Data Integration and Large Scale Analysis
12 Distributed ML Systems

Shafaq Siddiqi

Graz University of Technology, Austria
Slides credit: Matthias Boehm

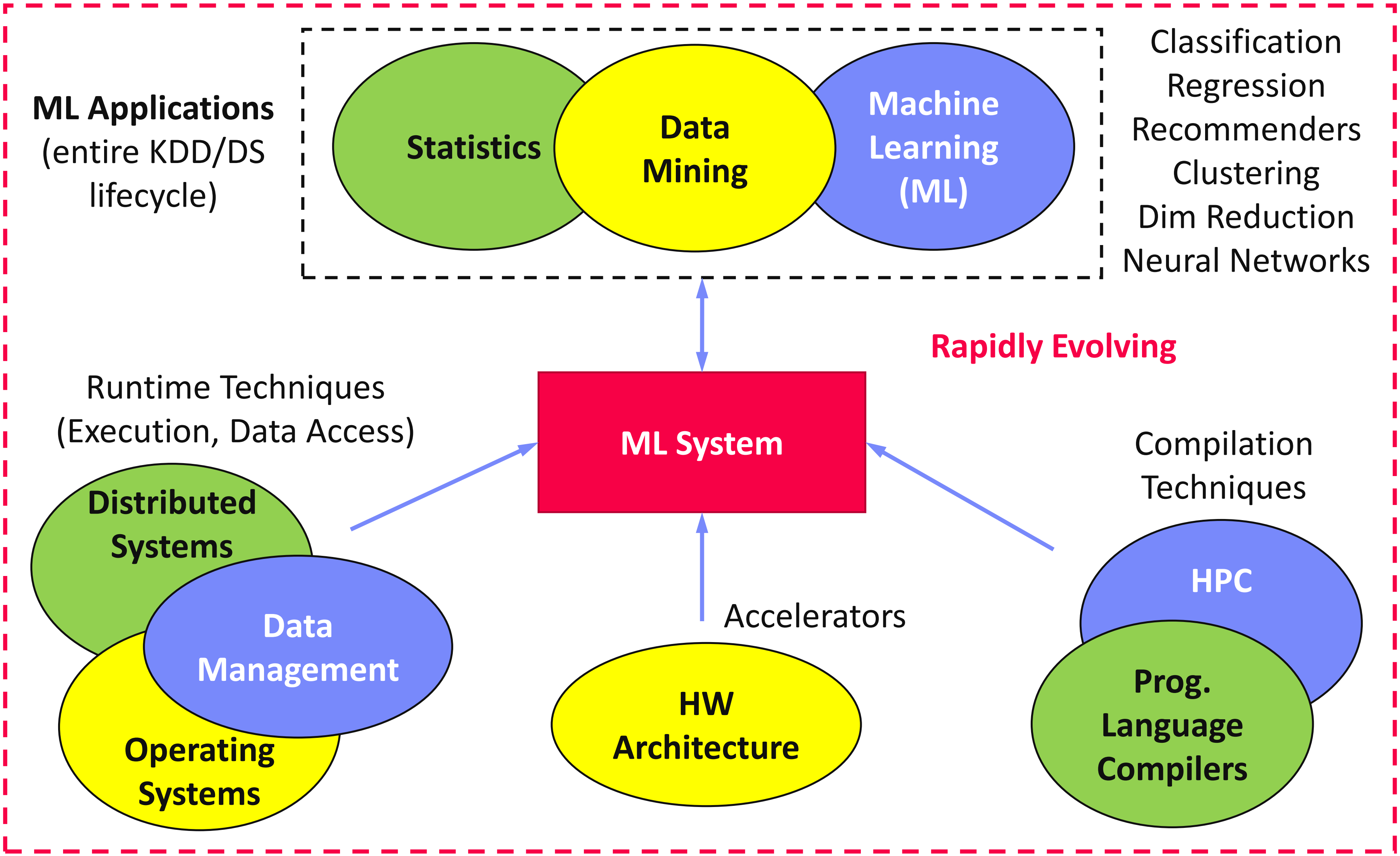
Last update: Jan 27, 2023

Agenda

- **Landscape of ML Systems**
- **Distributed Parameter Servers**
- **Q&A and Exam Preparation**

Landscape of ML Systems

What is an ML System?

