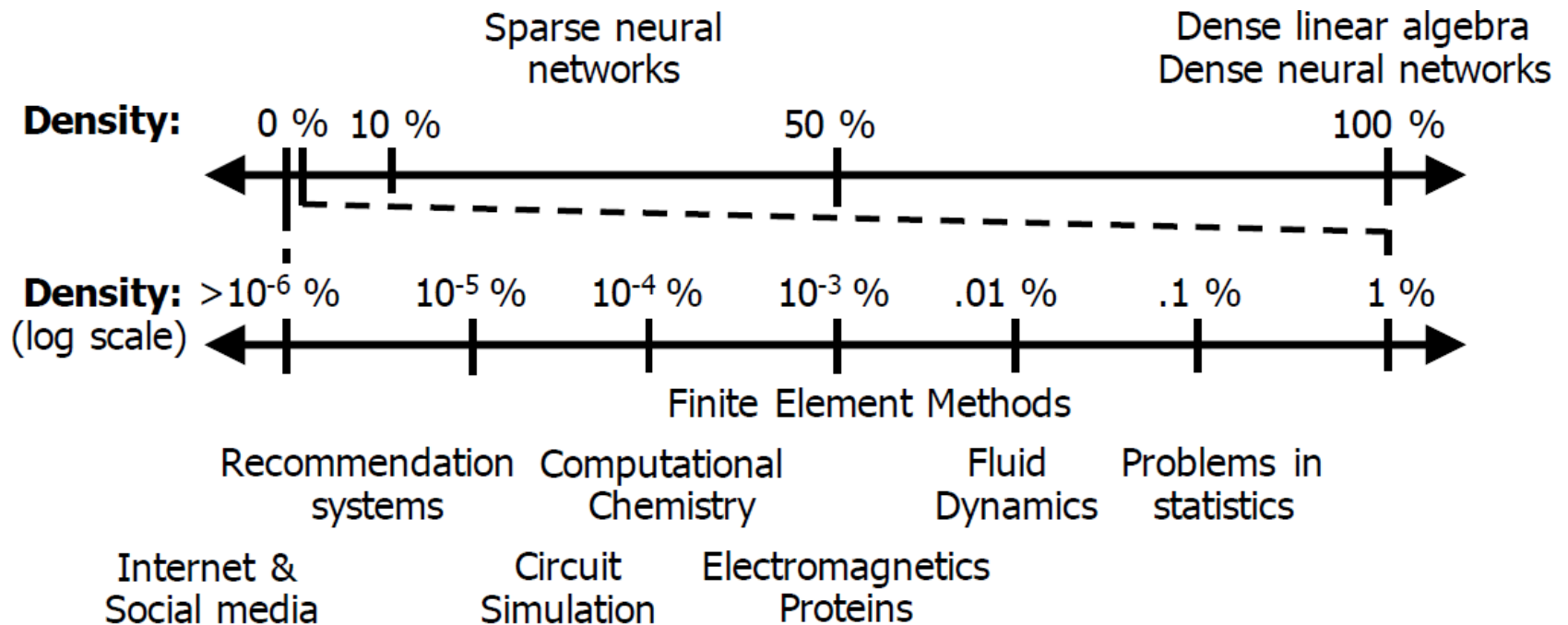


Accelerators (II)

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

$$\textit{anything} \times 0 = 0$$

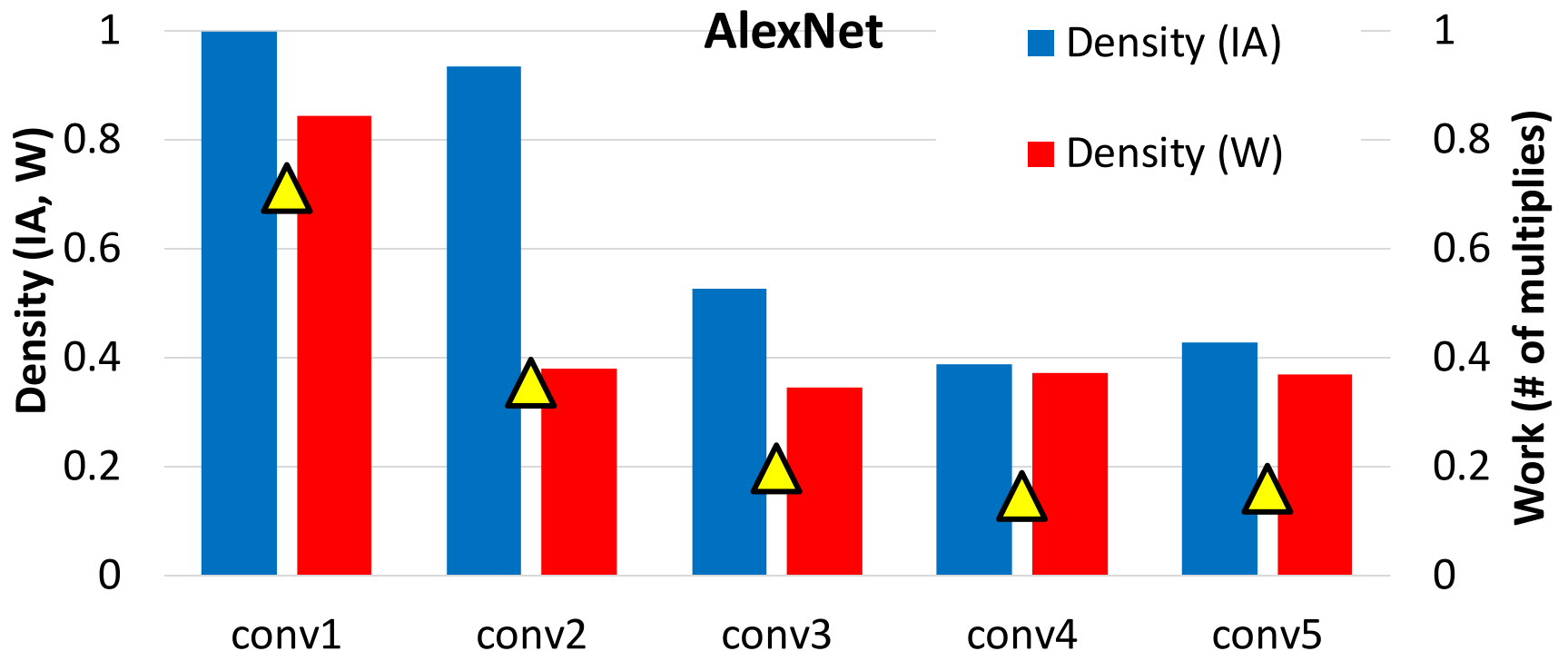
$$\textit{anything} + 0 = \textit{anything}$$



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

Motivation in DNNs

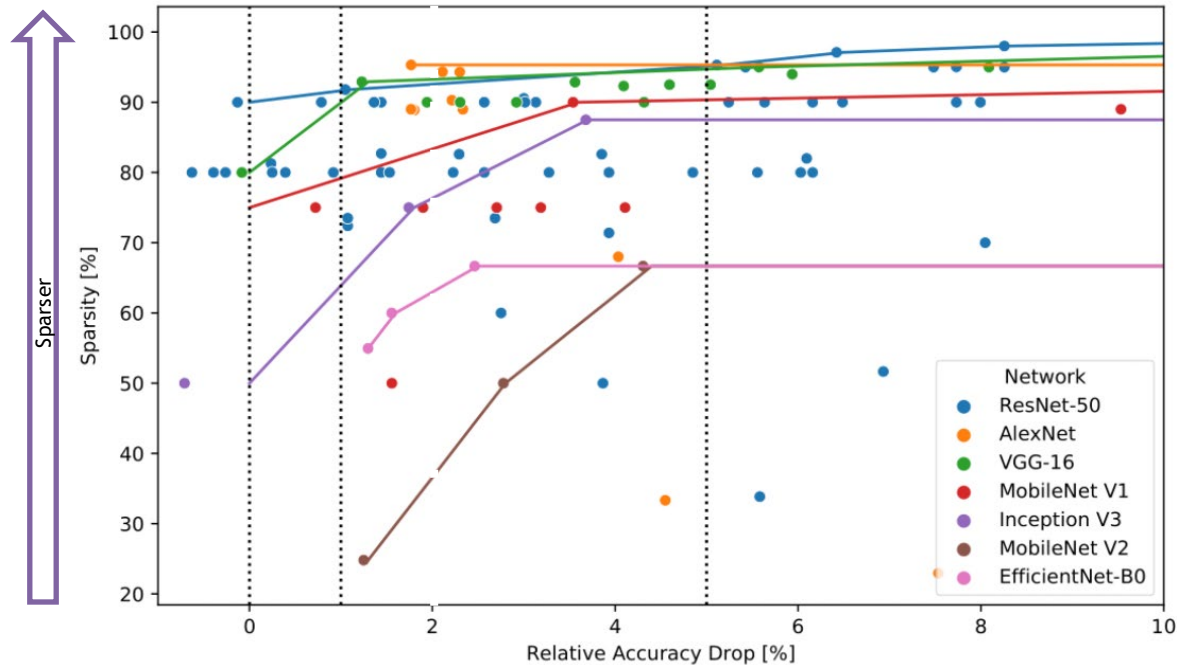
- Leverage CNN sparsity to improve energy-efficiency



SCNN, Parashar et.al., ISCA 2017

Exploitable Sparsity

Acceptable sparsity depends on target task and error tolerance



	Error Tolerance		
	$\leq 0\%$	$\leq 1\%^*$	$\leq 2\%$
ResNet-50	$\sim 90\%$	$\sim 90\%$	$\sim 91\%$
AlexNet			$\sim 93\%$
VGG-16	$\sim 80\%$	$\sim 88\%$	$\sim 92\%$
MobileNet V1	$\sim 72\%$	$\sim 79\%$	$\sim 82\%$
Inception V3	$\sim 50\%$	$\sim 62\%$	$\sim 73\%$
EfficientNet-B0			$\sim 52\%$
MobileNet V2			$\sim 25\%$

**MLPerf error tolerance*

Hoefler et al. arXiv, 2021

Hardware Sparse Acceleration Features

Format:



Choose tensor representations to save necessary storage spaces and energy associated 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

Hardware Sparse Acceleration Features

Format:



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

What is the chosen format?

Do all tensors share the same format?

Gating:



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

When is a storage access gated?

How much is the compute able to skip ahead?

Skipping:



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

At which storage level is the skipping performed?

What is the criteria for skipping?

Hardware Sparse Acceleration Features

Format:



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

Gating:



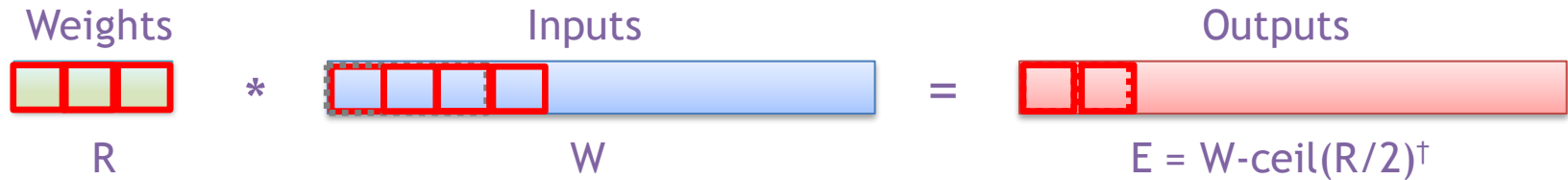
Explicitly eliminate ineffectual storage accesses and computes by letting the hardware unit staying idle for the cycle to save energy

Skipping:



