Architecture of ML Systems*
01 Introduction and System Landscape

Matthias Boehm

Graz University of Technology, Austria
Computer Science and Biomedical Engineering
Institute of Interactive Systems and Data Science
BMK endowed chair for Data Management

Last update: Aug 25, 2022

About Me

- **Since 09/2022 TU Berlin**, Germany
  - University professor for Big Data Engineering (DAMS)
  - https://github.com/apache/systemds
- **2018-2022 TU Graz**, Austria
  - BMK endowed chair for data management
  - **Data management for data science** (DAMS)
    (ML systems internals, end-to-end data science lifecycle)
- **2012-2018 IBM Research – Almaden**, USA
  - Declarative large-scale machine learning
  - Optimizer and runtime of **Apache SystemML**
- **2007-2011 PhD TU Dresden**, Germany
  - Cost-based optimization of integration flows
  - Systems support for time series forecasting
  - In-memory indexing and query processing

DB group

Agenda

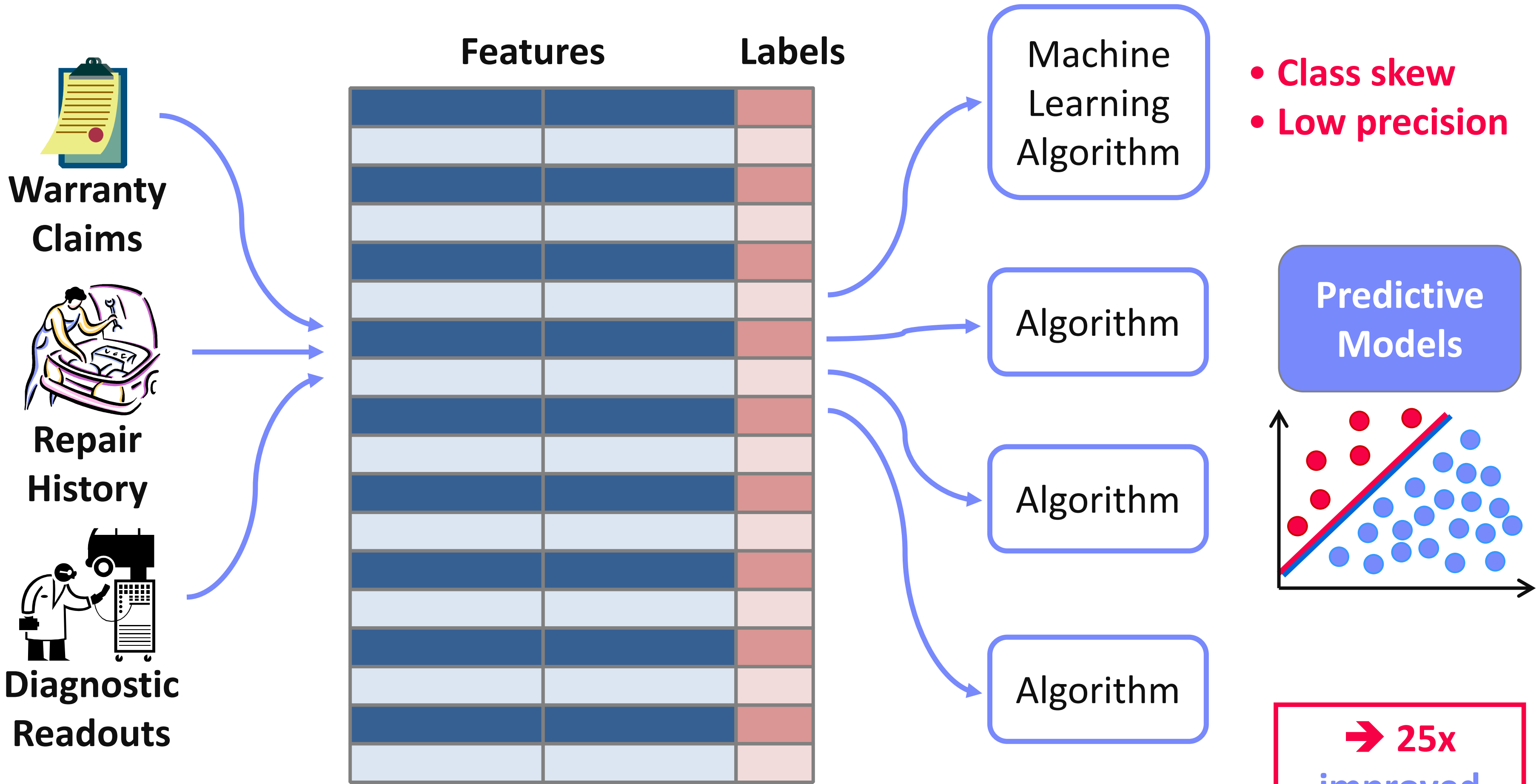
- **Motivation and Goals**
- **Course Organization, Outline, Exercise/Projects**
- **Data Science Lifecycle & ML Systems Stack**
- **Apache SystemDS and DAPHNE**

Motivation and Goals

Example ML Applications (Past/Present)

- **Transportation / Space**
  - **Lemon car detection and reacquisition** (classification, seq. mining)
  - **Airport passenger flows from WiFi data** (time series forecasting)
  - **Data analysis for assisted driving** (various use cases)
  - **Automotive vehicle development** (ML-assisted simulations)
  - Satellite senor analytics (regression and correlation)
  - Earth observation and **local climate zone classification** and monitoring
- **Finance**
  - Water cost index based on various influencing factors (regression)
  - **Insurance claim cost per customer** (model selection, regression)
  - **Financial analysts survey correlation** (bivariate stats w/ new tests)
- **Health Care**
  - **Breast cancer cell grow from histopathology images** (classification)
  - **Glucose trends and warnings** (clustering, classification)
  - Emergency room diagnosis / patient similarity (classification, clustering)
  - Patient survival analysis and prediction (Cox regression, Kaplan-Meier)

A Car Reacquisition Scenario



- Class skew
- Low precision

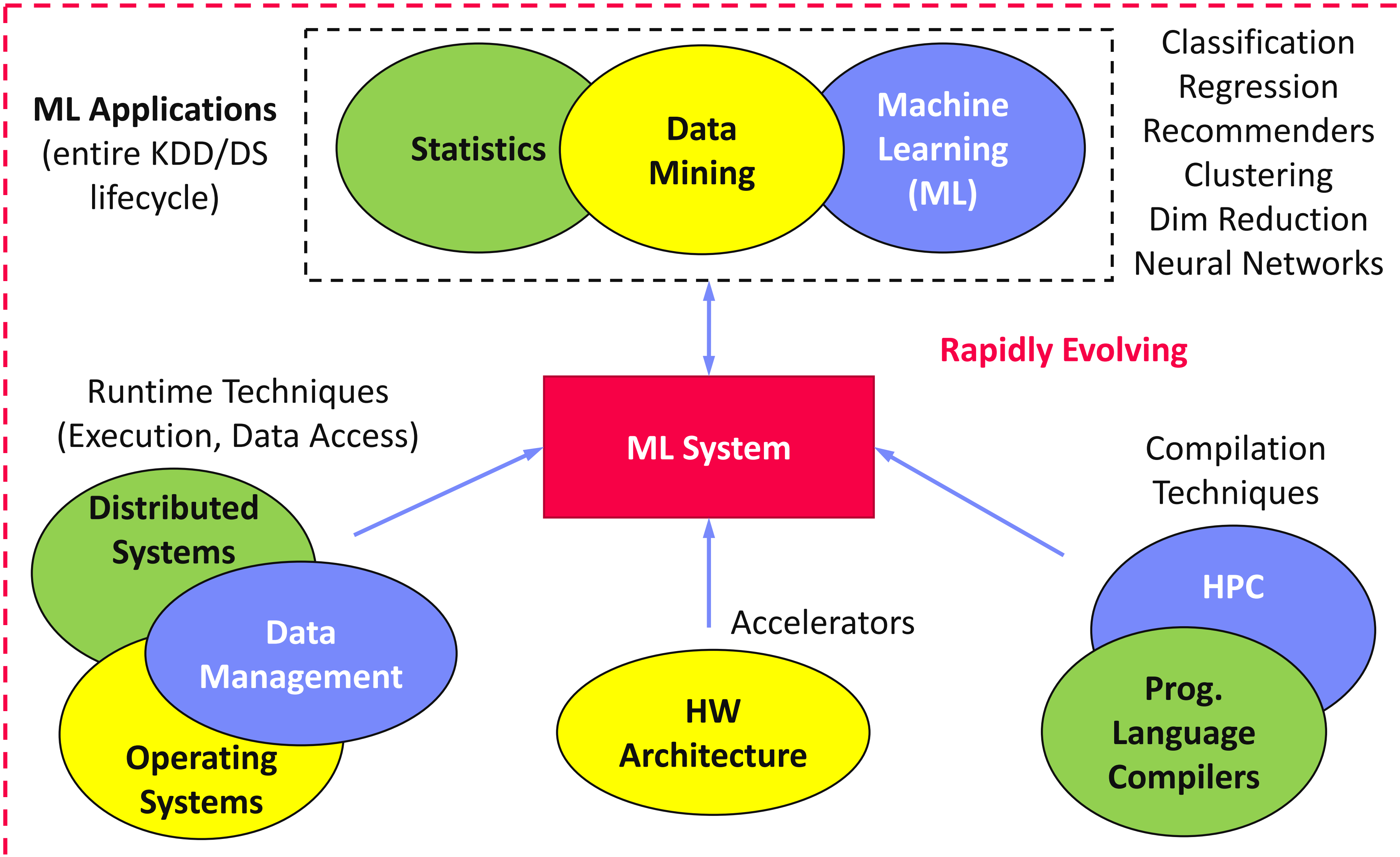
+ custom loss functions
+ hyper-parameter tuning

→ 25x improved precision

Example ML Applications (Past/Present), cont.

