UNIVERSIDADE DA CORUÑA ETH zürich

VENOM: A Vectorized N:M Format for Unleashing the Power of Sparse Tensor Cores

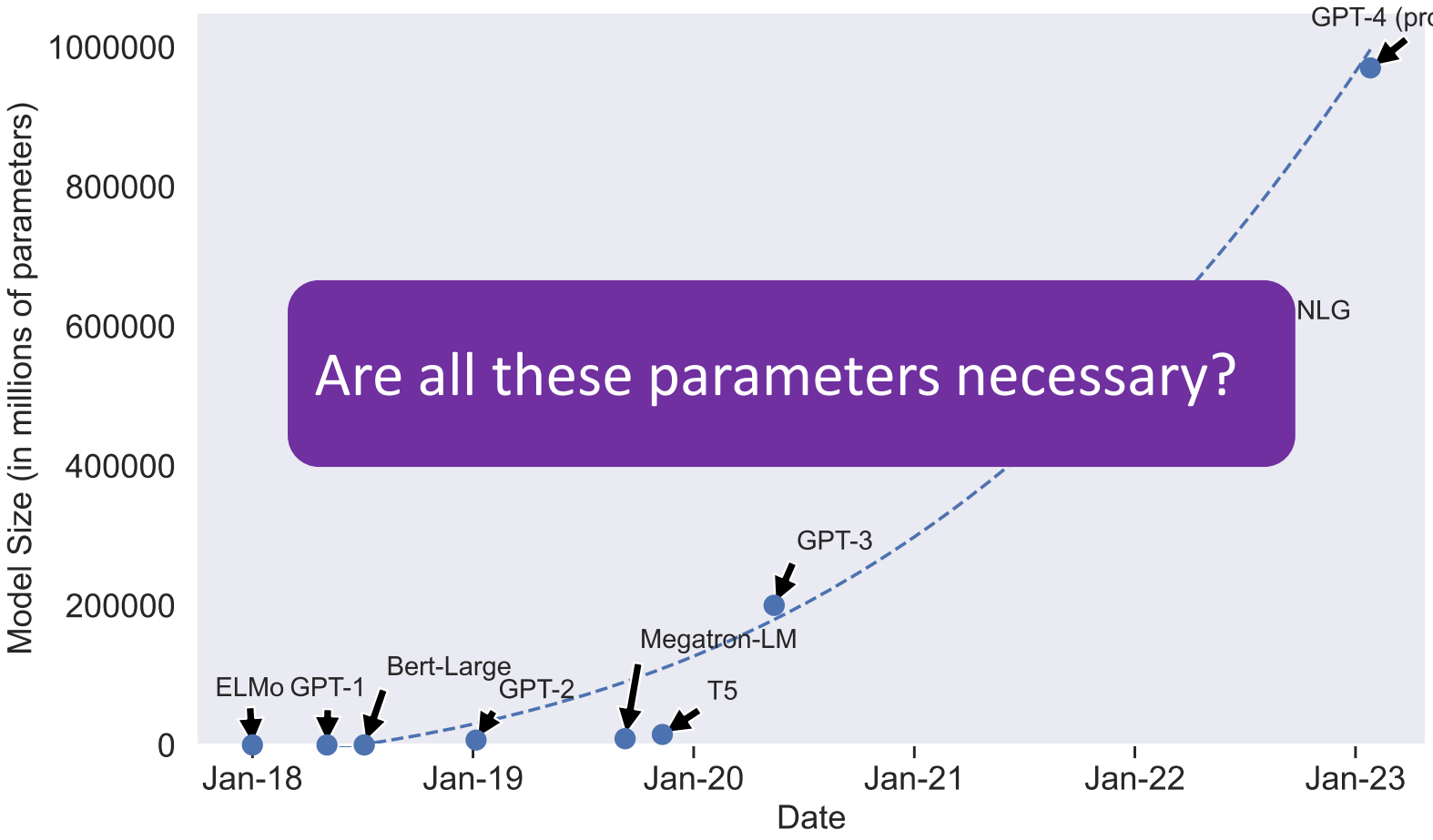
Roberto L. Castro*, Andrei Ivanov†, Diego Andrade*, Tal Ben-Nun†, Basilio B. Fraguela*, Torsten Hoefler†

Supercomputing'23, Denver, CO, Nov. 2023

† SPCL

* COMPUTER ARCHITECTURE GROUP UNIVERSITY OF A CORUÑA GAC.UDC.ES

Model's size evolution



Are all these parameters necessary?

Modern models have a training cost of millions of dollars (e.g., GPT-4 > $100M)

Inference costs, far exceed training costs when deploying a model at any reasonable scale

Model sparsification

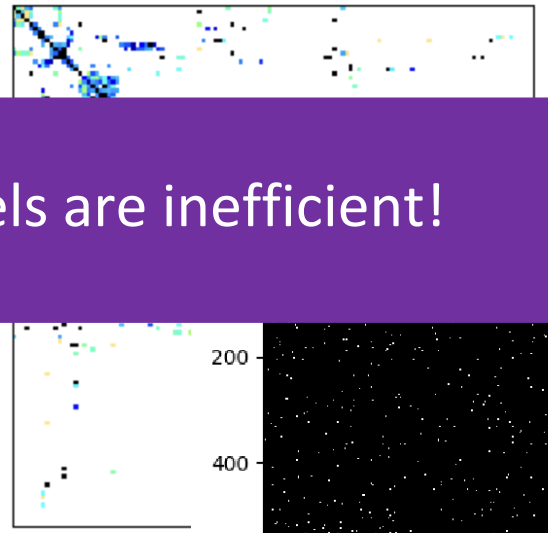
Intuition: Not all features are always relevant!

- ✅ Less overfitting
- ✅ Interpretability
- ✅ Parsimony

Sparse Matrix-Matrix Multiplication

AB=C

- ❖ A sparse matrix
- ❖ B and C dense matrices



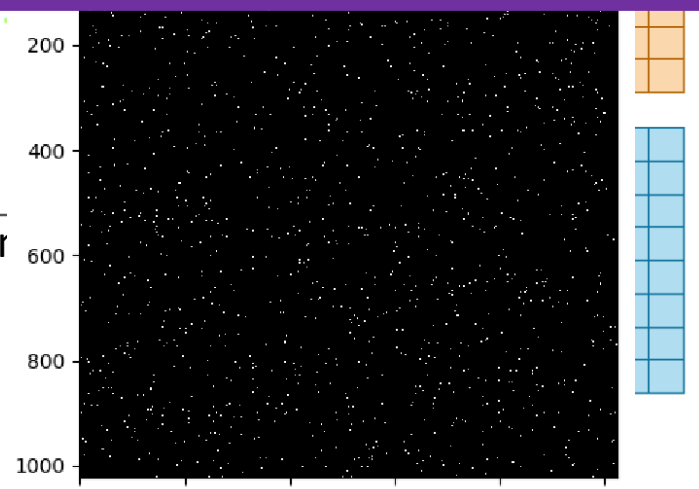
N

Existing GPU kernels are inefficient!

Sparsity in scientific problems

99%, quite structural (e.g., banded matrices)

source: sparse.tar



Sparsity in DL:

50%~90%, quite irregular

Hoefler et. al “Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks”, arXiv 2102.00554, Jan 2021

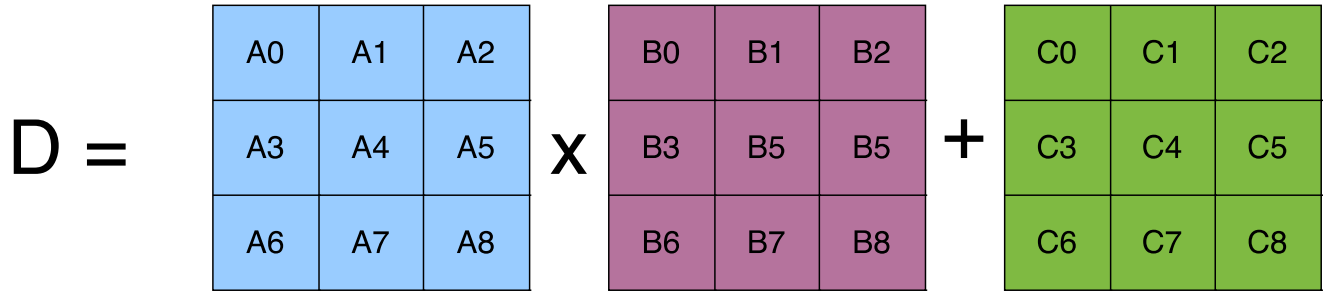
Tensor Core Units (TCU)
and HW support for structured sparsity

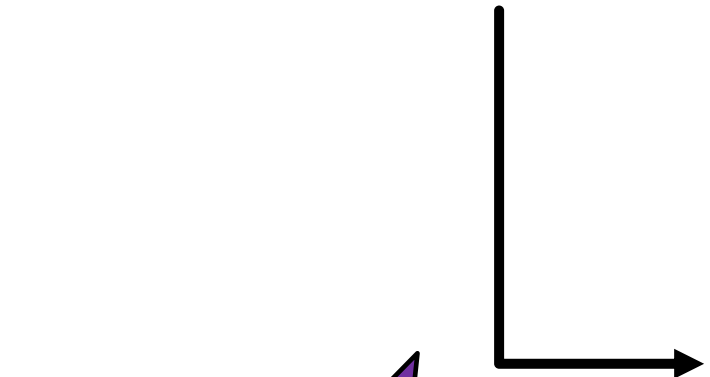
Tensor Core Units

Number Representation
- FP32,FP16,Int8, TF32,BF16
- ~16x

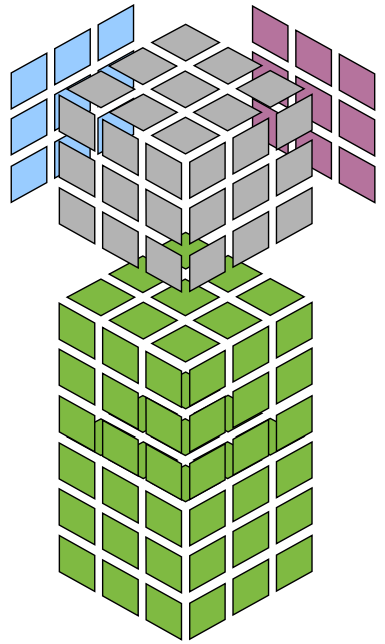
Process
- 28nm, 16nm, 7nm,5nm
- ~2.5x

Complex instructions
- HMMA, IMMA
- ~12.5x

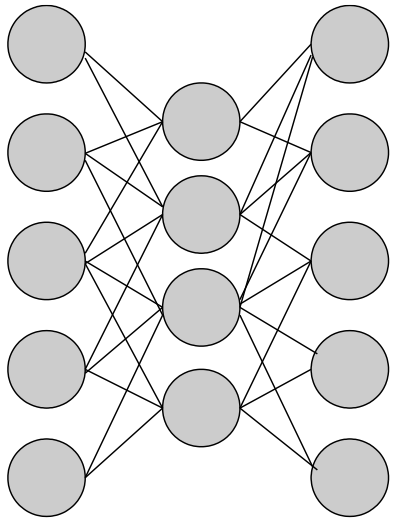


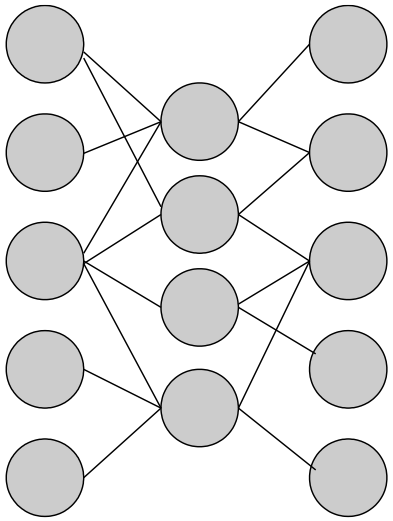


Structured Sparsity + ~2x



Sparse Tensor Cores – structured sparsity





Sparse Tensor Core (SPTC)

0	4	3	1	1	2	3	0
0	4	3	2	2	1	0	3
1	4	2	3	4	4	2	2
1	1	3	3	3	0	1	4
4	4	0	0	1	3	2	1
0	1	2	2	2	3	4	0
0	1	2	3	3	2	4	2
0	0	2	2	1	1	1	1

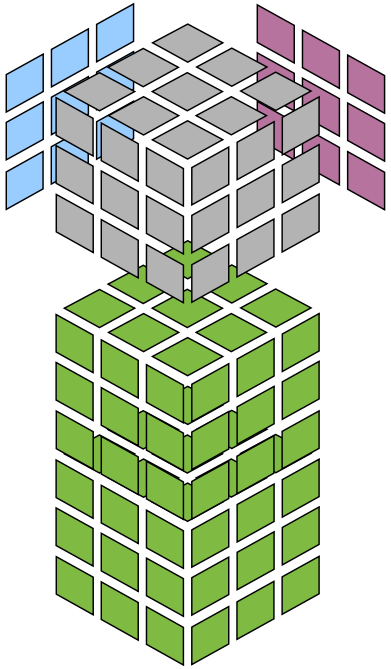
Dense

Prune

	4	3			2	3	
	4	3		2			3
	4		3	4	4		
		3	3	3			4
4	4				3	2	
		2	2		3	4	
		2	3	3		4	
		2	2	1	1		

