CSDS 451: Designing High Performant Systems for AI

Lecture 19

11/4/2025

Sanmukh Kuppannagari

sanmukh.kuppannagari@case.edu

https://sanmukh.research.st/

Case Western Reserve University

Outline

- Accelerating Sparse Transformers – Part II

Announcements

- WA 3 was out
- Due by next Saturday

Next Few Classes and Deadlines

- 11/6 – Modeling Cluster of Accelerators
- 11/11 – Distributed Training Techniques
- 11/13 – Distributed Training Techniques (WA 3 due)
- 11/18 – No Class (Traveling for a conference)
- 11/20 – Distributed Training Techniques

Next Few Classes and Deadlines

- 11/25 – Exam
- 11/27 – Thanksgiving Holiday
- 12/2 – Guest Lecture
- 12/4 – Guest Lecture (WA4 due)
- 12/10 – NO final. Projects Due

Outline

- Accelerating Sparse Transformers – Part II

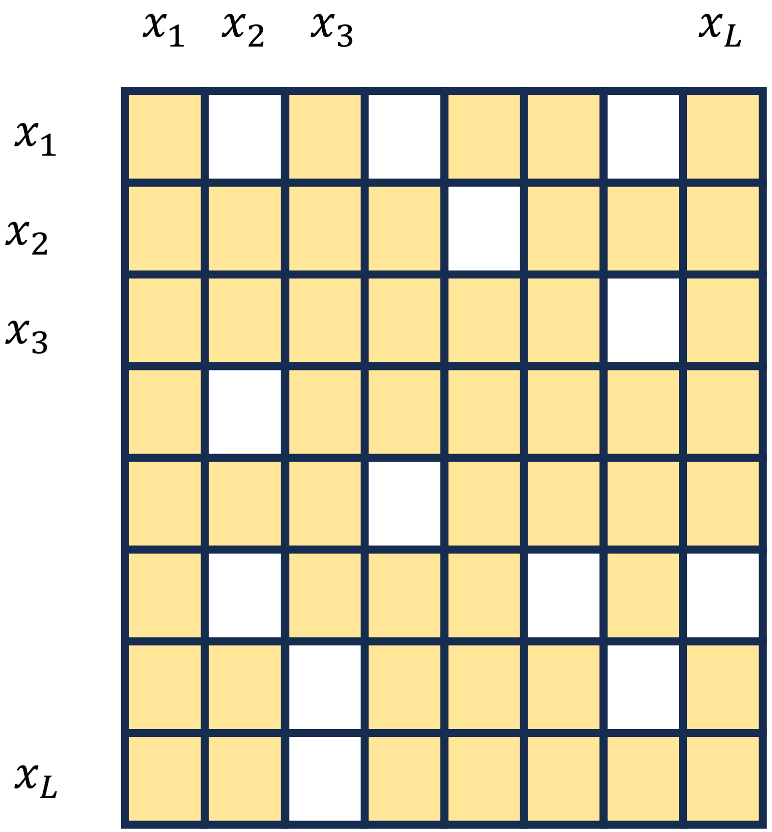
Attention with Sparse Attention Mask

- Three Key Operations

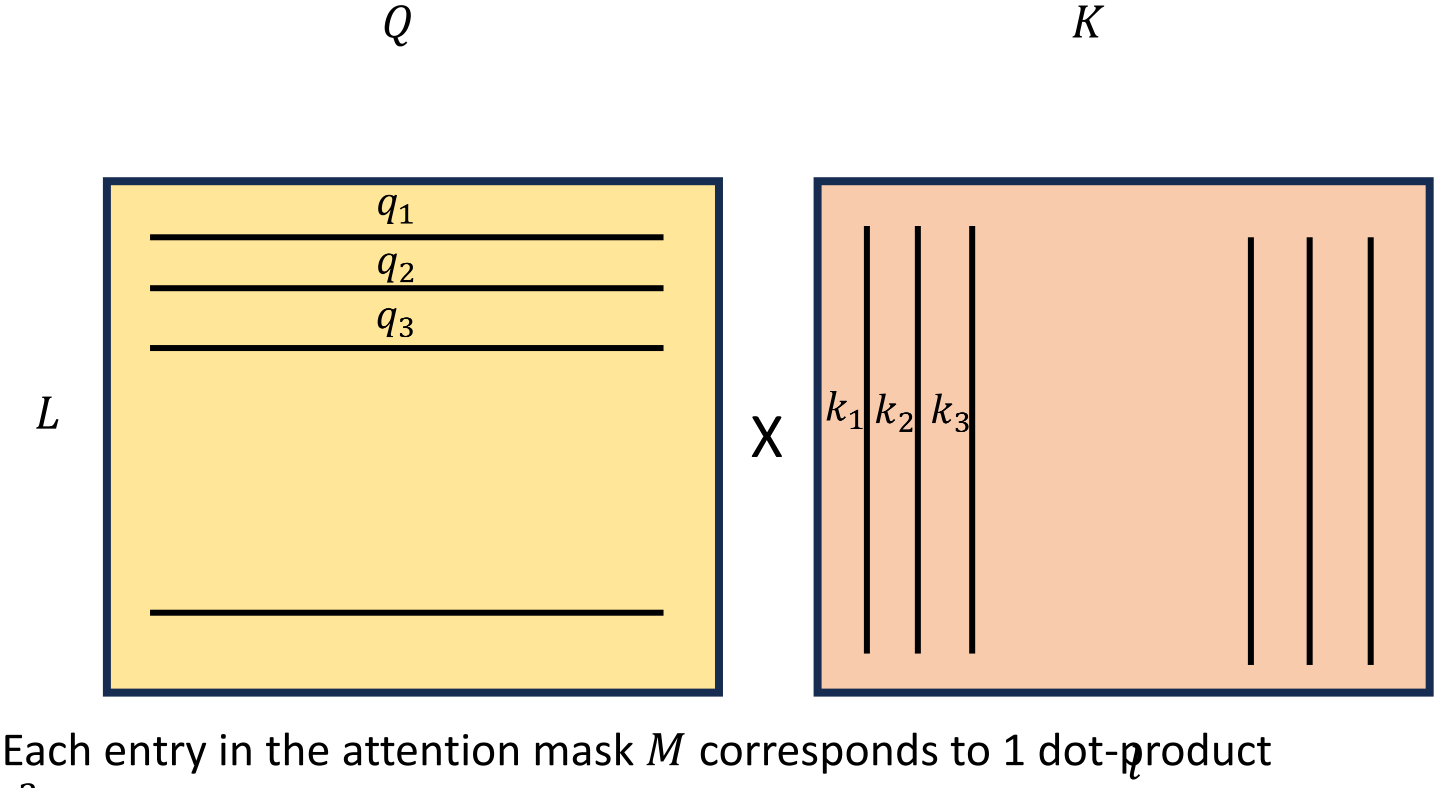
- Operation #1: $Y = \ QK^T |\, M$: Product of Q and K^T matrices under mask M
- Operation #2: $Z = Softmax(Y)$
- Operation #3: $O = ZV$: Product of Z and V matrices

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

QK Product with Attention Mask



M

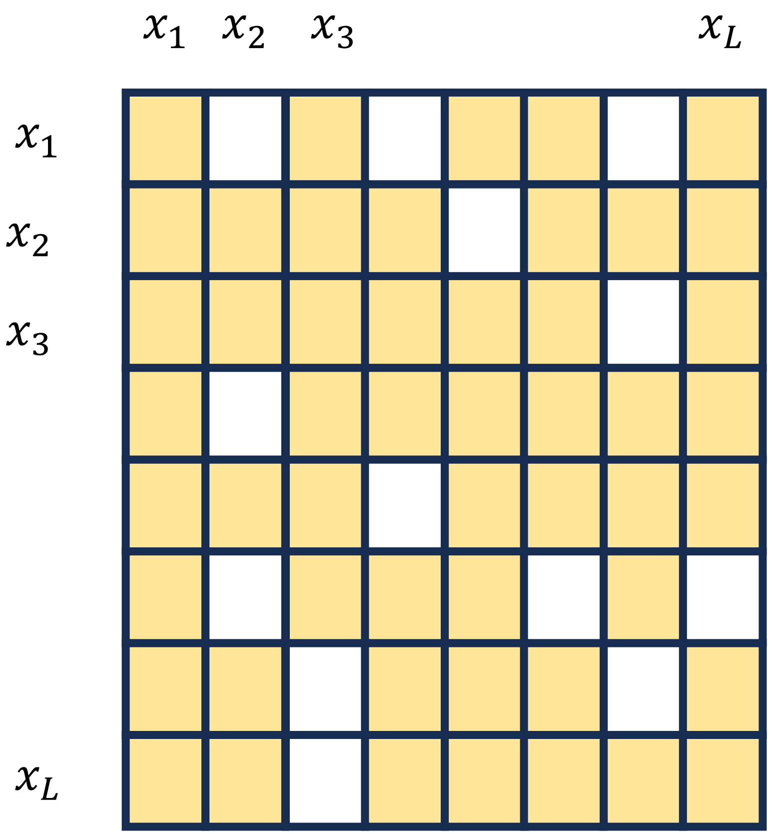


Each entry in the attention mask M corresponds to 1 dot-product

l^2 entries in total

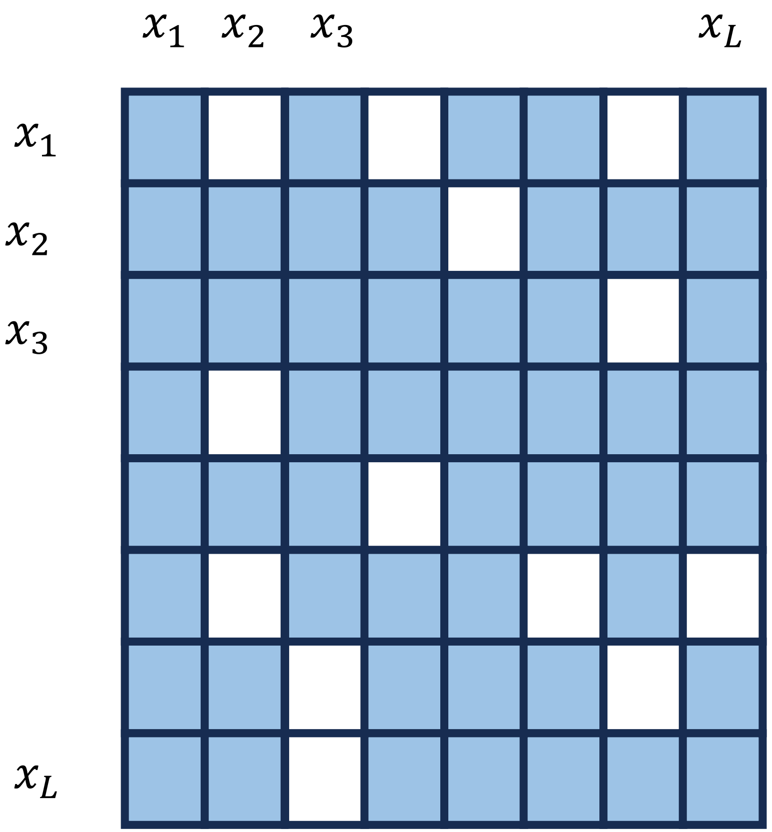
Sparsity factor s determines how many dot-products actually need to be computed

QK Product with Attention Mask



M

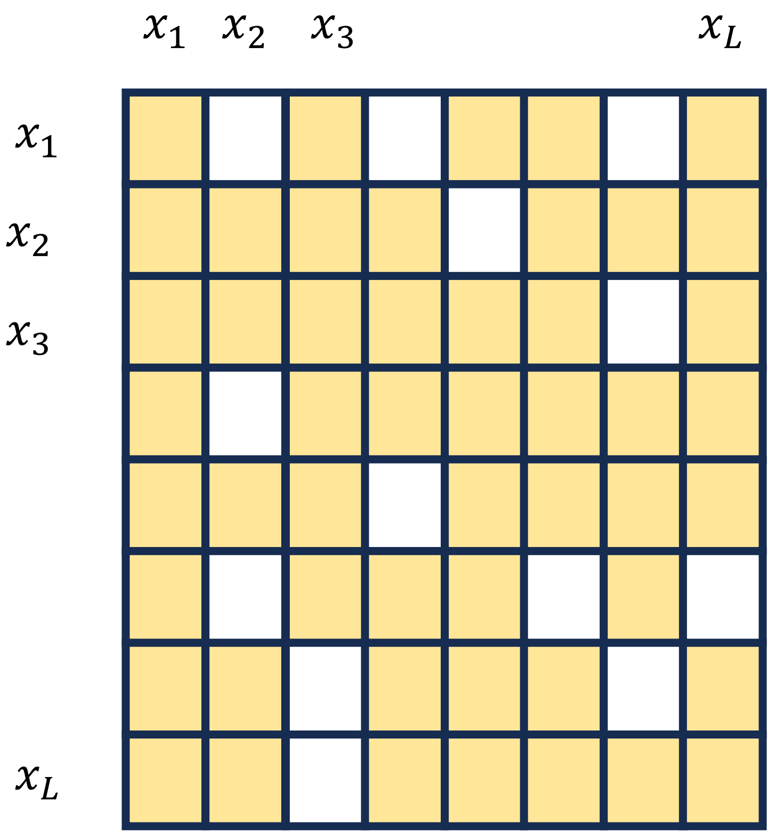
$Y = QK^T$



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

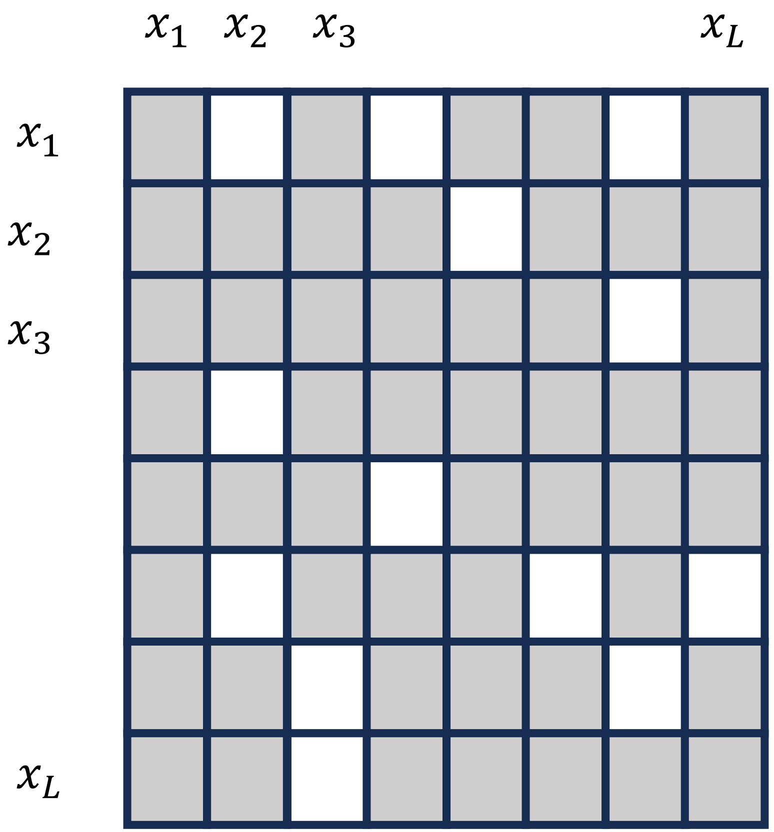
In practice, elements corresponding to 0 mask are set to $-\infty$: Why? Will ask in WA 3

Softmax with Attention Mask



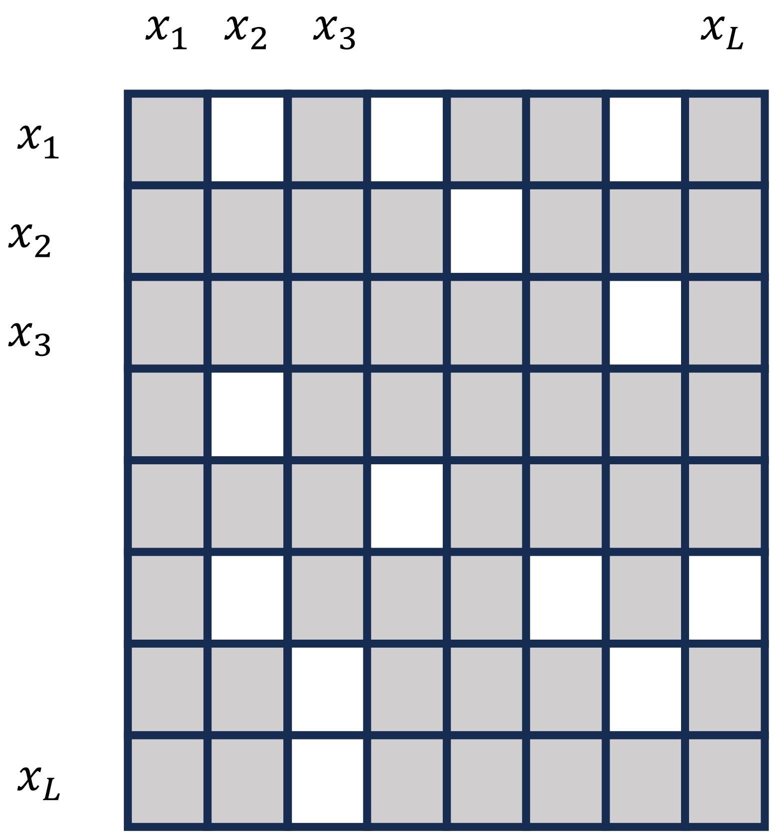
M

$Z = softmax(QK^T)$

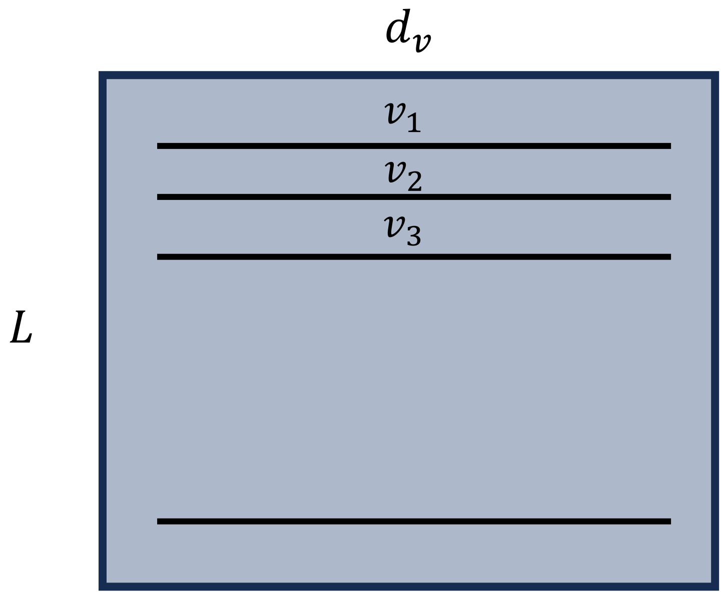


Softmax will replace all $-\infty$ elements with 0: How? Will ask in WA 3.
The pattern of 0 and non-zero still remains the same

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)

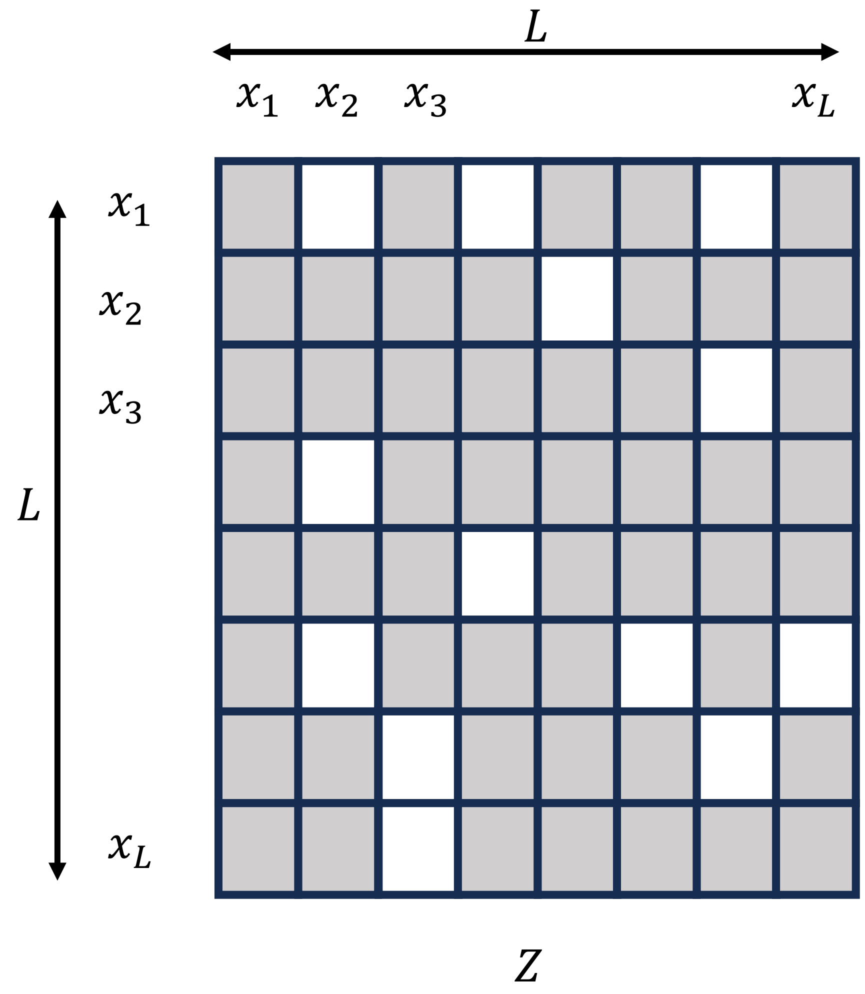
- Let nnz be the number of non-zero elements in Z

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

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

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)

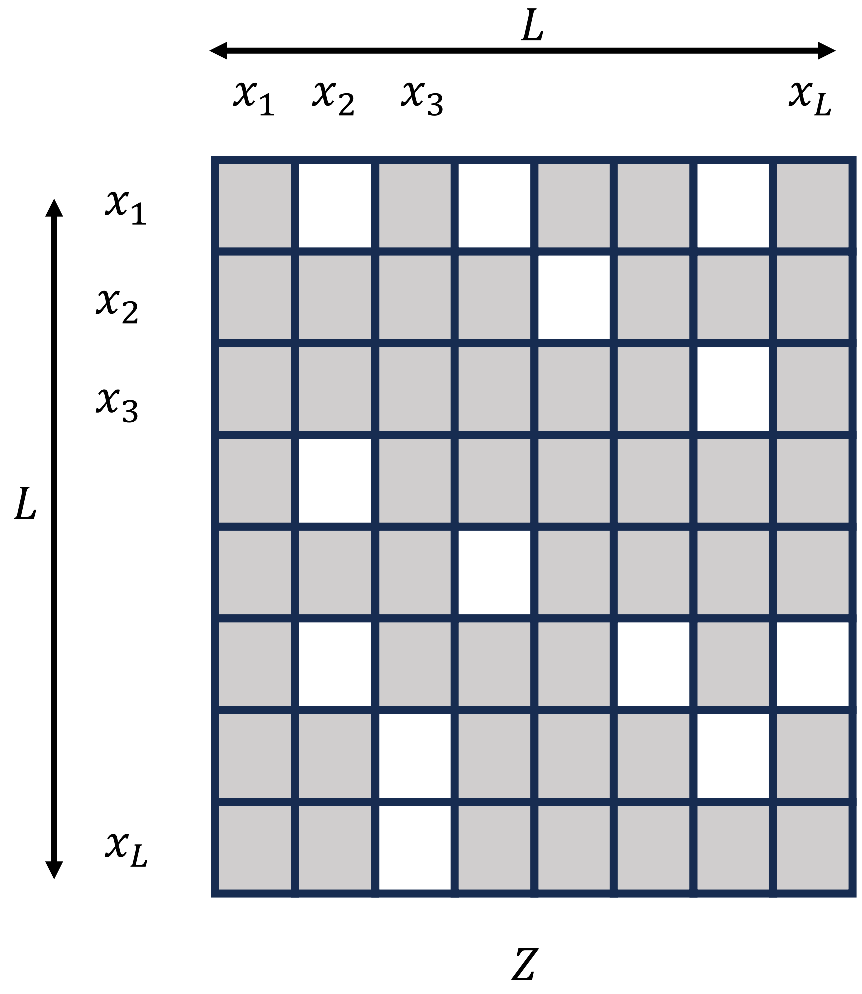
	0	1	2	3	4	5	6	7
Row Indices	0	0	1	4	4	5	5	5
Column Indices	0	2	1	0	1	1	2	3
Values	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0

