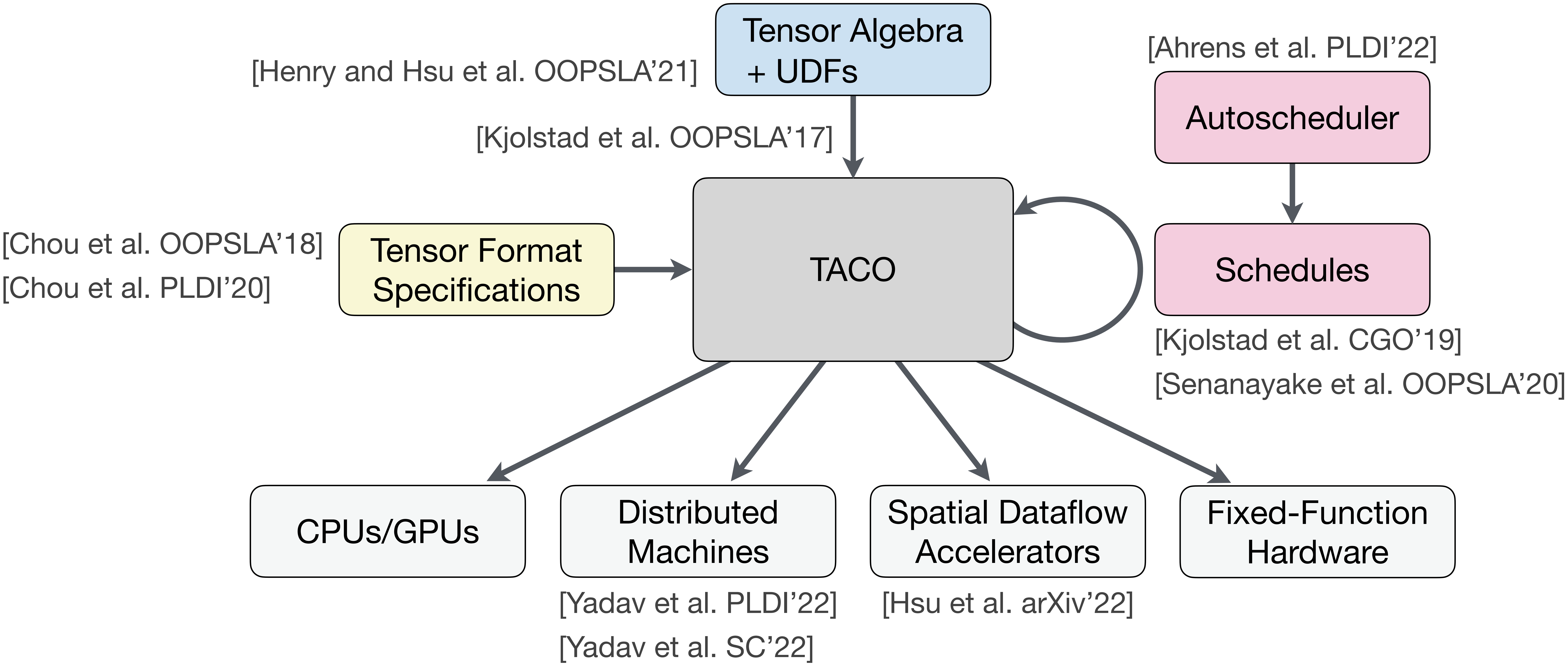
Software and Hardware for Sparse ML

Fredrik Kjolstad
Assistant Professor, Department of Computer Science
Stanford University

Background: Sparse Tensor Algebra Compilation



Tensor Algebra + UDFs

[Henry and Hsu et al. OOPSLA'21]

[Kjolstad et al. OOPSLA'17]

[Ahrens et al. PLDI'22]

Autoscheduler

Schedules

[Kjolstad et al. CGO'19]

[Senanayake et al. OOPSLA'20]

[Chou et al. OOPSLA'18]

[Chou et al. PLDI'20]

Tensor Format Specifications

TACO

CPUs/GPUs

Distributed Machines

[Yadav et al. PLDI'22]

[Yadav et al. SC'22]

Spatial Dataflow Accelerators

[Hsu et al. arXiv'22]

Fixed-Function Hardware

Other systems:

COMET [Mutlu et al. LCPC'20]

MLIR SparseTensor Dialect [Bik et al. TACO'22]

SPF [Zhao et al. arXiv'22]

SparseTIR [Ye et al. arXiv'22]

