TU Graz

SCIENCE
PASSION
TECHNOLOGY

Architecture of ML Systems
08 Data Access Methods

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: May 24, 2019

ISDS

Announcements/Org

- **#1 Programming/Analysis Projects**
  - **#1 Auto Differentiation**
  - **#5 LLVM Code Generator**
  - **#12 Information Extraction from Unstructured PDF/HTML**
  - ➔ Keep code PRs / status updates in mind

- **#2 Open Positions (2x PhD/Student Assistant)**
  - **ExDRa: Exploratory Data Science over Raw Data** (Siemens, DFKI, TU Berlin, TU Graz), **starting June 1**
  - **Federated ML + ML over raw data** (integration/cleaning/preprocessing)

- **#3 Open Master Thesis w/ AVL**
  - Topic: **Anomaly Detection on Test beds** (durability runs on engine test bed with periodically repeating cycles)
  - Contact: Dr. Christa Simon

Agenda

- **Motivation, Background, and Overview**
- **Caching, Partitioning, and Indexing**
- **Lossy and Lossless Compression**

Iterative, I/O-bound ML algorithms ➔ **Data access crucial for performance**

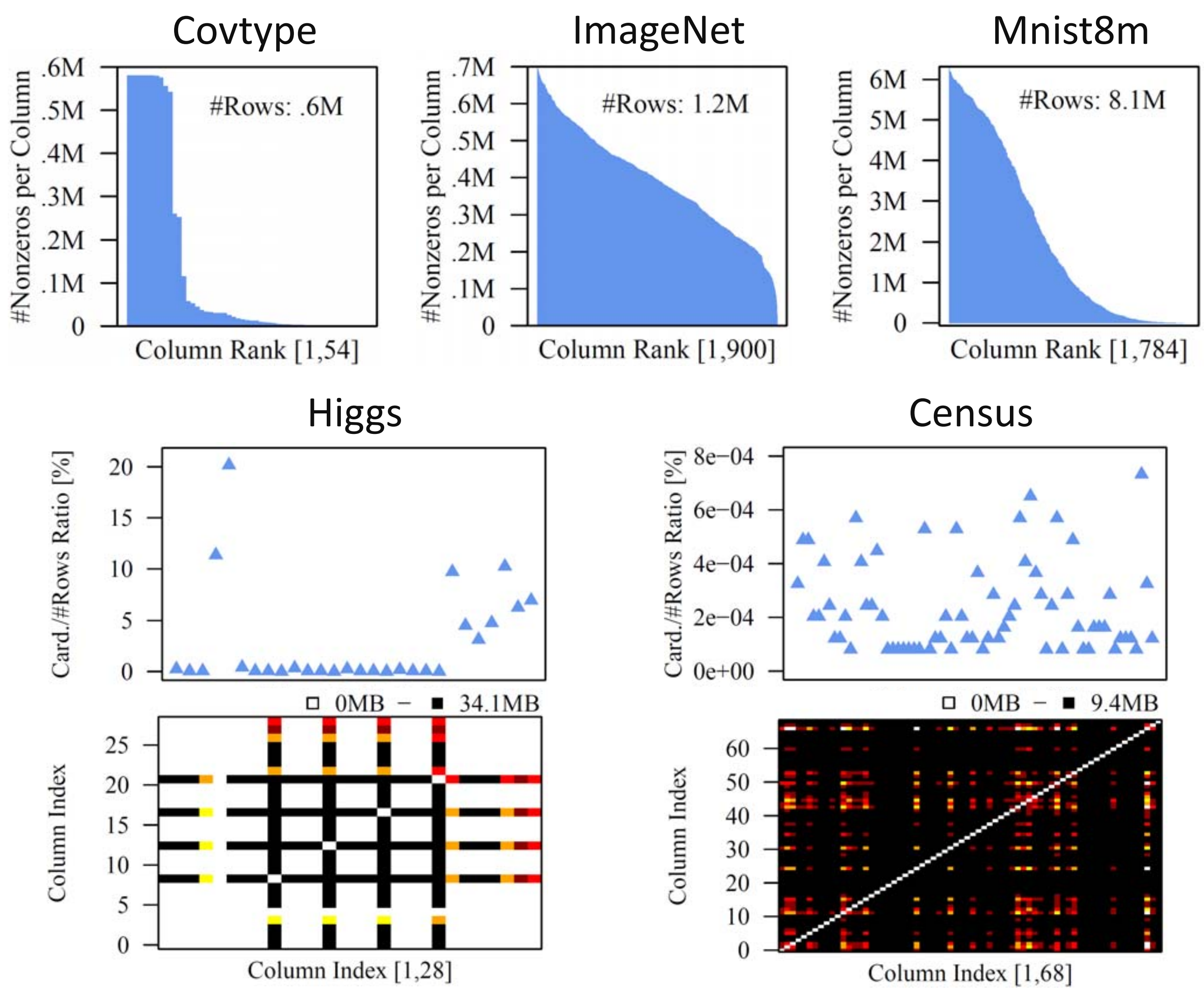
X

```
while(!converged) {
  … q = X %*% v …
}
```

Motivation, Background, and Overview

Motivation: Data Characteristics

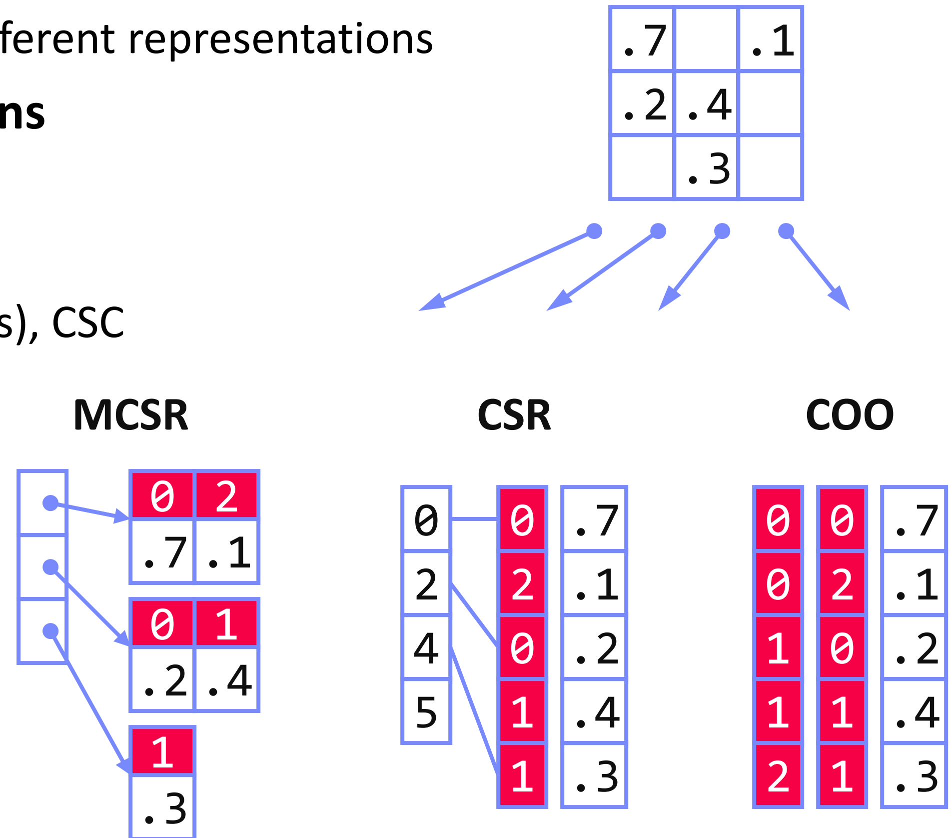
- **Tall and Skinny**
  (#rows >> #cols)
- **Non-Uniform Sparsity**
- **Low Column Cardinalities**
  (e.g., categorical, dummy-coded)
- **Column Correlations**
  (on census: **12.8x → 35.7x**)



