High Performance CNN Inference

CS 217 Stanford

Ardavan Pedram

Analysis of Convolution Approaches

1. **GEMM (default)**

Pros:
- Generic and stable
- Easy to implement (problem mapped into a BLAS call)
- Optimized solution if good BLAS is provided

Cons:
- Additional memory to store the intermediate data
- Rely heavily on optimized BLAS

2. **Spatial domain**

Pros:
- Avoids additional memory copy
- Speedy with optimized code

Cons:
- Rely on individually optimized kernels according to given params, or even given HW architecture

3. **FFT domain**

Pros:
- Lower computational complexity

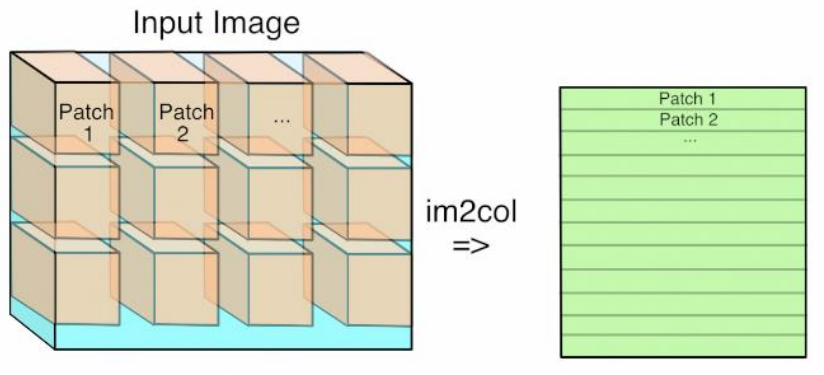
Cons:
- Additional memory to save FFT data
- Overhead is big for small kernel size, or large stride

https://www.slideshare.net/IntelSoftware/clcaffe-unleashing-the-power-of-intel-graphics-for-deep-learning-acceleration

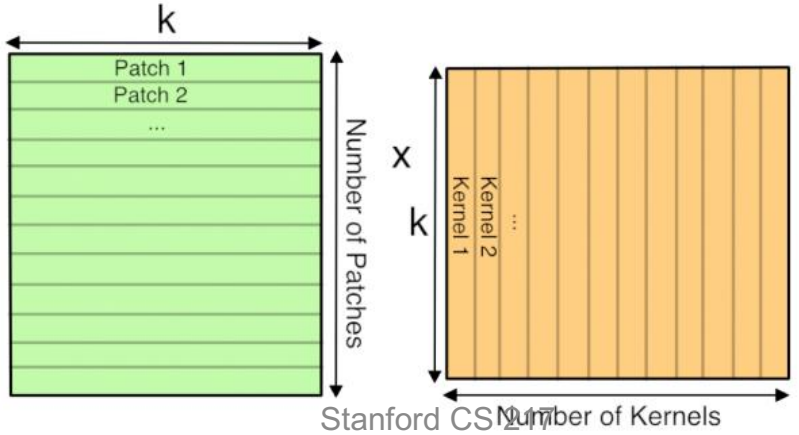
GEMM Based Convolution

Flatten input data and kernels, solve the convolution as a matrix multiplication problem

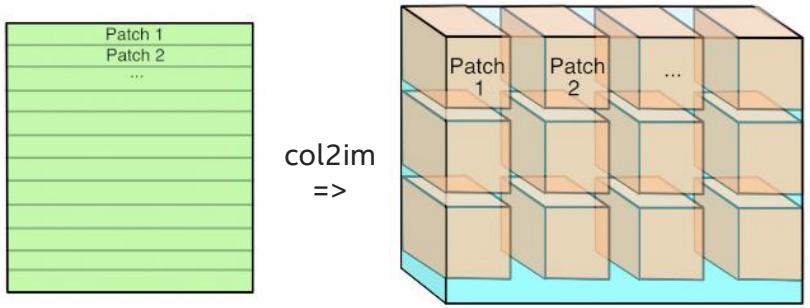
Step1: data flattening



Step2: matrix multiply
Usually mapped into a BLAS (Basic Linear Algebra Subprogram) call

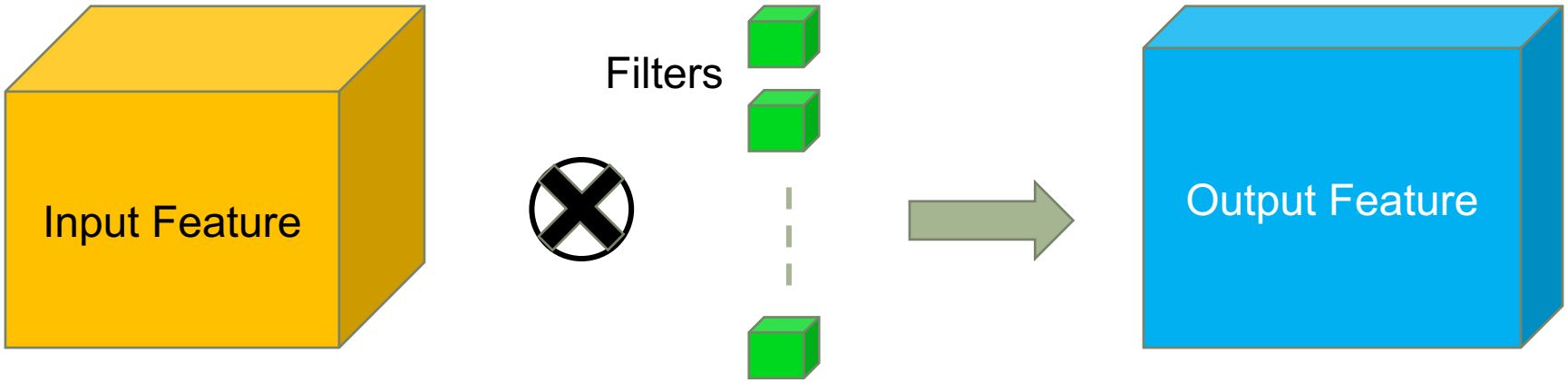


Step3: data unflattening



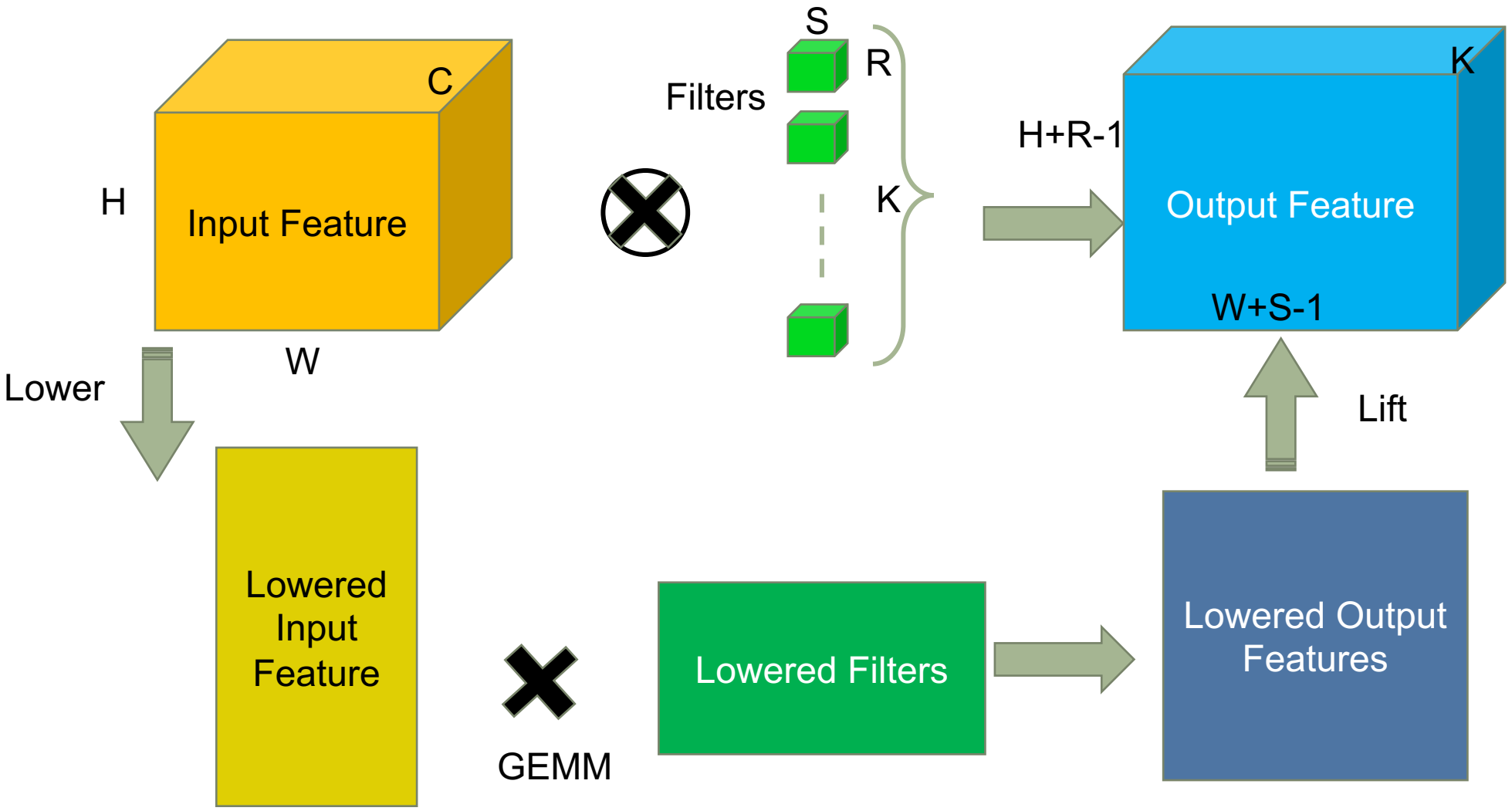
<Image source: http://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/>

Caffe Con Troll [Hadjis et. al.]



Can We turn the problem into GEMM?

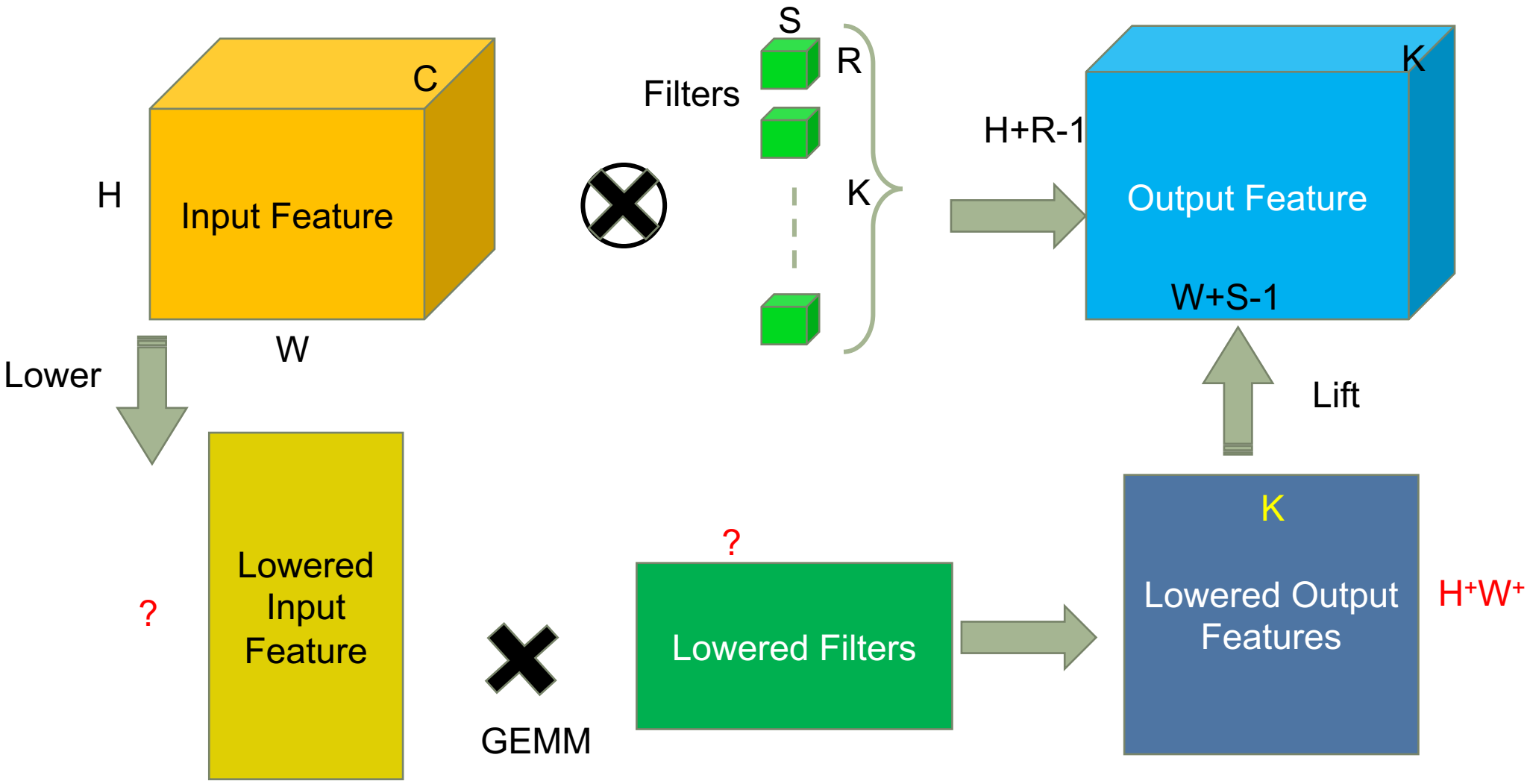
Im2col


C
H
Input Feature
W
Lower
Lowered Input Feature
S
R
Filters
K
GEMM
Lowered Filters
H+R-1
K
Output Feature
W+S-1
Lift
Lowered Output Features

