relationalAI

Molham Aref

Reinventing the Database for AI

July 19, 2019

relationalAI

We are a **mission-based team**

Scientific Impact

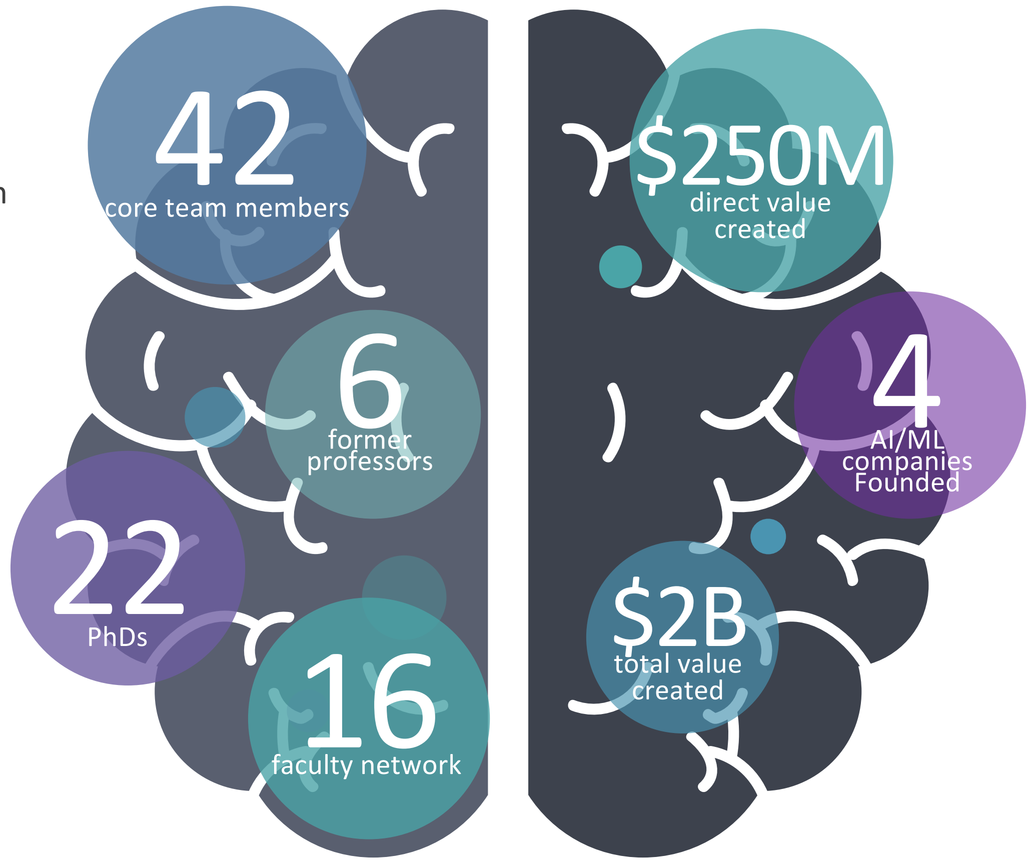
Deep computer science and mathematical expertise from several technical communities:

- Database systems and theory
- Machine learning
- Programming languages
- Operations research

2K+ publications

90K+ citations
(35K+ in last 5 years)

37+ award-winning papers
(3 this year!)



AI and ML Industrial Impact

- HNC Software ▶ FICO
- Retek ▶ Oracle Retail
- DemandTec ▶ IBM
- Brickstream ▶ FLIR
- Optimi ▶ Ericsson
- LogicBlox, Predictix ▶ infor

relationalAI

The Case for Relational Artificial Intelligence

A New Technology Category

relationalAI

What **if I tell you**

Databases should be Relational

Not **Controversial but it used to be**

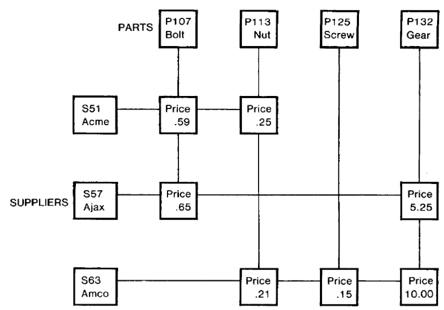
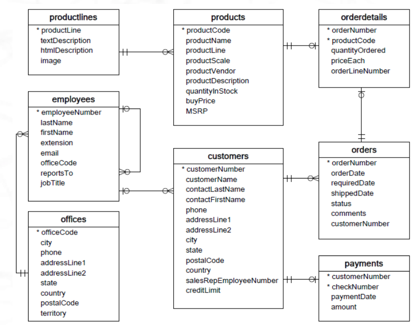


Fig. 1(a). A "Navigational" Database.

Navigational vs Relational



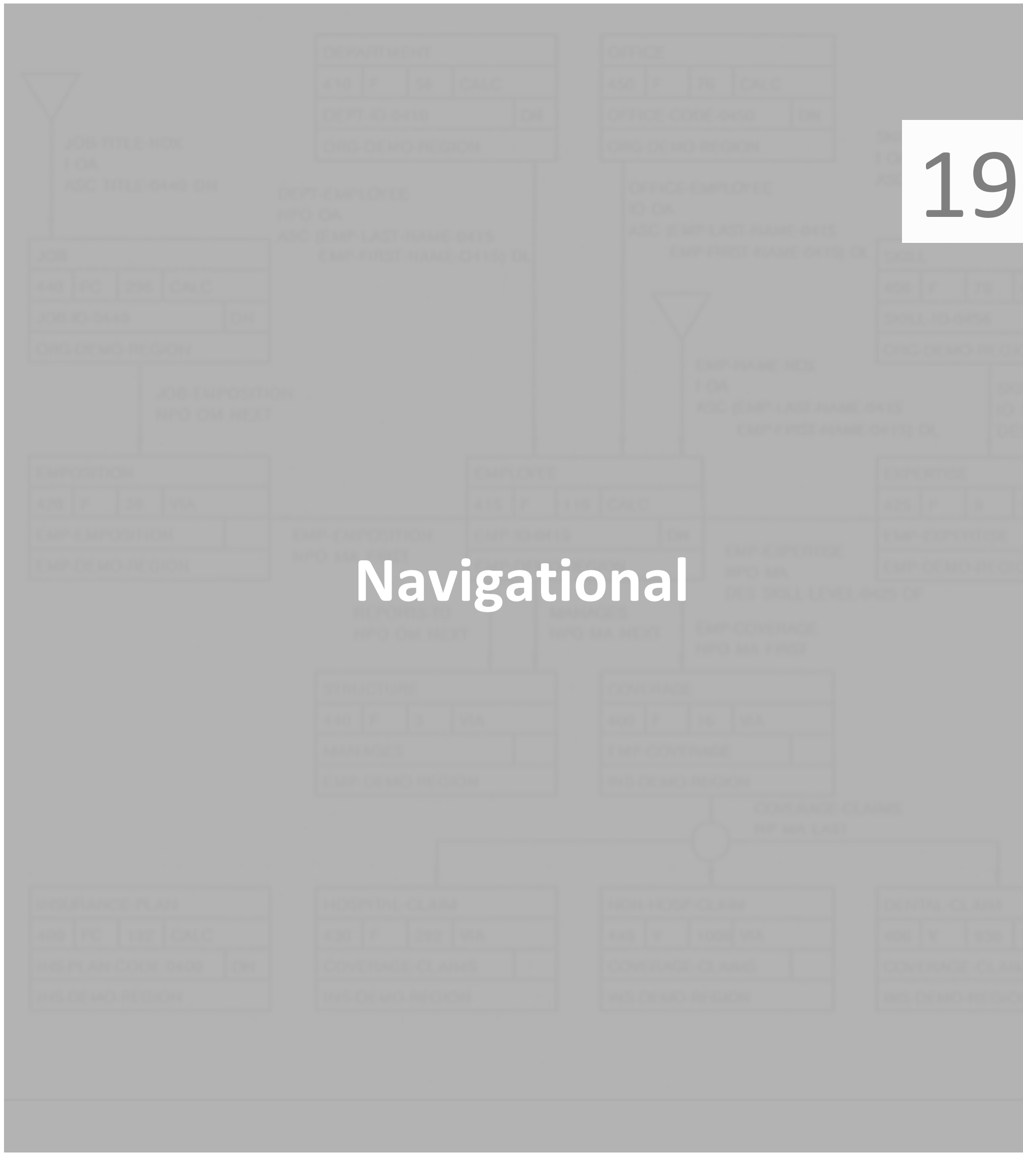
In the Navigational vs Relational DB wars of the 1980's,
Navigational DB's were the incumbent and Relational DBs were the underdog!

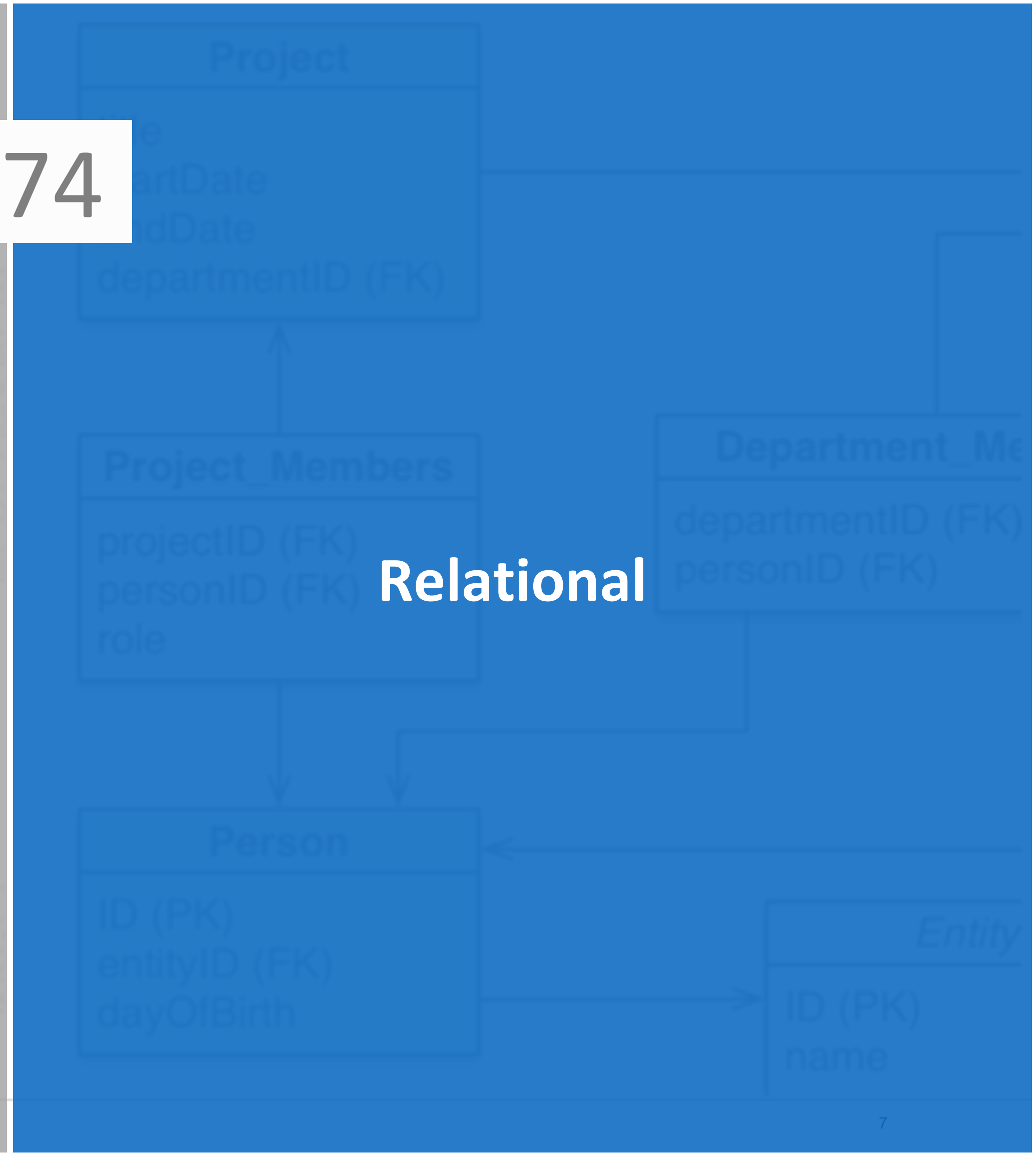
relationalAI

database

The Great Debate

1974





1974

Navigational

Charles Bachman

Weighing in with:

