Architecture of ML Systems 02 Languages, Architectures, and System Landscape

Matthias Boehm

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

SCIENCE
PASSION
TECHNOLOGY

Last update: Mar 22, 2019

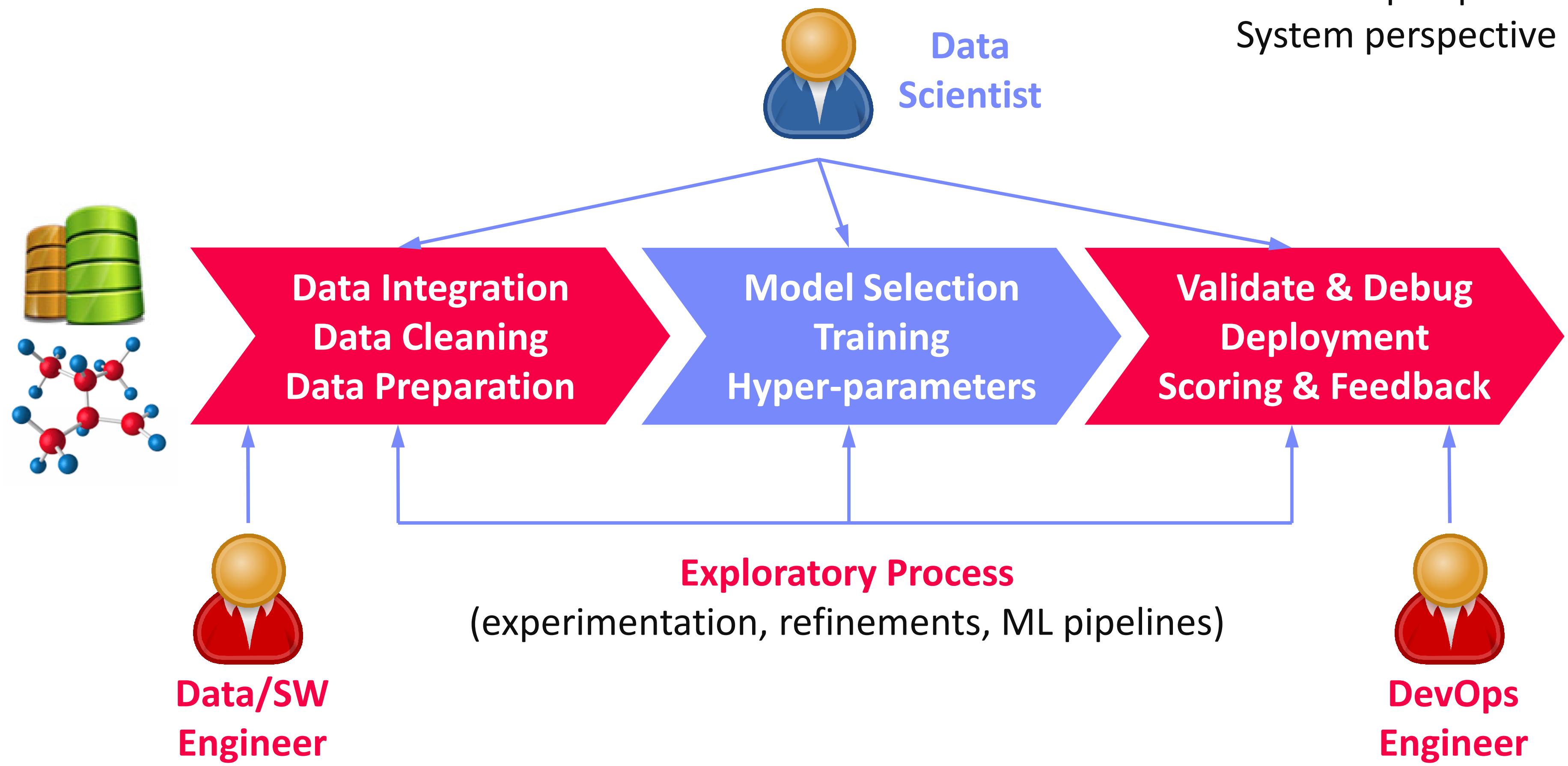
Agenda

- **Data Science Lifecycle**
- **ML Systems Stack**
- **System Architectures**
- **Discussion Programming Projects**