SpMM Storage Format - CSR

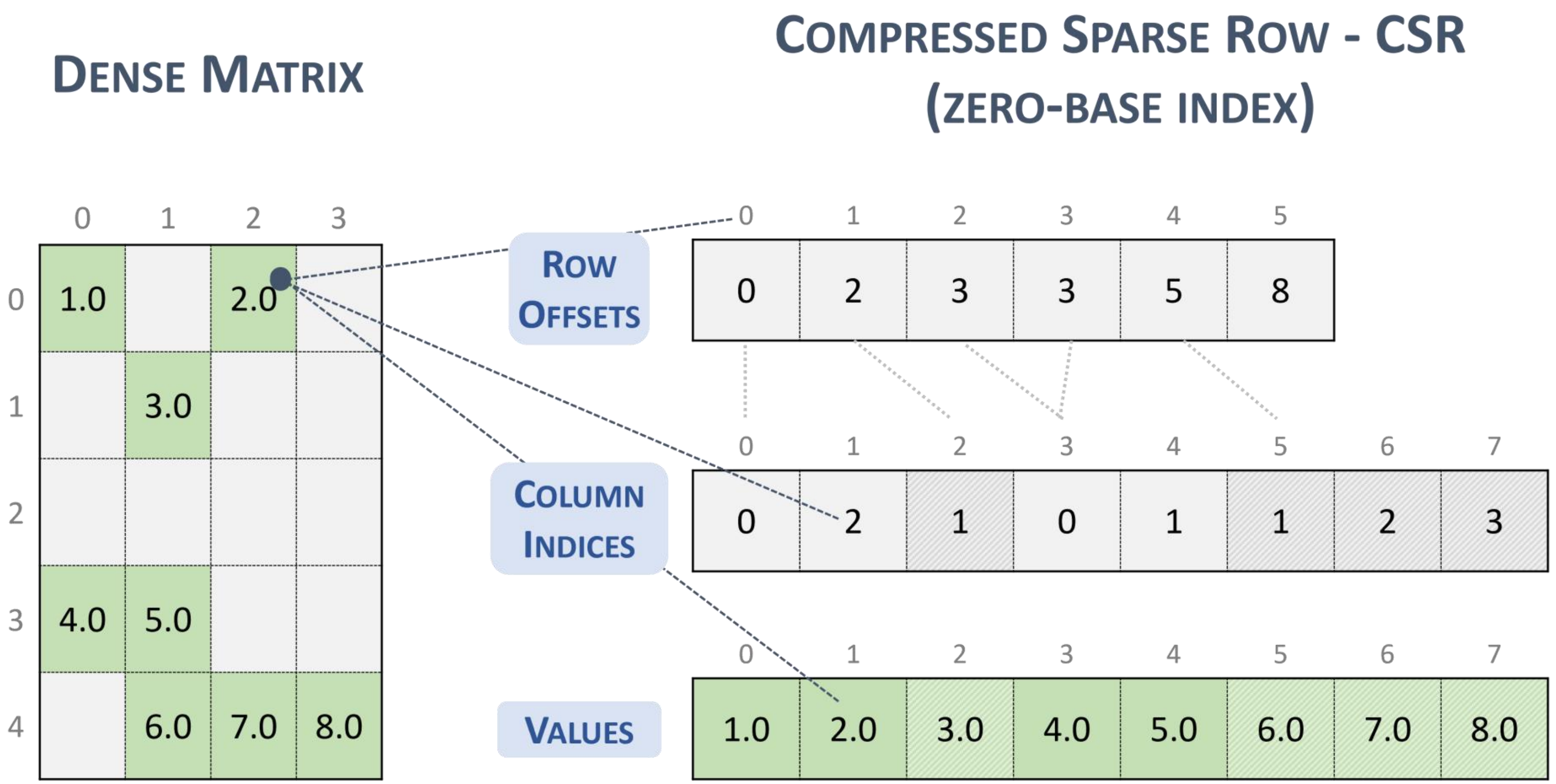
- Compressed Sparse Row Format

- 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

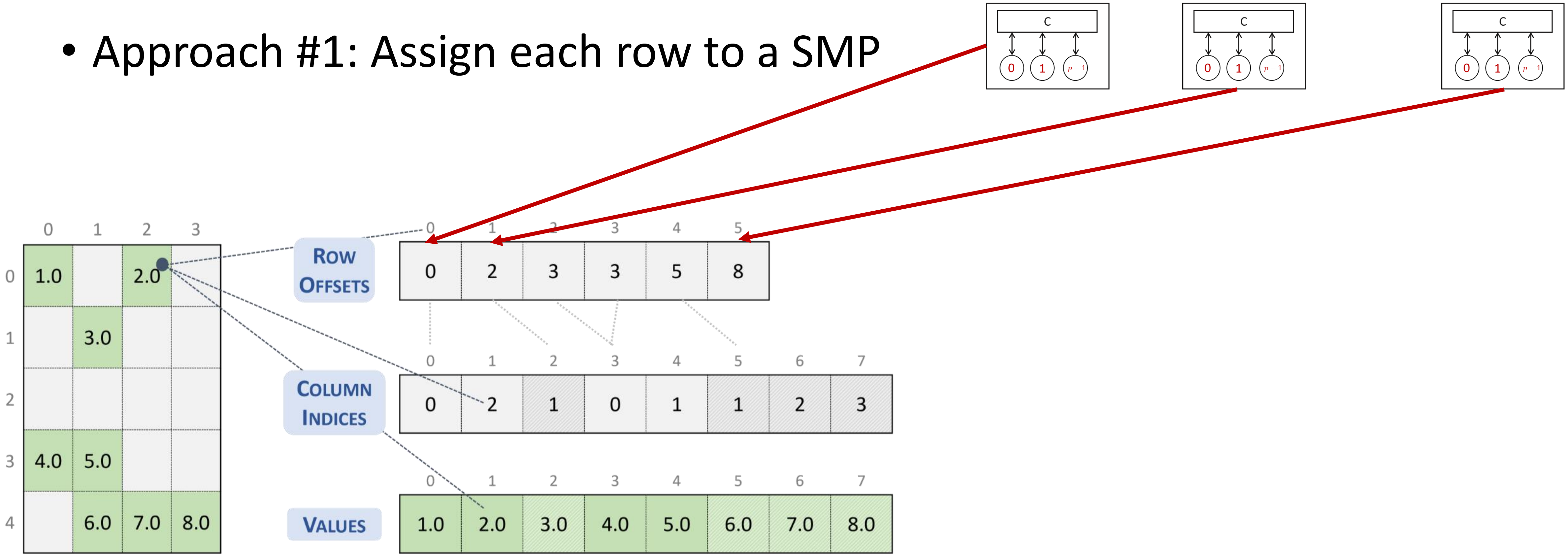


SpMM Storage Format - CSR

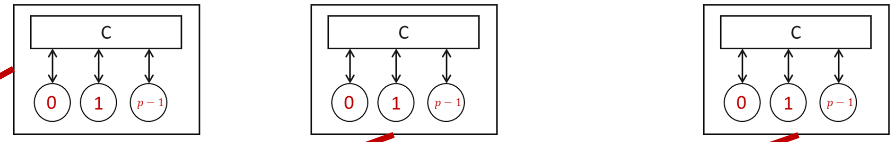


Sparse Dense Matrix Multiplication (SpMM)

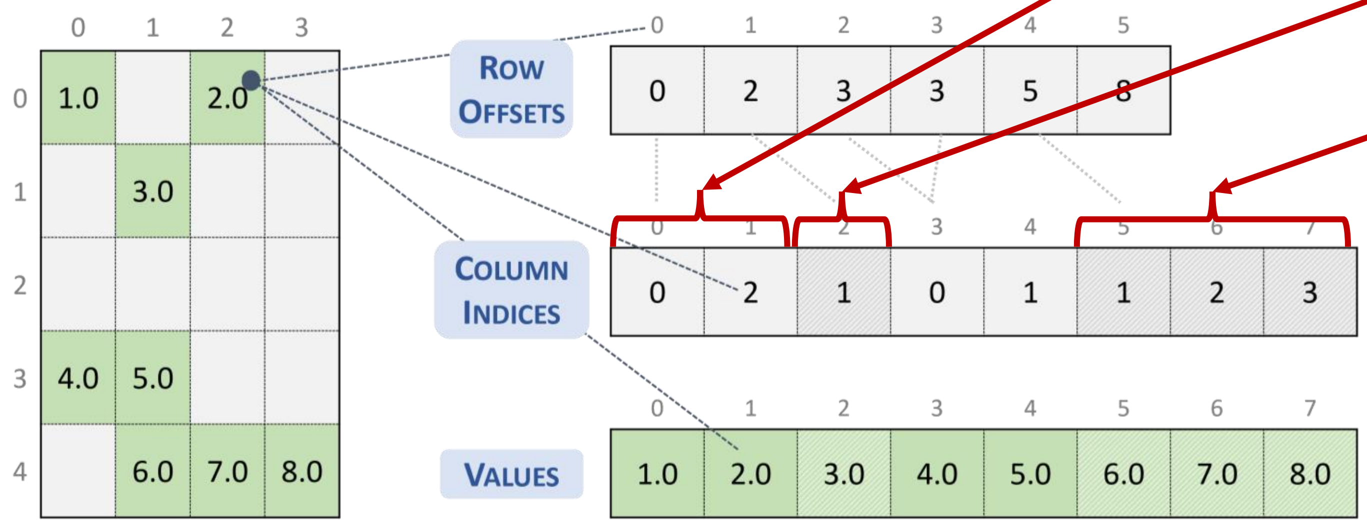
- Approach #1: Assign each row to a SMP



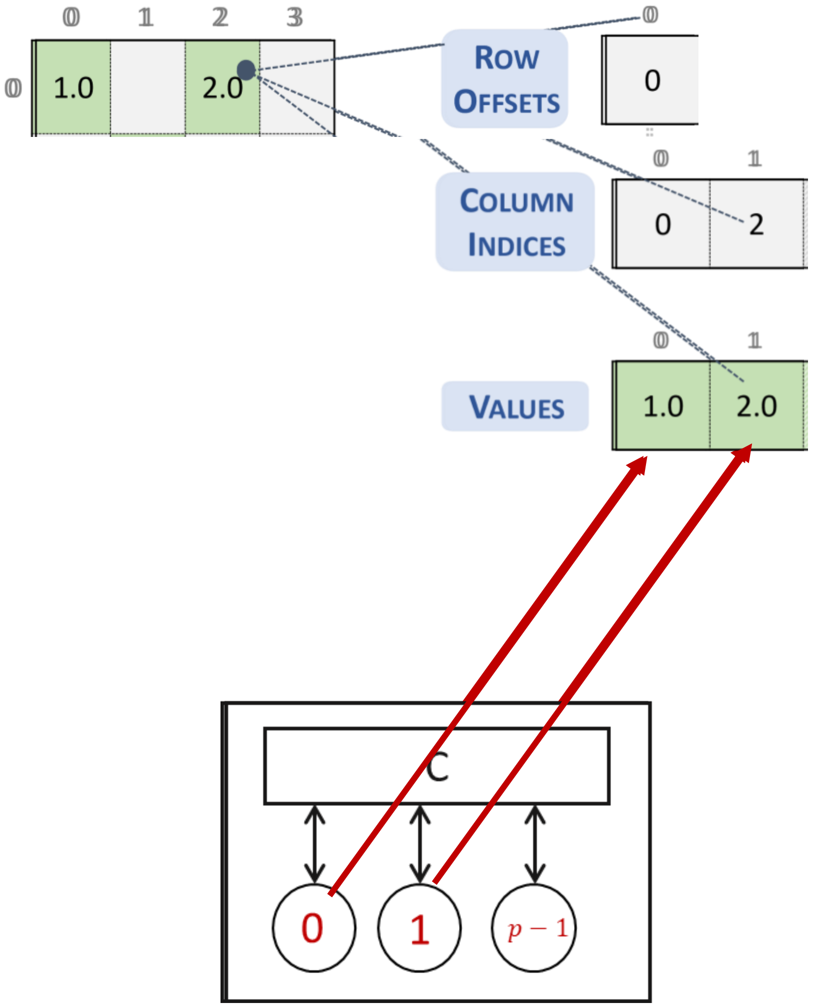
Sparse Dense Matrix Multiplication (SpMM)

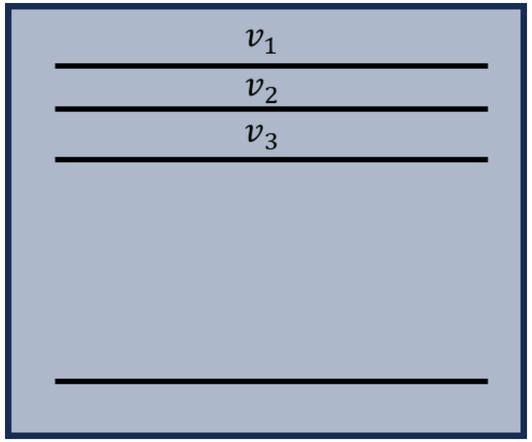


- Approach #1: Assign each row to a SMP
- Processors execute the non-zero elements of the row.



Sparse Dense Matrix Multiplication (SpMM)





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

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

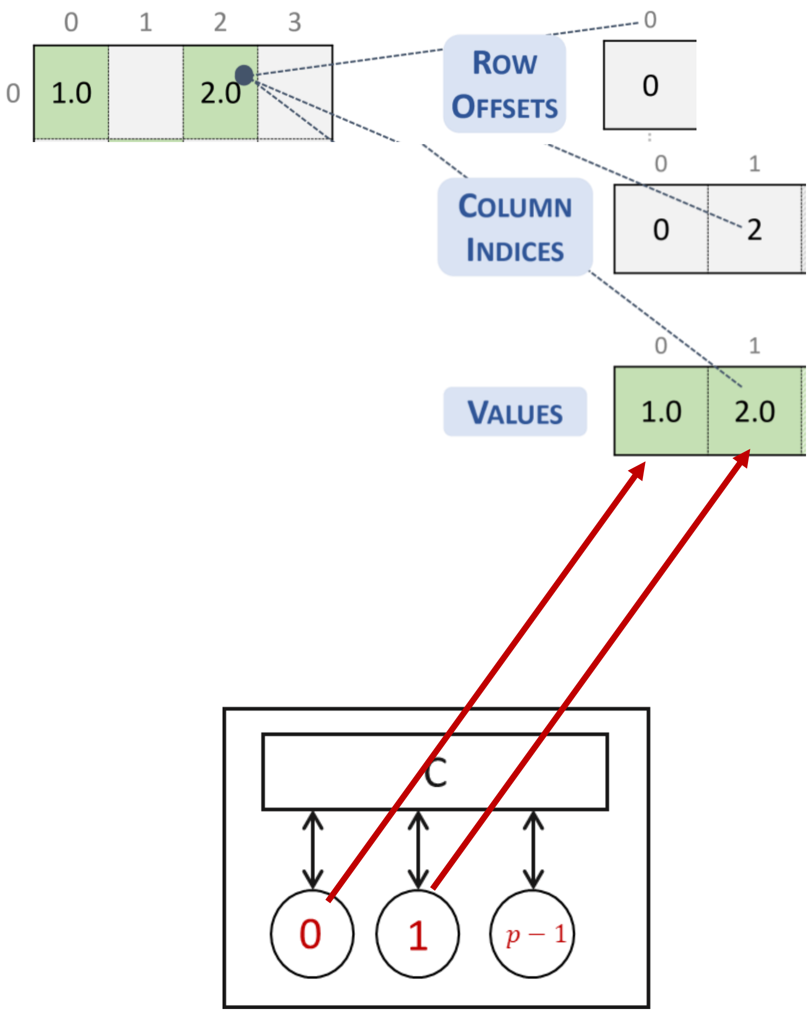
Processor 2: $O_{p2} = 2.0 \times v_3$

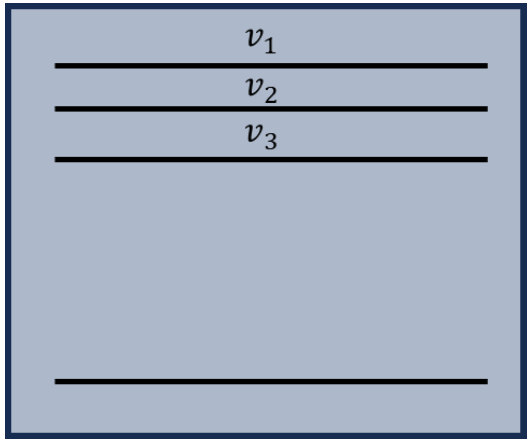
Note: Assuming processors are 1 indexed

Step 2: All-reduce to produce a single output

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

Sparse Dense Matrix Multiplication (SpMM)

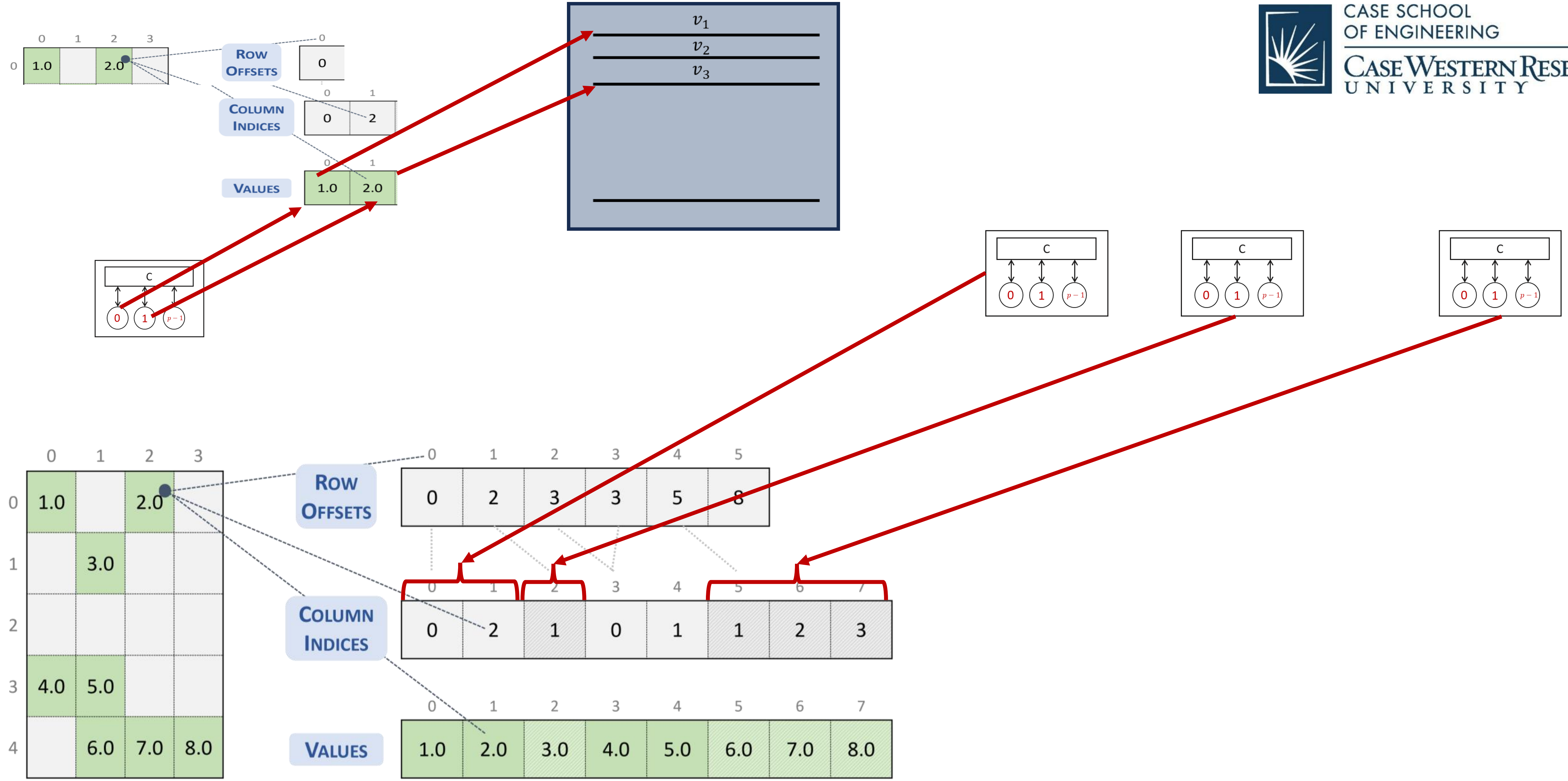




Question in WA 3

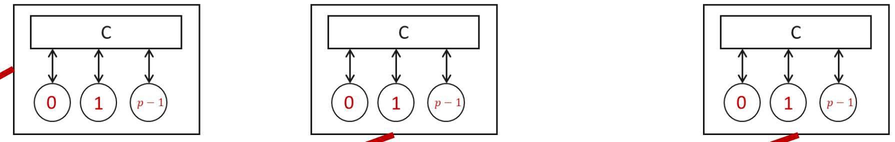
Given BlockID, ThreadID, can you write a GPU code for SpMM?

