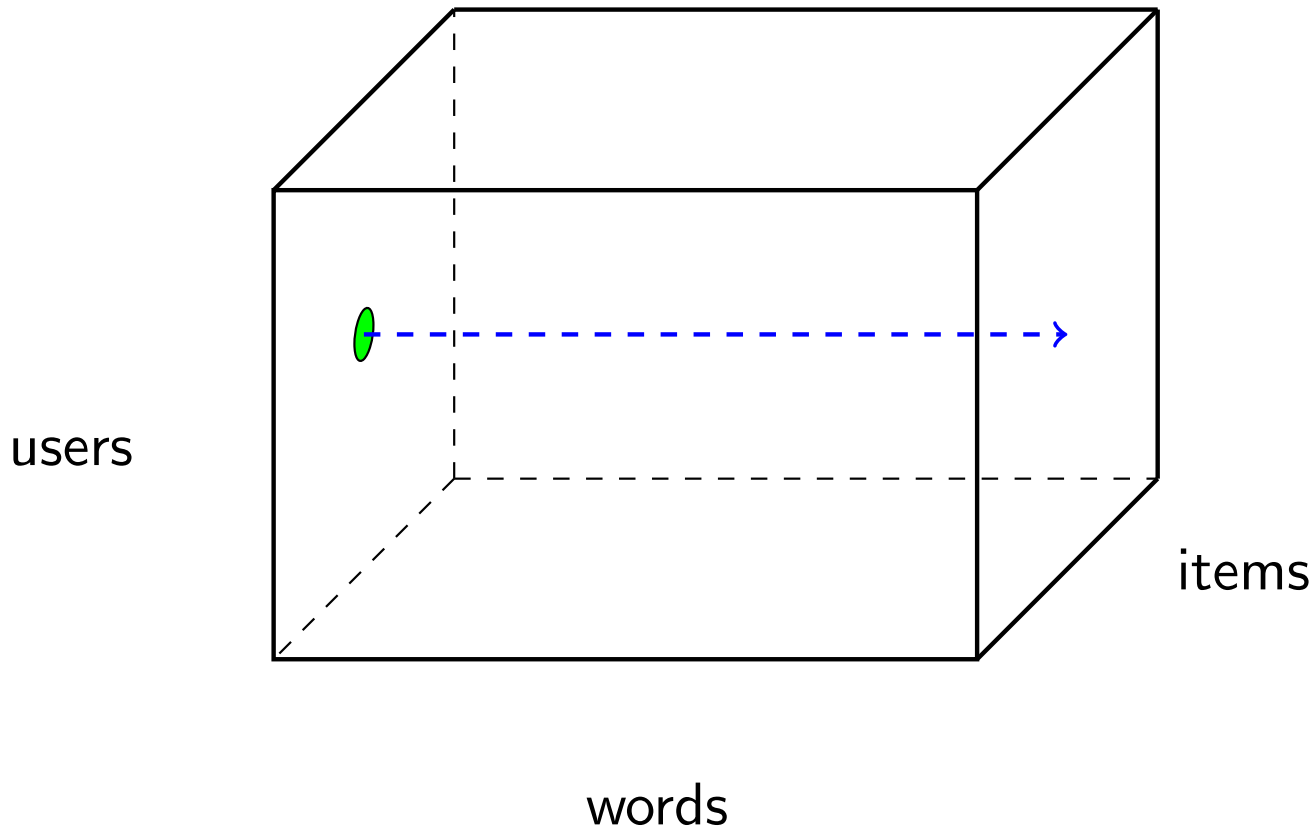
Tensor-Matrix Products with a Compressed Sparse Tensor

Shaden Smith George Karypis

University of Minnesota
Department of Computer Science & Engineering
shaden@cs.umn.edu

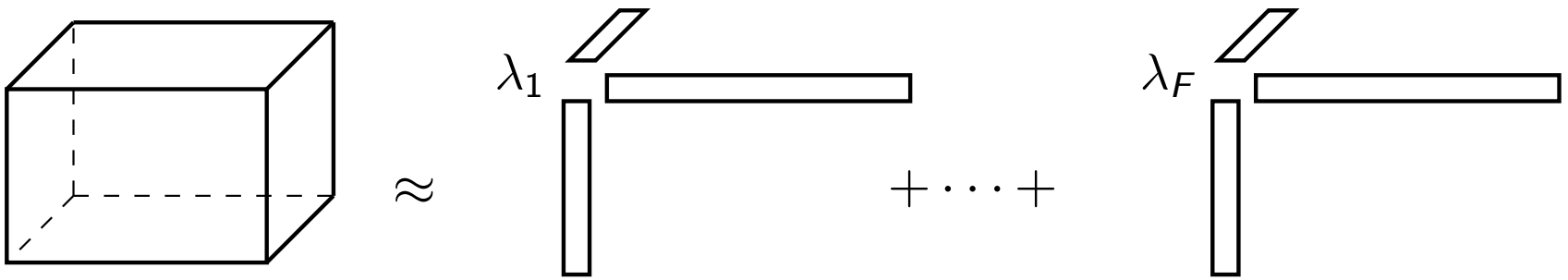
Tensor Introduction

- Tensors are the generalization of matrices to $\geq 3D$
- Tensors have m dimensions (or *modes*) and are $I_1 \times \ldots \times I_m$.



Canonical Polyadic Decomposition (CPD)

- The CPD is an extension of the SVD to tensors
- We compute matrices $\mathbf{A}_1, \ldots, \mathbf{A}_m$, each with F columns and λ, a vector of weights



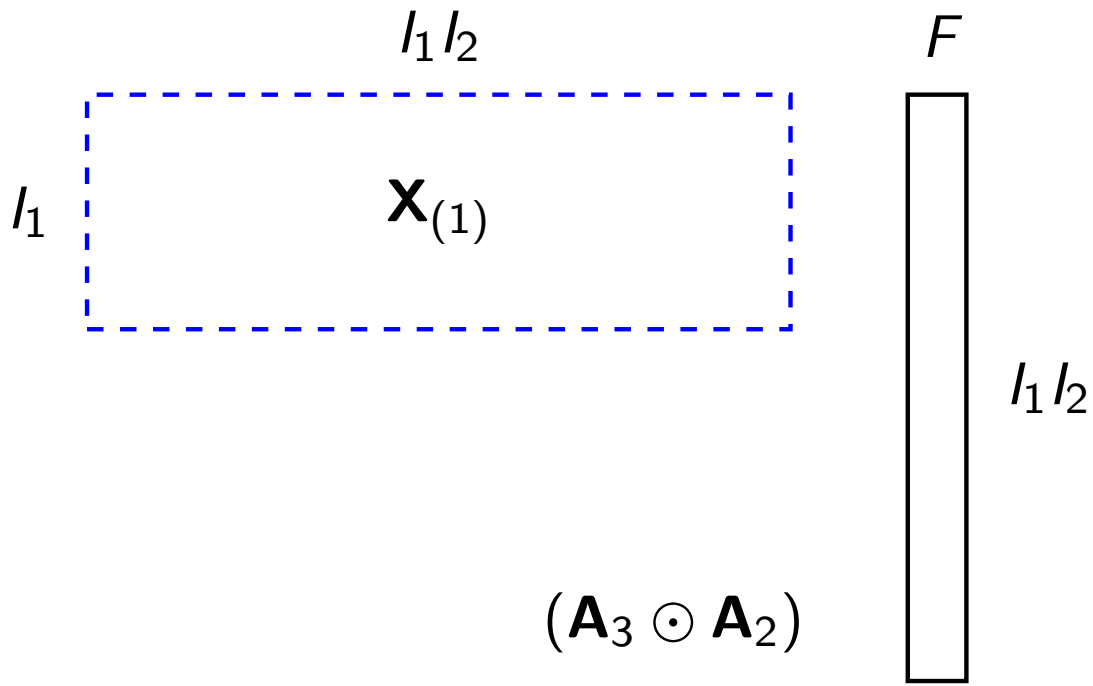
- Usually computed via *alternating least squares* (ALS)

MTTKRP

Matricized Tensor Times Khatri-Rao Product (MTTKRP)

- MTTKRP is the core computation of each iteration

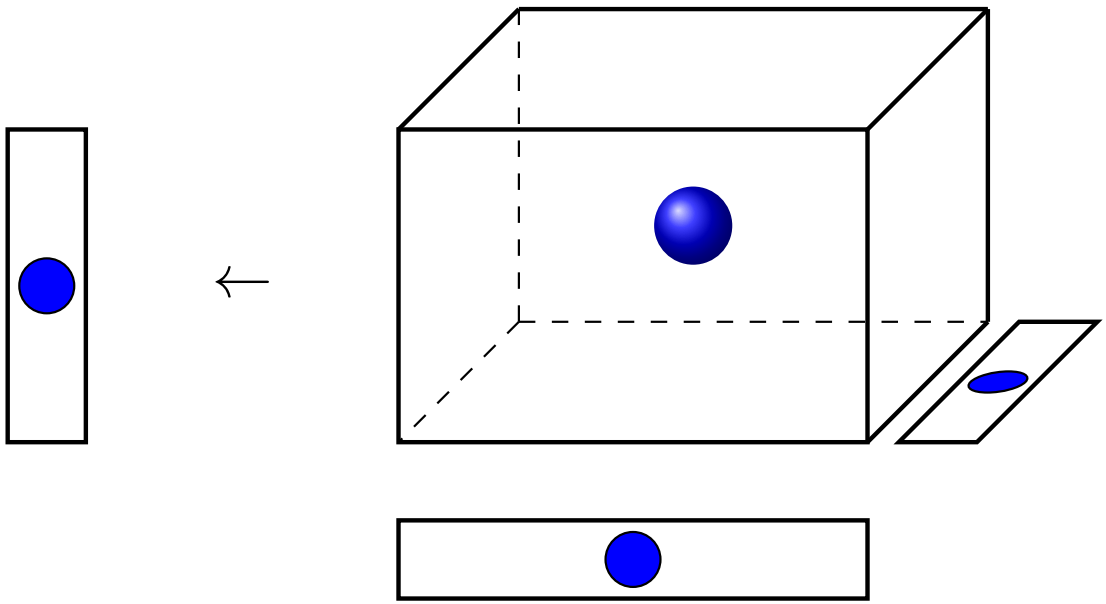
$$\mathbf{A}_1 = \mathbf{X}_{(1)} \left(\mathbf{A}_m \odot \cdots \odot \mathbf{A}_2 \right)$$



Related Work

Uncompressed Tensors

- Stored as a list of *coordinates*
- $(i, j, k) = v$ represents one nonzero

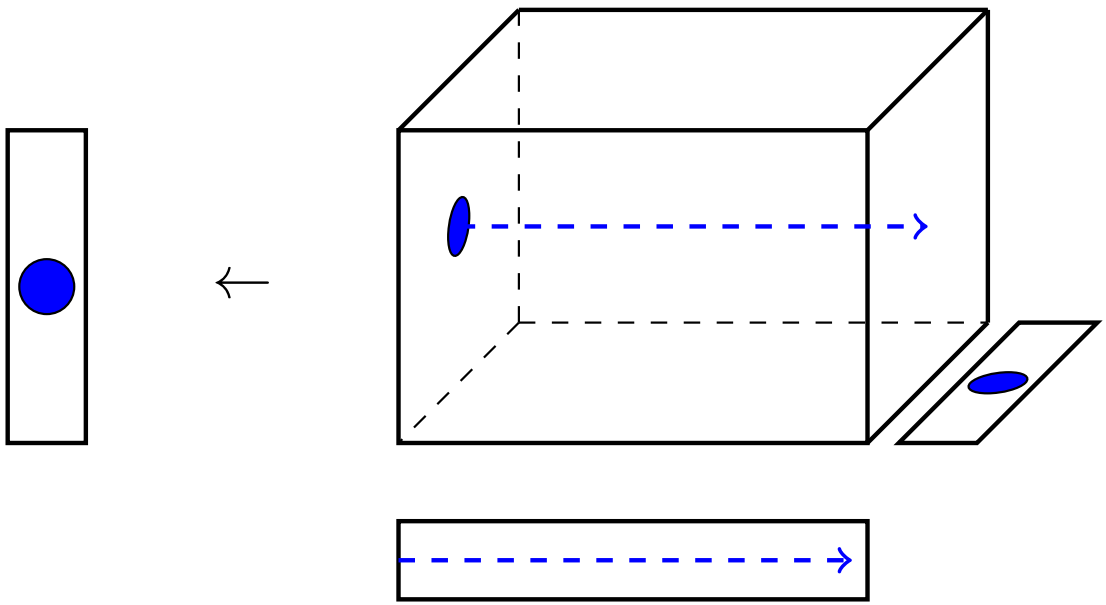


$$\mathbf{A}_1(i,:) \leftarrow \mathbf{A}_1(i,:) + \mathcal{X}(i,j,k)\,[\mathbf{A}_2(j,:) * \mathbf{A}_3(k,:)]$$

Compressed Tensors

SPLATT

- SPLATT uses a hierarchical storage scheme for 3D tensors
- This allows for operation reduction and coarse-grained parallelism

