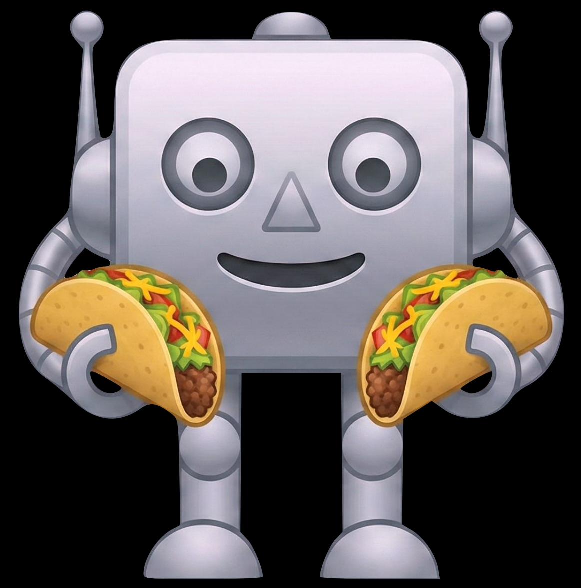
UNIVERSITY OF CAMBRIDGE



TACO

The Tensor Algebra Compiler

Fredrik Kjolstad, Shoaib Kamil, Stephen Chou, David Lugato,
and Saman Amarasinghe

26 November 2025 • University of Cambridge • Cambridge, UK

Carl Seifert
`cs2331@cam.ac.uk`

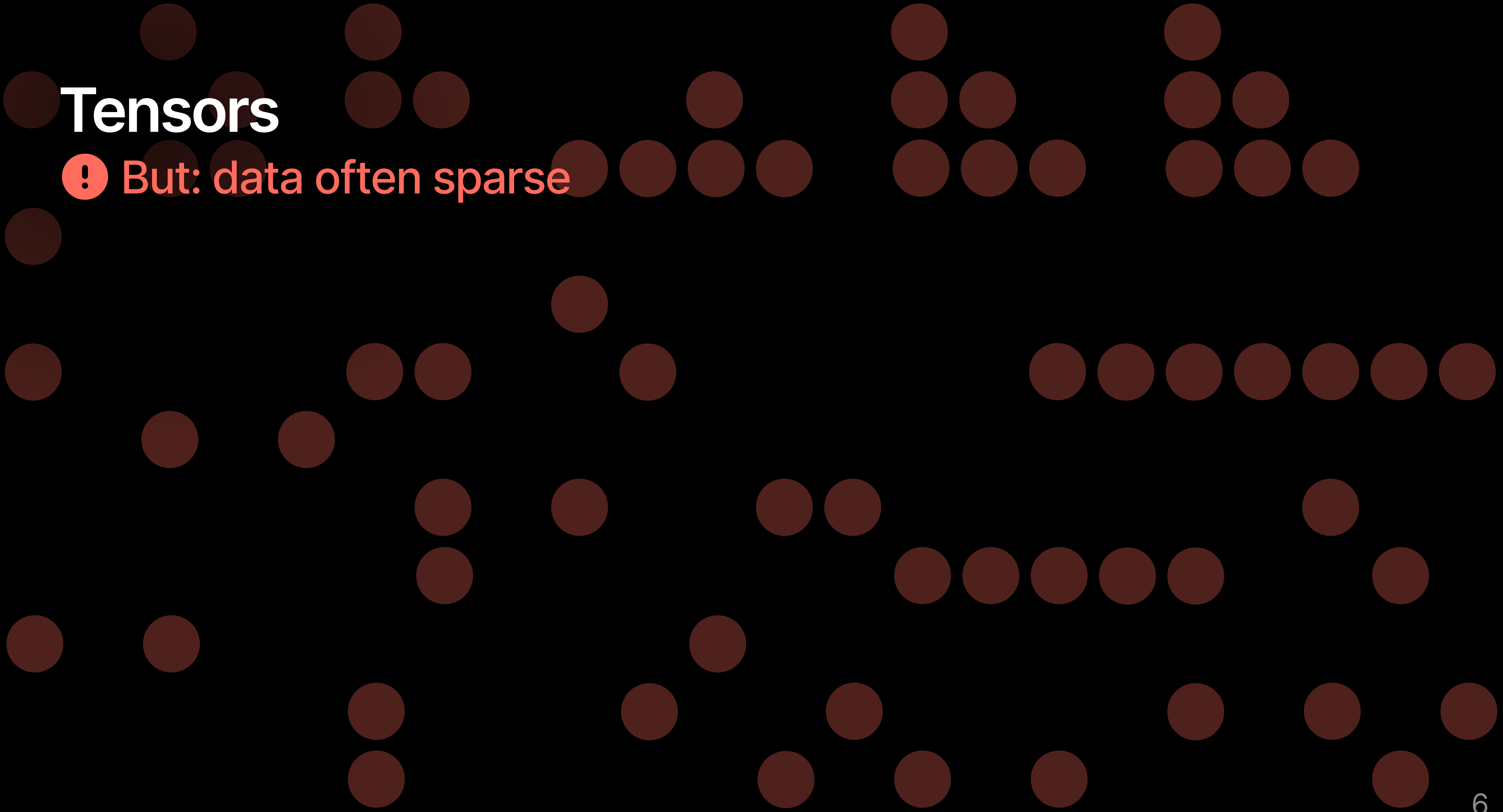
Tensors!

