CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

CSDS 451: Designing High Performant Systems for AI

Lecture 16

10/23/2024

Sanmukh Kuppannagari

sanmukh.kuppannagari@case.edu

https://sanmukh.research.st/

Case Western Reserve University

Outline

- Flash Attention - Memory Access Aware Attention Computation

Outline

- Flash Attention - Memory Access Aware Attention Computation

Transformer Models

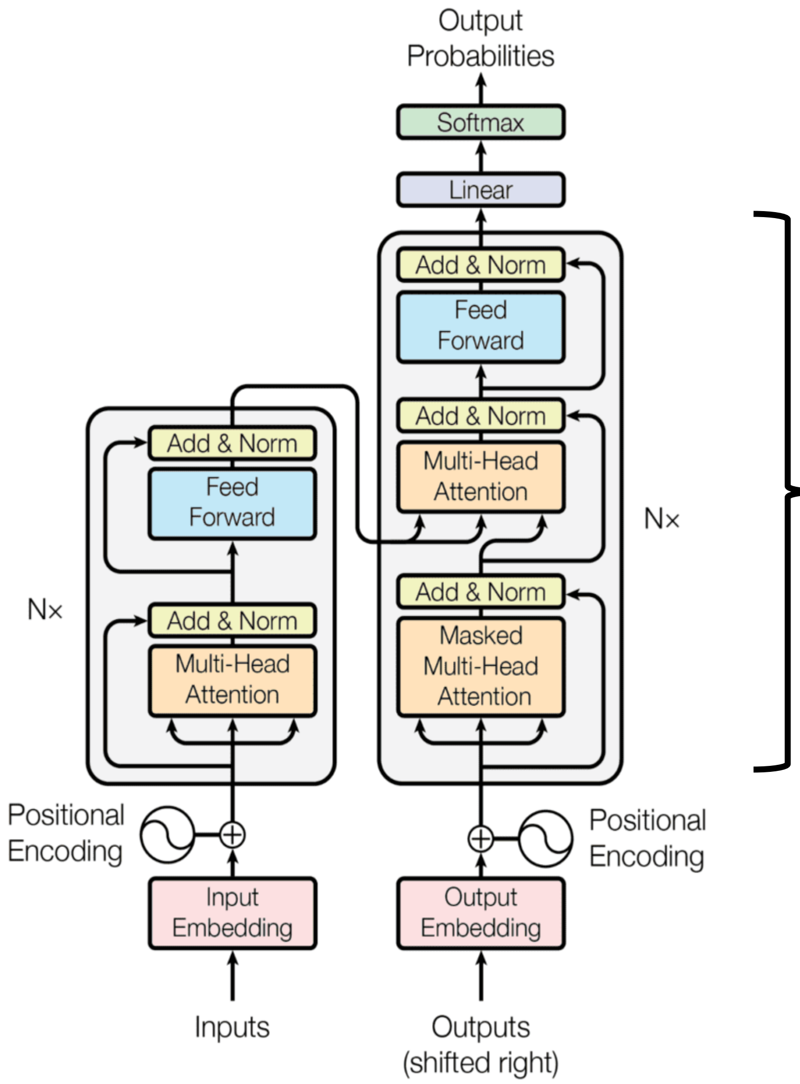
- Encoder-Decoder Architecture

- Encoder:
  - Map a sequence of input symbols $(x_1, x_2, \dots, x_n)$, to
  - A sequence of continuous representation $(z_1, z_2, \dots, z_n)$

- Decoder:
  - Given $(z_1, z_2, \dots, z_n)$ and output from previous iteration $(y_1^{t-1}, y_2^{t-1}, \dots, y_m^{t-1})$
  - Output $(y_1^t, y_2^t, \dots, y_m^t)$

Transformer Models

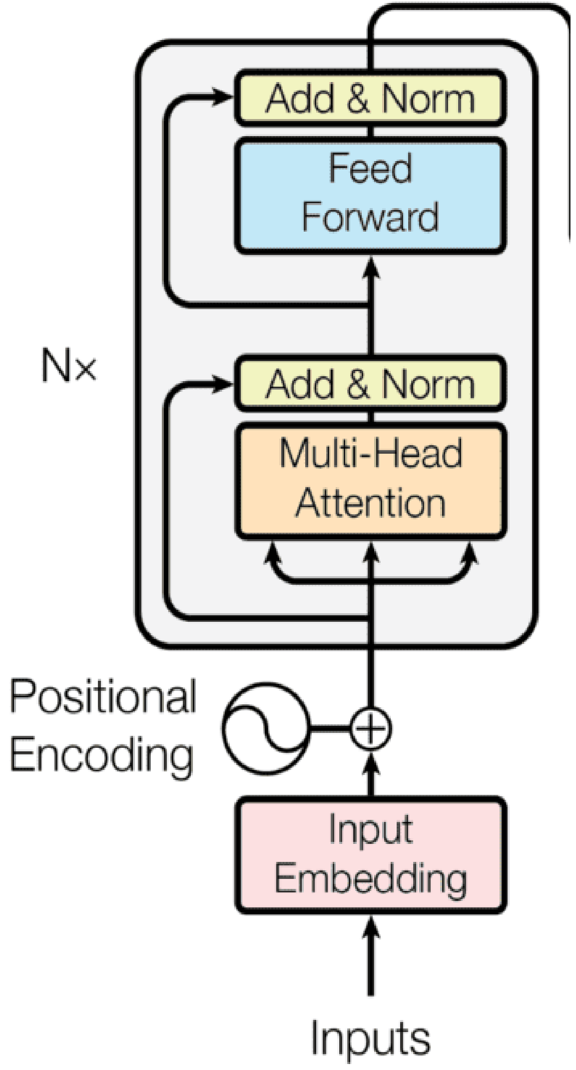
N Layers of Encoder Blocks



N Layers of Decoder Blocks

Recall: Multi-headed Attention

- Perform h attention computations
- Assuming dimension of embeddings is d_m (also called **model dimension**)
- Project Q, K, V using $W_i^Q \in R^{d_m \times d_k}, W_i^K \in R^{d_m \times d_k}, W_i^V \in R^{d_m \times d_v} \forall k \in \{1,2, \ldots, h\}$
- $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$
- $Concat(head_1, head_2, \ldots, head_h)W^O$ where $W^O \in R^{hd_v \times d_m}$



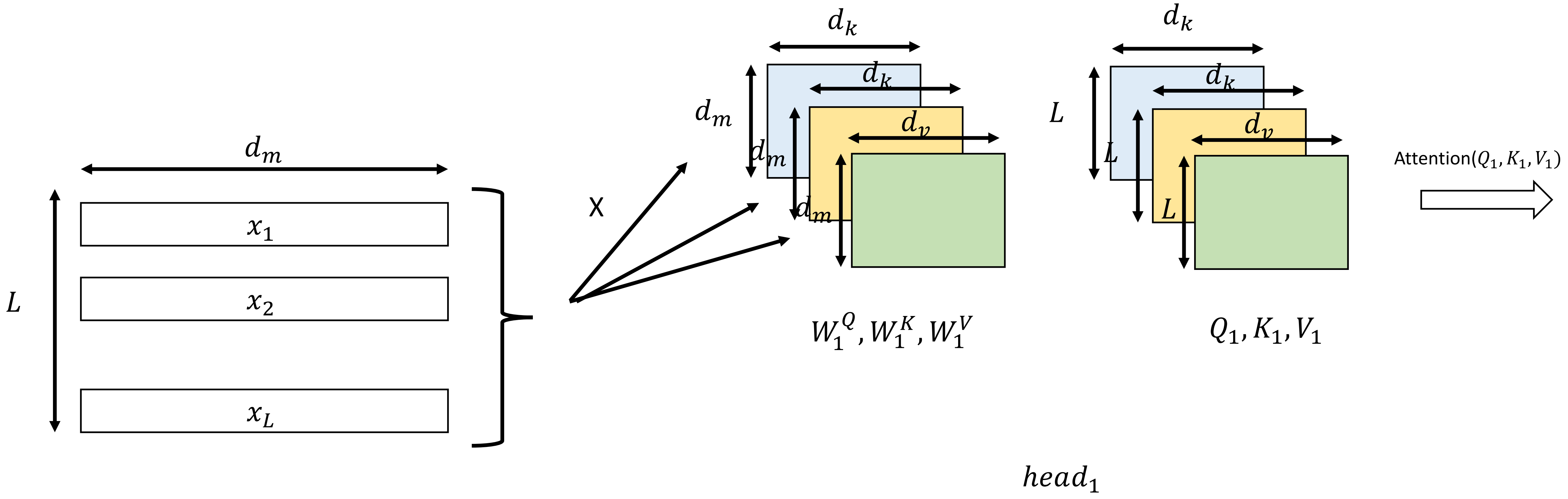
Recall: Multi-headed Attention

- Perform h attention computations
- Assuming dimension of embeddings is d_m (also called **model dimension**)
- Project X into Q, K, V using $W_i^Q \in R^{d_m \times d_k}, W_i^K \in R^{d_m \times d_k}, W_i^V \in R^{d_m \times d_v}\ \forall k \in \{1, 2, \dots, h\}$
- $head_i = Attention(Q, K, V)$
- $Concat(head_1, head_2, \dots, head_h)W^O$ where $W^O \in R^{hd_v \times d_m}$

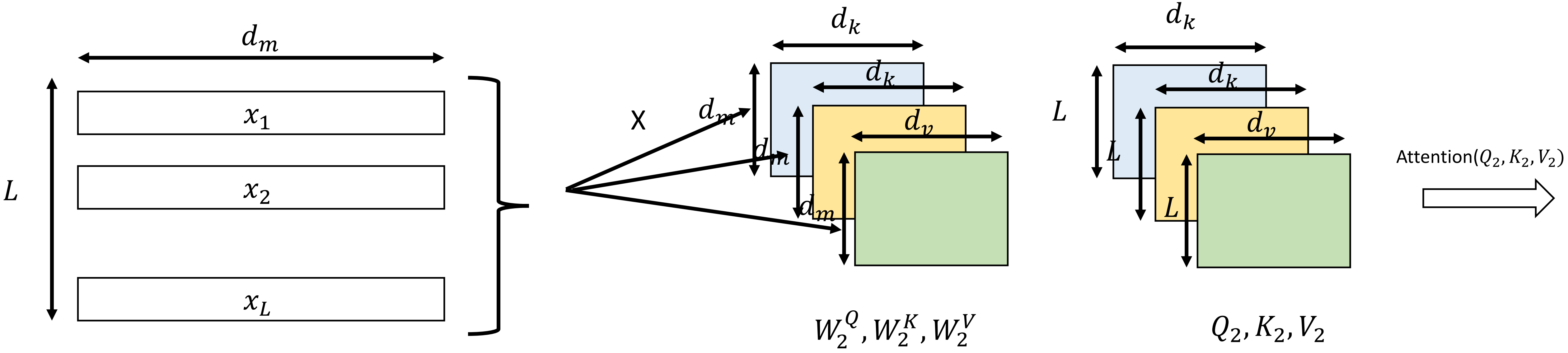
Most computation and memory intensive process.

We will focus on attention calculation for a single head. Multiple heads are processed in parallel using the same techniques

