6.5930/1

Hardware Architectures for Deep Learning

Sparse Matrix Multiplication Accelerator Architecture

April 10, 2024

Joel Emer and Vivienne Sze

Massachusetts Institute of Technology
Electrical Engineering & Computer Science

Goals of Today's Lecture

- Last lecture, how to systematically understand the translation of sparsity into reductions in energy consumption and processing cycles through dataflows that exploit sparsity for convolution.

- Today, how to systematically understand the translation of sparsity into reductions in energy consumption and processing cycles through dataflows that exploit sparsity for matrix multiply.

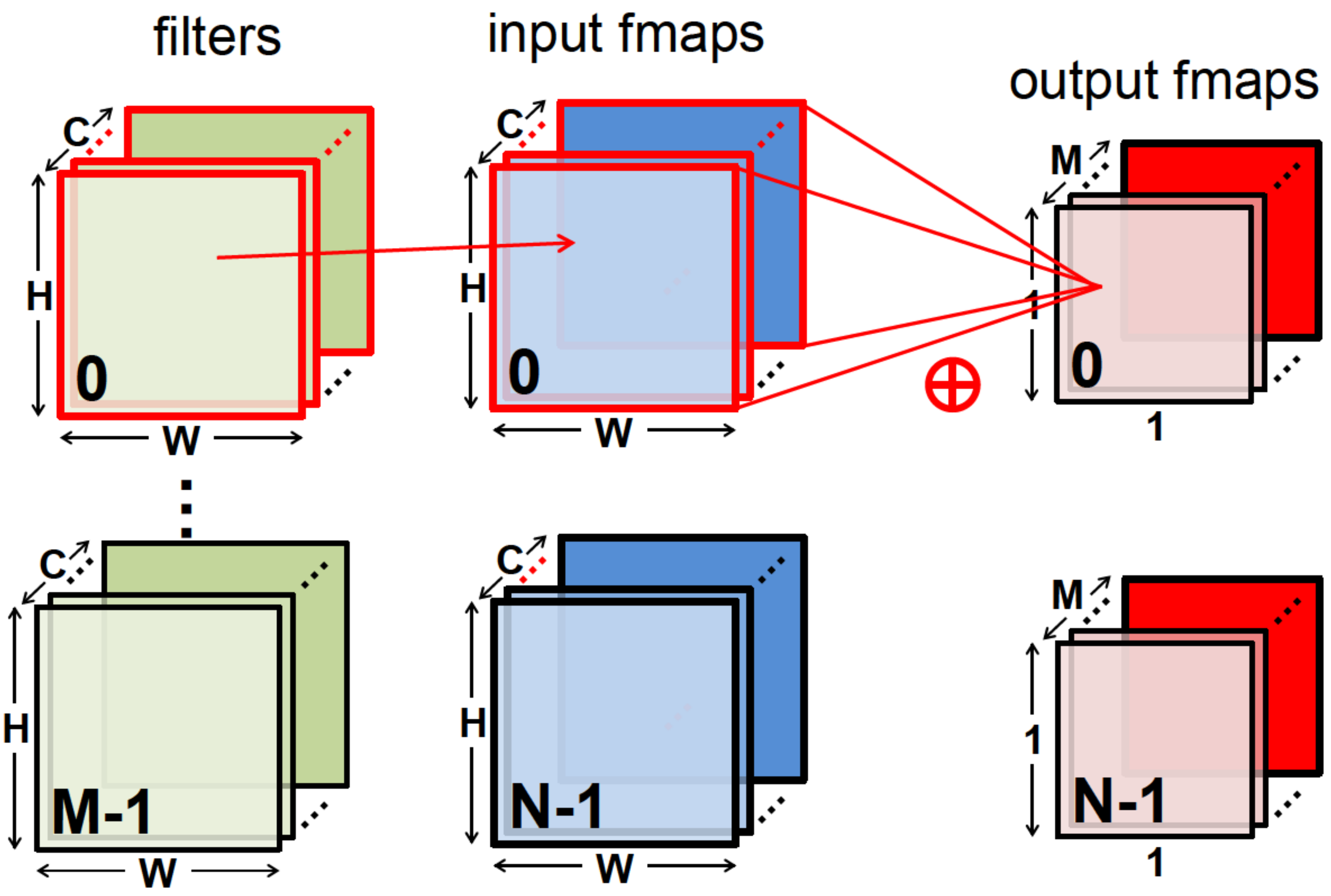
Resources

- **Course notes:** Chapter 8.2 and 8.3

- **Extensor:** Hegde, Pellauer, Crago, Jaleel, Solomonik, Emer, Fletcher, “ExTensor: An Accelerator for Sparse Tensor Algebra”, MICRO 2019
- **OuterSPACE**: *Pal, Beaumont, Park, Amarnath, Feng, Chakrabarti, Kim, Blaauw, Mudge, Dreslinski. “OuterSPACE: An Outer Product Based Sparse Matrix Multiplication Accelerator.” HPCA, 2018.*
- **Gamma:** Zhang, Attaluri, Emer, Sanchez. “Gamma: leveraging Gustavson’s algorithm to accelerate sparse matrix multiplication.” *ASPLOS 2021.*
- **EIE:** Han, Liu, Mao, Pu, Pedram, Horowitz, Dally. “EIE: efficient inference engine on compressed deep neural network”. ISCA 2016.
- **TeAAL:** Nayak, Odemuyiwa, Ugare, Fletcher, Pellauer, Emer. “TeAAL: A Declarative Framework for Modeling Sparse Tensor Accelerators”. Micro 2023.

FC: Exploiting Sparse Inputs & Sparse Weights

Fully-Connected (FC) Layer



Einsum for FC

$$O_{n,m,p,q} = I_{n,c,p+r,q+s} \times F_{m,c,r,s}$$

$$with\ R = H, S = W$$

$$O_{n,m,p,q} = I_{n,c,p+h,q+w} \times F_{m,c,h,w}$$

$$\text{note } P = 1, Q = 1 \rightarrow p = 0, q = 0$$

$$O_{n,m} = I_{n,c,h,w} \times F_{m,c,h,w}$$

$$flatten\ c, h, w \rightarrow chw$$

$$O_{n,m} = I_{n,chw} \times F_{m,chw}$$

FC as Matrix Multiplication

$$O_{n,m} = I_{n,chw} \times F_{m,chw}$$

$$relabel\, n \rightarrow m, m \rightarrow n, chw \rightarrow k$$

$$O_{m,n} = I_{m,k} \times F_{n,k}$$

$$relabel\, O \rightarrow Z, I \rightarrow A, F \rightarrow B$$

$$Z_{m,n} = A_{m,k} \times B_{n,k}$$

Einsum -> Sparse Computation

Einsum – Matrix Multiply

$$Z_{m,n} = A_{m,k} \times B_{n,k}$$

[TACO, Kjolstad et.al., ASE 2017]
[Timeloop, Parashar et.al., ISPASS 2019]

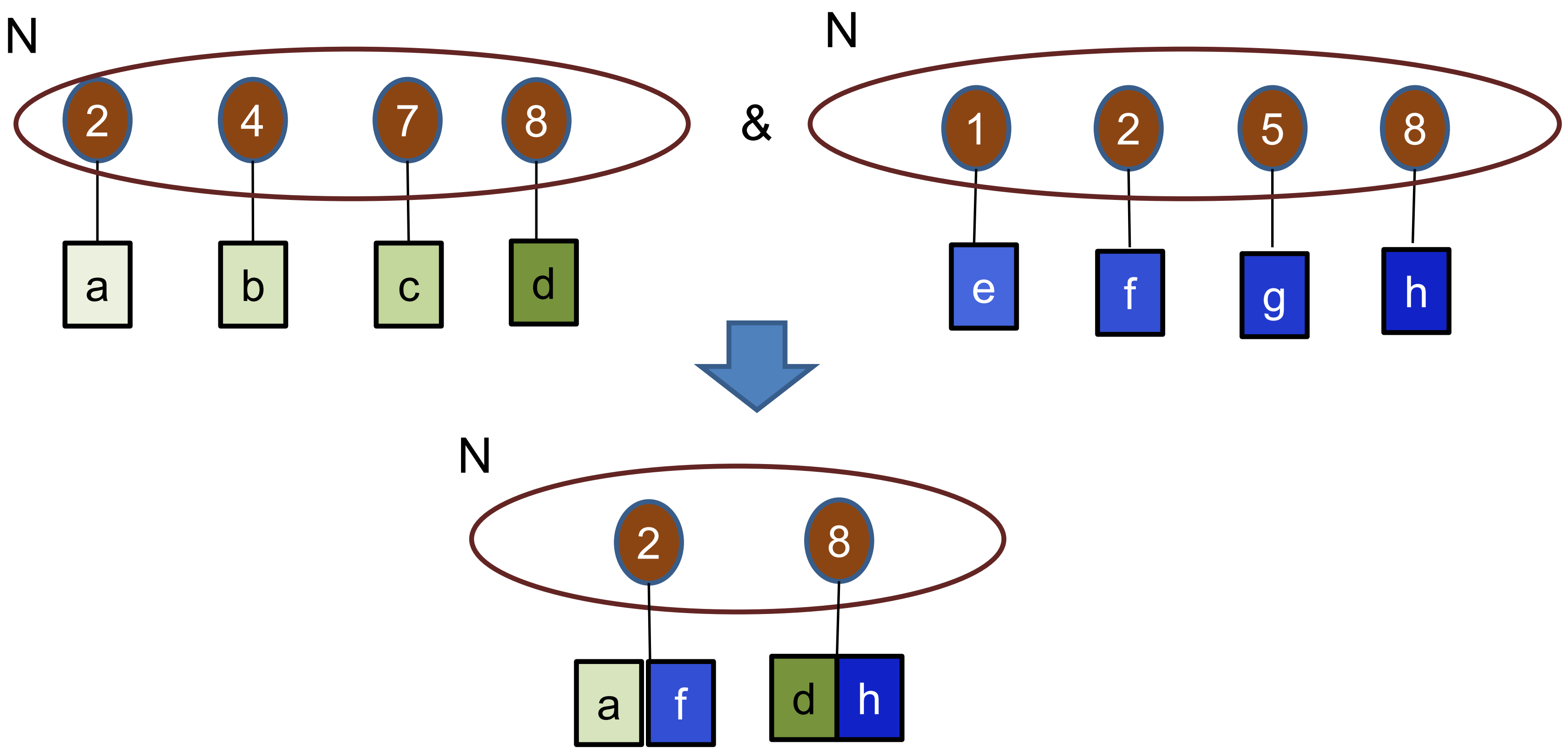
Einsum – Matrix Multiply

$$Z_{m,n} = A_{m,k} \times B_{n,k}$$

- **Shared indices -> intersection (&)**

[TACO, Kjolstad et.al., ASE 2017]
[Timeloop, Parashar et.al., ISPASS 2019]

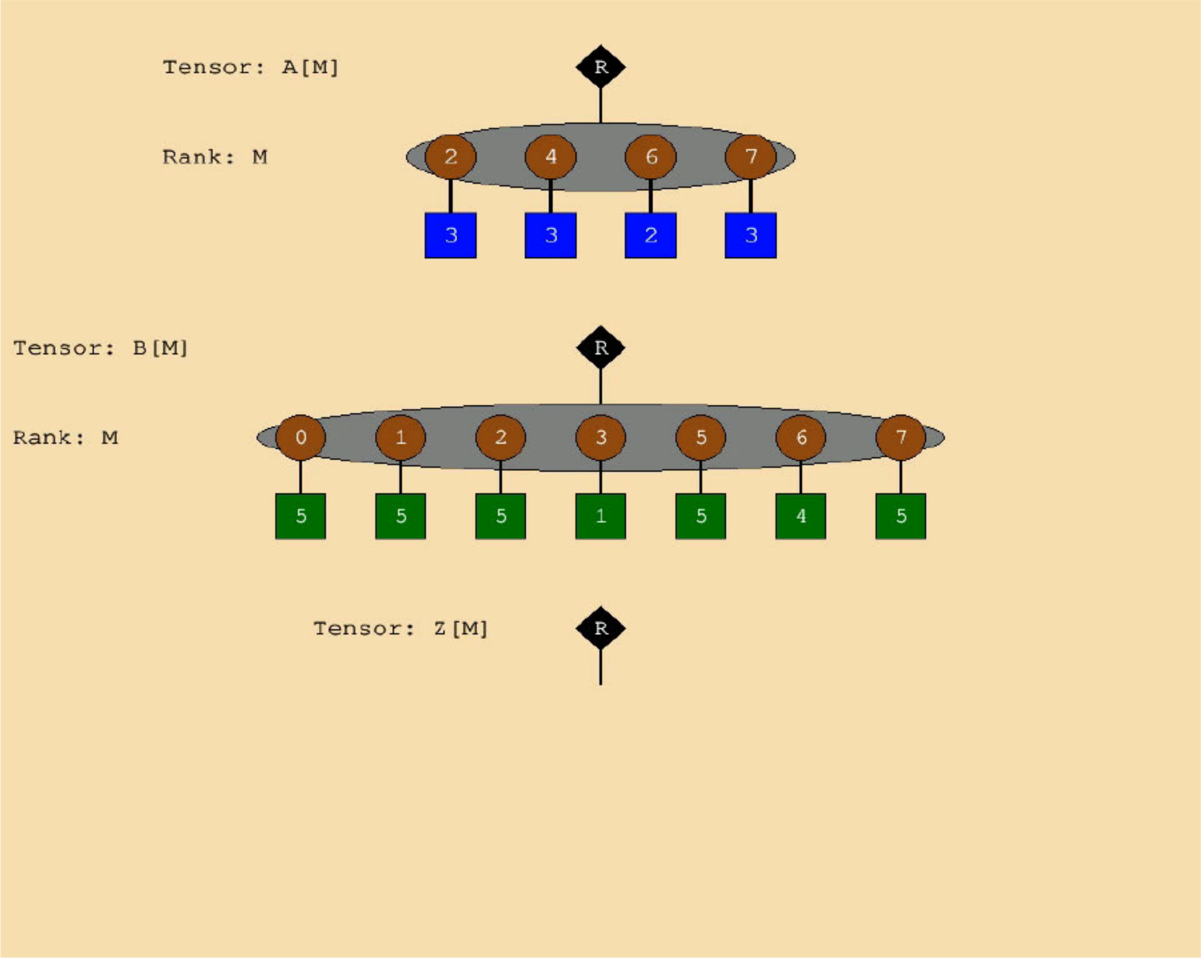
Fiber Intersection



Intersection

A_m x B_m

```
for (m, (a_val, b_val)) in a & b:
    ...
```



Einsum – Matrix Multiply

$$Z_{m,n} = A_{m,k} \times B_{n,k}$$

- **Shared indices -> intersection (&)**
- **Contracted indices -> reduction (+=)**

[TACO, Kjolstad et.al., ASE 2017]
[Timeloop, Parashar et.al., ISPASS 2019]

Einsum – Matrix Multiply

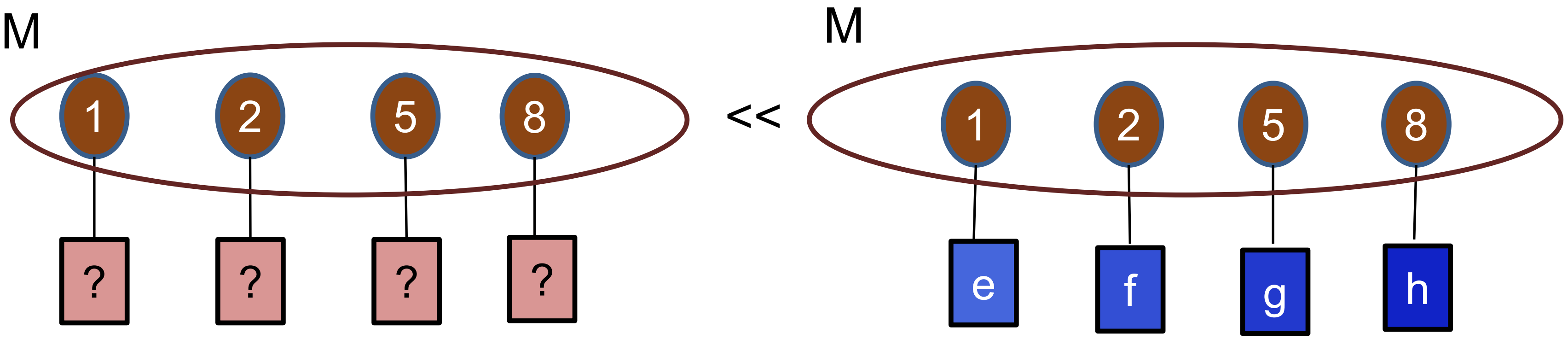
$$Z_{m,n} = A_{m,k} \times B_{n,k}$$

- Shared indices -> intersection (&)
- Contracted indices -> reduction (+=)
- Uncontracted indices -> populate output point (<<)

[TACO, Kjolstad et.al., ASE 2017]
[Timeloop, Parashar et.al., ISPASS 2019]

Populate

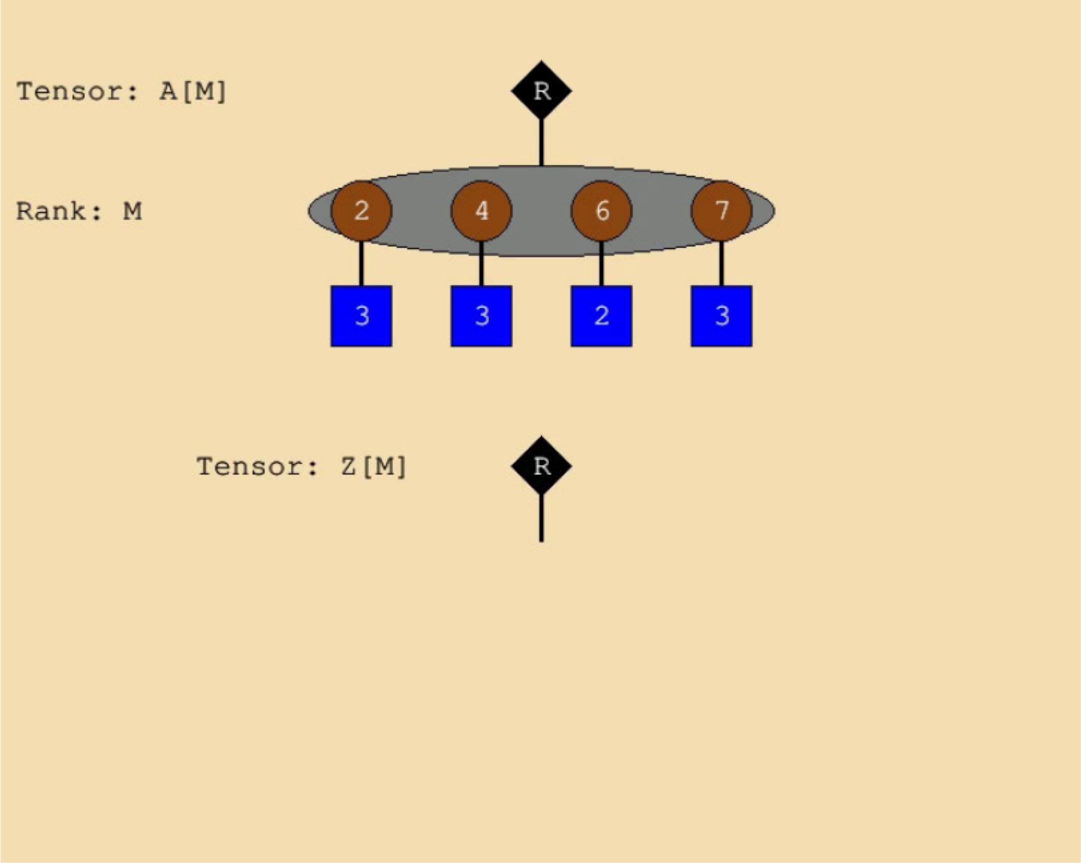
$$Z_m = A_m$$



Populate

$$Z_m = A_m$$

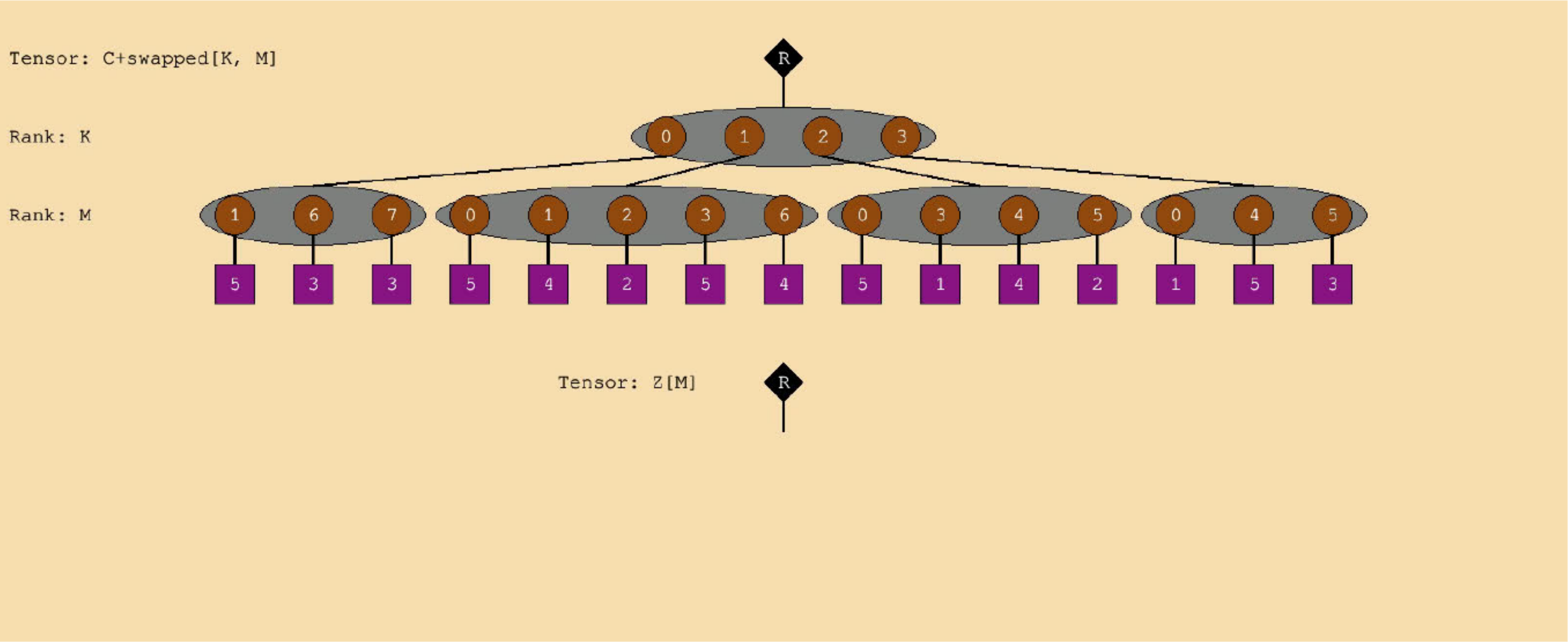
```
for m, (z_ref, a_val) in z << a:
    z_ref <<= a_val
```



