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
BMK endowed chair for Data Management

Last update: Jan 24, 2022

Announcements/Org

- **#1 Video Recording**
  - Link in **TUbe** & **TeachCenter** (lectures will be public)
  - Optional attendance (independent of COVID)
  - **Virtual lectures** (recorded) until end of the semester
    https://tugraz.webex.com/meet/m.boehm
- **#2 Programming Projects/Exercises**
  - Deadline Reminder: **Jan 21 11.59pm → Jan 28 11.59pm**
    (max 7 late days, with (2*late_days) point deduction)
  - Exercise submission in **TeachCenter**, projects via pull requests
- **#3 Course Evaluation and Exam**
  - Evaluation period: **Jan 01 – Feb 15**
  - Exam date: **Feb 04, 3pm** (90+min written exam)
  - Doodle for registered oral exam participants

16 Ex.
15 Proj.
(xxx+27 students)

Data Management Courses

Architecture of Database Systems (ADBS, WS)

DB system internals
+ prog. project

Architecture of ML Systems (AMLS, SS)

ML system internals

Prog. projects in **SystemDS**
[github.com/apache/systemds]

Master

Bachelor

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

Distributed
Data Management

Intro to Scientific Writing (WS)

Data Management / Databases (DM, SS+WS)

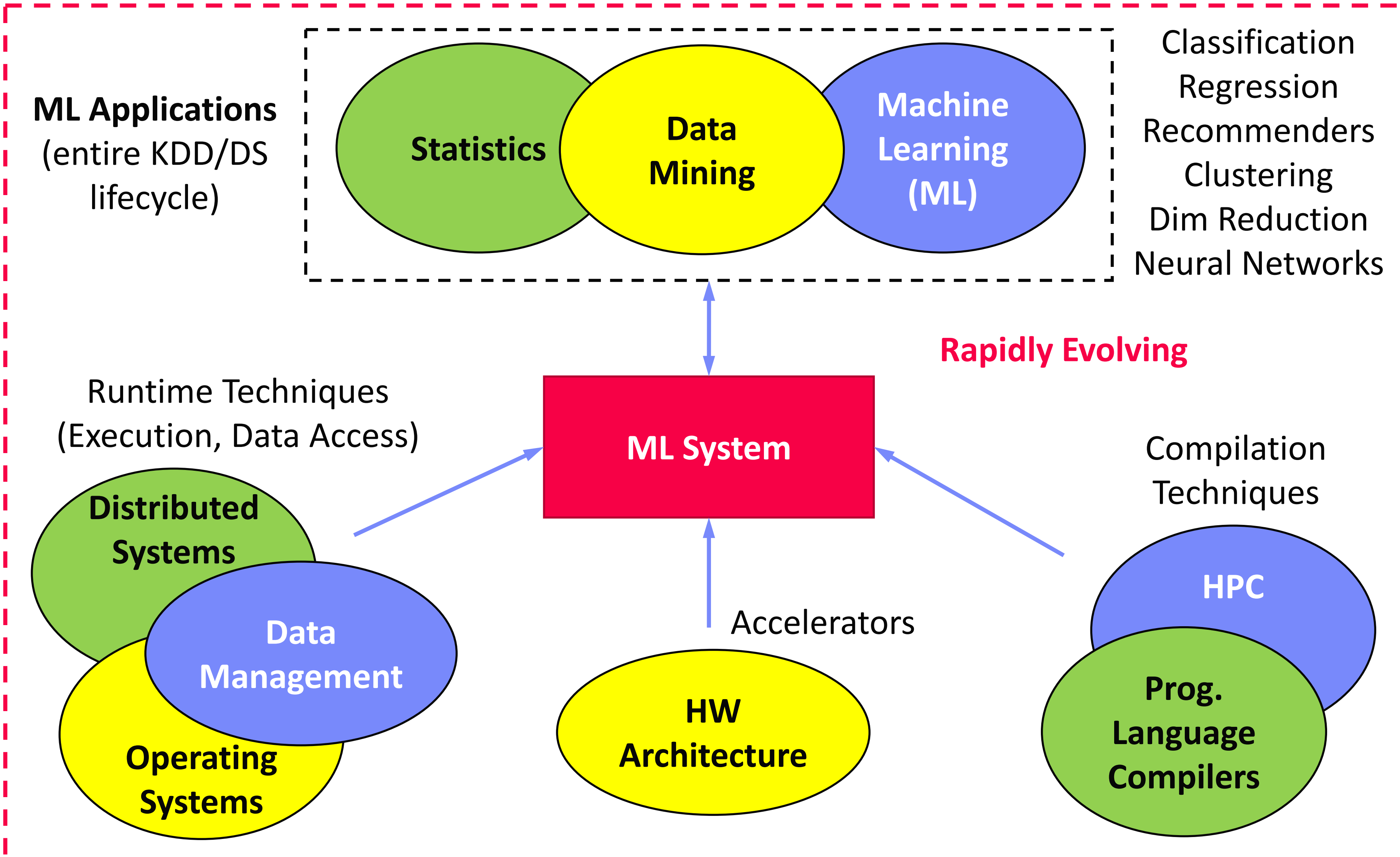
Data management from
user/application perspective

Agenda

- **Landscape of ML Systems**
- **Distributed Linear Algebra**
- **Distributed Parameter Servers**
- **Q&A and Exam Preparation (New)**

Landscape of ML Systems

What is an ML System?

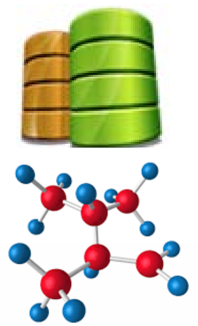


The Data Science Lifecycle

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)

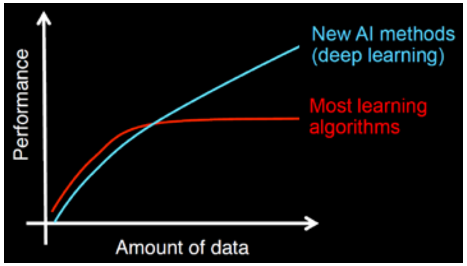
ML/DevOps Engineer

Key observation: SotA data integration/cleaning based on ML

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

Usage ← Model

- **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

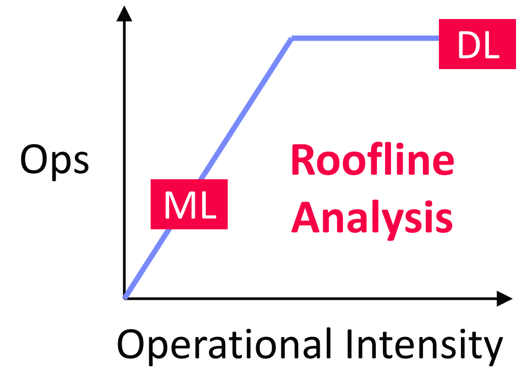
ML Apps & Algorithms	Supervised, unsupervised, RL linear algebra, libs, AutoML
Language Abstractions	Eager interpretation, lazy evaluation, prog. compilation
Fault Tolerance	Approximation, lineage, checkpointing, checksums, ECC
Execution Strategies	Local, distributed, cloud (data, task, parameter server)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



