Sparsity - 2

EECE695D: Efficient ML Systems

Spring 2025

Agenda

- Another approach for mask optimization
- Why would sparse models work?
- System considerations for sparsity
    - Unstructured sparsity
    - Structured sparsity

Another approach for mask optimization

Problem

- **Recall.** In the last class, we discussed a heuristic method to solve:

$$\begin{aligned} &\text{minimize}_{\mathbf{m},\mathbf{w}} \quad \hat{L}(\mathbf{m} \odot \mathbf{w}) \\ &\text{subject to} \quad \|\mathbf{m}\|_0 \leq \tau, \quad m_{ij} \in \{0,1\} \end{aligned}$$

- **Challenge.** Optimizing the discrete mask $\mathbf{m}$
  - Constrained optimization
  - Discrete optimization

Relaxation

- **Idea.** Remove the constraint by considering the Lagrangian relaxation

$$\hat{L}(\mathbf{m} \odot \mathbf{w}) + \lambda \|\mathbf{m}\|_0, \qquad m_i \in \{0,1\}$$

  - Tune λ to meet the sparsity constraint τ

- **Next challenge.** Optimizing discrete variables with a less heuristic way
  - Also common in other domains

- We will illustrate a simple approach by Srinivas et al. (2017)

Srinivas et al., "Training sparse neural networks," CVPR workshop 2017

Probabilistic gate

- **Idea.** Model $\mathbf{m}$ as a random vector with a latent variable
  - Example. Simply use

    $$m_i \sim \mathrm{Bern}(z_i)$$

    and optimize the continuous $\mathbf{z}$.
  - The optimand will then be:

    $$\mathbb{E}[\hat{L}(\mathbf{m} \odot \mathbf{w})] + \lambda \cdot \mathbb{E}\|\mathbf{m}\|_0 \quad = \quad \mathbb{E}[\hat{L}(\mathbf{m} \odot \mathbf{w})] + \lambda \|\mathbf{z}\|_1$$

    - Use Monte Carlo approach — sample $\mathbf{m}$ and optimize.

Probabilistic gate

- **Problem.** How do we compute the gradient for $\mathbf{z}$ w.r.t. the first term?

$$\mathbb{E}[\hat{L}(\mathbf{m} \odot \mathbf{w})] + \lambda \|\mathbf{z}\|_1$$

  - Solution. Simply ignore the gradient; pretend if we have

$$\frac{\partial m_i}{\partial z_i} = 1$$

    - This trick has a fancy name, called straight-through estimator (STE)
      - We will come back to this, in quantization lectures
      - Not really an unbiased estimate, but good enough

Other fixes

- **Problem#1.** We want $\|\mathbf{z}\|$ to be close to 0 or 1.
  - Solution. Add a regularizer, to make

$$\hat{L}(\mathbf{m} \odot \mathbf{w}) + \lambda_1 \|\mathbf{z}\|_1 + \lambda_2 \sum_i z_i(1 - z_i)$$

- **Problem#2.** How do we keep $\mathbf{z}_i \in [0,1]$?
  - Solution. Assume that there is yet another latent $\mathbf{u}$, such that

$$z_i = \text{sigmoid}(u_i)$$

Further readings

- More popular form is based on binary concrete distribution (a.k.a. Gumbel-softmax), instead of the Bernoulli distribution

    - Louizos et al., "Learning Sparse Neural Networks through L_0 regularization," ICLR 2017 (link)

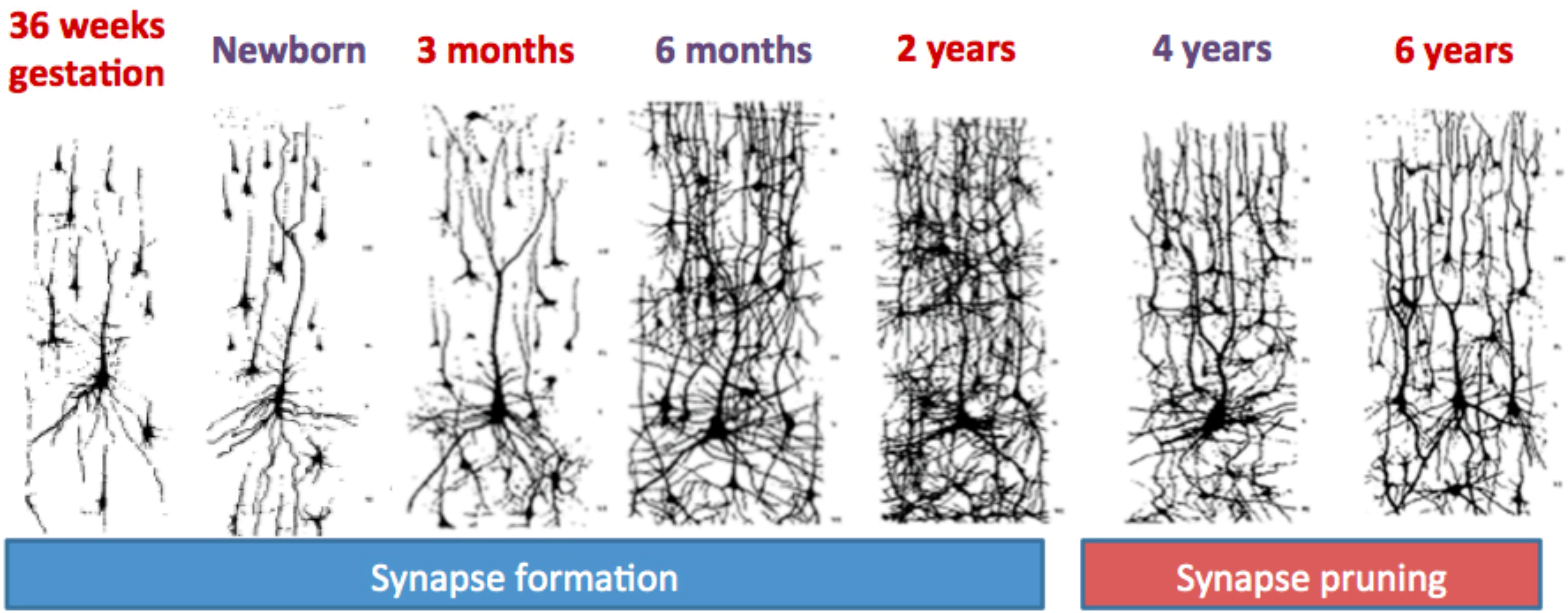
Why do sparse nets work?

Why should it work?

- **Question.** Why do we expect sparse models to work as well as dense models?
  - **Answer.** No concrete justification 🥲
  - Nevertheless, there are some motivations…

Why should it work?

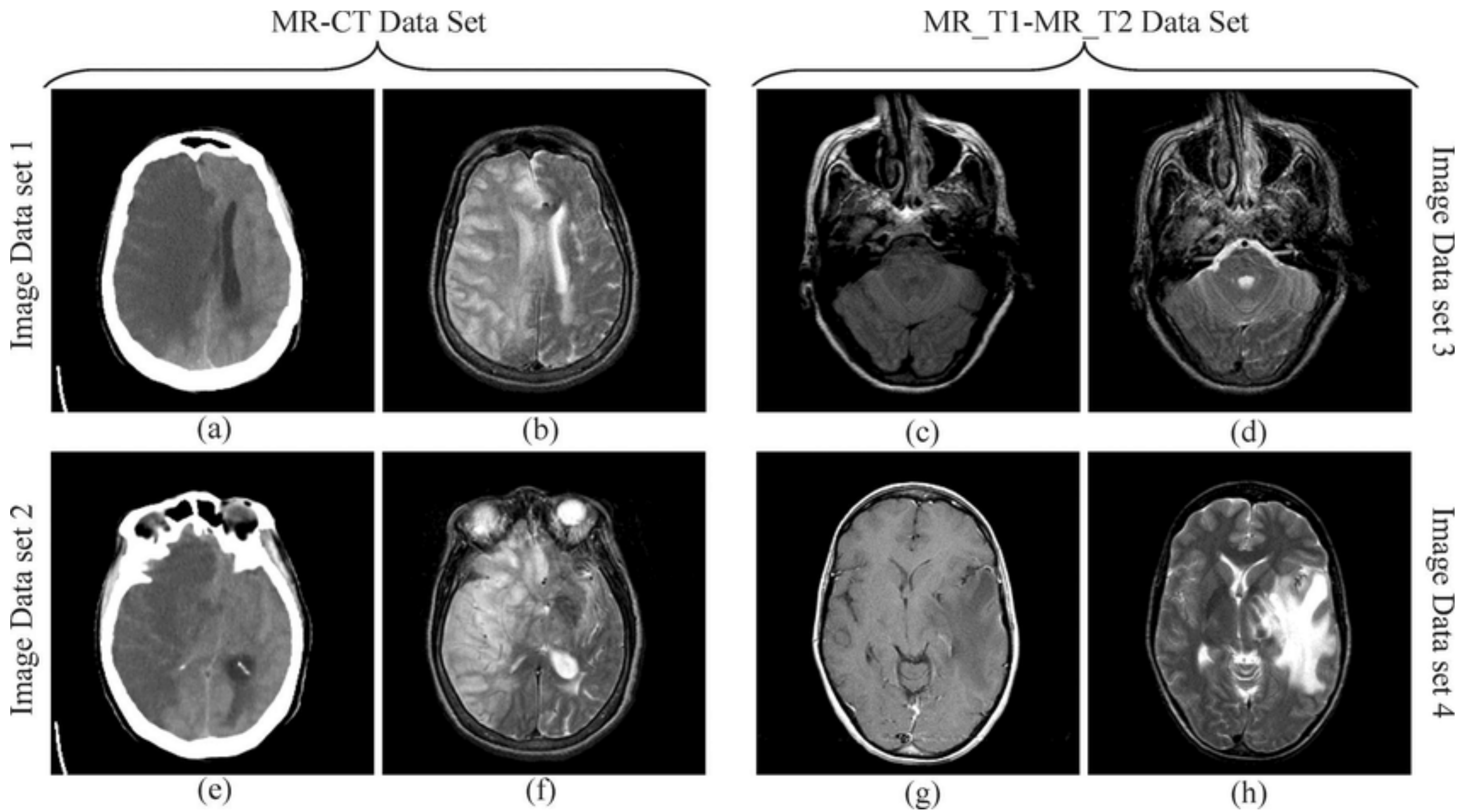
- **Biological motivation.** Human brain also does some sort of pruning.



C. A. Walsh, "Peter Huttenlocher (1931–2013)," Nature, 2013

Why should it work?

- **Natural sparsity.** Many natural data or relationships are actually sparse
  - e.g., simply irrelevant input features



Why should it work?