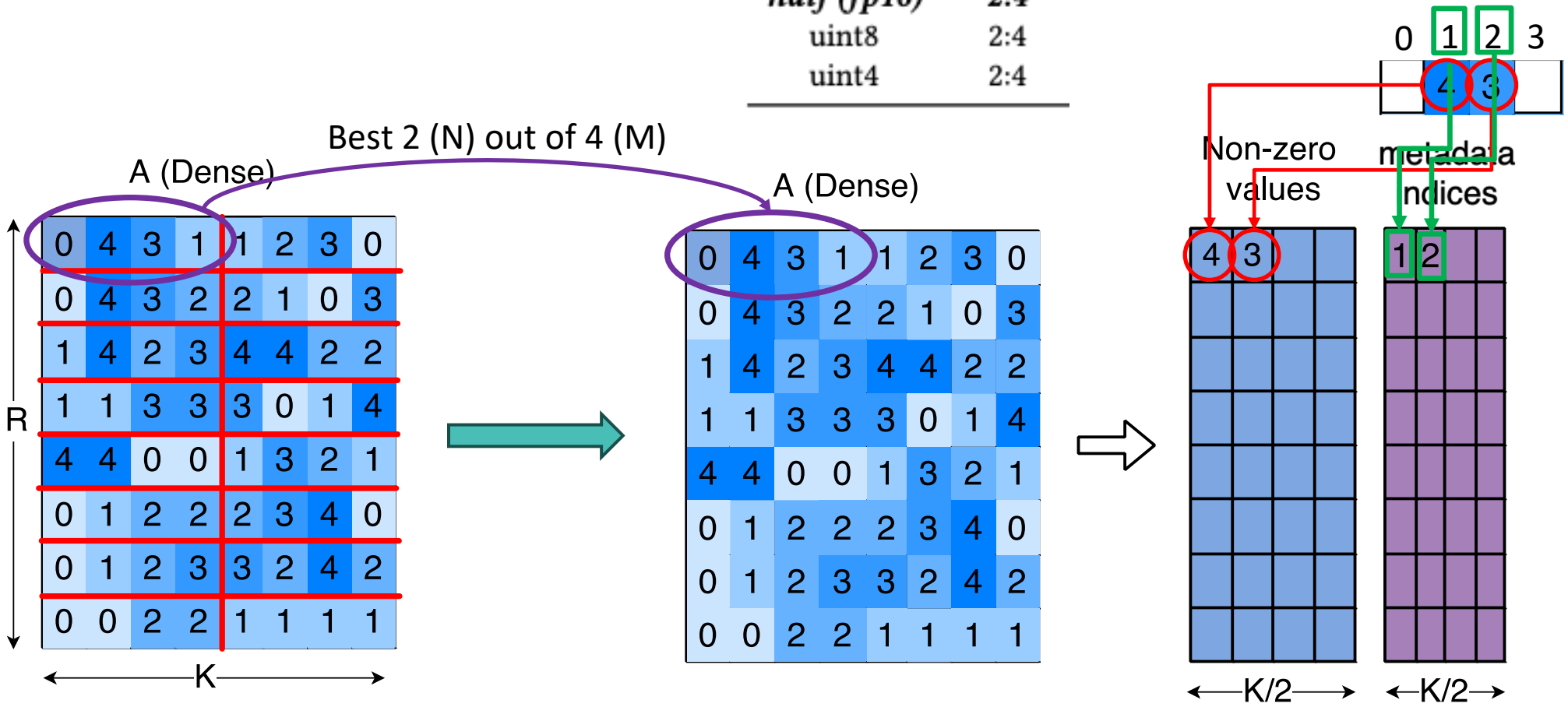
Sparse

Execute



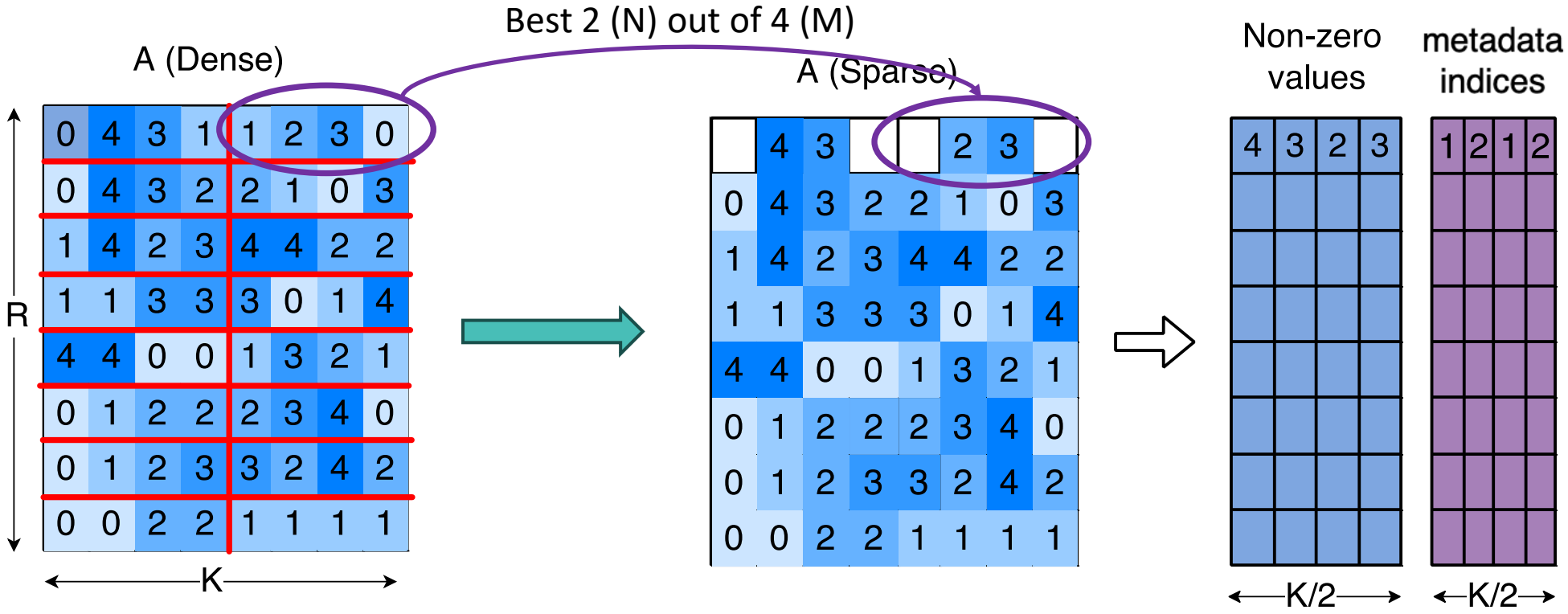
NVIDIA N:M format

Precision	Format
fp32	1:2
half (fp16)	***2:4***
uint8	2:4
uint4	2:4



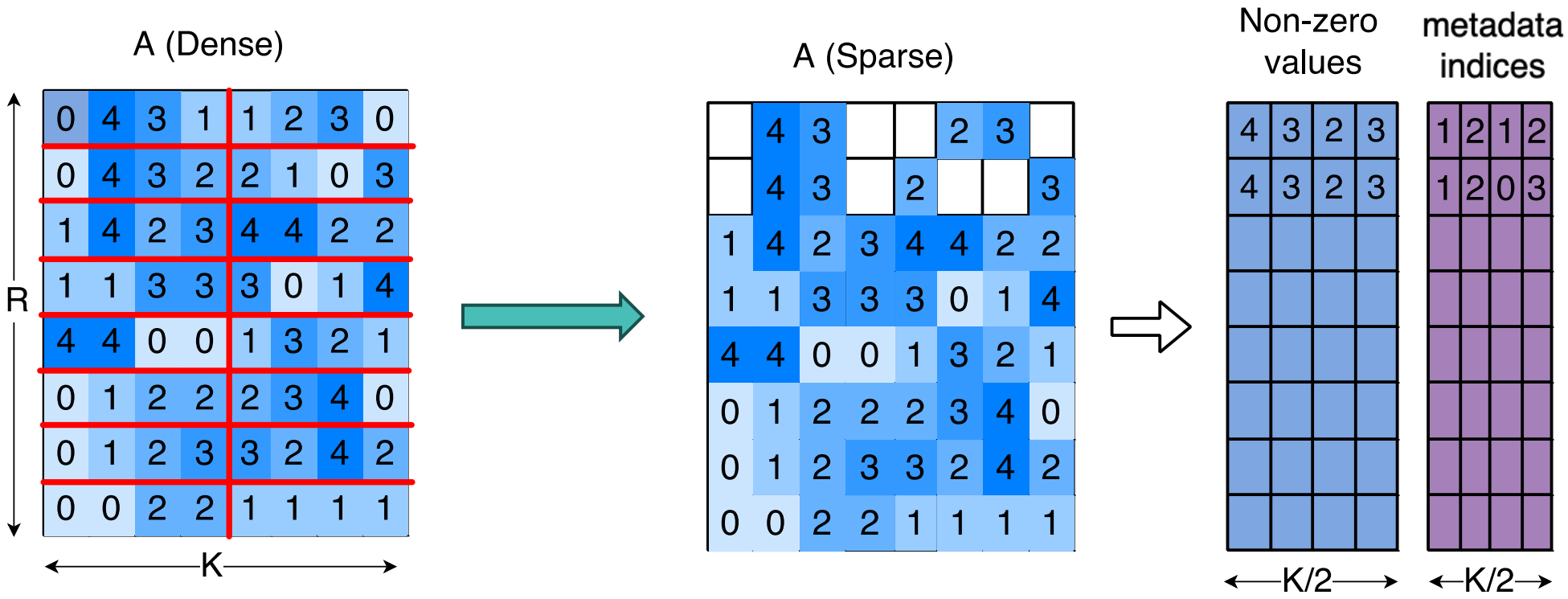
NVIDIA N:M format

Precision	Format
fp32	1:2
half (fp16)	***2:4***
uint8	2:4
uint4	2:4



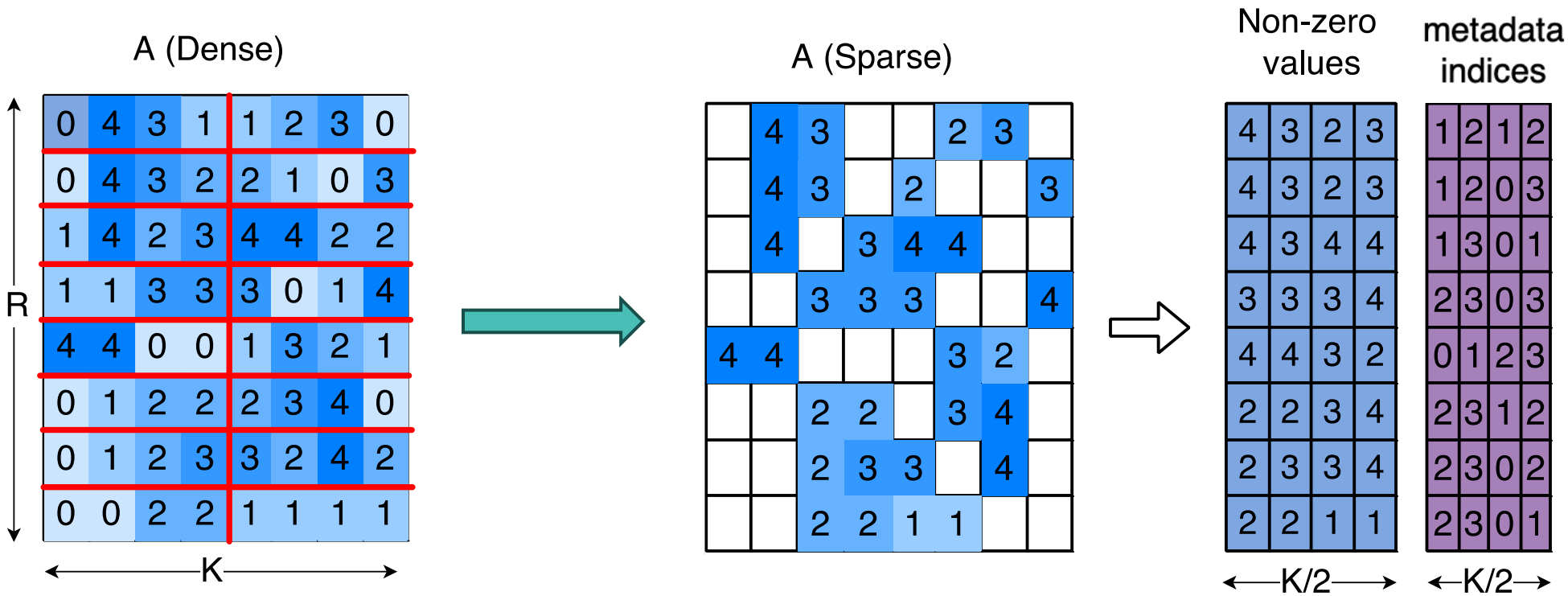
NVIDIA N:M format

Precision	Format
fp32	1:2
half (fp16)	***2:4***
uint8	2:4
uint4	2:4

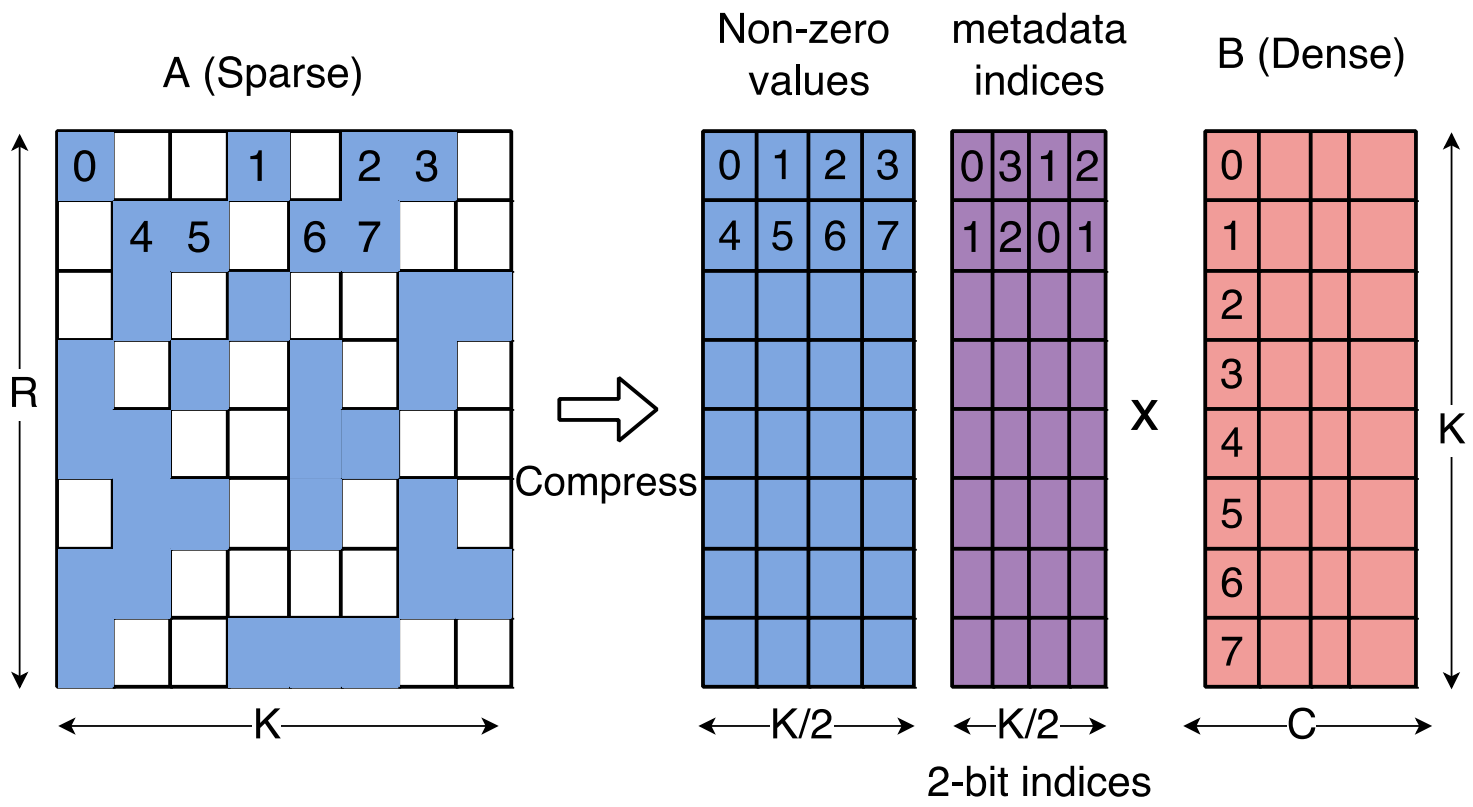


NVIDIA N:M format

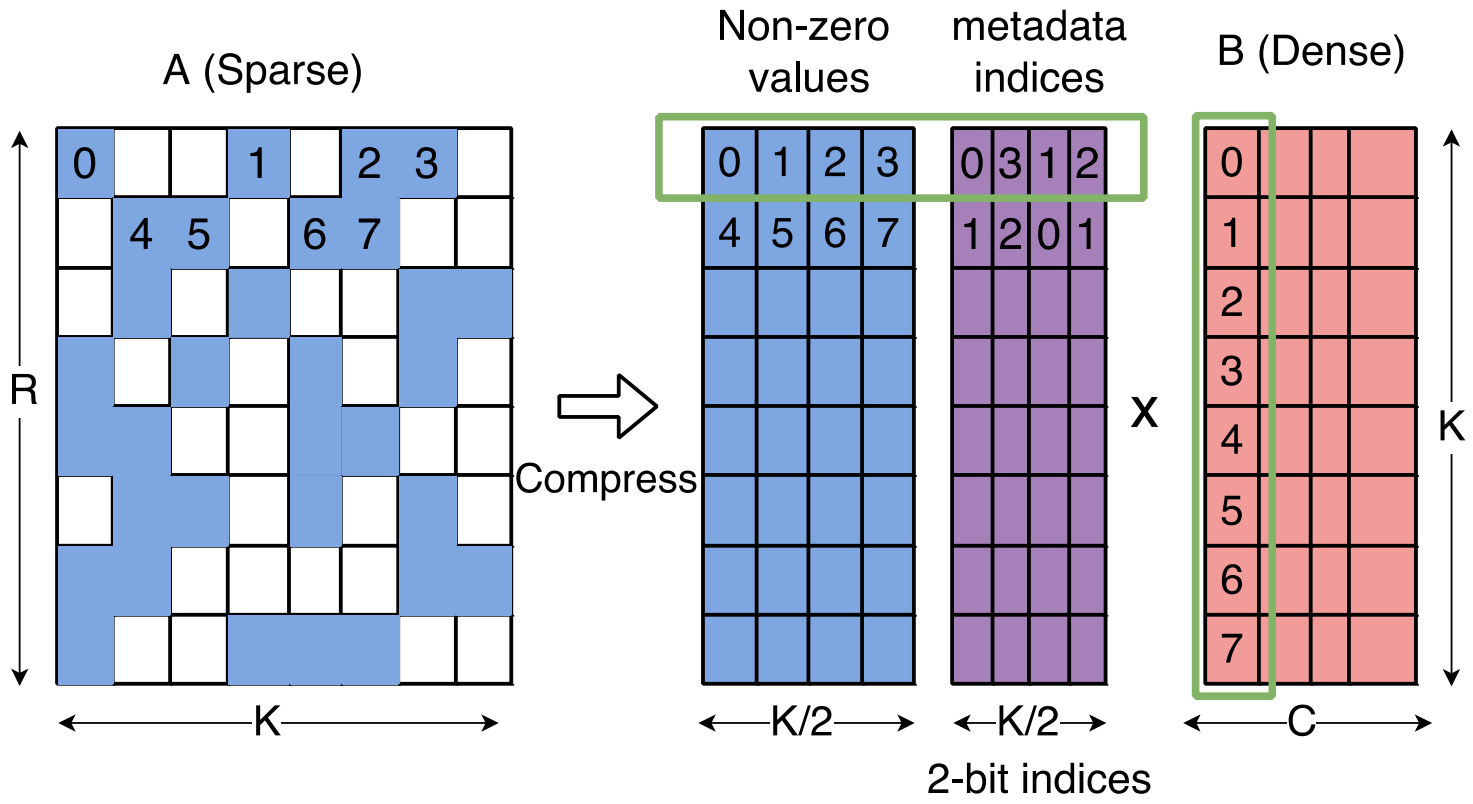
Precision	Format
fp32	1:2
half (fp16)	***2:4***
uint8	2:4
uint4	2:4



NVIDIA N:M format



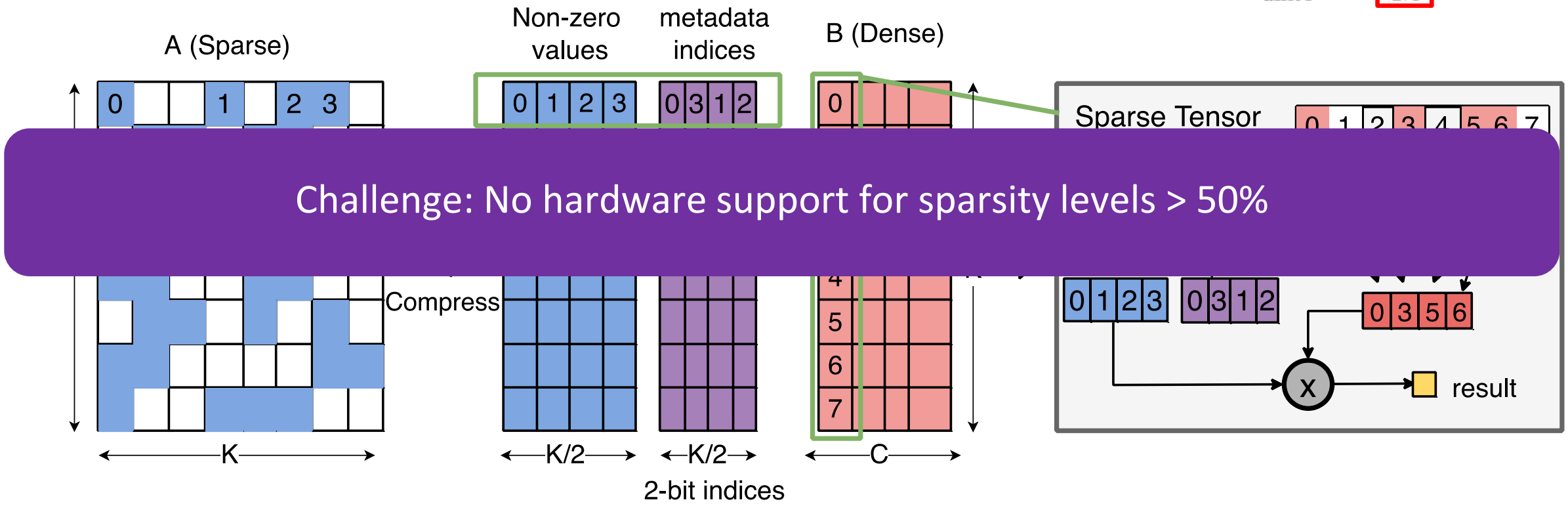
NVIDIA N:M format



NVIDIA N:M format

50% sparse matrices !!!

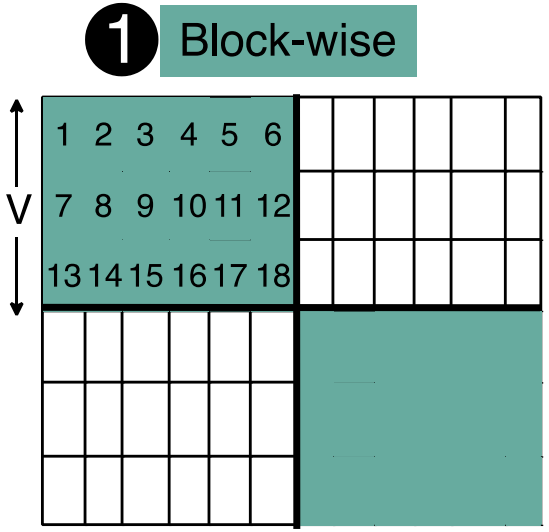
Precision	Format
fp32	1:2
half (fp16)	***2:4***
uint8	2:4
uint4	2:4



Challenge: No hardware support for sparsity levels > 50%

Existing sparse matrix formats

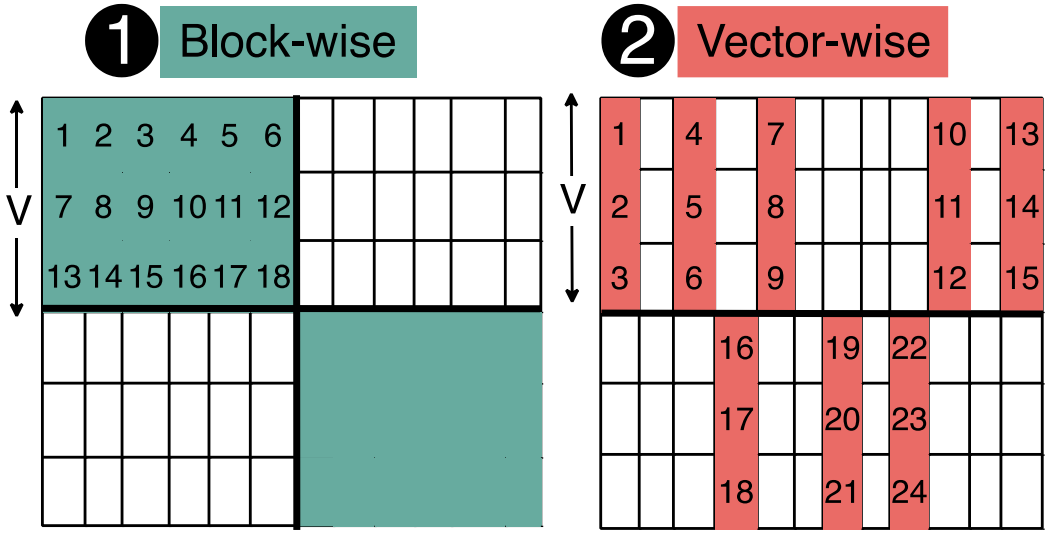
Existing Sparse Matrix Formats



Properties of existing sparse formats:

1. Block-wise
   - ☑ Very Regular