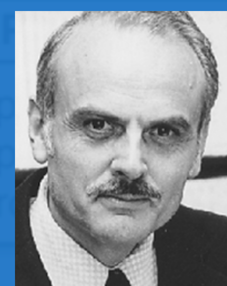
- Turing Award for Databases
- Integrated Data Store (IDS)
- Illustrious career at GE and Honeywell

Argument:

- Performance
  (it's impossible to implement the relational model efficiently)

- Programmers won't get it
  (Cobol programmers can't possibly understand relational languages)

Relational



Ted Codd

Weighing in with:

- Researcher at IBM

Argument:

- Separation of the What from the How
  (Argument for declarativity)

- Domain experts will get it
  (and they are cheaper and more plentiful than programmers)

1974

SO WHO WON?

Navigational

Charles Bachman

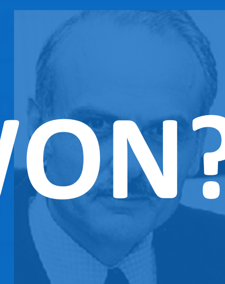
Weighing in with:

- Turing Award for Databases
- Integrated Data [illegible] (IDS)
- Illustrious career at [illegible]

Argument:

- Performance
  (it's impossible to implement the relational model efficiently)
- Programmers won't get it
  (Cobol programmers can't possibly understand relational languages)

Relational



Ted Codd

Weighing in with:

- Researcher at IBM

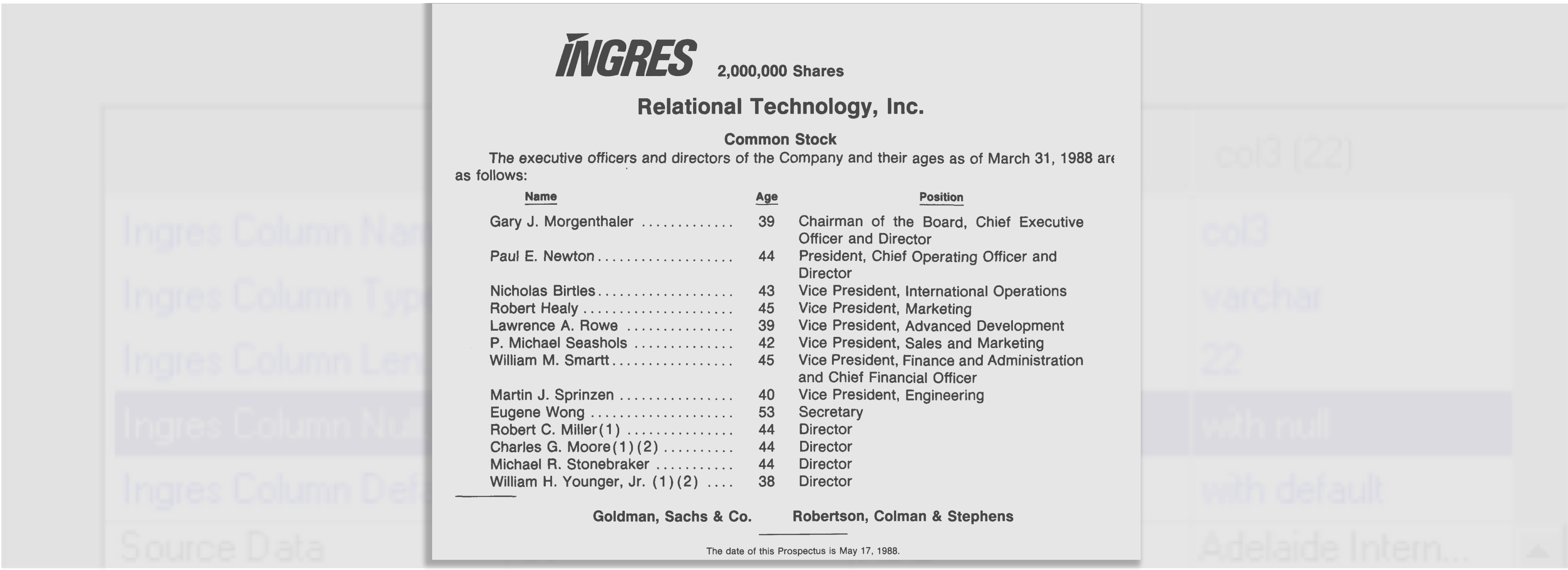
Argument:

- Separation of the What from the How
  (Argument for declarativity)
- Domain experts will get it
  (and they are cheaper and more plentiful than programmers)

ORACLE®

Oracle **(formerly Relational Software, Inc.)**

- Launched RDBMS in 1979
- IPO in 1986
- Current Market Cap: **$190.6B**



INGRES 2,000,000 Shares

Relational Technology, Inc.

Common Stock

The executive officers and directors of the Company and their ages as of March 31, 1988 are as follows:

Name	Age	Position
Gary J. Morgenthaler	39	Chairman of the Board, Chief Executive Officer and Director
Paul E. Newton	44	President, Chief Operating Officer and Director
Nicholas Birtles	43	Vice President, International Operations
Robert Healy	45	Vice President, Marketing
Lawrence A. Rowe	39	Vice President, Advanced Development
P. Michael Seashols	42	Vice President, Sales and Marketing
William M. Smartt	45	Vice President, Finance and Administration and Chief Financial Officer
Martin J. Sprinzen	40	Vice President, Engineering
Eugene Wong	53	Secretary
Robert C. Miller (1)	44	Director
Charles G. Moore (1) (2)	44	Director
Michael R. Stonebraker	44	Director
William H. Younger, Jr. (1) (2)	38	Director

Goldman, Sachs & Co. **Robertson, Colman & Stephens**

The date of this Prospectus is May 17, 1988.

INGRES

Ingres **(formerly Relational Technology, Inc.)**

- Launched RDBMS in 1981
- IPO’d in 1988 (sold prematurely to ASK in 1989)

relationalAI

RDBMS **Popularity**

DB-Engines Ranking May 2019

The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

Relational DBMS

1. Oracle

Relational DBMS

2. MySQL

Relational DBMS

3. Microsoft SQL Server

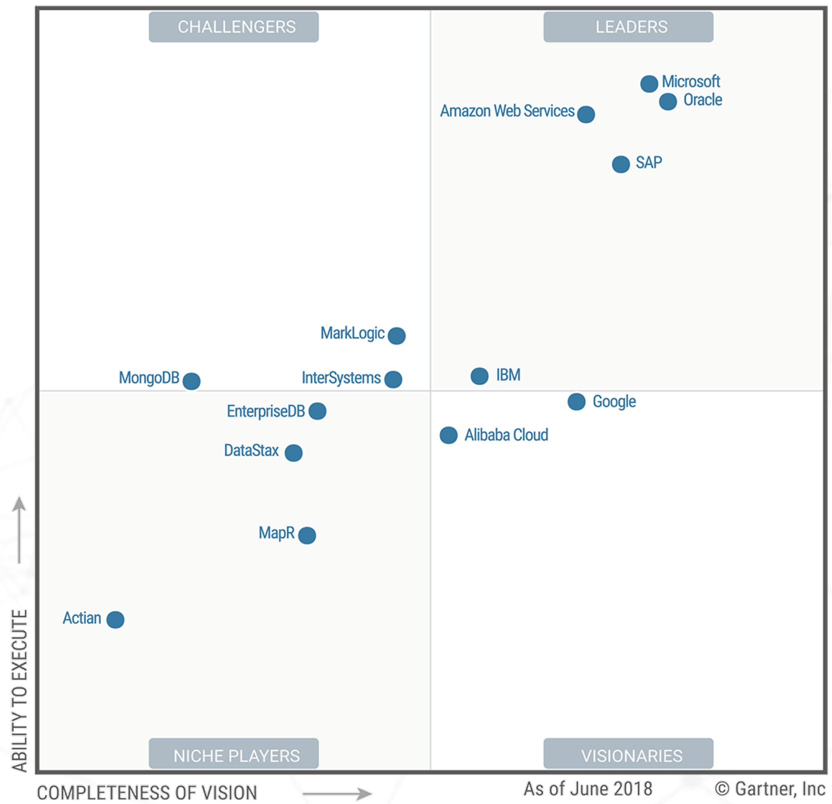
Relational DBMS

4. PostgresSQL

relationalAI

Analysts agree

Figure 1. Magic Quadrant for Operational Database Management Systems



Source: Gartner (October 2018)

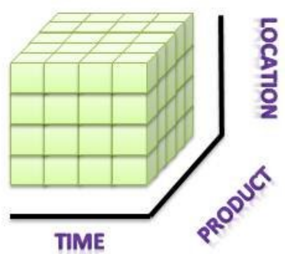
relationalAI

Why?

What **if I tell you**

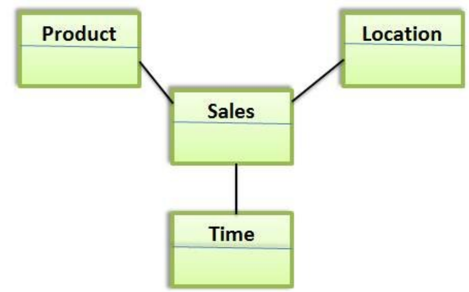
Business Intelligence should be Relational

Not **Controversial but it used to be**



MOLAP

MOLAP **vs** **ROLAP**



ROLAP

In the Multidimensional (i.e. Tensor) **vs** Relational OLAP wars of the 1990's, MOLAP was the incumbent and ROLAP was the underdog!

Tableau Software

- Launched in 2002
- IPO in 2013
- Current Market Cap: **$11.6B**

relationalAI

Analysts agree

relationalAI

Why?

relationalAI

What **if I tell you**

Artificial Intelligence should be Relational

relationalAI

What **if I tell you**

No way!!

Relational systems are **too slow!**

Tensors and linear algebra are the way we’ve always done it

TensorFlow

relationalAI

I am **here to tell you**

<u>Relational</u> Artificial Intelligence is Inevitable

Why?

Rest of the talk

relationalAI

The **Need for Speed**

"We track about **47 different hardware startups** that all have a unique approach" to accelerating AI.

Greg Brockman, CTO OpenAI, interviewed by Reid Hoffman, May 30, 2019

"**13 private chip companies focused on the AI market have raised more than $1.2 billion** in venture-capital funding"

- Barron's article "AI Chip Market Will Soar to $34 Billion in Five Years", Feb 20, 2019

"Today the job of training machine learning models is limited by compute, if we had faster processors we'd run bigger models...in practice we train on a reasonable subset of data that can finish in a matter of months. We could use improvements of several orders of magnitude – 100x or greater."

