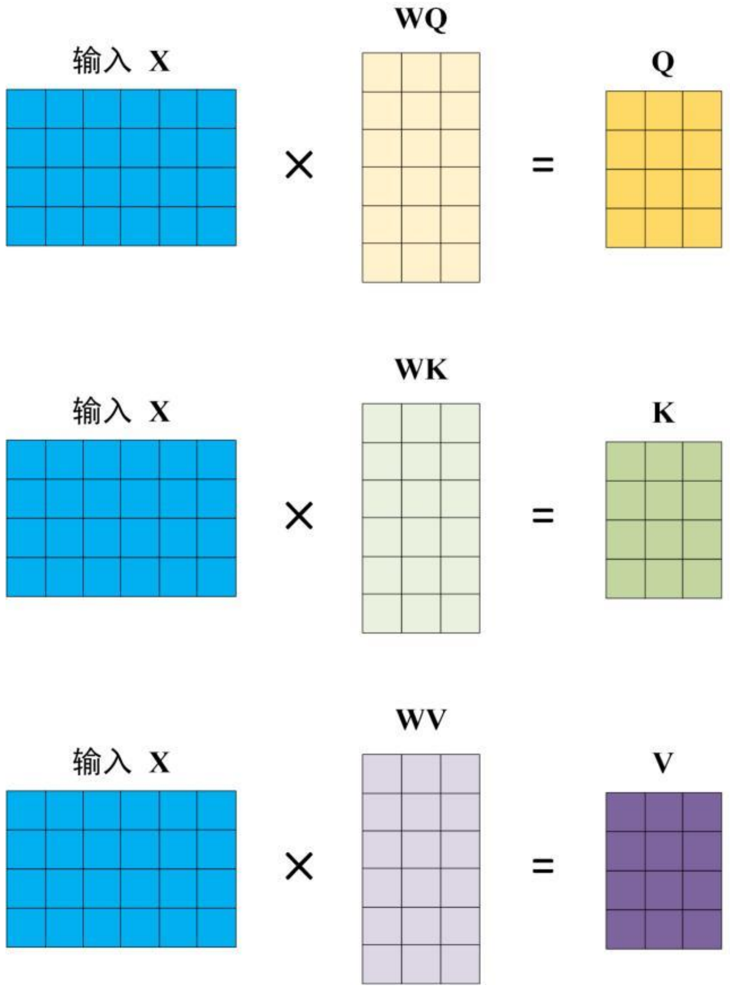
FlashAttention

Kun Yuan (袁 坤)

Center for Machine Learning Research @ Peking University

Self-Attention



$$Q = XW_Q \in \mathbb{R}^{N\times d}$$

N is the sequence length

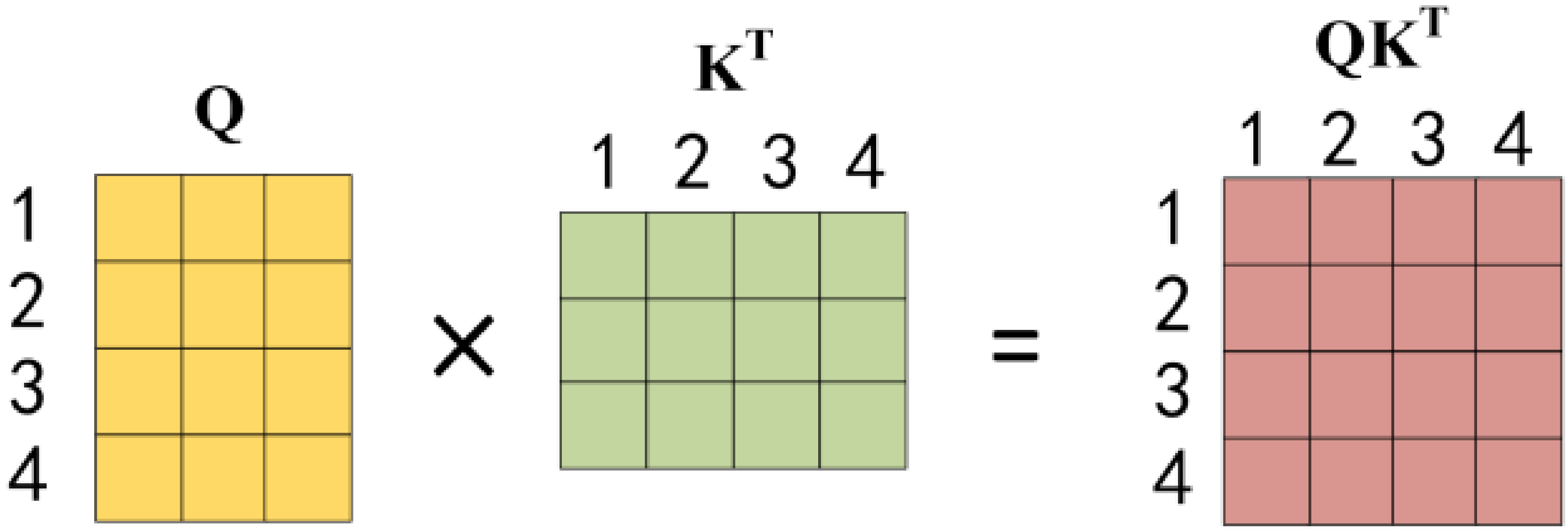
d is the embedding dimension

$$K = XW_K \in \mathbb{R}^{N\times d}$$

$$V = XW_V \in \mathbb{R}^{N\times d}$$

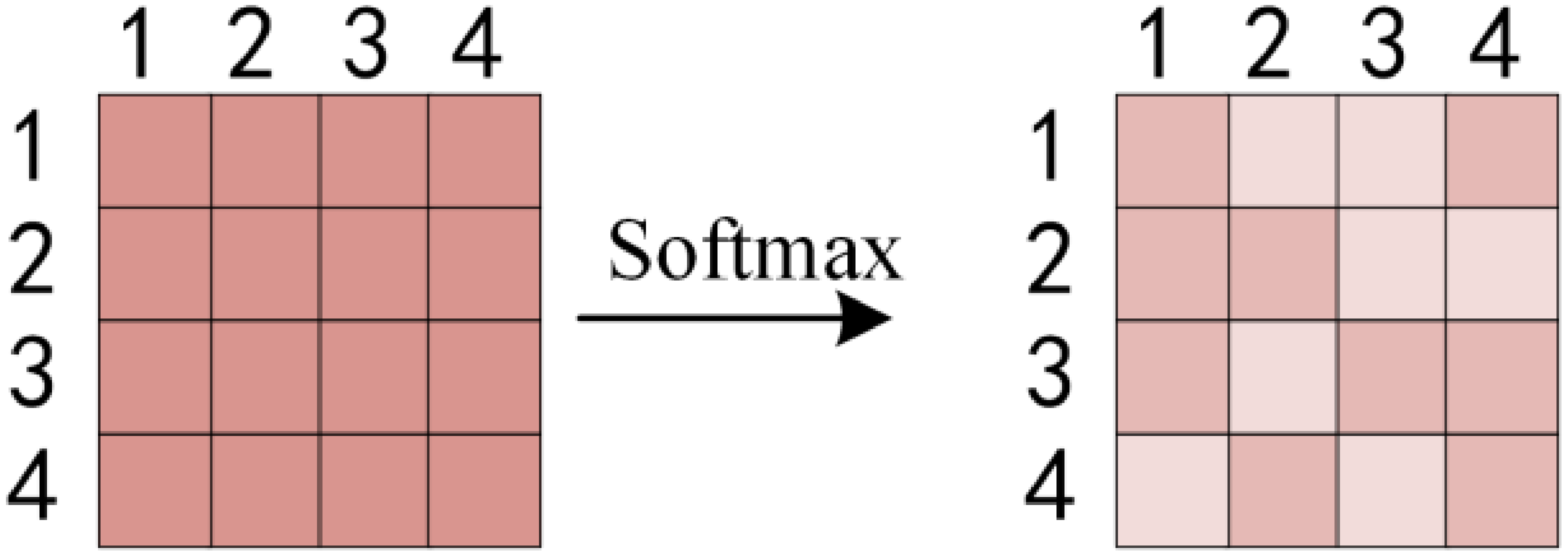
Self-Attention

$$S = QK^\top \in \mathbb{R}^{N \times N}$$



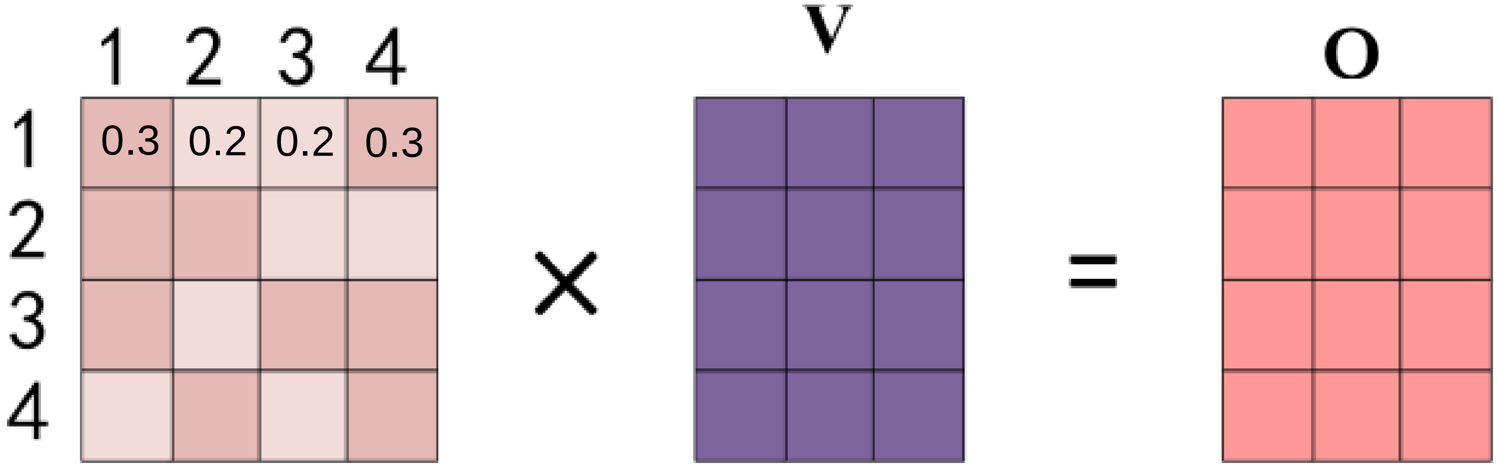
$$P = \mathrm{softmax}(S) \in \mathbb{R}^{N \times N}$$

(we ignore the scaling for simplicity)