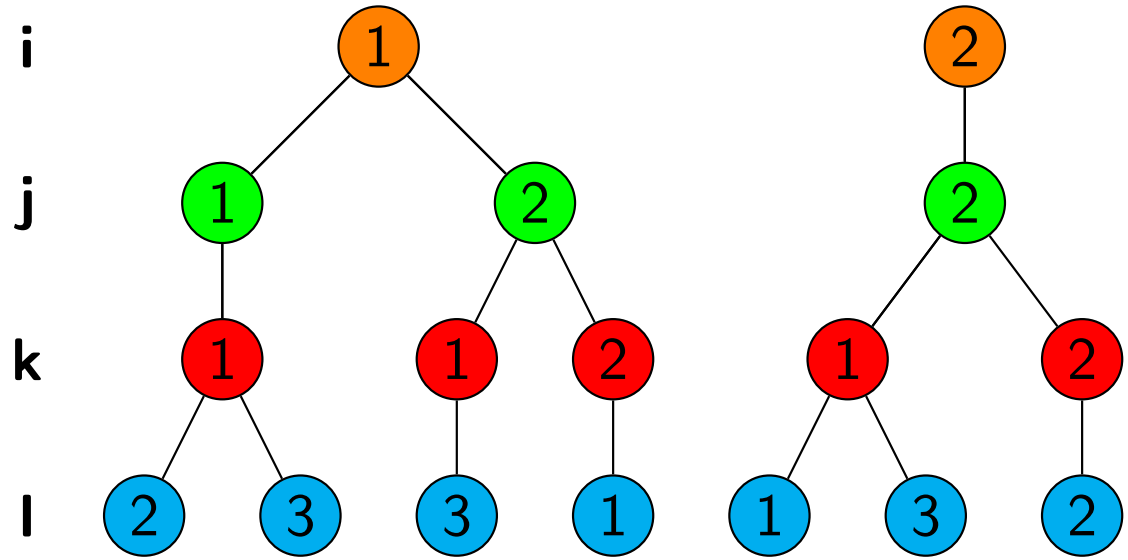


Contributions

Compressed Sparse Fiber (CSF)

i	j	k	l
1	1	1	2
1	1	1	3
1	2	1	3
1	2	2	1
2	2	1	1
2	2	1	3
2	2	2	2

→



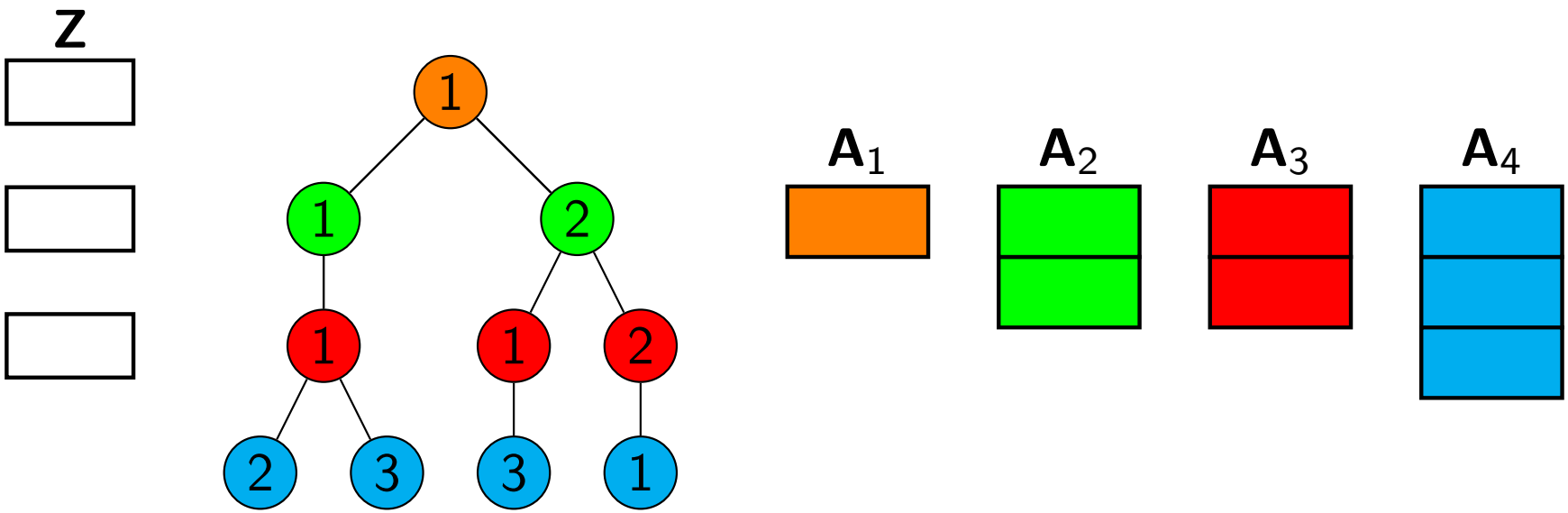
MTTKRP with a CSF Tensor

Objective

- We want to perform MTTKRP on each tensor mode with only one CSF representation
- There are three types of nodes in a tree: *root*, *internal*, and *leaf*
  - Each will have a tailored algorithm