Data Science Lifecycle

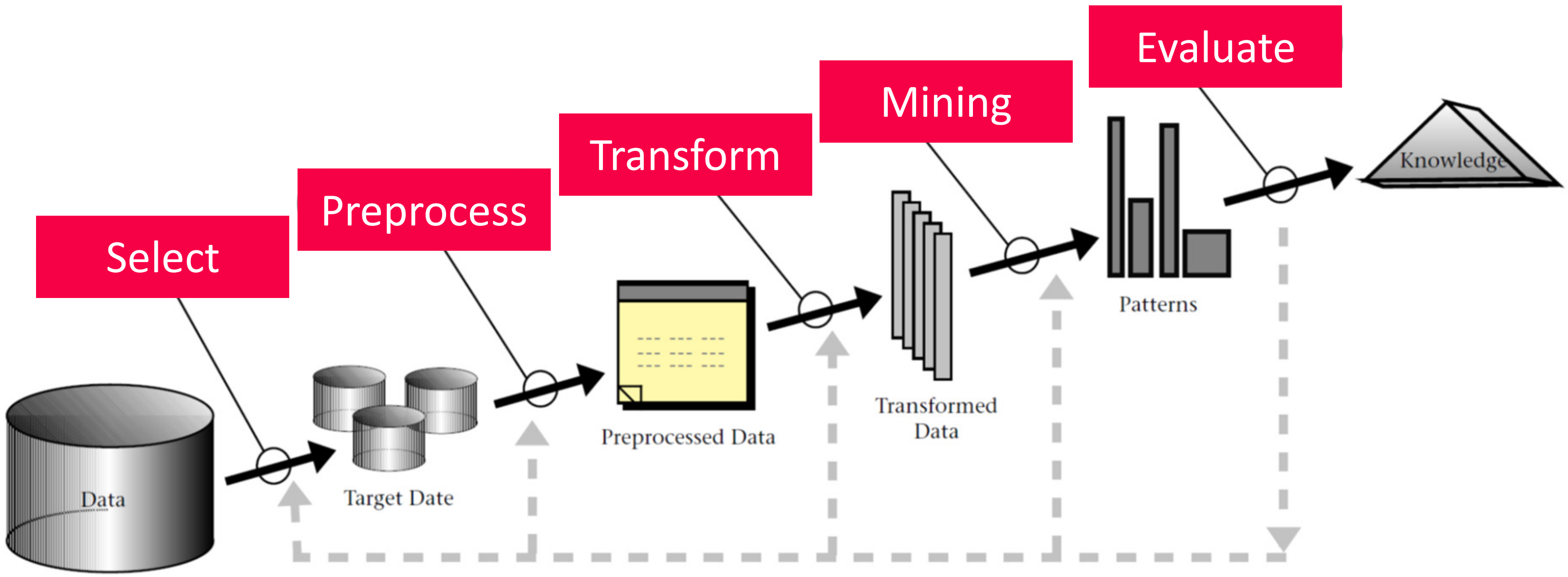
The Data Science Lifecycle

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



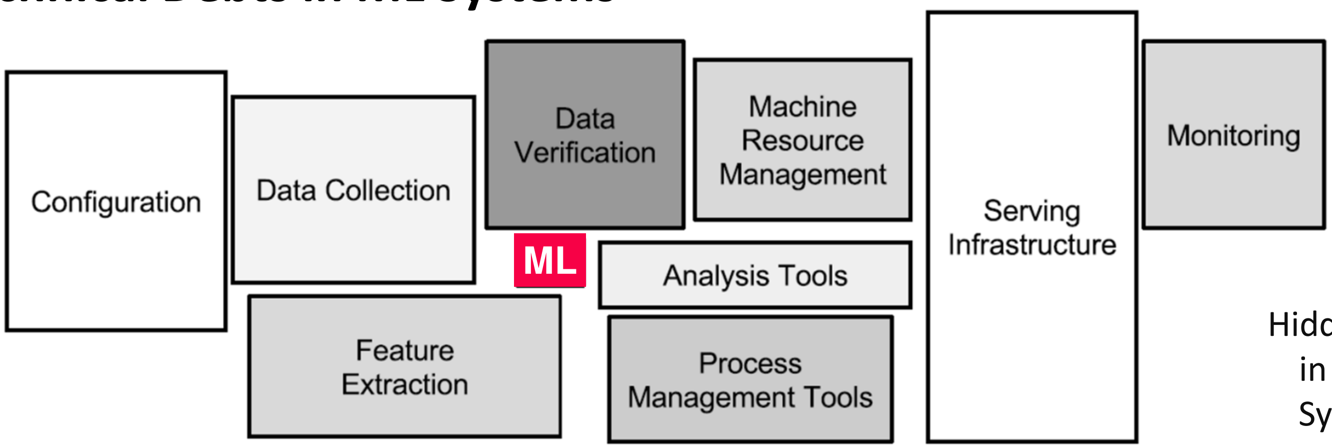
[Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth: From Data Mining to Knowledge Discovery in Databases. **AI Magazine 17(3) (1996)**]

