CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

CSDS 451: Designing High Performant Systems for AI

Lecture 18

10/30/2025

Sanmukh Kuppannagari

sanmukh.kuppannagari@case.edu

https://sanmukh.research.st/

Case Western Reserve University

Outline

- Accelerating Sparse Transformers – Part I

Announcements

- PA 2 due this Saturday

- WA 3 will be out by Saturday

Outline

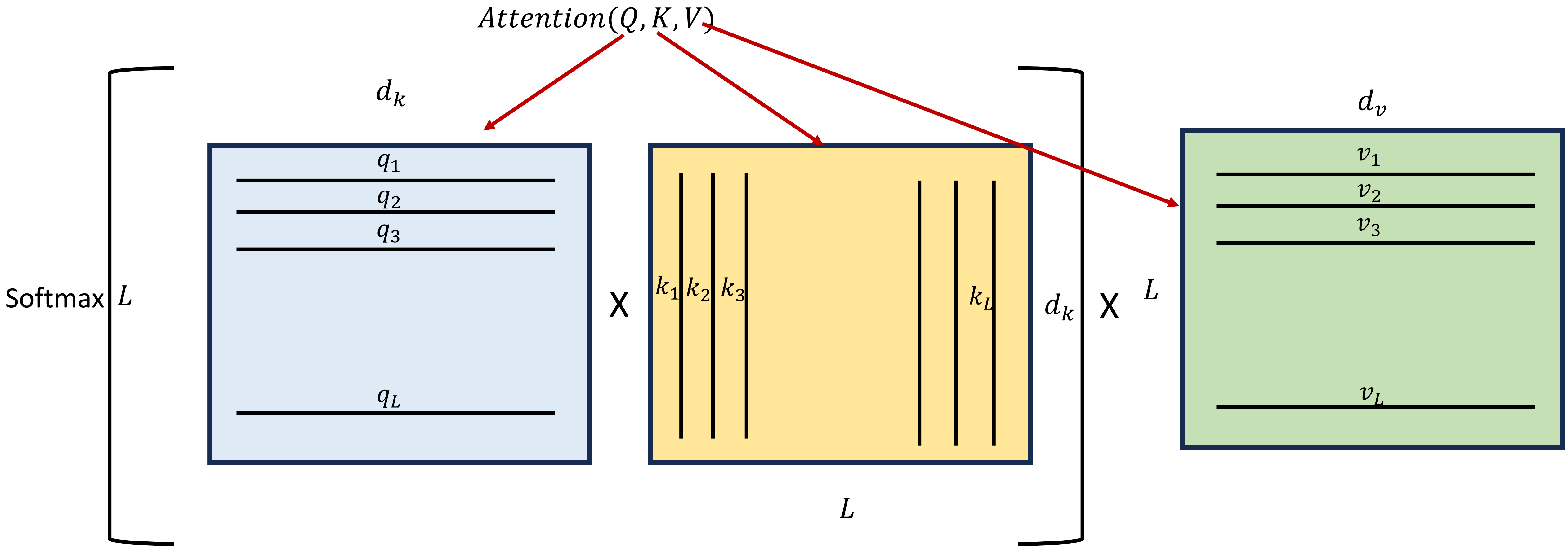
- Accelerating Sparse Transformers – Part I

Attention Mechanism – Fast Facts

- Three Key Operations
  - Operation #1: $Y = QK^T$: Product of Q and K^T
  - Operation #2: $Z = Softmax(Y)$
  - Operation #3: $O = ZV$: Product of Z and V matrices
  - Q, K^T, V, Z: Dense matrices

Case School of Engineering
Case Western Reserve University

Attention



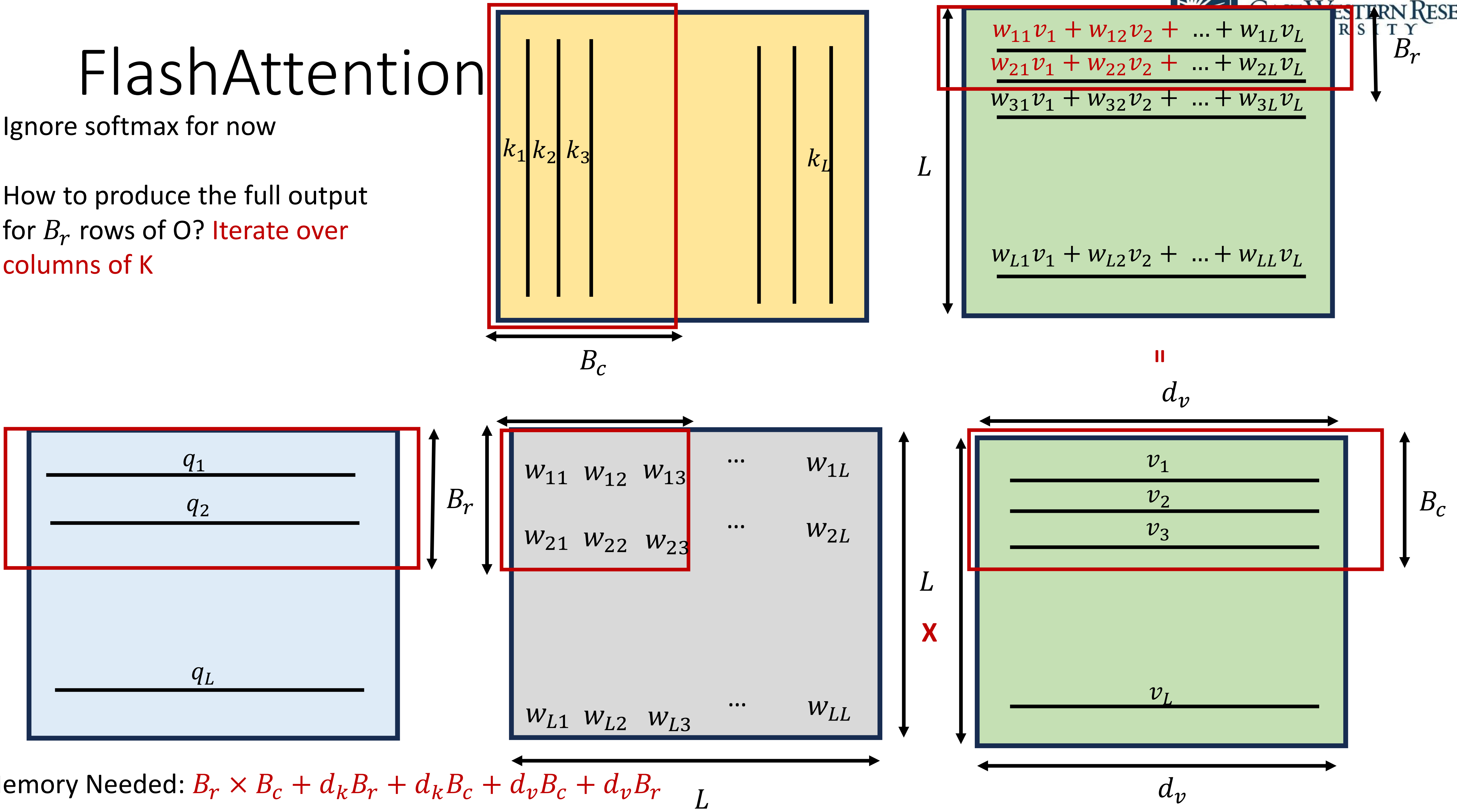
Q Matrix: Each row is a query
K^T Matrix: Each column is a key
V Matrix: Each row is a value

X: Matrix MM

FlashAttention

Ignore softmax for now

How to produce the full output for B_r rows of O? Iterate over columns of K

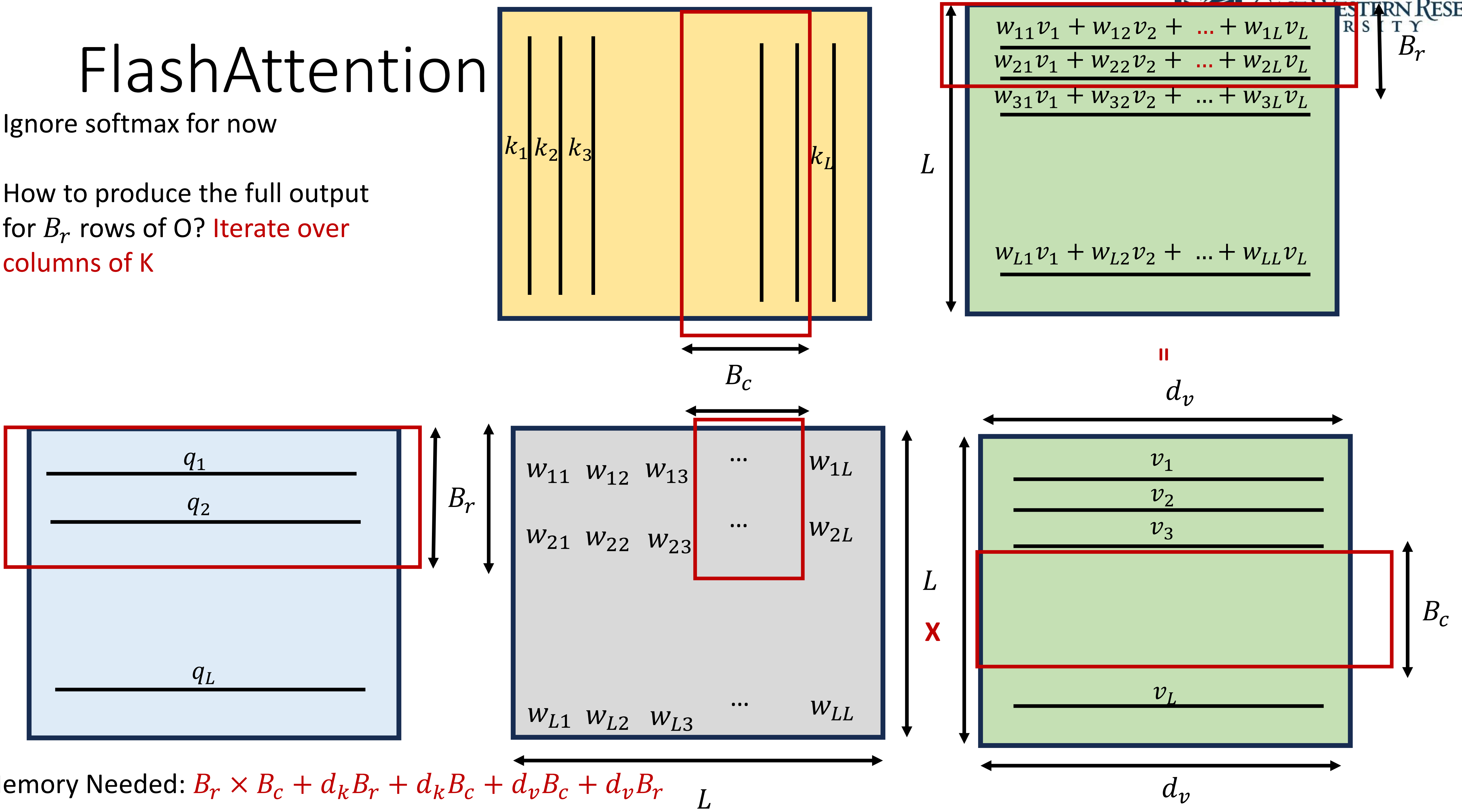


Memory Needed: $B_r \times B_c + d_k B_r + d_k B_c + d_v B_c + d_v B_r$

FlashAttention

Ignore softmax for now

How to produce the full output for B_r rows of O? Iterate over columns of K

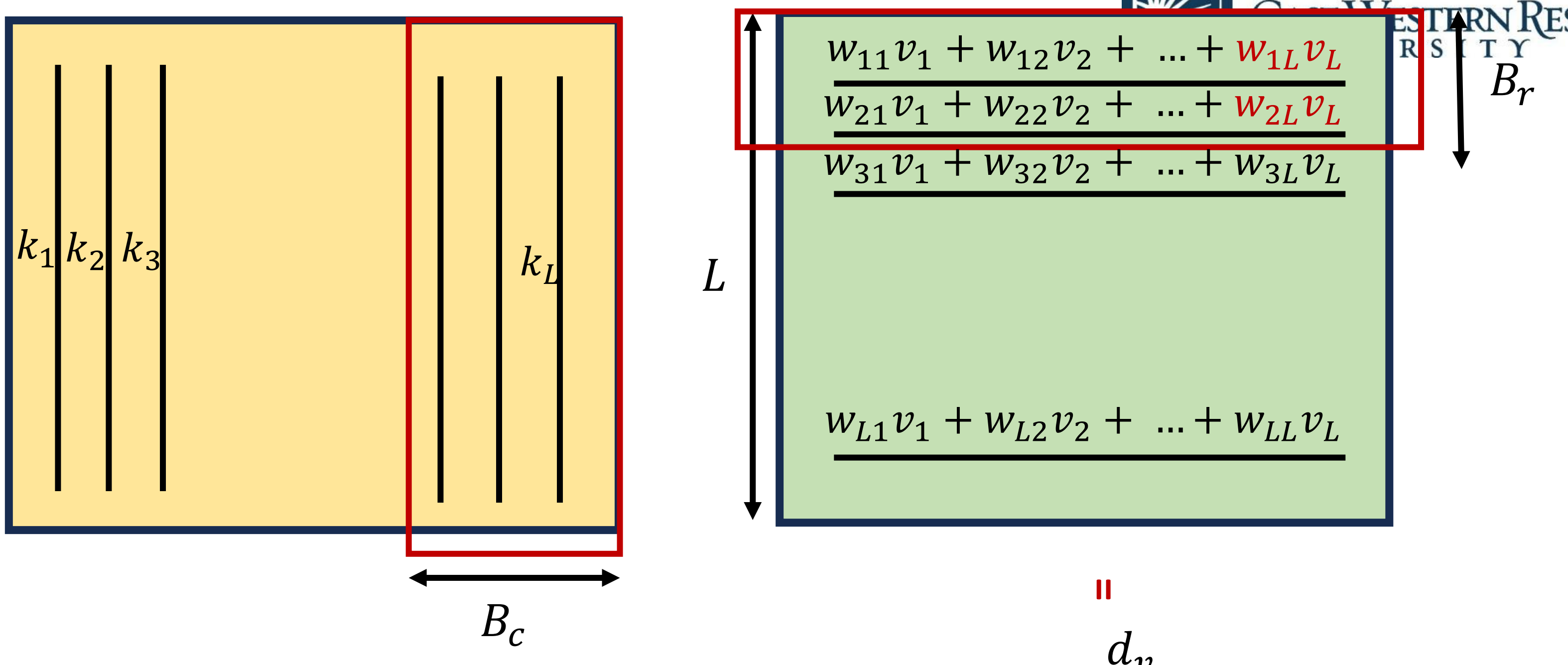


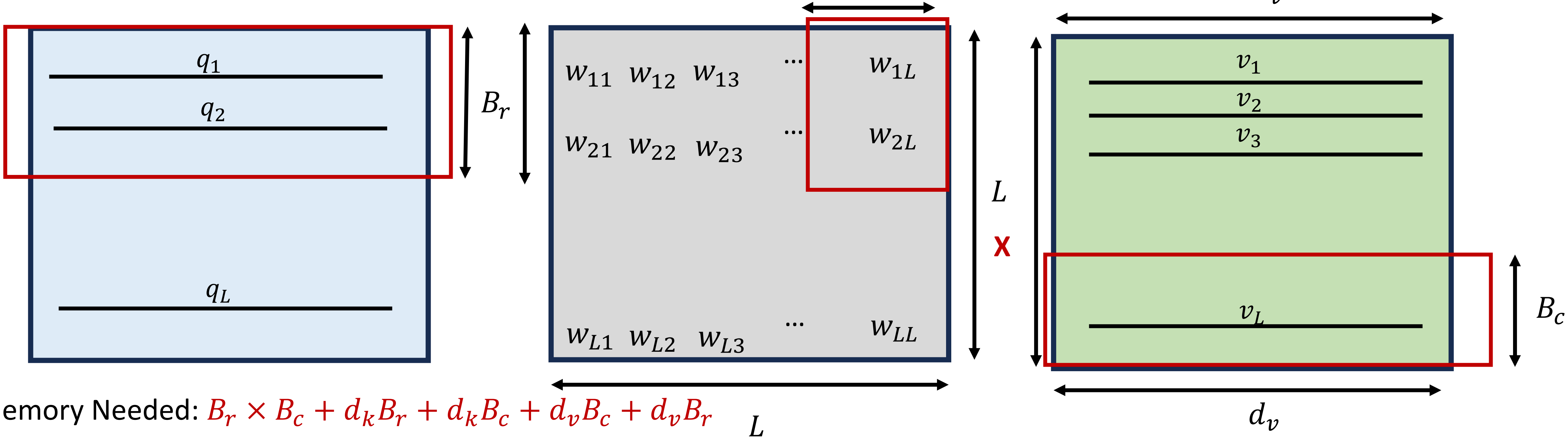
Memory Needed: $B_r \times B_c + d_k B_r + d_k B_c + d_v B_c + d_v B_r$