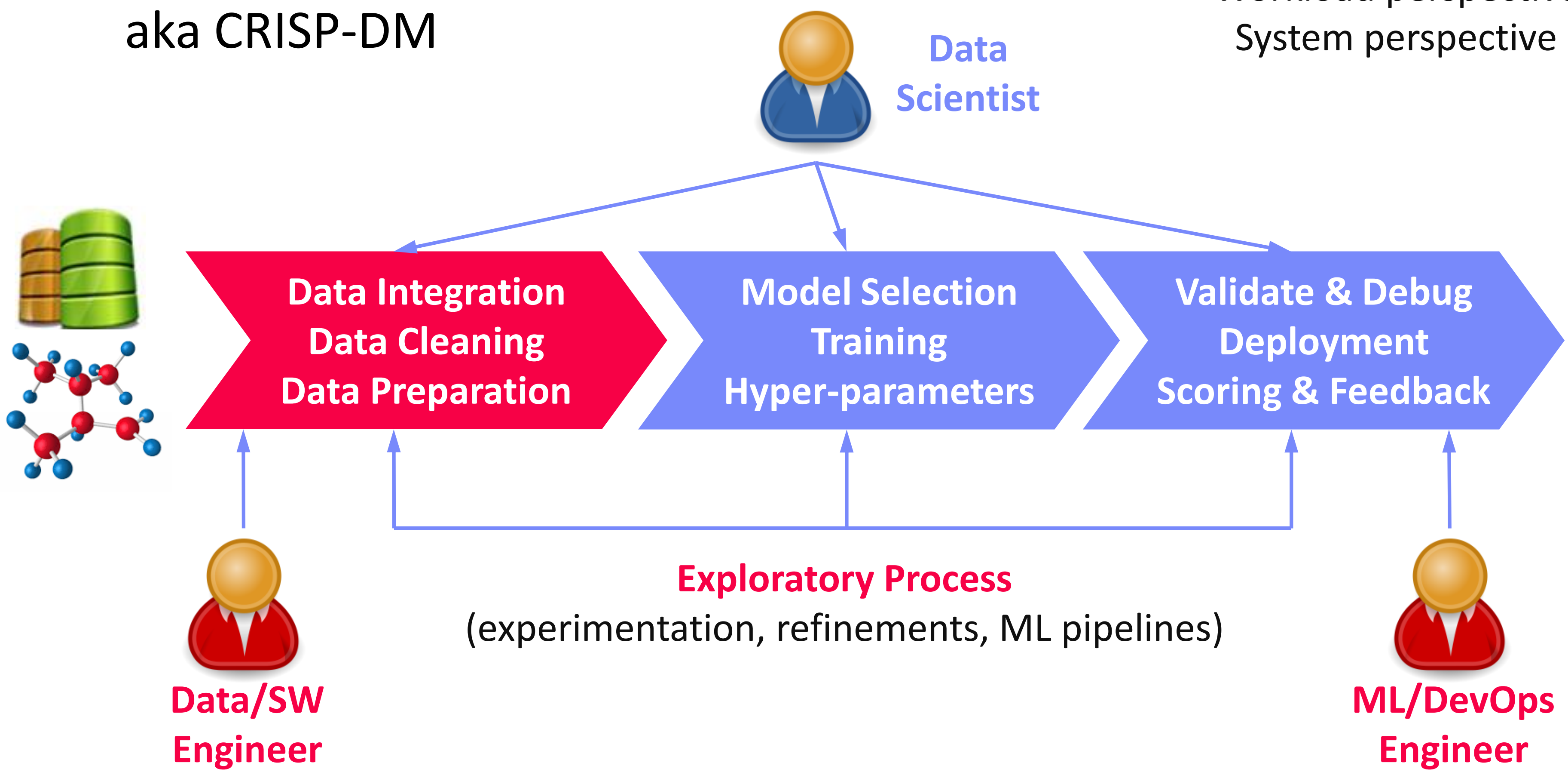


The Data Science Lifecycle

aka KDD Process

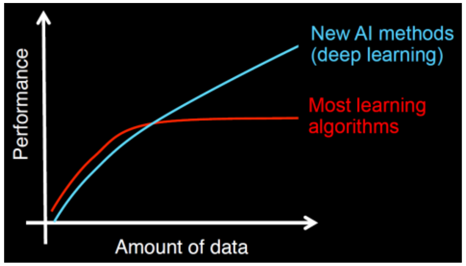
aka CRISP-DM

Data-centric View:
Application perspective
Workload perspective
System perspective



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

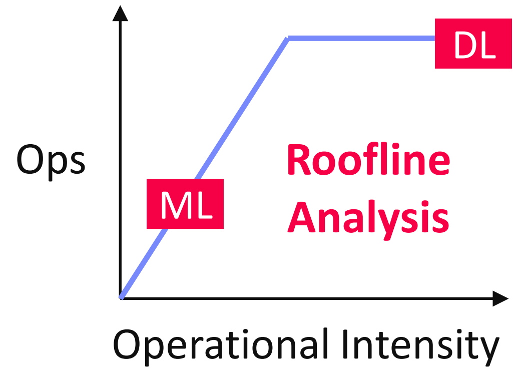
ML Apps & Algorithms	Supervised, unsupervised, RL linear algebra, libs, AutoML
Language Abstractions	Eager interpretation, lazy evaluation, prog. compilation
Fault Tolerance	Approximation, lineage, checkpointing, checksums, ECC
Execution Strategies	Local, distributed, cloud (data, task, parameter server)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



