SCIENCE
PASSION
TECHNOLOGY

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

Last update: Mar 13, 2020

Announcements/Org

- **#1 Video Recording**
  - Link in **TeachCenter** & **TUbe** (lectures will be public)
  - Streaming: https://tugraz.webex.com/meet/m.boehm
- **#2 Course Registrations AMLS** (as of Mar 06)
  - COVID-19 precautions **March 11 – April 19**
  - **Project selection** by **Apr 03**
- **#3 Study Abroad Fair** (**Mar 18, 10am-3pm**, INF 25d HS i4)
  - Info booths and short presentations on study abroad programs (e.g., exchange, research, summer)
- **#4 Catalyst Coding Contest** (**Apr 03, 3-8pm**)
  - **Hosted by: IT Community Styria**
  - INF 18, HS i1 (117 seats)
  - https://register.codingcontest.org/

36

IT Community Styria

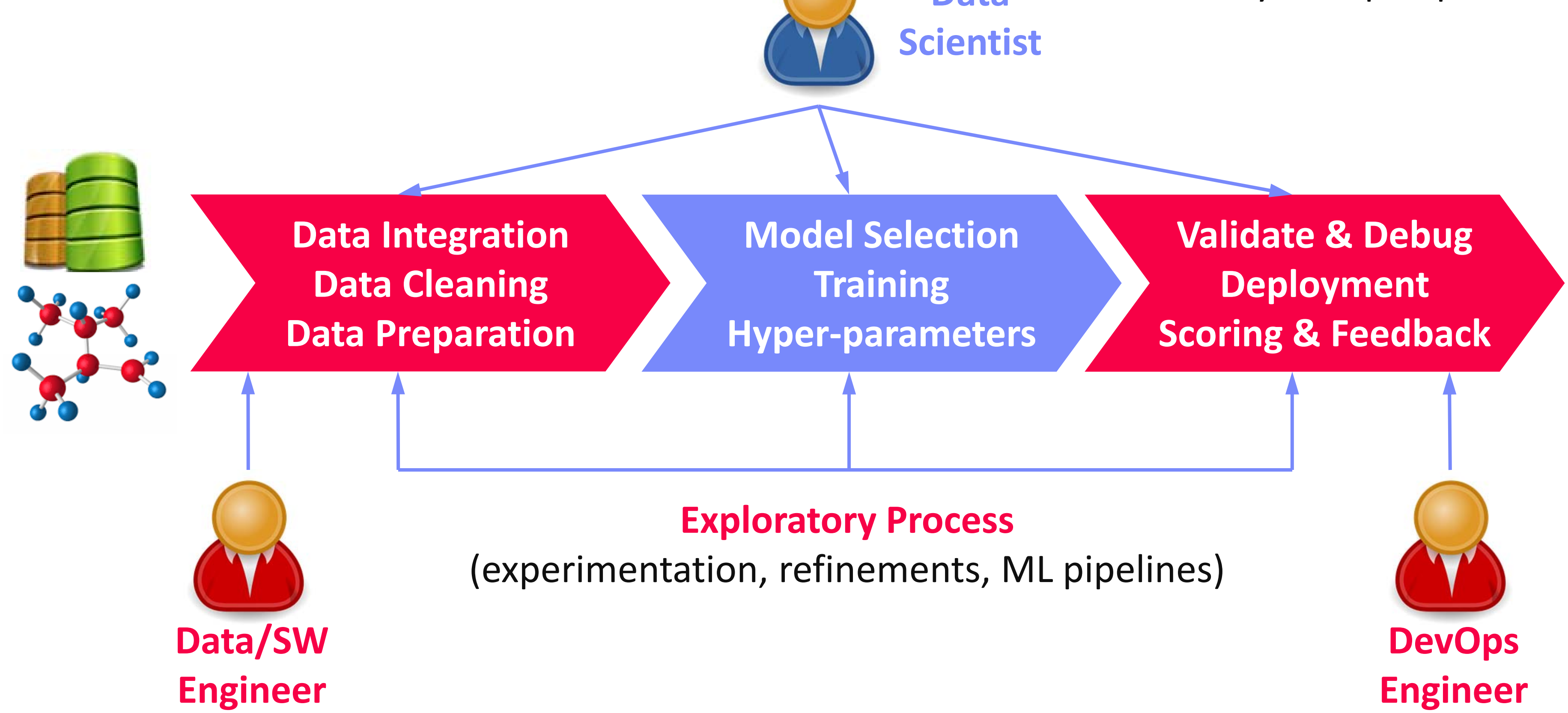
Agenda

- **Data Science Lifecycle**
- **ML Systems Stack**
- **Language Abstractions**
- **ML Systems Benchmarks**
- **Programming Projects**

