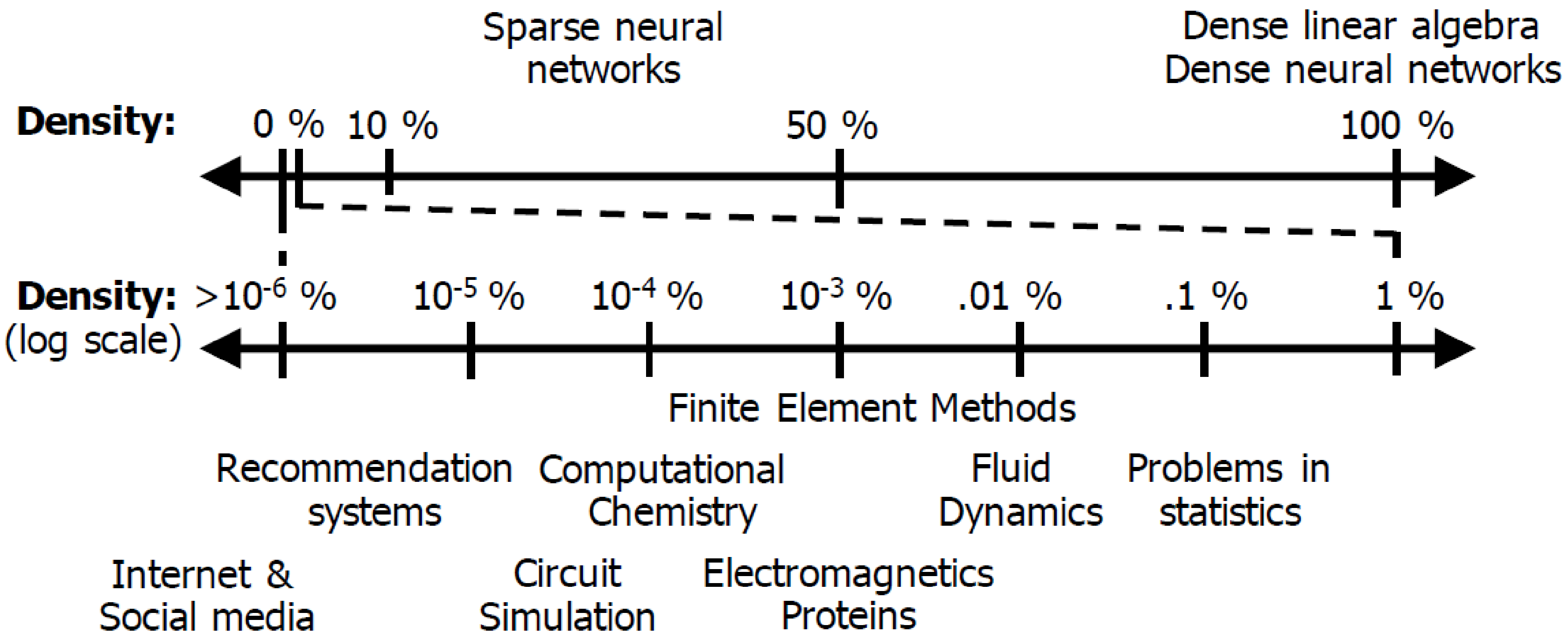
Accelerators (II)

Joel Emer

Massachusetts Institute of Technology
Electrical Engineering & Computer Science

Many problems use Sparse Tensors



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

Exploiting Sparsity

Sparse data can be compressed

Can save space and energy by avoiding manipulation of zero values

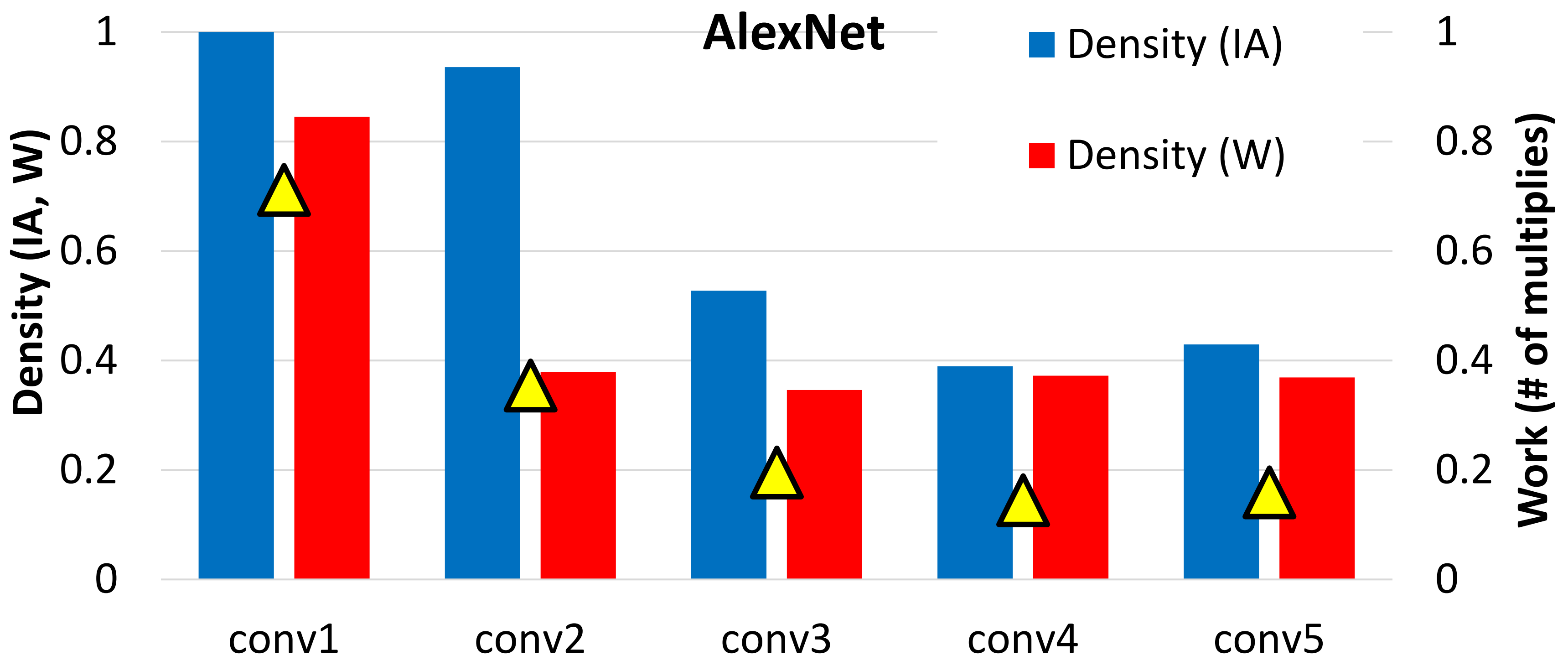
$$\boldsymbol{anything} \times \mathbf{0} = \mathbf{0}$$

$$\boldsymbol{anything} + \mathbf{0} = \boldsymbol{anything}$$

Can save time and energy by avoiding fetching unnecessary operands and avoiding **ineffectual** computations

Motivation in DNNs

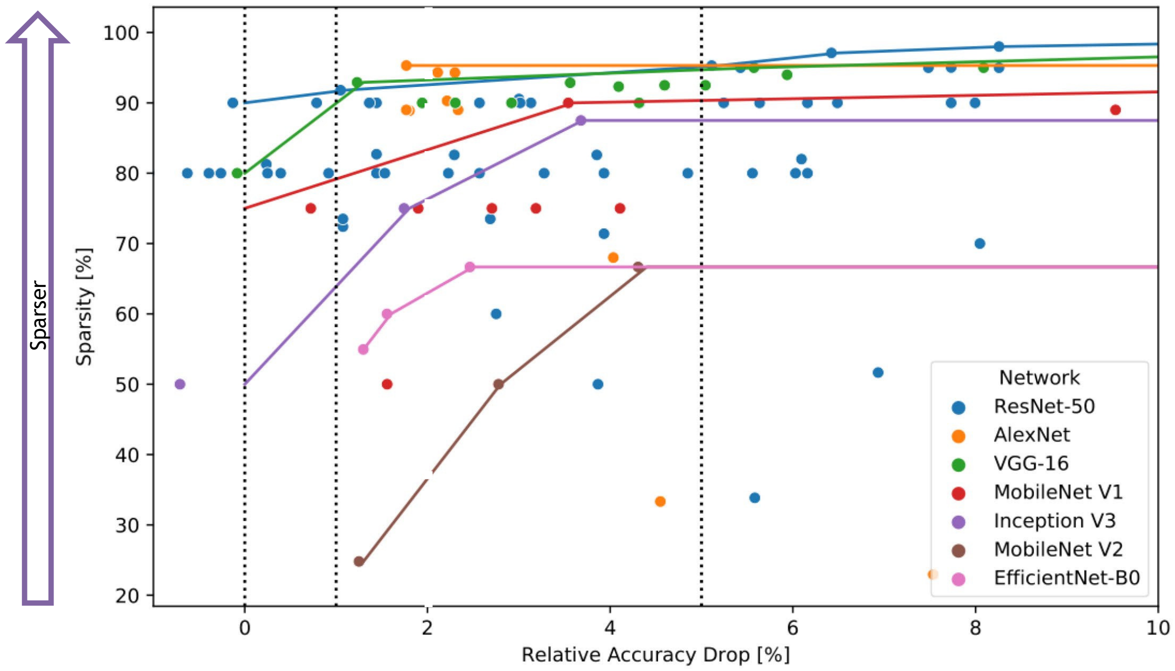
- Leverage CNN sparsity to improve energy-efficiency



SCNN, Parashar et.al., ISCA 2017

Exploitable Sparsity

Acceptable sparsity depends on target task and error tolerance



Error Tolerance

	≤0%	≤1%*	≤2%
ResNet-50	~90%	~90%	~91%
AlexNet			~93%
VGG-16	~80%	~88%	~92%
MobileNet V1	~72%	~79%	~82%
Inception V3	~50%	~62%	~73%
EfficientNet-B0			~52%
MobileNet V2			~25%

**MLPerf error tolerance*

Hoefler et al. arXiv, 2021

Hardware Sparse Acceleration Features

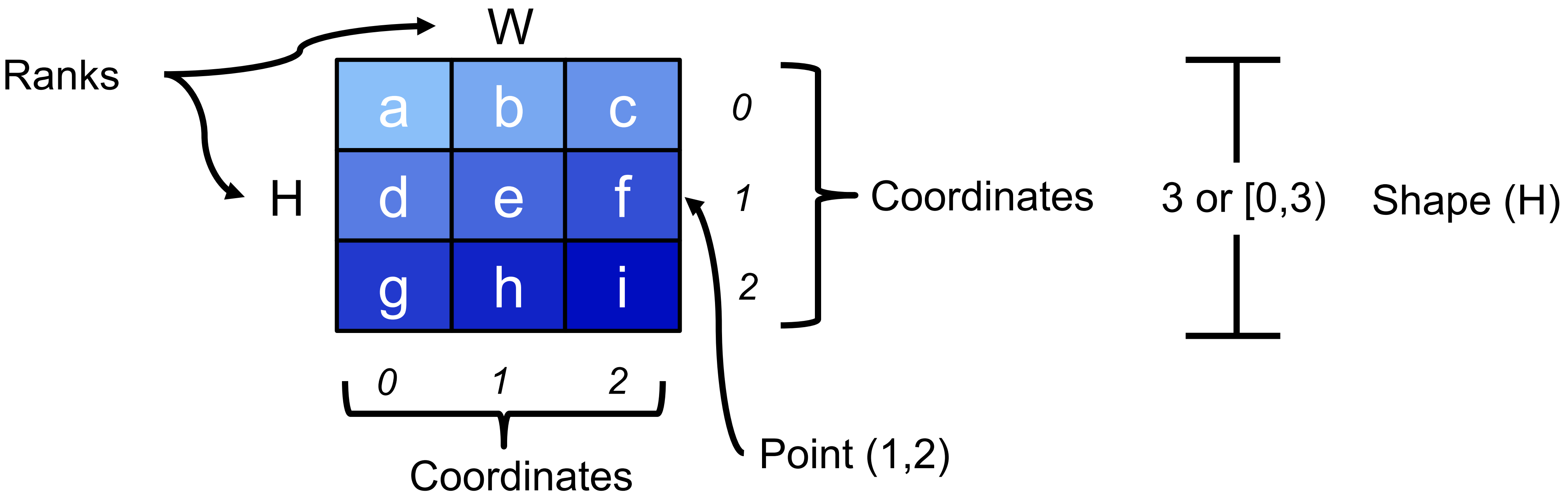
Format:
Choose tensor representations to save storage space and energy associated with zero accesses

Gating:
Explicitly eliminate ineffectual storage accesses and computes by letting the hardware unit staying idle for the cycle to save energy

Skipping:
Explicitly eliminate ineffectual storage accesses and computes by skipping the cycle to save energy and time

Separation of Concerns

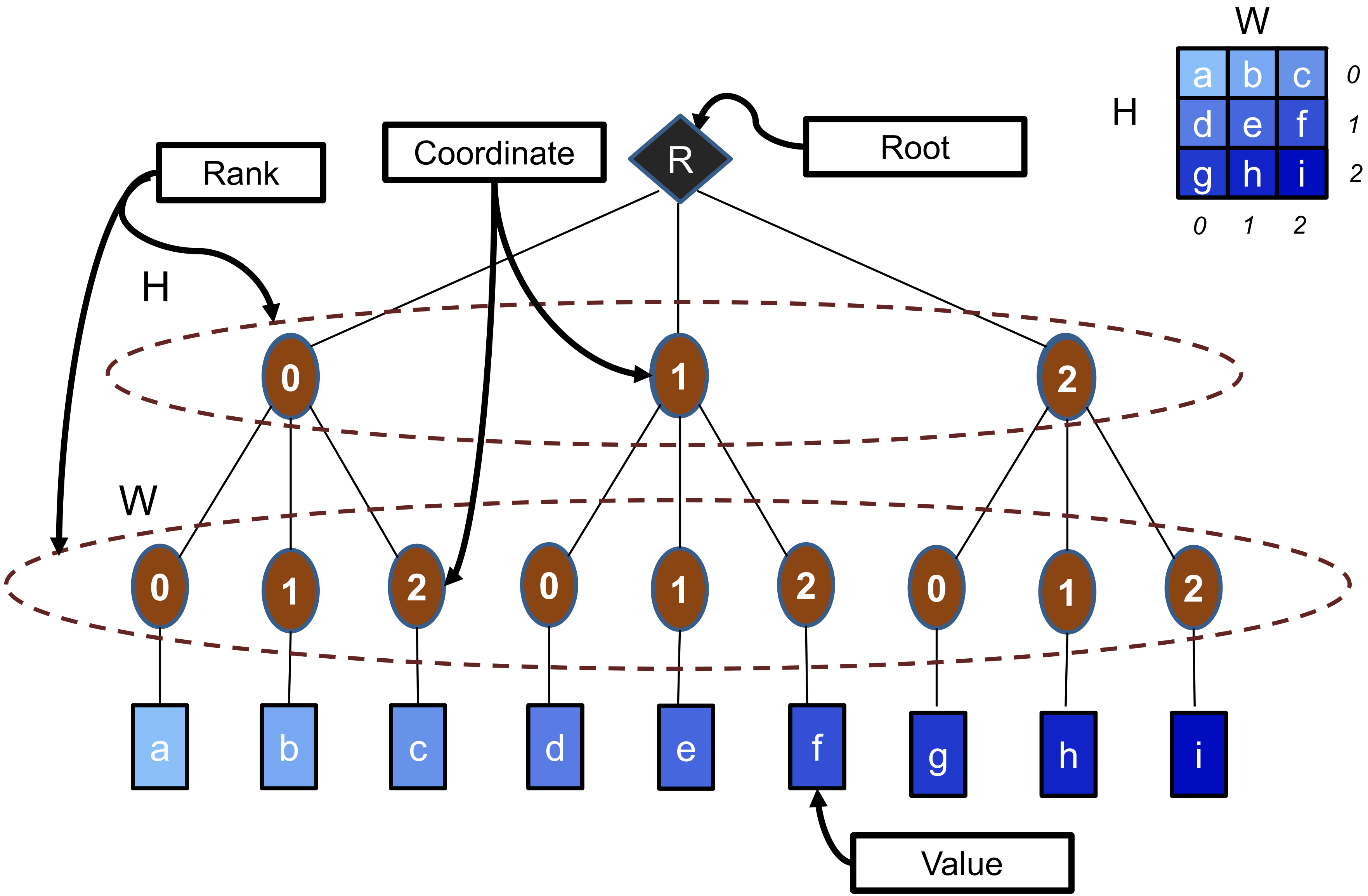
Tensor Data Terminology



- The elements of each “rank” (dimension) are identified by their “coordinates”, e.g., rank H has coordinates 0, 1, 2

