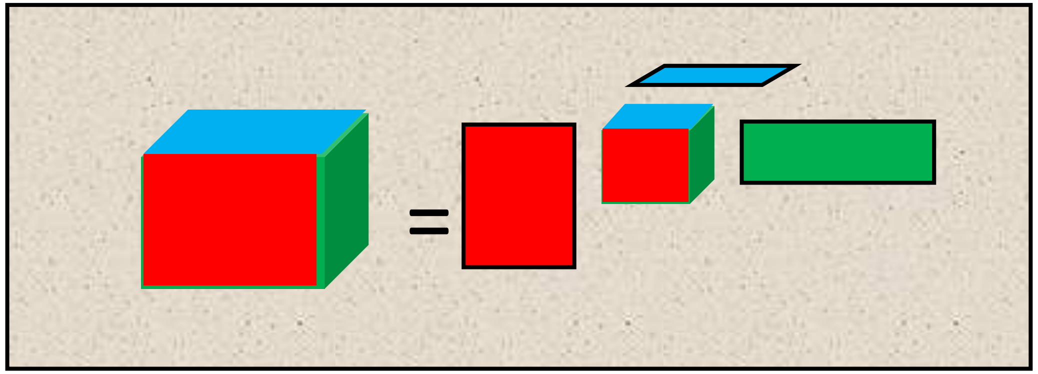
On Optimizing Distributed Non-Negative Tucker Decomposition

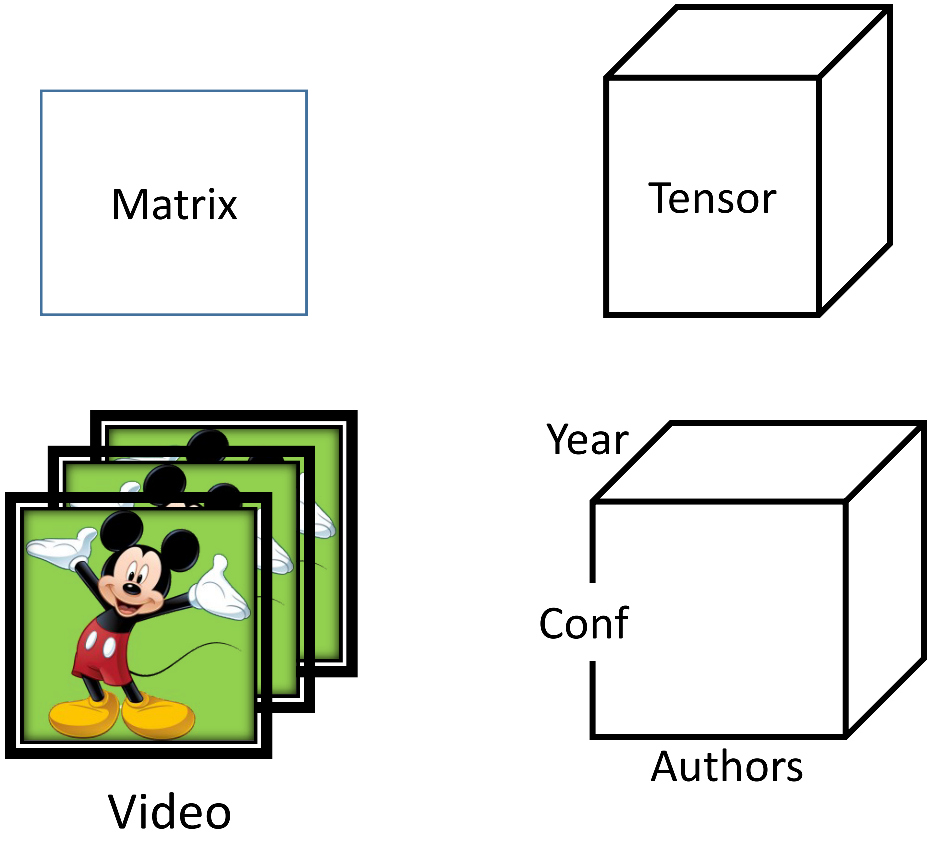
Venkatesan Chakaravarthy Shivmaran S Pandian
Saurabh Raje Yogish Sabharwal