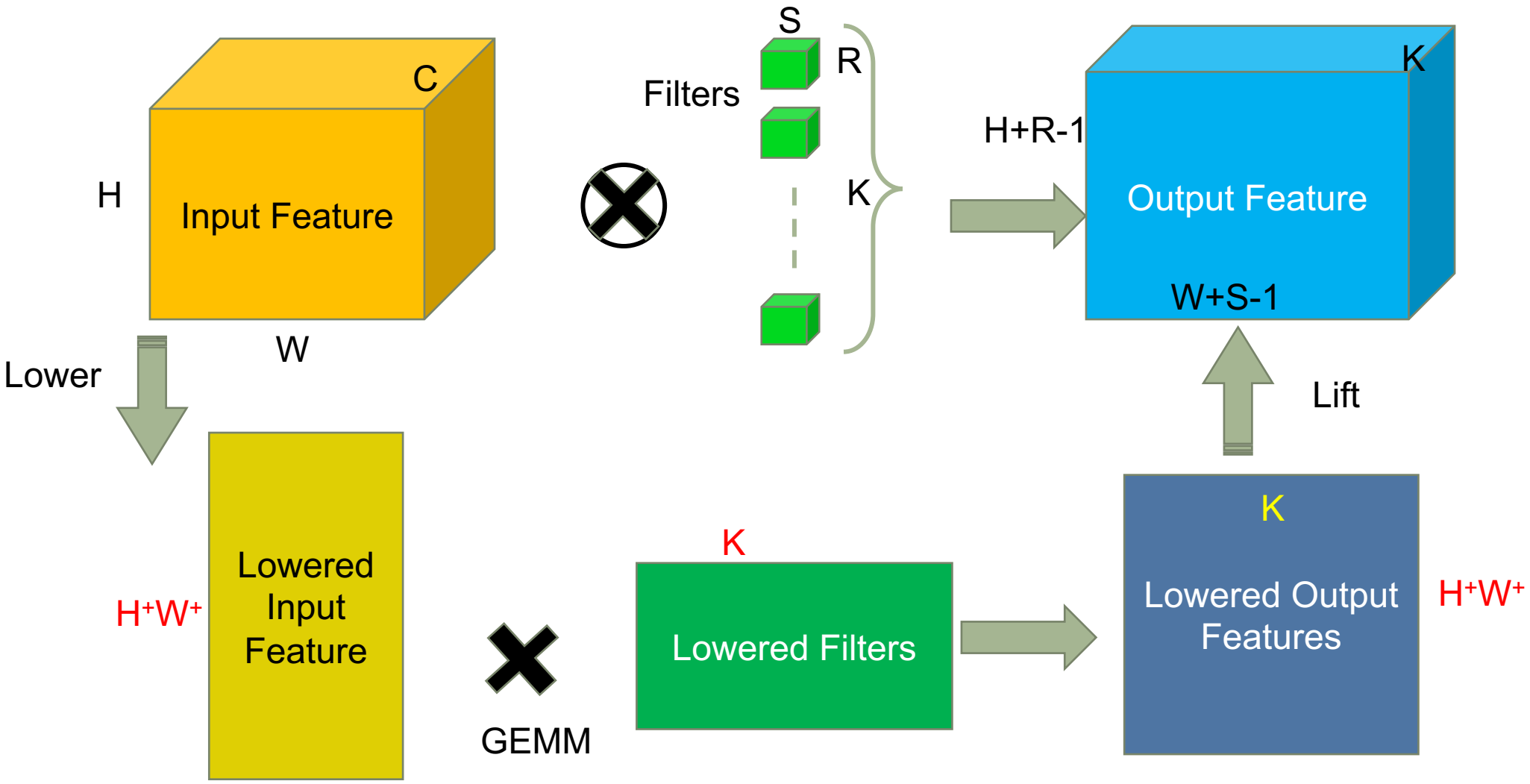
Turn a Tensor into Matrix

Remember What was a reduction dimension?

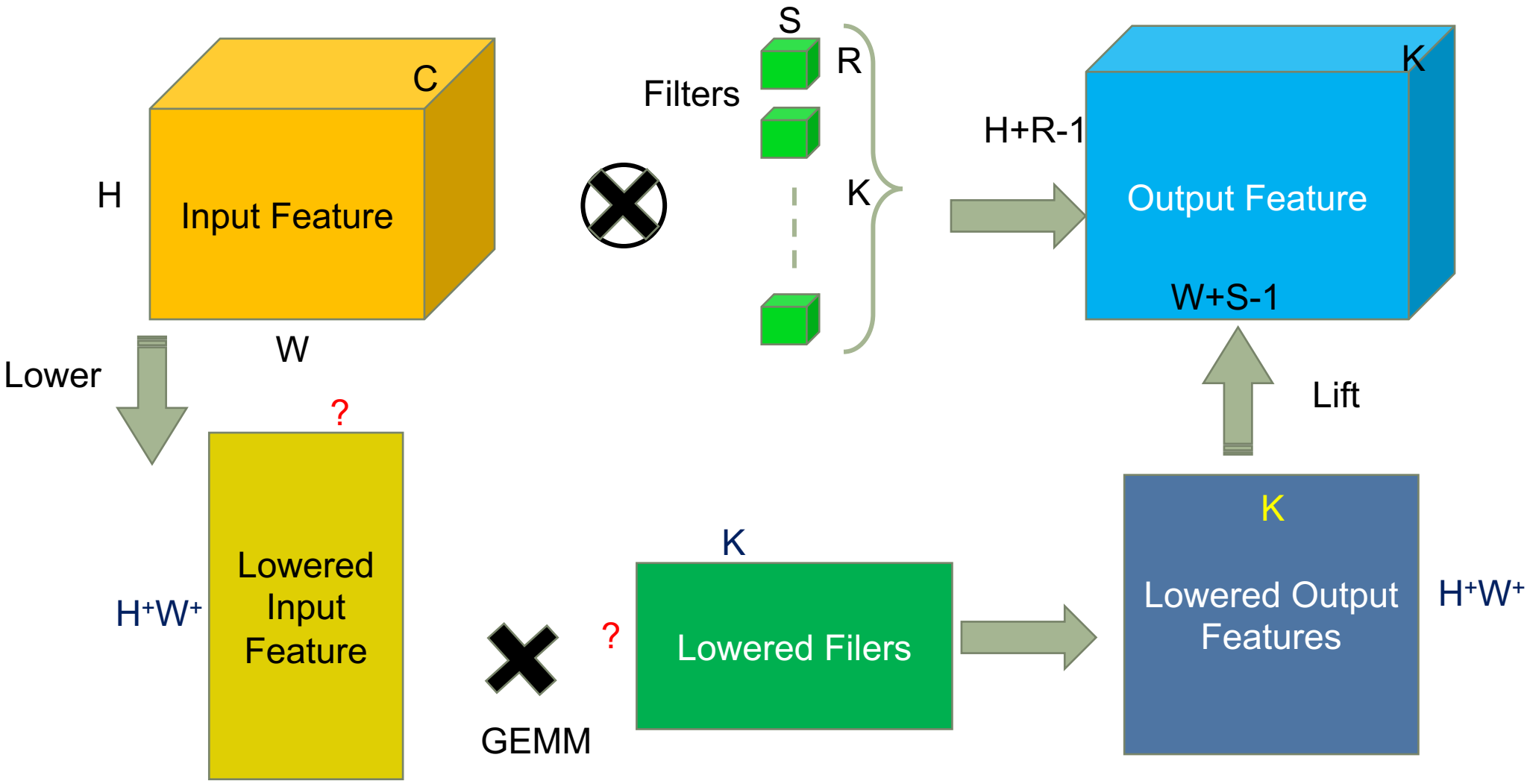
Im2col


C
H
Input Feature
W
Lower
?
Lowered Input Feature
S
R
Filters
K
GEMM
?
Lowered Filters
H+R-1
K
Output Feature
W+S-1
Lift
K
Lowered Output Features
H+W+

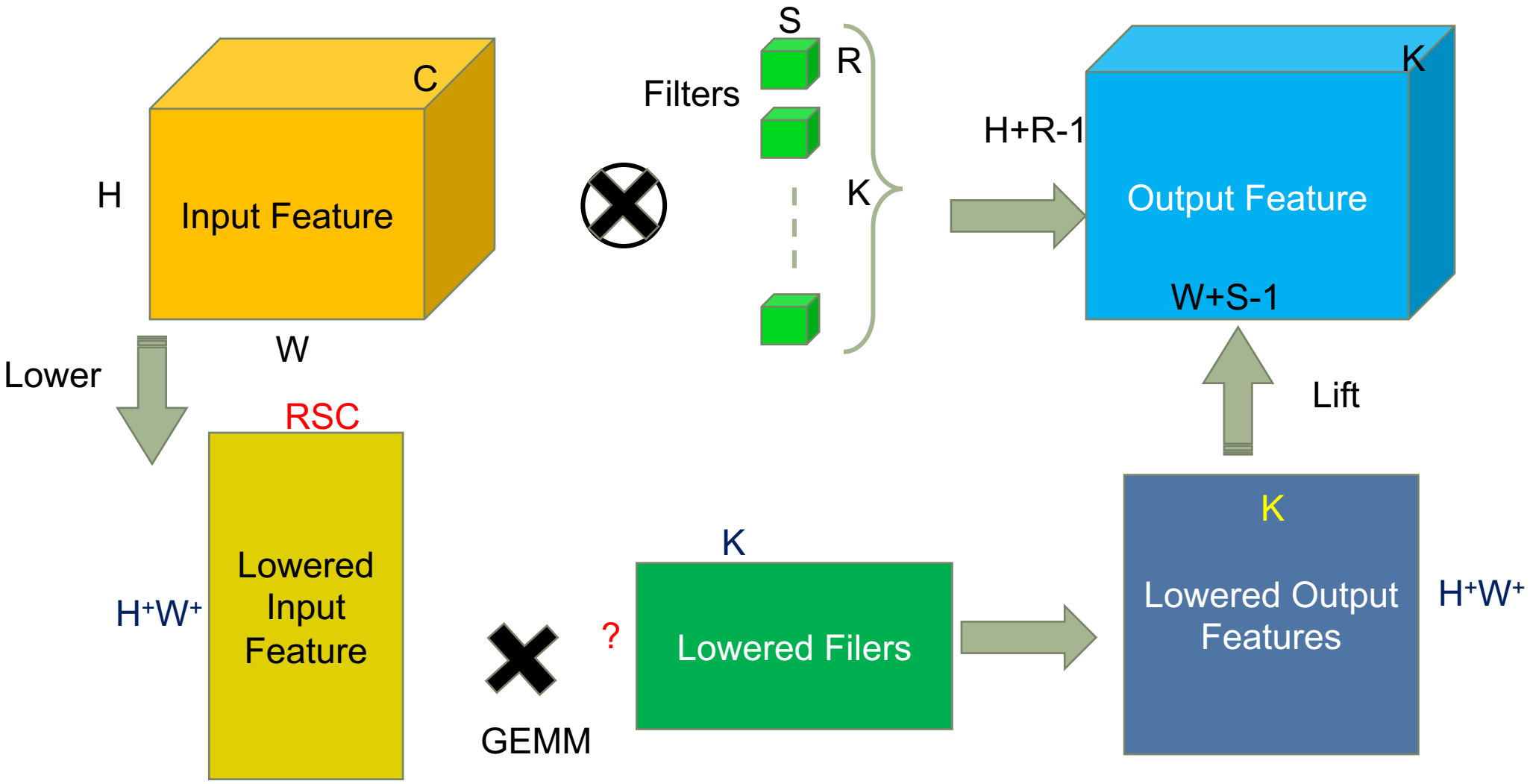
Im2col


C
H
Input Feature
W
Lower
S
R
Filters
K
H+R-1
K
Output Feature
W+S-1
Lift
H+W+
Lowered Input Feature
GEMM
K
Lowered Filters
K
Lowered Output Features
H+W+

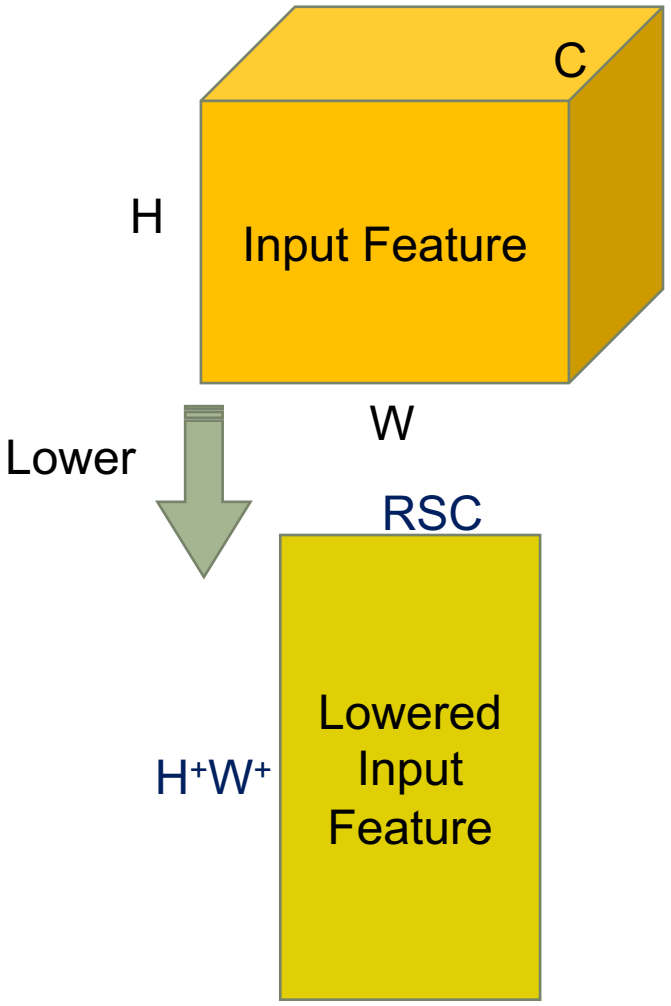
Im2col


C
H
Input Feature
W
Lower
?
H⁺W⁺
Lowered Input Feature
S
R
Filters
K
H+R-1
K
Output Feature
W+S-1
Lift
K
Lowered Output Features
H⁺W⁺
K
Lowered Filers
?
GEMM

Im2col


C
H
Input Feature
W
Lower
RSC
H⁺W⁺
Lowered Input Feature
S
R
Filters
K
H+R-1
K
Output Feature
W+S-1
Lift
K
Lowered Output Features
H⁺W⁺
K
Lowered Filers
?
GEMM

What Really Happened?