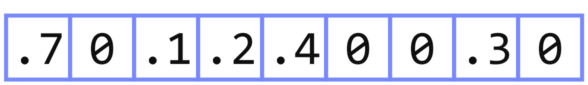
Recap: Matrix Formats

- **Matrix Block** (m x n)
  - A.k.a. tiles/chunks, most operations defined here
  - Local matrix: single block, different representations
- **Common Block Representations**
  - Dense (linearized arrays)
  - MCSR (modified CSR)
  - CSR (compressed sparse rows), CSC
  - COO (Coordinate matrix)

Example
3x3 Matrix



Dense (row-major)



O(mn)

O(m + nnz(X))

O(nnz(X))

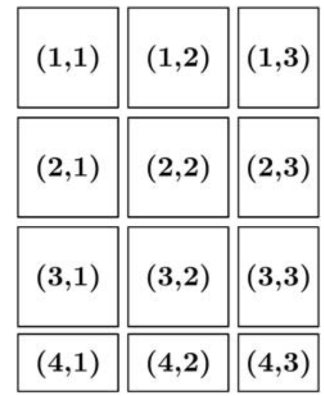
Recap: Distributed Matrices

- **Collection of "Matrix Blocks" (and keys)**
  - Bag semantics (duplicates, unordered)
  - Logical (Fixed-Size) Blocking

    + join processing / independence

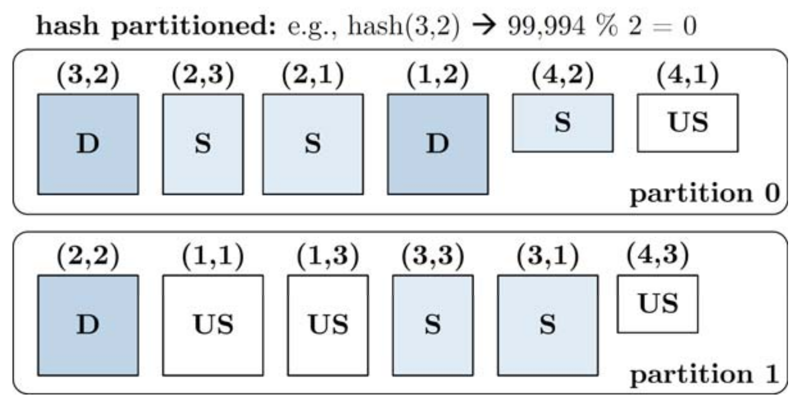
    - (sparsity skew)
  - E.g., SystemML on Spark:
    JavaPairRDD<MatrixIndexes,MatrixBlock>
  - Blocks encoded independently (dense/sparse)

Logical Blocking
3,400x2,700 Matrix
(w/ B_c=1,000)



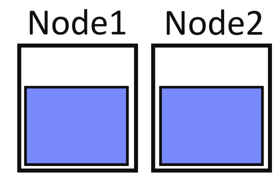
- **Partitioning**
  - Logical Partitioning
    (e.g., row-/column-wise)
  - Physical Partitioning
    (e.g., hash / grid)

Physical Blocking and Partitioning

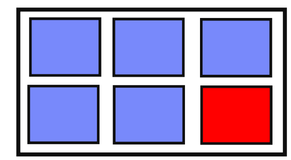


Overview Data Access Methods

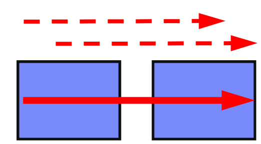
- **#1 (Distributed) Caching**
  - Keep read only feature matrix in (distributed) memory



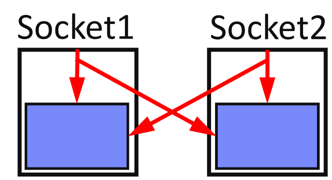
- **#2 Buffer Pool Management**
  - Graceful eviction of intermediates, out-of-core ops



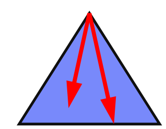
- **#3 Scan Sharing (and operator fusion)**
  - Reduce the number of scans as well as read/writes



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



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

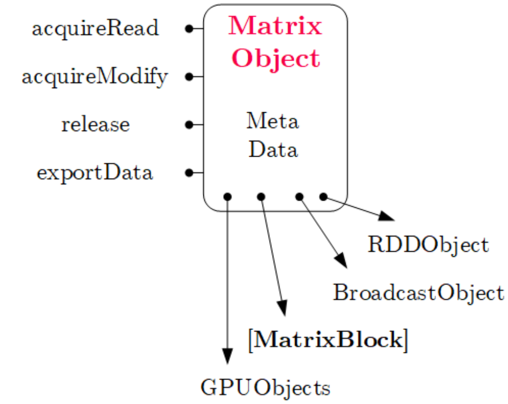


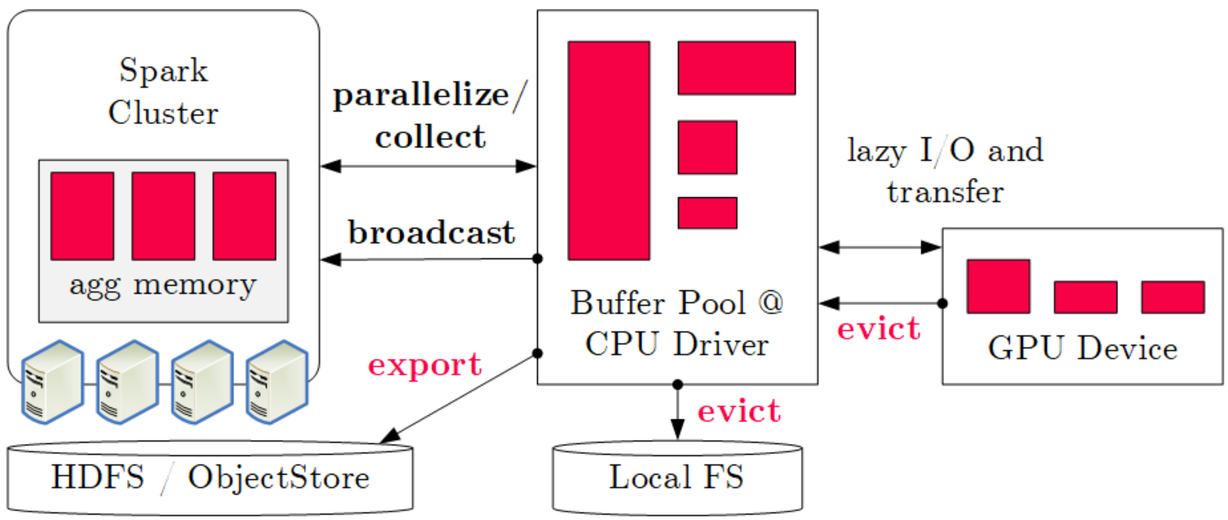
- **#6 Compression**
  - Fit larger datasets into available memory

Caching, Partitioning, and Indexing

Buffer Pool Management

- **#1 Classic Buffer Management**
  - Hybrid plans of in-memory and distributed ops
  - **Graceful eviction of intermediate variables**