Self-Attention

$$O = PV \in \mathbb{R}^{N \times d}$$



FLOPS

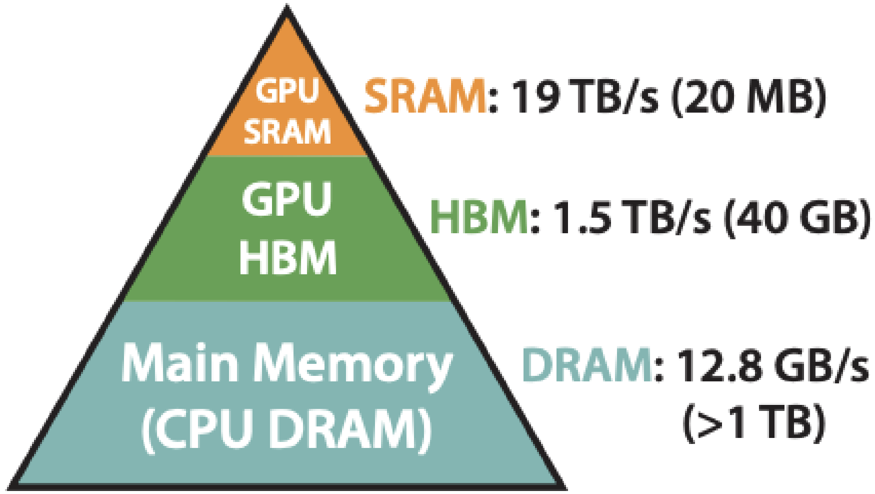
- The above attention process incurs $O(N^2d)$ FLOPS computation complexity

- Increases quadratically fast with sequence length N

- Various methods have been developed to reduce $O(N^2)$ to $O(N)$. These methods are not exact attention, and they typically fail to achieve remarkable acceleration

- The fundamental reason is that they cannot reduce **M**emory **A**ccess **C**ost (MAC)

Memory in GPU



Memory Hierarchy with Bandwidth & Memory Size

Fast but small

Large but slow

Execution Model in GPU. Load inputs from HBM to SRAM, computes, then writes outputs to HBM.

Since HBM is slow, MAC is primarily composed of **HBM reads and writes**

[FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness]

MAC in standard attention implementation

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

MAC cost is

$$\boldsymbol{4N^2 + 4dN}$$

Operation	MAC
Load Q and K	$2dN$
Write S	N^2
Read S	N^2
Write P	N^2
Load Q and V	$N^2 + dN$
Write O	dN

MAC consumes significant wall-clock time in transformer

- **Compute-bound operator:** computing time > accessing HBM time

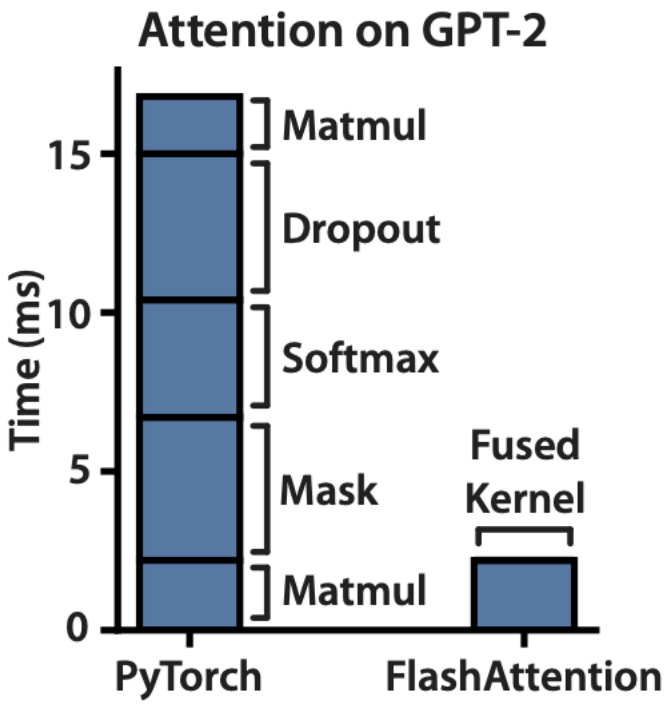
  Matrix multiplication; convolution

- **Memory-bound operator:** accessing HBM time > computing time

  Element-wise operator (activation, dropout); reduction (sum, softmax)

- Transformer includes many memory-bound operators

- Reducing MAC cost can significantly accelerate attention



FLASHATTENTION: Fast and Memory-Efficient Exact Attention with IO-Awareness

Tri Dao[†], Daniel Y. Fu [†], Stefano Ermon [†], Atri Rudra [‡], Christopher Ré [†]

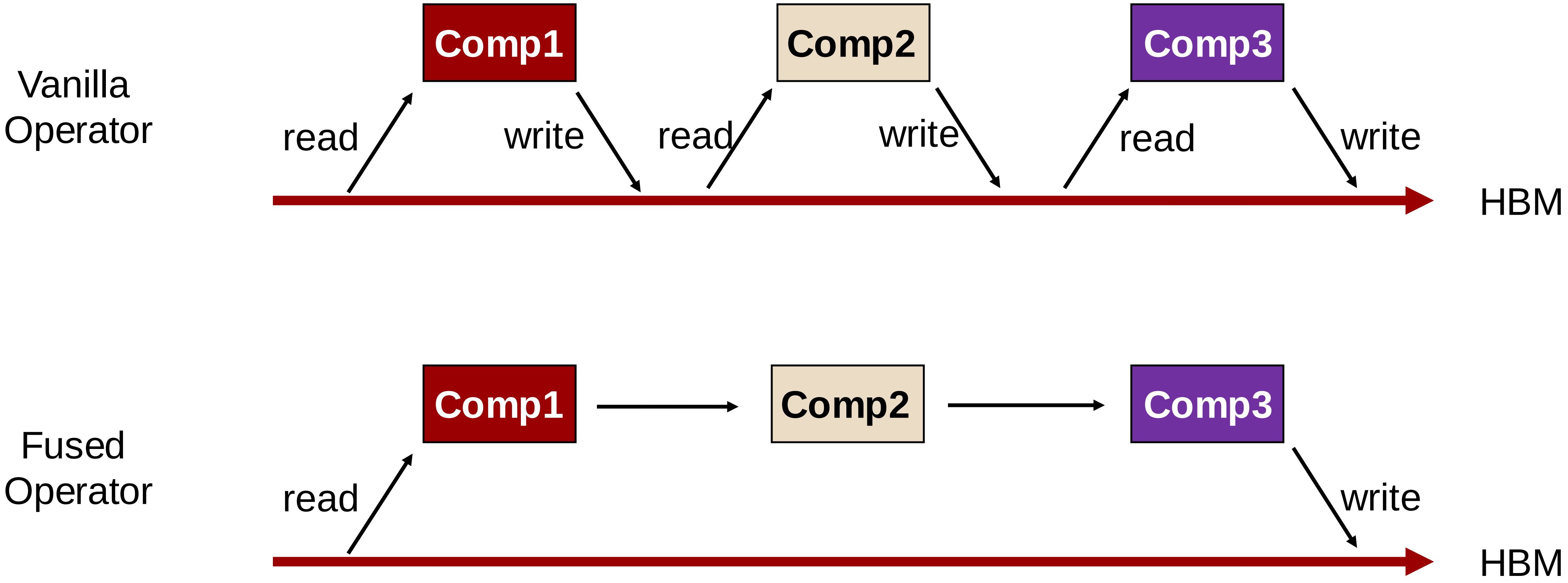
[†] Department of Computer Science, Stanford University

[‡] Department of Computer Science and Engineering, University at Buffalo, SUNY

{trid,danfu}@stanford.edu, ermon@stanford.edu, atri@buffalo.edu, chrismre@cs.stanford.edu

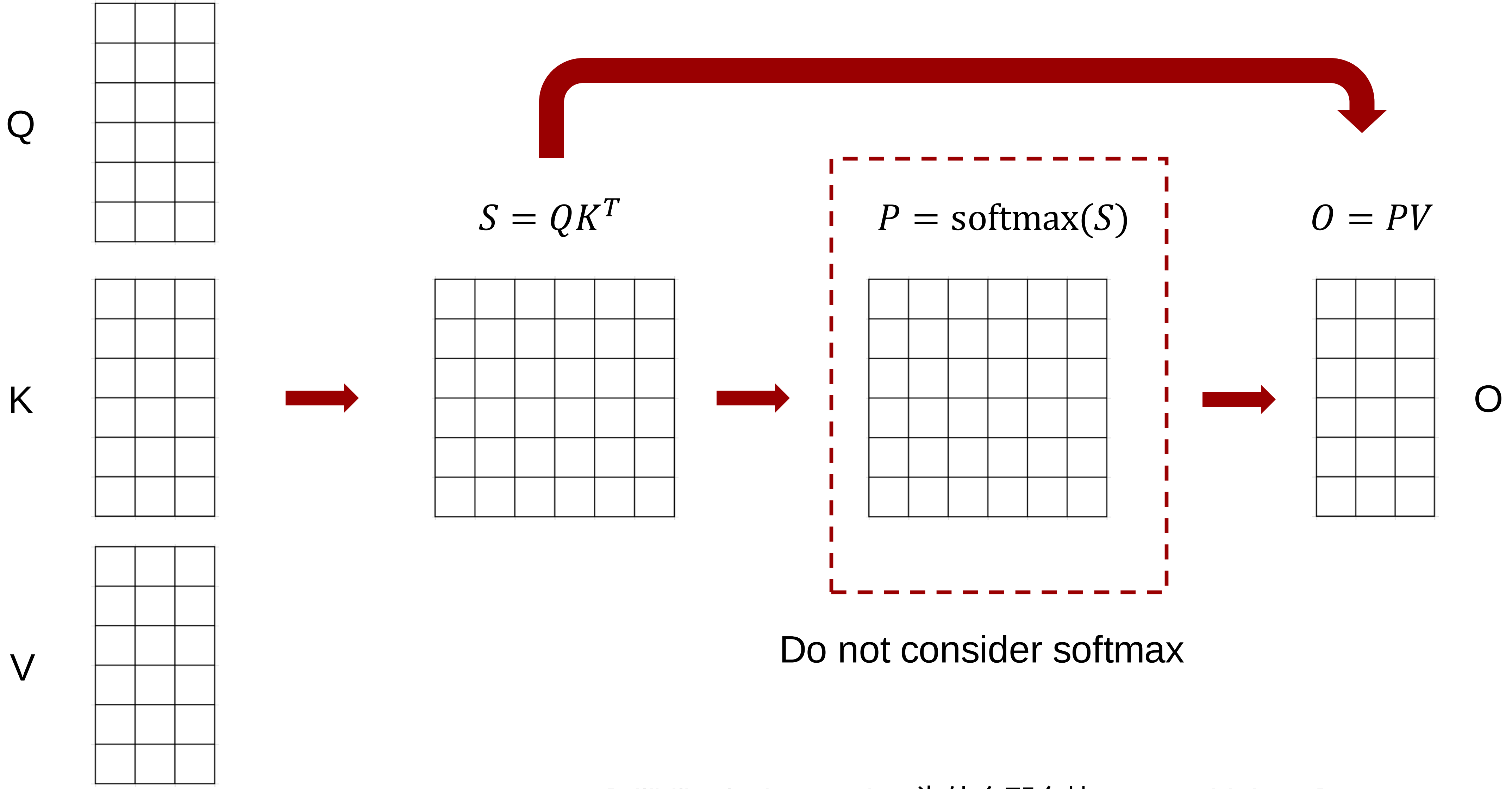
Core idea in FlashAttention: Kernal fusion