Data Science Lifecycle

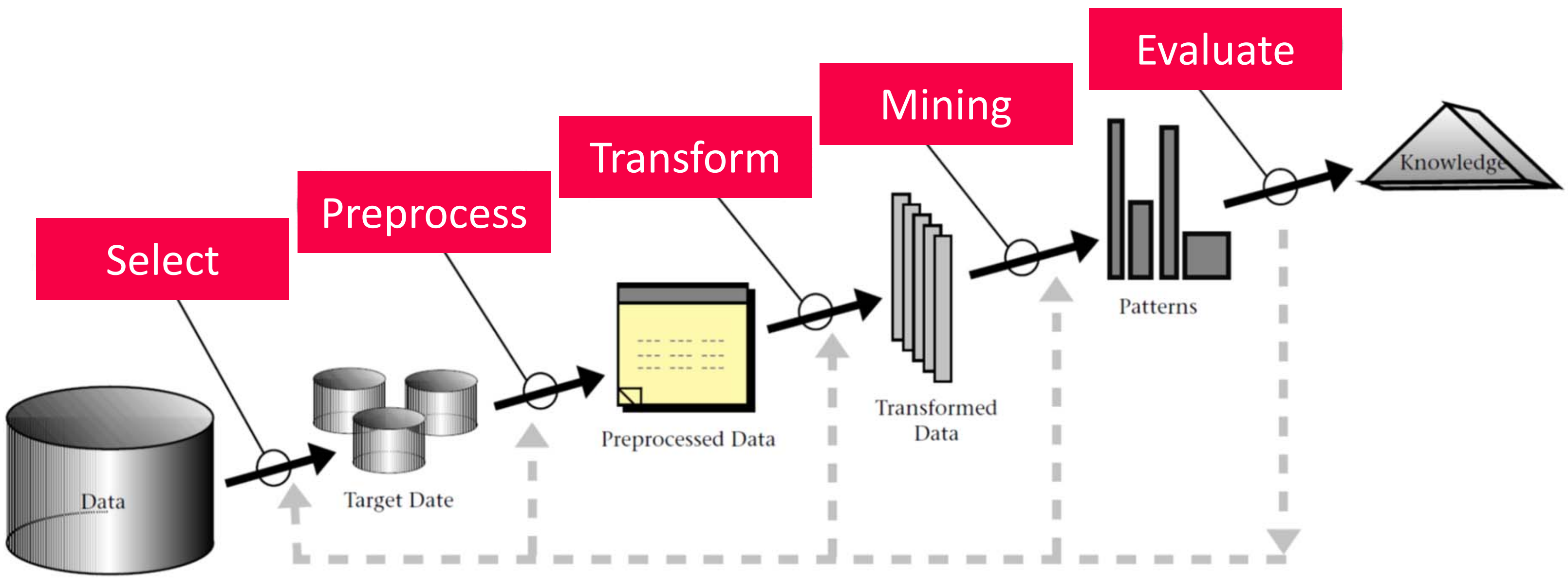
The Data Science Lifecycle

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



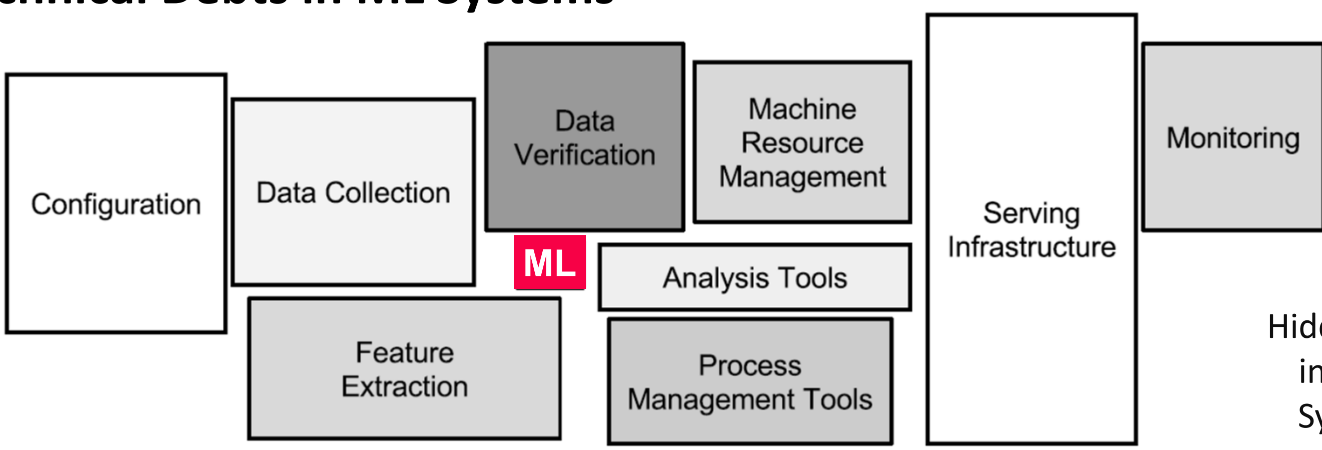
[Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth: From Data Mining to Knowledge Discovery in Databases. **AI Magazine 17(3) (1996)**]

