Data Integration and Analysis
13 Distributed ML Systems

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: Jan 31, 2019

Announcements/Org

- **#1 Video Recording**
  - Link in **TeachCenter** & **TUbe** (lectures will be public)
- **#2 DIA Projects**
  - **13 Projects** selected (various topics)
  - **3 Exercises** selected (distributed data deduplication)
  - → grace period: **end of Feb**
- **#3 Exam**
  - **Feb 3, 1pm – Feb 5, 2pm**, remote exam possible
  - Oral exam, **45min slots**, first-come, first-serve
  - https://doodle.com/poll/ikzsffek2vhd85q4
- **#4 Course Evaluation**
  - Evaluation time frame: **Jan 14 – Feb 14** → feedback

If problems, **please ask for help**

2/21

#5 Data Management Courses

Architecture of Database Systems (ADBS, **WS**)

DB system internals
+ prog. project

Architecture of ML Systems (AMLS, **SS**)

ML system internals
+ prog. project
in **SystemDS**

[github.com/tugraz-isds/systemds]

Master

Bachelor

Data Integration and Large-Scale Analysis (DIA, **WS**)

Distributed
Data Management
(usage and internals)

Data Management / Databases (DM, **SS+WS**)

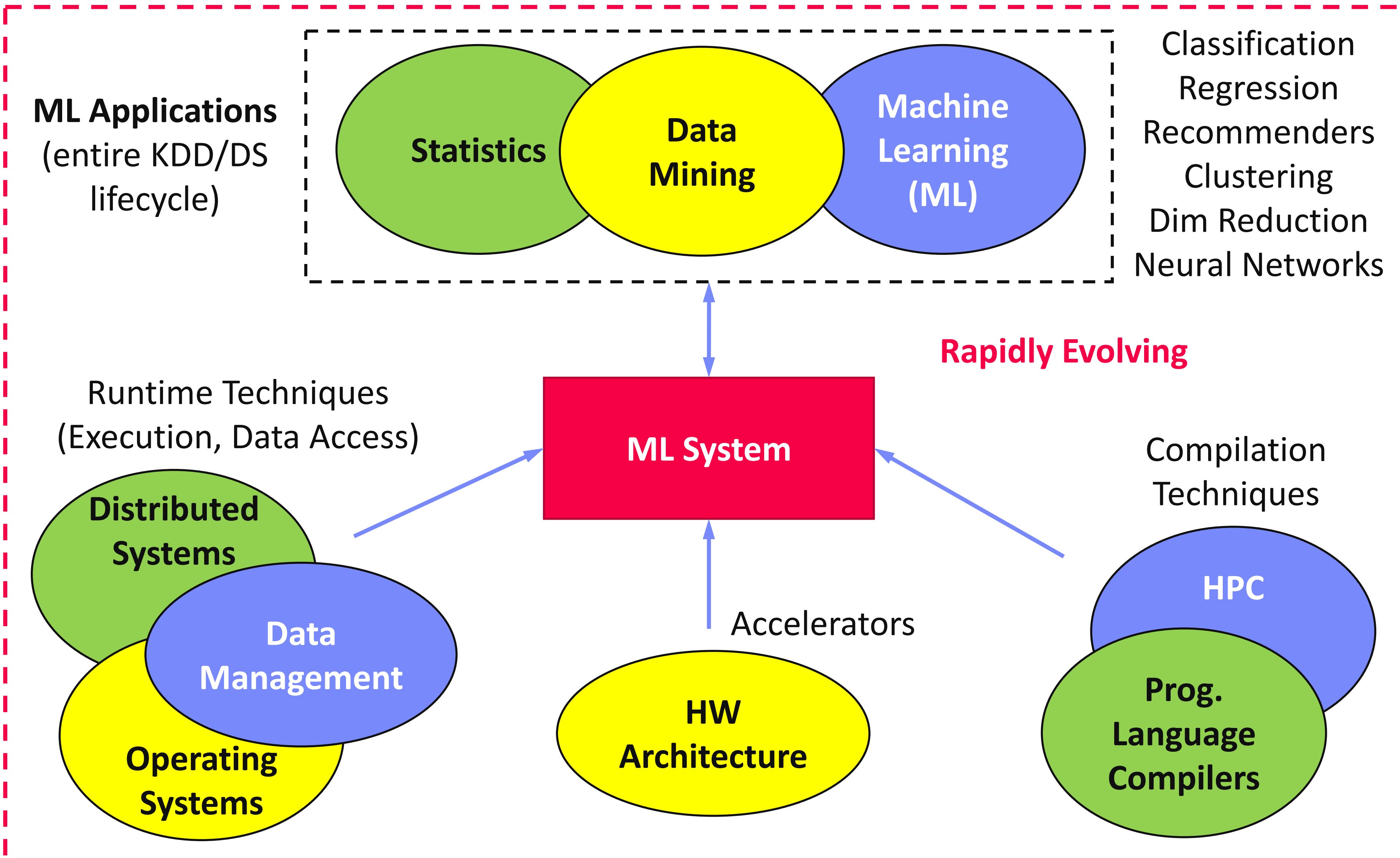
Data management from user/application perspective

Agenda

- **Landscape of ML Systems**
- **Distributed Linear Algebra**
- **Distributed Parameter Servers**

