Sparse Architectures

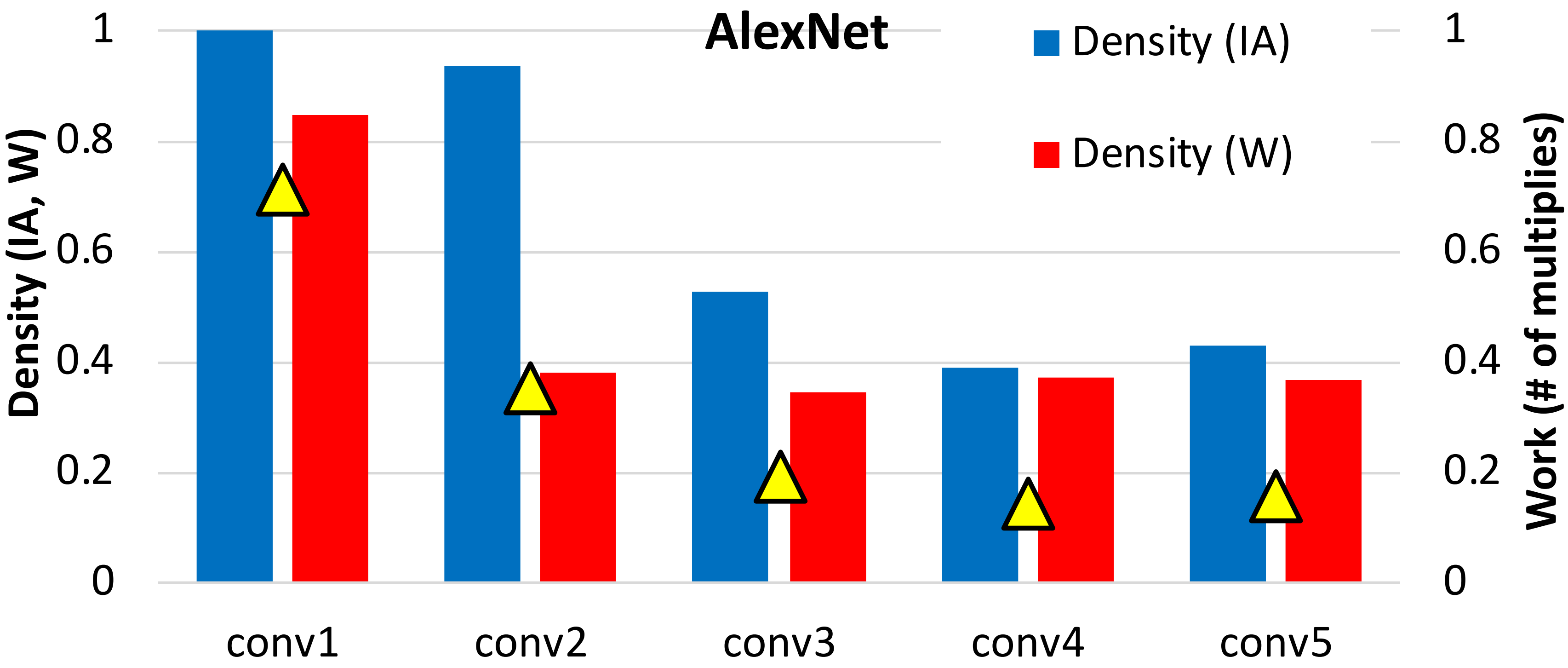
ISCA Tutorial (2019)

Website: http://eyeriss.mit.edu/tutorial.html

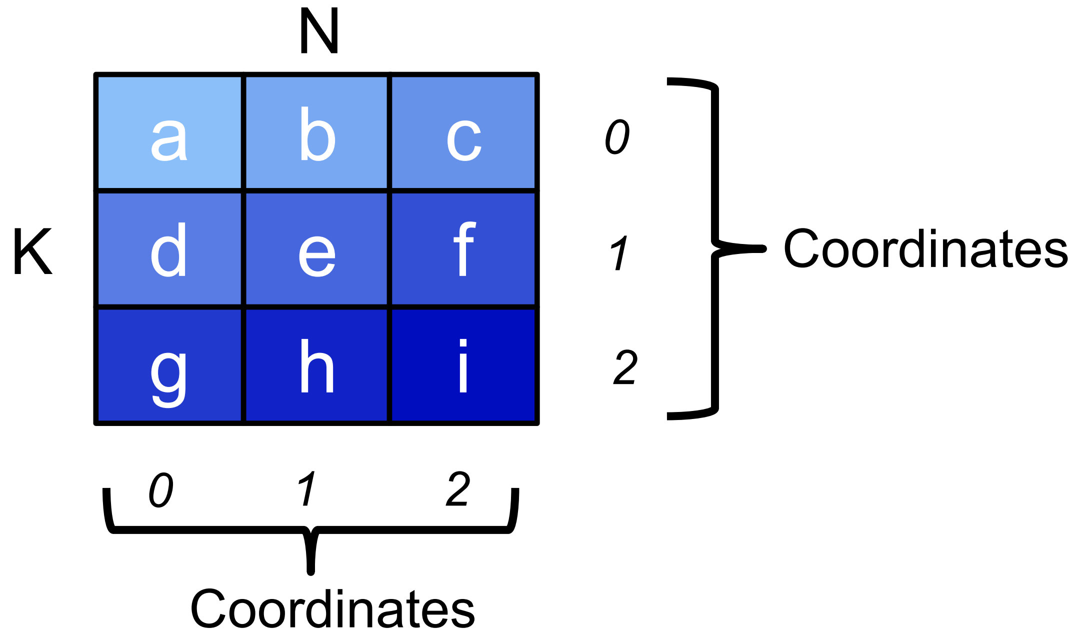
Joel Emer, Vivienne Sze, Yu-Hsin Chen

Motivation

- Leverage CNN sparsity to improve energy-efficiency

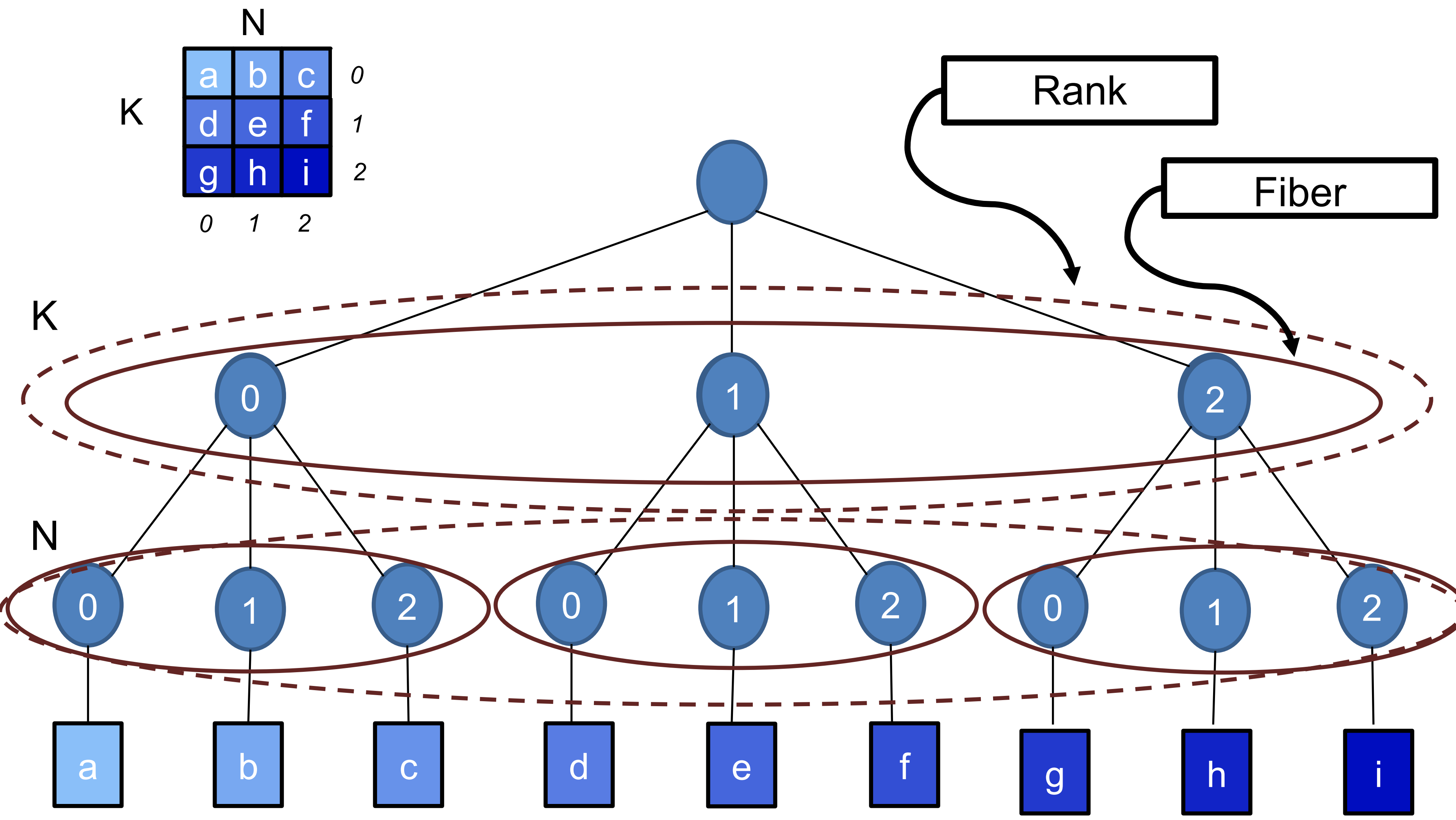


Tensor Data

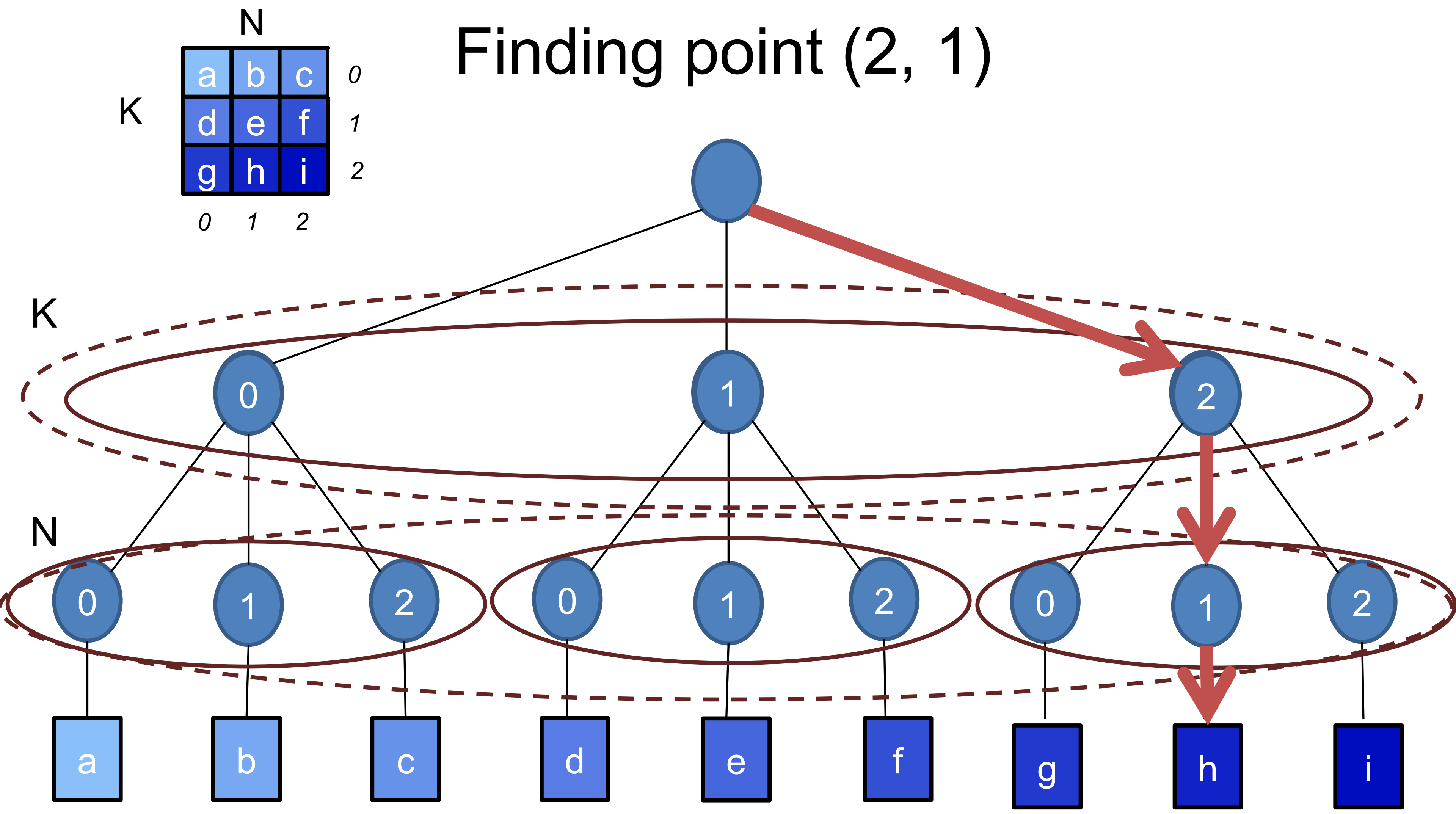


- The elements of each “rank” (dimension) are identified by their “coordinates”, e.g., rank K has coordinates 0, 1, 2
- Each element of the tensor is identified by the tuple of coordinates from each of its ranks, i.e., a “point”.