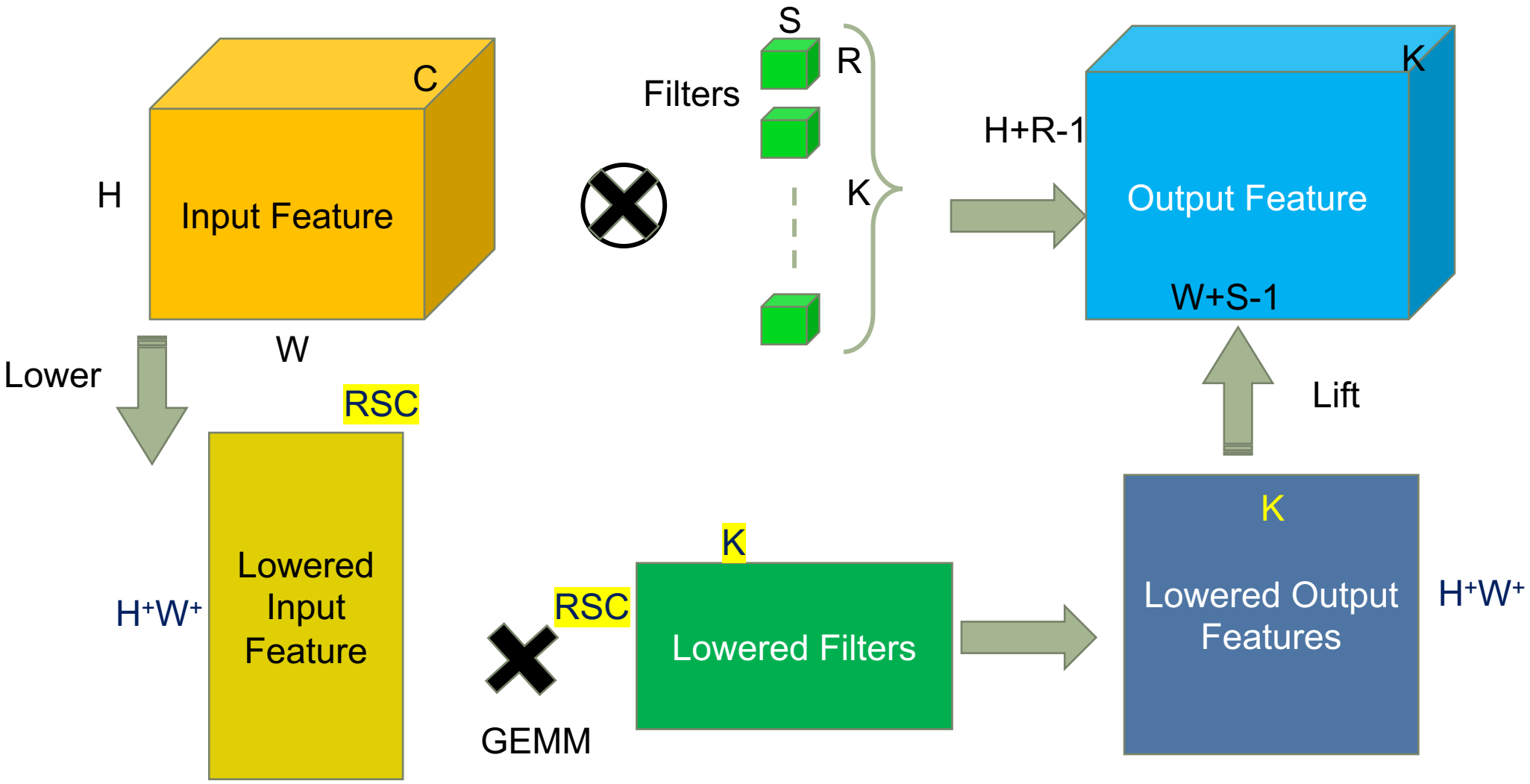
What was the tensor size?

What is the lowered matrix size?

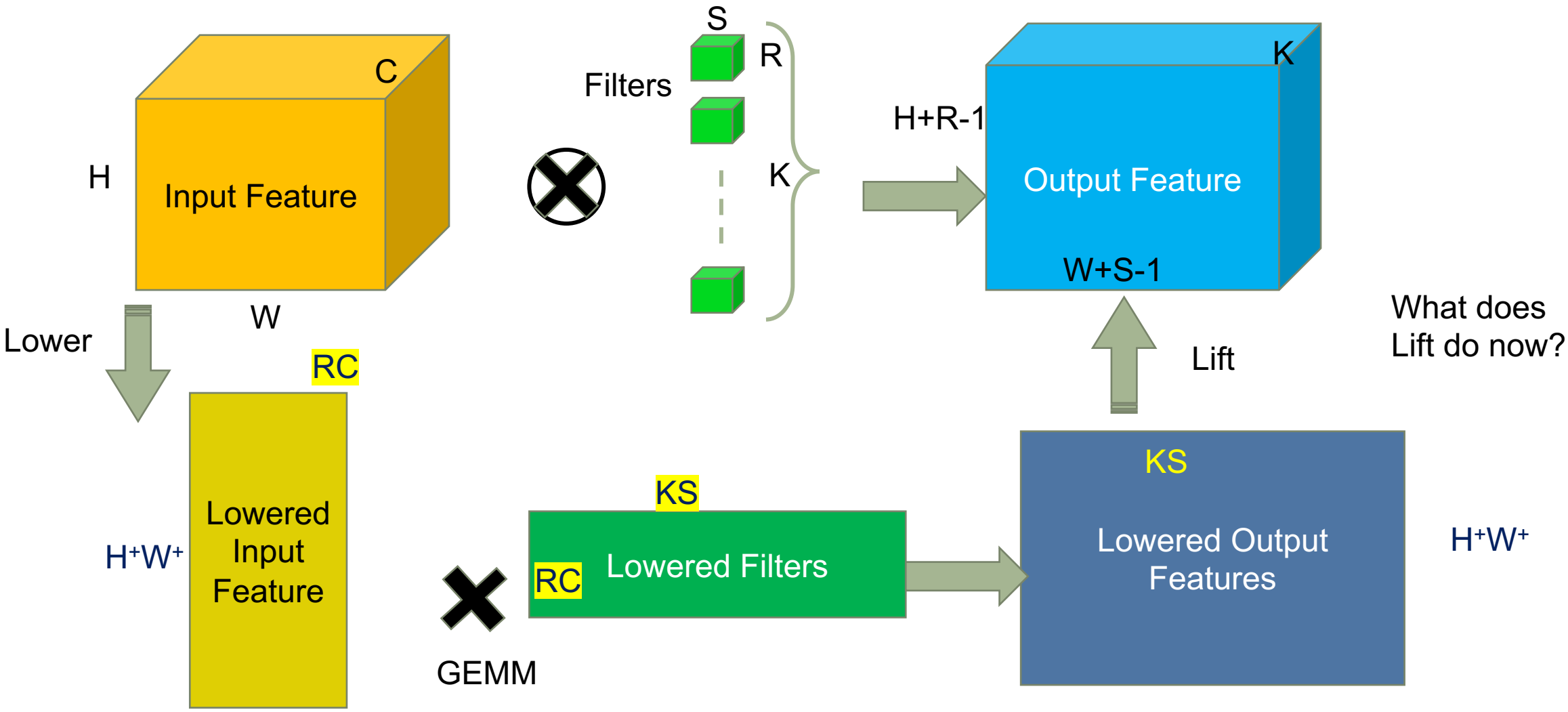
What does lowering mean?

What is the impact on Memory?

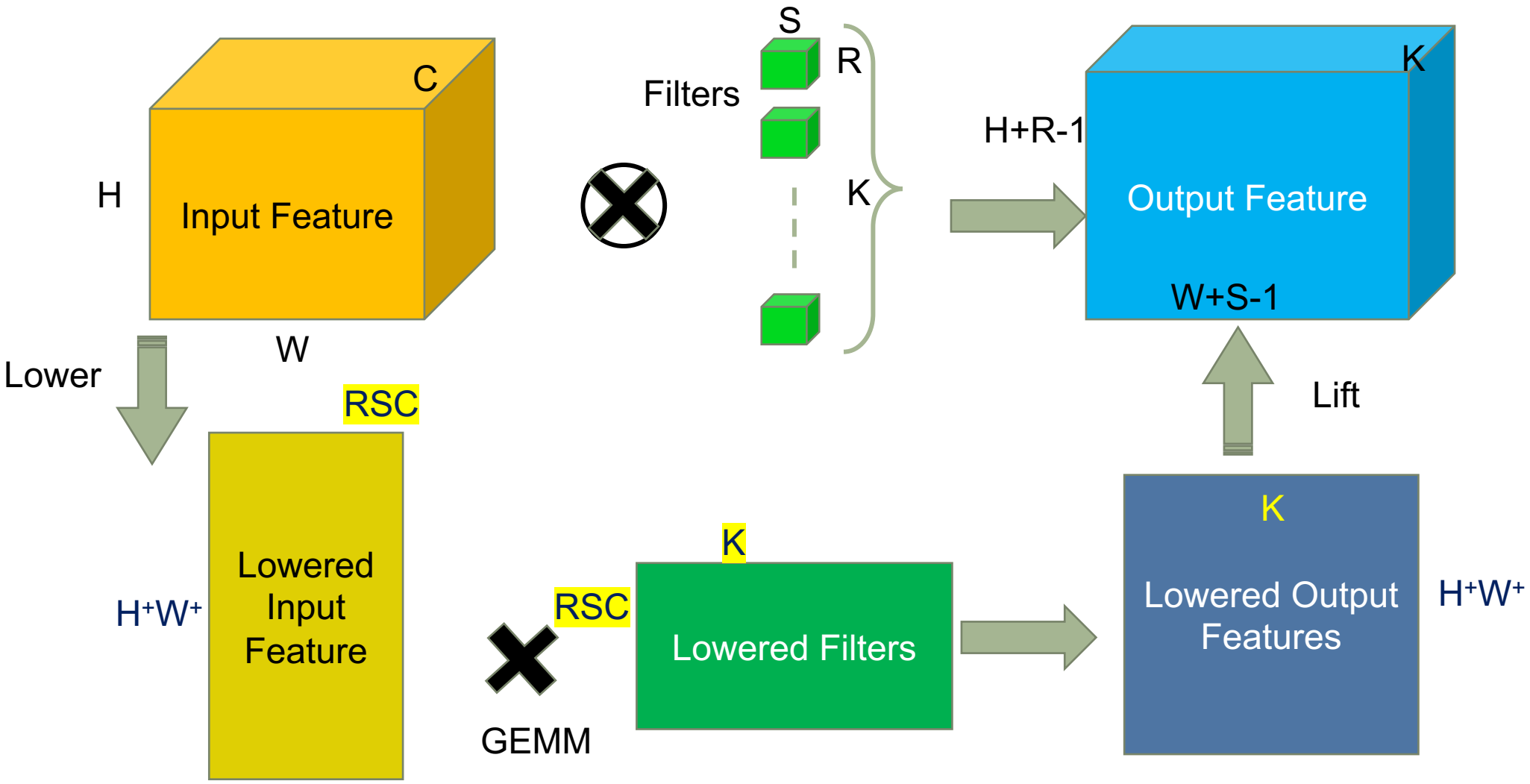
Are there other options?



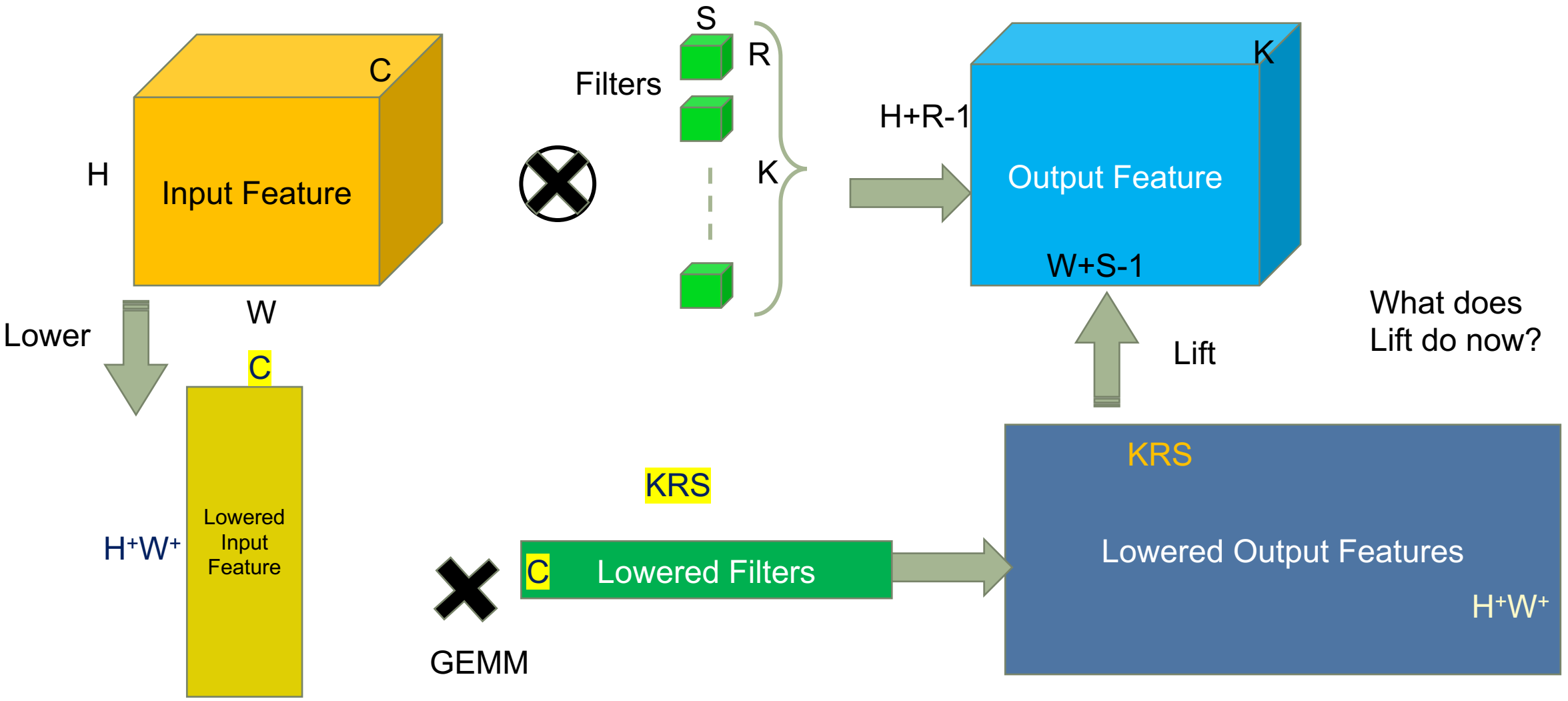
Play with the reduction dimension



Are there other options?