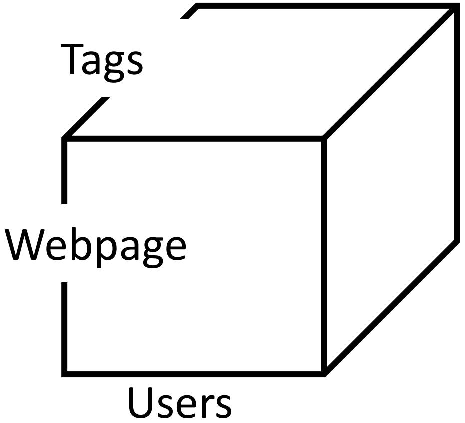
IBM Research - India

Presented by
Jee Whan Choi
University of Oregon



Tensors & Tucker Decomposition

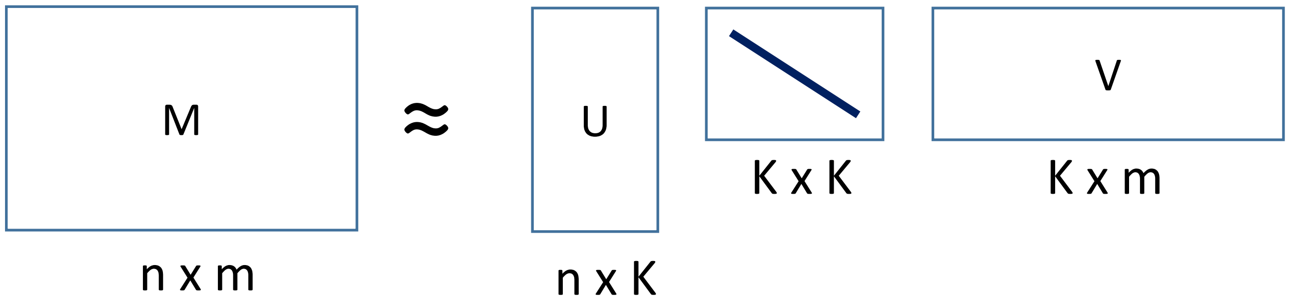




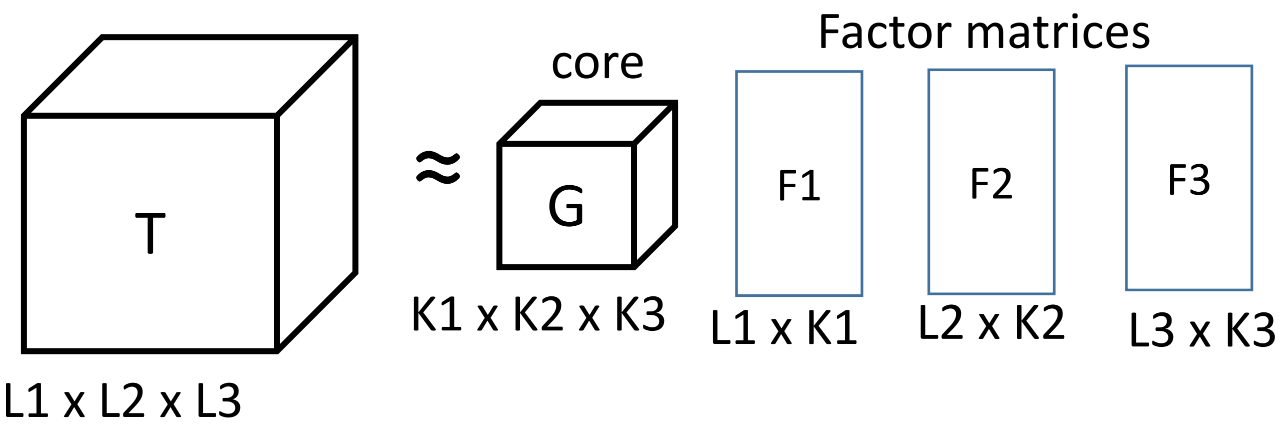
Delicious

- boolean

Singular Value Decomposition



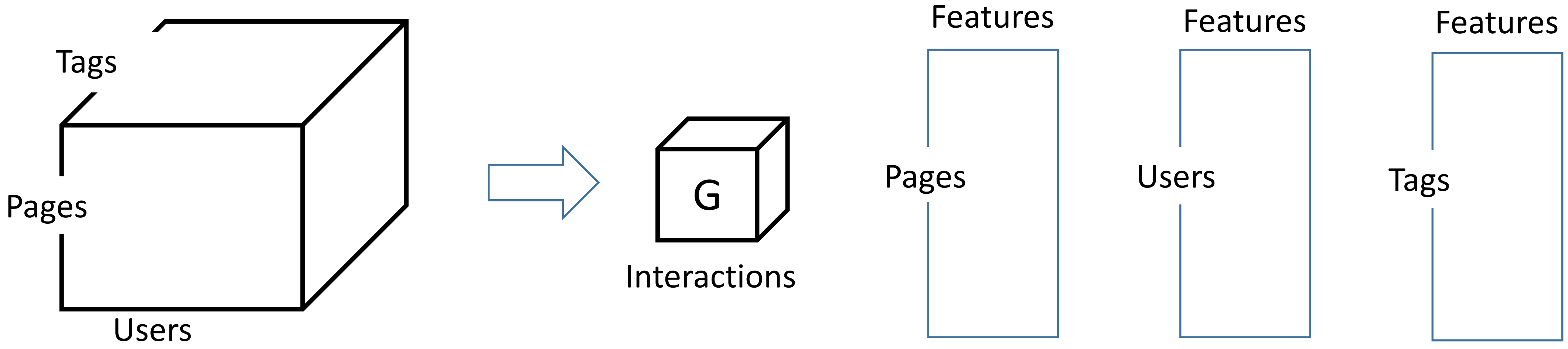
Tucker Decomposition

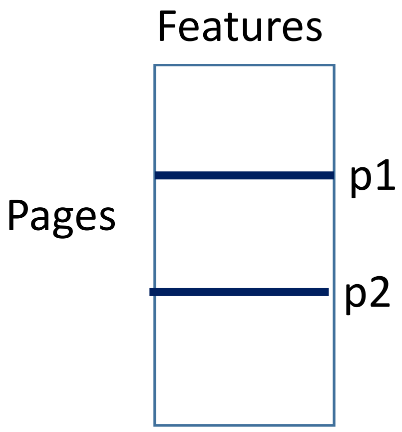


- K1, K2, K3 << L1, L2, L3
- Core much smaller compared to input tensor

Applications

- Data compression – especially for dense tensors
  - Core much smaller compared to input tensor
- Analysis similar to SVD for matrices
  - Principal component analysis, clustering, similarity and anomaly detection

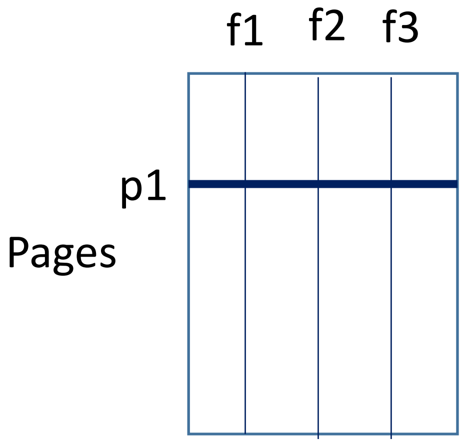




- Features → Top-K1 topics
- Compare and cluster pages based on the features

Non-negative Tucker Decomposition (NTD)

- Generalizes classical non-negative matrix factorization to higher dimensions
- Input tensor is non-negative.
- Imposes constraint that core tensor and factor matrices be non-negative.
- Direct interpretability of the decompositions



- Features are topics
- Entries are non-negative
- Entries can be interpreted as weights (importance) of the features to the page p1

Hidden topic identification

For each feature

- Identify top weight pages
- Analyze them to get the "topic" represented by the page.

For each page

- Identity top weight features
- These are topics relevant to the page

Prior Work

Tucker Decomposition

- HOSVD, ST-HOSVD and HOOI procedures
- Sequential, parallel and distributed settings
- Dense – [ABK 16, BK 07, CCJ+ 17, KS 08]
- Sparse – [BMVL 12, CCJ+ 18, KU 16]

CP Decomposition

- Special case of Tucker decomposition where core is diagonal
- Sequential, parallel and distributed settings : [KPA+ 16, KKU 16, KU 15, SK 15, SK 16, SK 17]

Non-negative Tensor Decomposition

- Non-negative matrix and tensor factorizations well-studied

  Cichocki, R. Zdunek, A. Phan, and S. Amari. Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation. John Wiley & Sons, 2009
- Non-negative CP decomposition
  - Distributed/Parallel implementation- [BHK 18]

Goal & Shoulders

Goal

- Develop an efficient distributed Non-negative Tucker decomposition for sparse tensors

MU-NTD Procedure

- Based on a procedure of [*Mørup, Hansen, Arnfred. Neural computation,2008]*
- Generalizes a classical procedure for non-negative matrix factorization *[Lee & Seung. Nature, 1999]*
- Given an NTD, produces a refined NTD with lesser error
  - Gradient descent
  - Multiplicative weight update strategy.
  - Alternative least squares paradigm
- Applied iteratively to obtain a local minima

HOOI Procedure

- A method based on element-wise Kronecker products [Kaya & Ucar, ICPP, 2016]
- Compressed sparse fiber (CSF) [Smith & Karypis, Euro-Par, 2017]
  - Tree-based representation of sparse tensors
  - Shares common computations across elements
  - Optimizes and reduces the computational load.

Our Contributions

- First distributed implementation for non-negative Tucker decomposition of sparse tensors

KronBU

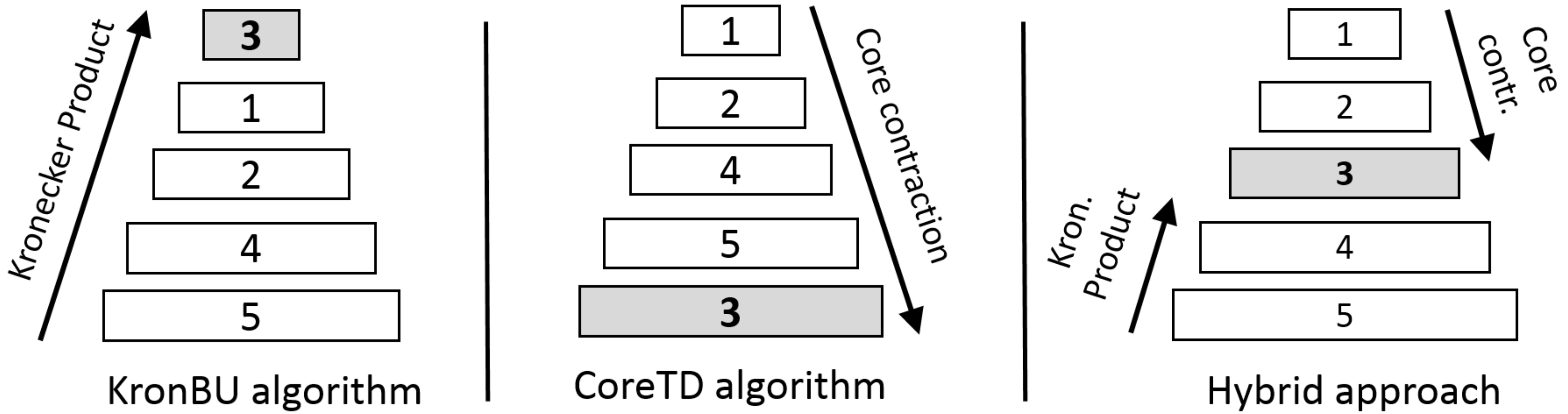
- Processes CSF-trees in bottom-up manner
- Via Kronecker products
- Incorporates known optimizations from HOOI
- Serves as baseline

CoreTD

- New operation called core contraction
- Unique to MU-NTD
- Processes CSF-trees in top-down manner

Hybrid procedure

- Contrasting traversals lead to tradeoffs in computational load
- Utilizes both Kronecker products and core contraction operations
- Processes CSF-trees simultaneously bottom-up and top-down
- Significant reduction in load and execution time



Our Contributions

Distributed Implementation

- Efficient distributed implementation based on above procedures.
- Distribution policy
  - Lite [CCJ+ 18] developed for HOOI works well for our setting as well.

Experimental evaluation

Tensors

- Large real-life sparse tensors
- From FROSTT repository.

System

- 32 to 512 MPI ranks

Performance

- CoreTD outperforms baseline KronBU
- Hybrid offers overall 4x gain over baseline