Tensors

❗ But: data often sparse

Tensors

- ❗ But: data often sparse



Tensors

❗ But: data often sparse

99.99999998%

Zeros in Amazon Review Tensor 2017

Tensors

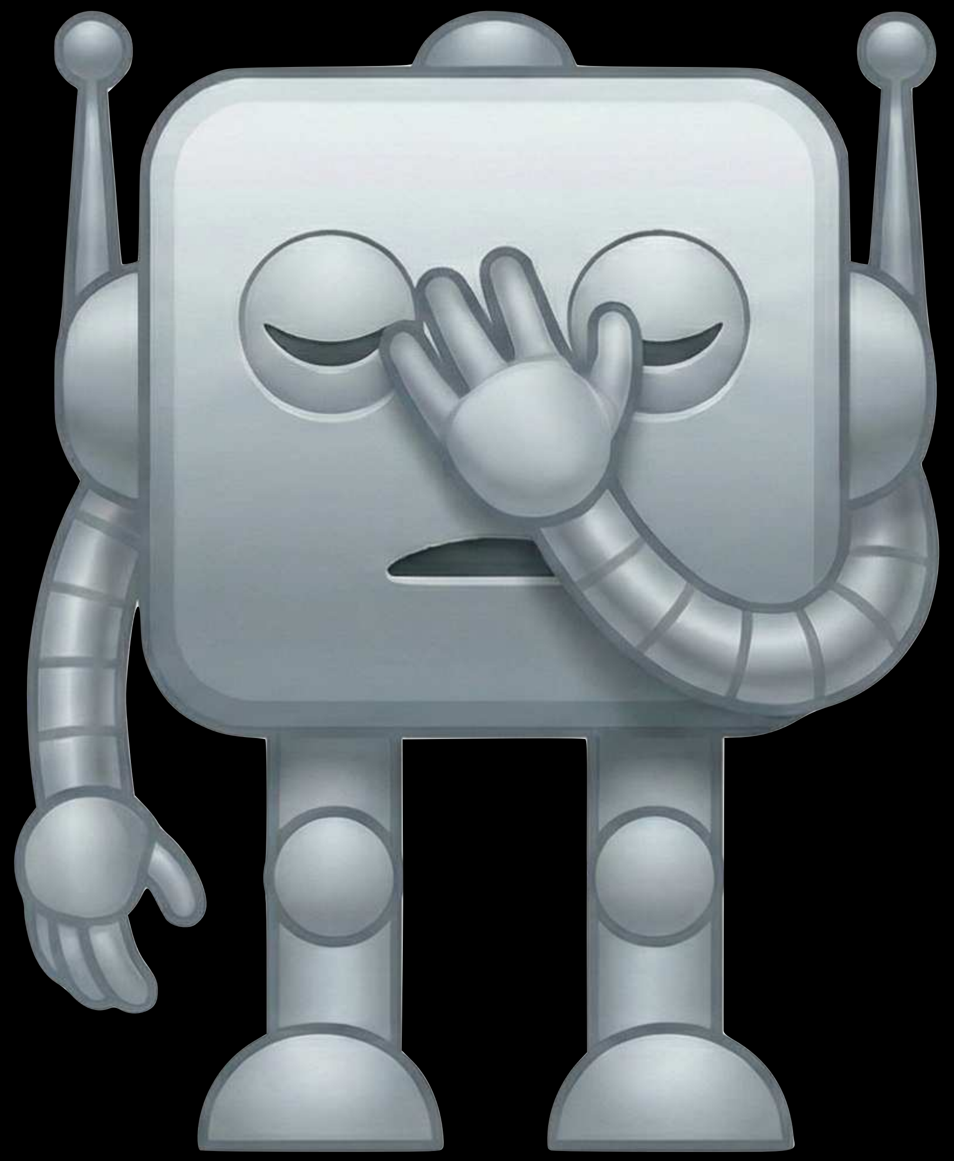
❗ But: data often sparse

Tensors

- But: data often sparse
- Many storage formats

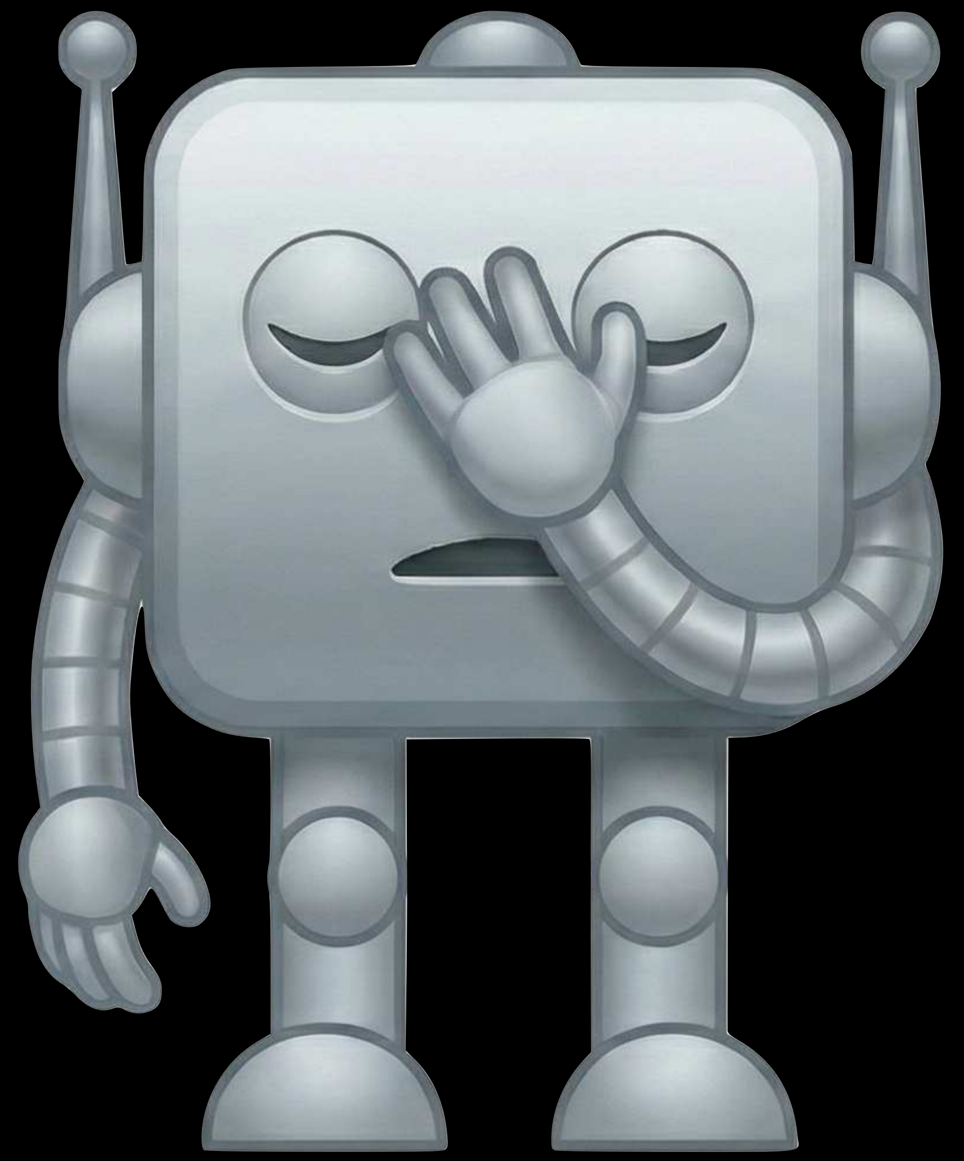
Tensors

- But: data often sparse
- Many storage formats



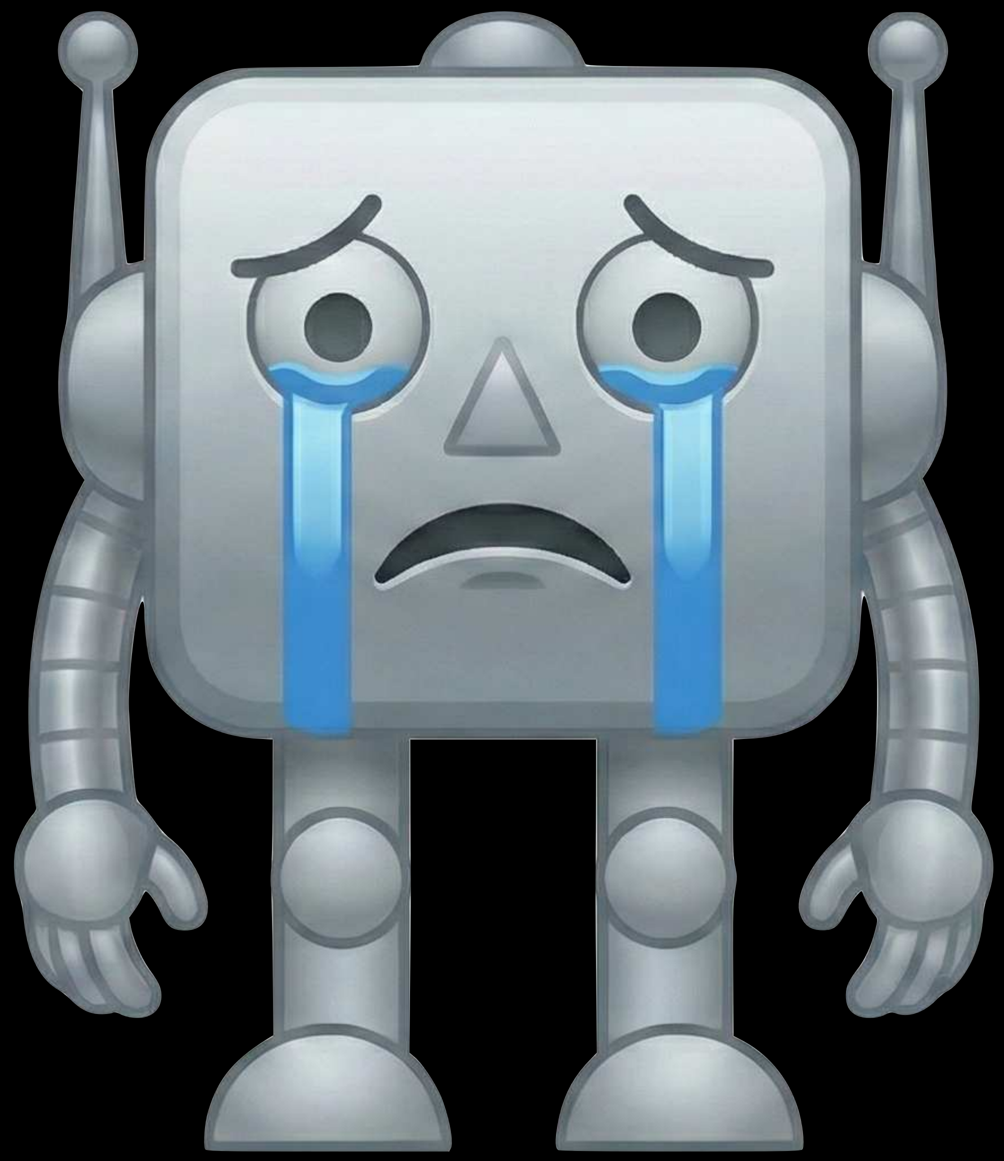
Tensors

- But: data often sparse
- Many storage formats
- Many operations