- **Production/Manufacturing**
  - Paper and fertilizer production (regression/classification, anomalies)
  - Semiconductor manufacturing, and material degradation modeling
  - Mixed waste sorting and recycling (composition, alignment, quality)
- **Other Domains**
  - Machine data: errors and correlation (bivariate stats, seq. mining)
  - Smart grid: energy demand/RES supply, weather models (forecasting)
  - Elastic flattening via sparse linear algebra (spring-mass system)
- **Information Extraction**
  - NLP contracts → rights/obligations (classification, error analysis)
  - PDF table recognition and extraction, OCR (NMF clustering, custom)
  - Learning explainable linguistic expressions (learned FOL rules, classification)
- **Algorithm Research** (+ various state-of-the art algorithms)
  - User/product recommendations via various forms of NMF
  - Localized, supervised metric learning (dim reduction and classification)
  - Learning word embeddings via orthogonalized skip-gram

What is an ML System?



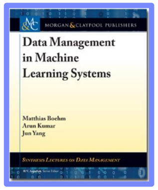
What is an ML System?, cont.

- **ML System**
  - **Narrow focus:** SW system that executes ML applications
  - **Broad focus:** Entire system (HW, compiler/runtime, ML application)
  - → Trade-off **runtime/resources** vs **accuracy**
  - → Early days: no standardizations (except some exchange formats), lots of different languages and system architectures, but many shared concepts

- **Course Objectives**
  - Architecture and internals of modern (large-scale) ML systems
    - **Macroscopic view** of ML pipelines and data science lifecycle
    - **Microscopic view** of ML system internals
  - **#1** Understanding of characteristics → **better evaluation / usage**
  - **#2** Understanding of effective techniques → **build/extend ML systems**

Course Organization, Outline, and Exercise/Projects

Partially based on



[Matthias Boehm, Arun Kumar, Jun Yang: Data Management in Machine Learning Systems. Synthesis Lectures on Data Management, Morgan & Claypool Publishers 2019]

Updates in SS2019, SS2020, SS2021, and SS2022

Basic Course Organization & Logistics

- **Staff**
  - Lecturer: Univ.-Prof. Dr.-Ing. Matthias Boehm, ISDS
  - Assistants: M.Sc. Sebastian Baunsgaard, M.Tech. Arnab Phani

- **Language**
  - Lectures and slides: **English**
  - Communication and examination: **English**/**German**/**Danish**

- **Course Format**
  - Block lectures **August 29 and 30, 8am-5pm** (with informal language)
  - 5 and 4 sessions per day with 15/30min breaks
  - Website: https://mboehm7.github.io/teaching/fs22_amls/index.htm
  - Grading: Pass/fail (with mandatory exercise/programming project)

- **Prerequisites (preferred)**
  - Basic courses Data Management/Databases, and
  - Basic courses on applied ML / Knowledge Discovery and Data Mining

Course Outline

A: ML Lifecycle Systems (August 29)

- **01 Introduction and System Landscape** [Aug 29, 8am]
- **02 Data Preparation, Cleaning, and Augmentation** [Aug 29, 10.15am]
- **03 Model Selection, Debugging/Explainability/Fairness** [Aug 29, 12.45pm]
- **Discussion/Implementation Programming Projects** [Aug 29, 3pm]
- **04 Model Deployment and Serving** [Aug 29, 3.30pm]

B: ML System Internals (August 30)

- **05 Compilation and Optimization Techniques** [Aug 30, 8am]
- **06 Execution and Parallelization Strategies** [Aug 30, 10.15am]
- **07 HW Accelerators and Data Access Methods** [Aug 30, 12.45am]
- **Discussion/Implementation Programming Projects** [Aug 30, 3pm]

Exercise / Projects (due Sep 20)

- **#1 Exercise on ML Pipelines**
  - https://mboehm7.github.io/teaching/fs22_amls/AMLS_2022_Exercise.pdf
  - **Data Prep:** Setup train/test/validation splits; perform data validation, data augmentation, feature engineering
  - **Modeling:** Compare multiple baseline models using an **OSS ML system**
  - **Tuning:** hyper-parameter tuning and cross validation
  - **Parallelization:** parallelize your ML pipeline (at least the tuning part)
  - **Debugging:** Perform model debugging and investigate explainability

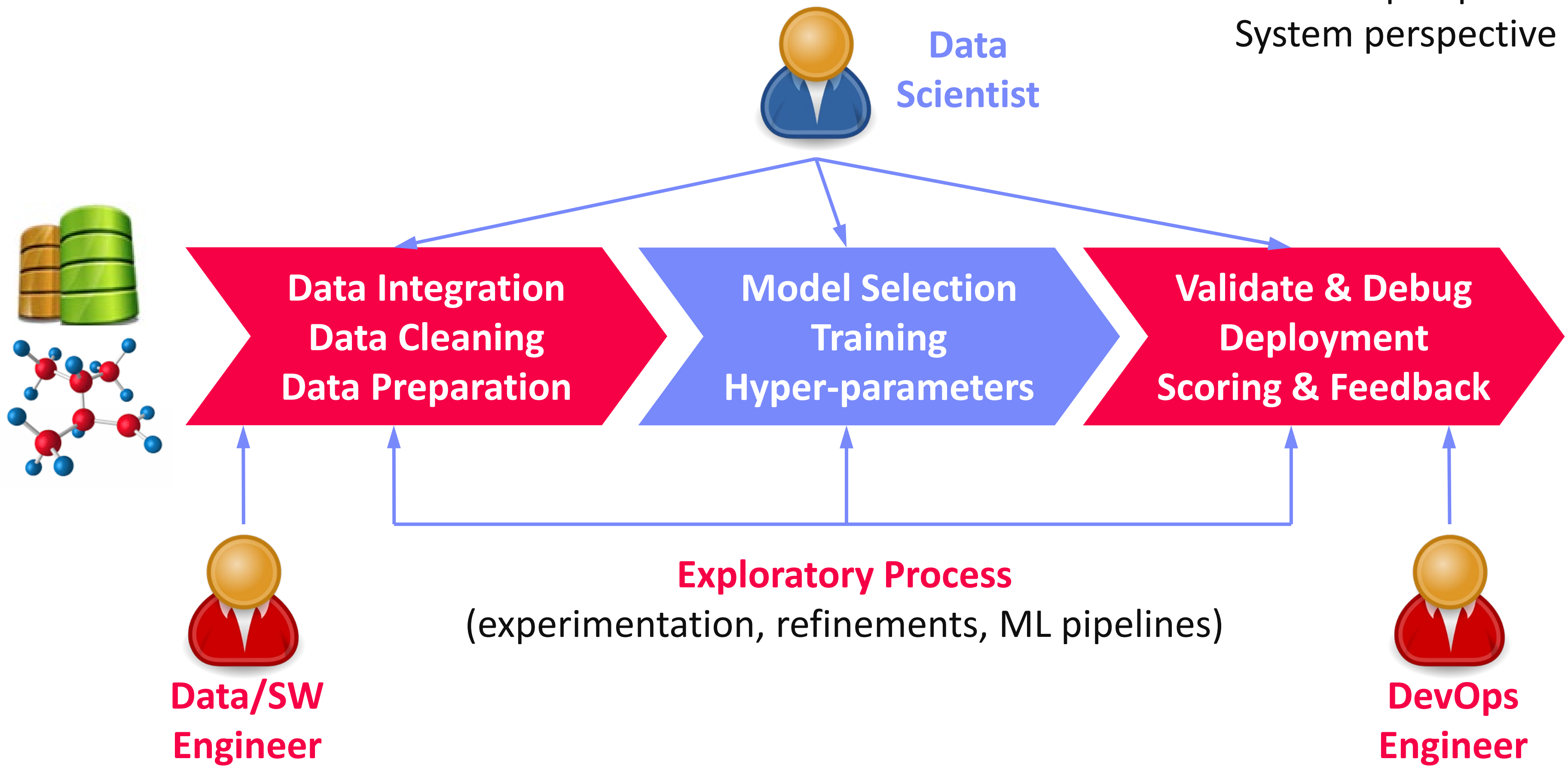
- **#2 Apache SystemDS Projects**
  - https://issues.apache.org/jira/secure/Dashboard.jspa?selectPageId=12335852#Filter-Results/12365413
  - Features across the stack (built-in scripts, APIs, compiler, runtime)

- **#3 DAPHNE Projects**
  - https://mboehm7.github.io/teaching/ss22_amls/AMLS_DAPHNE_projects.pdf
  - OSS since 03/2022; Features at level of runtime, compiler, tools