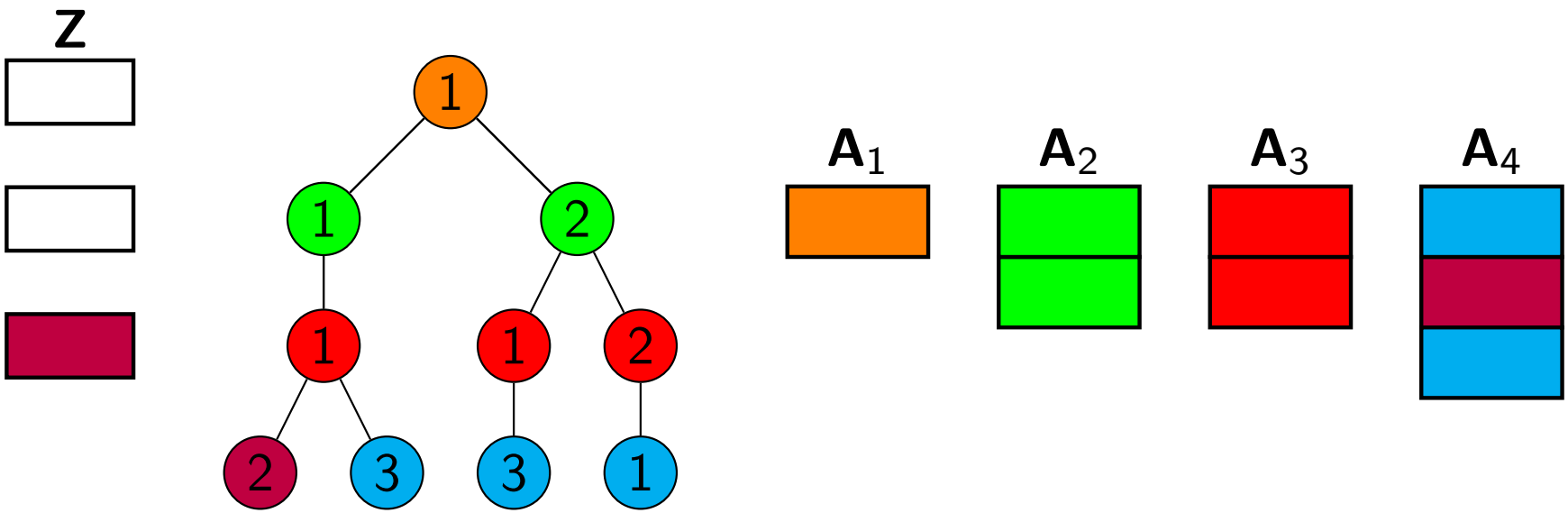
CSF-ROOT

- We do a depth-first traversal on the CSF structure



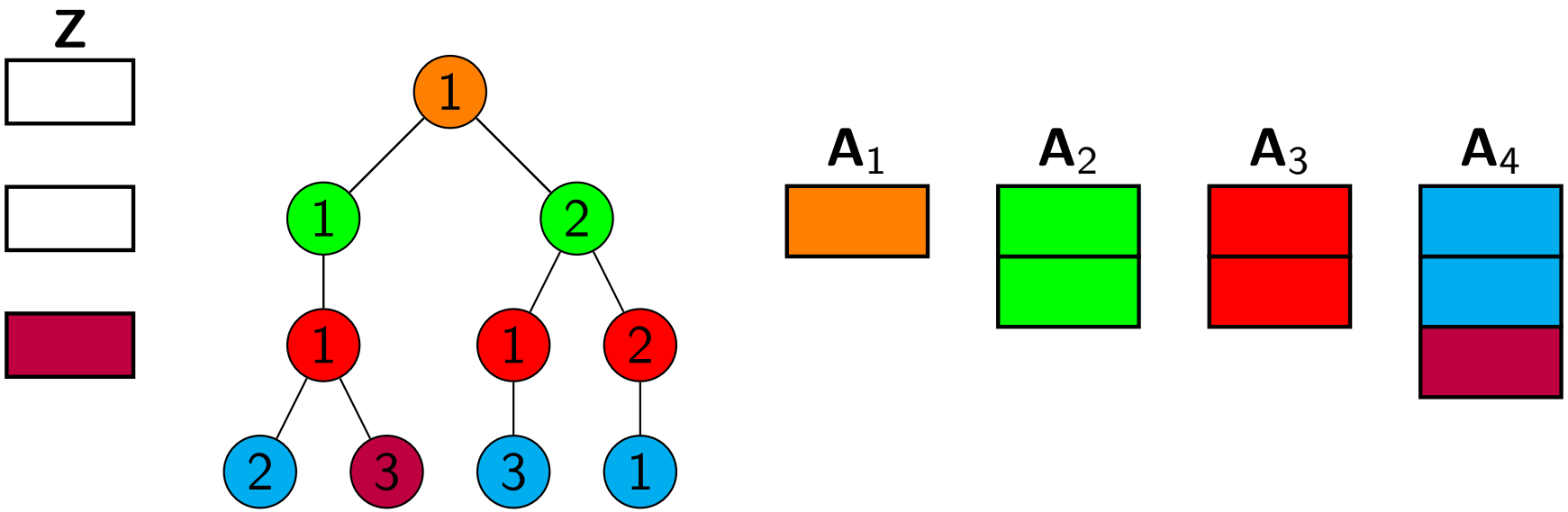
CSF-ROOT

- Inner products are accumulated in a buffer



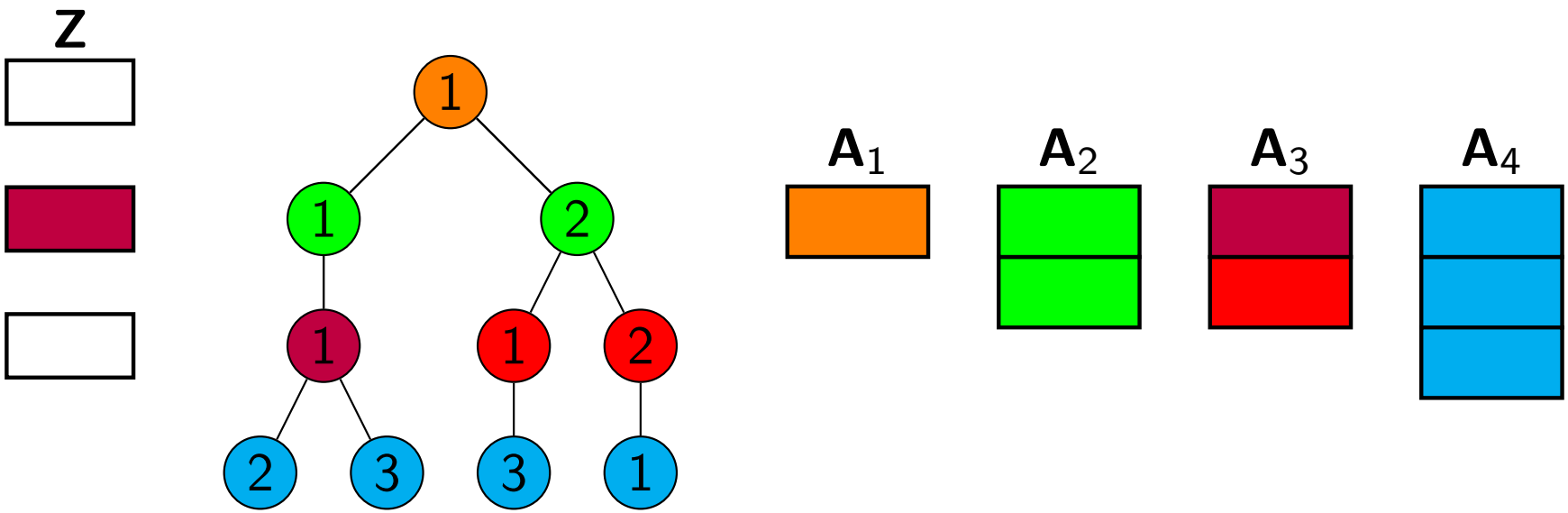
CSF-ROOT

- Inner products are accumulated in a buffer



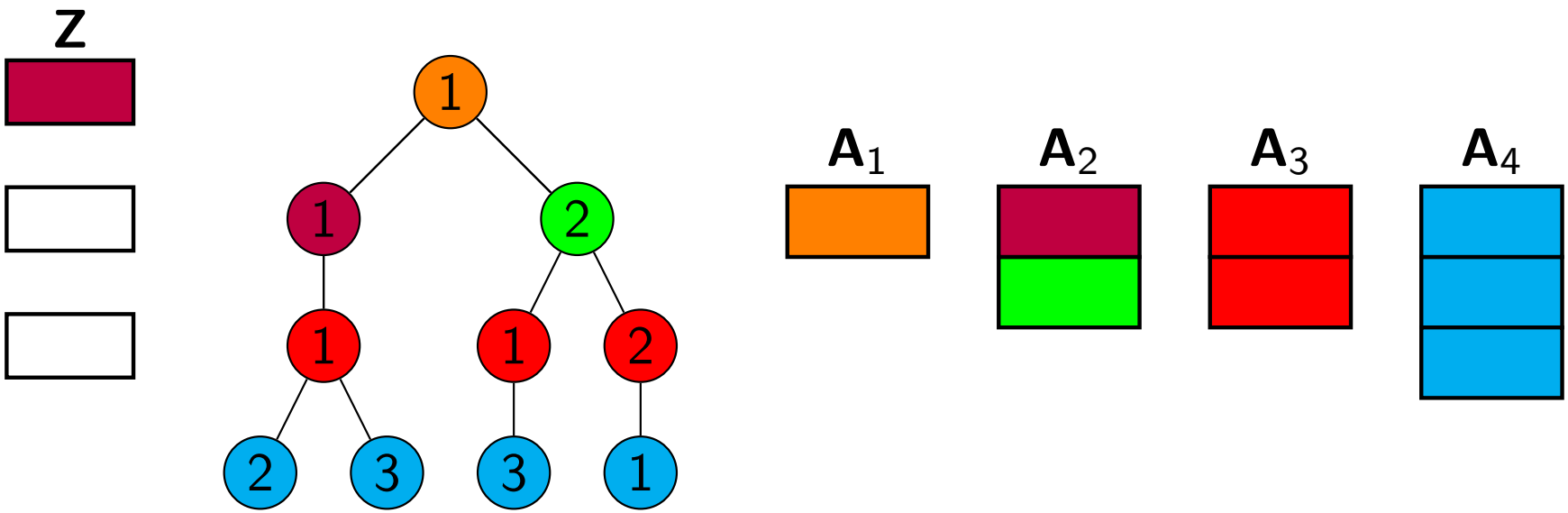
CSF-ROOT

- Hadamard products are then propagated up the CSF tree



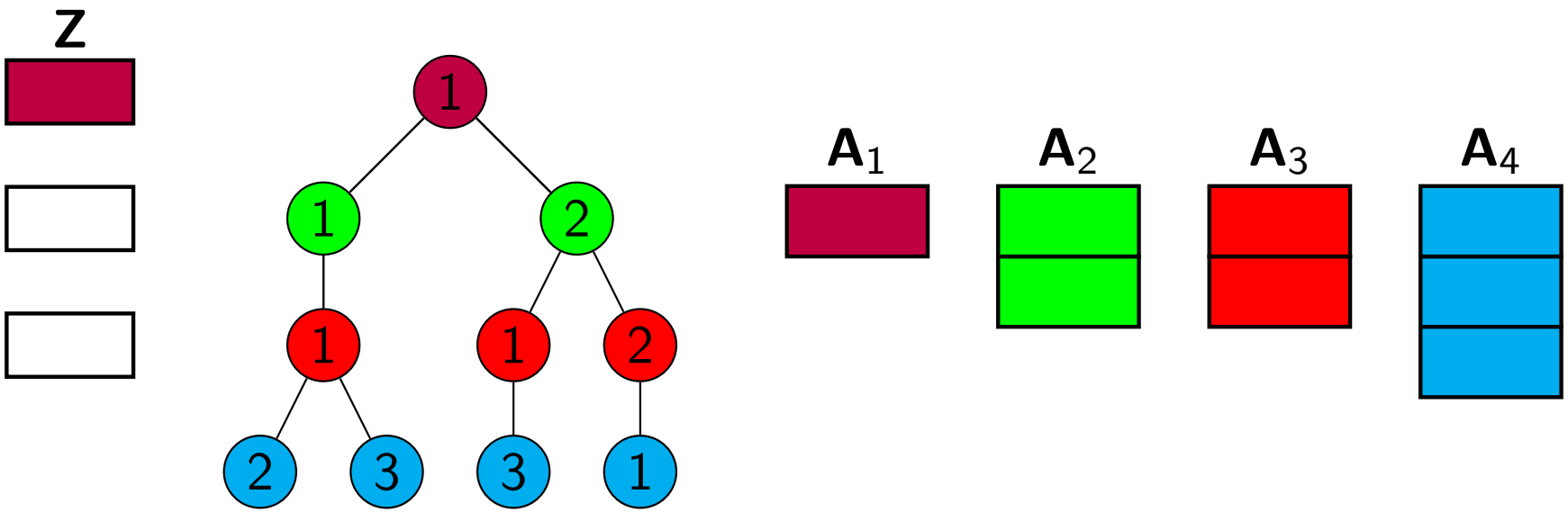
CSF-ROOT

- Hadamard products are then propagated up the CSF tree



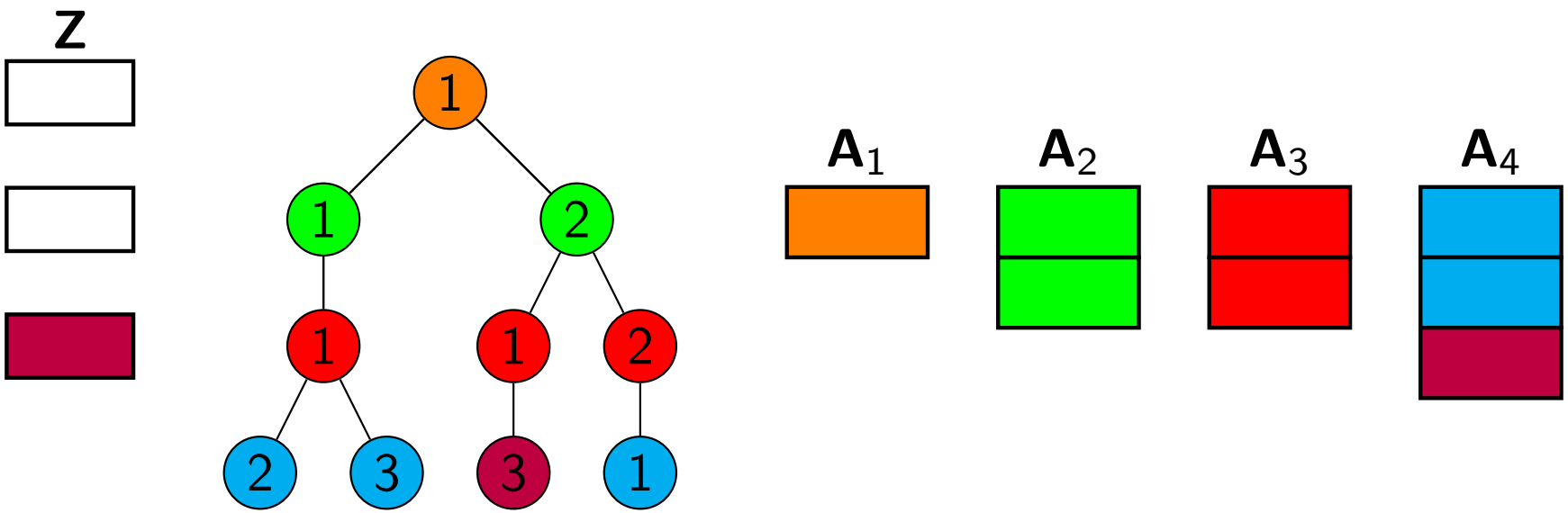
CSF-ROOT

- Results are accumulated when we reach the top



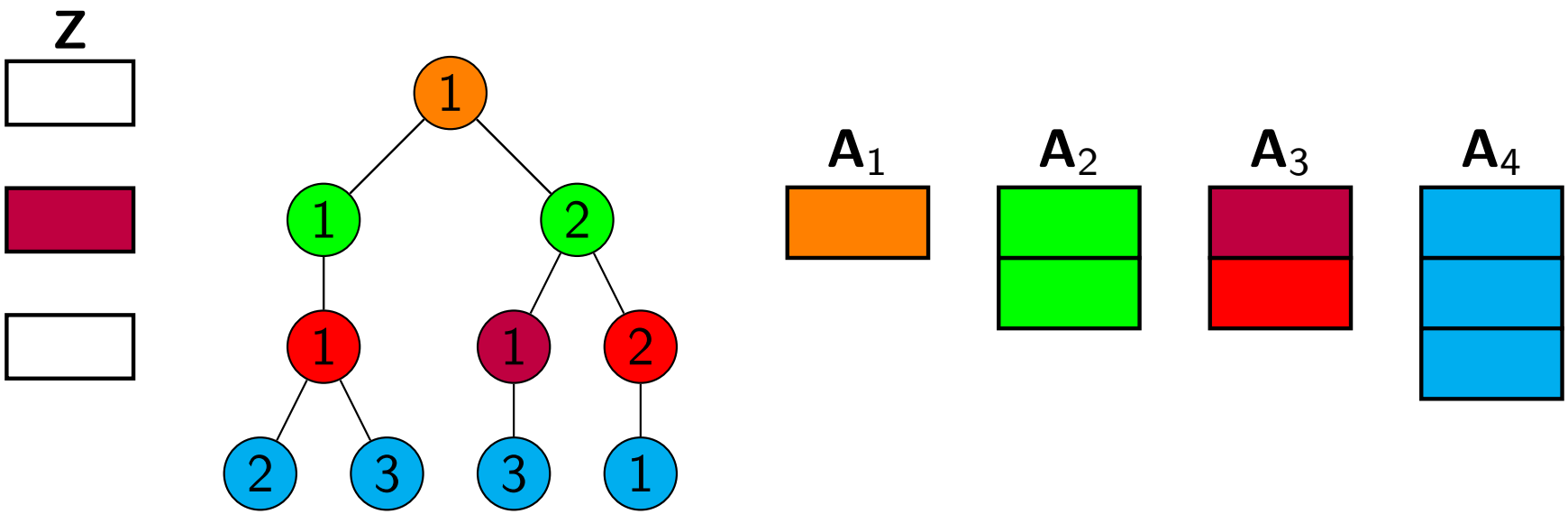
CSF-ROOT

- The traversal continues...



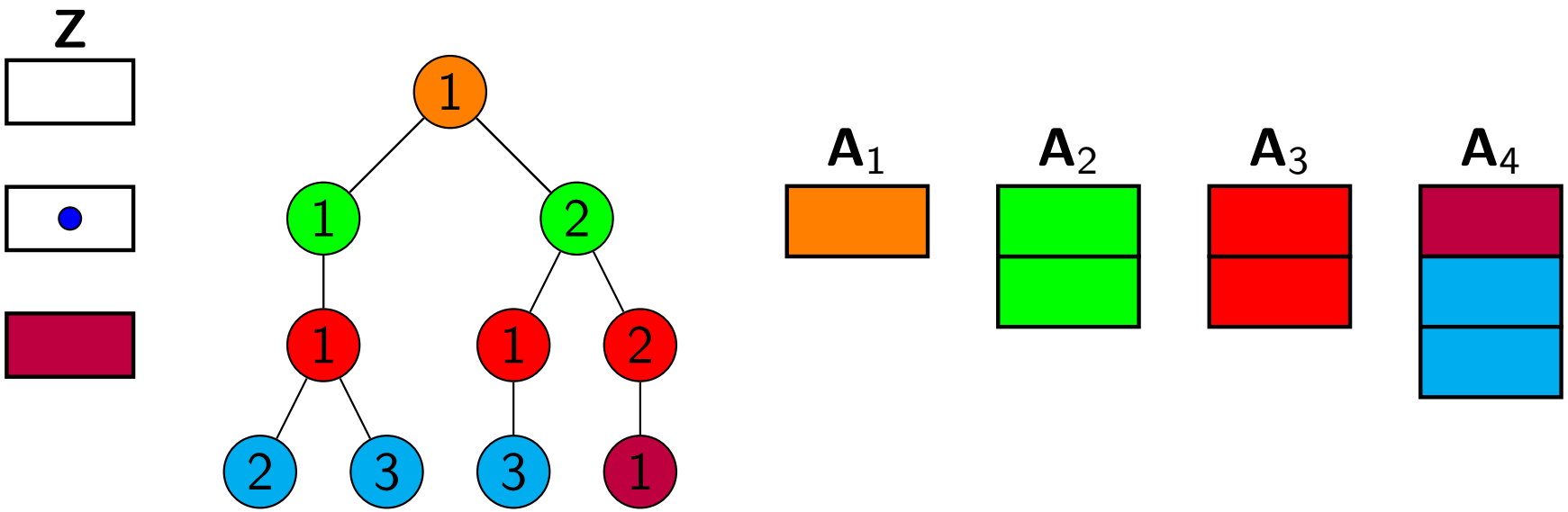
CSF-ROOT

- The traversal continues...