北京大學 PEKING UNIVERSITY



Reduce MAC significantly

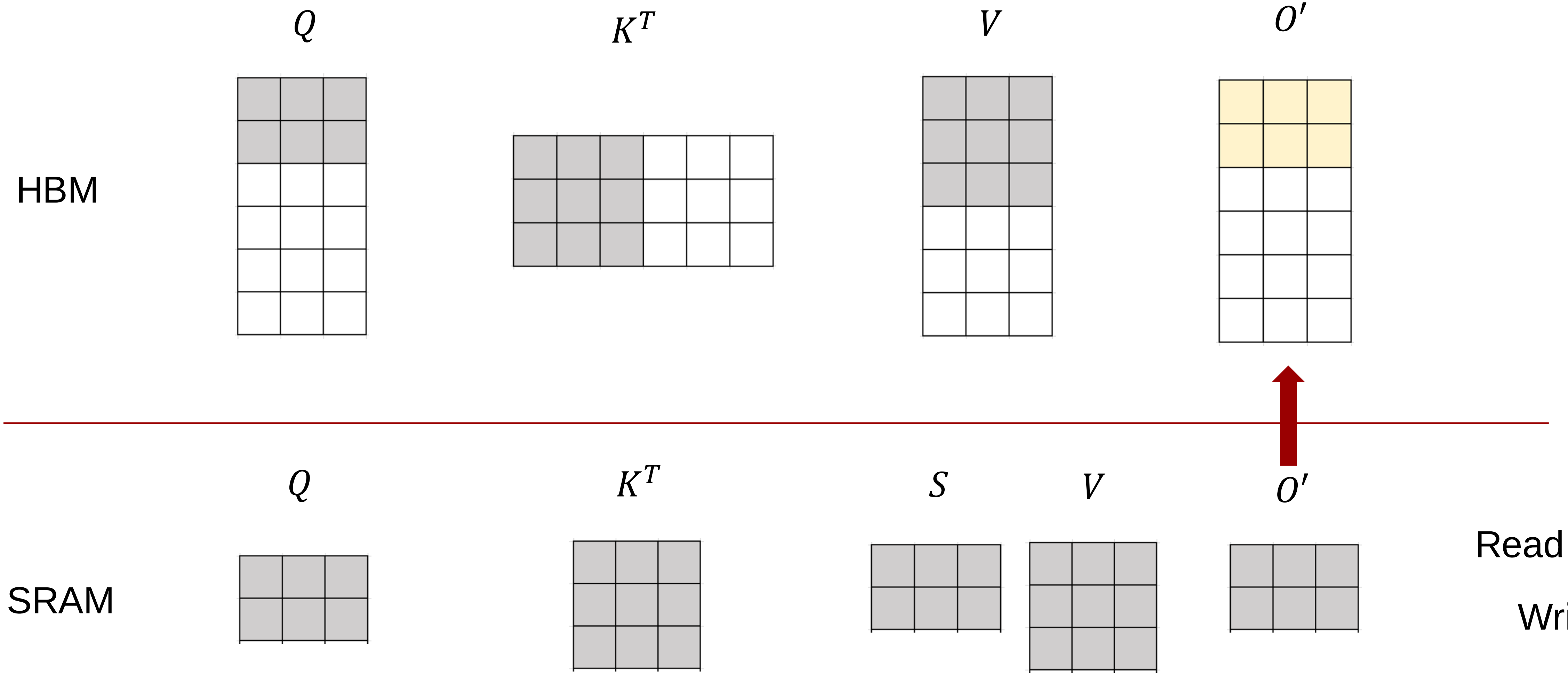
A simplified attention without softmax



Do not consider softmax

[Bilibili: Flash Attention 为什么那么快? By RethinkFun]

Kernal fusion in simplified attention

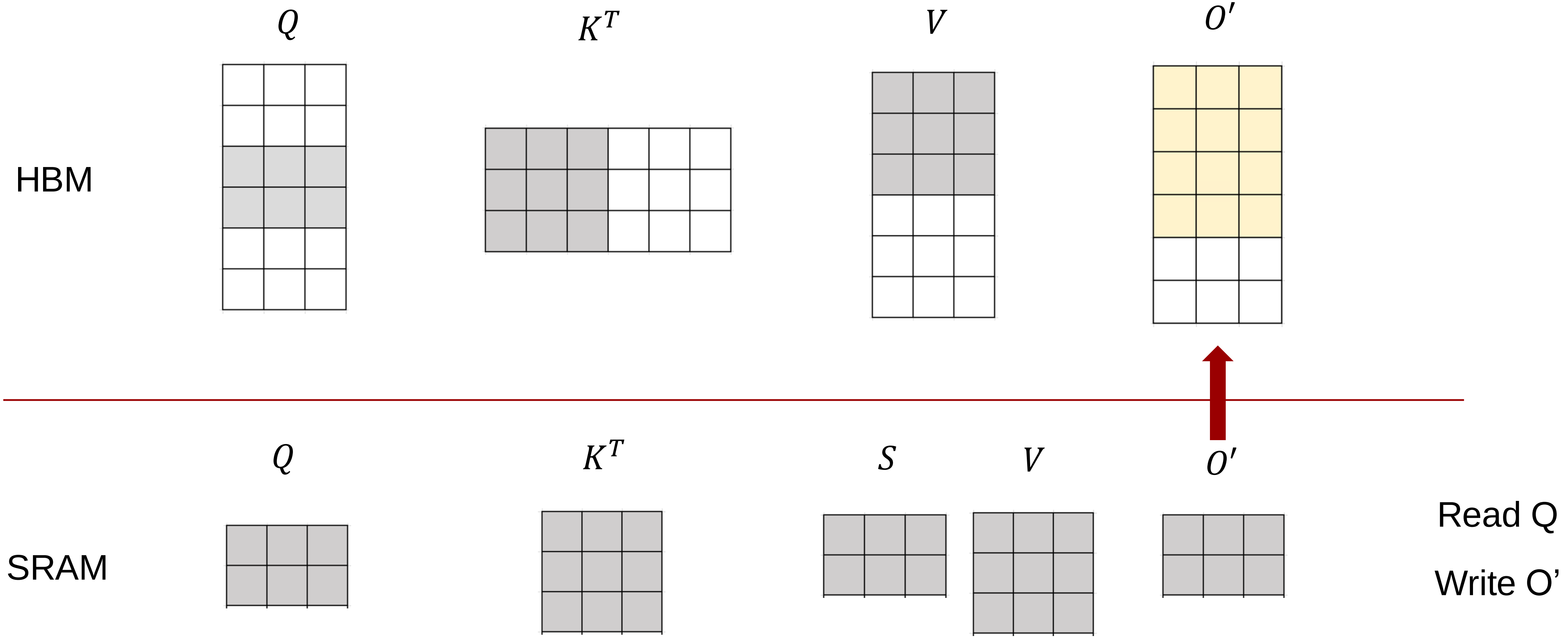


Read Q, K, V

Write O’

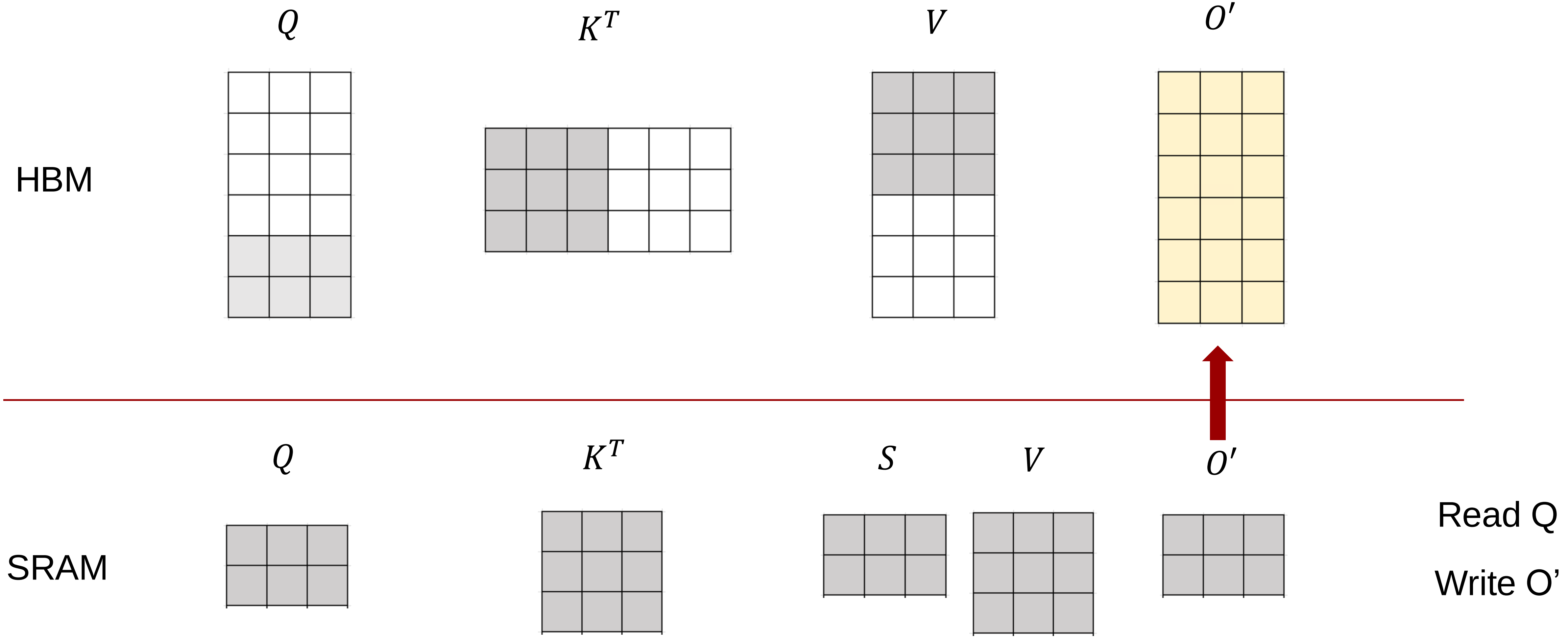
[Bilibili: Flash Attention 为什么那么快? By RethinkFun]