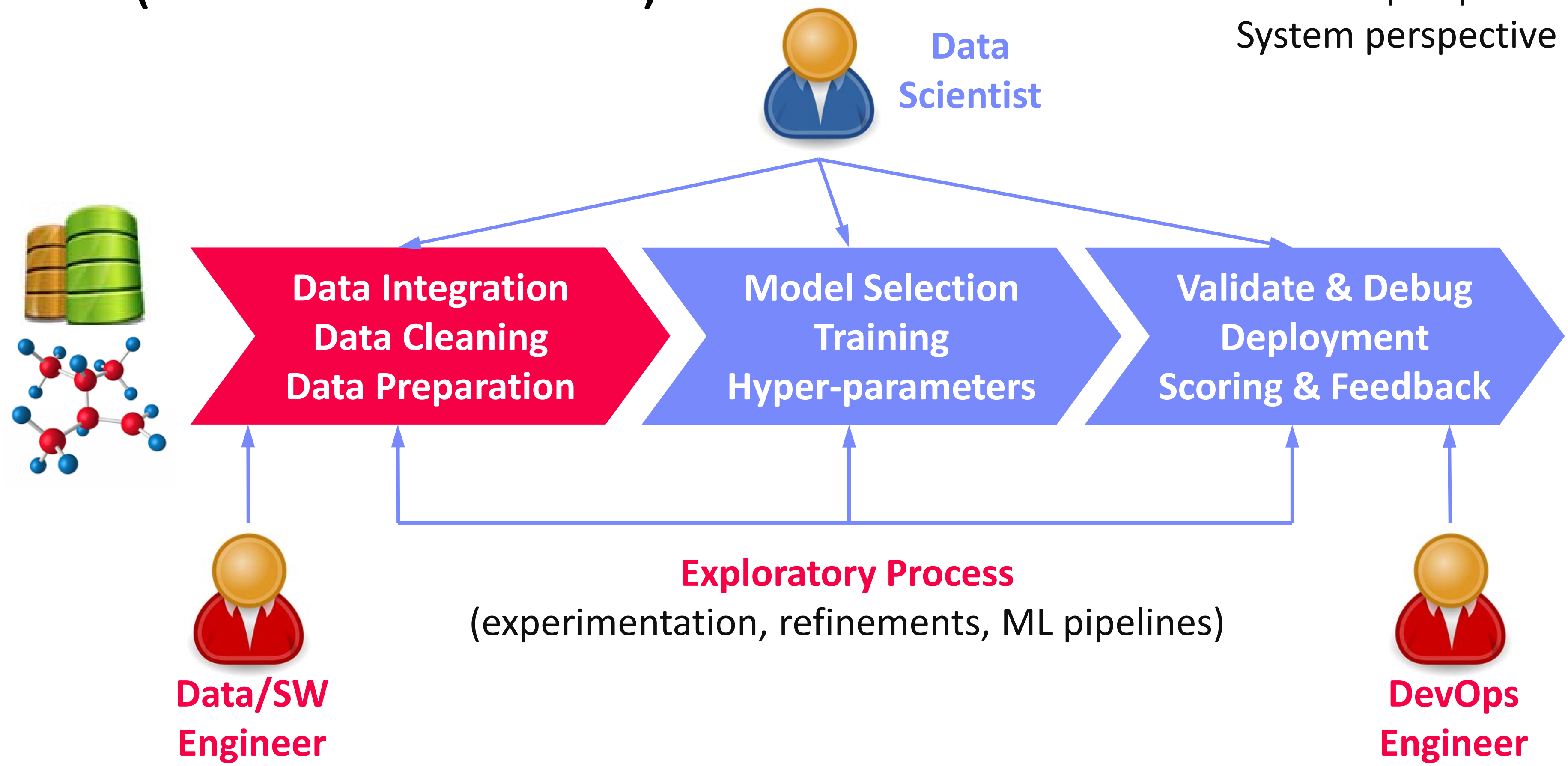
Landscape of ML Systems

What is an ML System?



The Data Science Lifecycle (aka KDD Process)

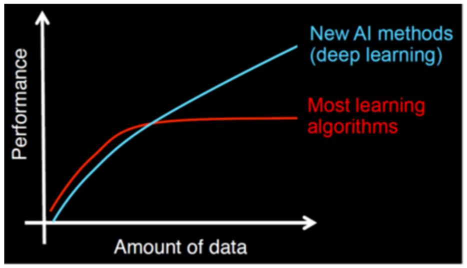
Data-centric View:
Application perspective
Workload perspective
System perspective



Driving Factors for ML

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

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



- **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

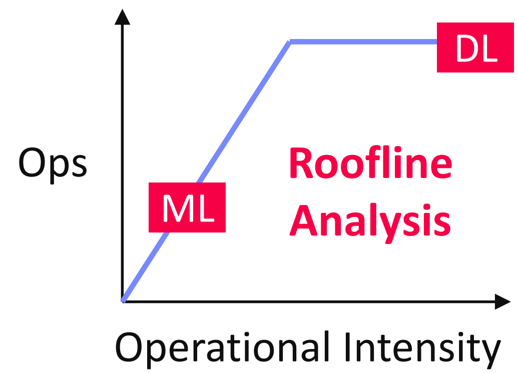
ML Apps & Algorithms	Supervised, unsupervised, RL linear algebra, libs, AutoML
Language Abstractions	Eager interpretation, lazy evaluation, prog. compilation
Fault Tolerance	Approximation, lineage, checkpointing, checksums, ECC
Execution Strategies	Local, distributed, cloud (data, task, parameter server)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