Populate+Reduce

$$Z_m = C_{k,m}$$

```
for k, c_m in c:
   for m, (z_ref, c_val) in z_m << c_m:
      z_ref += c_val
```



Einsum - Convolution

$$O_{p,q,m} = I_{c,p+r,q+s} \times F_{m,c,r,s}$$

- Shared indices -> intersection (&)
- Contracted indices -> reduction (+=)
- Uncontracted indices -> populate output point(<<)
- Index arithmetic -> projection

[Extensor, Hegde, et.al., MICRO 2019]

Sparse Matrix Multiply - spMspM

spMspM – Loopnest

$$Z_{m,n} = A_{m,k} \times B_{n,k}$$

Traversal (s to f): M, N, K

```
a_m = Tensor(M,K)   # Input A
b_n = Tensor(N,K)   # Input B
z_m = Tensor(M,N)   # Output Z

for m, (z_n, a_k) in z_m << a_m:
  for n, (z_ref, b_k) in z_n << b_n:
    for k, (a_val, b_val): a_k & b_k
          z_ref += a_val * b_val
```

Populate coord in Z_m for each non-empty coord in A_m

Populate coord in $Z_{m,n}$ for each non-empty coord in B_n

Intersect coords in fibers A_k and B_k

Reduce

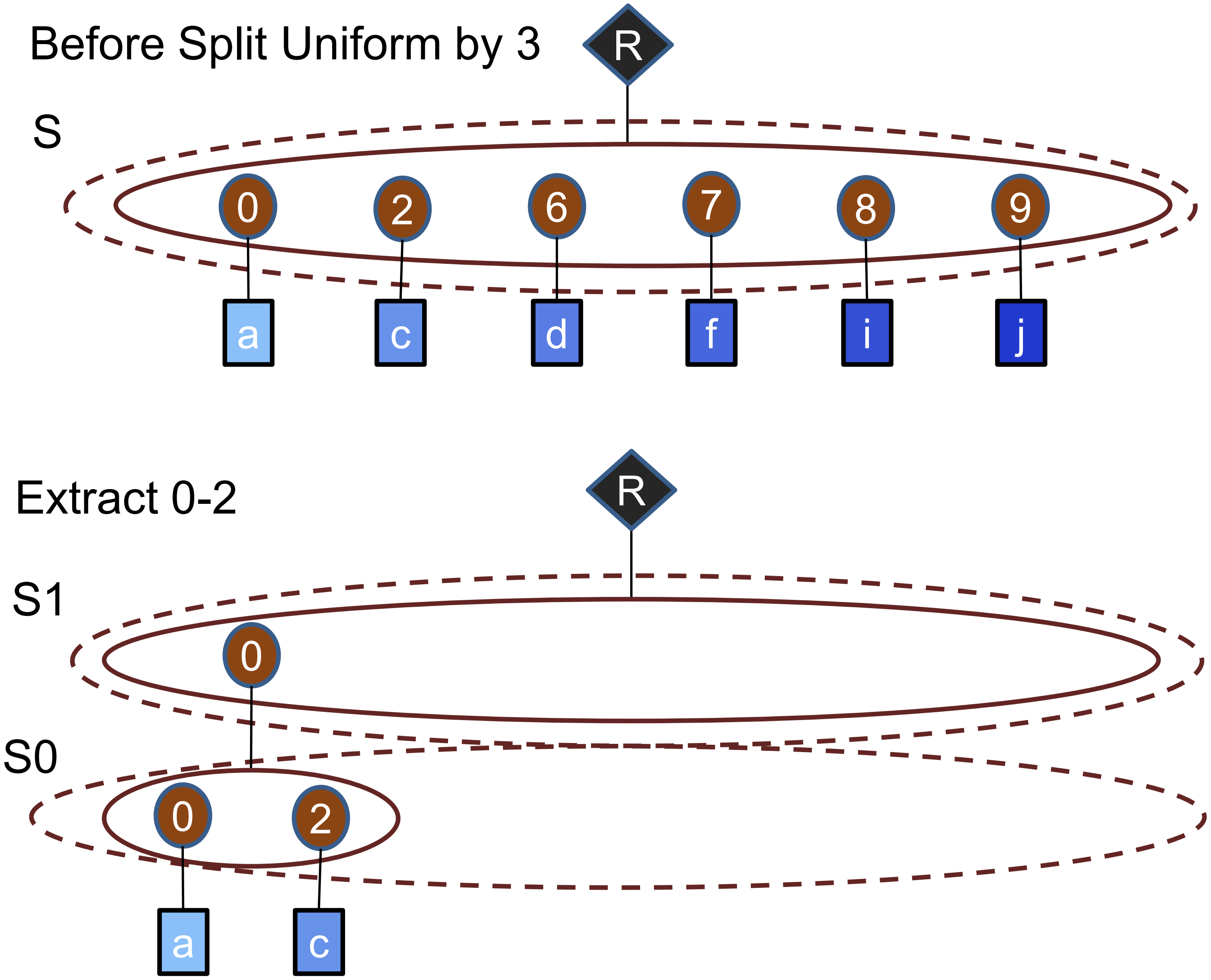
What dataflow is this?

Output stationary

Output Stationary - Animation

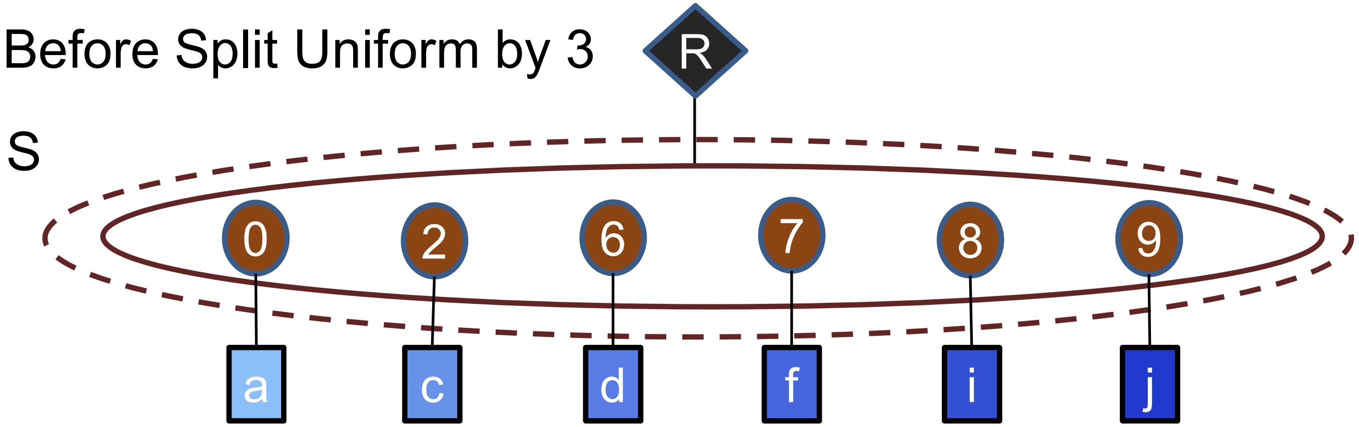
Sparse Data Tiling

Fiber Splitting Uniformly in Coordinate Space



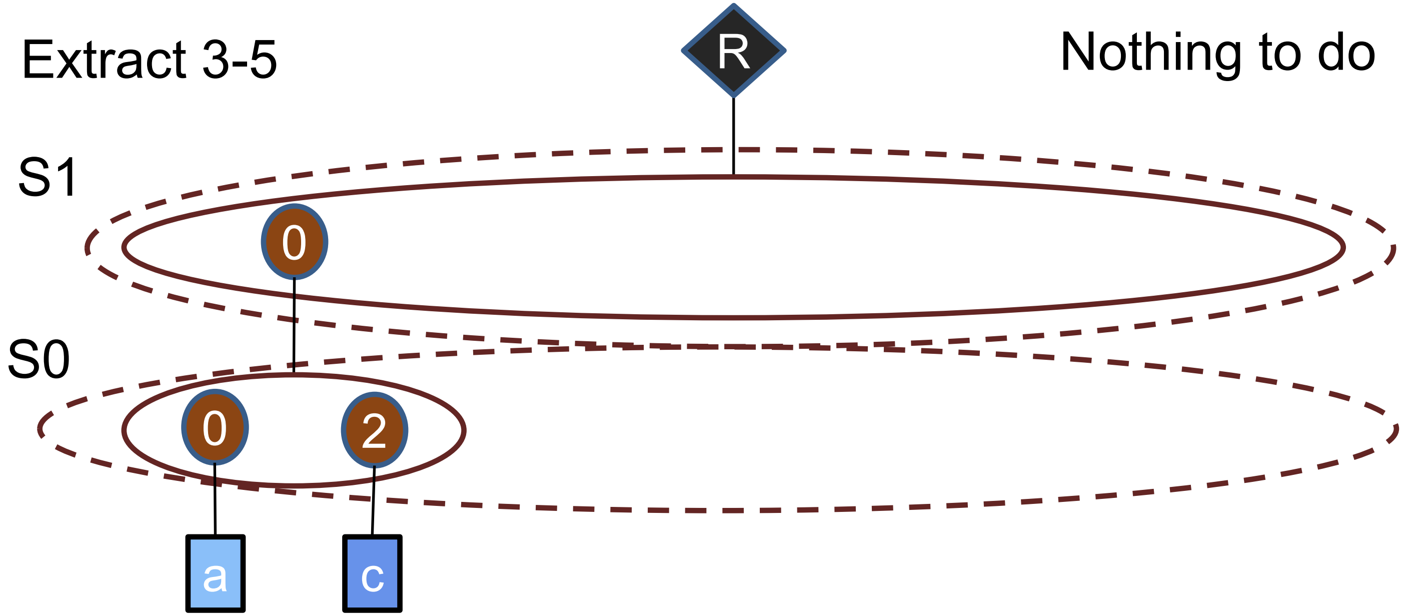
Fiber Splitting Uniformly in Coordinate Space

