Design and Implementation of the RelationalAI System

UW - Advanced Topics in Data Management
June 17, 2022

Martin Bravenboer
VP Engineering

RelationalAI

RelationalAI

The next-generation database system for intelligent data apps based on relational knowledge graphs

Innovations for Relational Knowledge Graphs

1. Immutability - Cloud native architecture
2. Expressive relational language (Rel)
3. Join algorithms
4. Semantic optimization
5. Vectorized and JIT compilation of WCOJ
6. Live - Incrementality (for data and logic)

Challenges in Database System Design and Implementation

Data structures and memory management

- In-memory performance for modern workloads exceeding available memory and disk
- Write-optimized data structures for modern workloads in cloud native architecture

Query processing

- Index selection (what indexes to define for a workload)
- Efficient evaluation of subqueries
- Relational query processing of graph workloads (complex joins)
- Materialized view selection (with views to materialize for a workload)
- Incremental computation (recursion) and maintenance wrt input changes

Concurrency and workload management

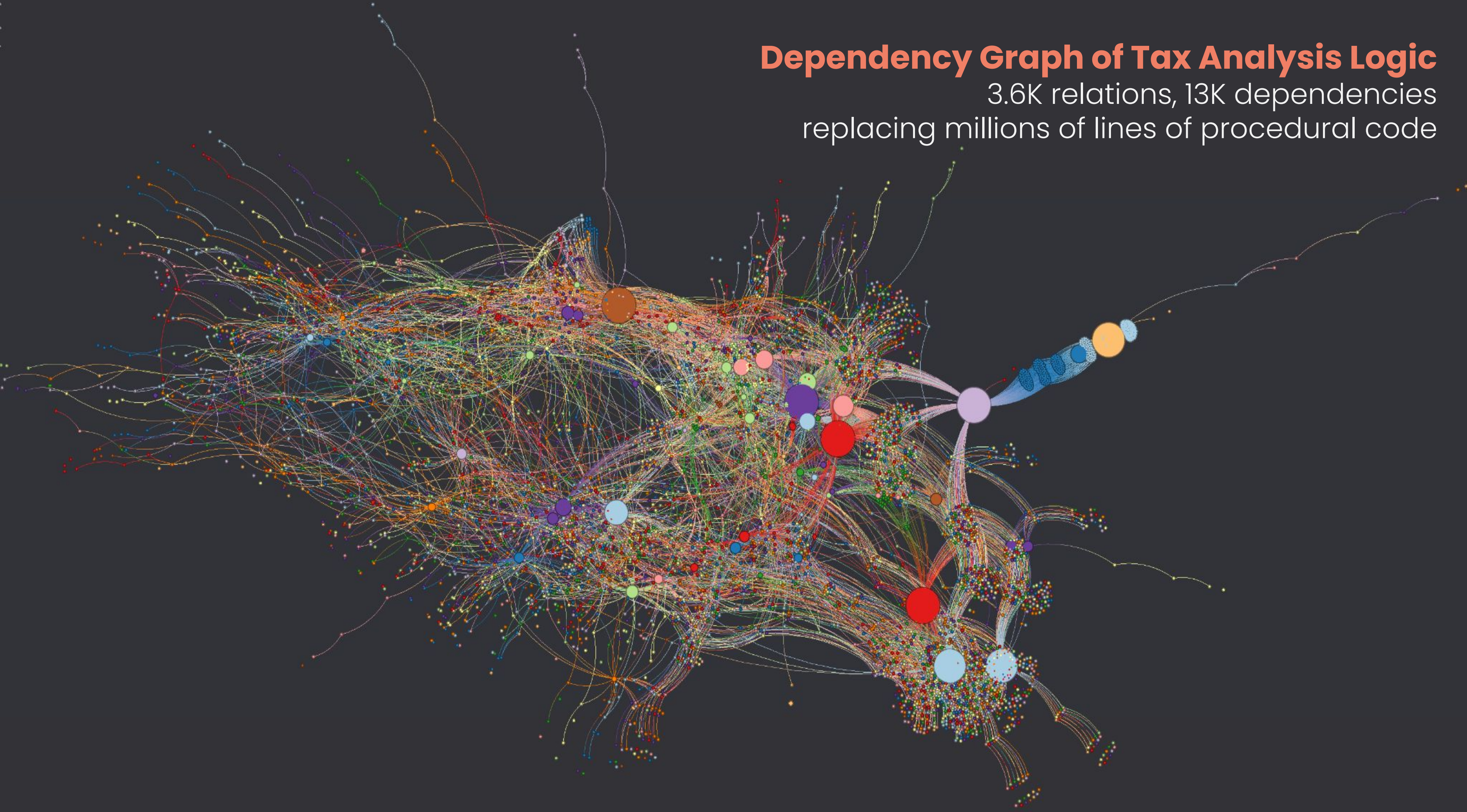
- Optimization of bottom-up vs top-down (demand-driven) evaluation
- Optimization of very large computation graphs
- Strong consistency, scalability of read-only and write workload

General Architecture

- Eliminate the split brain: moving computations to the data management system
- Maximal independence of application logic vs machine representation and organization of data (relational model)
- Language support for abstraction (libraries)
- Language support for schema abstraction (generic programming)

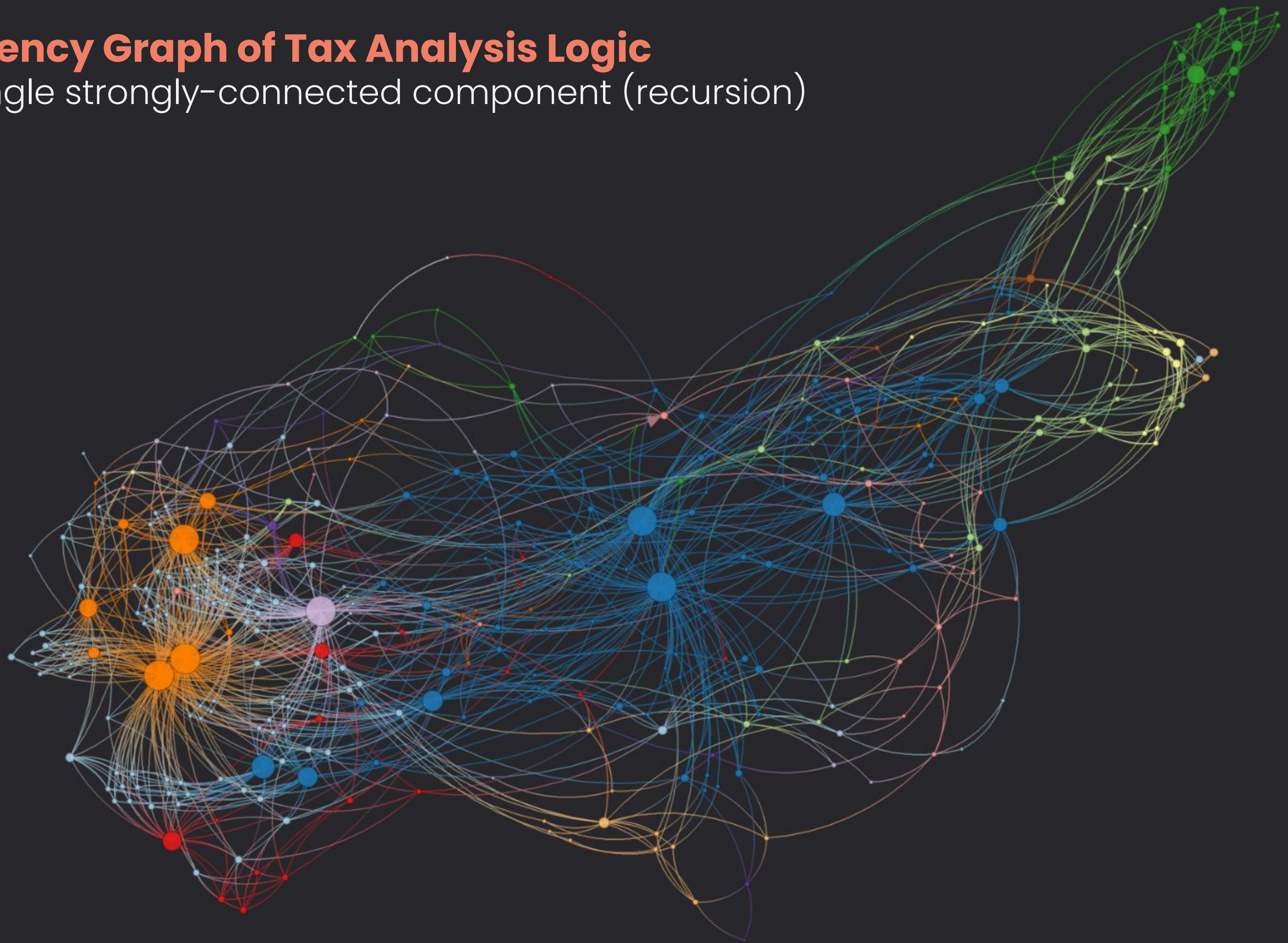
Dependency Graph of Tax Analysis Logic

3.6K relations, 13K dependencies
replacing millions of lines of procedural code



Dependency Graph of Tax Analysis Logic

Focus: Single strongly-connected component (recursion)



RelationalAI

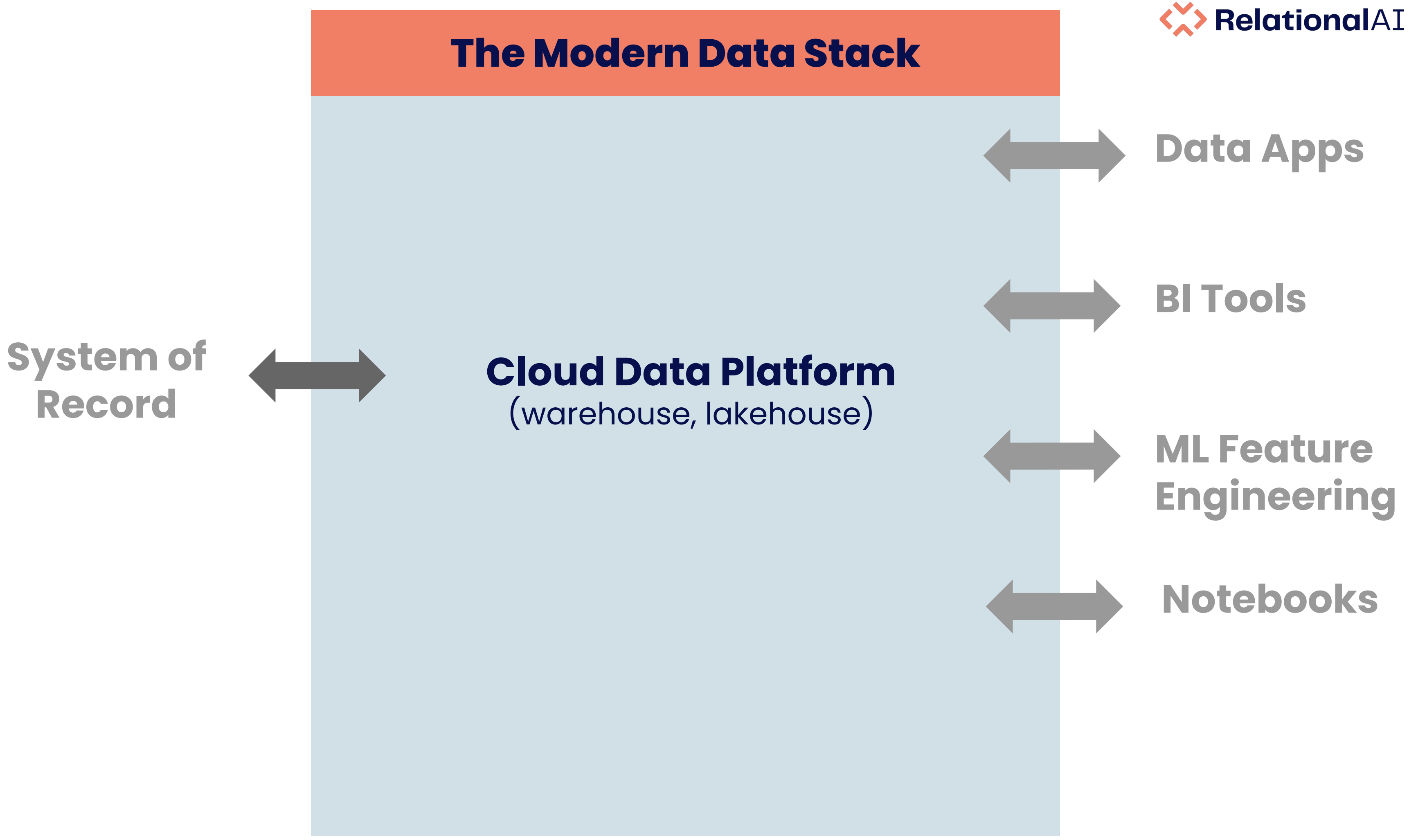
The Modern Data Stack

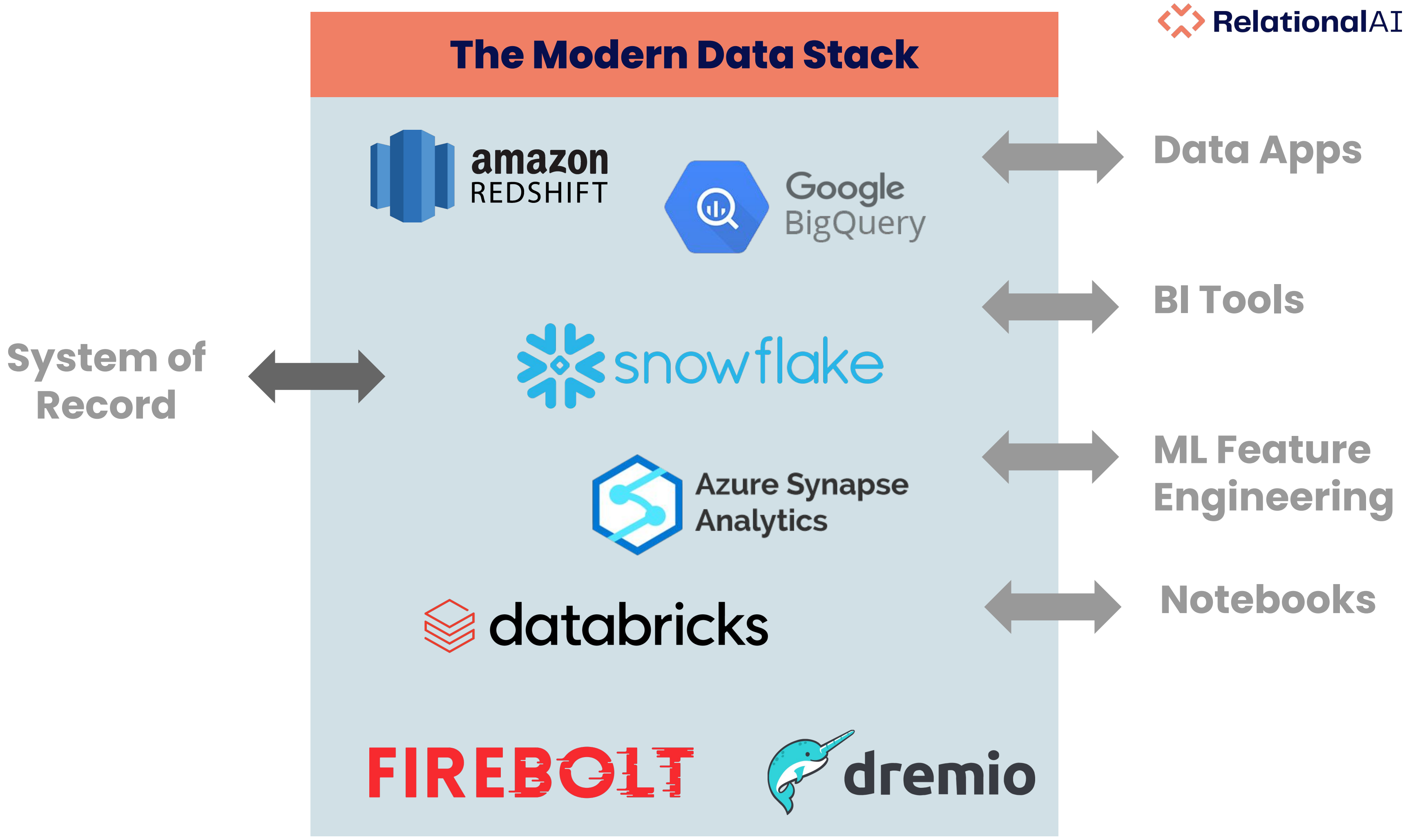
Modern database systems are cloud native

Modern database systems are implemented with cloud native architecture that **separates storage from compute**.

This architecture makes it possible to provide compelling features like:

- **Infinite storage** - store all your data regardless of structure or volume
- **Infinite compute** - run any number of workloads without concurrency limits
- **Versioning** - time-travel, zero-copy cloning
- **Fully managed** - workload management with minimal user intervention
- **Data sharing** - collaboration, live sharing, access to external data


The Modern Data Stack
System of Record
Cloud Data Platform
(warehouse, lakehouse)
Data Apps
BI Tools
ML Feature Engineering
Notebooks
RelationalAI


RelationalAI
The Modern Data Stack
amazon REDSHIFT
Google BigQuery
snowflake
Azure Synapse Analytics
databricks
FIREBOLT
dremio
System of Record
Data Apps
BI Tools
ML Feature Engineering
Notebooks

RelationalAI

The Semantic Layer

RelationalAI

Primary key

Hours
Minutes

Period
Minutes
Hours

Miles
Kilometers

Are these
exclusive

carrier	origin	destination	flight_num	flight_time	tail_num	dep_time	arr_time	dep_delay	arr_delay	taxi_out	taxi_in	distance	cancelled	diverted	id2
F9	MCI	DEN	818	89	N916FR	2005-09-26 08:23:00 UTC	2005-09-26 09:07:00 UTC	-7	-8	9	6	533	N	N	36606824
WN	LBB	ELP	819	41	N708SW	2000-08-18 07:30:00 UTC	2000-08-18 07:20:00 UTC	0	-5	7	2	295	N	N	4369021
NW	ATL	MEM	819	52	N607NW	2001-11-16 07:15:00 UTC	2001-11-16 07:29:00 UTC	-5	-9	19	3	332	N	N	11838308
DL	SLC	BOI	819	48	N296WA	2001-12-04 22:12:00 UTC	2001-12-04 23:53:00 UTC	7	[illegible]	49	4	291	N	N	12060416
WN	PHX	SAN	819	51	N391SW	2001-12-05 09:11:00 UTC	2001-12-05 09:17:00 UTC	11	[illegible]	[illegible]	[illegible]	304	N	N	12383068
WN	LAS	AUS	819	135	N519SW	2002-04-05 08:18:00 UTC	2002-04-05 12:47:00 UTC	8	[illegible]	[illegible]	[illegible]	1085	N	N	13763279
WN	SJC	LAS	819	63	N528SW	2002-07-14 06:30:00 UTC	2002-07-14 07:47:00 UTC	0	-3	11	3	386	N	N	15284777
WN	LAS	AUS	819	137	N502SW	2002-09-16 08:20:00 UTC	2002-09-16 12:52:00 UTC	0	-8	11	4	1085	N	N	16027516
DL	MSP	SLC	819	149	N3754A	2002-09-29 19:34:00 UTC	2002-09-29 21:35:00 UTC	9	15	25	7	991	N	N	16399211
DL	MSP	SLC	819	145	N3745B	2002-12-06 19:27:00 UTC	2002-12-06 21:16:00 UTC	-3	-9	18	6	991	N	N	17417961
WN	SJC	LAS	819	54	N730SW	2003-06-26 06:30:00 UTC	2003-06-26 07:44:00 UTC	0	-11	15	5	386	N	N	20460576
WN	SJC	LAS	819	60	N501SW	2003-09-15 06:30:00 UTC	2003-09-15 07:50:00 UTC	0	0	18	2	386	N	N	22113592
NW	ATL	MEM	819	51	N785NC	2003-11-20 07:11:00 UTC	2003-11-20 07:15:00 UTC	-9	-23	10	3	332	N	N	23383534
DL	MSP	ATL	819	120	N906DE	2004-02-10 09:59:00 UTC	2004-02-10 13:36:00 UTC	-6	0	30	7	906	N	N	25206983
US	CHS	CLT	820	38	N592US	2002-07-01 19:32:00 UTC	2002-07-01 20:54:00 UTC	37	64	9	35	168	N	N	15142411
FL	TPA	ATL	820	64	N955AT	2003-01-31 11:29:00 UTC	2003-01-31 12:55:00 UTC	-6	-5	13	9	406	N	N	17949329
WN	RDU	PHL	820	73	N382SW	2004-12-02 11:10:00 UTC	2004-12-02 12:31:00 UTC	0	-19	5	3	336	N	N	30796766
HP	PHX	BOS	820	0	N826AW	2005-10-25 00:00:00 UTC	2005-10-25 00:00:00 UTC	0	0	0	0	2300	Y	N	37174931
US	PBI	CLT	821	90	N624AU	2002-05-18 06:35:00 UTC	[illegible]	-5	-8	10	6	590	N	N	14247814
US	PBI	CLT	821	87	[illegible]	[illegible]	[illegible]	-7	-7	16	7	590	N	N	20289351
DL	GSO	CVG	821	70	[illegible]	[illegible]	[illegible]	1	11	14	7	330	N	N	21396171

Not a delay

Include helicopters

Risk of
messing up
aggregates

Possible values

The Semantic Layer and Data Apps

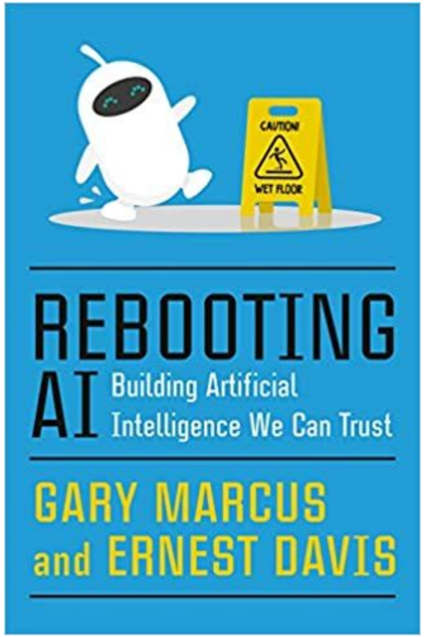
RelationalAI

Let's build a data app for an order database (TPC-H, Northwind etc)

Example functionality:

- What is the average **charge** of orders by week
- What percentage of orders were **late** this year
- If two consecutive orders for a customer are **late**, alert the account manager

The system ***cannot answer*** such questions if it does not know what **late** and **charge** mean to begin with!