Recall: Multi-headed attention

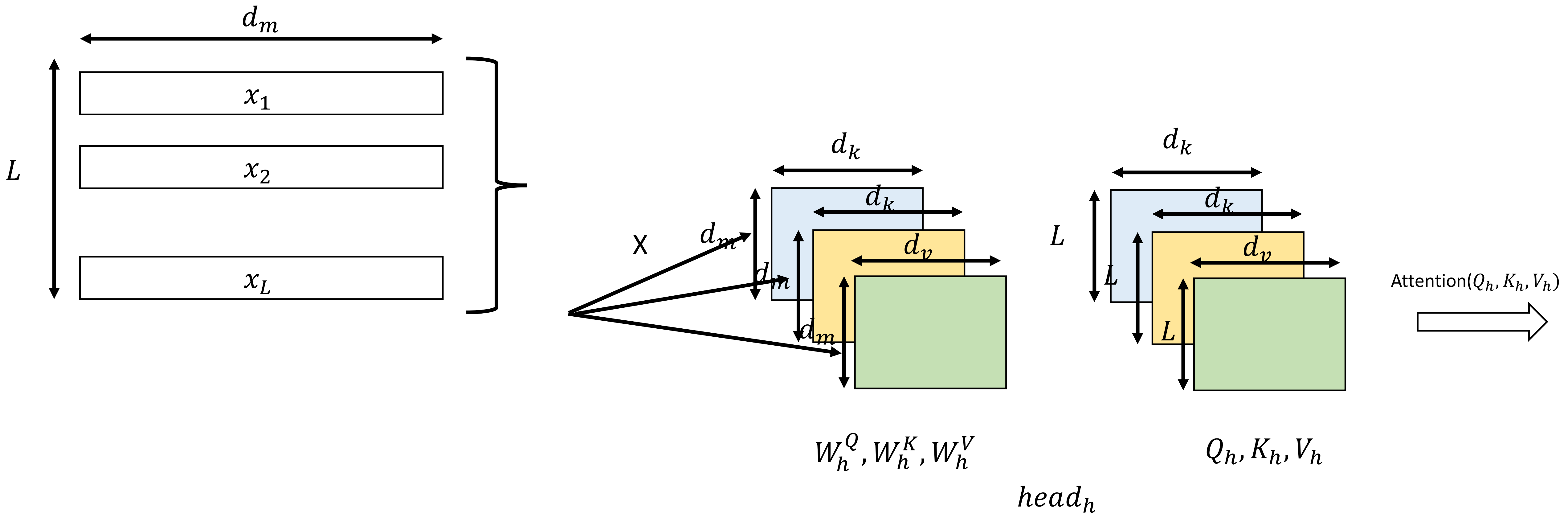


Recall: Multi-headed attention

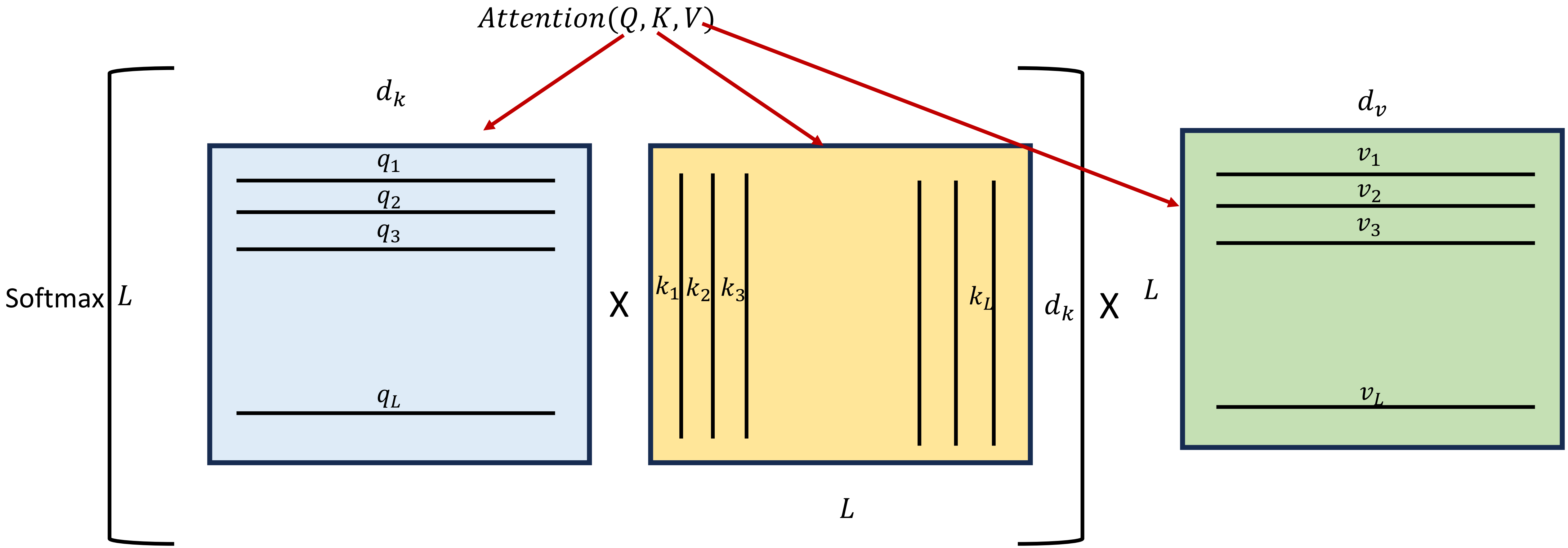


$head_2$

Recall: Multi-headed attention



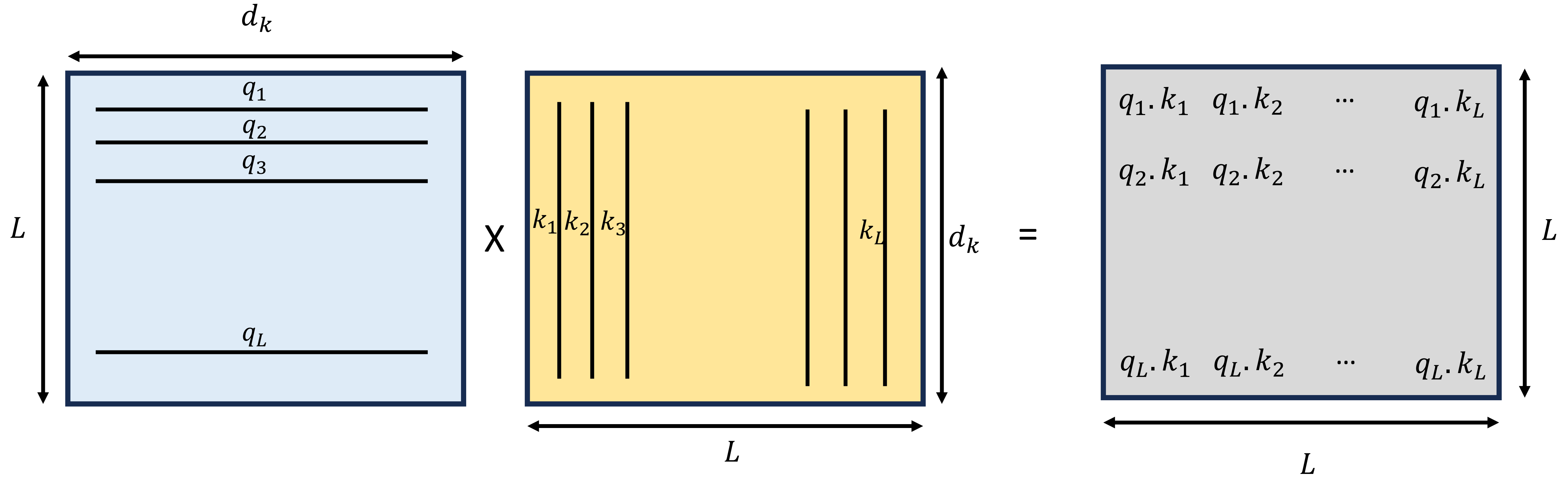
Zooming into Attention



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

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

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

Zooming into Attention

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

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

Exponent of dot product of query i with key j

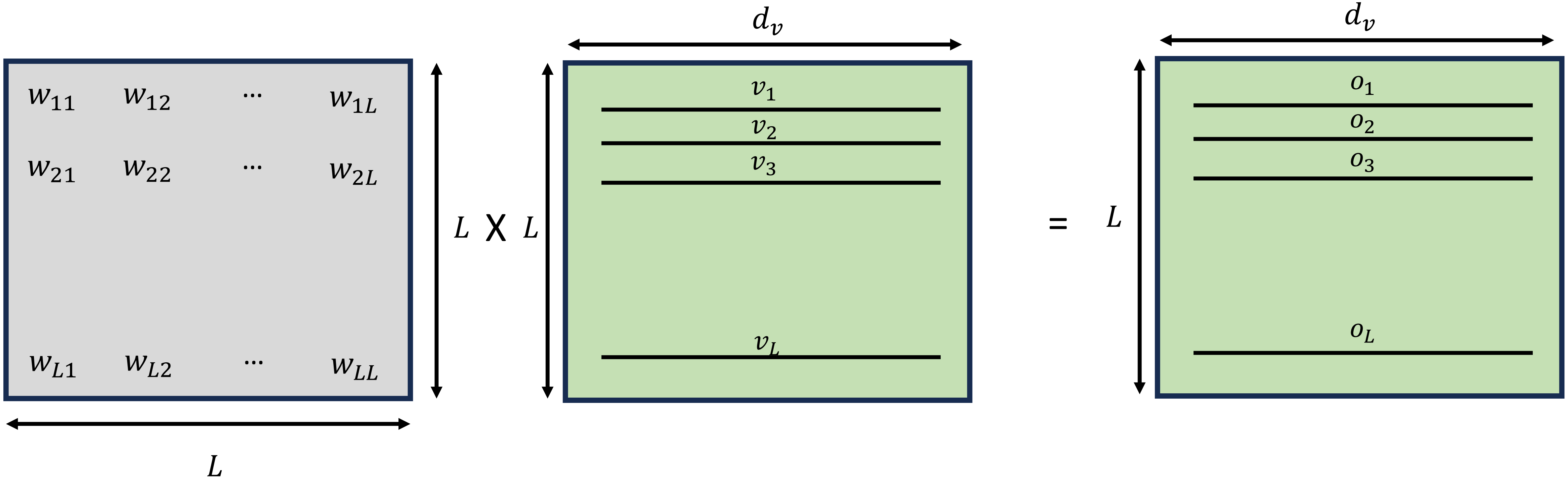
$$w_{ij} = \frac{e^{q_i.k_j}}{\sum_l e^{q_i.k_l}}$$

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

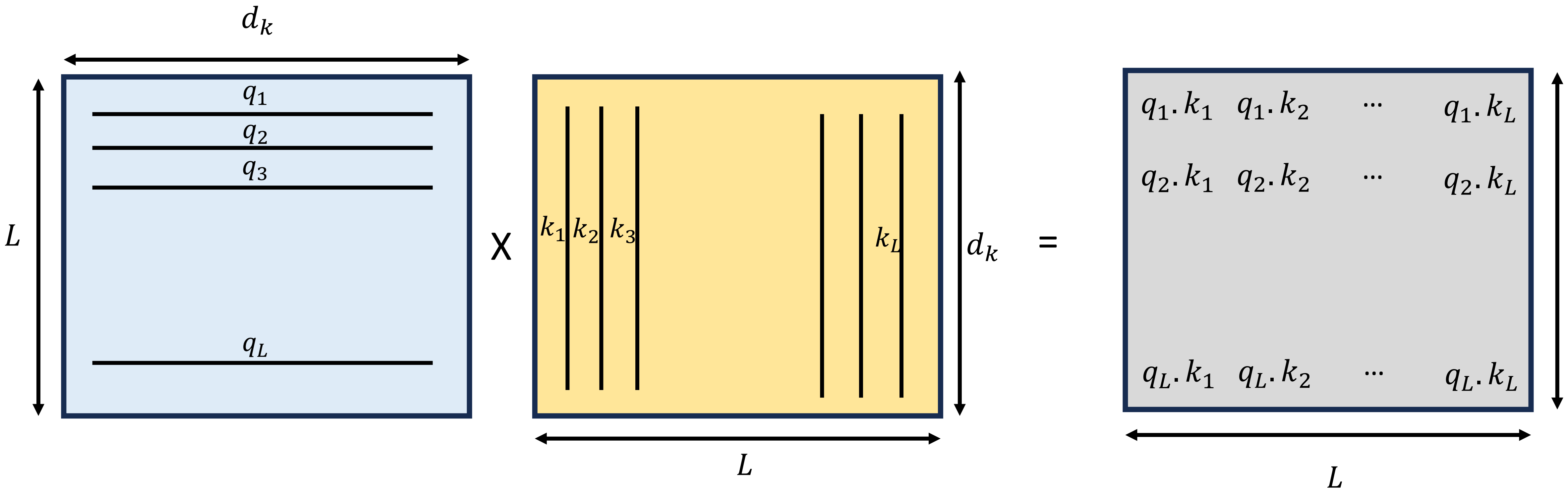
Zooming into Attention

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

Zooming into Attention

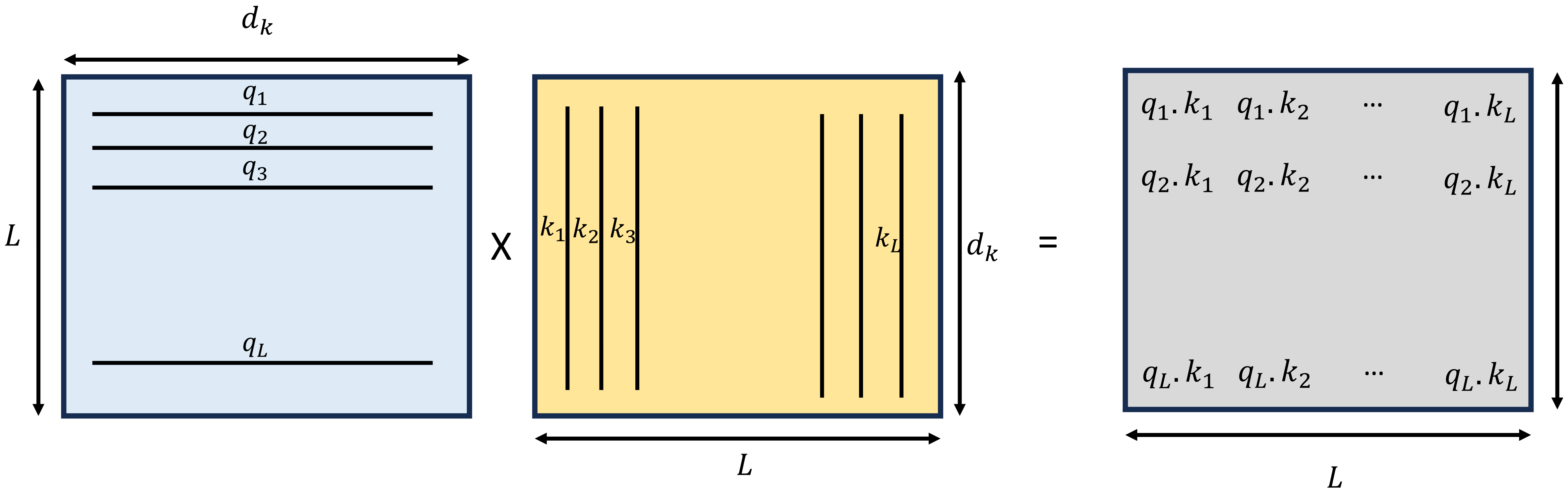


Problem: Lets Track the Memory Requirements



Memory Needed: ??