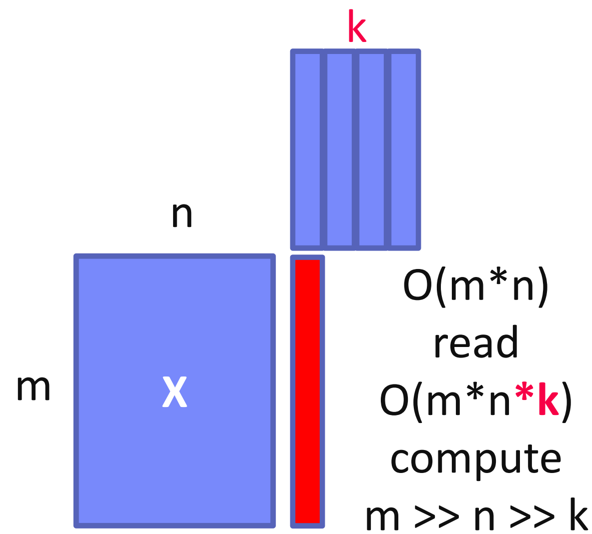




- **#2 Algorithm-Specific Buffer Management**
  - Operations/algorithms over out-of-core matrices and factor graphs
  - Examples: **RIOT** (op-aware I/O), **Elementary** (out-of-core factor graphs)

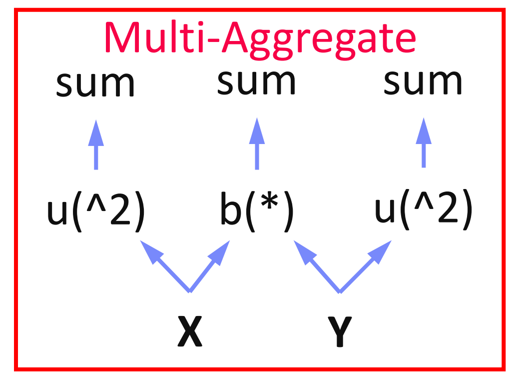
Scan Sharing

- **#1 Batching**
  - One-pass evaluation of multiple configurations
  - Use cases: EL, CV, feature selection, hyper parameter tuning
  - E.g.: **TUPAQ** [SoCC'16], **Columbus** [SIGMOD'14]



- **#2 Fused Operator DAGs**
  - Avoid unnecessary scans, (e.g., mmchain)
  - Avoid unnecessary writes / reads
  - Multi-aggregates, redundancy
  - E.g.: **SystemML codegen**

```
a = sum(X^2)
b = sum(X*Y)
c = sum(Y^2)
```



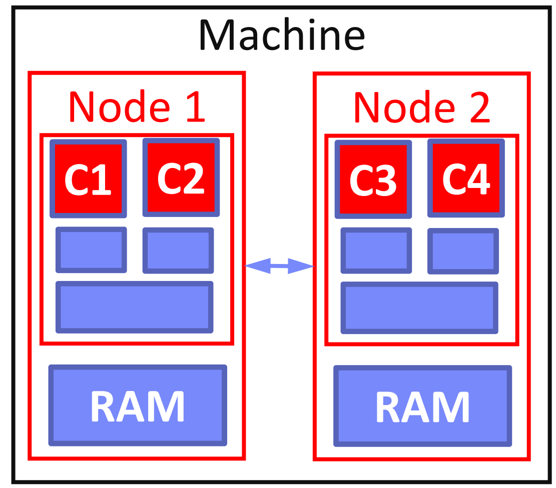
- **#3 Runtime Piggybacking**
  - Merge concurrent data-parallel jobs
  - "Wait-Merge-Submit-Return"-loop
  - E.g.: **SystemML parfor** [PVLDB'14]

```
parfor( i in 1:numModels )
  while( !converged )
    q = X %*% v; ...
```

In-Memory Partitioning (NUMA-aware)

- **NUMA-Aware Model and Data Replication**
  - Model Replication (06 Parameter Servers)
    - PerCore (BSP epoch), PerMachine (Hogwild!), PerNode (hybrid)
  - Data Replication
    - Partitioning (sharding)
    - Full replication

[Ce Zhang, Christopher Ré: DimmWitted: A Study of Main-Memory Statistical Analytics. **PVLDB 2014**]



- **AT MATRIX (Adaptive Tile Matrix)**
  - Recursive NUMA-aware partitioning into dense/sparse tiles
  - Inter-tile (worker teams) and intra-tile (threads in team) parallelization
  - Job scheduling framework from SAP HANA (horizontal range partitioning, socket-local queues with task-stealing)

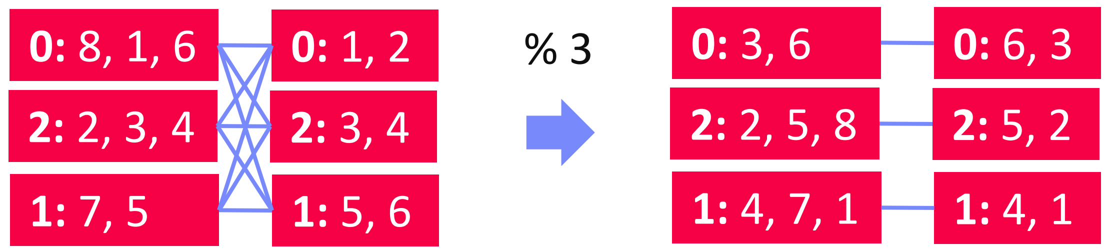
[David Kernert, Wolfgang Lehner, Frank Köhler: Topology-aware optimization of big sparse matrices and matrix multiplications on main-memory systems. **ICDE 2016**]

Distributed Partitioning

- **Spark RDD Partitioning**
  - Implicitly on every data shuffling
  - Explicitly via `R.repartition(n)`

Example Hash Partitioning:
For all (k,v) of R:
hash(k) % numPartitions → pid

- **Distributed Joins**



- **Single-Key Lookups** `v = C.lookup(k)`
  - **Without partitioning:** scan all keys (reads/deserializes out-of-core data)
  - **With partitioning:** lookup partition, scan keys of partition

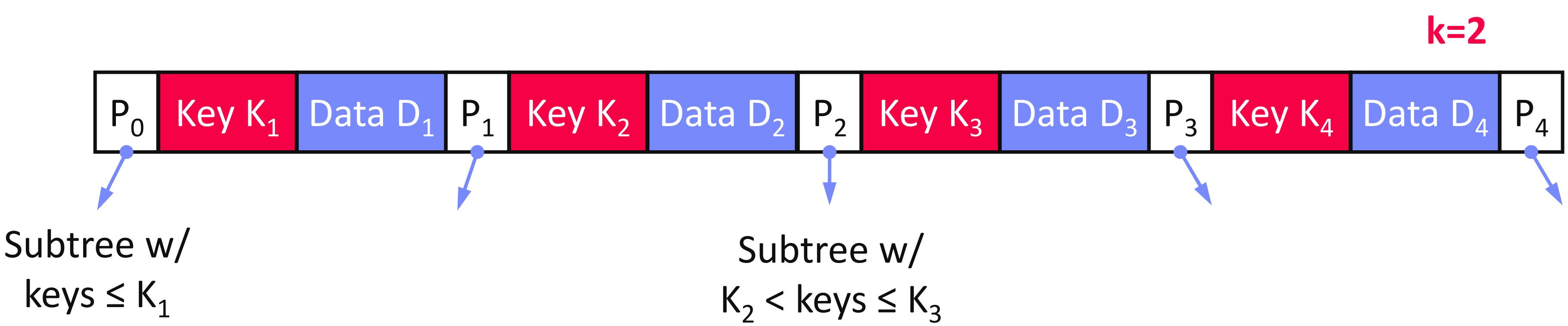
- **Multi-Key Lookups**
  - Without partitioning: scan all keys
  - With partitioning: lookup relevant partitions

```
//build hashset of required partition ids
HashSet<Integer> flags = new HashSet<>();
for( MatrixIndexes key : filter )
   flags.add(partitioner.getPartition(key));

//create partition pruning rdd
ppRDD = PartitionPruningRDD.create(in.rdd(),
    new PartitionPruningFunction(flags));
```