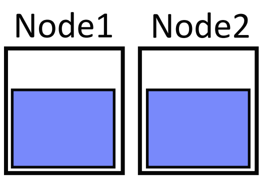
- **Graphics Processing Units (GPUs)**
  - Extensively used for deep learning training and scoring
  - NVIDIA Volta: "tensor cores" for 4x4 mm → 64 2B FMA instruction
- **Field-Programmable Gate Arrays (FPGAs)**
  - Customizable HW accelerators for prefiltering, compression, DL
  - Examples: Microsoft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

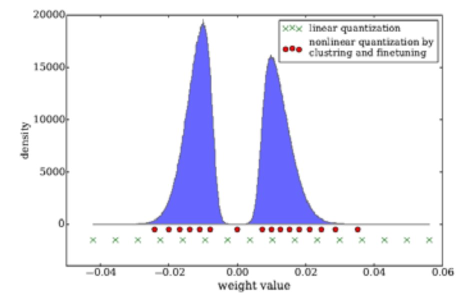
`vec(Berlin) – vec(Germany) + vec(France) ≈ vec(Paris)`

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



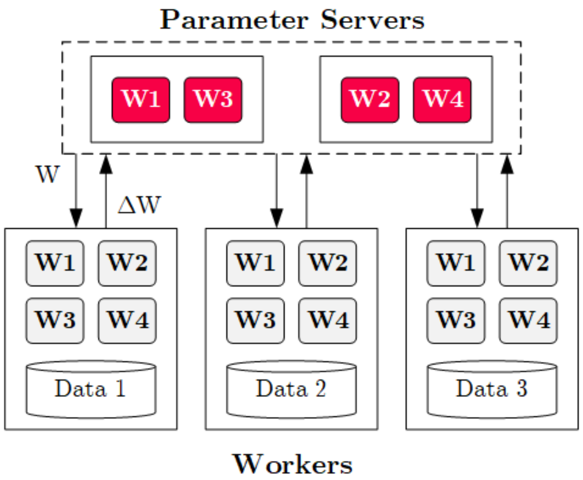
- **Lossy Compression ➔ Acc/Perf-Tradeoff**
  - Sparsification (reduce non-zero values)
  - Quantization (reduce value domain), learned
  - New data types: Intel Flexpoint (mantissa, exp)

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



Execution Strategies

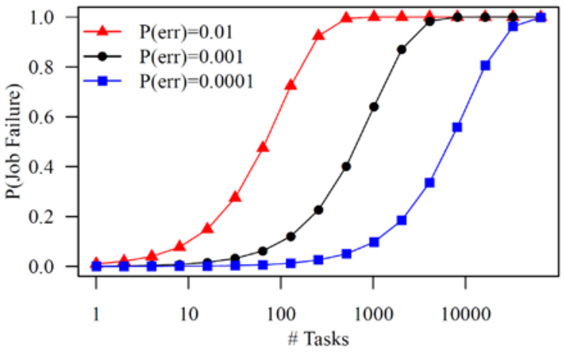
- **Batch Algorithms: Data and Task Parallel**
  - Data-parallel operations
  - Different physical operators
- **Mini-Batch Algorithms: Parameter Server**
  - Data-parallel and model-parallel PS
  - Update strategies (e.g., async, sync, backup)
  - Data partitioning strategies
  - Federated ML (trend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



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

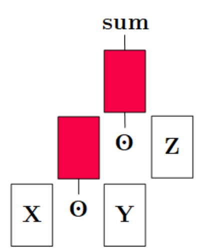
Language Abstractions

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

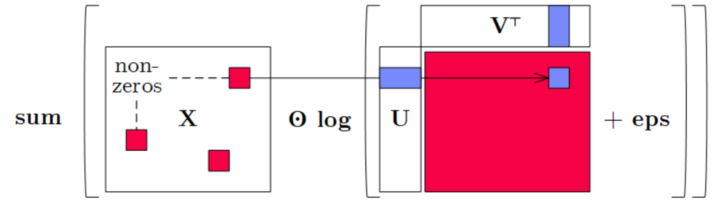
Apache SystemDS

- **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 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

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], **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)

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 (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))
...
```

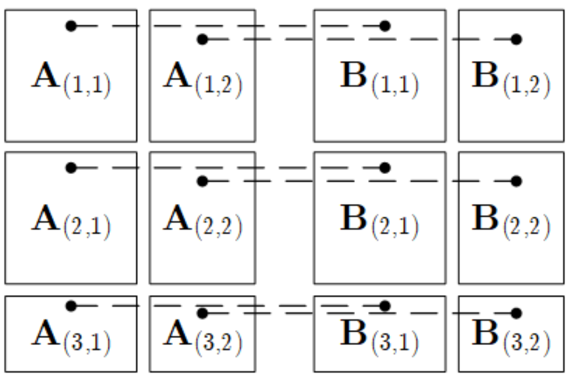
```
opt = keras.optimizers.rmsprop(
  lr=0.0001, decay=1e-6)