Problem: Lets Track the Memory Requirements



Memory Needed: $2Ld_k + L^2$

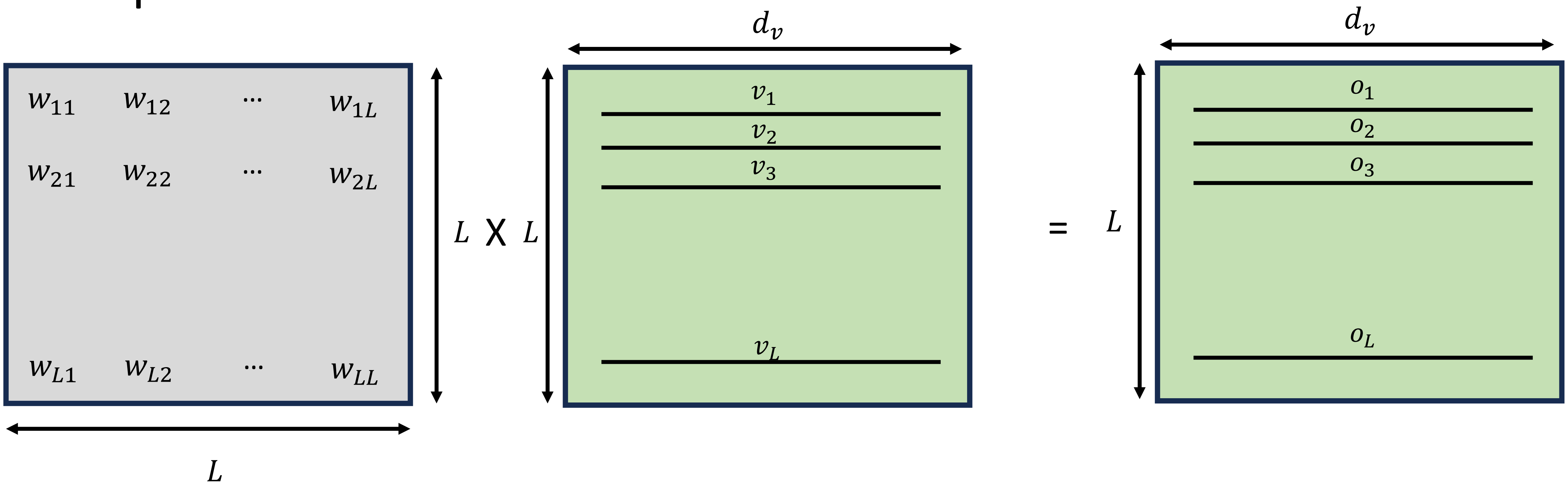
Problem: Lets Track the Memory Requirements

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

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

Memory Needed: L^2

Problem: Lets Track the Memory Requirements



Memory Needed: $L^2 + 2Ld_v$

Problem: Lets Track the Memory Requirements

- Atleast L^2 memory needed at any given time
- Assume, data type = 16 bits (2 Bytes)
- $L = 1024$
- Memory Needed = ??

Problem: Lets Track the Memory Requirements

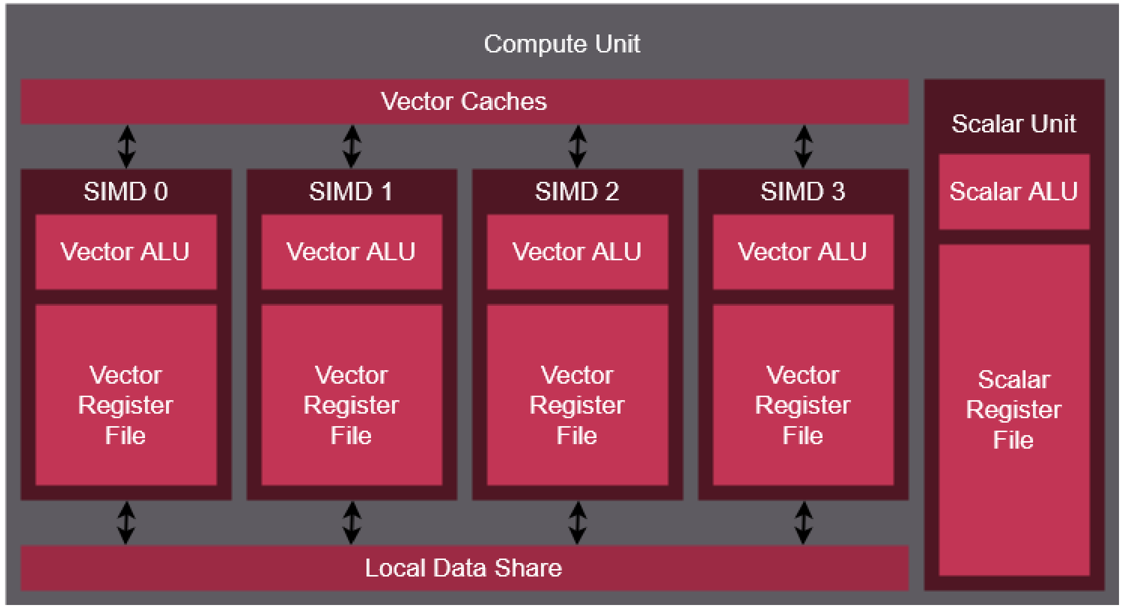
- Atleast L^2 memory needed at any given time
- Assume, data type = 16 bits (2 Bytes)
- $L = 1024$
- Memory Needed = 2 MB

Recall: Compute Unit

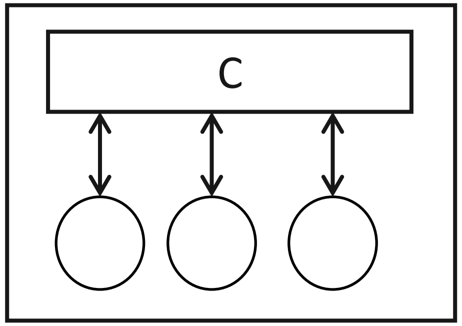
- Each VALU consists of Vector Register File
  - 64KB - 256 total registers – each register is 64 4-byte-wide entries
  - Private to each compute core

- Each CU consists of Local Data Share (LDS)
  - 64 KB – Shared Cache in our GPU model
  - Can be used to share data between all threads mapped to the same CU

- A smaller Vector Cache/L1 Cache is also available, not user controllable



=



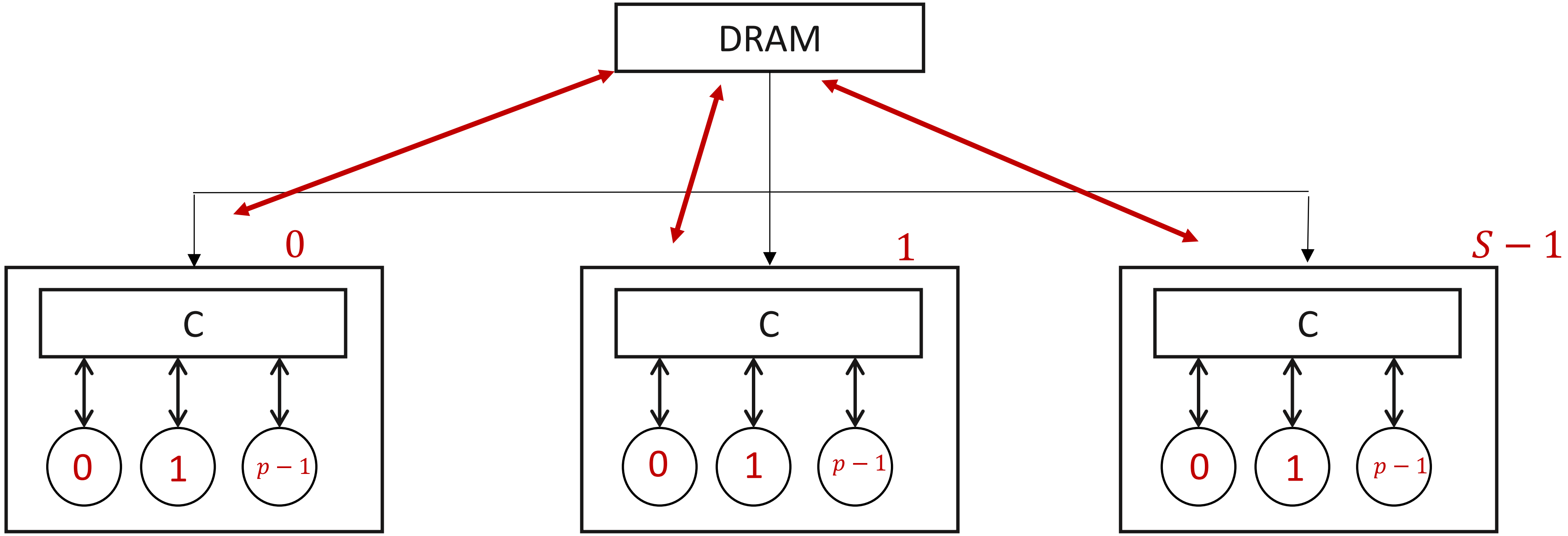
Problem

- 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

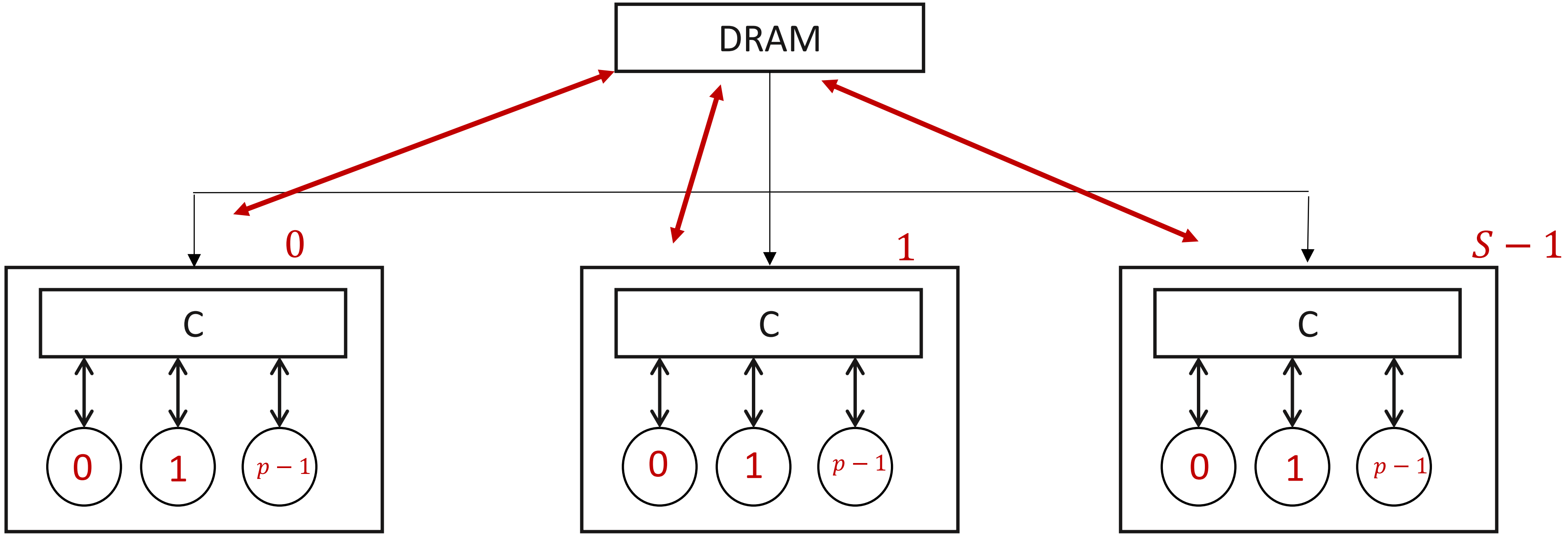
- This was the standard implementation in Pytorch before **FlashAttention**

Recall: GPU Modeling



- Excessive memory transfers on the red arrow
- From our GPU Model:
  - Time taken to transfer data from processor to cache = ??
  - Time taken to transfer data from cache to DRAM = ??

Recall: GPU Modeling



- Excessive memory transfers on the red arrow

- From our GPU Model:
  - Time taken to transfer data from processor to cache = 1 cycle
  - Time taken to transfer data from cache to DRAM = latency + data rate cycles

FlashAttention: Resources

- **Original FlashAttention:** Dao, T., Fu, D., Ermon, S., Rudra, A., & Ré, C. (2022). Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, *35*, 16344-16359.

- **FlashAttention2:** Dao, Tri. "Flashattention-2: Faster attention with better parallelism and work partitioning." *arXiv preprint arXiv:2307.08691* (2023).

- **FlashAttention3:** Shah, J., Bikshandi, G., Zhang, Y., Thakkar, V., Ramani, P., & Dao, T. (2024). Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *Advances in Neural Information Processing Systems*, *37*, 68658-68685.