Kernal fusion in simplified attention



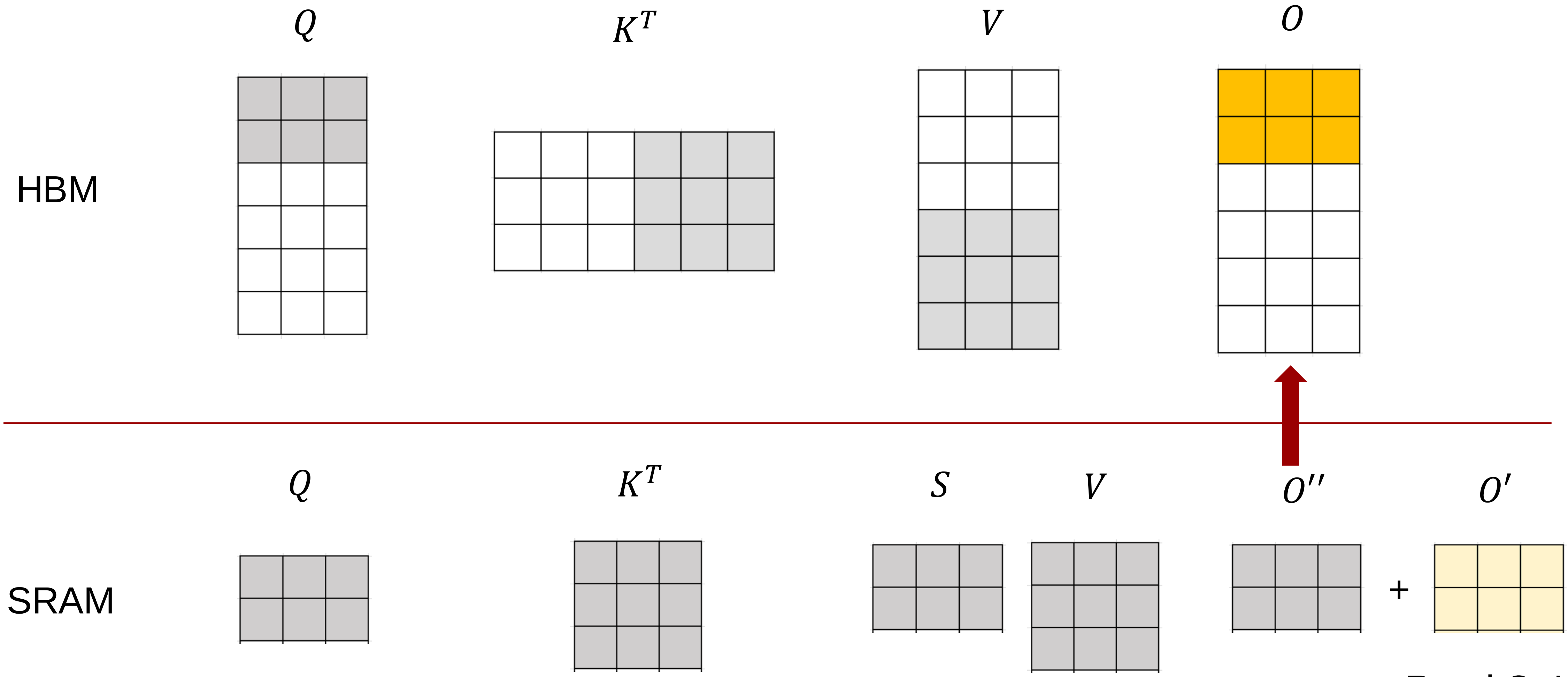
[Bilibili: Flash Attention 为什么那么快? By RethinkFun]

Kernal fusion in simplified attention



[Bilibili: Flash Attention 为什么那么快? By RethinkFun]

Kernal fusion in simplified attention

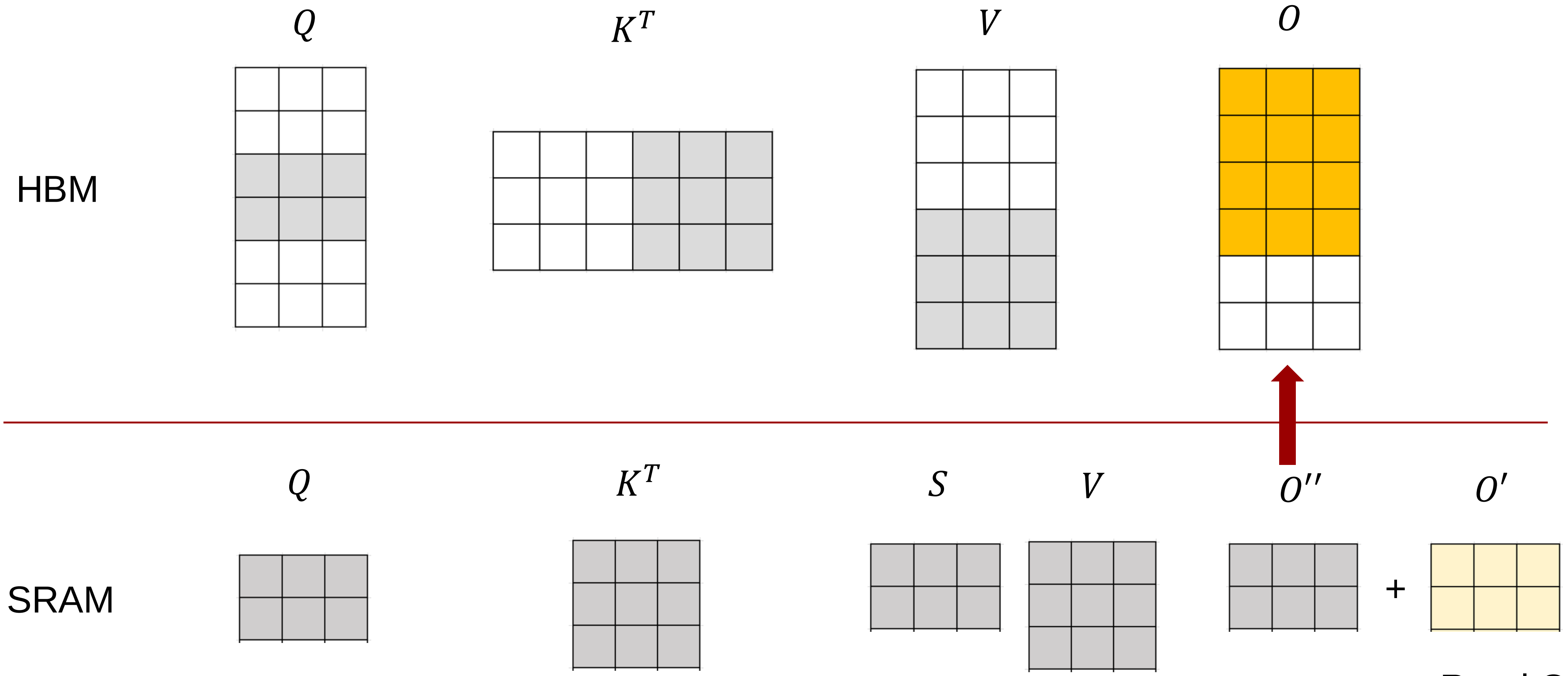


Read Q, K, V,O’

Write O

[Bilibili: Flash Attention 为什么那么快? By RethinkFun]

Kernal fusion in simplified attention

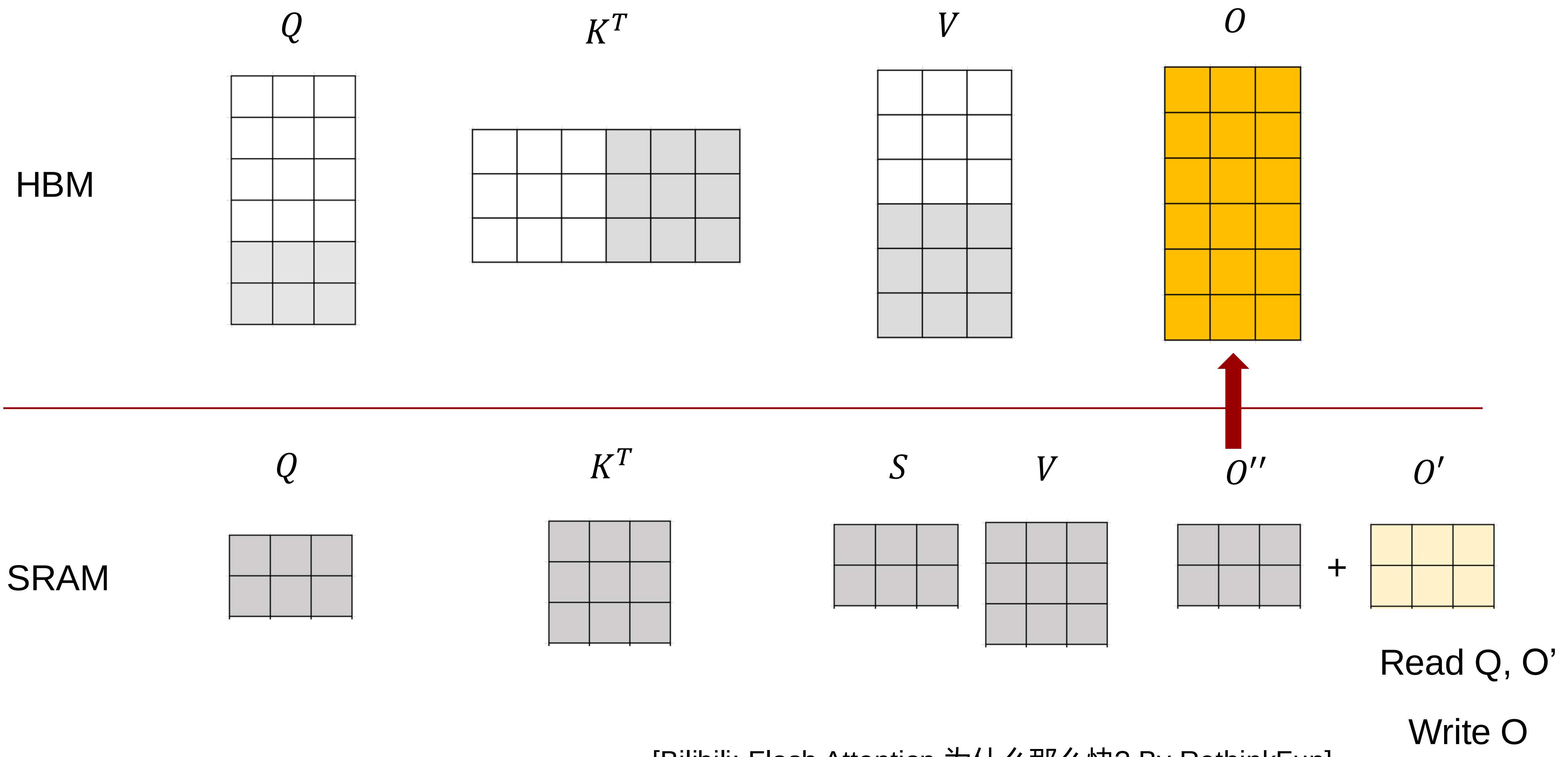


Read Q, O’

