defined in Rel itself:

```
def zscore_normalization[F, x...] = (F[x...] - mean[F]) / stddev[F]
```

Predict and **cost** functions can be used with **autodiff** and trained with gradient descent.

```
def mse[YHAT, Y] = sum[x... : (Y[x...] - YHAT[x...]) ^ 2] / count[Y]
def rmse[YHAT, Y] = sqrt[mse[YHAT, Y]]
```

$$\sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

Use to define a cost function for a specific linear regression model:

```
def cost[W] = rmse[predict_linear[features, W], life_satisfaction]
```

With our research network we have developed training methods that do not require creating a design matrix of features and **operate directly on the relational structure**, exploiting knowledge of the relational model (exactly as in **semantic optimization**)

Deep Learning in a Relational Language

Graph embeddings and neural networks are expressible in Rel and can be used for prediction (model deployment) as well as training.

```
def relu[v where v <= 0] = 0
def relu[v where v > 0] = v

def sigmoid[v] =
    1.0 / (1.0 + natural_exp[-1.0 * v])
```

$$y = relu\left(\sum_i w_i x_i + b\right)$$

$$sigmoid(z) = \frac{1}{1 + e^{-z}}$$

```
def activation[t, l, j] =
   sigmoid[sum[k: weights[l, j, k]  * activation[t, prevl, j]] + biases[l, j]]
   from prevl where next_layer[prevl] = l
```

(this is still very experimental)

Relational Machine Learning

- **A Layered Aggregate Engine for Analytics Workloads**
  M. Schleich, D. Olteanu, M. Abo Khamis, H. Ngo and X. Nguyen
  SIGMOD 2019

- **Learning Models over Relational Data Using Sparse Tensors and Functional Dependencies**
  Mahmoud Abo Khamis, Hung Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich
  PODS 2018, TODS 2020

- **The Relational Data Borg is Learning**
  Dan Olteanu, VLDB 2020 Keynote
  https://www.youtube.com/watch?v=0ic0jMjOpM0
  https://www.youtube.com/watch?v=kWm-0BnbEoU

- **Structure-Aware Machine Learning over Multi-Relational Databases**
  Maximilian Schleich, PhD thesis
  Honorable mention for the 2021 SIGMOD Jim Gray Doctoral Dissertation Award

- **Relational Knowledge Graphs as the Foundation for Artificial Intelligence**
  Molham Aref
  https://www.youtube.com/watch?v=VpyGbjUzG7Y

RelationalAI KGMS Capabilities

Capabilities covered today

Reasoning

- Recursion
- Incremental computation

Data integrity and knowledge

- General integrity constraints

Join algorithms

- Worst-case optimal joins

Semantic optimization

- Using deep mathematical knowledge to optimize programs

Machine learning

- Cost functions and autodiff

Key system-related capabilities

Storage/compute separation

- Scale arbitrarily for storage and up/down for compute.

Versioning and temporal features

- Dynamic knowledge graph (changes)
- Refer to past versions of the knowledge graph
- Fork a knowledge graph and submit pull request
- Reasoning with temporal relations

Ingest essentially any data

- CSV, JSON, Parquet, RDF, images, text, documents

Data and model sharing

- Enormous waste in the world copying data + cost of resulting inconsistency
- Stock markets, weather data, wikidata, etc

If you love this: RelationalAI is hiring!