Optimizing Tensor Computations:
From Applications to Compilation and Runtime Techniques

Matthias Boehm
TU Berlin
Berlin, Germany

Matteo Interlandi
Microsoft GSL
Los Angeles, CA, USA

Chris Jermaine
Rice University
Houston, TX, USA

SIGMOD 2023

Fri Jun 23, 9am – 10.30am

Who We Are

Matthias Boehm
TU Berlin
Berlin, Germany

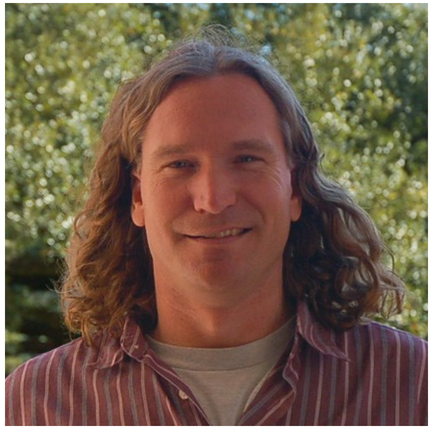
APACHE SOFTWARE FOUNDATION

SystemML → SystemDS

DAPHNE

Matteo Interlandi
Microsoft
Los Angeles, CA, USA

TQP



Chris Jermaine
Rice University
Houston, TX, USA

SimSQL, **MCDB**

TRA

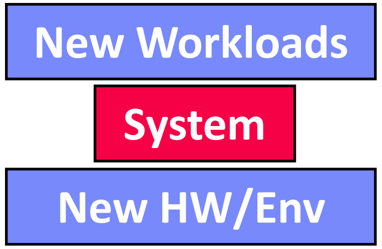
Agenda

- **A Case for Tensor Computations** [15min]
- **Selected Applications** [35min]
  - Query Processing and Data Analytics
  - Data Science Lifecycle Tasks
  - Simulation and Sampling
- **Selected Runtime Backends** [35min]
  - Tensor Query Processor (TQP)
  - Apache SystemML/SystemDS
  - Tensor Relational Algebra

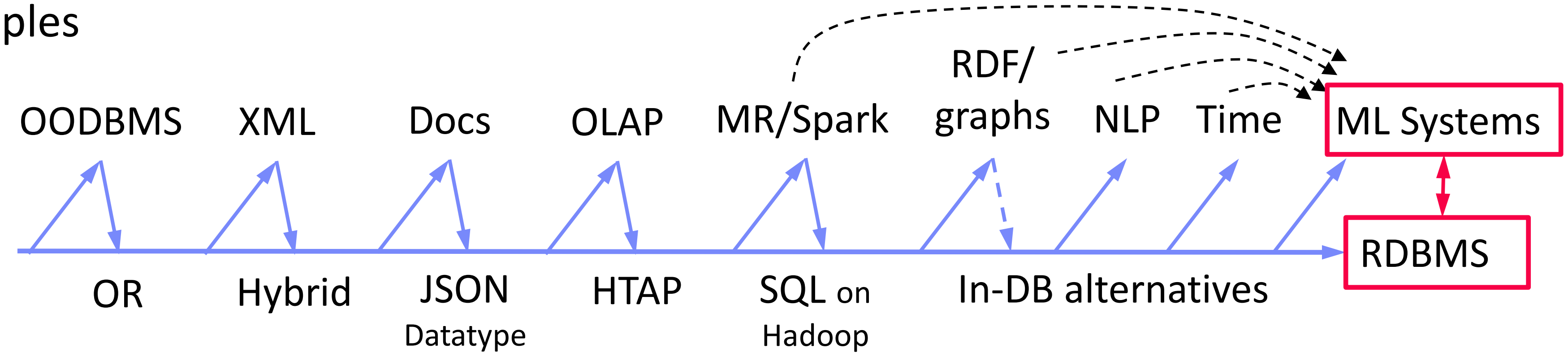
A Case for
Tensor Computations

Motivation: A Historic Perspective

- **Two Key Drivers of DB Research**
  - **New analysis workloads** (NLP, key/value, RDF/graphs, documents, time series, ML) and applications
  - **New HW/infrastructure** (multi-/many-core, cloud/serverless, scale-up/out, NUMA/HBM, RDMA, SSD/NVM, FPGA/GPU/ASIC)

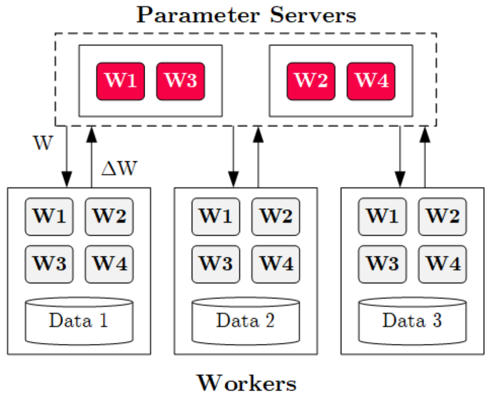


- **Past: Waves of General-purpose and Specialized Systems**
  - **Goal #1**: Avoid boundary crossing ➔ **General-purpose**
  - **Goal #2**: New workload + Performance ➔ **Specialized systems**
  - Some Examples

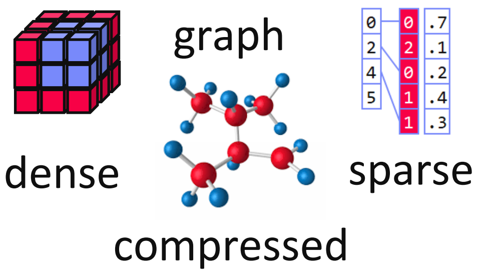


Motivation: Tensor Computations

- **Present: Narrow Focus (on Mini-batch SGD)**
  - Increasingly narrow training/inference focus on deep neural networks (DNN) mini-batch stochastic gradient descent (SGD)
  - Parameter servers and similar distribution strategies
  - Communication, security, acceleration primitives for narrow focus



- **Future: Broader Focus (on General Tensor Computations)**
  - General linear algebra programs and tensor computations
  - Different architectures (parameter servers, data- & task-task parallel, hybrid, recursive)
  - Wide variety of applications and workload characteristics

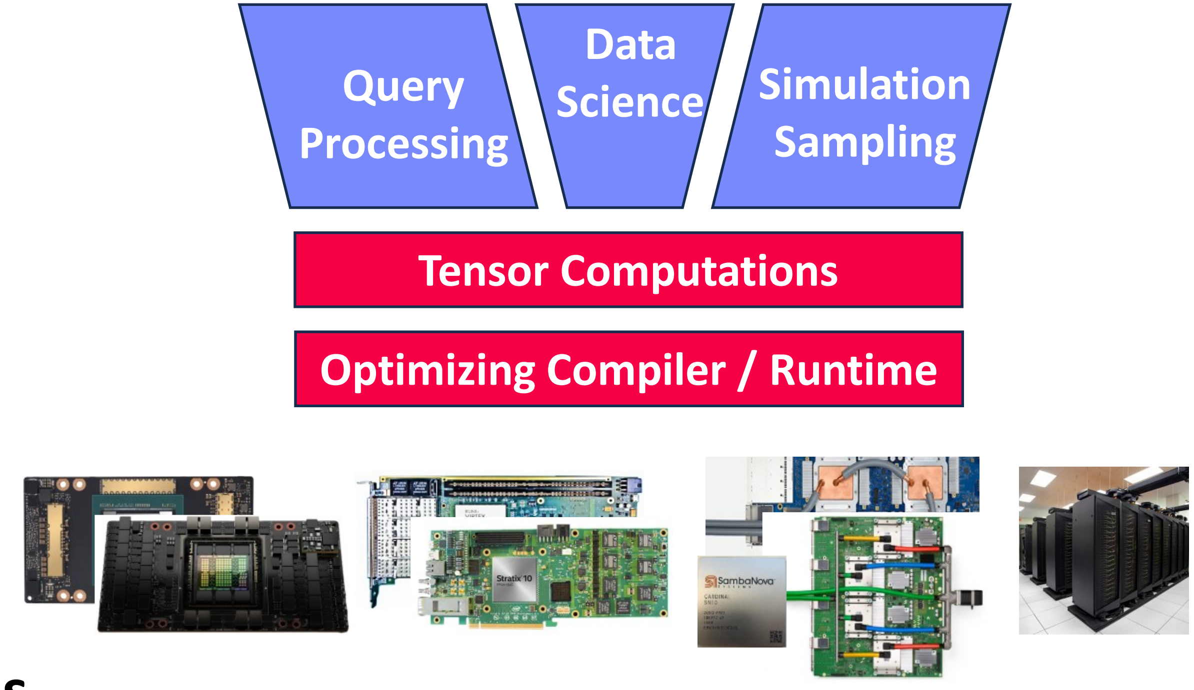


OLAP Queries / Data Frames / ML Systems

	DBMS	Data Frames	ML Systems
Language Abstraction	SQL	Relational Algebra ++	Linear Algebra ++
Workload	Repeated Queries	Explorative Operations	Iterative Algorithms
Infrastructure	Server (but RAWdb, DuckDB)	Stateless library (embedded)	Stateless library (embedded, but Serving)
Optimization	Join ordering, rewrites Phy. op selection Query compilation		mmchain opt, rewrites Phy. op selection Operation fusion/codegen
Runtime	Scans, large-small joins, aggregations; vectorization	Coarse-grained frame operations	Coarse-grained tensor operations vectorization / mini-batches
	Variety of value types		Increasing number of specialized value types
Storage	Page layouts w/ SMAs and compression	Arrays, open formats (e.g., Arrow, Parquet)	Dense / sparse / compressed blocks

➔ **Increasing Convergence of Optimization and Runtime Techniques**

Long-term Benefits



- **#1 Simplicity**
  - Coarse-grained frame/matrix/tensor data structures and operations
  - Reduced system infrastructure complexity (boundary crossing)
- **#2 Reuse of Compiler/Runtime Techniques**
  - Focused work and reuse of commonly used compiler/runtime techniques
  - Generality over hand-crafted specialized systems and algorithms
- **#3 Performance and Scalability**
  - Leverage HW Accelerators and distributed runtime backends ➔ Increasing specialization and rapid evolution
  - Homogeneous arrays and simple parallelization strategies

Build Libraries for Tensor Ops on HW X once and reuse

Selected Applications

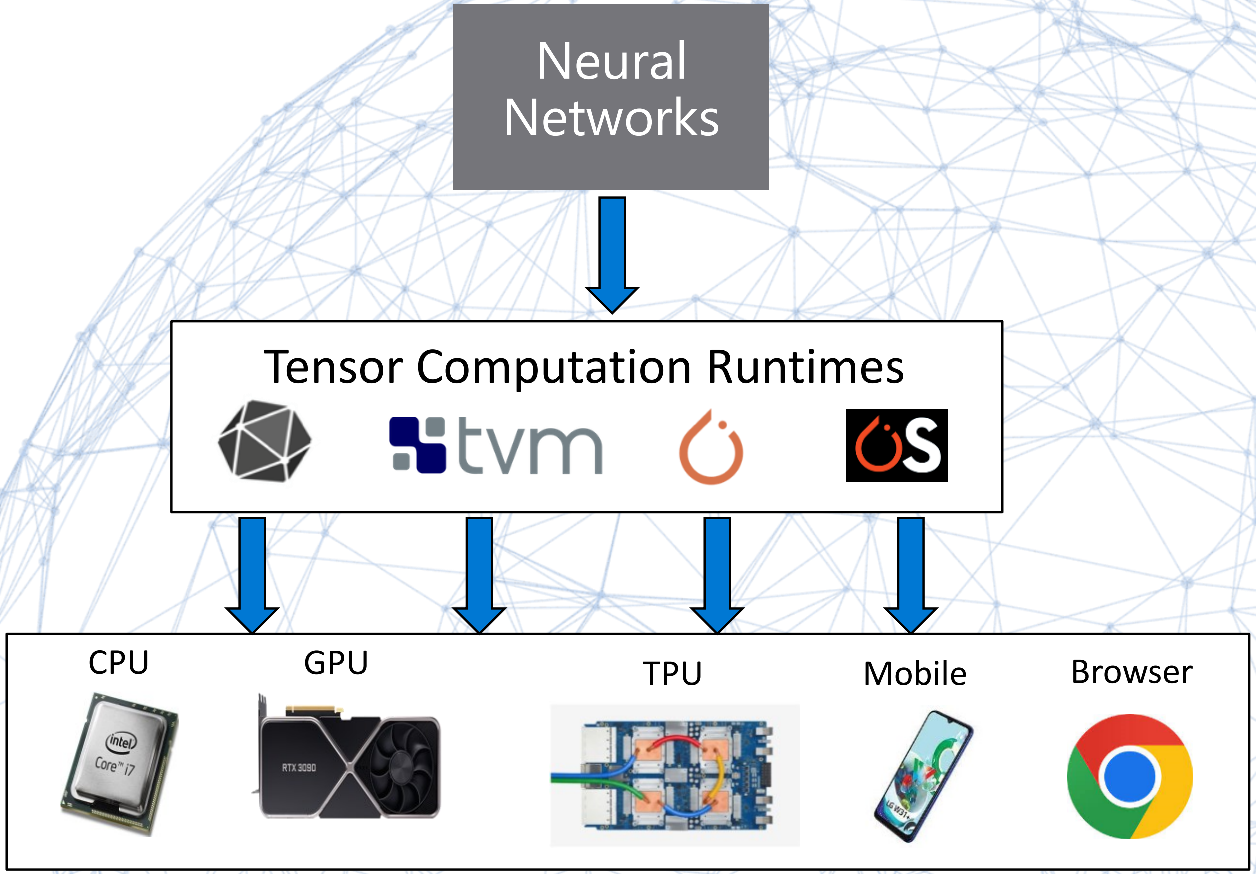
Query Processing and Data Analytics

Data Science Lifecycle Tasks

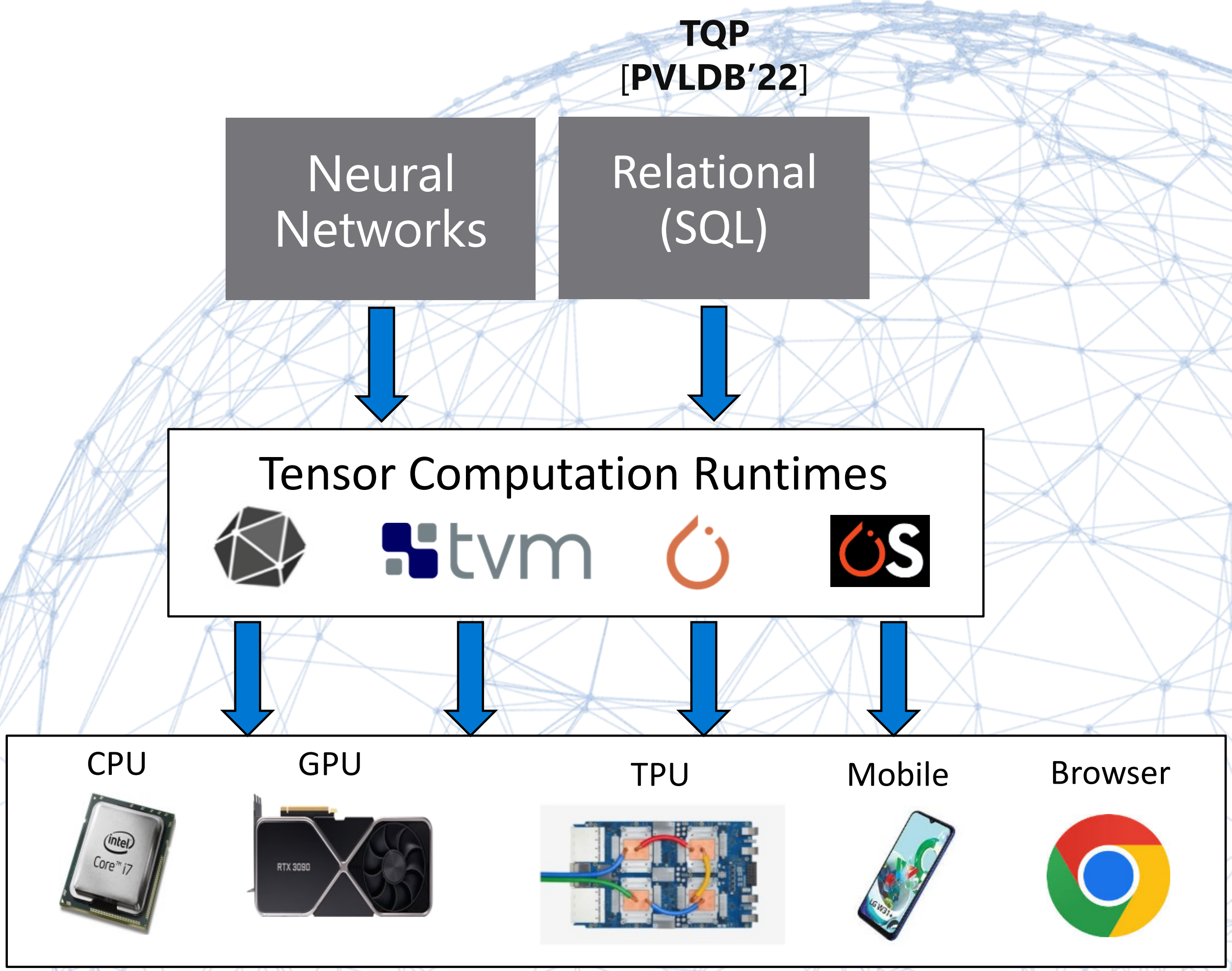
Simulation and Sampling

[35 min]