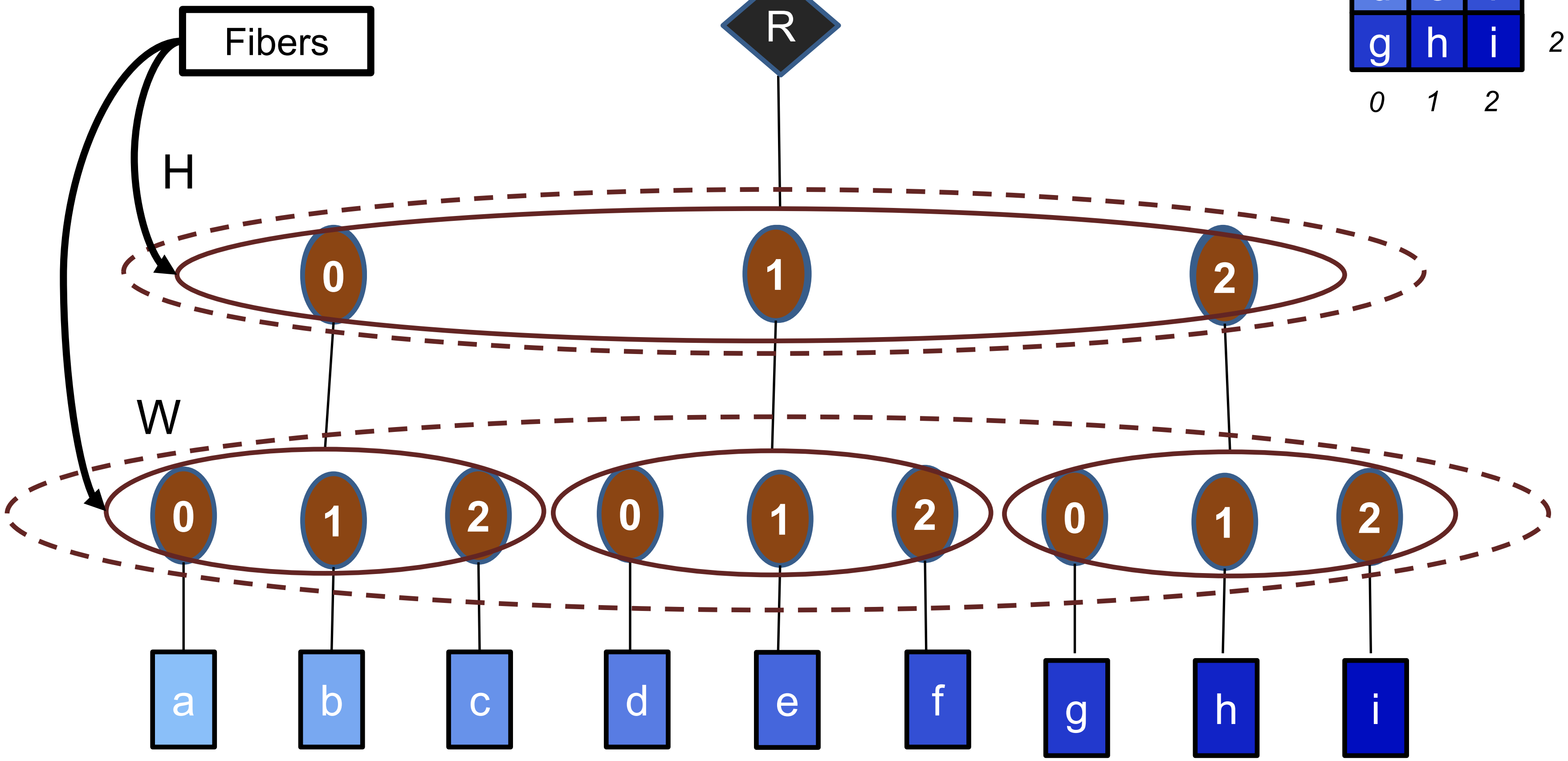
- Each element of the tensor is identified by the tuple of coordinates from each of its ranks, i.e., a “point”.
  So (1,2) -> “f”

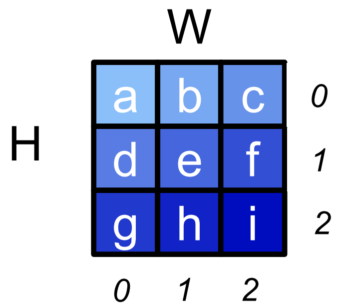
Tree-based Tensor Abstraction



Fibertree Tensor Abstraction

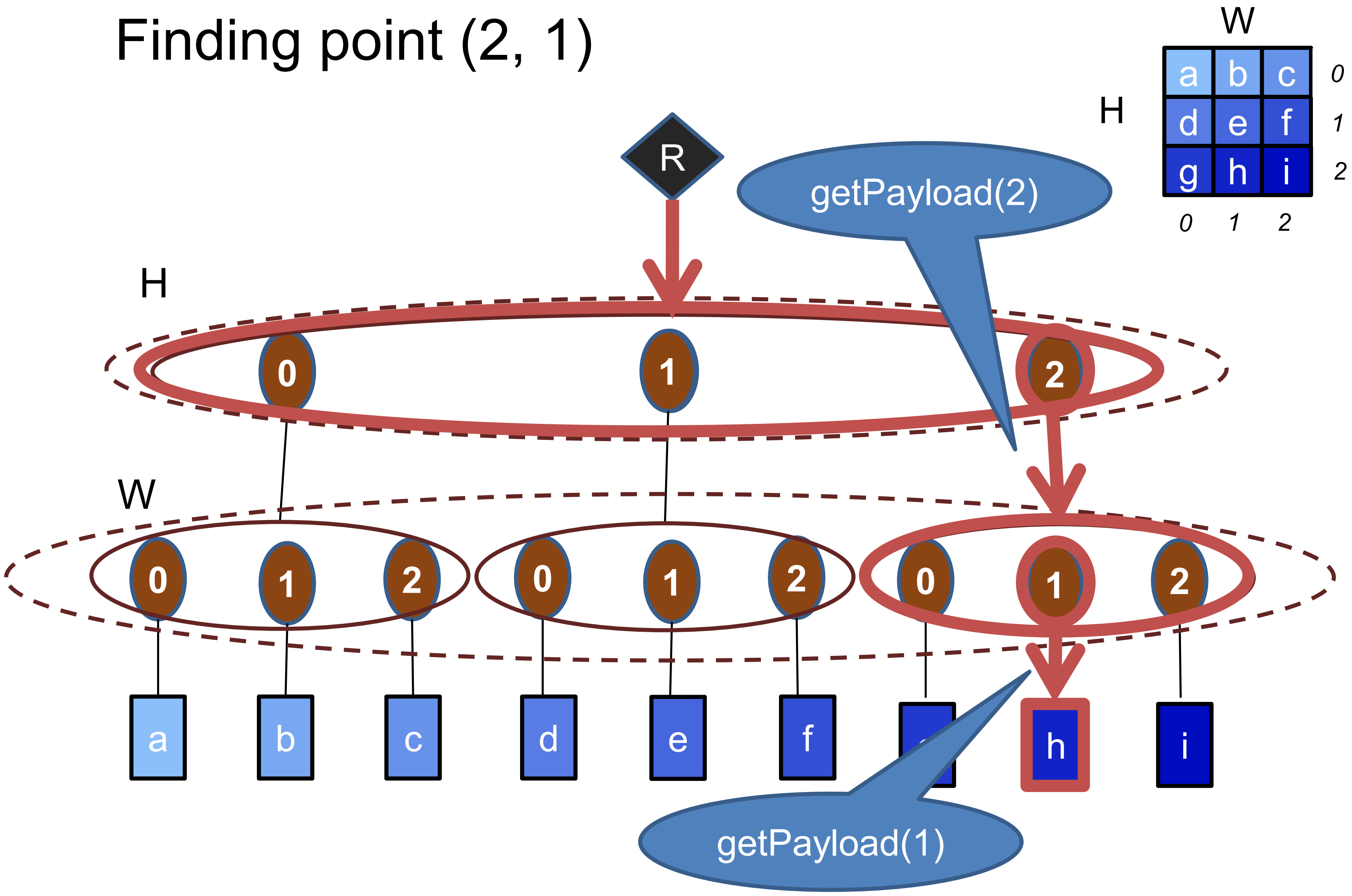
Each coordinate references a fiber





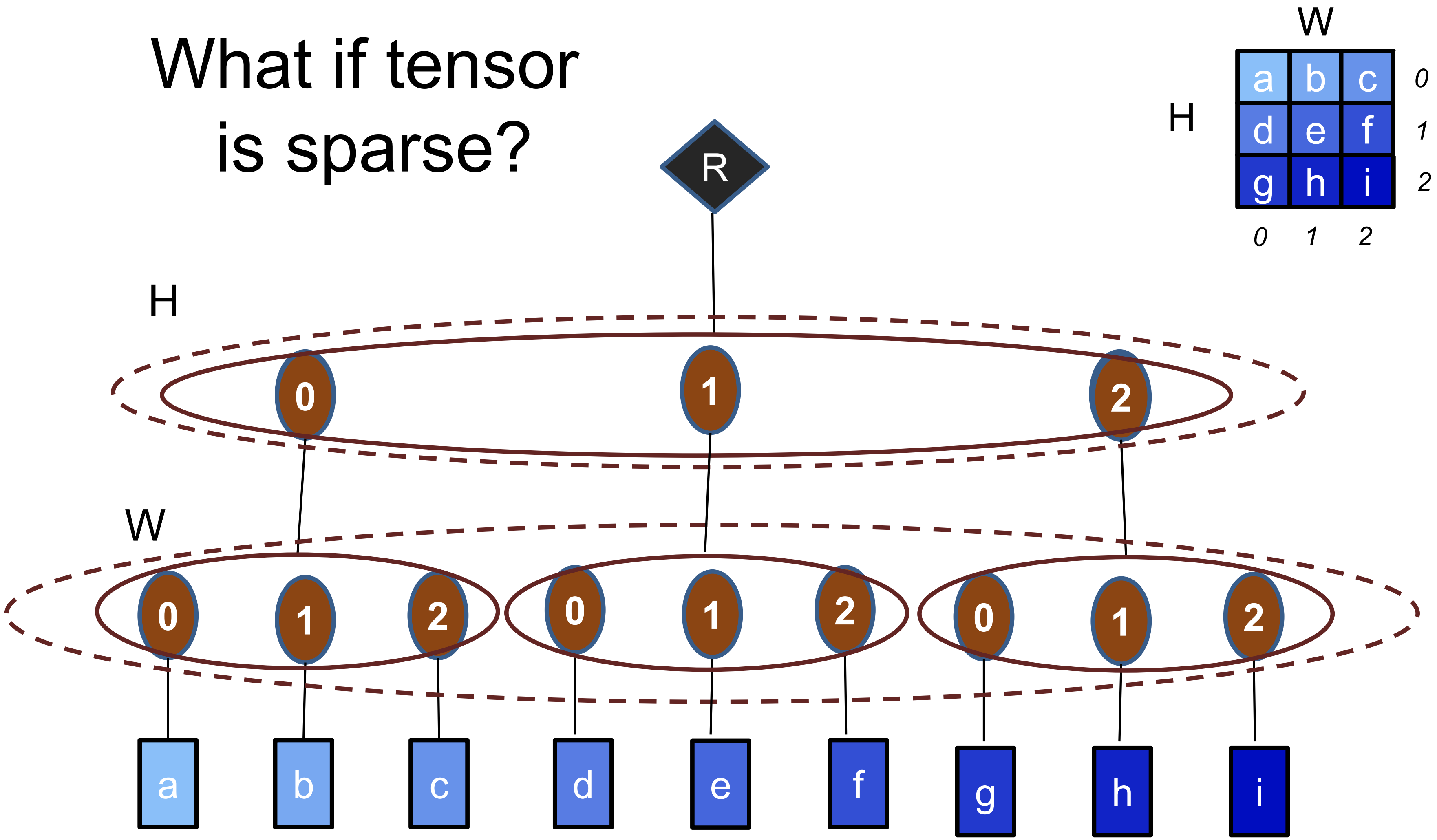
Fibertree Tensor Abstraction

Finding point (2, 1)