Greg Diamos, Senior Researcher, SVAIL, Baidu, From EE Times – September 27, 2016

AI's **biggest challenges are computational!**

ACCURACY

Search for better

- Parameters
- Hyper parameters
- Features
- Models

Don't make assumptions that you don't need to make (e.g. i.i.d. assumption)

VERSATILITY

Reasoning and (generalized) inference: From observations to unknowns in any time period

Inference of any property in the model (e.g., it's just as easy to infer price from sales as it is to infer sales from price)

ROBUSTNESS

Many "big data" problems are really a big collection of small data problems

Overcome challenges with small, incomplete, and dirty data problems by incorporating prior knowledge and expertise

SELF-SUPERVISION

"The future will be self-supervised" Yann LeCun

Build models of the world by observing it and searching model space for the models that have the most explanatory power

INTERPRETABILITY

Searching for models that are accurate and interpretable is harder than searching for accurate models

Interpretation in terms of prior knowledge and in language & ontology that humans understand

EXPLAINABILITY

Explainability typically implanted via separate shadow models that have to be learned

Explanation in terms of prior knowledge and in language & ontology that humans understand

FAIRNESS

It's not enough to exclude gender, ethnicity, race, age, etc as features to the models. Other features might be correlated.

Prejudice is a computational limitation: Reasoning about each person vs reasoning about the group

CAUSALITY

Understanding causality beyond A/B testing

Computationally very expensive

relationalAI

The Path to Performance: **Brawn**

Constant factors – Do same amount of work faster (i.e., brawn)

- Latency hiding: Memory hierarchy and network latencies (e.g., in memory and near-data computing)
- Parallelization: SIMD, multi-core, accelerators (e.g., GPU, TPU, FPGA)
- Specialization: Specialize for workload (e.g., JIT compilation), specialize for data

relationalAI

The Path to Performance: **Brains and Brawn**

Asymptotics – Do less work (i.e., brains)

- Specialize algorithm by exploiting problem structure
  - Algebraic (e.g., groups, semi rings, rings)
  - Combinatorial (e.g., fractional hypertree width)
  - Statistical (e.g., samples and sketches)
  - Geometric (e.g., fast multipole method)
- Solve similar but more tractable problem
  - Approximation (with error bars)

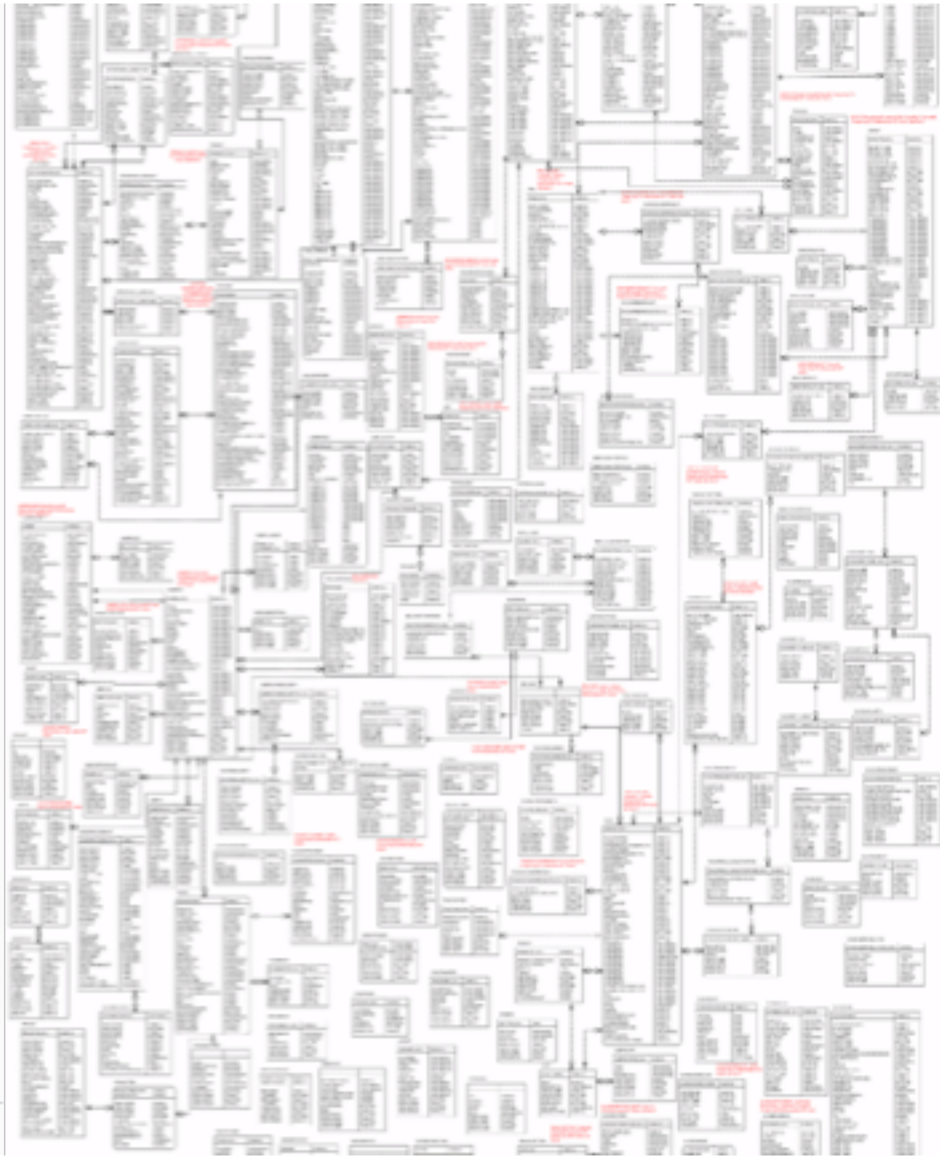
Brains

Do Less Work

relationalAI

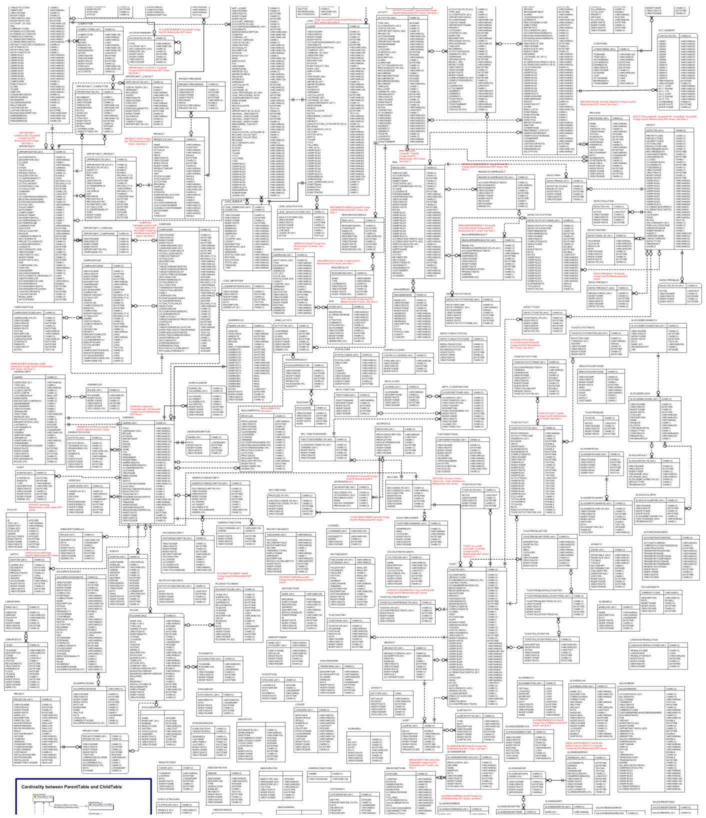
The **relational model dominates data management**

- The last 40 years have witnessed massive adoption of the relational model
    - It's hard to find any examples today of enterprises whose data isn't in a relational database
- Millions of human hours invested in building relational models and populating them with data
- Relational databases are rich with knowledge of the underlying domains that they model
- The availability and accuracy of large amounts of curated data has made it possible for humans (BI) and machines (AI) to **learn** from the past and to **predict** the future

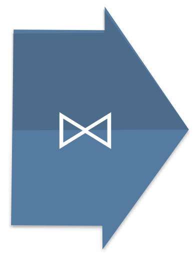


relationalAI

What's **the first thing we do when we build predictive models?**



Feature extraction query



We work hard to throw away **all** relational structure (and semi-structure) we worked so hard to build

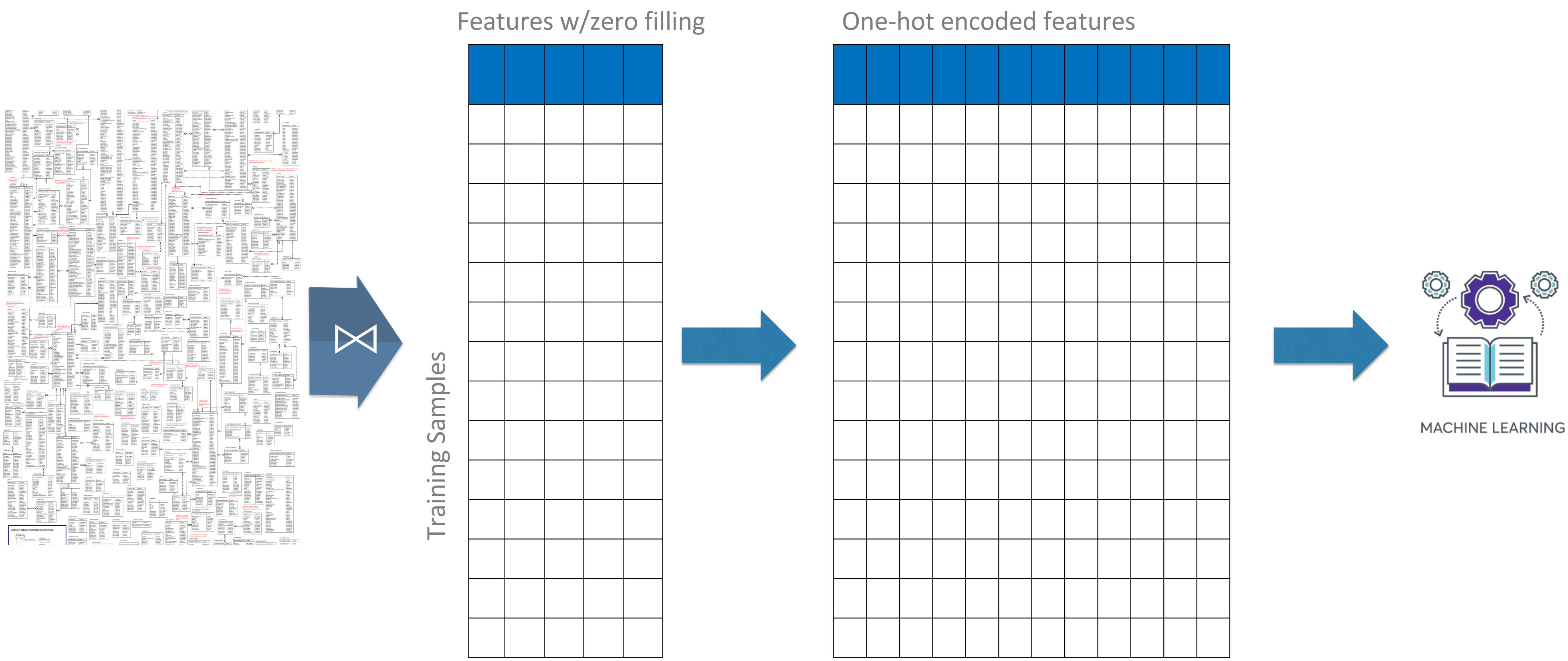
We end up throwing away important domain knowledge
that can help us build better AI models