# 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)

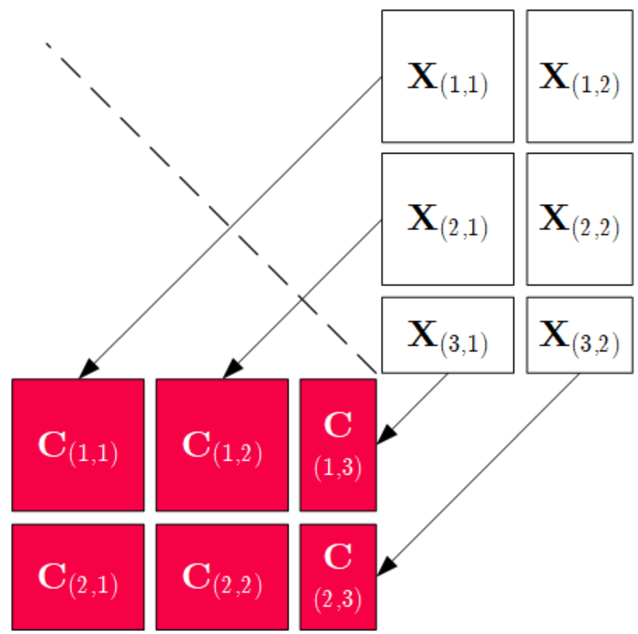
`C = A * B`



Note: also with
row/column vector rhs

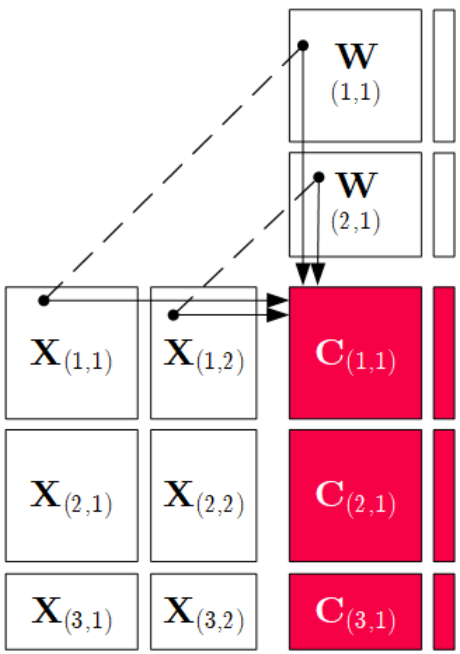
Transposition

`C = t(X)`



Matrix Multiplication

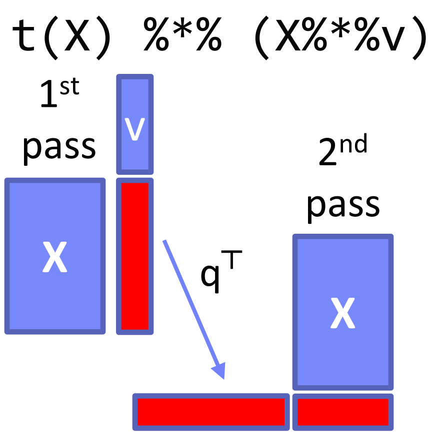
`C = X %*% W`



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

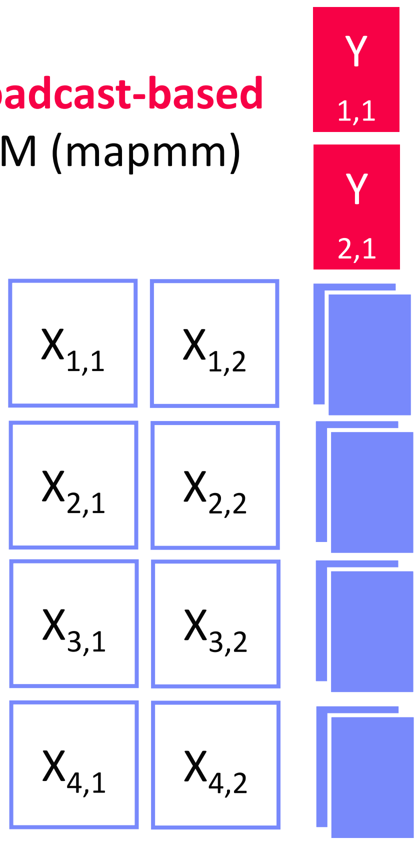


- **#2 Broadcast-Based Operators** (aka broadcast join)
  - SystemML **mapmm**, **mapmmchain**
- **#3 Co-Partitioning-Based Operators** (aka improved repartition join)
  - SystemML **zipmm**; Emma, Samsara OpAtB
- **#4 Shuffle-Based Operators** (aka repartition join)
  - SystemML **cpmm**, **rmm**; Samsara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

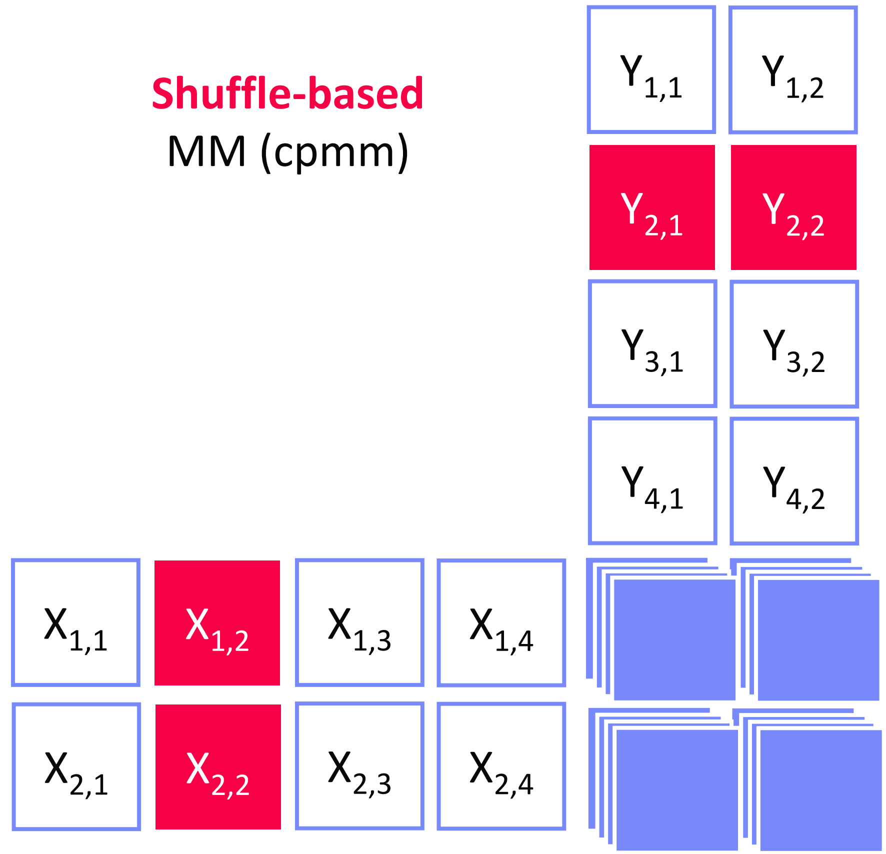
Physical Operator Selection, cont.