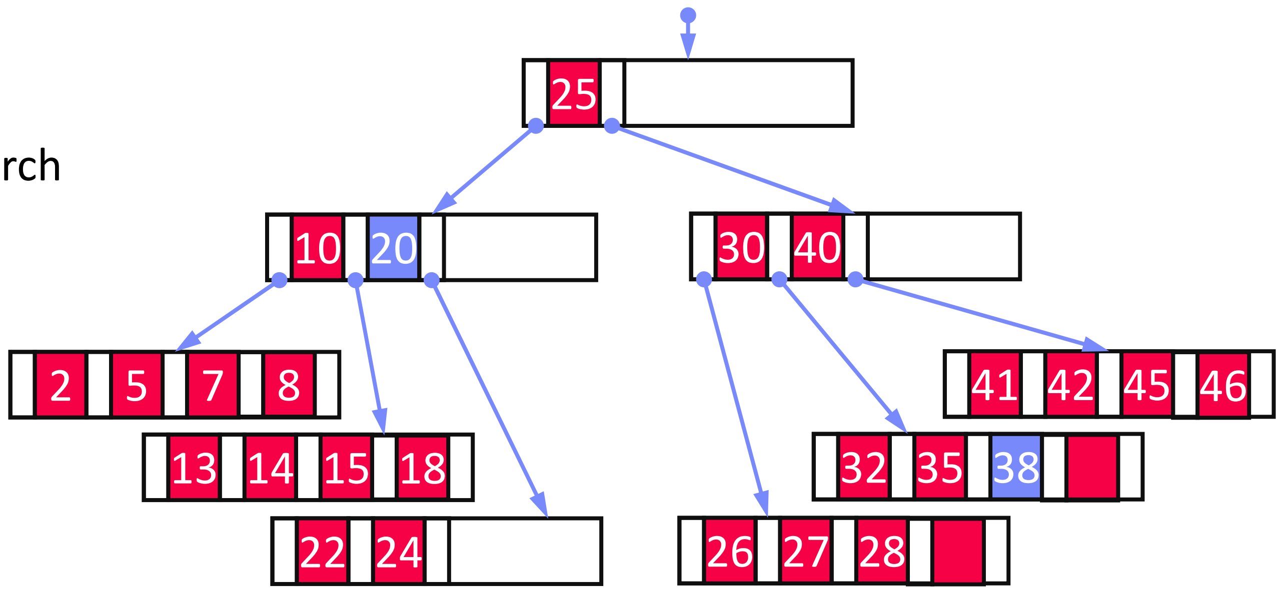
Recap: B-Trees

- **History B-Tree**
  - **B**ayer and McCreight 1972 (multiple papers), **B**lock-based, **B**alanced, **B**oeing
  - Multiway tree (node size = page size); designed for DBMS
- **Definition B-Tree k**
  - **Balanced tree:** All paths from root to leafs have equal length h
  - All nodes (except root=leaf) have **[k, 2k]** key entries
  - All nodes (except root, leafs) have **[k+1, 2k+1]** successors
  - Data is a record or a reference to the record (RID)



Recap: B-Trees, cont.

- **B-Tree Search**
  - Scan/binary search with nodes
  - Descend along matching key ranges



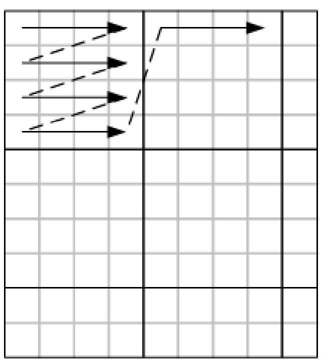
- **B-Tree Insertion**
  - Insert into leaf nodes
  - Split the 2k+1 entries into two leaf nodes
- **B-Tree Deletion**
  - Lookup key and delete if existing
  - Move entry from fullest successor; if underflow merge with sibling

Linearized Array B-Tree (LAB-Tree)

- **Basic Ideas**
  - B-tree over linearized array representation (e.g., row-/col-major, Z-order, UDF)
  - New **leaf splitting strategies**; dynamic **leaf storage format** (sparse and dense)
  - Various **flushing policies** for update batching (all, LRU, smallest page, largest page, largest page probabilistically, largest group)

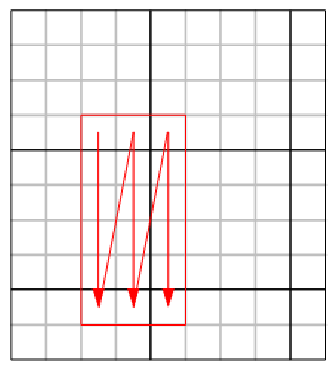
[Yi Zhang, Kamesh Munagala, Jun Yang: Storing Matrices on Disk: Theory and Practice Revisited. **PVLDB 2011**]

#1 Example linearized storage order



matrix A:
4 x 4 blocking
row-major block order
row-major cell order

#2 Example linearized iterator order



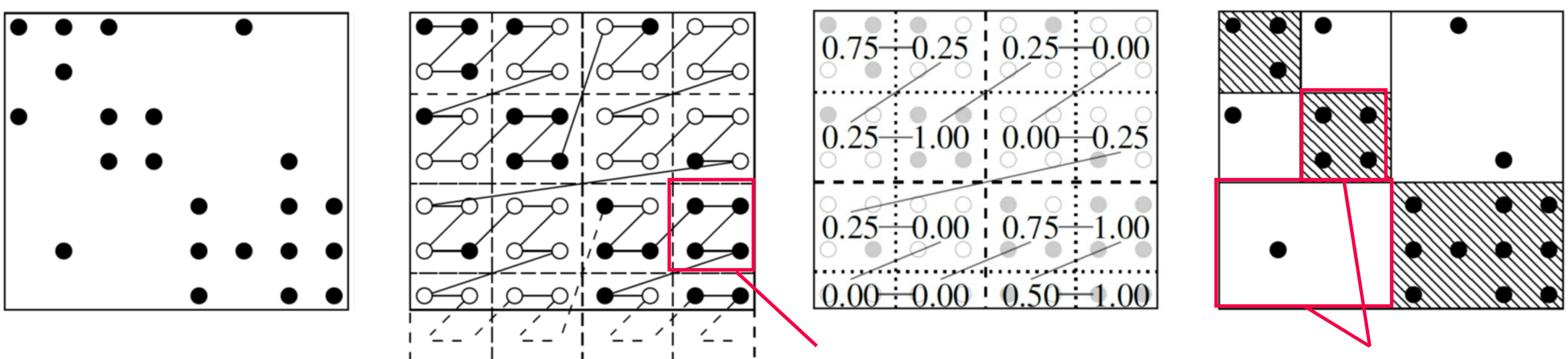
range query A[4:9,3:5]
with column-major
iterator order

Adaptive Tile (AT) Matrix

- **Basic Ideas**
  - Two-level blocking and NUMA-aware range partitioning (tiles, blocks)
  - Z-order linearization, and **recursive quad-tree partitioning** to find var-sized tiles (tile contains N blocks)

[David Kernert, Wolfgang Lehner, Frank Köhler: Topology-aware optimization of big sparse matrices and matrix multiplications on main-memory systems. **ICDE 2016**]

Input Matrix | **Z-ordering** | **Density Map** (see sparsity est.)