FlashAttention

- Key Idea: Memory Access Pattern Aware Attention
- (They mention I/O aware attention, but it maybe a misnomer. I/O refers to storage (SSD, HDD, etc.) in systems community)

FlashAttention

- Key Idea: Memory Access Pattern Aware Attention

- Specifically, perform computations similar to blocked matrix multiplications
  - Do not need to store the entire temporary $O(L \times L)$ matrix
  - Generate smaller blocks, use them for multiplication with values, discard them

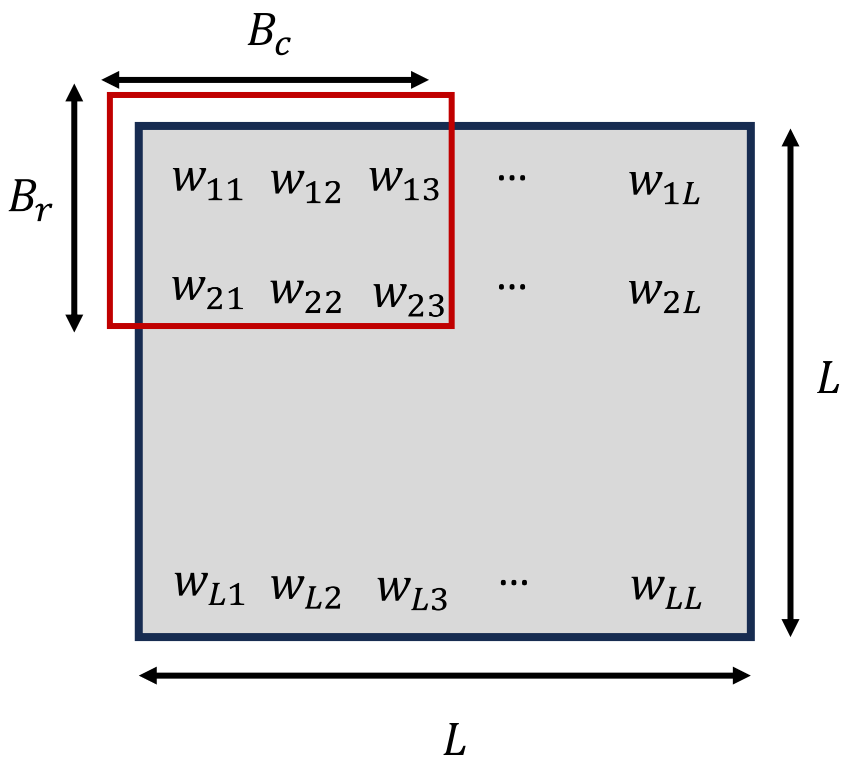
CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

FlashAttention

$$\underbrace{\begin{bmatrix} w_{11} & w_{12} & w_{13} & \cdots & w_{1L} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2L} \\ & & & & \\ w_{L1} & w_{L2} & w_{L3} & \cdots & w_{LL} \end{bmatrix}}_{L} \; L$$

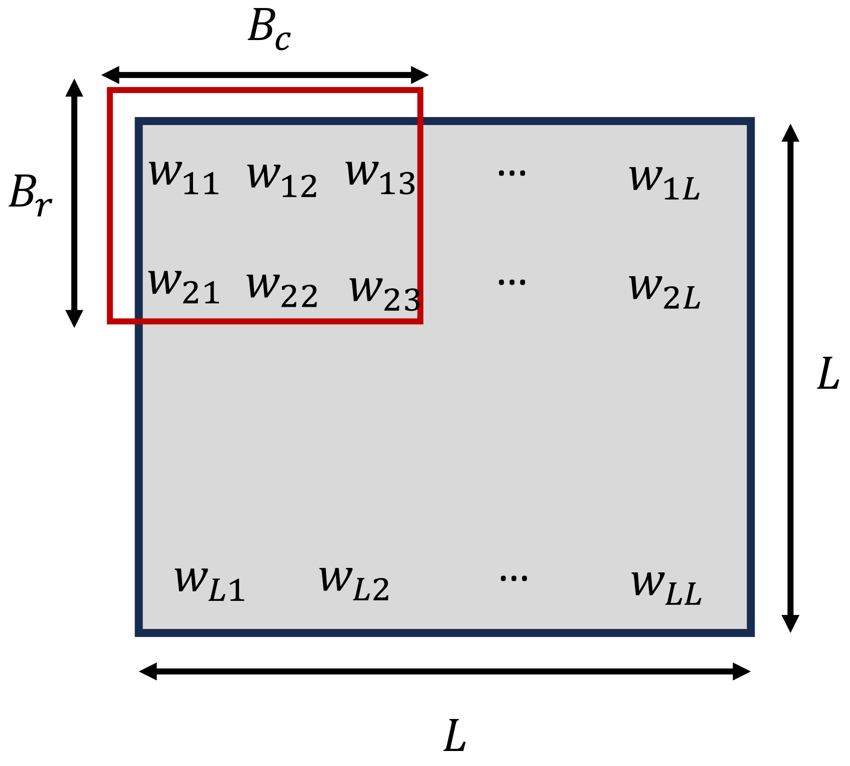
Memory Needed: L^2

FlashAttention



Memory Needed: ? ?

FlashAttention

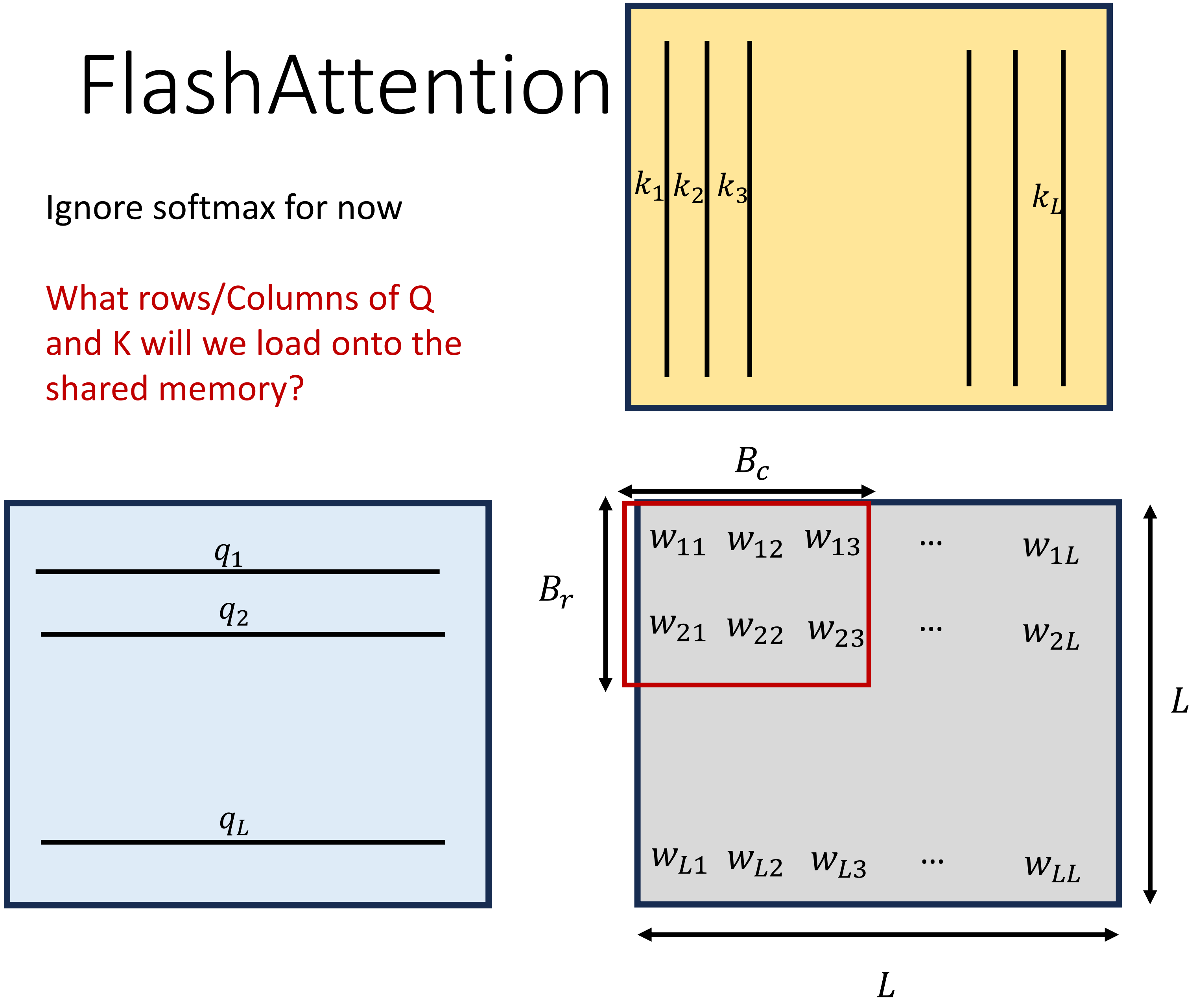


Memory Needed: $B_r \times B_c$

FlashAttention

Ignore softmax for now

What rows/Columns of Q and K will we load onto the shared memory?

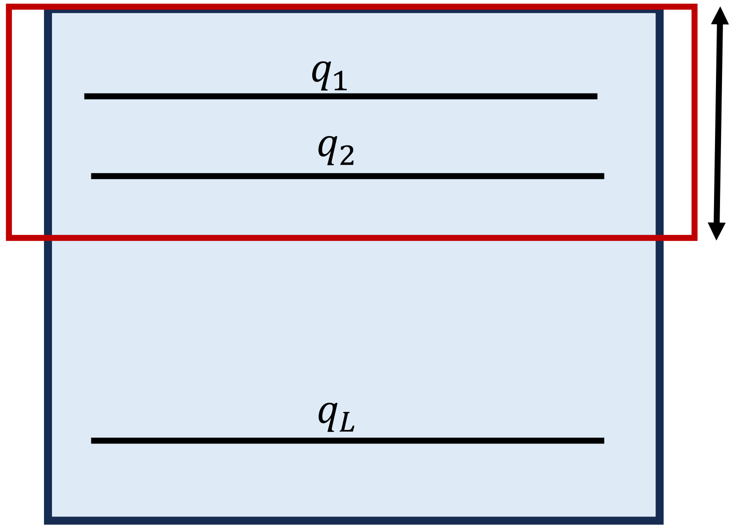


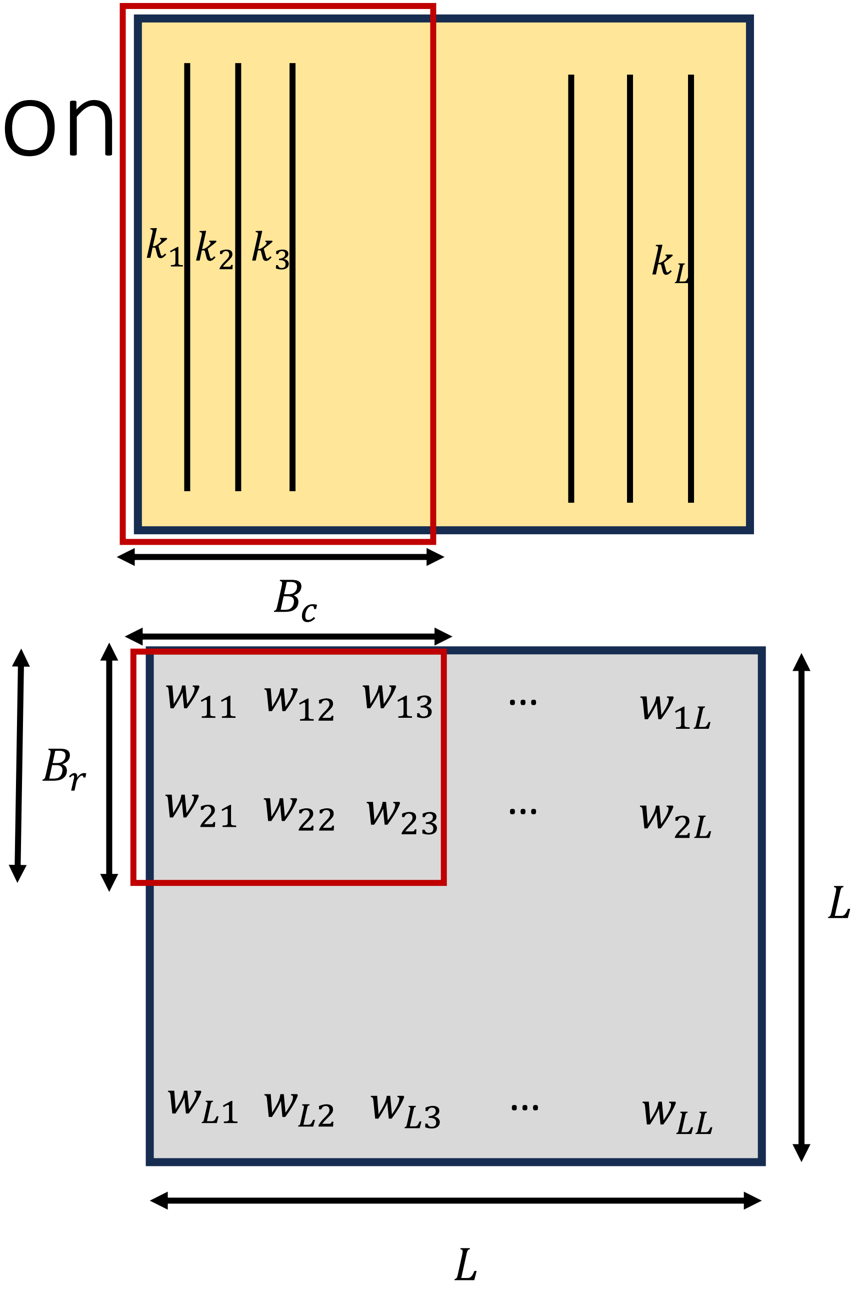
Memory Needed: $B_r \times B_c$

FlashAttention

Ignore softmax for now

What rows/Columns of Q and K will we load onto the shared memory?