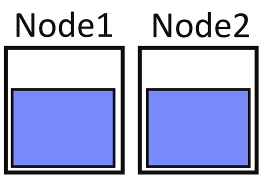
- **Graphics Processing Units (GPUs)**
  - Extensively used for deep learning training and scoring
  - NVIDIA Volta: "tensor cores" for 4x4 mm → 64 2B FMA instruction
- **Field-Programmable Gate Arrays (FPGAs)**
  - Customizable HW accelerators for prefiltering, compression, DL
  - Examples: Microsoft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

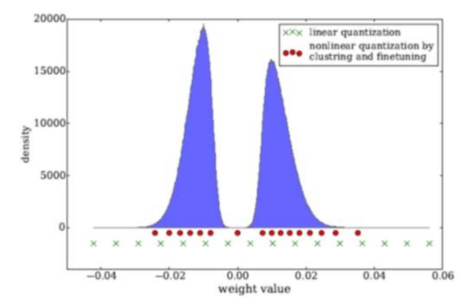
    `vec(Berlin) – vec(Germany) + vec(France) ≈ vec(Paris)`

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



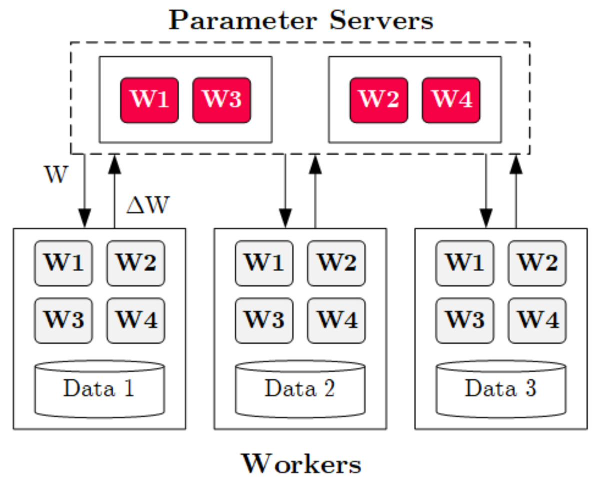
- **Lossy Compression → Acc/Perf-Tradeoff**
  - Sparsification (reduce non-zero values)
  - Quantization (reduce value domain), learned
  - New data types: Intel Flexpoint (mantissa, exp)

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



Execution Strategies

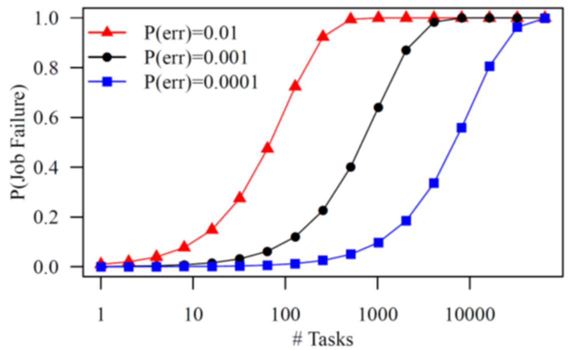
- **Batch Algorithms: Data and Task Parallel**
  - Data-parallel operations
  - Different physical operators
- **Mini-Batch Algorithms: Parameter Server**
  - Data-parallel and model-parallel PS
  - Update strategies (e.g., async, sync, backup)
  - Data partitioning strategies
  - Federated ML (trend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

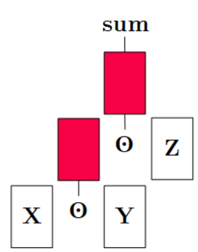
- **ML-specific Schemes** (exploit app characteristics)
  - Estimate contribution from lost partition to avoid strugglers
  - Example: user-defined “compensation” functions

Language Abstractions

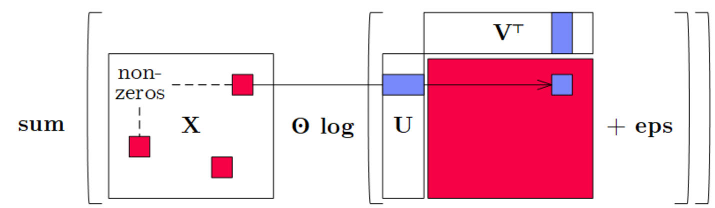
- **Optimization Scope**
  - **#1 Eager Interpretation** (debugging, no opt)
  - **#2 Lazy expression evaluation** (some opt, avoid materialization)
  - **#3 Program compilation** (full opt, difficult)

- **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]

Landscape of ML Systems

#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**

Distributed Linear Algebra

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 **DB** **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], **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)

```
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))
...
```

