CSDS 451: Designing High Performant Systems for AI

Lecture 17

10/28/2025

Sanmukh Kuppannagari

sanmukh.kuppannagari@case.edu

https://sanmukh.research.st/

Case Western Reserve University

Outline

- Sparse Transformers Basics
- Sparse Masks

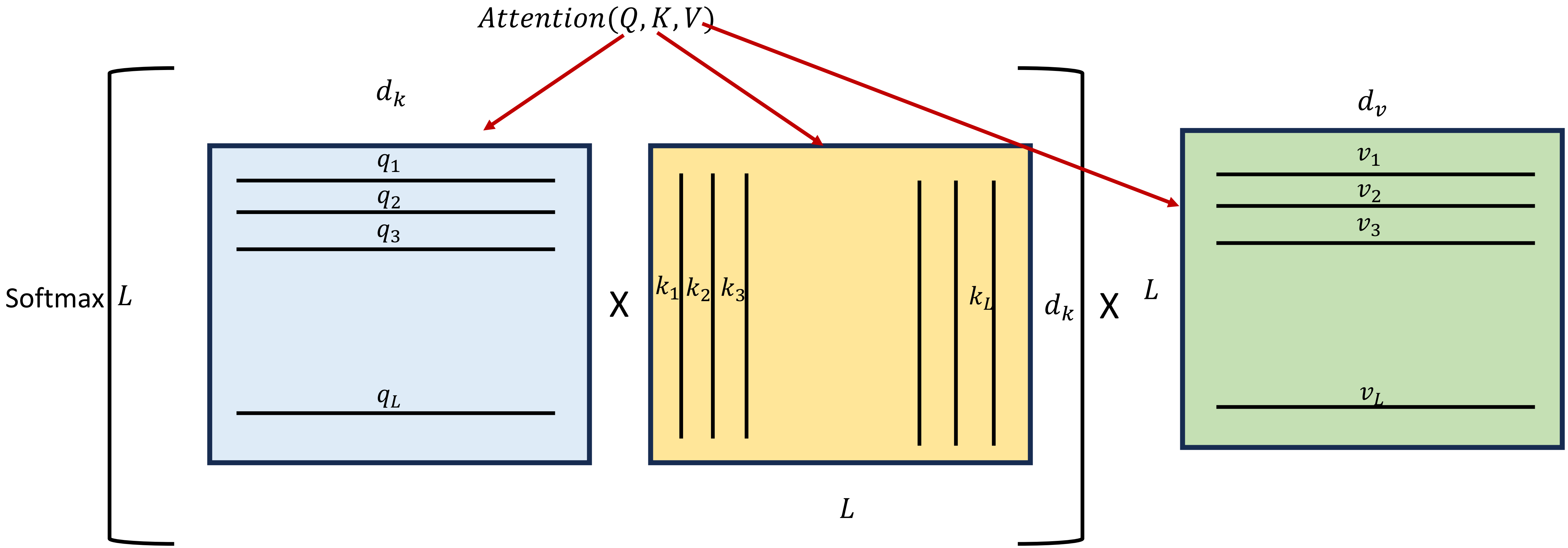
Outline

- Sparse Transformers Basics
- Sparse Masks

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

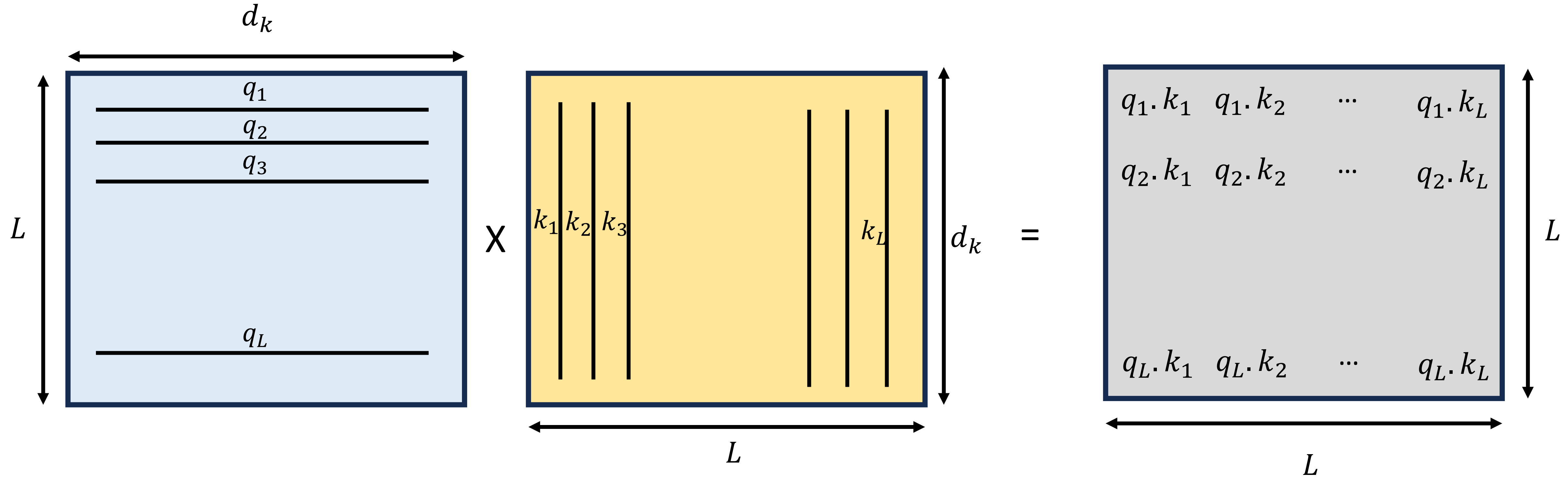
Attention



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

Zooming into Attention



Zooming into Attention

$$\text{Softmax}\underbrace{\left[\;\boxed{\begin{matrix} q_1.k_1 & q_1.k_2 & \cdots & \\ q_2.k_1 & q_2.k_2 & \cdots & q_2.k_L \\ & & & \\ q_L.k_1 & q_L.k_2 & \cdots & q_L.k_L \end{matrix}}\;\right]}_{L} = \underbrace{\boxed{\begin{matrix} w_{11} & w_{12} & \cdots & w_{1L} \\ w_{21} & w_{22} & \cdots & w_{2L} \\ & & & \\ w_{L1} & w_{L2} & \cdots & w_{LL} \end{matrix}}}_{L}$$

L (rows of each matrix)

$$[w_{i1}, w_{i2}, \ldots, w_{iL}] = softmax([q_1.k_1, q_1.k_2, \ldots, q_1.k_L])$$

Zooming into Attention

$$\text{Softmax}\left[L \underbrace{\begin{bmatrix} q_1.k_1 & q_1.k_2 & \cdots & \\ q_2.k_1 & q_2.k_2 & \cdots & q_2.k_L \\ & & & \\ q_L.k_1 & q_L.k_2 & \cdots & q_L.k_L \end{bmatrix}}_{L} \right] = \underbrace{\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1L} \\ w_{21} & w_{22} & \cdots & w_{2L} \\ & & & \\ w_{L1} & w_{L2} & \cdots & w_{LL} \end{bmatrix}}_{L} L$$

$$[w_{i1}, w_{i2}, \dots, w_{iL}] = softmax([q_i.k_1, q_i.k_2, \dots, q_i.k_L])$$

Exponent of dot product of query i with key j

$$w_{ij} = \frac{e^{q_i.k_j}}{\sum_l e^{q_i.k_l}}$$

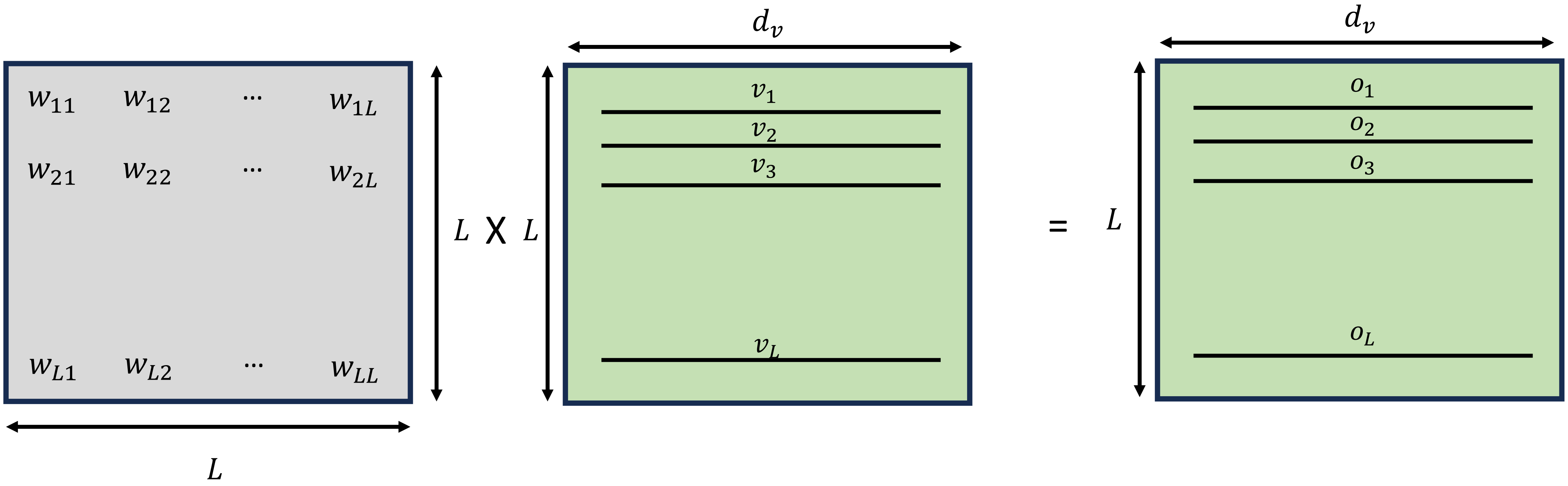
Sum of exponents of dot product of query i with ALL keys

CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

Zooming into Attention

$$\underbrace{\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1L} \\ w_{21} & w_{22} & \cdots & w_{2L} \\ & & & \\ w_{L1} & w_{L2} & \cdots & w_{LL} \end{bmatrix}}_{L} L \; \text{X} \; L \overbrace{\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \\ v_L \end{bmatrix}}^{d_v} = L \overbrace{\begin{bmatrix} w_{11}v_1 + w_{12}v_2 + \ldots + w_{1L}v_L \\ w_{21}v_1 + w_{22}v_2 + \ldots + w_{2L}v_L \\ w_{31}v_1 + w_{32}v_2 + \ldots + w_{3L}v_L \\ \\ w_{L1}v_1 + w_{L2}v_2 + \ldots + w_{LL}v_L \end{bmatrix}}^{d_v}$$

Zooming into Attention



Attention Mechanism – Fast Facts

- Naïve Matrix Multiplication based implementation requires materialization of L^2 sized temporary softmax matrix

- Compute Units quickly run out of memory, leading to transfers between global memory and shared cache of CU

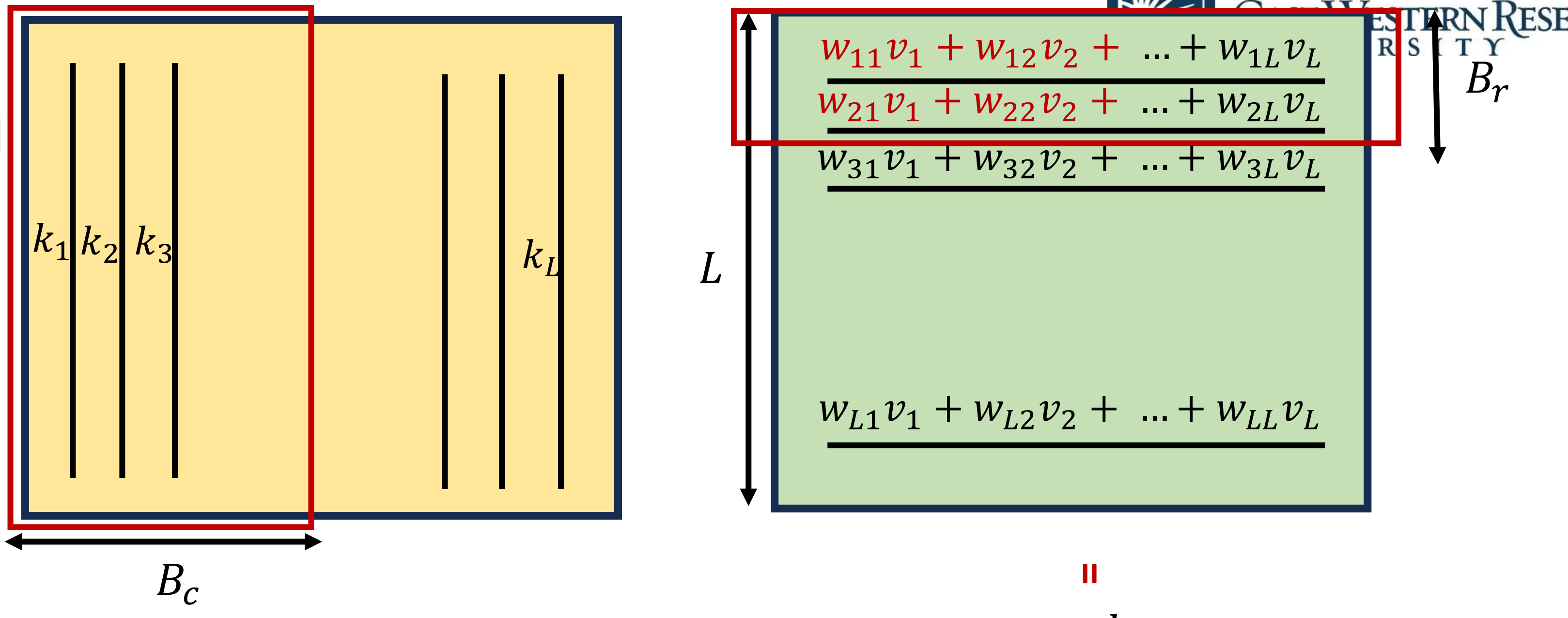
- This was the standard implementation in Pytorch before **FlashAttention**

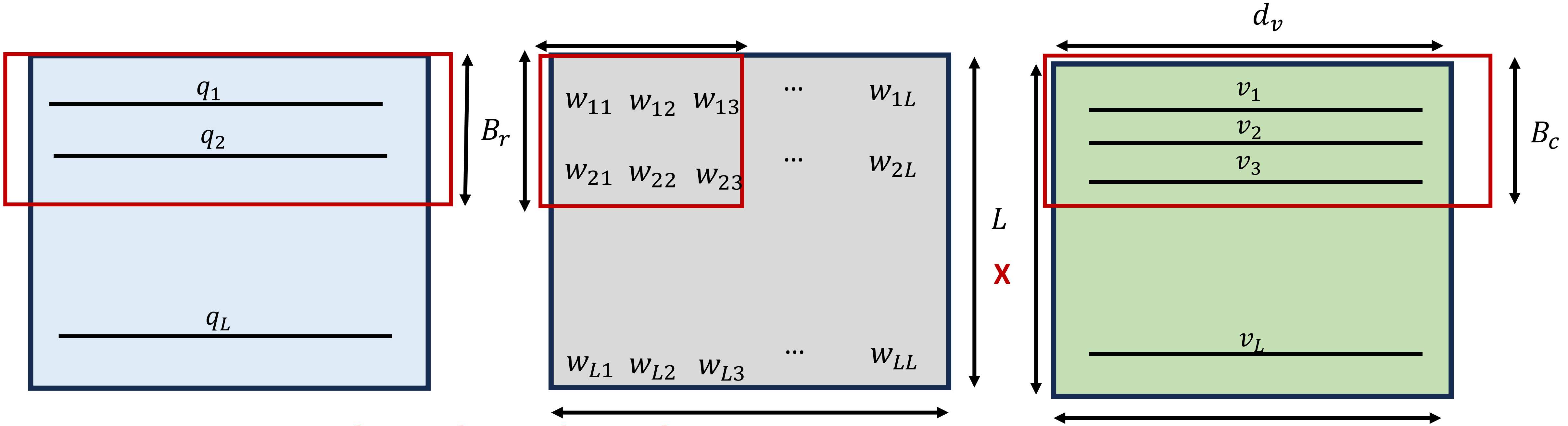
- Flashattention approach – Partition Z into independent tiles of size $B_r \times B_c$, and figure out the inputs tiles (Q, K, V) and output tiles (O) to produce