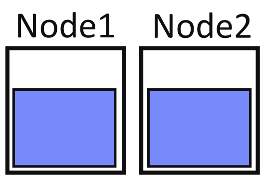
- **Graphics Processing Units (GPUs)**
  - Extensively used for deep learning training and scoring
  - NVIDIA Volta: “tensor cores” for 4x4 mm → 64 2B FMA instruction
- **Field-Programmable Gate Arrays (FPGAs)**
  - Customizable HW accelerators for prefiltering, compression, DL
  - Examples: Microsoft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

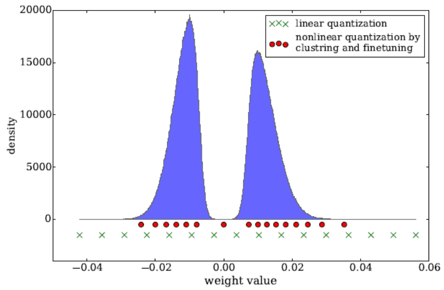
`vec(Berlin) – vec(Germany) + vec(France) ≈ vec(Paris)`

- **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)

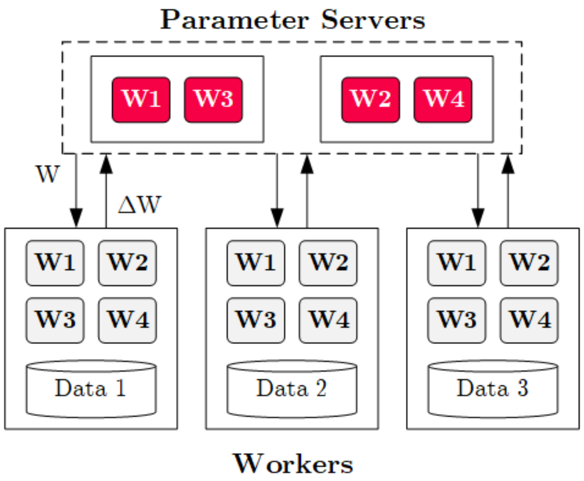
[**Credit:** Song Han'16]



