Functional Collection Programming with Semi-Ring Dictionaries

Amir Shaikhha, Mathieu Huot, Jaclyn Smith, Dan Olteanu

2022/8/3

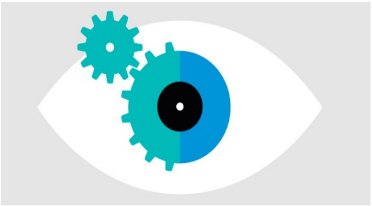
Data Science Workloads

DB Workloads

Data Warehouses (OLAP)

LA Workloads

Machine Learning

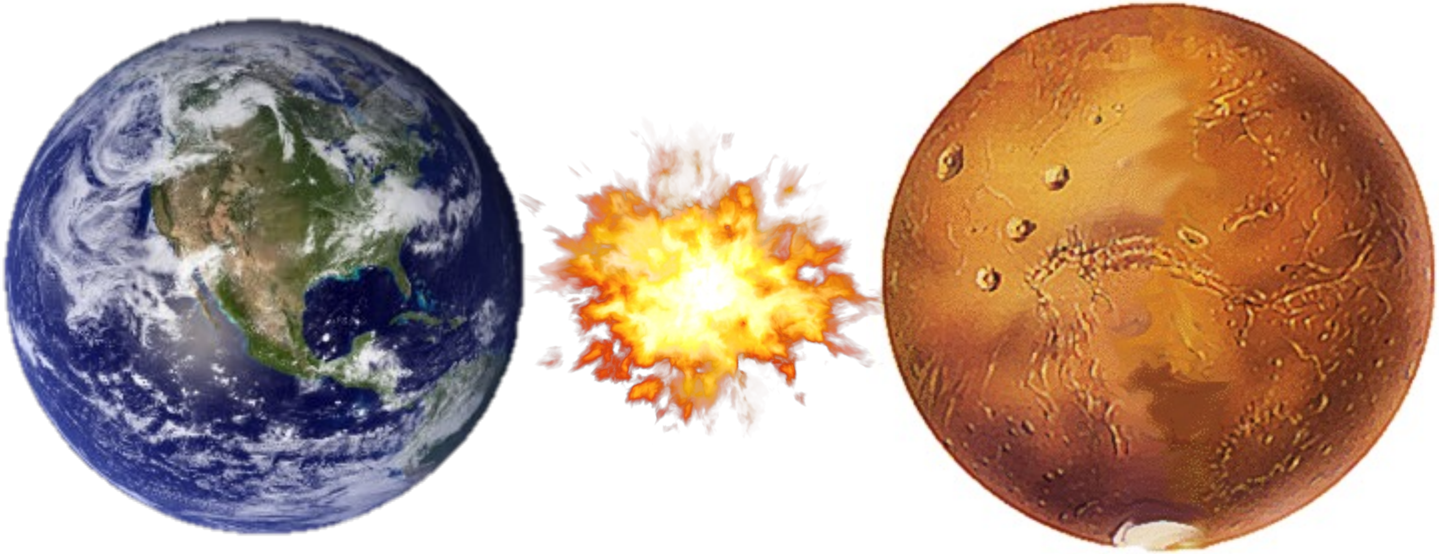