Features

Examples

ID	x1	x2	x3	…	y

The **wastefulness does not end there**



relationalAI

The **wastefulness does not end there**

Features

One-hot encoded features

Training Samples

MACHINE LEARNING

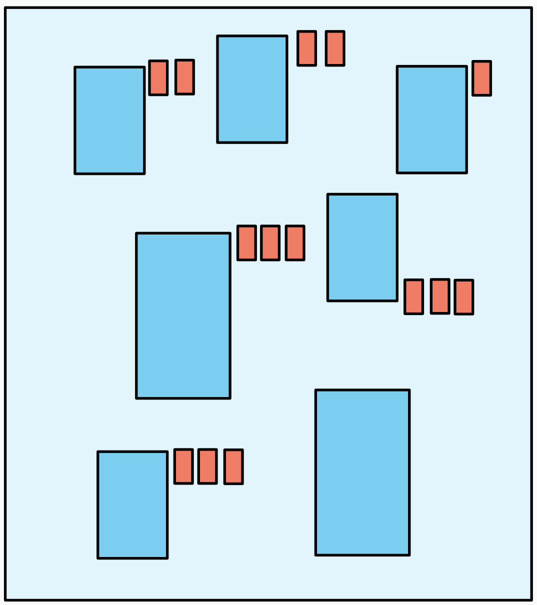
Revisit from first principles

- Avoid materializing the join
- Avoid filling in the zeros
- Avoid one-hot encoding
- Exploit relational structures to speed up learning
- Ideally, train models **faster** than the time it takes to produce the query output in the first place!

relationalAI

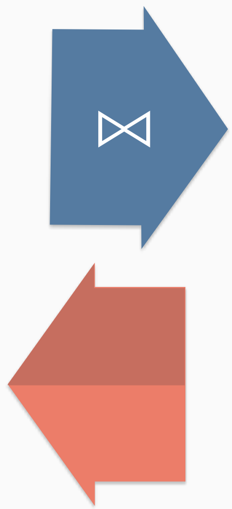
What **would a database do?**

1. Database



s: Sufficient statistics generated from model spec and feature extraction query. Computed via aggrefations

2. Feature extraction query



Features

Examples

ID	x1	x2	x3	…	y

3. Model specification
(e.g., “degree 2 ridge regression”)

relationalAI

Number of **Aggregates Varies By Model Class**

- **Supervised**
  - Regression

Model	# features	# params	# aggregates
Linear regression	n	n + 1	$\Theta(n^2)$
Polynomial regression	$\Theta(n^d)$	$\Theta(n^d)$	$\Theta(n^{2d})$
Factorization machines	$\Theta(n^d)$	$\Theta(nr)$	$\Theta(n^{2d})$

n: # input features
d: degree
r: rank

  - Classification

Model	# features	# aggregates
Decision trees	$\Theta(n)$	$\Theta(nbh)$

b: branching factor, *h*: depth (data-dependent)

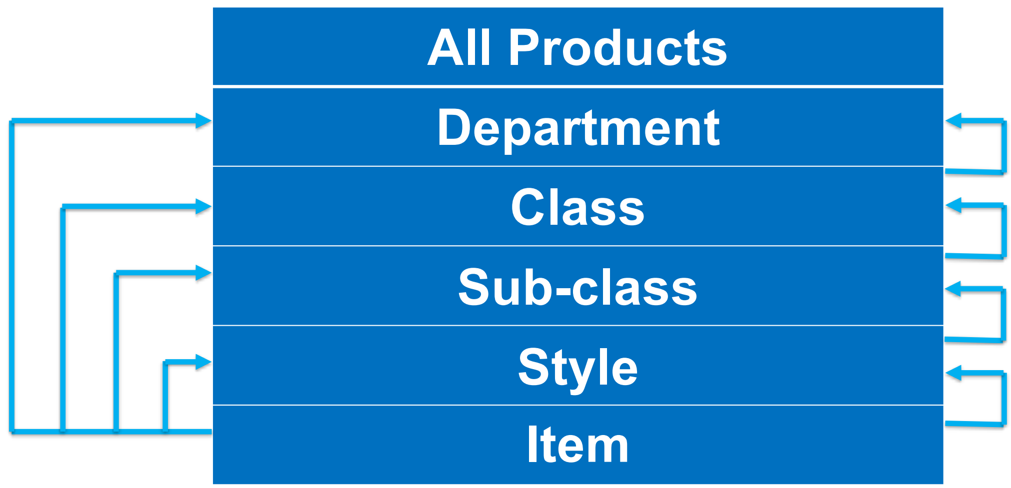
- **Unsupervised**

Model	# aggregates
K-means	$\Theta(kn)$
PCA	$\Theta(kn^2)$

k: # clusters

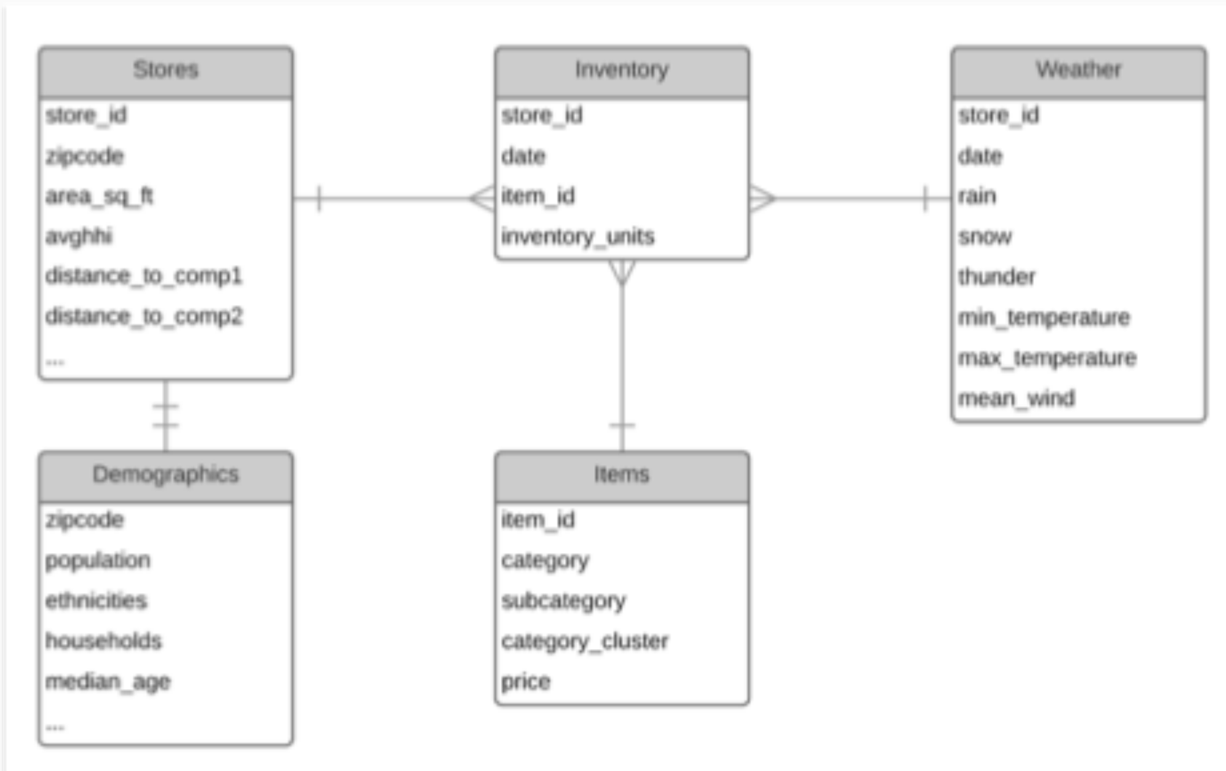
relationalAI

We **Efficiently Compute Those Aggregates**



relationalAI

Case Study: **Retail dataset**



relationalAI

Case Study: **Retail dataset**

Relation	Cardinality (# Tuples)	Degree (# k/v columns)	File size (csv)
Inventory	84,055,817	3 & 1	2 GB
Items	5,618	1 & 4	129 KB
Stores	1,317	1 & 14	139 KB
Demographics	1,302	1 & 15	161 KB
Weather	1,159,457	2 & 6	33 MB
		Total:	**2.1 GB**

relationalAI

Case Study: **Retail dataset – PostgreSQL & TensorFlow**

- The design matrix is constructed by joining together all the relations
- Train a linear regression model to predict sales by item, store, date from all the other features

Cardinality (# of tuples)	84,055,817
Degree (# of columns)	44 (3 & 41)
Size	23 GB
Time to compute in PostgreSQL	217 secs
Time to export from PostgreSQL	373 secs
Time to learn parameters with GD	> 12,000 secs

relationalAI

Case Study: **Retail dataset - comparison**

	Design matrix with PostgreSQL/TensorFlow		relationalAI	
	Time	Size	Time	Size
Original	--	2.1 GB	--	2.1 GB
Join Tables	217 secs	23 GB	--	--
Export DM	373 secs	23 GB	--	--
Aggregate	--	--	18 secs	37 KB
Parameter learning with GD	> 12 K secs	--	0.5 secs	--
Total	> 12.5 K secs		18.5 secs	
Improvement (1st Model)	**> 676x faster**		**11x smaller**	
Every model after	**> 24,000x faster**			

relationalAI

Does **it work for all model classes or methods?**

Supported methods include

- Linear regression
- Polynomial regression
- Factorization machines
- Decision trees
- Linear SVM
- Deep sum-product networks
- Naive Bayes Classifier (discrete case)
- Hidden Markov Model (discrete case)
- K-Means & K-Median clustering
- Gaussian Discriminant Analysis
- Linear Discriminant Analysis
- Principal component analysis
- Frequent item set mining (with Apriori algorithm)
- Computing empirical mutual information and entropy

(with more on the way)

relationalAI

So **what?**

Some context:

Moore's Law

gives us 2x speedup

every 1.5 years

According to Nvidia

GPUs give us a 2-10X

speed-up over CPUs

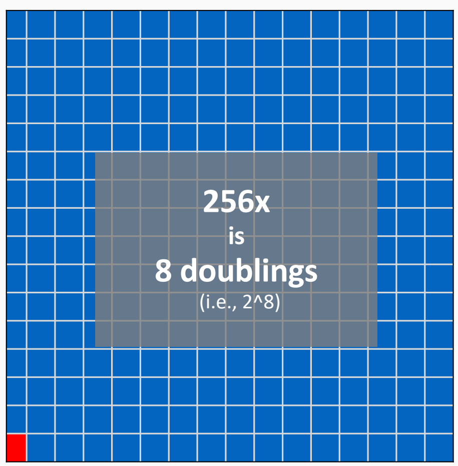
In other words, GPUs give us ~5 year advantage

