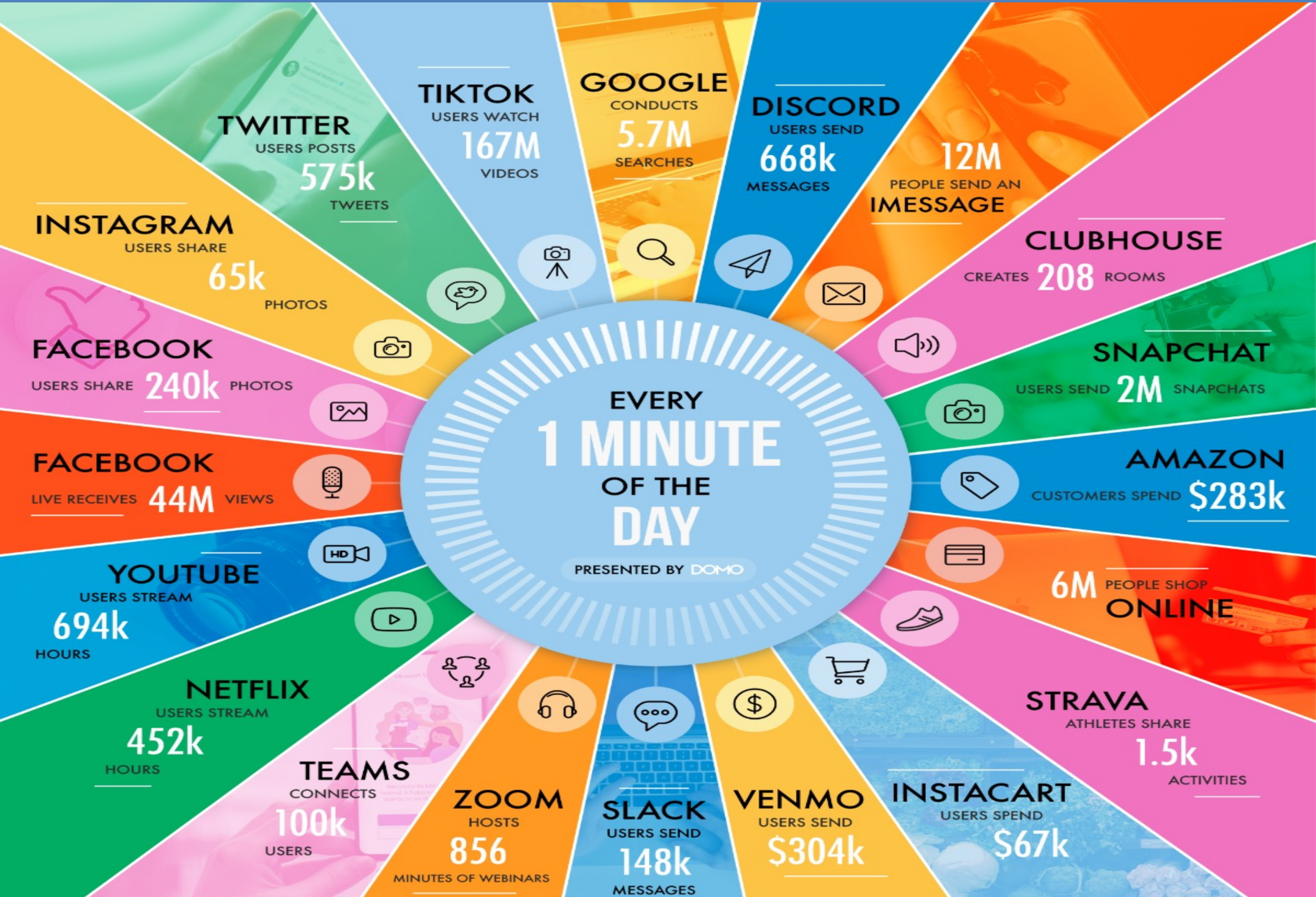
Sparse Tensor Algebra Compilation using Equality Saturation

Amir Shaikhha
joint work with Mathieu Huot, Shideh Hashemian,
Dan Olteanu, Jaclyn Smith, Dan Suciu, Max Schleich

2024/4/18



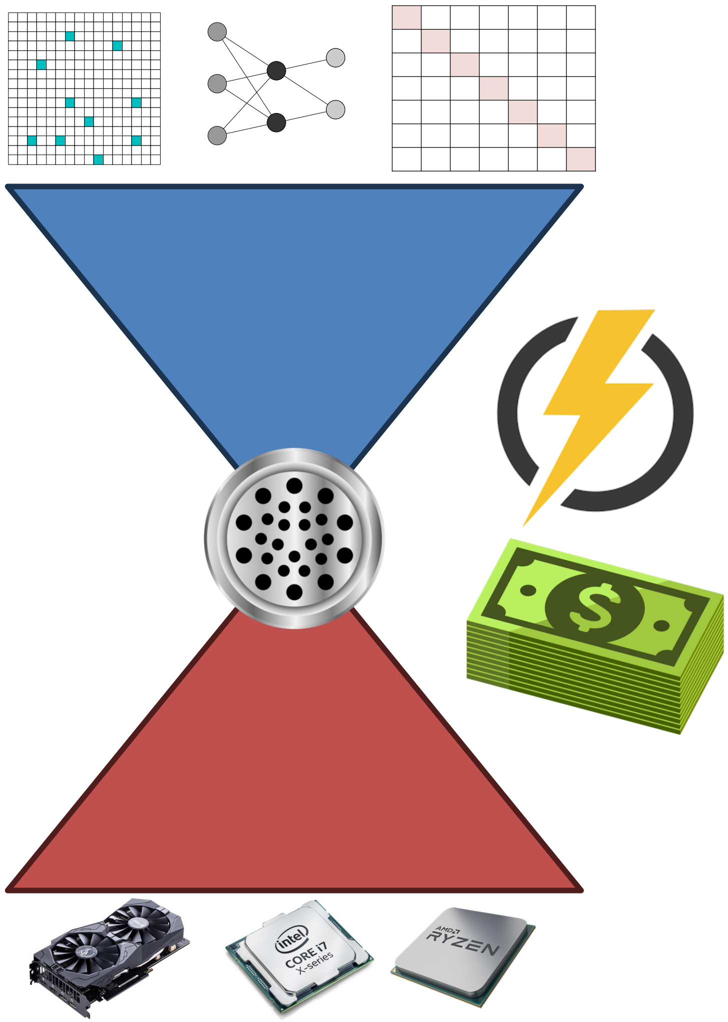
[https://www.visualcapitalist.com/from-amazon-to-zoom-what-happens-in-an-internet-minute-in-2021]

Data Processing

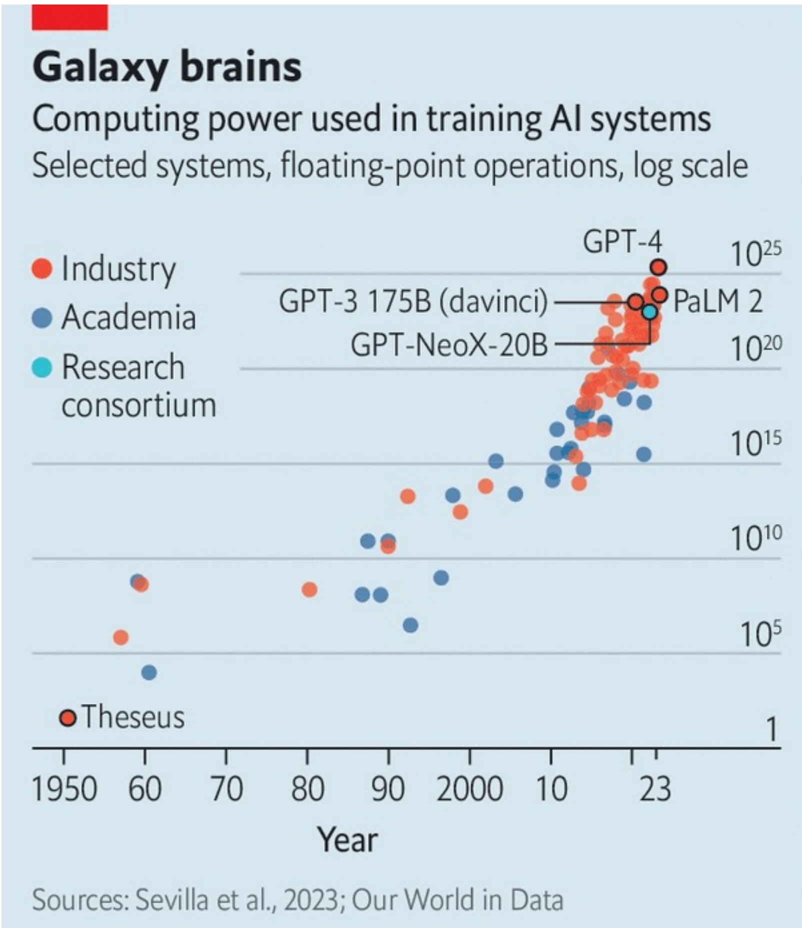
Data

Program

Hardware



Planet and Economic Crisis



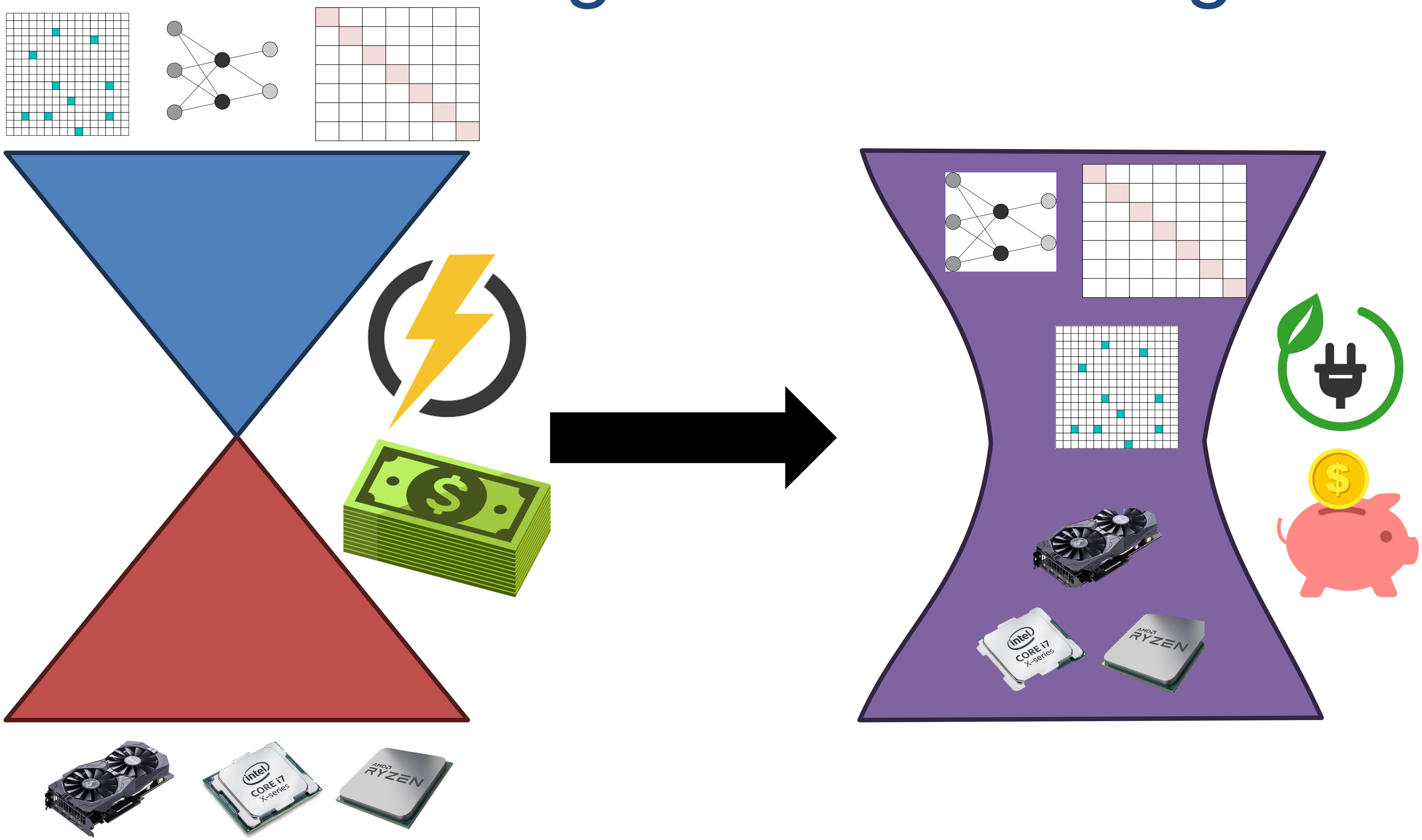
The Economist

GPT-4

5 years of 3K UK households

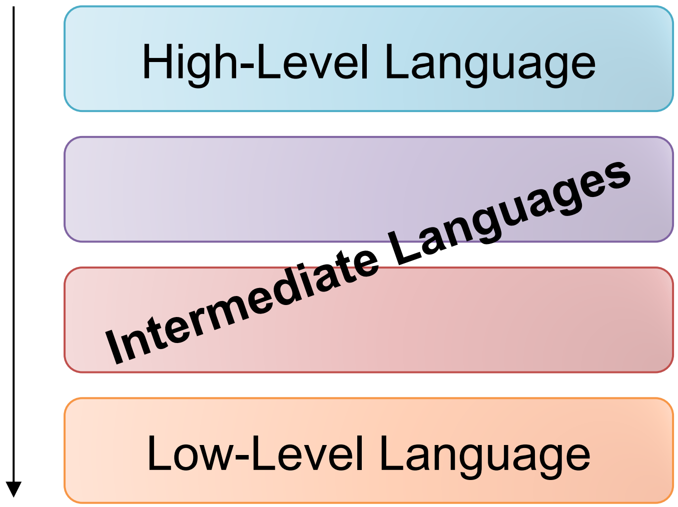
80M £

Revolutionalizing Data Processing



Language Design

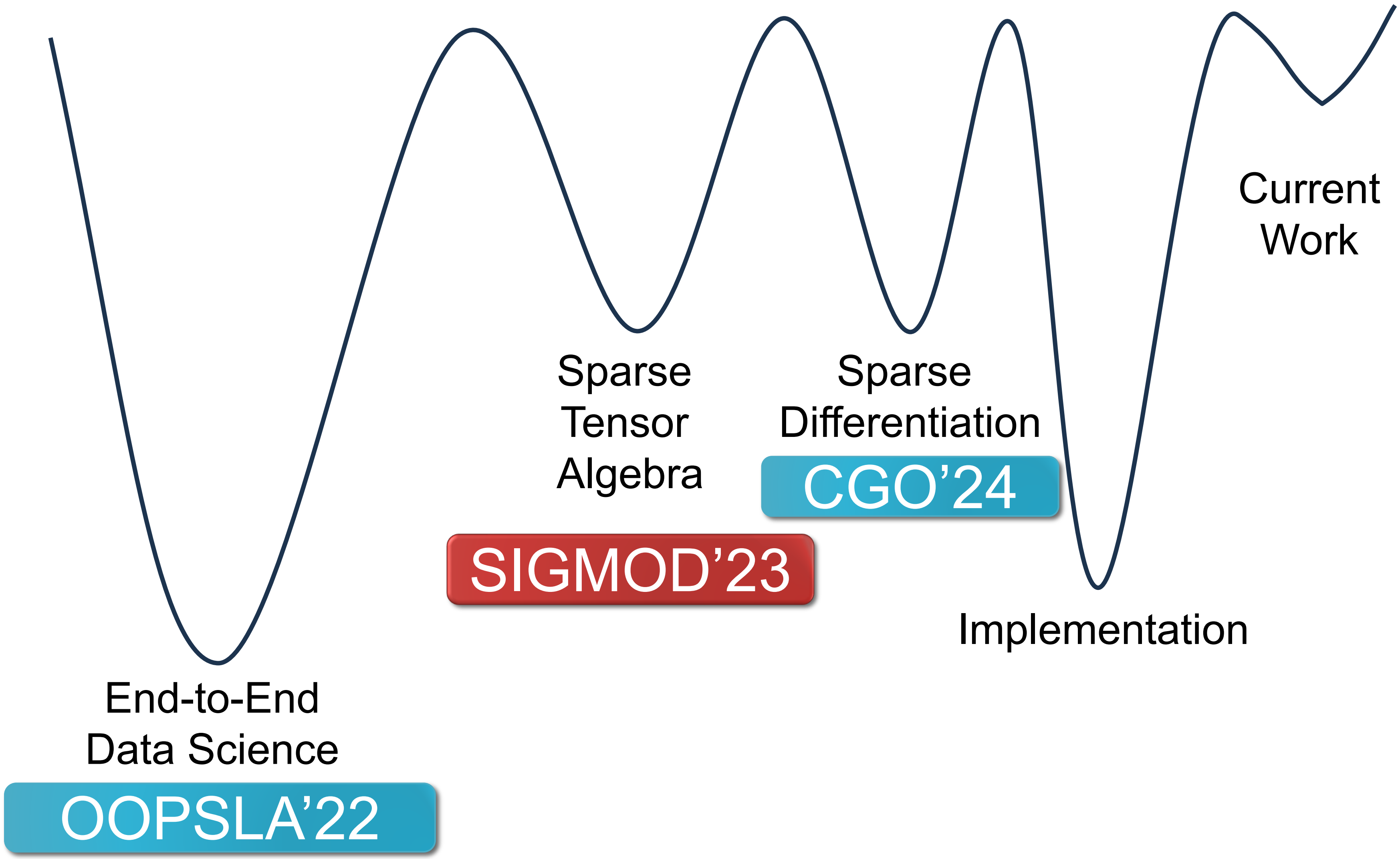
- Domain-Specific Languages (DSLs)
  - Languages for a particular domain
- Exploit
  - Domain knowledge
  - Algebraic structure
  - Structure of data
  - Algorithmic knowledge
  - Specialized data-structures