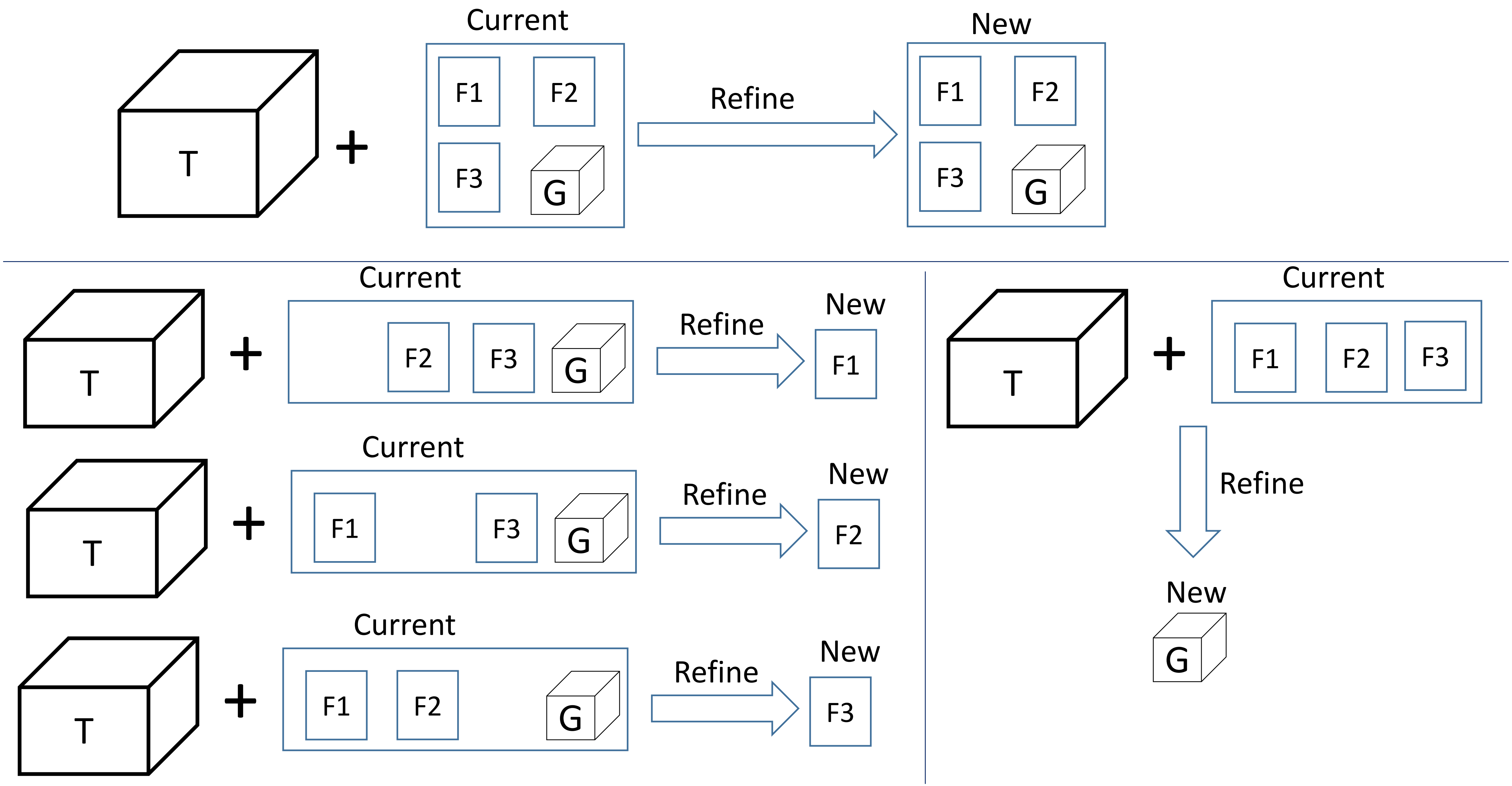
Scaling:

- 12x speedup
- Ideal – 16 x

Memory and Pre-processing Time

- KronBU and CoreTD use N CSF-trees one along each mode
- Hybrid uses a single tree
- Hybrid reduces memory and time for CSF-tree construction

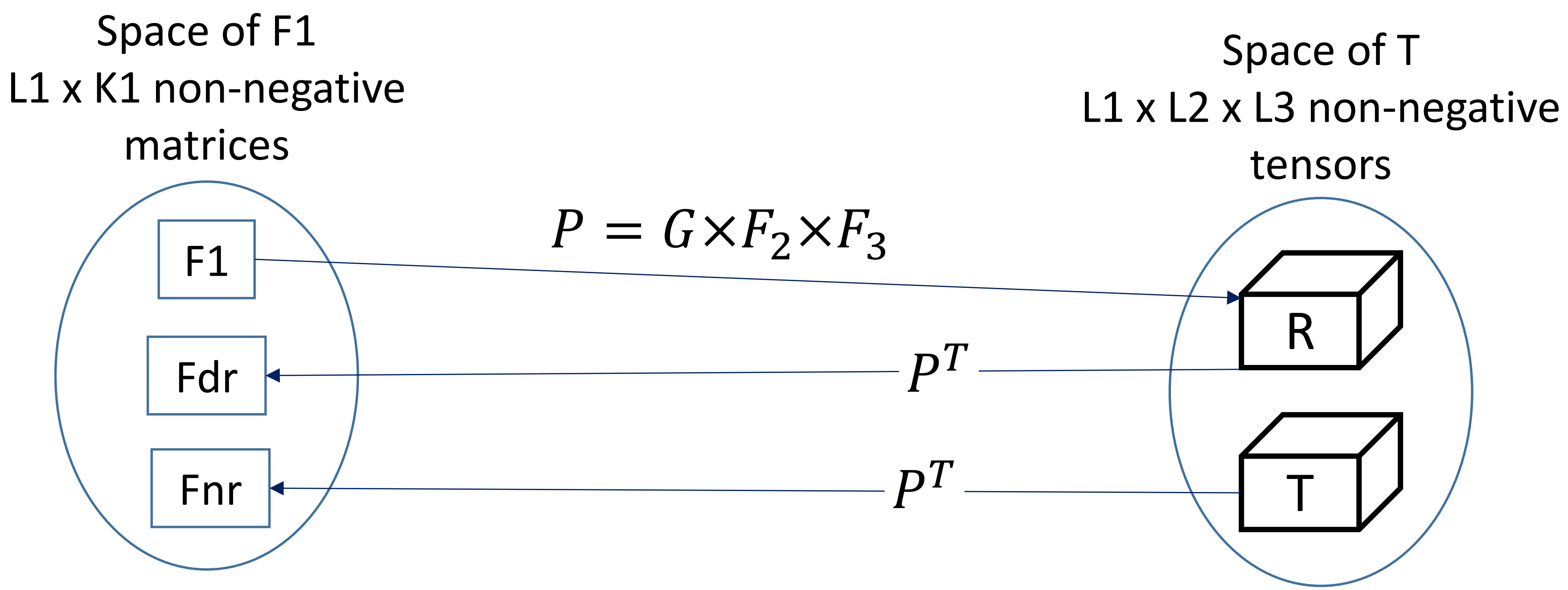
MU-NTD Procedure – ALS Paradigm



MU-NTD Procedure – Updating F1

Current: $T \approx G \times F_1 \times F_2 \times F_3 = R$

- Multiplicative weight update – gradient descent
- Generalized from NMF



- $F_1 = F_1 \star \frac{Fnr}{Fdr} = \frac{T \cdot P^T}{R \cdot P^T}$
- $P = G \times F_2 \times F_3$

Fdr is easy

Fnr is the main computation

Fnr – KronBU Algorithm

$$\begin{aligned} Fnr &= T \cdot P^T \\ &= T \cdot (G \times F_2 \times F_3)^T \end{aligned}$$

Reformulation

$$Fnr = (T \times F_2^T \times F_3^T) \cdot G^T$$

TTM Chain Z

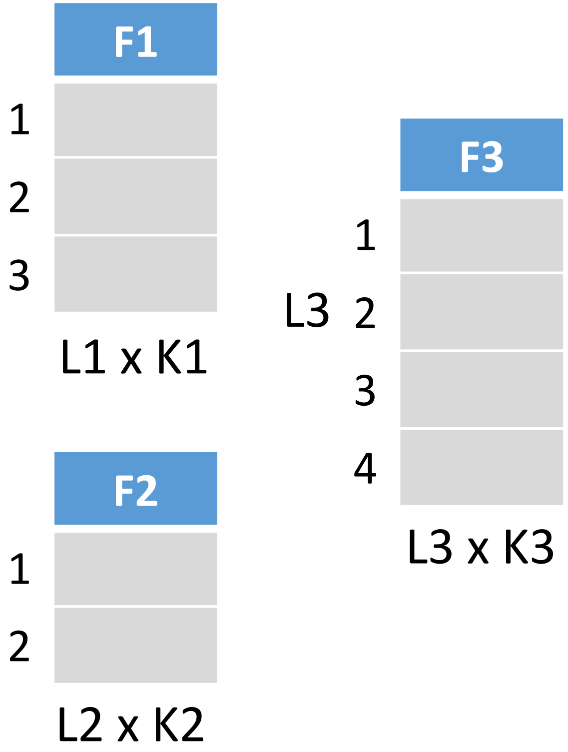
$$Z = T \times F_2^T \times F_3^T$$

- TTM Chain Z occurs in HOOI procedure
  - Element-wise Kronecker product [KU ‘16]
  - CSF-tree based optimization [SK ‘17]

Element-wise Kronecker Product

T	c1	c2	c3
e1	1	1	2
e2	2	2	3
e3	1	1	1
e4	2	2	4
e5	3	1	3

Z = L1 x (K2 K3)