   - ✗ Too restrictive weight selection
   - ✗ No hardware support

Existing Sparse Matrix Formats

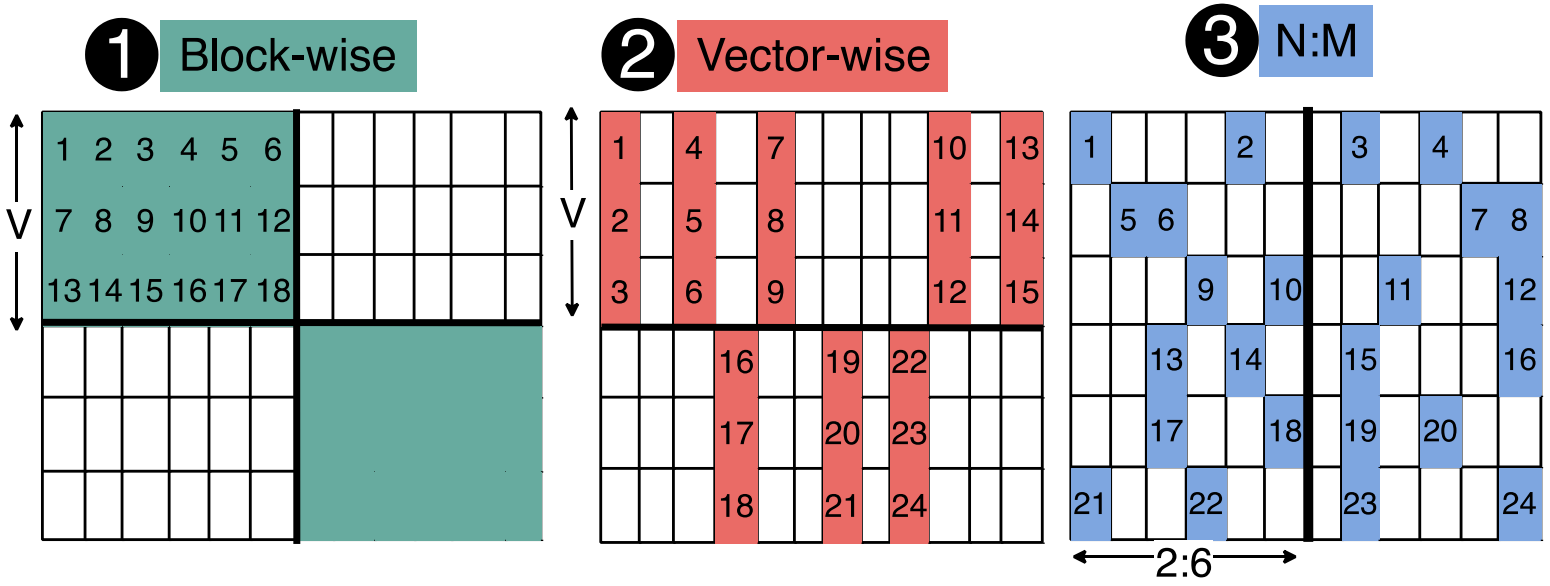


Properties of existing sparse formats:

1. Block-wise
   - ☑ Very Regular
   - ✗ Too restrictive weight selection
   - ✗ No hardware support
2. Vector-wise
   - ⚖ Accuracy-performance trade-offs
   - ✗ Performance still far from ideal
   - ✗ No hardware support

Roberto L. Castro et. al “Probing the Efficacy of Hardware-Aware Weight Pruning to Optimize the SpMM routine on Ampere GPUs”, PACT’22

Existing Sparse Matrix Formats



Properties of existing sparse formats:

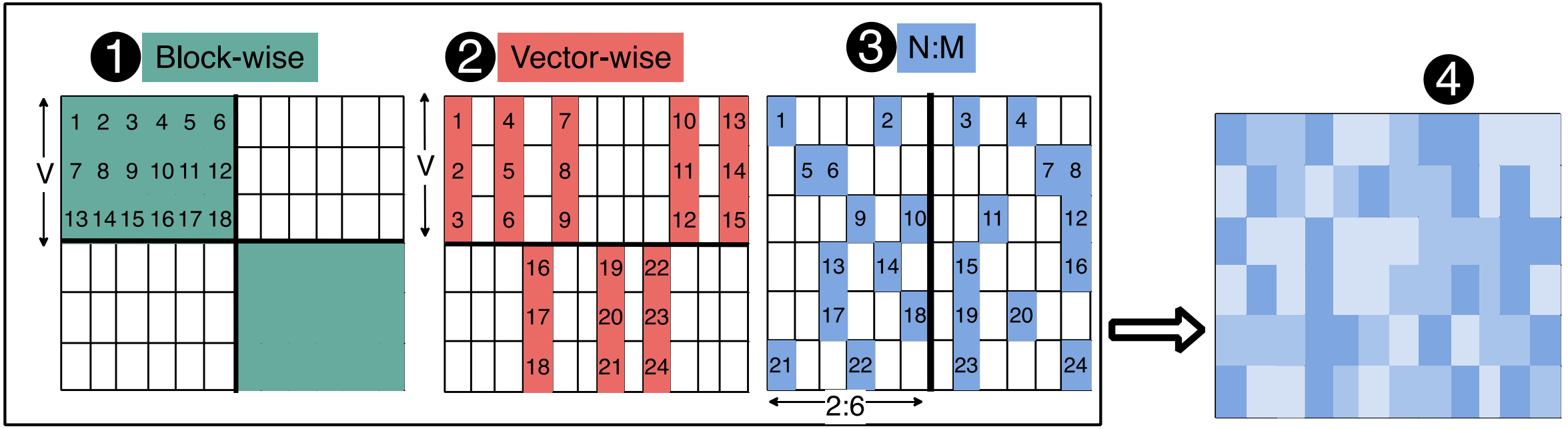
1. Block-wise
   - ✅ Very Regular
   - ❌ Too restrictive weight selection
   - ❌ No hardware support
2. Vector-wise
   - ⚖️ Accuracy-performance trade-offs
   - ❌ Performance still far from ideal
   - ❌ No hardware support
3. N:M
   - ✅ Very flexible
   - ✅ Hardware support but…
     - ❌ Restricted to 50% sparsity

Roberto L. Castro et. al “Probing the Efficacy of Hardware-Aware Weight Pruning to Optimize the SpMM routine on Ampere GPUs”, PACT’22