Write O

[Bilibili: Flash Attention 为什么那么快? By RethinkFun]

Kernal fusion in simplified attention



[Bilibili: Flash Attention 为什么那么快? By RethinkFun]

HBM accessing comparison

Vanila Attention

Operation	MAC
Load Q and K	$2dN$
Write S	N^2
Read S	N^2
Write P	N^2
Load Q and V	$N^2 + dN$
Write O	dN

$$\mathbf{4N^2 + 4dN}$$

Flash Attention

Operation	MAC
Load Q twice	$2dN$
Load K, V	$2dN$
Write O’	dN
Read O’	dN
Write O	dN

$$\mathbf{7dN}$$

Kernal fusion significantly saves MAC

- When $N \gg d$, FlashAttention significantly saves MAC $\mathbf{4N^2 + 4dN \gg 7dN}$

- The longer the sequence length is, the better that FlashAttention is

- The fundamental reason is that we fusion the intermediate operators, e.g., **do not store $\boldsymbol{S}$**

- But how to handle softmax?

Online softmax

- For numerical stability, the softmax of vector $x \in R^B$ is computed as

$$m(x) := \max_i \quad x_i, \qquad f(x) := \begin{bmatrix} e^{x_1 - m(x)} & \dots & e^{x_B - m(x)} \end{bmatrix},$$

$$\ell(x) := \sum_i f(x)_i, \qquad \mathrm{softmax}(x) := \frac{f(x)}{\ell(x)}.$$

- For vectors $x^{(1)}, x^{(2)}$, the concatenated $x = \begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix} \in R^{2B}$ is computed as follows:

$$m(x) = m(\begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix}) = \max(m(x^{(1)}), m(x^{(2)})),$$

$$f(x) = \begin{bmatrix} e^{m(x^{(1)}) - m(x)} f(x^{(1)}) & e^{m(x^{(2)}) - m(x)} f(x^{(2)}) \end{bmatrix},$$

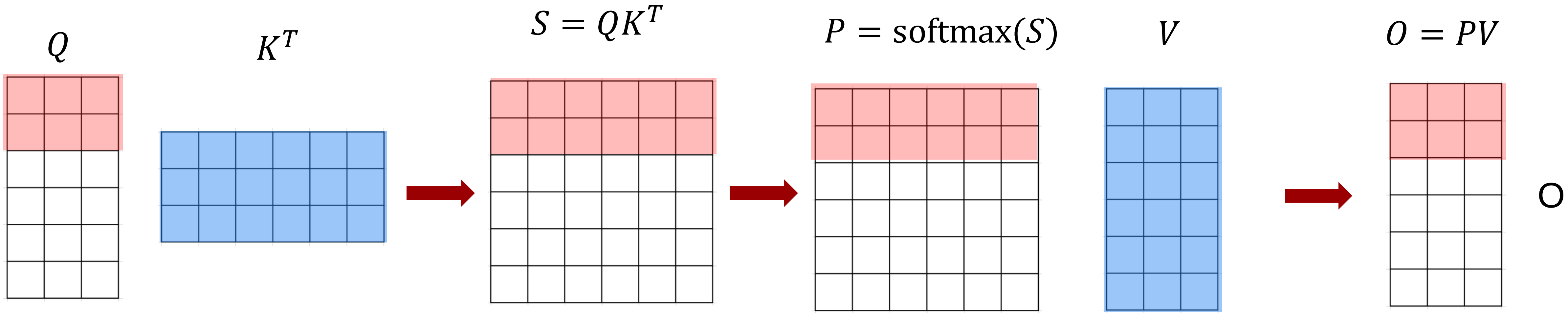
$$\ell(x) = \ell(\begin{bmatrix} x^{(1)} & x^{(2)} \end{bmatrix}) = e^{m(x^{(1)}) - m(x)} \ell(x^{(1)}) + e^{m(x^{(2)}) - m(x)} \ell(x^{(2)}),$$

$$\mathrm{softmax}(x) = \frac{f(x)}{\ell(x)}.$$

We can compute softmax one block at a time

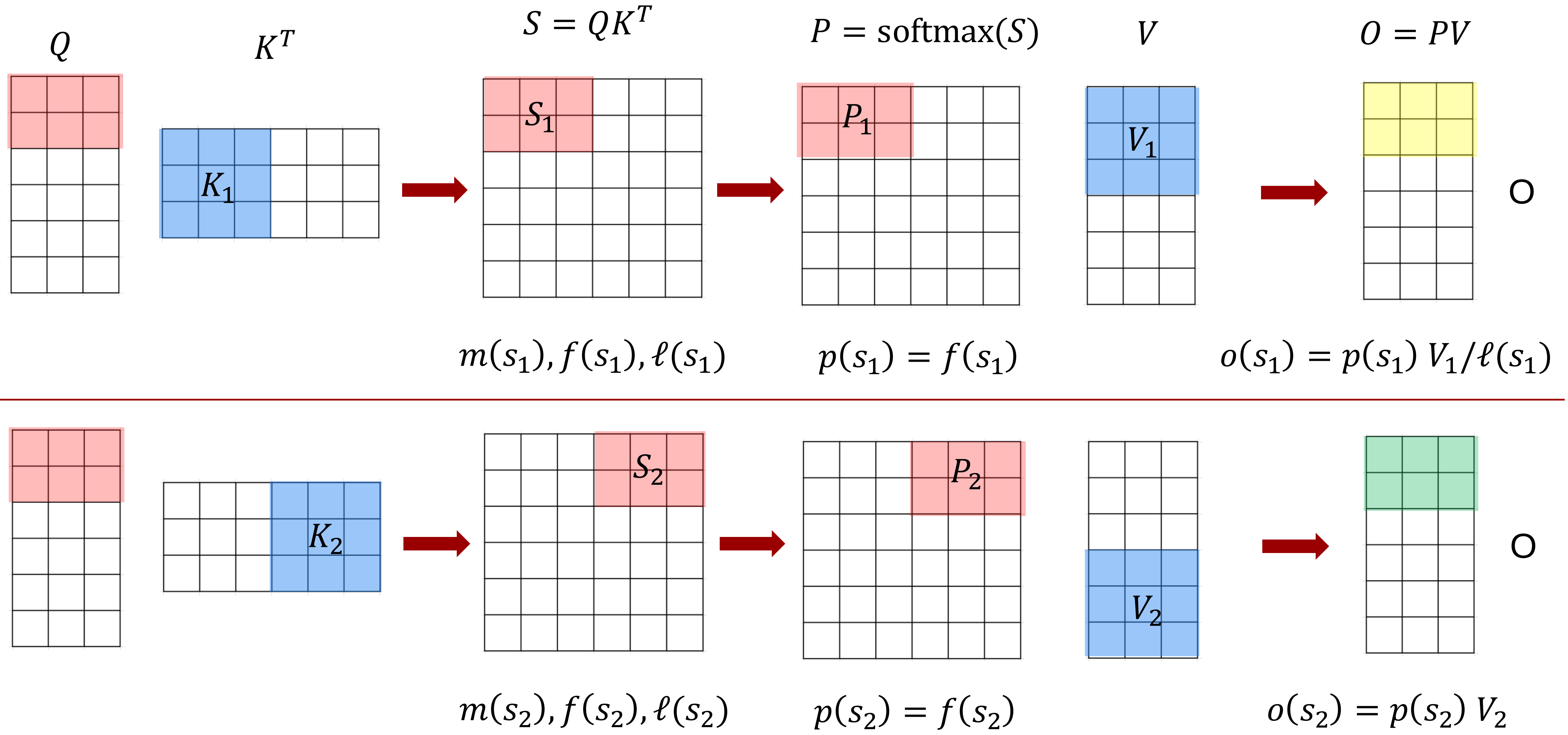
Flash attention

Standard attention needs to access the full matrices K and V



We can decompose K and V for kernel fusion

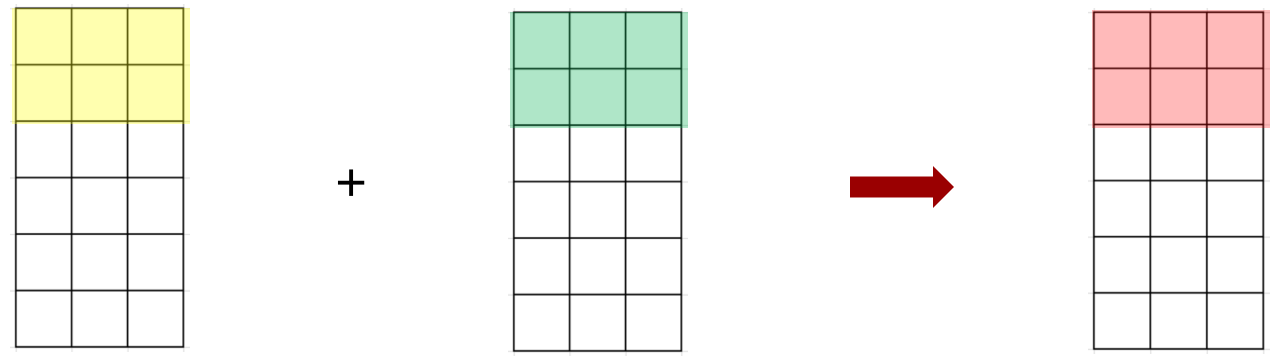
Flash attention



Flash attention

PEKING UNIVERSITY

$o(s_1) = p(s_1)\, V_1/\ell(s_1) \qquad o(s_2) = p(s_2)\, V_2 \qquad O = PV$



$$m(s) = m([s^{(1)} \;\; s^{(2)}]) = \max(m(s^{(1)}), m(s^{(2)})),$$

$$f(s) = \left[e^{m(s_1)-m(s)} f(s_1) \qquad e^{m(s_2)-m(s)} f(s_2)\right]$$

$$\ell(s) = e^{m(s_1)-m(s)} f(s_1) + e^{m(s_2)-m(s)}\, f(s_2)$$

No need to store intermediate results such as S and P !