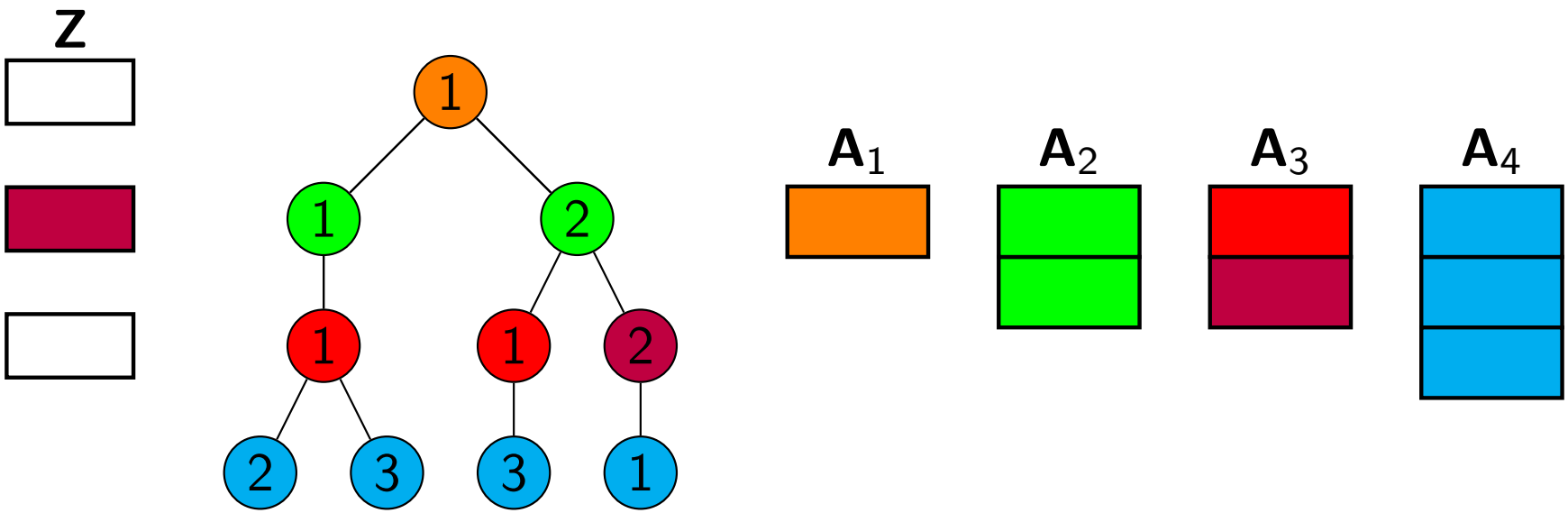
CSF-ROOT

- Partial results are kept in buffer



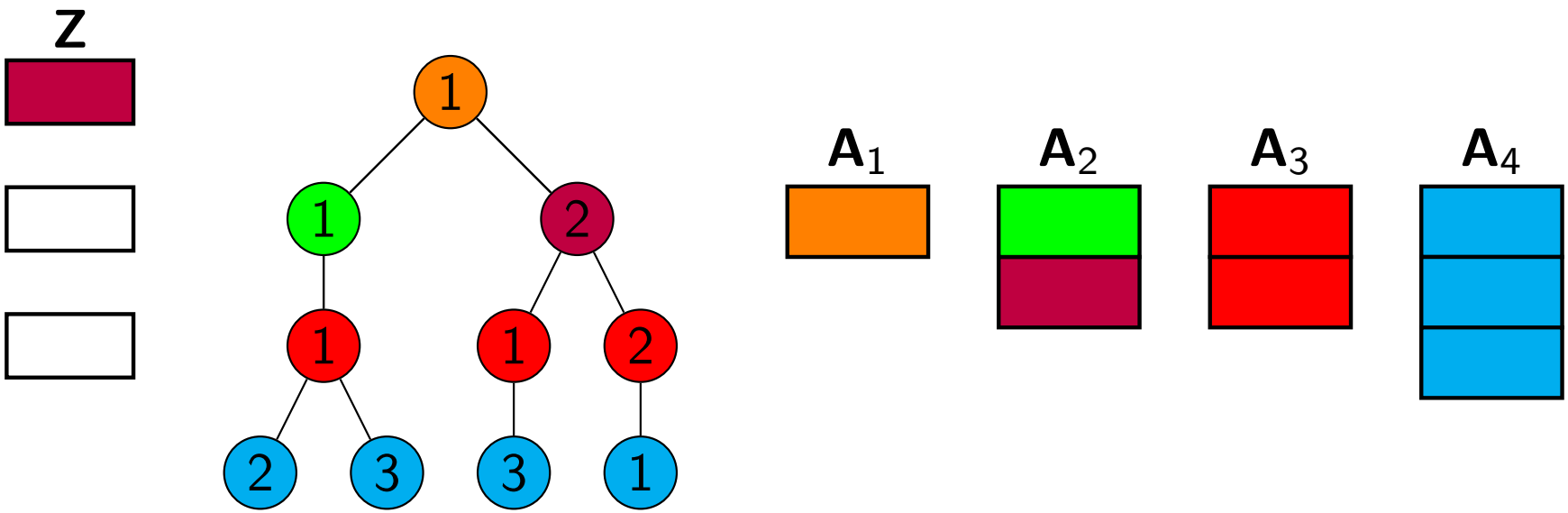
CSF-ROOT

- Inner products are accumulated in a buffer



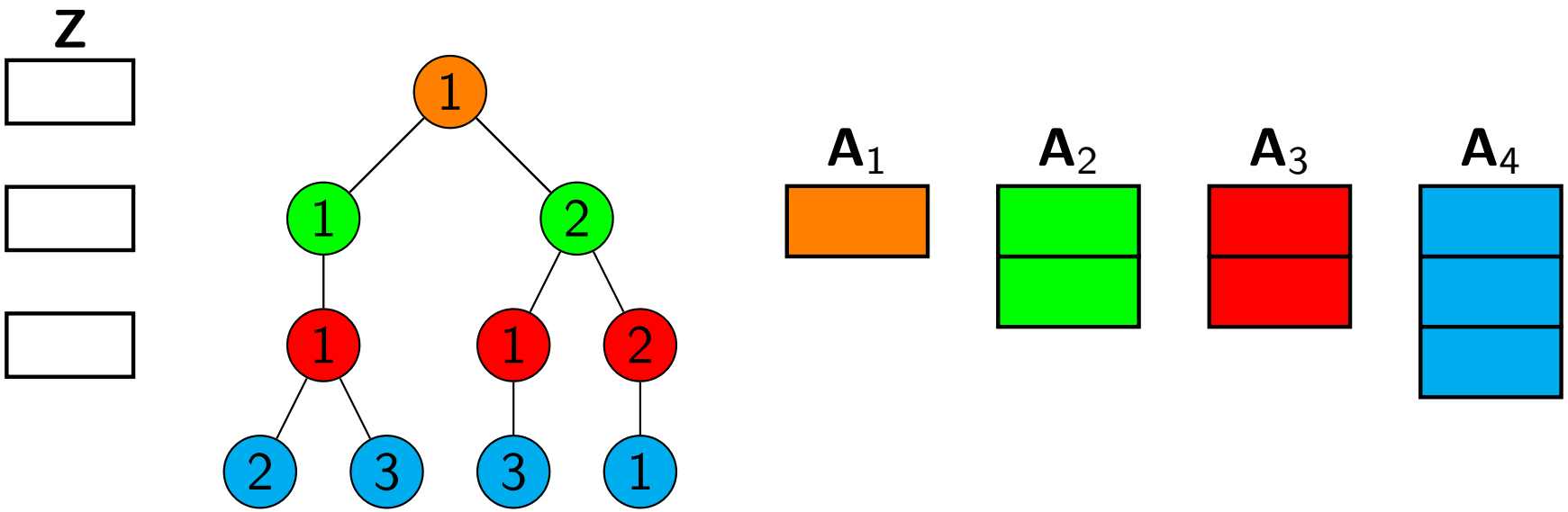
CSF-ROOT

- Inner products are accumulated in a buffer



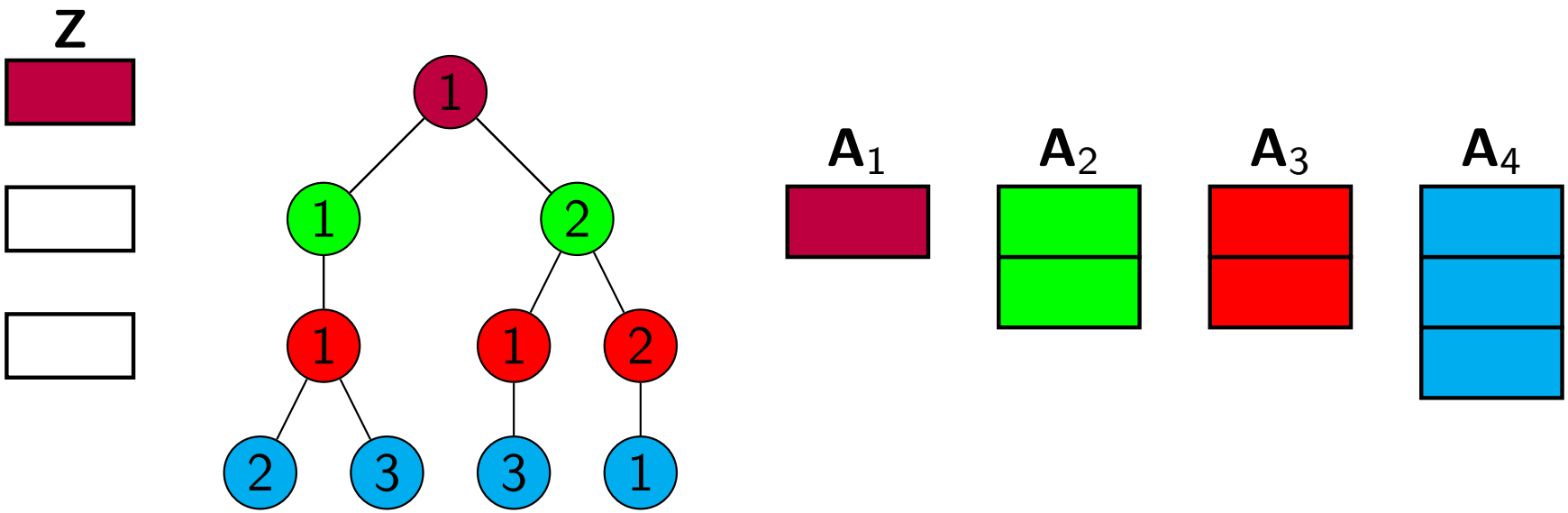
CSF-LEAF

- This time, Hadamard products are pushed *down* the tree



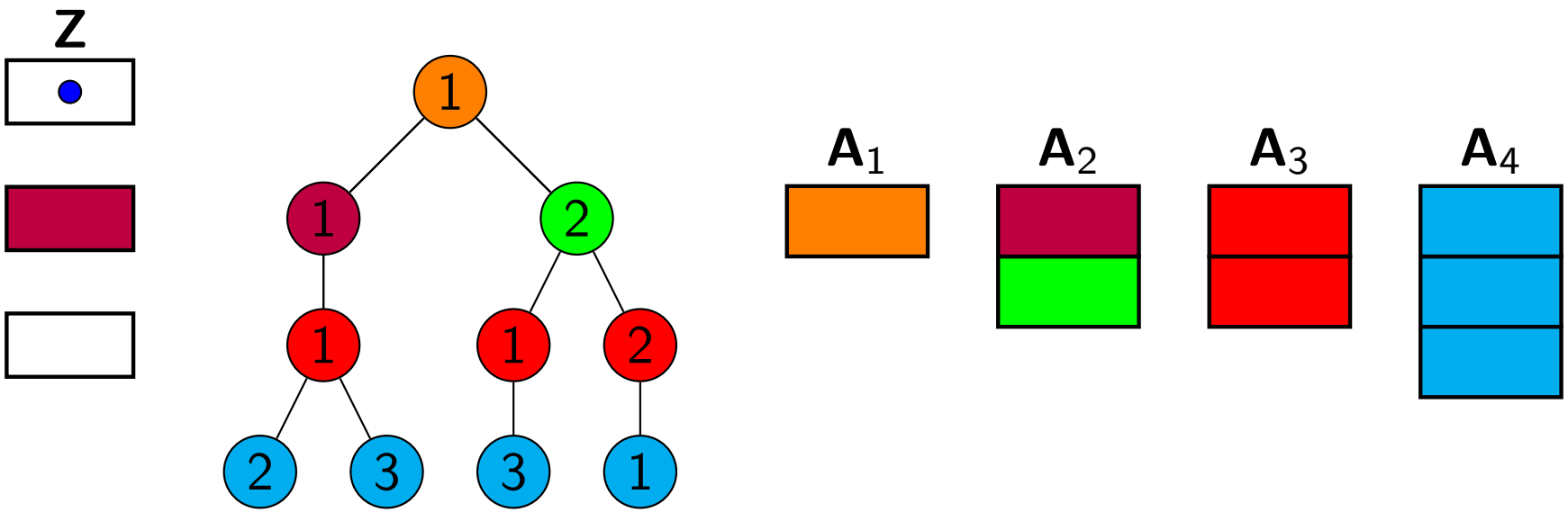
CSF-LEAF

- This time, Hadamard products are pushed *down* the tree



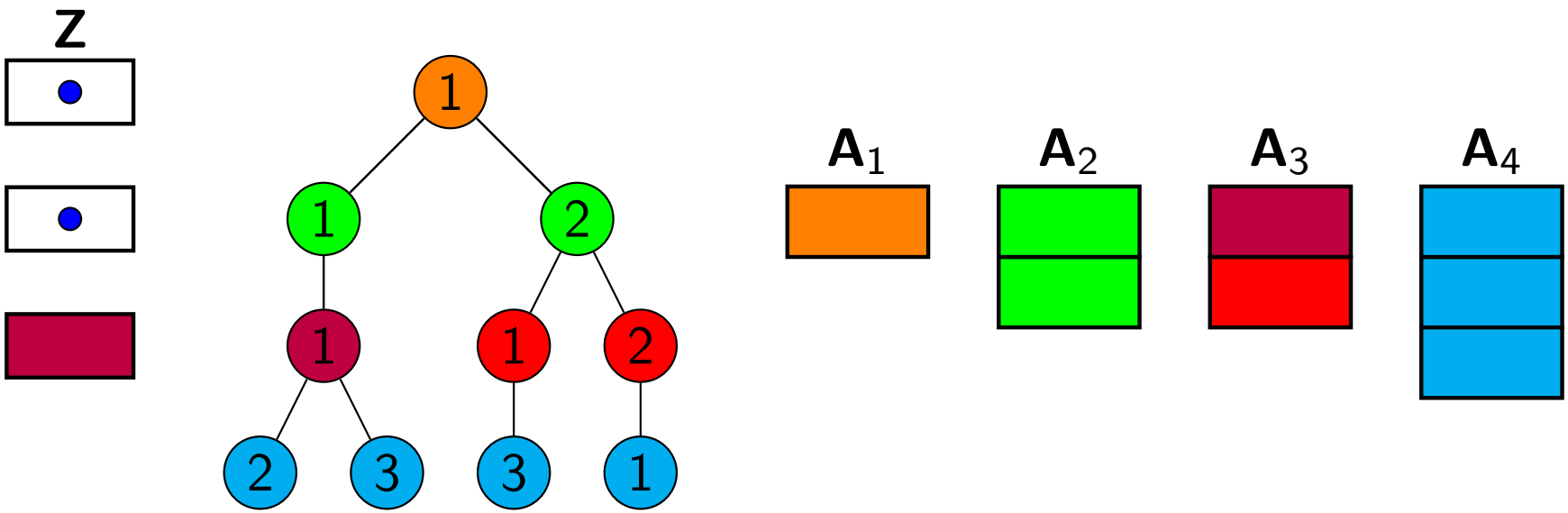
CSF-LEAF