Suppose, for example, that we hand you a piece of paper with this short passage:

> Two children, Chloe and Alexander, went for a walk. They both saw a dog and a tree. Alexander also saw a cat and pointed it out to Chloe. She went to pet the cat.

It is trivial to answer questions like "Who went for a walk?," in which the answer ("Chloe and Alexander") is directly spelled out in the text, but any competent reader should just as easily be able to answer questions that are not directly spelled out, like "Did Chloe see the cat?" and "Were the children frightened by the cat?" If you can't do that, you aren't really following the story. Because SQuAD didn't include any questions of this sort, it wasn't really a strong test of reading; as it turns out the new AI systems would not have been able to cope with them.[*2] By way of contrast, Gary tested the story on his daughter Chloe, then four and a half years old, and she had no trouble making the inference that the fictitious Chloe had seen a cat. (Her older brother, then not quite six years old, went a step further, musing about what would happen if the dog actually turned out to be a cat; no current AI could begin to do that.)

How many movies has Meryl Streep been in per decade

how many movies has meryl streep been in per decade ×

TRANSLATED

not right?

```
_count, decade:
  _count = count[movie_id:
    movie_release_date(movie_id, release_date) and
    movie_actor(movie_id, person_id) and
    date_year(release_date, year) and
    person_name(person_id, "Meryl Streep") and
    year_decade(year, decade)
    from release_date, person_id, year
  ]
```

RESULTS

_count	decade
5	1970
11	1980
12	1990
15	2010
22	2000

Showing all 5 results

What movies has Johnny Depp acted in since 2015

what movies has johnny depp acted in since 2015 ×

TRANSLATED

not right?

```
movie_id, title, release_date:
  movie_title(movie_id, title) and
  movie_release_date(movie_id, release_date) and
  movie_actor(movie_id, person_id) and
  date_after_year(release_date, 2015) and
  person_name(person_id, "Johnny Depp")
  from person_id
```

RESULTS

movie_id	title	release_date
166426	Pirates of the Caribbean: Dead Men Tell No Tales	2017-05-23
241259	Alice Through the Looking Glass	2016-05-25
259316	Fantastic Beasts and Where to Find Them	2016-11-16
290825	Yoga Hosers	2016-07-08
382079	Donald Trump's The Art of the Deal: The Movie	2016-02-10

Showing all 5 results

```
measure: cumulative_total_revenue {
    type: running_total
    sql: ${total_sale_price} ;;
}

measure: total_gross_margin {
    type: sum
    value_format_name: usd
    sql: ${gross_margin} ;;
}

measure: percent_of_total_gross_margin {
    type: percent_of_total
    sql: ${total_gross_margin} ;;
}
```

https://docs.looker.com/reference

```
dimension: is_order_paid {
    type: yesno
    sql: ${status} = 'paid' ;;
}

dimension: full_name {
    type: string
    sql: CONCAT(${first_name}, ' ', ${last_name}) ;;
}

dimension: profit {
    type: number
    sql: ${revenue} - ${cost} ;;
}

dimension: distance_to_pickup {
    type: distance
    start_location_field: customer.home_location
    end_location_field: rental.pickup_location
    units: miles
}

dimension: store_location {
    type: location
    sql_latitude: ${store_latitude} ;;
    sql_longitude: ${store_longitude} ;;
}
```

Malloy

https://github.com/looker-open-source/malloy

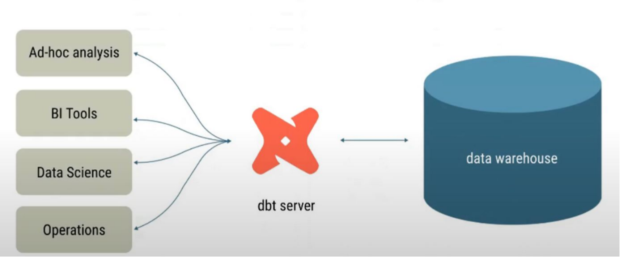
```
source: users is table('malloy-data.ecomm.users') {
    primary_key: id
    dimension: full_name is concat(first_name, ' ', last_name)
    measure: user_count is count()
}


source: iowa is table('malloy-data.iowa_liquor_sales.sales_deduped') {
    dimension: gross_margin is 100 * (state_bottle_retail - state_bottle_cost) / nullif(state_bottle_retail, 0)
    dimension: price_per_100ml is state_bottle_retail / nullif(bottle_volume_ml, 0) * 100
}


source: flights is table('malloy-data.faa.flights') {
    dimension: distance_km is distance / 1.609344
    measure: flight_count is count()
    rename: destination_code is destination
}
```

```
order_payments as (
   select
       order_id,
       {% for payment_method in payment_methods -%}
       sum(case when payment_method = '{{ payment_method }}'
               then amount else 0 end
       ) as {{ payment_method }}_amount,
       {% endfor -%}
       sum(amount) as total_amount
   from payments
   group by order_id)
```

```
customer_orders as (
    select
        customer_id,
        min(order_date) as first_order,
        max(order_date) as most_recent_order,
        count(order_id) as number_of_orders
    from orders
    group by customer_id)
```

```
gitlab_dotcom_issues_source AS (
    SELECT *
    FROM {{ ref('gitlab_dotcom_issues_source')}}
    {% if is_incremental() %}
      WHERE updated_at >= (SELECT MAX(updated_at) FROM {{this}})
    {% endif %})
```

```
upvote_count AS (
    SELECT
      awardable_id                                        AS dim_issue_id,
      SUM(IFF(award_emoji_name LIKE 'thumbsup%', 1, 0))   AS thumbsups_count,
      SUM(IFF(award_emoji_name LIKE 'thumbsdown%', 1, 0)) AS thumbsdowns_count,
      thumbsups_count - thumbsdowns_count                 AS upvote_count
    FROM gitlab_dotcom_award_emoji_source
    WHERE awardable_type = 'Issue'
    GROUP BY 1)
```



Knowledge Graphs

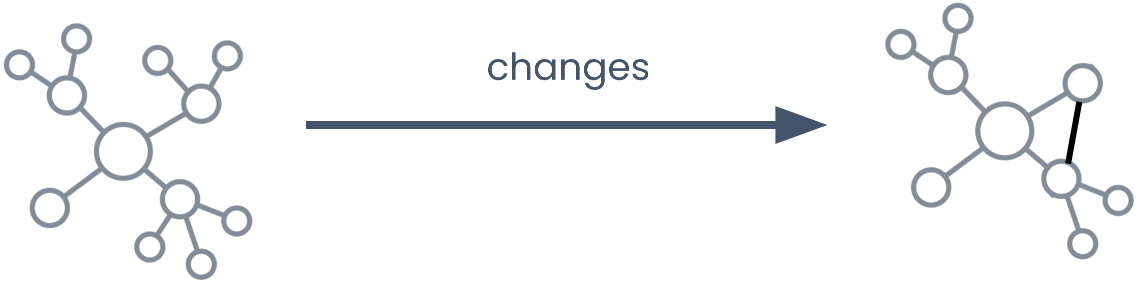
Semantic Layer

Reasoning

Views

RelationalAI

Data Apps, Reasoning & Knowledge



Views / Reasoning / Knowledge / The Semantic Layer

RelationalAI

The Semantic Layer - Rel

The Semantic Layer and Data Apps

RelationalAI

Let's build a data app for an order database (TPC-H, Northwind etc)

Example functionality:

- What is the average **charge** of orders by week
- What percentage of orders were **late** this year
- If two consecutive orders for a customer are **late**, alert the account manager

The system ***cannot answer*** such questions if it does not know what **late** and **charge** mean to begin with!

Data Apps, Reasoning & Knowledge

RelationalAI

Given: extendedprice, discount, tax

```
def item_revenue[o, num] =
    extendedprice[o, num] * (1 - discount[o, num])

def revenue[o] =
    sum[num: item_revenue[o, num]]


def item_charge[o, num] =
    item_revenue[o, num] * (1 + tax[o, num])

def charge[o] =
    sum[num: item_charge[o, num]]
```

Data Apps, Reasoning & Knowledge

RelationalAI

Given: commitdate, receiptdate

```
def received_late(o, num) =
    commitdate[o, num] < receiptdate[o, num]

def late(o) =
    exists(num: received_late(o, num))
```

RelationalAI

Primary key

Hours
Minutes

Period
Minutes
Hours

Miles
Kilometers

Are these exclusive

carrier	origin	destination	flight_num	flight_time	tail_num	dep_time	arr_time	dep_delay	arr_delay	taxi_out	taxi_in	distance	cancelled	diverted	id2
F9	MCI	DEN	818	89	N916FR	2005-09-26 08:23:00 UTC	2005-09-26 09:07:00 UTC	-7	-8	9	6	533	N	N	36606824
WN	LBB	ELP	819	41	N708SW	2000-08-18 07:30:00 UTC	2000-08-18 07:20:00 UTC	0	-5	7	2	295	N	N	4369021
NW	ATL	MEM	819	52	N607NW	2001-11-16 07:15:00 UTC	2001-11-16 07:29:00 UTC	-5	-9	19	3	332	N	N	11838308
DL	SLC	BOI	819	48	N296WA	2001-12-04 22:12:00 UTC	2001-12-04 23:53:00 UTC	7	41	49	4	291	N	N	12060416
WN	PHX	SAN	819	51	N391SW	2001-12-05 09:11:00 UTC	2001-12-05 09:17:00 UTC	11	[illegible]	[illegible]	[illegible]	304	N	N	12383068
WN	LAS	AUS	819	135	N519SW	2002-04-05 08:18:00 UTC	2002-04-05 12:47:00 UTC	8	[illegible]	[illegible]	[illegible]	1085	N	N	13763279
WN	SJC	LAS	819	63	N528SW	2002-07-14 06:30:00 UTC	2002-07-14 07:47:00 UTC	0	-3	11	3	386	N	N	15284777
WN	LAS	AUS	819	137	N502SW	2002-09-16 08:20:00 UTC	2002-09-16 12:52:00 UTC	0	-8	11	4	1085	N	N	16027516
DL	MSP	SLC	819	149	N3754A	2002-09-29 19:34:00 UTC	2002-09-29 21:35:00 UTC	9	15	25	7	991	N	N	16399211
DL	MSP	SLC	819	145	N3745B	2002-12-06 19:27:00 UTC	2002-12-06 21:16:00 UTC	-3	-9	18	6	991	N	N	17417961
WN	SJC	LAS	819	54	N730SW	2003-06-26 06:30:00 UTC	2003-06-26 07:44:00 UTC	0	-11	15	5	386	N	N	20460576
WN	SJC	LAS	819	60	N501SW	2003-09-15 06:30:00 UTC	2003-09-15 07:50:00 UTC	0	0	18	2	386	N	N	22113592
NW	ATL	MEM	819	51	N785NC	2003-11-20 07:11:00 UTC	2003-11-20 07:15:00 UTC	-9	-23	10	3	332	N	N	23383534
DL	MSP	ATL	819	120	N906DE	2004-02-10 09:59:00 UTC	2004-02-10 13:36:00 UTC	-6	0	30	7	906	N	N	25206983
US	CHS	CLT	820	38	N592US	2002-07-01 19:32:00 UTC	2002-07-01 20:54:00 UTC	37	64	9	35	168	N	N	15142411
FL	TPA	ATL	820	64	N955AT	2003-01-31 11:29:00 UTC	2003-01-31 12:55:00 UTC	-6	-5	13	9	406	N	N	17949329
WN	RDU	PHL	820	73	N382SW	2004-12-02 11:10:00 UTC	2004-12-02 12:31:00 UTC	0	-19	5	3	336	N	N	30796766
HP	PHX	BOS	820	0	N826AW	2005-10-25 00:00:00 UTC	2005-10-25 00:00:00 UTC	0	0	0	0	2300	Y	N	37174931
US	PBI	CLT	821	90	N624AU	2002-05-18 06:35:00 UTC	[illegible]	-5	-8	10	6	590	N	N	14247814
US	PBI	CLT	821	87	[illegible]	[illegible]	[illegible]	-7	-7	16	7	590	N	N	20289351
DL	GSO	CVG	821	70	[illegible]	[illegible]	[illegible]	1	11	14	7	330	N	N	21396171

Not a delay

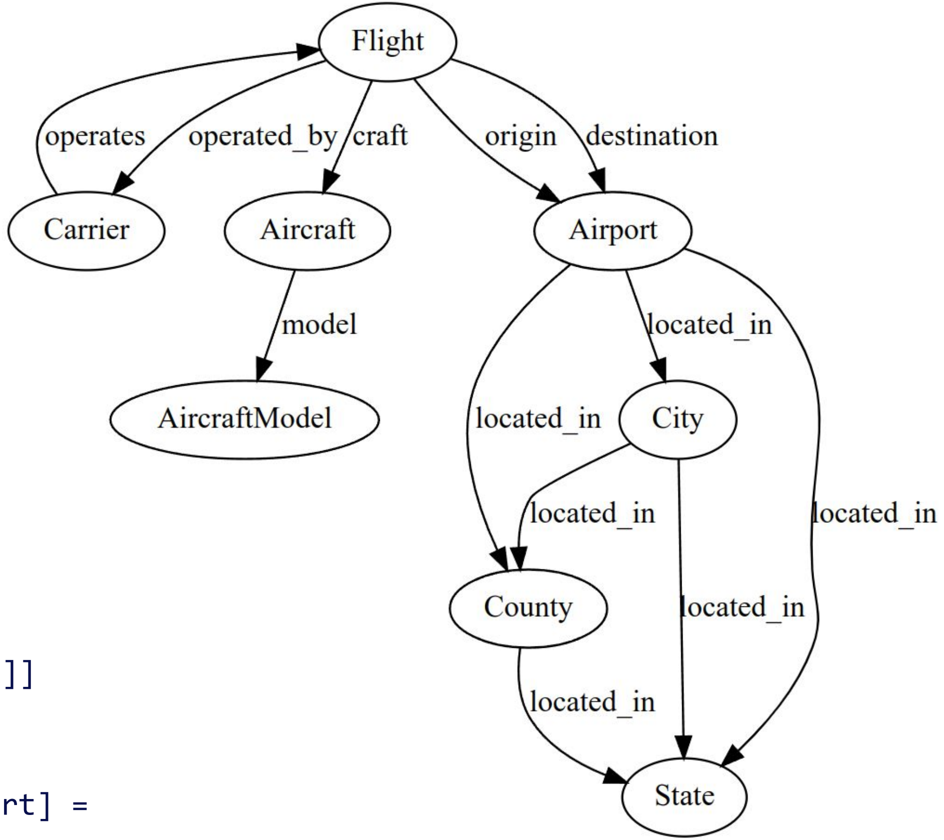
Include helicopters

Risk of messing up aggregates

Possible values

Better Conceptual Model

```
def Heliport(x in Airport) =
    fac_type(x, "HELIPORT")


def cancelled(f in Flight) =
    flight(f) and flight_cancelled(f, "Y")


def arrival_delay[f in Flight] =
    ^Minute[maximum[0, arr_delay[f]]


def coordinate[x in Airport] =
    ^LLA[latitude[x], longitude[x], elevation[x]]


def airport_distance[a1 in Airport, a2 in Airport] =
    distance[coordinate[a1], coordinate[a2]]
```



RelationalAI

Reasoning manages app logic with the data

Reasoning subsumes business logic now implemented procedurally in languages like Java, C#, Python, Scala, PL/SQL, T/SQL etc.

Fixing the **"split brain"** problem where the data is managed in one layer and knowledge/semantics in another will have huge impact.

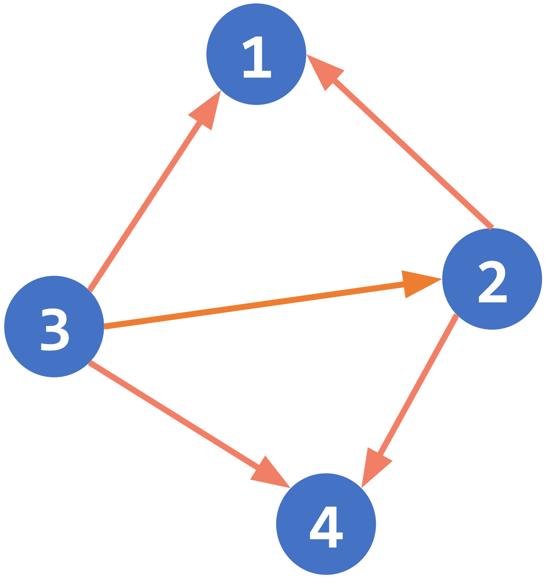
Bringing the app logic to the data makes it possible for one (cloud native) system to manage the semantics, integrity, and resources needed for the application.

With thanks to Peter Bailis…

RelationalAI

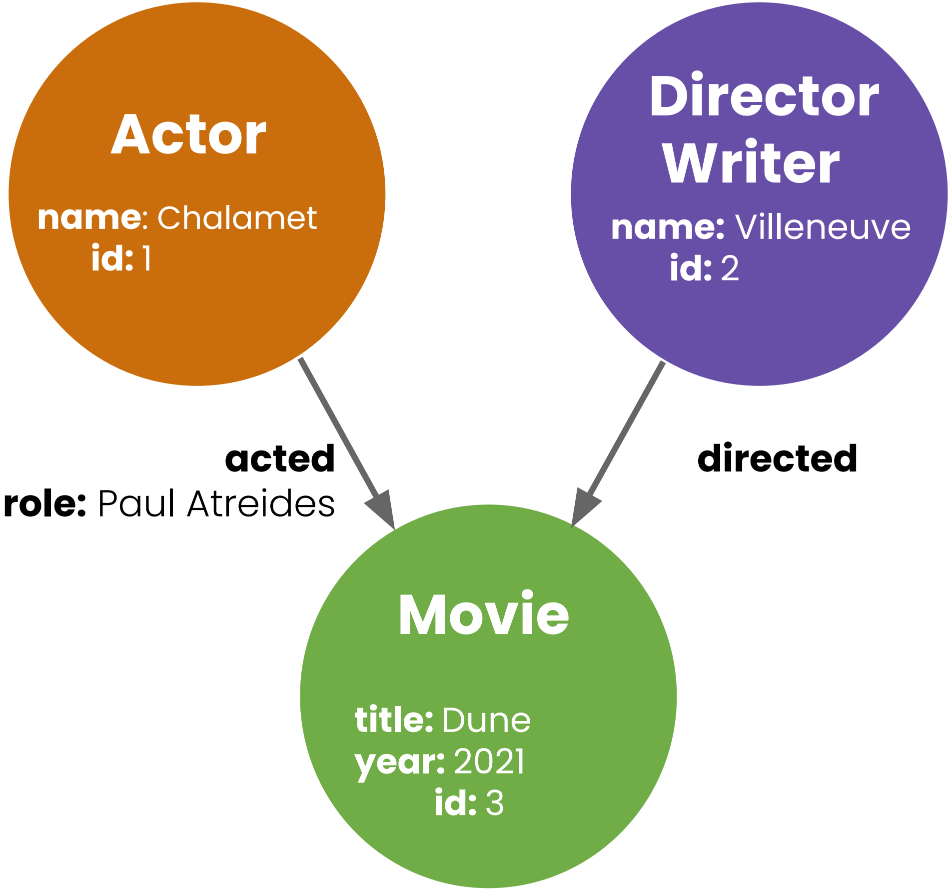
Relational Models

Directed Graphs as a Relation




```
edge(2, 1)
edge(2, 4)

edge(3, 1)
edge(3, 2)
edge(3, 4)
```

Labelled Property Graphs as Relational Graphs

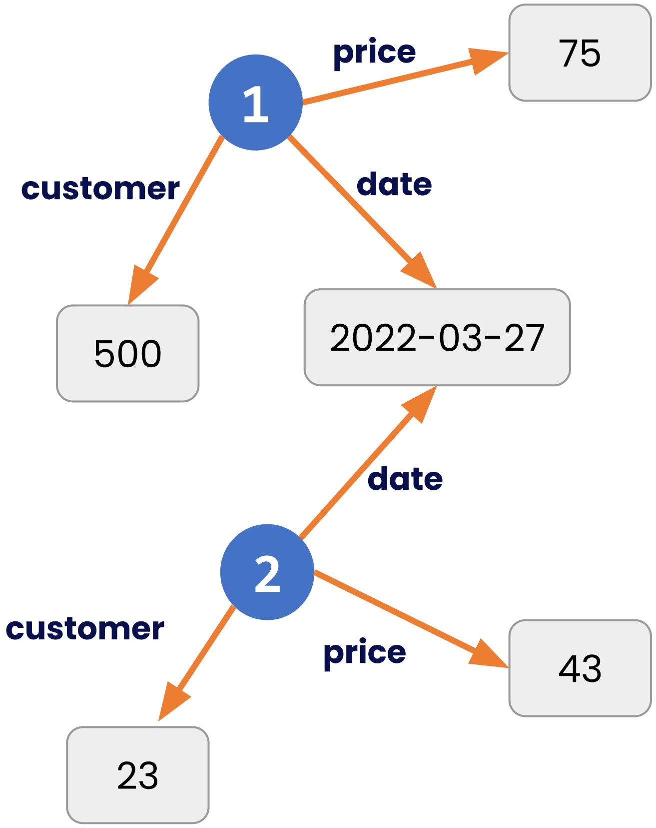



```
movie(3)
title(3, "Dune")
year(3, 2021)

director(2)
writer(2)
name(2, "Villeneuve")

directed(2, 3)

actor(1)
name(1, "Chalamet")

acted(1, 3)
role(1, 3, "Paul Atreides")
```

Tables as a Collection of Relations

orderkey	customer	date	price
1	500	2022-03-27	75
2	23	2022-03-27	43

```
customer(1, 500)
customer(2, 23)

date(1, 2022-03-27)
date(2, 2022-03-27)

price(1, 75)
price(2, 43)
```



SQL tables are in a sense a modularity construct, grouping relations with the same primary key.