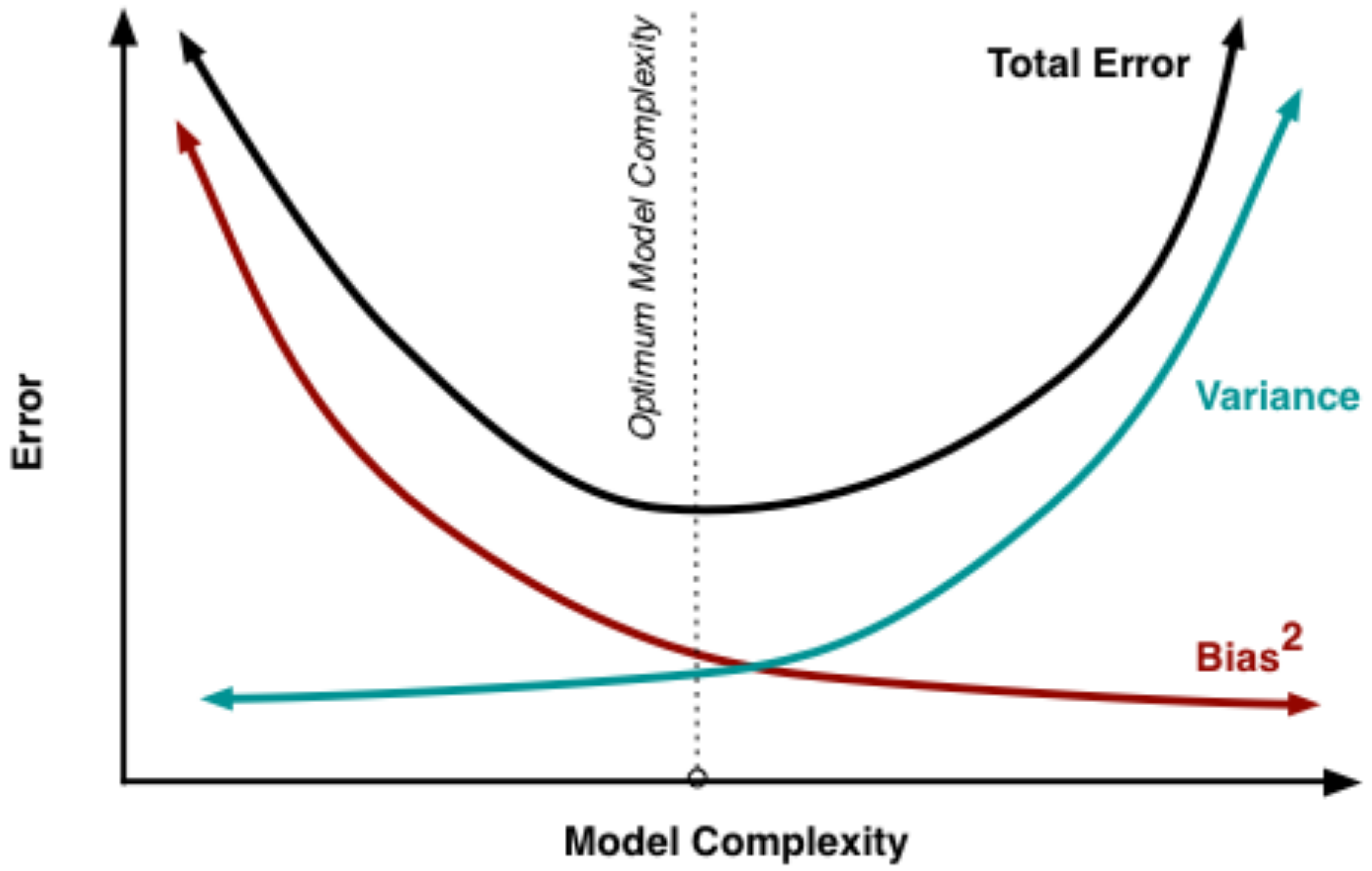
- **Theoretical guarantees.** We use much more parameters than what is theoretically sufficient.
  - We need only $\tilde{O}(\sqrt{N})$ weights to achieve zero training loss on N samples.

Theorem 1.1 (informal statement). *Let $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N) \in \mathbb{R}^d \times \{1, \ldots, C\}$ be a set of N labeled samples of a constant dimension d, with $\|\mathbf{x}_i\| \leq r$ for every i and $\|\mathbf{x}_i - \mathbf{x}_j\| \geq \delta$ for every $i \neq j$. Then, there exists a ReLU neural network $F : \mathbb{R}^d \to \mathbb{R}$ with width 12, depth $\tilde{O}\left(\sqrt{N}\right)$, and $\tilde{O}\left(\sqrt{N}\right)$ parameters, such that $F(\mathbf{x}_i) = y_i$ for every $i \in [N]$, where the notation $\tilde{O}(\cdot)$ hides logarithmic factors in N, C, r, δ^{-1}.*

Vardi et al., "On the Optimal Memorization Power of ReLU Neural Networks," ICLR 2022

Why should it work?

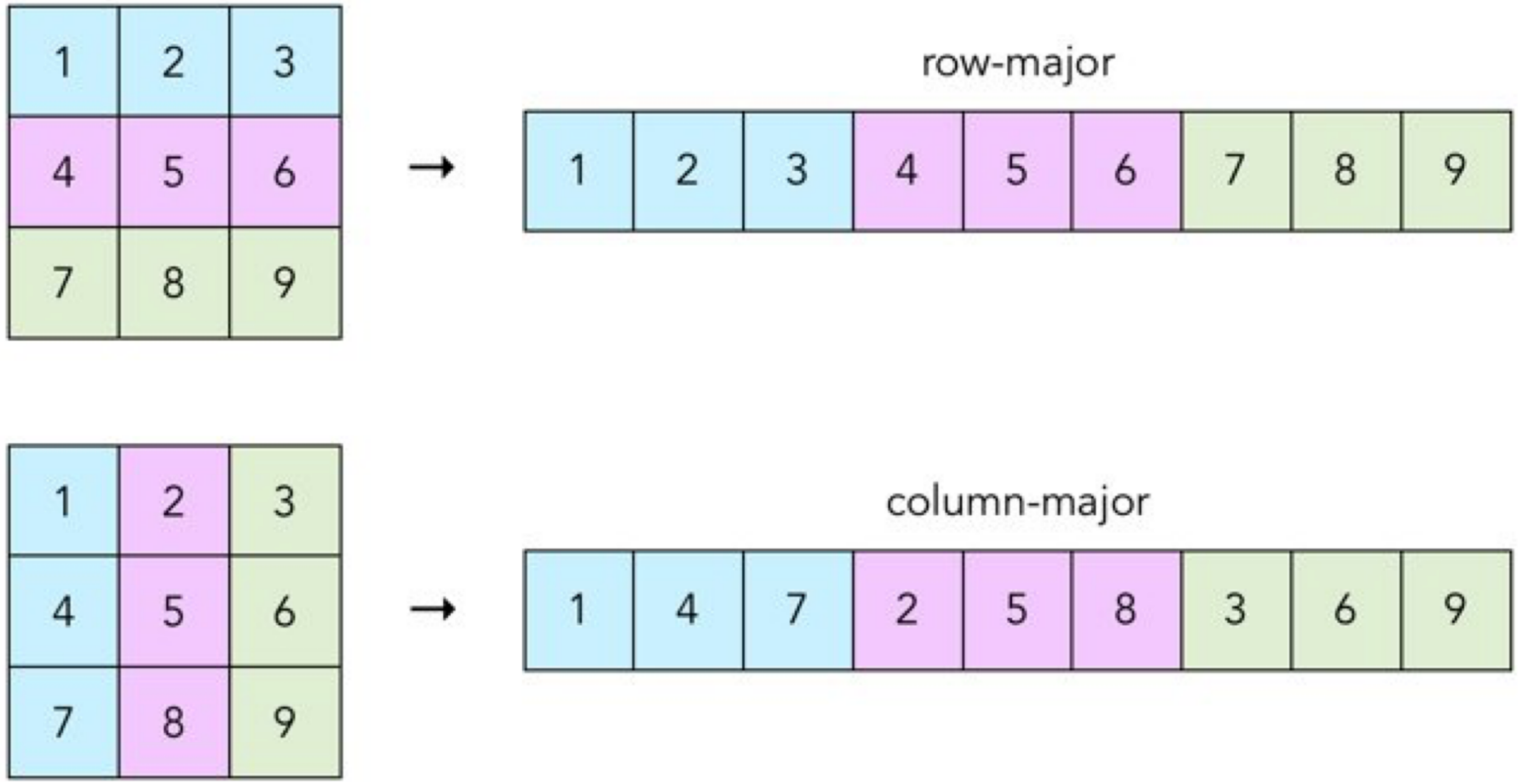
- **Generalization (depracated).** In the past, it was believed that less parameters will lead to better generalization, by avoiding overfitting.
    - This no longer seems to be a valid logic, and is empirically not true.



System considerations: Unstructured sparsity

Recap: Processing Dense Matrices

- Matrices are usually stored in either:
  - **Row-major.** C, NumPy, PyTorch, …
  - **Column-major.** MATLAB, Julia, Fortran, …



https://craftofcoding.wordpress.com/2017/02/03/column-major-vs-row-major-arrays-does-it-matter/