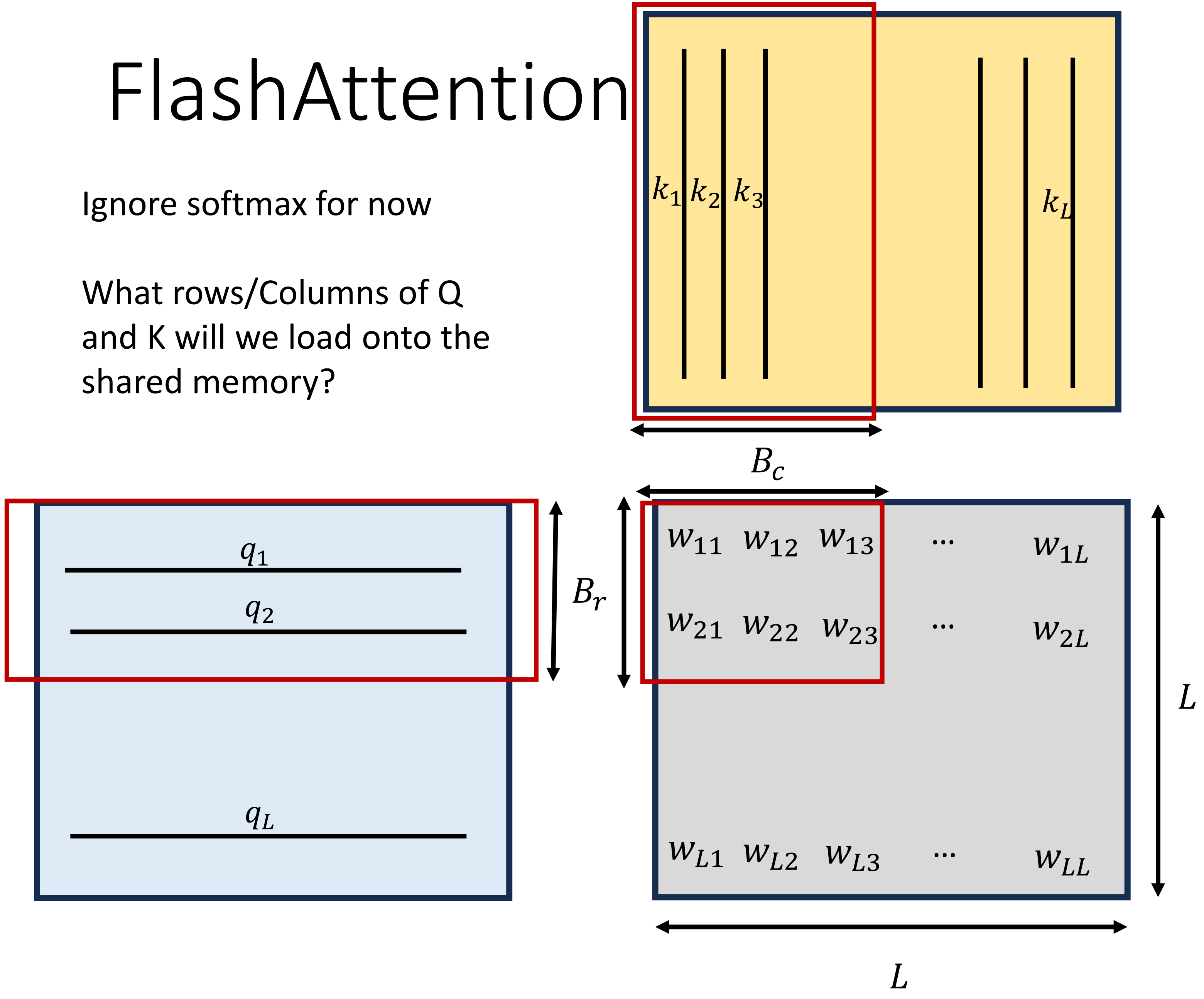


Memory Needed: $B_r \times B_c + ??$

FlashAttention

Ignore softmax for now

What rows/Columns of Q and K will we load onto the shared memory?



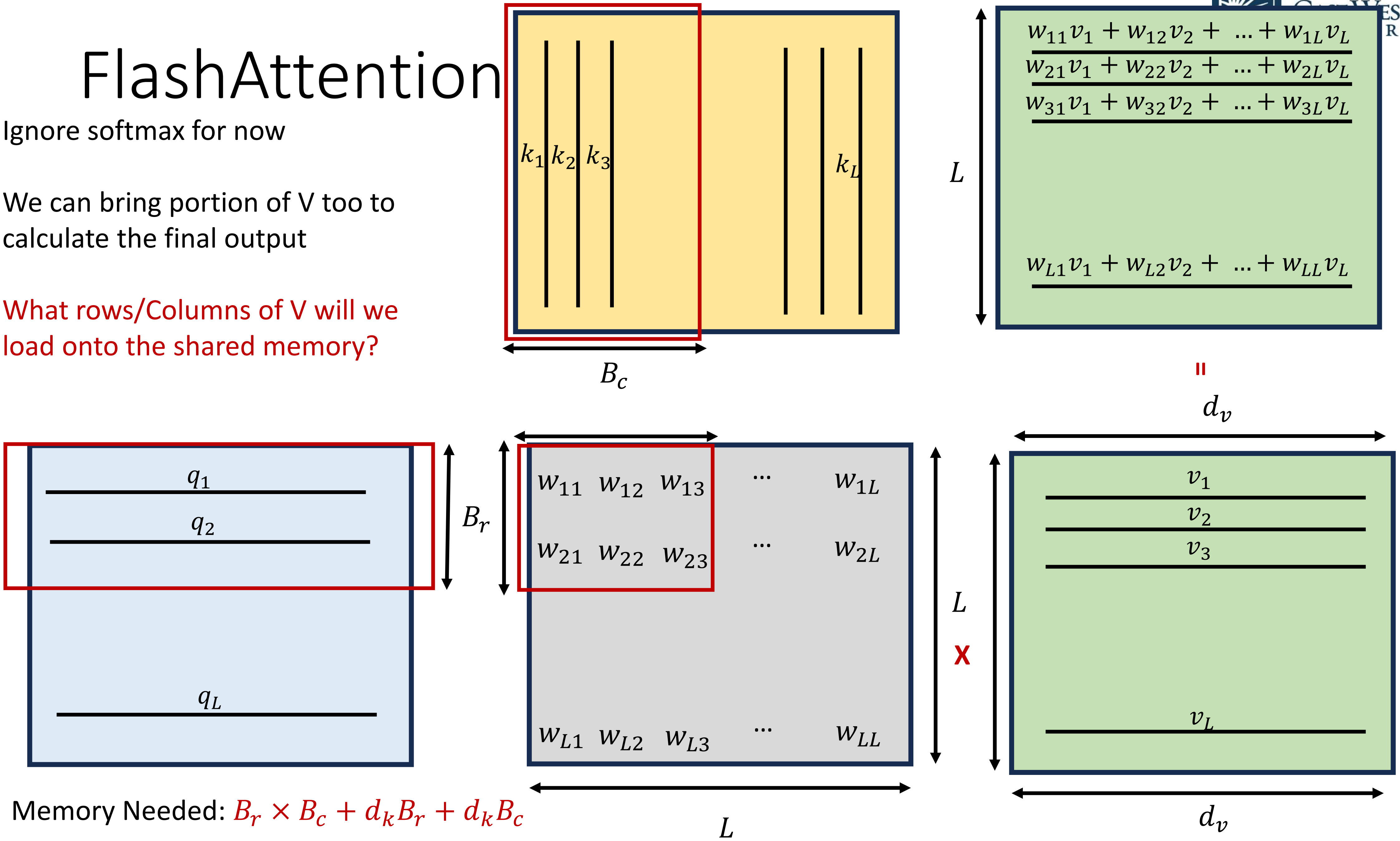
Memory Needed: $B_r \times B_c + d_k B_r + d_k B_c$

FlashAttention

Ignore softmax for now

We can bring portion of V too to calculate the final output

What rows/Columns of V will we load onto the shared memory?



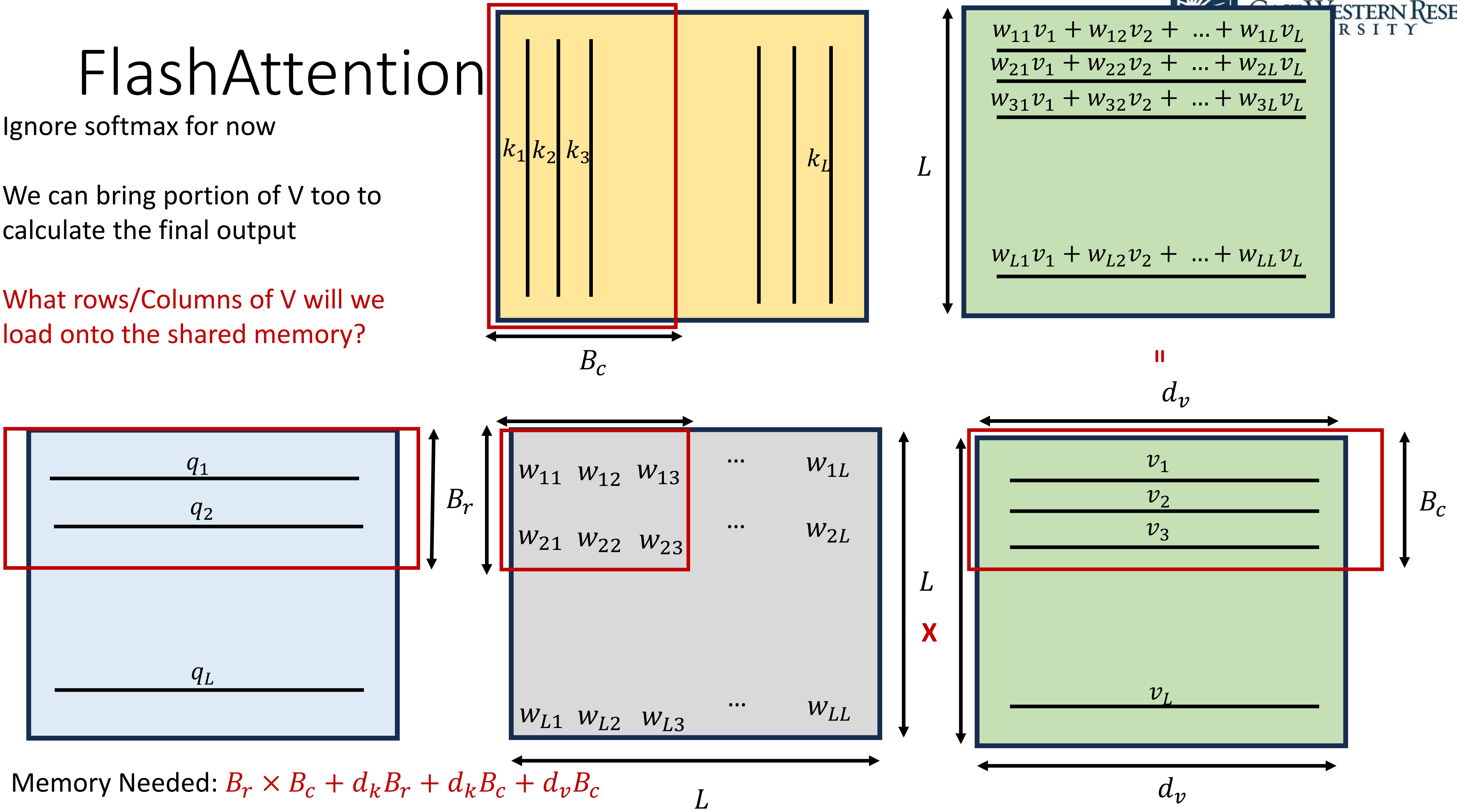
Memory Needed: $B_r \times B_c + d_k B_r + d_k B_c$

FlashAttention

Ignore softmax for now

We can bring portion of V too to calculate the final output

What rows/Columns of V will we load onto the shared memory?