- This time, Hadamard products are pushed *down* the tree



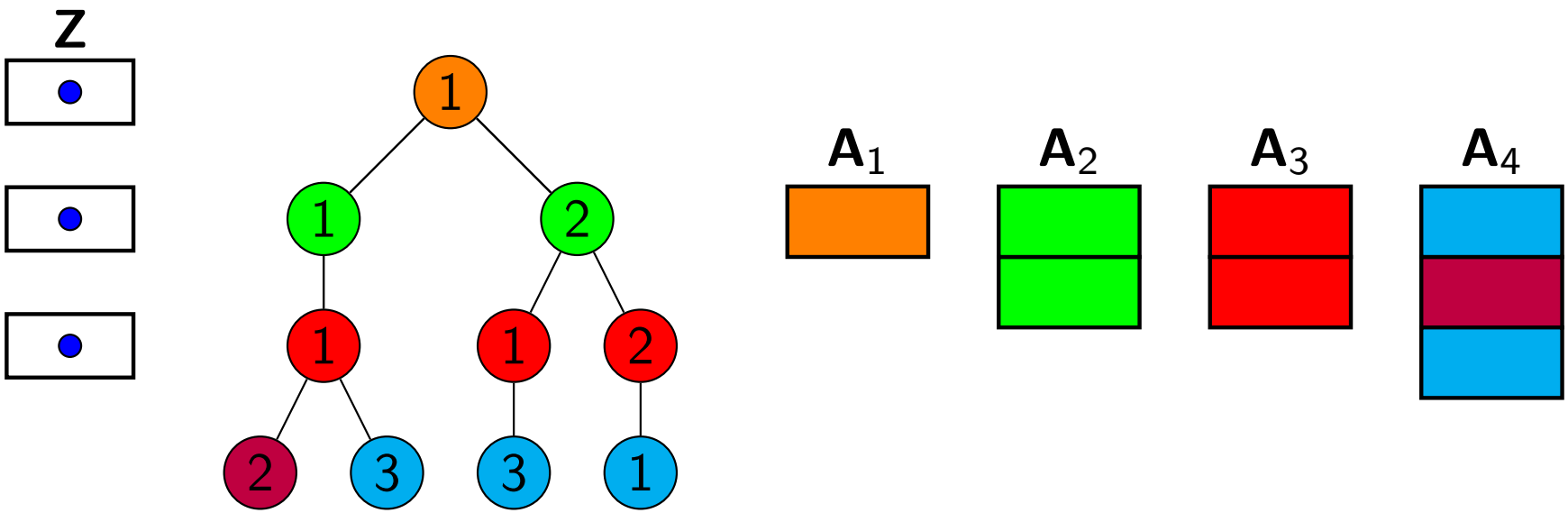
CSF-LEAF

- This time, Hadamard products are pushed *down* the tree



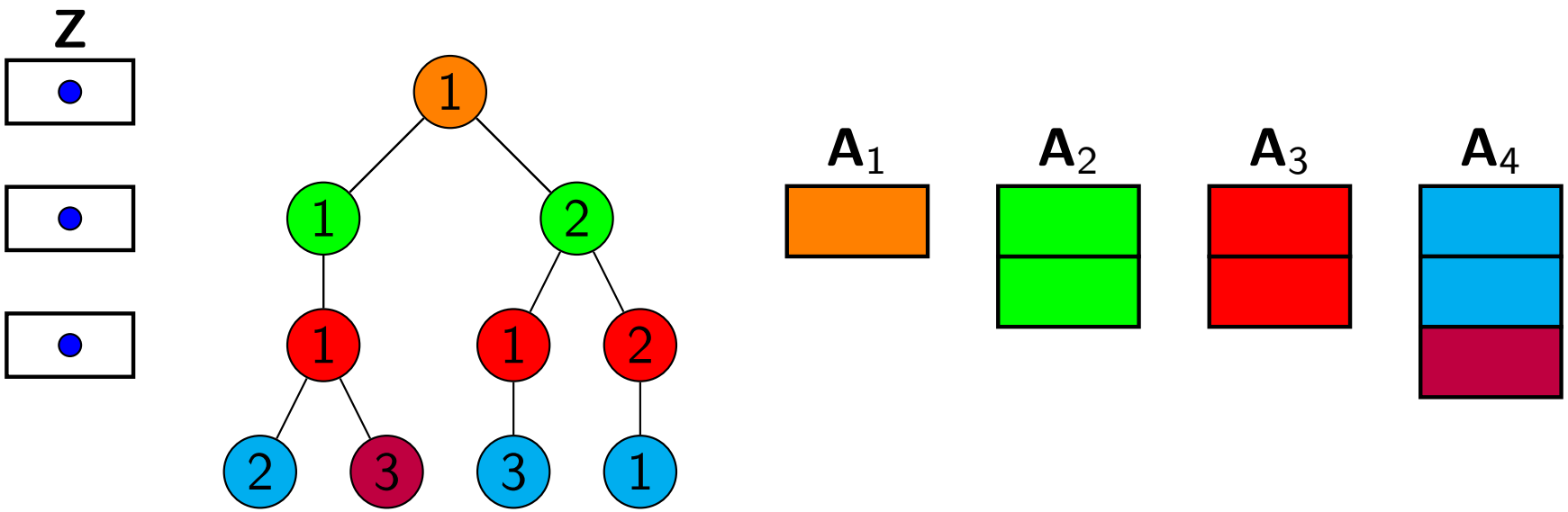
CSF-LEAF

- Leaves designate write locations



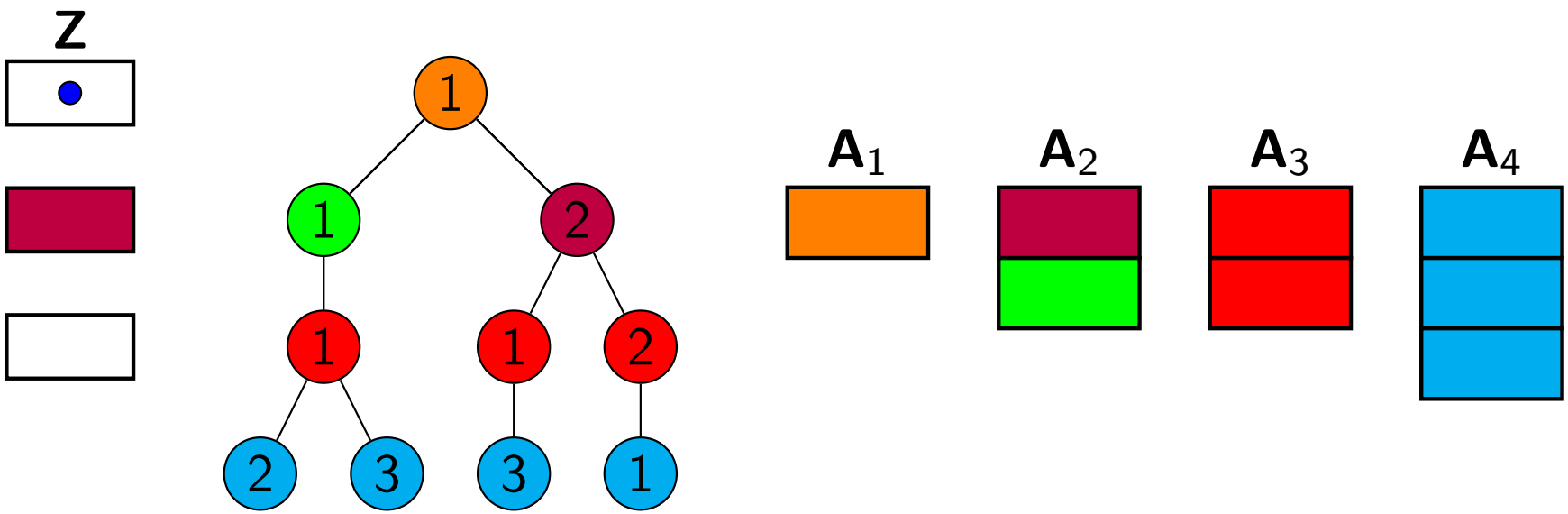
CSF-LEAF

- Leaves designate write locations



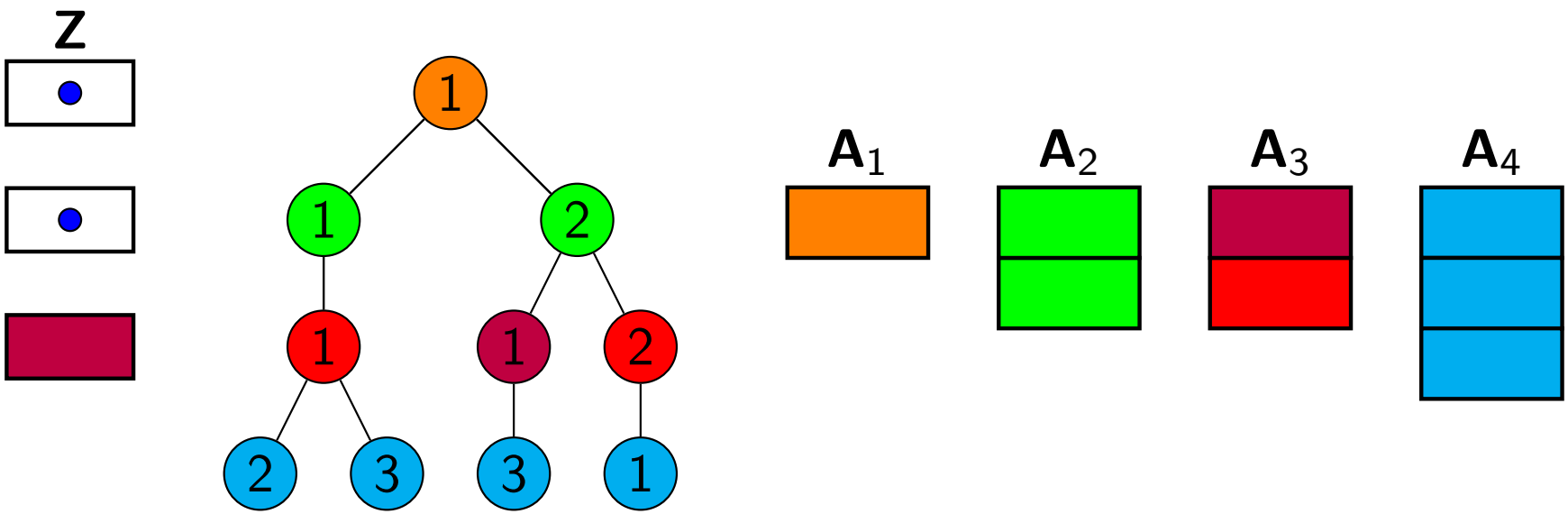
CSF-LEAF

- The traversal continues...



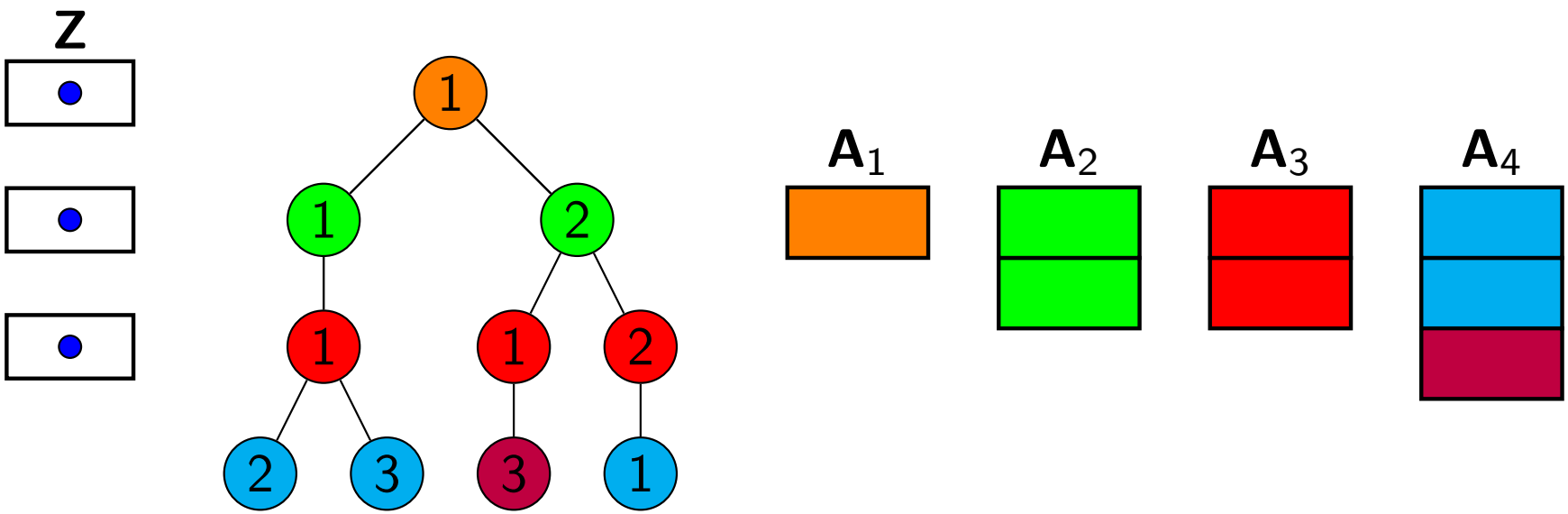
CSF-LEAF

- The traversal continues...



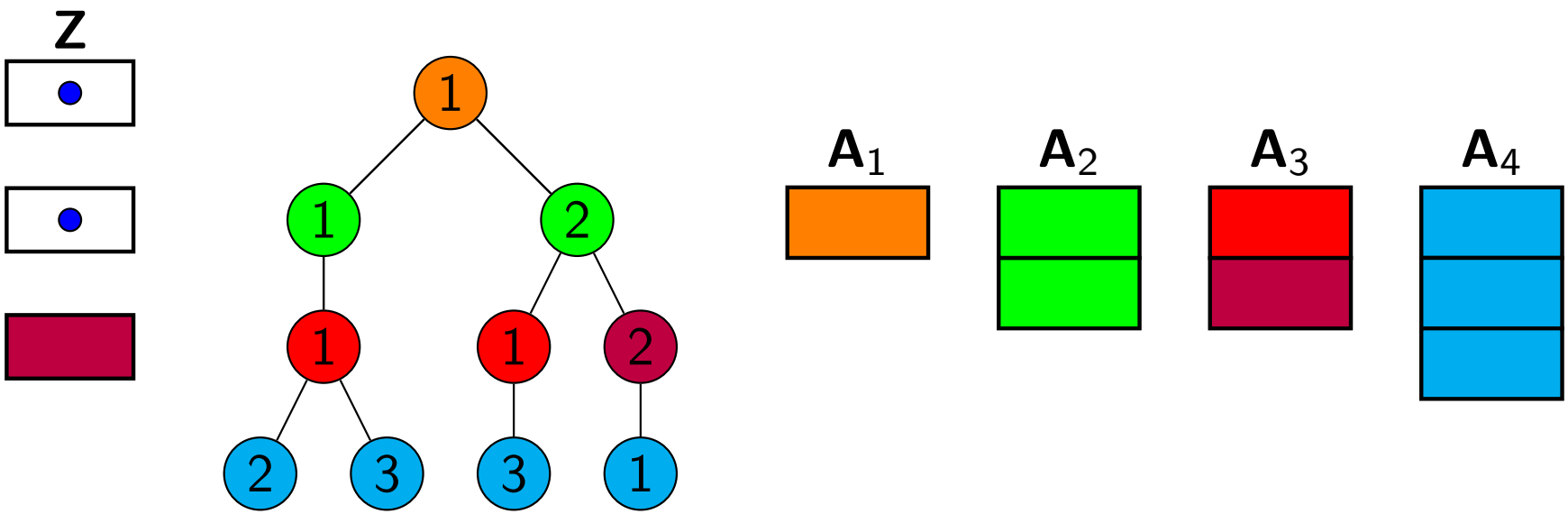
CSF-LEAF

- The traversal continues...