Execution Strategies

Apps
Lang
Faults
Exec
Data
HW

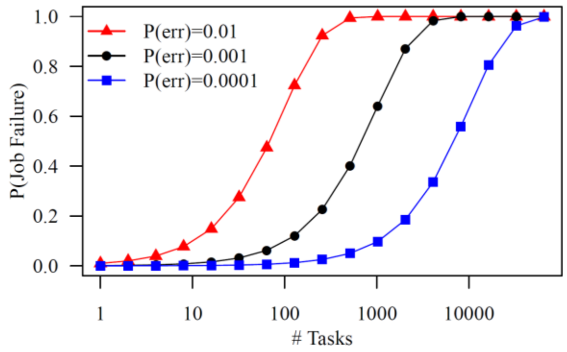
- **Batch Algorithms: Data and Task Parallel**
  - Data-parallel operations
  - Different physical operators
- **Mini-Batch Algorithms: Parameter Server**
  - Data-parallel and model-parallel PS
  - Update strategies (e.g., async, sync, backup)
  - Data partitioning strategies
  - Federated ML (trend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



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

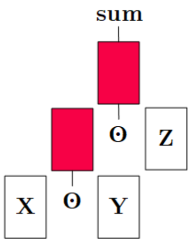
Language Abstractions

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

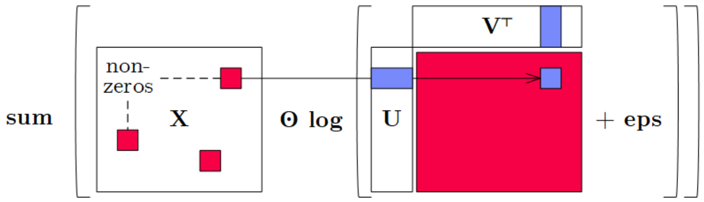
Apache SystemDS

- **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: SystemDS, 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

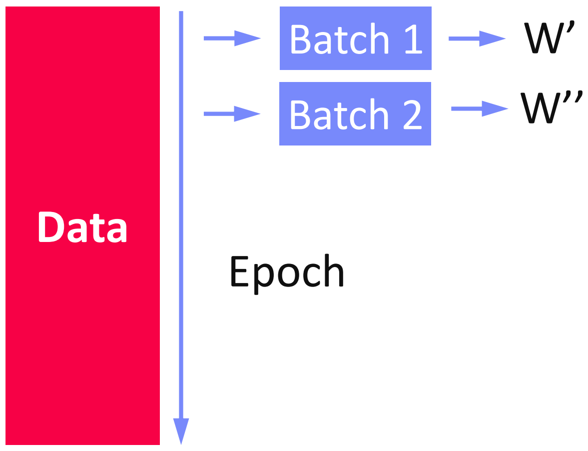
Tensors

Frames

Distributed 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** or w/ **sampling**)
  - For large and **highly redundant training sets**
  - **Applies to almost all iterative**, model-based ML algorithms (LDA, reg., class., factor., DNN)
  - **Stochastic Gradient Descent** (SGD)