The 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**

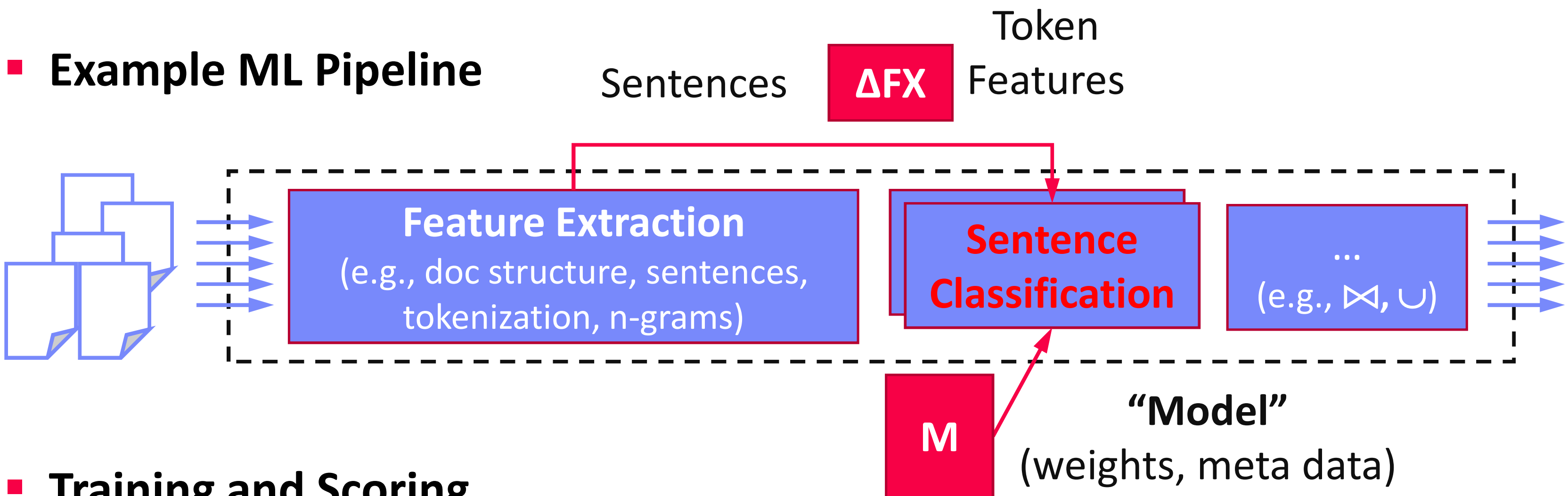


[D. Sculley et al.: Hidden Technical Debt in Machine Learning Systems. **NIPS 2015**]