Databases

Programming Languages

Outline


End-to-End
Data Science
OOPSLA'22
Sparse
Tensor
Algebra
SIGMOD'23
Sparse
Differentiation
CGO'24
Implementation
Current
Work

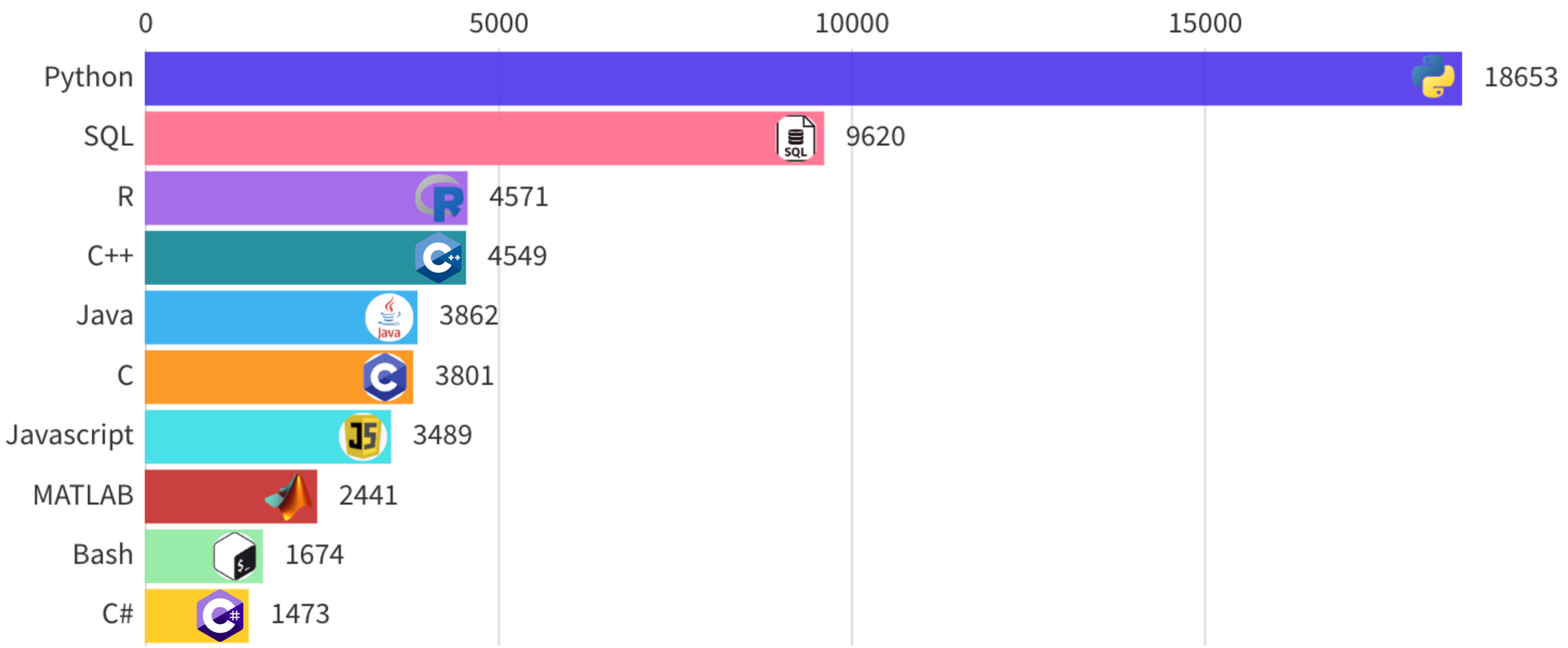
END-TO-END DATA SCIENCE

Functional Collection Programming with Semi-ring Dictionaries

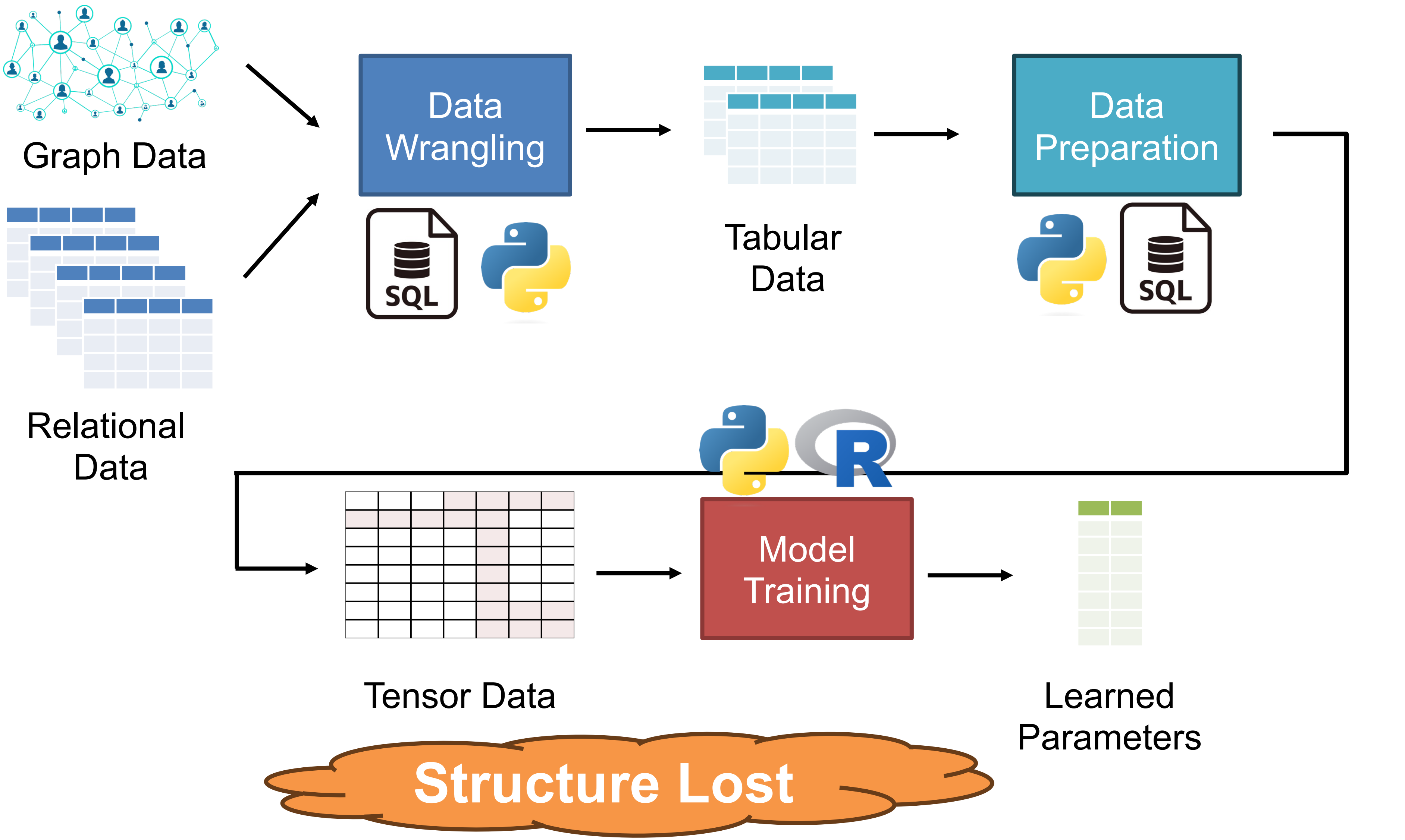
AMIR SHAIKHHA, University of Edinburgh, United Kingdom
MATHIEU HUOT, University of Oxford, United Kingdom
JACLYN SMITH, University of Oxford, United Kingdom
DAN OLTEANU, University of Zurich, Switzerland

OOPSLA'22

Data Science PLs

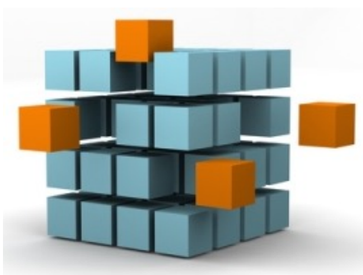


End-to-End Data Science



Data Science Workloads

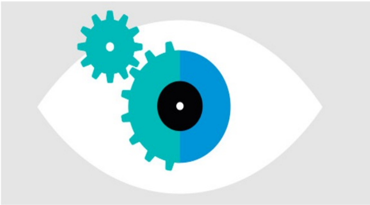
DB Workloads



Data Warehouses (OLAP)

LA Workloads

Machine Learning

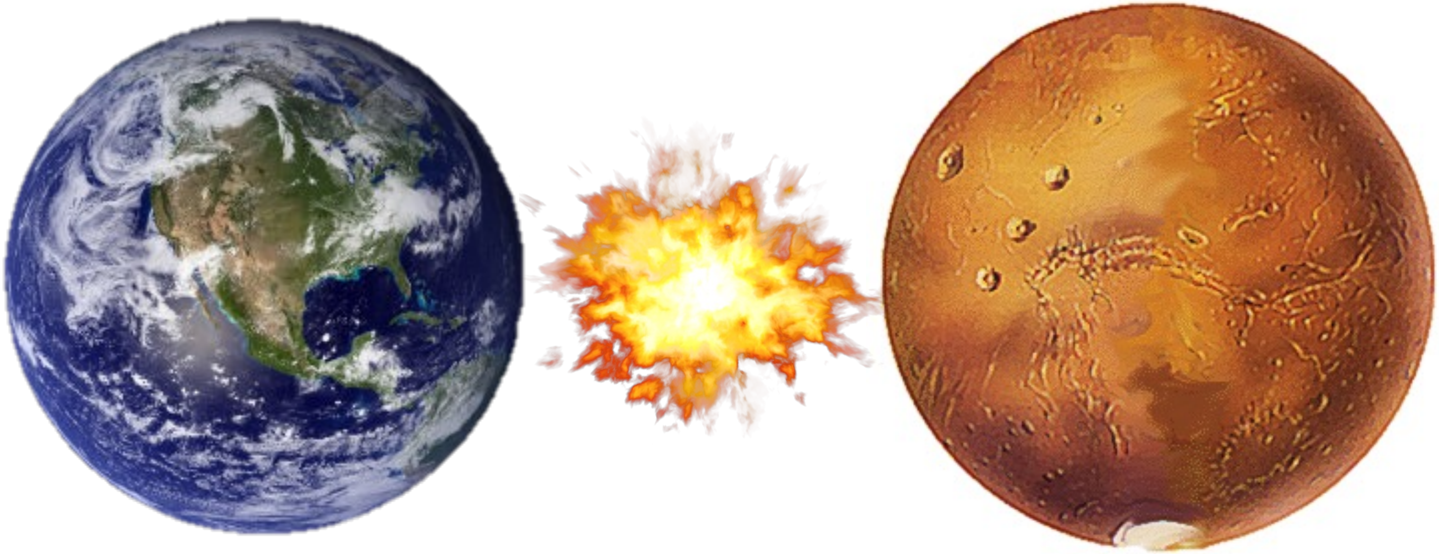


Computer Vision

Scientific Computing

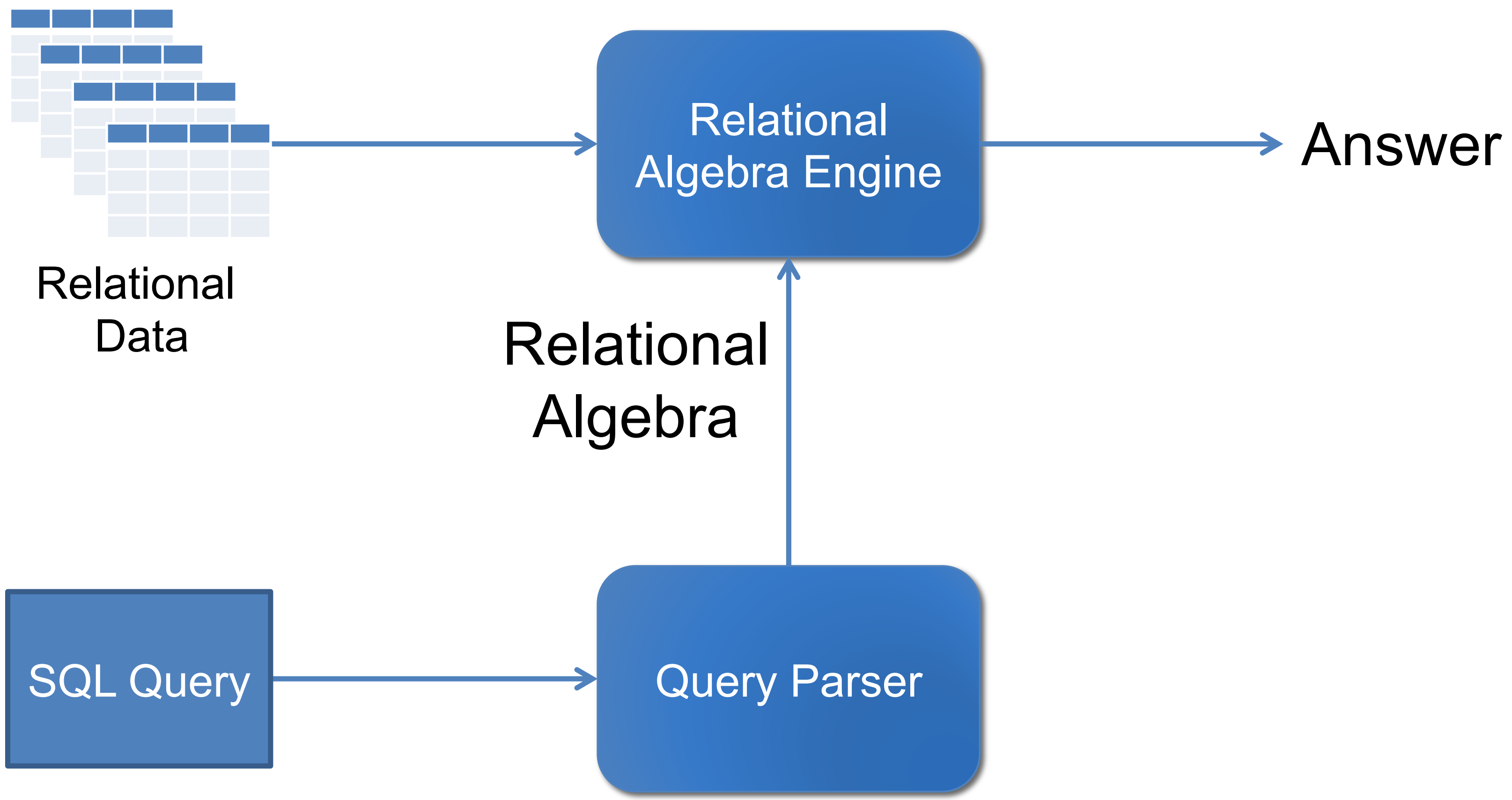
Graph Processing

Data Science Workloads



DB Workloads	LA Workloads
Relational Algebra Nested Relational Algebra	Linear Algebra Tensor Algebra
RDBMS, Pandas DataFrame	TensorFlow, PyTorch, scipy

