SystemDS: A Declarative ML System for the End-to-End Data Science Lifecycle

Matthias Boehm[1,2], Iulian Antonov[2], Sebastian Baunsgaard[1], Mark Dokter[2], Robert Ginthör[2], Kevin Innerebner[1], Florijan Klezin[2], Stefanie Lindstaedt[1,2], Arnab Phani[1], Benjamin Rath[1], Berthold Reinwald[3], Shafaq Siddiqi[1], Sebastian Benjamin Wrede[2]

[1] **Graz University of Technology**; Graz, Austria
[2] **Know-Center GmbH**; Graz, Austria
[3] **IBM Research – Almaden**; San Jose, CA, USA

Motivation SystemDS

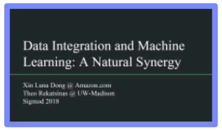
- **Existing ML Systems**
  - **#1 Numerical computing** frameworks
  - **#2 ML Algorithm libraries** (local, large-scale)
  - **#3 Linear algebra ML systems** (large-scale)
  - **#4 Deep neural network** (DNN) frameworks
  - **#5 Model management, and deployment**
- **Exploratory Data-Science Lifecycle**
  - **Open-ended problems** w/ underspecified objectives
  - Hypotheses, data integration, run analytics
  - **Unknown value** → lack of system infrastructure
    → **Redundancy of manual efforts and computation**

"Take these datasets and show value or competitive advantage"

- **Data Preparation Problem**
  - **80% Argument:** 80-90% time for finding, integrating, cleaning data
  - Diversity of tools ➔ boundary crossing, lack of optimization
  - **In-DBMS ML** toolkits **largely unsuccessful** (stateful, data loading, verbose)

Motivation SystemDS, cont.

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



- **Key Observation**
  - **SotA data integration based on ML**
    (e.g., data extraction, schema alignment, entity linking)
  - **Similar:** data cleaning, outlier detection, missing value imputation, semantic type detection, data augmentation, feature selection, hyper parameter optimization, model debugging

- **A Case for Declarative Data Science**
  - High-level abstractions (**R/Python**, **stateless**) for lifecycle tasks, implemented **in DSL for ML training/scoring**
  - **Avoid boundary crossing** and **optimizations across lifecycle**
  - Control compiler and runtime of utmost importance

Apache SystemML → SystemDS

Architecture and Preliminary Results

SystemML Background

High-Level SystemML Architecture

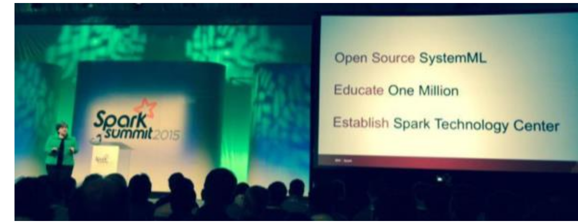
APIs: Command line, JMLC, Spark MLContext, Spark ML, (20+ scalable algorithms)

DML Scripts

Language

Compiler

Runtime



Apache SystemML™

05/2017 Apache Top-Level Project
11/2015 Apache Incubator Project
08/2015 Open Source Release

[SIGMOD'15,'17,'19]
[PVLDB'14,'16a,'16b,'18]
[ICDE'11,'12,'15]
[CIDR'17]
[VLDBJ'18]
[DEBull'14]
[PPoPP'15]

In-Memory Single Node (scale-up)

Hadoop or Spark Cluster (scale-out)

In-Progress:

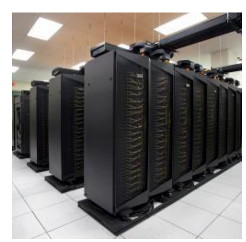
GPU

since 2014/16

Java

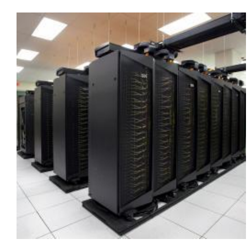
since 2012

hadoop



since 2010/11

Apache Spark



since 2015

Lessons Learned from SystemML

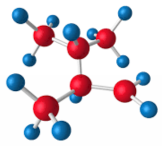
Why was SystemML not adopted in practice?