Before Split Uniform by 3



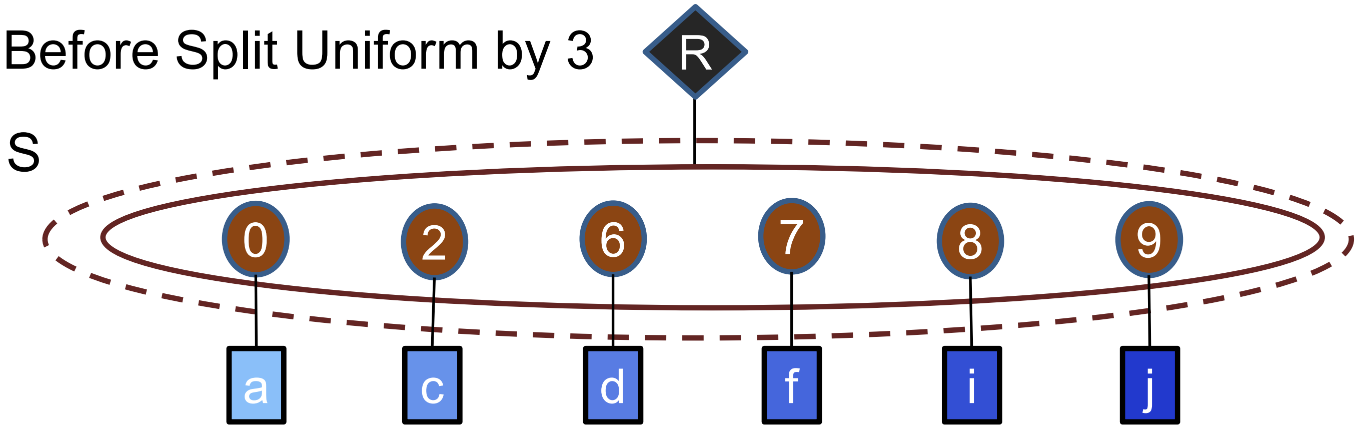
Extract 3-5

Nothing to do

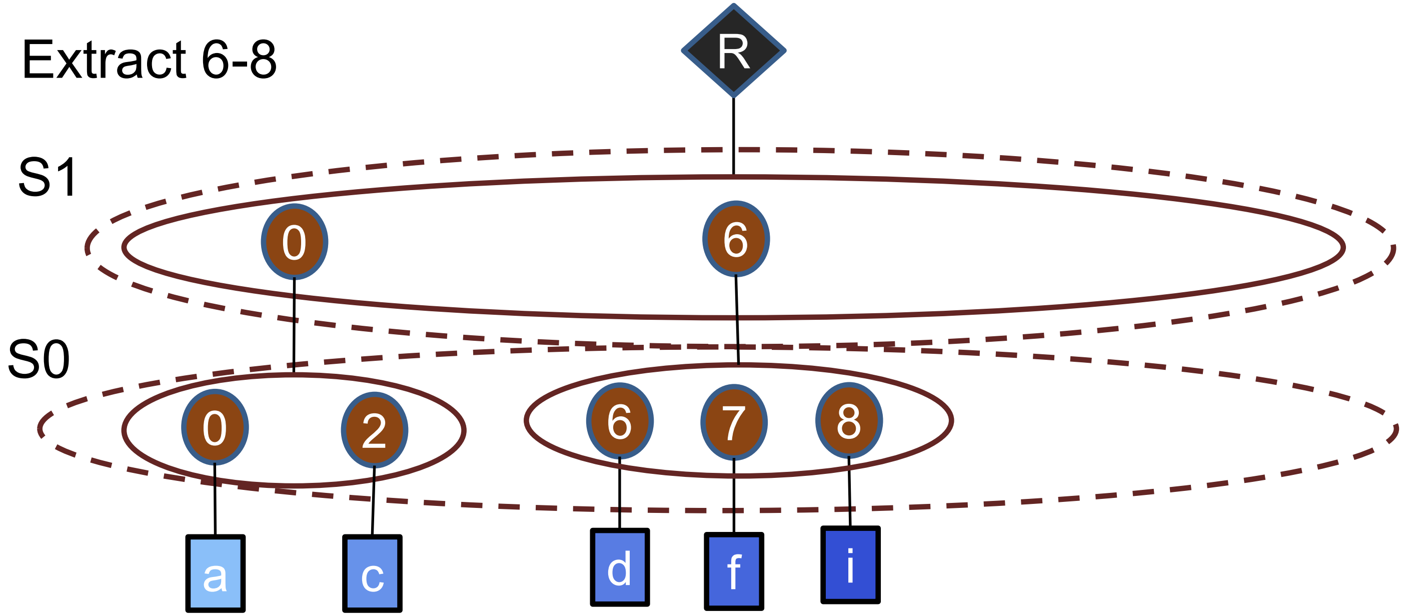


Fiber Splitting Uniformly in Coordinate Space

Before Split Uniform by 3

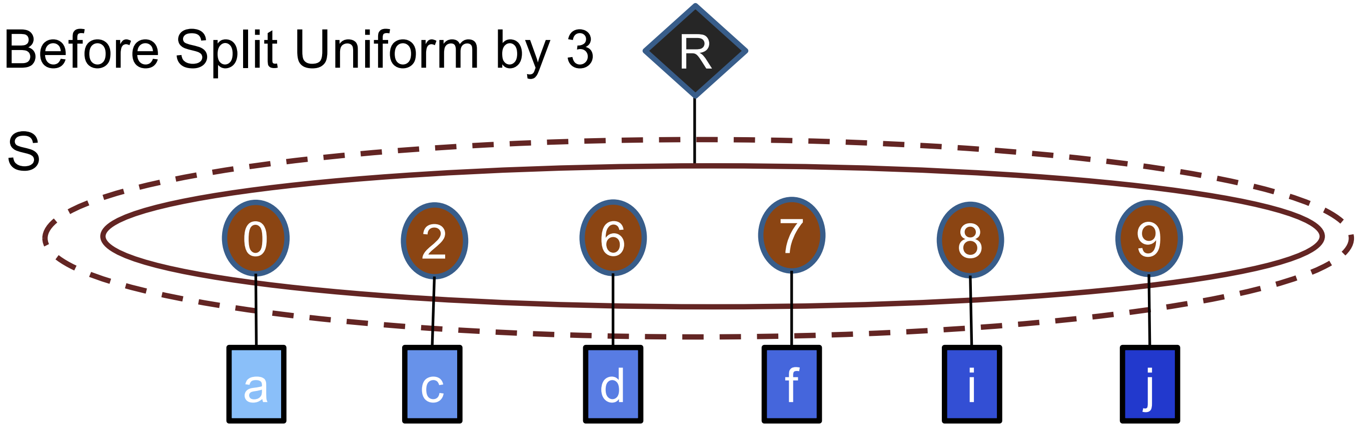


Extract 6-8

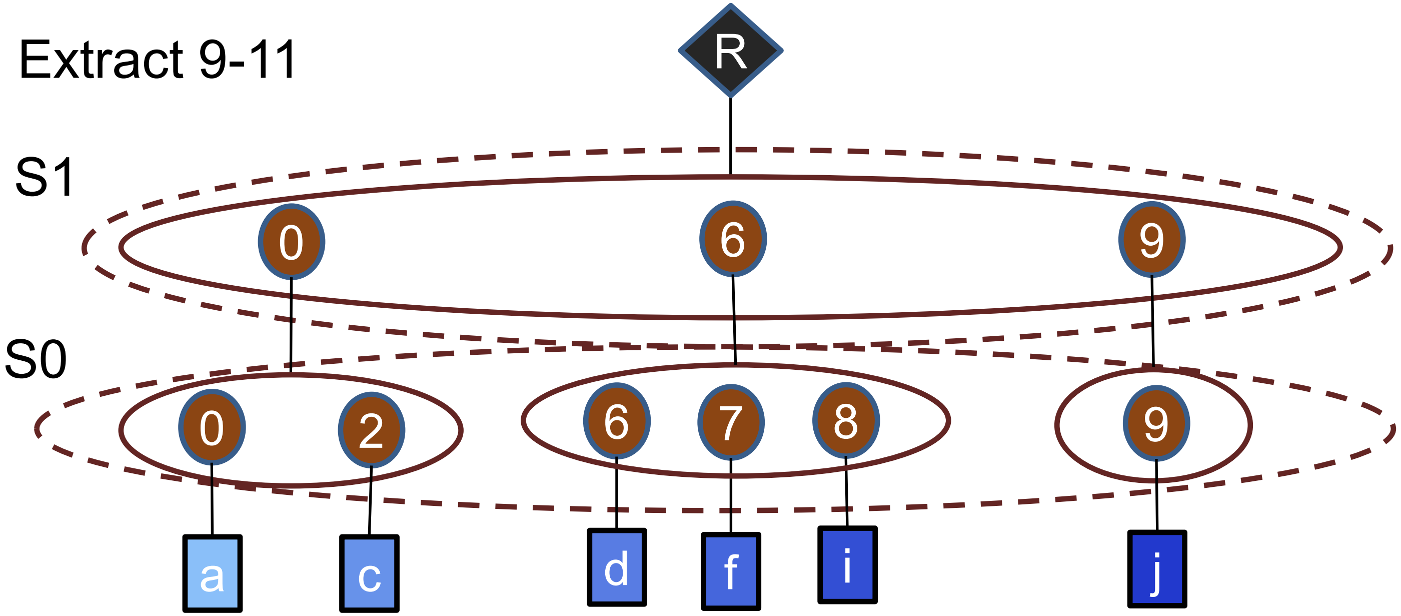


Fiber Splitting Uniformly in Coordinate Space

Before Split Uniform by 3

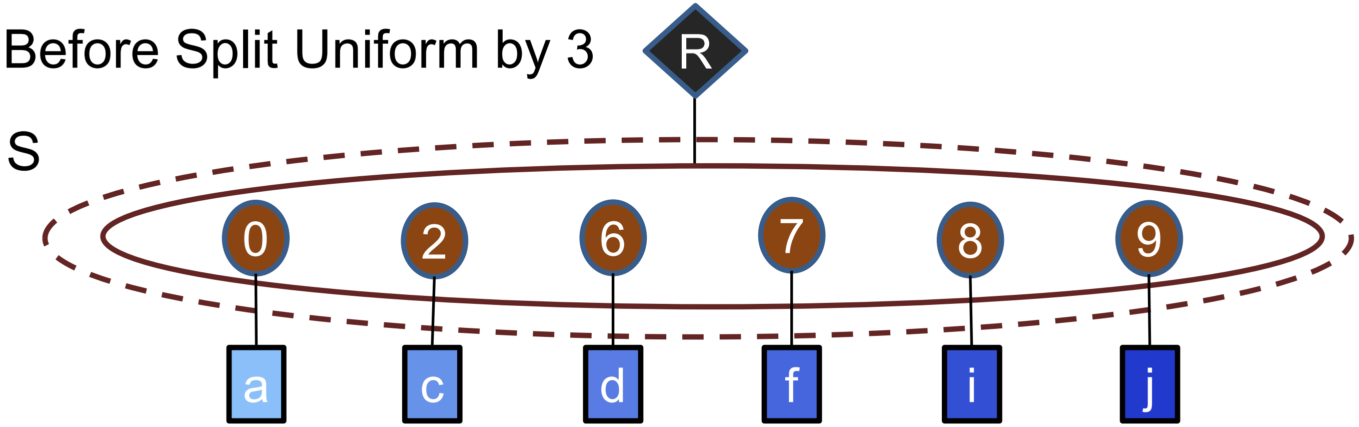


Extract 9-11

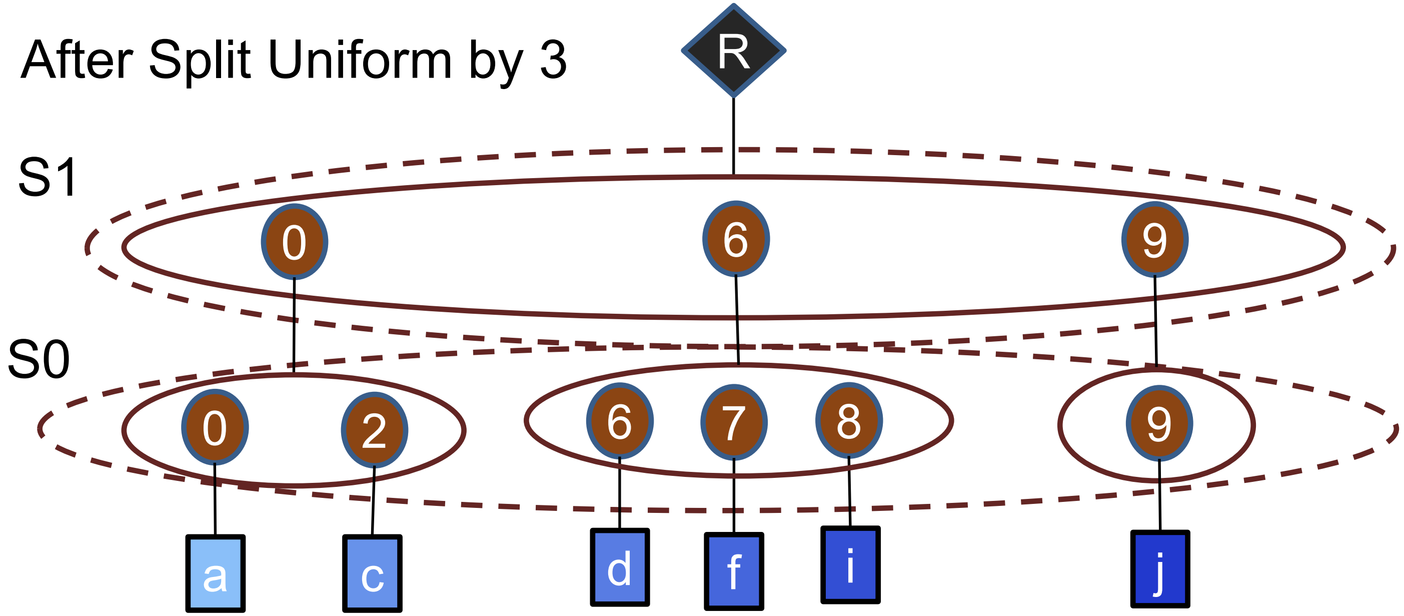


Fiber Splitting Uniformly in Coordinate Space

Before Split Uniform by 3

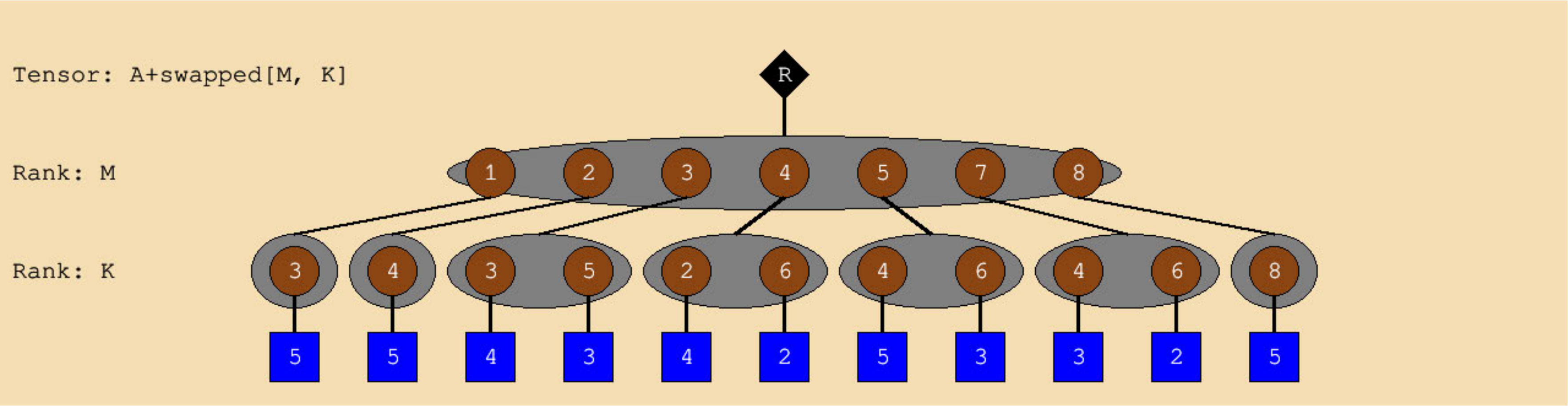


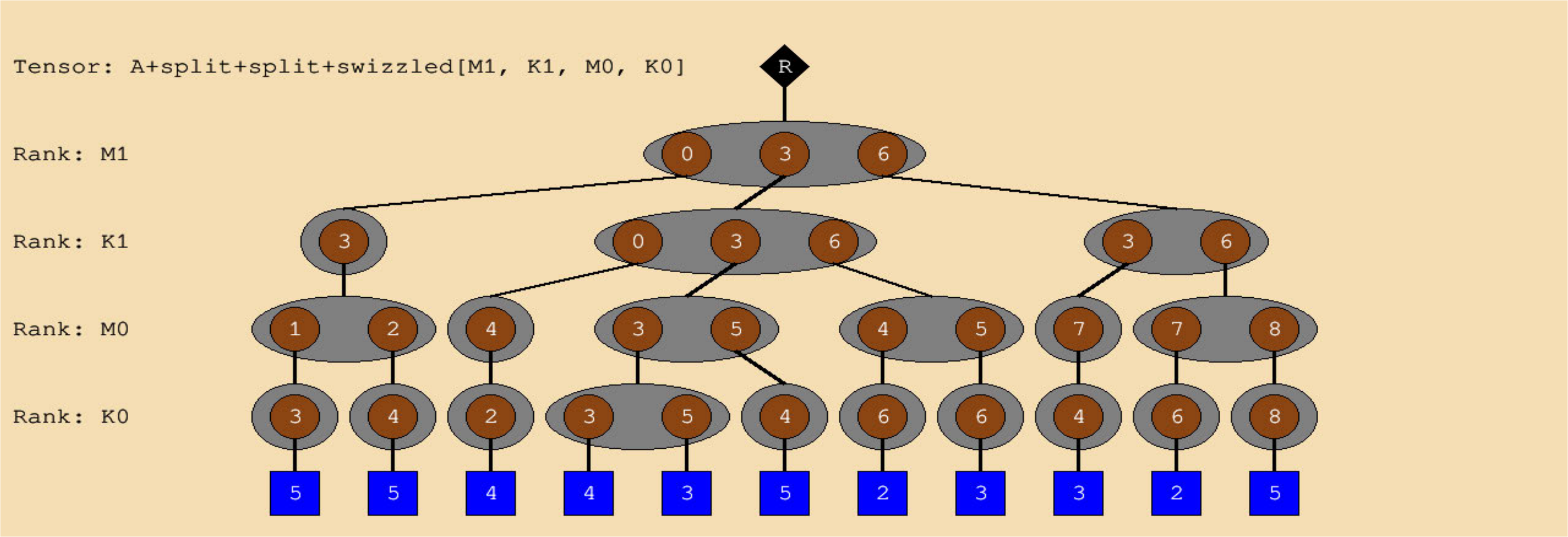
After Split Uniform by 3



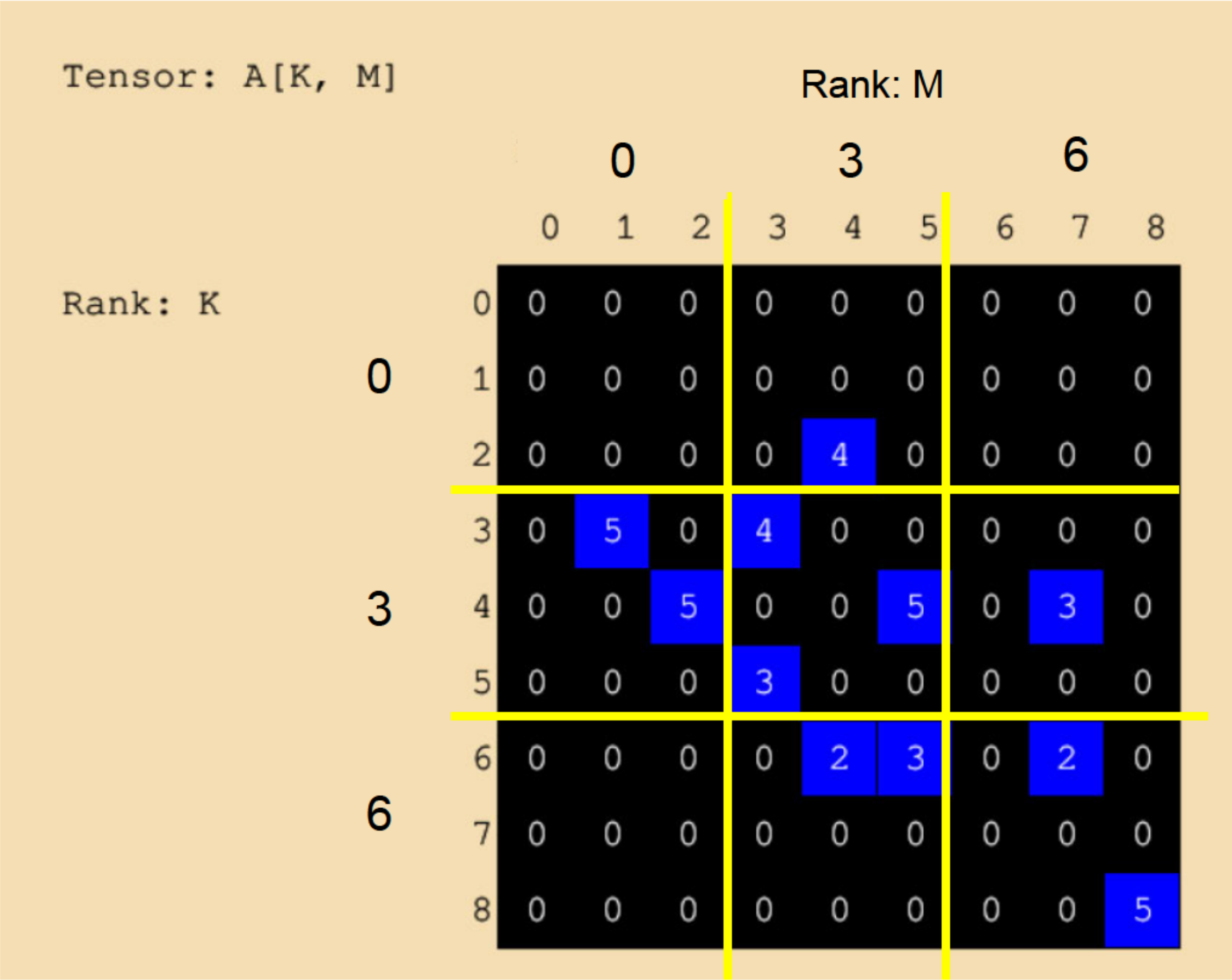
ExTensor

Tensor A – C-Space Split 3x3





Tensor A – Split 3x3 (uncompressed)



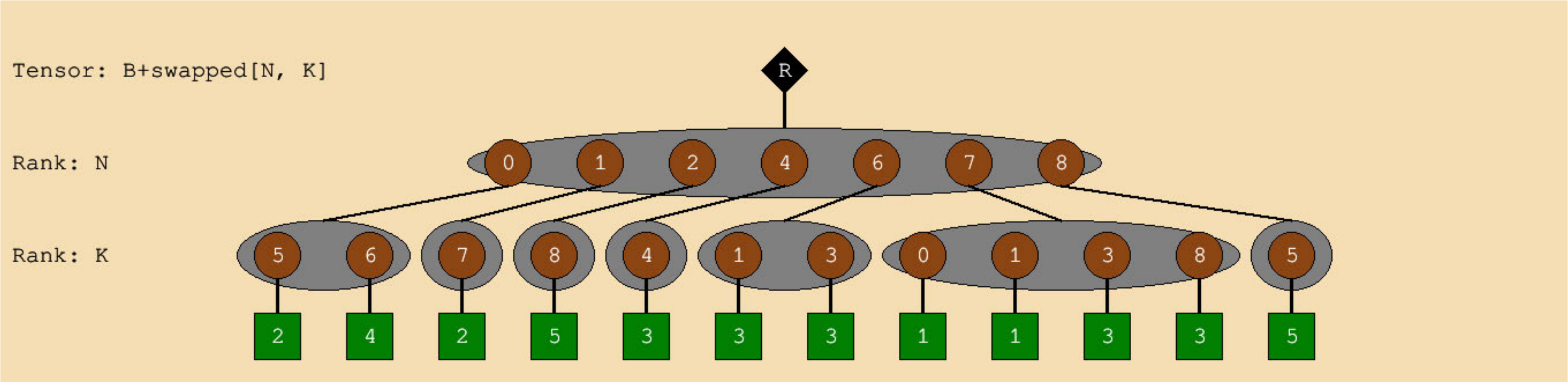
Tensor: A[K, M]

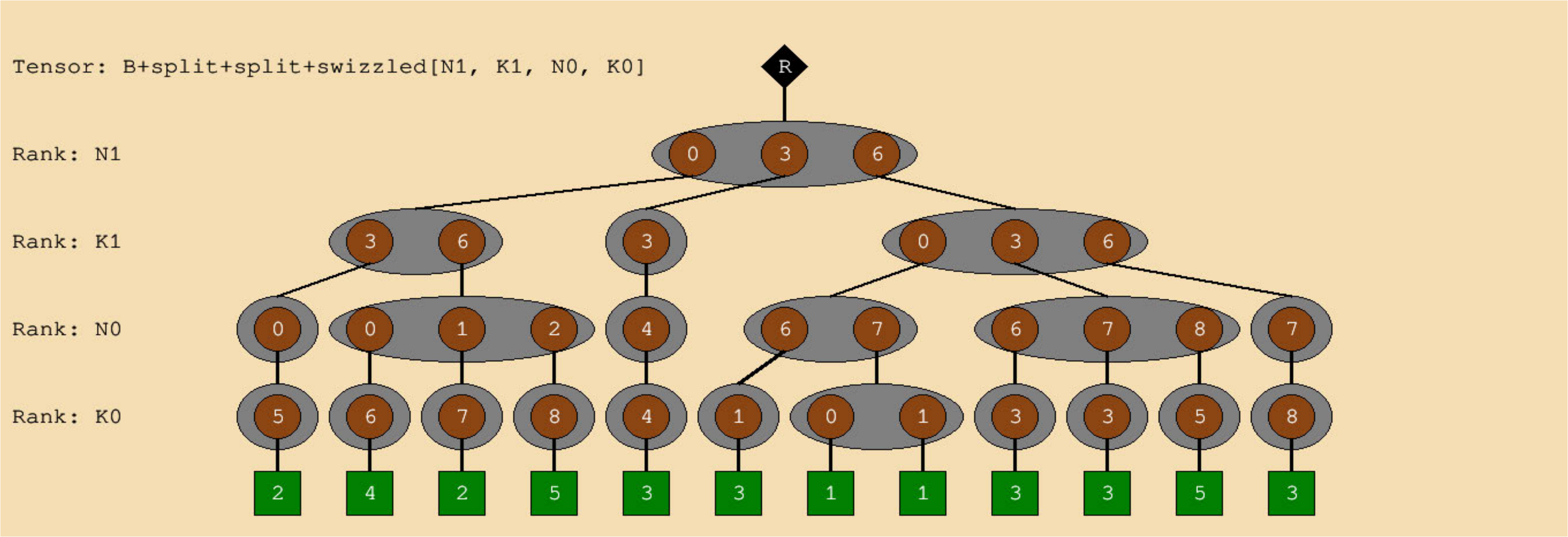
Rank: M

Rank: K

		0			3			6		
		0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0
	2	0	0	0	0	4	0	0	0	0
3	3	0	5	0	4	0	0	0	0	0
	4	0	0	5	0	0	5	0	3	0
	5	0	0	0	3	0	0	0	0	0
6	6	0	0	0	0	2	3	0	2	0
	7	0	0	0	0	0	0	0	0	0
	8	0	0	0	0	0	0	0	0	5

Tensor B – C-space Split – 3x3





Two-level ExTensor – Loop Nest

$$Z_{m1,n1,m0,n0} = A_{m1,k1,m0,k0} \times B_{n1,k1,n0,k0}$$

Schedule (s to f): M1, N1, K1, M0, N0, K0   Parallel K1

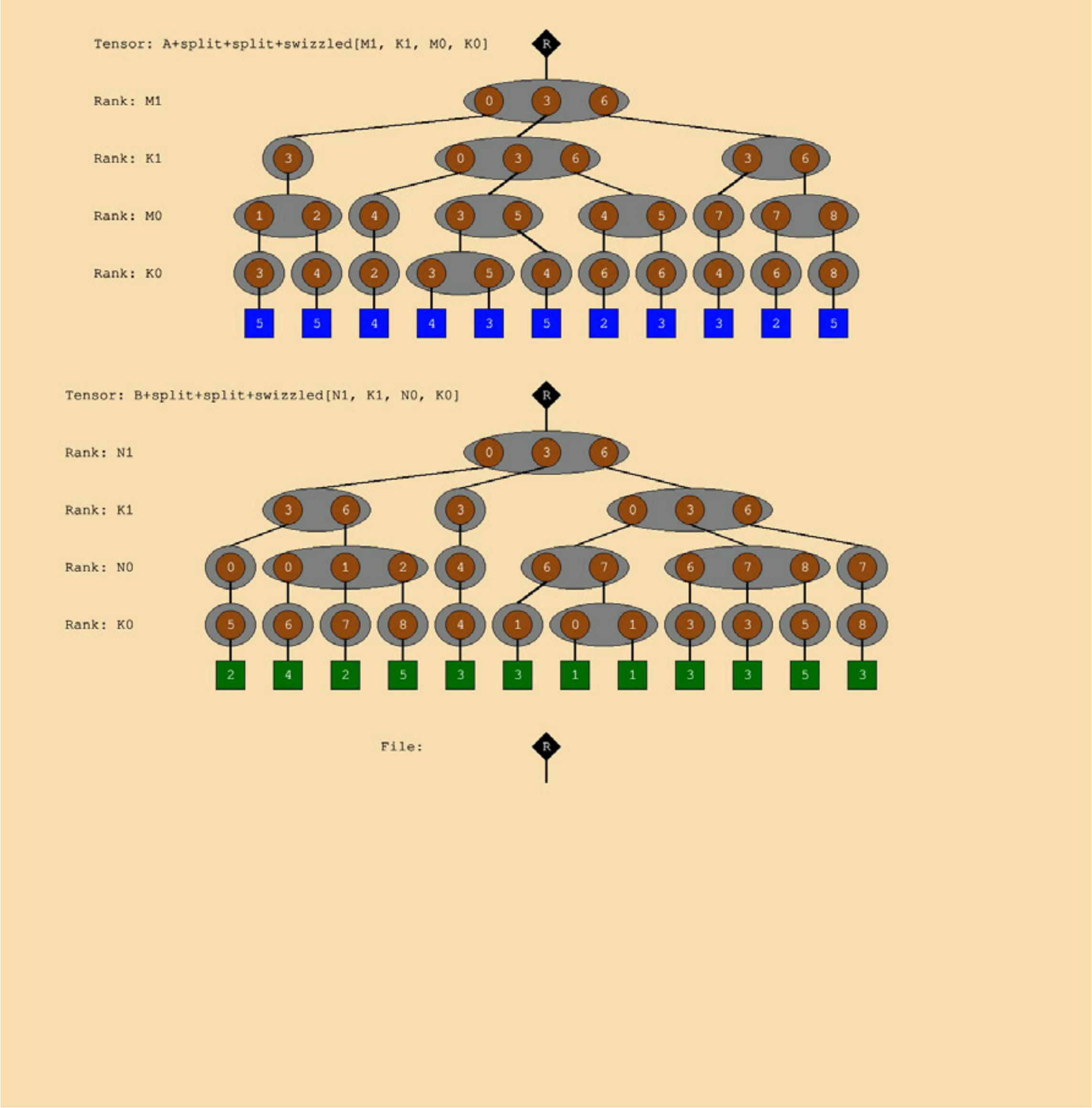
```
a_m1 = Tensor(M1,K1,M0,K0)    # Input A
b_n1 = Tensor(N1,K1,N0,K0)    # Input B
z_m1 = Tensor(M1,N1,M0,N0)    # Output Z

for m1, (z_n1, a_k1) in z_m1 << a_m1:
  for n1, (z_m0, b_k1) in z_n1 << b_n1:
    parallel-for k1, (a_m0, b_n0) in a_k1 & b_k1:
      for m0, (z_n0, a_k0) in z_m0 << a_m0:
        for n0, (z_ref, b_k0) in z_n0 << b_n0:
          for k0, (a_val, b_val)) in a_k0 & b_k0:
              z_ref += a_val * b_val
```

Populate a tile in Z for each tile in A

Intersect tiles in A and B

[Extensor, Hegde, et.al., MICRO 2019]

Two-level ExTensor Animation



Two-level ExTensor - Observations

- Tile corresponds to top two coordinates
- One traversal through the A tiles
- Multiple traversals through the B tiles
- Traversals in A and B stay within a tile and then move to another tile.
- Output tiles created successively
- Note output tile 0,0 is never created.