Overview

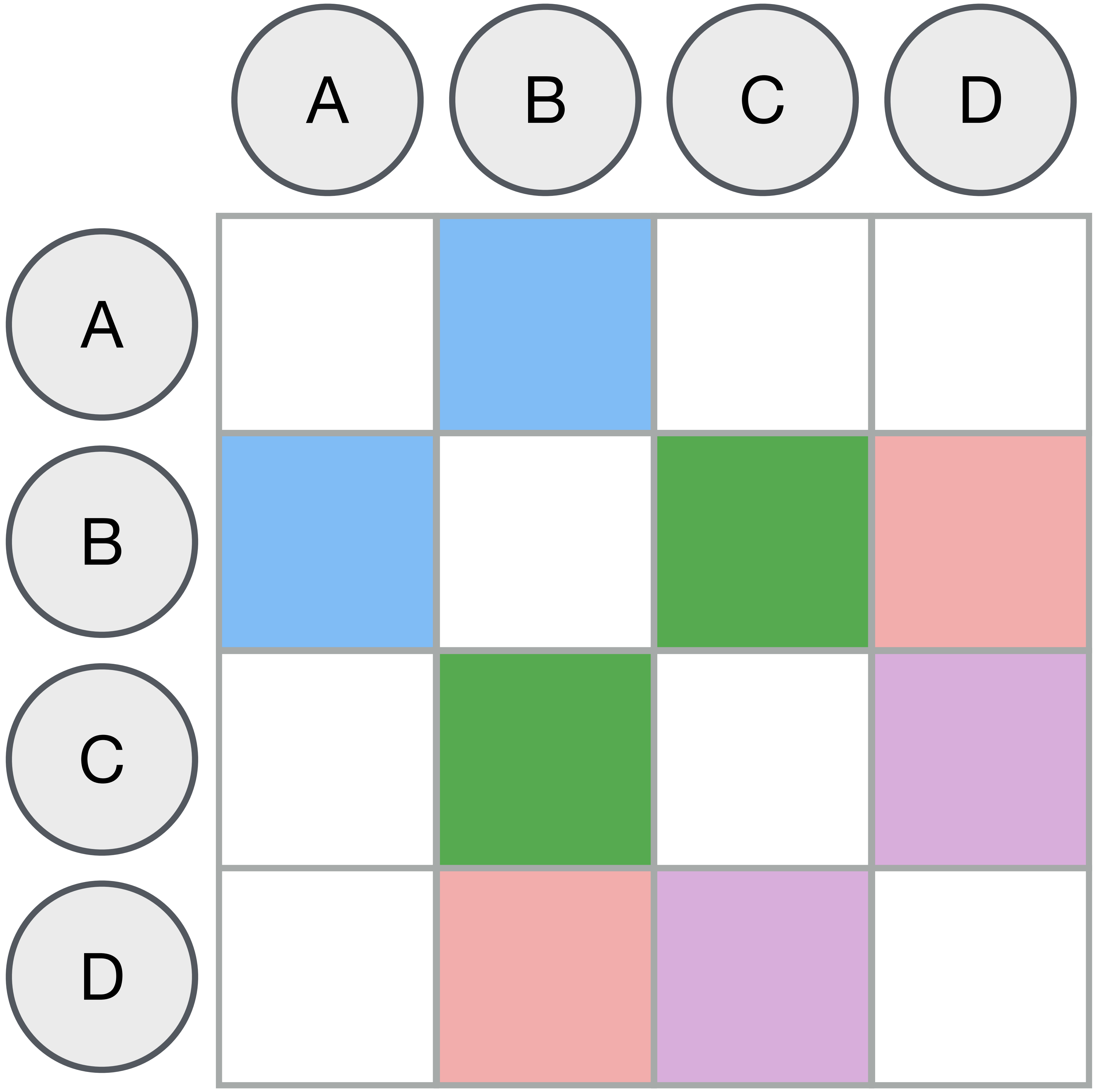
My view of sparsity

Why sparsity requires compilers and general hardware

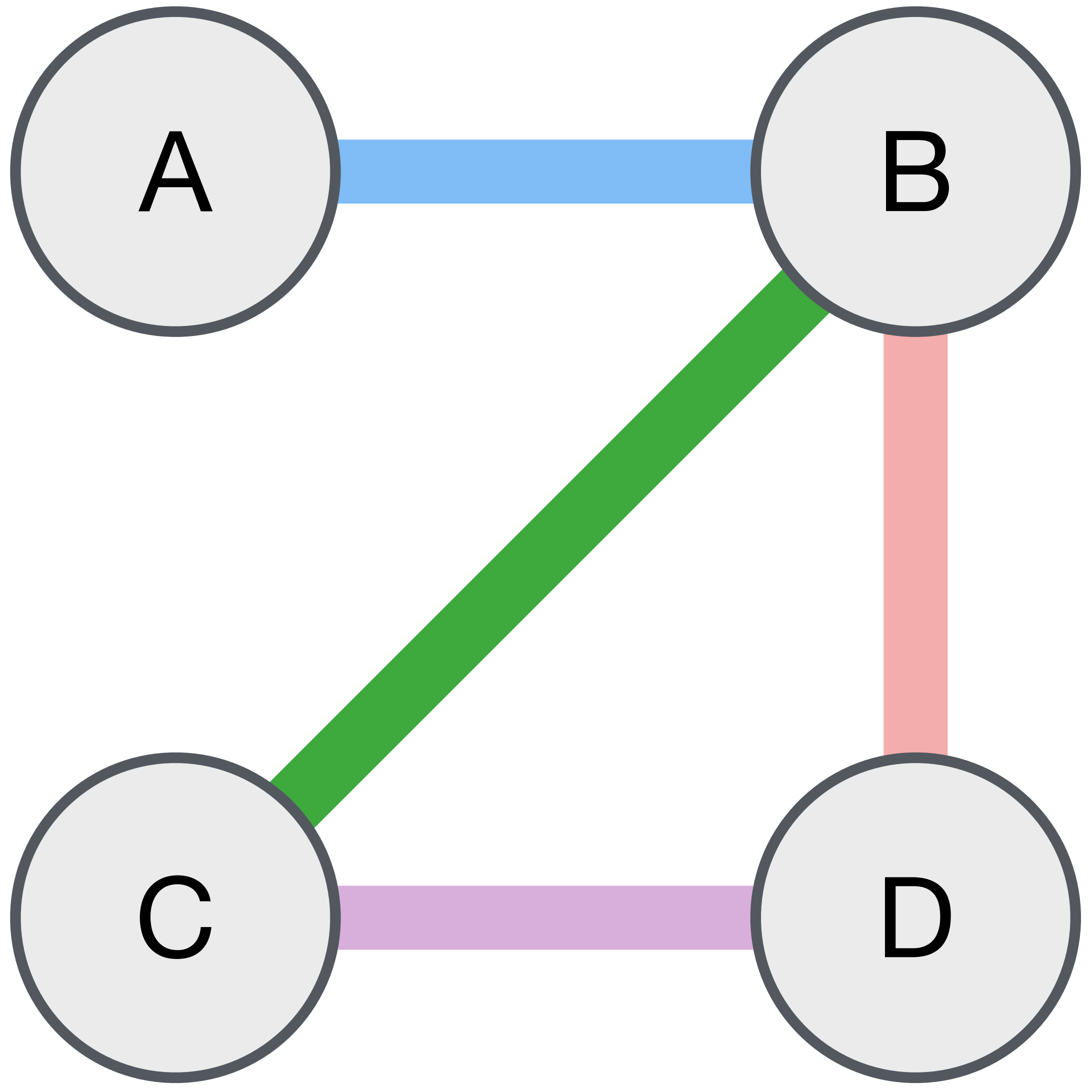
Thesis

Unlike dense neural networks that can be reduced to GEMM,
it will not be possible to reduce sparse neural networks
to one optimized function