Is there another option?



Conventional approach penalizes memory to optimize performance

casts convolution to Matrix-Matrix-Multiplication, in order to utilize high performance MMM implementations in Basic-Linear-Algebra-Subroutines(BLAS) libraries

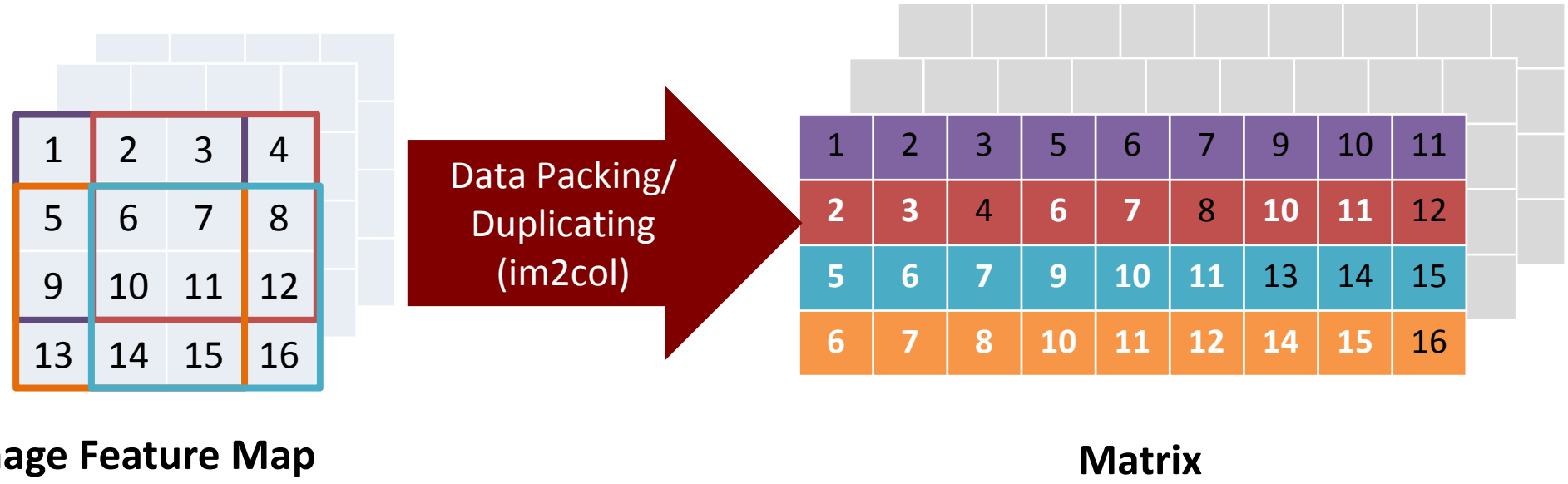


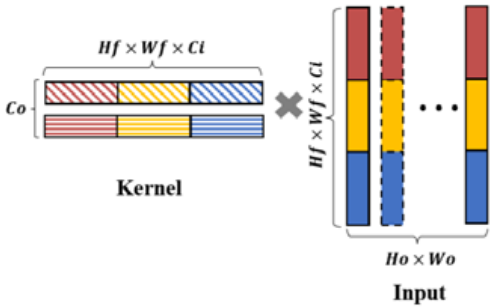
Image Feature Map

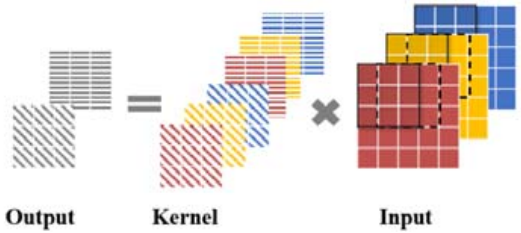
Matrix

Source: J. Zhang et. al

Direct Convolution is Better

Higher performance, zero memory overheads





	Matrix-Matrix-Multiplication	Direct Convolution
Packing	**Yes** • Over 10x additional memory for image data • Performance penalty	**No** • Zero memory overheads • No performance penalty
Computation Performance	**Less than** expected theoretic peak of GEMM	Close to system's theoretic peak

Source: J. Zhang et. al

A Deeper Dive Into Convolution Loops

- Look at GEMM Blocking
  - It is Simpler

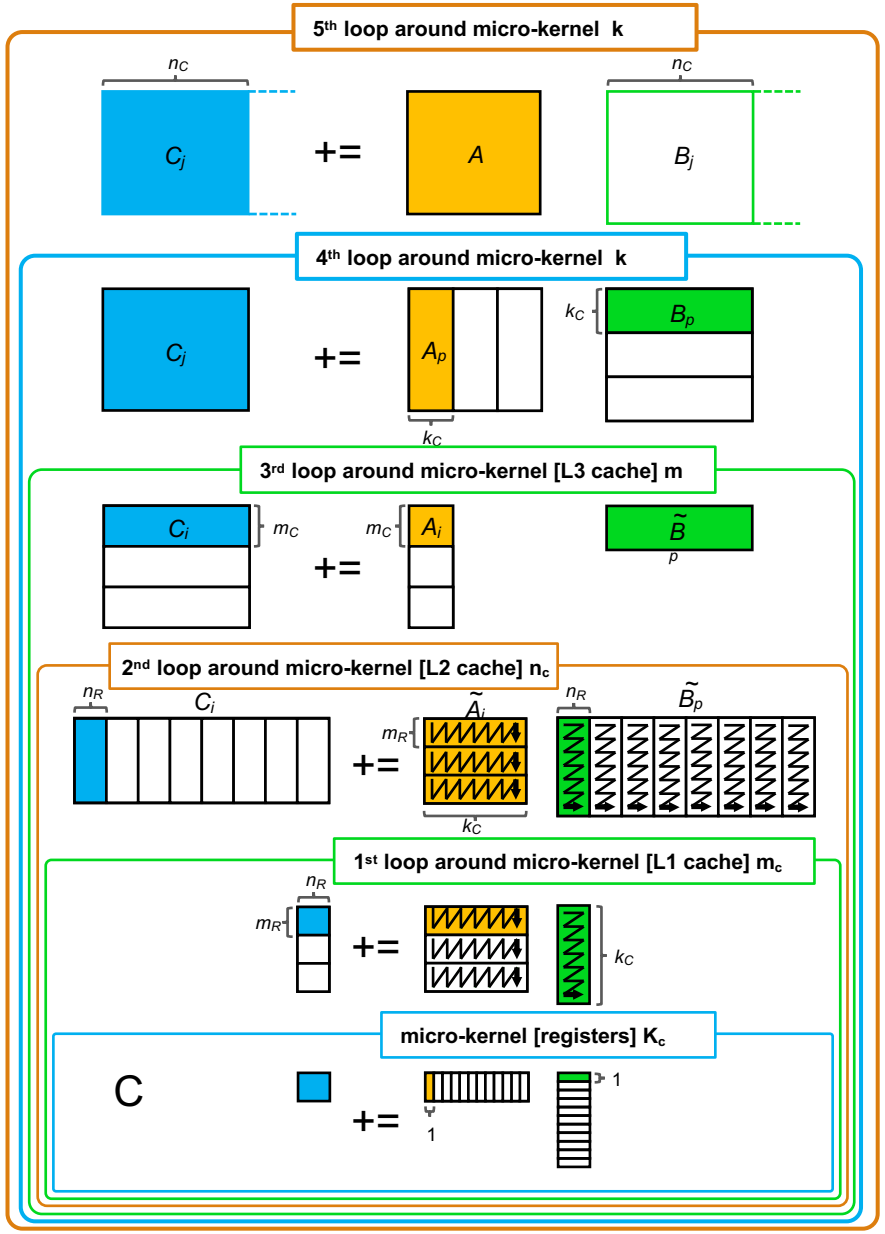
- Directly Analyze CNN Blocking

Recap of GEMM Loops

- Rules for each new character
  - Buffers
  - Re-fetch rate

- $m_r\ n_r\ k_c\ m_c\ n_c\ m$ k n
- $m_0\ n_0\ k_0\ m_1\ n_1\ m_2\ k_1$ n_2

$$
\begin{array}{l}
\textbf{for } j_c = 0, \ldots, n-1 \textbf{ in steps of } n_c \\
\quad \textbf{for } p_c = 0, \ldots, k-1 \textbf{ in steps of } k_c \\
\quad\quad B(p_c : p_c + k_c - 1, j_c : j_c + n_c - 1) \rightarrow B_c \\
\quad\quad \textbf{for } i_c = 0, \ldots, m-1 \textbf{ in steps of } m_c \\
\quad\quad\quad A(i_c : i_c + m_c - 1, p_c : p_c + k_c - 1) \rightarrow A_c \\
\hline
\quad\quad\quad \textbf{for } j_r = 0, \ldots, n_c - 1 \textbf{ in steps of } n_r \\
\quad\quad\quad\quad \textbf{for } i_r = 0, \ldots, m_c - 1 \textbf{ in steps of } m_r \\
\hline
\quad\quad\quad\quad\quad \textbf{for } p_r = 0, \ldots, k_c - 1 \textbf{ in steps of } 1 \\
\quad\quad\quad\quad\quad\quad C_c(i_r : i_r + m_r - 1, j_r : j_r + n_r - 1) \\
\quad\quad\quad\quad\quad\quad\quad += A_c(i_r : i_r + m_r - 1, p_r) \\
\quad\quad\quad\quad\quad\quad\quad\quad \cdot\ B_c(p_r, j_r : j_r + n_r - 1)
\end{array}
$$