- **Examples Distributed MM Operators**

Broadcast-based MM (mapmm)



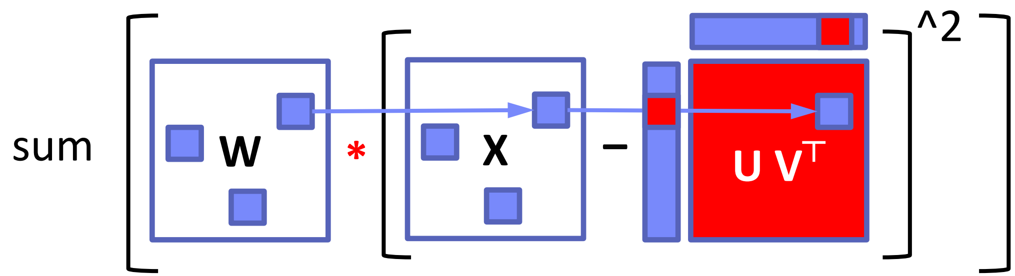
Shuffle-based MM (cpmm)



Sparsity-Exploiting Operators

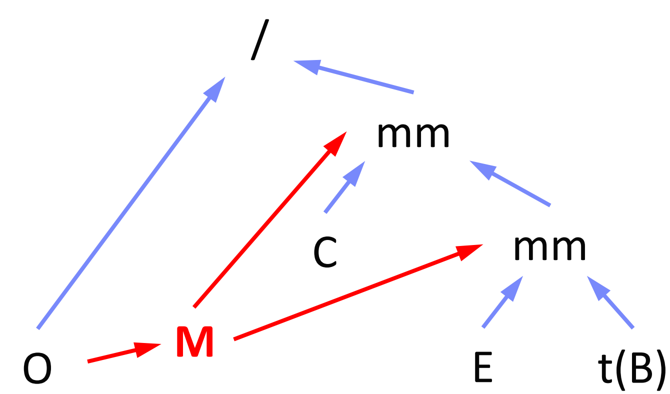
- **Goal:** Avoid dense intermediates and unnecessary computation
- **#1 Fused Physical Operators**
  - E.g., SystemML [PVLDB'16] wsloss, wcemm, wdivmm
  - Selective computation over non-zeros of **"sparse driver"**

`sum(W * (X – U %*% t(V))^2)`



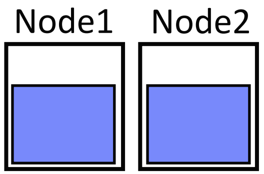
- **#2 Masked Physical Operators**
  - E.g., Cumulon MaskMult [SIGMOD'13]
  - Create mask of **"sparse driver"**
  - Pass mask to single masked matrix multiply operator

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

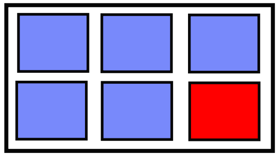


Overview Data Access Methods

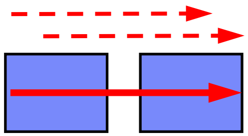
- **#1 (Distributed) Caching**
  - Keep read only feature matrix in (distributed) memory



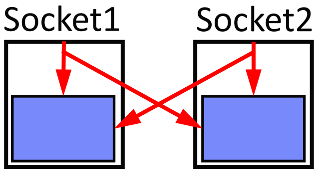
- **#2 Buffer Pool Management**
  - Graceful eviction of intermediates, out-of-core ops



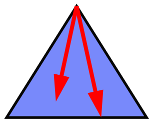
- **#3 Scan Sharing (and operator fusion)**
  - Reduce the number of scans as well as read/writes