Tree-based Tensor Representation

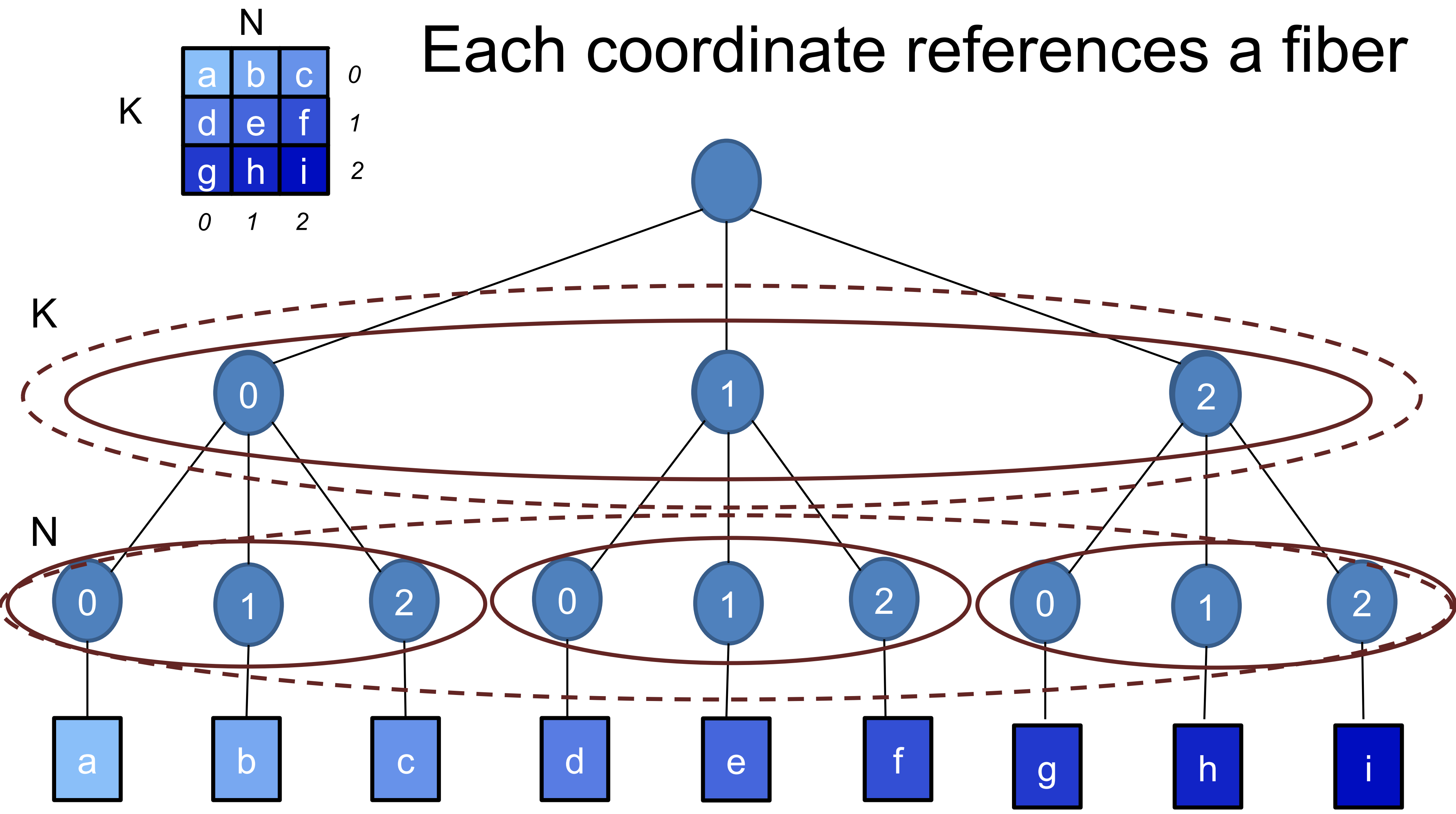


Tree-based Tensor Representation

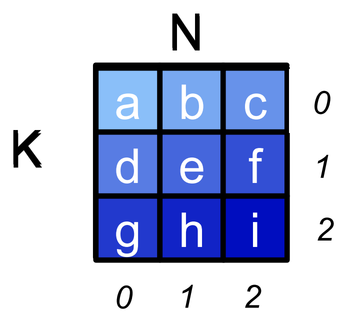


Tree-based Tensor Representation

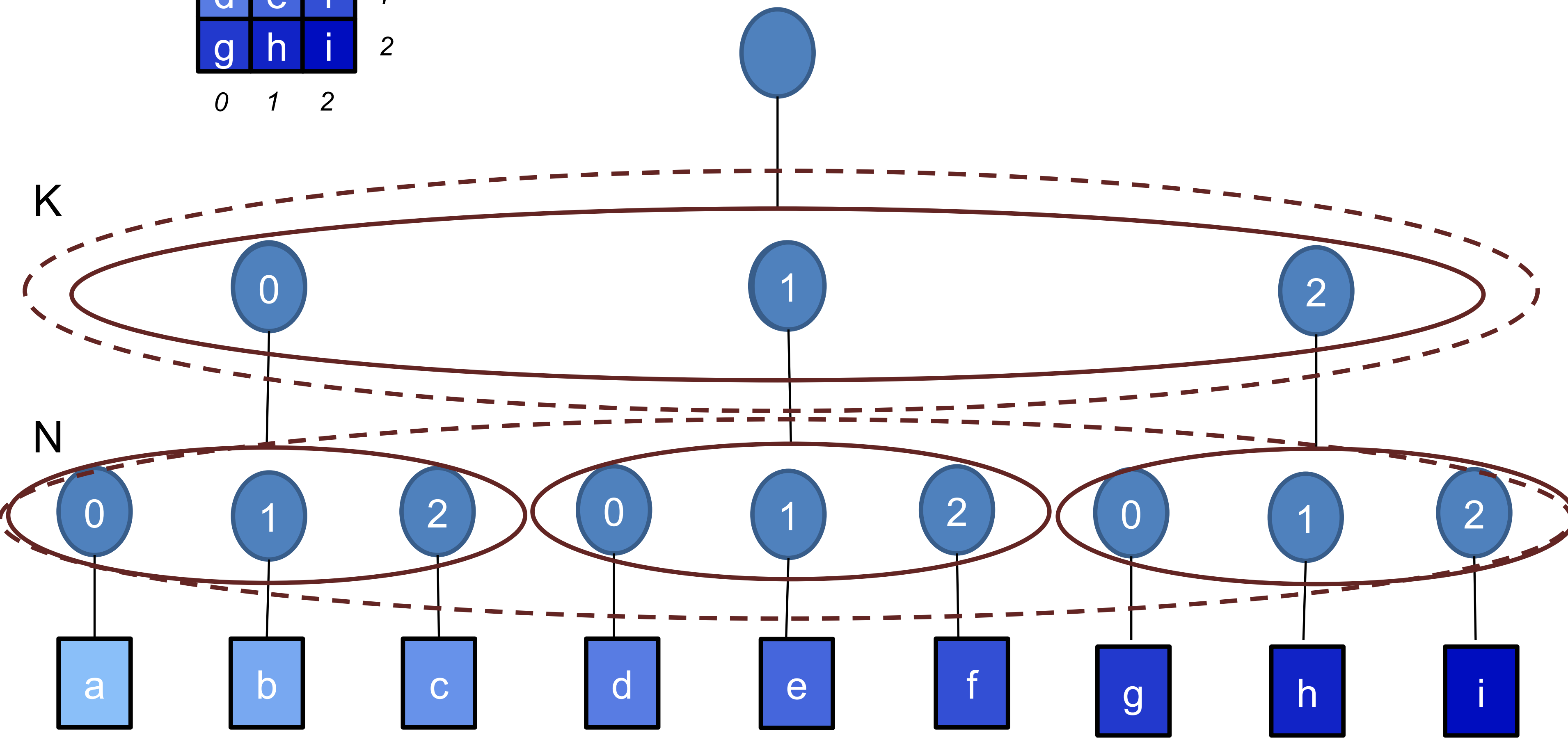
Each coordinate references a fiber



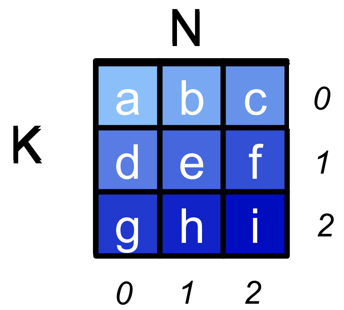
Rank-based Tensor Representation



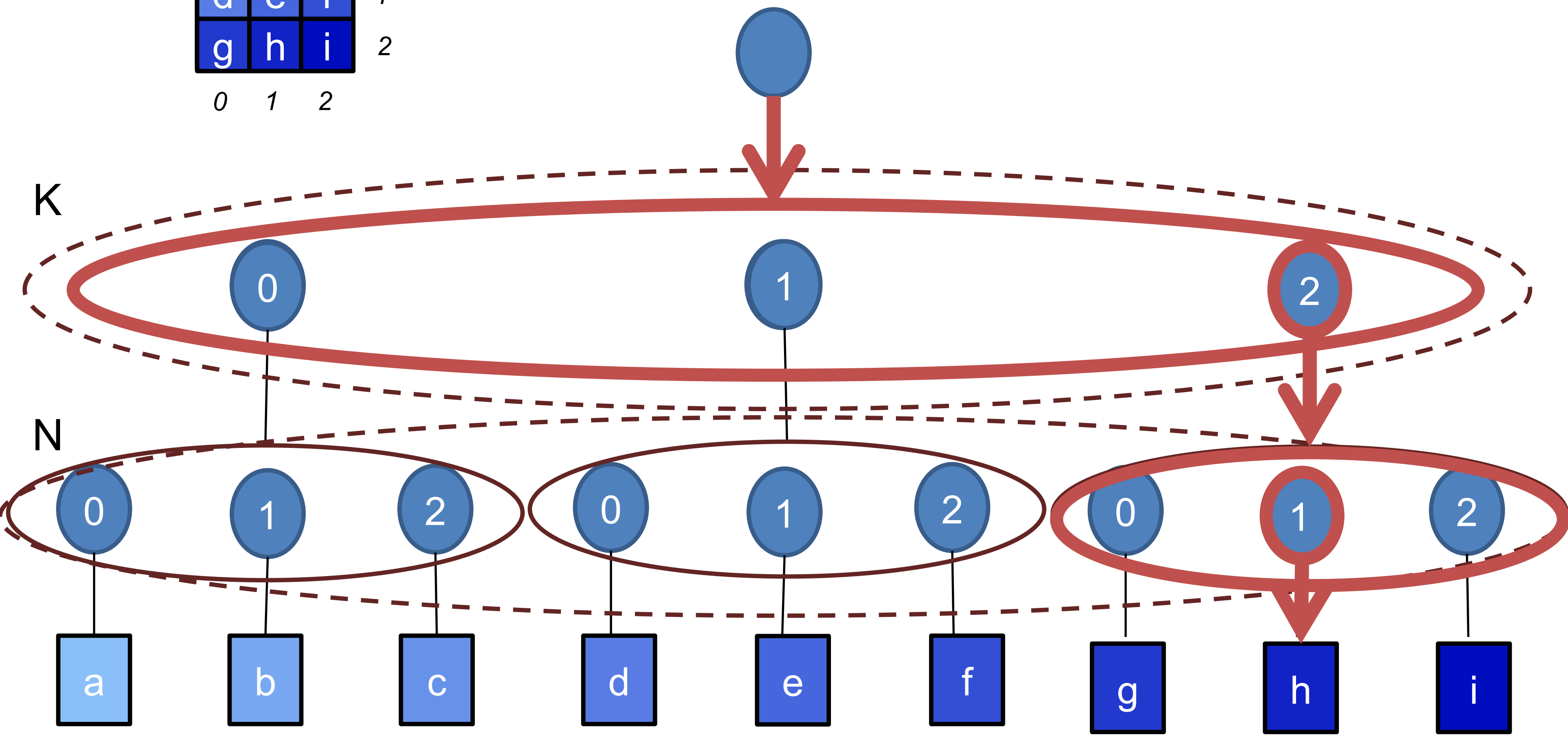
Each coordinate references a fiber



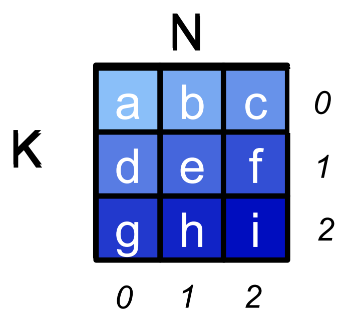
Rank-based Tensor Representation



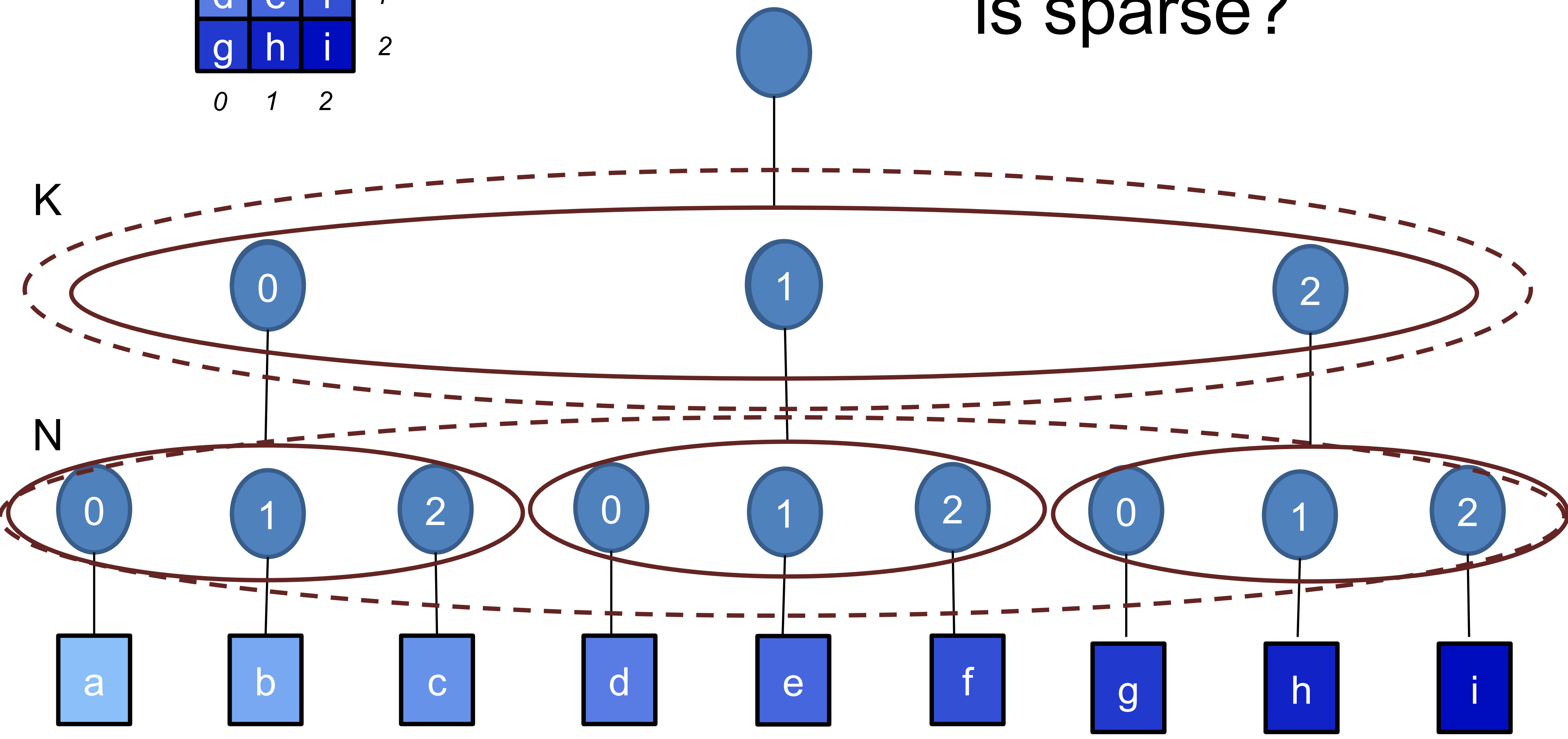
Finding point (2, 1)



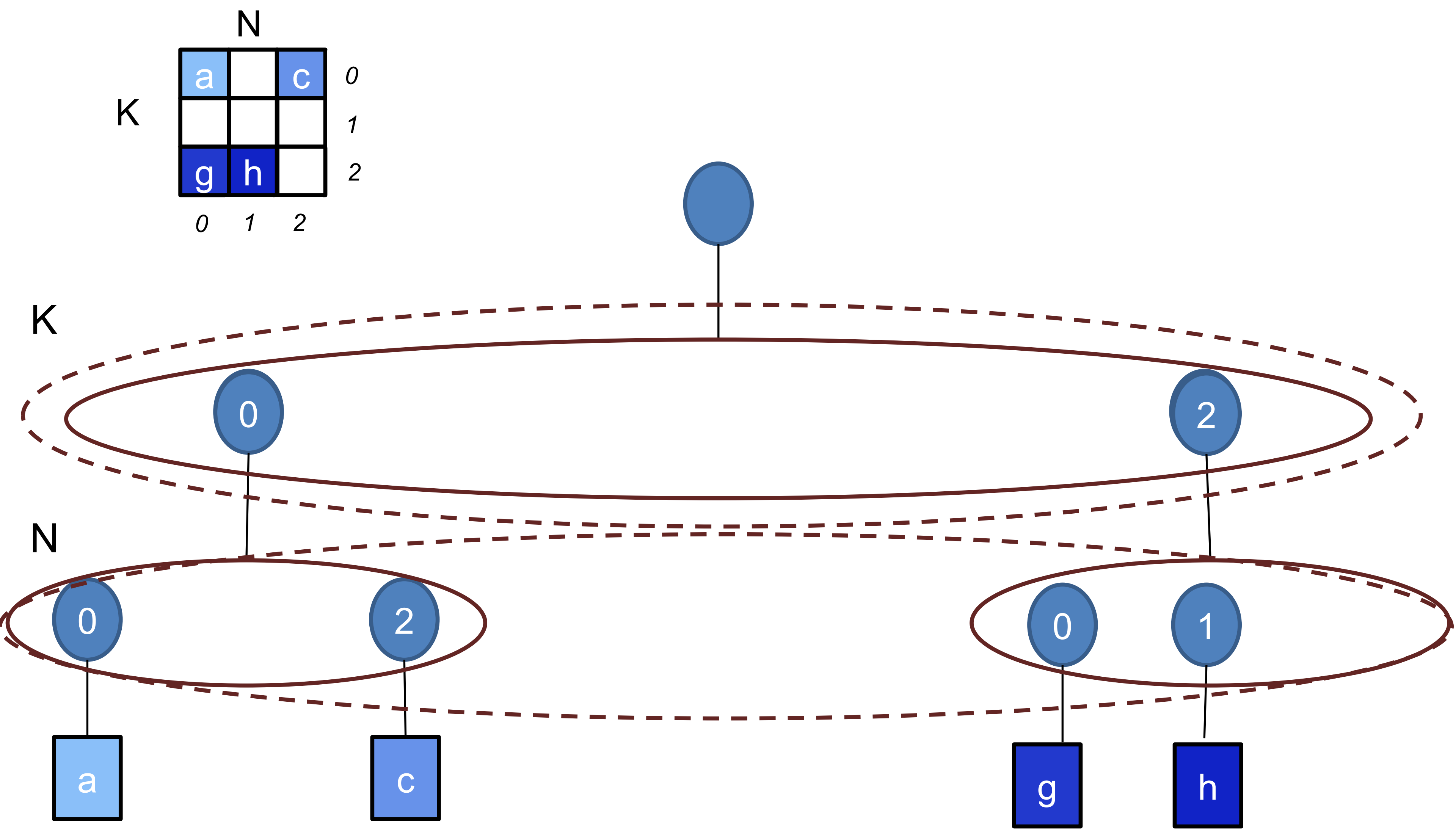
Rank-based Tensor Representation



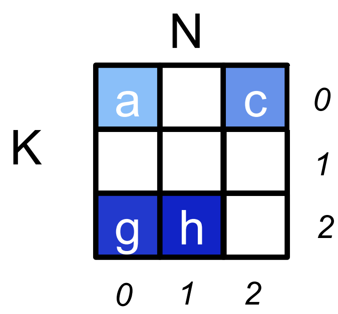
What if tensor is sparse?



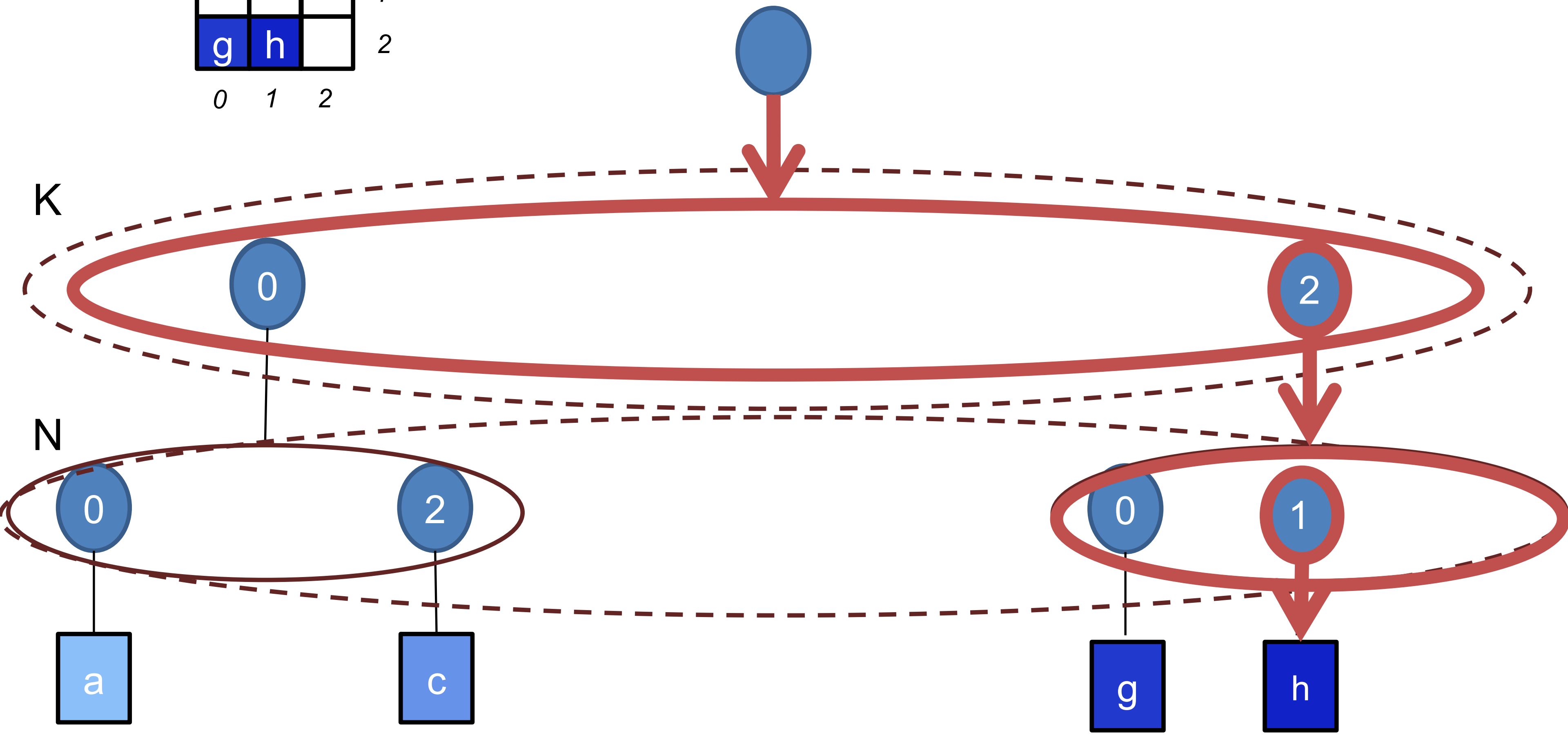
Sparse Tensor Representation



Sparse Tensor Representation

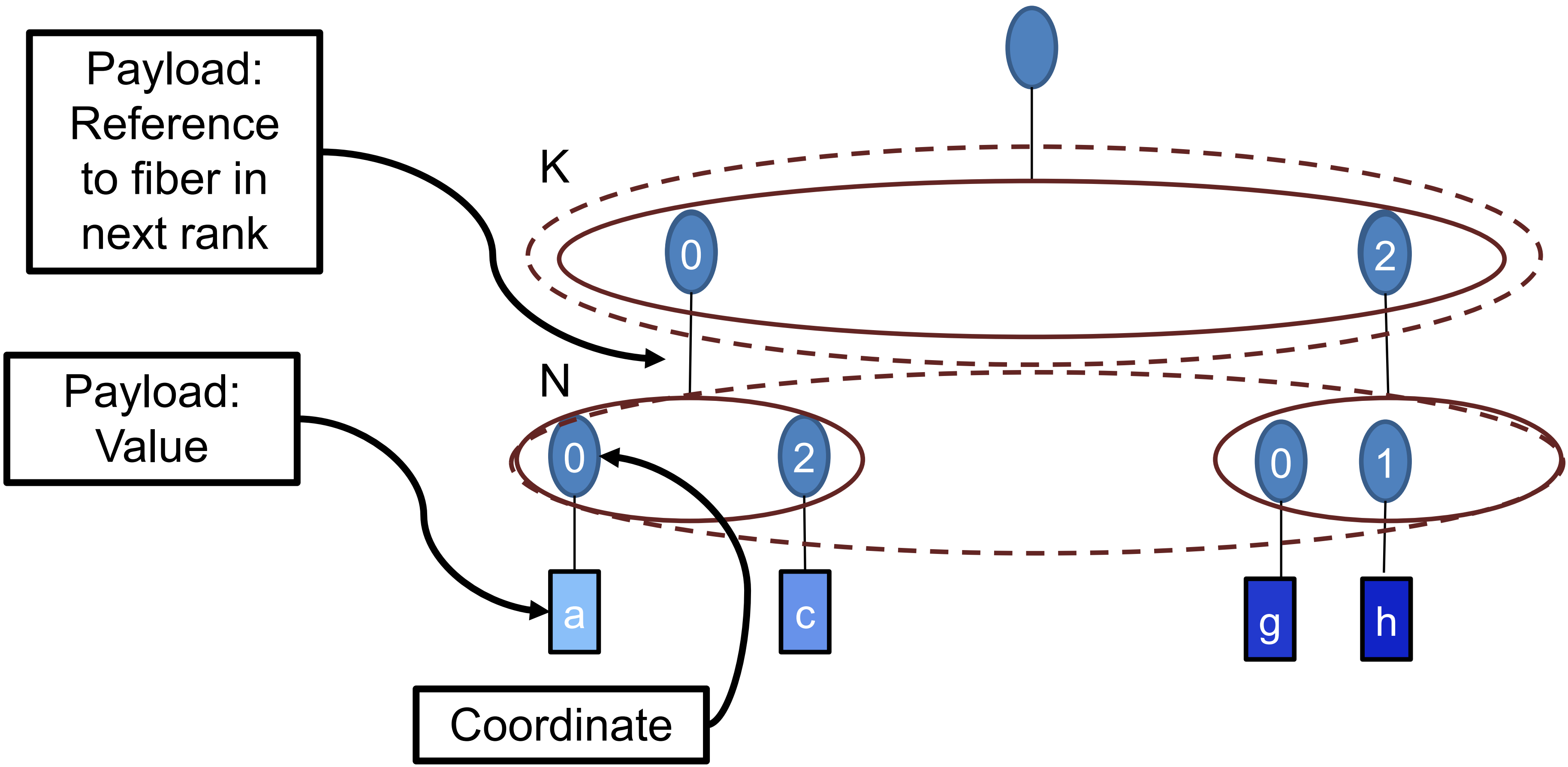


Finding point (2, 1)



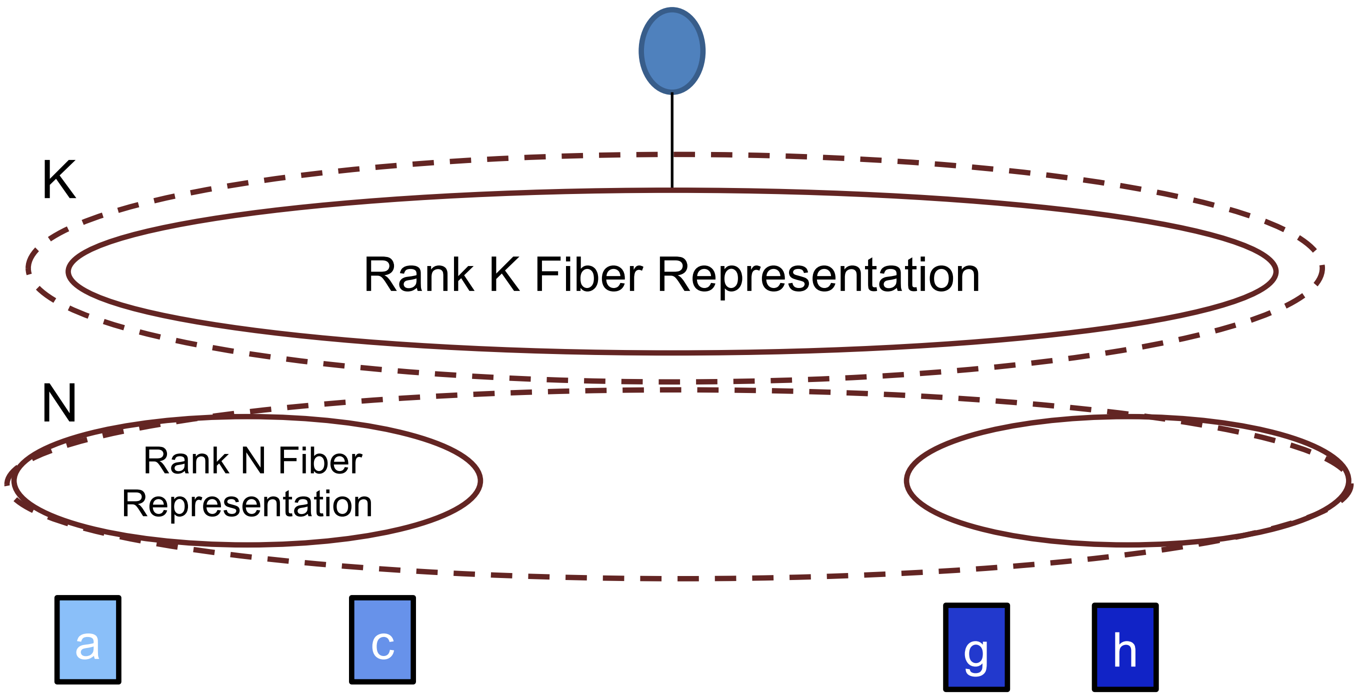
Information in a Fiber

- **Each fiber has set of (coordinate, “payload”) tuples**



Information in a Fiber

Method: payload = fiber.lookup(coordinate)



Fiber Representation Choices

Each fiber has set of (coordinate, “payload”) tuples

- **Implicit Coordinates**
  – Uncompressed (no meta-data required)
  – Compressed – e.g., run length encoded
- **Explicit Coordinates**
  – E.g., coordinate list
- **Space efficiency of a representation depends on sparsity**
  – Compressed format can have overhead relative to uncompressed format for dense data