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**

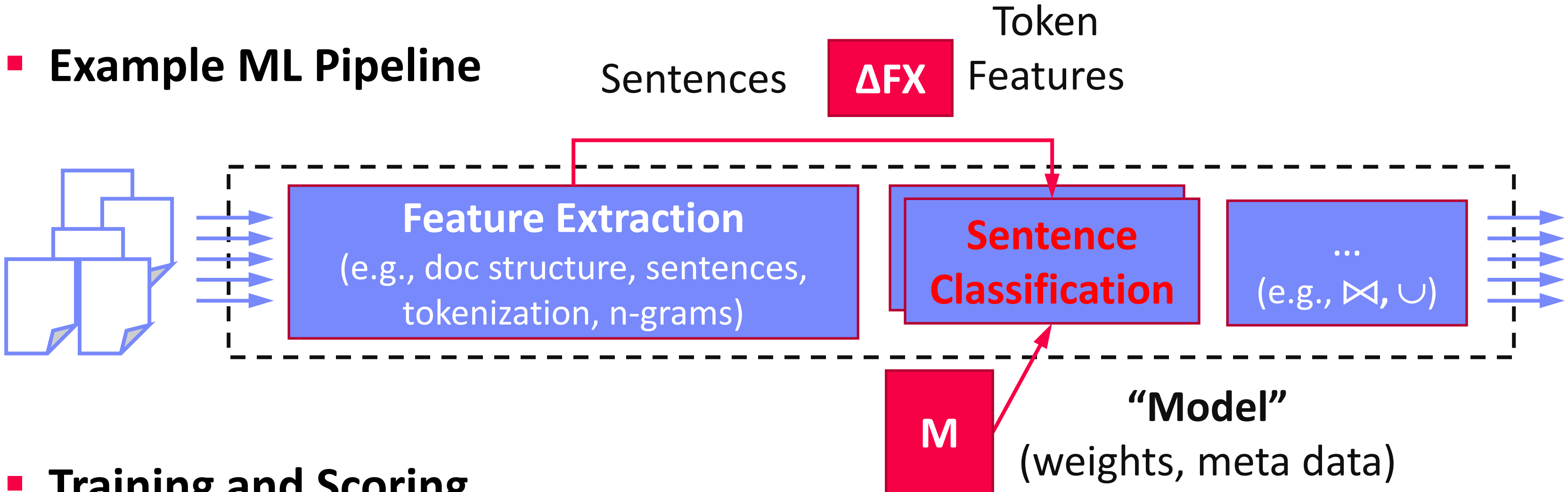


[D. Sculley et al.: Hidden Technical Debt in Machine Learning Systems. **NIPS 2015**]

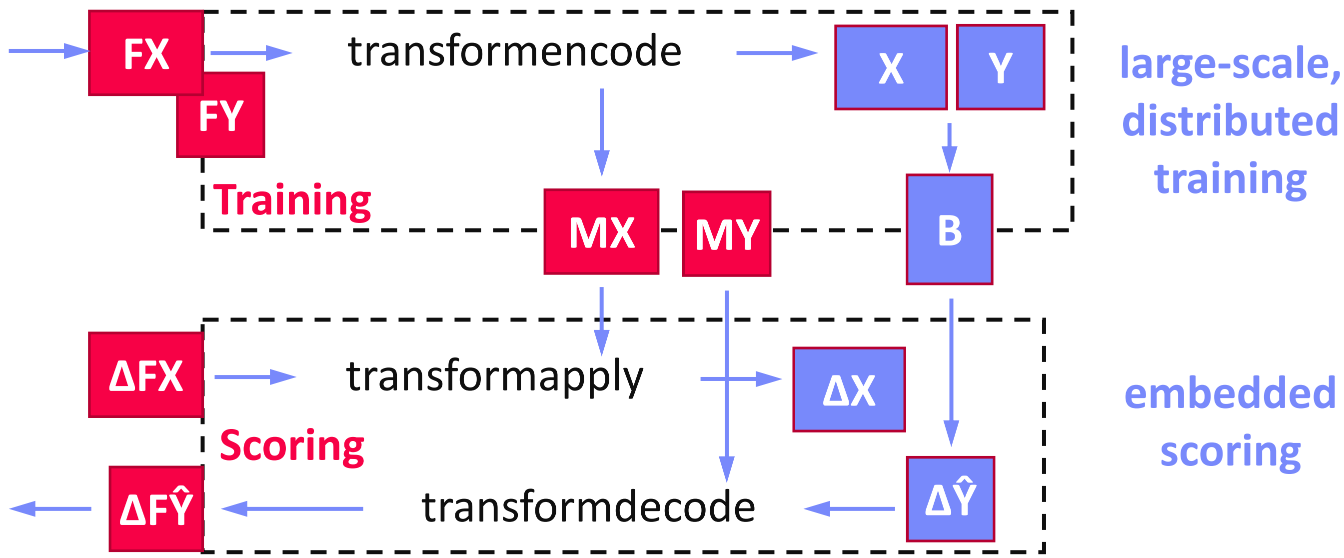
  - Glue code, pipeline jungles, dead code paths
  - Plain-old-data types, multiple languages, prototypes
  - Abstraction and configuration debts
  - Data testing, reproducibility, process management, and cultural debts