- **Statistical vs Hardware Efficiency** (batch size)
  - **Statistical efficiency:** # 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)
   }
}
```

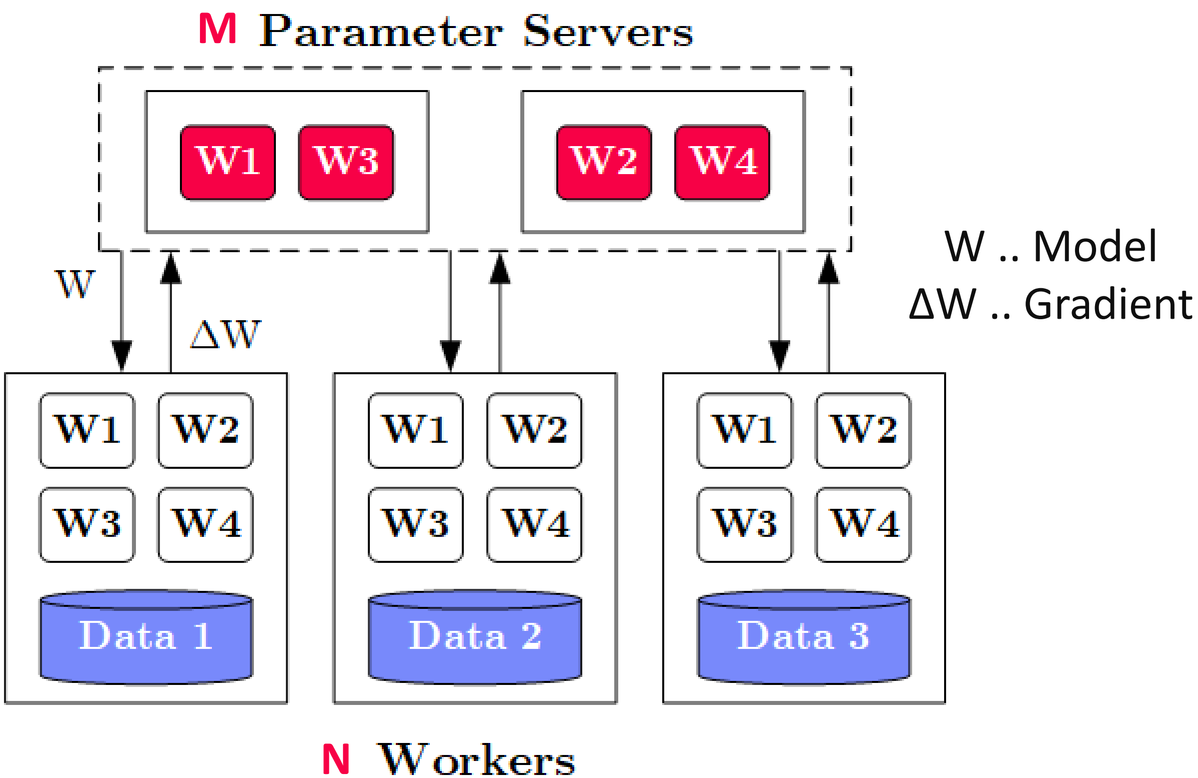
NN Forward Pass

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++;
} }
```

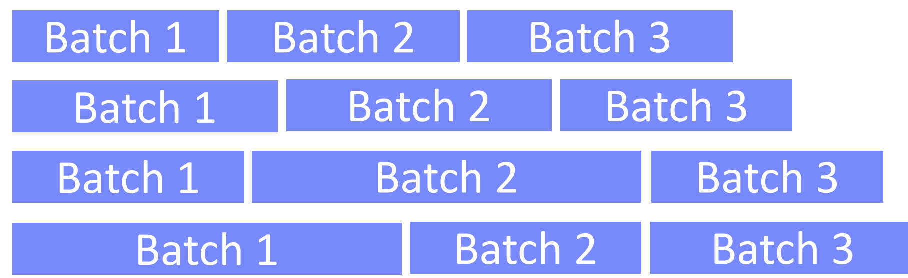
nfetch batches require **local gradient accrual** and **local model update**

[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**]