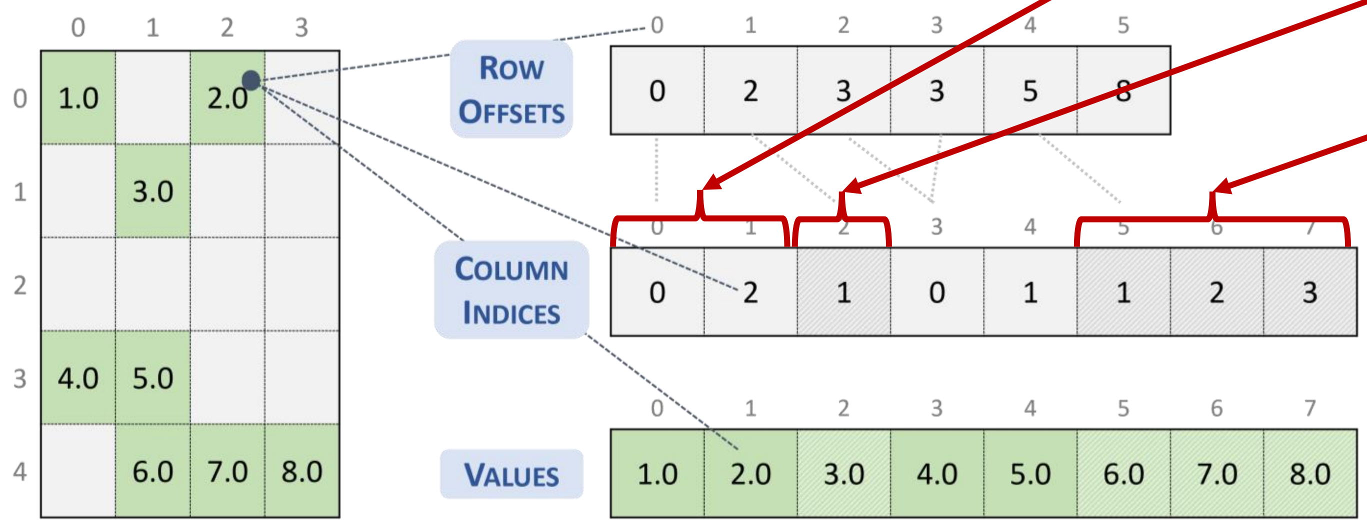
- Divide rows evenly among the blocks of GPU
- Within each block, divide column indices of rows evenly among the threads
- Each thread scales rows that it owns (based on column indices) of the second matrix
- Synchthreads
- All-reduce of the scaled outputs to produce final output (Recall Parallel Sum? 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
$p-1$
0 2 3 3 5 8
0 2 1 0 1 1 2 3
1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0
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

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 $\mathrm{Y} = QK^T$
  - 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

Today’s Lecture

- SDDMM Algorithm Design Idea
  - You will implement the algorithm in WA3
- Using Flash attention on sparse masks
  - Block Sparsity
- Algorithms to reduce the number of blocks in block sparsity

Today's Lecture

- SDDMM Algorithm Design Idea
  - You will implement the algorithm in WA3
- Using Flash attention on sparse masks
  - Block Sparsity
- Algorithms to reduce the number of blocks in block sparsity

Sampled Dense Dense Matrix Multiplication (SDDMM)

- Key Idea/Similarity with SpMM

- SpMM: $O = ZV$, Z: sparse matrix, V: dense matrix
  - For each non-zero element of Z at location Z_{ij}
    - Fetch row j of V
    - Multiply with the element
    - Add the result to the output O_i

- SDDMM: $O = AB|M$: A, B: dense matrices, M: sparse mask
  - For each non-zero element of M at location M_{ij}
    - Fetch row i of A, Fetch column j of B,
    - Perform dot product between the fetched row and column
    - Write the result to the output O_{ij}

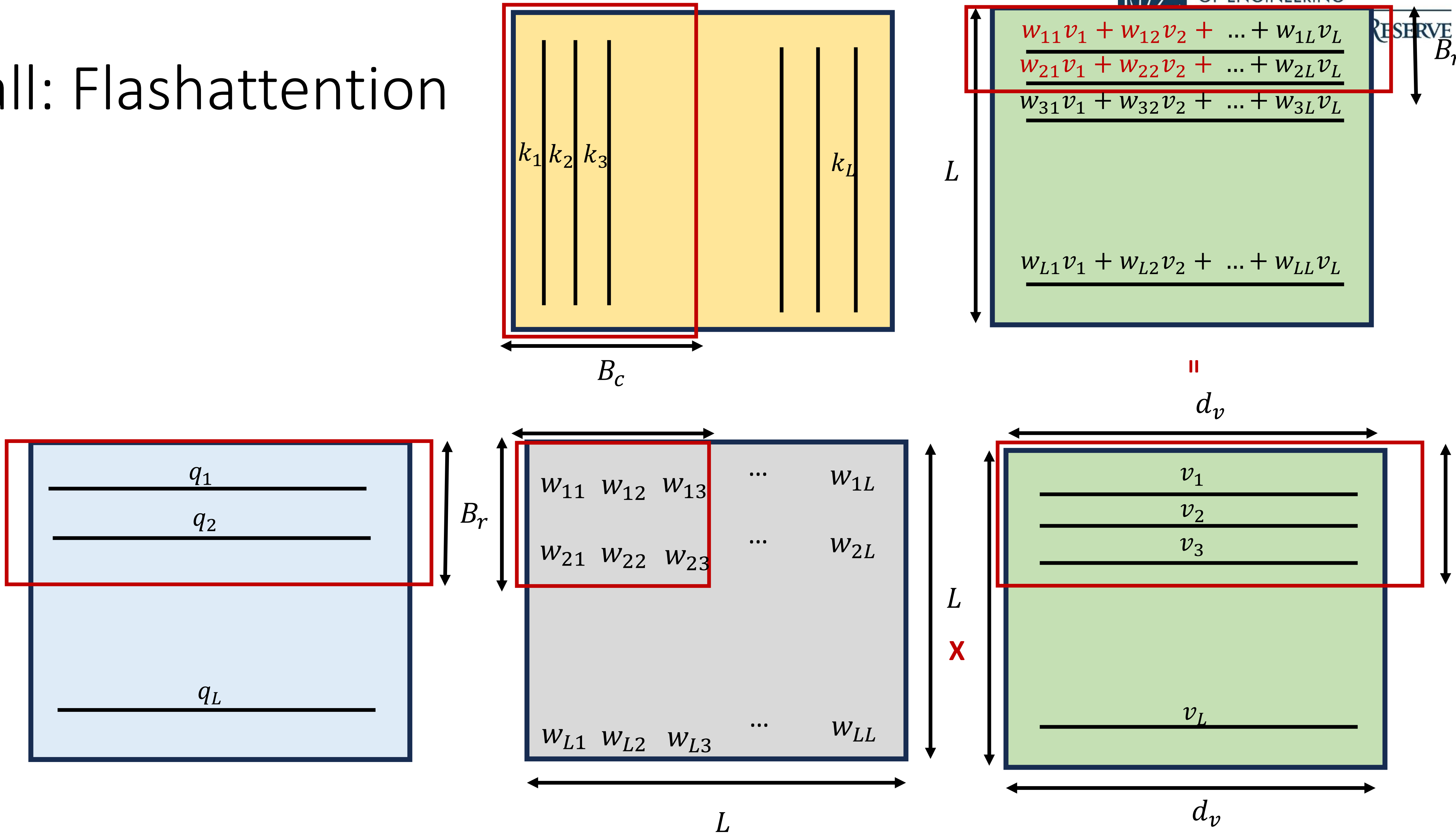
Sampled Dense Dense Matrix Multiplication (SDDMM)

- Key Idea/Similarity with SpMM
- For each non-zero, instead of a vector scaling operation
  - You perform a dot product
- WA3, second part of Q5
  - I will make that extra credits. (so 10 points extra credits)
  - Will not ask in exam
- (This lecture is already getting too big :’-))

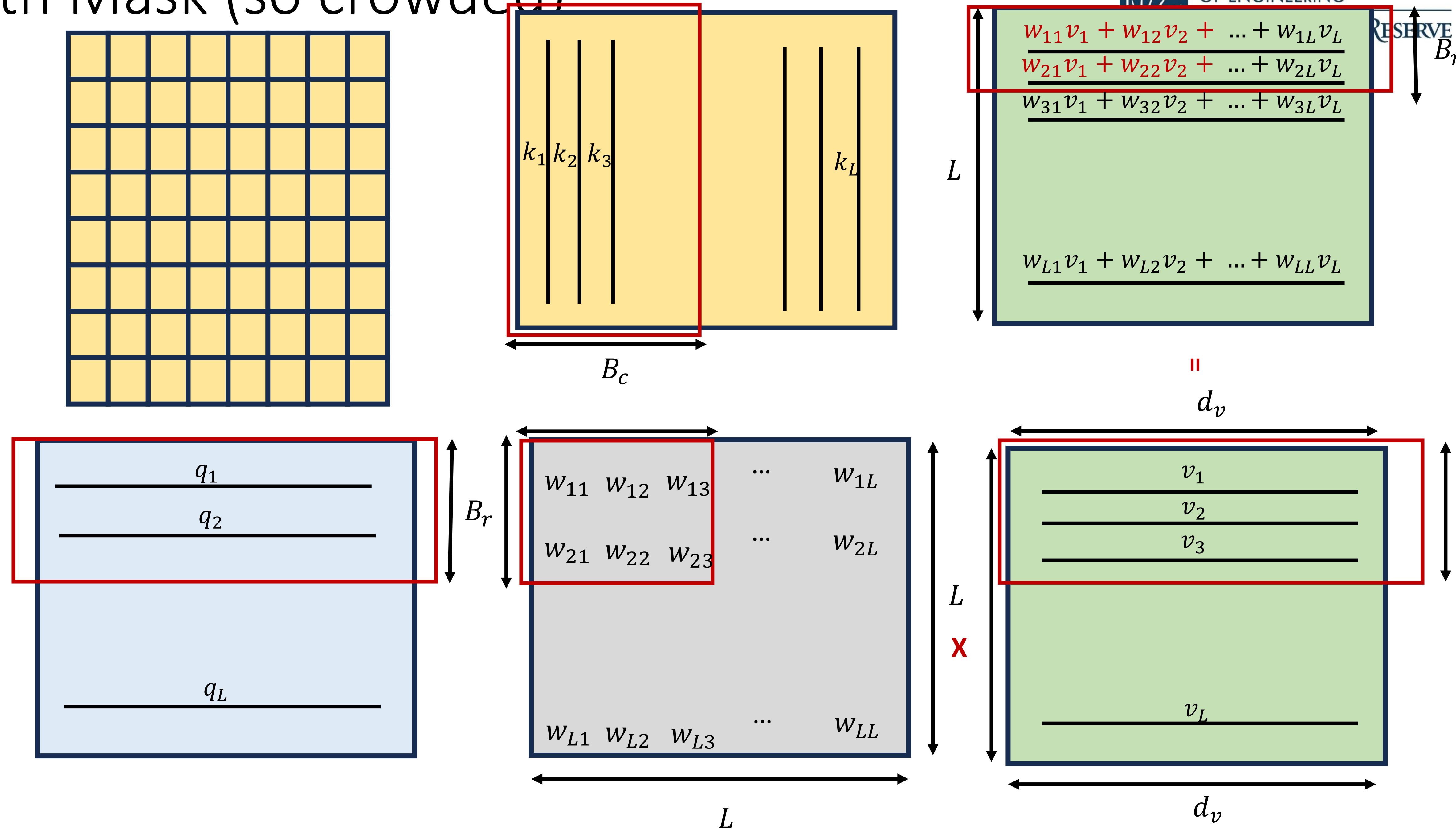
Today’s Lecture

- SDDMM Algorithm Design Idea
  - You will implement the algorithm in WA3
- Using Flash attention on sparse masks
  - Block Sparsity
- Algorithms to reduce the number of blocks in block sparsity

Recall: Flashattention



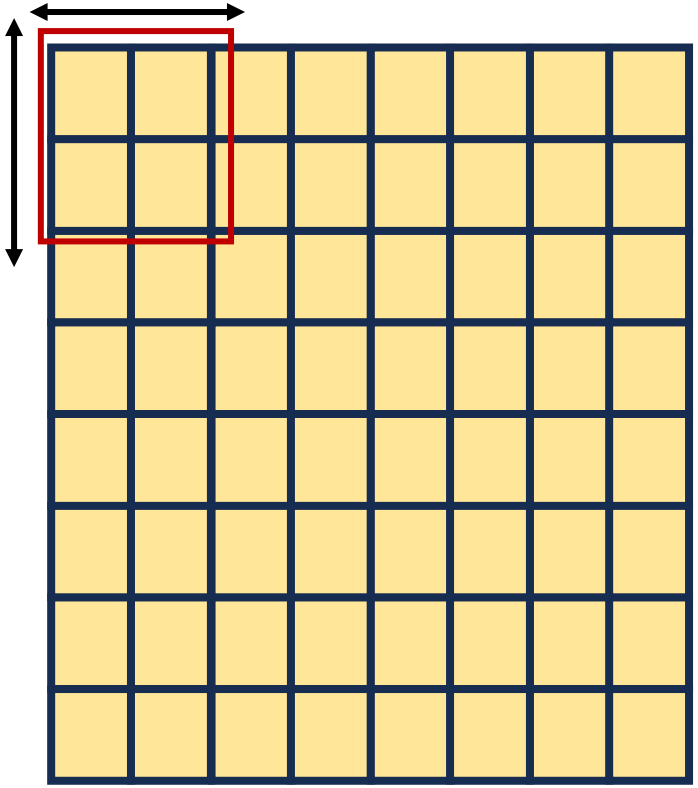
With Mask (so crowded)



Flash Attention on Attention Mask

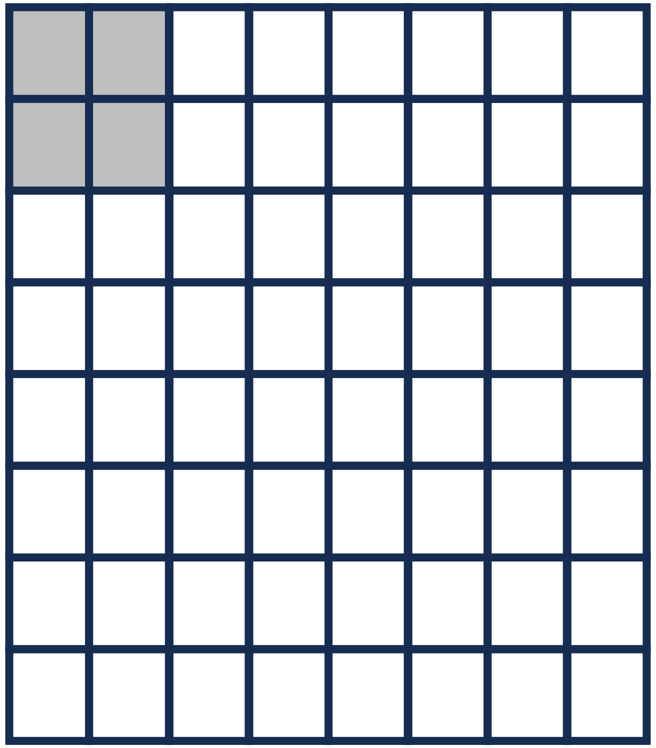
B_c M

B_r



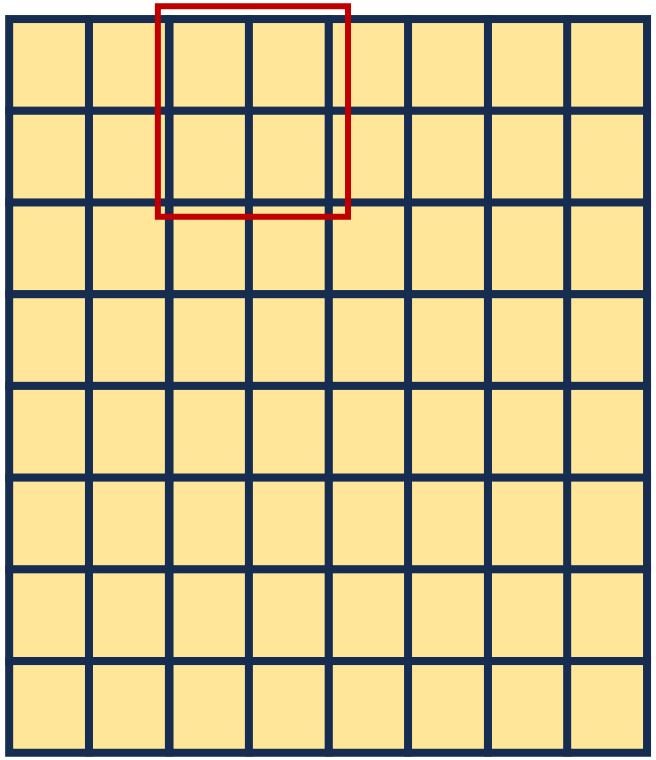
$$B_c = B_r = 2$$

QK^T/Y



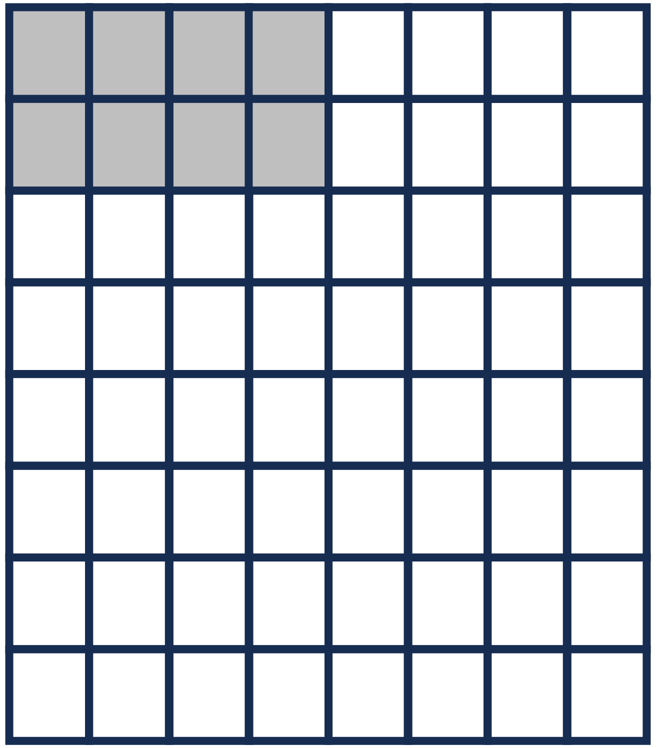
Flash Attention on Attention Mask

M



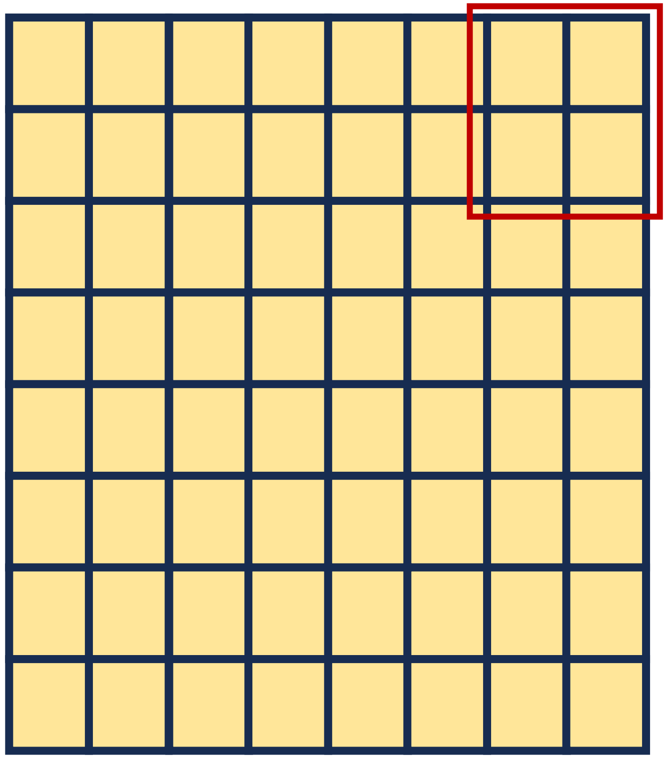
$B_c = B_r = 2$

QK^T/Y



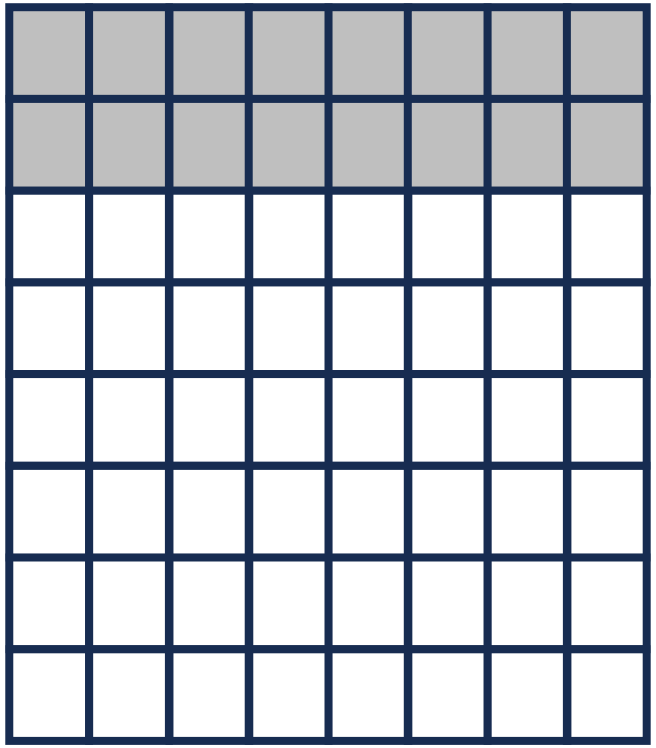
Flash Attention on Attention Mask

M



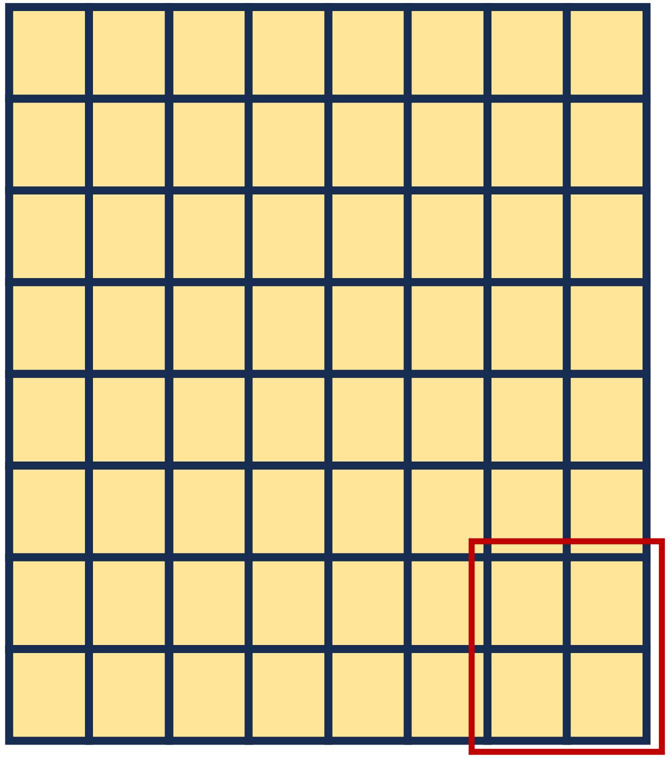
$B_c = B_r = 2$

QK^T/Y



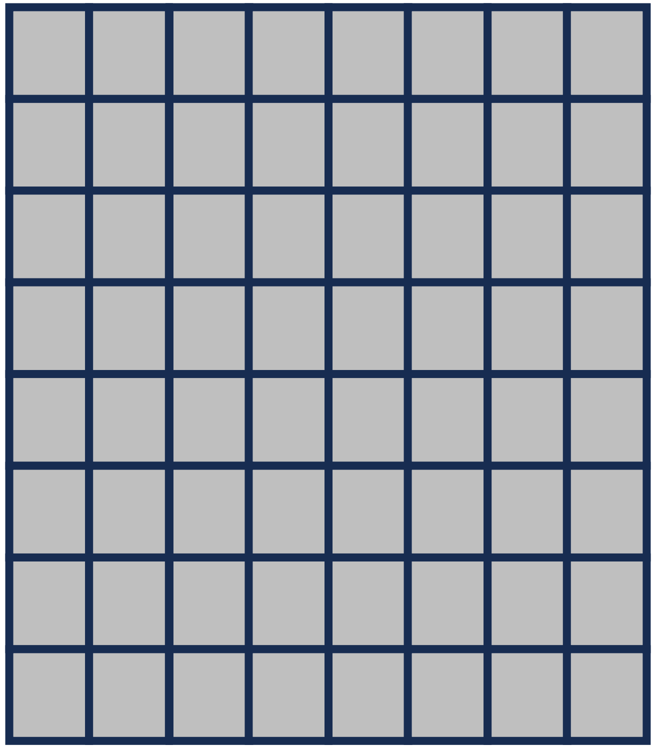
Flash Attention on Attention Mask

M



$B_c = B_r = 2$

QK^T/Y



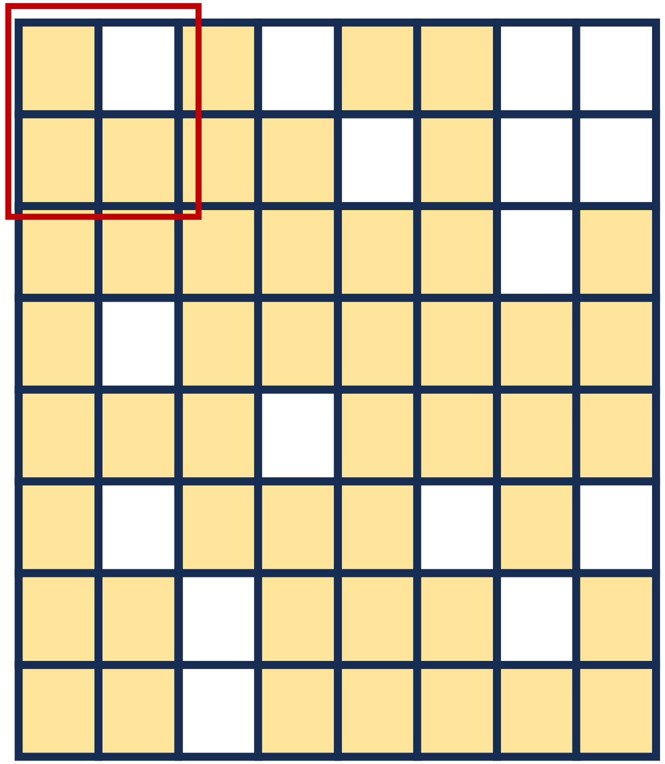
Flash Attention on Sparse Attention Mask