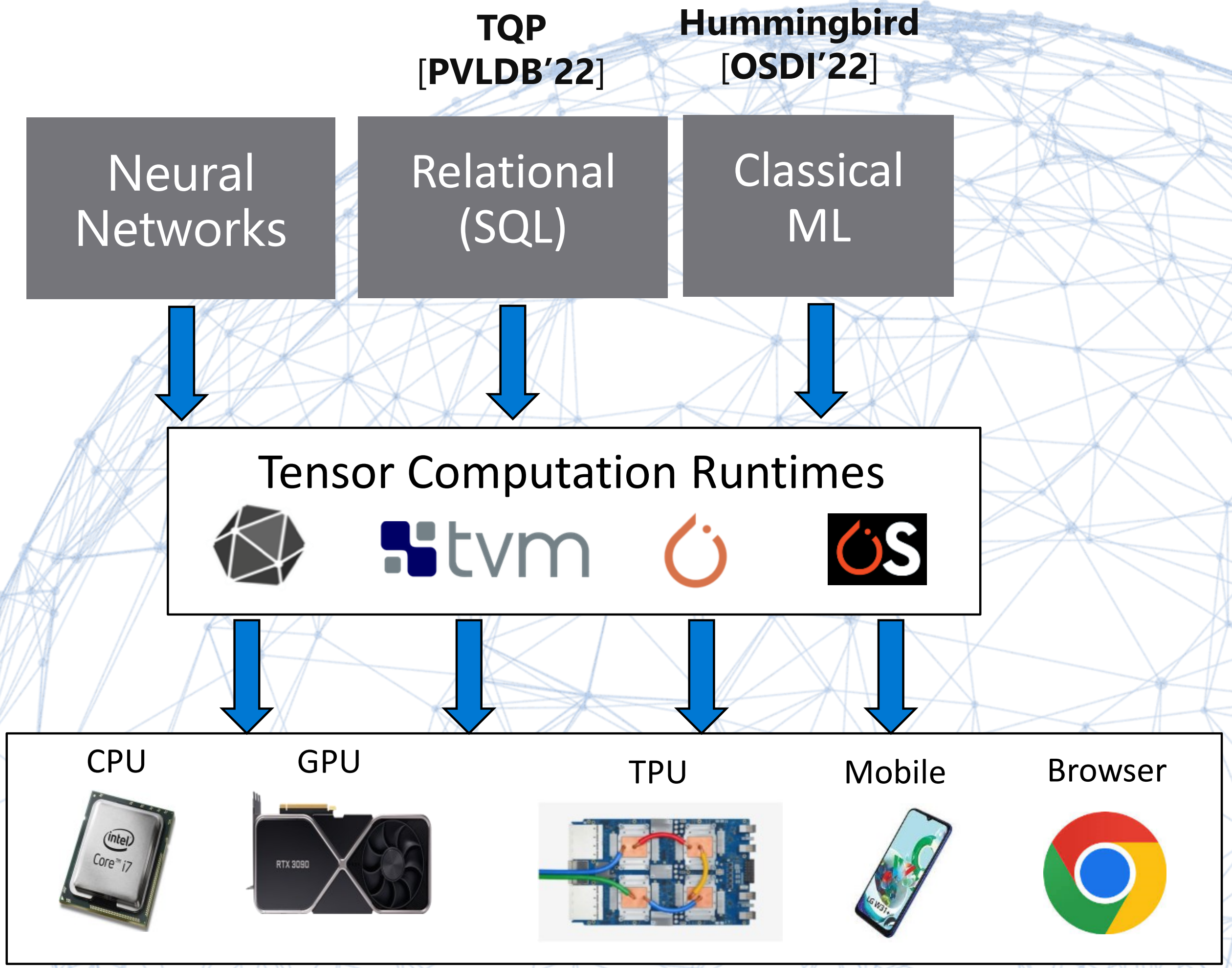
Query Processing and Data Analytics



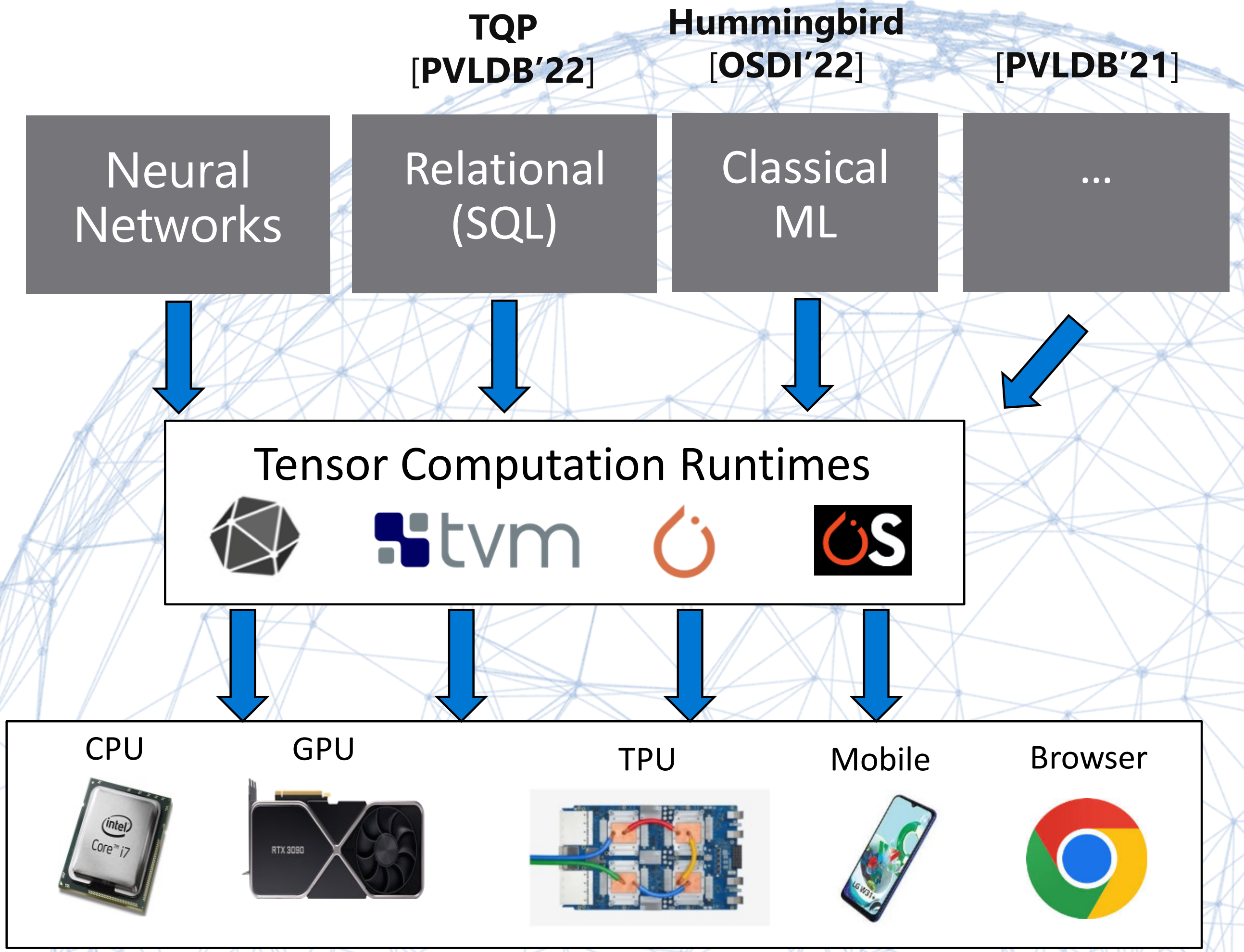
Query Processing and Data Analytics



Query Processing and Data Analytics



Query Processing and Data Analytics

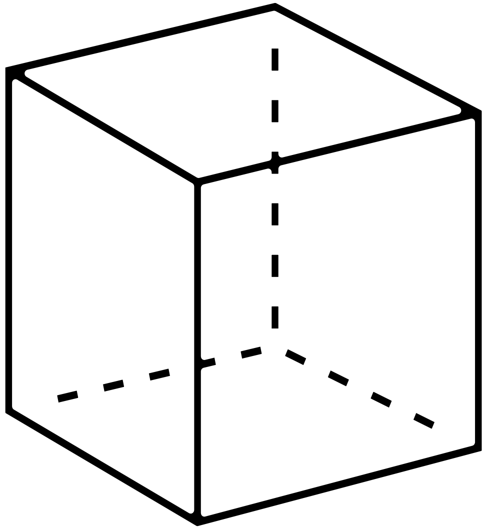


Questions:

1. How do we represent tabular data as tensors?
2. How can we map SQL operations into tensor programs?
3. How can we map traditional ML model into tensor computations?

Tensor data representation

TQP
[PVLDB’22]



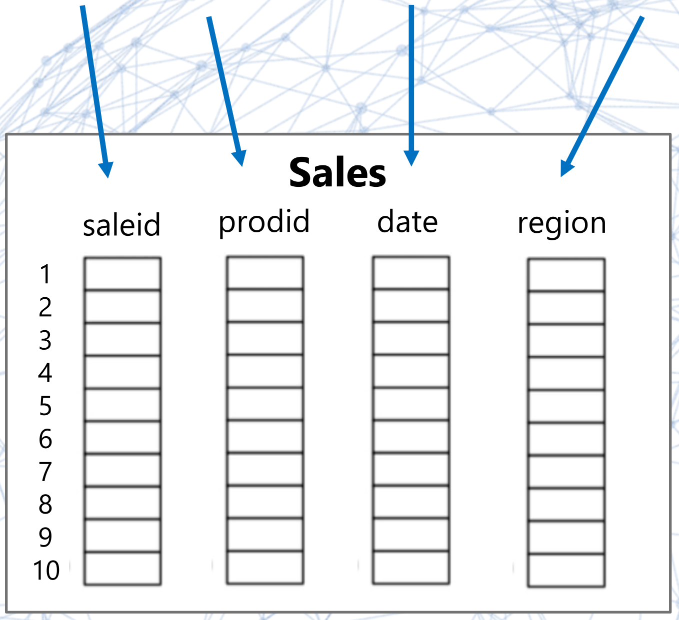
Def Tensor:

A multidimensional matrix that is a cornerstone data structure in AI

Numeric as 1-d tensor

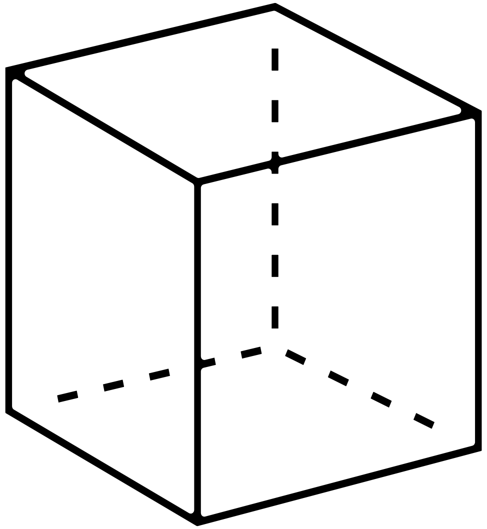
Dates as 1-d numeric tensor

Strings as UTF-8 2-d tensor (N x max_len)



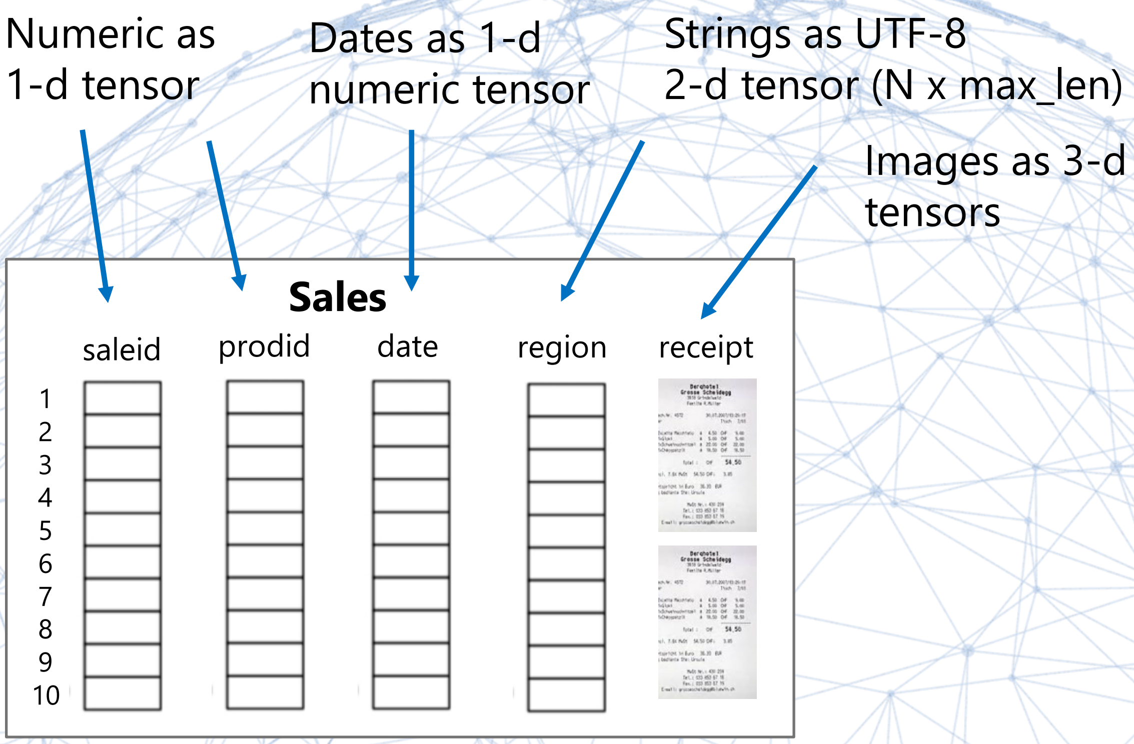
Tensor data representation

TQP
[CIDR'23]



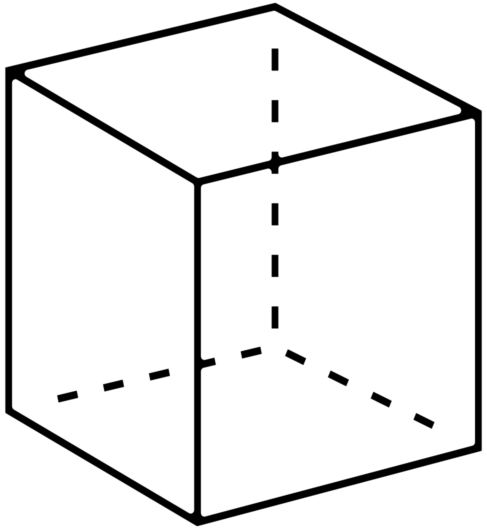
Def **Tensor:**

A multidimensional matrix that is a cornerstone data structure in AI



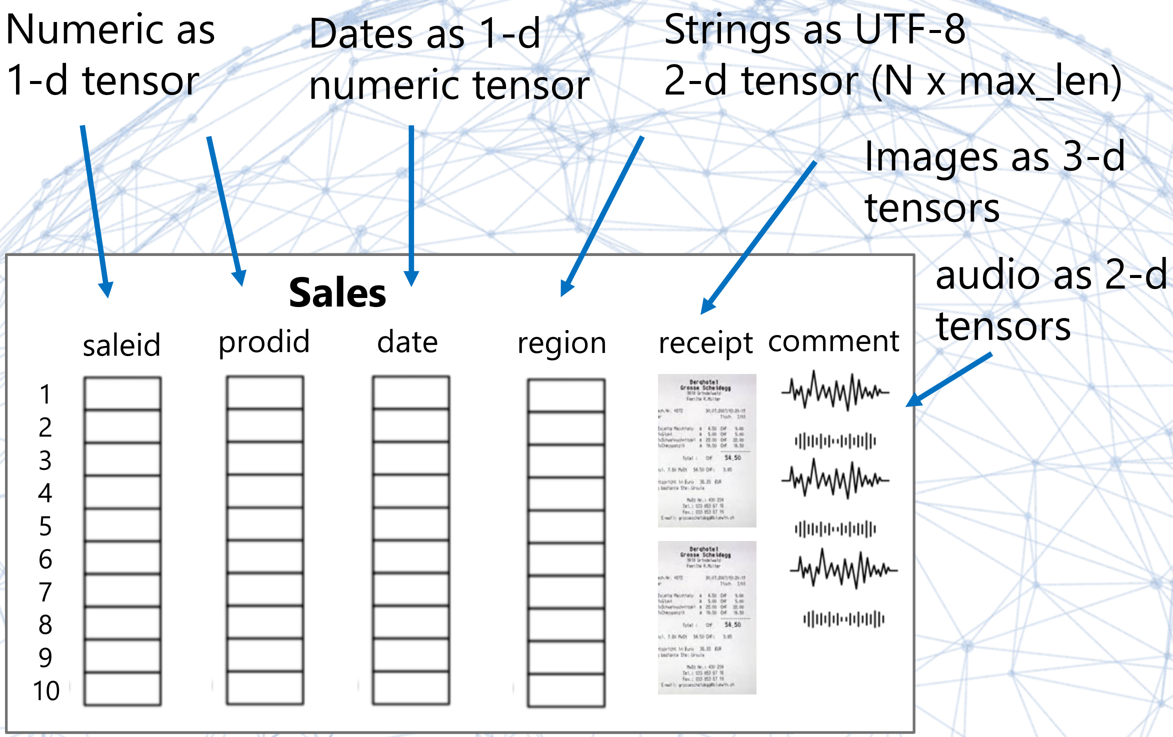
Tensor data representation

TQP
[CIDR'23]



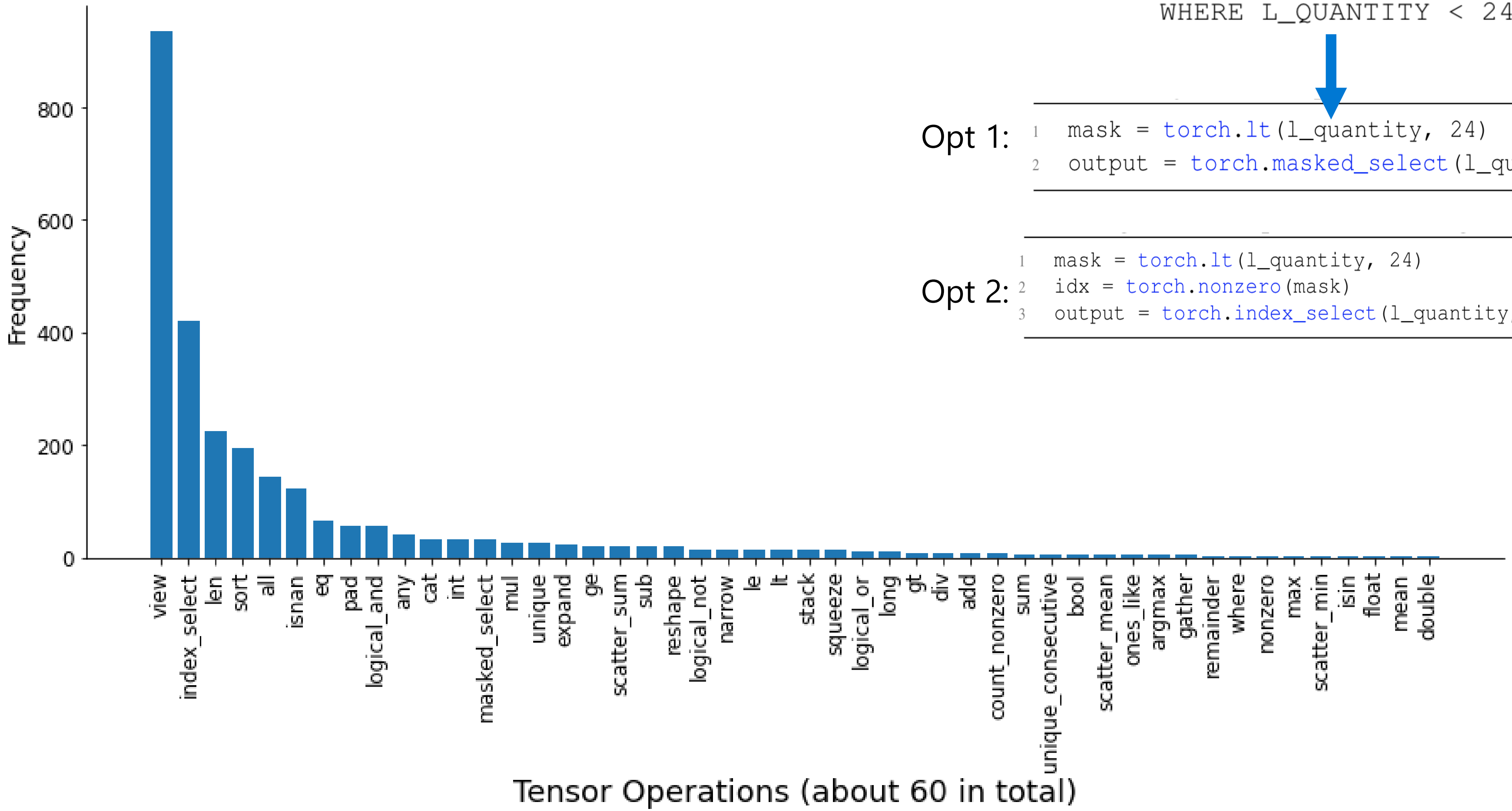
Def **Tensor**:

A multidimensional matrix that is a cornerstone data structure in AI



Implementing SQL operators using tensor ops

TQP
[PVLDB'22]




```
WHERE L_QUANTITY < 24
```

Opt 1:

```
mask = torch.lt(l_quantity, 24)
output = torch.masked_select(l_quantity, mask)
```

Opt 2:

```
mask = torch.lt(l_quantity, 24)
idx = torch.nonzero(mask)
output = torch.index_select(l_quantity, dim=0, idx)
```

Traditional ML: Prediction Serving

Hummingbird
[OSDI'20]

Starred 3.2k

Contributors 37

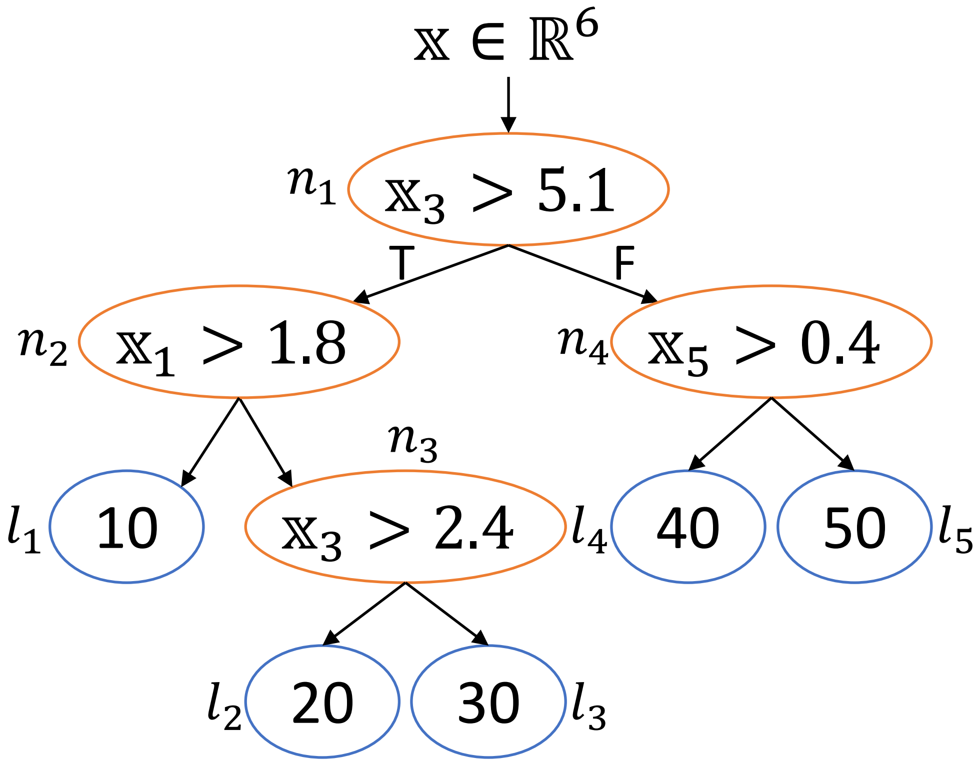
Used by 73

- Traditional ML models are composed by: **featurizers** and **ML models**

- <u>Each featurizer</u> is defined by an **algorithm**
  - e.g., compute the one-hot encoded version of the input feature
- <u>Each trained model</u> is defined by a **prediction** function
  - Prediction functions can be either **algebraic** (e.g., linear regression) or **algorithmic** (e.g., decision tree models)
  - Algebraic models are easy to translate: just implement the same formula in tensor algebra!