Relational DB



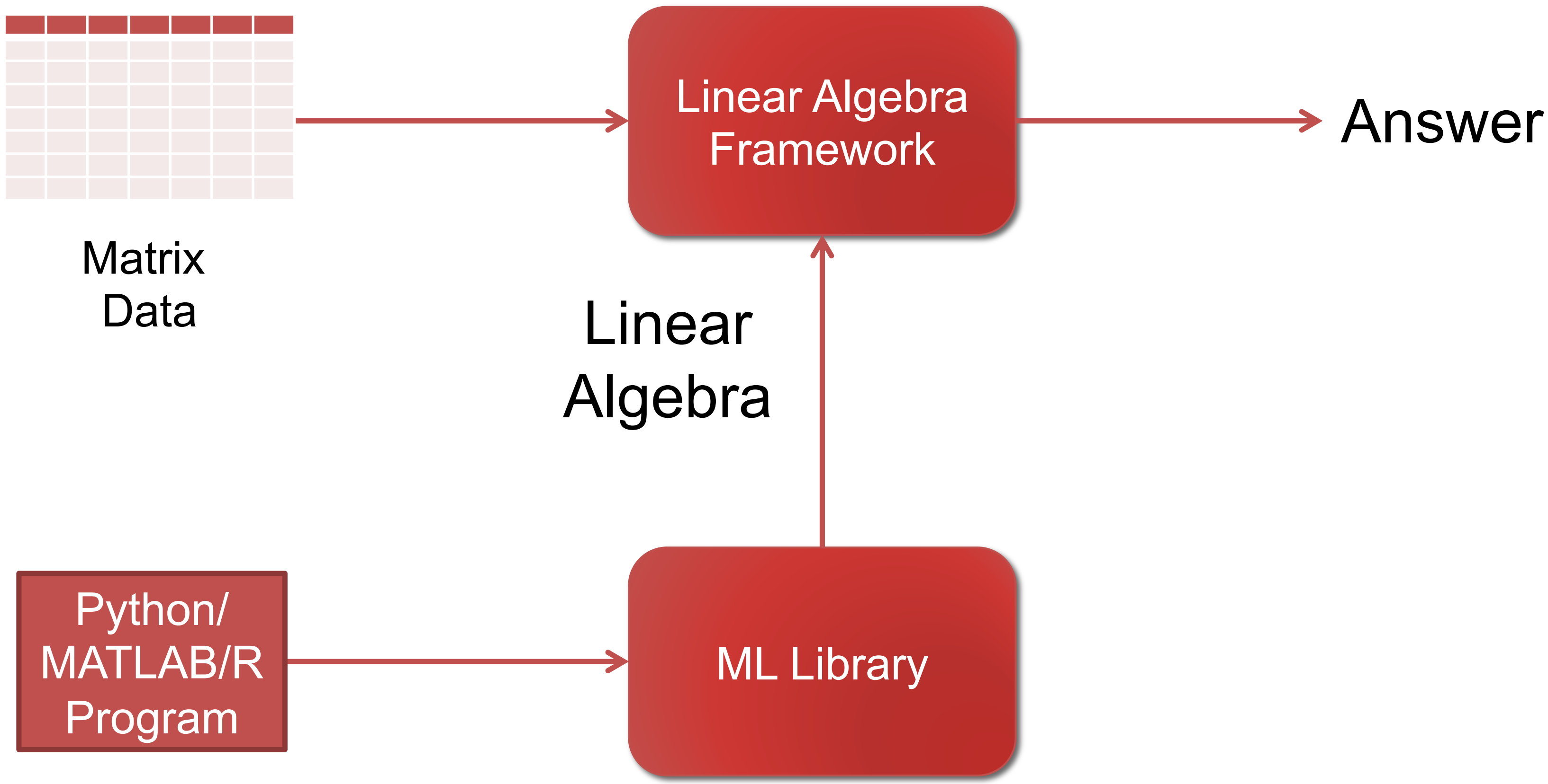
Relational Algebra

- An algebra for relational databases
- Selection (σ)
  - Filters out all tuples that do not satisfy a predicate
- Projection (π)
  - Filters out unnecessary columns of a relation
- Join (⋈)
  - Combines the tuples of two relations
  - A complex operator
- Group-By Aggregation (Γ)
  - Partitions data and aggregates!
  - Another complex operator

Relational Algebra Optimizations

- $\sigma_{c1}(\sigma_{c2}(R)) = \sigma_{c2}(\sigma_{c1}(R))$
- $\sigma_{c1\wedge \ldots \wedge cn}(R) = \sigma_{c1}(\ldots(\sigma_{cn}(R))\ldots)$
- $\pi_{a1}(R) = \pi_{a1}(\ldots(\pi_{an}(R))\ldots)$
- $R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$
- $R \bowtie S = S \bowtie R$
- $\sigma_{c1\wedge \ldots \wedge cn}(R \bowtie S) = \sigma_{c1\wedge \ldots \wedge ck}(R) \bowtie \sigma_{cp\wedge \ldots \wedge cn}(S)$
- …

ML Frameworks



Linear Algebra Optimizations

- M1 + M2 = M2 + M1
- M1 + 0 = 0 + M1 = M1
- M × I = I × M = M
- M × 0 = 0 × M = 0
- M1 × (M2 × M3) = (M1 × M2) × M3
- M1 × (M2 + M3) = M1 × M2 + M1 × M3
- …

Can we have a unified environment?

Similarity of Optimizations

$$Q(a,d) = \Gamma^{\#}_{a,d} R_1(a,b) \bowtie R_2(b,c) \bowtie R_3(c,d)$$

$$N(i,l) = \Sigma_{j,k} M_1(i,j) \cdot M_2(j,k) \cdot M_3(k,l)$$

$$Q'(a,c) = \Gamma^{\#}_{a,c} R_1(a,b) \bowtie R_2(b,c) \qquad Q(a,d) = \Gamma^{\#}_{a,d} Q'(a,c) \bowtie R_3(c,d)$$

$$N'(i,k) = \Sigma_j M_1(i,j) \cdot M_2(j,k) \qquad N(i,k) = \Sigma_k N'(i,k) \cdot M_3(k,l)$$

Pushing aggregates past joins

Matrix chain ordering

SDQL

Semi-Ring Dictionary Query Language

Semi-Ring

$\langle R, 0, 1, +, \times \rangle$

$\forall a, b, c \in R$

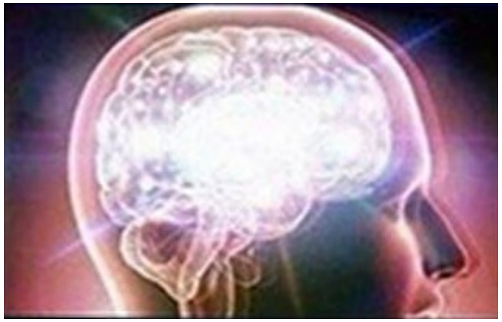
- $a+0=a$
- $a+b=b+a$
- $(a + b) + c = a + (b + c)$
- $a\times 1=1\times a=a$
- $a\times 0=0\times a=0$
- $(a \times b) \times c = a \times (b \times c)$
- $a \times (b + c) = (a \times b) + (a \times c)$
- $(a + b) \times c = (a \times c) + (b \times c)$

Factorization

Semi-Ring Examples

<ℝ, 0, 1, +, ×>
<ℕ, 0, 1, +, ×>

<{false,true},
false,true,∨,∧ >



<ℝ∪{+∞},
+∞, 0, min, +>

Semi-Ring Dictionaries

One collection to rule them all

```
{ key -> value }
```

```
Relation[T] = { T -> Bool }   (no duplicates)
Relation[T] = { T -> Int }    (with duplicates)
```

```
Vector[T] = {     Int      -> T }
Matrix[T] = { (Int, Int)   -> T }
```

Database Relations (Bag Semantics)

Relation R(A,B)

A	B
a_1	b_1
a_1	b_1
a_2	b_1
a_2	b_1
a_2	b_2

A	B	$\to$	$R(A, B)$
a_1	b_1	$\to$	2
a_2	b_1	$\to$	2
a_2	b_2	$\to$	1

`{ tuple -> multiplicity }`

Linear Algebra (Matrix)

Matrix M

	0	1	2
0	m_1	0	0
1	0	0	m_2
2	0	0	0
3	0	m_3	0

row	col	$\rightarrow$	$M_{row,col}$
0	0	$\rightarrow$	m_1
1	2	$\rightarrow$	m_2
3	1	$\rightarrow$	m_3

```
{ index -> value }
```

SDQL Examples

SDQL	C++
`sum(<key, val> in R) f(key, val)`	`double res = 0; for(auto&e : R) { res += f(e.key, e.val) }`
`sum(<key, val> in R) { g(key) -> f(val) }`	`dict<K,V> res = dict<K,V>(); for(auto&e : R) { res[g(e.key)] += f(e.val) }`

SDQL

```
sum(<key, val> in R)
   f(key, val)
```

C++

```
double res = 0;
for(auto&e : R) {
    res += f(e.key, e.val)
}
```

SDQL

```
sum(<key, val> in R)
   { g(key) -> f(val) }
```

C++

```
dict<K,V> res = dict<K,V>();
for(auto&e : R) {
    res[g(e.key)] += f(e.val)
}
```

Aggregations over Relations (Bag)

```
SELECT COUNT(*) FROM R
```

```
sum(<key, val> in R) val
```

```
SELECT SUM(A) FROM R
```

```
sum(<key, val> in R) key.A * val
```

```
SELECT B, SUM(A) FROM R GROUP BY B
```

```
sum(<key, val> in R) { key.B -> key.A * val }
```

Relational Algebra to SDQL

$[[\sigma_p(R)]]$ = `sum(x <- [[R]] )if(p(x.key)){ x.key } else { }`

$[[\pi_f(R)]]$ = `sum(x <- [[R]] ){ f(x.key)}`

$[[R \cup S]]$ = $[[R]] + [[S]]$

$[[R \cap S]]$ = `sum(x <- [[R]] )if([[S]](x.key)){ x.key } else { }`

$[[R - S]]$ = `sum(x <- [[R]] )if([[S]](x.key)){ } else { x.key }`

$[[R \times S]]$ = `sum(x <- [[R]] )sum(y <- [[S]] )`
`{ concat(x.key, y.key)}`

$[[R \bowtie_\theta S]]$ = $[[\sigma_\theta(R \times S)]]$

$[[\Gamma_{\emptyset;f}(e)]]$ = `sum(x <-[[e]])x.val*[[f]](x.key)`

$[[\Gamma_{g;f}(e)]]$ = `let tmp=sum(x <-[[e]]){[[g]](x.key)->x.val*[[f]](x.key)}`
`in sum(x <- tmp){ <key=x.key, val=x.val> -> 1 }`

Vector Operations

```
V1 + V2
```

```
V1 + V2
```

```
V1 .* V2
```

```
sum(<key, val> in V1) { key -> val * V2(key) }
```

```
V1 . V2
```

```
sum(<key, val> in V1) val * V2(key)
```

Linear Algebra to SDQL

$$
\begin{aligned}
[[V_1 + V_2]] &= [[V_1]] + [[V_2]] \\
[[a \cdot V]] &= [[a]] * [[V]] \\
[[V_1 \circ V_2]] &= \texttt{sum(x in } [[V_1]] \texttt{)\{ x.key -> x.val * } [[V_2]]\texttt{(x.key)\}} \\
[[V_1 \cdot V_2]] &= \texttt{sum(x in } [[V_1]] \texttt{)x.val * } [[V_2]]\texttt{(x.key)} \\
[[\textstyle\sum_{a \in V} a]] &= \texttt{sum(x in } [[V]] \texttt{)x.val}
\end{aligned}
$$

$$
\begin{aligned}
[[M_1^T]] &= \texttt{sum(row in } [[M_1]] \texttt{)sum(x in row.val)} \\
&\quad \texttt{\{ x.key -> \{ row.key -> x.val \} \}} \\
[[M_1 \circ M_2]] &= \texttt{sum(row in } [[M_1]] \texttt{)\{ row.key ->} \\
&\quad \texttt{sum(x in row.val)\{ x.key ->} \\
&\quad\quad \texttt{x.val*}[[M_2]]\texttt{(row.key)(x.key)\} \}} \\
[[M_1 \times M_2]] &= \texttt{sum(row in } [[M_1]] \texttt{)\{ row.key ->} \\
&\quad \texttt{sum(x in row.val)sum(y in } [[M_2]]\texttt{(x.key))} \\
&\quad\quad \texttt{\{ y.key -> x.val * y.val \} \}} \\
[[M \cdot V]] &= \texttt{sum(row in } [[M]] \texttt{)\{ row.key ->} \\
&\quad \texttt{sum(x in row.val)x.val * } [[V]]\texttt{(x.key)\}} \\
[[Trace(M)]] &= \texttt{sum(row in } [[M]] \texttt{)row.val(r.key)}
\end{aligned}
$$

Loop Optimizations

- Vertical Loop Fusion
- Horizontal Loop Fusion
- Loop-Invariant Code Motion (Hoisting)
- Loop Factorization
- Loop Memoization

Loop Memoization & Hoisting

```
sum(<r,r_v> in R)
  sum(<s,s_v> in S)
   if(jkR(r)==jkS(s)) then
   { concat(r,s)->r_v*s_v }
```

```
sum(<r,r_v> in R)
 let Sp = sum(<s,s_v> in S)
  { jkS(s) -> {s->s_v} } in
 sum(<s,s_v> in Sp(jkR(r)))
  { concat(r,s)->r_v*s_v }
```

```
let Sp = sum(<s,s_v> in S)
 { jkS(s) -> {s->s_v} } in
sum(<r,r_v> in R)
 sum(<s,s_v> in Sp(jkR(r)))
  { concat(r,s)->r_v*s_v }
```

Nested Loop Join -> Hash Join

Uniform Optimization

- Vertical Loop Fusion

Pipeline Query Engine | Deforestation, Pull/Push Arrays

- Horizontal Loop Fusion

Multi-aggregate Operator | Horizontal Fusion

- Loop Factorization + Memoization

Hash Join, Group Join | Matrix chain ordering

Data Layouts

- Relations
  - Row/Columnar layout
  - Standard Dictionary
  - Factorized

- Tensors
  - Dense (Row/Col Major)
  - COO
  - Compressed

Dictionary

<A=a_1, B=b_1>	1
<A=a_1, B=b_2>	1
<A=a_2, B=b_3>	1

Factorized

a_1	b_1	1
	b_2	1
a_2	b_3	1

Row

0	<A=a_1, B=b_1>
1	<A=a_1, B=b_2>
2	<A=a_2, B=b_3>

Columnar

<A=

0	a_1
1	a_1
2	a_2

, B=

0	b_1
1	b_2
2	b_3

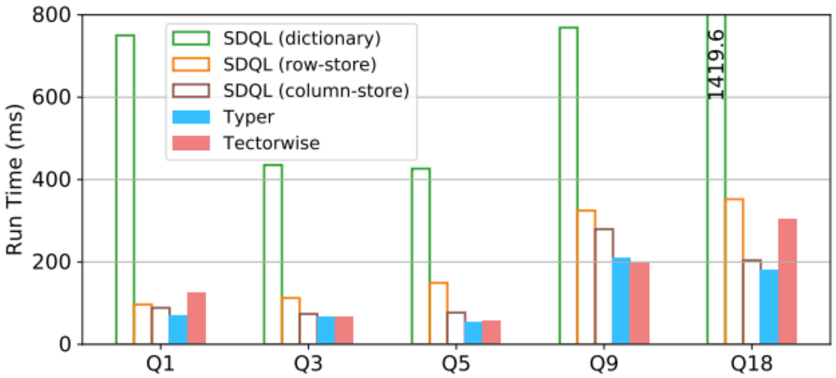
>

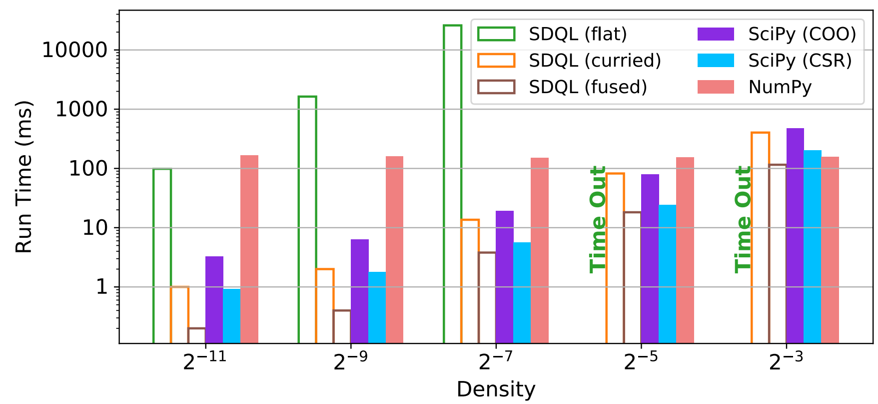
Semi-Ring Dictionaries

One collection to rule them all

- Relations
  - `Bag{T}` = `Dict{T, Int}`
  - `Set{T}` = `Dict{T, Bool}`
- Nested Relations
  - `Bag{Bag{T}}` = `Dict{Dict{T, Int}, Int}`
  - `Set{Set{T}}` = `Dict{Dict{T, Bool}, Bool}`
- Tensors
  - `SparseVector{T}` = `Dict{Int, T}`
  - `SparseMatrixCOO{T}` = `Dict{(Int, Int), T}`
  - `SparseMatrixTrie{T}` = `Dict{Int, Dict{Int, T}}`
  - `DenseVector{T}` = `Dict{DInt, T}`
  - `DenseMatrix{T}` = `Dict{DInt, Dict{DInt, T}}`