- Pretty much the same thing happens

- If a block has at least one non-zero element, Flashattention ignores the mask and computes the entire block

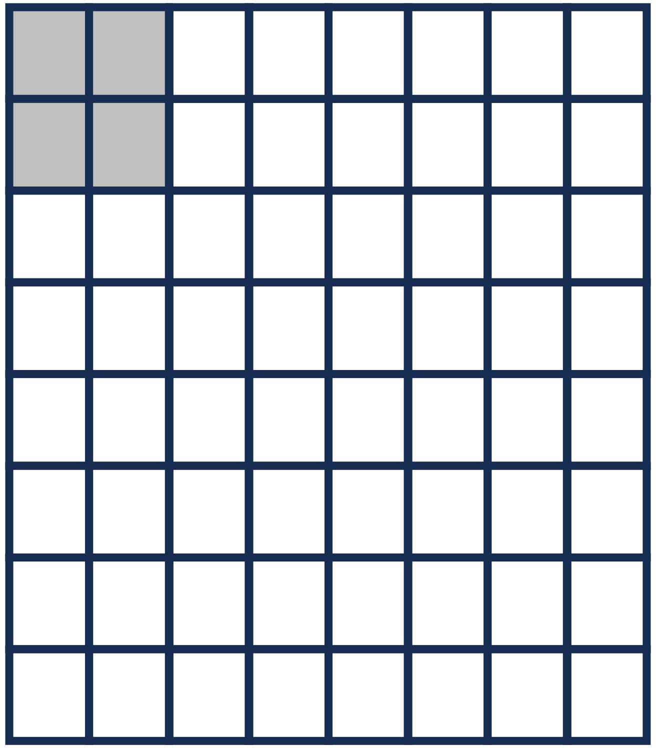
Flash Attention on Attention Mask

M



$B_c = B_r = 2$

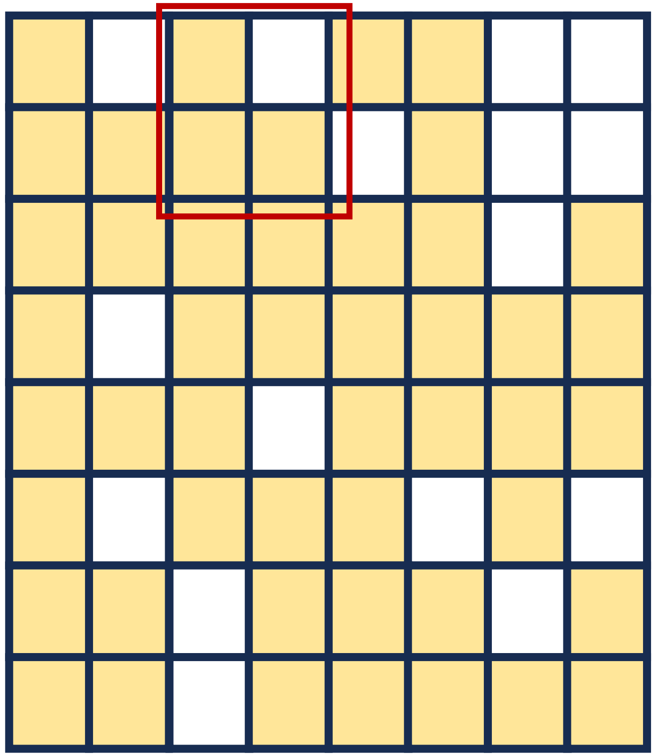
QK^T



- Flashattention will ignore the mask
- Notice how all the elements are getting calculated

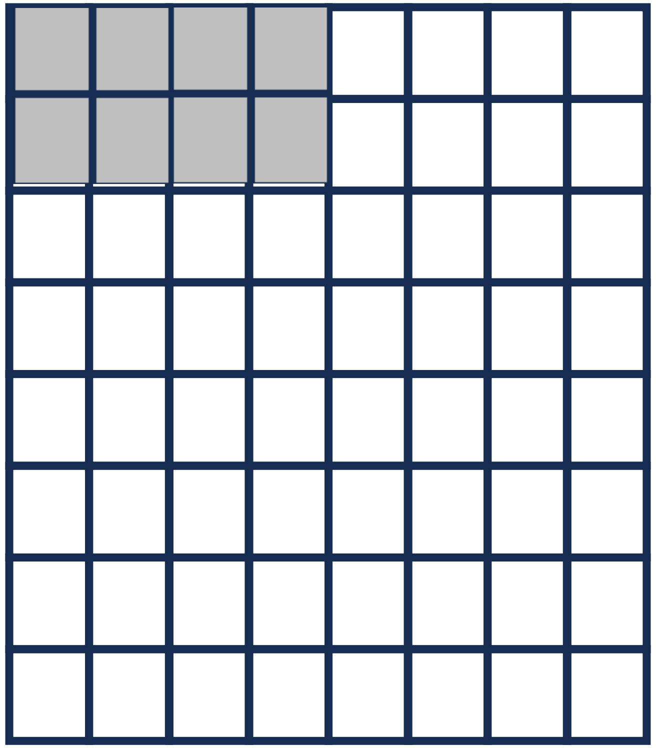
Flash Attention on Attention Mask

M



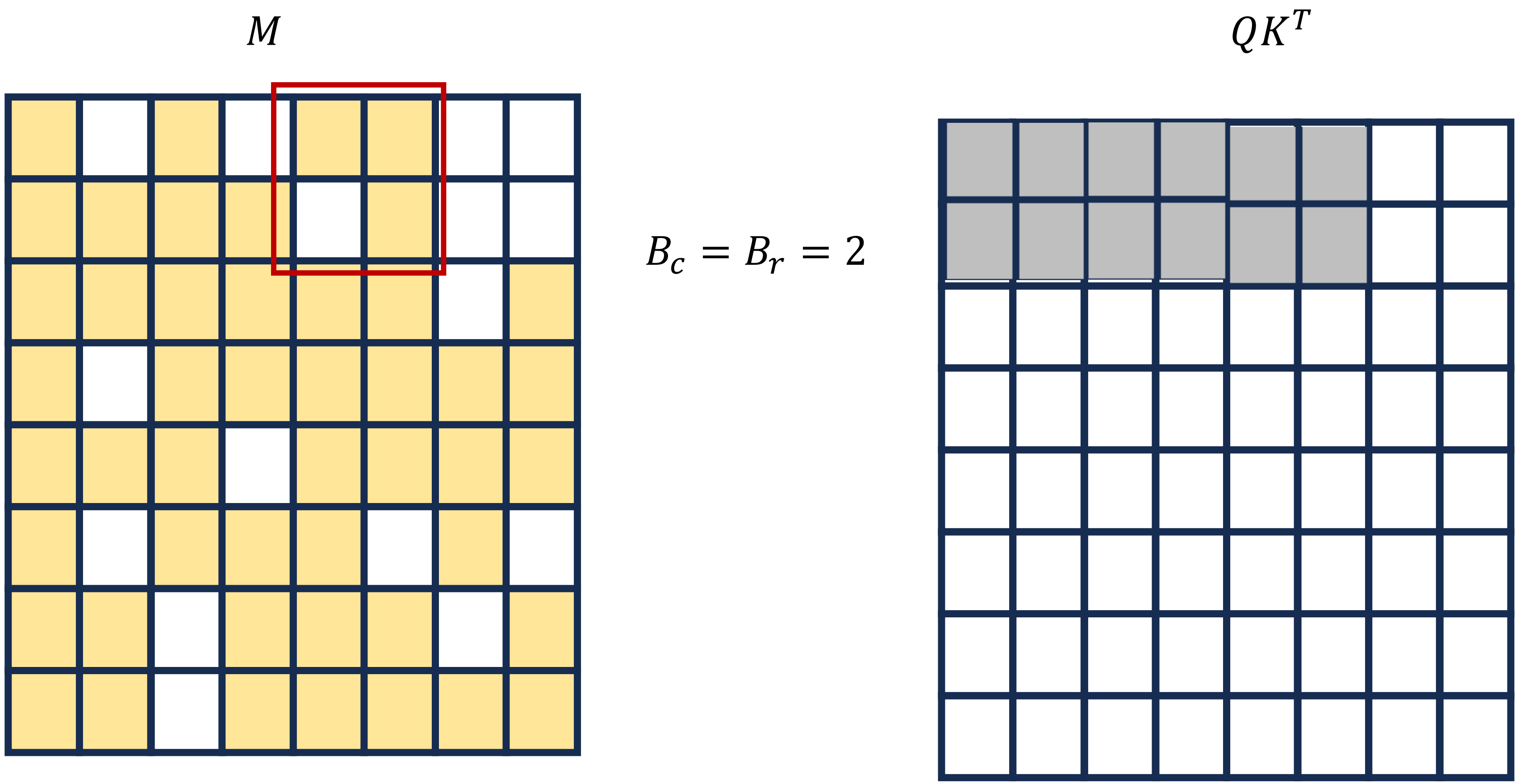
$B_c = B_r = 2$

QK^T



- Flashattention will ignore the mask
- Notice how all the elements are getting calculated

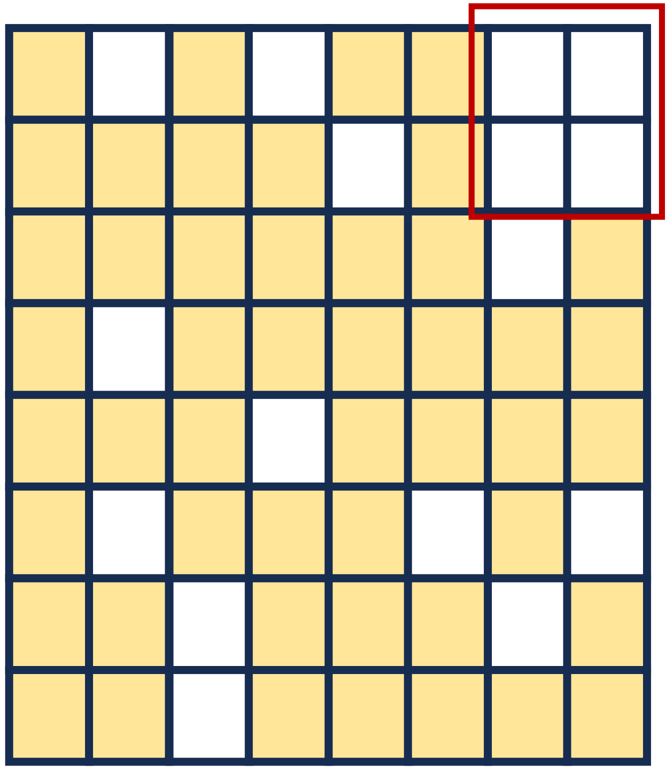
Flash Attention on Attention Mask



- Flashattention will ignore the mask
- Notice how all the elements are getting calculated

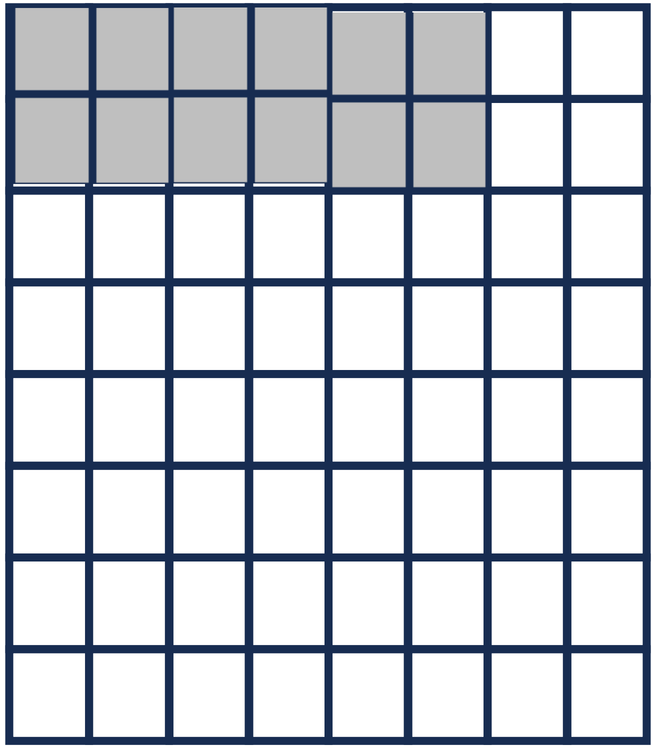
Flash Attention on Attention Mask

M



$B_c = B_r = 2$

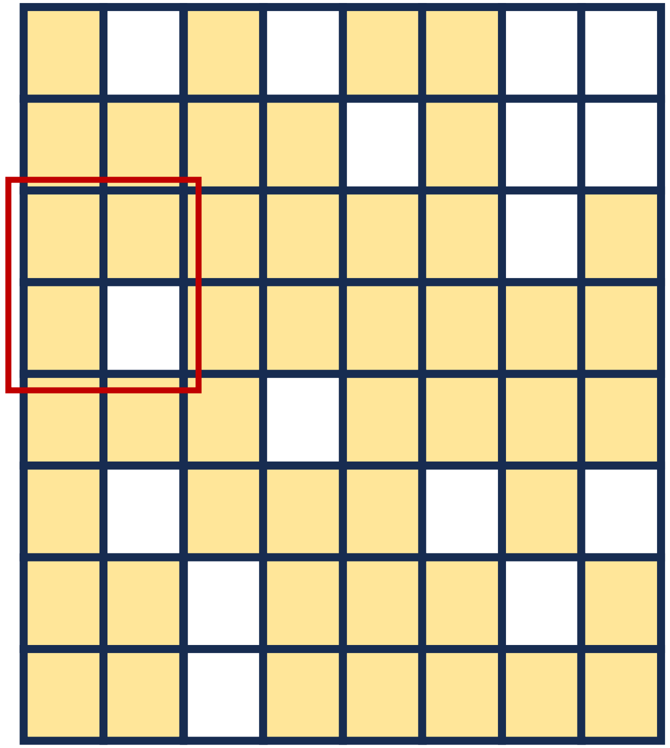
QK^T



- Flashattention will ignore the mask
- Notice how all the elements are getting calculated
- **If a block is fully empty, Flashattention will ignore it**

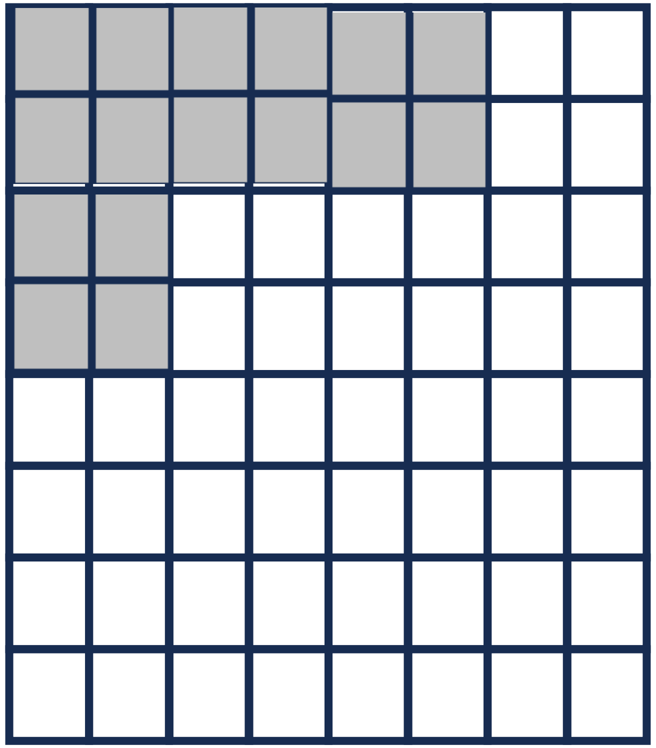
Flash Attention on Attention Mask

M



$$B_c = B_r = 2$$

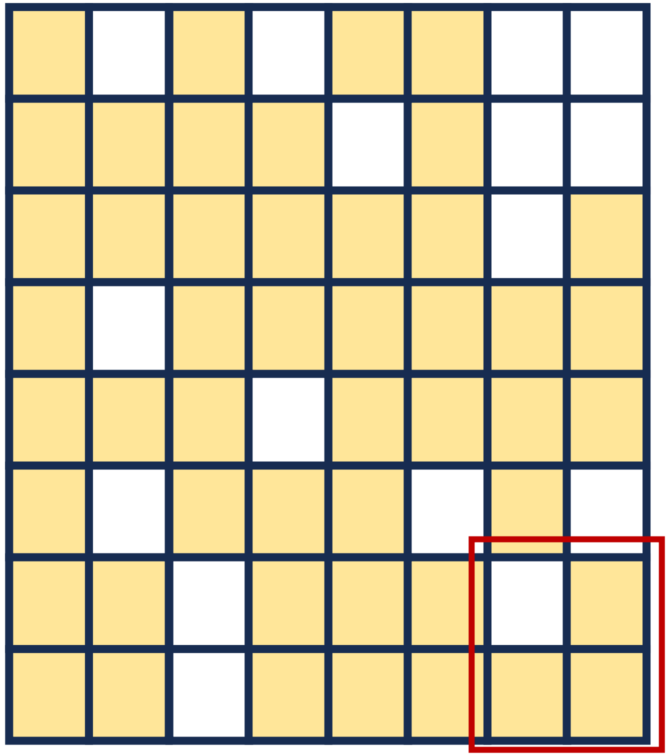
QK^T



- Flashattention will ignore the mask
- Notice how all the elements are getting calculated

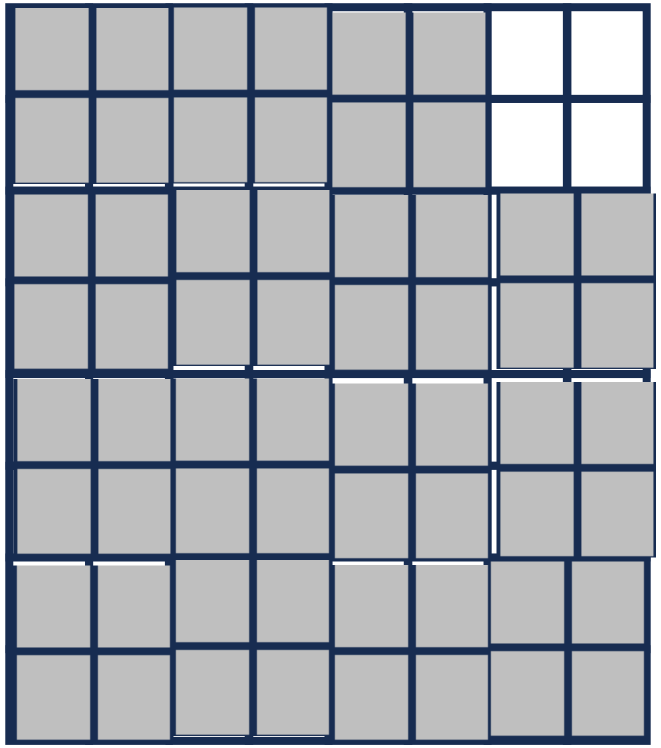
Flash Attention on Attention Mask

M



$B_c = B_r = 2$

QK^T



- Flashattention will ignore the mask
- Notice how all the elements are getting calculated

Flash Attention on Attention Mask

- Total Memory Needed?

Flash Attention on Attention Mask

- Total Memory Needed? Still $\mathrm{O}(M)$, cache size
- Total Computations?

Flash Attention on Attention Mask

- Total Memory Needed? Still $\mathrm{O}(M)$, cache size
- Total Computations? Still $O(L^2d)$ in worst case
- Performs a lot of non-useful computations

How to Improve Performance?

- ???

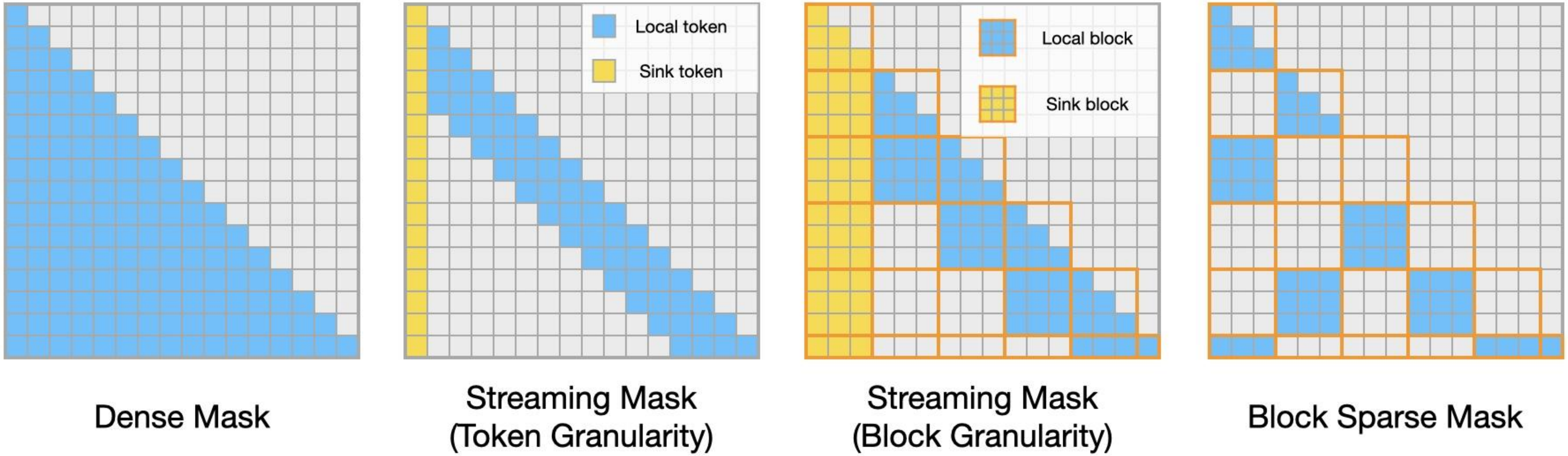
How to Improve Performance?