A Text Classification Scenario

- **Example ML Pipeline**

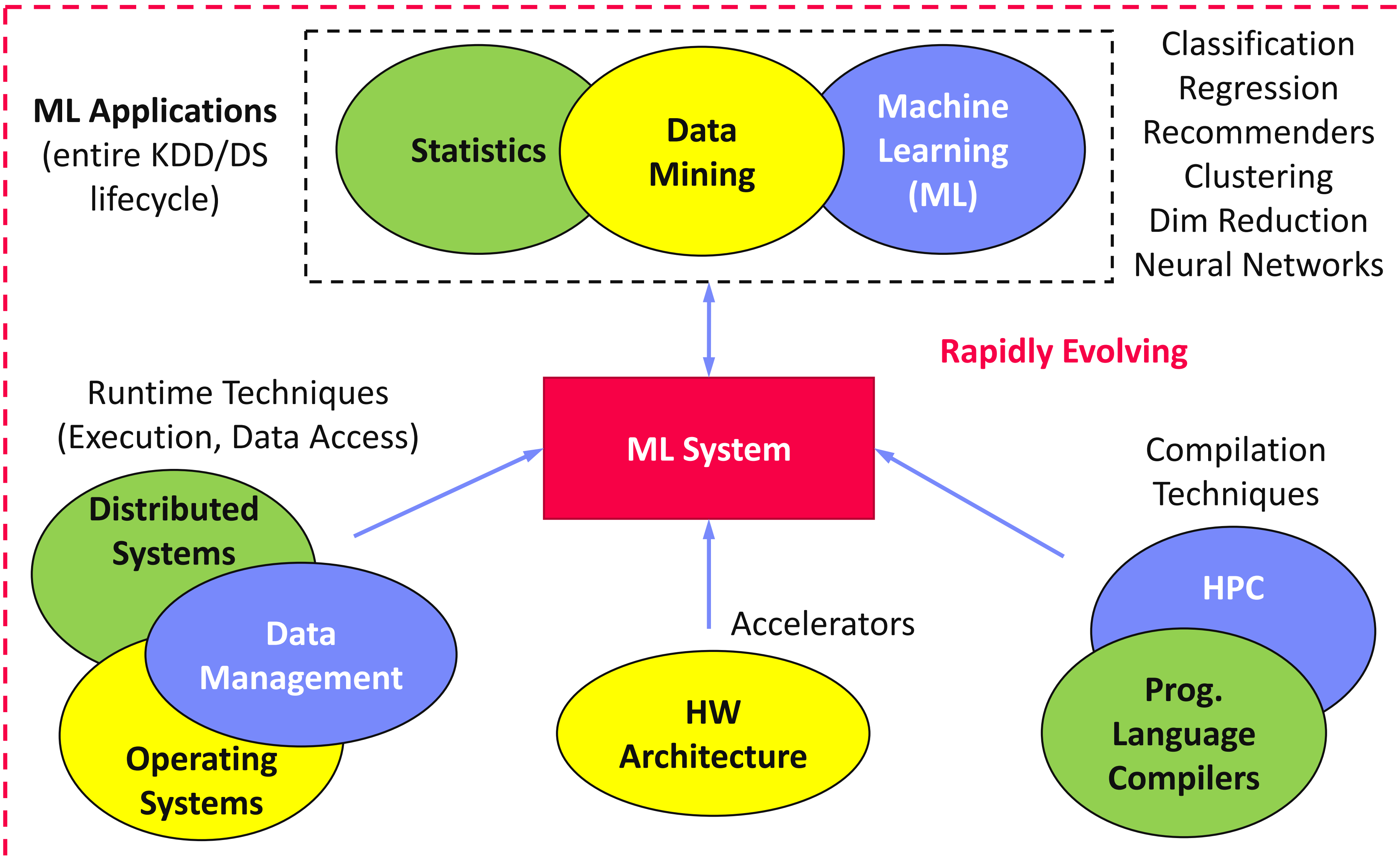


- **Training and Scoring**



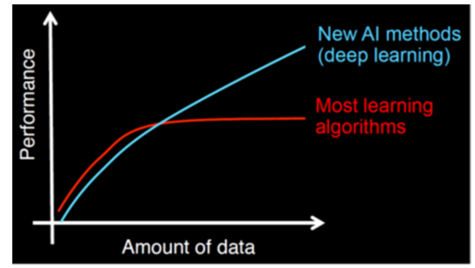
ML Systems Stack

What is an ML System?



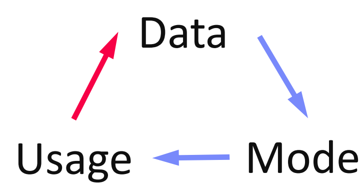
Driving Factors for ML

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



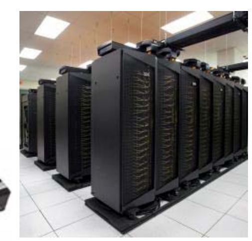
- **Improved Algorithms and Models**
  - Success across data and application domains (e.g., health care, finance, transport, production)
  - More complex models which leverage large data

Feedback Loop



- **Availability of Large Data Collections**
  - Increasing automation and monitoring ➔ data (simplified by cloud computing & services)
  - Feedback loops, **data programming/augmentation**

- **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

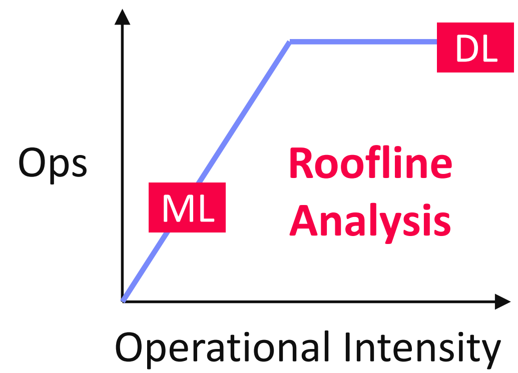
Concern	Layer	Examples
Hyper-parameter Tuning	ML Apps & Algorithms	Supervised, unsupervised, RL linear algebra, libs, AutoML
Model and Feature Selection	Language Abstractions	Eager interpretation, lazy evaluation, prog. compilation
Data Programming & Augmentation	Fault Tolerance	Approximation, lineage, checkpointing, checksums, ECC
Data Preparation (e.g., one-hot, binning)	Execution Strategies	Local, distributed, cloud (data, task, parameter server)
	Data Representations	Dense & sparse tensor/matrix; compress, partition, cache
Data Integration & Data Cleaning	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



