# TACO: The Tensor Algebra Compiler

Kjolstad, Kamil, Chou, Lugato, Amarasinghe (2017)

Jakub Bachurski (jkb55), 27 November 2024

# Let's get down to business

## to process the data

- Data analysts will just give you the linear algebra operations
- Library implementers will do the least amount of work to let you perform them

```
A = ttv(B, C, D)

A = innerprod(B, C)

A = mttkrp(B, {C, D, E}, F)
```

- If you've much of NumPy-like programming (e.g. PyTorch), you'll know the joy of reducing everything to just the kernels you've got.

# The abstract hits the concrete wall

```
A = ttv(B, C, D)

A = innerprod(B, C)

A = mttkrp(B, {C, D, E}, F)
```

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

$$A = B_{ijk} \cdot C_{ijk}$$

$$A_{ij} = B_{ikl} \cdot D_{lj} \cdot C_{kj}$$

(If an index is not on LHS, sum over it)

$$A_{ij} = B_{ik}C_{kj} \rightsquigarrow A_{ij} = \sum_k B_{ik}C_{kj}$$

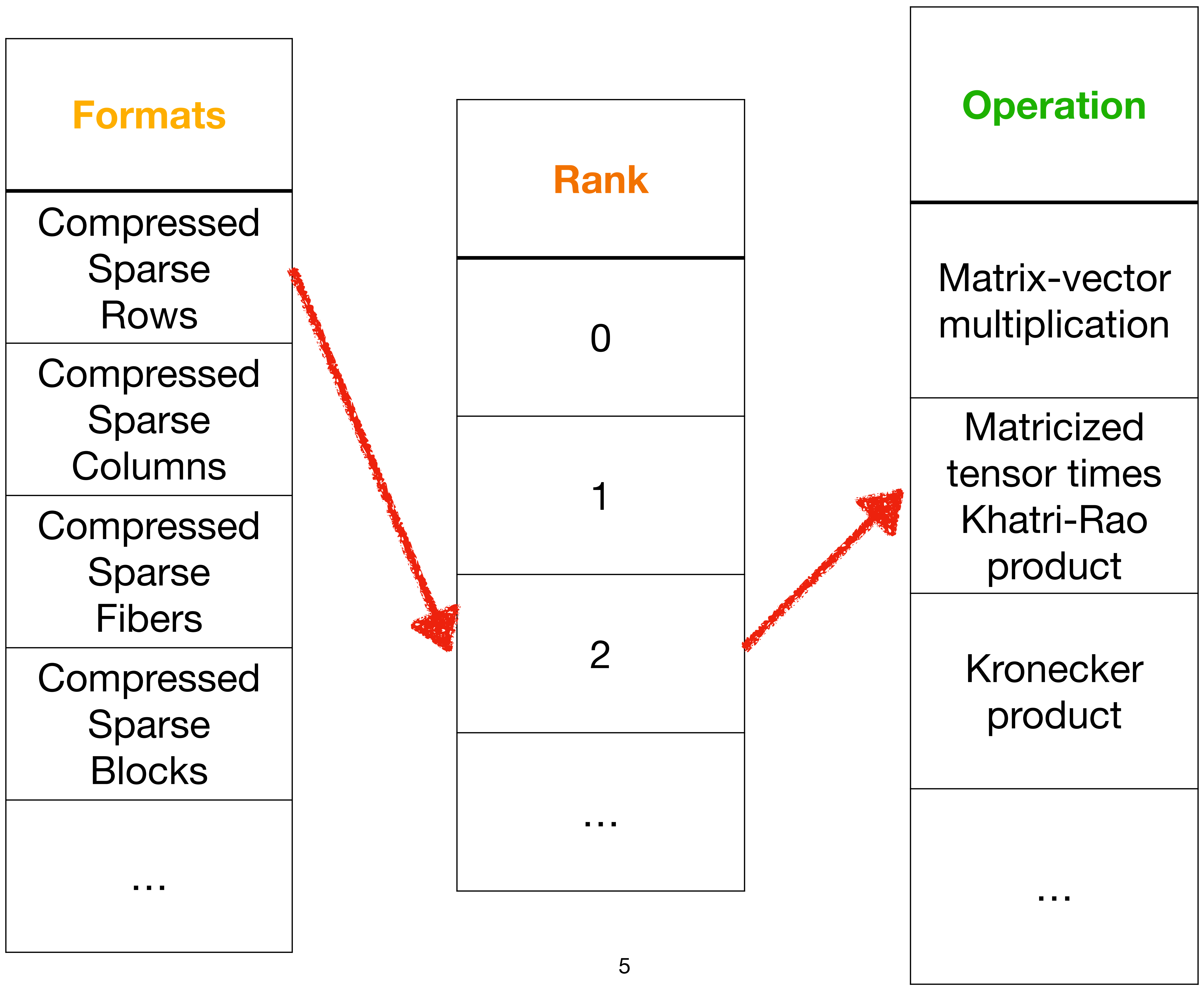
# Sparsity is all you need

**(sometimes)**

- Data analysts should do linear algebra as they please
- But most data is *empty* - e.g. near-zero
- $\Longrightarrow$ **sparse linear algebra**

# Curse of trade-offs

## Any colour, as long as it is black



# You’ve got problems

## We’ve got the solutions

- $\Longleftarrow$ Programmers need bespoke impl. suited to **storage** and **operations**
- $\Longrightarrow$ A unified **language** for storage and operations
- $\Longrightarrow$ A universal **compiler** into efficient algorithm implementations