- Idea #1: Design masks that have block like structure
  - Non-zeros are packed together to get dense blocks on which Flashattention performs the computations
- Idea #2: Reorder the attention mask to achieve better block like structure

Today’s Lecture

- SDDMM Algorithm Design Idea
  - You will implement the algorithm in WA3
- Using Flash attention on sparse masks
  - Block Sparsity
- Algorithms to reduce the number of blocks in block sparsity

#1 Design masks that have block like structure



One Example: https://github.com/mit-han-lab/Block-Sparse-Attention
We will not discuss how to build these.

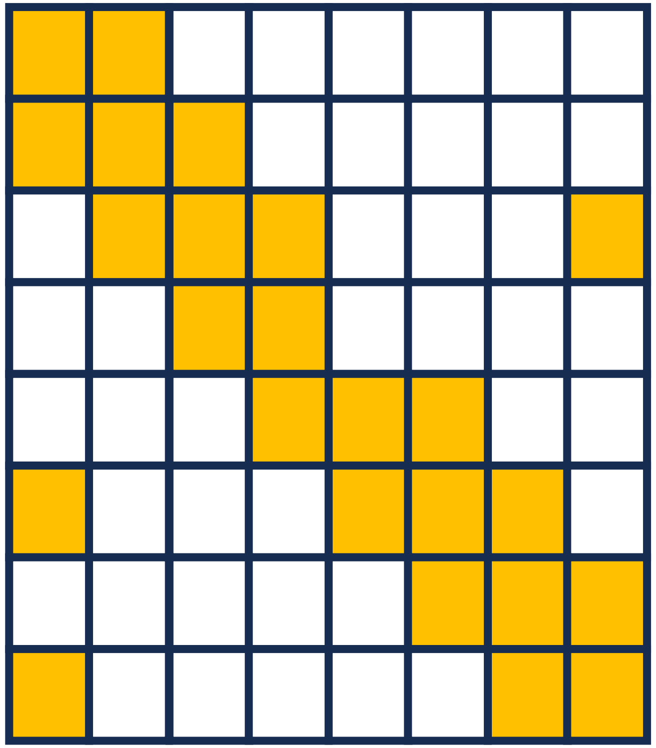
#2: Reorder Attention Masks

- Resources:
  - Sharma, Agniv, and Jonas Geiping. "Efficiently Dispatching Flash Attention For Partially Filled Attention Masks." *arXiv preprint arXiv:2409.15097* (2024).

- Other Resource: “Exploring Binary Block Masking with Hypergraph-Based Token Reordering for GPU-Efficient Attention,” Kutay Tasci, Nathaniel Tomczak, Sanmukh Kuppannagari (Under review)
  - (Our technique is much better than the above, but still under review so unavailable for you to read. ;-))