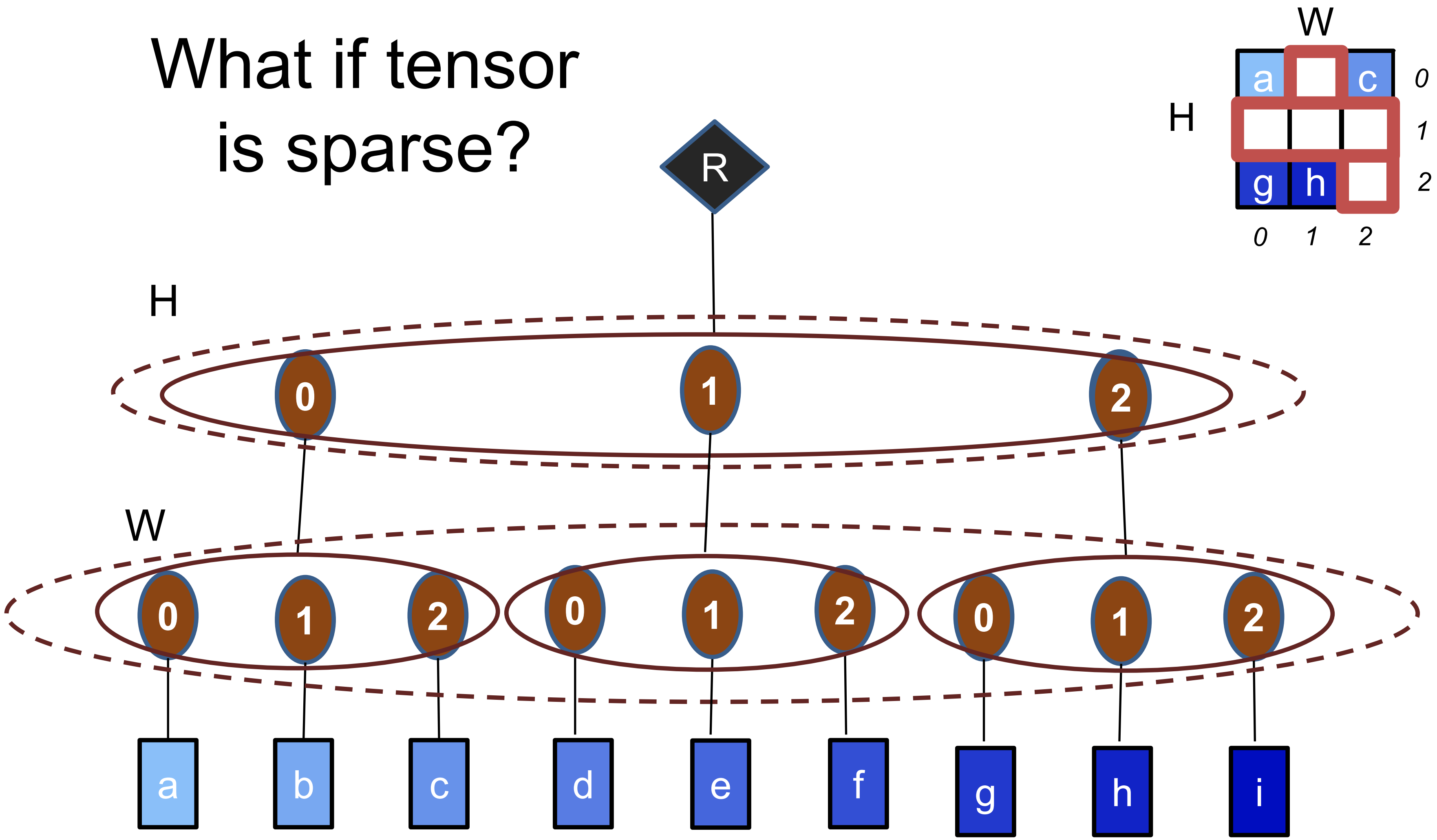
Fibertree Tensor Abstraction

What if tensor is sparse?



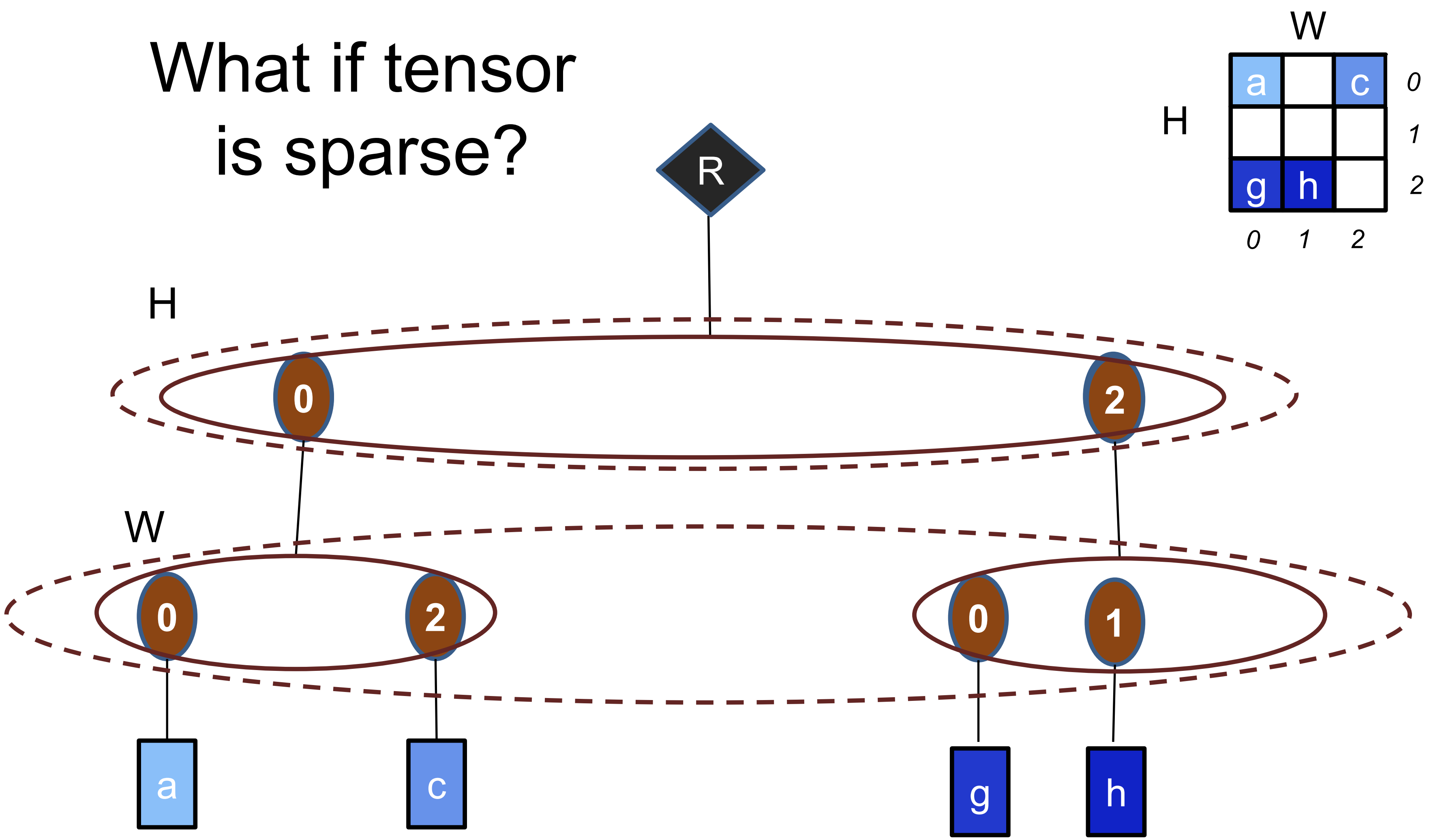
Fibertree Tensor Abstraction

What if tensor is sparse?



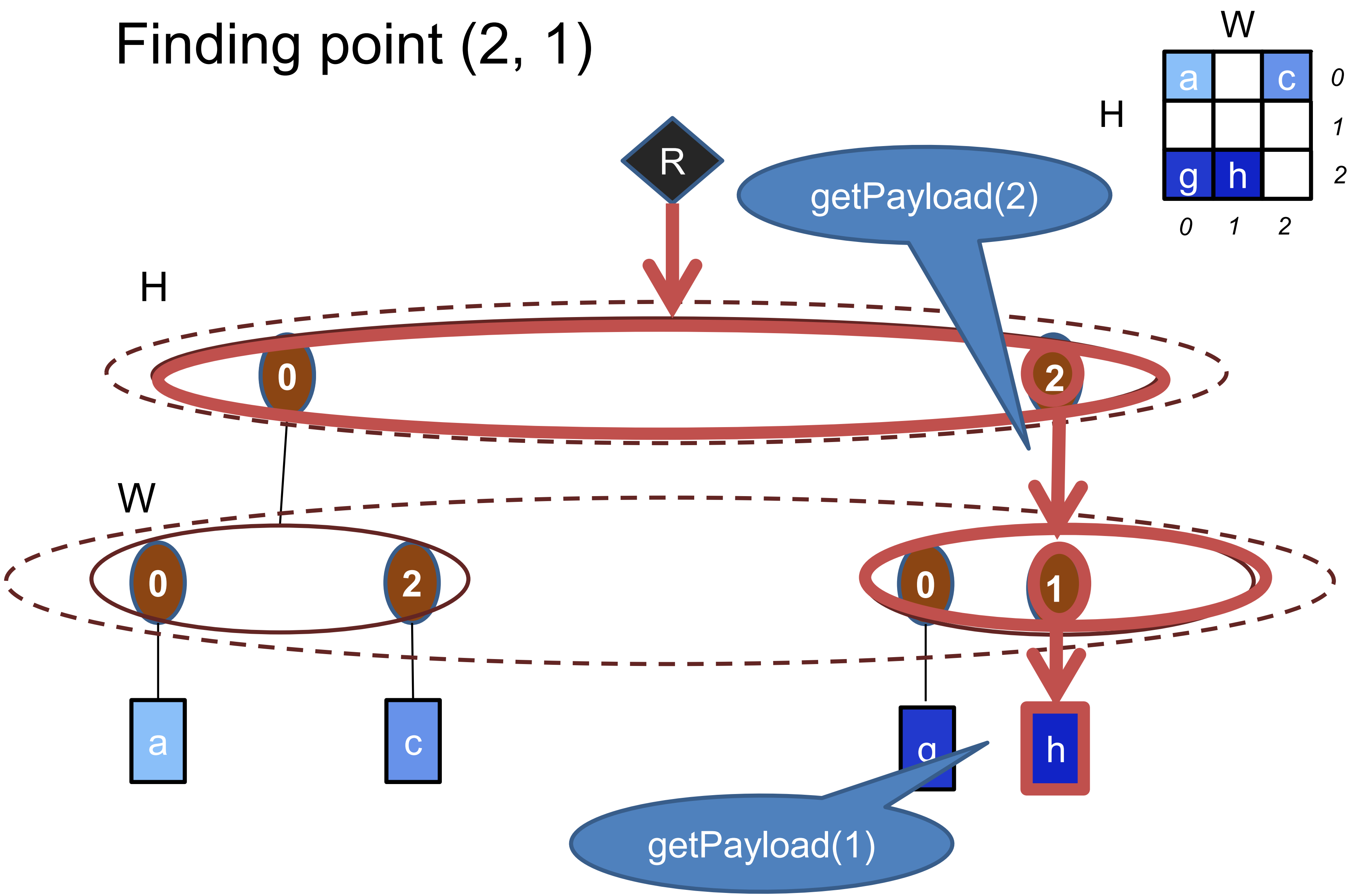
Fibertree Tensor Abstraction

What if tensor is sparse?



Fibertree Tensor Abstraction

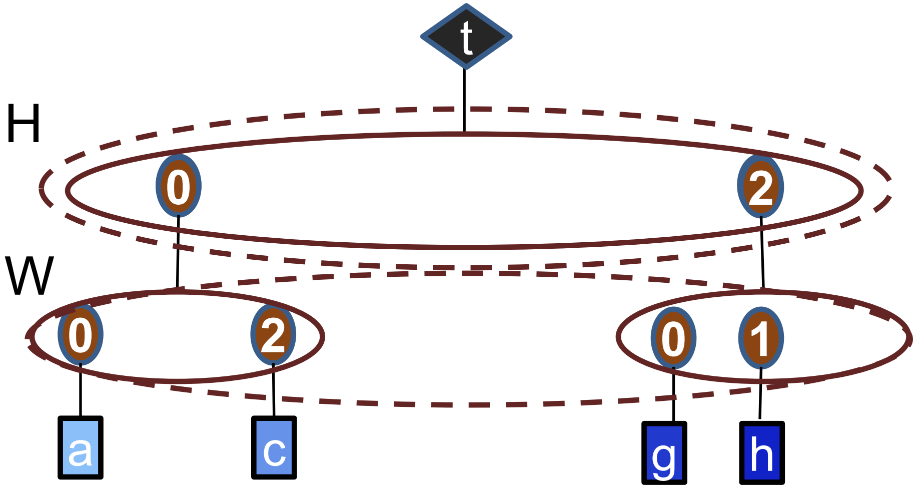
Finding point (2, 1)



Tensor Traversal (2-D)

```
# 2-D Tensor Traversal

t = Tensor(H,W)

sum = 0
for (h, t_h) in t:
  for (w, t_val) in t_h:
     sum += t_val
```

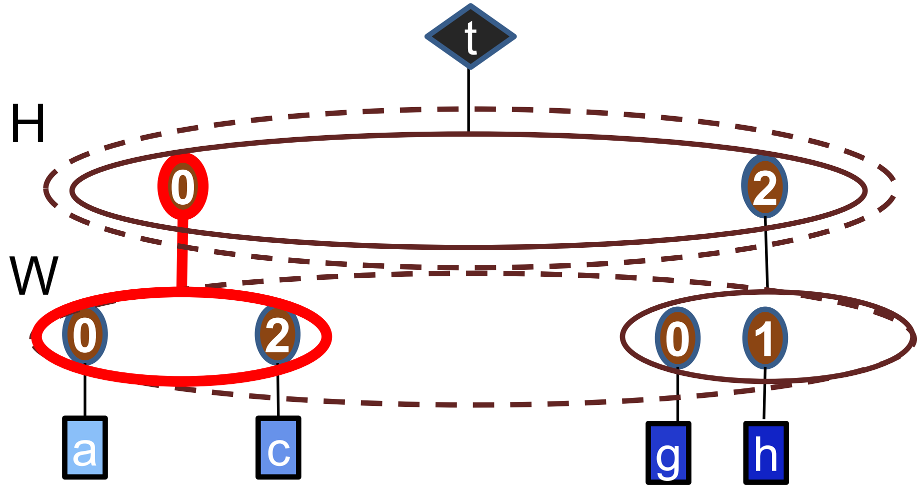
Each iteration returns a (coordinate, payload) tuple



t_pos	h	t_h_pos	w	t_val
0	0	?	?	?
0	0	0	0	a
0	0	1	2	c
1	2	?	?	?
…	…	…	…	…

Tensor Traversal (2-D)

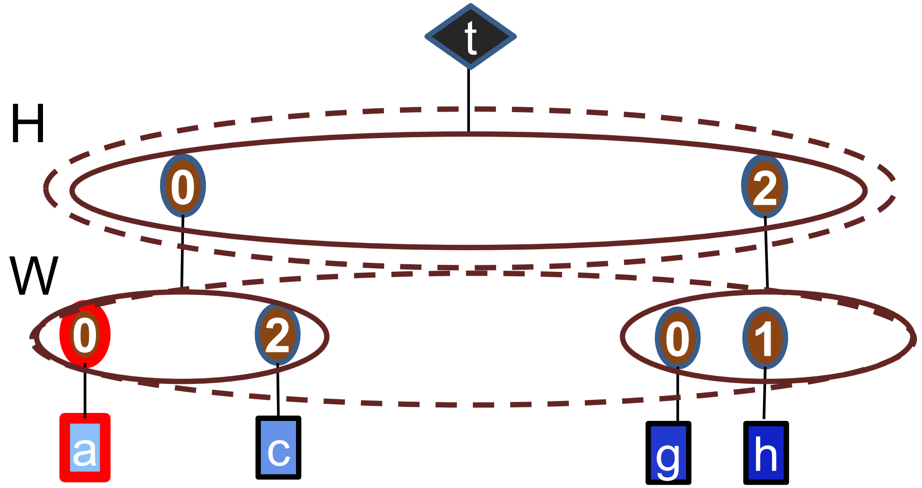
```
# 2-D Tensor Traversal

t = Tensor(H,W)

sum = 0
for (h, t_h) in t:
   for (w, t_val) in t_h:
      sum += t_val
```



t_pos	h	t_h_pos	w	t_val
0	0	?	?	?
0	0	0	0	a
0	0	1	2	c
1	2	?	?	?
…	…	…	…	…

Tensor Traversal (2-D)

