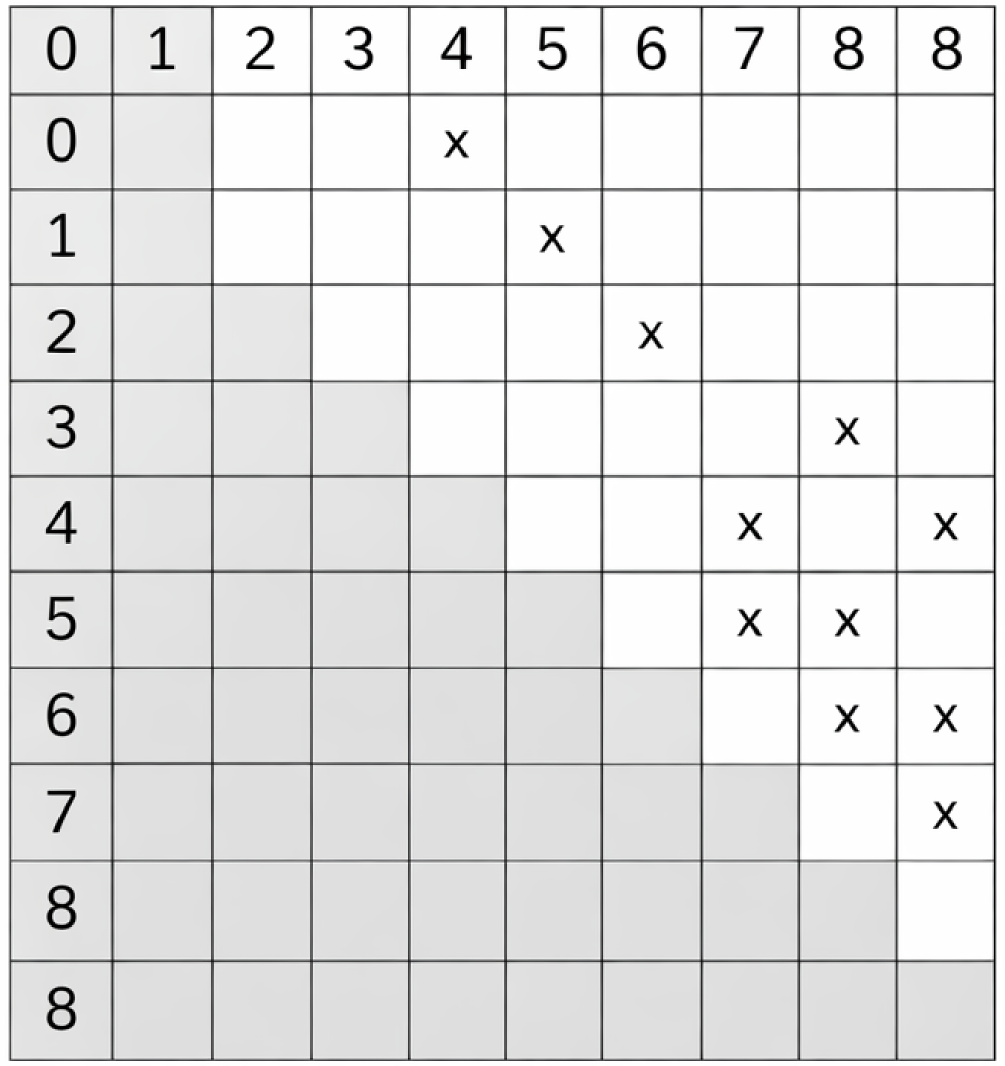
RICE UNIVERSITY

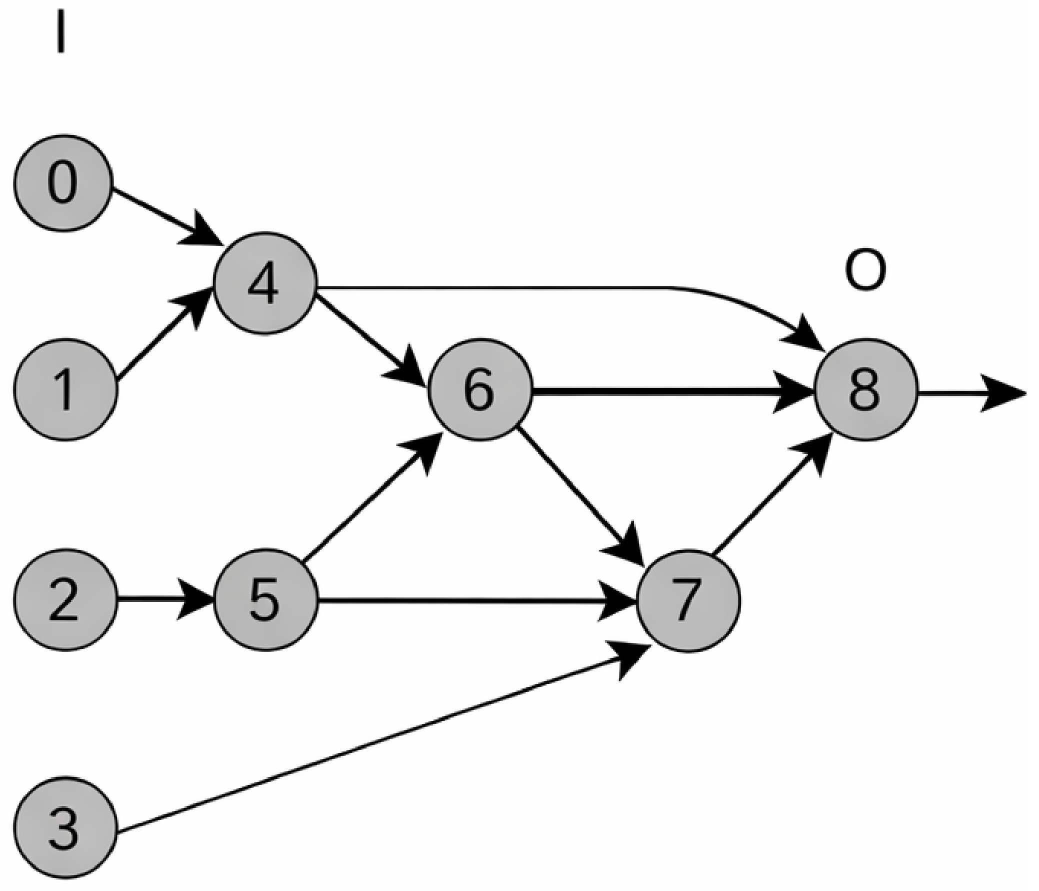
Week-3: Sparse Matrix Multiplication

Why Sparse Matrix Multiplication?

- Large graphs and pruned/structured-sparse models.
- Memory bandwidth dominates → exploit zeros.
- SpMM central to GNN layers: Y = A × X (A sparse, X dense).
- Goal: high throughput with irregular memory access

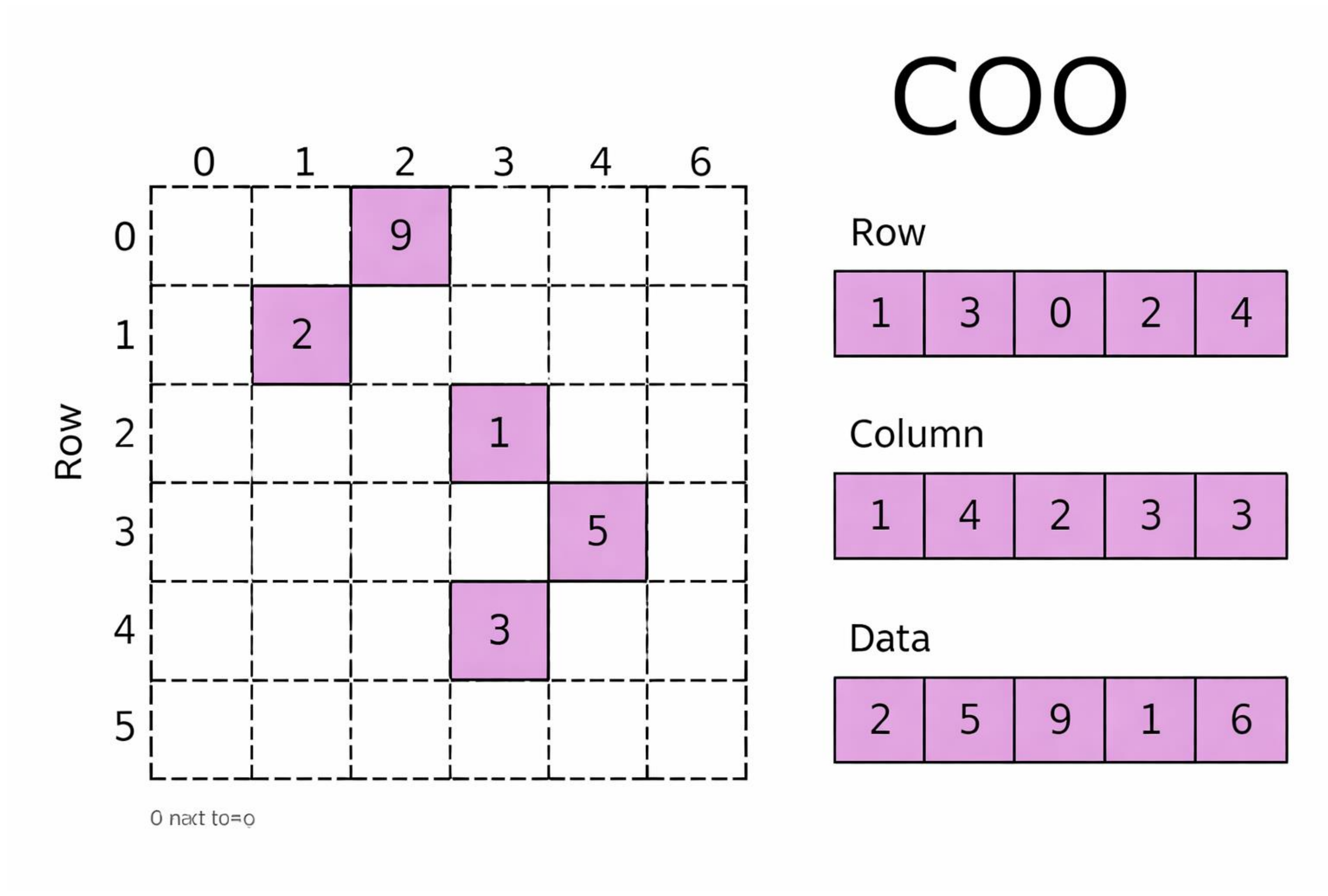


0	1	2	3	4	5	6	7	8	8
0				x					
1					x				
2						x			
3								x	
4							x		x
5							x	x	
6								x	x
7									x
8									
8									