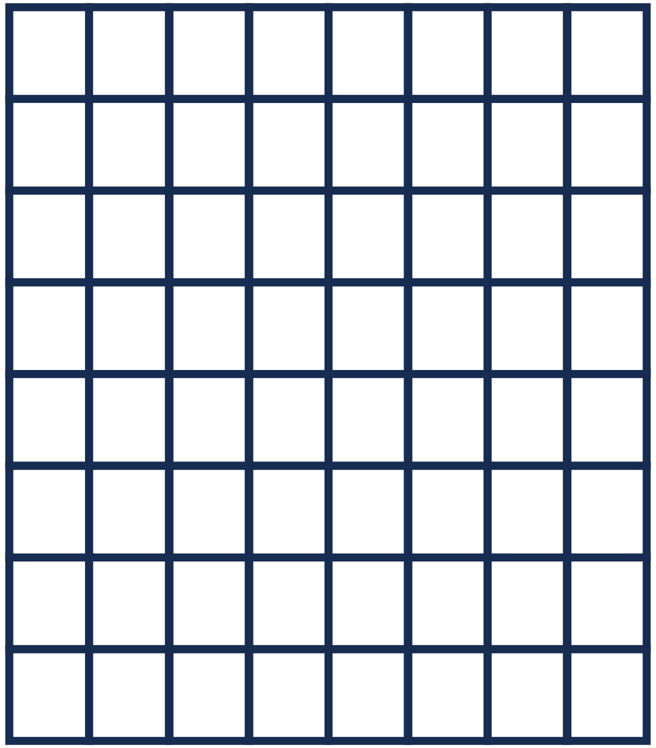
Introducing Sparsity in Flashattention

M



$B_c = B_r = 2$

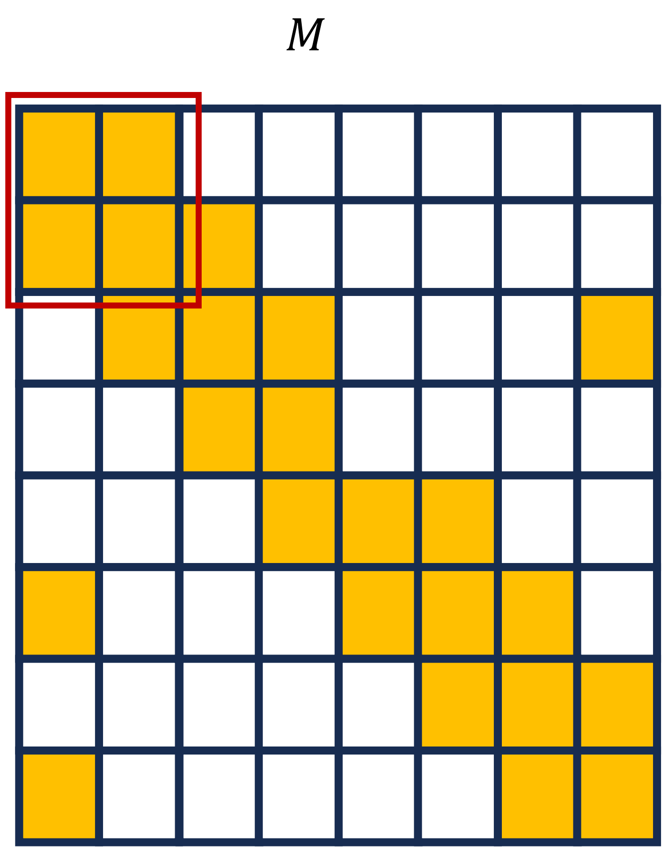
QK^T



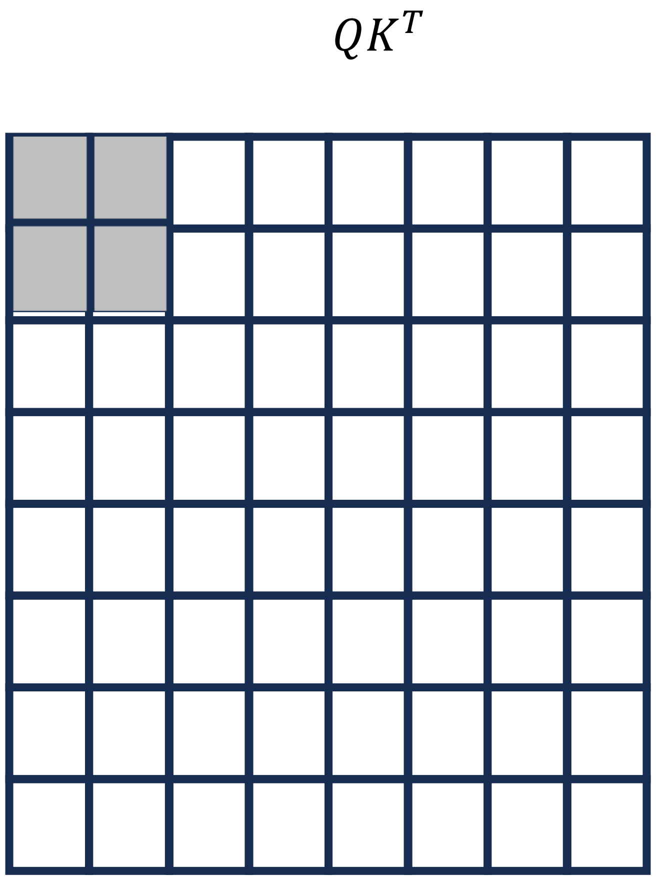
#B =0

Lets calculate the number of blocks computed by FlashAttention

Introducing Sparsity in Flashattention



$B_c = B_r = 2$

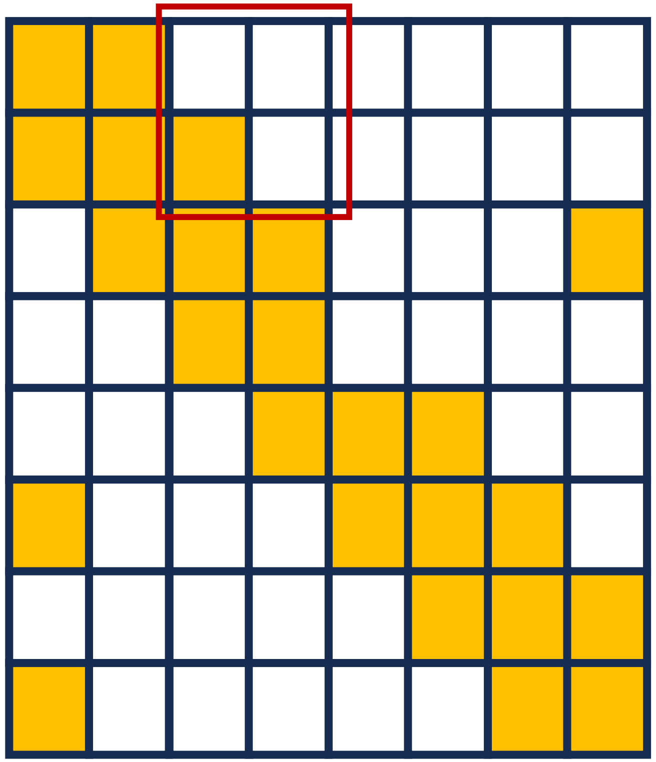


#B =1

Lets calculate the number of blocks computed by FlashAttention

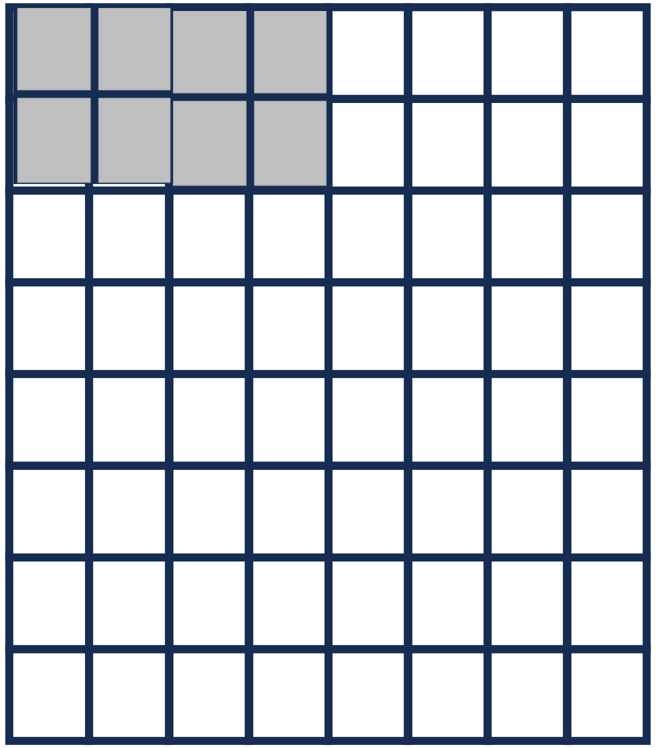
Introducing Sparsity in Flashattention

M



$B_c = B_r = 2$

QK^T

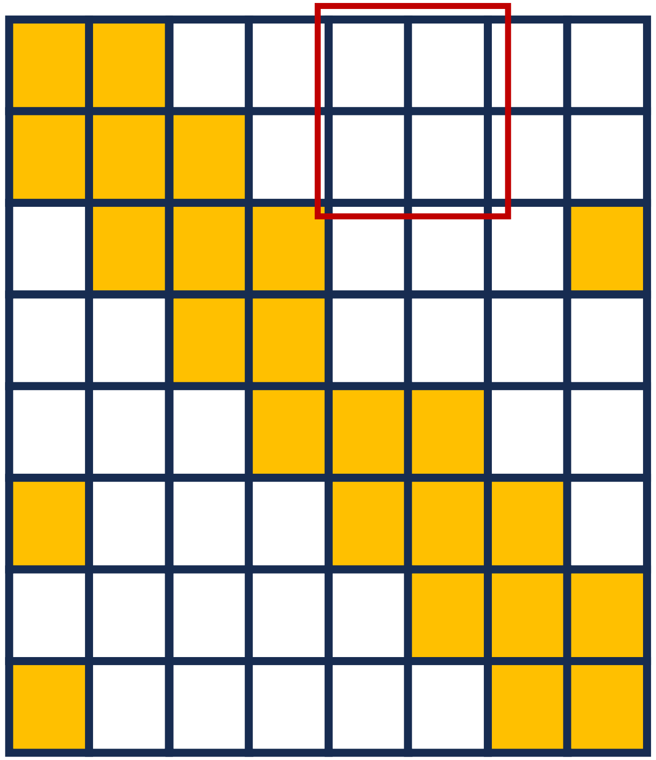


#B = 2

Lets calculate the number of blocks computed by FlashAttention

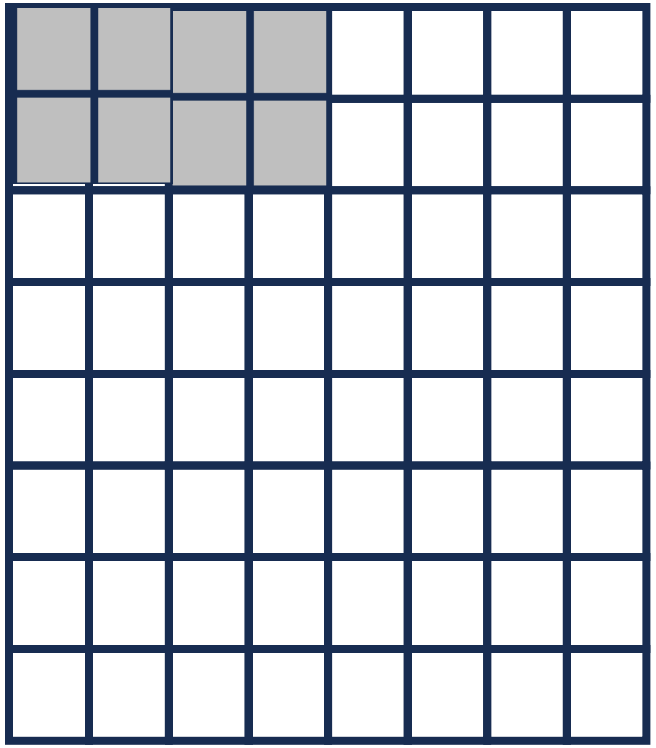
Introducing Sparsity in Flashattention

M



$B_c = B_r = 2$

QK^T

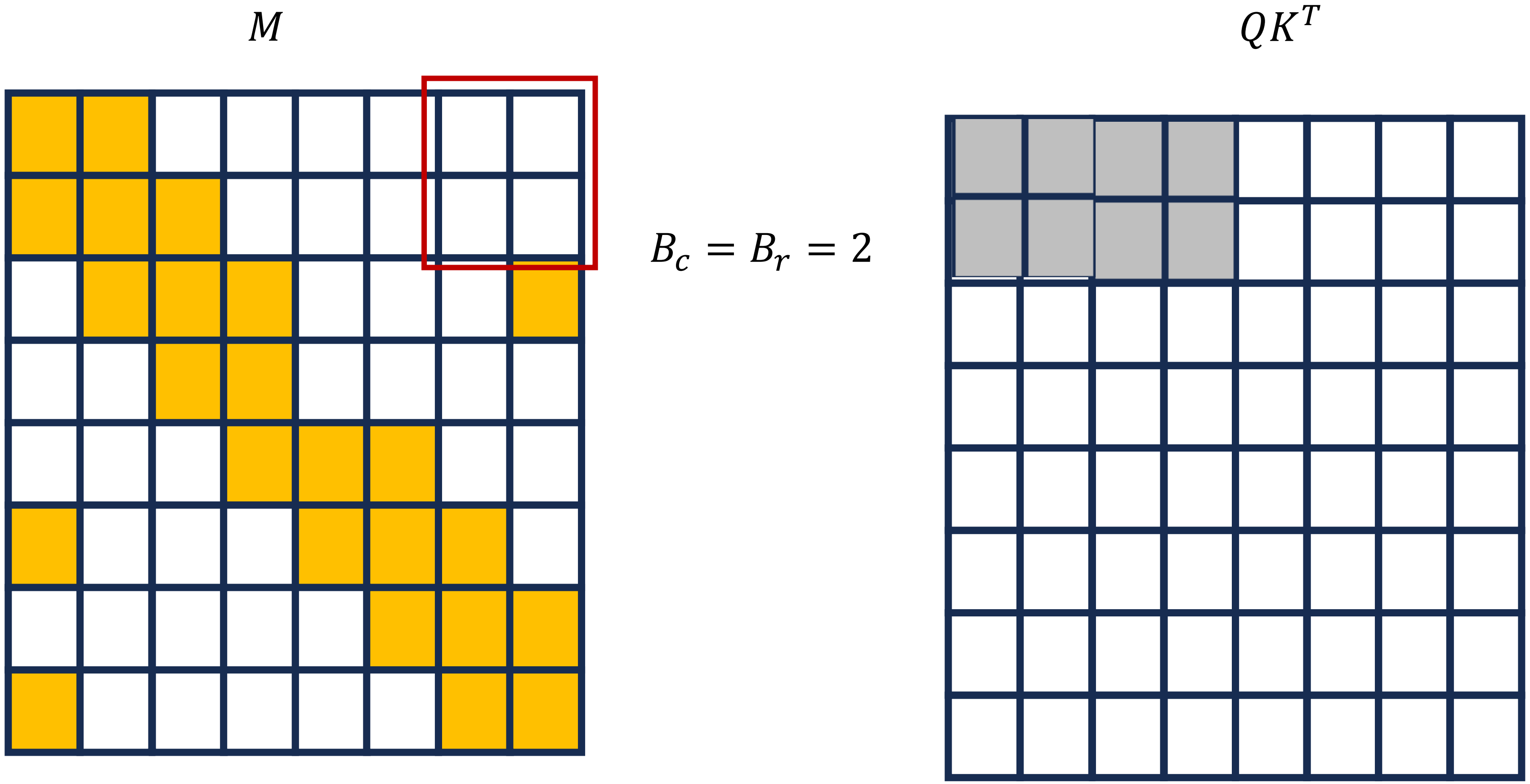


Do not process

#B = 2

Lets calculate the number of blocks computed by FlashAttention

Introducing Sparsity in Flashattention



$B_c = B_r = 2$

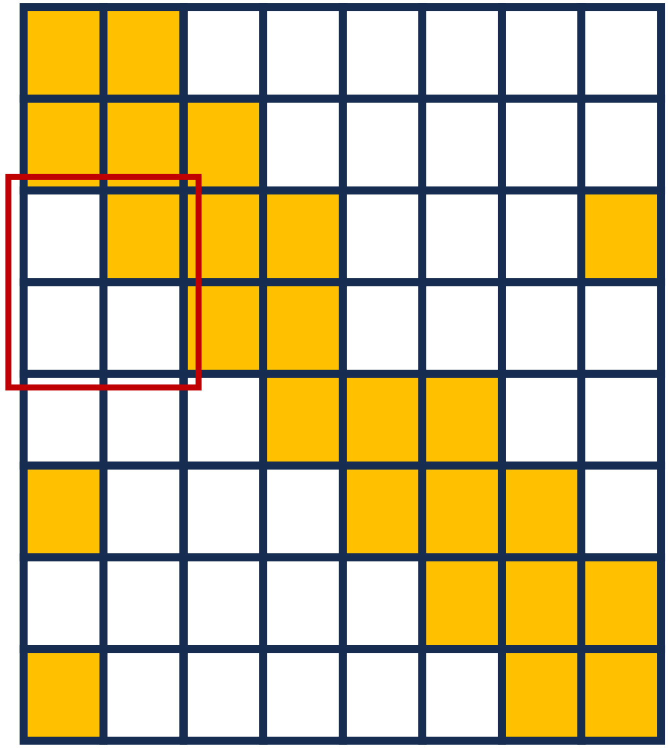
Do not process

#B = 2

Lets calculate the number of blocks computed by FlashAttention

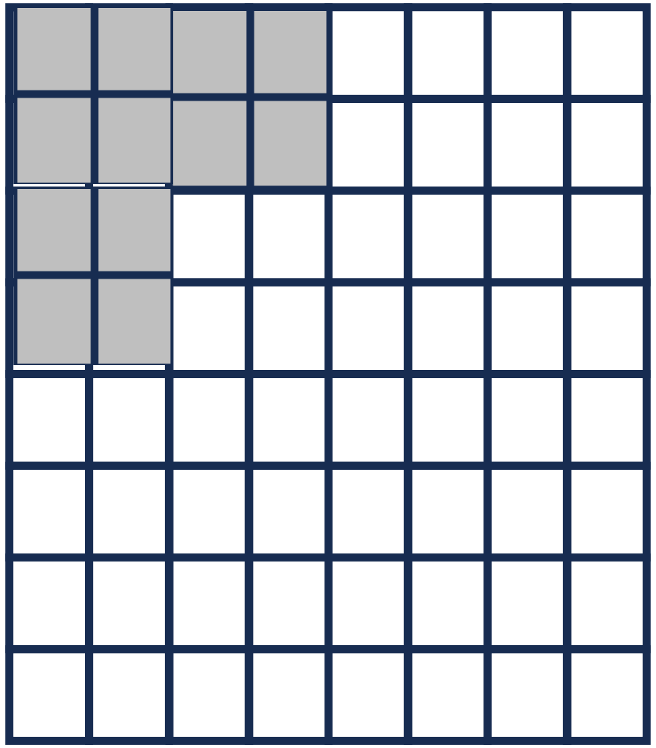
Introducing Sparsity in Flashattention

M



$B_c = B_r = 2$

QK^T

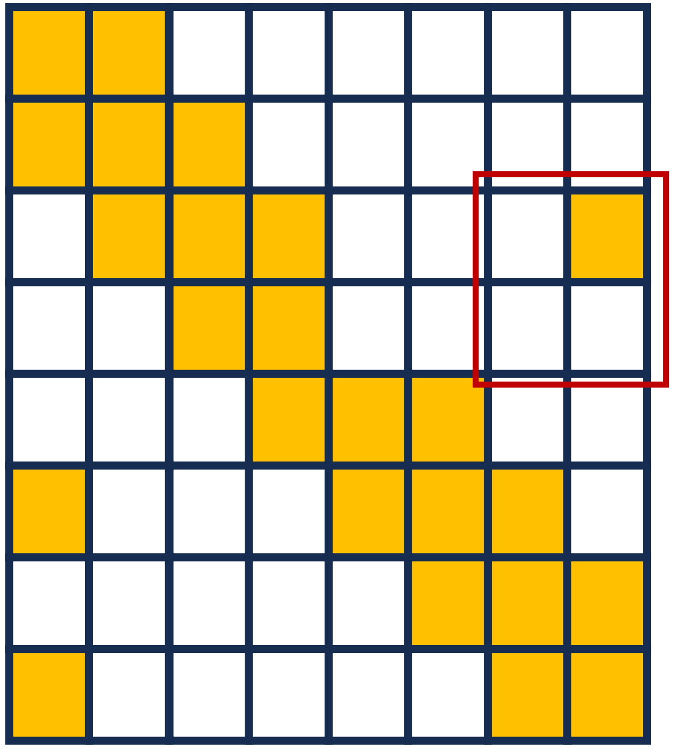


#B = 3

Lets calculate the number of blocks computed by FlashAttention

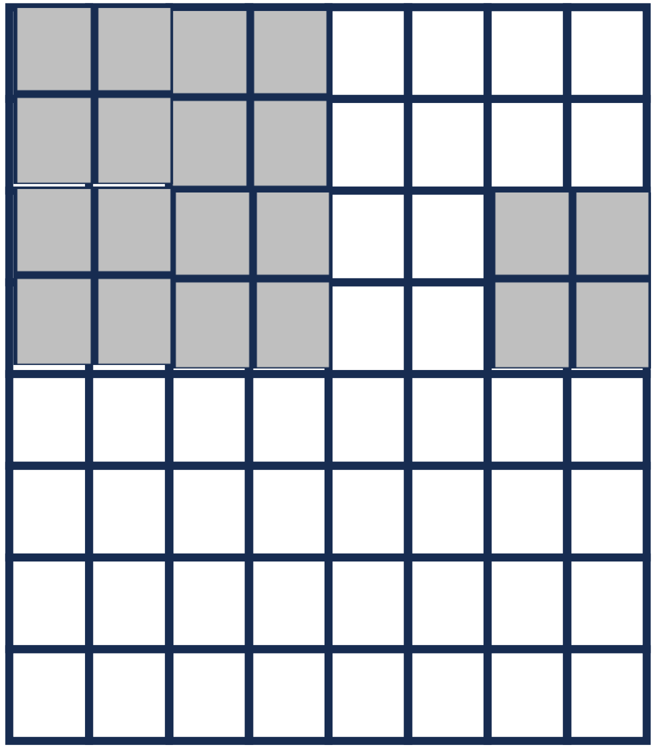
Introducing Sparsity in Flashattention

M



$B_c = B_r = 2$

QK^T

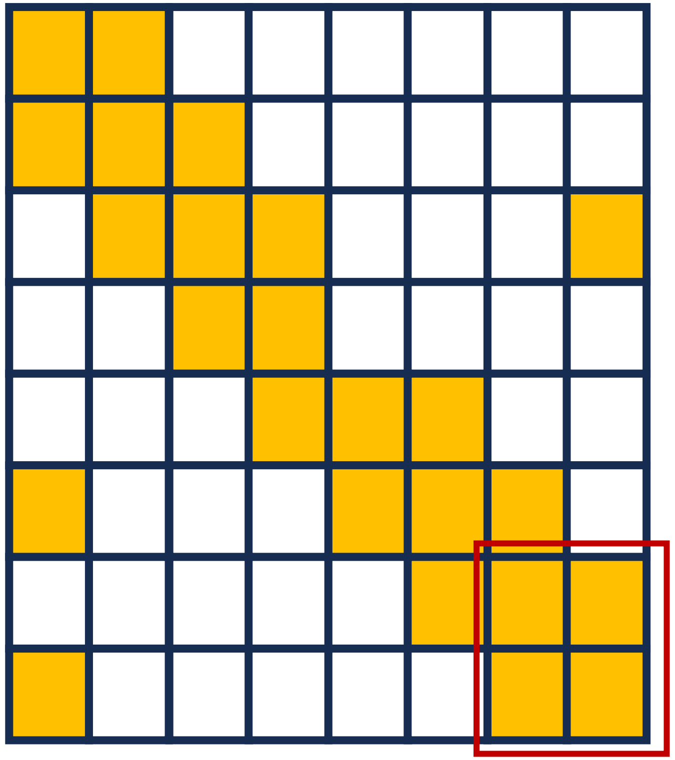


#B = 5

Lets calculate the number of blocks computed by FlashAttention

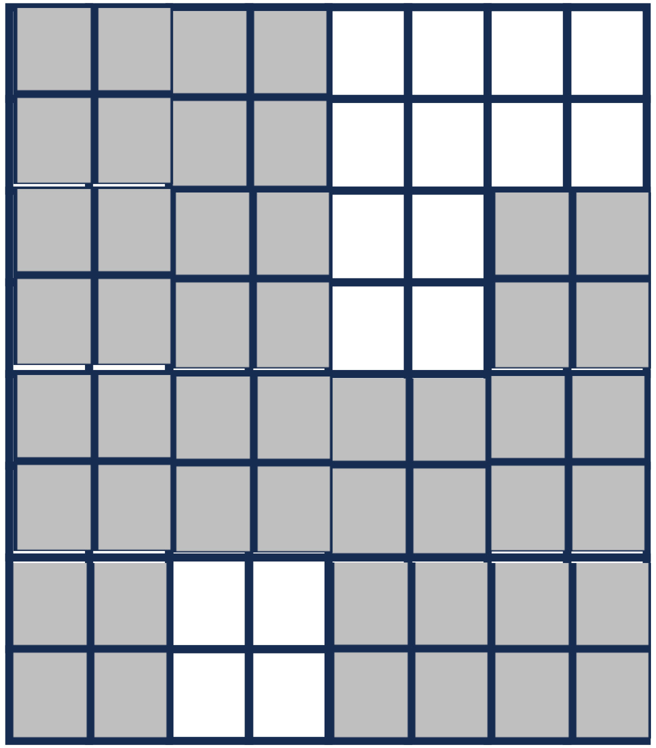
Introducing Sparsity in Flashattention

M



$B_c = B_r = 2$

QK^T

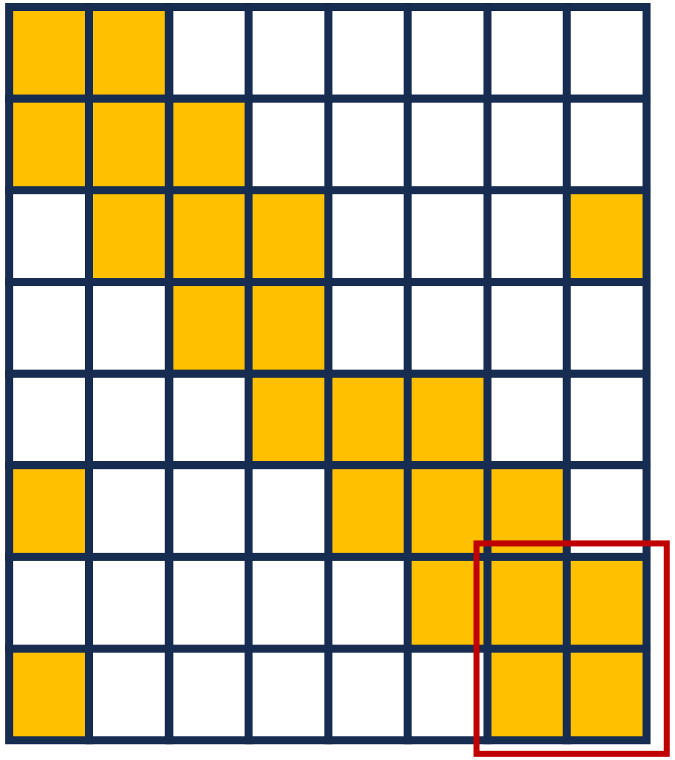


#B = ??

Lets calculate the number of blocks computed by FlashAttention

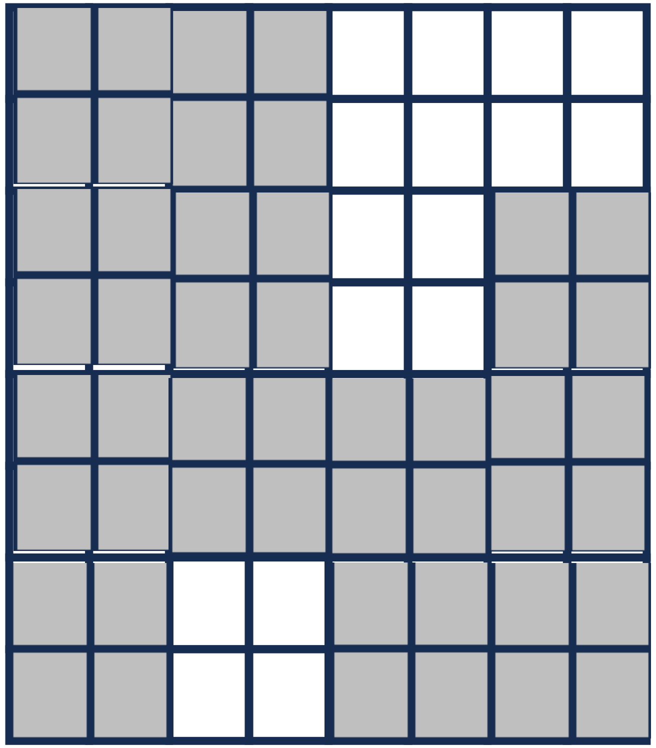
Introducing Sparsity in Flashattention

M



$B_c = B_r = 2$

QK^T

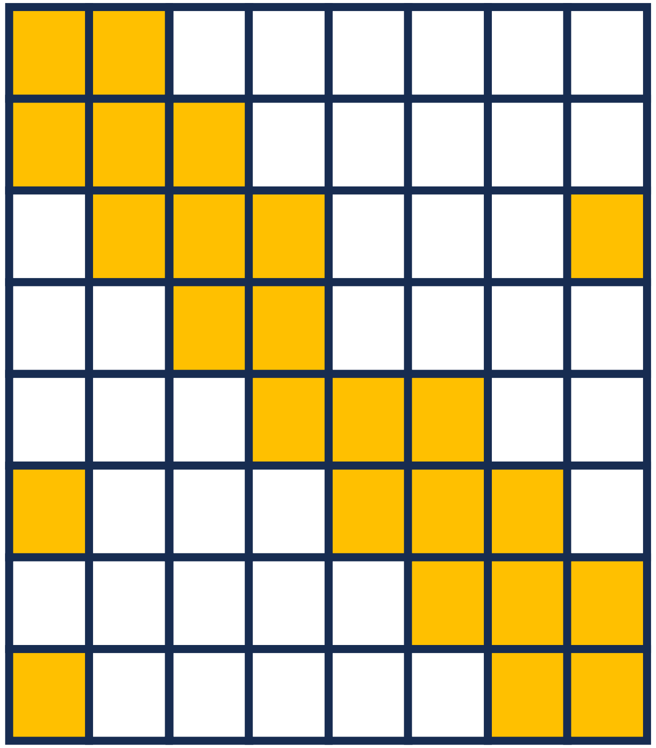


#B = 12

- Significant reduction in computations can be obtained if the number of blocks with non-zeros are less

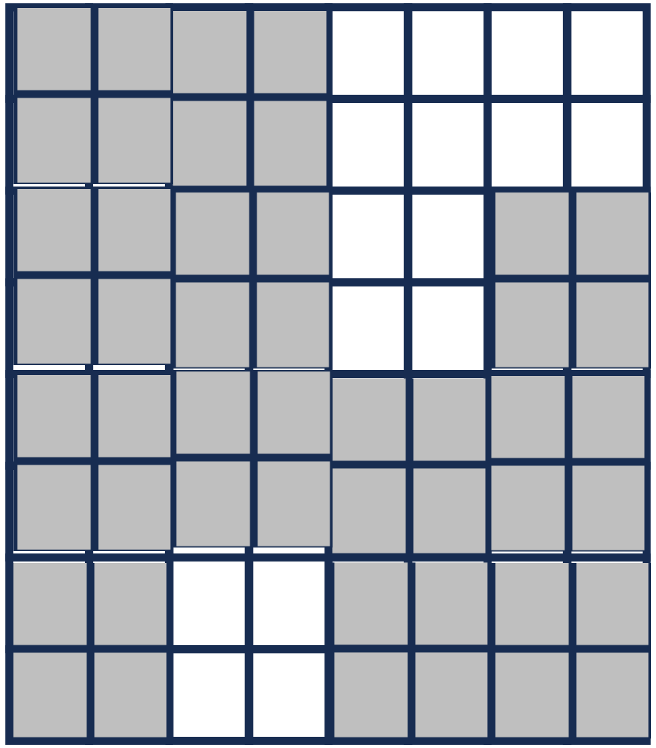
Introducing Sparsity in Flashattention

M



$B_c = B_r = 2$

QK^T

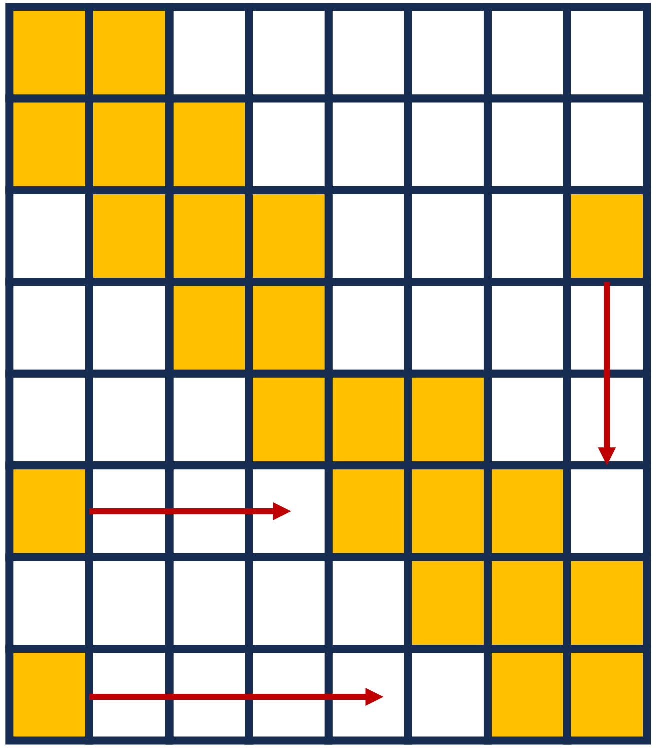


#B = 12

- What could have led to an even better reduction in computation?

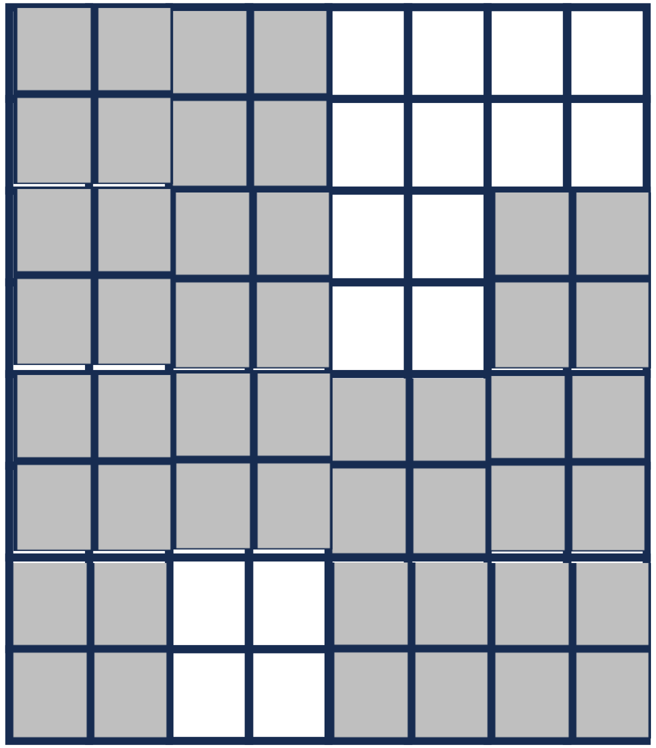
Introducing Sparsity in Flashattention

M



$B_c = B_r = 2$

QK^T

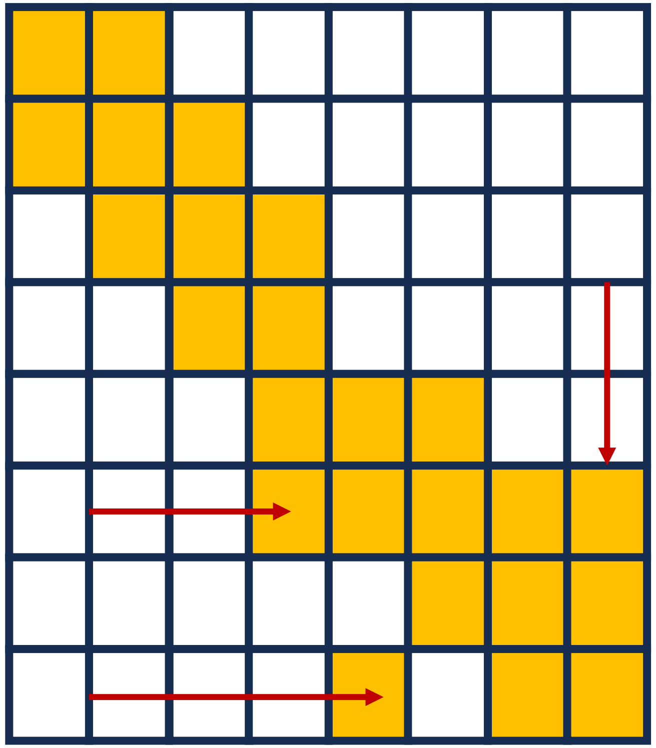


#B = 12

- What could have led to an even better reduction in computation?

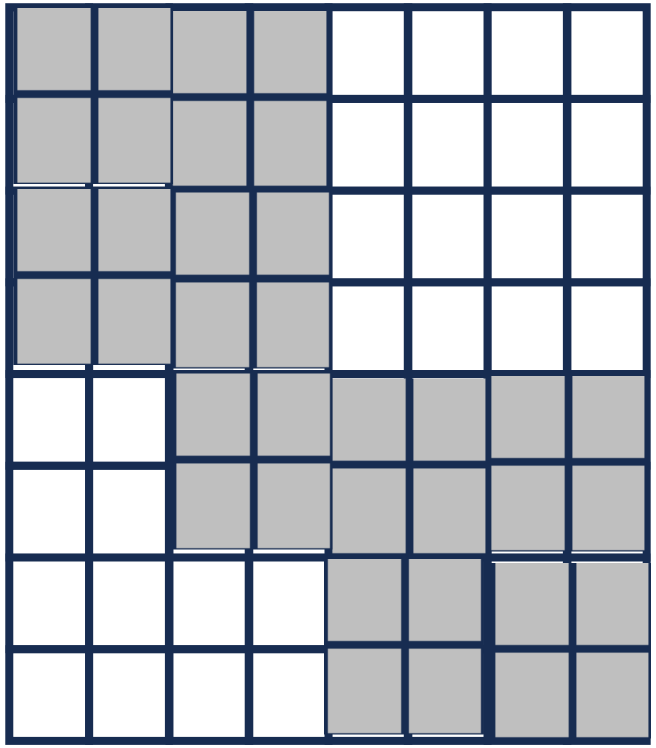
Introducing Sparsity in Flashattention

M



$B_c = B_r = 2$

QK^T

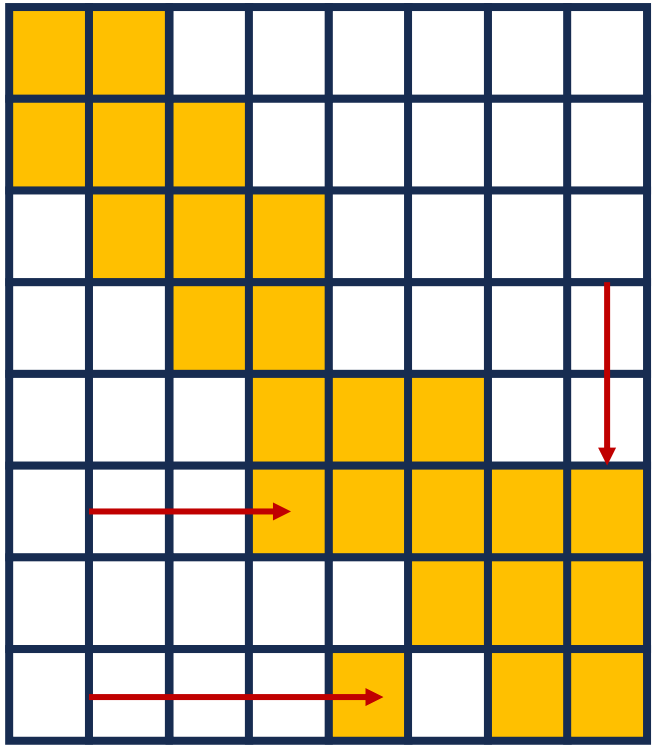


#B = 9

- What could have led to an even better reduction in computation? If non-zero elements are packed closed to the diagonal

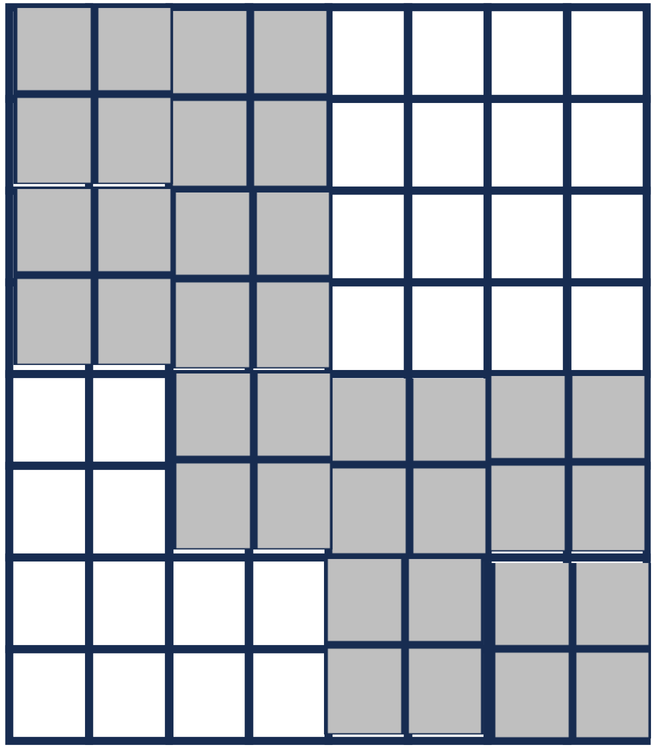
Introducing Sparsity in Flashattention

M



$B_c = B_r = 2$

QK^T



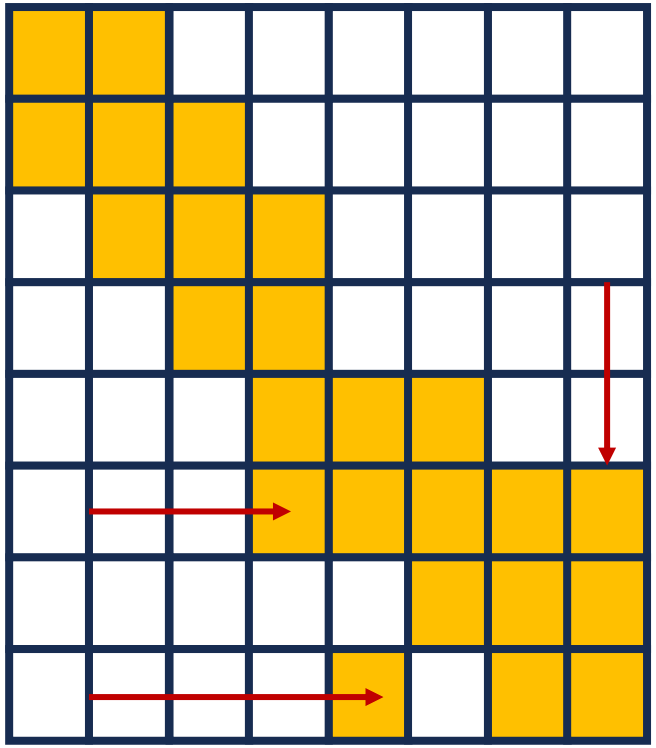
#B = 9

- However, we cannot simply take a non-zero and move it.