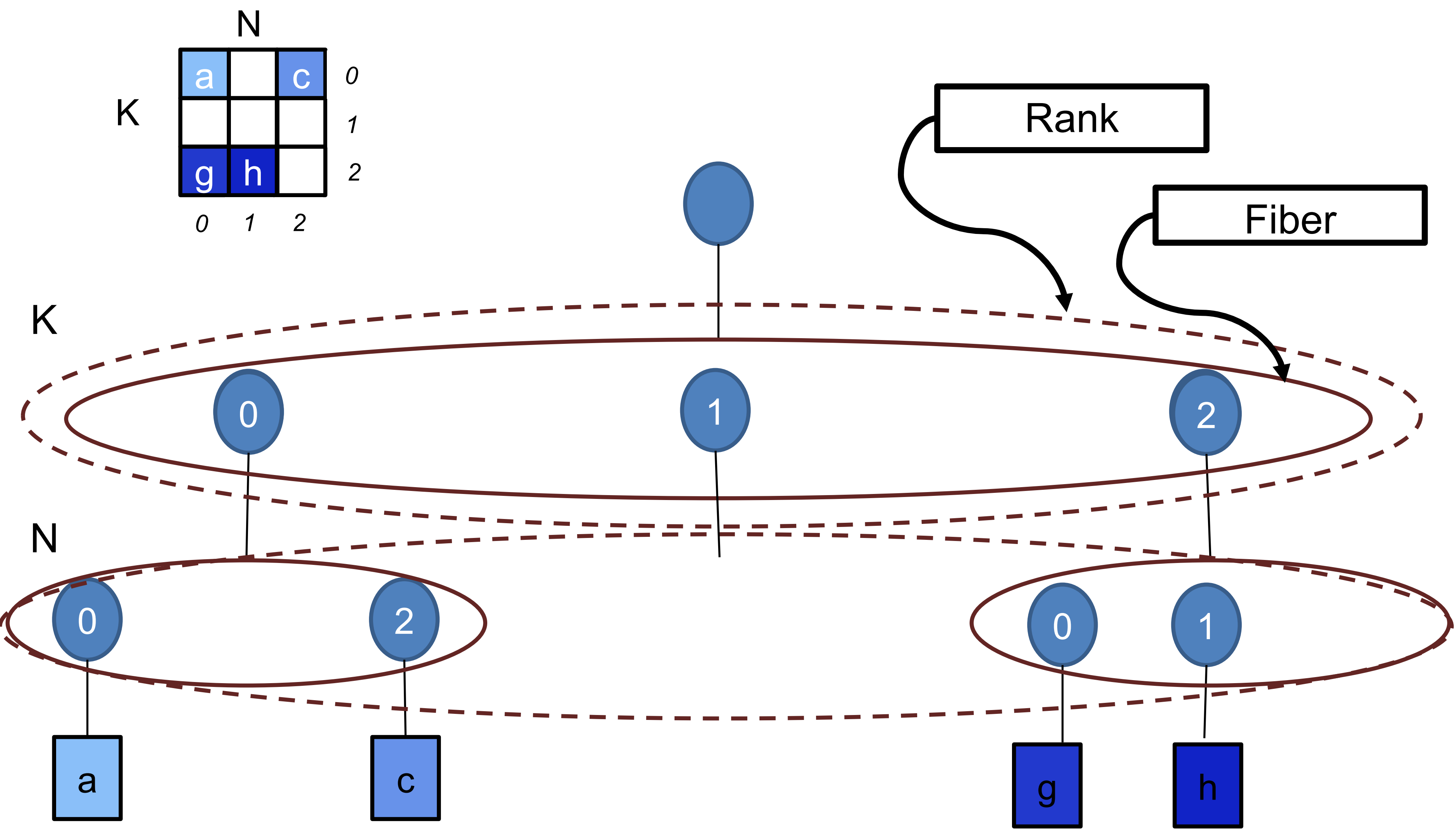
Compressed Implicit Coordinate Representations

- "Empty" coordinate compression via zero-run encoding
  - Run-length coding (RLE)
    - (run-length of zeros, non-zero payload)…
  - Significance map coding
    - (flag to indicate if non-zero, non-zero payload)…

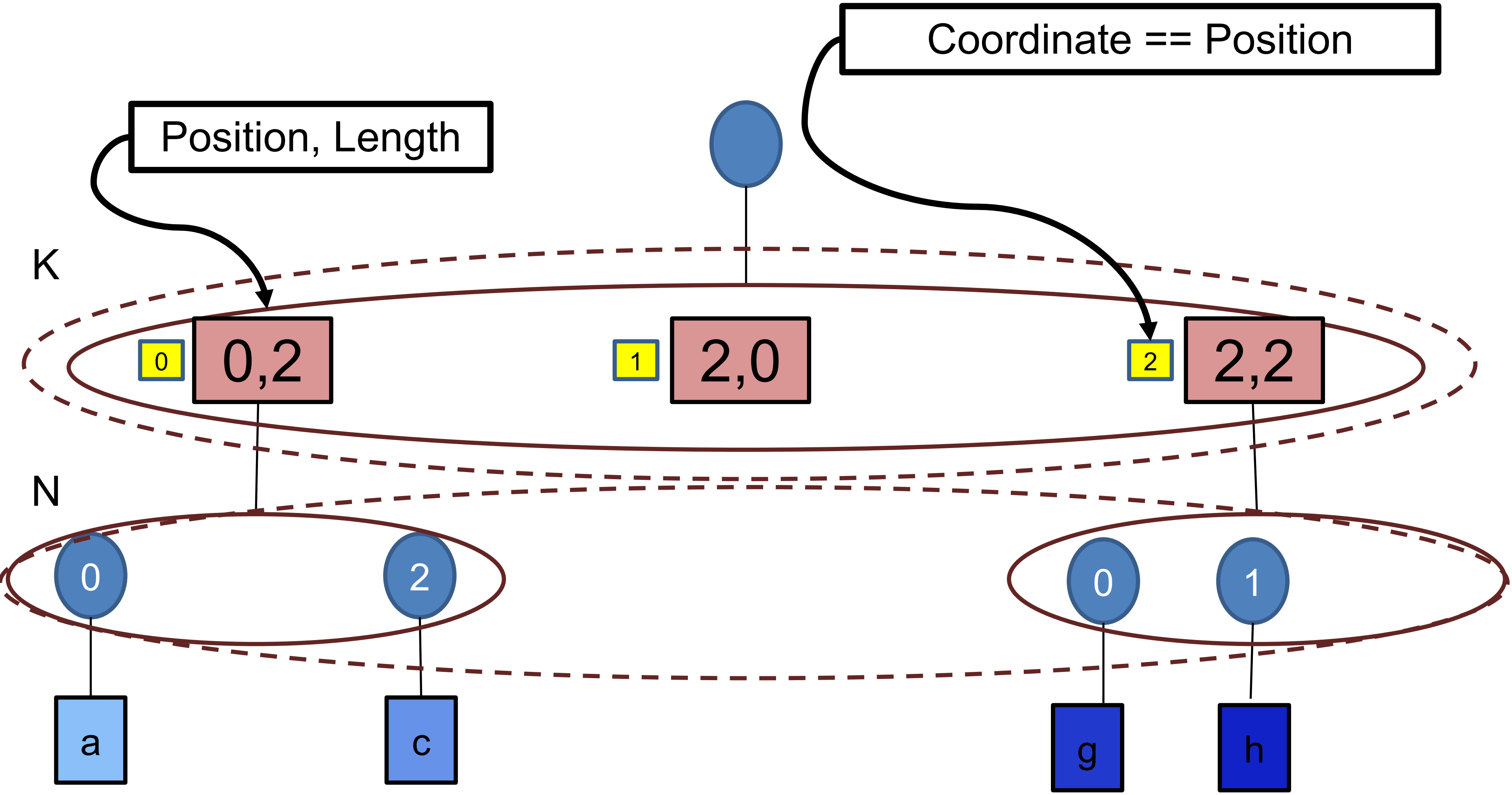
- Payload encoding
  - Fixed length payload
  - Variable length payload
    - E..g., Huffman coding

- Efficiency of different traversal patterns through the tensor is affected by encoding, e.g., finding the payload for a particular coordinate…