- **L1 Data Independence & Logical Operations**
  - Independence of **evolving technology stack** (MR → Spark, GPUs)
  - **Simplifies development** (libs) and **deployment** (large-scale vs. embedded)
  - **Enables adaptation** to cluster/data characteristics (dense/spare/compressed)

- **L2 User Categories** (|Alg. Users| **>>** |Alg. Developers|)
  - **Focus on ML researchers** and algorithm developers **is a niche**
  - Data scientists and domain experts **need higher-level abstractions**

- **L3 Diversity of ML Algorithms & Apps**
  - **Variety of algorithms** (batch 1st/2nd, mini-batch DNNs, hybrid)
  - Different parallelization, ML + rules, numerical computing

- **L4 Heterogeneous Structured Data**
  - Support for **feature transformations on 2D frames**
  - Many apps deal with **heterogeneous data and various structure**



SystemDS Architecture

(An open source ML **System** for the end-to-end **D**ata **S**cience lifecycle)

https://github.com/tugraz-isds/systemds,

forked from Apache SystemML 1.2 in Sep 2018

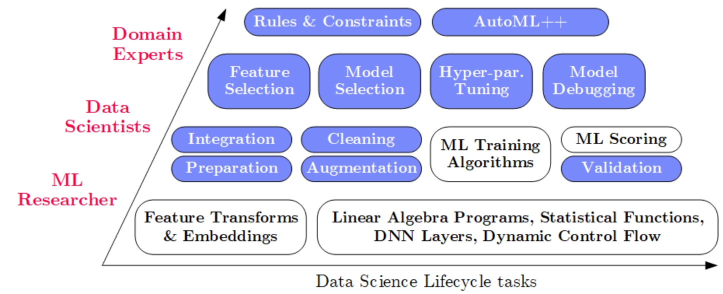
SystemDS 0.1 published Aug 31, 2019
SystemDS 0.2 upcoming

SystemDS Vision and Design

Game Plan

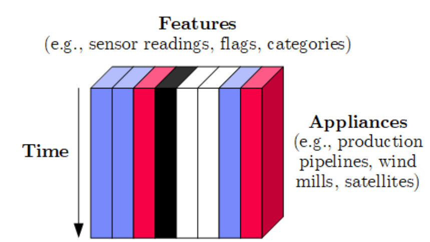
- **Objectives**
  - Effective and efficient **data preparation, ML, and model debugging at scale**
  - High-level abstractions for lifecycle tasks (L3/L4) and users (L2)

- **#1 Based on DSL for ML Training/Scoring**
  - Hierarchy of abstractions for DS tasks
  - ML-based SotA, interleaved, performance



- **#2 Hybrid Runtime Plans and Optimizing Compiler**
  - System infrastructure for diversity of algorithm classes
  - Different parallelization strategies and new architectures (Federated ML)
  - Abstractions → redundancy → automatic optimization

- **#3 Data Model: Heterogeneous Tensors**
  - Data integration/prep requires generic data model



Language Abstractions and APIs, cont.

- **Example: Stepwise Linear Regression**

User Script

```
X = read('features.csv')
Y = read('labels.csv')
[B,S] = steplm(X, Y,
  icpt=0, reg=0.001)
write(B, 'model.txt')
```

Facilitates optimization across data science lifecycle tasks

Built-in Functions

```
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)
} }
```

Feature Selection

```
m_lm = function(...) {
  if( ncol(X) > 1024 )
    B = lmCG(X, y, ...)
  else
    B = lmDS(X, y, ...)
}
```