# Contributions

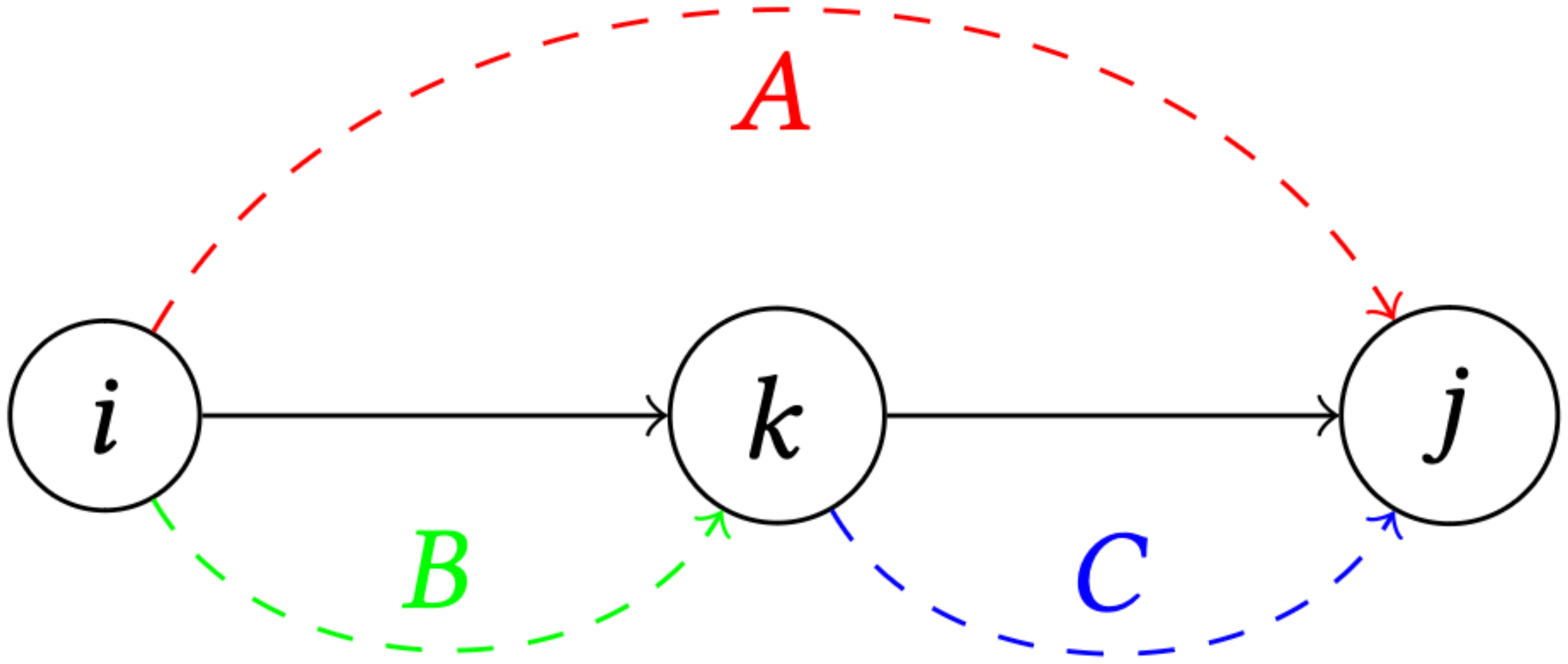
# Expression language

$$A_{i,j} = \sum_k B_{i,k} C_{k,j}$$

- **Tensor expressions** inspired by Ricci