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

1-D Output-Stationary Convolution



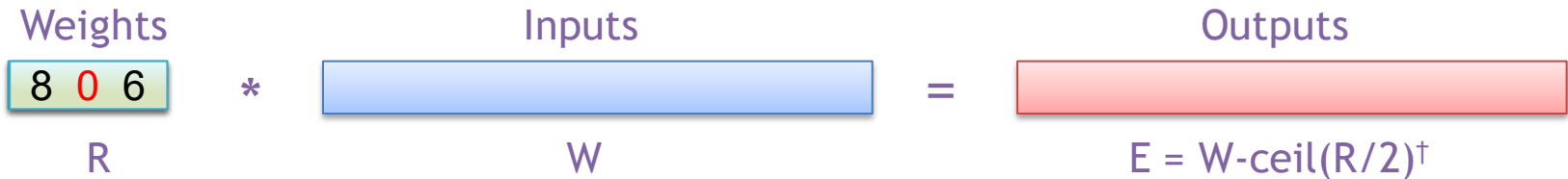
```
int i[W];      # Input activations
int f[S];      # Filter weights
int o[Q];      # Output activations

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

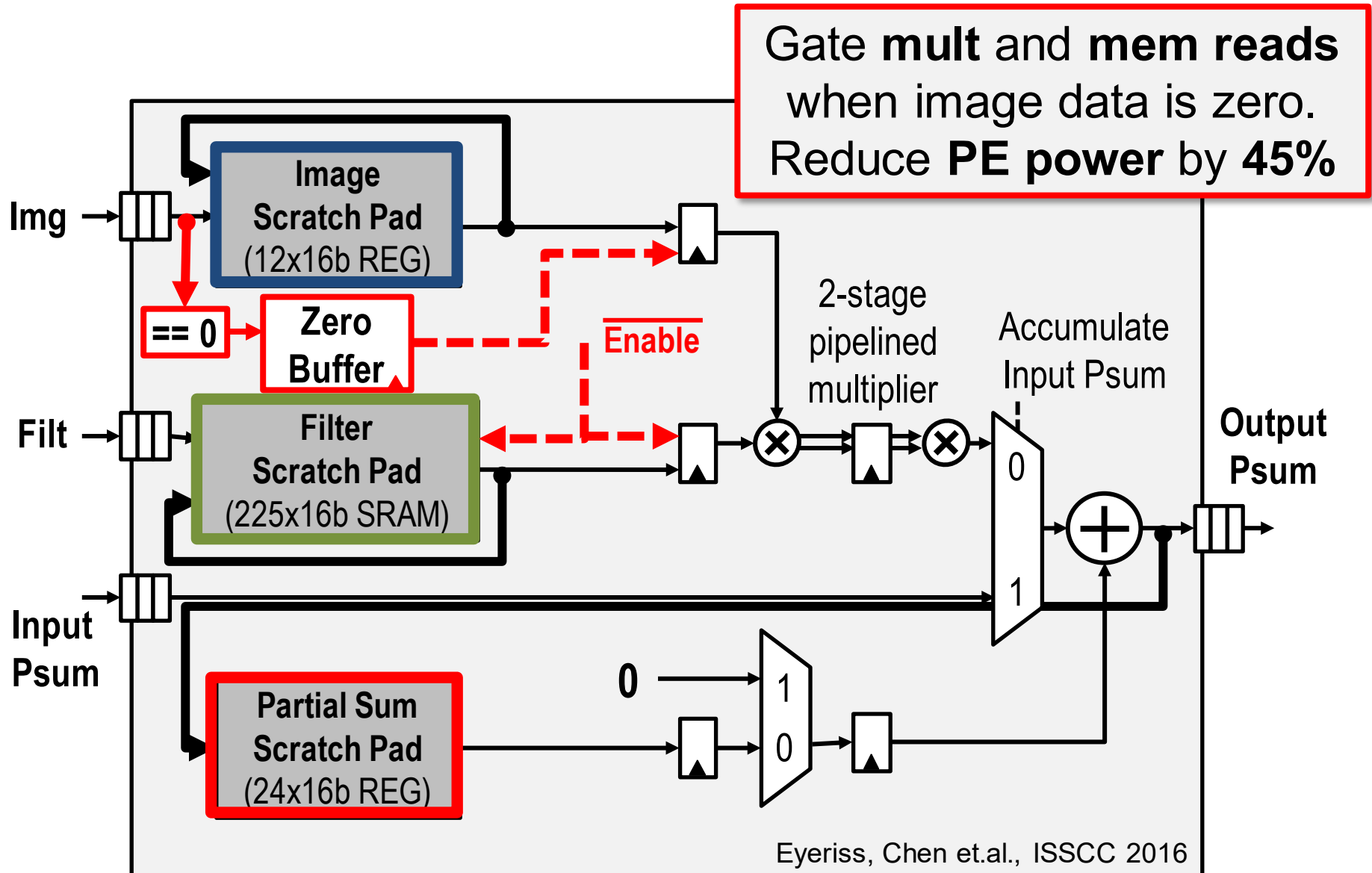
for q in [0..Q):
    for s in [0..S):
        if (!f[s]) o[q] += i[q+s]*f[r]
    }}
```

What did we save using the conditional execution?

What didn't we save using the conditional execution?

† Assuming: 'valid' style convolution

Eyeriss – Clock Gating



Sparse Tensor Representation

Hardware Sparse Acceleration Features

Format:



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

Gating:



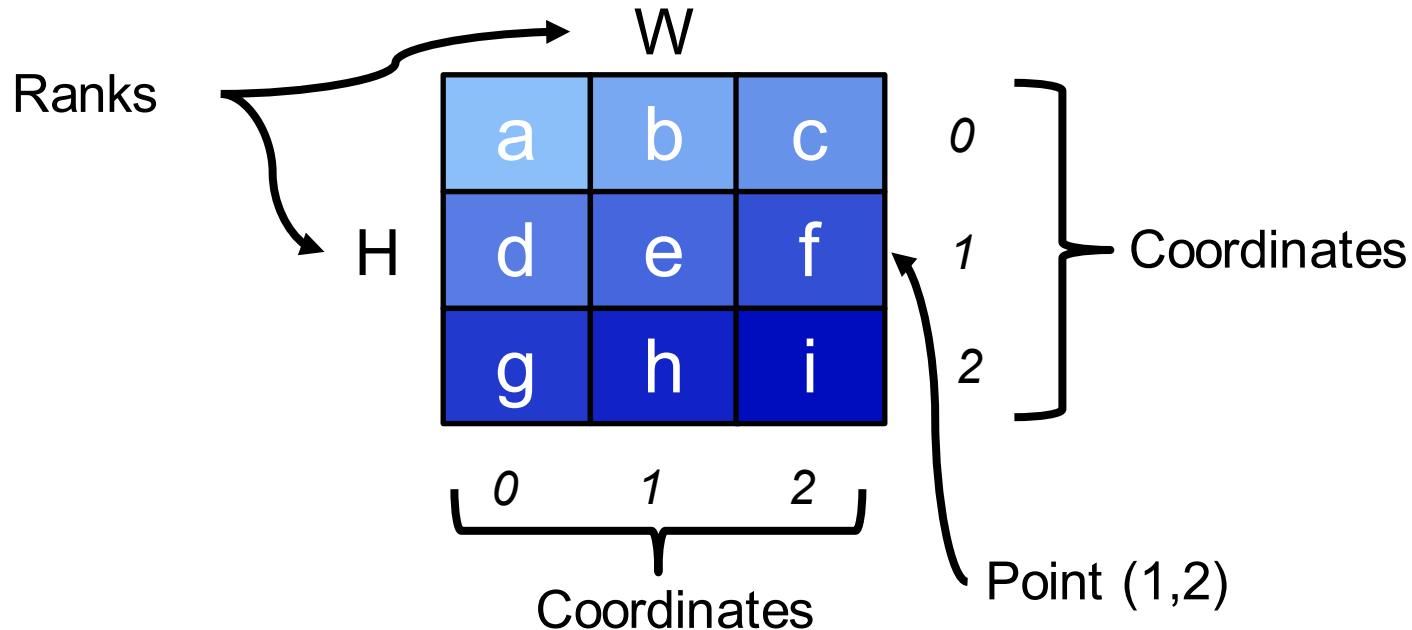
Explicitly eliminate ineffectual storage accesses and computes by letting the hardware unit staying idle for the cycle to save energy

Skipping:



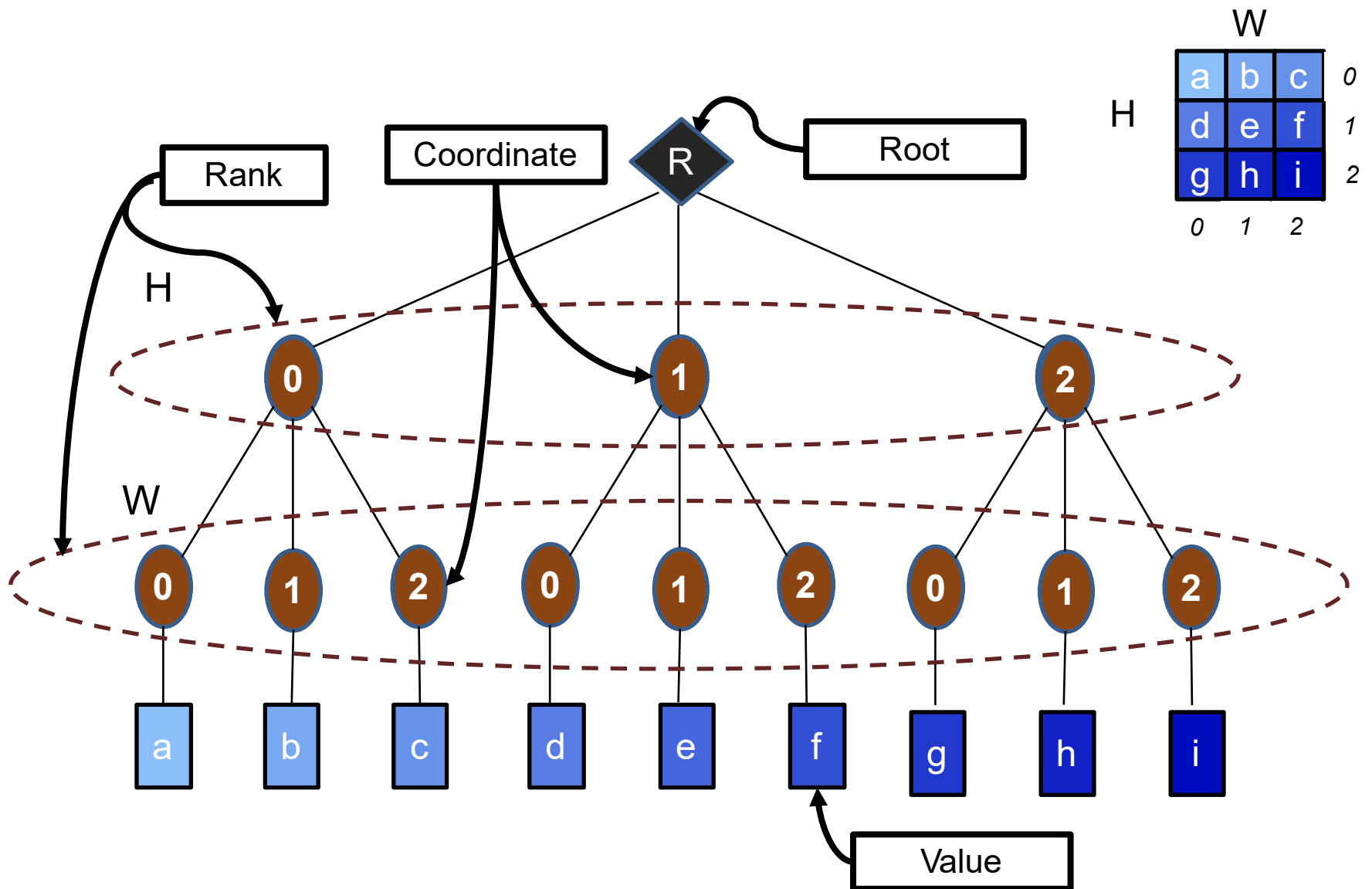
Explicitly eliminate ineffectual storage accesses and computes by skipping the cycle to save energy and time

Tensor Data Terminology



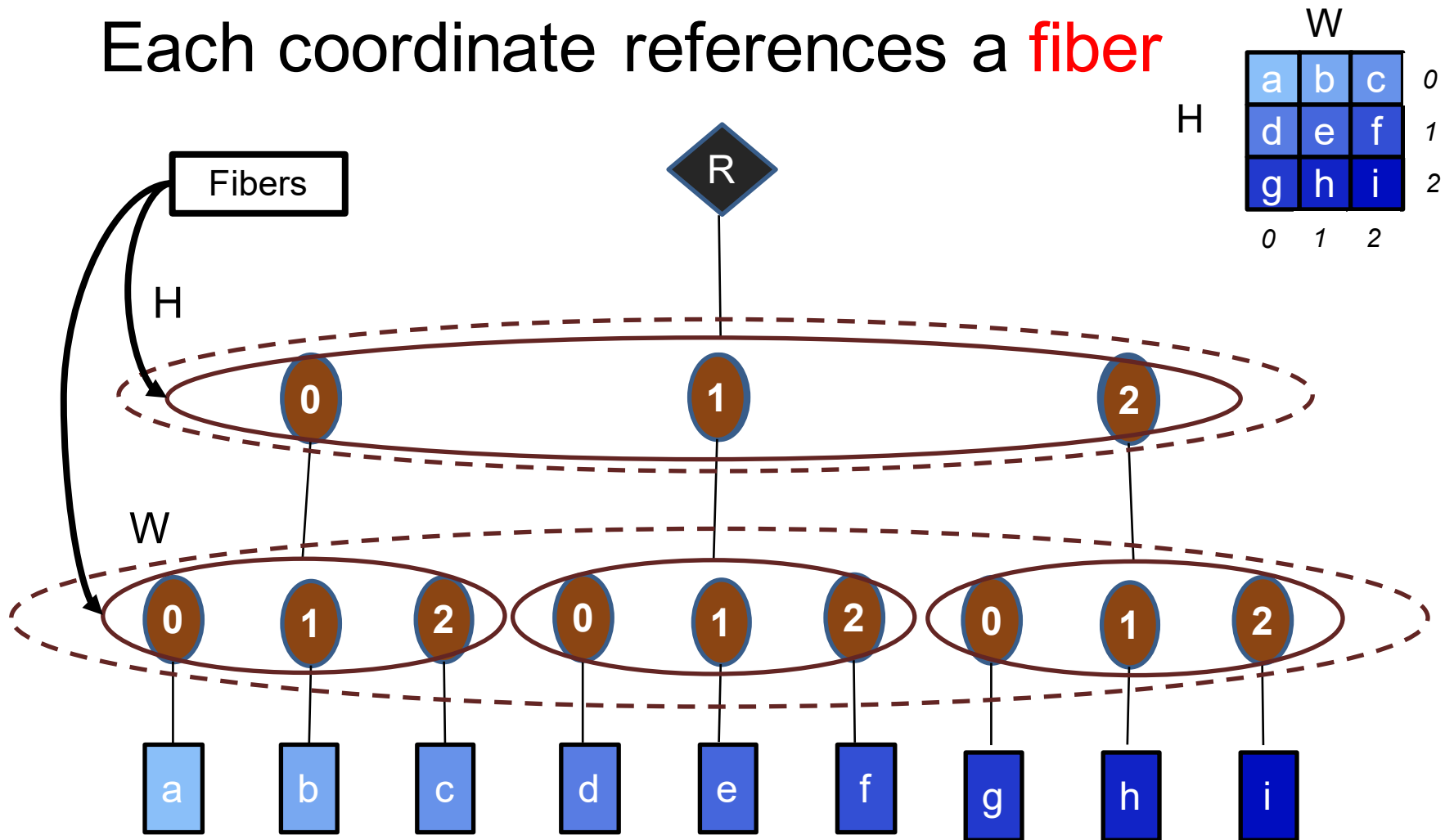
- The elements of each “rank” (dimension) are identified by their “coordinates”, e.g., rank H has coordinates 0, 1, 2
- Each element of the tensor is identified by the tuple of coordinates from each of its ranks, i.e., a “point”.
So (1,2) -> “f”

