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 15, 2020

ISDS

## Announcements/Org

- **#1 Video Recording**
  - Link in TeachCenter & TUbe (lectures will be public)
  - **Live streaming through TUbe**, starting May 08
  - Questions: https://tugraz.webex.com/meet/m.boehm
  - Online teaching extended until Jun 30; **exams via webex**

- **#2 AMLS Programming Projects**
  - **Status:** all project discussions w/ **15 students** (~**4 PRs**)
  - Awesome mix of projects (algorithms, compiler, runtime)
  - Email to m.boehm@tugraz.at if no project discussed yet
  - Soft deadline: **June 30**

- **#3 Open Positions**
  - **1x PhD student** (EU Project, DM+ML+HPC, 4 years, start ~11/2020)
  - **1x Research Student Assistant** (FFG Project, DM+ML, <=20h/week)

## Categories of Execution Strategies

Batch
**SIMD/SPMD**

Batch/Mini-batch,
Independent Tasks
**MIMD**

Mini-batch

**$05_a$ Data-Parallel Execution**
[Apr 03]

**$05_b$ Task-Parallel Execution**
[Apr 03]

**06 Parameter Servers**
(data, model)
[Apr 24]

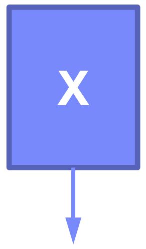
**07 Hybrid Execution and HW Accelerators** [May 08]

**08 Caching, Partitioning, Indexing, and Compression** [May 15]

## Agenda

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

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



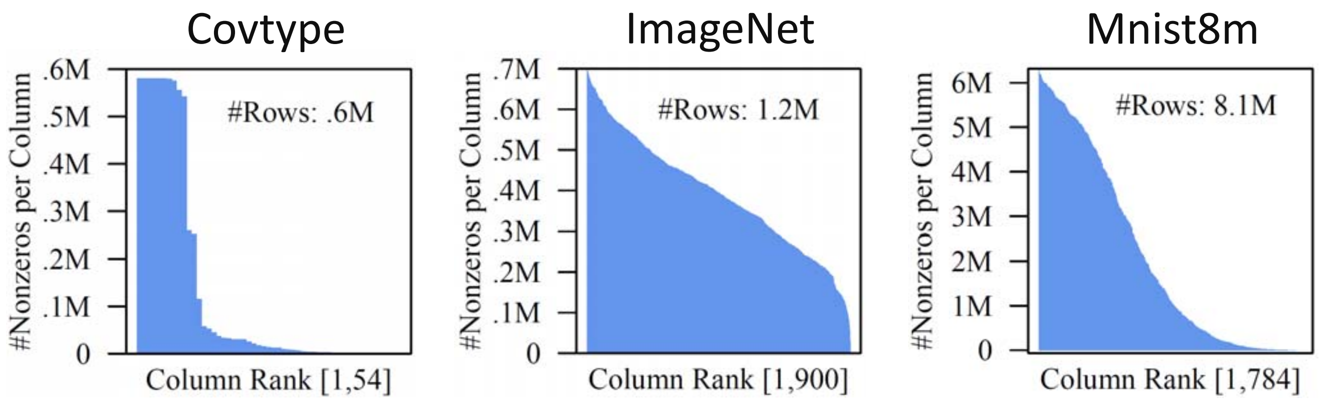
```
while(!converged) {
   … q = X %*% v …
}
```

**Data** **Weights**

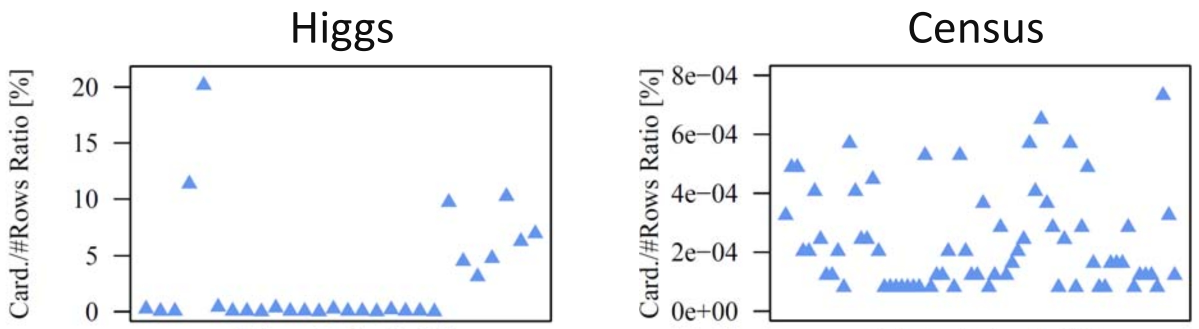
## Motivation, Background, and Overview

## Motivation: Data Characteristics

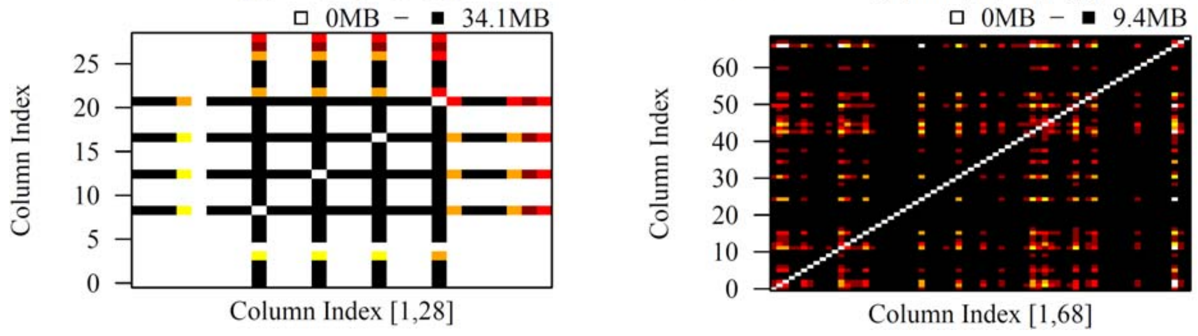
- **Tall and Skinny** (#rows >> #cols)
- **Non-Uniform Sparsity**