FlashAttention

Ignore softmax for now

How to produce the full output for B_r rows of O? Iterate over columns of K



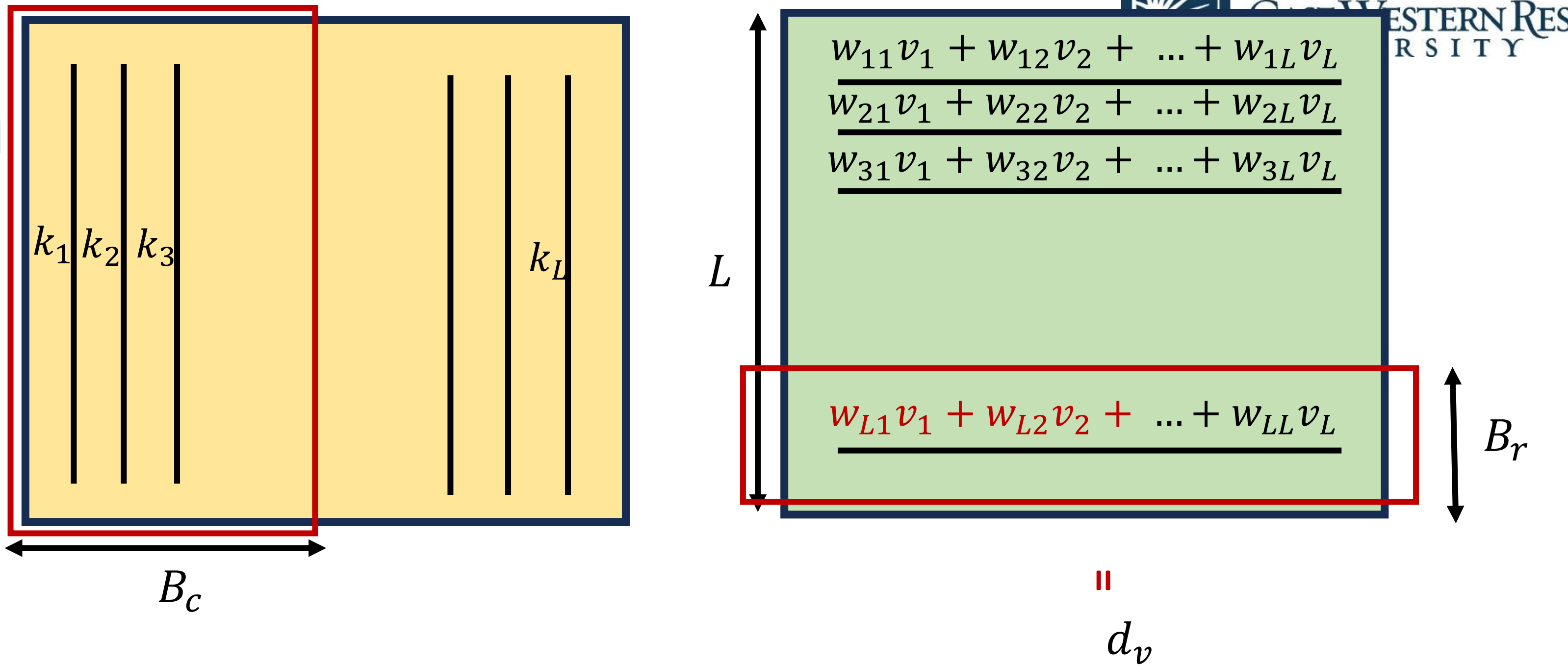


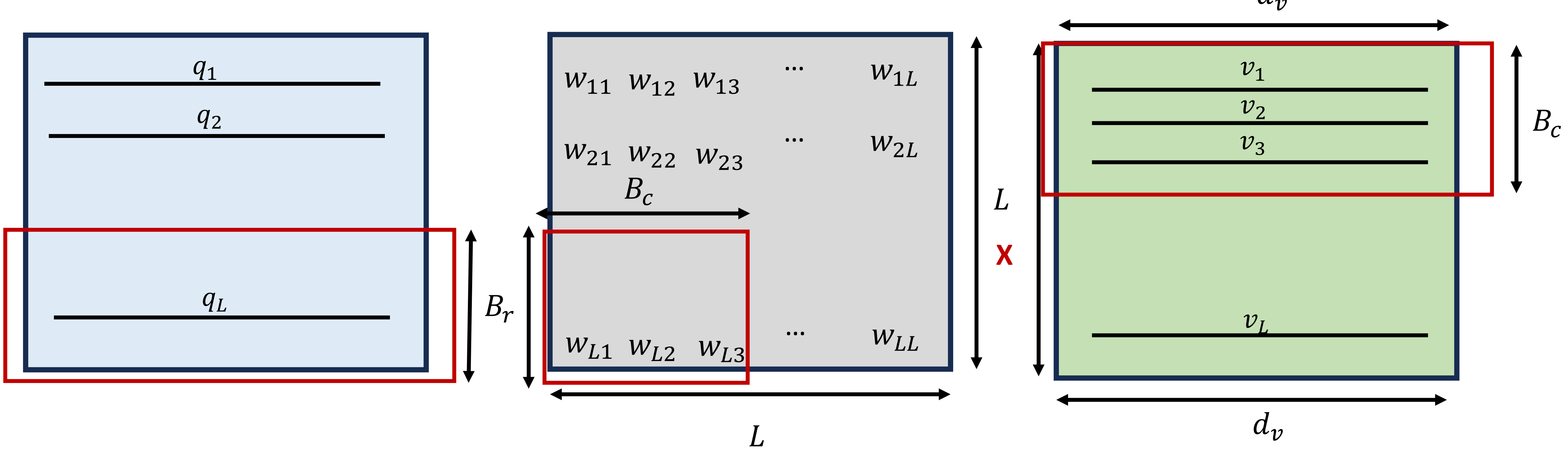
Memory Needed: $B_r \times B_c + d_k B_r + d_k B_c + d_v B_c + d_v B_r$

FlashAttention

Ignore softmax for now

How to produce the full output for B_r rows of O? Iterate over columns of K

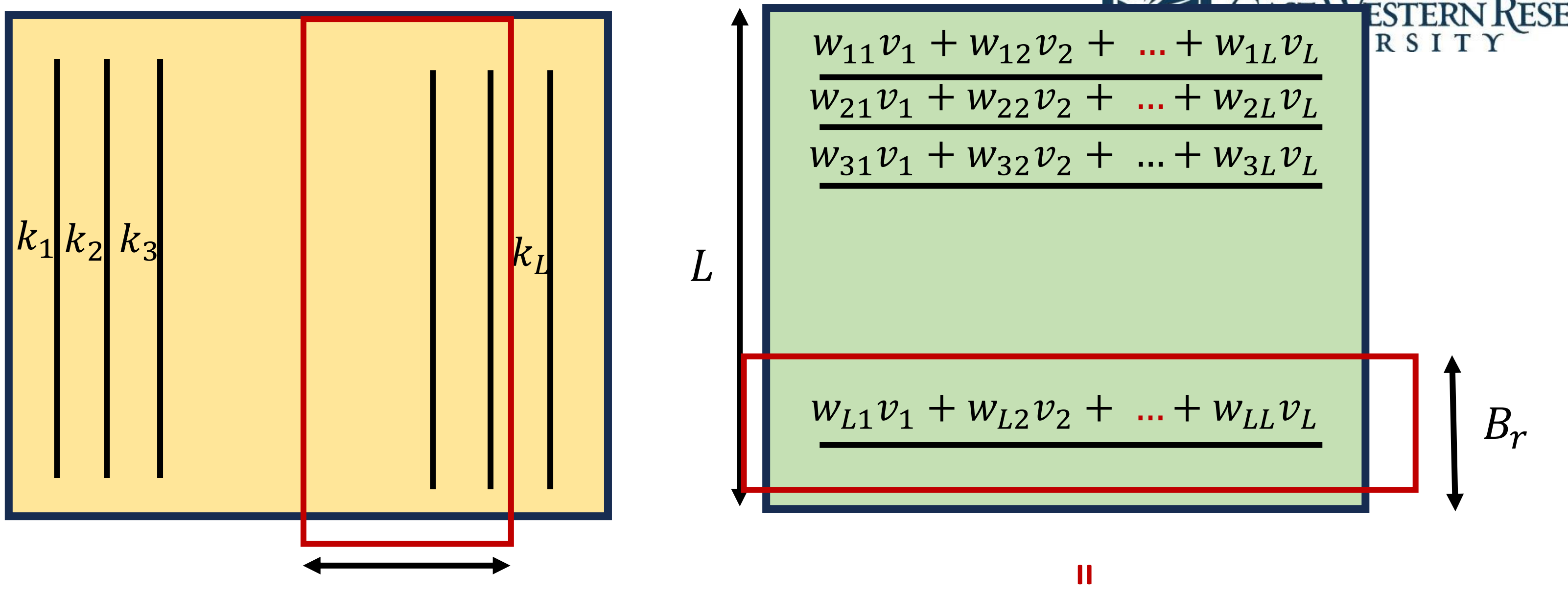


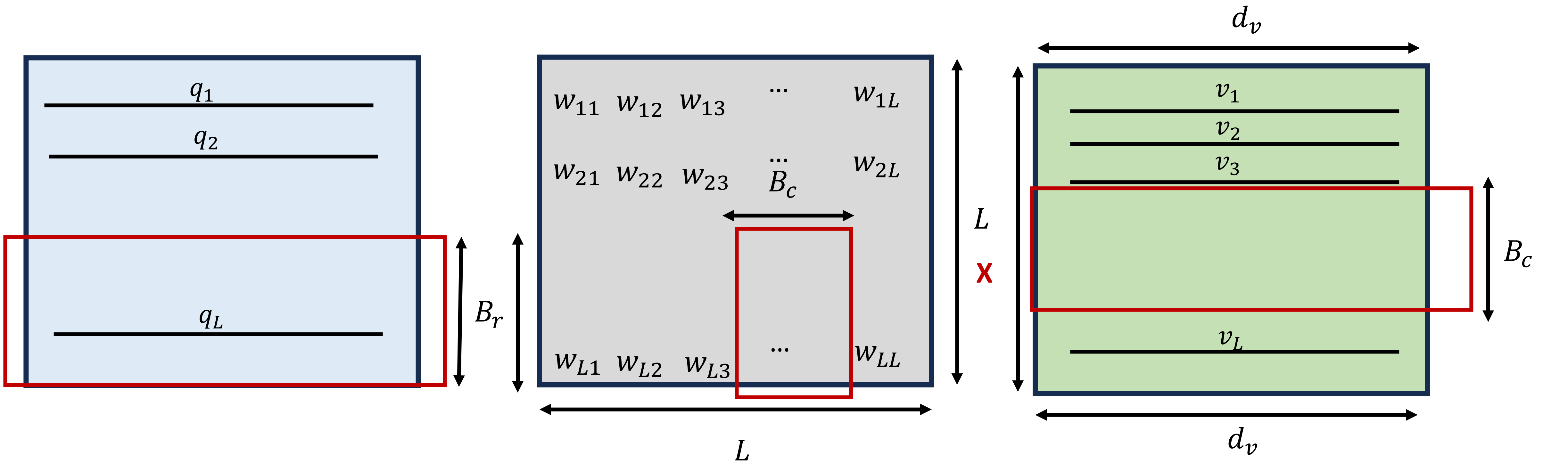


FlashAttention

Ignore softmax for now

How to produce the full output for B_r rows of O? Iterate over columns of K

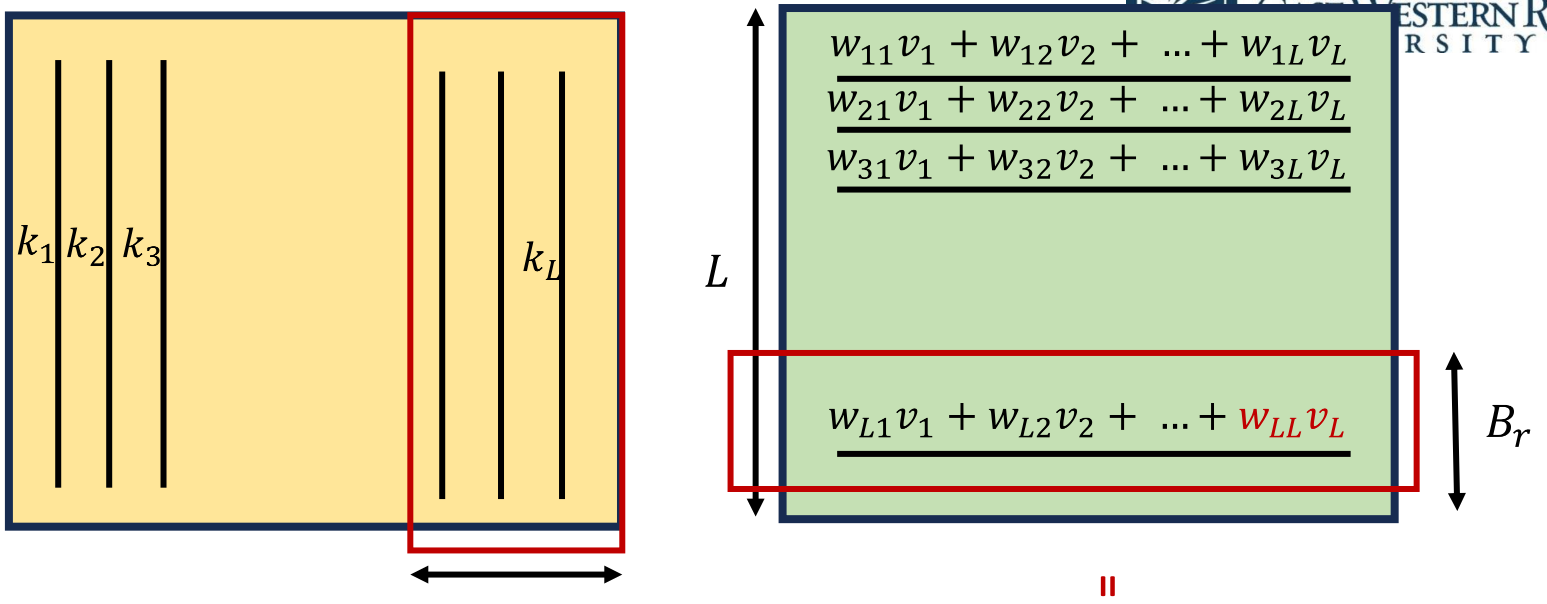


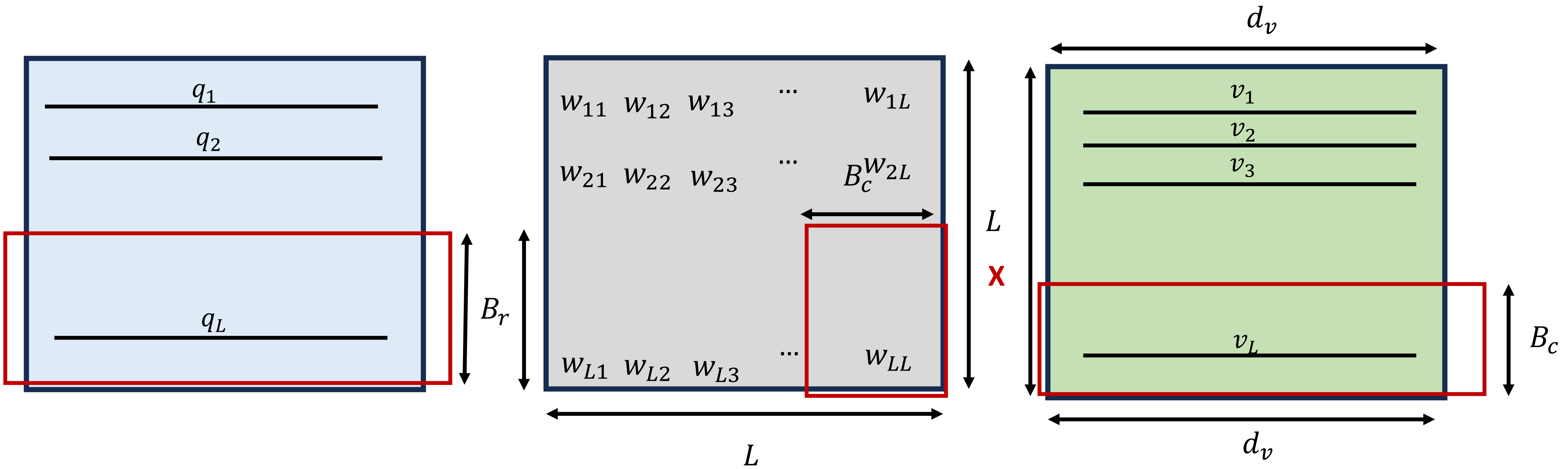


FlashAttention

Ignore softmax for now

How to produce the full output for B_r rows of O? Iterate over columns of K





Flash Attention

- Fuse and tile the three Key Operations
  - Operation #1: $Y = QK^T$: Product of Q and K^T
  - Operation #2: $Z = Softmax(Y)$
  - Operation #3: $O = ZV$: Product of Z and V matrices
  - Q, K^T, V, Z: Dense matrices

- Notice how you are taking a tile of Q, K, V and producing a partial sum for a tile of the output O

- This is also known as Fused attention

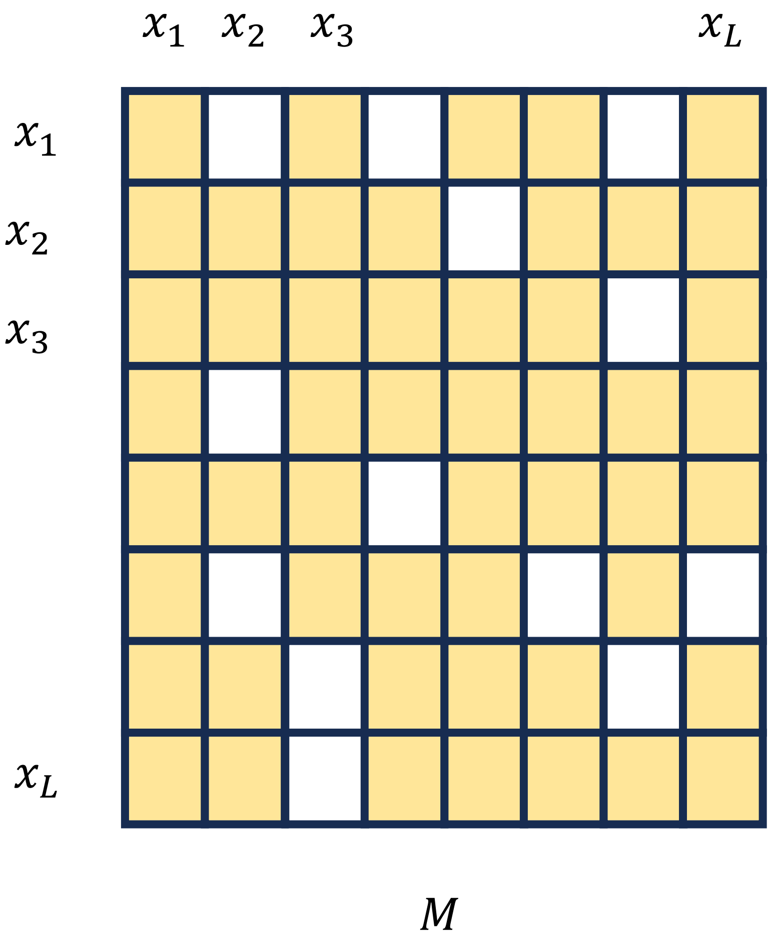
Attention with Sparse Attention Mask

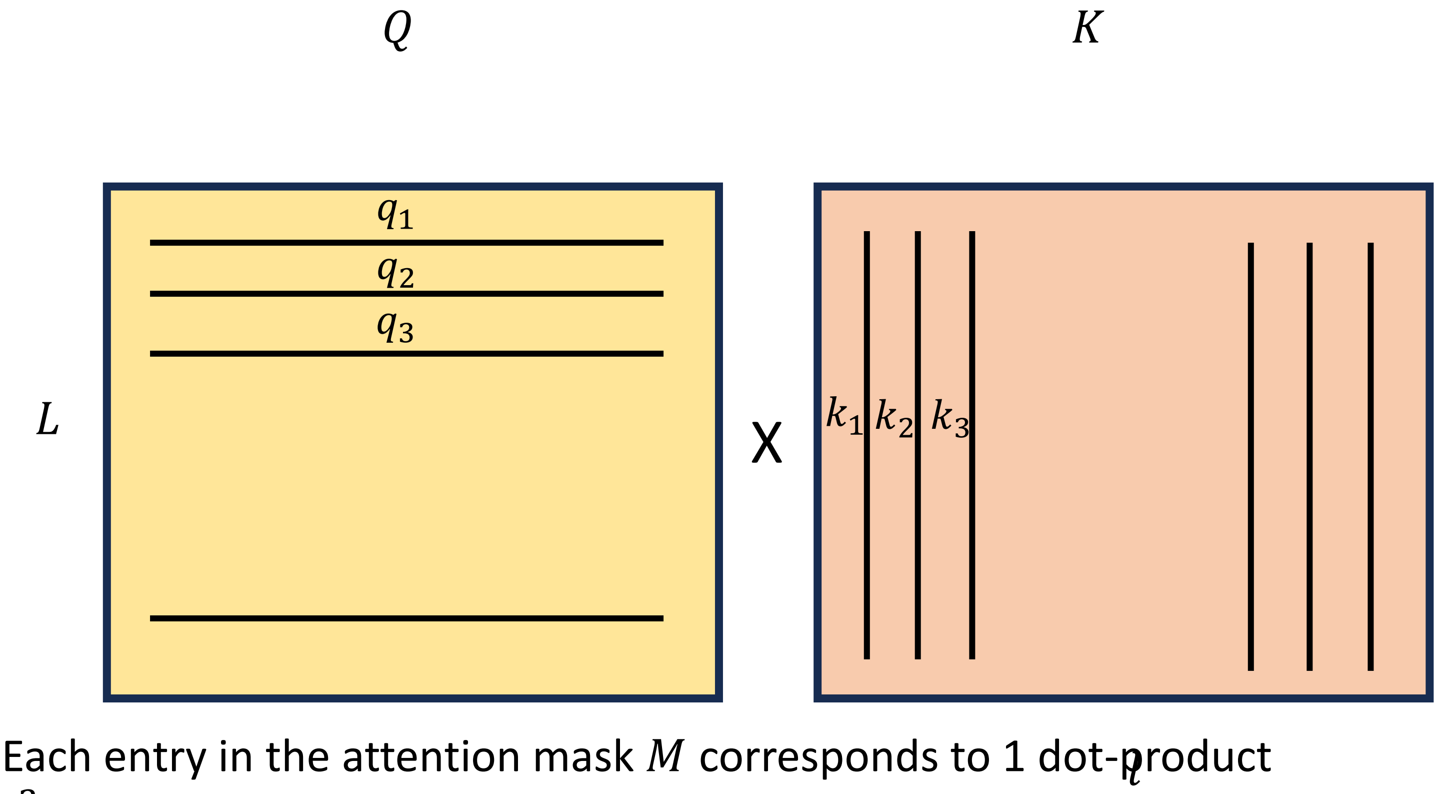
- Three Key Operations

- Operation #1: $Y = \ QK^T |\ M$: Product of Q and K^T matrices under mask M
- Operation #2: $Z = Softmax(Y)$
- Operation #3: $O = ZV$: Product of Z and V matrices