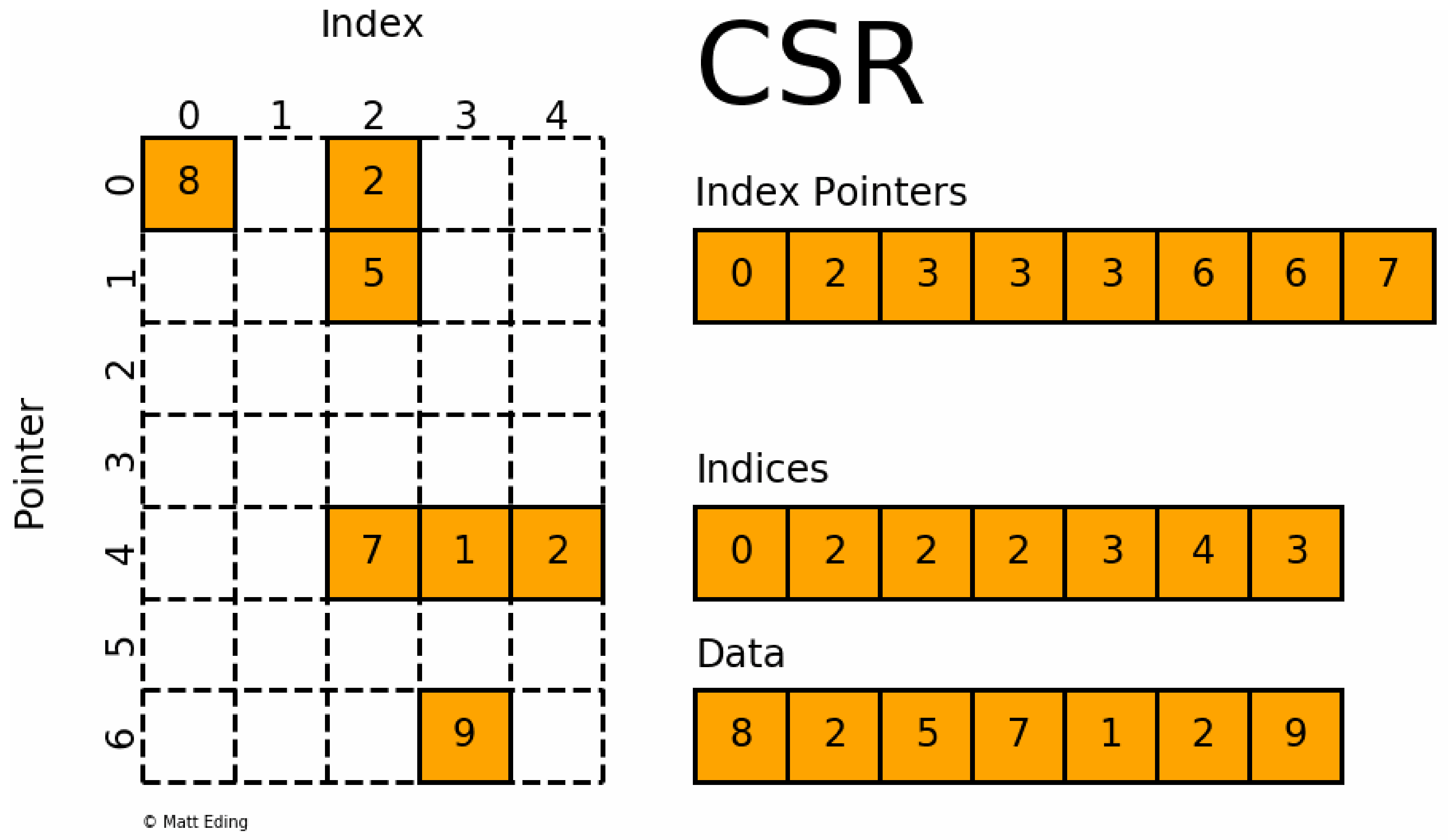
Sparse Formats -- COO

- COO (Coordinate / Triplet format) stores a sparse matrix as an explicit list of non-zero entries.
- COO uses three equal-length arrays:
  - – row[i] – row index of the i-th non-zero
  - – col[i] – column index of the i-th non-zero
  - – val[i] – value of the i-th non-zero



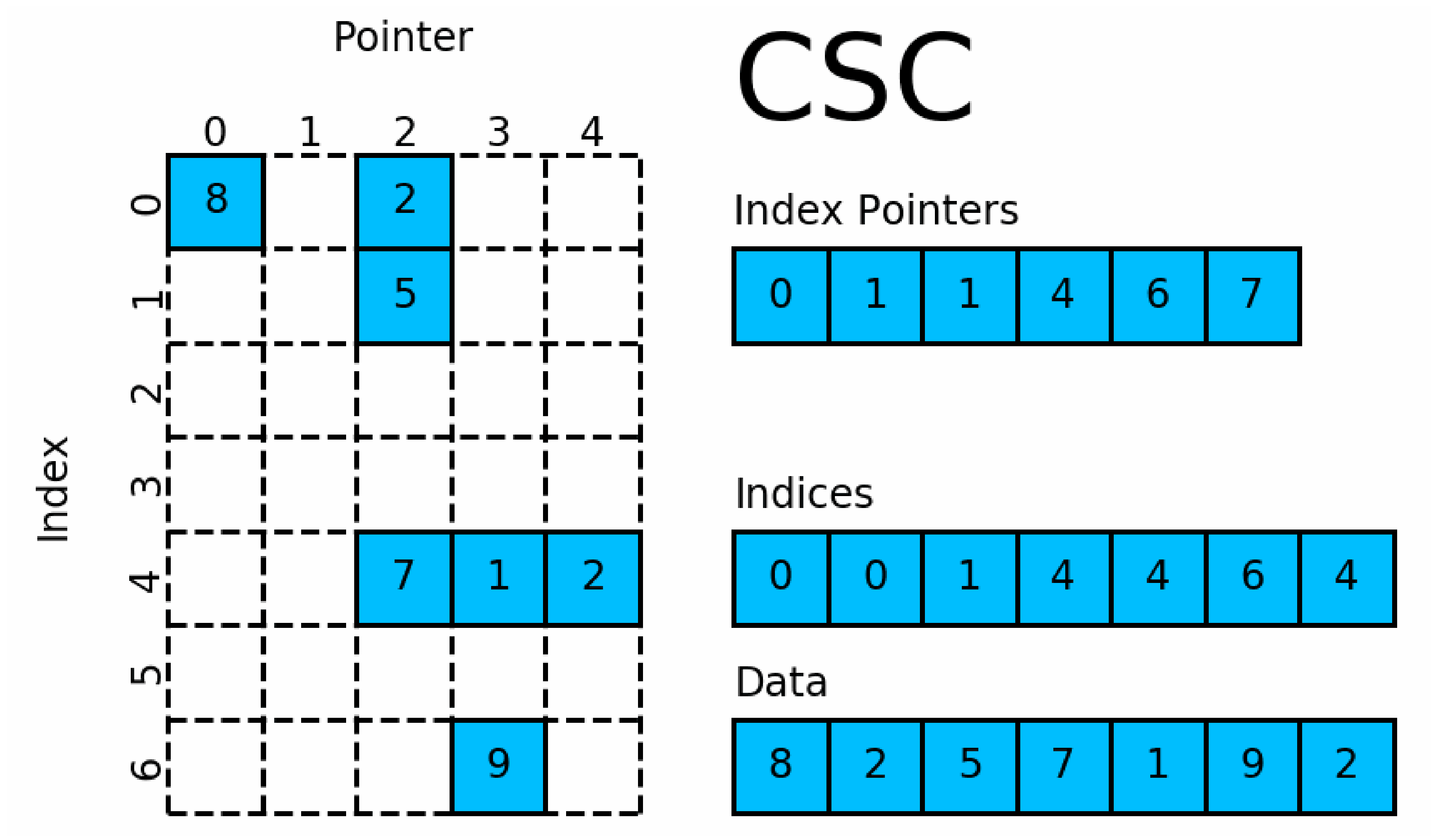
Sparse Formats -- CSR

- CSR (Compressed Sparse Row) is a common way to store a sparse matrix efficiently by keeping only the non-zero entries. It uses three 1D arrays:
- values
  - Stores all non-zero matrix entries, row by row.
  - Length = number of non-zeros (nnz).
- col_ind (column indices)
  - Same length as values.
  - col_ind[k] gives the column index of values[k].
  - So (values[k], col_ind[k]) together represent a non-zero matrix entry.
- row_ptr (row pointer / row offsets)
  - Length = (#rows + 1).
  - row_ptr[i] gives the starting index in values (and col_ind) of row i.



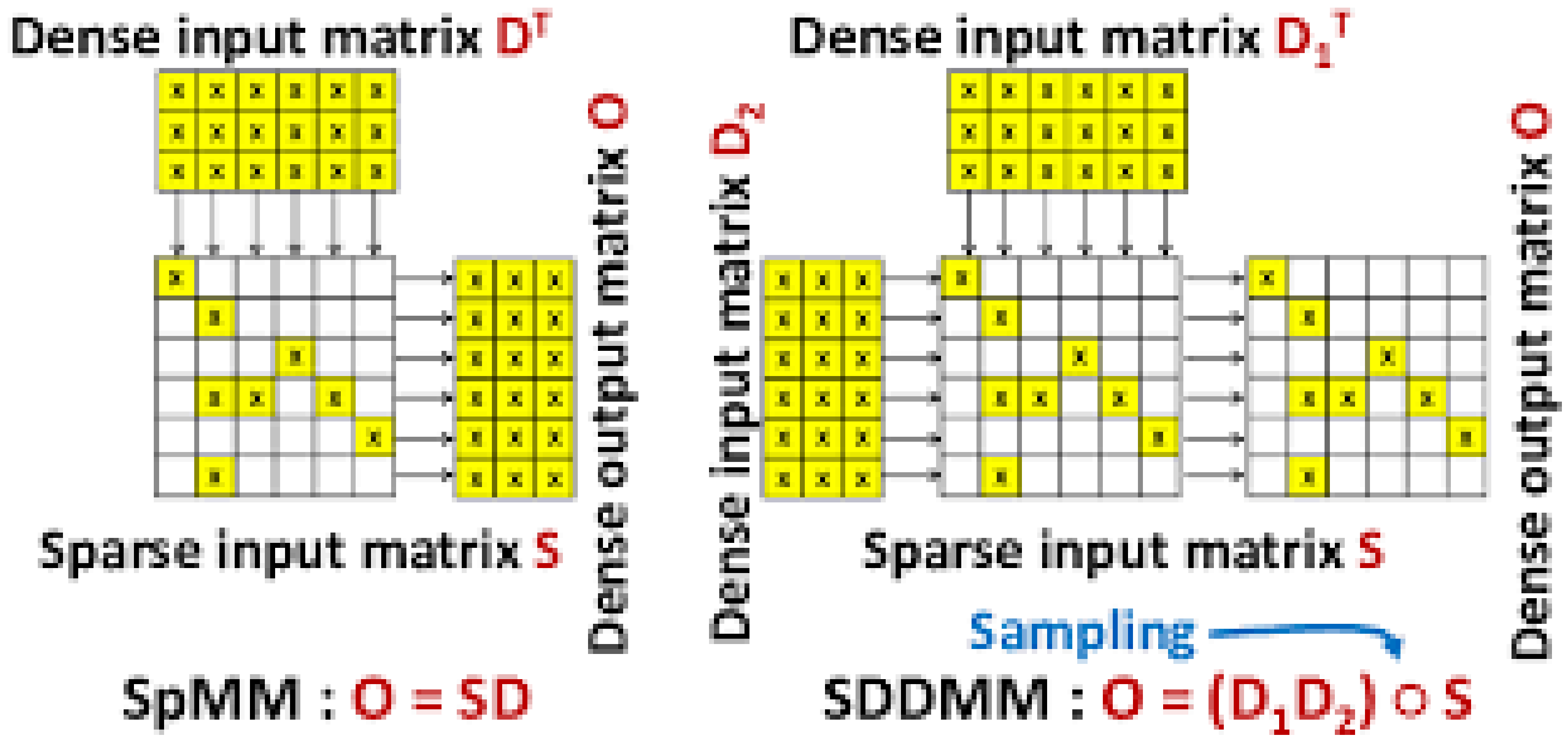
Sparse Formats -- CSC

- CSC (Compressed Sparse Column format) stores a sparse matrix by compressing non-zero entries by column.
  - Data: all non-zero values ordered by column;
  - index[i], which gives the row index of the i-th non-zero;
  - Index Pointers: col_ptr[j] and col_ptr[j+1] indicate the range in values (and row_ind) corresponding to column j.



