6.5930/1

Hardware Architectures for Deep Learning

Sparse Architectures – Part 1

April 3, 2024

Joel Emer and Vivienne Sze

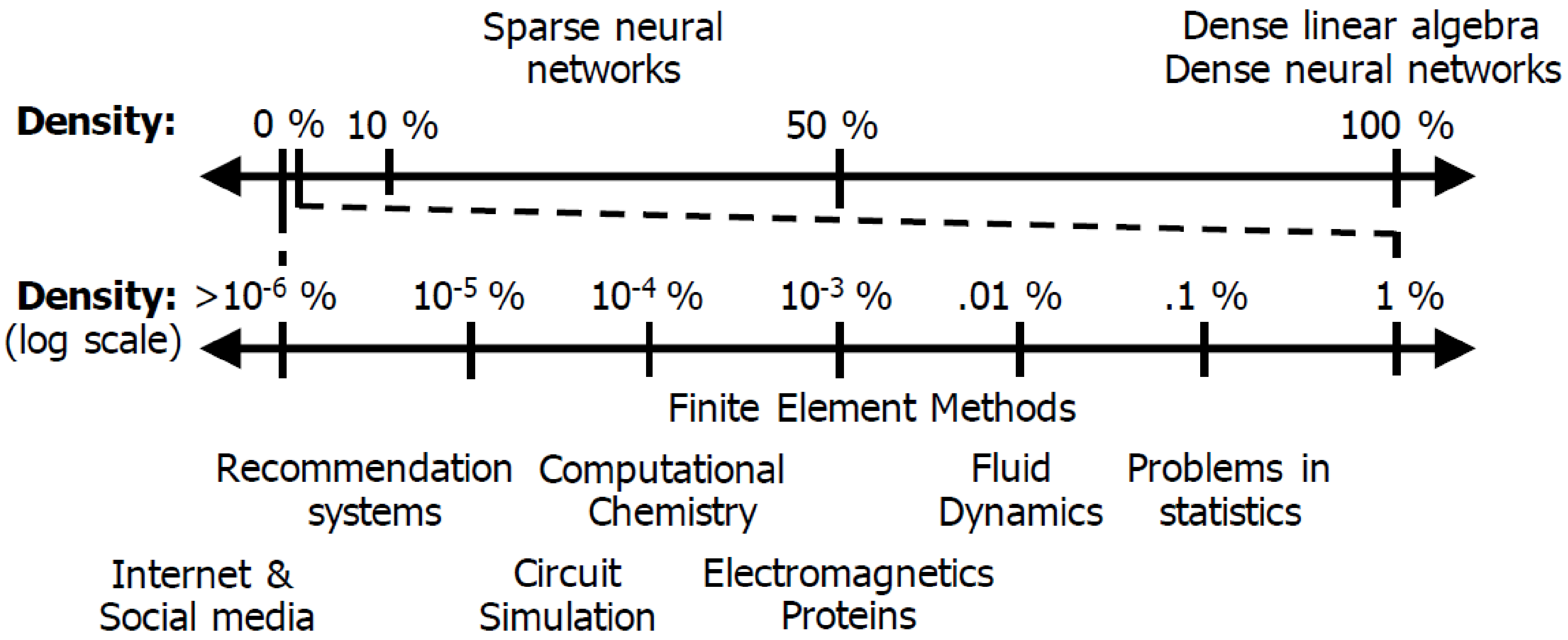
Massachusetts Institute of Technology
Electrical Engineering & Computer Science

Goals of Today's Lecture

- Last lecture, we discussed how to make weights and activations of DNN models sparse
- Sparsity of DNNs on the order of 30-70%, while existing software libraries (e.g., sparse BLAS) designed for >99%
  - Need specialized hardware to exploit!
- Today and in the next lecture, we will discuss how to translate sparsity into reductions in energy consumption and processing cycles
  - First, discuss the representation of sparse data
  - Second, present some architectures that exploit sparsity

Resources: Course notes - Chapter 8.2 and 8.3

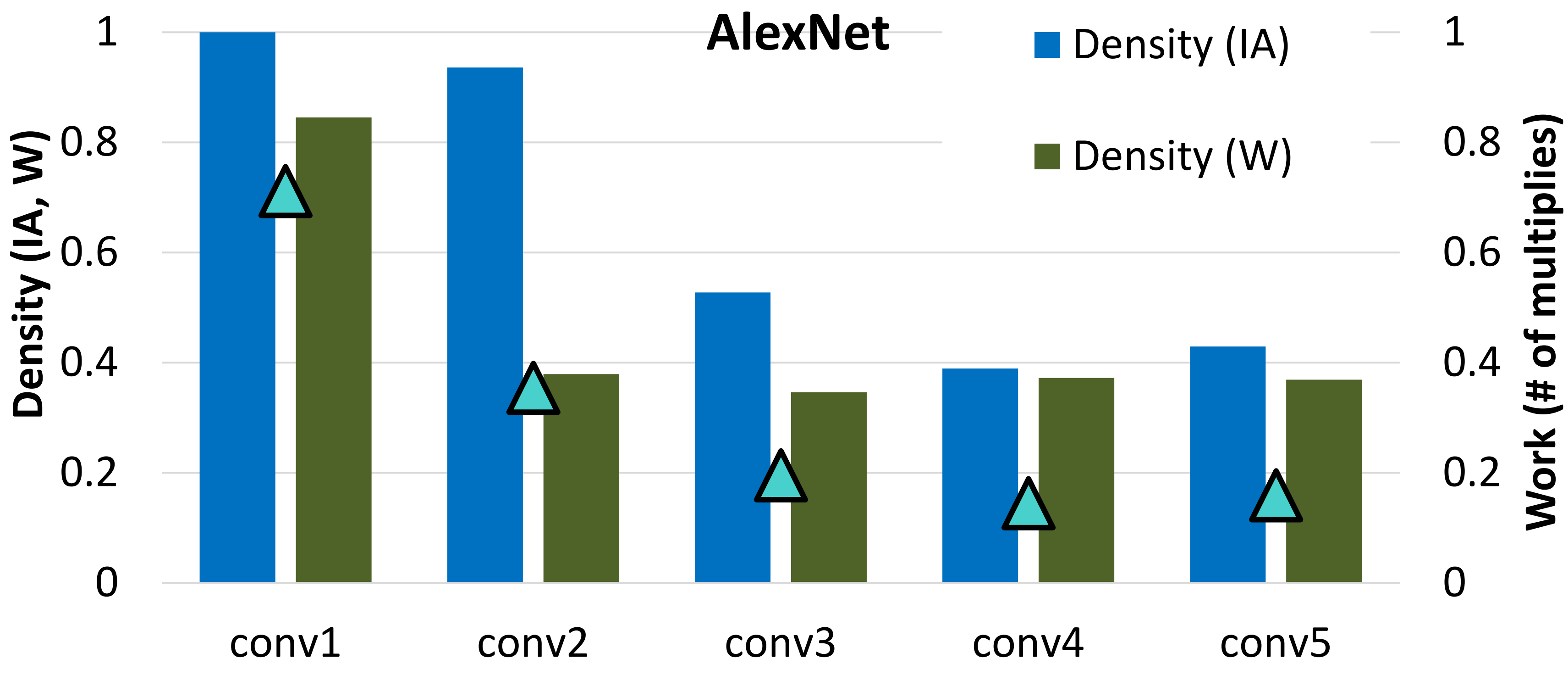
Many problems use Sparse Tensors



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

Motivation

- Leverage CNN sparsity to improve energy-efficiency



[Parashar, et.al., SCNN, ISCA 2017]

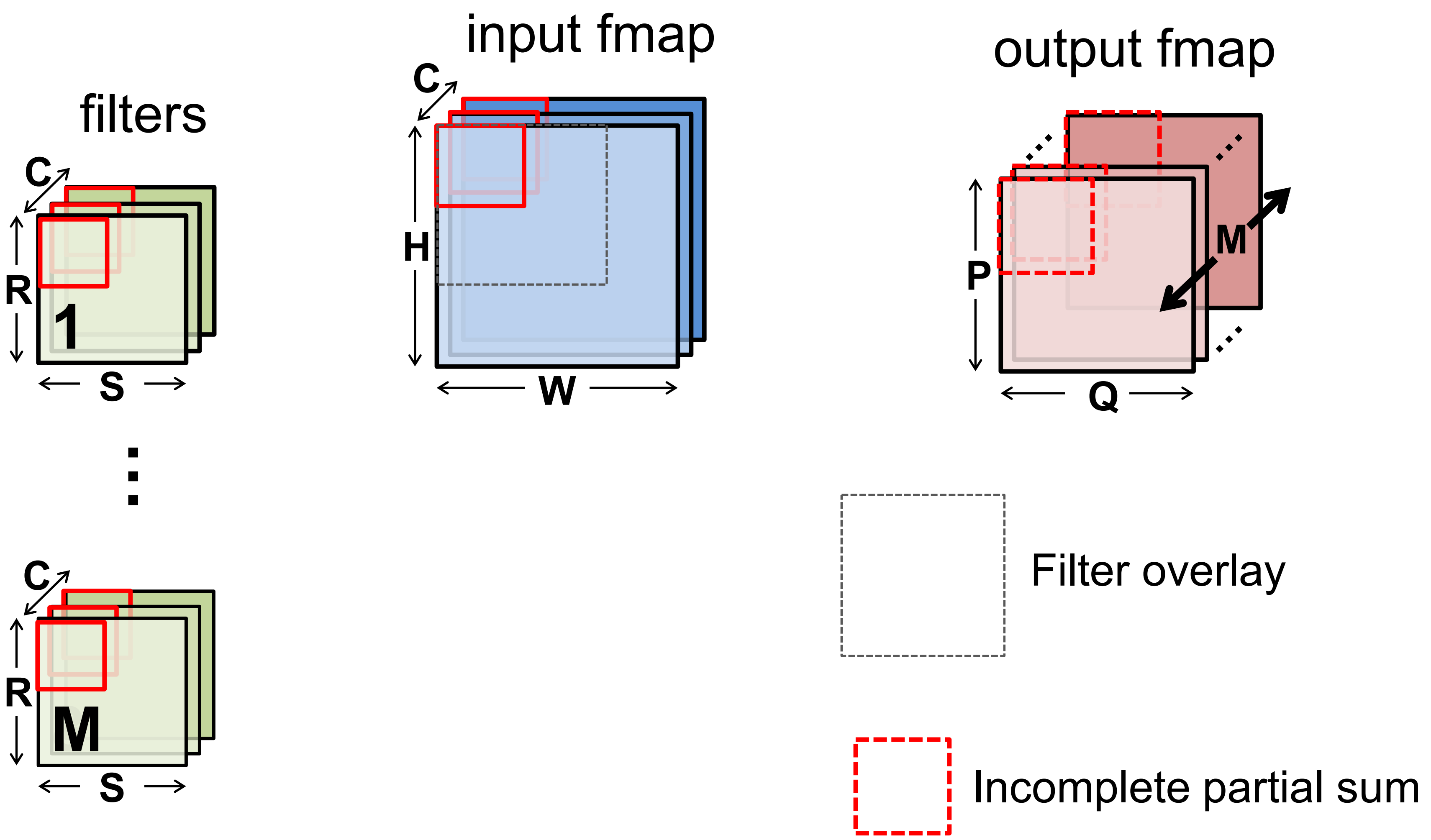
Aspects of Scheduling - Sparsity

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

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

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

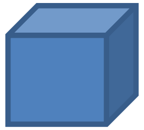
CONV Layer



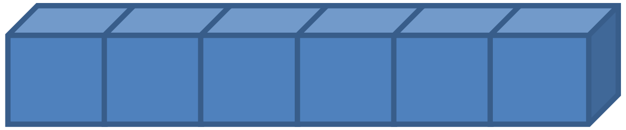
filters
C
R
1
S
C
R
M
S
input fmap
C
H
W
output fmap
P
M
Q
Filter overlay
Incomplete partial sum

Tensors

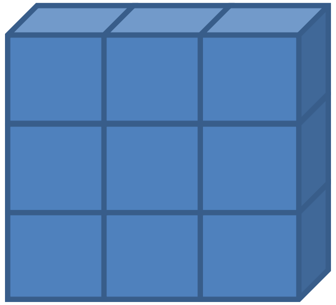
Rank-0 - Scalar



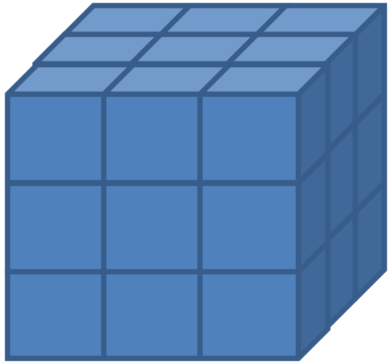
Rank-1 - Vector



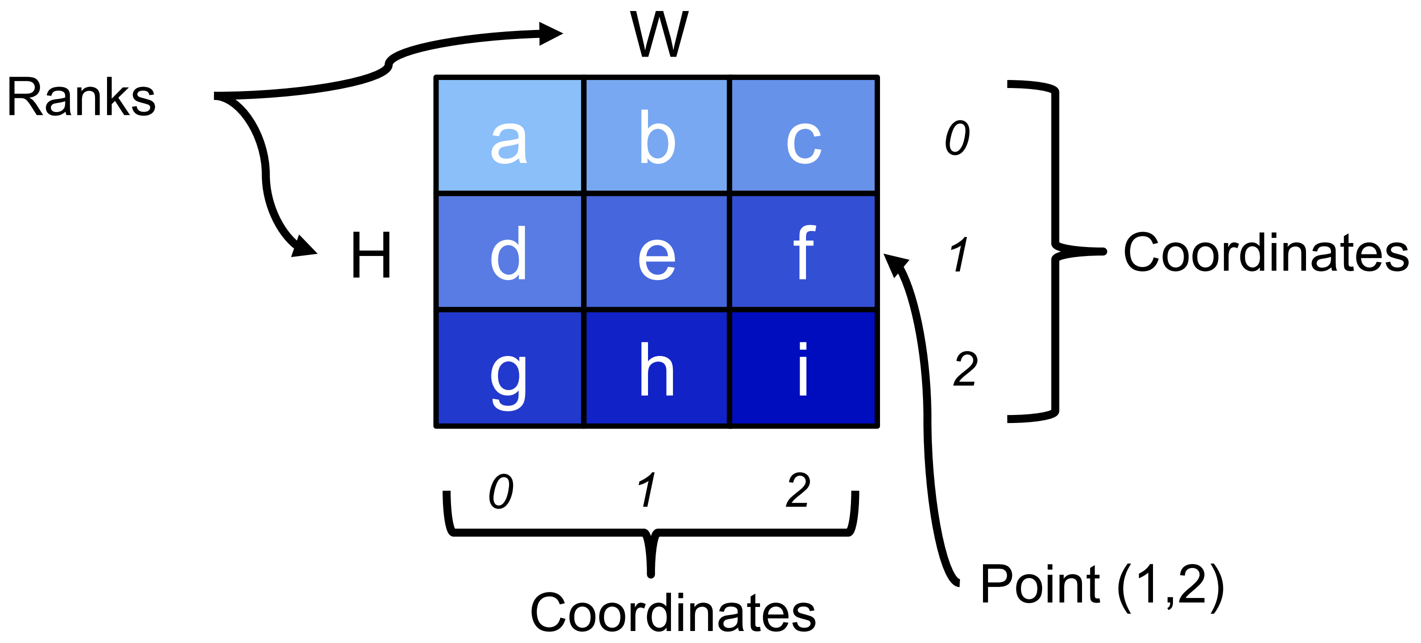
Rank-2 - Matrix



Rank-3 - Cube



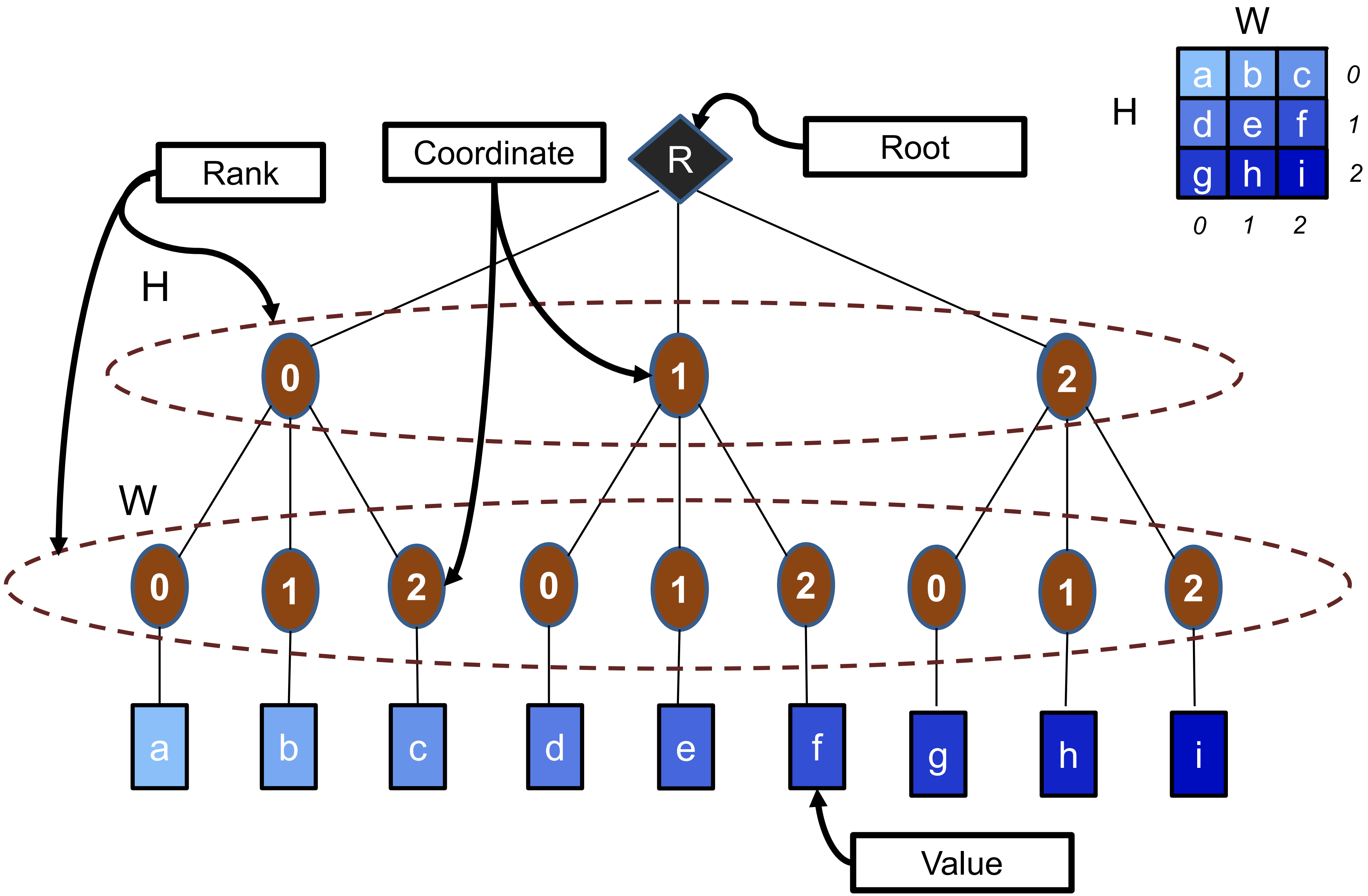
Tensor Data Terminology



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

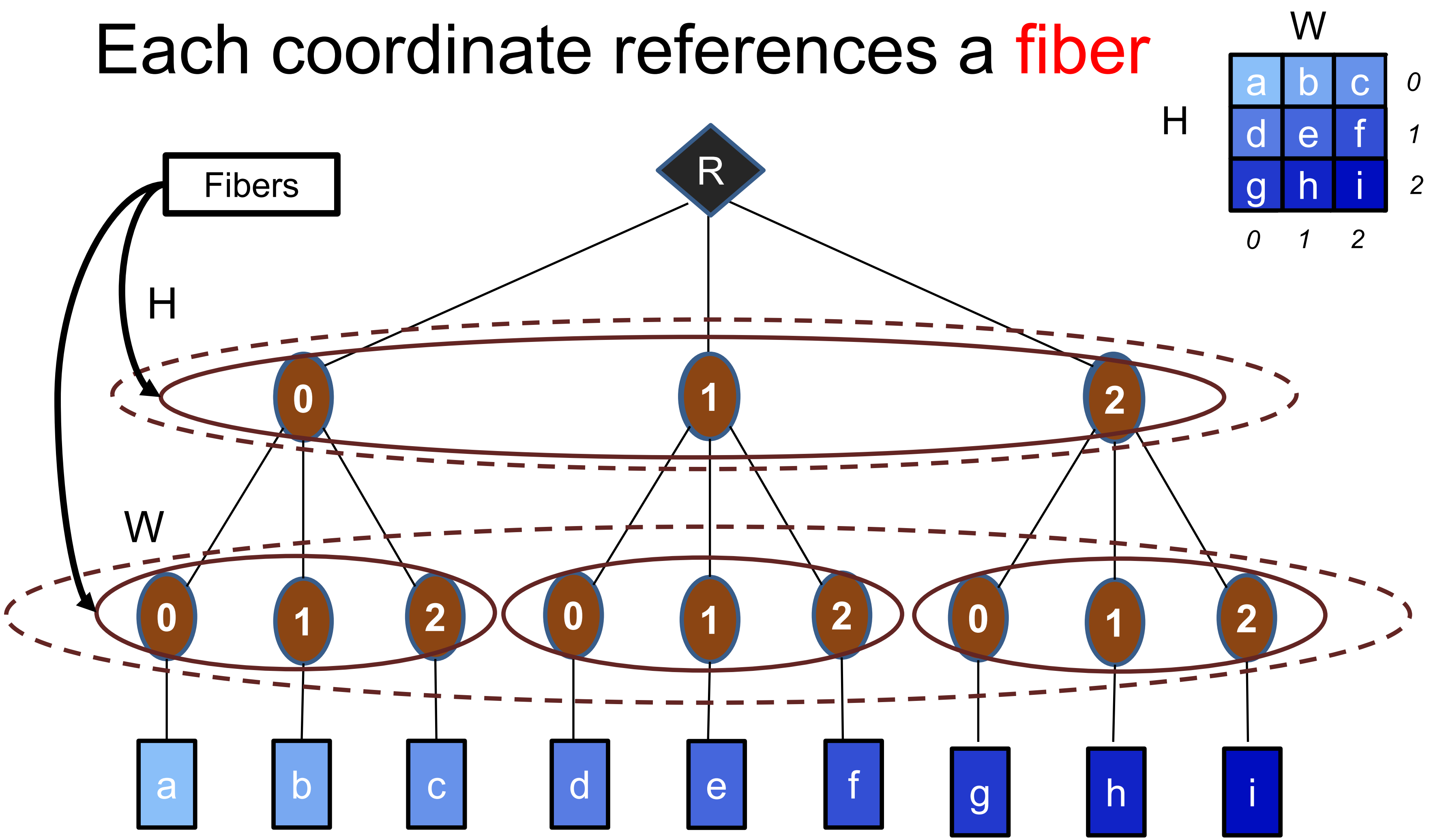
- Each element of the tensor is identified by the tuple of coordinates from each of its ranks, i.e., a “point”.
  So (1,2) -> “f”

Tree-based Tensor Abstraction



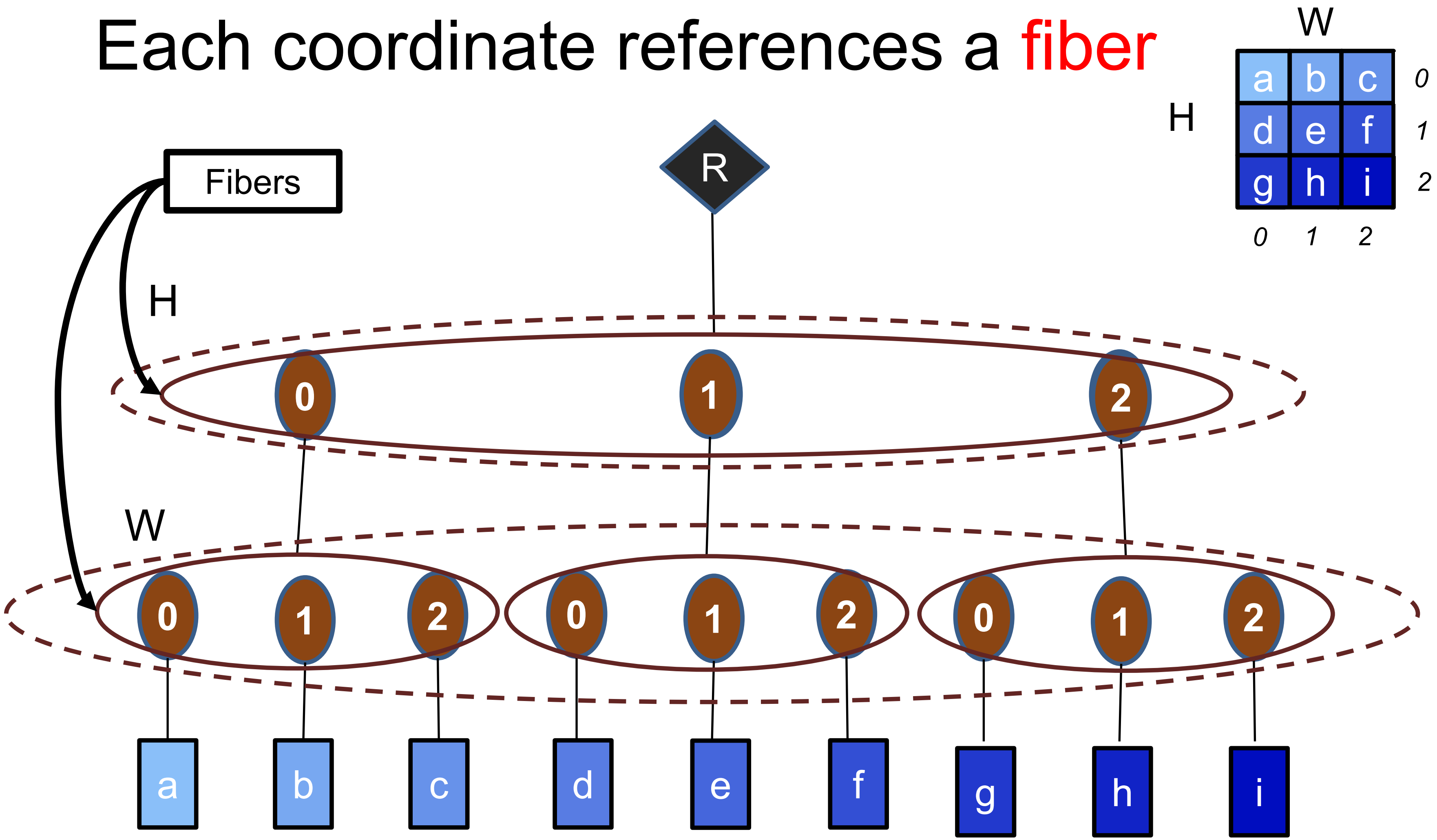
Tree-based Tensor Abstraction

Each coordinate references a fiber



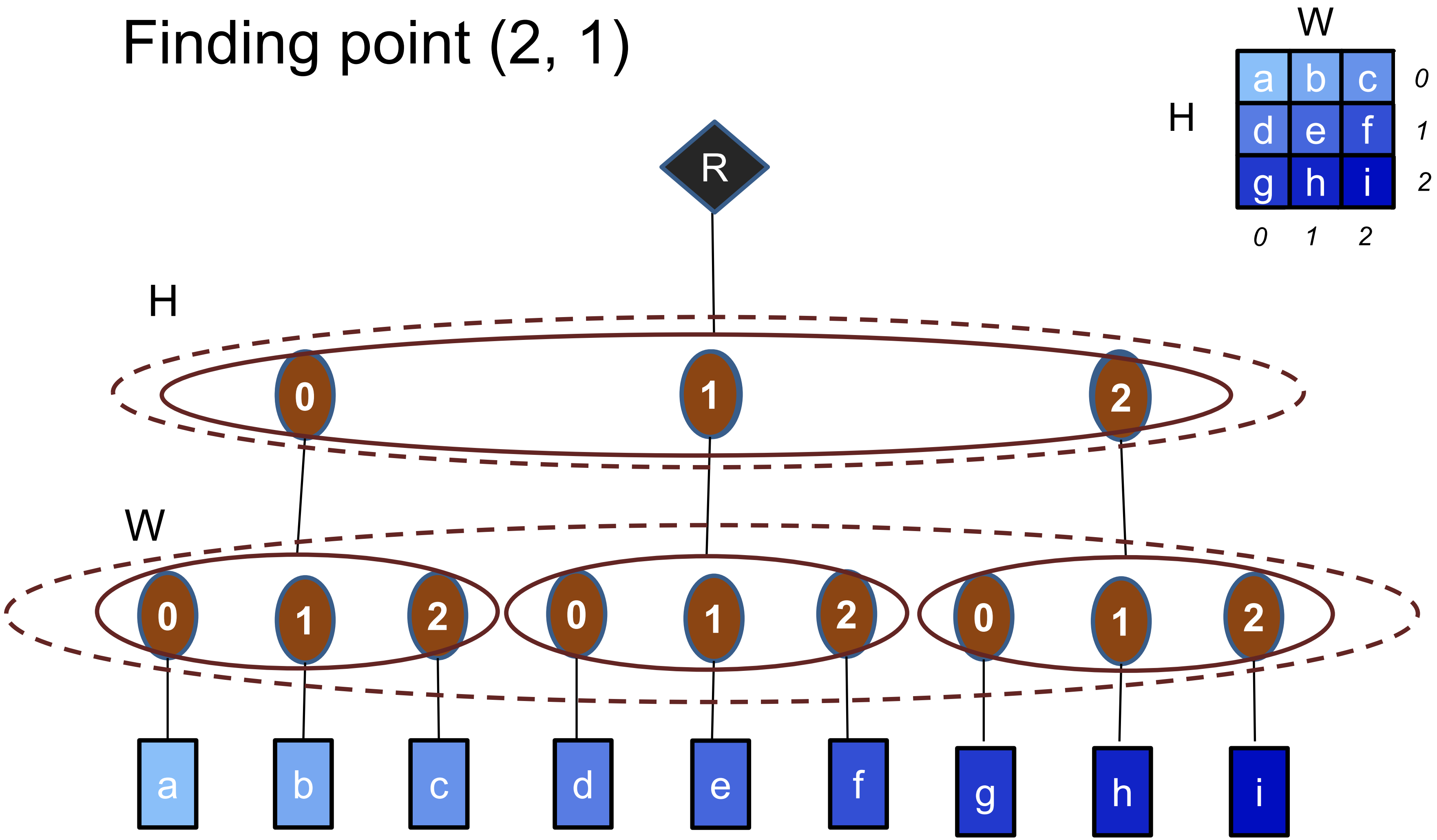
Fibertree Tensor Abstraction

Each coordinate references a fiber



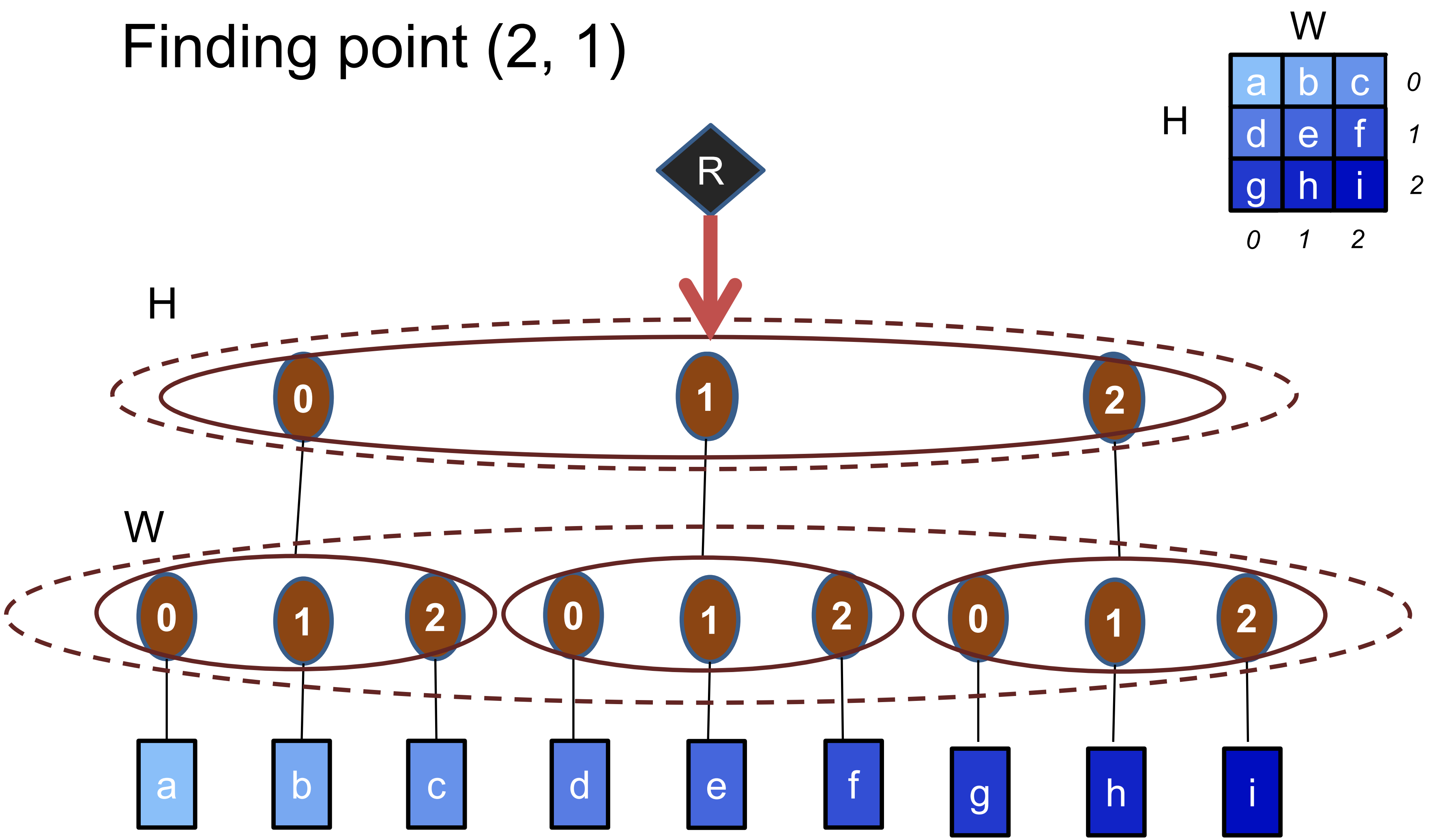
Fibertree Tensor Abstraction

Finding point (2, 1)



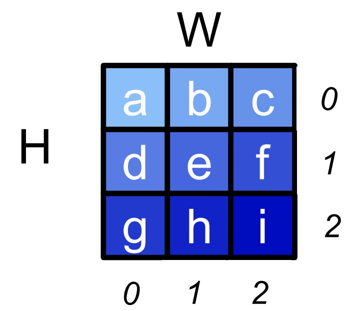
Fibertree Tensor Abstraction

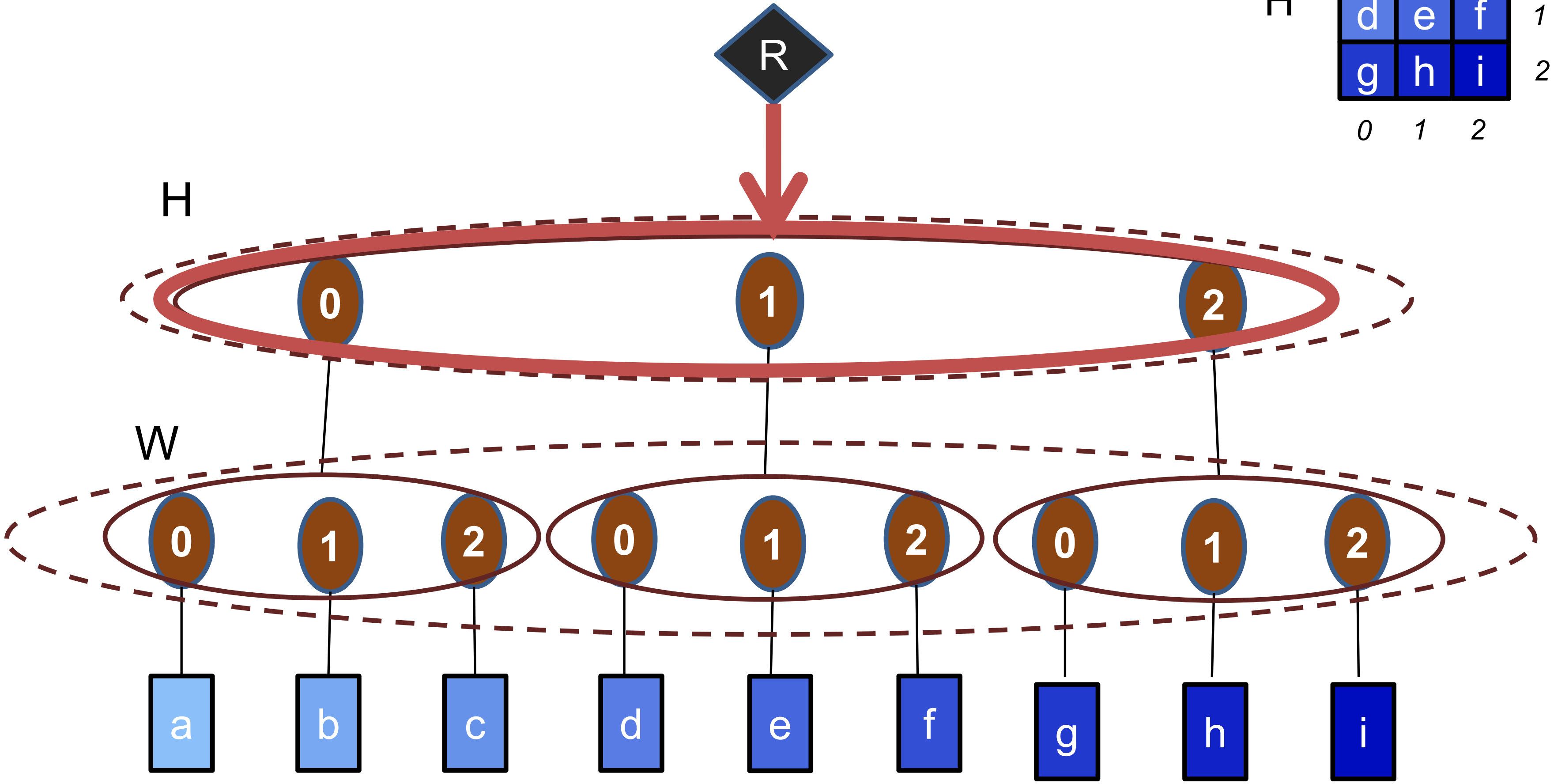
Finding point (2, 1)



Fibertree Tensor Abstraction

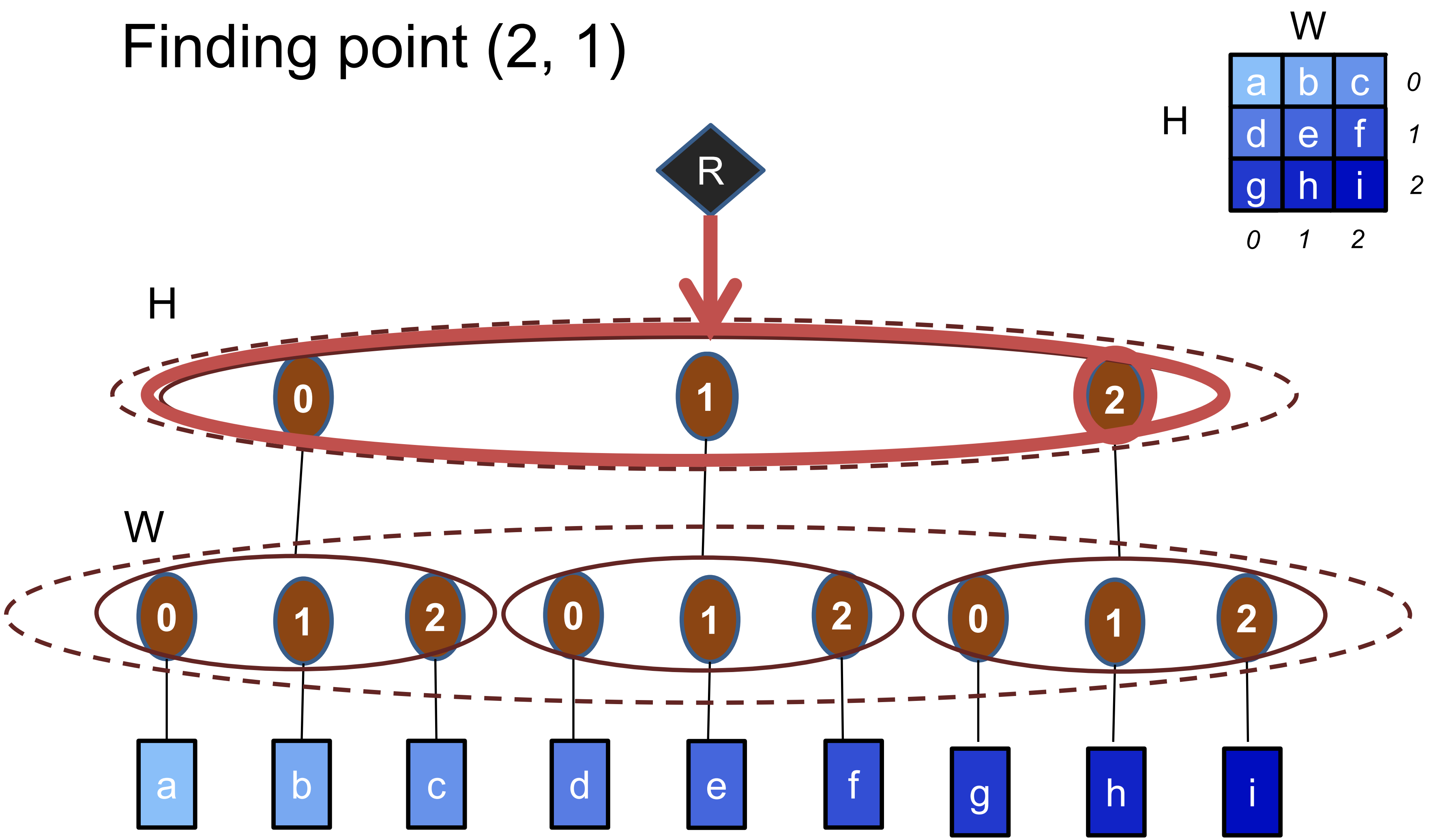
Finding point (2, 1)





Fibertree Tensor Abstraction

Finding point (2, 1)

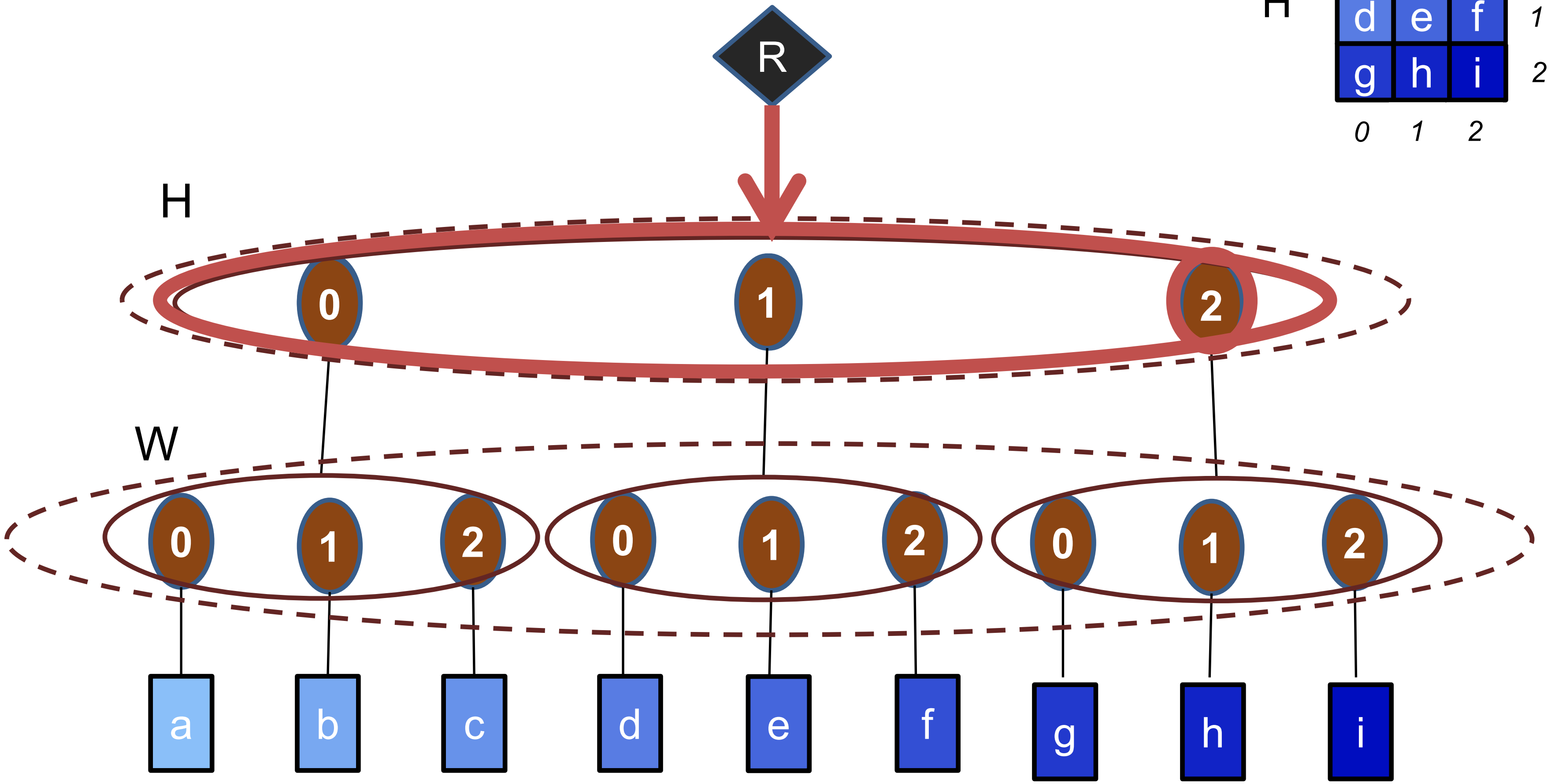


Fibertree Tensor Abstraction

Finding point (2, 1)