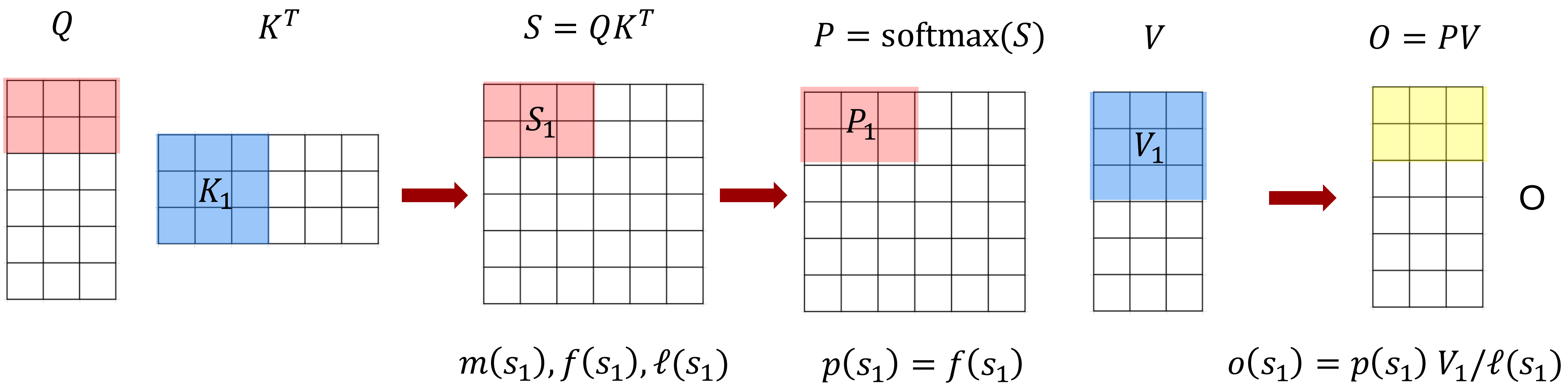
Only stores m, ℓ and O

Recall that $o(s_1) = f(s_1)\, V_1/\ell(s_1)$ and $o(s_2) = f(s_2)\, V_2$

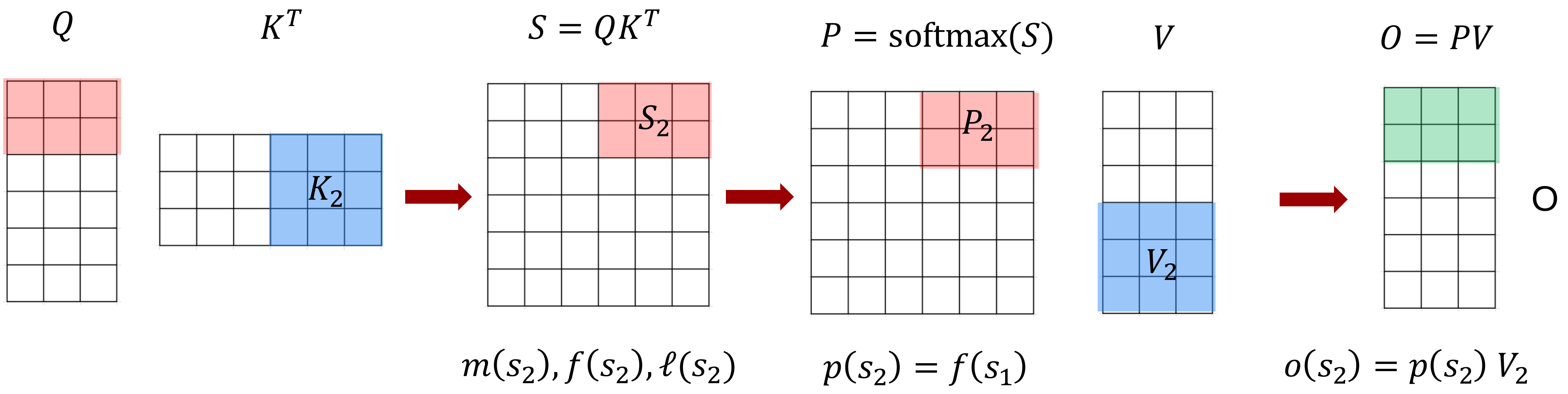
$$o = [e^{m(s_1)-m(s)} f(s_1) V_1 + e^{m(s_2)-m(s)} f(s_2) V_2]/\ell(s)$$

$$= \left[\ell(s_1) e^{m(s_1)-m(s)} o(s_1) + e^{m(s_2)-m(s)} f(s_2) V_2\right]/\ell(s)$$

Flash attention



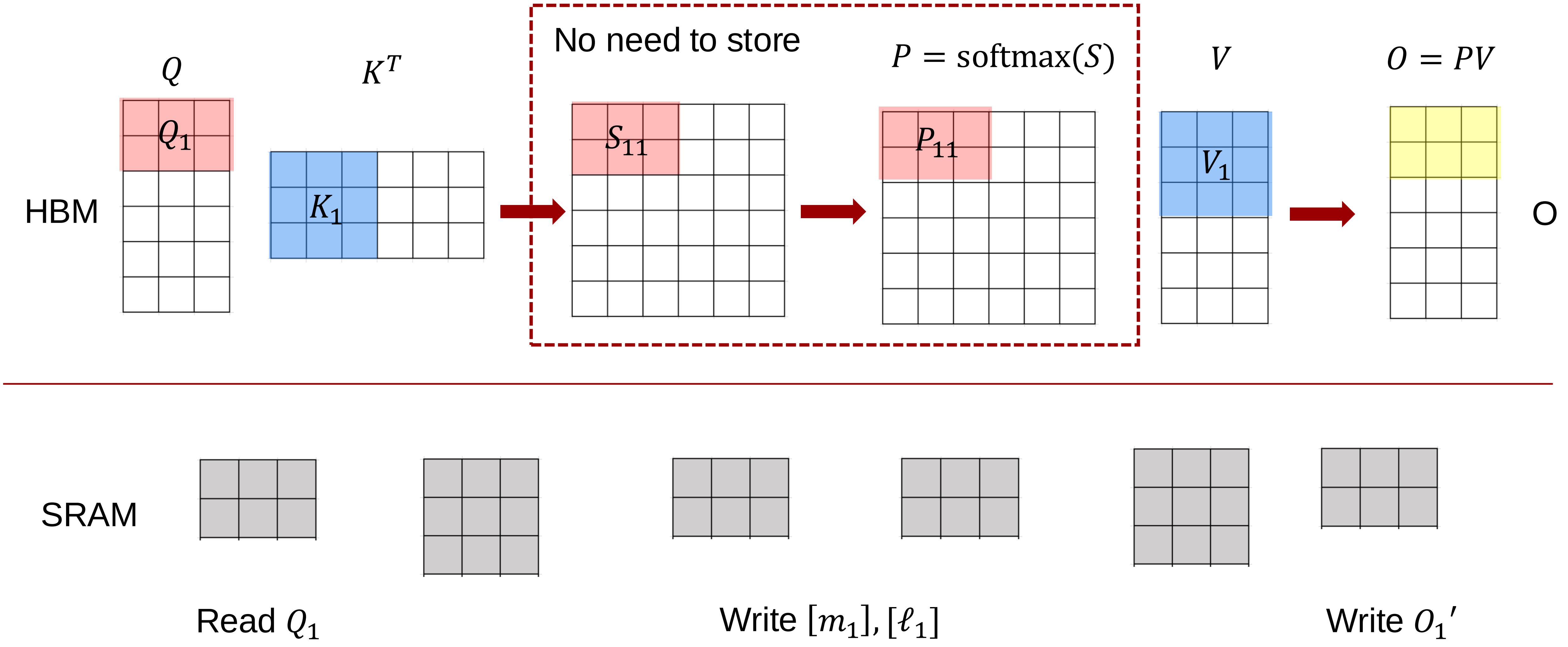
Flash attention



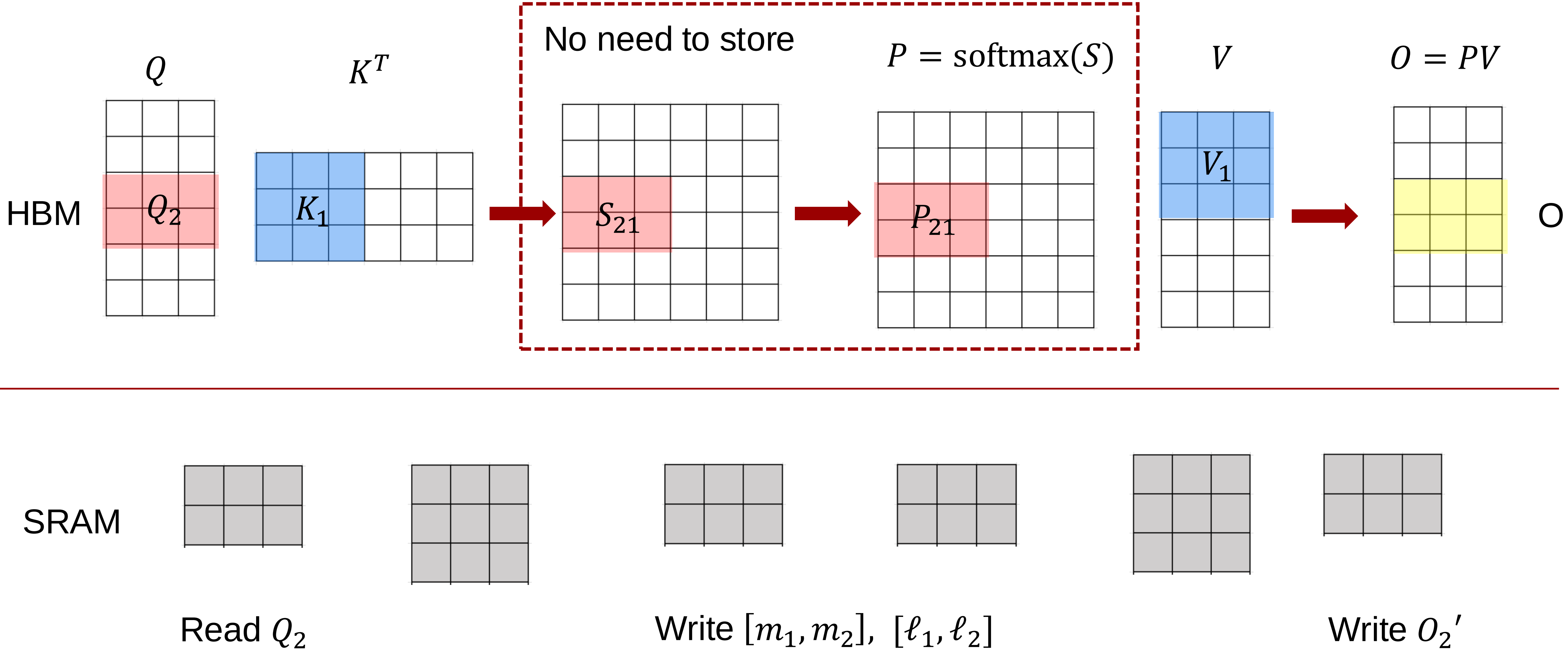
If we fix Q and traverse K and V, we need to access 2Nd HBM

To save HBM, we can fix K and V but traverse Q, similar to scenarios in Page 13 – Page 18.