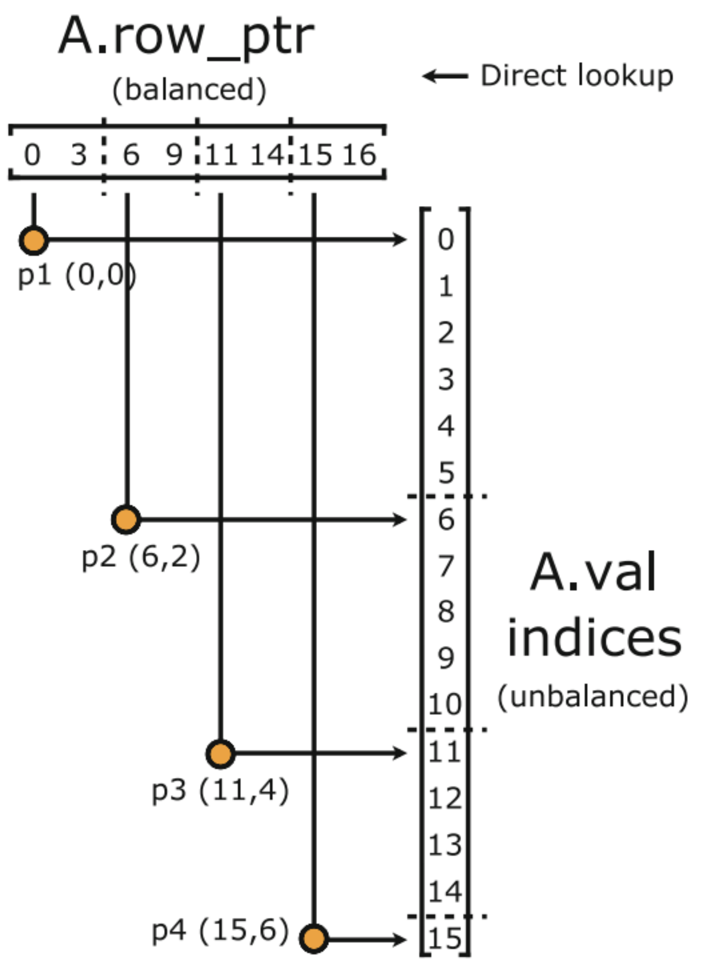
SpMV vs SpMM

- SpMV: y = A × X (dense vector)
- SpMM: Y = A × X (dense matrix, feature dim F)
- SpMM exposes more reuse along F → better GPU utilization
- Common in GNNs: message passing via SpMM

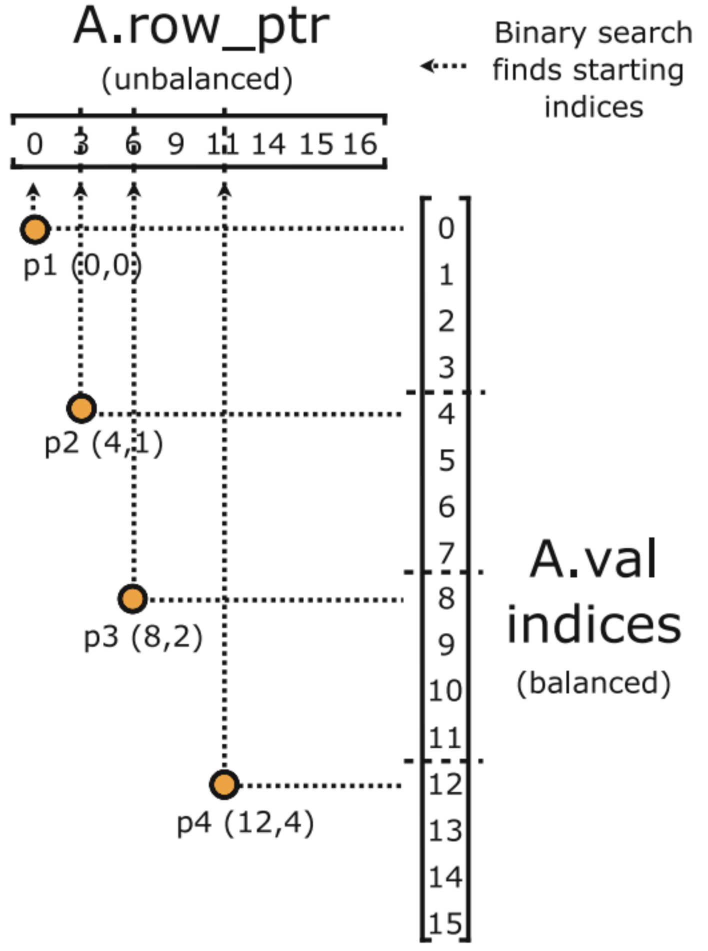


Work Decomposition on GPU

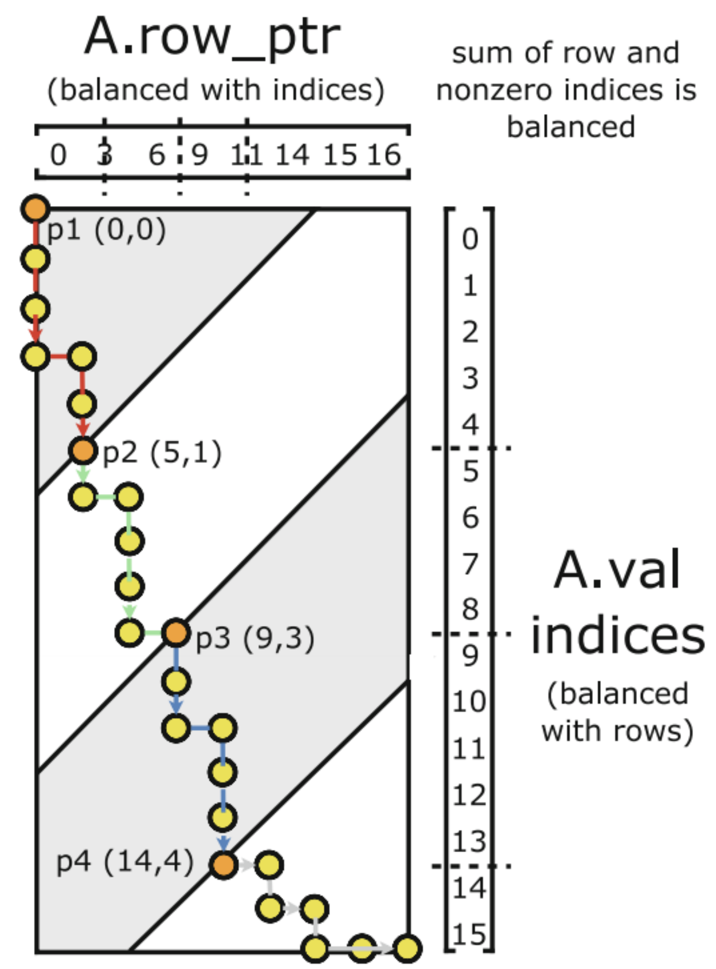
- Per-row, per-nonzero, or per-tile assignment
- Use warps for rows with many nnz; threads for small rows
- Load balance: segmented reduction, binning by row length
- Coalesce loads from X using col_ind