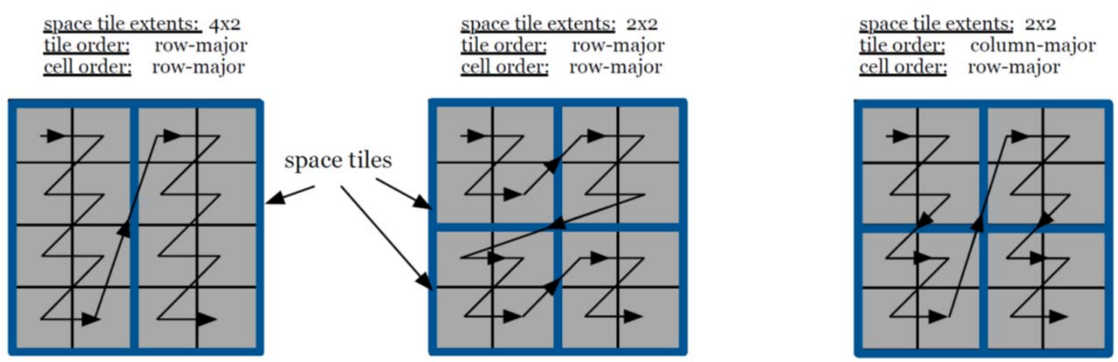
block

tiles

TileDB Storage Manager

- **Basic Ideas**
  - Storage manager for 2D arrays of different data types (incl. vector, 3D)
  - **Two-level blocking** (space/data tiles), update batching via **fragments**

[Stavros Papadopoulos, Kushal Datta, Samuel Madden, Timothy G. Mattson: The TileDB Array Data Storage Manager. **PVLDB 2016**]



Fragment #1 (dense)

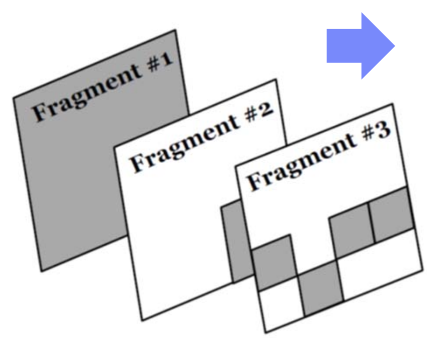
	1	2	3	4
1	0 a	1 bb	4 e	5 ff
2	2 ccc	3 dddd	6 ggg	7 hhhh
3	8 i	9 jj	12 m	13 nn
4	10 kkk	11 llll	14 ooo	15 pppp

Fragment #2 (dense)

	1	2	3	4
1				
2				
3			112 M	113 NN
4			114 OOO	115 PPPP

Fragment #3 (sparse)

	1	2	3	4
1				
2				
3	208 u		212 x	213 yy
4		211 wwww		



Collective logical array view

	1	2	3	4
1	0 a	1 bb	4 e	5 ff
2	2 ccc	3 dddd	6 ggg	7 hhhh
3	208 u	9 jj	212 x	213 yy
4	10 kkk	211 wwww	114 OOO	115 PPPP

Pipelining for Mini-batch Algorithms

- **Motivation**
  - Overlap data access and computation in mini-batch algorithms (e.g., DNN)
  - Specify approach and configuration at level of linear algebra program

  ➔ **Simple pipelining of I/O and compute via queueing / prefetching**

- **Example TensorFlow**
  - #1: Queueing and threading
  - #2: **Dataset API prefetching**

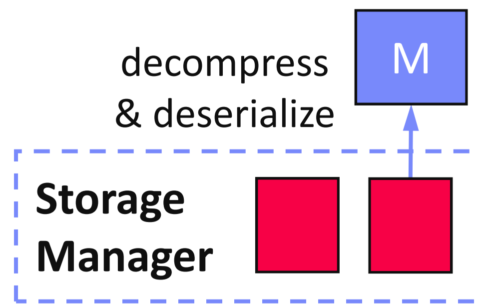

```
dataset = dataset.batch(batch_size=32)
dataset = dataset.prefetch(buffer_size=1)
```

[**Credit:** https://www.tensorflow.org/guide/performance/datasets]

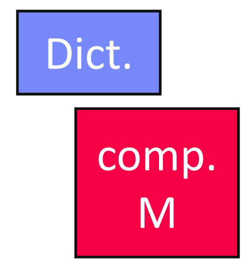
Lossy and Lossless Compression

Overview Lossless Compression Techniques

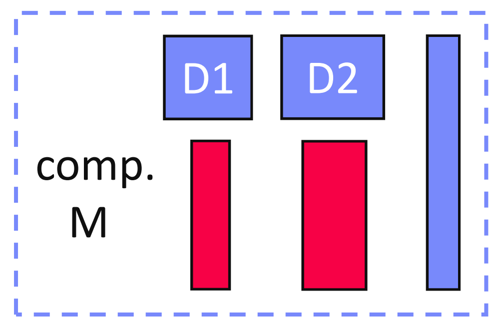
- **#1 Block-Level General-Purpose Compression**
  - Heavyweight or lightweight compression schemes
  - Decompress matrices block-wise for each operation
  - E.g.: Spark RDD compression (Snappy/LZ4), **SciDB** SM [SSDBM'11], **TileDB** SM [PVLDB'16], scientific formats **NetCDF**, **HDF5** at chunk granularity



- **#2 Block-Level Matrix Compression**
  - Compress matrix block with homogeneous encoding scheme
  - Perform LA ops over compressed representation
  - E.g.: **CSR-VI** (dict) [CF'08], **cPLS** (grammar) [KDD'16], **TOC** (LZW w/ trie) [CoRR'17]



- **#3 Column-Group-Level Matrix Compression**
  - Compress column groups w/ heterogeneous schemes
  - Perform LA ops over compressed representation
  - E.g.: **SystemML CLA** (RLE, OLE, DDC, UC) [PVLDB'16]



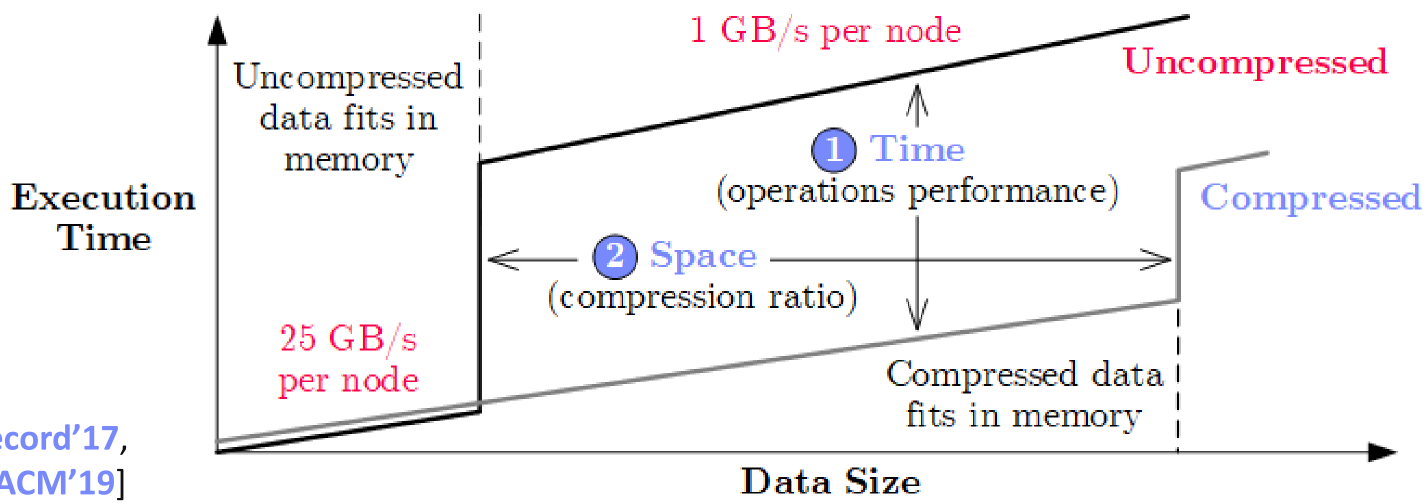
CLA: Compressed Linear Algebra

[Ahmed Elgohary et al: Compressed Linear Algebra for Large-Scale Machine Learning. **PVLDB 2016**]