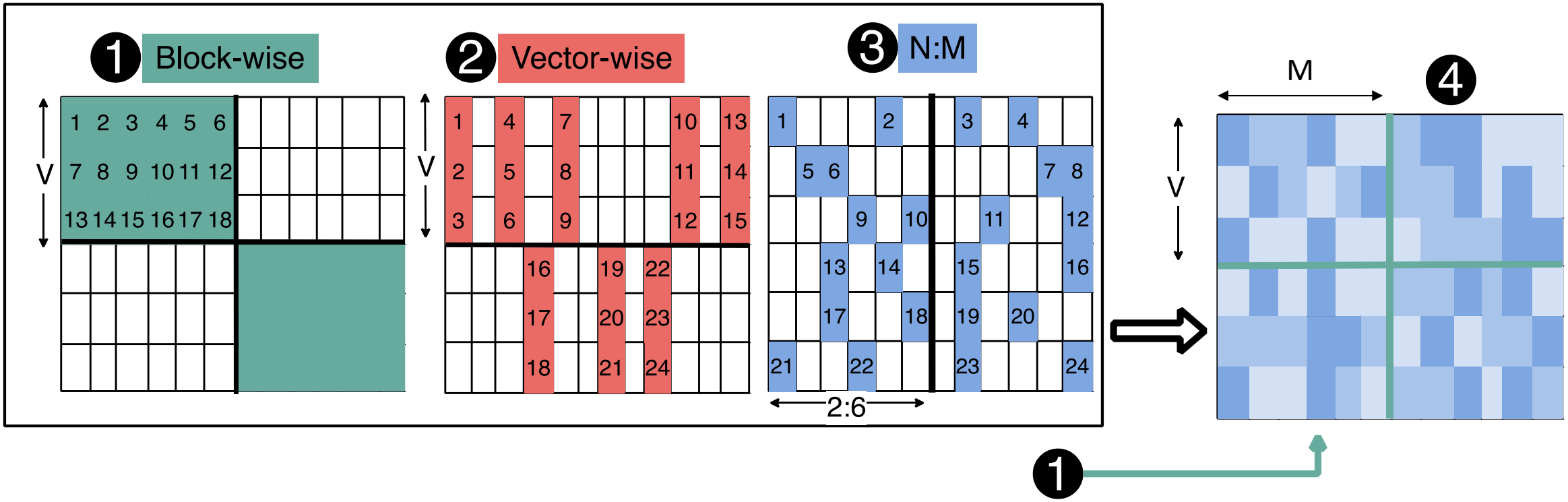
VENOM 

VENOM icon was generated with DALL-E 3

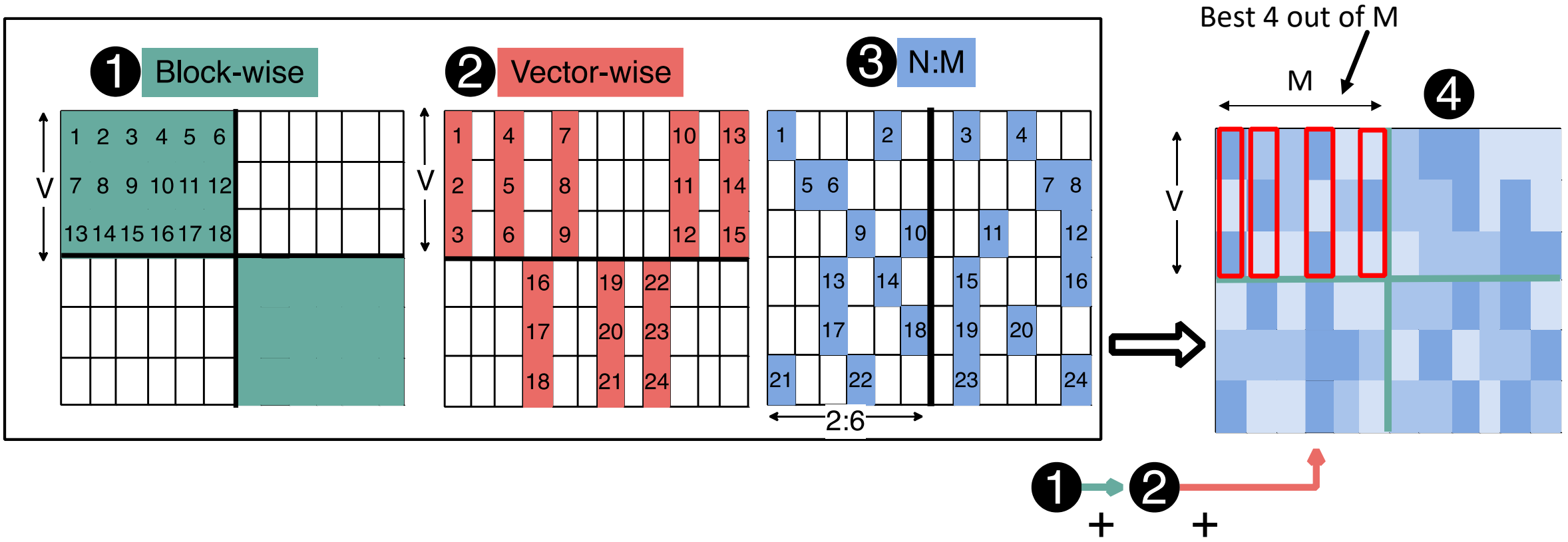
VENOM: The V:N:M Format



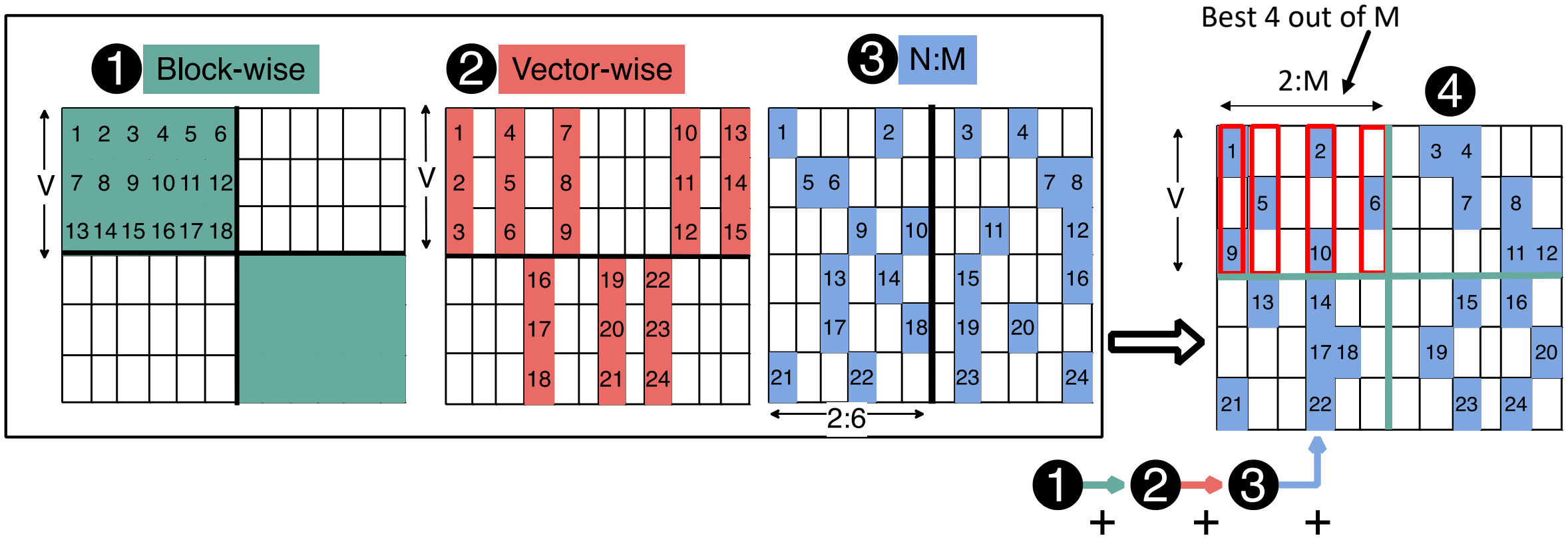
VENOM: The V:N:M Format



VENOM: The V:N:M Format

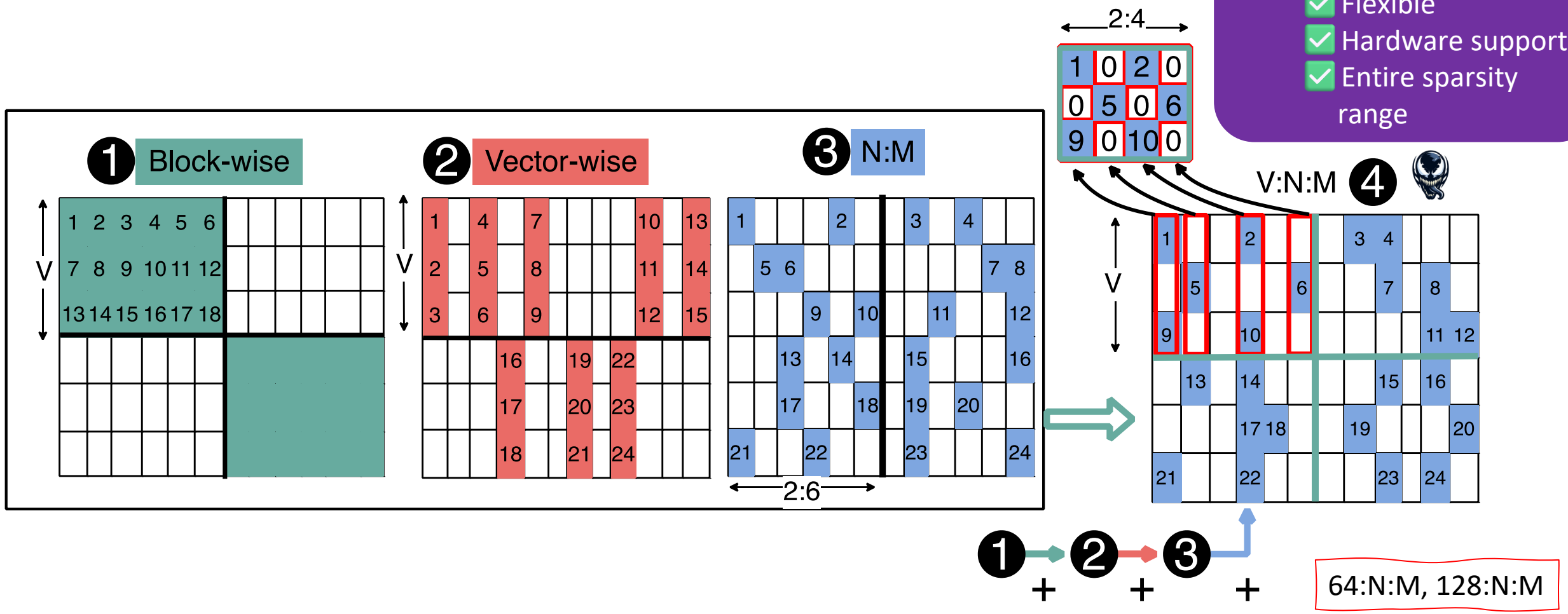


VENOM: The V:N:M Format

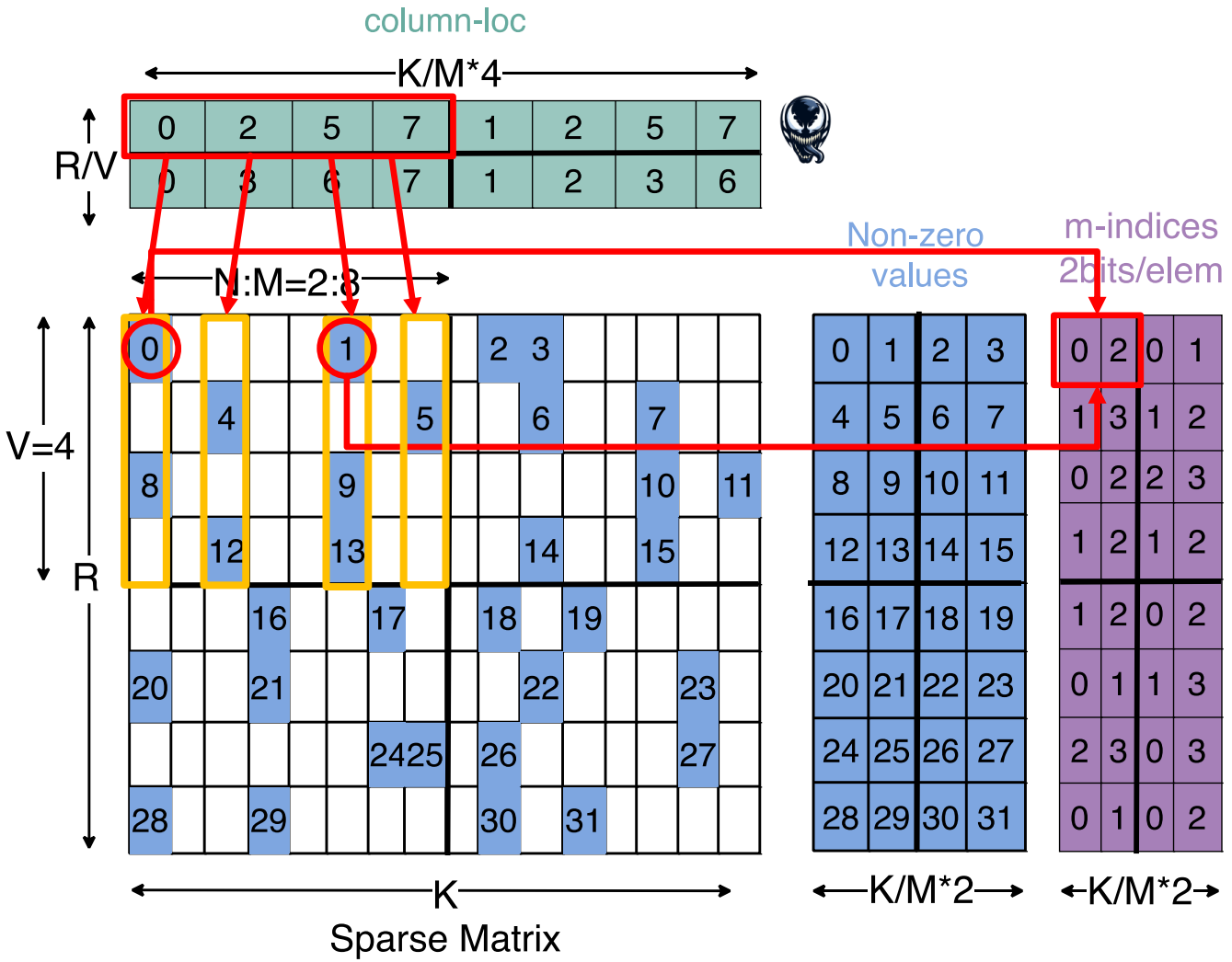


VENOM: The V:N:M Format

V:N:M
- Very Regular
- Flexible
- Hardware support
- Entire sparsity range



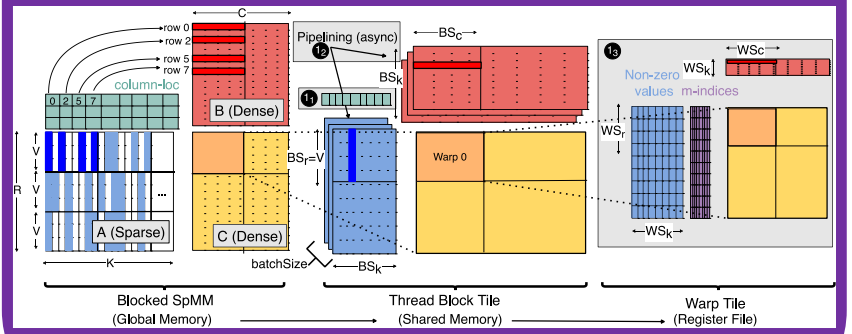
VENOM: The V:N:M Format



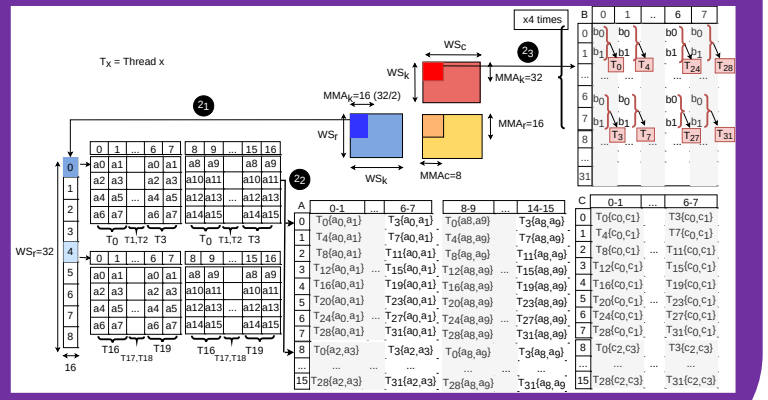
Spatha 🗡️

Design