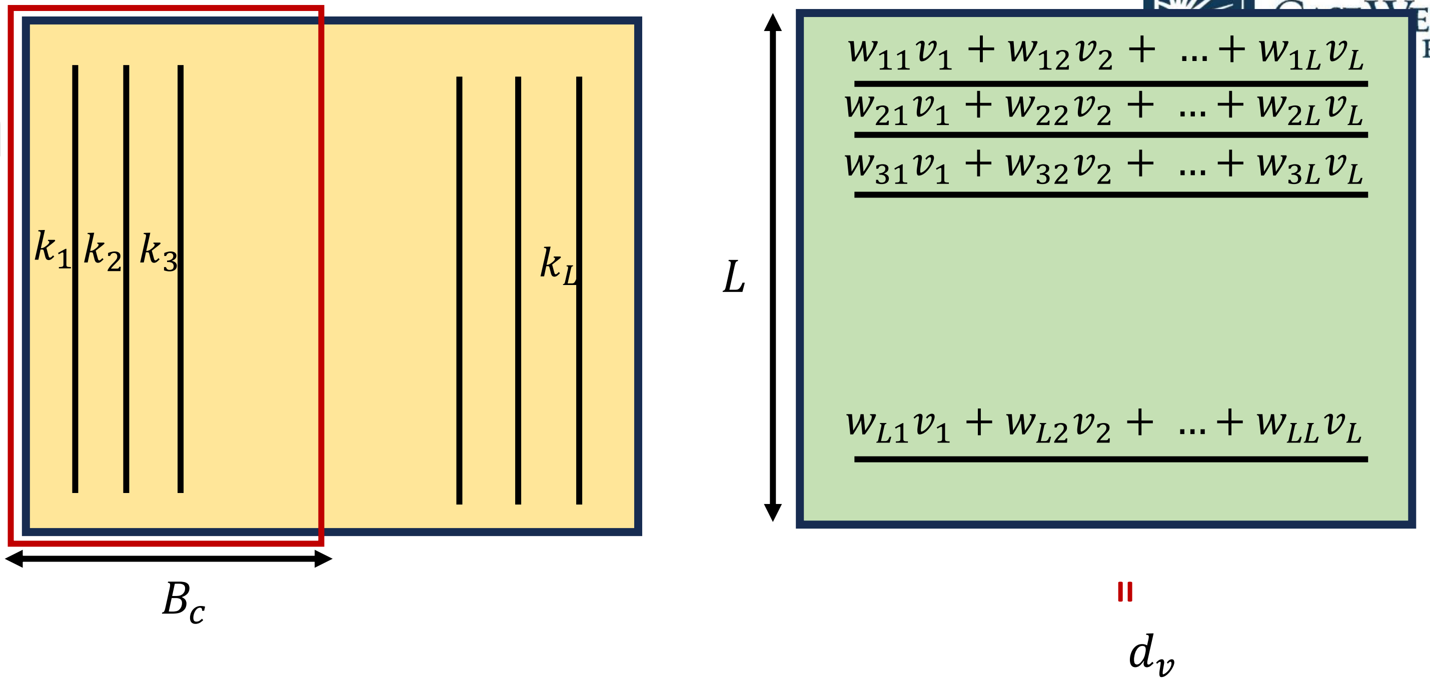


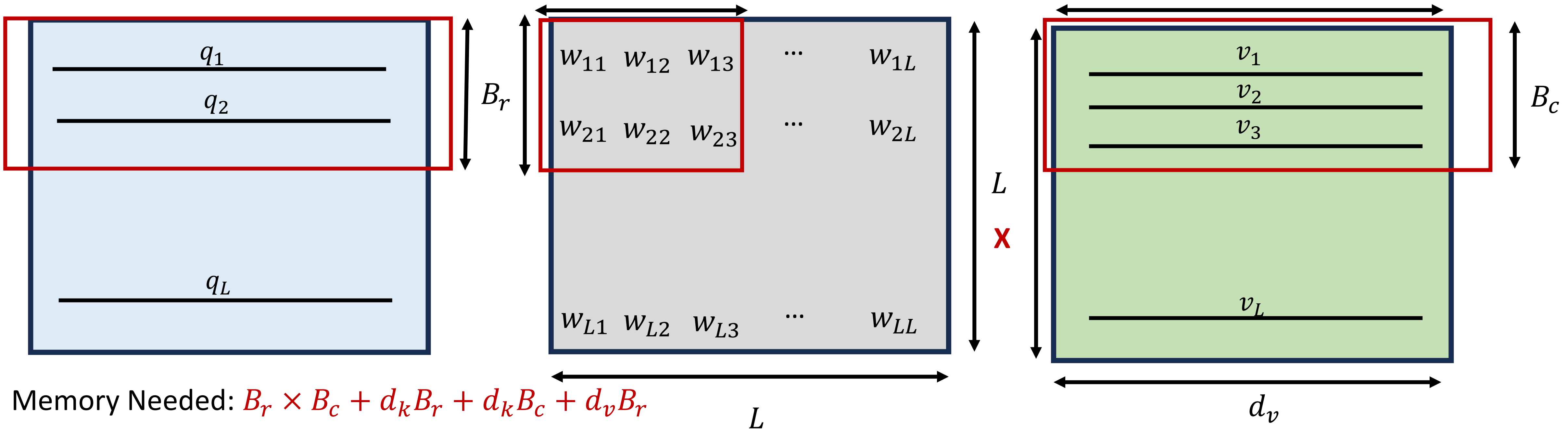
Memory Needed: $B_r \times B_c + d_k B_r + d_k B_c + d_v B_c$

FlashAttention

Ignore softmax for now

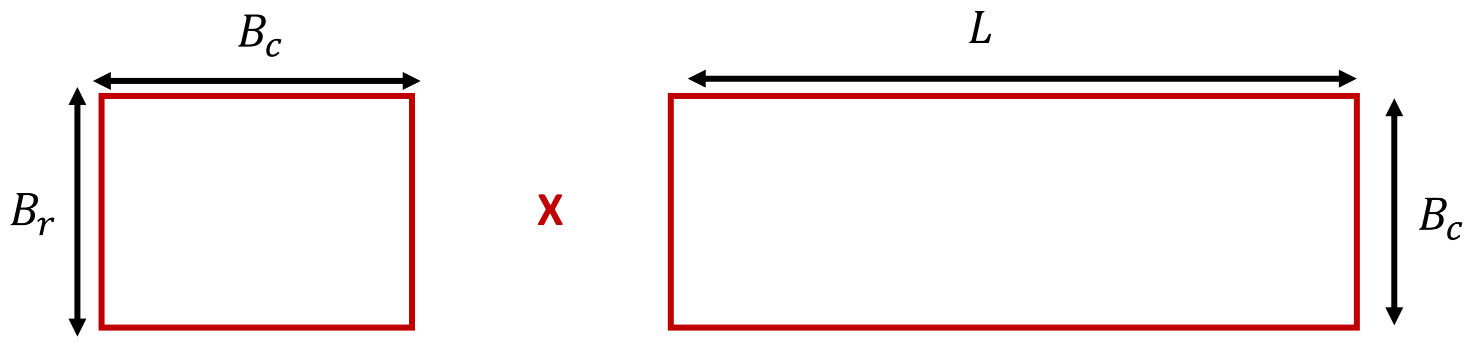
What rows/Columns of O will this produce (that will again be temporarily stored in the shared memory)?





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

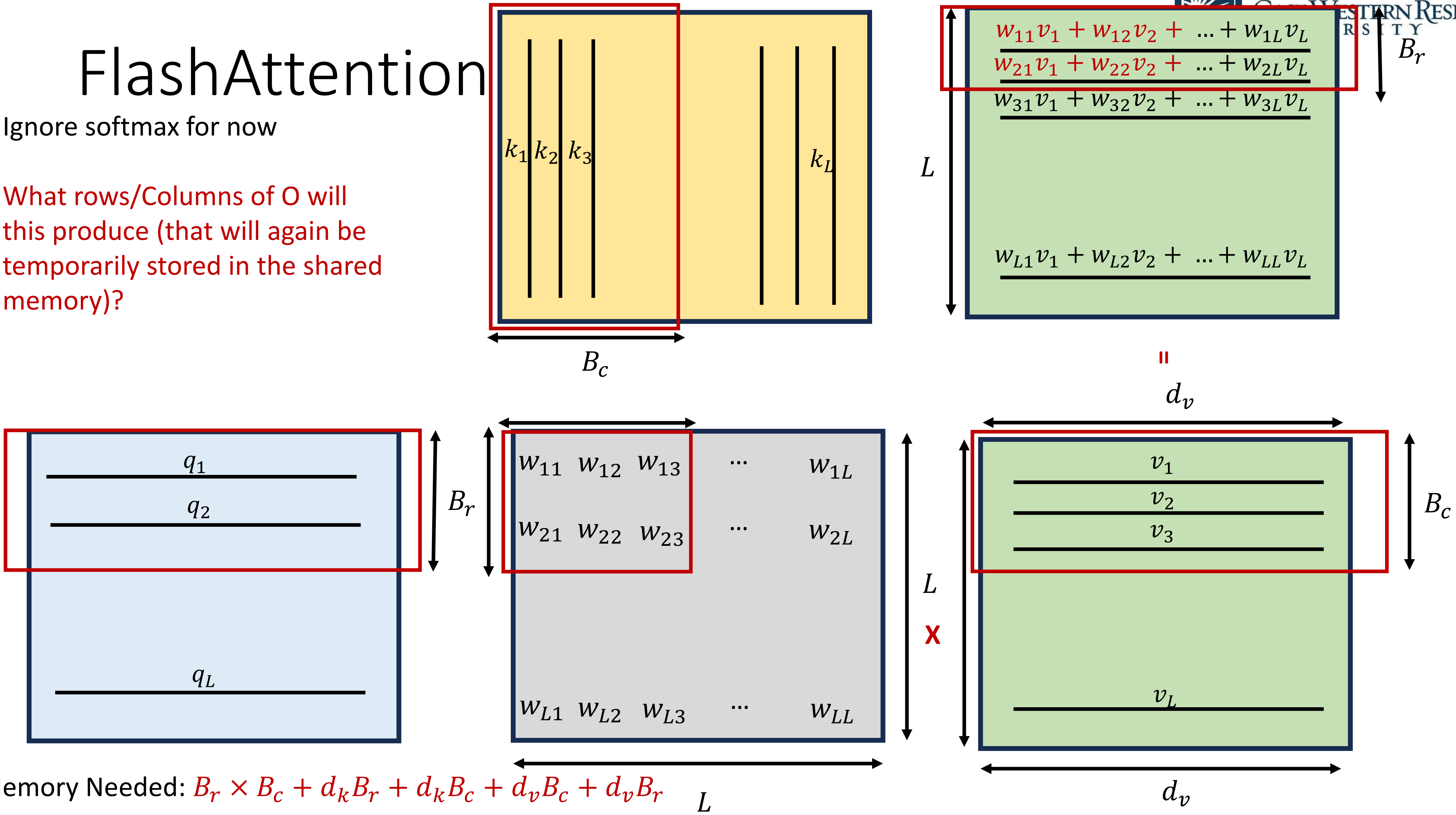
FlashAttention



FlashAttention

Ignore softmax for now

What rows/Columns of O will this produce (that will again be temporarily stored in the shared memory)?