Tree-based Tensor Abstraction



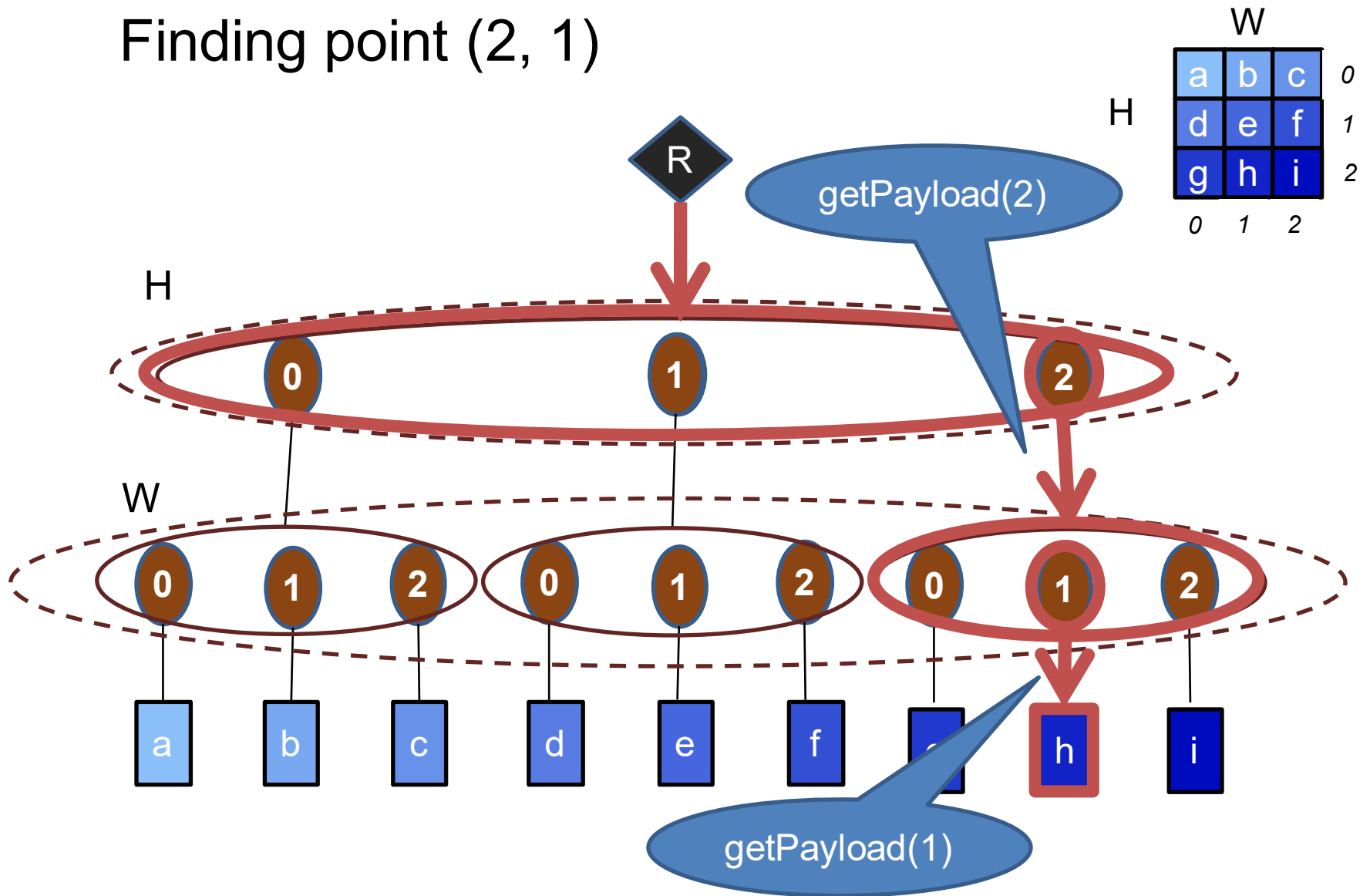
Fibertree Tensor Abstraction

Each coordinate references a **fiber**



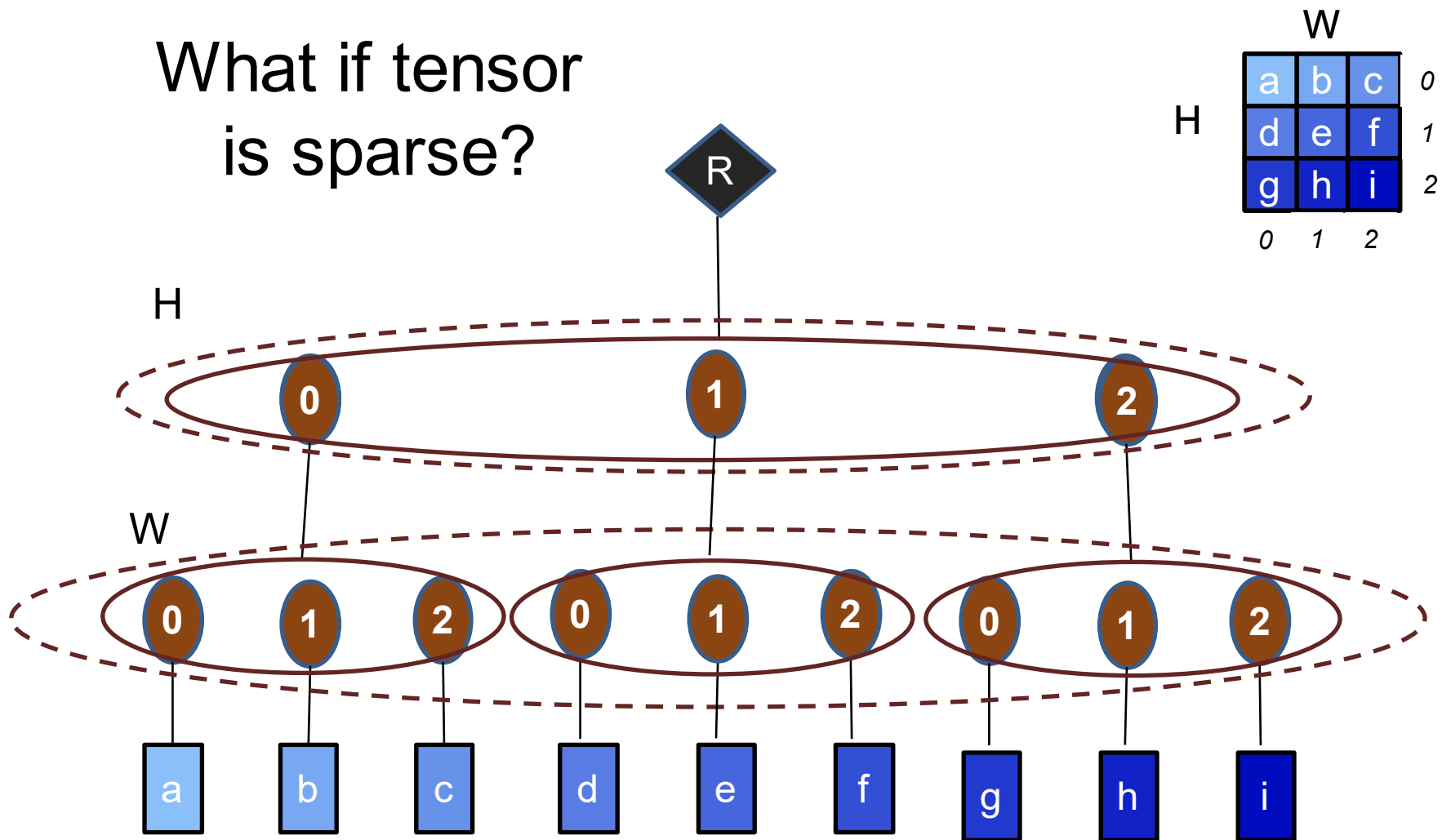
Fibertree Tensor Abstraction

Finding point (2, 1)



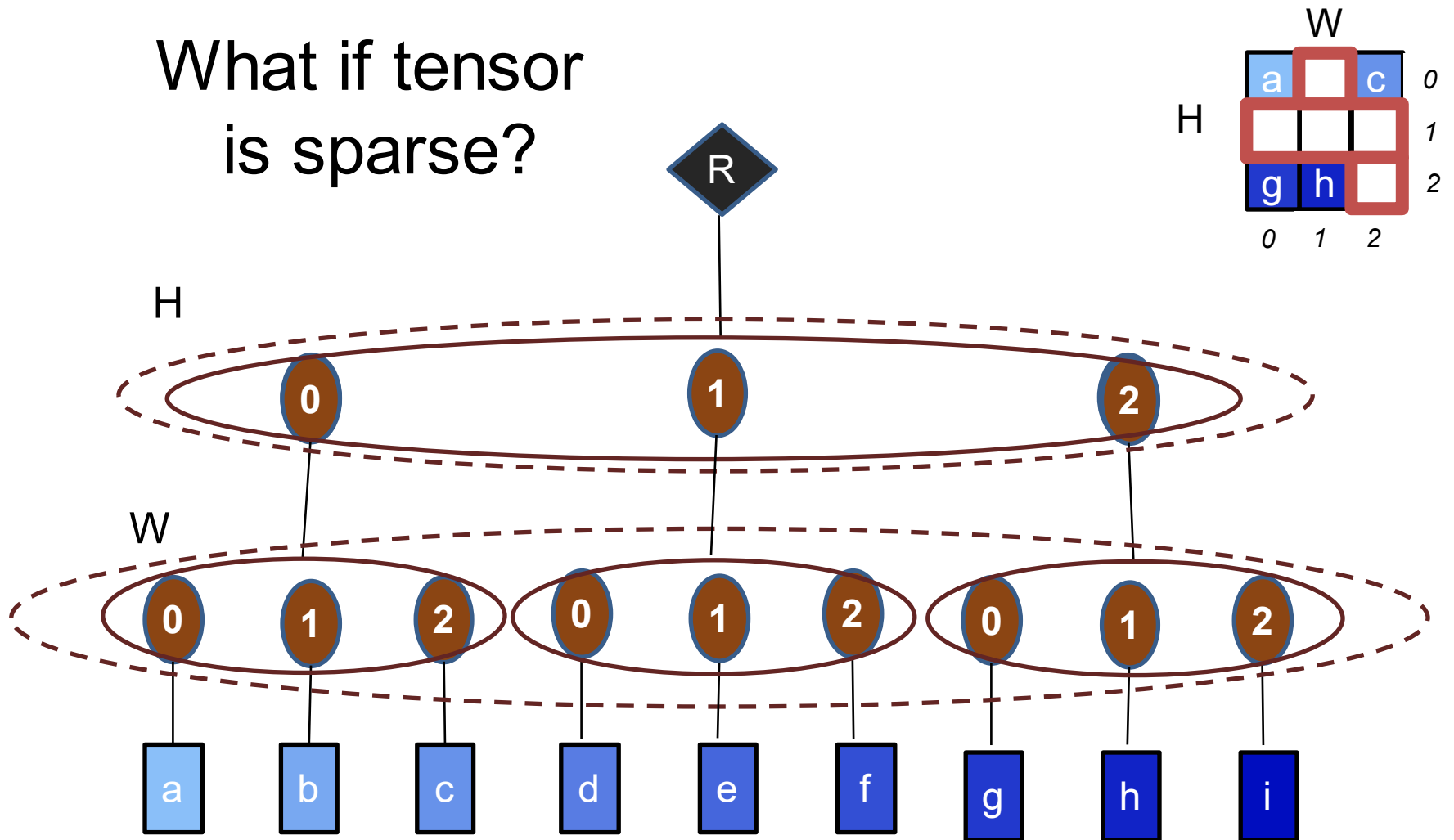
Fibertree Tensor Abstraction

What if tensor
is sparse?