FlashAttention

Ignore softmax for now

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





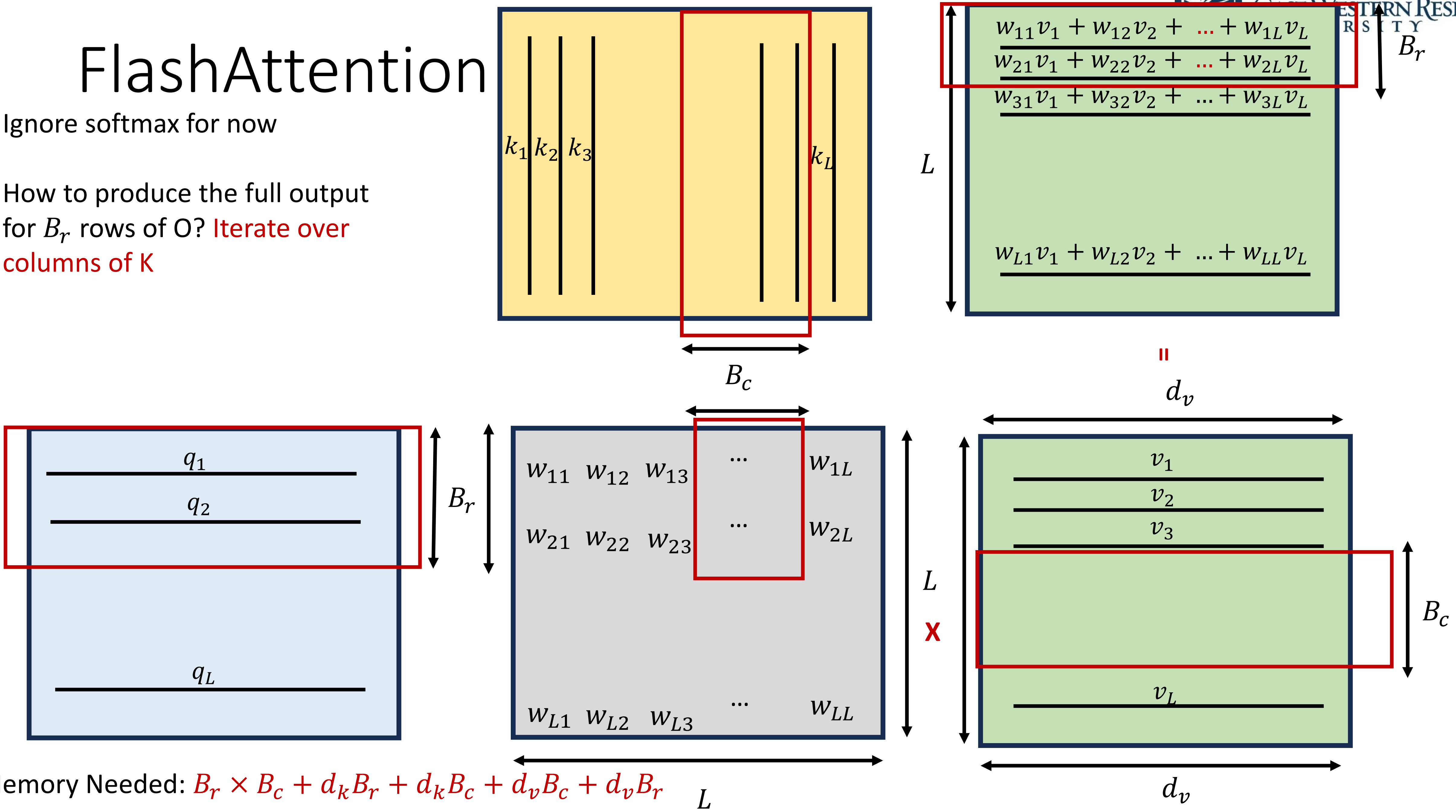
Memory Needed: $B_r \times B_c + d_k B_r + d_k B_c + d_v B_c + d_v B_r$

L

FlashAttention

Ignore softmax for now

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



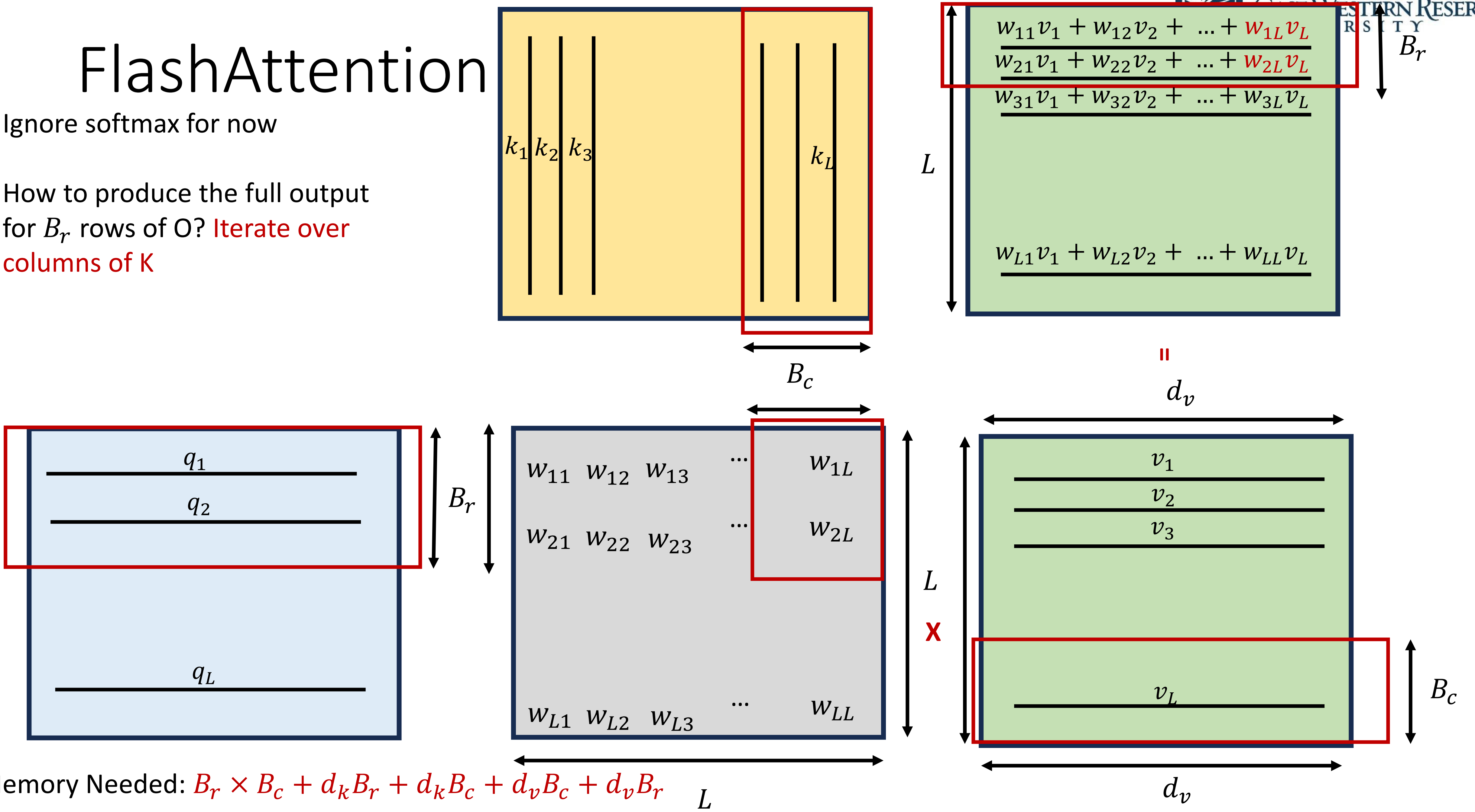
Memory Needed: $B_r \times B_c + d_k B_r + d_k B_c + d_v B_c + d_v B_r$

CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

FlashAttention

Ignore softmax for now

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

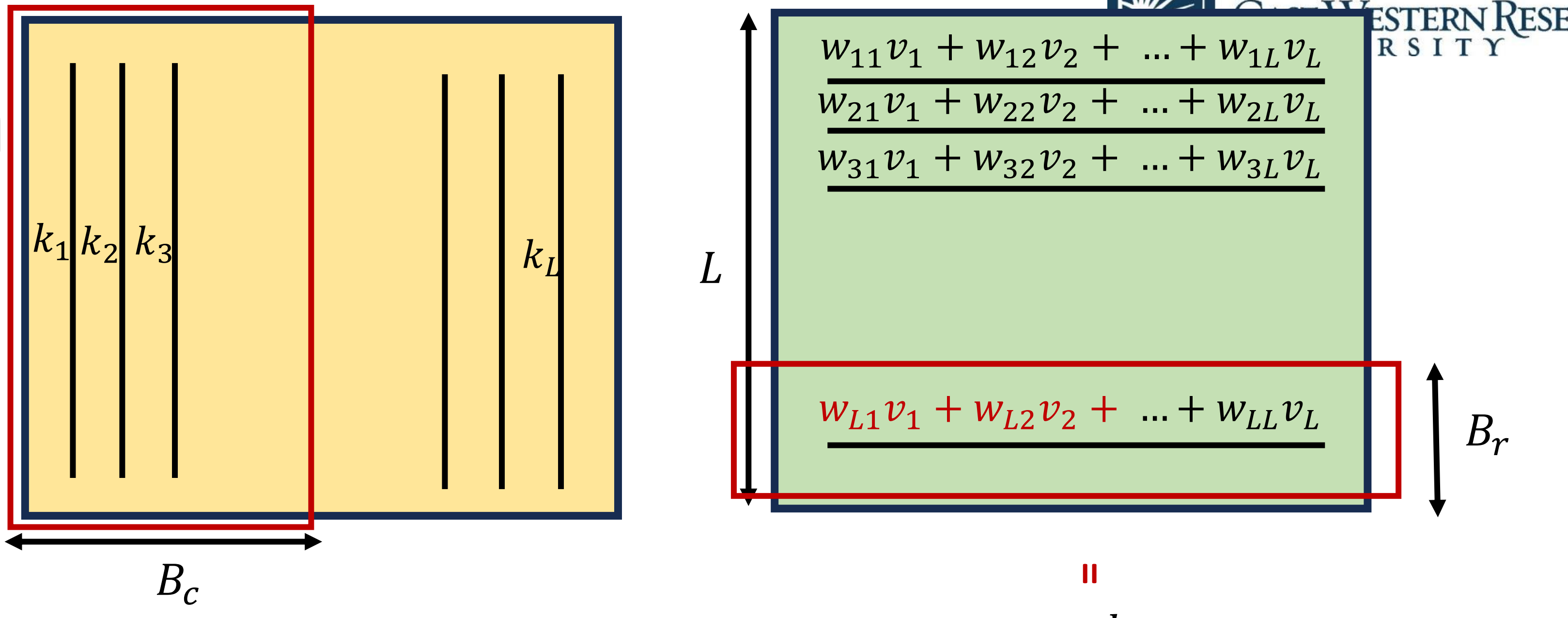


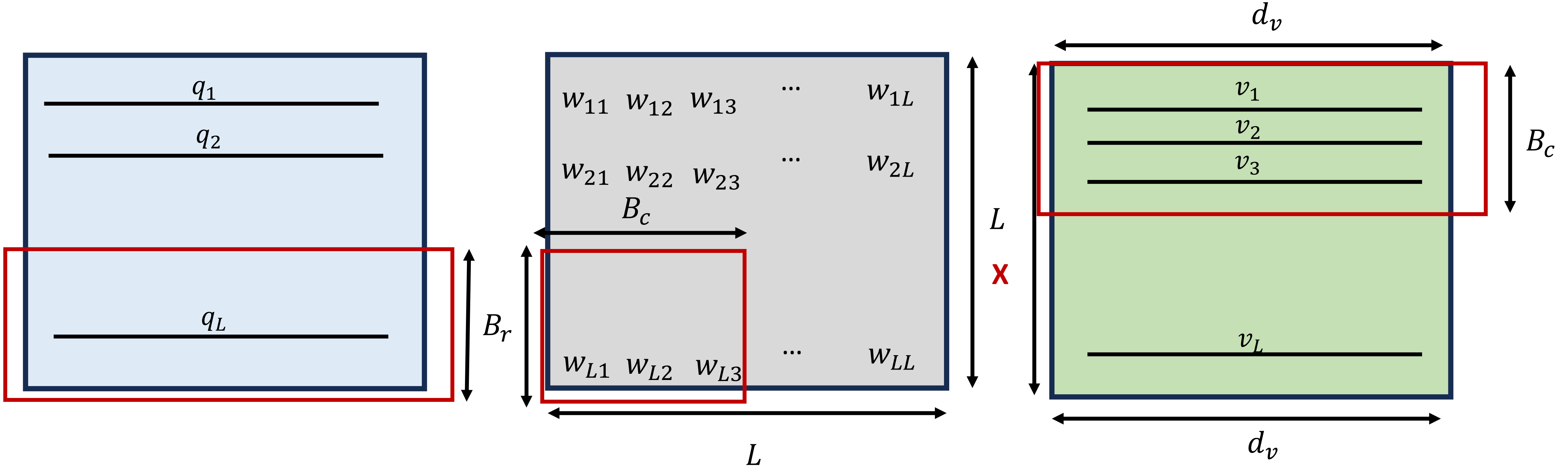
Memory Needed: $B_r \times B_c + d_k B_r + d_k B_c + d_v B_c + d_v B_r$