CASE SCHOOL
OF ENGINEERING
CASE WESTERN RESERVE
UNIVERSITY

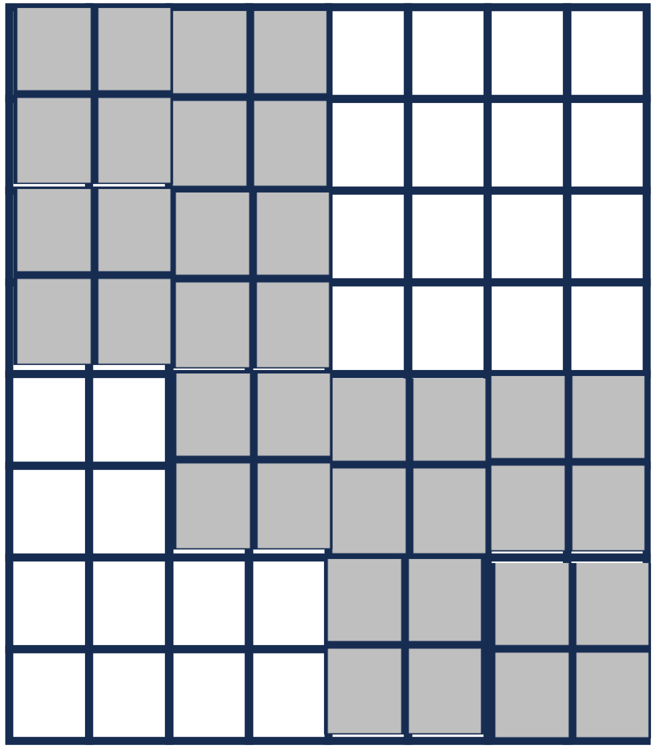
Introducing Sparsity in Flashattention

M



$B_c = B_r = 2$

QK^T



#B = 9

- However, we cannot simply take a non-zero and move it.
- Solution: Matrix Reordering

Matrix Reordering

- A key preprocessing step in scientific computations

- Given a sparse matrix, reorder rows/columns with the following objective:
  - Minimize bandwidth: Longest distance across row from the diagonal elements
  - Fill in: Number of entries that change from 0 to non-zero, typically after matrix factorization

- This link provides a good visualization: https://www.mathworks.com/help/matlab/math/sparse-matrix-reordering.html

Matrix Reordering

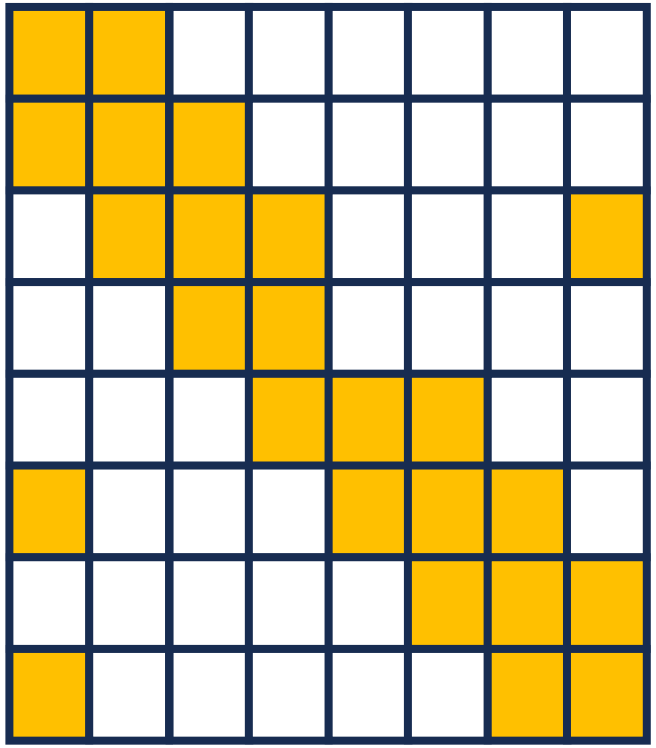
- In our case, the sparse matrix is an adjacency matrix of a graph
  - For undirected graphs: adjacency matrix is symmetric
  - For masks that are non-causal, attention masks are symmetric
- And we do not need to do any factorization, so we don’t care about Fill in
- So, the problem boils down to: Relabel the vertices of the graph to minimize bandwidth

Bandwidth

- Definition: For a matrix M with entries denoted using i, j

- Bandwidth is the number K s.t. :

- $M_{ij} = 0$, if $|i - j| > K$



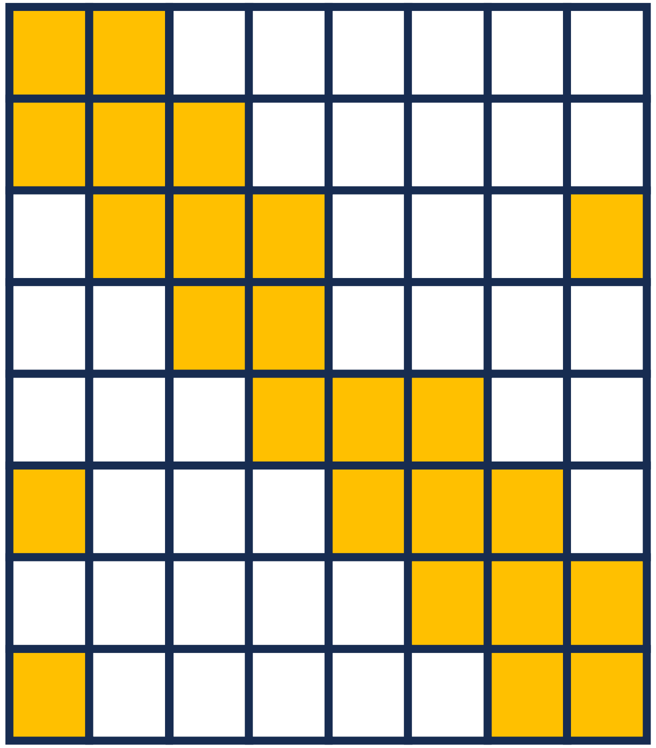
Bandwidth

- Definition: For a matrix M with entries denoted using i, j

- Bandwidth is the number K s.t. :

- $M_{ij} = 0$, if $|i - j| > K$

- Procedure:
  - For each row i, find j such that all elements outside the band $i - j, i + j$ are 0.
  - Maximum across all the rows is the bandwidth



Bandwidth

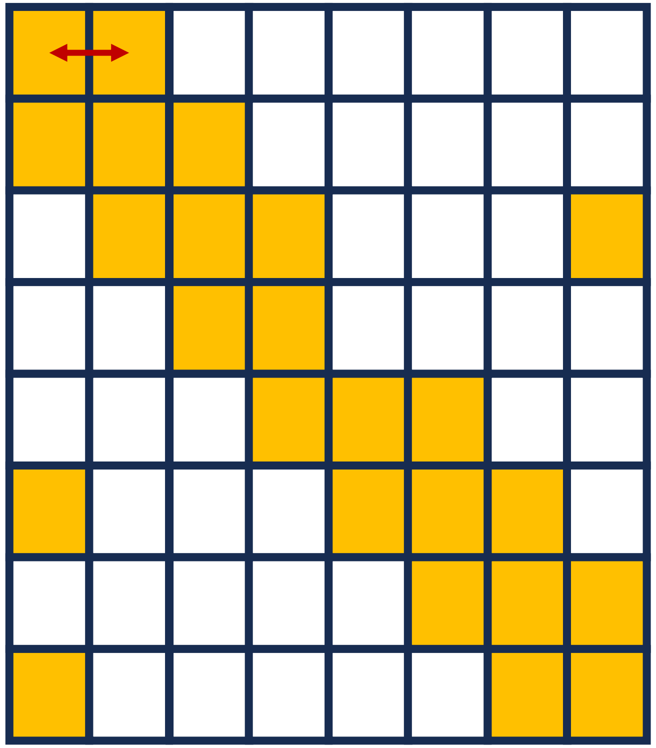
- Definition: For a matrix M with entries denoted using i, j

- Bandwidth is the number K s.t. :

- $M_{ij} = 0$, if $|i - j| > K$

- Procedure:
  - For each row i, find j such that all elements outside the band $i - j, i + j$ are 0.
  - Maximum across all the rows is the bandwidth

$i = 0$
$j = 1$
$K = 1$



Bandwidth

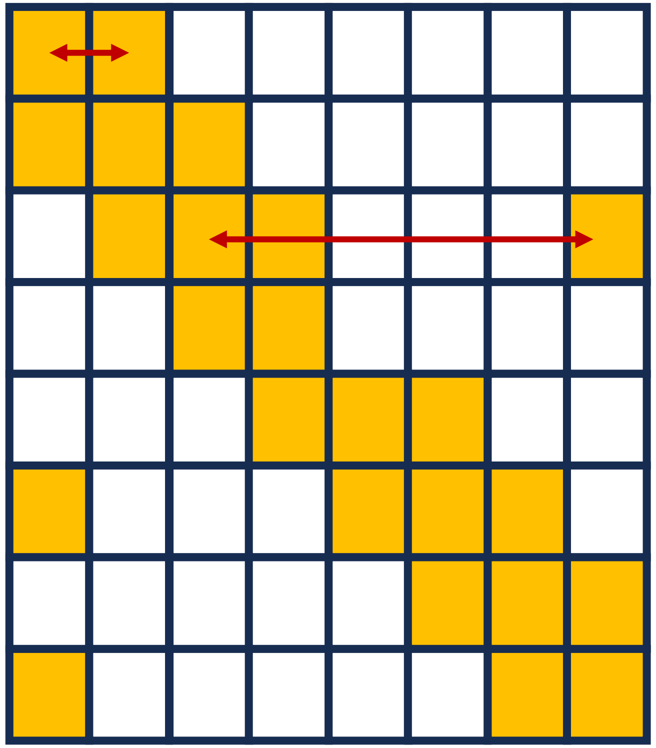
- Definition: For a matrix M with entries denoted using i, j

- Bandwidth is the number K s.t. :

- $M_{ij} = 0$, if $|i - j| > K$

- Procedure:
  - For each row i, find j such that all elements outside the band $i - j, i + j$ are 0.
  - Maximum across all the rows is the bandwidth

$i = 2$
$j = ??$
$K = ??$



Bandwidth

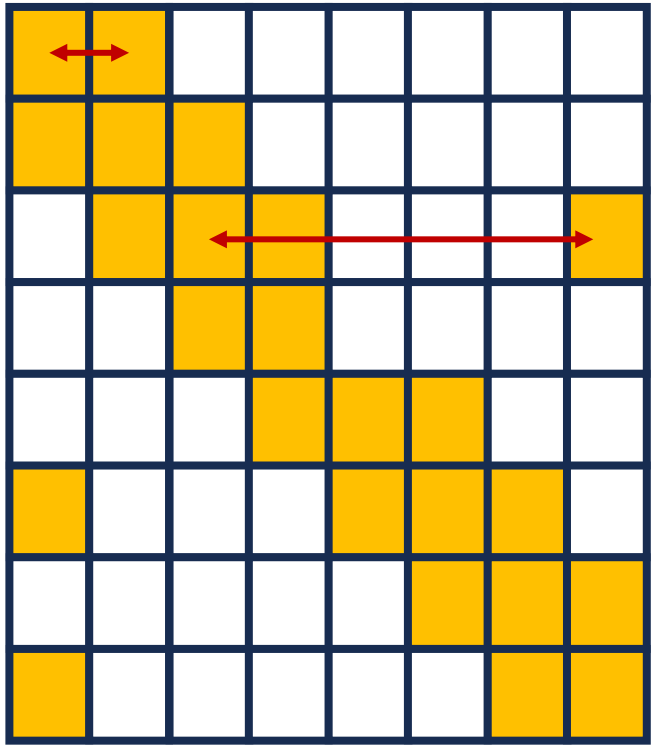
- Definition: For a matrix M with entries denoted using i, j

- Bandwidth is the number K s.t. :

- $M_{ij} = 0$, if $|i - j| > K$

- Procedure:
  - For each row i, find j such that all elements outside the band $i - j, i + j$ are 0.
  - Maximum across all the rows is the bandwidth

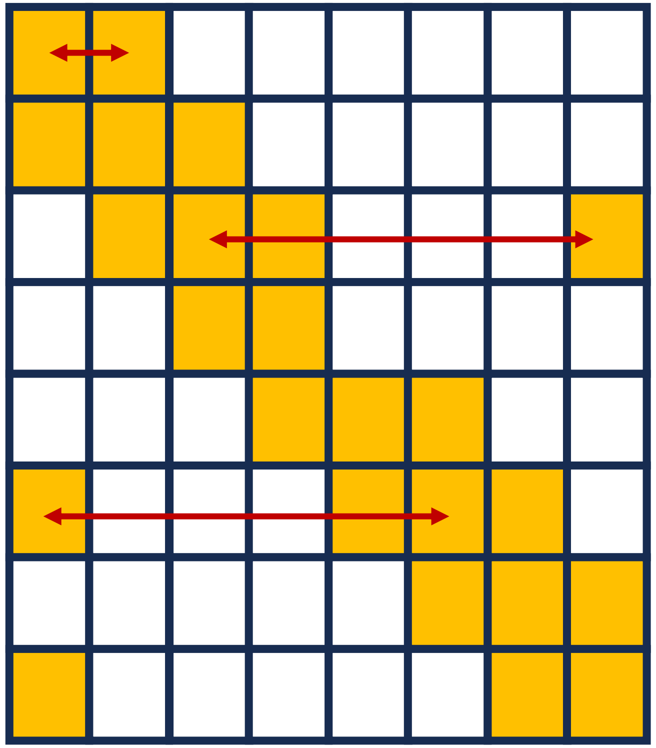
$i = 2$
$j = 7$
$K = 5$



Bandwidth

- Definition: For a matrix M with entries denoted using i, j
- Bandwidth is the number K s.t. :
- $M_{ij} = 0$, if $|i - j| > K$
- Procedure:
  - For each row i, find j such that all elements outside the band $i - j, i + j$ are 0.
  - Maximum across all the rows is the bandwidth

$i = 5$
$j = 0$
$K = 5$



Bandwidth

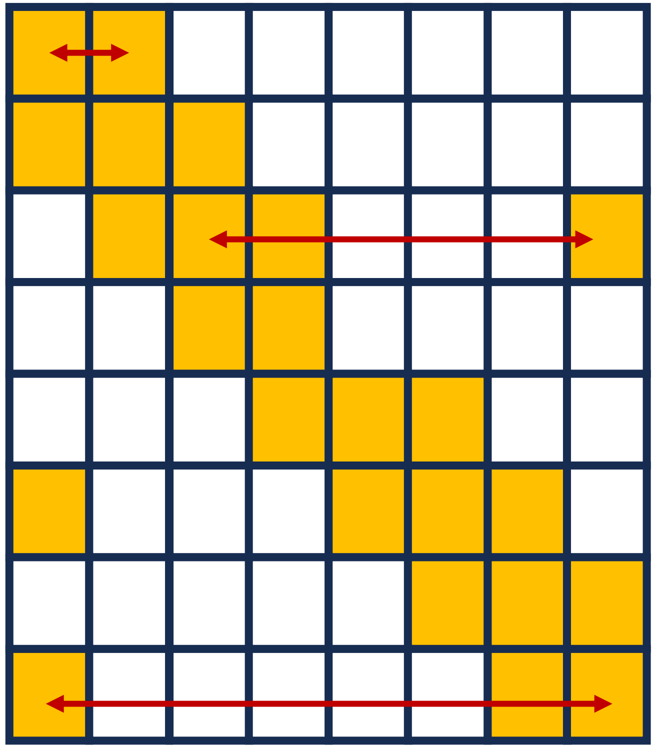
- Definition: For a matrix M with entries denoted using i, j

- Bandwidth is the number K s.t. :

- $M_{ij} = 0$, if $|i - j| > K$

- Procedure:
  - For each row i, find j such that all elements outside the band $i - j, i + j$ are 0.
  - Maximum across all the rows is the bandwidth

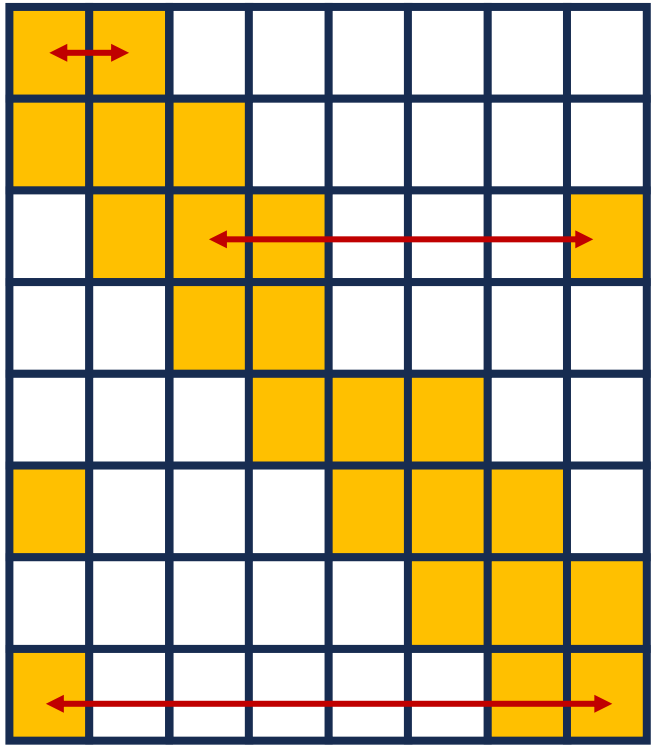
$i = 7$
$j = 0$
$K = 7$



Bandwidth

- Definition: For a matrix M with entries denoted using i, j
- Bandwidth is the number K s.t. :
- $M_{ij} = 0$, if $|i - j| > K$
- Bandwidth = 7
- Procedure:
  - For each row i, find j such that all elements outside the band $i - j, i + j$ are 0.
  - Maximum across all the rows is the bandwidth

$i = 7$
$j = 0$
$K = 7$

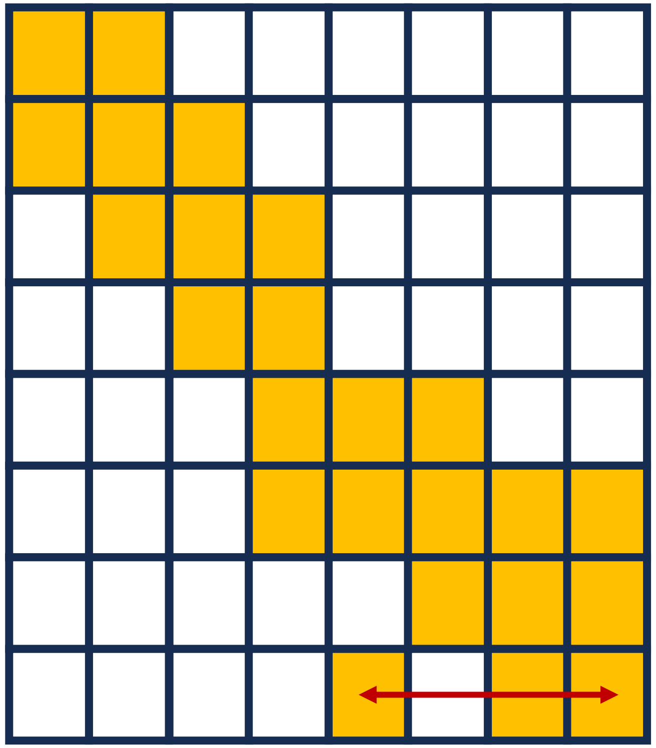


Bandwidth – After Reordering

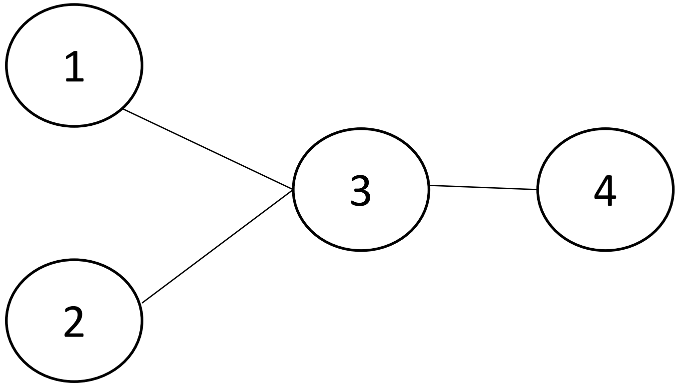
- Definition: For a matrix M with entries denoted using i, j
- Bandwidth is the number K s.t. :
- $M_{ij} = 0\,,\ \ \text{if}\ |i - j| > K$
- Bandwidth = 3

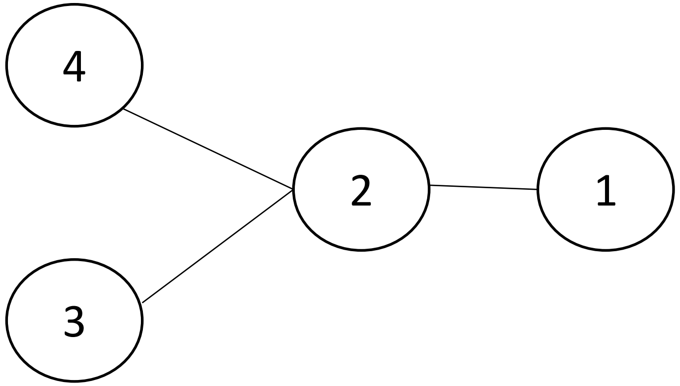
A lower bandwidth is being used as a proxy for better packing