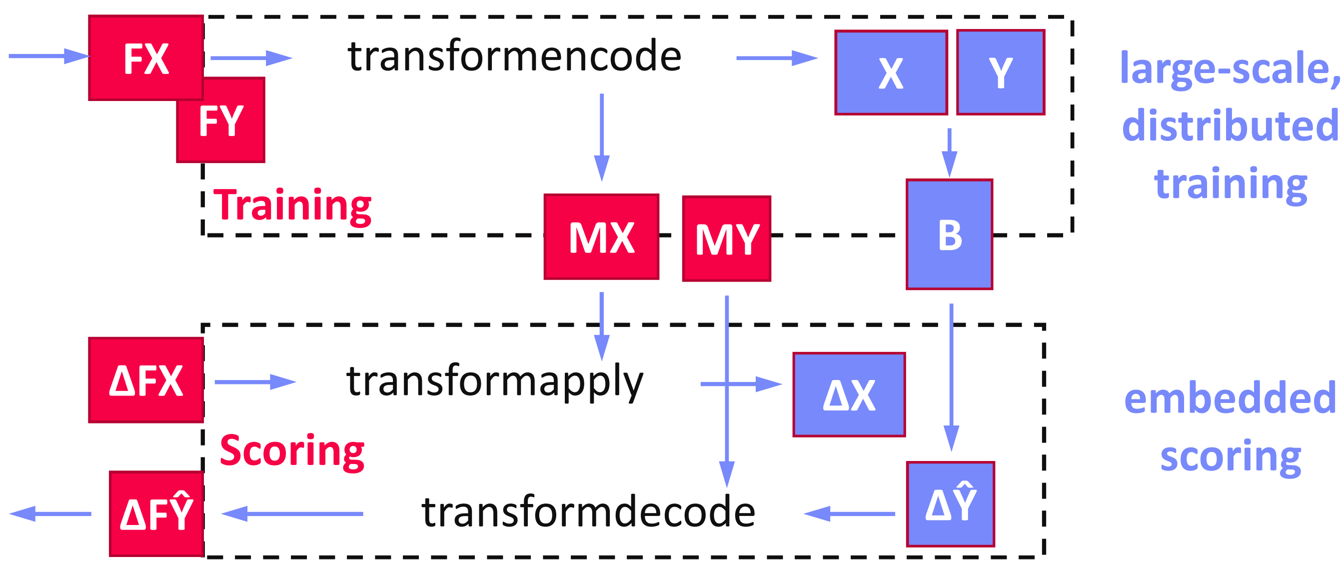
  - Glue code, pipeline jungles, dead code paths
  - Plain-old-data types, multiple languages, prototypes
  - Abstraction and configuration debts
  - Data testing, reproducibility, process management, and cultural debts

A Text Classification Scenario

- **Example ML Pipeline**



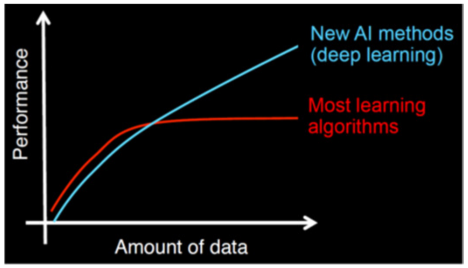
- **Training and Scoring**



ML Systems Stack

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, data programming/augmentation

Feedback Loop

Data → Model → Usage → Data

- **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

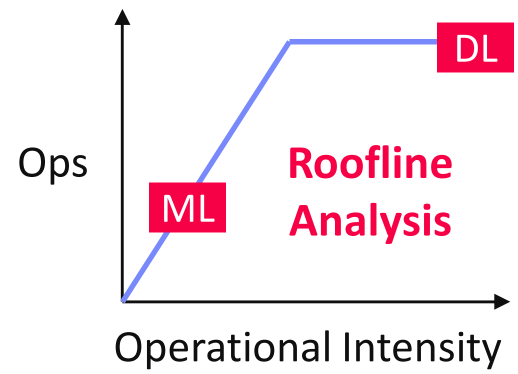
ML Apps & Algorithms	Supervised, unsupervised, RL linear algebra, libs, AutoML
Language Abstractions	Eager interpretation, lazy evaluation, prog. compilation
Fault Tolerance	Approximation, lineage, checkpointing, checksums, ECC
Execution Strategies	Local, distributed, cloud (data, task, parameter server)
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



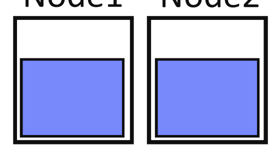
- **Graphics Processing Units (GPUs)**
  - Extensively used for deep learning training and scoring
  - NVIDIA Volta: "tensor cores" for 4x4 mm → 64 2B FMA instruction
- **Field-Programmable Gate Arrays (FPGAs)**
  - Customizable HW accelerators for prefiltering, compression, DL
  - Examples: Microsoft Catapult/Brainwave Neural Processing Units (NPUs)
- **Application-Specific Integrated Circuits (ASIC)**
  - Spectrum of chips: DL accelerators to computer vision
  - Examples: Google TPUs (64K 1B FMA), NVIDIA DLA, Intel NNP