Recap: Processing Dense Matrices

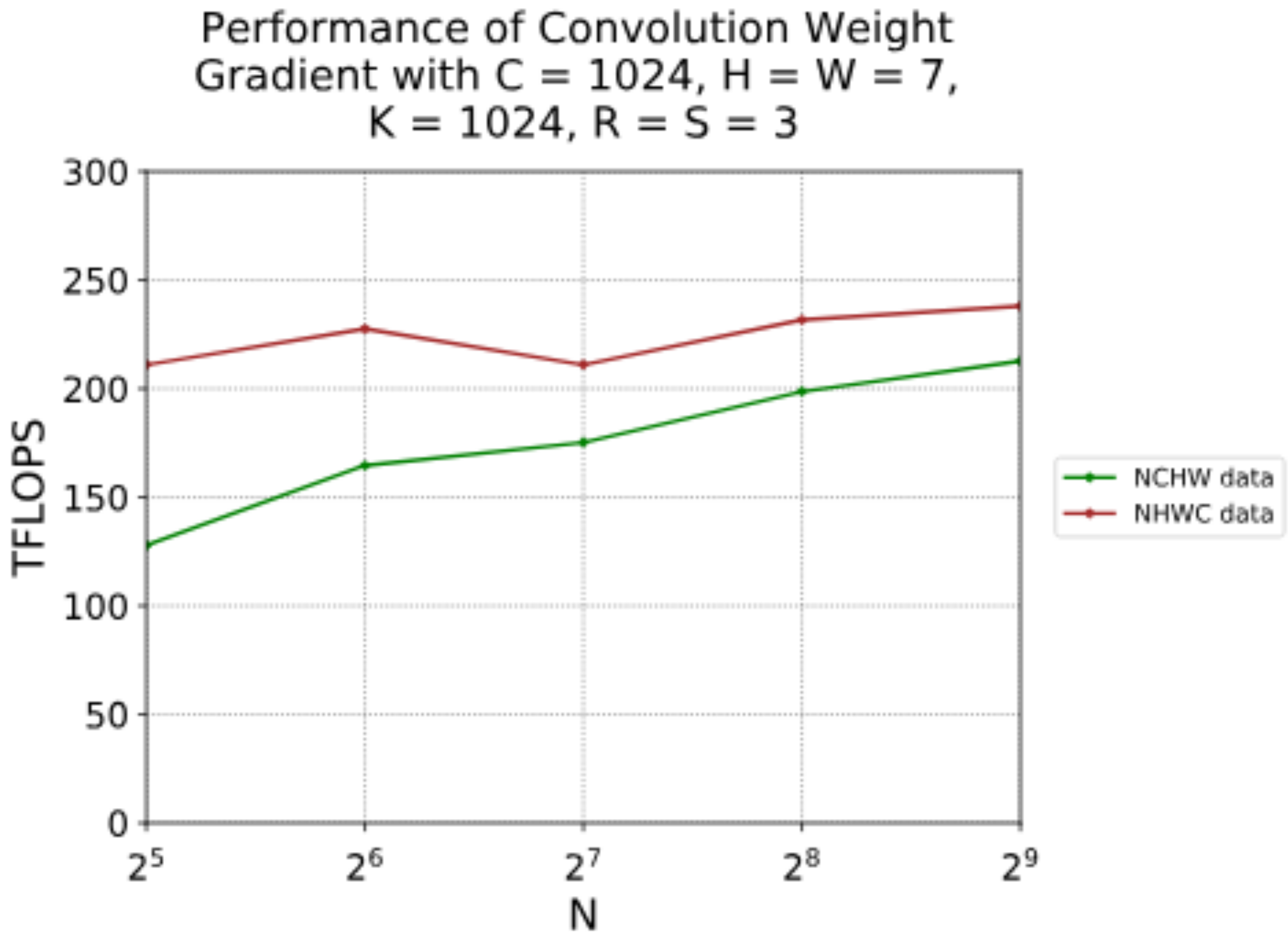
- The storage format affects the runtime & arithmetic intensity
- **Reason 1.** Alters the memory access pattern
  - Example. If the matrix A is in row-major, which code will run faster? (on CPU, one is 15x faster than another; see link)

```
// loop1 accesses data in matrix 'a' in row major order,
// since i is the outer loop variable, and j is the
// inner loop variable.
int loop1(int a[4000][4000]) {
 int s = 0;
 for (int i = 0; i < 4000; ++i) {
   for (int j = 0; j < 4000; ++j) {
     s += a[i][j];
   }
 }
 return s;
}
```

```
// loop2 accesses data in matrix 'a' in column major order
// since j is the outer loop variable, and i is the
// inner loop variable.
int loop2(int a[4000][4000]) {
 int s = 0;
 for (int j = 0; j < 4000; ++j) {
   for (int i = 0; i < 4000; ++i) {
     s += a[i][j];
   }
 }
 return s;
}
```

Recap: Processing Dense Matrices

- **Reason 2.** Some HWs and kernels are customized for certain formats
  - Example. For conv2d, tensor core implementations are written for NHWC while PyTorch default is NCHW (link)

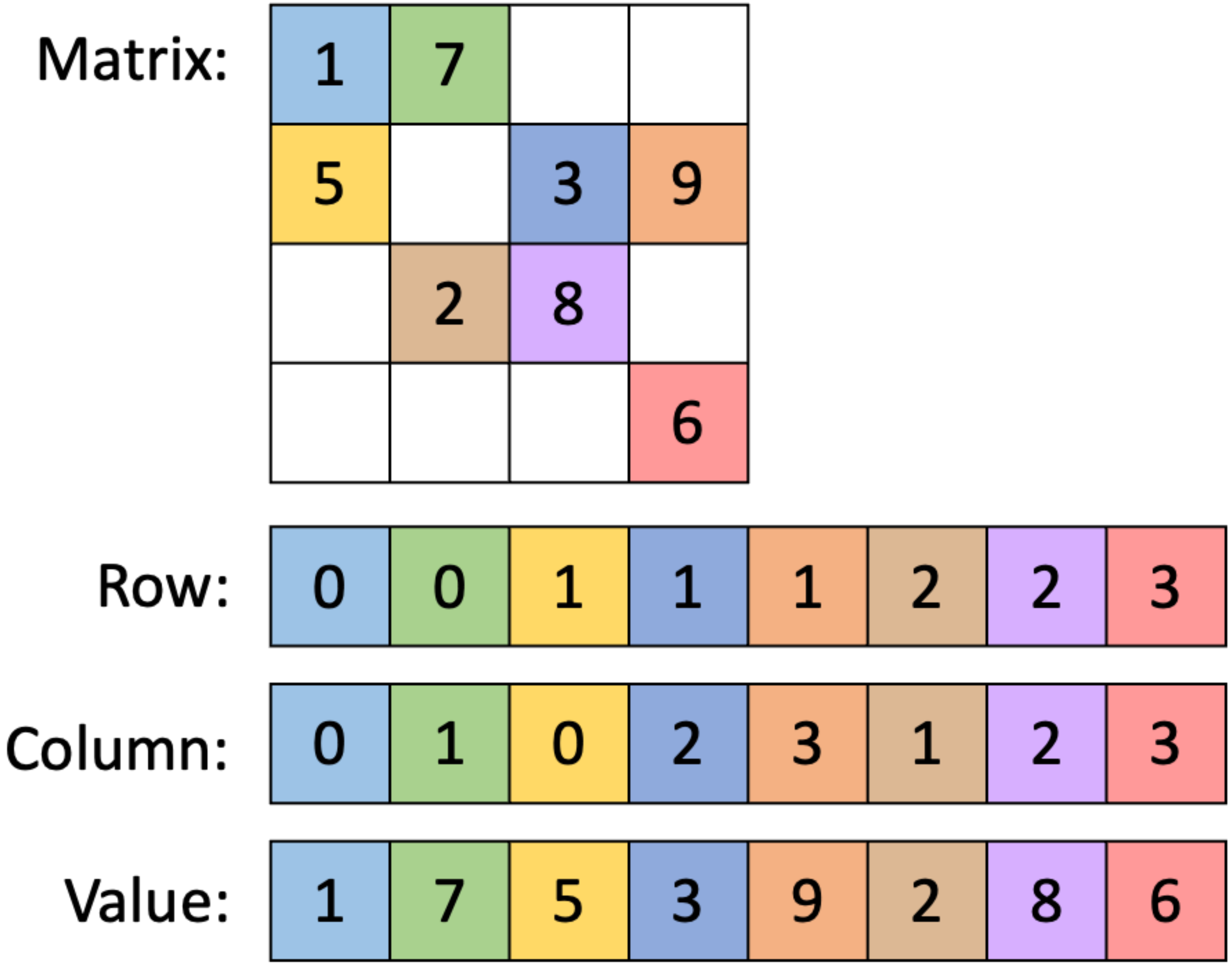


Sparse matrices, unstructured

- There are various formats to store unstructured sparse matrices
  - Unstructured: no designated patterns on 0s.
  - Quick look at two popular options: **COO**, **CSR**
    - Different pros & cons
      - SpMV (Sparse Matrix-Vector Mult.)
      - Storage

COO (Coordinate)

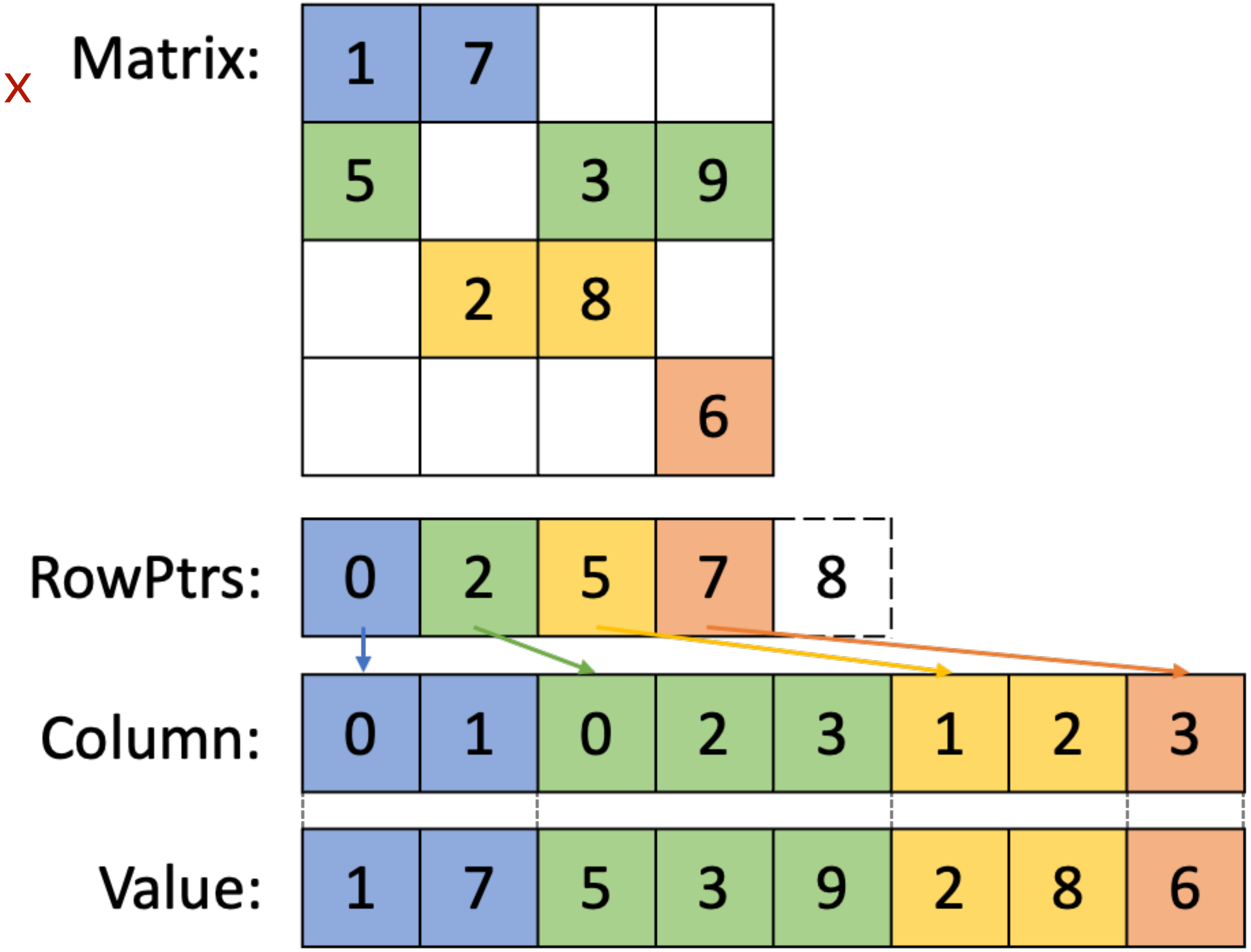
- For each nonzero, store (row, col, val) separately
- Flexible editing
- PyTorch default



Hwu et al., "Programming Massively Parallel Processors," Elsevier, 2022

CSR (Compressed Sparse-Row)

- For each nonzero, store (col, val) with the pointers for the column idx where each row starts at
- cuSPARSE default



Hwu et al., "Programming Massively Parallel Processors," Elsevier, 2022

Storage

- Suppose that we have an **NxN matrix** with **K nonzero** elements.
- Suppose that we use **COO**
  - **Val.** K Bytes (if using INT8)
  - **Col.** K Bytes (2K if 256 < N < 65536)
  - **Row.** K Bytes (2K if 256 < N < 65536)

  ⇒ 3K Bytes

- If Sparsity $\geq$ 66.6%, we are good.