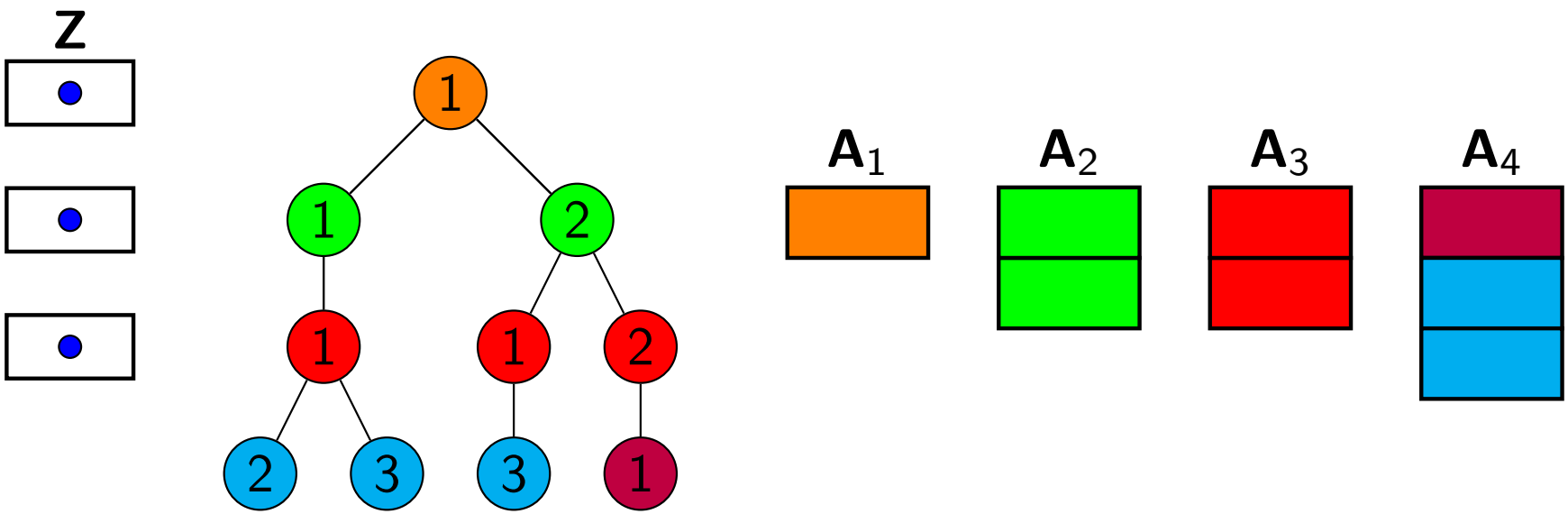
CSF-LEAF

- The traversal continues...



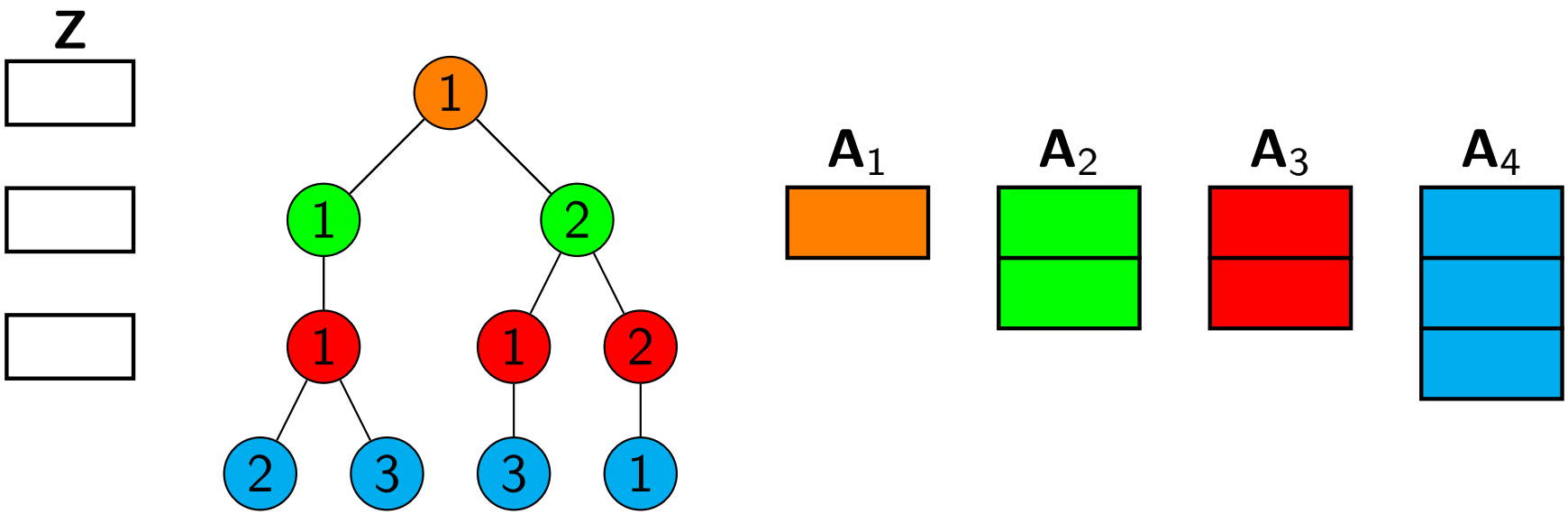
CSF-LEAF

- The traversal continues...