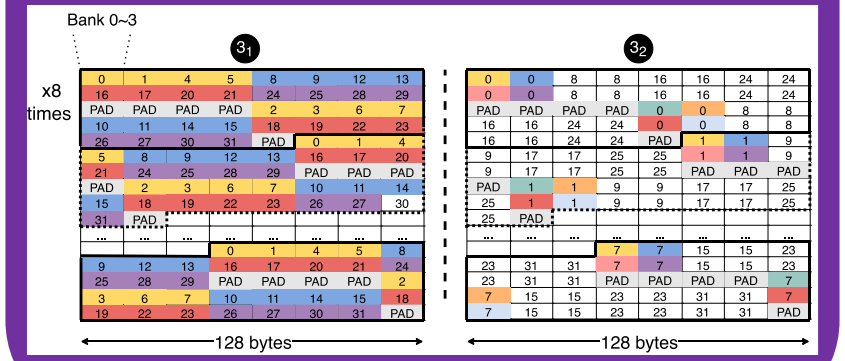
Stage 1. Data Movement



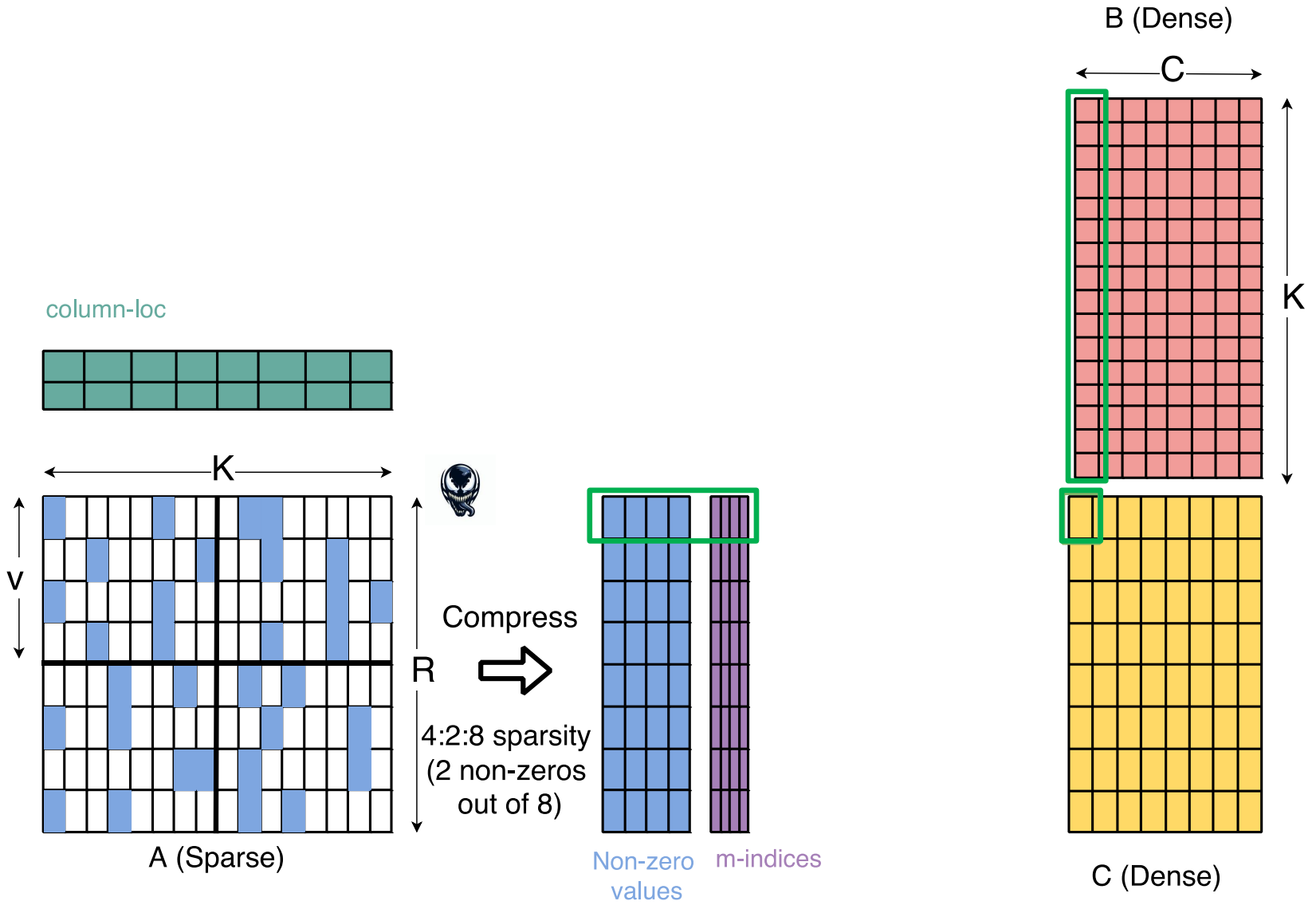
Stage 2. Computation



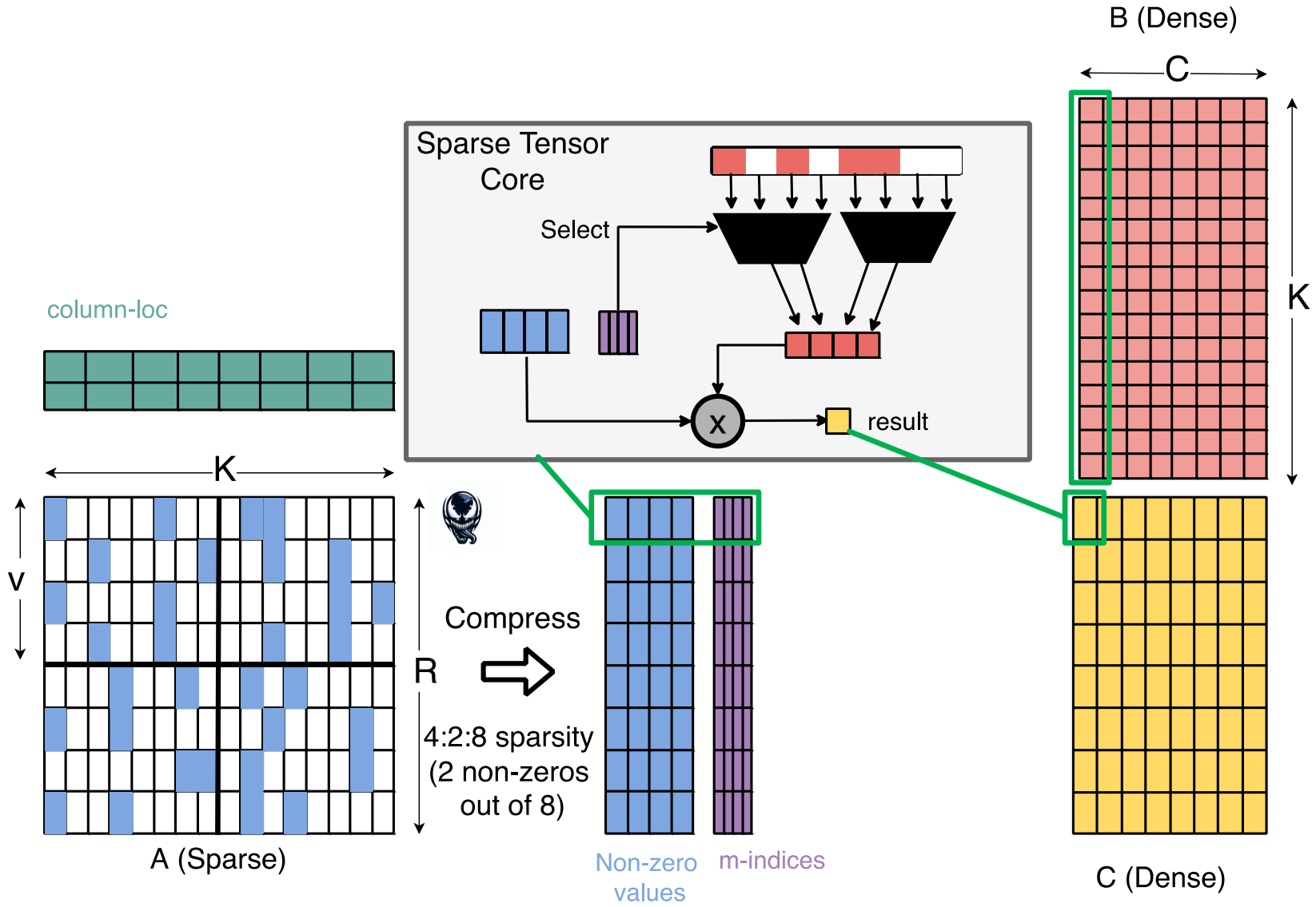
Stage 3. Result storage



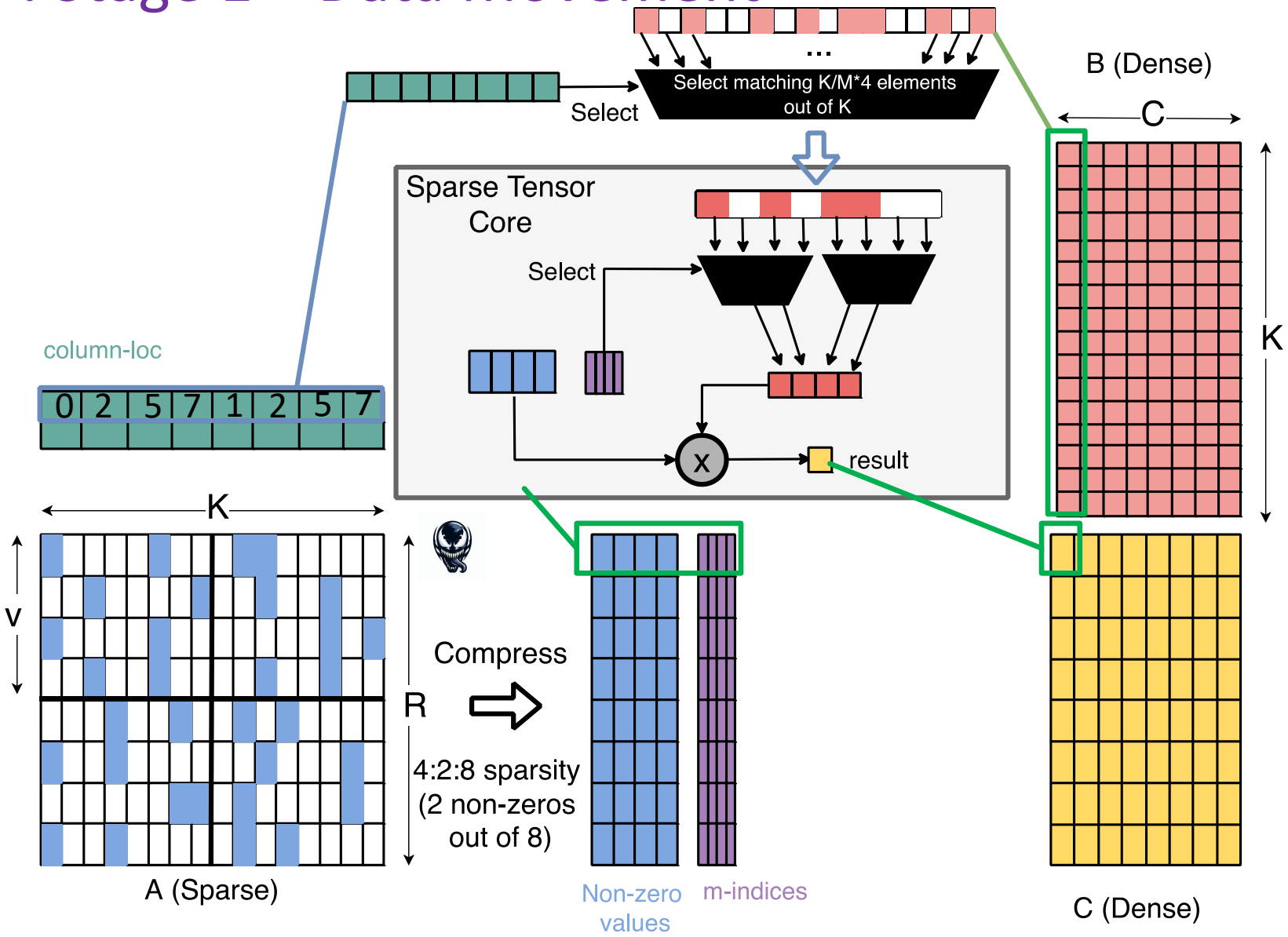
Spatha 🗡️. Stage 1 – Data Movement



Spatha 🗡️. Stage 1 – Data Movement

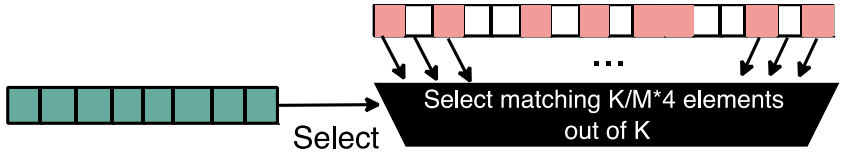


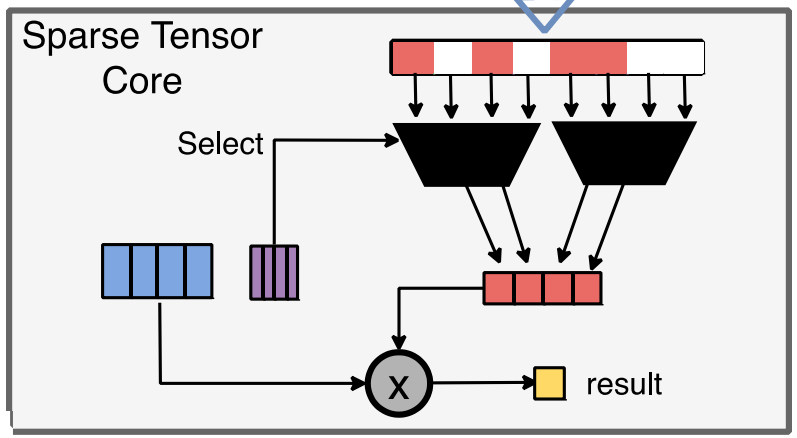
Spatha . Stage 1 – Data Movement

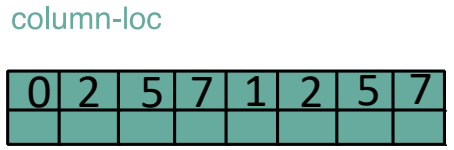


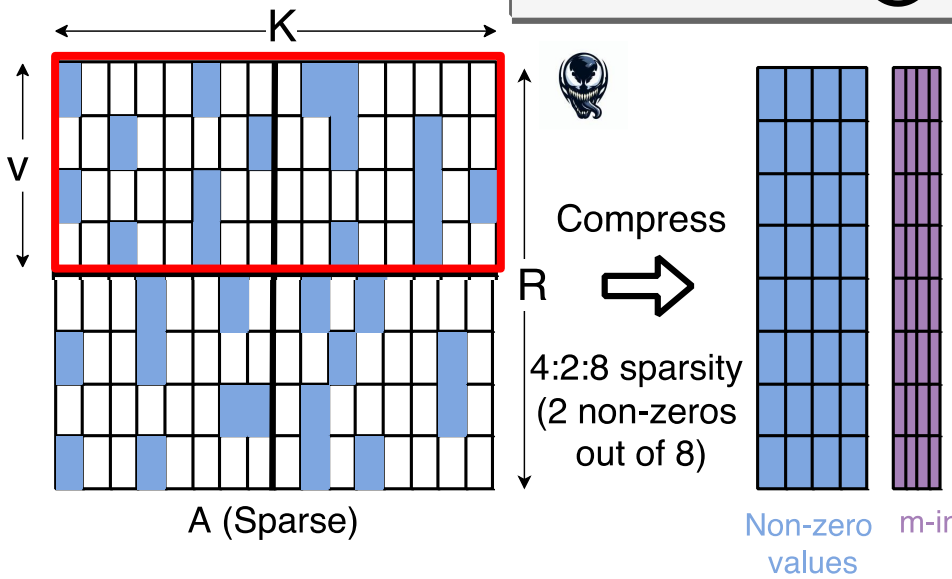
Spatha . Stage 1 – Data Movement

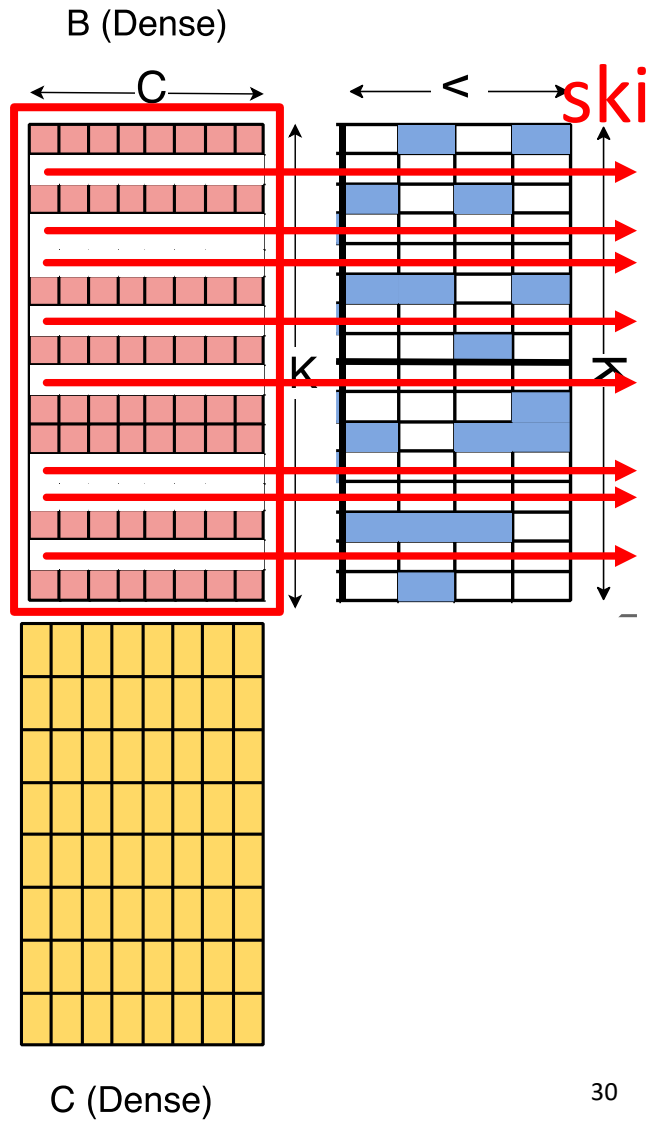
Memory bound problem!!