Translating Trees

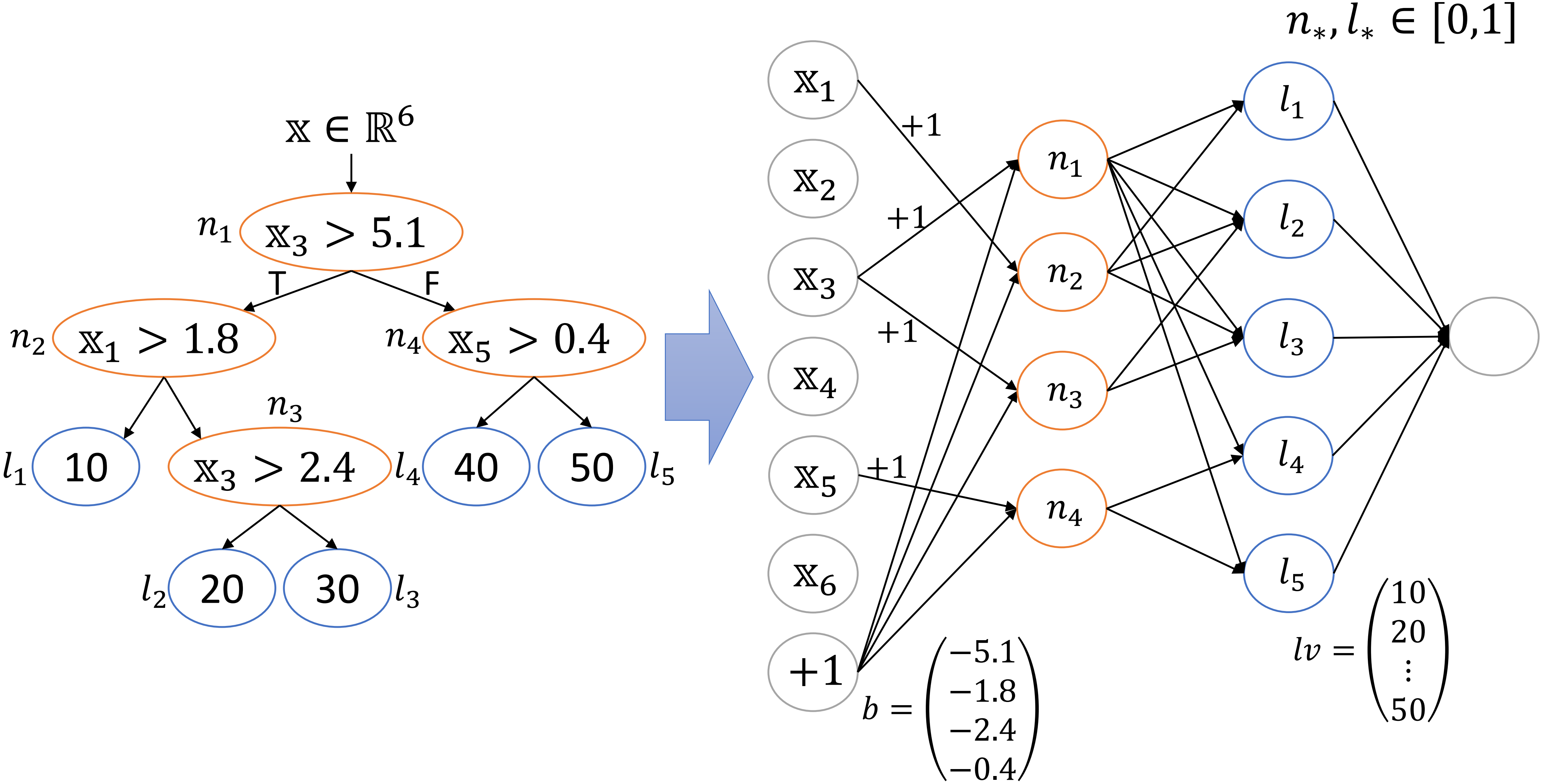
Hummingbird [OSDI'20]



Internal node: n_*
Leaf node: l_*

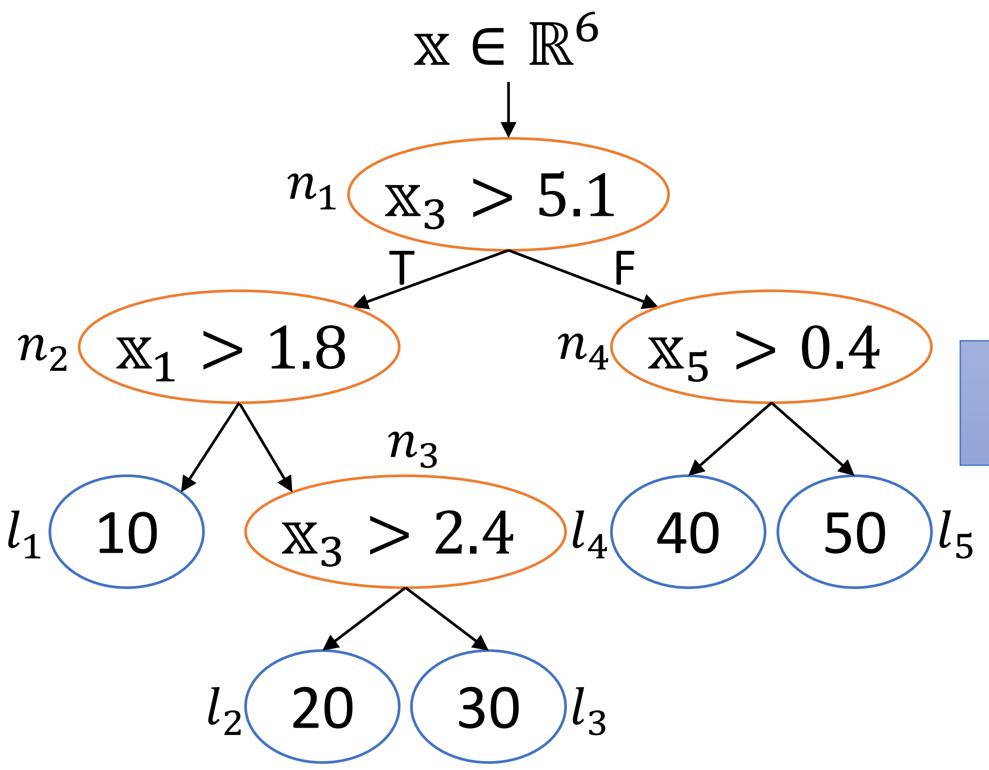
Translating Trees

Hummingbird
[OSDI'20]

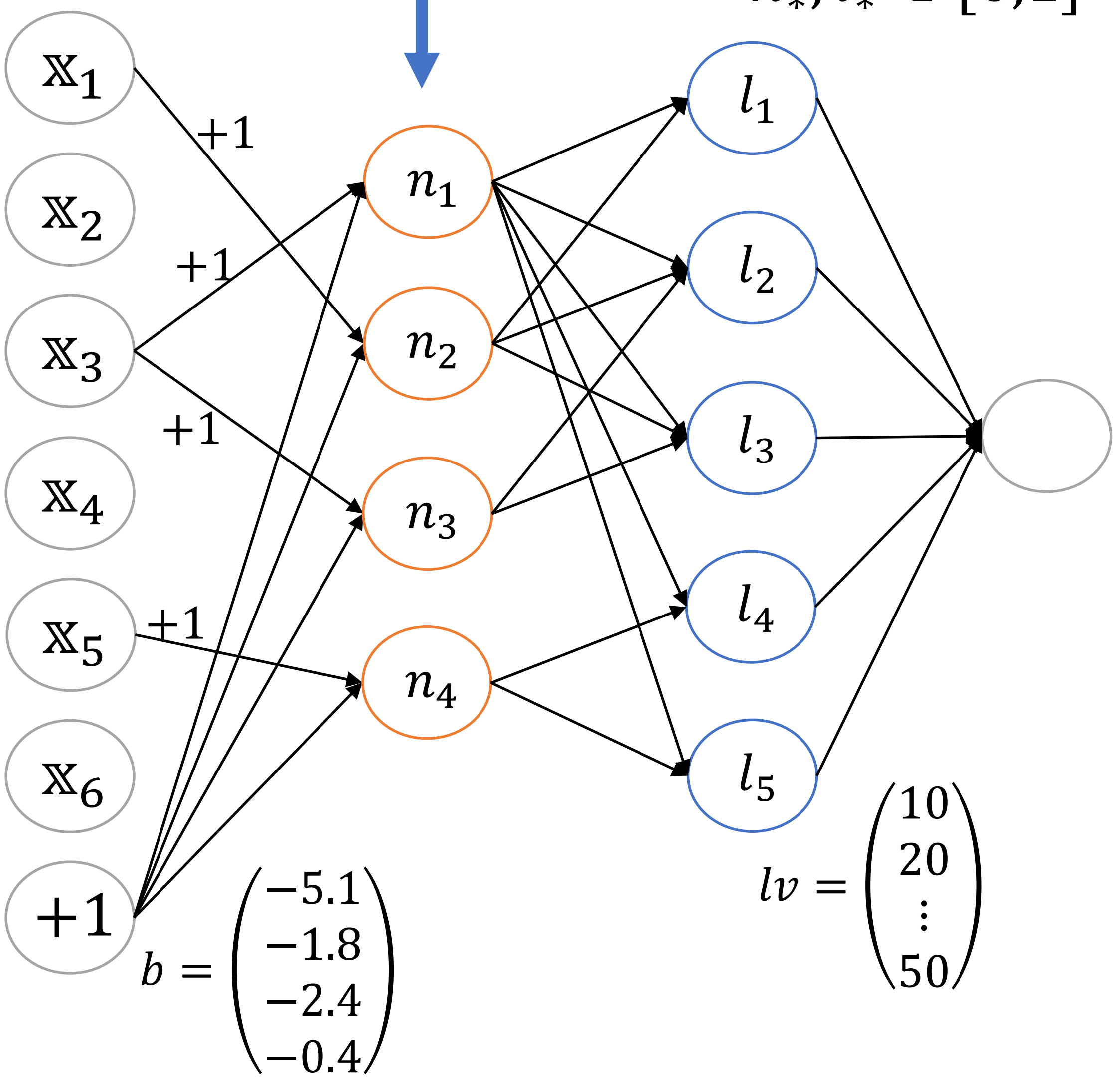


Translating Trees

Hummingbird
[OSDI'20]

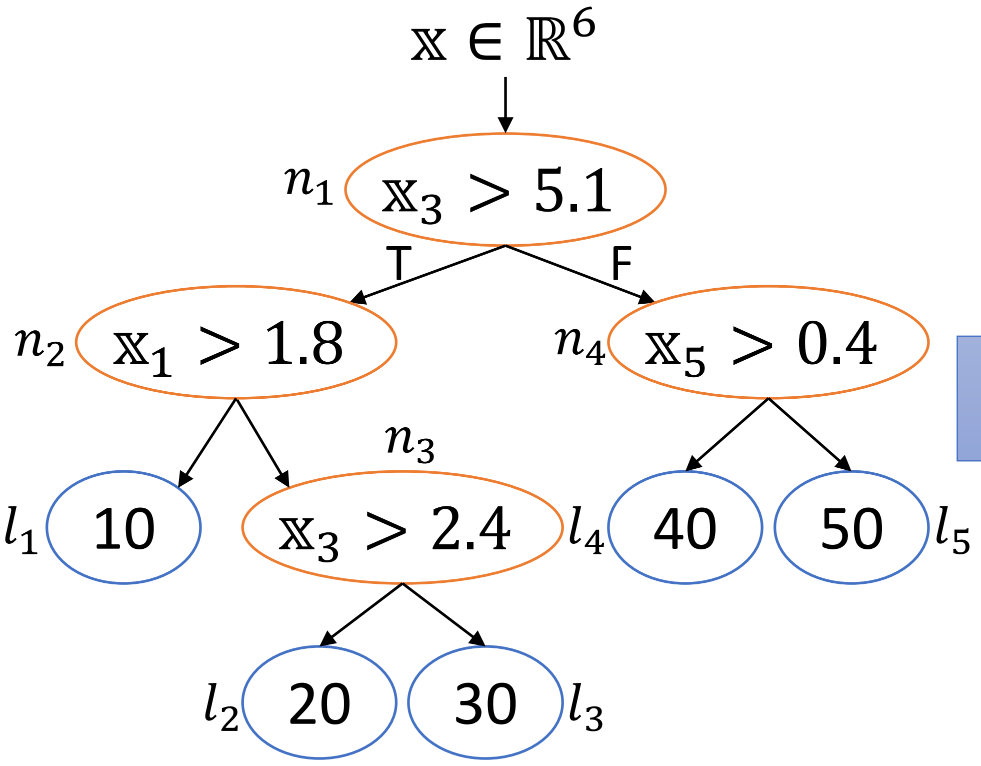


Evaluate all conditions together

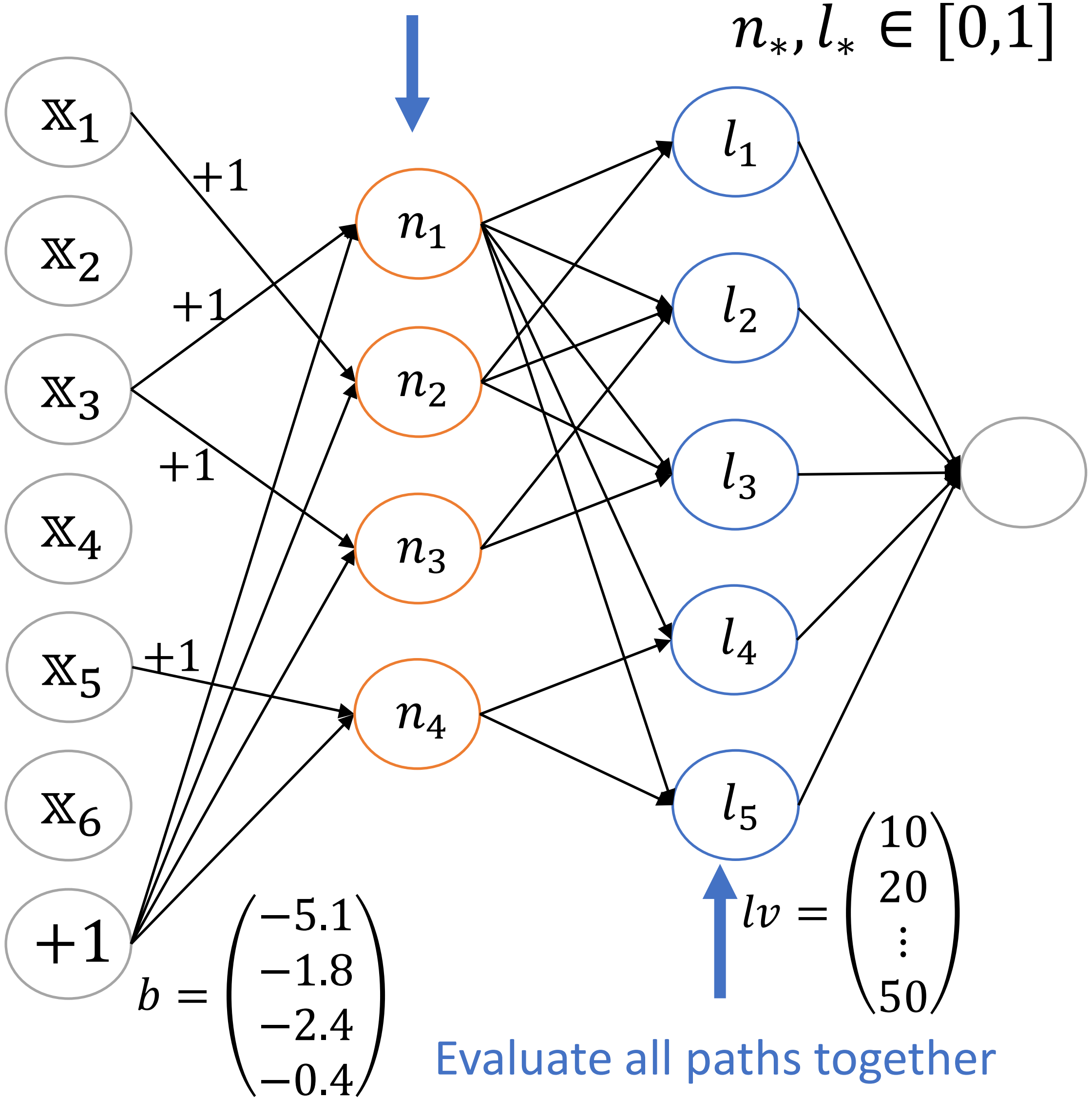


Hummingbird
[OSDI'20]

Translating Trees



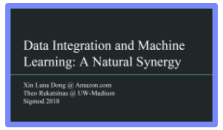
Evaluate all conditions together



Evaluate all paths together

DS Lifecycle: Data Cleaning & Deduplication

[Xin Luna Dong, Theodoros Rekatsinas: Data Integration and Machine Learning: A Natural Synergy. **SIGMOD 2018**]

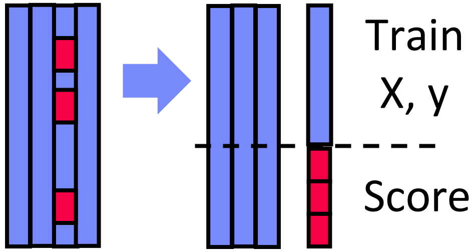


- **Key Observation**
  - State-of-the-art data cleaning based on ML (ML for DI, DI for ML)
  - **Examples:** Data extraction, schema alignment, entity resolution, data validation, data cleaning, outlier detection, missing value imputation, semantic type detection, data augmentation, feature selection, feature engineering, feature transformations
- **Outliers**
  - Winsorizing
  - Outlier by standard dev / quantiles

```
# compute quantiles for lower and upper
ql = quantile(X, 0.05);
qu = quantile(X, 0.95);
# replace values outside [ql,qu] w/ ql and qu
Y = min(qu, max(ql, X));
```

- **Missing Value Imputation**
  - Imputation by mean / mode [for MCAR]
  - Imputation by mice() [for MAR]

[Jose Cambronero, John K. Feser, Micah Smith, Samuel Madden: Query Optimization for Dynamic Imputation. **PVLDB 2017**]



Categorical attr. → **classification**
Numeric attr. → **regression**

DS Lifecycle: Data Cleaning & Deduplication, cont.

- **Entity Resolution Blocking**
  - Compute word/attribute embeddings + tuple embeddings
  - Locality-Sensitive Hashing (LSH) for blocking
  - K hash functions h(t) → k-dim hash-code
  - L hash tables,
    each k hash functions

[Muhammad Ebraheem et al: Distributed Representations of Tuples for Entity Resolution. **PVLDB 2018**]

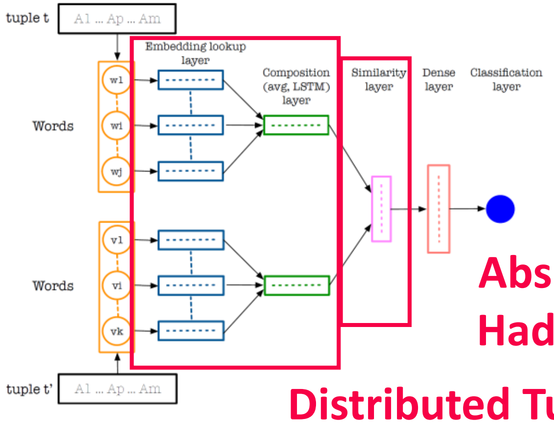
[Saravanan Thirumuruganathan et al. Deep Learning for Blocking in Entity Matching […]. **PVLDB 2021**]

V %*% H

```
h1=[-1, 1,1], h2=[ 1,1, 1],
h3=[-1,-1,1], h4=[-1,1,-1],
```

```
v[t1]=[0.45,0.8,0.85]  [1.2,2.1,-0.4,-0.5] → [1,1,-1,-1] → [12]  Hash
v[t2]=[0.4,0.85,0.75]  [1.2,2.0,-0.5,-0.3]   [1,1,-1,-1]   [12]  bucket
```

- **Deep Learning for ER**
  - Automatic representation learning from text (avoid feature engineering)
  - Leverage pre-trained word embeddings for semantics (no syntactic limitations)
  - **Examples:** DeepER, Magellan

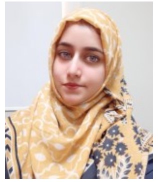


Abs Difference / Hadamard Prod.

Distributed Tuple Repr.

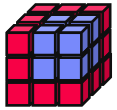
[SAGA @ SIGMOD’24a]

DS Lifecycle: Data Cleaning Pipelines



- **Automatic Generation of Cleaning Pipelines**
  - Library of robust, parameterized **data cleaning primitives**
  - **Enumeration of DAGs** of primitives & **hyper-parameter optimization** (GA, HB)

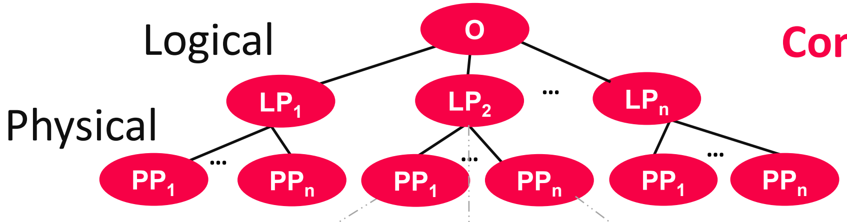
Data Samples



Target App

Rules/Objectives

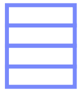
Data- and Task-parallel Computation



Outlier Detection → MVI → Deduplication → Resolve Mislabels

$P_{1:}$ gmm → imputeFD → mergeDup → delML

$P_{n:}$ outlierBySd → mice → delDup → voting



Top-k Pipelines

Debugging

Other Systems:
BoostClean,
HoloClean,
Raha-Baran,
Learn2Clean,
DiffPrep

University	Country
TU Graz	Austria
TU Graz	Austria
TU Graz	Germany
IIT	India
IIT	IIT
IIT	Pakistan
IIT	India
SIBA	Pakistan
SIBA	null
SIBA	null

Dirty Data

University	Country
TU Graz	Austria
TU Graz	Austria
TU Graz	Austria
IIT	India
IIT	India
IIT	India
IIT	India
SIBA	Pakistan
SIBA	Pakistan
SIBA	Pakistan

After **imputeFD(0.5)**

A	B	C	D
0.77	0.80	1	1
0.96	0.12	1	1
0.66	0.09	null	1
0.23	0.04	17	1
0.91	0.02	17	null
0.21	0.38	17	1
0.31	null	17	1
0.75	0.21	20	1
null	null	20	1
0.19	0.61	20	1
0.64	0.31	20	1

Dirty Data

A	B	C	D
0.77	0.80	1	1
0.96	0.12	1	1
0.66	0.09	17	1
0.23	0.04	17	1
0.91	0.02	17	1
0.21	0.38	17	1
0.31	0.29	17	1
0.75	0.21	20	1
0.41	0.24	20	1
0.19	0.61	20	1
0.64	0.31	20	1