Recall …

Tensors are relations!

$$\begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 2 & 0 & 1 \end{bmatrix}$$

i	j	#
0	0	1
0	1	2
1	1	1
2	0	2
?	?	?

RelationalAI

Tensors as Relations

vector

$$\begin{bmatrix} 4 \\ 1 \\ 8 \end{bmatrix}$$

```
(1, 4)
(2, 1)
(3, 8)
```

binary relation

matrix

$$\begin{bmatrix} -1.3 & 0.6 \\ 20.4 & 5.5 \\ 9.7 & -6.2 \end{bmatrix}$$

```
(1, 1, -1.3)
(1, 2,  0.6)
(2, 1, 20.4)
(2, 2,  5.5)
(3, 1,  9.7)
(3, 2, -6.2)
```

ternary relation

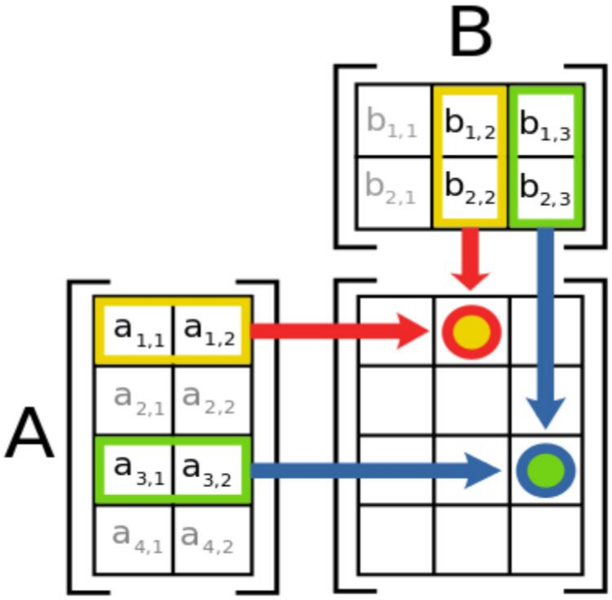
A relational database system that is effective for tensors would be an outstanding proof-point for the relational model.

(and imagine the data management benefits this would have for ML systems!)

RelationalAI

Tensors as Relations: Matrix Multiplication

Deep Learning with Relations at NeurIPS



Matrix multiplication diagram.svg, CC BY-SA 3.0, User:Bilou

Math

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

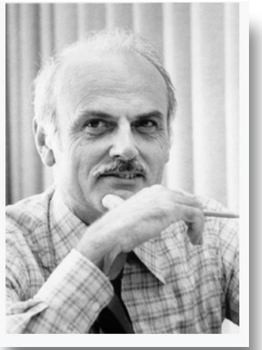
Rel *Our new relational language*

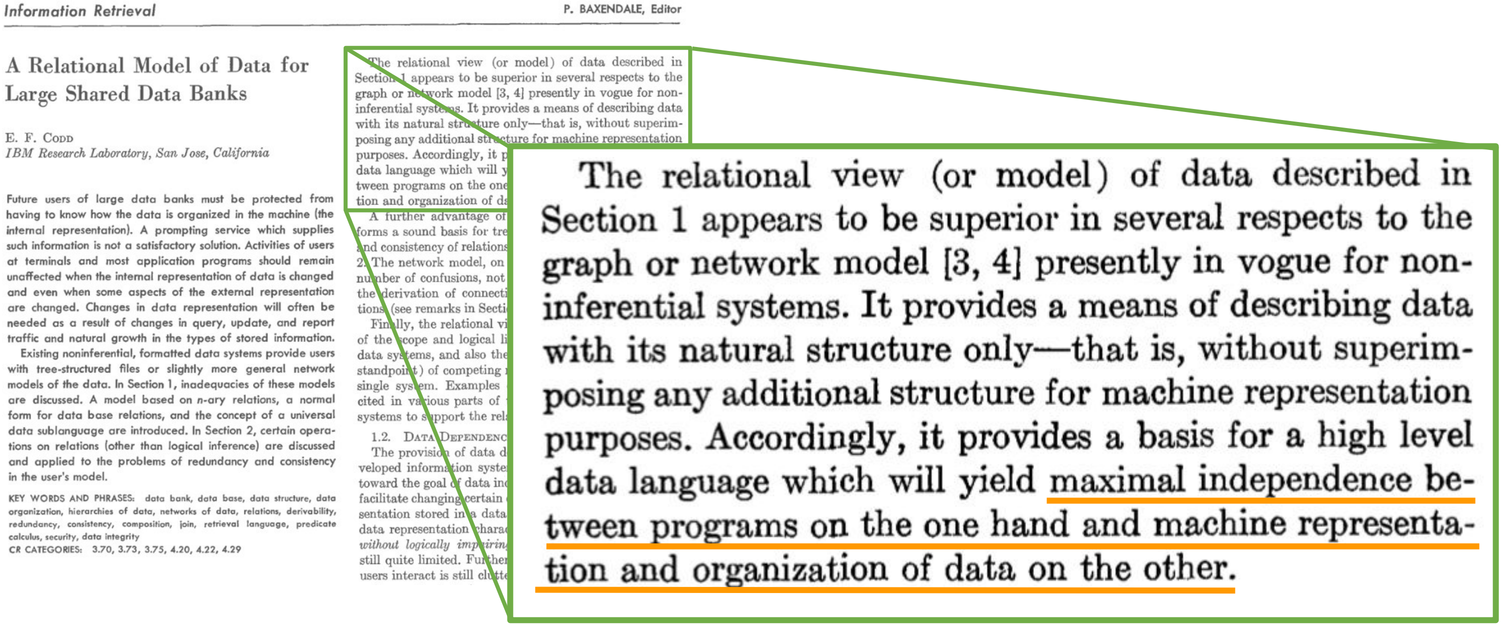
```
def C[i, j] = sum[k: A[i, k] * B[k, j]]
```

SQL

```
SELECT A.row, B.col, SUM(A.val * B.val)
FROM A INNER JOIN B ON A.col = B.row
GROUP BY A.row, B.col
```

The Essence of the Relational Model





Information Retrieval — P. BAXENDALE, Editor

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

Existing noninferential, formatted data systems provide users with tree-structured files or slightly more general network models of the data. In Section 1, inadequacies of these models are discussed. A model based on *n*-ary relations, a normal form for data base relations, and the concept of a universal data sublanguage are introduced. In Section 2, certain operations on relations (other than logical inference) are discussed and applied to the problems of redundancy and consistency in the user's model.

KEY WORDS AND PHRASES: data bank, data base, data structure, data organization, hierarchies of data, networks of data, relations, derivability, redundancy, consistency, composition, join, retrieval language, predicate calculus, security, data integrity

CR CATEGORIES: 3.70, 3.73, 3.75, 4.20, 4.22, 4.29

The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for non-inferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

Have relational database systems been sufficiently ambitious on this point

RelationalAI

Architecture

RelationalAI

Cloud Native Deployment Architecture

VSCode | RAI Console | RAI CLI | Data Apps | SQL Apps | Legend Apps

RAI SDK *(Python, Julia, JS, Go, Java, C#)*

Cloud Region

Services *(JSON, Arrow)*

aws

Engine
Transient
RAM, SSD cache

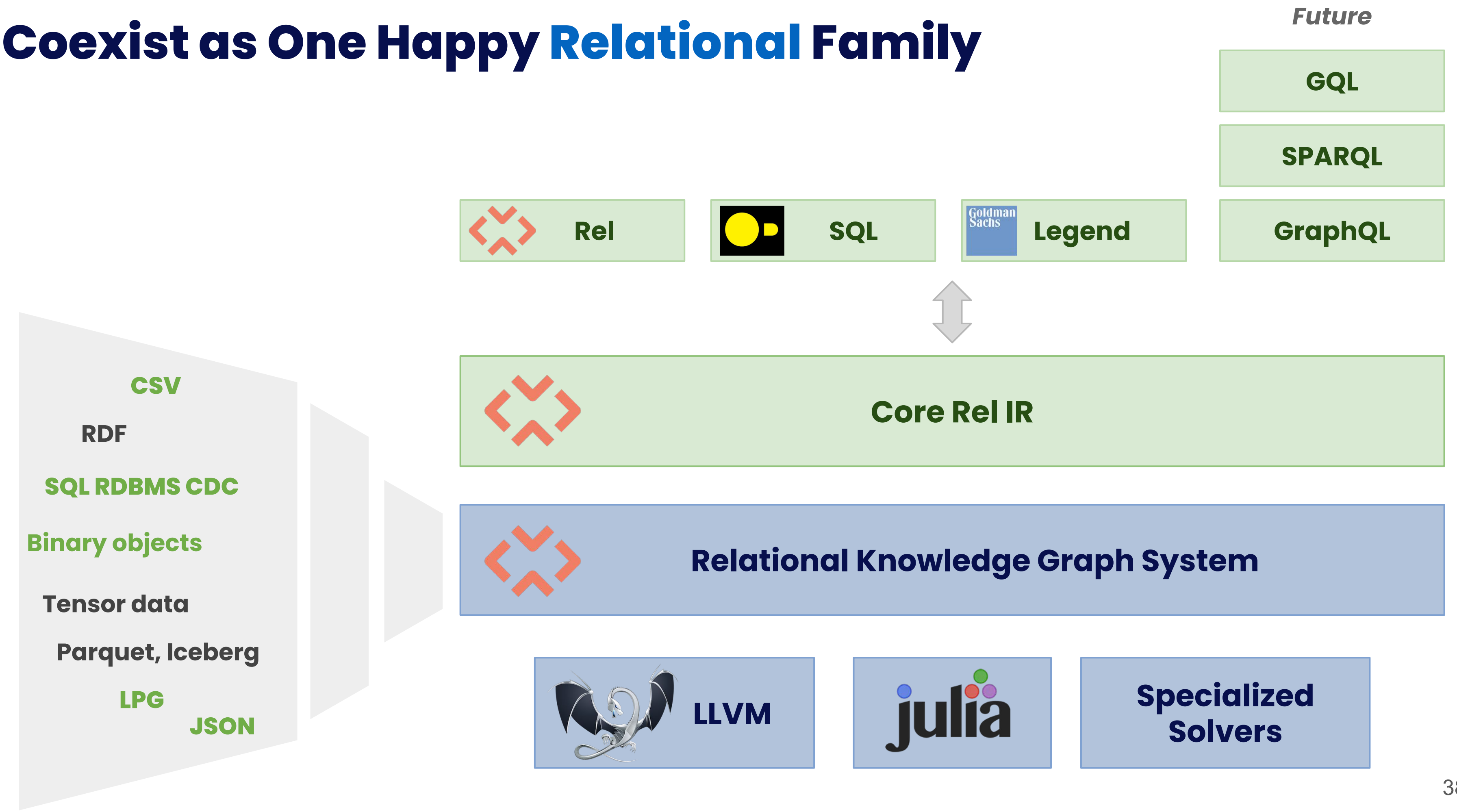
Engine
Transient
RAM, SSD cache

Serverless Engine
Transient
RAM, SSD cache

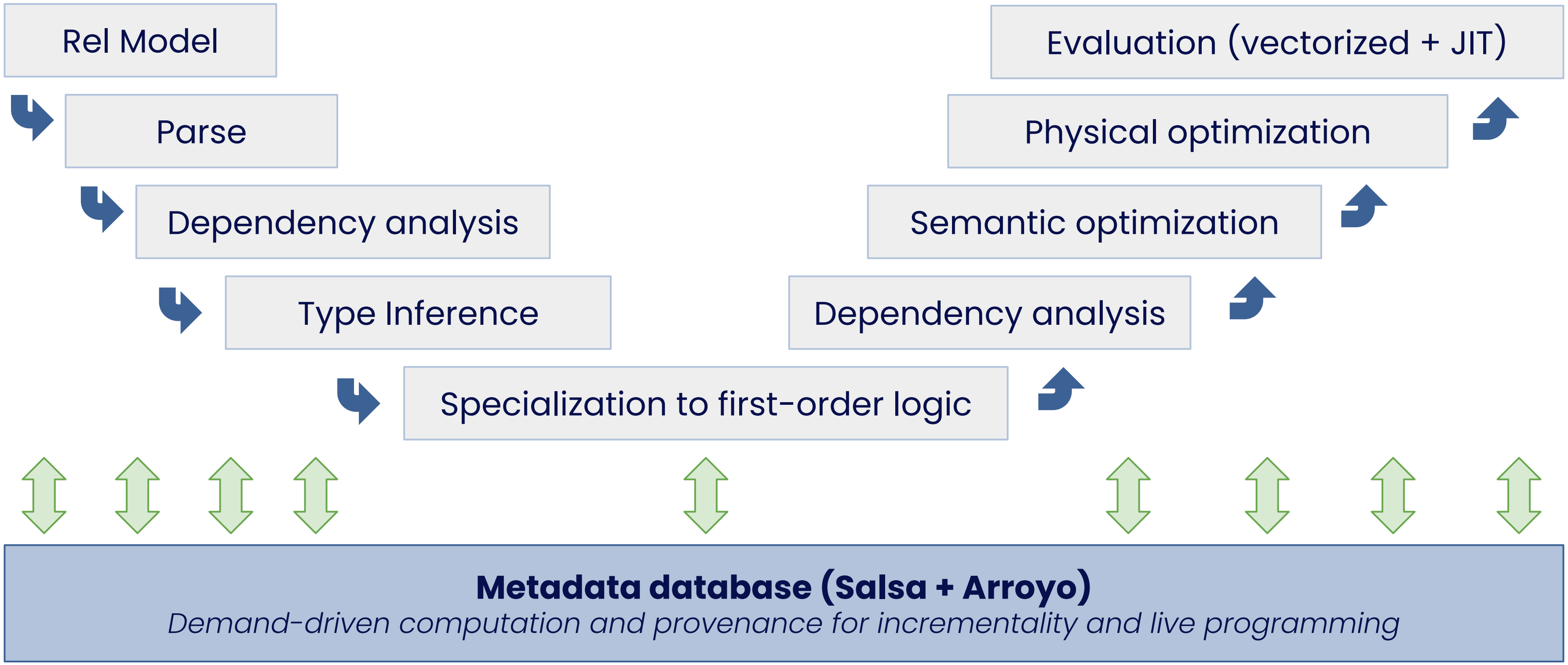
Scalable, durable object storage
Immutable, versioned, write-optimized, paged data structures

CAS Key-Value Store
(Only database root pointers)

Coexist as One Happy Relational Family



Internal Engine Architecture



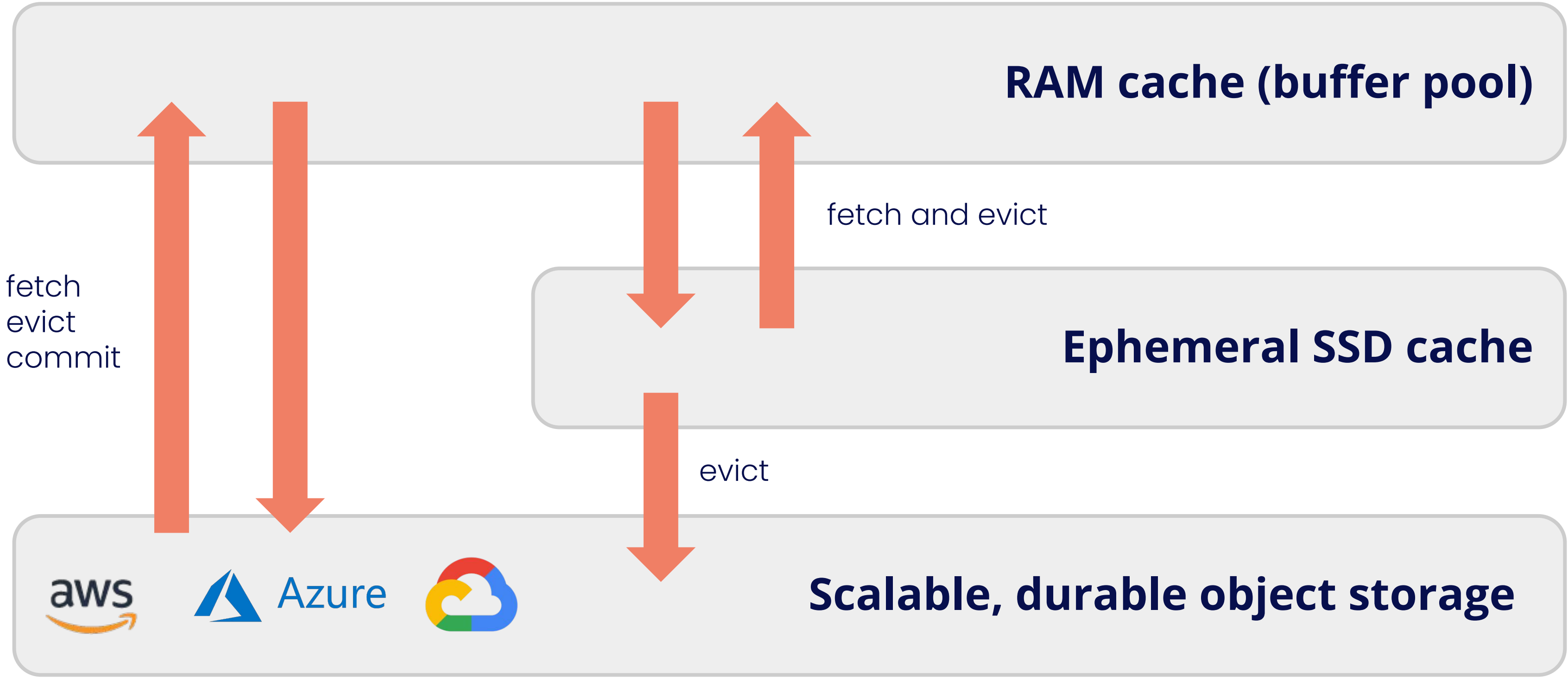
RelationalAI

Core Innovations for
Relational Knowledge Graphs

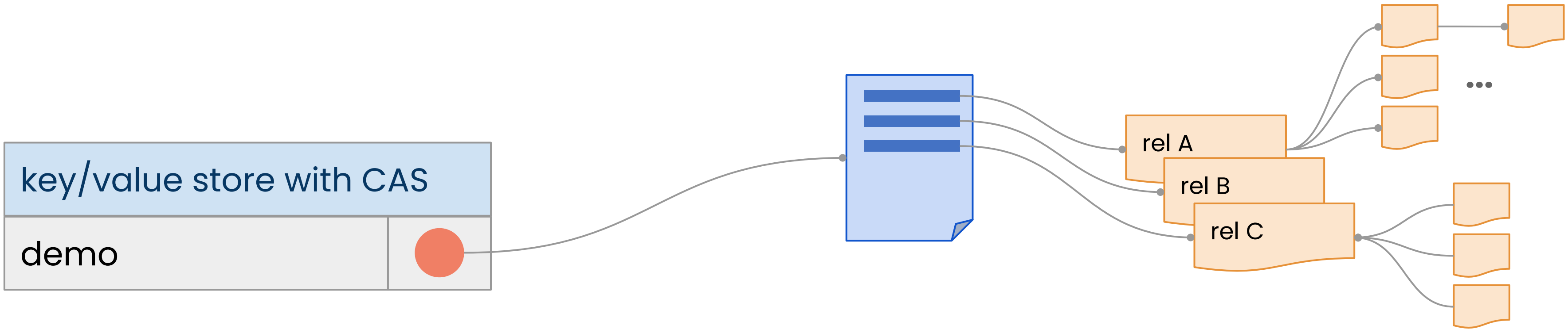
Immutable Data Structures for Cloud Object Storage