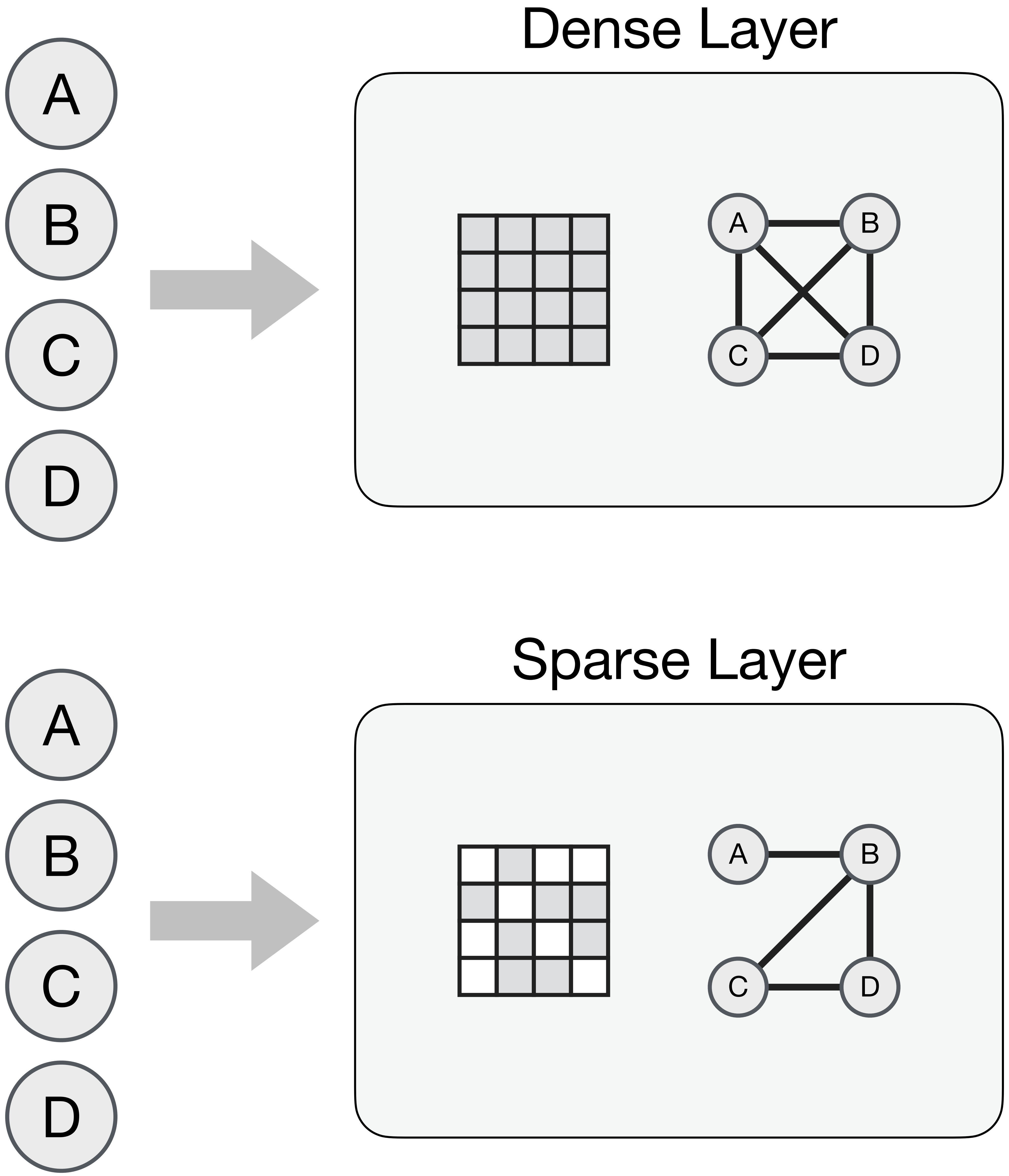
Sparsity as system connectivity



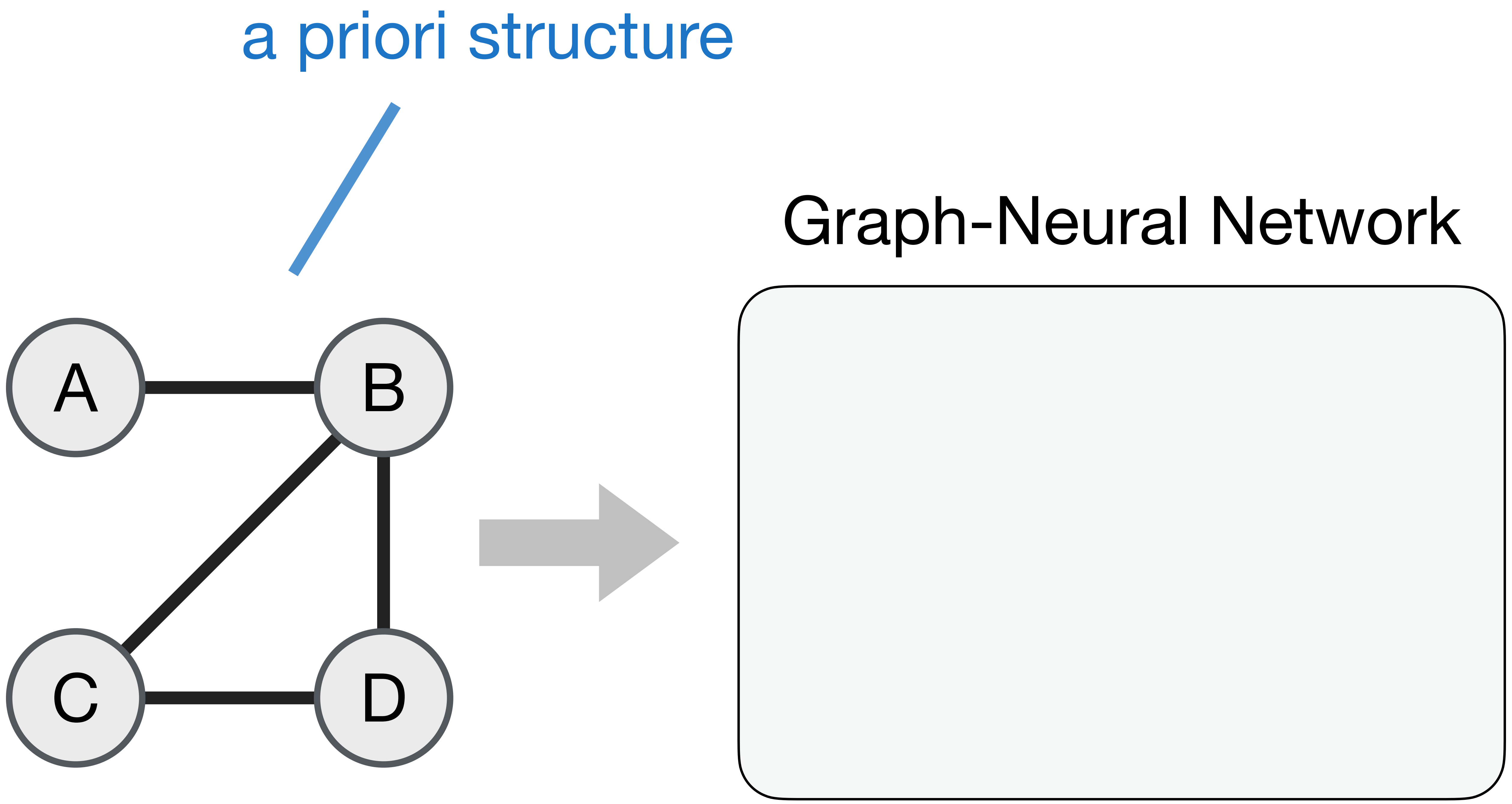
	A	B	C	D
A		🟦		
B	🟦		🟩	🟥
C		🟩		🟪
D		🟥	🟪	



Two types of sparsity: learned and a priory


Dense Layer

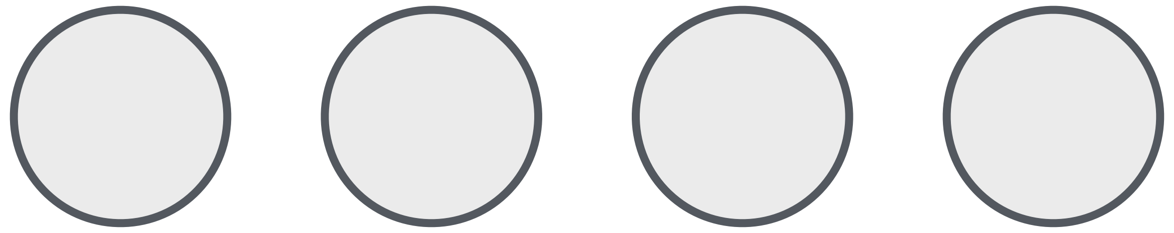
Sparse Layer


a priori structure

Graph-Neural Network

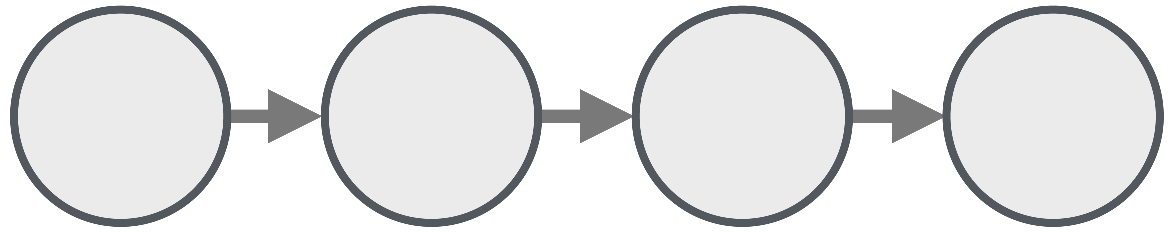
Structure of input data

Sets



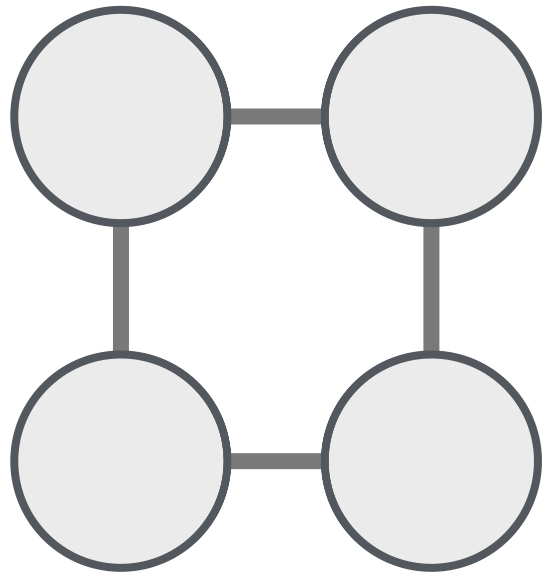
- No a priori connections
- Postulate fully connected layer
- Then may try to learn sparsity

Sequences



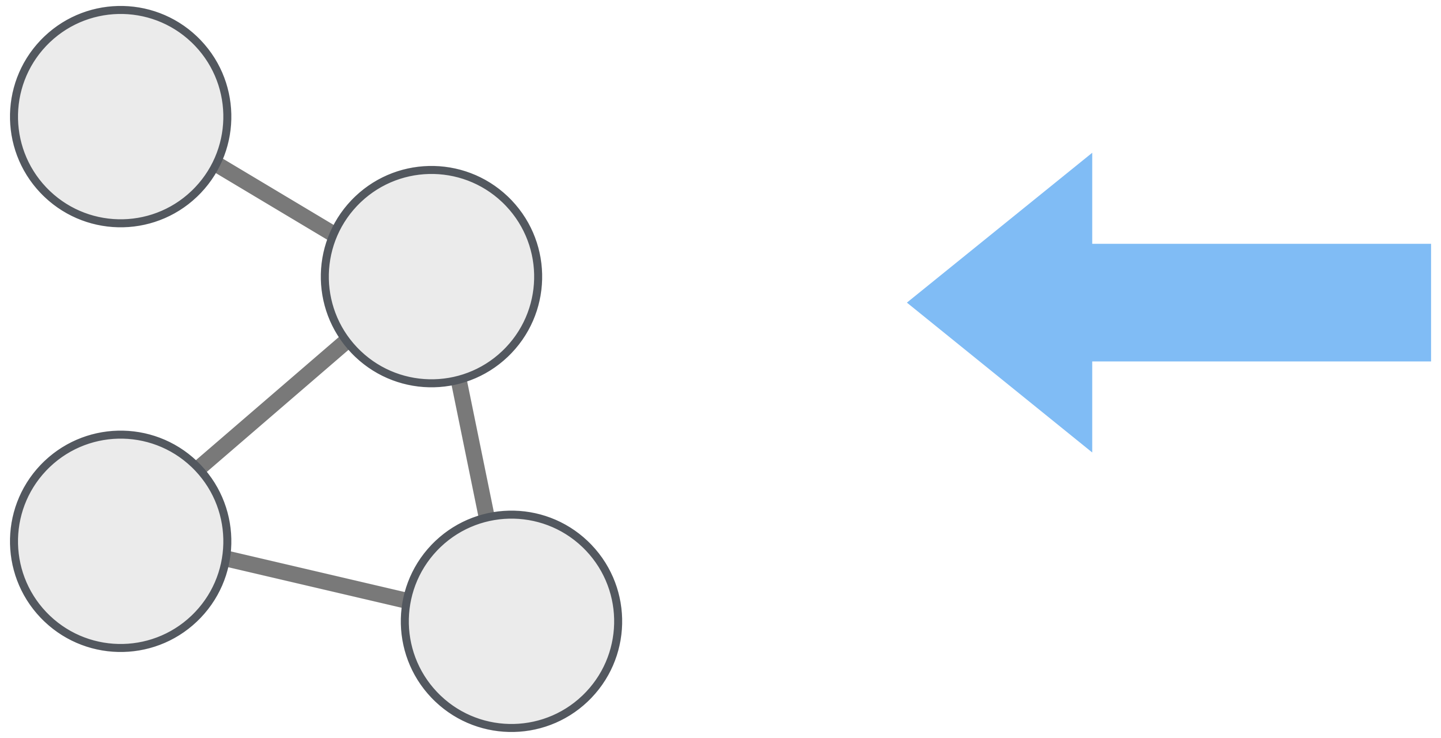
- Triangular matrices in transformers
- Recurrences in RNNs