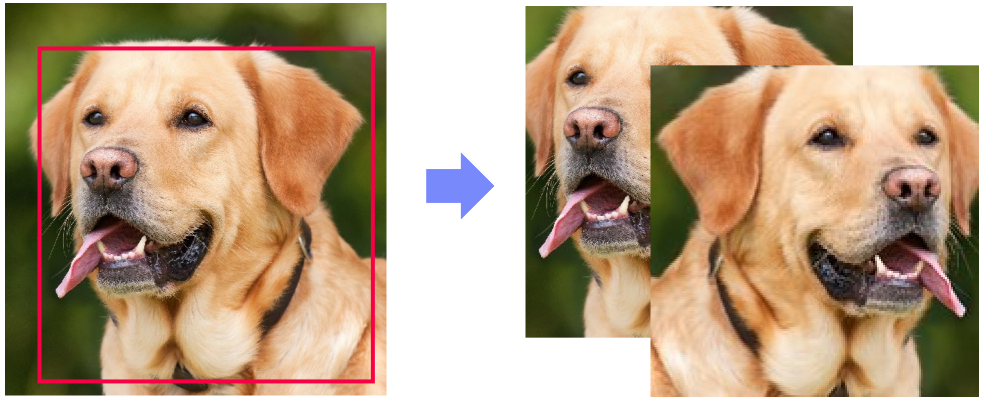
After **MICE**

DS Lifecycle: Data Augmentation

- **Data Augmentation Overview**
  - Complex ML models / deep NNs need lots of labeled data to avoid overfitting ➔ **expensive**
  - Augment training data by **synthetic labeled data**
  - **#1: Movement/selection** (translation, rotation, reflection, cropping)
  - **#2: Distortions** (stretching, shearing, lens distortions, color, mixup of images)
  - ➔ **Clean mapping to linear algebra operations**

AlexNet

[Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton: ImageNet Classification with Deep Convolutional Neural Networks. **NIPS 2012**]



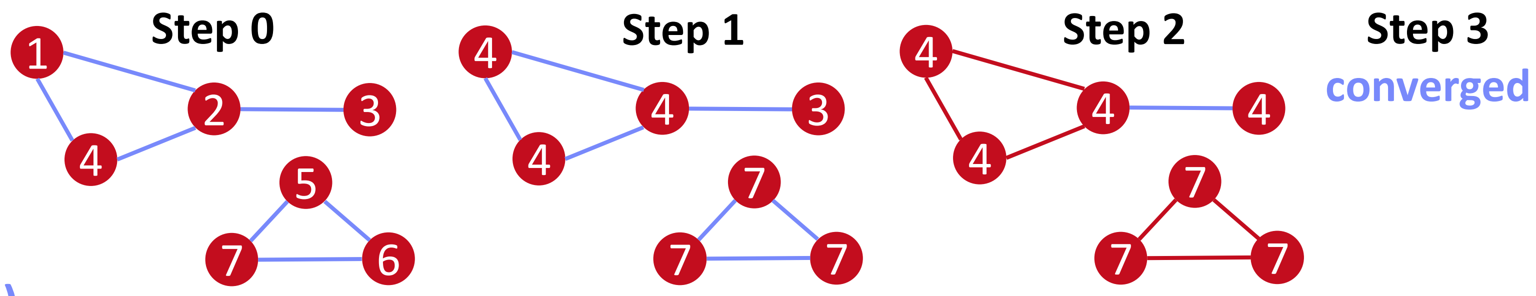
- **Effects of Data Augmentation**
  - **#1 Regularization** for reduced generalization error, not always training error
  - **#2 Invariance** increase by averaging features of augmented data points
  - ➔ **Data Augmentation as a Kernel**
    - Kernel metric for augmentation selection
    - Affine transforms on approx. kernel features

[Tri Dao, Albert Gu, Alexander Ratner, Virginia Smith, Chris De Sa, Christopher Ré: A Kernel Theory of Modern Data Augmentation. **ICML 2019**]

[[SIGMOD'10,
SIGMOD'20 Test of Time]

DS Lifecycle: Graph Processing

- **Connected Components**
  - Compute connected components (subgraphs)
  - Vertex-centric processing
  - **Propagate max(current, msgs)** if != current to neighbors, terminate if no msgs



- **Connected Components in Linear Algebra**

```
# initialize state with vertex ids
c = seq(1,nrow(G));
diff = Inf; iter = 1;
# iterative computation of connected components
while( diff > 0 & (maxi==0 | iter<=maxi) ) {
  u = max(rowMaxs(G * t(c)), c);
  diff = sum(u != c)
  c = u; # update assignment
  iter = iter + 1;
}
```

- **Other Examples**
  - Page Rank, Shortest Paths

DS Lifecycle: Training Decision Trees

- **Input:** X (recoded/binned)
- Main Algorithm

```
X2 = encode(X);
while( length(queue) > 0 ) {
  node0 = remove(queue, 1);
  [...] = findBestSplit(X2, ...)
  if( validSplit )
    M[, 2*nID-1:2*nID] <- (f,v);
  else
    M[,2*nID] = labelLeaf(…);
  putInQueueCond(ILeft);
  putInQueueCond(IRight); }
```

- **Output:** linearized tree

```
(L1)            |d<5|
               /     \
(L2)        P1:2    |a<7|
                    /   \
(L3)             P2:2  P3:1

--> M :=
[[4, 5, 0, 2, 1, 7, 0, 0, 0, 0, 0, 2, 0, 1]]
  |(L1)| |  (L2)   | |       (L3)         |
```

```
# evaluate features and feature splits
# (both categorical and numerical are treated similarly by
# finding a cutoff point in the recoded/binned representation)
R = matrix(0, rows=3, cols=nrow(feat));
parfor( i in 1:nrow(feat) ) {
  f = as.scalar(feat[i]);            # feature
  beg = as.scalar(foffb[1,f])+1; # feature start in X2
  end = as.scalar(foffe[1,f]);   # feature end in X2
  belen = end-beg; #numFeat - 1
  while(FALSE){} # make beg/end known

  # construct 0/1 predicate vectors with <= semantics
  # find rows that match at least one value and appear in I
  # (vectorized evaluation, each column in P is a split candidate)
  fP = upper.tri(target=matrix(1,belen,belen), diag=TRUE);

  vI = seq(1,belen);
  if( max_values < 1.0 & ncol(fP)>10 ) {
    rI2 = rand(rows=ncol(fP),cols=1,seed=seed) <= (ncol(fP)^max_values/ncol(fP));
    fP = removeEmpty(target=fP, margin="cols", select=t(rI2));
    vI = removeEmpty(target=vI, margin="rows", select=rI2);
  }
  P = matrix(0, ncol(X2), ncol(fP));
  P[beg:end-1,1:ncol(fP)] = fP;
  Ileft = (t(X2 %*% P) * I) != 0;
  Iright = (Ileft==0) * I;

  # compute information gain for all split candidates
  ig = as.scalar(computeImpurity(y2, I, impurity))
       - rowSums(Ileft)/numI * computeImpurity(y2, Ileft, impurity)
       - rowSums(Iright)/numI * computeImpurity(y2, Iright, impurity);
  ig = replace(target=ig, pattern=NaN, replacement=0);
```

Task-parallel over features

Optional value sampling O(log m)

Vectorized <= split evaluation (matmult)

Vectorized information gain computation

Feature Matrix X2

1	0	0	0	1	0	0
0	0	1	0	0	1	0
0	0	1	0	1	0	0
0	1	0	1	0	0	0
1	0	0	1	0	0	0
0	1	0	0	0	1	0
0	1	0	0	0	0	1
1	0	0	1	0	0	0
0	1	0	0	1	0	0
1	0	0	0	0	0	1
0	0	1	0	0	1	0
0	0	1	0	0	0	1

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	1	1	1
0	0	0	0	1	1	1
0	0	0	0	0	1	1
0	0	0	0	0	0	1

0	1	1	1
0	0	1	1
0	1	1	1
1	1	1	1
1	1	1	1
0	0	1	1
0	0	0	1
1	1	1	1
0	1	1	1
0	0	0	1
0	0	1	1
0	0	0	1

[SliceLine @ SIGMOD'21c]

sliceline

[**Credit:** sliceline, Silicon Valley, HBO]

DS Lifecycle: Model Debugging

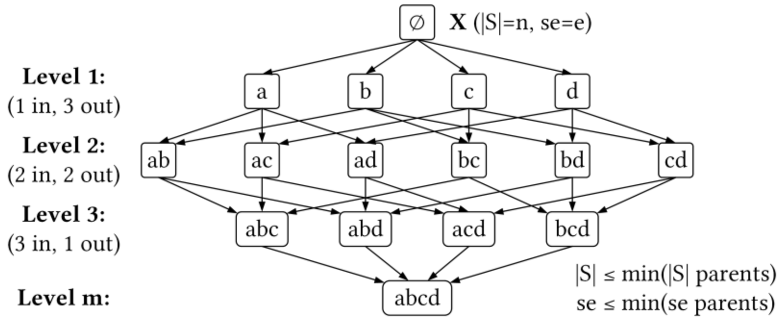
- **Problem Formulation**
  - **Intuitive slice scoring function**
  - Exact top-k slice finding
  - $|S| \geq \sigma \wedge sc(S) > 0$
  - $\alpha \in (0,1]$

$$sc = \alpha\left(\frac{\bar{e}(S)}{\bar{e}(X)} - 1\right) - (1-\alpha)\left(\frac{|X|}{|S|} - 1\right)$$

$$= \alpha\left(\frac{|X|}{|S|} \cdot \frac{\sum_{i=1}^{|S|} es_i}{\sum_{i=1}^{|X|} e_i} - 1\right) - (1-\alpha)\left(\frac{|X|}{|S|} - 1\right)$$

slice error **slice size**

- **Properties & Pruning**
  - Monotonicity of slice sizes, errors
  - **Upper bound sizes/errors/scores**
    → pruning & termination



- **Linear-Algebra-based Slice Finding**
  - Recoded matrix **X**, error vector e
  - **Vectorized implementation in linear algebra**
    (join & eval via sparse-sparse matrix multiply per lattice level)
  - Local and distributed task/data-parallel execution

Candidate Slices

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

Data

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

== Level

Fairness and Explainability

- **Fairness Problem Formulation**
  - A **fairness specification** given by a triplet (g, f, ε) induces (|g(D)|choose 2) **fairness constraints** on pairs of groups
  - A fairness specification is satisfied by a classifier h on D iff all induced fairness constraints are satisfied, i.e., ∀gi,gj ∈ g(D), |f(h,gi)−f(h,gj)| ≤ ε
  - **Unconstrained optimization problem**

[H. Zhang et al: OmniFair: A Declarative System for Model-Agnostic Group Fairness in Machine Learning, **SIGMOD 2021**]

max accuracy
s.t. fairness

max accuracy
+ fairness

- **Explainability via LIME**
  - **Sample perturbations** of prediction input (e.g., hide parts of image, attribute values)
  - **Locally weighted regression**

Loss Function

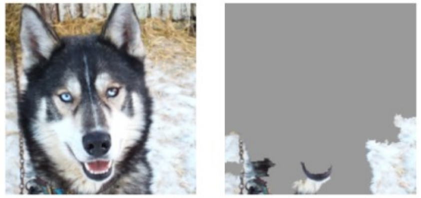
Regularizer

$$\xi(x) = \underset{g\in G}{\operatorname{argmin}} \; \mathcal{L}(f,g,\pi_x) + \Omega(g)$$

Linear Models

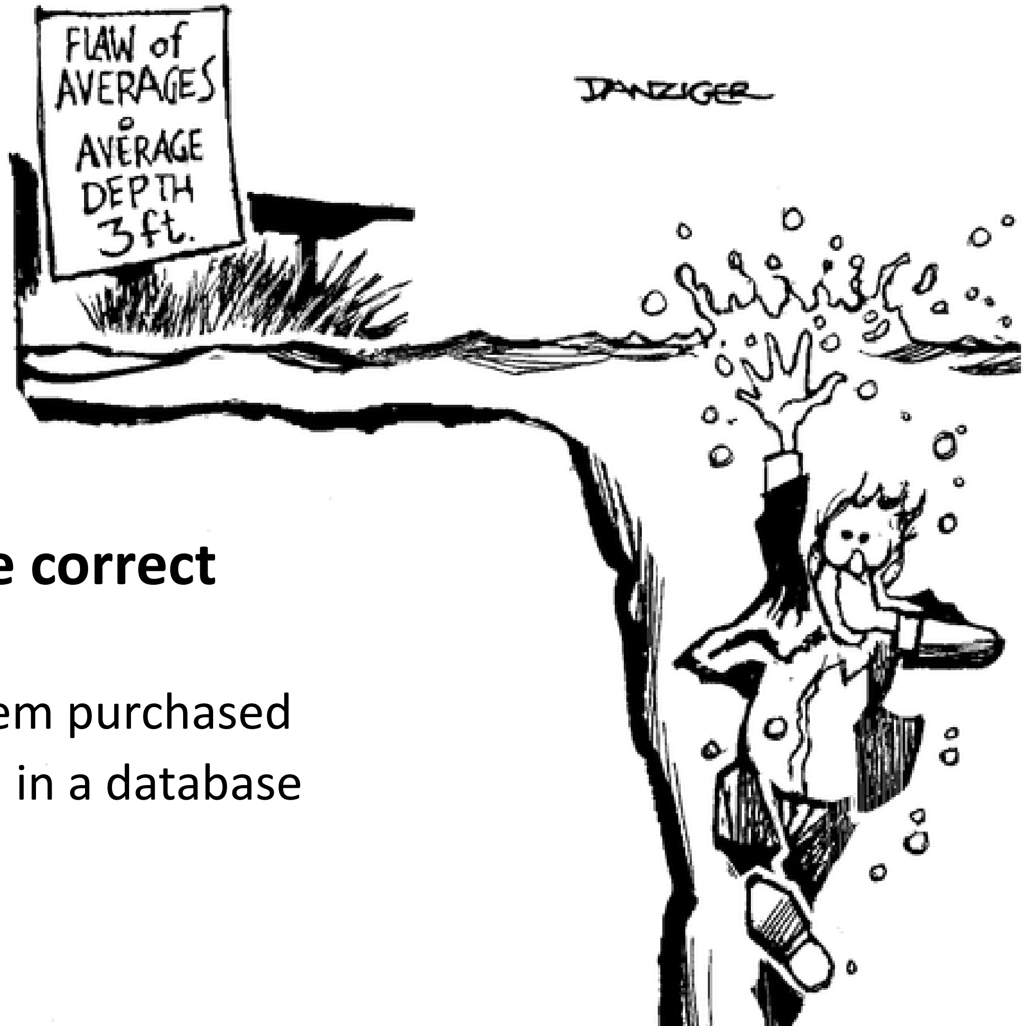
Local Kernel

[Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin: Why Should I Trust You?: Explaining the Predictions of Any Classifier, **KDD 2016**]



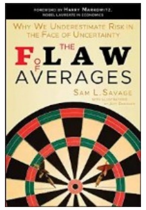
(a) Husky classified as wolf (b) Explanation

Sampling and Simulations



- **Classical data analysis assumes database data are correct**
  - Ex: I have a customer *c* who places an order $\mathbf{x}_c$
  - $\mathbf{x}_c[i]$ represents the dollars spent on the *i*th inventory item purchased
  - We would typically assume that the $\mathbf{x}_c[i]$ value recorded in a database reflects reality
  - But does it always? **No!**
  - Can we afford to ignore the possibility $\mathbf{x}_c[i]$ is incorrect?

Even if correct on average, can't afford to ignore the implications of the extremes…



Flaw of Averages
[Savage09]

To Quantify Risk: Use a Statistical Model

Mean vector: gives average purchase $$ for this customer

$$\mu = \begin{bmatrix} 15.6 \\ 5.1 \\ 3.2 \\ 6.9 \end{bmatrix}$$

To generate a purchase vector:

1) Sample $\mathbf{x} \sim \text{Normal}(\mu, \Sigma)$
2) For $i \in \{1 \ldots 4\}$
3) With probability 0.2
4) $\mathbf{x}[i] \leftarrow 0$

matrix: gives correlations in purchase $$ for this customer

Strong correlation between Item 1 and item 4

$$\Sigma = \left(\begin{bmatrix} 10.2 \\ 2.5 \\ 3.8 \\ 7.1 \end{bmatrix} \begin{bmatrix} 10.2 & 2.5 & 3.8 & 7.1 \end{bmatrix} \right) \circ \begin{bmatrix} 1 & 0.1 & -0.1 & 0.95 \\ 0.1 & 1 & 0.2 & 0.1 \\ -0.1 & 0.2 & 1 & -0.1 \\ 0.95 & 0.1 & -0.1 & 1 \end{bmatrix}$$

Vector of std devs

Correlation matrix

$$\bar{\mathbf{x}} = \begin{bmatrix} 22.8 \\ 1.6 \\ 4.5 \\ 0 \end{bmatrix}$$

Simulates the case where the 8.2 was never recorded

Given Such a Model

$$\bar{\mathbf{x}} = \begin{bmatrix} 22.8 \\ 1.6 \\ 4.5 \\ 0 \end{bmatrix}$$

- **We can look at an observed purchase vector and "correct" it**
  - The first entry is high (22.8 vs mean of 15.6)
  - But the last entry is low (0 vs mean of 6.9)
  - How is this possible with a correlation of 0.95?
  - According to our model: most likely the 0 is wrong
  - But we don't know the correct vector
  - Go Bayesian! Use Bayes' rule to sample from $p(\mathbf{x}|\bar{\mathbf{x}}, \mu, \Sigma)$

Typically using MCMC

$$\mathbf{x} = \begin{bmatrix} 22.8 \\ 1.6 \\ 4.5 \\ 8.6 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} 22.8 \\ 1.6 \\ 4.5 \\ 6.9 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} 22.8 \\ 1.6 \\ 4.5 \\ 7.5 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} 0 \\ 1.6 \\ 4.5 \\ 0 \end{bmatrix}$$

How to Facilitate in a Database?

- **Need the ability to move from relations to tensors**
  - Math uses tensors (vectors, matrices, etc.), not relations!
- **And the ability to sample random values from distributions**

Assume sales data are stored relationally:

```
sale (cust_id, sale_date, prod_id, amt)
```

First step, transform into vectors:

Agg function to create a vector from (`int`, `numerical`) pairs, the int is the dim

Creates a tuple of type (`int`, `numerical`)

```
CREATE VIEW vecs AS
SELECT VECTORIZE (label_scalar (prod_id, amt))
   AS sale_vec, cust_id, sale_date
FROM sale
GROUP BY cust_id, sale_date
```

Linear Algebra on an RDBMS [ICDE17]

Now We Have a Set of Vectors

- **Compute observed mean and covariance on a per-customer basis**
  - Will serve as an empirical prior for each customer
  - So-called “empirical Bayes”

We now have a view containing sales vectors

```
vecs (sale_vec, cust_id, sale_date)
```

And now one that has the empirical covariance matrix:

It’s easy to create a table having the mean vector for each customer:

```
CREATE VIEW covars AS
SELECT outer_product (v.sale_vec - m.avg_sale,
   v.sale_vec - m.avg_sale) AS covar, v.cust_id
FROM vecs AS v, means AS m
WHERE v.cust_id = m.cust_id
GROUP BY v.cust_id
```

```
CREATE VIEW means AS
SELECT AVG (sale_vec) AS avg_sale, cust_id
FROM vecs
GROUP BY cust_id
```

We Are Ready To Create a Simulated Table

MCDB
[SIGMOD08]

- **Invoke a special UDF called a "VG function"**
  - In our case, `ResampleVec` encapsules an MCMC algorithm to "fix" sales vector

Here are the views we've created:

```
vecs (sale_vec, cust_id, sale_date)
means (avg_sale, cust_id)
covars (covar, cust_id)
```

It's easy to create a table having the mean vector for each cust:

```
CREATE TABLE simulated_sales AS
  FOR EACH v IN vecs
     WITH sale_vec AS ResampleVec (v.sale_vec,
        (SELECT m.avg_sale FROM means AS m WHERE m.cust_id = v.cust_id),
        (SELECT c.covar FROM covars AS c WHERE c.cust_id = v.cust_id))
     SELECT v.cust_id, v.sale_date, s.value
     FROM sale_vec s
```

For each sale, use VG function to sample new vec

ResampleVec is a VG function that simulates the sale

VG function accepts old vec

Subquery to get the mean

Subquery to get the covariance

Then stitch together output tuples

Now, queries over this simulated table return a **distribution** of results. Ex:

```
SELECT AVG (s.value)
FROM simulated_sales s, cust c
WHERE s.cust_id = c.cust_id AND
  c.region = 'northeast' AND
  s.sale_date = '12-25-22'
```

Stochastic Simulations

SimSQL [SIGMOD13]

- **Can use these tools to build database-valued Markov chains**
  - Faciltates very large scale Bayesian machine learning
  - Can illustrate with a toy example

This table has a single tuple:

`people (start_loc)`

Vector where `start_loc[i]` tells us how many people are currently located at city `i`

This table has info on cities:

`city (city_id, transition_probs)`

Vector where `transition_probs[i]` tells us probability of transitioning from `city_id` to city `i`

Goal: implement a simple Markov chain that has people moving from city to city accorging to the specified probabilities

Initialize the simulation:

```
CREATE TABLE locations[0] (cur_loc) AS
    SELECT * FROM people
```

Simulate the transitions out of each city:

```
CREATE TABLE transitions[i] (next_pos) AS
  FOR EACH c in city
    WITH next_pos AS Multinomial (
      (SELECT l.loc[c.city_id]
         FROM locations[i - 1] AS l),
       c. transition_probs)
    SELECT n.value
    FROM next_pos s
```

And aggregate over all of the source cities to find the number of people in each destination:

```
CREATE TABLE locations[i] AS
  SELECT SUM (next_pos)
  FROM transitions[i]
```

Note how use of vectors makes this quite simple!