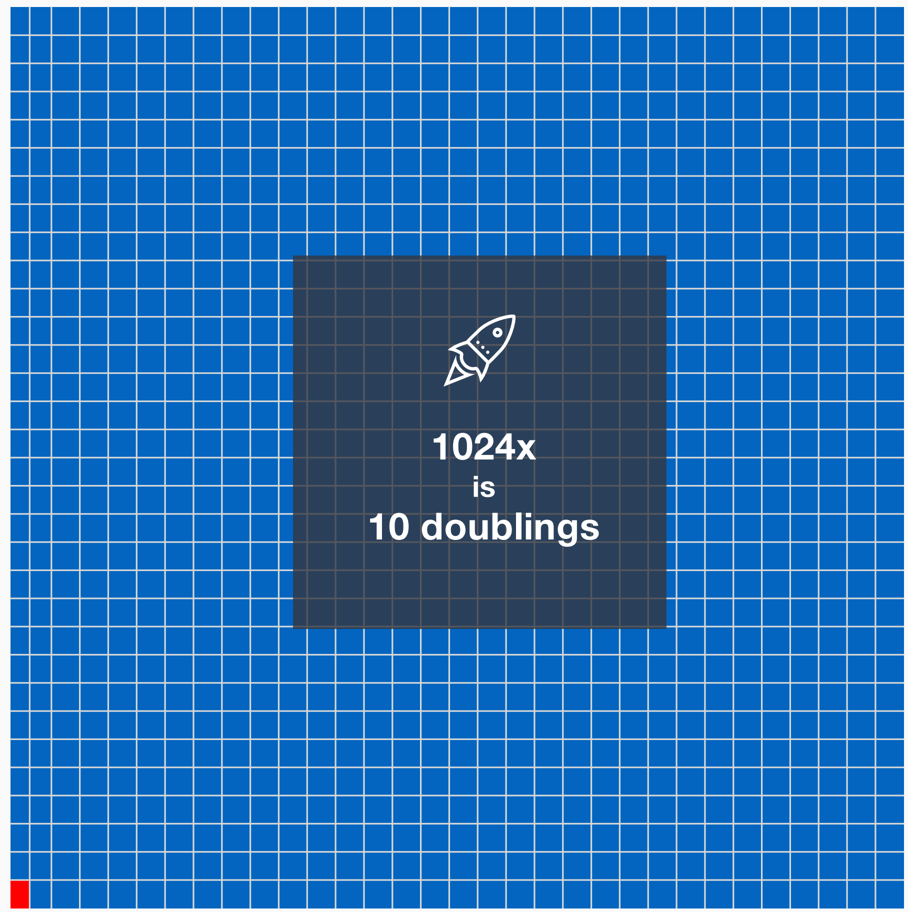
relationalAI

So **what?**

What are the implications of

2-3 orders of magnitude speed-up?



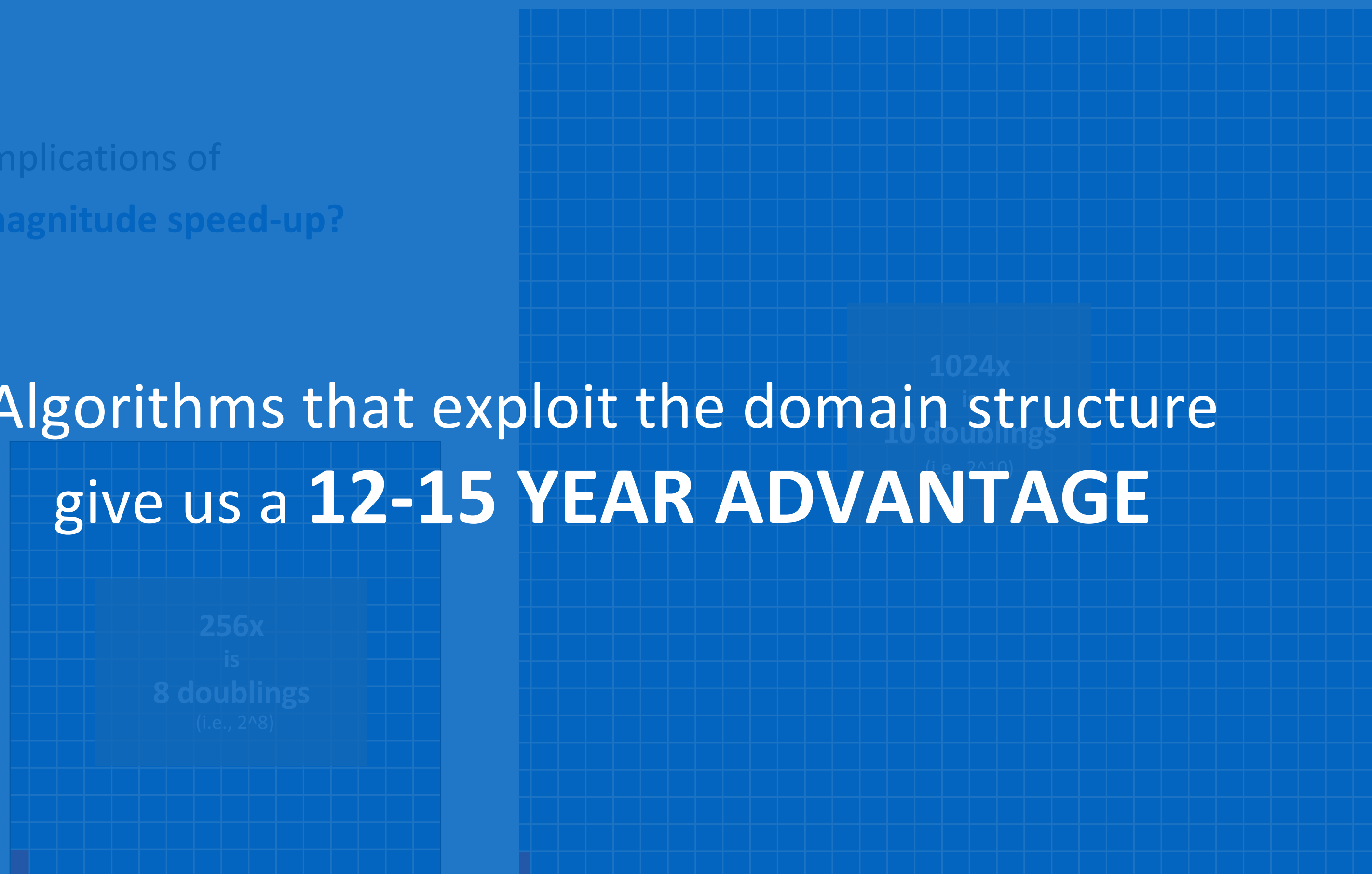


relationalAI

So **what?**

What are the implications of

2-3 orders of magnitude speed-up?



Algorithms that exploit the domain structure

give us a **12-15 YEAR ADVANTAGE**

AI's **biggest challenges are computational!**

ACCURACY

Search for better

- Parameters
- Hyper parameters
- Features
- Models

Don't make assumptions that you don't need to make (e.g. i.i.d. assumption)

VERSATILITY

Reasoning and (generalized) inference: From observations to unknowns in any time period

Inference of any property in the model (e.g., it's just as easy to infer price from sales as it is to infer sales from price)

ROBUSTNESS

Many "big data" problems are really a big collection of small data problems

Overcome challenges with small, incomplete, and dirty data problems by incorporating prior knowledge and expertise

SELF-SUPERVISION

"The future will be self-supervised" Yann LeCun

Build models of the world by observing it and searching model space for the models that have the most explanatory power

INTERPRETABILITY

Searching for models that are accurate and interpretable is harder than searching for accurate models

Interpretation in terms of prior knowledge and in language & ontology that humans understand

EXPLAINABILITY

Explainability typically implanted via separate shadow models that have to be learned

Explanation in terms of prior knowledge and in language & ontology that humans understand

FAIRNESS

It's not enough to exclude gender, ethnicity, race, age, etc as features to the models. Other features might be correlated.

Prejudice is a computational limitation: Reasoning about each person vs reasoning about the group

CAUSALITY

Understanding causality beyond A/B testing

Computationally very expensive

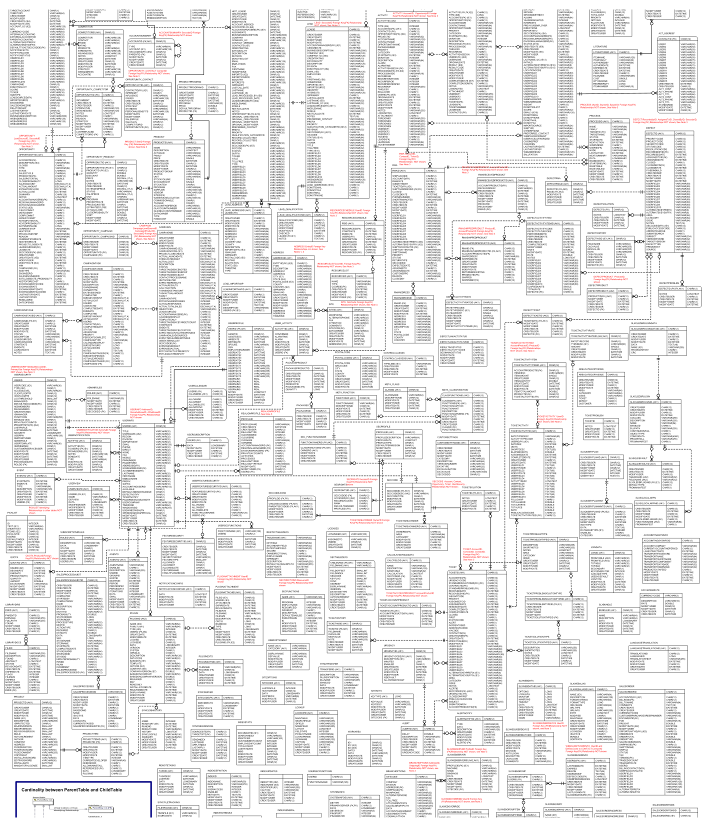
relationalAI

Statistical Relational Learning

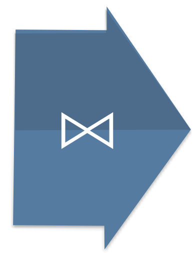
Relational generative models

relationalAI

What else **do we throw away when we build the feature matrix?**



Feature extraction query



Translation to feature matrix assumes each entity is independent of the others (iid assumption)

This is often not true - e.g. related sku's or related people

Features

Examples

ID	x1	x2	x3	…	y

relationalAI

What if we **don’t make the i.i.d assumption?**

Features

Pairs of Entities

ID	x1	x2	x3	…	y	ID	x1	x2	x3	…	y

relationalAI

What if we **don't make the i.i.d assumption?**

Features

All

ID	x1	x2	x3	...	y	ID	x1	x2	x3	...	y

...