- **Small Column Cardinalities**
- **Small Val Range**

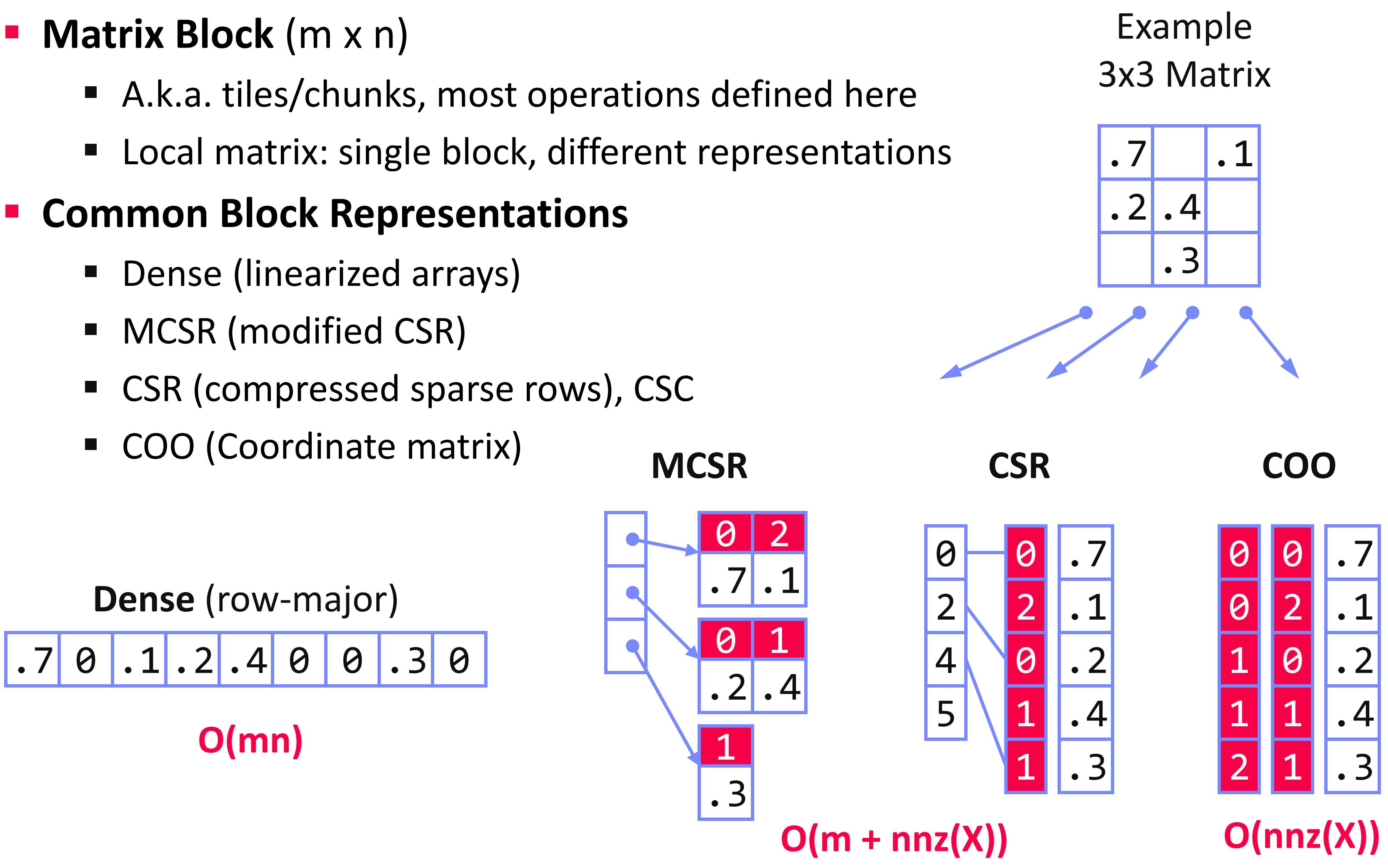


- **Column Correlations** (on census: 12.8x → 35.7x)



## 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)



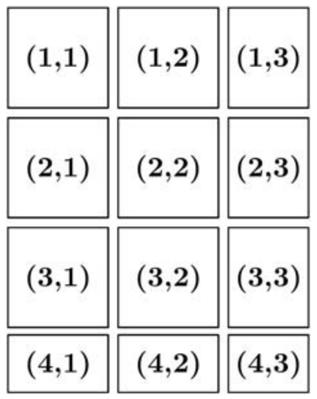
## Recap: Distributed Matrix Representations

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

    **+ join processing / independence**

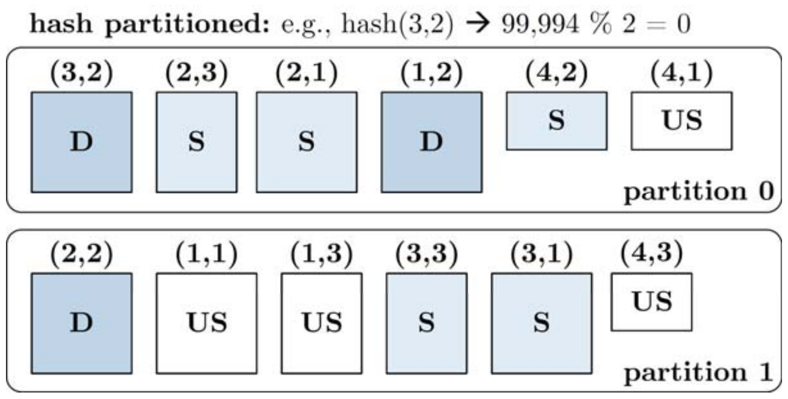
    **- (sparsity skew)**
  - E.g., SystemDS on Spark: `JavaPairRDD<MatrixIndexes,MatrixBlock>`
  - Blocks encoded independently (dense/sparse)

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



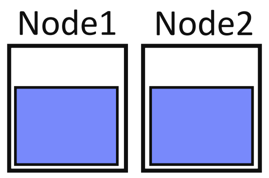
- **Partitioning**
  - Logical Partitioning (e.g., row-/column-wise)
  - Physical Partitioning (e.g., hash / grid)

Physical Blocking and Partitioning

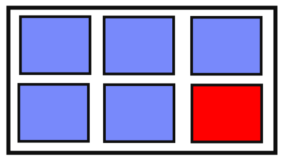


## Overview Data Access Methods

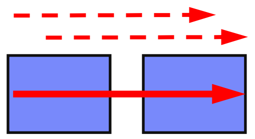
- **#1 (Distributed) Caching**
  - Keep read only feature matrix in (distributed) memory



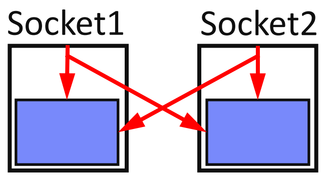
- **#2 Buffer Pool Management**
  - Graceful eviction of intermediates, out-of-core ops



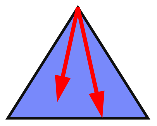
- **#3 Scan Sharing (and operator fusion)**
  - Reduce the number of scans as well as read/writes



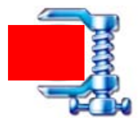
- **#4 NUMA-Aware Partitioning and Replication**
  - Matrix partitioning / replication → data locality



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



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



## Caching, Partitioning, and Indexing

#2 Buffer Pool Management

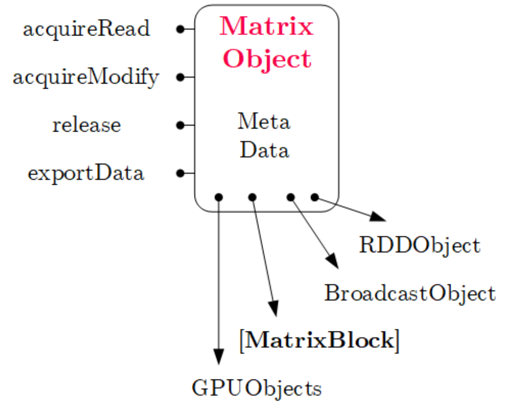
#3 Scan Sharing (and operator fusion)

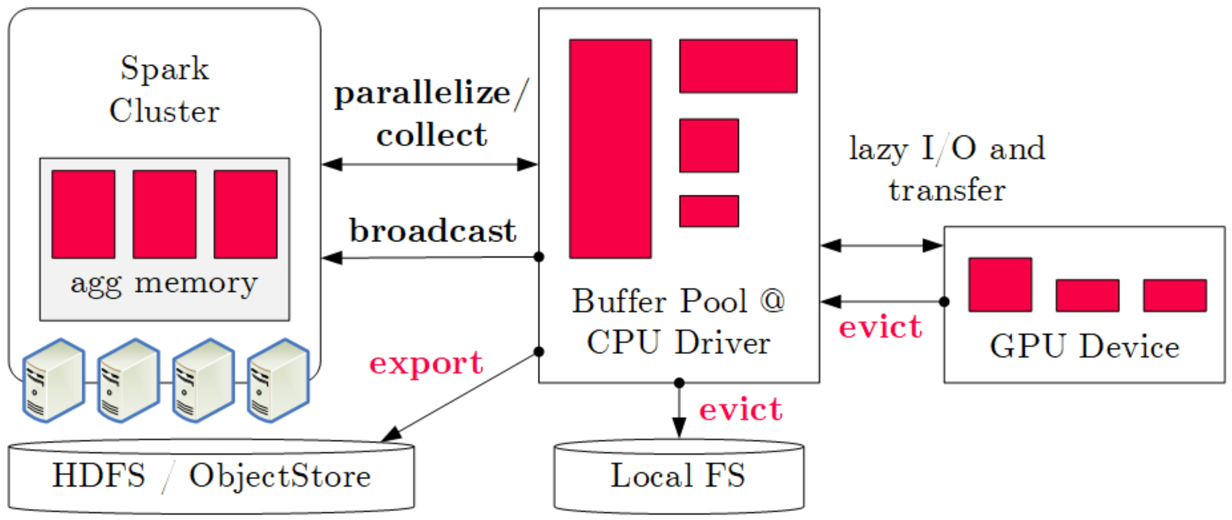
#4 NUMA-Aware Partitioning and Replication