ExTensor - Concepts

- **Hierarchical Sparse Tiling**
- **Hierarchical Intersection**
- **Optimized intersection unit**

OuterSPACE

OuterSPACE - Einsum

$$Z_{m,n} = A_{m,k} \times B_{n,k}$$

$$\Downarrow$$

$$T_{k,m,n} = A_{k,m} \times B_{k,n}$$

$$Z_{m,n} = T_{m,n,k}$$

Note: Indices rearranged for improved readability

OuterSPACE – Einsum+Schedule

$$T_{k,m,n} = A_{k,m} \times B_{k,n}$$

Outer Product

Loop order (s to f) K, M, N

Parallelize across M

$$Z_{m,n} = T_{m,n,k}$$

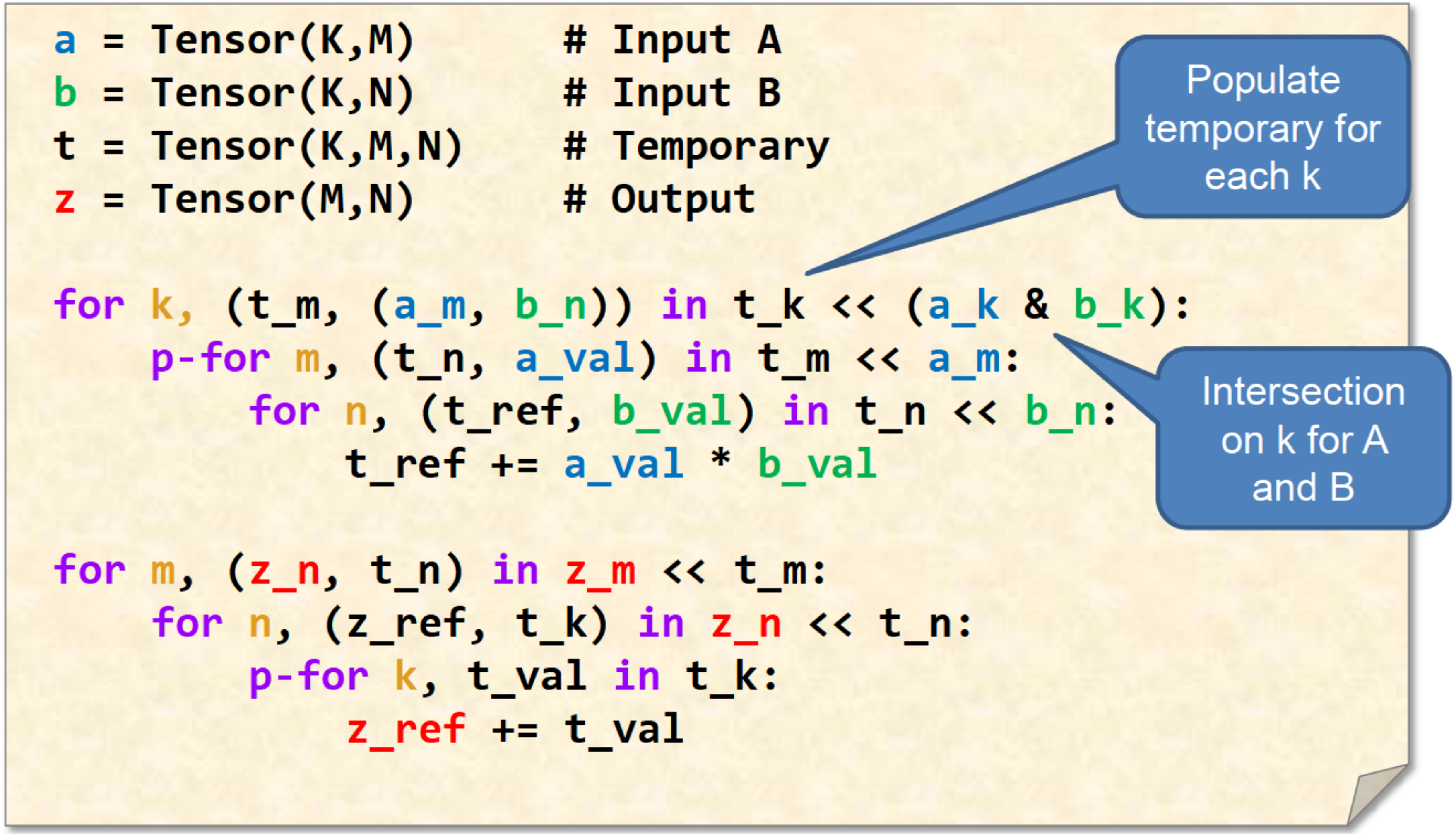
Loop order (s to f): M, N, K

Parallelize across: K

Note: Indices rearranged for improved readability

*Tiling not modeled

OuterSPACE – Loopnest

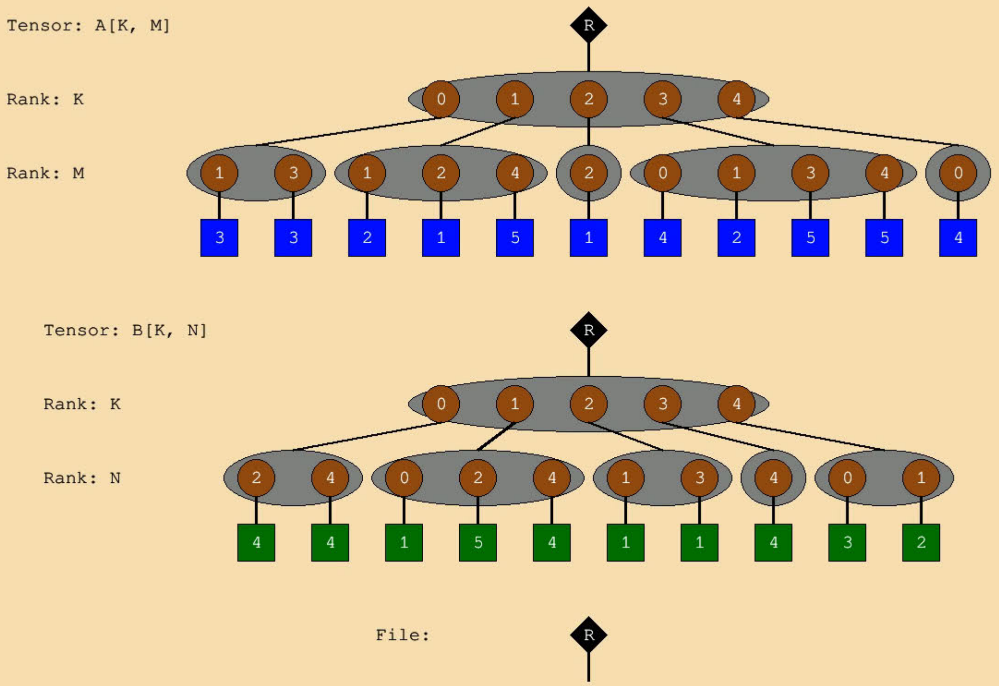


```
a = Tensor(K,M)       # Input A
b = Tensor(K,N)       # Input B
t = Tensor(K,M,N)     # Temporary
z = Tensor(M,N)       # Output

for k, (t_m, (a_m, b_n)) in t_k << (a_k & b_k):
    p-for m, (t_n, a_val) in t_m << a_m:
        for n, (t_ref, b_val) in t_n << b_n:
            t_ref += a_val * b_val

for m, (z_n, t_n) in z_m << t_m:
    for n, (z_ref, t_k) in z_n << t_n:
        p-for k, t_val in t_k:
            z_ref += t_val
```

Populate temporary for each k

Intersection on k for A and B

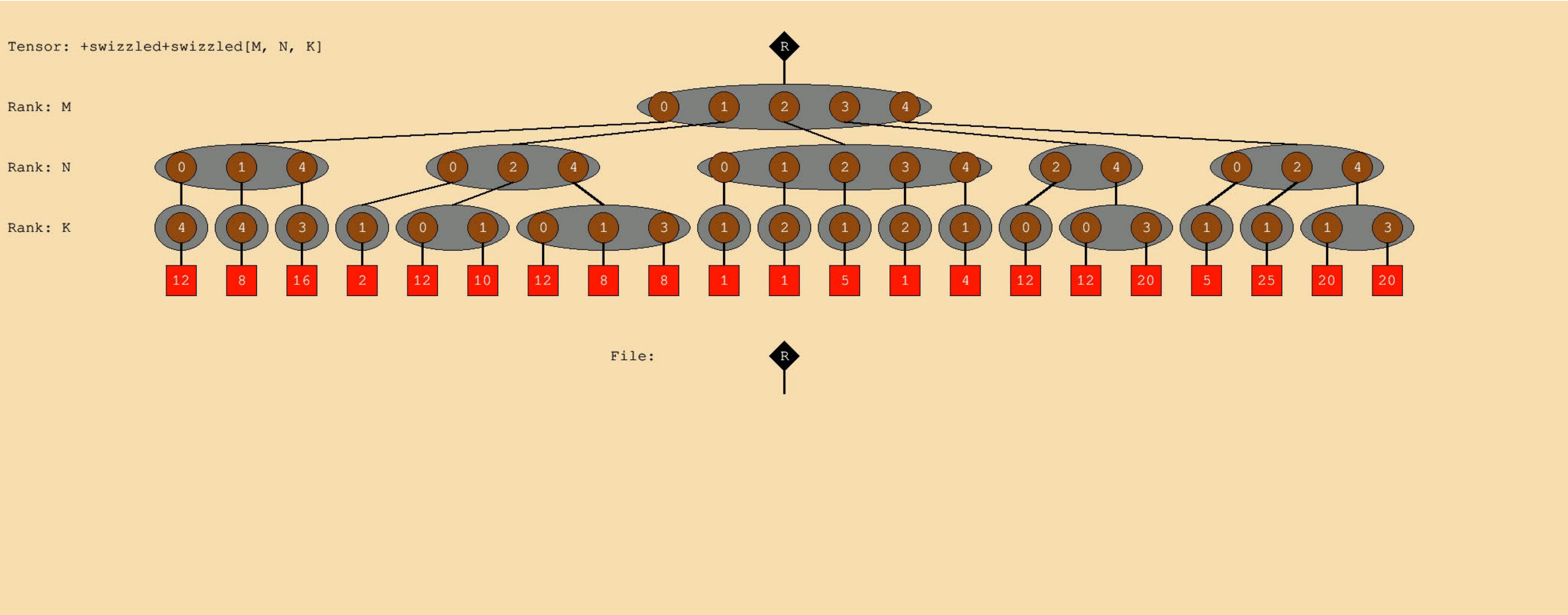
OuterSPACE – Step 1



OuterSPACE – Step 1 - Observations

- **Concordant traversal of B**
  - **with multicast use of a B_n element in a step**
- **Concordant traversal of A**
  - **with parallel access to elements in A_m fiber**
- **Works on one element of T_k fiber of T matrix at a time**
- **Parallel append traversal to multiple T_n fibers of T matrix**

OuterSPACE – Step 2

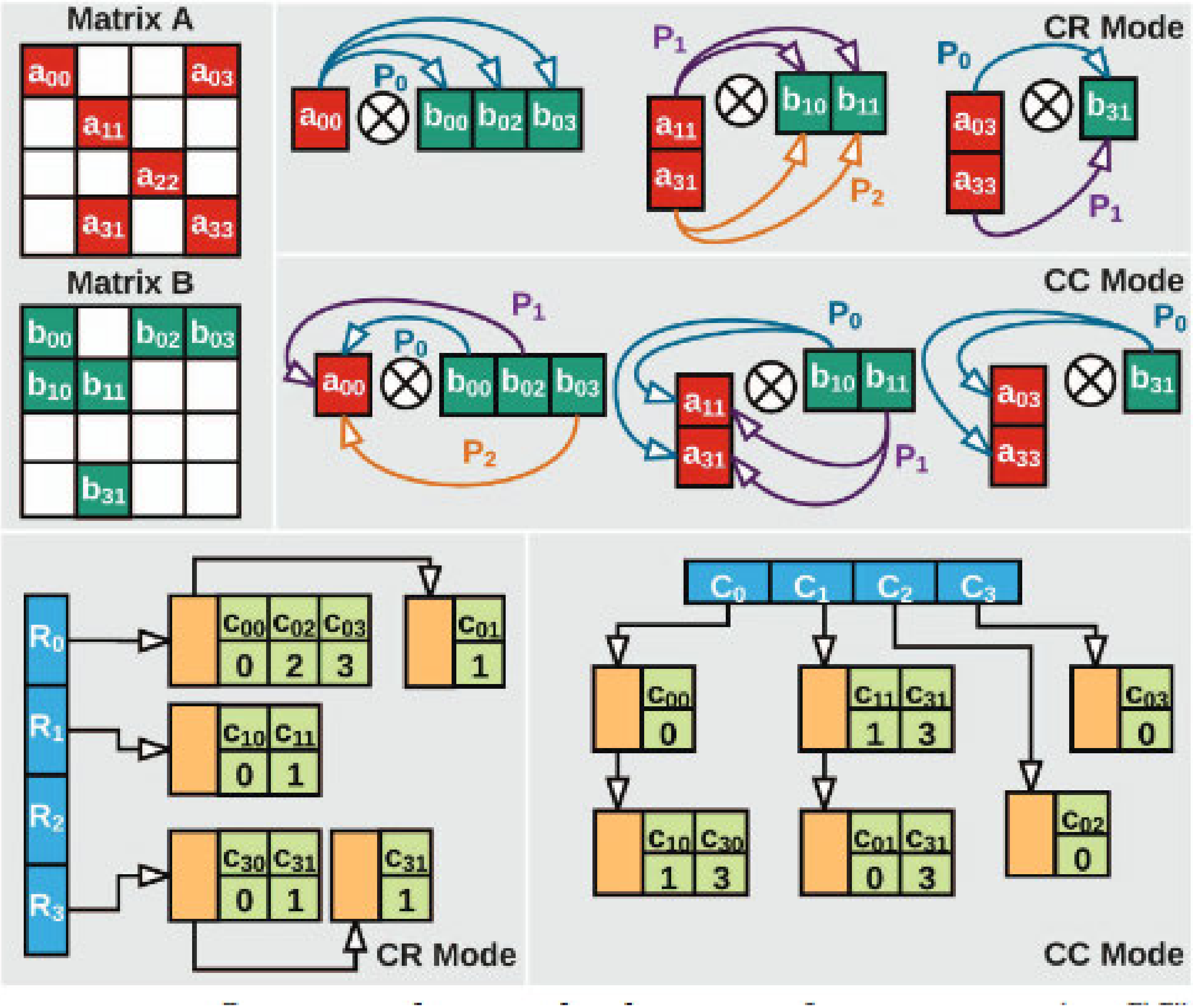


OuterSPACE – Step 2 - Observations

- **Concordant traversal of T tensor**
  - **with parallel access to elements in T_k fiber**
- **Concordant traversal of A**
  - **with parallel access to A_m fiber**
- **Works on one output K matrix at a time**
- **Parallel append traversal of Z matrix**

- **But creation order of T matrix (K,M,N) is different than consumption order (M,N,K)!**

OuterSPACE - Design



[OuterSPACE, Pal, et.al., HPCA 2018]

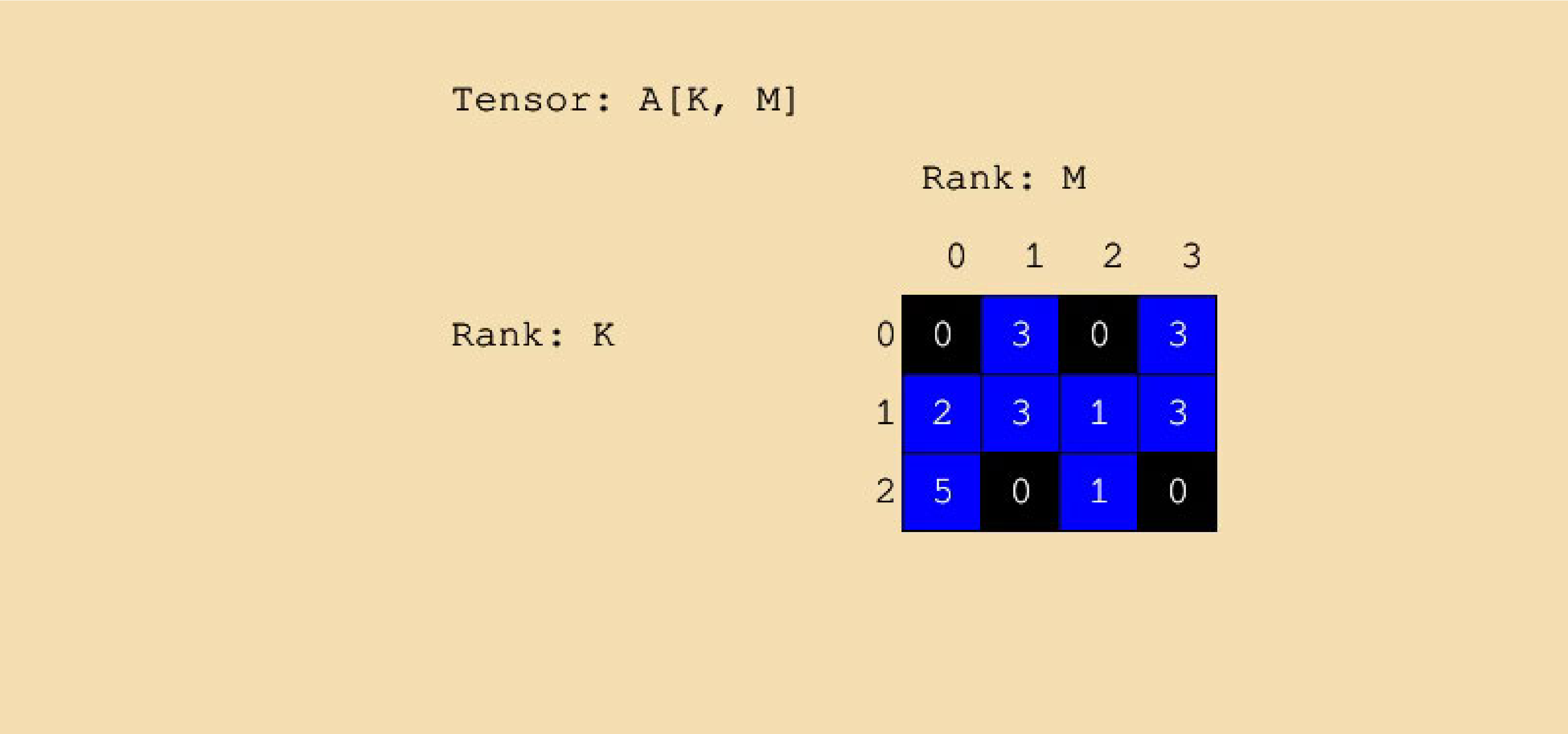
OuterSPACE - Concepts

- Two step process: partial output creation then reducing partial outputs
- Create multiple partial output tiles using outer product
- Efficient format for different traversal orders on creation/consumption of partial result matrices.