ID	x1	x2	x3	...	y

relationalAI

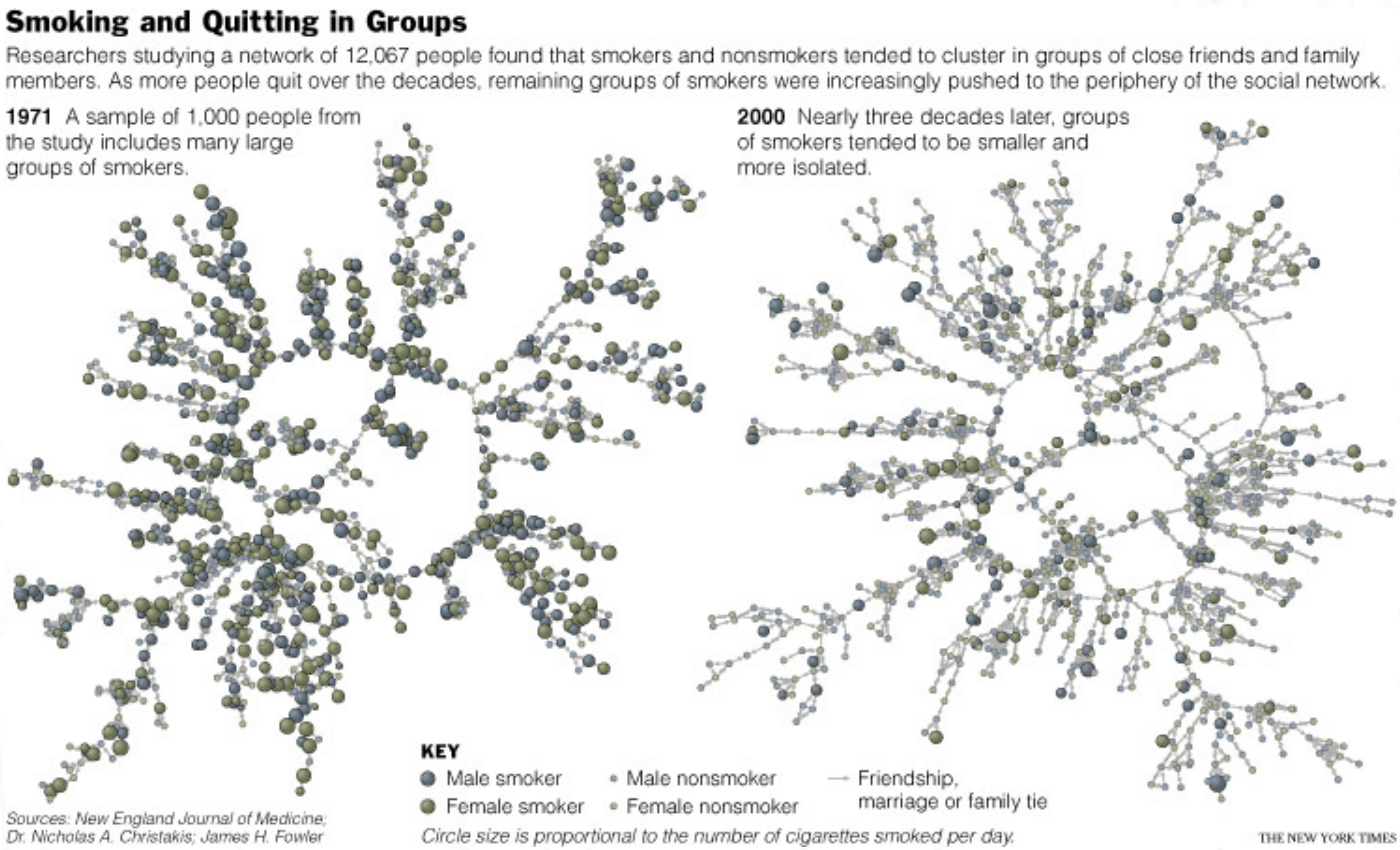
Statistical Relational Learning

- Statistical Relational models generalize PGMs in the same way that first order logic generalizes propositional logic – they allow us to quantify over individuals/entities
  - Allows for generalization (e.g. item, sub-class, class, dept, etc.)
  - Ability to predict link-based patterns (e.g. inter item dependencies at sub-class, class, dept etc.)
  - Models a varied number of observations for each object/relation. (e.g. friends, colleagues, etc.)
- Variants
  - MLN in various flavors, PSL, RDN, BoostSRL, ProbLog, etc.

relationalAI

Statistical Relational Learning

- Inference
  - Unlike “traditional” methods where prediction is the input applied to the parameters of the model class, inference in SRL requires expensive optimization or (approximate) integration over possible worlds
- Learning
  - Unlike traditional learning algorithms, just one instance to learn from (the relational DB)
  - Structure learning uses inference during each step


Smoking and Quitting in Groups
Researchers studying a network of 12,067 people found that smokers and nonsmokers tended to cluster in groups of close friends and family members. As more people quit over the decades, remaining groups of smokers were increasingly pushed to the periphery of the social network.
1971 A sample of 1,000 people from the study includes many large groups of smokers.
2000 Nearly three decades later, groups of smokers tended to be smaller and more isolated.
KEY
Male smoker
Female smoker
Male nonsmoker
Female nonsmoker
Friendship, marriage or family tie
Circle size is proportional to the number of cigarettes smoked per day.
Sources: New England Journal of Medicine; Dr. Nicholas A. Christakis; James H. Fowler
THE NEW YORK TIMES

relationalAI

CERTAIN KNOWLEDGE WITH INTEGRITY CONSTRAINTS

A logical Knowledge Base is a set of Integrity Constraints that define a set of possible worlds:

```
person(x)
smokes(x) -> person(x)
cancer(x) -> person(x)
friends(x, y) -> person(x), person(y)
```

relationalAI

CERTAIN KNOWLEDGE WITH INTEGRITY CONSTRAINTS

A logical Knowledge Base is a set of Integrity Constraints that define a set of possible worlds:

```
person(x)
smokes(x) -> person(x)
cancer(x) -> person(x)
friends(x, y) -> person(x), person(y)
```

Smoking causes cancer
Friends have similar smoking habits

relationalAI

CERTAIN KNOWLEDGE WITH INTEGRITY CONSTRAINTS

A logical Knowledge Base is a set of Integrity Constraints that define a set of possible worlds:

```
person(x)
smokes(x) -> person(x)
cancer(x) -> person(x)
friends(x, y) -> person(x), person(y)
```

Smoking causes cancer
Friends have similar smoking habits

```
w1   smokes(x) -> cancer(x)
w2   smokes(x), friends(x, y) -> smokes(y)
```

How do you make this tractable?

Approximate answer by converting into convex continuous optimization problem

Exploit group symmetry → lifted inference and approximate lifted inference

Avoid grounding altogether → in-database learning

Leveraging database semantics to avoid having to cluster -> in-database SPNs

Stay tuned

Brawn

Do same amount of work faster

relationalAI

The Path to Performance: **Brawn**

Constant factors – Do same amount of work faster (i.e., brawn)

- Latency hiding: Memory hierarchy and network latencies (e.g., in memory and near-data computing)
- Parallelization: SIMD, multi-core, accelerators (e.g., GPU, TPU, FPGA)
- Specialization: Specialize for workload (e.g., JIT compilation), specialize for data

Motivation **for implementation strategy**

and 

3 to 5 years building something similar in prior lives using C++ without ability to specialize for queries or data sets

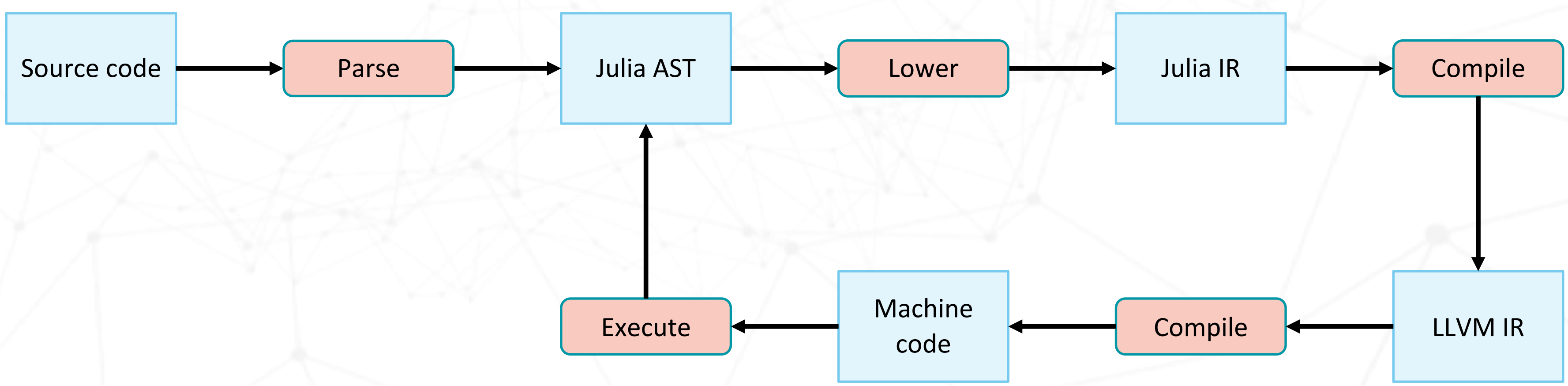
relationalAI

Julia **in a nutshell**

"Looks like Python, feels like LISP, runs like C"

Julia is fast, dynamic, optionally typed, and multi-dispatched

- Feels like Lisp: Hygienic macros, code quoting, generated functions
- Runs like C: Specialization based on type inference, inlining, unboxing, LLVM to gen assembly



relationalAI

Brains and Brawn: **Systems Programming in Julia**

- ****Specialization****
  - **Query evaluation**: Just-in-time compiled query plans

- Specialization
  - **Data types**: e.g., fixed-precision decimals

Just-in-Time **Query Compilation**

- **Query compilation has only recently replaced interpretation in modern database systems**

```
select A, B, C
from R, S, T
where …
group by …
```

→

```
pushq    %rbp
movq     %rsp, %rbp
testq    %rdi, %rdi
negq     %rdi
movq     %rdi, %rax
…
```

- **But, state of the practice is surprisingly primitive**
  - Typically: variations on template expansion in C/C++
  - Ad-hoc methods to generate code: e.g., write a text file and invoke gcc
  - Cumbersome engineering effort
- **Better: use a language with proper staged metaprogramming support**
  - e.g., LegoDB using Scala/LMS/Squid
- **Julia is very appealing from this point of view!**

relationalAI

Simplified **TPC-H Q1: from SQL to Julia to Native Code**

```
select
   sum(l_extprice * (100 - l_discount) * (100 + l_tax))
from
   lineitem
```

From SQL to Julia with runtime code generation

```
sum = 0
for i in 1:size
   sum += l_extprice[i] * (100 - l_discount[i]) * (100 + l_tax[i])
end
return sum
```