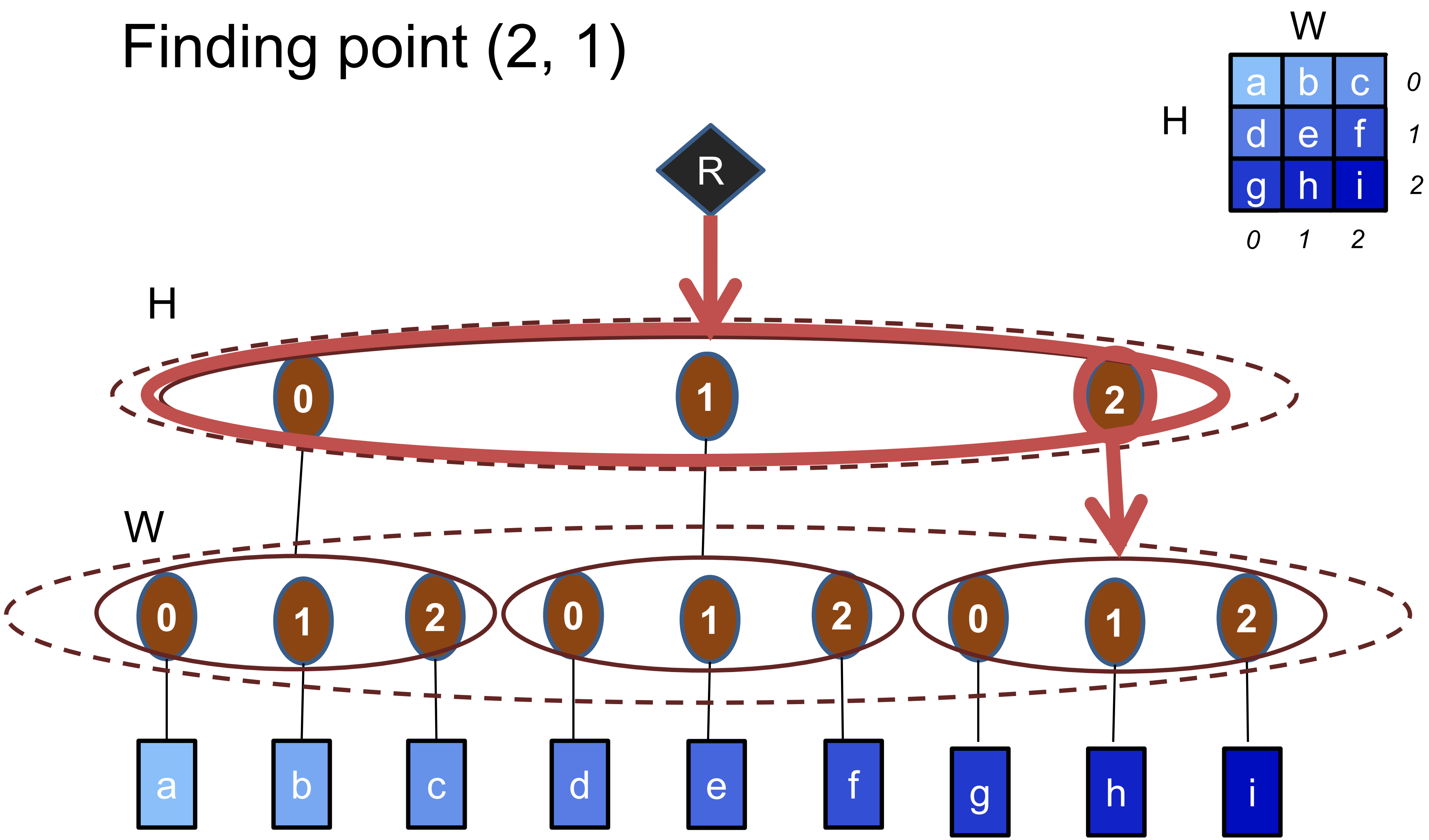
W

H	0	1	2
0	a	b	c
1	d	e	f
2	g	h	i



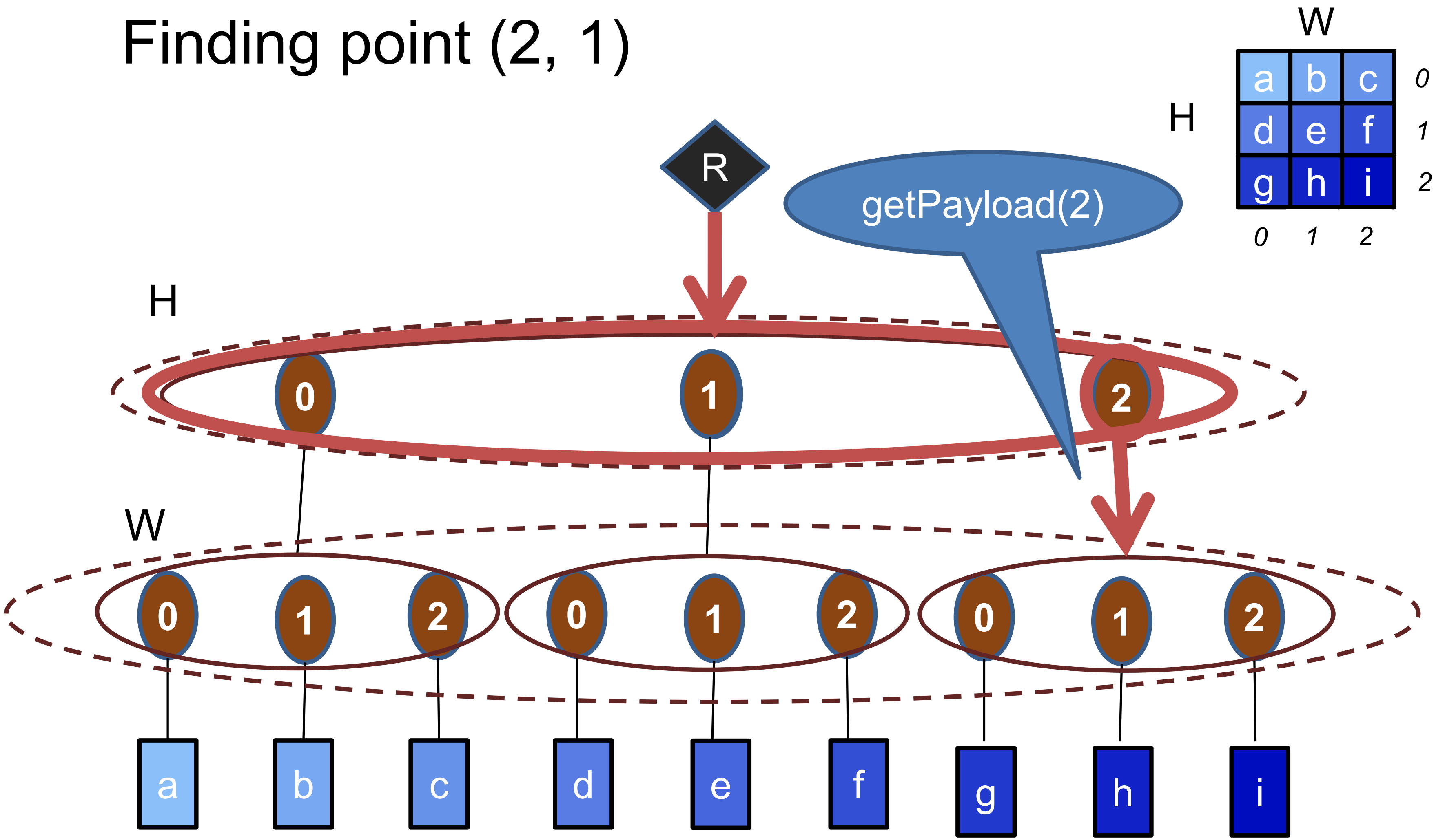
Fibertree Tensor Abstraction

Finding point (2, 1)



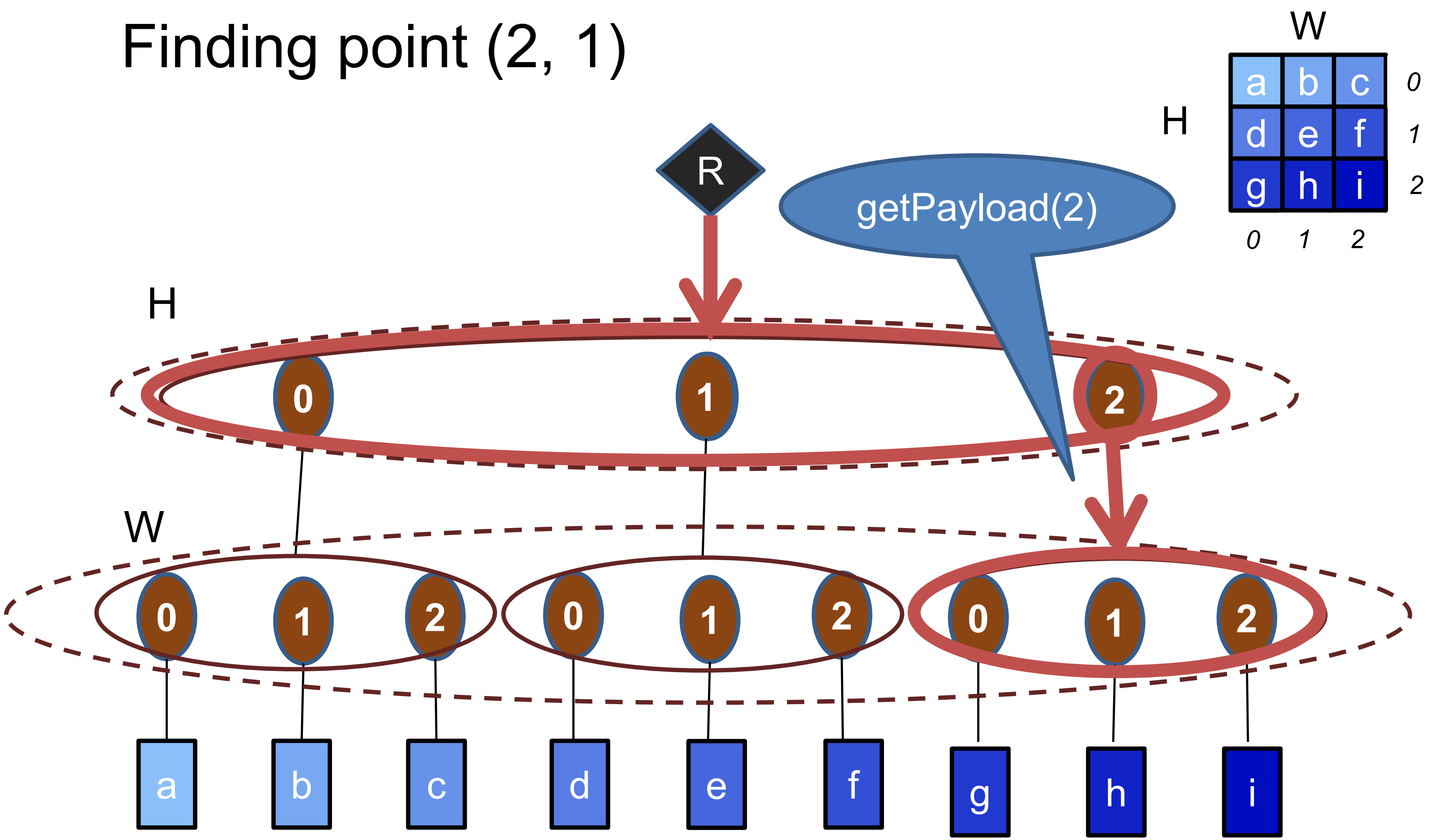
Fibertree Tensor Abstraction

Finding point (2, 1)



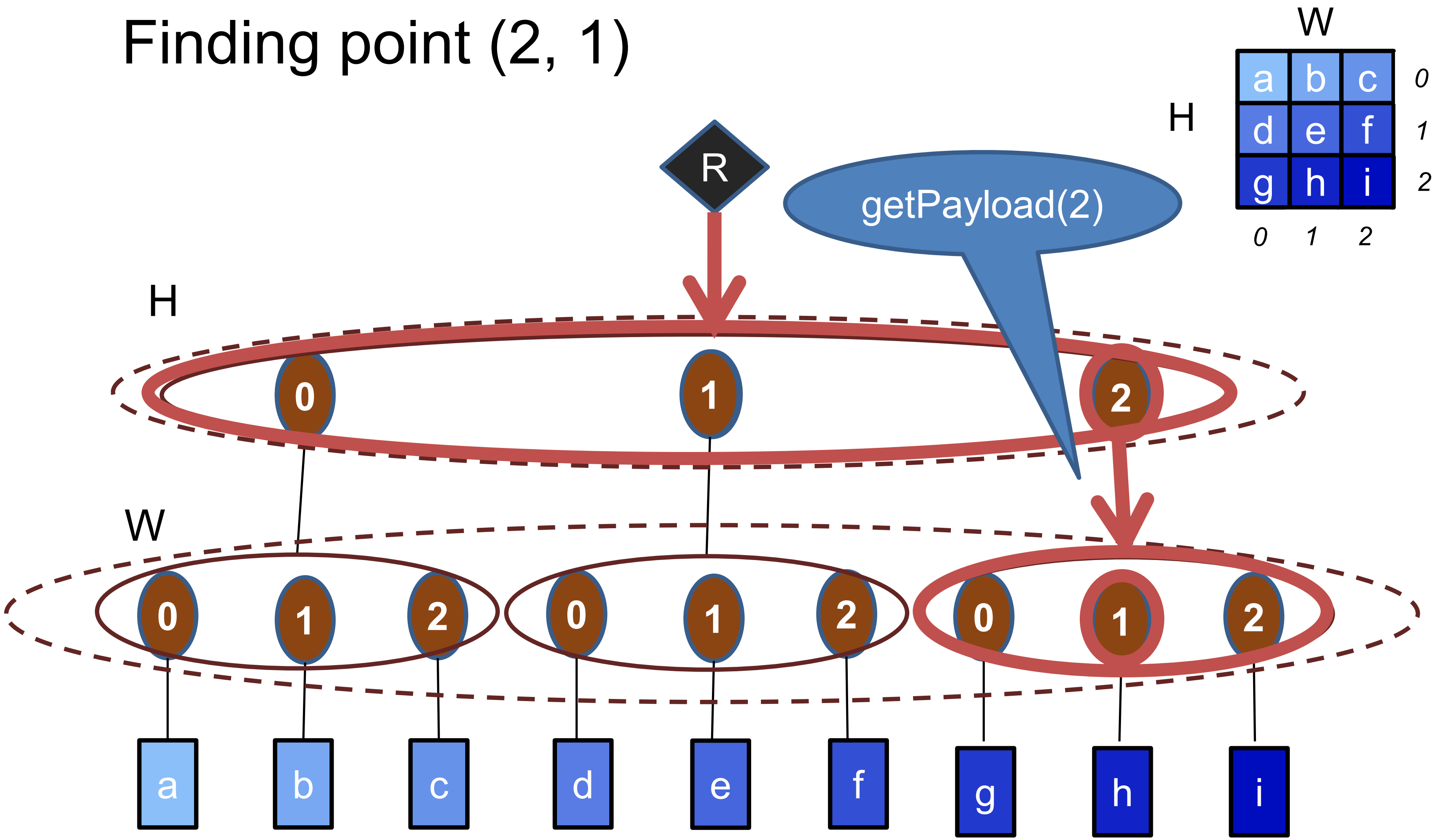
Fibertree Tensor Abstraction

Finding point (2, 1)



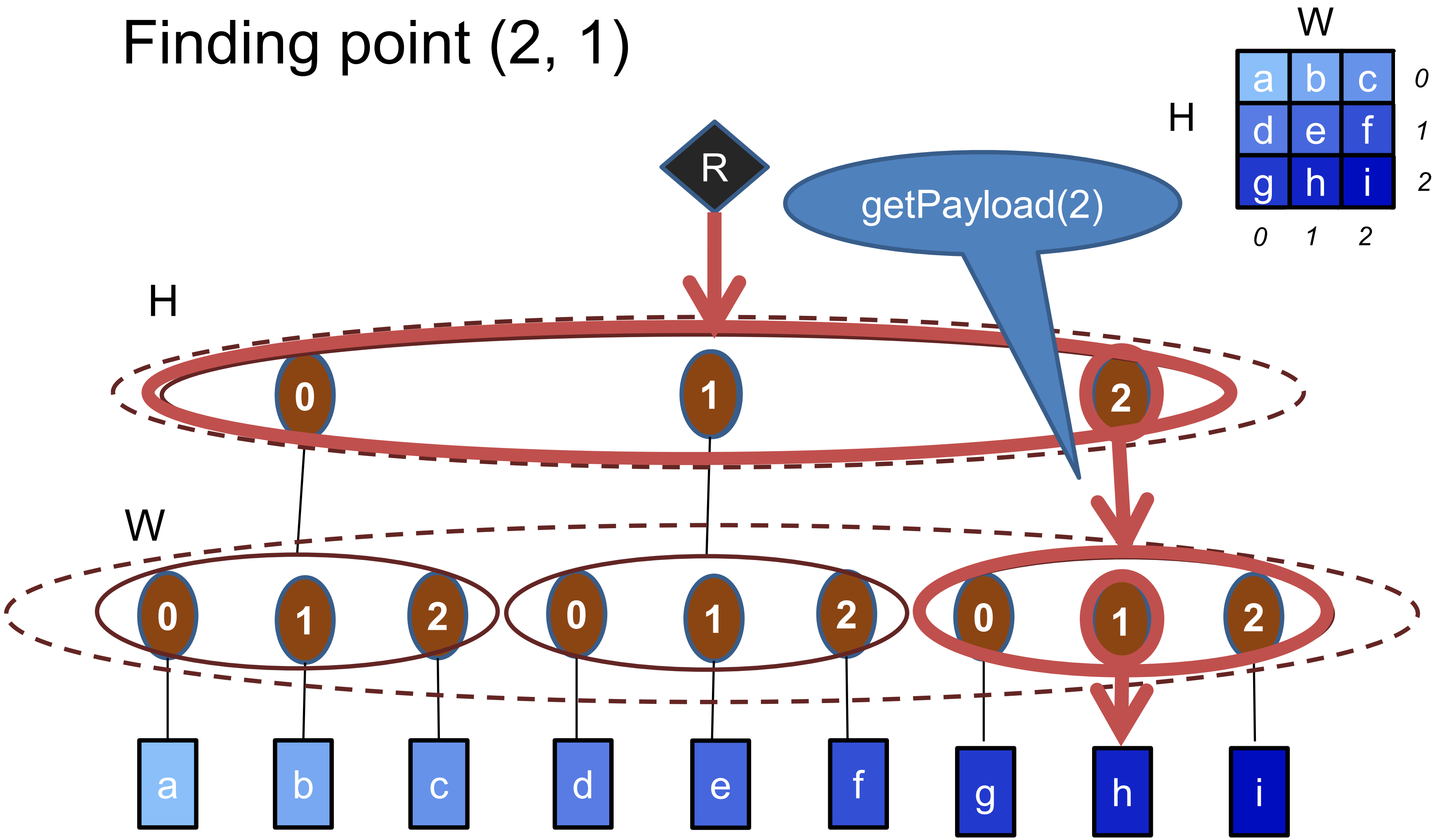
Fibertree Tensor Abstraction

Finding point (2, 1)



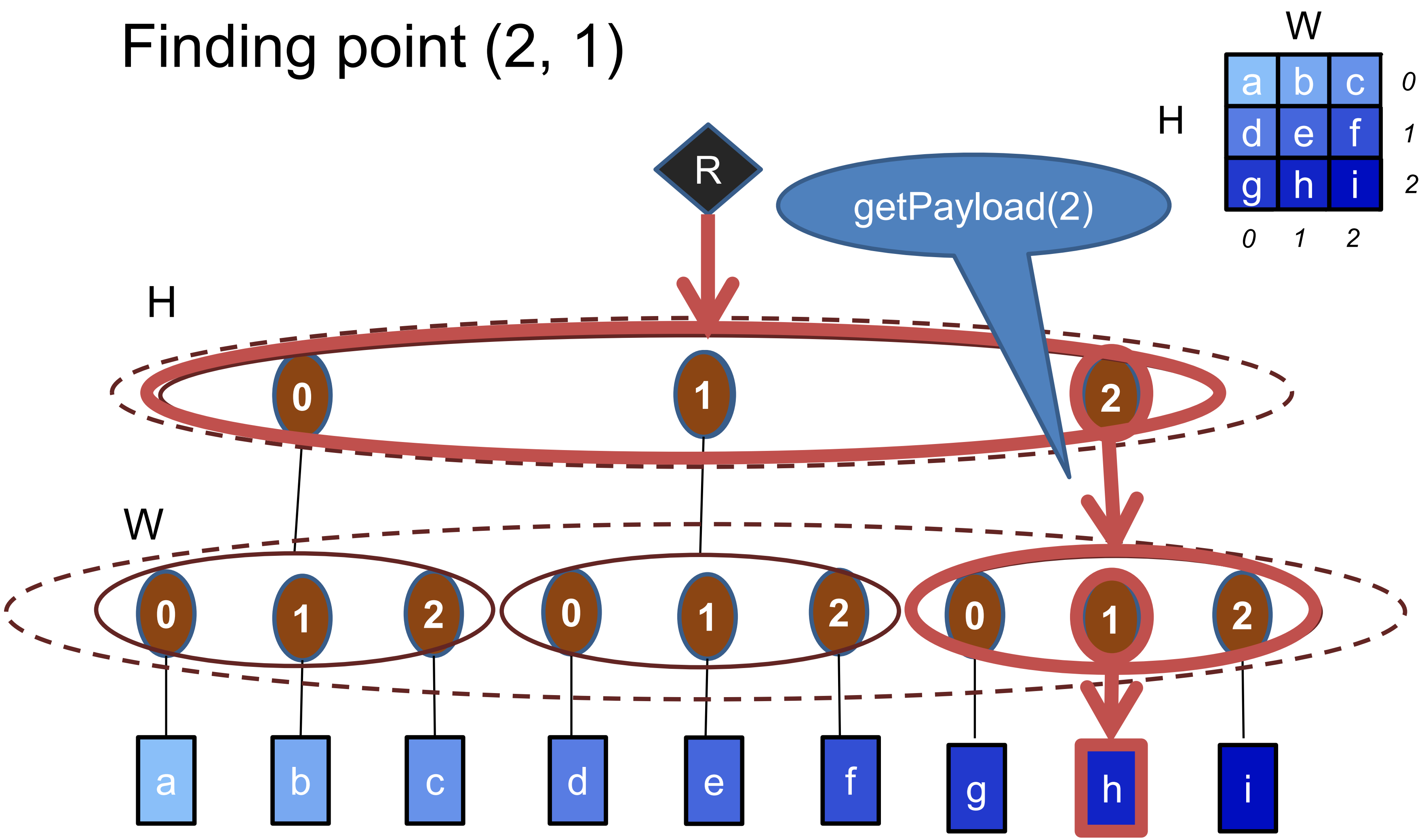
Fibertree Tensor Abstraction

Finding point (2, 1)



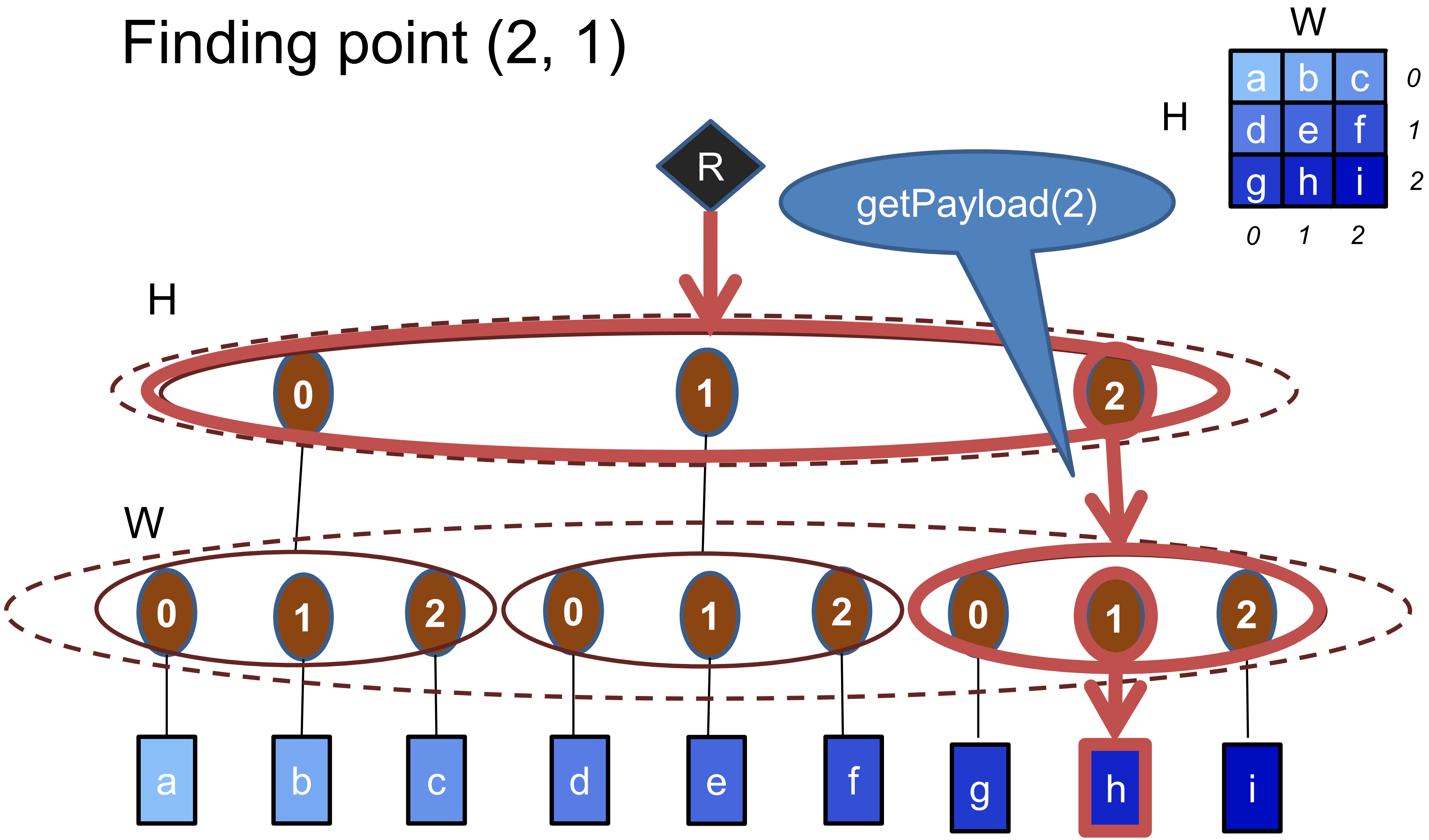
Fibertree Tensor Abstraction

Finding point (2, 1)



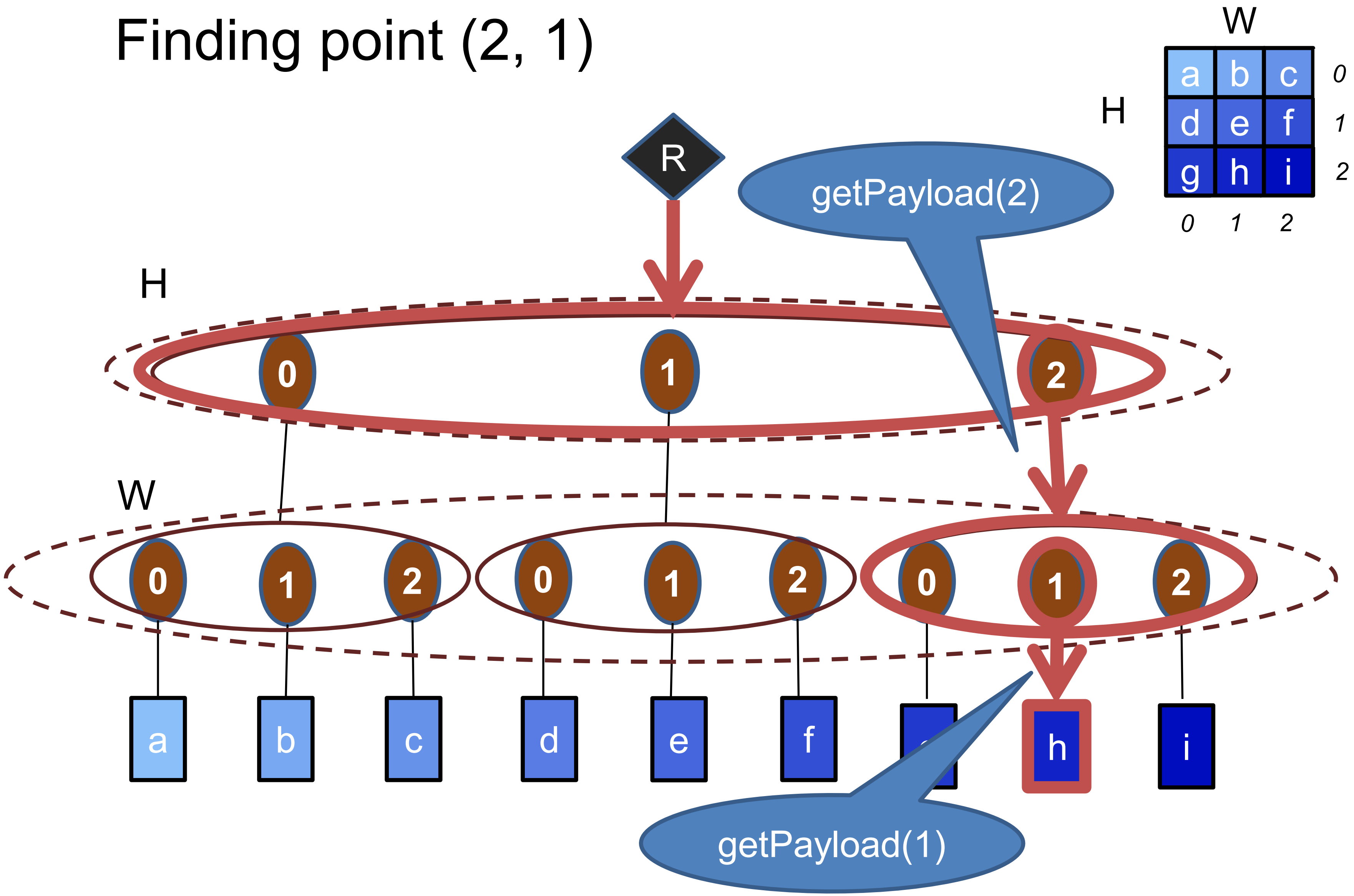
Fibertree Tensor Abstraction

Finding point (2, 1)

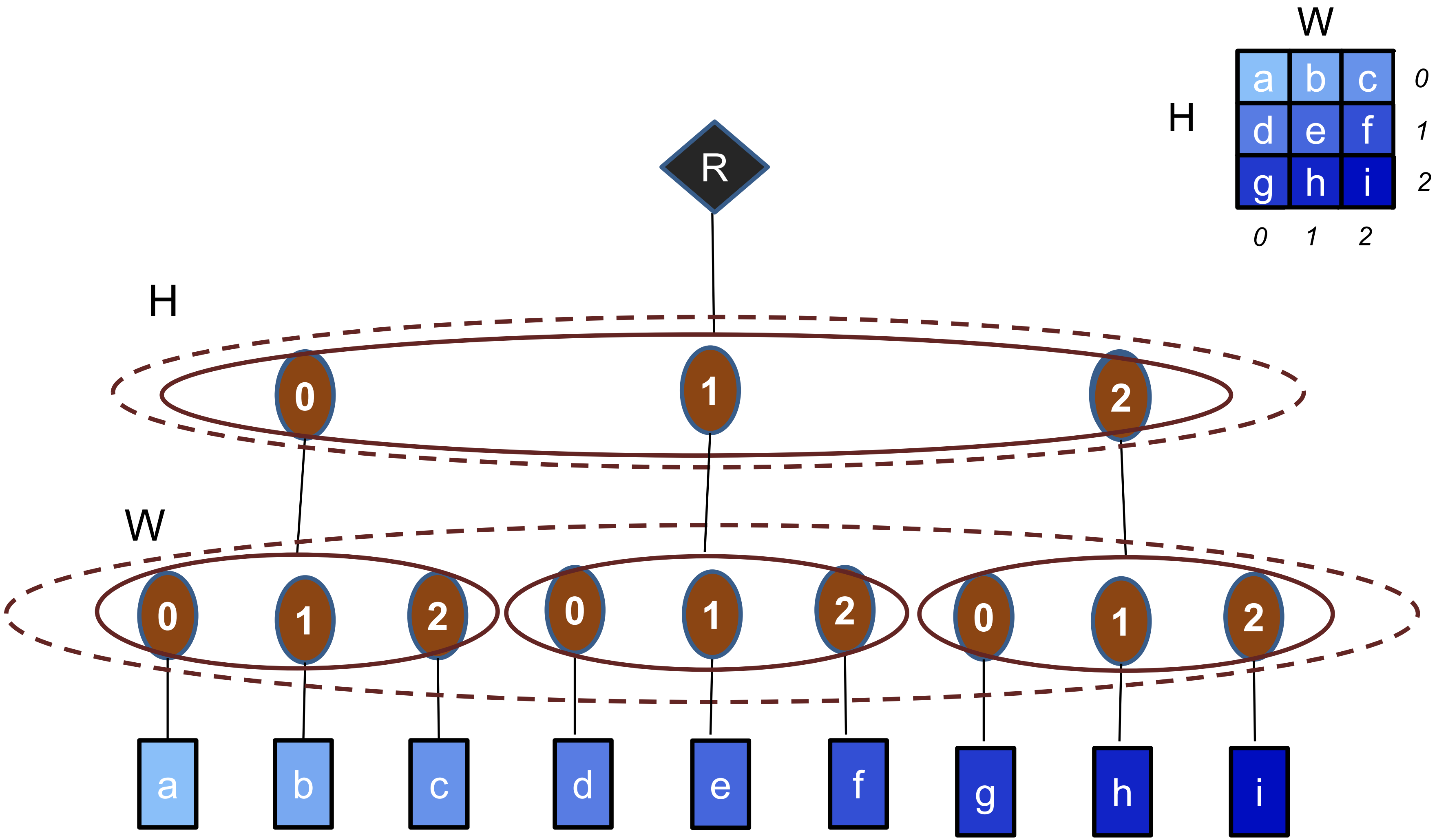


Fibertree Tensor Abstraction

Finding point (2, 1)

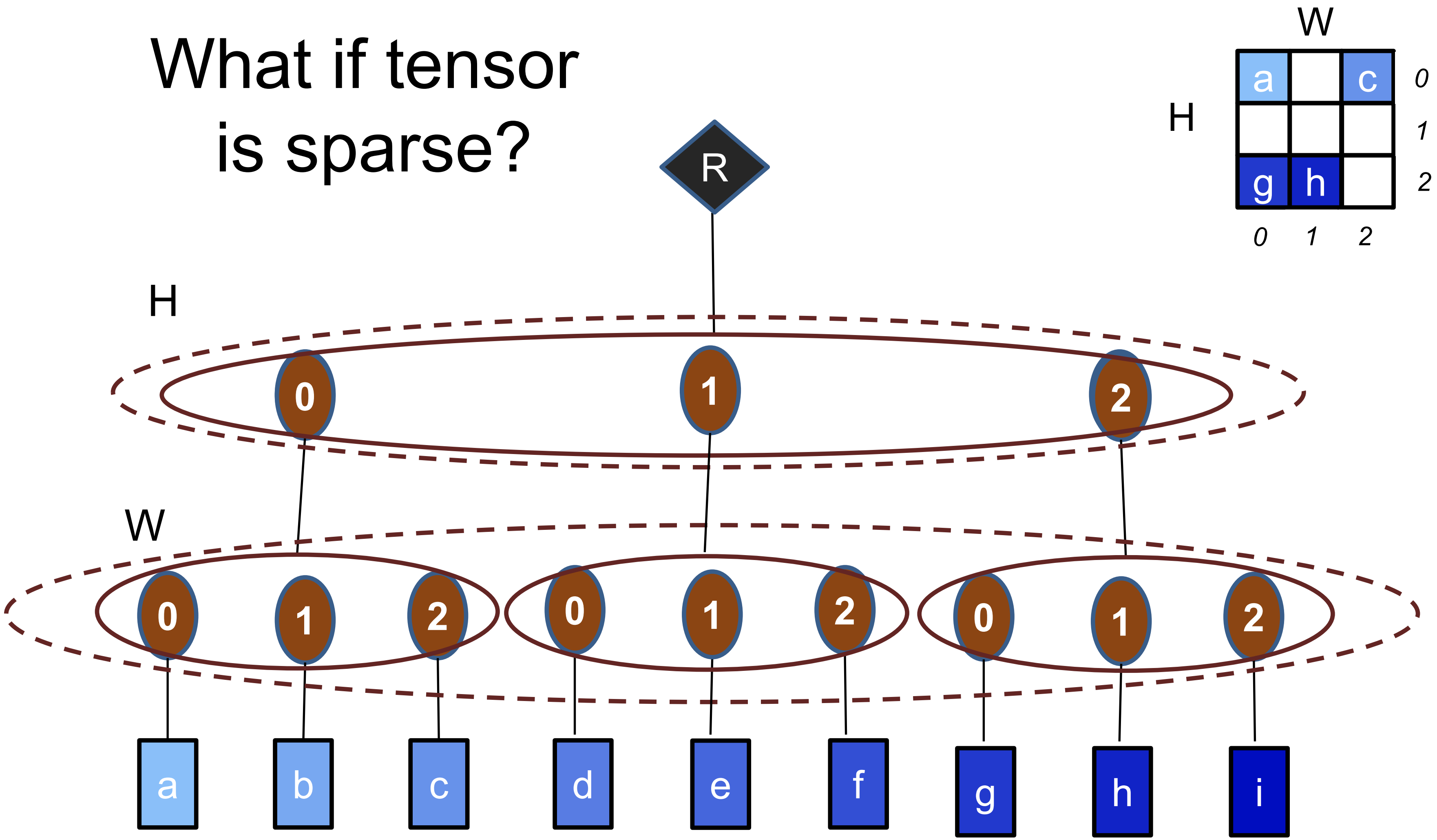


Fibertree Tensor Abstraction



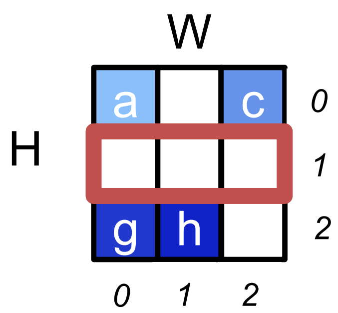
Fibertree Tensor Abstraction

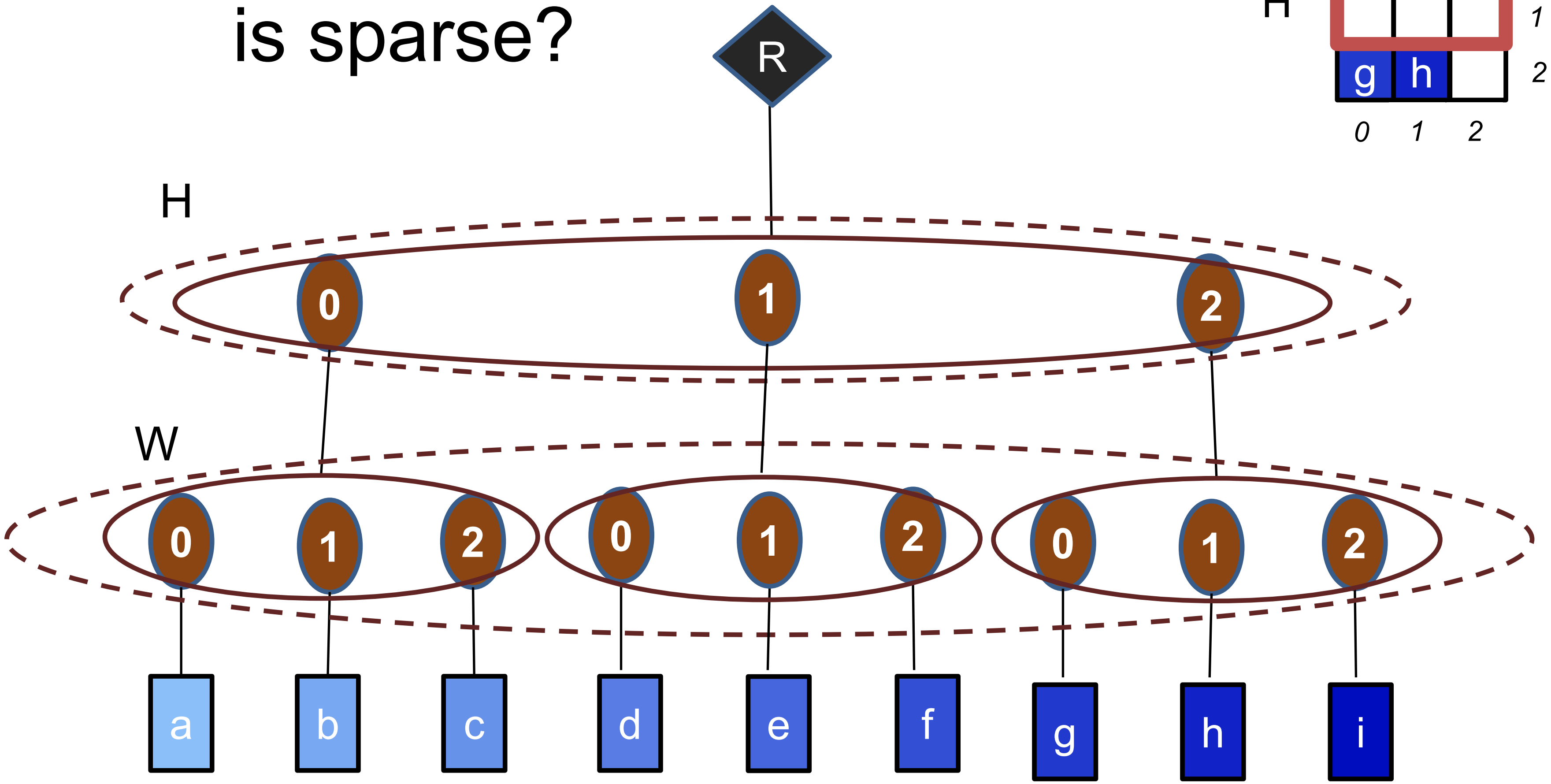
What if tensor is sparse?



Fibertree Tensor Abstraction

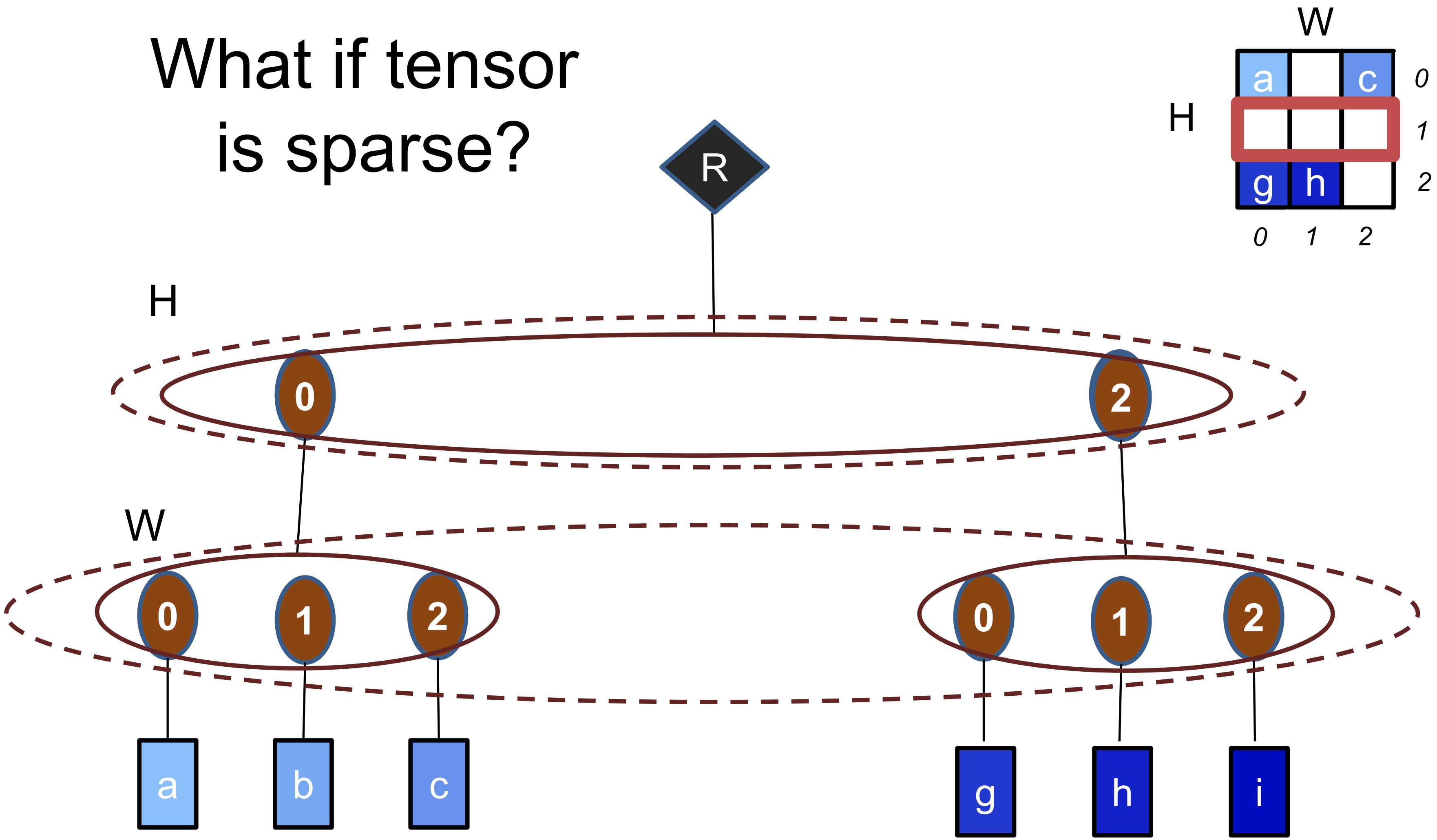
What if tensor is sparse?





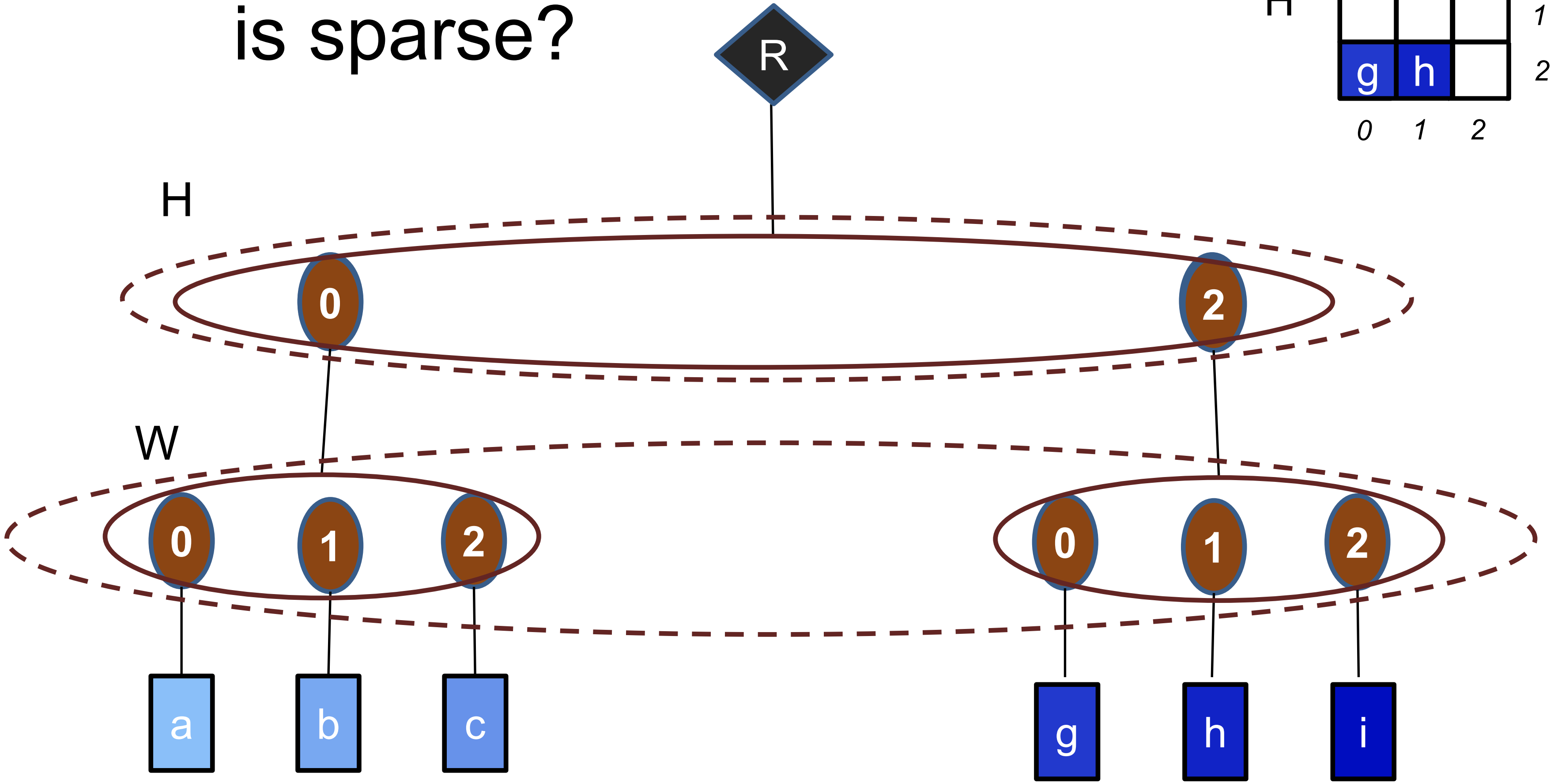
Fibertree Tensor Abstraction

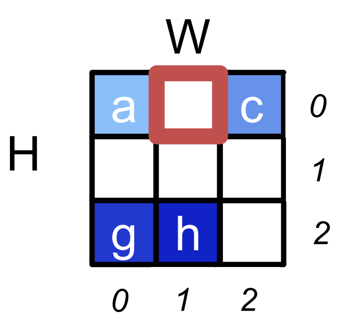
What if tensor is sparse?



Fibertree Tensor Abstraction

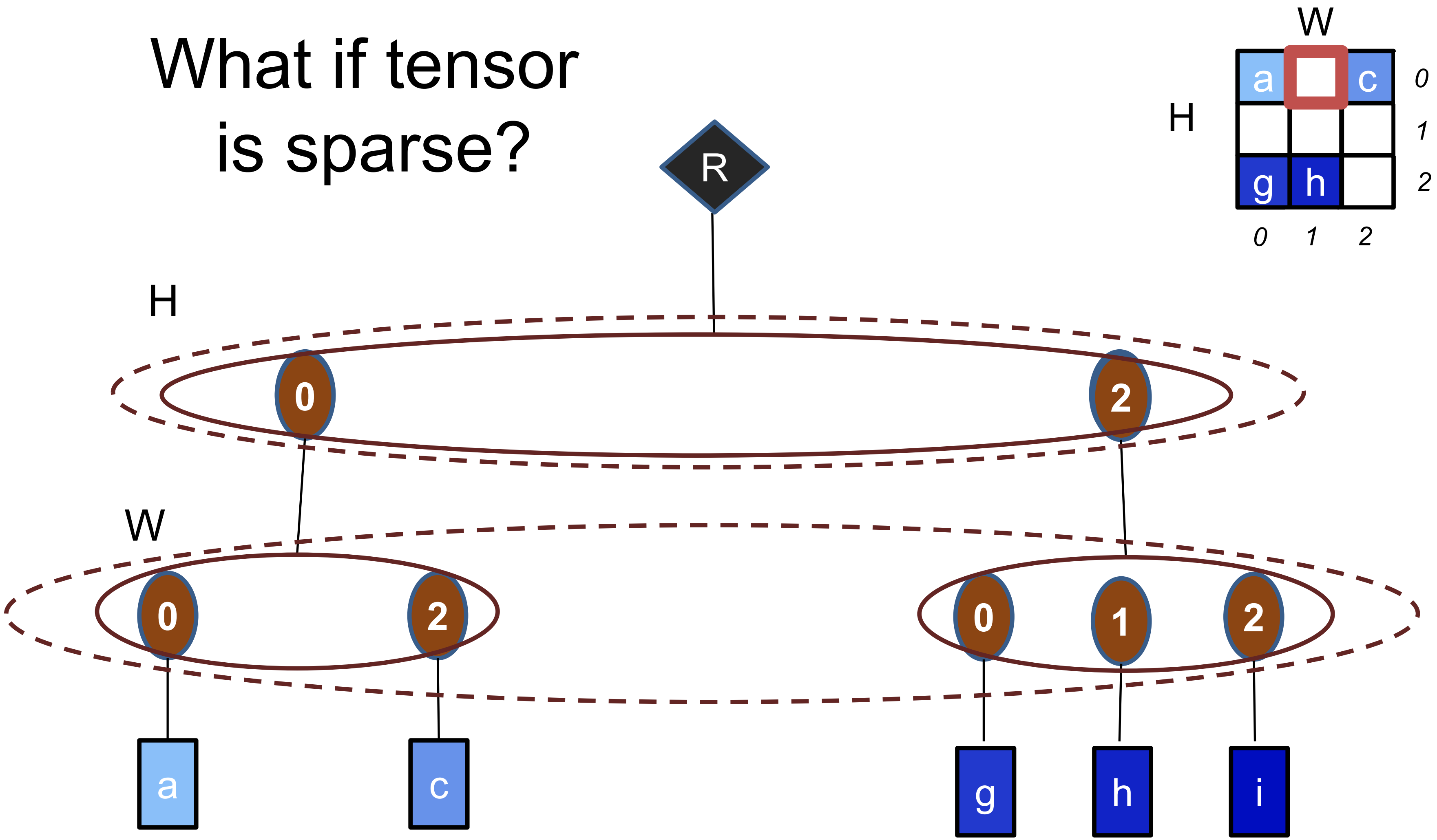
What if tensor is sparse?