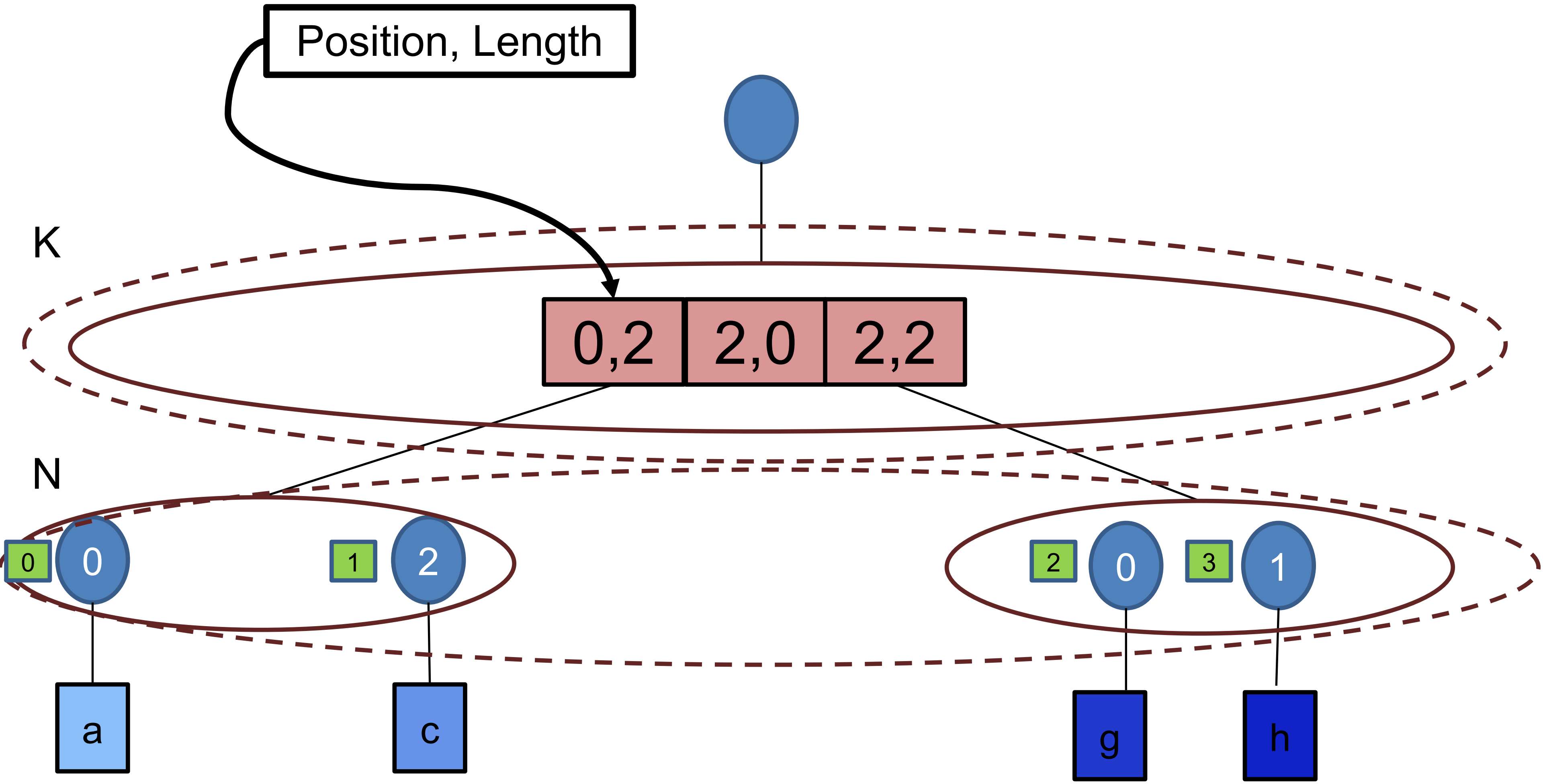
Uncompressed/Compressed Representation



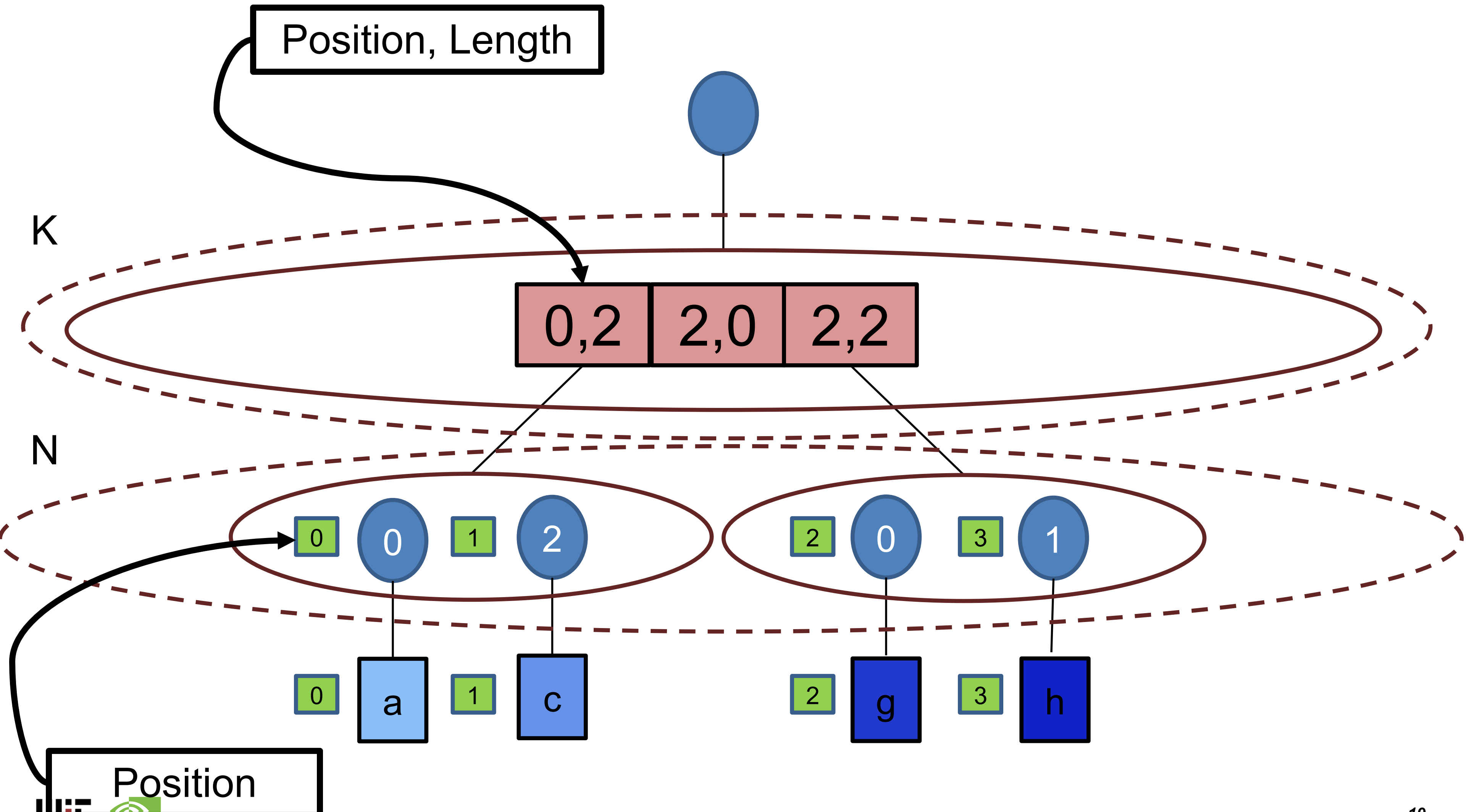
Uncompressed/Compressed Representation



Uncompressed/Compressed Representation

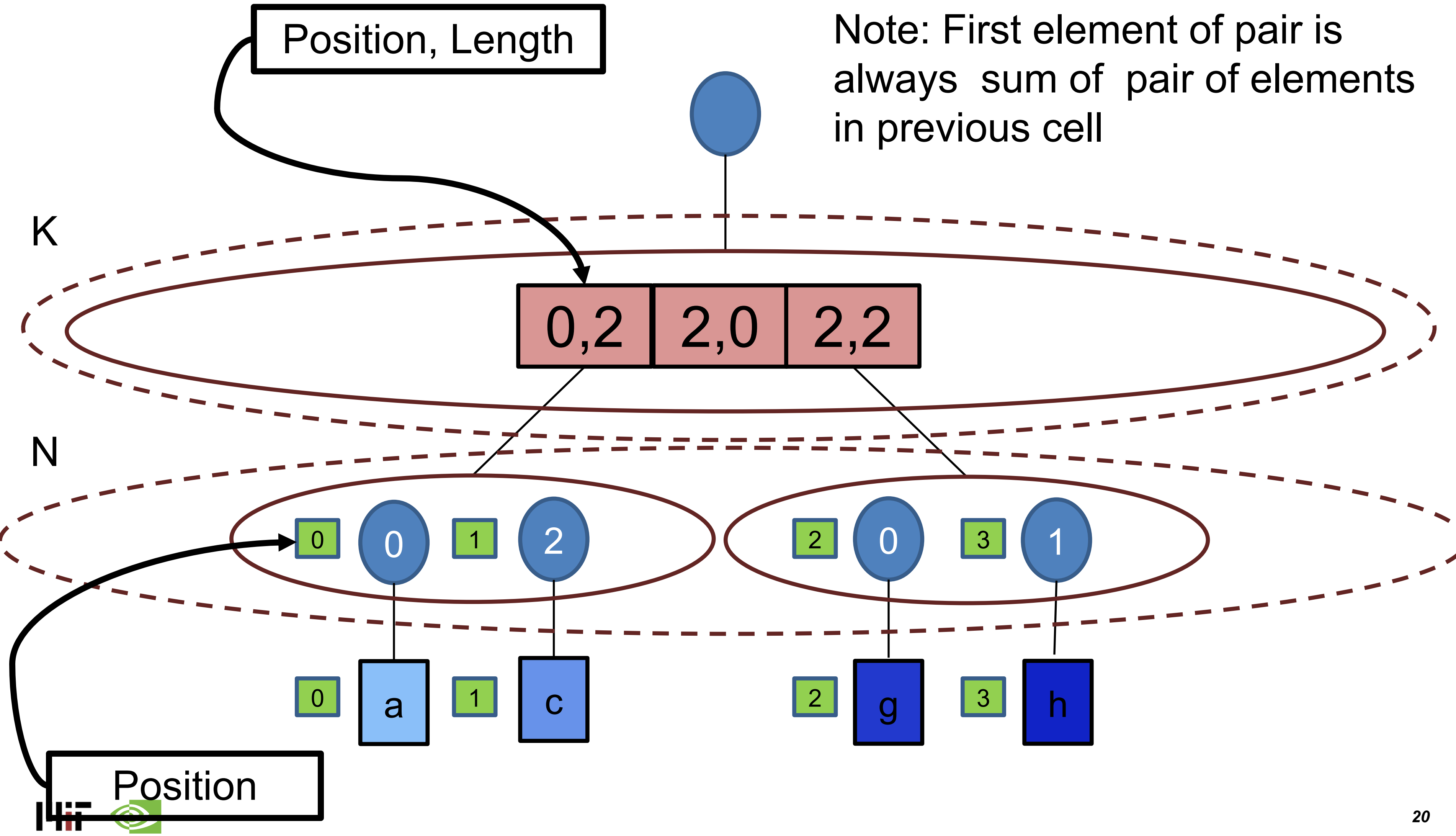


Uncompressed/Compressed Representation

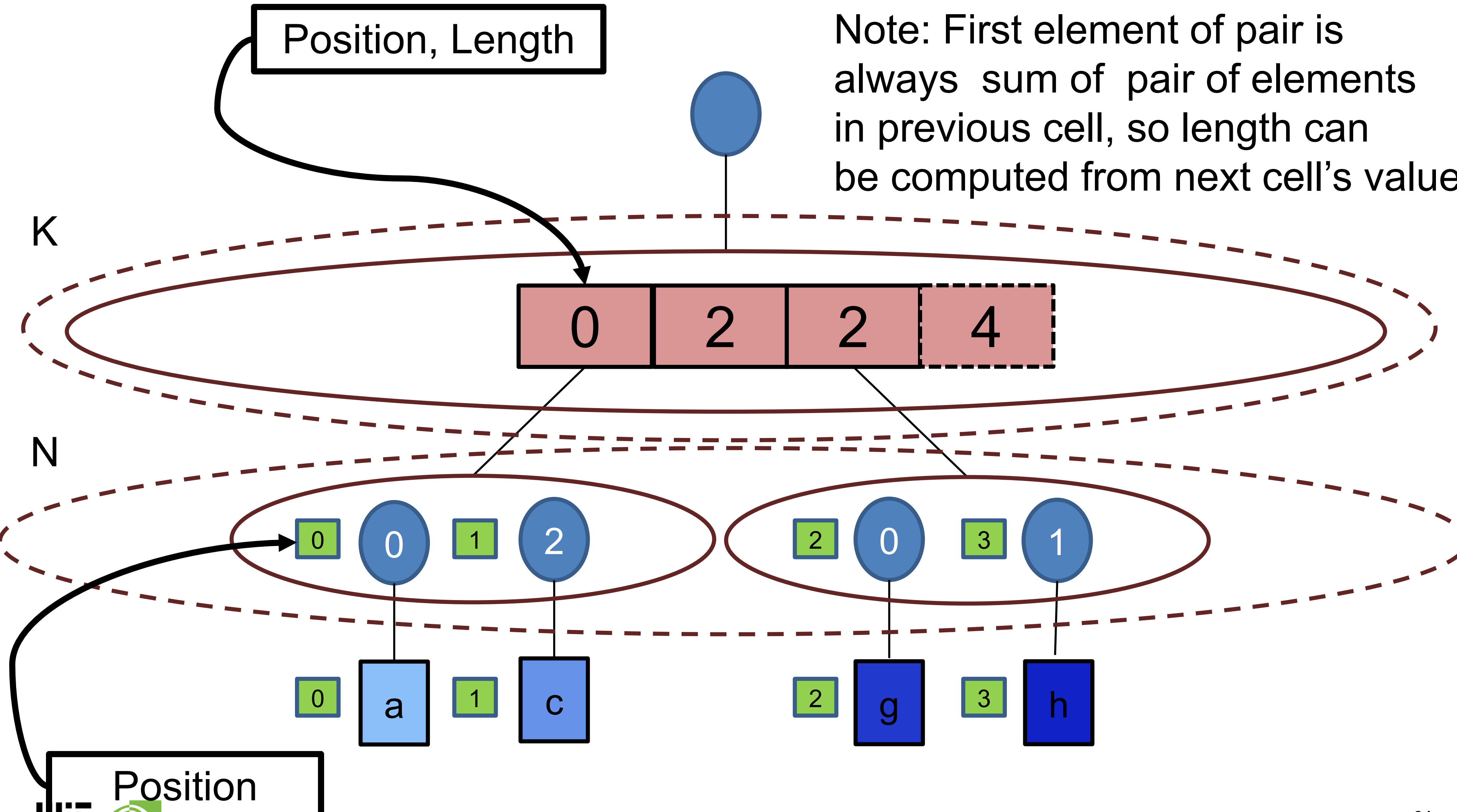


Uncompressed/Compressed Representation

Note: First element of pair is always sum of pair of elements in previous cell

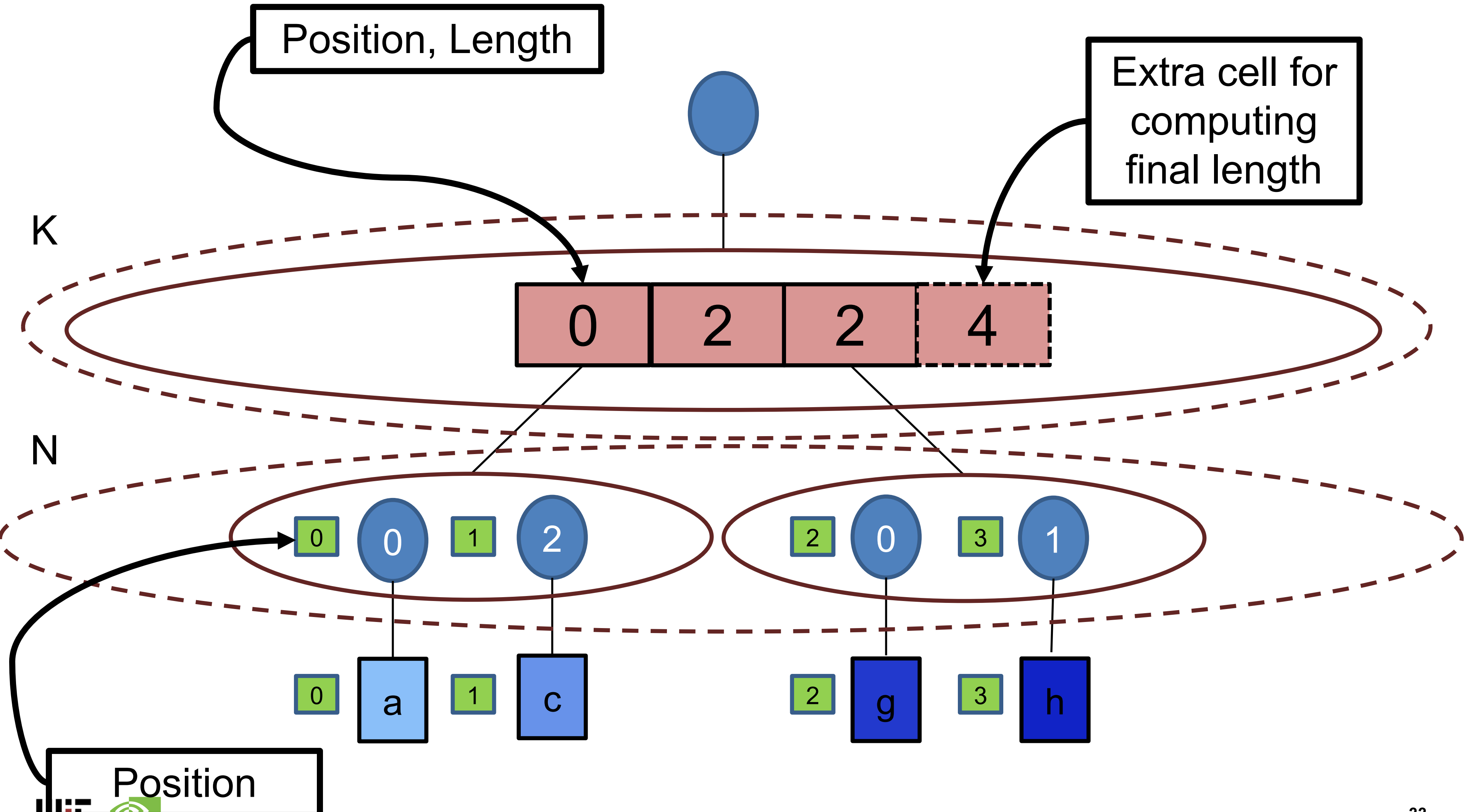


Uncompressed/Compressed Representation

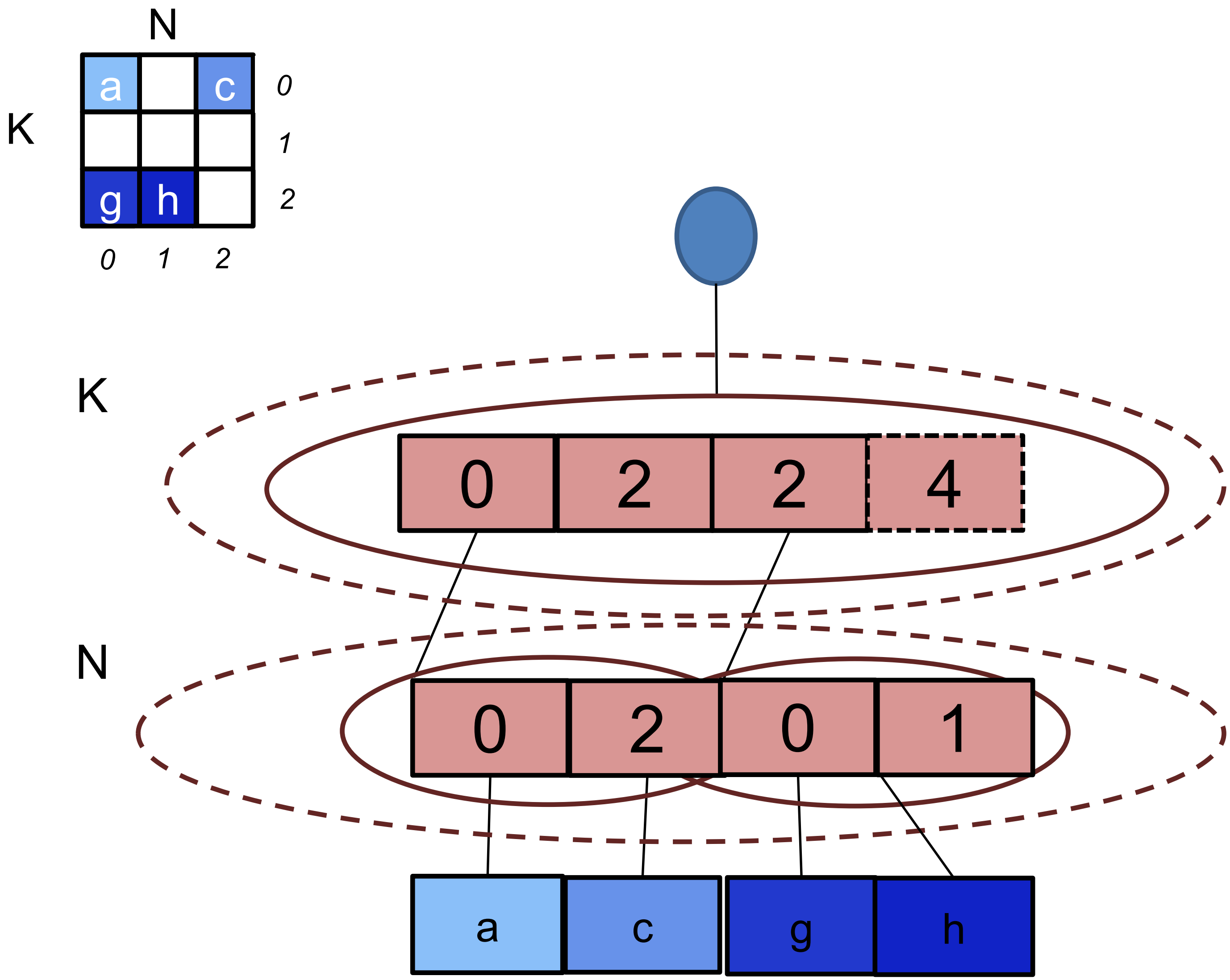


Note: First element of pair is always sum of pair of elements in previous cell, so length can be computed from next cell's value.

Uncompressed/Compressed Representation

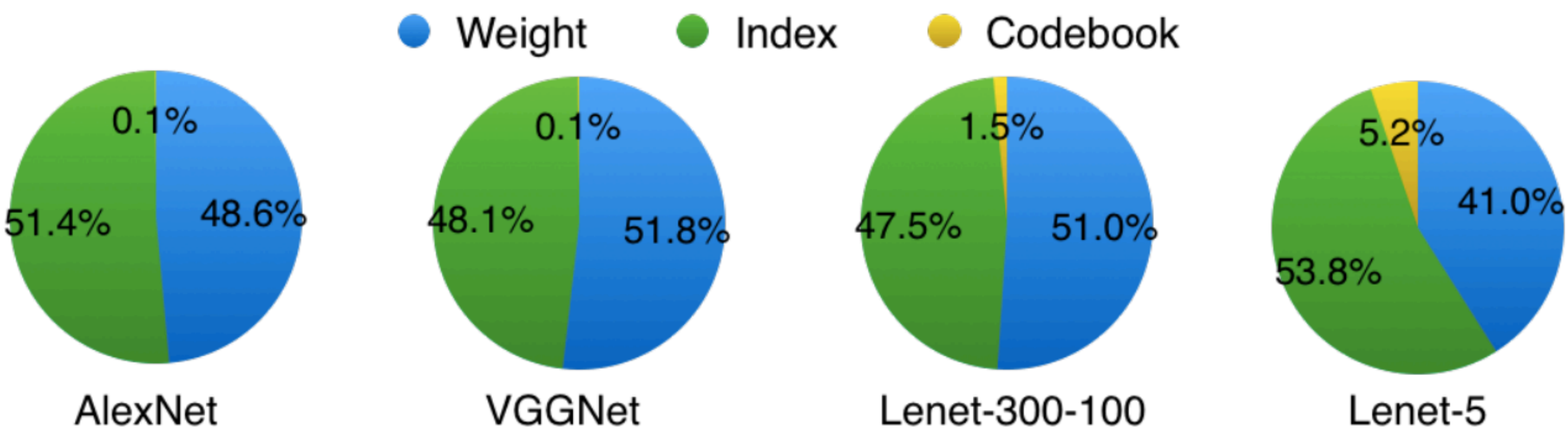


Compressed Sparse Row (CSR)



Compression Overhead

*Index (non-zero position info – e.g., **IA** and **JA** for CSR) accounts for approximately half of storage for fine grained pruning*



[Han et al., ICLR 2016]

Explicit Coordinate Representations

- Coordinate/Payload list
  - (coordinate, non-zero payload)…
- Hash table (per fiber)
  - (coordinate -> payload) mapping
- Hash table (per rank)
  - (fiber_id, coordinate -> payload) mapping
- Bit vector of non-zero coordinates
  - Uncompressed payload

Per Rank Tensor Representations

- Uncompressed [U]
    -
- Run-length Encoded [R]
    -
- Coordinate/Payload List [C]
    -
- Hash Table (per rank) [H_r]
- Hash Table (per fiber) [H_f]
    -

Inspired by collaboration with Kjolstad in [Kjolstad, TACO]

Payload Needed to Find Fiber in a Rank

– Uncompressed [U]
  • Payload*: None

– Run-length Encoded [R]
  • Payload*: Pointer to fiber data structure

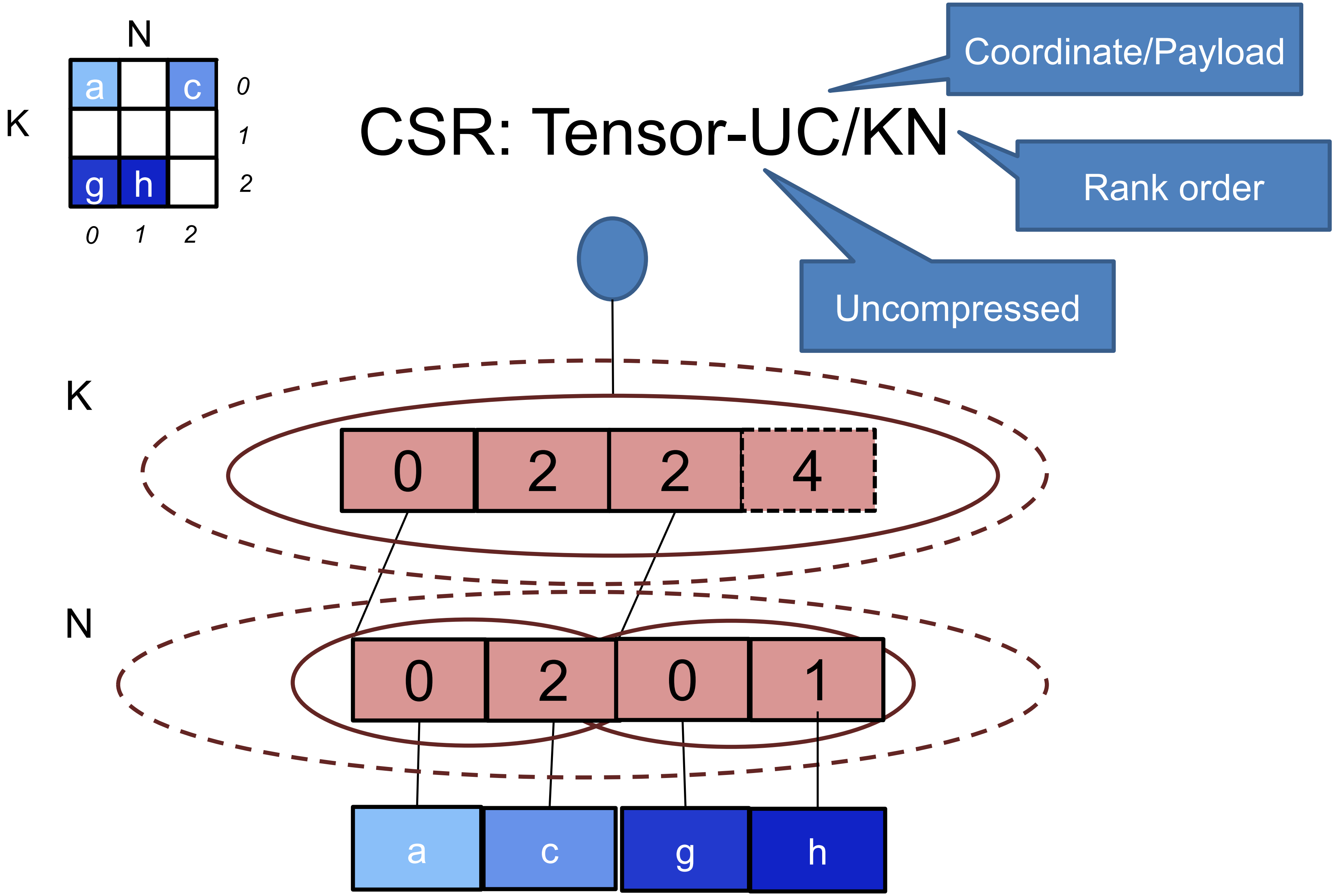
– Coordinate/Payload List [C]
  • Payload*: Pointer to fiber data structure

– Hash Table (per rank) [H_r]
  • Payload*: fiber_id

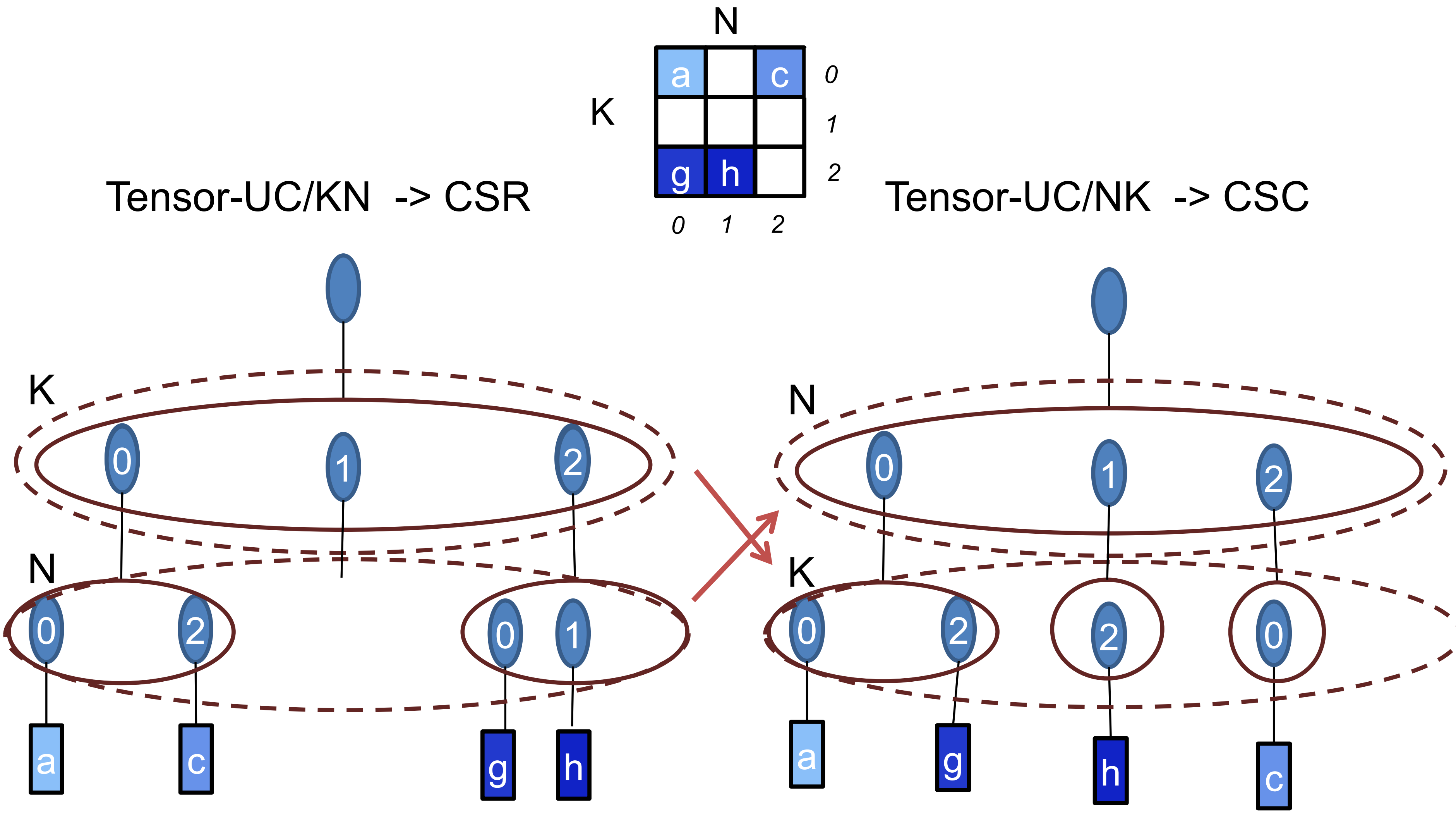
– Hash Table (per fiber) [H_f]
  • Payload*: Pointer to fiber data structure

*Payload needed in preceding rank to perform “lookup():”

Notation for CSR



Representation of Order of Ranks



Traversal Efficiency

Efficiency of different traversal patterns through the tensor is affected by encoding, e.g., finding the payload for a particular coordinate…

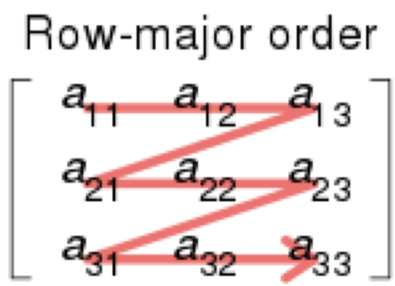
- Operations:
  - payload = Tensor.Locate(coordinate… | point)
  - (coordinate, payload) = Tensor.Next(rank_traversal_order)

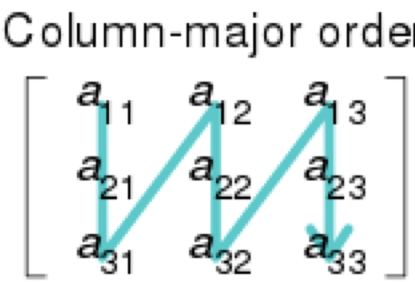
Tensor.next() is a very common operation and its efficiency is highly dependent on representation, both order of ranks and representation of each rank….