Grids



- Pixel locality in CNNs

Graphs

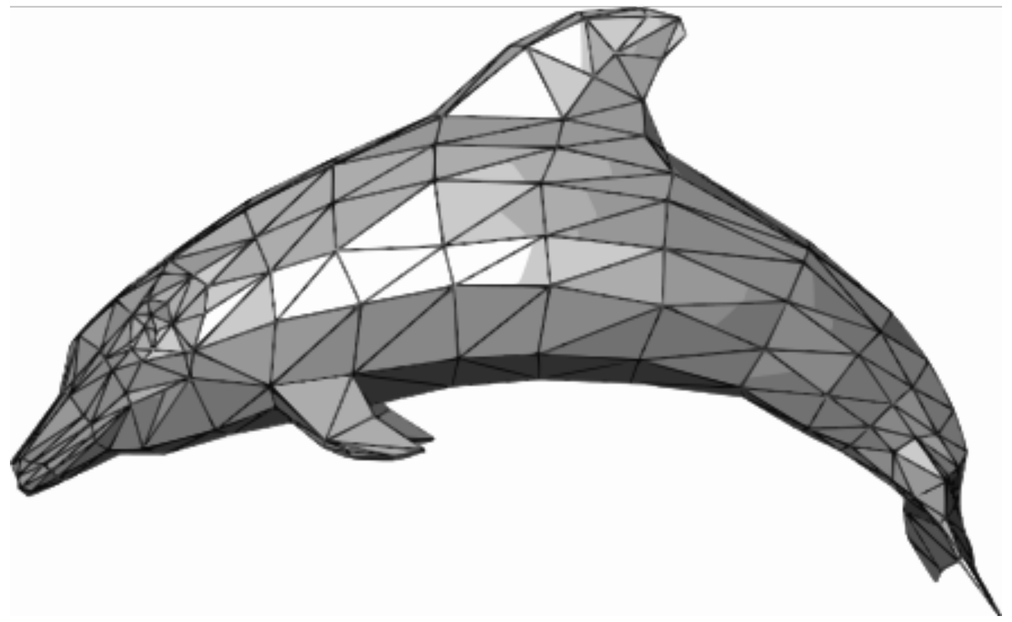


Relational Data

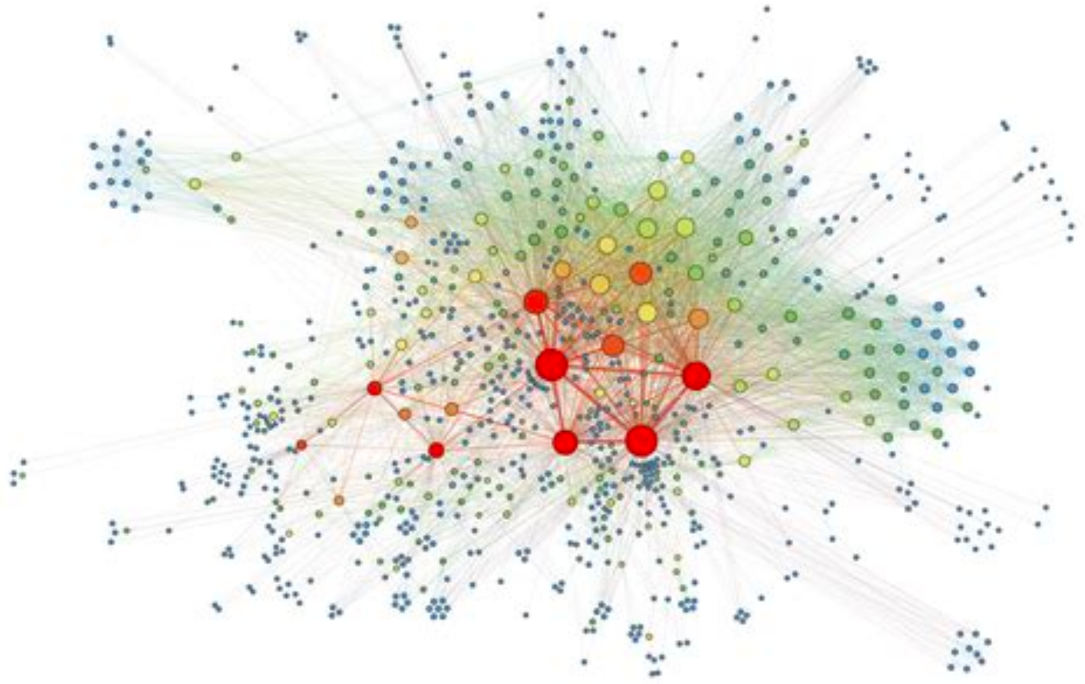
John	Jill
Jill	Kim
Kim	Mary
Mary	Kim

How sparse is graph/relational data? Often asymptotically sparse.

Conditioned Meshes



Power-law graphs



Assume an average degree of 150 (e.g., 150 friends)

Each matrix row then has 150 nonzeros

At 10,000 rows: $\frac{150 \cdot 10{,}000}{10{,}000^2} = 1.5\,\%$ nonzeros

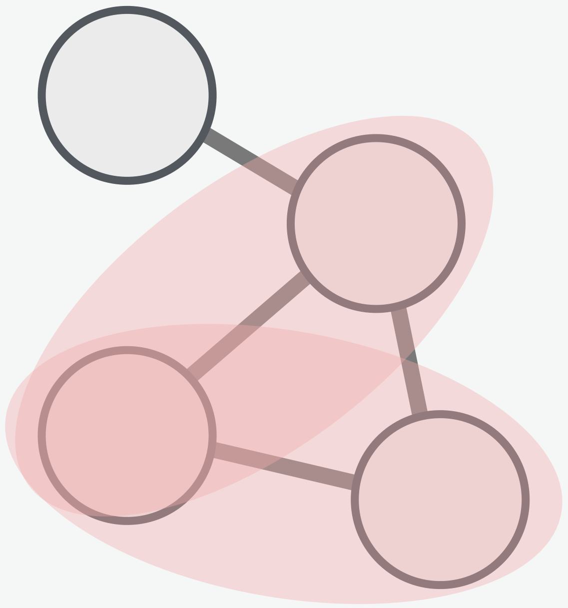
At 100,000 rows: $\frac{150 \cdot 100{,}000}{100{,}000^2} = 0.15\,\%$ nonzeros

Matrix components: $O(n^2)$
Nonzeros: $O(n)$
Fraction of nonzeros: $O(1/n)$

Should we build graph frameworks or sparse tensor frameworks?

Graph Framework

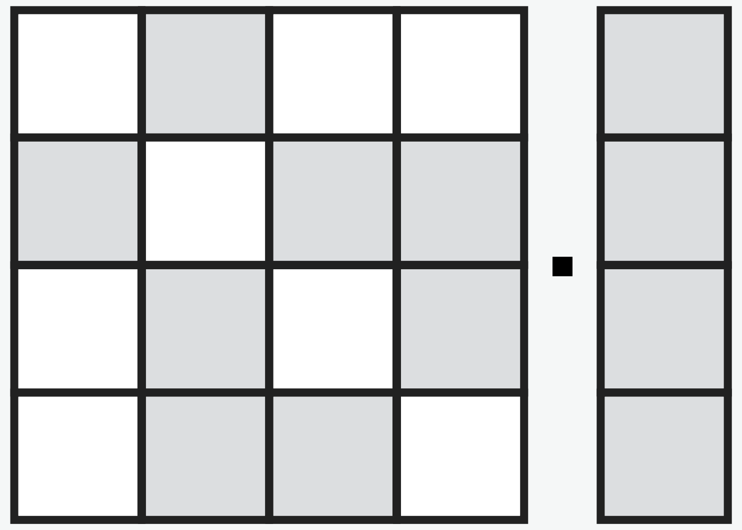
PyG



- Lower level
- Can hand-implement fusion

Sparse Tensor Framework

(Sparse) PyTorch



- Higher-level natural notation
  - Same notation as in papers
  - Composes with dense/CNNs
  - Easy to compose multiple graphs
- Compiler can
  - Fuse computation
  - Reorder and tile
  - Port across machines

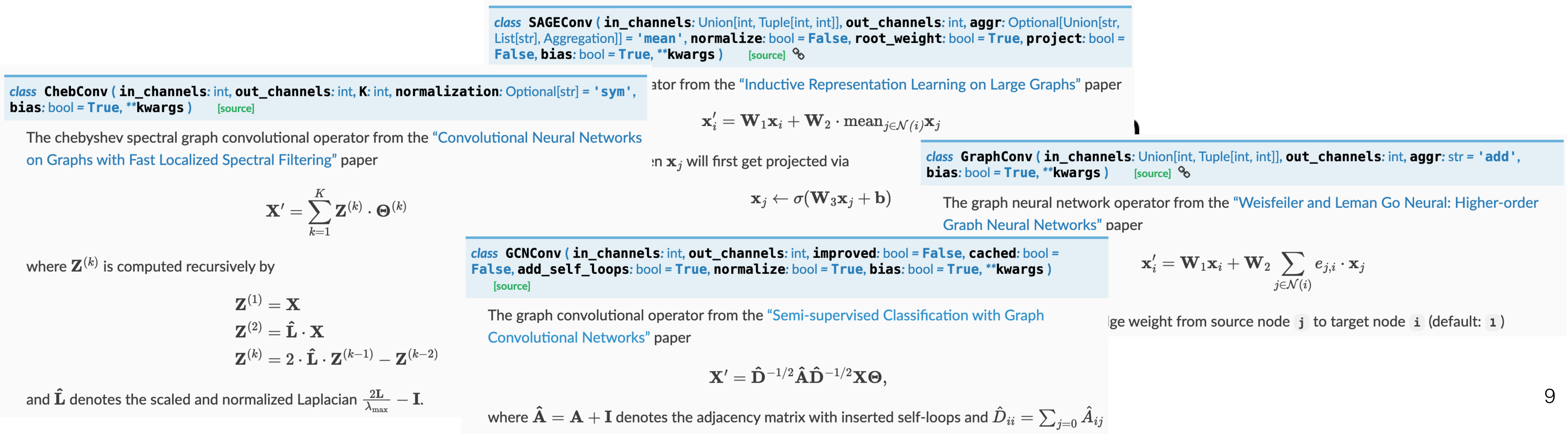
How general should tensor frameworks be?