RAI Storage and Memory Management

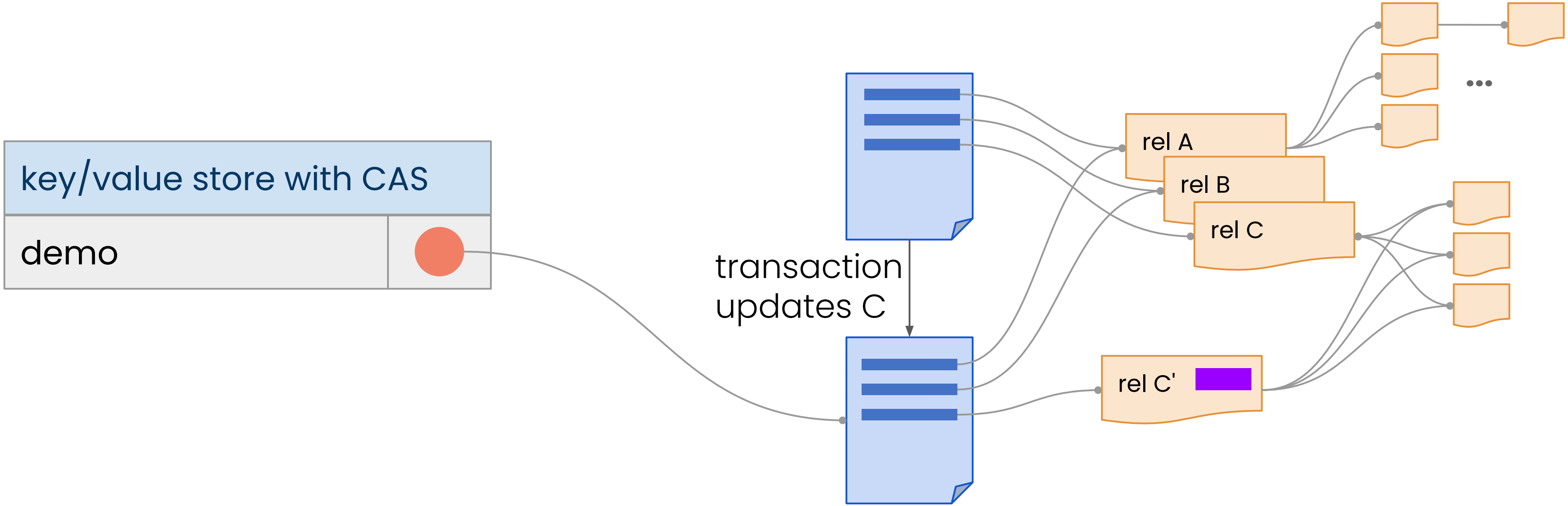
(inspired by Snowflake and Umbra/Leanstore)



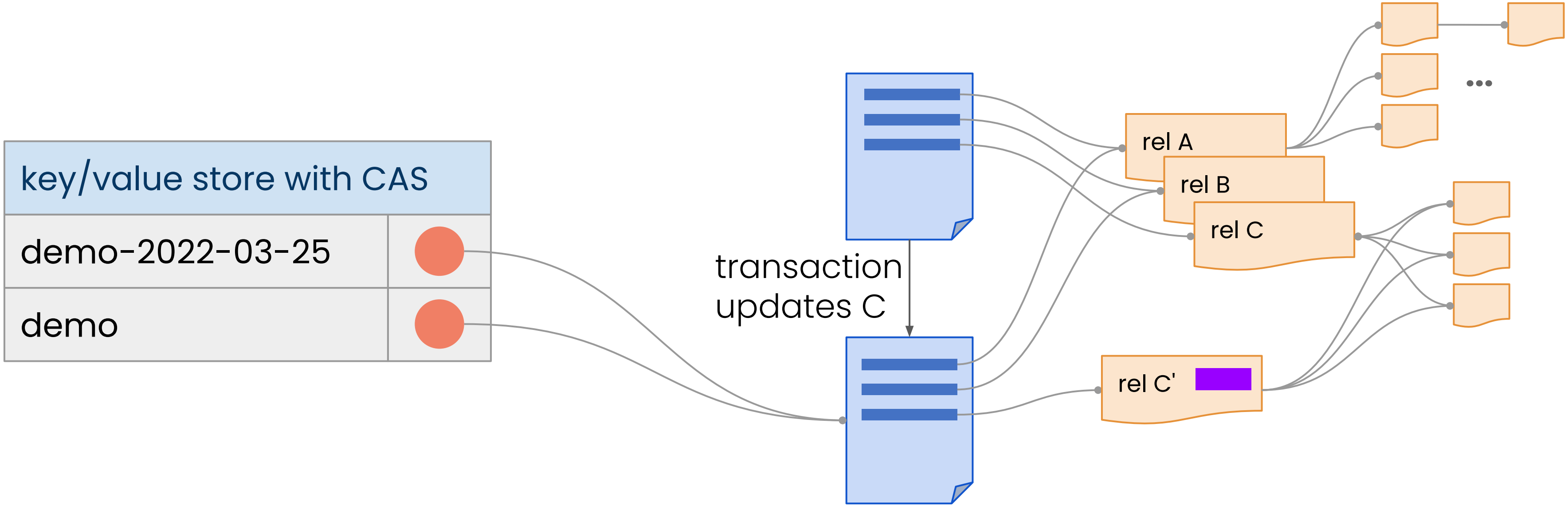
RAI databases are immutable, including the catalog



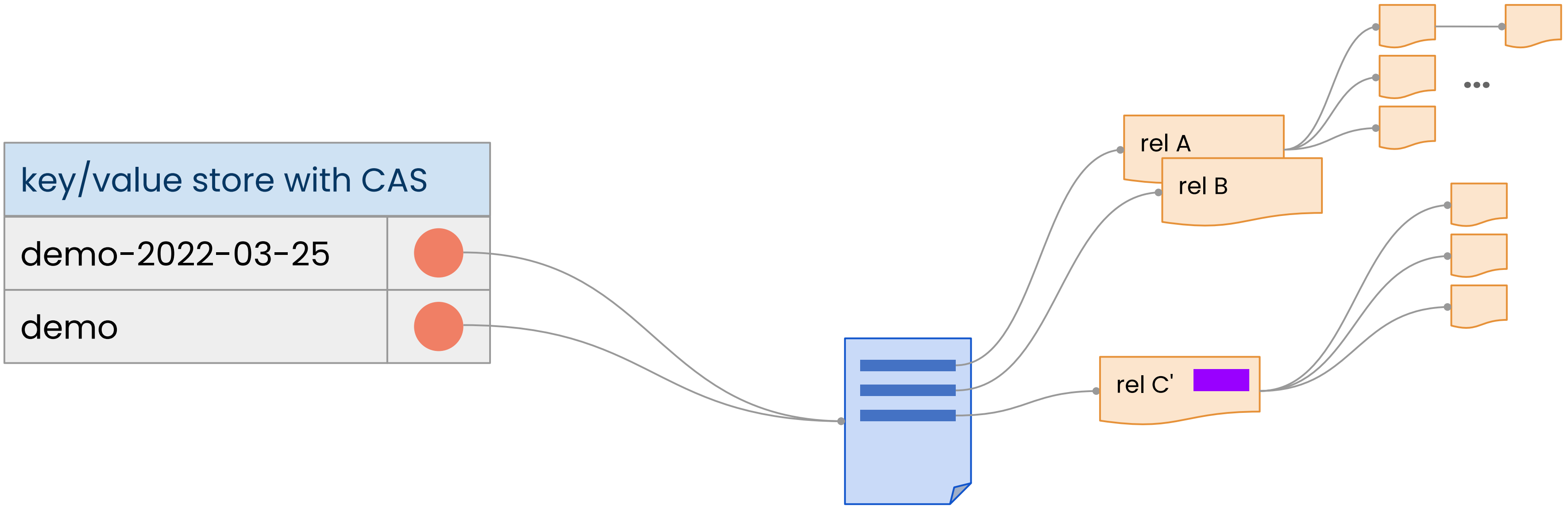
RAI databases are immutable, including the catalog



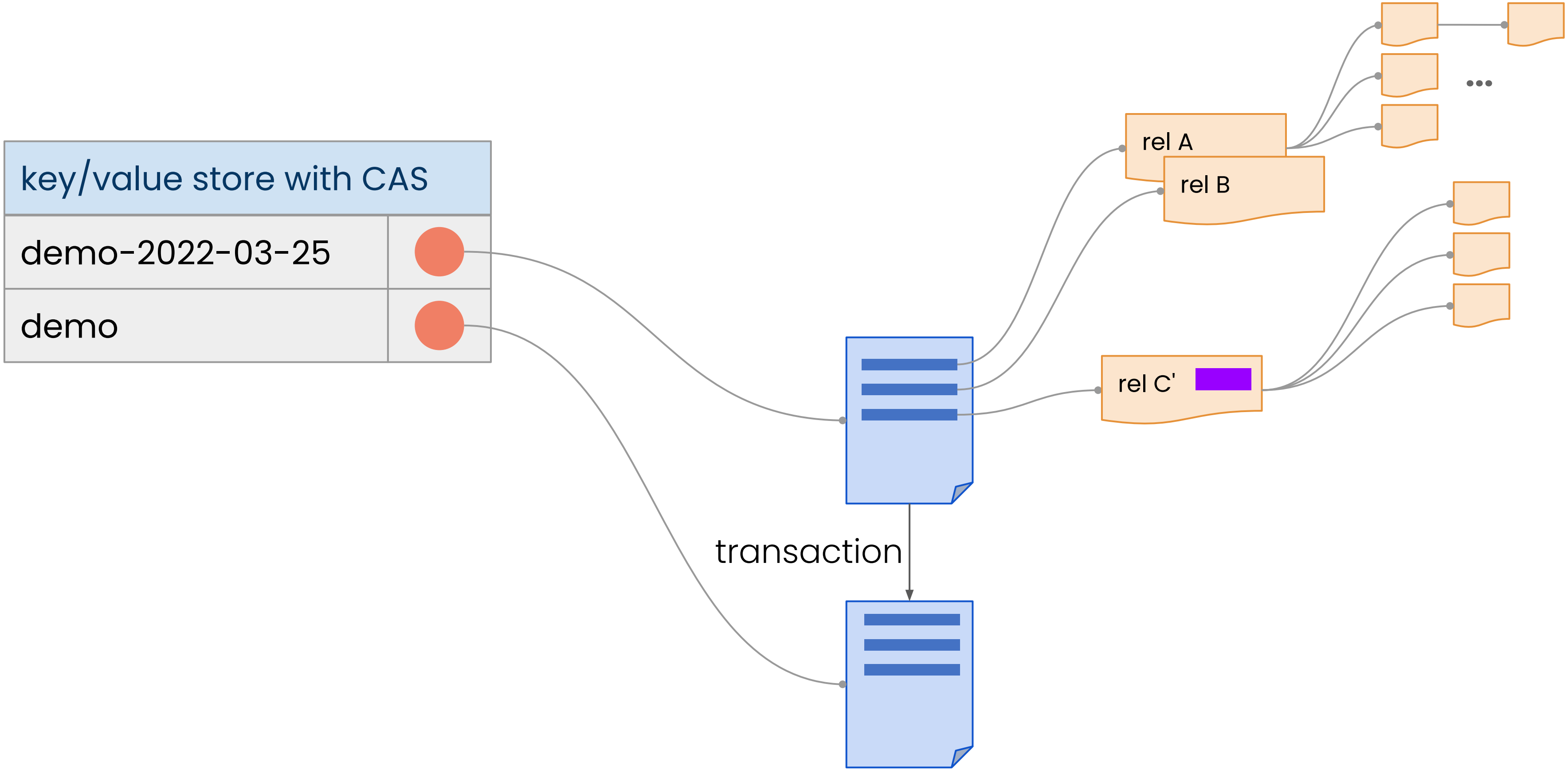
RAI databases are immutable, including the catalog



RAI databases are immutable, including the catalog



RAI databases are immutable, including the catalog



RelationalAI

Key: immutable tables ⟶ immutable catalog

Isolation: strict serializability

- **Must:** Anything weaker causes inconsistencies for data apps *(depending on lock granularity)*
- No locks need to be acquired *(concurrent writes can be executed optimistically)*
- Effectively unlimited read scalability
- No limit on the duration of a transaction

DDL is atomic

- **Must:** Schema changes are common in data apps and live programming
- Cloning a database is an atomic O(1) operation
- Perfect for as-of (system time) queries, what-if analysis

Write-optimized data structures 💕 immutable object storage

- **Must:** Removing write amplification is critical for object storage (Bε-tree)
- Group commits and variable page sizes to reduce write throughput needs

No transaction log is needed for durability or recovery

- Previous version immutable. Commits atomic in KV store (CAS)

Storage Management: Influences and Resources

Elastic Storage Management

- **The Snowflake Elastic Data Warehouse**
  Dageville et al., SIGMOD 2016
- **Building an Elastic Query Engine on Disaggregated Storage**
  Vuppalapati et al., NSDI 2020

Write Optimization

- **Lower Bounds for External Memory Dictionaries**
  Brodal et al., SODA 2003
- **An Introduction to Bε -trees and Write-Optimization**
  Bender et al., :login: magazine, 2015
- **Design and Implementation of the LogicBlox System**
  Aref et al. SIGMOD 2015

In-Memory Performance

- **LeanStore: In-Memory Data Management Beyond Main Memory**
  Leis et al., ICDE 2018
- **Umbra: A Disk-Based System with In-Memory Performance**
  Neumann et al., CIDR 2020

RelationalAI

Core Innovations for
Relational Knowledge Graphs

Rel

A Productive and Expressive Relational Language

Datalog and First-order Logic

Transitive closure

```
ancestor(x, y) :- parent(x, y)
ancestor(x, y) :- parent(x, t) and ancestor(t, y)

reachable(x, y) :- edge(x, y)
reachable(x, y) :- edge(x, t) and reachable(t, y)
```

Functional dependency

```
function_age()     :- forall(x, v, w: age(x, v)     and age(x, w)     implies v = w)
function_name()    :- forall(x, v, w: name(x, v)    and name(x, w)    implies v = w)
function_address() :- forall(x, v, w: address(x, v) and address(x, w) implies v = w)
```

Average

```
average_sales(x, y, w)   :- sum_sales(x, y, u)   and count_sales(x, y, v)   and w = u / v
average_returns(x, y, w) :- sum_returns(x, y, u) and count_returns(x, y, v) and w = u / v
```

Datalog

Good

- Outstanding formal foundation
- Mutually recursive definitions

More is needed

- Classic Datalog (globally stratified) is too limited for graph workloads:
    - Value creation in recursion
    - Aggregation in recursion
    - Negation in recursion
- Datalog does not support abstraction (similar to SQL, Cypher, SPARQL etc)
    - Abstract over concrete relations
    - Abstract over schema

Rel: Datalog is the IR

Rel – Design Objectives

Small core	Designed for growth: whole is greater than sum of the parts
Declarative	Maximize opportunities for executing programs in different ways
Relational	Data independence *(representation, ordering, semantic stability)*
Abstraction	Libraries of reusable functionality (eg statistics, graph analytics) Encourage an ecosystem of reusable components
Abstraction without regret	Aggressive optimizations to compile abstraction cost aways.
Schema abstraction	Logically treating schema as data to support schema-generic logic Prevent the need for code generators Support interactive schema discovery (reflection)
Live programming	Support arbitrary schema changes Ingest data without upfront schema into an efficient representation Incorrect application logic is a valid state Support gradually enforcing a schema with integrity constraints

RelationalAI

Primary key

Hours
Minutes

Period
Minutes
Hours

Miles
Kilometers

Are these exclusive

carrier	origin	destination	flight_num	flight_time	tail_num	dep_time	arr_time	dep_delay	arr_delay	taxi_out	taxi_in	distance	cancelled	diverted	id2
F9	MCI	DEN	818	89	N916FR	2005-09-26 08:23:00 UTC	2005-09-26 09:07:00 UTC	-7	-8	9	6	533	N	N	36606824
WN	LBB	ELP	819	41	N708SW	2000-08-18 07:30:00 UTC	2000-08-18 07:20:00 UTC	0	-5	7	2	295	N	N	4369021
NW	ATL	MEM	819	52	N607NW	2001-11-16 07:15:00 UTC	2001-11-16 07:29:00 UTC	-5	-9	19	3	332	N	N	11838308
DL	SLC	BOI	819	48	N296WA	2001-12-04 22:12:00 UTC	2001-12-04 23:53:00 UTC	7	[illegible]	49	4	291	N	N	12060416
WN	PHX	SAN	819	51	N391SW	2001-12-05 09:11:00 UTC	2001-12-05 09:17:00 UTC	11	[illegible]	[illegible]	[illegible]	304	N	N	12383068
WN	LAS	AUS	819	135	N519SW	2002-04-05 08:18:00 UTC	2002-04-05 12:47:00 UTC	8	[illegible]	[illegible]	[illegible]	1085	N	N	13763279
WN	SJC	LAS	819	63	N528SW	2002-07-14 06:30:00 UTC	2002-07-14 07:47:00 UTC	0	-3	11	3	386	N	N	15284777
WN	LAS	AUS	819	137	N502SW	2002-09-16 08:20:00 UTC	2002-09-16 12:52:00 UTC	0	-8	11	4	1085	N	N	16027516
DL	MSP	SLC	819	149	N3754A	2002-09-29 19:34:00 UTC	2002-09-29 21:35:00 UTC	9	15	25	7	991	N	N	16399211
DL	MSP	SLC	819	145	N3745B	2002-12-06 19:27:00 UTC	2002-12-06 21:16:00 UTC	-3	-9	18	6	991	N	N	17417961
WN	SJC	LAS	819	54	N730SW	2003-06-26 06:30:00 UTC	2003-06-26 07:44:00 UTC	0	-11	15	5	386	N	N	20460576
WN	SJC	LAS	819	60	N501SW	2003-09-15 06:30:00 UTC	2003-09-15 07:50:00 UTC	0	0	18	2	386	N	N	22113592
NW	ATL	MEM	819	51	N785NC	2003-11-20 07:11:00 UTC	2003-11-20 07:15:00 UTC	-9	-23	10	3	332	N	N	23383534
DL	MSP	ATL	819	120	N906DE	2004-02-10 09:59:00 UTC	2004-02-10 13:36:00 UTC	-6	0	30	7	906	N	N	25206983
US	CHS	CLT	820	38	N592US	2002-07-01 19:32:00 UTC	2002-07-01 20:54:00 UTC	37	64	9	35	168	N	N	15142411
FL	TPA	ATL	820	64	N955AT	2003-01-31 11:29:00 UTC	2003-01-31 12:55:00 UTC	-6	-5	13	9	406	N	N	17949329
WN	RDU	PHL	820	73	N382SW	2004-12-02 11:10:00 UTC	2004-12-02 12:31:00 UTC	0	-19	5	3	336	N	N	30796766
HP	PHX	BOS	820	0	N826AW	2005-10-25 00:00:00 UTC	2005-10-25 00:00:00 UTC	0	0	0	0	2300	Y	N	37174931
US	PBI	CLT	821	90	N624AU	2002-05-18 06:35:00 UTC	[illegible]	-5	-8	10	6	590	N	N	14247814
US	PBI	CLT	821	87	[illegible]	[illegible]	[illegible] 07:43:00 UTC	-7	-7	16	7	590	N	N	20289351
DL	GSO	CVG	821	70	[illegible]	[illegible]	[illegible] 08:12:00 UTC	1	11	14	7	330	N	N	21396171

Not a delay

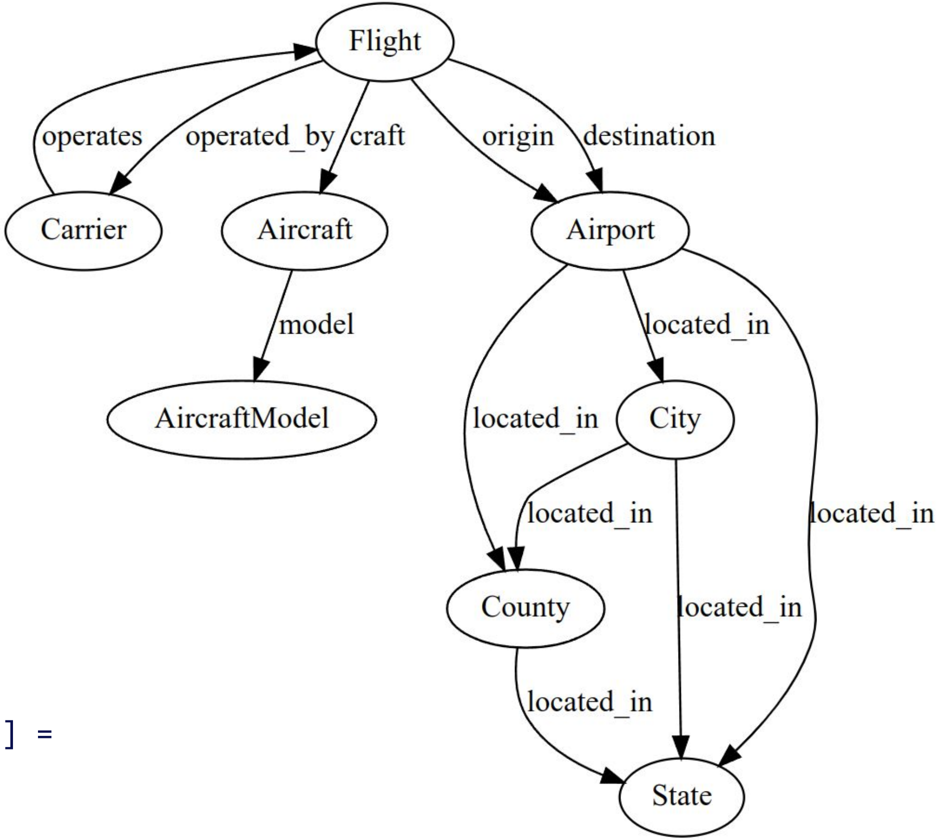
Include helicopters

Risk of messing up aggregates

Possible values

Better Conceptual Model

```
def Heliport(x in Airport) =
    fac_type(x, "HELIPORT")

def cancelled(f in Flight) =
    flight(f) and flight_cancelled(f, "Y")

def origin(f in Flight, a in Airport) =
    flight_origin(f, code) and
    airport_code(a, code)
    from code

def destination(f in Flight, a in Airport) =
    flight_destination(f, code) and
    airport_code(a, code)
    from code

def airport_distance[a1 in Airport, a2 in Airport] =
    distance[coordinate[a1], coordinate[a2]]

def located_in(x, y) =
    exists(t: located_in(x, t) and located_in(t, y))
```



Data Integrity

RelationalAI

Nodes involved in relationships

```
ic forall(f, ap: origin(f, ap) implies Flight(f) and Airport(ap))
```

Required relationships

```
ic forall(f: Flight(f) implies exists origin[f])
```

Functional dependency (flight can have only one origin)

```
ic forall(x, v, w: origin(x, v) and origin(x, w) implies v = w)
```

Arbitrarily complex

```
ic forall(f in cancelled: not exists flight_duration[f])
ic forall(f in flight: cancelled(f) xor diverted(f) xor arrived(f))
```

Aggregation

Total number of flights

```
count[Flight]
```

```
37,561,525
```

Carrier with most flights

```
c: count[f: operated_by(f, c)]
```

```
Southwest          5,775,777
Delta              4,477,929
American           4,434,727
```

Carriers mean arrival delay

```
c: mean[f.arrival_delay for f where operated_by(f, c)]
```

```
Airtran            15 min
Atlantic Coast     13 min
United Airlines    13 min
...
Aloha Airlines      6 min
Hawaiian Airlines   3 min
```

Airport ratio of cancelled arriving flights

```
ap: ratio[cancelled, ap.arriving_flight]
```

```
Unalaska            19%
Worcester Regional  11%
Nantucket Memorial   9%
```

Abstraction and Value Types

Recall from the model

```
def airport_distance[ap1 in Airport, ap2 in Airport] =
    distance[coordinate[ap1], coordinate[ap2]]

def coordinate[a in Airport] =
    ^LLA[latitude[a], longitude[a], elevation[a]]

def arrival_delay[f in Flight] =
    ^Minute[maximum[0, arr_delay[f]]
```

Units of measurements to prevent miscalculation

```
def LengthUnit = :Feet; :Meters; :Miles; :Kilometers

value type Length = LengthUnit, Number
value type Degree = Number
value type LLA = Degree, Degree, Length

def distance[x in LLA, y in LLA] =
    haversine[earth_radius, x, y]

def earth_radius = ^Length[:Kilometers, 6378.1]
```

The type system of Rel prevents a runtime cost of tracking units of measurement.