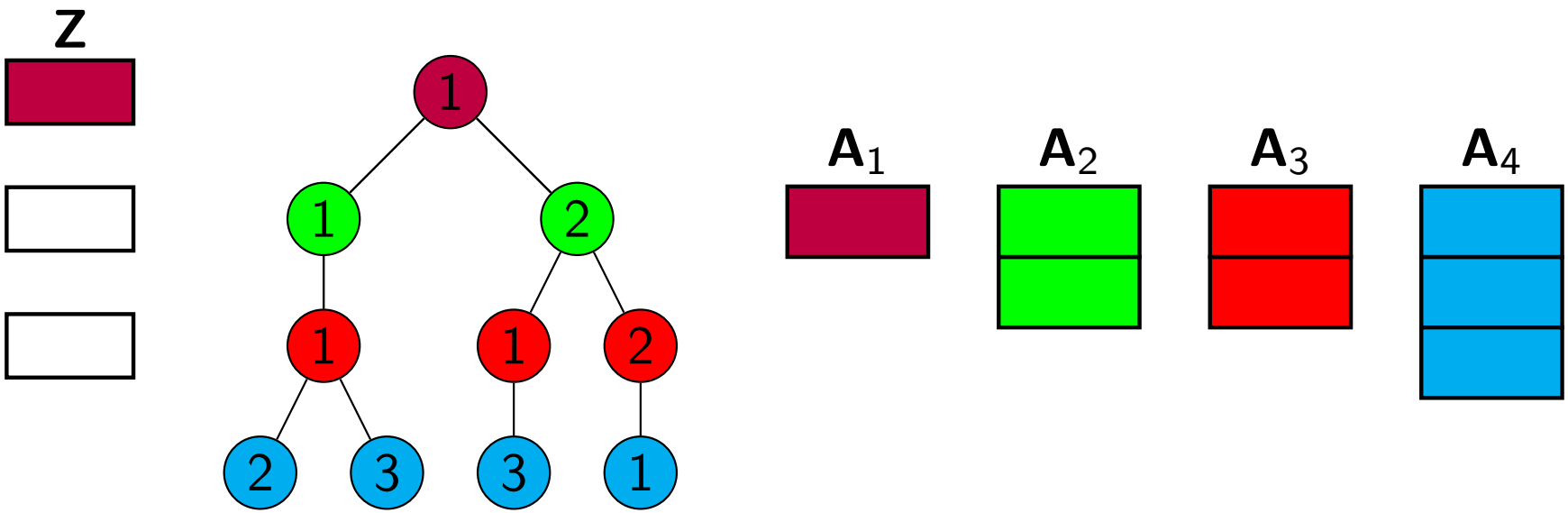
CSF-INTERNAL

- Internal nodes use a combination of CSF-ROOT and CSF-LEAF



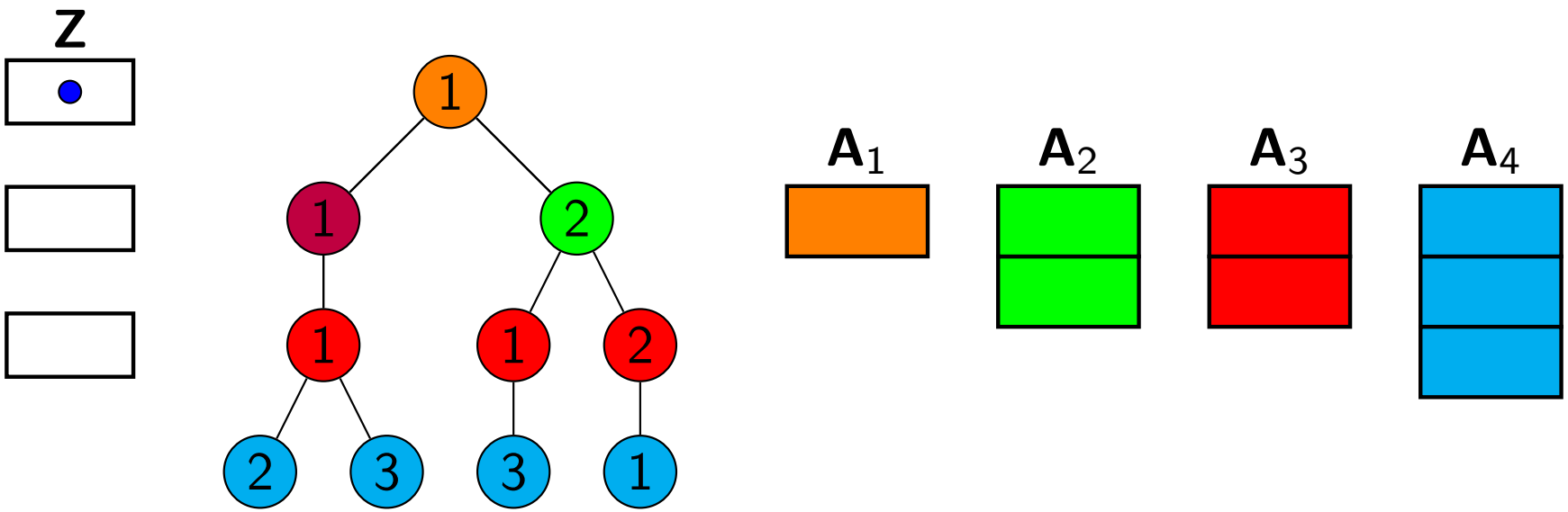
CSF-INTERNAL

- Hadamard products are pushed down to the output level



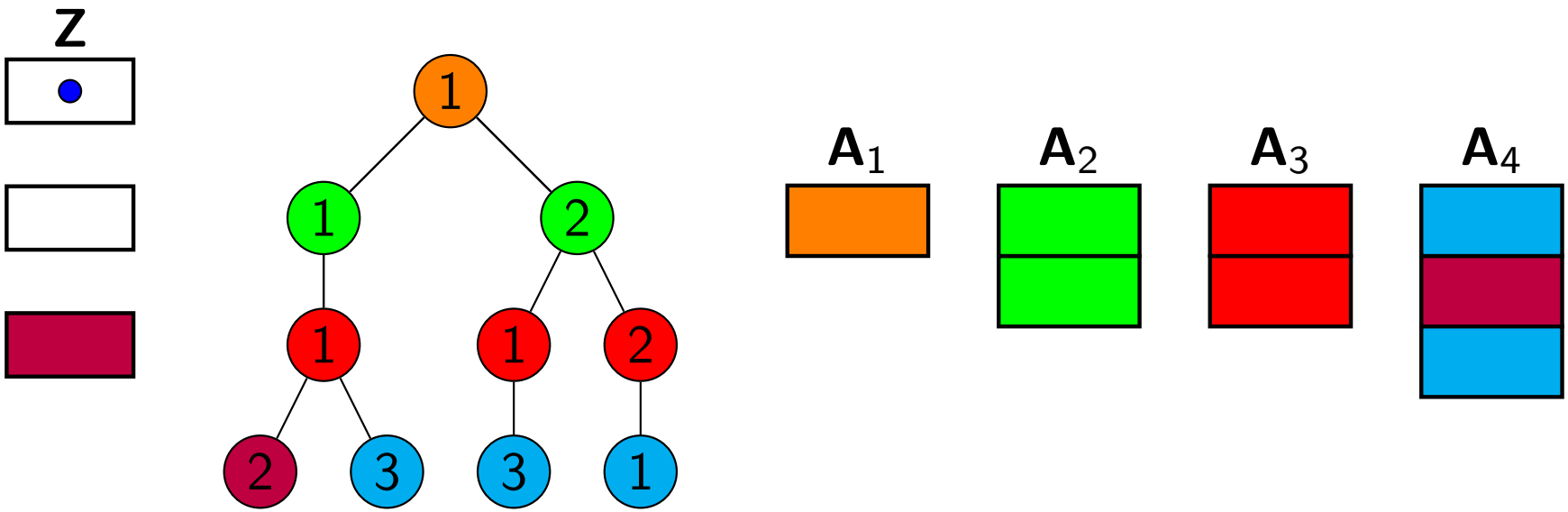
CSF-INTERNAL

- CSF-ROOT next pulls up to the output level



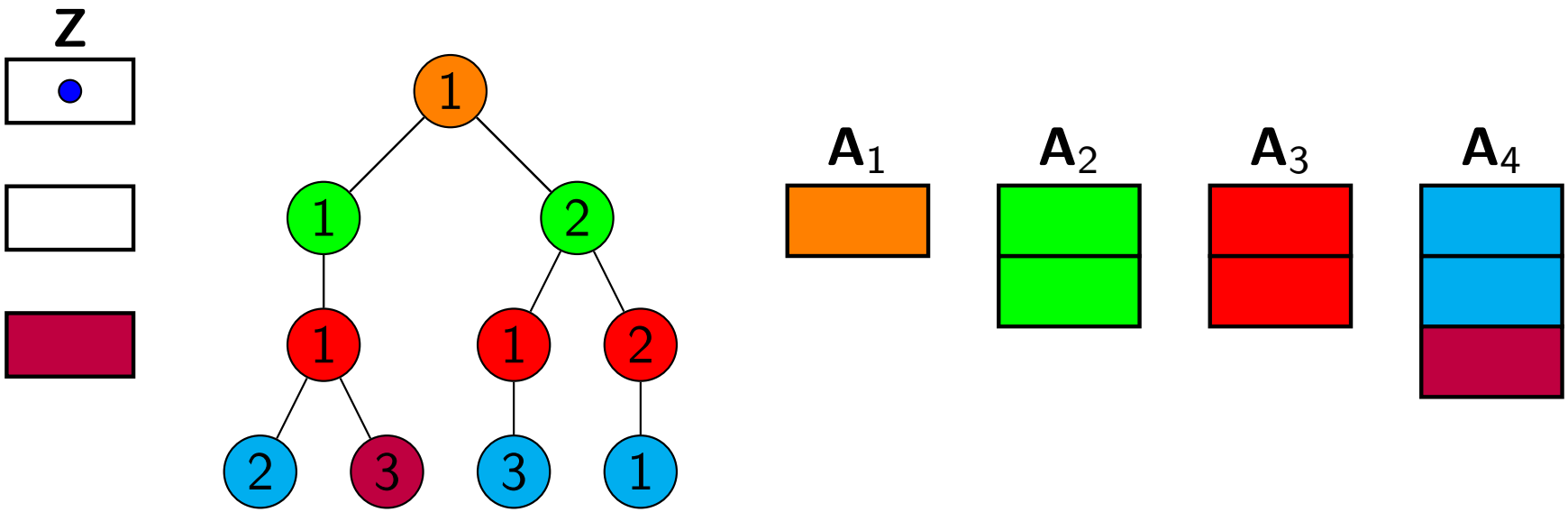
CSF-INTERNAL

- CSF-ROOT next pulls up to the output level



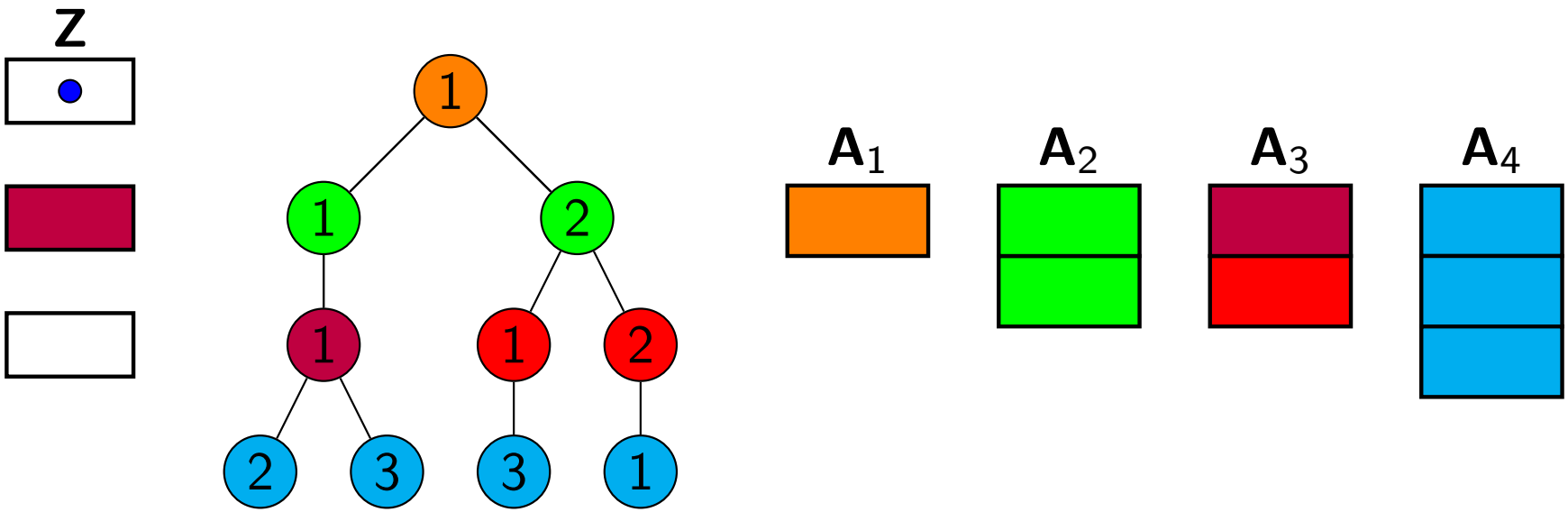
CSF-INTERNAL

- CSF-ROOT next pulls up to the output level



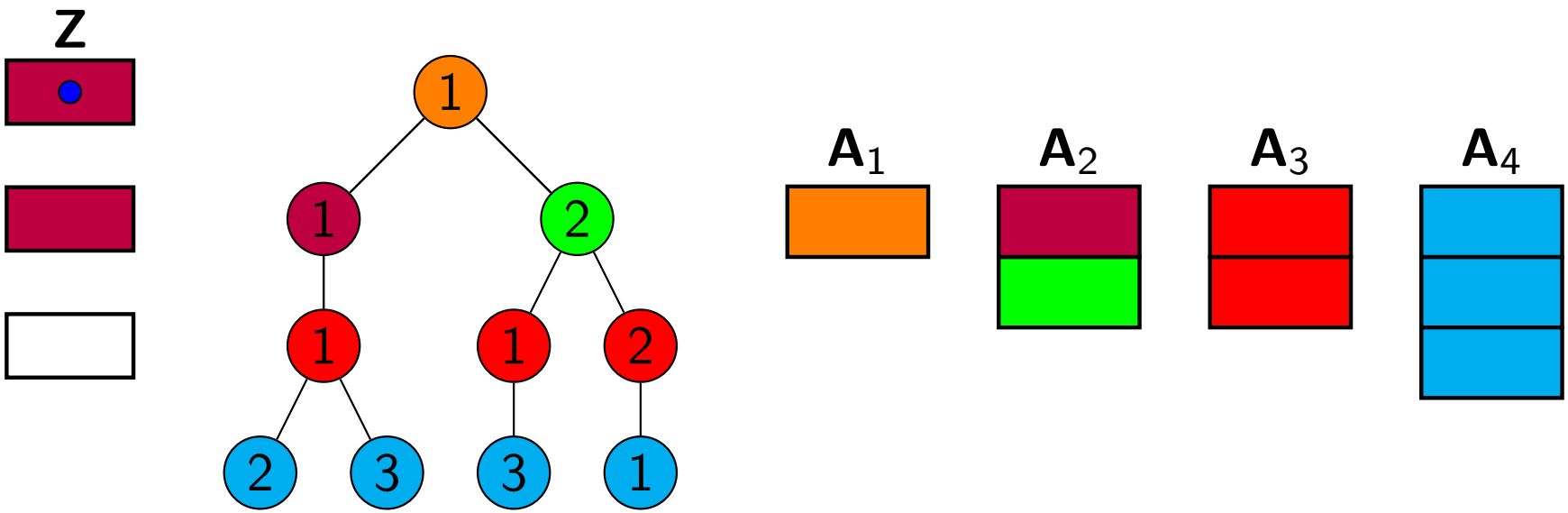
CSF-INTERNAL

- CSF-ROOT next pulls up to the output level

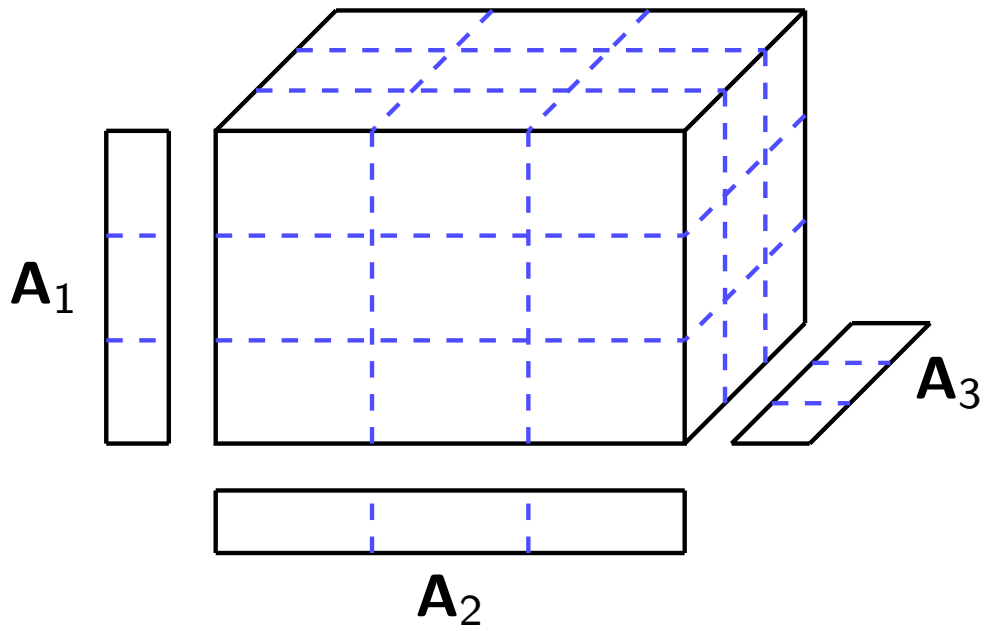


CSF-INTERNAL

- CSF-ROOT next pulls up to the output level



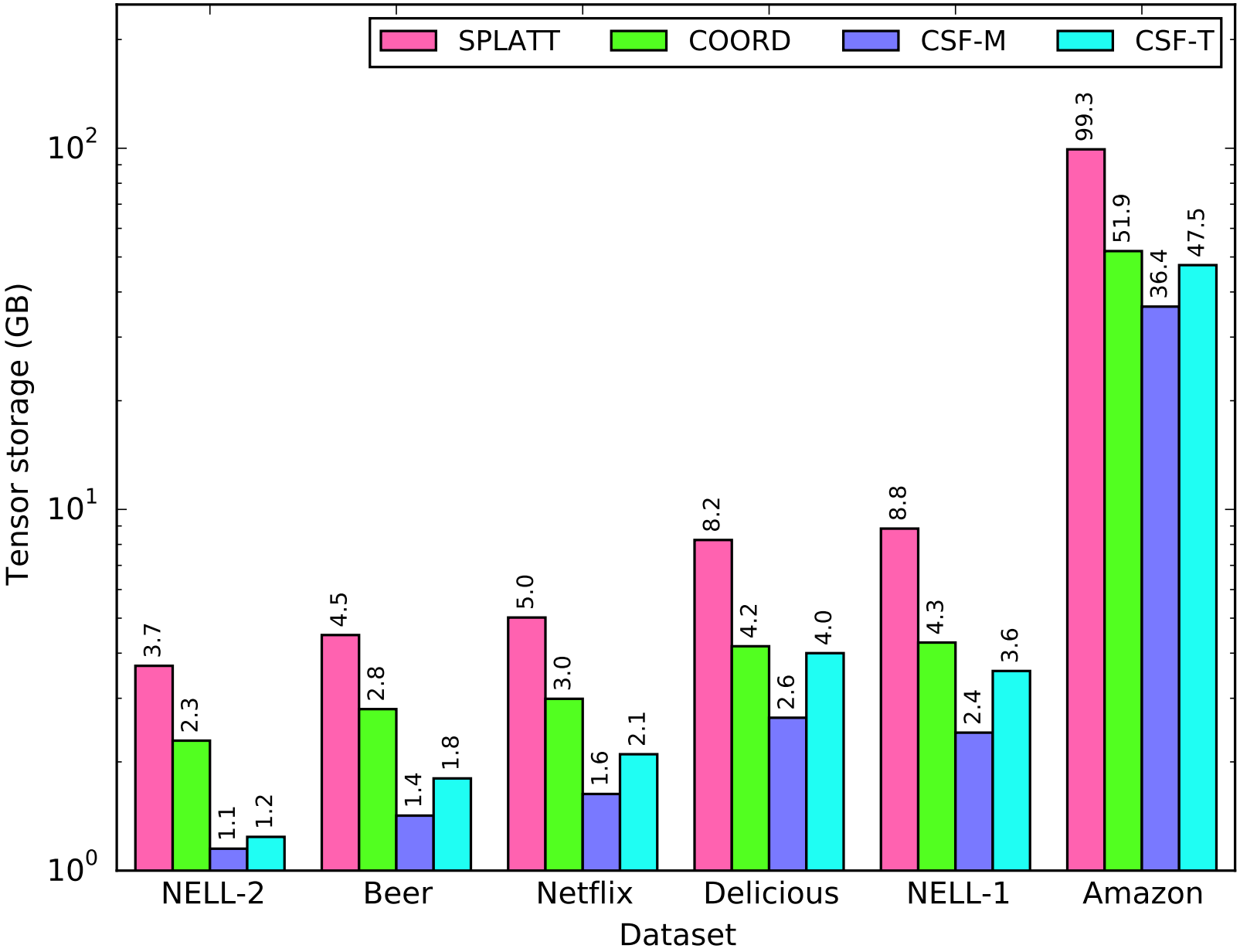
Parallelism – Tiling



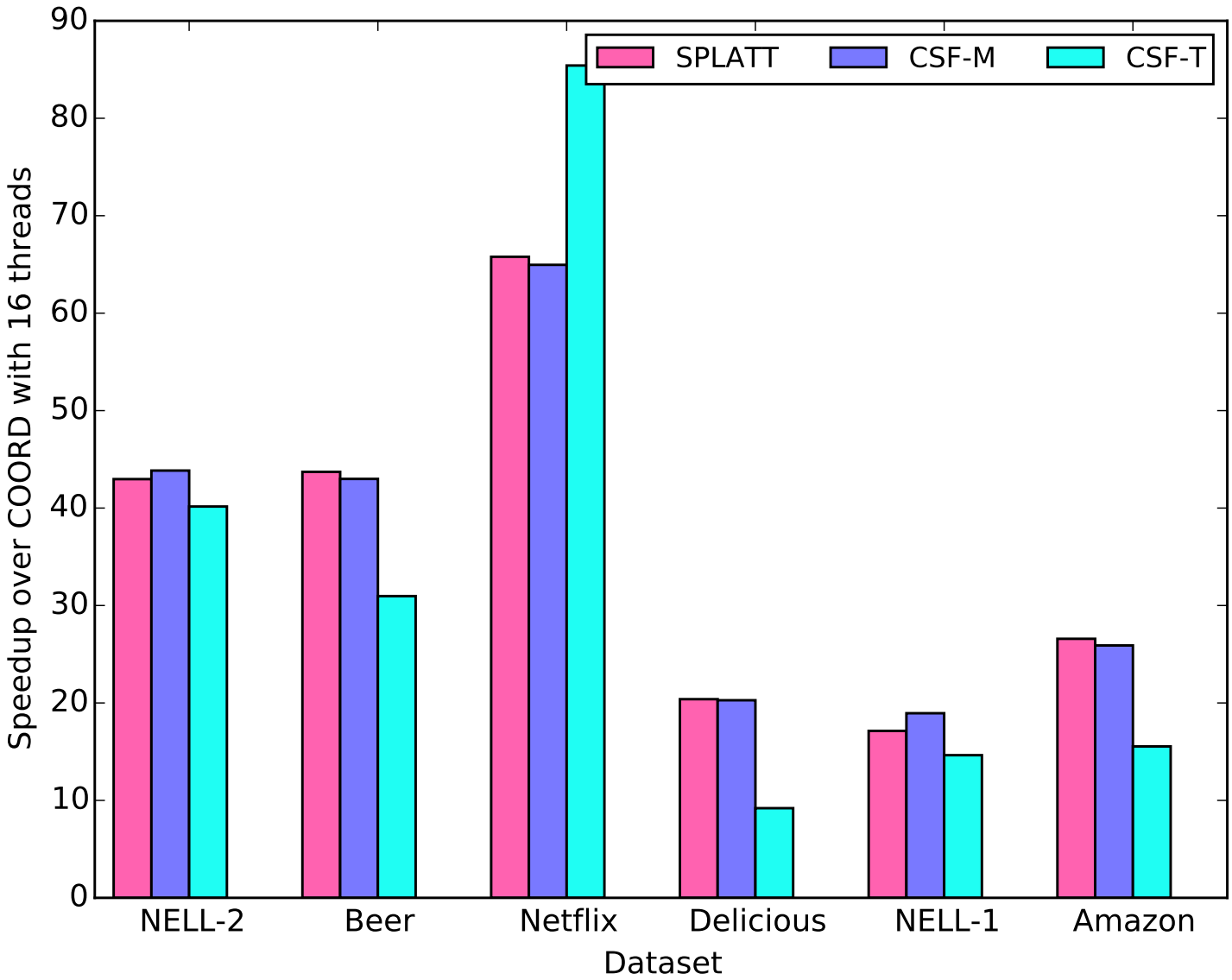
Datasets

Dataset	$\mathbf{I_1}$	$\mathbf{I_2}$	$\mathbf{I_3}$	**nnz**
NELL-2	12K	9K	28K	77M
Beer	33K	66K	960K	94M
Netflix	480K	18K	2K	100M
Delicious	532K	17M	3M	140M
NELL-1	3M	2M	25M	143M
Amazon	5M	18M	2M	1.7B