References

[**SIGMOD'24**] Shafaq Siddiqi, Matthias Boehm: Saga: A Scalable Framework for Optimizing Data Cleaning Pipelines for Machine Learning Applications, SIGMOD 2024
[**SIGMOD'23**] Peng Li, Zhiyi Chen, Xu Chu, Kexin Rong: DiffPrep: Differentiable Data Preprocessing Pipeline Search for Learning over Tabular Data, SIGMOD 2023
[**PVLDB'22**] Dong He, Supun Chathuranga Nakandala, Dalitso Banda, Rathijit Sen, Karla Saur, Kwanghyun Park, Carlo Curino, Jesús Camacho-Rodríguez, Konstantinos Karanasos, Matteo Interlandi: Query Processing on Tensor Computation Runtimes. Proc. VLDB Endow. 15(11): 2811-2825 (2022)
[**PVLDB'21**] Saravanan Thirumuruganathan, Han Li, Nan Tang, Mourad Ouzzani, Yash Govind, Derek Paulsen, Glenn Fung, AnHai Doan: Deep Learning for Blocking in Entity Matching: A Design Space Exploration. Proc. VLDB Endow. 14(11): 2459-2472 (2021)
[**SIGMOD'21**] Hantian Zhang, Xu Chu, Abolfazl Asudeh, Shamkant B. Navathe: OmniFair: A Declarative System for Model-Agnostic Group Fairness in Machine Learning. SIGMOD 2021
[**SIGMOD’21**] Svetlana Sagadeeva, Matthias Boehm: SliceLine: Fast, Linear-Algebra-based Slice Finding for ML Model Debugging. SIGMOD Conference 2021: 2290-2299
[**OSDI'20**] Supun Nakandala, Karla Saur, Gyeong-In Yu, Konstantinos Karanasos, Carlo Curino, Markus Weimer, Matteo Interlandi: A Tensor Compiler for Unified Machine Learning Prediction Serving. OSDI 2020
[**ICML'19**] Tri Dao, Albert Gu, Alexander Ratner, Virginia Smith, Chris De Sa, Christopher Ré: A Kernel Theory of Modern Data Augmentation. ICML 2019
[**PVLDB'18**] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, Nan Tang: Distributed Representations of Tuples for Entity Resolution. PVLDB 2018
[**SIGMOD'18**] Xin Luna Dong, Theodoros Rekatsinas: Data Integration and Machine Learning: A Natural Synergy. SIGMOD 2018
[**PVLDB'17**] José Cambronero, John K. Feser, Micah J. Smith, Samuel Madden: Query Optimization for Dynamic Imputation. PVLDB 2017
[**KDD'16**] Marco Túlio Ribeiro, Sameer Singh, Carlos Guestrin: "Why Should I Trust You?": Explaining the Predictions of Any Classifier. KDD 2016
[**NeurIPS'12**] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton: ImageNet Classification with Deep Convolutional Neural Networks. NeurIPS 2012
[**SIGMOD'10**] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, Grzegorz Czajkowski: Pregel: a system for large-scale graph processing. SIGMOD 2010

Selected Runtime Backends

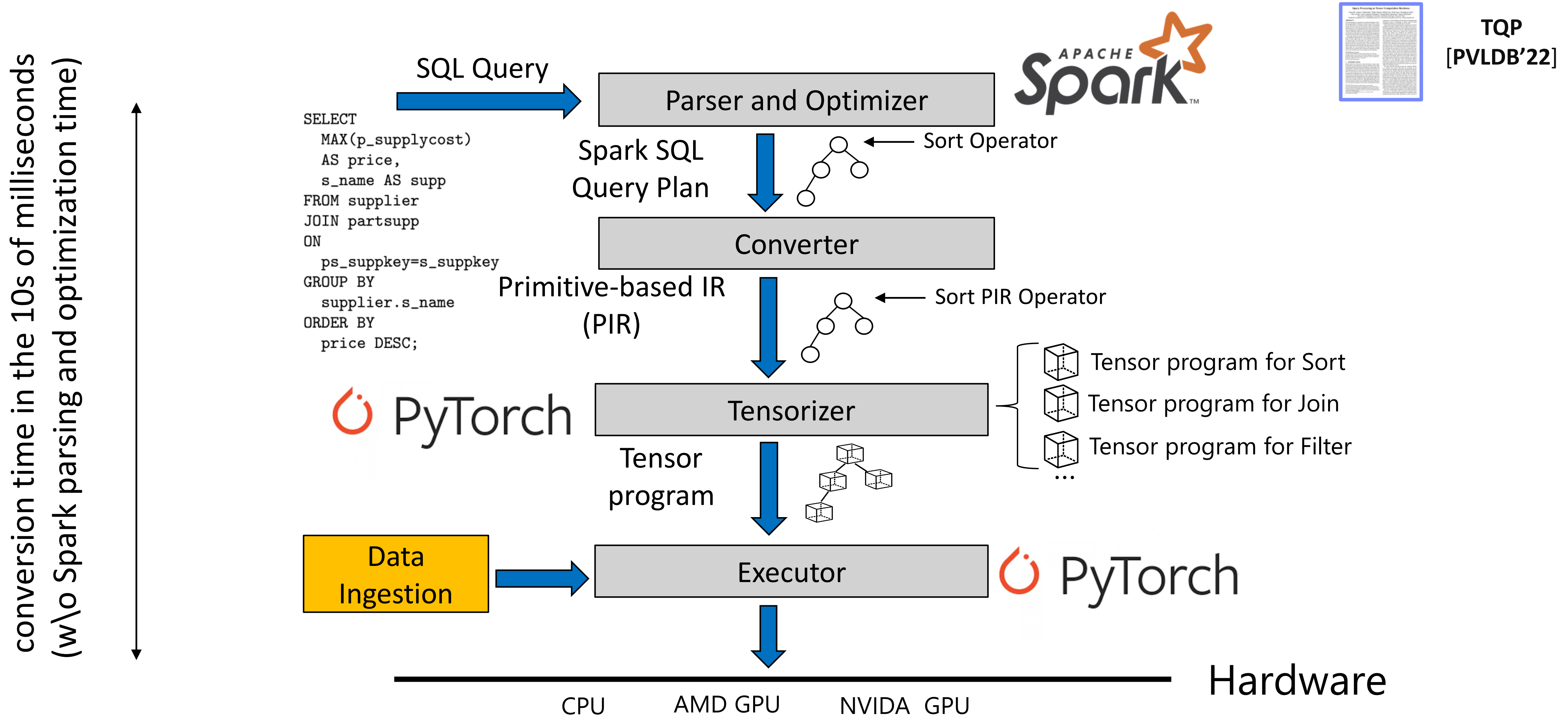
Tensor Query Processor (RDBMS→MLSys)

Apache SystemDS (MLSys)

Tensor Relational Algebra (MLSys → RDBMS)

[35 min]

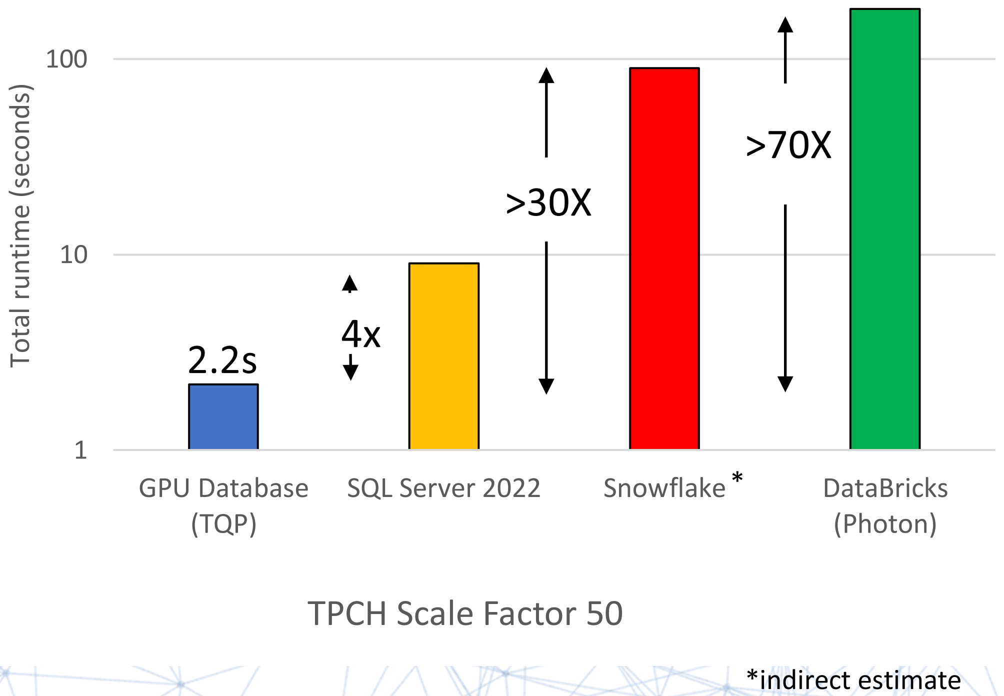
SQL to Tensor conversion: Tensor Query Processor (TQP)



Pros and Con of Tensor Query Processor

Pro:

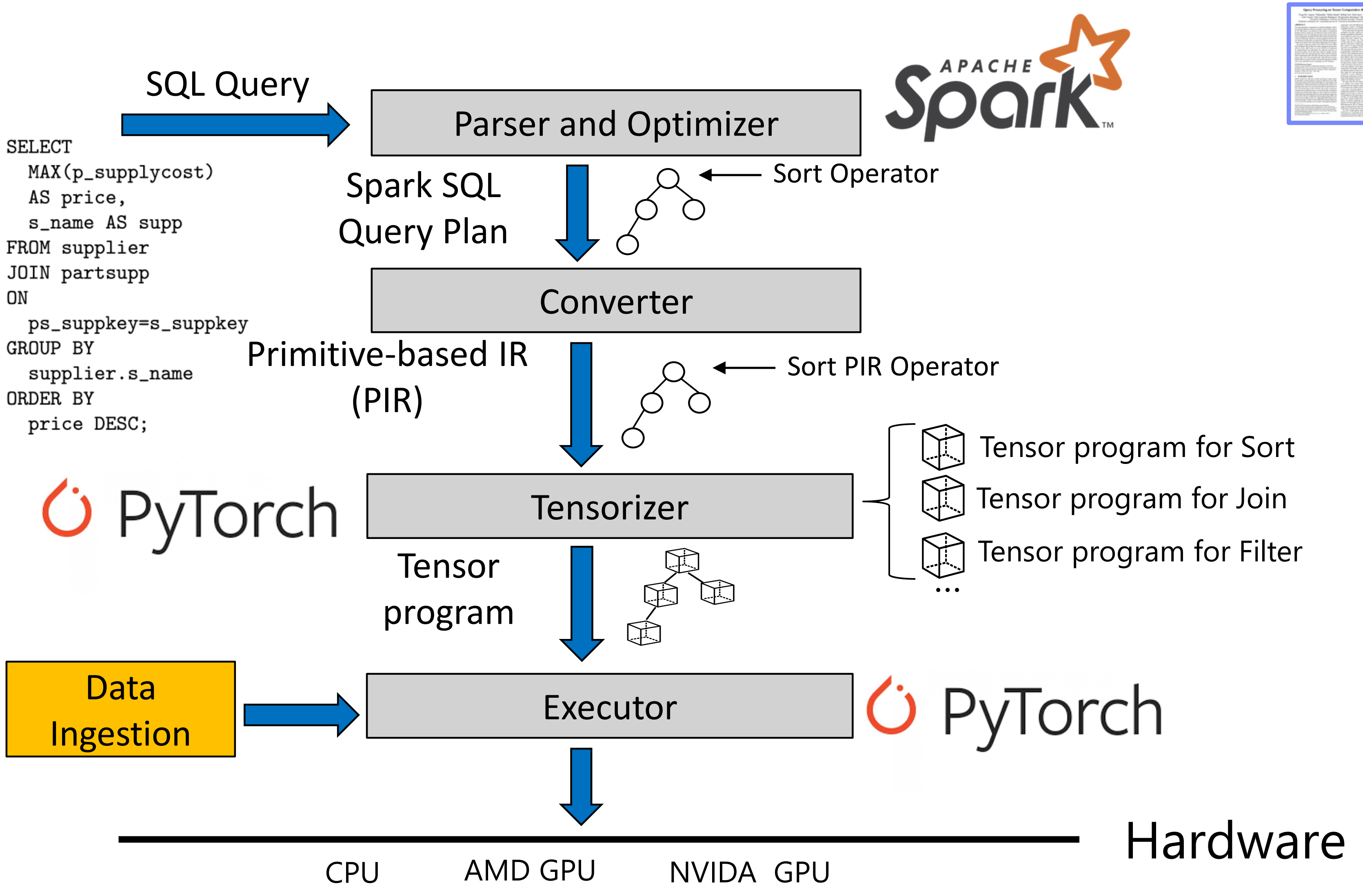
- Scalable Approach (no need to reimplement the query processor for each new hardware)
- Leverage the massive investments in special HW
- Low engineering effort: TQP + HB in 20k lines of Python code
- Great performance



Limitations:

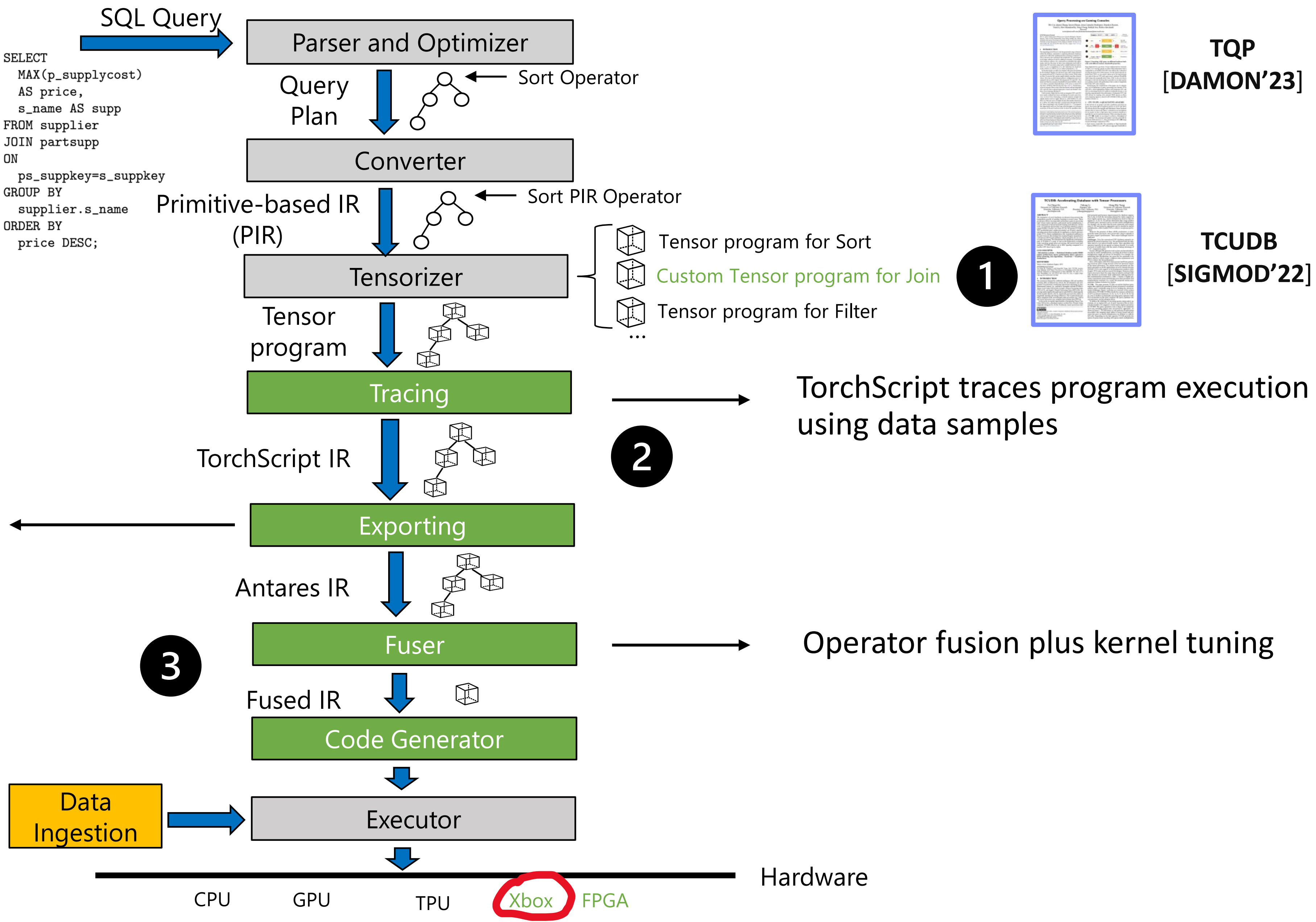
- Can only target devices supported by PyTorch (e.g., no Xbox)
- Sub-operators require many data materializations
- Cannot take advantage of advanced neural network compilers techniques (e.g., no operator fusion, no kernel tuning)
- Some operators are hard to implement efficiently

SQL to Tensor conversion: Tensor Query Processor (TQP)



TQP
[PVLDB'22]

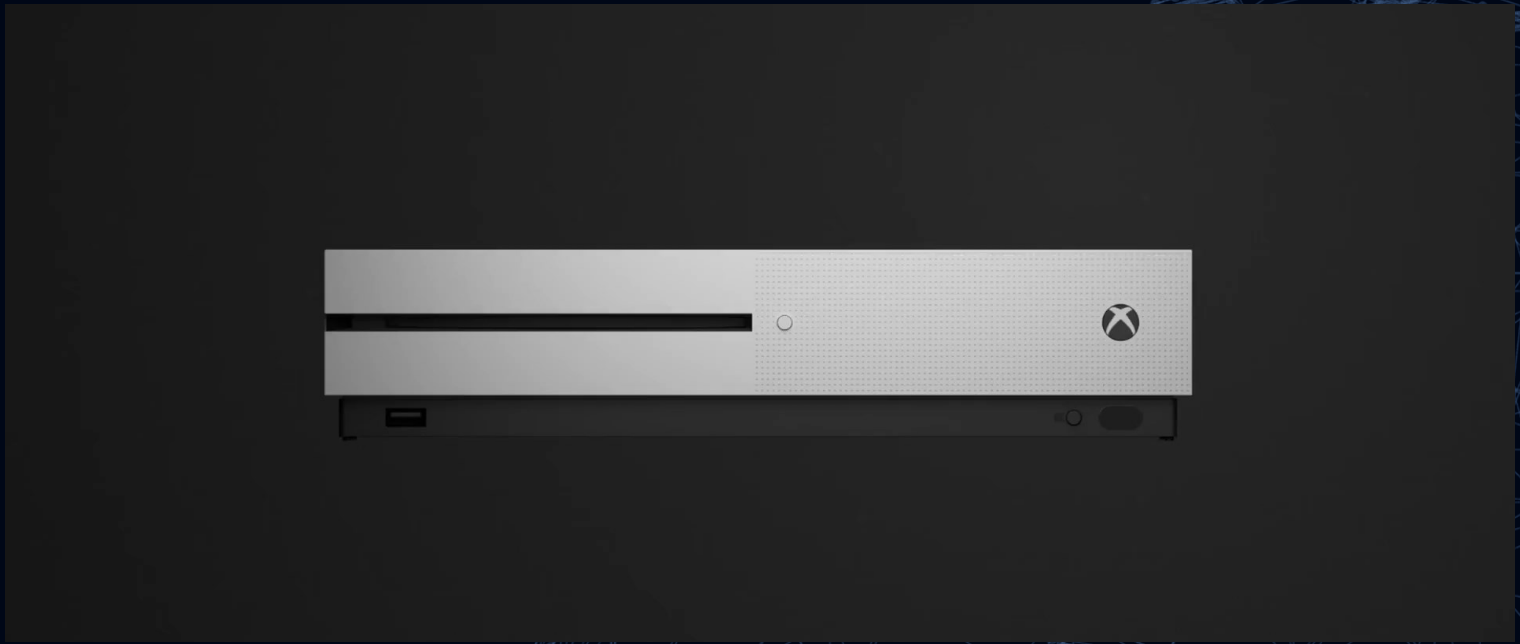
Extending TQP



[Antares](https://github.com/microsoft/antares) is a Cross-compiling Engine for Microsoft 1st/2nd party devices

https://github.com/microsoft/antares

xCloud

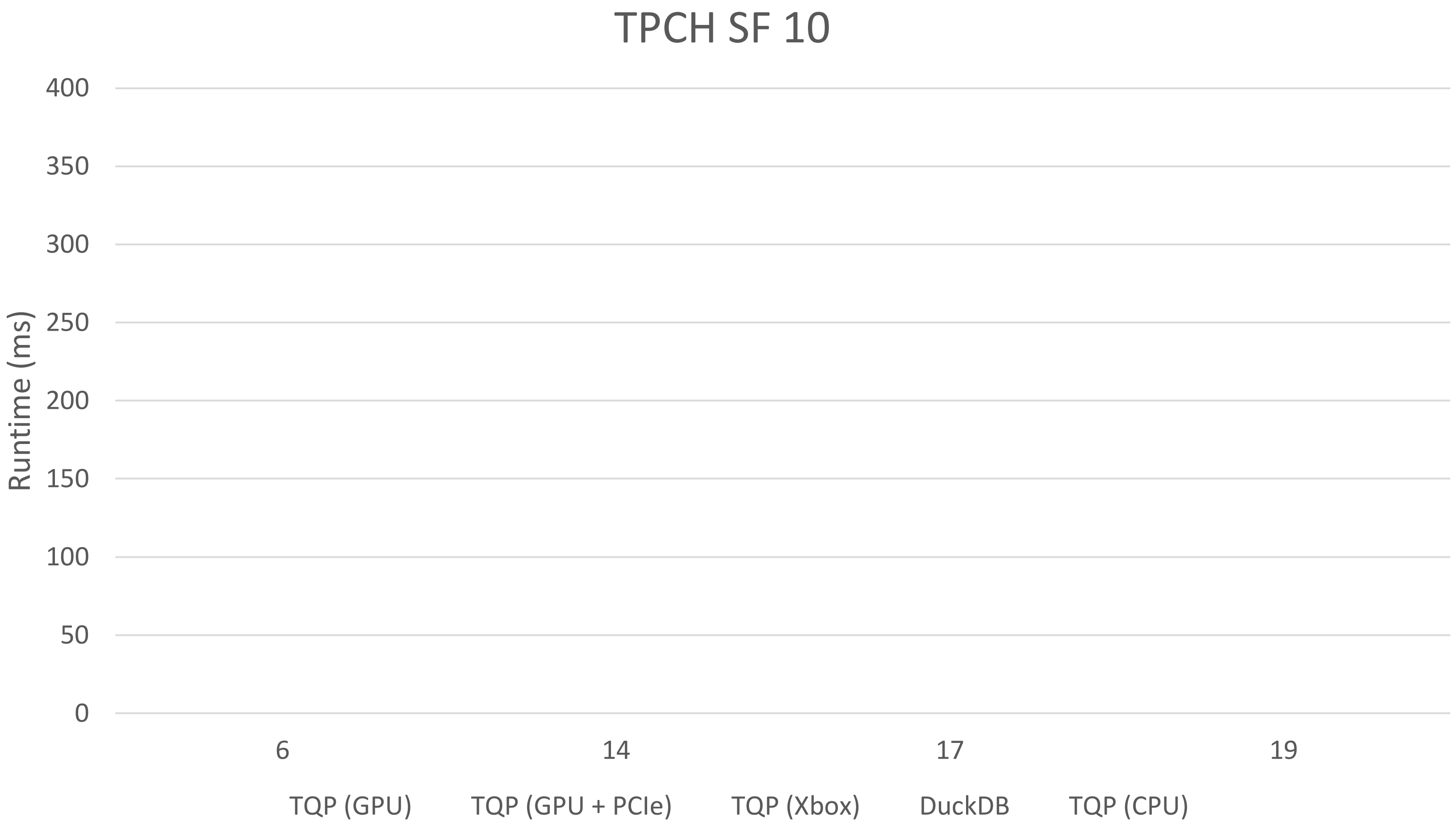


Interesting HW configuration: APU design where CPU and GPU share HBM (no PCI-e)