Statically Rel guarantees that the correct conversions are applied and no incompatible values can be used in operations.

Schema Abstraction

Rel is not a dynamic language (nor a triple store). Rel exposes the schema logically as data and uses partial evaluation methods to infer and specialize the program to the schema.

Count all nodes

```
count[x, v: flight_graph(x, v)]
```

```
38,061,144
```

Count all nodes, grouped by type

```
x: count[v: flight_graph(x, v)]
```

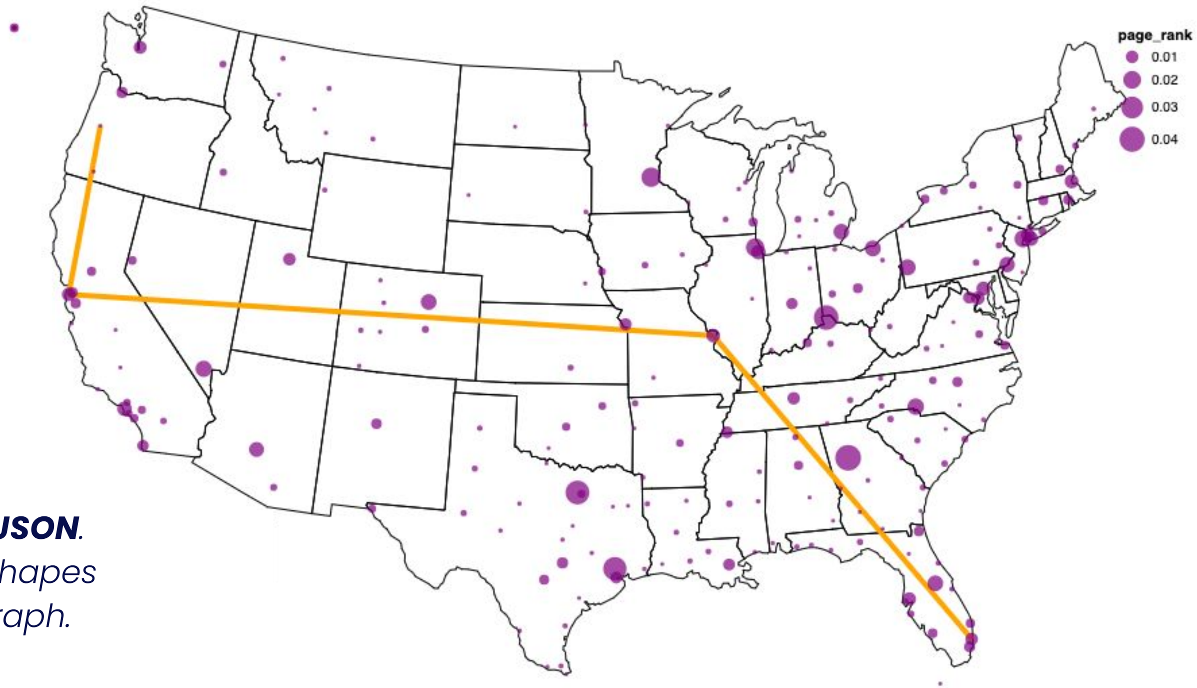
Flight	37,561,525
Aircraft	359,928
AircraftModel	60,461
City	50,944
Airport	19,793
Heliport	5,135
County	3,009
Major	270
State	58
Carrier	21

Graph Analytics

Rel can express graph algorithms, for example **pagerank** and **shortest path**.

Shown: pagerank for major airports

Highlighted is a shortest path between two nodes.



*Rel supports **geographical data** and **JSON**. The maps are computed in Rel from shapes of the states, part of the knowledge graph. Visualization is Vega-Lite.*

RelationalAI

Basic graph algorithms

Neighbor (undirected edge)

```
def neighbor(x, y) = edge(x, y) or edge(y, x)
def cn[x, y] = count[intersect[neighbor[x], neighbor[y]]]
```

Degree

```
def outdegree[x] = count[edge[x]]
def degree[x] = count[neighbor[x]]
```

Similarity

```
def cosine_sim[x, y]  = cn[x, y] / sqrt[degree[x] * degree[y]]
def jaccard_sim[x, y] = cn[x, y] / count[neighbor[x]] + count[neighbor[y]] - cn[x, y]
```

Transitive closure (reachability)

```
def reachable(x, y) = edge(x, y)
def reachable(x, y) = exists(t: edge(x, t) and reachable(t, y))
```

Basic graph algorithms

> The purpose of the semantic optimizer of RelationalAI is to automate this optimization by using the algebraic properties of minimum.

Weakly connected components

```
def wcc[x] = min[reachable_undirected[x]]
```

Weakly connected components (without reachable)

```
def wcc[x] = minimum[ min[neighbor[x]], min[wcc[z] for z in neighbor[x] ]
```

Strongly connected components

```
def scc[x] = min[v: reachable(x, v) and reachable(v, x)]
```

Basic graph algorithms

Breadth-first search

```
def bfs[x in root] = 0
def bfs[x] = min[ bfs[x];  bfs[y: edge(y, x)] + 1  ]
```

Shortest Distance

The purpose of the semantic optimizer of RelationalAI is to automate this optimization by using the algebraic properties of minimum and addition.

Shortest distance between two nodes

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

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

Shortest distance between two nodes (Bellman-Ford)

```
def shortest_distance[x, y] =
    min[ distance[x, y];
         (shortest_distance[x, t] + distance[t, y] from t)]
```

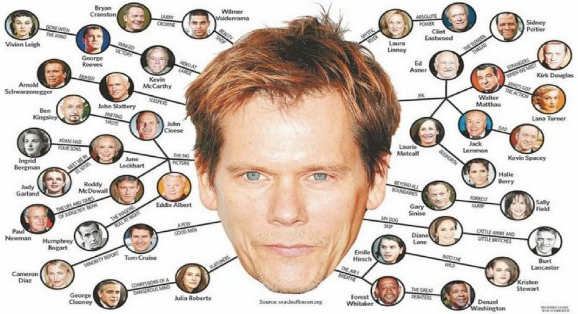
Semantic Optimizer: Push Demand into Recursion

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

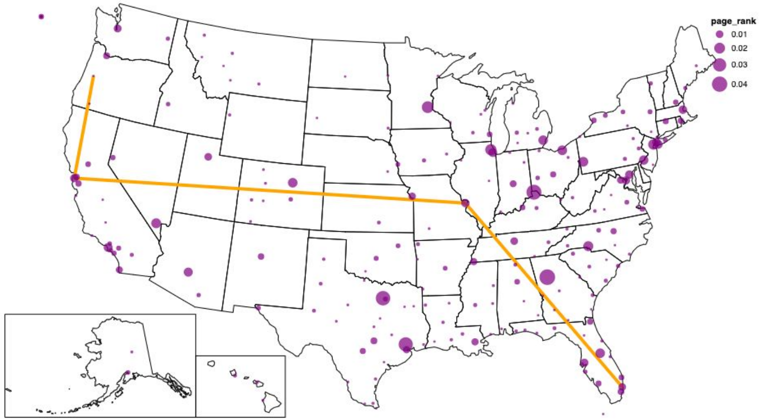
```
def bacon_number[p] =
    shortest_distance[(co_star, 1)[KevinBacon, p]
```

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)
    ]
```



Pagerank



Non-monotonic, relying on reaching a fixpoint

```
def damping = 0.85
def pagerank[x in node] = 1.0, not(pagerank(x, _))
def pagerank[y in node] =
    (1.0 - damping) +
    damping * sum[pagerank[x] / outdegree[x] for x where edge(x, y)]
```

Iterative

```
def damping = 0.85
def pagerank[x in node, 0] = 1.0
def pagerank[y in node, i in range[0, 20, 1]] =
    (1.0 - damping) +
    damping * sum[pagerank[x, i - 1] / outdegree[y] for x where edge(x, y)]
```

TigerGraph Graph Data Science Library

Pagerank

```
HeapAccum<Vertex_Score>(top_k, score DESC) @@top_scores_heap;
MaxAccum<FLOAT> @@max_diff = 9999;
SumAccum<FLOAT> @sum_recvd_score = 0;
SumAccum<FLOAT> @sum_score = 1;
SetAccum<EDGE> @@edge_set;

Start = {v_type};
WHILE @@max_diff > max_change
   LIMIT max_iter DO
       @@max_diff = 0;
   V = SELECT s
        FROM Start:s -(e_type:e)- v_type:t
        ACCUM
           t.@sum_recvd_score += s.@sum_score/(s.outdegree(e_type))
        POST-ACCUM
           s.@sum_score = (1.0-damping) + damping * s.@sum_recvd_score,
            s.@sum_recvd_score = 0,
            @@max_diff += abs(s.@sum_score - s.@sum_score');
END; # END WHILE loop
```

WCC

```
MinAccum<INT> @min_cc_id = 0;
MapAccum<INT, INT> @@comp_sizes_map;
MapAccum<INT, ListAccum<INT>> @@comp_group_by_size_map;

Start = {v_type};

S = SELECT x
   FROM Start:x
   POST-ACCUM x.@min_cc_id = getvid(x);

WHILE (S.size()>0) DO
   S = SELECT t
       FROM S:s -(e_type:e)- v_type:t
        ACCUM t.@min_cc_id += s.@min_cc_id
        HAVING t.@min_cc_id != t.@min_cc_id';
END;
```

https://github.com/tigergraph/gsql-graph-algorithms

Recursion: Program Analysis (Doop)

```
def VarPointsTo(var, heap) =
    AssignHeapAllocation(var, heap)


def VarPointsTo(to, heap) =
    Assign(from, to) and
    VarPointsTo(from, heap)


def VarPointsTo(to, heap) =
    LoadInstanceField(base, signature, to) and
    VarPointsTo(base, baseheap) and
    InstanceFieldPointsTo(baseheap, signature, heap)


def InstanceFieldPointsTo(baseheap, signature, heap) =
    StoreInstanceField(from, base, signature) and
    VarPointsTo(base, baseheap) and
    VarPointsTo(from, heap)
```

Strictly Declarative Specification of Sophisticated Points-to Analyses

Martin Bravenboer   Yannis Smaragdakis
Department of Computer Science
University of Massachusetts, Amherst
Amherst, MA 01003, USA
martin.bravenboer@acm.org   yannis@cs.umass.edu

Abstract

We present the Doop framework for points-to analysis of Java programs. Doop builds on the idea of specifying pointer analysis algorithms declaratively, using Datalog: a logic-based language for defining (recursive) relations. We carry the declarative approach further than past work by describing the full end-to-end analysis in Datalog and optimizing aggressively using a novel technique specifically targeting highly recursive Datalog programs.

As a result, Doop achieves several benefits, including full order-of-magnitude improvements in runtime. We compare Doop with Lhoták and Hendren's Paddle, which defines the state of the art for context-sensitive analyses. For the exact same logical points-to definitions (and, consequently, identical precision) Doop is more than 15x faster than Paddle for a 1-call-site sensitive analysis of the DaCapo benchmarks, with lower but still substantial speedups for other important analyses. Additionally, Doop scales to very precise analyses that are impossible with Paddle and Whaley et al.'s bddbddb, directly addressing open problems in past literature. Finally, our implementation is modular and can be easily configured to analyses with a wide range of characteristics, largely due to its declarativeness.

Categories and Subject Descriptors F.3.2 [*Logics and Meanings of Programs*]: Semantics of Programming Languages—Program Analysis; D.1.6 [*Programming Techniques*]: Logic Programming

General Terms Algorithms, Languages, Performance

1. Introduction

Points-to (or *pointer*) analysis intends to answer the question "what objects can a program variable point to?" This question forms the basis for practically all higher-level program analyses. It is, thus, not surprising that a wealth of research has been devoted to efficient and precise pointer analysis techniques. *Context-sensitive* analyses are the most common class of precise points-to analyses. Context sensitive analysis approaches qualify the analysis facts with a *context* abstraction, which captures a static notion of the dynamic context of a method. Typical contexts include abstractions of method call-sites (for a *call-site sensitive* analysis—the traditional meaning of "context-sensitive") or receiver objects (for an *object-sensitive* analysis).

In this work we present Doop: a general and versatile points-to analysis framework that makes feasible the most precise context-sensitive analyses reported in the literature. Doop implements a range of algorithms, including context insensitive, call-site sensitive, and object-sensitive analyses, all specified modularly as variations on a common code base. Compared to the prior state of the art, Doop often achieves speedups of an order-of-magnitude for several important analyses.

The main elements of our approach are the use of the Datalog language for specifying the program analyses, and the aggressive optimization of the Datalog program. The use of Datalog for program analysis (both low-level [13,23,29] and high-level [6,9]) is far from new. Our novel optimization approach, however, accounts for several orders of magnitude of performance improvement: unoptimized analyses typically run over 1000 times more slowly. Generally our optimizations fit well the approach of handling program facts as a database, by specifically targeting the indexing scheme and the incremental evaluation of Datalog implementations. Furthermore, our approach is entirely Datalog based, encoding declaratively the logic required both for call graph construction as well as for handling the full semantic complexity of the Java language (e.g., static initialization, finalization, reference objects, threads, exceptions, reflection, etc.). This makes our pointer analysis specifications elegant, modular, but also efficient and easy to tune. Generally, our work is a strong data point in support of declarative languages: we argue that prohibitively much human effort is required for implementing and optimizing complex mutually-recursive definitions at an operational level of abstraction. On the other

OOPSLA 2009, October 25–29, 2009, Orlando, Florida, USA.

Syntactic Second-order Features

Transitive closure (reachability)

```
def ancestor(x, y) = parent(x, y)
def ancestor(x, y) = exists(t: parent(x, t) and ancestor(t, y))
```

Abstract

```
def tc[E](x, y) = E(x, y)
def tc[E](x, y) = exists(t: E(x, t) and tc[E](t, y))
```

Use

```
def ancestor = tc[parent]
```

Syntactic Second-order Features

Mean (average)

```
sum[sales] / count[sales]
```

Abstract

```
def mean[F] = sum[F] / count[F]
```

Use

```
mean[sales]
```

Syntactic Second-order Features

Functional dependency

```
forall(x, v, w: origin(x, v) and origin(x, w) implies v = w)
```

Abstract

```
def function(R) =
    forall(k..., v1, v2 where R(k..., v1) and R(k..., v2): v1 = v2)
```

Use

```
function(origin)
```

Library Example: Graph Analytics

```
module graph_analytics[G]
  with G use node, edge

  def neighbor(x, y) = edge(x, y) or edge(y, x)
  def outdegree[x] = count[edge[x]]
  def degree[x] = count[neighbor[x]]
  def cn[x, y] = count[intersect[neighbor[x], neighbor[y]]]

  def reachable = edge; reachable.edge
  def reachable_undirected = neighbor; reachable_undirected.neighbor

  def scc[x] = min[v: reachable(x, v) and reachable(v, x)]
  def wcc[x] = min[reachable_undirected[x]]

  def cosine_sim[x, y]  = cn[x, y] / sqrt[degree[x] * degree[y]]
  def jaccard_sim[x, y] = cn[x, y] / count[neighbor[x]] + count[neighbor[y]] - cn[x, y]
  ...
end
```

Library Example: Relational Algebra to Calculus

```
def intersect[R, S](x...) = R(x...) and S(x...)
def union[R, S](x...)     = R(x...) or S(x...)
def diff[R, S](x...)      = R(x...) and not S(x...)

def subset[R, S]   = forall(x... where R(x...): S(x...))
def disjoint(R, S) = empty(R ∩ S)
def empty(R)       = not exists(x...: R(x...))

def (∩) = intersect
def (∪) = union
def (×) = cart
def (⊂) = proper_subset
def (⊆) = subset
```

Library Example: Statistics

RelationalAI features a large library of reusable functionality implemented in Rel.

```
def mean[F] = sum[F] / count[F]

def frequency[R, elem] = count[x...: R(x..., elem)]

def mse[Yhat, Y] = sum[x: (Y[x] - Yhat[x]) ^ 2] / count[Y]

def rmse[Yhat, Y] = sqrt[mse[Yhat, Y]]
```

$$MSE = \frac{1}{n} \Sigma \left(y - \widehat{y} \right)^2$$

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

Library Example: Machine Learning

Generic abstractions for feature scaling

```
def mean_normalization[F][x...] =
    (F[x...] - mean[F]) / (max[F] - min[F]), (max[F] > min[F])

def min_max_normalization[F][x...] =
    (F[x...] - min[F]) / (max[F] - min[F]), (max[F] > min[F])

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

```
{%- if include_columns=='*' -%}
{%- set all_source_columns = adapter.get_columns_in_relation(source_table) | map(attribute='quoted') -%}
{% set include_columns = all_source_columns %}
{%- endif -%}

-- generate a CTE for each source column, a single row containing the aggregates
with
{% for source_column in source_columns %}
    {{ source_column }}_aggregates as (
        select
            min({{ source_column }}) as min_value,
            max({{ source_column }}) as max_value
        from {{ source_table }}
    )
{% if not loop.last %}, {% endif %}
{% endfor %}

select
    {% for column in include_columns %}
        source_table.{{ column }},
    {% endfor %}
    {% for source_column in source_columns %}
        ({{ source_column }} - {{ source_column }}_aggregates.min_value)
            / ({{ source_column }}_aggregates.max_value - {{ source_column }}_aggregates.min_value) as {{ source_column }}_scaled
    {% if not loop.last %}, {% endif %}
    {% endfor %}
from
    {% for source_column in source_columns %}
        {{ source_column }}_aggregates,
    {% endfor %}
    {{ source_table }} as source_table
```

Library Example: Machine Learning

The (simplified) linear prediction function uses schema abstraction (f) to compute a prediction for a module of features (Feature).

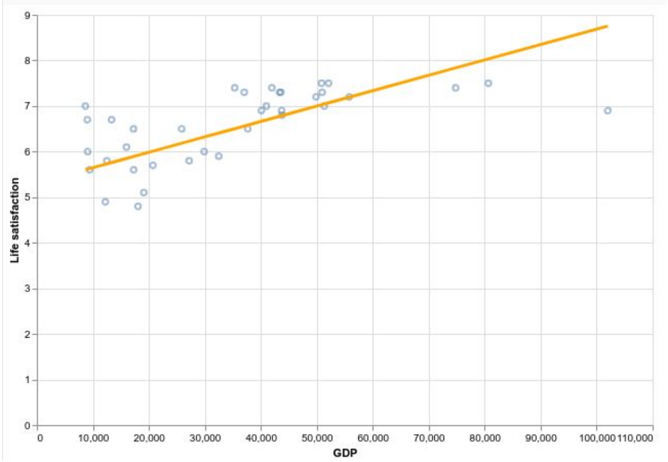
```
def linear_predict[Feature, Weight][x...] =
    sum[f: Weight[f] * Feature[f, x...]] +
    sum[f: Weight[f, Feature[f, x...]]] +
    Weight[:bias]


def linear_regression[Feature, Response, Weight] =
    minimize[rmse[linear_predict[Feature, Weight], Response]]
```

Rel => Core Rel generates a sum of the features (which typically have a specific schema).