Experimental Results

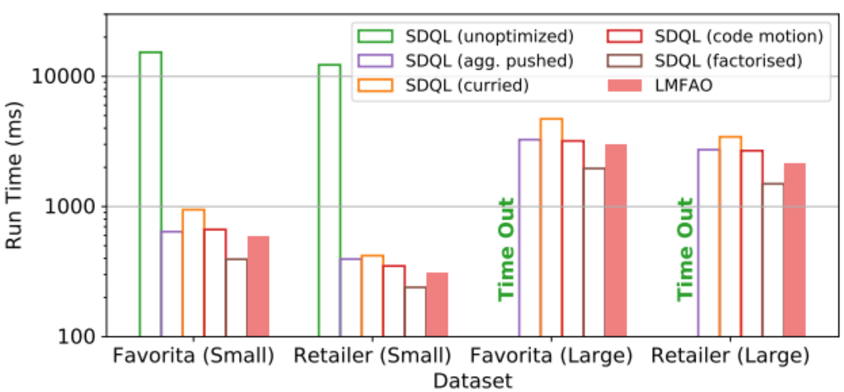


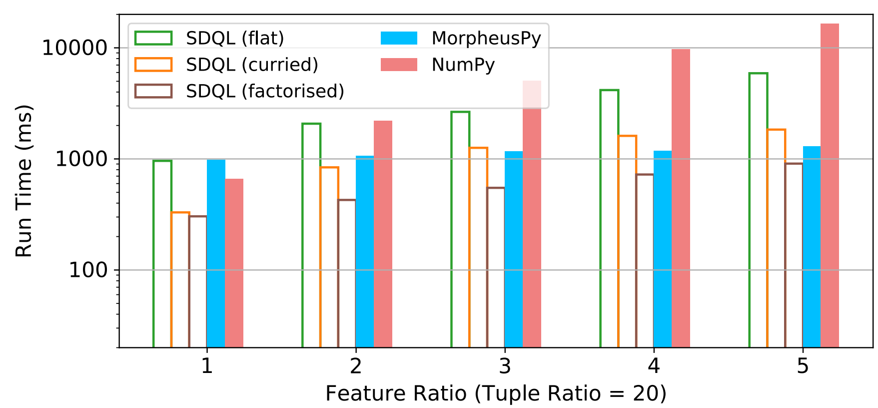


DB

LA

DB + LA





Competitive with (or better than) specialized systems

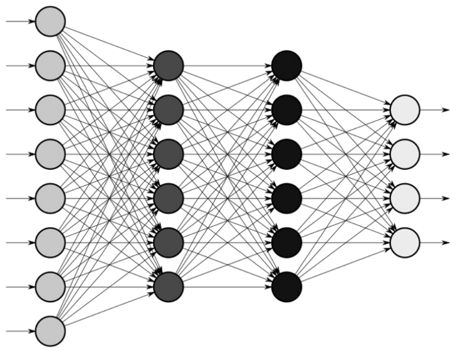
SPARSE TENSOR ALGEBRA

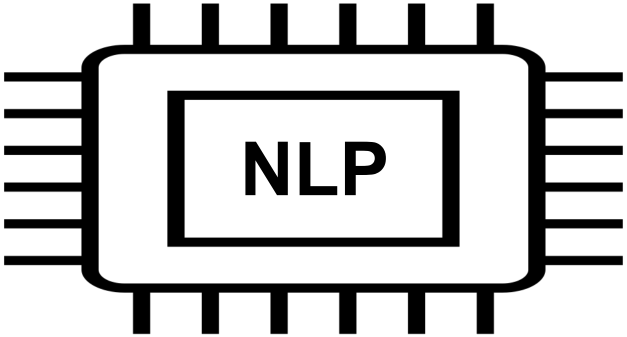
Optimizing Tensor Programs on Flexible Storage

MAXIMILIAN SCHLEICH, RelationalAI, USA
AMIR SHAIKHHA, University of Edinburgh, United Kingdom
DAN SUCIU, University of Washington, USA

SIGMOD'23

Tensors

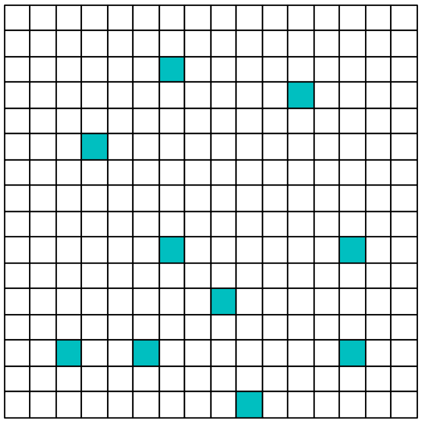


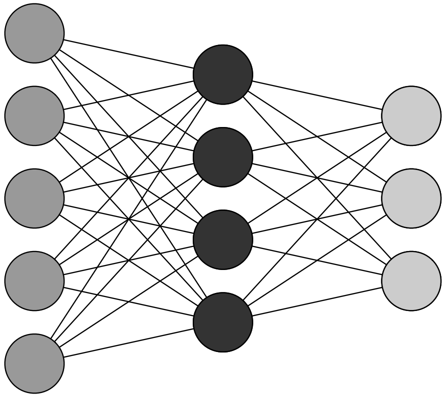


Sparse Tensors

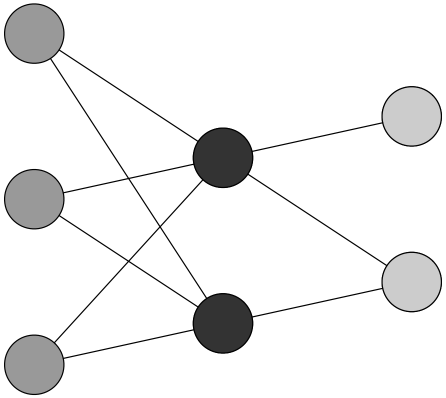
Adjacency

Matrix





Sparsification



Sparse Matrix as Relation

A

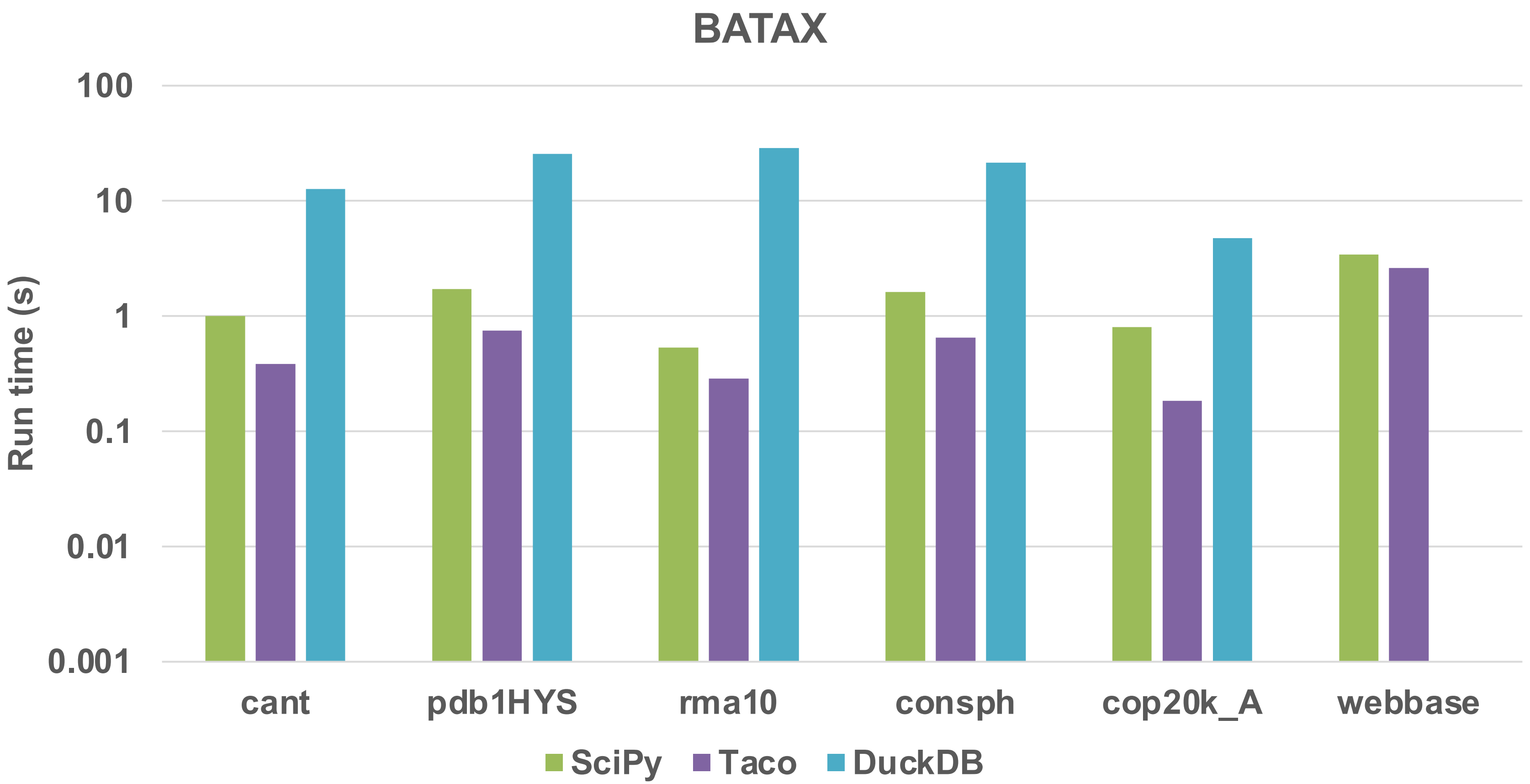
	0	1	2	3	4	5
0	5	1				
1	7	3				
2						
3	8			4	9	

$$C_{ik} = \sum_{j} A_{ij} B_{jk}$$

Row	Col	Val
0	0	5
0	1	1
1	0	7
1	1	3
3	0	8
3	3	4
3	4	9

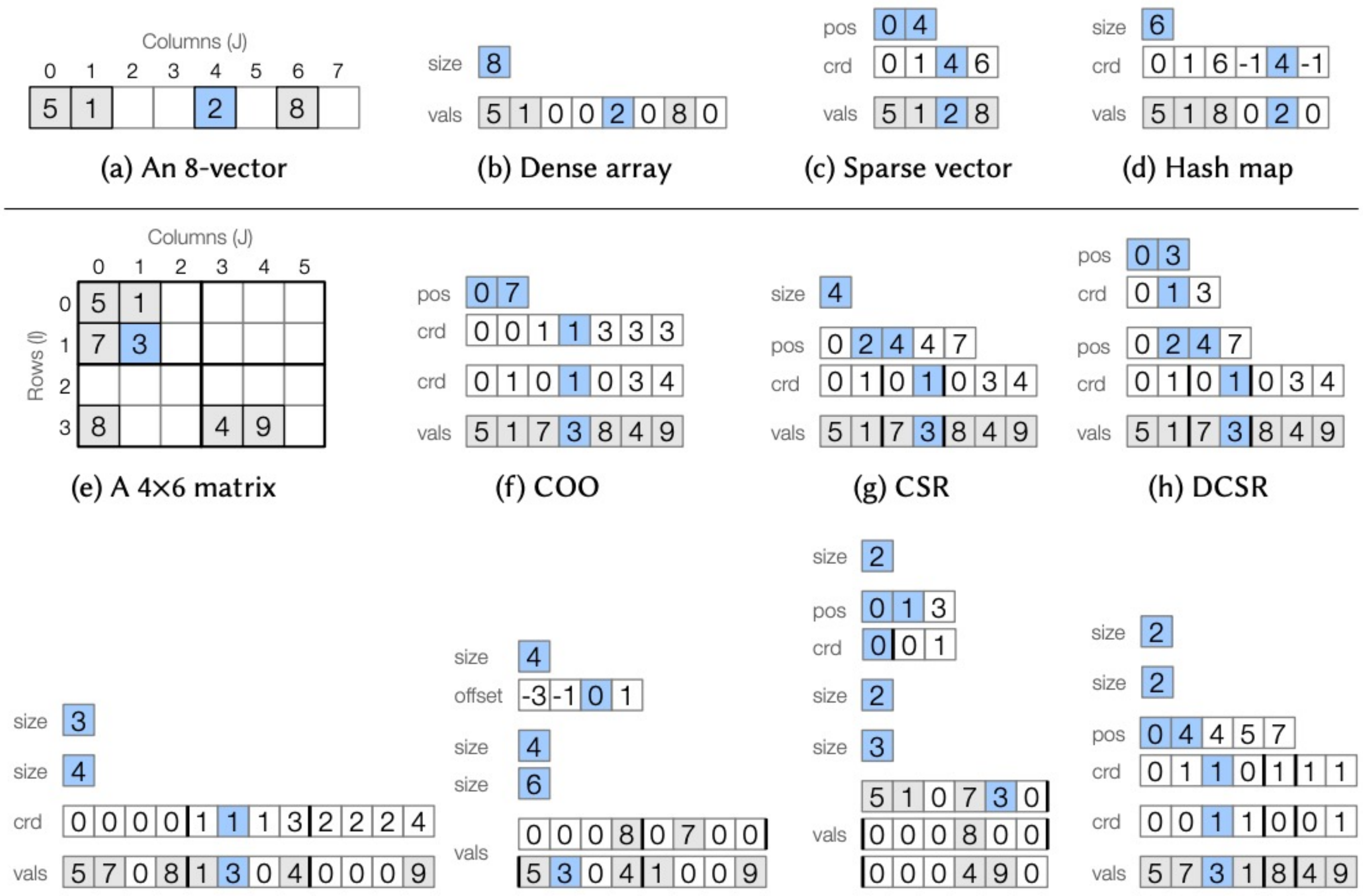
```
SELECT A.row, B.col,
       SUM(A.val*B.val)
FROM A, B
WHERE A.col = B.row
GROUP BY A.row, B.col
```

Are Database Engines Competitive?



No, because they do not support optimized storage formats for sparse tensors

Sparse Storage Formats



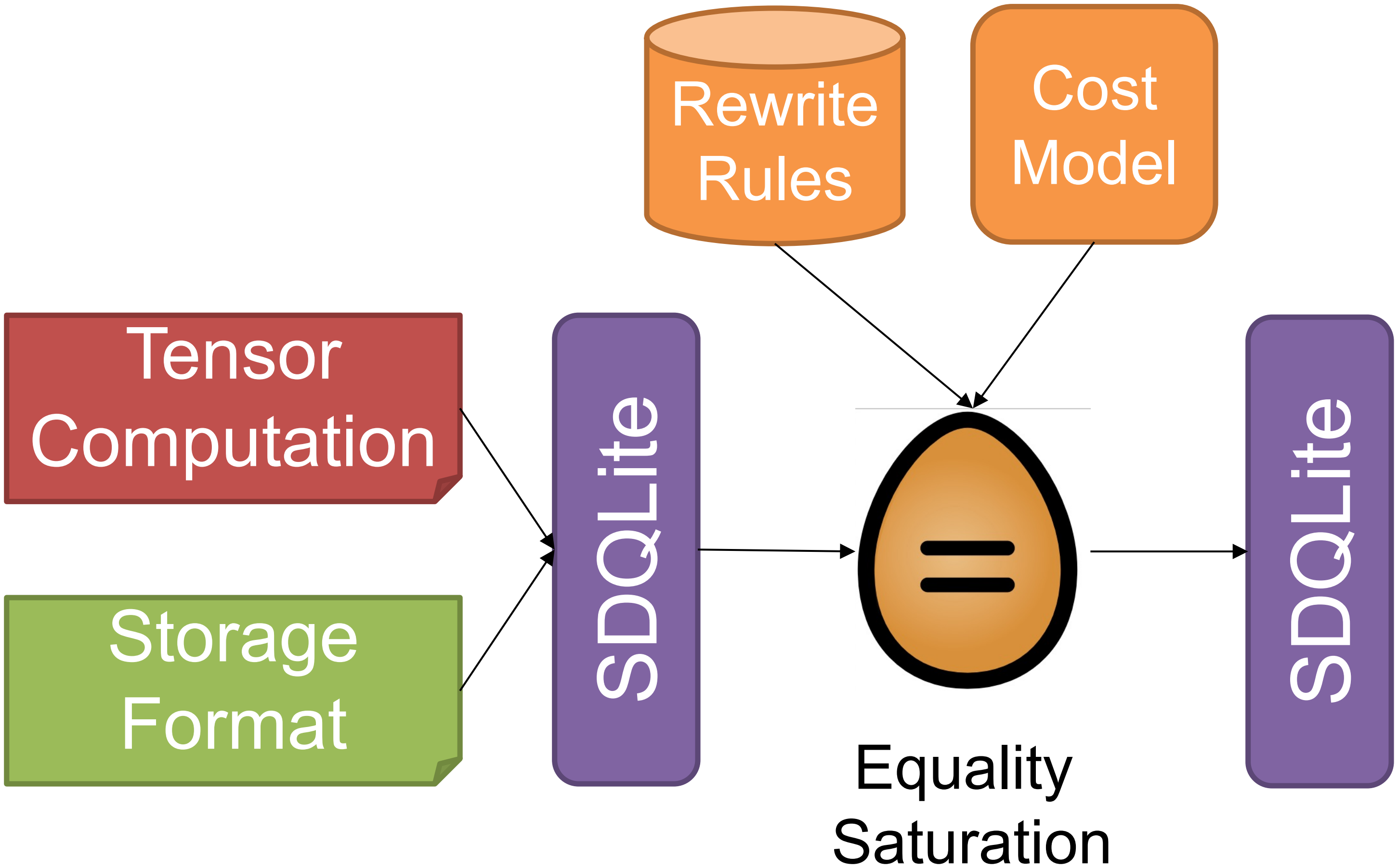
(a) An 8-vector (b) Dense array (c) Sparse vector (d) Hash map