- **Key Idea**
  - Use lightweight database compression techniques
  - Perform LA operations **on compressed matrices**
- **Goals of CLA**
  - Operations performance close to uncompressed
  - Good compression ratios

X

```
while(!converged) {
  … q = X %*% v …
}
```

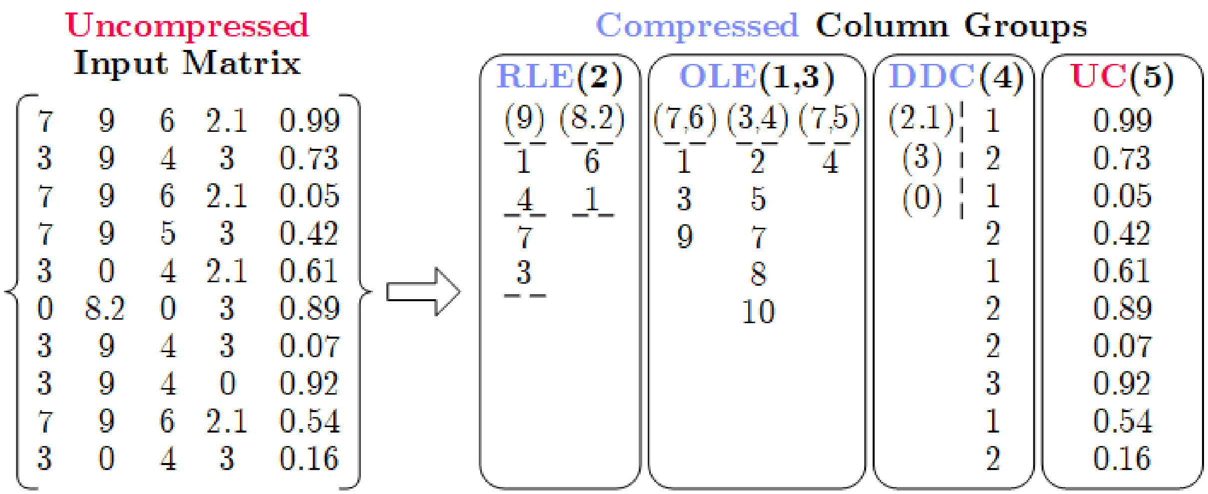


[**SIGMOD Record'17**, **VLDBJ'18**, **CACM'19**]

CLA: Compressed Linear Algebra, cont. (2)

- **Overview Compression Framework**
  - Column-wise matrix compression (values + compressed offsets / references)
  - Column co-coding (column groups, encoded as single unit)
  - Heterogeneous column encoding formats (w/ dedicated physical encodings)

- **Column Encoding Formats**
  - Offset-List (OLE)
  - Run-Length (RLE)
  - Dense Dictionary Coding (DDC)*
  - Uncompressed Columns (UC)



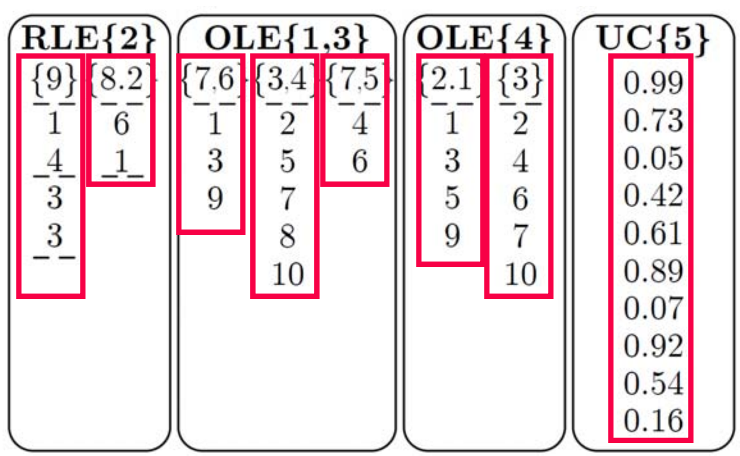
* DDC1/2 in VLDBJ'17

- **Automatic Compression Planning (sampling-based)**
  - Select column groups and formats per group (data dependent)

CLA: Compressed Linear Algebra, cont. (3)

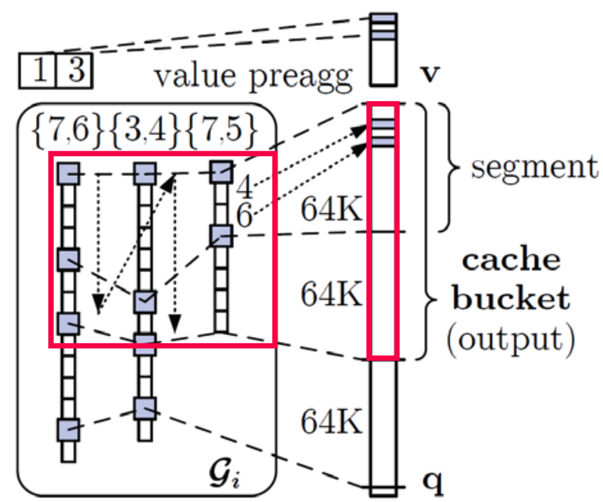
- **Matrix-Vector Multiplication**
  - Naïve: for each tuple, pre-aggregate values, add values at offsets to q
    Example: q = X v, with v = (7, 11, 1, 3, 2)

9*11=99 90.2 55 25 54 6.3 9



162.3
134.5
160.4
162.8
32.5
155
133.1
125.8
161.4
34.3

➔ cache unfriendly on output (q)

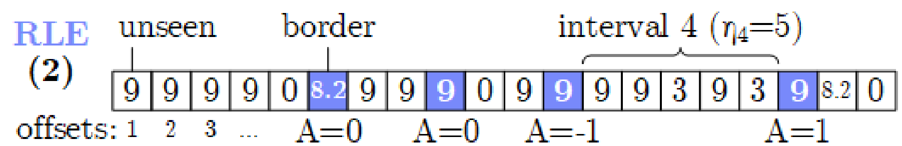


  - Cache-conscious: Horizontal, segment-aligned scans, maintain positions

- **Vector-Matrix Multiplication**
  - Naïve: **cache-unfriendly on input (v)**
  - Cache-conscious: again use horizontal, segment-aligned scans

CLA: Compressed Linear Algebra, cont. (4)

- **Estimating Compressed Size:** $S^C = \min(S^{OLE}, S^{RLE}, S^{DDC})$
  - \# of distinct tuples d_i: **"Hybrid generalized jackknife" estimator** [JASA'98]
  - \# of OLE segments b_{ij}: **Expected value under maximum-entropy model**
  - \# of non-zero tuples z_i: **Scale from sample with "coverage" adjustment**
  - \# of runs r_{ij}: **maxEnt model + independent-interval approx.** (~ Ising-Stevens)