#5 Index Structures

## Buffer Pool Management

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

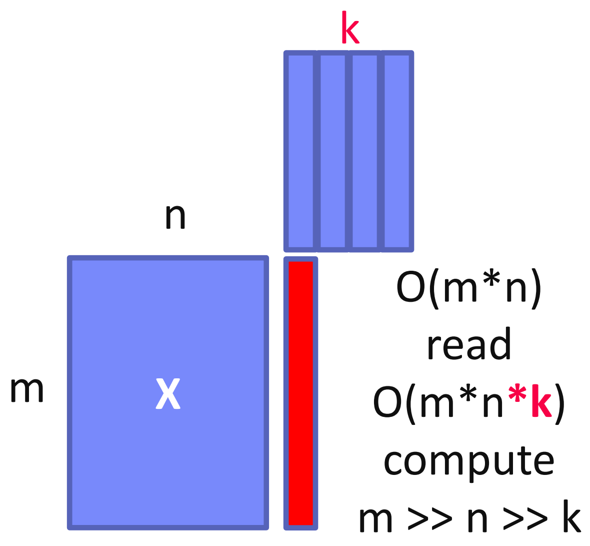




- **#2 Algorithm-Specific Buffer Management**
  - Operations/algorithms over out-of-core matrices and factor graphs
  - Examples: **RIOT** [CIDR'2009] (ops), **Elementary** [SIGMOD'13] (factor graphs)

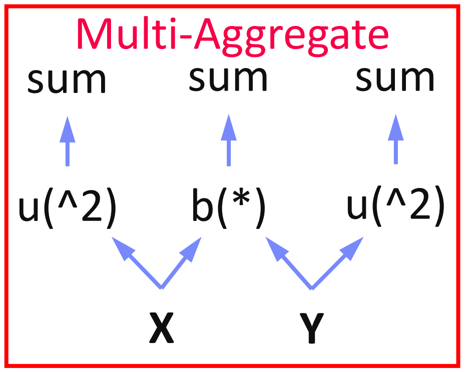
## Scan Sharing

- **#1 Batching**
  - One-pass evaluation of multiple configurations
  - Use cases: EL, CV, feature selection, hyper parameter tuning, multi-user scoring
  - 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** [PVLDB'18]