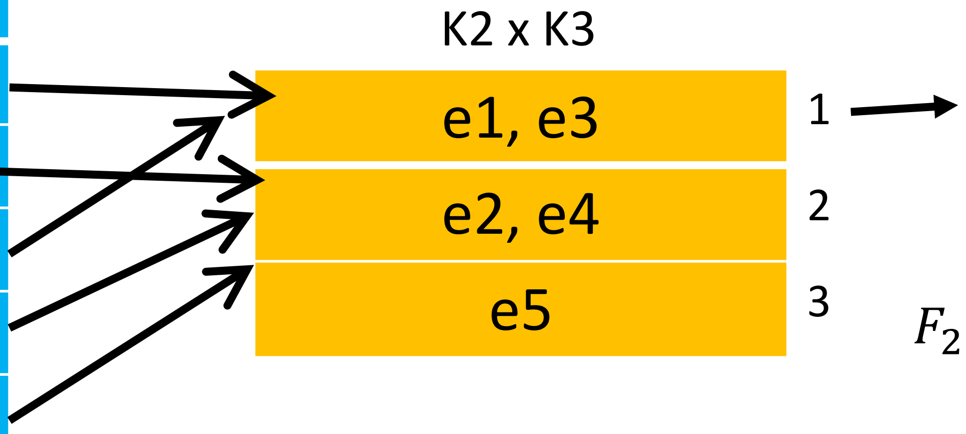


Action of element e

$$Z[c_1(e)] \mathrel{+}= F_2[c_2(e)] \otimes F_3[c_3(e)]$$

Fnr – KronBU Algorithm

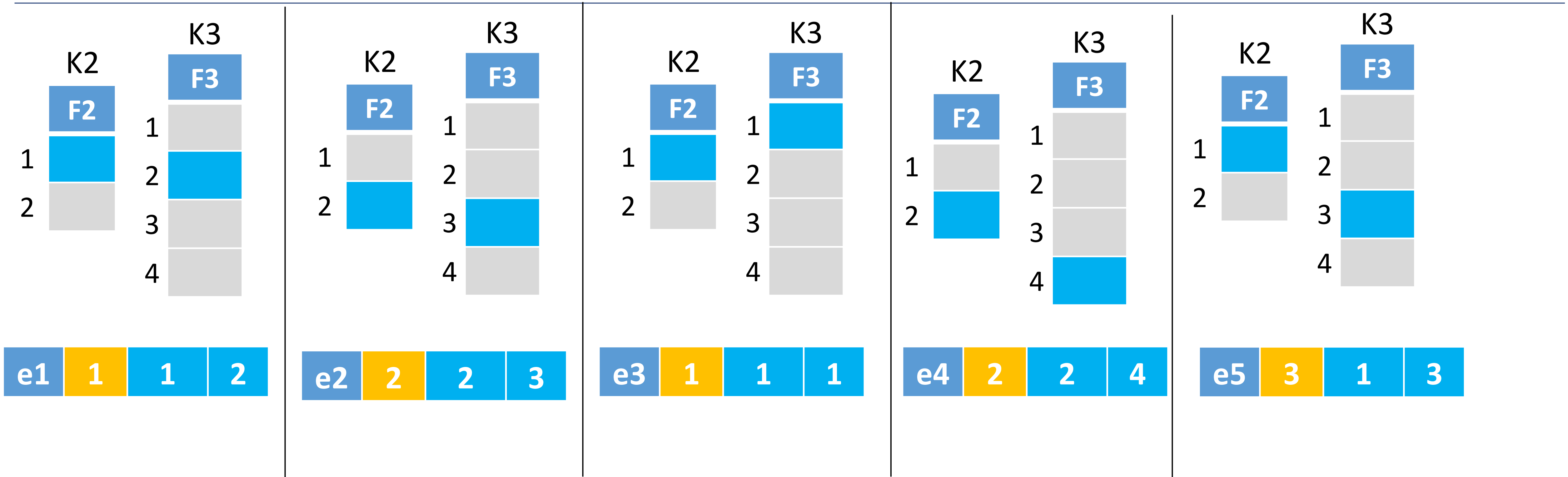
T	c1	c2	c3
e1	1	1	2
e2	2	2	3
e3	1	1	1
e4	2	2	4
e5	3	1	3



$$(F_2[1] \otimes F_3[2]) + (F_2[1] \otimes F_3[1])$$

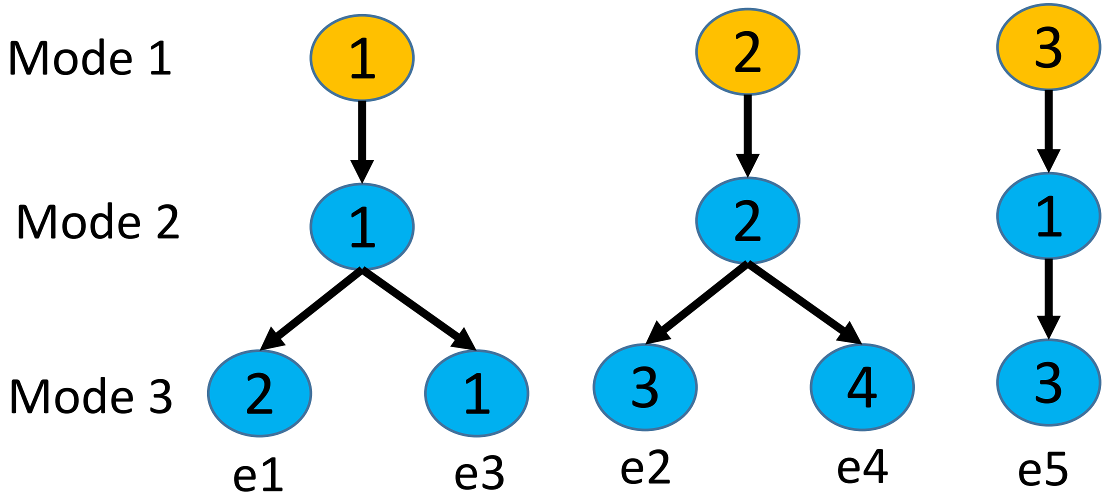
Distributivity

$$F_2[1] \otimes (F_3[2] + F_3[1])$$



KronBU Algorithm – Compressed Sparse Fiber (CSF) Trees

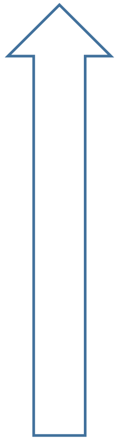
T	c1	c2	c3
e1	1	1	2
e2	2	2	3
e3	1	1	1
e4	2	2	4
e5	3	1	3



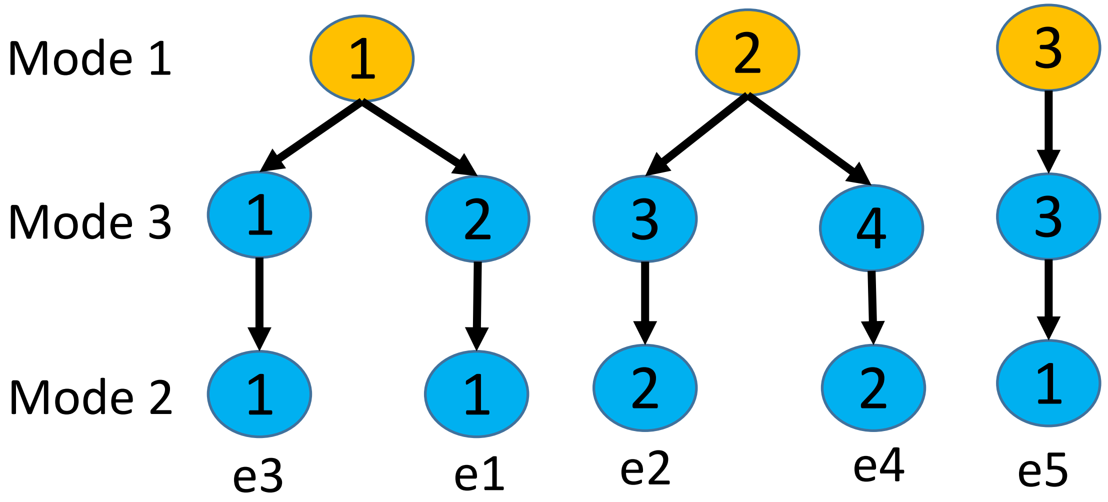
- Update Z
- Sum children
- Multiply by node