calculus (Einstein summation)

```
A = einsum("ik,kj->ij", B, C)
```

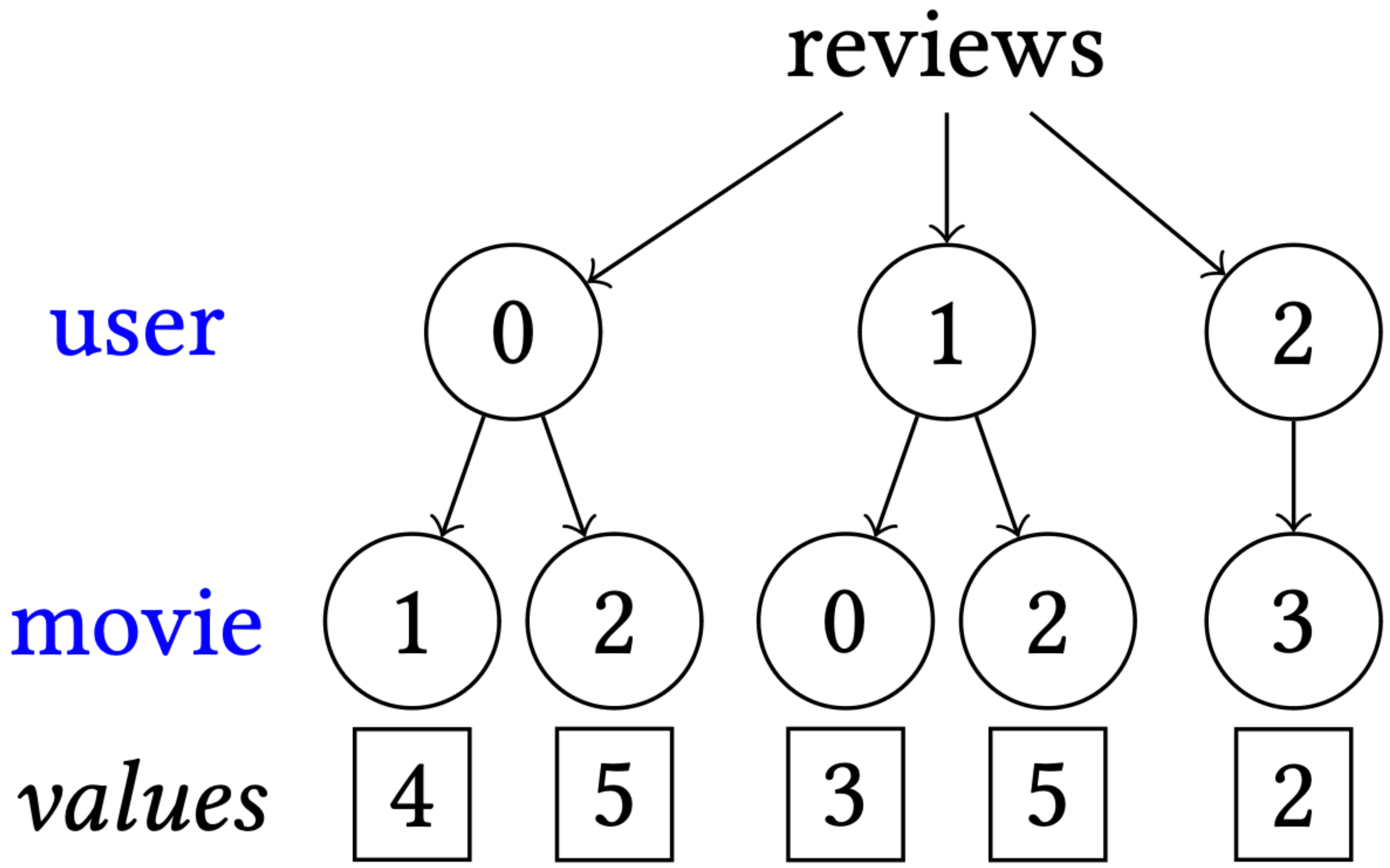
- Graphic representation: **iteration graphs**