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



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

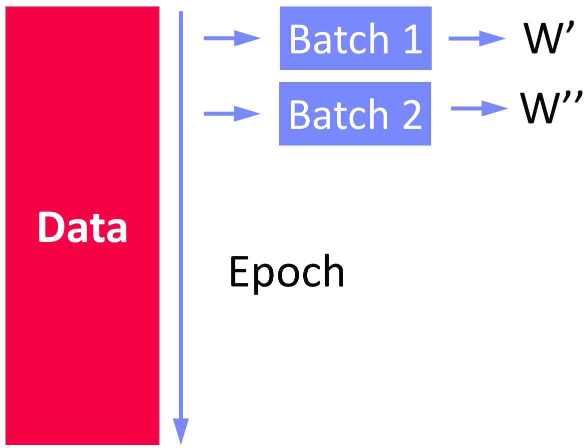


- **#6 Compression**
  - Fit larger datasets into available memory

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** 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)
   }
}
```

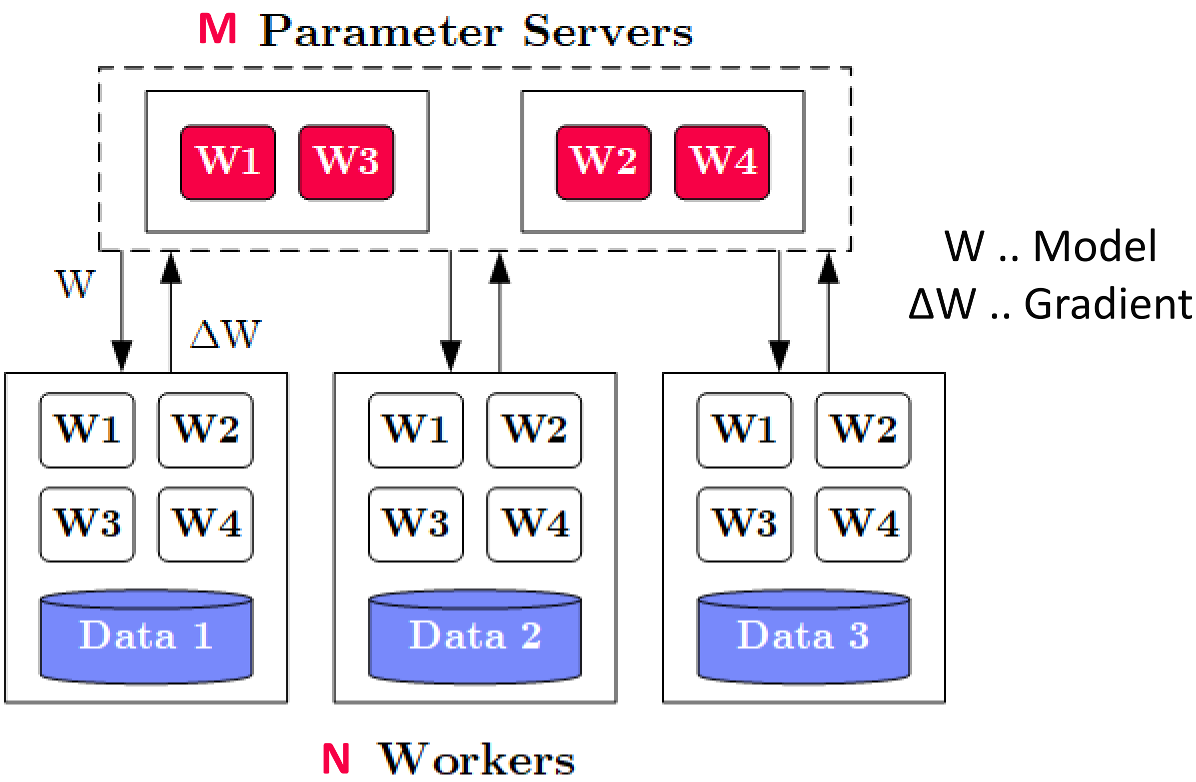
NN Forward Pass

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

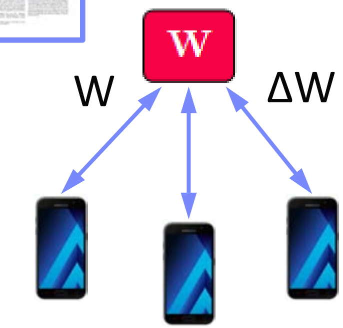
but, stale model updates

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

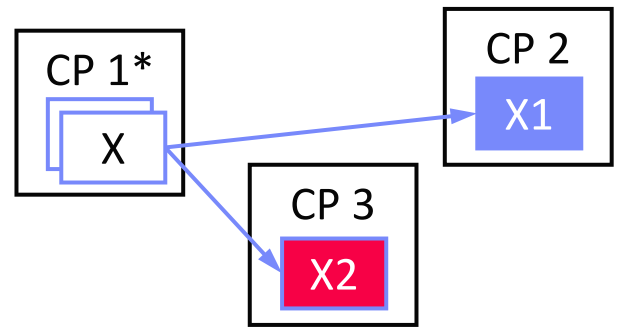
[Martín Abadi et al: TensorFlow: A System for Large-Scale Machine Learning. **OSDI 2016**]

Federated ML

[Keith Bonawitz et al.: Towards Federated Learning at Scale: System Design. **MLSys 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.

SIEMENS

Bundesministerium Verkehr, Innovation und Technologie

FFG Forschung wirkt.

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

TU Graz

Technische Universität Berlin

Q&A and Exam Preparation

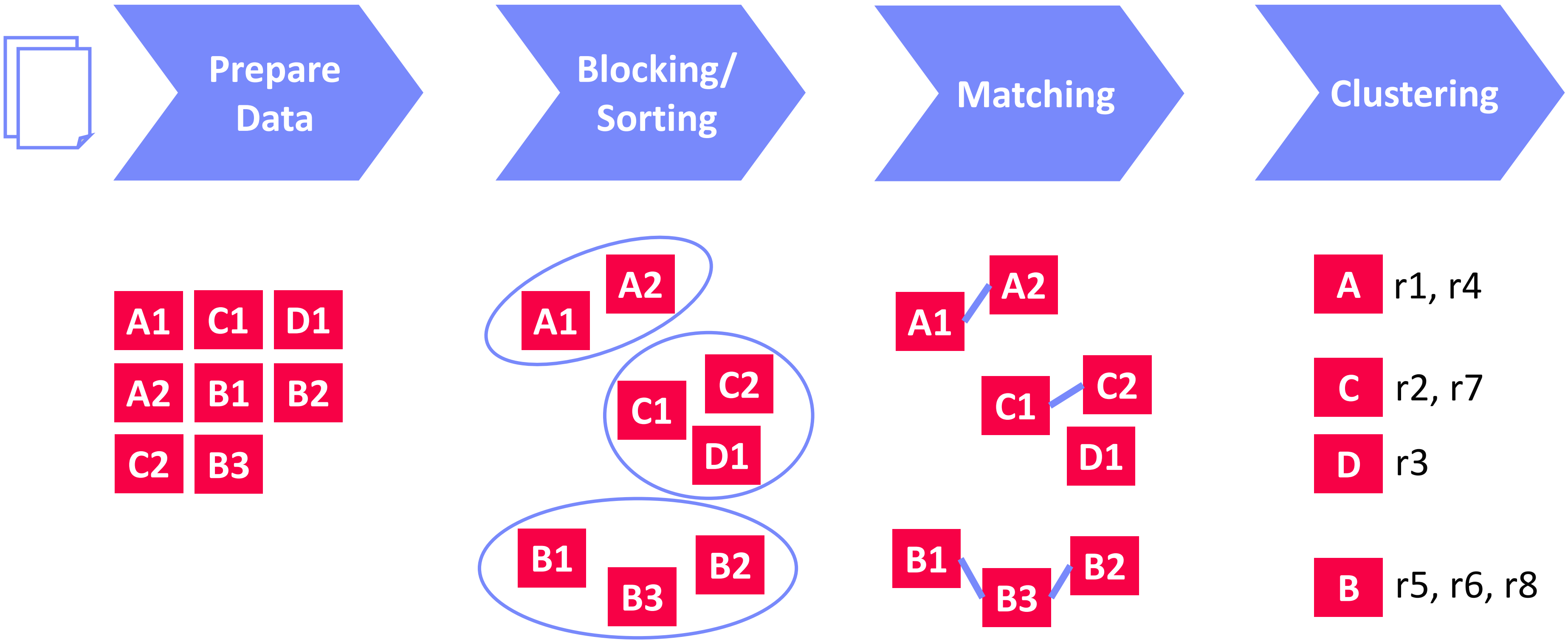
Example Exam DIA WS20/21 v2
(90min for 100/100 points)

https://mboehm7.github.io/teaching/ws2021_dia/ExamDIA_v1.pdf

https://mboehm7.github.io/teaching/ws2021_dia/ExamDIA_v2.pdf

Task 1: Entity Resolution

- **a) Explain the phases of a typical entity resolution pipeline and discuss example techniques for the individual phases.** [16/100 points]



Task 1: Entity Resolution, cont.

- **b) Assume two publication datasets A and B that need deduplication. Explain the following two categories of schema matching techniques.** [4/100 points]

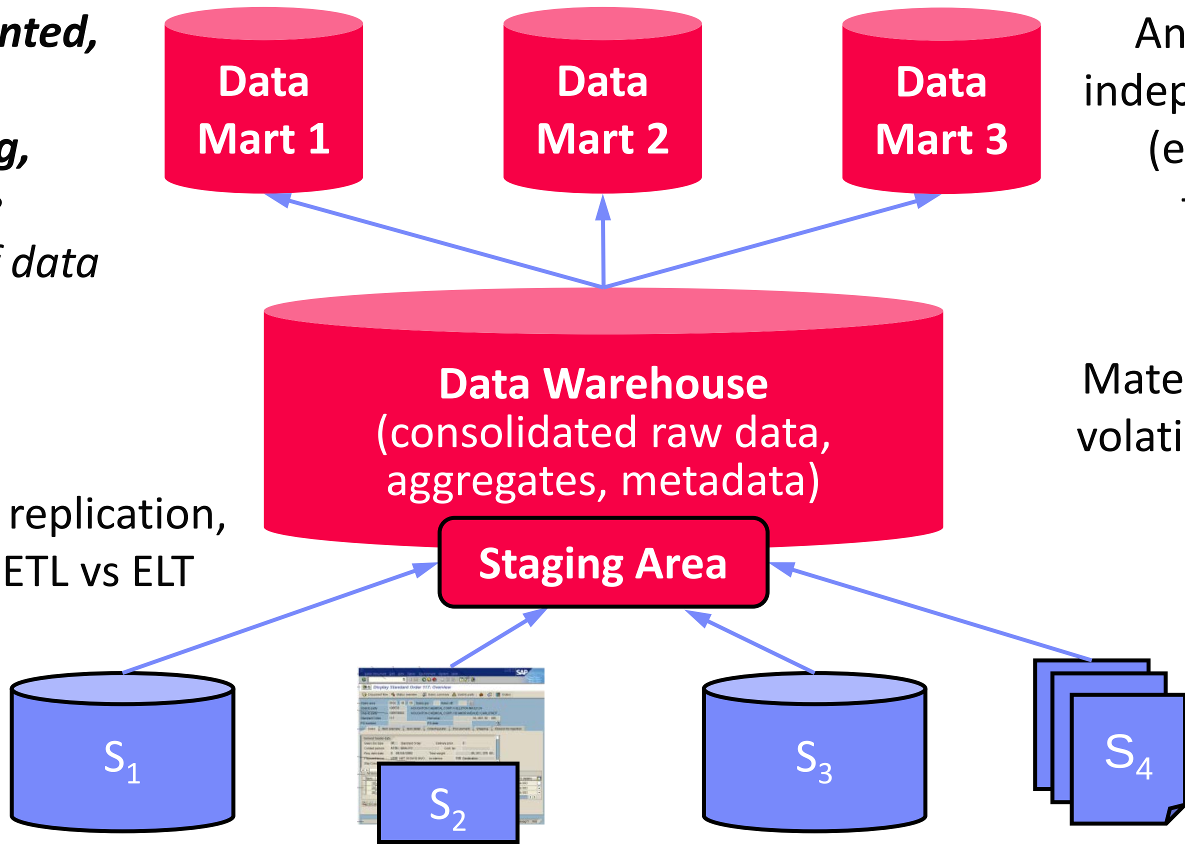
- **Schema-based Matching:**
  - Find similarities among (groups of) attributes of S1 and S2
  - **Examples:** match paper title and author attributes based on attribute similarity

- **Instance-based Matching:**
  - Find similarities among (groups of) attributes of S1 and S2, with the help of instance data in S1 and S2
  - **Examples:** match paper titles and author attributes based on term frequencies, string similarity of example papers (e.g., after capitalization of words, splitting of author lists)

Task 2: Data Warehousing

- **a) Describe the overall system architecture of a data warehouse, name its components, and briefly describe their purpose.** [5/100 points]

subject-oriented, integrated, time-varying, non-volatile *collection of data*



Analysis-centric independent subsets (e.g., geo, org, functional)

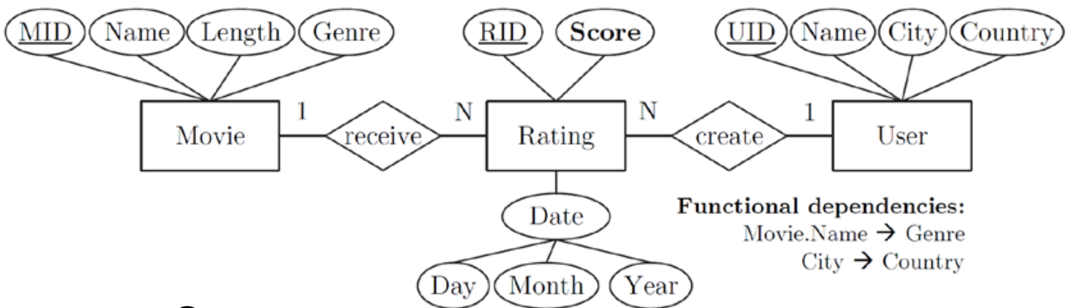
Materialized, non-volatile integration

Async replication, and ETL vs ELT

Operational source systems

Task 2: Data Warehousing, cont.

- **b) Given below entity relationship (ER) diagram, create the corresponding star and snowflake schemas. Data types can be ignored, but indicate primary and foreign key constraints.** [5+5/100 points]



- **Star Schema**

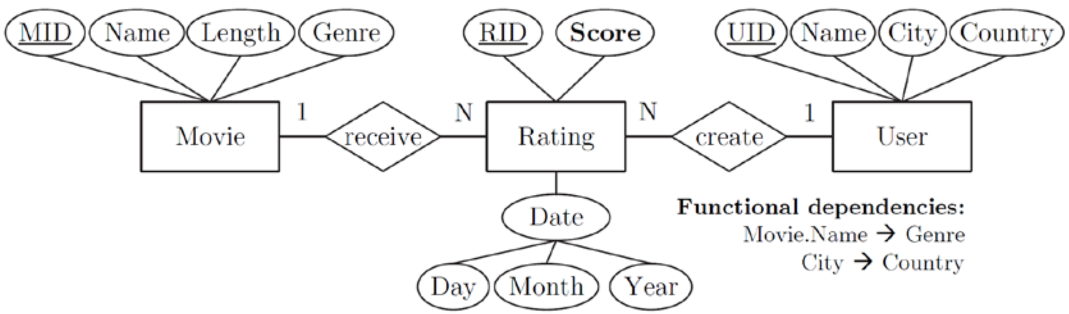
Movies
MID
Name
Length
Genre

Ratings