Predictable usage pattern: gamers mostly play during the evenings (AKA dark time)

TPCH on Xbox (SF 10, P100)

TQP
[DAMON'23]

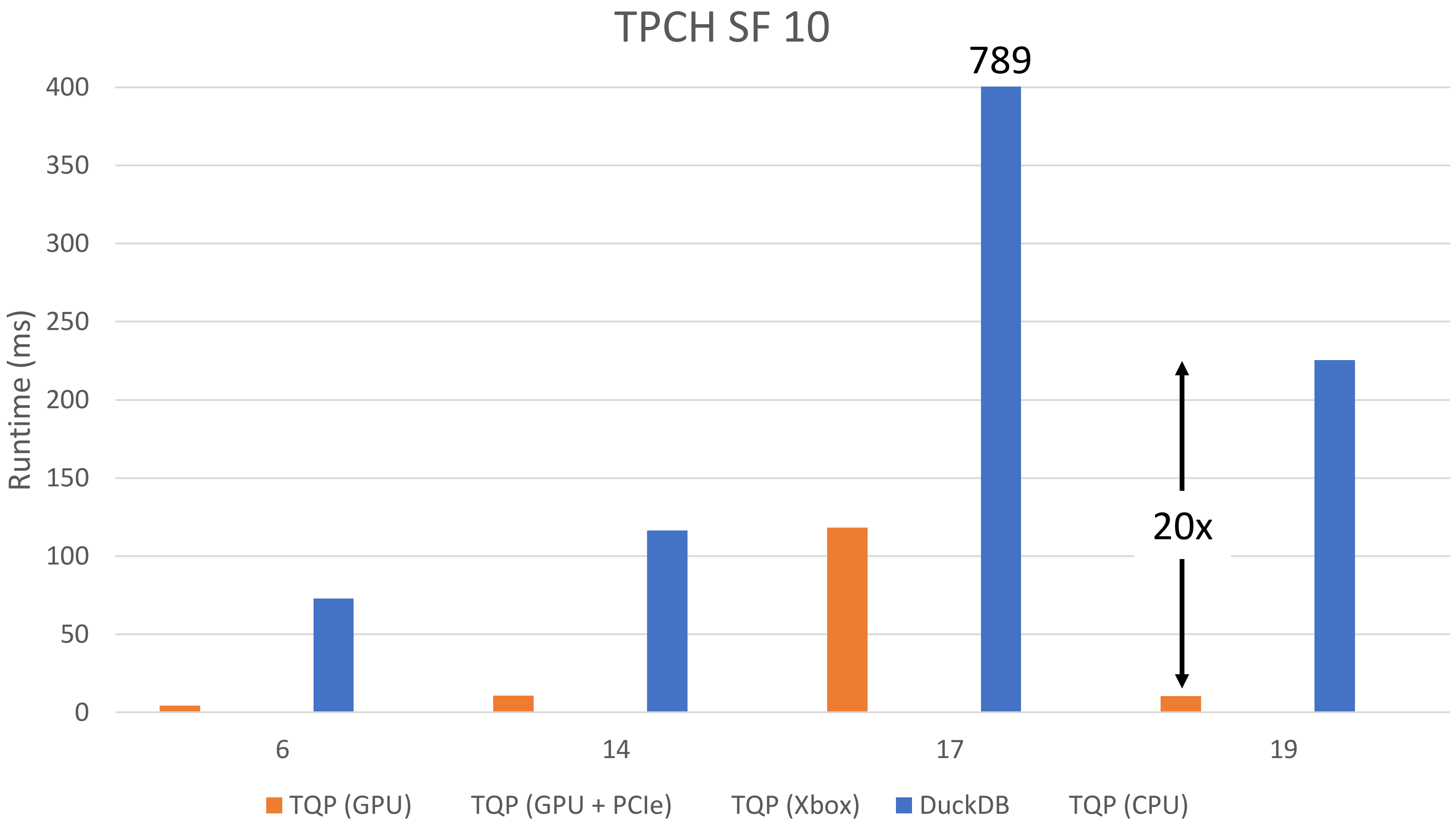


GPU: NVIDIA P100
Xbox: Series X
CPU: Xeon E5-2690 (14 cores)

	Xeon E5-2690	P100	Xbox Series X
Memory Bandwidth (GB/s)	154	732	560
Unidirectional PCIv3 (GB/s)	-	16	-
Theoretical TFLOPs	1.4	9.5	12.0

TPCH on Xbox (SF 10, P100)

TQP
[DAMON'23]

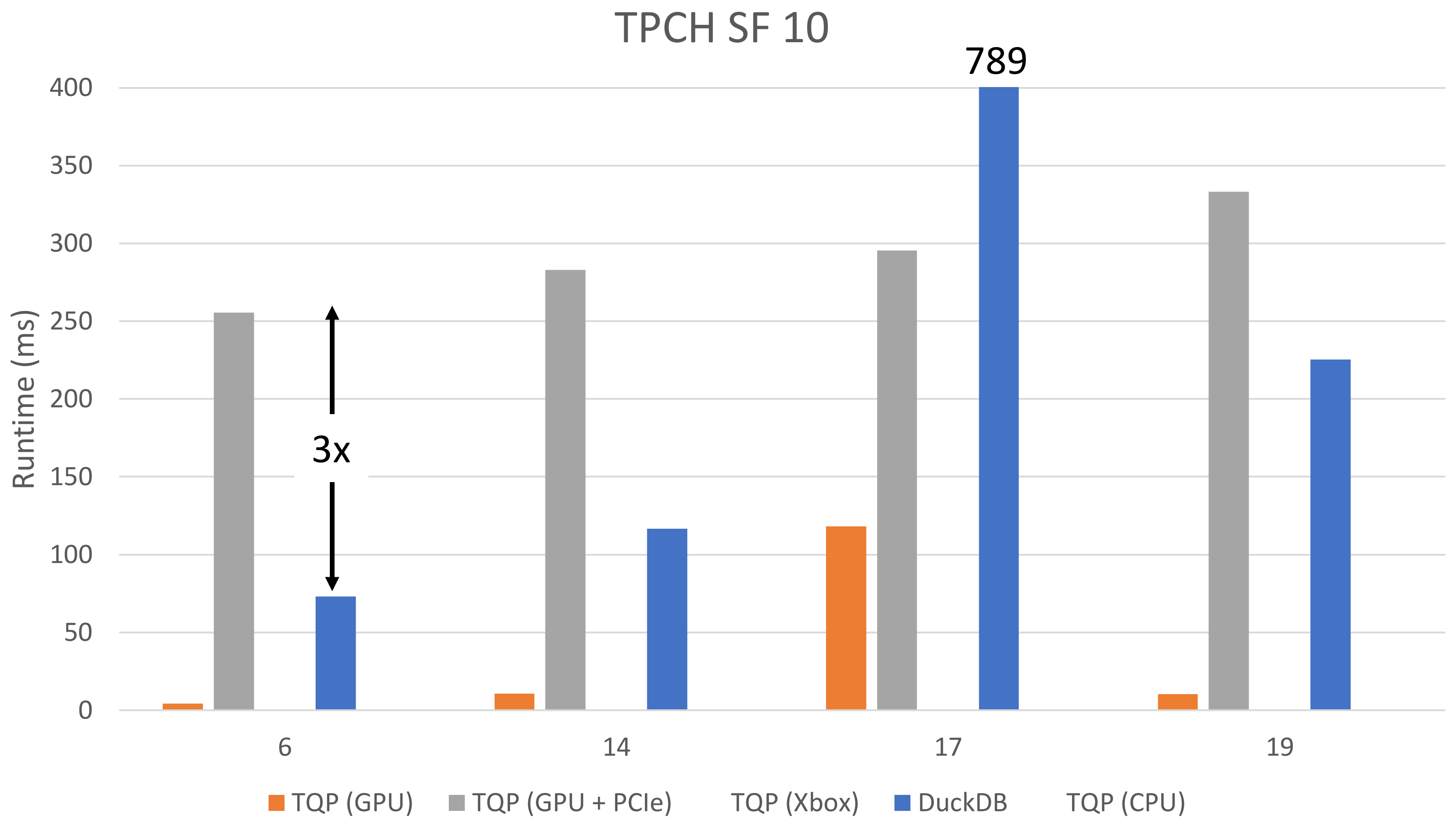


GPU: NVIDIA P100
Xbox: Series X
CPU: Xeon E5-2690 (14 cores)

	Xeon E5-2690	P100	Xbox Series X
Memory Bandwidth (GB/s)	154	732	560
Unidirectional PCIv3 (GB/s)	-	16	-
Theoretical TFLOps	1.4	9.5	12.0

TQP
[DAMON'23]

TPCH on Xbox (SF 10, P100)

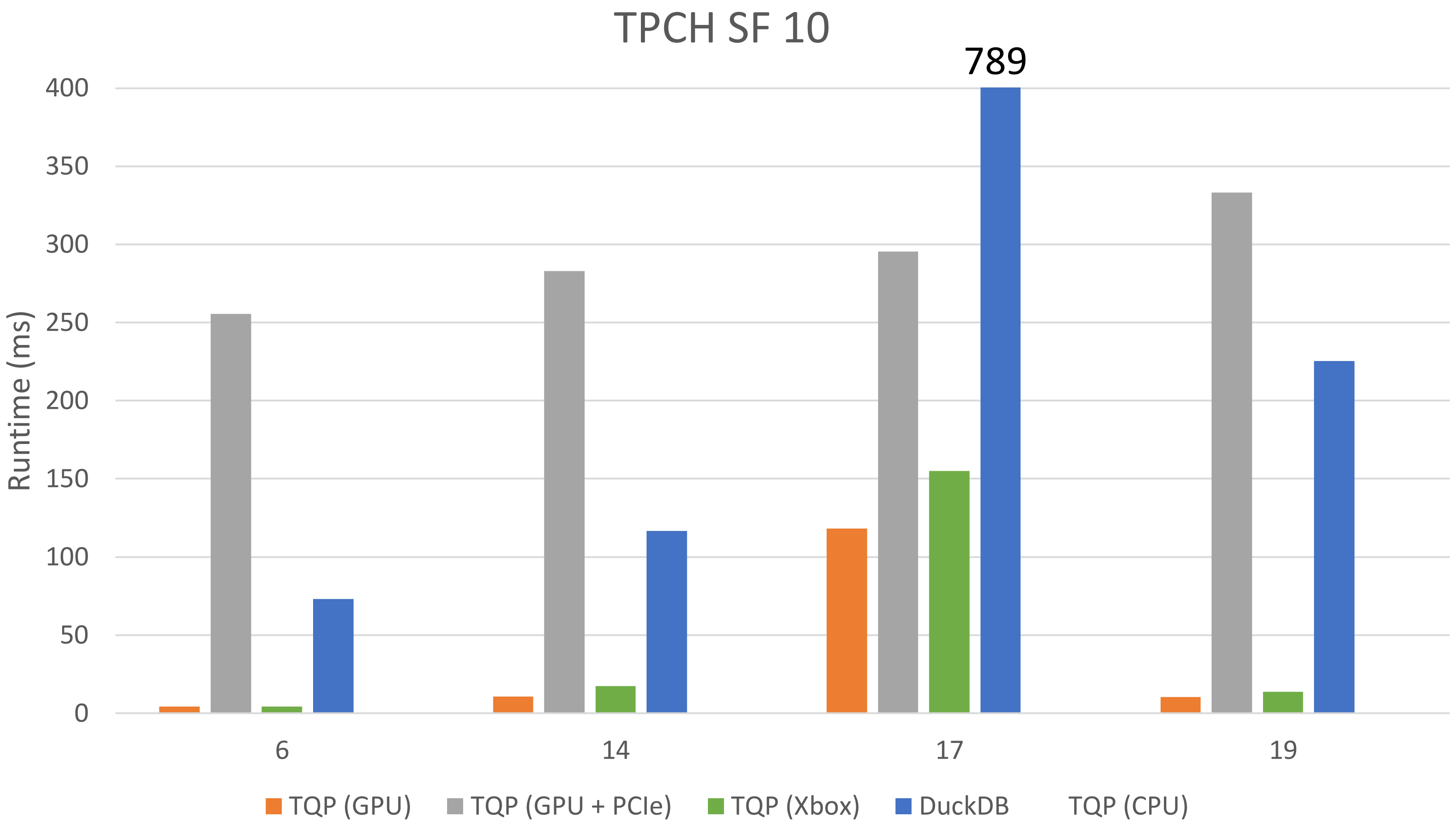


GPU: NVIDIA P100
Xbox: Series X
CPU: Xeon E5-2690 (14 cores)

	Xeon E5-2690	P100	Xbox Series X
Memory Bandwidth (GB/s)	154	732	560
Unidirectional PCIv3 (GB/s)	-	16	-
Theoretical TFLOps	1.4	9.5	12.0

TPCH on Xbox (SF 10, P100)

TQP
[DAMON'23]



GPU: NVIDIA P100
Xbox: Series X
CPU: Xeon E5-2690 (14 cores)

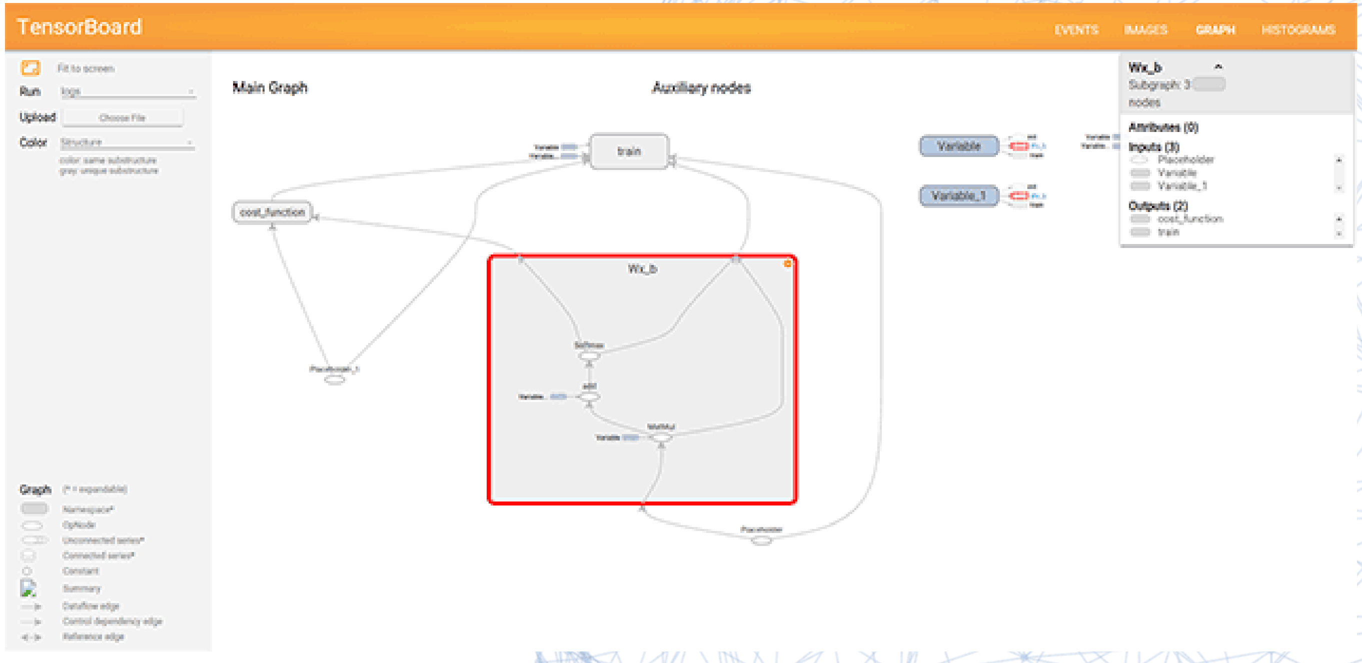
	Xeon E5-2690	P100	Xbox Series X
Memory Bandwidth (GB/s)	154	732	560
Unidirectional PCIv3 (GB/s)	-	16	-
Theoretical TFLOPs	1.4	9.5	12.0

AI-Centric Database System

TQP DEMO
[PVLDB'22]

Broader implications of having a DBMS on an ML runtime backend

Integration with AI ecosystem

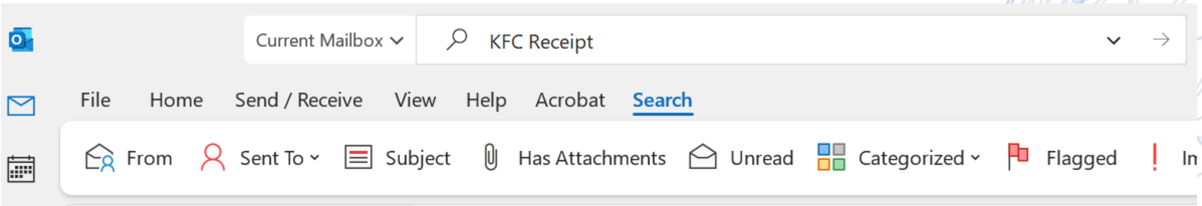


AI-Centric Database System

TQP
[CIDR'23]

Broader implications of having a DBMS on an ML runtime backend

Multi-modal data support



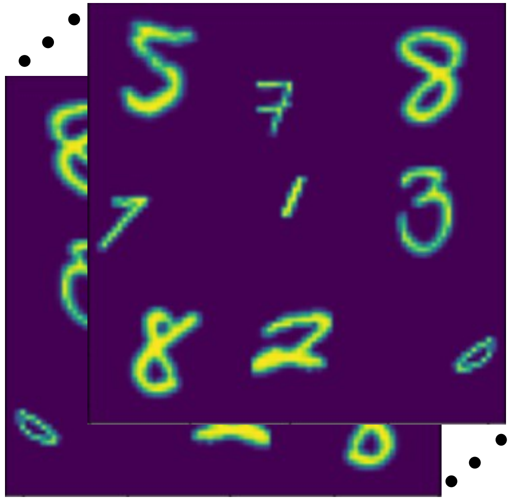

```sql
SELECT
    input AS images,
    image_text_similarity_model("KFC Receipt", input) AS score
FROM attachments
ORDER BY score DESC
LIMIT 1
```

AI-Centric Database System

TQP
[CIDR'23]

Broader implications of having a DBMS on an ML runtime backend

Automatic Differentiation



Example Query Inputs

```
SELECT Digit, Size, COUNT(*)
FROM parseImageToTable(Image)
GROUP BY Digit, Size
```

Trainable Table Value Function

Example Query Outputs

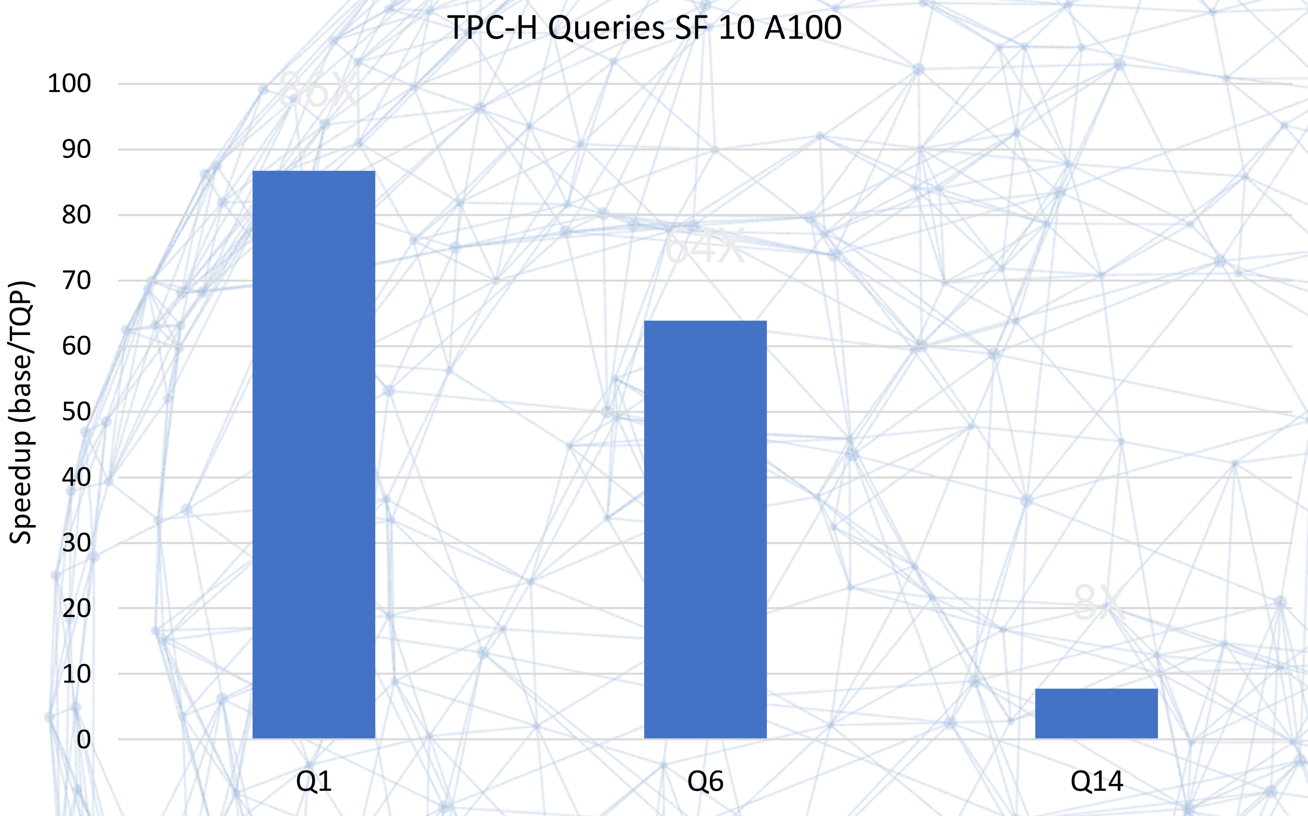
Digit	Size	Count
0	Small	1
	Large	0
1	Small	1
	Large	0
2	Small	0
	Large	1
3	Small	0
	Large	1
4	Small	0
	Large	0
5	Small	0
	Large	1
6	Small	0
	Large	0
7	Small	2
	Large	0
8	Small	0
	Large	2
9	Small	0
	Large	0