Kernel library

- Fixed number of hand-optimized operations
- Fixes tensor formats

Full tensor support

- General tensor algebra
- User-defined functions
- Tensor reshapes and composition
- Portable across tensor formats



class **`SAGEConv`** (**`in_channels`**: Union[int, Tuple[int, int]], **`out_channels`**: int, **`aggr`**: Optional[Union[str, List[str], Aggregation]] = **`'mean'`**, **`normalize`**: bool = **`False`**, **`root_weight`**: bool = **`True`**, **`project`**: bool = **`False`**, **`bias`**: bool = **`True`**, **`**kwargs`**) [source]

ator from the "Inductive Representation Learning on Large Graphs" paper

$$\mathbf{x}'_i = \mathbf{W}_1 \mathbf{x}_i + \mathbf{W}_2 \cdot \mathrm{mean}_{j \in \mathcal{N}(i)} \mathbf{x}_j$$

n $\mathbf{x}_j$ will first get projected via

$$\mathbf{x}_j \leftarrow \sigma(\mathbf{W}_3 \mathbf{x}_j + \mathbf{b})$$

class **`ChebConv`** (**`in_channels`**: int, **`out_channels`**: int, **`K`**: int, **`normalization`**: Optional[str] = **`'sym'`**, **`bias`**: bool = **`True`**, **`**kwargs`**) [source]

The chebyshev spectral graph convolutional operator from the "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering" paper

$$\mathbf{X}' = \sum_{k=1}^{K} \mathbf{Z}^{(k)} \cdot \mathbf{\Theta}^{(k)}$$

where $\mathbf{Z}^{(k)}$ is computed recursively by

$$\mathbf{Z}^{(1)} = \mathbf{X}$$
$$\mathbf{Z}^{(2)} = \hat{\mathbf{L}} \cdot \mathbf{X}$$
$$\mathbf{Z}^{(k)} = 2 \cdot \hat{\mathbf{L}} \cdot \mathbf{Z}^{(k-1)} - \mathbf{Z}^{(k-2)}$$

and $\hat{\mathbf{L}}$ denotes the scaled and normalized Laplacian $\frac{2\mathbf{L}}{\lambda_{\max}} - \mathbf{I}$.

class **`GraphConv`** (**`in_channels`**: Union[int, Tuple[int, int]], **`out_channels`**: int, **`aggr`**: str = **`'add'`**, **`bias`**: bool = **`True`**, **`**kwargs`**) [source]

The graph neural network operator from the "Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks" paper

$$\mathbf{x}'_i = \mathbf{W}_1 \mathbf{x}_i + \mathbf{W}_2 \sum_{j \in \mathcal{N}(i)} e_{j,i} \cdot \mathbf{x}_j$$

ge weight from source node `j` to target node `i` (default: `1`)

class **`GCNConv`** (**`in_channels`**: int, **`out_channels`**: int, **`improved`**: bool = **`False`**, **`cached`**: bool = **`False`**, **`add_self_loops`**: bool = **`True`**, **`normalize`**: bool = **`True`**, **`bias`**: bool = **`True`**, **`**kwargs`**) [source]

The graph convolutional operator from the "Semi-supervised Classification with Graph Convolutional Networks" paper

$$\mathbf{X}' = \hat{\mathbf{D}}^{-1/2} \hat{\mathbf{A}} \hat{\mathbf{D}}^{-1/2} \mathbf{X} \mathbf{\Theta},$$

where $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ denotes the adjacency matrix with inserted self-loops and $\hat{D}_{ii} = \sum_{j=0} \hat{A}_{ij}$

How to implement a sparse PyTorch in software and hardware

Factorization

$$A = B \odot (CD)$$

factorize

```
Matrix T = gemm(C,D);
Matrix A = spelmul(B,T);
```

Kernel Library

`gemm` `spelmul` `ttv` `spmv` `mttkrp` `ttm` `matadd` …

Compilation

$$A = B \odot (CD)$$

compile

```
int pA2 = 0;
for (int pB1 = B1_pos[0];
     pB1 < B1_pos[1]; pB1++) {
  int i = B1_crd[pB1];
  for (int pB2 = B2_pos[pB1];
       pB2 < B2_pos[pB1+1]; pB2++) {
    int j = B2_crd[pB2];
    double t = 0.0;
    for (int k = 0; k < O; k++) {
      int pC2 = i * O + k;
      int pD2 = k * N + j;
      t += C[pC2] * D[pD2];
    }
    A[pA2++] = B[pB2] * t;
  }
}
```

Factorization in **dense** tensor algebra

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

↓

Transposes → GEMM → Transposes → GEMM → …

- Works pretty well for dense (at least on shared memory machines)
- The cost of transpose is modest
- The benefit of handwritten GEMM is large

Unlike dense neural networks that can be reduced to GEMM, it will not be possible to reduce sparse neural networks to one optimized function

Factorization in **sparse** tensor algebra

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}} \overline{P_{ip}}$$

↓

Transposes → SpGEMM → Transposes → SpGEMM → …

Requires sorting and potentially data structure conversion

Compilers better able to produce competitive code

Need to flatten data structures (e.g., COO triplets to pairs)

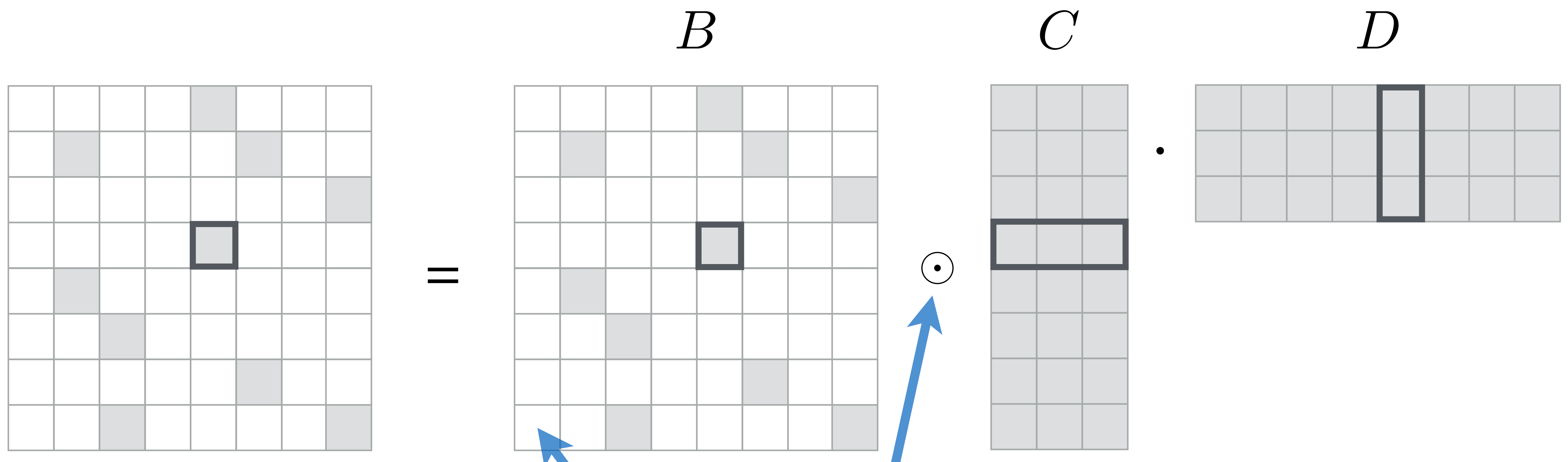
No fusion across operation

↓

Potential asymptotic complexity slowdown!

Factorization destroys fusion

Sampled Dense-Dense Matrix Multiplication (SDDMM)