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



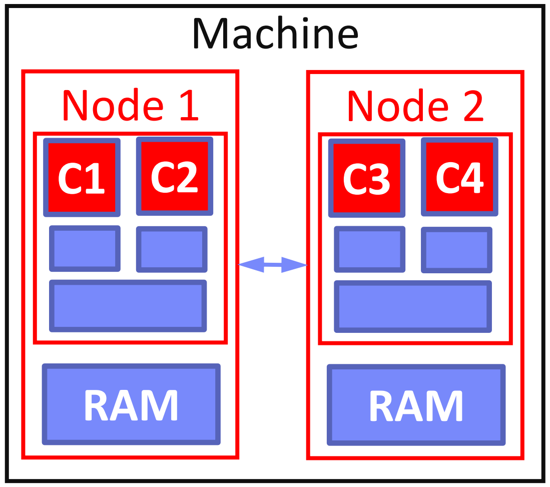
- **#3 Runtime Piggybacking**
  - Merge concurrent data-parallel jobs
  - "Wait-Merge-Submit-Return"-loop
  - E.g.: **SystemML parfor** [PVLDB'14]

```
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
- **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)

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



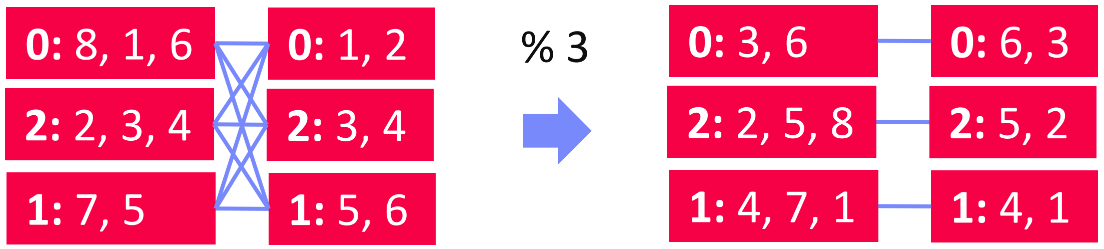
[David Kernert, Wolfgang Lehner, Frank Köhler: Topology-aware optimization of big sparse matrices and matrix multiplications on main-memory systems. **ICDE 2016**]

## 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**
  - R3 = R1.join(R2)



- **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-Tree Overview

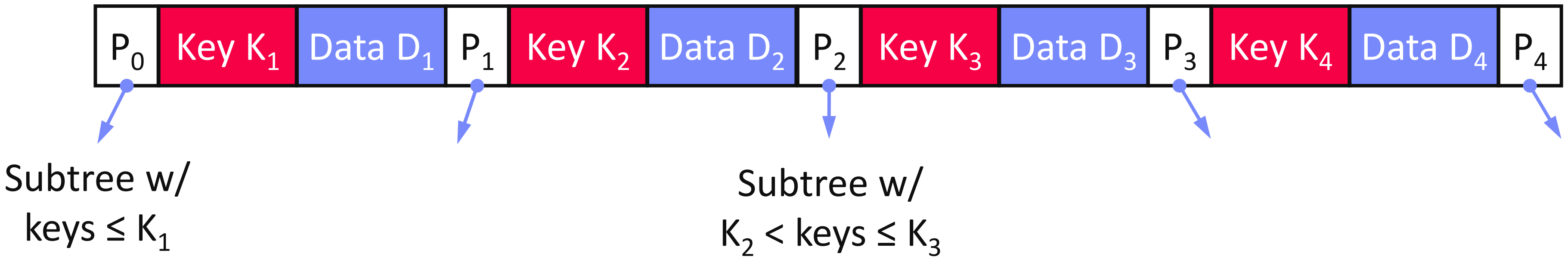
[Rudolf Bayer, Edward M. McCreight: Organization and Maintenance of Large Ordered Indices. **Acta Inf. (1) 1972**]

- **History B-Tree**
  - **B**ayer and McCreight 1972, **B**lock-based, **B**alanced, **B**oeing Labs
  - **Multiway tree** (node size = page size); designed for DBMS
  - Extensions: **B+-Tree/B*-Tree** (data only in leafs, double-linked leaf nodes)
- **Definition B-Tree (k, h)**
  - All paths from root to leafs have equal length h
  - All nodes (except root) 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)

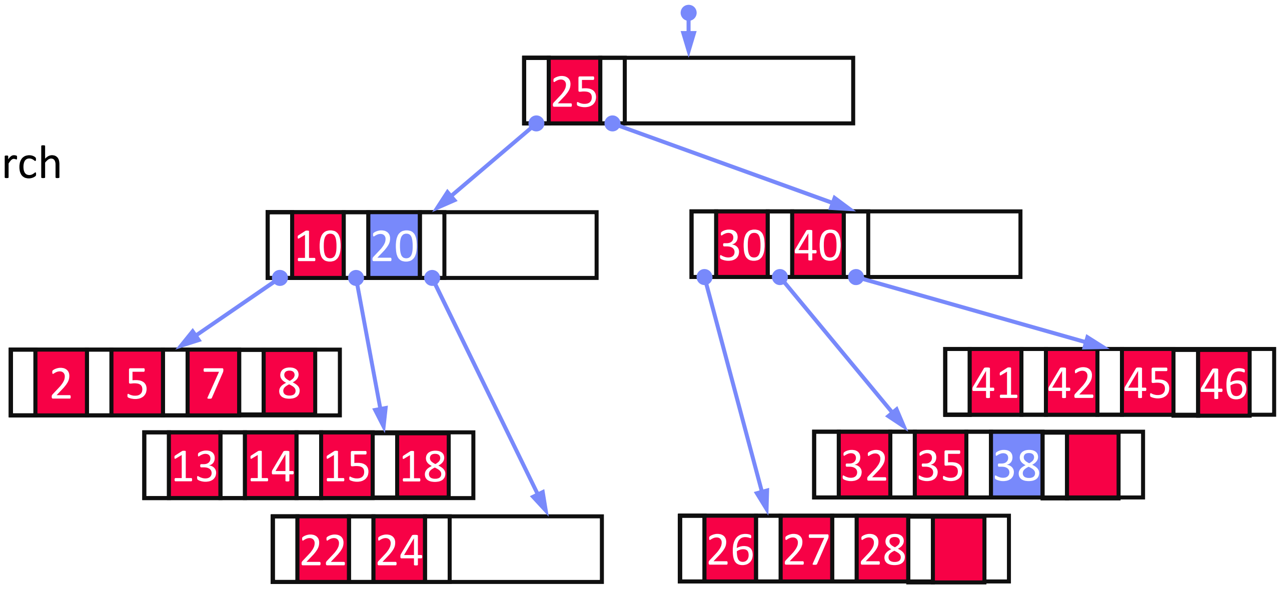
$$\lceil \log_{2k+1}(n+1) \rceil \leq h \leq \left\lceil \log_{k+1}\left(\frac{n+1}{2}\right) \right\rceil + 1$$

All nodes adhere to max constraints

**k=2**



## Recap: B-Tree Overview, cont.



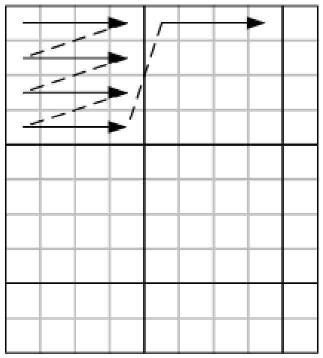
- **B-Tree Search**
  - Scan/binary search within nodes
  - Descend along matching key ranges
- **B-Tree Insertion**
  - Insert into leaf nodes
  - Split the 2k+1 entries into two leaf nodes
- **B-Tree Deletion**
  - Lookup key and delete if existing
  - Move entry from fullest successor; if underflow merge with sibling

## Linearized Array B-Tree (LAB-Tree)

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