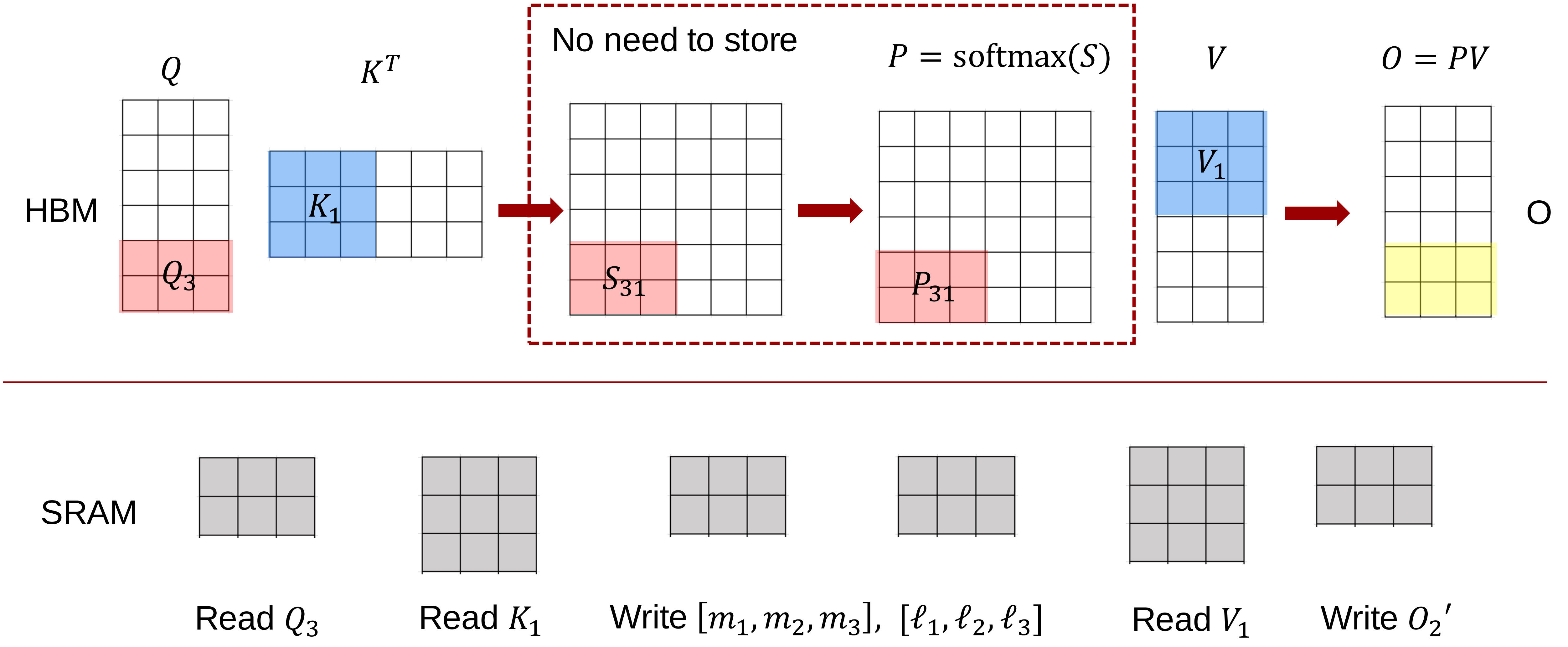
Flash attention



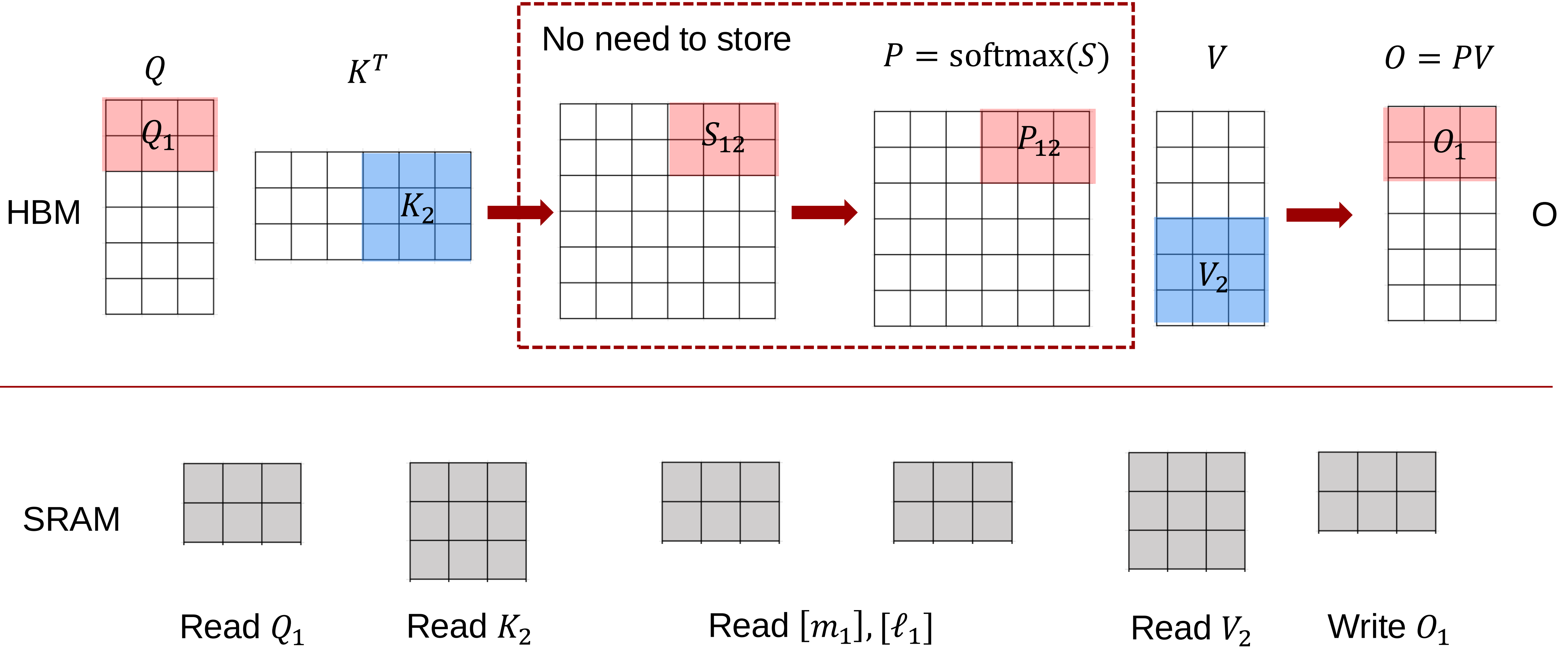
Flash attention



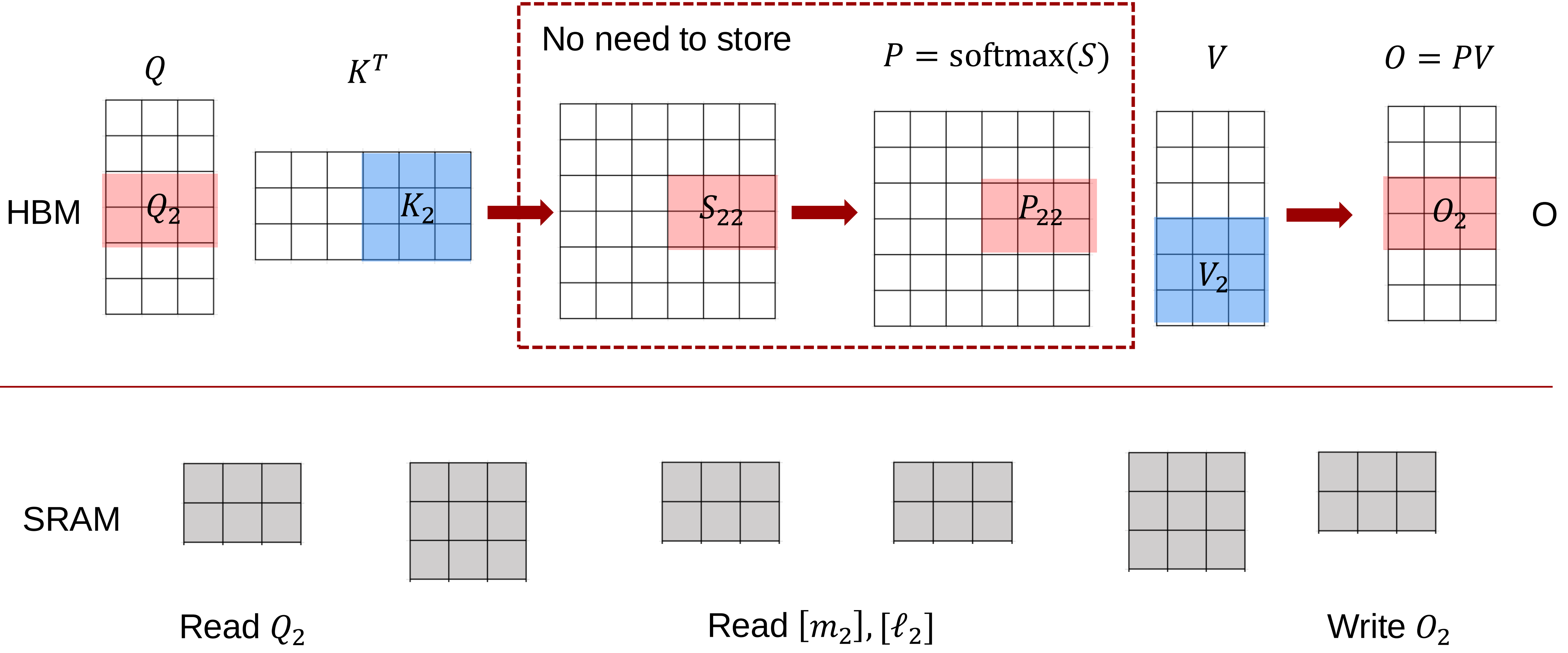
Flash attention



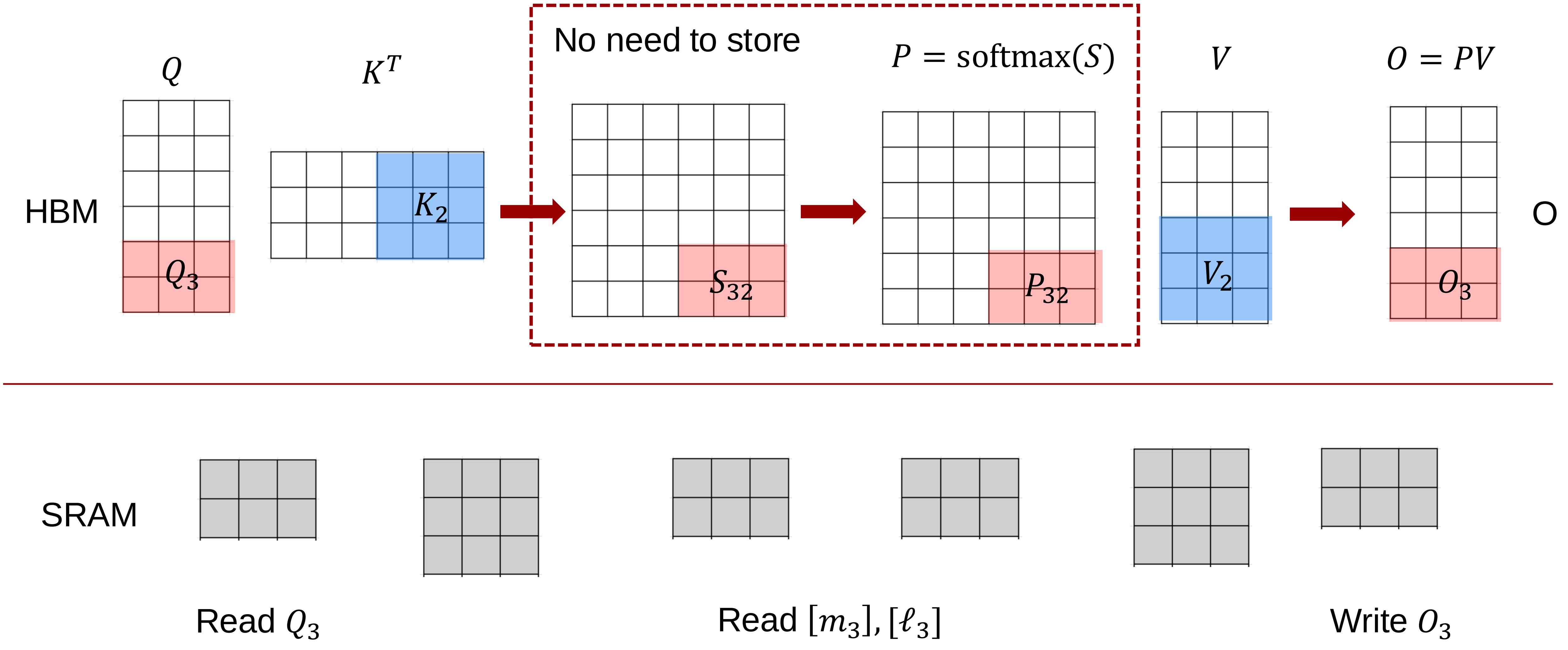
Flash attention



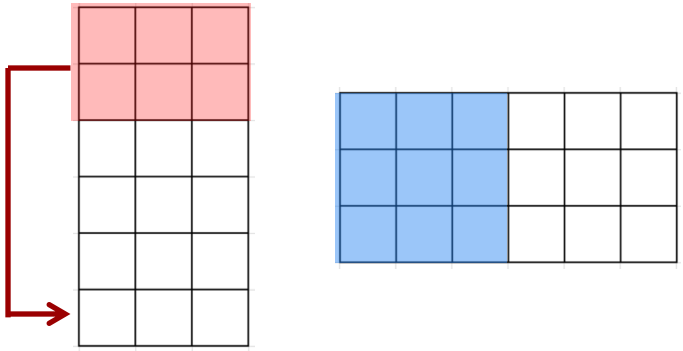
Flash attention



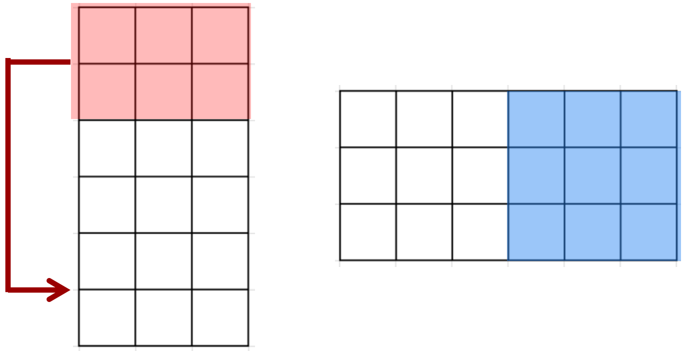
Flash attention



Flash attention: HBM cost

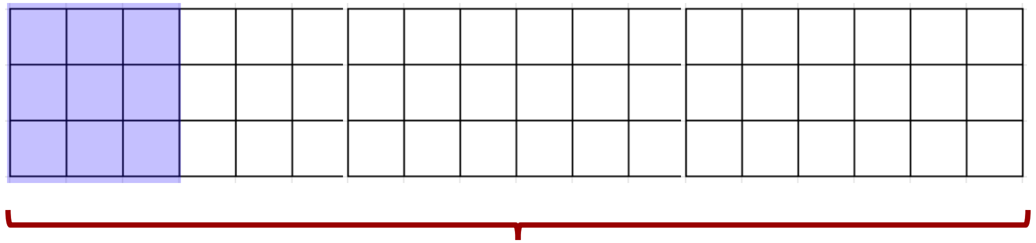


Fix K_1, traverse Q



Fix K_2, traverse Q

M parameters



Nd parameters

$$\boxed{\frac{Nd}{M} \times Nd = N^2 d^2 / M}$$

HBM cost

Flash attention: HBM and memory cost

HBM in standard attention

$$\boldsymbol{N^2 + dN}$$

HBM in flash attention

$$\boldsymbol{N^2 d^2 / M}$$

Since $M \gg d^2$, Flash attention saves significant HBM cost; M=10^6 while $d = 64 \sim 128$

Memory in standard attention

$$\boldsymbol{N^2 d}$$

Memory in flash attention

$$\boldsymbol{Nd}$$

No need to materialize attention score and probability, saves significant memory

Flash attention: algorithm

北京大学 PEKING UNIVERSITY

Algorithm 1 FlashAttention

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N\times d}$ in HBM, on-chip SRAM of size M.

Set block sizes $B_c = \left\lceil \frac{M}{4d} \right\rceil, B_r = \min\left(\left\lceil \frac{M}{4d} \right\rceil, d\right)$.

Initialize $\mathbf{O} = (0)_{N\times d} \in \mathbb{R}^{N\times d}, \ell = (0)_N \in \mathbb{R}^N, m = (-\infty)_N \in \mathbb{R}^N$ in HBM.

Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.

Divide $\mathbf{O}$ into T_r blocks $\mathbf{O}_i, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_i, \dots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks $m_1, \dots, m_{T_r}$ of size B_r each.