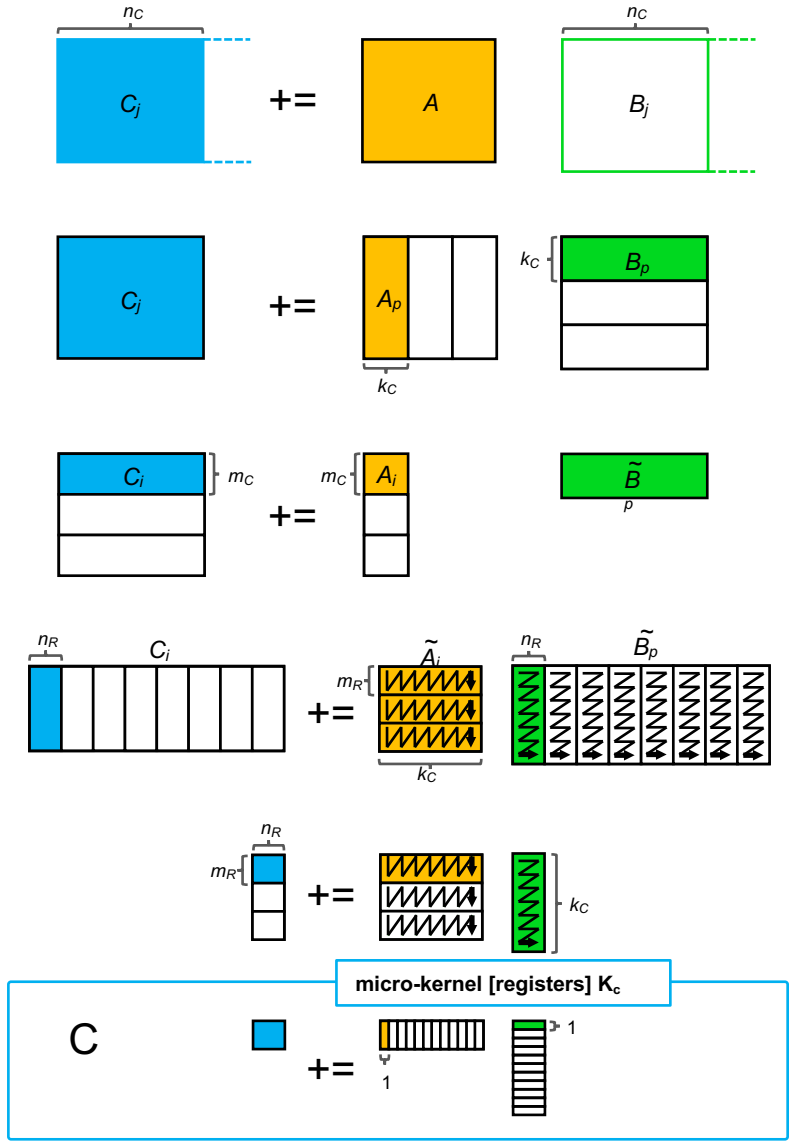


Recap of GEMM Loops

- Rules for each new character
  - Buffers
  - Re-fetch rate

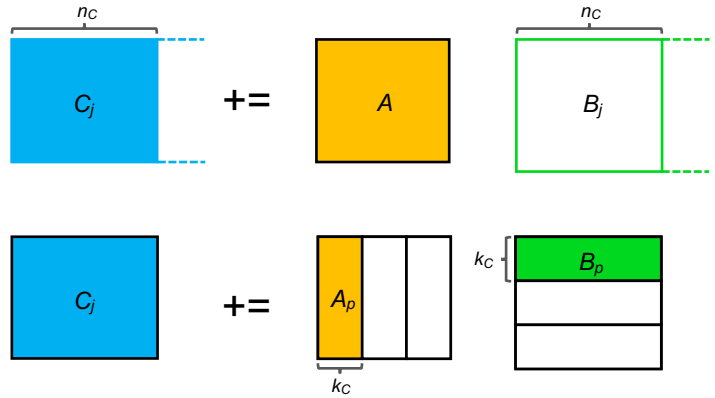
- $m_r\, n_r\, k_c$
- $m_0\, n_0\, k_0$

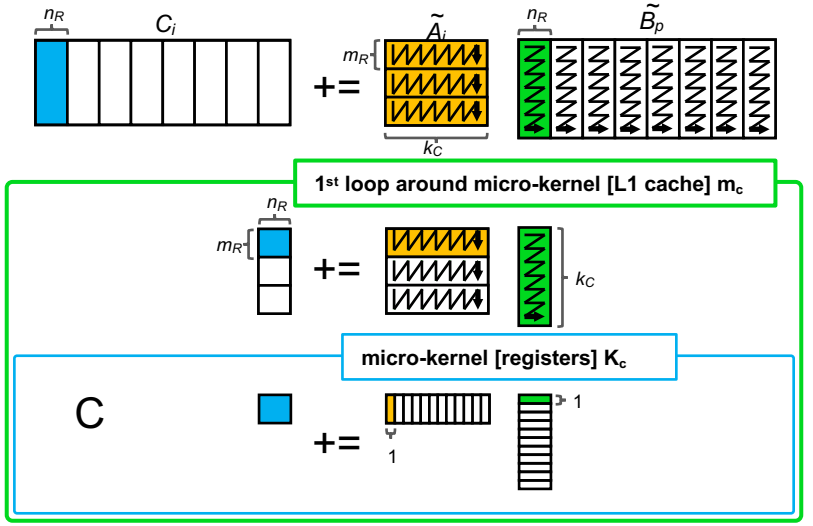
$$
\begin{array}{l}
\textbf{for } j_c = 0, \ldots, n-1 \textbf{ in steps of } n_c \\
\quad \textbf{for } p_c = 0, \ldots, k-1 \textbf{ in steps of } k_c \\
\quad\quad B(p_c : p_c + k_c - 1, j_c : j_c + n_c - 1) \rightarrow B_c \\
\quad\quad \textbf{for } i_c = 0, \ldots, m-1 \textbf{ in steps of } m_c \\
\quad\quad\quad A(i_c : i_c + m_c - 1, p_c : p_c + k_c - 1) \rightarrow A_c \\
\hline
\quad\quad\quad \textbf{for } j_r = 0, \ldots, n_c - 1 \textbf{ in steps of } n_r \\
\quad\quad\quad\quad \textbf{for } i_r = 0, \ldots, m_c - 1 \textbf{ in steps of } m_r \\
\quad\quad\quad\quad\quad \textbf{for } p_r = 0, \ldots, k_c - 1 \textbf{ in steps of } 1 \\
\quad\quad\quad\quad\quad\quad C_c(i_r : i_r + m_r - 1, j_r : j_r + n_r - 1) \\
\quad\quad\quad\quad\quad\quad\quad \mathrel{+}= A_c(i_r : i_r + m_r - 1, p_r) \\
\quad\quad\quad\quad\quad\quad\quad \cdot\ B_c(p_r, j_r : j_r + n_r - 1)
\end{array}
$$



- Rules for each new character
  - Buffers
  - Re-fetch rate

- $m_r\, n_r\, k_c\, m_c$
- $m_0\, n_0\, k_0\, m_1$



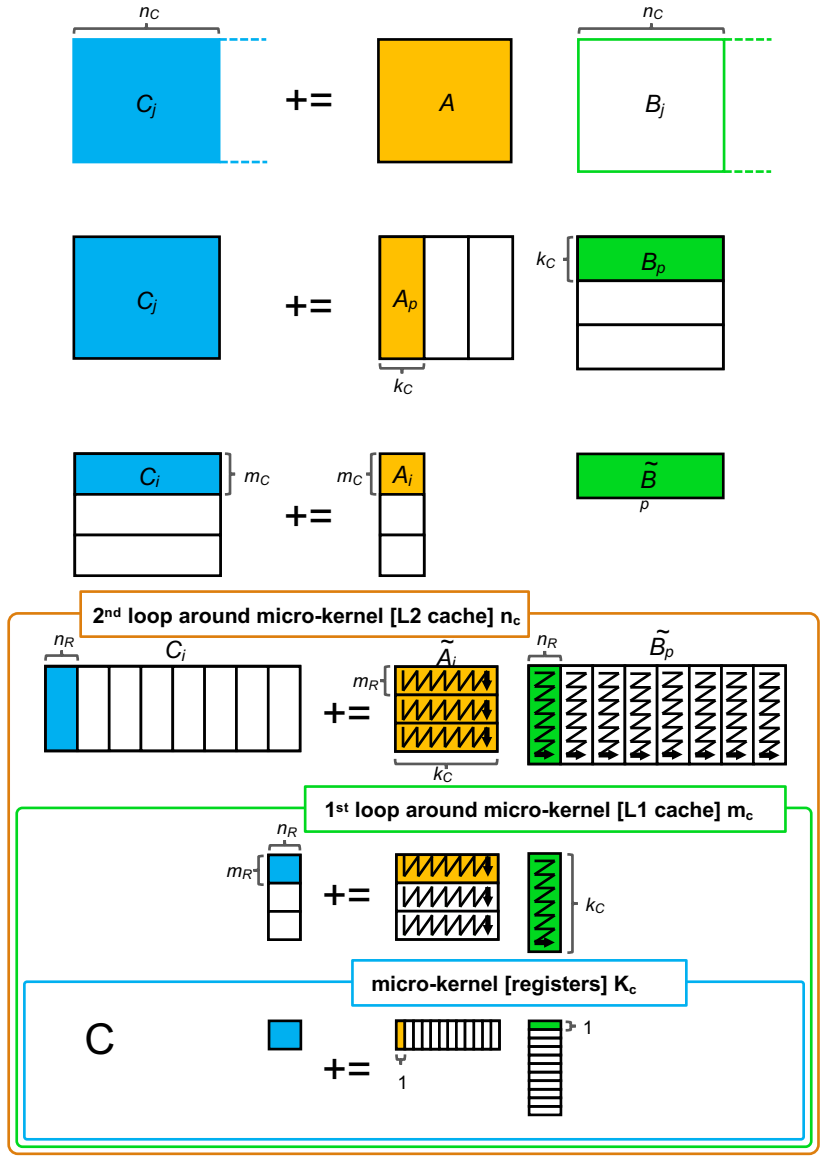


$$
\begin{array}{l}
\textbf{for } j_c = 0, \ldots, n-1 \textbf{ in steps of } n_c \\
\quad \textbf{for } p_c = 0, \ldots, k-1 \textbf{ in steps of } k_c \\
\quad\quad B(p_c : p_c + k_c - 1, j_c : j_c + n_c - 1) \rightarrow B_c \\
\quad\quad \textbf{for } i_c = 0, \ldots, m-1 \textbf{ in steps of } m_c \\
\quad\quad\quad A(i_c : i_c + m_c - 1, p_c : p_c + k_c - 1) \rightarrow A_c \\
\hline
\quad\quad\quad \textbf{for } j_r = 0, \ldots, n_c - 1 \textbf{ in steps of } n_r \\
\quad\quad\quad\quad \textbf{for } i_r = 0, \ldots, m_c - 1 \textbf{ in steps of } m_r \\
\hline
\quad\quad\quad\quad\quad \textbf{for } p_r = 0, \ldots, k_c - 1 \textbf{ in steps of } 1 \\
\quad\quad\quad\quad\quad\quad C_c(i_r : i_r + m_r - 1, j_r : j_r + n_r - 1) \\
\quad\quad\quad\quad\quad\quad\quad += A_c(i_r : i_r + m_r - 1, p_r) \\
\quad\quad\quad\quad\quad\quad\quad \cdot\ B_c(p_r, j_r : j_r + n_r - 1)
\end{array}
$$

- Rules for each new character
  - Buffers
  - Re-fetch rate

- $m_r\, n_r\, k_c\, m_c\, n_c$
- $m_0\, n_0\, k_0\, m_1\, n_1$

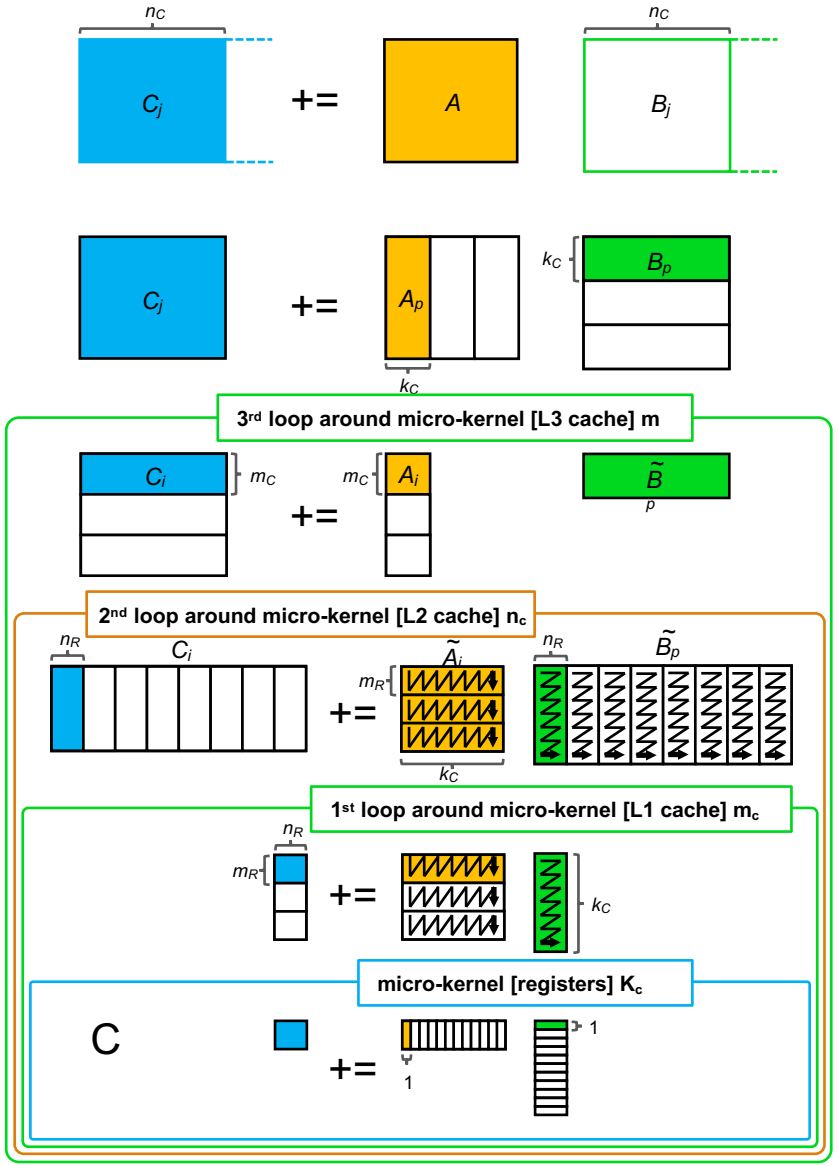
$$
\begin{array}{l}
\textbf{for } j_c = 0, \ldots, n-1 \textbf{ in steps of } n_c \\
\quad \textbf{for } p_c = 0, \ldots, k-1 \textbf{ in steps of } k_c \\
\quad\quad B(p_c : p_c + k_c - 1, j_c : j_c + n_c - 1) \rightarrow B_c \\
\quad\quad \textbf{for } i_c = 0, \ldots, m-1 \textbf{ in steps of } m_c \\
\quad\quad\quad A(i_c : i_c + m_c - 1, p_c : p_c + k_c - 1) \rightarrow A_c \\
\quad\quad\quad \textbf{for } j_r = 0, \ldots, n_c - 1 \textbf{ in steps of } n_r \\
\quad\quad\quad\quad \textbf{for } i_r = 0, \ldots, m_c - 1 \textbf{ in steps of } m_r \\
\hline
\quad\quad\quad\quad\quad \textbf{for } p_r = 0, \ldots, k_c - 1 \textbf{ in steps of } 1 \\
\quad\quad\quad\quad\quad\quad C_c(i_r : i_r + m_r - 1, j_r : j_r + n_r - 1) \\
\quad\quad\quad\quad\quad\quad\quad \mathrel{+}= A_c(i_r : i_r + m_r - 1, p_r) \\
\quad\quad\quad\quad\quad\quad\quad \cdot\ B_c(p_r, j_r : j_r + n_r - 1)
\end{array}
$$