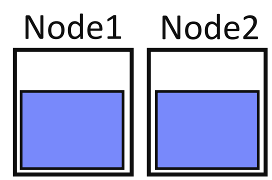
- **Graphics Processing Units (GPUs)**
  - Extensively used for deep learning training and scoring
  - NVIDIA Volta: "tensor cores" for 4x4 mm → 64 2B FMA instruction
- **Field-Programmable Gate Arrays (FPGAs)**
  - Customizable HW accelerators for prefiltering, compression, DL
  - Examples: Microsoft 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

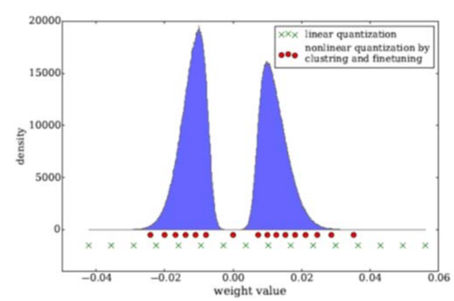
```
vec(Berlin) – vec(Germany)
+ vec(France) ≈ vec(Paris)
```

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



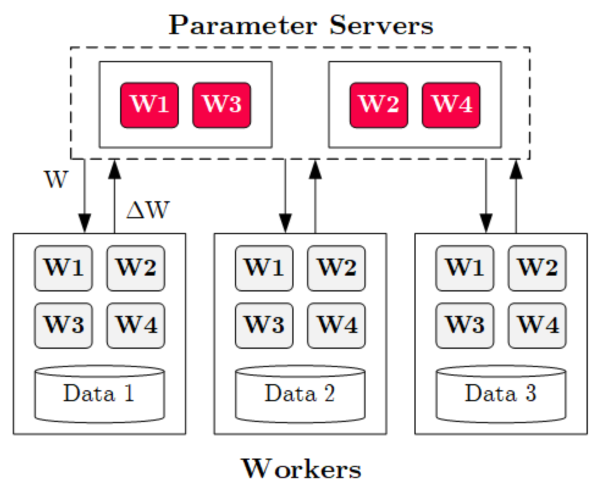
- **Lossy Compression → Acc/Perf-Tradeoff**
  - Sparsification (reduce non-zero values)
  - Quantization (reduce value domain), learned
  - New data types: Intel Flexpoint (mantissa, exp)

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



Execution Strategies

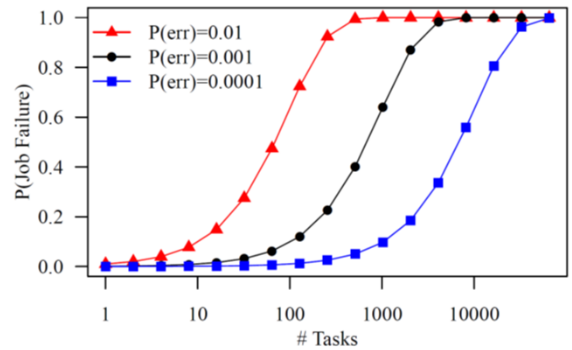
- **Batch Algorithms: Data and Task Parallel**
  - Data-parallel operations
  - Different physical operators
- **Mini-Batch Algorithms: Parameter Server**
  - Data-parallel and model-parallel PS
  - Update strategies (e.g., async, sync, backup)
  - Data partitioning strategies
  - Federated ML (trend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

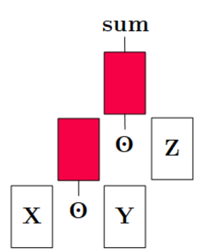
- **ML-specific Schemes** (exploit app characteristics)
  - Estimate contribution from lost partition to avoid strugglers
  - Example: user-defined "compensation" functions

Language Abstractions

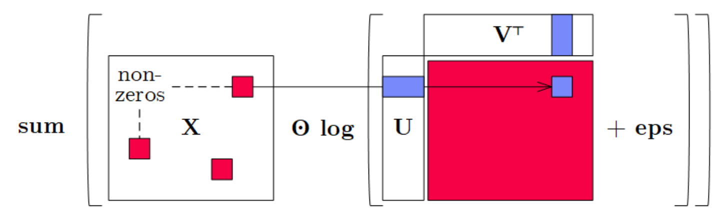
- **Optimization Scope**
  - **#1 Eager Interpretation** (debugging, no opt)
  - **#2 Lazy expression evaluation** (some opt, avoid materialization)
  - **#3 Program compilation** (full opt, difficult)

- **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



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

JAX AIDA Dask Ludwig

TUPAQ Mlbase Tupleware GraphLab HP Distributed R

Kasen Emma Cümülön(-D) OptiML

Glade Cumulon **SystemDS** 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 PyTorch

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 DL4J Caffe

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, AIDA)

- **Example SQL**

Matrix Product in 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

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

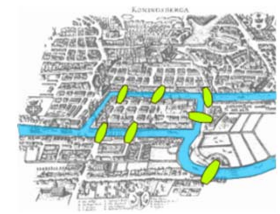
Optimization w/ UDA

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

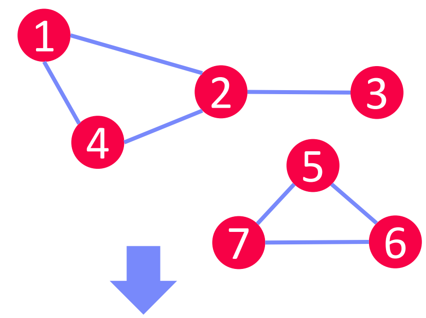
Graph-based Systems

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

- **Google Pregel**
  - Name: Seven Bridges of Koenigsberg (Euler 1736)
  - “Think-like-a-vertex” (vertex-centric processing)
  - Iterative processing in super steps, comm.: message passing