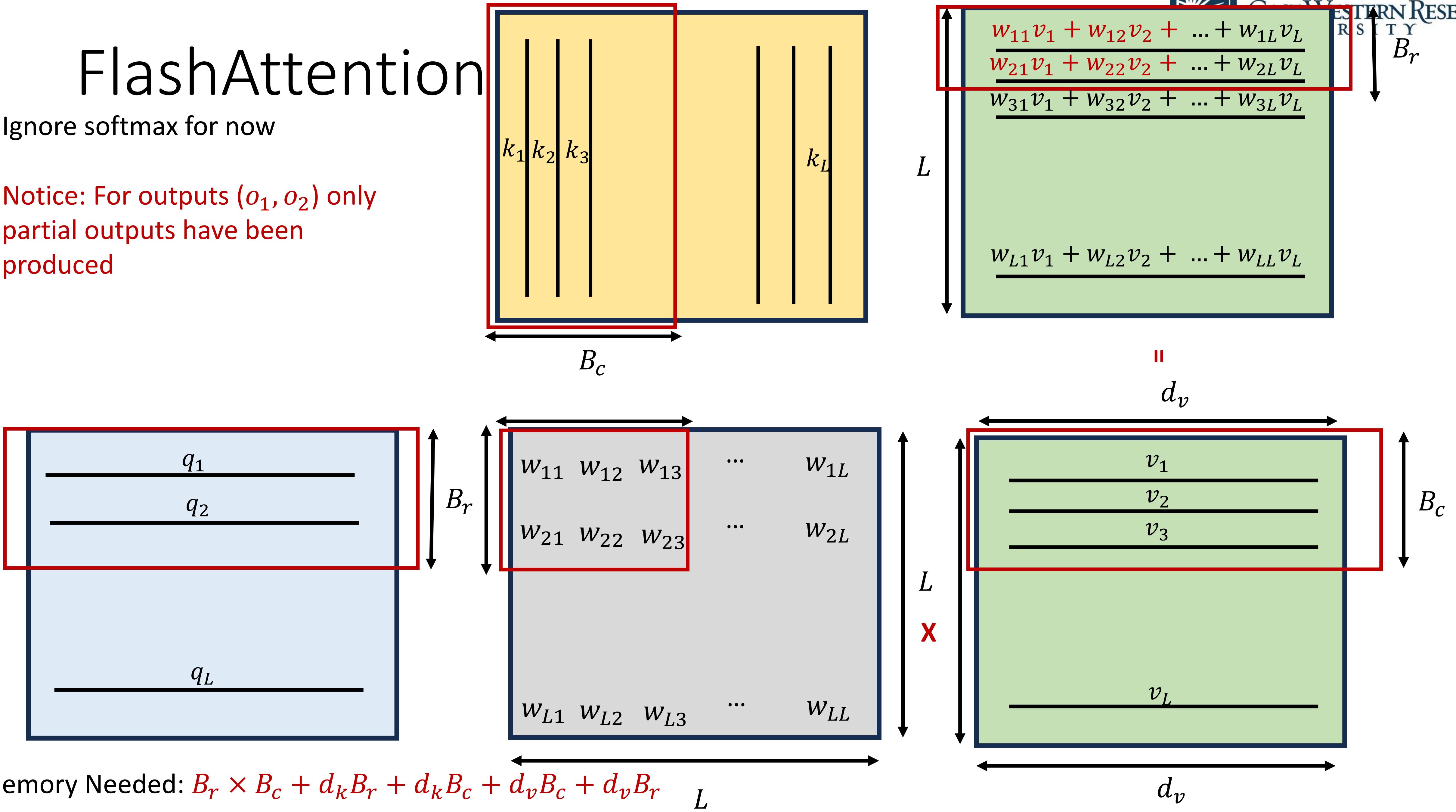


Memory Needed: $B_r \times B_c + d_k B_r + d_k B_c + d_v B_c + d_v B_r$

FlashAttention

Ignore softmax for now

Notice: For outputs (o_1, o_2) only partial outputs have been produced

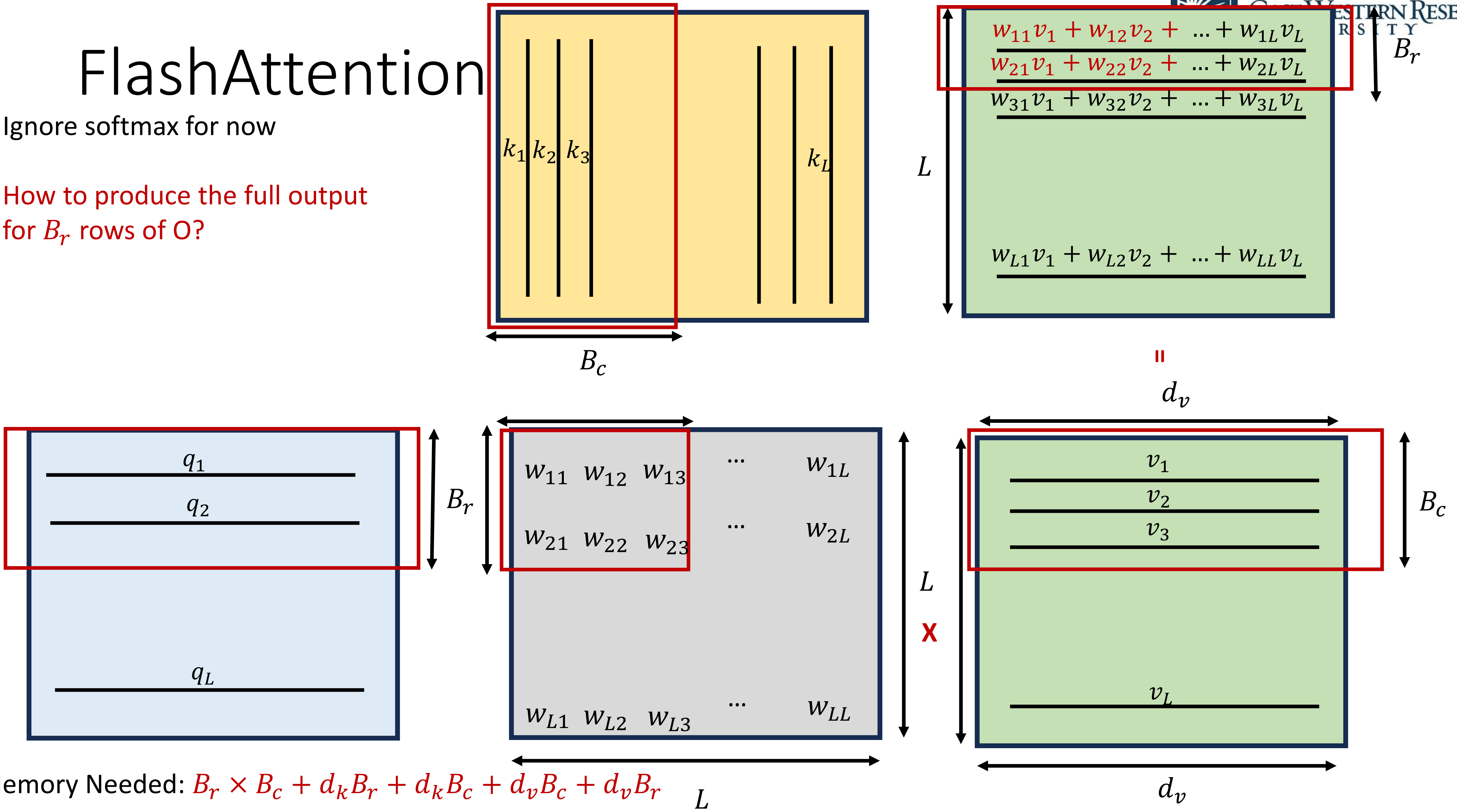


Memory Needed: $B_r \times B_c + d_k B_r + d_k B_c + d_v B_c + d_v B_r$

FlashAttention

Ignore softmax for now

How to produce the full output for B_r rows of O?

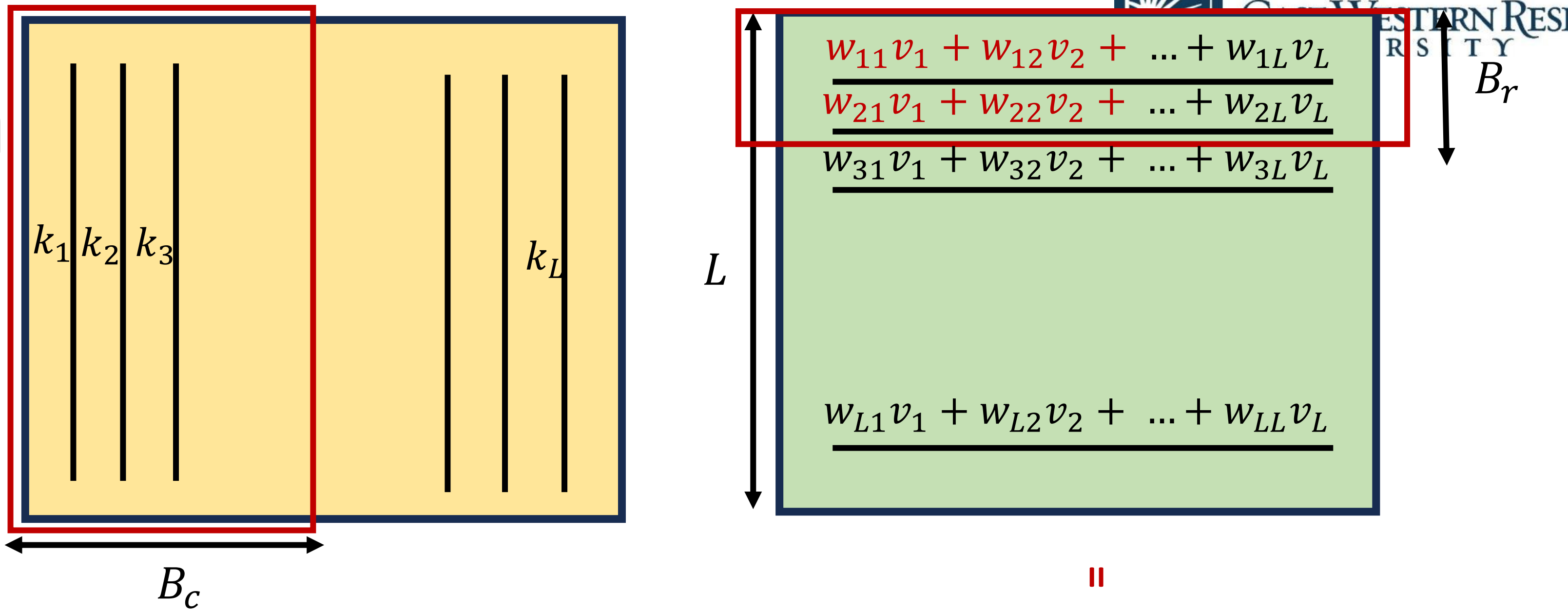


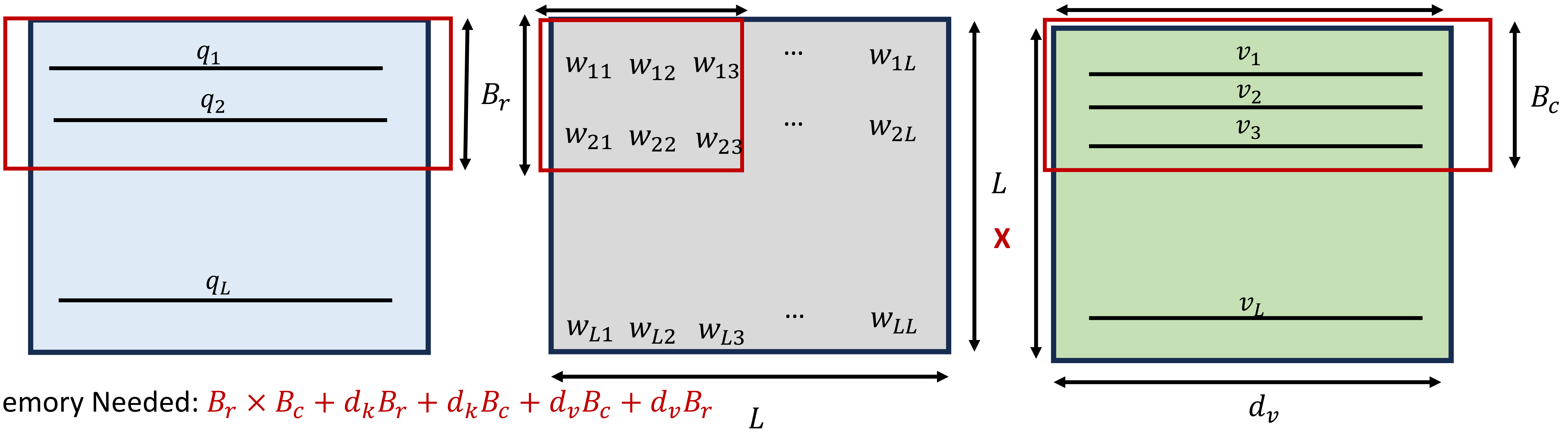
Memory Needed: $B_r \times B_c + d_k B_r + d_k B_c + d_v B_c + d_v B_r$