(a) Row split

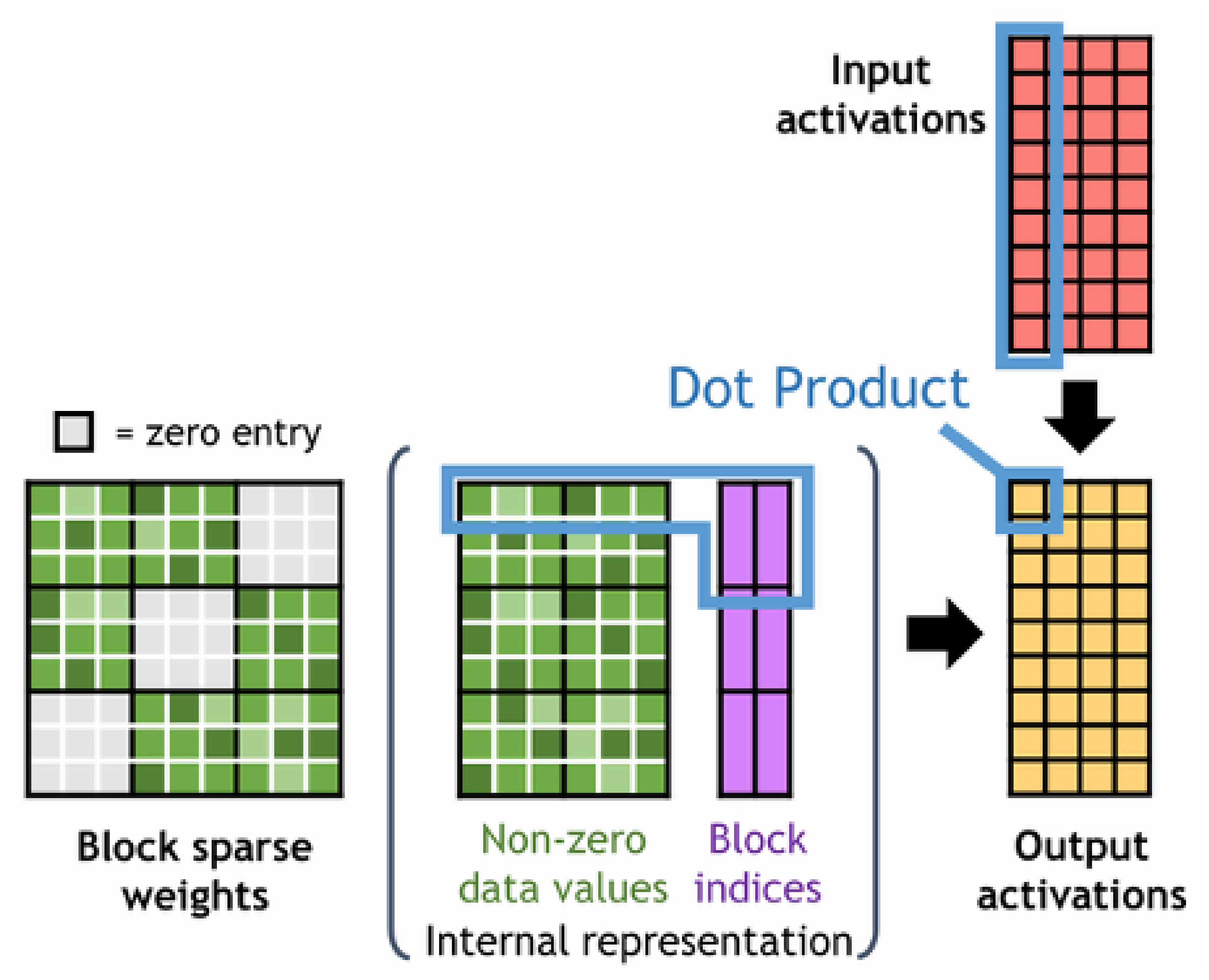


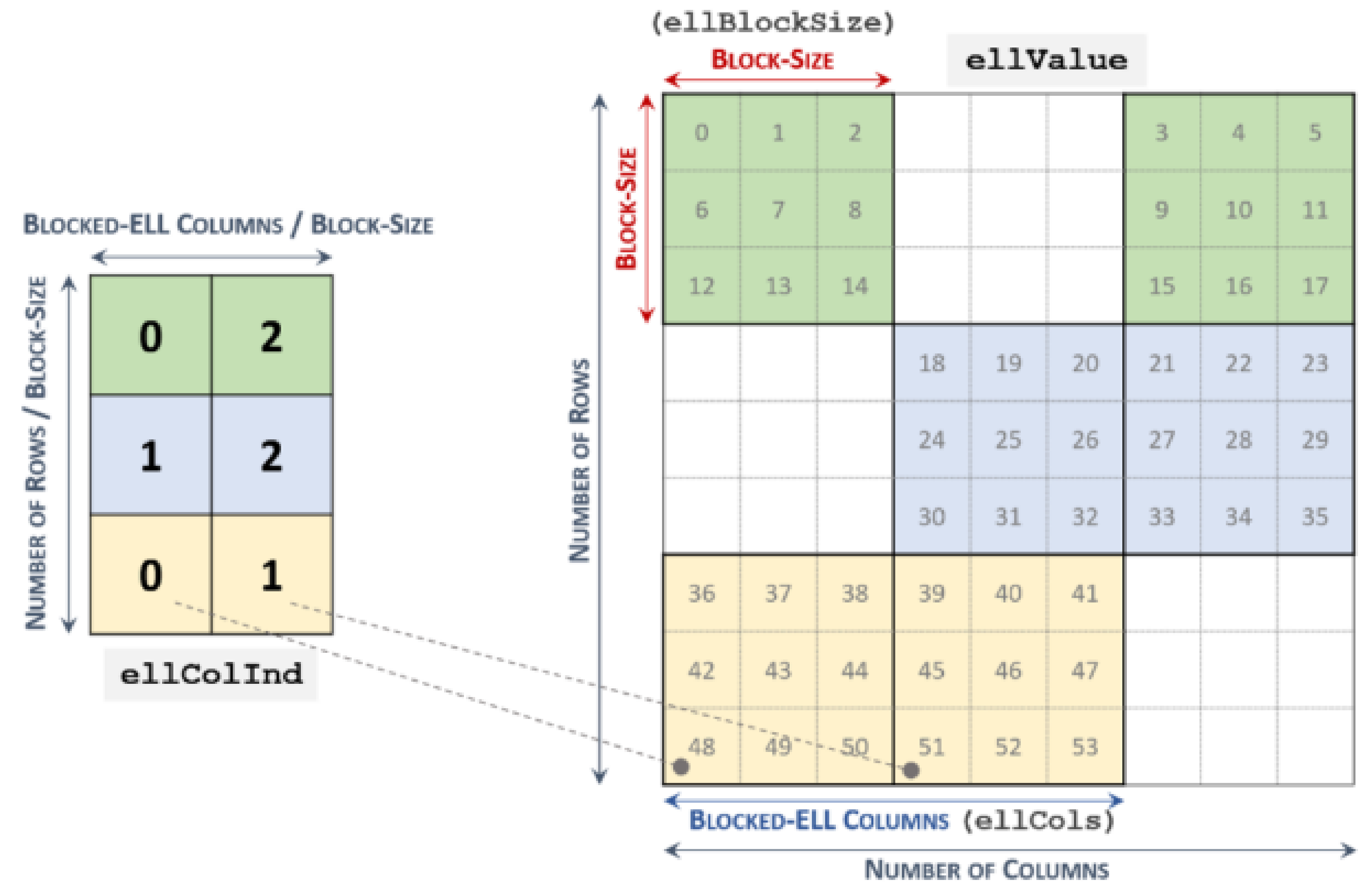
(b) Nonzero split



(c) Merge path

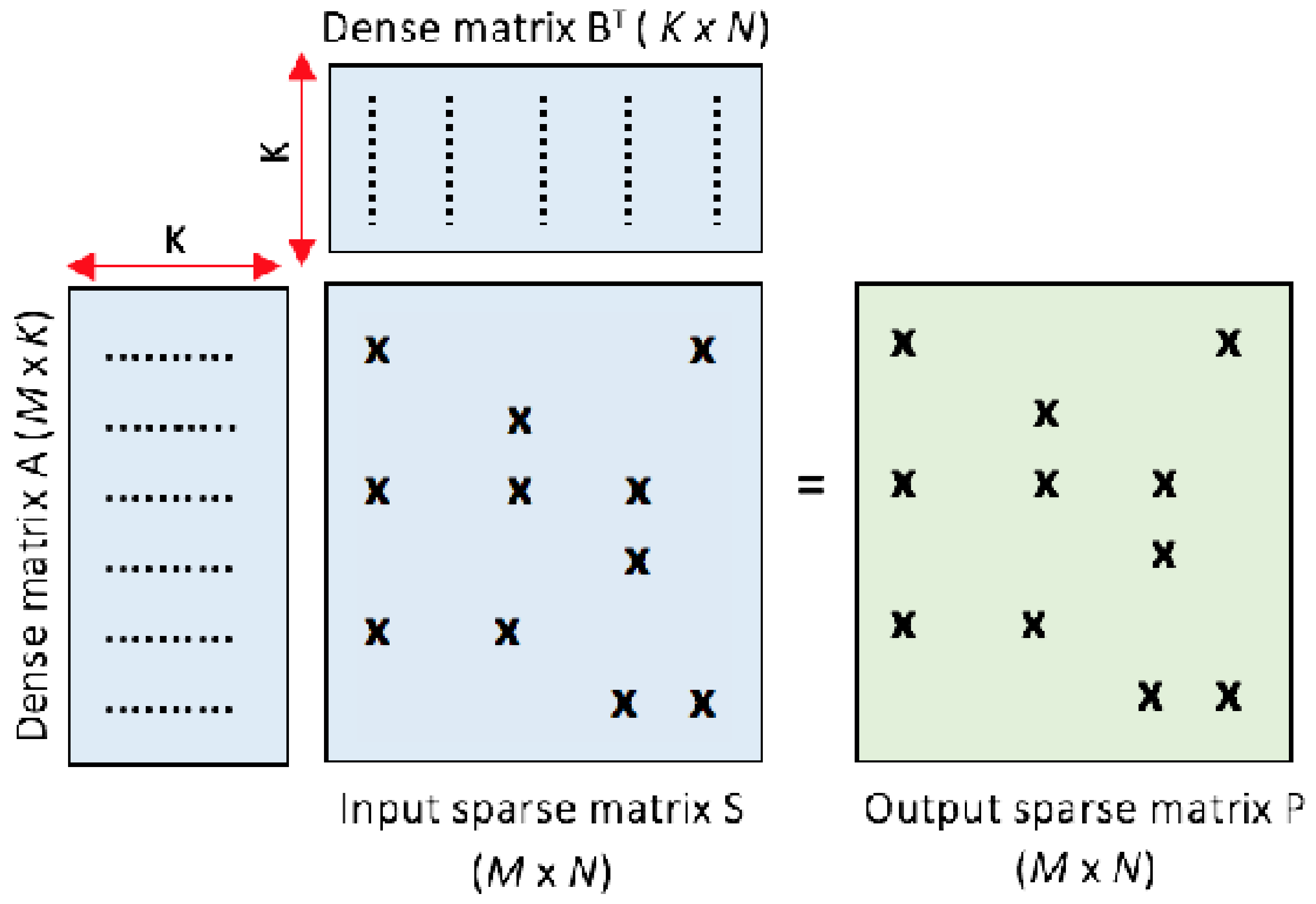
Blocked SpMM




(ellBlockSize)
Block-Size
ellValue
Blocked-ELL Columns / Block-Size
Number of Rows / Block-Size
ellColInd
Number of Rows
Blocked-ELL Columns (ellCols)
Number of Columns

Sampled Dense-Dense Matrix Multiplication

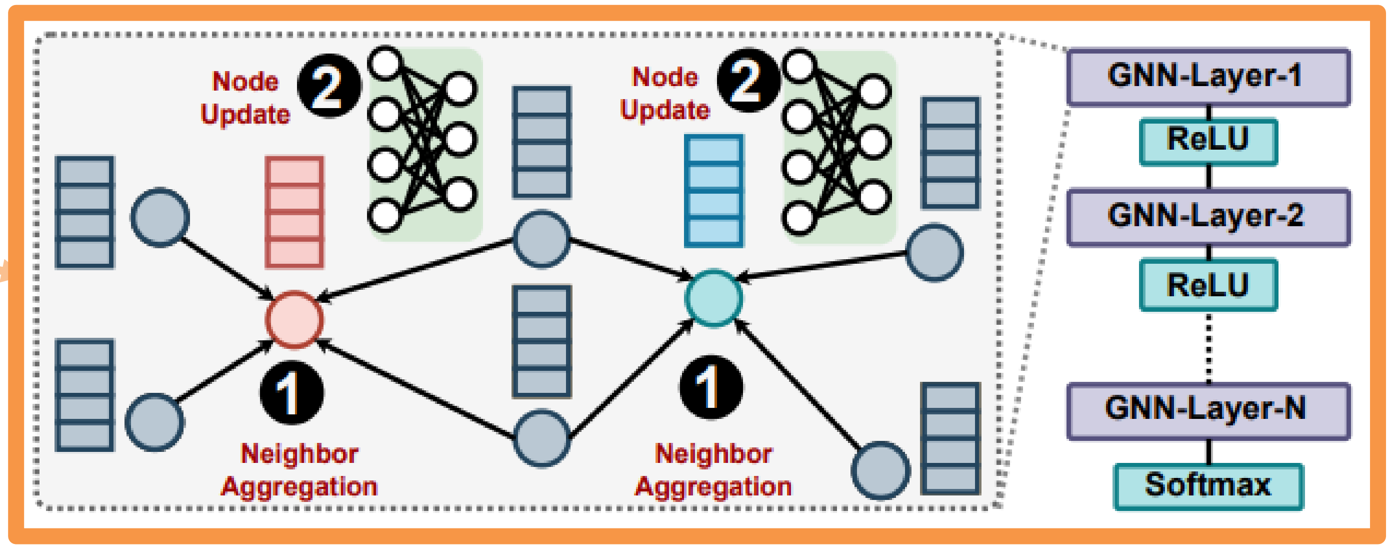
- **Graph Neural Networks (GNNs):**
  Computing attention weights or edge scores (e.g., in Graph Attention Networks or message passing). The adjacency matrix defines the sparse mask S.
- **Recommendation Systems:**
  Used when only a subset of interactions between users and items are observed.
- **Sparse attention mechanisms:**
  Efficient computation of attention over structured sparse graphs.