# To store a sparse tensor

## Compressed Sparse Fiber (CSF) crash course

$$\text{reviews} = \begin{array}{cc} \text{movie} & \\ \begin{bmatrix} 0 & 4 & 5 & 0 \\ 3 & 0 & 5 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} & \text{user} \end{array} \Longrightarrow$$



# Storage language

$(\text{dense}_{\text{user}},\ \text{dense}_{\text{movie}})$

**CSR**

$(\text{dense}_{\text{user}},\ \text{sparse}_{\text{movie}})$

**CSC**

$(\text{sparse}_{\text{user}},\ \text{dense}_{\text{movie}})$

**CSF (row-major)**

$(\text{sparse}_{\text{user}},\ \text{sparse}_{\text{movie}})$

**CSF (column-major)**

$(\text{sparse}_{\text{movie}},\ \text{sparse}_{\text{user}})$

# API

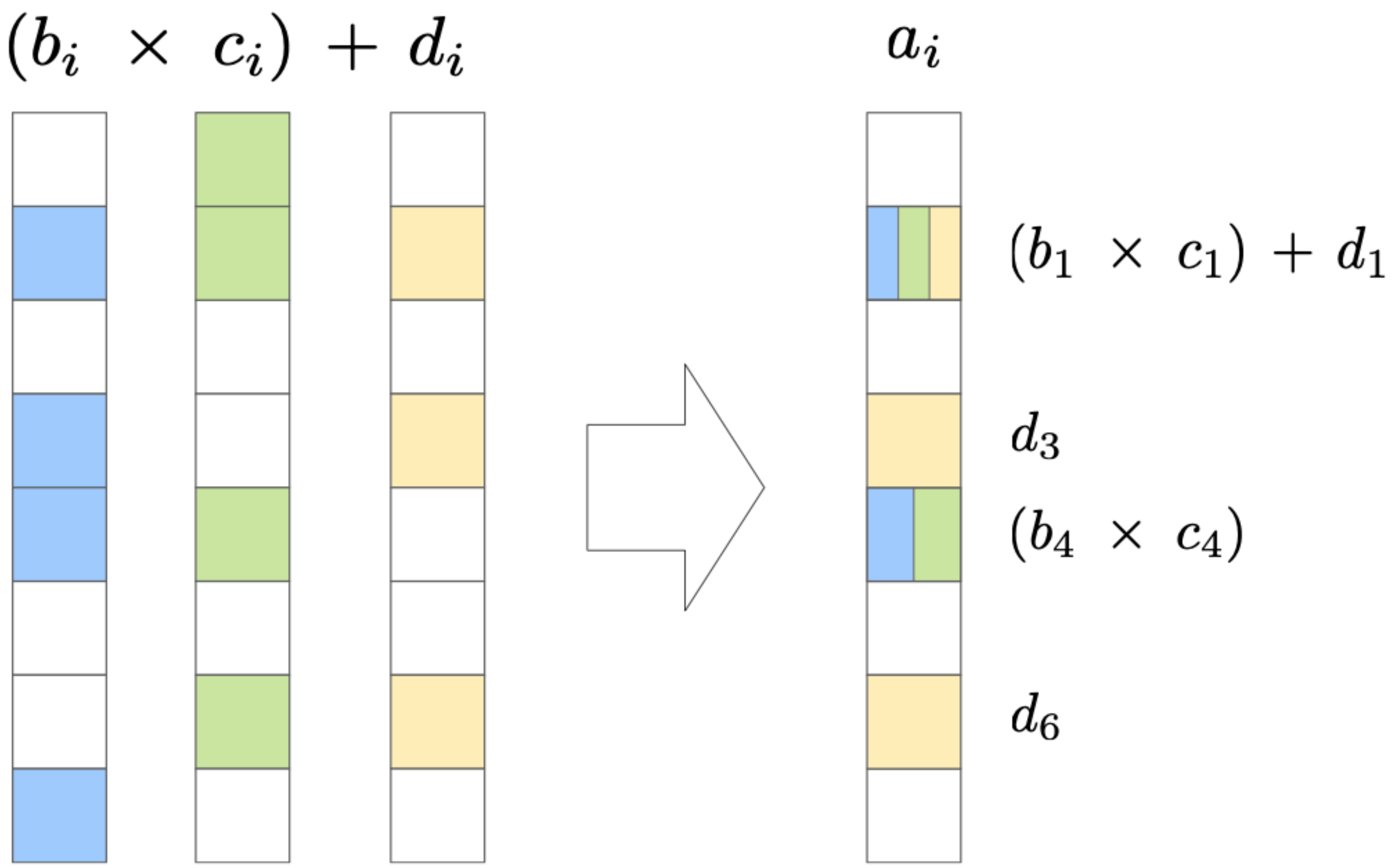
## DSL embedded in C++

```
// -   storage   -
Format csr({Dense,Sparse});
Tensor<double> A({64,42}, csr);
Format csf({Sparse,Sparse,Sparse});
Tensor<double> B({64,42,512}, csf);
// - expressions -
IndexVar i, j, k;
A(i,j) = B(i,j,k) * c(k);
```

# Sparse computation is just *merge-sort*

$$x + 0 = x \qquad x \times 0 = 0$$

$$a_i = (b_i \times c_i) + d_i \quad \rightsquigarrow \quad a_i = (b_i \wedge c_i) \vee d_i$$