Computer Vision

Scientific Computing

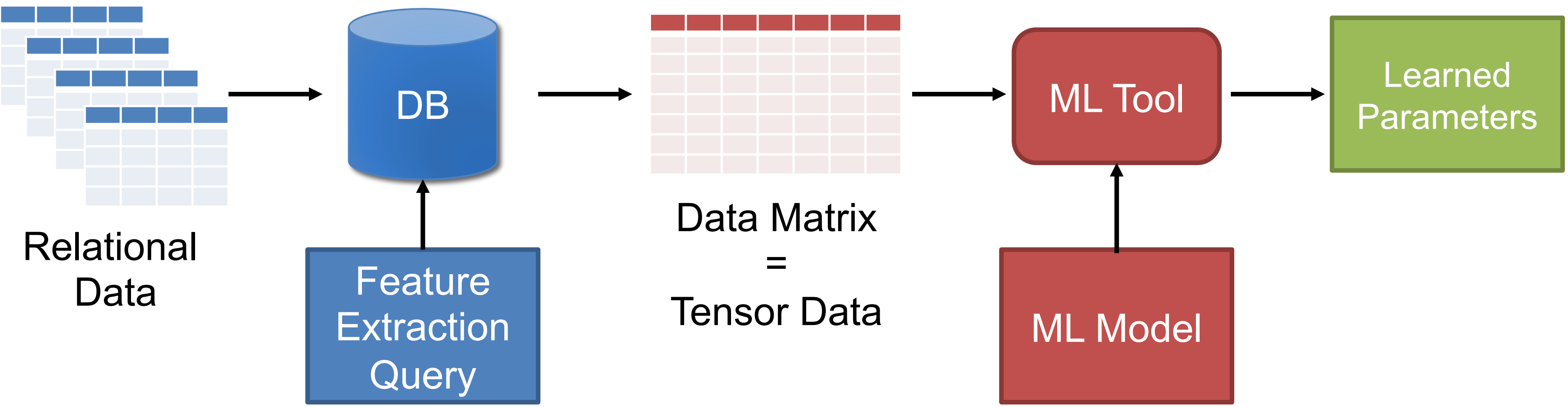
Graph Processing

Data Science

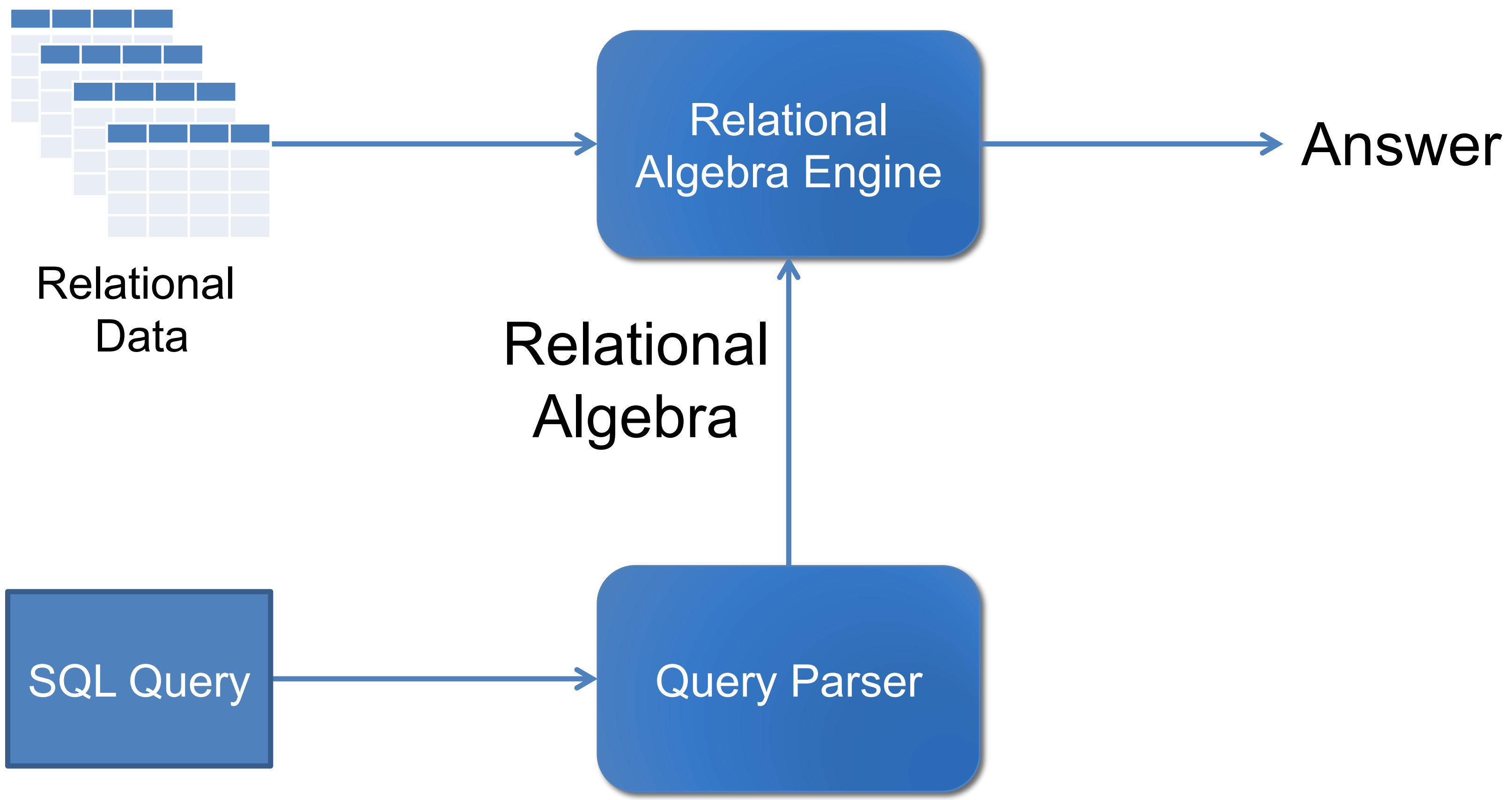


DB Workloads	LA Workloads
Relational Algebra Nested Relational Algebra	Linear Algebra Tensor Algebra
RDBMS, Pandas DataFrame	TensorFlow, PyTorch, scipy