Concordant traversal orders

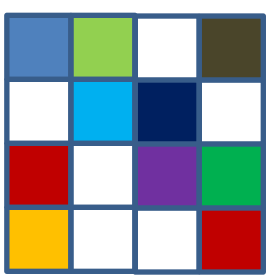
CSR and CSC each has a natural (or "concordant"*) traversal order

Processing Order

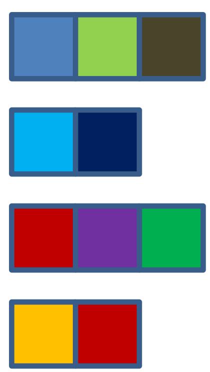




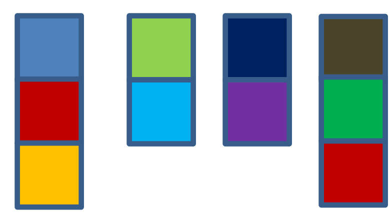
Original Matrix



Compressed Sparse Row (CSR)



Compressed Sparse Column (CSC)



* Term from Michael Pellauer

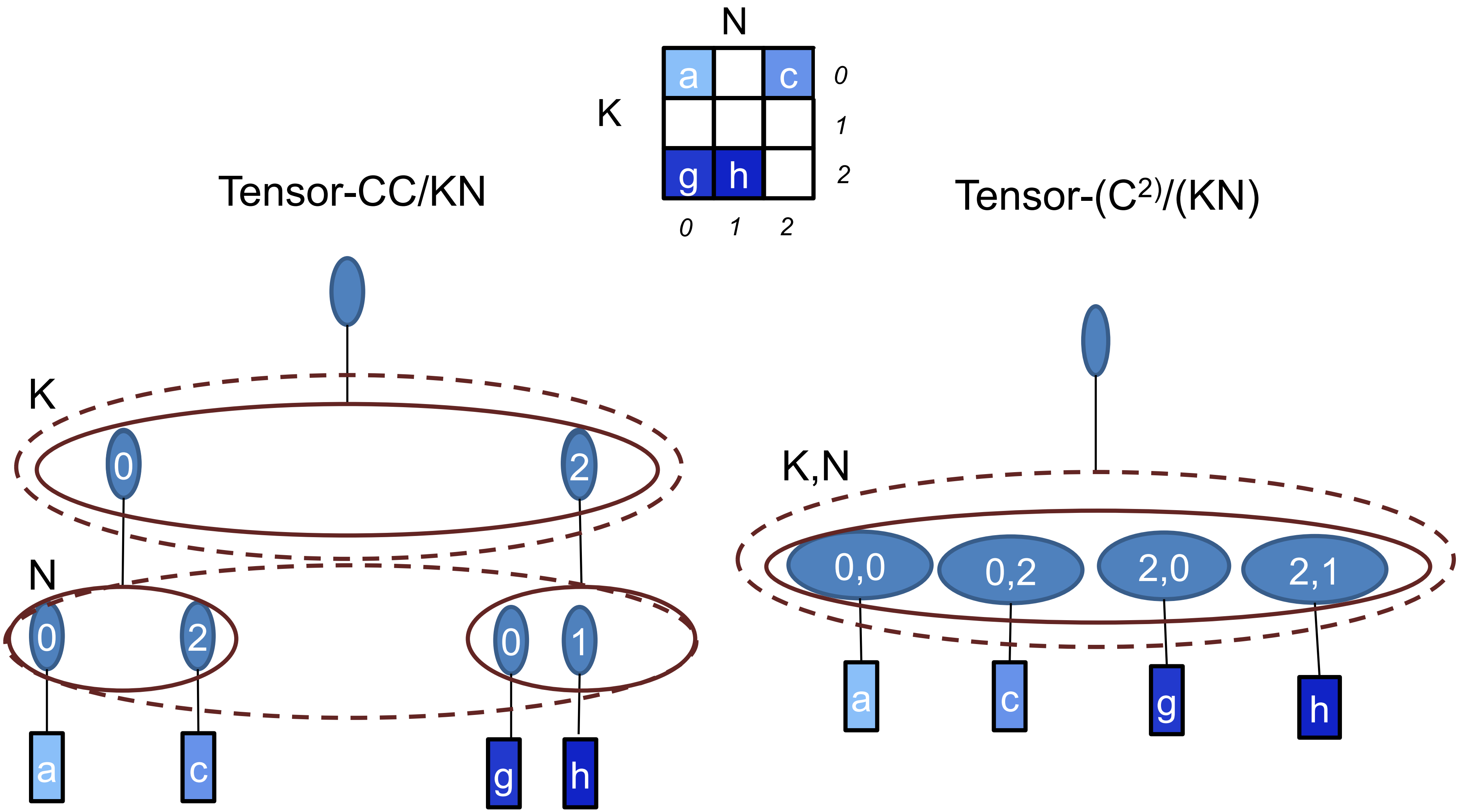
Example Traversal Efficiency

- Locate efficiency:
  - Uncompressed – direct reference - O(1)
  - Run length encoded – linear search – O(n)
  - Hash table – multiple references and compute – O(1)
  - Coordinate/Payload list – binary search – O(log n)

- Next efficiency (concordant traversal)
  - Uncompressed – sequential reference, good spatial locality - O(1)
  - Run length encoded – sequential reference – O(1)
  - Coordinate/Payload list - same as uncompressed

- Next efficiency (discordant traversal)
  - Essentially as good (or bad) as locate….

Merging Ranks



Merging Ranks

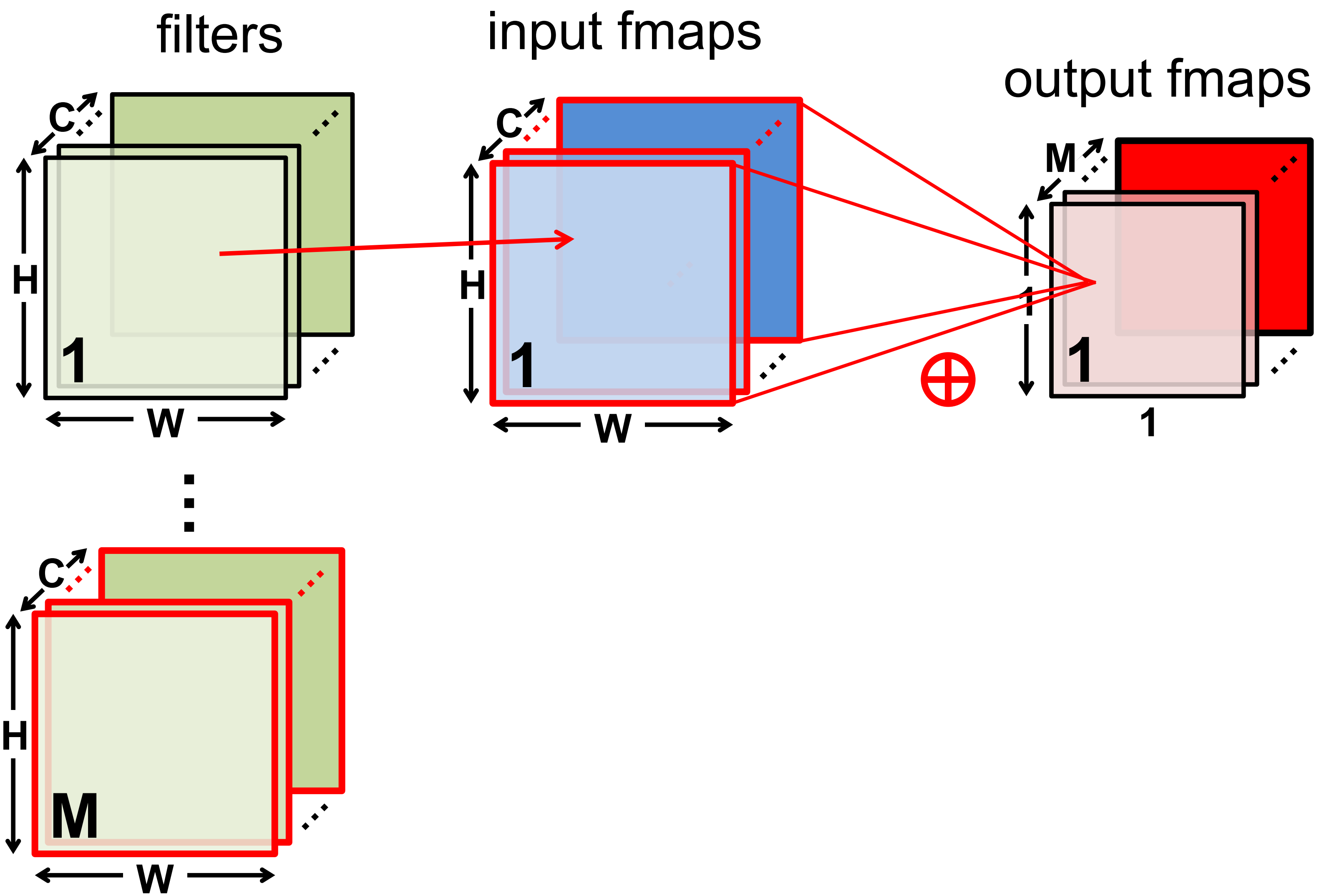
- For efficiency one can form new representations where the data structure for two or more ranks are combined:

- Examples:
  - Tensor-(C^2)

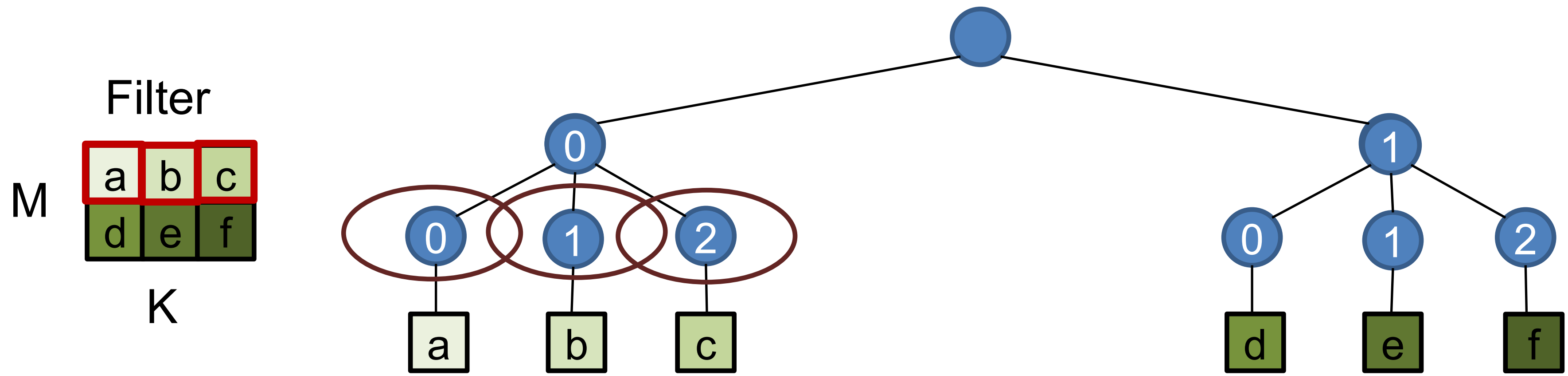
    List of (coordinate tuple,payload) - COO

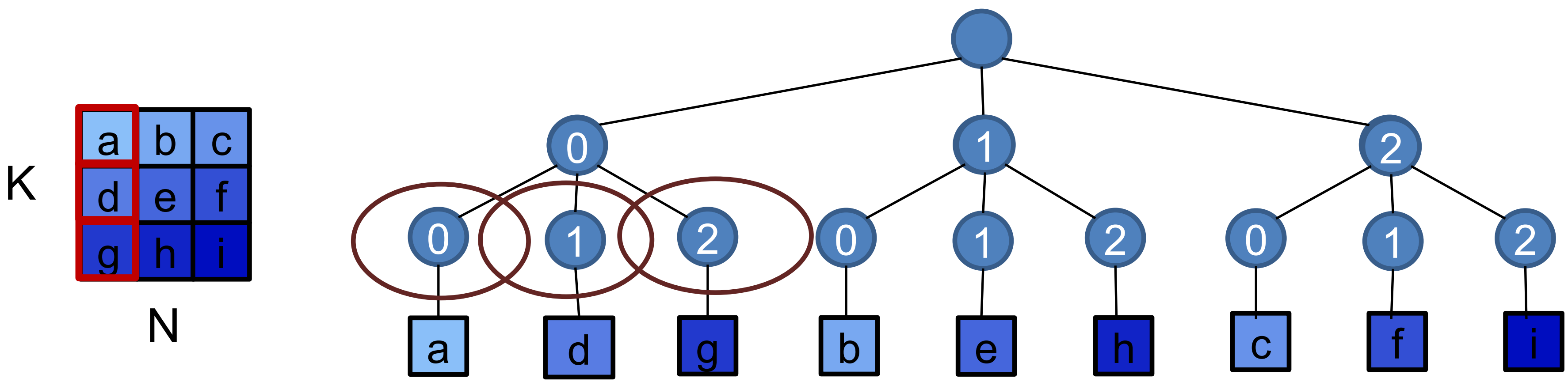
  - Tensor-(H^2)
    - Hash table with coordinate tuple as key
  - Tensor-(U^2)
    - Flattened array
    - Coordinates can be recovered with modulo arithmetic on “position”
  - Tensor-(R^2)
    - Flattened run-length encoded sequence

Fully-Connected (FC) Layer

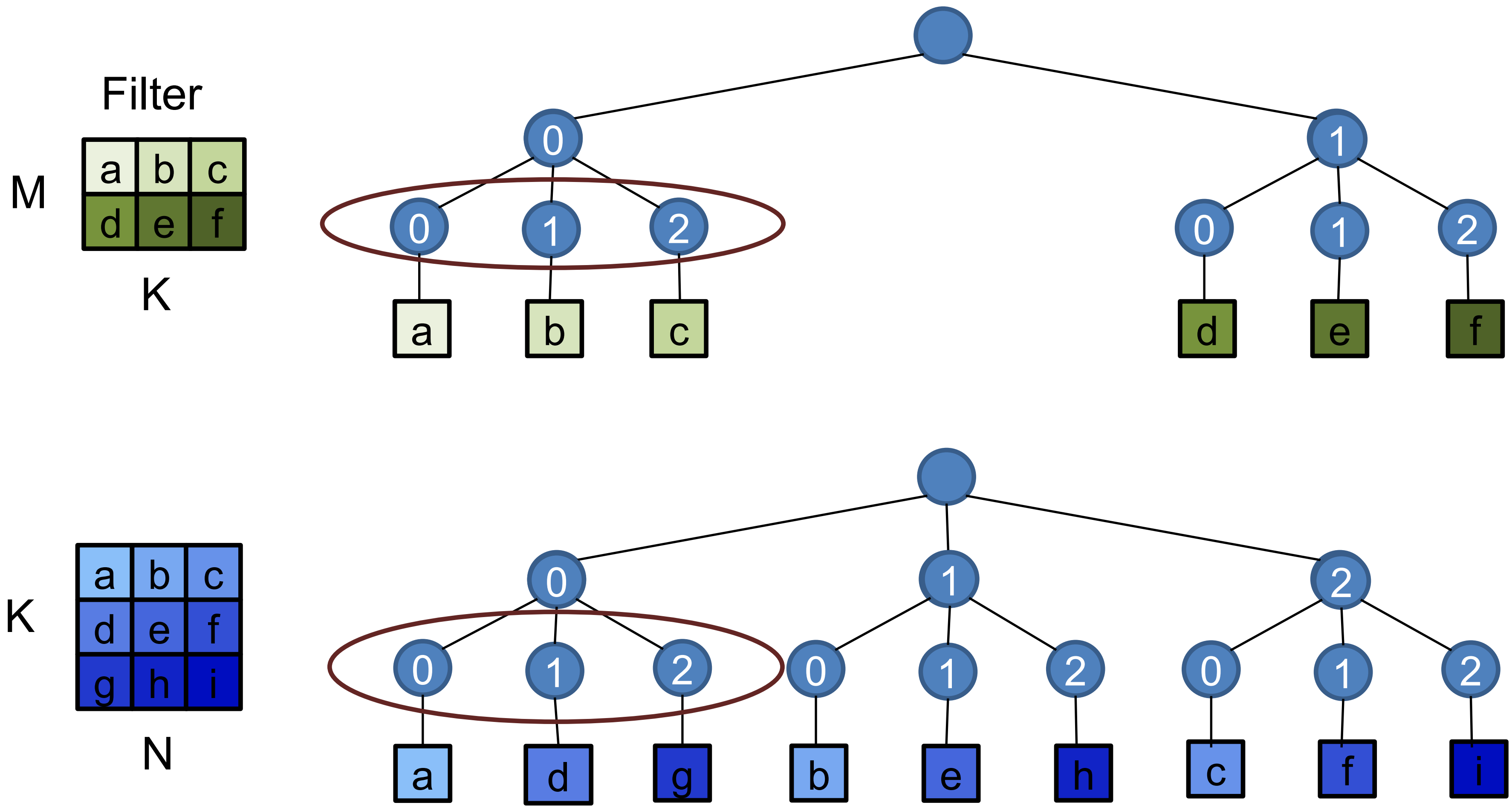


Output-Stationary Operations

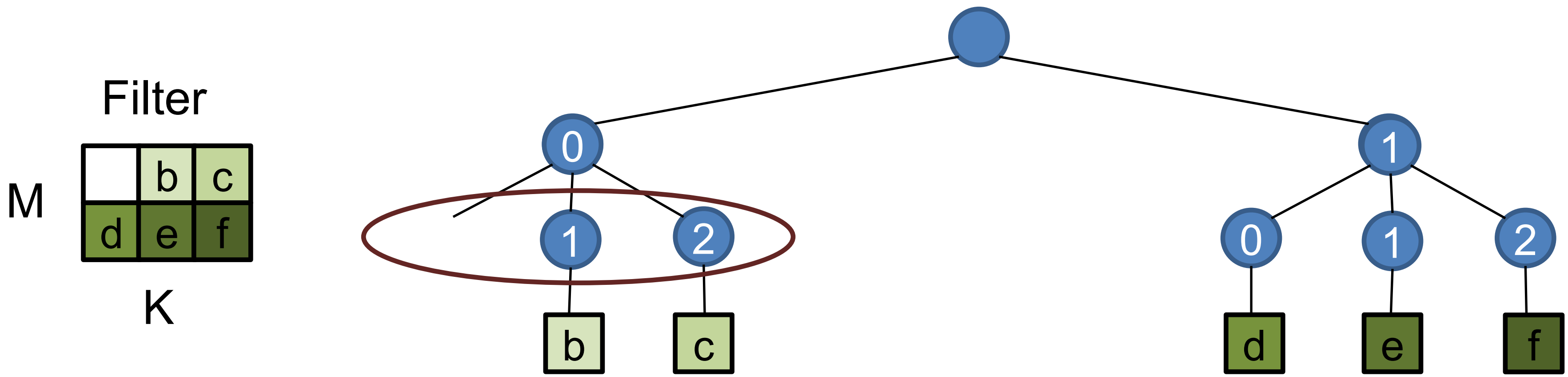


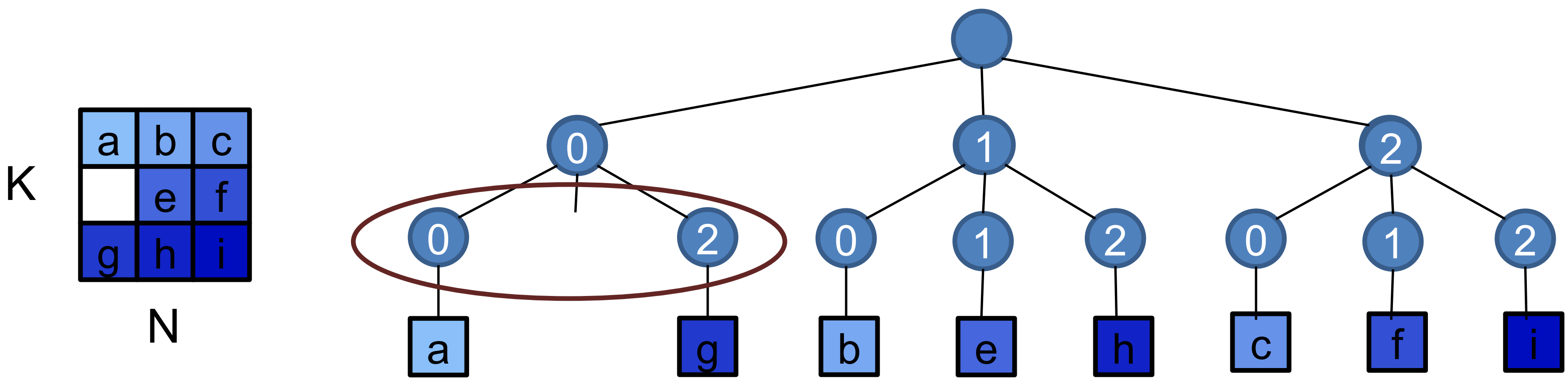


Output-Stationary Lists



Output-Stationary Intersection Lists



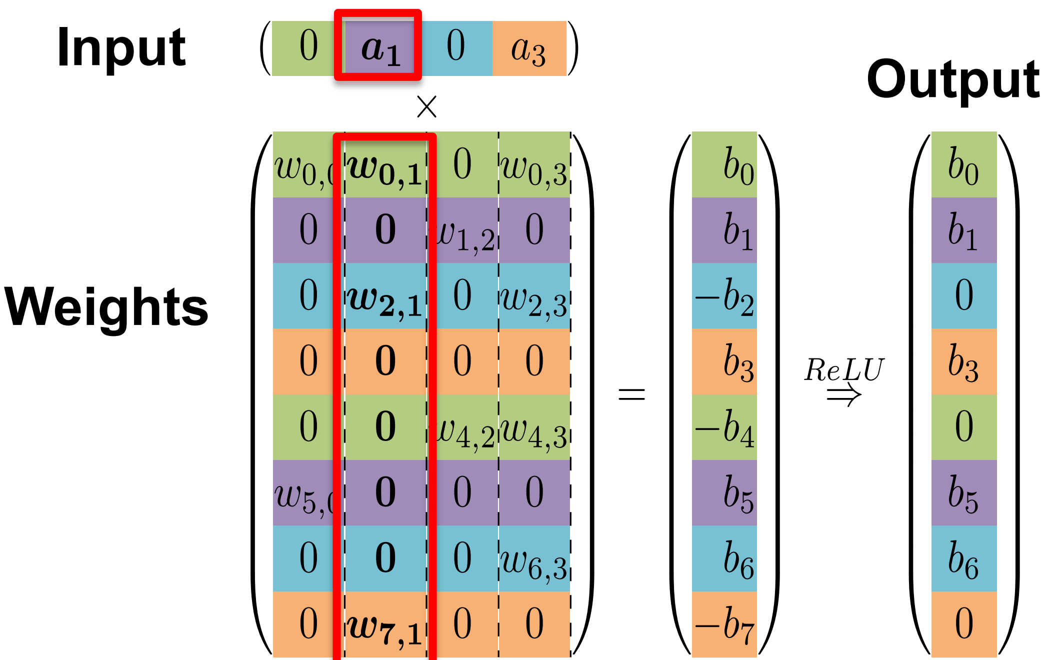


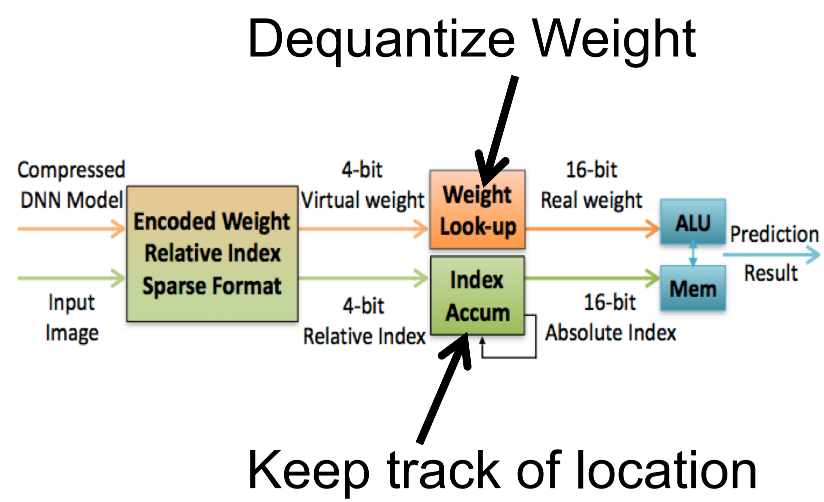
2-2 is the only "effectual" computation

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

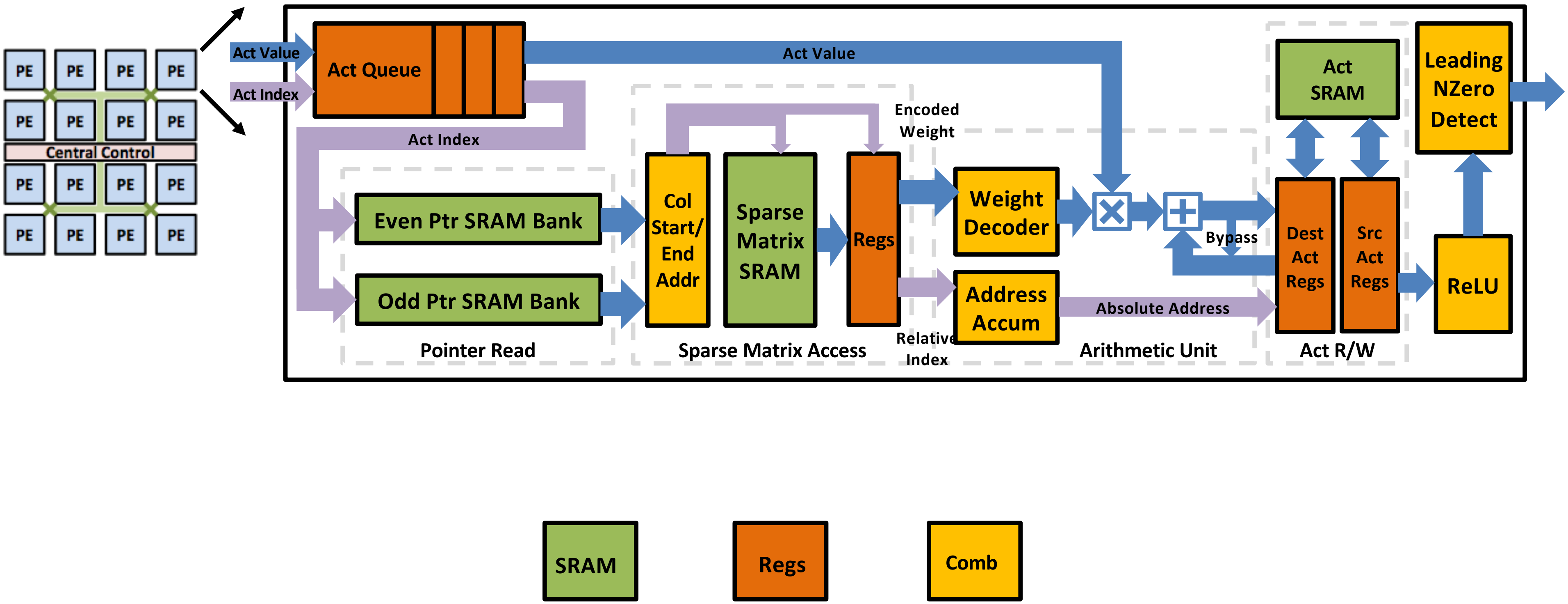




Output Stationary Dataflow

[Han et al., ISCA 2016]