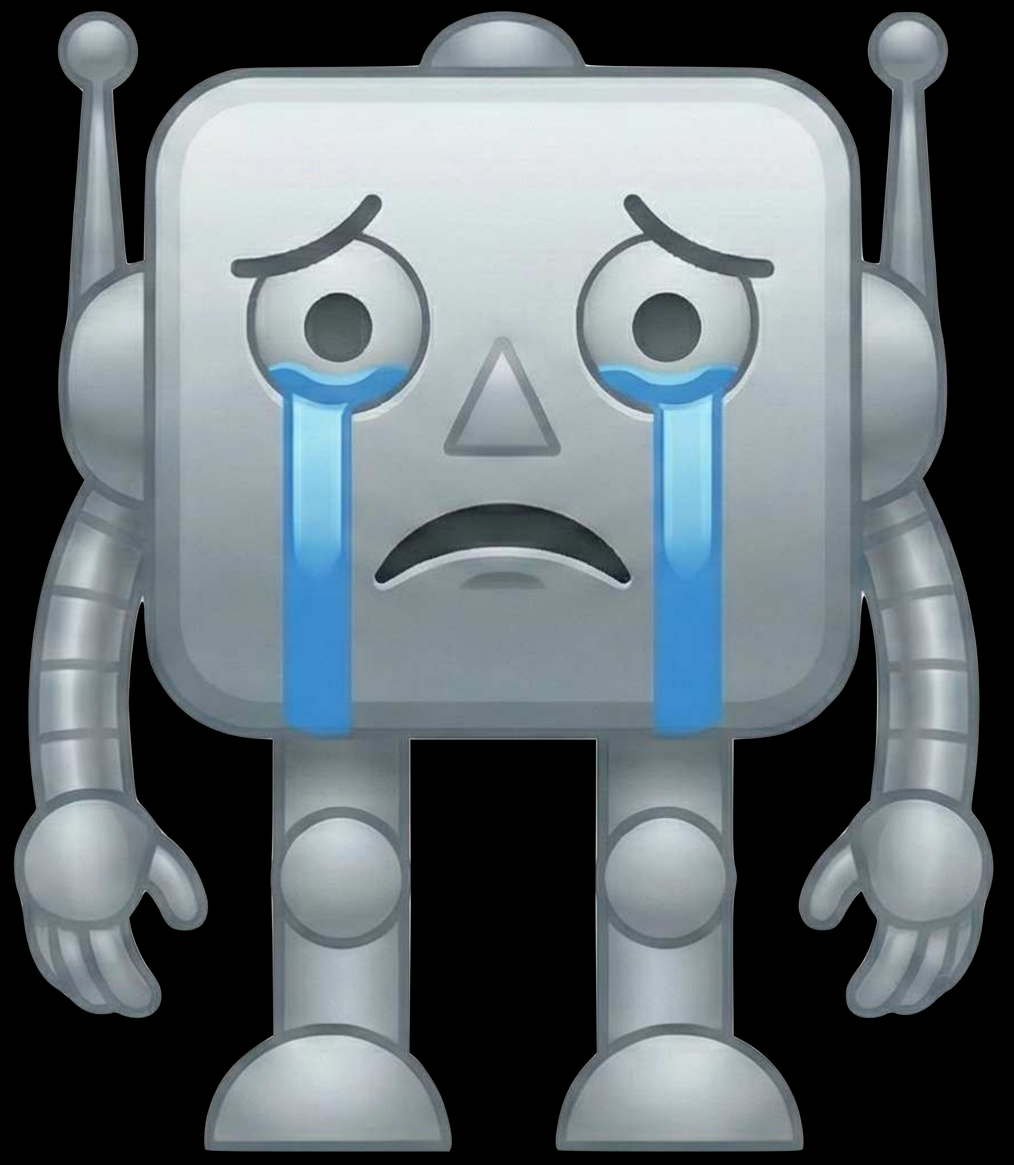
Tensors

- But: data often sparse
- Many storage formats
- Many operations



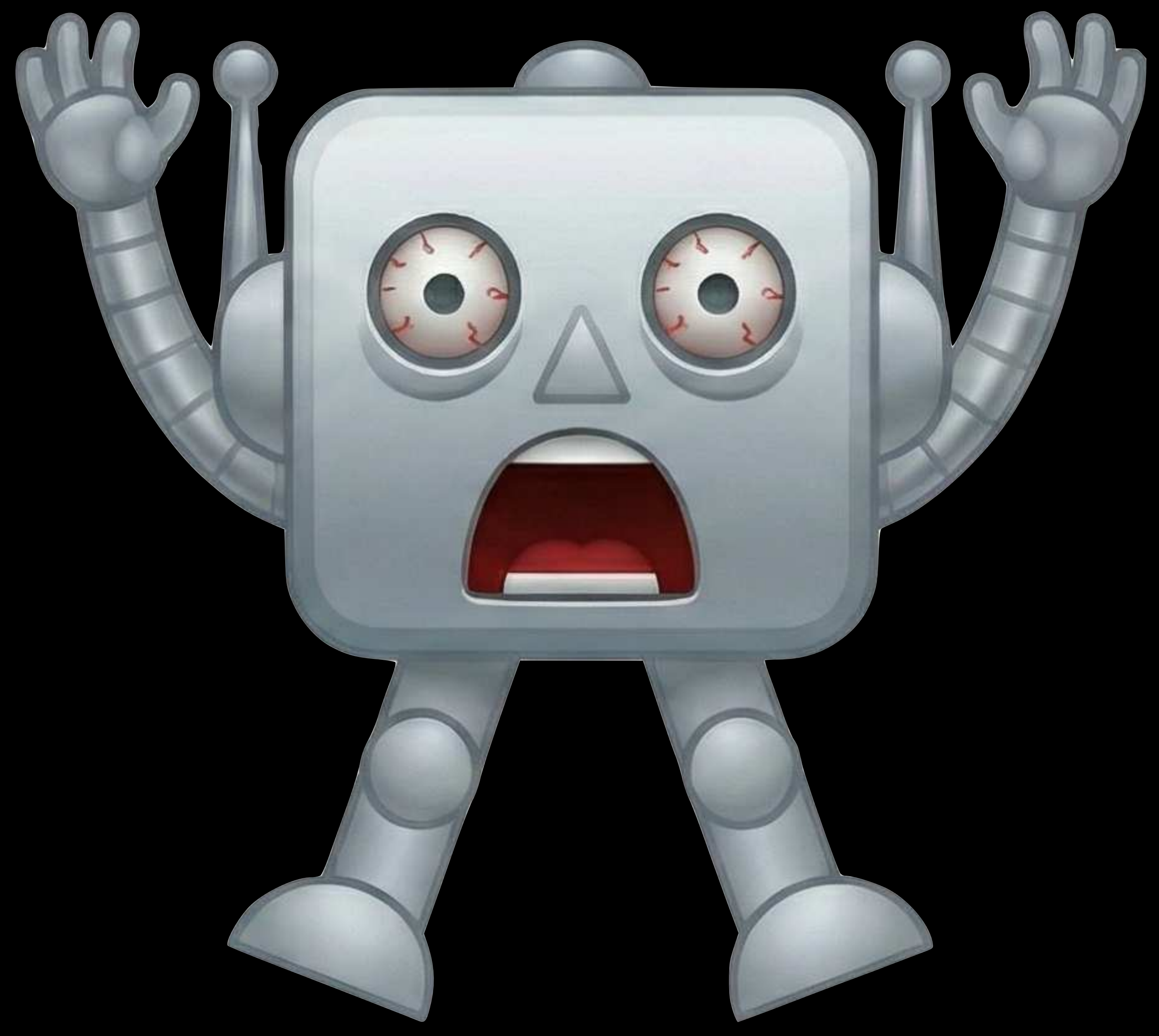
Tensors

- ❗ But: data often sparse
- ❗ Many storage formats
- ❗ Many operations
- ❗ Many versions of operations, hand-optimised kernels



Tensors

- But: data often sparse
- Many storage formats
- Many operations
- Many versions of operations, hand-optimised kernels



Goals

- Many storage formats
- Many different operations
- Auto-generated kernels for any expression