Dense matrix B^T (K x N)
K
K
Dense matrix A (M x K)
=
Input sparse matrix S (M x N)
Output sparse matrix P (M x N)

Background

Graph Neural Network Basics.

$$a_v^{(k+1)} = \mathit{Aggregate}^{(k+1)}(h_u^{(k)} | u \in \mathbf{N}(v) \cup h_v^{(k)})$$
$$h_v^{(k+1)} = \mathit{Update}^{(k+1)}(a_v^{(k+1)})$$



Basic computation in GNNs.

- Neighbor aggregation (SpMM-like).

$$\hat{\mathbf{X}} = (\mathbf{F}_{N\times N} \odot \mathbf{A}_{N\times N}) \cdot X_{N\times D})$$

- Edge feature computation (SDDMM-like).

$$\mathbf{F} = (\mathbf{X}_{N\times D} \cdot \mathbf{X}_{N\times D}^T) \odot \mathbf{A}_{N\times N}$$

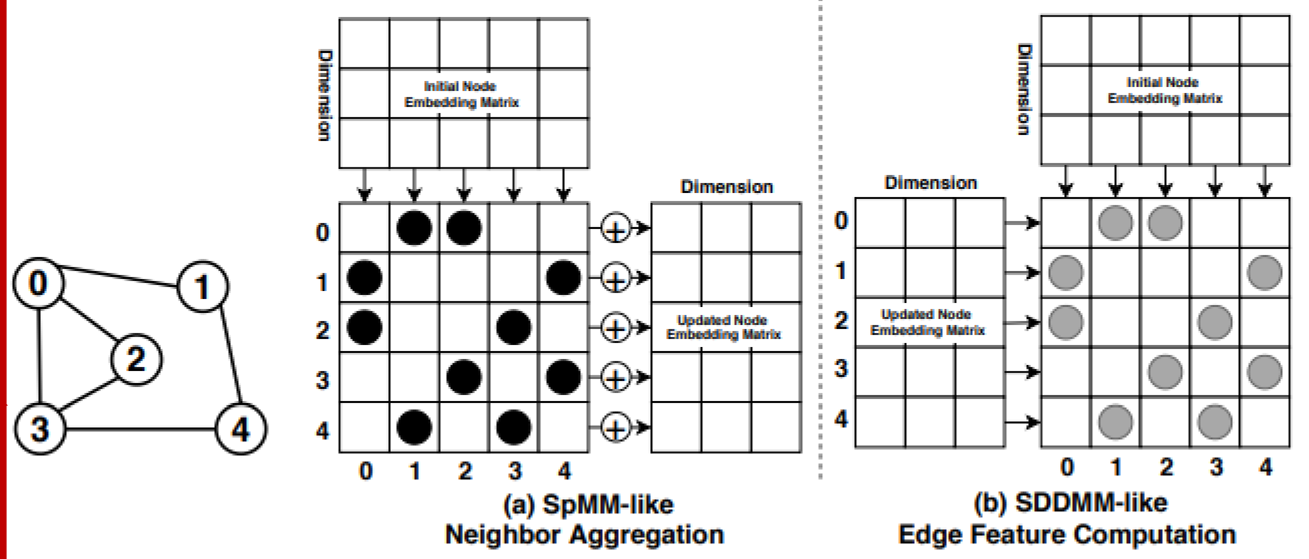

Dimension
Initial Node Embedding Matrix
Dimension
Updated Node Embedding Matrix
(a) SpMM-like Neighbor Aggregation
Dimension
Initial Node Embedding Matrix
Dimension
Updated Node Embedding Matrix
(b) SDDMM-like Edge Feature Computation

Figure 2. SpMM-like and SDDMM-like Operation in GNNs. Note that "→" indicates loading data; "⊕" indicates neighbor embedding accumulation.

Challenges

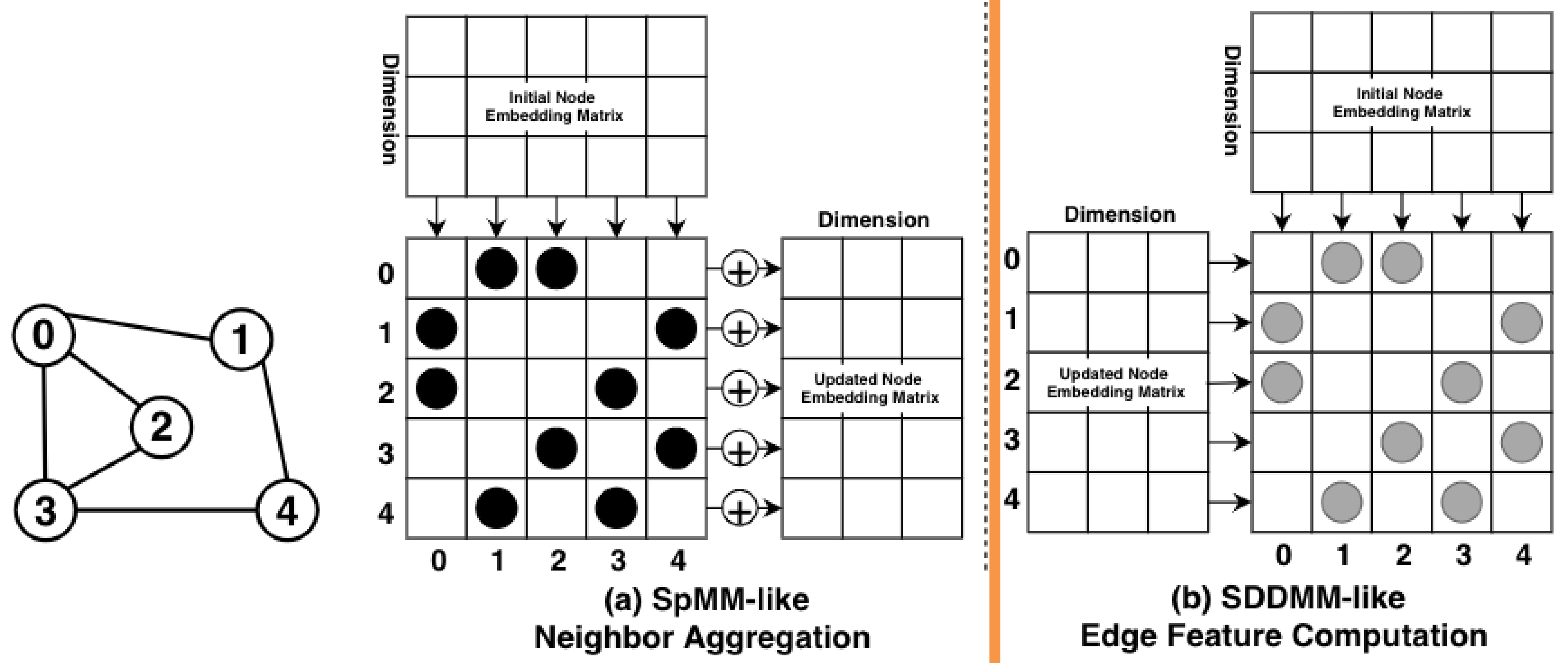
- Existing **deep-learning frameworks** are optimized for dense neural network operations.
- Existing major **sparse computation kernels** (e.g., cuSPARSE) leverage CUDA cores.
- Existing **Tensor-Core based kernels** (e.g., Block-SpMM) rely on rigid input sparsity pattern (e.g., block sparsity).

Lack of efficient support for sparse graph neural network computation.

Underutilize the latest GPU with new hardware feature that can offer high-performance computation.

Limits its applicability towards different sparse inputs settings.

Motivations



Dimension
Initial Node Embedding Matrix
Updated Node Embedding Matrix
(a) SpMM-like Neighbor Aggregation
(b) SDDMM-like Edge Feature Computation

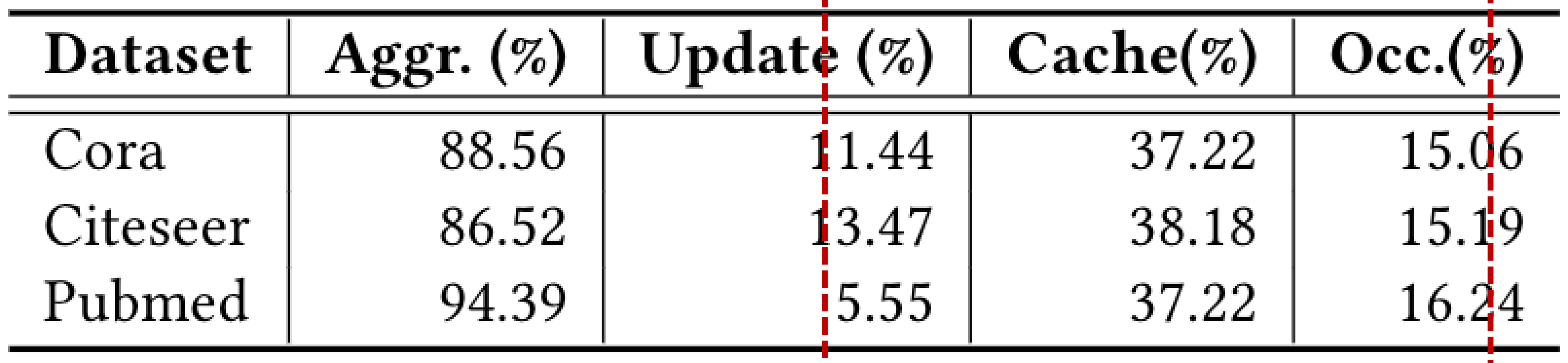
Figure 2. SpMM-like and SDDMM-like Operation in GNNs. Note that "→" indicates loading data; "⊕" indicates neighbor embedding accumulation.