(e) A 4×6 matrix (f) COO (g) CSR (h) DCSR

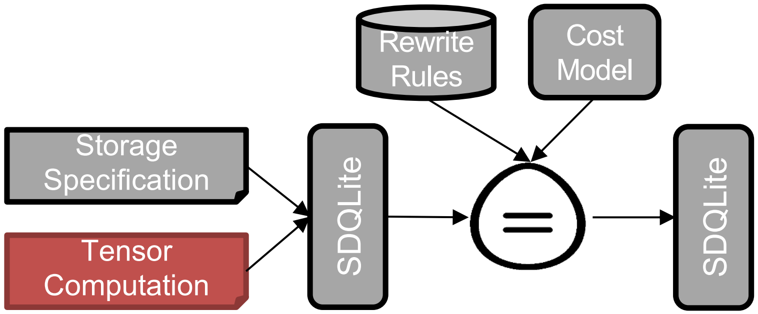
(i) ELL (j) DIA (k) BCSR (l) CSB

Chou, Stephen, Fredrik Kjolstad, and Saman Amarasinghe. "Format abstraction for sparse tensor algebra compilers.", OOPSLA'18

Flexible Storage Formats in SDQLite



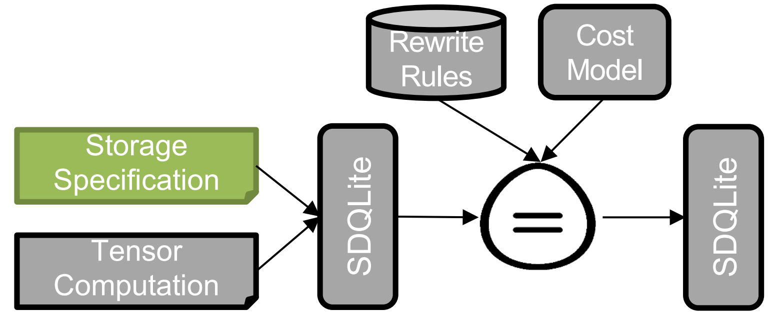
Tensor Computation



$$C_{ik} = \sum_{j} A_{ij} B_{jk}$$

```
sum(<(i,j),A_v> in A,
    <(j,k),B_v> in B)
  { (i,k) -> A_v*B_v }
```

Storage Specification

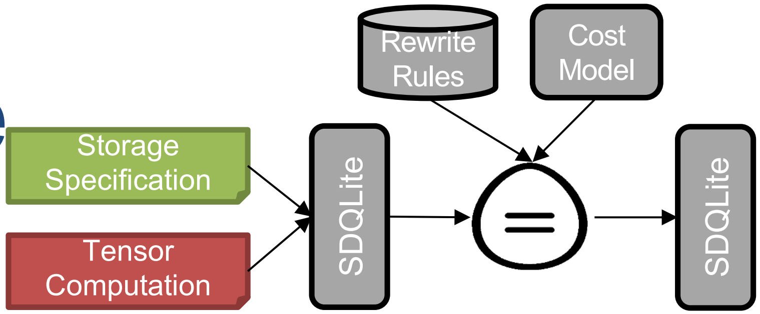


size	4						
pos	0	2	4	4	4	7	
cols	0	1	0	1	0	3	4
vals	5	1	7	3	8	4	9

	0	1	2	3	4	5
0	5	1				
1	7	3				
2						
3	8			4	9	

```
sum(<r,_> in 0:size)
    sum(<i,c> in cols(pos(r):pos(r+1)))
        { (r,c) -> vals(i) }
```

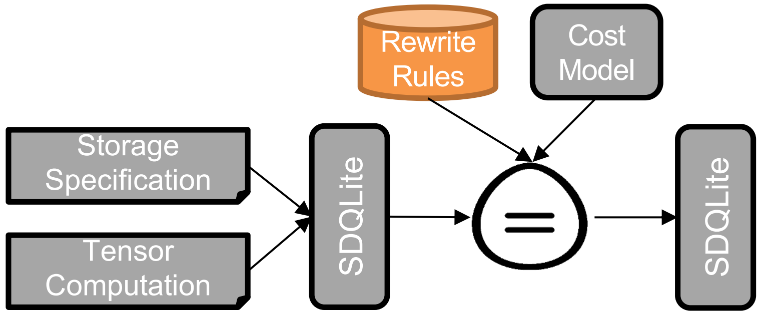
Computation+Storage




```
let A =
    sum(<r,_> in 0:A_size)
        sum(<i,c> in A_cols(A_pos(r):A_pos(r+1)))
            { (r,c) -> A_vals(i) } in
let B =
    sum(<r,_> in 0:B_size)
        sum(<i,c> in B_cols(B_pos(r):B_pos(r+1)))
            { (r,c) -> B_vals(i) } in
sum(<(i,j),A_v> in A, <(j,k),B_v> in B)
  { (i,k) -> A_v*B_v }
```

Inefficient

Rewrite Rules



- We have 44 rewrite rules
- Loop Fusion

```
let A =
    sum(<k1,v1> in D)
        { k1 -> f(k1,v1) } in
sum(<k2,v2> in A)
    g(k2,v2)
```

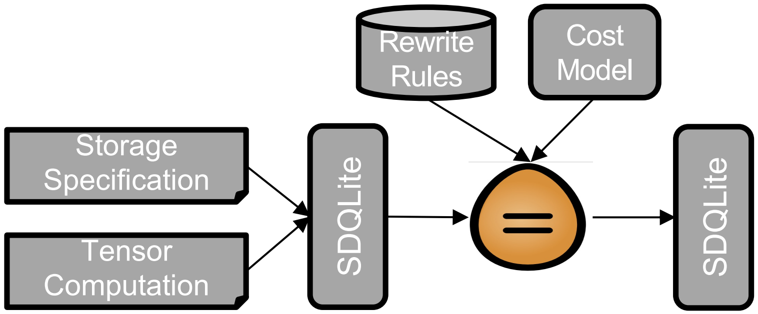
```
sum(<k1,v1> in D)
    let v2 = f(k1,v1) in
    g(k1,v2)
```

- Factorization

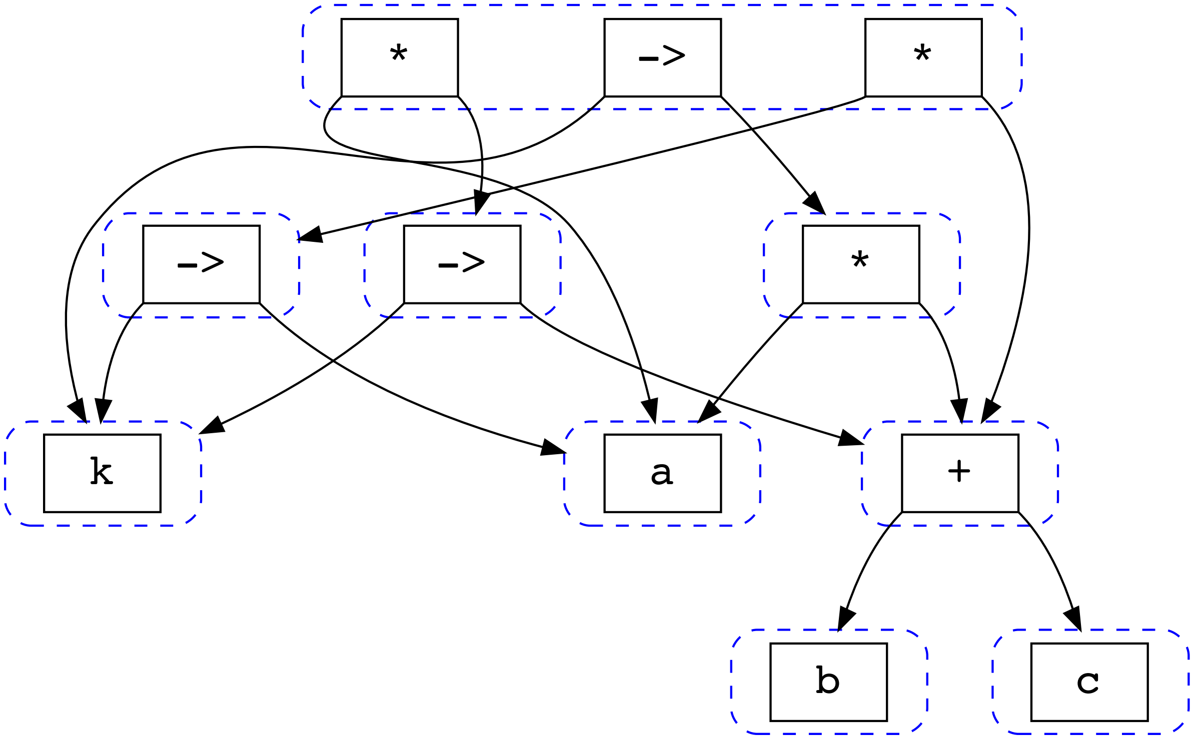
```
sum(<k,v> in A)
    e * f(k,v)
```

```
e * sum(<k,v> in A)
        f(k,v)
```

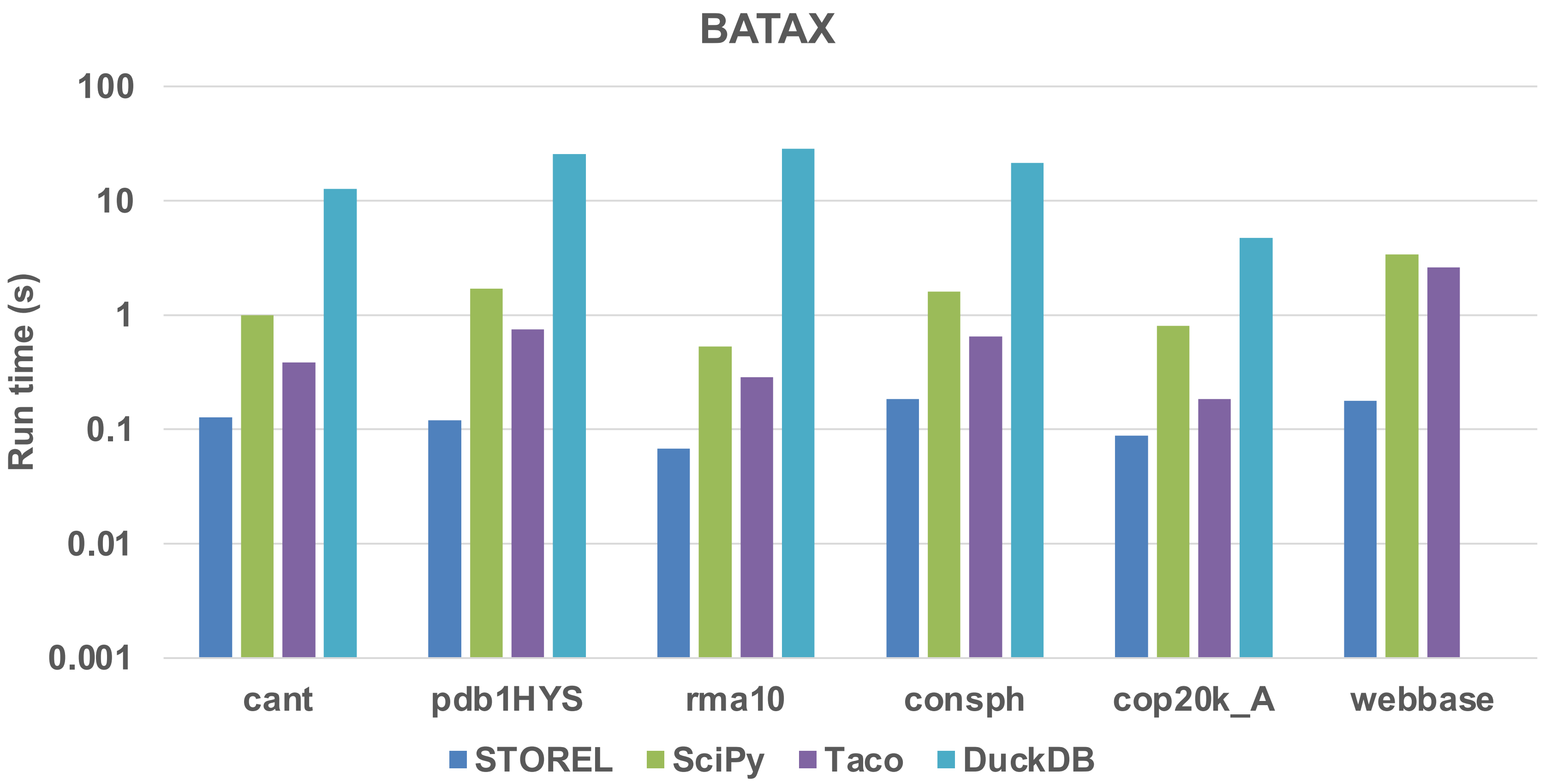
Equality Saturation



- E-graph: Compressed representation of Search Space



Performance Results



Optimizations + Compressed Storage

SPARSE DIFFERENTIATION

A Tensor Algebra Compiler for Sparse Differentiation

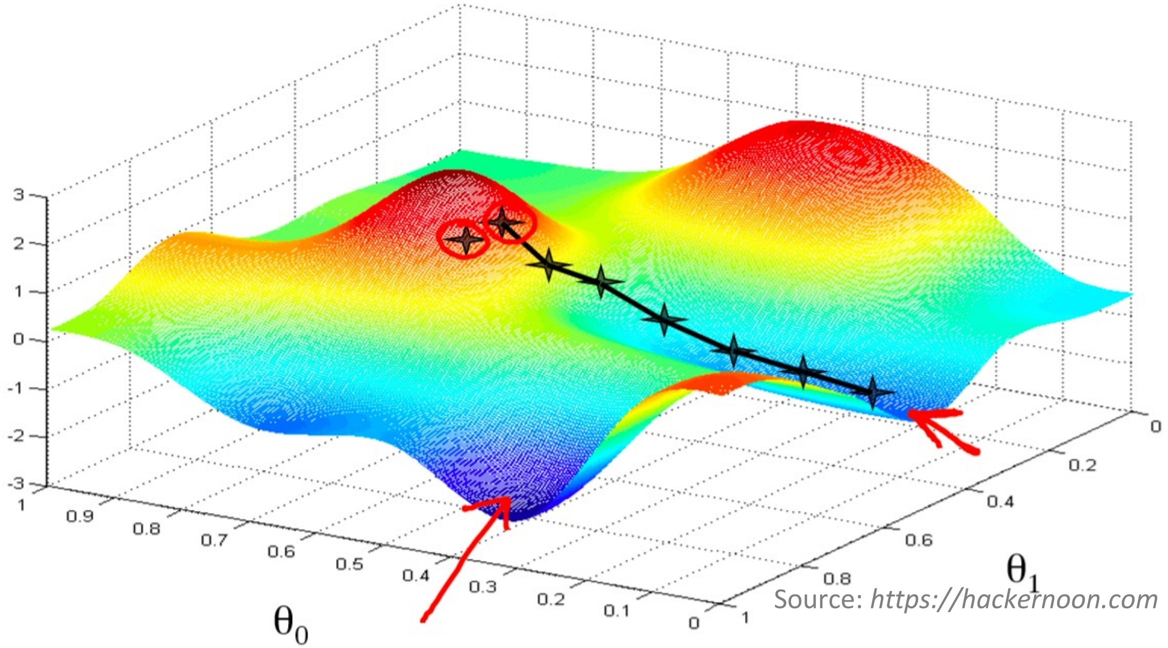
Amir Shaikhha
School of Informatics
University of Edinburgh
Edinburgh, United Kingdom
amir.shaikhha@ed.ac.uk

Mathieu Huot
Department of Computer Science
University of Oxford
Oxford, United Kingdom
mathieu.huot@stx.ox.ac.uk

Shideh Hashemian
School of Informatics
University of Edinburgh
Edinburgh, United Kingdom
s.hashemian@sms.ed.ac.uk

CGO'24

Gradient Descent Based Algorithms



Source: *https://hackernoon.com*

```
Repeat until converges {
```

$$\theta_i = \theta_i - \alpha \frac{\partial}{\partial \theta} f(\theta_i)$$

```
}
```

Derivative of a function

Automatic Differentiation (AD)

- Differentiable Programming
- A systematic approach for computing the derivative of a function
- Functions represented as programs

$$f: \mathbb{R}^n \rightarrow \mathbb{R}^m \quad \Longrightarrow \quad \frac{\partial f}{\partial x}: \mathbb{R}^n \rightarrow \mathbb{R}^{m \times n}$$