MID
UID
DID
Score

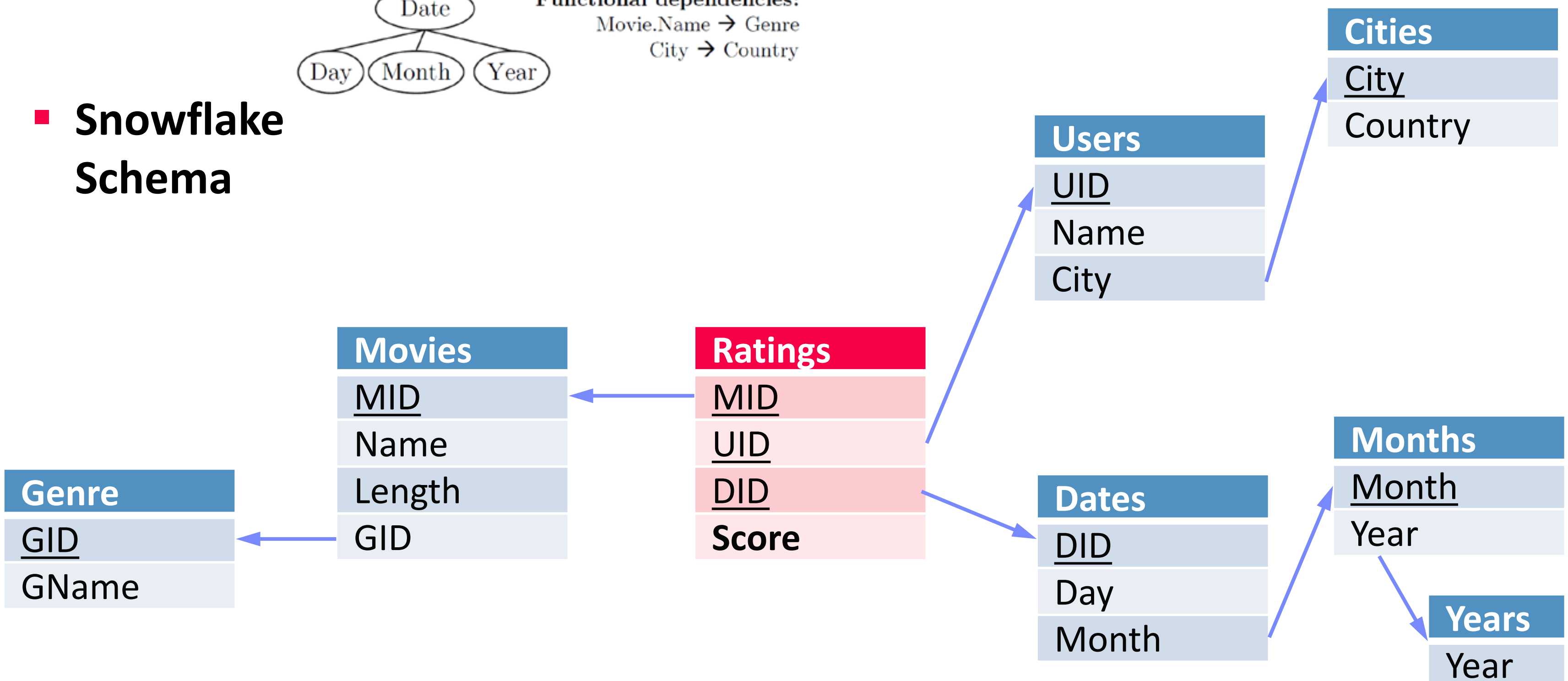
Users
UID
Name
City
Country

Dates
DID
Day
Month
Year

Task 2: Data Warehousing, cont.



- **Snowflake Schema**



Task 3: Data Cleaning

- **a) In the context of missing value imputation, describe the following types of missing data.** [9/100 points]

ID	Position	Salary ($)	
1	Manager	**null**	(3500)
2	Secretary	2200	
3	Manager	3600	
4	Technician	**null**	(2400)
5	Technician	2500	
6	Secretary	**null**	(2000)

- **Missing Completely at Random (MCAR):**
  - Missing values are randomly distributed across all records

ID	Position	Salary ($)
1	Manager	3500
2	Secretary	2200
3	Manager	3600
4	**Technician**	**null**
5	**Technician**	**null**
6	Secretary	2000

- **Missing at Random (MAR):**
  - Missing values are randomly distributed within one or more sub-groups of records
  - Missing values depend on the recorded but not on the missing values, and **can be recovered**

ID	Position	Salary ($)
1	Manager	3500
2	Secretary	**null**
3	Manager	3600
4	Technician	**null**
5	Technician	2500
6	Secretary	**null**

<= 2400 missing

- **Not Missing at Random (NMAR):**
  - Missing data depends on the missing values themselves
  - E.g., missing low salary, age, weight, etc.

Task 3: Data Cleaning

- **b) Given the data below, name two techniques for missing value imputation (1x MCAR, 1x MAR), and impute the values.** [5/100 points]

Name	Age	Salary
Red	45	4500
Orange	50	NULL
Yellow	20	2000
Green	40	4000
Blue	25	2500
Violet	35	NULL

  - **MCAR:** mean imputation (4500+2000+4000+2500)/4 = **3250**
  - **MAR:** linear regression, functional dependencies (Age * 100) = **5000** and **3500**

- **c) Explain the difference between Outlier Detection and Anomaly Detection, with at least one example strategy for each.** [6/100 points]

- **Outlier Detection:**
  - Remove likely incorrect values from data analysis
  - Classification, clustering, pattern recognition (e.g., outlierByIQR)
- **Anomaly Detection:**
  - Find rare / anomalous data points / subsequences
  - Classification / max k-nearest neighbor (e.g., matrix profile)

Task 4: Data Provenance

- **a) Explain the general goal and concept of data provenance, and distinguish why-provenance and how-provenance.** [5/100 points]

- **Data Provenance:**
  - Track and understand data origins and transformations of data (**where?**, **when?**, **who?**, **why?**, **how?**)
  - Information about the **origin** and **creation process** of data

- **Why-Provenance:**
  - Which input tuples contributed to an output tuple t in query Q
  - **Representation:** Set of **witnesses** w for tuple t

- **How-Provenance:**
  - How tuples where combined in the computation of an output
  - **Representation:** **provenance polynomials**

Task 4: Data Provenance, cont.

- **b) Given below tables R and S (w/ tuples r_i and s_i), query Q and the results O, specify the provenance polynomials for tuples in O.** [3/100 points]

R

	A	B
r_1	X	1
r_2	Y	2
r_3	Z	1

S

	C	D
s_1	1	A
s_2	2	B
s_3	2	A
s_4	2	C