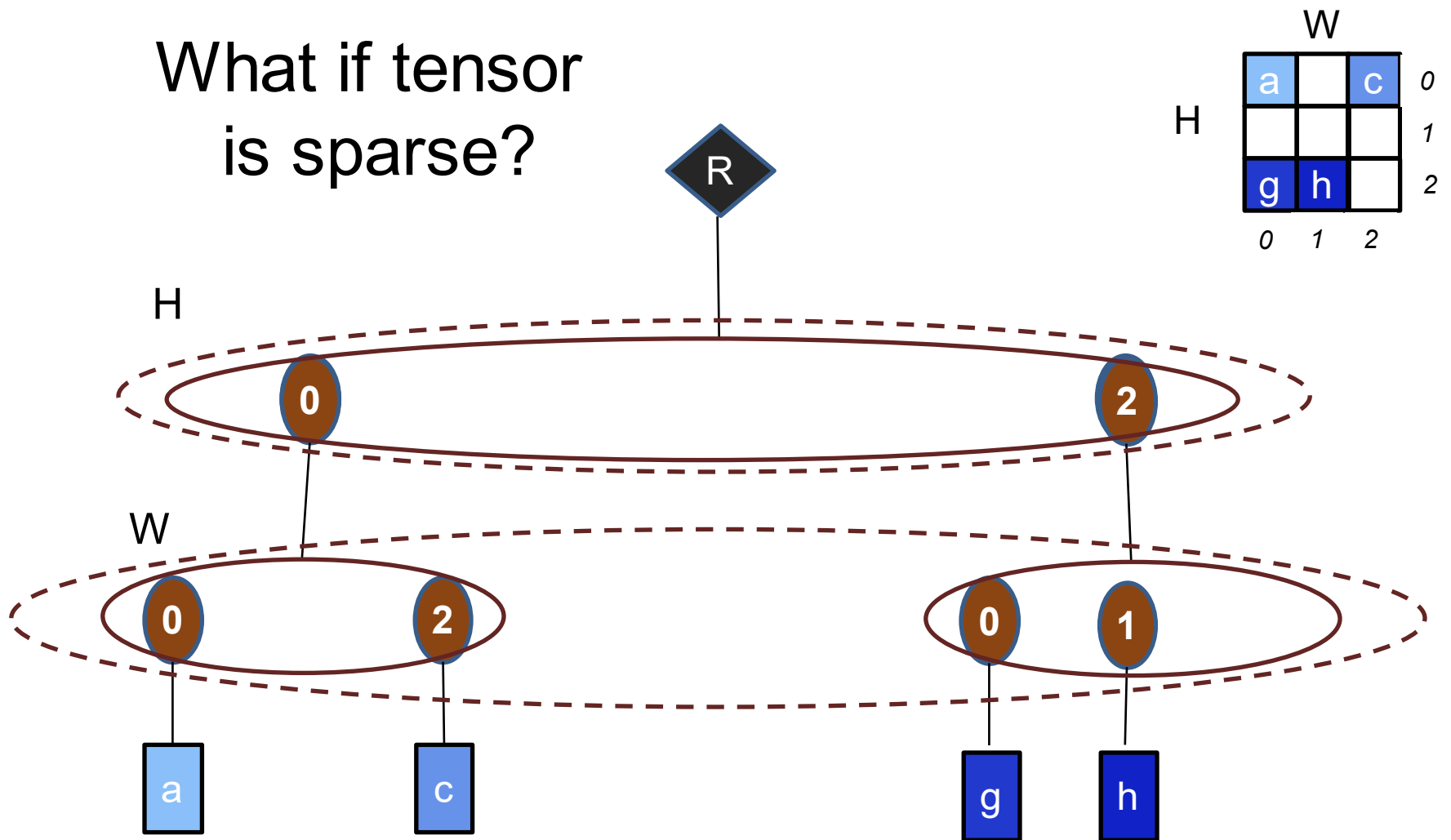
Fibertree Tensor Abstraction

What if tensor
is sparse?



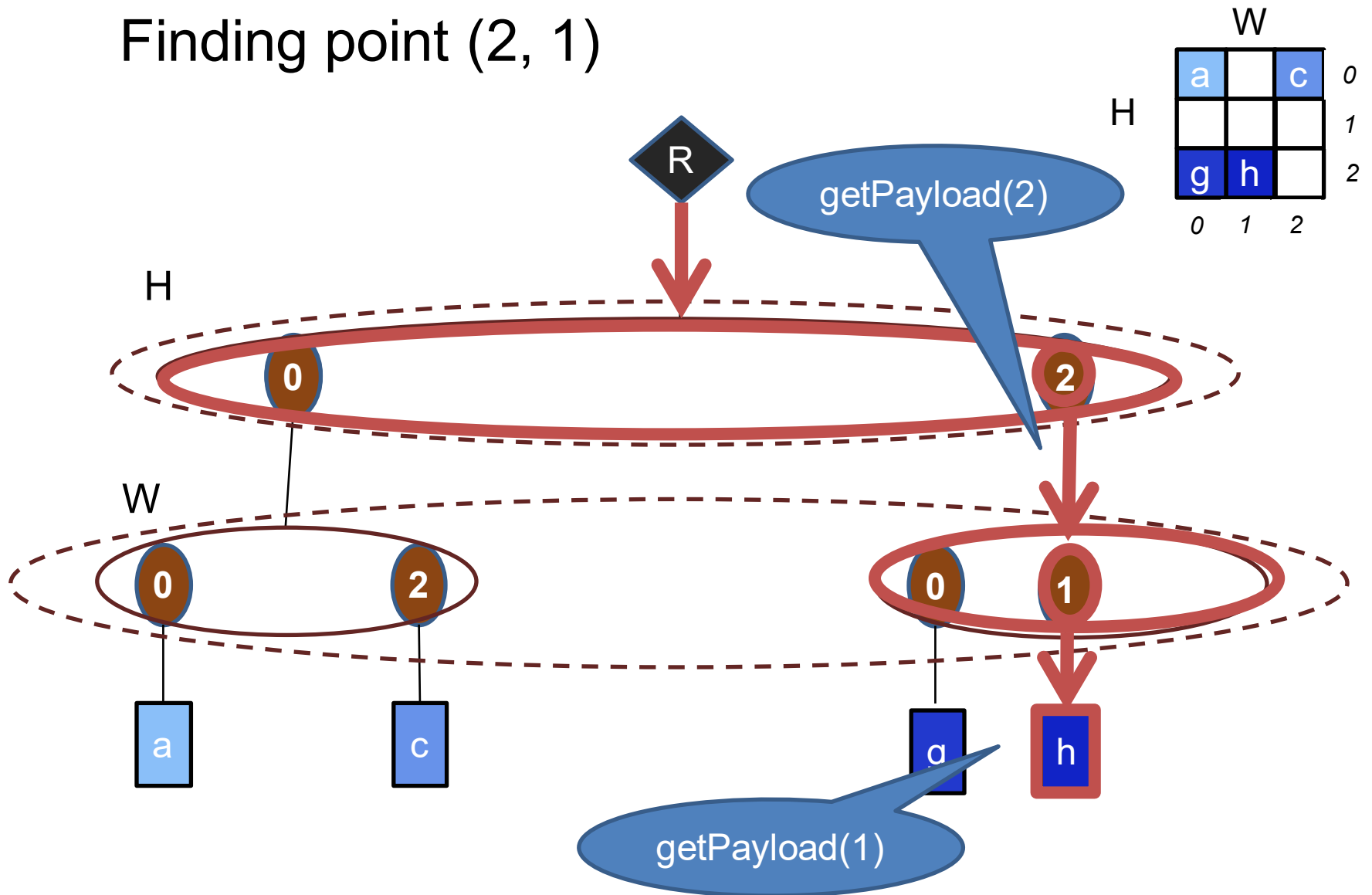
Fibertree Tensor Abstraction

What if tensor
is sparse?



Fibertree Tensor Abstraction

Finding point (2, 1)



Tensor Traversal (2-D)

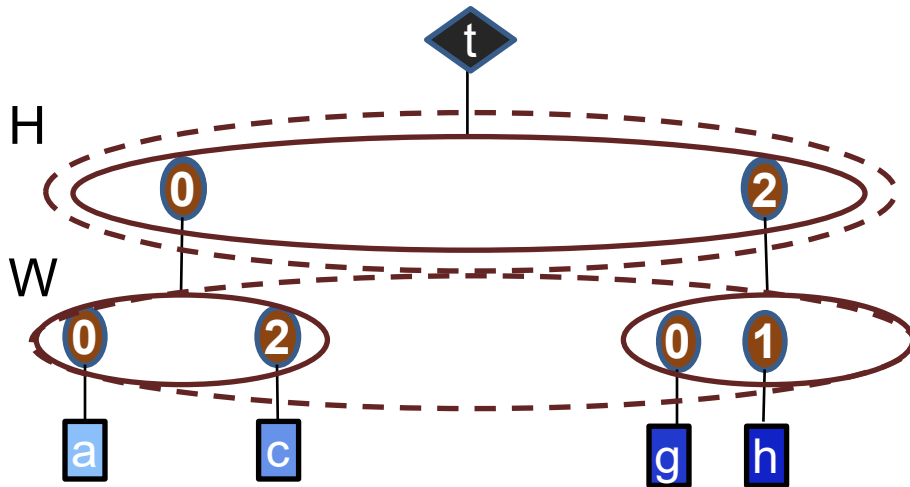
```
# 2-D Tensor Traversal
```

```
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	?	?	?
...	...	...	...	...

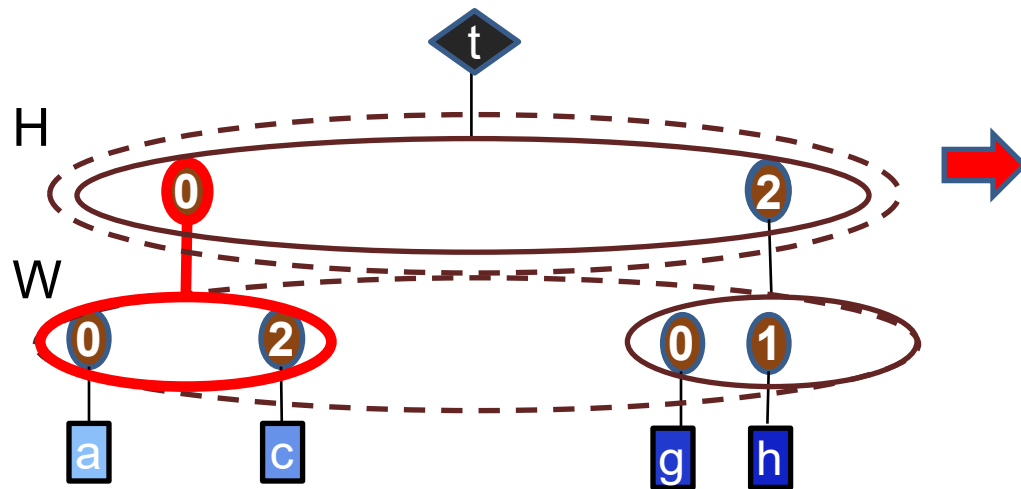
Tensor Traversal (2-D)

```
# 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
```

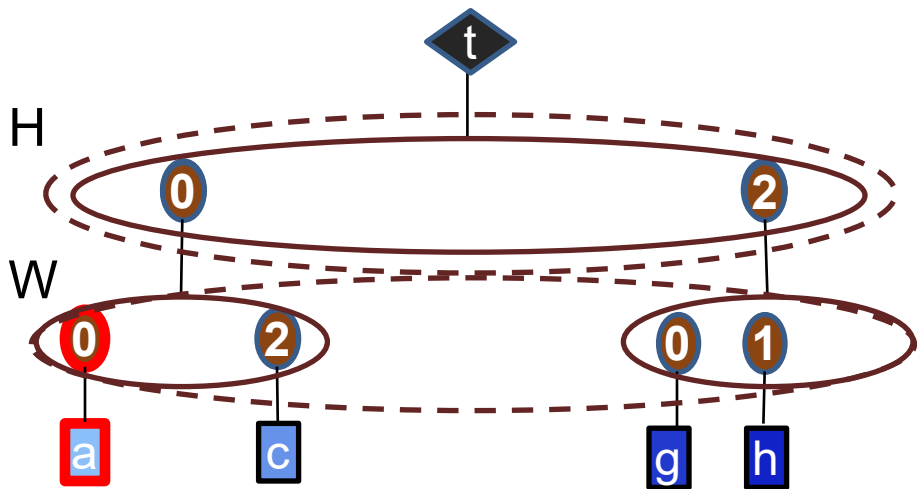
```
t = Tensor(H,W)
```

```
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
```

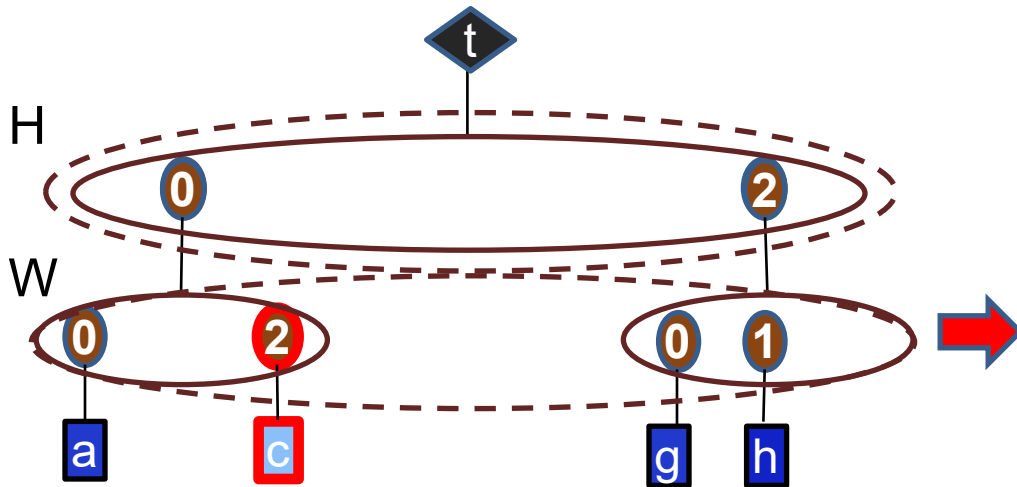
```
t = Tensor(H,W)
```

```
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	?	?	?
...	...	...	...	...