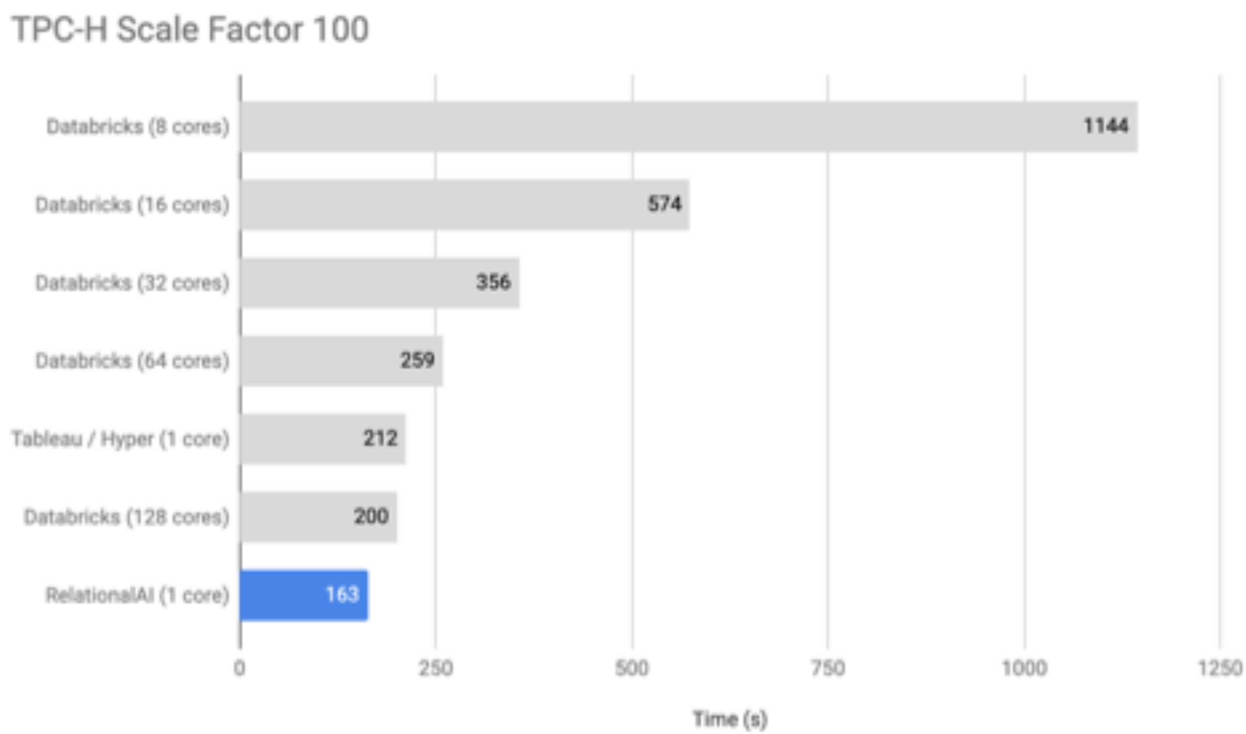
From Julia to LLVM to optimized x86-64 *

```
  testq   %rcx, %rcx
  jle     L71
  movq    (%rdi), %r8
  movq    (%rsi), %r9
  movq    (%rdx), %r10
  xorl    %edi, %edi
  xorl    %eax, %eax
L32:
  movl    $100, %esi
  subq    (%r9,%rdi,8), %rsi
  movq    (%r10,%rdi,8), %rdx
  addq    $100, %rdx
  imulq   (%r8,%rdi,8), %rsi
  imulq   %rdx, %rsi
  addq    %rsi, %rax
  addq    $1, %rdi
  cmpq    %rdi, %rcx
  jne     L32
  retq
L71:
  xorl    %eax, %eax
  retq
```

(*) The loop actually even gets vectorized, but we produced simpler code here for presentation purposes

relationalAI

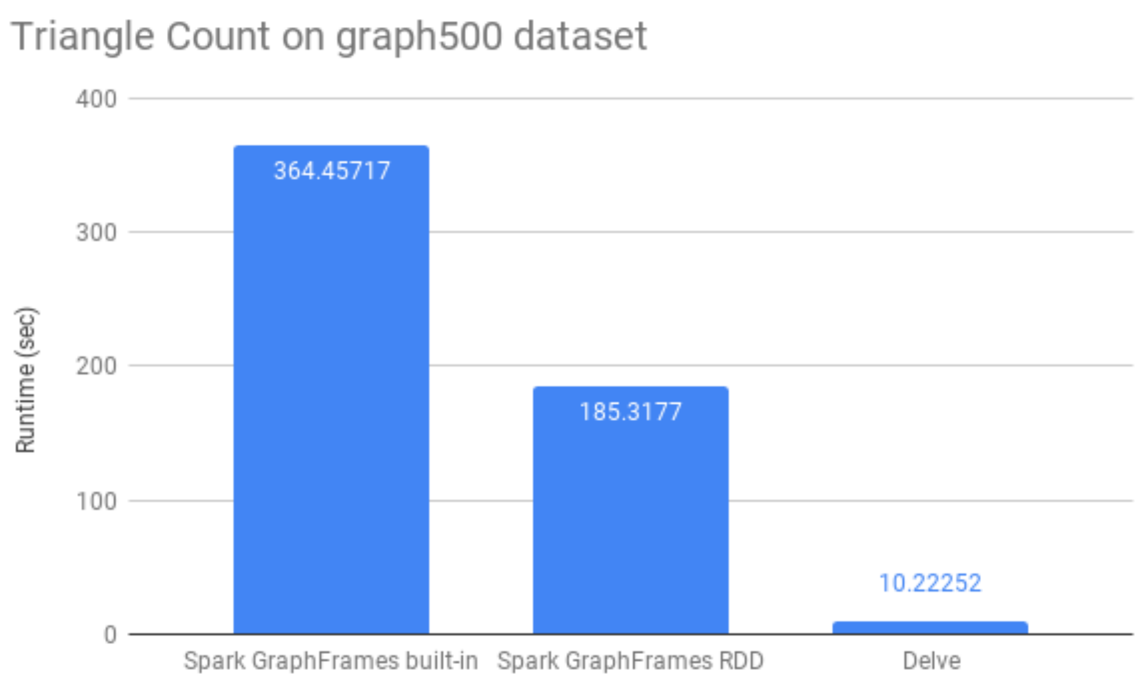
BI benchmark: **vs Tableau/Hyper and Databricks Spark**



Spark numbers based on Databricks hardware and TPCH setup. Snowflake benchmarks closer to Spark than Hyper.

relationalAI

Brains and Brawn Together: **3-Clique Graph benchmark vs Databricks Spark**



All benchmarks run on 1 core laptop.

relationalAI

Brains and Brawn: **Systems Programming in Julia**

- Specialization
  - **Query evaluation**: Just-in-time compiled query plans

- **Specialization**
  - **Data types**: e.g., fixed-precision decimals

relationalAI

Abstraction **without regret by example: Fixed-precision decimals**

Fixed-precision decimals are an important data type in database systems (e.g., for currencies), and avoid the inexact representation problems of floats:

```
julia> 0.3333 + 0.33333
0.6666300000000001    # oops
```

The Julia ecosystem has a FixedPointDecimal package for this purpose

```
julia> T = FixedDecimal{Int64,5}
FixedDecimal{Int64,5}

julia> T(0.3333) + T(0.33333)
FixedDecimal{Int64,5}(0.66663) # much better!
```

But… is this really going to be efficient enough? (Most database systems need special code to “compile away” fixed precision decimal operations into simple operations on integers…)

Here's the FixedDecimal datatype and its addition operation...

```
struct FixedDecimal{T <: Integer, f} <: Real
    i::T

    function Base.reinterpret(::Type{FixedDecimal{T, f}}, i::Integer) where {T, f}
        n = max_exp10(T)
        if f >= 0 && (n < 0 || f <= n)
            new{T, f}(i % T)
        else
            _throw_storage_error(f, T, n)
        end
    end
end

+(x::FixedDecimal{T, f}, y::FixedDecimal{T, f}) where {T, f} =
    reinterpret(FD{T, f}, x.i+y.i)
```

... and lo, the Julia compiler produces a tiny # of ops on integers, just as required!

```
julia> @code_native +(T(0.3333),T(0.33333))
decl %eax
movl (%esi), %eax
decl %eax
addl (%edi), %eax
retl
```

Moreover, this will be inlined at the call site in any practical example!

relationalAI

What about **Parallelization and Accelerators?**

» Manual » Parallel Computing Edit on GitHub

Parallel Computing

For newcomers to multi-threading and parallel computing it can be useful to first appreciate the different levels of parallelism offered by Julia. We can divide them in three main categories :

1. Julia Coroutines (Green Threading)
2. Multi-Threading
3. Multi-Core or Distributed Processing

We will first consider Julia Tasks (aka Coroutines) and other modules that rely on the Julia runtime library, that allow us to suspend and resume computations with full control of inter-`Tasks` communication without having to manually interface with the operating system's scheduler. Julia also supports communication between `Tasks` through operations like `wait` and `fetch`. Communication and data synchronization is managed through `Channel`s, which are the conduits that provide inter-`Tasks` communication.

Julia also supports experimental multi-threading, where execution is forked and an anonymous function is run across all threads. Known as the fork-join approach, parallel threads execute independently, and must ultimately be joined in Julia's main thread to allow serial execution to continue. Multi-threading is supported using the `Base.Threads` module that is still considered experimental, as Julia is not yet fully thread-safe. In particular segfaults seem to occur during I\O operations and task switching. As an up-to-date reference, keep an eye on the issue tracker. Multi-Threading should only be used if you take into consideration global variables, locks and atomics, all of which are explained later.

In the end we will present Julia's approach to distributed and parallel computing. With scientific computing in mind, Julia natively implements interfaces to distribute a process across multiple cores or machines. Also we will mention useful external packages for distributed programming like `MPI.jl` and `DistributedArrays.jl`.

High-level GPU programming in Julia

Tim Besard
Computer Systems Lab
Ghent University, Belgium
Tim.Besard@elis.ugent.be

Pieter Verstraete
Ghent University, Belgium

Bjorn De Sutter
Computer Systems Lab
Ghent University, Belgium
Bjorn.DeSutter@elis.ugent.be

Abstract

GPUs are popular devices for accelerating scientific calculations. However, as GPU code is usually written in low-level languages, it breaks the abstractions of high-level languages popular with scientific programmers. To overcome this, we present a framework for CUDA GPU programming in the high-level Julia programming language. This framework compiles Julia source code for GPU execution, and takes care of the necessary low-level interactions using modern code generation techniques to avoid run-time overhead.

Evaluating the framework and its APIs on a case study comprising the trace transform from the field of image processing, we find that the impact on performance is minimal, while greatly increasing programmer productivity. The metaprogramming capabilities of the Julia language proved invaluable for enabling this. Our framework significantly improves usability of GPUs, making them accessible for a wide range of programmers. It is available as free and open-source software licensed under the MIT License.

Categories and Subject Descriptors D.3.4 [*Programming Languages*]: Processors—Code generation, Compilers, Run-time environments

Keywords Julia, GPU, CUDA, LLVM, Metaprogramming

1. Introduction

GPUs can significantly speed up certain workloads. However, targeting GPUs requires serious effort. Specialized machine code needs to be generated through the use of a vendor-supplied compiler. Because of the architectural set-up, initiating execution on the coprocessor is often quite complex as well. Even though the vendors try hard to supply toolchains that support different developer environments and offer convenience functionality to lower the burden, they are essentially playing catch-up.

While coprocessor hardware improves program efficiency, high-level languages are becoming a popular choice because of their improved programmer productivity. Languages such as Python or Julia provide a user-friendly development environment. Low-level details are hidden from view, and secondary tasks such as dependency management and compiling and linking are automatically taken care of.

For users of these high-level languages, jumping through the many hoops of GPU development is often an exceptionally large burden. A lot of low-level knowledge is required, and many of the user-friendly abstractions break down. For example, when using Python to target NVIDIA GPUs using the CUDA toolkit, the developer needs to write GPU kernels in CUDA C, and interact with the CUDA API in order to compile the code, prepare the hardware and launch the kernel. The situation is even worse for languages unsupported by the CUDA toolkit, such as Julia, in which case there are only superficial or no CUDA API wrappers at all.

Ideally, it should be possible to develop and execute high-level GPU kernels without much extra effort: writing kernels in high-level source code, while the interpreter for that language takes care of compiling the necessary functions to GPU machine code. Low-level details should be automated, or at least wrapped in user-friendly language constructs.

This paper presents a framework to target NVIDIA GPUs, and by extent other accelerators, directly in the Julia programming language: Kernels can be written in high-level Julia code. We also created high-level CUDA API wrappers to support the natural use of the CUDA API from within Julia. The framework provides a user-friendly GPU kernel programming and execution interface that automates driver interactions and abstracts GPU-specific details without introducing any run-time overhead. All code implementing this framework is available as open-source code on GitHub.

In Section 2 we describe relevant technologies and the motivation for our work. Section 3 provides an overview of our framework, each component explained in detail in Sections 4 to 6. Finally, we evaluate our work in Section 7.