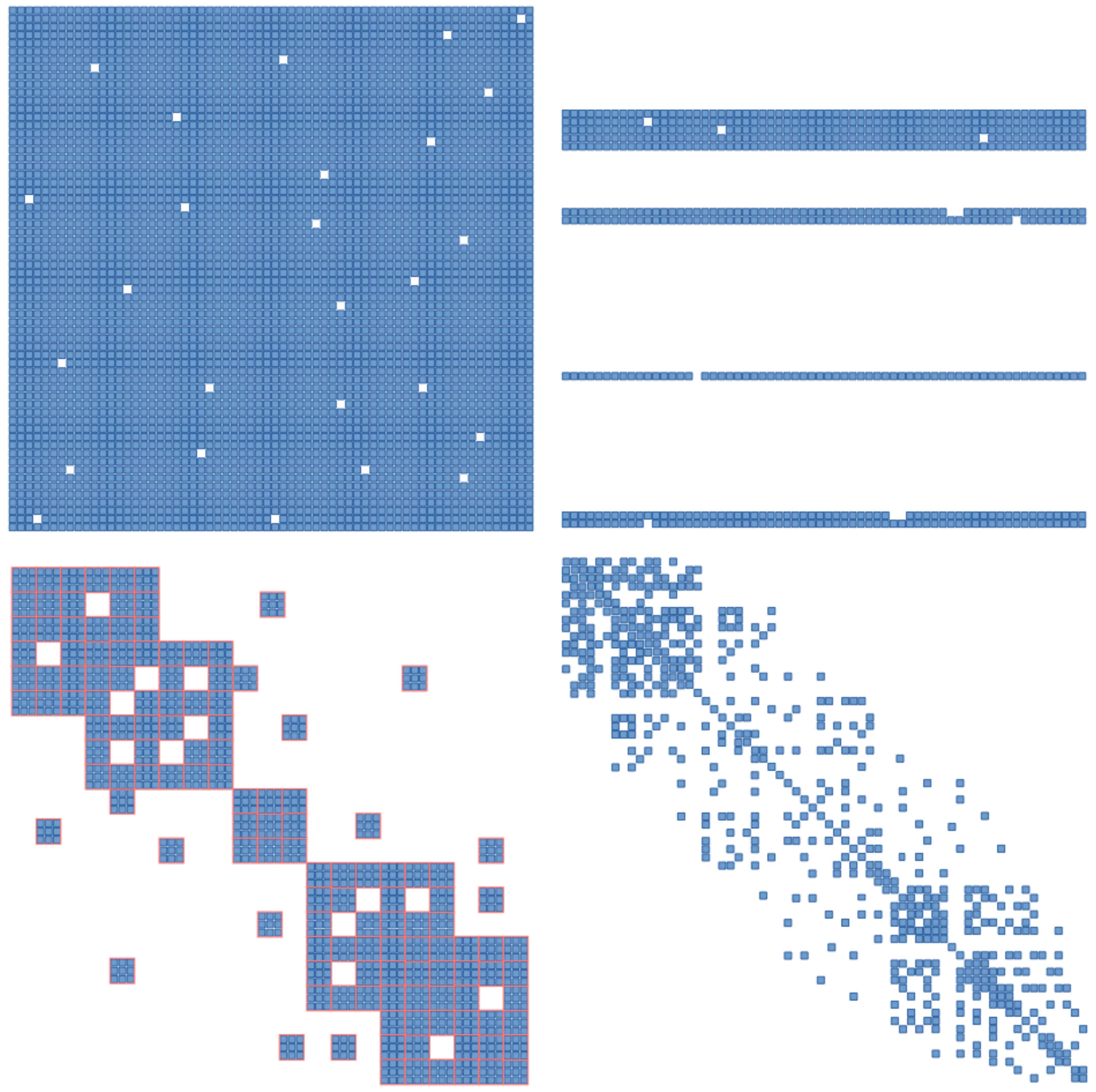
# Compiler

- The compiler constructs ***merge lattices*** for the given expressions
- Generates C code which reads data & writes result in given formats
- **Most general** code generation approach so far
- Parallelised by **OpenMP**

# Conclusions

# Results

- Consistently **strong performance** against **many libraries**. *Concerns?*
  - Sparse tensors often overlooked
  - Libraries often very general — ‘best-effort’?
- Benchmarking isn’t obvious:
  - General: available operations limited in baselines
  - Data sparsity operations!



# Limitations / Future Work

- Tensor expressions cannot transpose (e.g. $A = B^T \Leftrightarrow A_{ij} = B_{ji}$)
- Other storage formats (e.g. Compressed Sparse Blocks, Dictionary of Keys)
- Non-CPU devices
- Scheduling / optimisations to match best-effort performance

# Summary

- **Design** & **implementation** of (domain-specific) programming language
- Sparse tensors are **important**, but usually need to be **hand-implemented**
- *Algorithmic insights often useful:*
  - Iteration graphs and storage trees
  - Two-way merge and merge lattices

$$A_{i,j} = \sum_k B_{i,k} C_{k,j}$$

# You need models!

# (but question them)

# Thank you!

**Thoughts?**