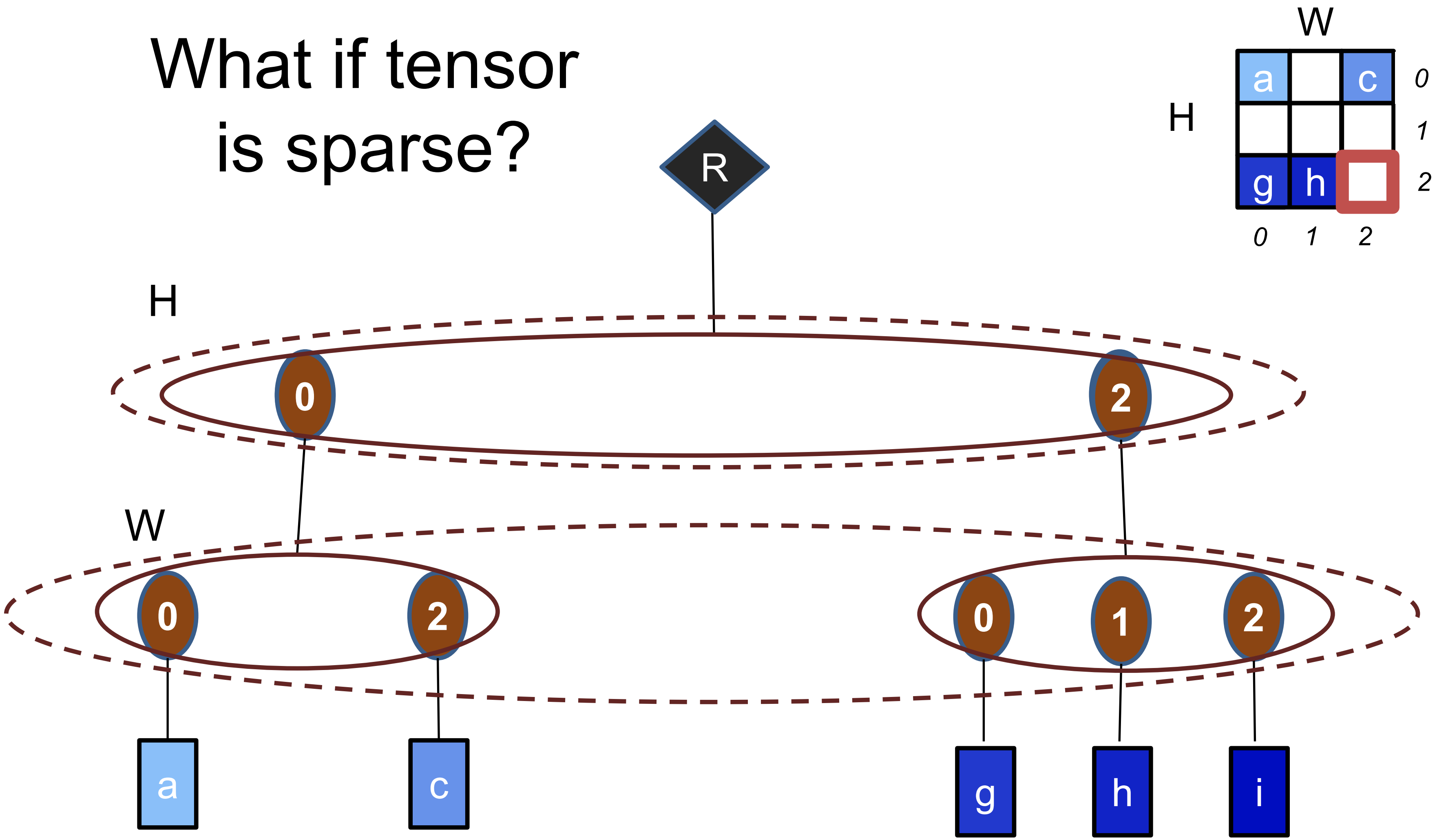
Fibertree Tensor Abstraction

What if tensor is sparse?



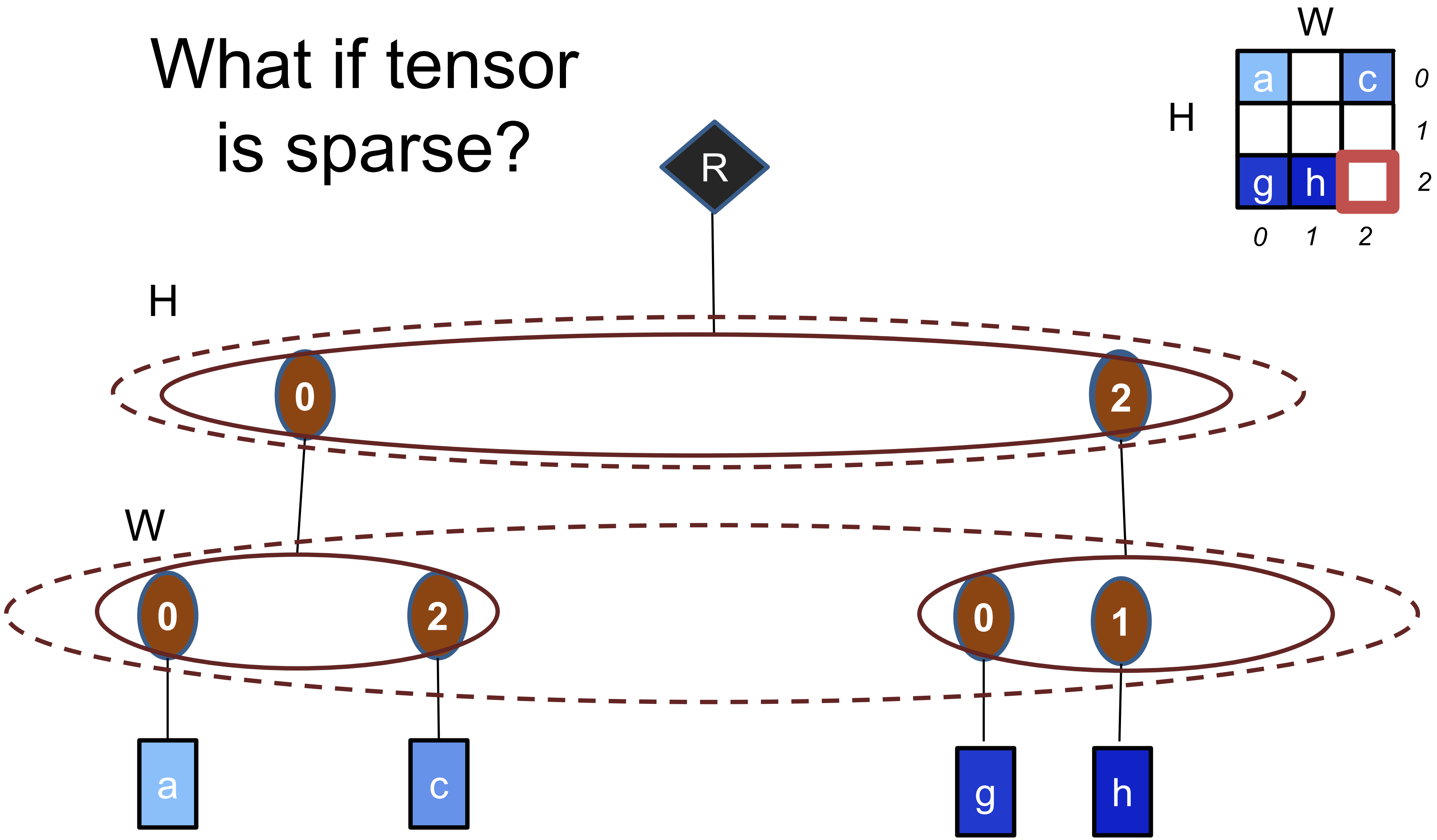
Fibertree Tensor Abstraction

What if tensor is sparse?



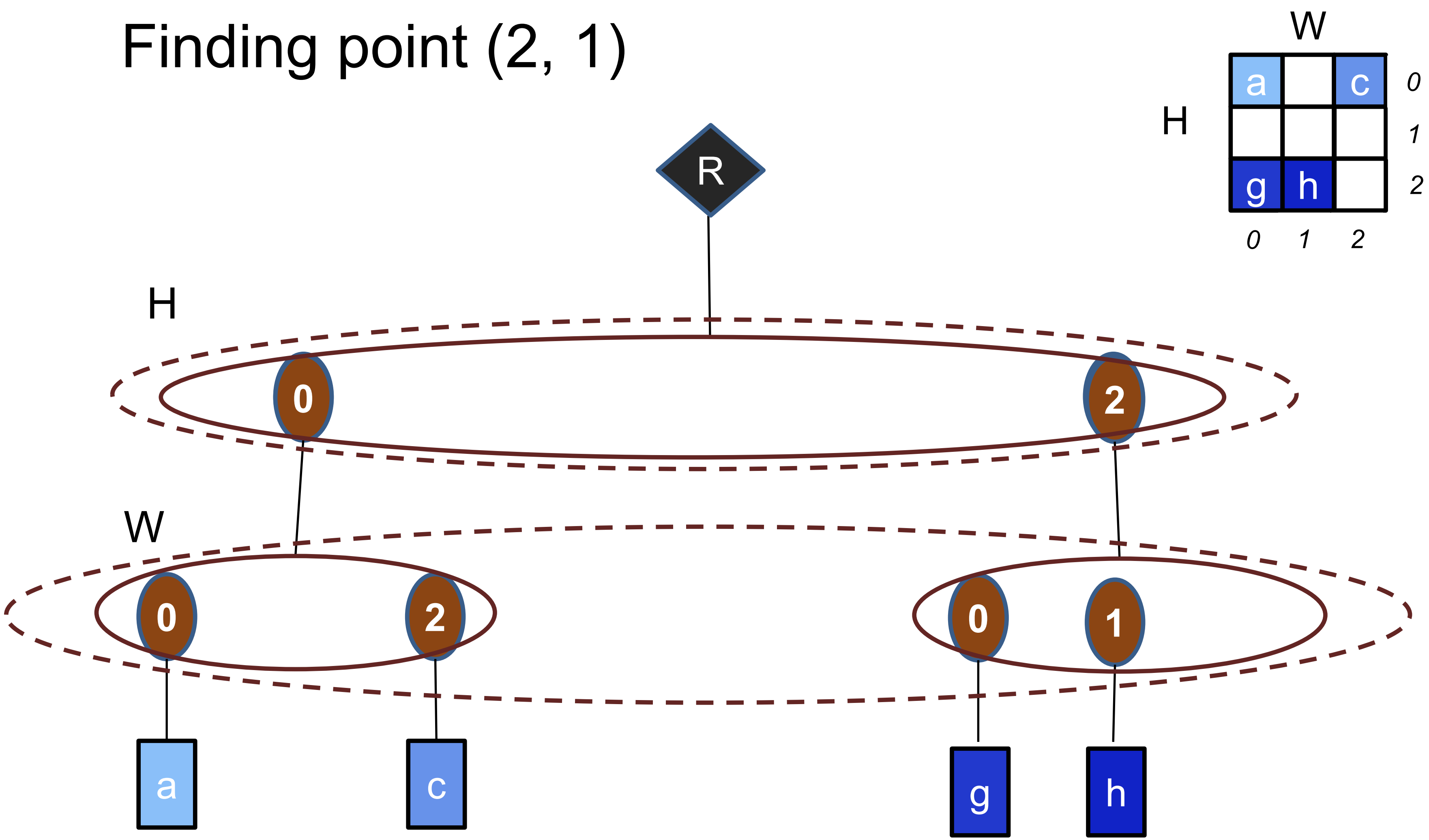
Fibertree Tensor Abstraction

What if tensor is sparse?



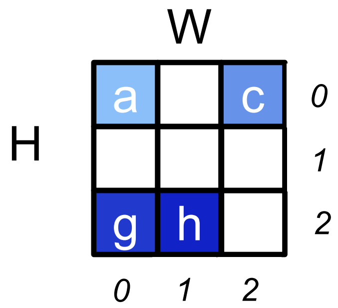
Fibertree Tensor Abstraction

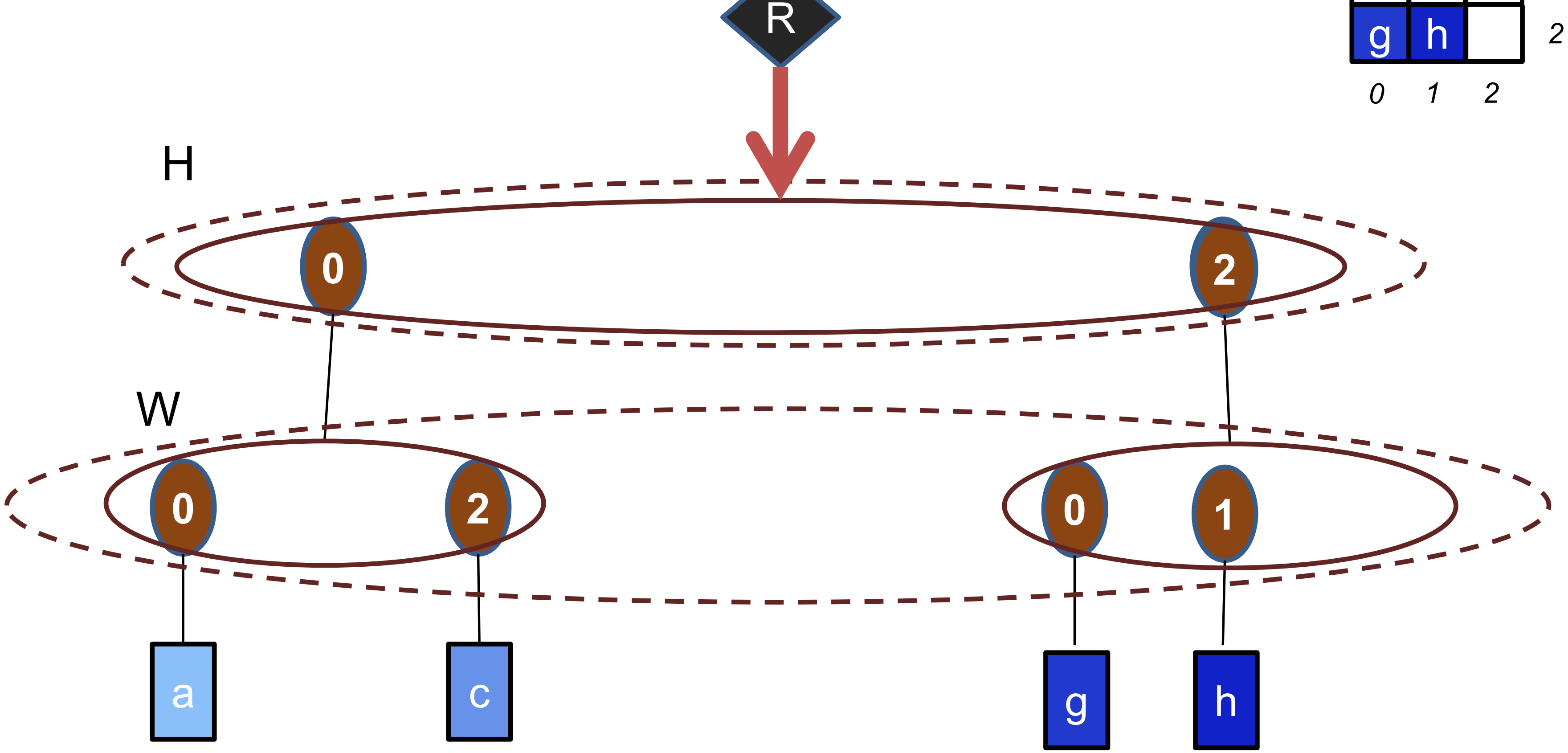
Finding point (2, 1)



Fibertree Tensor Abstraction

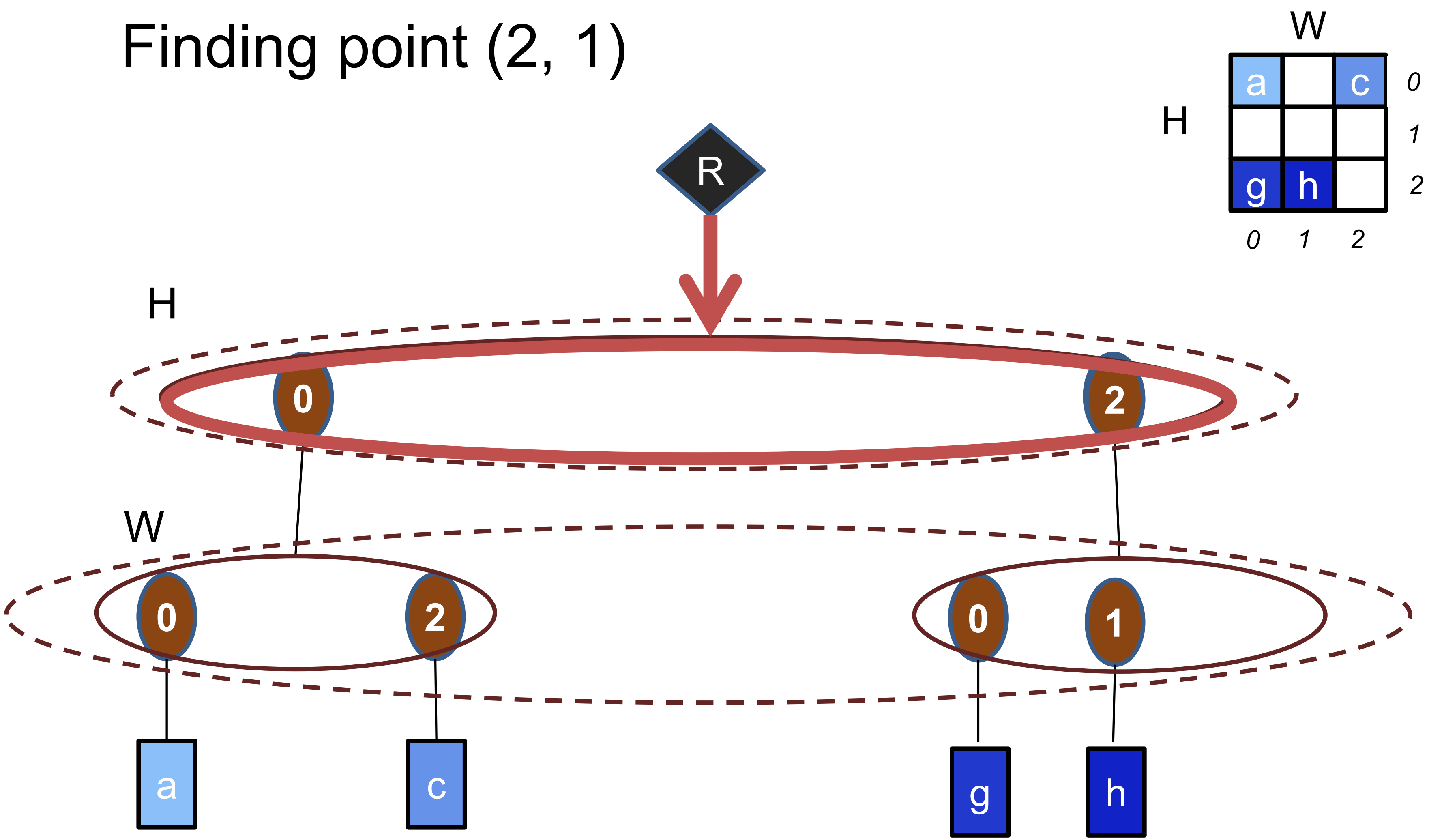
Finding point (2, 1)





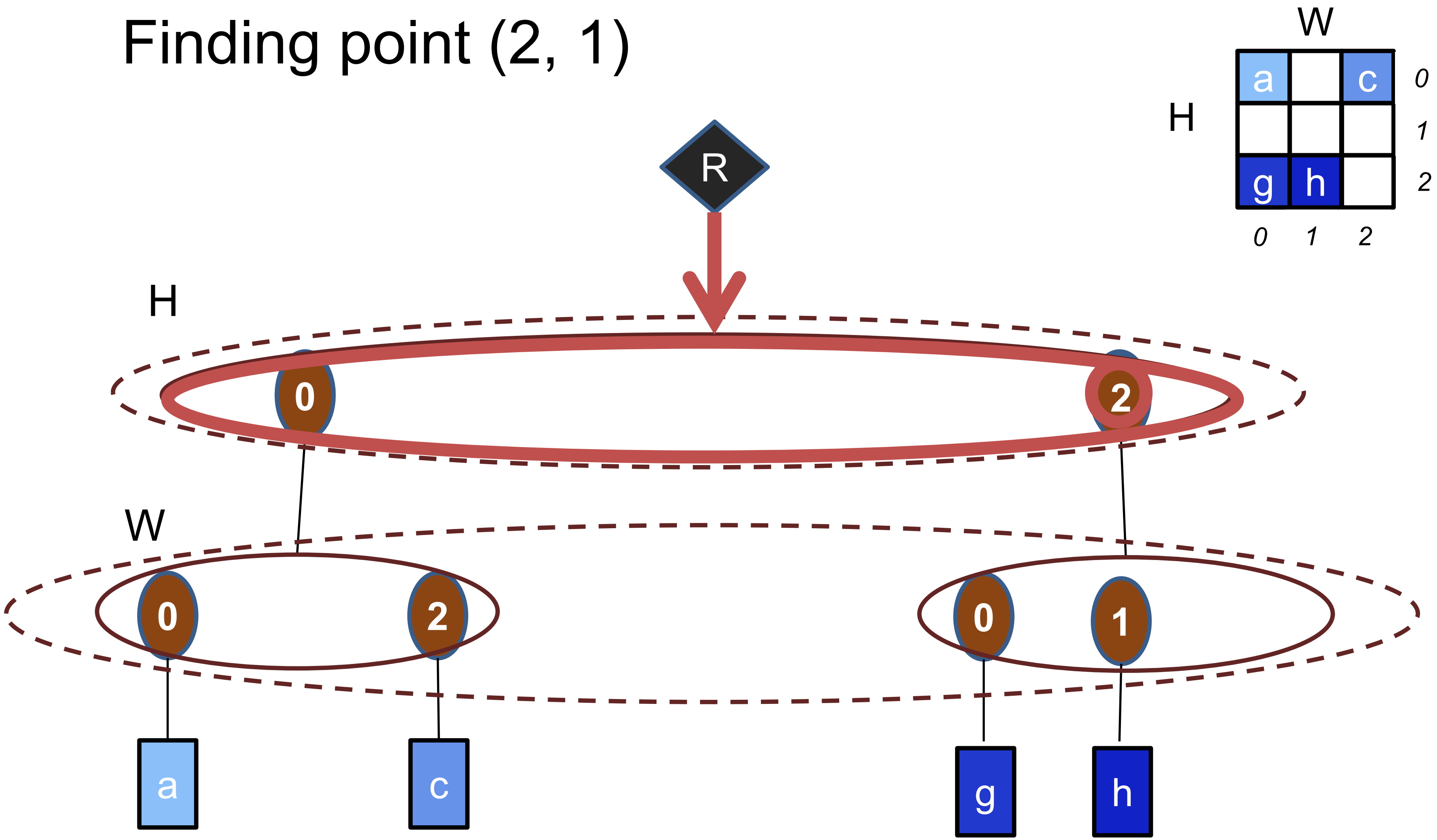
Fibertree Tensor Abstraction

Finding point (2, 1)



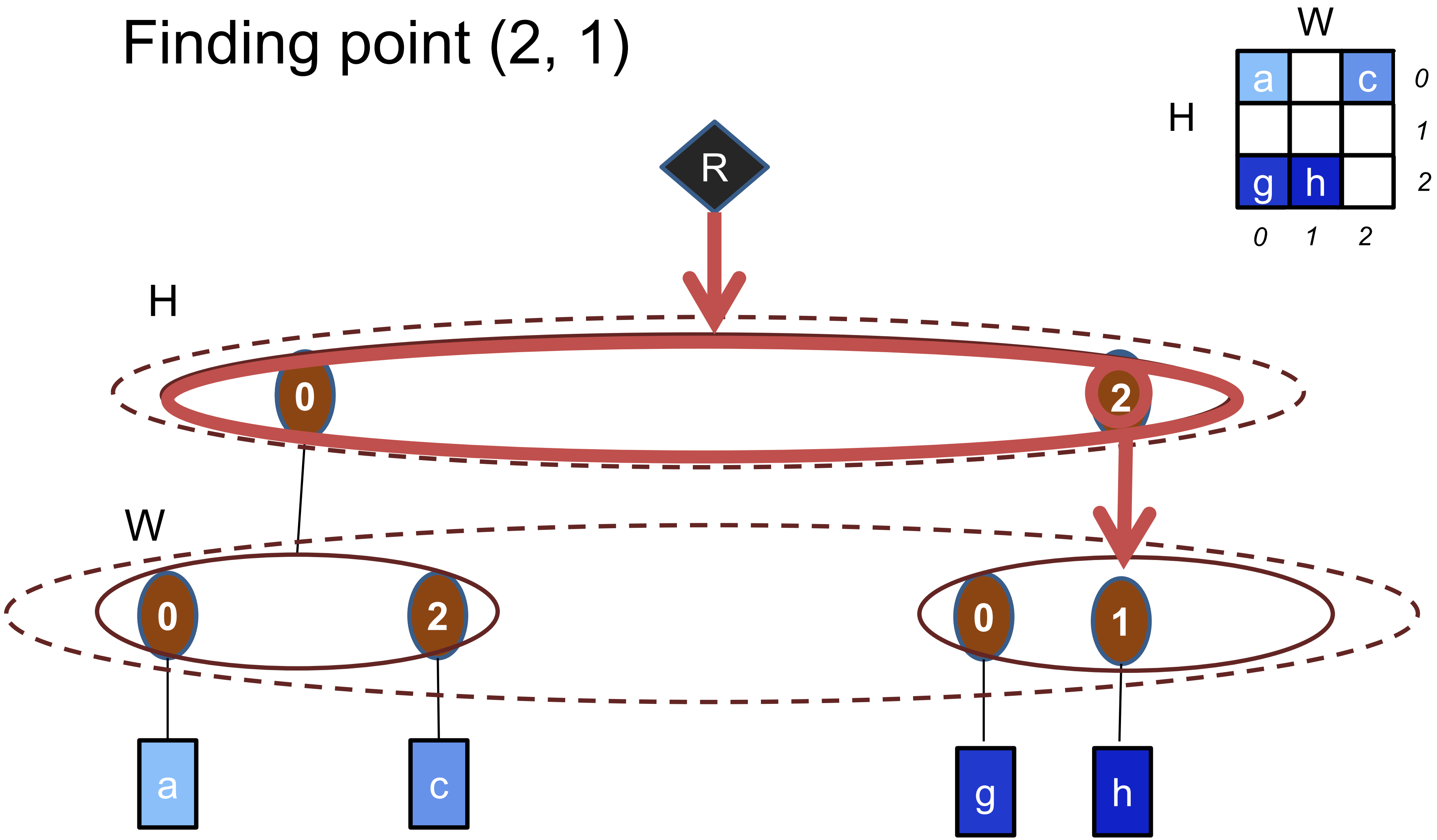
Fibertree Tensor Abstraction

Finding point (2, 1)



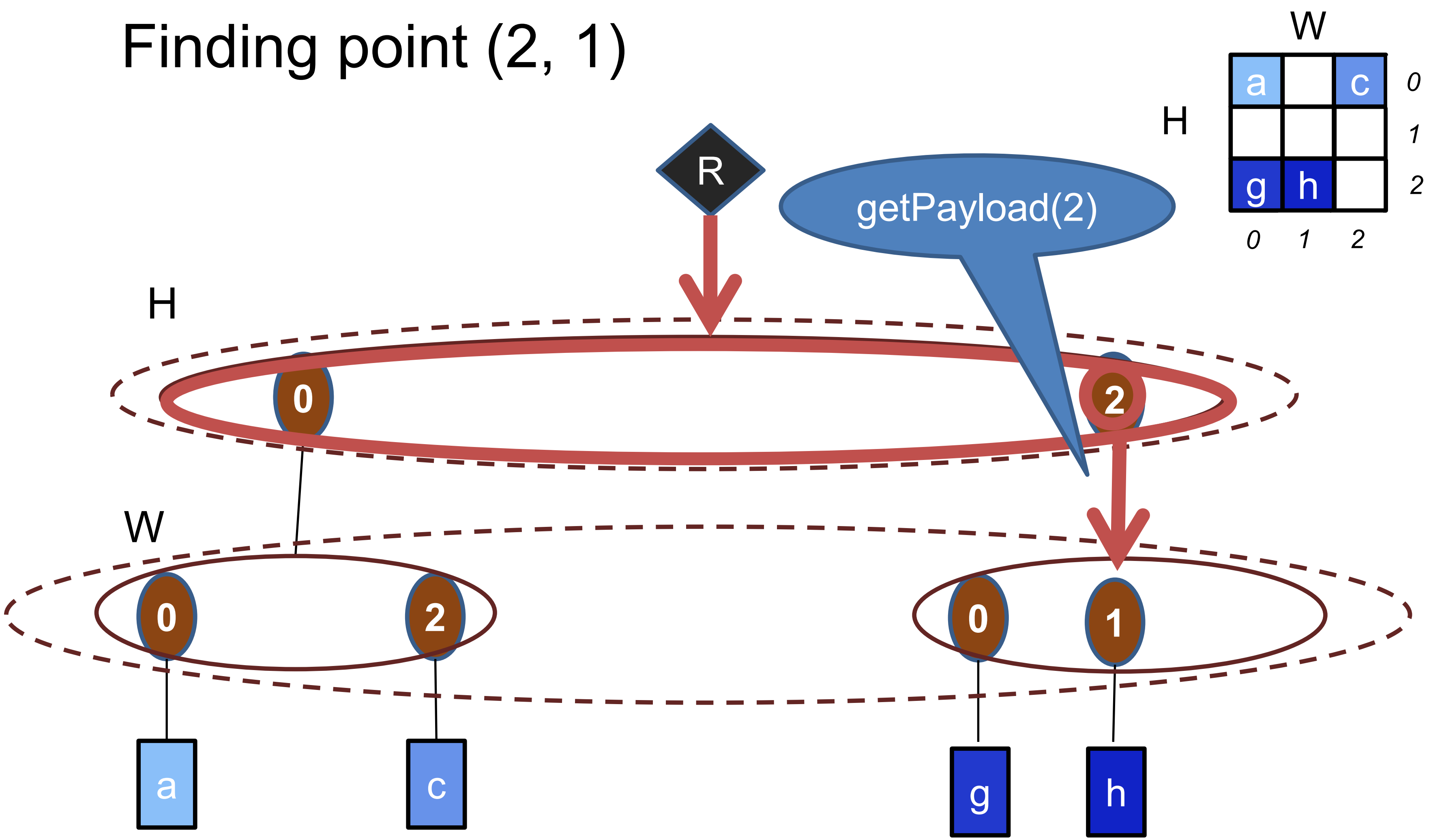
Fibertree Tensor Abstraction

Finding point (2, 1)



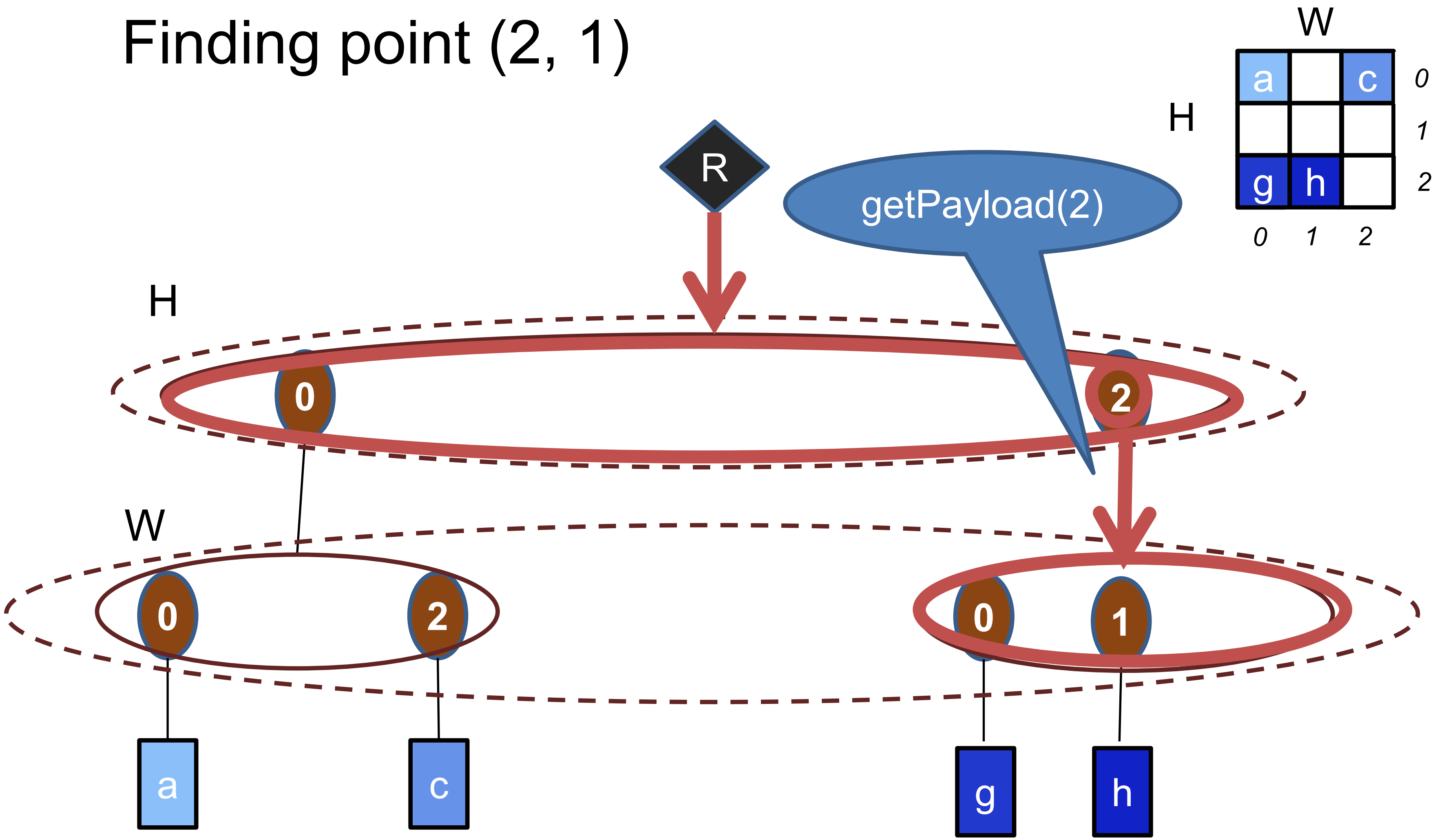
Fibertree Tensor Abstraction

Finding point (2, 1)



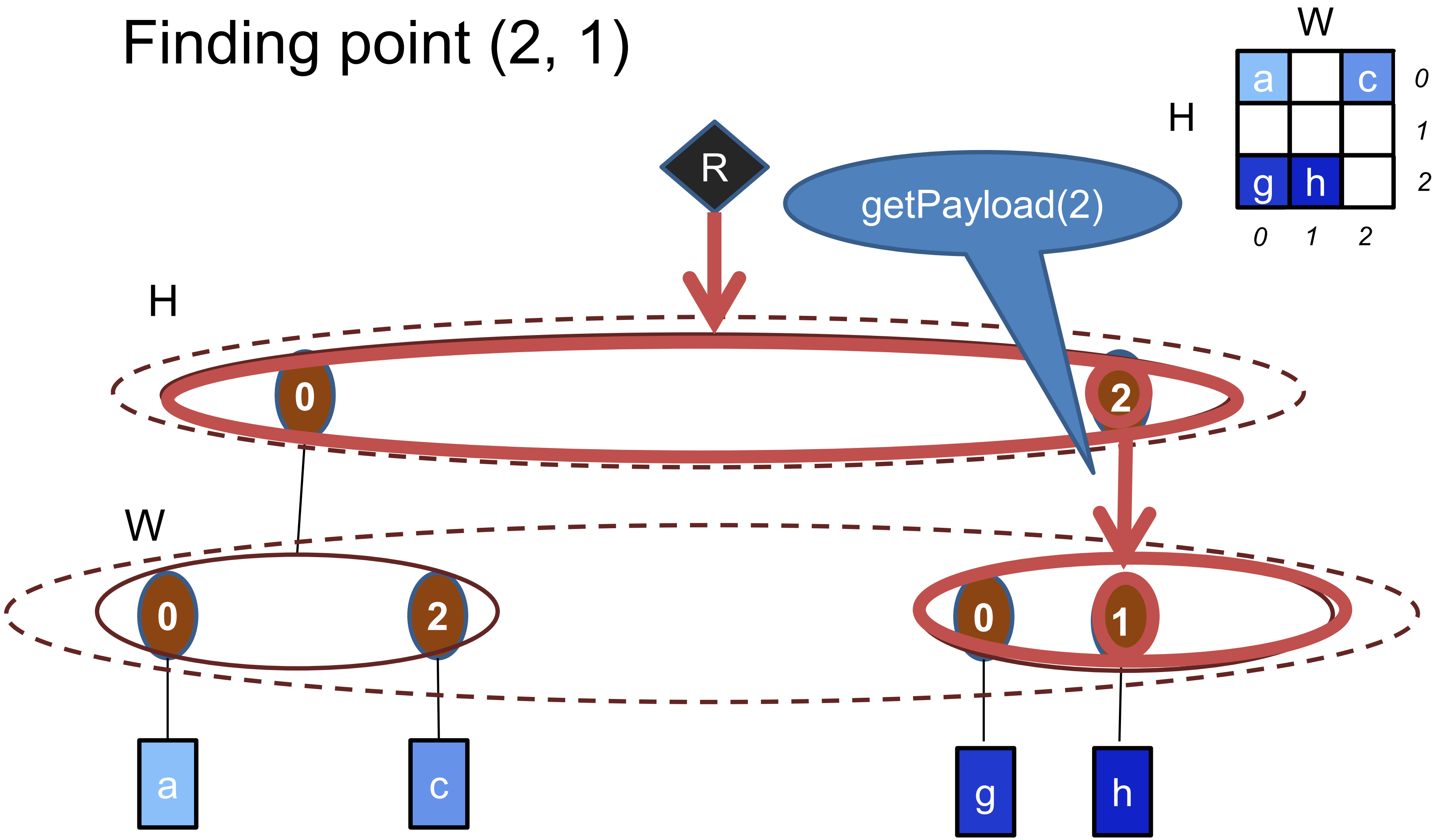
Fibertree Tensor Abstraction

Finding point (2, 1)



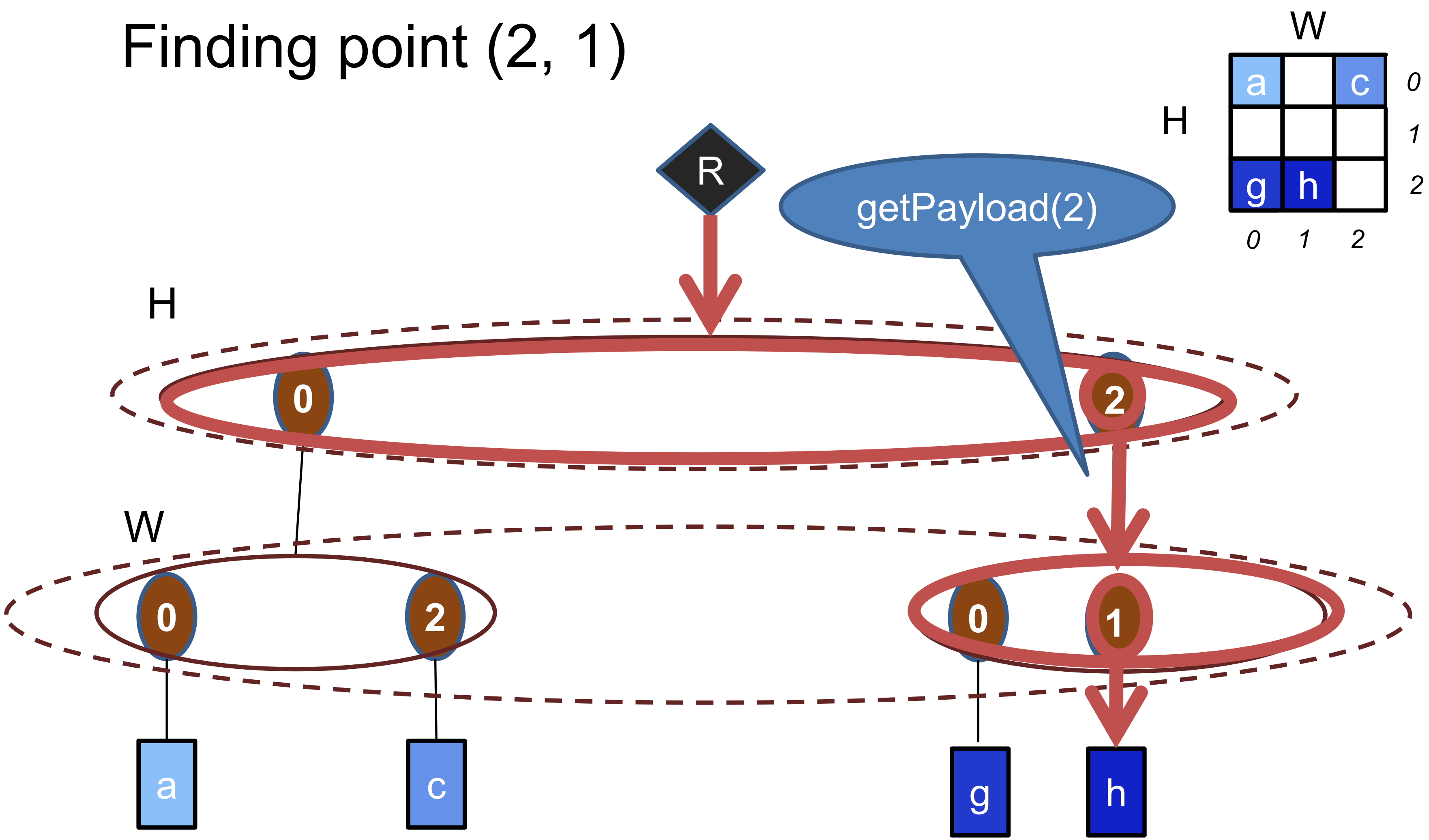
Fibertree Tensor Abstraction

Finding point (2, 1)



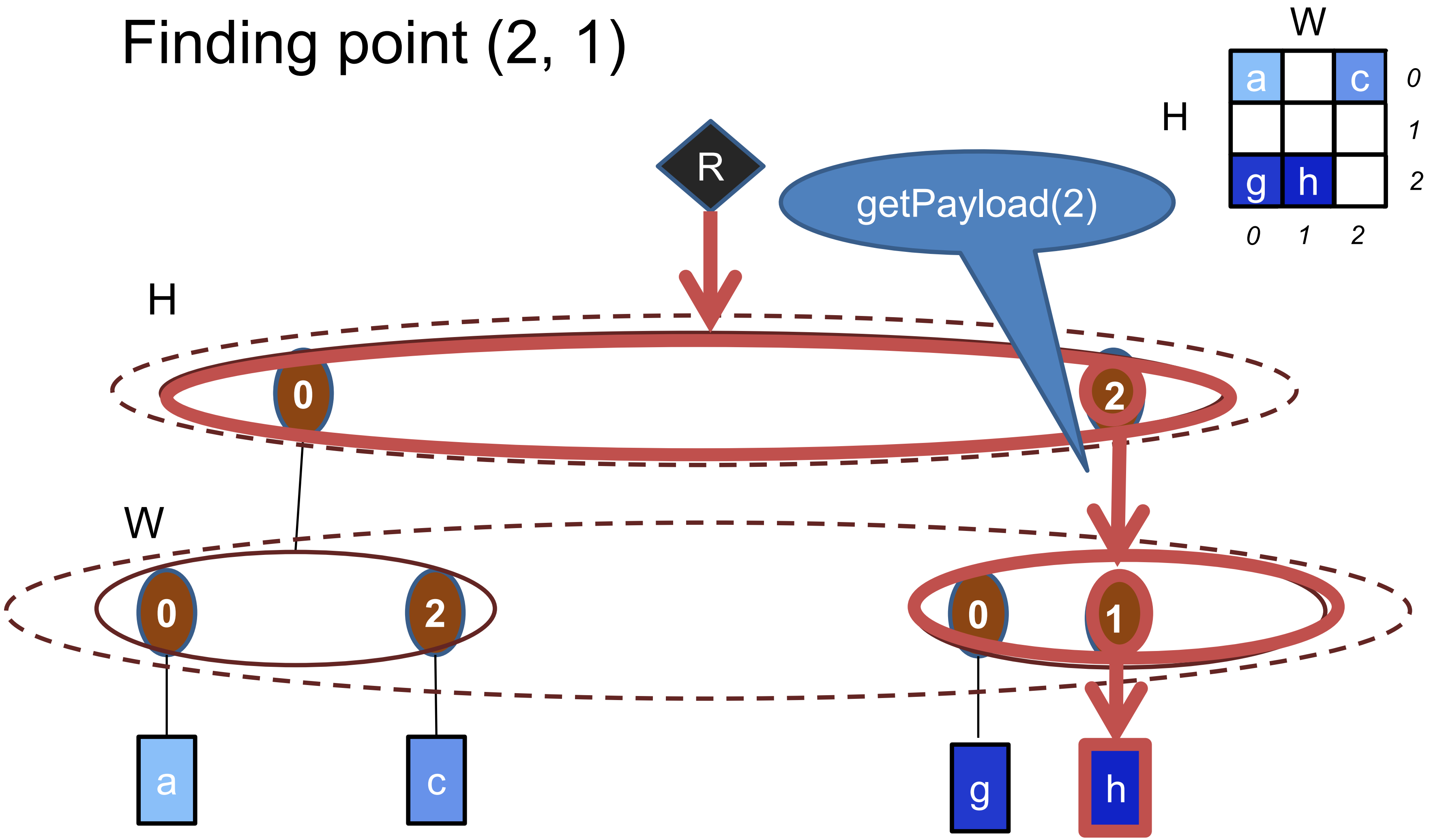
Fibertree Tensor Abstraction

Finding point (2, 1)