AD in a Nutshell

$$\mathcal{D}_{\mathcal{T}}[\![\text{Num}]\!] \quad = \quad \text{Num} \times \text{Num}$$

Original Tangent

Dual Number

$$\mathcal{D}_{\mathcal{T}}[\![\texttt{Array<M>}]\!] \quad = \quad \texttt{Array<}\mathcal{D}_{\mathcal{T}}[\![\text{M}]\!]\texttt{>}$$

$$\mathcal{D}_{\mathcal{T}}[\![\text{T}_1 \Rightarrow \text{T}_2]\!] \quad = \quad \mathcal{D}_{\mathcal{T}}[\![\text{T}_1]\!] \Rightarrow \mathcal{D}_{\mathcal{T}}[\![\text{T}_2]\!]$$

$$\mathcal{D}_{\mathcal{T}}[\![\text{M}_1 \times \text{M}_2]\!] \quad = \quad \mathcal{D}_{\mathcal{T}}[\![\text{M}_1]\!] \times \mathcal{D}_{\mathcal{T}}[\![\text{M}_2]\!]$$

Sparse Differentiation

- Tensor Libraries: TensorFlow, PyTorch

```
term = lambda V1: tensordot(V1, V2, 1)
dTerm = jacobian(term, V1)
```

- Function
- Support
- Lack of
  - Cryptic
  - Not expressive enough

Challenges for Sparse Differentiation

- Differentiated functions require control-flow constructs / User-Defined Functions
  - Beyond tensor algebra
- Sparse tensor programs are complicated
  - Imperative loopy code (co-iteration)
  - Data formats (CSR, CSC, CSF)

Imperative Loops in Sparse TA

```
for (int i = 0; i < m; i++) {
  for (int j = 0; j < n; j++) {
    int pB2 = i * n + j;
    int pA2 = i * n + j;
    for (int k = 0; k < p; k++) {
      int pB3 = pB2 * p + k;
      A[pA2] += B[pB3] * c[k];
    }
  }
}
```

Fig. 1. $A_{ij} = \sum_k B_{ijk} c_k$

```
for (int pB1 = B1_pos[0];
         pB1 < B1_pos[1];
         pB1++) {
  int i = B1_idx[pB1];
  for (int pB2 = B2_pos[pB1];
           pB2 < B2_pos[pB1+1];
           pB2++) {
    int j = B2_idx[pB2];
    int pA2 = i * n + j;
    for (int pB3 = B3.pos[pB2];
             pB3 < B3.pos[pB2+1];
             pB3++) {
      int k = B3_idx[pB3];
      A[pA2] += B[pB3] * c[k];
    }
  }
}
```

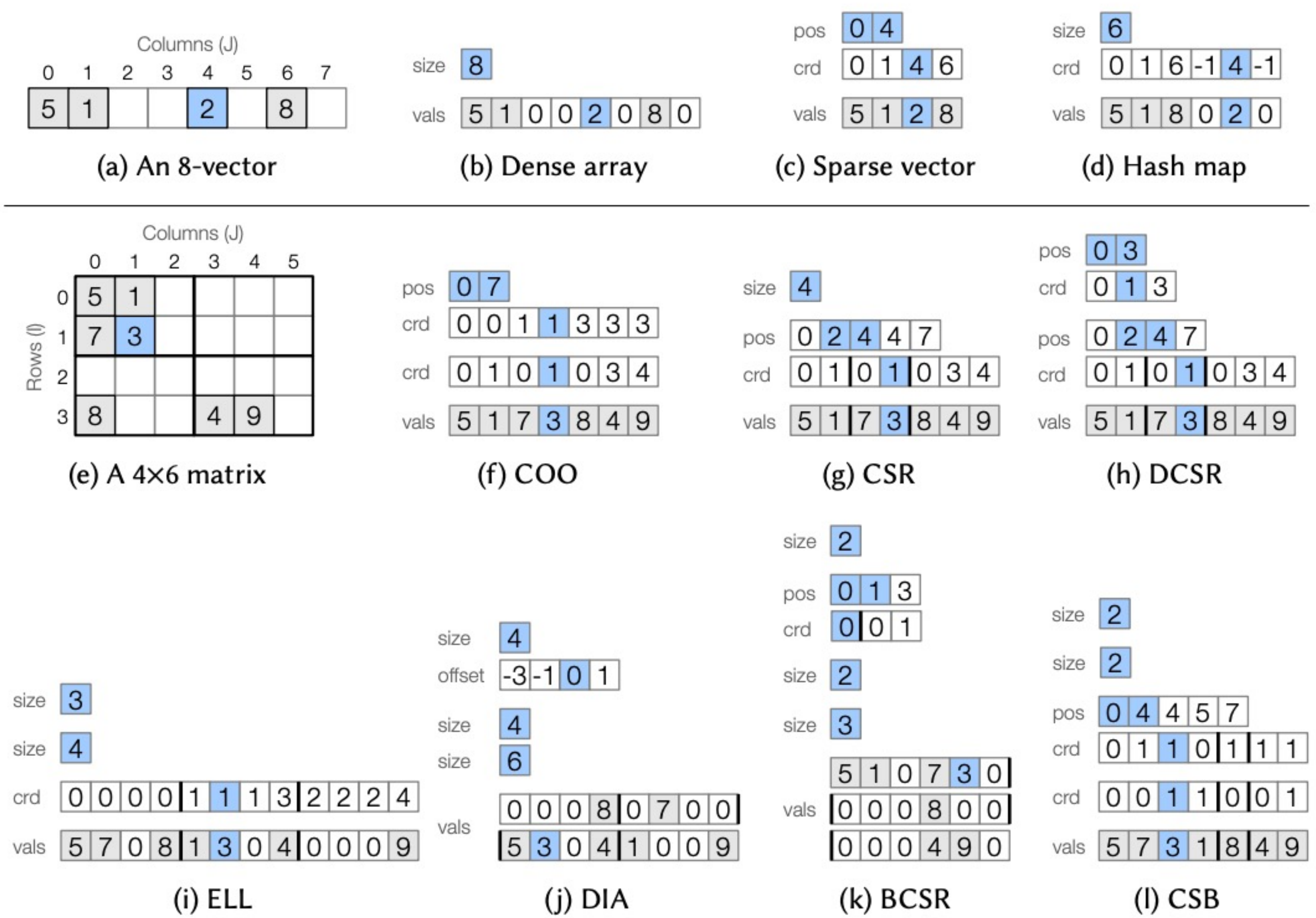
Fig. 2. $A_{ij} = \sum_k B_{ijk} c_k$ (sparse B)

```
for (int pB1 = B1_pos[0];
         pB1 < B1_pos[1];
         pB1++) {
  int i = B1_idx[pB1];
  for (int pB2 = B2_pos[pB1];
           pB2 < B2_pos[pB1+1];
           pB2++) {
    int j = B2_idx[pB2];
    int pA2 = i * n + j;
    int pB3 = B3_pos[pB2];
    int pc1 = c1_pos[0];
    while (pB3 < B3_pos[pB2+1] &&
           pc1 < c1_pos[1]) {
      int kB = B3_idx[pB3];
      int kc = c1_idx[pc1];
      int k = min(kB, kc);
      if (kB == k && kc == k) {
        A[pA2] += B[pB3] * c[pc1];
      }
      if (kB == k) pB3++;
      if (kc == k) pc1++;
    }
  }
}
```

Fig. 3. $A_{ij} = \sum_k B_{ijk} c_k$ (sparse B, c)

Fredrik Kjolstad et al.
"The Tensor Algebra Compiler", OOPSLA'17

Sparse Storage Formats



Chou, Stephen, Fredrik Kjolstad, and Saman Amarasinghe. "Format abstraction for sparse tensor algebra compilers.", OOPSLA'18

Our solution: Separation of Concerns

- Logical SDQL
  - Tensor algebra + more
  - AD friendly
- Physical SDQL
  - Sparse storage formats
  - Efficient

Logical SDQL

$$\sum_i A_i B_i$$

```
sum(<i,a> in A) a * B(i)
```

AD in Logical SDQL

$$\frac{\partial \sum_i A_i B_i}{\partial B}$$

```
gradient    (sum(<i,a> in A) a * B(i)) B
```

AD in Logical SDQL (cont.)

$\mathcal{D}_\tau[\![\mathtt{T}]\!]$ Tensorized FAD on Types

$$\mathcal{D}_\tau[\![\ \mathtt{D}\ ]\!] = \mathtt{D} \otimes \tau$$
$$\mathcal{D}_\tau[\![\ \mathtt{bool}\ ]\!] = \mathtt{real}$$
$$\mathcal{D}_\tau[\![\ \mathtt{int}\ ]\!] = \mathtt{real}$$

$\mathcal{D}_\tau[\![\Gamma]\!]$ Tensorized FAD on Context

$$\mathcal{D}_\tau[\![\emptyset]\!] = \emptyset$$
$$\mathcal{D}_\tau[\![\Gamma, \mathtt{x}:\mathtt{T}]\!] = \mathcal{D}_\tau[\![\Gamma]\!], \mathtt{x}:\mathtt{T}, \mathtt{x'}:\mathcal{D}_\tau[\![\mathtt{T}]\!]$$

$\mathcal{D}_\tau[\![\mathtt{e}]\!]$ Tensorized FAD on Expressions – Invariant: If $\Gamma \vdash \mathtt{e}:\ \mathtt{T}$, then $\mathcal{D}_\tau[\![\Gamma]\!] \vdash \mathcal{D}_\tau[\![\mathtt{e}]\!] : \mathcal{D}_\tau[\![\mathtt{T}]\!]$

$$\mathcal{D}_\tau[\![\ \mathtt{sum(<k,v>\ in\ e1)\ e2}\ ]\!] = \mathtt{sum(<k,v>\ in\ e1)\ let\ <k',\ v'>\ =\ <0,}\ \mathcal{D}_\tau[\![\mathtt{e1(k)}]\!]\mathtt{>\ in}\ \mathcal{D}_\tau[\![\mathtt{e2}]\!]$$
$$\mathcal{D}_\tau[\![\ \mathtt{let\ x\ =\ e1\ in\ e2}\ ]\!] = \mathtt{let\ <x,\ x'>\ =\ <e1,}\ \mathcal{D}_\tau[\![\mathtt{e1}]\!]\mathtt{>\ in}\ \mathcal{D}_\tau[\![\mathtt{e2}]\!]$$
$$\mathcal{D}_\tau[\![\ \mathtt{if\ e1\ then\ e2}\ ]\!] = \mathtt{if\ e1\ then}\ \mathcal{D}_\tau[\![\mathtt{e2}]\!]$$
$$\mathcal{D}_\tau[\![\ \mathtt{\{\ e1\ ->\ e2\ \}}\ ]\!] = \mathtt{\{\ e1\ ->}\ \mathcal{D}_\tau[\![\mathtt{e2}]\!]\ \mathtt{\}}$$
$$\mathcal{D}_\tau[\![\ \mathtt{e1\ *\ e2}\ ]\!] = \mathtt{e1} * \mathcal{D}_\tau[\![\mathtt{e2}]\!] + \mathcal{D}_\tau[\![\mathtt{e1}]\!] \star^{T} [\tau]\ \mathtt{e2}$$
$$\mathcal{D}_\tau[\![\ \mathtt{e1(e2)}\ ]\!] = \mathcal{D}_\tau[\![\mathtt{e1}]\!]\mathtt{(e2)}$$
$$\mathcal{D}_\tau[\![\ \mathtt{e1\ +\ e2}\ ]\!] = \mathcal{D}_\tau[\![\mathtt{e1}]\!] + \mathcal{D}_\tau[\![\mathtt{e2}]\!]$$
$$\mathcal{D}_\tau[\![\ \mathtt{uop(e)}\ ]\!] = \mathtt{uop'(e)} * \mathcal{D}_\tau[\![\mathtt{e}]\!]$$
$$\mathcal{D}_\tau[\![\ \mathtt{x}\ ]\!] = \mathtt{x'} \qquad \mathcal{D}_\tau[\![\ \mathtt{r}\ ]\!] = \mathtt{zero}[\tau] \qquad \mathcal{D}_\tau[\![\ \mathtt{n}\ ]\!] = \mathcal{D}_\tau[\![\mathtt{false}]\!] = \mathcal{D}_\tau[\![\mathtt{true}]\!] = 0$$

$$\mathtt{e1} \star^{T} [\mathtt{real}]\ \mathtt{e2} \triangleq \mathtt{e1\ *\ e2}$$
$$\mathtt{e1} \star^{T} [\mathtt{tensor\ n}]\ \mathtt{e2} \triangleq \mathtt{sum(<i\_1,r\_2>\ in\ e1)\ \ldots\ sum(<i\_m,v>\ in\ r\_m))}$$
$$\text{if } \mathtt{e1:\ tensor\ (m\ +\ n)} \qquad \mathtt{\{i\_1\ ->\ \ldots\ \{\ i\_m\ ->\ 1\ \}\ \ldots\ \}\ *\ e2\ *\ v}$$

AD in Logical SDQL (cont.)

```
gradient  (sum(<i,a> in A) a * B(i)) B
```

⇓ Tensorized AD

```
let A′ = {} in
let B′ = sum(<i,_> in B) {i -> {i -> 1}} in
sum(<i, a> in A)
  let <i′, a′> = <{}, A′(i)> in
  a * B′(i) + a′ * B(i)
```

⇓ Optimizations

```
sum(<i, a> in A) { i -> a }
```

Optimizations in Logical SDQL

```
let A′ = {} in
let B′ = sum(<i,_> in B) {i -> {i -> 1}} in
sum(<i, a> in A)
  let <i′, a′> = <{}, A′(i)> in
  a * B′(i) + a′ * B(i)
```

```
let B′ = sum(<i,_> in B) {i -> {i -> 1}} in
sum(<i, a> in A)
  let <i′, a′> = <{}, {}> in
  a * B′(i) + a′ * B(i)
```

Optimizations in Logical SDQL (cont.)

```
let B′ = sum(<i,_> in B) {i -> {i -> 1}} in
sum(<i, a> in A)
  let <i′, a′> = <{}, {}> in
  a * B′(i) + a′ * B(i)
```

```
let B′ = sum(<i,_> in B) {i -> {i -> 1}} in
sum(<i, a> in A)
  a * B′(i) + {} * B(i)
```

Optimizations in Logical SDQL (cont.)

```
let B′ = sum(<i,_> in B) {i -> {i -> 1}} in
sum(<i, a> in A)
  a * B′(i)
```