Element-wise multiplication

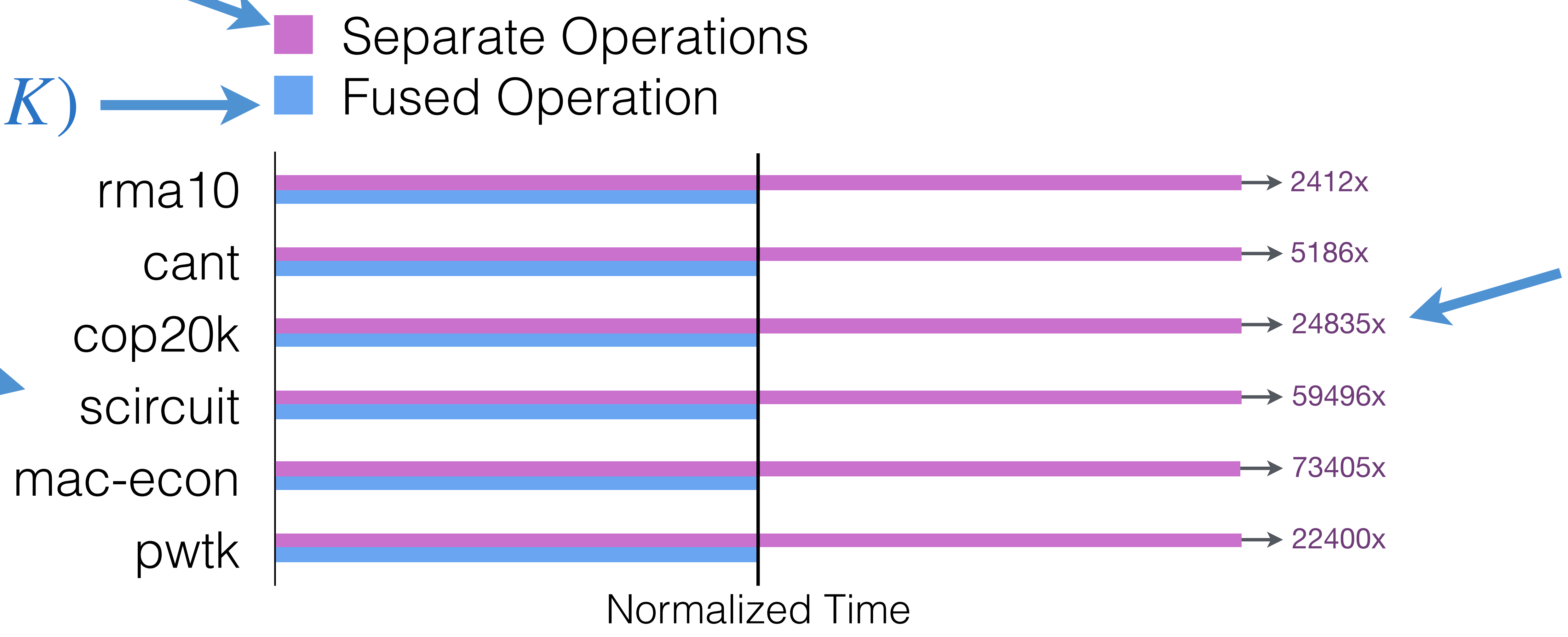
This dot product need not be computed

~~64 inner product~~

10 inner product

$O(IJK)$

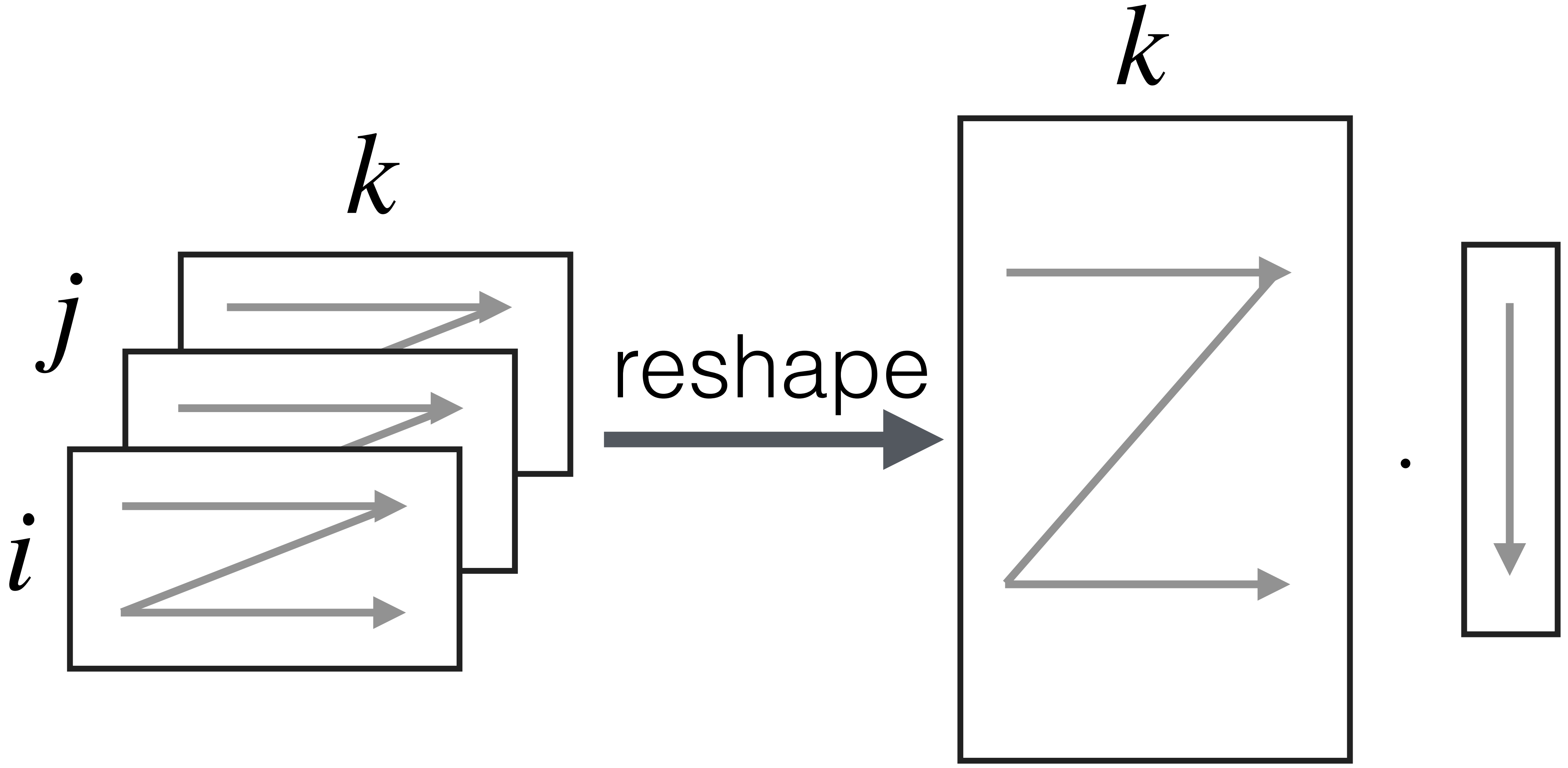
$O(\mathrm{NNZ}_B \cdot K)$