Apply separate optimization on one direction only would hardly work

GPUs fit GNNs

Sparse MM on CUDA core

le 1. Profiling of GCN Sparse Operations.

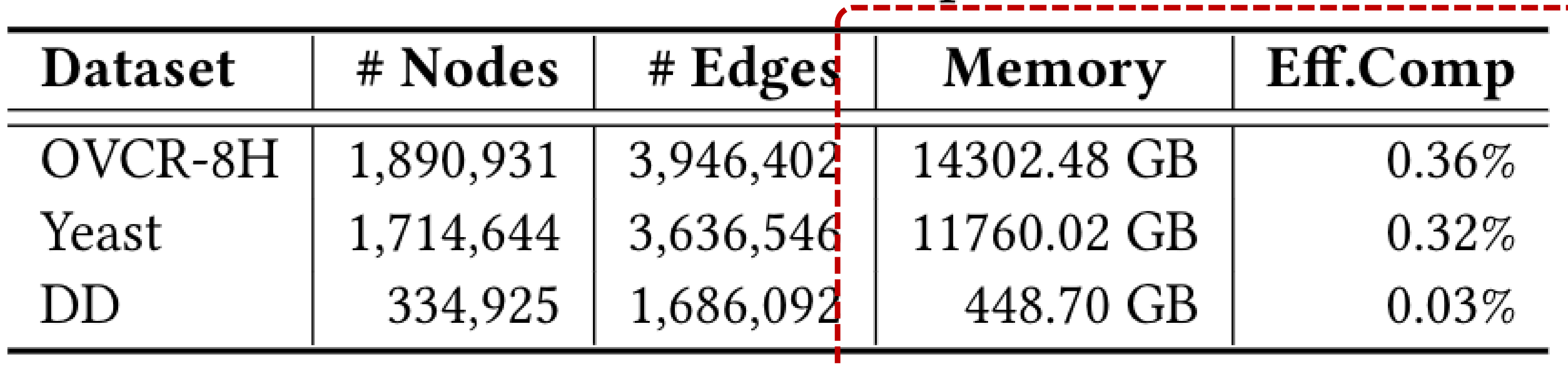


Dataset	Aggr. (%)	Update (%)	Cache(%)	Occ.(%)
Cora	88.56	11.44	37.22	15.06
Citeseer	86.52	13.47	38.18	15.19
Pubmed	94.39	5.55	37.22	16.24

GNNs fit GPUs

Dense MM on CUDA core

Medium-size Graphs in GNNs.



Dataset	# Nodes	# Edges	Memory	Eff.Comp
OVCR-8H	1,890,931	3,946,402	14302.48 GB	0.36%
Yeast	1,714,644	3,636,546	11760.02 GB	0.32%
DD	334,925	1,686,092	448.70 GB	0.03%

Question:

How could we match the sparse GNN workload with GPUs to achieve high computation efficiency and better utilization of GPU resources?

TC-GNN Overview

"Let the input sparse graph fit the dense computation of Tensor Core"

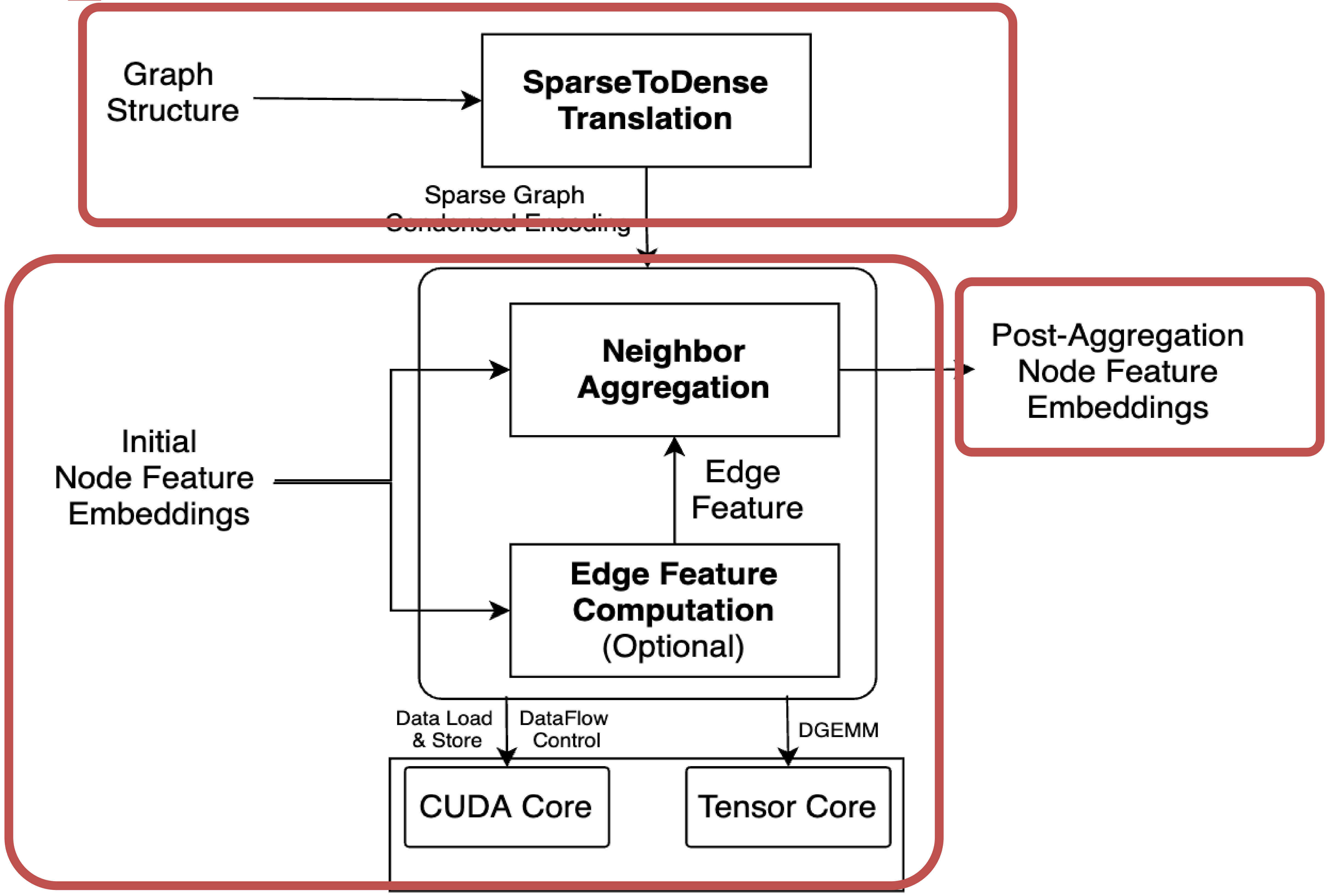
- The first TC-based GNN acceleration design on GPUs.
- At the input level technique.
- At the kernel level innovation.
- At the framework level design.

Sparse graph translation (SGT) technique condense non-zero elements from sparse tiles into a fewer number of "dense" tiles

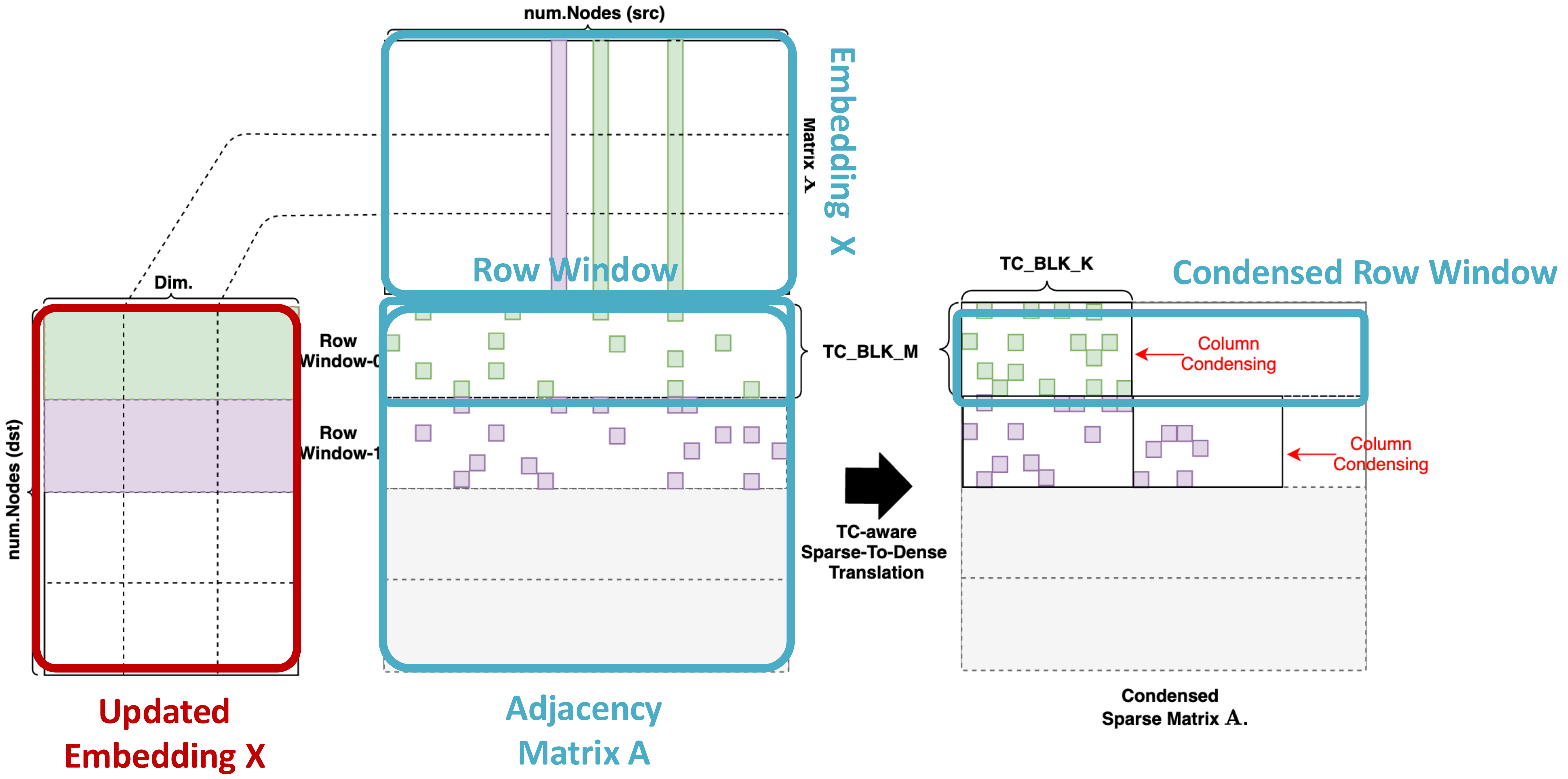
TC-GNN exploits the benefits of CUDA core and tensor core collaboration.