Matrix:

1	7		
5		3	9
	2	8	
			6

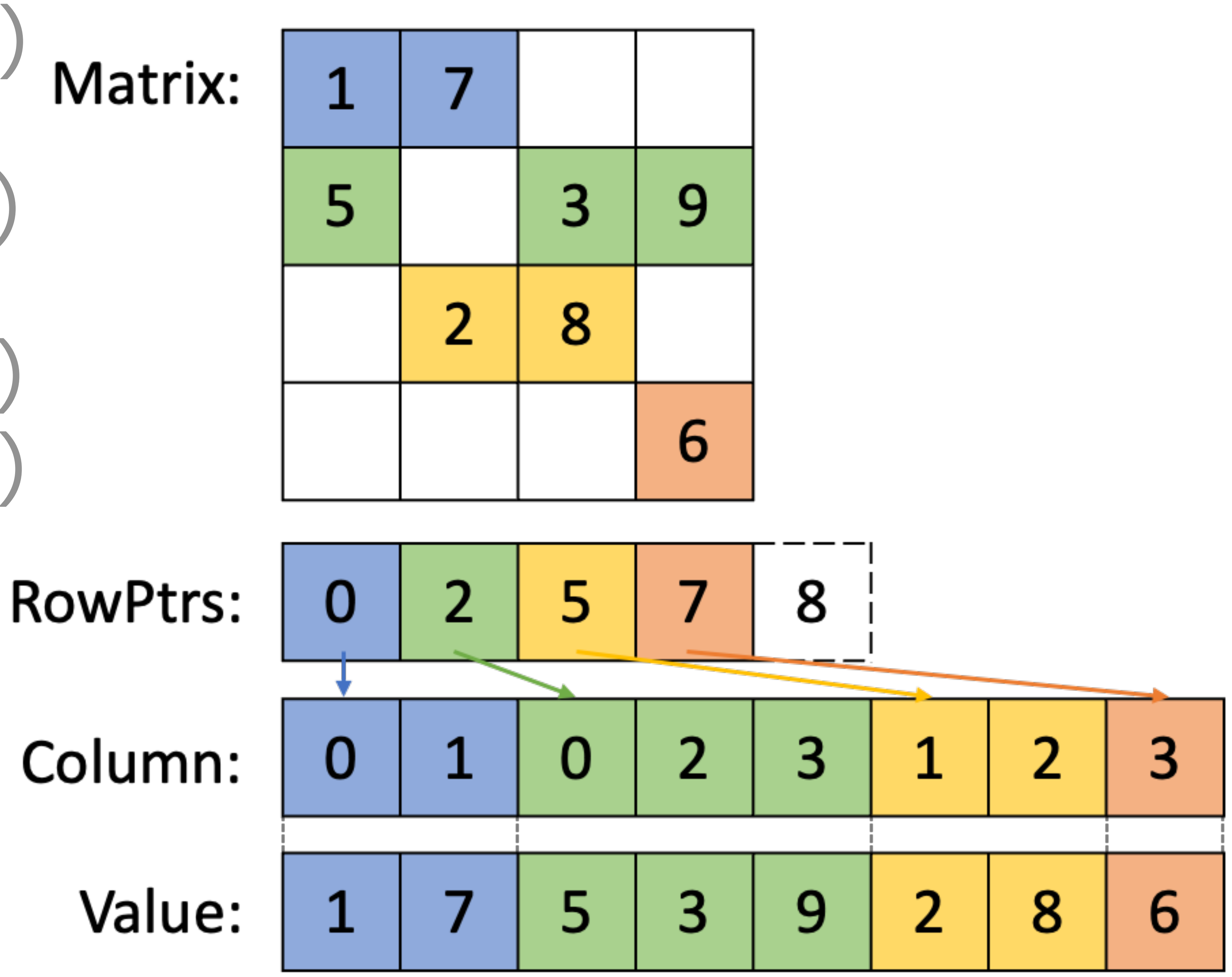
Row:	0	0	1	1	1	2	2	3
Column:	0	1	0	2	3	1	2	3
Value:	1	7	5	3	9	2	8	6

Storage

- Consider the case of **CSR**
  - **Val.** K Bytes (if using INT8)
  - **Col.** K Bytes (2K if 256 < N <= 65536)
  - **Row.** 2N Bytes (if 256 < K <= 65536)
    N Bytes (if K <= 256)

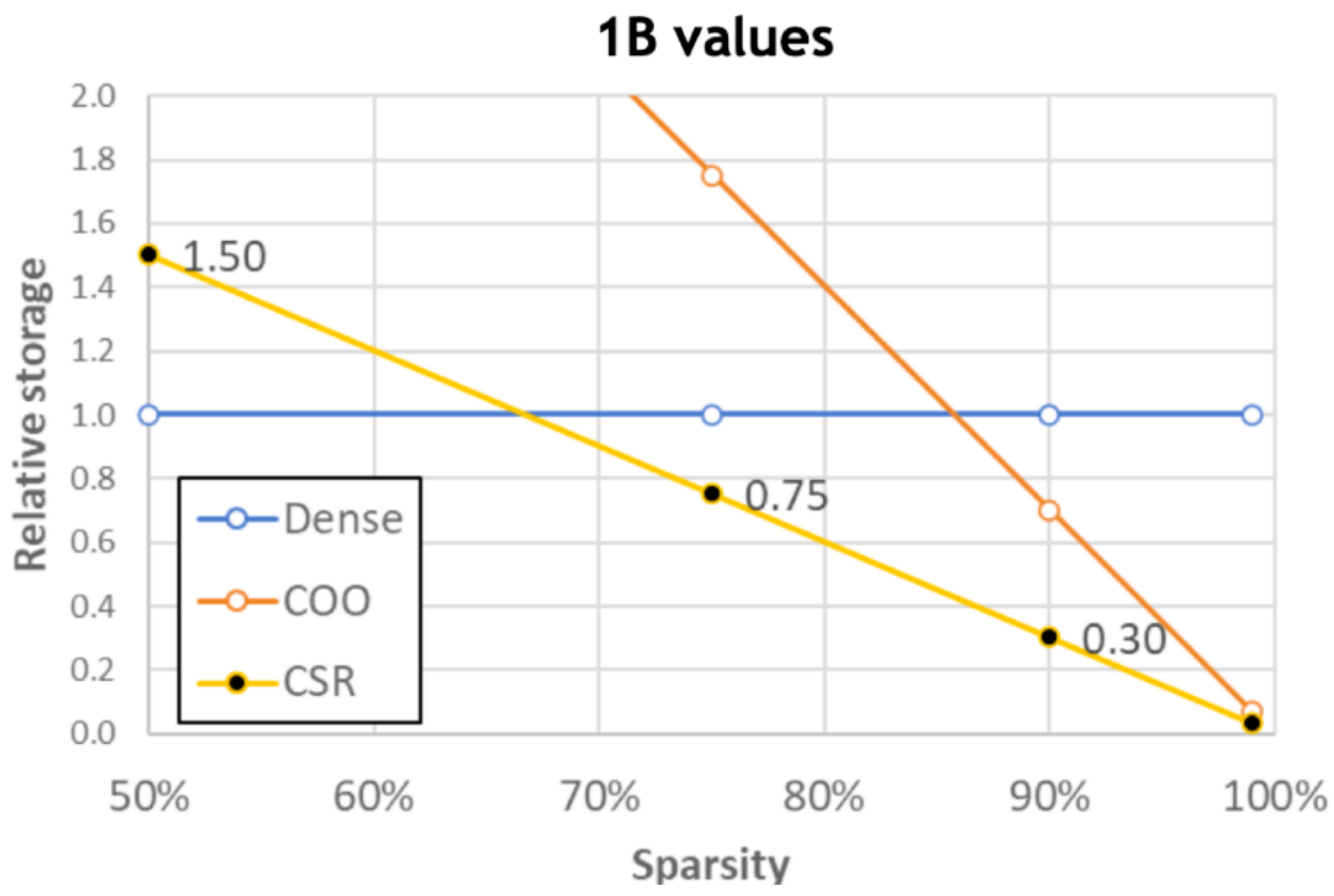
  $\Rightarrow$ 2K + 2N Bytes (2K + N if very sparse)

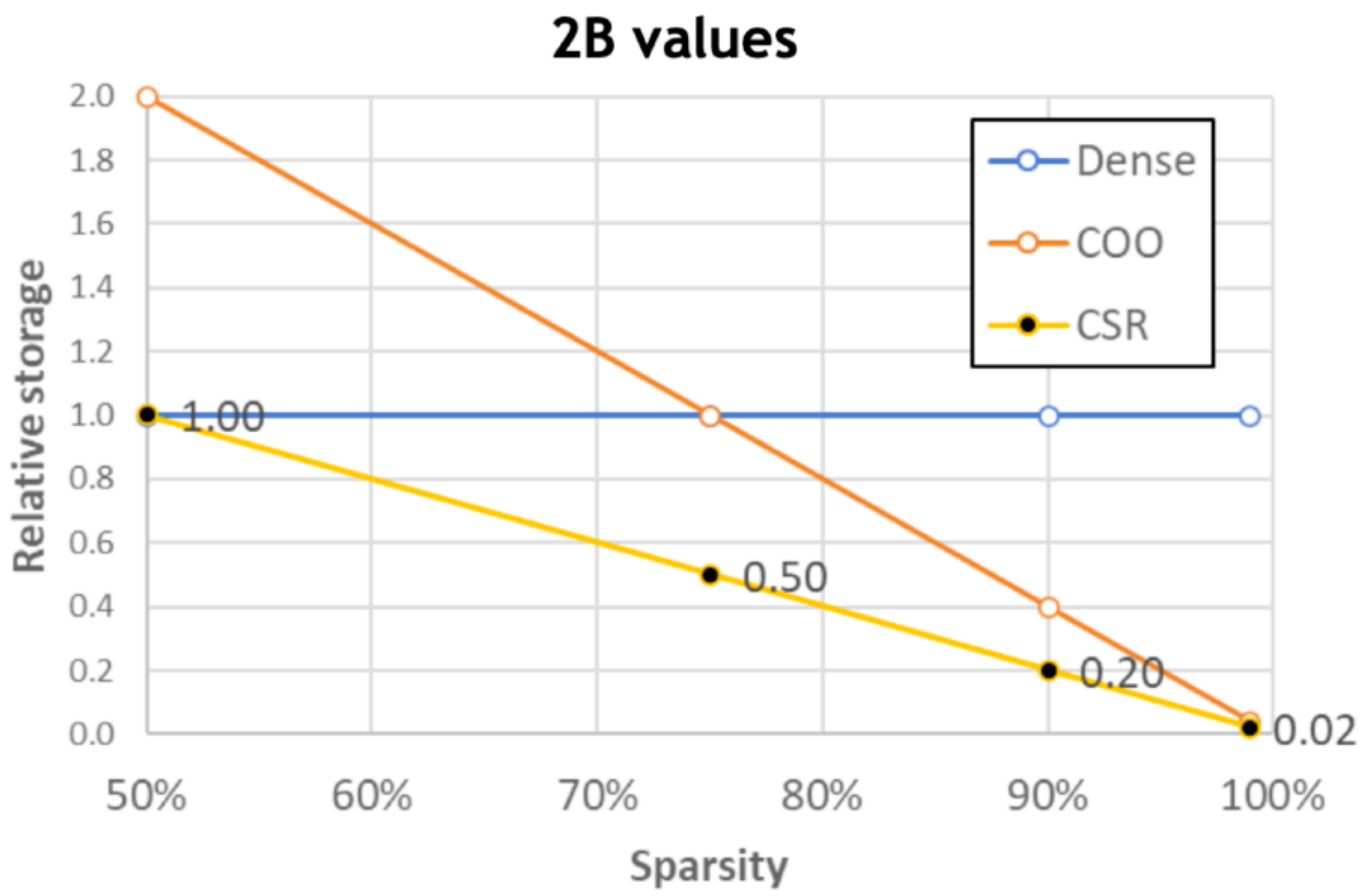
- If Sparsity $\geq$ 50%, we are good.