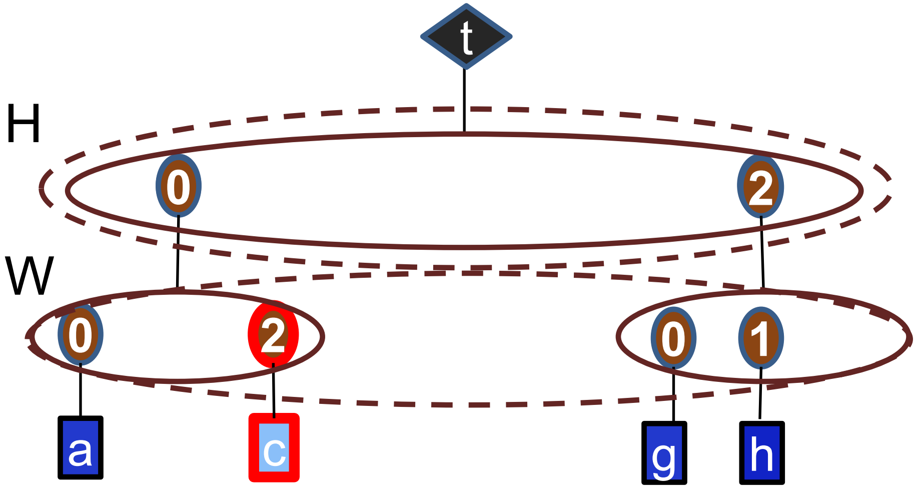
```
# 2-D Tensor Traversal

t = Tensor(H,W)

sum = 0
for (h, t_h) in t:
  for (w, t_val) in t_h:
     sum += t_val
```



t_pos	h	t_h_pos	w	t_val
0	0	?	?	?
0	0	0	0	a
0	0	1	2	c
1	2	?	?	?
…	…	…	…	…

Tensor Traversal (2-D)

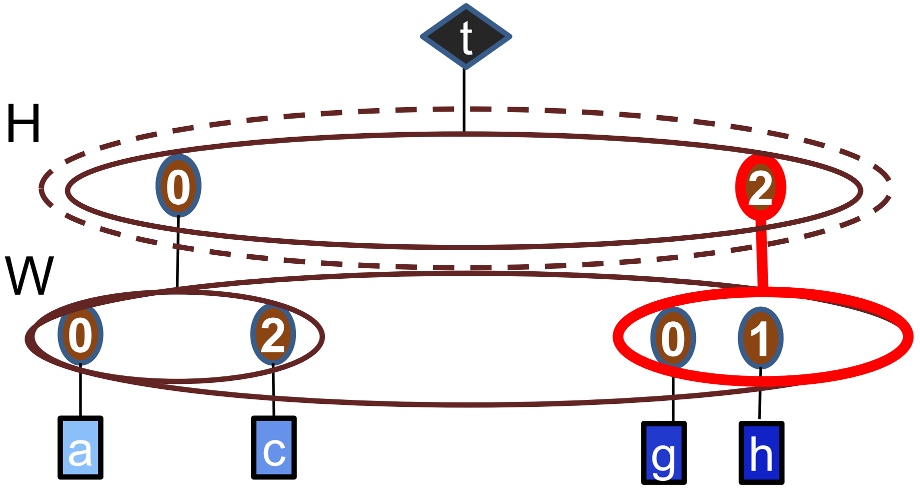
```
# 2-D Tensor Traversal

t = Tensor(H,W)

sum = 0
for (h, t_h) in t:
   for (w, t_val) in t_h:
      sum += t_val
```



t_pos	h	t_h_pos	w	t_val
0	0	?	?	?
0	0	0	0	a
0	0	1	2	c
1	2	?	?	?
…	…	…	…	…

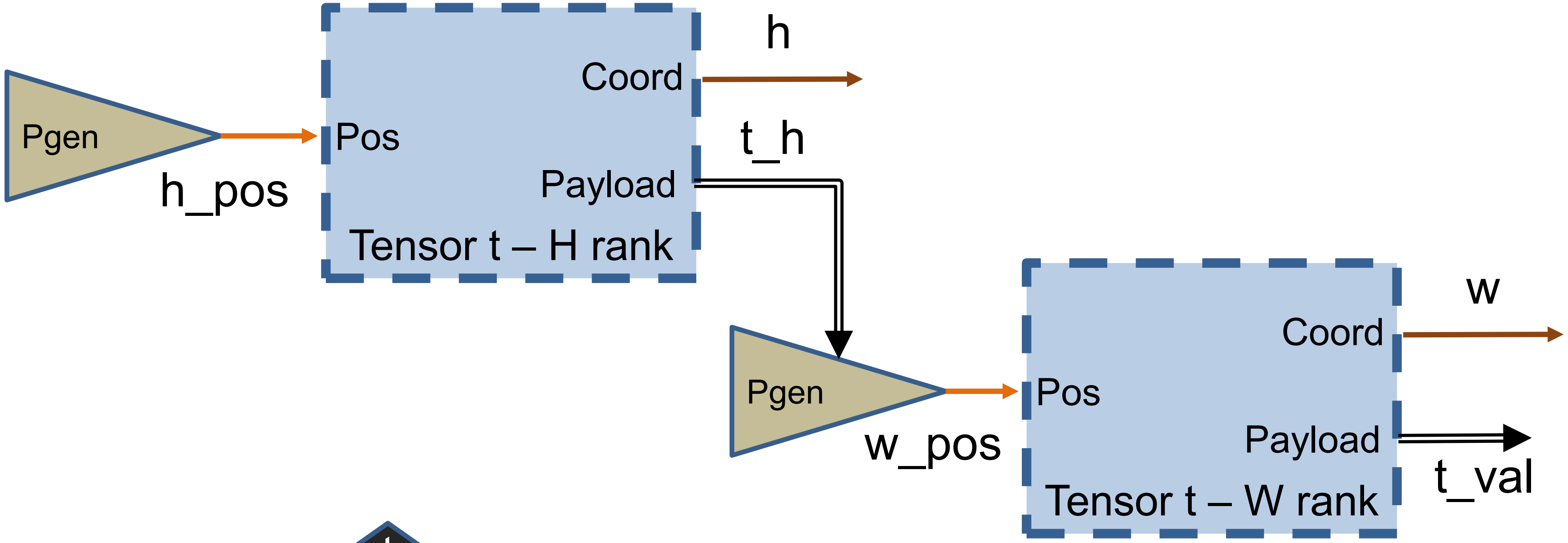
Tensor Traversal (2-D)

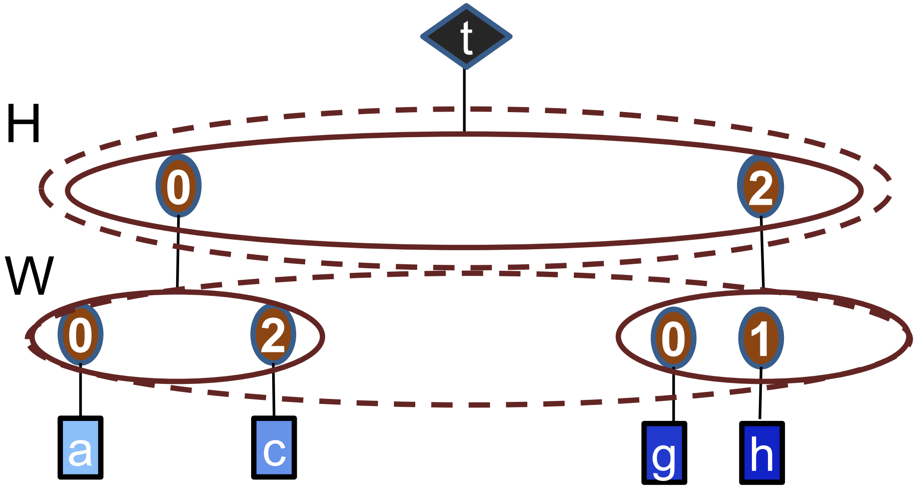
```
# 2-D Tensor Traversal

t = Tensor(H,W)

sum = 0
for (h, t_h) in t:
   for (w, t_val) in t_h:
      sum += t_val
```



t_pos	h	t_h_pos	w	t_val
0	0	?	?	?
0	0	0	0	a
0	0	1	2	c
1	2	?	?	?
…	…	…	…	…

Tensor Traversal (2-D)



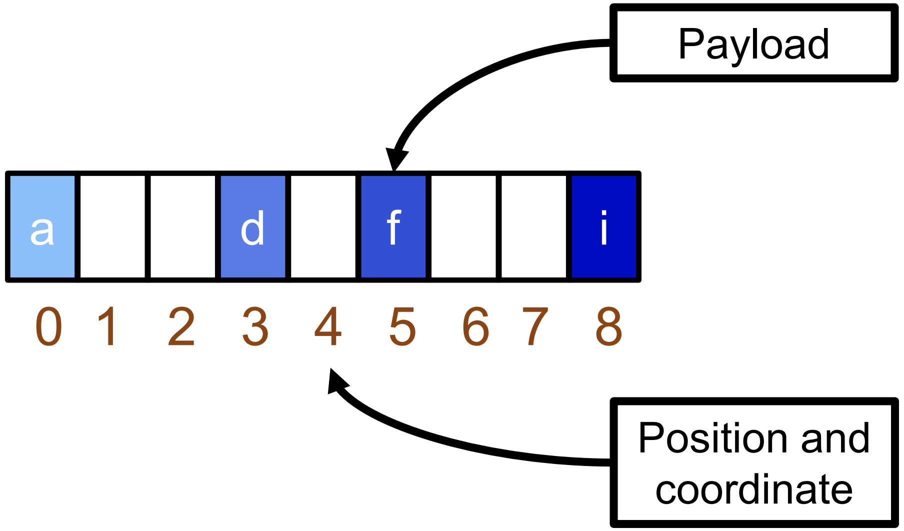


Concordant Traversal

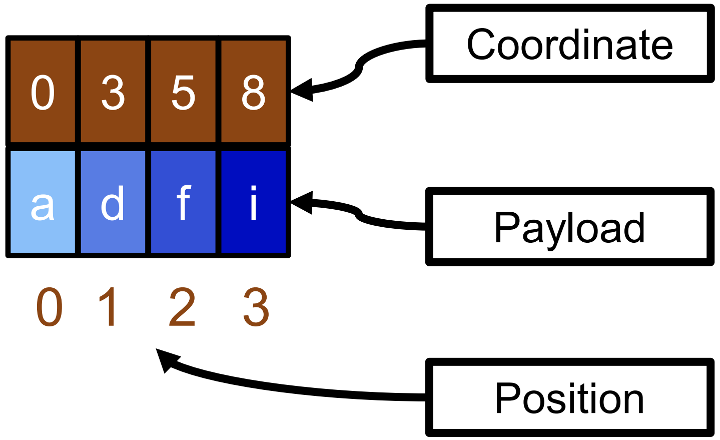
Example Fiber Representations

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

Array



Coordinate/Payload List



Data in a fiber is accessed by its position or offset in memory

Fiber Representation Choices

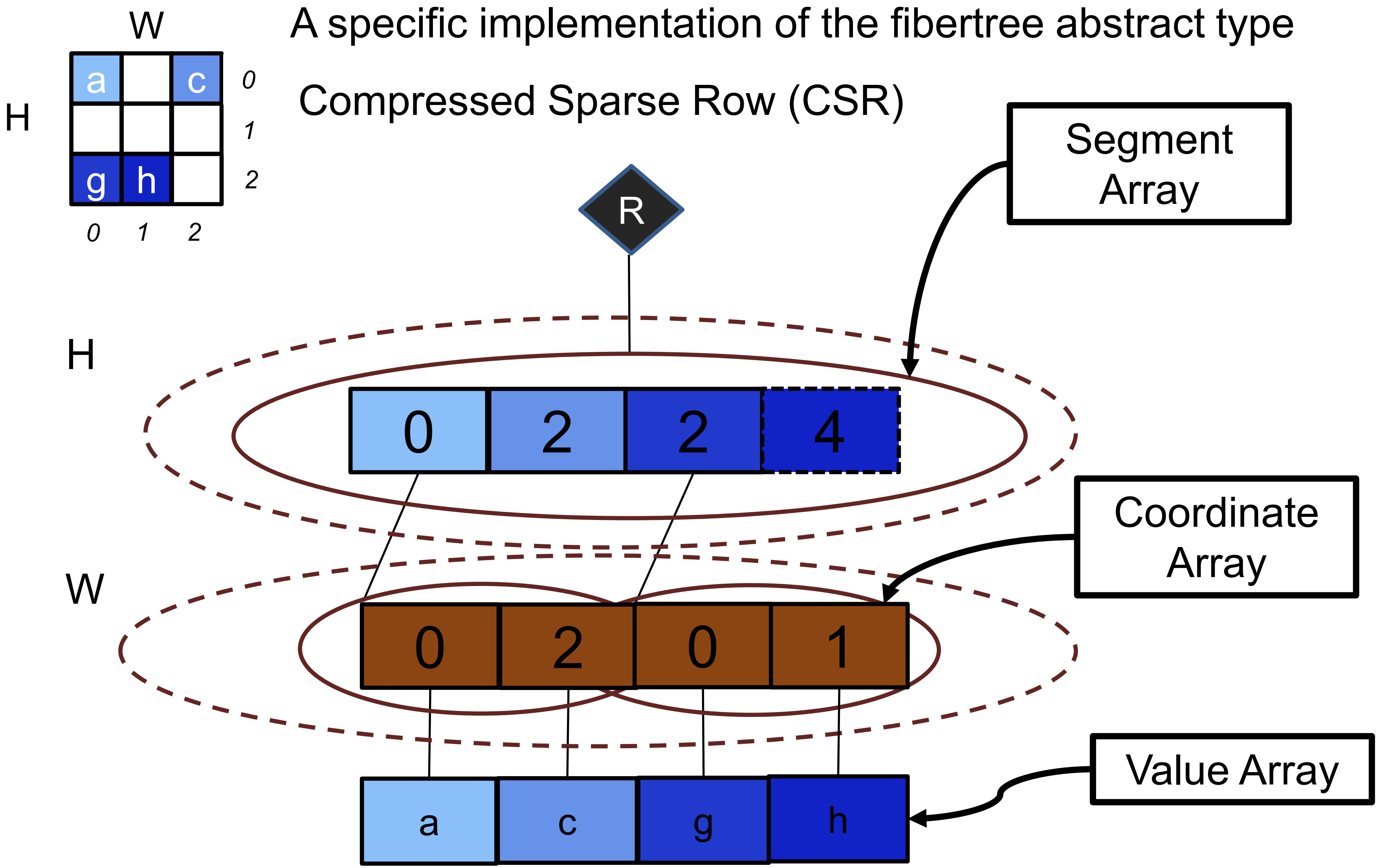
- Implicit Coordinates
  - Uncompressed (no metadata required)
  - Compressed – e.g., run length encoded
- Explicit Coordinates
  - E.g., coordinate/payload list
- Compressed vs Uncompressed
  - Compressed/uncompressed is an attribute of the representation*.
  - Uncompressed means size is proportional to maximum coordinate value
  - Compressed formats will have metadata overhead relative to uncompressed formats. For dense data, this may cost more than just using an uncompressed format.
  - Space efficiency of a representation depends on sparsity

*Note: sparsity/density is an attribute of the data.