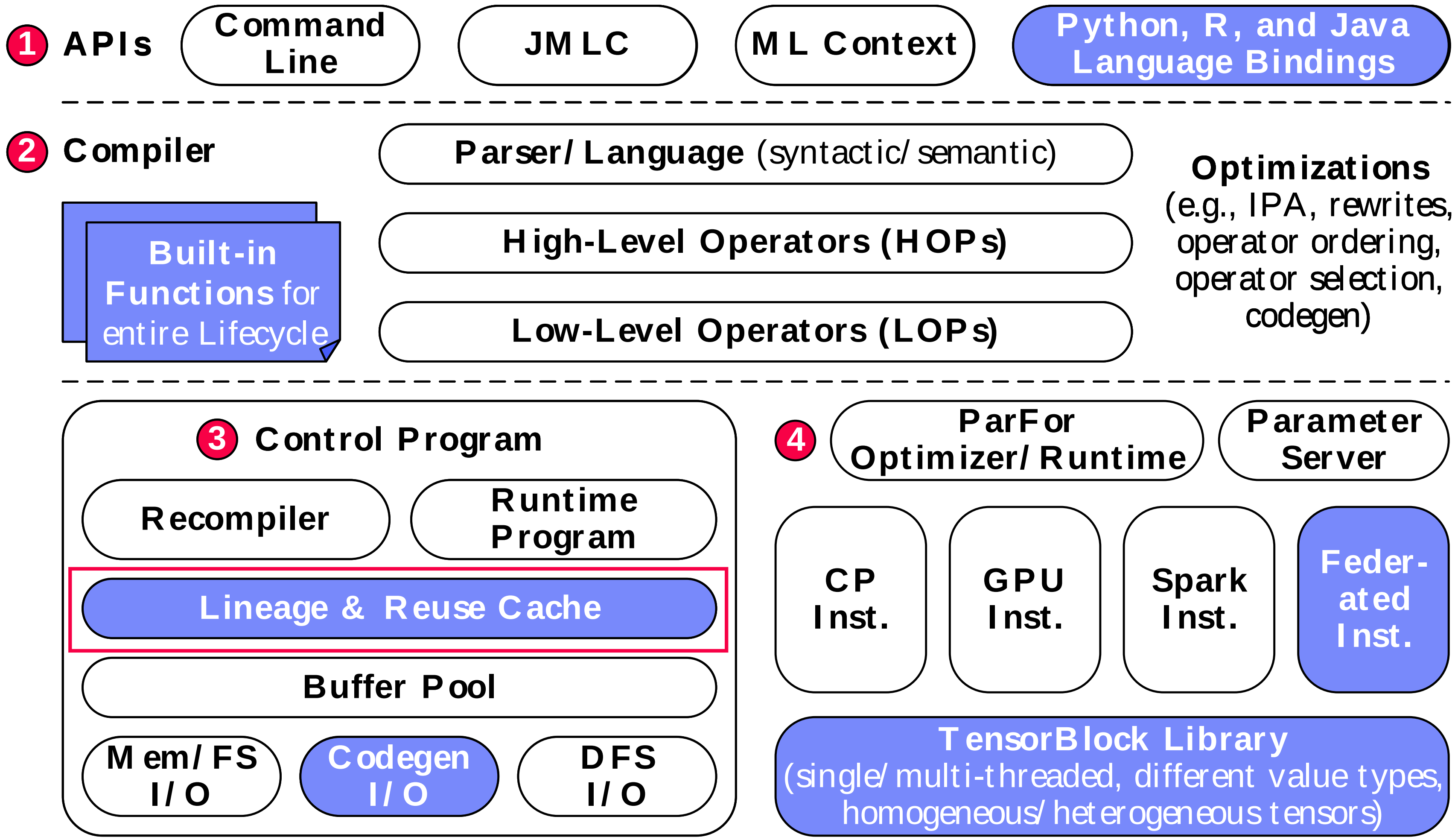
ML Algorithms

```
m_lmCG = function(...) {
  while( i<maxi&nr2>tgt ) {
    q = (t(X) %*% (X %*% p))
      + lambda * p
    beta = ... }
}
```

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

Linear Algebra Programs

System Architecture



Lineage and Reuse

- **Problem**
  - **Exploratory data science** (data preprocessing, model configurations)
  - **Reproducibility** and **explainability** of trained models (data, parameters, prep)

➔ **Lineage/Provenance as Key Enabling Technique**

  - Model versioning, reuse of intermediates, incremental maintenance, auto differentiation, and debugging (results and intermediates, convergence behavior via query processing over lineage traces)

- **a) Efficient Lineage Tracing**
  - Tracing of inputs, literals, and **non-determinism**
  - **Trace lineage of logical operations** for all live variables, store along outputs, program/output reconstruction possible:

```
X = eval(deserialize(serialize(lineage(X))))
```

  - **Proactive deduplication** of lineage traces for loops, (and functions)

Lineage and Reuse, cont.