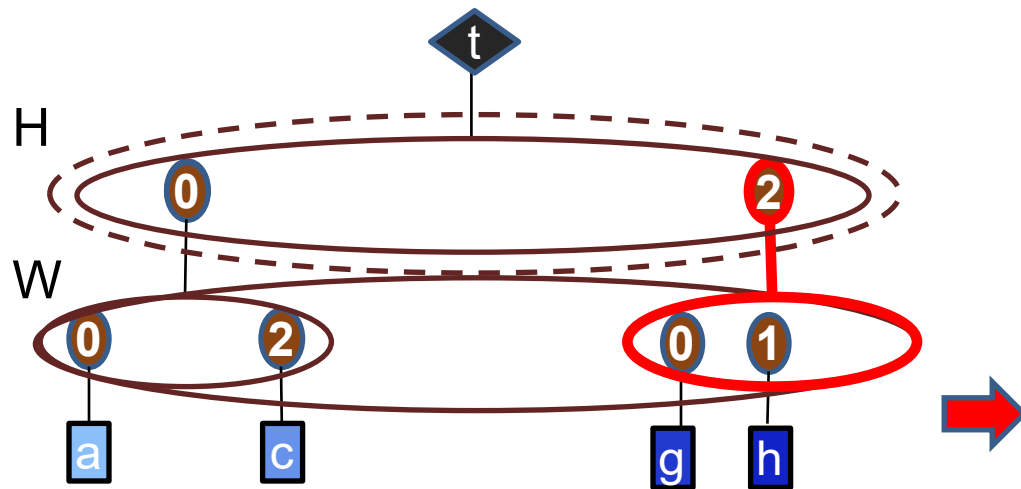
Tensor Traversal (2-D)

```
# 2-D Tensor Traversal
```

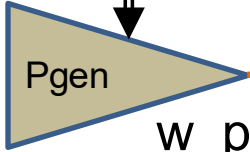
```
t = Tensor(H,W)
```

```
sum = 0
```

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



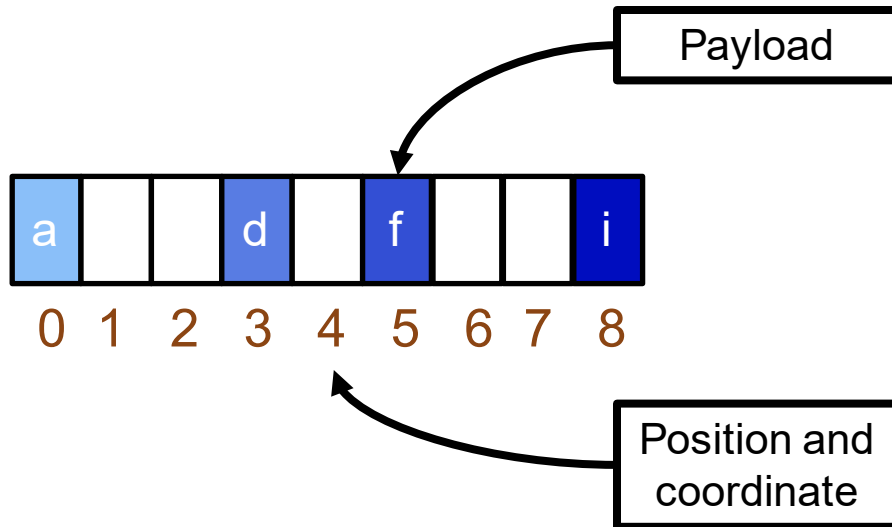
t_pos	h	t_h_pos	w	t_val
0	0	?	?	?
0	0	0	0	a
0	0	1	2	c
1	2	?	?	?
...	...	...	...	...



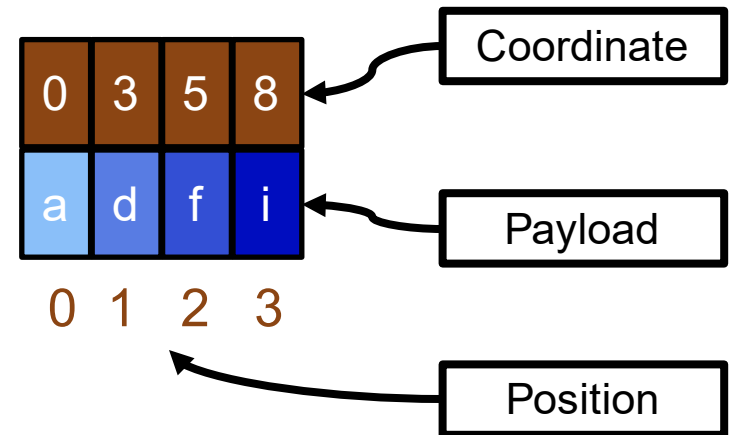
Example Fiber Representations

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

Array



Coordinate/Payload List



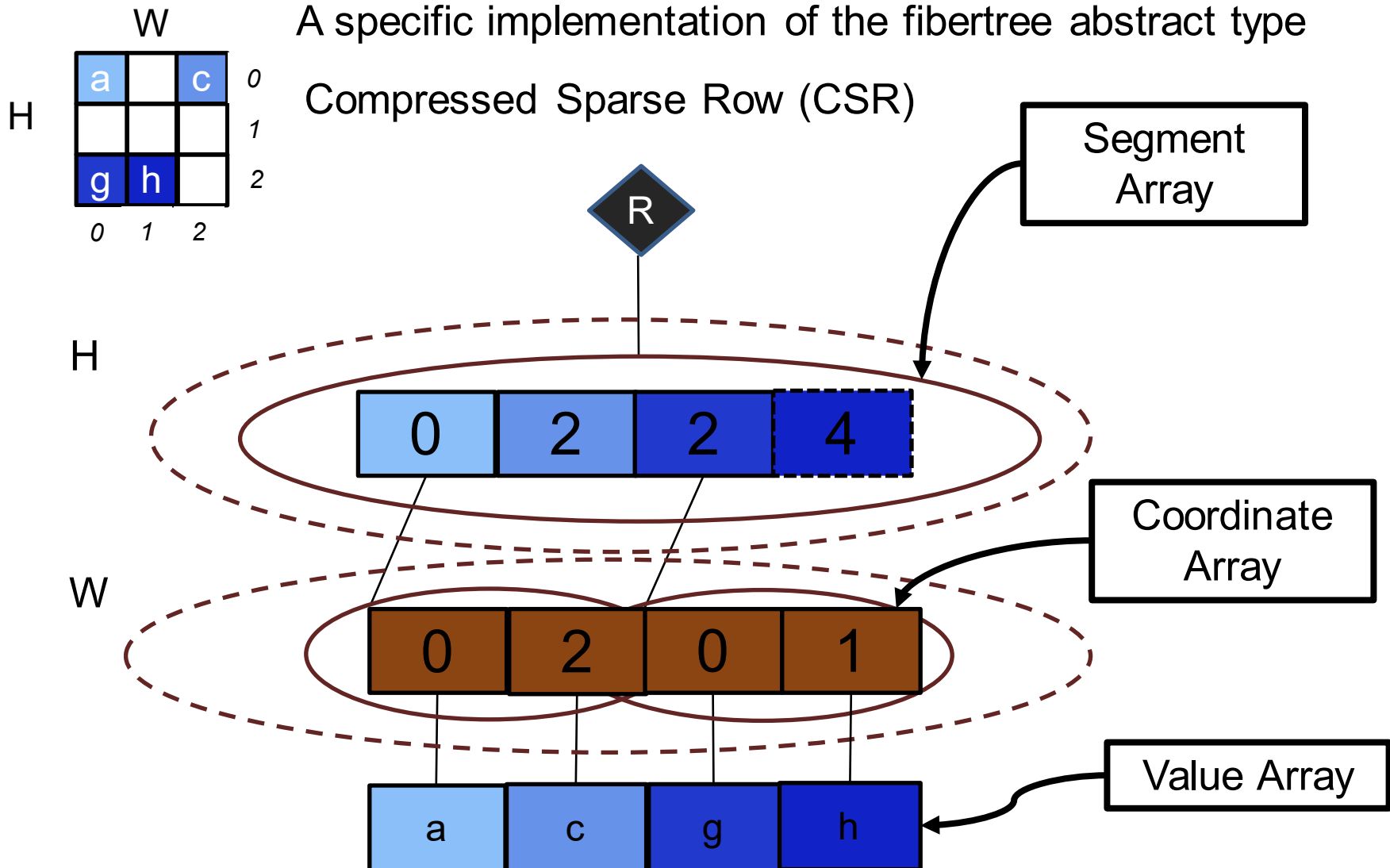
Data in a fiber is accessed by its **position** or offset in memory