Uncompressed/Compressed Representation

A specific implementation of the fibertree abstract type

Compressed Sparse Row (CSR)



Tensor Traversal (CSR Style)

```
# 2-D Tensor Traversal (CSR)

t_segs = Array(H)
t_coords = Array(W)
t_vals = Array(W)

sum = 0
for t_h_pos in [0,H):
  h = t_h_pos
  t_w_start = t_segs[t_h_pos]
  t_w_len = t_segs[t_h_pos+1]-t_w_start
  for t_w_pos in [t_w_start, t_w_len):
     h = t_coords[t_w_pos]
     t_val = t_vals[t_w_pos]
     sum += t_val
```

For uncompressed rank coordinate equals position

Coordinates not actually used in this example

Separation of Concerns

L23-27

Hardware Sparse Acceleration Features

Format:
Choose tensor representations to save storage space and energy associated with zero accesses

Gating:
Explicitly eliminate ineffectual storage accesses and computes by letting the hardware unit staying idle for the cycle to save energy

Skipping:
Explicitly eliminate ineffectual storage accesses and computes by skipping the cycle to save energy and time

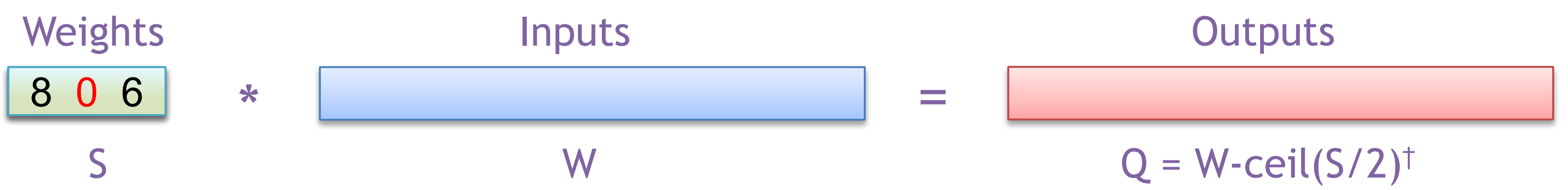
1-D Output-Stationary Convolution


```
int i[W];      # Input activations
int f[S];      # Filter weights
int o[Q];      # Output activations

for q in [0..Q):
       for s in [0…S):
            o[q] += i[q+s]*f[s];
}}
```

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

† Assuming: ‘valid’ style convolution

1-D Output-Stationary Convolution




```
int i[W];      # Input activations
int f[S];      # Filter weights
int o[Q];      # Output activations

for q in [0..Q):
       for s in [0..S):
           if (!f[s]) o[q] += i[q+s]*f[r]
}}
```

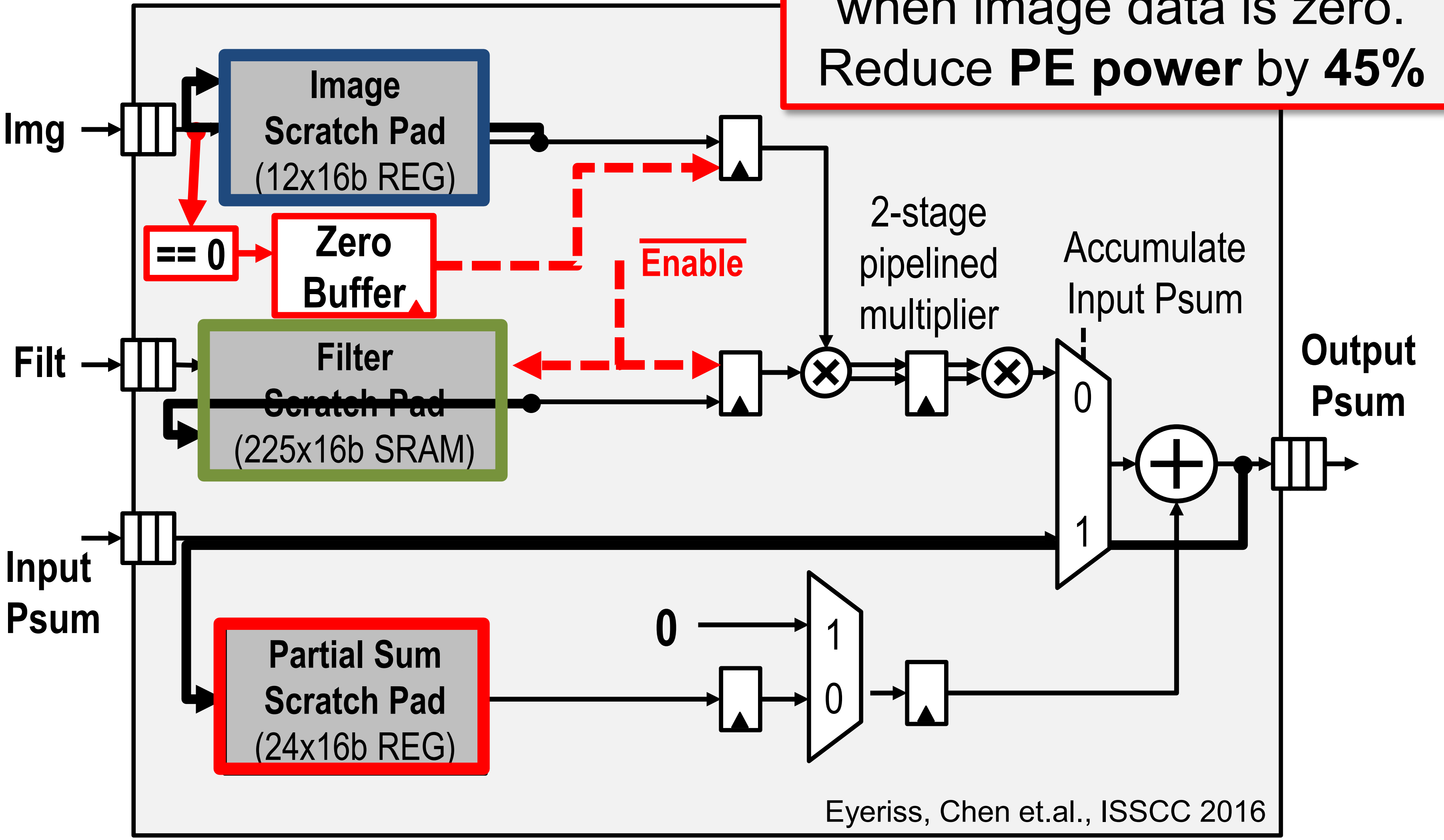
What did we save using the conditional execution?

What didn’t we save using the conditional execution?

† Assuming: ‘valid’ style convolution

Eyeriss – Clock Gating

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



CONV: Exploiting Sparse Weights

Separation of Concerns

Hardware Sparse Acceleration Features

Format:
Choose tensor representations to save storage space and energy associated with zero accesses

Gating:
Explicitly eliminate ineffectual storage accesses and computes by letting the hardware unit staying idle for the cycle to save energy

Skipping:
Explicitly eliminate ineffectual storage accesses and computes by skipping the cycle to save energy and time

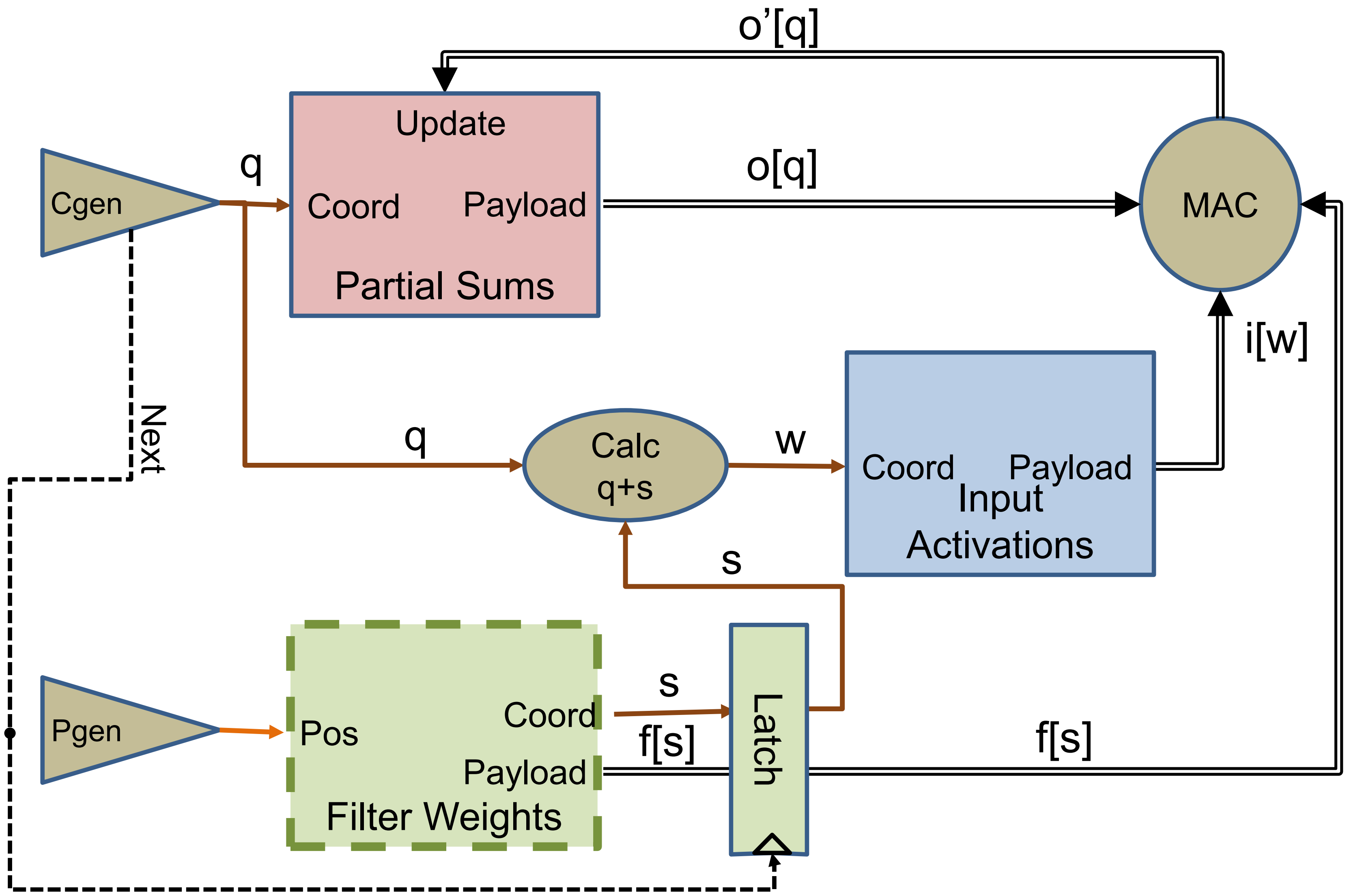
Weight Stationary - Sparse Weights

$$O_q = I_{q+s} \times F_s$$

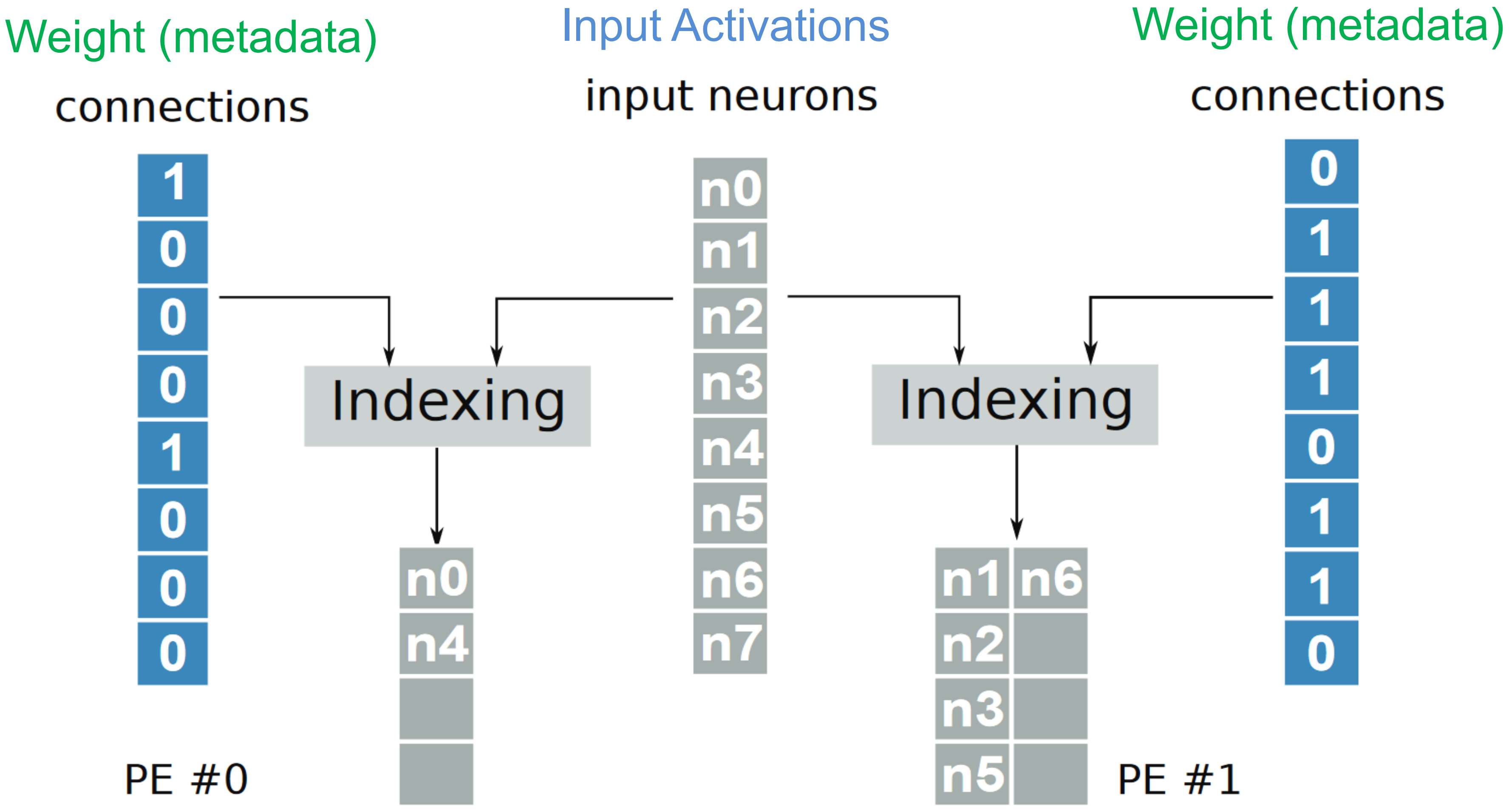
```
i = Array(W)          # Input activations
f = Tensor(S)         # Filter weights
o = Array(Q)          # Output activations

for (s, f_val) in f:
  for q in [0, Q):
    w = q + s
    o[q] += i[w] * f_val
```