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

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

  for $1 \le i \le T_r$ **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.

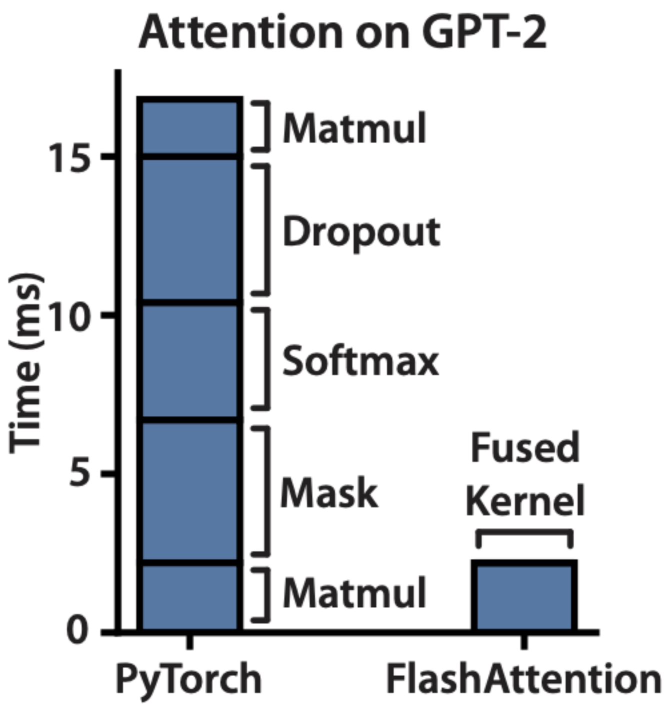
    Write $\ell_i \leftarrow \ell_i^{\mathrm{new}}$, $m_i \leftarrow m_i^{\mathrm{new}}$ to HBM.

  end for

end for

Return $\mathbf{O}$.

Experiments



BERT 8 * A100; target accuracy of 72.0%

BERT Implementation	Training time (minutes)
Nvidia MLPerf 1.1 [58]	20.0 ± 1.5
FlashAttention (ours)	$\mathbf{17.4} \pm 1.4$

Experiments

Model implementations	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Huggingface [87]	18.2	9.5 days (1.0×)
GPT-2 small - Megatron-LM [77]	18.2	4.7 days (2.0×)
GPT-2 small - FlashAttention	18.2	**2.7 days (3.5×)**
GPT-2 medium - Huggingface [87]	14.2	21.0 days (1.0×)
GPT-2 medium - Megatron-LM [77]	14.3	11.5 days (1.8×)
GPT-2 medium - FlashAttention	14.3	**6.9 days (3.0×)**

Experiments

Long range arena

Models	ListOps	Text	Retrieval	Image	Pathfinder	Avg	Speedup
Transformer	36.0	63.6	81.6	42.3	72.7	59.3	-
FlashAttention	37.6	63.9	81.4	43.5	72.7	59.8	2.4×
Block-sparse FlashAttention	37.0	63.0	81.3	43.6	73.3	59.6	**2.8×**
Linformer [84]	35.6	55.9	77.7	37.8	67.6	54.9	2.5×
Linear Attention [50]	38.8	63.2	80.7	42.6	72.5	59.6	2.3×
Performer [12]	36.8	63.6	82.2	42.1	69.9	58.9	1.8×
Local Attention [80]	36.1	60.2	76.7	40.6	66.6	56.0	1.7×
Reformer [51]	36.5	63.8	78.5	39.6	69.4	57.6	1.3×
Smyrf [19]	36.1	64.1	79.0	39.6	70.5	57.9	1.7×

北京大學
PEKING UNIVERSITY

Thank you!

Kun Yuan homepage: https://kunyuan827.github.io/