- Q, K^T, V: Dense matrices
- Z: Sparse matrix

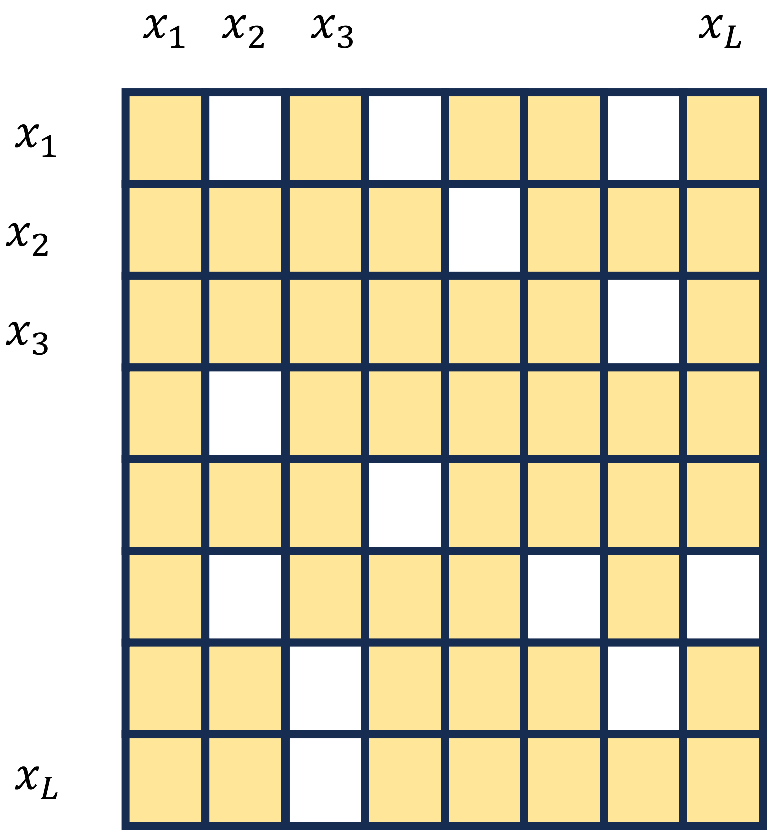
QK Product with Attention Mask





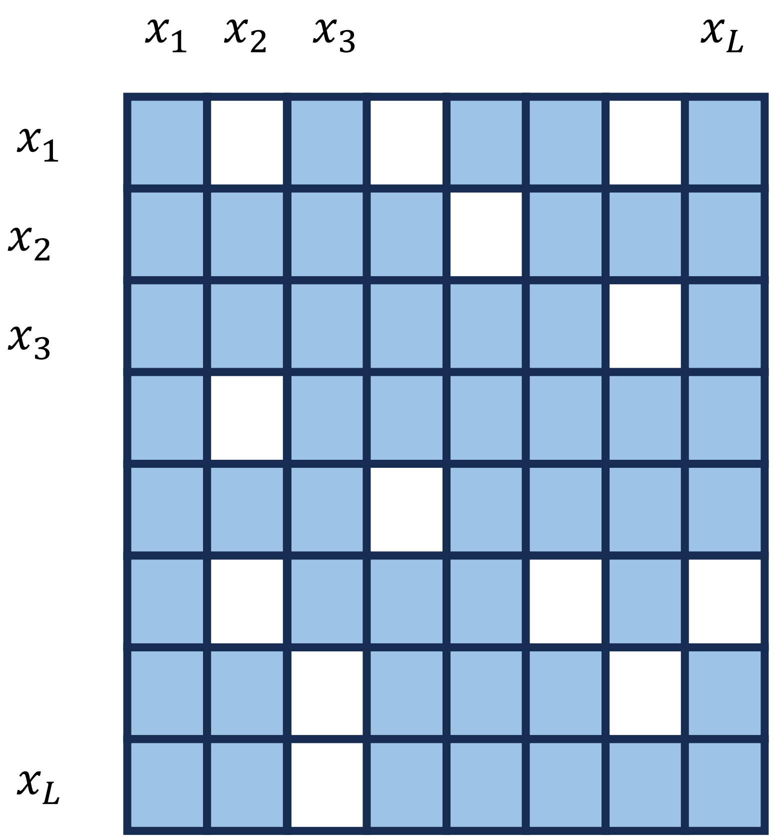
Each entry in the attention mask M corresponds to 1 dot-product
l^2 entries in total
Sparsity factor s determines how many dot-products actually need to be computed

QK Product with Attention Mask



M

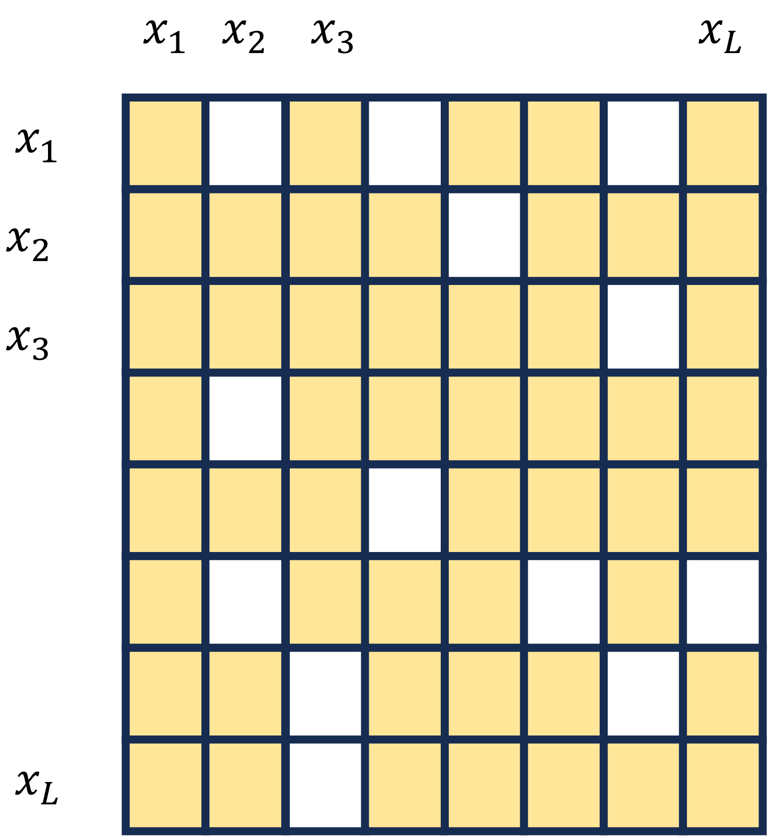
$$Y = QK^T$$



Product will lead to a l^2 sized matrix with 0 and non-zero elements similar to mask: How? Will ask in WA 3.

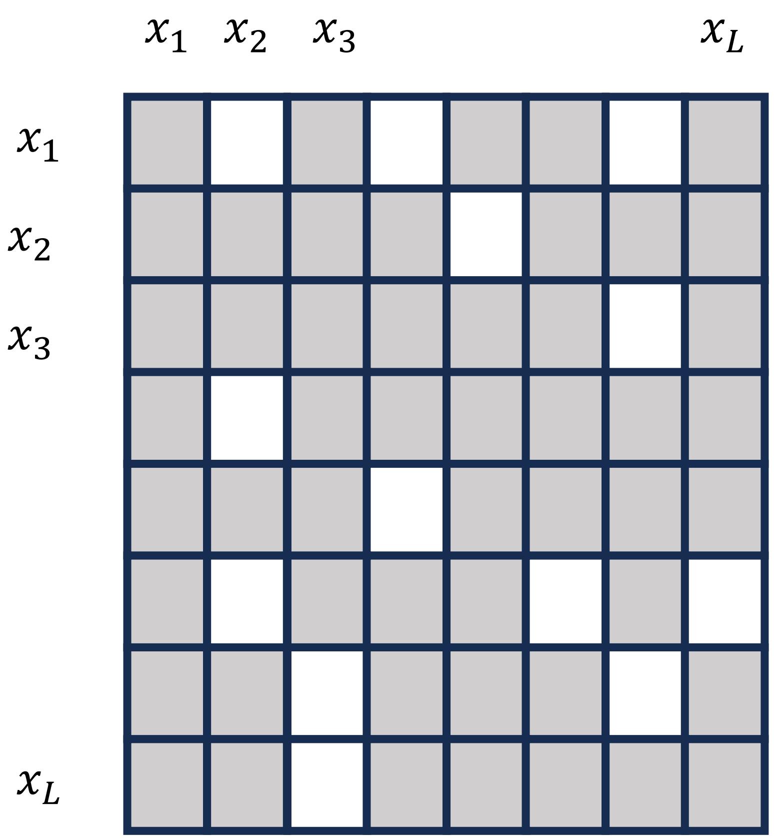
In practice, elements corresponding to 0 mask are set to $-\infty$: Why? Will ask in WA 3

Softmax with Attention Mask



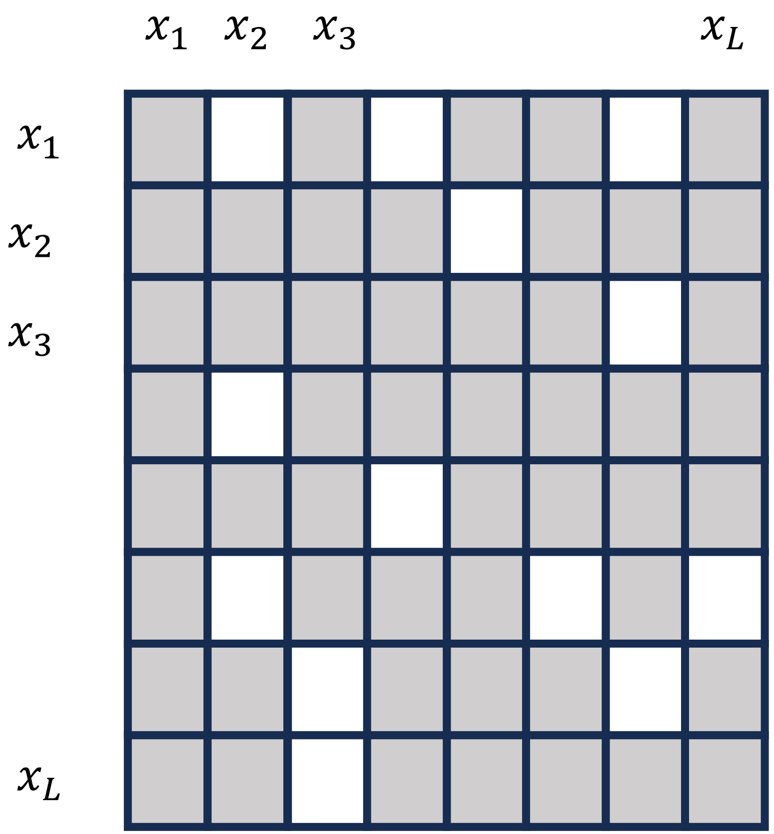
M

$Z = softmax(QK^T)$

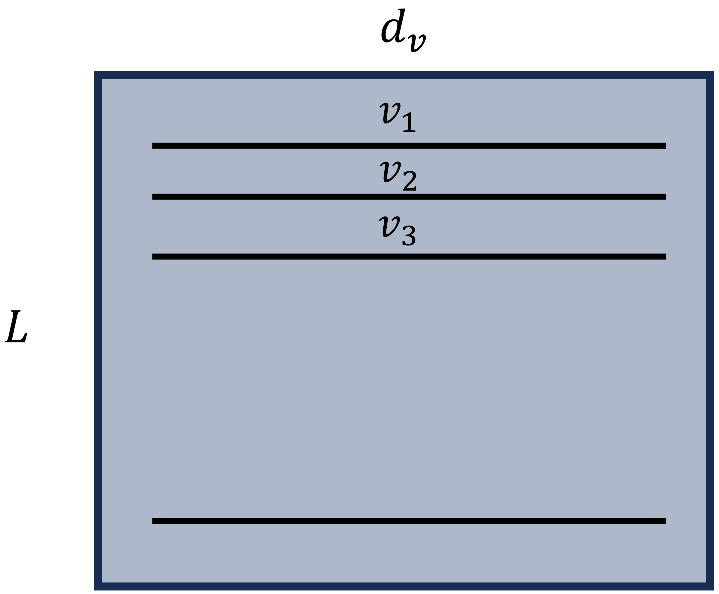


Softmax will replace all $-\infty$ elements with 0: How? Will ask in WA 3.
The pattern of 0 and non-zero still remains the same

Product with V Matrix



$Z = softmax(QK^T)$



V

Product of a sparse matrix ($Z = softmax(QK^T)$) and dense V matrix

Attention with Sparse Attention Mask

- Three Key Operations

- Operation #1: $Y = QK^T | M$: Product of Q and K^T matrices under mask M
- Operation #2: $Z = Softmax(Y)$
- Operation #3: $O = ZV$: Product of Z and V matrices

- Q, K^T, V: Dense matrices
- Z: Sparse matrix

Attention with Sparse Attention Mask

- Three Key Operations

- Operation #1: $Y = \; QK^T |\, M$: **Product of two dense matrices under a mask**
- Operation #2: $Z = Softmax(Y)$
- Operation #3: $O = ZV$: Product of Z and V matrices **Product of a sparse and dense matrix**

- Q, K^T, V: Dense matrices
- Z: Sparse matrix

Attention with Sparse Attention Mask

- Three Key Operations

- Operation #1: $Y = \ QK^T|\ M$: **Product of two dense matrices under a mask**
- Operation #2: $Z = Softmax(Y)$
- Operation #3: $O = ZV$: Product of Z and V matrices **Product of a sparse and dense matrix**

- Q, K^T, V: Dense matrices
- Z: Sparse matrix

$Y = QK^T | M$: Product of two dense matrices under a mask

- Known as Sampled Dense Dense Matrix Multiplication
- Just a handful of papers on accelerating this kernel – we will discuss in next class

Attention with Sparse Attention Mask

- Three Key Operations

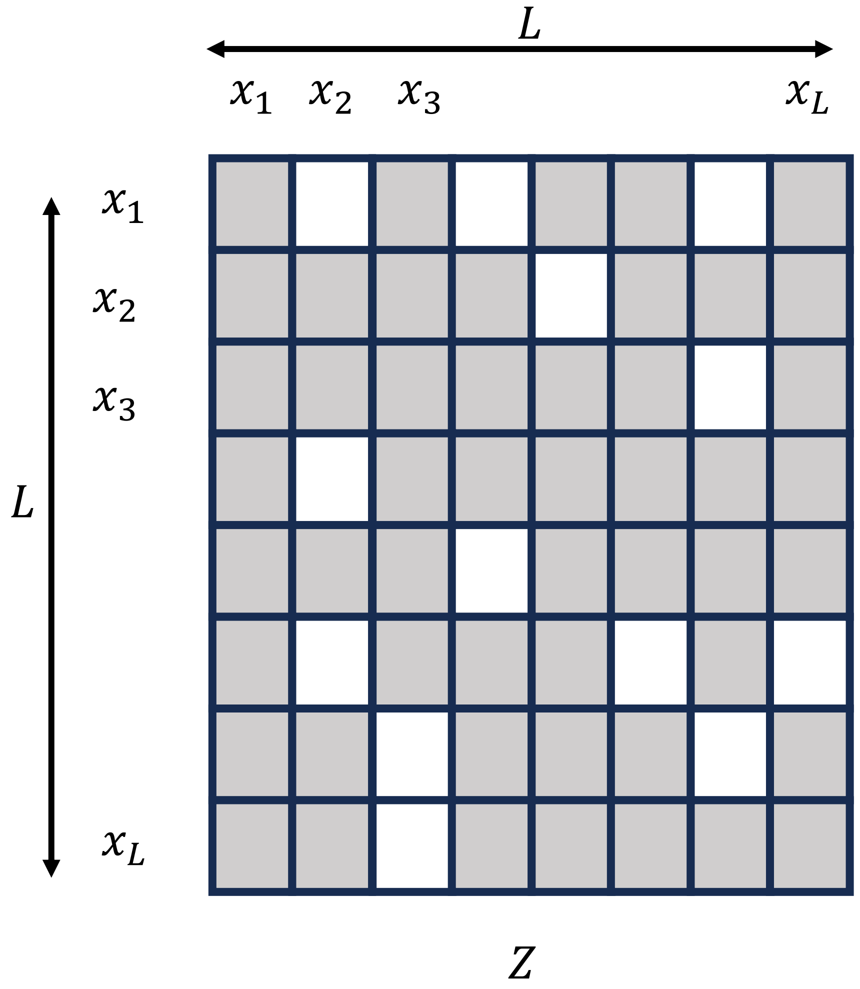
- Operation #1: $Y = QK^T | M$: **Product of dense matrices under a sparse mask**
- Operation #2: $Z = Softmax(Y)$
- Operation #3: $O = ZV$: Product of Z and V matrices: **Product of a sparse and dense matrix**