Digi		
0		
1		
2		
3		
4		
5		
6		
7		
8		
9	Small	0
	Large	2

AI-Centric Database System

Broader implications of having a DBMS on an ML runtime backend

Tensorframes
as accelerator for
Pandas Dataframes



Apache SystemDS

Apache SystemML™

07/2020 Renamed to SystemDS
05/2017 Apache Top-Level Project
11/2015 Apache Incubator Project
08/2015 Open Source Release

APIs: Command line, JMLC, Spark MLContext, Spark ML, (20+ Scalable Algorithms)

DML Scripts

Language

Compiler

Runtime

Write Once, Run Anywhere

[SIGMOD'15,'17,'19,'21abc,'23abc,'24]
[PVLDB'14,'16ab,'18,'22]
[ICDE'11,'12,'15]
[CIDR'17,'20]
[VLDBJ'18]
[CIKM'22]
[DEBull'14]
[PPoPP'15]

In-Memory Single Node
(scale-up)

Hadoop or Spark Cluster
(scale-out)

Federated
(LA progs, PS)

Other Prototype Backends:
Netezza
Apache Flink

GPUs

since 2014/16

Java

since 2012

hadoop

since 2010/11

Apache Spark

since 2015

exdra

since 2019

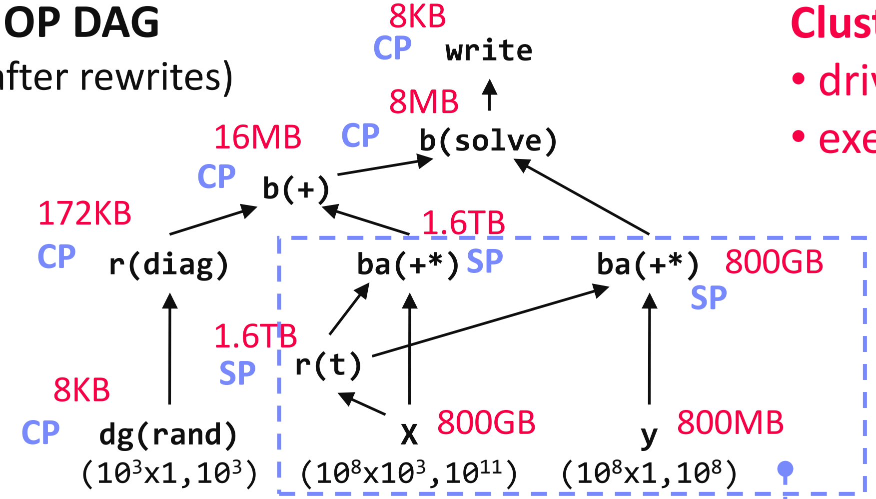
Apache SystemDS: Compilation & Execution

LinregDS (Direct Solve)

```
X = read($1);
y = read($2);
intercept = $3;
lambda = 0.001;
...
if( intercept == 1 ) {
  ones = matrix(1, nrow(X), 1);
  X = append(X, ones);
}
I = matrix(1, ncol(X), 1);
A = t(X) %*% X + diag(I)*lambda;
b = t(X) %*% y;
beta = solve(A, b);
...
write(beta, $4);
```

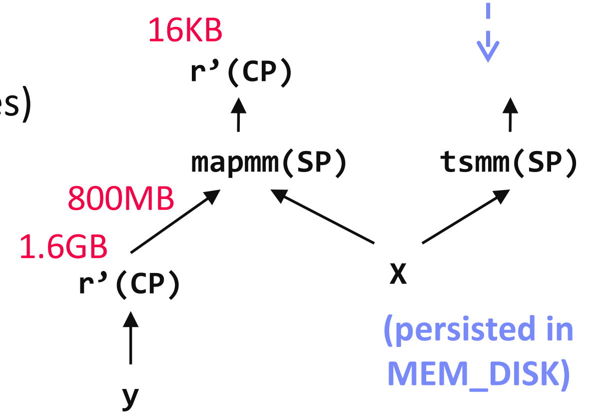
Scenario:
X: 10^8 x 10^3, 10^{11}
y: 10^8 x 1, 10^8

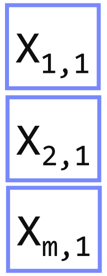
HOP DAG
(after rewrites)



Cluster Config:
- driver mem: 20 GB
- exec mem: 60 GB

LOP DAG
(after rewrites)





➔ **Distributed Matrices**
- **Fixed-size matrix blocks**
- **Data-parallel operations**

➔ **Hybrid Runtime Plans:**
- **Size propagation / memory estimates**
- **Integrated CP / Spark runtime**
- **Dynamic recompilation during runtime**

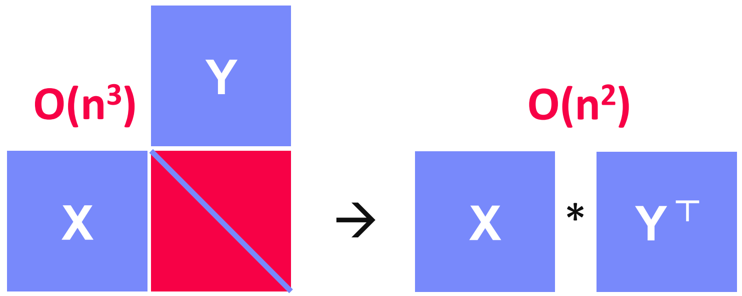
Other Systems:
In-RDBMS, RIOT

[DEBull’14, CIDR’17, SIGMOD’19]

Apache SystemDS: Rewrites

- **Example Static Rewrites** (size-independent)
  - Common Subexpression Elimination
  - Constant Folding / Branch Removal / Block Sequence Merge
  - **Static Simplification Rewrites**
  - Right/Left Indexing Vectorization
  - For Loop Vectorization
  - Spark checkpoint/repartition injection

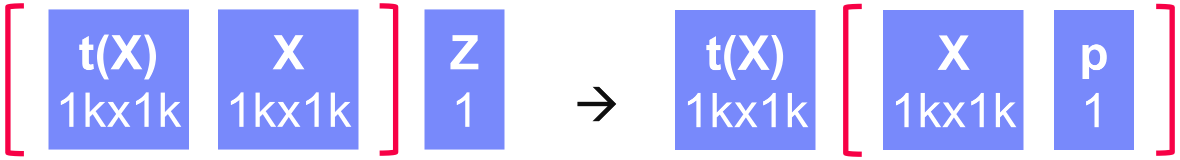
`trace(X%*%Y) → sum(X*t(Y))`



`sum(λ*X) → λ*sum(X)`
`sum(X+Y) → sum(X)+sum(Y)`

- **Example Dynamic Rewrites** (size-dependent)
  - **Dynamic Simplification Rewrites**
  - **Matrix Mult Chain Optimization**

`rowSums(X) → X, iff ncol(X)=1`
`sum(X^2) → X%*%t(X), iff ncol(X)=1`



2,002 MFLOPs **4 MFLOPs**

Size propagation and sparsity estimation

Other Systems:
TensorFlow, PyTorch

Sparsity Estimation & Sparse DP Enum
[MNC @ SIGMOD’19]

[SPOOF @ CIDR'17, PVLDB'18]

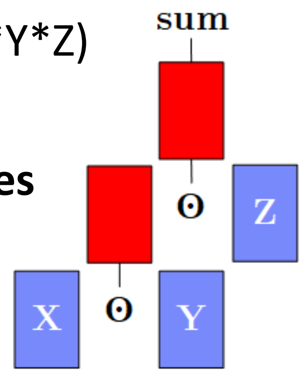
Apache SystemDS: Operator Fusion & Codegen

- **Motivation:** DAGs of linear algebra (LA) operations and statistical functions with materialized intermediates → **ubiquitous fusion opportunities**

- **Examples**

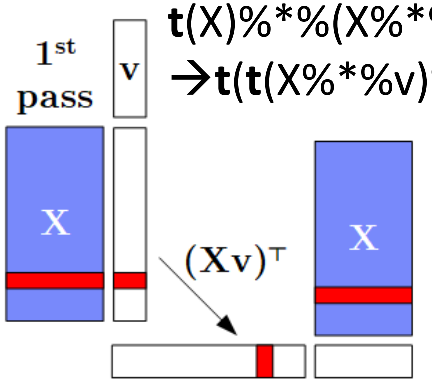
sum(X*Y*Z)

a) Intermediates

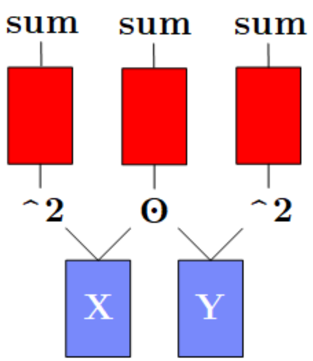


b) Single-Pass

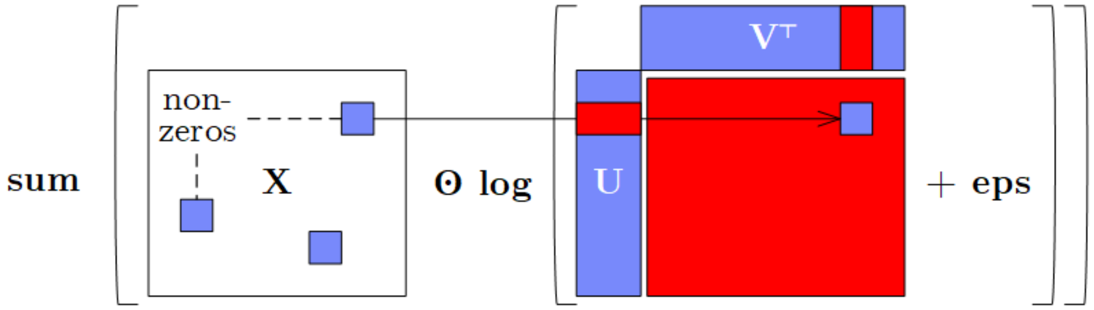
t(X)%*%(X%*%v)
→**t(t(**X%*%v)%*%X)

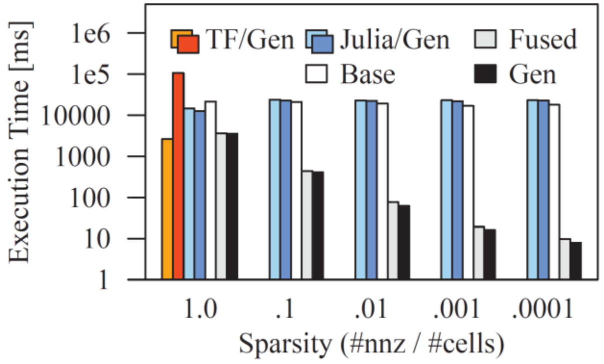


c) Multi-Aggregates



d) Sparsity Exploitation





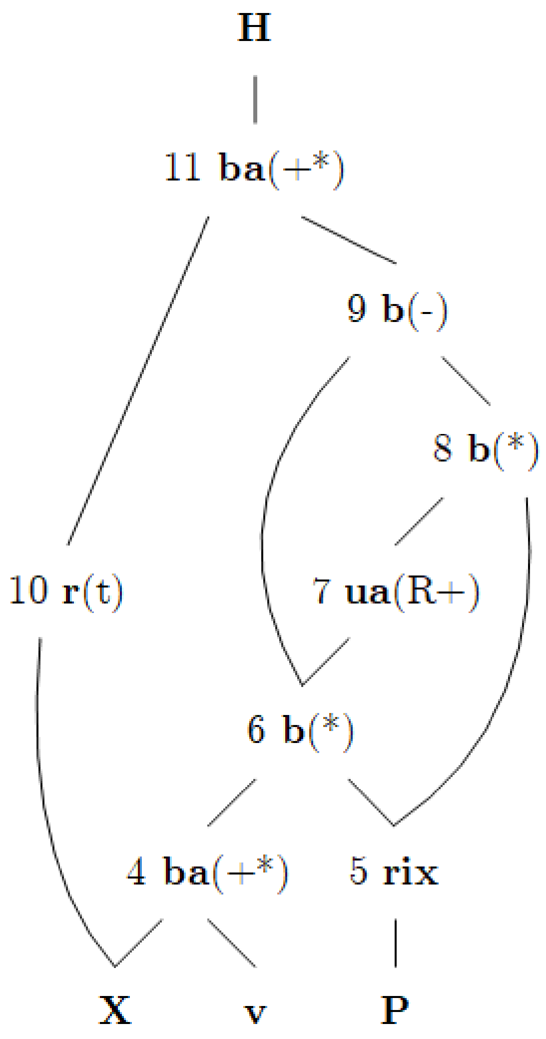
Other Systems:

BTO, Tupleware, Kasen, Weld, TACO, Julia, TF XLA, JAX, TVM, DAPHNE, PyTorch, Triton

SystemDS: Operator Fusion & Codegen, cont.

- **MLogreg Inner Loop** (main expr on feature matrix X)

```
1: Q = P[, 1:k] * (X %*% v)
2: H = t(X) %*% (Q - P[, 1:k] * rowSums(Q))
```

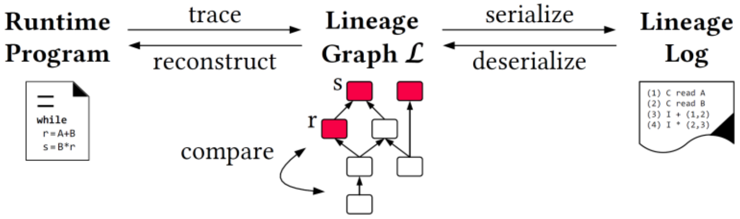



```
public final class TMP25 extends SpoofRow {
  public TMP25() {
    super(RowType.COL_AGG_B1_T, true, 5);
  }
  protected void genexecDense(double[] a, int ai,
   SideInput[] b, double[] c,..., int len) {
    double[] TMP11 = getVector(b[1].vals(rix),...);
    double[] TMP12 = vectMatMult(a, b[0].vals(rix),...);
    double[] TMP13 = vectMult(TMP11, TMP12, 0, 0,...);
    double TMP14 = vectSum(TMP13, 0, TMP13.length);
    double[] TMP15 = vectMult(TMP11, TMP14, 0,...);
    double[] TMP16 = vectMinus(TMP13, TMP15, 0, 0,...);
    vectOuterMultAdd(a, TMP16, c, ai, 0, 0,...); }
  protected void genexecSparse(double[] avals, int[] aix,
   int ai, SideInput[] b, ..., int len) {...}
}
```

[LIMA @ SIGMOD'21]


Apache SystemDS: Lineage-based Reuse

- **Lineage as Key Enabling Technique**
  - Trace lineage of ops (incl. non-determinism), dedup for loops/funcs
  - Model versioning, data reuse, incr. maintenance, autodiff, debugging



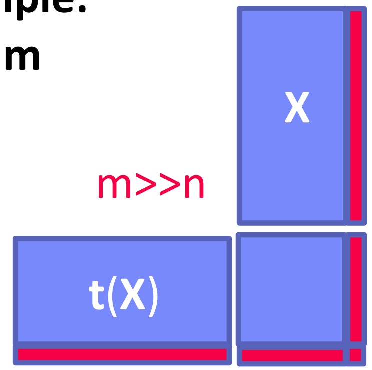

```
for( i in 1:numModels )
  R[,i] = lm(X, y, lambda[i,], ...)
```

- **Full Reuse of Intermediates**
  - Before executing instruction, probe output lineage in cache `Map<Lineage, MatrixBlock>`
  - Cost-based/heuristic caching and eviction decisions (compiler-assisted)

```
m_lmDS = function(...) {
  l = matrix(reg,ncol(X),1)
  A = t(X) %*% X + diag(l)
  b = t(X) %*% y
  beta = solve(A, b) ...}
```

- **Partial Reuse of Intermediates**
  - **Problem:** Often partial result overlap
  - Reuse partial results via dedicated rewrites (compensation plans)

Example: steplm



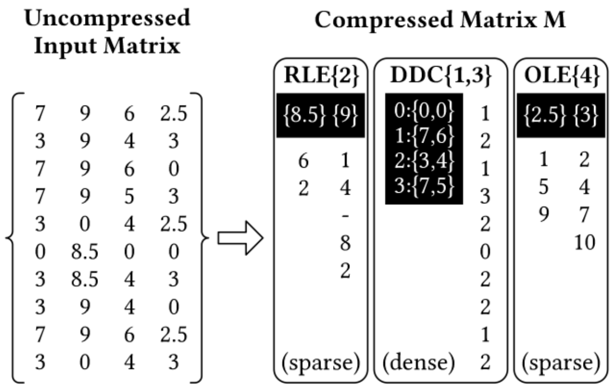

```
m_steplm = function(...) {
  while( continue ) {
    parfor( i in 1:n ) {
      if( !fixed[1,i] ) {
        Xi = cbind(Xg, X[,i])
        B[,i] = lm(Xi, y, ...)
    } }
    # add best to Xg (AIC)
} }
```

Other Systems:

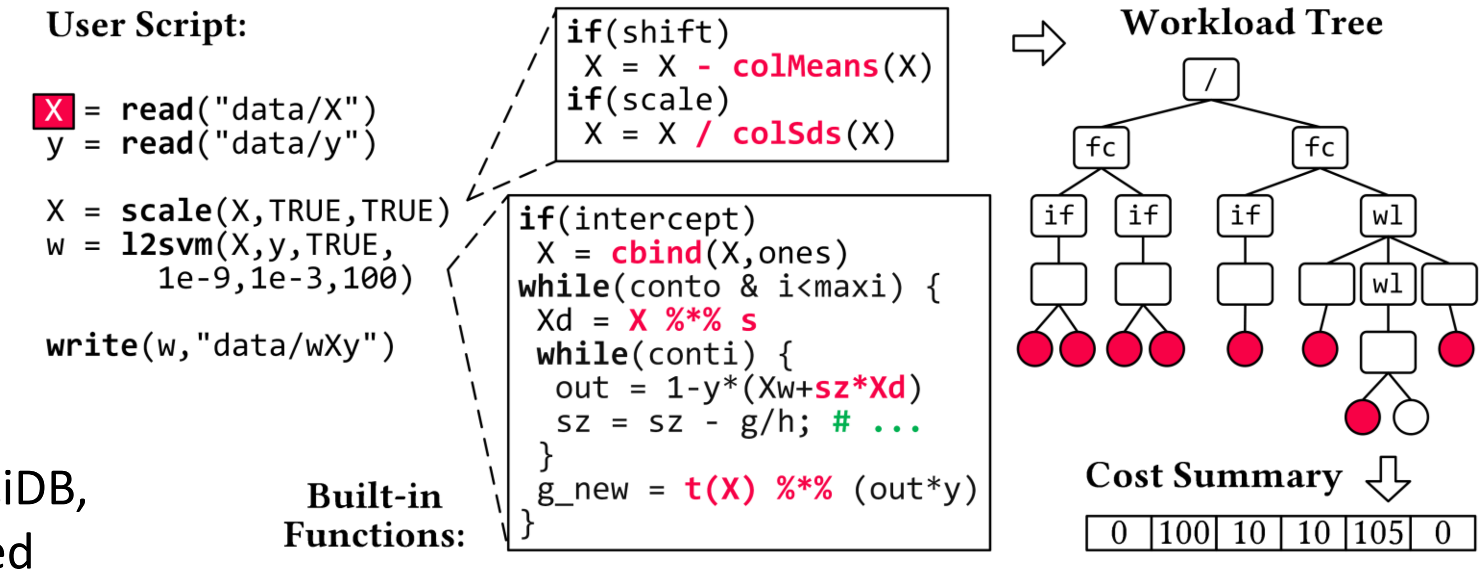
COLUMBUS, KeystoneML, Helix, PRETZEL, MISTIQUE, Alpine Meadow, Collaborative Optimizer

Apache SystemDS: Workload-aware CLA

- **Lossless Matrix Compression**
  - **Improved general applicability** (adaptive compression time, new compression schemes, new kernels, intermediates, workload-aware)
  - Sparsity → **Redundancy exploitation** (data redundancy, structural redundancy)



- **Workload-aware Compression**
  - Workload summary → compression
  - Compressed Representation → execution planning



Other Systems:
TOC, Spark/Flink, NetCDF/HDF5, SciDB, TileDB, Sprintz, Grammar, Factorized

Runtime for Tensor-Relational Computations

- **Tensor-based computations can easily be specified relationally**
  - Why? Consider Einstein summation notation
- **These are all relational computations!**
  - Always an (optional) aggregation
  - On top of a projection
  - On top of an (optional) join tree
- **Ex: MatMul**

$$\forall_{ik} C_{ik} \leftarrow \sum_j A_{ij} \times B_{jk}$$

  - First join A and B on *j* index
  - Then projection to multiply matched entries
  - Then aggregate, grouping on *i* and *k* indices

Example… Multiplying

$$\mathrm{SoftMax}(W^{(2)} \times \mathrm{ReLU}(W^{(1)} \times X))$$

$$A_i \leftarrow \sum_j W^{(1)}_{i,j} \times X_j$$

$$\sum_{\emptyset} \mathrm{ReLU}(A_i)$$

$$\sum_j W^{(2)}_{i,j} \times B_j$$

$$D_{\emptyset} \leftarrow \sum_i \exp(C_i)$$

$$E_i \leftarrow \sum_{\emptyset} \frac{\exp(C_i)}{D_{\emptyset}}$$

General Relativity [Einstein16]

But Pure Relational Can Be Slow

- **Relational is good: we know how to scale relational computations**
  - But pure relational (each entry in a tensor is a tuple) will not perform well
  - Why? Relational mult of two 80K by 80K matrices produces 512 trillion intermediate tuples
  - Small overhead associated with each tuple means performance is poor
- **Tensor relations allow the best of both**

Consider the matrix:

$$\begin{bmatrix} 1.4 & 2.2 & 1.2 & 2.1 \\ 2.3 & 2.6 & 1.1 & 2.2 \\ 1.4 & 1.0 & 1.1 & 1.4 \\ 1.1 & 1.4 & 2.5 & 2.3 \end{bmatrix}$$

Decompose into a set of (`rowID, colID, chunk`) triples:

$$\bar{R} = \left\{ \left(\langle 1,1\rangle, \begin{bmatrix} 1.4 & 1.2 \\ 2.3 & 2.6 \end{bmatrix}\right), \left(\langle 1,2\rangle, \begin{bmatrix} 1.2 & 2.1 \\ 1.1 & 2.2 \end{bmatrix}\right), \left(\langle 2,1\rangle, \begin{bmatrix} 1.4 & 1.0 \\ 1.1 & 1.4 \end{bmatrix}\right), \left(\langle 2,2\rangle, \begin{bmatrix} 1.1 & 1.4 \\ 2.5 & 2.3 \end{bmatrix}\right) \right\}$$

Tensor Relational Algebra [VLDB21a]

Tensor Relations

- **These are relations**
  - So distributing to multiple machines/devices (GPUs) is easy
  - Scaling to very large computations (operands don't fit in GPU RAM) is also easy
  - But actual data manipulations over sub-tensors done with efficient CPU/GPU kernels
- **Example: MatMul over tensor relations**

Kernels

```
SELECT lhs.rowID, rhs.colID
  mat_sum (mat_mul (lhs.chunk, rhs.chunk))
FROM A AS lhs, B AS rhs
WHERE lhs.colID = rhs.rowID
GROUP BY lhs.rowID, rhs.colID
```

Imagine two 80K by 80K matrices decomposed into 1K by 1K chunks

Now only 80 X 80 X 80 = 512K intermediate tuples from join; low overhead, but plenty of parallelism

Tensor-Relational Computations: Use a DBMS?