End-to-End Data Science



Relational DB



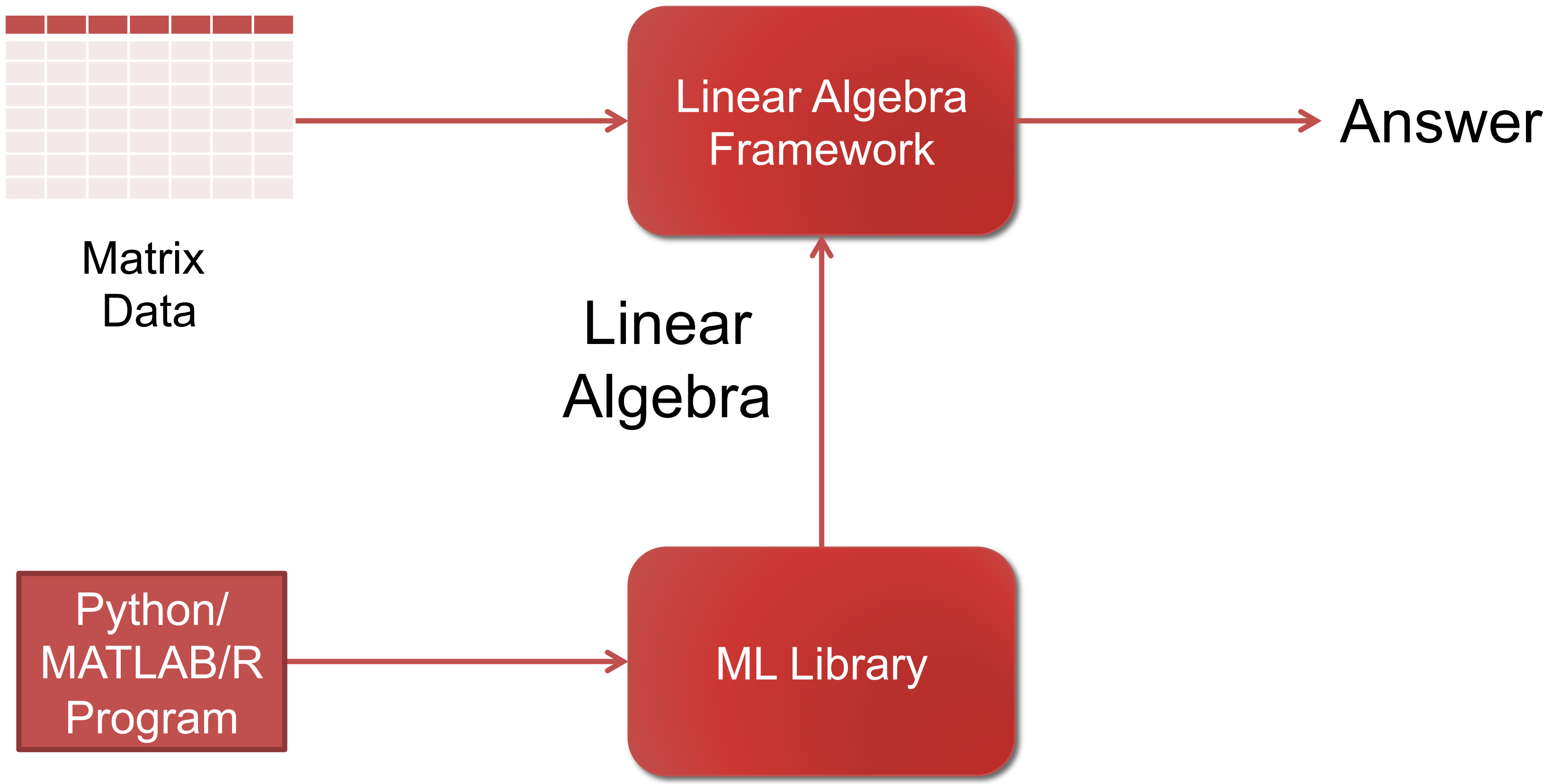
Relational Algebra

- 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

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

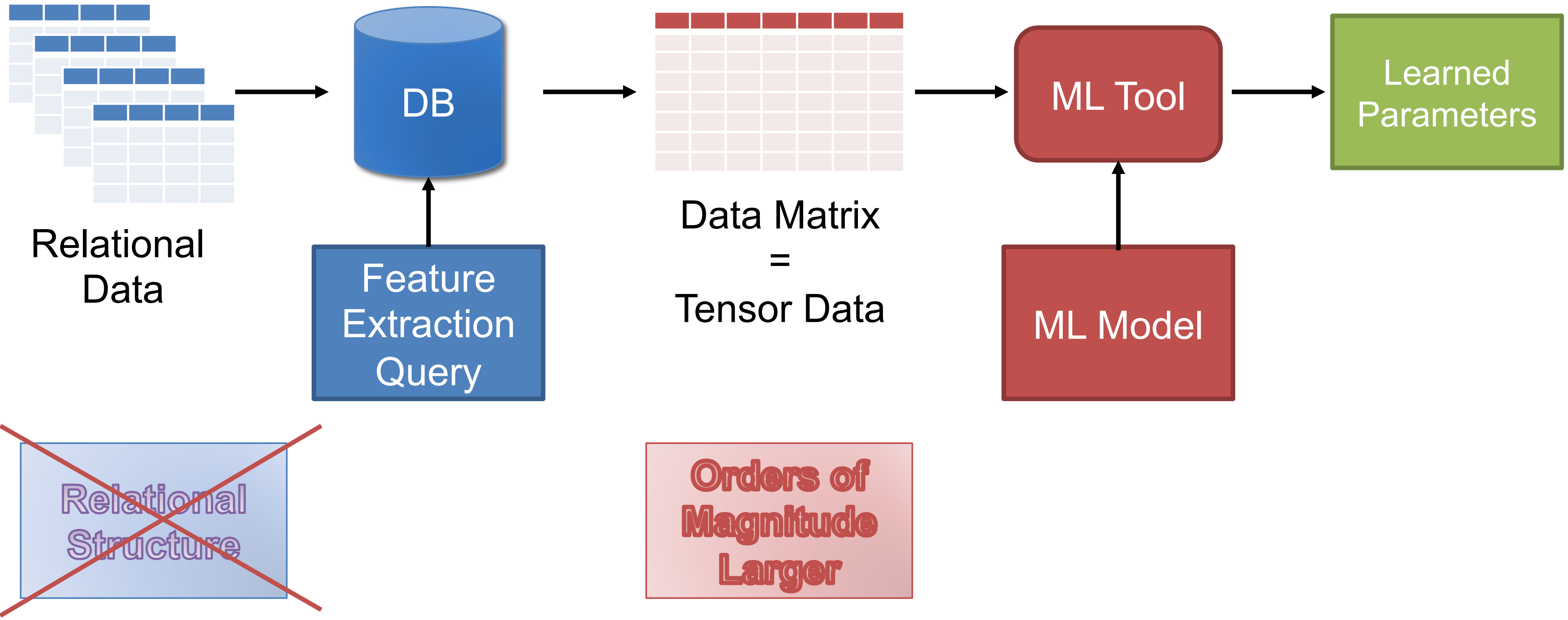
ML Frameworks



Linear Algebra Optimizations

- $M1 + M2 = M2 + M1$
- $M1 + 0 = 0 + M1 = M1$
- $M \times I = I \times M = M$
- $M \times 0 = 0 \times M = 0$
- $M1 \times (M2 \times M3) = (M1 \times M2) \times M3$
- $M1 \times (M2 + M3) = M1 \times M2 + M1 \times M3$
- ...

Issues with Pipelines for Data Science



- Materialize query result
- Export from DBMS and import into ML tool

Can we have a unified environment?

Approaches

- In-DB ML as LA
  - Morpheus
- In-DB ML as DB
  - LMFAO
- In-DB ML as new language
  - IFAQ

Similarity of Optimizations

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

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

FAQ: Questions Asked Frequently

Mahmoud Abo Khamis[*†] Hung Q. Ngo[*†] Atri Rudra[†]

[*] LogicBlox Inc.
{mahmoud.abokhamis,hung.ngo}@logicblox.com

[†] Department of Computer Science and Engineering
University at Buffalo, SUNY
{mabokham,hungngo,atri}@buffalo.edu

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

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

Pushing aggregates past joins

Matrix chain ordering

SDQL

Semi-Ring Dictionary Query Language

Semi-Ring

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

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

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

Semi-Ring Examples

- Real and natural numbers
  - $a \times (b + c) = (a \times b) + (a \times c)$
- Boolean
  - $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$
- Tropical semi-ring
  - $a + \max(b, c) = \max(a + b, a + c)$
  - $a + \min(b, c) = \min(a + b, a + c)$

Semi-Ring Dictionaries

One collection to rule them all

`Relation[T] = Dict[T, Bool]` *(no duplicates)*
`Relation[T] = Dict[T, Int]` *(with duplicates)*

`Vector[T] = Dict[Int, T]`
`Matrix[T] = Dict[(Int, Int), T]`

Database Relations (Bag Semantics)

- Dictionaries of tuples to multiplicities

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

Linear Algebra (Matrix)