Data Science Lifecycle and System Landscape

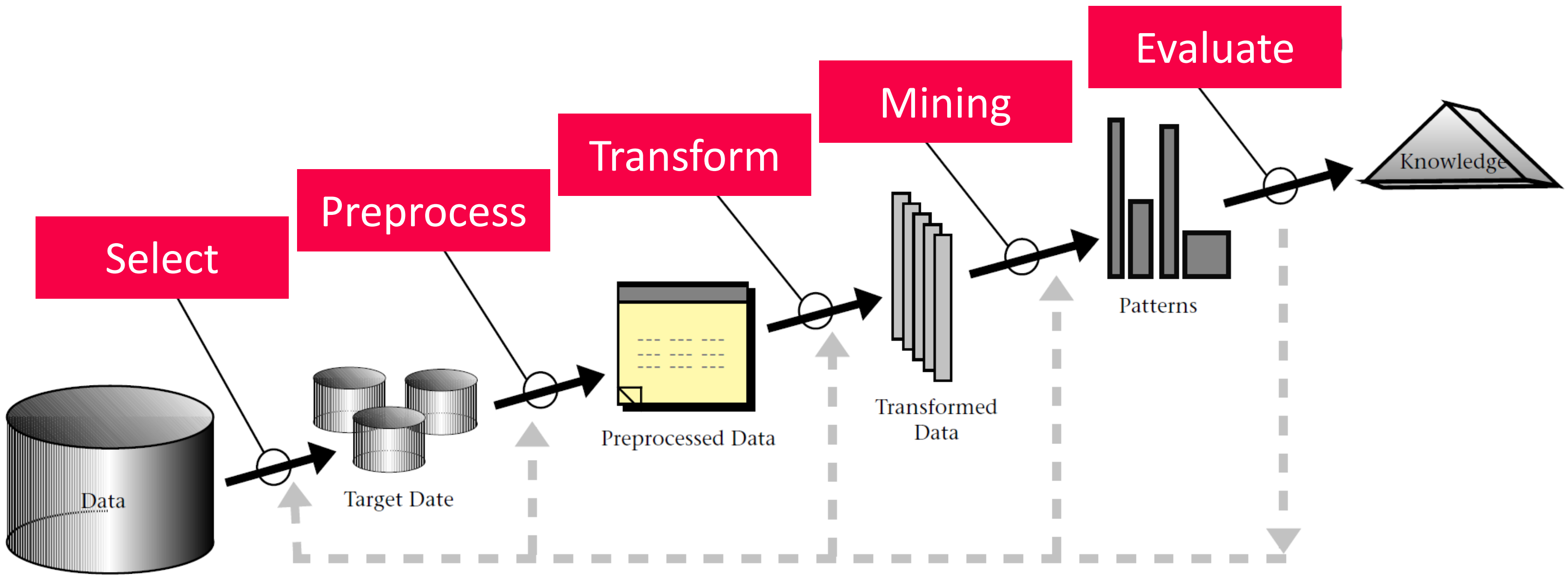
The Data Science Lifecycle

Data-centric View:
Application perspective
Workload perspective
System perspective



The Data Science Lifecycle, cont.

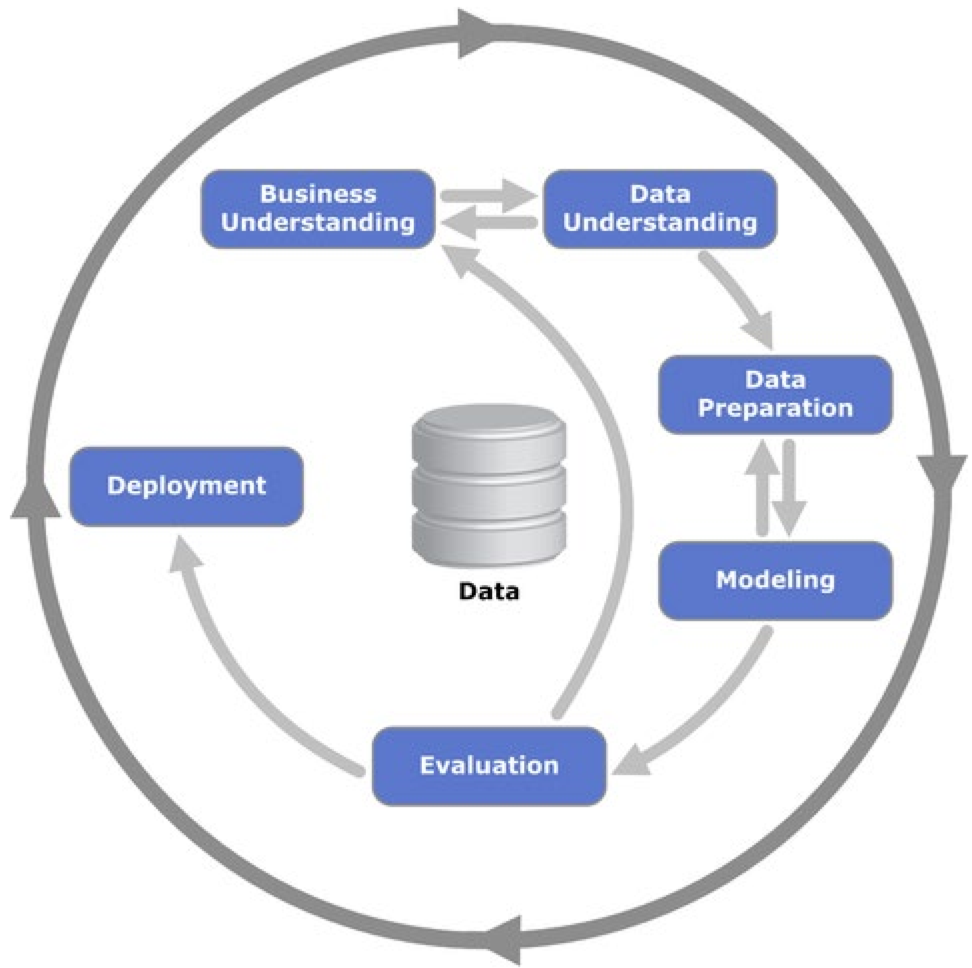
- **Classic KDD Process** (Knowledge Discovery in Databases)
  - Descriptive (association rules, clustering) and predictive



[Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth: From Data Mining to Knowledge Discovery in Databases. **AI Magazine 17(3) (1996)**]

The Data Science Lifecycle, cont.

- **CRISP-DM**
  - **CR**oss-**I**ndustry **S**tandard **P**rocess for **D**ata **M**ining
  - Additional focus on business understanding and deployment



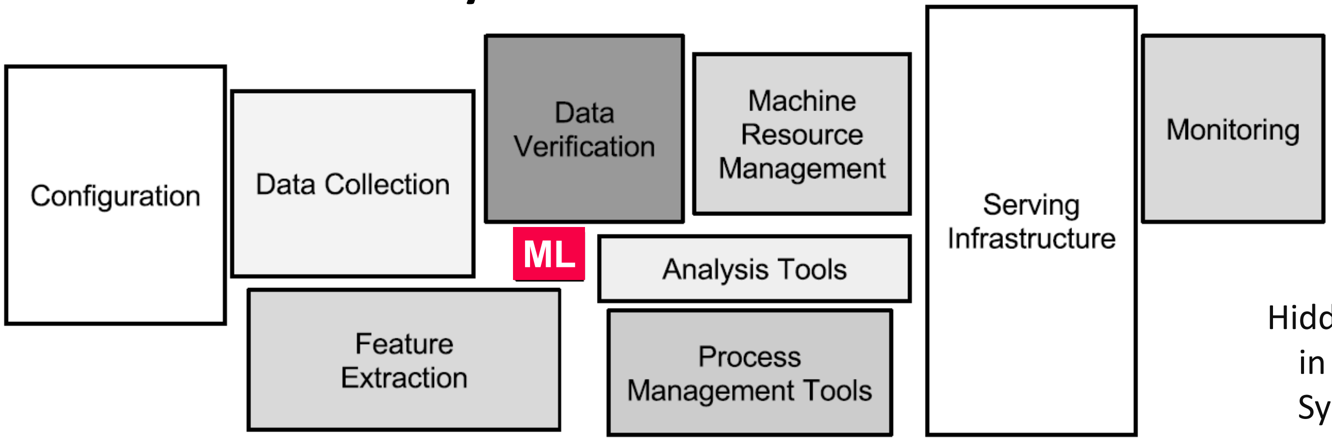
[https://statistik-dresden.de/archives/1128]

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**

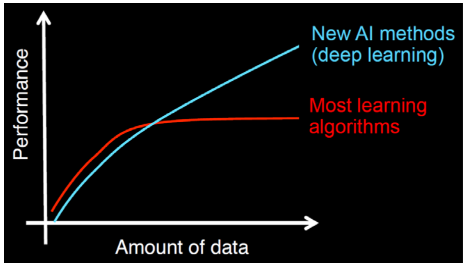


[D. Sculley et al.: Hidden Technical Debt in Machine Learning Systems. **NIPS 2015**]

  - 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

Driving Factors for ML

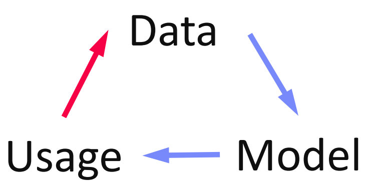
[**Credit:** Andrew Ng'14]



- **Improved Algorithms and Models**
  - Success across data and application domains (e.g., health care, finance, transport, production)
  - More complex models which leverage large data

- **Availability of Large Data Collections**
  - Increasing automation and monitoring → data (simplified by cloud computing & services)
  - Feedback loops, **simulation/data prog./augmentation** → Trend: **self-supervised learning**

Feedback Loop



- **HW & SW Advancements**
  - Higher performance of hardware and infrastructure (cloud)
  - Open-source large-scale computation frameworks, ML systems, and vendor-provides libraries

Stack of ML Systems

Deployment & Scoring

Validation & Debugging

Training

Hyper-parameter Tuning

Model and Feature Selection

Data Programming & Augmentation

Data Preparation
(e.g., one-hot, binning)