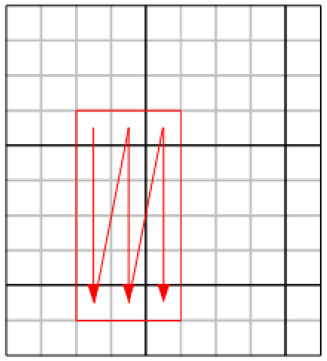
- **Basic Ideas**
  - B-tree over linearized array representation (e.g., row-/col-major, Z-order, UDF)
  - New leaf splitting strategies; dynamic leaf storage format (sparse and dense)
  - Various flushing policies for update batching (all, LRU, smallest page, largest page, largest page probabilistically,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

**#1** Example linearized storage order



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

**#2** Example linearized iterator order

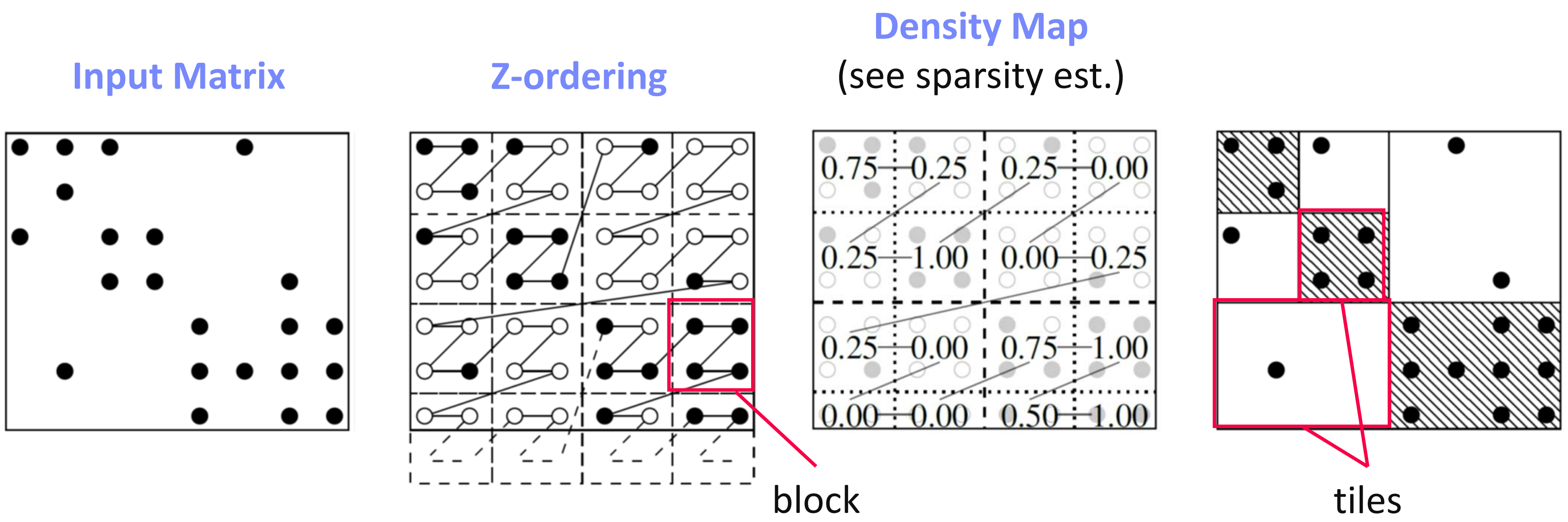


**range query A[4:9,3:5]**
with column-major
iterator order

## Adaptive Tile (AT) Matrix

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

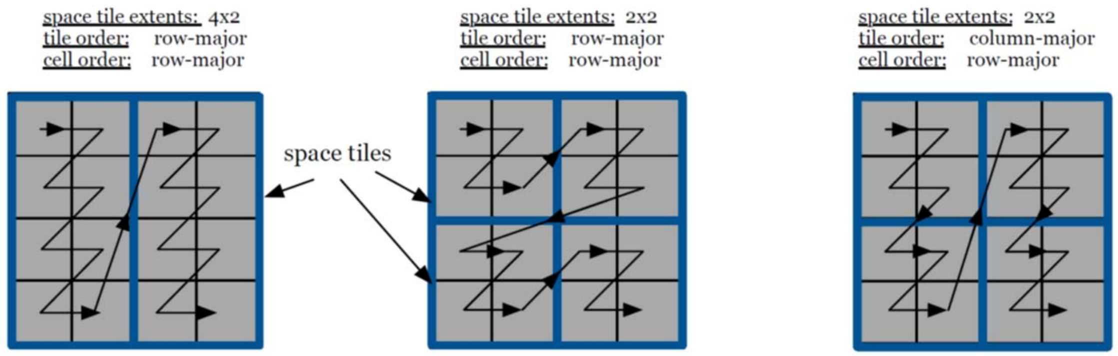
- **Basic Ideas**
  - Two-level blocking and NUMA-aware range partitioning (tiles, blocks)
  - Z-order linearization, and **recursive quad-tree partitioning** to find var-sized tiles (tile contains N blocks)

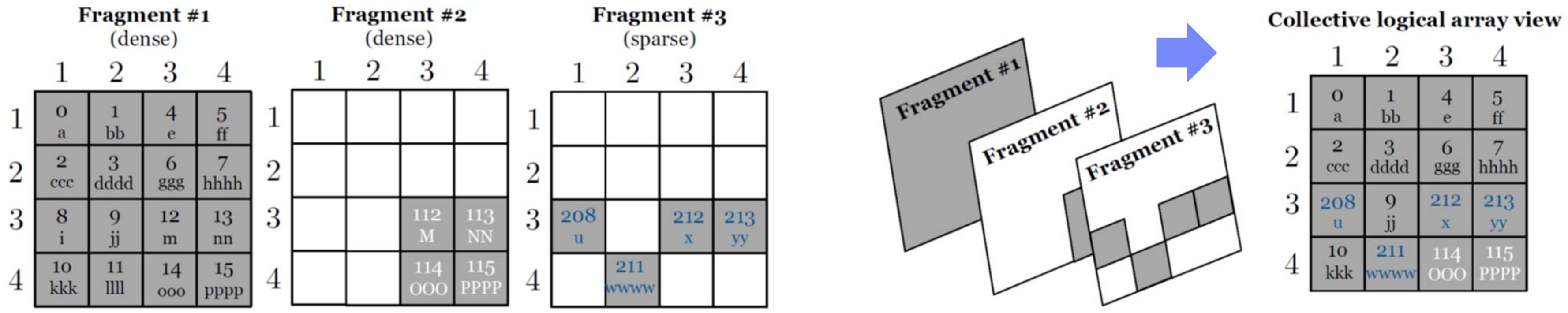


## TileDB Storage Manager

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

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



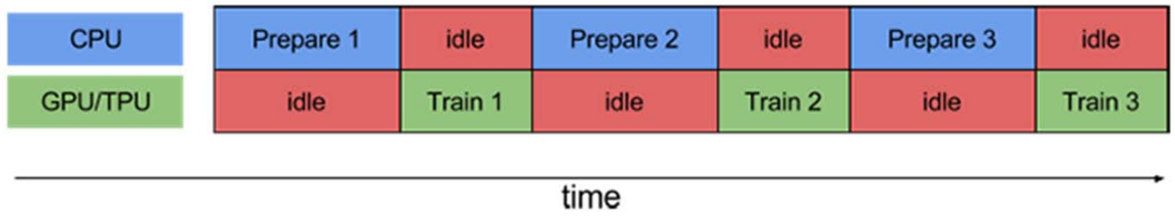


## Pipelining for Mini-batch Algorithms

- **Motivation**
  - Overlap data access and computation in mini-batch algorithms (e.g., DNN)

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

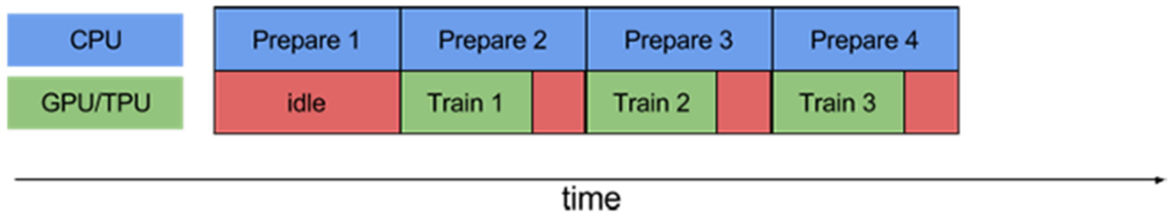
- **Example TensorFlow**
  - **#1** Queueing and Threading



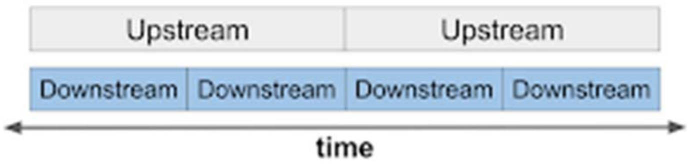
  - **#2 Dataset API Prefetching**

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

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



  - **#3** Reuse via **Data Echoing**



[https://ai.googleblog.com/2020/05/speeding-up-neural-network-training.html]

## Lossy and Lossless Compression

#6 Compression

## Recap: Database Compression Schemes

- **Null Suppression**
  - Compress integers by **omitting leading zero** bytes/bits (e.g., NS, gamma)