- Dictionaries of indices to values

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

SDQL

Core Grammar

e ::= **sum**(x **in** e) e | { e -> e, ... }

| $\{\ \}_{T,T}$ | e(e)

| < a = e, ... > | e.a | **not** e

| **let** x = e **in** e | x | **if** e **then** e **else** e

| e + e | e * e | **promote**$_{S,S}$(e)

| n | r | **false** | **true** | c

T ::= { T -> T } | < a:T, ... > | S | U

S ::= **int** | **real** | **bool** | [cf. Table 1]

U ::= **string** | **dense_int**

K ::= Type | SM(S)

SDQL Examples

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

$[[V_1 + V_2]] = [[V_1]] + [[V_2]]$

$[[a \cdot V]] = [[a]] * [[V]]$

$[[V_1 \circ V_2]] =$ `sum(x in` $[[V_1]]$ `){ x.key -> x.val *` $[[V_2]]$`(x.key)}`

$[[V_1 \cdot V_2]] =$ `sum(x in` $[[V_1]]$ `)x.val *` $[[V_2]]$`(x.key)`

$[[\sum_{a \in V} a]] =$ `sum(x in` $[[V]]$ `)x.val`

$[[M_1^T]] =$ `sum(row in` $[[M_1]]$ `)sum(x in row.val)`
`{ x.key -> { row.key -> x.val } }`

$[[M_1 \circ M_2]] =$ `sum(row in` $[[M_1]]$ `){ row.key ->`
`sum(x in row.val){ x.key ->`
`x.val*`$[[M_2]]$`(row.key)(x.key)} }`

$[[M_1 \times M_2]] =$ `sum(row in` $[[M_1]]$ `){ row.key ->`
`sum(x in row.val)sum(y in` $[[M_2]]$`(x.key))`
`{ y.key -> x.val * y.val } }`

$[[M \cdot V]] =$ `sum(row in` $[[M]]$ `){ row.key ->`
`sum(x in row.val)x.val *` $[[V]]$`(x.key)}`

$[[Trace(M)]] =$ `sum(row in` $[[M]]$ `)row.val(r.key)`

Min/Max aggregations

```
SELECT MIN(A) FROM R
```

```
sum(<key, val> in R) promote[min_sum](key.A)
```

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

```
sum(<key, val> in R) { key.B -> promote[max_sum](key.A) }
```

Semi-Ring types

Name	Type	Domain	Addition	Multiplication	Zero	One	Ring
Real Sum-Product	`real`	$\mathbb{R}$	$+$	$\times$	`0`	`1`	✓
Integer Sum-Product	`int`	$\mathbb{Z}$	$+$	$\times$	`0`	`1`	✓
Natural Sum-Product	`nat`	$\mathbb{N}$	$+$	$\times$	`0`	`1`	✗
Min-Product	`mnpr`	$(0, \infty]$	min	$\times$	∞	`1`	✗
Max-Product	`mxpr`	$[0, \infty)$	max	$\times$	`0`	`1`	✗
Min-Sum	`mnsm`	$(-\infty, \infty]$	min	$+$	∞	`0`	✗
Max-Sum	`mxsm`	$[-\infty, \infty)$	max	$+$	$-\infty$	`0`	✗
Max-Min	`mxmn`	$[-\infty, \infty]$	max	min	$-\infty$	$+\infty$	✗
Boolean	`bool`	$\{T, F\}$	$\vee$	$\wedge$	`false`	`true`	✗

Loop Optimizations

- Vertical Loop Fusion
- Horizontal Loop Fusion
- Loop-invariant code motion
- Loop factorization
- Loop memoization