element

Bottom-up



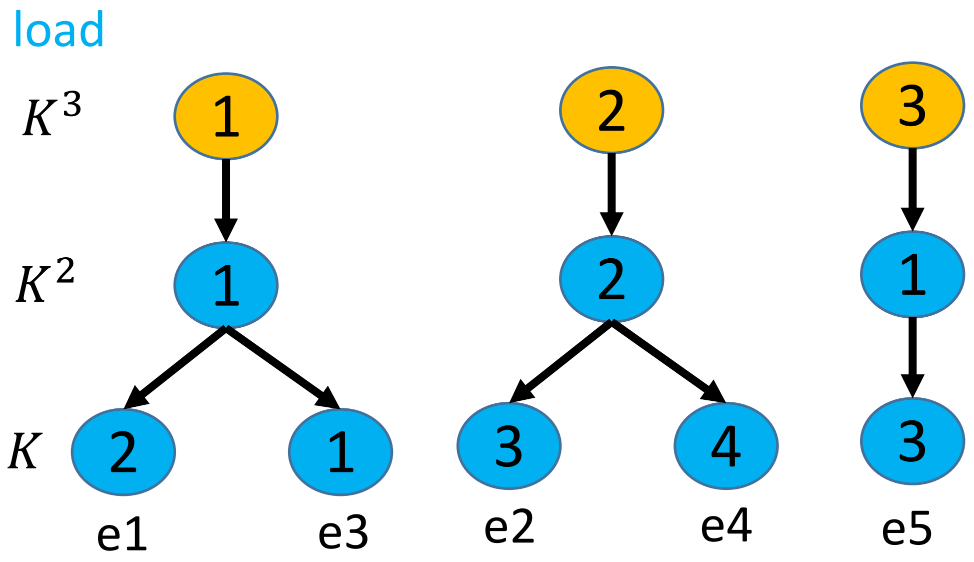
Alternative CSF-trees



Mode Ordering

Load Analysis

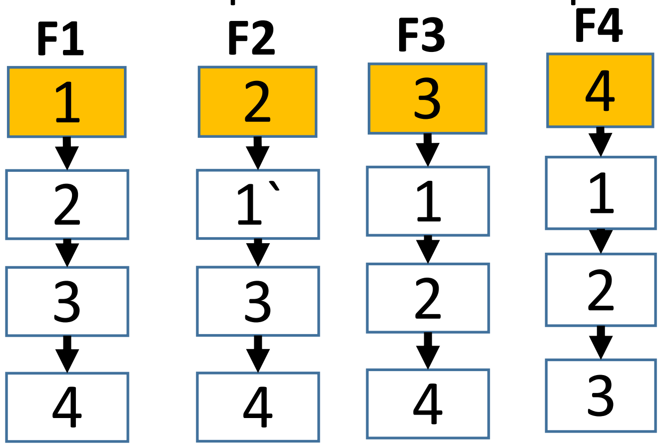
- Kronecker product expands output
  - $v_3 \leftarrow (v_1 \otimes v_2)$
  - $l_1 \otimes l_2 \rightarrow (l_1 \cdot l_2)$
- N! trees are possible
- Exhaustive search



Ideal greedy ordering [SK 17] : Sort by increasing order of length

Mode Placement Constraint

- Factor matrices get computed along each mode F1, F2, …, F_N
- The output gets produced at the top
  - Mode n under consideration must be placed at the top.
  - Assuming L1 < L2 < L3 < L4



Longer modes → More load

- Along the long mode, high fan-in
- More nodes at the top

Flickr
Load
GFLOPS

Mode	Length	KronBU
1	731	6.1
2	319 K	9.3
3	1.6 M	29
4	28 M	313

Top-Down Approach – CoreTD Algorithm

$$Fnr = T \cdot (G \times F_2 \times F_3)^T$$

- Directly evaluate Fnr

Reformulation

$$Fnr = (T \times F_2^T \times F_3^T) \cdot G^T$$

TTM Chain Z

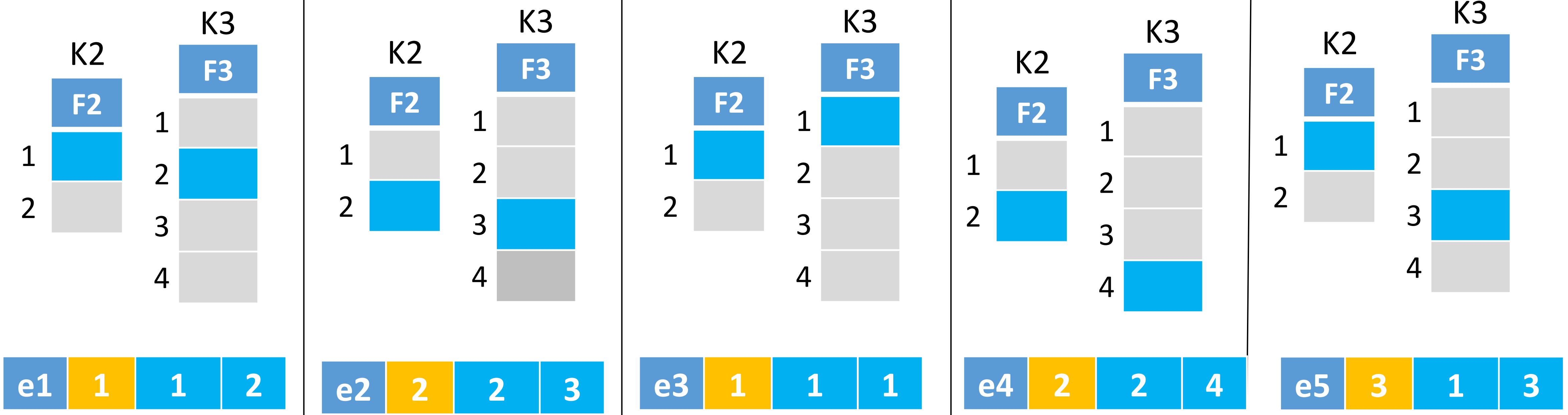
$$Z = T \times F_2^T \times F_3^T$$

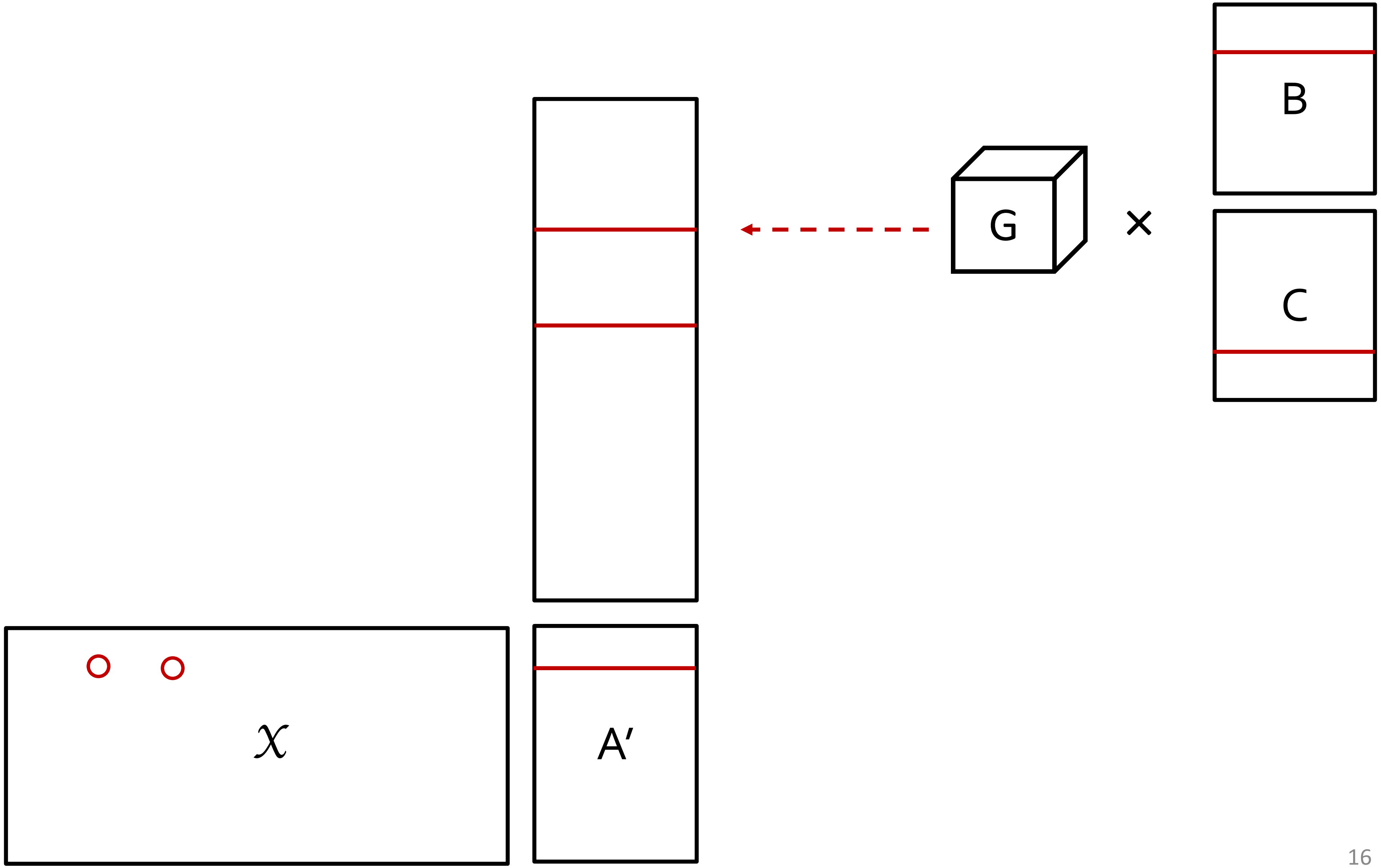
T	c1	c2	c3
e1	1	1	2
e2	2	2	3
e3	1	1	1
e4	2	2	4
e5	3	1	3