```
SELECT DISTINCT S.D
  FROM R, S
  WHERE R.B=S.C
```

⇒

O	Provenance Polynomials?
A	
B	
C	

A: r1 x s1 + r3 x s1 + r2 x s3
(equivalent: **(r1 + r3) x s1 + r2 x s3**)

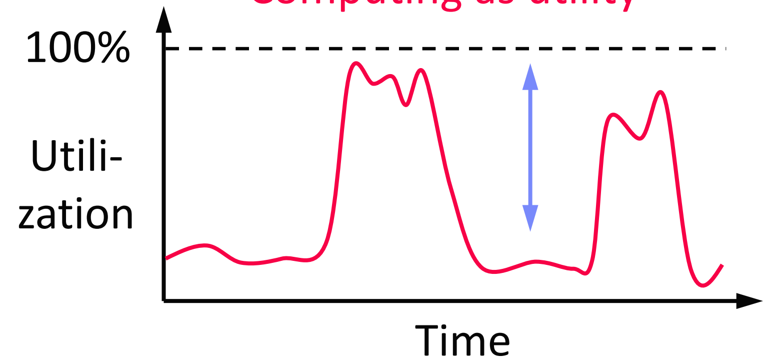
B: r2 x s2

C: r2 x s4

Task 5: Cloud Computing

- **a) Explain the motivation of cloud computing in terms of overall goal, key drivers, and advantages.** [4/100 points]

"Computing as utility"

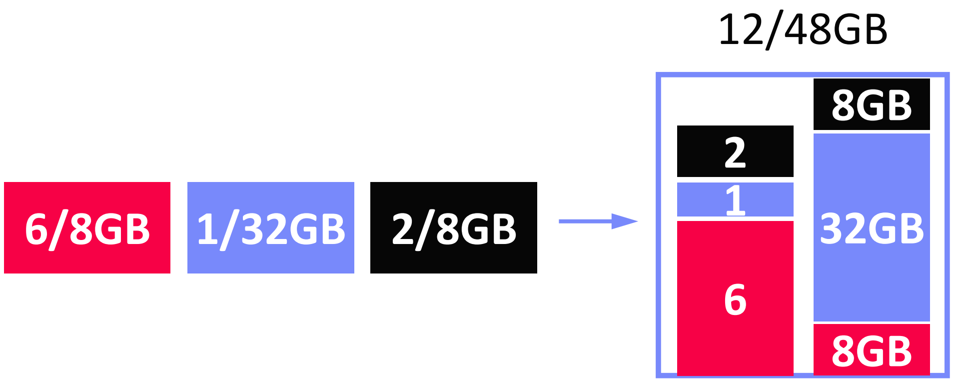


- **Argument #1: Pay as you go**
  - No upfront cost for infrastructure
  - Variable utilization ➔ over-provisioning
  - **Pay per use or acquired resources**
- **Argument #2: Economies of Scale**
  - Purchasing and managing IT infrastructure at scale ➔ **lower cost** (applies to both HW resources and IT infrastructure/system experts)
  - Focus on **scale-out on commodity HW** over scale-up ➔ **lower cost**
- **Argument #3: Elasticity**
  - Assuming perfect scalability, work done in **constant time * resources**
  - Given virtually unlimited resources allows to reduce time as necessary

Task 5: Cloud Computing, cont.

- **b) Explain the concept of resource allocation for multiple resources such as CPU and memory (dominant resource calculation in YARN).** [3/100 points]

- **Multi-Metric Scheduling**
  - Multiple metrics: **dominant resource calculator**
  - All constraints of relevant metrics must be respected
  - Focus on bottleneck resource during scheduling



Task 6: Distributed, Data-parallel Computation

- **Given a distributed dataset (left), describe a data-parallel approach of imputing the missing values (NULL) of Attr1 with its mode, and Attr2 with its mean. Describe strategies for improving the performance. Finally, fill in the concrete imputed values (right).** [12+5+3/100 points]

Attr1	Attr2
X	3
X	4
NULL	1
Y	7
X	2
Y	NULL
X	1
X	2
Y	5
NULL	NULL
Z	8
NULL	4