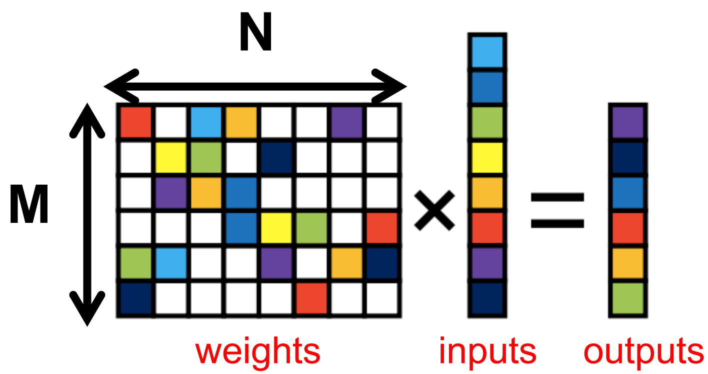
PE Architecture

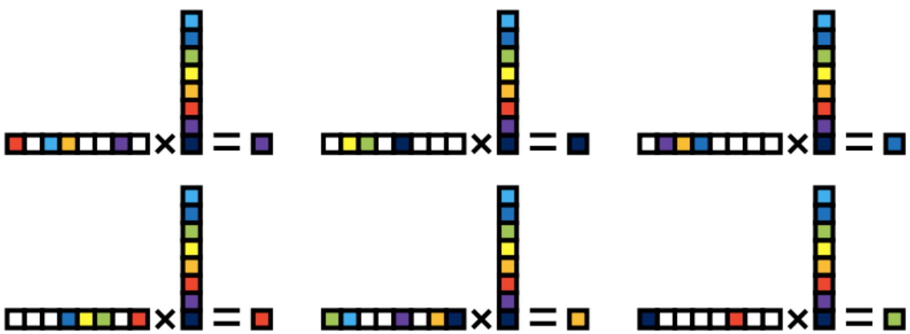


Impact of Representation on Dataflow

From SpMxV research

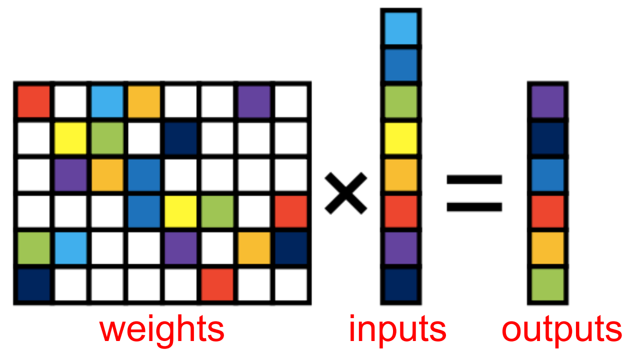
Compressed Sparse Row (CSR)

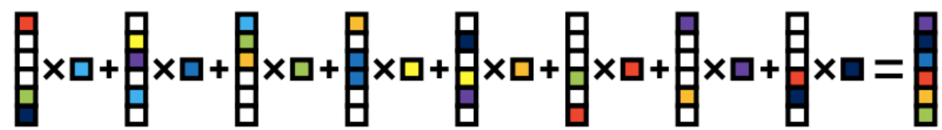




Output stationary

Compressed Sparse Column (CSC)





Input stationary

CSC reduces memory bandwidth over CSR (when not **M** >> **N**)

*For DNN, **M** = # of filters,*
***N** = # of weights per filter*

[Dorrance et al., FPGA 2014]

Sparse Accelerators

1-D Output-Stationary Convolution


```
int i[W];      # Input activations
int w[R];      # Filter weights
int o[E];      # Output activations

for (e = 0; e < E; e++) {
       for (r = 0; r < R; r++) {
          o[e] += i[e+r]*w[r];
}}
```

What opportunity(ies) exist if some of the values are zero?

Can avoid reading operands, doing multiply and updating output

† Assuming: ‘valid’ style convolution

1-D Output-Stationary Convolution

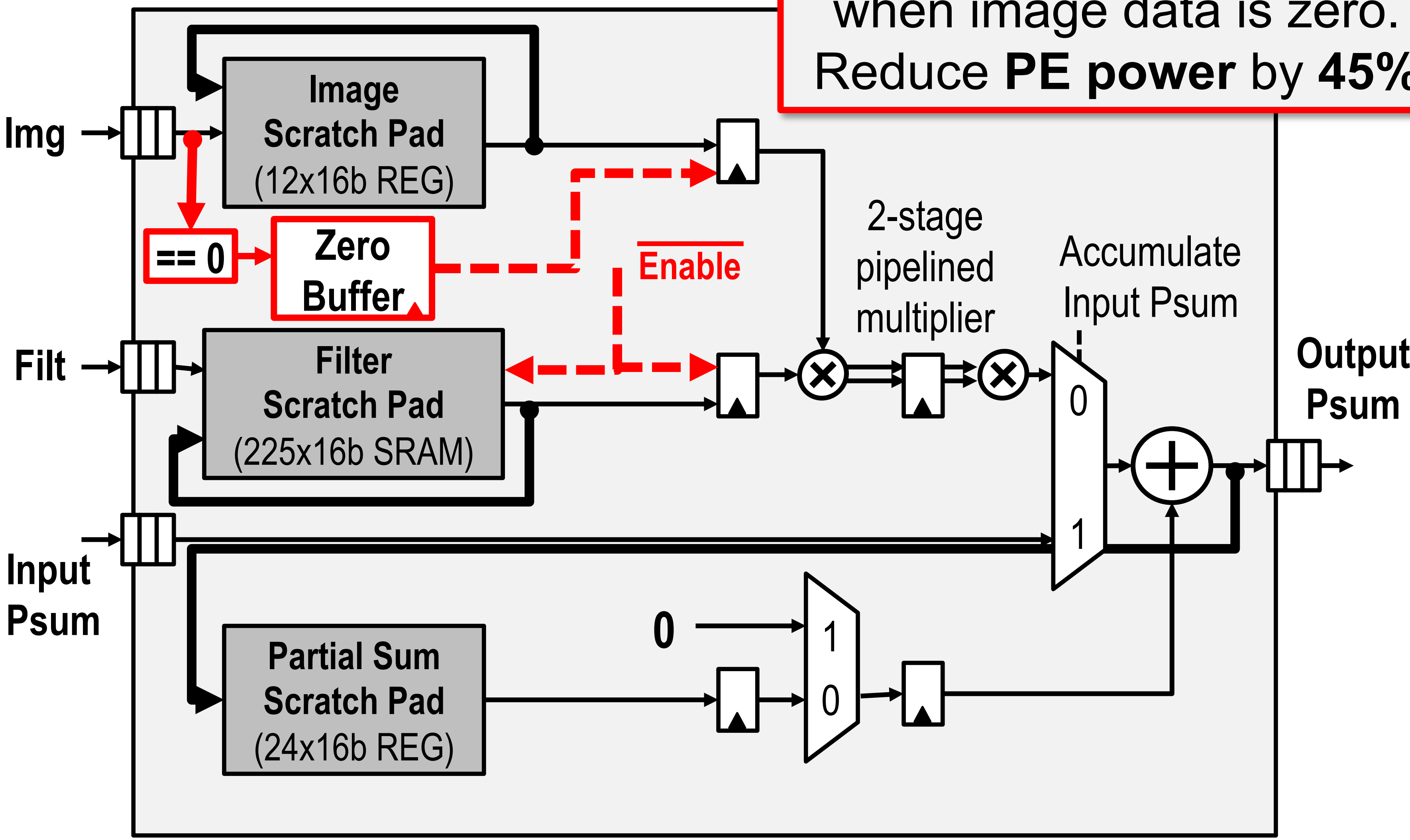

```
int i[W];      # Input activations
int w[R];      # Filter weights
int o[E];      # Output activations

for (e = 0; e < E; e++) {
       for (r = 0; r < R; r++) {
          if (!w[r]) o[e] += i[e+r]*w[r];
}}
```

Saved energy but not time

† Assuming: 'valid' style convolution

Eyeriss – Clock Gating

Skip **mult** and **mem reads** when image data is zero. Reduce **PE power** by **45%**



Compressed Weights

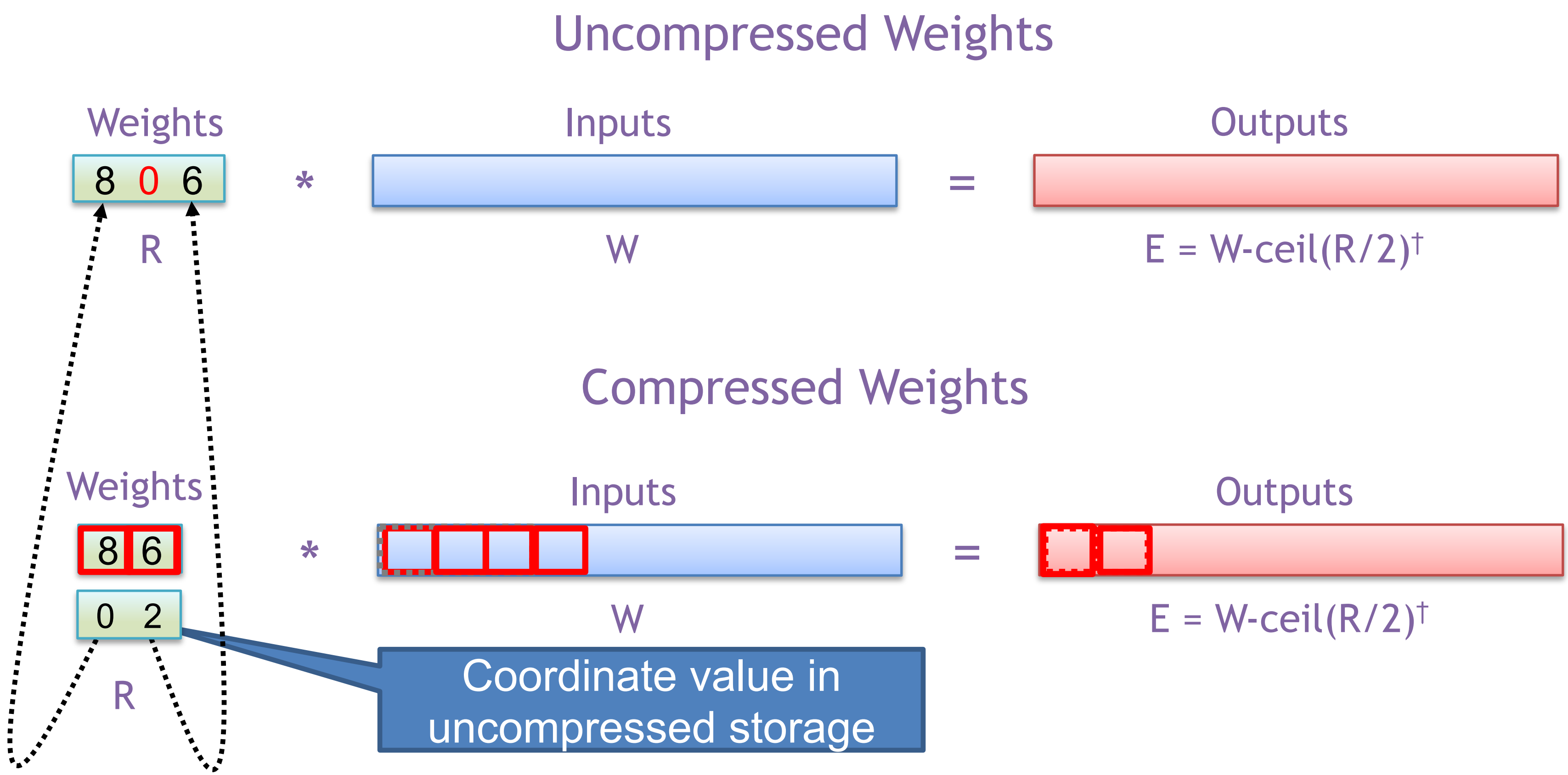
Compressed Storage of Weights

Uncompressed Weights

Compressed Weights

† Assuming: ‘valid’ style convolution

Compressed Storage of Weights



† Assuming: ‘valid’ style convolution

Compressed Weights 1-D Convolution

Weight values (wv)

Weights

8	6
0	2

R

* Inputs (W) = Outputs ($E = W\text{-ceil}(R/2)^{\dagger}$)

Weight indices (wi)

Big enough to hold worst case size

```
int i[W];                # Input activations
int wv[R], wc[R];        # Compressed filter weights
int o[E];                # Output activations

for (e = 0; e < E; e++) {
       for (r = 0; r < R; r++) {
          break if !wi[r].valid;
          o[e] += i[e+wc[r]] * wv[r];
}}
```

or pre-computed length

Table lookup of weight index

No

Direct access into compressed buffer

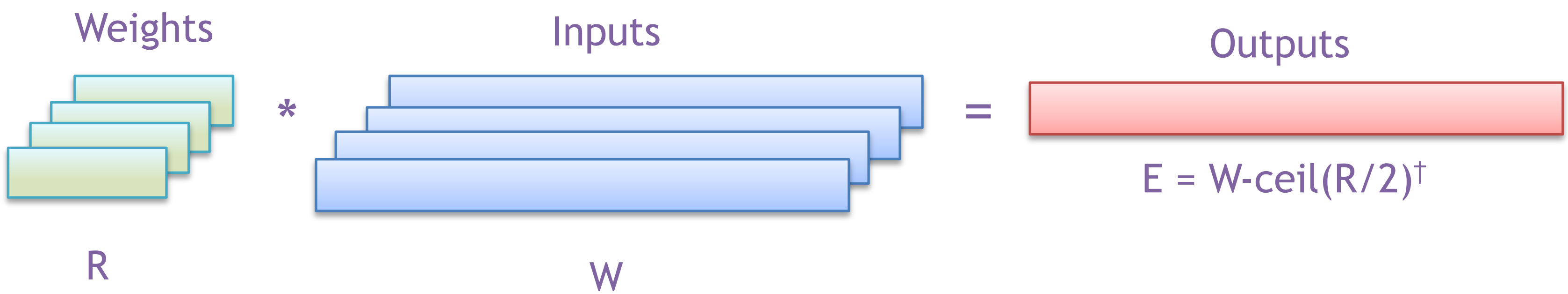
To Extend to Other Dimensions of DNN

- Need to add loop nests for:
  - 2-D input activations and filters
  - Multiple input channels
  - Multiple output channels

- Add parallelism…

Compressed Inputs

Multi-Input Channel 1-D Convolution

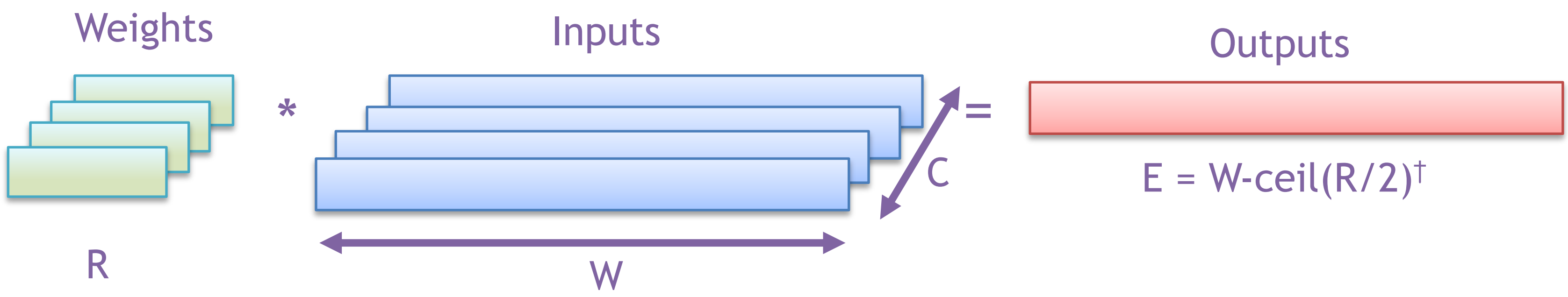



```
int i[C][W];            # Multi-input channel activations
int w[C][R];            # Filter weights
int o[E];               # Output activations

for (w = 0; w < W; w++) {
  for (r = 0; r < R; r++) {
    parallel-for (c = 0; c < C; c++) {
      o[w-r] += i[c][w]*w[c][r];
}}}
```

Note opportunity for spatial sum

† Assuming: ‘valid’ style convolution

Multi-Input Channel 1-D Convolution



Input values and coordinates are now 2D

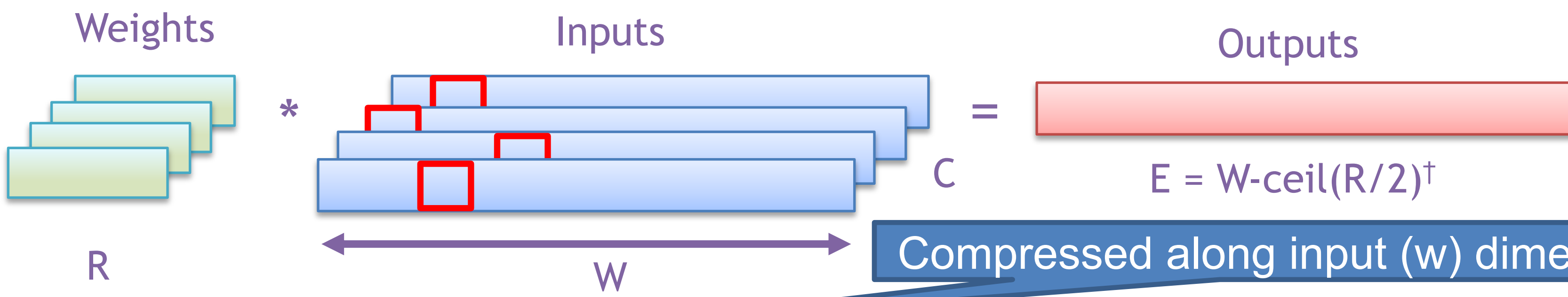
```
int iv[C][W], ic[C][W];   # Compressed input activations
int w[C][R];              # Filter weights
int o[E];                 # Output activations
```

Should we compress along C or W dimension?