Storage

- In other words, the break-even sparsity of storage depends on…
  - Matrix dimensions
  - Precision
- Usually, requires at least 50%…

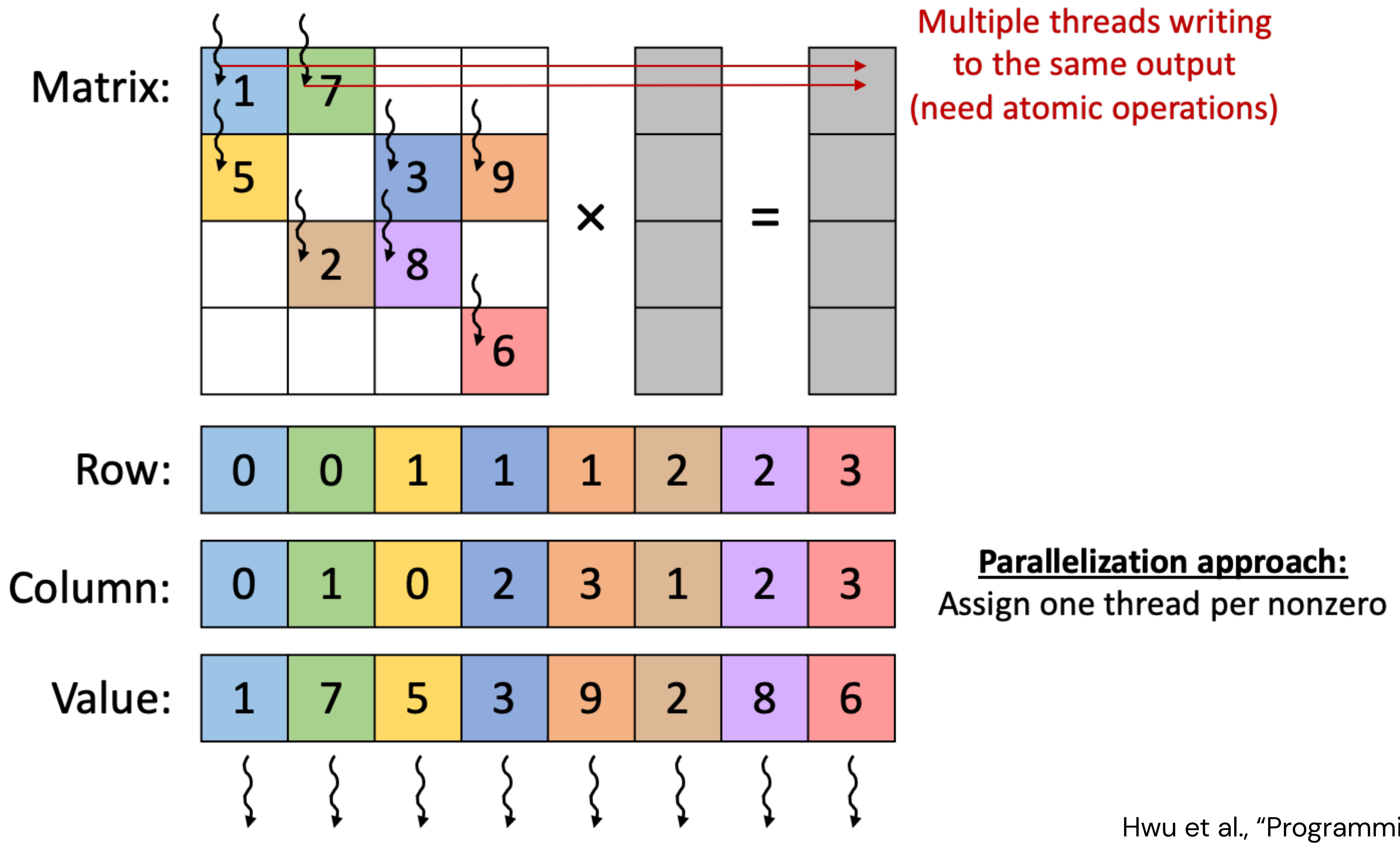




P. Micikevicius, Invited Talk @ Sparse Neural Network Workshop, 2021

SpMV

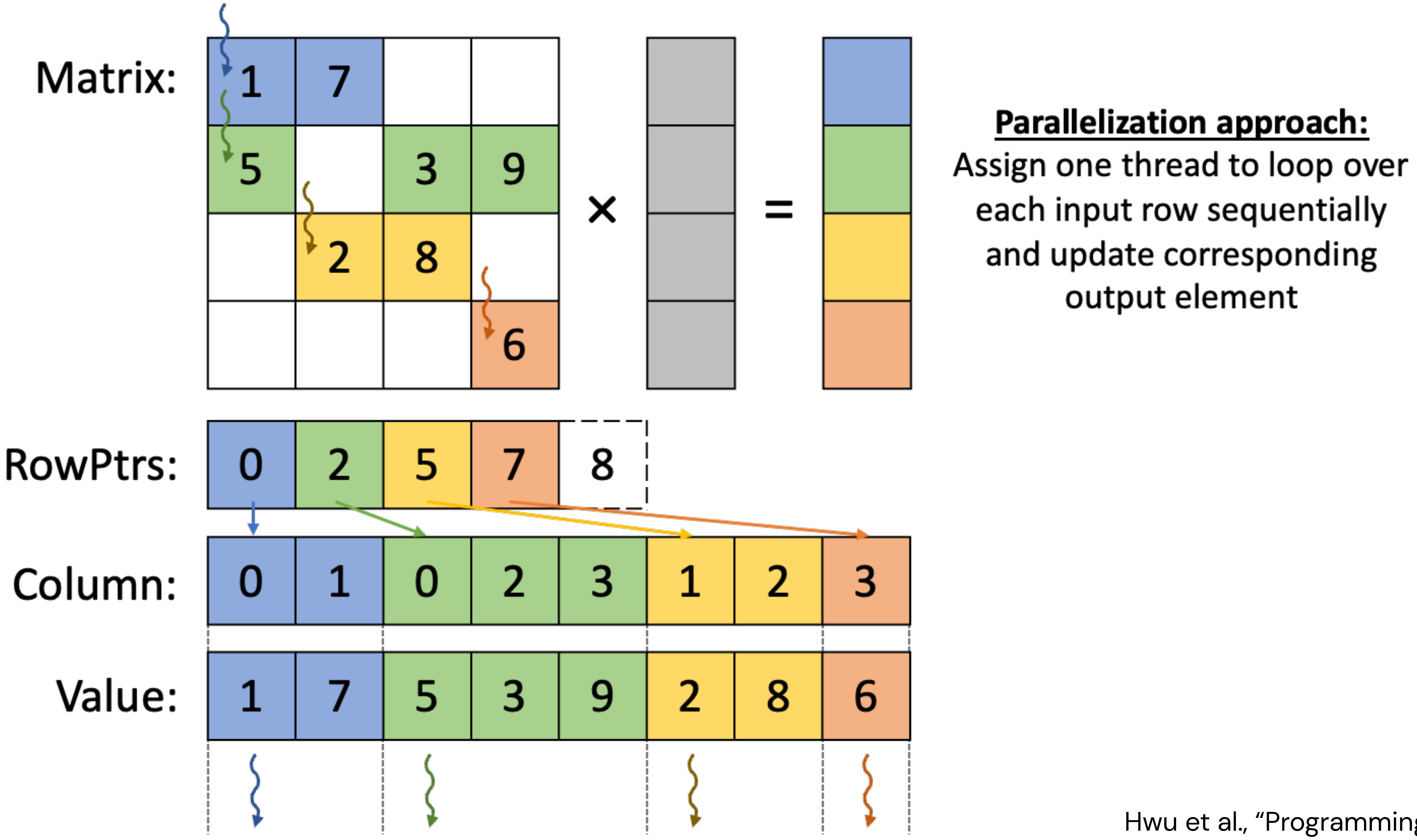
- If we use **COO**:
  - assign one thread per nonzero
  - coalesced memory access