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

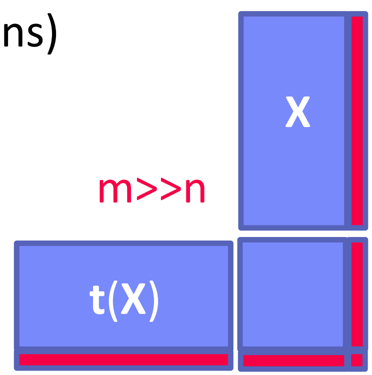
$O(k(mn^2+n^3)) \rightarrow O(mn^2+kn^3)$

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

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

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

m>>n

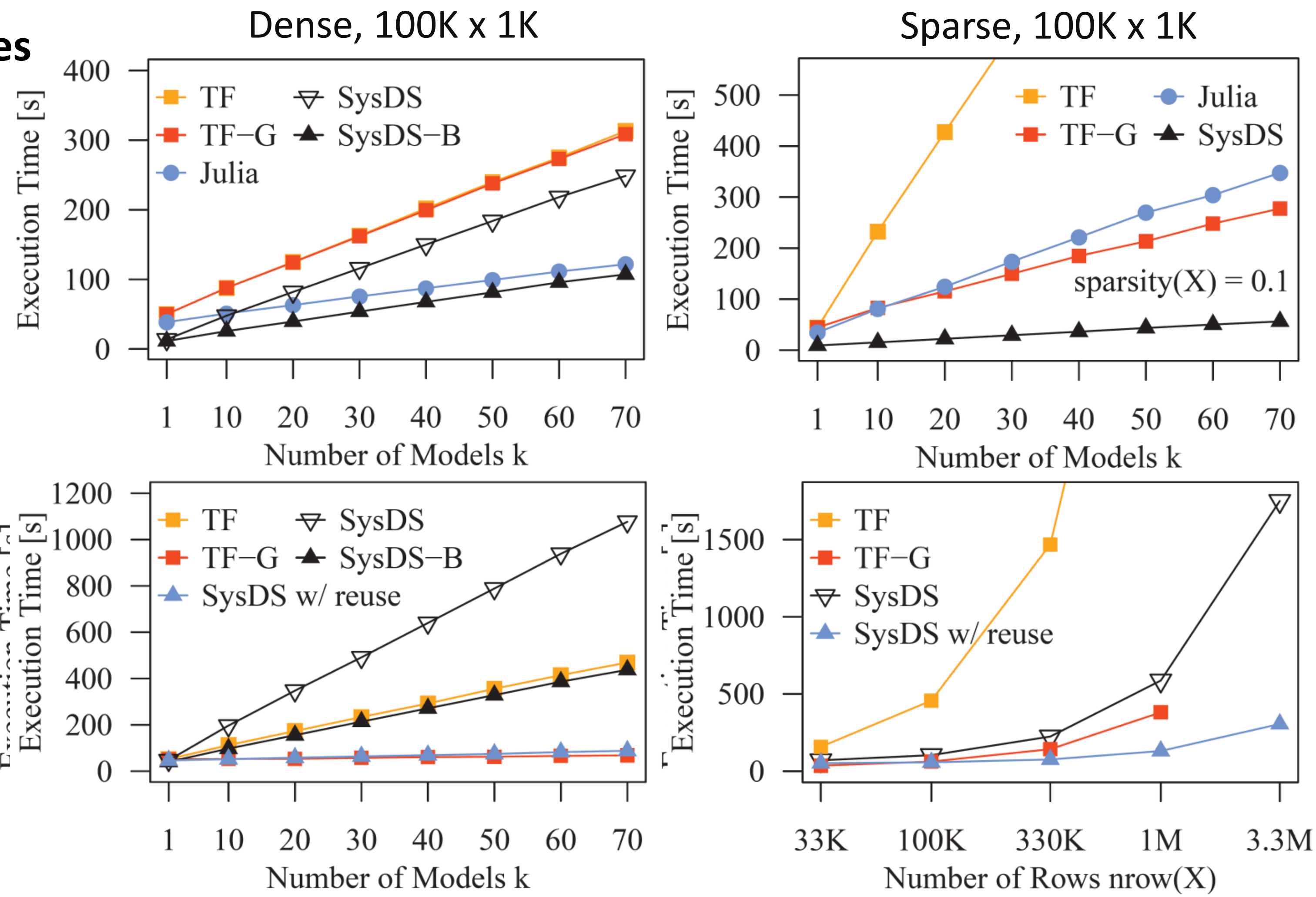



```
m_steplm = function(...) {
  while( continue ) {
    parfor( i in 1:n ) {
      if( !fixed[1,i] ) {
        Xi = cbind(Xg, X[,i])
        B[,i] = lm(Xi, y, ...)
    } }
    # add best to Xg
    # (AIC)
} }
```