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

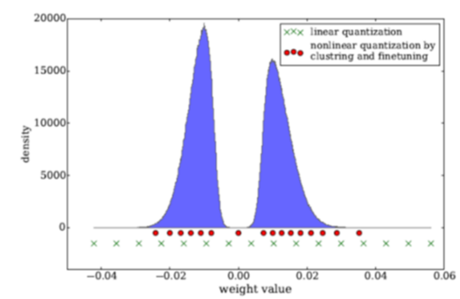
`vec(Berlin) – vec(Germany) + vec(France) ≈ vec(Paris)`

- **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
  - New data types: Intel Flexpoint (mantissa, exp)

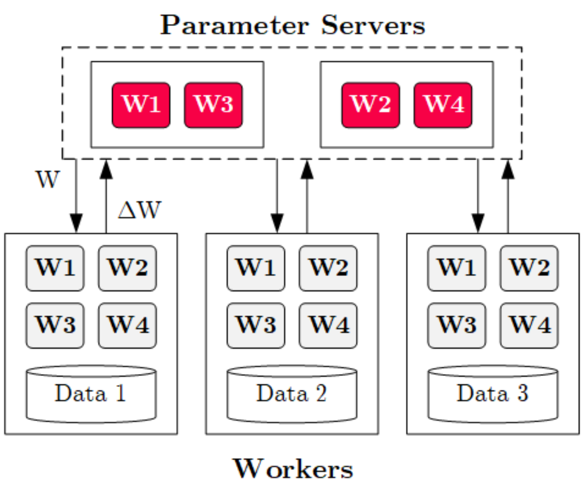
[**Credit:** Song Han'16]



Execution Strategies

Apps
Lang
Faults
Exec
Data
HW

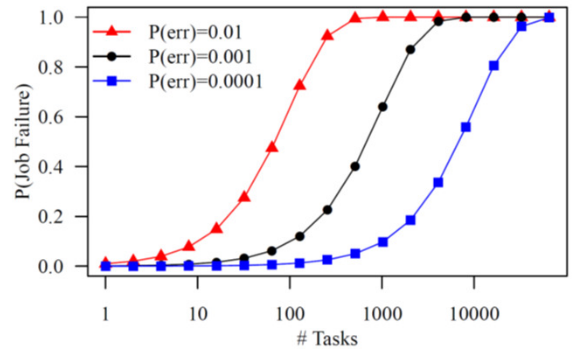
- **Batch Algorithms: Data and Task Parallel**
  - Data-parallel operations
  - Different physical operators
- **Mini-Batch Algorithms: Parameter Server**
  - Data-parallel and model-parallel PS
  - Update strategies (e.g., async, sync, backup)
  - Data partitioning strategies
  - Federated ML (trend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

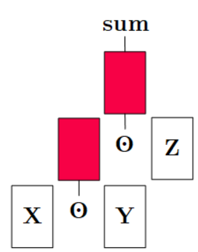
- **ML-specific Schemes** (exploit app characteristics)
  - Estimate contribution from lost partition to avoid strugglers
  - Example: user-defined "compensation" functions

Language Abstractions

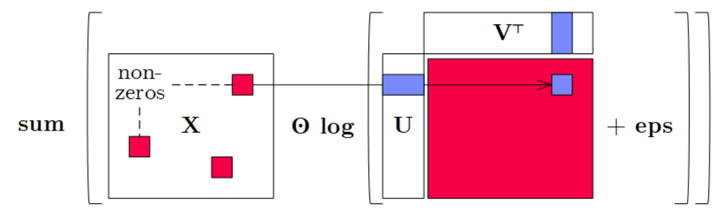
- **Optimization Scope**
  - **#1 Eager Interpretation** (debugging, no opt)
  - **#2 Lazy expression evaluation** (some opt, avoid materialization)
  - **#3 Program compilation** (full opt, difficult)

- **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



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]

Language Abstractions and System Architectures

Landscape of ML Systems

TUPAQ Mlbase Tupleware GraphLab HP Distributed R

Kasen Emma Cümülön(-D) OptiML

Glade Cumulon DMac RIOT-DB

LINVIEW Photon ML **SystemML** RIOT

Hemingway Mahout Samsara SAP HANA MS (Rev) R

Velox F Brainwash ORE **BigR** SciDB

Longview TensorDB DeepDive AzureML Fa R4ML

Columbus SimSQL Orion Zombie ScalOps MXNet

R BUDS Matlab Santoku LibFM KeystoneML Torch

Julia scikit-learn Sherlock ModelHub BigDL TensorFlow

Weka ModelDB Hamlet

Mahout CNTK Theano

SPSS VW Spark ML MADlib

SAS Spark R Flink ML Bismarck Keras Singa Caffe DL4J

Landscape of ML Systems, cont.