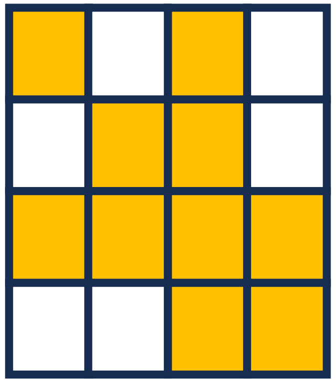
$K = 3$



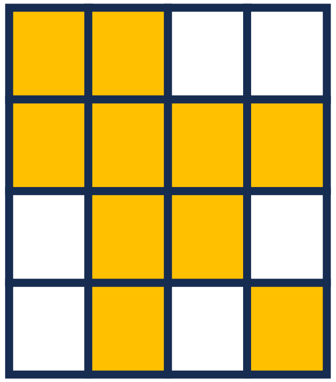
Graph Vertex Relabling



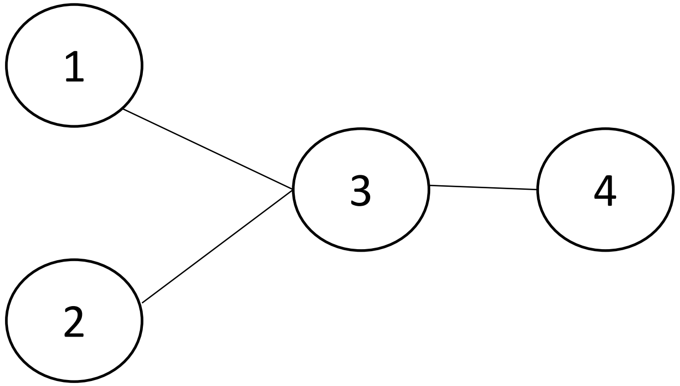


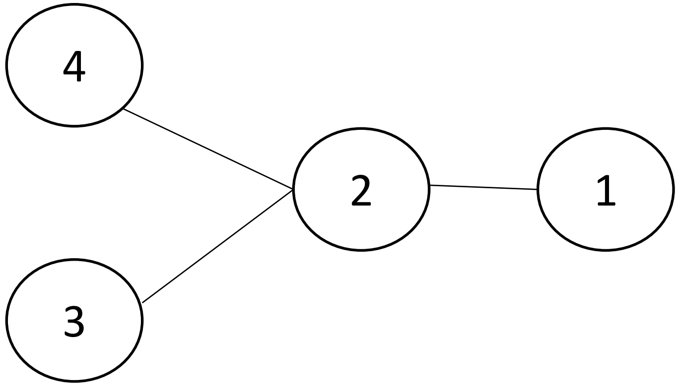


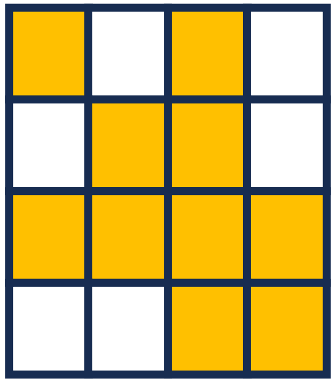
Notice how the adjacency matrix changes with the reordering of the vertices



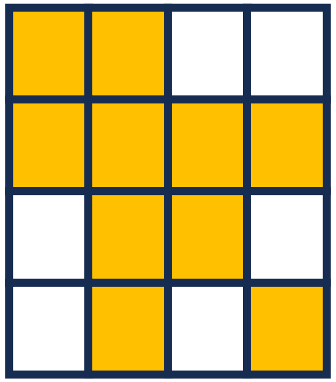
Graph Vertex Relabling



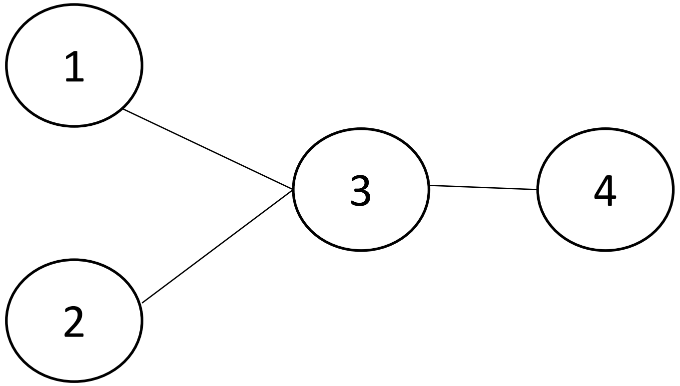


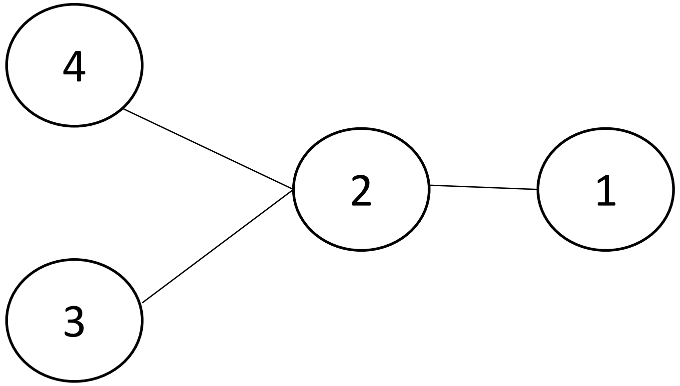


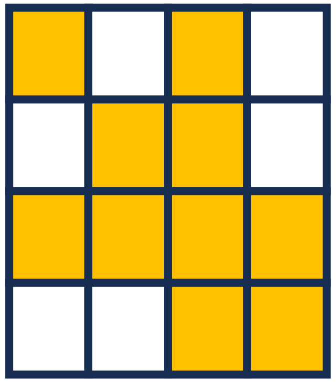
Graph Bandwidth Minimization:
Finding a re-ordering of graph to minimize the bandwidth of its corresponding adjacency matrix



Graph Vertex Relabling

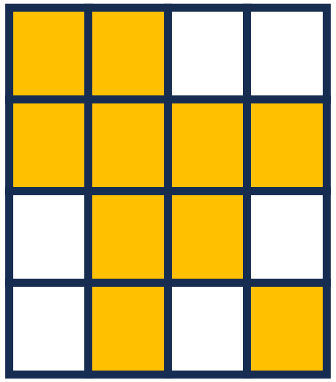




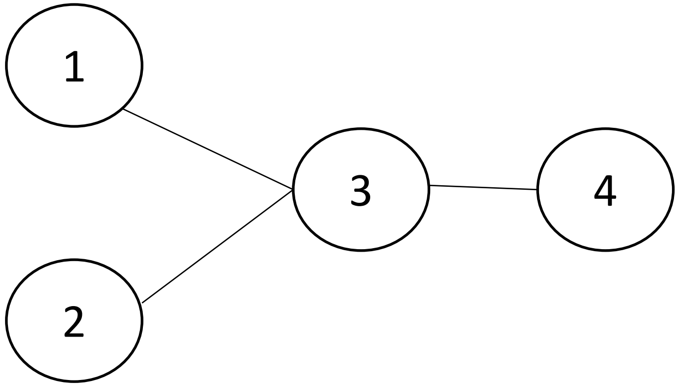


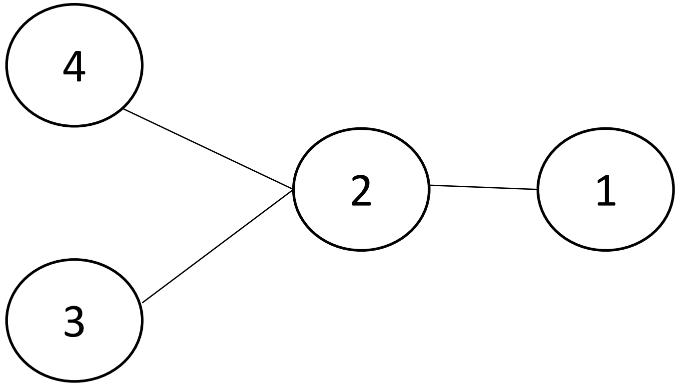
https://en.wikipedia.org/wiki/Graph_bandwidth

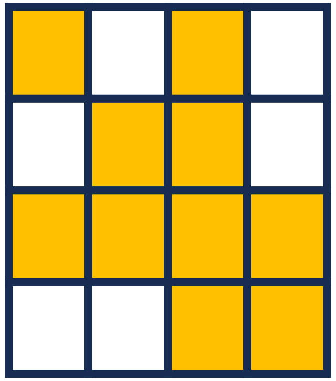
This problem is NP-hard



Graph Vertex Relabling



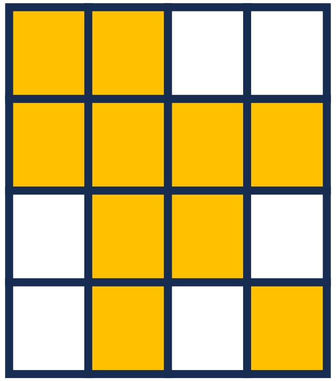




https://en.wikipedia.org/wiki/Graph_bandwidth

This problem is NP-hard

Fast heuristics exists



Cuthill–McKee algorithm (CM)

- Given a square symmetric matrix that is an adjacency matrix of a graph, relabel the vertices to obtain low bandwidth in the matrix

- Cuthill, Elizabeth, and James McKee. "Reducing the bandwidth of sparse symmetric matrices." *Proceedings of the 1969 24th national conference*. 1969. https://dl.acm.org/doi/pdf/10.1145/800195.805928

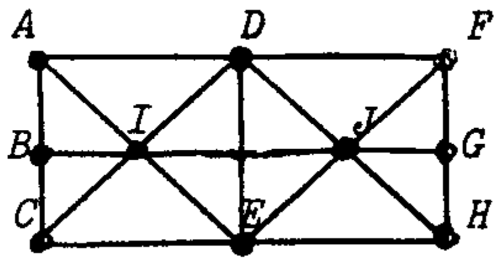
- Type written, uses fortran to discuss the algorithm. It is a fun read.

Cuthill–McKee algorithm (CM)

- Algorithm: (https://en.wikipedia.org/wiki/Cuthill%E2%80%93McKee_algorithm)

- Let $R = \{\}$
- Choose a vertex x with lowest degree (randomly select if multiple) and set $R_0 = \{x\}$. Append R_0 to R
- While $|R| < n$, for $i = 1,2, \dots$
  - Identify all the neighbors of R_{i-1} and put them into a set A_i (remove any vertices already in R)
  - Sort A_i in ascending order of degree, call this sorted set R_i
  - Append R_i to R

Cuthill–McKee algorithm (CM)

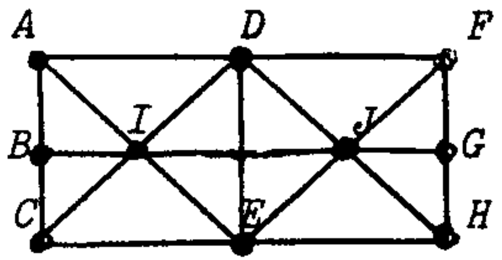


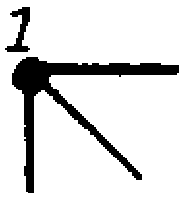
Cuthill–McKee algorithm (CM)

- Algorithm: (https://en.wikipedia.org/wiki/Cuthill%E2%80%93McKee_algorithm)

- Let $R = \{\}$
- Choose a vertex x with lowest degree (randomly select if multiple) and set $R_0 = \{x\}$. Append R_0 to R
- While $|R| < n$, for $i = 1,2, \ldots$
  - Identify all the neighbors of R_{i-1} and put them into a set A_i (remove any vertices already in R)
  - Sort A_i in ascending order of degree, call this sorted set R_i
  - Append R_i to R

Cuthill–McKee algorithm (CM)



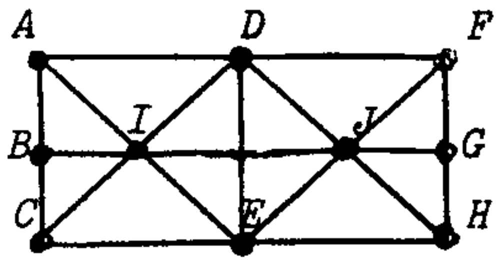


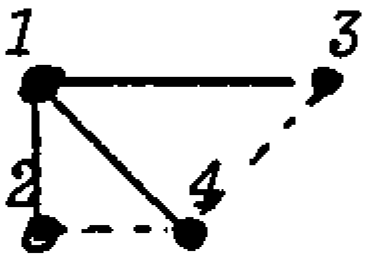
Cuthill–McKee algorithm (CM)

- Algorithm: (https://en.wikipedia.org/wiki/Cuthill%E2%80%93McKee_algorithm)

- Let $R = \{\}$
- Choose a vertex x with lowest degree (randomly select if multiple) and set $R_0 = \{x\}$. Append R_0 to R
- While $|R| < n$, for $i = 1,2, \ldots$
  - Identify all the neighbors of R_{i-1} and put them into a set A_i (remove any vertices already in R)
  - Sort A_i in ascending order of degree, call this sorted set R_i
  - Append R_i to R

Cuthill–McKee algorithm (CM)



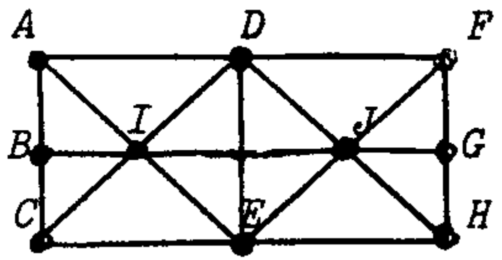


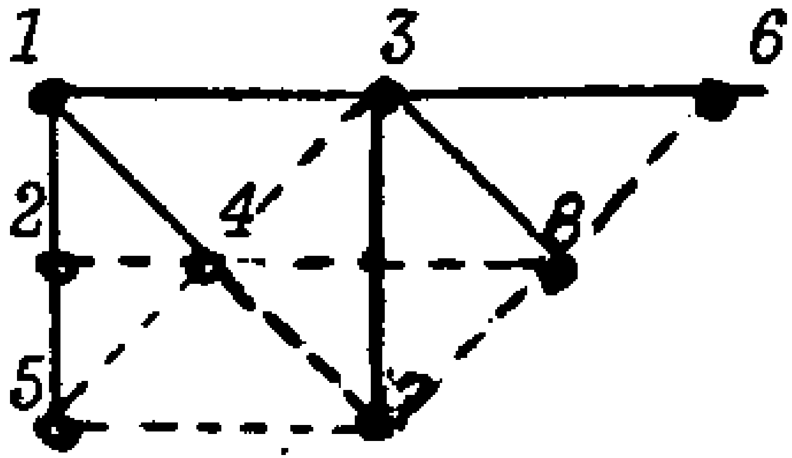
Cuthill–McKee algorithm (CM)

- Algorithm: (https://en.wikipedia.org/wiki/Cuthill%E2%80%93McKee_algorithm)

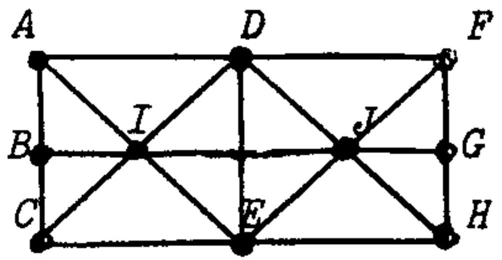
- Let $R = \{\}$
- Choose a vertex x with lowest degree (randomly select if multiple) and set $R_0 = \{x\}$. Append R_0 to R
- While $|R| < n$, for $i = 1,2, \dots$
  - Identify all the neighbors of R_{i-1} and put them into a set A_i (remove any vertices already in R)
  - Sort A_i in ascending order of degree, call this sorted set R_i
  - Append R_i to R

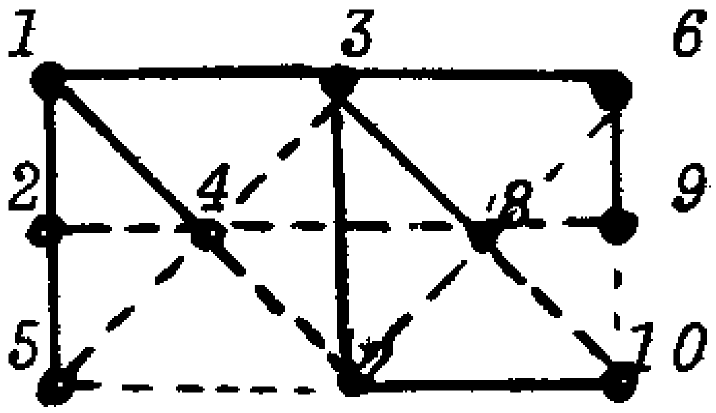
Cuthill–McKee algorithm (CM)





Cuthill–McKee algorithm (CM)





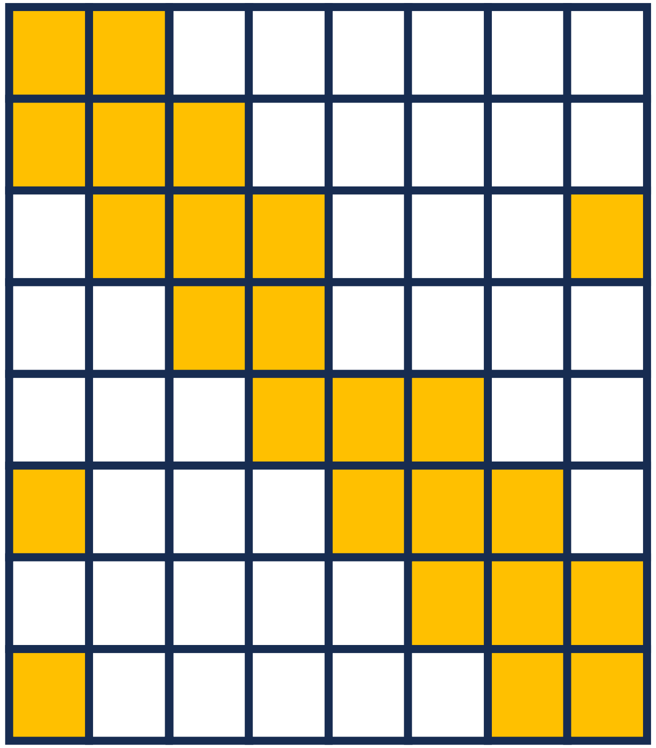
Cuthill–McKee algorithm (CM)

- Ungraded HW assignment: Try to run the algorithm on a graph of your choice and see if you understand.

- Reverse Cuthill–McKee algorithm (RCM): Same as CM, but reverse the node numbering
  - Leads to lower fill-in after factorization (beyond the scope of this class)

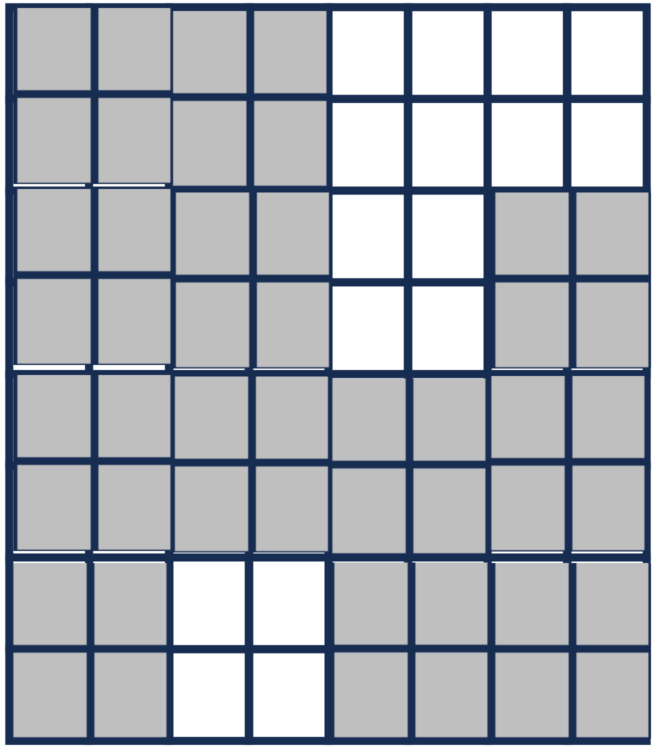
Going back: Introducing Sparsity in Flashattention

M



$B_c = B_r = 2$

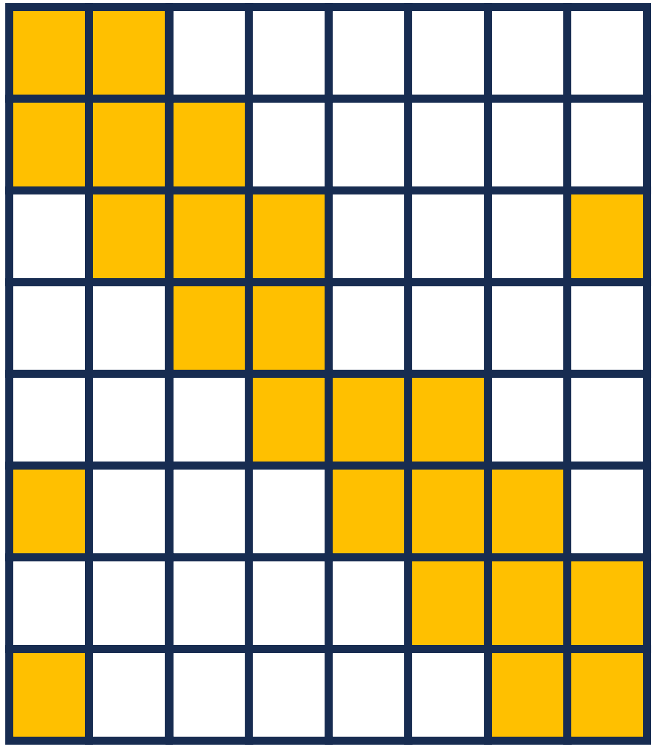
QK^T



- What could have led to an even better reduction in computation?

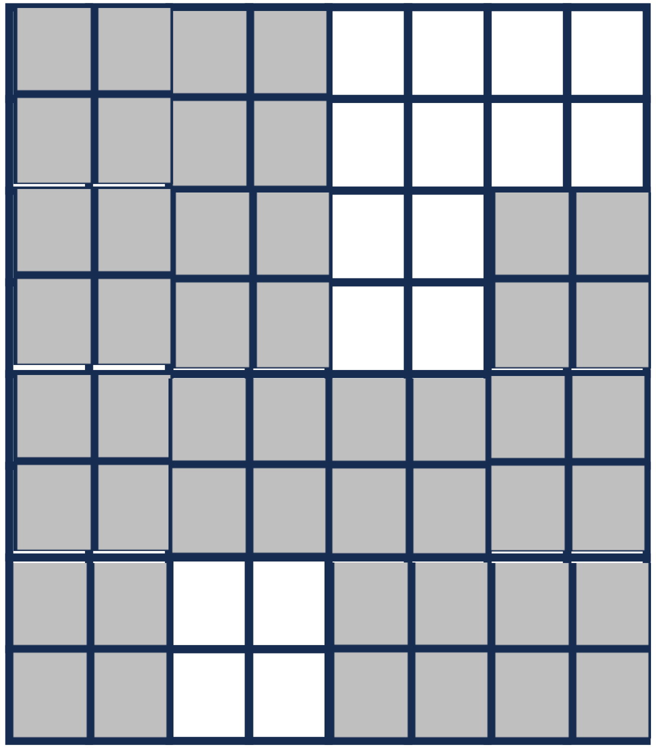
Going back: Introducing Sparsity in Flashattention

M



$B_c = B_r = 2$

QK^T



- What could have led to an even better reduction in computation? RCM algorithm before calling the modified flashattention

Sparsity in Attention

- Reorder attention masks to reduce the number of blocks
- Call Flashattention, to only operate on non-zero blocks
- Caveats:
  - Reordering takes time, so not applicable to all use cases
  - Sometimes the block structure may become worse
- Our proposed technique: Using bandwidth as a proxy for reducing the number of blocks works well for scientific computing, but not for attention masks
  - Use hypergraph partitioning techniques that directly optimize for reducing the number of blocks

Next Class

- 11/6 Lecture 20
  - Modeling Cluster of Accelerators

CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

Thank You

- Questions?

- Email: sanmukh.kuppannagari@case.edu