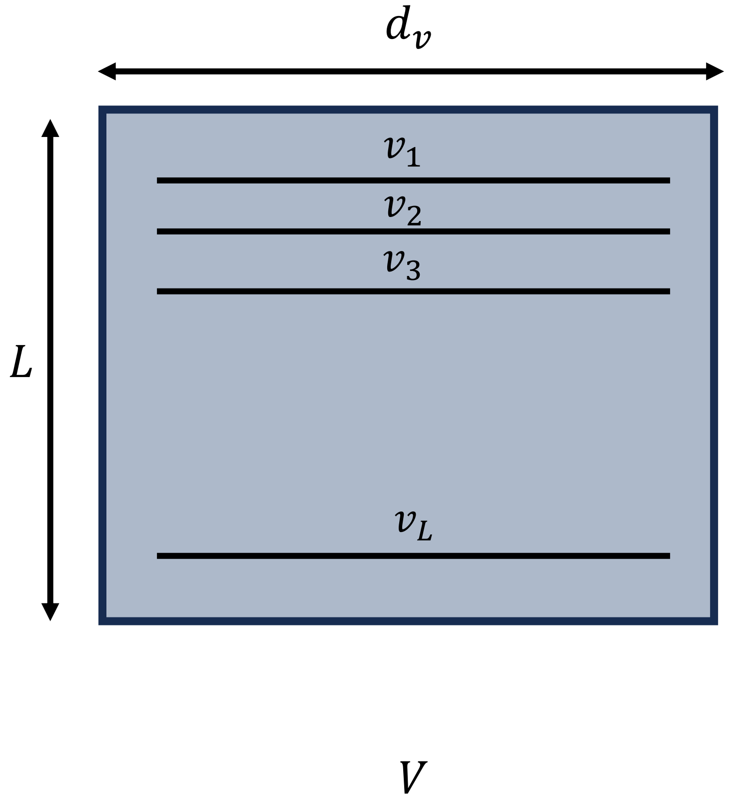
- Q, K^T, V: Dense matrices
- Z: Sparse matrix

Sparse-Dense Matrix Multiplication (SpMM)

- An important kernel in scientific computing
- In machine learning, an important kernel in Graph Neural Networks (GNNs), pruned CNNs, and Sparse Transformers
- Extensive Research has been performed (and still continuing) on accelerating SpMM

Sparse Dense Matrix Multiplication (SpMM)

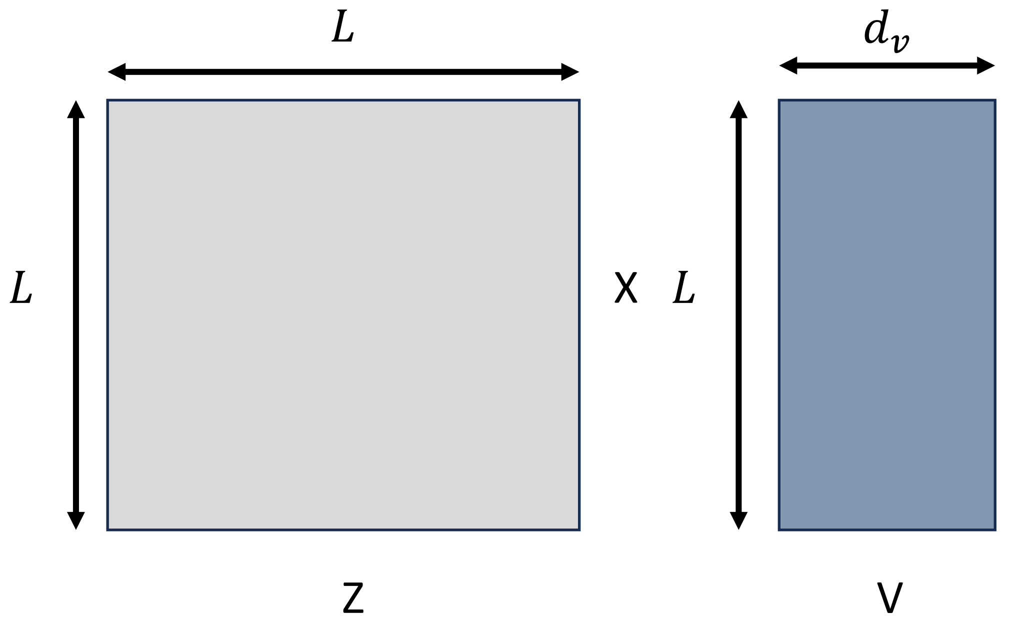




Product of a sparse matrix ($Z = softmax(QK^T)$) and dense V matrix

Sparse Dense Matrix Multiplication (SpMM)

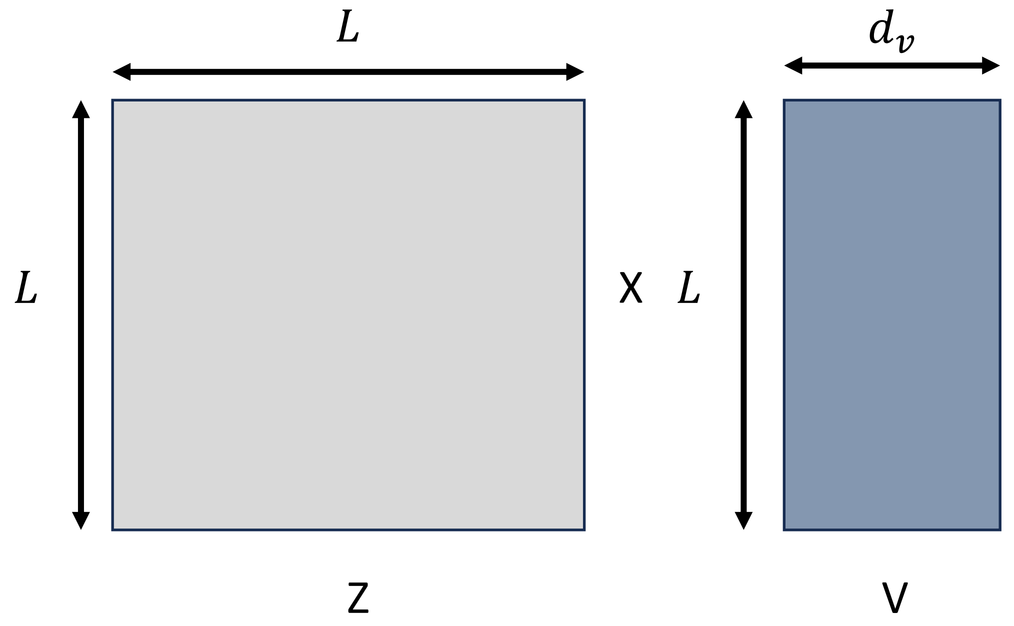
- Let nnz be the number of non-zero elements in Z
- Assume $nnz = L$, what is the time complexity of performing dense dense matrix multiplication on these matrices?



Sparse Dense Matrix Multiplication (SpMM)

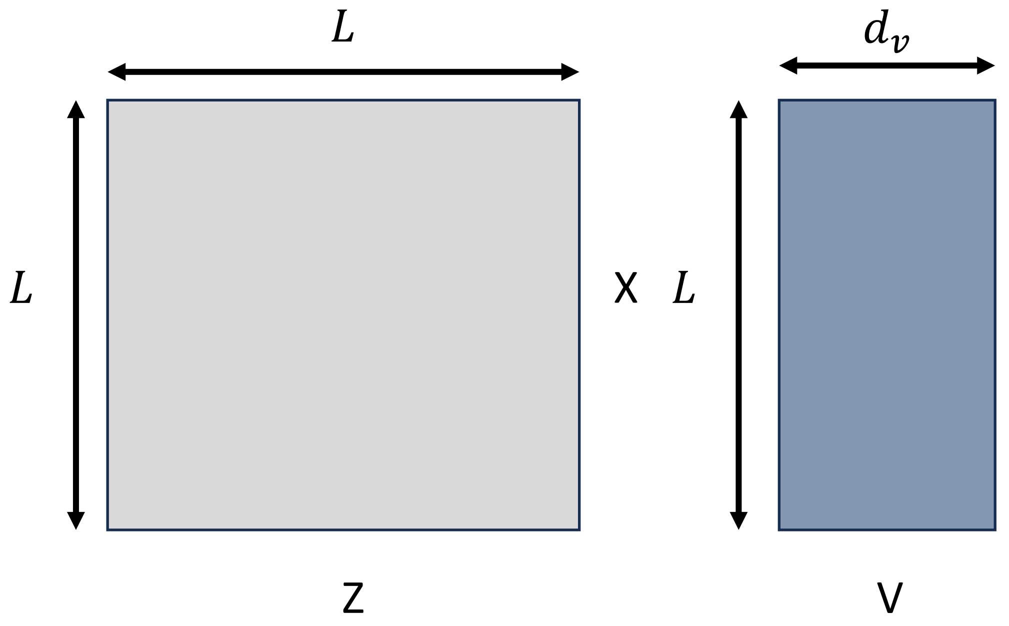
- Let nnz be the number of non-zero elements in Z

- Assume $nnz = L, : O(L^2 d_v)$



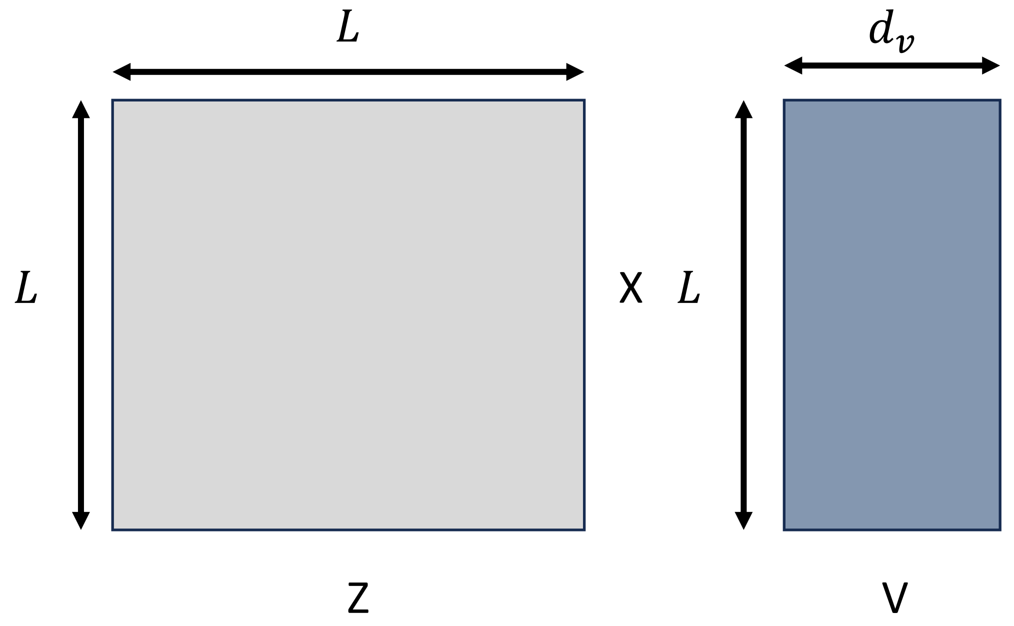
Sparse Dense Matrix Multiplication (SpMM)

- Assume $nnz = L$
- What will be the time complexity of the best sequential algorithm?



Sparse Dense Matrix Multiplication (SpMM)

- Assume $nnz = L$
- What will be the time complexity of the best sequential algorithm?
- Sequential Algorithm: Iterate through non-zeros of Z and scale and add the rows of the second matrix V



Sparse Dense Matrix Multiplication (SpMM)

$$\begin{bmatrix} 0 & 2 & 0 \\ 1 & 2 & 0 \\ 0 & 3 & 5 \end{bmatrix} \times \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \\ c_1 & c_2 \end{bmatrix} = ??$$

Sparse Dense Matrix Multiplication (SpMM)

$$
\begin{bmatrix} 0 & 2 & 0 \\ 1 & 2 & 0 \\ 0 & 3 & 5 \end{bmatrix} \times \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \\ c_1 & c_2 \end{bmatrix} = \begin{bmatrix} 2b_1 & 2b_2 \\ & \\ & \end{bmatrix}
$$

Sparse Dense Matrix Multiplication (SpMM)

$$\begin{bmatrix} 0 & 2 & 0 \\ 1 & 2 & 0 \\ 0 & 3 & 5 \end{bmatrix} \times \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \\ c_1 & c_2 \end{bmatrix} = \begin{bmatrix} 2b_1 & 2b_2 \\ a_1 + 2b_1 & a_2 + 2b_2 \\ & \end{bmatrix}$$

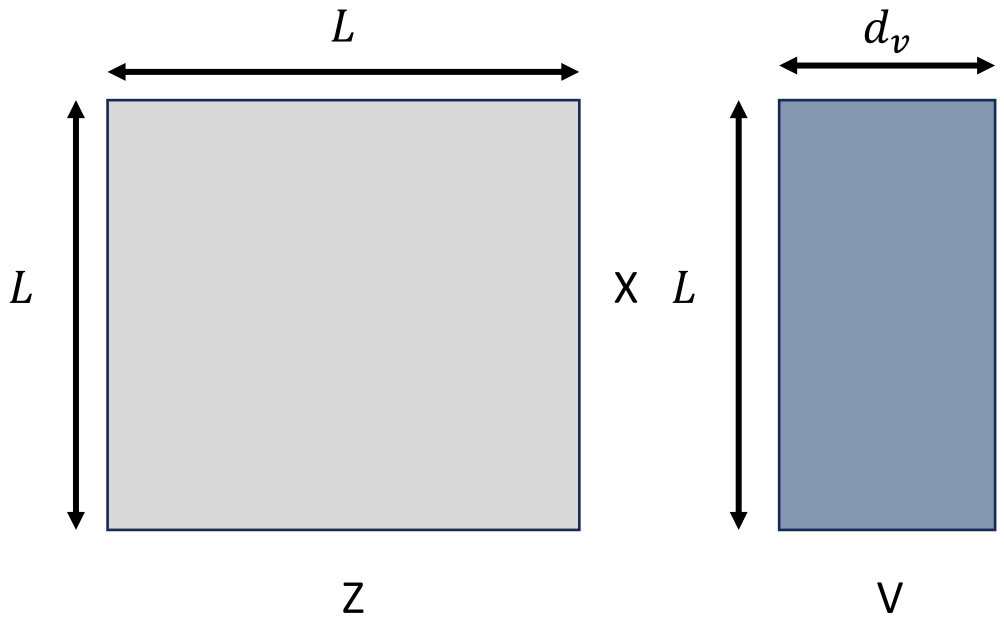
Sparse Dense Matrix Multiplication (SpMM)

$$\begin{bmatrix} 0 & 2 & 0 \\ 1 & 2 & 0 \\ 0 & 3 & 5 \end{bmatrix} \times \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \\ c_1 & c_2 \end{bmatrix} = \begin{bmatrix} 2b_1 & 2b_2 \\ a_1 + 2b_1 & a_2 + 2b_2 \\ 3b_1 + 5c_1 & 3b_2 + 5c_2 \end{bmatrix}$$

Sparse Dense Matrix Multiplication (SpMM)

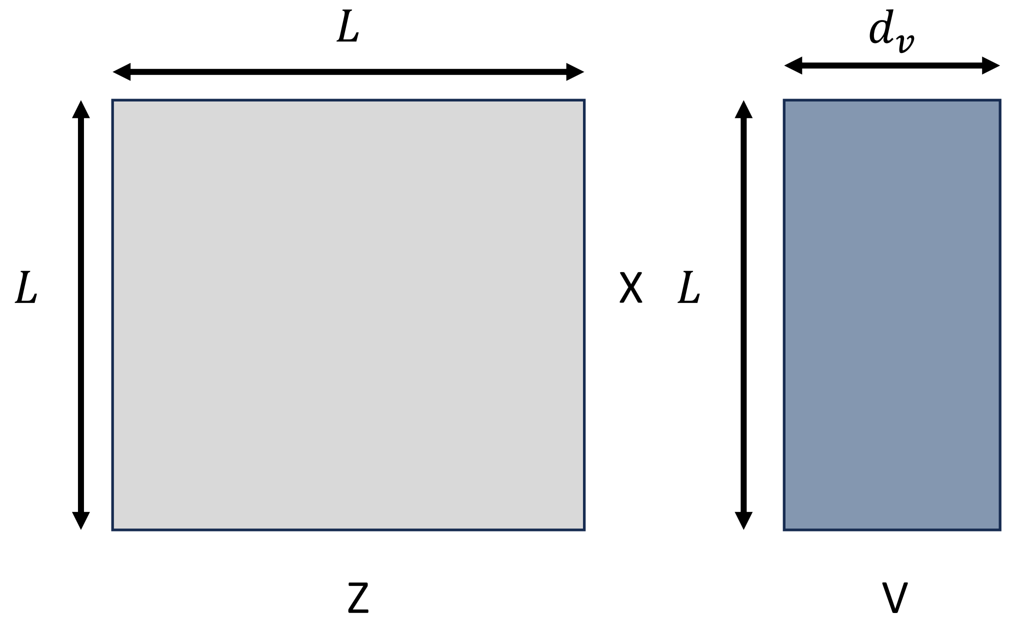
- Assume $nnz = L$

- What will be the time complexity of the best sequential algorithm? $O(Ld_v)$



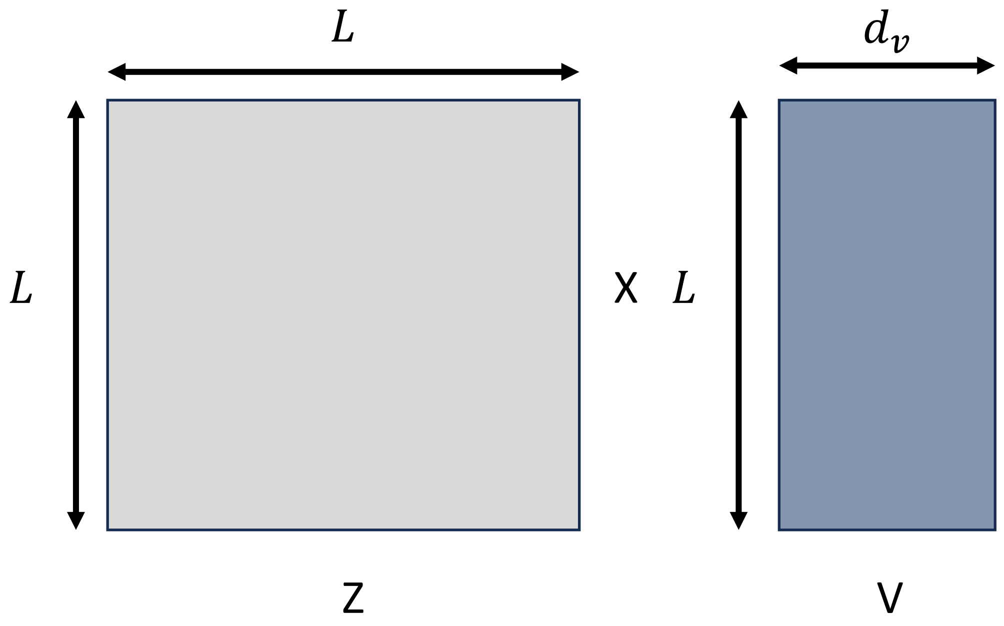
Sparse Dense Matrix Multiplication (SpMM)

- What will be the time complexity of the best sequential algorithm? $O(Ld_v)$
- If we use dense matrix multiplication, is that work optimal?