Q&A and Exam Preparation

Selected Example Questions

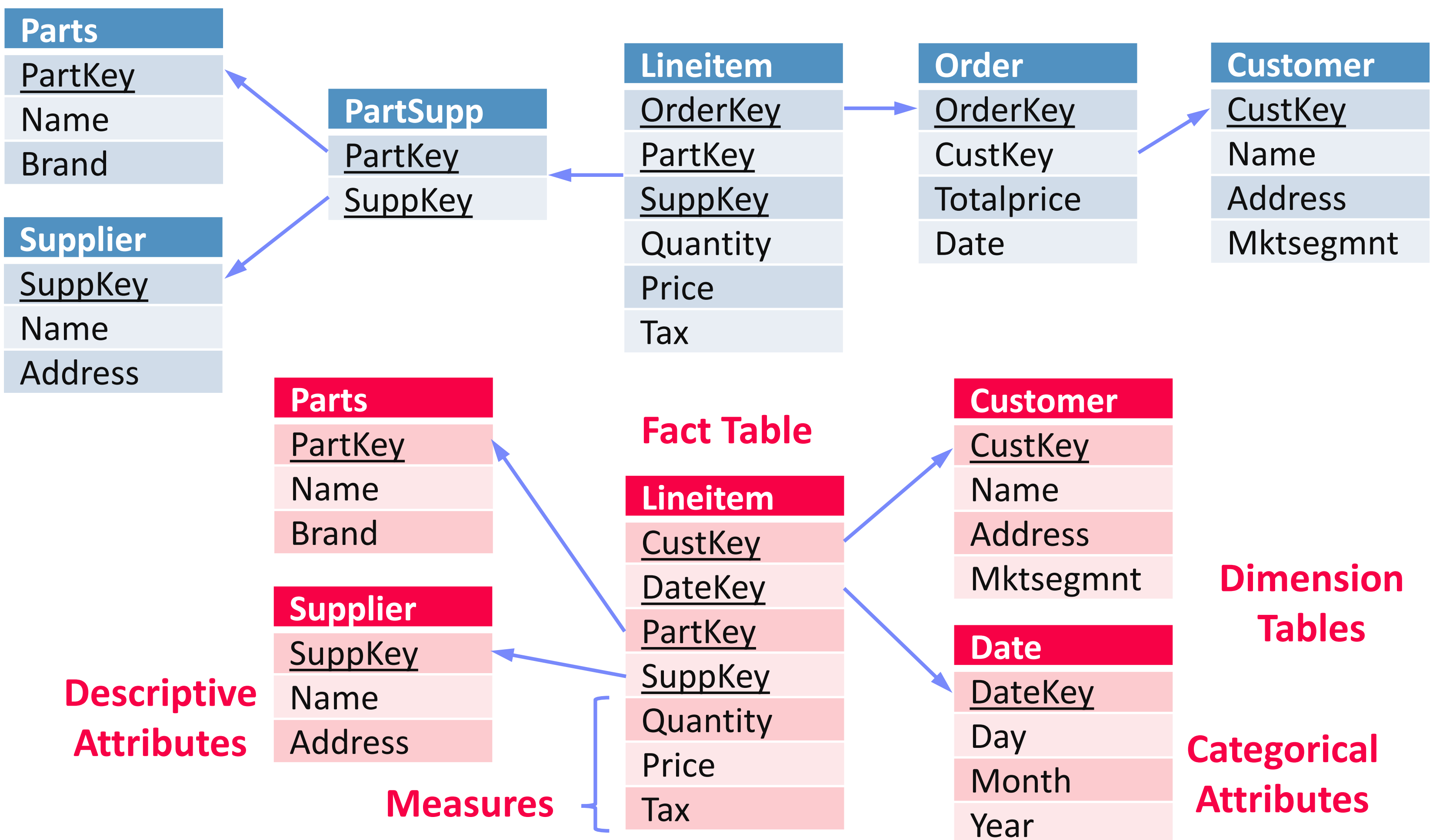
Multiple choice question (40/100)

- **Given the dataset identify the type of missingness.**
  a. Missing Completely at Random (MCAR)
  b. Missing at Random (MAR)
  c. Missing Not at Random (MNAR)
  d. All of above

Profession	Salary
A	2000
C	5000
B	2300
C	?
C	?
A	2000

Data Warehousing

- **Given the following normalized schema, create a** **Star Schema** **that covers all information. Annotate the key concepts.** [15/100 points]