Concordant traversal

Weight Stationary - Sparse Weights



Cambricon-X – Activation Access



Cambricon-X – Zhang et.al., Micro 2016

CONV: Exploiting Sparse Inputs & Sparse Weights

Einsum – Matrix Multiply

$$O_q = I_{q+s} \times F_s$$

- **Shared indices -> intersection**

L23-39

Einsum – Matrix Multiply

$$O_q = I_{q+s} \times F_s$$

- **Shared indices -> intersection**
- **Contracted indices -> reduction**

L23-40

Einsum – Matrix Multiply

$$O_q = I_{q+s} \times F_s$$

- Shared indices -> intersection
- Contracted indices -> reduction
- Uncontracted indices -> populate output point

Einsum - Convolution

$$O_q = I_{q+s} \times F_s$$

- **Shared indices -> intersection**
- **Contracted indices -> reduction**
- **Uncontracted indices -> populate output point**
- **Index arithmetic -> projection**

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

Output Stationary - Sparse Weights & Inputs

$$O_q = I_{q+s} \times F_s$$

```
i = Tensor(W)         # Input activations
f = Tensor(S)         # Filter weights
o = Array(Q)          # Output activations

for q in [0,Q):
  for (s, (f_val, i_val)) in f.project(+q) & i:
    o[q] += i_val * f_val
```

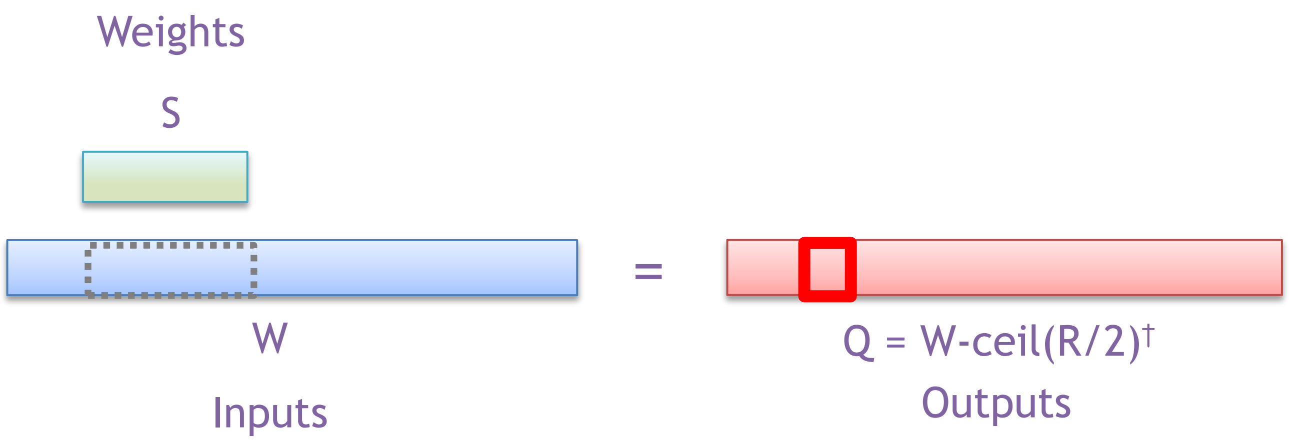
Intersection

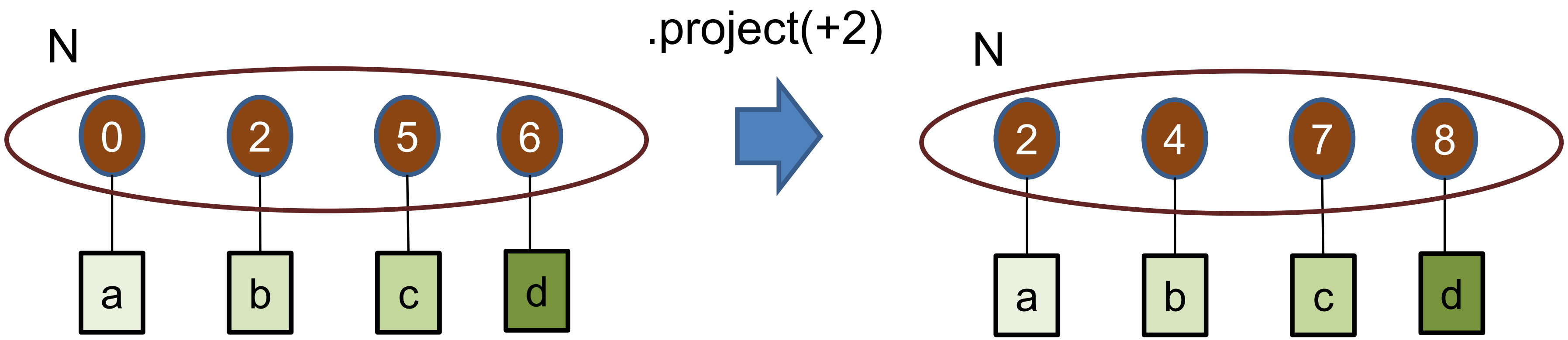
Projection

Reduction

Populate Output

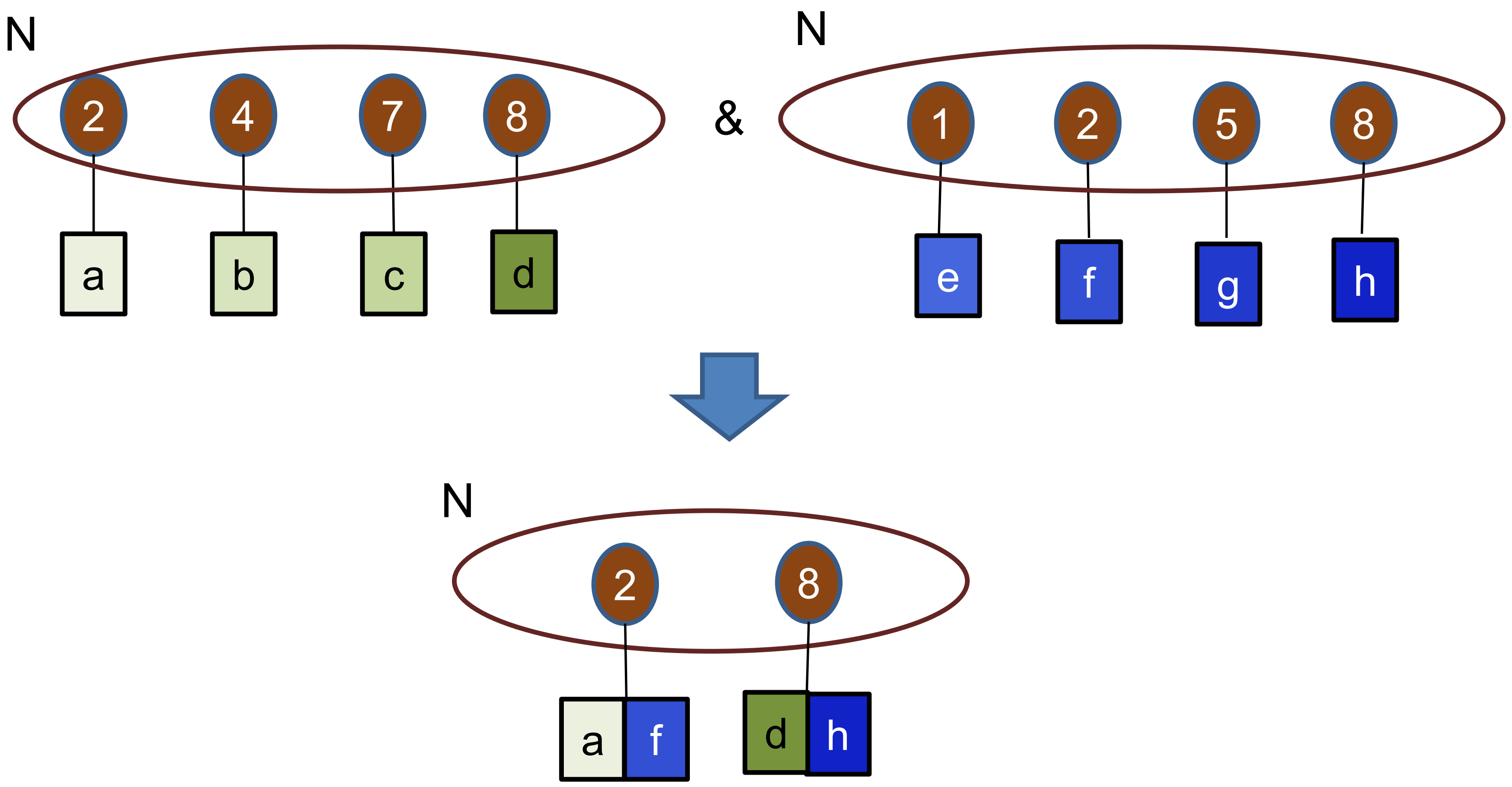
Fiber Coordinate Projection



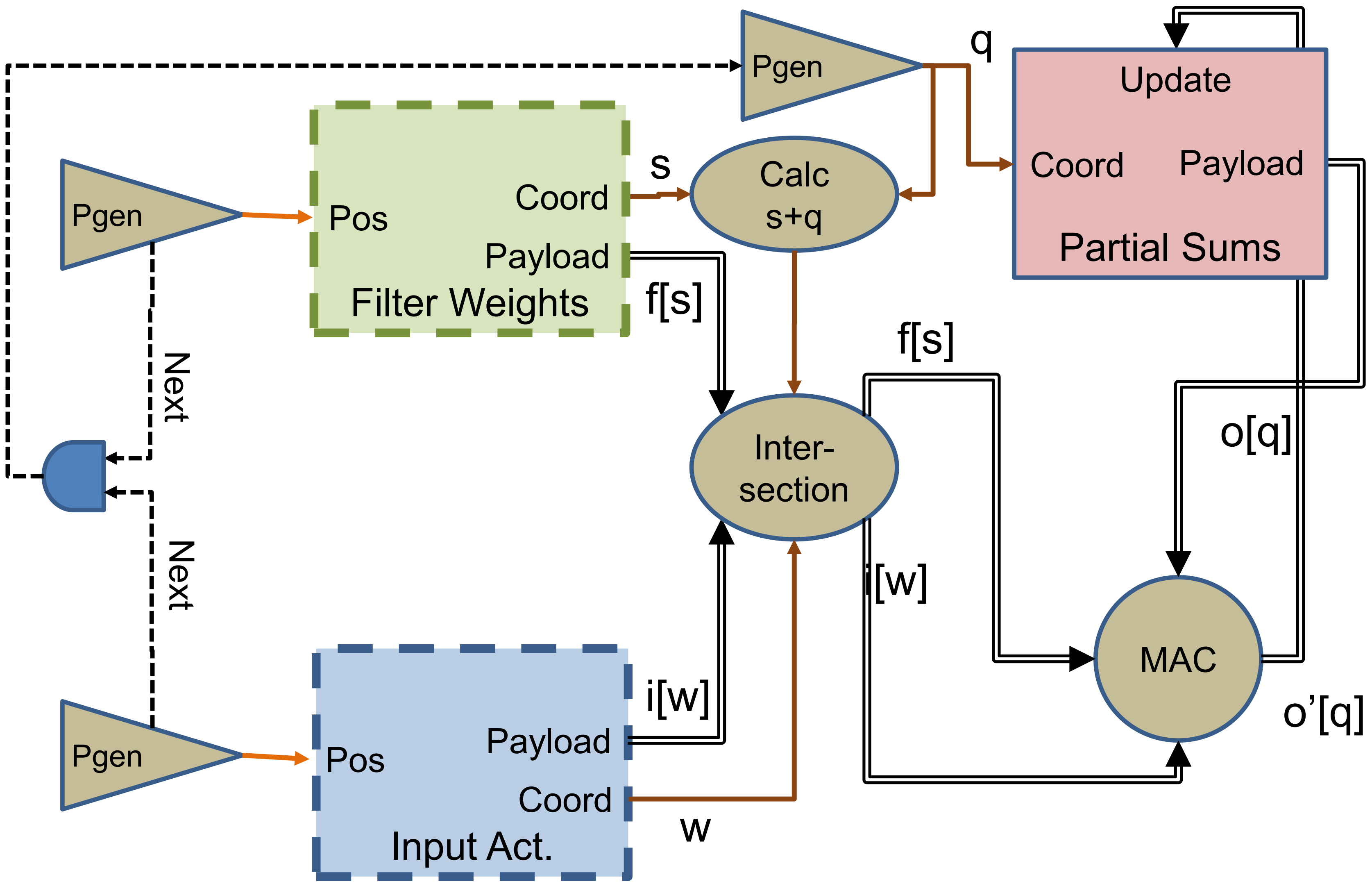


fiber-projection

Fiber Intersection



Output Stationary - Sparse Weights & Inputs



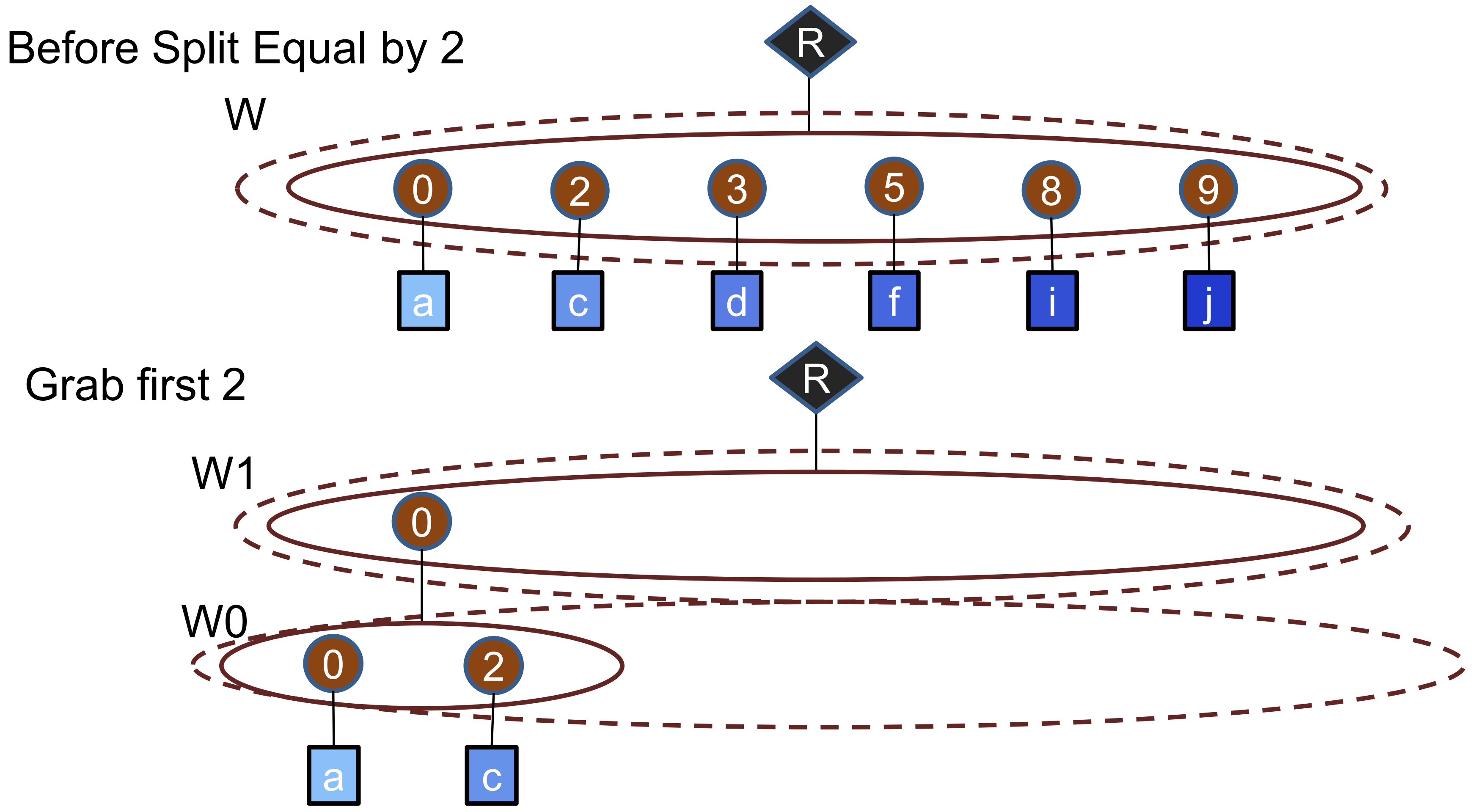
To Extend to Other Dimensions of DNN

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

- **Need to add loop nests for traversing the iteration space of:**
  - **2-D input activations and filters**
  - **Multiple input channels**
  - **Multiple output channels**