TACO

The Tensor Algebra Compiler

TACO

The Tensor Algebra Compiler

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

TACO

The Tensor Algebra Compiler

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

SPARSE

SPARSE SPARSE SPARSE

DENSE DENSE

TACO

The Tensor Algebra Compiler

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

SPARSE

SPARSE SPARSE SPARSE

DENSE DENSE

Expression, format

TACO

The Tensor Algebra Compiler

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

SPARSE

SPARSE SPARSE SPARSE

DENSE DENSE

Expression, format → Compiler

TACO

The Tensor Algebra Compiler

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

A: i DENSE, j SPARSE

B: i DENSE, j SPARSE, k SPARSE

c: k SPARSE

C

Expression, format → Compiler → C code

TACO

The Tensor Algebra Compiler

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

C

Expression, format → Compiler → C code

1

2

3

1 Sparse Storage Formats

Store only sparse indices + boundaries

❶ Sparse Storage Formats

Store only sparse indices + boundaries

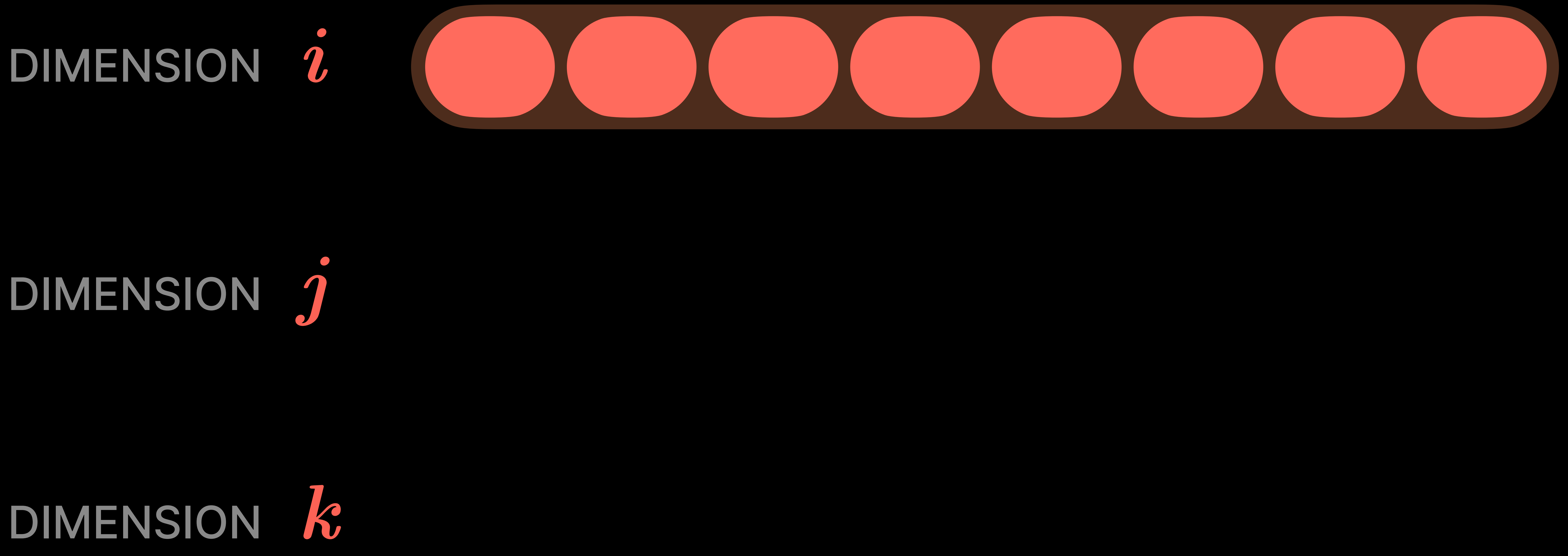
DIMENSION i

DIMENSION j

DIMENSION k

❶ Sparse Storage Formats

Store only sparse indices + boundaries

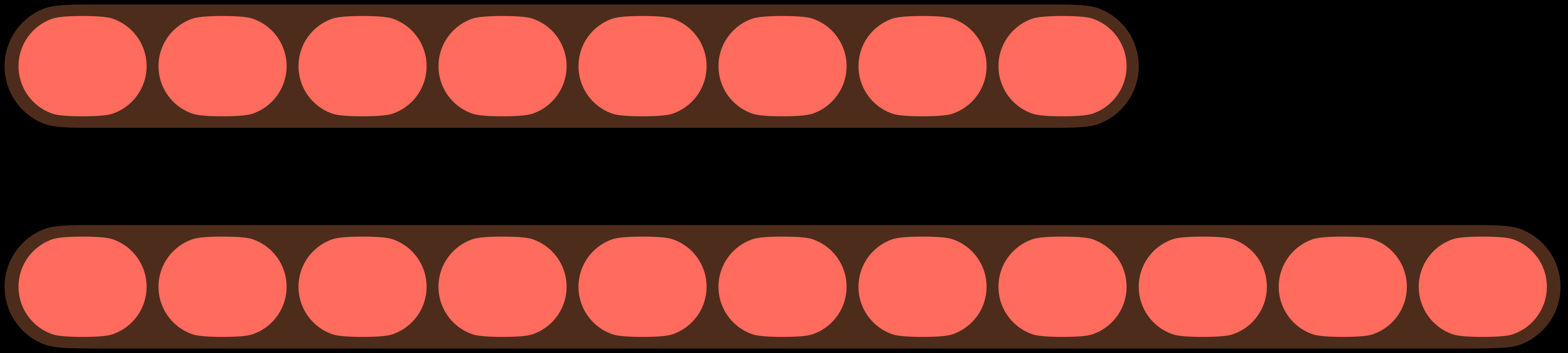


❶ Sparse Storage Formats

Store only sparse indices + boundaries

DIMENSION i

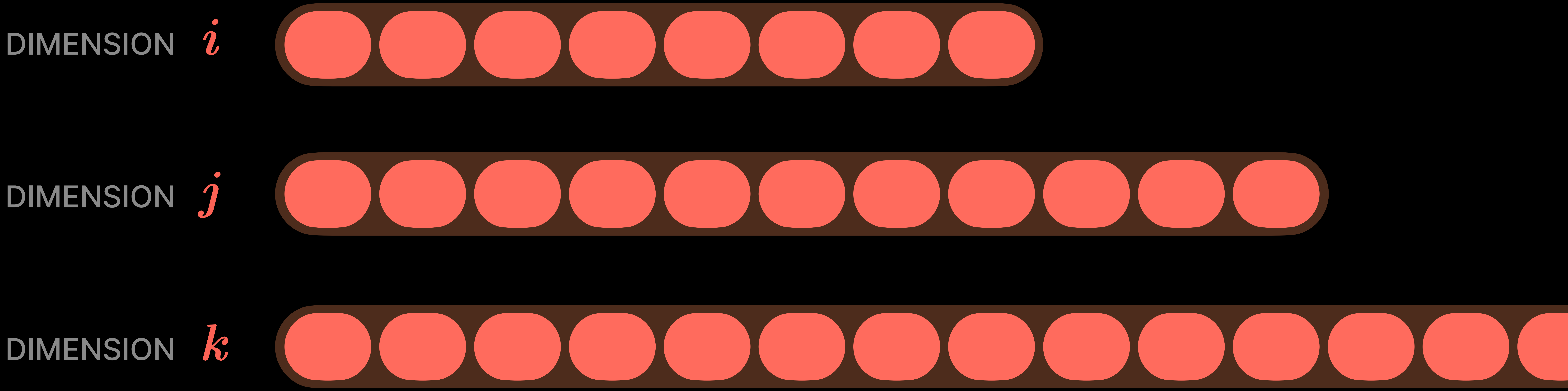
DIMENSION j



DIMENSION k

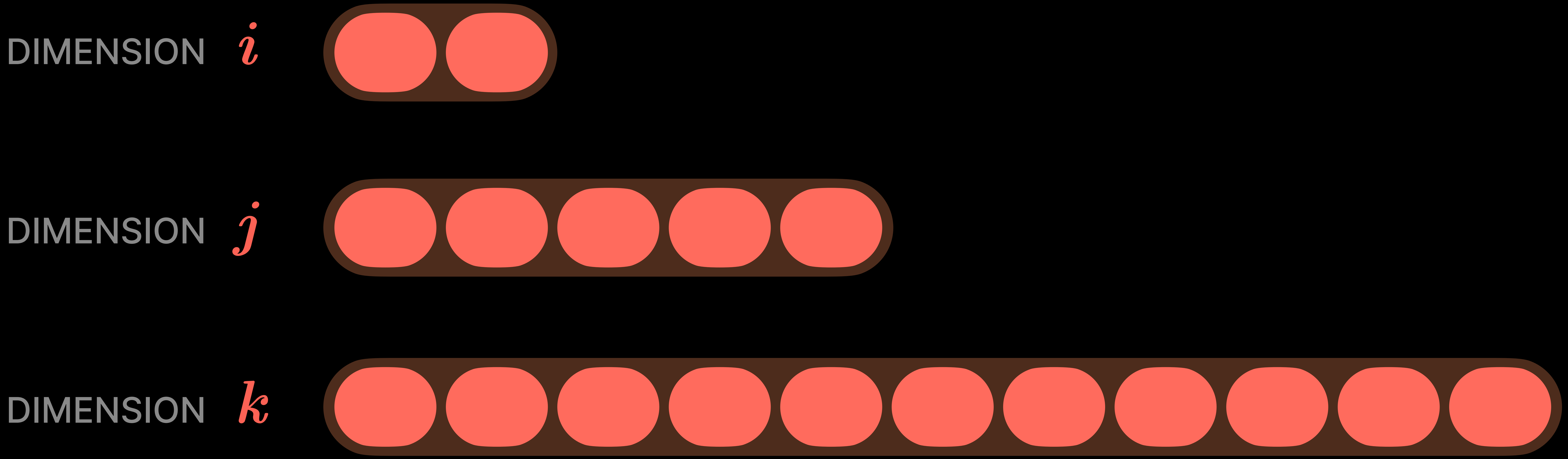
1 Sparse Storage Formats

Store only sparse indices + boundaries