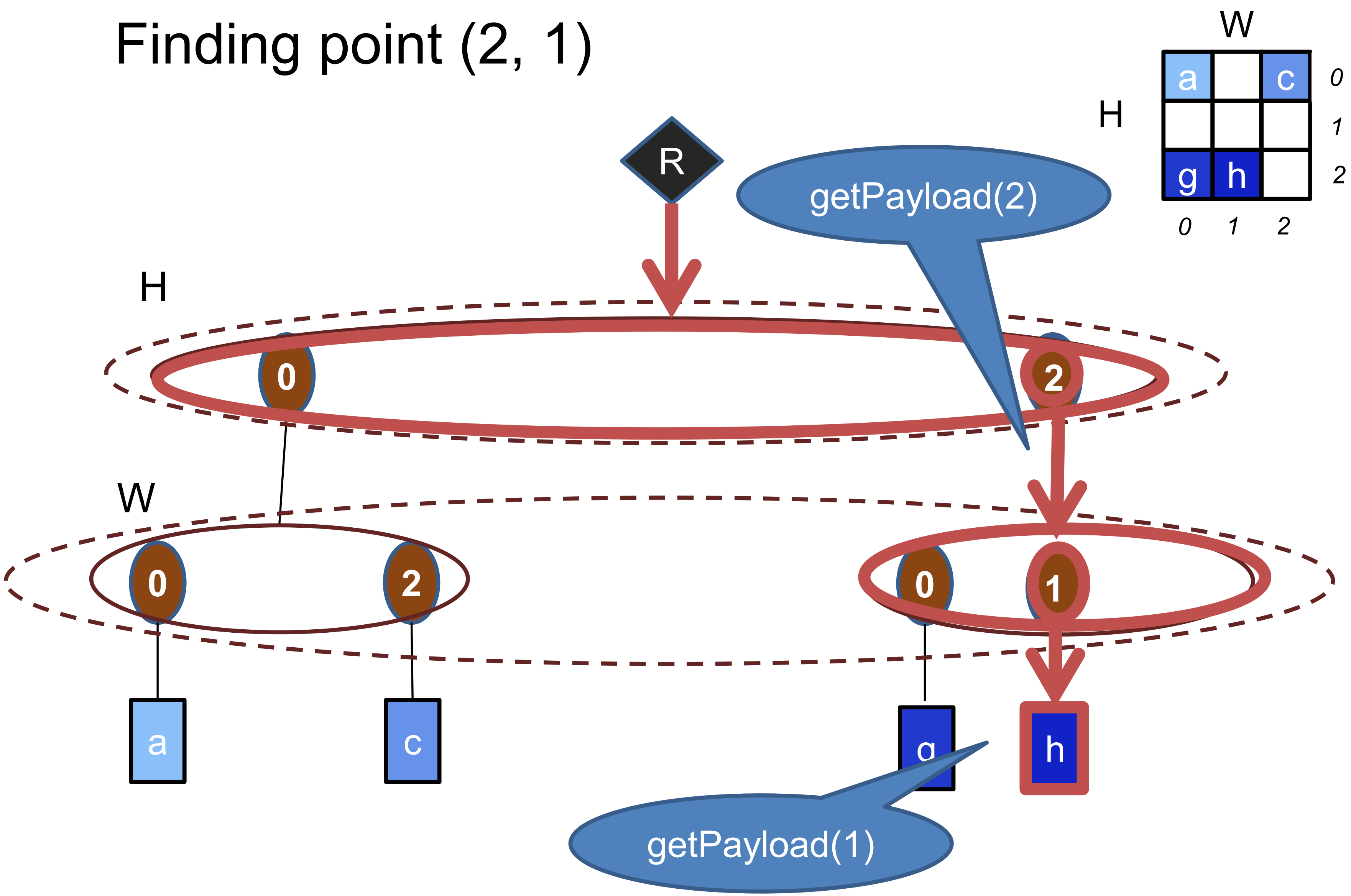
Fibertree Tensor Abstraction

Finding point (2, 1)



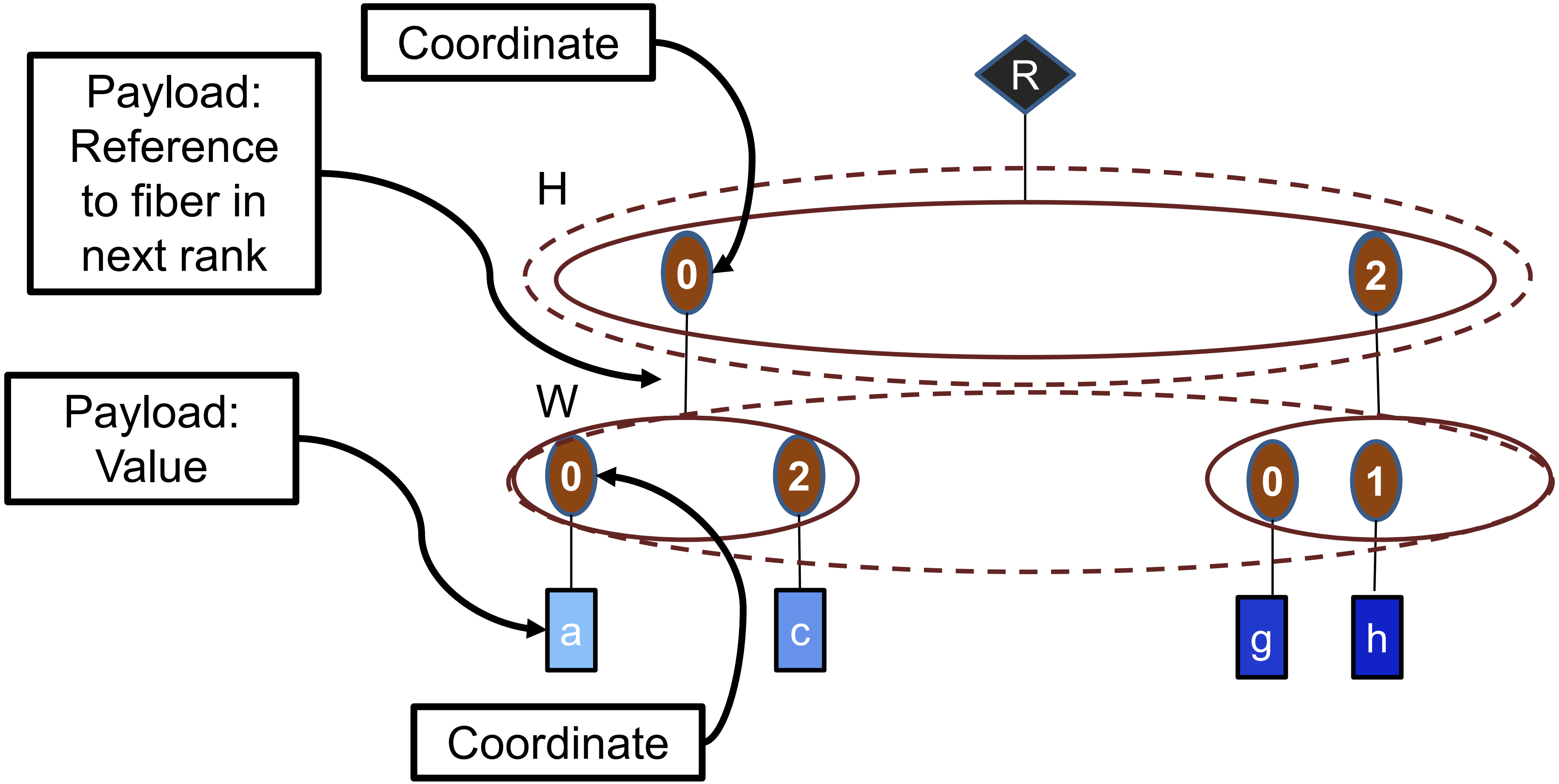
Fibertree Tensor Abstraction

Finding point (2, 1)



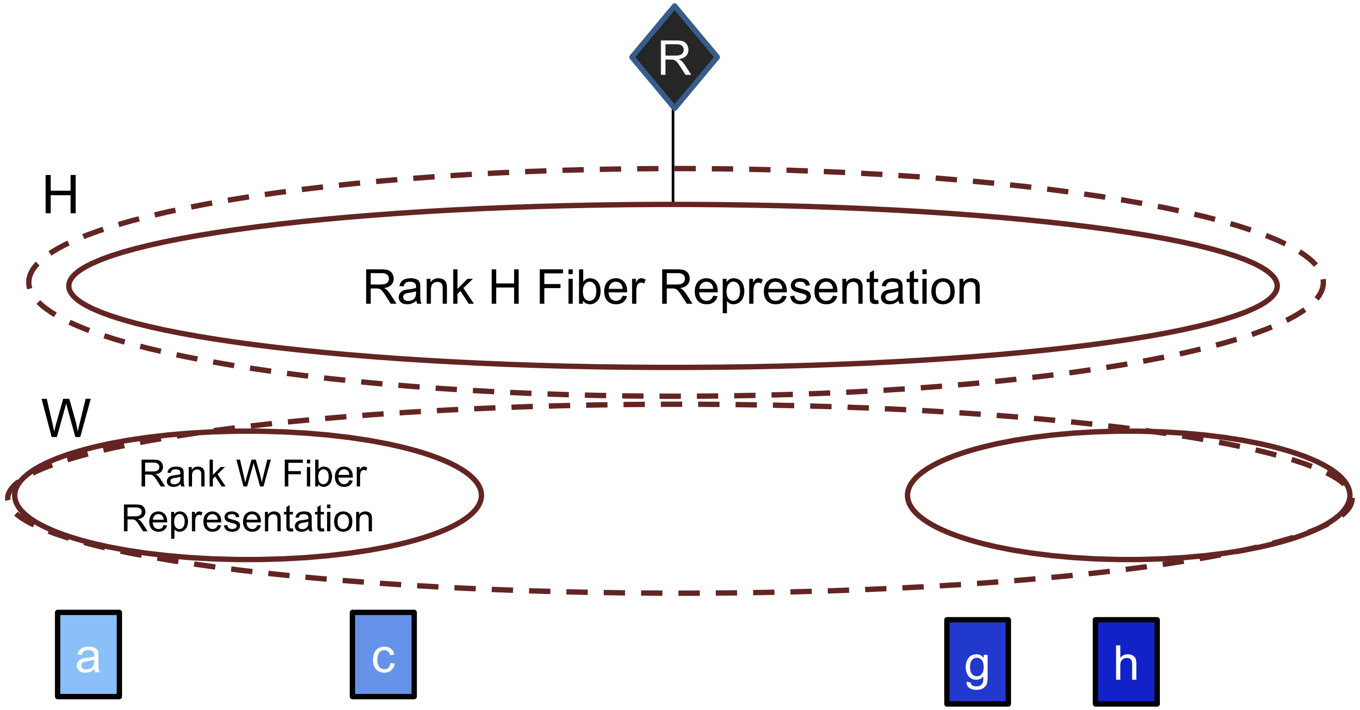
Information in a Fiber

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



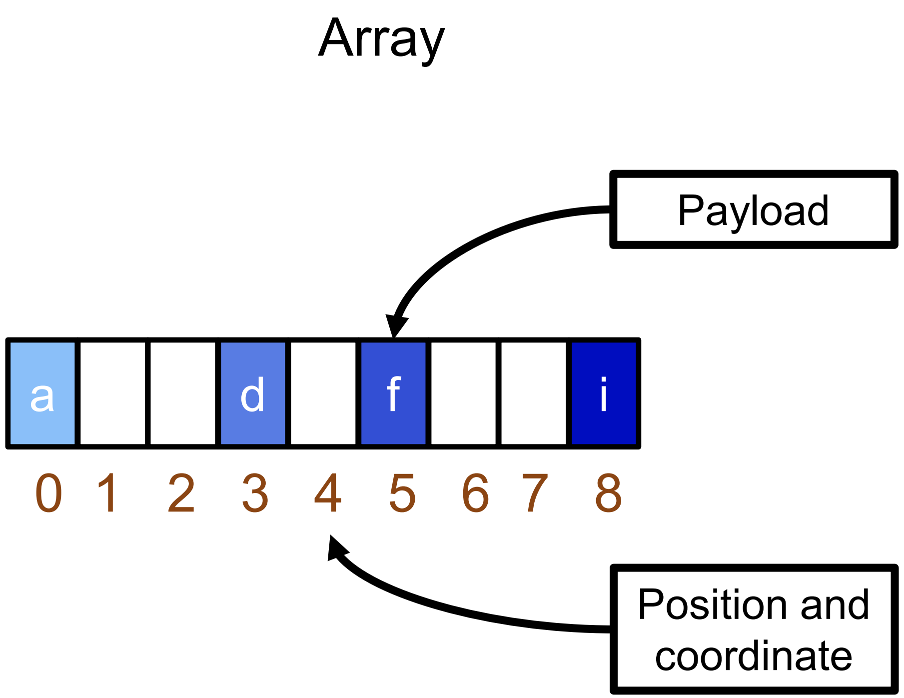
Information in a Fiber

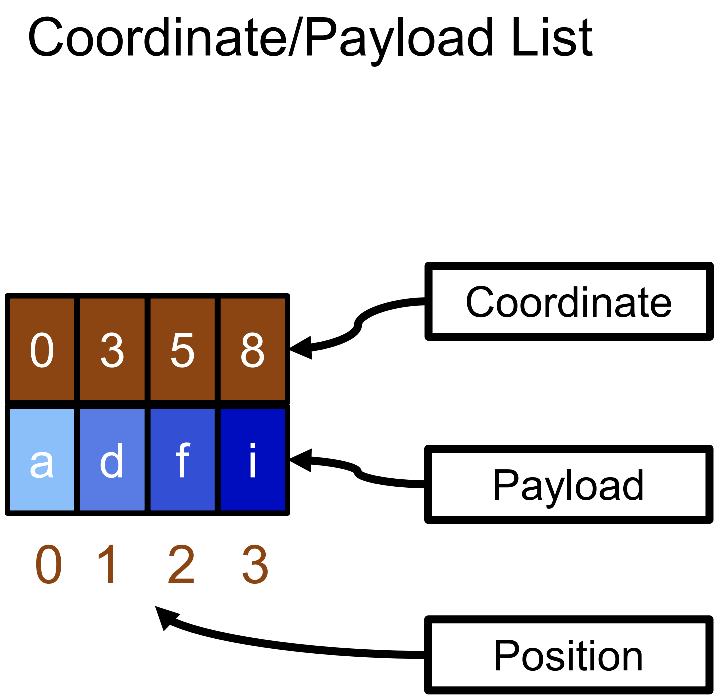
Method: maybe(payload) = fiber.getPayload(coordinate)



Example Fiber Representations

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





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.

Implicit Coordinates: RLE

Example

Input: **0**, **0**, 12, **0**, **0**, **0**, **0**, 53, **0**, **0**, 22

Method 1: Run Length Coding

Rather than send zero, send "run length" of zeros

e.g., 5 bits for run length and 16 bits for non-zero value

Output: $\underbrace{\textbf{2},}_{\text{5b}}\ \underbrace{12,}_{\text{16b}}$ **4**, 53, **2**, 22

Occupancy

Total Number of Bits: (5+16) * 3 = 63

Implicit Coordinates: Significance Map

Example

Input: **0**, **0**, 12, **0**, **0**, **0**, **0**, 53, **0**, **0**, 22

Method 2: Significance Map Coding
(a variant of this is referred to as bitmask coding)

Send one bit to indicate if significant (i.e., non-zero);
if significant, send 16 bits for non-zero value

Output: $\underbrace{0, 0, 1,}_{3b} \underbrace{12,}_{16b}$ 0, 0, 0, 0, 1, 53, 0, 0, 1, 22

Shape

Total Number of Bits: (1*11) + (16*3) = 11 + 48 = 59

Occupancy

How does this compare to Run Length Coding? Better

Implicit Coordinates: Huffman Encoding

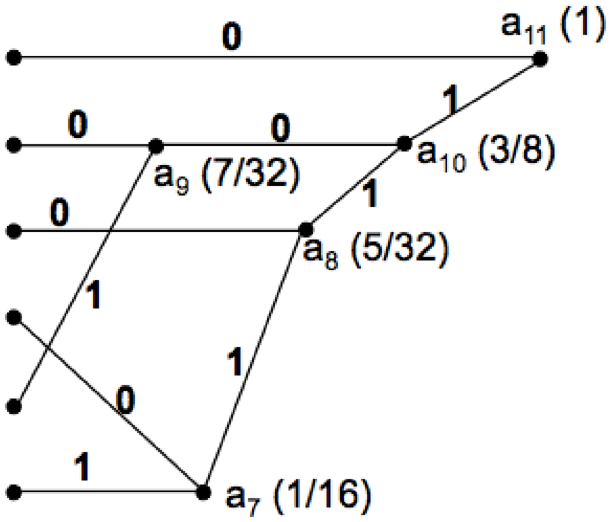
Example

Input: **0**, **0**, 12, **0**, **0**, **0**, **0**, 53, **0**, **0**, 22

<u>Method 3</u>: Huffman Coding

Assign number of bits based on probability of occurrence

Message	Codeword	Probability
a_1	0	p_1 (5/8)
a_2	100	p_2 (3/32)
a_3	110	p_3 (3/32)
a_4	1110	p_4 (1/32)
a_5	101	p_5 (1/8)
a_6	1111	p_6 (1/32)



Assign codewords directly to values or to values and run-lengths

Quantization and Compression

- Quantization + Significance Map Coding

  Example:

  Value: 16’b0 → Compressed Code: {1’b0}

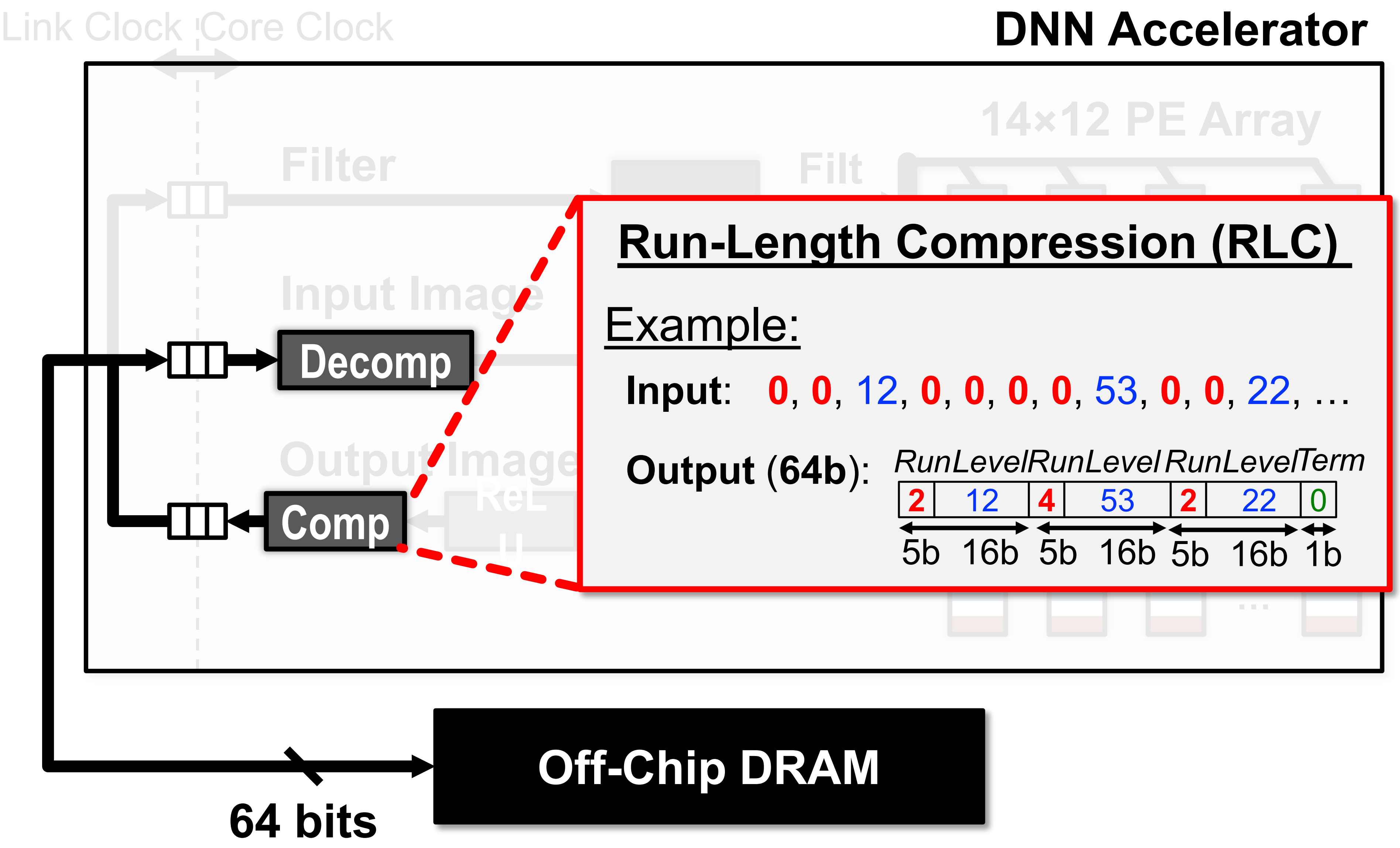
  Value: 16’bx → Compressed Code: {1’b1, 16’bx}

- Tested on AlexNet → 2× overall BW Reduction

Layer	Filter / Image bits (0%)	Filter / Image BW Reduc.	IO / HuffIO (MB/frame)	Voltage (V)	MMACs/ Frame	Power (mW)	Real (TOPS/W)
General CNN	16 (0%) / 16 (0%)	1.0x	—	1.1	—	**288**	**0.3**
AlexNet l1	7 (21%) / 4 (29%)	1.17x / 1.3x	1 / 0.77	0.85	105	85	0.96
AlexNet l2	7 (19%) / 7 (89%)	1.15x / **5.8x**	3.2 / 1.1	0.9	224	55	1.4
AlexNet l3	8 (11%) / 9 (82%)	1.05x / 4.1x	6.5 / 2.8	0.92	150	77	0.7
AlexNet l4	9 (04%) / 8 (72%)	1.00x / 2.9x	5.4 / 3.2	0.92	112	95	0.56
AlexNet l5	9 (04%) / 8 (72%)	1.00x / 2.9x	3.7 / 2.1	0.92	75	95	0.56
Total / avg.	—	—	19.8 / **10**	—	—	**76**	**0.94**
LeNet-5 l1	3 (35%) / 1 (87%)	1.40x / 5.2x	0.003 / 0.001	0.7	0.3	25	1.07
LeNet-5 l2	4 (26%) / 6 (55%)	1.25x / 1.9x	0.050 / 0.042	0.8	1.6	35	1.75
Total / avg.	—	—	0.053 / **0.043**	—	—	**33**	**1.6**

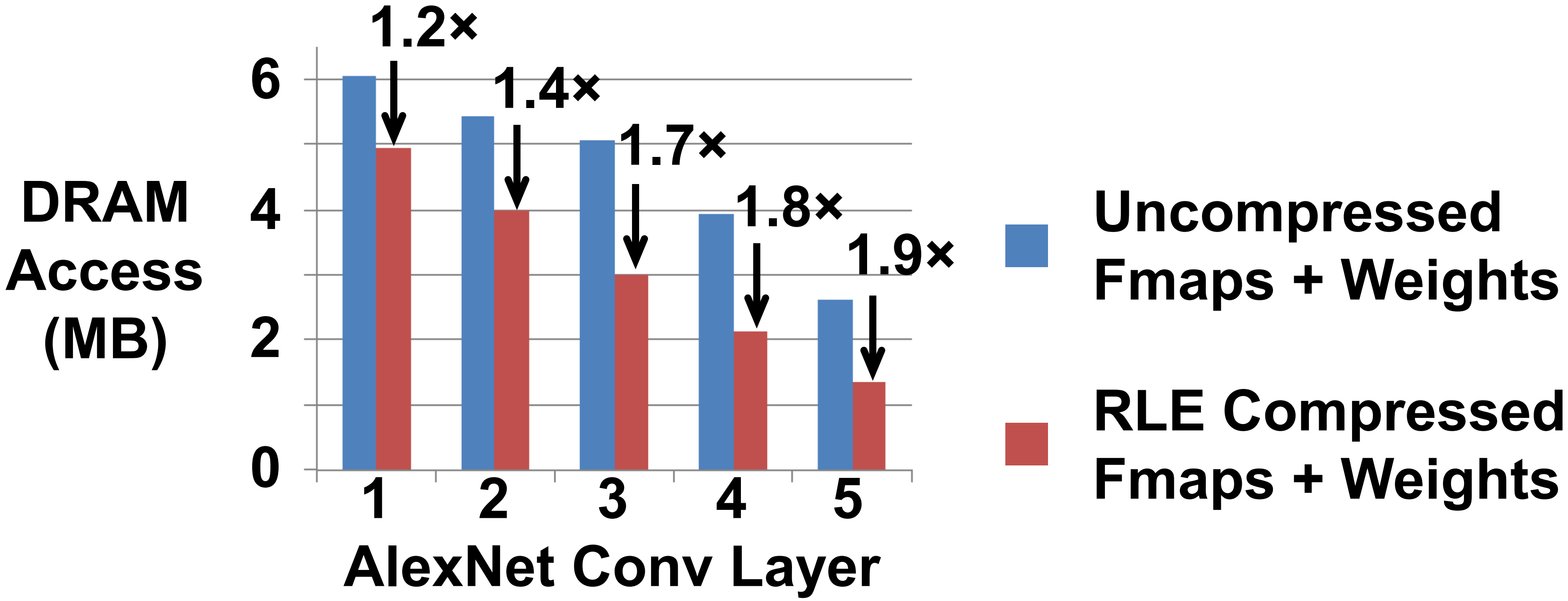
[**Moons**, *VLSI* 2016]

I/O Compression in Eyeriss



[Chen, *ISSCC* 2016]

Compression Reduces DRAM BW



Simple RLC within 5% - 10% of theoretical entropy limit

[**Chen**, *ISSCC* 2016]

From information theory,

$$\text{Entropy} \quad H = -\sum_{i=0}^{L-1} p_i \cdot \log_2 p_i$$

↑

minimum average number of bits required to code f

Compressed Implicit Coordinate Representations

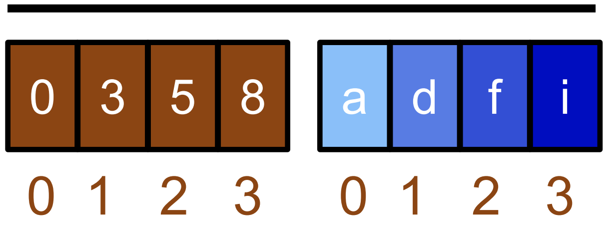
- "Empty" coordinate compression via zero-run encoding
  - Run-length coding (RLE)
    - (run-length of zeros, non-zero payload)…
  - Significance map coding
    - (flag to indicate if non-zero, non-zero payload)…
- Payload encoding
  - Fixed length payload
  - Variable length payload
    - E..g., Huffman coding
- Efficiency of different traversal patterns through the tensor is affected by encoding, e.g., finding the payload for a particular coordinate…

Compressed Explicit Coordinate Representations

- Coordinate list representation
  - Struct of arrays form

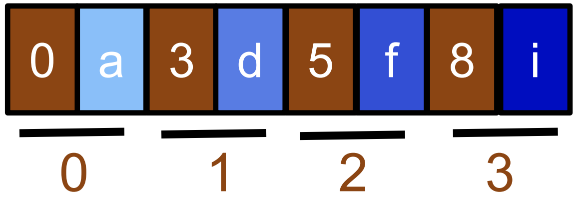
    (coordinate of non-zero value)…

    (non-zero payload)…



  - Array of structs form

    (coordinate of non-zero value, non-zero payload)…

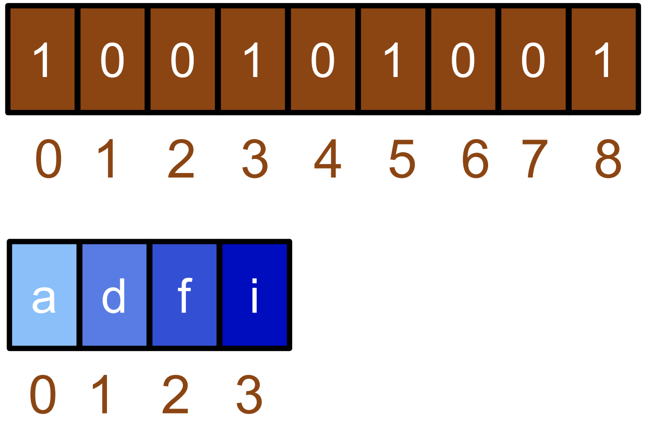


Black bar show scope of struct

- Payload encoding
  - Explicit
    - Immediate value
    - Pointer
  - Implicit
    - Offset of coordinate is offset of payload

More Explicit Coordinate Representations

- Coordinate Bitmask

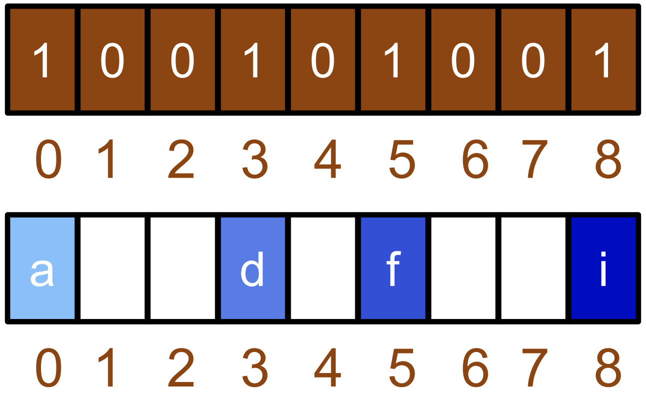


Any complexity with lookupPayload()?

May require a population count of the coordinate array.

Have we seen a representation like this?

Yes, the Eyeriss input activations used for gating were sort of like that….

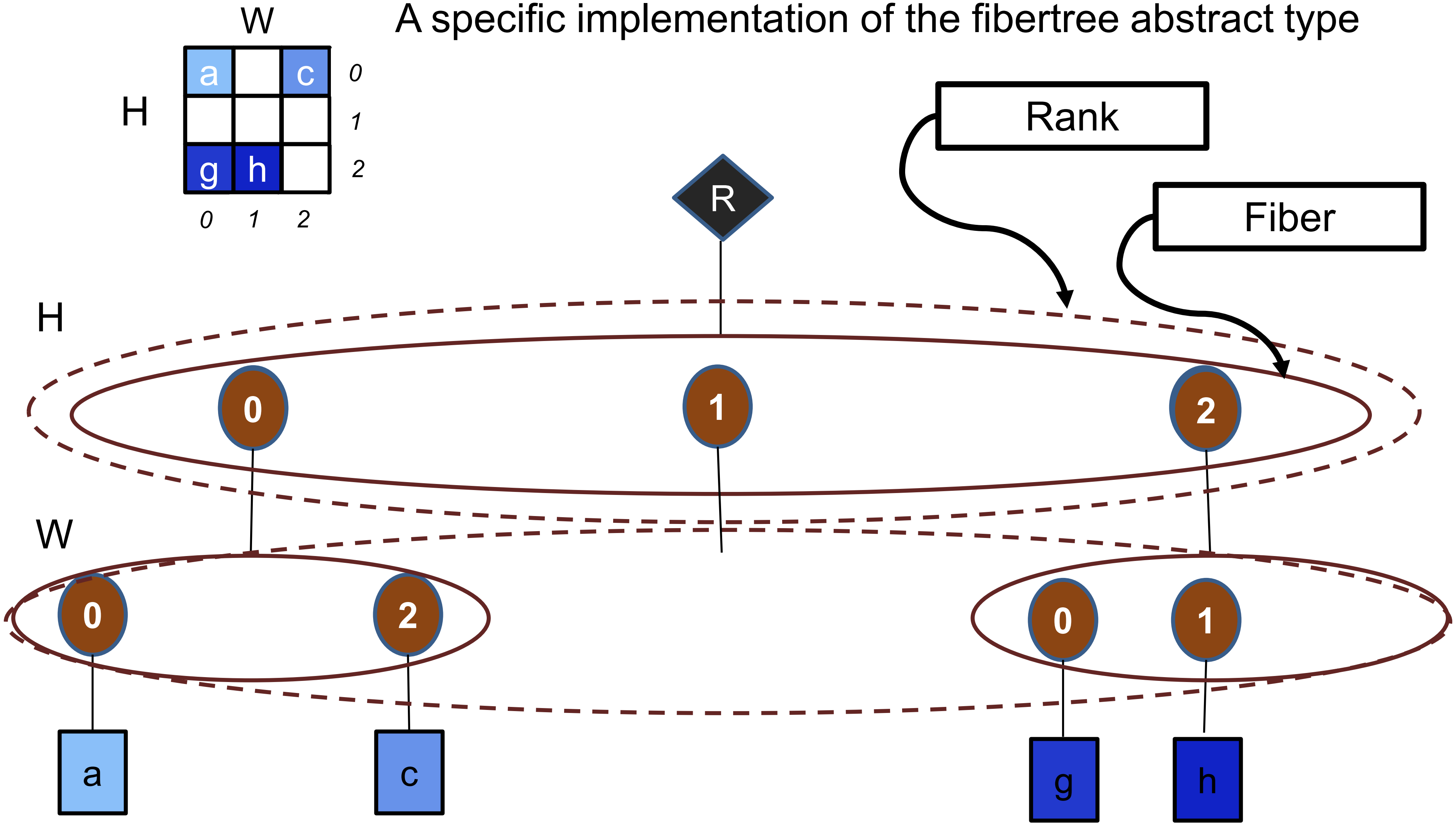


Is this useful even with no compression?

Yes, cheap check for zeros.

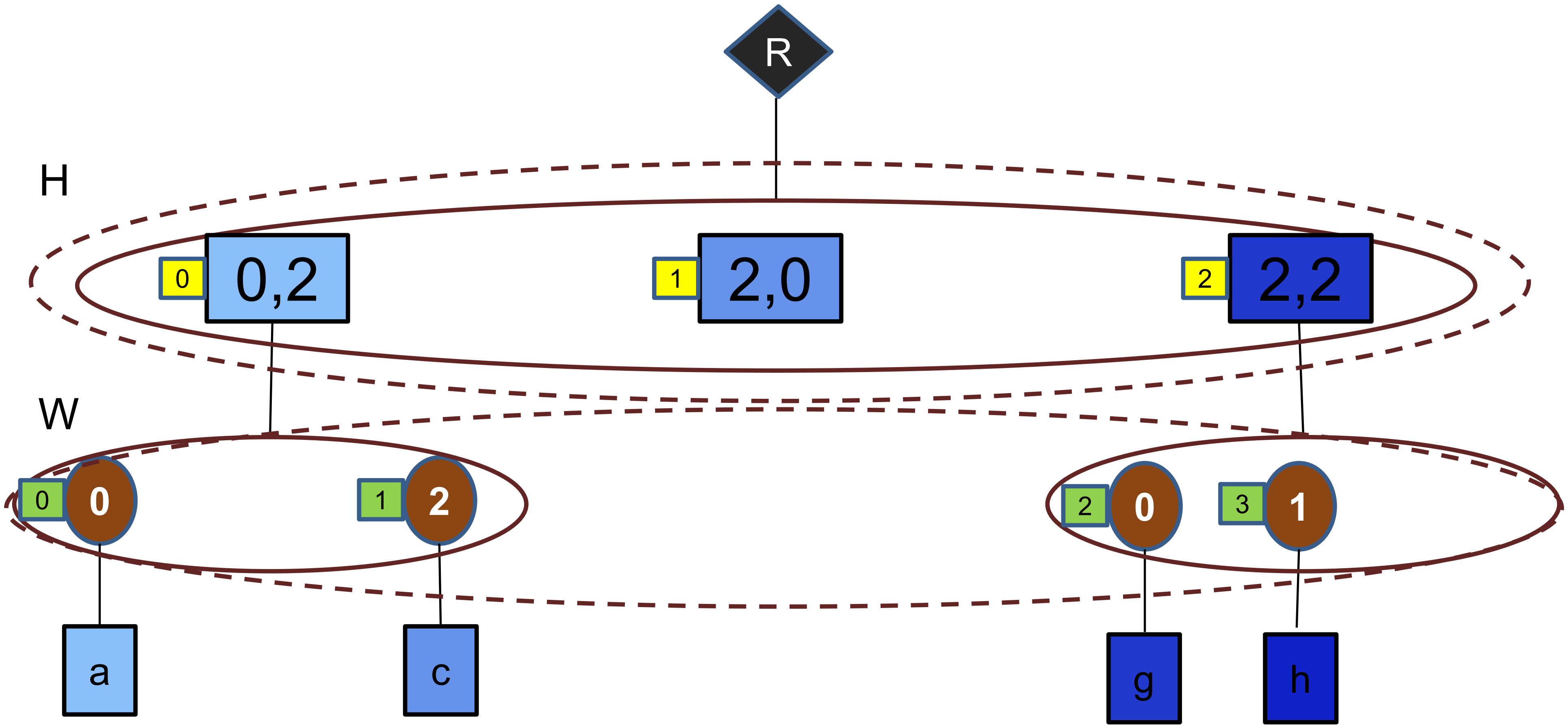
Uncompressed/Compressed Representation

A specific implementation of the fibertree abstract type



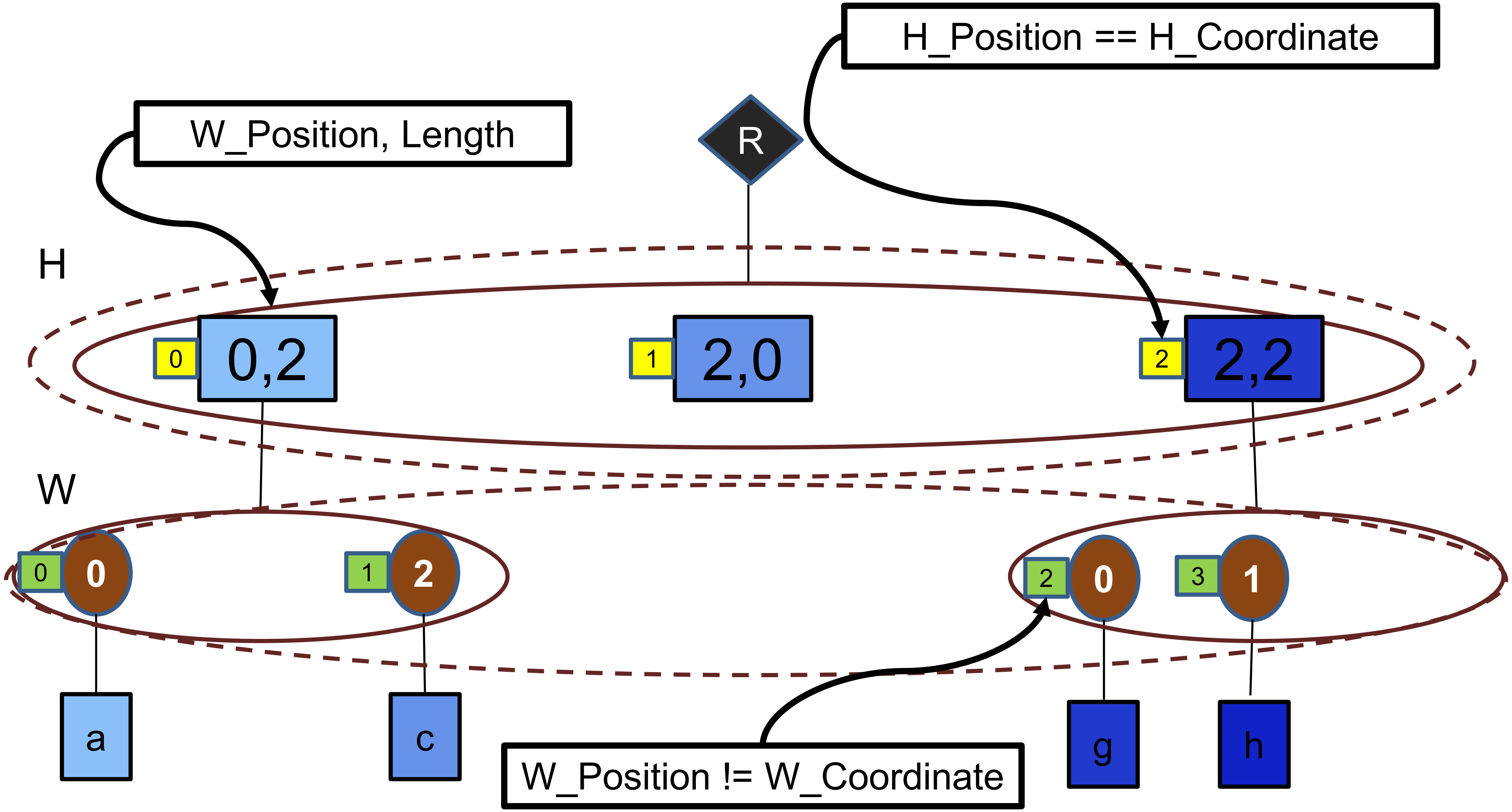
Uncompressed/Compressed Representation

A specific implementation of the fibertree abstract type



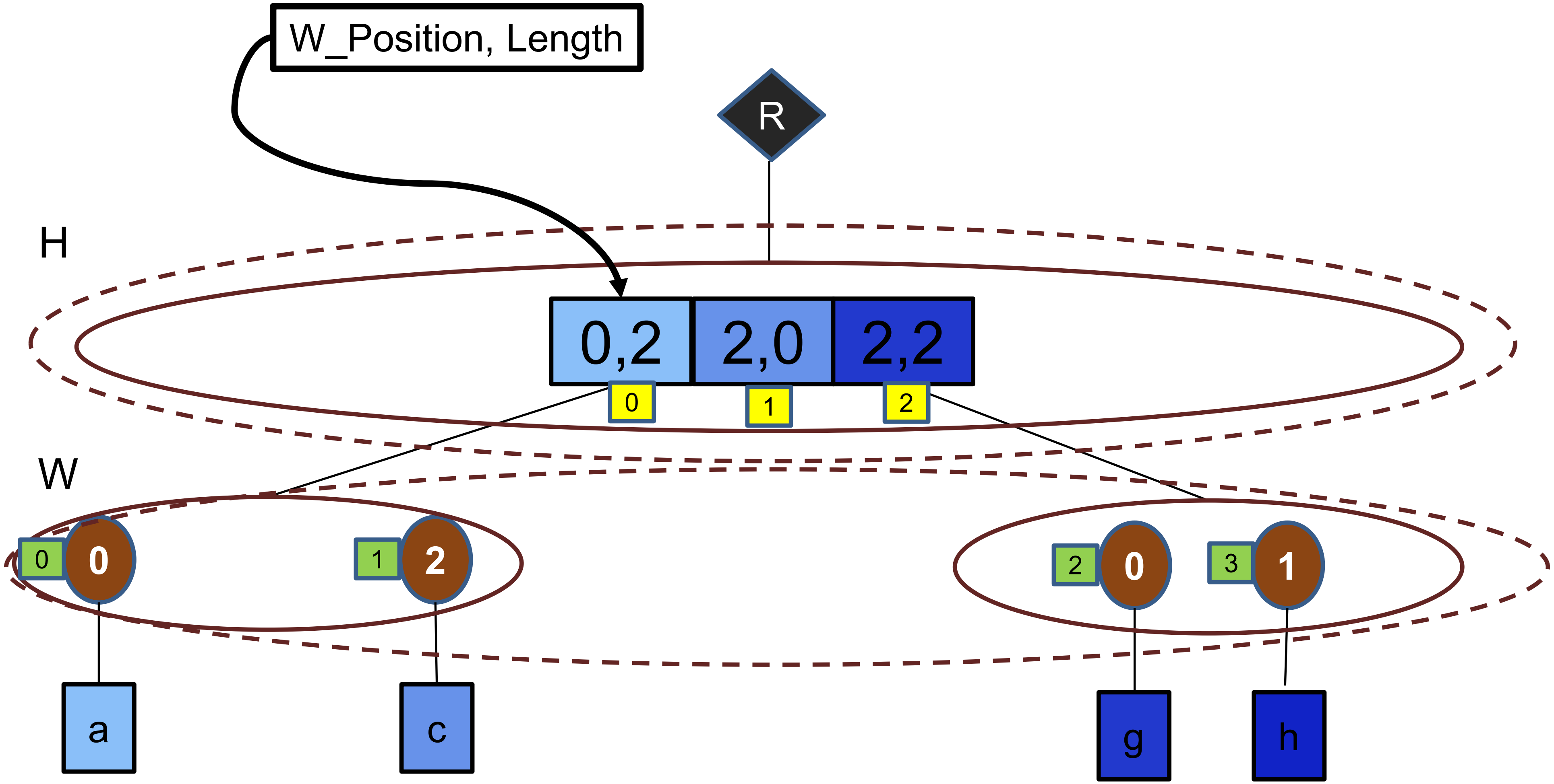
Uncompressed/Compressed Representation

A specific implementation of the fibertree abstract type



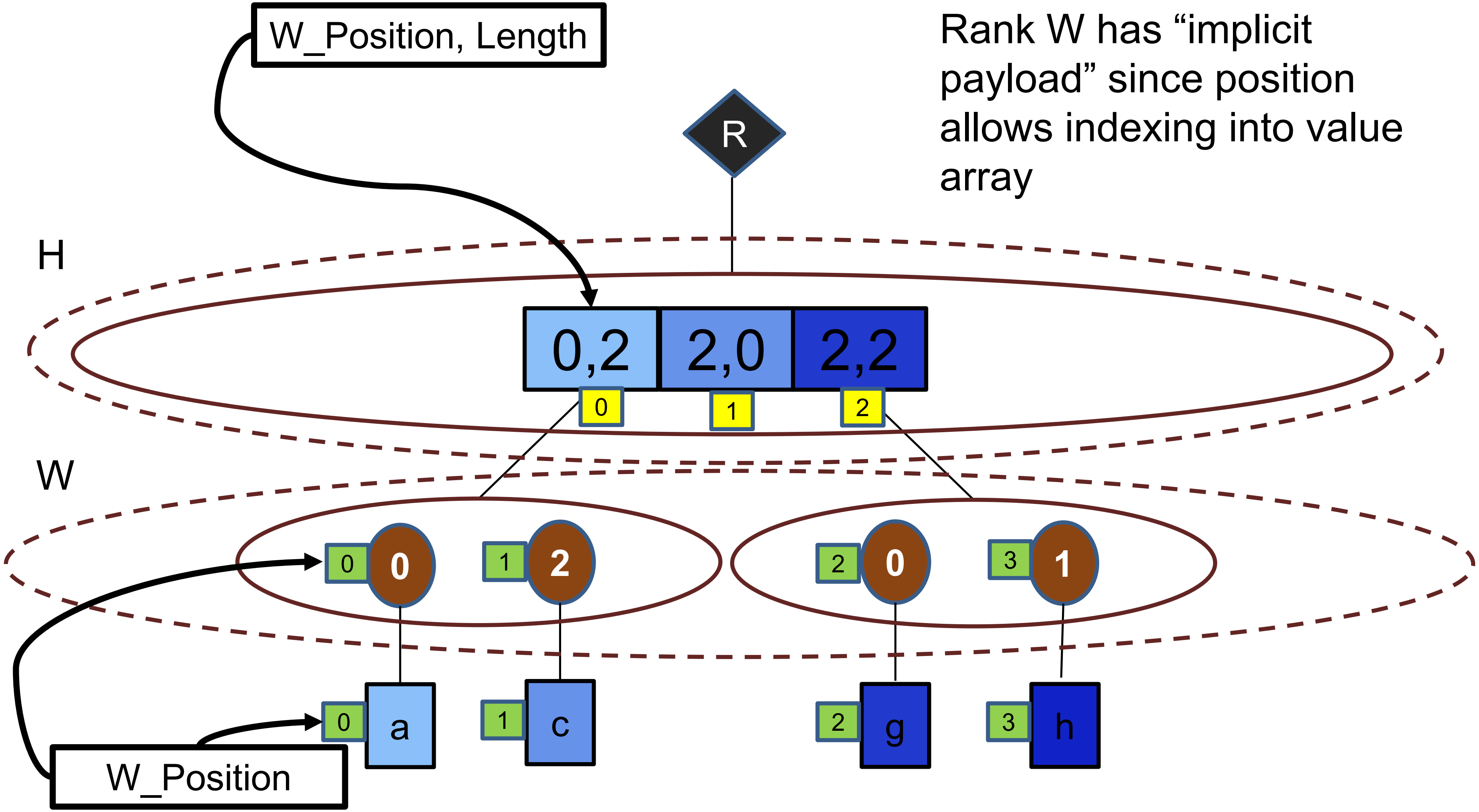
Uncompressed/Compressed Representation

A specific implementation of the fibertree abstract type



Uncompressed/Compressed Representation

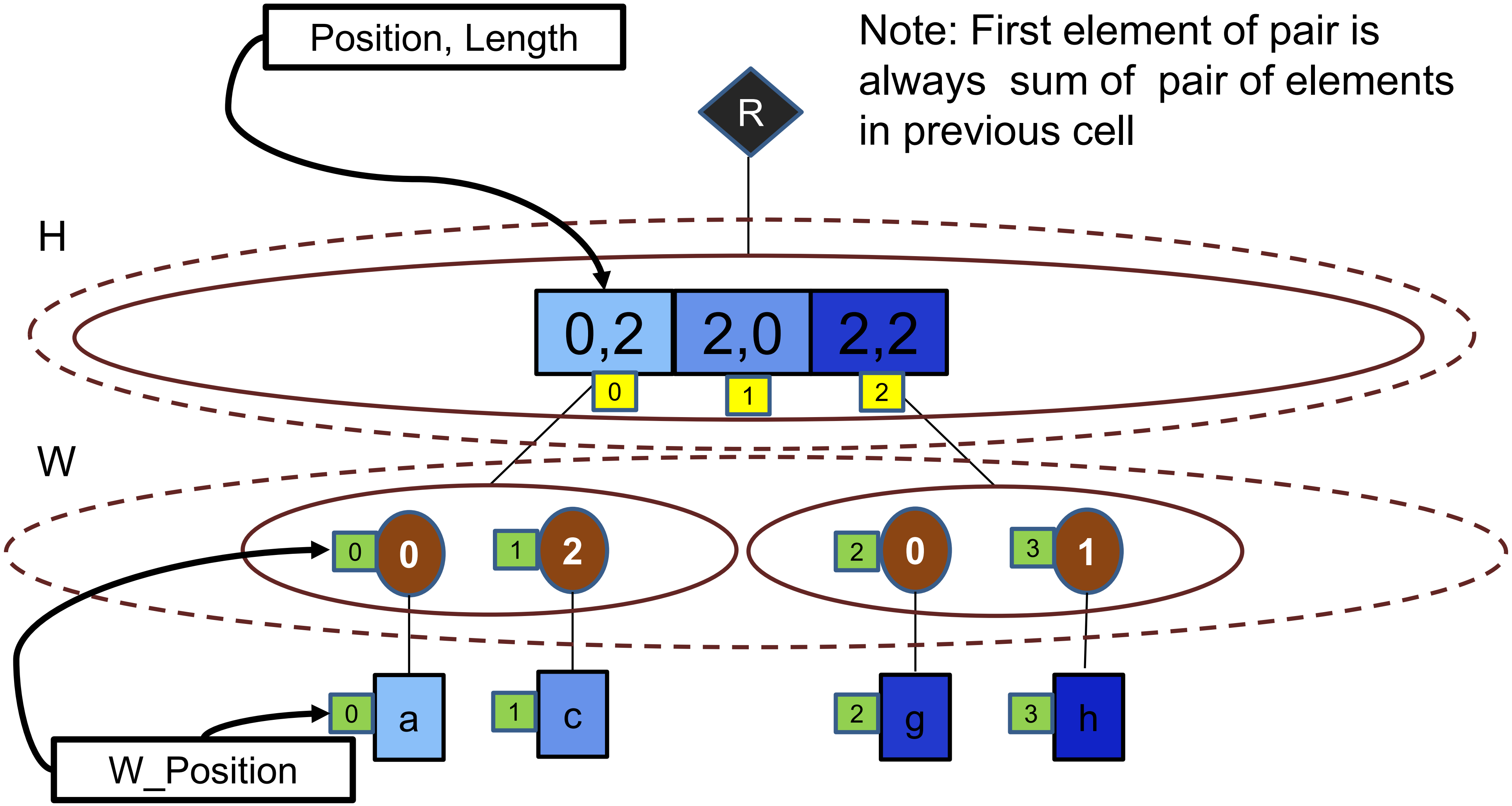
A specific implementation of the fibertree abstract type