FlashAttention

Ignore softmax for now

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

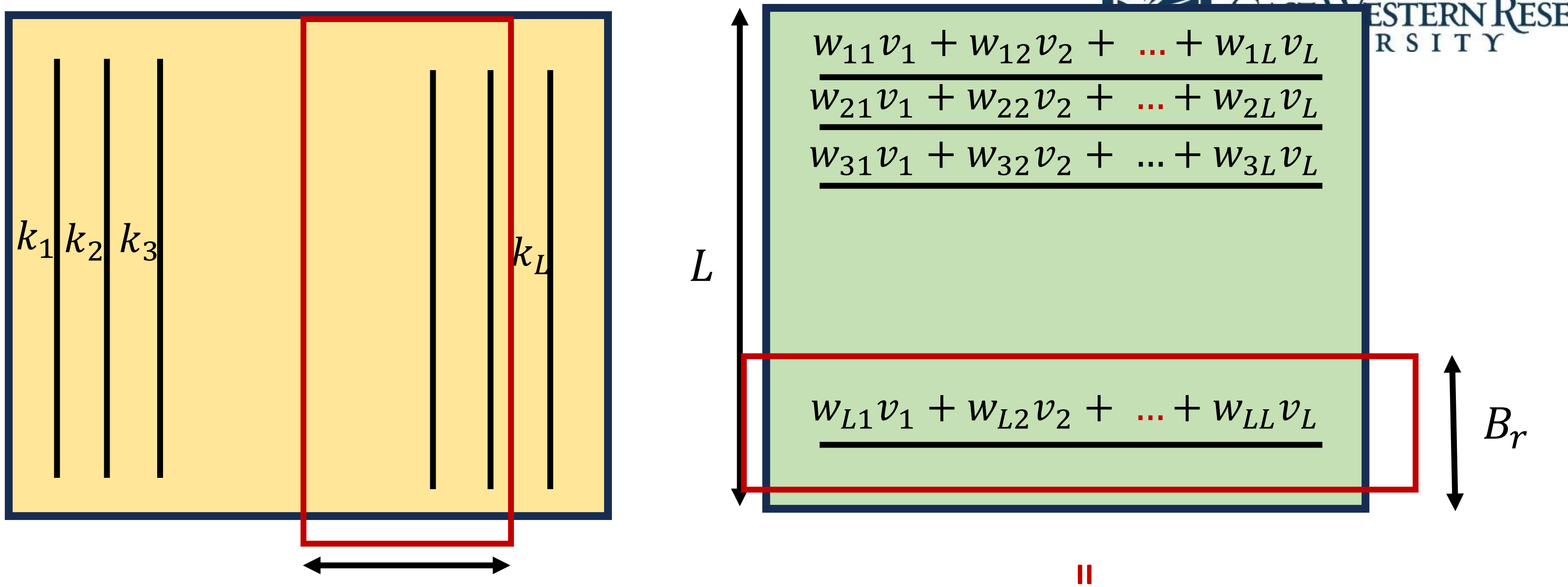


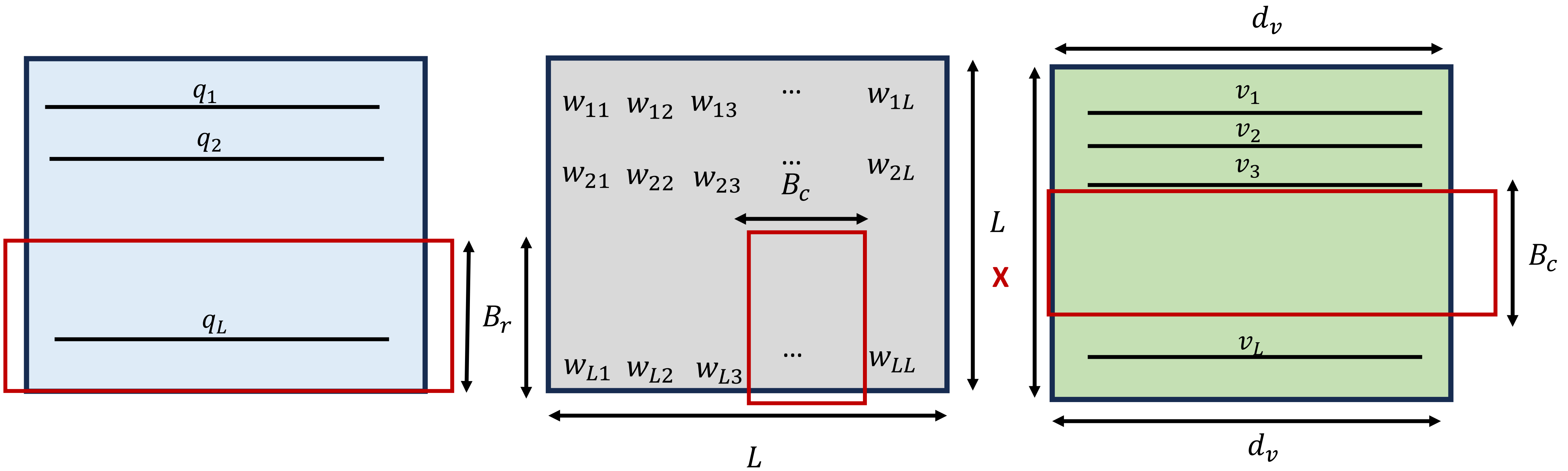


FlashAttention

Ignore softmax for now

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

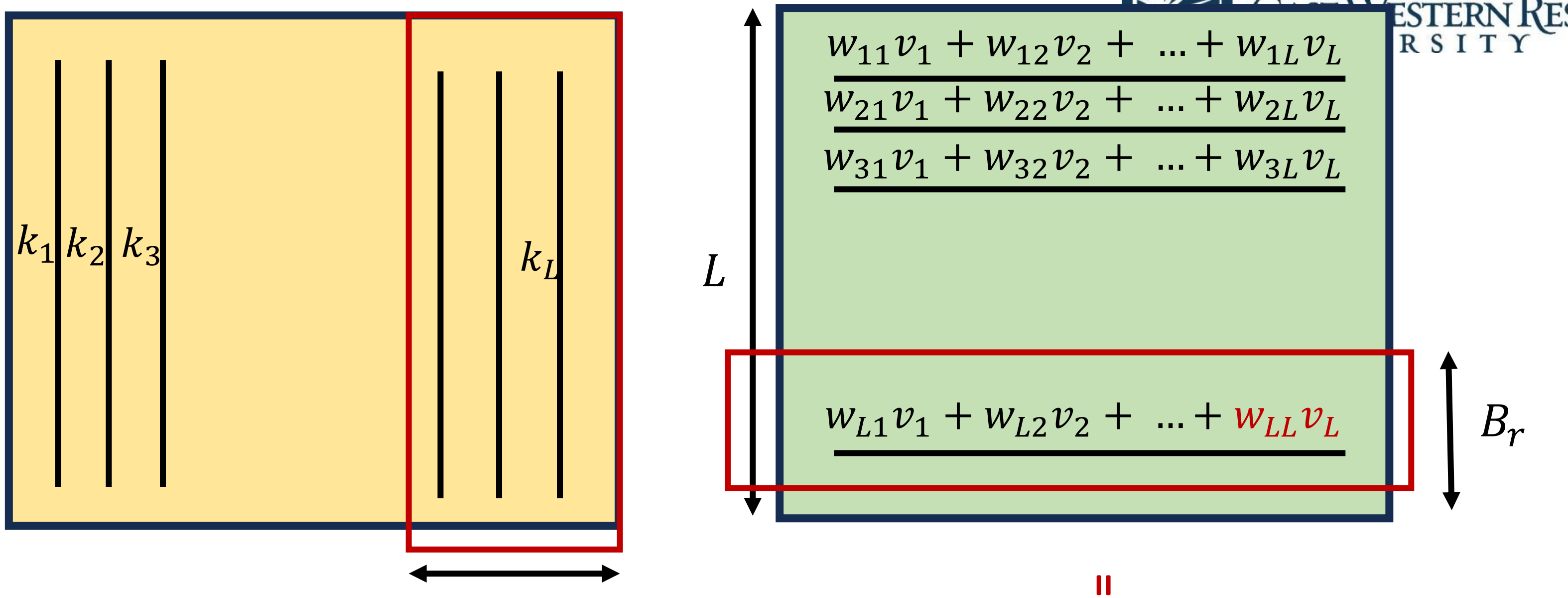


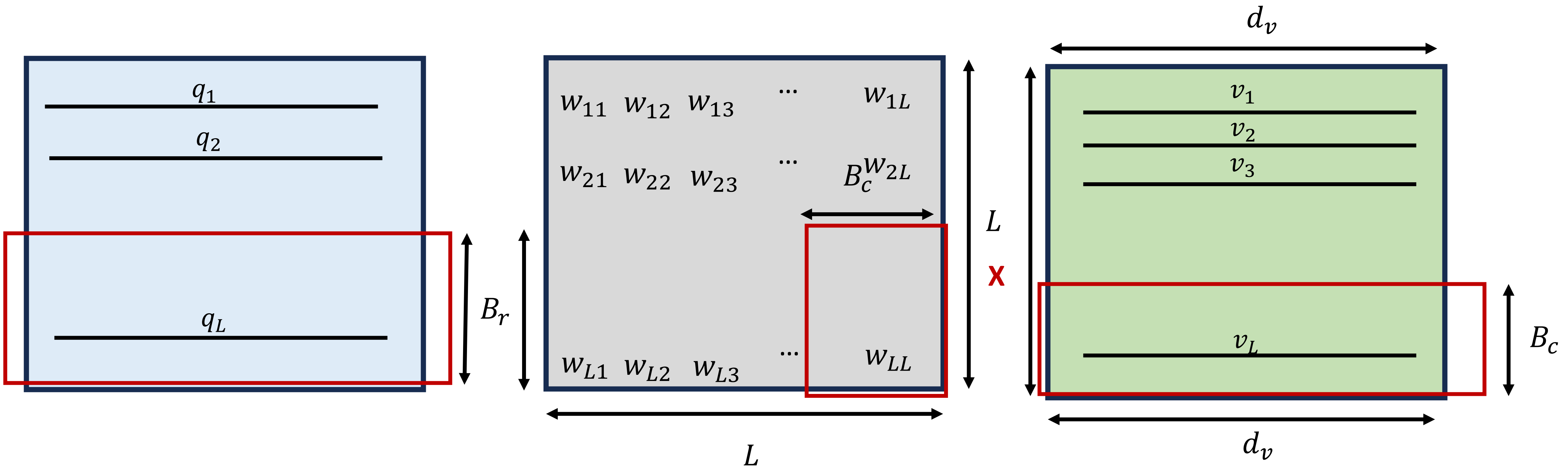


FlashAttention

Ignore softmax for now

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





How do we compute “partial” softmaxes

- $[w_{i1}, w_{i2}, w_{i3}] = [w_{i1}, w_{i2}, w_{i3}] \times l/l_2$

- In practice,
  - Output O is rescaled
  - In softmax computation, scaling by the maximum value of row is performed to avoid numerical stability issues
- Ungraded HW Assignment: Try to understand this. Will not ask in exam.

On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
On chip, compute $m_i^{\mathrm{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\mathrm{new}} = e^{m_i - m_i^{\mathrm{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\mathrm{new}})^{-1}(\mathrm{diag}(\ell_i) e^{m_i - m_i^{\mathrm{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\mathrm{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.
Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{new}}$ to HBM.

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

Flash attention

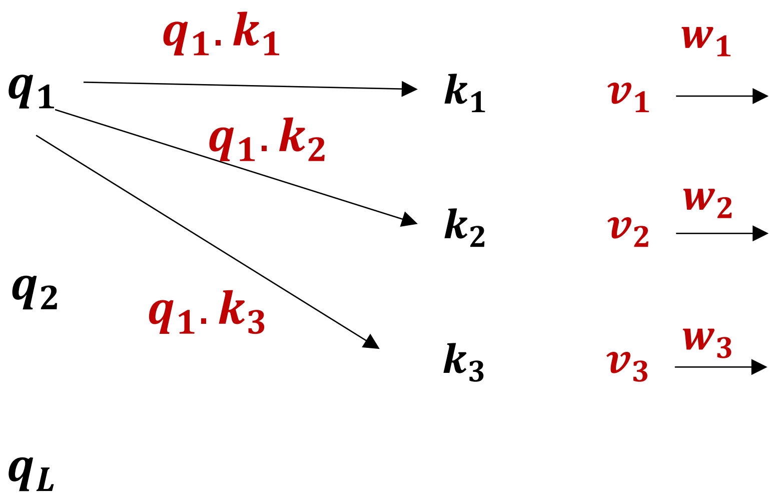
- Reduces the memory requirements from $O(L^2)$ to $??$

Sparse Transformers

- Flashattention reduces the memory requirements from $O(L^2)$ to $B_r \times B_c$
- However, no change it computations – still require $O(L^2)$ computations
- For longer values of L this can be exorbitant
- Solution: Sparse Transformers

Key Idea

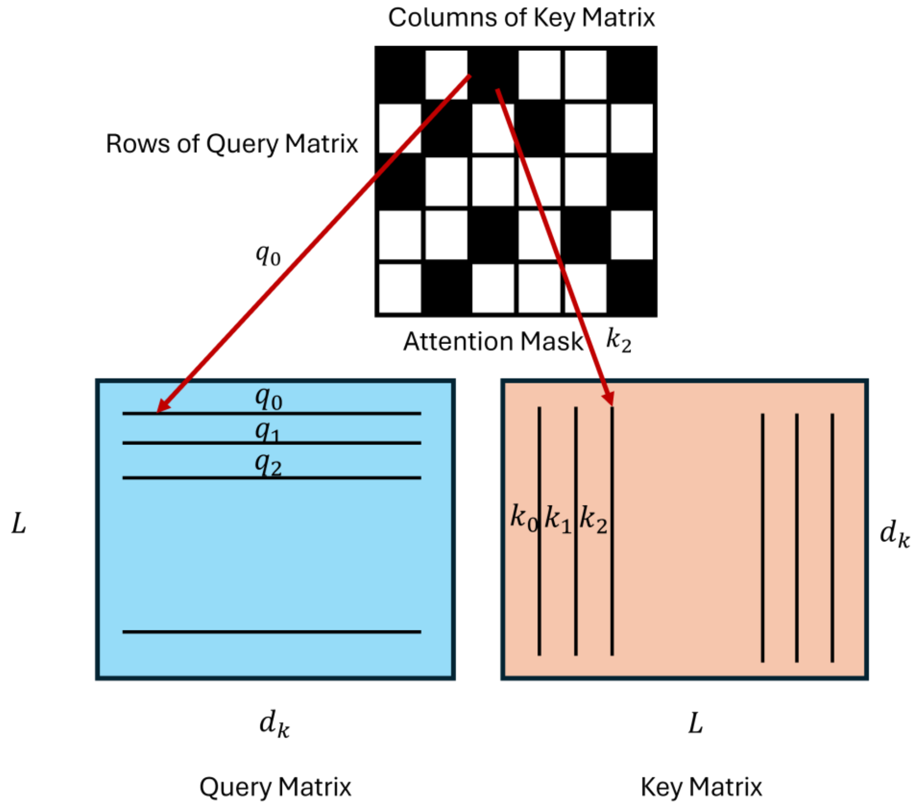
- Considering soft-lookup view of attention
- Not every query-key pairs need to interact
- As keys and queries are projection of tokens -> Not every pair of token need to interact



$$w_i = \frac{e^{q_1.k_i}}{\sum_j e^{q_1.k_j}}$$

Introducing Sparsity in Attention

- Key Idea: Not every pair of query and key/value need to interact
- If we represent the query-key interactions as an adjacency matrix, we want to prune the entries of the matrix



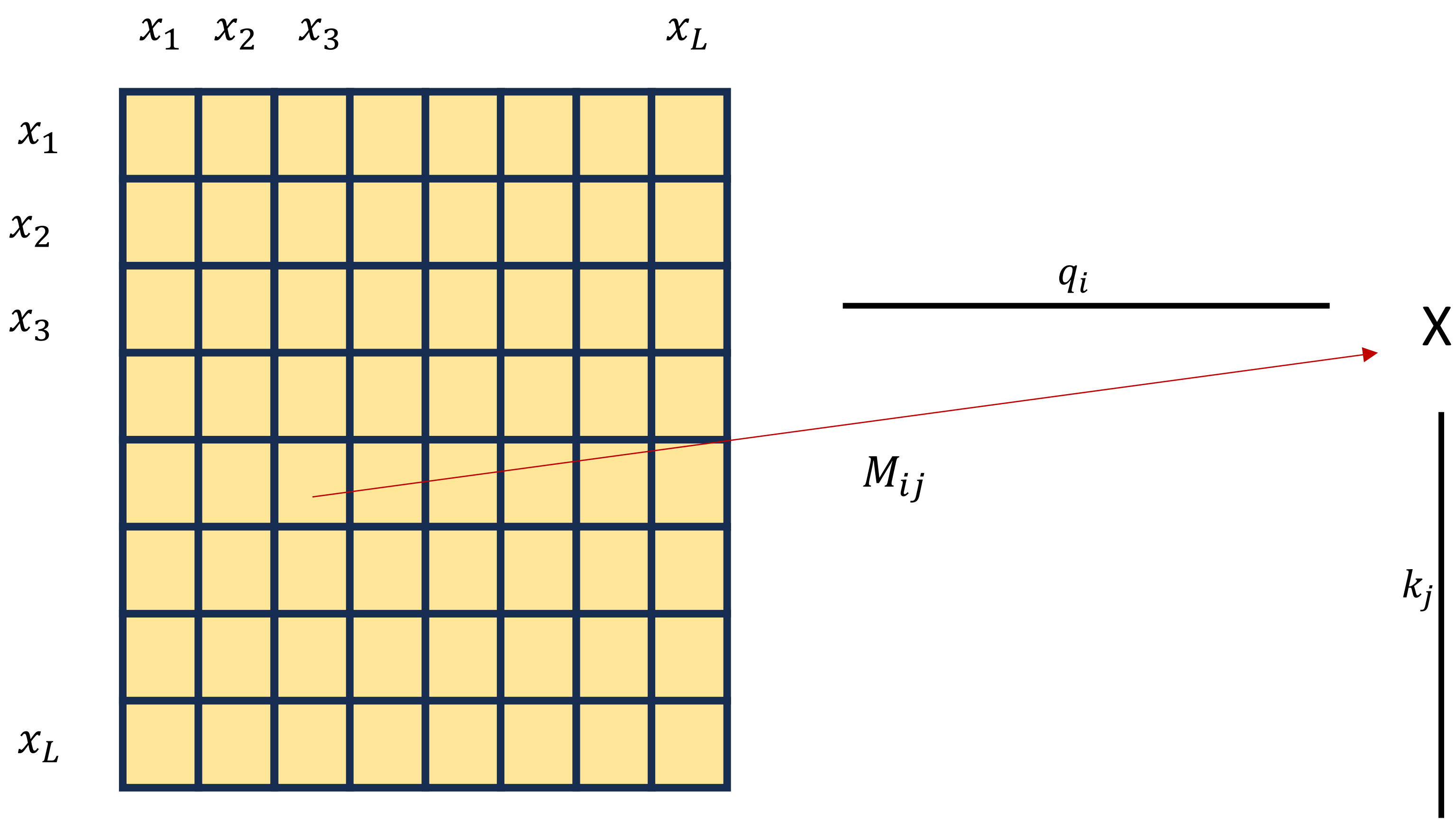
Attention Mask

Self-Attention in Transformer Blocks:

All 1s adjacency matrix

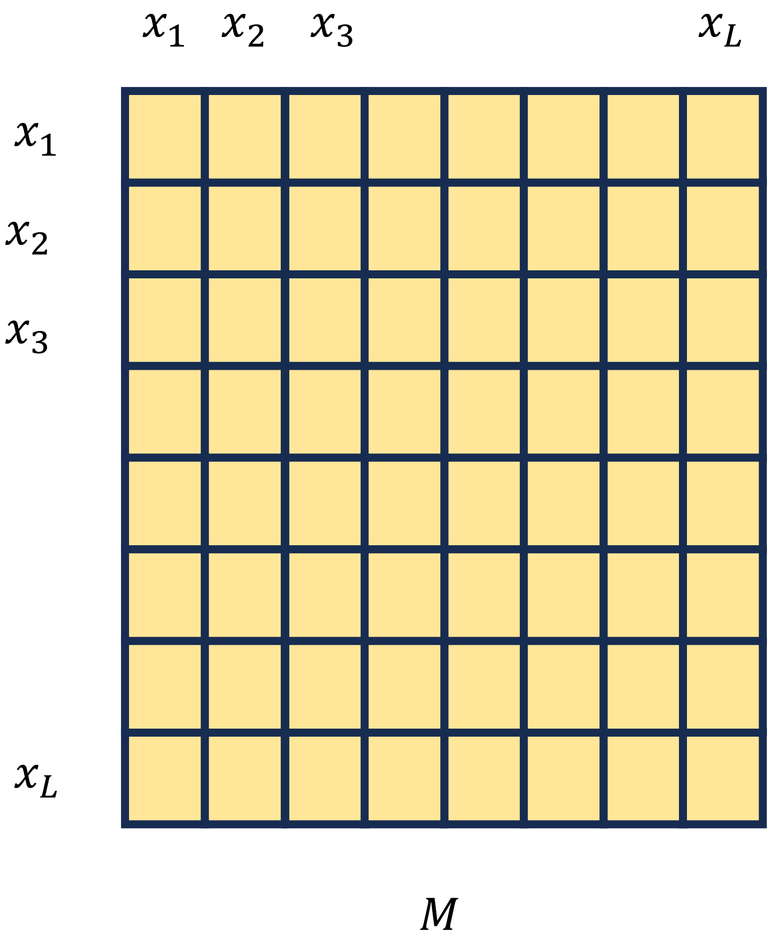
(colored cell is 1)

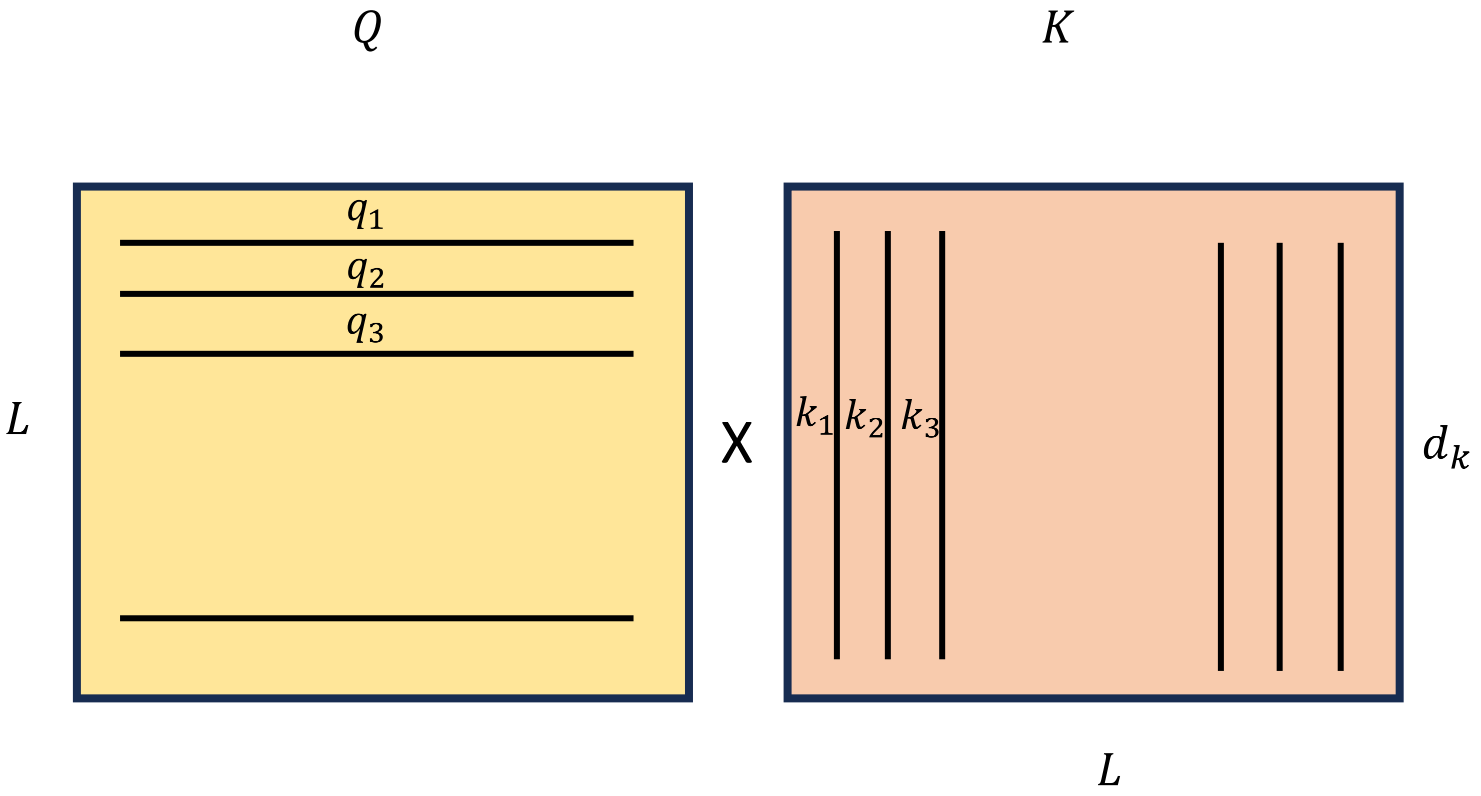
Each row corresponds to a query



Each column corresponds to a key

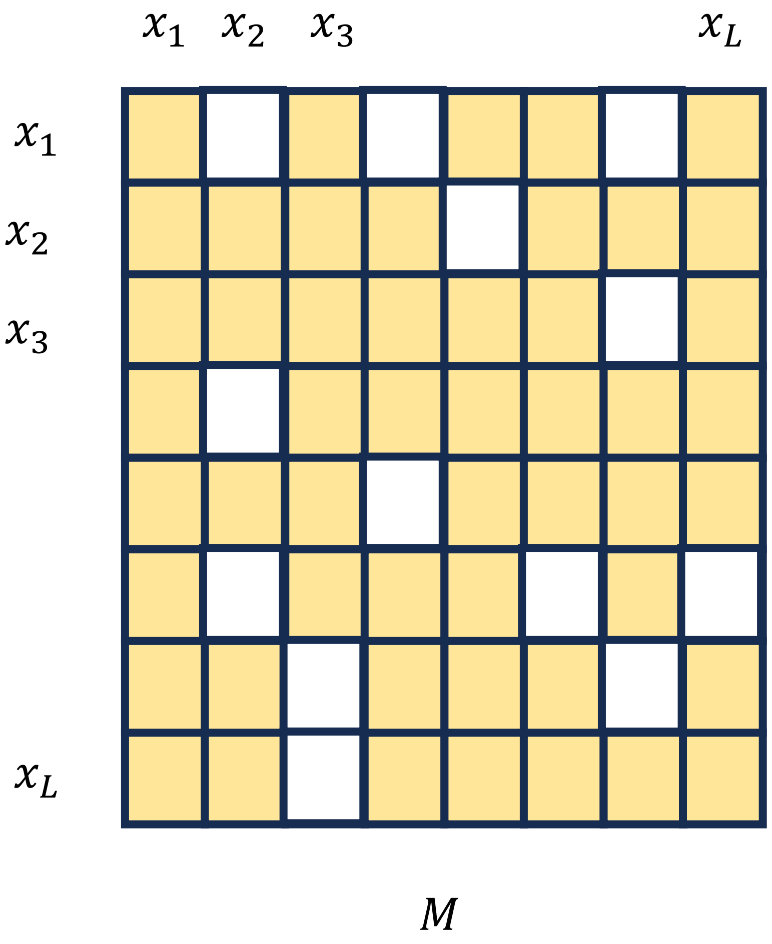
Attention Mask

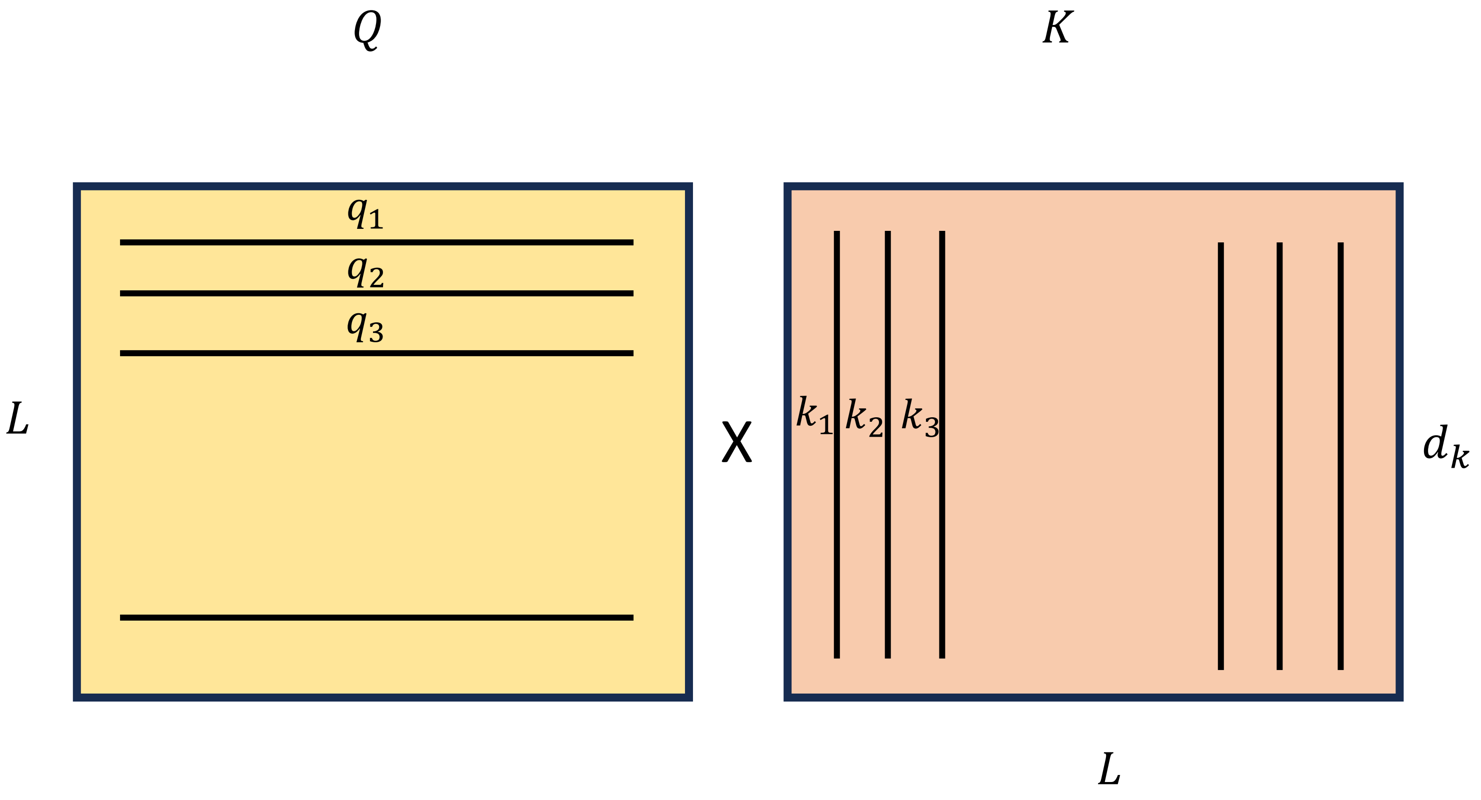




The L^2 entries of M represent the L^2 dot-products needed for the matrix multiplication QK^T

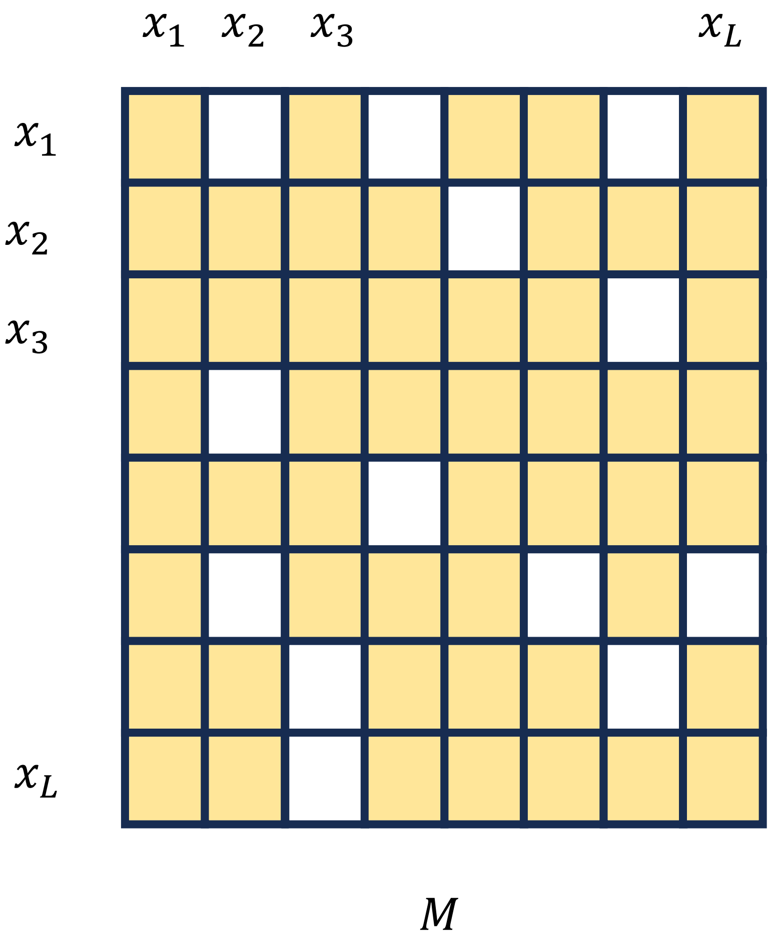
QK Product with Attention Mask

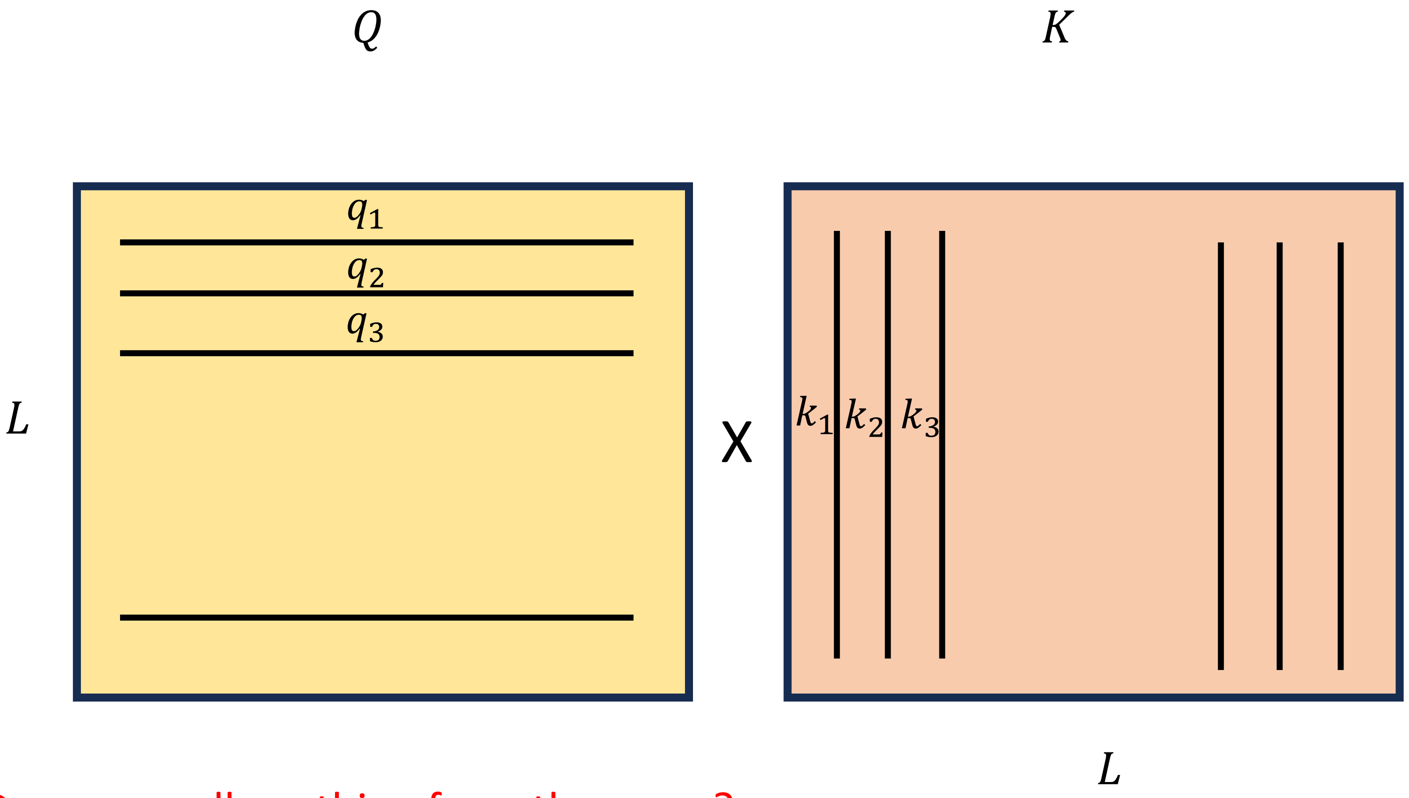




If some of the L^2 entries of M are 0, the corresponding dot-products should not be computed