Example: Gradient Descent

Simplified batch gradient descent:

```
def max_k = 200
def alpha = 0.01

def predict[i] = linear_predict[features, weight[i]]
def predict_error[i] = rmse[response, predict[i]]
def gradient = jacobian[predict_error, weight]

def weight[i, f] =
    weight[i - 1, f] - alpha * gradient[i - 1, i - 1, f],
    i < max_k
```



Instantiation:

```
def features:gdp_per_capita = min_max_normalization[gdp_per_capita]
def response = life_satisfaction
```

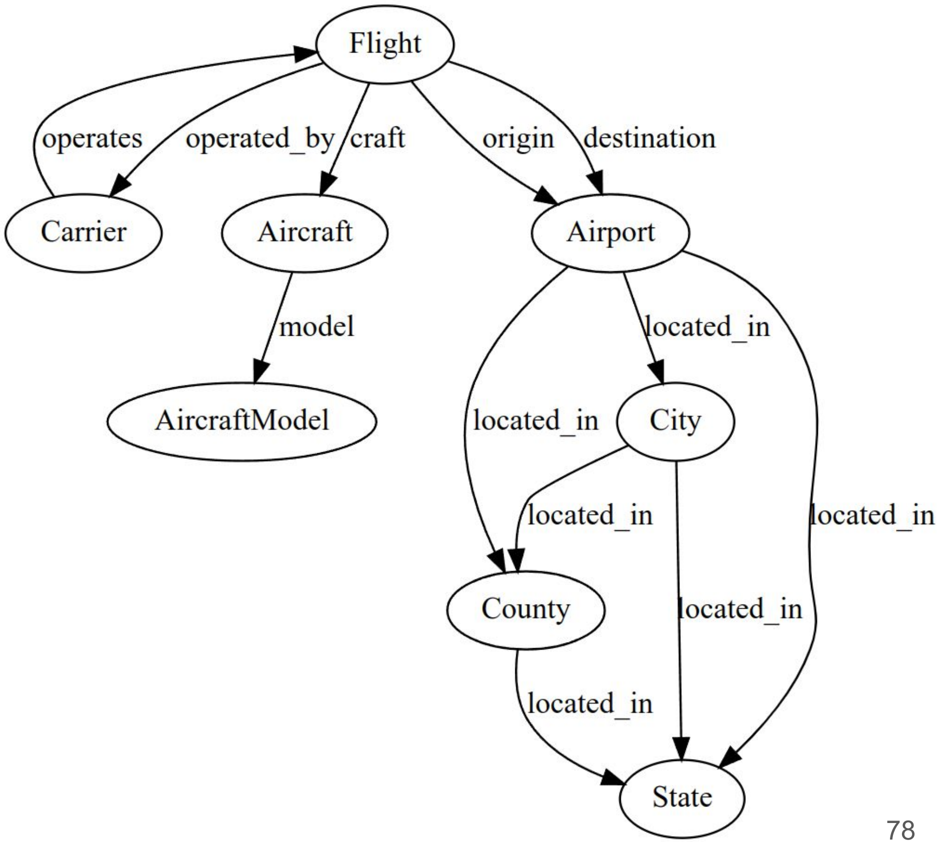
(This is for illustration purposes: linear regression does not normally use gradient descent)

Schema Abstraction

Query the schema and visualize with graphviz

```
module schema_graph[G]
    def node(x) = G(x, _)
    def edge(e, tx, ty) =
        G(e, x, y) and
        G(tx, x) and
        G(ty, y) and
        Entity(x) and
        Entity(y)
        from x, y
end

def output = graphviz[schema_graph[flight_graph]]
```

Schema = data: library applies to both



Schema Abstraction

Schema: shortest path from Flight to State

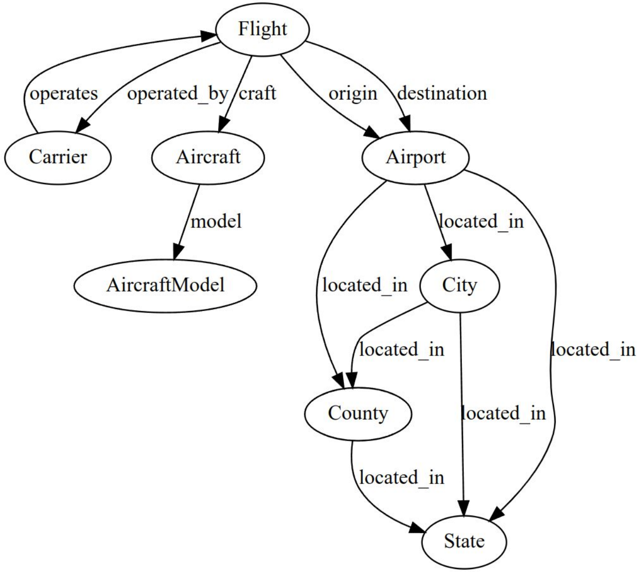
```
shortest_path[schema_graph[flight_graph], :Flight, :State]
```

```
Flight -> destination -> Airport -> located_in -> State
Flight -> origin      -> Airport -> located_in -> State
```

Schema: all acyclic paths from Flight to State

```
acyclic_path[schema_graph[flight_graph], :Flight, :State]
```

```
Flight -> destination -> Airport -> located_in -> City -> located_in -> County -> located_in -> State
Flight -> destination -> Airport -> located_in -> City -> located_in -> State
Flight -> destination -> Airport -> located_in -> County -> located_in -> State
Flight -> destination -> Airport -> located_in -> State
...
```



Note: The path algorithms are written in Rel (not foreign functions)

Feature Engineering: Describe

Similar to Dataframes, **`describe`**, **implemented in Rel**, generically reports statistics for a collection of relations.

```
describe[airport]
```

```
            Elevation  State    Facility     ....
---------------------------------------
min              -210     AK    AIRPORT
max            12,442     WY    ULTRALIGHT
mean            1,143
std             1,444
25%               270
50%               745
75%             1,220
unique                    58    7
mode                      TX    AIRPORT
```

(Furnace Creek, CA)
(Berthoud Pass, CO)

```
describe[t: ActualAirport <: airport[t]]
```

```
            Elevation  ...
------------------
max          9,927
```

(Lake County, CO)

Describe Implementation in Rel

```
def describe[R][column] = describe_full[R[column]]

def describe_full[R, :count] = count[R]
def describe_full[R, :min]   = min[R]
def describe_full[R, :max]   = max[R]

def describe_full[R, :unique]    = count[last[R :> (x: not Number(x))]]
def describe_full[R, :mode]      = mode[R :> (x: not Number(x))]
def describe_full[R, :mode_freq] = max[frequency[R :> (x: not Number(x))]]

def describe_full[R, :mean]  = mean[R :> Number]
def describe_full[R, :std]   = sample_stddev[R :> Number]
def describe_full[R, :"25%"] = percentile[(R :> Number), 25]
def describe_full[R, :"50%"] = median[R :> Number]
def describe_full[R, :"75%"] = percentile[(R :> Number), 75]
```

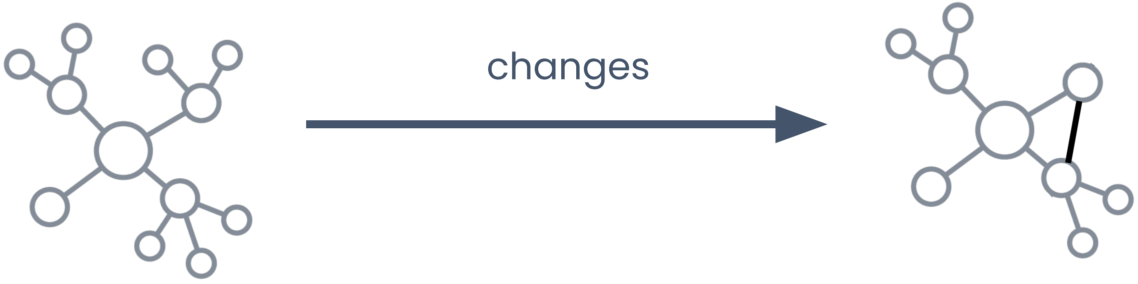
This implementation feels very dynamic in nature but this is all handled at compile-time and the logic is specialized to the actual `R`.

RelationalAI

Core Innovations for
Relational Knowledge Graphs

Incremental Computation

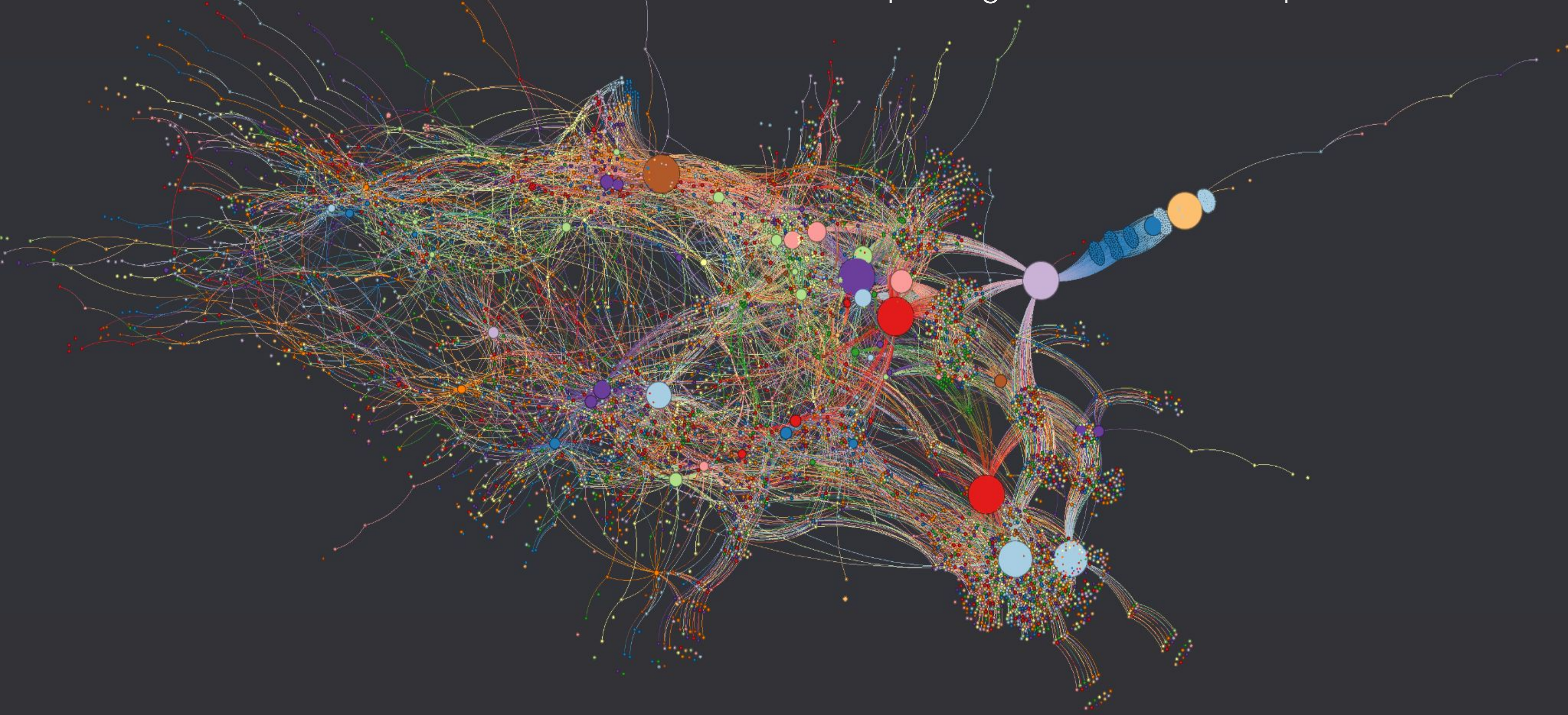
Incremental Computation



View / Reasoning / Knowledge / Semantics Layer

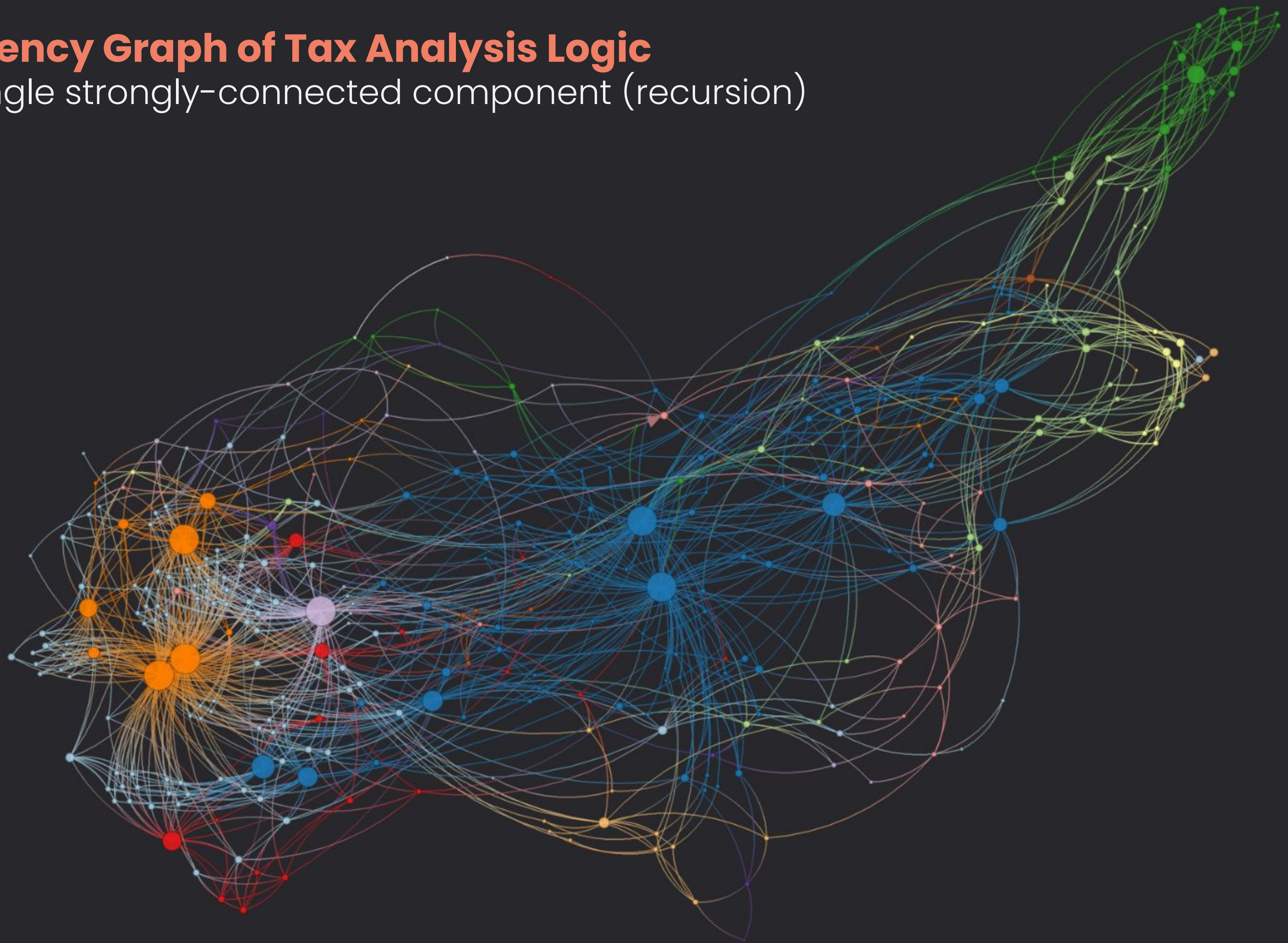
Dependency Graph of Tax Analysis Logic

3.6K relations, 13K dependencies
replacing millions of lines of procedural code



Dependency Graph of Tax Analysis Logic

Focus: Single strongly-connected component (recursion)

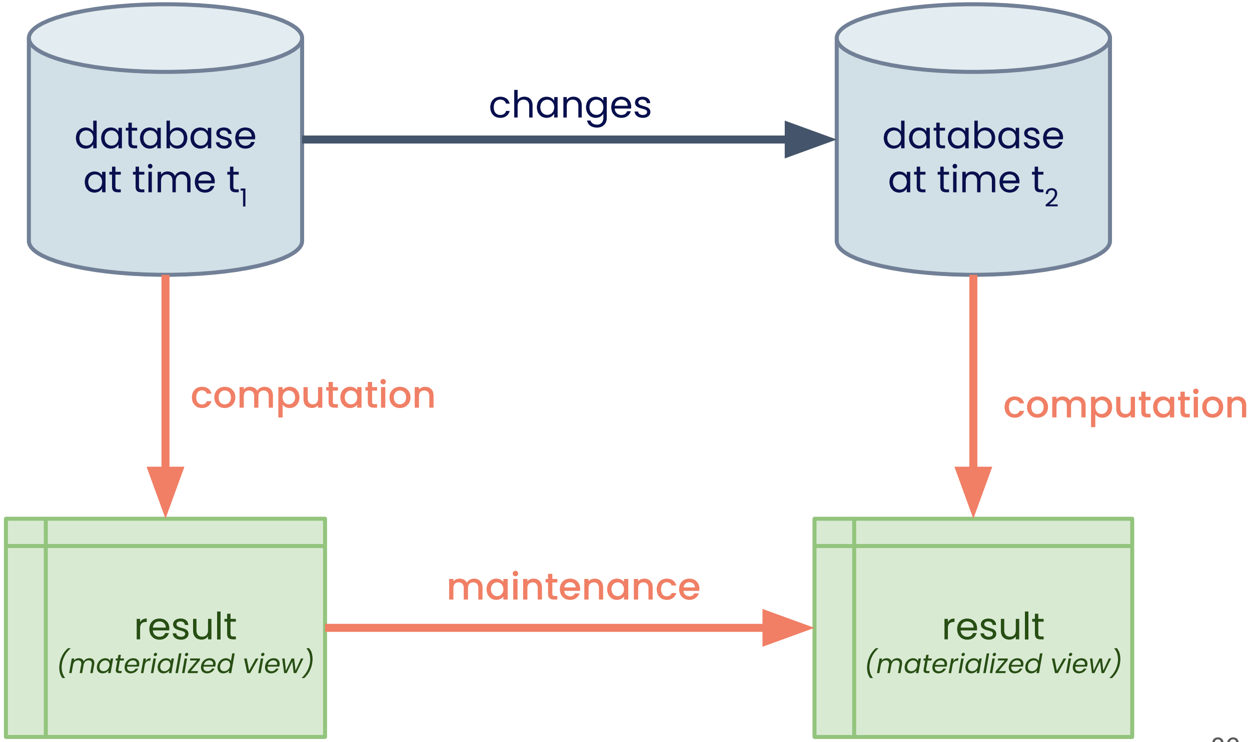


Incremental Computation

Goal: maintain computations (views) incrementally wrt changes in the inputs.

Inputs can change along two dimensions:

I) Changes caused by changes to the state of the database



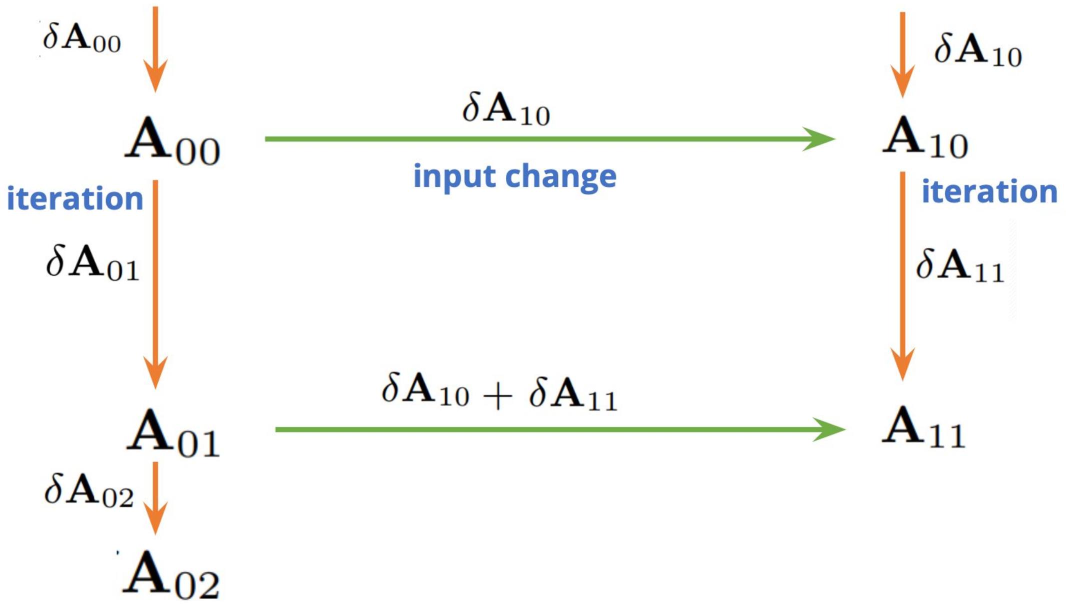
Incremental Computation

Goal: maintain computations (views) incrementally wrt changes in the inputs.

Inputs can change along two dimensions:

I) Changes caused by changes to the state of the database

II) Changes caused by iterative computations



The Incremental Maintenance Stack

RAI aims to support incremental processing of changes to code as well as data.

Dependency tracking to determine which computations are affected by a change.

Demand-driven execution to only compute what users are actively interested in.

Differential computation to incrementally maintain even general recursion.

Semantic information to determine that a recursive computation is monotonic

Semantic optimization to recover better maintenance algorithms where possible.