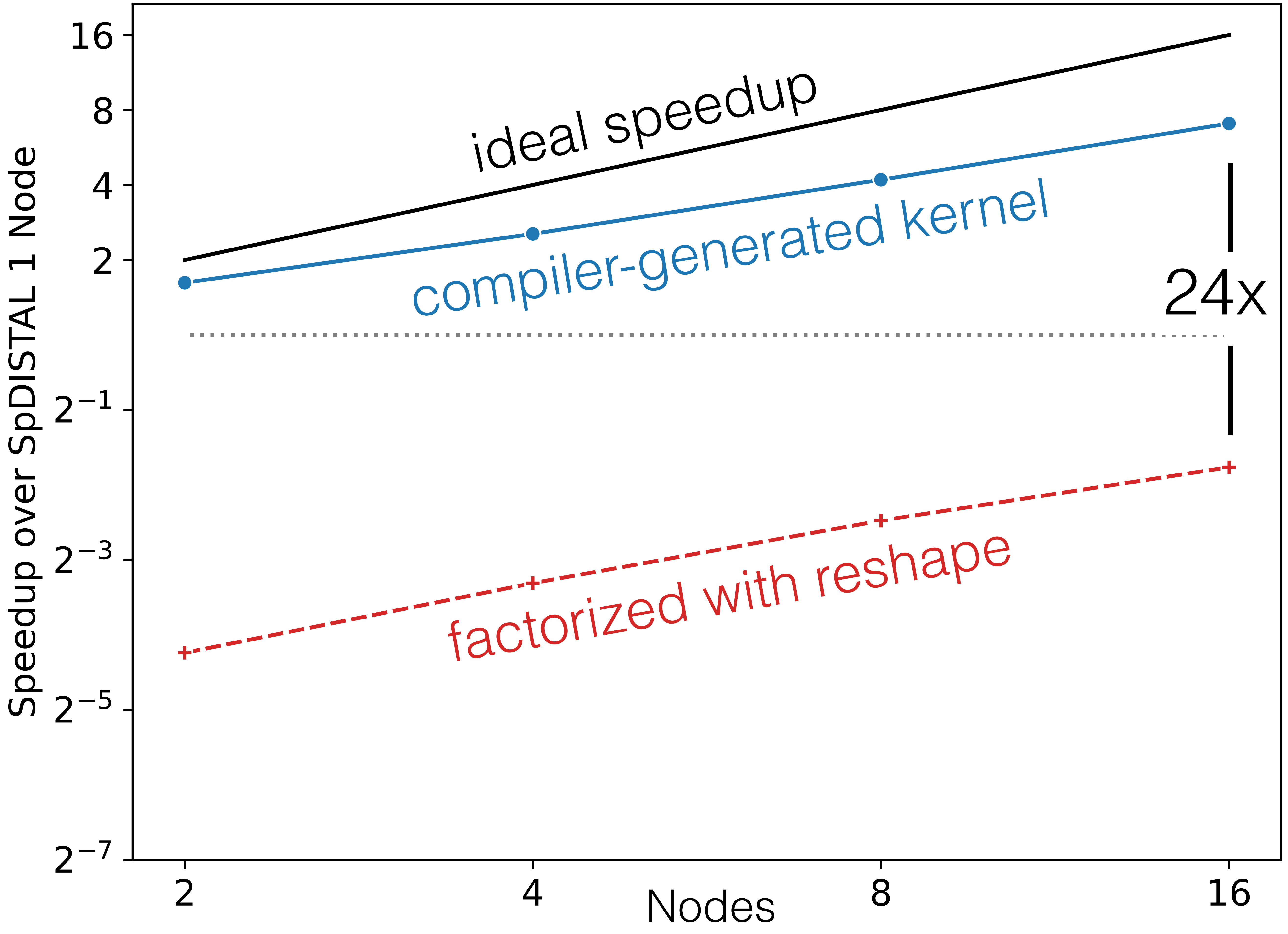


Sparse matrices

Factorization forces data movement

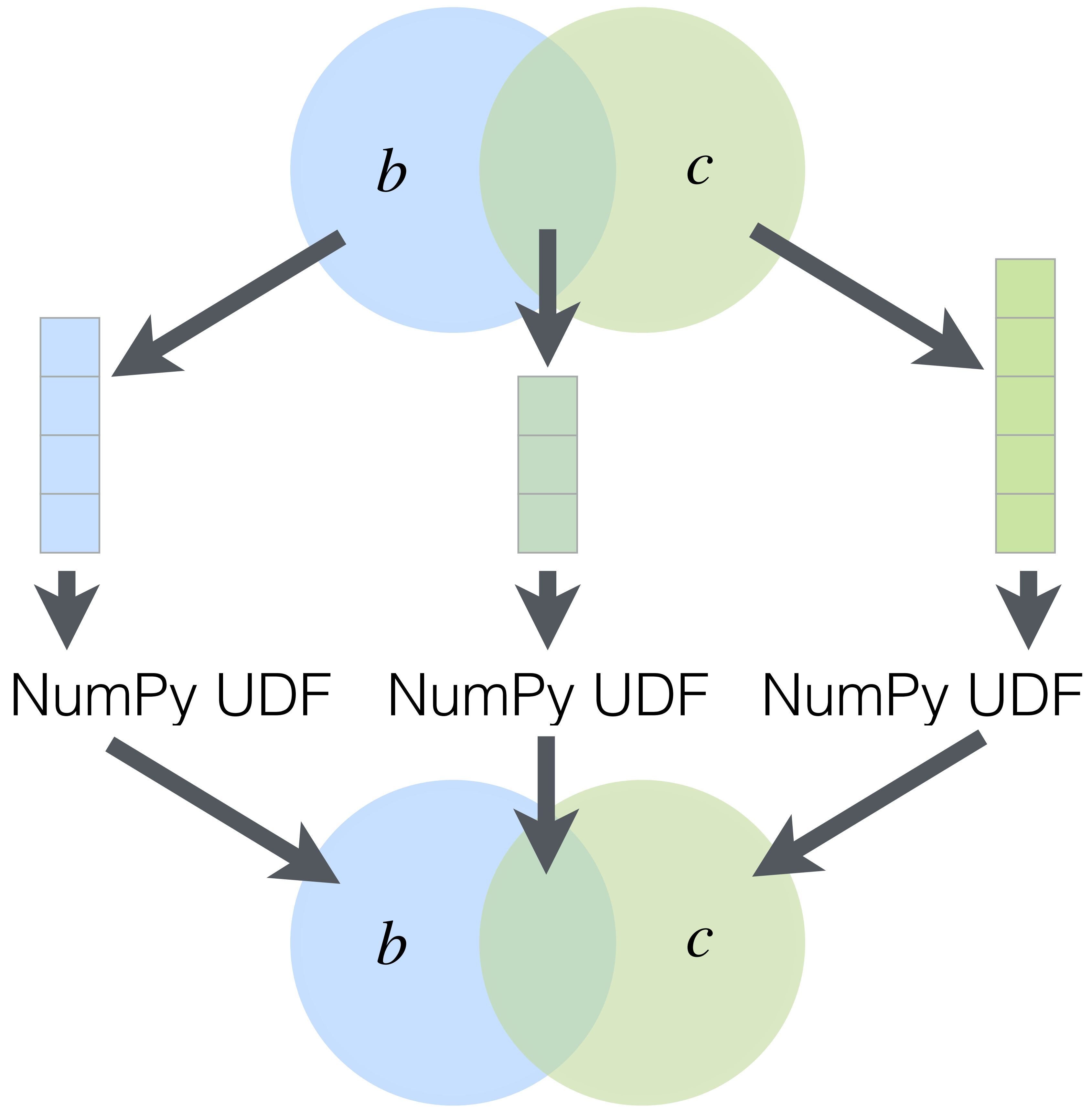
$$A_{ij} = B_{ijk} c_k$$



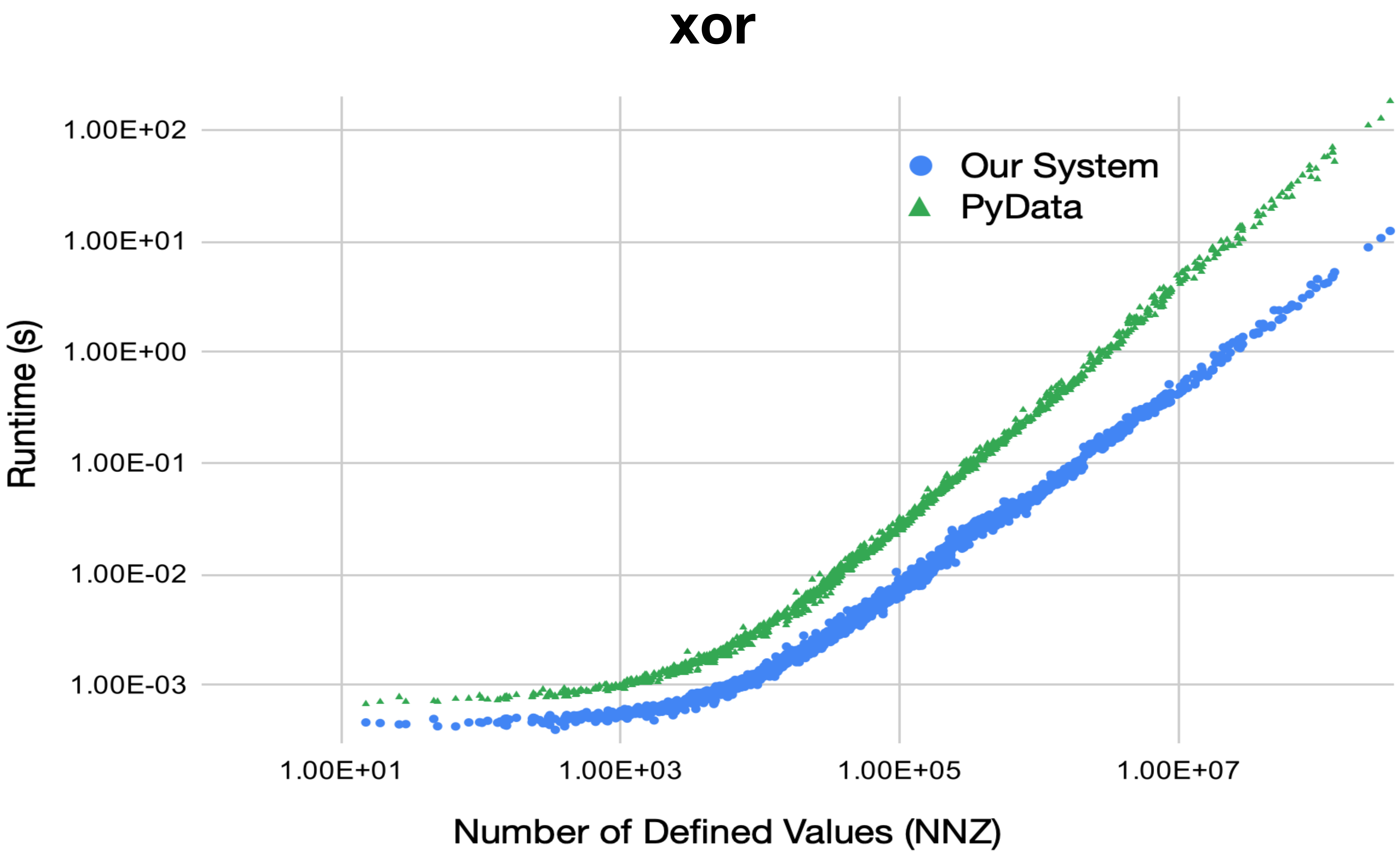


DISTAL [Yadav et al. PLDI'22]
[Yadav et al. SC'22]