Data Integration & Data Cleaning

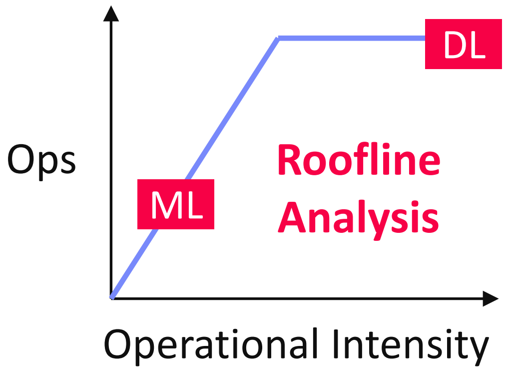
ML Apps & Algorithms	Supervised, unsupervised, RL linear algebra, libs, AutoML
Language Abstractions	Eager interpretation, lazy evaluation, prog. compilation
Fault Tolerance	Approximation, lineage, checkpointing, checksums, ECC
Execution Strategies	Local, distributed, cloud (data, task, parameter server)
Data Representations	Dense & sparse tensor/matrix; compress, partition, cache
HW & Infrastructure	CPUs, NUMA, GPUs, FPGAs, ASICs, RDMA, SSD/NVM

Improve **accuracy** vs. **performance** vs. **resource requirements**

→ **Specialization & Heterogeneity**

Accelerators (GPUs, FPGAs, ASICs)

- **Memory- vs Compute-intensive**
  - **CPU:** dense/sparse, large mem, high mem-bandwidth, moderate compute
  - **GPU:** dense, small mem, slow PCI, very high mem-bandwidth / compute



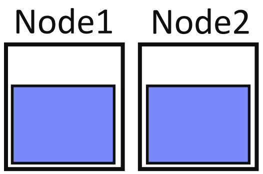
- **Graphics Processing Units (GPUs)**
  - Extensively used for deep learning training and scoring
  - NVIDIA Volta: "tensor cores" for 4x4 mm → 64 2B FMA instruction
- **Field-Programmable Gate Arrays (FPGAs)**
  - Customizable HW accelerators for prefiltering, compression, DL
  - Examples: Microsoft Catapult/Brainwave Neural Processing Units (NPUs)
- **Application-Specific Integrated Circuits (ASIC)**
  - Spectrum of chips: DL accelerators to computer vision
  - Examples: Google TPUs (64K 2B FMA), NVIDIA DLA, Intel NNP, IBM TrueNorth
- **Quantum Computers?**
  - Examples: IBM Q (Qiskit), Google Sycamore (Cirq → TensorFlow Quantum)

Data Representation

- **ML- vs DL-centric Systems**
  - **ML:** dense and sparse matrices or tensors, different sparse formats (CSR, CSC, COO), frames (heterogeneous)
  - **DL:** mostly dense tensors, relies on embeddings for NLP, graphs

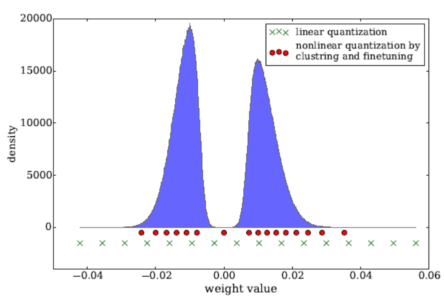
```
vec(Berlin) – vec(Germany)
+ vec(France) ≈ vec(Paris)
```

- **Data-Parallel Operations for ML**
  - Distributed matrices: RDD<`MatrixIndexes,MatrixBlock`>
  - Data properties: **distributed caching**, **partitioning**, **compression**



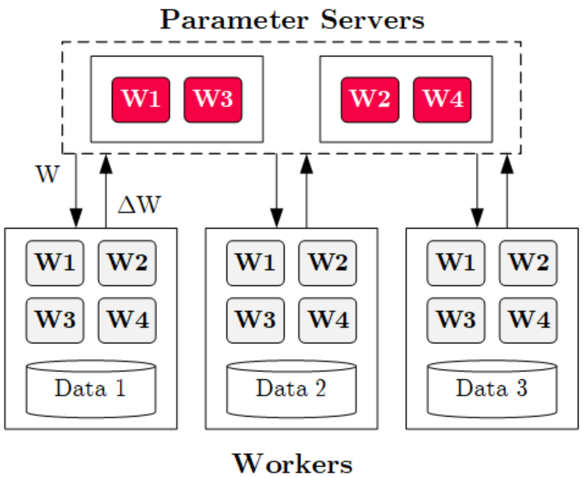
- **Lossy Compression ➔ Acc/Perf-Tradeoff**
  - Sparsification (reduce non-zero values)
  - Quantization (reduce value domain), learned
  - Data types: **bfloat16**, Intel Flexpoint (mantissa, exp)

[**Credit:** Song Han'16]



Execution Strategies

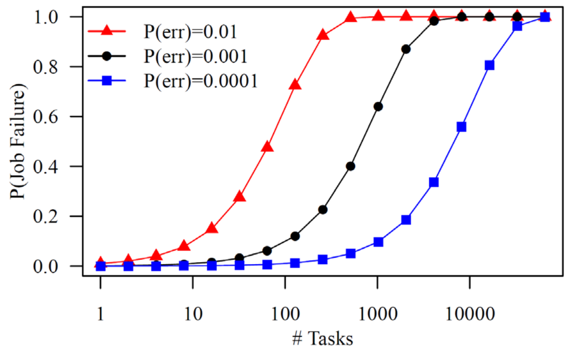
- **Batch Algorithms: Data and Task Parallel**
  - Data-parallel operations
  - Different physical operators
- **Mini-Batch Algorithms: Parameter Server**
  - Data-parallel and model-parallel PS
  - Update strategies (e.g., async, sync, backup)
  - Data partitioning strategies
  - Federated ML (trend 2018)



- **Lots of PS Decisions ➔ Acc/Perf-Tradeoff**
  - Configurations (#workers, batch size/param schedules, update type/freq)
  - Transfer optimizations: lossy compression, sparsification, residual accumulation, gradient clipping, and momentum corrections

Fault Tolerance & Resilience

- **Resilience Problem**
  - Increasing error rates at scale (soft/hard mem/disk/net errors)
  - Robustness for preemption
  - **Need cost-effective resilience**



- **Fault Tolerance in Large-Scale Computation**
  - Block replication (min=1, max=3) in distributed file systems
  - ECC; checksums for blocks, broadcast, shuffle
  - Checkpointing (MapReduce: all task outputs; Spark/DL: on request)
  - Lineage-based recomputation for recovery in Spark

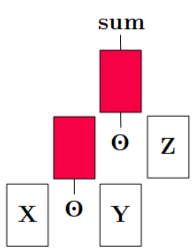
- **ML-specific Schemes** (exploit app characteristics)
  - Estimate contribution from lost partition to avoid strugglers
  - Example: user-defined “compensation” functions

Language Abstractions

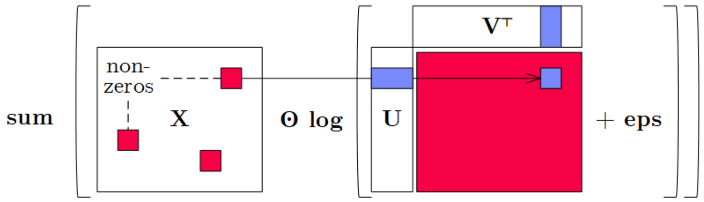
- **Optimization Scope**
  - **#1 Eager Interpretation** (debugging, no opt)
  - **#2 Lazy expression evaluation** (some opt, avoid materialization)
  - **#3 Program compilation** (full opt, difficult)

- **Optimization Objective**
  - Most common: **min time** s.t. memory constraints
  - Multi-objective: **min cost** s.t. time, **min time** s.t. acc, **max acc** s.t. time

- **Trend: Fusion and Code Generation**
  - Custom fused operations
  - Examples: SystemML, Weld, Taco, Julia, TF XLA,TVM, TensorRT



Sparsity-Exploiting Operator



Landscape of ML Systems, cont.

#1 Language Abstraction

Linear Algebra Programs

Computation Graphs

Algorithm Libraries

Operator Libraries

#2 Execution Strategies

Other Language Abstractions
- UDF-based Systems
- Graph-based Systems
- Linear Algebra Systems
- ML Libraries
- DNN Frameworks
- Feature-centric Frameworks

Parameter Server (Model-Parallel)

Task-Parallel Constructs

Data-Parallel Operations

#3 Distribution

Local (single node)

HW accelerators (GPUs, FPGAs, ASICs)

Distributed

#4 Data Types

Collections

Graphs

Matrices

Tensors

Frames

ML Applications

- **ML Algorithms (cost/benefit – time vs acc)**
  - Unsupervised/supervised; batch/mini-batch; first/second-order ML
  - Mini-batch DL: variety of NN architectures and SGD optimizers
- **Specialized Apps: Video Analytics in NoScope (time vs acc)**
  - Difference detectors / specialized models for "short-circuit evaluation"

[**Credit:** Daniel Kang'17]

- **AutoML (time vs acc)**
  - Not algorithms but tasks (e.g., **`doClassify`**`(X, y)` + search space)
  - Examples: MLBase, Auto-WEKA, TuPAQ, Auto-sklearn, Auto-WEKA 2.0
  - AutoML services at Microsoft Azure, Amazon AWS, Google Cloud
- **Data Programming and Augmentation (acc?)**
  - Generate **noisy labels for pre-training**
  - Exploit expert rules, simulation models, rotations/shifting, and labeling IDEs (Software 2.0)

[**Credit:** Jonathan Tremblay'18]

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

Background and System Architecture

https://github.com/apache/systemds

Landscape of ML Systems

- **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"

[NIPS 2015]
[DEBull 2018]