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 }
```

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

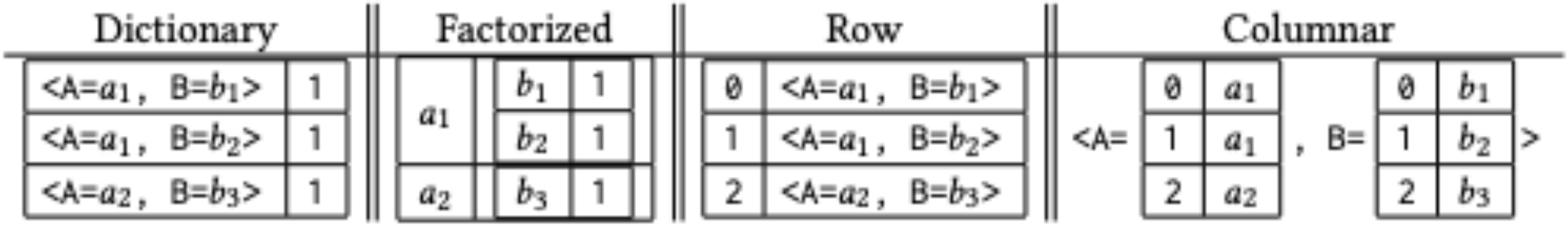
- Loop Factorization + Memoization

Hash Join, Group Join | Matrix chain ordering

Data Layouts

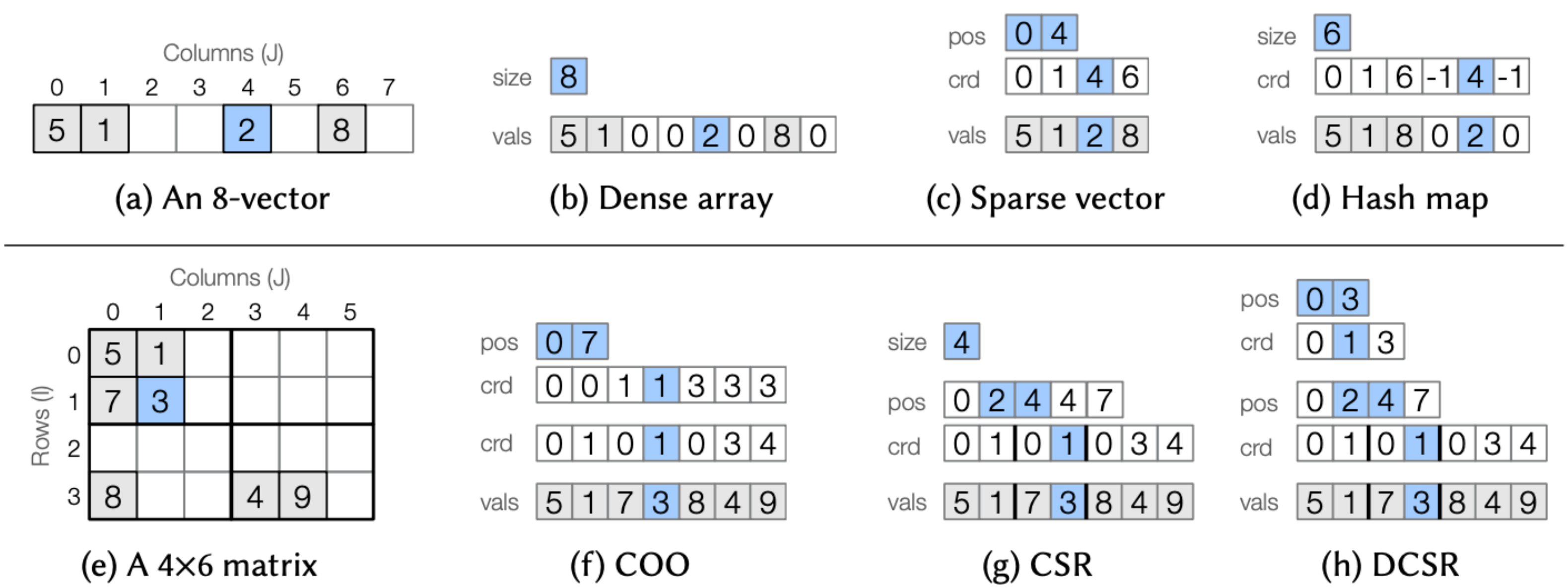
- Relations
  - Row/Columnar layout
  - Standard Dictionary
  - Factorized (by Tries)

- Matrices
  - Dense (Row/Column Major)
  - COO
  - Compressed (by Tries)



Sparse Tensors

TACO: The Tensor Algebra Compiler



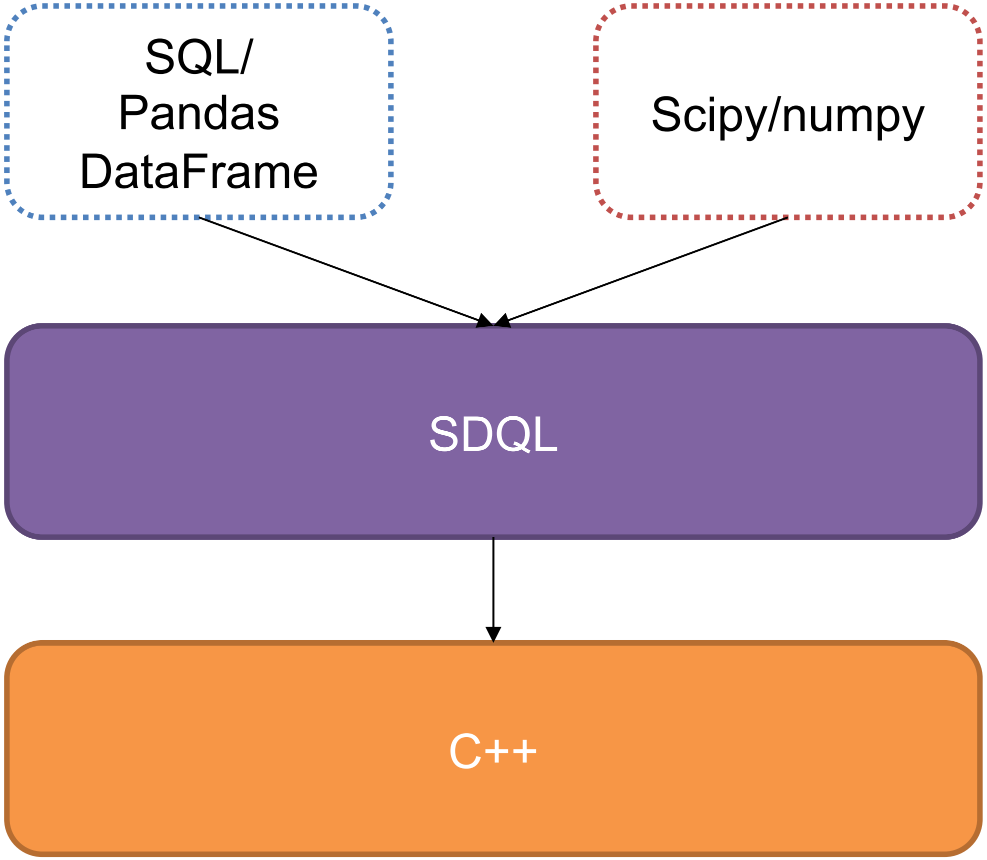
Can be subsumed by SDQL using nested dictionaries

Semi-Ring Dictionaries

One collection to rule them all

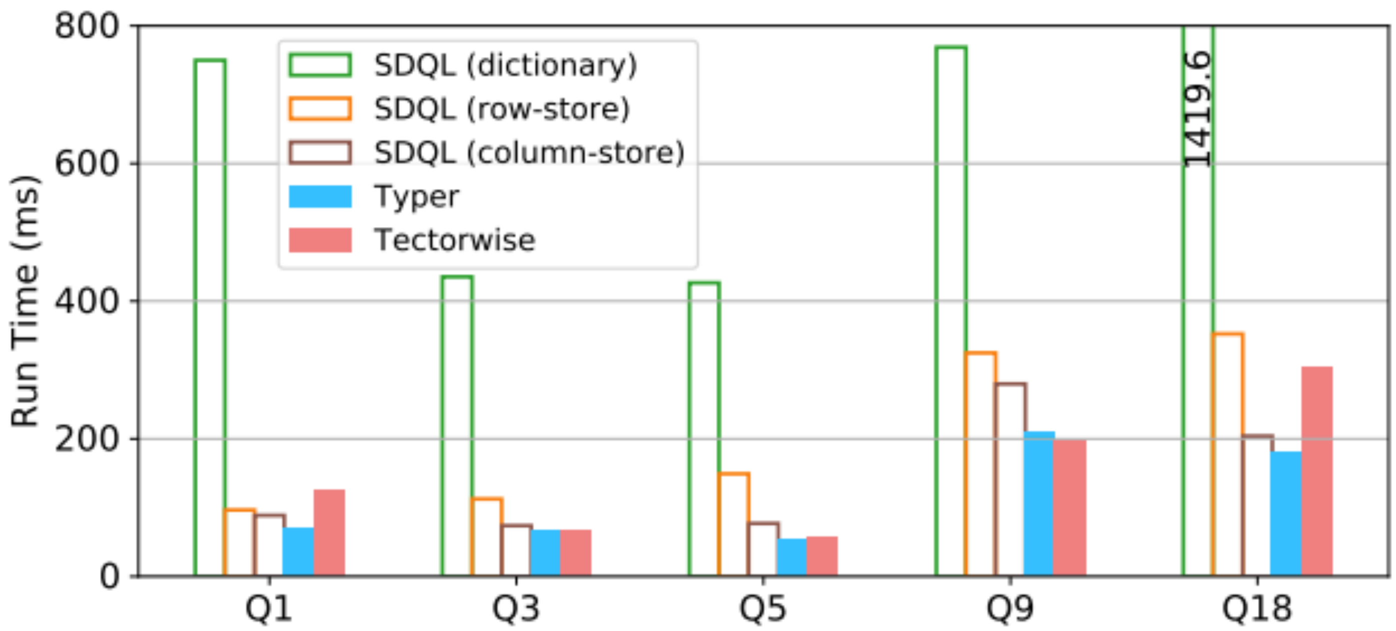
- Relation[T]