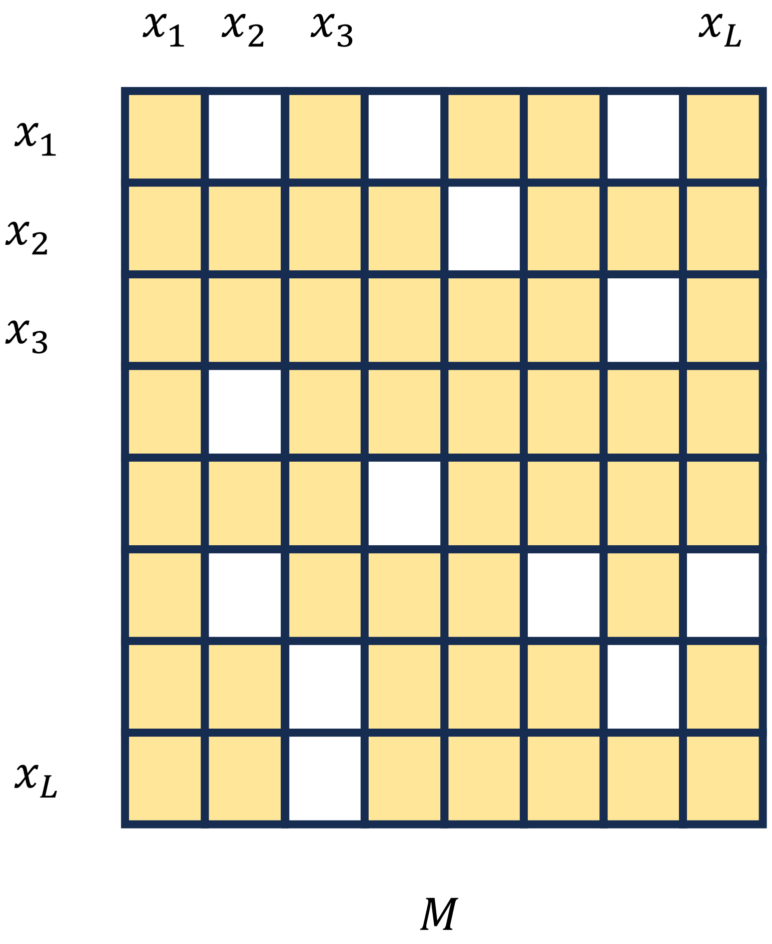
QK Product with Attention Mask

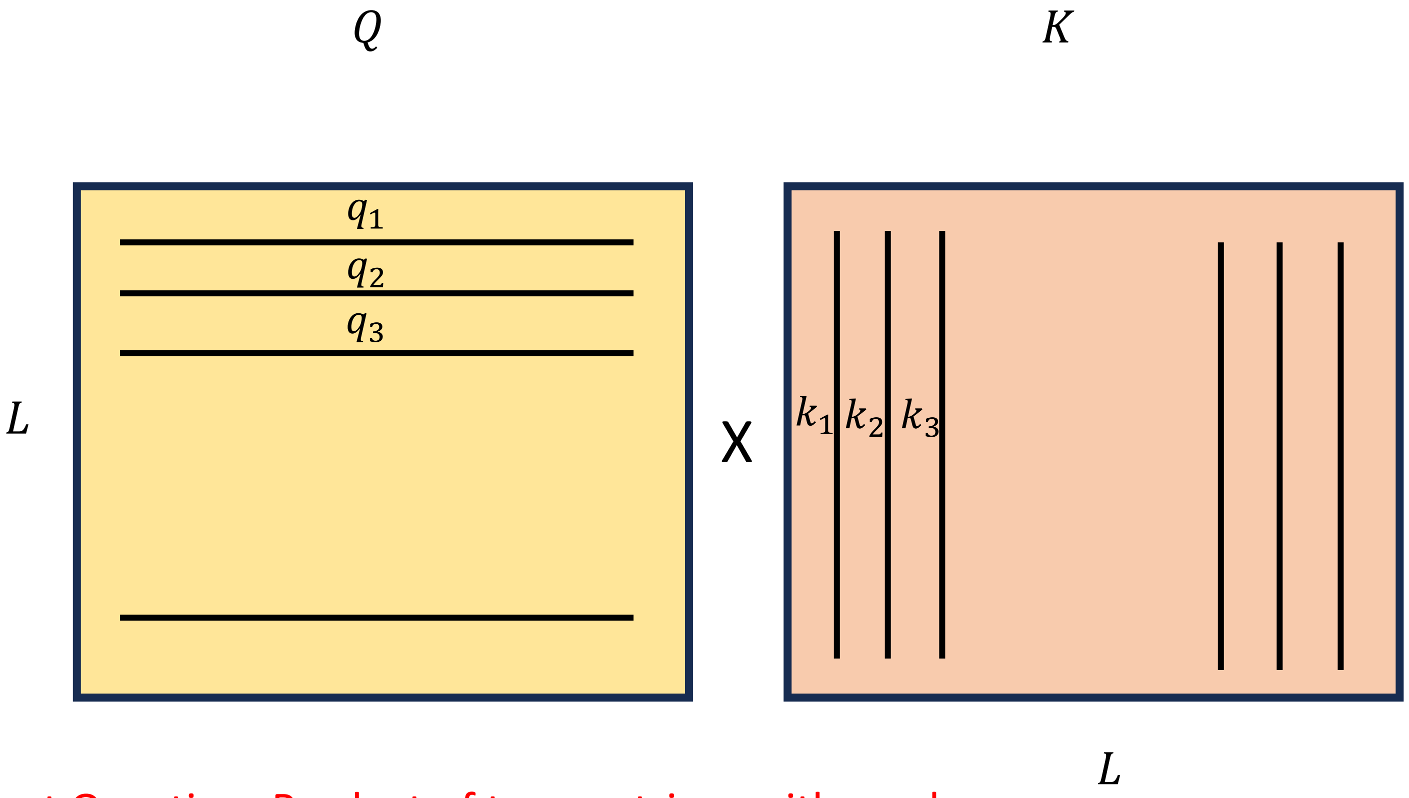




Do you recall anything from the exam?

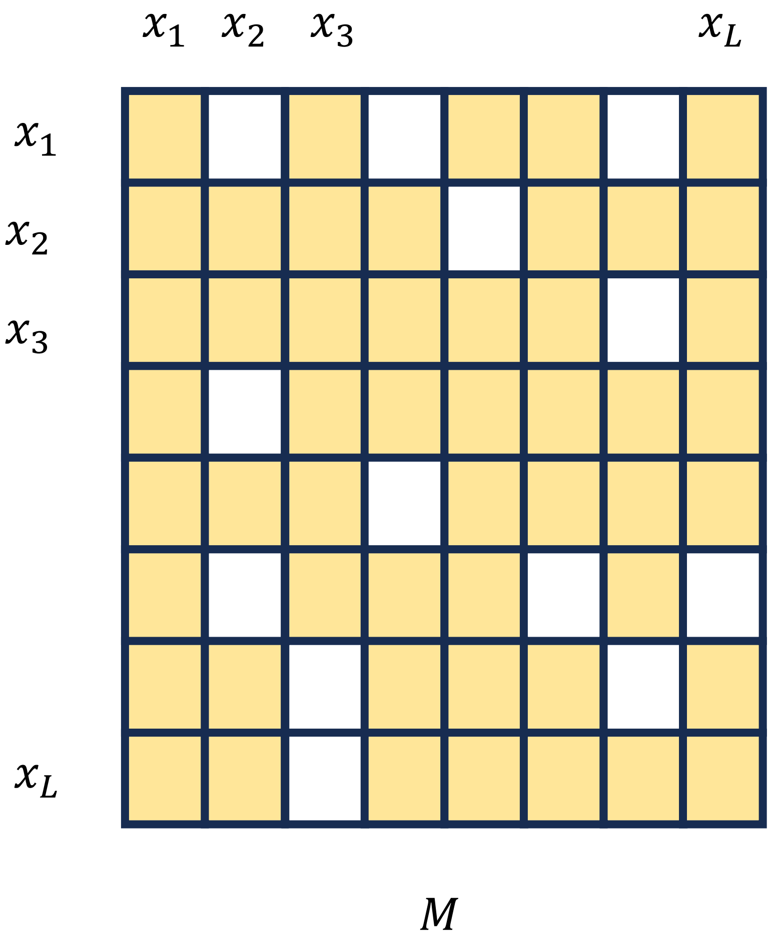
QK Product with Attention Mask

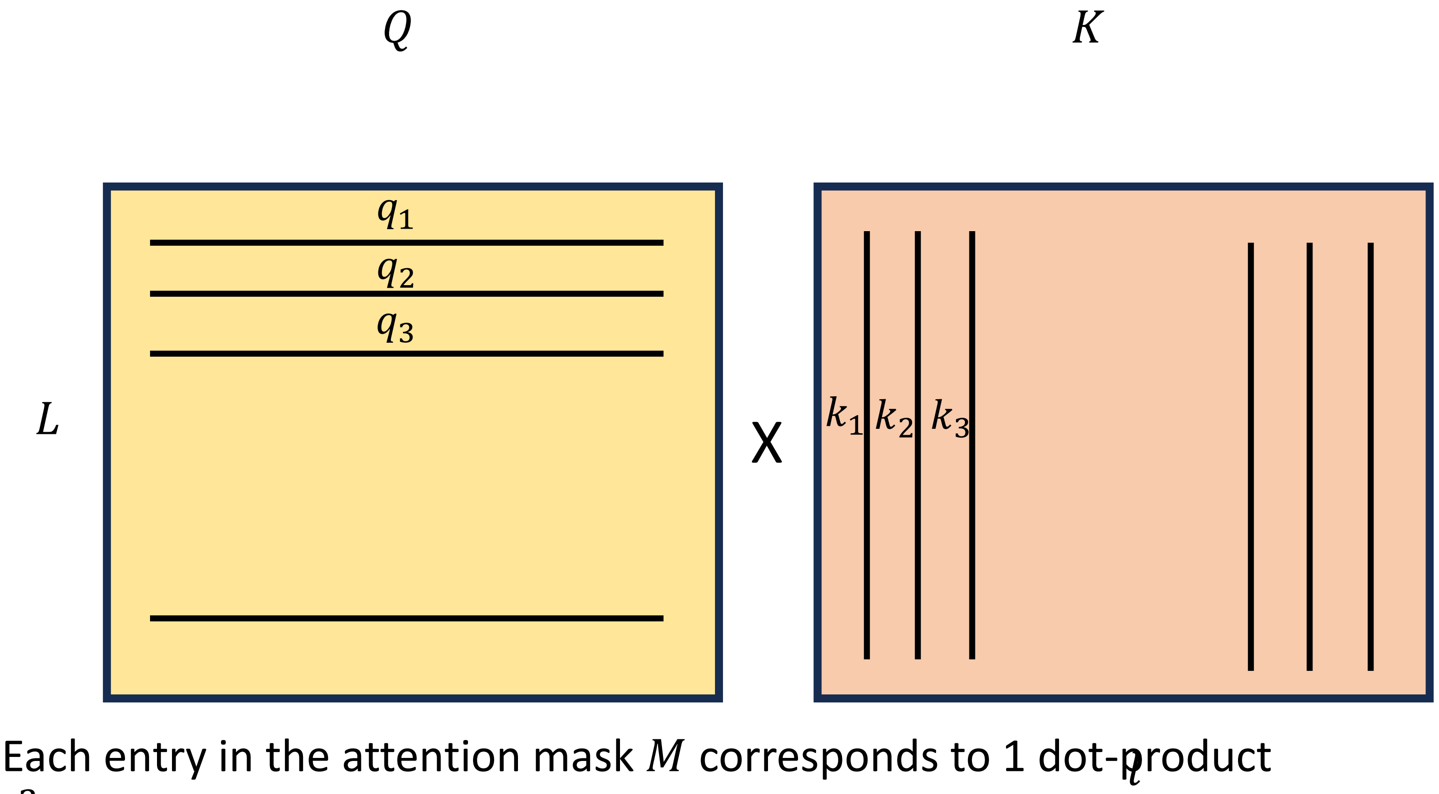




Last Question: Product of two matrices with mask

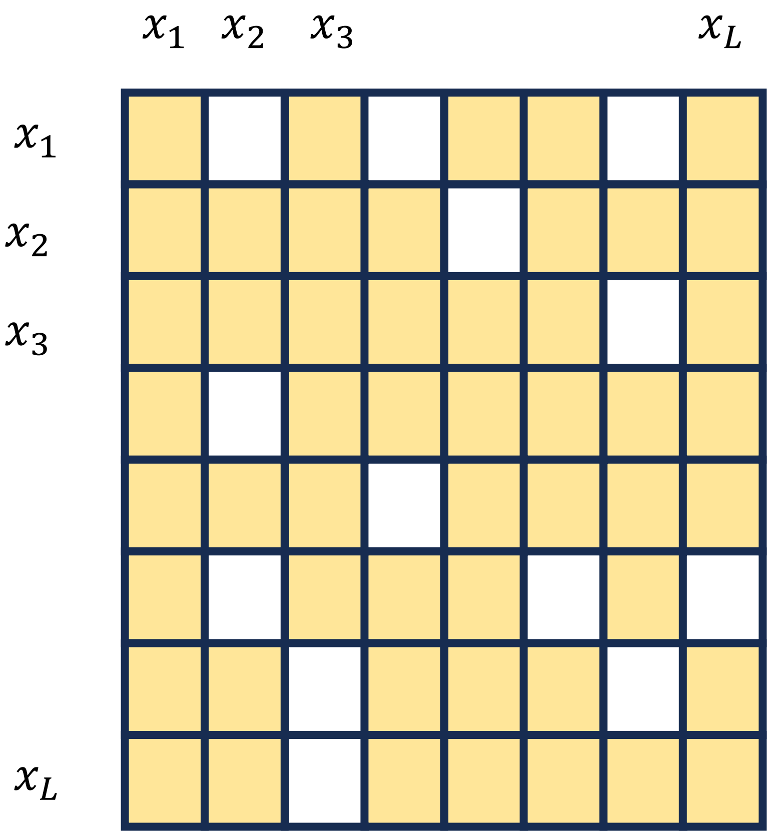
QK Product with Attention Mask





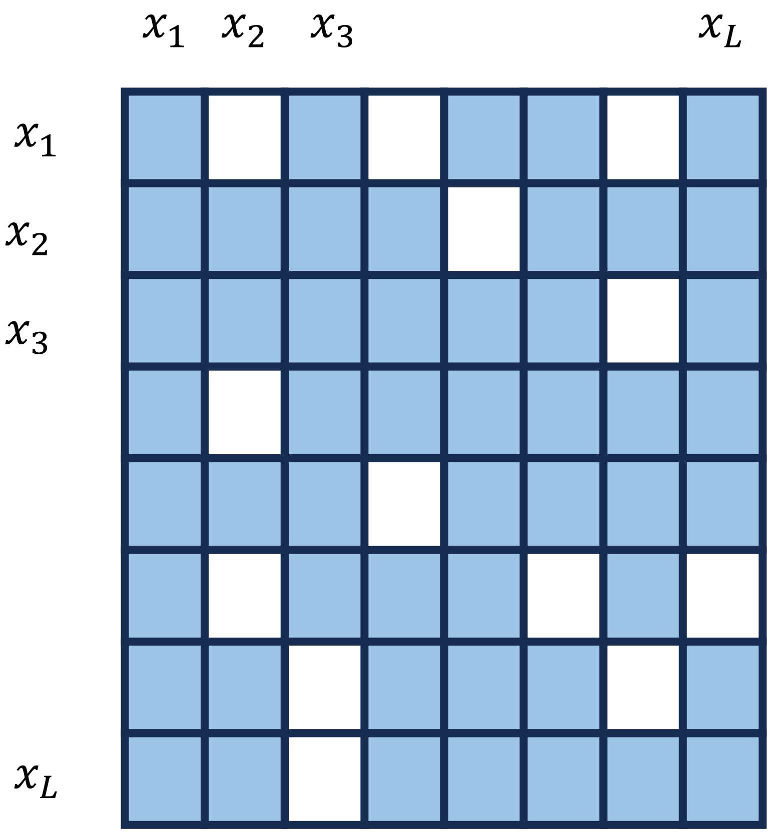
Each entry in the attention mask M corresponds to 1 dot-product
l^2 entries in total
Sparsity factor s determines how many dot-products actually need to be computed