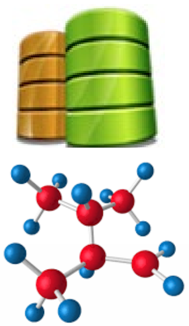
- **Data Preparation Problem**
  - **80% Argument:** 80-90% time for finding, integrating, cleaning data
  - Diversity of tools ➔ boundary crossing, lack of optimization

The Data Science Lifecycle

Data-centric View:
Application perspective
Workload perspective
System perspective

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

Data Scientist



Data Integration
Data Cleaning
Data Preparation

Model Selection
Training
Hyper-parameters

Validate & Debug
Deployment
Scoring & Feedback

Data/SW Engineer

Exploratory Process
(experimentation, refinements, ML pipelines)

DevOps Engineer

Key observation: SotA data integration/cleaning based on ML

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"**

Apache SystemML/SystemDS

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

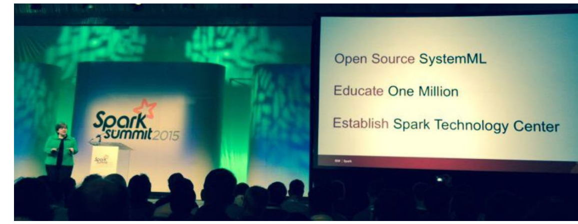
DML Scripts

Language

Compiler

Runtime

Write Once, Run Anywhere



Apache SystemML™

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

[SIGMOD'15,'17,'19,'21abc,'23]
[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)

In-Progress:

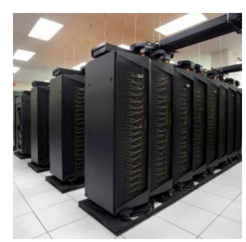
GPU

since 2014/16

Java

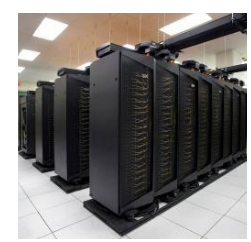
since 2012

hadoop



since 2010/11

Apache Spark



since 2015

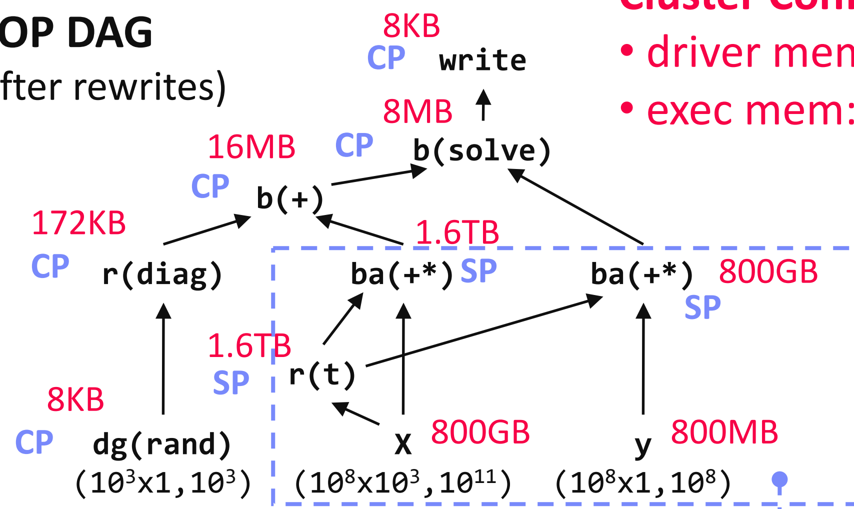
Basic HOP and LOP DAG Compilation

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);
```

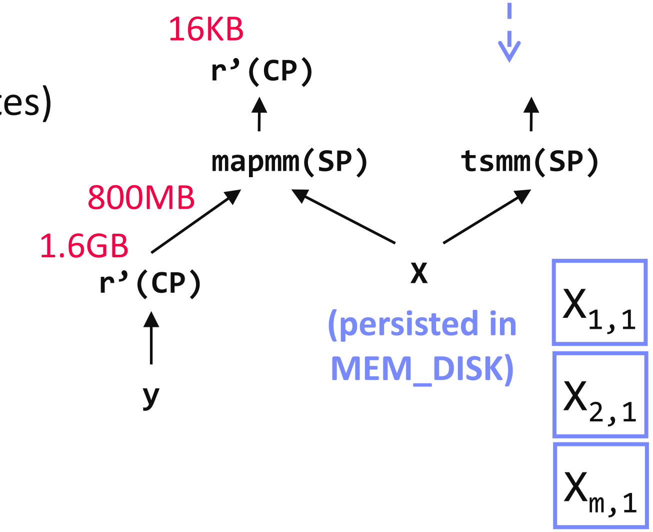
Scenario:
X: $10^8 \times 10^3$, 10^{11}
y: $10^8 \times 1$, 10^8

HOP DAG
(after rewrites)



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

LOP DAG
(after rewrites)



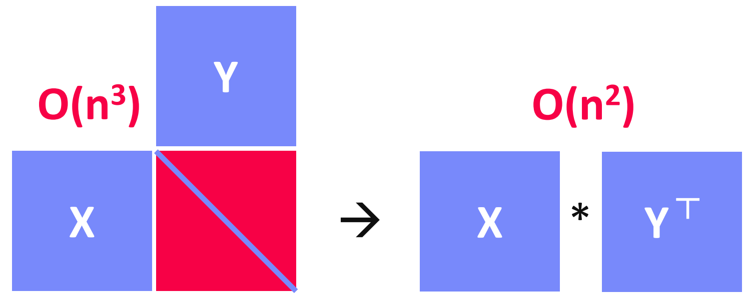
➔ **Hybrid Runtime Plans:**
- **Size propagation / memory estimates**
- **Integrated CP / Spark runtime**
- **Dynamic recompilation during runtime**

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

Static and Dynamic Rewrites

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

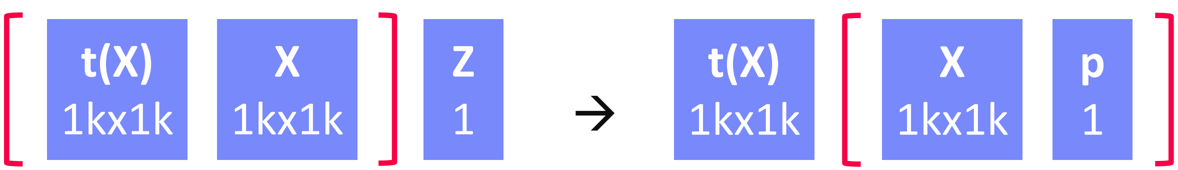
`trace(X%*%Y) → sum(X*t(Y))`



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

- **Example Dynamic Rewrites** (size-dep.)
  - 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

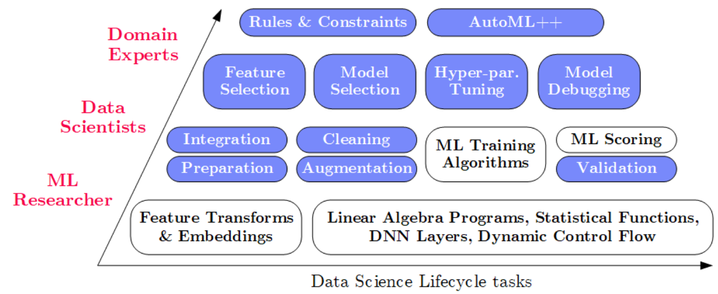
Size propagation and sparsity estimation

Apache SystemDS Design

Apache SystemML (since 2010)
→ SystemDS (09/2018)
→ Apache SystemDS (07/2020)

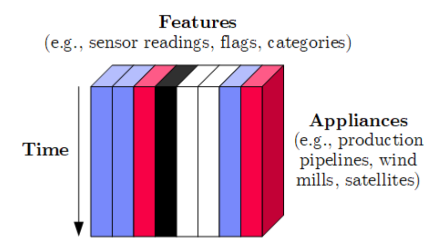
- **Objectives**
  - Effective and efficient **data preparation, ML, and model debugging at scale**
  - High-level abstractions for different lifecycle tasks and users

- **#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

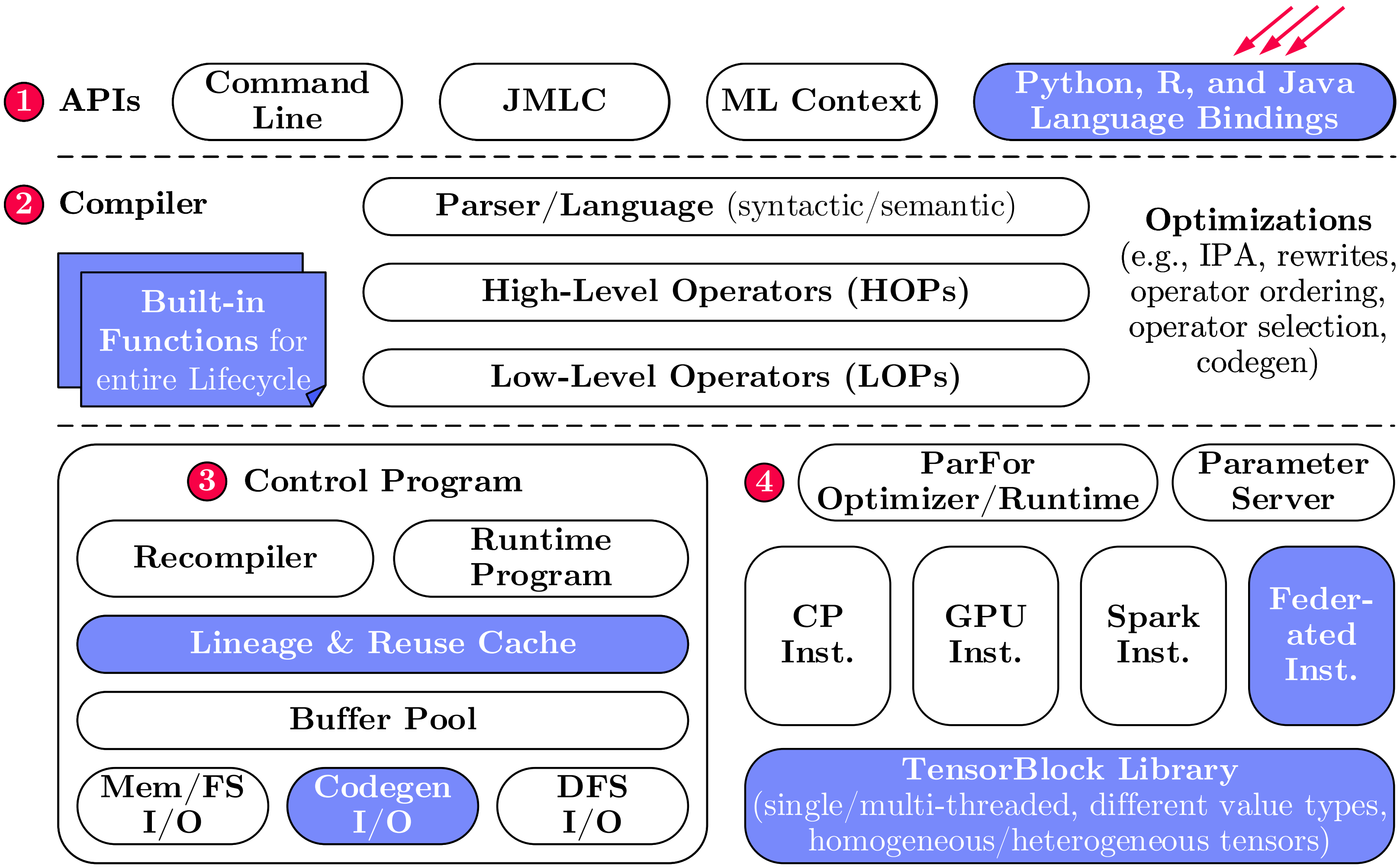
```
m_lmCG = function(...) {
  while( i<maxi&nr2>tgt ) {
    q = (t(X) %*% (X %*% p))
      + lambda * p
    beta = ... }
}
```