Traversing Sparse Tensors

Concordant Traversal - Uncompressed

Traversal order (slowest to fastest): K, M



```
Tensor: A[K, M]
```

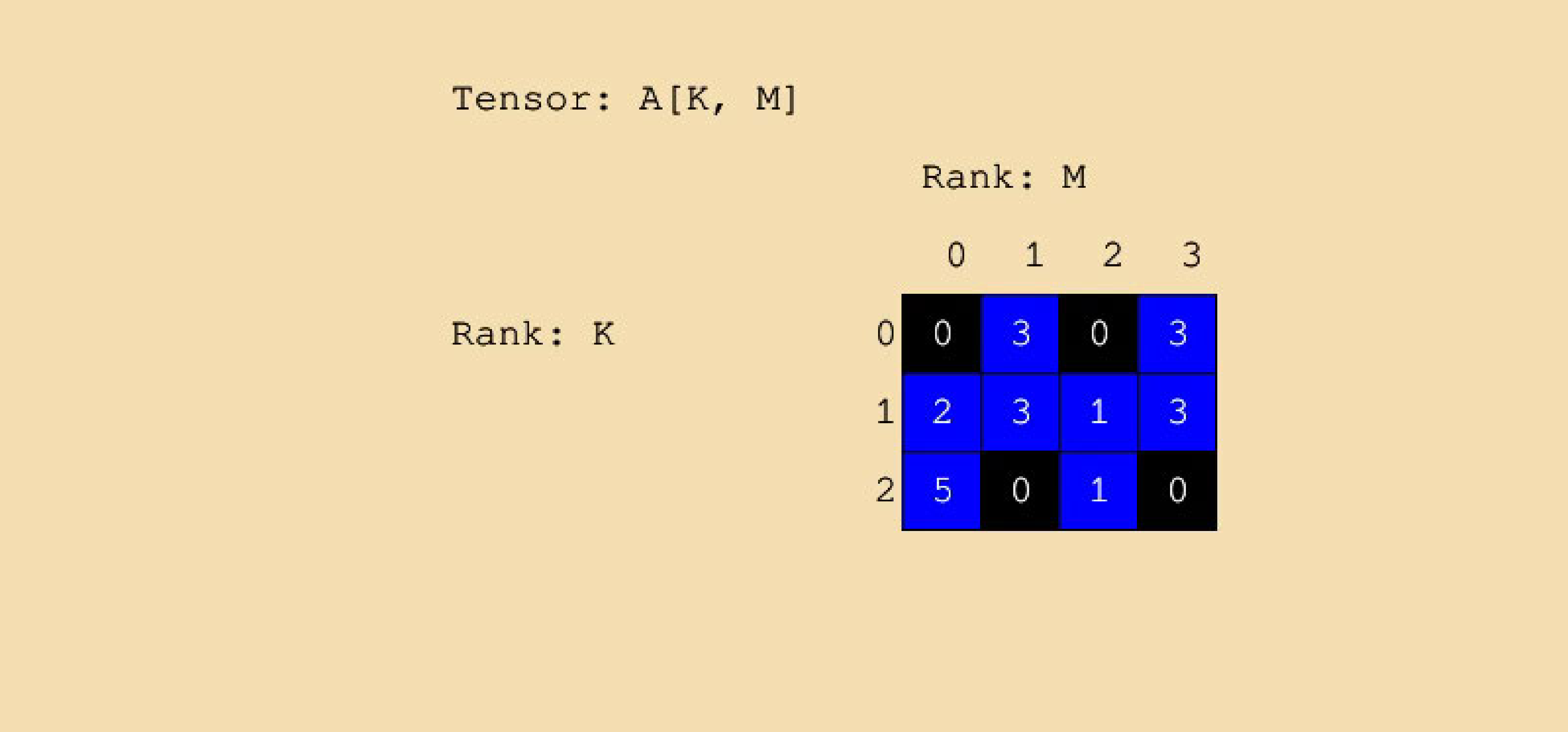
Rank: M

Rank: K

	0	1	2	3
0	0	3	0	3
1	2	3	1	3
2	5	0	1	0

Discordant Traversal - Uncompressed

Traversal order (slowest to fastest): K,M



```
Tensor: A[K, M]
```

```
Rank: M
```

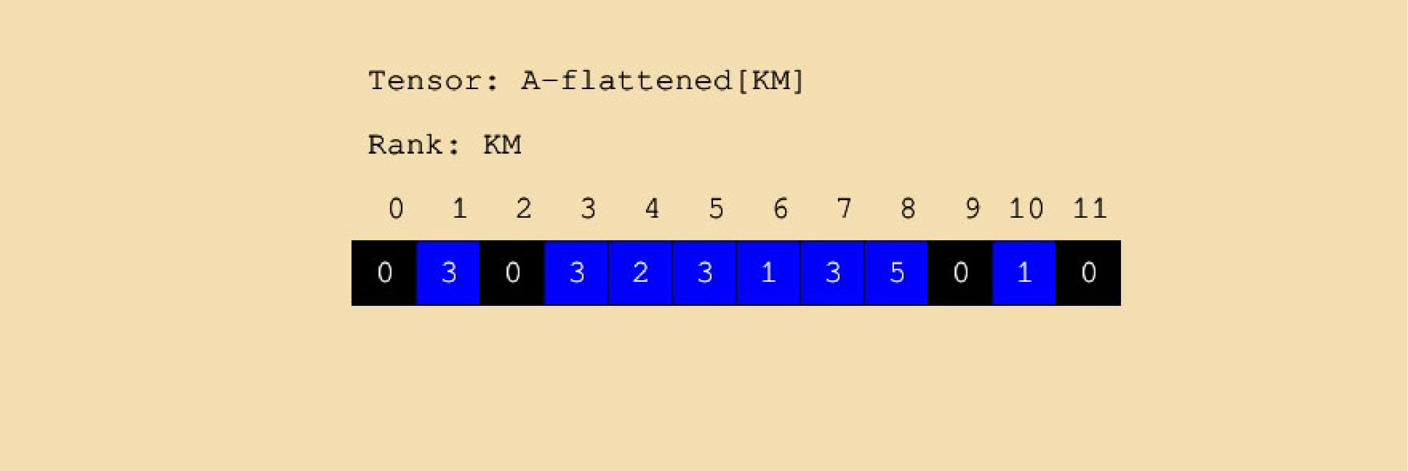
```
Rank: K
```

	0	1	2	3
0	0	3	0	3
1	2	3	1	3
2	5	0	1	0

Any difficulties with the pattern? Not good with block memory reads

Traversal - Flattened

Traversal order (slowest to fastest): K,M

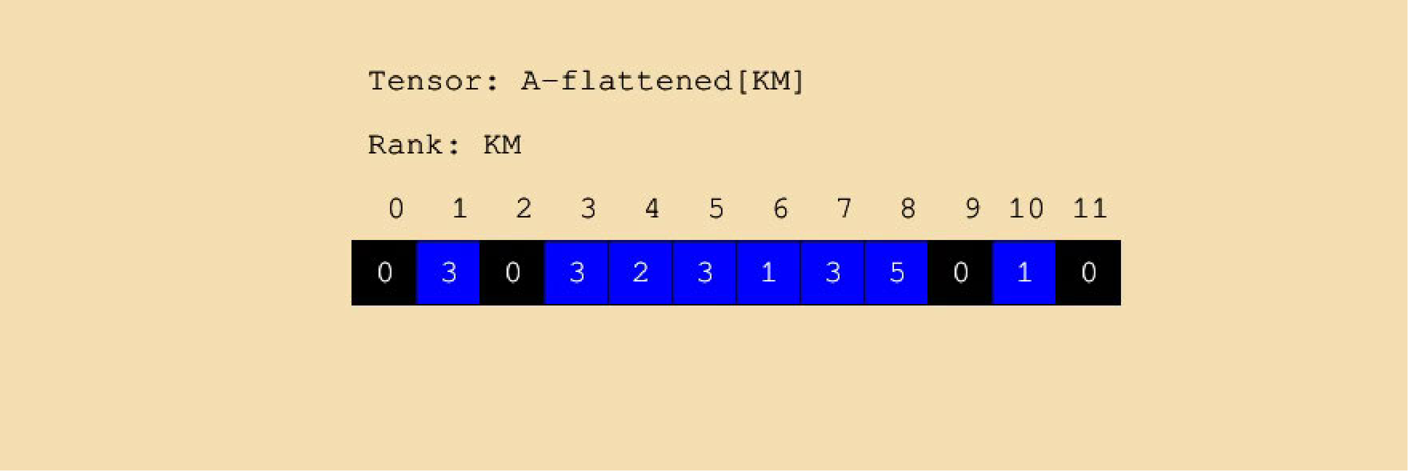


Tensor: A-flattened[KM]

Rank: KM

0	1	2	3	4	5	6	7	8	9	10	11
0	3	0	3	2	3	1	3	5	0	1	0

Traversal order (slowest to fastest): M,K



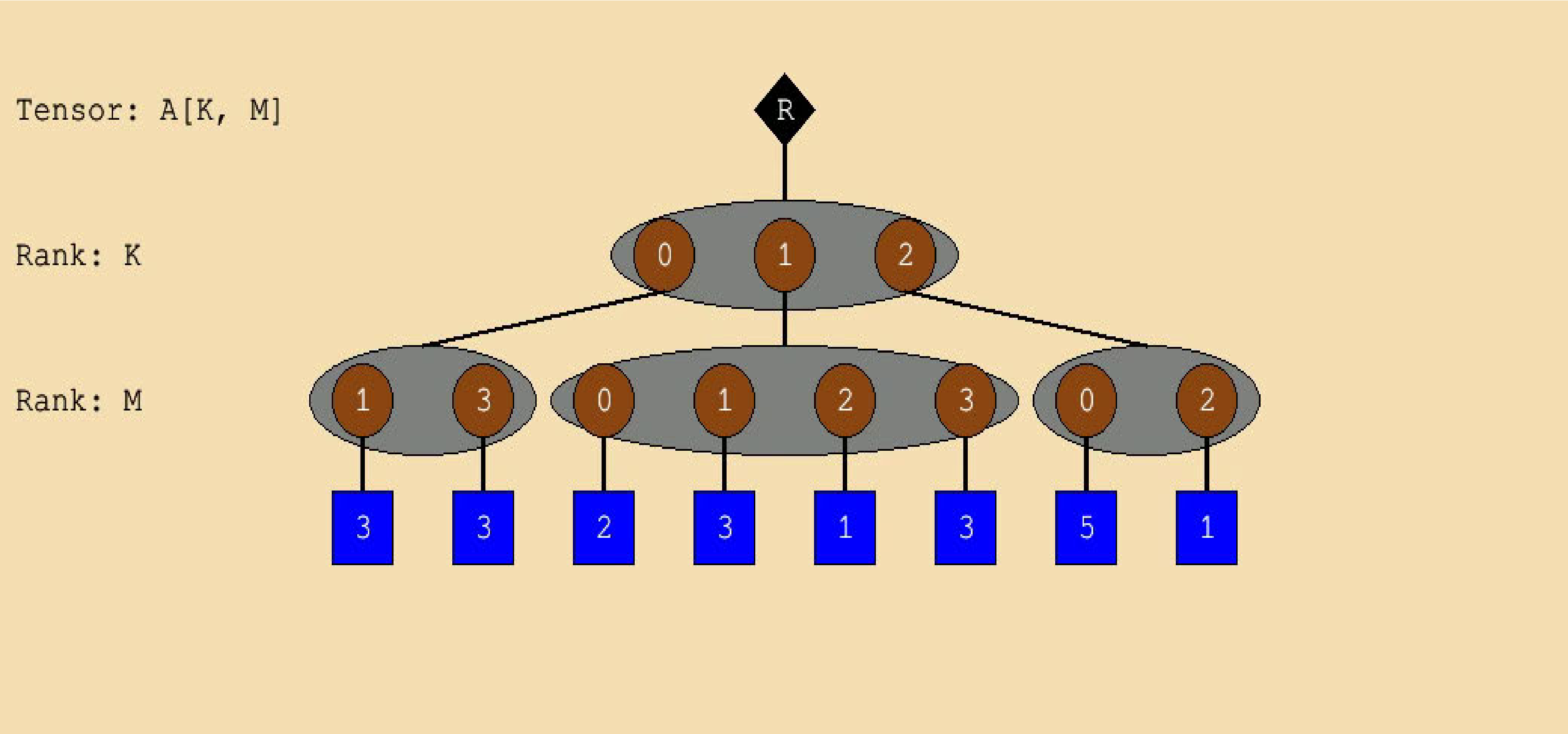
Tensor: A-flattened[KM]

Rank: KM

0	1	2	3	4	5	6	7	8	9	10	11
0	3	0	3	2	3	1	3	5	0	1	0

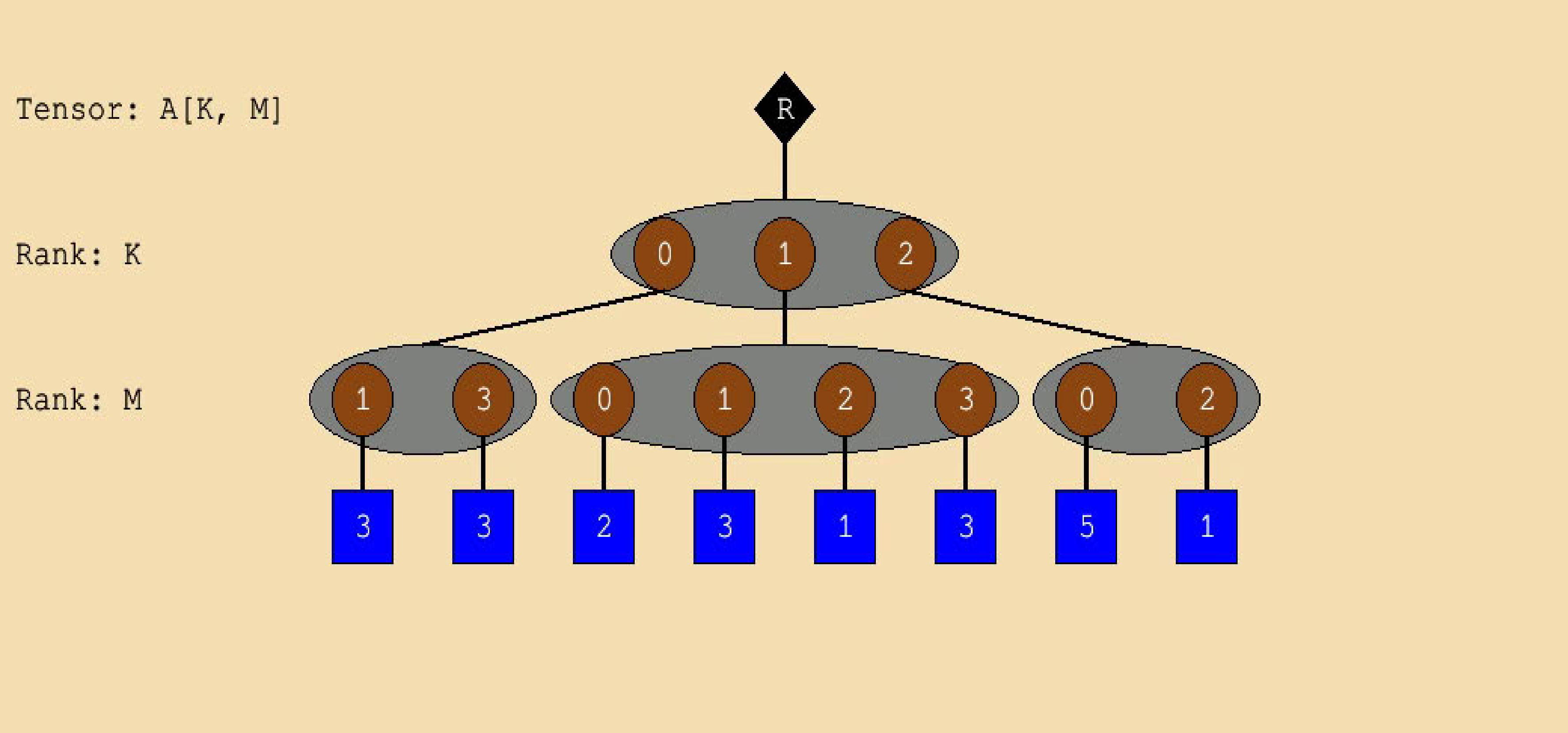
Concordant Traversal - Fibertree

Traversal order (slowest to fastest): K, M

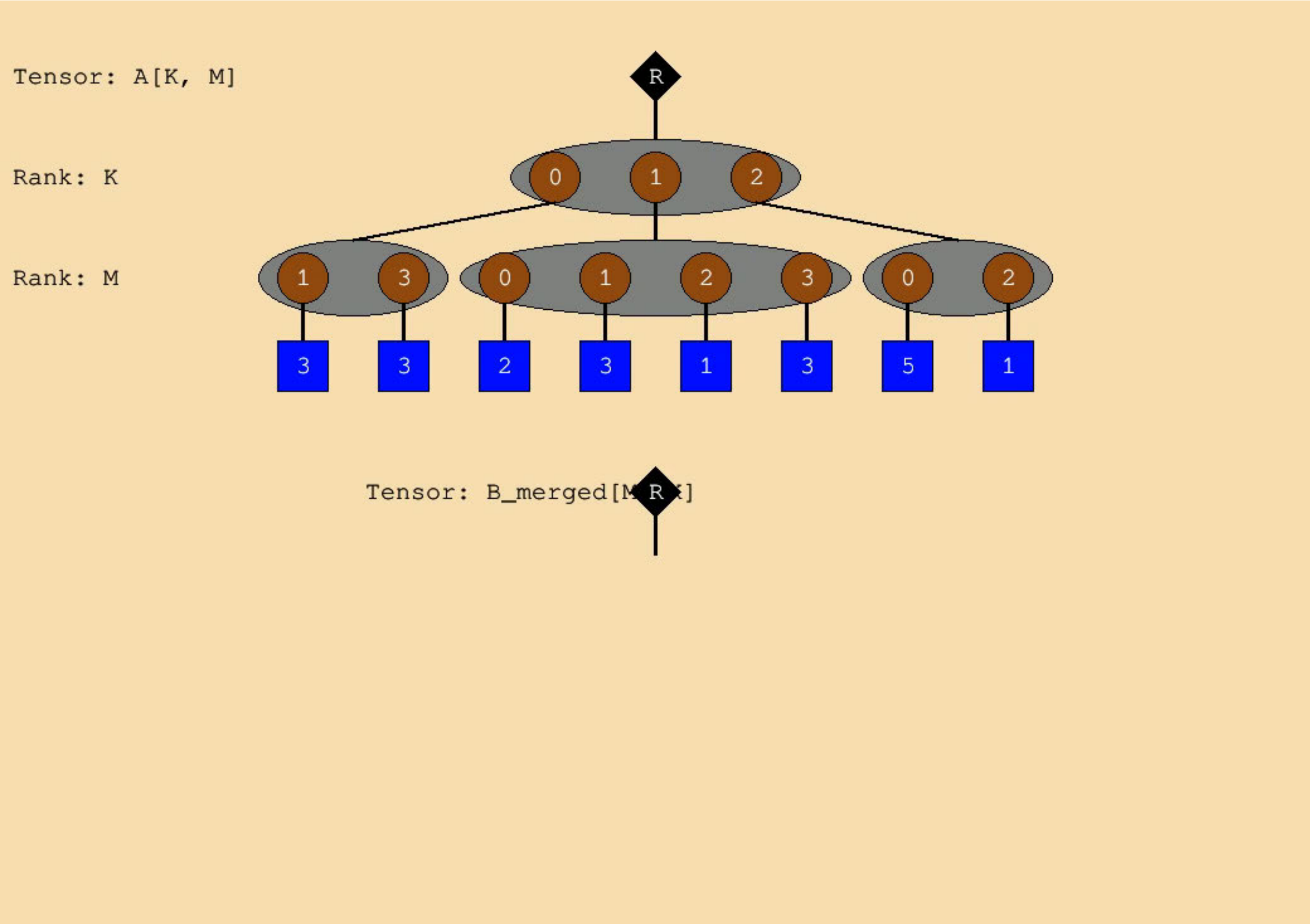


Discordant Traversal - Fibertree

Traversal order (slowest to fastest): M,K



Rank Swizzle/Merger



Take lowest untaken coordinate in input M-fibers and place into result at location with coordinates reversed

Gamma

Gamma Dataflow

$$Z_{m,n} = A_{m,k} \times B_{n,k}$$

$$\Downarrow$$

$$T_{m,k,n} = right(A_{m,k}, B_{k,n})$$

- Traversal (s to f): M, K, N
- Parallel K, M*

$$Z_{m,n} = T_{m,n,k} \times \ A_{k,m}$$

- Traversal (s to f): M, N, K
- Parallel M*

*Not modelled

Gamma Loopnest (no M parallelism)