#1 Language Abstraction

- Linear Algebra Programs
- Computation Graphs
- Algorithm Libraries
- Operator Libraries

#2 Execution Strategies

- Parameter Server (Modell-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

UDF-based Systems

- **User-defined Functions (UDF)**
  - Data type: Input usually collections of cells, **rows, or blocks**
  - Implement loss and overall optimizer by yourself / UDF abstractions
  - Examples: **data-parallel** (e.g., Spark MLlib) or **In-DBMS analytics** (MADlib)

- **Example SQL**

Matrix Product in SQL

```sql
SELECT A.i, B.j,
  SUM(A.val*B.val)
FROM A, B
WHERE A.j = B.i
GROUP BY A.i, B.j;
```

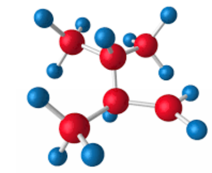
Matrix Product w/ UDF

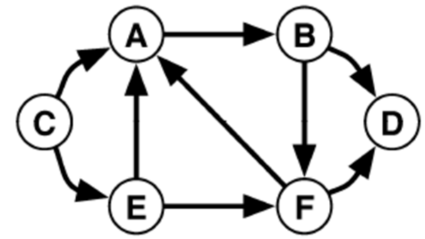
```sql
SELECT A.i, B.j,
  dot(A.row, B.col)
FROM A, B;
```

Optimization w/ UDA

```
Init(state)
Accumulate(state,data)
Merge(state,data)
Finalize(state,data)
```

Graph-based Systems



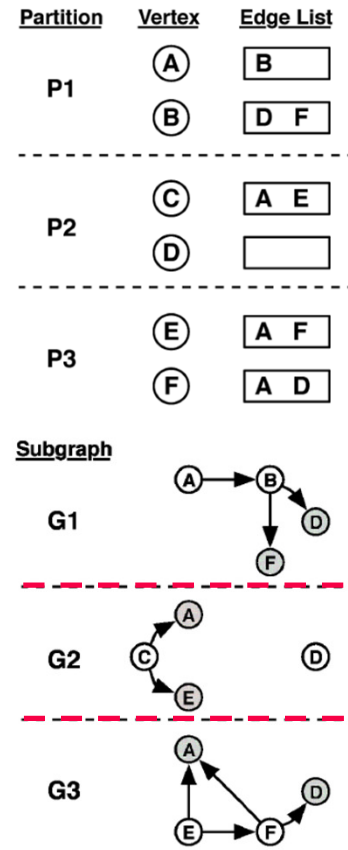


- **Large-scale Graph Processing**
  - Natively represent graph as nodes/edges
- **Think like a vertex**
  - Partition: a collection of vertices
  - Computation: a vertex and its edges
  - Communication: 1-hop at a time (e.g., A→B→D)

  [Grzegorz Malewicz et al: **Pregel:** a system for large-scale graph processing. **SIGMOD 2010**]

- **Think like a graph**
  - Partition: a proper subgraph
  - Computation: a subgraph
  - Communication: multiple-hops at a time e.g., A→D
  - **Graph partitioning**

  [Yuanyuan Tian et al: From "Think Like a Vertex" to "Think Like a Graph". **PVLDB 2013**]



Linear Algebra Systems

- **Comparison Query Optimization**
  - **Rule- and cost-based rewrites and operator ordering**
  - **Physical operator selection and query compilation**
  - Linear algebra / other ML operators, DAGs, control flow, sparse/dense formats

PL → **Compilers for Large-scale ML** ← **HPC**
DB → **Compilers for Large-scale ML**

- **#1 Interpretation** (operation at-a-time)
  - Examples: **R**, **PyTorch**, **Morpheus** [PVLDB'17]
- **#2 Lazy Expression Compilation** (DAG at-a-time)
  - Examples: **RIOT** [CIDR'09], **Mahout Samsara** [MLSystems'16]
  - Examples w/ control structures: **Weld** [CIDR'17], **OptiML** [ICML'11], **Emma** [SIGMOD'15]
- **#3 Program Compilation** (entire program)
  - Examples: **SystemML** [PVLDB'16], **Julia Cumulon** [SIGMOD'13], **Tupleware** [PVLDB'15]

Optimization Scope

```
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");
```

Linear Algebra Systems, cont.