- **Programming Model**
  - Represent graph as collection of vertices w/ edge (adjacency) lists
  - Implement algorithms via Vertex API
  - Terminate if all vertices halted / no more msgs




```
public abstract class Vertex {
  public String getID();
  public long superstep();
  public VertexValue getValue();

  public compute(Iterator<Message> msgs);
  public sendMsgTo(String v, Message msg);
  public void voteToHalt();
}
```

2 [1, 3, 4]
7 [5, 6]
4 [1, 2]
1 [1, 2, 4]

Worker 1

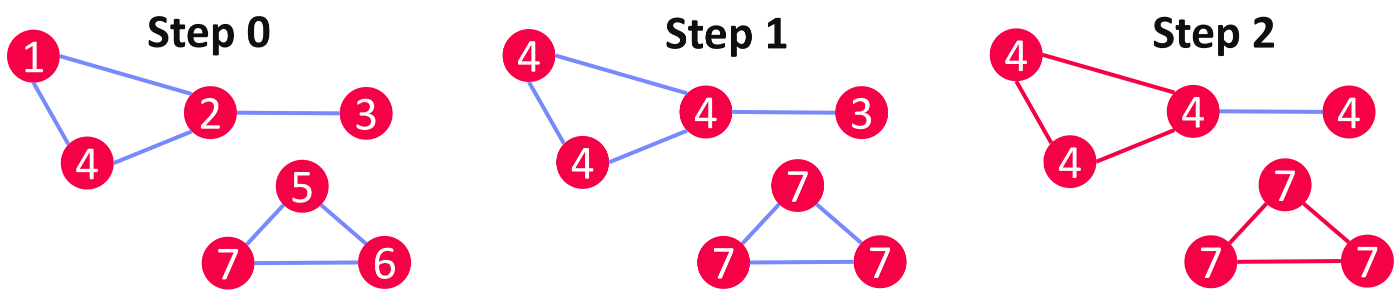
---

5 [6, 7]
3 [2]
6 [5, 7]

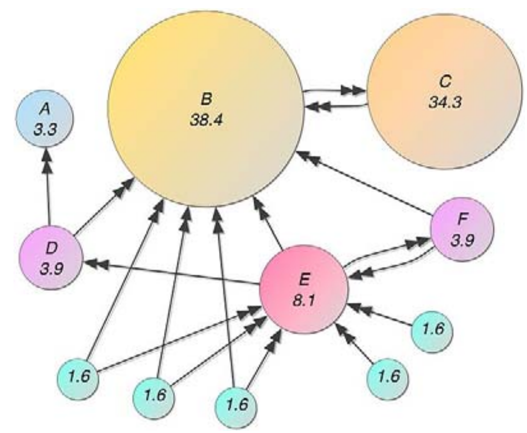
Worker 2

Graph-based Systems, cont.

- **Example1: Connected Components**
  - Determine connected components of a graph (subgraphs of connected nodes)
  - Propagate max(current, msgs) if != current to neighbors, terminate if no msgs



- **Example 2: Page Rank**
  - Ranking of webpages by importance / impact
  - **#1:** **Initialize vertices** to 1/numVertices()
  - **#2:** **In each super step**
    - Compute current vertex value:
      `value = 0.15/numVertices()+0.85*sum(msg)`
    - Send to all neighbors:
      `value/numOutgoingEdges()`



[**Credit:** https://en.wikipedia.org/wiki/PageRank]

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

DB

PL **HPC**

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], **TensorFlow** [OSDI'16] **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.

Note: TF 2.0

[Dan Moldovan et al.: AutoGraph: Imperative-style Coding with Graph-based Performance. **SysML 2019**.]

- **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)

```
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 (1.x)

```
# 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

SparkML/
MLlib

Single-node Example (Python)

```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)