- **Compression Planning**
  - **#1 Classify compressible columns**
    - Draw random sample of rows (from transposed X)
    - Classify C^C and C^{UC} based on estimate compression ratio
  - **#2 Group compressible columns** (exhaustive $O(m^m)$, greedy $O(m^3)$)
    - Bin-packing-based column partitioning
    - Greedy grouping per bin **w/ pruning and memoization $O(m^2)$**
  - **#3 Compression**
    - Extract uncompressed offset lists and exact compression ratio
    - Graceful corrections and UC group creation

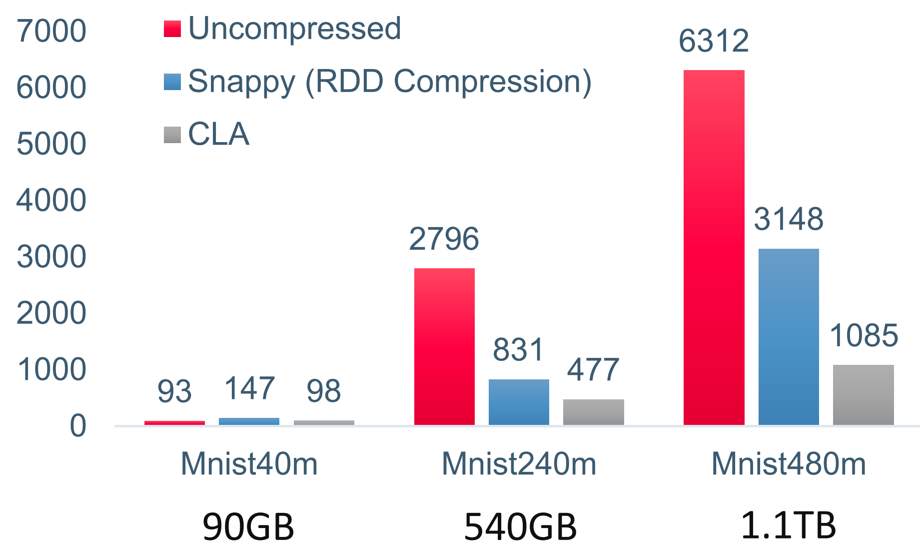
CLA: Compressed Linear Algebra, cont. (5)

- **Experimental Setup**
  - **LinregCG**, **10 iterations** (incl. compression), InfiMNIST data generator
  - 1+6 node cluster (216GB aggregate memory), Spark 2.3, SystemML 1.1

Compression Ratios

Dataset	Gzip	Snappy	CLA
Higgs	1.93	1.38	**2.17**
Census	17.11	6.04	**35.69**
Covtype	10.40	6.13	**18.19**
ImageNet	5.54	3.35	**7.34**
Mnist8m	4.12	2.60	**7.32**
Airline78	7.07	4.28	**7.44**

End-to-End Performance [sec]



- **Open Challenges**
  - **Ultra-sparse datasets**, tensors, **automatic operator fusion**
  - Operations beyond matrix-vector/unary, applicability to deep learning?

Block-level Compression w/ D-VI, CSR-VI, CSX

- **CSR-VI (CSR-Value Indexed) / D-VI**
  - **Create dictionary** for distinct values
  - **Encode 8 byte values as 1, 2, or 4-byte codes** (positions in the dictionary)
  - Extensions w/ delta coding of indexes
  - Example CSR-VI matrix-vector multiply
    c = A %*% b

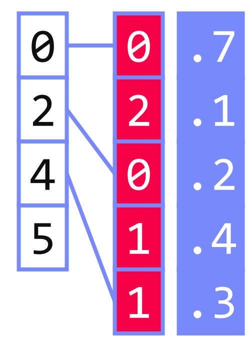
[Kornilios Kourtis, Georgios I. Goumas, Nectarios Koziris: Optimizing sparse matrix-vector multiplication using index and value compression. **CF 2008**]

[Vasileios Karakasis et al.: An Extended Compression Format for the Optimization of Sparse Matrix-Vector Multiplication. **IEEE Trans. Parallel Distrib. Syst. 2013**]

```
for(int i=0; i<a.nrow; i++) {
   int pos = A.rptr[i];
   int end = A.rptr[i+1];
   for(int k=pos; k<end; k++)
      b[i] += dict[A.val[k]] * b[A.ix[k]];
}
```

value decoding
(MV over compressed representation)

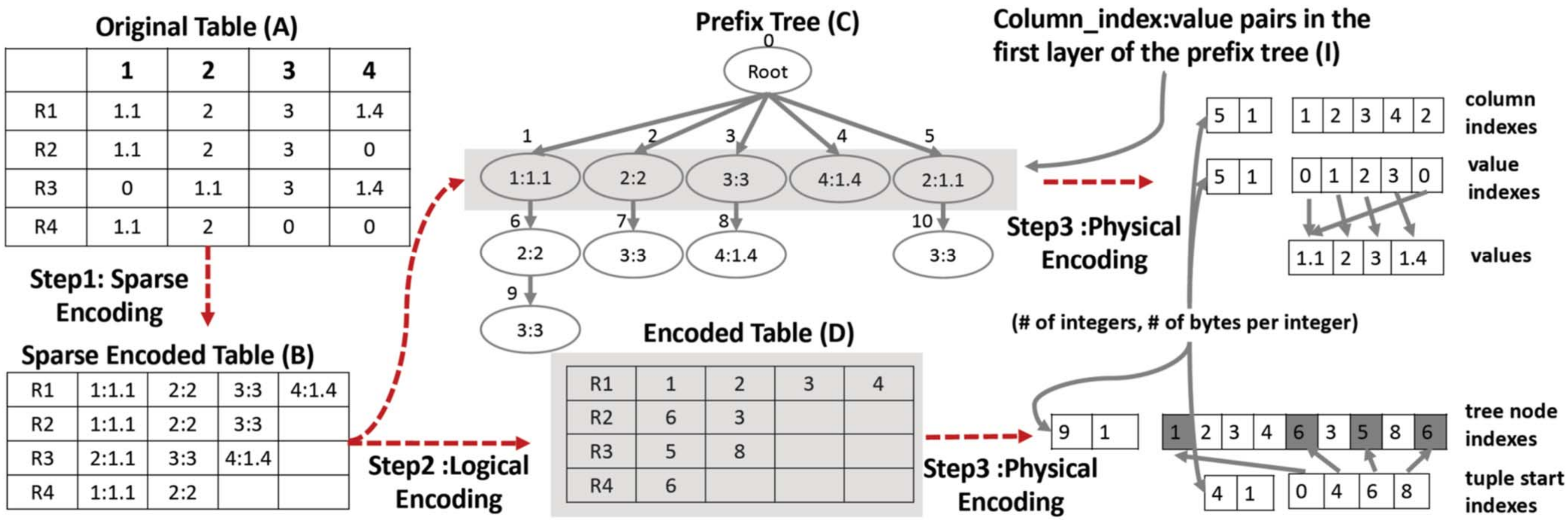
CSR



Tuple-oriented Compression (TOC)

[Fengan Li, Lingjiao Chen, Yijing Zeng, Arun Kumar, Xi Wu, Jeffrey F. Naughton, Jignesh M. Patel: Tuple-oriented Compression for Large-scale Mini-batch Stochastic Gradient Descent, **SIGMOD 2019**]

- **Motivation**
  - DNN and ML often trained with mini-batch SGD
  - Effective compression for small batches (#rows)