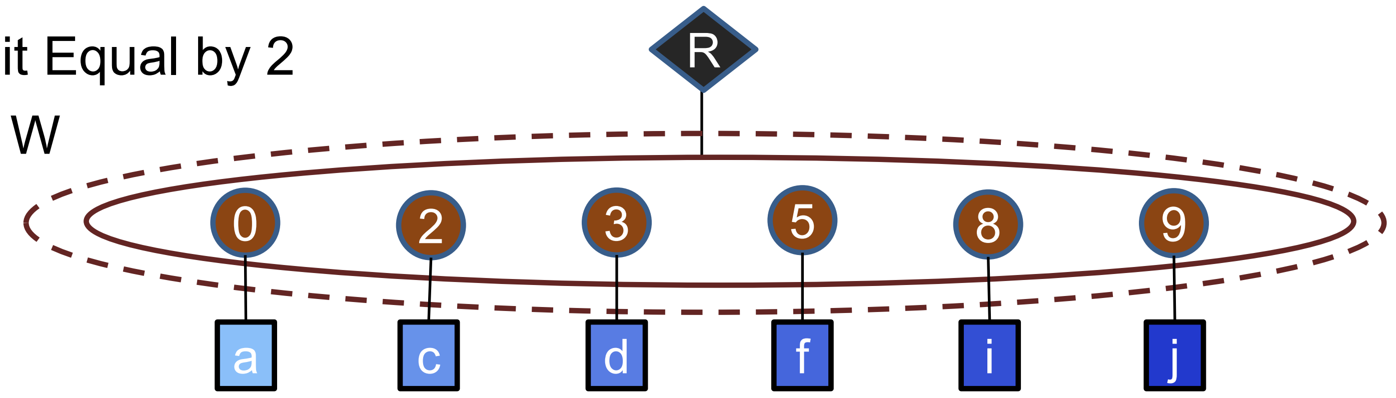
- **Add parallelism…**

Fiber Splitting Equally in Position Space

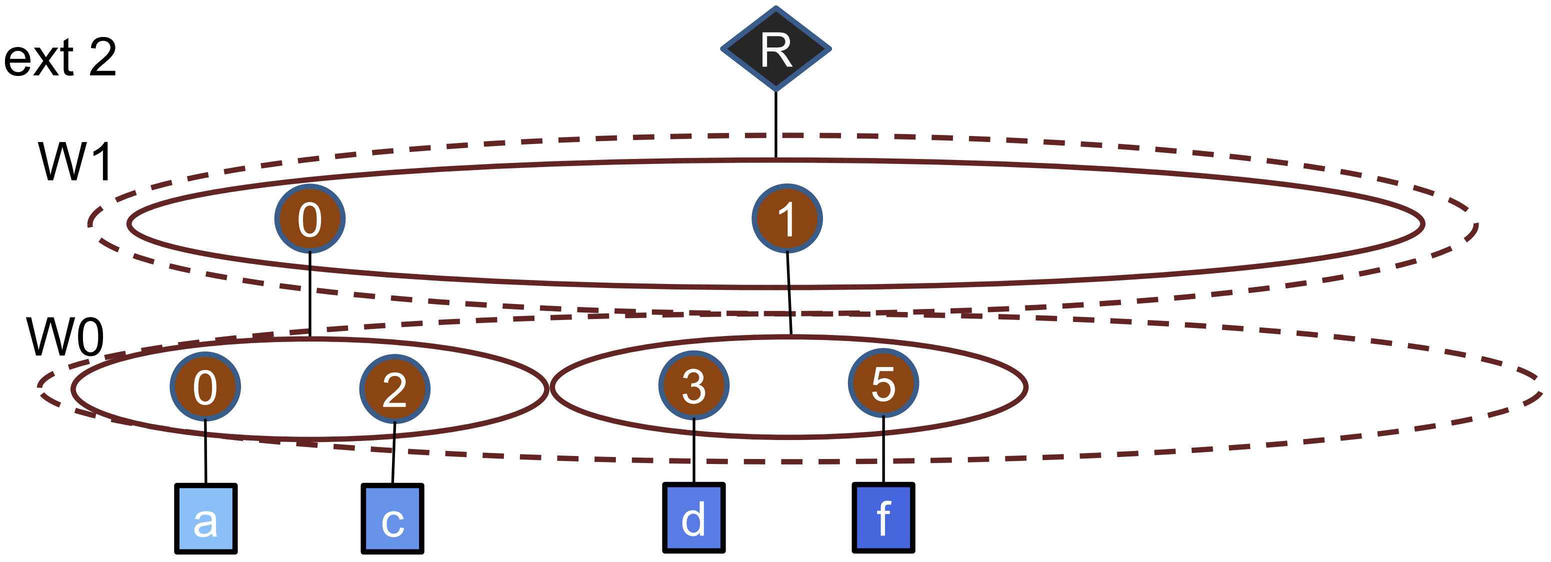


Fiber Splitting Equally in Position Space

Before Split Equal by 2

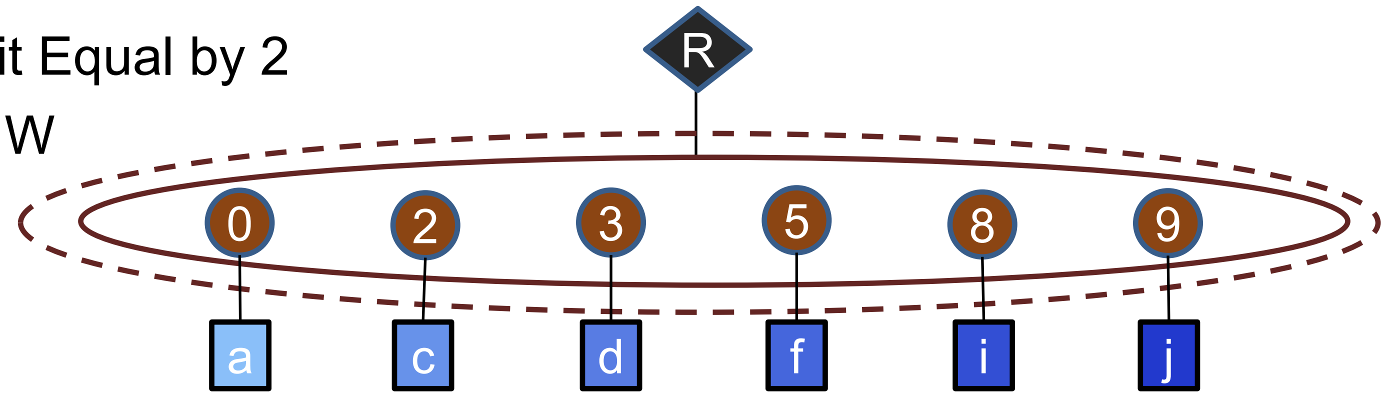


Grab next 2

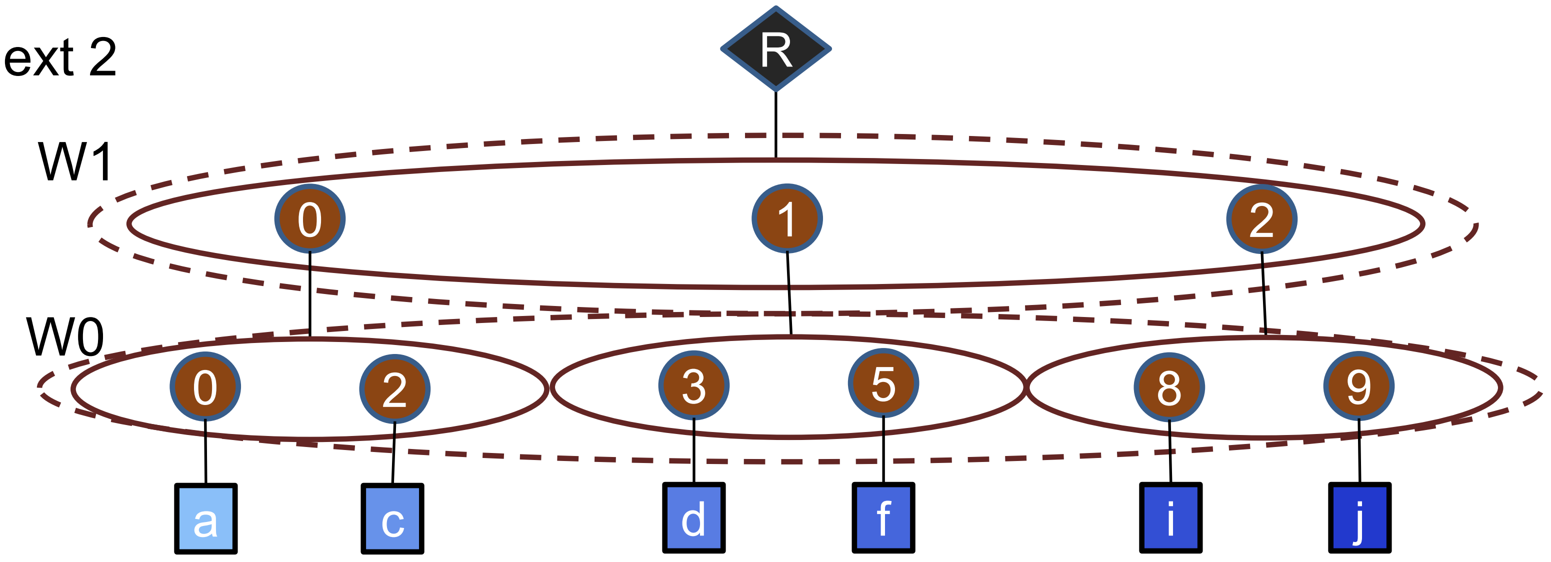


Fiber Splitting Equally in Position Space

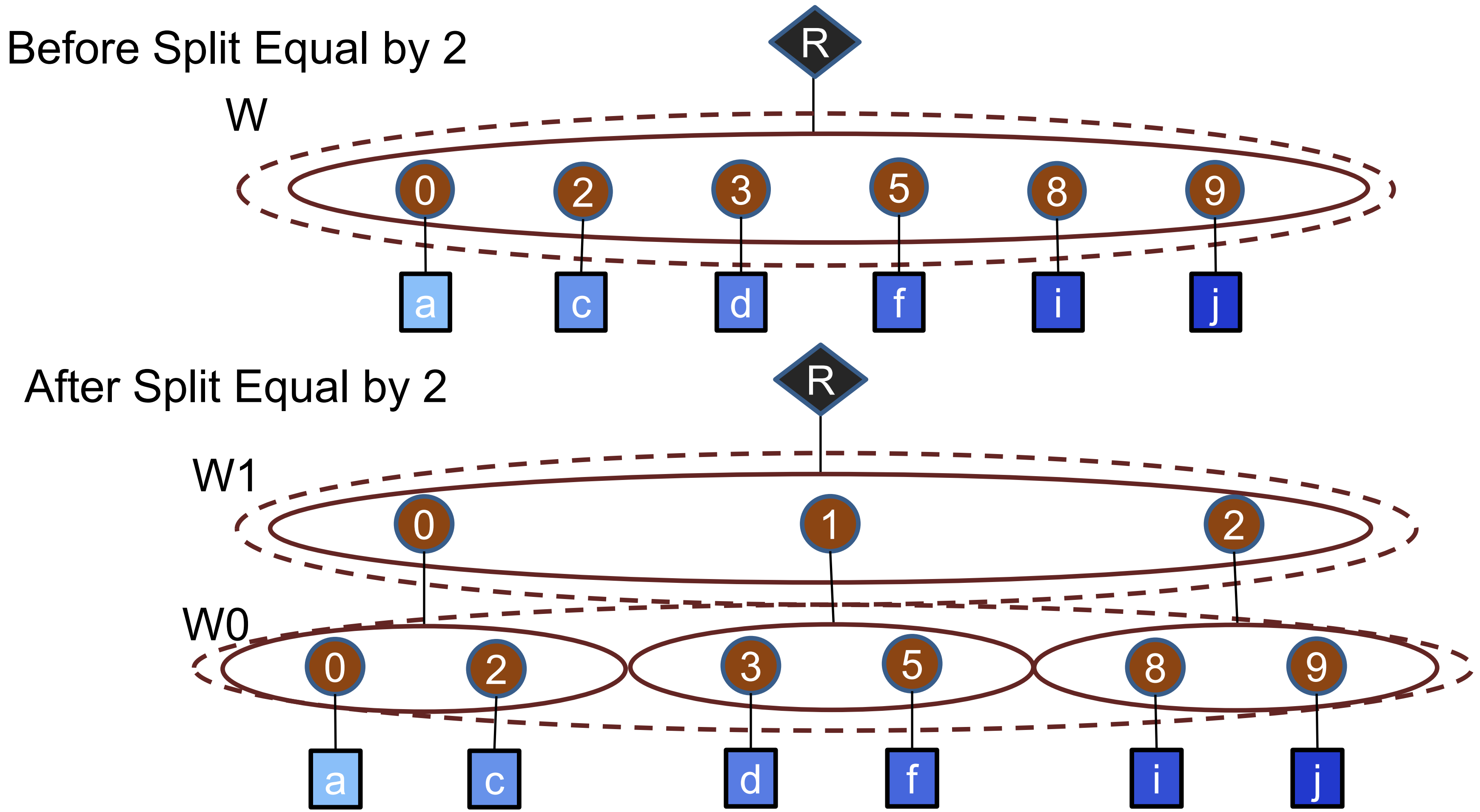
Before Split Equal by 2



Grab next 2



Fiber Splitting Equally in Position Space



Parallel Weight Stationary - Sparse Weights

```
i = Array(W)        # Input activations
f = Tensor(S)       # Filter weights
o = Array(Q)        # Output activations

for (s1, f_split) in f.splitEqual(2):
  for q1 in [0, Q/4):
     parallel-for (s0, f_val) in f_split:
       parallel-for q0 in [0, 4):
         q = q1*4 + q0
         w = q + s
         o[q] += i[w] * f_val
```