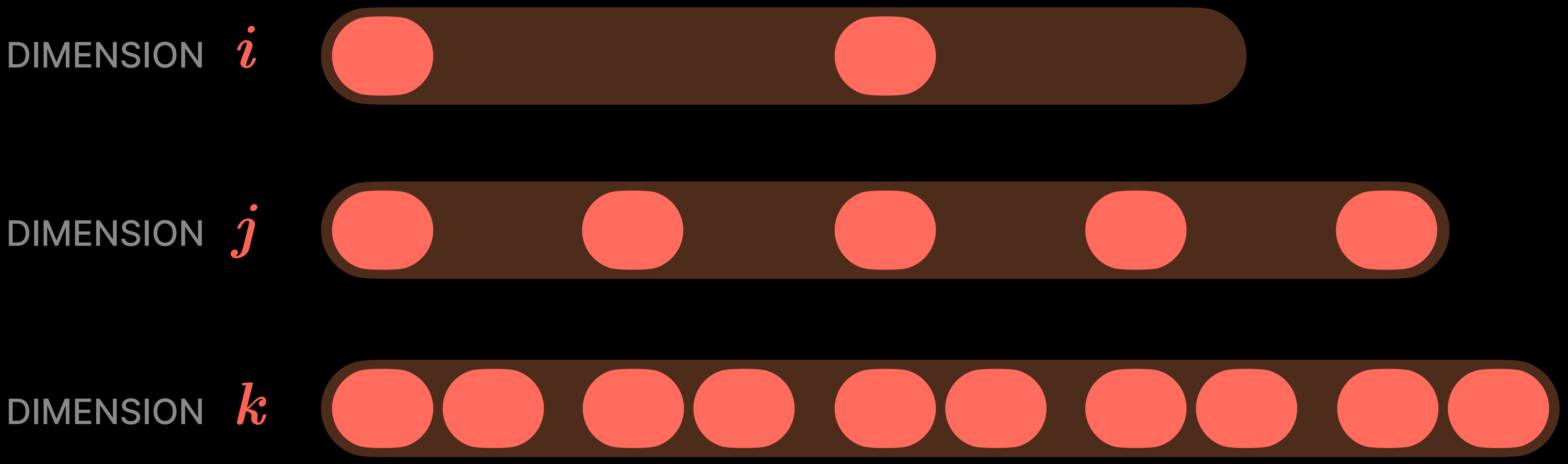
❶ Sparse Storage Formats

Store only sparse indices + boundaries



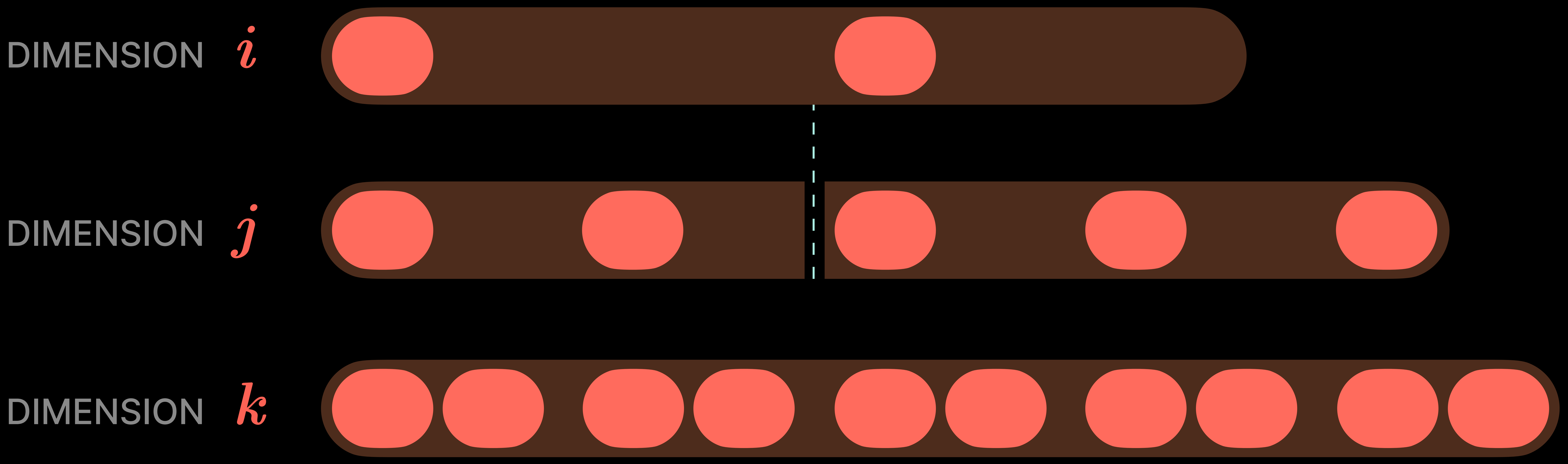
1 Sparse Storage Formats

Store only sparse indices + boundaries



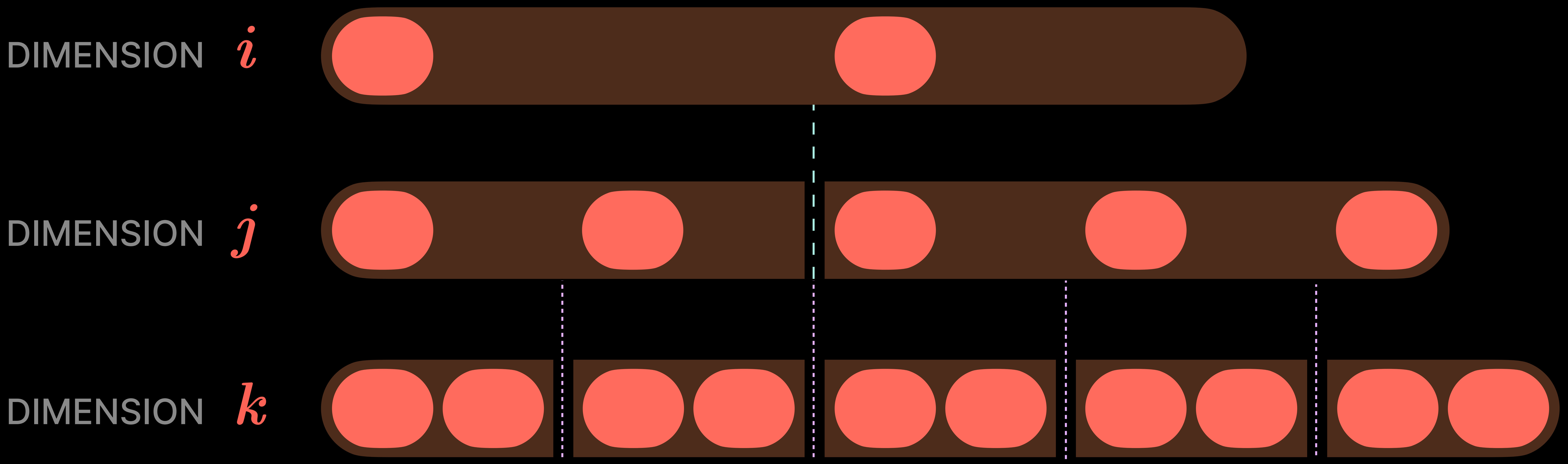
❶ Sparse Storage Formats

Store only sparse indices + boundaries



❶ Sparse Storage Formats

Store only sparse indices + boundaries



1 2 3

❷ Iteration Graphs

Represent co-iterable indices

❷ Iteration Graphs

Represent co-iterable indices

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

2 Iteration Graphs

Represent co-iterable indices

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

❷ Iteration Graphs

Represent co-iterable indices

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

DIMENSION i

DIMENSION j

DIMENSION k

❷ Iteration Graphs

Represent co-iterable indices

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

B A

DIMENSION i

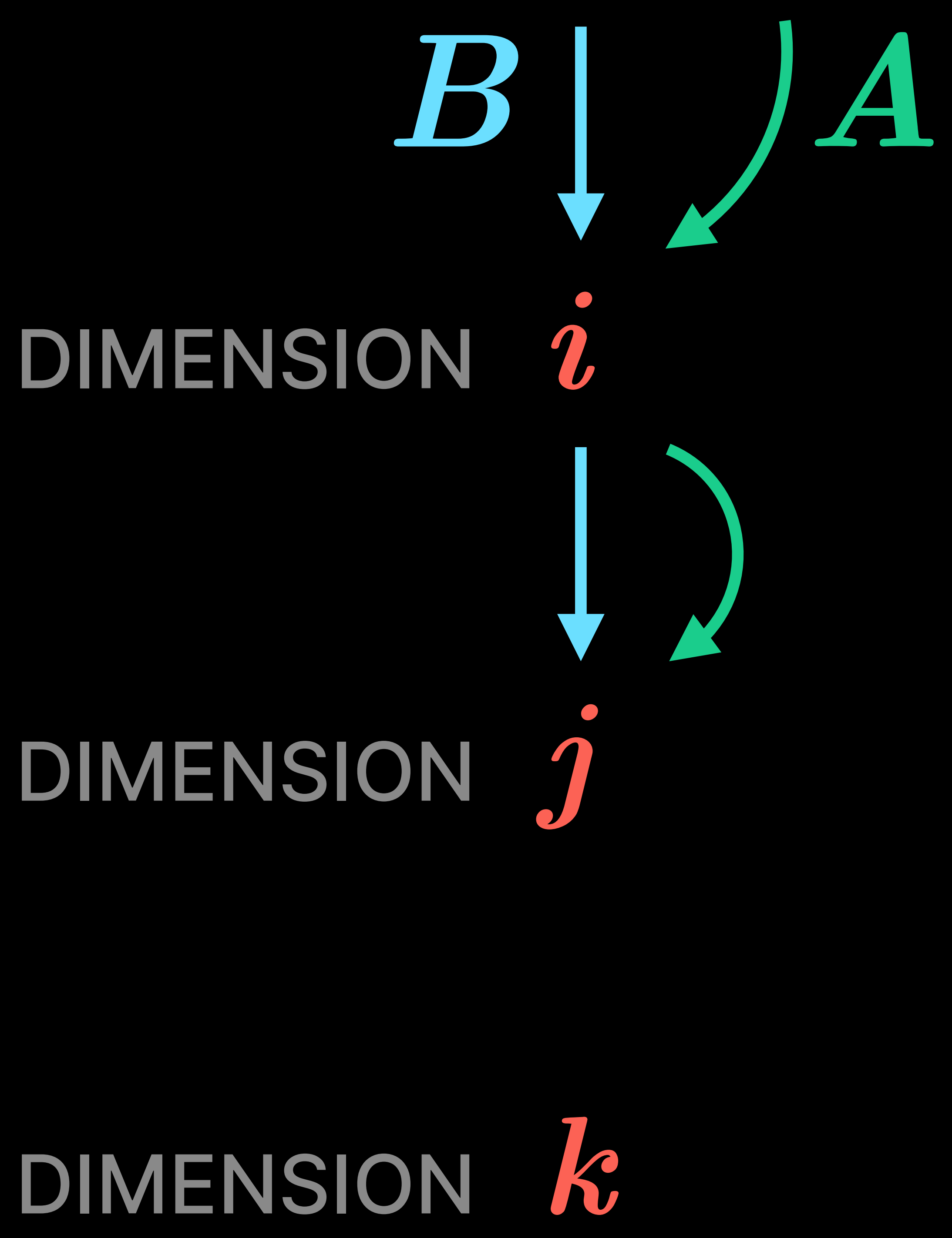
DIMENSION j

DIMENSION k

2 Iteration Graphs

Represent co-iterable indices

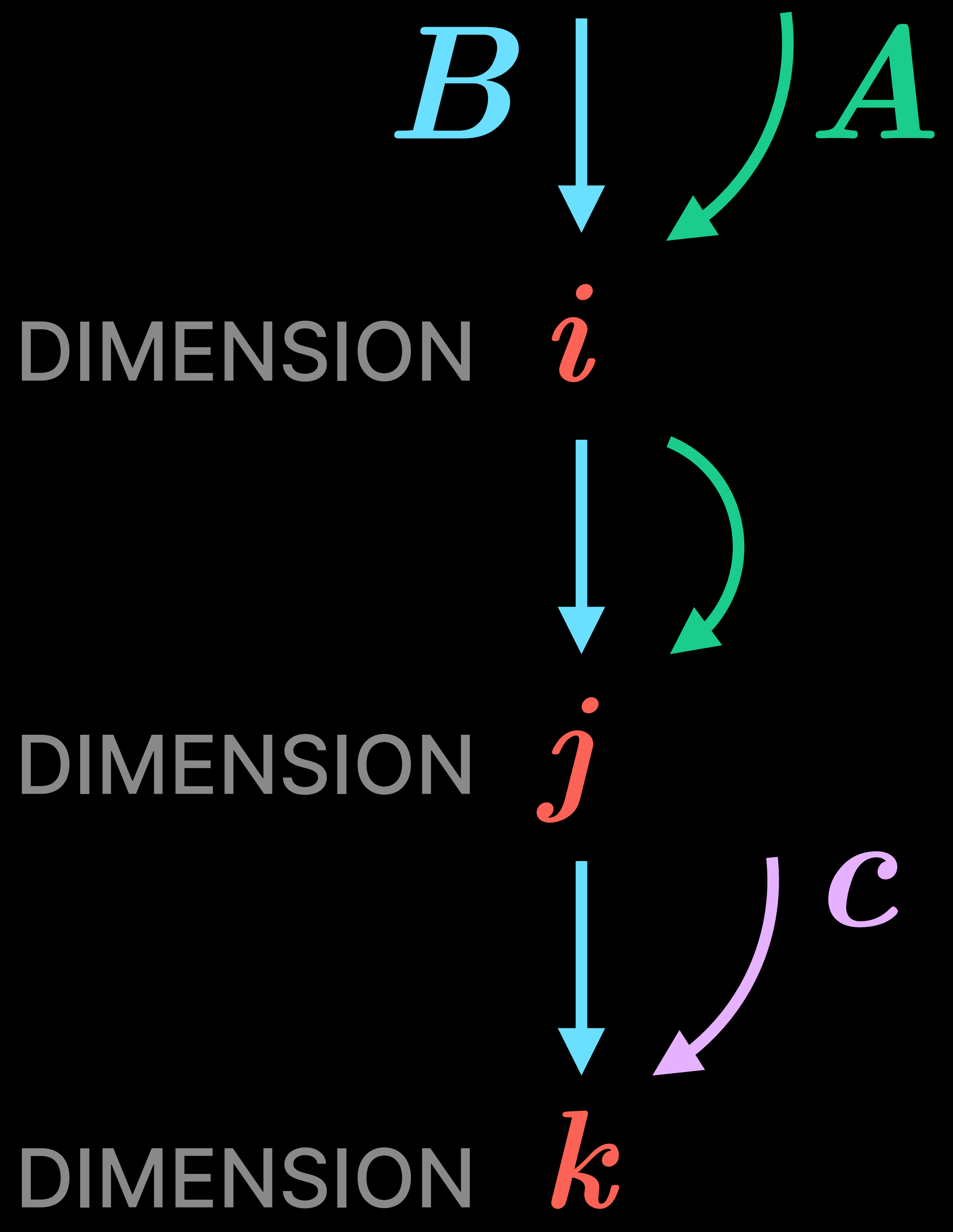
$$A_{ij} = B_{ijk} \cdot c_k$$



Iteration Graphs

Represent co-iterable indices

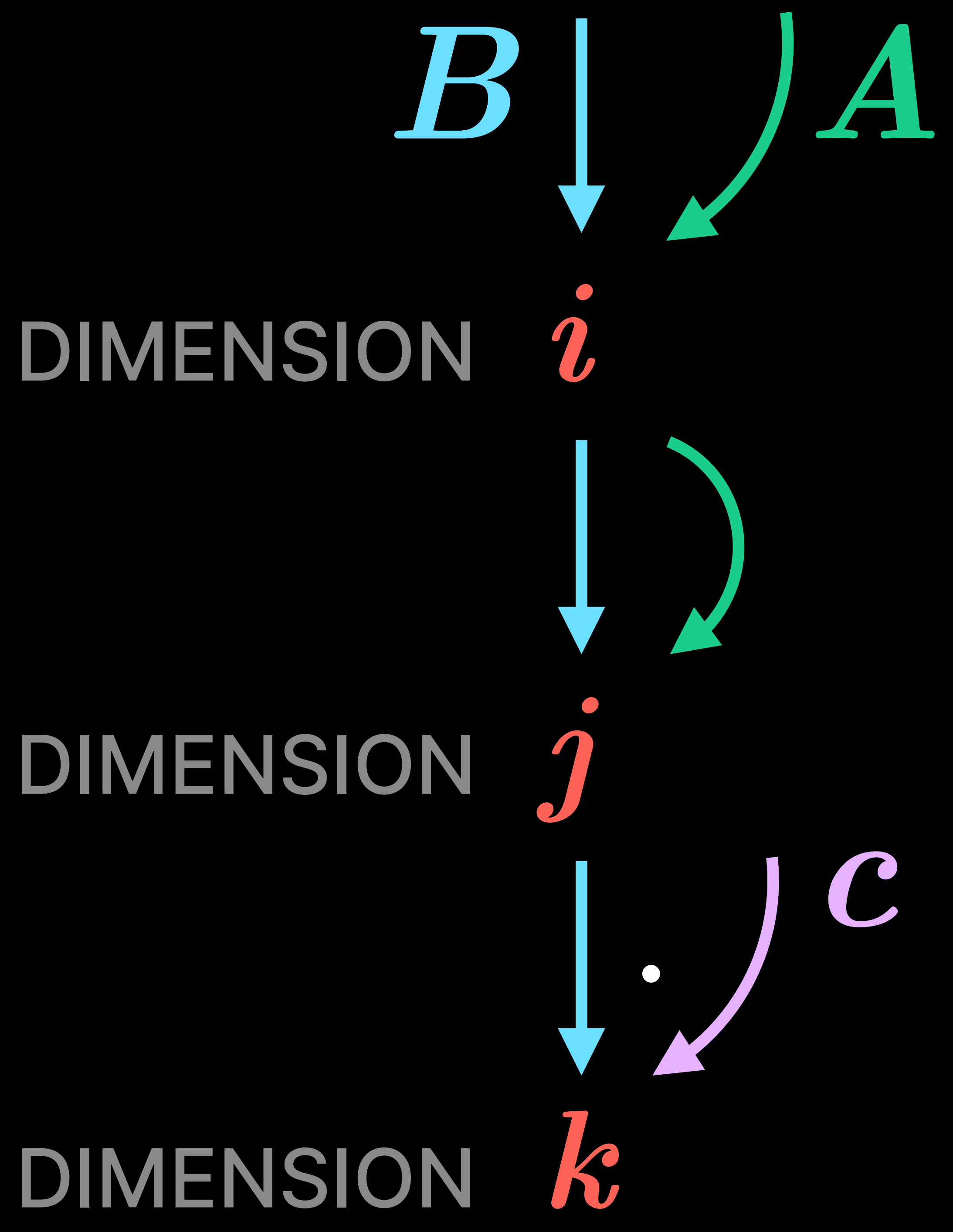
$$A_{ij} = B_{ijk} \cdot c_k$$



2 Iteration Graphs

Represent co-iterable indices

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



1

2

3

3 Merge Lattices

Basis for optimisation of co-iteration over indices (two-way merge)