Rank W has “implicit payload” since position allows indexing into value array

Uncompressed/Compressed Representation

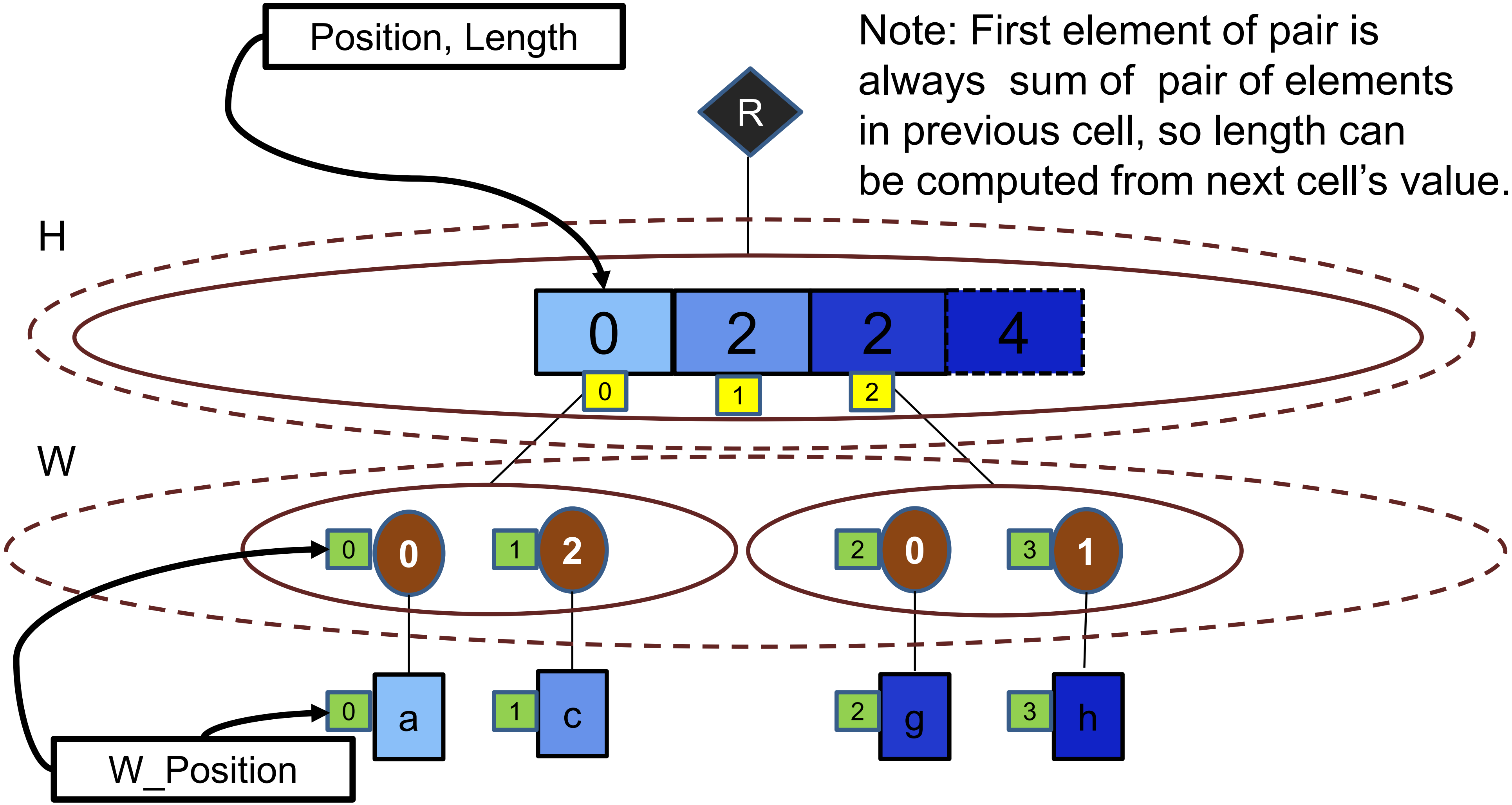
A specific implementation of the fibertree abstract type



Uncompressed/Compressed Representation

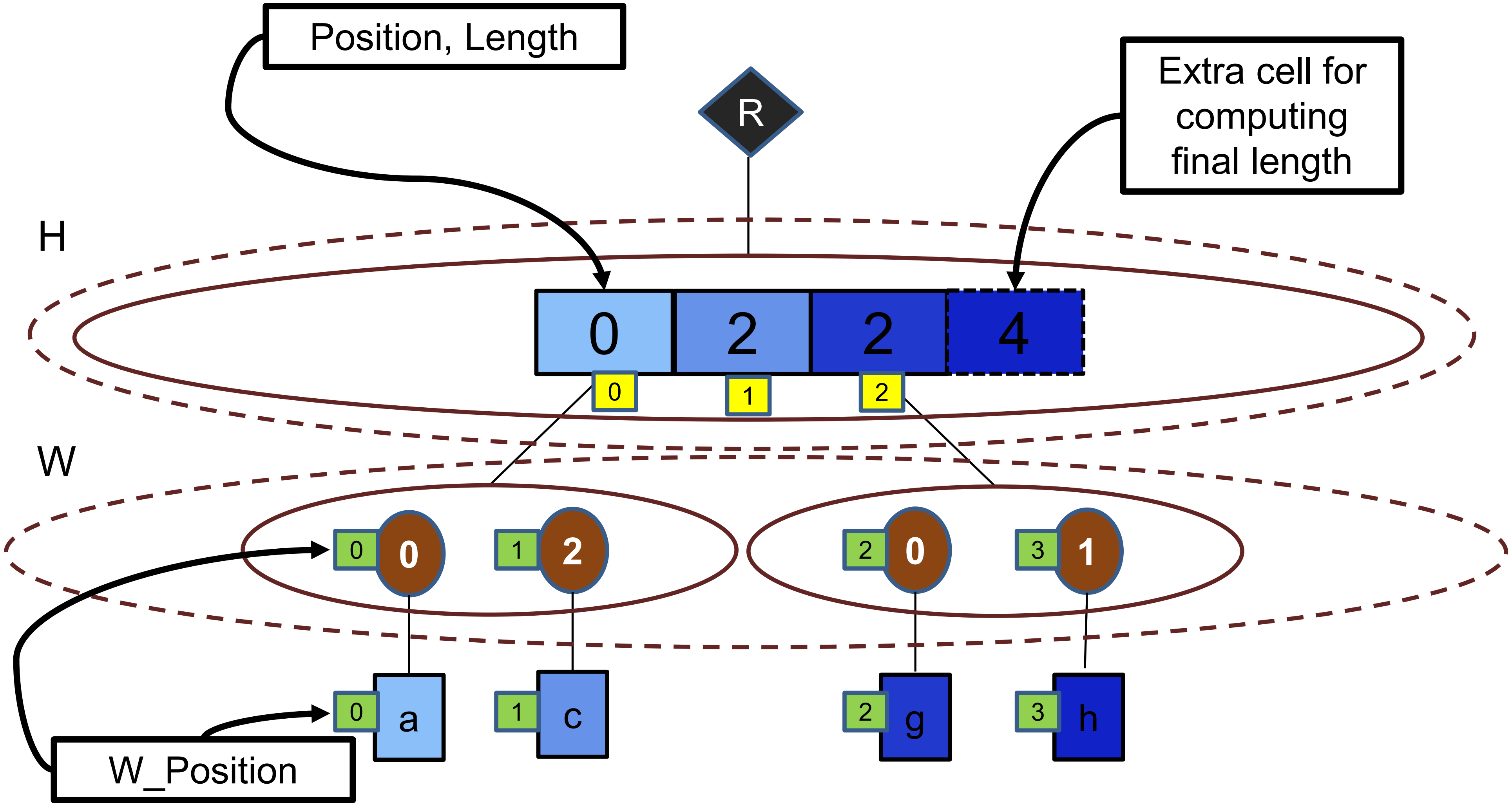
A specific implementation of the fibertree abstract type

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

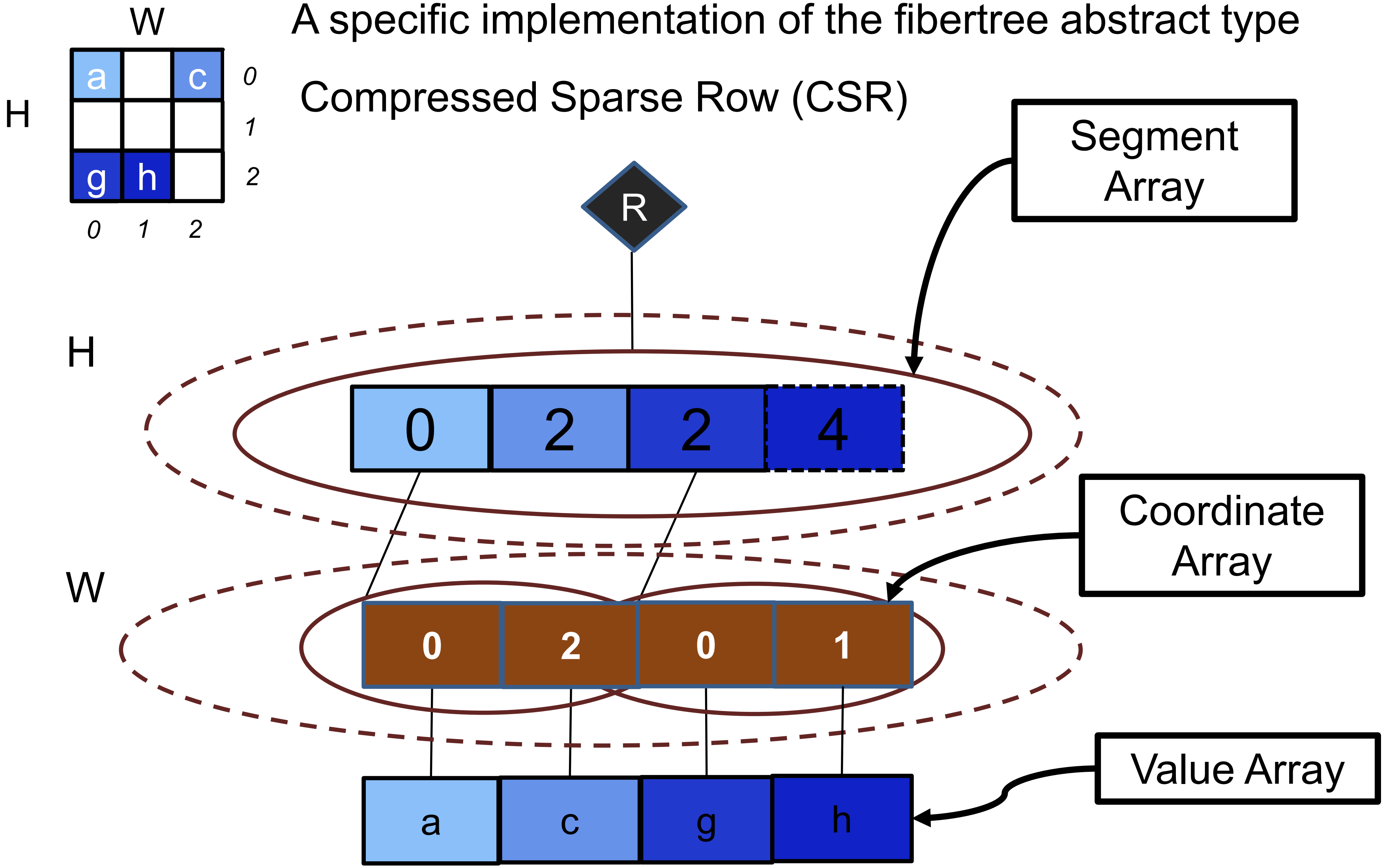
A specific implementation of the fibertree abstract type



Uncompressed/Compressed Representation

A specific implementation of the fibertree abstract type

Compressed Sparse Row (CSR)



Explicit Coordinate Representations

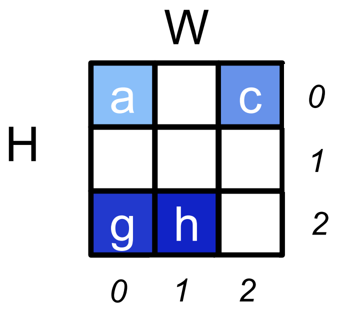
- Coordinate/Payload list
  - (coordinate, non-zero payload)… (array of structs)
  - (coordinate)… , (non-zero payload)… (struct of arrays)
- Hash table (per fiber)
  - (coordinate -> payload) mapping
- Hash table (per rank)
  - (fiber_id, coordinate -> payload) mapping
- Bit vector of non-zero coordinates
  - Compressed or 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]
- Tagged union of any combination of previous types

Inspired by collaboration with Kjolstad in [Kjolstad, OOPSLA17], [Chou, OOPSLA18]

Notation for CSR

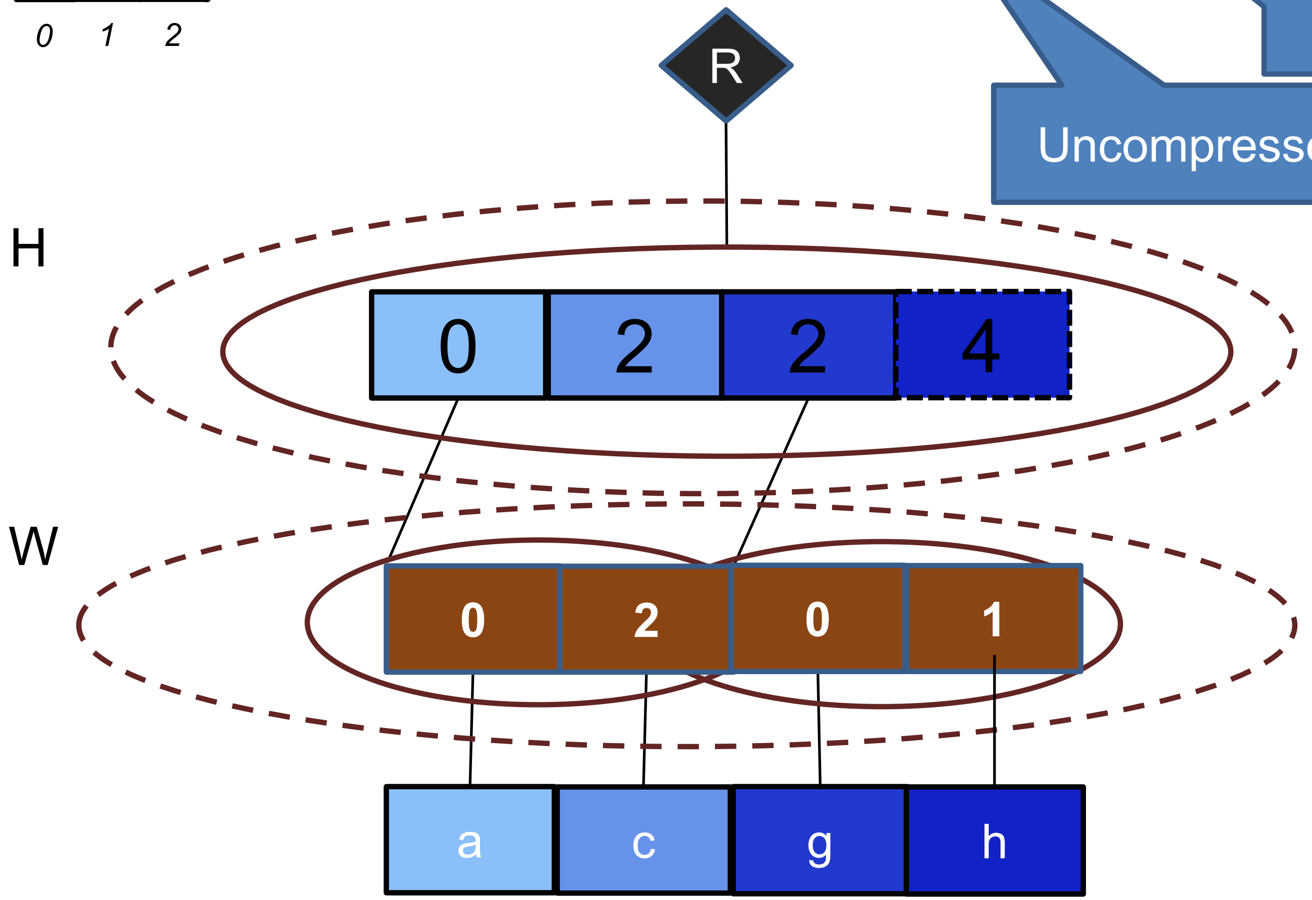


CSR: Tensor<U,C>(H,W)

Coordinate/Payload

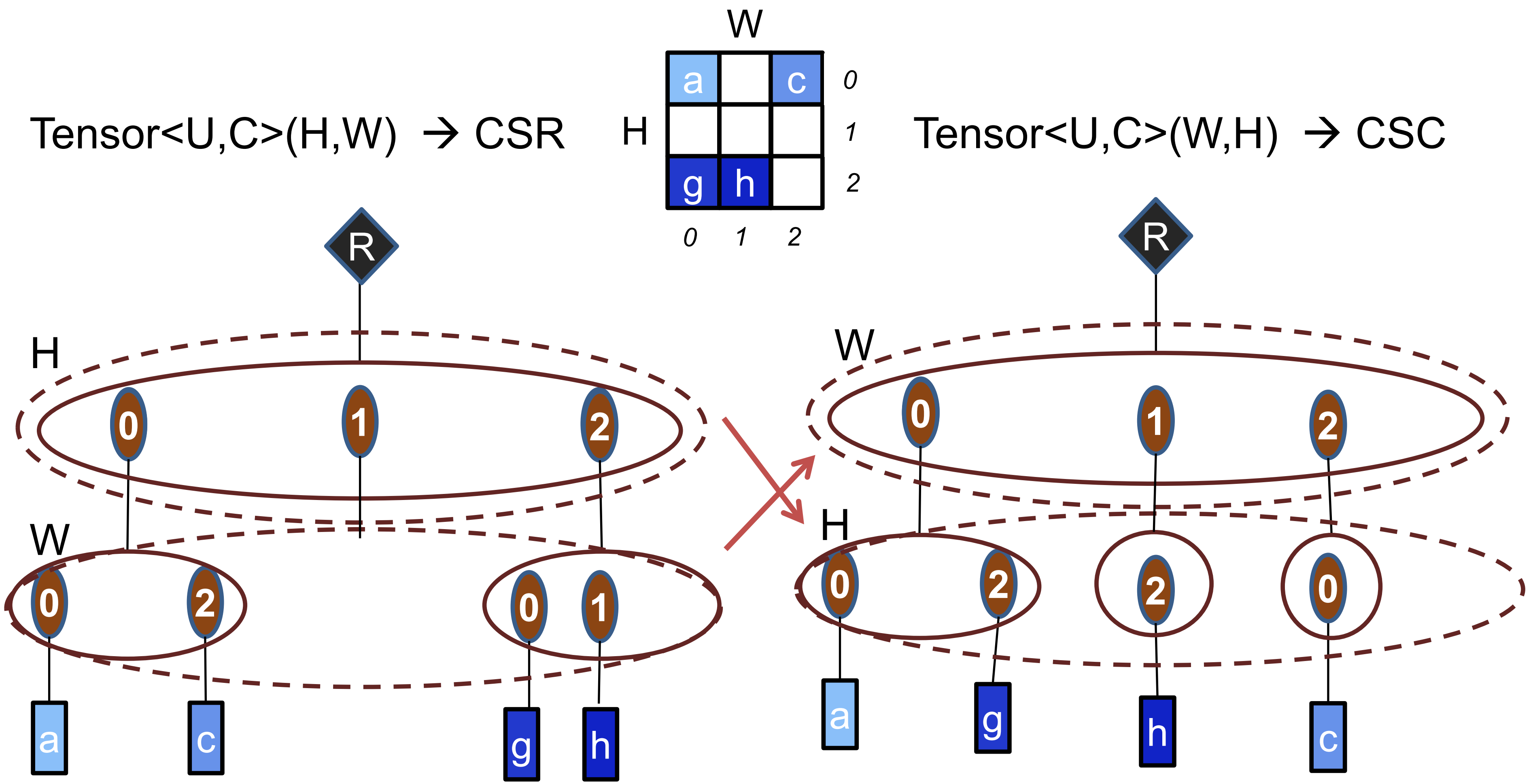
Uncompressed

Rank order



Representation of Order of Ranks

Differentiating CSR and CSC



Can be though of as a rank swap

Traversal Efficiency

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

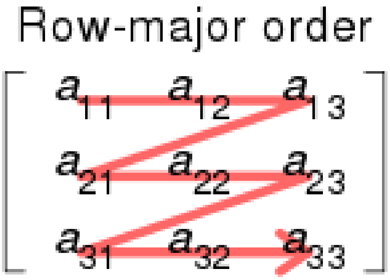
- Operations:
  - maybe(payload) = Fiber.getPayload(coordinate)
  - (coordinate, payload) = Fiber.getNext(rank_traversal_order)

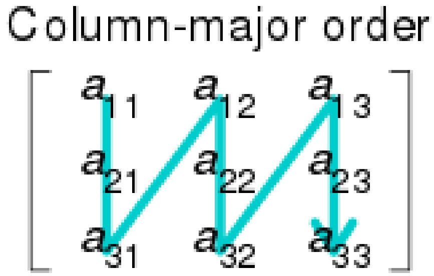
Fiber.getNext() is a useful iterator and its efficiency is highly dependent on representation, both order of ranks and representation of each rank….

Concordant traversal orders

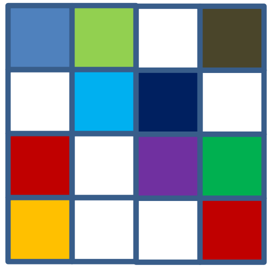
CSR and CSC each has a natural (or “concordant”*) traversal order

Processing Order

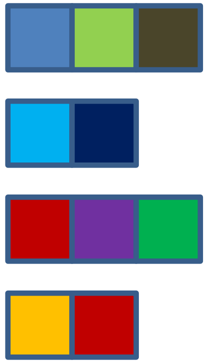




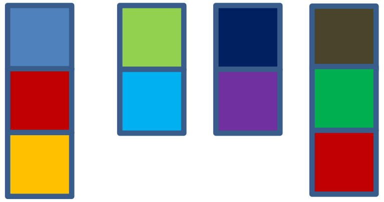
Original Matrix



Compressed Sparse Row (CSR)



Compressed Sparse Column (CSC)



* Term from Michael Pellauer

Example Traversal Efficiency

- Efficiency of getPayload():
  - 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)

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

- Efficiency of getNext() (discordant traversal)
  - Essentially as good (or bad) as getPayload-method….

Traversing a Sparse Tensor

$$Z = T_h$$

```
# 1-D Tensor Traversal

t = Tensor(H)

z = 0
for (h, t_val) in t:
    z += t_val
```

Each iteration returns a (coordinate, payload) tuple

Iteration generated by repeated calls to getNext()

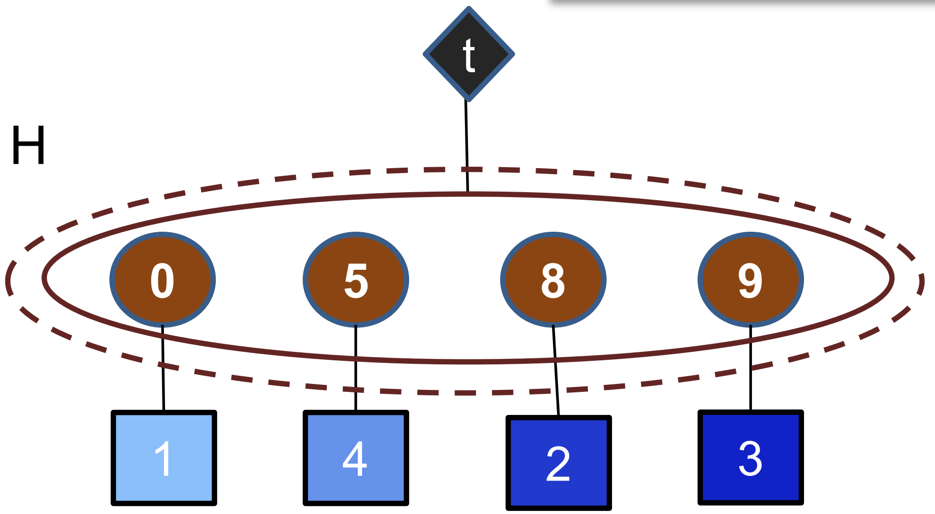
Traversing a Sparse Tensor

$$Z = T_h$$

```
# 1-D Tensor Traversal

t = Tensor(H)

z = 0
for (h, t_val) in t:
    z += t_val
```

Each iteration returns a (coordinate, payload) tuple

Iteration generated by repeated calls to getNext()



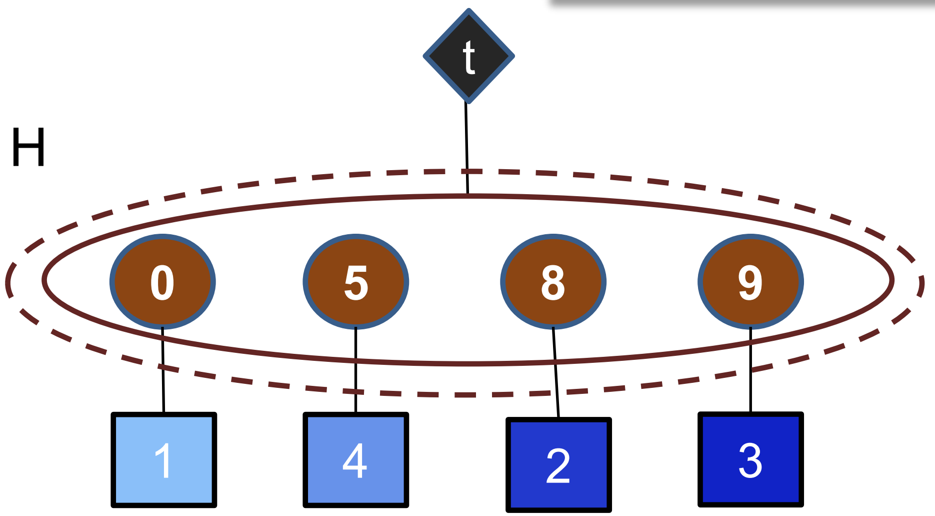
Traversing a Sparse Tensor

$$Z = T_h$$

```
# 1-D Tensor Traversal

t = Tensor(H)

z = 0
for (h, t_val) in t:
    z += t_val
```

Each iteration returns a (coordinate, payload) tuple

Iteration generated by repeated calls to getNext()



t_pos	h	t_val	z
0	0	1	1
1	5	4	5
2	8	2	7
3	9	3	10

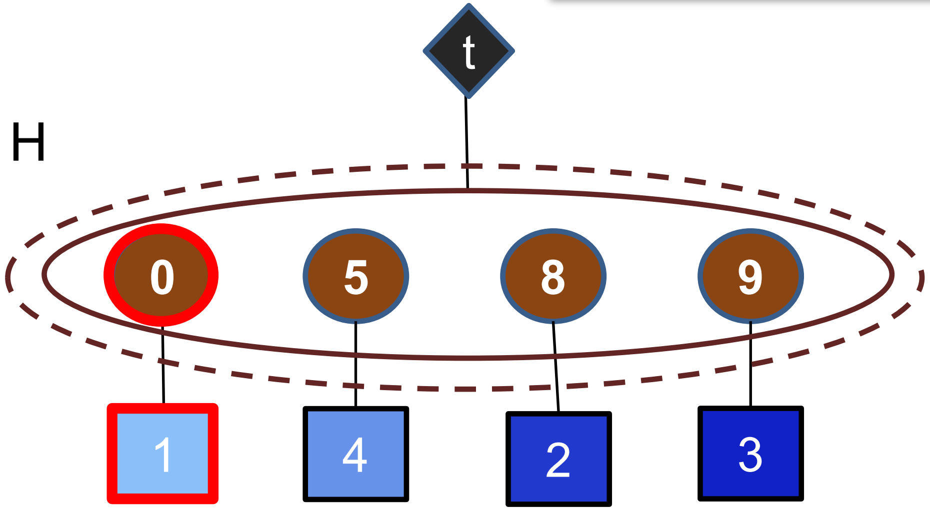
Traversing a Sparse Tensor

$$Z = T_h$$

```
# 1-D Tensor Traversal

t = Tensor(H)

z = 0
for (h, t_val) in t:
    z += t_val
```

Each iteration returns a (coordinate, payload) tuple

Iteration generated by repeated calls to getNext()



t_pos	h	t_val	z
0	0	1	1
1	5	4	5
2	8	2	7
3	9	3	10

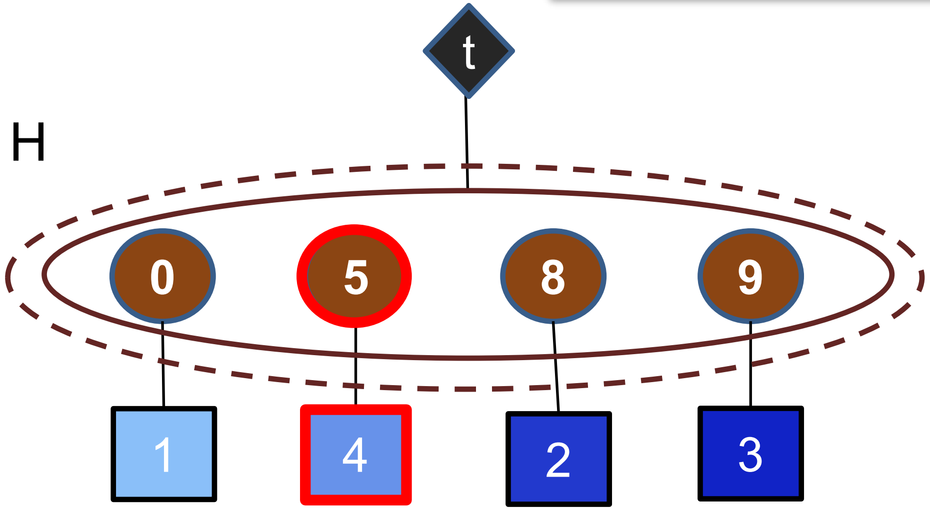
Traversing a Sparse Tensor

$$Z = T_h$$

```
# 1-D Tensor Traversal

t = Tensor(H)

z = 0
for (h, t_val) in t:
    z += t_val
```

Each iteration returns a (coordinate, payload) tuple

Iteration generated by repeated calls to getNext()



t_pos	h	t_val	z
0	0	1	1
1	5	4	5
2	8	2	7
3	9	3	10

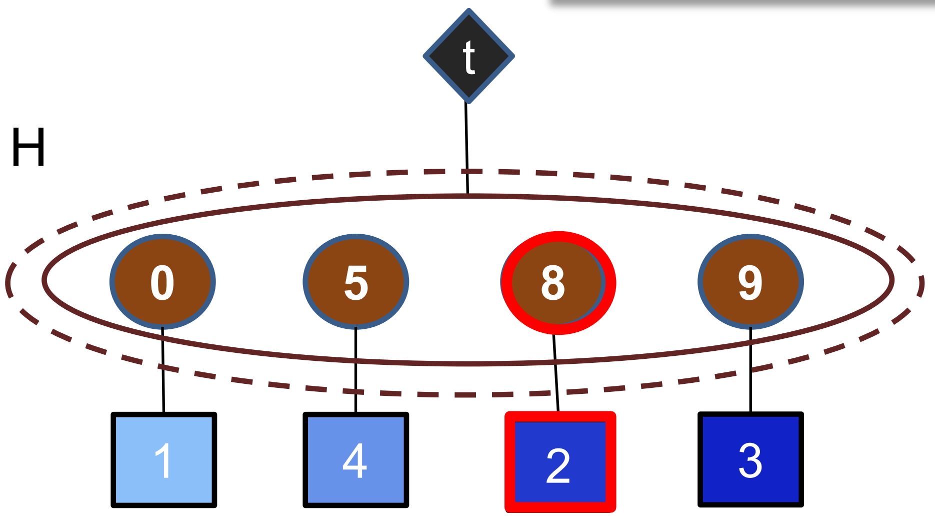
Traversing a Sparse Tensor

$$Z = T_h$$

```
# 1-D Tensor Traversal

t = Tensor(H)

z = 0
for (h, t_val) in t:
    z += t_val
```

Each iteration returns a (coordinate, payload) tuple

Iteration generated by repeated calls to getNext()



t_pos	h	t_val	z
0	0	1	1
1	5	4	5
2	8	2	7
3	9	3	10

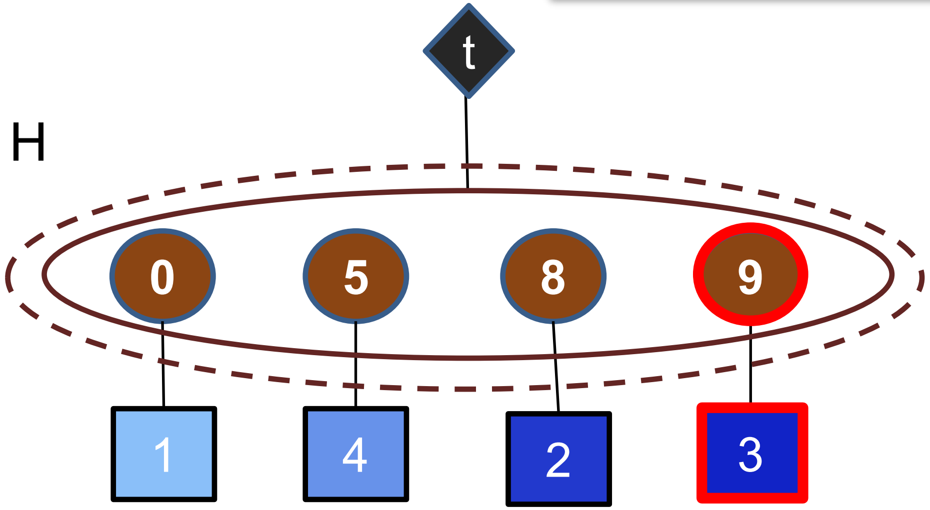
Traversing a Sparse Tensor

$$Z = T_h$$

```
# 1-D Tensor Traversal

t = Tensor(H)

z = 0
for (h, t_val) in t:
    z += t_val
```

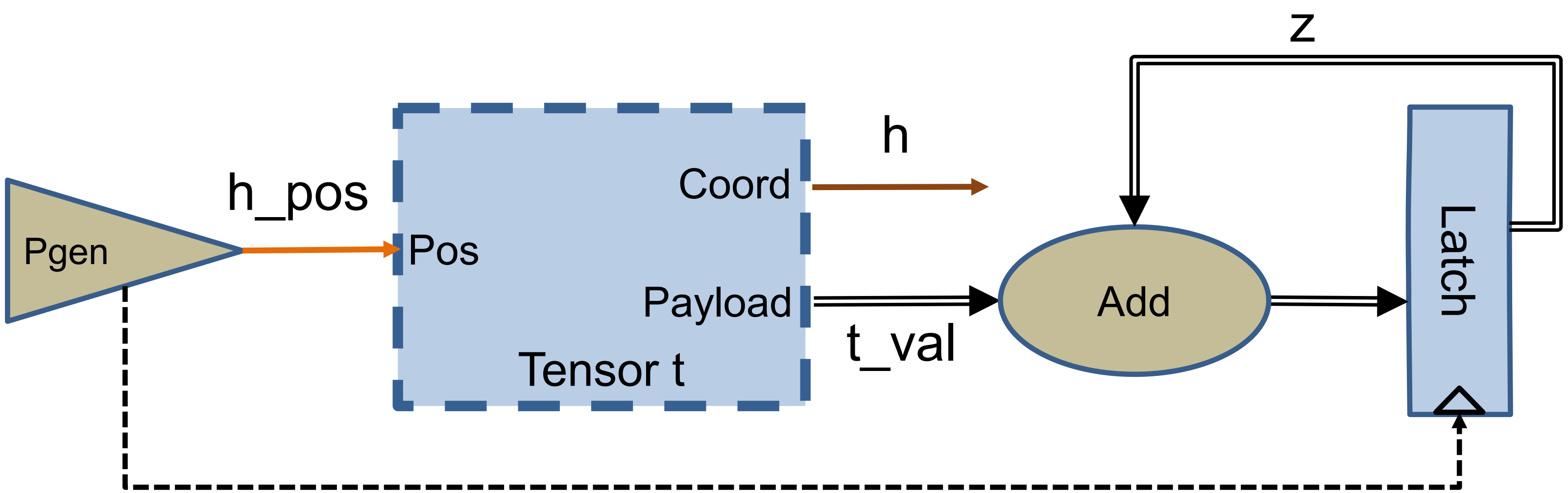
Each iteration returns a (coordinate, payload) tuple

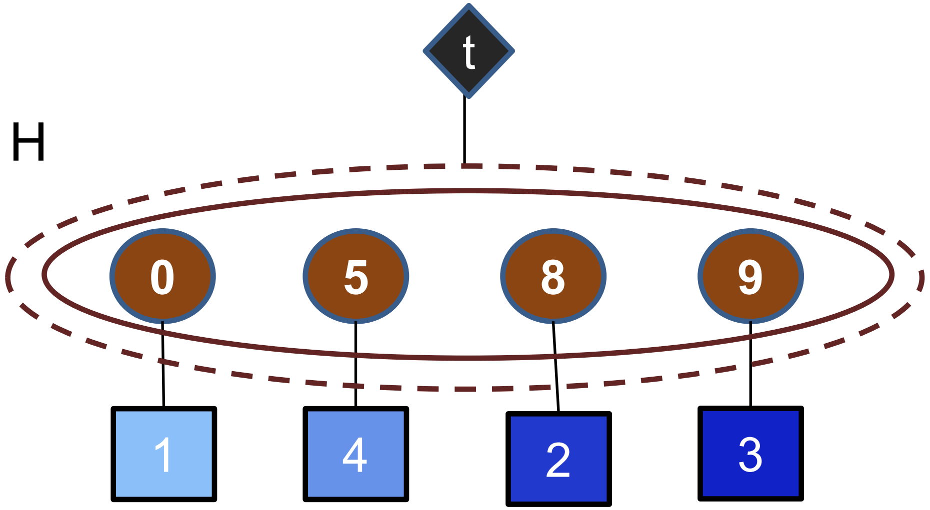
Iteration generated by repeated calls to getNext()