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



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

CONV: Exploiting Sparse Weights

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

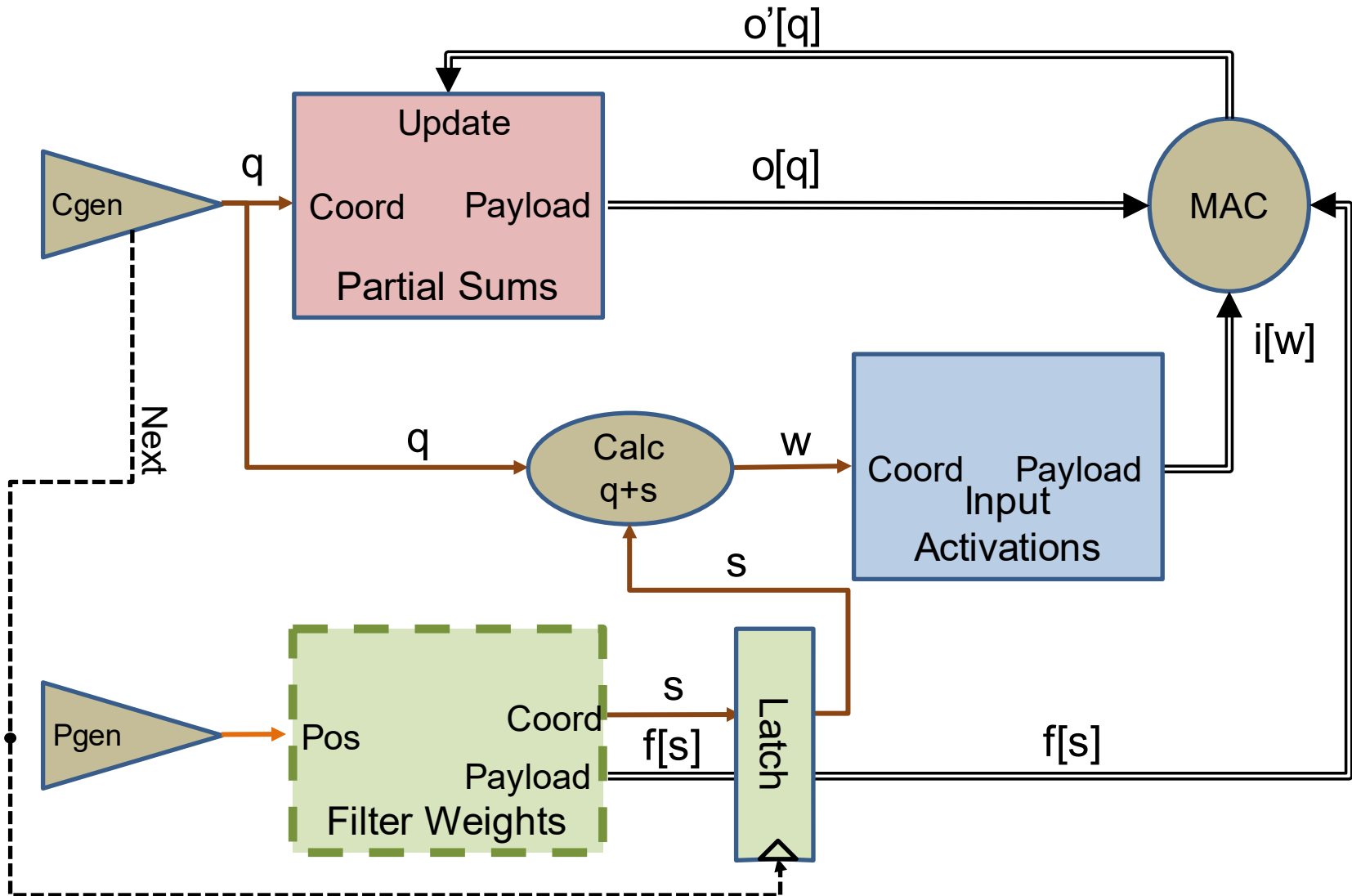
Weight Stationary - Sparse Weights

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

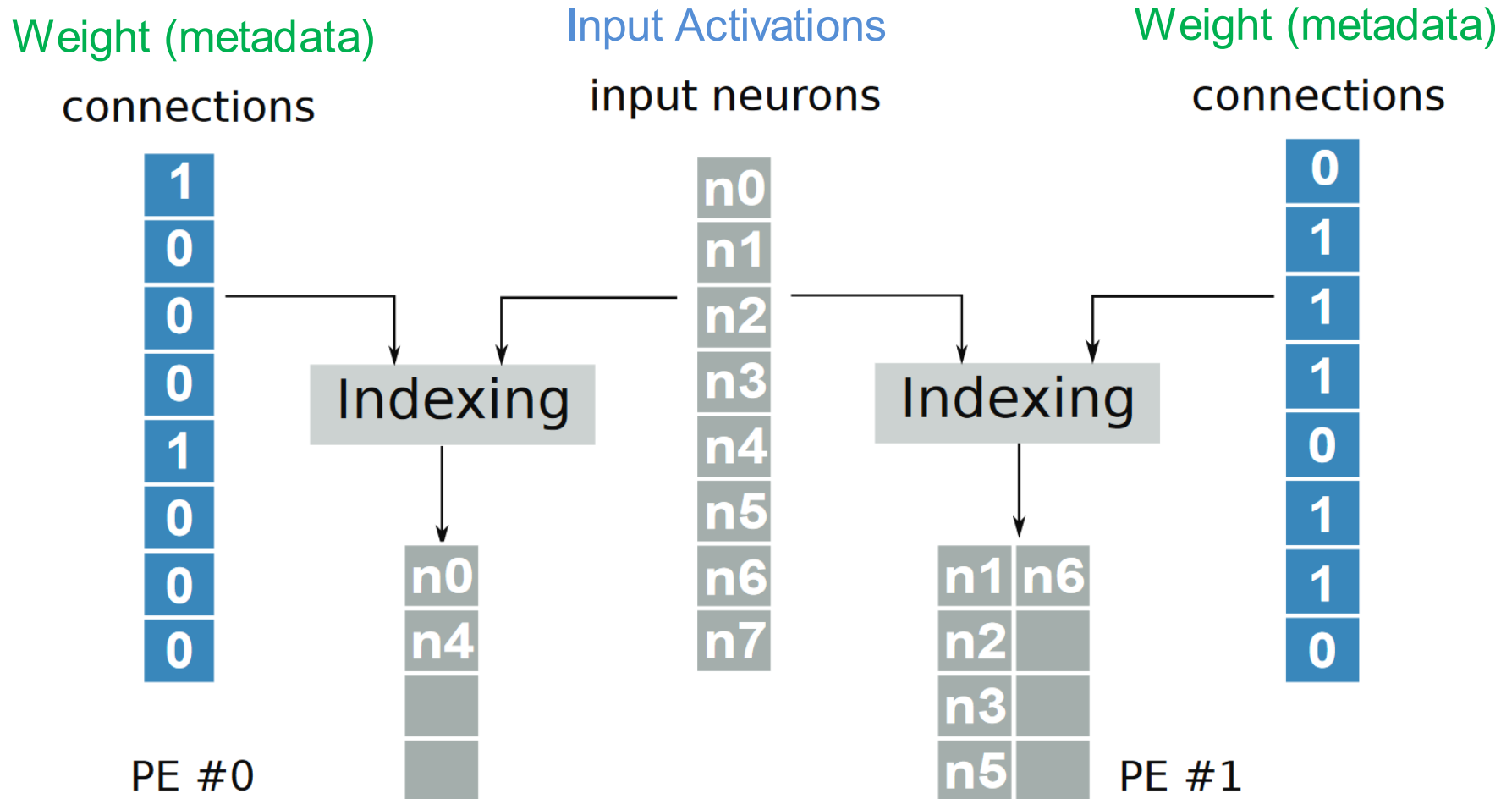
for (s, f_val) in f:
    for q in [0, Q):
        w = q + s
        o[q] += i[w] * f_val
```

Concordant traversal

Weight Stationary - Sparse Weights



Cambricon-X – Activation Access



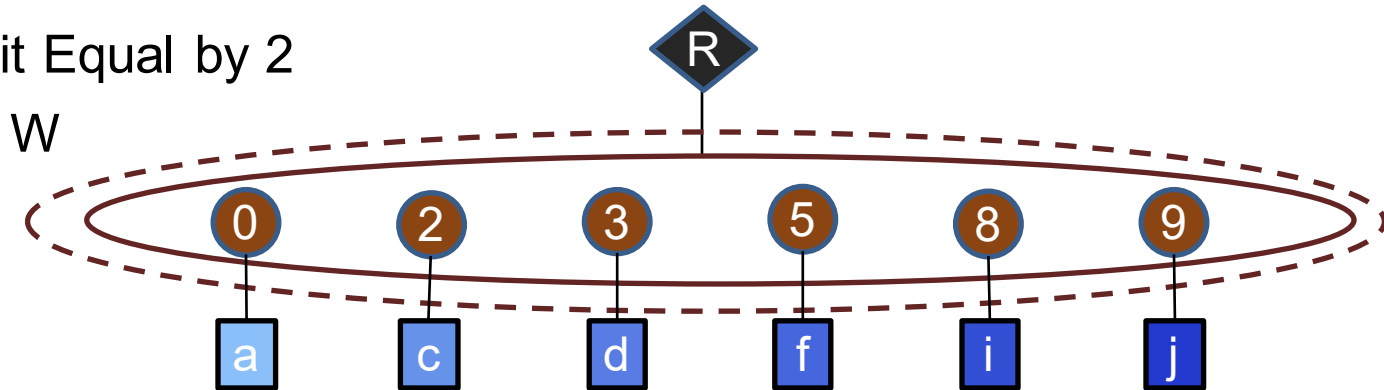
Cambricon-X – Zhang et.al., Micro 2016

To Extend to Other Dimensions of DNN

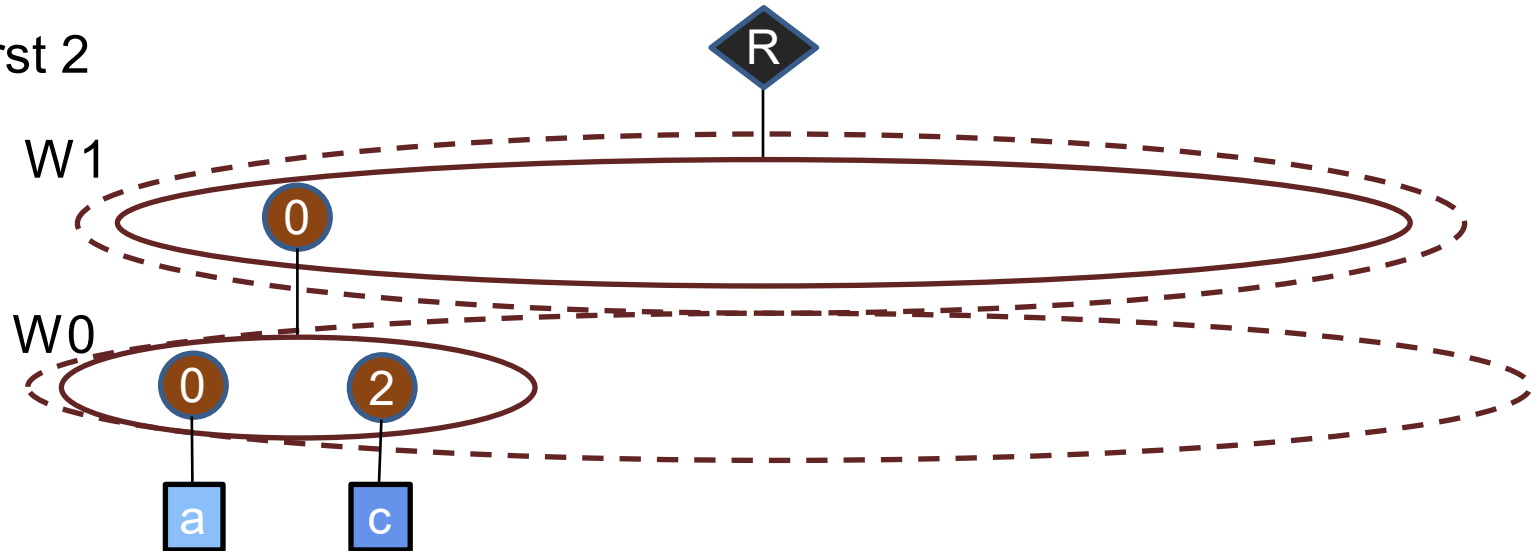
- **Need to add loop nests for:**
 - **2-D input activations and filters**
 - **Multiple input channels**
 - **Multiple output channels**
- **Add parallelism...**

Fiber Splitting Equally in Position Space

Before Split Equal by 2

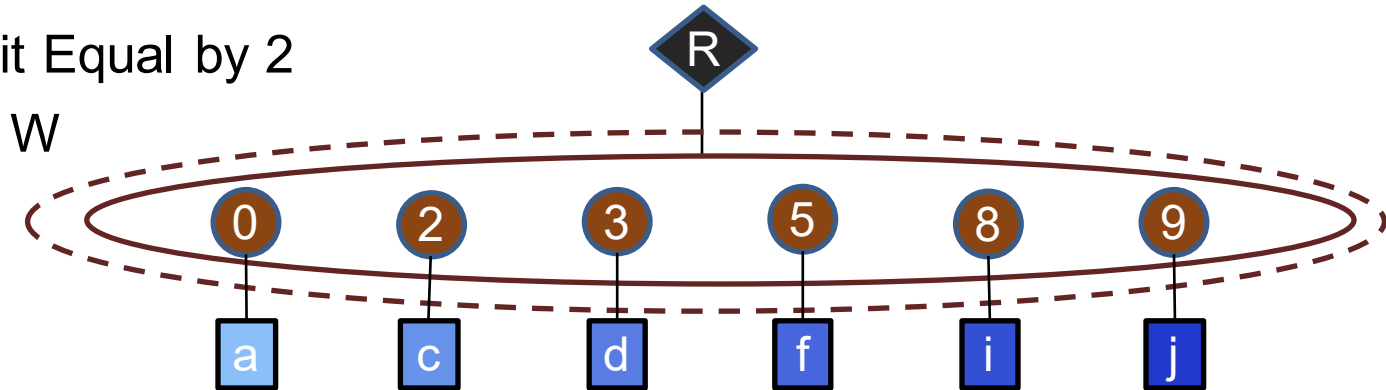