```
                                    106
00000000 00000000 00000000 01101010
                        11 01101010
```

- **Run-Length Encoding**
  - Compress sequences of equal values by **runs** of (value, start, run length)

```
1 1 1 1 7 7 7 7 7 3 3 3 3 3 3 ...
1,1,4   7,5,5     3,10,6
```

- **Dictionary Encoding**
  - Compress column w/ few distinct values as **pos in dictionary** (→ code size)

```
1 7 7 3 1 7 1 3 3 7 1 3 3 7 3 ...
1,3,7  dictionary (code size 2 bit)
1 3 3 2 1 3 1 2 2 3 1 2 2 3 2 ...
```

- **Delta Encoding**
  - Compress sequence w/ small changes by storing **deltas to previous value**

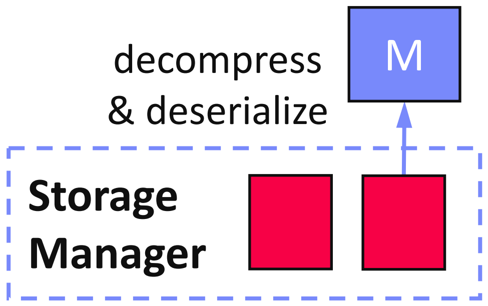
```
20 21 22 20 19 18 19 20 21 20 ...
 0  1  1 -2 -1 -1  1  1  1 -1...
```

- **Frame-of-Reference Encoding**
  - Compress values by storing **delta to reference value** (outlier handling)

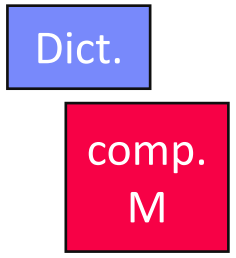
```
20 21 22 20 71 70 71 69 70 21 ...
21          70             22
-1  0  1 -1  1  0  1 -1  0 -1 ...
```

## Overview Lossless Compression Techniques

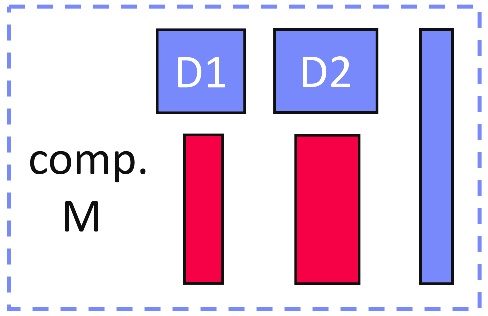
- **#1 Block-Level General-Purpose Compression**
  - Heavyweight or lightweight compression schemes
  - Decompress matrices block-wise for each operation
  - E.g.: Spark RDD compression (Snappy/LZ4), **SciDB** SM [SSDBM'11], **TileDB** SM [PVLDB'16], scientific formats **NetCDF**, **HDF5** at chunk granularity



- **#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) [SIGMOD'19]



- **#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]



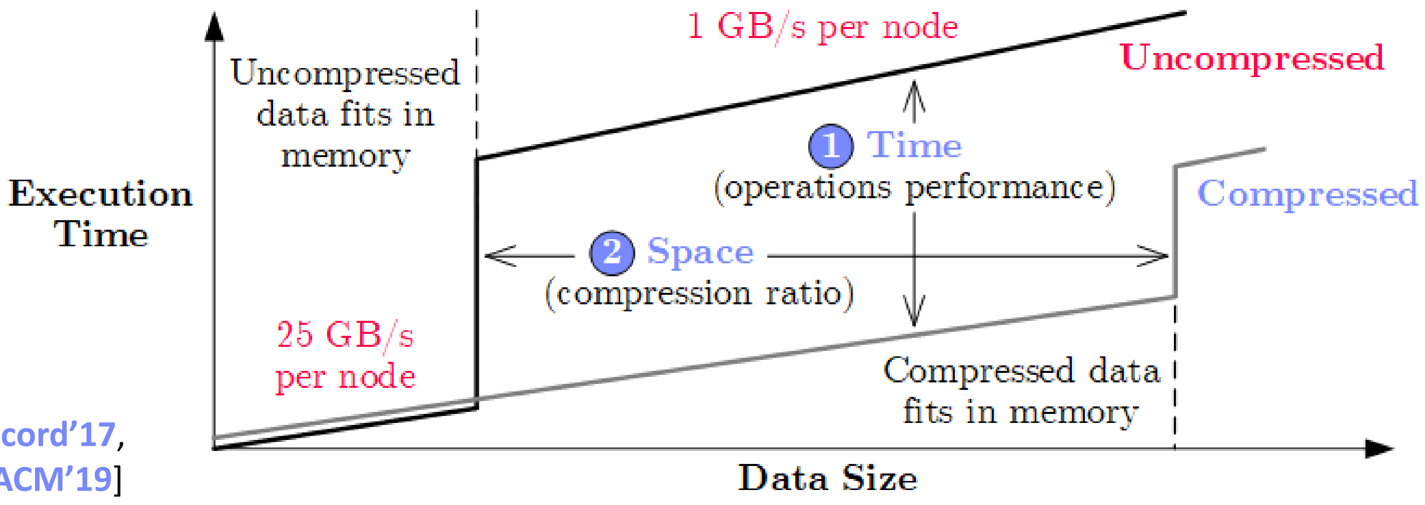
## CLA: Compressed Linear Algebra

[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**

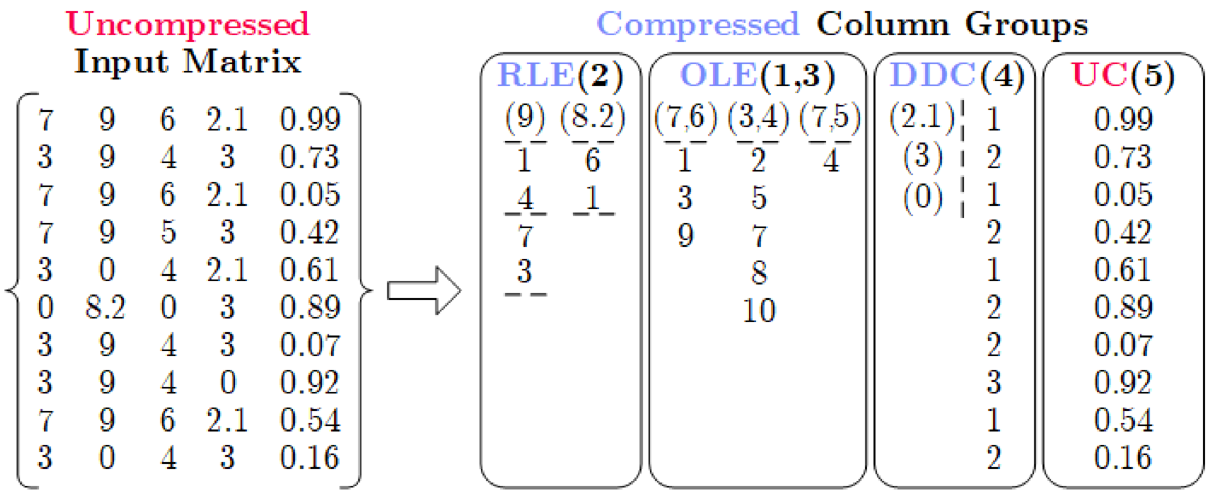
```
while(!converged) {
   … q = X %*% v …
}
```



[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)



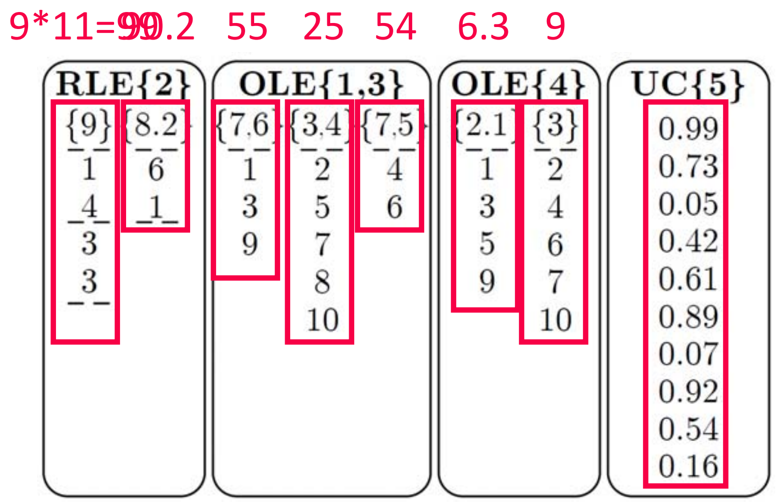
* DDC1/2 in VLDBJ'17

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

## CLA: Compressed Linear Algebra, cont. (3)

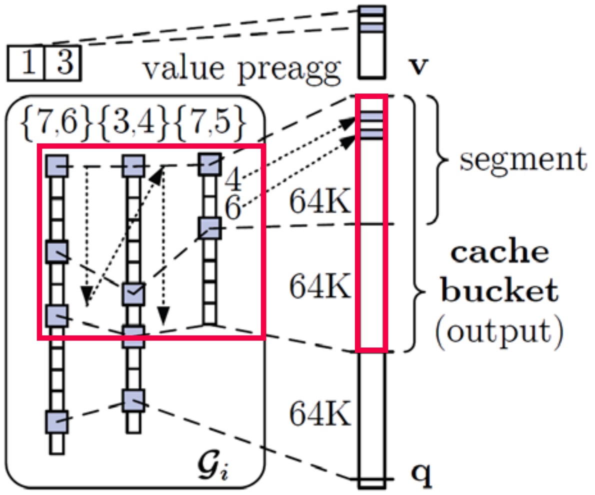
- **Matrix-Vector Multiplication**
  - Naïve: for each tuple, pre-aggregate values, add values at offsets to q

    Example: q = X v, with v = (7, 11, 1, 3, 2)



| |
|---|
| 162.3 |
| 134.5 |
| 160.4 |
| 162.8 |
| 32.5 |
| 155 |
| 133.1 |
| 125.8 |
| 161.4 |
| 34.3 |

➔ cache unfriendly on output (q)

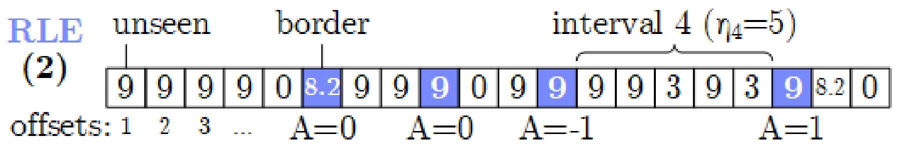


  - Cache-conscious: Horizontal, segment-aligned scans, maintain positions

- **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

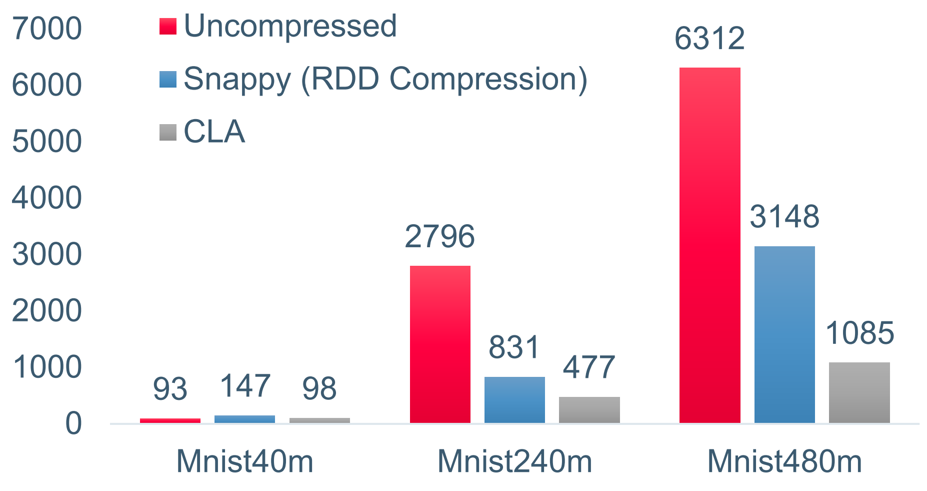
## CLA: Compressed Linear Algebra, cont. (5)

- **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]



90GB 540GB 1.1TB

- **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

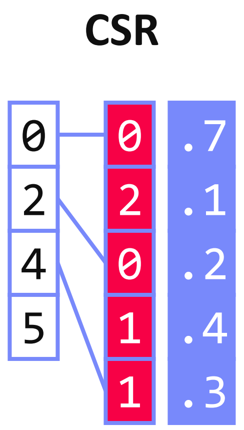
- **CSR-VI (CSR-Value Indexed) / D-VI**
  - **Create dictionary** for distinct values
  - **Encode 8 byte values as 1, 2, or 4-byte codes** (positions in the dictionary)
  - Extensions w/ delta coding of indexes
  - Example CSR-VI matrix-vector multiply
    **c = A %*% b**

[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]];
}
```