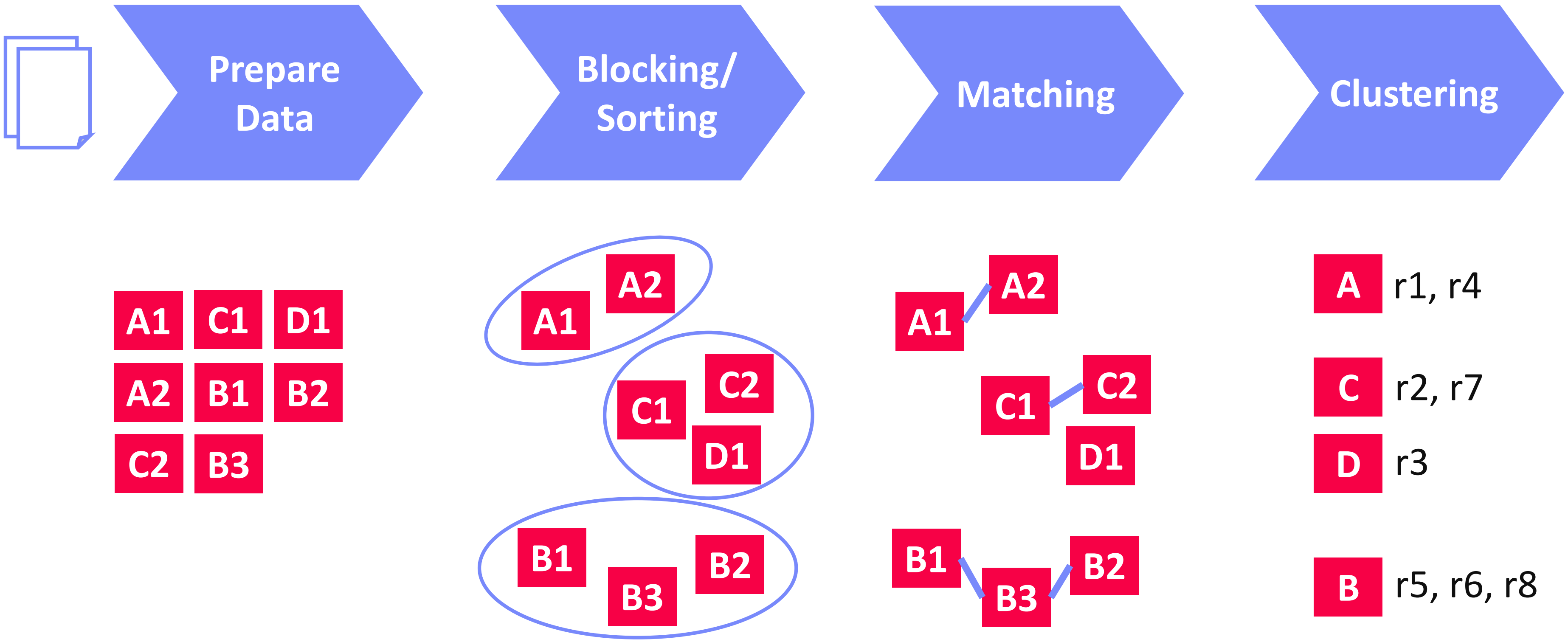
Message-oriented Middleware

- **Describe the Message Delivery Guarantees At-Most-Once, At-Least-Once and Exactly-Once, and indicate which of them require persistent storage before sending.** [6/100 points]

Name	Description	Storage
At-Most-Once	Send and forget, never sent message twice (even on failures)	No
At-Least-Once	Store and forward, replay stream from (acknowledged) checkpoint	Yes
Exactly-Once	Store and forward, replay stream from (acknowledged) checkpoint, transactional delivery	Yes

Schema Matching / Entity Linking

- **Explain the phases of a typical Entity Resolution Pipeline with example techniques for the individual phases.** [20/100 points]



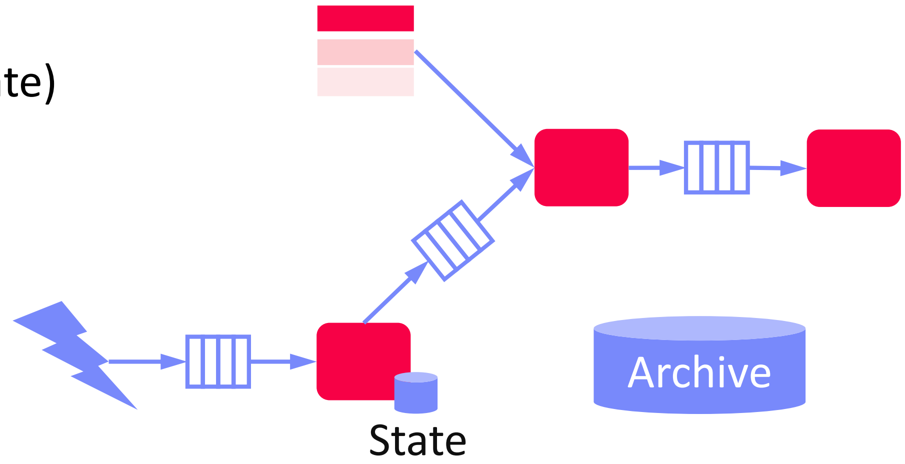
Data Parallel Computation / Stream Mining