```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)
```

DNN 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

Feature-centric Tools

- **DeepDive**
  - Knowledge base construction via SQL/MLNs
  - **Grounding:** SQL queries → factor graph
  - **Inference:** statistical inference on factor graph
  - Incremental maintenance via sampling / variational approach

[Jaeho Shin et al: Incremental Knowledge Base Construction Using DeepDive. **PVLDB 2015**]

- **Overton (Apple)**
  - Building, monitoring, improving ML pipelines
  - High-level abstractions: tasks and payloads
  - Data slicing, multi-task learning, data augmentation

[Christopher Ré et al: Overton: A Data System for Monitoring and Improving Machine-Learned Products, **CIDR 2020**]

- **Ludwig (Uber AI)**
  - Data types and configuration files
  - Encoders, combiners, decoders
  - **Example** "visual question answering":

[Piero Molino, Yaroslav Dudin, Sai Sumanth Miryala: Ludwig: a type-based declarative deep learning toolbox. **CoRR 2019**]

ML Systems Benchmarks

“Big Data” Benchmarks w/ ML Components

- **BigBench**
  - 30 workloads (6 statistics, 17 data mining)
  - Different data sources, processing types
  - **Note:** TPCx-BB, TPCx-HS [TPCTC 2016]

[Ahmad Ghazal et al: **BigBench:** towards an industry standard benchmark for big data analytics. **SIGMOD 2013**]

- **HiBench (Intel)**
  - MapReduce Micro benchmarks (WC, TeraSort)
  - IR/ML (e.g., PageRank, K-means, Naïve Bayes)

[Lan Yi, Jinquan Dai: Experience from Hadoop Benchmarking with **HiBench:** From Micro-Benchmarks Toward End-to-End Pipelines. **WBDB 2013**]

- **GenBase**
  - Preprocessing and ML in array databases

[Rebecca Taft et al: **GenBase:** a complex analytics genomics benchmark. **SIGMOD 2014**]

- **SparkBench**
  - Existing library algorithms (ML, Graph, SQL, stream)
  - ML: LogReg, SVM, matrix factorization, PageRank

[Dakshi Agrawal et al: **SparkBench** - A Spark Performance Testing Suite. **TPCTC 2015**]

Linear Algebra and DNN Benchmarks

- **SLAB: Scalable LA Benchmark (UCSD)**
  - **Ops:** TRANS, NORM, GRM, MVM, ADD, GMM
  - **Pipelines/Decompositions:** MMC, SVD
  - **Algorithms:** OLS, LogReg, NMF, HRSE

[Anthony Thomas, Arun Kumar: A Comparative Evaluation of Systems for Scalable Linear Algebra-based Analytics. **PVLDB 2018**]

- **DAWNBench (Stanford)**
  - Image Classification ImageNet: 93% top-5 val err
  - Image Classification CIFAR10: 94% test accuracy
  - Question Answering SQuAD: 0.75 F1 measure

[Cody Coleman et al.: DAWNBench: An End-to-End Deep Learning Benchmark and Competition, **ML Systems Workshop 2017**]

- **MLPerf**
  - Image classification ImageNet, object detection COCO, translation WMT En-Ger, recommendation MovieLens, reinforcement learning GO
  - **Train to target accuracy**

[Peter Mattson et al.: MLPerf Training Benchmark, **MLSys 2020**]

DNN Benchmarks, cont.

[**MLPerf** v0.6: https://mlperf.org/training-results-0-6/]

Closed Division Times

#	Submitter	System	Processor	#	Accelerator	#	Software	Benchmark results (minutes): Image classification, ImageNet, ResNet-50 v1.5	Object detection, light-weight, COCO, SSD w/ ResNet-34	Object detection, heavy-wt., COCO, Mask-R-CNN	Translation, recurrent, WMT E-G, NMT	Translation, non-recur., WMT E-G, Transformer	Recom-mendation, MovieLens-20M, NCF	Reinforce-ment Learning, Go, Mini Go	Details	Code	Notes
Available in cloud																	
0.6-1	Google	TPUv3.32			TPUv3	16	TensorFlow, TPU 1.14.1.dev	42.19	12.61	107.03	12.25	10.20	[1]		details	code	none
0.6-2	Google	TPUv3.128			TPUv3	64	TensorFlow, TPU 1.14.1.dev	11.22	3.89	57.46	4.62	3.85	[1]		details	code	none
0.6-3	Google	TPUv3.256			TPUv3	128	TensorFlow, TPU 1.14.1.dev	6.86	2.76	35.60	3.53	2.81	[1]		details	code	none
0.6-4	Google	TPUv3.512			TPUv3	256	TensorFlow, TPU 1.14.1.dev	3.85	1.79		2.51	1.58	[1]		details	code	none
0.6-5	Google	TPUv3.1024			TPUv3	512	TensorFlow, TPU 1.14.1.dev	2.27	1.34		2.11	1.05	[1]		details	code	none
0.6-6	Google	TPUv3.2048			TPUv3	1024	TensorFlow, TPU 1.14.1.dev	1.28	1.21			0.85	[1]		details	code	none
Available on-premise																	
0.6-7	Intel	32x 2S CLX 8260L	CLX 8260L	64			TensorFlow						[1]	14.43	details	code	none
0.6-8	NVIDIA	DGX-1			Tesla V100	8	MXNet, NGC19.05	115.22					[1]		details	code	none
0.6-9	NVIDIA	DGX-1			Tesla V100	8	PyTorch, NGC19.05		22.36	207.48	20.55	20.34	[1]		details	code	none
0.6-10	NVIDIA	DGX-1			Tesla V100	8	TensorFlow, NGC19.05						[1]	27.39	details	code	none
0.6-11	NVIDIA	3x DGX-1			Tesla V100	24	TensorFlow, NGC19.05						[1]	13.57	details	code	none
0.6-12	NVIDIA	24x DGX-1			Tesla V100	192	PyTorch, NGC19.05			22.03			[1]		details	code	none
0.6-13	NVIDIA	30x DGX-1			Tesla V100	240	PyTorch, NGC19.05		2.67				[1]		details	code	none