Hwu et al., "Programming Massively Parallel Processors," Elsevier, 2022

SpMV

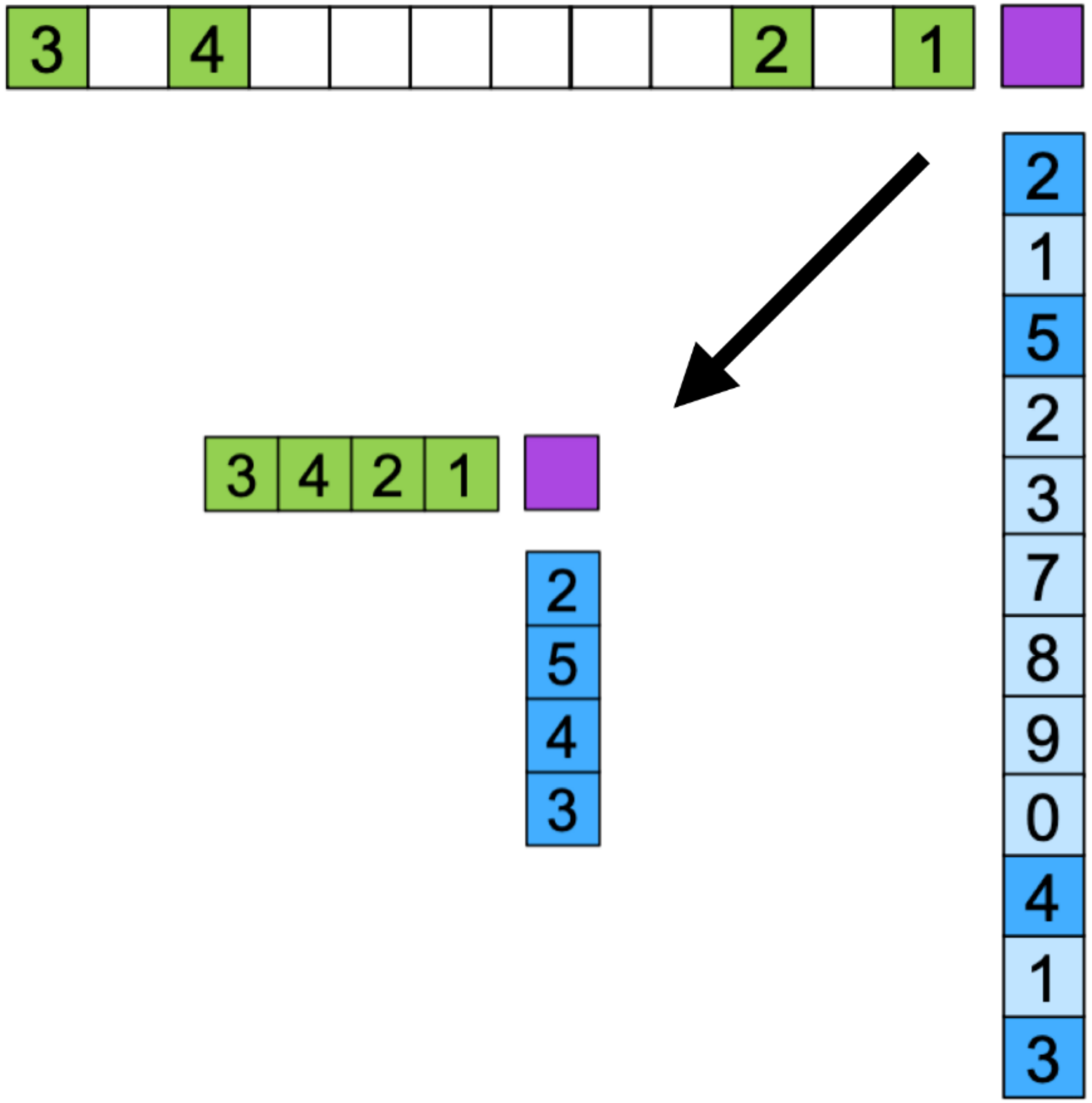
- If we use **CSR**:
  - Each thread writes on only one output
  - Dependent memory access



Hwu et al., “Programming Massively Parallel Processors,” Elsevier, 2022

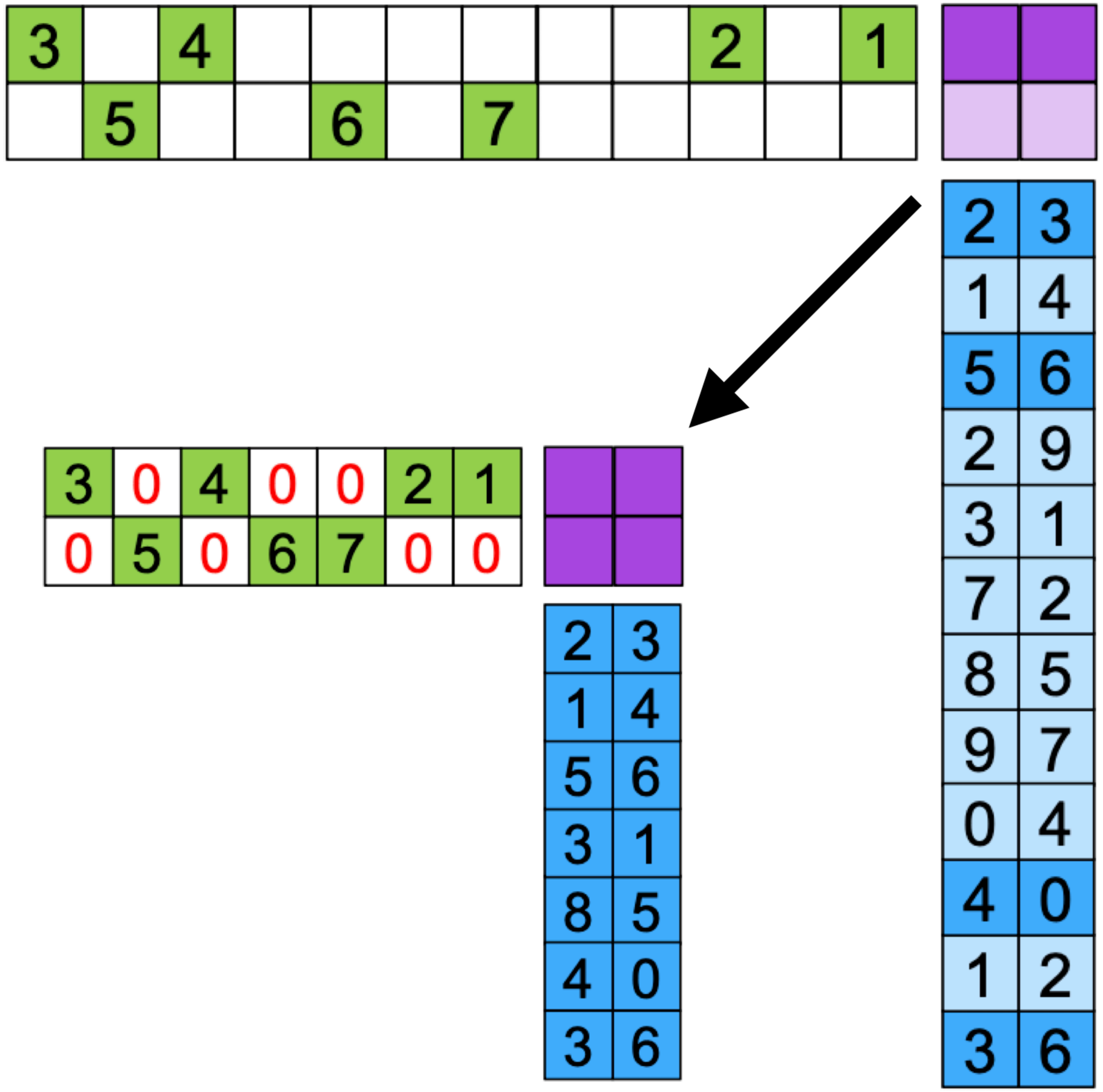
SpMV on GPU

- On GPU, we conventionally do:
  - Fetch nonzeros from the sparse matrix
  - Fetch corresponding dense elements
  - Use tensor cores for matmuls



SpMV on GPU

- **Problem.** More overhead if we group rows
  - Wasted computation
  - Time for fetching values from the dense matrix



SpMV on GPU

- **Solution.**
  - Custom kernels (⇒)
    (but we won't go deep here; see link)
  - Structures in zeros