$O(n^2(mn^2+n^3)) \rightarrow O(n^2(mn+n^3))$

Experiments (Hyper-Param Opt)

- **Baselines**
  (**TF1.13**)
- **Full Reuse**
  (**TF2.0**)

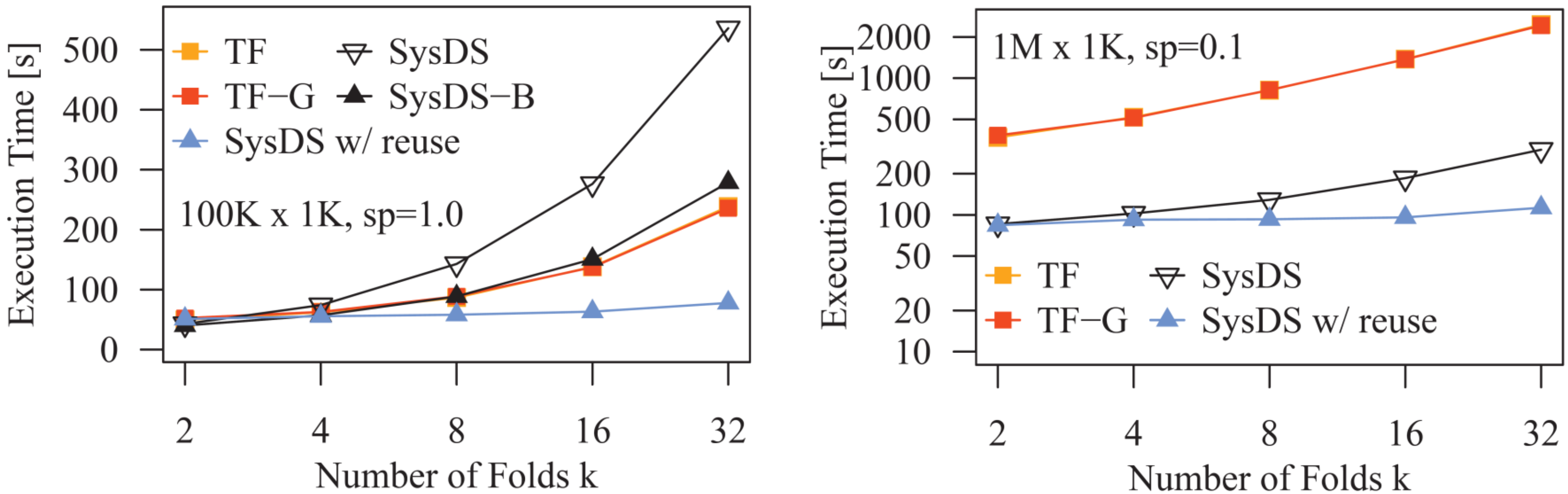


Experiments (Cross Validation)

- **Full Reuse (TF2.0)**

Dense, 100K x 1K

Sparse, 1M x 1K



#1 Competitive baseline performance ML training (dense, sparse)

#2 Large improvements due to **fine-grained redundancy elimination**

Conclusions

- **Summary:** **SystemML is dead**, **long live SystemDS**
  - Vision and system architecture of SystemDS
  - Selected research directions and preliminary results

→ **Apache SystemDS?**

- **#1 Support for data science lifecycle tasks** (data prep, training, debugging), **users w/ different expertise** (ML researcher, data scientist, domain expert)

- **#2 Support for local, distributed, and federated ML,** optimizing compiler and parallelization strategies

- **#3 Underlying data model of heterogeneous tensors** w/ native support for lineage tracing and exploitation, and automatic data reorganization and specialization

- **We're open:** use as baseline or testbed, integrate your work

https://github.com/tugraz-isds/systemds

Thanks

Iulian Antonov, Sebastian Baunsgaard, Mark Dokter, Robert Ginthör, Kevin Innerebner, Florijan Klezin, Stefanie Lindstaedt, Philipp Ortner, Norbert Pfeifer, Arnab Phani, Benjamin Rath, Berthold Reinwald, Svetlana Sagadeeva, Afan Secic, Shafaq Siddiqi, Sebastian Benjamin Wrede