❸ Merge Lattices

Basis for optimisation of co-iteration over indices (two-way merge)

Multiple **sparse** tensor parts to iterate
over within in one dimension,
need lots of conditions

❸ Merge Lattices

Basis for optimisation of co-iteration over indices (two-way merge)

Multiple **sparse** tensor parts to iterate
over within in one dimension,
need lots of conditions

→ split each dimension loop into multiple

3 Merge Lattices

Basis for optimisation of co-iteration over indices (two-way merge)

Multiple **sparse** tensor parts to iterate over within in one dimension, need lots of conditions

→ split each dimension loop into multiple

Example step:

$a = b + c$

3 Merge Lattices

Basis for optimisation of co-iteration over indices (two-way merge)

Multiple **sparse** tensor parts to iterate over within in one dimension, need lots of conditions

→ split each dimension loop into multiple

Example step:

$a = b + c$

$$\mathbf{b} + \mathbf{c} \quad \textit{iff} \quad \boldsymbol{b} \wedge \boldsymbol{c}$$

❸ Merge Lattices

Basis for optimisation of co-iteration over indices (two-way merge)

Multiple **sparse** tensor parts to iterate over within in one dimension, need lots of conditions

→ split each dimension loop into multiple

Example step:

$a = b + \mathrm{c}$

$$\begin{tikzcd}
\boxed{\mathbf{b} + \mathbf{c} \quad \textit{iff} \quad \boldsymbol{b} \wedge \boldsymbol{c}} \arrow[d, "\neg \boldsymbol{c}"'] \\
\boxed{\text{just } \mathbf{b}}
\end{tikzcd}$$