```
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

Apache SystemDS Architecture

> **83,400** tests
> **8,500** DSL tests



[M. Boehm, I. Antonov, S. Baunsgaard, M. Dokter, R. Ginthör, K. Innerebner, F. Klezin, S. N. Lindstaedt, A. Phani, B. Rath, B. Reinwald, S. Siddiqui, S. Benjamin Wrede: SystemDS: A Declarative Machine Learning System for the End-to-End Data Science Lifecycle. **CIDR 2020**]

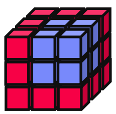
Data Cleaning Pipelines

[under submission]

[WIP] **WashHouse**:
Data Cleaning Benchmark

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

Data Samples

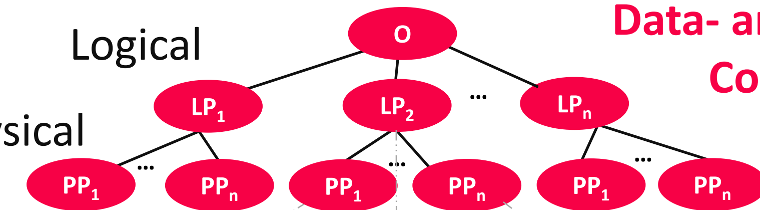


Target App

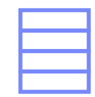
Rules/Objectives

Logical

Physical



Data- and Task-parallel Computation

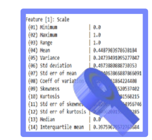


Top-k Pipelines

Outlier Detection → MVI → Deduplication → Resolve Mislabels

P_1: gmm → imputeFD → mergeDup → delML

P_n: outlierBySd → mice → delDup → voting



Debugging

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)**

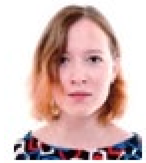
A	B	C	D
0.77	0.80	1	1
0.96	0.12	1	1
0.66	0.09	null	1
0.23	0.04	17	1
0.91	0.02	17	null
0.21	0.38	17	1
0.31	null	17	1
0.75	0.21	20	1
null	null	20	1
0.19	0.61	20	1
0.64	0.31	20	1

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**

SliceLine for Model Debugging



[SIGMOD'21c]

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

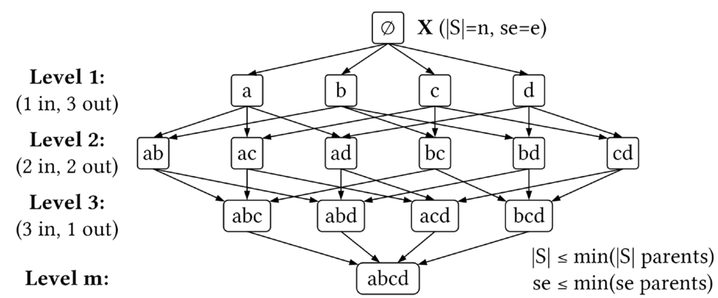
- **Problem Formulation**
  - Intuitive slice scoring function
  - Exact top-k slice finding
  - $|S| \geq \sigma \wedge sc(S) > 0$
  - $\alpha \in (0,1]$

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

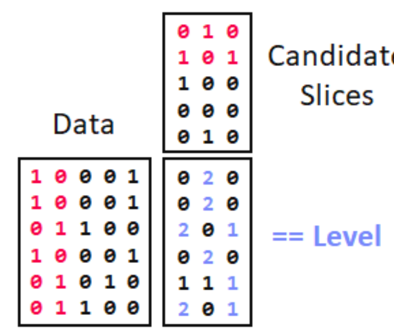
$$= \alpha\left(\frac{|X|}{|S|} \cdot \frac{\sum_{i=1}^{|S|} es_i}{\sum_{i=1}^{|X|} e_i} - 1\right) - (1-\alpha)\left(\frac{|X|}{|S|} - 1\right)$$

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)
  - Local and distributed task/data-parallel execution



Multi-Level Lineage Tracing & Reuse


[SIGMOD'21a]

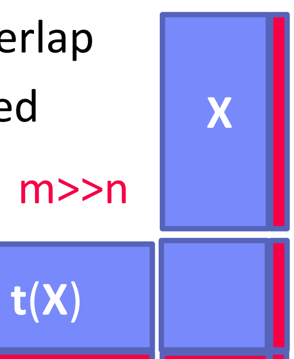
- **Lineage as Key Enabling Technique**
  - Trace lineage of operations (incl. non-determinism), dedup for loops/functions
  - Model versioning, data reuse, incremental maintenance, autodiff, debugging

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

```
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) ...}
```

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

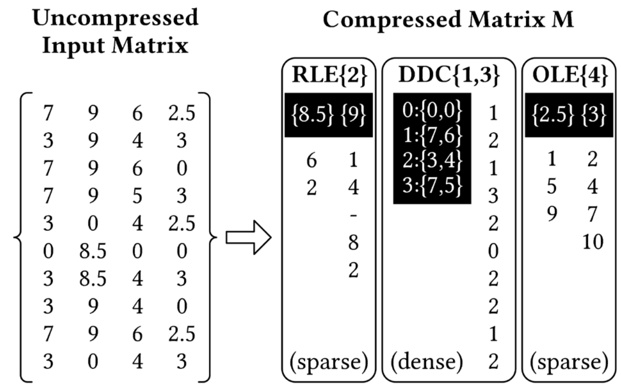



```
m_steplm = function(...) {
  while( continue ) {
    parfor( i in 1:n ) {
      if( !fixed[1,i] ) {
        Xi = cbind(Xg, X[,i])
        B[,i] = lm(Xi, y, ...)
    } }
    # add best to Xg
    # (AIC)
} }
```

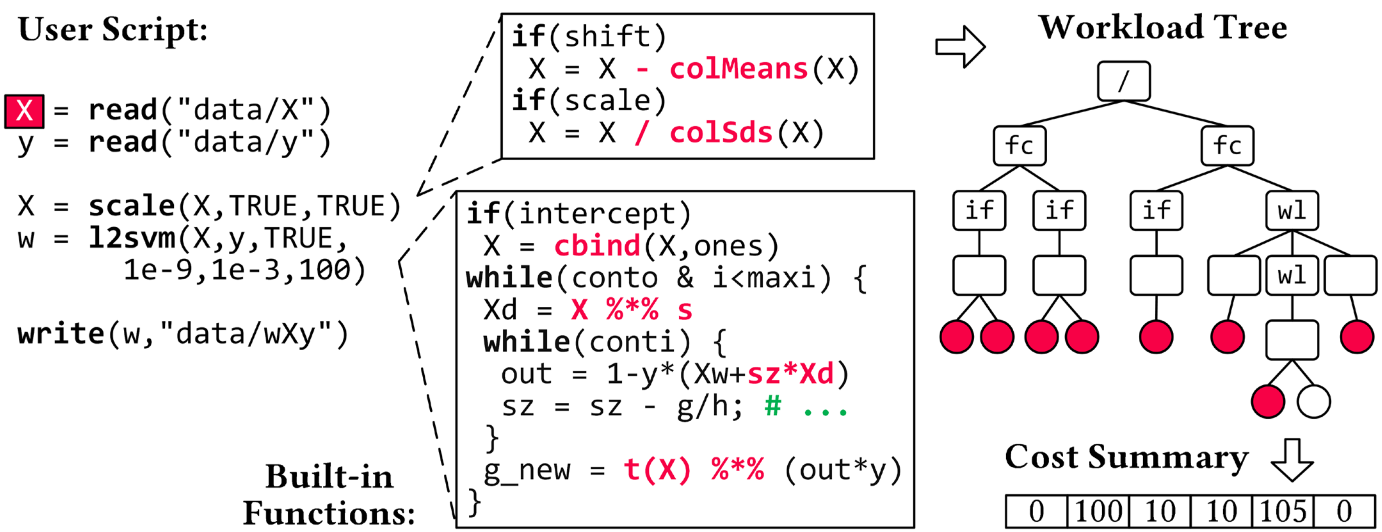
Compressed Linear Algebra Extended

[SIGMOD 2023]