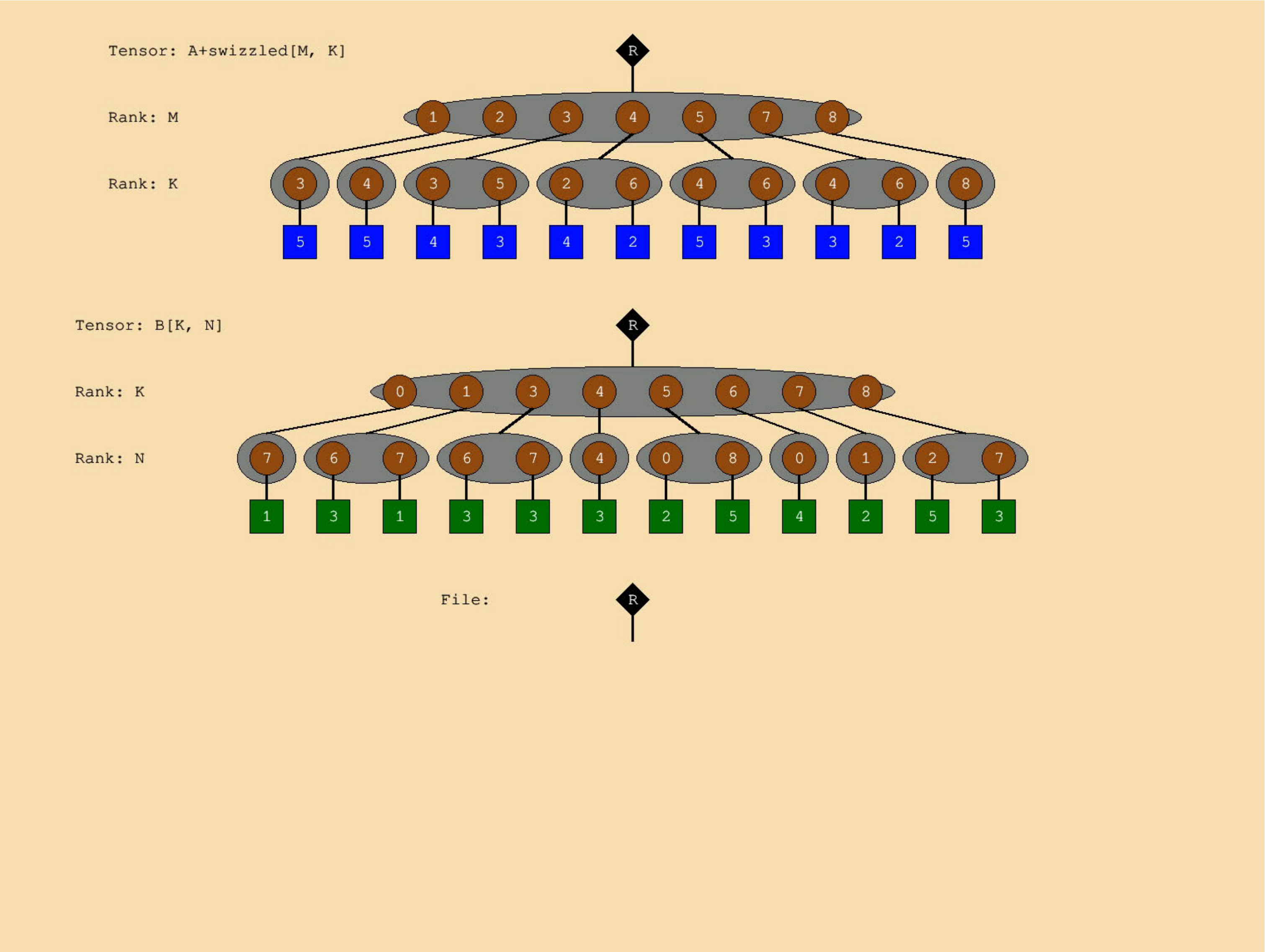
```
a = Tensor(M,K)       # Input A
b = Tensor(K,N)       # Input B
t = Tensor(M,K,N)     # Temporary
z = Tensor(M,N)       # Output

for m, (t_k, a_k) in t_m << a_m:
  p-for k, (t_n, (a_val, b_n)) in t_k << (a_k & b_k):
    for n, (t_ref, b_val) in t_n << b_n:
      t_ref <<= b_val

# swizzle ranks of t here…

for m, (z_n, (t_n, a_k)) in z_m << (t_m & a_m):
  for n, (z_ref, t_k) in z_n << t_n:
    for k, (t_val, a_val) in t_k & a_k:
      z_ref += t_val * a_val
```

Gamma - Step 1

$$T_{m,k,n} = right(A_{m,k}, B_{k,n})$$

Traversal (s to f): M, K, N

Parallel K, M*



*Not modelled

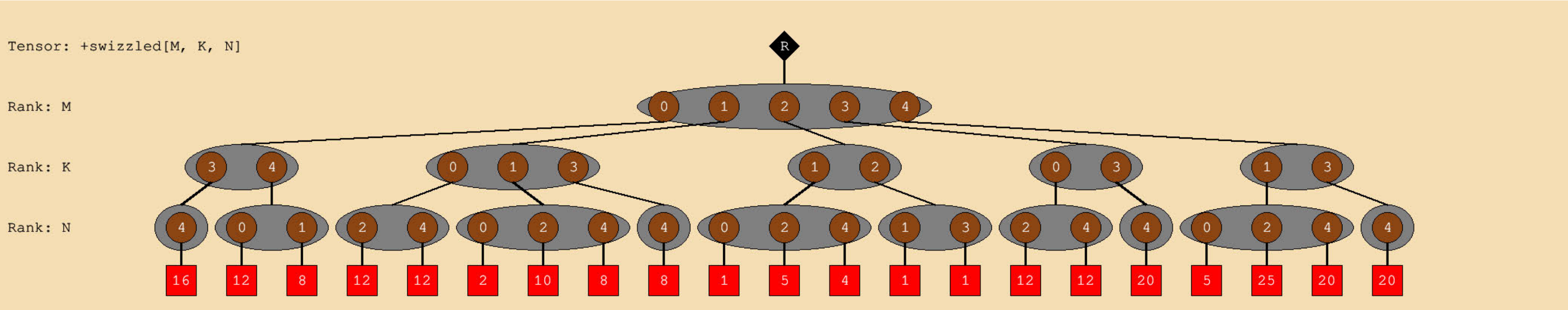
Gamma – Step 1 - Observations

- There is a single concordant traversal of A
- The same B_n fibers are fetched multiple times.

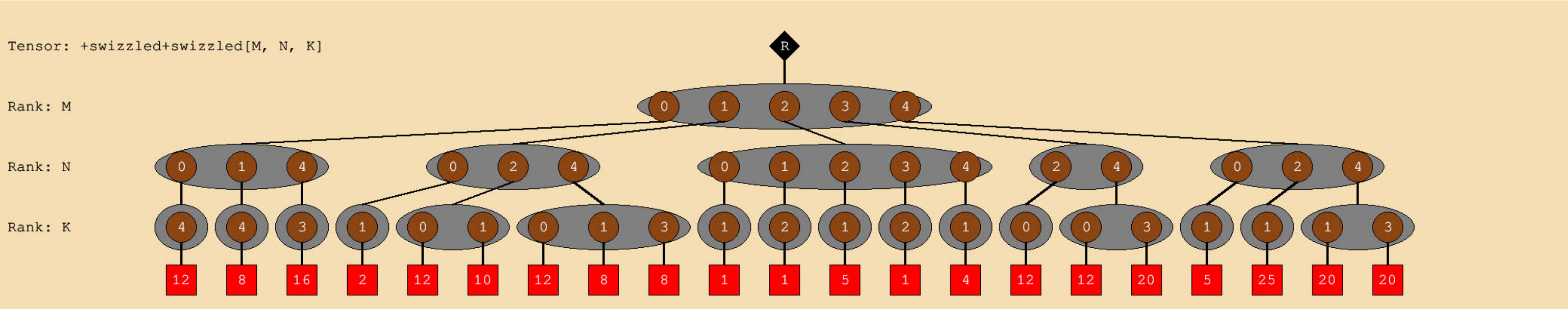
- For each specific M, the processing is parallel across K
  - And the T_n fibers below are created concordently
  - Thus, creating T in a manner that allows for it to be rank swizzled

Gamma - Rank Swizzled T

T[M,K,N]



T’[M,N,K]



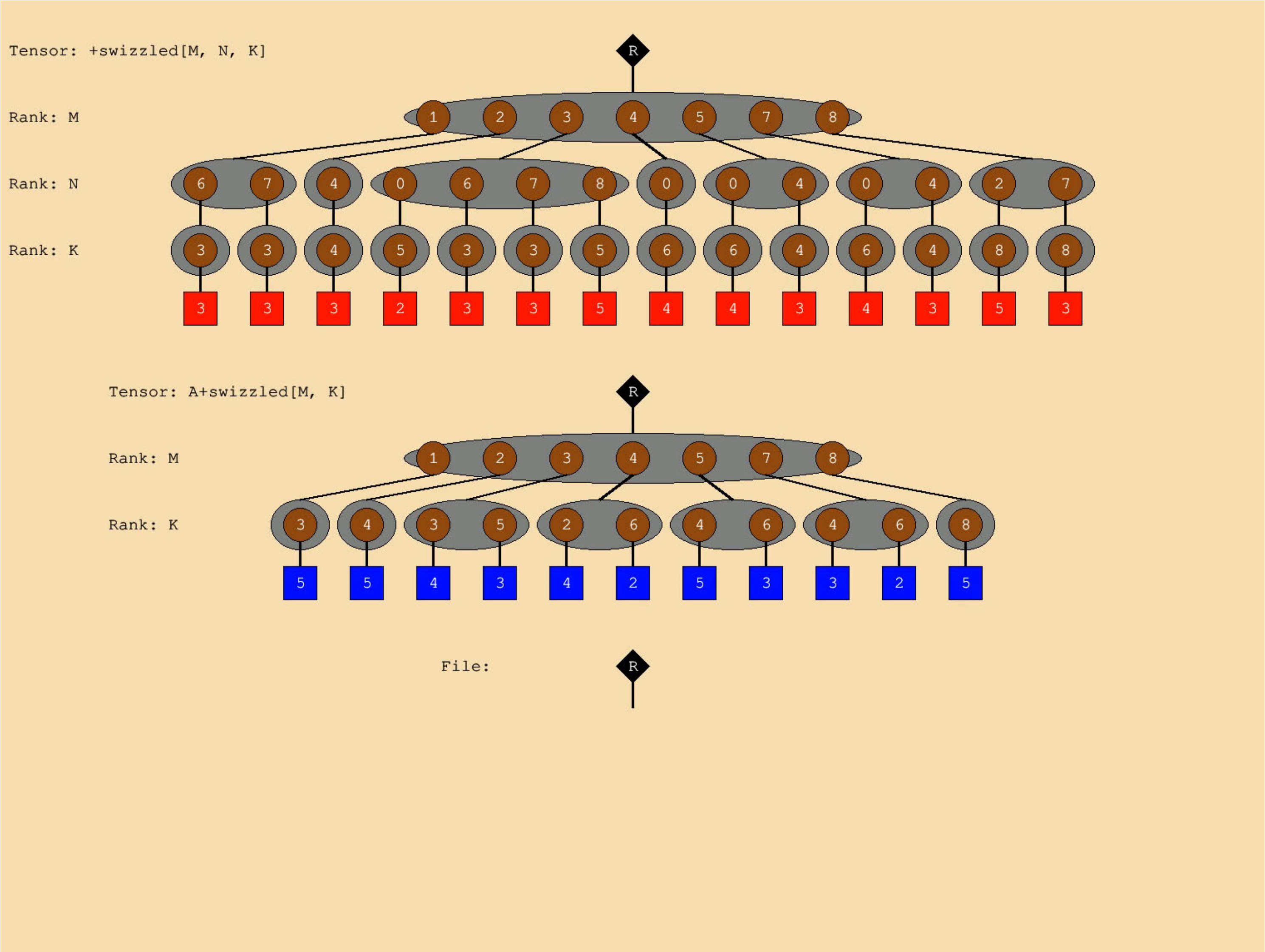
Since elements of each K fiber in T[M,K,N] are processed in parallel and elements in N fibers are created concordantly, the head elements needed for the swizzle are available!

Gamma – Step 2

$$Z_{m,n} = T_{m,n,k} \times \ A_{k,m}$$

Traversal (s to f): M, N, K

Parallel M*

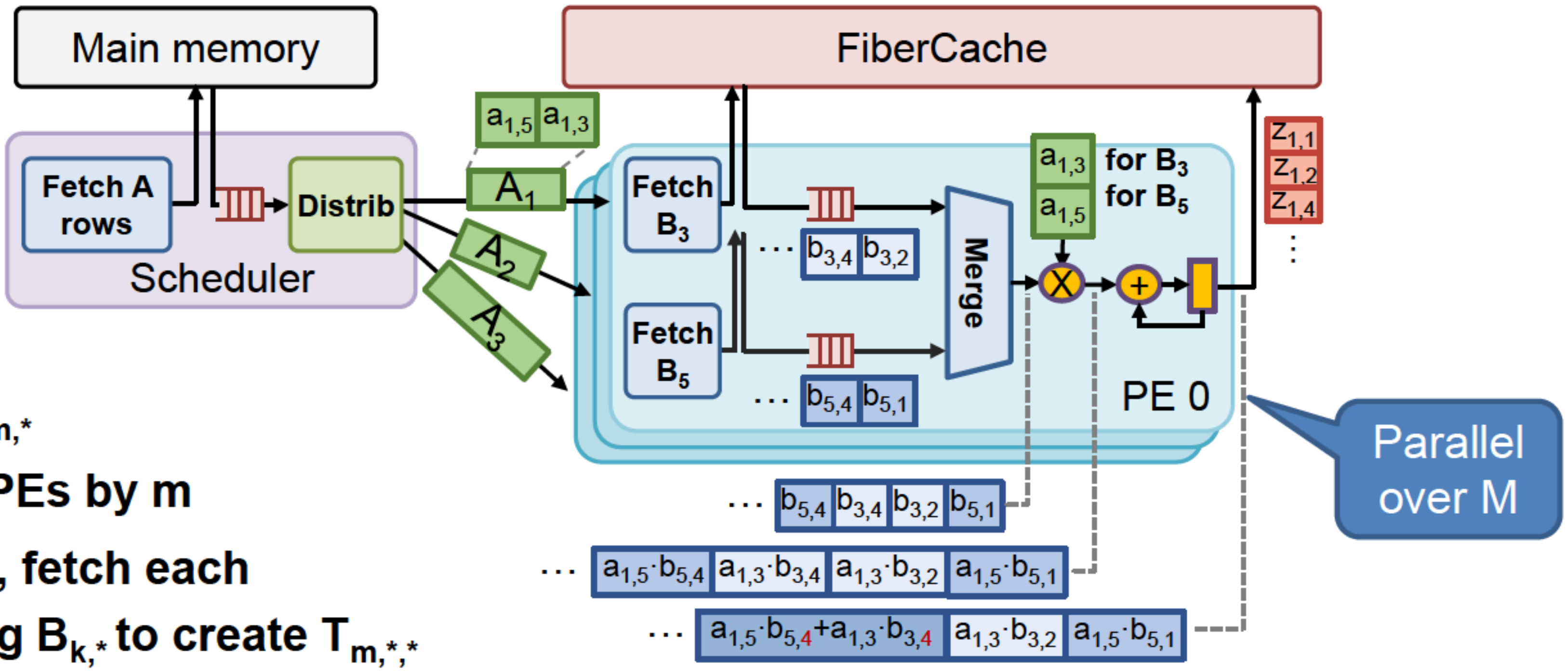


*Not modelled

Gamma – Step 2 - Observations

- Exactly one concordant traversal of (swizzled) T tensor
- Concordant traversal of (swizzled) T that means it can be created in pipeline and consumed immediately without being held in its entirety in a buffer.
- Note that A_k fibers are re-read repeatedly but are small since they are post-intersection.
- Output Z is created concordantly

Gamma – Block Diagram

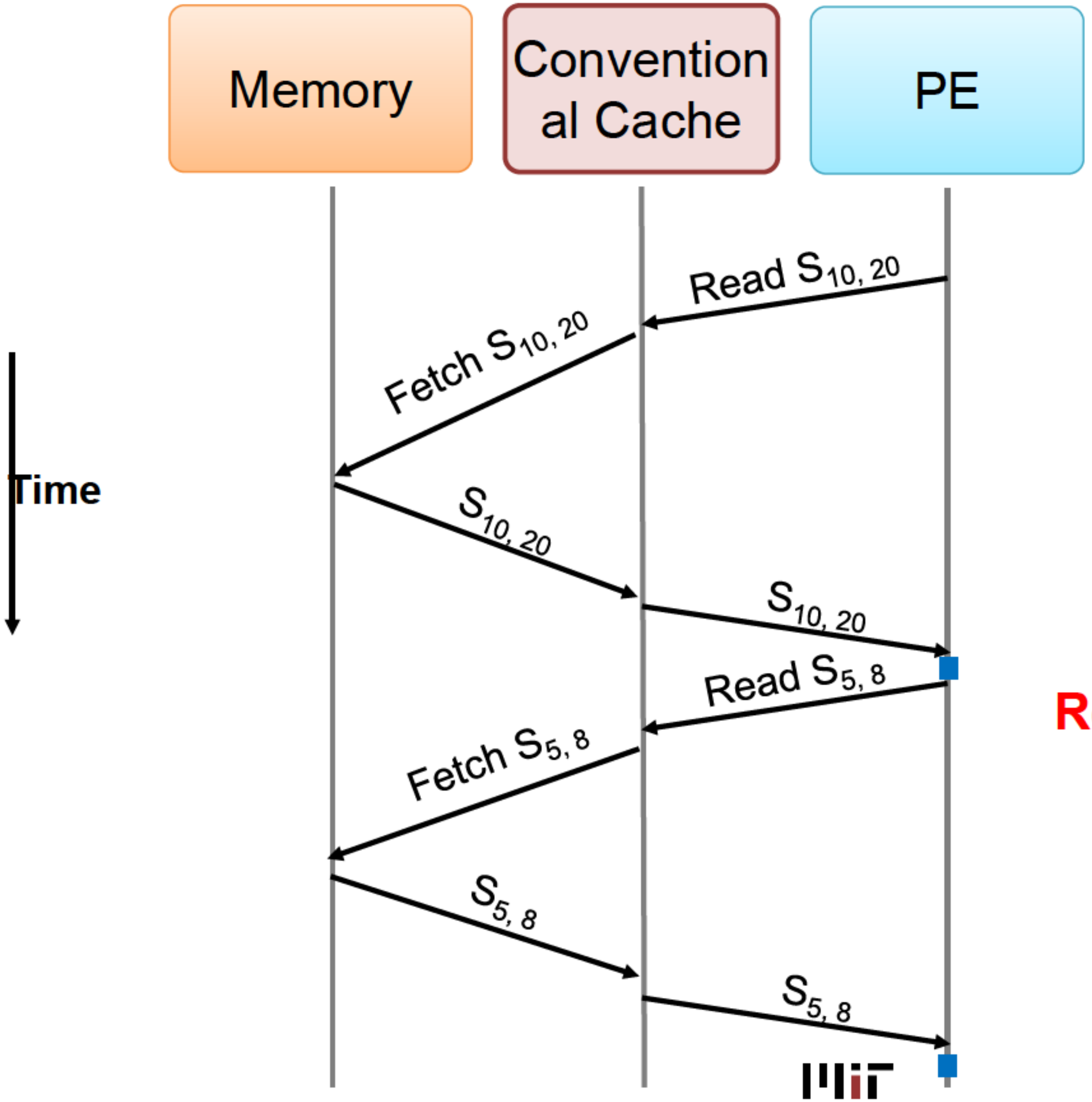


- **Fetch each $A_{m,*}$**
- **Distribute to PEs by m**
- **Across all k's, fetch each corresponding $B_{k,*}$ to create $T_{m,*,*}$**
- **Send to merger**
- **Merge to swizzle ranks**
- **Fetch $A_{m,k}$ for $B_{k,n}$ and multiply**
- **Accumulate products for $A_{m,n}$**
- **Save result**

Key: $A_{m,k}$ $B_{k,n}$ $Z_{m,n}$

[Gamma, Zhang, et.al., ASPLOS 2021]