Grab first 2

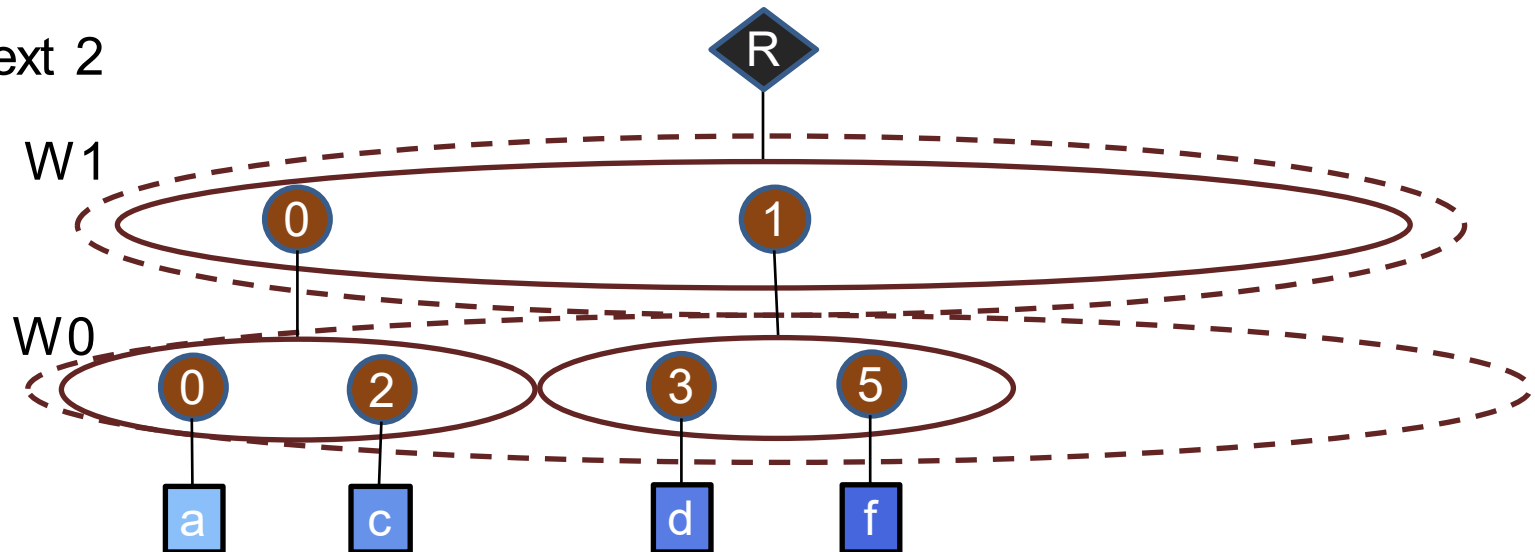


Fiber Splitting Equally in Position Space

Before Split Equal by 2

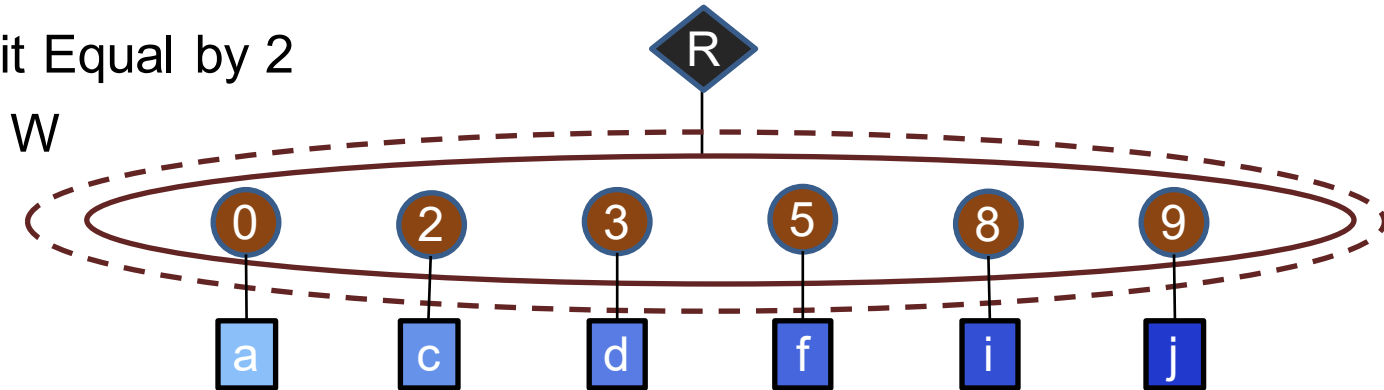


Grab next 2

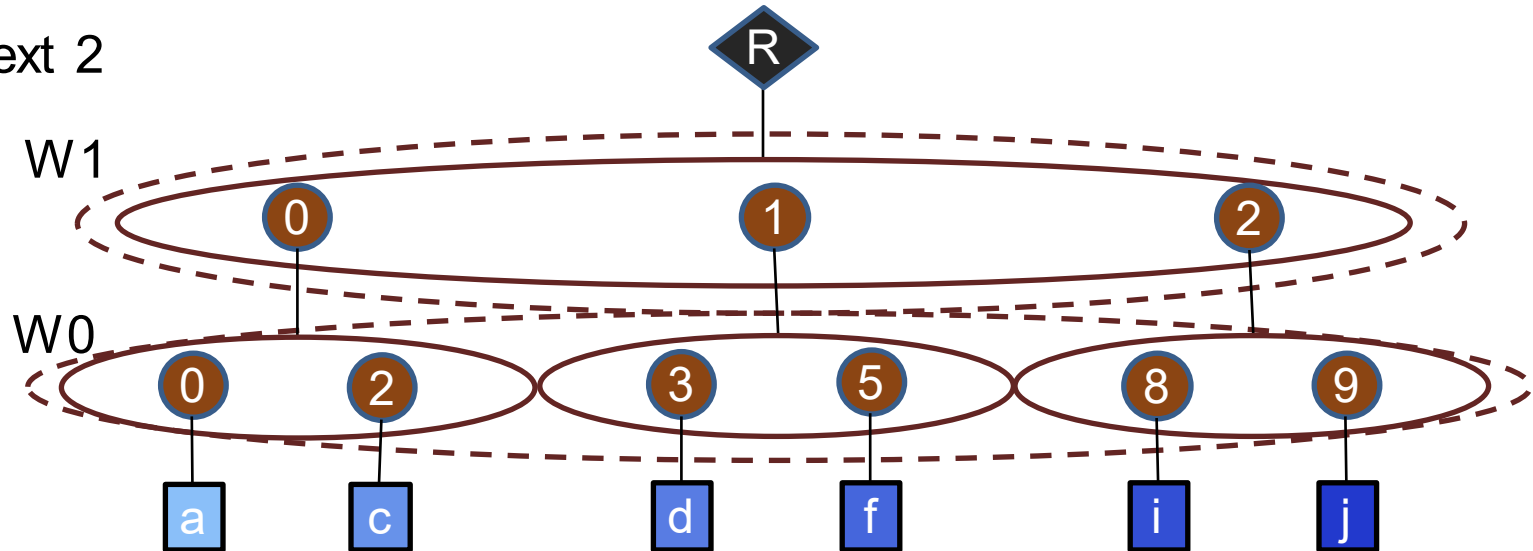


Fiber Splitting Equally in Position Space

Before Split Equal by 2

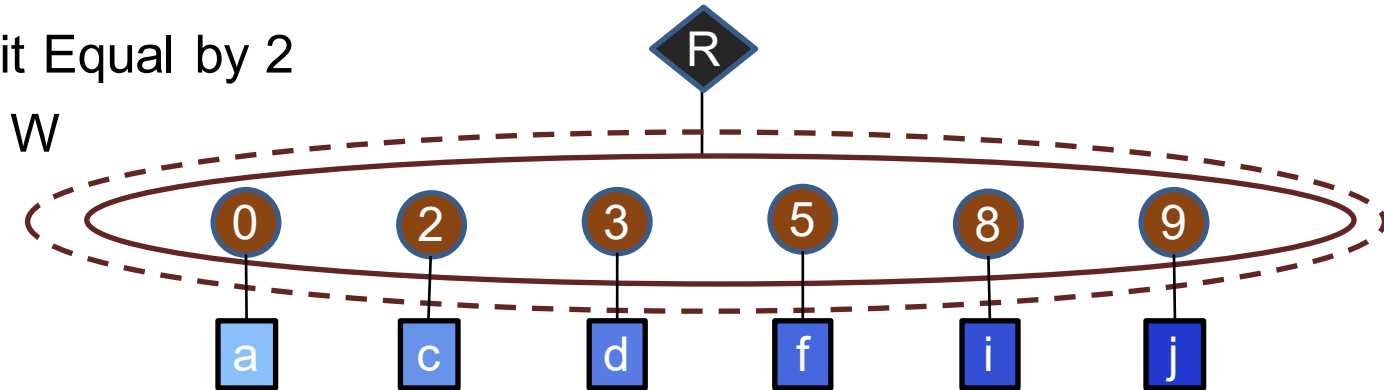


Grab next 2

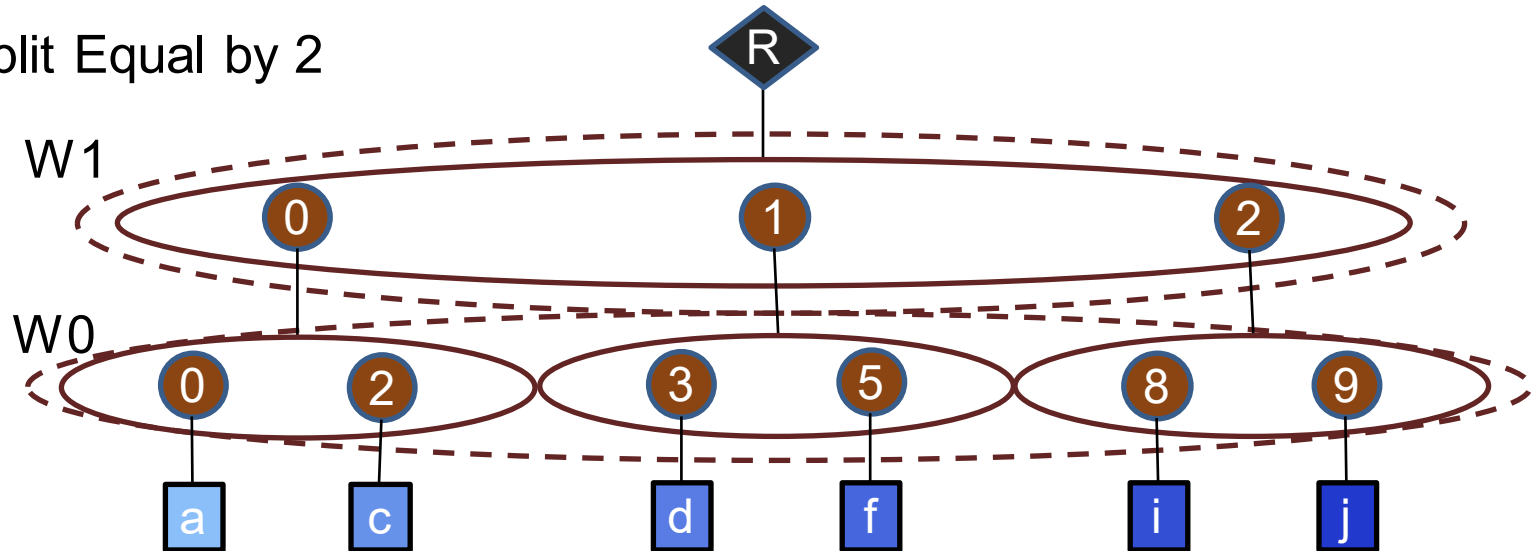


Fiber Splitting Equally in Position Space

Before Split Equal by 2



After Split Equal by 2



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

Accumulate multiple outputs each spatially

Look up input activation

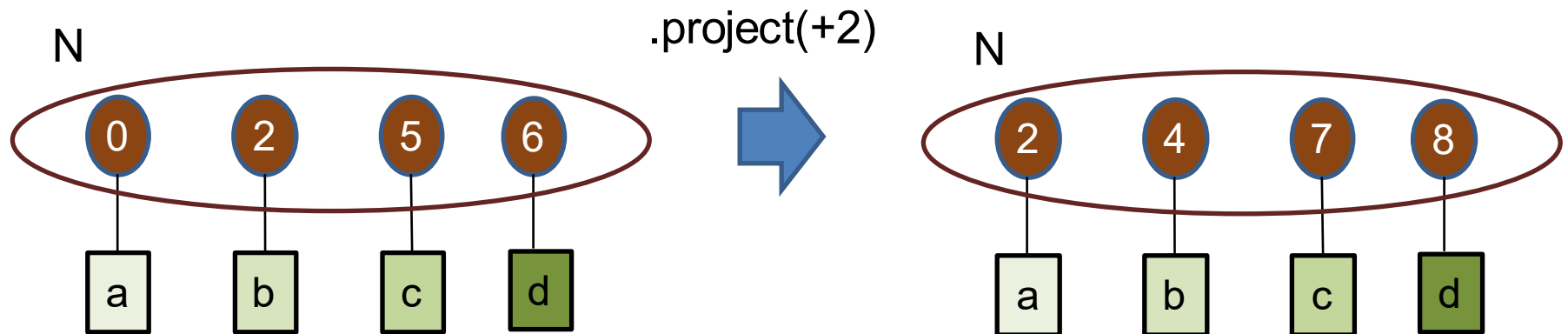
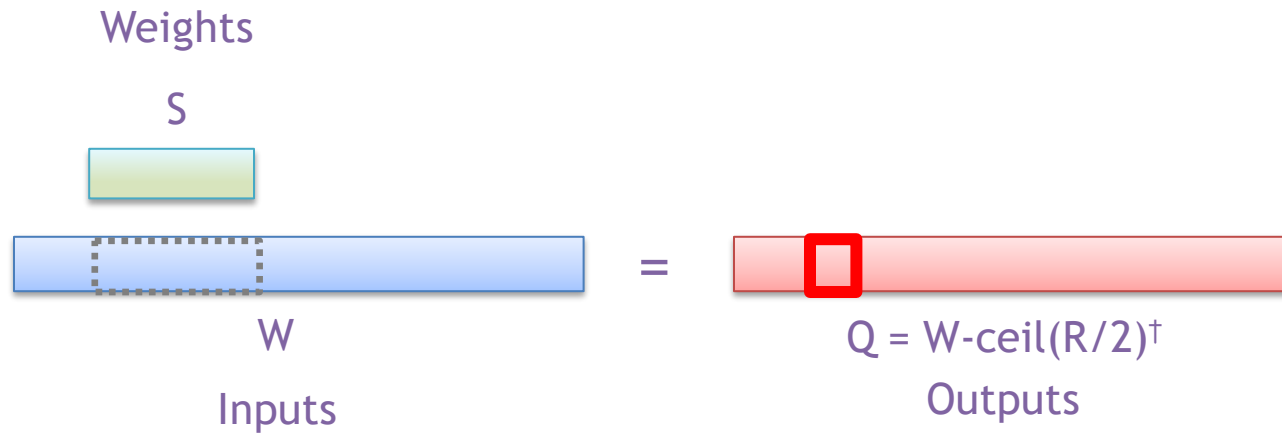
CONV: Exploiting Sparse Inputs & Sparse Weights

Output Stationary - Sparse Weights & Inputs

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

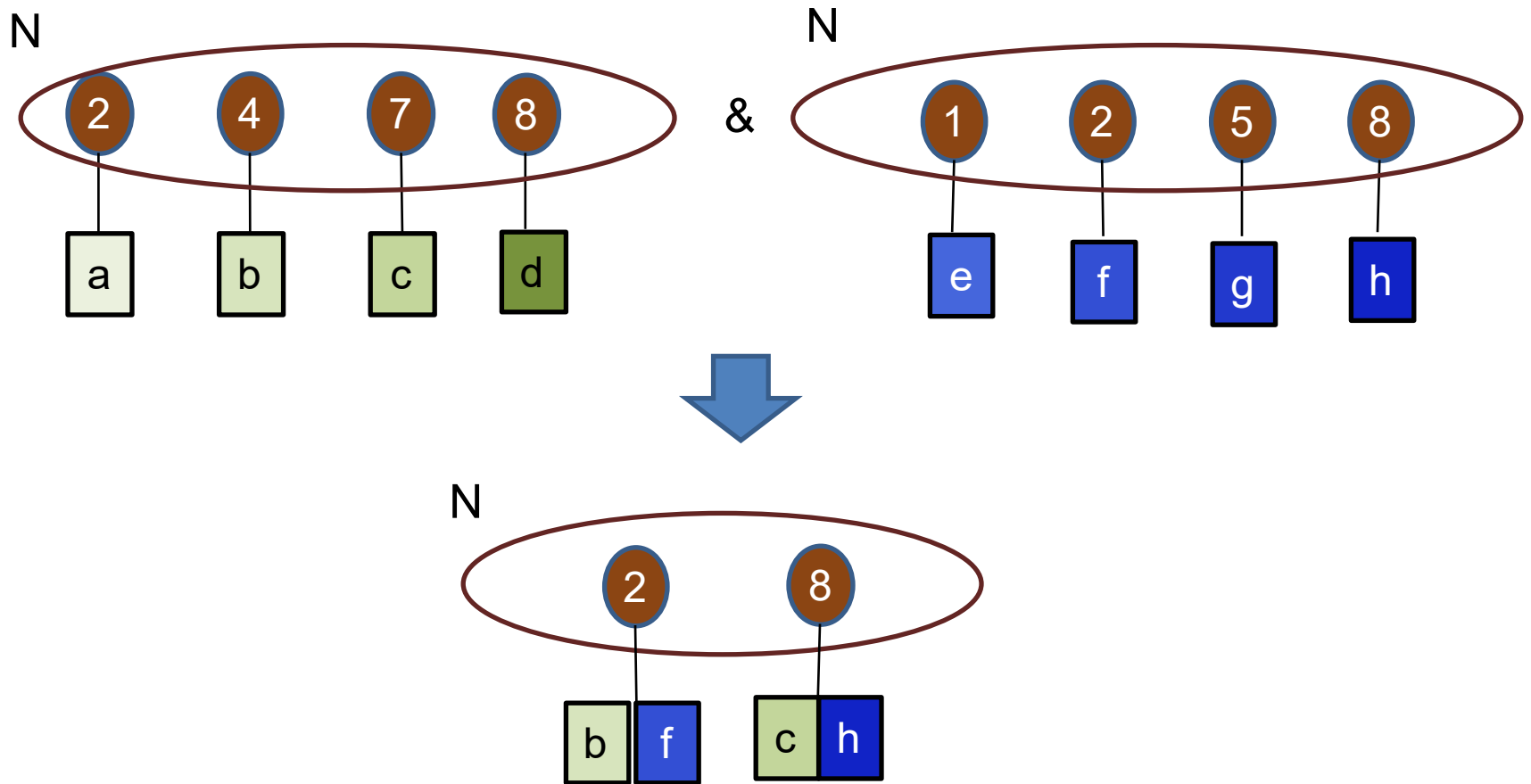
for q in [0,Q):
    for (s, (f_val, i_val)) in f.project(+q) & i:
        o[q] += i_val * f_val
```