Element-wise Core Contraction

$$F_1[c_1(e)] \mathrel{+}= G \times F_2[c_2(e)] \times F_3[c_3(e)]$$

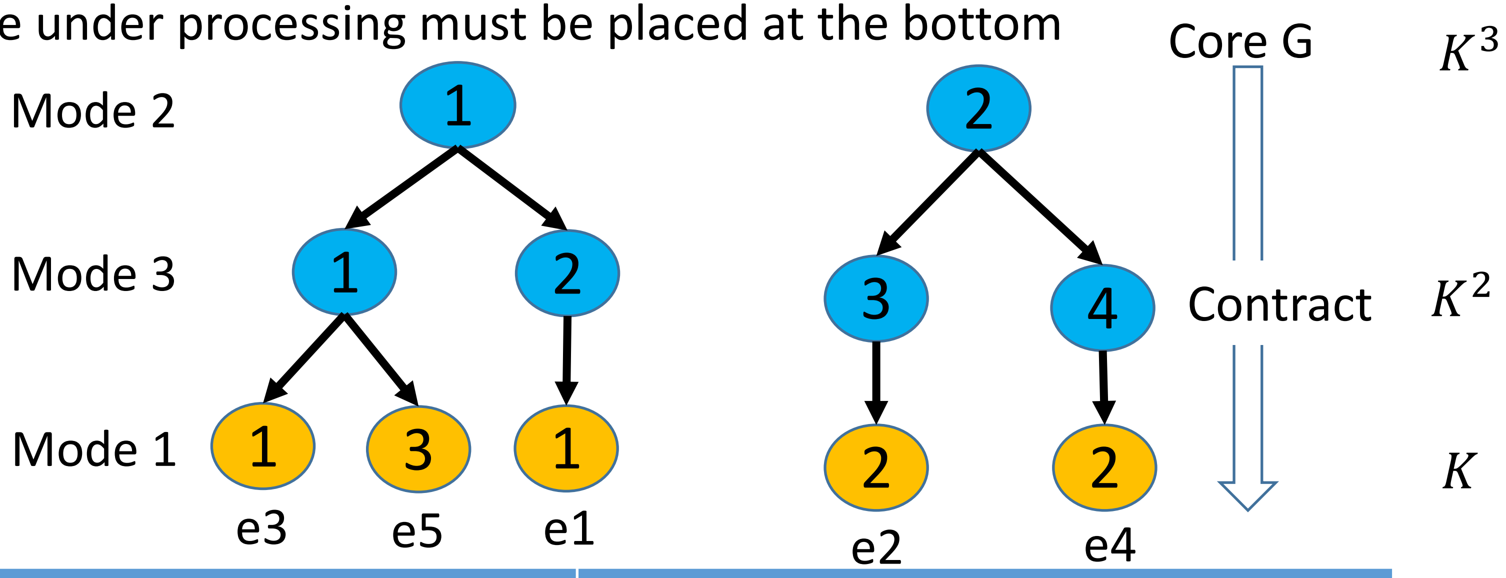
- $G = K_1 \times K_2 \times K_3$
- $G \times F_2[c_2(e)] = K_1 \times K_2$
- $G \times F_3[c_3(e)] = K_1$




B
G
×
C
𝒳
A′

CoreTD Algorithm – CSF-trees

- Top-down process
- Output gets generated at the leaf level
- Mode under processing must be placed at the bottom



T	c1	c2	c3
e1	1	1	2
e2	2	2	3
e3	1	1	1
e4	2	2	4
e5	3	1	1

KronBU	CoreTD
Kronecker Product	Core contraction
Expansion	Contracts
Bottom-up	Top-down
Desired output at the top	Desired output at the bottom
Mode n must be placed at top	Mode n must be placed at bottom
High load on long modes	High load on short modes

Mode Placement Constraints

- All trees have same load pattern
- Mode n must be placed at bottom or top

Hybrid Algorithm

Element-wise contribution

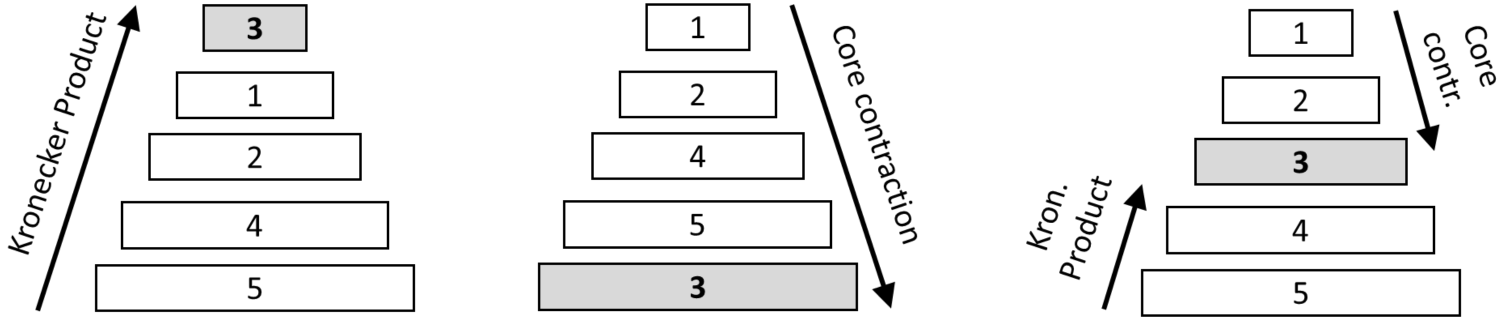
- Combines Kronecker product and core contraction.

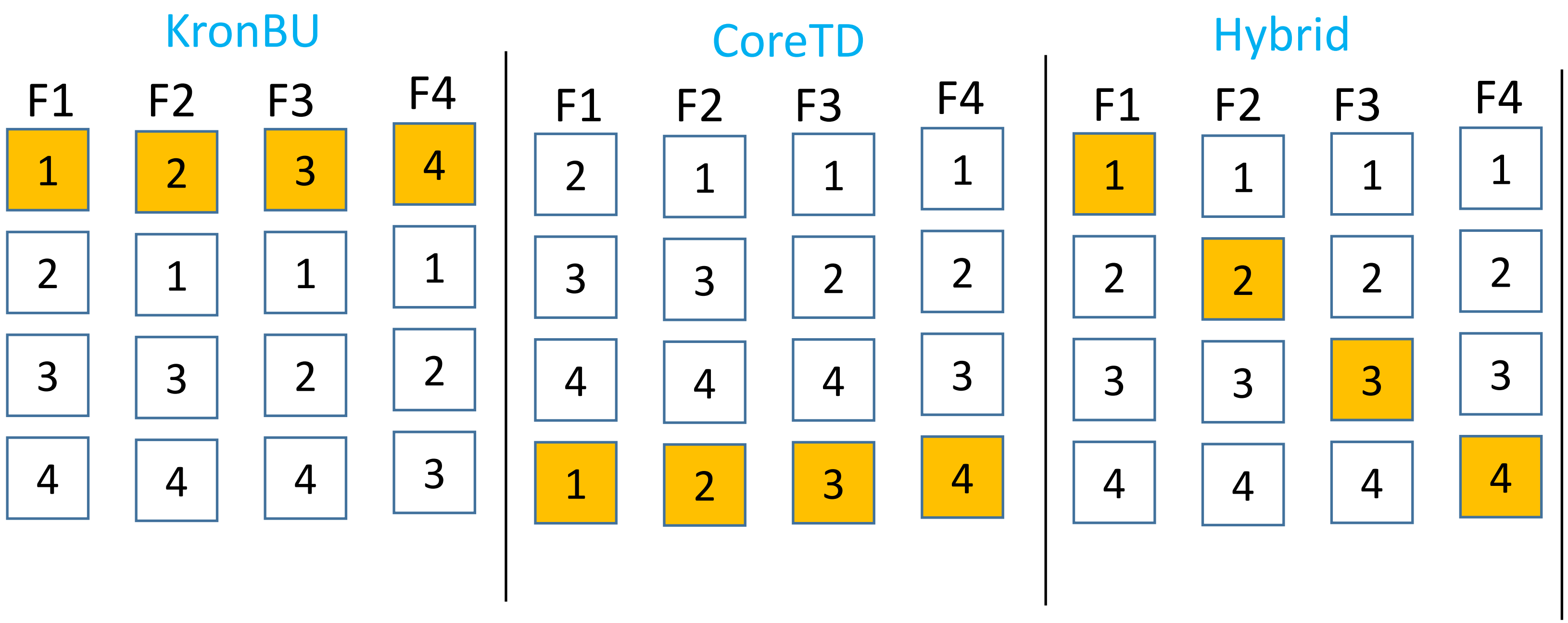
Kronecker product • Core contraction

$$\boxed{F_1[c_1(e)] \otimes F_2[c_2(e)]} \bullet \boxed{G \times F_4[c_4(e)] \times F_5[c_5(e)]}$$

Meet in the Middle

- No mode placement constraints
- Any tree can be used along any mode
- Ideal greedy ordering for all modes





Flickr – Load (GFLOPS)

Mode	Length	KronBU	CoreTD	Hybrid
1	731	6.1	28	6.1
2	319 K	9.3	22	6.1
3	1.6 M	29	6.5	6.1
4	28 M	313	6.1	6.1
Total	-	357.4	62.6	24.4

Distributed Implementation – Distribution Policy

- Elements are distributed among the processors
- Distribution policy is critical