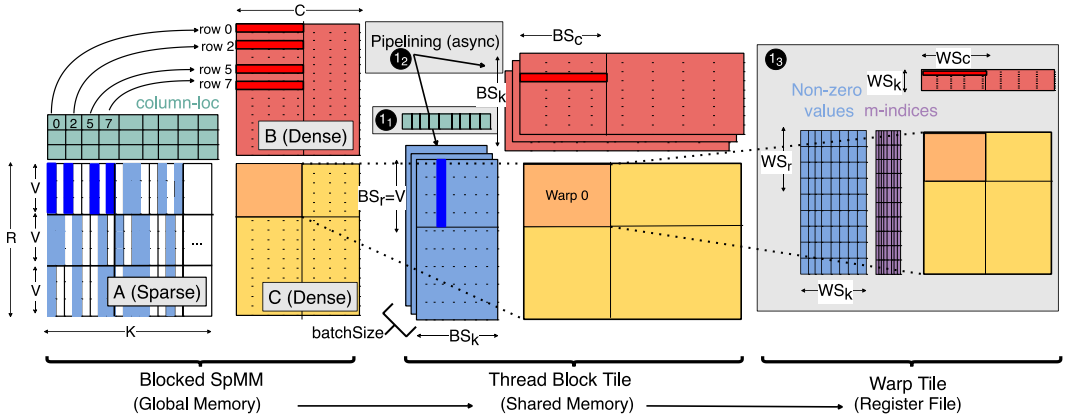


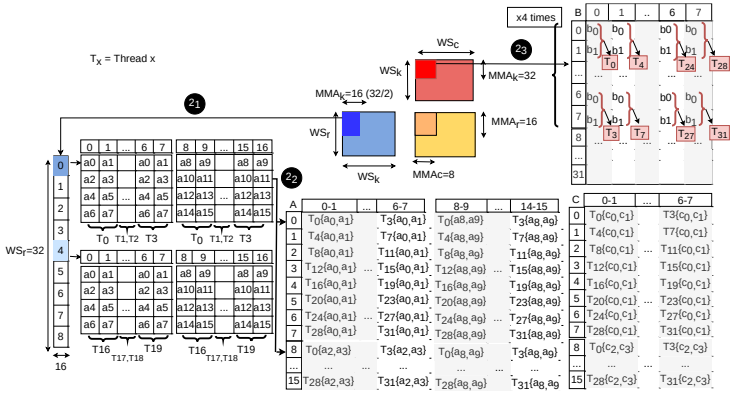


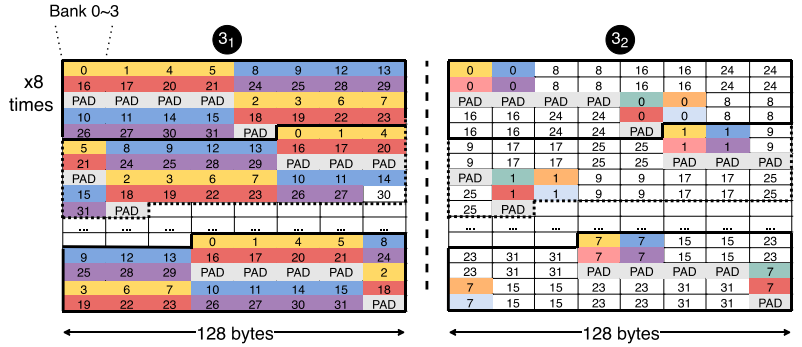
Stage 1 – Data Movement

Stage 2 – Computation

Stage 3 – Result storage



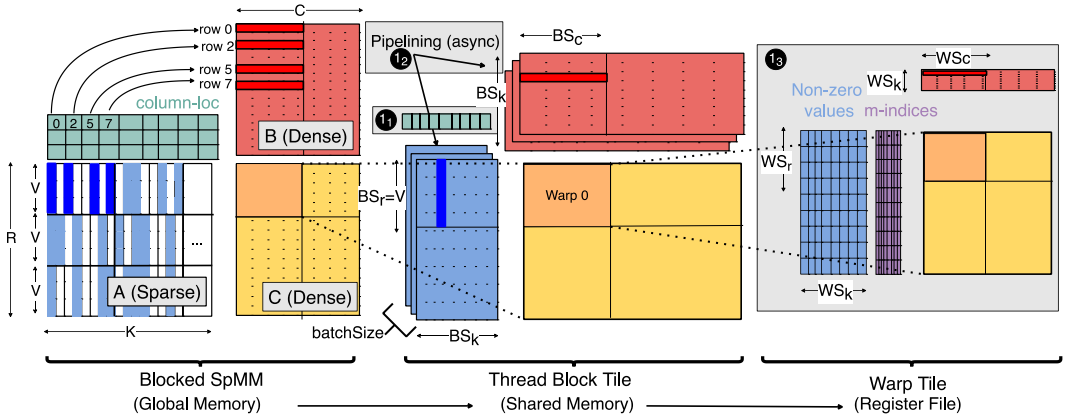


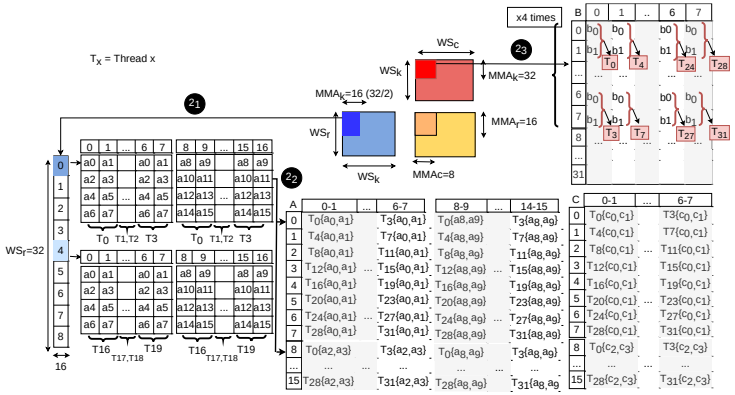


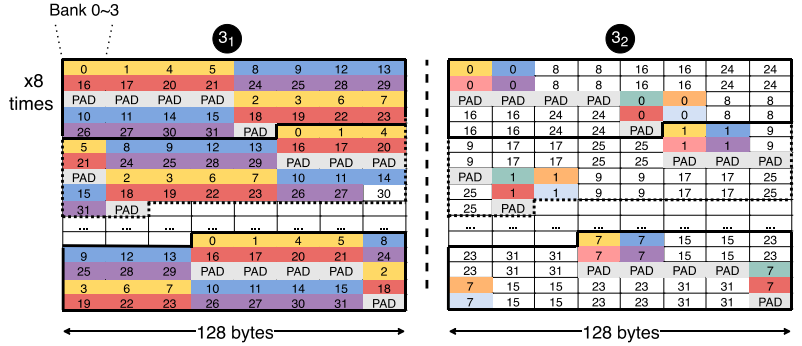
Stage 1 – Data Movement

Stage 2 – Computation

Stage 3 – Result storage



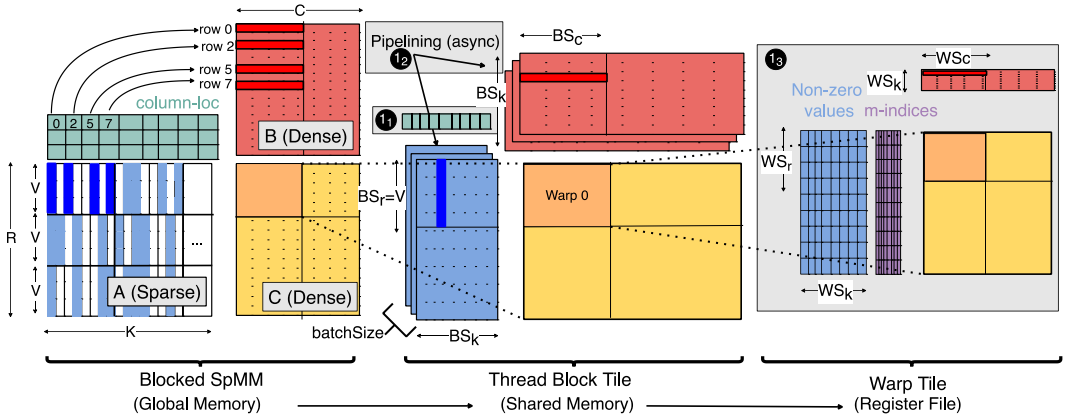


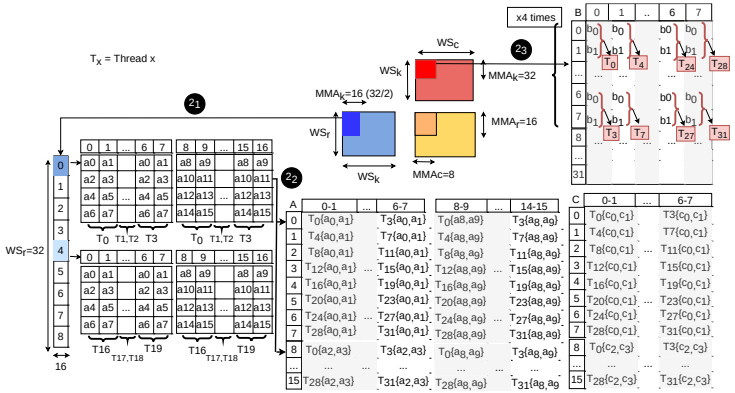


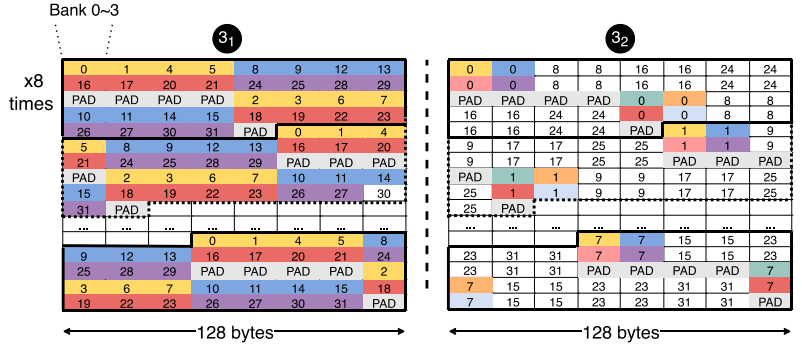
Stage 1 – Data Movement

Stage 2 – Computation

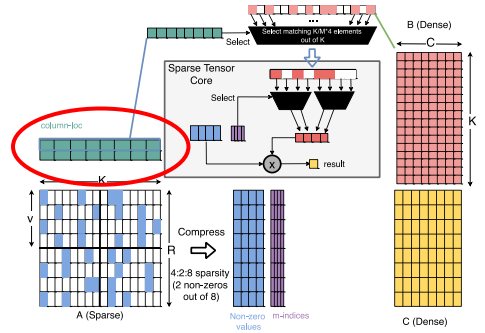
Stage 3 – Result storage

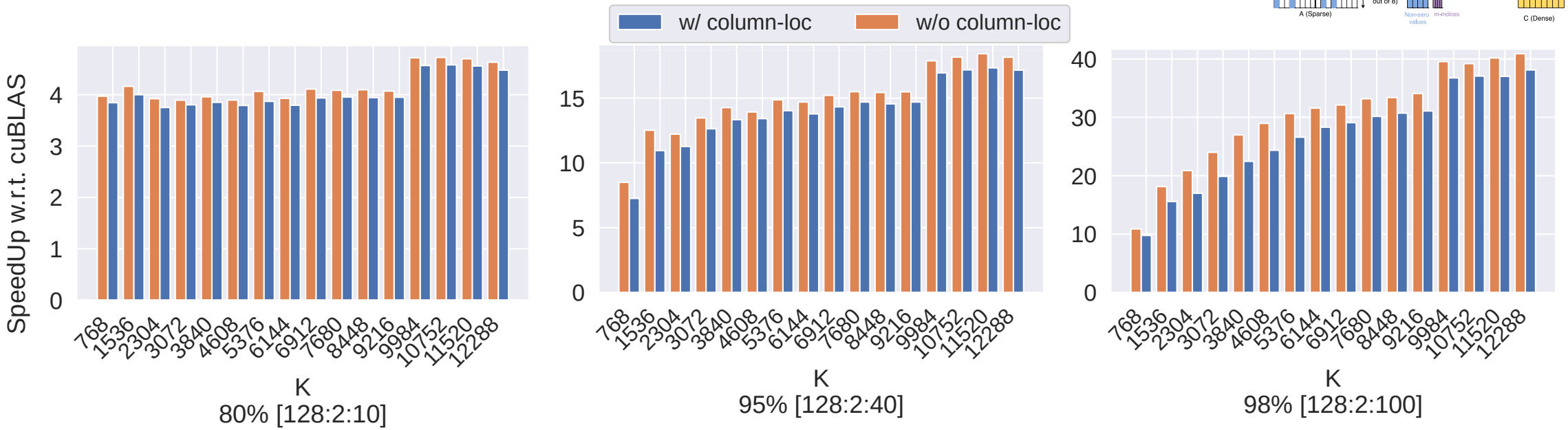






Ablation study – Spatha performance and column-loc overhead





Ideal 5x

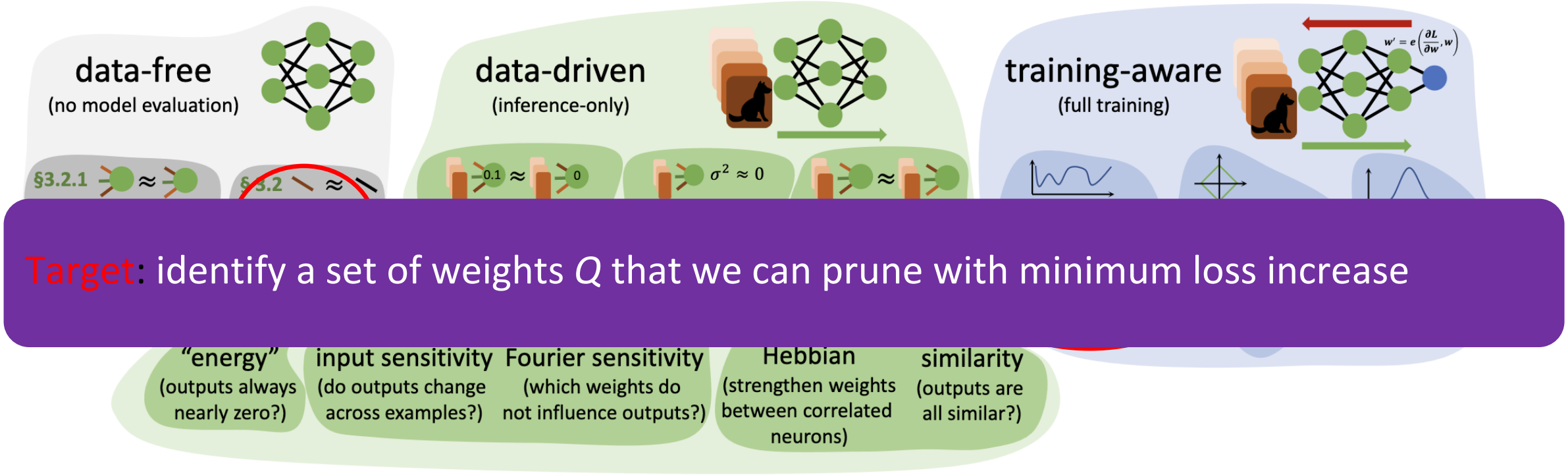
Ideal 20x

Ideal 50x

Sparsity [%] (V:N:M)

Sparsification with VENOM

Sparsification with VENOM



Magnitude pruning

- Based on weights absolute magnitude

✅ Straightforward

✅ Moderate sparsity levels

❌ Becomes challenging with high sparsities

Second order pruning

- Based on second-order derivative ("gradient") information
- Find the set of weights whose removal will incur in a minimal increase in loss

✅ Successfully applied to high sparsity ratios (>90%)

Hoefler et. al "Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks", arXiv 2102.00554, Jan 2021

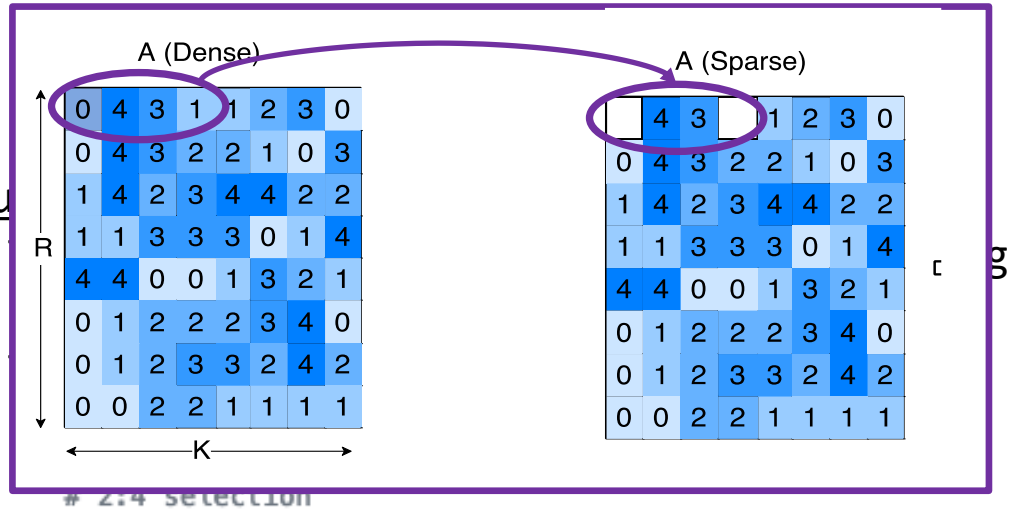
Second-order pruning

Saliency score (oBERT):

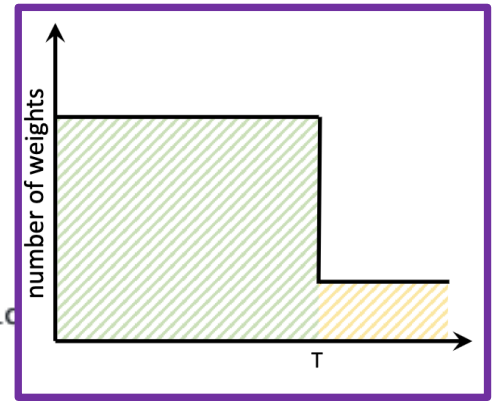
$$\rho_{\mathrm{Q}} = \frac{1}{2}\left(\mathbf{E}_{\mathrm{Q}}\mathbf{w}^{*}\right)^{\top}\left(\mathbf{E}_{\mathrm{Q}}\widehat{\mathbf{F}}^{-1}(\mathbf{w}^{*})\mathbf{E}_{\mathrm{Q}}^{\top}\right)^{-1}\mathbf{E}_{\mathrm{Q}}\mathbf{w}^{*}.$$

where,

- $w^* \in \mathbb{R}^d$ is a well optimized dense model
- d total number of weights
- $F^{-1}(w) \in \mathbb{R}^{dxd}$ is the Fisher matrix