RelationalAI

Algorithms for Incremental Computation

- Semi-naive evaluation for stratified Datalog
- Generalized semi-naive evaluation (recognize more logic as monotonic)
- Differential dataflow for general non-monotonic logic

Naive

for $t = 1, 2, \ldots$ **do**
$R_t = F(R_{t-1})$
if $R_t = R_{t-1}$ **return** R_t
end

F : recursive program

Generalized Semi-naive

for $t = 1, 2, \ldots$ **do**
$\delta R_t = F(R_{t-1}) \ominus R_{t-1}$
$R_t = R_{t-1} \oplus \delta R_t$
if $\delta R_t = \varnothing$ **return** R_t
end

Differential Program

RelationalAI

Incremental Computation: Resources and Influences

- **Convergence of Datalog over (Pre-) Semirings**
  Abo Khamis, Ngo, Pichler, Suciu, Wang, PODS 2022 (Best paper award)

- **Differential dataflow**
  McSherry, Murray, Isaacs, Isard, CIDR 2013

- **Reconciling Differences**
  Green, Ives, Tannen, Theory of Computing Systems 2011

- **F-IVM: Incremental View Maintenance with Triple Lock Factorization Benefits**
  Nikolic and Olteanu, SIGMOD 2018

RelationalAI

Core Innovations for Relational Knowledge Graphs

Join Algorithms

Knowledge Graphs need different 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)
```

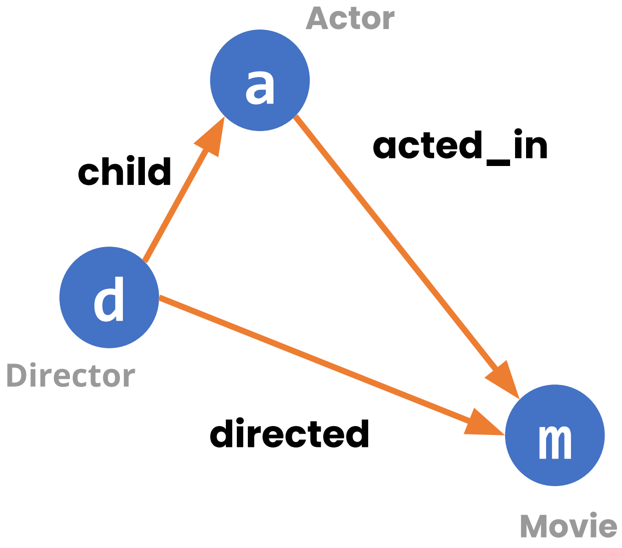
Binary join options:

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

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

```
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

Three ways of looking at WCOJ

RelationalAI

We use **worst-case optimal join algorithms**. This is a new class of algorithms whose properties and trade-offs are not yet well understood.

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

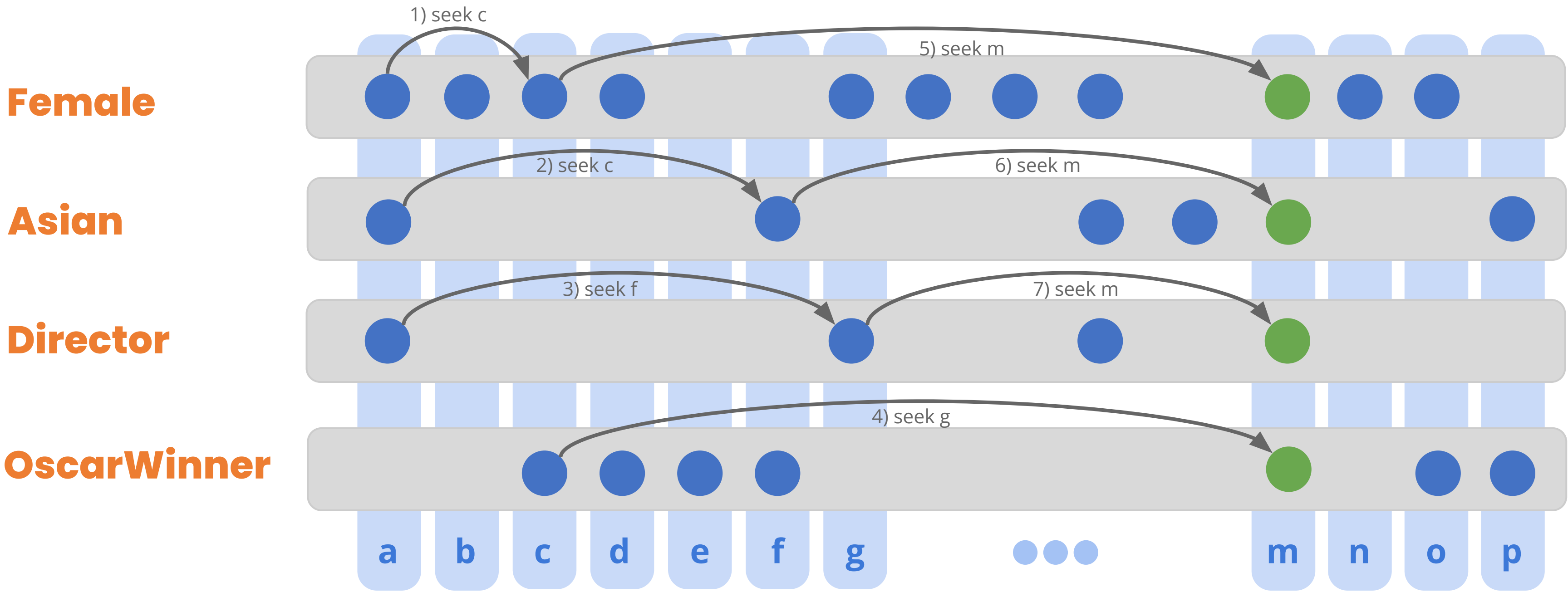
We look at the properties from three angles:

⇒ **Exploit sparsity in data**

⇒ **Recast the subquery problem and embrace correlation**

⇒ **Recast index selection problem**

WCOJ uses sparsity of all relations to narrow down search



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

Worst-case Optimal Joins: Basic Background

Multi-way joins are used **continuously**, 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, _)
```

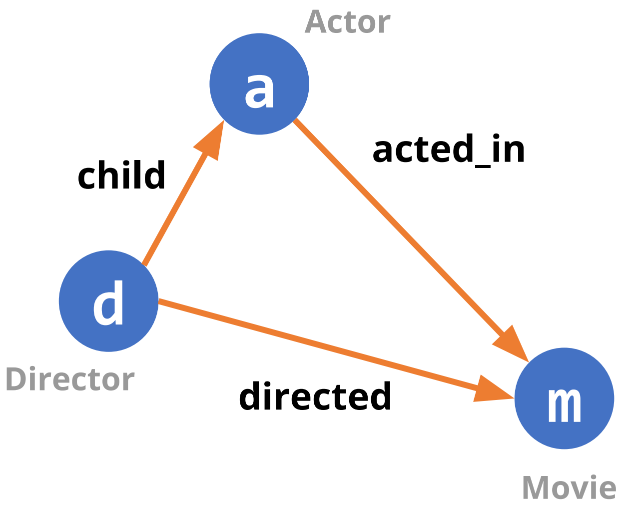
find directors **d** who `directed` some movie and have *some* `child`

```
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)



WCOJ exploits all correlation **simultaneously**

How we recast the subquery problem

```
select
  user.id,
  (
    select count(*)
    from post
    where post.user_id = user.id
  )
from user
where user.country = 'Mordor'
```

Two undesirable approaches

(SQL systems attempt to rewrite and decorrellate to avoid these)

Top-down: Nested Loop

Outer query
`for each row do`
Subquery

Bottom-up: (over)-compute once and reuse

S = Subquery

Outer query
`join with S`

We address subqueries with two powerful and general methods

1. Uncorrelated subqueries are handled by **semantic optimizer**
2. Embrace correlation: **WCOJ is also a correlated join device!**

How we recast the index selection problem

Index-selection and auto-tuning is an unsolved problem.

RelationalAI users cannot be asked to manually define indexes, and even supervised tuning approaches are not acceptable.

Our solution:

- **Everything is an index in our graph-like schemas**

  Compare: RDF triple stores that create indexes for all orderings
  Compare: SQL table stores with an index for every functional dependency

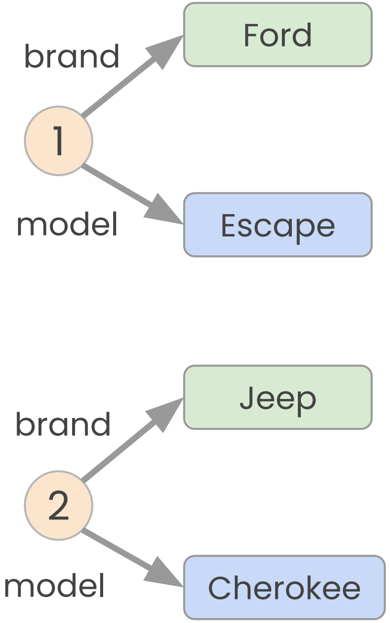
- **WCOJ is a device to create composite indexes on-the-fly, cheaply**

RelationalAI

How we recast the index selection problem

WCOJ is a device to create composite indexes on-the-fly, cheaply

Graph



Index Building Blocks

brand

1	Ford
2	Jeep

Ford	1
Jeep	2

model

1	Escape
2	Cherokee

Escape	1
Cherokee	2

Indexes available w/o sorting

Ford
Jeep

Ford	Escape
Jeep	Cherokee

Ford	Escape	1
Jeep	Cherokee	2

Ford	1
Jeep	2

Ford	1	Escape
Jeep	2	Cherokee

Escape
Cherokee

Escape	Ford
Cherokee	Jeep

Escape	Ford	1
Cherokee	Jeep	2

Escape	1
Cherokee	2

Escape	1	Ford
Cherokee	2	Jeep

1
2

1	Ford
2	Jeep

1	Ford	Escape
2	Jeep	Cherokee

1	Escape
2	Cherokee

1	Escape	Ford
2	Cherokee	Jeep

RelationalAI

Our Evaluation Strategy: Compiled and Vectorized

	Tuple at-a-time Interpreter	Compiler	Vectorized Interpreter
Low latency	✓	✗	✓
Good performance per tuple	✗	✓	✓

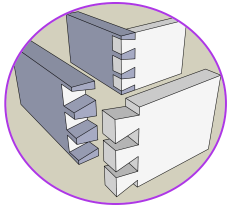
Vectorized WCOJ is an open research problem!

Compiler and vectorized interpreter are implemented in Julia, which helps with the maintenance concerns of two back-ends.

Compiled and vectorized evaluation can be mixed in single plan!

julia

Dovetail Join Compiler *(not yet published)*



RelationalAI

Dovetail Join is a new join algorithm invented in January 2019.

It addresses typical sources of inefficiency with worst-case optimal join algorithms:

OVERHEAD	ADDRESSED VIA
Runtime bookkeeping for join state	Encode as finite state machine
Overhead from abstract iterators	Works directly on raw iterators
Dynamic dispatch	Specialization

Dovetail/FSM is an implementation of Dovetail that leverages Julia's runtime code generation to produce ultra-efficient join kernels.

Join Algorithms: Resources and Influences

Worst-case optimal join algorithms

- **Worst-case Optimal Join Algorithms**
  Ngo, PODS 2012 (Best paper award)
- **Leapfrog Triejoin: A Simple, Worst-Case Optimal Join Algorithm**
  Veldhuizen, ICDT 2015 (Best Newcomer Award)
- **A Worst-case Optimal Join Algorithm for SPARQL**
  Hogan, ISWC 2019
- **Worst-Case Optimal Graph Joins in Almost No Space**
  Arroyuelo, SIGMOD 2021

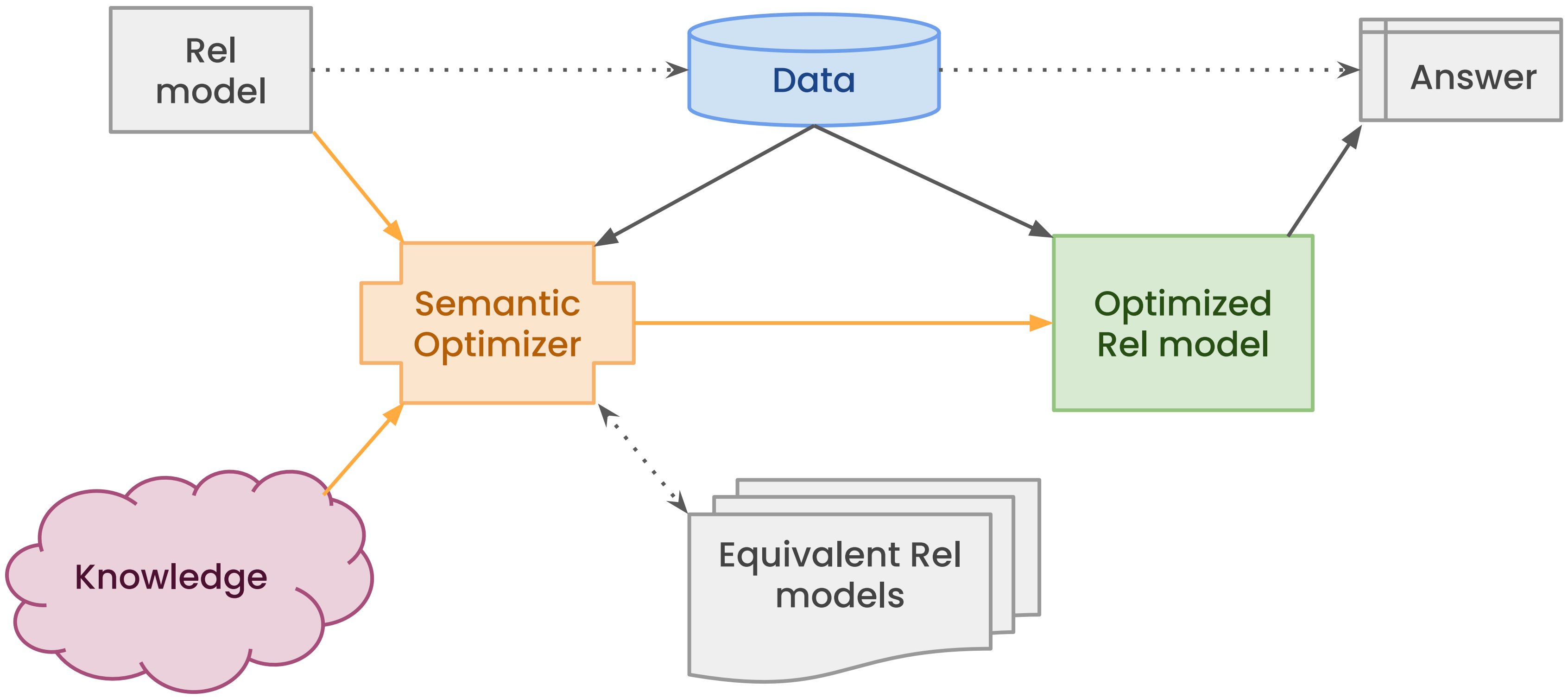
Correlated Subqueries

- **Unnesting Arbitrary Queries**
  Neumann, BTW 2015
- **How Materialize and other databases optimize SQL subqueries**
  Brandon, Materialize Deep Dive, March 2021

Core Innovations for
Relational Knowledge Graphs

Semantic Optimization

Semantic Optimization



What Knowledge

User-specified constraints

- Functional dependencies etc
- Total functions, disjoint etc

Mathematical axioms

- Semirings, rings, fields, lattices, …

Learned from the data

- Data: Summary statistics, histograms
- Query: Samples cardinality estimation

$x + y = y + x$	commutativity of $+$
$x \times y = y \times x$	commutativity of $\times$
$z \times (x + y) = z \times x + z \times y$	distributivity of $\times$ over $+$
$x \times 1 = x$	identity of $\times$ is 1
$x + 0 = x$	identity of $+$ is 0
$x \times 0 = 0$	0 is an annihilator

$\min(x, y) = \min(y, x)$	commutativity of $\min$
$x + y = y + x$	commutativity of $+$
$z + \min(x, y) = \min(z + x, z + y)$	distributivity of $+$ over $\min$
$x + 0 = x$	identity of $+$ is 0
$\min(x, +\infty) = x$	identity of $\min$ is $+\infty$
$x + \infty = \infty$	∞ is an annihilator

Semantic Optimization

Using mathematical knowledge in semantic optimization

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

optimizer

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

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

~~optimizer~~

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

```
count[f × g]
```

optimizer

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

Semantic Optimization is not Syntactic or Ad-hoc

```
count[a, b, c: R(a) and S(b) and T(c) and a < b < c]
```

optimizer

```
sum[b: count[a: R(a) and S(b) and a < b] *
       count[c: S(b) and T(c) and b < c]]
```

Semantic Optimization is not Syntactic or Ad-hoc

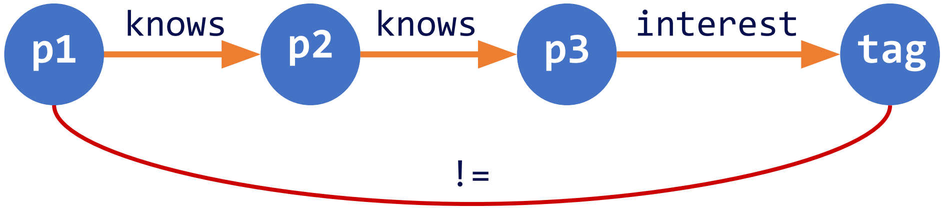
```
count[x, y: R(x) and S(y) and x != y]
```

optimizer

$$\mathbf{1}_{x \neq y} = 1 - \mathbf{1}_{x=y}$$

```
count[R] * count[S] - count[x, y: R(x) and S(y) and x = y]
```

LSQB Query 6




```
def q6 =
    count[p1, p2, p3, tag:
        knows(p1, p2) and
        knows(p2, p3) and
        interest(p3, tag) and
        p1 != p3]
```

optimizer

RAI on 1 core: **11**s
Umbra on 1 core: **76**s
Umbra on 48 cores: **2.5**s

```
def q6   = sum[tmp[p3] for p1, p2, p3 where knows(p1, p2) and knows(p2, p3)] - err1 - err2
def err2 = sum[tmp[p3] for p1, p2, p3 where knows(p1, p2) and knows(p2, p3) and p1 = p3]
def err1 = sum[tmp[p1] for p1, p2 where knows(p1, p2)]
def tmp[p3] = count[tag: interest(p3, tag)]
```

[LSQB: A Large-Scale Subgraph Query Benchmark]

Semantic Optimization: Running Total

```
def running_total[t] =
    sum[series[prev] for prev where prev <= t]
```

optimizer

Knowledge: ordering on the temporal dimension

```
def running_total[t] =
    series[t], first(t)

def running_total[t] =
    running_total[previous[t]] + series[t]
```

```
def partial_order(D, ⩽) =
    reflexive(D, ⩽) and
    antisymmetric(D, ⩽) and
    transitive(D, ⩽)

def reflexive(D, ~) =
    forall(a ∈ D: a ~ a)

def transitive(D, ~) =
    forall(a ∈ D, b ∈ D, c ∈ D:
        a ~ b and b ~ c implies a ~ c)
```

(imagine not having to remember window function syntax!)

Semantic Optimization: Push Agg into Recursion

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

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

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

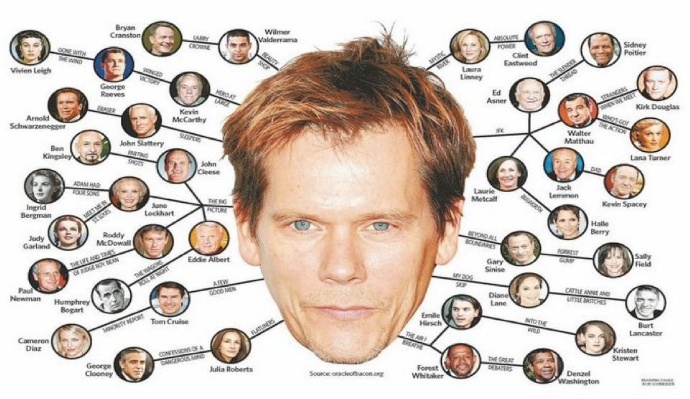
optimizer

```
def shortest_path[x, y] =
    min[edge[x, y]; shortest_path[x, t] + edge[t, y] from t]
```

Semantic Optimizer: Push Demand into Recursion

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)
    ]
```

Optimization supports Abstraction

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

No need for separate single-source vs all-pairs definitions
Reuse the very large path relation.

```
def scc[x] = min[v: reachable(x,v) and reachable(v, x)]
```

Reuse the very large reachable relation.

```
def wcc[x] = min[reachable_undirected[x]]
```

Reuse the very large reachable_undirected relation.