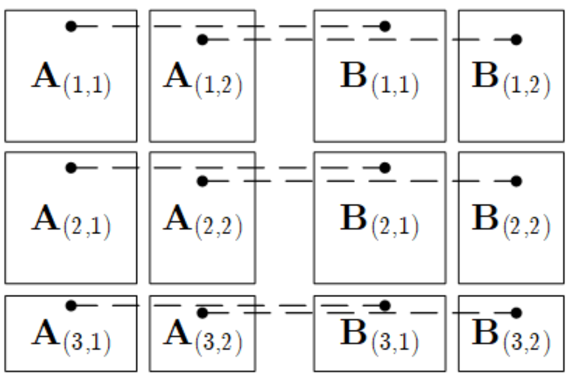
```
opt = keras.optimizers.rmsprop(
  lr=0.0001, decay=1e-6)

# 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

Distributed Matrix Operations

Elementwise Multiplication
(Hadamard Product)

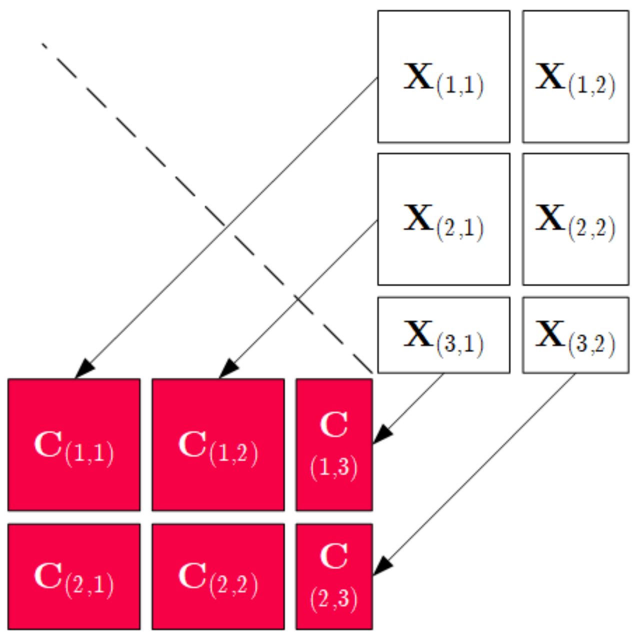
`C = A * B`



Note: also with
row/column vector rhs

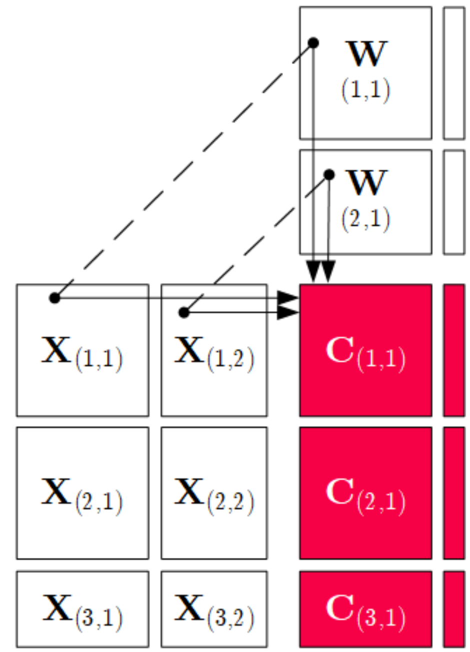
Transposition

`C = t(X)`



Matrix Multiplication

`C = X %*% W`

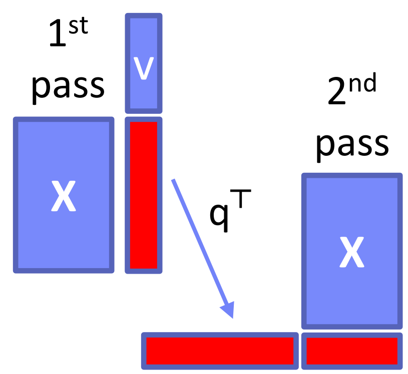


Note: 1:N join

Physical Operator Selection

- **Common Selection Criteria**
  - **Data and cluster characteristics** (e.g., data size/shape, memory, parallelism)
  - **Matrix/operation properties** (e.g., diagonal/symmetric, sparse-safe ops)
  - **Data flow properties** (e.g., co-partitioning, co-location, data locality)
- **#0 Local Operators**
  - SystemML mm, tsmm, mmchain; Samsara/Mllib local
- **#1 Special Operators** (special patterns/sparsity)