QK Product with Attention Mask



M

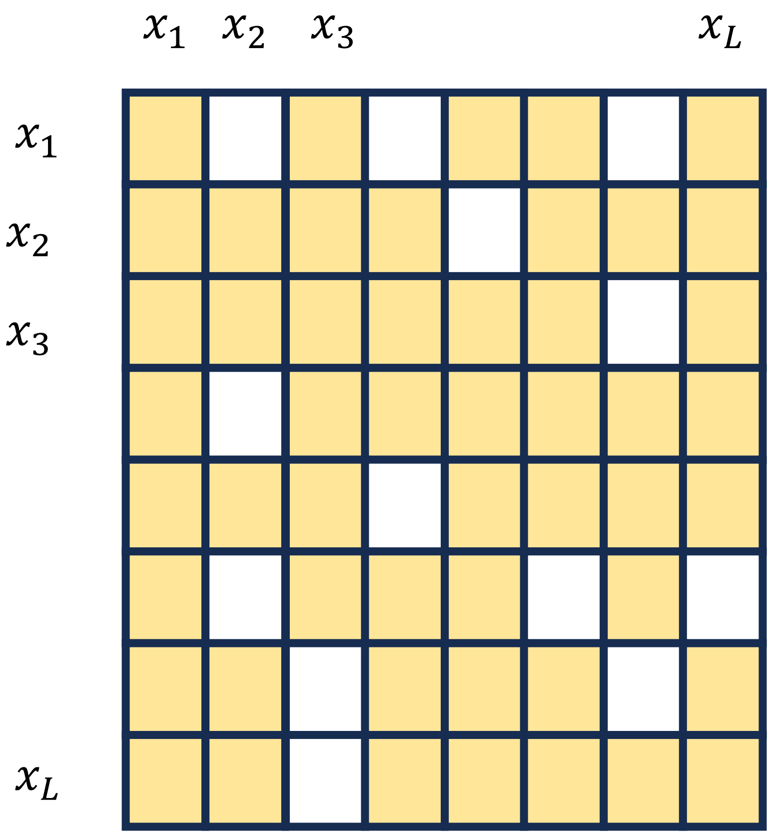
QK^T



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

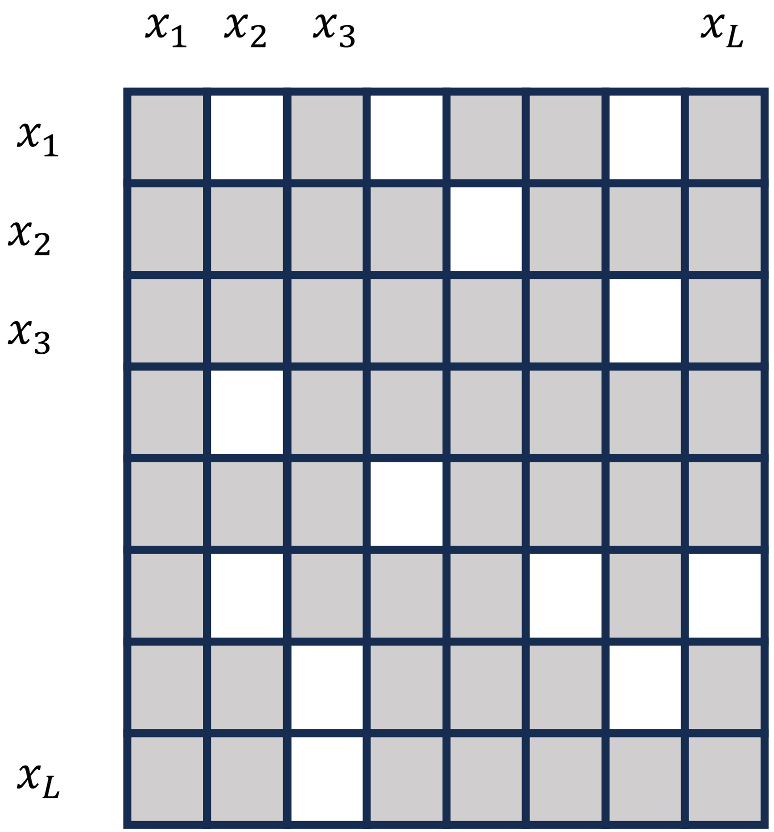
In practice, elements corresponding to 0 mask are set to $-\infty$: Why? Will ask in WA 3

Softmax with Attention Mask



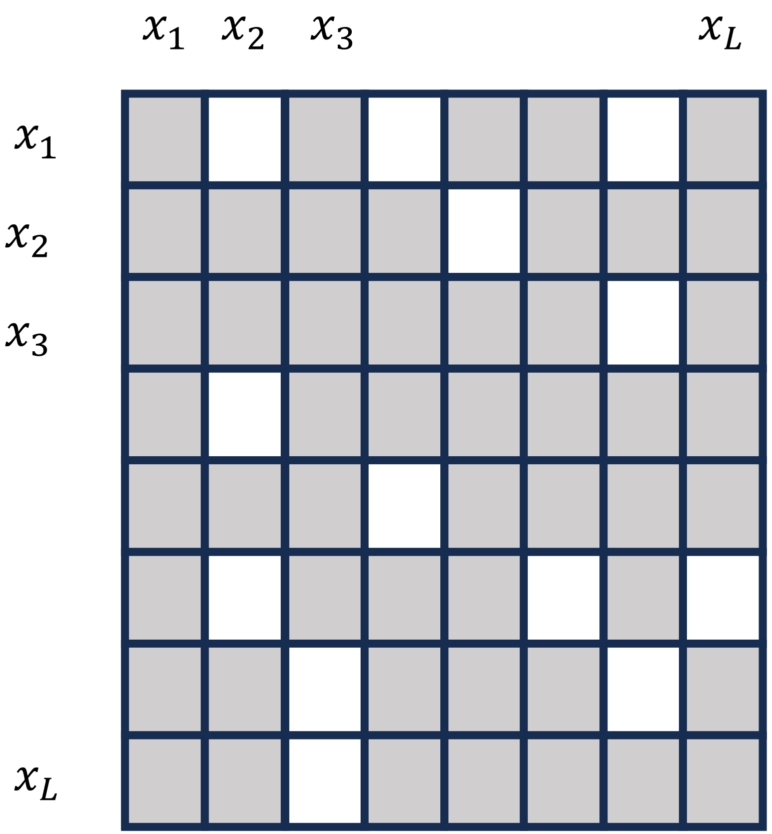
M

$Z = softmax(QK^T)$

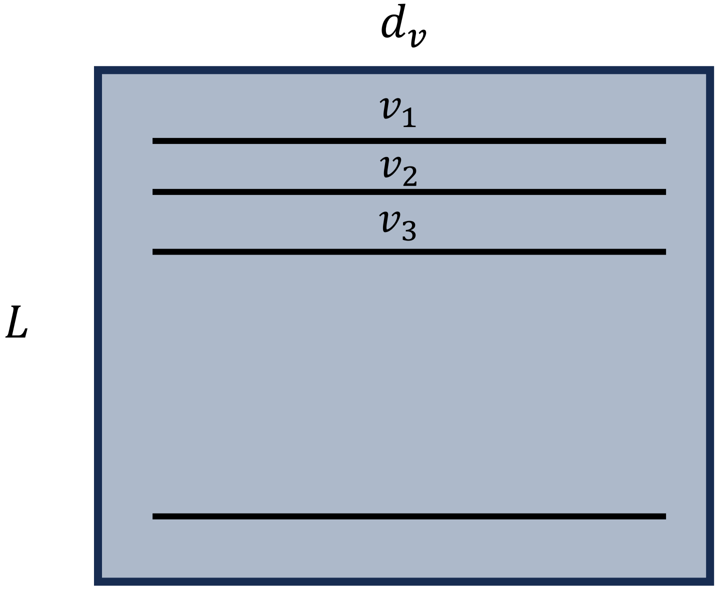


Softmax will replace all $-\infty$ elements with 0: How? Will ask in WA 3.
The pattern of 0 and non-zero still remains the same

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

Theoretical Reduction in Computation

- For sparsity factor $0 \leq s \leq 1$ defined as the ratio of the number of non-zero elements to the number of zero elements

- KQ^T : $O(sL^2 \times d_k)$: How? Will be a question in WA3

- ZV: $O(sL^2 \times d_v)$: How? Will be a question in WA3

- For very small values of s, significant reduction in computations

Reduction in Computations

- In practice, pytorch algorithms do not fully exploit this sparsity.
  - Recall our implementation that in the last question of Exam 1, it was not work-optimal
- We will discuss this issue and potential solutions (some developed by my students) in the next class (or two)

Research on Sparse Transformers

- Two main directions
- #1 Build masks that reduce the computations but still preserve accuracy
  - Today's class
- #2 Exploit Sparsity on Hardware such as GPUs
  - Next one or two classes

Outline

- Sparse Transformers Basics
- Sparse Masks

Types of Attention Masks

- Two main types

- Dynamic Attention: Only consider query-key pairs for attending that are the most similar
  - No explicit attention mask
  - Pairs to attend determined at runtime

- Static Attention: Fixed attention mask used to determine attending query-key pairs

Types of Attention Masks

- Two main types

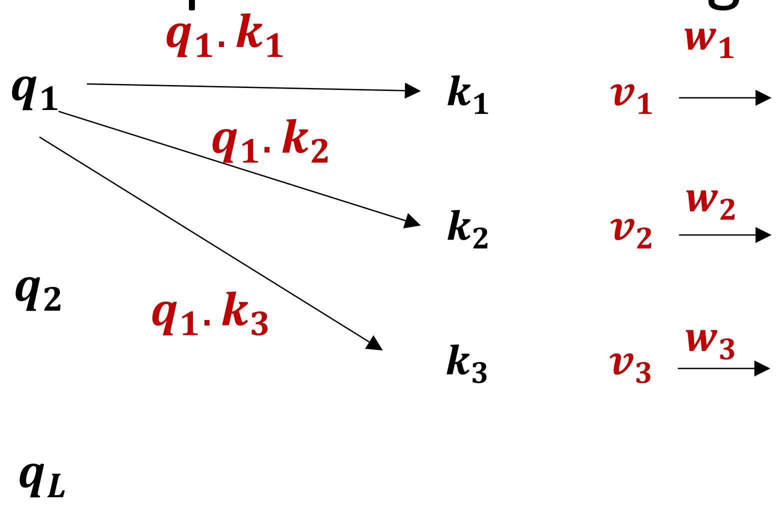
- **Dynamic Attention: Only consider query-key pairs for attending that are the most similar**
  - No explicit attention mask
  - Pairs to attend determined at runtime

- Static Attention: Fixed attention mask used to determine attending query-key pairs

Reformer

- Key Idea: Reduce the $O(L^2)$ complexity of attention calculation by only considering key values pairs that are more similar

- Less similar key-value pairs -> small weights -> less information gain



Kitaev, N., Kaiser, L., & Levskaya, A. (2019, September). Reformer: The Efficient Transformer. In *International Conference on Learning Representations*.

Reformer

- How do we know what key-value pairs are more similar without computing the values???

Reformer

- How do we know what key-value pairs are more similar without computing the values

- Locality Sensitive Hashing

Locality Sensitive Hashing

- A technique to perform efficient nearest neighbor search
- Hashes “similar” items into same “buckets” with “high” probability

Locality Sensitive Hashing

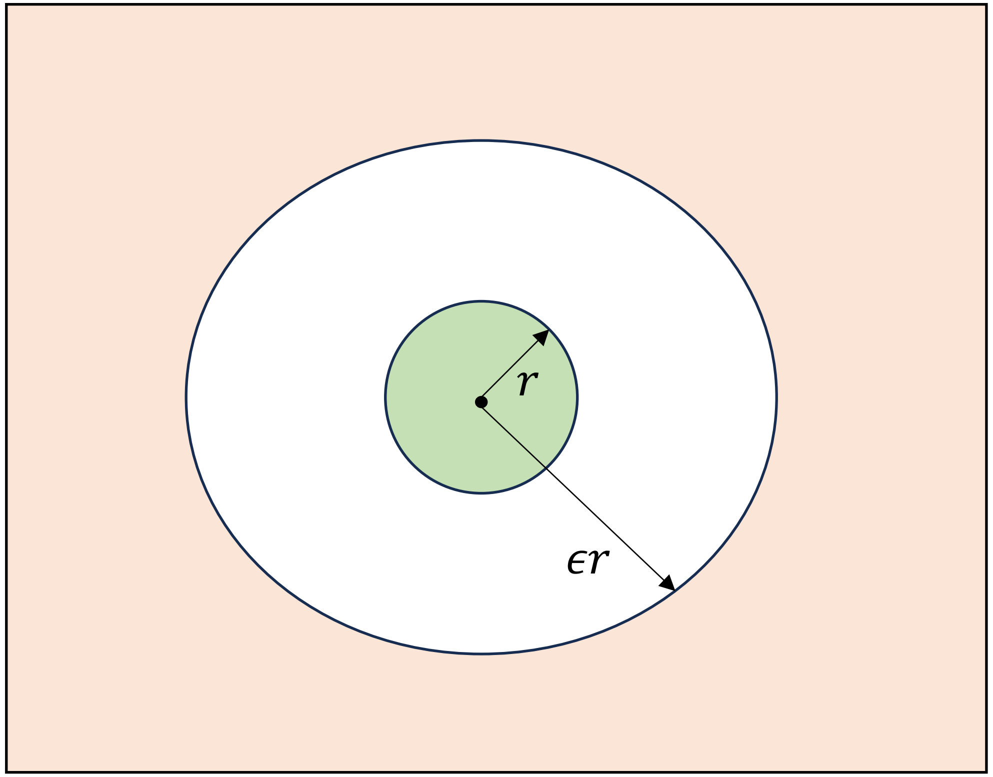
- Consider a set of hash functions $h \in \mathcal{F}$, where $h: M \to S$
  - M: some metric space with distance metric d
  - S: buckets

- Consider the following additional parameters
  - $r > 0$: a threshold value
  - $\epsilon > 0$: an approximation factor
  - p_1, p_2: probability values

Locality Sensitive Hashing

- The set of hash functions $\mathcal{F}$ is called an LSH family if,
- For any two points $a, b \in M$ and a randomly selected hash function $h \in \mathcal{F}$
- If $d(a, b) \leq r$, then $h(a) = h(b)$ with probability $\geq p_1$
- If $d(a, b) > \epsilon r$, then $h(a) = h(b)$ with probability $\leq p_2$

Locality Sensitive Hashing



High Probability $> p1$ of conflict

Low Probability $< p2$ of conflict

M

Locality Sensitive Hashing

- Key Design Parameters:

- Determining the hash functions that can achieve this property

- Look at this survey for more information: Jafari, Omid, Preeti Maurya, Parth Nagarkar, Khandker Mushfiqul Islam, and Chidambaram Crushev. "A survey on locality sensitive hashing algorithms and their applications." *arXiv preprint arXiv:2102.08942* (2021).