0.6-14	NVIDIA	48x DGX-1			Tesla V100	384	PyTorch, NGC19.05				1.99		[1]		details	code	none
0.6-15	NVIDIA	60x DGX-1			Tesla V100	480	PyTorch, NGC19.05					2.05	[1]		details	code	none
0.6-16	NVIDIA	130x DGX-1			Tesla V100	1040	MXNet, NGC19.05	1.69					[1]		details	code	none
0.6-17	NVIDIA	DGX-2			Tesla V100	16	MXNet, NGC19.05	57.87									
0.6-18	NVIDIA	DGX-2			Tesla V100	16	PyTorch, NGC19.05		12.21	101.00	10.94	11.04					
0.6-19	NVIDIA	DGX-2H			Tesla V100	16	MXNet, NGC19.05	52.74									
0.6-20	NVIDIA	DGX-2H			Tesla V100	16	PyTorch, NGC19.05		11.41	95.20	9.87	9.80					
0.6-21	NVIDIA	4x DGX-2H			Tesla V100	64	PyTorch, NGC19.05		4.78	32.72							
0.6-22	NVIDIA	10x DGX-2H			Tesla V100	160	PyTorch, NGC19.05					2.41					
0.6-23	NVIDIA	12x DGX-2H			Tesla V100	192	PyTorch, NGC19.05			18.47							
0.6-24	NVIDIA	15x DGX-2H			Tesla V100	240	PyTorch, NGC19.05		2.56								
0.6-25	NVIDIA	16x DGX-2H			Tesla V100	256	PyTorch, NGC19.05				2.12						
0.6-26	NVIDIA	24x DGX-2H			Tesla V100	384	PyTorch, NGC19.05				1.80						
0.6-27	NVIDIA	30x DGX-2H, 8 chips each			Tesla V100	240	PyTorch, NGC19.05		2.23								
0.6-28	NVIDIA	30x DGX-2H			Tesla V100	480	PyTorch, NGC19.05					1.59					
0.6-29	NVIDIA	32x DGX-2H			Tesla V100	512	MXNet, NGC19.05	2.59									
0.6-30	NVIDIA	96x DGX-2H			Tesla V100	1536	MXNet, NGC19.05	1.33									

96 x DGX-2H = 96 * 16 = 1536 V100 GPUs

➔ ~ 96 * $400K = **$35M – $40M**

[https://www.forbes.com/sites/tiriasresearch/2019/06/19/nvidia-offers-a-turnkey-supercomputer-the-dgx-superpod/#693400f43ee5]

AutoML and Data Cleaning

- **MLBench**
  - Compare **AutoML** w/ human experts (Kaggle)
  - Classification, regression; AUC vs Runtime

[Yu Liu, Hantian Zhang, Luyuan Zeng, Wentao Wu, Ce Zhang: MLBench: Benchmarking Machine Learning Services Against Human Experts. **PVLDB 2018**]

- **(Open Source) AutoML Benchmark**
  - 39 classification datasets, AUC metric, 10-fold CV
  - Extensible metrics, OS **AutoML** frameworks, datasets

[Pieter Gijsbers et al.: An Open Source AutoML Benchmark. **Automated ML Workshop 2019**]

- **CleanML**
  - Train/Test on **dirty vs clean data** (2x2)
  - Missing values, outliers, duplicates, mislabels

[Peng Li et al: CleanML: A Benchmark for Joint Data Cleaning and Machine Learning, **CoRR 2019**]

- **Meta Worlds Benchmark**
  - **Meta-reinforcement** and **multi-task learning**
  - 50 robotic manipulation tasks (e.g., get coffee, open window, pick & place)

[Tianhe Yu et al: Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning, **CoRL 2019**]

Programming Projects

Refinement until March 27

(bring you own if you want)

Project Selection by April 03

Example Projects APIs/Tools

- **#1 Extended Python and Java Language Bindings**
- **#2 Auto Differentiation** (builtin function and compiler)
- **#3 Built-in Functions for Regression, Classification, Clustering**
- **#4 Built-in Functions for Time Series Missing Value Imputation**
- **#5 DL-based Entity Resolution Primitives** (baseline implementation)
- **#6 Model Selection Primitives** (BO, multi-armed bandit, hyperband)
- **#7 Neural Collaborative Filtering** (see MLPerf benchmark)
- **#8 Quantum Neural Networks** (Grover's Quantum Search, Qiskit/TFQ)

- **#9 SLAB Benchmark** (benchmark driver, summary)
- **#10 Documentation and Tutorials** (for different target users)
- **#11 Extended Test Framework** (comparisons, caching, remove redundancy)
- **#12 Performance Testsuite** (extend algorithm-level suite)
- **#13 ONNX Graph Importer/Exporter** (DML script / HOP DAG generation)

Example Projects Compiler/Runtime

- **#14** Loop Vectorization Rewrites (more general framework)
- **#15** Canonicalization Rewrite Framework (refactoring, new rewrites)
- **#16** Extended CSE & Constant Folding (commutativity, one-shot)
- **#17** Extended Matrix Multiplication Chain Opt (sparsity, rewrites)
- **#18** Extended Update In-Place Framework (reference counting)
- **#19** Operator Scheduling Algorithms (baselines)

- **#20** Lazy / Asynchronous Instruction Evaluation
- **#21** SLIDE Operators and Runtime Integration (Sub-LInear DL Engine)
- **#22** Compression Planning Extensions (co-coding search algorithm)
- **#23** Feature Transform: Equi-Height/Custom Binning (local, distributed)
- **#24** Extended Intel MKL-DNN Runtime Operations (beyond conv2d)
- **#25** Extended I/O Framework for Other Formats (e.g., NetCDF, HDF5, Arrow)
- **#26** Protobuf reader/writer into Data Tensor (local, distributed)

Summary and Q&A

- **Data Science Lifecycle**
- **ML Systems Stack**
- **Language Abstractions**
- **ML System Benchmarks**
- **Programming Projects** (first come, first serve)

- **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