```
1: data-parallel group-by [Attr1,count]
   → (X:5),(Y,3),(Z,1)
2: data-parallel sum(Attr2)
   → 37
3: data-parallel count(Attr2)
   → 10
4: Apply mode and mean to input data
```

with shuffling

Imputed

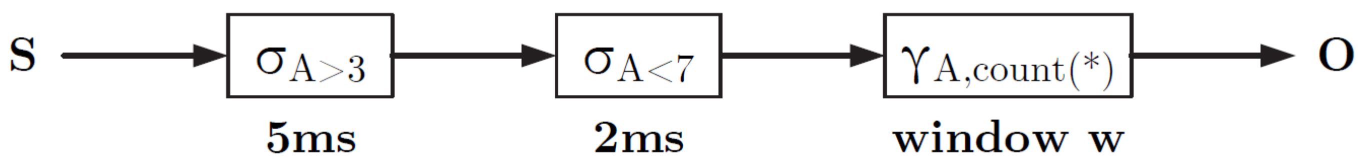
Attr1	Attr2
X	3
X	4
X	1
Y	7
X	2
Y	3.7
X	1
X	2
Y	5
X	3.7
Z	8
X	4

Performance Improvements:

- Pre-aggregation/combine (groupByKey → reduceByKey)
- Caching for multi-pass computation
- Fusion of passes 1-3 with multiple outputs

Task 7: Stream Processing

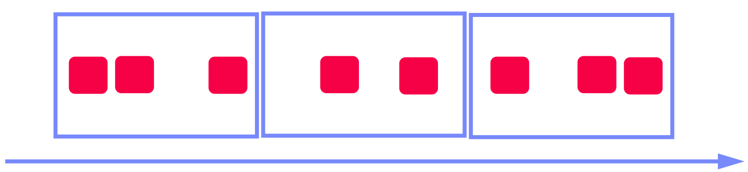
- **Assume an input stream S with schema S(A,T) (where T is event time, and A is an integer column) and a continuous query Q with stream window aggregation. Compute the maximum output stream rate (tuples/second) for the following windows.** [4/100 points]



Max 200 tuples/s

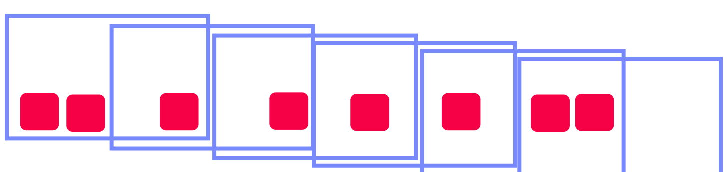
Max 3 tuples/window

- **Tumbling Window (size 200ms):**



→ **15 Tuples/s**

- **Sliding Window (size 500ms, step 100ms):**

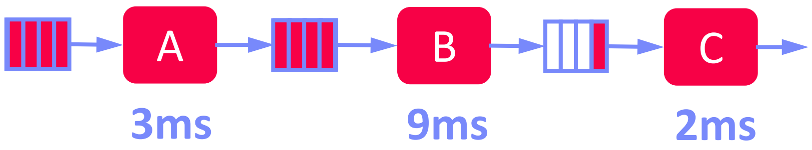


→ **30 Tuples/s**

Task 7: Stream Processing

- **b) Explain the following three techniques for handling overload situations in stream processing engines?** [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**
  - Data flow partitioning (distribute the query)
  - Key range partitioning (distribute the data stream

Summary and Q&A

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

Thanks

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

- **#1 Projects and Exercises**
  - **Feb 28, 11.59pm** last chance exercise submission (7 late days)

- **#2 Course Evaluation and Exam**
  - Evaluation period: **Dec 15 – Jan 31** (1/120)
  - Exam date: **Feb 04, 3pm** in HS i13 (47/120)