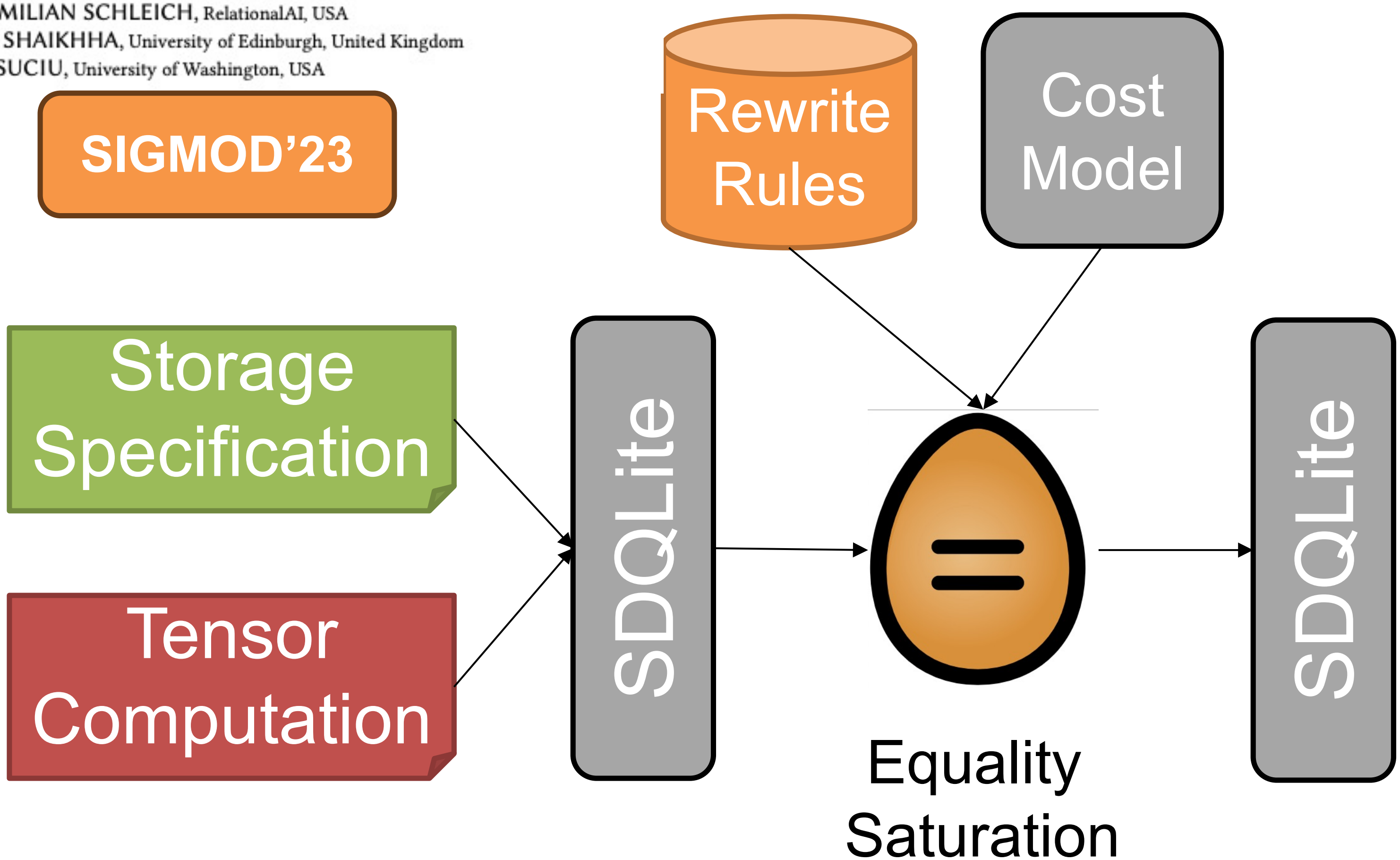
```
sum(<i, a> in A)
  a * {i -> 1}
```

Physical SDQL == SDQLite

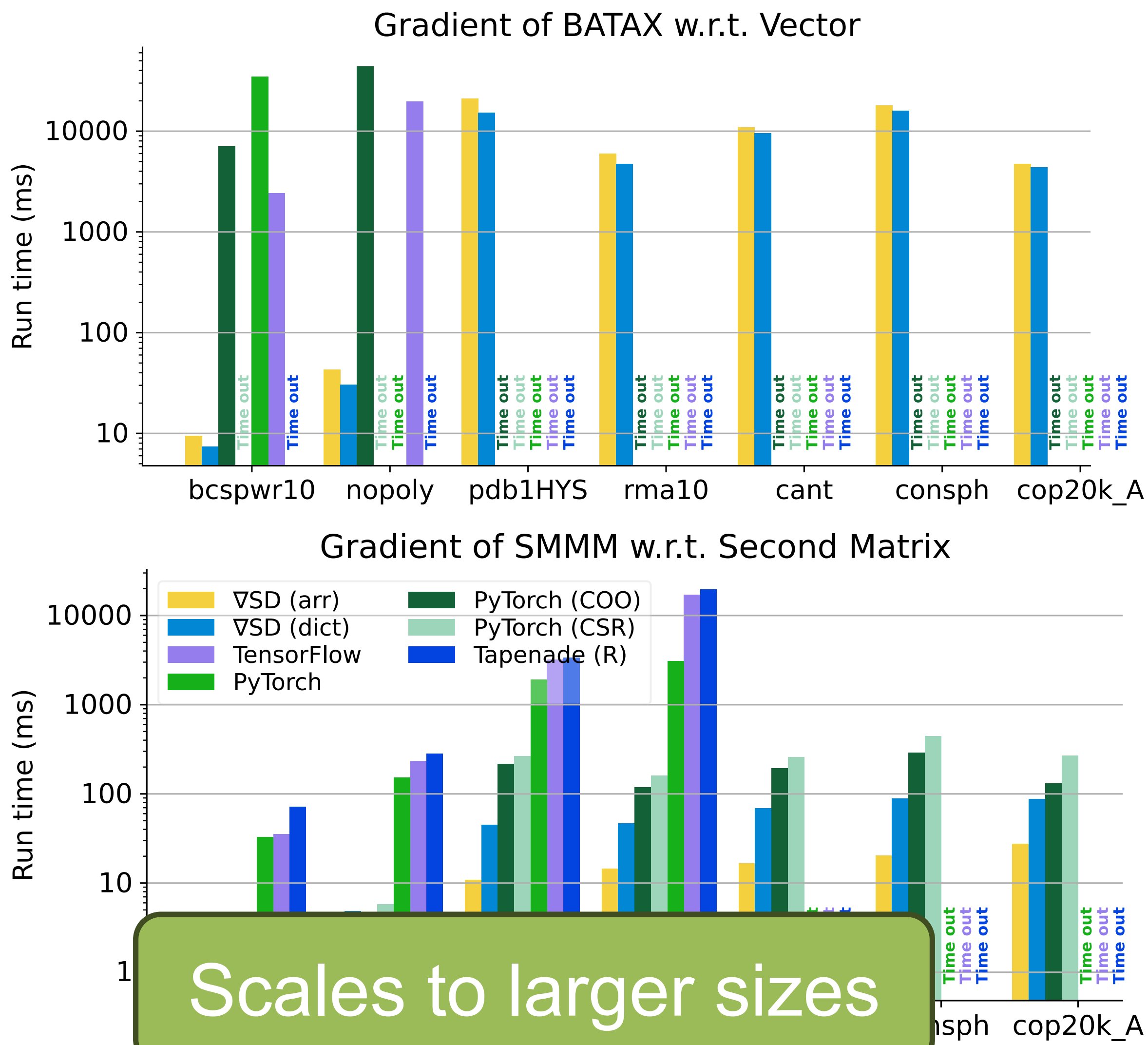
Optimizing Tensor Programs on Flexible Storage

MAXIMILIAN SCHLEICH, RelationalAI, USA
AMIR SHAIKHHA, University of Edinburgh, United Kingdom
DAN SUCIU, University of Washington, USA

SIGMOD'23



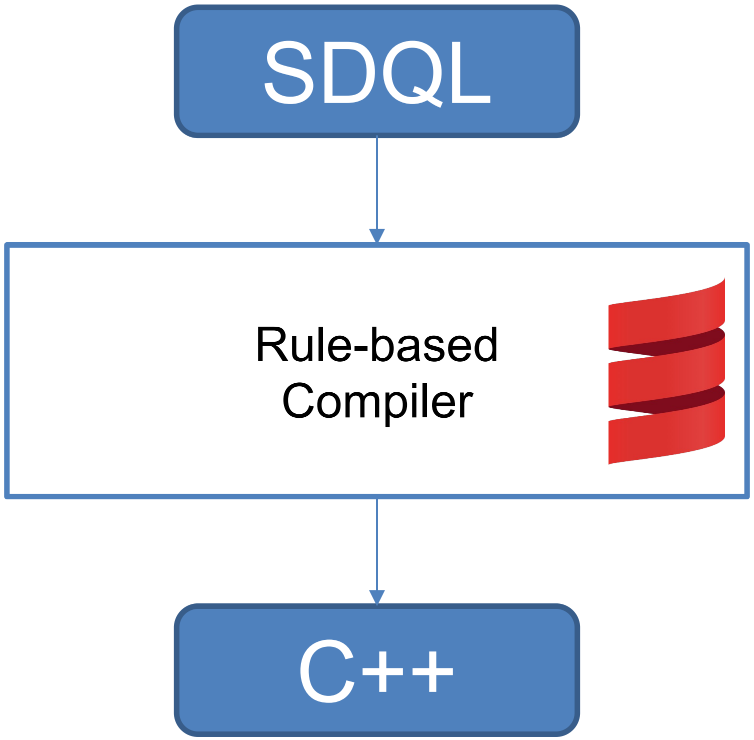
Performance Results



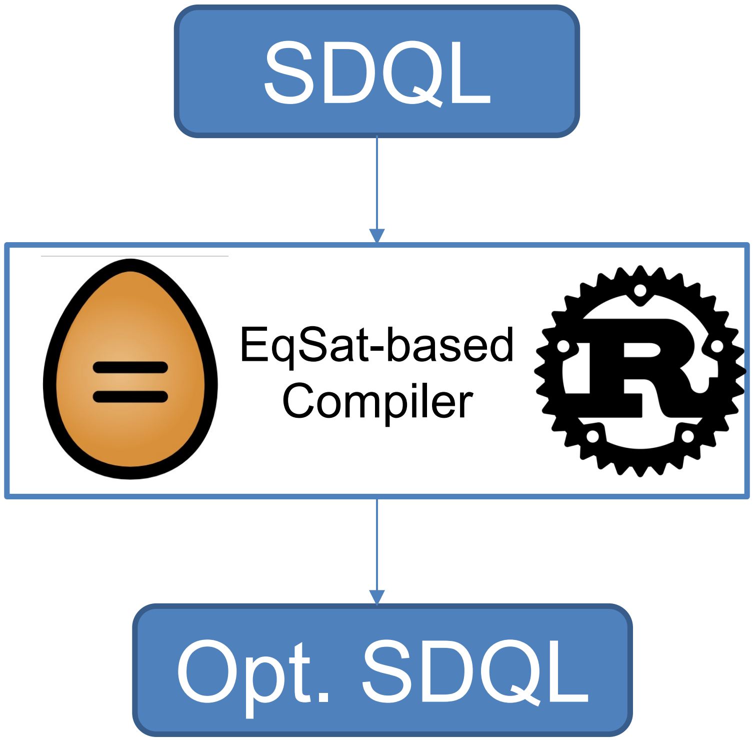
Scales to larger sizes

IMPLEMENTATION

Compiler v1 [OOPSLA'22]



Towards Eq-Sat-Based Compiler



Challenges with Equality Saturation

- Scalability
  - Variable binding
- Scalability
  - Associativity/Commutativity
- Scalability
  - One directional rules
- Scalability
  - Search space is BIG!
- Analysis
  - Cardinality estimation
  - Cost estimation

Variable binding

- De bruijn indexing

Sketch-Guided Equality Saturation

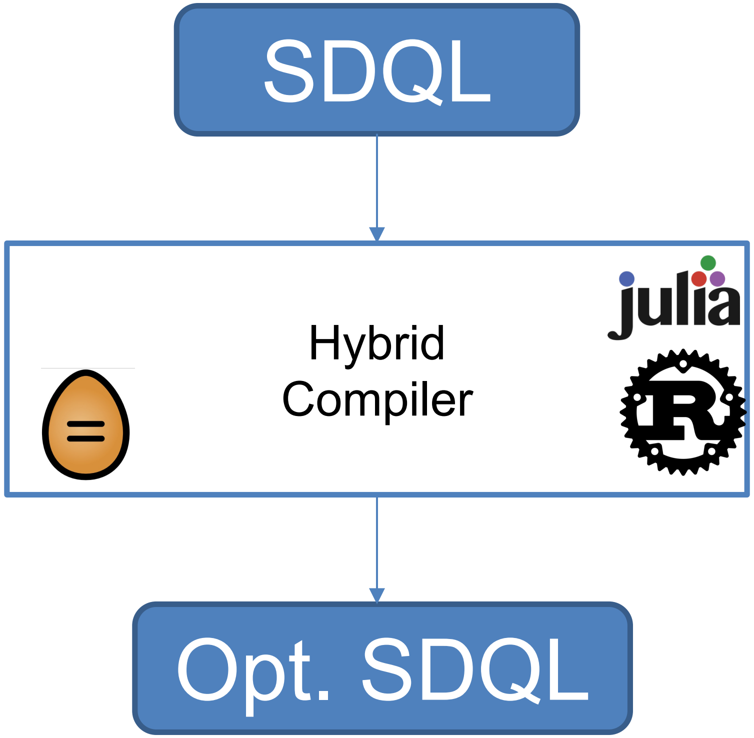
Scaling Equality Saturation to Complex Optimizations in Languages with Bindings

Thomas Kœhler
University of Glasgow
Scotland, UK
thomas.koehler@thok.eu

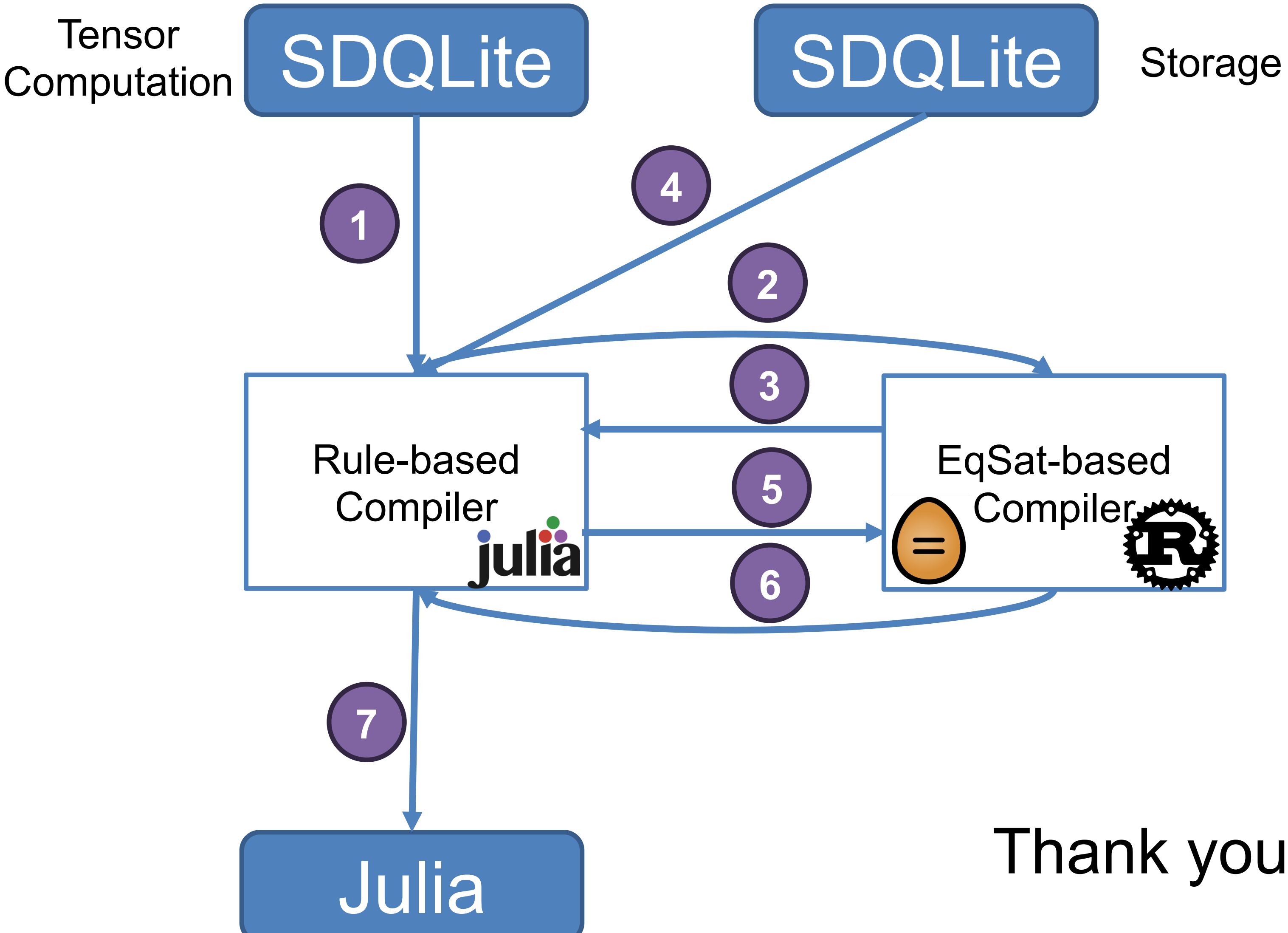
Phil Trinder
University of Glasgow
Scotland, UK
Phil.Trinder@glasgow.ac.uk

Michel Steuwer
University of Edinburgh
Scotland, UK
michel.steuwer@ed.ac.uk

Hybrid Compiler

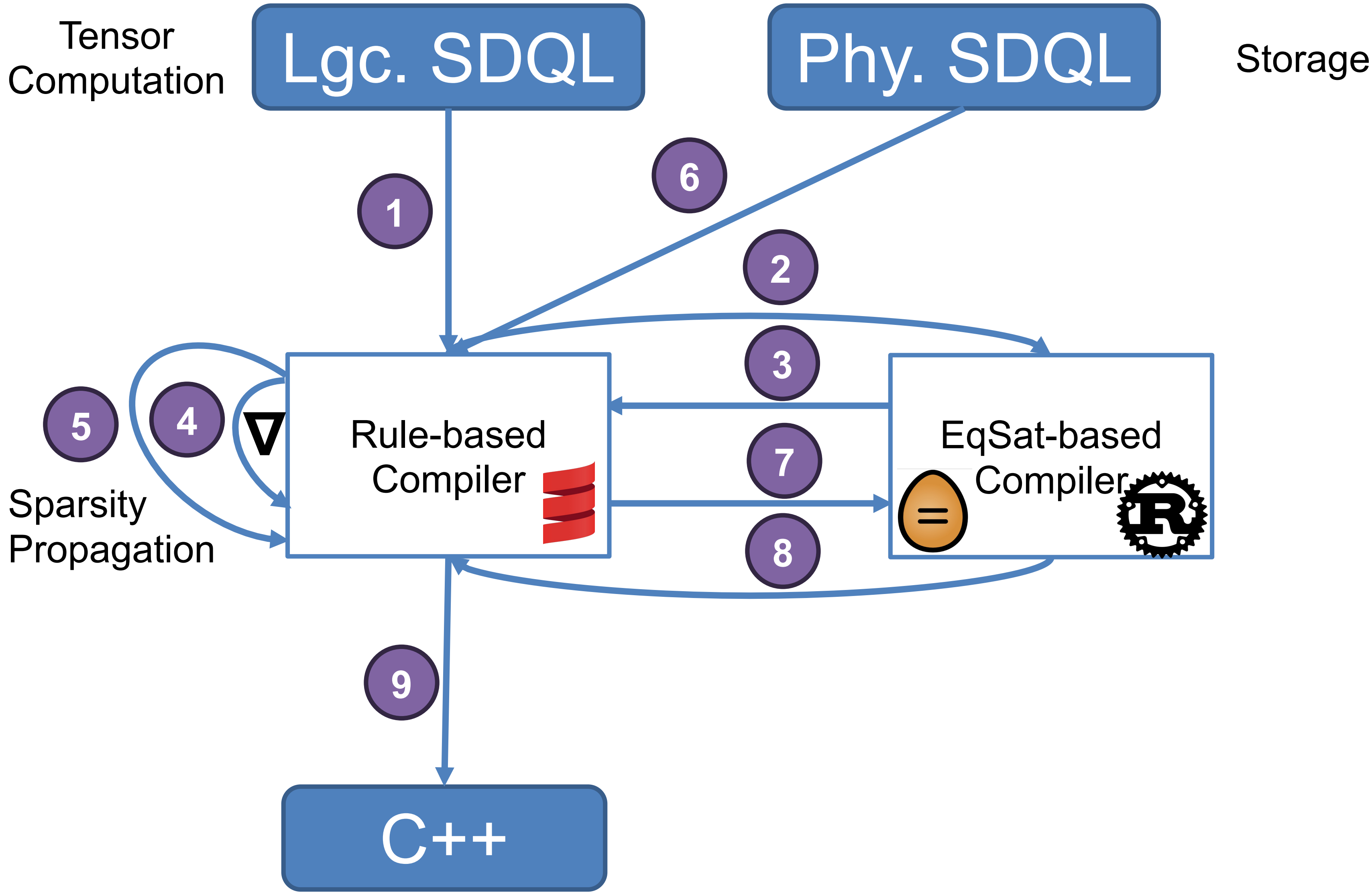


Compiler v2 [SIGMOD'23]



Thank you Remy!