```
def mean[R] = sum[R] / count[R]
```

Pretty bad without aggregation optimization

Semantic Optimization: Resources and Influences

- **FAQ: Questions Asked Frequently**
  Khamis, Ngo, Rudra, PODS 2016 (Best Paper Award)
- **What Do Shannon-type Inequalities, Submodular Width, and Disjunctive Datalog Have to Do with One Another**
  Khamis, Ngo, Suciu, PODS 2017
- **Precise complexity analysis for efficient Datalog queries**
  Tekle et al., PPDP 2010
- **Functional Aggregate Queries with Additive Inequalities**
  Khamis et al., PODS 2019
- **Convergence of Datalog over (Pre-) Semirings**
  Khamis, Ngo, Pichler, Suciu, Wang, PODS 2022 (Best paper award)
- **Factorised representations of query results: size bounds and readability**
  Olteanu, Zavodny, ICDT 2012 (2022 Test of time award)

RelationalAI

Core Innovations for
Relational Knowledge Graphs

Live Programming and Incrementality

RelationalAI

The Incremental Maintenance Stack

RAI aims to support incremental processing of changes to code as well as data.

Dependency tracking to determine which computations are affected by a change.

Demand-driven execution to only compute what users are actively interested in.

Differential computation to incrementally maintain even general recursion.

Semantic information to determine that a recursive computation is monotonic

Semantic optimization to recover better maintenance algorithms where possible.

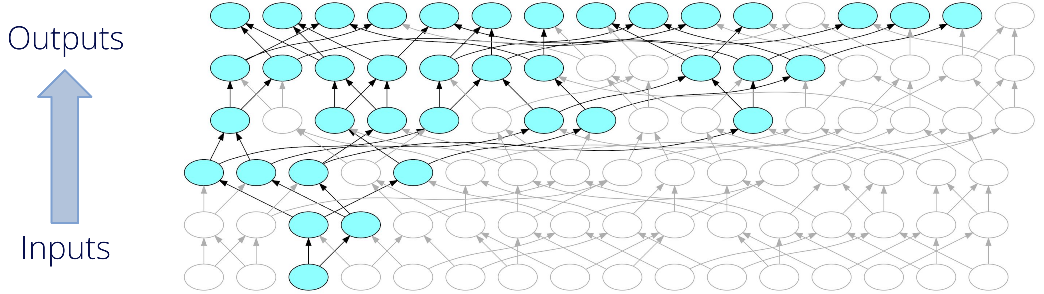
Incrementality and Demand-driven Evaluation

Eagerly maintaining the entire model is not a good idea at this scale.

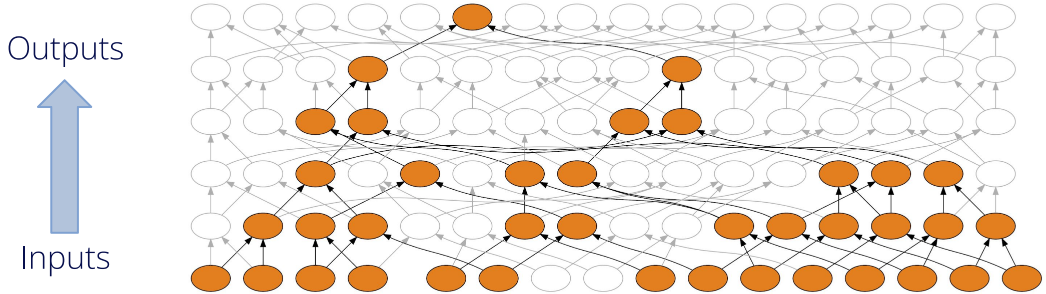
RAI is entirely **demand-driven**, which means that computations only happen when the result is needed (or when executed in the background to catch up). The architecture is based on PL incremental compiler research for IDEs.

Challenges:

- when to do invalidation and evaluation
- incrementally maintaining cyclic computation (scc)

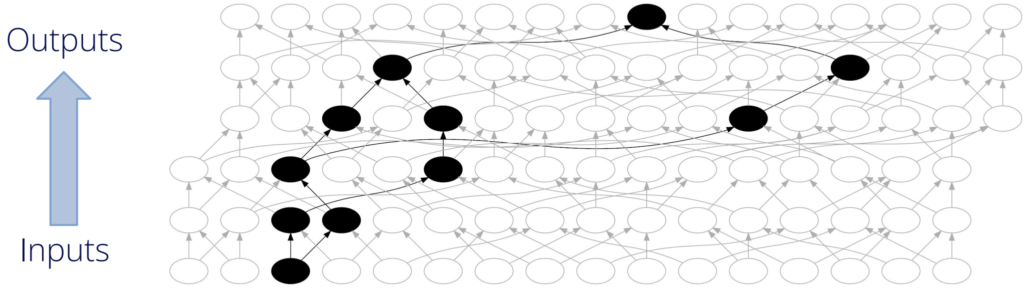


Eager maintenance is bad



Lazy maintenance is bad

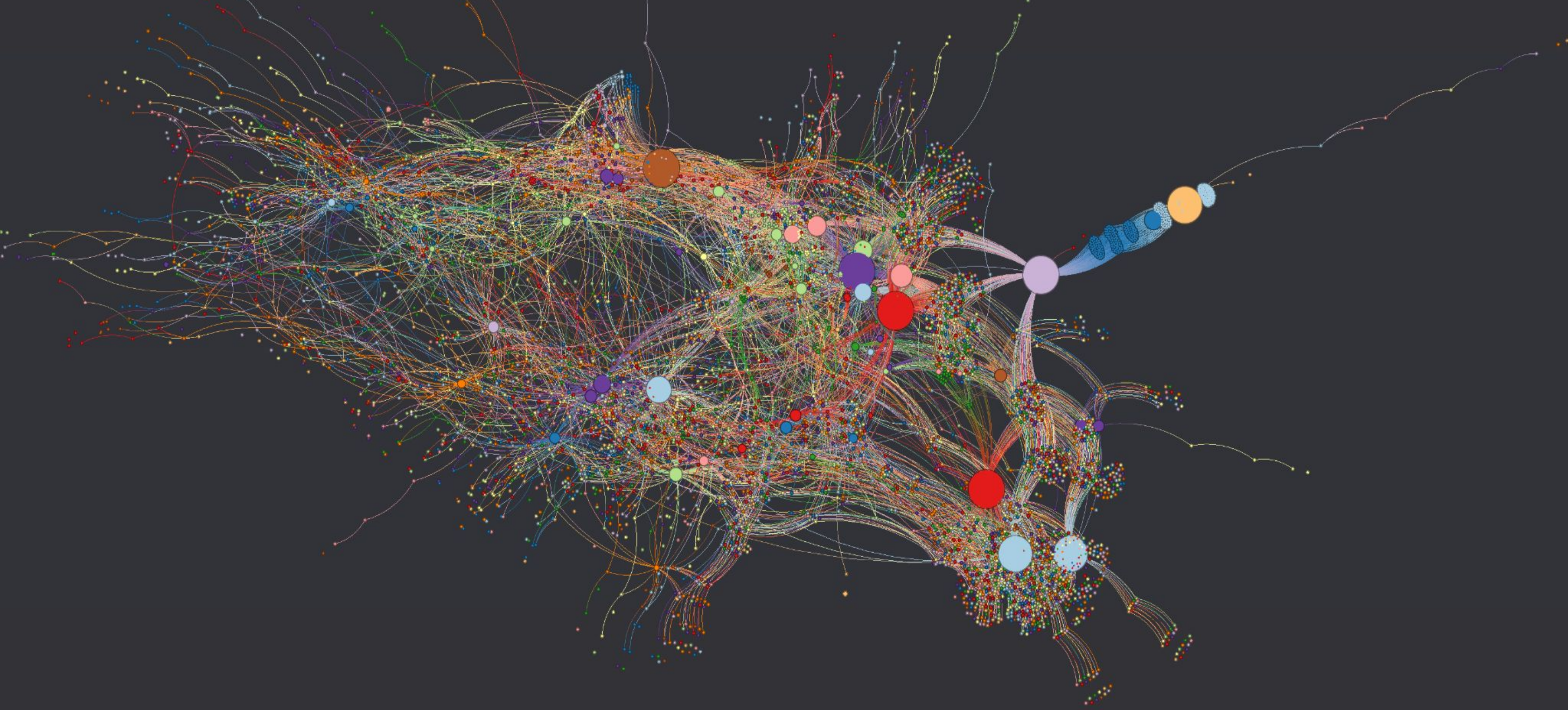
detecting dirty computations is too expensive when an output is quered.



Best: Eager invalidation lazy evaluation

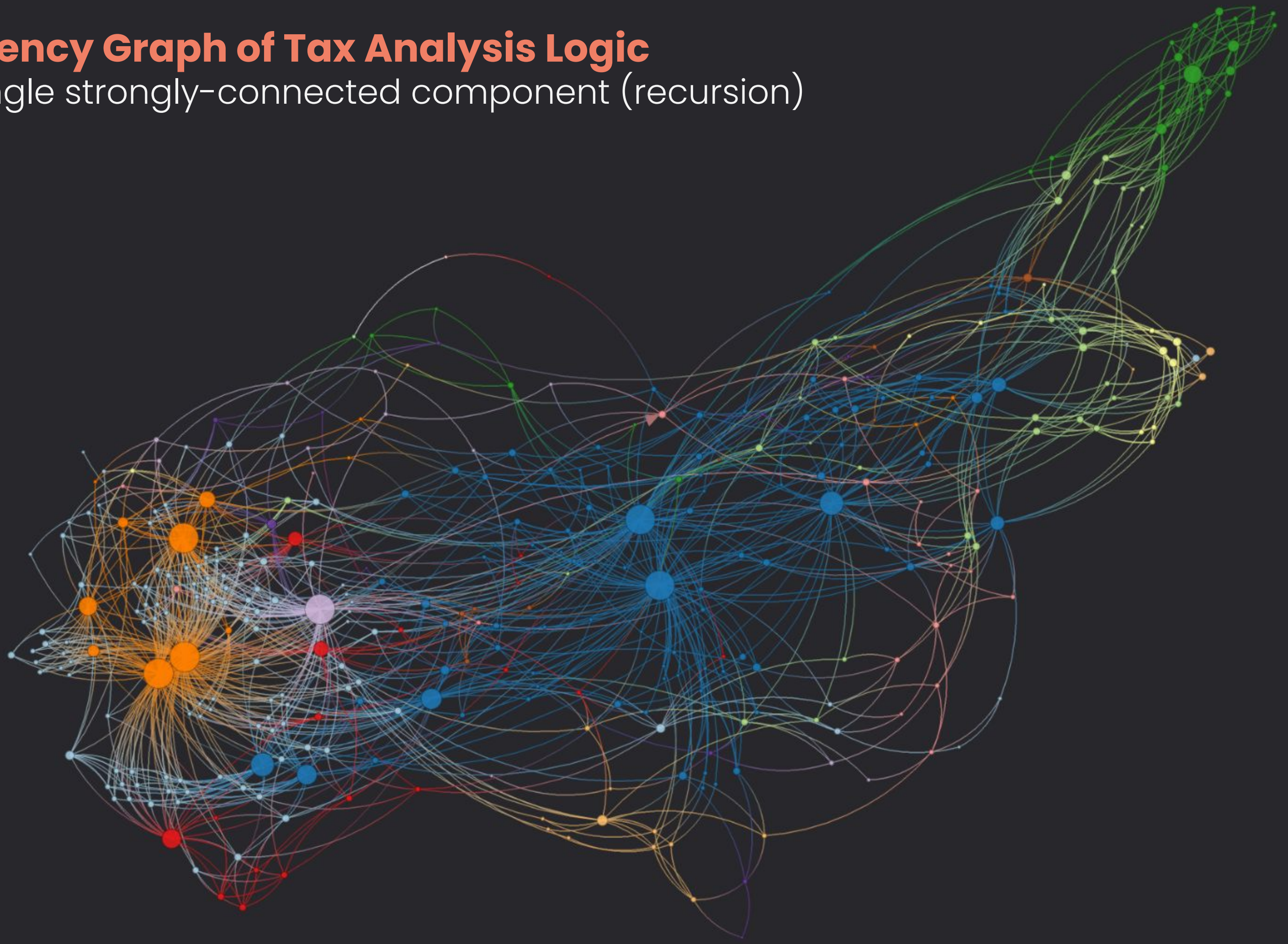
Dependency Graph of Tax Analysis Logic

3.6K relations, 13K dependencies
replacing millions of lines of procedural code



Dependency Graph of Tax Analysis Logic

Focus: Single strongly-connected component (recursion)



Dependency Graph of Tax Analysis Logic

Focus: Single node with many dependencies

Incrementality and Demand-driven Evaluation

The architecture is based on PL incremental compiler research for IDEs.

Key ingredients:

- Precise dependency tracking (treat access to the catalog as queries)
- Memoization and invalidation (on input changes)

We've open-sourced **Salsa.jl**, our framework for writing responsive compilers.

- Responsive compilers - Nicholas Matsakis - PLISS 2019
- JuliaCon 2020 - Salsa.jl - Nathan Daly

RelationalAI

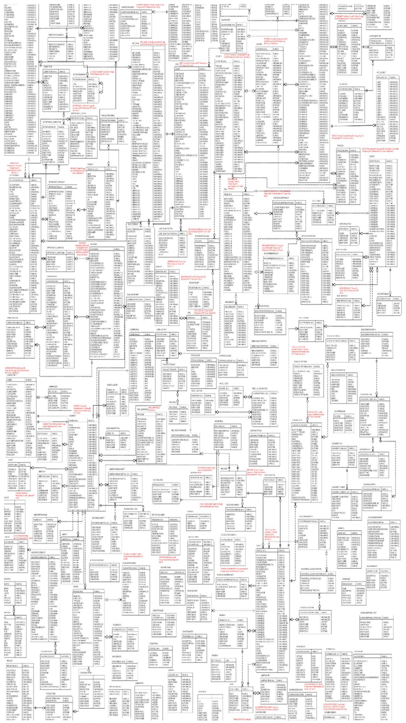
Core Innovations for
Relational Knowledge Graphs

Relational Models for Machine Learning

Unconstrained Optimization Models

Current Practice in Machine Learning

Beautiful relational schema without redundancy



Feature extraction query

Step 1: **throw away all** the structure and knowledge on the data set (eg dependencies).

Design matrix: the ultimate denormalization

k	f1	f2	f3	...	y

TensorFlow

PyTorch

scikit learn

sku	store	date	sold
1	S1	2022-03-26	5
1	S1	2022-03-27	7
1	S1	2022-03-28	3

sku	color	price
1	Red	$5.14

store	city	size
S1	Seattle	4000 sqft

date	temp
2022-03-26	53

city	state
Seattle	WA

sku	store	date	sold	color	price	city	size	state	temp
1	S1	2022-03-26	5	Red	$5.14	Seattle	4000 sqft	WA	53
1	S1	2022-03-27	7	Red	$5.14	Seattle	4000 sqft	WA	53
1	S1	2022-03-28	3	Red	$5.14	Seattle	4000 sqft	WA	53

sku	store	date	**sold**	**Red**	**Green**	price	**Seattle**	**San Diego**	size	WA	CA	temp
1	S1	2022-03-26	5	1	0	$5.14	1	0	4000 sqft	1	0	53
1	S1	2022-03-27	7	1	0	$5.14	1	0	4000 sqft	1	0	53

Relational Modelling for Machine Learning

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**.

Key innovations:

- Rel language - concisely expressing generic machine learning models
- **Automatic differentiation** of relational cost function
- **Semantic optimizer** - exploit relational structure and independence
- Optimization method executed iteratively in RAI system
- Execute large numbers of aggregations efficiently

Rel - Math for Linear Regression

Generic models

This is in a reusable library. Note this uses Rel schema abstraction (features is schema)

```
def predict_linear[X, M][k...] =
    sum[f: M[f] * X[f, k...]] + sum[f: M[f, X[f, k...]]] + M[:bias]

def linear_regression[X, Y, M] =
    minimize[rmse[predict_linear[X, M], Y]]
```

Application-specific instantiation

```
def features[:gdp_per_capita] = ...
def response = life_satisfaction

def model = linear_regression[features, response, initial_point]
```

Semantic Optimization for Covariance Matrix

sku	store	date	sold	Red	Green	price	Seattle	San Diego	size	WA	CA
1	S1	2022-03-26	5	1	0	$5.14	1	0	4000 sqft	1	0
1	S1	2022-03-27	7	1	0	$5.14	1	0	4000 sqft	1	0

Generic covariance matrix:

```
def covariance[j, k] =
    sum[st, sk, d: design_matrix[j, st, sk, d] * design_matrix[k, st, sk, d]]
```

Imagine the specialize to price and size:

```
def covariance[:price, :size] =
    sum[st, sk, d: design_matrix[:size, st, sk, d] * design_matrix[:price, st, sk, d]]
```

Price is independent of **store** and **date**
Size is independent of **sku** and **date**

```
def covariance[:price, :size] =
    (sum[st: features[:price, st]] * count[stores] * count[dates]) *
    (sum[sk: features[:size, sk]] * count[skus] * count[dates])
```

Relational Machine Learning: Resources and Influences

- **A Layered Aggregate Engine for Analytics Workloads**
  Schleich, Olteanu, Khamis, Ngo, Nguyen, SIGMOD 2019
- **Learning Models over Relational Data Using Sparse Tensors and Functional Dependencies**
  Khamis, Ngo, Nguyen, Olteanu, Schleich, PODS 2018, TODS 2020
- **The Relational Data Borg is Learning**
  Olteanu, VLDB 2020 Keynote (youtube recording: /watchv=0ic0jMjOpM0, /watchv=kWm-0BnbEoU)
- **Structure-Aware Machine Learning over Multi-Relational Databases**
  Schleich, PhD thesis, Honorable mention for the 2021 SIGMOD Jim Gray Doctoral Dissertation Award
- **Relational Knowledge Graphs as the Foundation for Artificial Intelligence**
  Aref (youtube recording: /watchv=VpyGbjUzG7Y)
- **Rk-means: Fast Clustering for Relational Data**
  Curtin, Moseley, Ngo, Nguyen, Olteanu, Schleich, AISTATS 2020

RelationalAI

Core Innovations for
Relational Knowledge Graphs

Relational Models for Mathematical Optimization

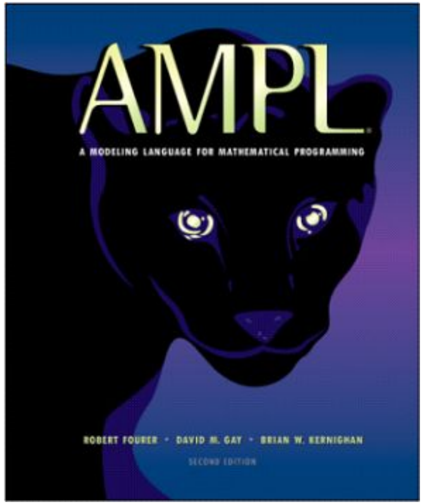
Constrained Optimization Models

RelationalAI

Optimization

GUROBI OPTIMIZATION

FICO® Xpress Optimization



Unconstrained Optimization

- Objective: the error/loss function
- Solver: differentiable function, often gradient descent
- All solutions are acceptable

Constrained optimization

- Objective: minimize or maximize the function
- Solver: LP, ILP, MIP etc
- Not all solutions are acceptable: constraints
- Mathematical optimization problems are specified in high-level math expressions (AMPL, JuMP). The problems are easily written in Rel

IBM CPLEX

Model for Manufacturing Problem

Given: P, a set of products

a_j = tons per hour of product j, for each $j \in P$

b = hours available at the mill

c_j = profit per ton of product j, for each $j \in P$

u_j = maximum tons of product j, for each $j \in P$

Define variables: X_j = tons of product j to be made, for each $j \in P$

Maximize: $$\sum_{j \in P} c_j X_j$$

Subject to: $$\sum_{j \in P} (1/a_j) X_j \leq b$$

$$0 \leq X_j \leq u_j, \text{ for each } j \in P$$

```
@variable(model, make[products])
@objective(model, Max, sum(prod_profit[p] * make[p] for p in products))
@constraint(model, sum(1 / prod_rate[p] * make[p] for p in products) <= 40)
@constraint(model, [p in products], 0 <= make[p] <= prod_max[p])
```

JUMP

```
var Make{p in PROD}
maximize Profit: sum{p in PROD} prod_profit[p] * Make[p];
subject to Time: sum{p in PROD} (1 / prod_rate[p]) * Make[p] <= 40;
subject to Limit{p in PROD}: 0 <= Make[p] <= prod_max[p]
```

AMPL

Relational Model

RelationalAI

Rel supports expressing the objective function and constraints.

The system grounds the constraint in the database and pass the problem to a solver (eg CPLEX, Gurobi, Xpress)

```
def total_profit =
    sum[prod_profit[p] * make[p] for p in products]

def time_avail() =
    sum[(1 / prod_rate[p]) * make[p] for p in products] ≤ avail

def demand_market() =
    forall(p in products: make[p] ≤ prod_market[p])
```

Optimization happens in the dependency graph, so inputs to the solver can computed Rel definitions or even other optimization problems.

RelationalAI

Core Innovations for
Relational Knowledge Graphs

Interfaces: SQL 💗 Rel

DuckDB-based SQL Interface

RelationalAI

DuckDB

DuckDB is an embeddable SQL OLAP database management system with great performance, excellent quality, small footprint and enjoying quick adoption.

RAI uses DuckDB for SQL support. Rel is used to model SQL tables, which are used by DuckDB for SQL query evaluation. Individual 'columns' can be data vs views.

DuckDB has outstanding support for working with a dynamic catalog.

Other approaches we evaluated:

- Calcite
- DuckDB query plan
- PostgreSQL parser

```
module order
    def orderkey[o] = ...
    def customer[o] = ...
    def orderdate[o] = ...
    def totalprice[o] = sum[num: charge[o, num]]
end
```

```
SELECT orderkey, customer, orderdate
FROM order
WHERE totalprice > 100
```

RelationalAI is partner of DuckDB Labs and member of the DuckDB foundation

RelationalAI

Recap

Large scale reasoning

SQL support with DuckDB engine

Immutable database in durable object storage, including immutable catalog. Write-optimized.

Relational models for **mathematical optimization**

Vectorized engine and **compiled WCOJ** algorithms, addressing subquery and index selection.

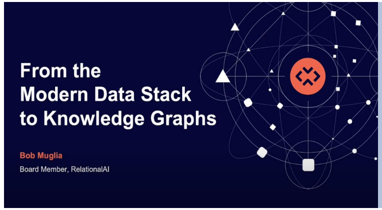
Relational machine learning utilizing semantic optimization.

Rel - An expressive relational language

Semantic optimization

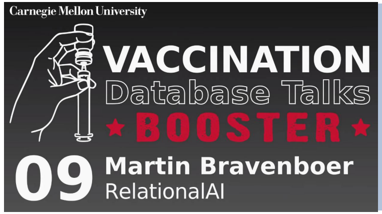
Incremental computation for fixpoint computation and database changes

Learn More



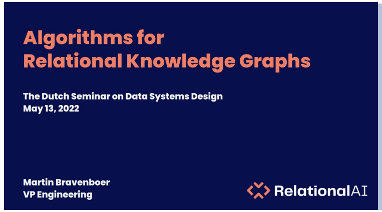
"KGC Bob Muglia" for modern data stack and relational knowledge graph

[Youtube]



"CMU RelationalAI" for RAI system overview

[Youtube]



"DSDSD Bravenboer" for different RAI system overview

[Youtube]

https://twitter.com/RelationalAI

RelationalAI

Thank you!