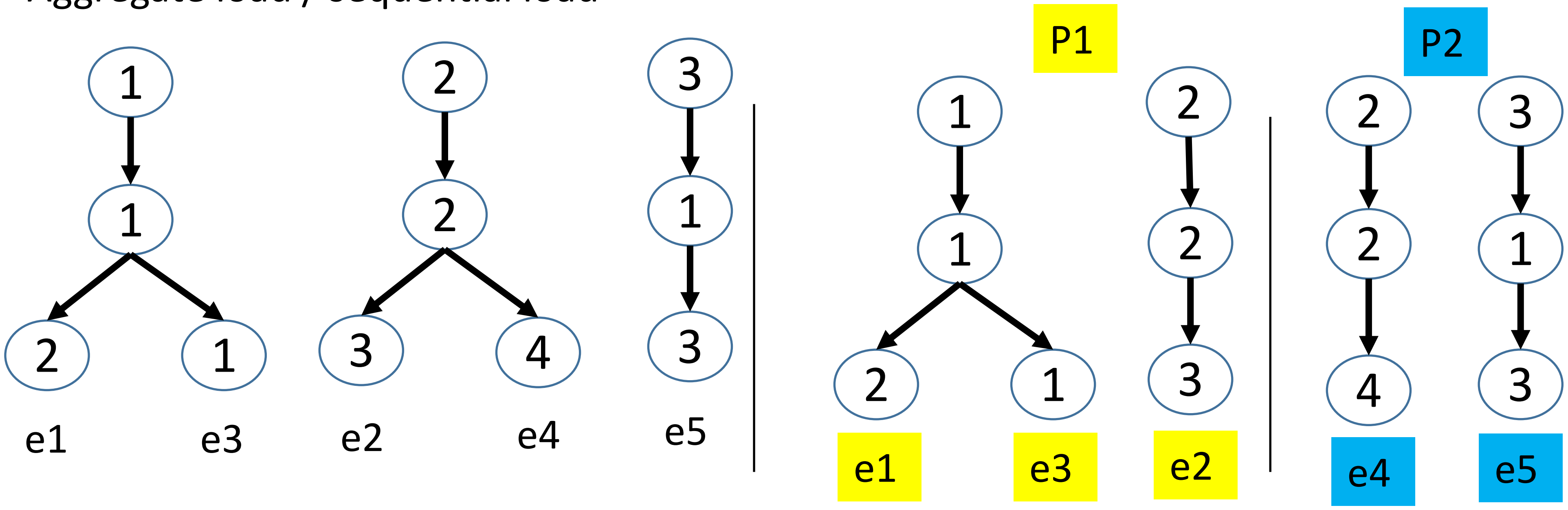
	T	c1	c2	c3
P1	e1	1	1	2
P1	e2	2	2	3
P1	e3	1	1	1
P2	e4	2	2	4
P2	e5	3	1	3

Load Balance:

- Processor should receive an equal number of elements
- Load imbalance = max-load / avg-load

Distributional Redundancy

- Processors build local trees
- Aggregate load / sequential load



- Lite [CCJ +18] : designed for HOOI has good performance for us as well

Experimental Evaluation

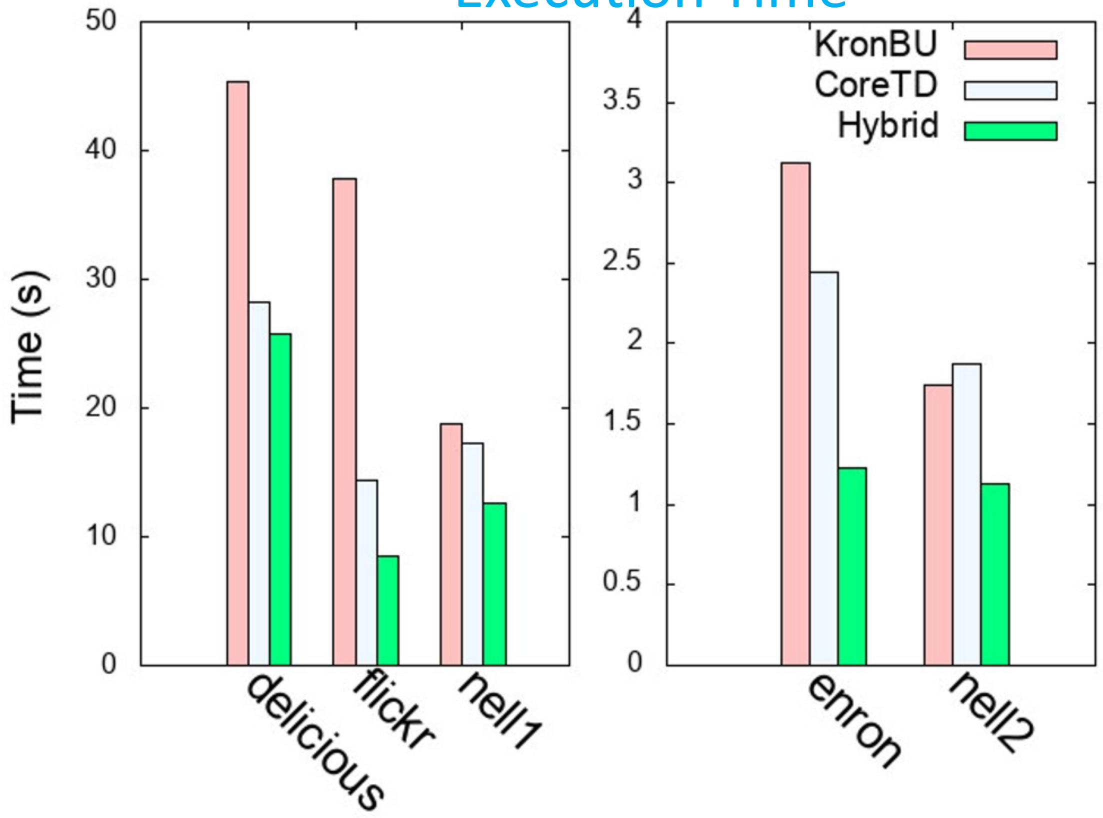
- R92 cluster – 2 to 32 nodes.
- 16 MPI ranks per node, each mapped to a core. So, 32 to 512 MPI ranks
- Dataset : FROSTT repository

Tensor	L_1	L_2	L_3	L_4	nnz
delicious	532K	17.2M	2.4M	1.4K	140M
enron	6K	5K	244K	1K	54M
flickr	319K	28M	1.6M	731	112M
nell1	2.9M	2.1M	25.4M	-	143M
nell2	12K	9K	28K	-	77M