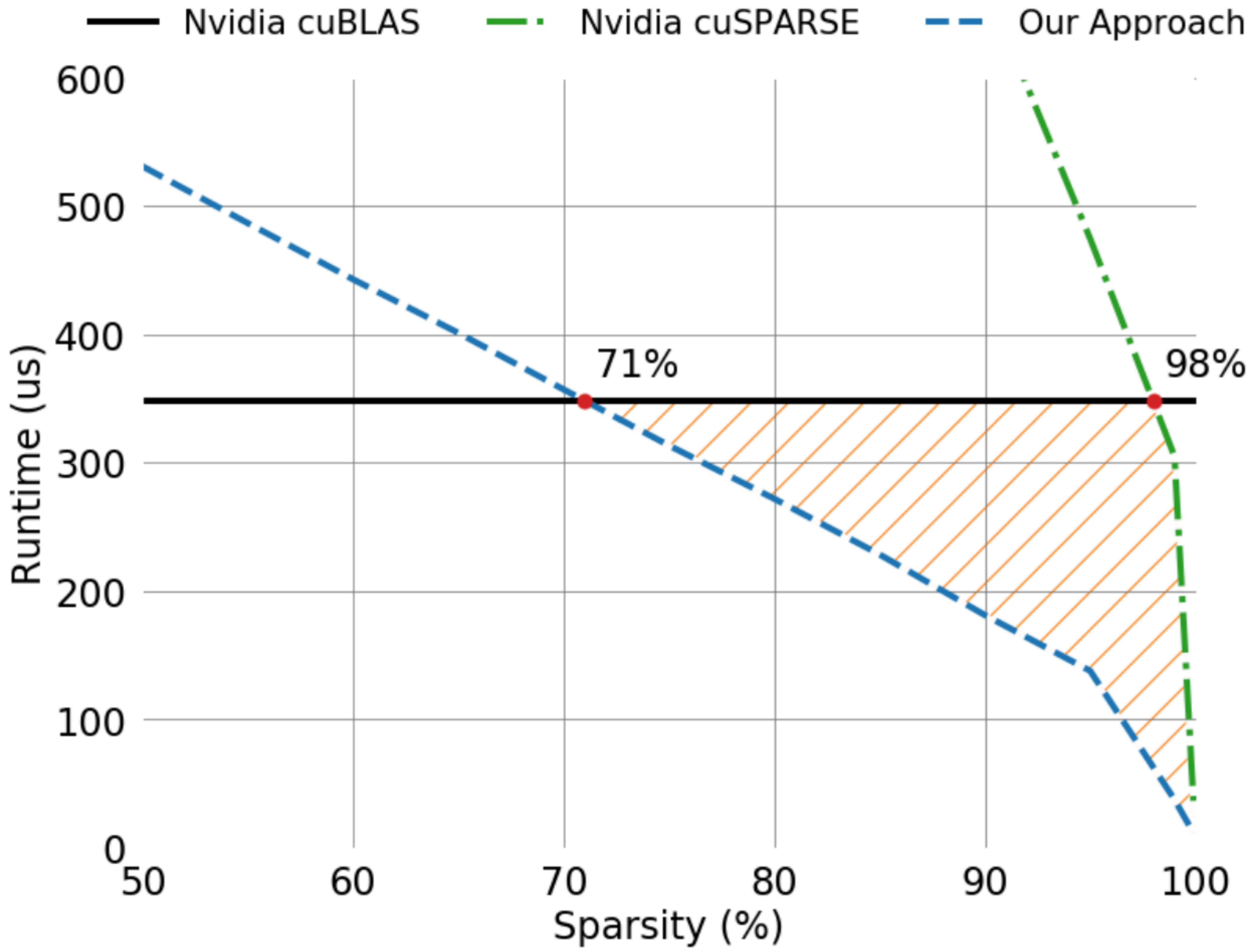


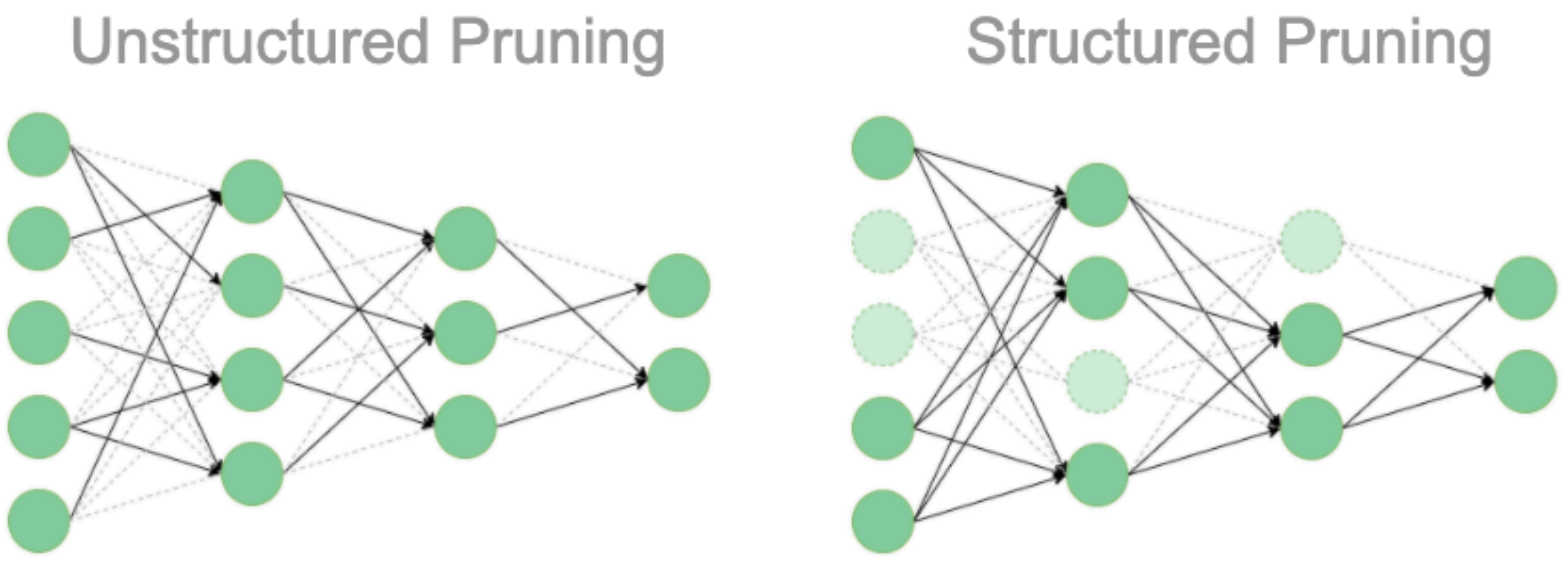
Fig. 1. **Sparse matrix–matrix multiplication runtime for a weight-sparse long short-term memory network problem**. Input size 8192, hidden size 2048, and batch size 128 in single-precision on an Nvidia V100 GPU with CUDA 10.1. Using our approach, sparse computation exceeds the performance of dense at as low as 71% sparsity. Existing vendor libraries require 14× fewer non-zeros to achieve the same performance. This work enables speedups for all problems in the highlighted region.

Gale et al., "Sparse GPU Kernels for Deep Learning," SuperComputing 2020

System considerations: Structured sparsity

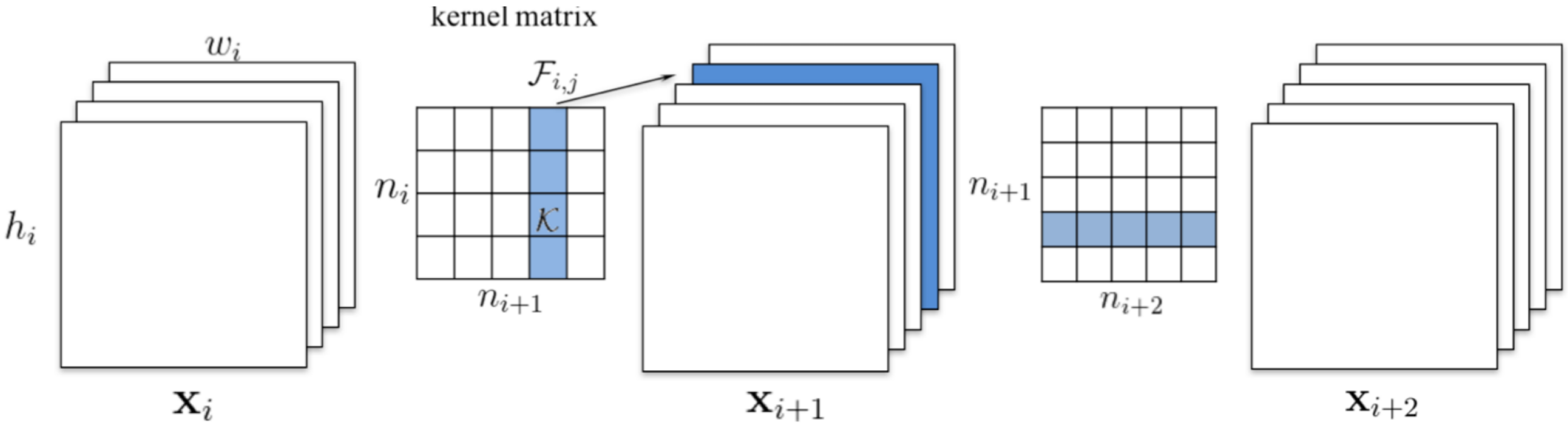
Structured Sparsity

- Pruning a group of weights at once
  - The pruned model becomes a small dense model
  - Less sparsity can be achieved
    - However, real advantages in runtime & memory



Structured Sparsity

- **ConvNets.** Prune a convolution filter ⇒ Remove an output channel
  ⇒ Prunes subsequent filters



Li et al., "Pruning filters for efficient ConvNets," ICLR 2017

Structured Sparsity

- **Transformers.** Many variants
  - Transformer block
  - Single layer
    - MHSA
    - FFN
  - Attention head
  - Neurons in the FFN layer

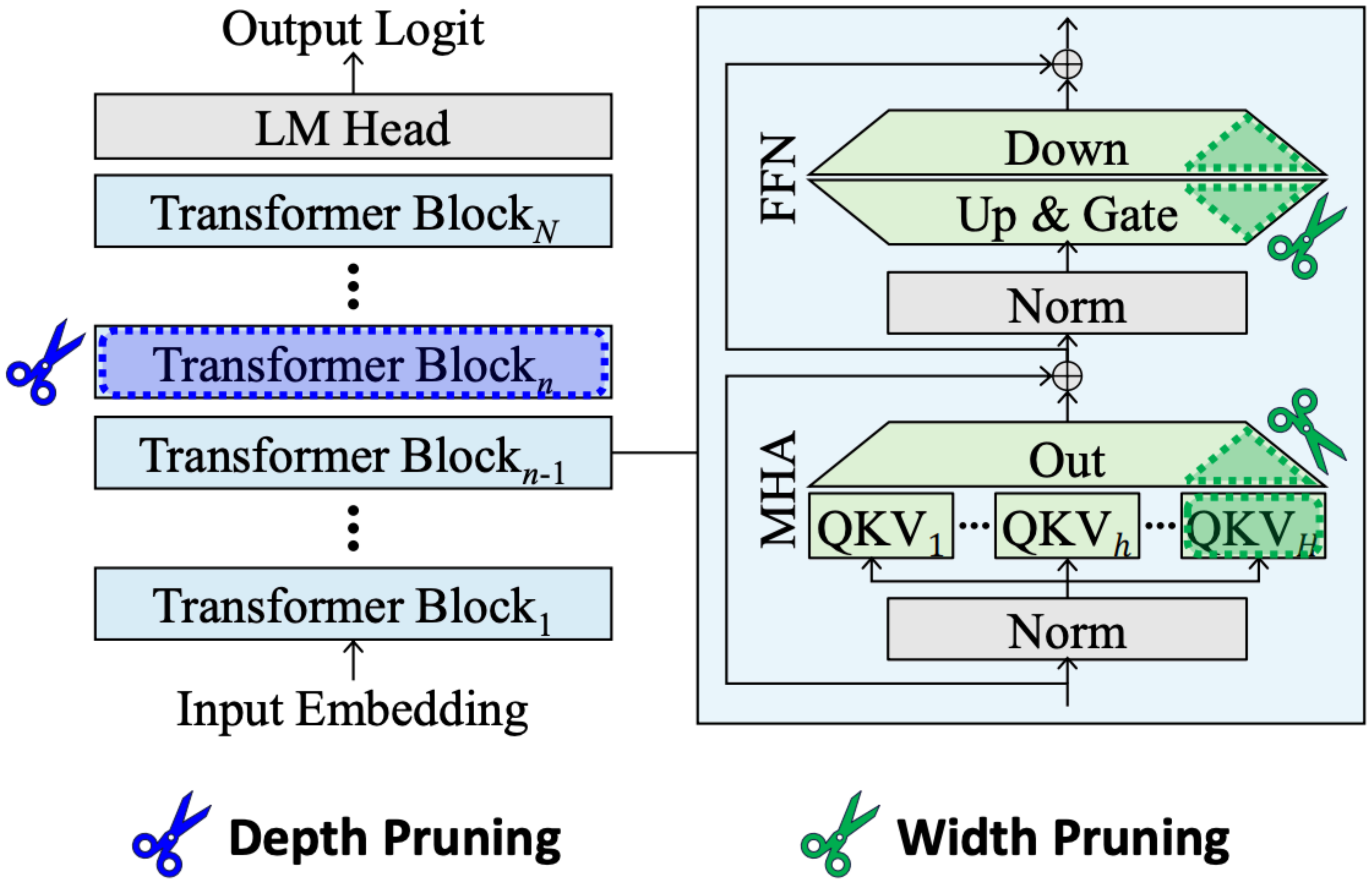


Figure 3: Comparison of pruning granularities. Width pruning reduces the size of weight matrices while maintaining the number of matrix-level operations. Depth pruning eliminates entire Transformer blocks, or individual MHA and FFN modules, leading to fewer memory accesses and matrix-level operations.

Kim et al., "Shortened LLaMA: A Simple Depth Pruning for Large Language Models," arXiv 2024.