  - SystemML **tsmm**, **mapmmchain**; Samsara AtA

`t(X) %*% (X%*%v)`

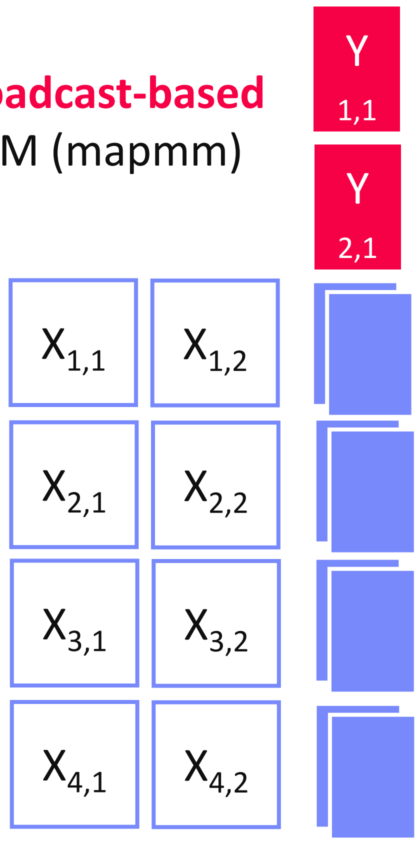


- **#2 Broadcast-Based Operators** (aka broadcast join)
  - SystemML **mapmm**, **mapmmchain**
- **#3 Co-Partitioning-Based Operators** (aka improved repartition join)
  - SystemML **zipmm**; Emma, Samsara OpAtB
- **#4 Shuffle-Based Operators** (aka repartition join)
  - SystemML **cpmm**, **rmm**; Samsara OpAB

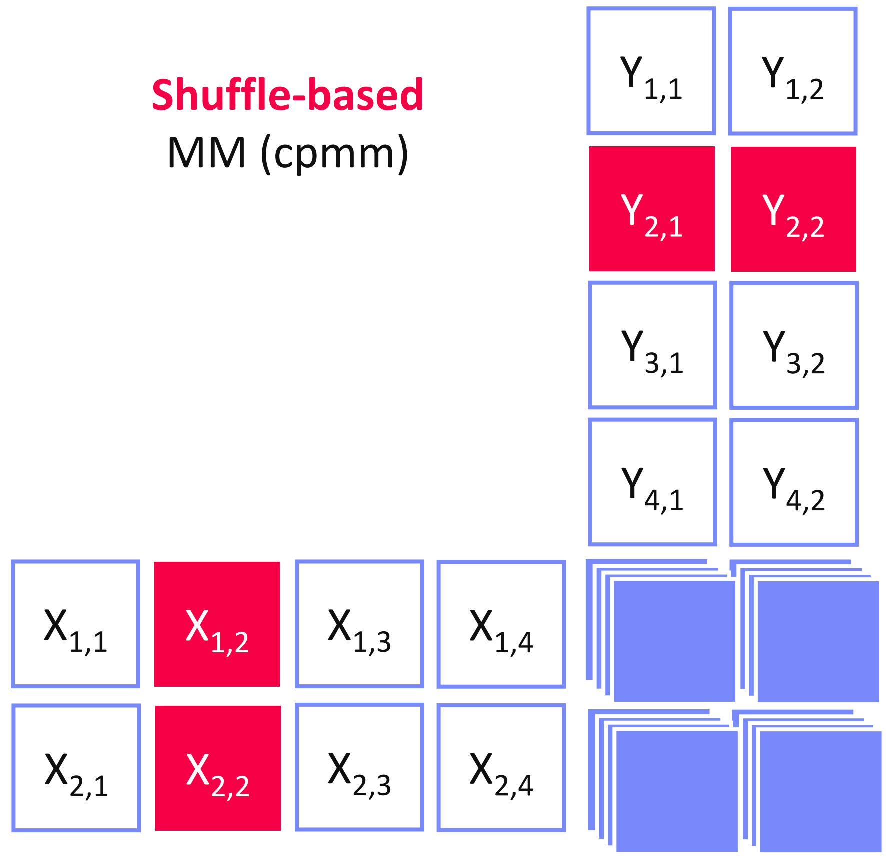
Physical Operator Selection, cont.

- **Examples Distributed MM Operators**

Broadcast-based
MM (mapmm)



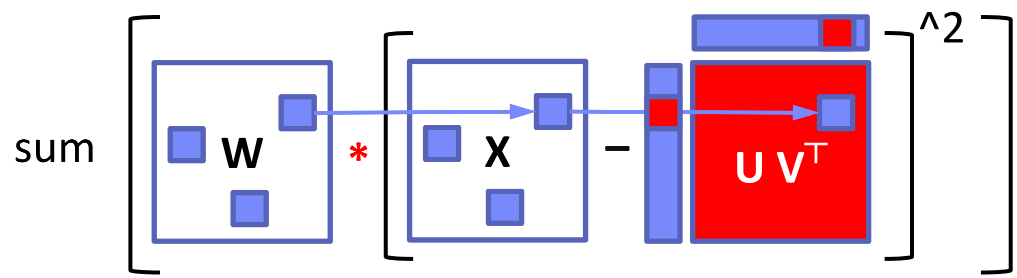
Shuffle-based
MM (cpmm)



Sparsity-Exploiting Operators

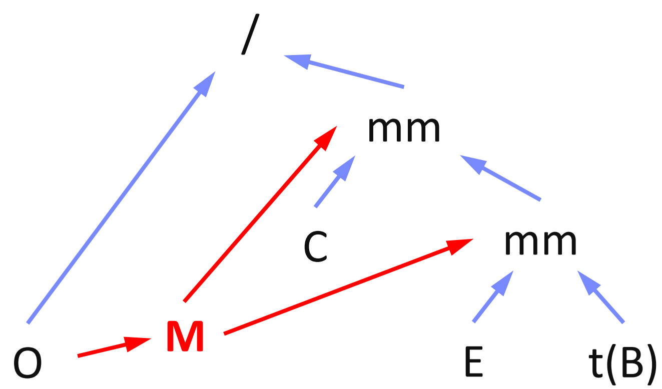
- **Goal:** Avoid dense intermediates and unnecessary computation
- **#1 Fused Physical Operators**
  - E.g., SystemML [PVLDB'16] wsloss, wcemm, wdivmm
  - Selective computation over non-zeros of *"sparse driver"*