Reformer

- Hashing Scheme: angular locality sensitive hashing
- For a key with dimension d^k, obtain b values as follows:
- Fix a random matrix R of size $d^k \times b/2$
- $h(x) = \arg\max\{xR; -xR\}$; -> concatenation

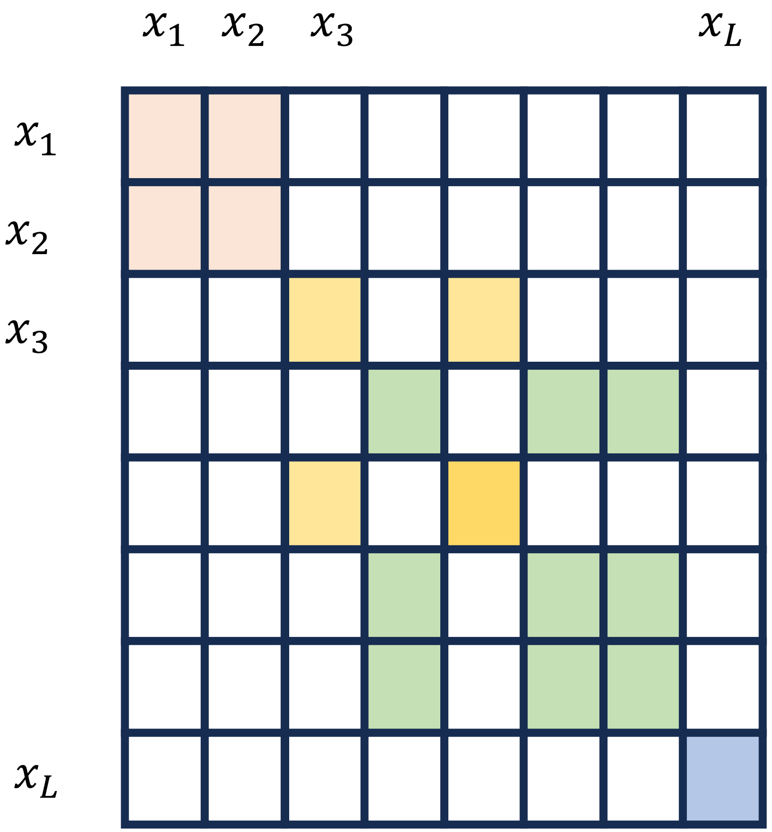
Reformer

- Intuition: The dimension with largest value for points close to each other should be same even after projection.

- For formal proof, and details on how to create the random matrix, and other parameters, see:
  - Andoni, A., Indyk, P., Laarhoven, T., Razenshteyn, I., & Schmidt, L. (2015). Practical and optimal LSH for angular distance. *Advances in neural information processing systems*, *28*.

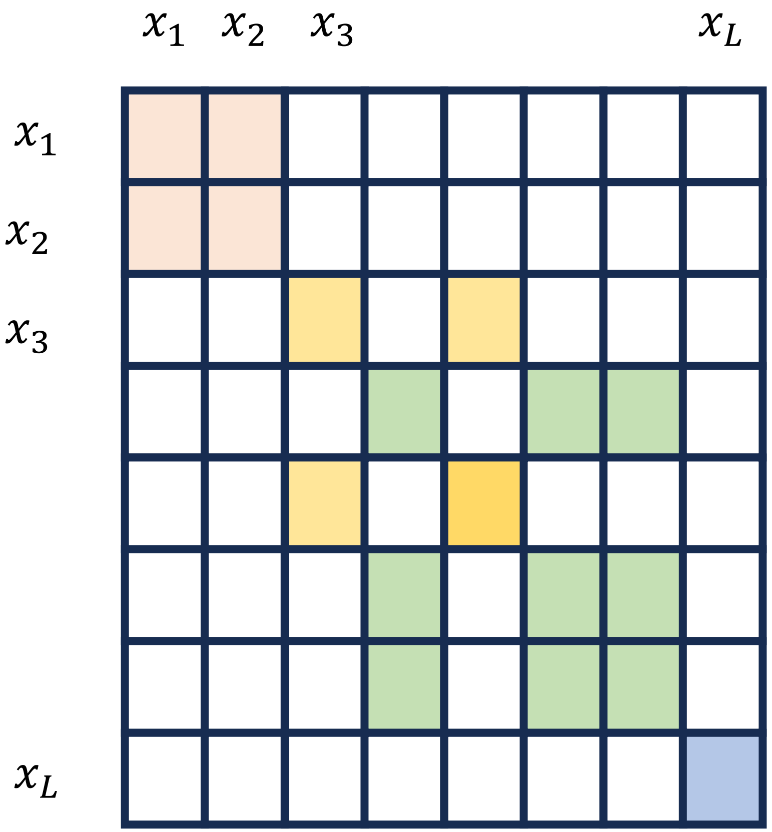
Reformer

- Attention with LSH
- Sparse adjacency matrix
- However, inefficient for computation
  - Why???



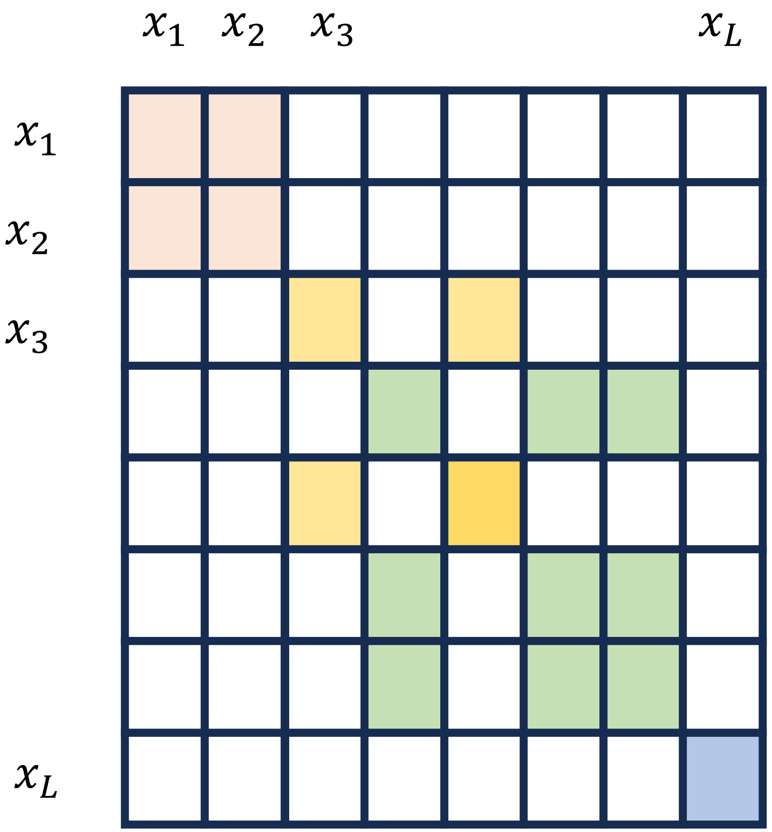
Reformer

- Attention with LSH
- Sparse adjacency matrix
- However, inefficient for computation
  - Too many random accesses
- How do we fix???



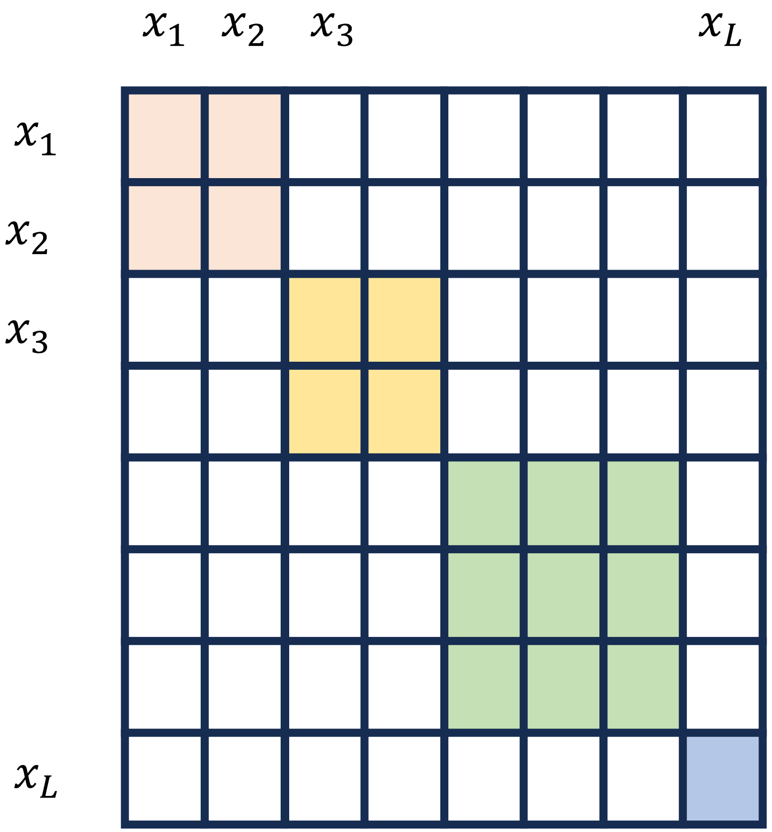
Reformer

- Attention with LSH
- Sparse adjacency matrix
- However, inefficient for computation
  - Too many random accesses
- Sort the sequence to cluster buckets together



Reformer

- Sort the sequence to cluster buckets together
- Perform attention mechanism on blocks
  - Smaller matrix multiplications
- (Make a note of this sorting technique, we will discuss more sophisticated methods in the next one/two classes)



Reformer

- Other Contributions:

- Multi-round LSH attention
  - Single hash computation may skip some connections due to probabilistic nature
  - Do n different (and parallel) LSH computations to find out the neighbors for populating the adjacency matrix

- $N_i = \cup_{r=1}^{n} N_i^r$
- N_i^r: neighbors of query i obtained in ith LSH computation
- N_i: all neighbors of query i used to populate the adjacency matrix

Types of Attention Masks

- Two main types

- Dynamic Attention: Only consider query-key pairs for attending that are the most similar
  - No explicit attention mask
  - Pairs to attend determined at runtime

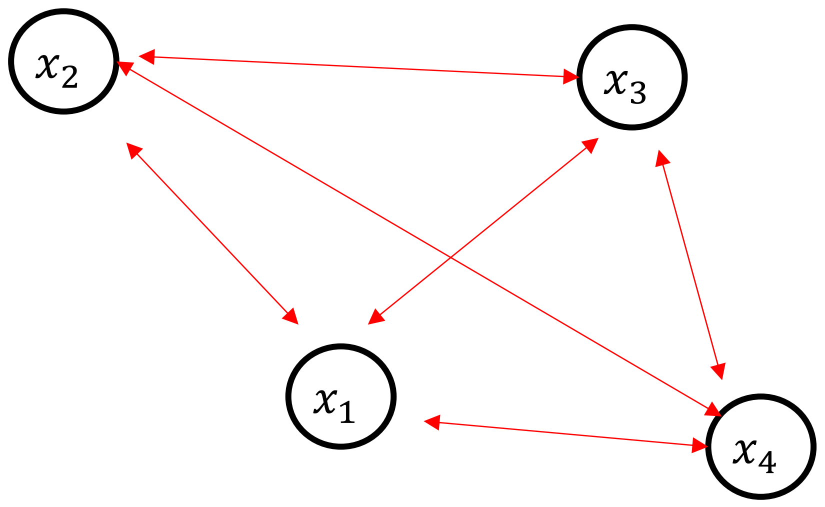
- **Static Attention: Fixed attention mask used to determine attending query-key pairs**

Big Bird

- Key Idea:
  - Think of a Transformer model as information flows between tokens of the input sequence
  - A transformer block represents single edge connections in a graph with L nodes
  - A Transformer model with L_m layers can be thought of as information flow using paths of length L_m

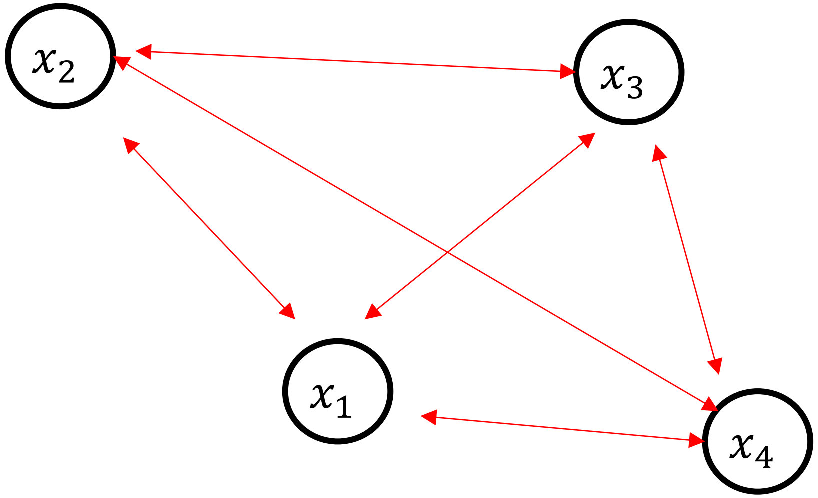
Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., ... & Ahmed, A. (2020). Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, *33*, 17283-17297.

Big Bird



Transformer: Information flow between all tokens

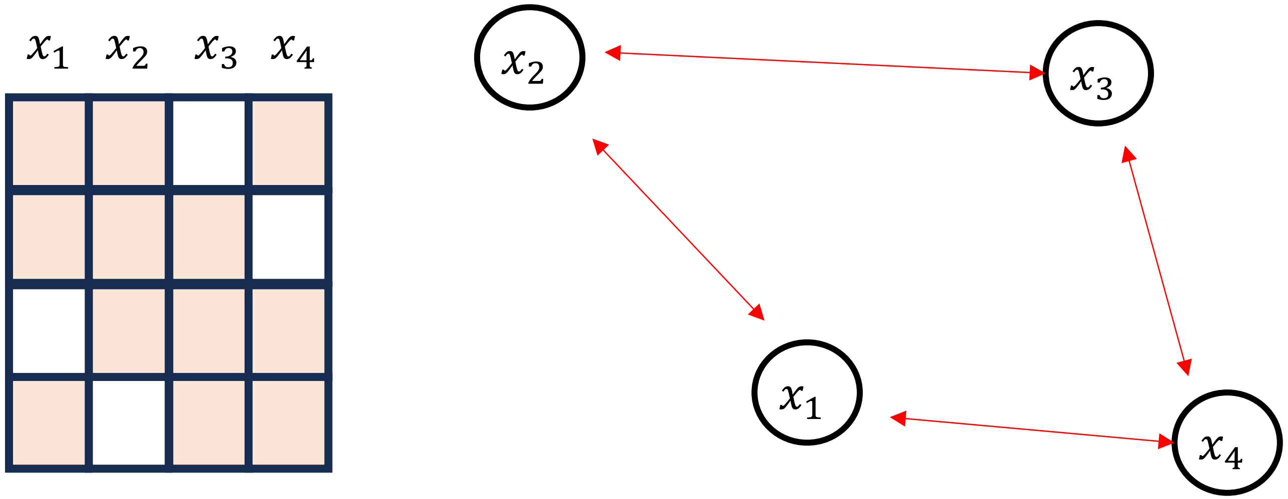
Big Bird



Fully Connected Attention: Information flow between all tokens in Single Layer

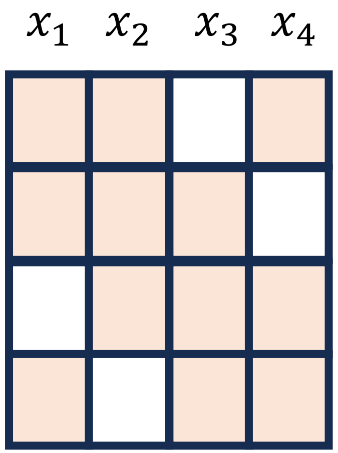
Big Bird

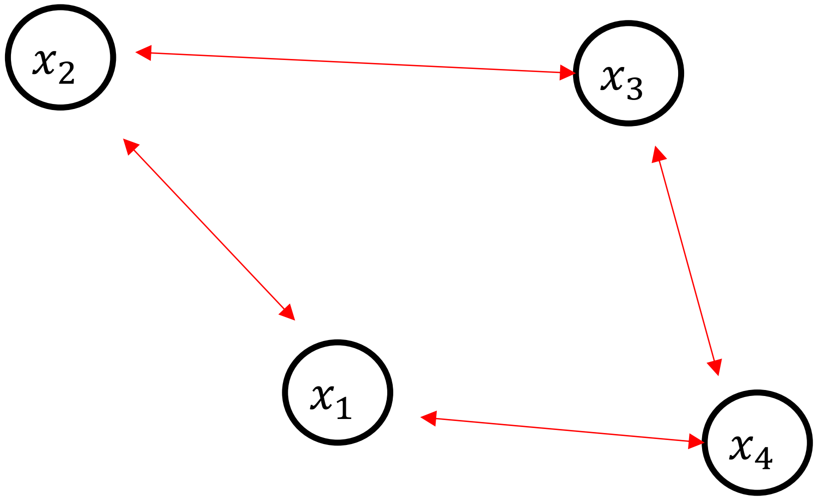
- Consider the following adjacency matrix for attention mechanism
- Which two tokens have no information flow in a single layer?