- Rules for each new character
  - Buffers
  - Re-fetch rate

- $m_r\, n_r\, k_c\, m_c\, n_c\, m$
- $m_0\, n_0\, k_0\, m_1\, n_1\, m_2$

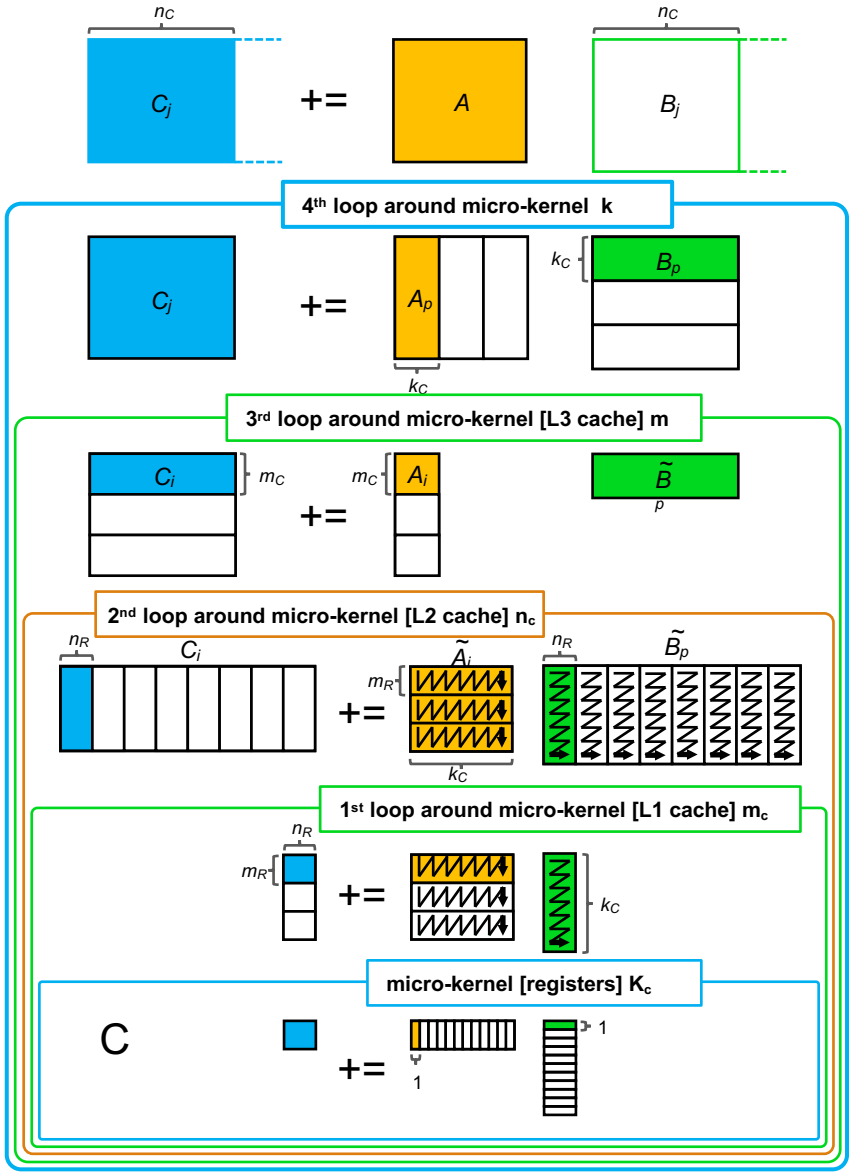
$$
\begin{array}{l}
\textbf{for } j_c = 0, \ldots, n-1 \textbf{ in steps of } n_c \\
\quad \textbf{for } p_c = 0, \ldots, k-1 \textbf{ in steps of } k_c \\
\quad\quad B(p_c : p_c + k_c - 1, j_c : j_c + n_c - 1) \rightarrow B_c \\
\quad\quad \textbf{for } i_c = 0, \ldots, m-1 \textbf{ in steps of } m_c \\
\quad\quad\quad A(i_c : i_c + m_c - 1, p_c : p_c + k_c - 1) \rightarrow A_c \\
\hline
\quad\quad\quad \textbf{for } j_r = 0, \ldots, n_c - 1 \textbf{ in steps of } n_r \\
\quad\quad\quad\quad \textbf{for } i_r = 0, \ldots, m_c - 1 \textbf{ in steps of } m_r \\
\hline
\quad\quad\quad\quad\quad \textbf{for } p_r = 0, \ldots, k_c - 1 \textbf{ in steps of } 1 \\
\quad\quad\quad\quad\quad\quad C_c(i_r : i_r + m_r - 1, j_r : j_r + n_r - 1) \\
\quad\quad\quad\quad\quad\quad\quad \mathrel{+}= A_c(i_r : i_r + m_r - 1, p_r) \\
\quad\quad\quad\quad\quad\quad\quad \cdot\ B_c(p_r, j_r : j_r + n_r - 1)
\end{array}
$$



- Rules for each new character
  - Buffers
  - Re-fetch rate

- $m_r\, n_r\, k_c\, m_c\, n_c\, m$ k
- $m_0\, n_0\, k_0\, m_1\, n_1\, m_2\, k_1$

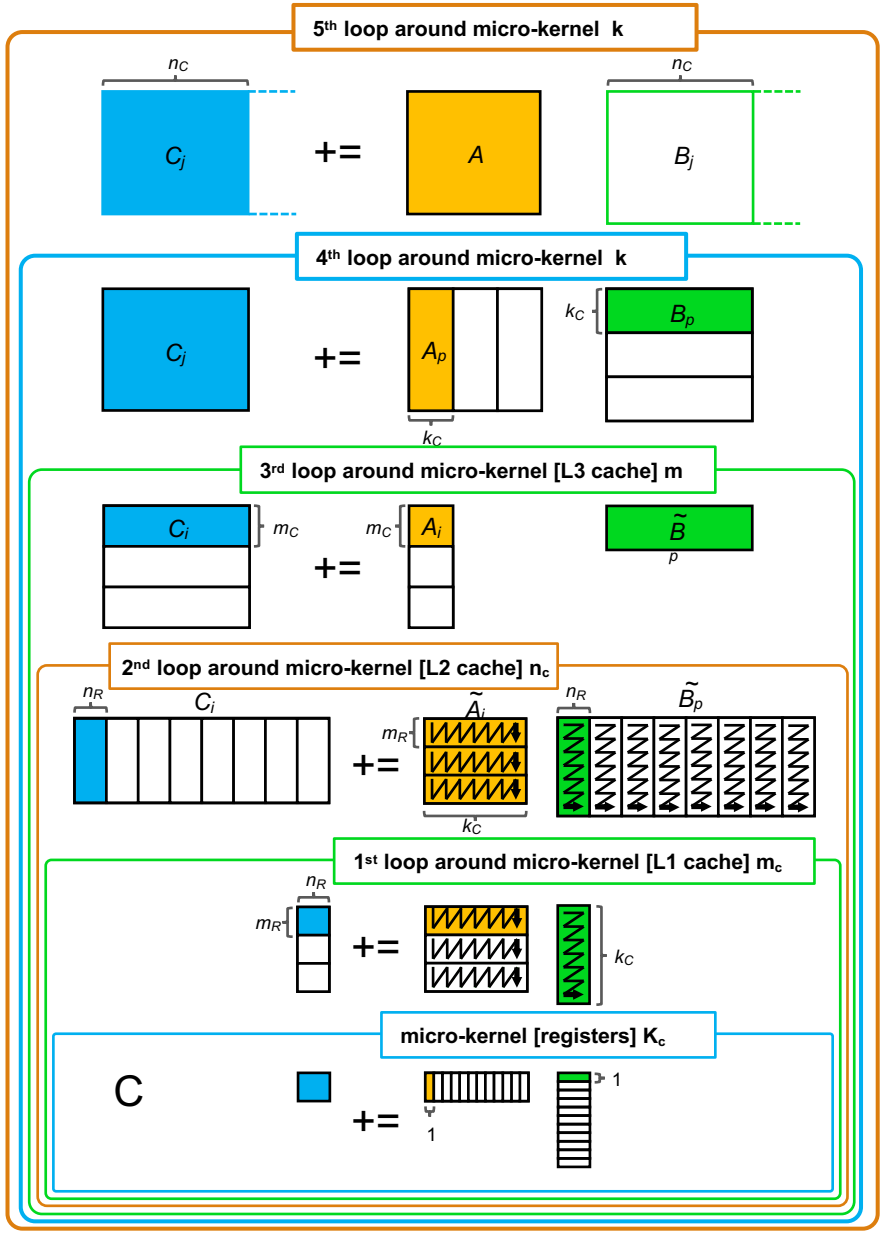
$$
\begin{array}{l}
\textbf{for } j_c = 0, \ldots, n-1 \textbf{ in steps of } n_c \\
\quad \textbf{for } p_c = 0, \ldots, k-1 \textbf{ in steps of } k_c \\
\quad\quad B(p_c : p_c + k_c - 1, j_c : j_c + n_c - 1) \rightarrow B_c \\
\quad\quad \textbf{for } i_c = 0, \ldots, m-1 \textbf{ in steps of } m_c \\
\quad\quad\quad A(i_c : i_c + m_c - 1, p_c : p_c + k_c - 1) \rightarrow A_c \\
\hline
\quad\quad\quad \textbf{for } j_r = 0, \ldots, n_c - 1 \textbf{ in steps of } n_r \\
\quad\quad\quad\quad \textbf{for } i_r = 0, \ldots, m_c - 1 \textbf{ in steps of } m_r \\
\hline
\quad\quad\quad\quad\quad \textbf{for } p_r = 0, \ldots, k_c - 1 \textbf{ in steps of } 1 \\
\quad\quad\quad\quad\quad\quad C_c(i_r : i_r + m_r - 1, j_r : j_r + n_r - 1) \\
\quad\quad\quad\quad\quad\quad\quad += A_c(i_r : i_r + m_r - 1, p_r) \\
\quad\quad\quad\quad\quad\quad\quad\quad \cdot\ B_c(p_r, j_r : j_r + n_r - 1)
\end{array}
$$



- Rules for each new character
  - Buffers
  - Re-fetch rate

- $m_r\ n_r\ k_c\ m_c\ n_c\ m$ k n
- $m_0\ n_0\ k_0\ m_1\ n_1\ m_2\ k_1$ n_2

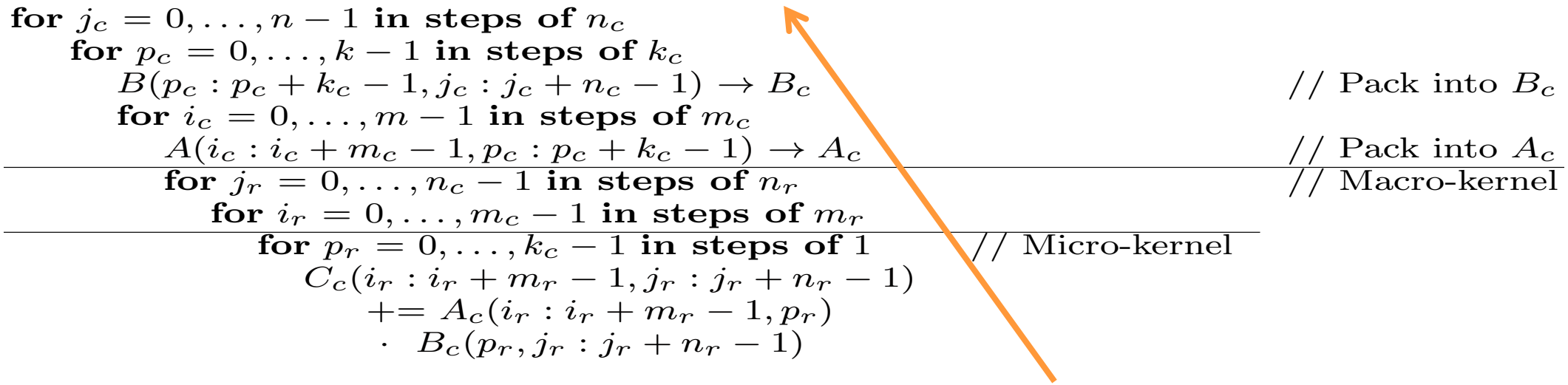
$$
\begin{array}{l}
\textbf{for } j_c = 0, \ldots, n-1 \textbf{ in steps of } n_c \\
\quad \textbf{for } p_c = 0, \ldots, k-1 \textbf{ in steps of } k_c \\
\quad\quad B(p_c : p_c + k_c - 1, j_c : j_c + n_c - 1) \rightarrow B_c \\
\quad\quad \textbf{for } i_c = 0, \ldots, m-1 \textbf{ in steps of } m_c \\
\quad\quad\quad A(i_c : i_c + m_c - 1, p_c : p_c + k_c - 1) \rightarrow A_c \\
\hline
\quad\quad\quad \textbf{for } j_r = 0, \ldots, n_c - 1 \textbf{ in steps of } n_r \\
\quad\quad\quad\quad \textbf{for } i_r = 0, \ldots, m_c - 1 \textbf{ in steps of } m_r \\
\hline
\quad\quad\quad\quad\quad \textbf{for } p_r = 0, \ldots, k_c - 1 \textbf{ in steps of } 1 \\
\quad\quad\quad\quad\quad\quad C_c(i_r : i_r + m_r - 1, j_r : j_r + n_r - 1) \\
\quad\quad\quad\quad\quad\quad\quad += A_c(i_r : i_r + m_r - 1, p_r) \\
\quad\quad\quad\quad\quad\quad\quad\ \cdot\ B_c(p_r, j_r : j_r + n_r - 1)
\end{array}
$$



GEMM Blocking

- Loop representation string
- Rules for each new character
  - Buffers
  - Re-fetch rate

- $m_r\ n_r\ k_c\ m_c\ n_c\ m\ \ k\ \ n$
- $m_0\ n_0\ k_0\ m_1\ n_1\ m_2\ k_1\ \ n_2$



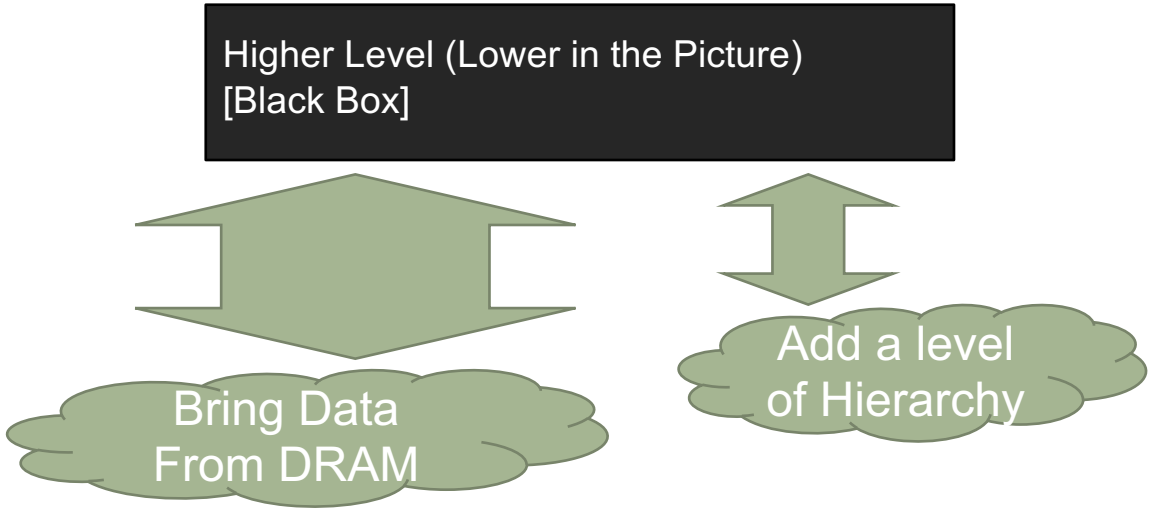
Recursive Formulation

- Fixed order as follows:

$$\_\_\_\_\_ N_{i-1} M_{i-1} K_{i-1} \rightarrow$$

- From left to right observing
  - N: New A

Buffer Type	Buffer Size	Buffer Refetch Rate



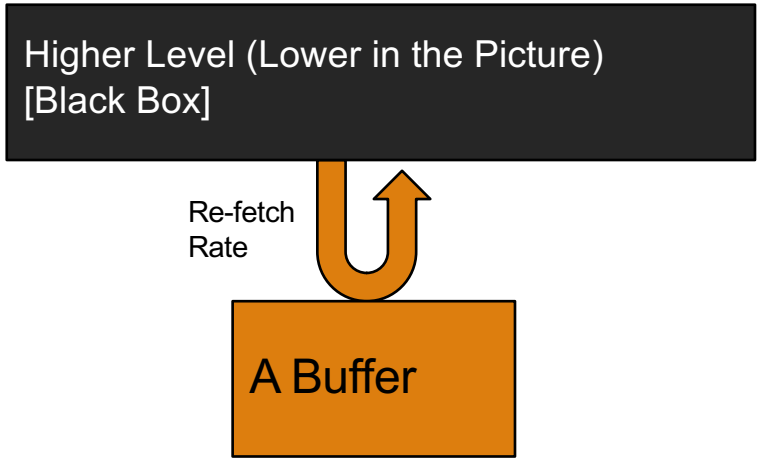
Recursive Formulation

- Fixed order as follows:

$$\_\_\_\_\_ N_{i-1} M_{i-1} K_{i-1} N_i \longrightarrow$$

Buffer Type	Buffer Size	Buffer Refetch Rate
A_i	$M_{i-1}\ K_{i-1}$	N_i / N_{i-1}

- From left to right observing
  - N: New A



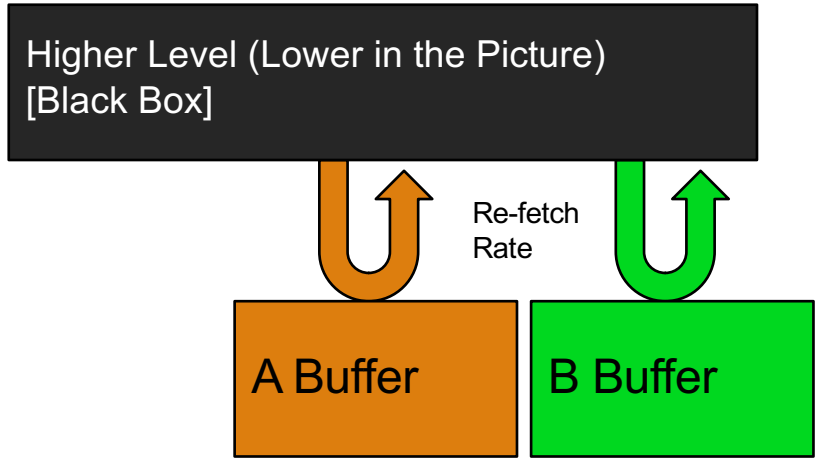
Recursive Formulation

- Fixed order as follows:

$$\_\_\_\_\_ N_{i-1}M_{i-1}K_{i-1}N_iM_i \longrightarrow$$

- From left to right observing
  - N: New A
  - M: New B

Buffer Type	Buffer Size	Buffer Refetch Rate
A_i	$M_{i-1}\ K_{i-1}$	N_i / N_{i-1}
B_i	$N_{i-1}\ K_{i-1}$	M_i / M_{i-1}



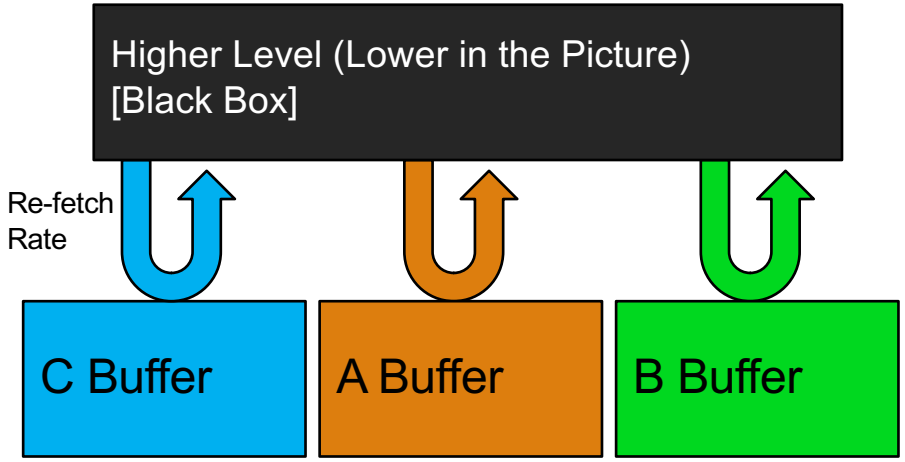
Recursive Formulation

- Fixed order as follows:

$$\_\_\_\_\_ N_{i-1}M_{i-1}K_{i-1}N_iM_iK_i\ldots\ldots$$

Buffer Type	Buffer Size	Buffer Refetch Rate
A_i	$M_{i-1}\,K_{i-1}$	N_i / N_{i-1}
B_i	$N_{i-1}\,K_{i-1}$	M_i / M_{i-1}
C_i	$M_{i-1}\,N_{i-1}$	$2K_i / K_{i-1}$

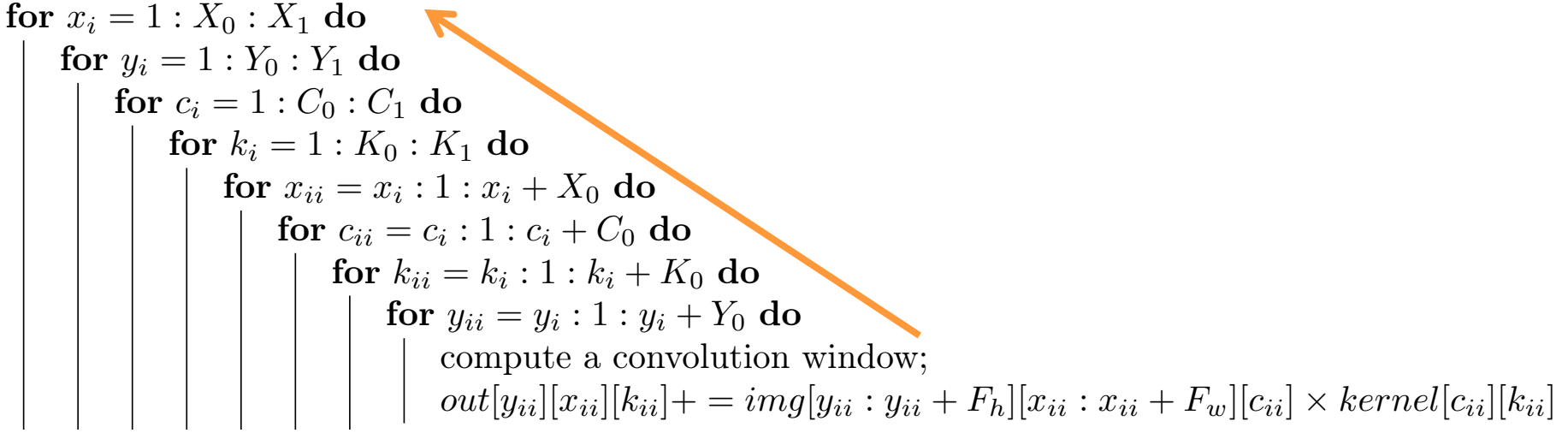
- From left to right observing
  - N: New A
  - M: New B
  - K: New C



Formal Derivation

- Loop representation string
- Rules for each new character
  - Buffers
  - Re-fetch rate

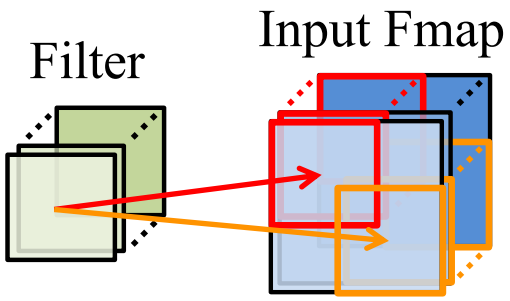
- $Y_0K_0C_0X_0K_1C_1Y_1X_1$:



Types of Data Reuse in DNN

Convolutional Reuse

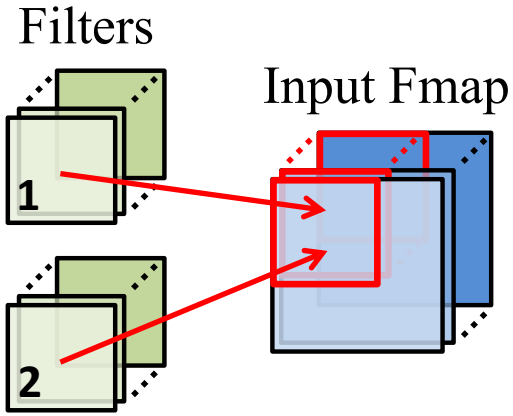
CONV layers only
(sliding window)



Reuse: Activations
Filter weights

Fmap Reuse

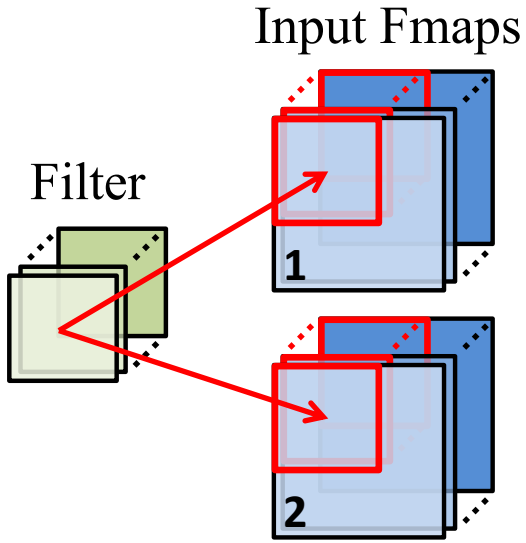
CONV and FC layers



Reuse: Activations

Filter Reuse

CONV and FC layers
(batch size > 1)



Reuse: Filter weights

Source: Yu-Hsin Chen et. al.

Dataflow Taxonomy Summary

- **Weight Stationary**
  - minimize movement of **weights**
- **Output Stationary**
  - minimize movement of partial sums
- **Input Stationary**
  - minimize movement of input activations
- **No Local Reuse**
  - don't use any local PE storage. Maximize **global buffer size**

Source: Yu-Hsin Chen et. al.

Dataflow Taxonomy Summary

- **Weight Stationary**
  - minimize movement of **weights**
- **Output Stationary**
  - minimize movement of partial sums
- **Input Stationary**
  - minimize movement of input activations
- **No Local Reuse**
  - don't use any local PE storage. Maximize **global buffer size**

Does this define the space of all possible dataflows?

Source: Yu-Hsin Chen et. al.

Dataflow Taxonomy Summary

- **Weight Stationary**
  - minimize movement of **weights**
- **Output Stationary**
  - minimize movement of partial sums
- **Input Stationary**
  - minimize movement of input activations
- **No Local Reuse**
  - don't use any local PE storage. Maximize **global buffer size**

Does this define the space of all possible dataflows?

Not at all!

Yes!
If applied
Recursively

Can we have multiple ***stationariness*** in an architecture?

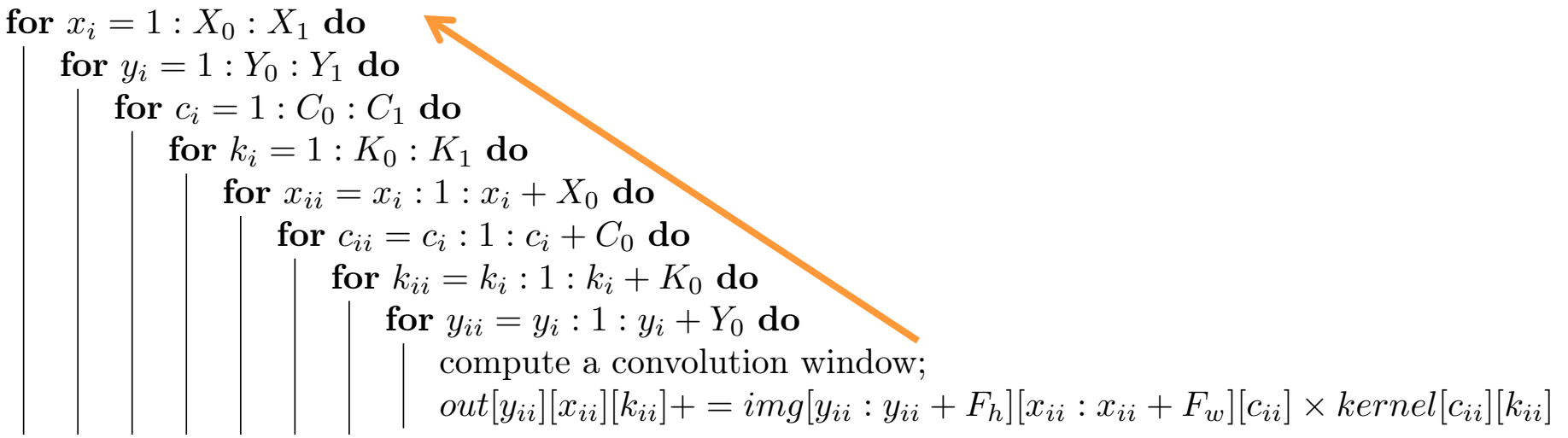
Yes!
For each layer of
Memory Hierarchy

Source: Yu-Hsin Chen et. al.