```
sum(W * (X – U %*% t(V))^2)
```



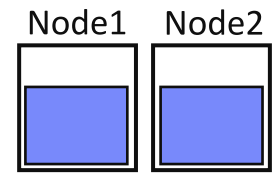
- **#2 Masked Physical Operators**
  - E.g., Cumulon MaskMult [SIGMOD'13]
  - Create mask of *"sparse driver"*
  - Pass mask to single masked matrix multiply operator

```
O / (C %*% E %*% t(B))
```

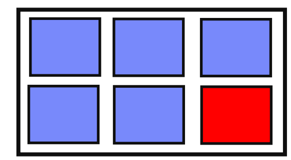


Overview Data Access Methods

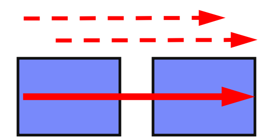
- **#1 (Distributed) Caching**
  - Keep read only feature matrix in (distributed) memory



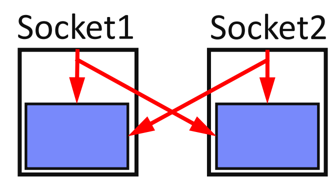
- **#2 Buffer Pool Management**
  - Graceful eviction of intermediates, out-of-core ops



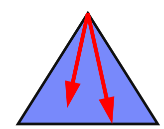
- **#3 Scan Sharing (and operator fusion)**
  - Reduce the number of scans as well as read/writes



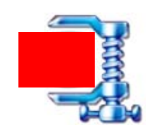
- **#4 NUMA-Aware Partitioning and Replication**
  - Matrix partitioning / replication → data locality



- **#5 Index Structures**
  - Out-of-core data, I/O-aware ops, updates



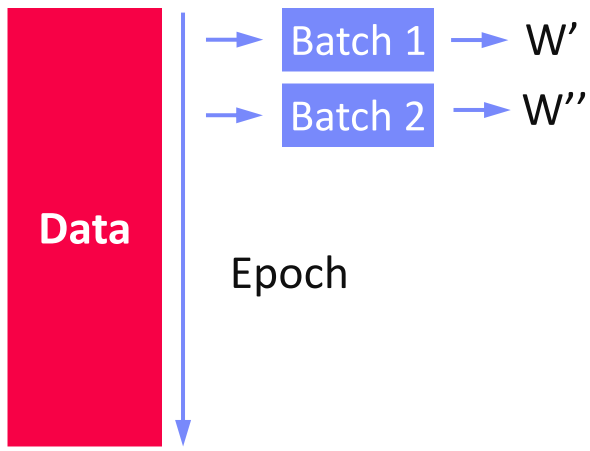
- **#6 Compression**
  - Fit larger datasets into available memory



Distributed Parameter Servers

Background: Mini-batch ML Algorithms

- **Mini-batch ML Algorithms**
  - Iterative ML algorithms, where each iteration only uses a **batch of rows** to make the next model update (in **epochs** over the data)
  - For large and **highly redundant training sets**
  - **Applies to almost all iterative**, model-based ML algorithms (LDA, reg., class., factor., DNN)



- **Statistical vs Hardware Efficiency** (batch size)
  - **Statistical efficiency:** number of accessed data points to achieve certain accuracy
  - **Hardware efficiency:** number of independent computations to achieve high hardware utilization (parallelization at different levels)
  - **Beware higher variance / class skew for too small batches!**

→ **Training Mini-batch ML Algorithms sequentially is hard to scale**

Background: Mini-batch DNN Training (LeNet)

[Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner: Gradient-Based Learning Applied to Document Recognition, **Proc of the IEEE 1998**]

```
# Initialize W1-W4, b1-b4
# Initialize SGD w/ Nesterov momentum optimizer
iters = ceil(N / batch_size)

for( e in 1:epochs ) {
   for( i in 1:iters ) {
      X_batch = X[((i-1) * batch_size) %% N + 1:min(N, beg + batch_size - 1),]
      y_batch = Y[((i-1) * batch_size) %% N + 1:min(N, beg + batch_size - 1),]

      ## layer 1: conv1 -> relu1 -> pool1
      ## layer 2: conv2 -> relu2 -> pool2
      ## layer 3:  affine3 -> relu3 -> dropout
      ## layer 4:  affine4 -> softmax
      outa4 = affine::forward(outd3, W4, b4)
      probs = softmax::forward(outa4)

      ## layer 4:  affine4 <- softmax
      douta4 = softmax::backward(dprobs, outa4)
      [doutd3, dW4, db4] = affine::backward(douta4, outr3, W4, b4)
      ## layer 3:  affine3 <- relu3 <- dropout
      ## layer 2: conv2 <- relu2 <- pool2
      ## layer 1: conv1 <- relu1 <- pool1