Automatic Full Compilation of Julia Programs and ML Models to Cloud TPUs

Keno Fischer [1] **Elliot Saba** [1]

Abstract

Google's Cloud TPUs are a promising new hardware architecture for machine learning workloads. They have powered many of Google's milestone machine learning achievements in recent years. Google has now made TPUs available for general use on their cloud platform and as of very recently has opened them up further to allow use by non-TensorFlow frontends. We describe a method and implementation for offloading suitable sections of Julia programs to TPUs via this new API and the Google XLA compiler. Our method is able to completely fuse the forward pass of a VGG19 model expressed as a Julia program into a single TPU executable to be offloaded to the device. Our method composes well with existing compiler-based automatic differentiation techniques on Julia code, and we are thus able to also automatically obtain the VGG19 backwards pass and similarly offload it to the TPU. Targeting TPUs using our compiler, we are able to evaluate the VGG19 forward pass on a batch of 100 images in 0.23s which compares favorably to the 52.4s required for the original model on the CPU. Our implementation is less than 1000 lines of Julia, with no TPU specific changes made to the core Julia compiler or any other Julia packages.

1 Introduction

One of the fundamental changes that has enabled the steady progress of machine learning techniques over the past several years has been the availability of vast amounts of compute power to train and optimize machine learning models. Many fundamental techniques are decades old, but only the compute power available in recent years was able to deliver sufficiently good results to be interesting for real world problems. A significant chunk of this compute power has been available on Graphics Processing Units (GPUs) whose vector compute capability, while originally intended for graphics have shown to deliver very good performance on the kind of matrix-heavy operations generally performed in machine learning models.

The real world success of these approaches and of GPUs in this space in particular has set off a flurry of activity among hardware designers to create novel accelerators for machine learning workloads. However, while GPUs have a relatively long history of support in software systems, this generally does not extend to new, non-GPU accelerators and developing software for these systems remains a challenge.

In 2017, Google announced that they would make their proprietary Tensor Processing Unit (TPU) machine learning accelerator available to the public via their cloud offering. Originally, the use of TPUs was restricted to applications written using Google's TensorFlow machine learning framework. Fortunately, in September 2018, Google opened up access to TPUs via the IR of the lower level *XLA* ("Accelerated Linear Algebra") compiler. This IR is general purpose and is an optimizing compiler for expressing arbitrary computations of linear algebra primitives and thus provides a good foundation for targeting TPUs by non-Tensorflow users as well as for non-machine learning workloads.

In this paper, we present initial work to compile general Julia code to TPU using this interface. This approach is in contrast to the approach taken by TensorFlow (Abadi et al., 2016), which does not compile Python code proper, but rather uses Python to build a computational graph, which is then compiled. It is aesthetically similar to JAX (Frostig et al., 2018), which does aim to offload computations written in Python proper by tracing and offloading high-level array operations. Crucially, however, we do not rely on tracing, instead we leverage Julia's static analysis and compilation capabilities to compile the full program, including any control flow to the device. In particular, our approach allows users to take advantage of the full expressiveness of the Julia programming language in writing their models. This includes higher-level features such as multiple dispatch, higher order functions and existing libraries such as those for differential equation solvers (Rackauckas & Nie, 2017) and generic linear algebra routines. Since it operates on pure

[1] Julia Computing, Inc.. Correspondence to: Keno Fischer <keno@juliacomputing.com>.

Preliminary work.

relationalAI

Closing

One more time

AI’s **biggest opportunities are relational!**

ACCURACY

Search for better

- Parameters
- Hyper parameters
- Features
- Models

Don’t make assumptions that you don’t need to make (e.g. i.i.d. assumption)

VERSATILITY

Reasoning and (generalized) inference: From observations to unknowns in any time period

Inference of any property in the model (e.g., it’s just as easy to infer price from sales as it is to infer sales from price)

ROBUSTNESS

Many ”big data” problems are really a big collection of small data problems

Overcome challenges with small, incomplete, and dirty data problems by incorporating prior knowledge and expertise

SELF-SUPERVISION

“The future will be self-supervised” Yann LeCun

Build models of the world by observing it and searching model space for the models that have the most explanatory power

INTERPRETABILITY

Searching for models that are accurate and interpretable is harder than searching for accurate models

Interpretation in terms of prior knowledge and in language & ontology that humans understand

EXPLAINABILITY

Explainability typically implanted via separate shadow models that have to be learned

Explanation in terms of prior knowledge and in language & ontology that humans understand

FAIRNESS

It’s not enough to exclude gender, ethnicity, race, age, etc as features to the models. Other features might be correlated.

Prejudice is a computational limitation: Reasoning about each person vs reasoning about the group

CAUSALITY

Understanding causality beyond A/B testing

Computationally very expensive

relationalAI

Why **hasn't this happened yet?**

AI investment is focused on consumer AI

- Deep learning for images, speech, text → not relational data (yet)

Weaknesses of implementations of relational data management systems

- Abstraction leads to regret
- Can guarantee correct answer but can't guarantee optimal path to get there
- Limitations on expressiveness, i.e. I can't always ask the question I want to ask

Inertia — we have something that (sort of) works and we're getting by. "you can't expect us to rewrite all this code and retrain all those data scientists and programmers"

- The number of models that haven't been built is **>>>** the number of models that have
- The number of future modelers is **>>>** the number of current modelers
- The number of domain experts is **>>>** the number of modelers and data scientists

relationalAI

Why **Now?**

- We invented a new generation of (meta) algorithms that provide optimal solutions to large problem classes
  - OOM **more power** for OOM **better intelligence**

- New generation of compilers that eliminate the cost of abstraction
  - Allow us to specialize for workload
  - Allow us to specialize for datasets

- Backlash against Hadoop (Map-Reduce), NoSQL, ML Frameworks – “the emperor has no clothes” is in the air
  - Require you to sell your soul for scalability and/or performance
  - Harder to program and operate

relationalAI

What **are we doing about it?**

We built a system that gives you abstraction without regret

How are we going to do that?

- Constant factors
- Asymptotic factors

We're going to meet people where they are:

- Tables and SQL if you are an analyst
- Tensors & Linear Algebra if you are a data scientist

We're going to simplify and consolidate analytics:

- The building blocks for next gen AI (e.g. fast aggregation, factoring, multi-way evaluation, JIT, accelerators) building blocks for all enterprise analytics: BI, graphs, rules, planning, mathematical optimization.

We're going to stage it. We're going to consolidate and checkpoint our gains as we go.

- AutoML (with automatic feature engineering and relational statistics) -> Data scientist
- Data Management Systems for Analytics (aka data lakes) -> Data scientist
- Business Intelligence & Data Warehouses -> Analyst & End User

Product: **Never have to start from scratch again**

Data

- General: e.g. Weather, Events, Consumer, Sentiment
- Domain and industry specific: e.g. securities, crypto currencies
- Competitor: e.g. price

Templates

- Industry: retail, financial services, technology & software.
- Problem class: (product) knowledge graphs, recommender systems, anomaly detection, portfolio optimization

Tools

- Data scientists: Notebooks (e.g. Jupyter)
- Domain modelers: e.g. ontology editors (e.g. Jupyter, NORMA, Protégé)
- Analysts: e.g. BI and spreadsheets

Engine

- Database
- AI and Analytics

relationalAI

References

Incomplete list

relationalAI

Underlying magic: **Worst-case optimal join algorithms**

- Worst-Case Optimal Join Algorithms: Techniques, Results, and Open Problems. Ngo. (Gems of PODS 2018)
- Worst-Case Optimal Join Algorithms: Techniques, Results, and Open Problems. Ngo, Porat, Re, Rudra. (Journal of the ACM 2018)
- What do Shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? Abo Khamis, Ngo, Suciu, (PODS 2017 - Invited to Journal of ACM)
- Computing Join Queries with Functional Dependencies. Abo Khamis, Ngo, Suciu. (PODS 2017)
- Joins via Geometric Resolutions: Worst-case and Beyond. Abo Khamis, Ngo, Re, Rudra. (PODS 2015, Invited to TODS 2015)
- Beyond Worst-Case Analysis for Joins with Minesweeper. Abo Khamis, Ngo, Re, Rudra. (PODS 2014)
- Leapfrog Triejoin: A Simple Worst-Case Optimal Join Algorithm. Veldhuizen (ICDT 2014 - Best Newcomer)
- Skew Strikes Back: New Developments in the Theory of Join Algorithms. Ngo, Re, Rudra. (Invited to SIGMOD Record 2013)
- Worst Case Optimal Join Algorithms. Ngo, Porat, Re, Rudra. (PODS 2012 – Best Paper)

Underlying magic: **Optimal query plans for worst-case optimal joins**

- Juggling functions inside a database, Abo Khamis, Ngo, Suciu (Invited to SIGMOD Record)
- On Functional Aggregate Queries with Additive Inequalities. Abo Khamis, Curtin, Moseley, Ngo, Nguyen, Olteanu, Schleich. PODS 2019
- What do Shannon-type Inequalities, Submodular Width, and Disjunctive Datalog have to do with one another? Abo Khamis, Ngo, Suciu, (PODS 2017 - Invited to Journal of ACM)
- FAQ: Questions Asked Frequently, Abu Khamis, Ngo, Rudra, (PODS 2016 – Best Paper, Invited to Journal of ACM)


FAQ: Questions Asked Frequently

Juggling Functions Inside a Database

What do Shannon-type Inequalities, Submodular Width, and Disjunctive Datalog have to do with one another?

ABSTRACT

1. INTRODUCTION

Underlying magic: **In-database relational learning**

- Rk-means: Fast Clustering for Relational Data. Curtin, Moseley, Ngo, Nguyen, Olteanu, Schleich. Submitted to NeurIPS 2019
- On coresets for logistic regression. Curtin, Moseley, Pruhs, Samadian. Submitted to NeurIPS 2019
- SolverBlox: Algebraic Modeling in Datalog. Borraz-Sanchez, Klabjan, Pasalic, Aref. (Declarative Logic Programming – Morgan & Claypool 2018)
- In-Database Learning with Sparse Tensors, Abo Khamis, Ngo, Nguyen, Olteanu, Schleich (PODS 2018 - Invited to Journal of TODS)
- AC/DC: In-Database Learning Thunderstruck, Abo Khamis, Ngo, Nguyen, Olteanu, Schleich (DEEM 2018)
- Modelling Machine Learning Algorithms on Relational Data with Datalog. Makrynioti, Vasiloglou, Pasalic, Vassalos. (DEEM 2018)
- In-Database Factorized Learning, Ngo, Nguyen, Olteanu, Schleich (AMW 2017)
- Data Science with Linear Programming. Makrynioti, Vasiloglou, Pasalic, Vassalos. (DeLBP 2017)

relationalAI

Underlying magic: **Julia**

- Julia: Dynamism and Performance Reconciled by Design, Jeff Bezanson, Jiahao Chen, Ben Chung, Stefan Karpinski, Viral B. Shah, Lionel Zoubritzky, Jan Vitek (OOPSLA 2018)
- Julia Subtyping: A Rational Reconstruction, Francesco Zappa Nardelli, Julia Belyakova, Artem Pelenitsyn, Benjamin Chung, Jeff Bezanson, Jan Vitek (OOPSLA 2018)
- Julia: A fresh approach to numerical computing, Jeff Bezanson, Alan Edelman, Stefan Karpinski, Viral B. Shah (SIAM Review 2017)

relationalAI

THANK YOU