Get groups of two weights

Work on two weights in parallel

Work on four outputs at once

Calculate coordinates

Look up input activation

Accumulate multiple outputs each spatially

Separation of Concerns

Multi-head Attention (without initial embedding step)

$$K_{b,h,m,e} = I_{b,m,d} \times WK_{d,h,e}$$

$$Q_{b,h,m,e} = I_{b,m,d} \times WQ_{d,h,e}$$

$$QK_{b,h,m,p}^{B,H,M,P=M} = Q_{b,h,p,e}^{B,H,M,E} \times K_{b,h,m,e}$$

$$SN_{b,h,m,p} = exp\left(QK_{b,h,m,p}\right)$$

$$SD_{b,h,p} = SN_{b,h,m,p}$$

$$A_{b,h,m,p} = SN_{b,h,m,p}/SD_{b,h,p}$$

$$V_{b,h,m,f} = I_{b,m,d} \times WV_{d,h,f}$$

$$AV_{b,h,p,f}^{B,H,P=M,F} = A_{b,h,m,p} \times V_{b,h,m,f}$$

$$C_{b,p,h\times F+f}^{B,P=M,G=H\times F} = AV_{b,h,p,f}$$

$$Z_{b,p,d} = C_{b,p,f} \times WZ_{g,d}$$

Passes of a Cascade of Einsums

Pass: a traversal of every element of a particular fiber of a particular rank and tensor; each time an element must be revisited after visiting every other element of that fiber, there is an additional pass

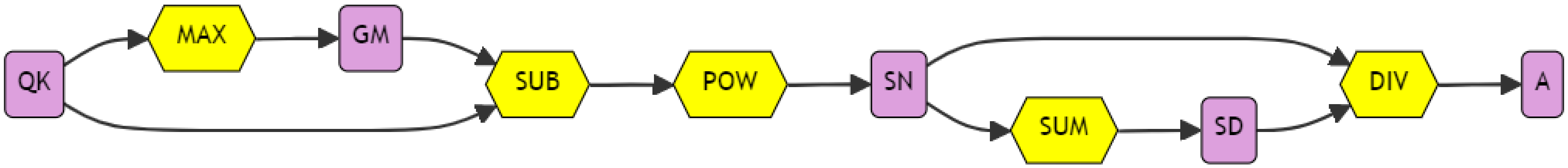
1D Softmax

$$N_m = e^{I_m}$$

$$D = N_m$$

$$A_m = N_m / D$$

Softmax for Numerically Stable Attention



$$\boxed{GM_p = QK_{m,p} :: \bigvee_m \boxed{\max}(\cup)}$$

$$SN_{m,p} = \boxed{e^{QK_{m,p} - GM_p}}$$

$$SD_p = SN_{m,p}$$

$$A_{m,p} = SN_{m,p} / SD_p$$

Many Attention Variants

3-pass cascade

$$QK_{m,p} = Q_{e,p} \times K_{e,m}$$
$$GM_p = QK_{m,p} :: \bigvee_{m} \max(\cup)$$

$$SN_{m,p} = e^{QK_{m,p} - GM_p}$$
$$SD_p = SN_{m,p}$$

$$A_{m,p} = SN_{m,p} / SD_p$$
$$AV_{f,p} = A_{m,p} \times V_{f,m}$$

1-pass cascade (FuseMax)

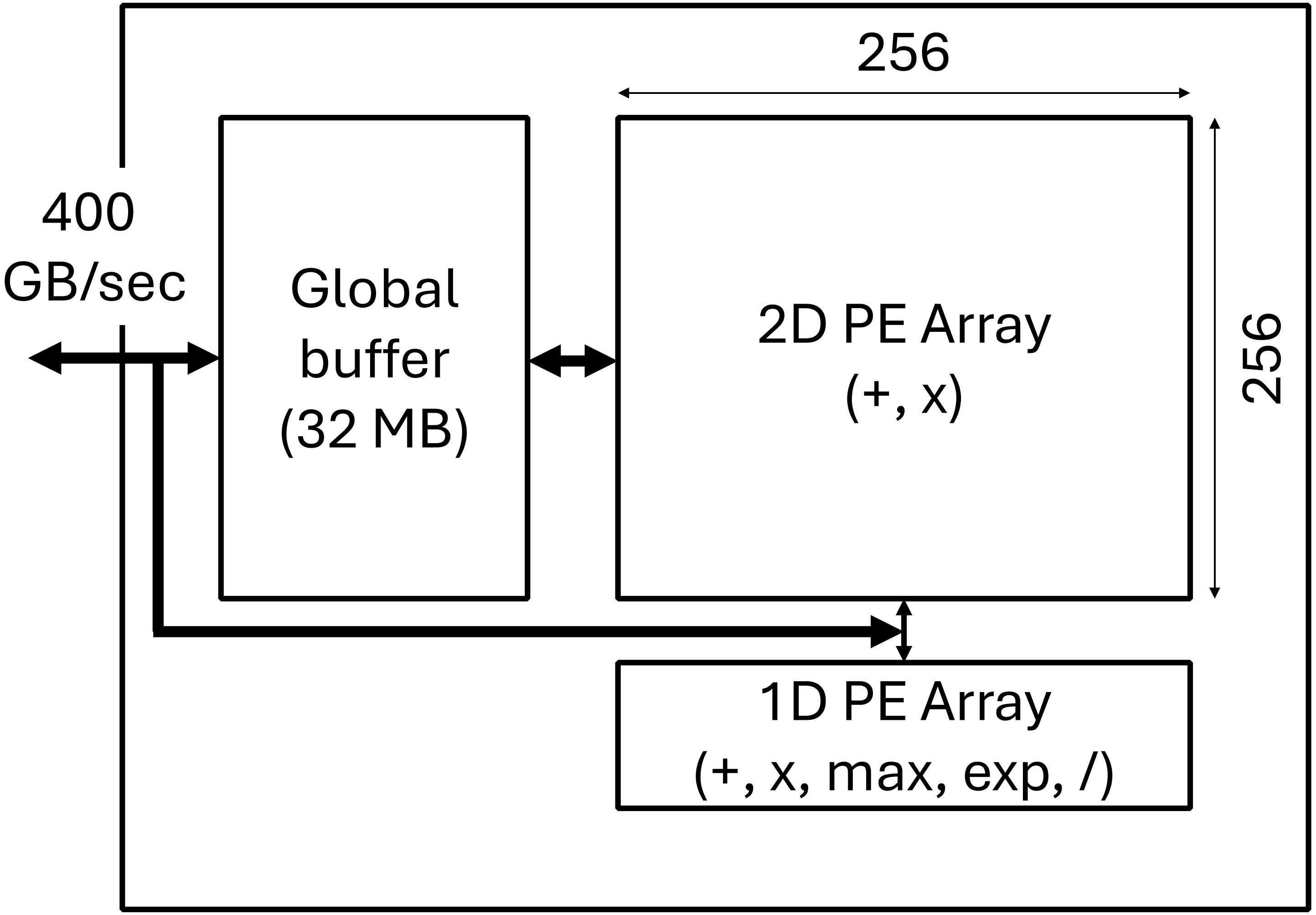
$$BQK_{m1,m0,p} = Q_{e,p} \times BK_{e,m1,m0}$$
$$LM_{m1,p} = BQK_{m1,m0,p} :: \bigvee_{m0} \max(\cup)$$
$$RM_{m1+1,p} = \max(RM_{m1,p}, LM_{m1,p})$$
$$SLN_{m1,m0,p} = e^{BQK_{m1,m0,p} - RM_{m1+1,p}}$$
$$SLD_{m1,p} = SLN_{m1,m0,p}$$
$$SLNV_{f,m1,p} = SLN_{m1,m0,p} \times BV_{f,m1,m0}$$
$$PRM_{m1,p} = e^{RM_{m1,p} - RM_{m1+1,p}}$$
$$SPD_{m1,p} = RD_{m1,p} \times PRM_{m1,p}$$
$$RD_{m1+1,p} = SLD_{m1,p} + SPD_{m1,p}$$
$$SPNV_{f,m1,p} = RNV_{f,m1,p} \times PRM_{m1,p}$$
$$RNV_{f,m1+1,p} = SLNV_{f,m1,p} + SPNV_{f,m1,p}$$
$$AV_{f,p} = RNV_{f,M1,p} / RD_{M1,p}$$

Many Attention Variants

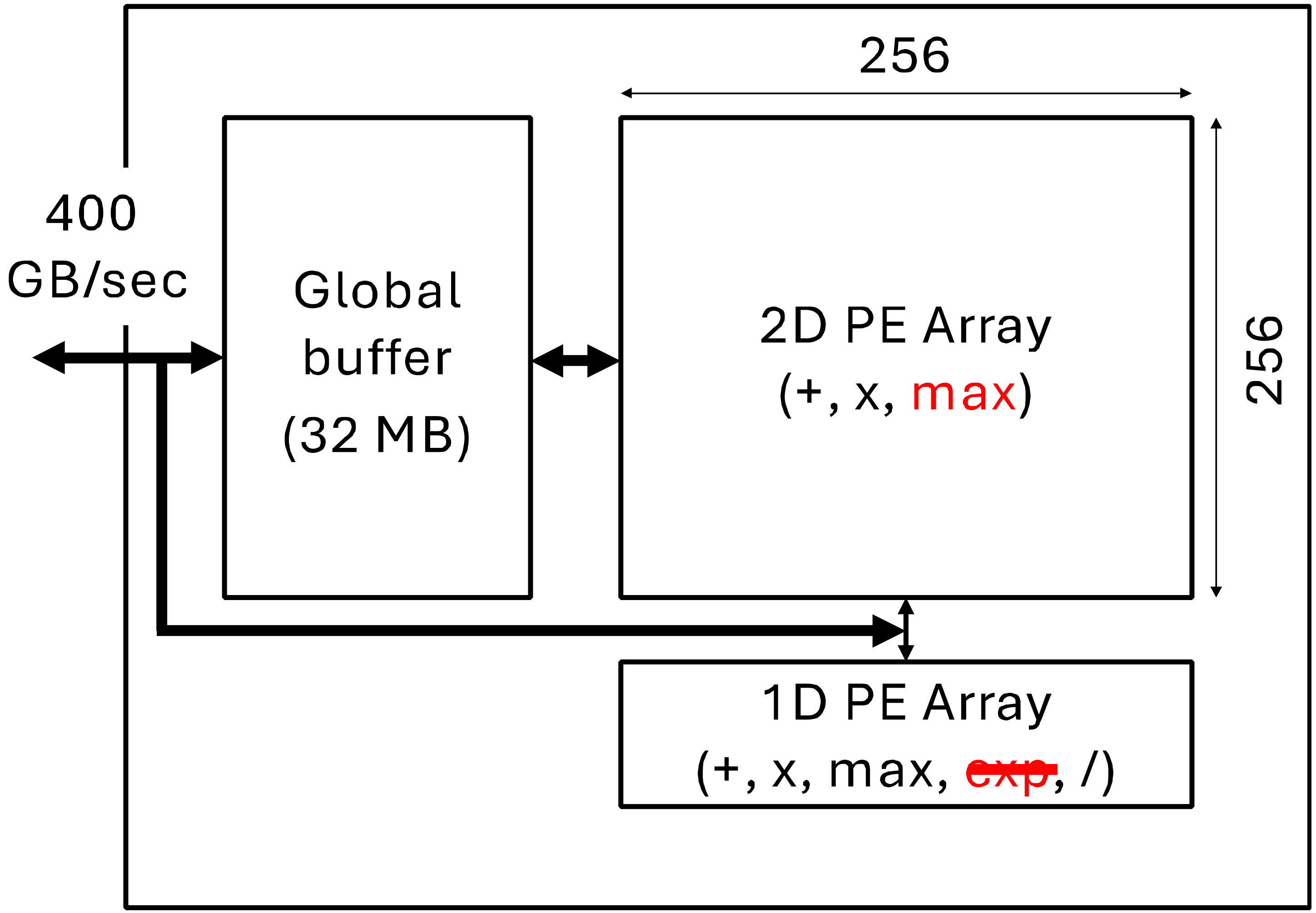
3-pass	2-pass	1-pass
PyTorch [42]	TileFlow [62]	FlashAttention [15]
TensorFlow [2]	Choi et al. [12]	FlashAttention-2 [14]
FLAT [28]		Rabe and Staats [47]
E.T. [6]		

TABLE I: Classifying prior attention algorithms.

Spatial Architectures for Transformers

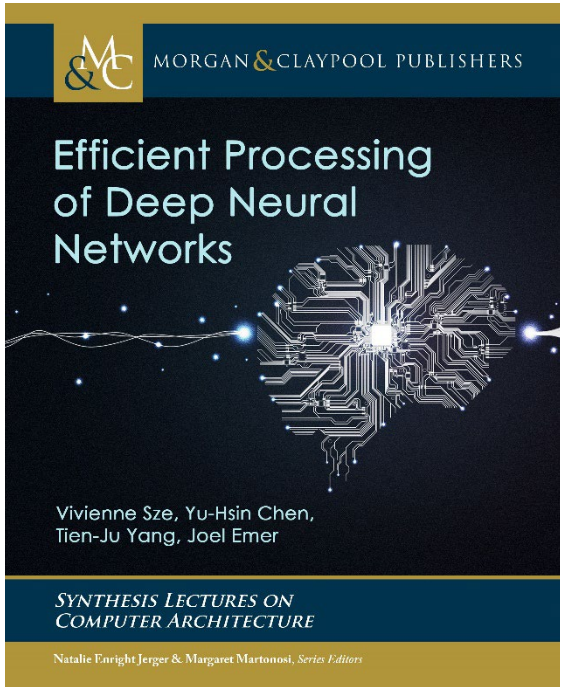


FuseMax Architecture



Performance on End-to-End Inference

Hardware Architecture for Deep Learning



Part I Understanding Deep Neural Networks
Introduction
Overview of Deep Neural Networks

Part II Design of Hardware for Processing DNNs
Key Metrics and Design Objectives
Kernel Computation
Designing DNN Accelerators
Operation Mapping on Specialized Hardware

Part III Co-Design of DNN Hardware and Algorithms
Reducing Precision
Exploiting Sparsity
Designing Efficient DNN Models
Advanced Technologies

6.593[01] – Coming Spring 2025

Thank you!