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

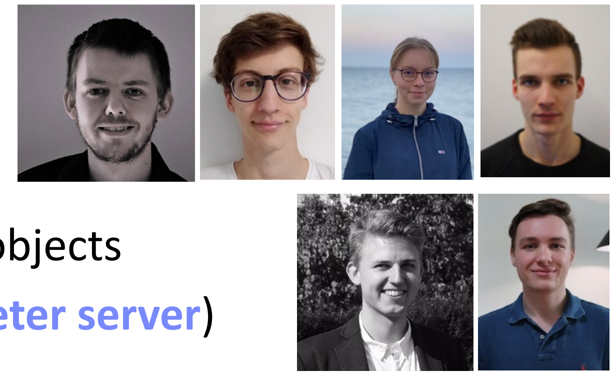


- **Workload-aware Compression**
  - Workload summary → compression
  - Compression → execution planning



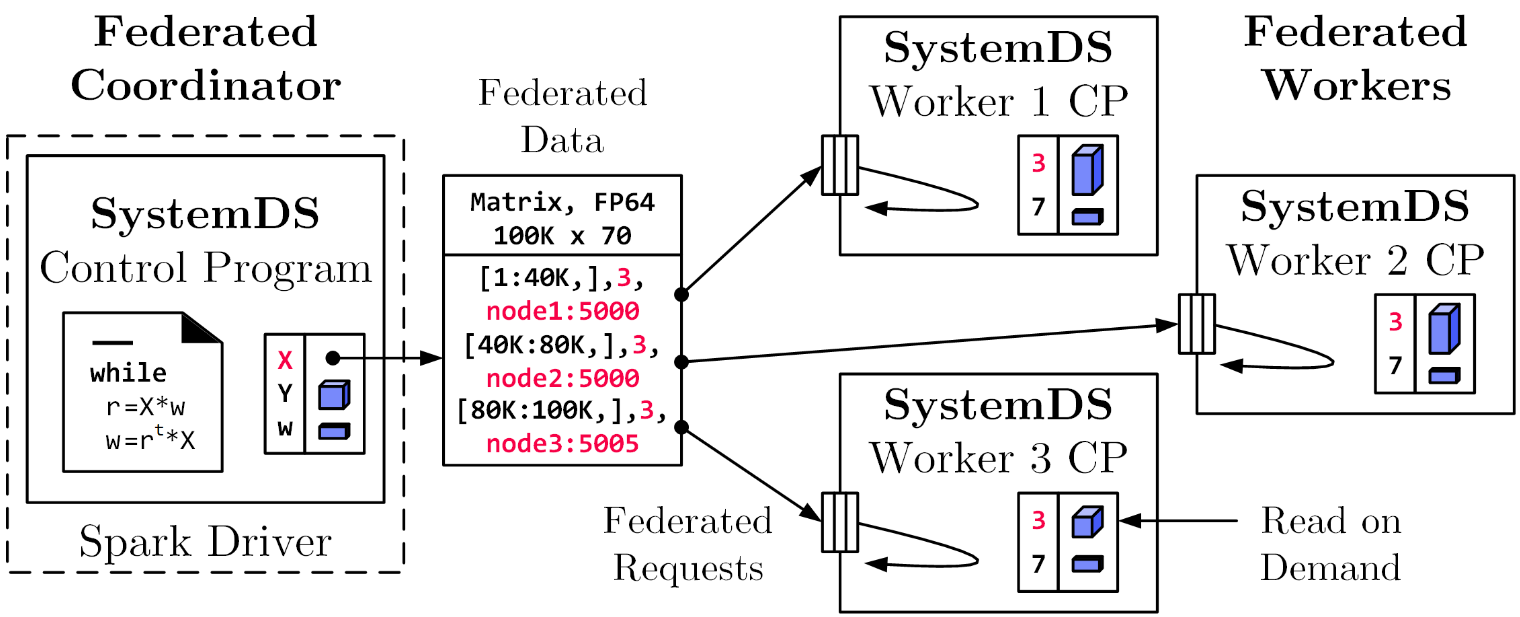
Federated Learning

[SIGMOD 2021b, CIKM 2022]



- **Federated Backend**
  - **Federated data** (matrices/frames) as meta data objects
  - **Federated linear algebra**, (and **federated parameter server**)

```
X = federated(addresses=list(node1, node2, node3),
  ranges=list(list(0,0), list(40K,70), ..., list(80K,0), list(100K,70)));
```



- **Federated Requests:** `READ, PUT, GET, EXEC_INST, EXEC_UDF, CLEAR`

Integrated **D**ata **A**nalysis **P**ipelines
for Large-scale Data Management,
HPC, and Machi**ne** Learning;
DAPHNE daughter of river god Peneus
(fountains, streams), chased by Apollo

[Louvre, Paris]

The DAPHNE project is funded by the European Union's Horizon 2020 research and innovation program under grant agreement number 957407 for the time period from Dec/2020 through Nov/2024.

DAPHNE

An Open and Extensible System Infrastructure for Integrated Data Analysis Pipelines

https://daphne-eu.eu/

Motivation

➔ DAPHNE Overall Objective: Open and extensible system infrastructure

- **Integrated Data Analysis Pipelines**
  - Open data formats, query processing
  - Data preprocessing and cleaning
  - ML model training and scoring
  - HPC, custom codes, and simulations

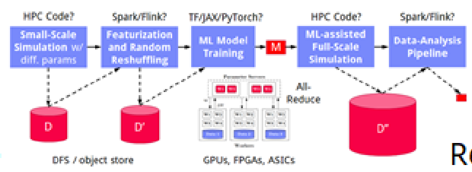
Deployment Challenges

Dev Teams

Programming Models

Different Systems/ Libraries

Cluster **Under-utilization**

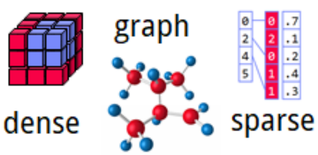


Data/File Exchange

Resource Managers

- **Hardware Challenges**
  - DM+ML+HPC share compilation and runtime techniques / converging cluster hardware
  - **End of Dennard scaling:** $P = \alpha\, CFV^2$ (power density 1)
  - **End of Moore's law**
  - **Amdahl's law:** $sp = 1/s$
  - ➔ **Increasing Specialization**

#1 Data Representations



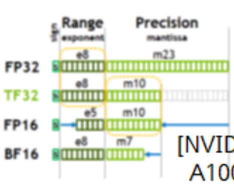
Sparsity Exploitation from Algorithms to HW

#2 Data Placement

Local vs **distributed**

#3 Data (Value) Types

FP32, FP64, INT8, INT32, INT64, UINT8, BF16, TF32, FlexPoint



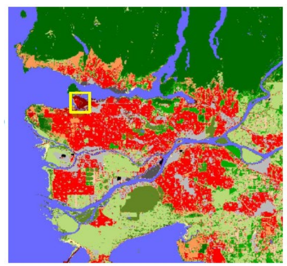
[NVIDIA A100]

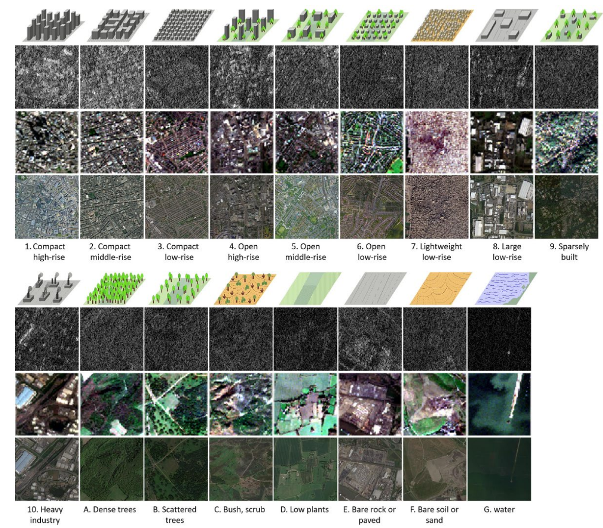
DAPHNE Use Cases

[Xiao Xiang Zhu et al: So2Sat LCZ42: A Benchmark Dataset for the Classification of Global Local Climate Zones. **GRSM 2020**]

[So2Sat LC42 Dataset https://mediatum.ub.tum.de/1454690]

- **DLR Earth Observation**
  - **ESA Sentinel-1/2** datasets → 4PB/year
  - Training of local climate zone classifiers on **So2Sat LCZ42** (15 experts, 400K instances, 10 labels each, 85% confidence, ~55GB H5)
  - **ML pipeline:** preprocessing, ResNet18, climate models