TC-GNN integrates with the popular Pytorch framework.

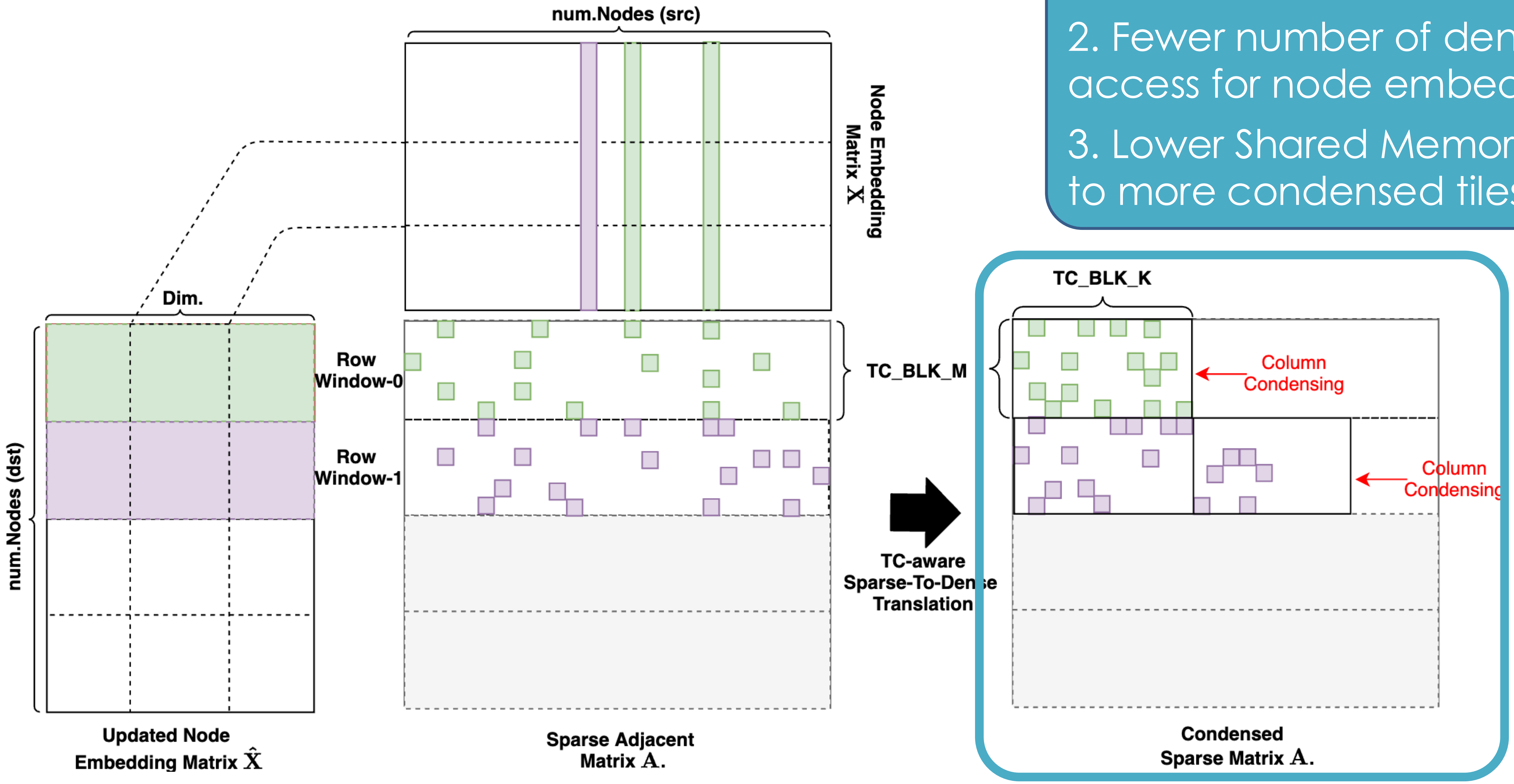
Overall Design


Graph Structure
SparseToDense Translation
Sparse Graph Condensed Encoding
Initial Node Feature Embeddings
Neighbor Aggregation
Edge Feature
Edge Feature Computation (Optional)
Post-Aggregation Node Feature Embeddings
Data Load & Store
DataFlow Control
DGEMM
CUDA Core
Tensor Core

Sparse Graph Translation



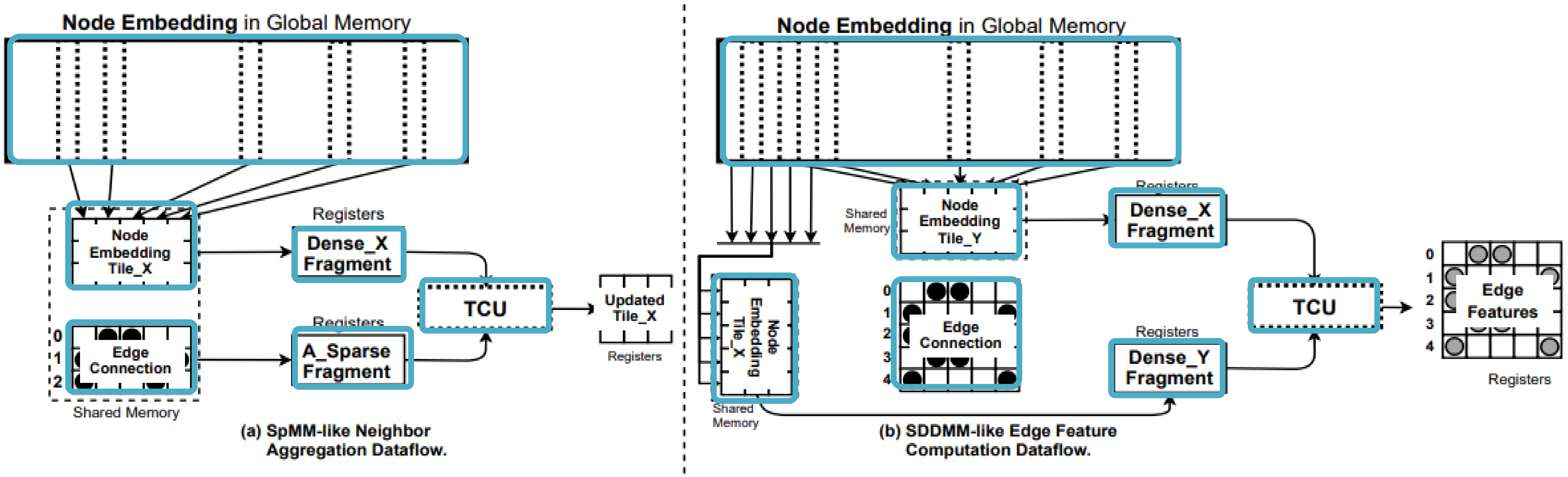
Sparse Graph Translation



num.Nodes (src)
Node Embedding Matrix $\mathbf{X}$
Dim.
num.Nodes (dst)
Row Window-0
Row Window-1
TC_BLK_M
TC-aware Sparse-To-Dense Translation
TC_BLK_K
Column Condensing
Column Condensing
Updated Node Embedding Matrix $\hat{\mathbf{X}}$
Sparse Adjacent Matrix $\mathbf{A}$.
Condensed Sparse Matrix $\mathbf{A}$.

1. Fewer number of iterations for Calling TC WMMA primitives.
2. Fewer number of dense row access for node embedding vector.
3. Lower Shared Memory Usage due to more condensed tiles loading.

TC-optimized Dataflow

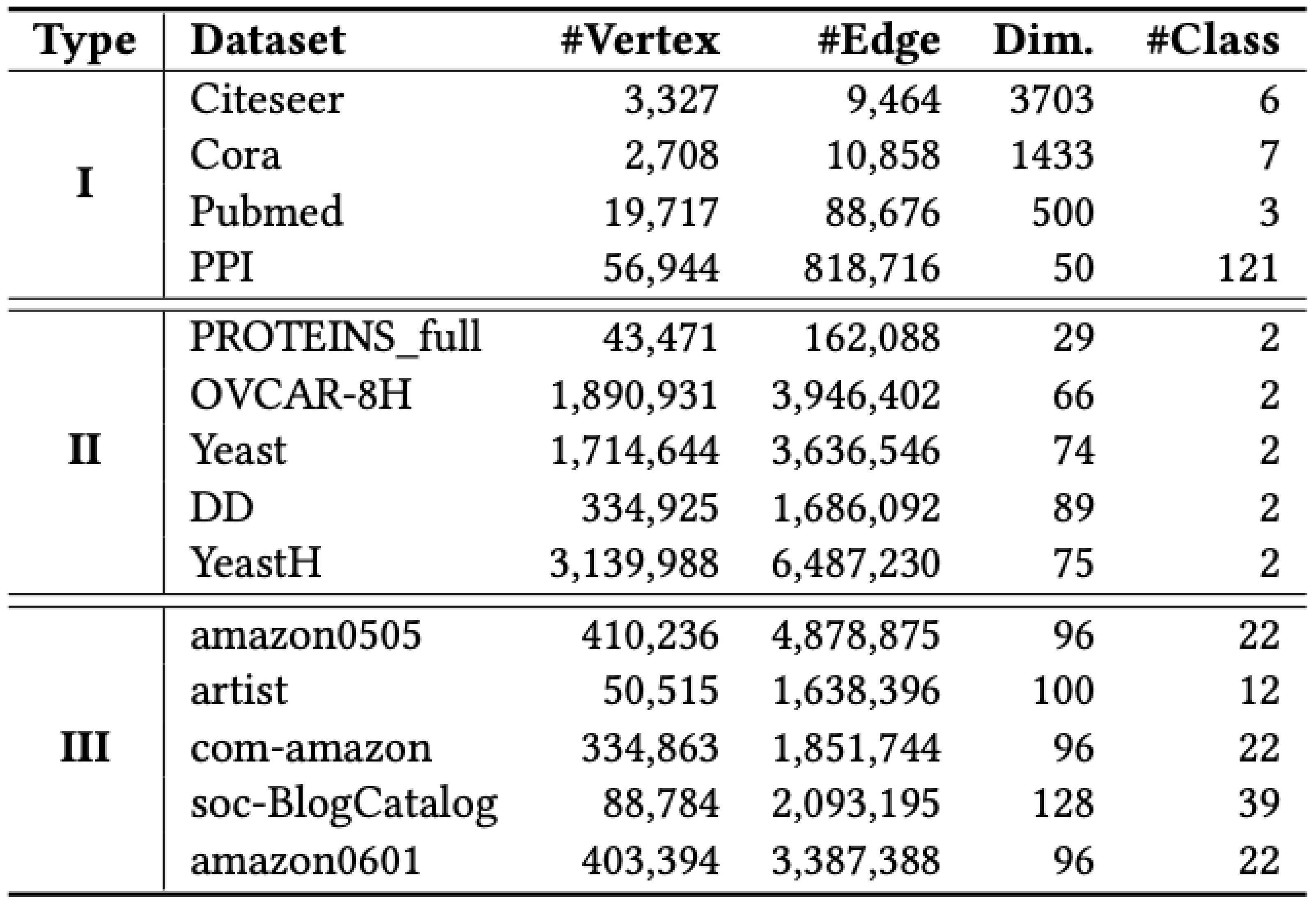


TC-Optimized Dataflow Design for (a) Neighbor Aggregation and (b) Edge Feature Computing in GNNs