Conventional caches suffer from dependent reads

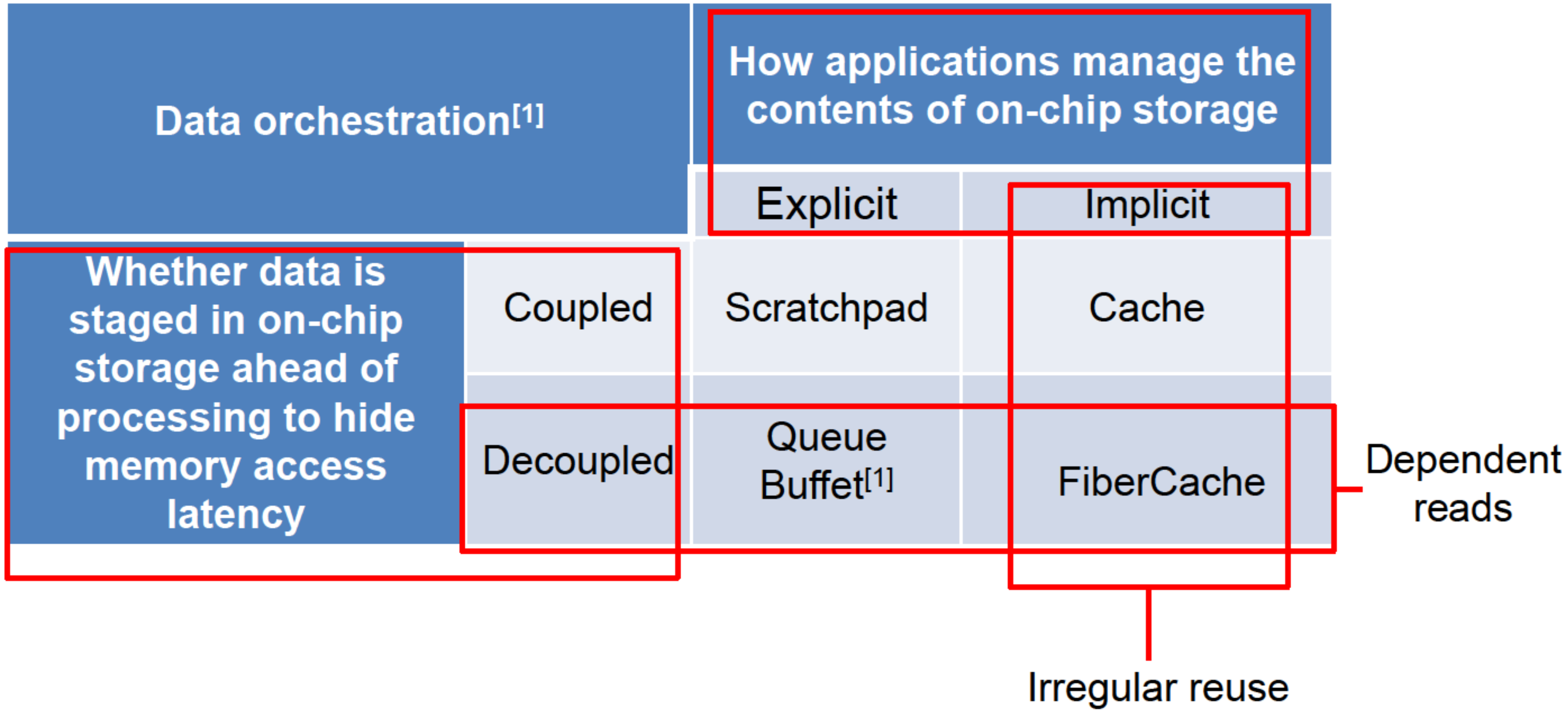


Long roundtrip latency

Too many fibers

Requires huge buffers
to hide latency

Decoupled implicit data orchestration



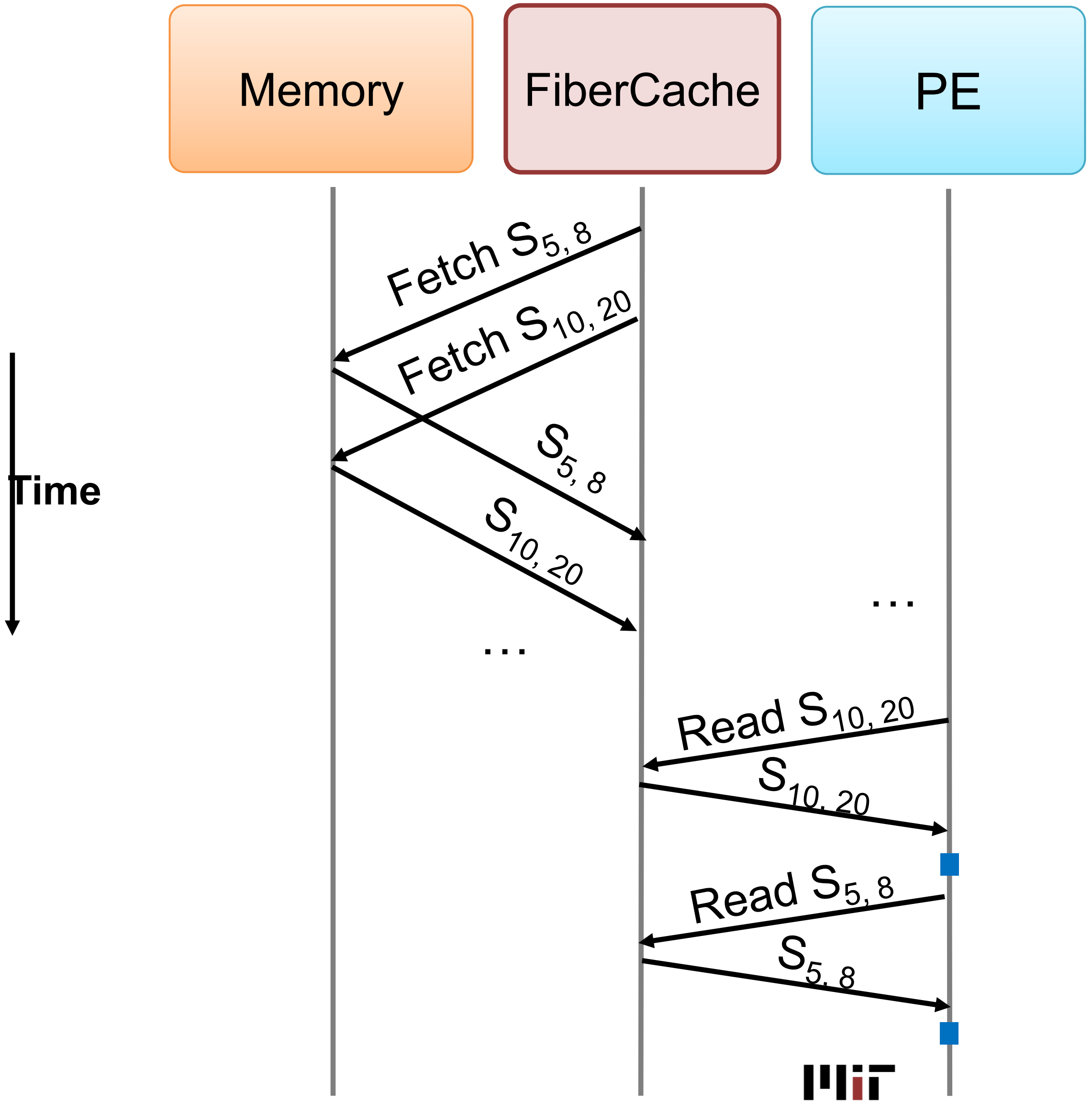
Data orchestration[1]		How applications manage the contents of on-chip storage	
		Explicit	Implicit
Whether data is staged in on-chip storage ahead of processing to hide memory access latency	Coupled	Scratchpad	Cache
	Decoupled	Queue Buffet[1]	FiberCache

Dependent reads

Irregular reuse

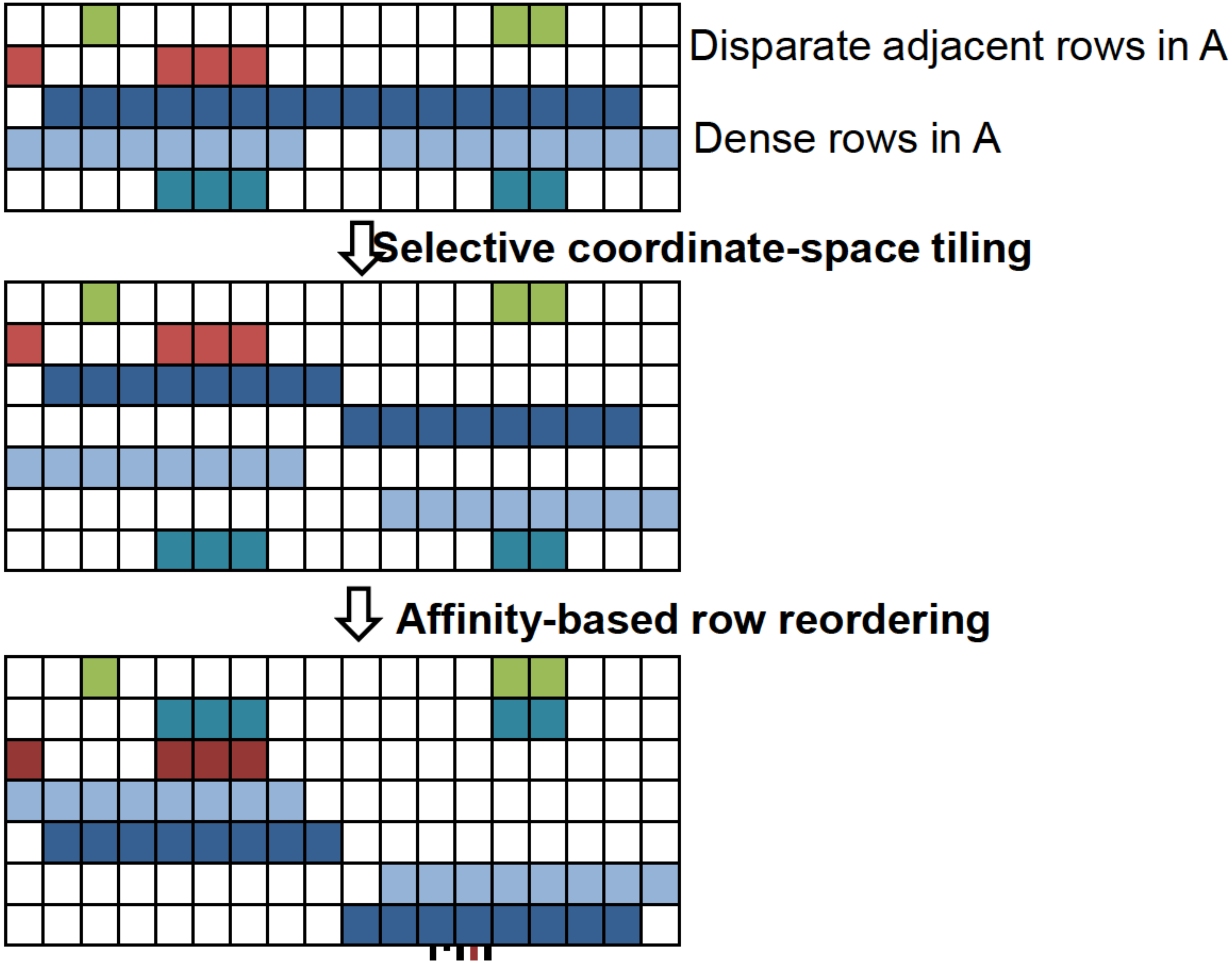
[1] Pellauer et al., ASPLOS 2019

FiberCache decouples read roundtrips and memory latencies



Low roundtrip latency easy to cover with small buffers

Preprocessing matrix A for GAMMA



Disparate adjacent rows in A

Dense rows in A

Selective coordinate-space tiling

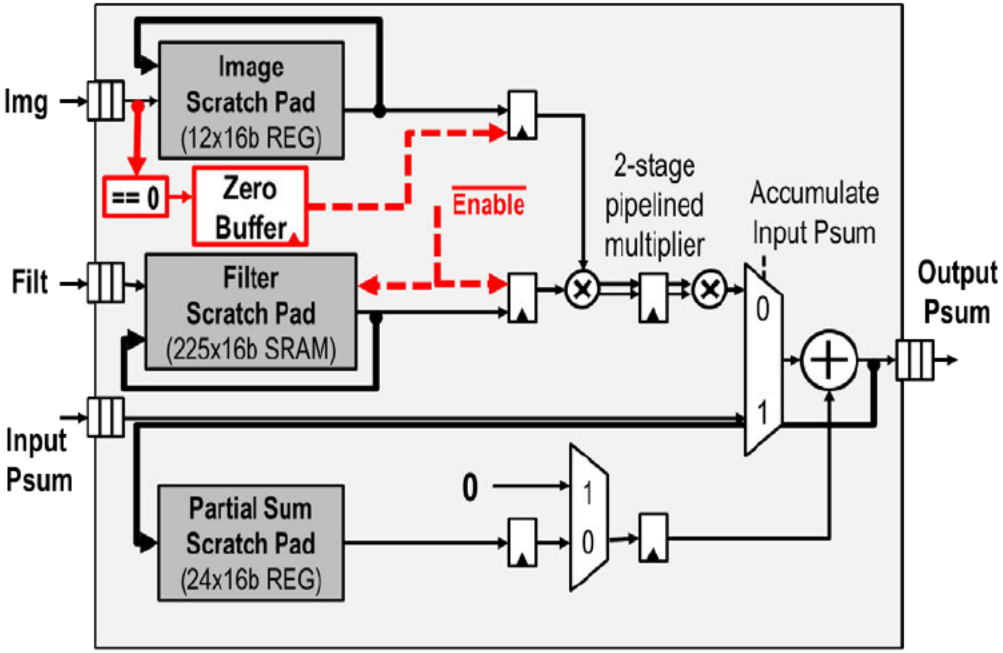
Affinity-based row reordering

Gamma Concepts

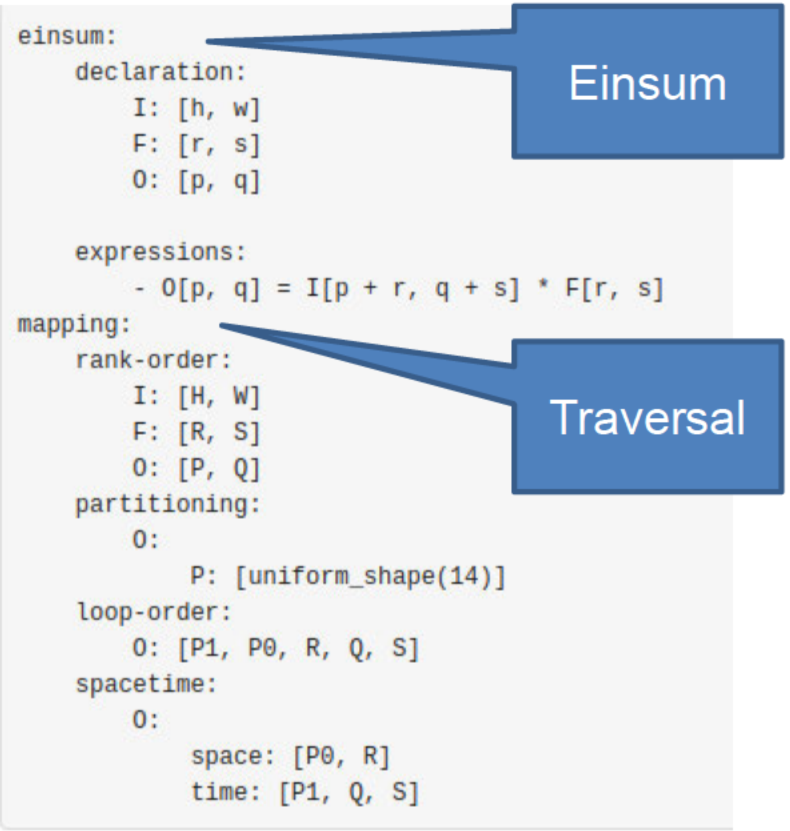
- Pipeline computations with small intermediate storage
- Use parallelism/merger to do pipelined rank swizzle
- Decoupled/implicit fibercache to hold B fibers that might be reused
- Reorder A to maximize effectiveness of fibercache

TeAAL – Modeling Sparse Dataflows

TeAAL - Eyeriss Design

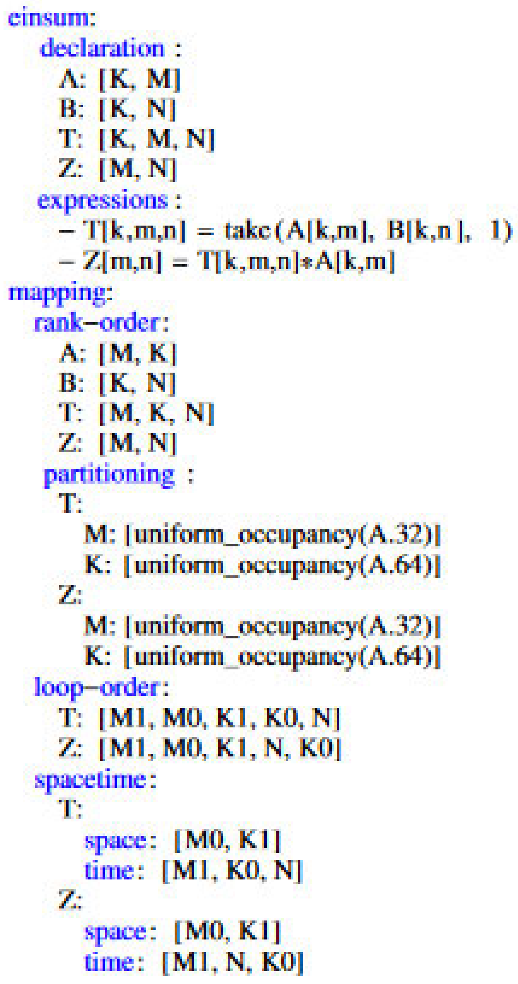

Img
Image Scratch Pad (12x16b REG)
== 0
Zero Buffer
Enable
Filt
Filter Scratch Pad (225x16b SRAM)
Input Psum
Partial Sum Scratch Pad (24x16b REG)
0
1
0
2-stage pipelined multiplier
Accumulate Input Psum
0
1
Output Psum

Eyeriss [JSSC2017]



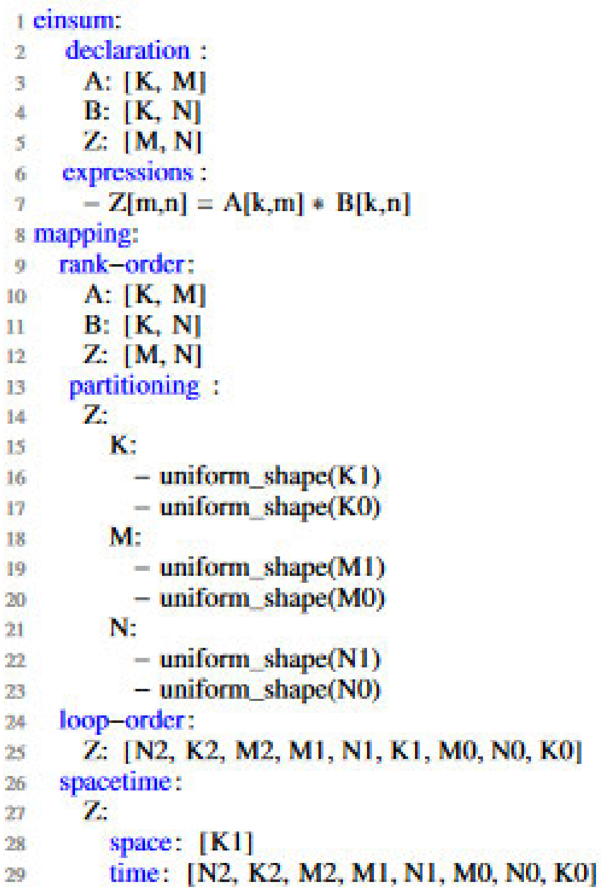
```
einsum:
    declaration:
        I: [h, w]
        F: [r, s]
        O: [p, q]

    expressions:
        - O[p, q] = I[p + r, q + s] * F[r, s]
mapping:
    rank-order:
        I: [H, W]
        F: [R, S]
        O: [P, Q]
    partitioning:
        O:
            P: [uniform_shape(14)]
    loop-order:
        O: [P1, P0, R, Q, S]
    spacetime:
        O:
            space: [P0, R]
            time: [P1, Q, S]
```

TeAAL - Matrix Multiplication Designs



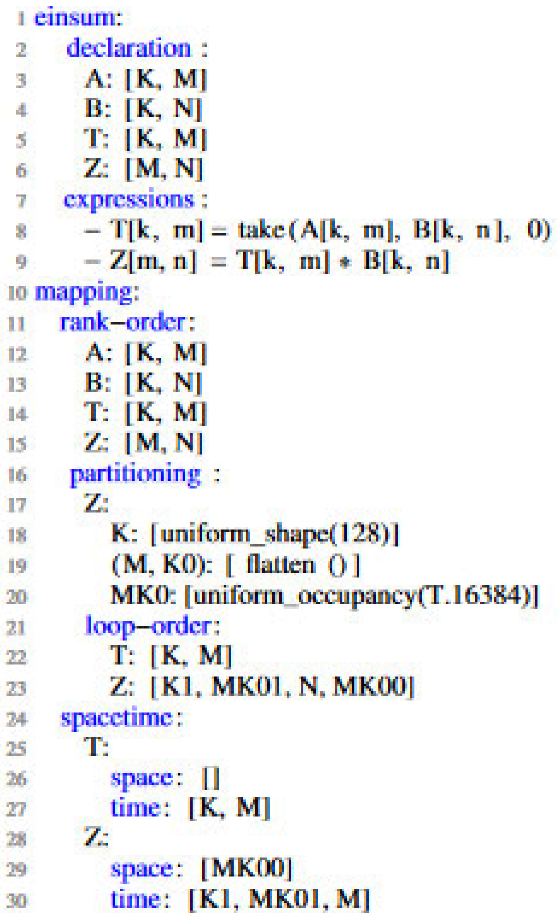
```
einsum:
  declaration :
    A: [K, M]
    B: [K, N]
    T: [K, M, N]
    Z: [M, N]
  expressions :
    − T[k,m,n] = take(A[k,m], B[k,n], 1)
    − Z[m,n] = T[k,m,n]∗A[k,m]
mapping:
  rank−order:
    A: [M, K]
    B: [K, N]
    T: [M, K, N]
    Z: [M, N]
  partitioning :
    T:
      M: [uniform_occupancy(A.32)]
      K: [uniform_occupancy(A.64)]
    Z:
      M: [uniform_occupancy(A.32)]
      K: [uniform_occupancy(A.64)]
  loop−order:
    T: [M1, M0, K1, K0, N]
    Z: [M1, M0, K1, N, K0]
  spacetime:
    T:
      space: [M0, K1]
      time: [M1, K0, N]
    Z:
      space: [M0, K1]
      time: [M1, N, K0]
```

(a) Gamma accelerator [50].



```
einsum:
  declaration :
    A: [K, M]
    B: [K, N]
    Z: [M, N]
  expressions :
    − Z[m,n] = A[k,m] ∗ B[k,n]
mapping:
  rank−order:
    A: [K, M]
    B: [K, N]
    Z: [M, N]
  partitioning :
    Z:
      K:
        − uniform_shape(K1)
        − uniform_shape(K0)
      M:
        − uniform_shape(M1)
        − uniform_shape(M0)
      N:
        − uniform_shape(N1)
        − uniform_shape(N0)
  loop−order:
    Z: [N2, K2, M2, M1, N1, K1, M0, N0, K0]
  spacetime:
    Z:
      space: [K1]
      time: [N2, K2, M2, M1, N1, M0, N0, K0]
```

(b) ExTensor accelerator [14].