Big Bird

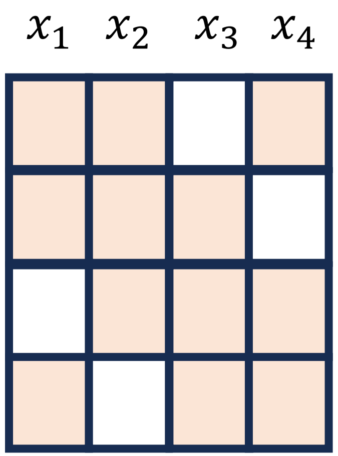
- Consider the following adjacency matrix for attention mechanism
- $x_1 : x_3,\ x_2 : x_4$
- How many layers will be needed to ensure information flows between them???

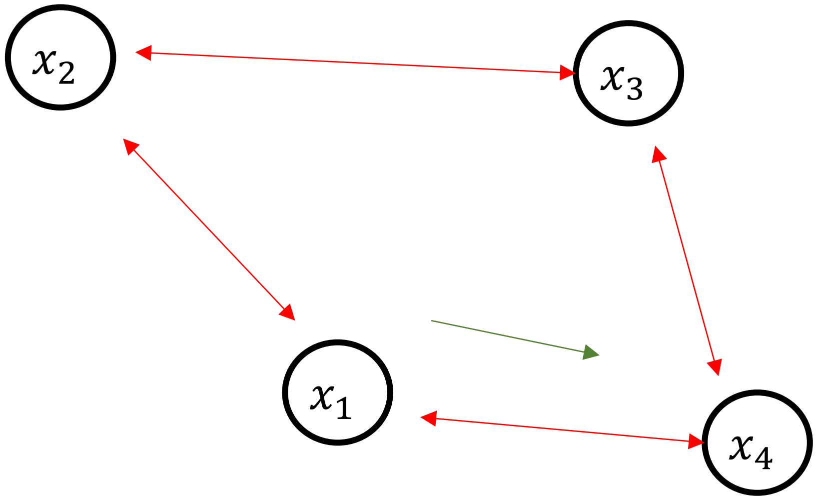




Big Bird

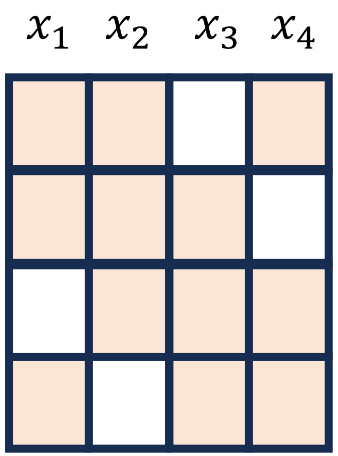
- Consider the following adjacency matrix for attention mechanism
- $x_1 : x_3, x_2 : x_4$
- ≥ 2

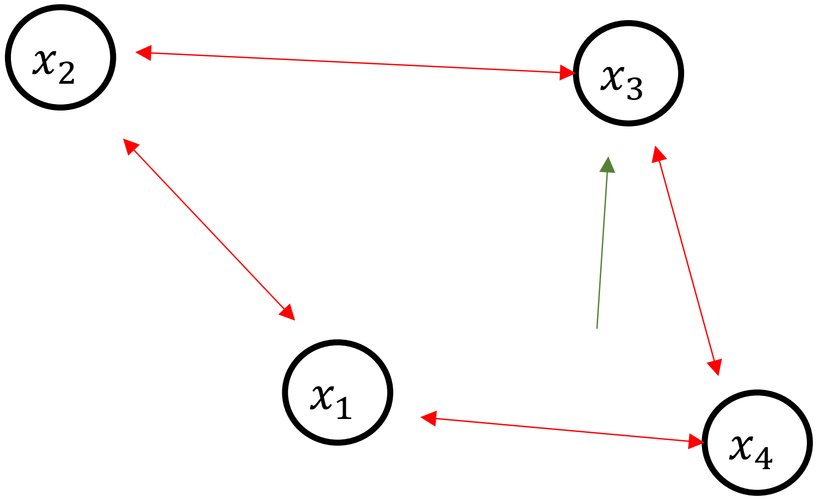




Big Bird

- Consider the following adjacency matrix for attention mechanism
- $x_1 : x_3, x_2 : x_4$
- ≥ 2





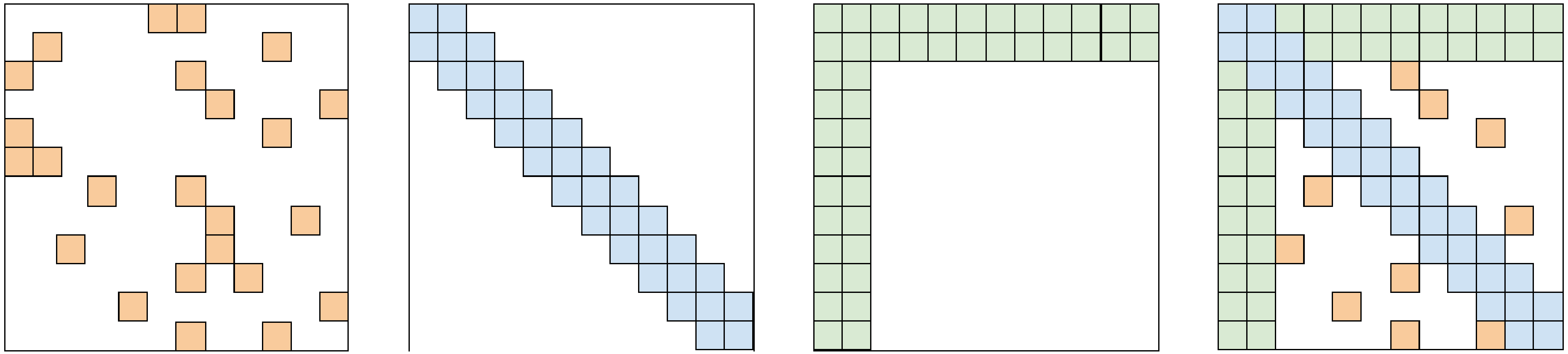
Big Bird

- Key Idea:
  - Think of a Transformer model as information flows between tokens of the input sequence
  - A transformer block represents single edge connections in a graph with L nodes
  - A Transformer model with L_m layers can be thought of as information flow using paths of length L_m

- Objective: Find graphs (adjacency matrix) with $O(L)$ nodes and small path lengths

Big Bird

- Consists of three types of connections



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Big Bird

- Random Attention
  - Inspired by graph theory (graph sparsification problem)
  - Random connections between tokens

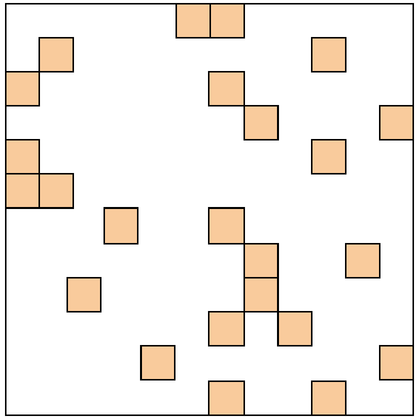
- Erdos-Renyi model for random graph construction
  - Each edge randomly chosen with fixed probability

- Property: Shortest path between any two nodes is logarithmic in the number of nodes.

- So, for L tokens, only $\mathrm{L_m} = O(\log L)$ layers needed for full information flow

Big Bird

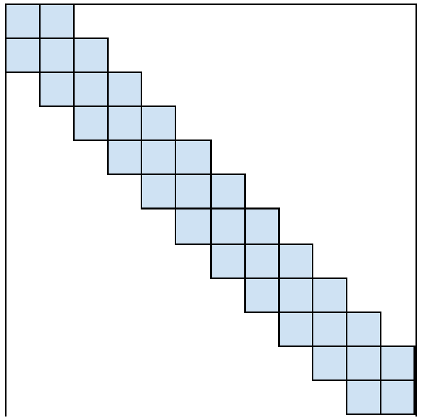
- Random Attention: Select edges with a fixed probability



(a) Random attention

Big Bird

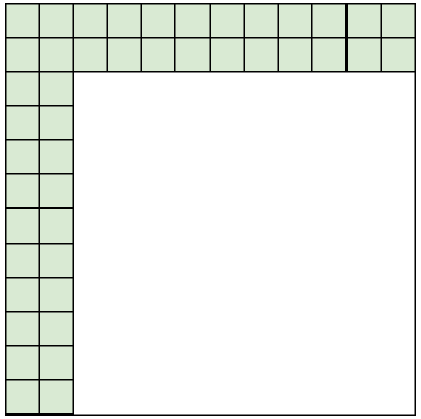
- Windowed Attention:
  - Inspired from linguistics/computational biology
  - Sequences have locality of reference
  - w: window size is the key parameter



(b) Window attention

Big Bird

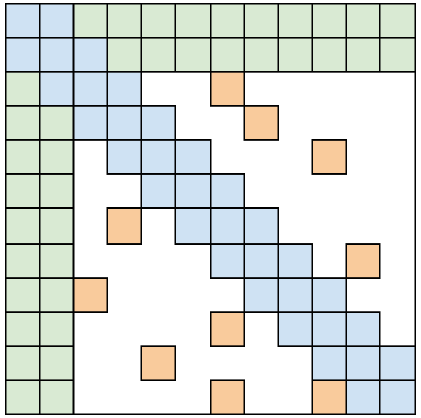
- Global Attention: Marks some token as global (or add new tokens) and connects them to all the tokens in the sequence
  - Obtained using the theoretical analysis in the Reformer paper
  - g: number of global tokens is the key parameter



(c) Global Attention

Big Bird

- Combining everything, we get
- $O(L)$ connections – proof in the paper. We will not discuss here.

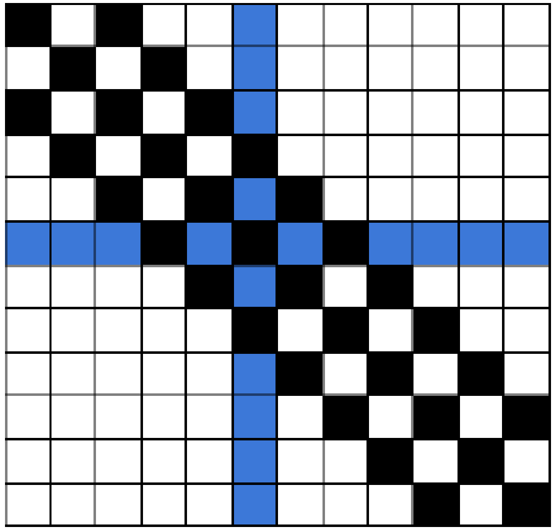


(d) BIGBIRD

Other Attention Patterns

- 1D Dilated Attention: Create uniform gaps of a set size between attended tokens across a row
- Window size: w
- Dilation factor: r

```
if ((abs(i - j) < w) && (abs(i - j) % r == 0)) {
  return 1;
} else {
  return 0;
}
```



I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The longdocument transformer," arXiv preprint arXiv:2004.05150, 2020

Other Attention Patterns

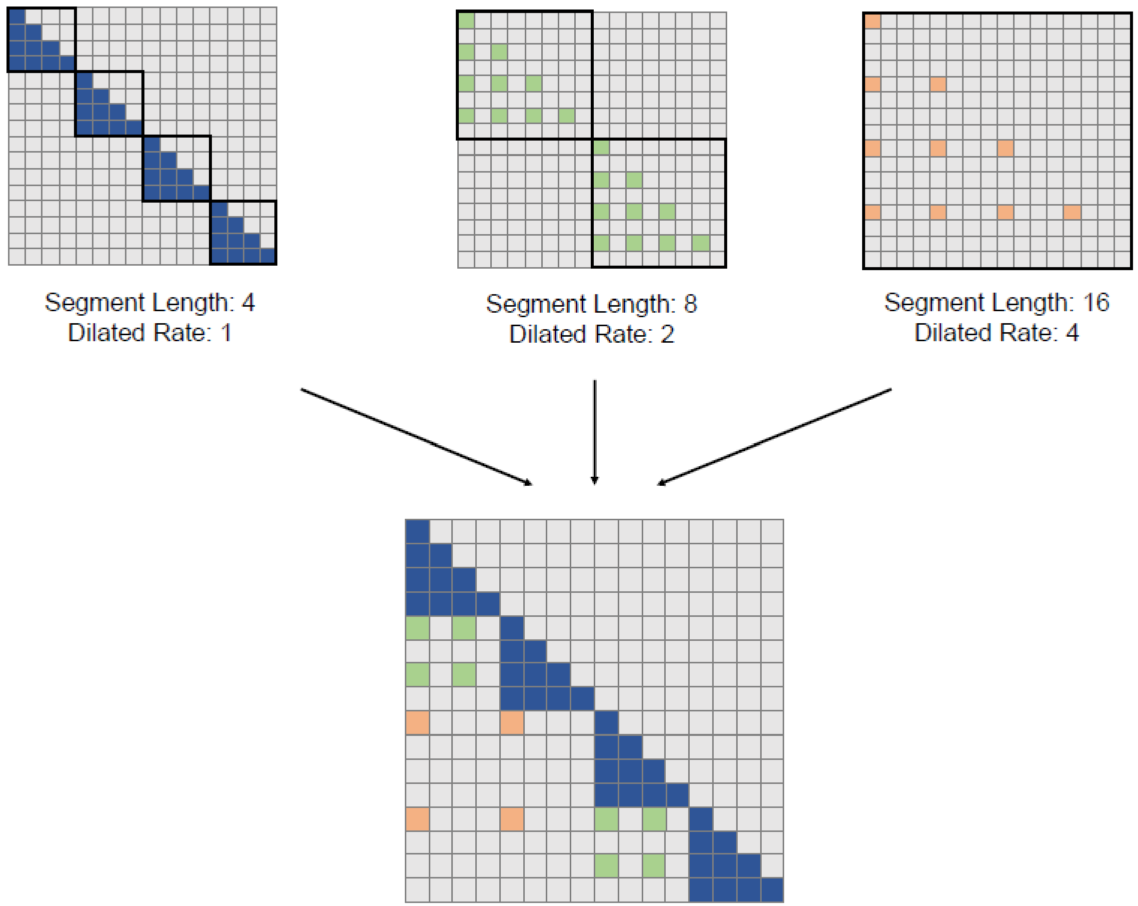
- 2D Dilated Attention: Dilation along both dimensions
  - Probability of query and key attending exponentially reduces with distance between the corresponding tokens

```
if (floor(i/(L/w)) == floor(j/(L/w))) {
  i_w = i % w;
  j_w = j % w;
  if ((i_w % r == 0) && (j_w % r == 0)) {
    return 1;
  } else {
    return 0;
  }
} else {
  return 0;
}
```

J. Ding, S. Ma, L. Dong, X. Zhang, S. Huang, W. Wang, N. Zheng, and F. Wei, "Longnet: Scaling transformers to 1,000,000,000 tokens," 2023. [Online]. Available: https://arxiv.org/abs/2307.02486

Other Attention Patterns

- 2D Dilated Attention
- Understand the equation
- May ask a question in WA 3



Next Class

- 10/30 Lecture 18
  - Accelerating Transformer Model: Sparse Transformers II
    - Hardware Acceleration of Sparse Transformers

Thank You

- Questions?

- Email: sanmukh.kuppannagari@case.edu