Fiber Coordinate Projection

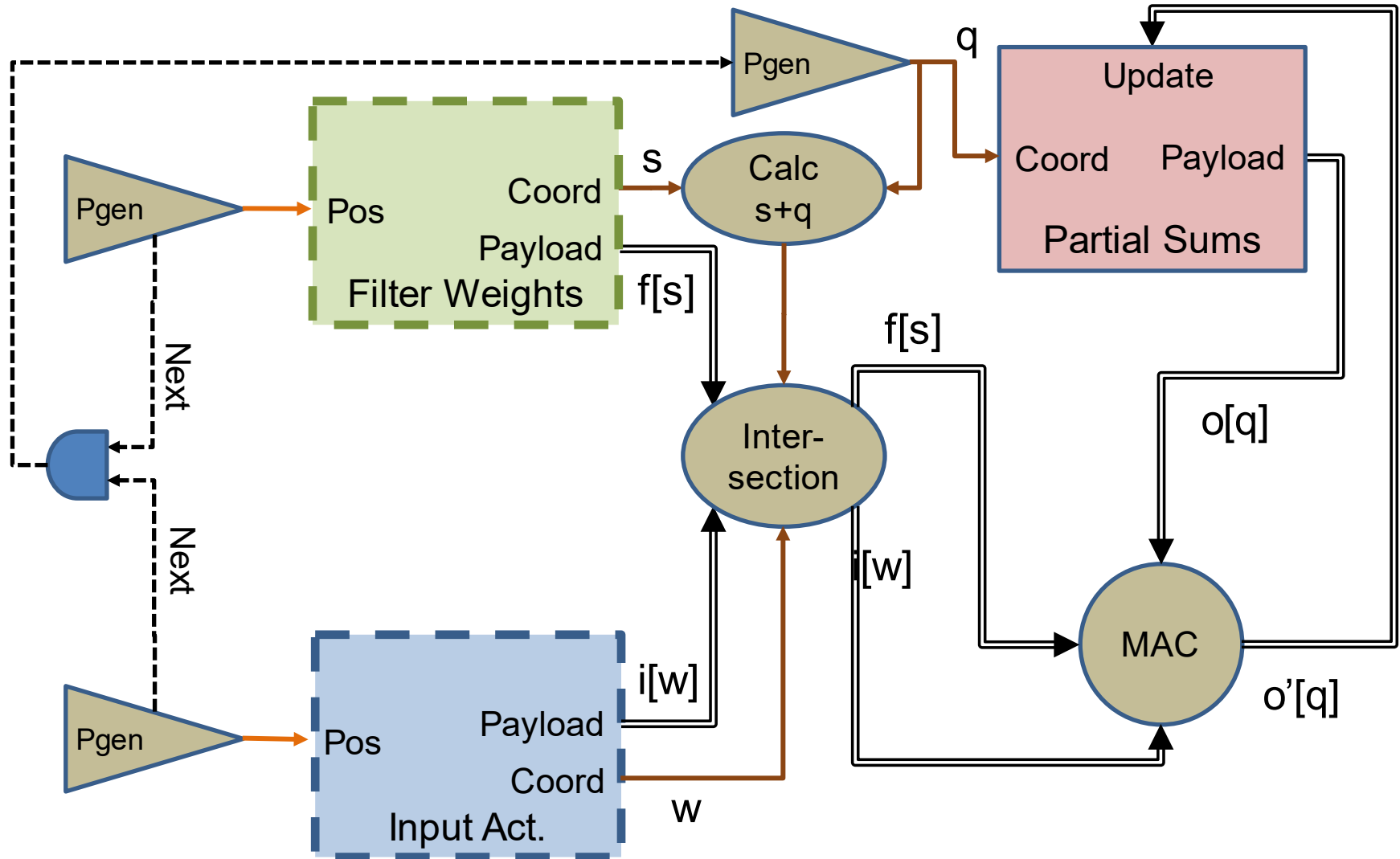


fiber-projection

Fiber Intersection

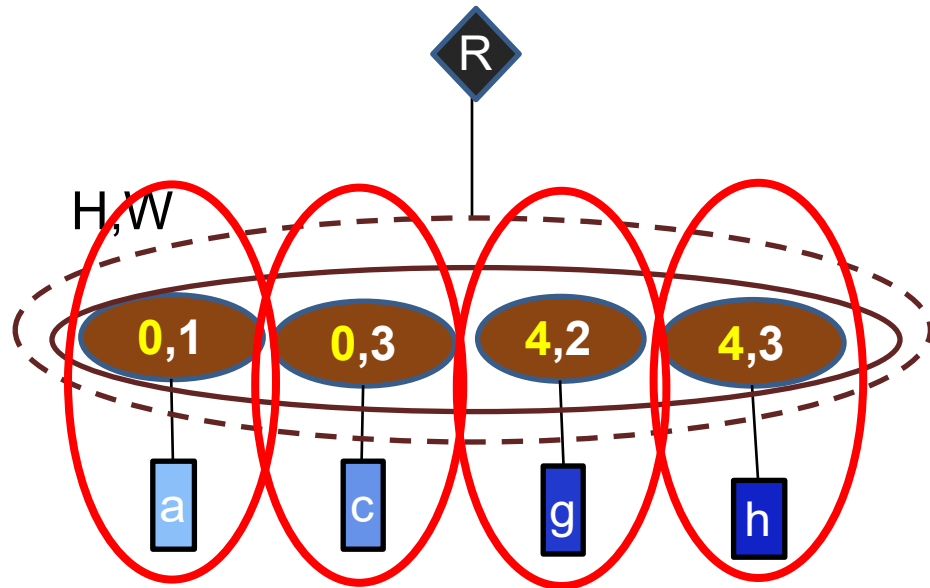
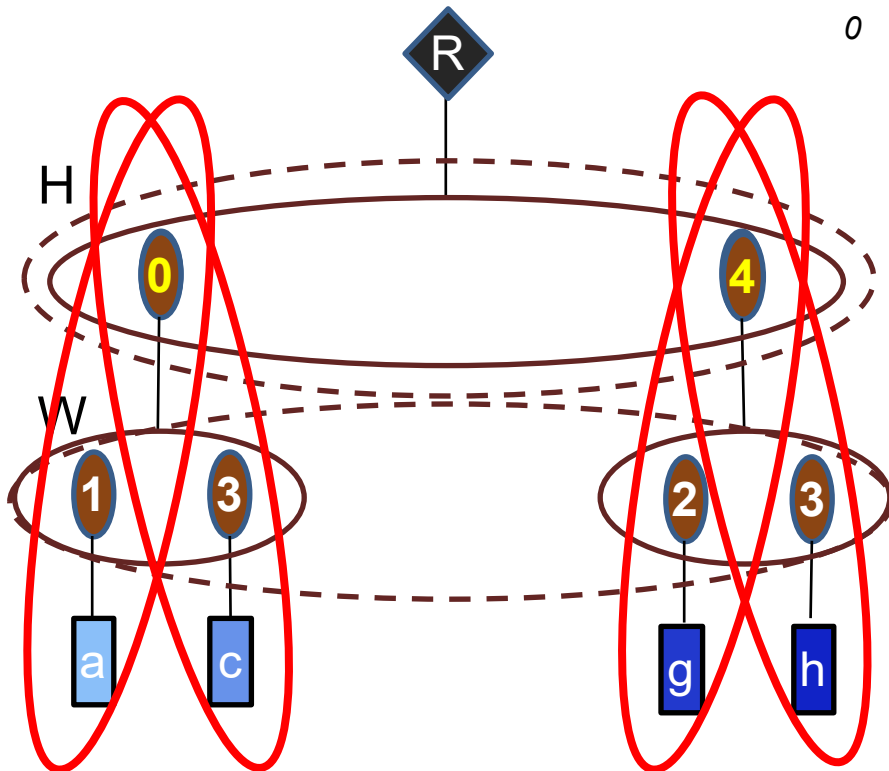
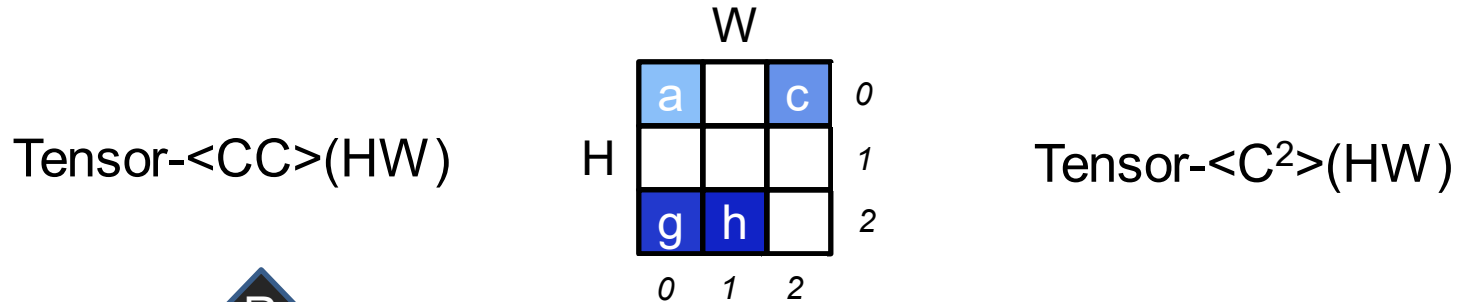


Output Stationary - Sparse Weights & Inputs



Flattening Ranks

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



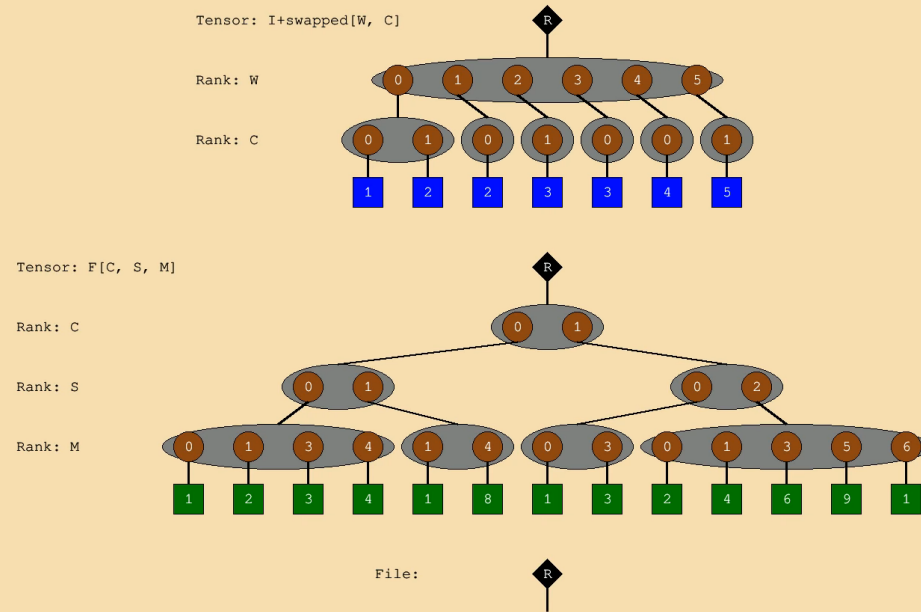
Row Stationary – Sparse Inputs & Activations

```
i = Tensor(CW)      # Input activations (CW flattened)
f = Tensor(C, SM)    # Filter weights (SM flattened)
o = Array(M, Q)      # Output activations

for ((c, w), i_val) in i:
    f_c = f.getPayload(c)
    f_c_split = f_c.splitEven(2)
    parallel-for (_, f_sm) in f_c_split:
        for ((s, m), f_val) in f_sm if w-Q <= s < w:
            q = w - s
            o[m, q] += i_val * f_val
```

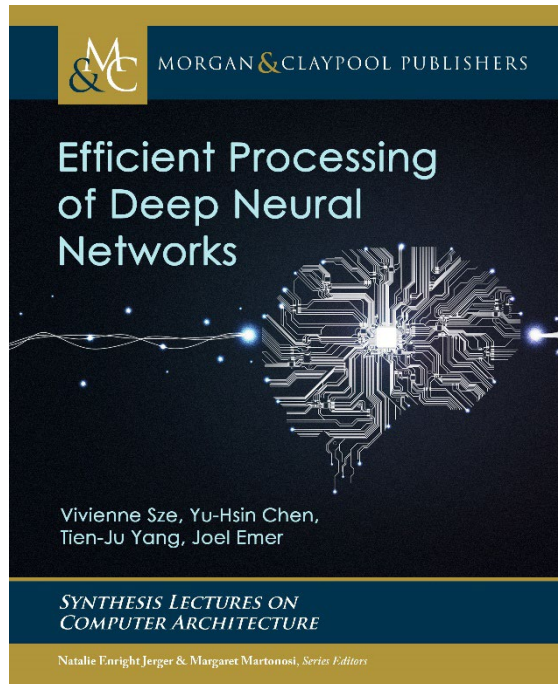
Eyeriss V2 – Chen et.al., JETCAS 2018

Row Stationary – Sparse Inputs & Activations



Eyeriss V2 – Chen et.al., JETCAS 2018

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 2023

Thank you!

*Next Lecture:
Virtualization and Security*