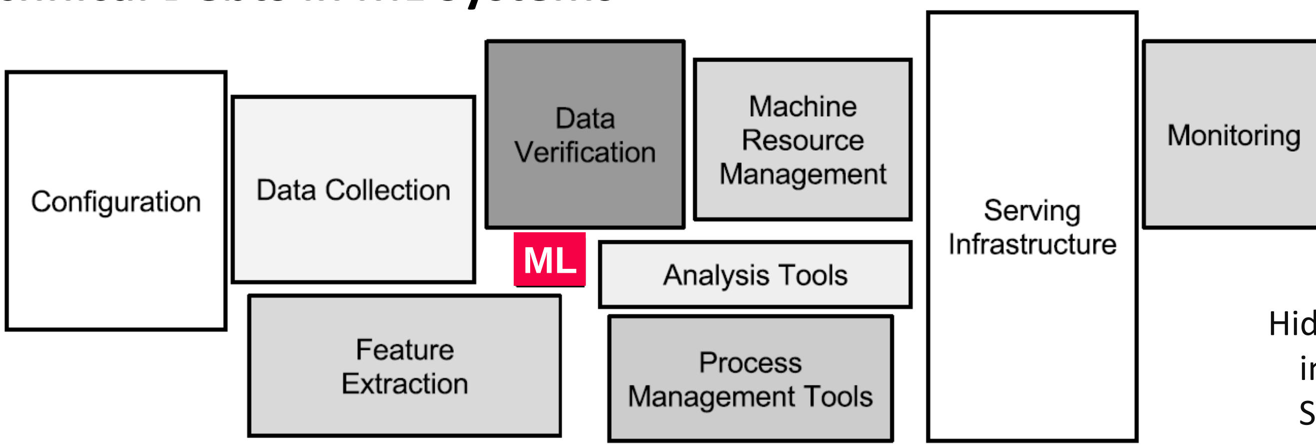
as well as entire **Apache SystemML** team, **CIDR** reviewers, **ExDRa project** team

The 80% Argument

- **Data Sourcing Effort**
  - Data scientists spend **80-90% time** on finding relevant datasets and data integration/cleaning.

[Michael Stonebraker, Ihab F. Ilyas: Data Integration: The Current Status and the Way Forward. **IEEE Data Eng. Bull. 41(2) (2018)**]

- **Technical Debts in ML Systems**



[D. Sculley et al.: Hidden Technical Debt in Machine Learning Systems. **NIPS 2015**]

  - Glue code, pipeline jungles, dead code paths
  - Plain-old-data types, multiple languages, prototypes
  - Abstraction and configuration debts
  - Data testing, reproducibility, process management, and cultural debts

Example: Linear Regression Conjugate Gradient

Note:
#1 Data Independence
#2 Implementation-Agnostic Operations

```
X = read($1); # n x m matrix
y = read($2); # n x 1 vector
maxi = 50; lambda = 0.001;
intercept = $3;
...
r = -(t(X) %*% y);
norm_r2 = sum(r * r); p = -r;
w = matrix(0, ncol(X), 1); i = 0;
while(i<maxi & norm_r2>norm_r2_trgt)
{
   q = (t(X) %*% (X %*% p))+lambda*p;
   alpha = norm_r2 / sum(p * q);
   w = w + alpha * p;
   old_norm_r2 = norm_r2;
   r = r + alpha * q;
   norm_r2 = sum(r * r);
   beta = norm_r2 / old_norm_r2;
   p = -r + beta * p; i = i + 1;
}
write(w, $4, format="text");
```