3 Merge Lattices

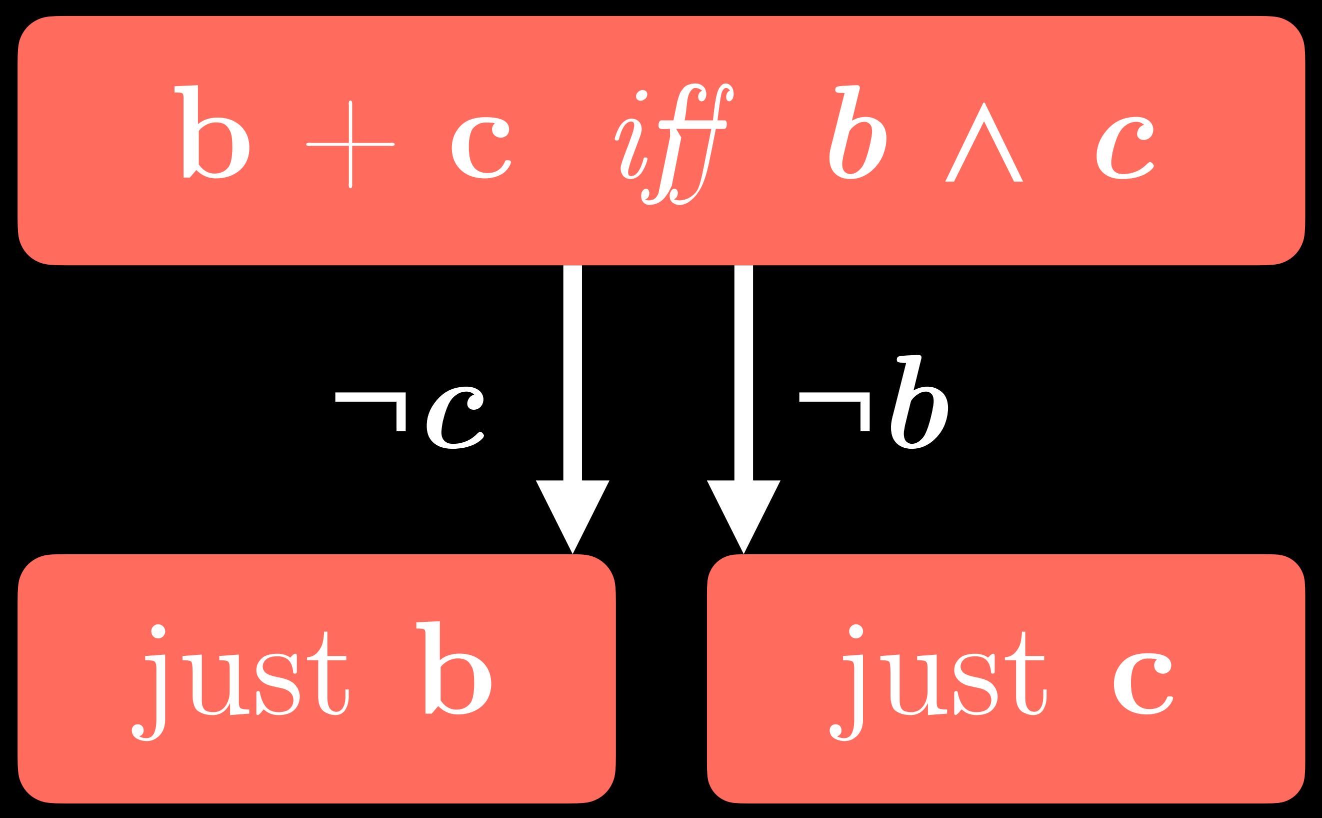
Basis for optimisation of co-iteration over indices (two-way merge)

Multiple **sparse** tensor parts to iterate over within in one dimension, need lots of conditions

→ split each dimension loop into multiple

Example step:

$a = b + c$



❸ Merge Lattices

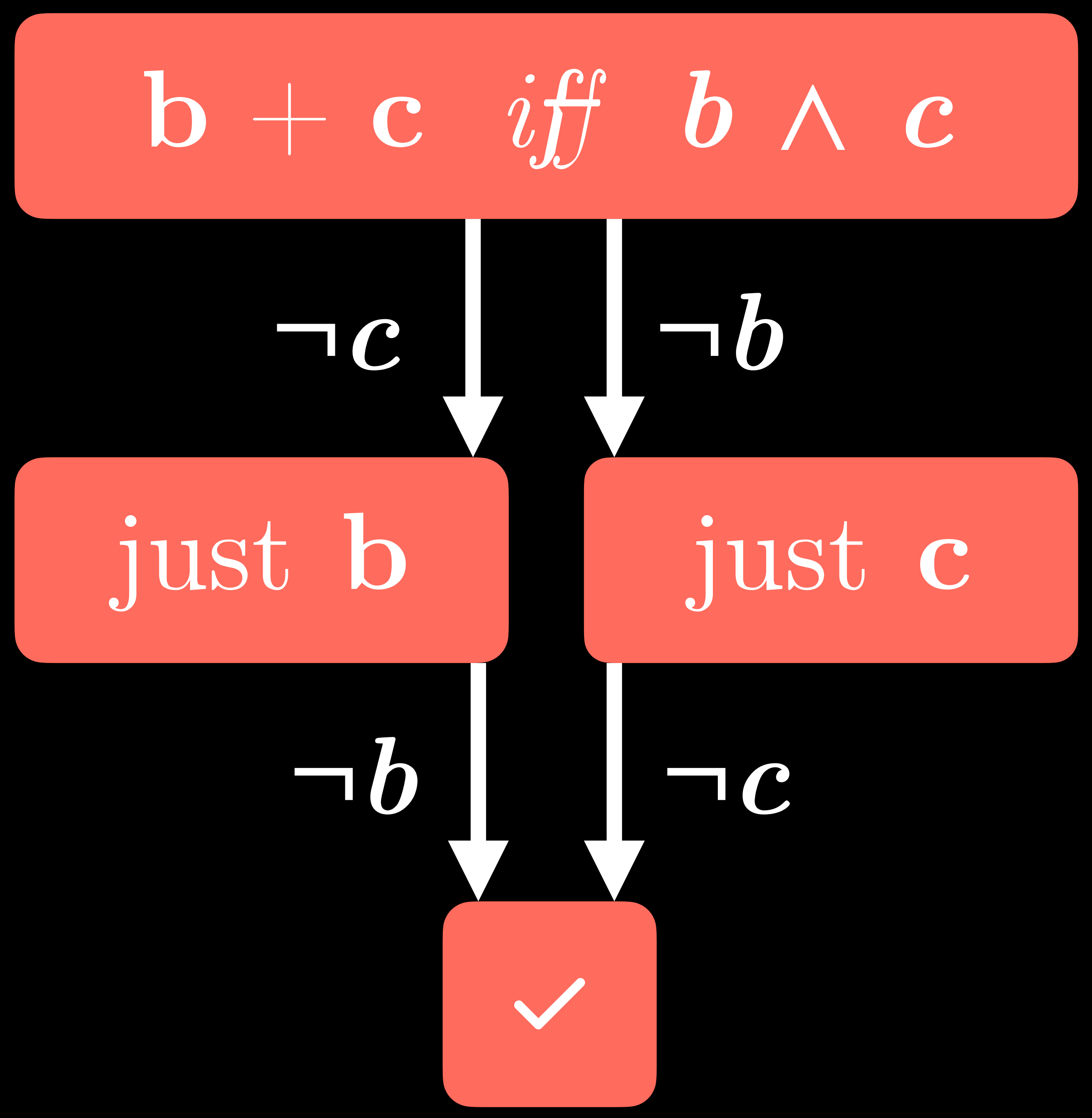
Basis for optimisation of co-iteration over indices (two-way merge)

Multiple **sparse** tensor parts to iterate over within in one dimension, need lots of conditions

→ split each dimension loop into multiple

Example step:

$a = b + c$



3 Merge Lattices

Basis for optimisation of co-iteration over indices (two-way merge)

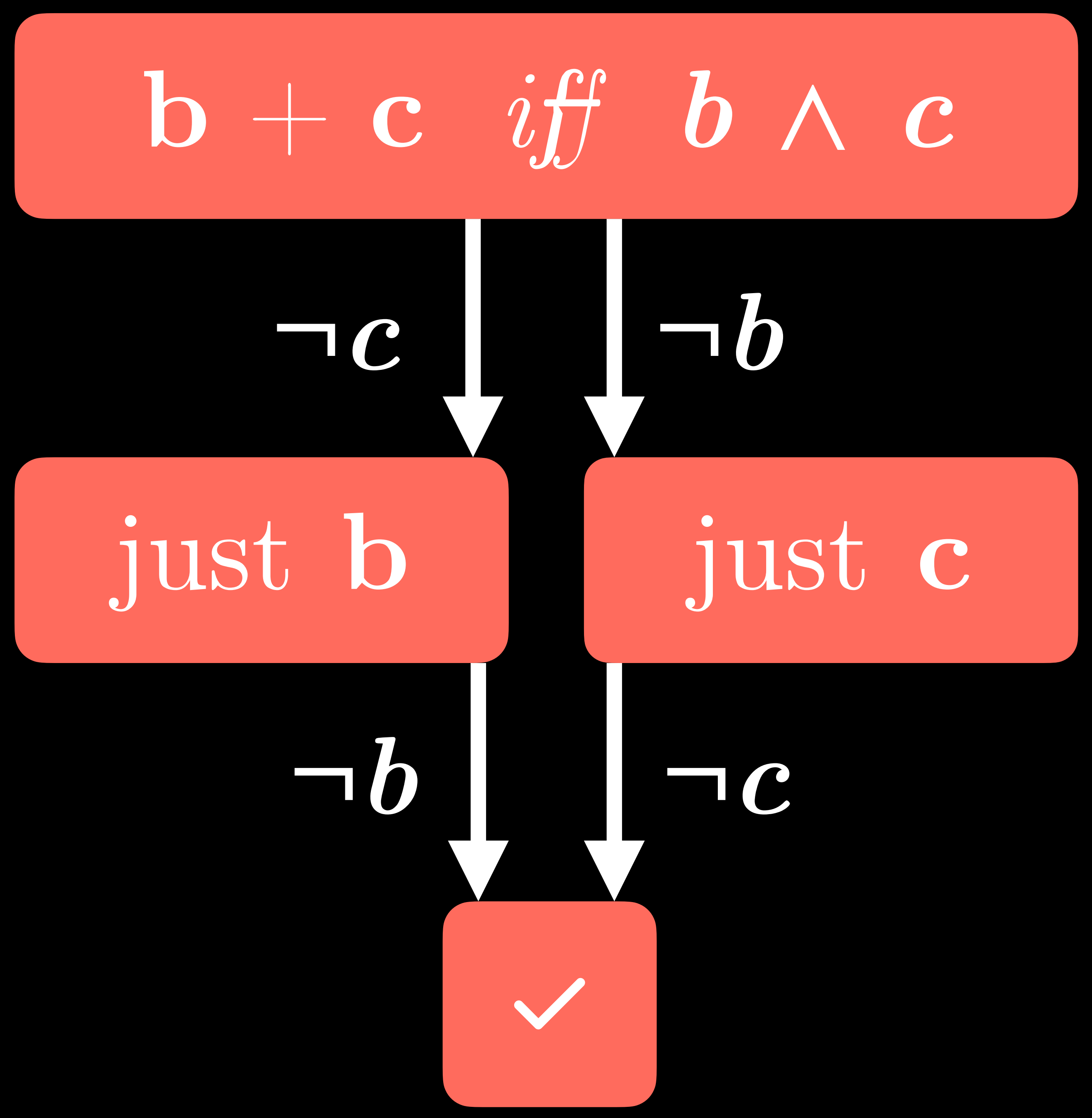
Multiple **sparse** tensor parts to iterate over within in one dimension, need lots of conditions