Factorization prevents efficient user-defined function support



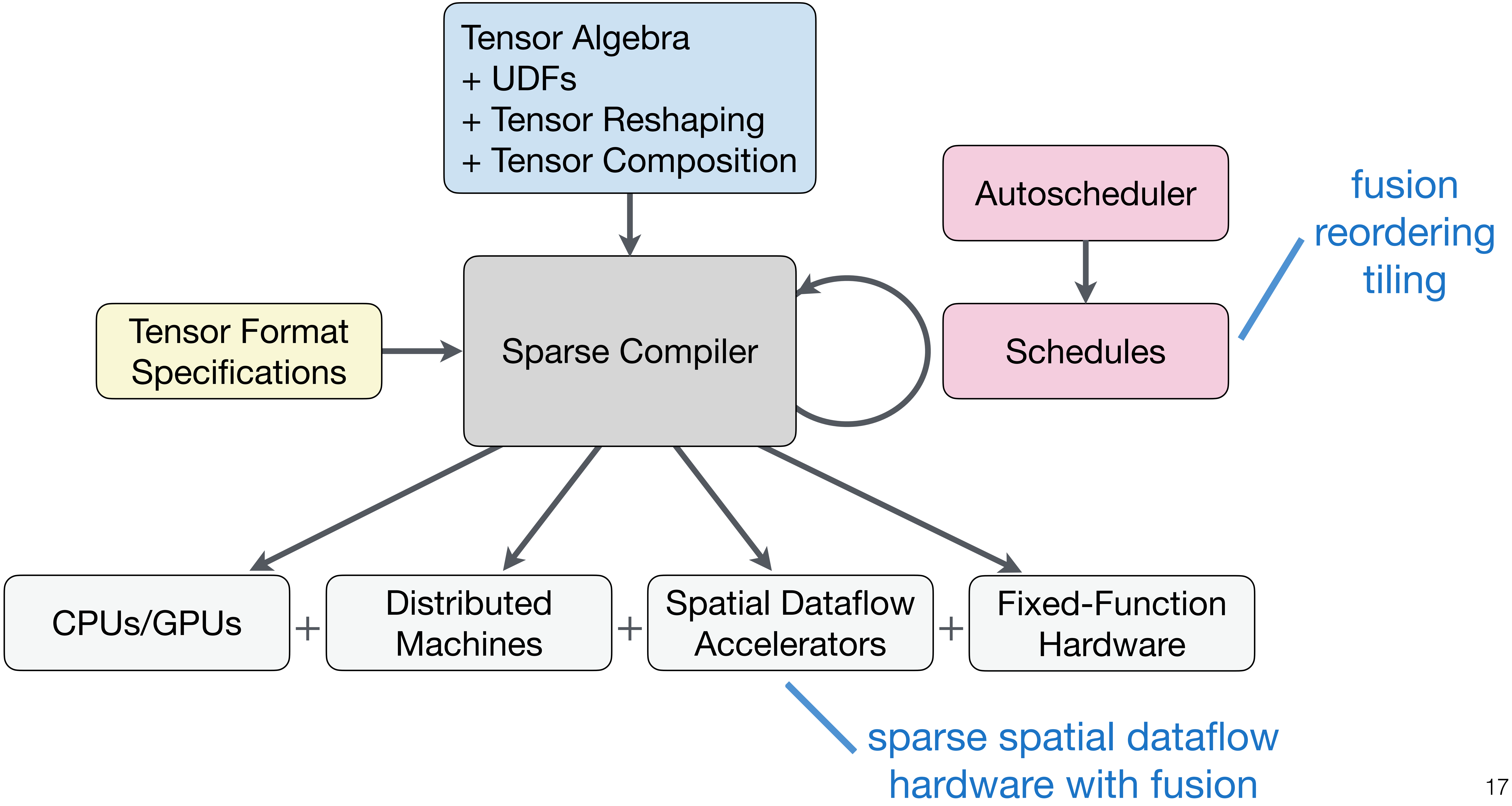
NumPy UDF NumPy UDF NumPy UDF



Average: 7.5x cost factorization

[Henry and Hsu et al. OOPSLA'21]

Compiler design for general sparse tensor operations


Tensor Algebra
+ UDFs
+ Tensor Reshaping
+ Tensor Composition

Tensor Format Specifications

Sparse Compiler

Autoscheduler

Schedules

fusion
reordering
tiling

CPUs/GPUs + Distributed Machines + Spatial Dataflow Accelerators + Fixed-Function Hardware

sparse spatial dataflow hardware with fusion

Hardware design for general sparse tensor operations

Sparse tensor algebra accelerators must support:

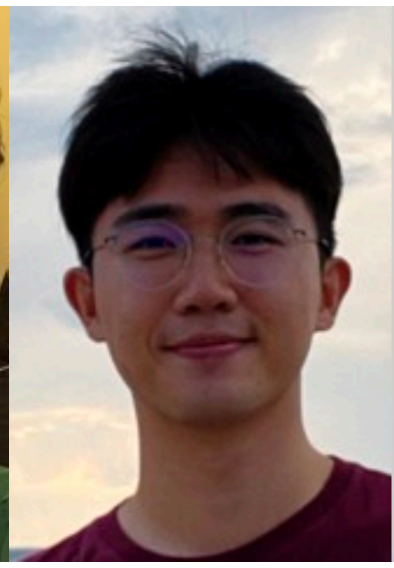
1. **Generality:** arbitrary tensor algebra operations
2. **Data Structures:** dense and sparse data structures
3. **Fusion:** Fusion across operations
4. **Reordering:** Changing the order they process tensor dimensions

The Sparse Abstract Machine

- Abstract spatial dataflow machine architecture
- Supports all four properties (generality, data structures, fusion, and reordering)
- Also supports tiling, parallelization, vectorization, and bitvector wire protocols
- Implemented in a simulator and first prototype taped out next month
- Straightforward to compile tensor algebra to the sparse abstract machine


Olivia
Hsu


Maxwell
Strange

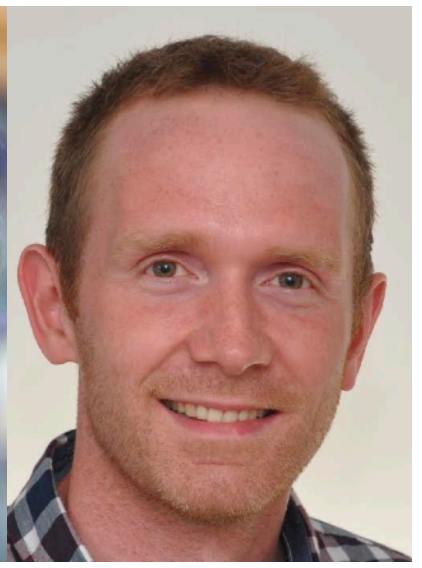

Jaeyeon
Won


Ritvik
Sharma


Kunle
Olukotun


Joel
Emer


Mark
Horowitz


Fred
Kjolstad

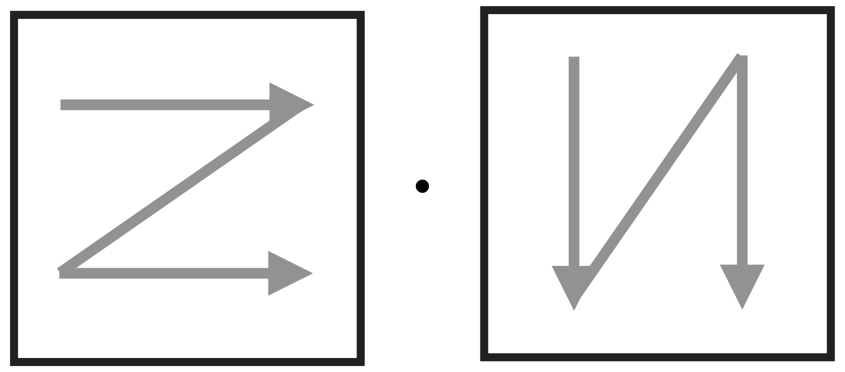
arxiv.org/abs/2208.14610

Inner-product sparse matrix multiplication

hierarchical scanners load coordinates for different tensor dimensions

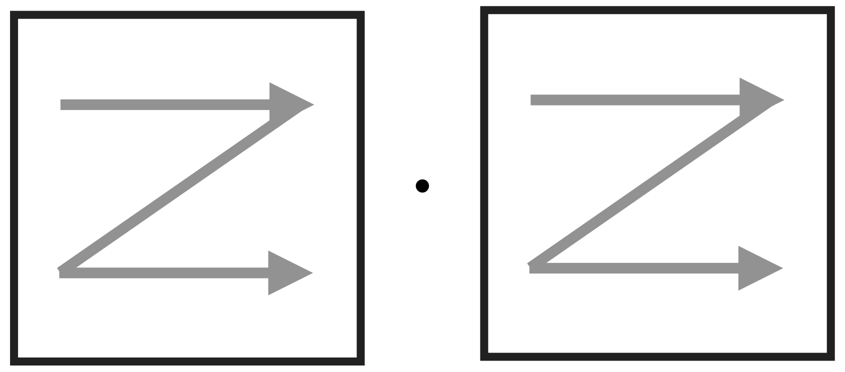
intersection filters coordinates

repeater implements dimension broadcasting



Gustafson sparse matrix multiplication

asymptotically less work,
because intersection occurs earlier



asymptotically more temporary memory,
because it must reduce whole rows

Fused SDDMM

$$A_{ij} = B \odot (CD)$$

D, 0

Level Scanner
C: level i compressed
ref
ref
crd

D, 0

Level Scanner
B: level i compressed
ref
ref
crd

D, 0

Intersecter
ref1
crd1
ref2
crd2
ref1
ref2
crd

Xi coordinate stream

Repeater
D: level i
crd
ref
ref

Level Scanner
B: level j compressed
ref
ref
crd

Level Scanner
D: level j compressed
ref
ref
crd

Intersecter
ref1
crd1
ref2
crd2
crd
ref1
ref2

Xj coordinate stream

Bj reference stream

Repeater
C: level j
ref
crd
ref

Level Scanner
C: level k compressed
ref
ref
crd

Level Scanner
D: level k compressed
ref
ref
crd

Intersecter
ref1
crd1
ref2
crd2
ref1
crd
ref2

Bj reference stream

Repeater
B: level k
ref
crd
ref

Array
C Values
ref
vals

Array
B Values
ref
vals

Array
D Values
ref
vals

Multiplier
vals
vals
vals

Multiplier
vals
vals
vals

Scalar Reducer
vals
vals

Xi coordinate stream

Xj coordinate stream

Level Writer
X: level i compressed
crd

Level Writer
X: values compressed
vals

Level Writer
X: level j compressed
crd

$$O(\mathrm{NNZ}_B \cdot K)$$

Conclusion and references

Unlike dense neural networks that can be reduced to GEMM,
it will not be possible to reduce sparse neural networks
to one optimized function

Compilation Approach

[Kjolstad et al. OOPSLA’17]
[Kjolstad et al. MIT’20]

Scheduling Language

[Kjolstad et al. CGO’19]
[Senanayake et al. OOPSLA’20]

Distributed Compilation

[Yadav et al. PLDI’22]
[Yadav et al. SC’22]

Sparse Abstract Machine

[Hsu et al. arXiv’22]

Format Abstractions

[Chou et al. OOPSLA’18]
[Chou et al. PLDI’20]

Autoscheduling

[Ahrens et al. PLDI’22]

User-Defined Functions

[Henry and Hsu et al. OOPSLA’21]

Verification

[Kovach and Kjolstad arXiv’22]