- $E_Q \in \mathbb{R}^{|Q|x\,d}$ is a matrix composed of the corresponding canonical basis vectors $e_k(\forall k \in Q)$ arranged in rows




```
# 2:4 selection
e_24 = torch.tensor([
    [1, 1, 0, 0],
    [1, 0, 1, 0],
    [1, 0, 0, 1],
    [0, 1, 1, 0],
    [0, 1, 0, 1],
    [0, 0, 1, 1],
], dtype=torch.long, devic
```



- One-shot pruning

✅ Computationally affordable (6 combination)

✅ Good model accuracy recovery (50% sparsity)

Eldar et. al "The Optimal BERT Surgeon: Scalable and Accurate Second-Order Pruning for Large Language Models", 2022 (oBERT)

Second-order pruning

For N:M sparsity:

- The Fisher matrix will be MxM, representing correlations between all the weights in the group
- Evaluating all the E_Q canonical vectors turns into an M-combinatorial problem

❌ Possibilities to consider for each group Q is $\binom{M}{N}$

For V:N:M sparsity:

- The Fisher matrix will be $(MxV)x(MxV)$, representing correlations between all the weights in the group
- The number of canonical vectors to evaluate increases even further:

$$\binom{M'}{N}^V \cdot \binom{M}{M'}, M' = 4$$

☠ Intractable combinatorial problem

Second-order pruning

For V:N:M sparsity, we propose:

✅ A pair-wise approach to describe the set of canonical vectors:

$$E_Q = [[1, 0], [0, 1], [1, 1]]$$

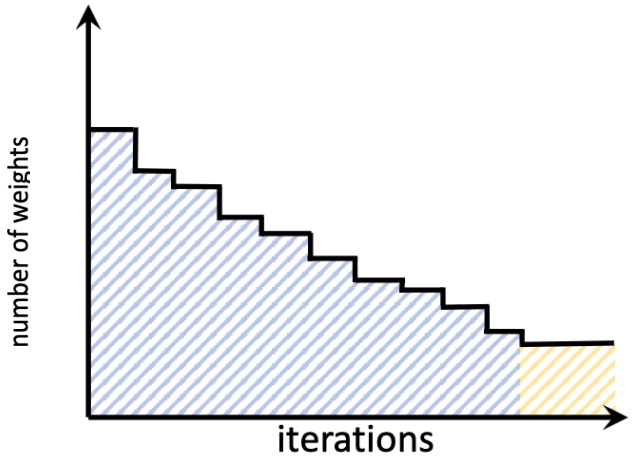
✅ A gradual pruning setup over one-shot pruning which performs V:N:M pruning across different β steps, for increasing sparsity levels.

Example: V:N:M = V:2:6

Steps (β=4): V:5:6 -> V:4:6 -> V:3:6 -> V:2:6

Saliency score (oBERT):

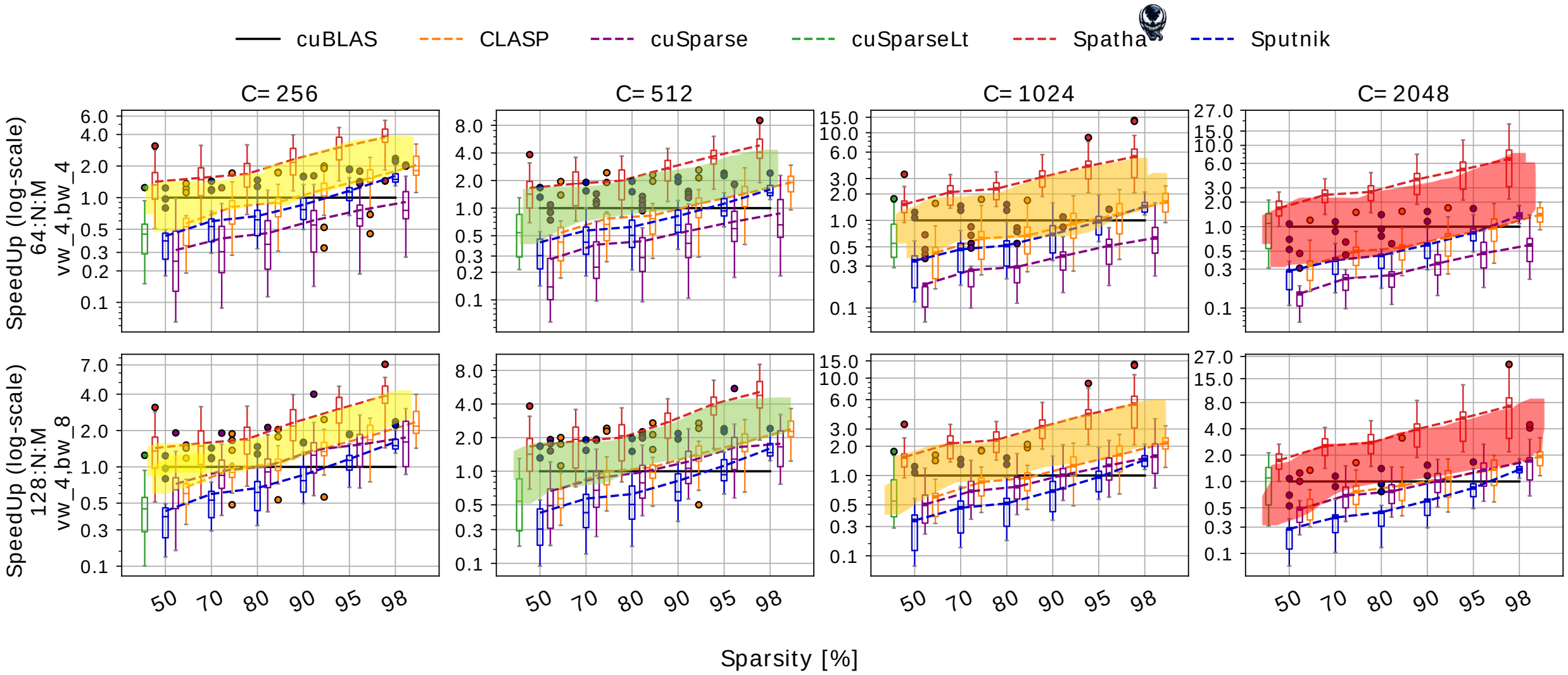
$$\rho_{\mathrm{Q}} = \frac{1}{2} \left(\mathbf{E}_{\mathrm{Q}} \mathbf{w}^*\right)^{\top} \left(\mathbf{E}_{\mathrm{Q}} \widehat{\mathbf{F}}^{-1}(\mathbf{w}^*) \mathbf{E}_{\mathrm{Q}}^{\top}\right)^{-1} \mathbf{E}_{\mathrm{Q}} \mathbf{w}^*.$$