      # Optimize with SGD w/ Nesterov momentum W1-W4, b1-b4
      [W4, vW4] = sgd_nesterov::update(W4, dW4, lr, mu, vW4)
      [b4, vb4] = sgd_nesterov::update(b4, db4, lr, mu, vb4)
   }
}
```

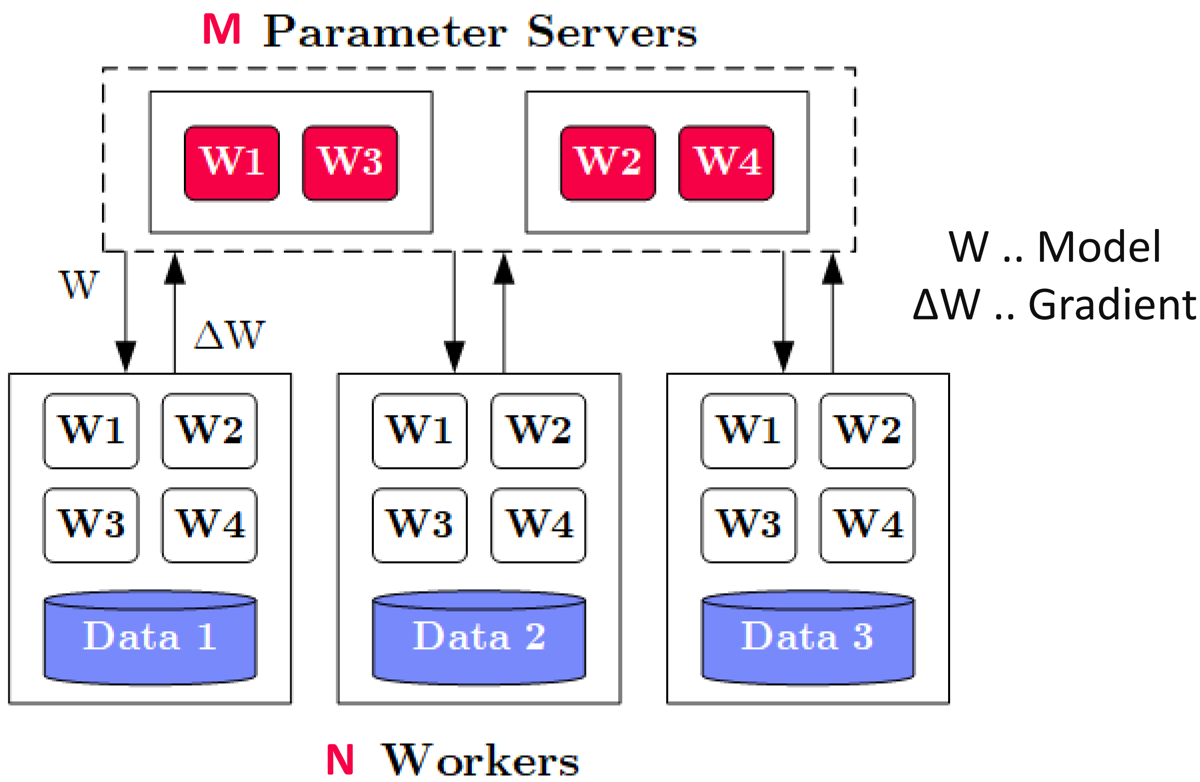
NN Forward Pass

NN Backward Pass
→ Gradients

Model Updates

Overview Data-Parallel Parameter Servers

- **System Architecture**
  - M Parameter Servers
  - N Workers
  - Optional Coordinator



- **Key Techniques**
  - Data partitioning D → workers Di (e.g., disjoint, reshuffling)
  - Updated strategies (e.g., synchronous, asynchronous)
  - Batch size strategies (small/large batches, hybrid methods)

History of Parameter Servers

- **1st Gen: Key/Value**
  - **Distributed key-value store** for parameter exchange and synchronization
  - Relatively high overhead
- **2nd Gen: Classic Parameter Servers**
  - **Parameters as dense/sparse matrices**
  - Different **update/consistency strategies**
  - Flexible configuration and fault tolerance
- **3rd Gen: Parameter Servers w/ improved data communication**
  - Prefetching and range-based pull/push
  - Lossy or lossless compression w/ compensations
- **Examples**
  - TensorFlow, MXNet, PyTorch, CNTK, Petuum

[Alexander J. Smola, Shravan M. Narayanamurthy: An Architecture for Parallel Topic Models. **PVLDB 2010**]

[Jeffrey Dean et al.: Large Scale Distributed Deep Networks. **NIPS 2012**]

[Mu Li et al: Scaling Distributed Machine Learning with the Parameter Server. **OSDI 2014**]

[Jiawei Jiang, Bin Cui, Ce Zhang, Lele Yu: Heterogeneity-aware Distributed Parameter Servers. **SIGMOD 2017**]

[Jiawei Jiang et al: SketchML: Accelerating Distributed Machine Learning with Data Sketches. **SIGMOD 2018**]

Basic Worker Algorithm (batch)

```
for( i in 1:epochs ) {
   for( j in 1:iterations ) {
      params = pullModel(); # W1-W4, b1-b4 lr, mu
      batch = getNextMiniBatch(data, j);
      gradient = computeGradient(batch, params);
      pushGradients(gradient);
   }
}
```

[Jeffrey Dean et al.: Large Scale Distributed Deep Networks. **NIPS 2012**]

Extended Worker Algorithm (nfetch batches)