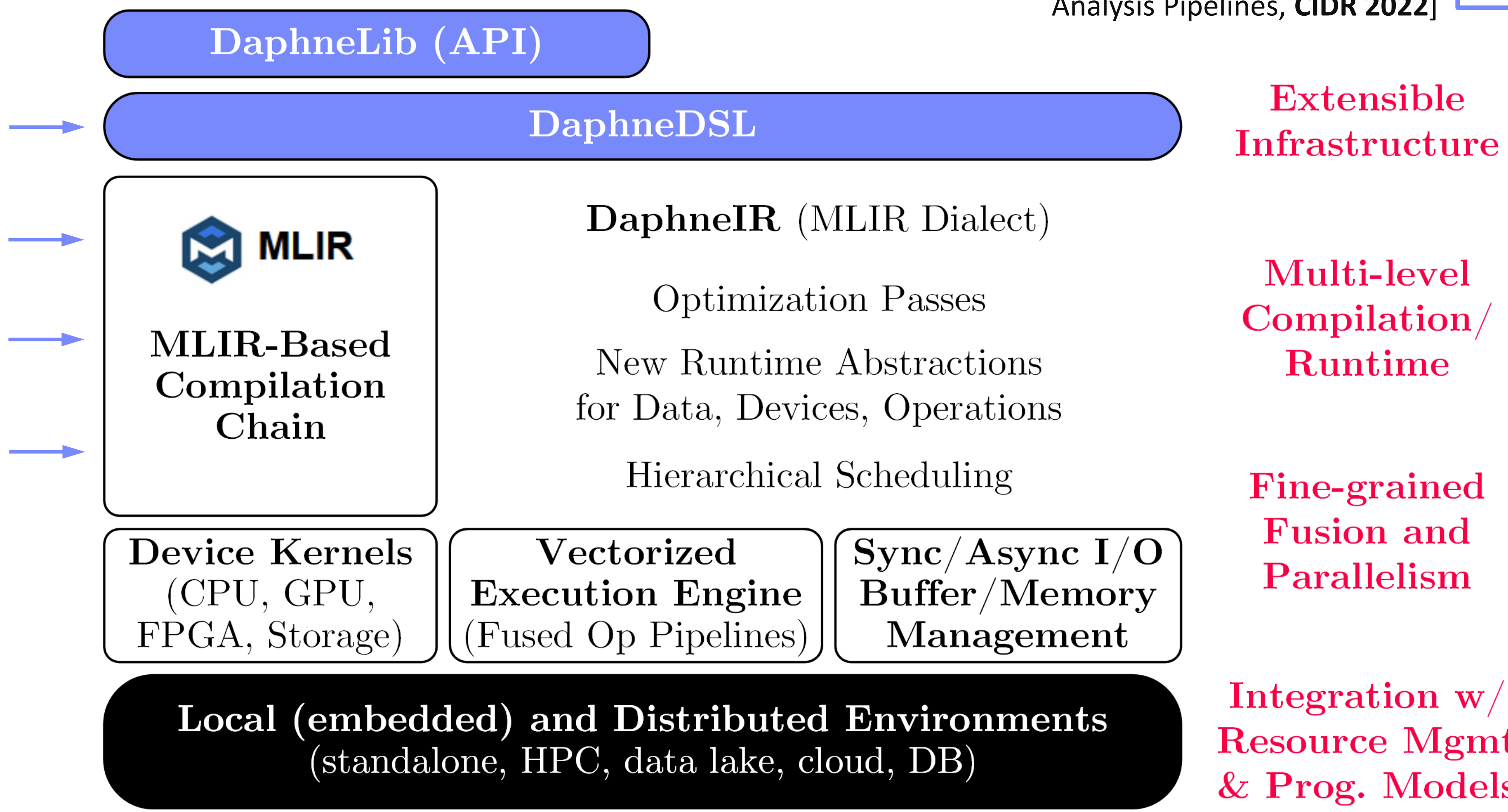


- **IFAT Semiconductor Ion Beam Tuning**
- **KAI Semiconductor Material Degradation**
- **AVL Vehicle Dev Process** (ejector geometries, KPIs)

- **ML-assisted simulations, data cleaning, augmentation**

DAPHNE System Architecture

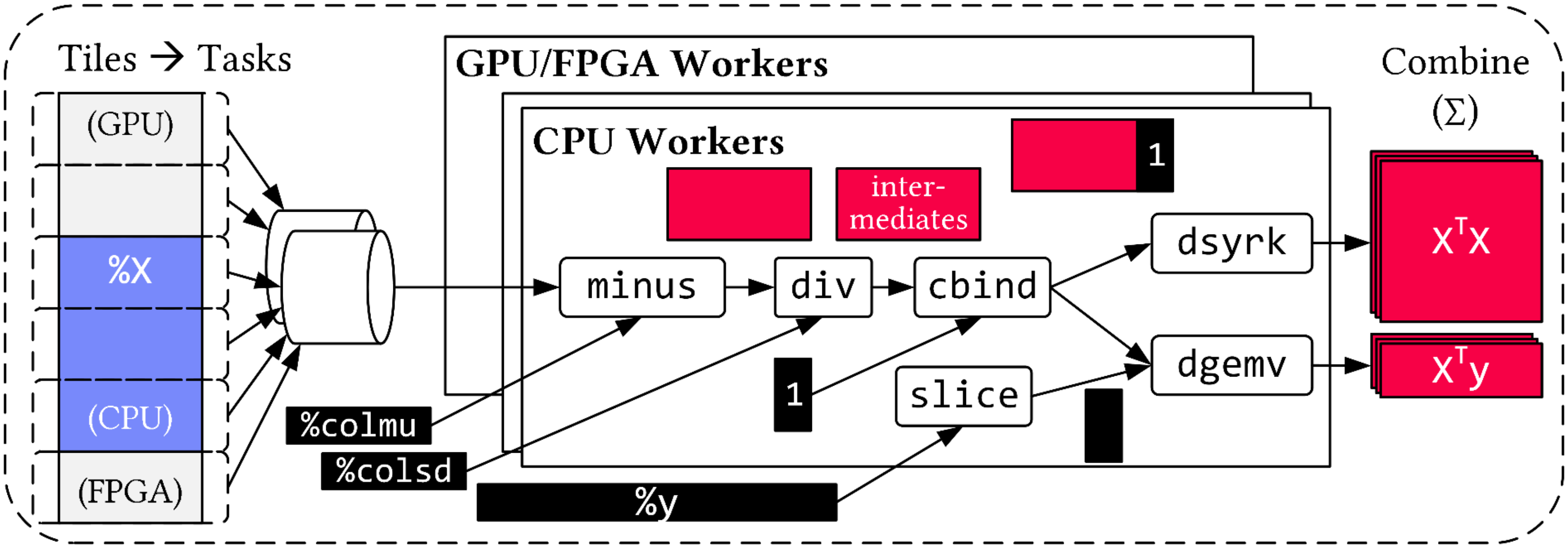
[Patrick Damme et al.: DAPHNE: An Open and Extensible System Infrastructure for Integrated Data Analysis Pipelines, **CIDR 2022**]



Vectorized (Tiled) Execution

AH
AK

```
(%9, %10) = fusedPipeline1(%X, %y, %colmu, %colsd) {
```



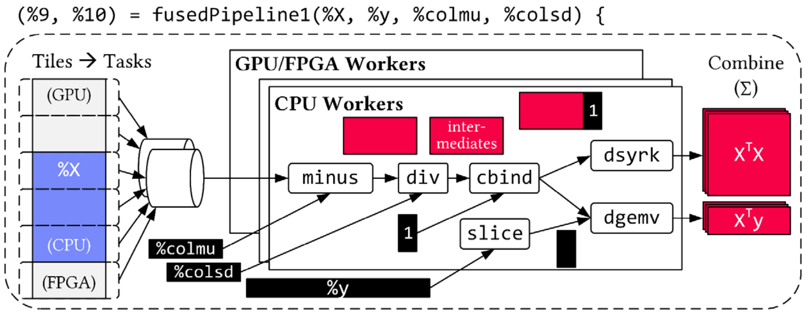
Default Parallelization
Frame & Matrix Ops

Locality-aware,
Multi-device Scheduling

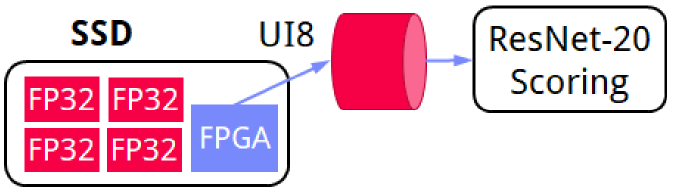
Fused Operator Pipelines
on Tiles/Scalars + Codegen

Vectorized (Tiled) Execution, cont.

- **#1 Zero-copy Input Slicing**
  - Create view on sliced input (no-op)
  - All kernels work on views



- **#2 Sparse Intermediates**
  - Reuse dense/sparse kernels
  - Sparse pipeline intermediates for free
- **#3 Fine-grained Control**
  - Task sizes (dequeue, data access) vs data binding (cache-conscious ops)
  - Scheduling for load balance (e.g., sparse operations)
- **#4 Computational Storage**
  - Task queues connect eBPF programs, async I/O into buffers, and op pipelines



Summary and Q&A

Thanks

- **Motivation and Goals**
- **Course Organization, Outline, Exercise/Projects**
- **Data Science Lifecycle & ML Systems Stack**
- **Apache SystemDS and DAPHNE**

- **Recommended Reading** (a critical perspective on a broad sense of ML systems)
  - [M. Jordan: SysML: Perspectives and Challenges. Keynote at **SysML 2018**]
  - ***"ML […] is far from being a solid engineering discipline that can yield robust, scalable solutions to modern data-analytic problems"***
  - https://www.youtube.com/watch?v=4inIBmY8dQI