Sparse Dense Matrix Multiplication (SpMM)

- What will be the time complexity of the best sequential algorithm? $O(Ld_v)$
- If we use dense matrix multiplication, is that work optimal? No. $O(Ld_v) \ll O(L^2 d_v)$



Sparse Dense Matrix Multiplication (SpMM)

- Let nnz be the number of non-zero elements in Z
- Key Challenge: Cannot use dense-dense matrix multiplication techniques
  - Inefficient Storage
  - Non-useful computations
- Optimizations
  - Efficient Storage of Sparse Matrices
  - Work optimal task scheduling

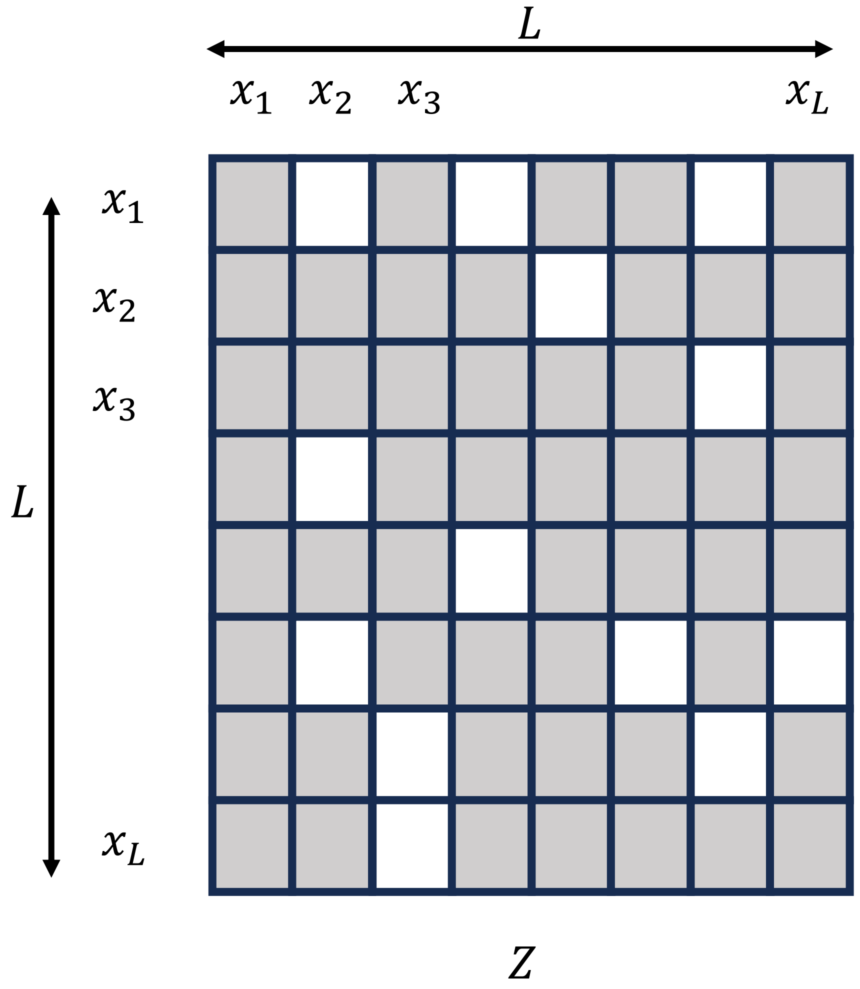
Sparse Dense Matrix Multiplication (SpMM)

- Let nnz be the number of non-zero elements in Z

- Key Challenge: Cannot use dense-dense matrix multiplication techniques
  - Inefficient Storage
  - Non-useful computations

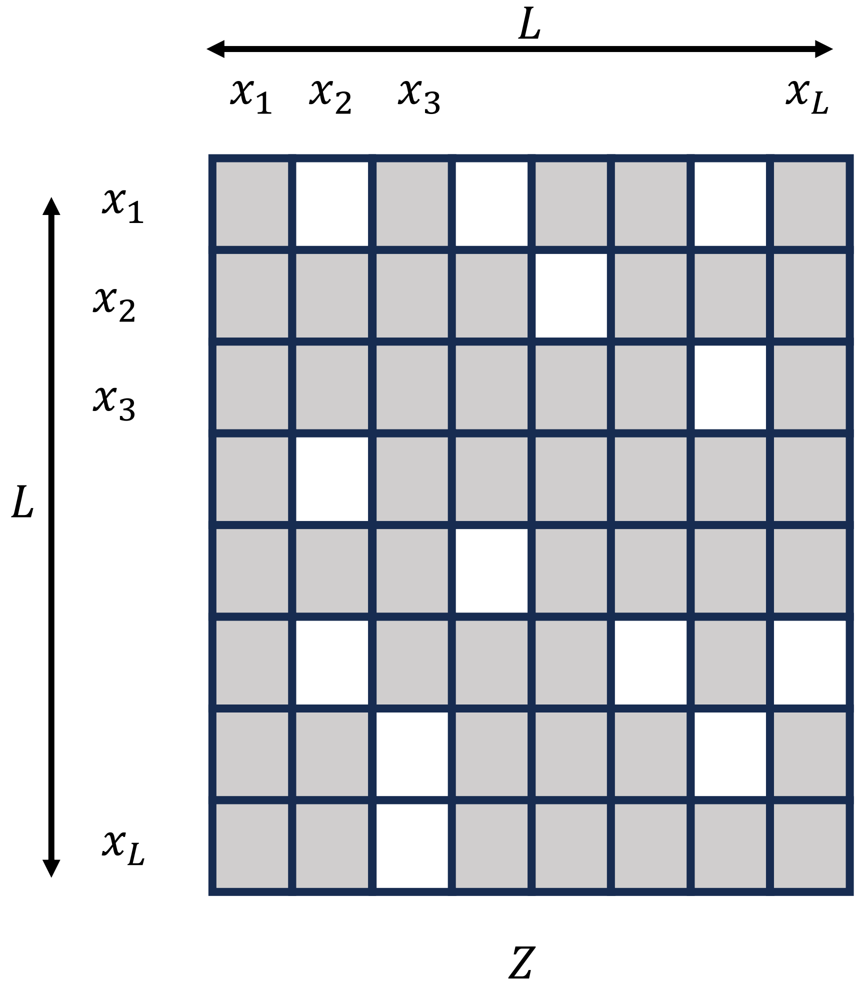
- Optimizations
  - **Efficient Storage of Sparse Matrices**
  - Work optimal task scheduling

Sparse Dense Matrix Multiplication (SpMM)



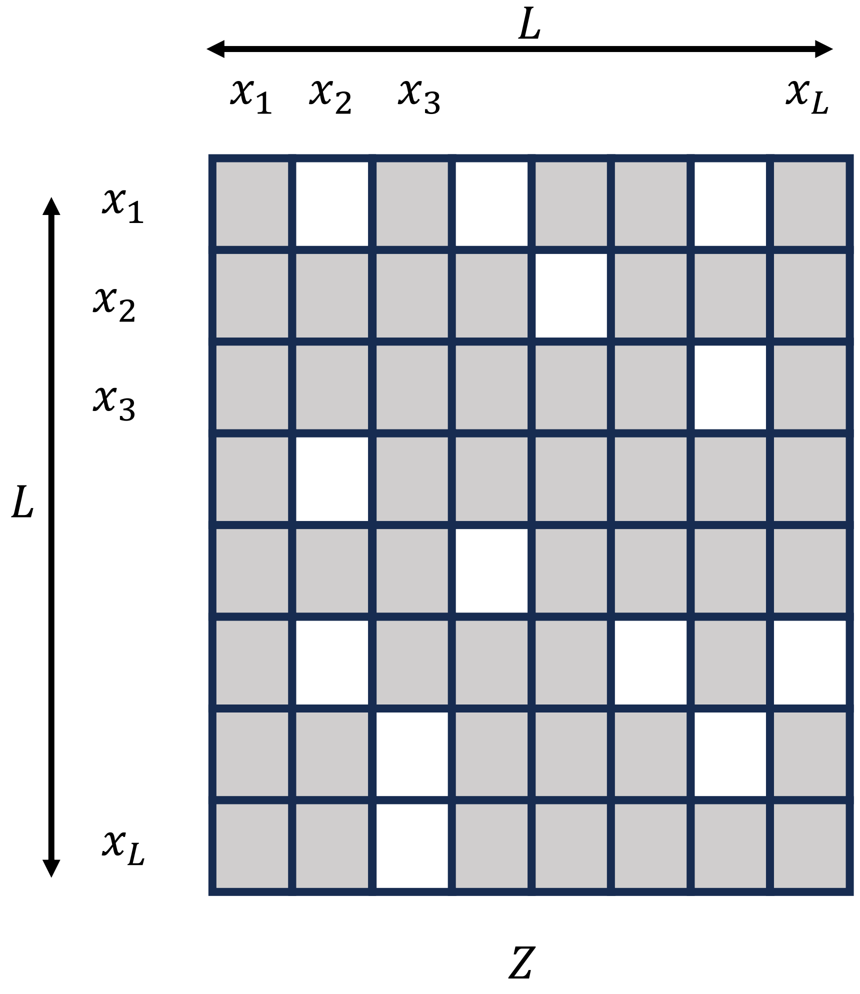
- Let nnz be the number of non-zero elements in Z
- What is the total storage if Z is stored as a dense matrix?

Sparse Dense Matrix Multiplication (SpMM)



- Let nnz be the number of non-zero elements in Z
- What is the total storage if Z is stored as a dense matrix? $L \times L$

Sparse Dense Matrix Multiplication (SpMM)

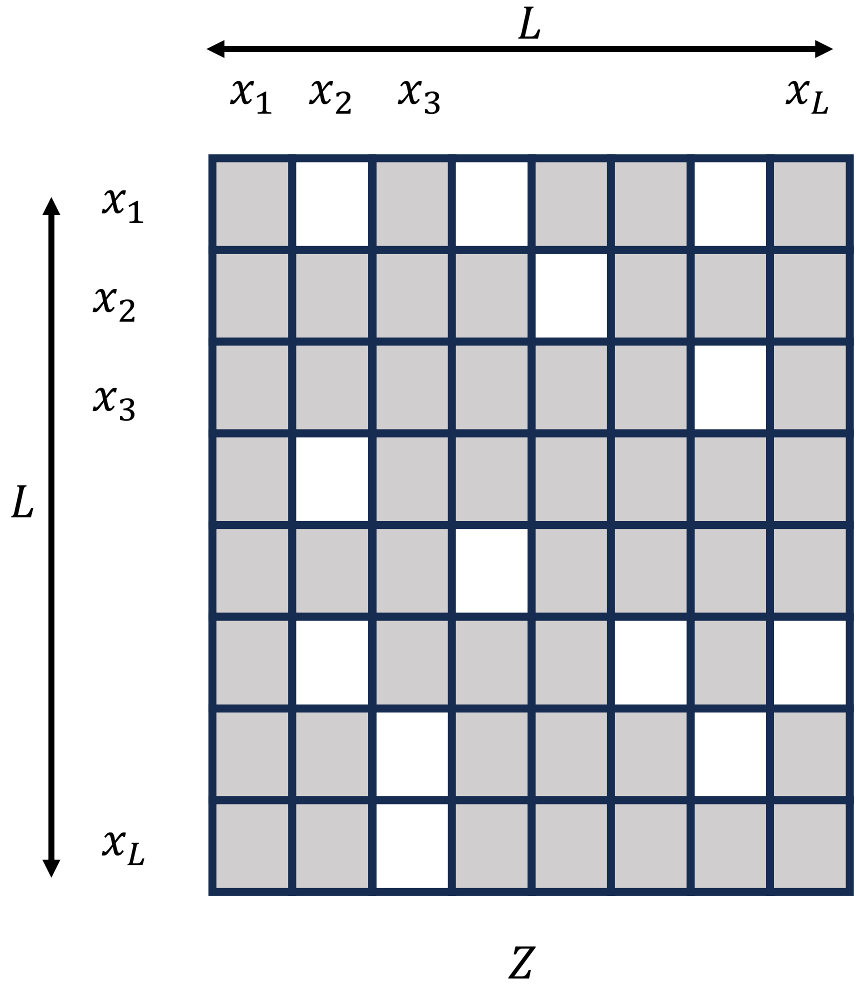


- Let nnz be the number of non-zero elements in Z

- What is the total storage if Z is stored as a dense matrix? $L \times L$

- if $\mathrm{nnz} \leq 0.1 \times (L \times L)$

Sparse Dense Matrix Multiplication (SpMM)



- Let nnz be the number of non-zero elements in Z
- What is the total storage if Z is stored as a dense matrix? $L \times L$
- if $\mathrm{nnz} \leq 0.1 \times (L \times L)$
- 0.1: sparsity factor s
- **99% percent of storage can be optimized**

Sparse Dense Matrix Multiplication (SpMM)

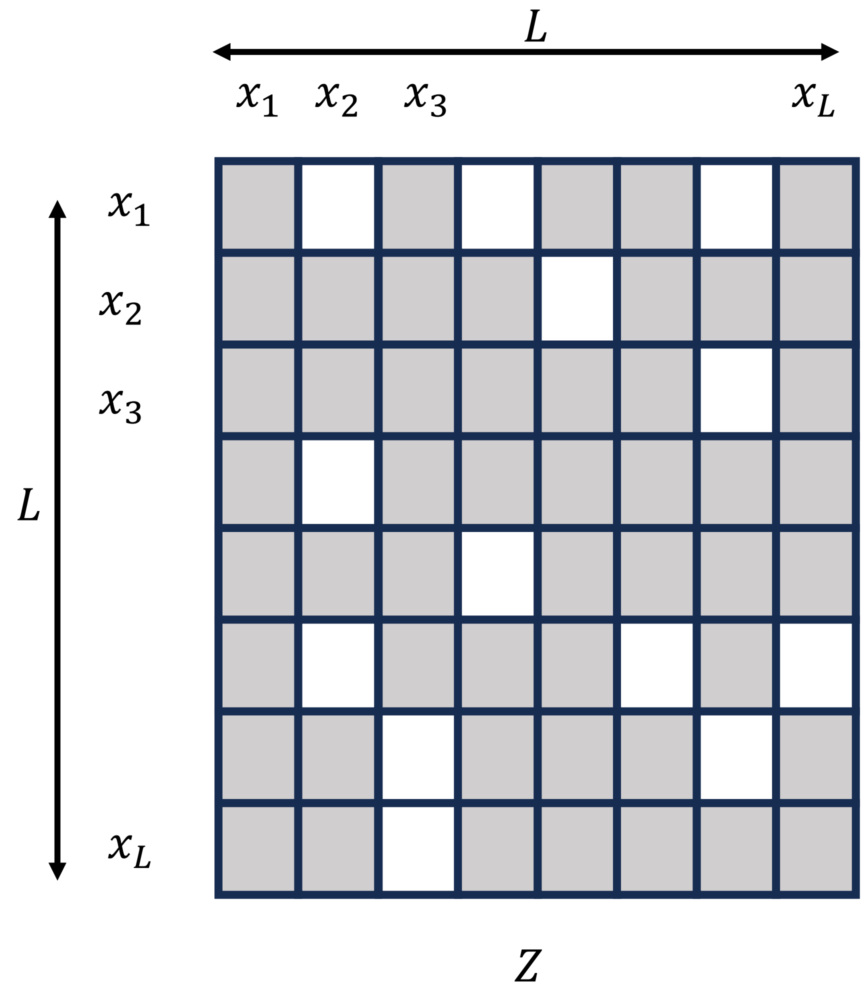
- Several formats have been proposed with different pros and cons
  - Suitable for different use cases

- https://docs.nvidia.com/cuda/cusparse/index.html#matrix-formats

- A few examples
  - Coordinate (COO)
  - Compressed Sparse Row/Column (CSR/CSC)
  - Sliced Ellpack
  - Block Sparse Row (BSR)

SpMM Storage Format - COO

- Coordinate format
- Each non-zero entry is assigned a coordinate
  - RowID,ColumnID
- Three arrays, each of size nnz are used to store the matrix
  - Row: Array of RowIDs
  - Column: Array of ColumnIDs
  - Value: Array of Values



SpMM Storage Format - COO

Dense Matrix

	0	1	2	3
0	1.0		2.0	
1		3.0		
2				
3	4.0	5.0		
4		6.0	7.0	8.0

Coordinate Format - COO (zero-base index)

	0	1	2	3	4	5	6	7
Row Indices	0	0	1	4	4	5	5	5
Column Indices	0	2	1	0	1	1	2	3
Values	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0

SpMM Storage Format - COO

- Storage requirements?