Lossy Compression

- **Overview**
  - Extensively used in DNN (runtime vs accuracy) ➔ **data format + compute**
  - **Careful manual application** regarding data and model
  - **Note:** ML algorithms approximate by nature + noise generalization effect

- **Background Floating Point Numbers (IEEE 754)**
  - Sign s, Mantissa m, Exponent e: `value = s * m * 2`^{e} (simplified)

Precision	Sign	Mantissa	Exponent
Double (FP64)	1	52	11
Single (FP32)	1	23	8
Half (FP16)	1	10	5
Quarter (FP8)	1	3	4
Half-Quarter (FP4)	1	1	2

[bits]

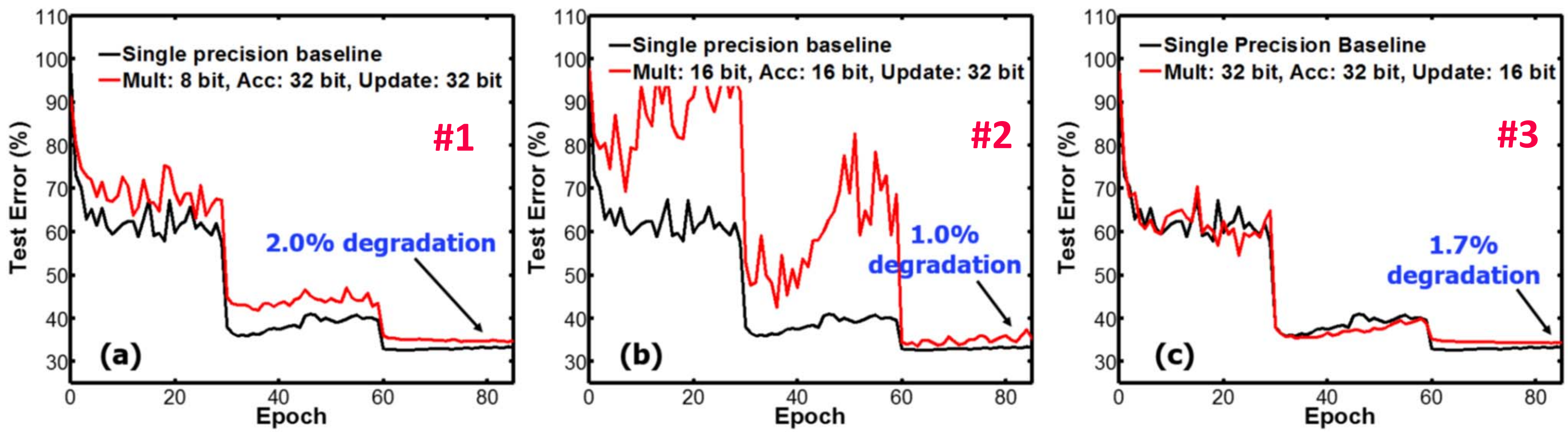
Low and Ultra-low FP Precision

- **Model Training w/ low FP Precision**
  - Trend: from **FP32/FP16** to **FP8**
  - **#1: Precision of intermediates** (weights, act, errors, grad) → loss in accuracy
  - **#2: Precision of accumulation** → impact on convergence (swamping s+L)
  - **#3: Precision of weight updates** → loss in accuracy

see **05 Execution Strategies**, SIMD
→ speedup/reduced energy

[Naigang Wang et al.: Training Deep Neural Networks with **8-bit** Floating Point Numbers. **NeurIPS 2018**]

- **Example ResNet18 over ImageNet**



Low and Ultra-low FP Precision, cont.

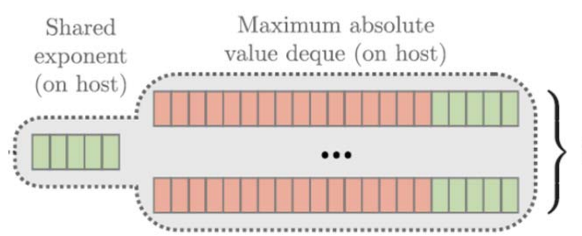
- **Numerical Stable Accumulation**
  - **#1 Sorting ASC + Summation** (accumulate small values first)
  - **#2 Kahan Summation** w/ error independent of number of values n

```
sumOld = sum;
sum = sum + (input + corr);
corr = (input + corr) - (sum - sumOld);
```

- **#3 Chunk-based Accumulation**
  - Divide long dot products into smaller chunks
  - Hierarchy of partial sums → **FP16 accumulators**

[N. Wang et al.: Training Deep Neural Networks with **8-bit** Floating Point Numbers. **NeurIPS 2018**]

- **#4 Stochastic Rounding**
  - Replace nearest with probabilistic rounding
  - Probability accounts for number of bits
- **#5 Intel FlexPoint**
  - Blocks of values w/ shared exponent (16bit w/ 5bit shared exponent)

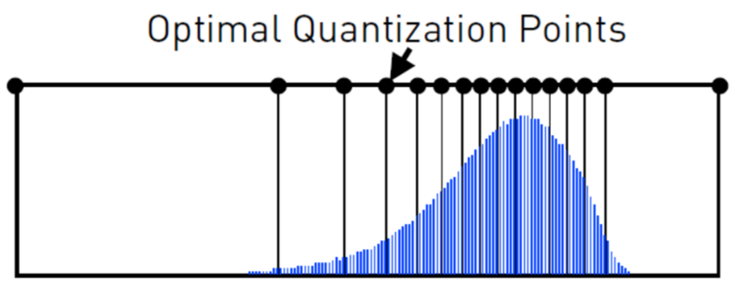


[Credit: Intel @ **NIPS 2017**]

Low Fixed-Point Precision

- **Motivation**
  - Forward-pass for model scoring (inference) can be done in **UINT8** and below
  - **Static, dynamic, and learned quantization schemes**

- **#1 Quantization (reduce value domain)**
  - **Split value domain into N buckets** such that $k = \log_2 N$ can encode the data
  - Static quantization very simple but inefficient on skewed data
  - Learned quantization schemes
    - Dynamic programming
    - Various heuristics
    - Example systems: **ZipML**, **SketchML**



[Hantian Zhang, Jerry Li, Kaan Kara, Dan Alistarh, Ji Liu, Ce Zhang: ZipML: Training Linear Models with End-to-End Low Precision, and a Little Bit of Deep Learning. **ICML 2017**]

Other Lossy Compression Techniques

- **#2 Mantissa Truncation**
  - Mantissa truncation of FP32 from 23bit to 16bit for remote transfers
  - E.g., **TensorFlow**, **PStore**
- **#3 Sparsification (reduce #non-zeros)**
  - **Value clipping:** zero-out very small values below a threshold
- **#4 No FK-PK joins in Factorized Learning**
  - View the **foreign key** as lossy compressed representation of the joined attributes
- **#5 Sampling**
  - User specifies **approximation contract** for error (regression/classification) and scale
  - Estimate minimum necessary sample size for **maximum likelihood estimators**

[Yongjoo Park, Jingyi Qing, Xiaoyang Shen, Barzan Mozafari: BlinkML: Efficient Maximum Likelihood Estimation with Probabilistic Guarantees. **SIGMOD 2019**]

Summary and Conclusions

- **Data Access Methods → High Performance Impact**
  - Caching, Partitioning, and Indexing
  - Lossy and Lossless Compression

- **Next Lectures**
  - **09 Data Acquisition, Cleaning, and Preparation** [Jun 07]
  - **10 Model Selection and Management** [Jun 14]
  - **11 Model Deployment and Serving** [Jun 21]
  - **12 Project Presentations, Conclusions, Q&A** [Jun 28]