  - `Bag{T}` = `Dict{T, Int}`
  - `Set{T}` = `Dict{T, Bool}`
- Nested Relations
  - `Bag{Bag{T}}` = `Dict{Dict{T, Int}, Int}`
  - `Set{Set{T}}` = `Dict{Dict{T, Bool}, Bool}`
- Tensors
  - `SparseVector{T}` = `Dict{Int, T}`
  - `SparseMatrixCOO{T}` = `Dict{(Int, Int), T}`
  - `SparseMatrixTrie{T}`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

Compilation Pipeline



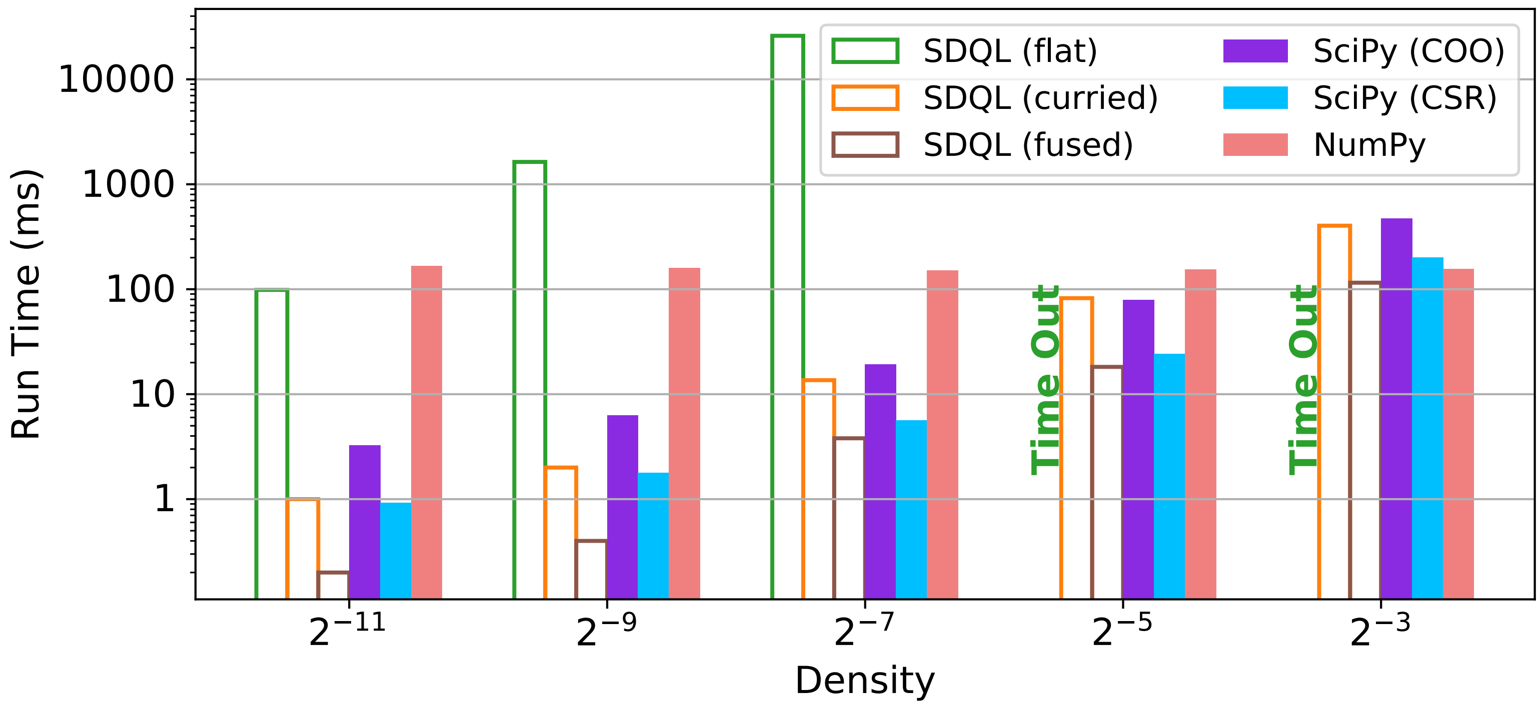
DB Experiments

- Typer: Open-source version of HyPer
  - query compilation-based
- Tectorwise: Open-source version of Vectorwise
  - vectorization-based



LA Experiments

- SciPy
- NumPy



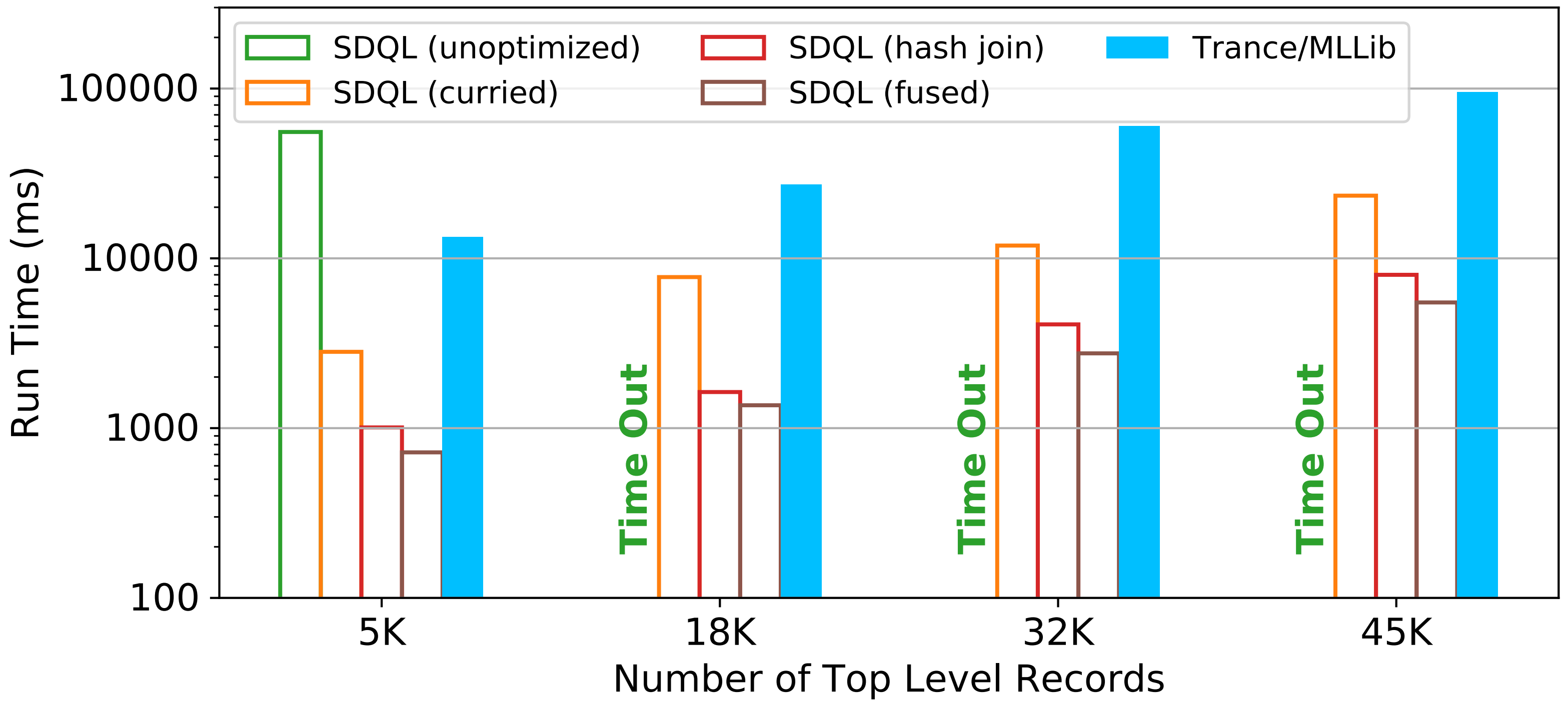
LA Experiments

- Taco: state-of-the-art sparse tensor compiler