- **DBMS engine can be improved. Consider MatMul on multiple machines/devices**

Imagine two 80K by 80K matrices decomposed into 6400, 1K by 1K chunks; perform compute on 256 machines/devices

```
SELECT lhs.rowID, rhs.colID
  mat_sum (mat_mul (lhs.chunk, rhs.chunk))
FROM A AS lhs, B AS rhs
WHERE lhs.colID = rhs.rowID
GROUP BY lhs.rowID, rhs.colID
```


Distributed Numerical and Machine Learning Computations via
Two-Phase Execution of Aggregated Join Trees

Agg'ed Join Trees
[VLDB21b]

Join produces 6400 x 80 result chunks

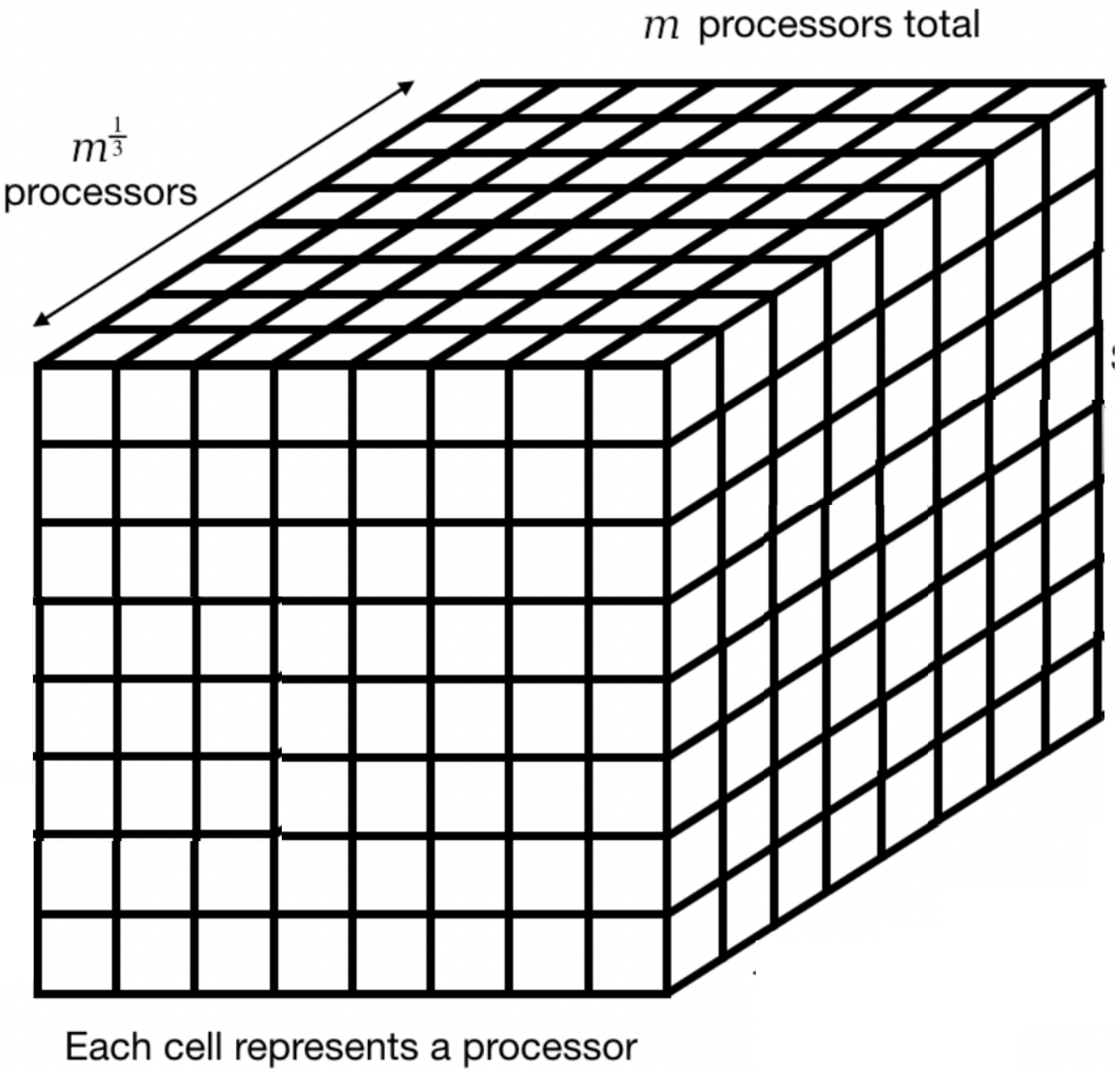
Hash partitioned into 6400 buckets to aggregate

Join can only keep 80 machines/devices busy (just 80 unique join keys)

Agg requires transferring approximately 6400 x 80 = 512000 chunks across machines/devices

However, there's an algorithm that can do the mult with 3 x 8 x 6400 = 153600 xfers, all 256 busy

Can Do Much Better




A three-dimensional approach to parallel matrix multiplication

3D MatMul
[IBM95]

Optimal 3D algorithm

Replicate A $m^{1/3}$ times

Replicate B $m^{1/3}$ times

Reduce: $m^{1/3}$ xfers of partial outputs

Asymptotically less comm than relational by a $m^{1/3}$ factor

Each processor performs pairwise mult of local chunks

What Is the Issue?

- **DBMS treats all relational ops as separate**
  - Best way to do the join: hash A, hash B—just xfer A and B once
  - But this sets you up for a terrible aggregation…
  - Xfer equivalent to 80x the size of input matrix in our example
  - Compare to 8x in the case of the 3D in our example
- **3D multiply xfers A and B eight times during join---so join is more expensive**
  - But then aggregation is set up nicely, so much faster
  - Trade-off: expensive join for inexpensive agg is not available to a DBMS

Tensor-Relational: New Engine Needed

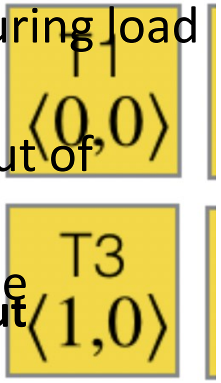
- **Such computations are fundamentally different from classical relational**
  - Few tuples, but very large
  - Computationally intensive part is running the kernels, **not** linking tuples
  - Time required to transfer subtensors across machines dominates
- **How should a tensor-relational engine work?**
  - Use relational lineage as input to an optimization problem
  - Where to place kernels
  - So as to minimize communication, while ensuring load balancing
  - Can plan everything, down to swaps in and out of device memory!
  - Then execute on a specialized dataflow engine

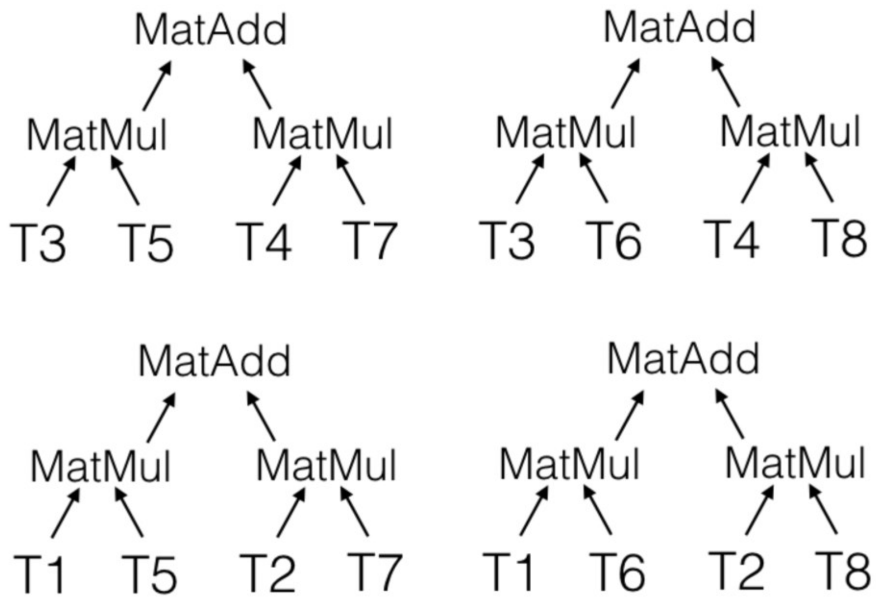
```
SELECT lhs.rowID, rhs.colID
  mat_sum (mat_mul (lhs.chunk, rhs.chunk))
FROM A AS lhs, B AS rhs
WHERE lhs.colID = rhs.rowID
GROUP BY lhs.rowID, rhs.colID
```

Given this

First create variants of A, B that have tensor IDs, dropping tensors

Execute the computation over tuples with tensors removed, **but only collect lineage**





How Well Does This Work?

- **Simple comparison**
  - Tensor-relational runtime with lineage-based planning vs. broadcast matrix multiply
  - CPU cluster, Amazon EC2
  - Multiply non-square matrices of size $N \times K$ and $K \times M$, split into 1K by 1K chunks

N	M	K	BMM	Planning
Cluster with 2 workers				
1K	1K	1000K	17.19s	5.03s
1K	40K	40K	16.98s	7.49s
Cluster with 7 workers				
1K	1K	1000K	16.62s	2.99s
1K	40K	40K	16.87s	3.15s
Cluster with 16 workers				
1K	1K	1000K	18.40s	2.50s
1K	40K	40K	19.78s	2.57s

References

[**DaMoN'23**] Wei Cui, Qianxi Zhang, Spyros Blanas, Jesús Camacho-Rodríguez, Brandon Haynes, Yinan Li, Ravi Ramamurthy, Peng Cheng, Rathijit Sen, Matteo Interlandi: Query Processing on Gaming Consoles. DaMoN 2023

[**CIDR'23**] Apurva Gandhi, Yuki Asada, Victor Fu, Advitya Gemawat, Lihao Zhang, Rathijit Sen, Carlo Curino, Jesus Camacho-Rodriguez, and Matteo Interlandi: The Tensor Data Platform: Towards an AI-centric Database System. CIDR 2023.

[**SIGMOD'23**] Sebastian Baunsgaard, Matthias Boehm: AWARE: Workload-aware, Redundancy-exploiting Linear Algebra. SIGMOD 2023

[**CIDR'22**] Patrick Damme et al:: DAPHNE: An Open and Extensible System Infrastructure for Integrated Data Analysis Pipelines. CIDR 2022

[**CIKM'22**] Sebastian Baunsgaard, Matthias Boehm, Kevin Innerebner, Mito Kehayov, Florian Lackner, Olga Ovcharenko, Arnab Phani, Tobias Rieger, David Weissteiner, Sebastian Benjamin Wrede: Federated Data Preparation, Learning, and Debugging in Apache SystemDS. CIKM 2022

[**PVLDB'22a**] Dong He, Supun Chathuranga Nakandala, Dalitso Banda, Rathijit Sen, Karla Saur, Kwanghyun Park, Carlo Curino, Jesús Camacho-Rodríguez, Konstantinos Karanasos, Matteo Interlandi: Query Processing on Tensor Computation Runtimes. PVLDB 15(11): 2811-2825 (2022)

[**PVLDB'22b**] Yuki Asada, Victor Fu, Apurva Gandhi, Advitya Gemawat, Lihao Zhang, Vivek Gupta, Ehi Nosakhare, Dalitso Banda, Rathijit Sen, Matteo Interlandi: Share the Tensor Tea: How Databases can Leverage the Machine Learning Ecosystem. PVLDB 15(12): 3598-3601 (2022)

[**PVLDB'22c**] Arnab Phani, Lukas Erlbacher, Matthias Boehm: UPLIFT: Parallelization Strategies for Feature Transformations in Machine Learning Workloads. PVLDB 2022

[**PVLDB21a**] Binhang Yuan, Dimitrije Jankov, Jia Zou, Yuxin Tang, Daniel Bourgeois, Chris Jermaine: Tensor Relational Algebra for Distributed Machine Learning System Design. Proc. VLDB Endow. 14(8): 1338-1350 (2021)

[**PVLDB21b**] Dimitrije Jankov, Binhang Yuan, Shangyu Luo, Chris Jermaine: Distributed Numerical and Machine Learning Computations via Two-Phase Execution of Aggregated Join Trees. Proc. VLDB Endow. 14(7): 1228-1240 (2021)

[**SIGMOD'21**] Arnab Phani, Benjamin Rath, Matthias Boehm: LIMA: Fine-grained Lineage Tracing and Reuse in Machine Learning Systems. SIGMOD 2021

[**SIGMOD'21**] Sebastian Baunsgaard, Matthias Boehm, Ankit Chaudhary, Behrouz Derakhshan, Stefan Geißelsöder, Philipp M. Grulich, Michael Hildebrand, Kevin Innerebner, Volker Markl, Claus Neubauer, Sarah Osterburg, Olga Ovcharenko, Sergey Redyuk, Tobias Rieger, Alireza Rezaei Mahdiraji, Sebastian Benjamin Wrede, Steffen Zeuch: ExDRa: Exploratory Data Science on Federated Raw Data. SIGMOD 2021

[**CIDR'20**] Matthias Boehm, Iulian Antonov, Sebastian Baunsgaard, Mark Dokter, Robert Ginthör, Kevin Innerebner, Florijan Klezin, Stefanie N. Lindstaedt, Arnab Phani, Benjamin Rath, Berthold Reinwald, Shafaq Siddiqui, Sebastian Benjamin Wrede: SystemDS: A Declarative Machine Learning System for the End-to-End Data Science Lifecycle. CIDR 2020

[**SIGMOD'19**] Johanna Sommer, Matthias Boehm, Alexandre V. Evfimievski, Berthold Reinwald, Peter J. Haas: MNC: Structure-Exploiting Sparsity Estimation for Matrix Expressions. SIGMOD 2019

References, cont.

[**PVLDB’18**] Matthias Boehm, Berthold Reinwald, Dylan Hutchison, Prithviraj Sen, Alexandre V. Evfimievski, Niketan Pansare: On Optimizing Operator Fusion Plans for Large-Scale Machine Learning in SystemML. Proc. VLDB Endow. 11(12): 1755-1768 (2018)

[**VLDBJ’18**] Ahmed Elgohary, Matthias Boehm, Peter J. Haas, Frederick R. Reiss, Berthold Reinwald: Compressed linear algebra for large-scale machine learning. VLDB J. 27(5): 719-744 (2018)

[**CIDR’17**] Tarek Elgamal, Shangyu Luo, Matthias Boehm, Alexandre V. Evfimievski, Shirish Tatikonda, Berthold Reinwald, Prithviraj Sen: SPOOF: Sum-Product Optimization and Operator Fusion for Large-Scale Machine Learning. CIDR 2017

[**ICDE17**] Shangyu Luo, Zekai J. Gao, Michael N. Gubanov, Luis Leopoldo Perez, Christopher M. Jermaine: Scalable Linear Algebra on a Relational Database System. ICDE 2017: 523-534

[**PVLDB’16b**] Ahmed Elgohary, Matthias Boehm, Peter J. Haas, Frederick R. Reiss, Berthold Reinwald: Compressed Linear Algebra for Large-Scale Machine Learning. PVLDB 2016

[**PVLDB’16a**] Matthias Boehm, Michael Dusenberry, Deron Eriksson, Alexandre V. Evfimievski, Faraz Makari Manshadi, Niketan Pansare, Berthold Reinwald, Frederick Reiss, Prithviraj Sen, Arvind Surve, Shirish Tatikonda: SystemML: Declarative Machine Learning on Spark. PVLDB 9(13): 1425-1436 (2016)

Botong Huang, Matthias Boehm, Yuanyuan Tian, Berthold Reinwald, Shirish Tatikonda, Frederick R. Reiss: Resource Elasticity for Large-Scale Machine Learning. SIGMOD 2015

[**DEBul’14**] Matthias Boehm, Douglas R. Burdick, Alexandre V. Evfimievski, Berthold Reinwald, Frederick R. Reiss, Prithviraj Sen, Shirish Tatikonda, Yuanyuan Tian: SystemML's Optimizer: Plan Generation for Large-Scale Machine Learning Programs. IEEE Data Eng. Bull. 37(3): 52-62 (2014)

[**PVLDB’14**] Matthias Boehm, Shirish Tatikonda, Berthold Reinwald, Prithviraj Sen, Yuanyuan Tian, Douglas Burdick, Shivakumar Vaithyanathan: Hybrid Parallelization Strategies for Large-Scale Machine Learning in SystemML. PVLDB. 7(7): 553-564 (2014)

[**SIGMOD13**] Zhuhua Cai, Zografoula Vagena, Luis Leopoldo Perez, Subramanian Arumugam, Peter J. Haas, Christopher M. Jermaine: Simulation of database-valued Markov chains using SimSQL. SIGMOD Conference 2013: 637-648

[**Savage09**] Savage, Sam L., and Harry M. Markowitz. *The flaw of averages: Why we underestimate risk in the face of uncertainty*. John Wiley & Sons, 2009.

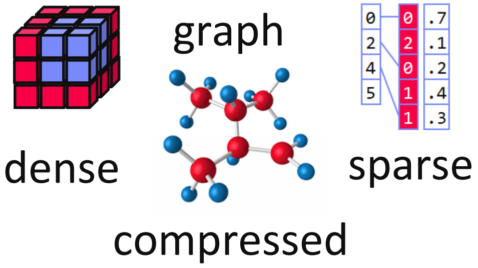
[**SIGMOD08**] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher M. Jermaine, Peter J. Haas: MCDB: a monte carlo approach to managing uncertain data. SIGMOD Conference 2008: 687-700

[**IBM95**] Agarwal, Ramesh C., et al. "A three-dimensional approach to parallel matrix multiplication." *IBM Journal of Research and Development* 39.5 (1995): 575-582.

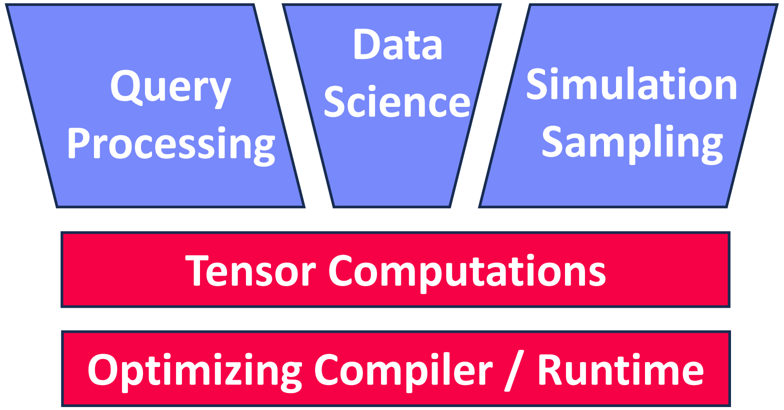
[**Einstein16**] Einstein, Albert (1916). "The Foundation of the General Theory of Relativity". Annalen der Physik.

Conclusions

- **Future: Broader Focus (on General Tensor Computations)**
  - General linear algebra programs and tensor computations
  - Different architectures (parameter servers, data-/task-task parallel)
  - Wide variety of applications and workload characteristics



- **Long-term Benefits**
  - Simplicity
  - Reuse of Compilation and Runtime Techniques
  - Performance and Scalability



- **Lots of Open Challenges and Research Opportunities**