- **Some Examples …**

Apache SystemML™

```
X = read("./X");
y = read("./y");
p = t(X) %*% y;
w = matrix(0,ncol(X),1);

while(...) {
  q = t(X) %*% X %*% p;
  ...
}
```

(Custom DSL w/ R-like syntax; program compilation)

MAHOUT

```
var X = drmFromHDFS("./X")
val y = drmFromHDFS("./y")
var p = (X.t %*% y).collect
var w = dense(...)
X = X.par(256).checkpoint()

while(...) {
  q = (X.t %*% X %*% p)
      .collect
  ...
}
```

(Embedded DSL in Scala; lazy evaluation)

TensorFlow

```
# read via queues
sess = tf.Session()
# ...
w = tf.Variable(tf.zeros(...,
  dtype=tf.float64))

while ...:
  v1 = tf.matrix_transpose(X)
  v2 = tf.matmult(X, p)
  v3 = tf.matmult(v1, v2)
  q = sess.run(v3)
  ...
```

(Embedded DSL in Python; lazy [and eager] evaluation)

ML Libraries

- **Fixed algorithm implementations**
  - Often on top of existing linear algebra or UDF abstractions

NumPy

SparkML/
MLlib

Single-node Example (Python)

```
from numpy import genfromtxt
from sklearn.linear_model \
  import LinearRegression

X = genfromtxt('X.csv')
y = genfromtxt('y.csv')

reg = LinearRegression()
  .fit(X, y)
out = reg.score(X, y)
```

Distributed Example (Spark Scala)

```
import org.apache.spark.ml
.regression.LinearRegression

val X = sc.read.csv('X.csv')
val y = sc.read.csv('y.csv')
val Xy = prepare(X, y).cache()

val reg = new LinearRegression()
  .fit(Xy)
val out reg.transform(Xy)
```

DL Frameworks

- **High-level DNN Frameworks**
  - Language abstraction for DNN construction and model fitting
  - Examples: Caffe, Keras

```
model = Sequential()
model.add(Conv2D(32, (3, 3),
padding='same',

input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(
  MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
...
```

```
opt = keras.optimizers.rmsprop(
  lr=0.0001, decay=1e-6)

# Let's train the model using RMSprop
model.compile(loss='cat…_crossentropy',
  optimizer=opt,
  metrics=['accuracy'])

model.fit(x_train, y_train,
  batch_size=batch_size,
  epochs=epochs,
  validation_data=(x_test, y_test),
  shuffle=True)
```

- **Low-level DNN Frameworks**
  - Examples: TensorFlow, MXNet, PyTorch, CNTK

A Critical Perspective on ML Systems (broad sense)

- **Recommended Reading**
  - 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

Programming Projects

Example Projects (to be refined by Mar 29)

- **#1: Auto Differentiation**
  - Implement auto differentiation for deep neural networks
  - Integrate auto differentiation framework in compiler or runtime
- **#2: Sparsity-Aware Optimization of Matrix Product Chains**
  - Integrate sparsity estimators into DP algorithm
  - Extend DP algorithm for DAGs and other operations
- **#3 Parameter Server Update Schemes**
  - New PS update schemes: e.g., stale-synchronous, Hogwild!
  - Language and local/distributed runtime extensions
- **#4 Extended I/O Framework for Other Formats**
  - Implement local readers/writers for NetCDF, HDF5, libsvm, and/or Arrow
- **#5: LLVM Code Generator**
  - Extend codegen framework by LLVM code generator
  - Native vector library, native operator skeletons, JNI bridge

Example Projects, cont. (to be refined by Mar 29)

- **#6 Data Validation Scripts**
  - Implement recently proposed integrity constraints
  - Write DML scripts to check a set of constraints on given dataset
- **#7 Data Cleaning Primitives**
  - Implement scripts or physical operators to perform data imputation and data cleaning (find and remove/fix incorrect values)
- **#8 Data Preparation Primitives**
  - Extend `transform` functionality for distributed binning
  - Needs to work in combination w/ dummy coding, recoding, etc