Compiler v3 [CGO'24]



Challenges with EqSat

- Scalability
  - DB/PL/ML ideas
  - For Reverse-mode AD, limited use of EqSat
- Analysis

Better Together: Unifying Datalog and Equality Saturation

YIHONG ZHANG, University of Washington, USA
YISU REMY WANG, University of Washington, USA
OLIVER FLATT, University of Washington, USA
DAVID CAO, UC San Diego, USA
PHILIP ZUCKER, Draper Laboratory, USA
ELI ROSENTHAL, Google, USA
ZACHARY TATLOCK, University of Washington, USA
MAX WILLSEY, University of Washington, USA

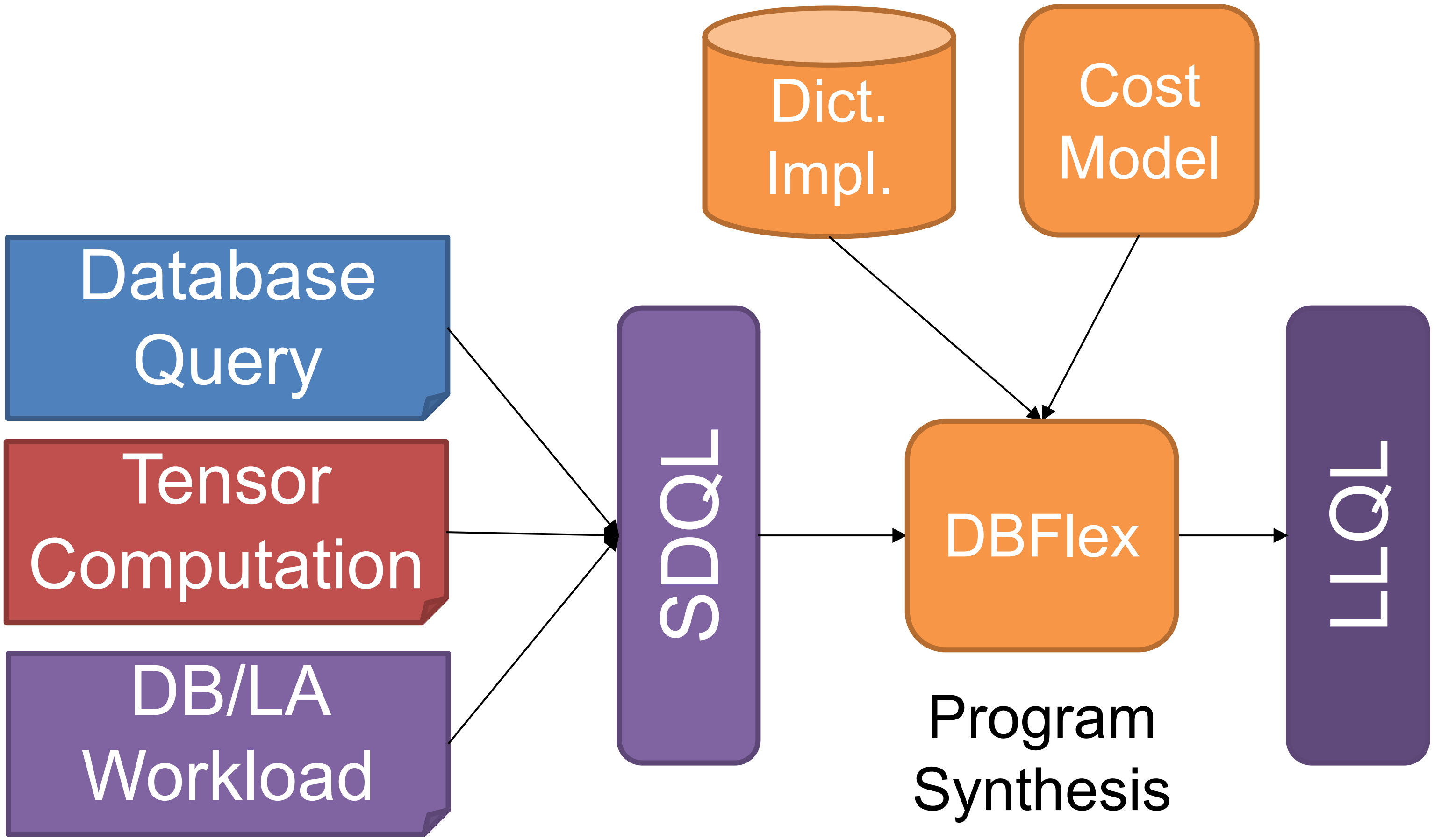
Latent Idiom Recognition for a Minimalist Functional Array Language using Equality Saturation

Jonathan Van der Cruysse
McGill University
Montreal, Quebec, Canada
jonathan.vandercruysse@mail.mcgill.ca

Christophe Dubach
McGill University & Mila
Montreal, Quebec, Canada
christophe.dubach@mcgill.ca

CURRENT WORK

ML for Dictionary Tuning



CGO'23

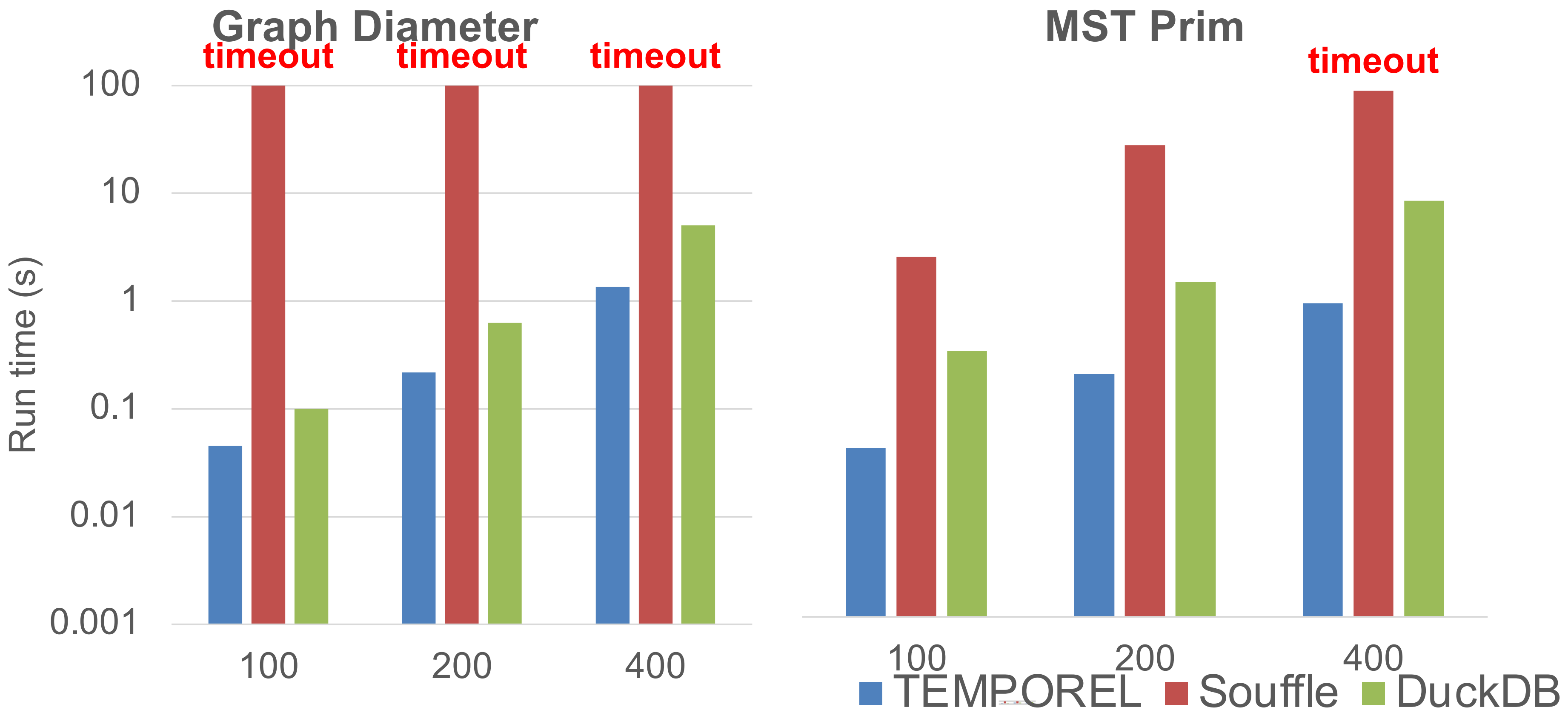
ECOOP'23

Recursive Queries

Optimizing Nested Recursive Queries

AMIR SHAIKHHA, University of Edinburgh, United Kingdom
DAN SUCIU, University of Washington, USA
MAXIMILIAN SCHLEICH, RelationalAI, USA
HUNG NGO, RelationalAI, USA

- Datalog → TempoDL → SDQL + Recursion



SIGMOD'24

Advanced Joins in SDQL

Worst-case Optimal Join Algorithms

Hung Q. Ngo
University at Buffalo, SUNY
hungngo@buffalo.edu

Ely Porat
Bar-Ilan University
porately@cs.biu.ac.il

Christopher Ré
University of Wisconsin–Madison
chrisre@cs.wisc.edu

Atri Rudra
University at Buffalo, SUNY
atri@buffalo.edu

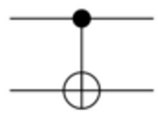
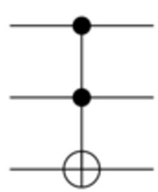
Leapfrog Triejoin: A Simple, Worst-Case Optimal Join Algorithm

Todd L. Veldhuizen
LogicBlox Inc.
Two Midtown Plaza
1349 West Peachtree Street NW
Suite 1880, Atlanta GA 30309
tveldhui@{logicblox.com,acm.org}

Free Join: Unifying Worst-Case Optimal and Traditional Joins

YISU REMY WANG, University of Washington, USA
MAX WILLSEY, University of Washington, USA
DAN SUCIU, University of Washington, USA

Quantum Simulation in SDQL

Operator	Gate(s)	Matrix
Controlled Not (CNOT, CX)		$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}$
Controlled Z (CZ)		$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&-1\end{bmatrix}$
SWAP		$\begin{bmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{bmatrix}$
Toffoli (CCNOT, CCX, TOFF)		$\begin{bmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&0&1\\0&0&0&0&0&0&1&0\end{bmatrix}$

Conclusion

- Data science is a double-edged sword!
- Radical rethinking of data science pipelines
- Language design
  - SDQL (Semi-ring Dictionary)
  - TempoDL (Low-Level Datalog) [SIGMOD'24]
  - TondIR (Python to SQL) [ICDE'24]
  - STUR (Structured Tensor Algebra) [OOPSLA'23]
  - BTL (Probablistic Language) [CC'23]
- Leverage structure
- Optimize across pipeline

Thank you

BACKUP

Vertical Loop Fusion

```
let y=sum(x in e1)                         sum(x in e1)
{x.key->f1(x.val)}                    ⤳    { x.key -> f2(f1(x.val))}
sum(x in y){x.key->f2(x.val)}
```

```
let At = sum(row in A) sum(x in row.val) { x.key -> {row.key -> x.val } }
sum(row in At) { row.key ->
  sum(x in row.val) sum(y in A(x.key))
    { y.key -> x.val * y.val } }
```

↓

```
sum(row in A)
  sum(x in row.val) { x.key ->
    sum(y in row.val) { y.key ->
      x.val * y.val } }
```

Horizontal Loop Fusion

```
let y1=sum(x in e1)f1(x)                let tmp = sum(x in e1)
let y2=sum(x in e1)f2(x)        ⇝       <y1 = f1(x), y2 = f2(x)>
f3(y1, y2)                              f3(tmp.y1, tmp.y2)
```

```
let Rsum = sum(r in R) r.key.A * r.val in
let Rcount = sum(r in R) r.val in
Rsum / Rcount
```

```
let RsumRcount = sum(r in R) < Rsum = r.key.A * r.val, Rcount = r.val > in
RsumRcount.Rsum / RsumRcount.Rcount
```

Loop Factorization

- Scalars

```
sum(x in NR) sum(y in x.key.C) x.key.A * x.val * y.key.D * y.val
```

```
sum(x in NR) x.key.A * x.val * sum(y in x.key.C) y.key.D * y.val
```

- Dictionaries

```
sum(x in NR) sum(y in x.key.C) { x.key.B -> x.key.A * x.val * y.key.D * y.val }
```

```
sum(x in NR) { x.key.B -> x.key.A * x.val * sum(y <- x.key.C) y.key.D * y.val }
```

Other Approaches

	Expressiveness					Data Representation					Specialization				
	Relational Algebra	Nested Rel. Calc.	Group-by Aggregates	Efficient Equi-Joins	Linear Algebra	Set & Bag	Dense Array	Sparse Tensor	Dictionary	Semi-rings	Loop Fusion	Loop Hoisting	Loop Memoization	Code Generation	Vectorization
SDQL (This Paper)	●	●	●	●	●	●	●	●	●	●	●	●	●	●	○
Query Compilers (HyPer)	●	○	●	●	○	●	●	○	●	○	◐	◐	○	●	○
Vectorized Query Engines (Vectorwise)	●	○	●	●	○	●	●	○	●	○	◐	◐	○	○	●
Monad Calculus (NRC^+)	●	●	○	○	○	●	○	○	○	○	◐	◐	○	○	○
Monoid Comprehension	●	●	●	○	○	●	○	○	○	○	◐	◐	○	○	○
Monad Calc. + Agg. (Kleisli, Trance)	●	●	●	○	◐	●	○	○	○	○	◐	◐	○	●	○
Lang. Integrated Queries (LINQ, CompComp)	●	●	●	○	●	●	○	○	○	○	◐	◐	○	○	○
Functional Lists (Generalized Stream Fusion)	●	●	●	○	●	●	◐	○	○	○	●	◐	○	●	●
Functional APL (Futhark, SAC)	◐	◐	◐	○	●	◐	●	○	○	○	●	◐	◐	●	●
Dense LA Library (NumPy)	○	○	○	○	●	○	●	○	○	○	○	○	○	○	●
Dense LA DSL (Lift,Halide,LGen)	○	○	○	○	●	○	●	○	○	○	●	◐	○	●	●
Sparse LA Library (SPLATT, SciPy)	○	○	○	○	●	○	●	◐	○	○	○	○	○	○	◐
Sparse LA DSL (TACO)	○	○	○	○	●	○	●	●	○	○	◐	◐	○	●	○
DB/LA by casting to LA (Morpheus)	◐	○	●	●	●	●	●	○	○	○	○	○	○	○	●
DB/LA by casting to DB (LMFAO)	●	○	●	●	◐	●	●	◐	○	●	◐	◐	○	●	○
DB/LA by new DSL (IFAQ)	●	○	●	●	●	●	○	●	●	●	◐	●	◐	●	○