FlashAttention

Ignore softmax for now

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



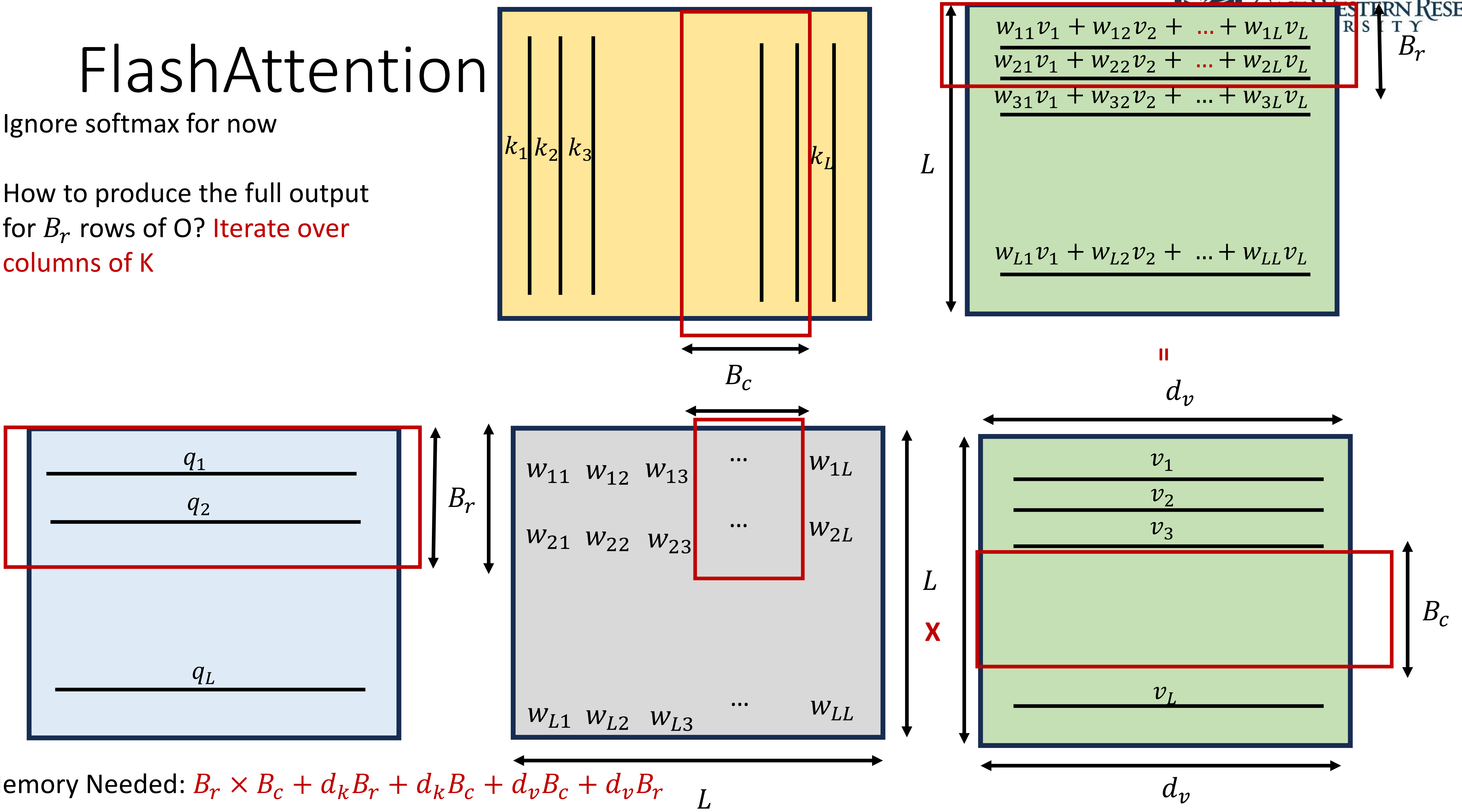


Memory Needed: $B_r \times B_c + d_k B_r + d_k B_c + d_v B_c + d_v B_r$

FlashAttention

Ignore softmax for now

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



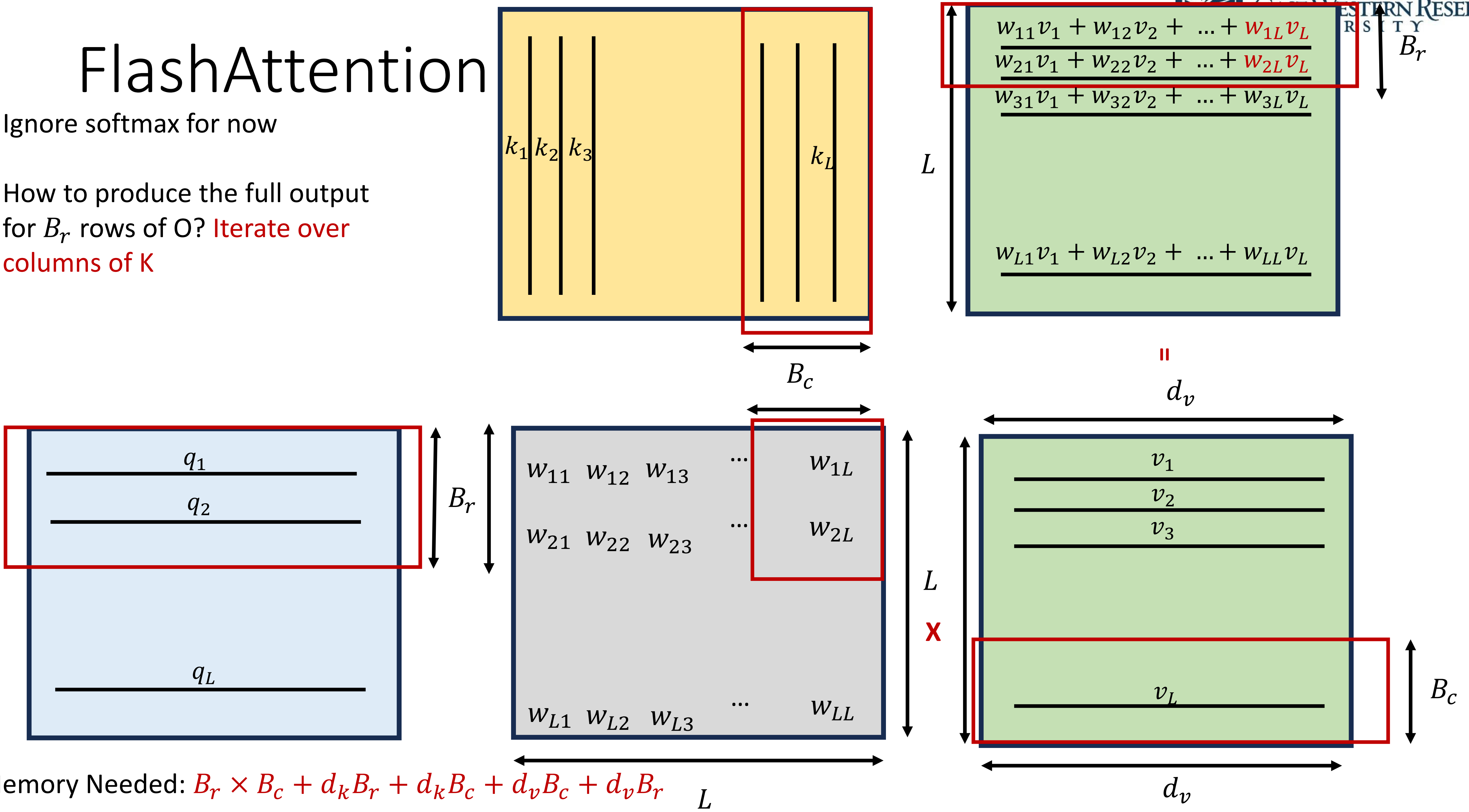
Memory Needed: $B_r \times B_c + d_k B_r + d_k B_c + d_v B_c + d_v B_r$

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

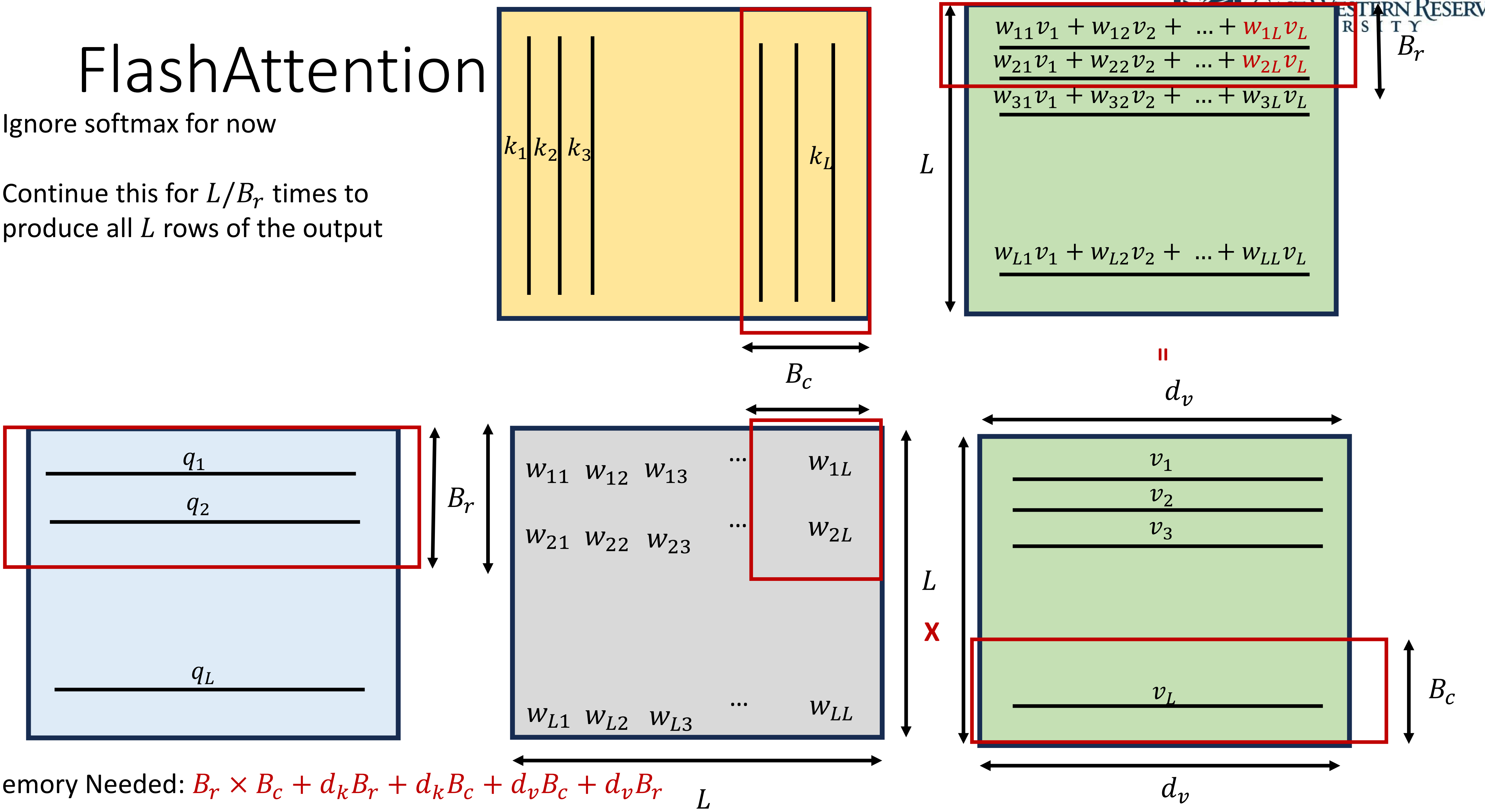


Memory Needed: $B_r \times B_c + d_k B_r + d_k B_c + d_v B_c + d_v B_r$

FlashAttention

Ignore softmax for now

Continue this for L/B_r times to produce all L rows of the output



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

FlashAttention - PseudoCode

for $1 \leq j \leq T_c$ **do**

  Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.

  for $1 \leq i \leq T_r$ **do**

    Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.

    On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.

    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.

  end for

end for

For each column block of K, V: $T_c = L/B_c$

Iterate through row blocks of Q to produce O: $T_r = L/B_r$

Additional bookkeeping to compute softmax that we ignored till now.

FlashAttention

- How do we determine B_r, B_c?
- $B_r \times B_c + d_k B_r + d_k B_c + d_v B_c + d_v B_r \leq S$
- Typically sizes of {64,128} used

How do we compute “partial” softmaxes

- Concept similar to running averages

$$[w_{i1}, w_{i2}, w_{i3}, w_{i4}, w_{i5}, w_{i6}] = \ softmax([q_i.k_1, q_i.k_2, q_i.k_3, q_i.k_4, q_i.k_5, q_i.k_6])$$

- Assuming $B_c = 2$

How do we compute “partial” softmaxes

- Concept similar to running averages

$$[w_{i1}, w_{i2}, w_{i3}, w_{i4}, w_{i5}, w_{i6}] = \left[e^{q_i.k_1}, e^{q_i.k_2}, e^{q_i.k_3}, e^{q_i.k_4}, e^{q_i.k_5}, e^{q_i.k_6}\right] / \sum_j e^{q_i.k_j}$$

- Assuming $B_c = 2$

How do we compute "partial" softmaxes

- Concept similar to running averages

$$[w_{i1}, w_{i2}, w_{i3}] = \left[e^{q_i.k_1}, e^{q_i.k_2}, e^{q_i.k_3}\right] / \boxed{\sum_{j=\{1,2,3\}} e^{q_i.k_j}} \rightarrow l$$

- Assuming $B_c = 2$

How do we compute “partial” softmaxes

- Concept similar to running averages

$$[w_{i4}, w_{i5}, w_{i6}] = \left[e^{q_i.k_4}, e^{q_i.k_5}, e^{q_i.k_6}\right] / \left(\sum_{j=\{4,5,6\}} e^{q_i.k_j} + l\right)$$

- Assuming $B_c = 2$

How do we compute "partial" softmaxes

- Concept similar to running averages

$$[w_{i4}, w_{i5}, w_{i6}] = \left[e^{q_i.k_4}, e^{q_i.k_5}, e^{q_i.k_6}\right] / \left(\sum_{j=\{4,5,6\}} e^{q_i.k_j} + l\right)$$

- Assuming $B_c = 2$

- We need to recompute $[w_{i1}, w_{i2}, w_{i3}]$ Why???

How do we compute "partial" softmaxes

- Concept similar to running averages

$$[w_{i4}, w_{i5}, w_{i6}] = [e^{q_i.k_4}, e^{q_i.k_5}, e^{q_i.k_6}]/\boxed{\left(\sum_{j=\{4,5,6\}} e^{q_i.k_j} + l\right)} \rightarrow l_2$$

- Assuming $B_c = 2$

- We need to recompute $[w_{i1}, w_{i2}, w_{i3}]$ Why??? The sum in denominator has changed.

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.

Flashattention – Scheduling on GPUs

- Each block $B_r \times B_c$ is mapped to a SMP
- Flashattention 2 optimizations
  - Improve partitioning of blocks into warps
  - Improve online softmax/output calculation
- Flashattention 3 optimizations
  - Nvidia Hopper specific optimizations – asynchronous scheduling of online softmax/output calculation and matrix multiplications

Next Class

- 10/28 Lecture 17
  - Accelerating Transformer Model: Sparsity

Thank You

CASE SCHOOL OF ENGINEERING
CASE WESTERN RESERVE UNIVERSITY

- Questions?

- Email: sanmukh.kuppannagari@case.edu