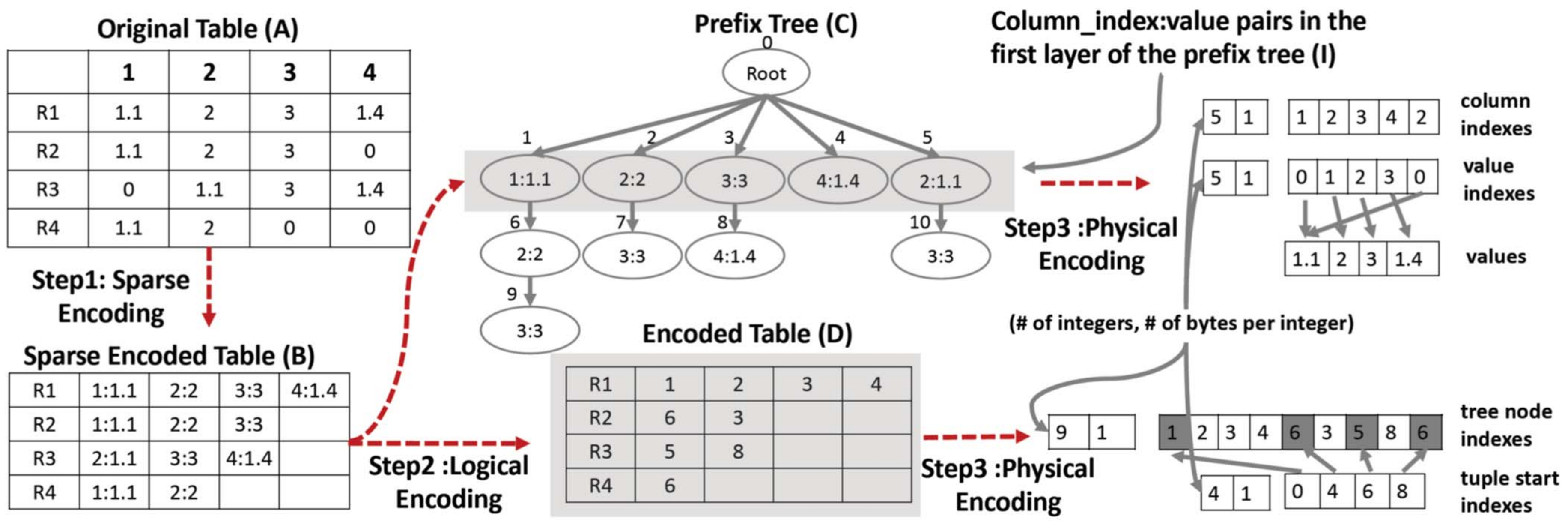
**value decoding**
(MV over compressed representation)



# Tuple-oriented Compression (TOC)

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

[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**]



Original Table (A)

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| R1 | 1.1 | 2 | 3 | 1.4 |
| R2 | 1.1 | 2 | 3 | 0 |
| R3 | 0 | 1.1 | 3 | 1.4 |
| R4 | 1.1 | 2 | 0 | 0 |

Sparse Encoded Table (B)

| | | | | |
|---|---|---|---|---|
| R1 | 1:1.1 | 2:2 | 3:3 | 4:1.4 |
| R2 | 1:1.1 | 2:2 | 3:3 | |
| R3 | 2:1.1 | 3:3 | 4:1.4 | |
| R4 | 1:1.1 | 2:2 | | |

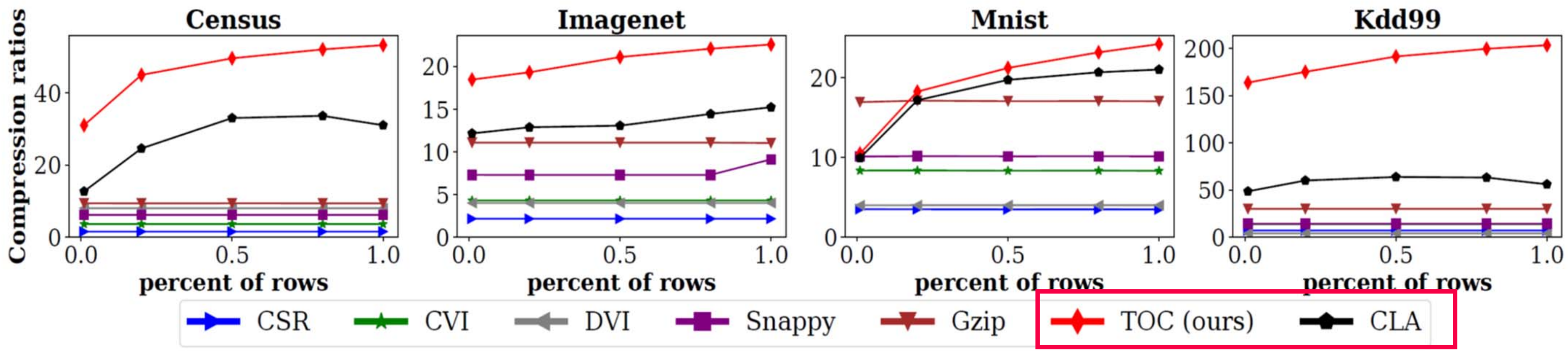
Encoded Table (D)

| | | | | |
|---|---|---|---|---|
| R1 | 1 | 2 | 3 | 4 |
| R2 | 6 | 3 | | |
| R3 | 5 | 8 | | |
| R4 | 6 | | | |

## Tuple-oriented Compression (TOC), cont.

- **Example**
  **Compression Ratios**

[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**]