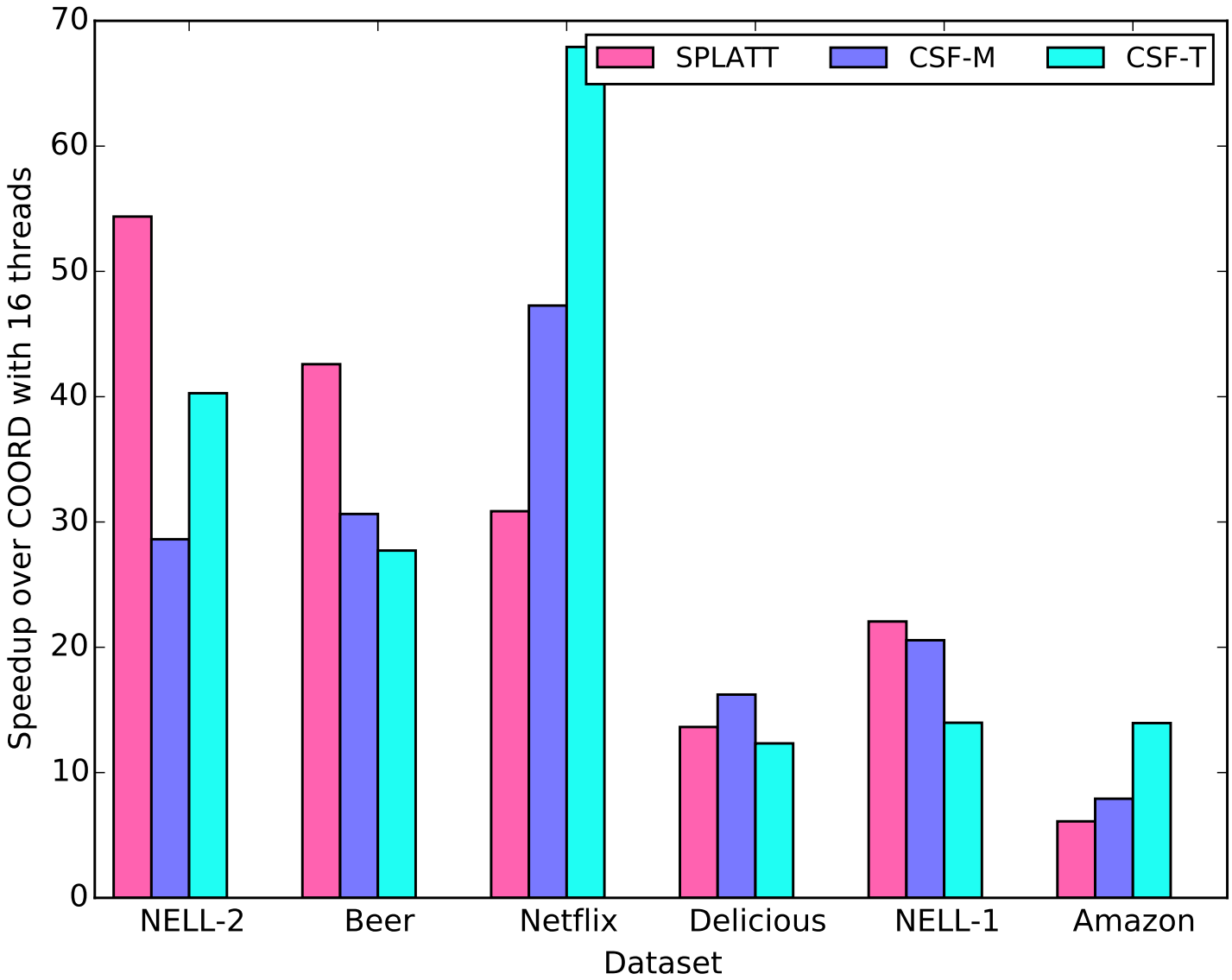
Storage Comparison



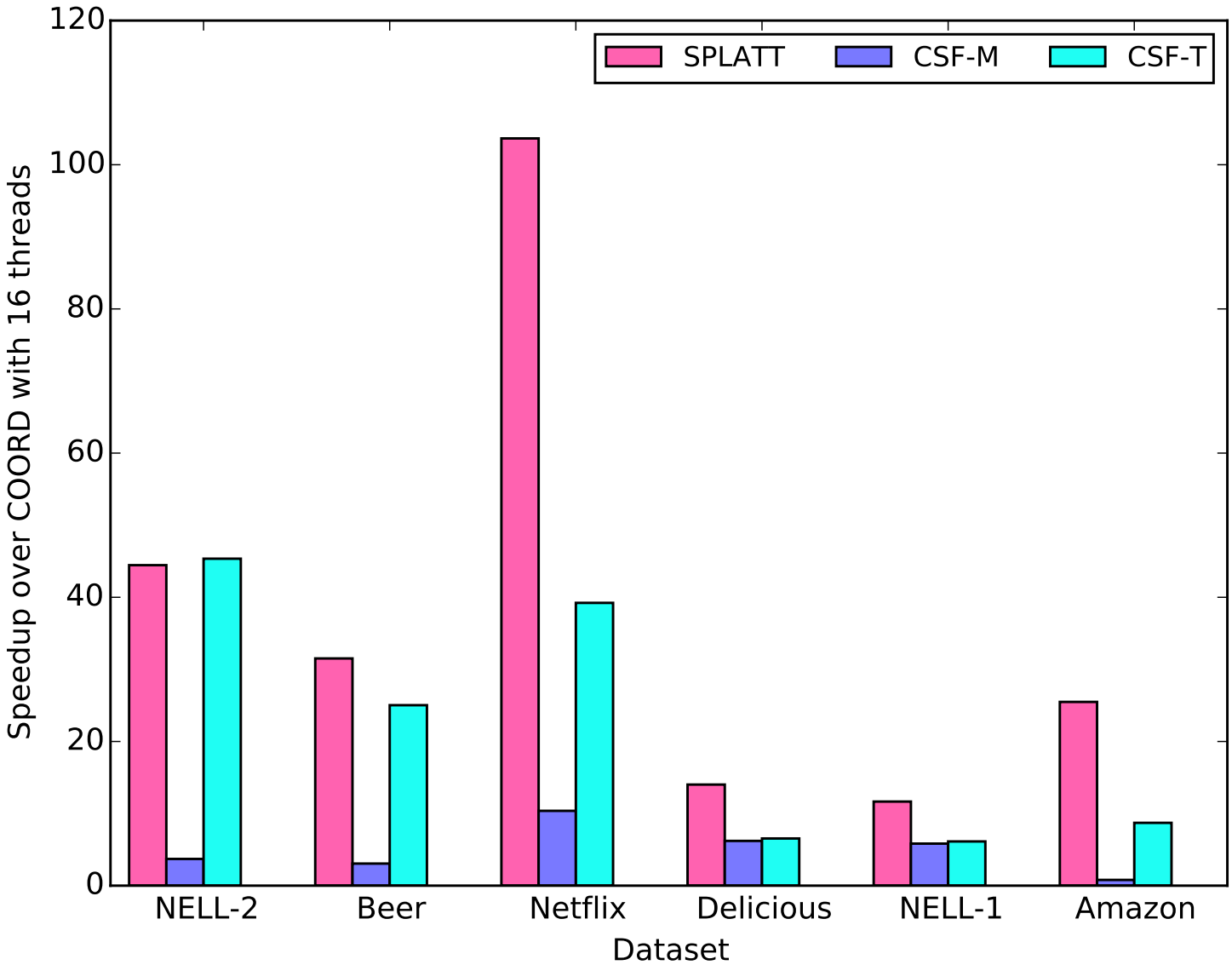
CSF-ROOT


SPLATT
CSF-M
CSF-T
Speedup over COORD with 16 threads
0
10
20
30
40
50
60
70
80
90
NELL-2
Beer
Netflix
Delicious
NELL-1
Amazon
Dataset

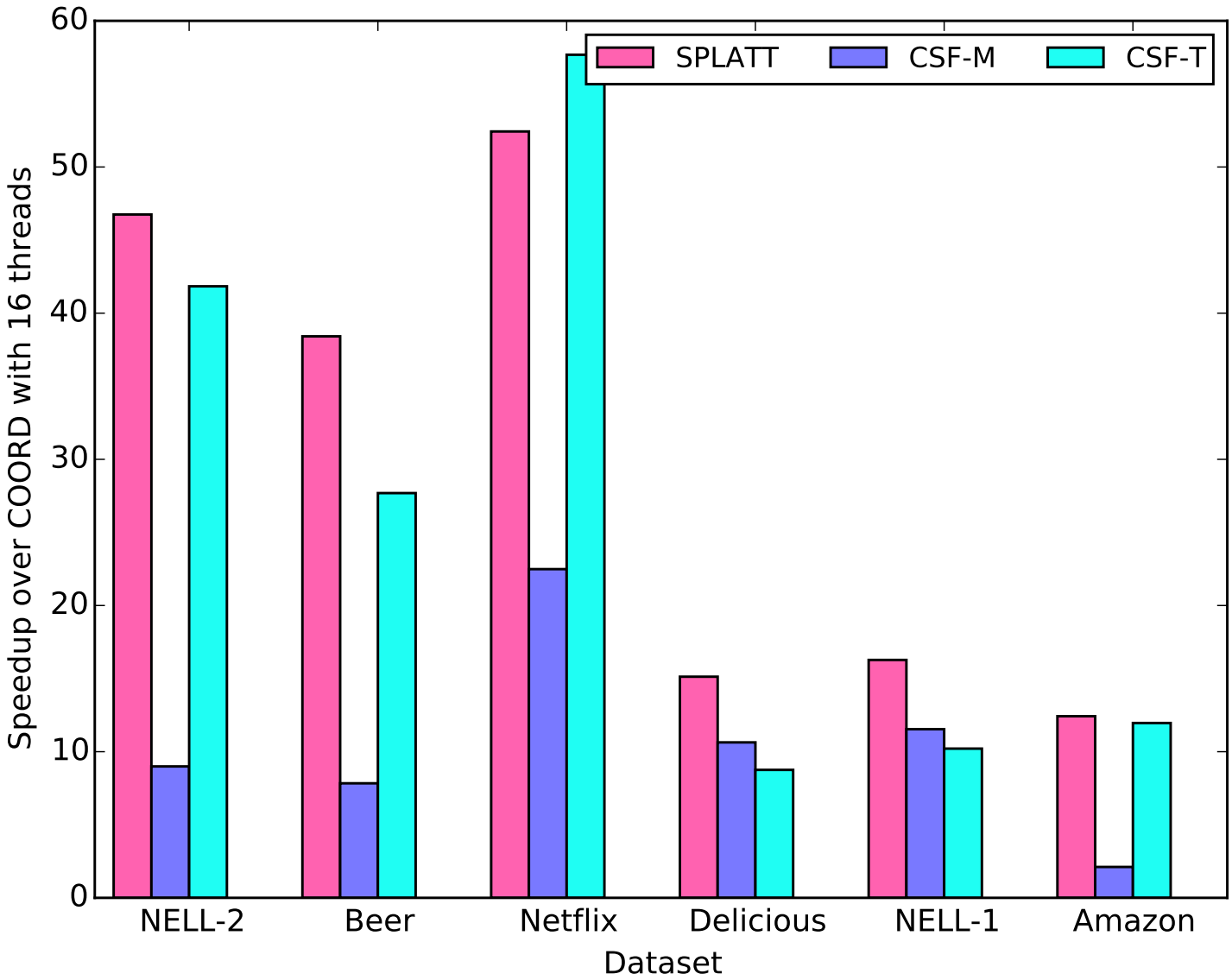
CSF-INTERNAL



CSF-LEAF


SPLATT
CSF-M
CSF-T
Speedup over COORD with 16 threads
0
20
40
60
80
100
120
NELL-2
Beer
Netflix
Delicious
NELL-1
Amazon
Dataset

MTTKRP


SPLATT
CSF-M
CSF-T
Speedup over COORD with 16 threads
0
10
20
30
40
50
60
NELL-2
Beer
Netflix
Delicious
NELL-1
Amazon
Dataset

Conclusions

Compressed Sparse Fiber

- CSF uses 58% less memory than SPLATT while maintaining 81% of its performance
- CSF and related algorithms are now included in SPLATT

http://cs.umn.edu/~splatt/