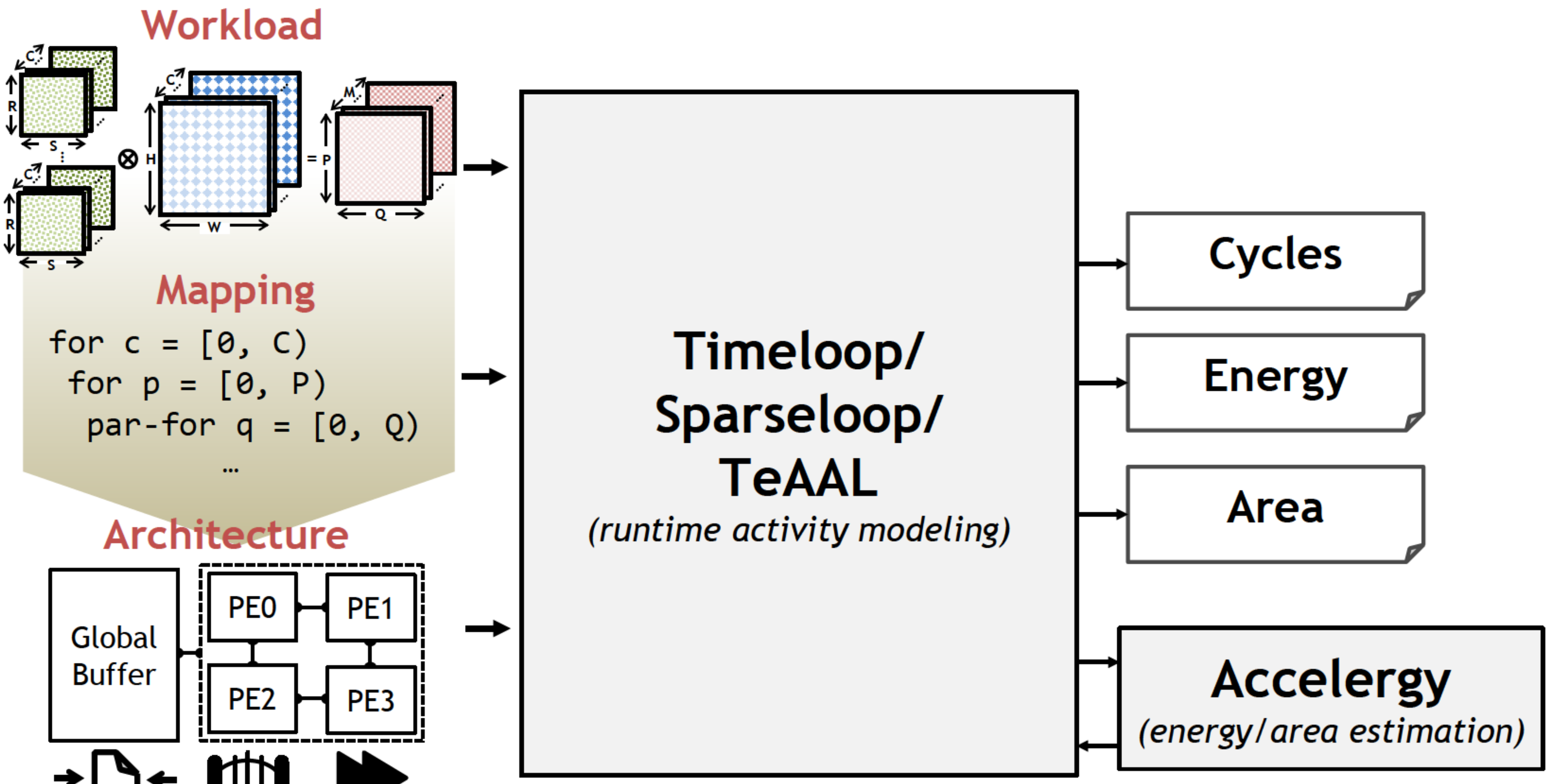


```
einsum:
  declaration :
    A: [K, M]
    B: [K, N]
    T: [K, M]
    Z: [M, N]
  expressions :
    − T[k, m] = take(A[k, m], B[k, n], 0)
    − Z[m, n] = T[k, m] ∗ B[k, n]
mapping:
  rank−order:
    A: [K, M]
    B: [K, N]
    T: [K, M]
    Z: [M, N]
  partitioning :
    Z:
      K: [uniform_shape(128)]
      (M, K0): [ flatten () ]
      MK0: [uniform_occupancy(T.16384)]
  loop−order:
    T: [K, M]
    Z: [K1, MK01, N, MK00]
  spacetime:
    T:
      space: []
      time: [K, M]
    Z:
      space: [MK00]
      time: [K1, MK01, M]
```

(c) SIGMA accelerator [34].

[Nayak, TeAAL, MICRO2023]

Modeling Infrastructure



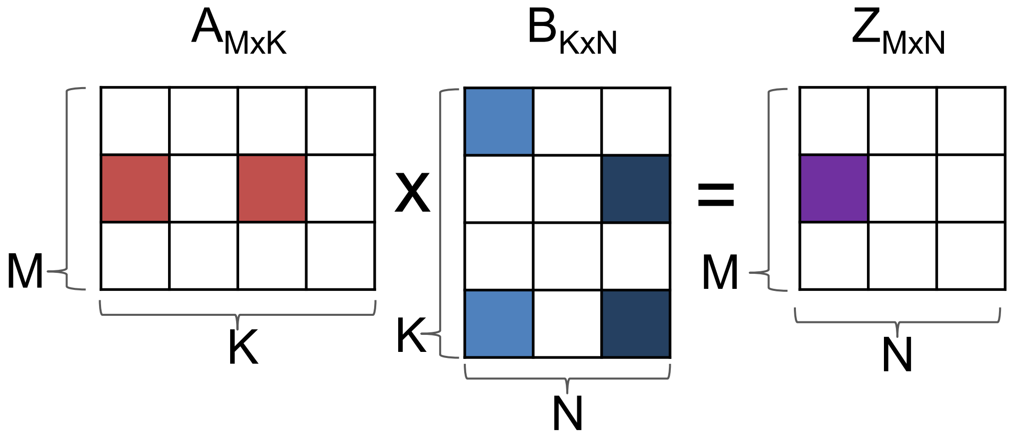
```
for c = [0, C)
 for p = [0, P)
  par-for q = [0, Q)
   …
```

timeloop.csail.mit.edu

accelergy.mit.edu

Summary

spMspM dataflows



$$Z_{m,n} = \sum_k A_{m,k} B_{k,n}$$

Output Stationary
Inner-product

```
for m in [0, M)
 for n in [0, N)
  for k in [0, K)
   Z[m,n] += A[m,k] * B[k,n]
```

A-stationary – Column major
Outer-product

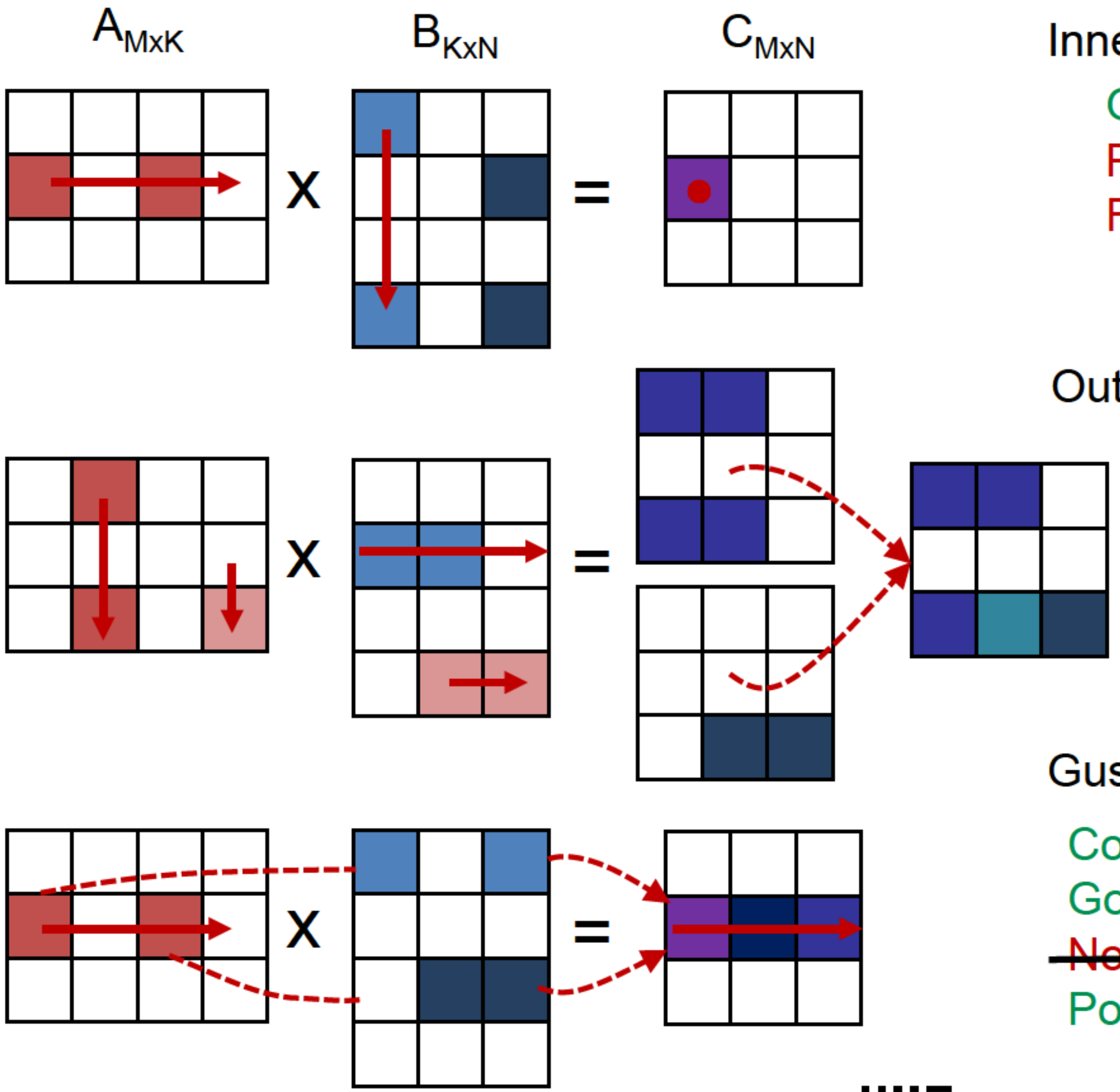
```
for k in [0, K)
 for m in [0, M)
  for n in [0, N)
   Z[m,n] += A[m,k] * B[k,n]
```

A-stationary – Row major
Gustavson

```
for m in [0, M)
 for k in [0, K)
  for n in [0, N)
   Z[m,n] += A[m,k] * B[k,n]
```

$$Z_m = \sum_k a_{m,k} B_k$$

spMspM dataflows



Inner-product dataflow

Good output reuse
Poor input reuse
Frequent Ineffectual intersections

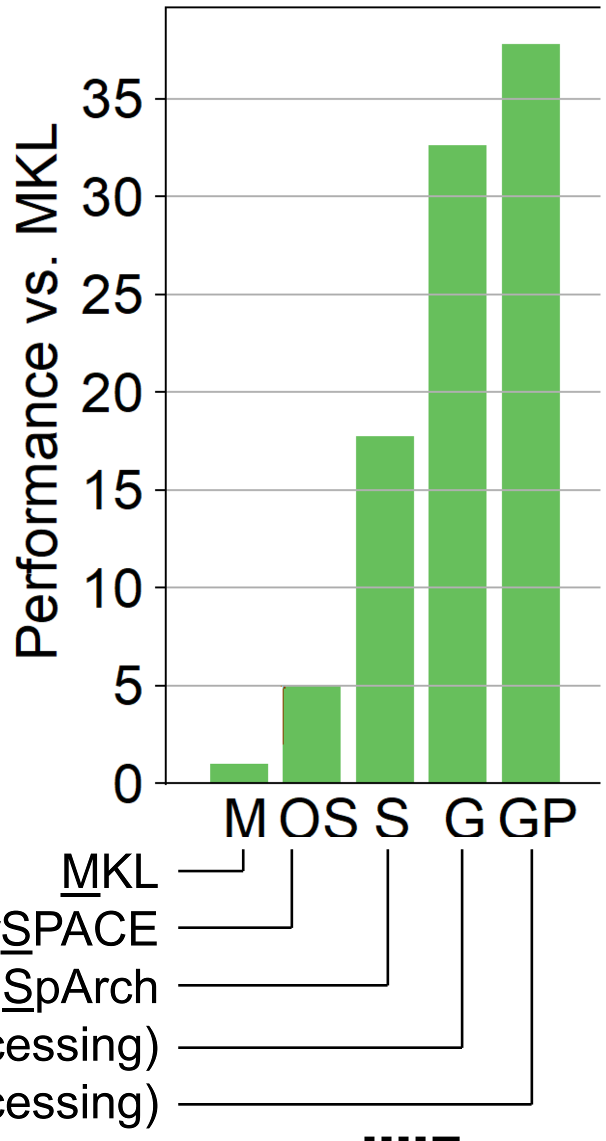
Outer-product dataflow

Good input reuse
Poor output reuse
Expensive matrix combinations

Gustavson dataflow

Consistent formats of in-/output
Good reuse on A and C
~~Nominally poor reuse of B~~
Potentially good reuse of B

Speedups over Intel MKL on common-set matrices



EIE

EIE

```
i = Tensor(CHW)         # Input activations
f = Tensor(CHW, M)      # Filter weights
o = Array(M)            # Output activations

for chw, i_val in i:
  parallel-for f_split in f.splitUniform(M/#PEs):
    f_m = f_split.getPayload(chw)
    for (m, f_val) in f_m:
      o[m] += i_val * f_val
```

Get an input

Split filter weights uniformly in coordinate space among the the PEs. This is done statically before the run.

Broadcast chw (and i_val) to all PEs so we can get the weights from each PE’s split of output channels for the current input.

Local accumulation

Traverse the weights for the output channels

EIE: A Sparse Linear Algebra Engine

- Process Fully Connected Layers (after Deep Compression)
- Store weights column-wise in Run Length format (i.e., CSC format)
- Read relative column when input is non-zero

Supports Fully Connected Layers Only

Input

$$\begin{pmatrix} 0 & \boxed{\boldsymbol{a_1}} & 0 & a_3 \end{pmatrix}$$

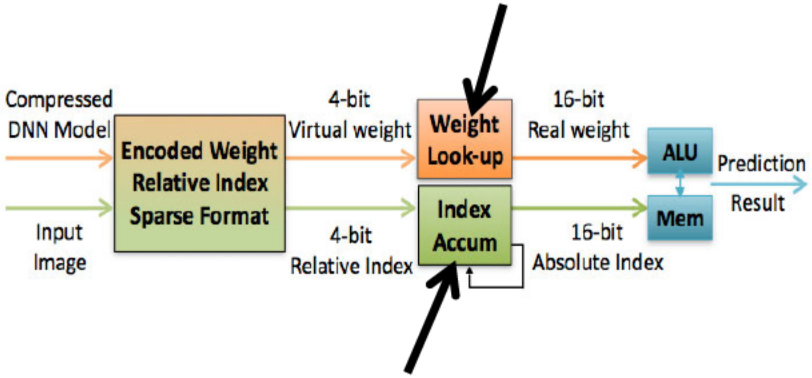
$$\times$$

Weights

Output

$$\begin{pmatrix} w_{0,0} & \boldsymbol{w_{0,1}} & 0 & w_{0,3} \\ 0 & \boldsymbol{0} & w_{1,2} & 0 \\ 0 & \boldsymbol{w_{2,1}} & 0 & w_{2,3} \\ 0 & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & w_{4,2} & w_{4,3} \\ w_{5,0} & \boldsymbol{0} & 0 & 0 \\ 0 & \boldsymbol{0} & 0 & w_{6,3} \\ 0 & \boldsymbol{w_{7,1}} & 0 & 0 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ -b_2 \\ b_3 \\ -b_4 \\ b_5 \\ b_6 \\ -b_7 \end{pmatrix} \overset{ReLU}{\Rightarrow} \begin{pmatrix} b_0 \\ b_1 \\ 0 \\ b_3 \\ 0 \\ b_5 \\ b_6 \\ 0 \end{pmatrix}$$

Dequantize Weight

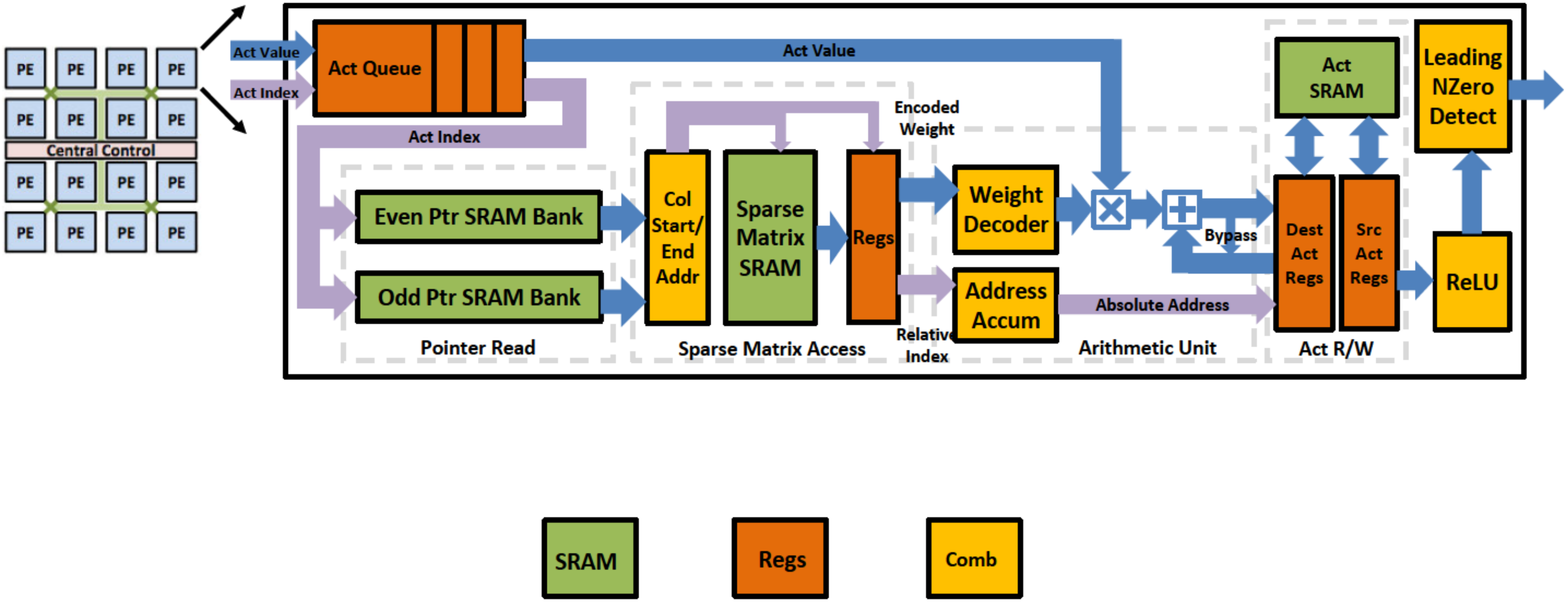


Keep track of location

Output Stationary Dataflow

[EIE, Han et al., ISCA 2016]

PE Architecture



[EIE, Han et al., ISCA 2016]

Summary

- Design attributes of spMspM accelerators:
  - Data can be tiled to improve locality
  - Sparse data makes intersection an explicit operation
  - Intersection can be hierarchical – intersecting at higher levels of the fibertree
  - There are three major dataflows for spMspM
  - spMspM can be broken into multiple pipelined stages
  - Rank swapping can be required to achieve concordant traversals
  - Rank swapping can be implemented with a “merge” unit
  - Data movement can be optimized via data format selection
  - Data movement can be reduced with explicit-decoupled caching
- Most of the above can be expressed as a scheduled Einsum
- A loop nest implementation can be inferred from a scheduled Einsum
- Lots of interesting variations in spMspM acceleration!

Thank You