t_pos	h	t_val	z
0	0	1	1
1	5	4	5
2	8	2	7
3	9	3	10

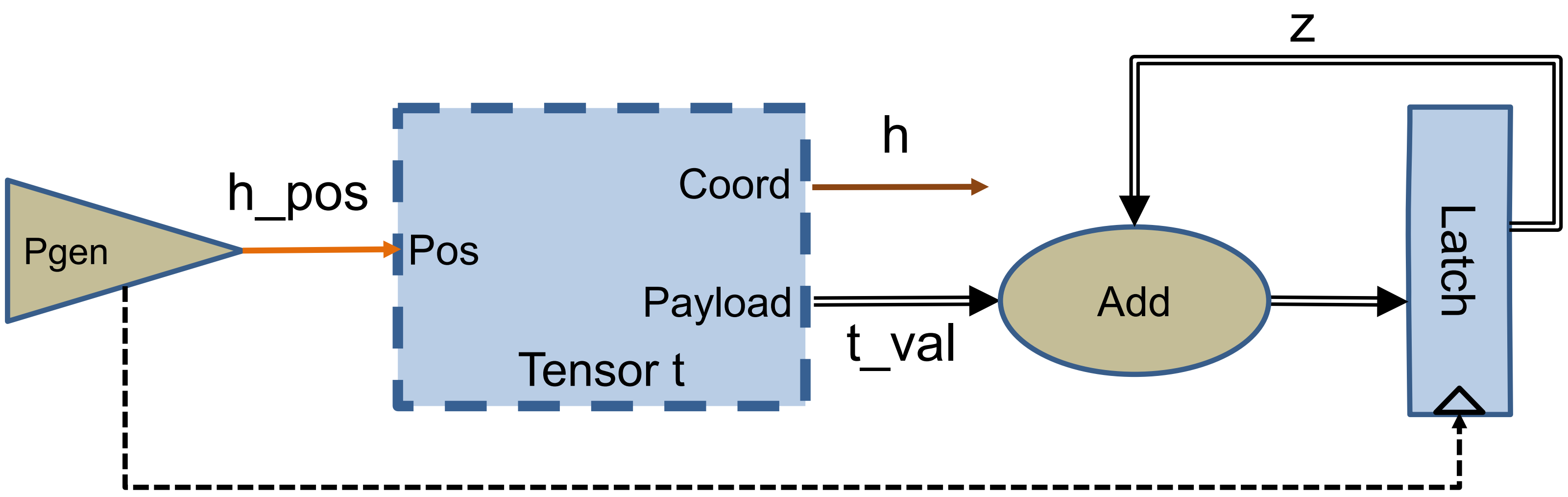
Traversing a Sparse Tensor

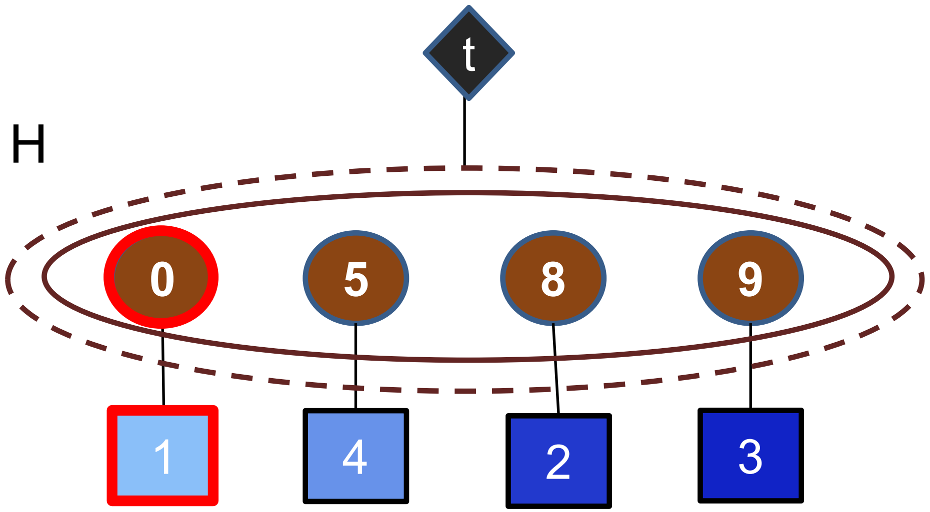




t_pos	h	t_val	z
0	0	1	1
1	5	4	5
2	8	2	7
3	9	3	10

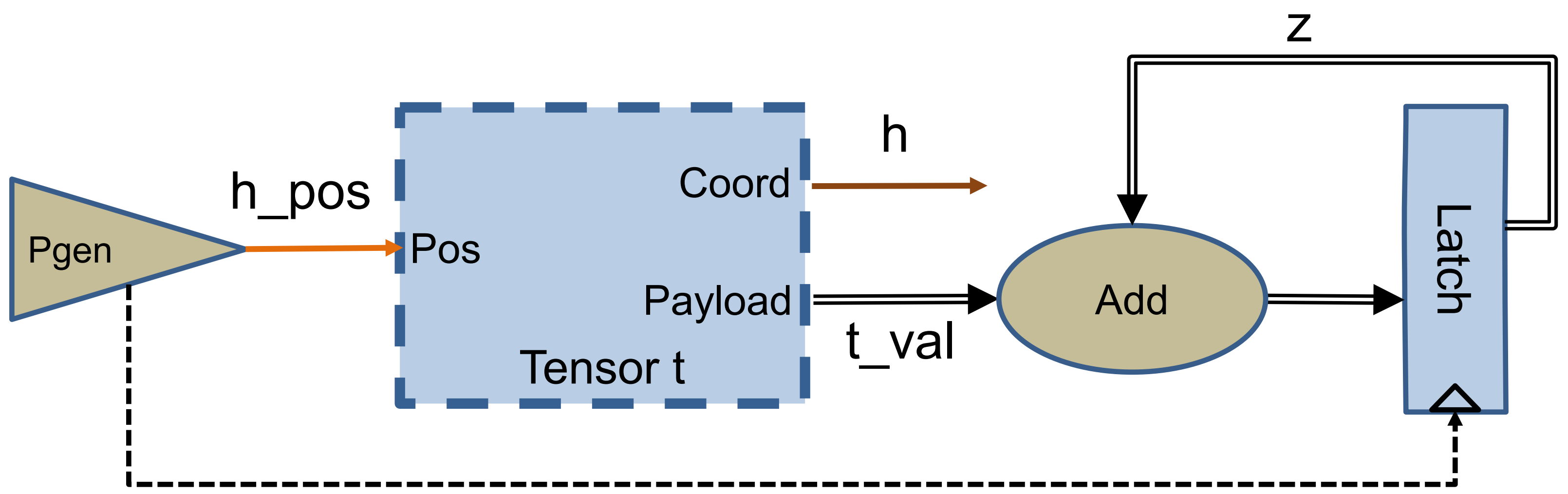
Traversing a Sparse Tensor

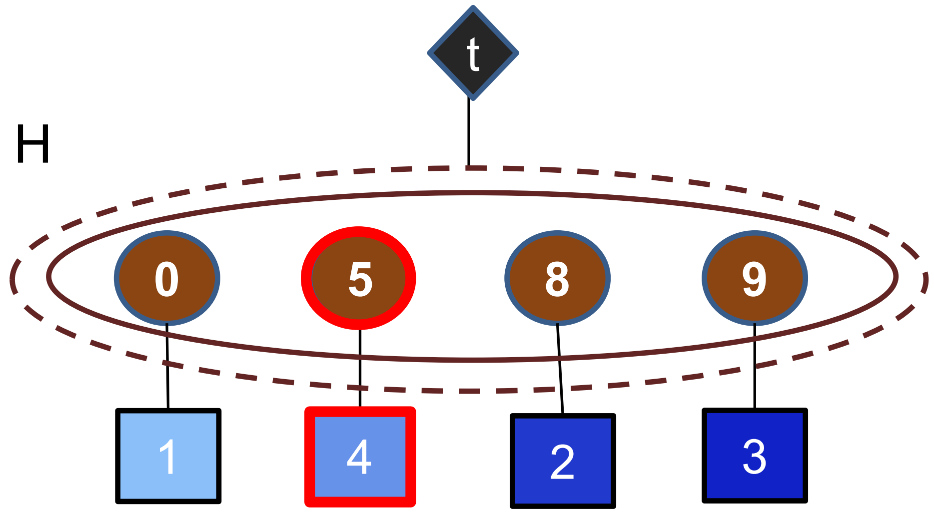




t_pos	h	t_val	z
0	0	1	1
1	5	4	5
2	8	2	7
3	9	3	10

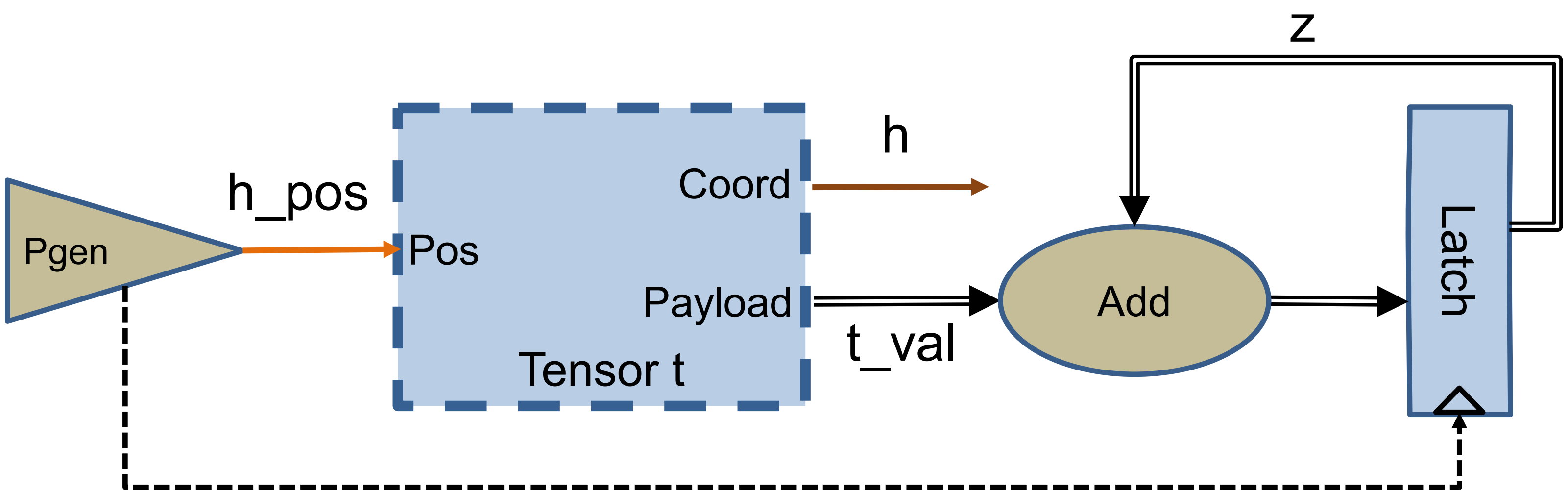
Traversing a Sparse Tensor

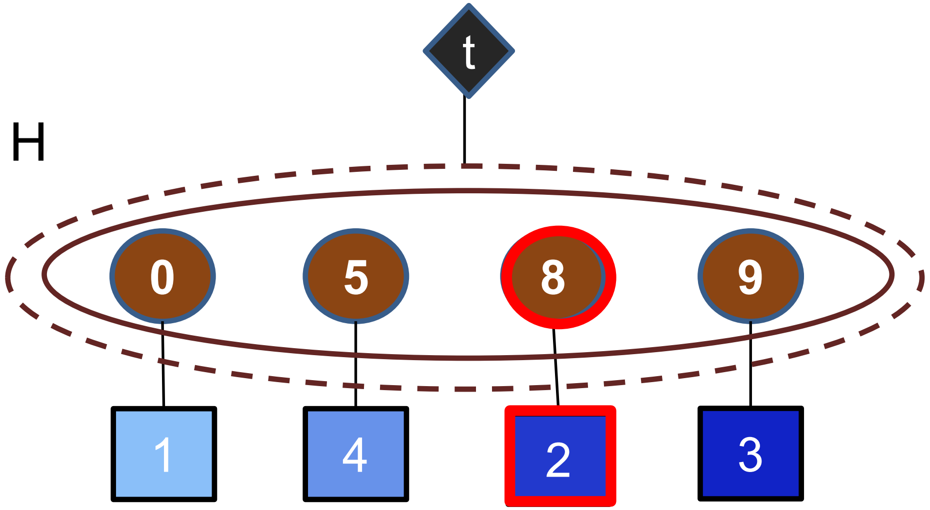




t_pos	h	t_val	z
0	0	1	1
1	5	4	5
2	8	2	7
3	9	3	10

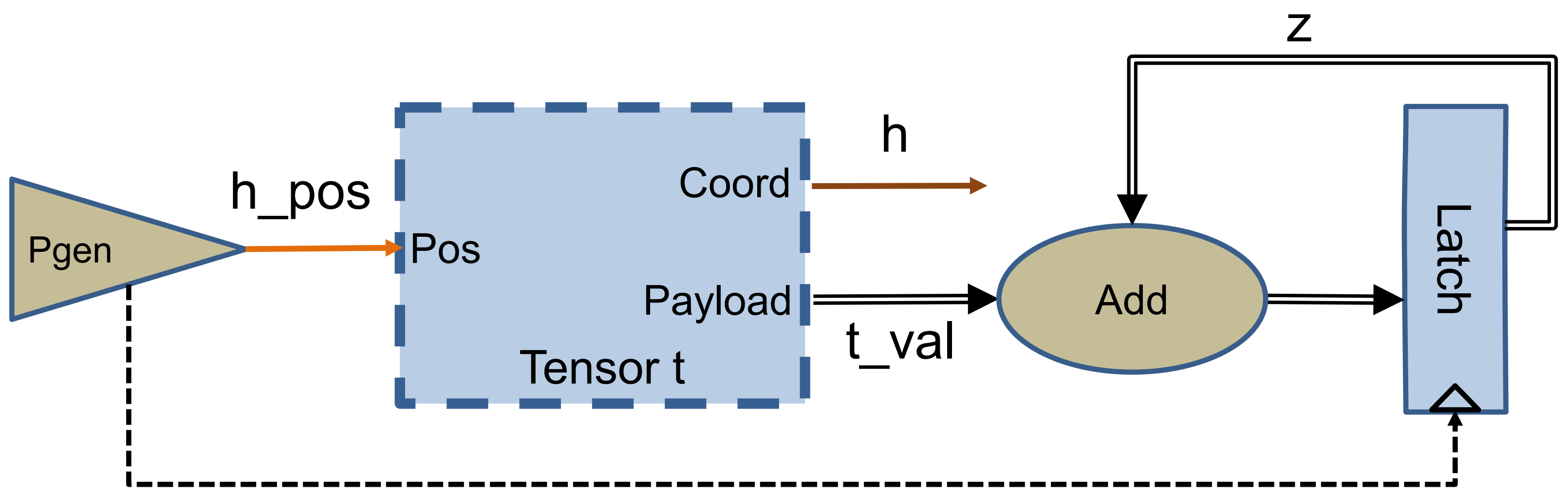
Traversing a Sparse Tensor

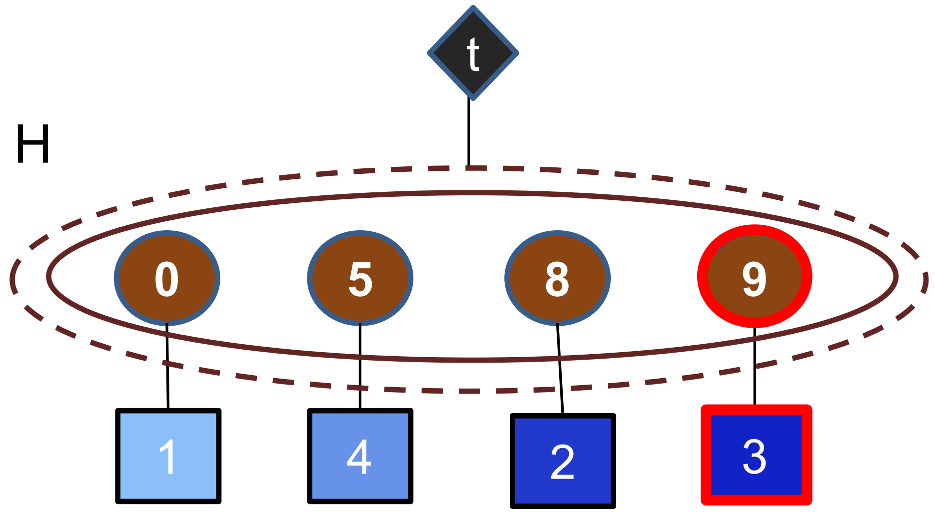




t_pos	h	t_val	z
0	0	1	1
1	5	4	5
2	8	2	7
3	9	3	10

Traversing a Sparse Tensor





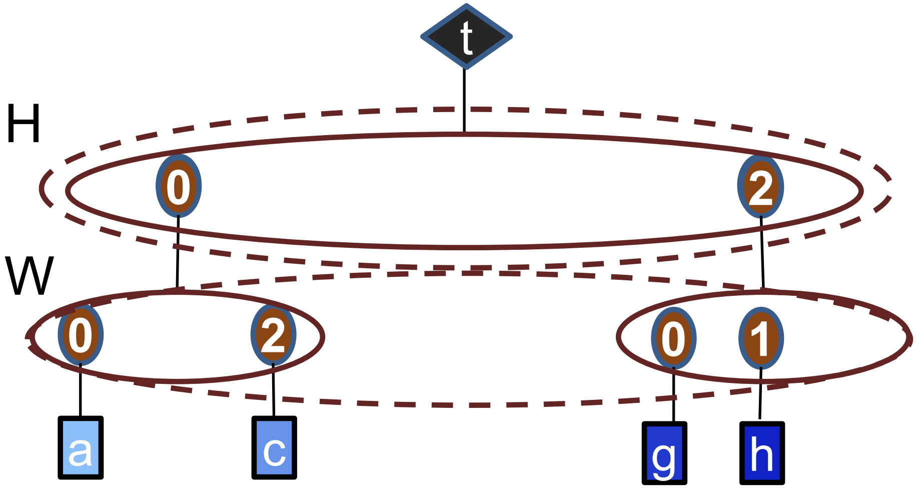
t_pos	h	t_val	z
0	0	1	1
1	5	4	5
2	8	2	7
3	9	3	10

Tensor Traversal (2-D)

$$Z = T_{h,w}$$

```
# 2-D Tensor Traversal

t = Tensor(H,W)

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

Each iteration returns a (coordinate, payload) tuple

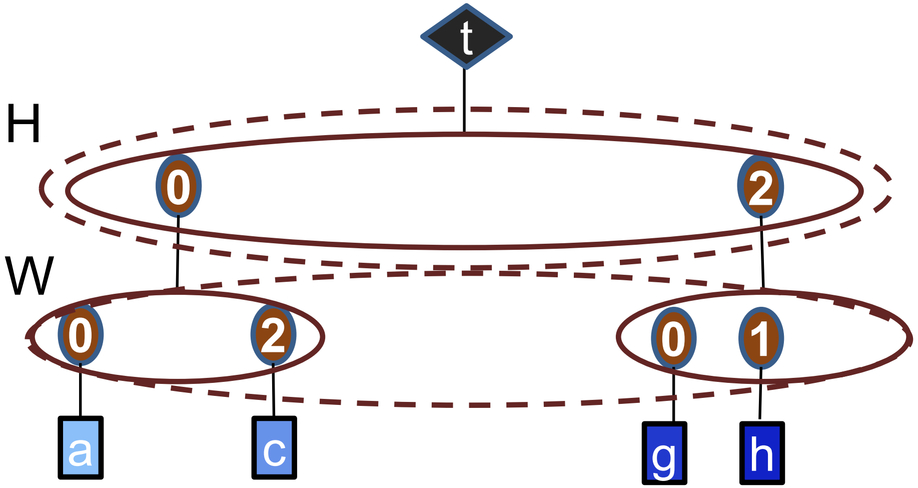


Tensor Traversal (2-D)

$$Z = T_{h,w}$$

```
# 2-D Tensor Traversal

t = Tensor(H,W)

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

Each iteration returns a (coordinate, payload) tuple

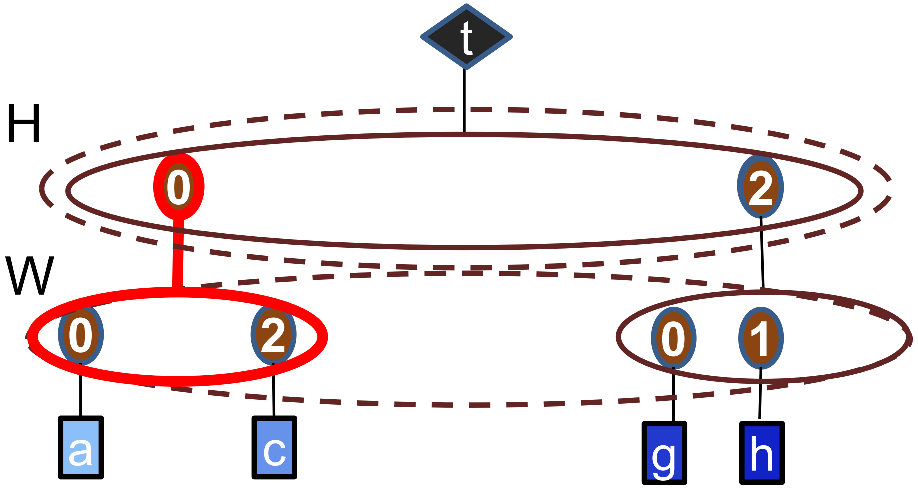


t_pos	h	t_h_pos	w	t_val
0	0	?	?	?
0	0	0	0	a
0	0	1	2	c
1	2	?	?	?
…	…	…	…	…

Tensor Traversal (2-D)

$$Z = T_{h,w}$$

```
# 2-D Tensor Traversal

t = Tensor(H,W)

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

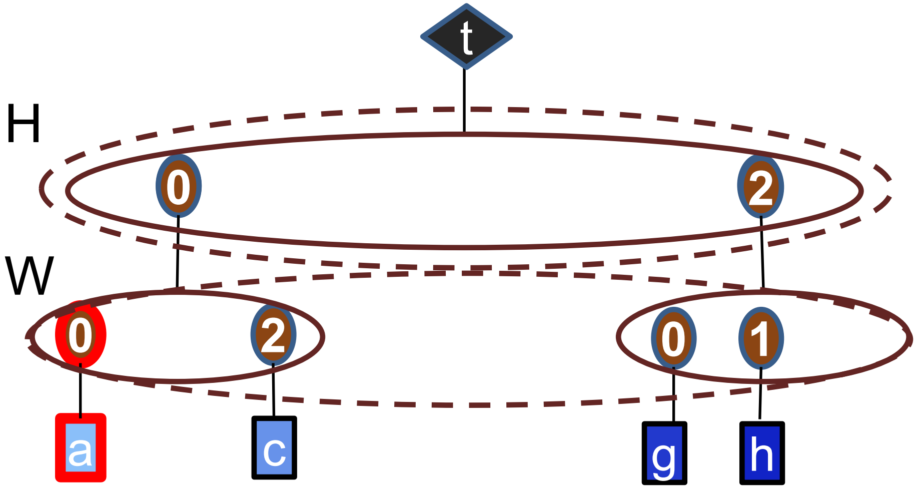


t_pos	h	t_h_pos	w	t_val
0	0	?	?	?
0	0	0	0	a
0	0	1	2	c
1	2	?	?	?
…	…	…	…	…

Tensor Traversal (2-D)

$$Z = T_{h,w}$$

```
# 2-D Tensor Traversal

t = Tensor(H,W)

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

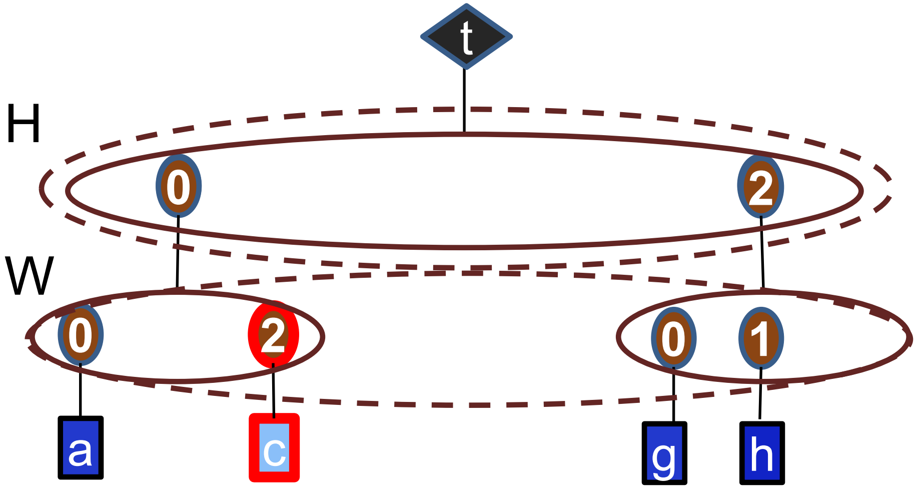


t_pos	h	t_h_pos	w	t_val
0	0	?	?	?
0	0	0	0	a
0	0	1	2	c
1	2	?	?	?
…	…	…	…	…

Tensor Traversal (2-D)

$$Z = T_{h,w}$$

```
# 2-D Tensor Traversal

t = Tensor(H,W)

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

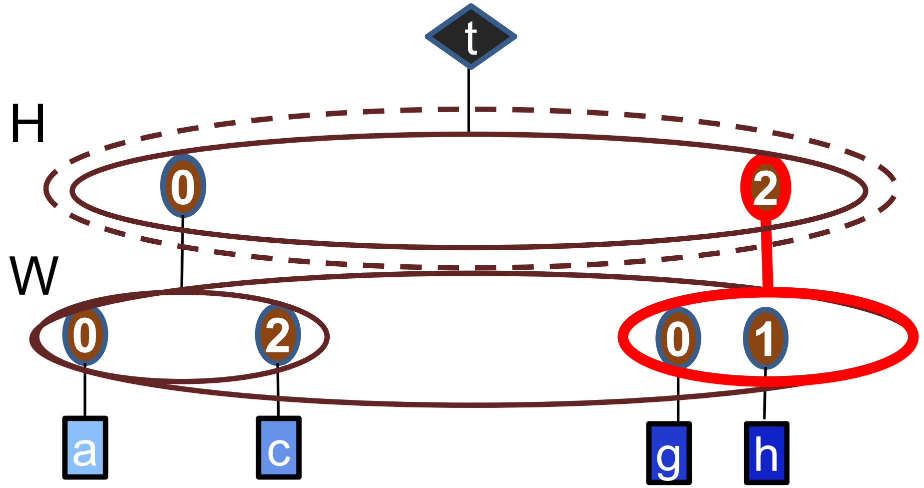


t_pos	h	t_h_pos	w	t_val
0	0	?	?	?
0	0	0	0	a
0	0	1	2	c
1	2	?	?	?
…	…	…	…	…

Tensor Traversal (2-D)

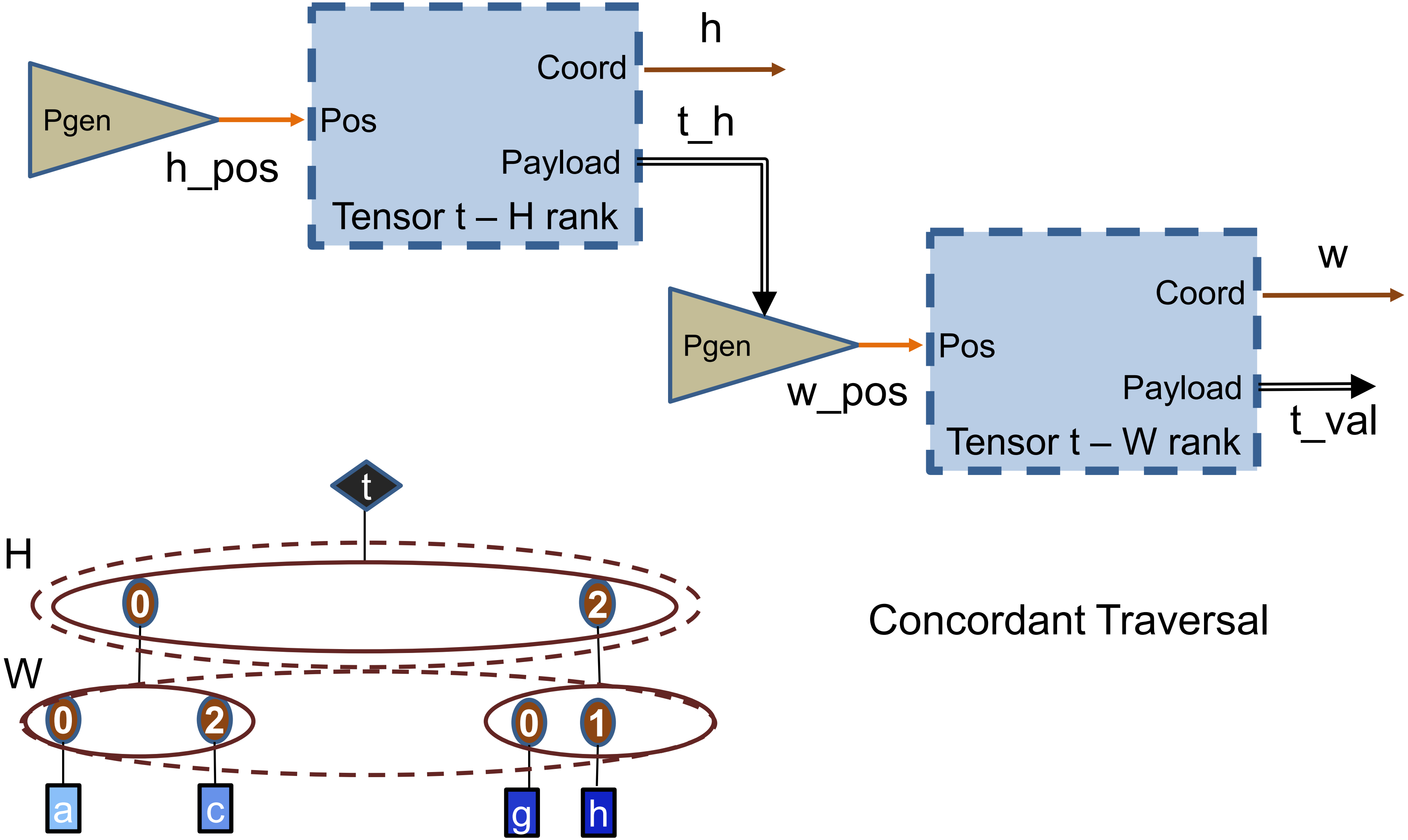
$$Z = T_{h,w}$$

```
# 2-D Tensor Traversal

t = Tensor(H,W)

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

Abstraction versus Implementation

- Abstraction
  - An interface and semantics
    - Attributes: No implementation, data layout or timing
    - Use: implementation-agnostic understanding
    - Examples:
      - Fibers
      - Fibertree
- Implementation
  - Specific implementation of an abstract spec
    - Attributes: Concrete implementation, data layout and timing
    - Examples:
      - Fibers → uncompressed array, coordinate/payload list
      - Fiber-tree → CSR, CSC, CSF, COO…