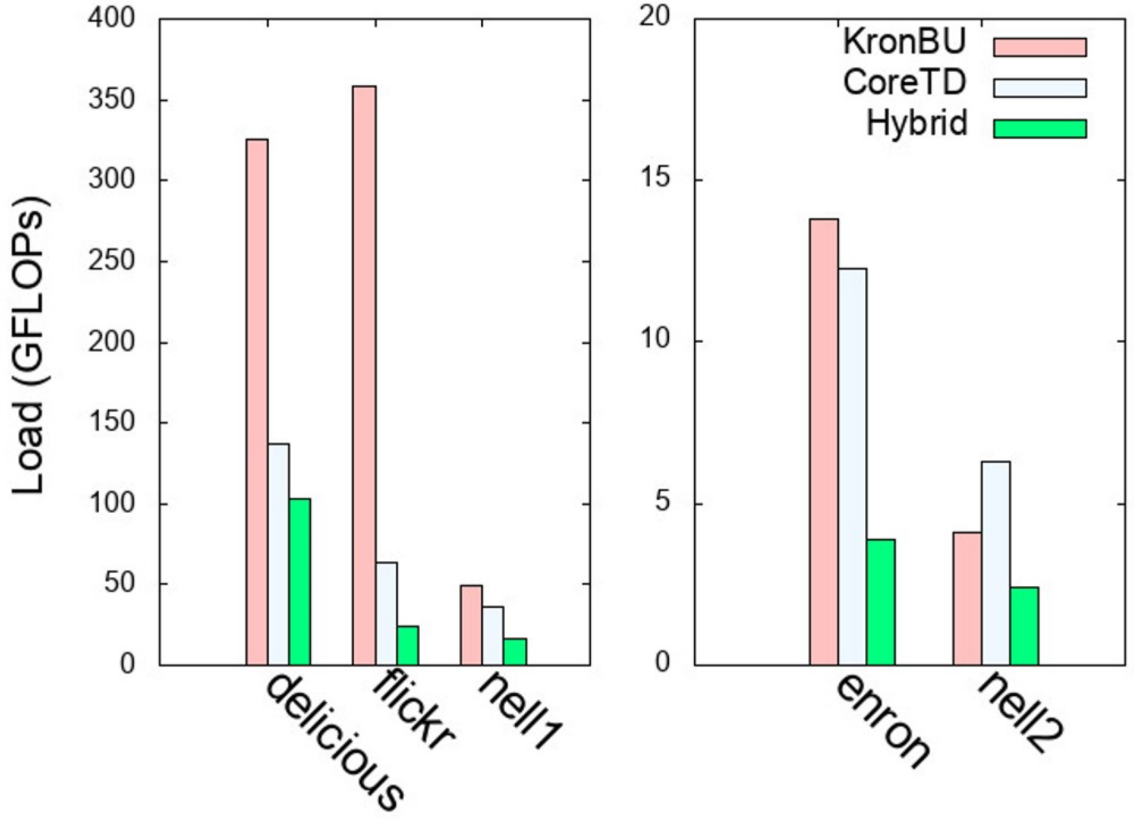
Comparison of Algorithms

- 32 ranks
- Up to 4x improvement over baseline KronBU

Execution Time



Computational Load



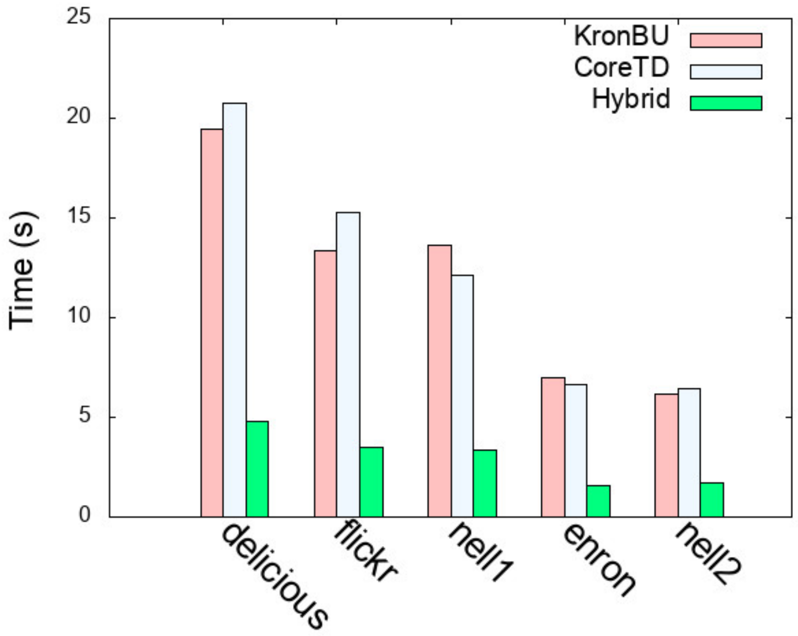
Evaluation of Distribution Policy

- Optimal value for both = 1

	Redundancy			Load imbalance		
-	**32**	**128**	**512**	**32**	**128**	**512**
delicious	1.00	1.00	1.00	1.03	1.08	1.19
flickr	1.00	1.00	1.00	1.03	1.08	1.27
nell1	1.00	1.00	1.00	1.01	1.01	1.03
enron	1.01	1.02	**1.06**	1.06	1.15	**1.36**
nell2	1.00	1.00	1.00	1.01	1.01	1.02

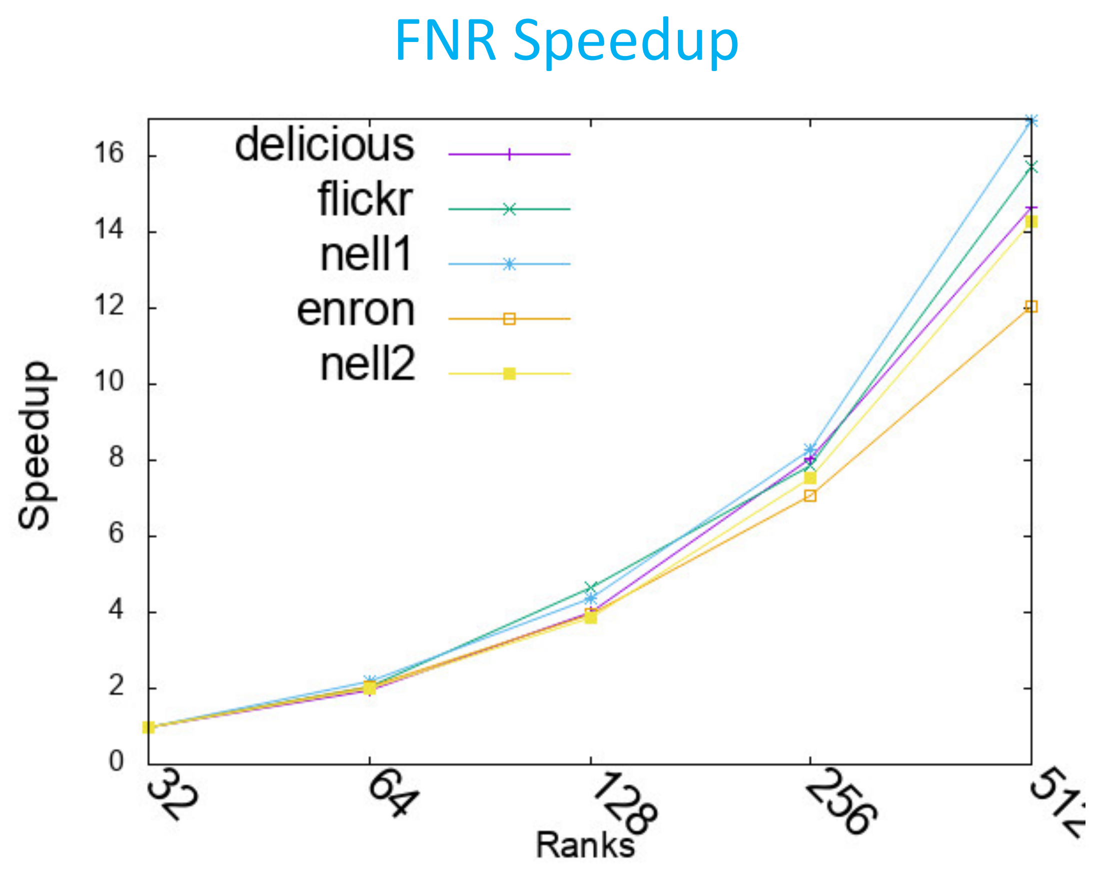
Tree Construction Time

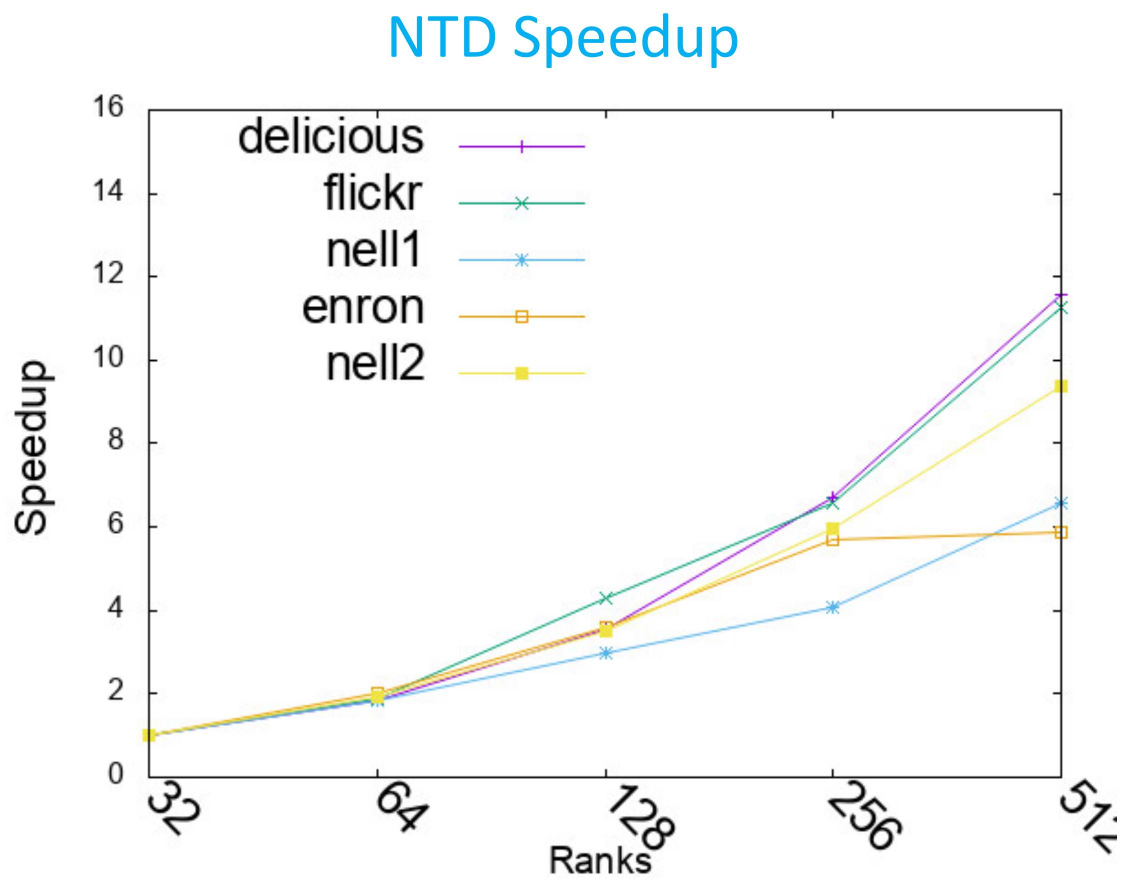
- KronBU and CoreTD – N trees one along each mode
- Hybrid – a single tree



Strong Scaling (for Hybrid)

- Speedup from 32 to 512 ranks
- Ideal speedup = 16x





- Factor matrix transfer time – does not scale

Conclusions

- First distributed implementation of non-negative Tucker decomposition
- Based on MU-NTD procedure
- Hybrid algorithm

Future Work

- Improve factor matrix transfer → better overall scaling
- Other NTD procedures