SpMM Storage Format - COO

- Storage requirements? $3nnz$

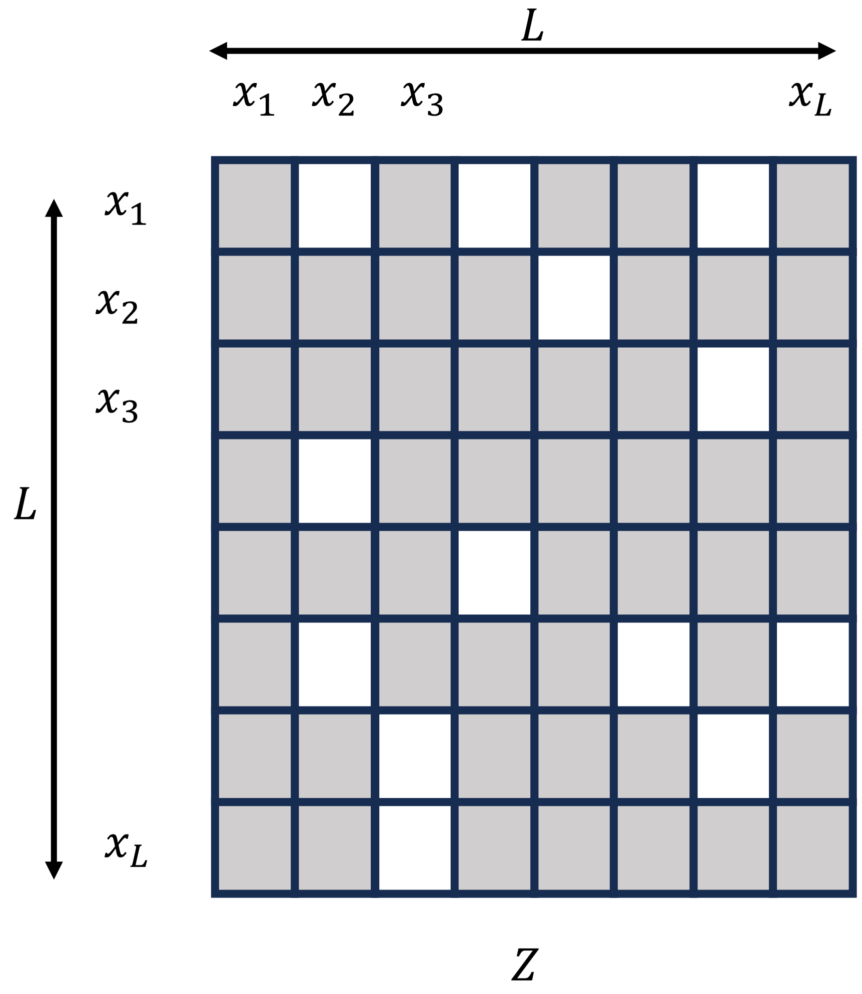
- Each array is of size nnz

SpMM Storage Format - CSR

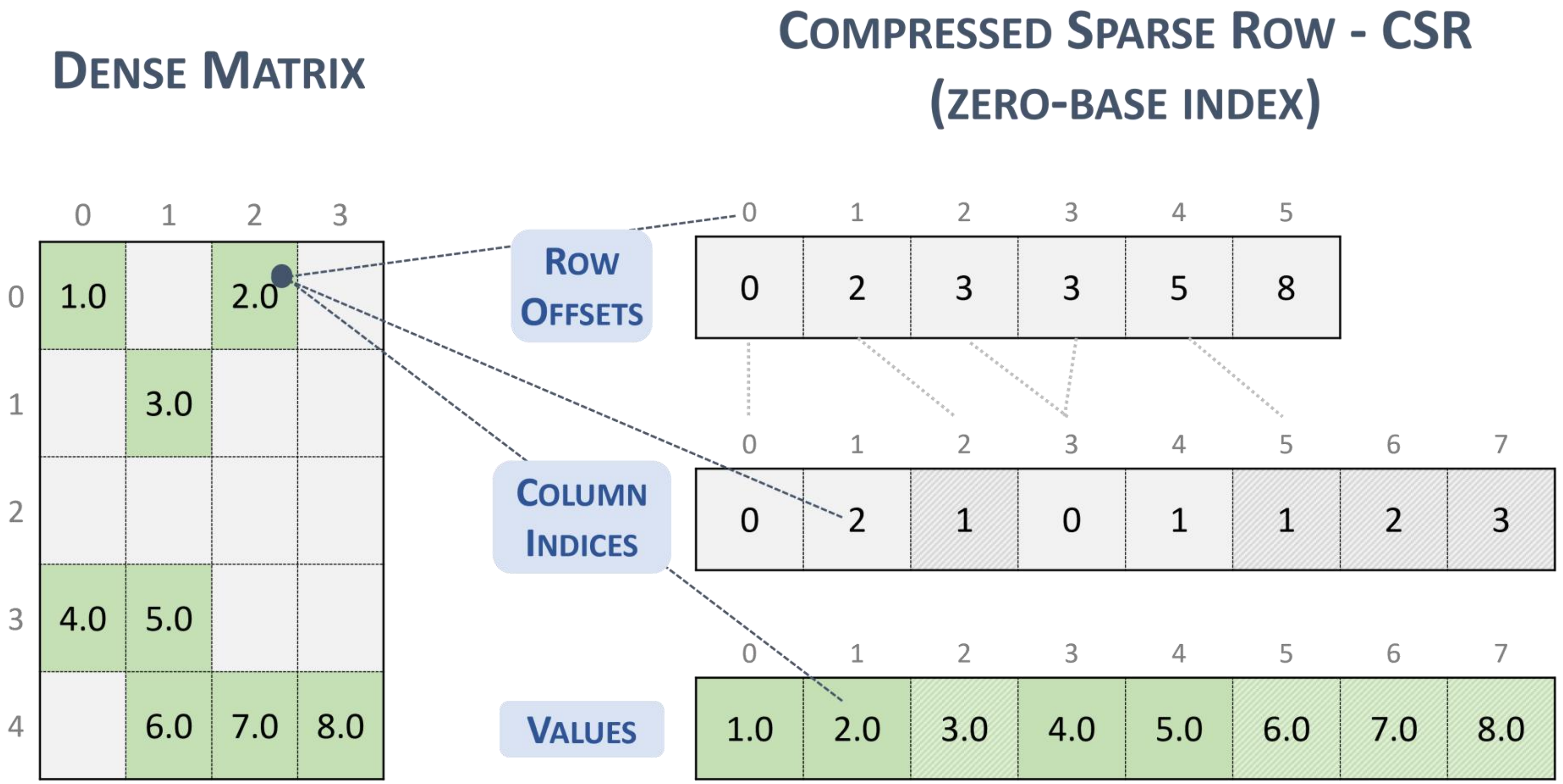
- Compressed Sparse Row Format

- Row array is compressed so that its entries point to offsets in column whenever new row starts

- Row: Array of offsets into Column – size: L (number of rows)
- Column: Array of ColumnIDs – size: nnz
- Value: Array of Values – size: nnz



SpMM Storage Format - CSR



SpMM Storage Format - CSR

- Storage requirements?

SpMM Storage Format - CSR

- Storage requirements? $2nnz + L$

- If $L < nnz$, more efficient representation than COO

- Let $L = 10{,}000$ and sparsity factor $s = 0.1$

- $nnz = 0.1 \times 10{,}000^2 = 10^7 \gg L$

SpMM Storage Format - CSC

- Similar to CSR, but row and column arrays are exchanged
- Row: Array of IDs for the nnz elements
- Column: Offset into the Row Array. Size: Number of columns in the matrix
- If number of columns << number of rows, a more efficient format than CSR

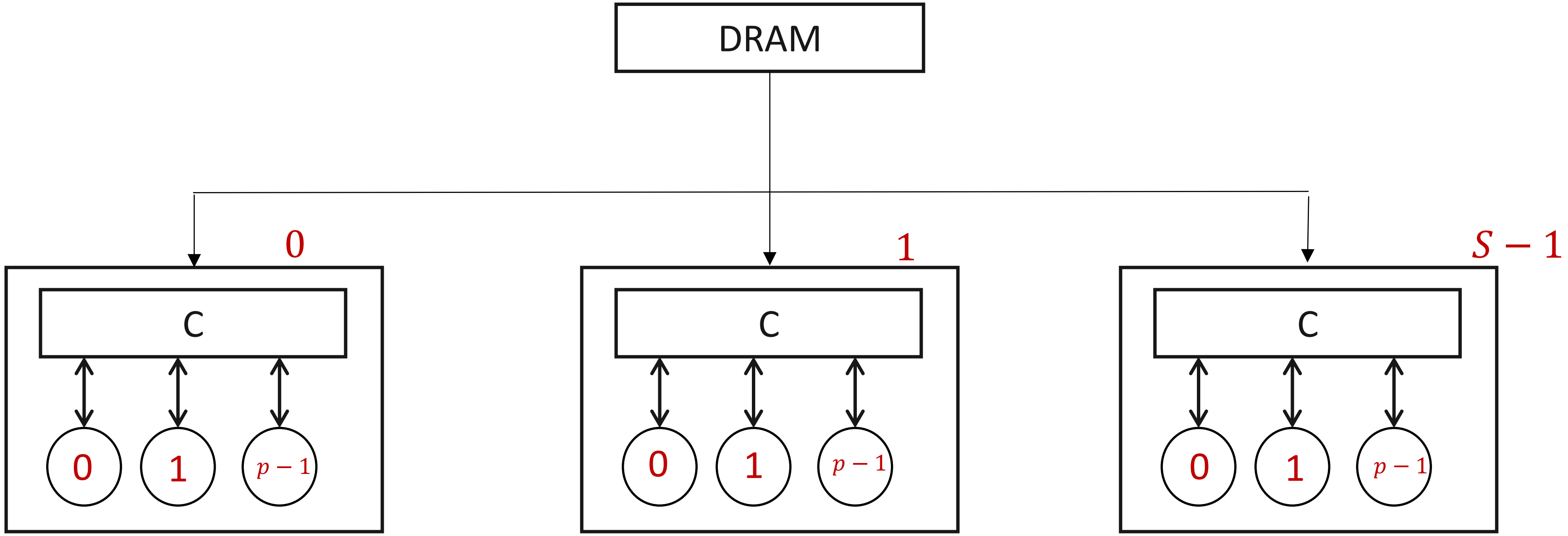
Sparse Dense Matrix Multiplication (SpMM)

- Let nnz be the number of non-zero elements in Z
- Key Challenge: Cannot use dense-dense matrix multiplication techniques
  - Inefficient Storage
  - Non-useful computations
- Optimizations
  - Efficient Storage of Sparse Matrices
  - **Work optimal task scheduling**
- Resource for the next few slides: https://people.eecs.berkeley.edu/~aydin/spmm_europar2018.pdf
  - Design Principles for Sparse Matrix Multiplication on the GPU

Sparse Dense Matrix Multiplication (SpMM)

- Parallelizing SpMM on GPUs
- Storage Format: CSR
- Recall: GPU Model
- How do we assign work to SMPs and Processors within SMPs?

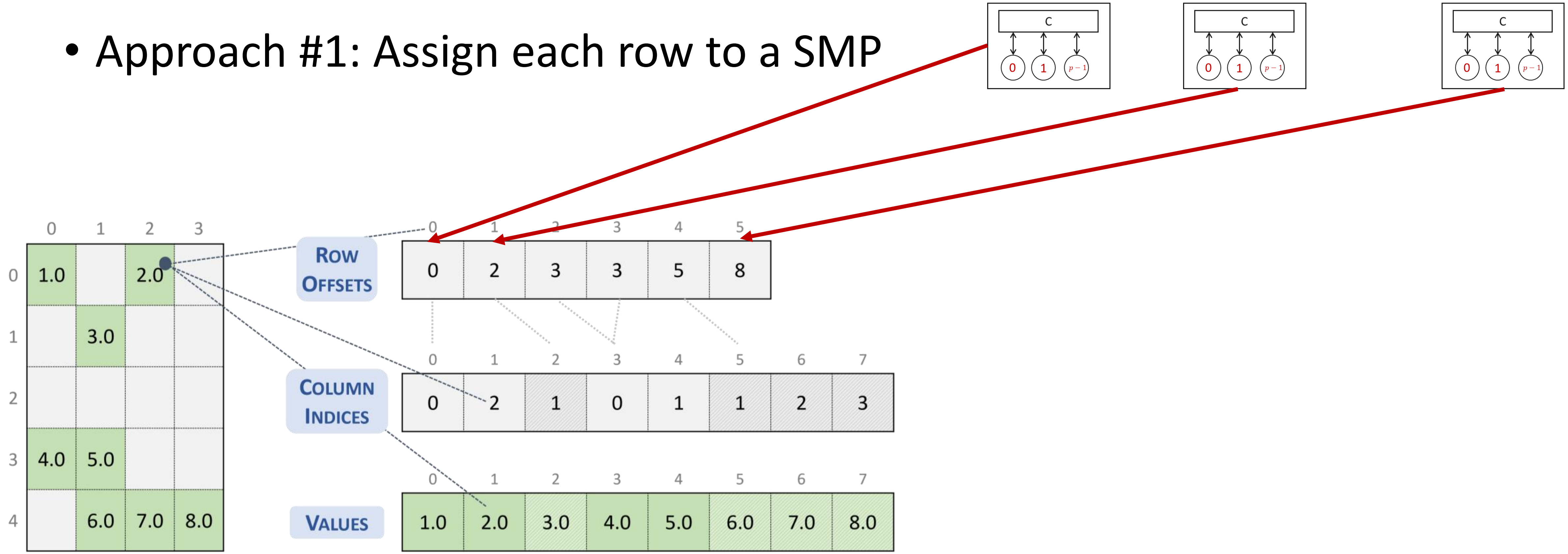
Recall: Modeling GPU Architectures



- S Blocks/SMPs
- p Threads/Processors per block

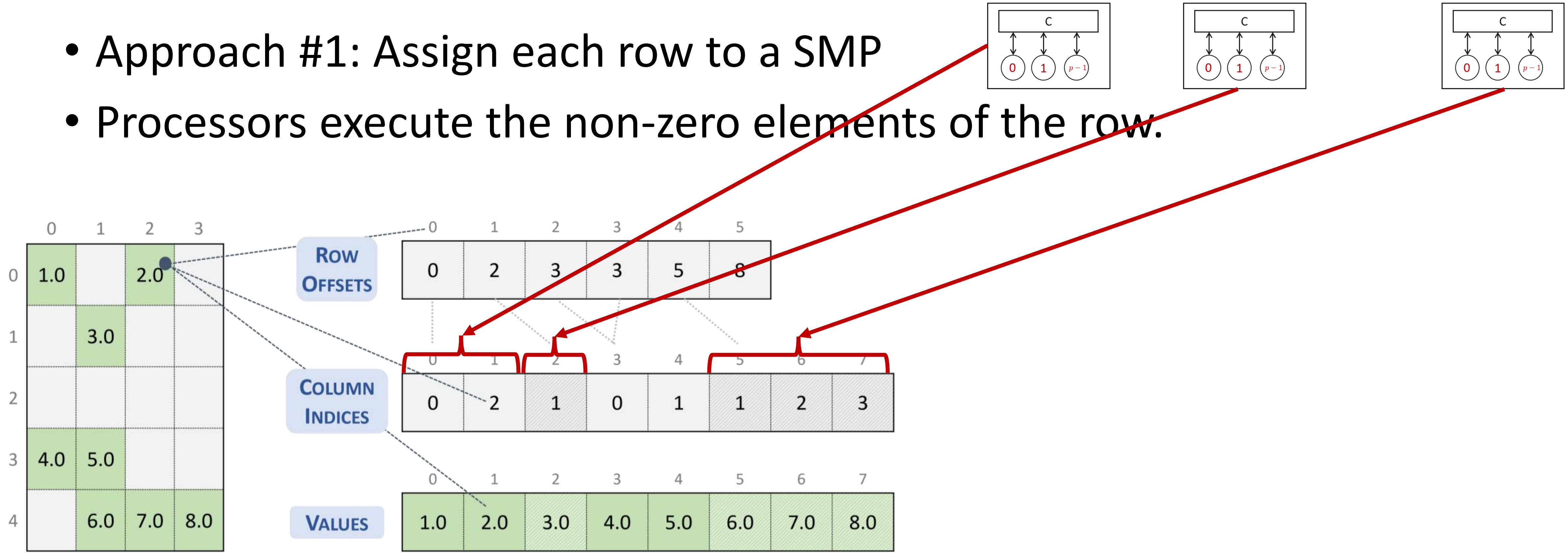
Sparse Dense Matrix Multiplication (SpMM)

- Approach #1: Assign each row to a SMP

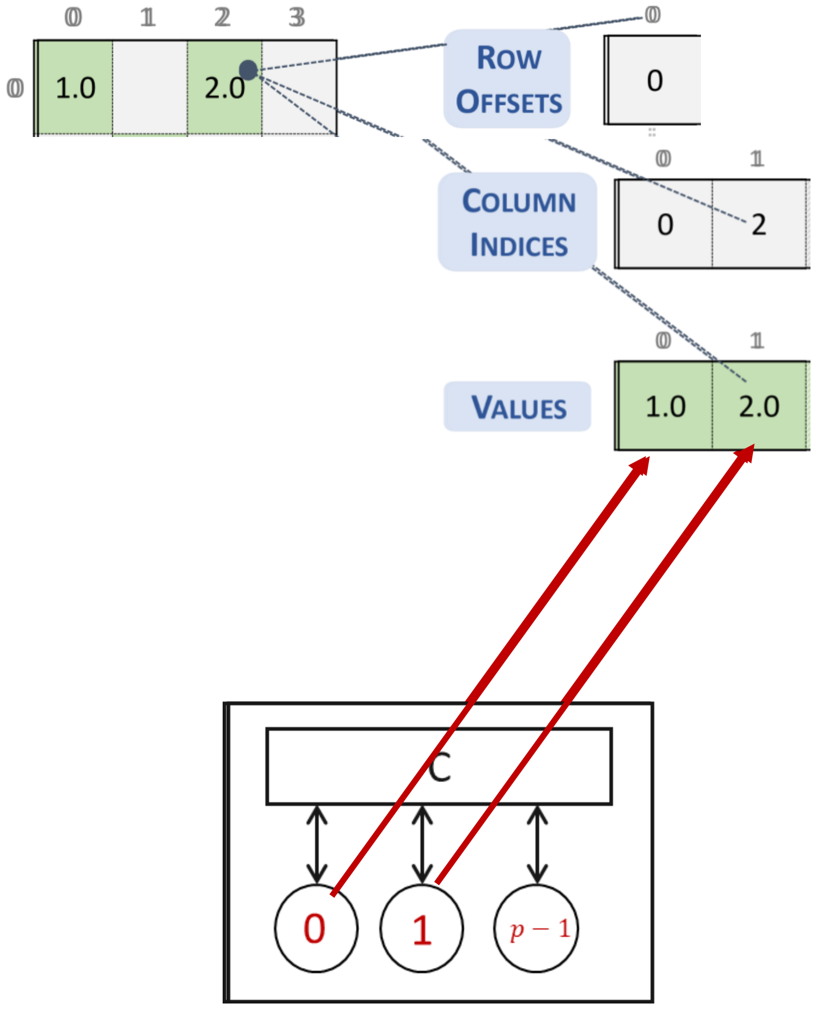


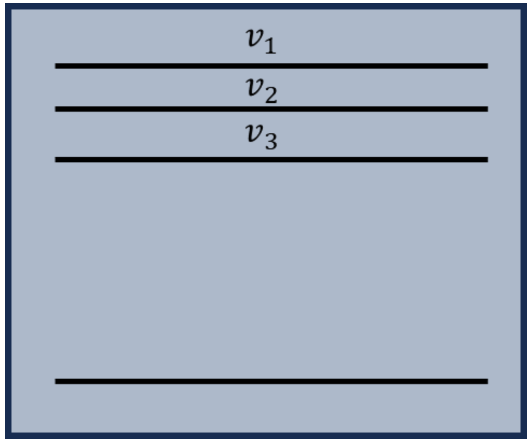
Sparse Dense Matrix Multiplication (SpMM)

- Approach #1: Assign each row to a SMP
- Processors execute the non-zero elements of the row.