- **Assume three nodes with CPU and memory capacity N1 (32 cores, 64 GB), N2 (16 cores, 64 GB), N3 (64 cores, 128 GB) and a stream of resource requests R1 . . . R7. Schedule these requests to available resources (assign requests to nodes) in order to maximize the number of fulfilled requests. (4 points)**
- **R1: (30 cores, 8 GB)**
- **R2: (6 cores, 32 GB)**
- **R3: (8 cores, 64 GB)**
- **R4: (10 cores, 32 GB)**
- **R5: (8 cores, 32 GB)**
- **R6: (16 cores, 64 GB)**
- **R7: (16 cores, 16 GB)**

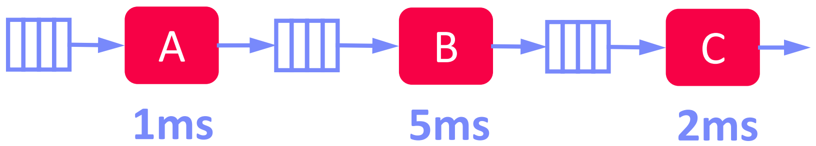
	N1 (32/64)	N2 (16/64)	N3 (64/128)
requests	R6	R2, R4	R1, R3, R5, R7
utilization	16/64	16/64	62/120

Stream Processing

- **Describe the concept of Continuous Queries and a Basic System Infrastructure to process them over incoming data streams.** [8/100 points]
  - Deployed Data flow graphs
  - **Nodes:** asynchronous ops (w/ state) (e.g., separate threads / queues)
  - **Edges:** data dependencies (tuple/message streams)
  - **Push model:** data production controlled by source



- **Given the continuous query A-B-C, what are the resulting perf characteristics?** [4 points]
  - Max throughput
  - Min tuple latency



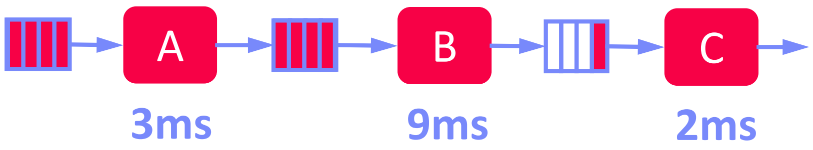
$1/\max(C(op_i))$ = 1/5 tuples/ms = 200 tuples/s

$\mathrm{sum}(C(op_i))$ = 1ms + 5ms + 2ms = 8ms

Stream Processing, cont.

- **Describe the three classes of techniques for handling overload situations in continuous queries?** [6/100 points]

- **#1 Back Pressure**
  - Graceful handling of overload w/o data loss
  - Slow down sources
  - E.g., blocking queues



Self-adjusting operator scheduling
Pipeline runs at rate of slowest op

- **#2 Load Shedding**
  - #1 Random-sampling-based load shedding
  - #2 Relevance-based load shedding
  - #3 Summary-based load shedding (synopses)
- **#3 Distributed Stream Processing** (see last part)
  - Data flow partitioning (distribute the query)
  - Key range partitioning (distribute the data stream

Summary and Q&A

- **Landscape of ML Systems**
- **Distributed Parameter Servers**
- **Q&A and Exam Preparation**

Thanks

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

- **#2 Course Evaluation and Exam**
  - Evaluation period: **Jan 15 – Feb 15** (1/98)
  - Exam date: **Feb 10**