**Take-away:** specialized lossless matrix compression
➔ **reduce memory bandwidth requirements and #FLOPs**

## 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]

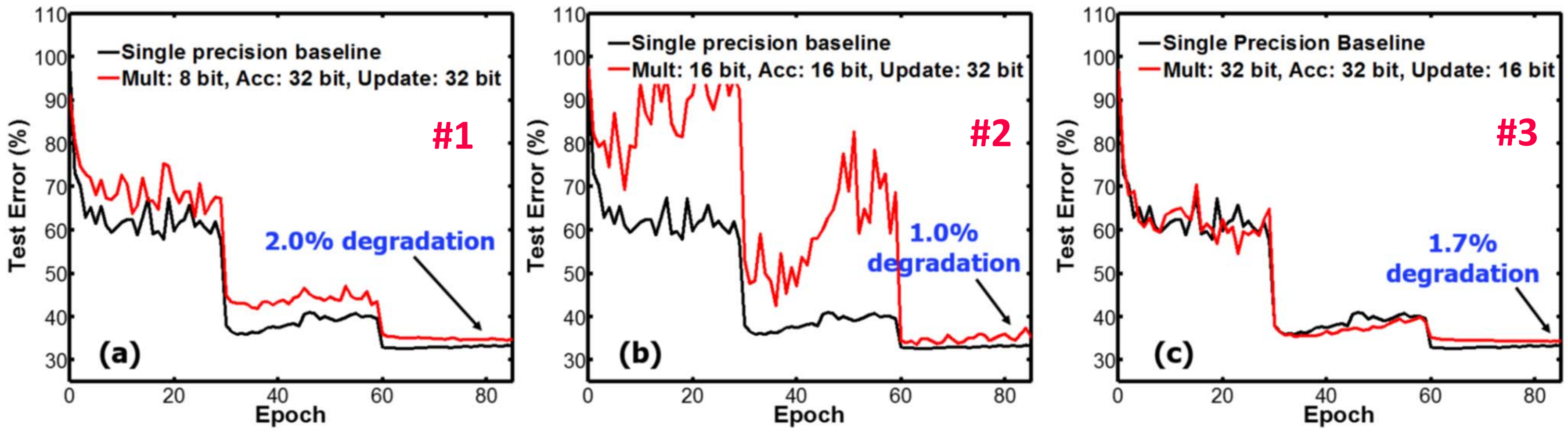
## Low and Ultra-low FP Precision

- **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

- **Example ResNet18 over ImageNet**

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



Test Error (%)
Epoch
Single precision baseline
Mult: 8 bit, Acc: 32 bit, Update: 32 bit
#1
2.0% degradation
(a)
Single precision baseline
Mult: 16 bit, Acc: 16 bit, Update: 32 bit
#2
1.0% degradation
(b)
Single Precision Baseline
Mult: 32 bit, Acc: 32 bit, Update: 16 bit
#3
1.7% degradation
(c)

## Low and Ultra-low FP Precision, cont.

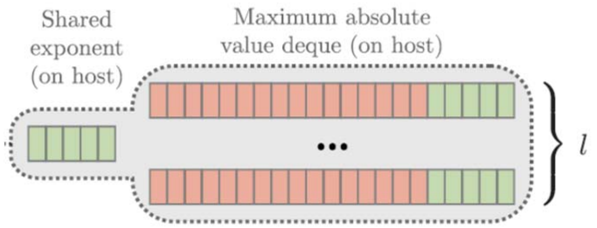
- **Numerical Stable Accumulation**
  - **#1 Sorting ASC + Summation** (accumulate small values first)
  - **#2 Kahan Summation** w/ error independent of number of values n

```
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` / Google `bfloat16`**
  - Blocks of values w/ shared exponent (16bit w/ 5bit shared exponent)


Shared exponent (on host)
Maximum absolute value deque (on host)
...
$l$

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

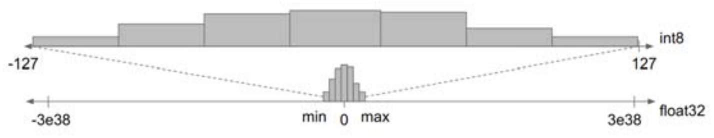
## Fixed-Point Arithmetic

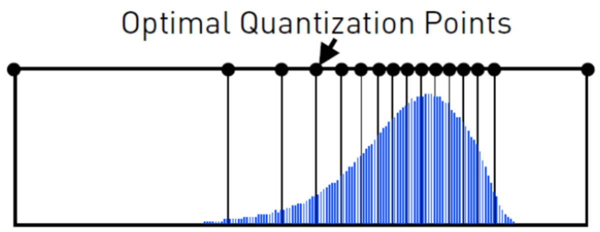
Recommended "Reading"
[Inside TensorFlow: Model Optimization Toolkit (Quantization and Pruning), **YouTube, 2020**]

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

- **#1 Quantization** (reduce value domain)
  - **Split value domain into N buckets** such that k = $\log_2$ N can encode the data
  - **a) Static Quantization** (e.g., min/max) per tensor or per tensor channel
  - **b) Learned Quantization** Schemes
    - Dynamic programming
    - Various heuristics
    - Example systems: **ZipML**, **SketchML**

[https://blog.tensorflow.org/2020/04/quantization-aware-training-with-tensorflow-model-optimization-toolkit.html]





[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 Techniques

[https://blog.tensorflow.org/2019/05/tf-model-optimization-toolkit-pruning-API.html]

- **#2 Sparsification/Pruning** (reduce #non-zeros)
  - **Value clipping:** zero-out very small values below a threshold to reduce size of weights
  - **Training w/ target sparsity:** remove connections

| Sparse Accuracy | NNZ |
|---|---|
| 78.1% @ sp=1.0 | 27.1M |
| 78.0% @ sp=0.5 | 13.6M |
| 76.1% @ sp=0.25 | 6.8M |
| 74.6% @ sp=0.125 | 3.3M |

- **#3 Mantissa Truncation**
  - Truncate m of FP32 from 23bit to 16bit
  - E.g., **TensorFlow** (transfers), **Pstore**

[Souvik Bhattacherjee et al: PStore: an efficient storage framework for managing scientific data. **SSDBM 2014**]

- **#4 Aggregated Data Representations**
  - a) Dim reduction (e.g., auto encoders)
  - b) No FK-PK joins in Factorized Learning (**foreign key** as lossy compressed rep)

[Amir Ilkhechi et al: DeepSqueeze: Deep Semantic Compression for Tabular Data, **SIGMOD 2020**]

[Arun Kumar et al: To Join or Not to Join?: Thinking Twice about Joins before Feature Selection. **SIGMOD 2016**]

- **#5 Sampling**
  - User specifies **approximation contract** for error (regression/classification) and scale
  - Min sample size for **max likelihood estimators**

[Yongjoo Park et al: BlinkML: Efficient Maximum Likelihood Estimation with Probabilistic Guarantees. **SIGMOD 2019**]

## Summary and Conclusions

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

- **Next Lectures**
  - **May 21/22:** Ascension Day (Christi Himmelfahrt)
  - **09 Data Acquisition, Cleaning, and Preparation** [May 29]
  - **10 Model Selection and Management** [Jun 05]
  - **11 Model Debugging Techniques** [Jun 12]
  - **12 Model Serving Systems and Techniques** [Jun 19]

(**Part B:** ML Lifecycle Systems)