Evaluation

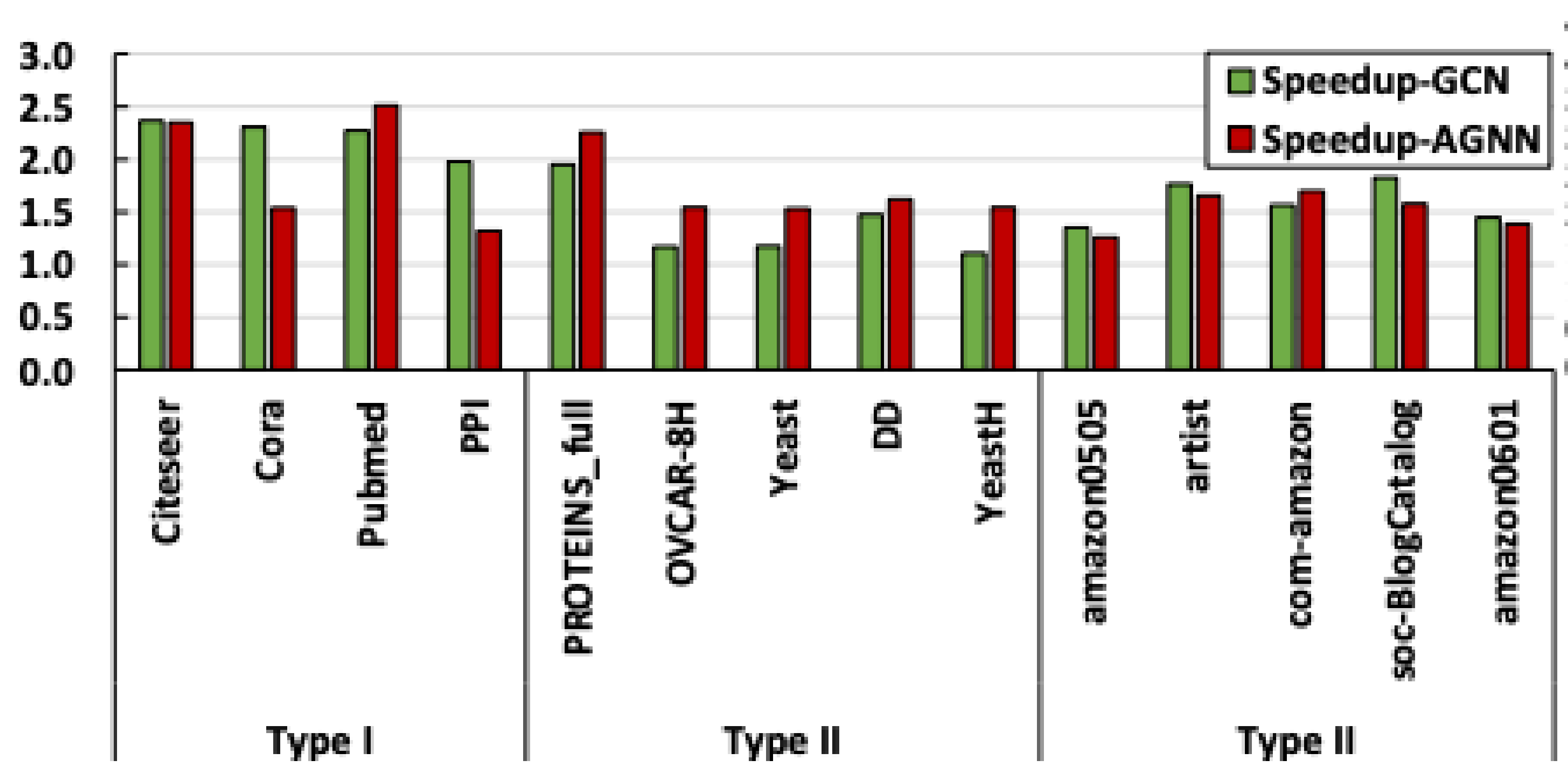
- Baseline:
  - Deep Graph Library (DGL)
  - PyTorch Geometric (PyG)
- GNN model:
  - GCN (Graph Convolutional Network)
  - AGNN (Attention-based GNN)
- Platform:
  - A desktop server with 8-core 16-thread Intel Xeon Silver 4110 CPU (64GB host memory) and NVIDIA RTX3090 GPU (24GB device memory)

Table 4. Datasets for Evaluation.

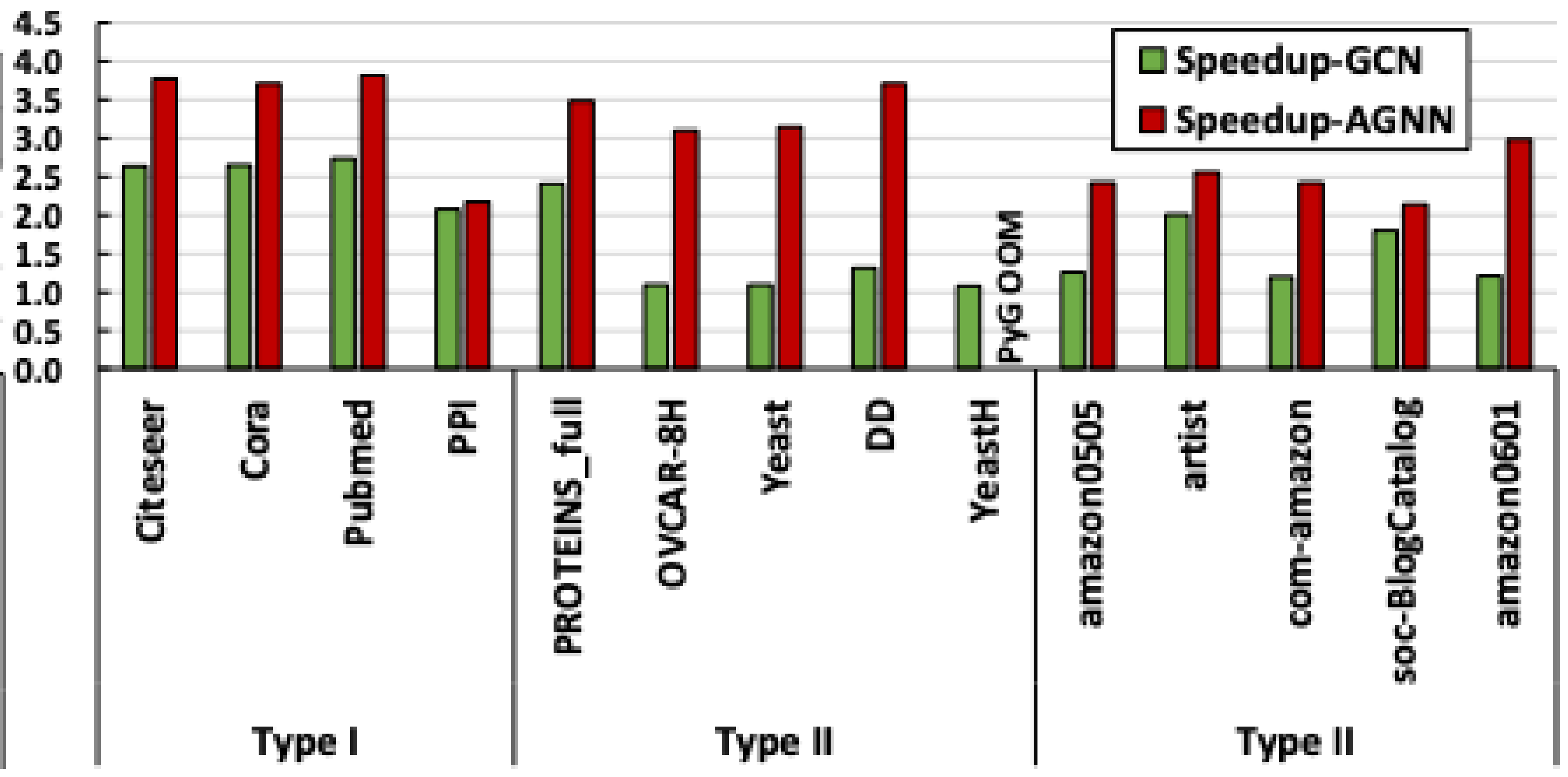


Type	Dataset	#Vertex	#Edge	Dim.	#Class
I	Citeseer	3,327	9,464	3703	6
	Cora	2,708	10,858	1433	7
	Pubmed	19,717	88,676	500	3
	PPI	56,944	818,716	50	121
II	PROTEINS_full	43,471	162,088	29	2
	OVCAR-8H	1,890,931	3,946,402	66	2
	Yeast	1,714,644	3,636,546	74	2
	DD	334,925	1,686,092	89	2
	YeastH	3,139,988	6,487,230	75	2
III	amazon0505	410,236	4,878,875	96	22
	artist	50,515	1,638,396	100	12
	com-amazon	334,863	1,851,744	96	22
	soc-BlogCatalog	88,784	2,093,195	128	39
	amazon0601	403,394	3,387,388	96	22

End-to-end Performance: DGL & PyG



(a)



(b)

Speedup over (a) DGL and (b) PyG on GCN and AGNN.

Avg: 1.70X

Operator Performance (dgl.op)

- SpMM (**dgl.op.copy_u_sum)**

	dgl.op (ms)	TC-GNN (ms)
PROTEINS_full	0.088	0.044
OVCAR-8H	1.295	1.018
Yeast	1.183	0.862
DD	0.454	0.287
SW-620H	1.291	1.018

Avg: 1.50X

- SDDMM (**dgl.op.u_dot_v)**

	dgl.op (ms)	TC-GNN (ms)
PROTEINS_full	0.062	0.019
OVCAR-8H	0.466	0.054
Yeast	0.401	0.051
DD	0.170	0.026
SW-620H	0.476	0.055

Avg: 6.98X

Kernel Performance (cuSPARSE)

- SpMM w.r.t cuSPARSE with different embedding dimension. (GFLOPS)

	D (16)		D (32)		D (64)	
	cuSPARSE	TC-GNN	cuSPARSE	TC-GNN	cuSPARSE	TC-GNN
PROTEINS_full	90.13	130.89	170.55	226.63	276.46	348.73
OVCAR-8H	135.26	143.54	237.81	239.05	237.96	340.02
Yeast	135.42	157.97	238.12	261.76	230.25	366.61
DD	156.17	207.04	309.67	350.57	467.94	498.02
SW-620H	135.22	143.56	237.72	239.17	238.13	340.04

Avg: 1.23X

RICE UNIVERSITY

yuke.wang@rice.edu