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
```

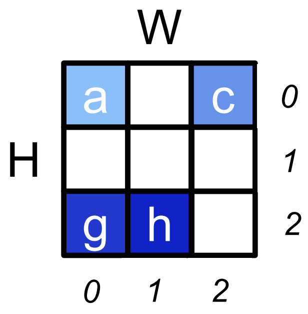
For uncompressed rank coordinate equals position

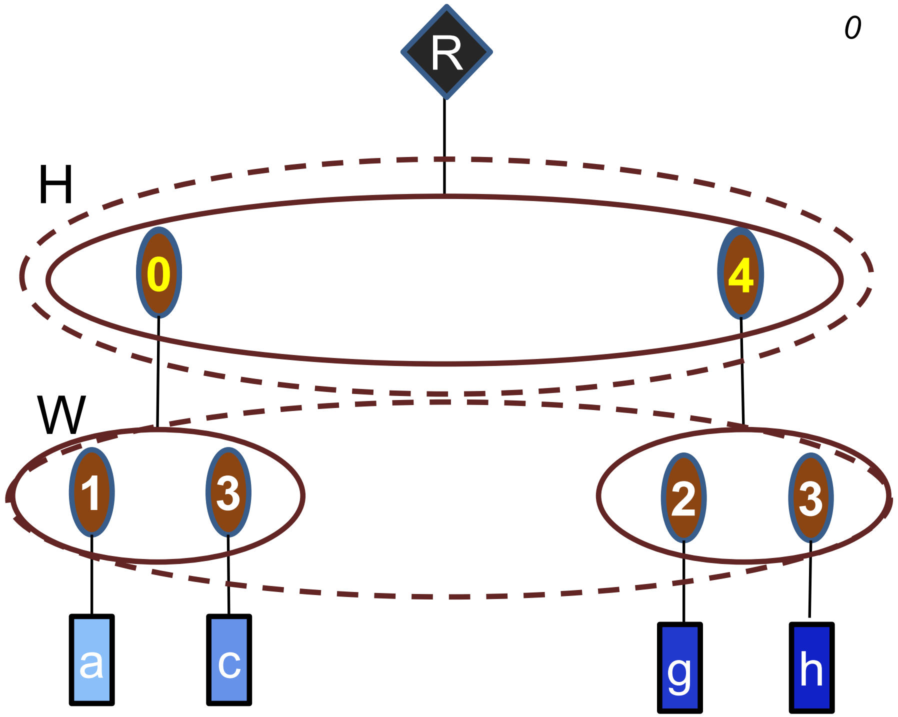
Coordinates not actually used in this example

Merging Ranks

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

Tensor-<C,C>(H,W)

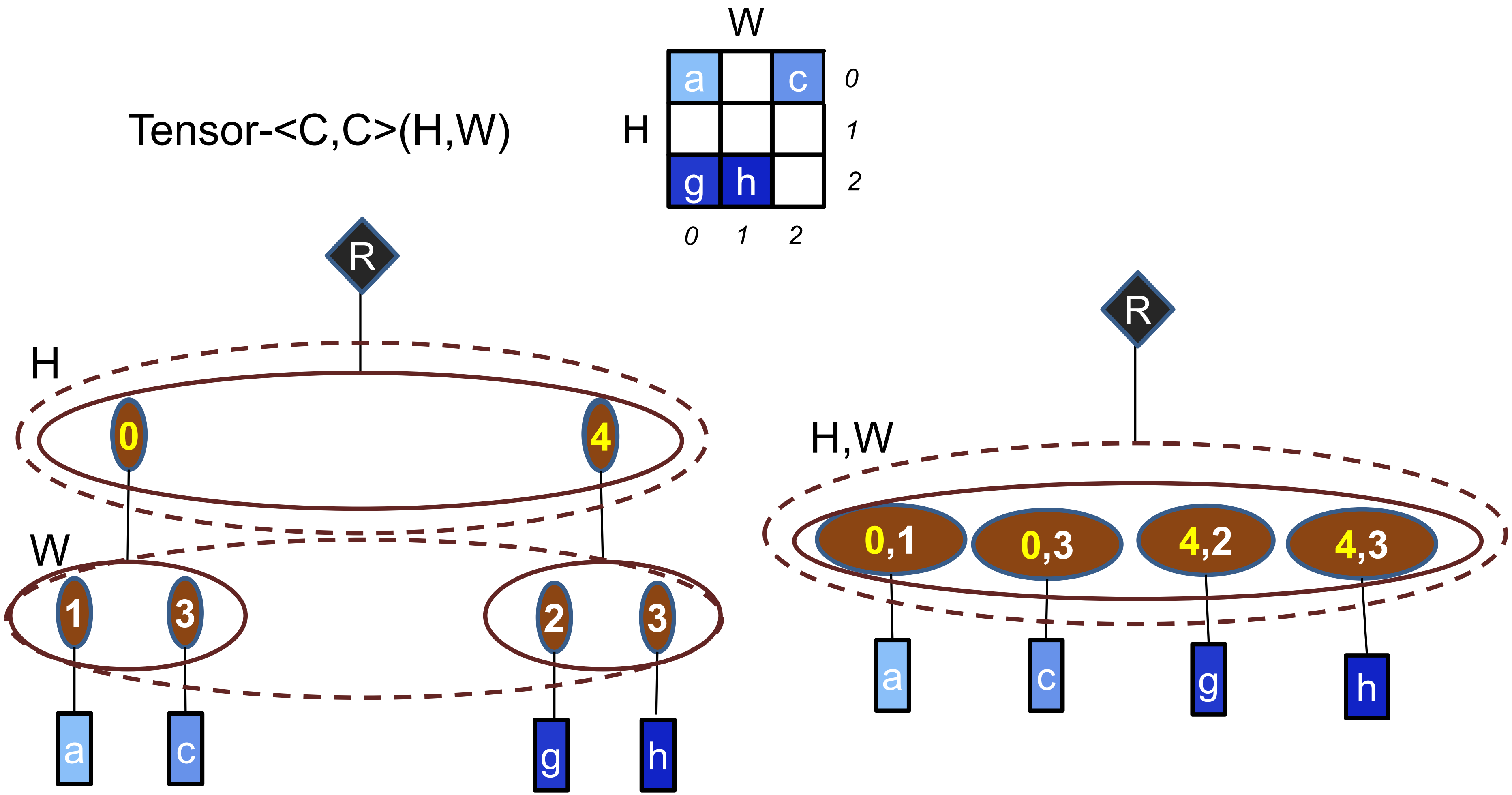




Merging Ranks

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

Tensor-<C,C>(H,W)

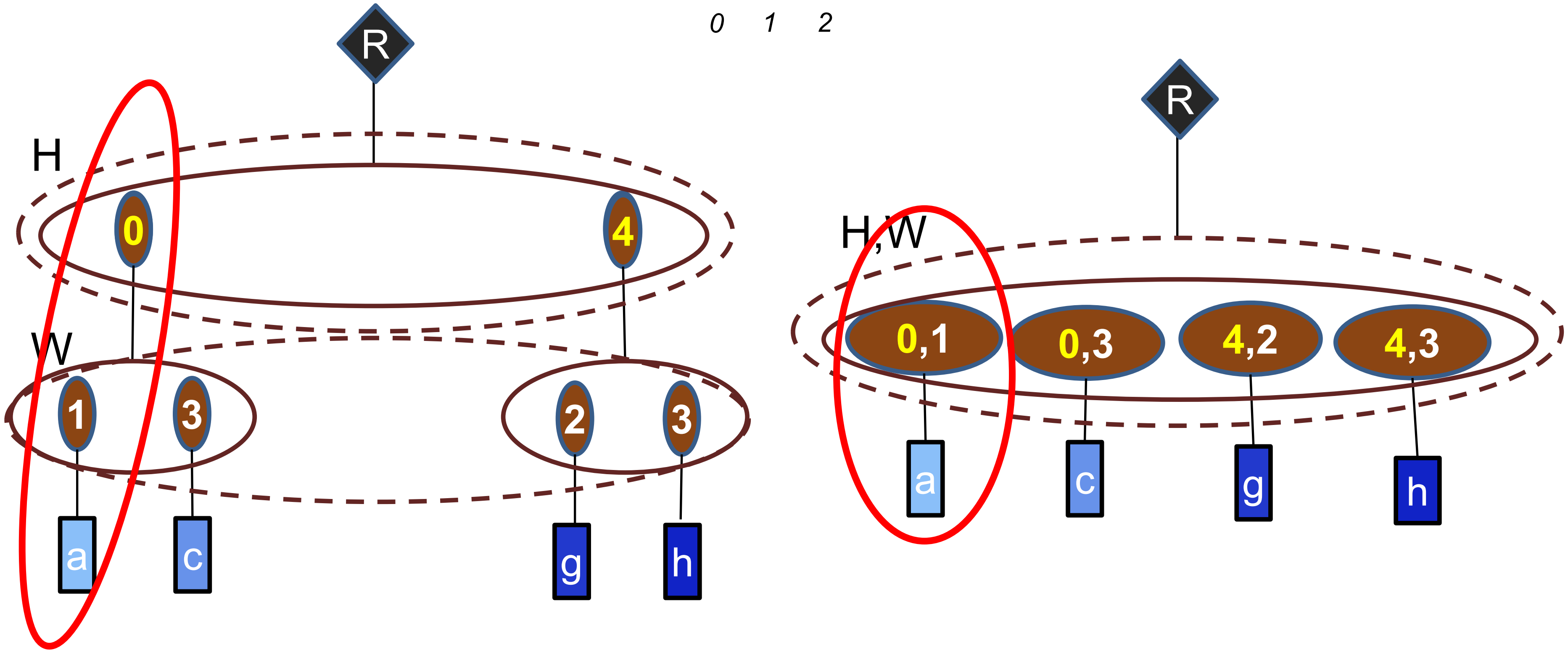


Merging Ranks

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

Tensor-<C,C>(H,W)

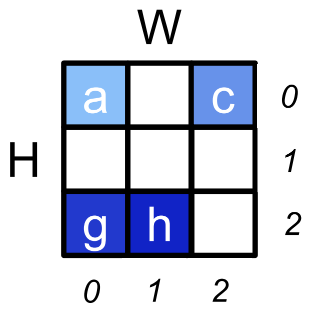
H \ W	0	1	2
0	a		c
1			
2	g	h	

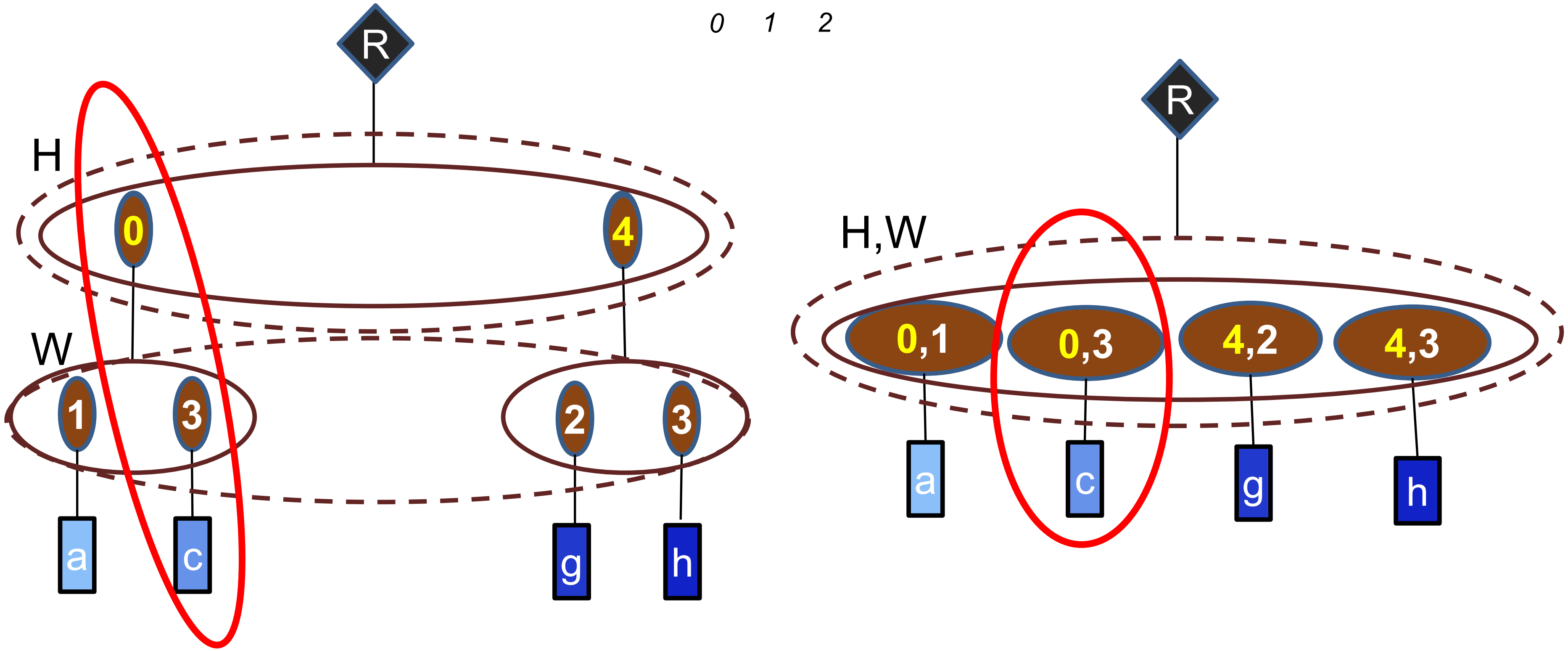


Merging Ranks

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

Tensor-<C,C>(H,W)

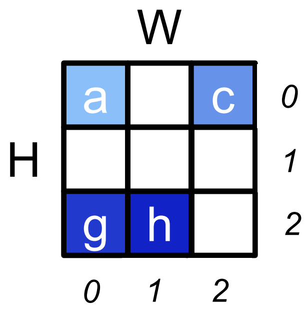


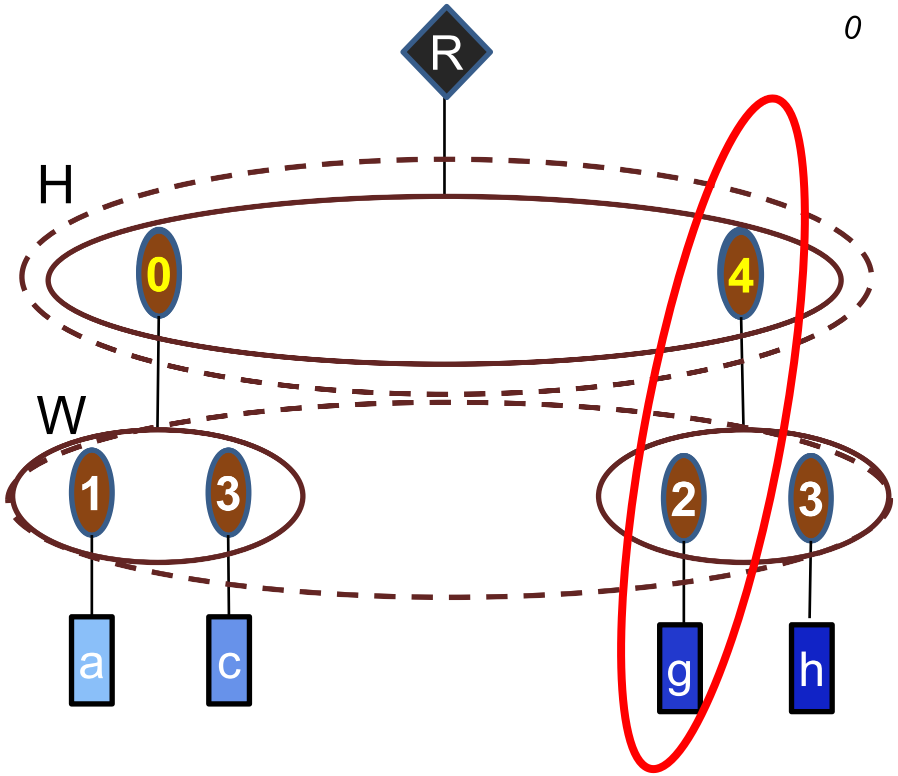


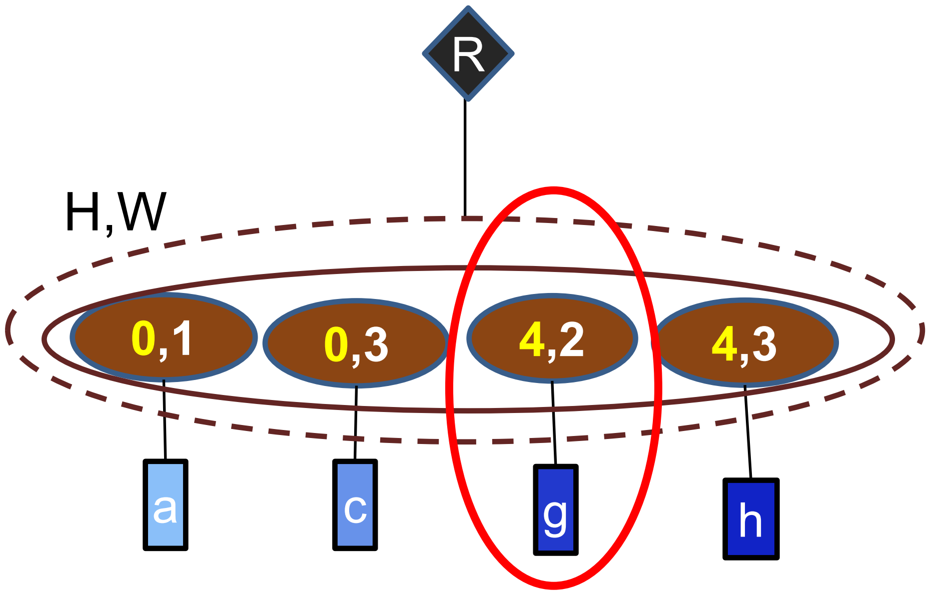
Merging Ranks

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

Tensor-<C,C>(H,W)





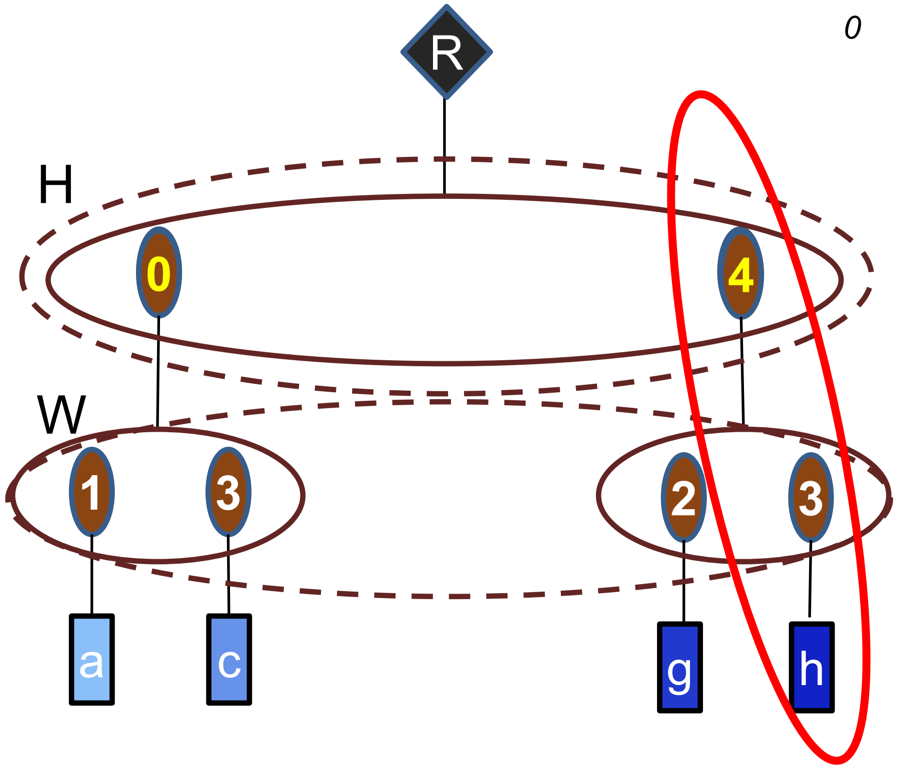


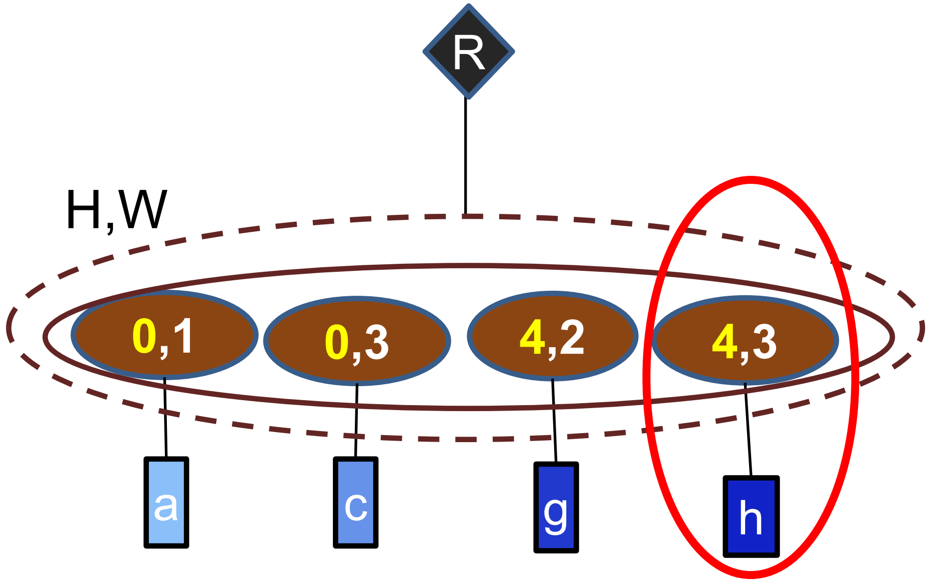
Merging Ranks

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

Tensor-<C,C>(H,W)

H \ W	0	1	2
0	a		c
1			
2	g	h	

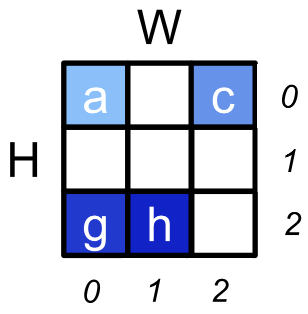




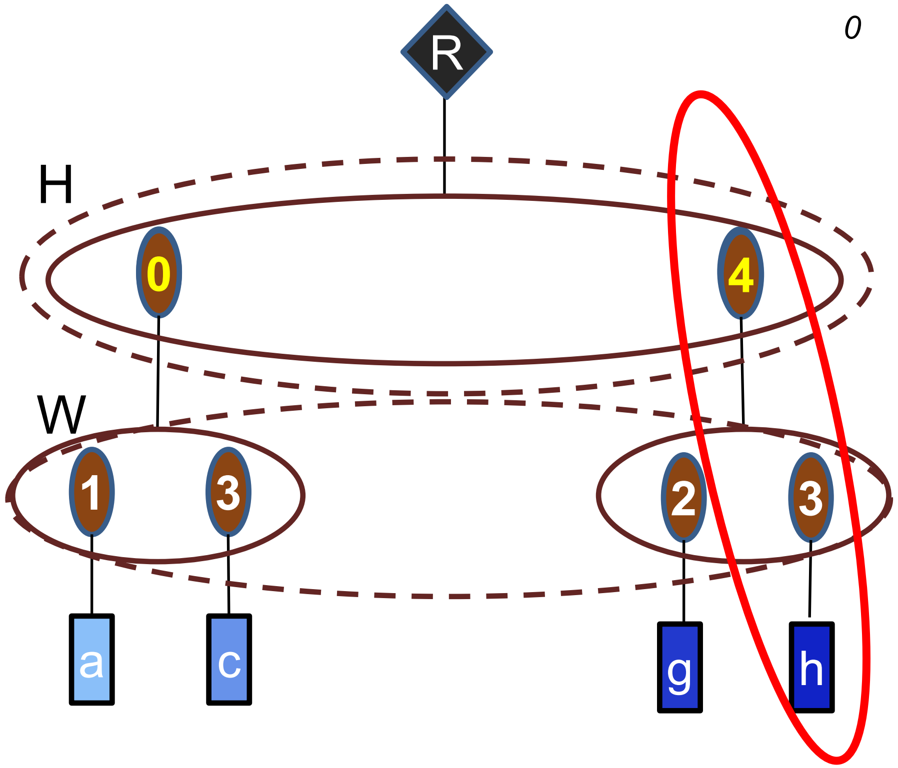
Merging Ranks

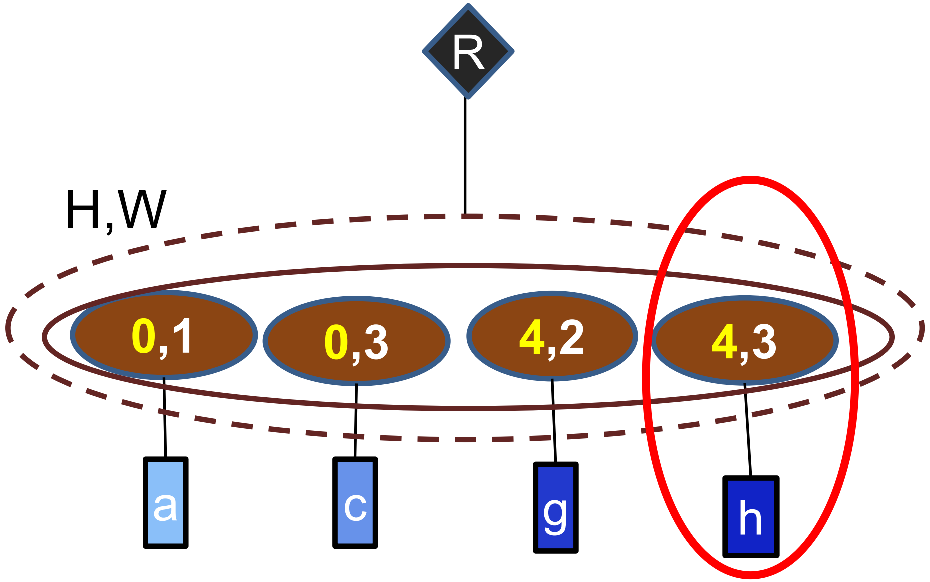
For efficiency one can form new representations where the data structure for two or more ranks are combined.

Tensor-<C,C>(H,W)



Tensor-<C^2>(H,W)



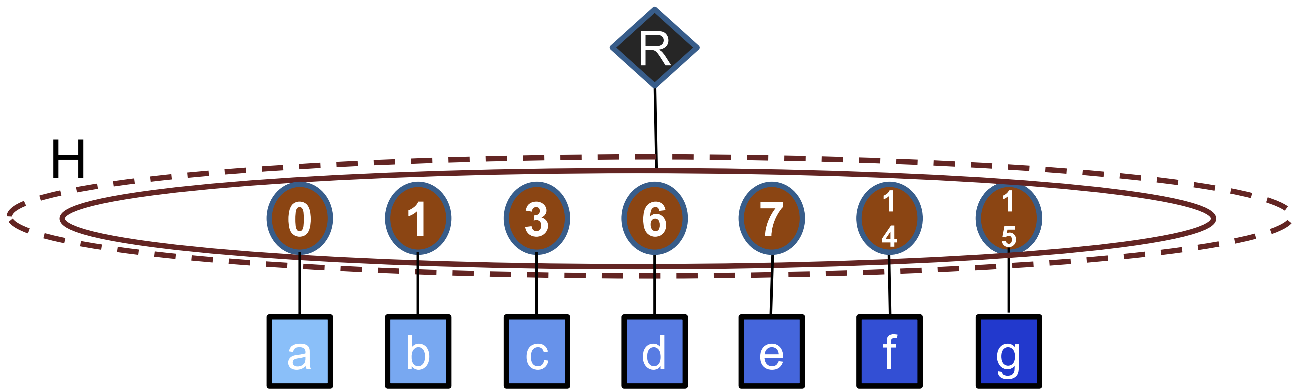


Merging Ranks

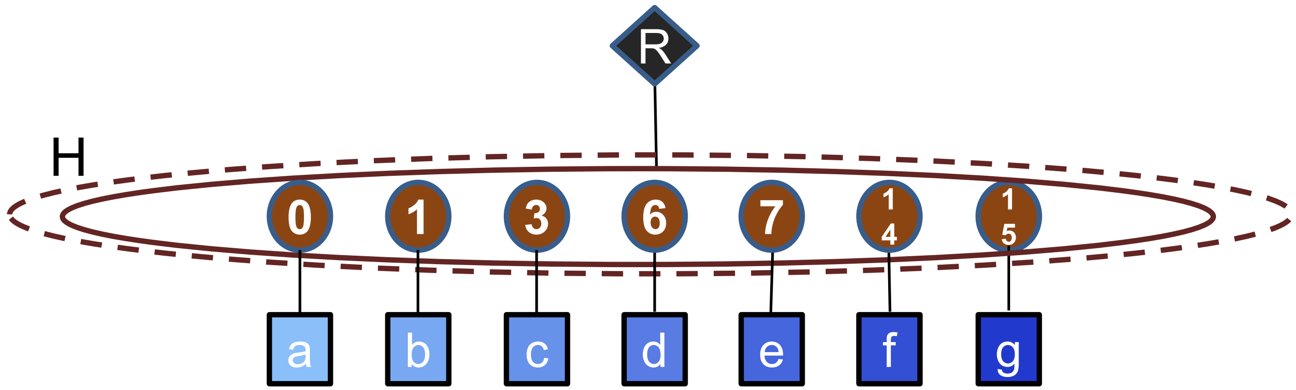
- For efficiency one can form new representations where the data structure for two or more ranks are combined:

- 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

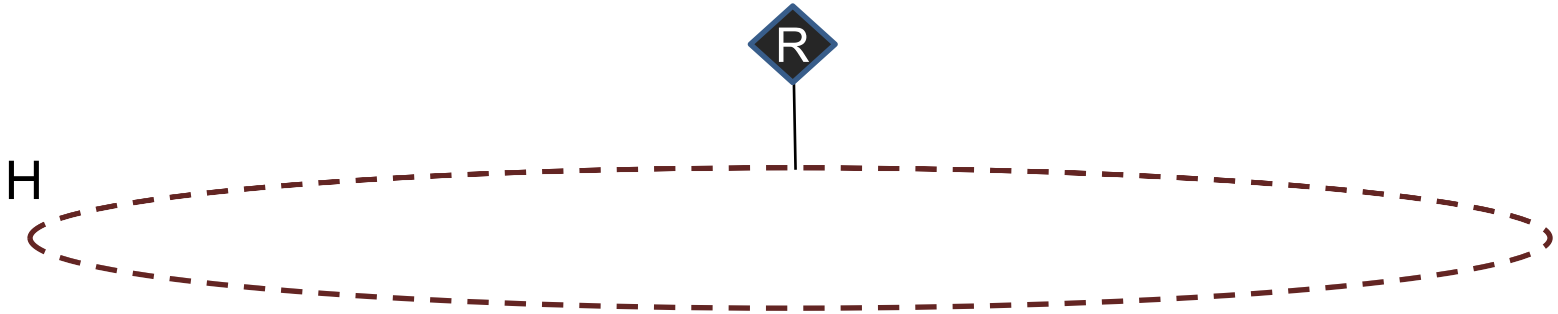
Splitting Fibers – Coordinate Space



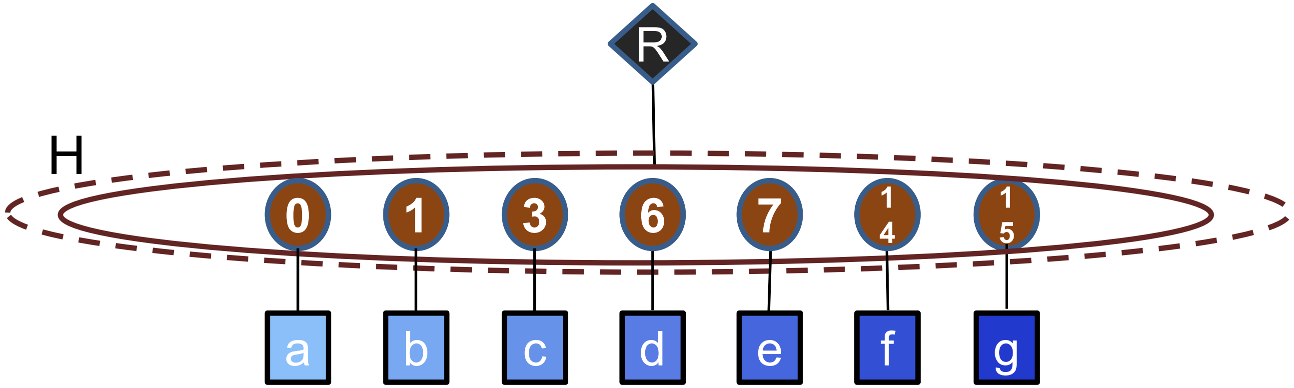
Splitting Fibers – Coordinate Space



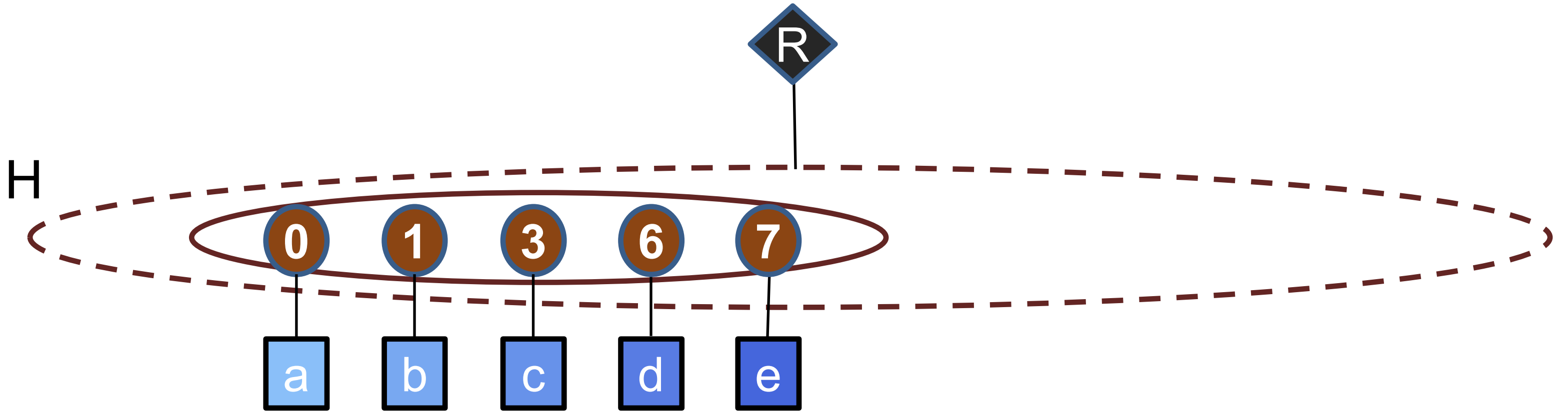
Split uniformly by coordinates (groups of 8 coordinates)



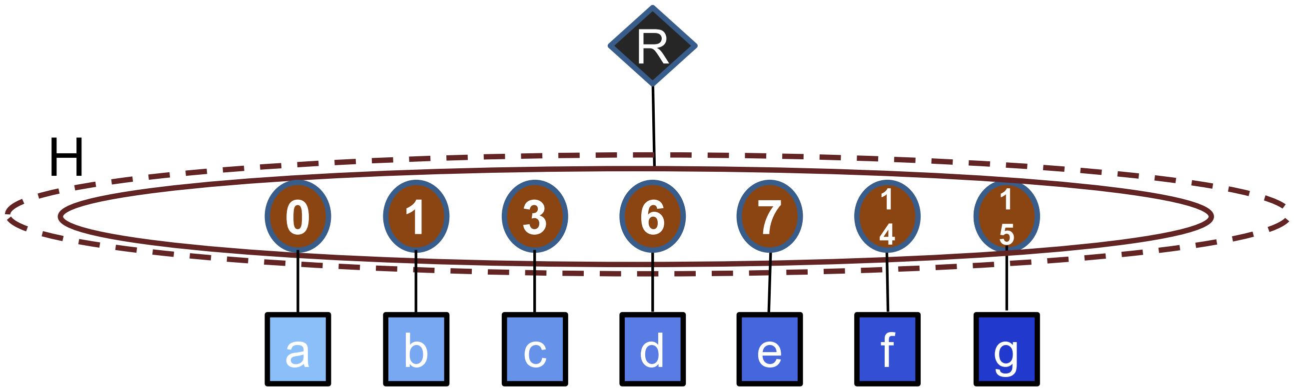
Splitting Fibers – Coordinate Space



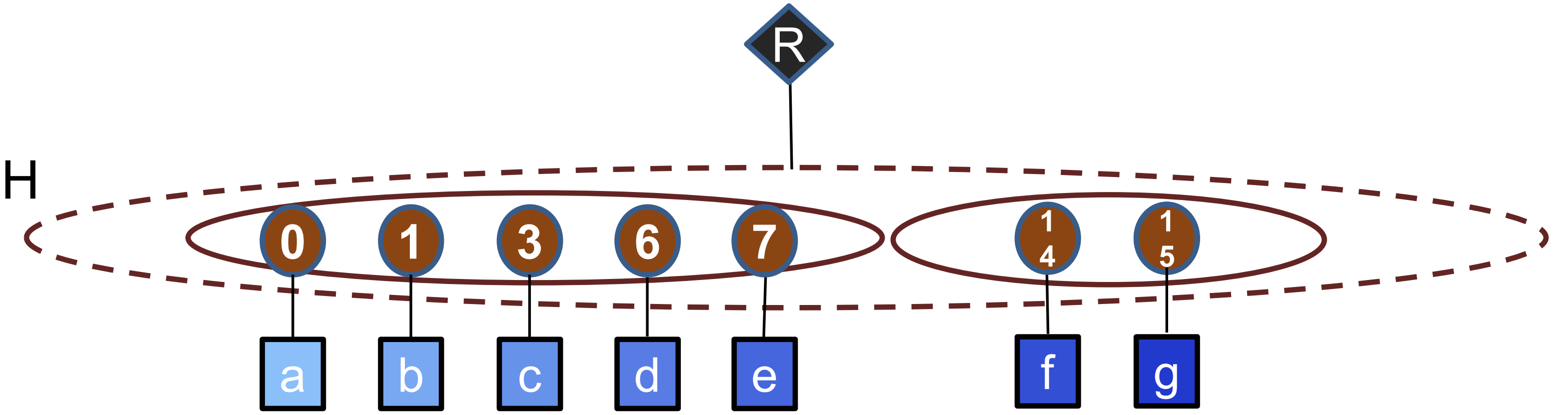
Split uniformly by coordinates (groups of 8 coordinates)



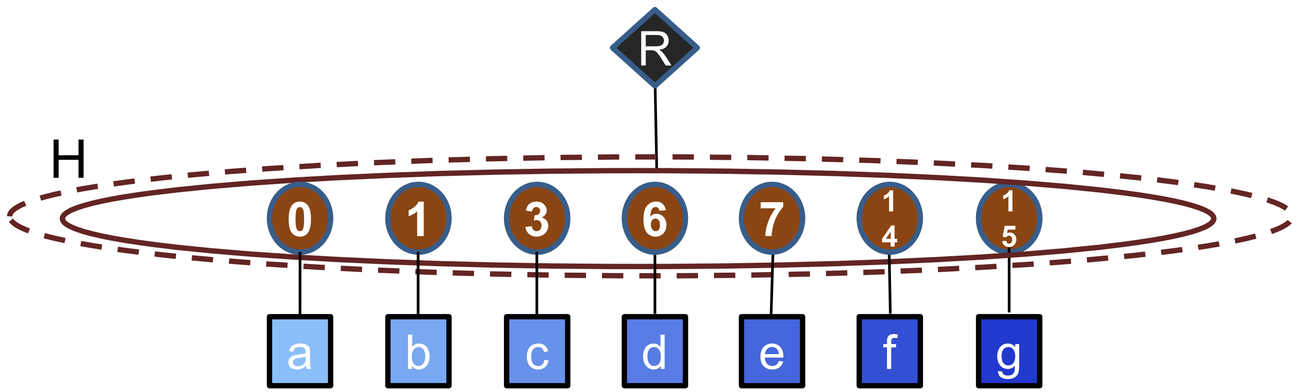
Splitting Fibers – Coordinate Space



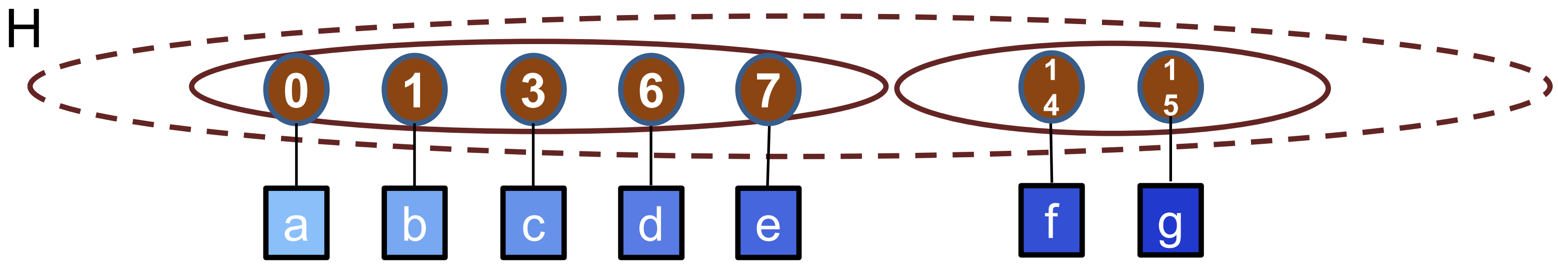
Split uniformly by coordinates (groups of 8 coordinates)



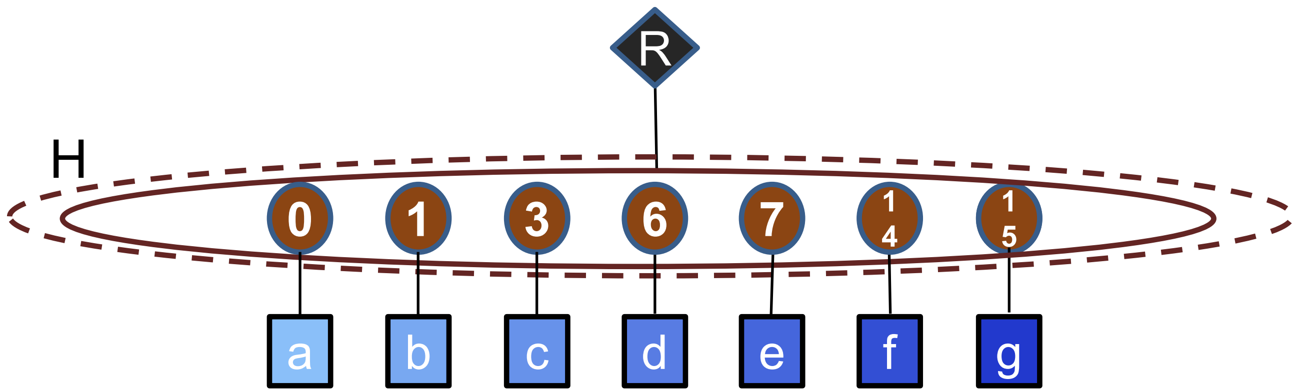
Splitting Fibers – Coordinate Space



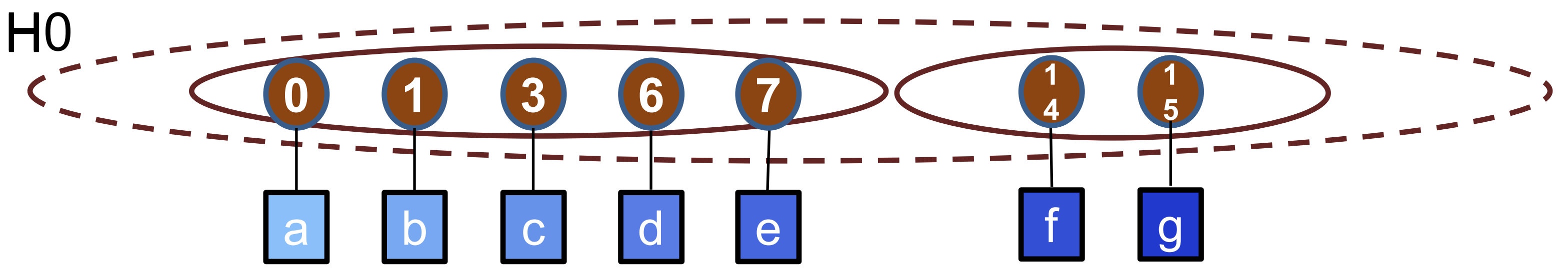
Split uniformly by coordinates (groups of 8 coordinates)



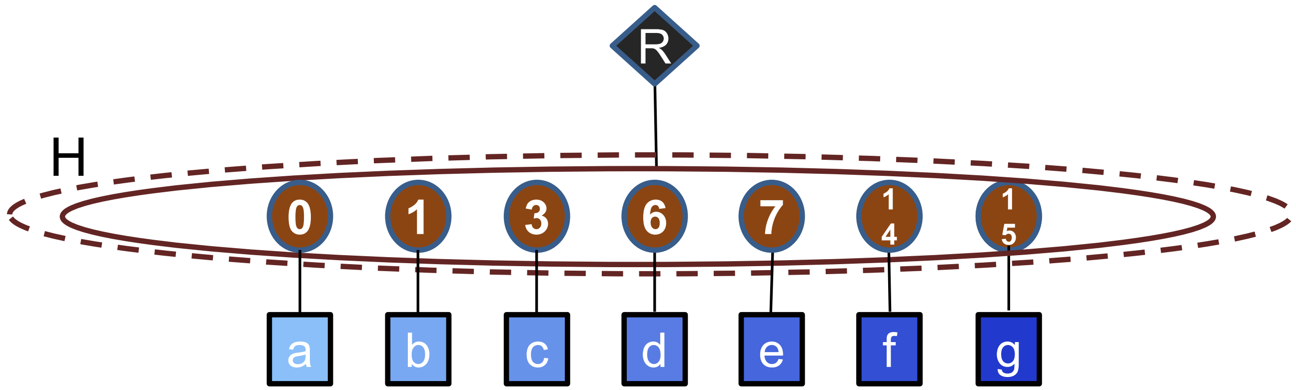
Splitting Fibers – Coordinate Space



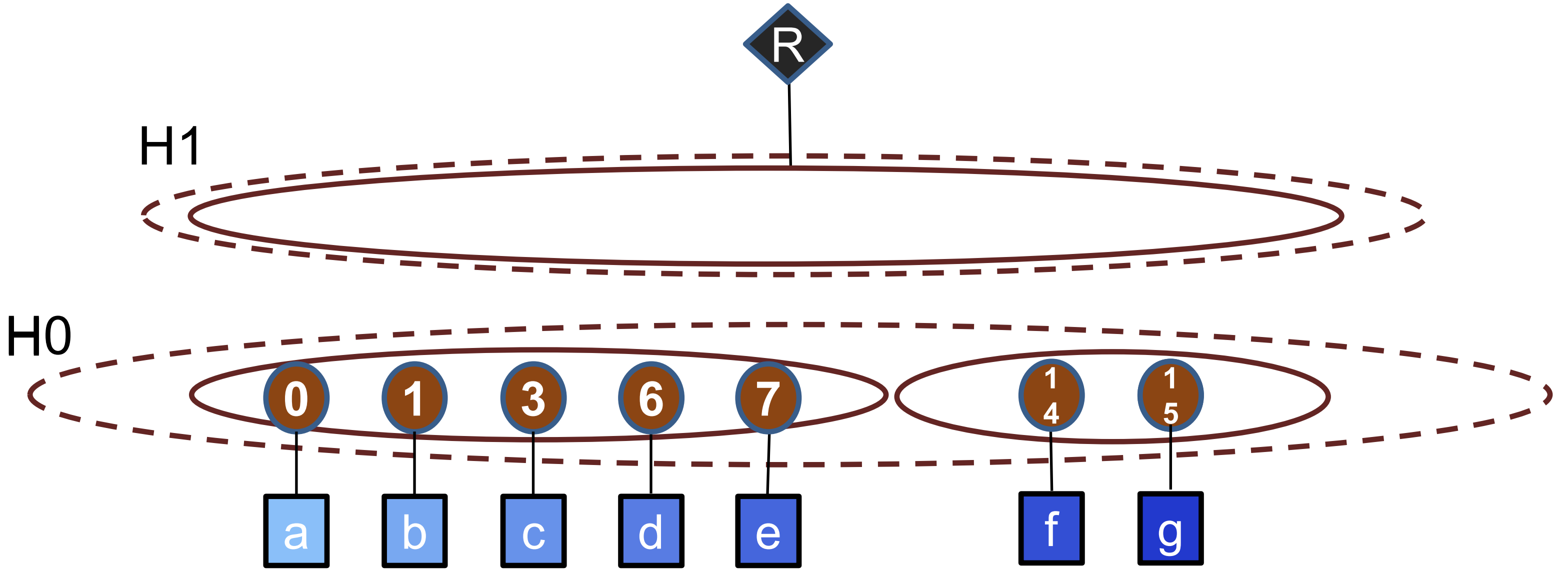
Split uniformly by coordinates (groups of 8 coordinates)



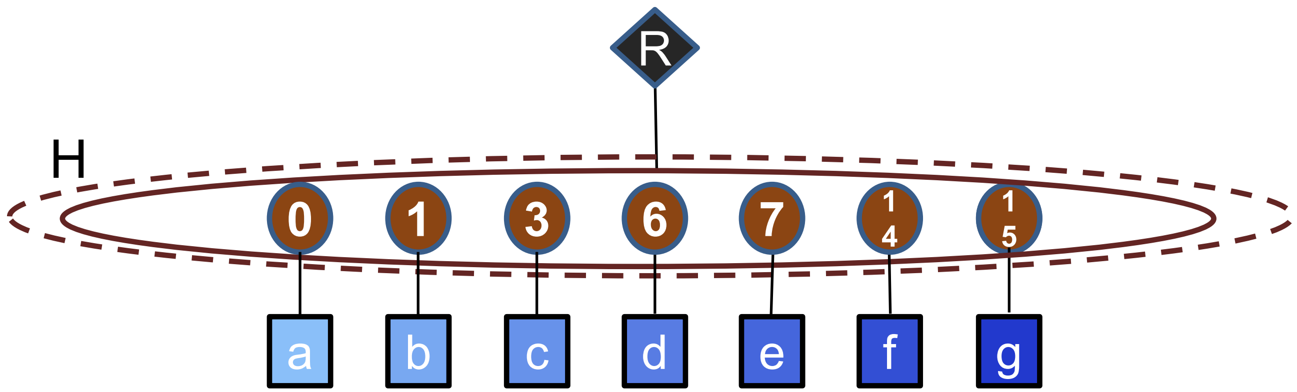
Splitting Fibers – Coordinate Space



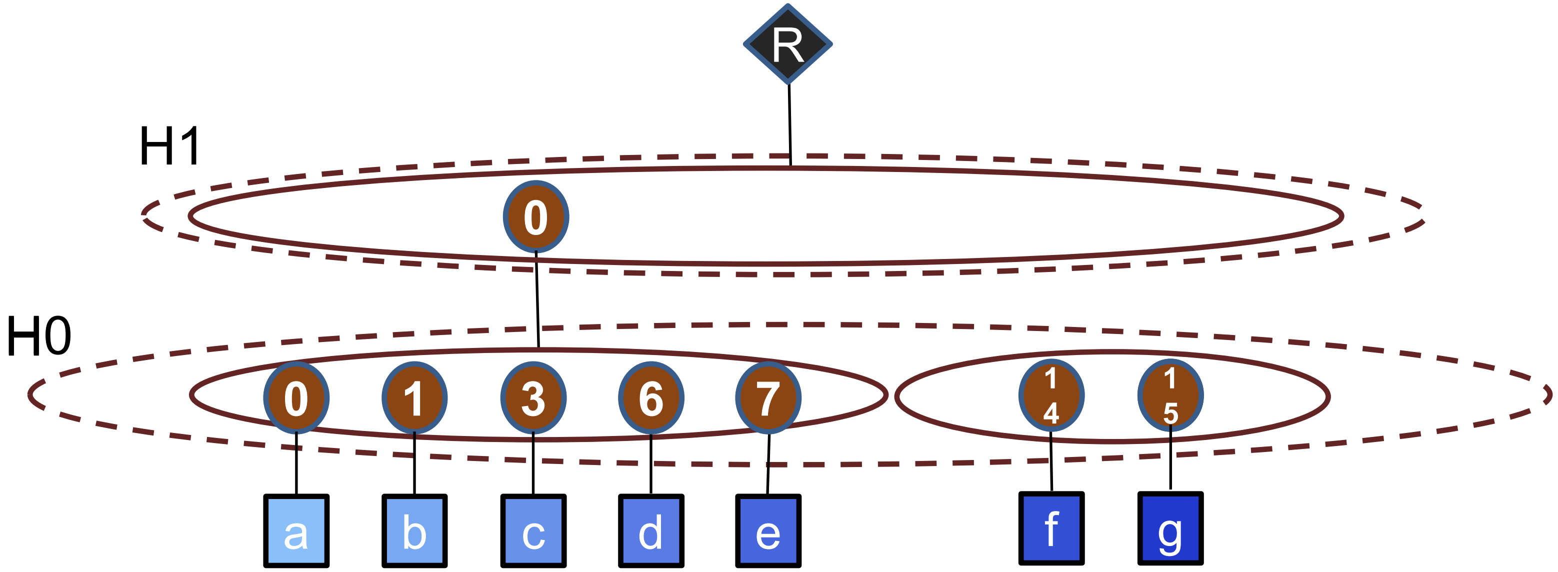
Split uniformly by coordinates (groups of 8 coordinates)



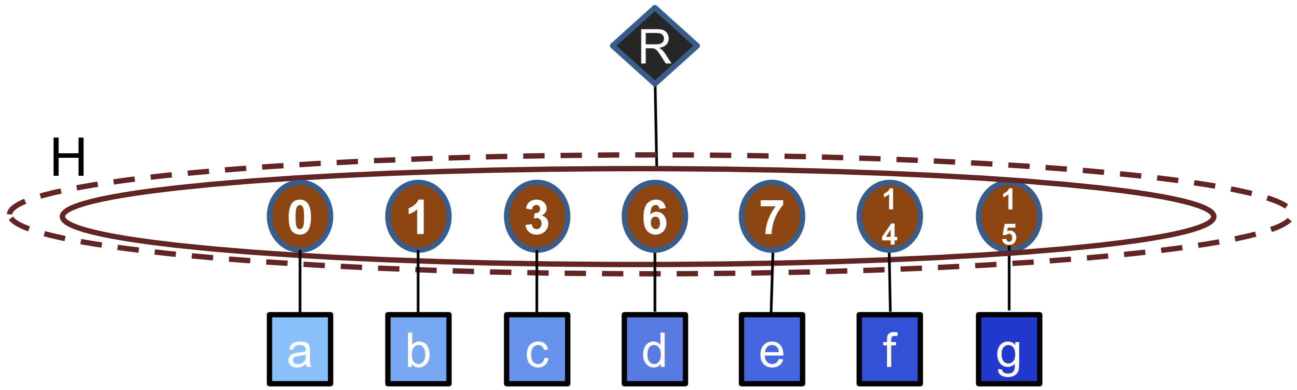
Splitting Fibers – Coordinate Space



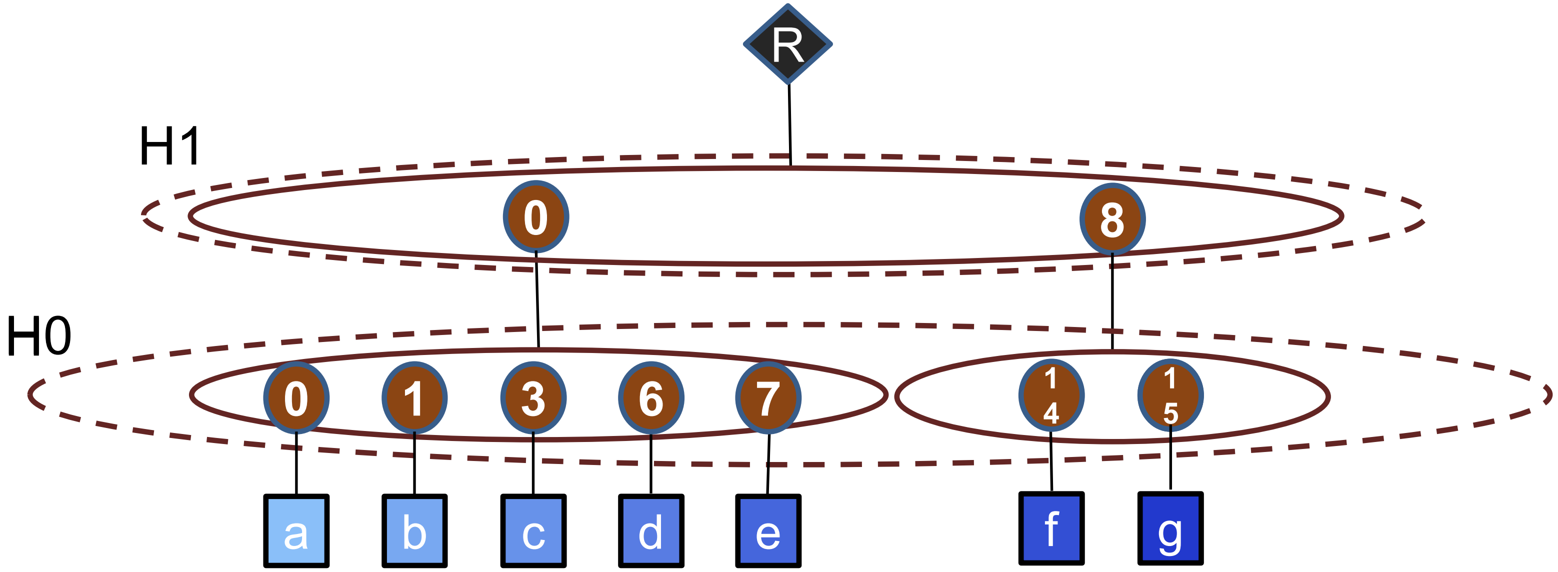
Split uniformly by coordinates (groups of 8 coordinates)



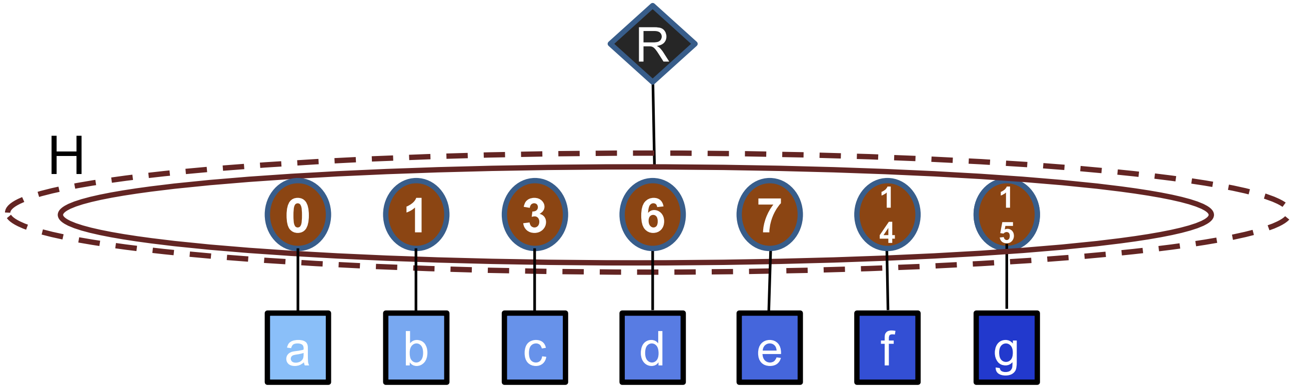
Splitting Fibers – Coordinate Space



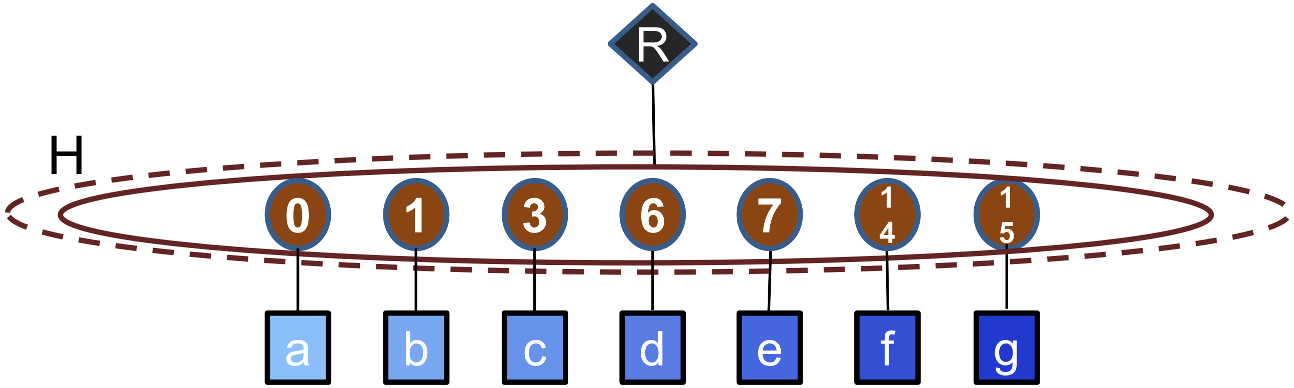
Split uniformly by coordinates (groups of 8 coordinates)



Splitting Fibers – Position Space

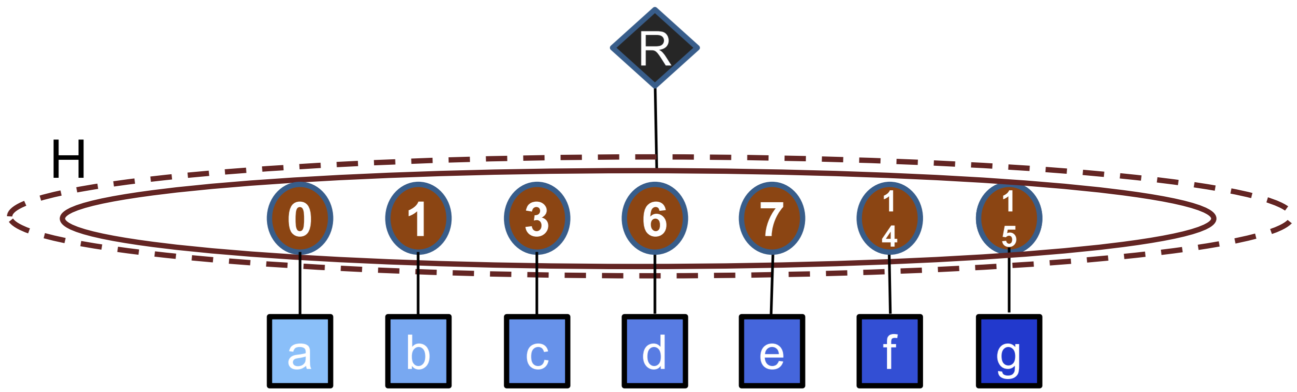


Splitting Fibers – Position Space



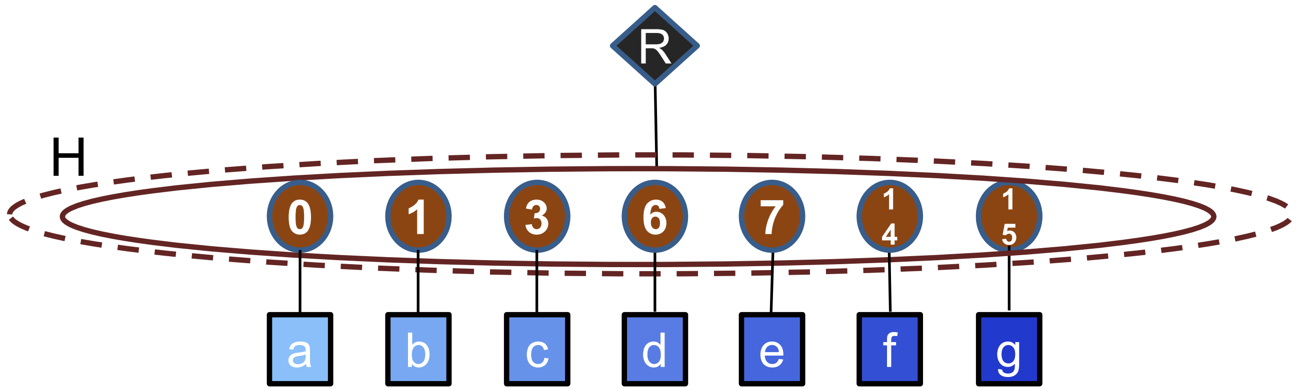
Split evenly by occupancy (groups of 4)

Splitting Fibers – Position Space

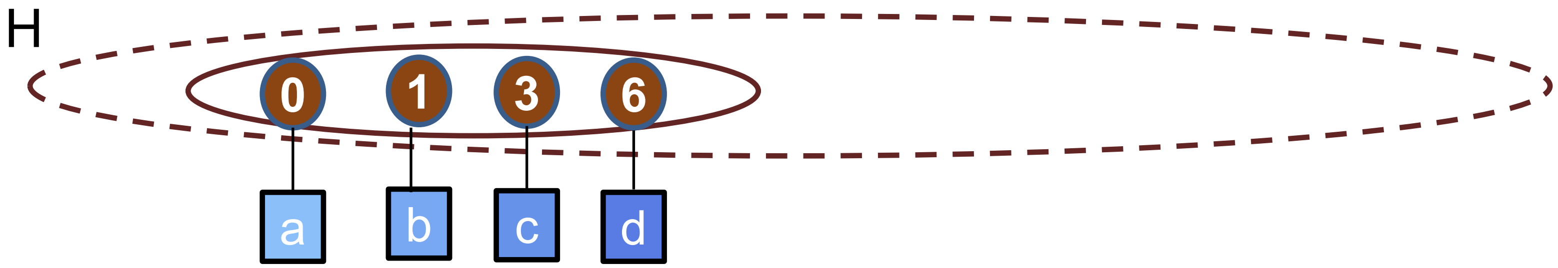


Split evenly by occupancy (groups of 4)

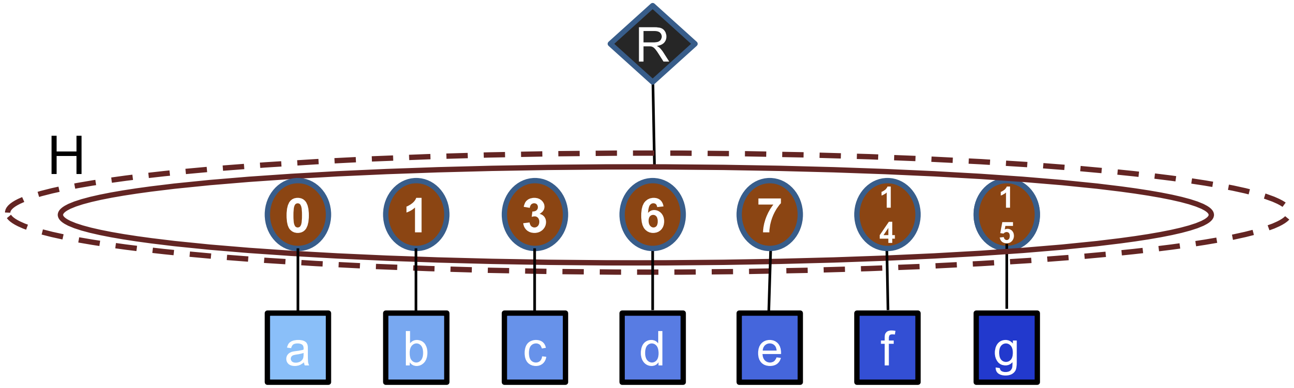
Splitting Fibers – Position Space



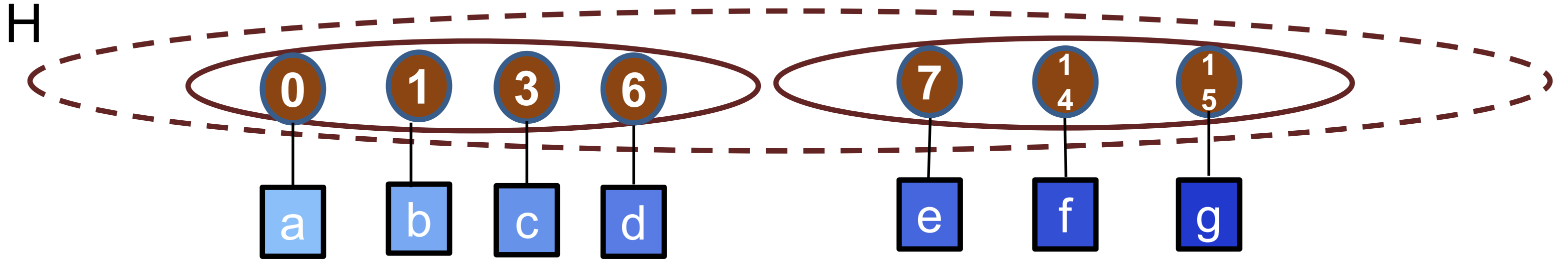
Split evenly by occupancy (groups of 4)



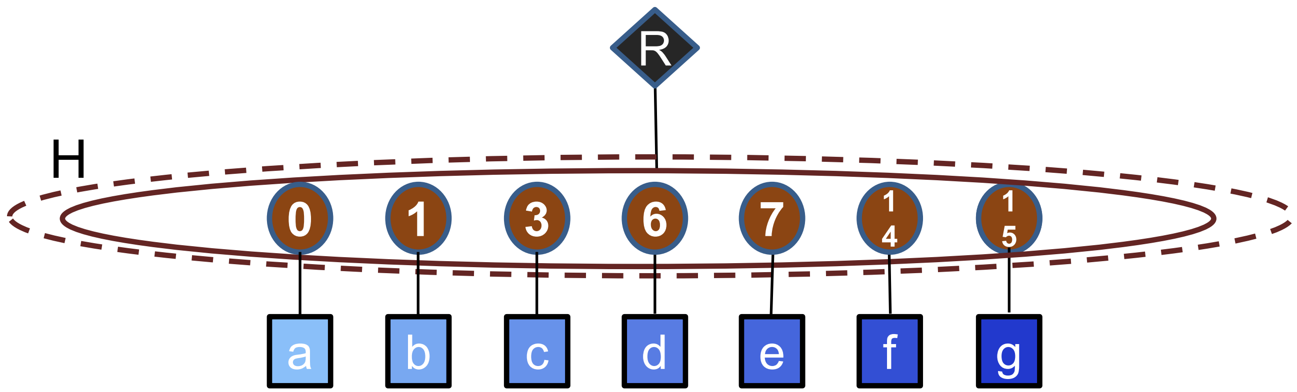
Splitting Fibers – Position Space



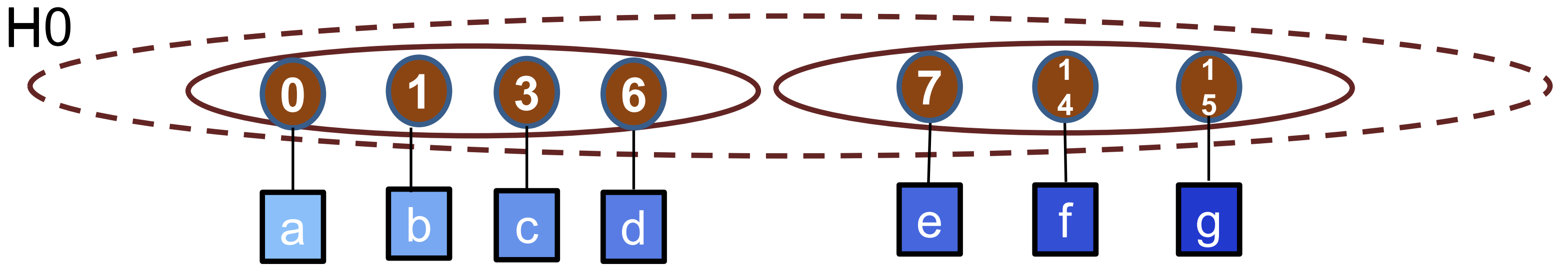
Split evenly by occupancy (groups of 4)



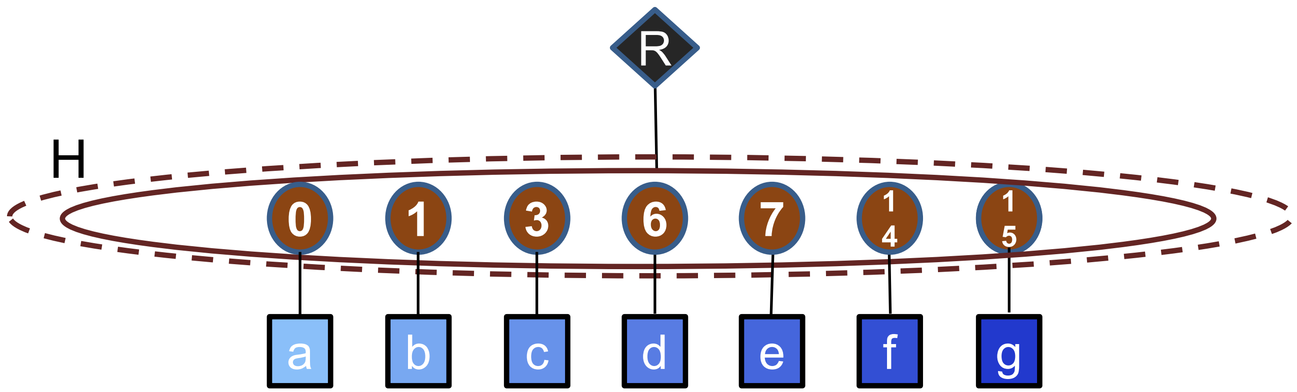
Splitting Fibers – Position Space



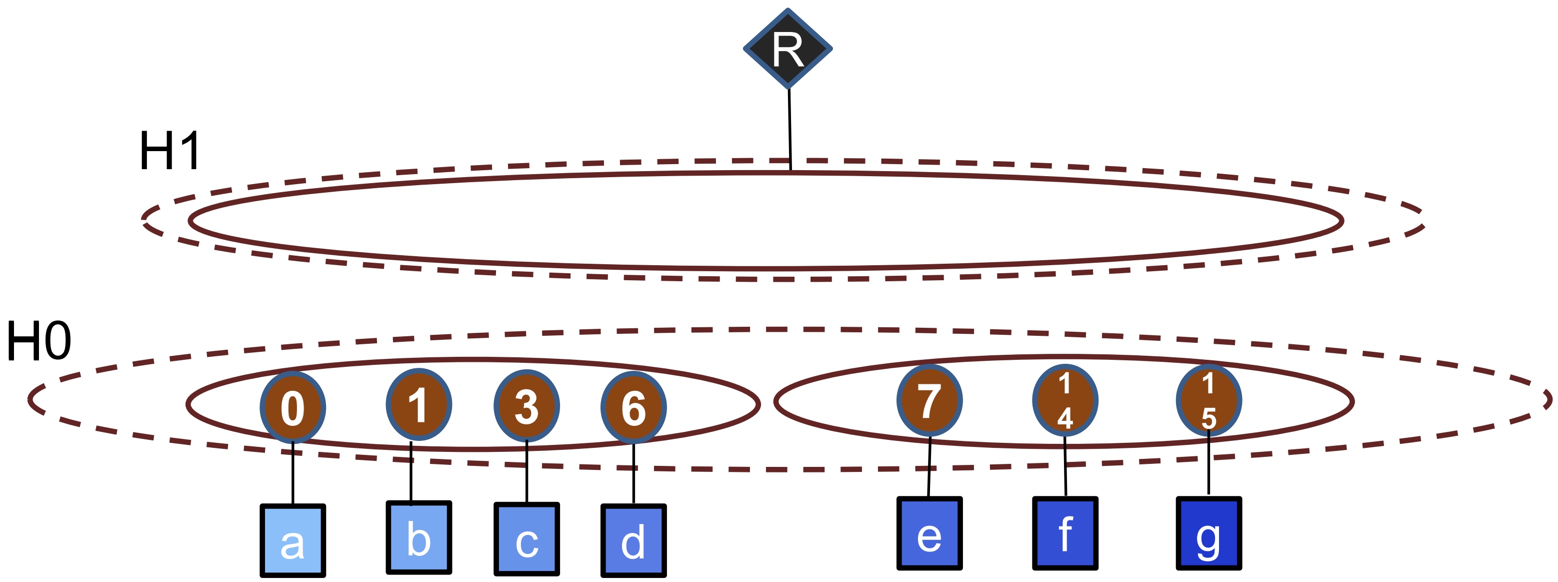