Let's see

† Assuming: 'valid' style convolution

Compressed Sparse W-dimension



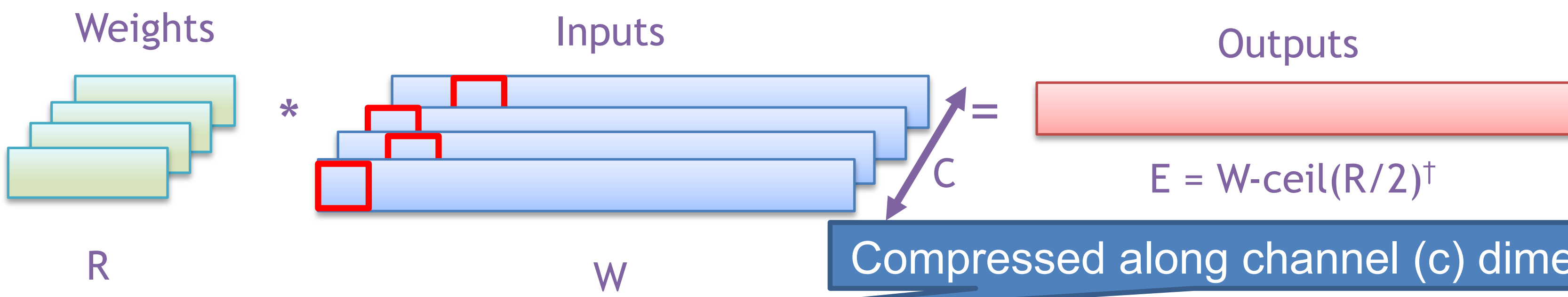
Compressed along input (w) dimension

```
int iv[C][W], icw[C][W];  # Compressed input activations
int w[C][R];              # Filter weights
int o[E];                 # Output activations

for (w = 0; w < W; w++) {
  for (r = 0; r < R; r++) {
    parallel-for (c = 0; c < C; c++) {
          break if !icw[c][w].valid;
          o[icw[c][w]-r] += iv[c][w]*w[c][r];
}}}
```

Running parallel COMPRESSed w's

The variation of these index values with different c's will prevent synchronized spatial sum

Compressed Sparse C-dimension



Compressed along channel (c) dimension

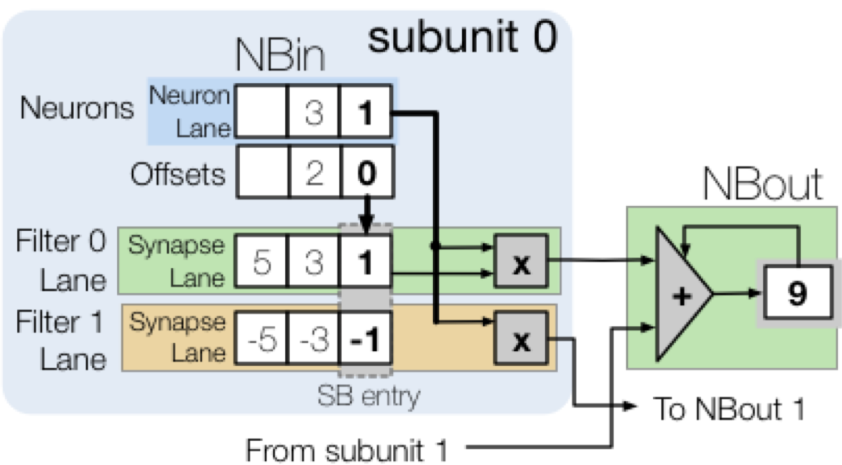
```
int iv[C][W], icc[C][W];  # Compressed input activations
int w[C][R];              # Filter weights
int o[E];                 # Output activations

for (r = 0; r < R; r++) {
  for (w = 0; w < W; w++) {
    parallel-for (c = 0; c < C; c++) {
          break if !icc[c][w].valid;
          o[w-r] += iv[c][w]*w[icc[c][w]][r];
}}}
```

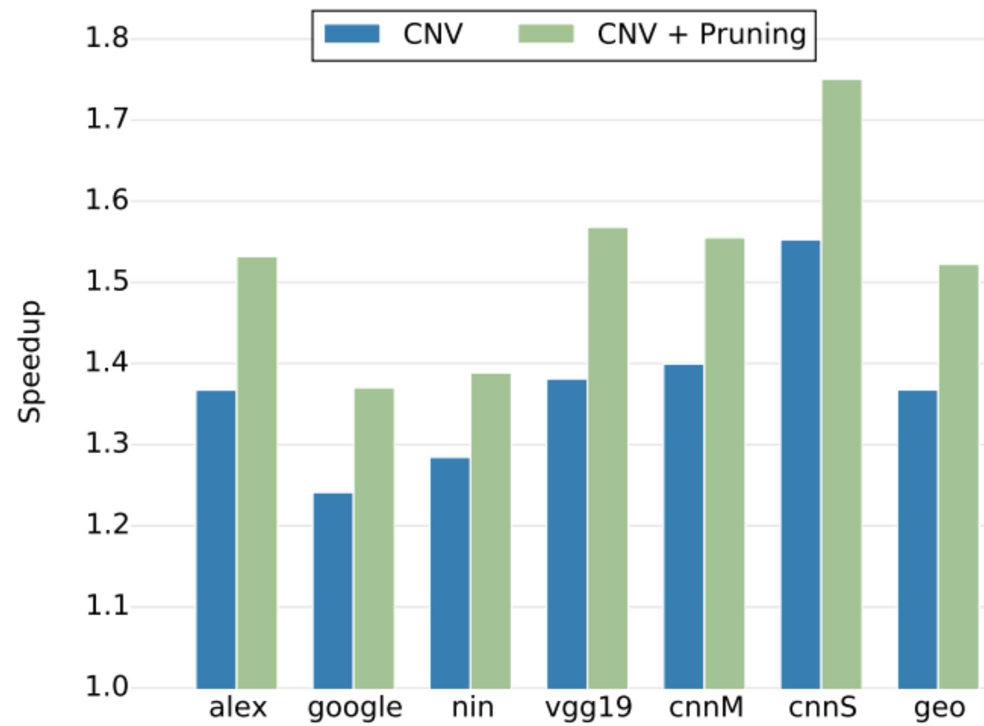
Running parallel COMPRESSed c's

Note we now have a synchronized spatial sum

Cnvlutin

- Process Convolution Layers
- Built on top of DaDianNao (4.49% area overhead)
- Speed up of 1.37x (1.52x with activation pruning)





[Albericio et al., ISCA 2016]

Compressing Inputs + Weights

Output Stationary – Sparse W&I

Weights

8	0	6

R

*

Inputs

4	0	0	0	3	0	0	8	0	2

W

=

Outputs

1/3	0/3	1/3	0/3

E = W-ceil(R/2)†

```
int i[W];                 # Input activations
int w[R];                 # Filter weights
int o[E];                 # Output activations

for (e = 0; e < E; e++) {
       parallel-for (r = 0; r < R; r++) {
          next if w[r] == 0;
          next if i[e+r] == 0;
          o[e] += i[e+r] * w[r];
}}
```

How often is work done in inner loop? Not very much!

Flattened Inputs & Weights

```
int i[C][W*H];          # Flattened input activations
int w[C][M*R*S];        # Flattened filter weights
int o[M][E][F];         # Output activations

for mrs2 = [0..MRS2) {
  for c2 = [0..C) {
     for wh1 = [0..WH1) {
       for mrs1 = [0..MRS1) {
        parallel-for wh0 = [0..WH0) x
                     mrs0 = [0..MRS0) {
          m = Mcoord(mrs2, mrs1, mrs0);
          e = Wcoord(wh1,wh0)-Rcoord(mrs2, mrs1, mrs0);
          f = Hcoord(wh1,wh0)-Scoord(mrs2, mrs1, mrs0);
          o[m][e][f] += i[c2][wh1*WH0+wh0]
                      * w[c2][mrs2*MRS1*MRS0+mrs1*MRS0+mrs0];
}}}}
```

At inner loop inputs are stationary across steps mrs1

Any opportunity for spatial sum?

No

Sparse CNN (SCNN)

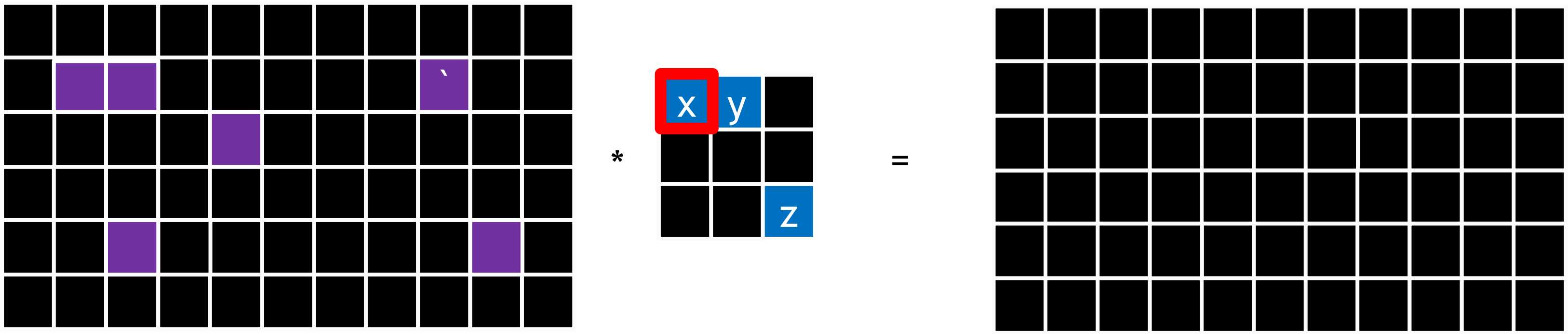
- Architecture to exploit sparsity

Intuition behind SCNN

Forget the sliding windows based convolution

Observation

Each non-zero activation must be multiplied by each non-zero weight

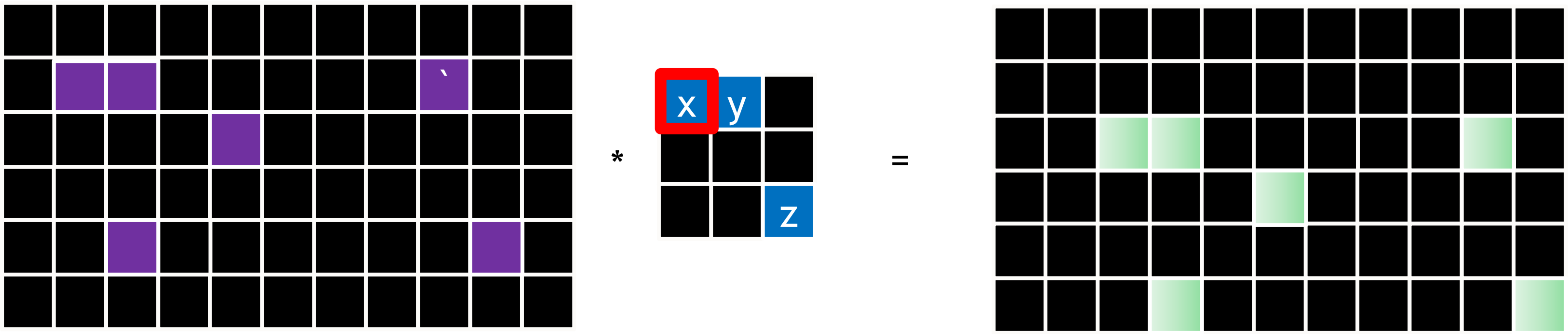


Intuition behind SCNN

Forget the sliding windows based convolution

Observation

Each non-zero activation must be multiplied by each non-zero weight

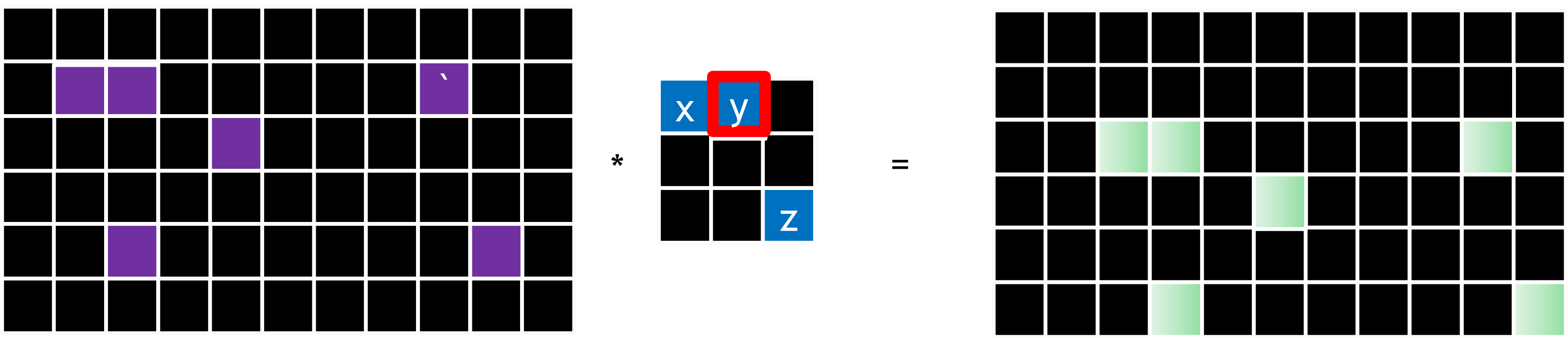


Intuition behind SCNN

Forget the sliding windows based convolution

<u>Observation</u>

Each non-zero activation must be multiplied by each non-zero weight

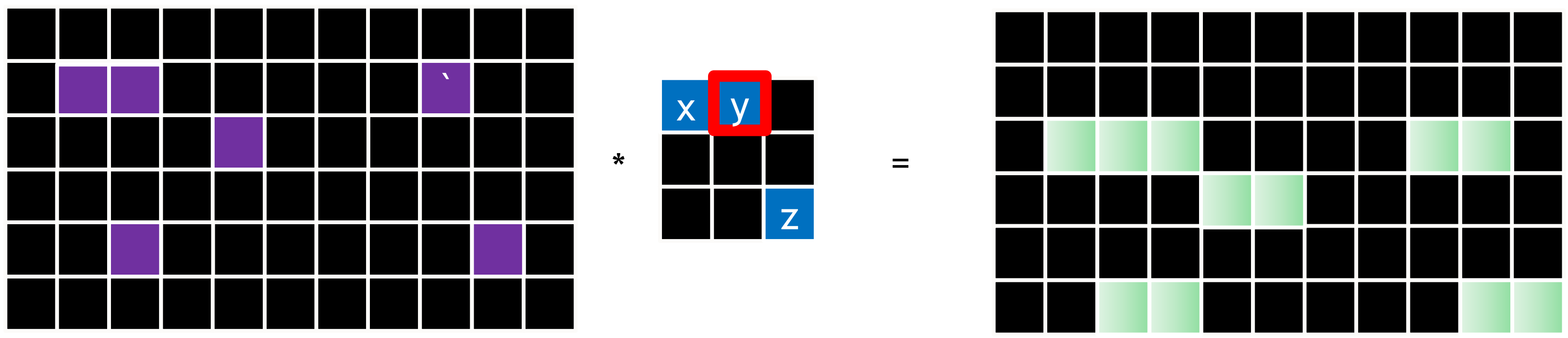


Intuition behind SCNN

Forget the sliding windows based convolution

Observation

Each non-zero activation must be multiplied by each non-zero weight

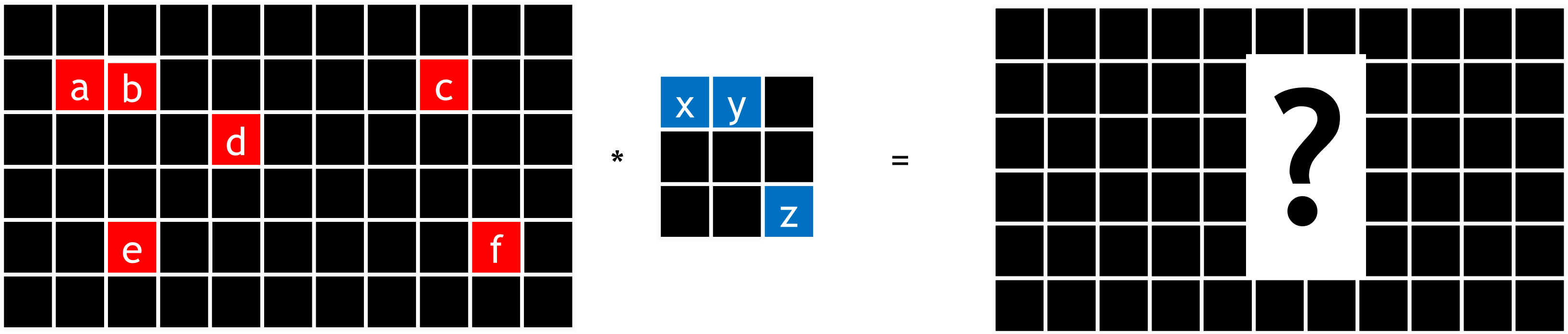


Intuition behind SCNN

Forget the sliding windows based convolution

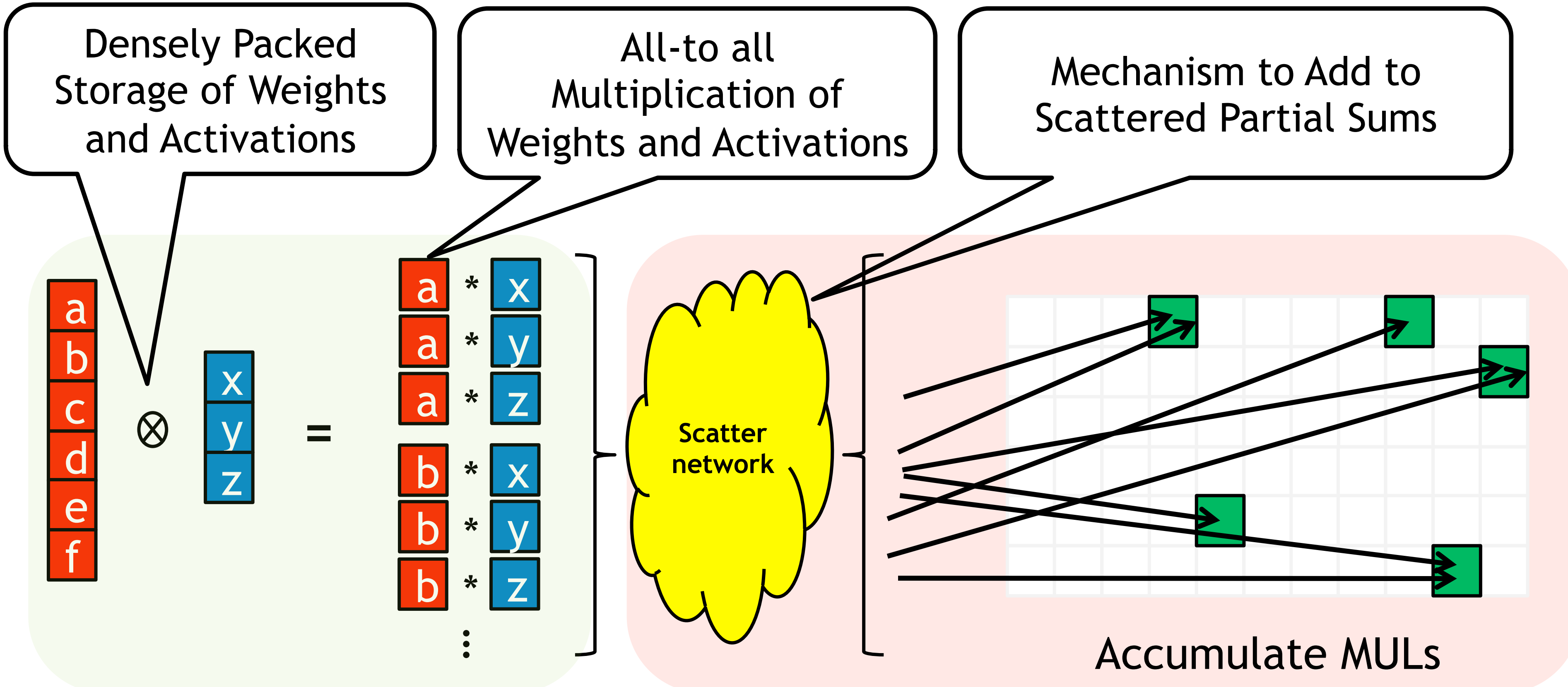
Observation

Each non-zero activation must be multiplied by each non-zero weight



Sparse CNN (SCNN)

Supports Convolutional Layers



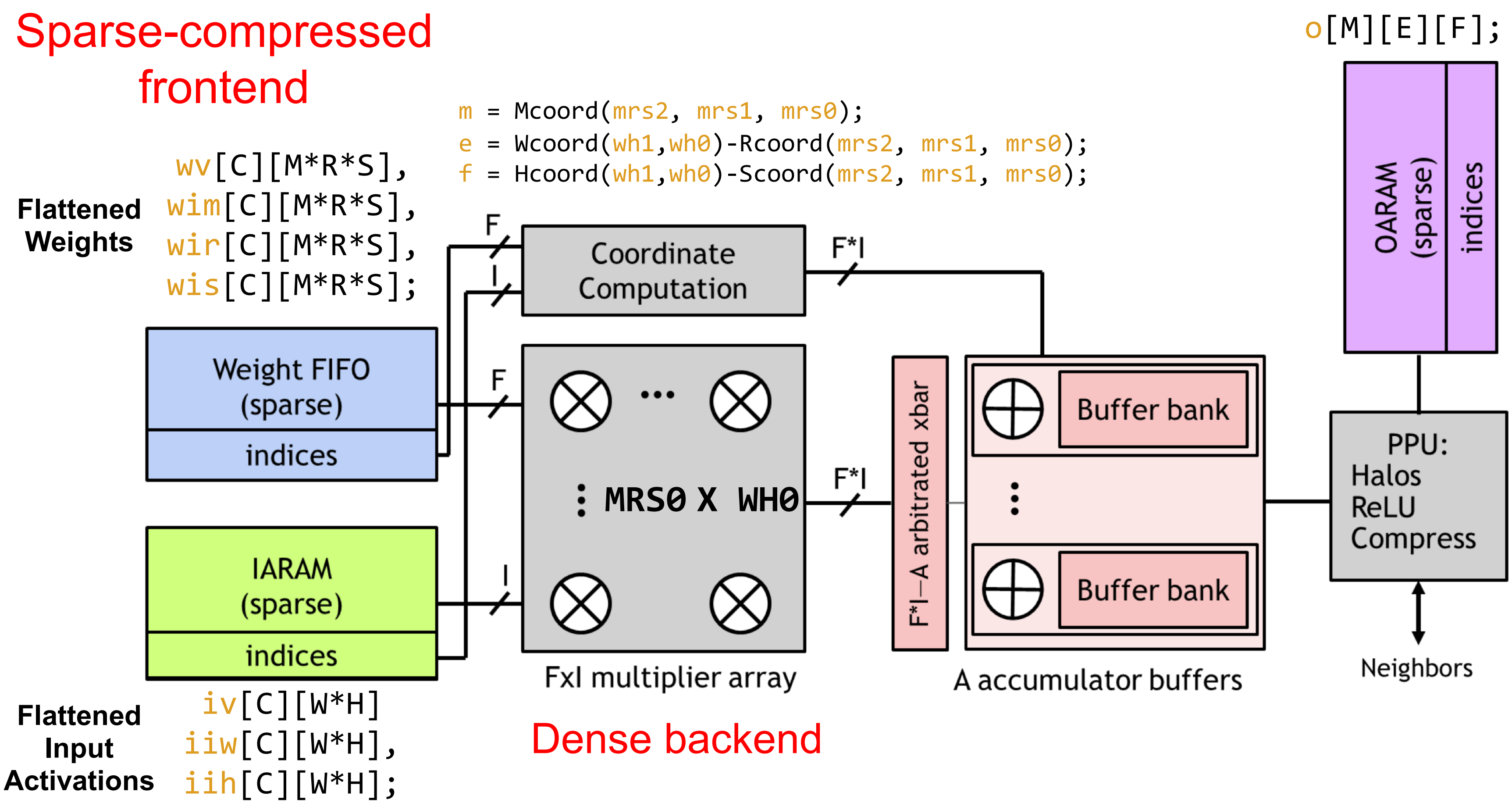
PE frontend

PE backend

Input Stationary Dataflow

[Parashar et al., ISCA 2017]