Structured Sparsity

- **Neuron Merging.** If two neurons are similar, we can merge instead of removing
  - Less retraining needed

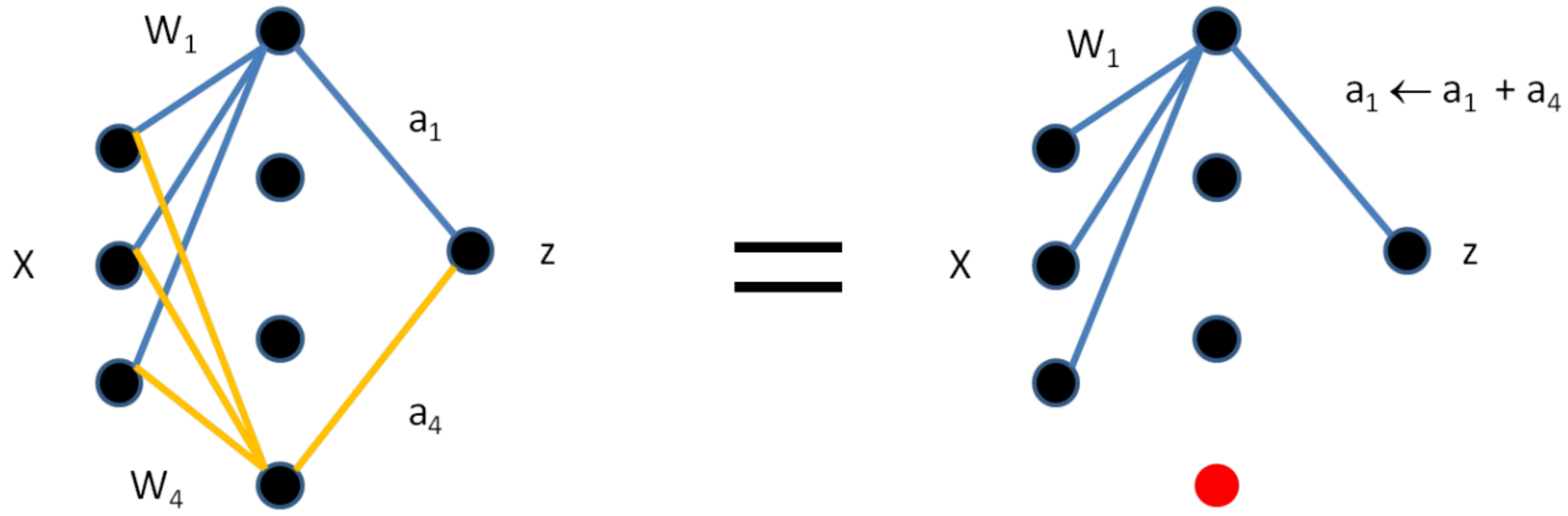
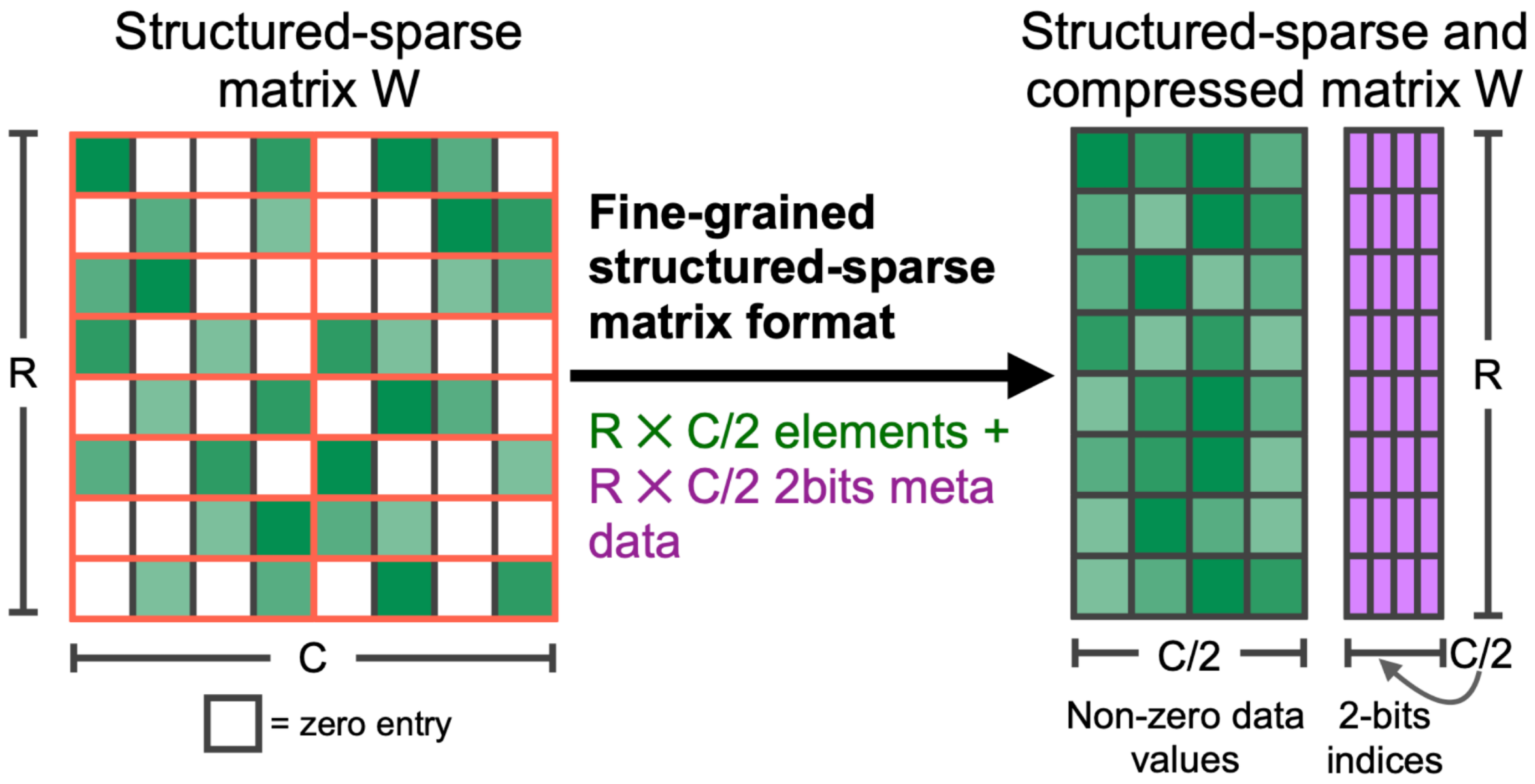


Figure 1: A toy example showing the effect of equal weight-sets ($W_1 = W_4$). The circles in the diagram are neurons and the lines represent weights. Weights of the same colour in the input layer constitute a weight-set.

Srinivas and Babu, "Data-free parameter pruning for deep neural networks," BMVC 2016

Structured + Fine-Grained Sparsity

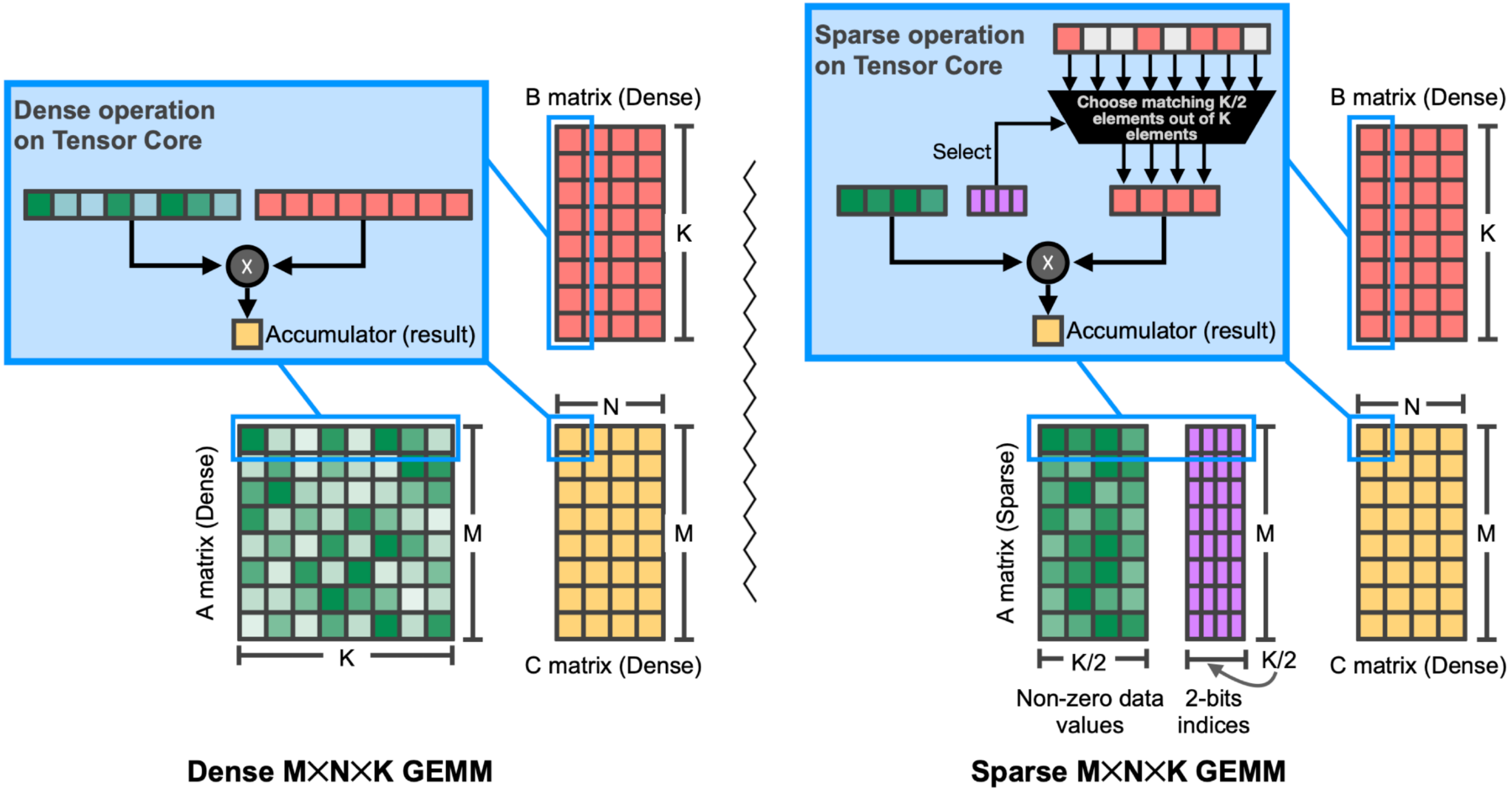
- **2:4 Sparsity (NVIDIA).** Constrain to have at least 2 zeros in length-4 blocks
  - 50% sparsity with usually no quality drop
  - Metadata can be very small; 2 bits per nonzero.



Mishra et al., "Accelerating sparse deep neural networks," arXiv 2021.

Structured + Fine-Grained Sparsity

- Requires customized HW and engines (Sparse Tensor Cores, TensorRT 8.0)



Mishra et al., "Accelerating sparse deep neural networks," arXiv 2021.

Other examples

- **NAVER + Samsung**
  - Specialized HW with fixed-to-fixed encoding for sparsity (link)
- **Neural Magic**
  - CPU runtime for on-device acceleration (DeepSparse)

Mishra et al., “Accelerating sparse deep neural networks,” arXiv 2021.

Remarks

- We have skipped the whole ideas of **activation sparsity**:

$$\mathbf{WX} \rightarrow \mathbf{WX}_{\text{sparse}}$$

  - See following references:
    - https://proceedings.mlr.press/v119/kurtz20a.html
    - https://www.jmlr.org/papers/v22/21-0366.html

That’s it for today 🙌