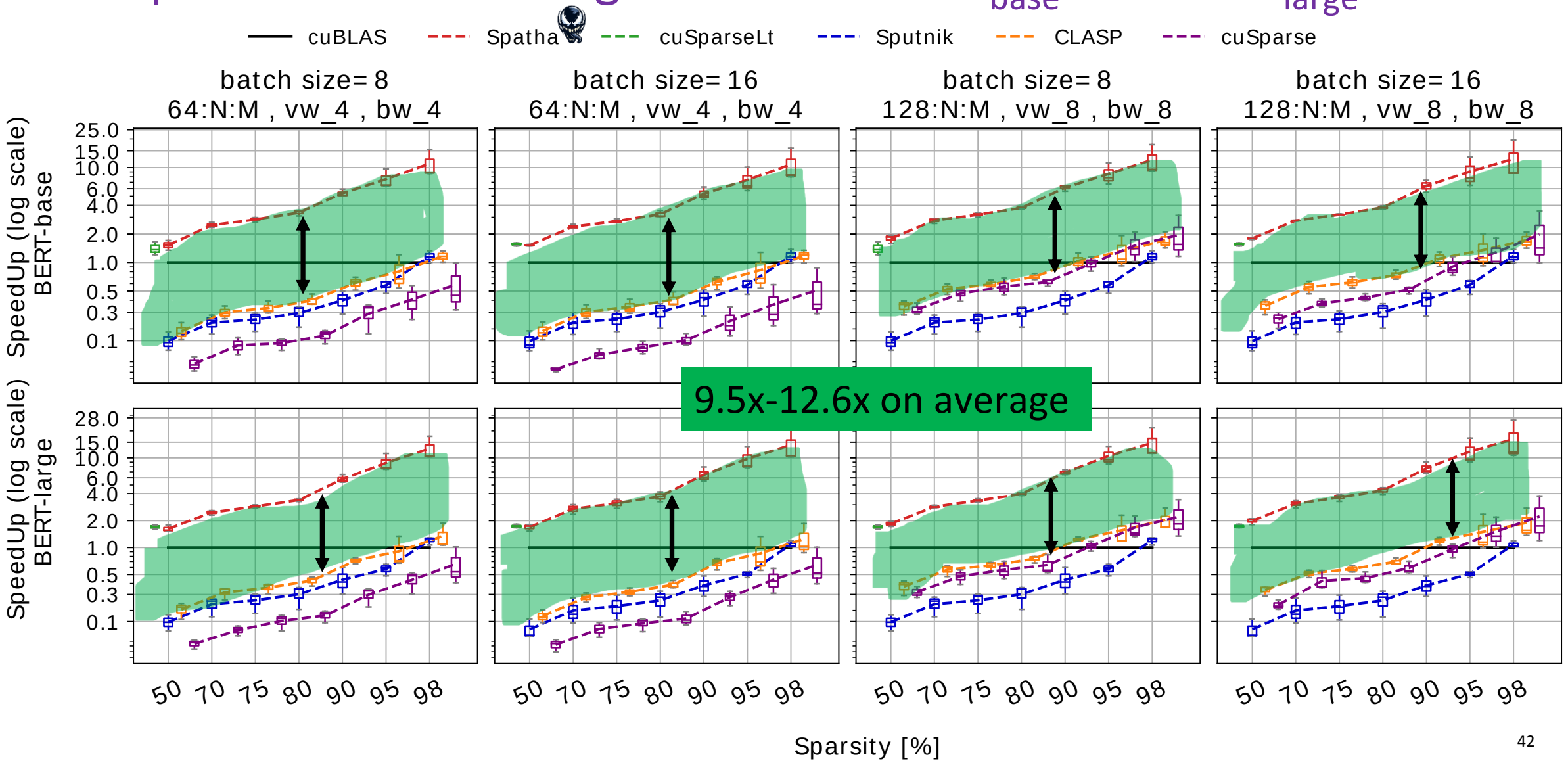
Eldar et. al “The Optimal BERT Surgeon: Scalable and Accurate Second-Order Pruning for Large Language Models”, 2022 (oBERT)

Evaluation

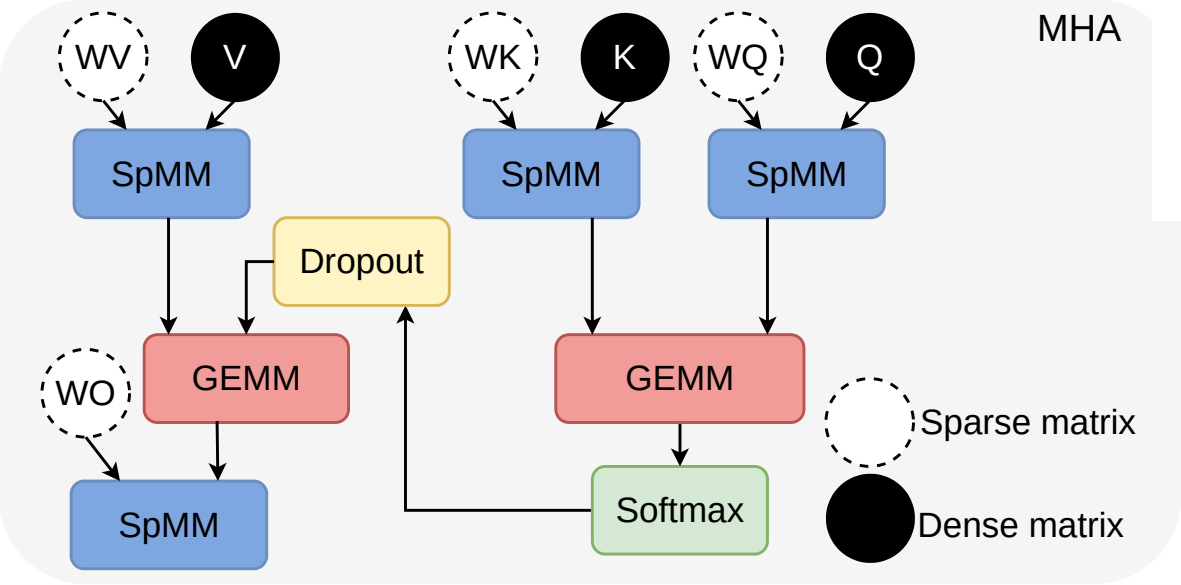
Comparison with existing libraries – DLMC dataset



Comparison with existing libraries - $BERT_{base}$ and $BERT_{large}$



Case study. Sparse LLMs



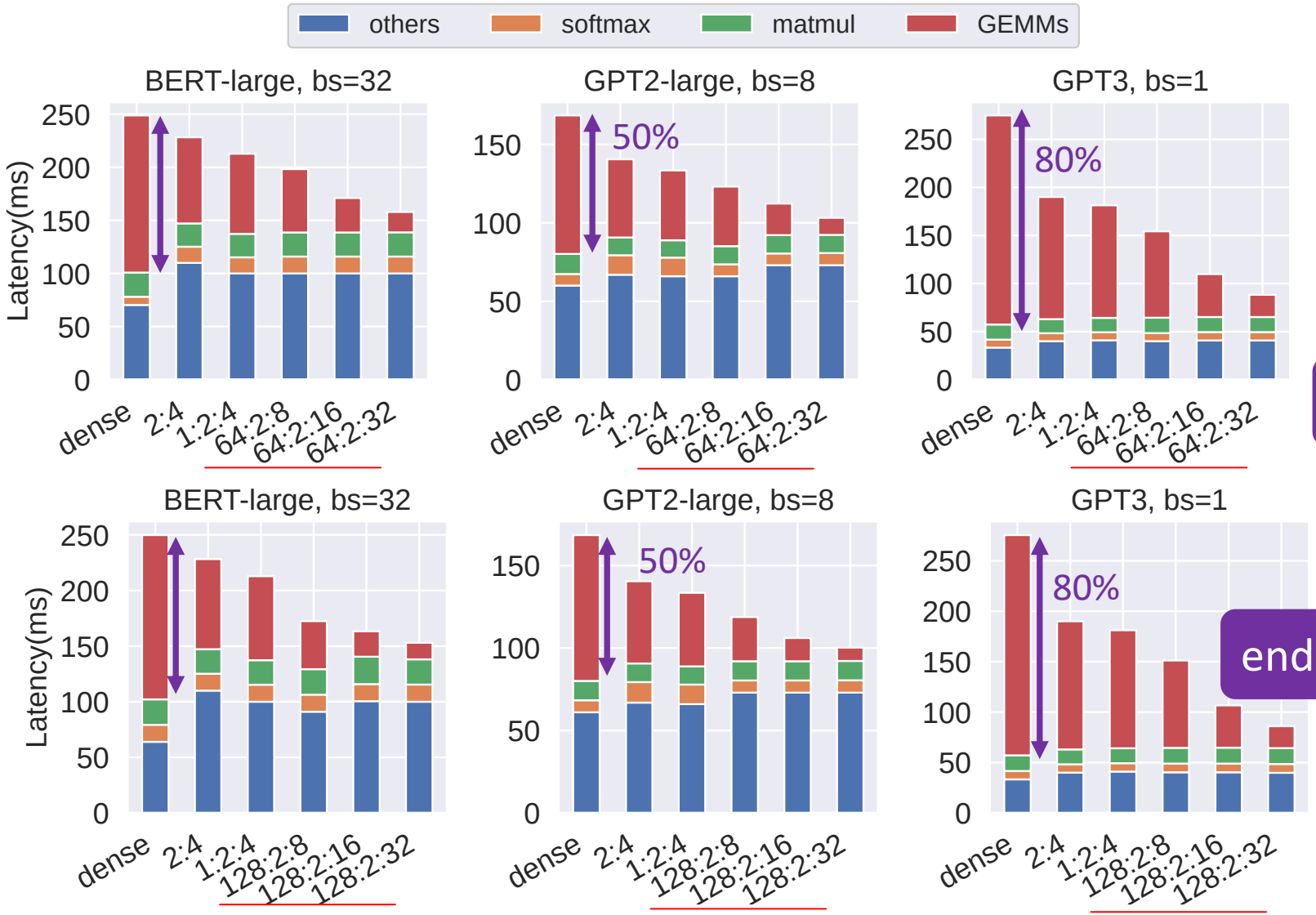
Sparsity	1:N:M	64:N:M	128:N:M	vw_8
75% (2:8)	88.61	88.47	87.94	88.55
87.5% (2:16)	87.73	86.50	85.01	86.90

Table 2: F1 score of BERT$_{base}$ on the SQuADv1.1. Dense model F1=88.43

75% (2:8) →
- 1:N:M, 64:N:M and vw_8 -> slightly **improve** the **original model** accuracy
- 128:N:M format presents a 0.005% accuracy loss

87.5% (2:16) →
- 1:2:16 recovers 99% of the original accuracy
- 64:2:16 and vw_8 pruning recover 98%
- 128:2:16 approach recovers 96%

Case study. Sparse LLMs



Tensor time contraction

BERT-large → 9.95×

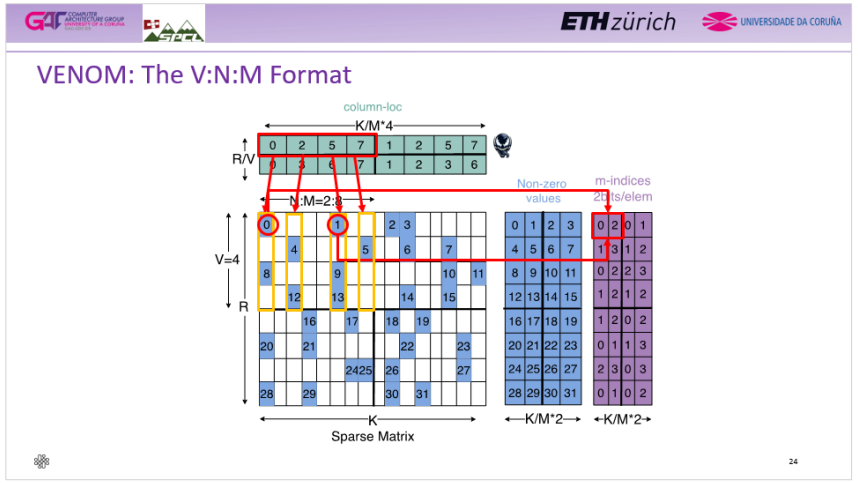
GPT2-large → 10.84×

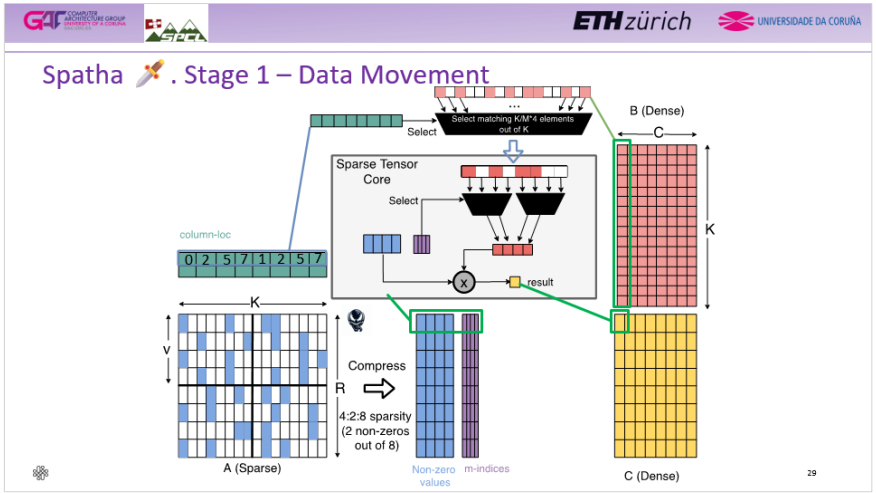
end-to-end latency limited by GEMM time

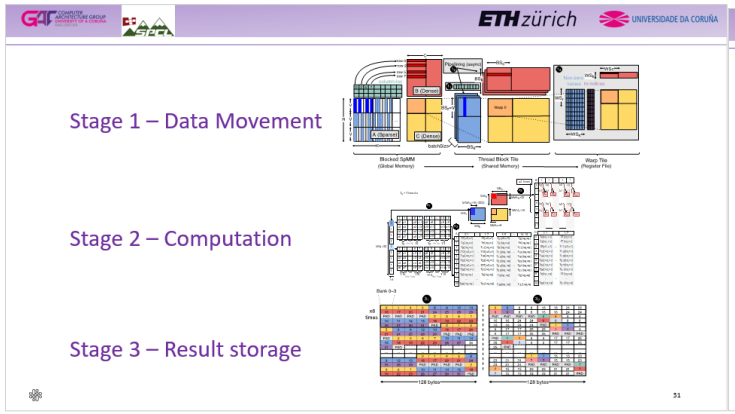
GPT3 → 11×

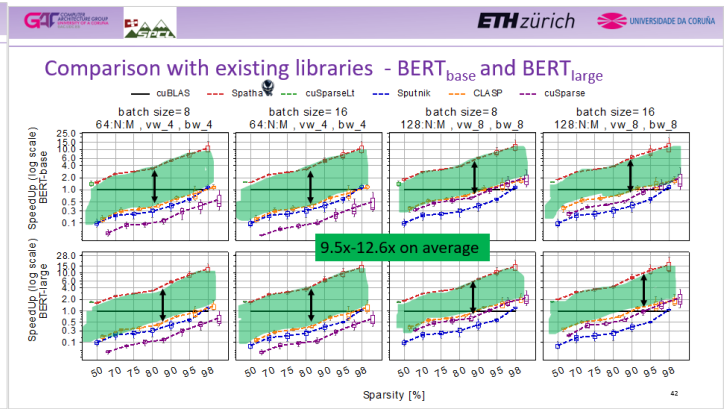
end-to-end time reduction of up to 3.20×

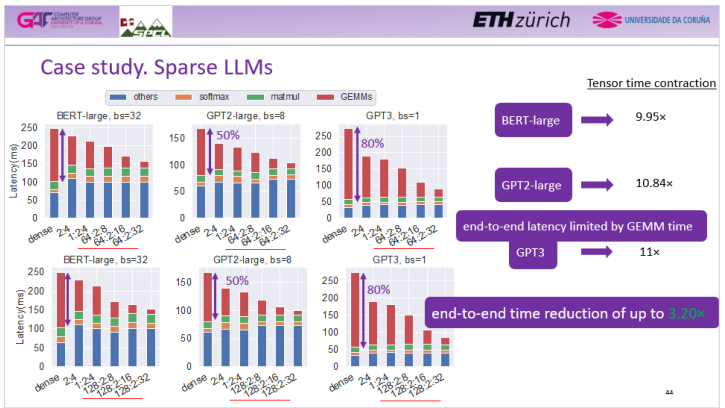
VENOM: A Vectorized N:M format for Unleashing the Power of Sparse Tensor Cores











https://github.com/UDC-GAC/venom