Split evenly by occupancy (groups of 4)



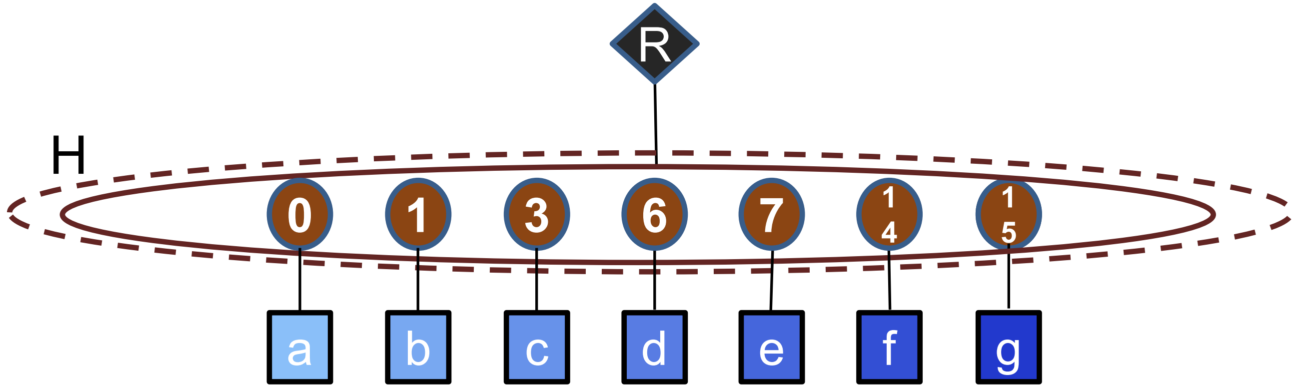
Splitting Fibers – Position Space



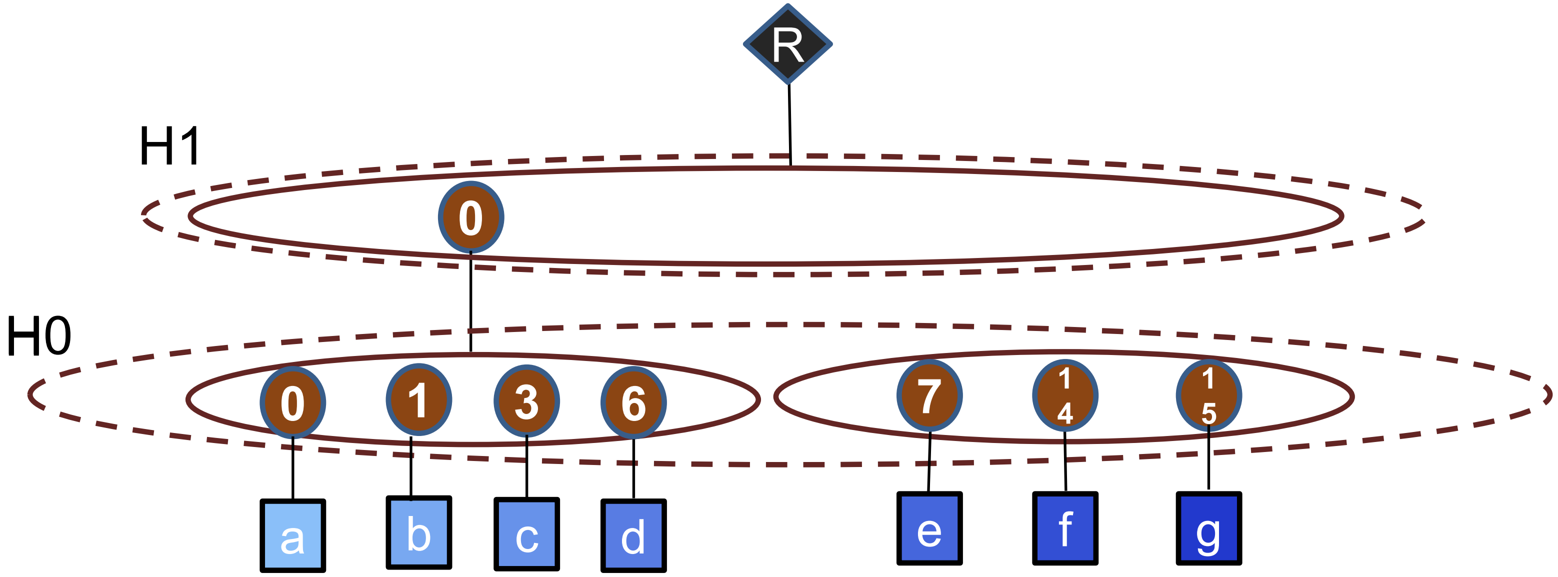
Split evenly by occupancy (groups of 4)



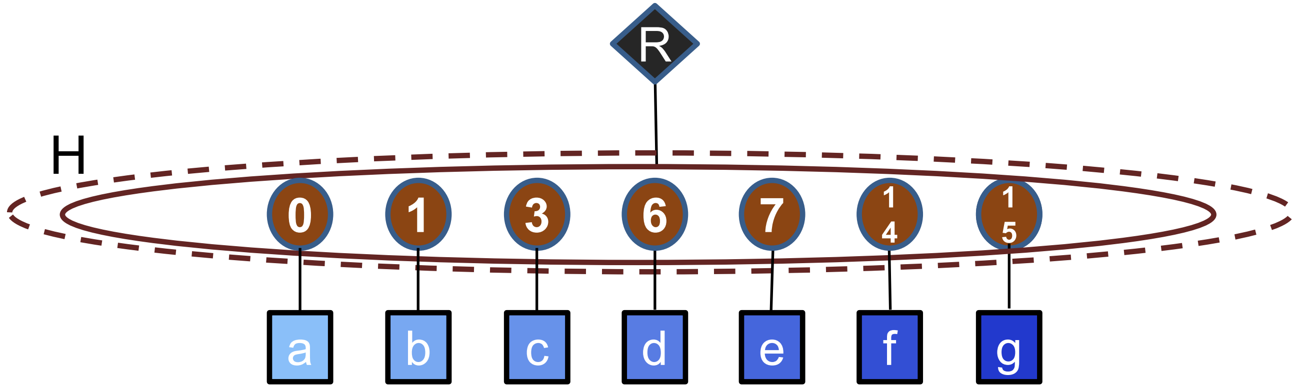
Splitting Fibers – Position Space



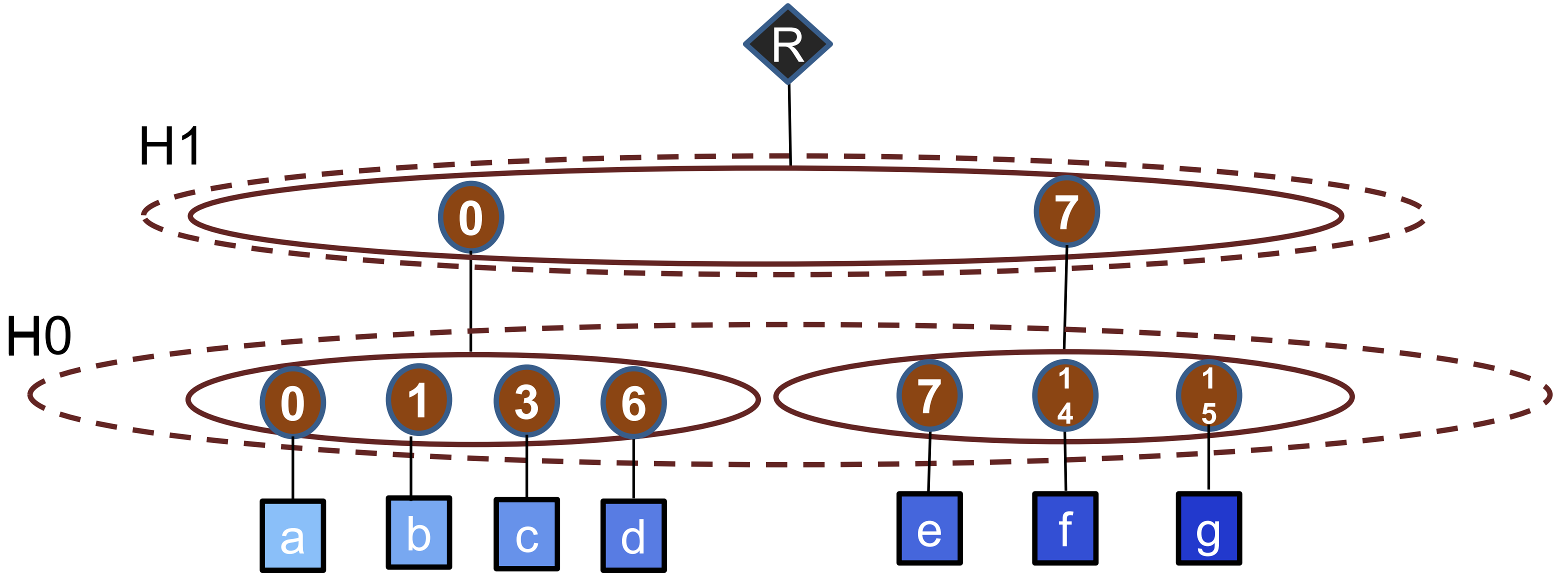
Split evenly by occupancy (groups of 4)



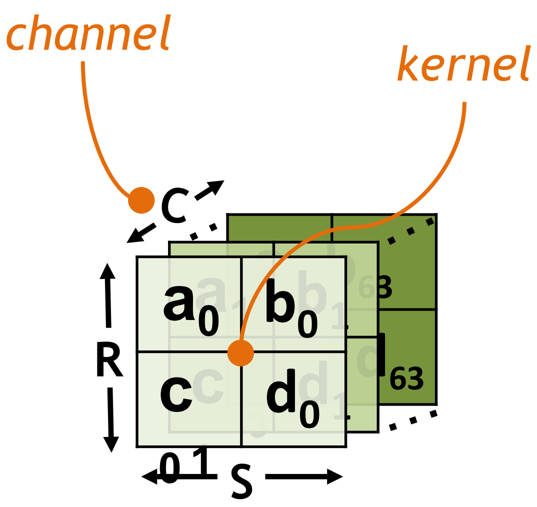
Splitting Fibers – Position Space



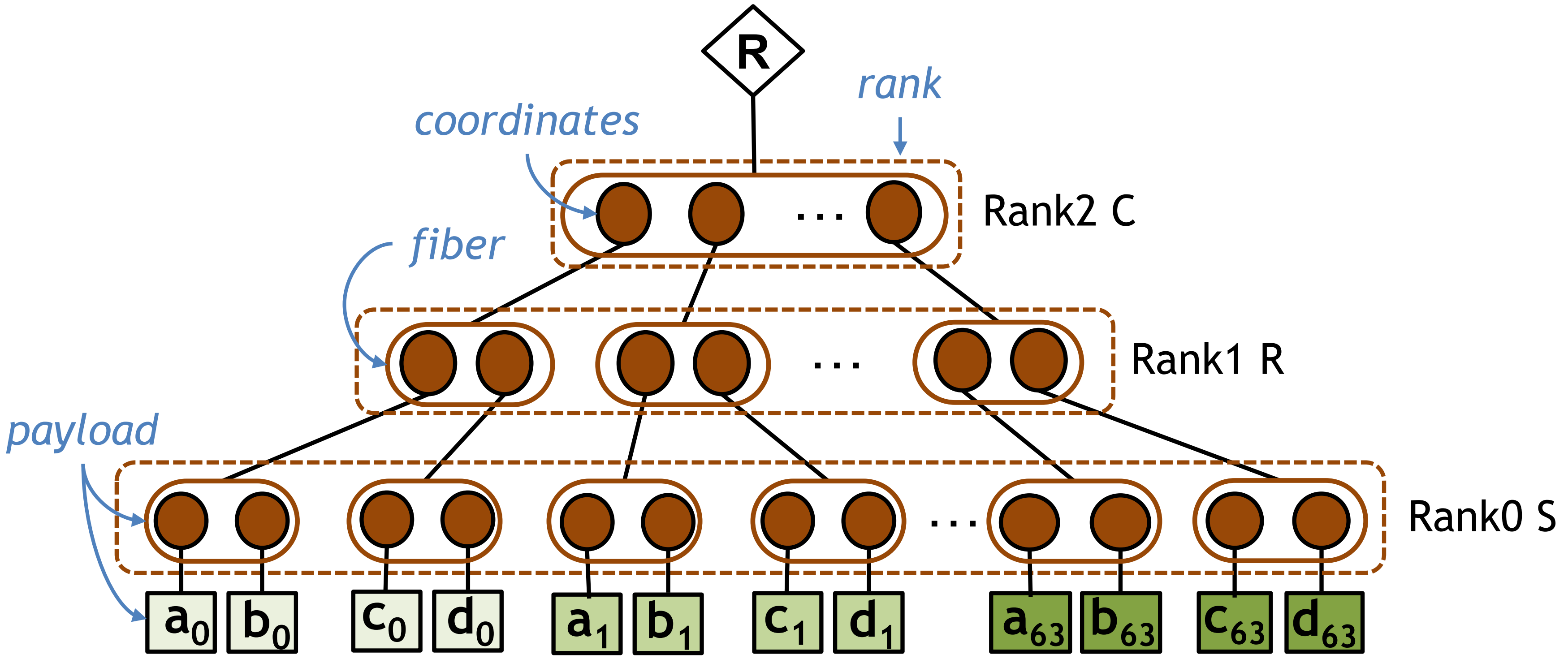
Split evenly by occupancy (groups of 4)



Fibertree Representation of Weight Pruning



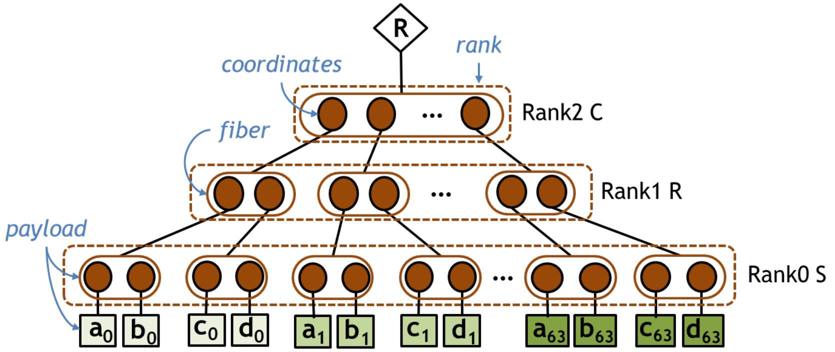
Example 3D Weight Tensor



Fibertree Representation

Each dimension in the original tensor is represented as a rank in the tree

Specification of Channel-based Sparsity

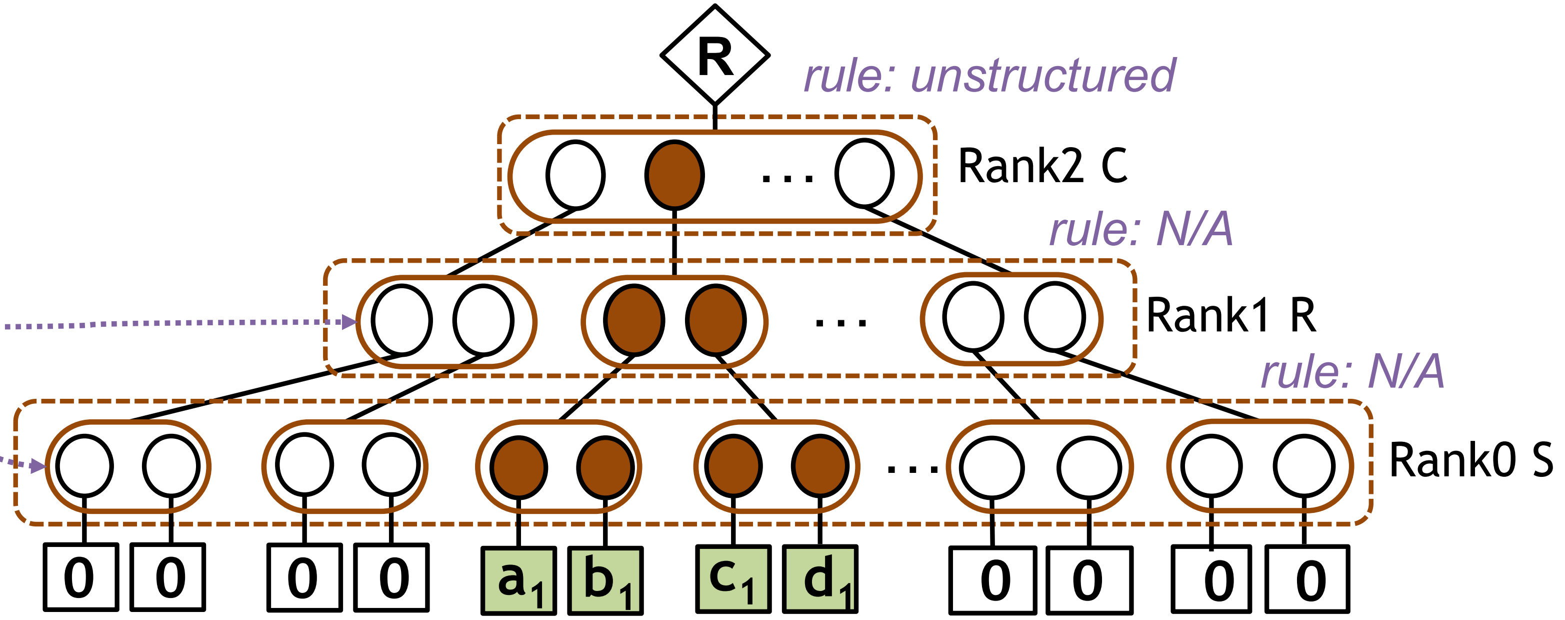


Fibertree Representation of Dense Tensor

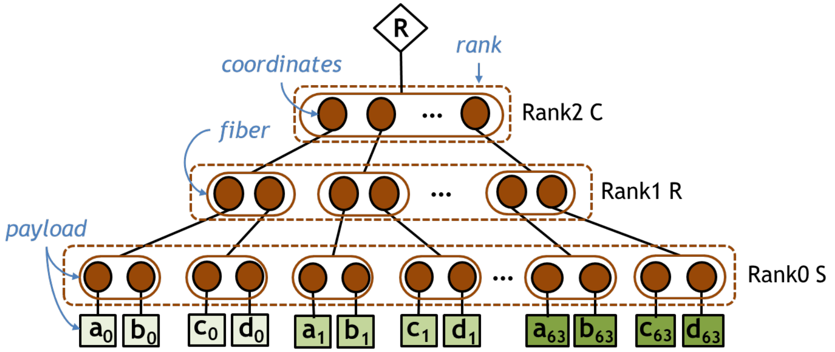
Apply Per-rank Pruning Rule

(identical rules are applied to all fibers in each rank)

Implicit pruning of the coordinates in the payload

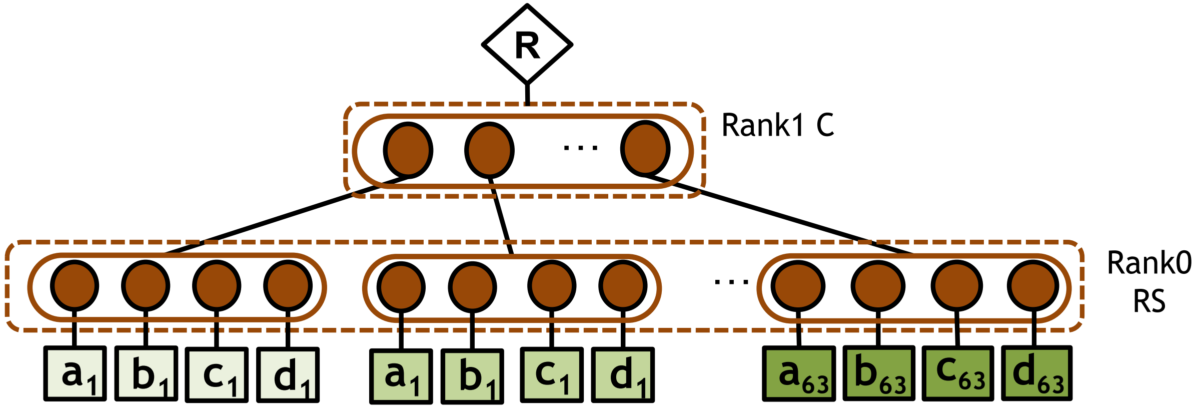


Flattening: Specification of Sub-kernel Sparsity

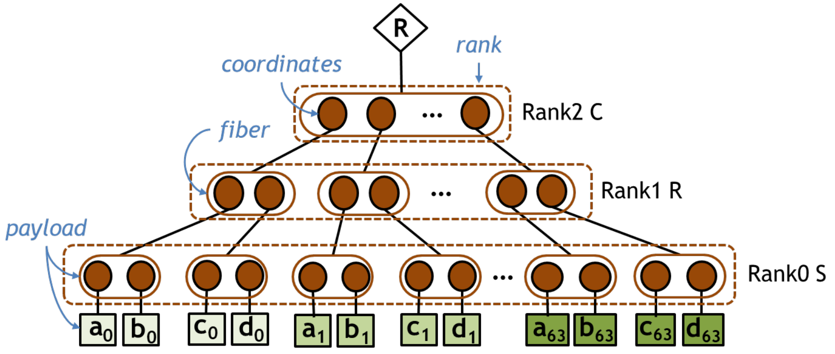


Fibertree Representation of Dense Tensor

① Flatten a Subset of Ranks

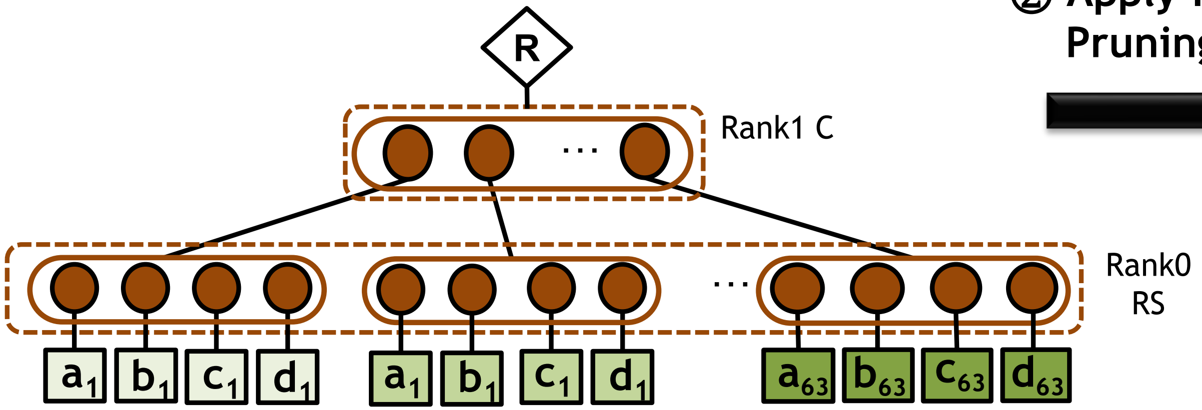


Flattening: Specification of Sub-kernel Sparsity

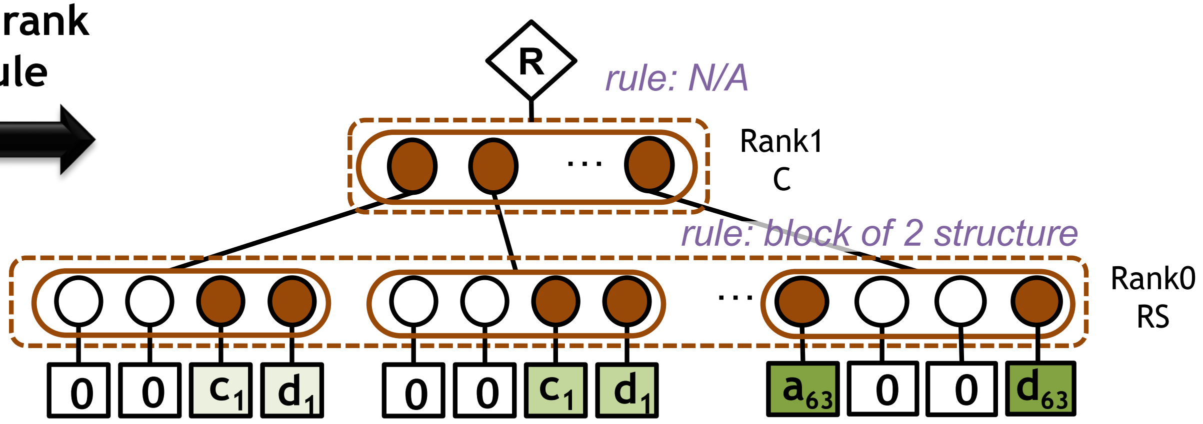


Fibertree Representation of Dense Tensor

① Flatten a Subset of Ranks

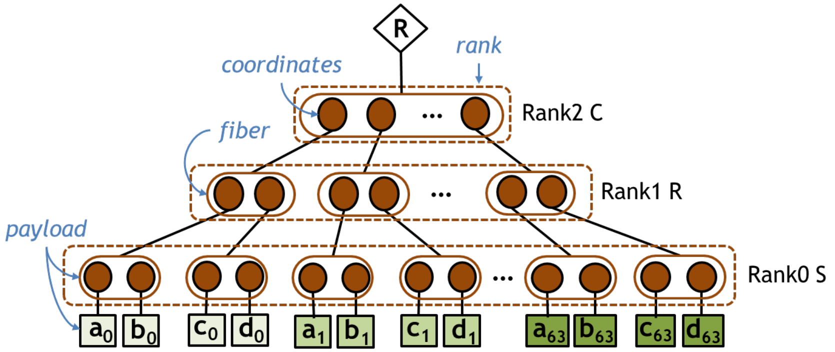


② Apply Per-rank Pruning Rule



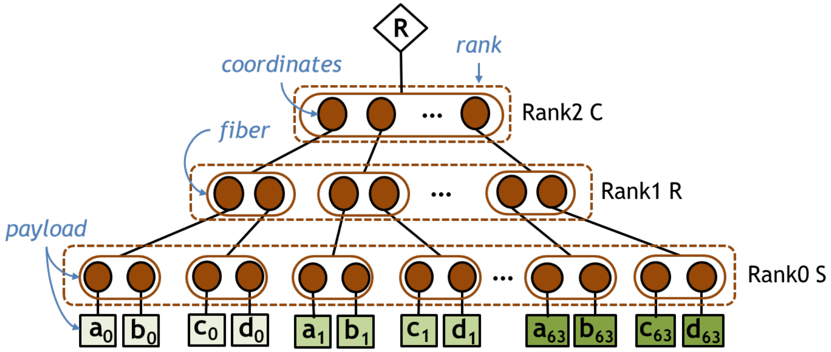
$C \rightarrow RS\ (2\ of\ 4)$

Flattening: Specification of Unstructured Sparsity



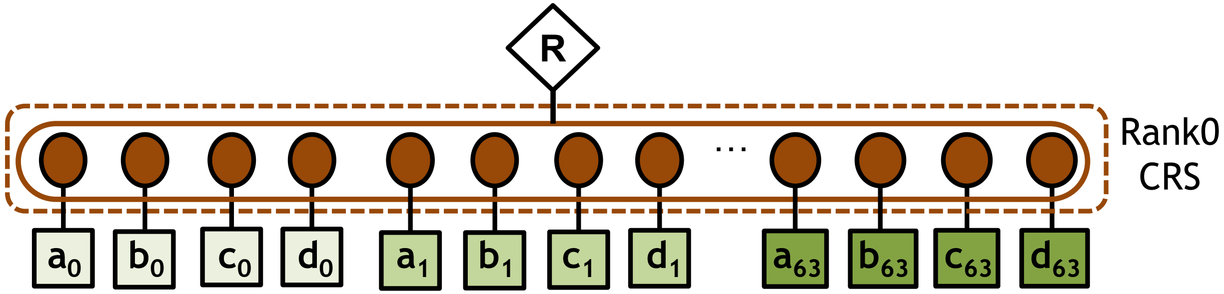
Fibertree Representation of Dense Tensor

Flattening: Specification of Unstructured Sparsity

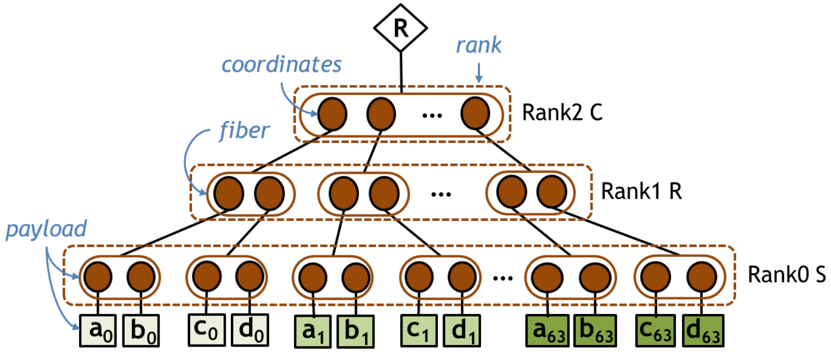


Fibertree Representation of Dense Tensor

① Flatten
All Ranks



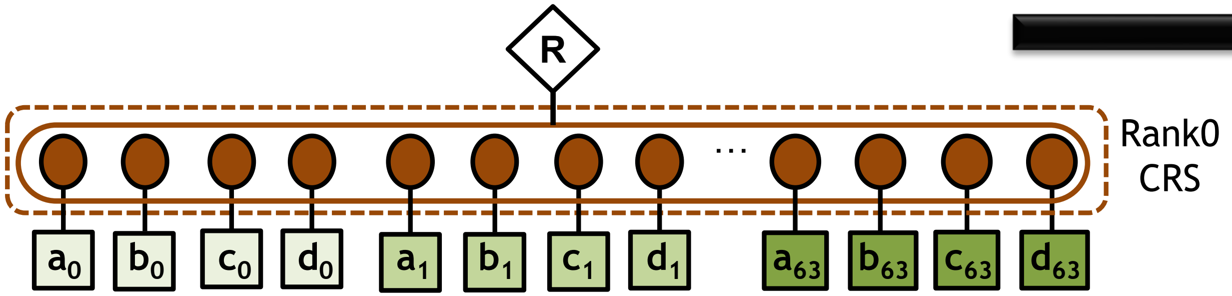
Flattening: Specification of Unstructured Sparsity

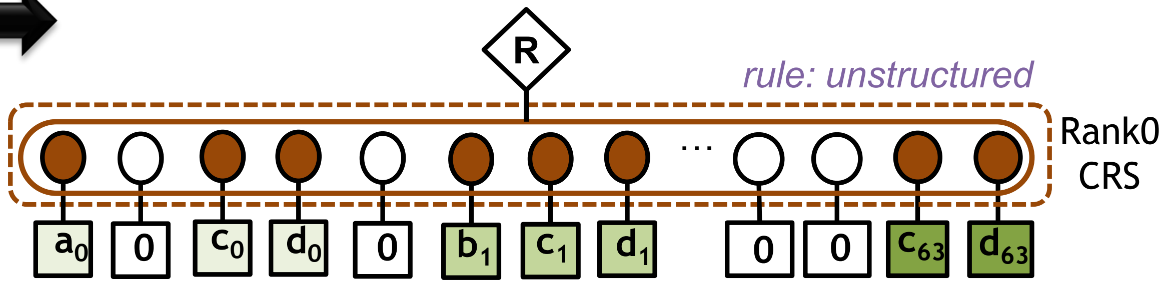


Fibertree Representation of Dense Tensor

① Flatten All Ranks

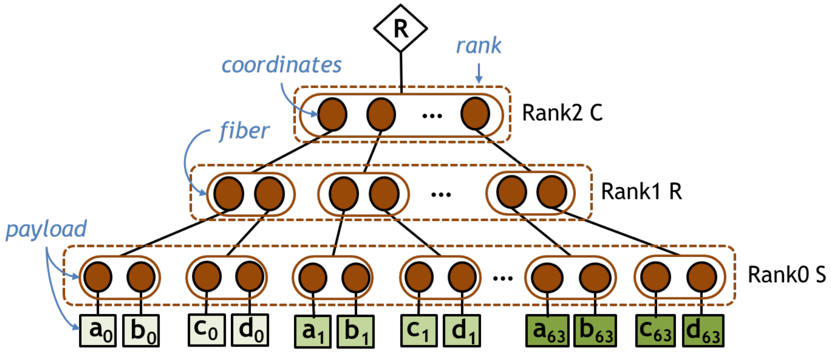
② Apply Per-rank Pruning Rule





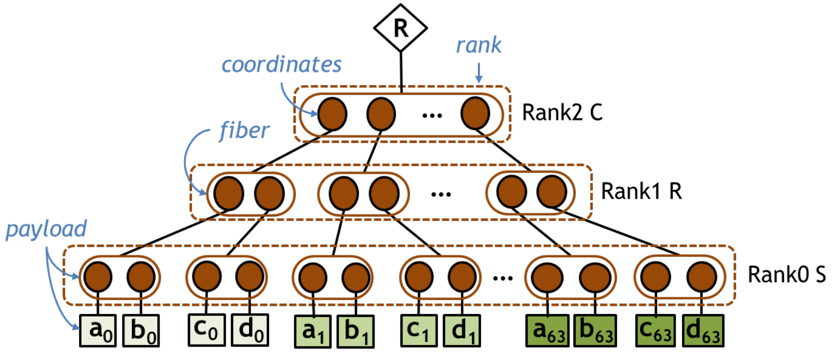
$CRS(unstructured)$

Reordering & Partitioning: Specification of 2:4 Sparsity



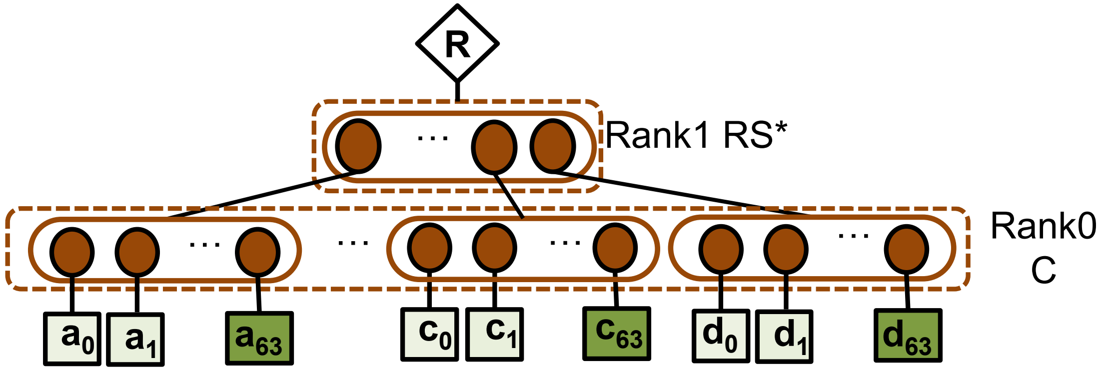
Fibertree Representation of Dense Tensor

Reordering & Partitioning: Specification of 2:4 Sparsity



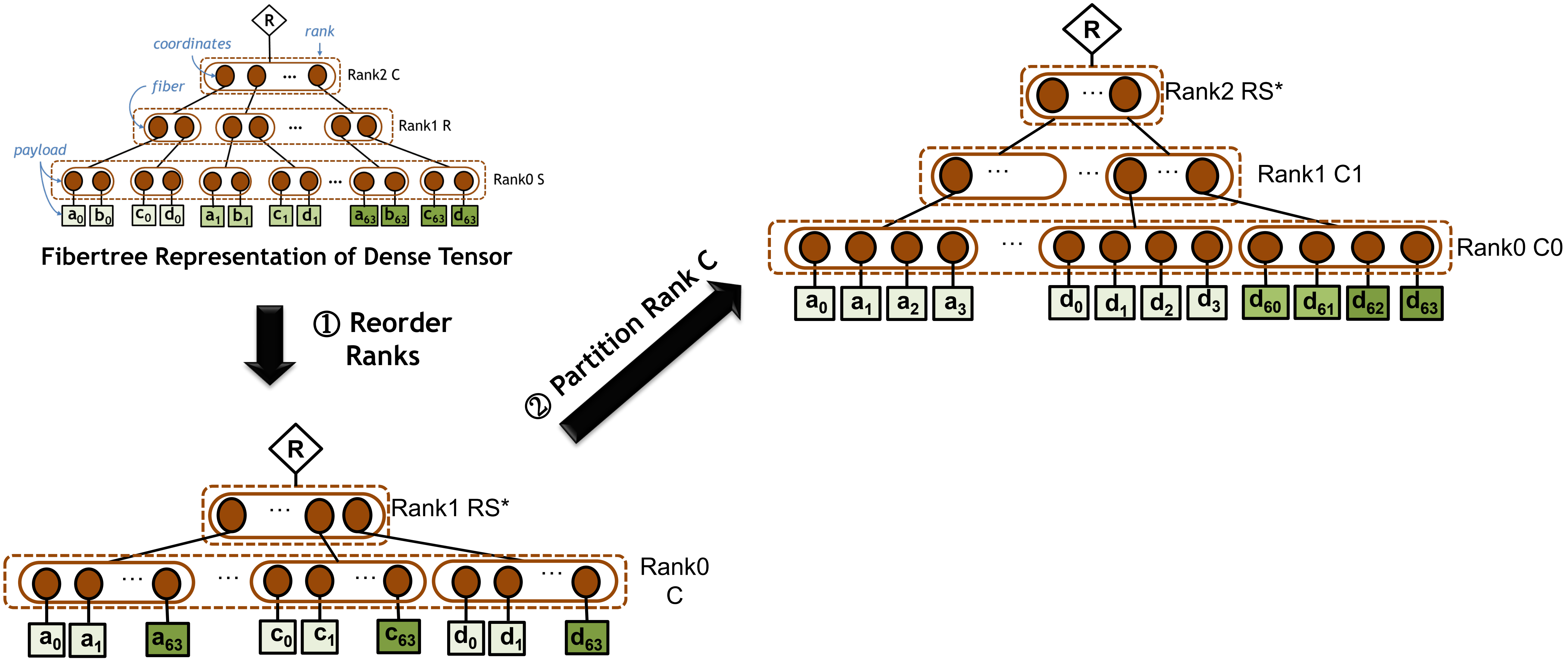
Fibertree Representation of Dense Tensor

① Reorder Ranks



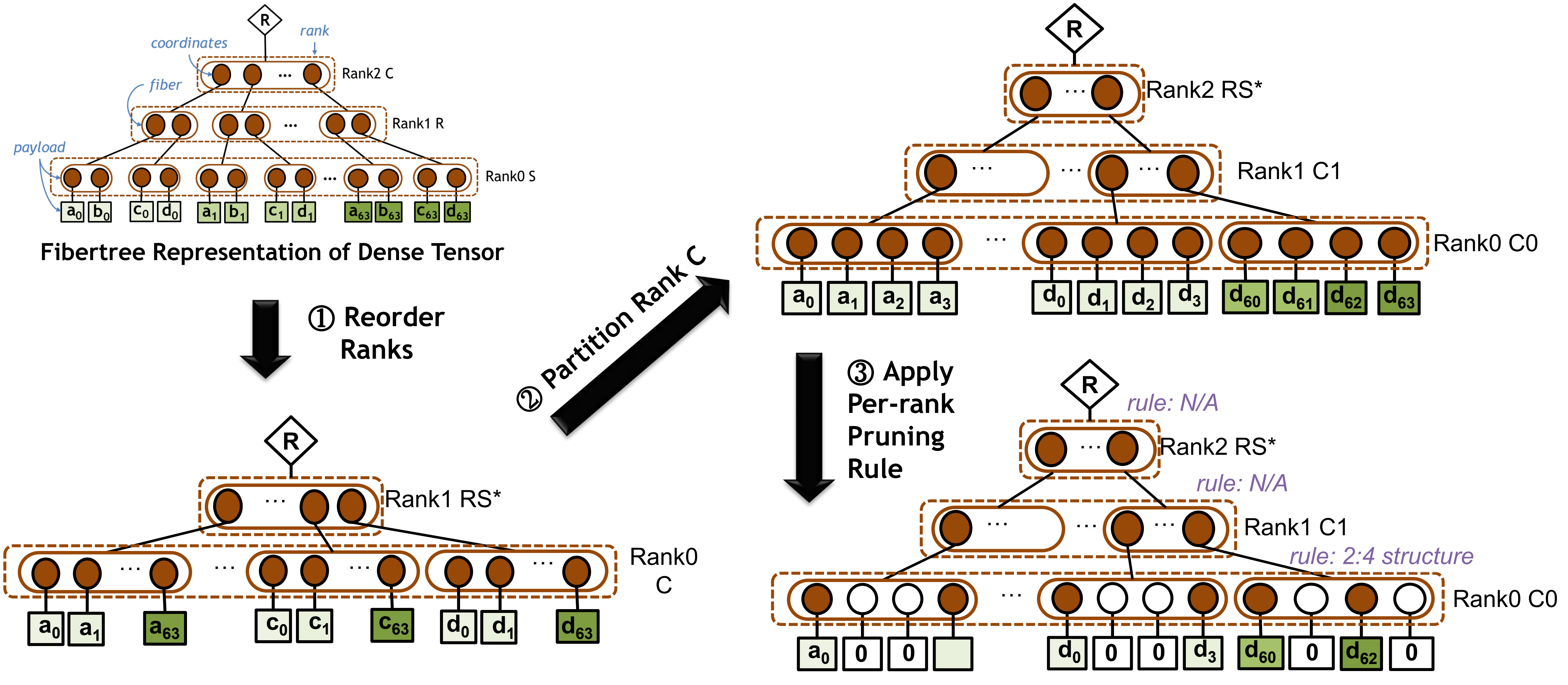
*flattened for ease of presentation, not required

Reordering & Partitioning: Specification of 2:4 Sparsity



*flattened for ease of presentation, not required

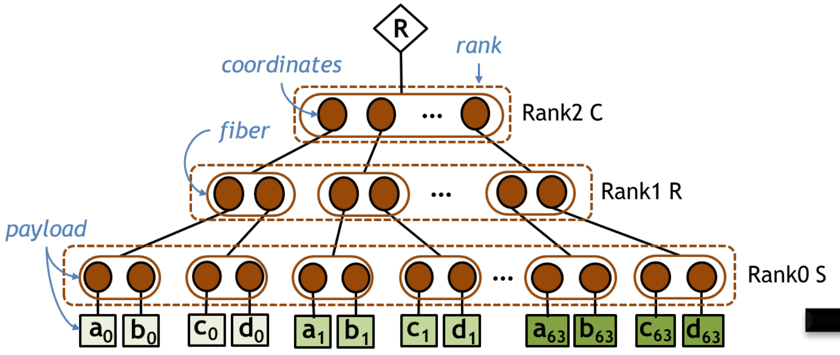
Reordering & Partitioning: Specification of 2:4 Sparsity



$$RS \rightarrow C_1 \rightarrow C_0(2{:}4)$$

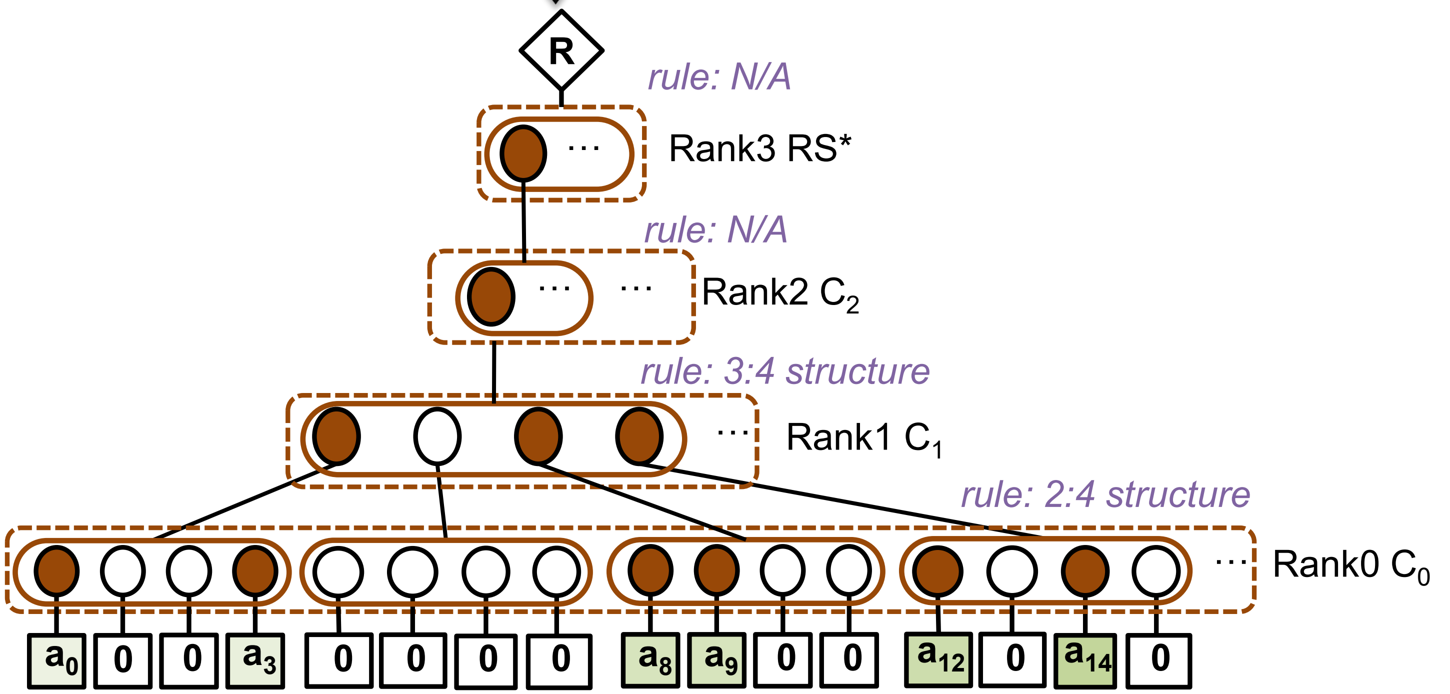
*flattened for ease of presentation, not required

Hierarchical Structured Sparsity (HSS)



Fibertree Representation of Dense Tensor

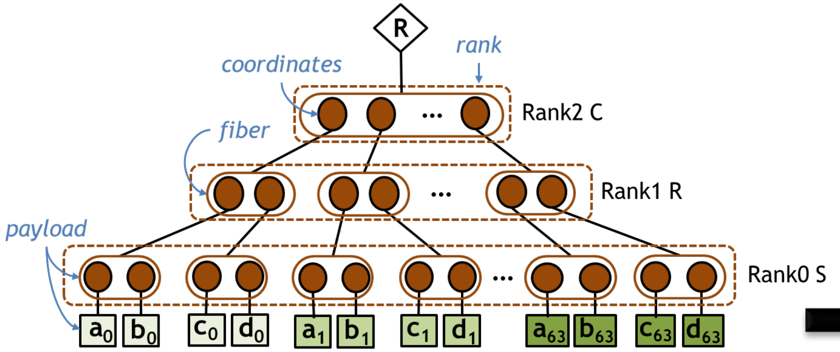
① **Reorder Ranks**
② **Partition Rank C into N ranks (N>=2) , e.g., N=3 as shown below**
③ **Apply Per-rank Pruning Rule**



$RS \rightarrow C_2 \rightarrow C_1(3:4) \rightarrow C_0(2:4)$

*flattened for ease of presentation, not required

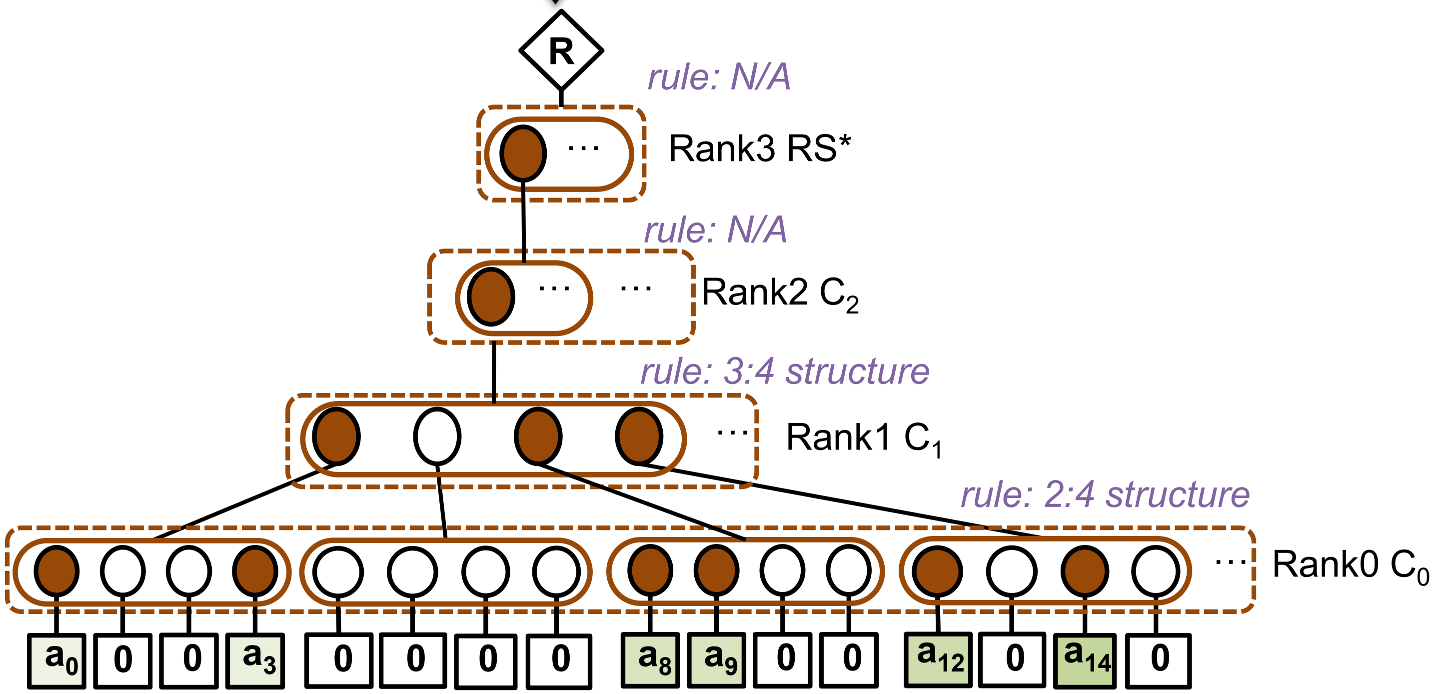
Hierarchical Structured Sparsity (HSS)



Fibertree Representation of Dense Tensor

① Reorder Ranks
② Partition Rank C into N ranks (N>=2) , e.g., N=3 as shown below
③ Apply Per-rank Pruning Rule

- **N-1 rank HSS defined as**
  - $RS \rightarrow C_{N-1} \rightarrow C_{N-2}(G_{N-2}{:}\, H_{N-2}) \rightarrow \ldots \rightarrow C_1(3{:}\,4) \rightarrow C_0(2{:}\,4)$
- **HSS qualitative difference: allows pruning rules for more thank one ranks**
- **HSS provides a systematic and modularized way to represent a large number of sparsity degrees**



$RS \rightarrow C_2 \rightarrow C_1(3{:}\,4) \rightarrow C_0(2{:}\,4)$

*flattened for ease of presentation, not required

Next: Sparse Architectures