```
gradientAcc = matrix(0,...);
for( i in 1:epochs ) {
   for( j in 1:iterations ) {
      if( step mod nfetch = 0 )
         params = pullModel();
      batch = getNextMiniBatch(data, j);
      gradient = computeGradient(batch, params);
      gradientAcc += gradient;
      params = updateModel(params, gradients);
      if( step mod nfetch = 0 ) {
         pushGradients(gradientAcc); step = 0;
         gradientAcc = matrix(0, ...);
      }
      step++;
}  }
```

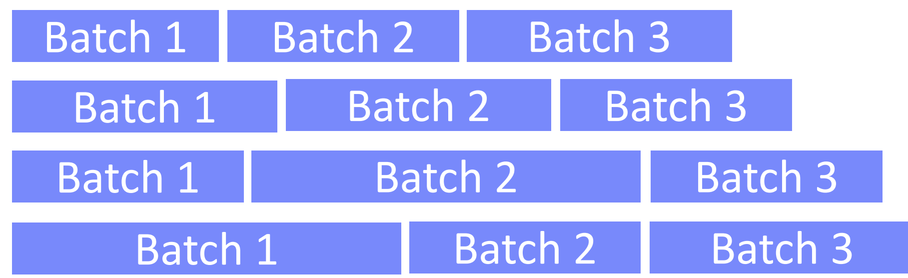
nfetch batches require **local gradient accrual** and **local model update**

[Jeffrey Dean et al.: Large Scale Distributed Deep Networks. **NIPS 2012**]

Update Strategies

- **Bulk Synchronous Parallel (BSP)**
  - Update model w/ accrued gradients
  - Barrier for N workers

- **Asynchronous Parallel (ASP)**
  - Update model for each gradient
  - No barrier



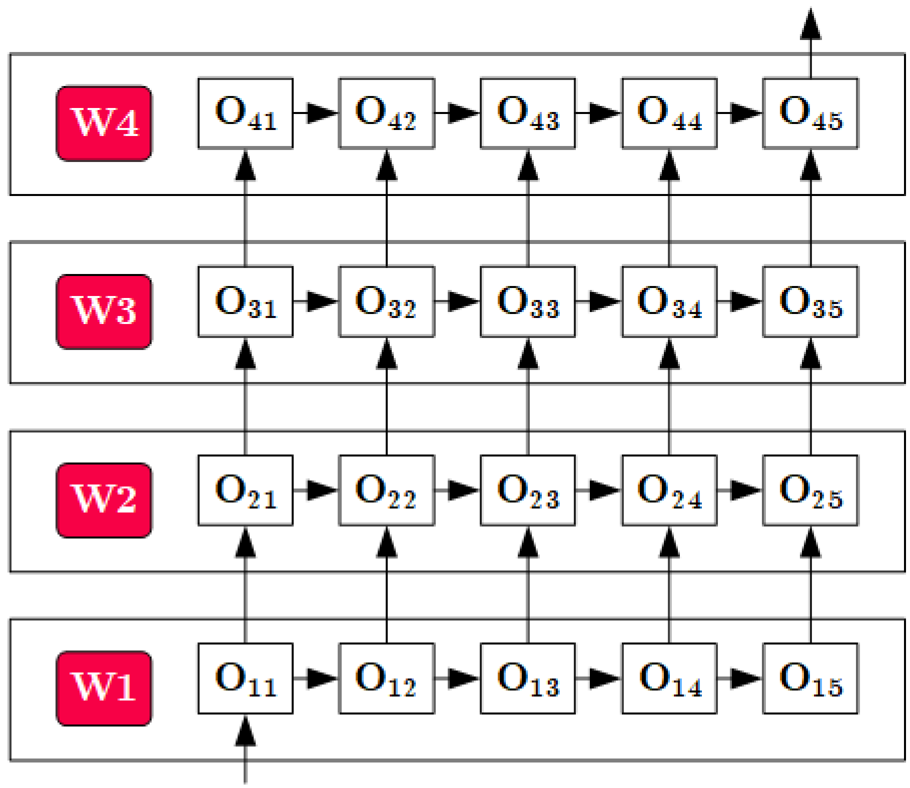
but, stale model updates

- **Synchronous w/ Backup Workers**
  - Update model w/ accrued gradients
  - Barrier for N of N+b workers

[Martín Abadi et al: TensorFlow: A System for Large-Scale Machine Learning. **OSDI 2016**]

Overview Model-Parallel Execution

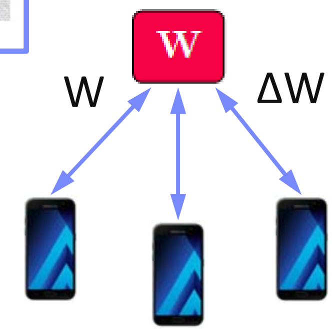
- **System Architecture**
  - Nodes act as workers and parameter servers
  - Data Transfer for boundary-crossing data dependencies
- **Pipeline Parallelism**



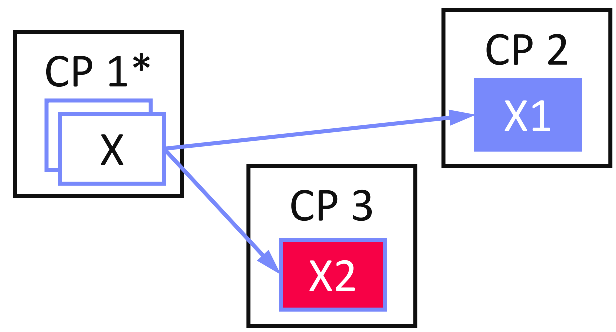
Workers w/ disjoint network/model partitions

Federated ML

[Keith Bonawitz et al.: Towards Federated Learning at Scale: System Design. SysML 2019]



- **Motivation Federated ML**
  - Learn model **w/o central data consolidation**
  - **Privacy + data/power caps** vs **personalization and sharing**
- **Data Ownership ➔ Federated ML in the enterprise** (machine vendor – middle-person – customer equipment)
- **Federated ML Architecture**
  - Multiple control programs w/ single master
  - Federated tensors (metadata handles)
  - **Federated instructions** and **parameter server**



- **ExDRa Project** (Exploratory Data Science over Raw Data)
  - **Basic approach:** Federated ML + ML over raw data
  - System infra, integration, data org & reuse, Exp DB, geo-dist.

Bundesministerium Verkehr, Innovation und Technologie

FFG Forschung wirkt.

Gefördert im Programm "IKT der Zukunft" vom Bundesministerium für Verkehr, Innovation, und Technologie (BMVIT)

SIEMENS

TU Graz

Technische Universität Berlin

Summary and Q&A

- **Landscape of ML Systems**
- **Distributed Linear Algebra**
- **Distributed Parameter Servers**

Thanks

(please, participate in the **course evaluation**)

- **Projects and Exercises**
  - 13 projects + 3 exercises → **grace period: end of Feb**
  - In case of problem: ask for help + problem scaling possible

- **Oral Exams** **[Feb 3 - Feb 5]** (11 participants?)