Read matrices from HDFS/S3

Compute initial gradient

Compute conjugate gradient

Compute step size

Update model and residuals

→ **"Separation of Concerns"**

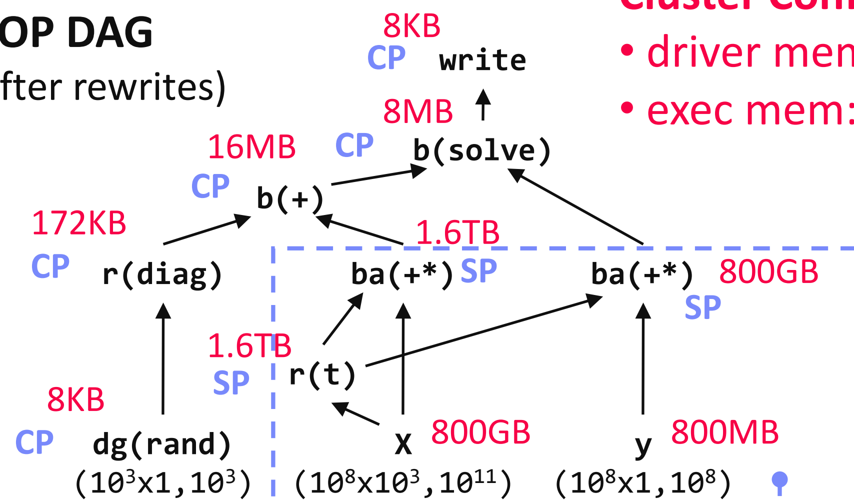
Basic HOP and LOP DAG Compilation

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);
```

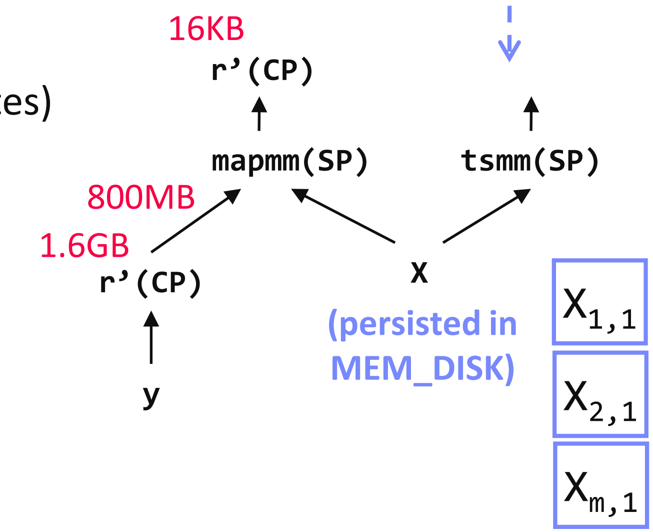
Scenario:
X: 10^8 x 10^3, 10^{11}
y: 10^8 x 1, 10^8

HOP DAG
(after rewrites)



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

LOP DAG
(after rewrites)

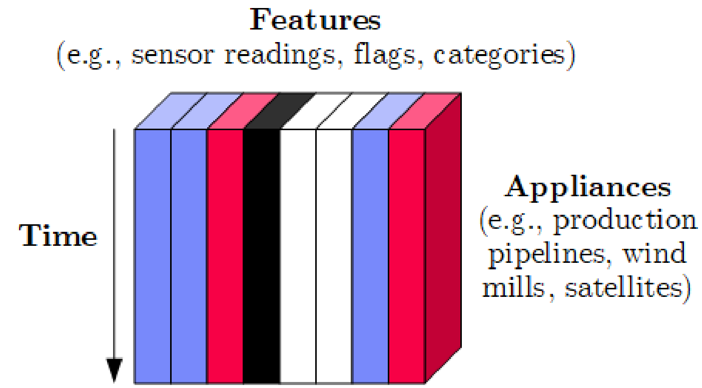


➔ **Hybrid Runtime Plans:**
- **Size propagation / memory estimates**
- **Integrated CP / Spark runtime**
- **Dynamic recompilation during runtime**

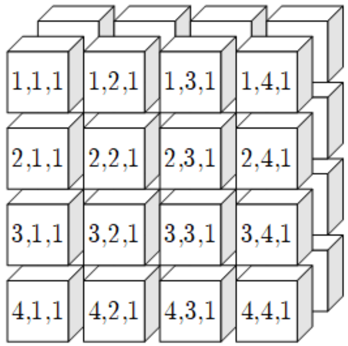
➔ **Distributed Matrices**
- **Fixed-size (squared) matrix blocks**
- **Data-parallel operations**

Data Model: Heterogeneous Tensors

- **Basic Tensor Block**
  - **BasicTensorBlock:** homogeneous tensors (FP32, FP64, INT32, INT64, BOOL, STRING/JSON)
  - **DataTensorBlock:** composed from basic TBs
  - Represents local tensor (CPU/GPU)



- **Distributed Tensor Representation**
  - Collection of **fix-sized tensor blocks**
  - Squared blocking schemes in n-dim space (e.g., 1024^2, 128^3, 32^4, 16^5, 8^6, 8^7)
  - PairRDD<TensorIndex,TensorBlock>



- **Federated Tensor Representation**
  - Collection of meta data handles to `TensorObjects`, each of which might refer to data on a different worker instance (local or distributed)
  - Generalizes to federated tensors of CPU and GPU data objects