Sparse Dense Matrix Multiplication (SpMM)





Step 1: Scale the rows of the second matrix with assigned non-zero values

Processor 1: $O_{p1} = 1.0 \times v_1$

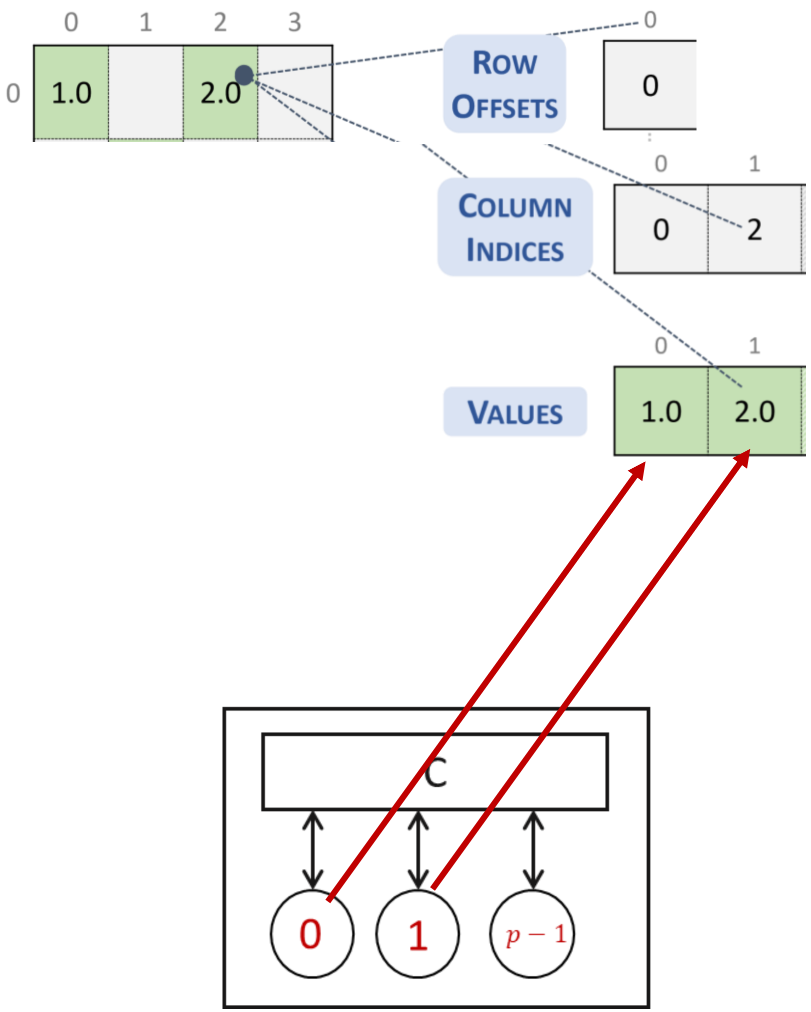
Processor 2: $O_{p2} = 2.0 \times v_3$

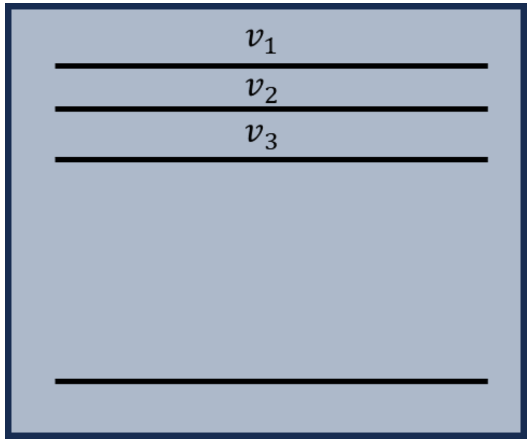
Note: Assuming processors are 1 indexed

Step 2: All-reduce to produce a single output

$O_1 = \ \ O_{p1} + O_{p2} + \cdots$

Sparse Dense Matrix Multiplication (SpMM)

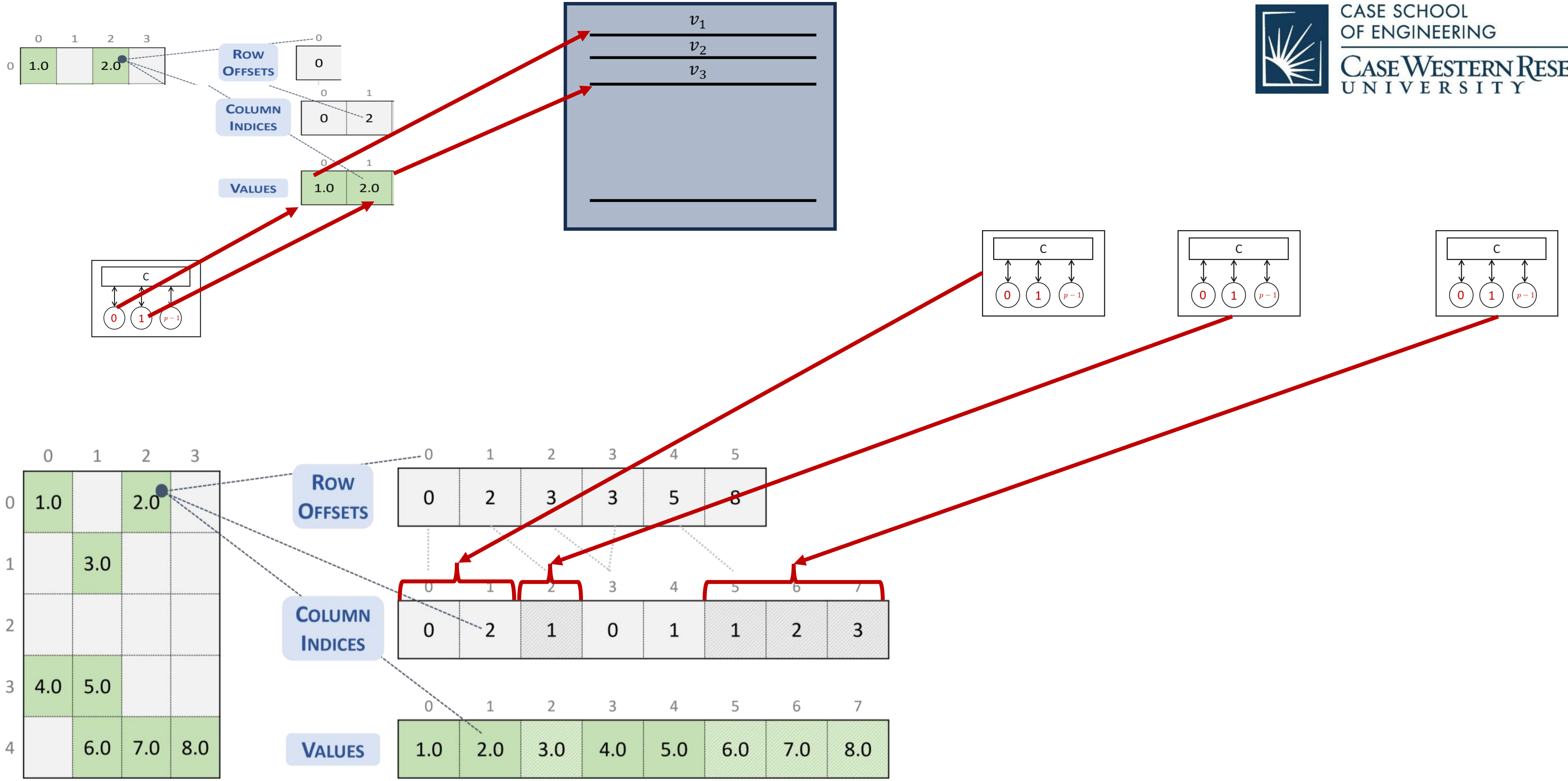




Ungraded HW – may ask in WA 3

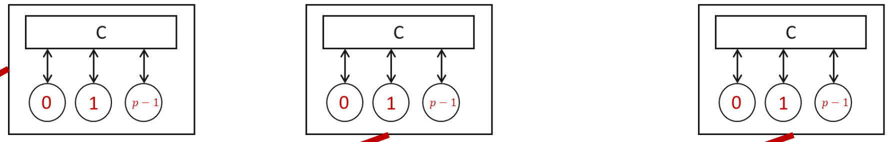
Given BlockID, ThreadID, can you write a GPU code for SpMM?

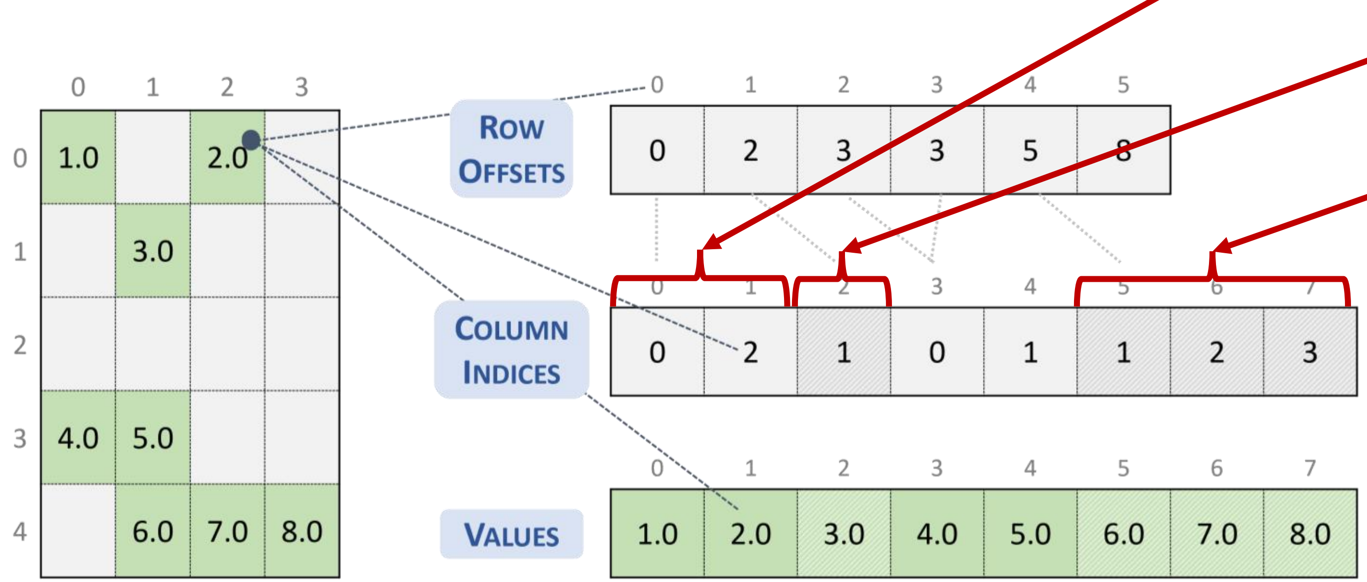
- Divide rows evenly among the blocks of GPU
- Within each block, divide column indices of rows evenly among the threads
- Each thread scales rows that it owns (based on column indices) of the second matrix
- Synchthreads
- All-reduce of the scaled outputs to produce final output (Recall Parallel Sum? In practice, GPUs have APIs to do this)


Row Offsets
Column Indices
Values
1.0
2.0
v_1
v_2
v_3
C
0
1
$p-1$
3.0
4.0
5.0
6.0
7.0
8.0

Sparse Dense Matrix Multiplication (SpMM)

- Approach #1: Assign each row to a SMP
- Processors execute the non-zero elements of the row.



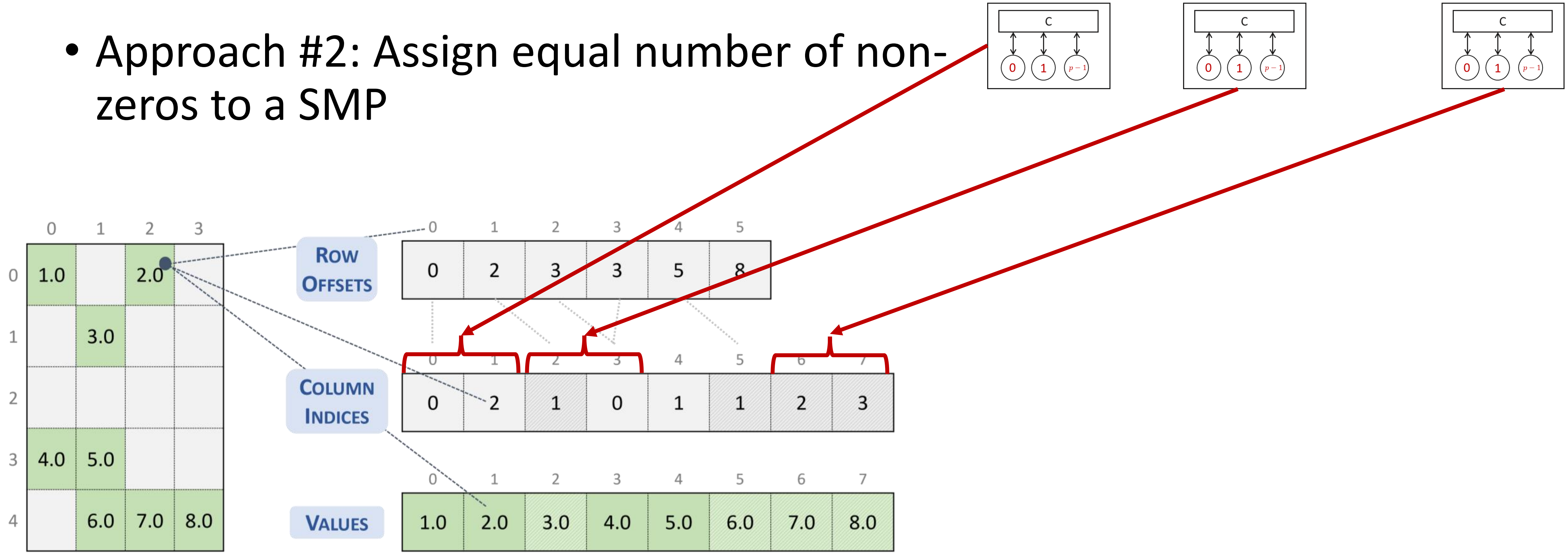


Pros: CSR format naturally leads to the assignment

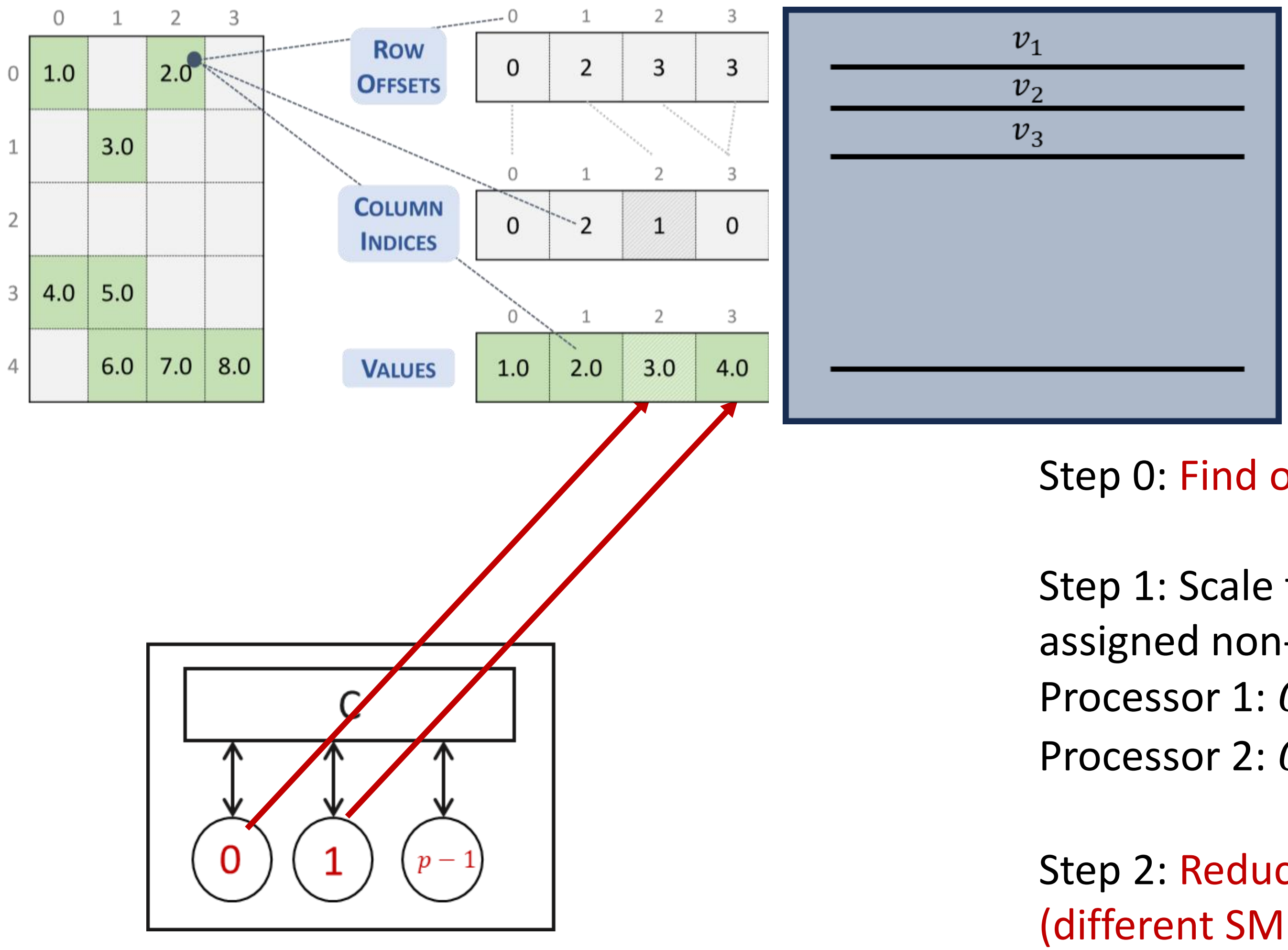
Cons: Load imbalance across SMPs

Sparse Dense Matrix Multiplication (SpMM)

- Approach #2: Assign equal number of non-zeros to a SMP



Sparse Dense Matrix Multiplication (SpMM)



Step 0: Find out row indices of the assigned columns

Step 1: Scale the rows of the second matrix with assigned non-zero values

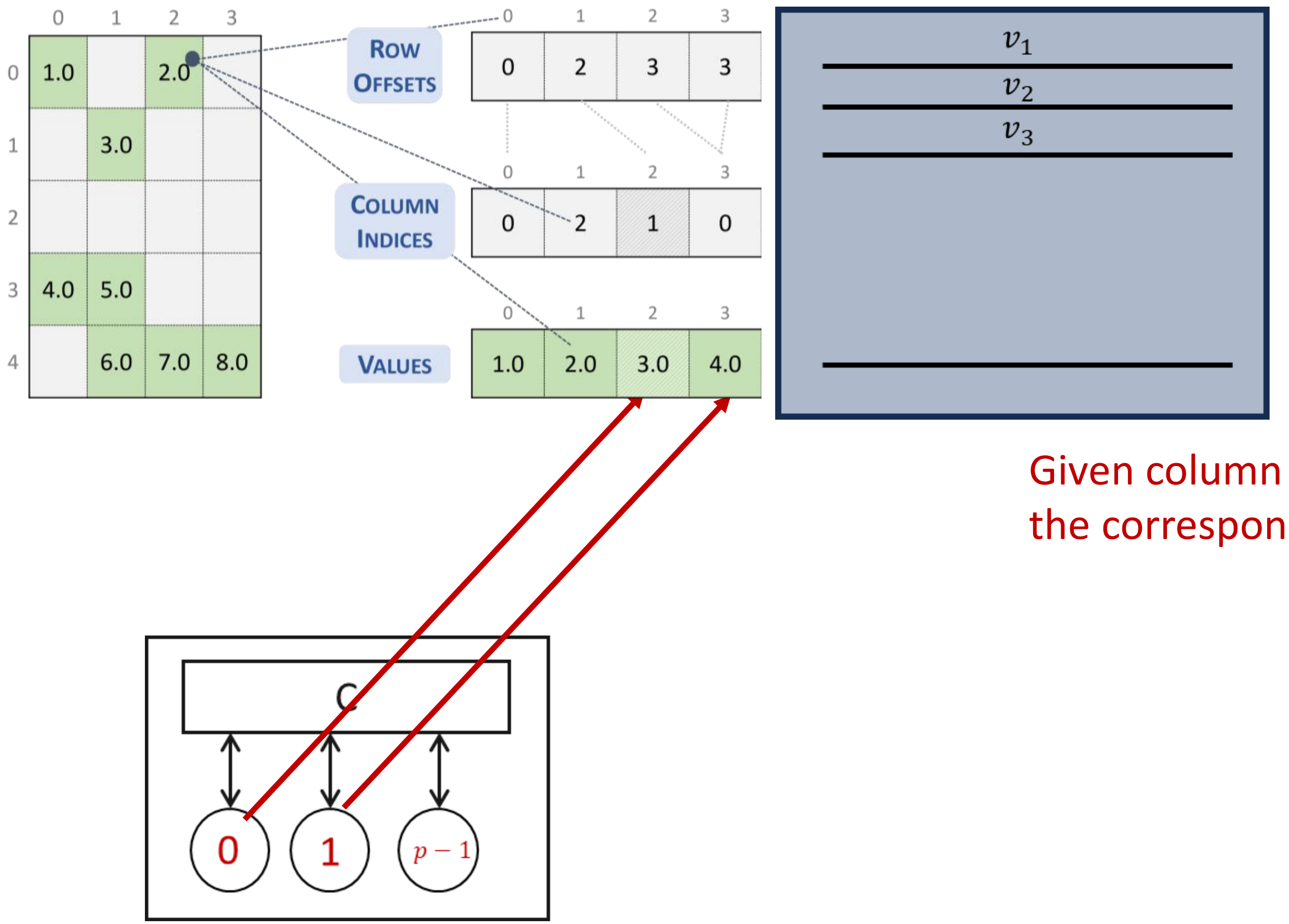
Processor 1: $O_{p1} = 3.0 \times v_2$

Processor 2: $O_{p2} = 2.0 \times v_1$

Note: Assuming processors are 1 indexed

Step 2: Reduce and add to appropriate row output (different SMPs can write to same row of the output)