	Sparsity	2^{-11}		2^{-9}		2^{-7}		2^{-5}		2^{-3}	
Kernel	LA Formulation	SDQL	taco	SDQL	taco	SDQL	taco	SDQL	taco	SDQL	taco
TTV	$A_{ij} = \Sigma_k B_{ijk} c_k$	621.8	466.3	621.8	544.9	632.0	866.2	661.8	2088.1	729.4	6742.7
TTM	$A_{ijk} = \Sigma_k B_{ijl} C_{kl}$	4534.2	5936.2	4679.6	7851.6	4764.2	15563.9	5189.2	46153.7	7146.6	169865.5
MTTKRP	$A_{ij} = \Sigma_{k,l} B_{ikl} C_{kj} D_{lj}$	5.6	4.3	18.4	17.3	32.2	60.4	103.2	388.1	723.8	4371.1

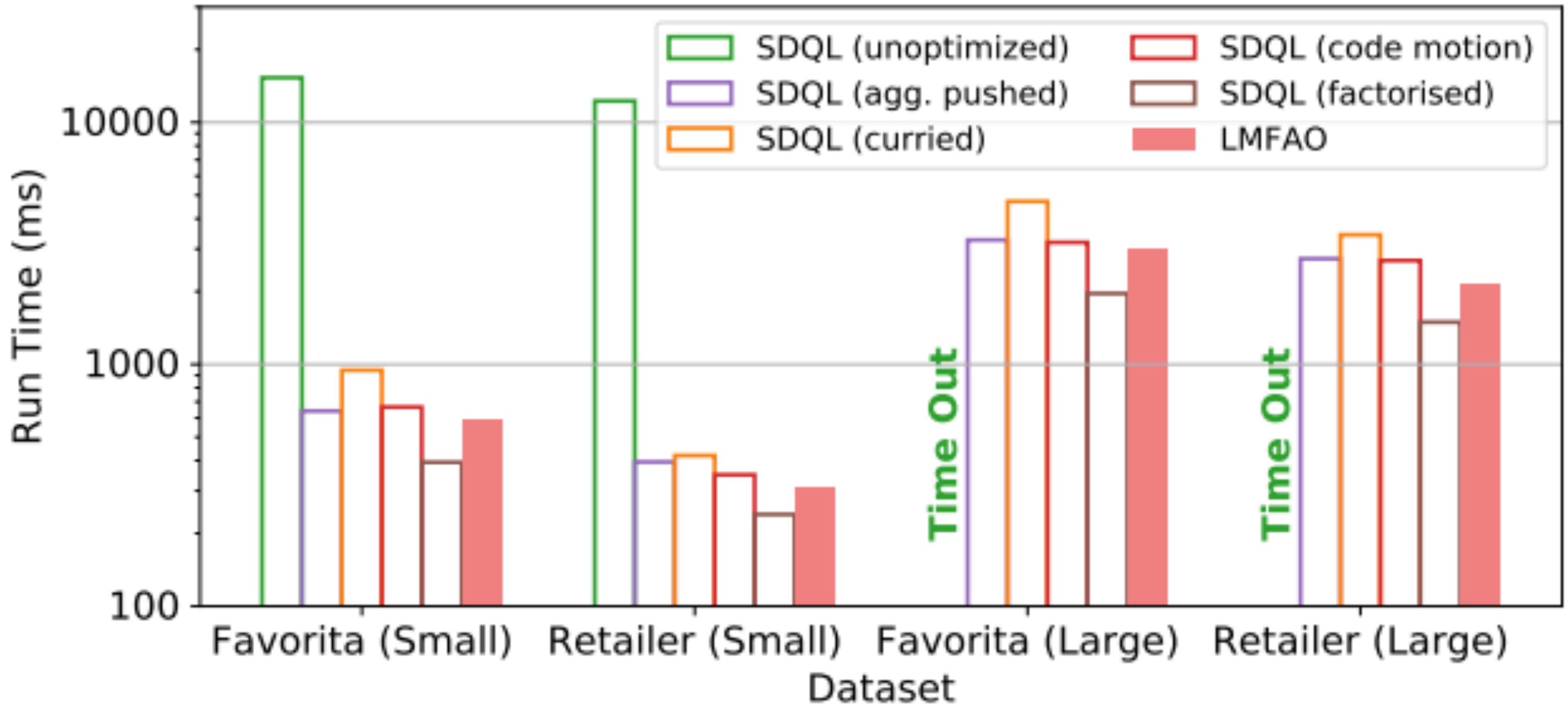
DB & LA Experiments

- Trance: DB engine for nested data
- MLLib: ML library



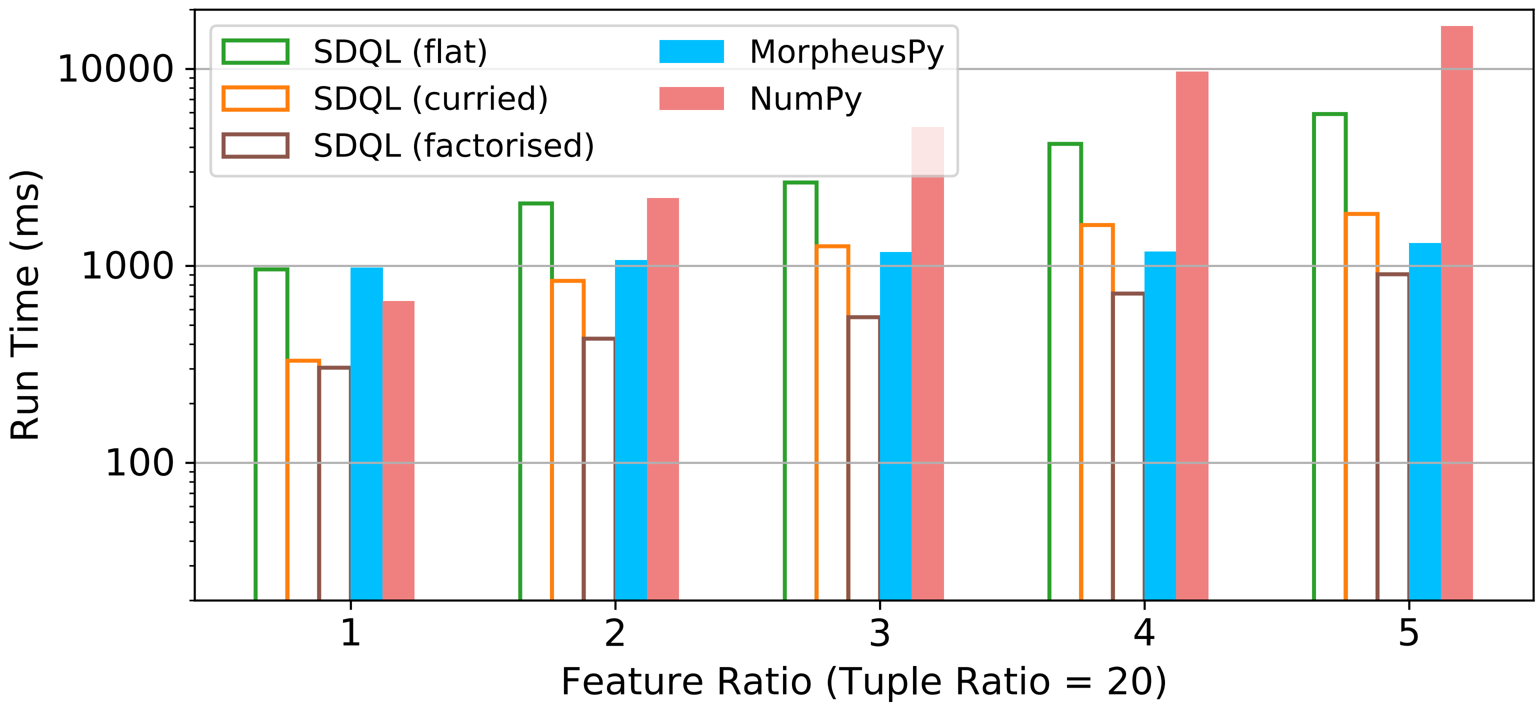
DB & LA Experiments

- LMFAO: DB/LA by DB



DB & LA Experiments

- Morpheus: DB/LA by LA



Current Work – Ordered Dictionaries

- Dictionaries with hash tables
- Dictionaries with ordered tables
  - Balanced trees
  - Sorted arrays
- Support for Parallelization

Hinted Dictionaries: Efficient Functional Ordered Sets and Maps

Amir Shaikhha ✉
University of Edinburgh, UK

Mahdi Ghorbani ✉
University of Edinburgh, UK

Hesam Shahrokhi ✉
University of Edinburgh, UK

ECOOP'22

Conclusion

- Semi-ring-based language
  - Relational Algebra
  - Nested Relational Algebra
  - Linear Algebra
- Optimizations inside and across DB/LA
- Competitive with specialized systems
  - Sparse Linear Algebra
  - Analytical Databases
  - In-Database Machine Learning

THANK YOU

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)	●	○	●	●	●	●	○	●	●	●	◐	●	◐	●	○