SCNN PE microarchitecture

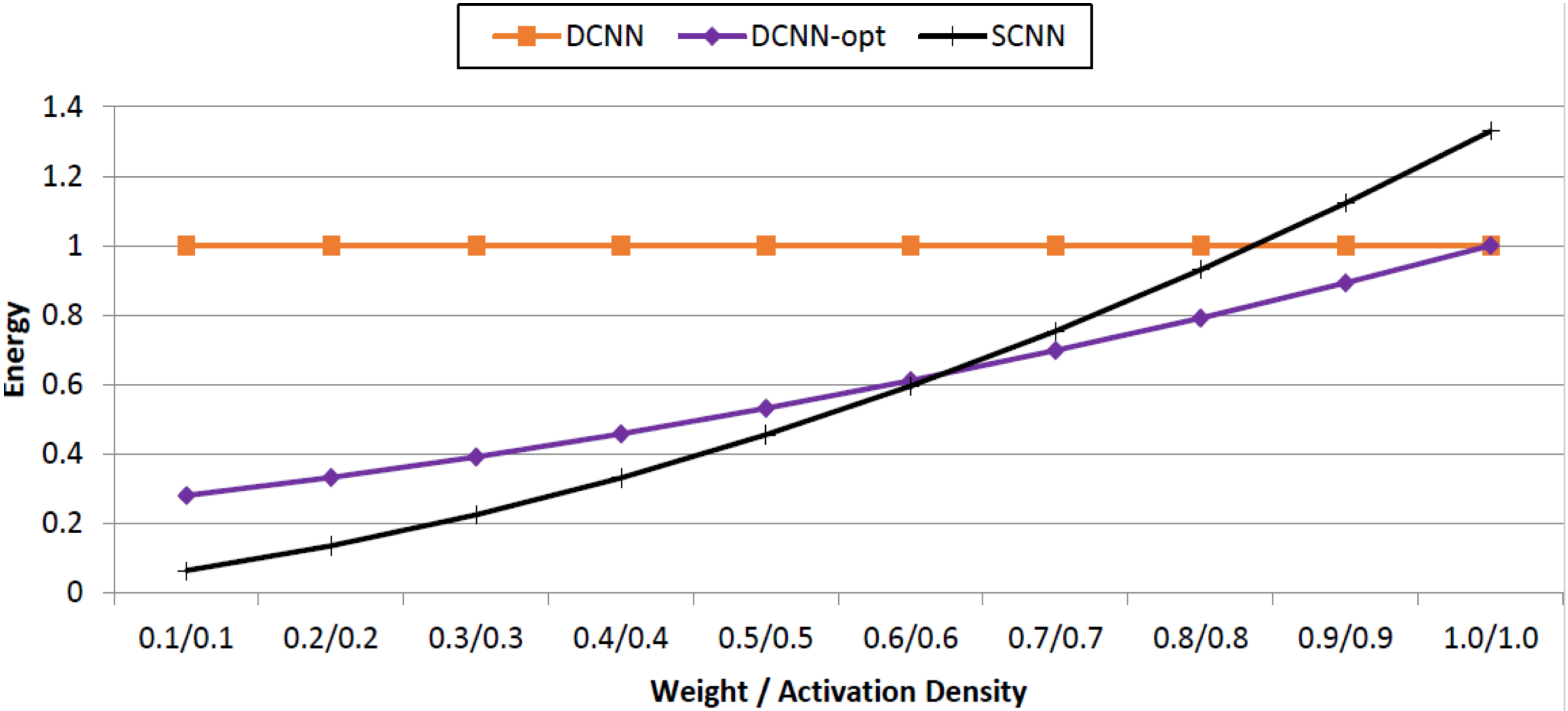


[Parashar et al., ISCA 2017]

Flattened Inputs & Weights

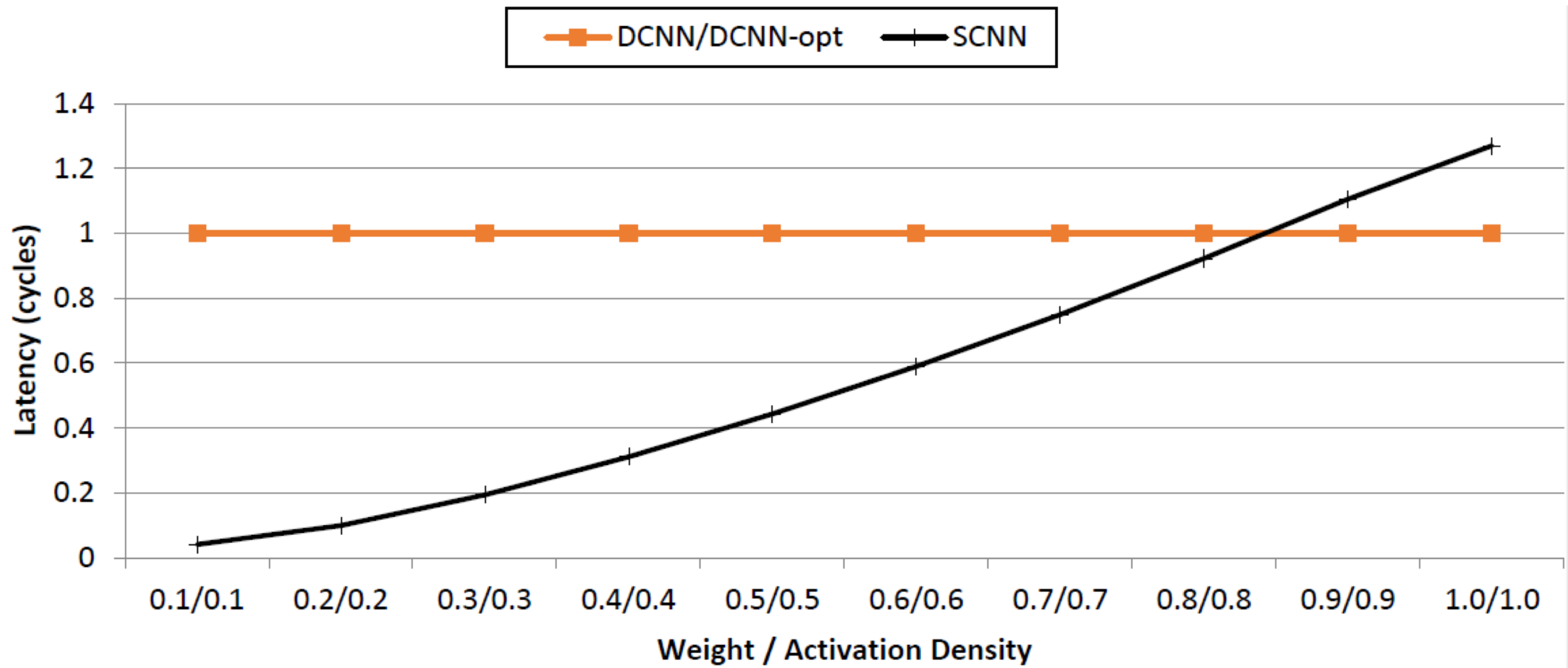
```
int iv[C][W*H] iiw[C][W*H], iih[C][W*H];
int wv[C][M*R*S], wim[C][M*R*S], wir[C][M*R*S], wis[C][M*R*S];
int o[M][E][F];

for mrs2 = [0..MRS2) {
  for c2 = [0..C) {
    for wh1 = [0..WH1) {
      for mrs1 = [0..MRS1) {
        parallel-for wh0 = [0..WH0) x
                     mrs0 = [0..MRS0) {
          break if !ii[c2][mrs2*MRS1*MRS0+mrs1*MRS0+mrs0].v;
          break if !wi[c2][wh1*WH0+wh0].v;
          m = Mcoord(mrs2, mrs1, mrs0);
          e = Wcoord(wh1,wh0)-Rcoord(mrs2, mrs1, mrs0);
          f = Hcoord(wh1,wh0)-Scoord(mrs2, mrs1, mrs0);
          o[m][e][f] += i[c2][wh1*WH0+wh0]
                      * w[c2][mrs2*MRS1*MRS0+mrs1*MRS0+mrs0];
}}}}
```

SCNN Energy Versus Density



[Parashar et al., ISCA 2017]

SCNN Latency Versus Density



[Parashar et al., ISCA 2017]

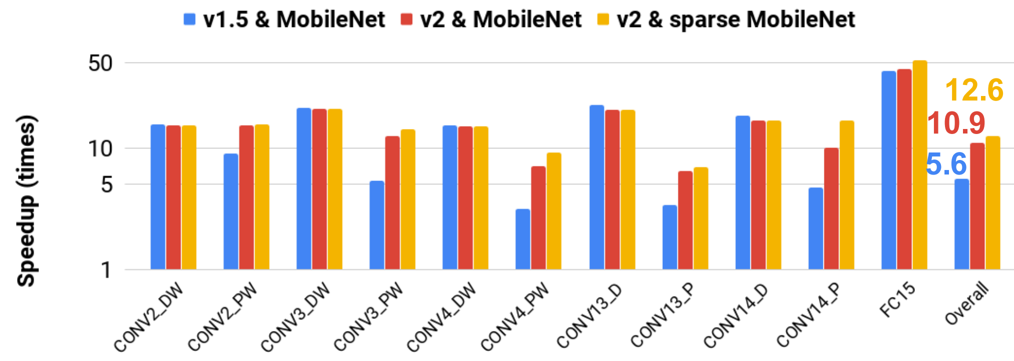
Eyeriss – V2

- Architecture to accommodate variety sparsity

Eyeriss v2: Balancing Flexibility and Efficiency

Efficiently supports

- Wide range of filter shapes
  - Large and Compact
- Different Layers
  - CONV, FC, depth wise, etc.
- **Wide range of sparsity**
  - **Dense and Sparse**
- Scalable architecture



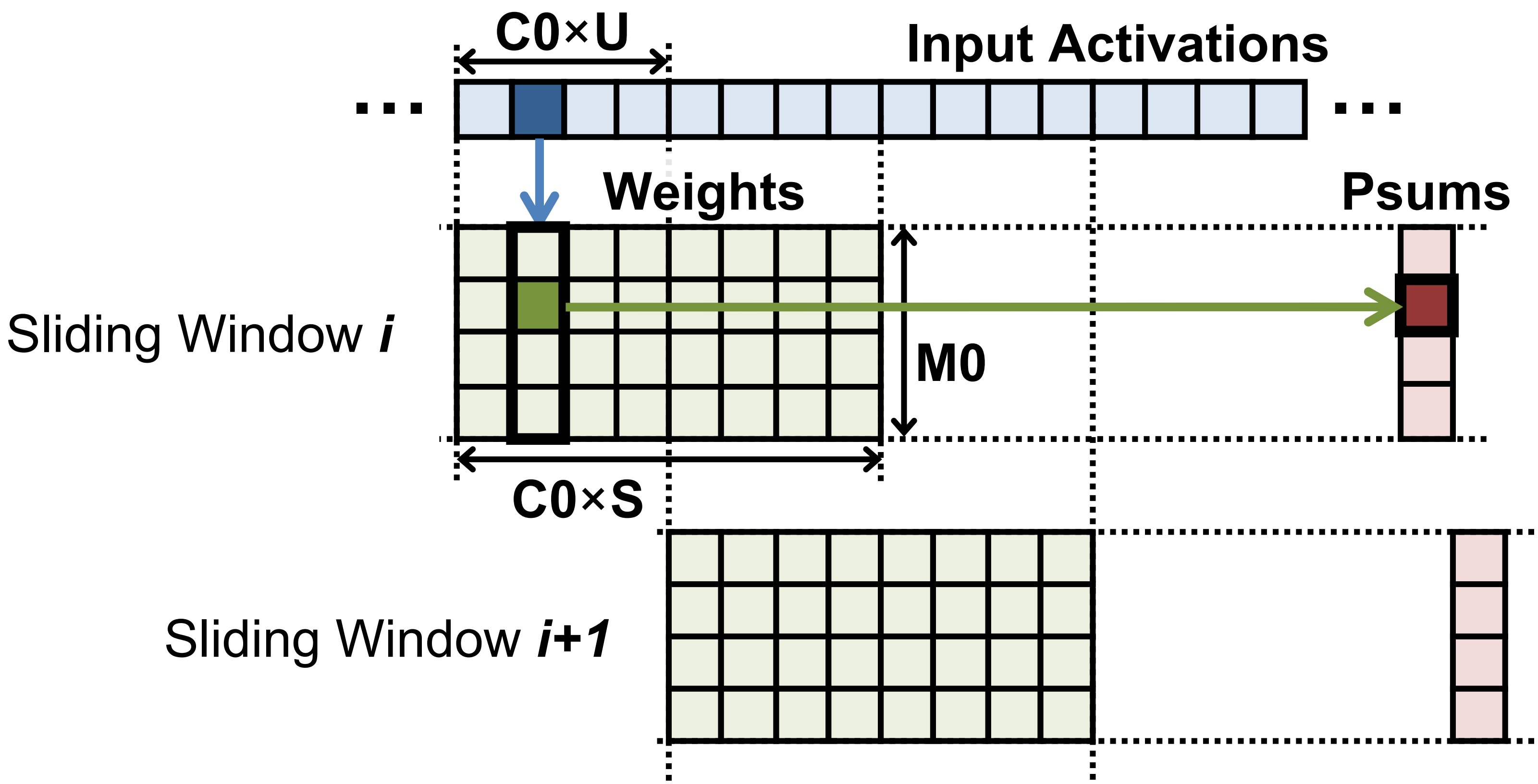
Speed up over Eyeriss v1 scales with number of PEs

# of PEs	256	1024	16384
AlexNet	17.9x	71.5x	1086.7x
GoogLeNet	10.4x	37.8x	448.8x
MobileNet	15.7x	57.9x	873.0x

[Chen et al., JETCAS 2019]

Over an order of magnitude faster and more energy efficient than Eyeriss v1

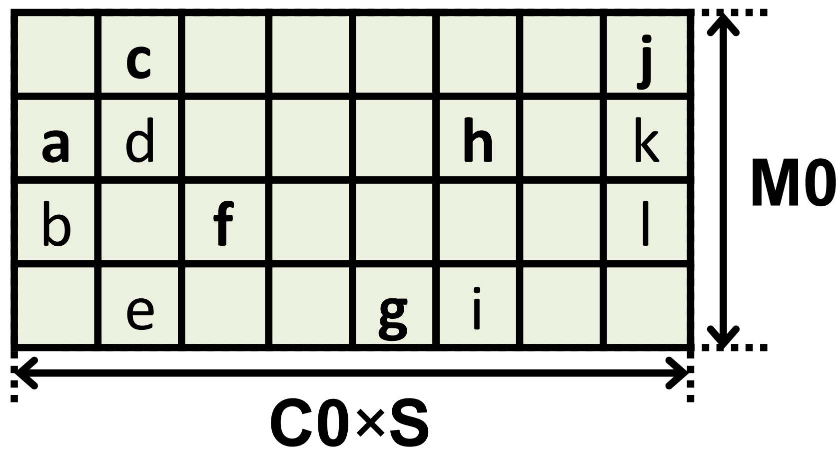
Eyeriss v2: Processing In PE



- **M0** : # of output channels processed in a PE
- **C0** : # of input channels processed in a PE
- **S** : filter width
- **U** : stride

Eyeriss v2: Compressed Data Format

Weight Matrix



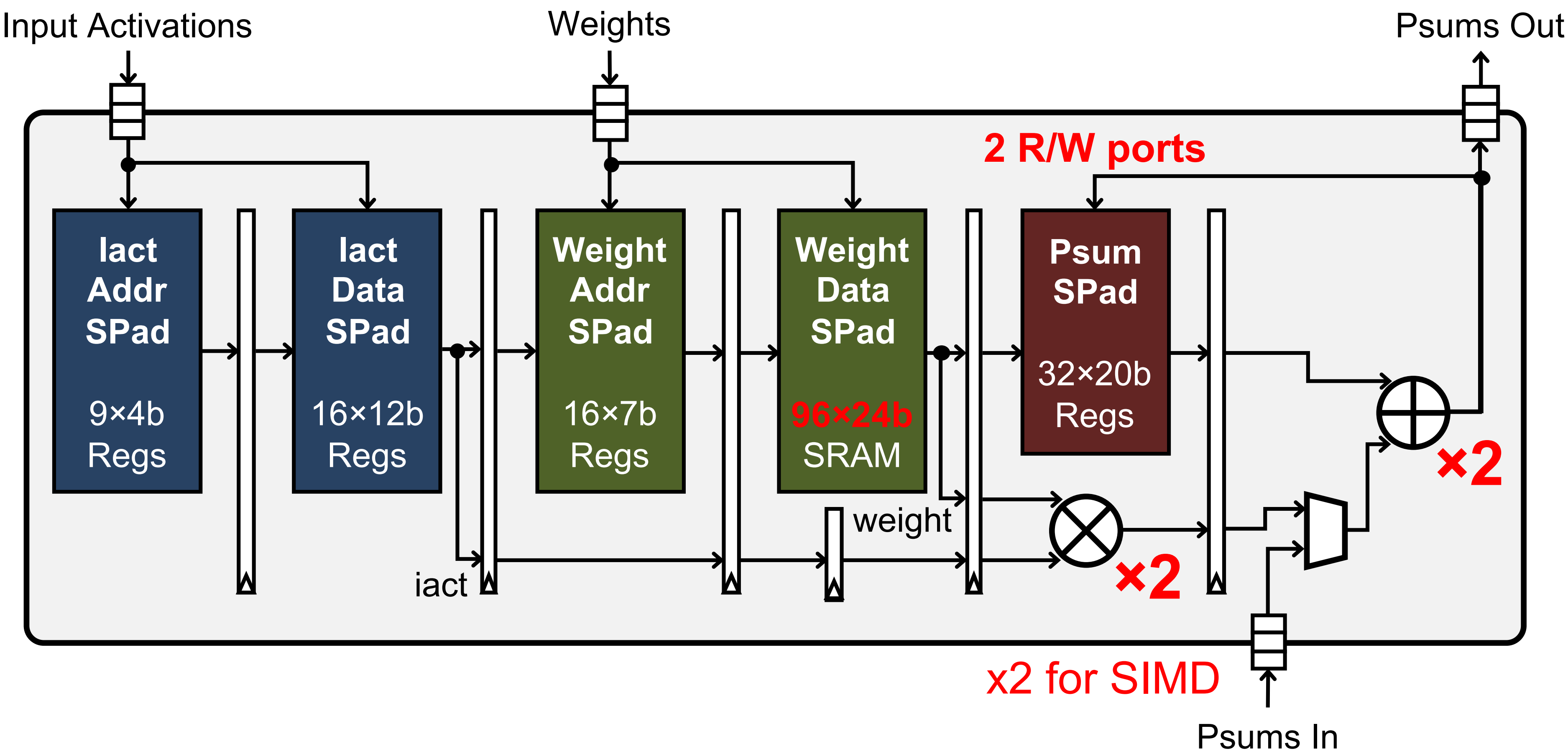
CSC Compressed Data:

data vector: {**a**, b, **c**, d, e, **f**, **g**, **h**, i, **j**, k, l}

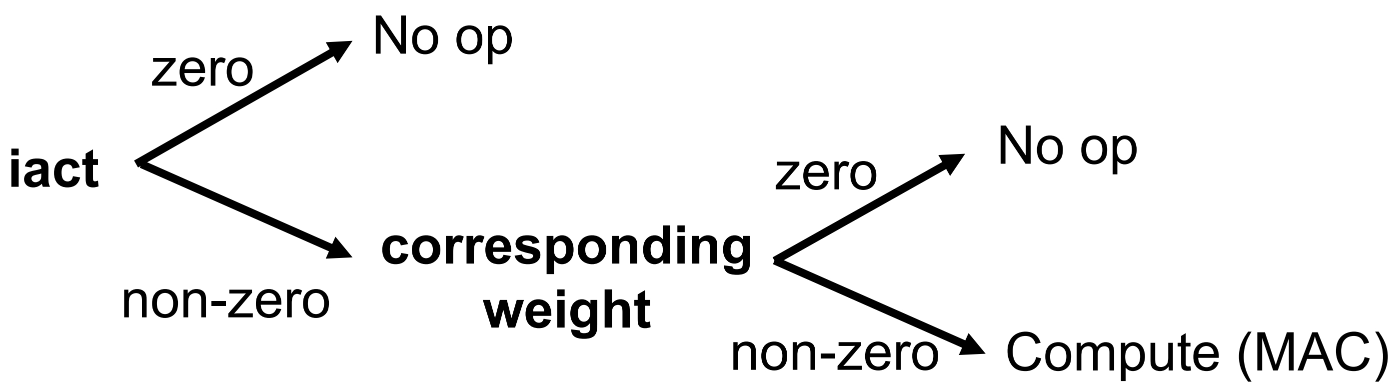
count vector: {1, 0, 0, 0, 1, 2, 3, 1, 1, 0, 0, 0}

address vector: {0, 2, 5, 6, 6, 7, 9, 9, 12}

Eyeriss v2: PE Architecture



Decision Tree in Eyeriss v2 PE



- **If the iact is zero**, the CSC format will ensure that it is not read from the spad and therefore no cycles are wasted.
- **If the iact is not zero**, its value will be fetched from the iact data SPad and passed to the next pipeline stage.
  - **If there are non-zero weights corresponding to the non-zero iacts,** they will be passed down the pipeline for computation. The zero weights will be skipped since the weights are also encoded with the CSC format.
  - **If there are no non-zero weights corresponding to the non-zero iacts,** the non-zero iacts will not be further passed down in the pipeline.

Summary

- Processing Irregular (Gather-Scatter)
  - If weights and inputs compressed to dense (gather); output scatter
  - If weights and inputs uncompressed sparse (scatter); output gather
- Overhead (must not exceed benefits of sparsity)
  - Storage of location information for compressed data
  - Logic for checking if either inputs are zero
- Underutilization
  - Number of parallel cores (tiling) → maximize parallelism, but minimize underutilization
  - Flatten to 1-D avoid fragmentation from limits of each dimension
- Workload Imbalance
- **Lots of challenges in sparse deep neural network acceleration!**