→ split each dimension loop into multiple

→ Optimise by eliminating points/loops

Example step:

$a = b + c$



1 2 3

Evaluation

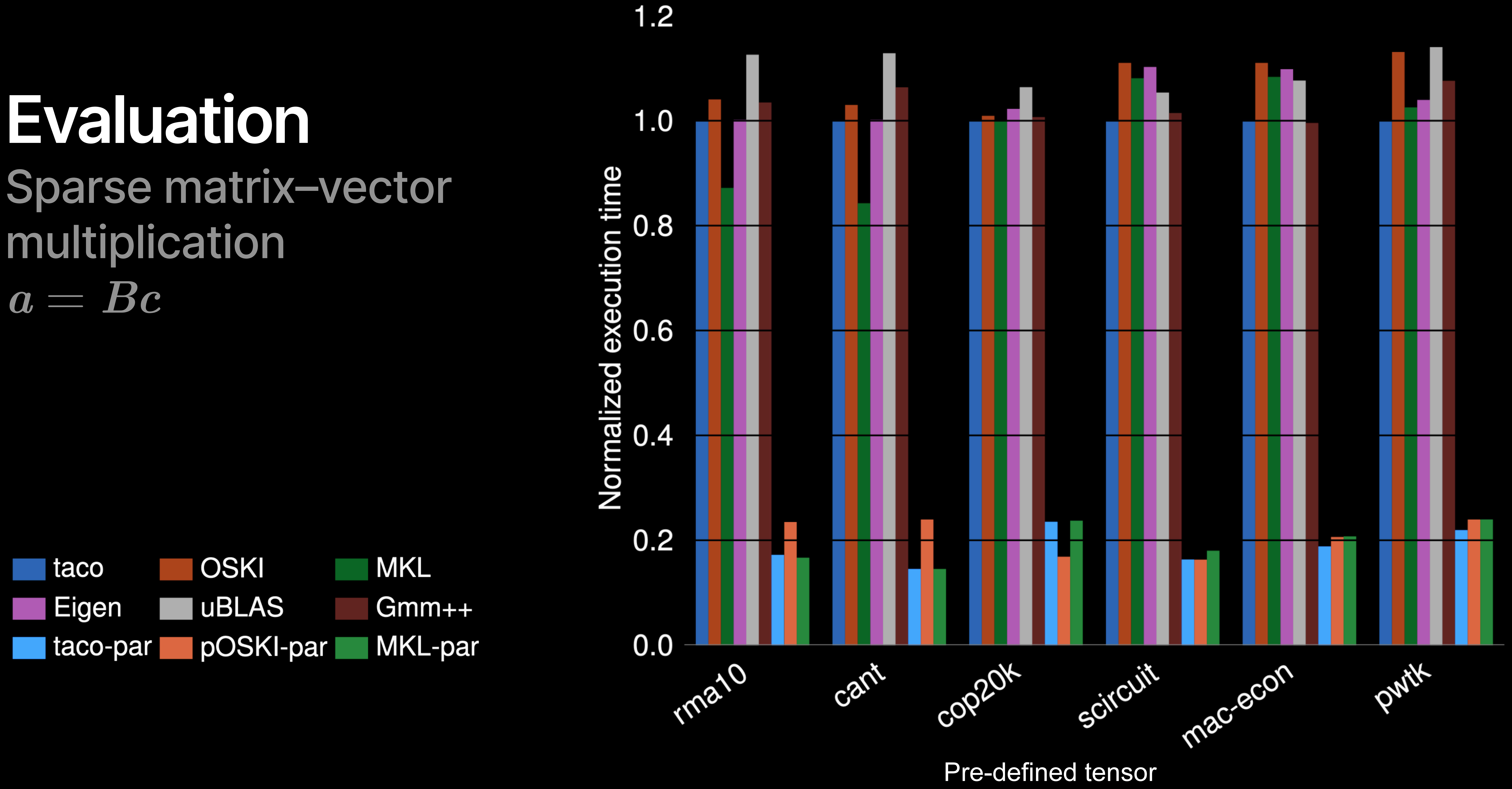
Sparse matrix–vector multiplication

$a = Bc$

Evaluation

Sparse matrix–vector multiplication

$a = Bc$



Evaluation

Evaluation

- Can compete with manually written kernels

Evaluation

- Can compete with manually written kernels
- Can reduce number of mathematical operations

Evaluation

- Can compete with manually written kernels
- Can reduce number of mathematical operations
- Can directly emit tensor without dense→sparse conversion

Evaluation

- Can compete with manually written kernels
- Can reduce number of mathematical operations
- Can directly emit tensor without dense→sparse conversion
- Further measurements: no ideal tensor format

Conclusion and Thoughts

Conclusion and Thoughts

- General compiler for algebraic tensor expressions

Conclusion and Thoughts

- General compiler for algebraic tensor expressions
- Abstractions → flexibility + versatility (storage, iteration graphs, merge lattices)

Conclusion and Thoughts

- General compiler for algebraic tensor expressions
- Abstractions → flexibility + versatility (storage, iteration graphs, merge lattices)
- No performance-generality tradeoff

Conclusion and Thoughts

- General compiler for algebraic tensor expressions
- Abstractions → flexibility + versatility (storage, iteration graphs, merge lattices)
- No performance-generality tradeoff
- FOSS software

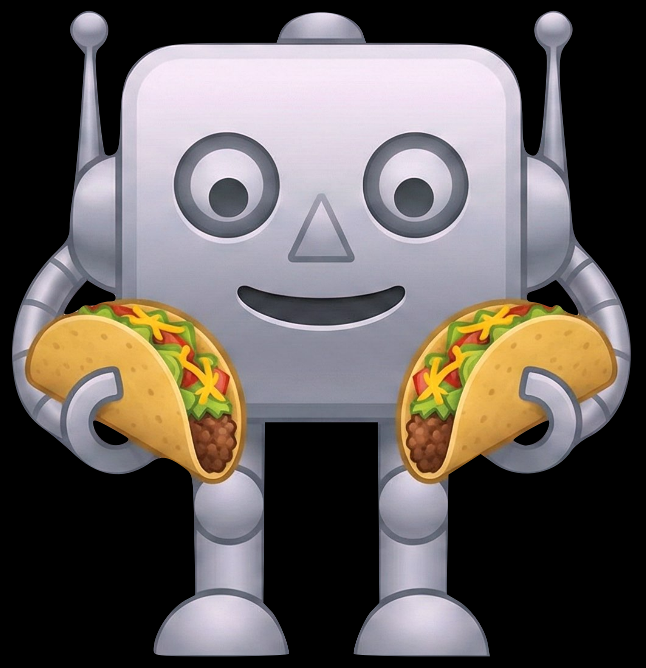
Conclusion and Thoughts

- General compiler for algebraic tensor expressions
- Abstractions → flexibility + versatility (storage, iteration graphs, merge lattices)
- No performance-generality tradeoff
- FOSS software
- Limitations
  parallel execution mode, (transpose/format*), conversion when merging;
  (no algebraic transformations)

Conclusion and Thoughts

- General compiler for algebraic tensor expressions
- Abstractions → flexibility + versatility (storage, iteration graphs, merge lattices)
- No performance-generality tradeoff
- FOSS software
- Limitations
  parallel execution mode, (transpose/format*), conversion when merging;
  (no algebraic transformations)
- Future work
  dimension formats (crd*), /temporary dense workspaces*[1]), JIT

[1] F. Kjolstad, W. Ahrens, S. Kamil and S. Amarasinghe, "Tensor Algebra Compilation with Workspaces," 2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), Washington, DC, USA, 2019, pp. 180-192, doi: 10.1109/CGO.2019.8661185.



UNIVERSITY OF CAMBRIDGE

TACO

The Tensor Algebra Compiler

Fredrik Kjolstad, Shoaib Kamil, Stephen Chou, David Lugato, and Saman Amarasinghe

Carl Seifert
cs2331@cam.ac.uk

Font: rsms.me/inter. Symbols: fonts.google.com/icons. Robot image has been generated by Nano Banana Pro.