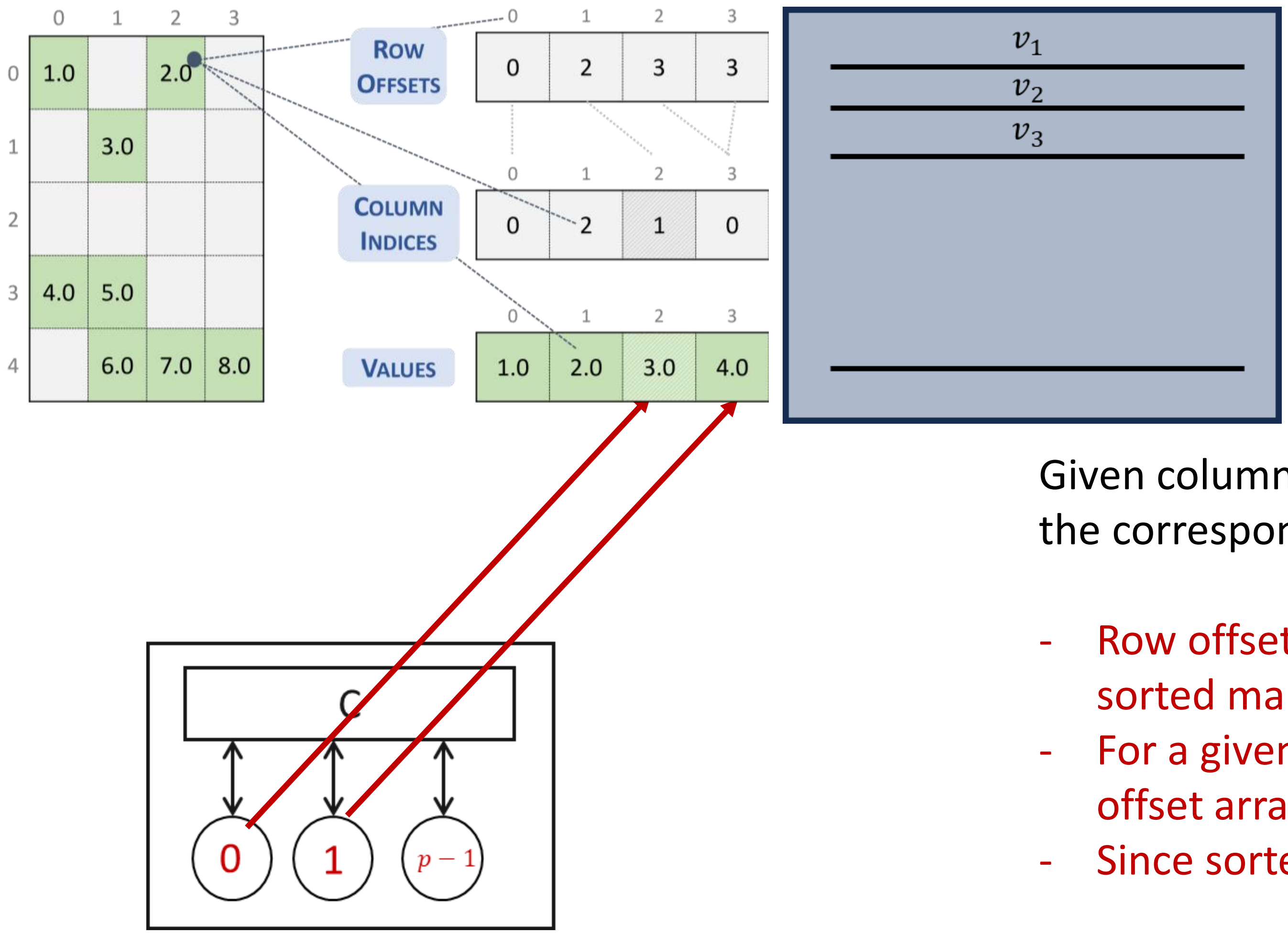
Sparse Dense Matrix Multiplication (SpMM)



Given column indices assignment, how can we find the corresponding row indices?

Note: Assuming processors are 1 indexed

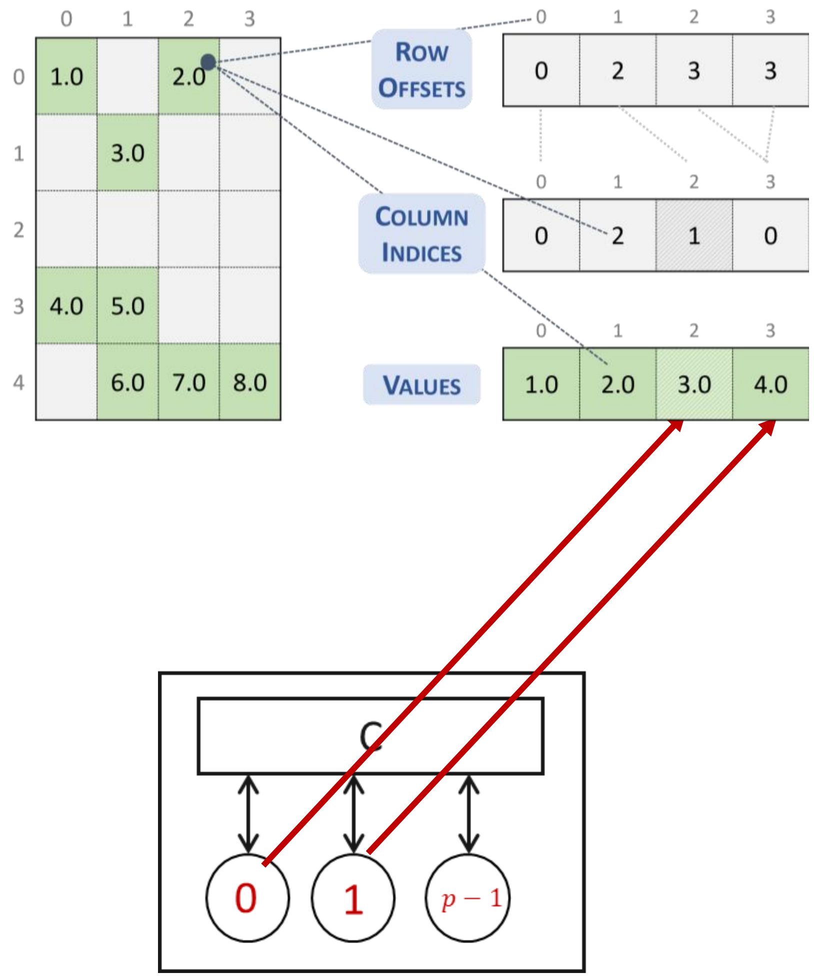
Sparse Dense Matrix Multiplication (SpMM)

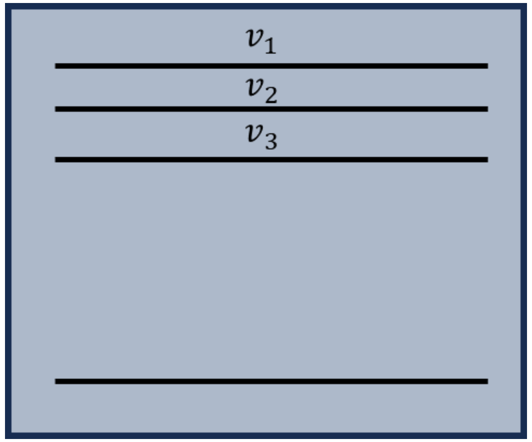


Given column indices assignment, how can we find the corresponding row indices? Binary Search

- Row offset array stores offsets of columns in a sorted manner
- For a given column offset, **Search** it in the row offset array
- Since sorted array – Binary search

Sparse Dense Matrix Multiplication (SpMM)





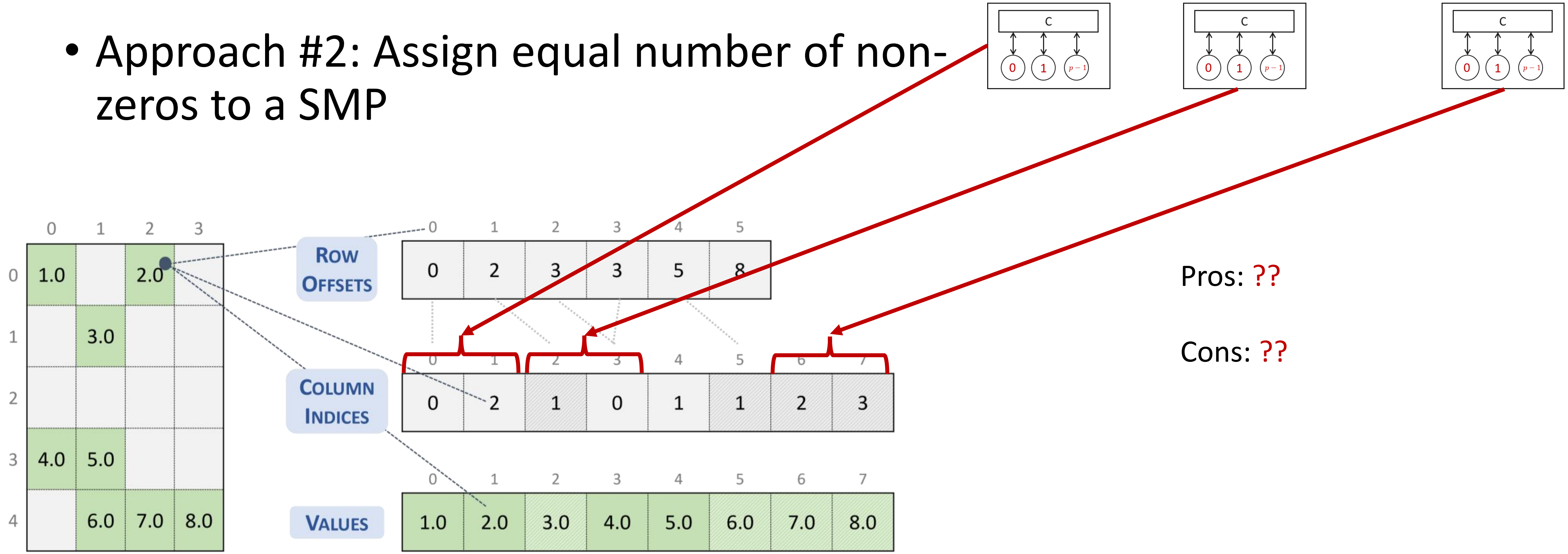
Given column indices assignment, how can we find the corresponding row indices? Binary Search

- Row offset array stores offsets of columns in a sorted manner
- For a given column offset, **Search** it in the row offset array
- Since sorted array – Binary search

- Ungraded HW Assignment: write a GPU code with BlockID and ThreadID. (will not test this approach in the exam)

Sparse Dense Matrix Multiplication (SpMM)

- Approach #2: Assign equal number of non-zeros to a SMP

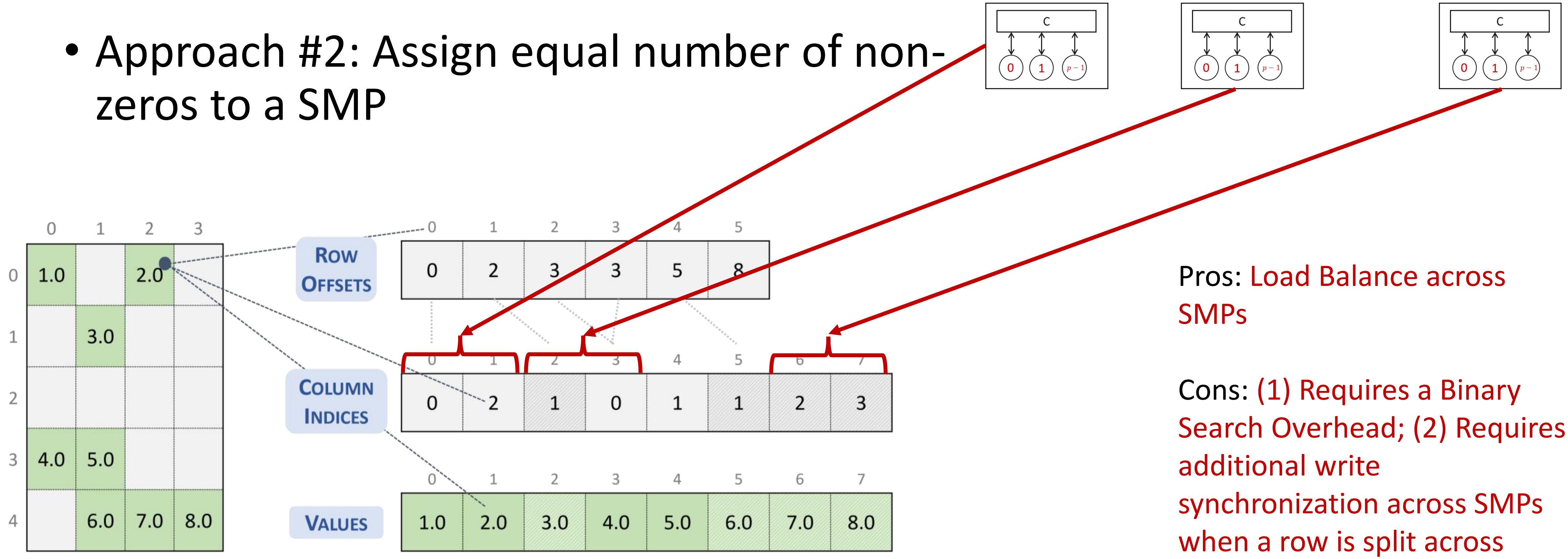


Pros: ??

Cons: ??

Sparse Dense Matrix Multiplication (SpMM)

- Approach #2: Assign equal number of non-zeros to a SMP

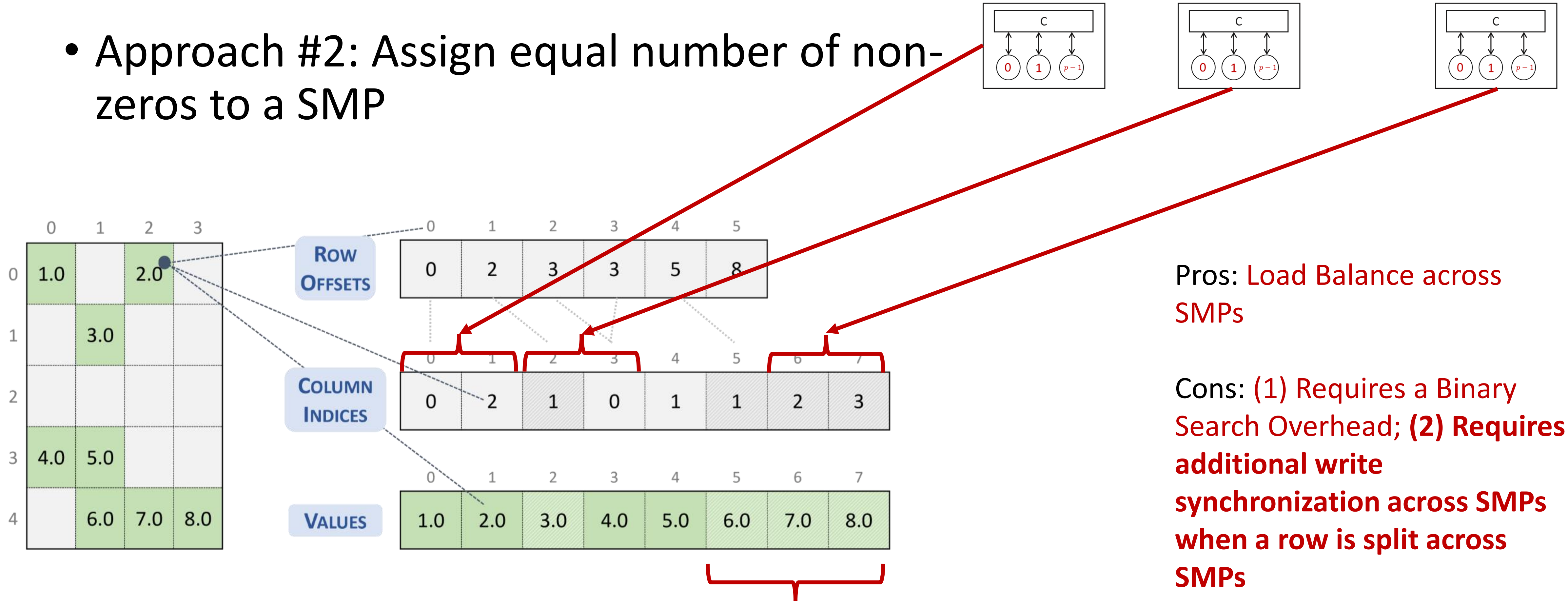


Pros: Load Balance across SMPs

Cons: (1) Requires a Binary Search Overhead; (2) Requires additional write synchronization across SMPs when a row is split across SMPs

Sparse Dense Matrix Multiplication (SpMM)

- Approach #2: Assign equal number of non-zeros to a SMP



Split across two processors

Pros: Load Balance across SMPs

Cons: (1) Requires a Binary Search Overhead; **(2) Requires additional write synchronization across SMPs when a row is split across SMPs**

Attention with Sparse Attention Mask

- Three Key Operations

- Operation #1: $Y = \ QK^T |\ M$: **Product of dense matrices under a sparse mask**
- Operation #2: $Z = Softmax(Y)$
- Operation #3: $O = ZV$: Product of Z and V matrices: **Product of a sparse and dense matrix**

- Q, K^T, V: Dense matrices
- Z: Sparse matrix

$QK^T | M$

- Pytorch Way

- Dense-dense matrix multiplication of $Y = QK^T$
  - Computation Complexity: $O(L^2 d_k)$
  - Storage Requirements: $O(L^2)$ (or $B_r \times B_C + d\ (B_r + B_c)$ if using flashattention)

- Invalidate entries (Set them to $-\infty$) in Y using mask M
  - Alternatively, initialize Y to $-\infty$ and only update the valid entries

$QK^T | M$

- Can we directly use SpMM kernel?

- No. But similar ideas can be utilized – Sampled Dense Dense Matrix Multiplication

- We will discuss in next class

Next Class

- 11/4 Lecture 19
    - Accelerating Transformer Model: Sparse Transformers Acceleration II

Thank You

- Questions?
- Email: sanmukh.kuppannagari@case.edu