Formal Derivation

- Loop representation string
- Rules for each new character
  - Buffers
  - Re-fetch rate

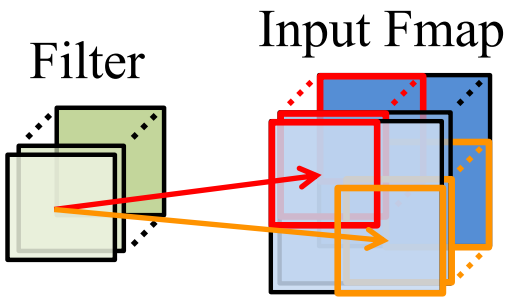
- $Y_0K_0C_0X_0K_1C_1Y_1X_1$:



Types of Data Reuse in DNN

Convolutional Reuse

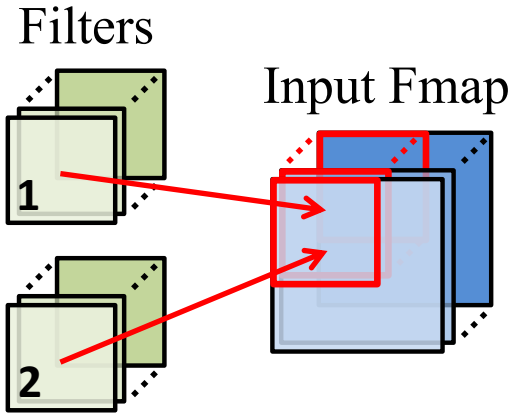
CONV layers only
(sliding window)



Reuse: Activations
Filter weights

Fmap Reuse

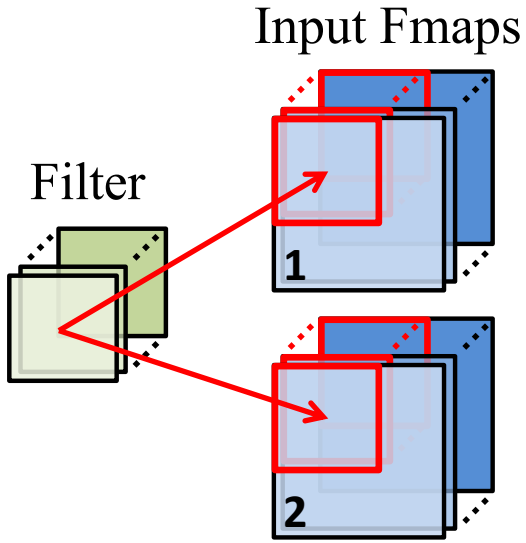
CONV and FC layers



Reuse: Activations

Filter Reuse

CONV and FC layers
(batch size > 1)



Reuse: Filter weights

Source: Yu-Hsin Chen et. al.

Recursive Formulation

- Fixed order as follows:

$$\_\_\_\_\_ Y_{i-1} X_{i-1} C_{i-1} K_{i-1} \cdots\cdots \rightarrow$$

- From left to right observing

Higher Level
[Black Box]

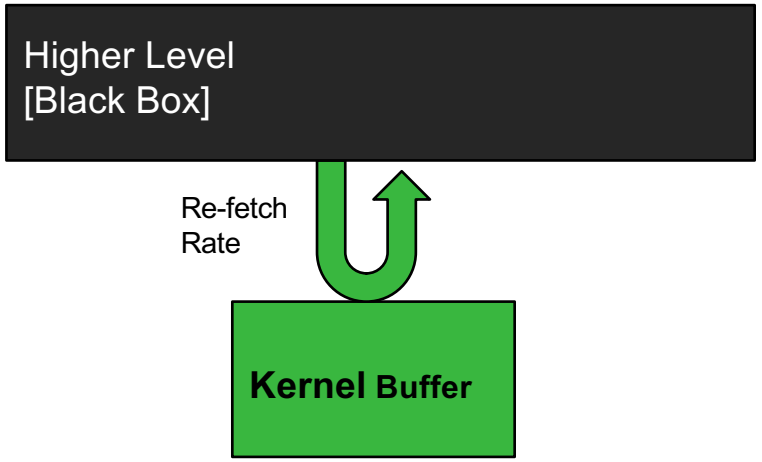
Recursive Formulation

Buffer Name	Buffer Size	Buffer Refetch Rate
KB_i	$C_{i-1}K_{i-1}F_hF_w$	$(X_iY_i)/(X_{i-1}Y_{i-1})$

- Fixed order as follows:

$$\_\_\_\_\_ Y_{i-1}X_{i-1}C_{i-1}K_{i-1}Y_iX_i\cdots\cdots \rightarrow$$

- From left to right observing
  - X or Y: New KB



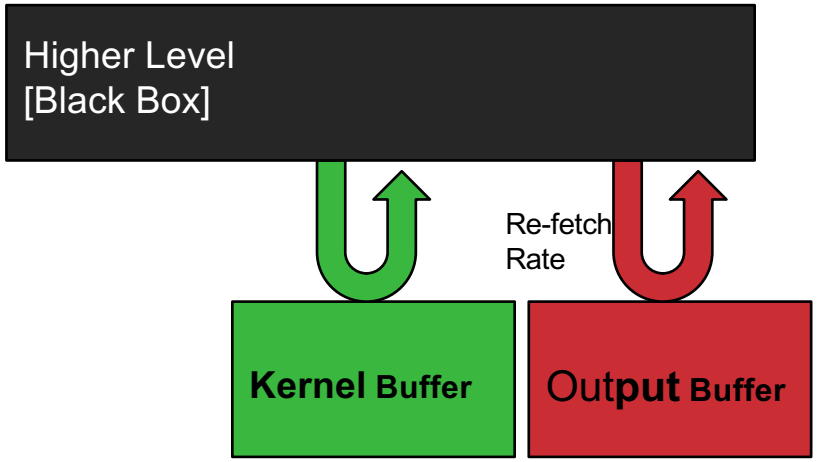
Recursive Formulation

Buffer Name	Buffer Size	Buffer Refetch Rate
KB_i	$C_{i-1}K_{i-1}F_hF_w$	$(X_iY_i)/(X_{i-1}Y_{i-1})$
OB_i	$Y_iX_iK_{i-1}$	$2C_i/C_{i-1}$

- Fixed order as follows:

$$\_\_\_\_\_ Y_{i-1}X_{i-1}C_{i-1}K_{i-1}Y_iX_iC_i\ldots\ldots$$

- From left to right observing
  - X or Y: New KB
  - C: New OB



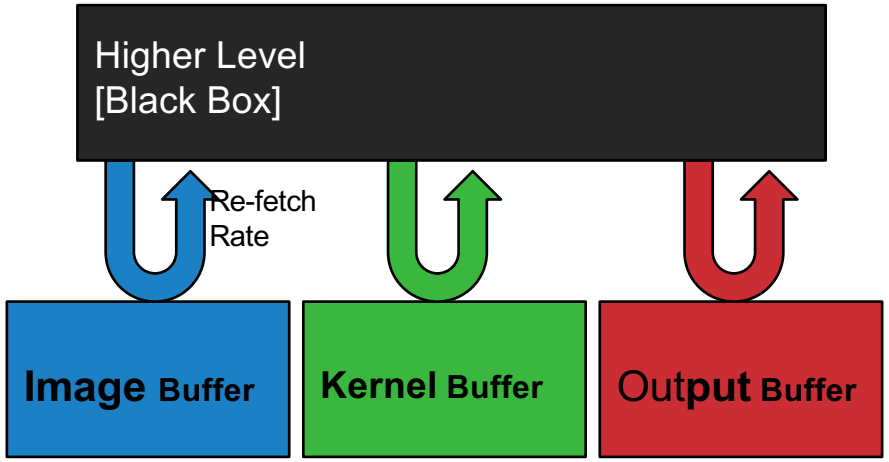
Recursive Formulation

Buffer Name	Buffer Size	Buffer Refetch Rate
KB_i	$C_{i-1}K_{i-1}F_hF_w$	$(X_iY_i)/(X_{i-1}Y_{i-1})$
OB_i	$Y_iX_iK_{i-1}$	$2C_i/C_{i-1}$
IB_i	$(Y_i+F_h-1)(X_i+F_w-1)C_i$	$(K_i(Y_i+F_h-1)(X_i+F_w-1))/(K_{i-1}Y_iX_i)$

- Fixed order as follows:

$$\_\_\_\_\_ Y_{i-1}X_{i-1}C_{i-1}K_{i-1}Y_iX_iC_iK_i\ldots\ldots$$

- From left to right observing
  - X or Y: New KB
  - C: New OB
  - K: New IB



How Does Parallelism Work?

- Unroll Loops in Space
- Replicate Compute Units
- Perform Loops in parallel
  - SIMD
  - Multiple Cores
  - Multiple CPUs/GPUs

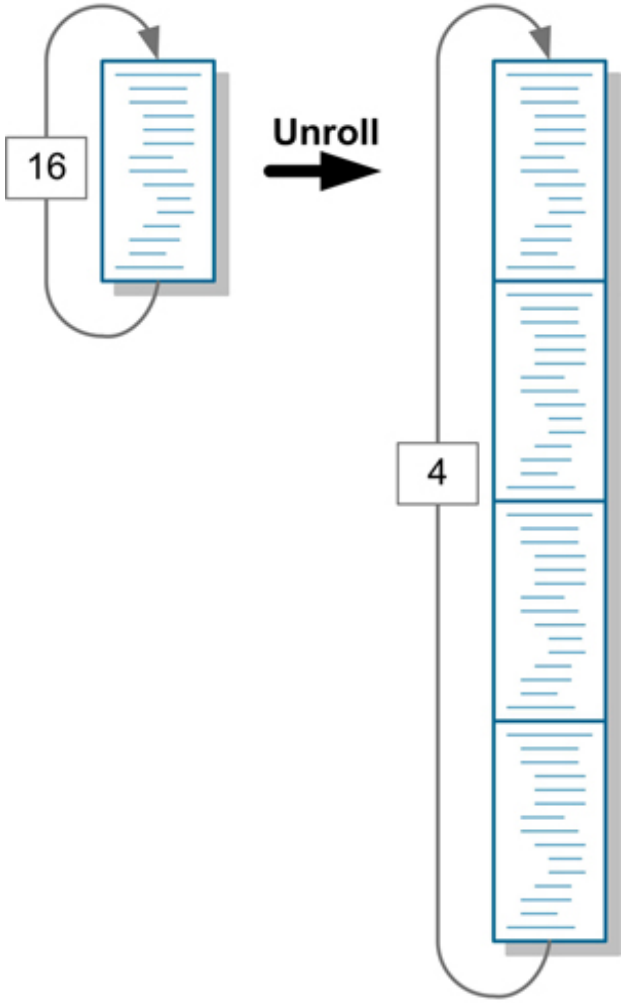
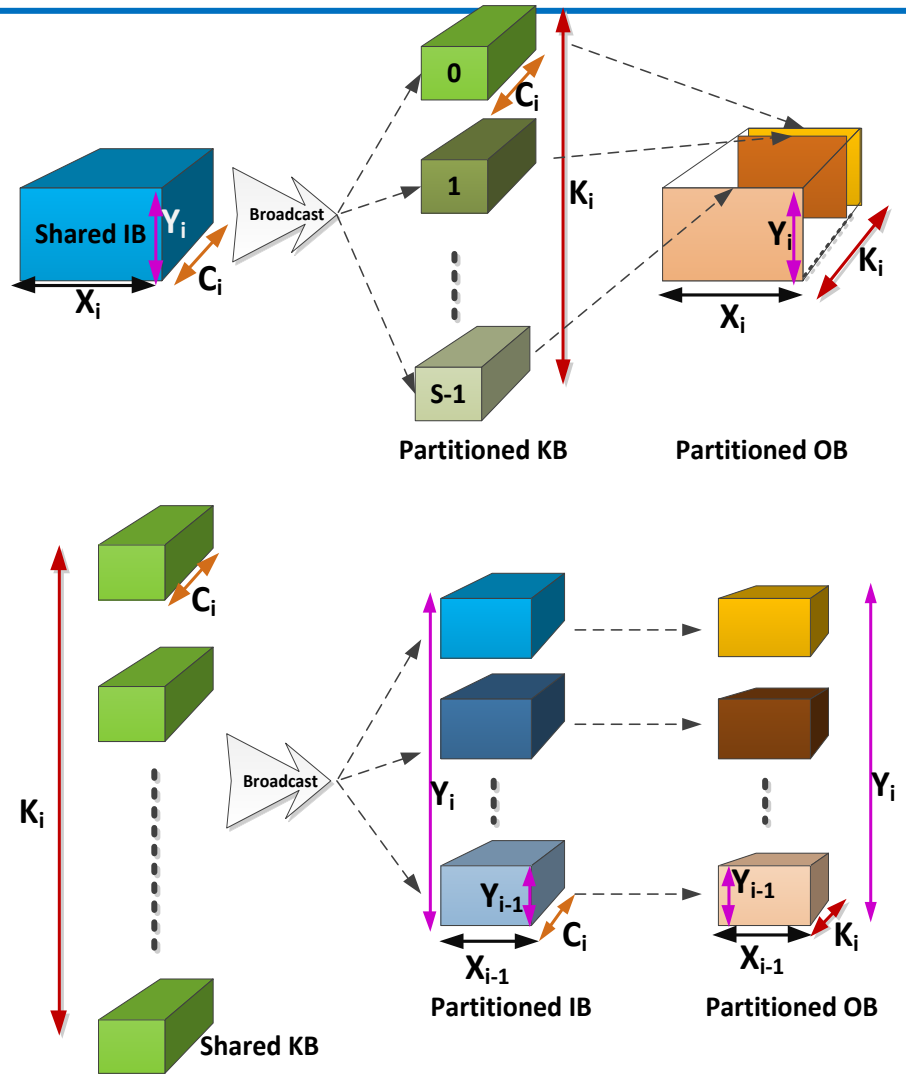


Figure 5. Loop unrolled four times.

Multi-Core Parallelism and Partitioning

- Across channels (C)
  - Reduction
  - Synchronization
- Across kernels (K)
  - Synchronization
- Across image (X or Y)
  - Independent

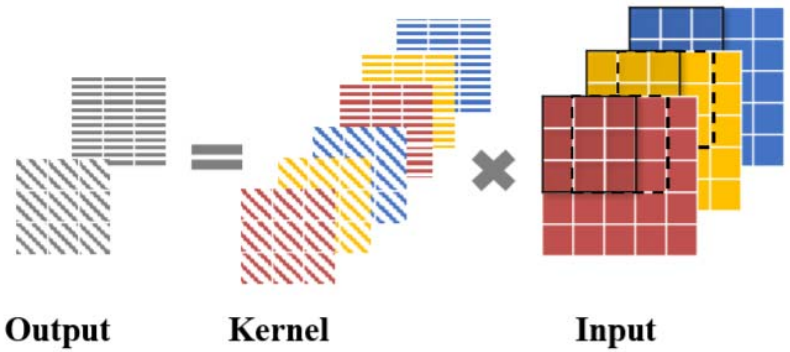
- Broadcast operation
- Partitioned memory
- Nested parallelism



High Performance Direct Convolution

Map application to system architecture to fully exploit system capacity

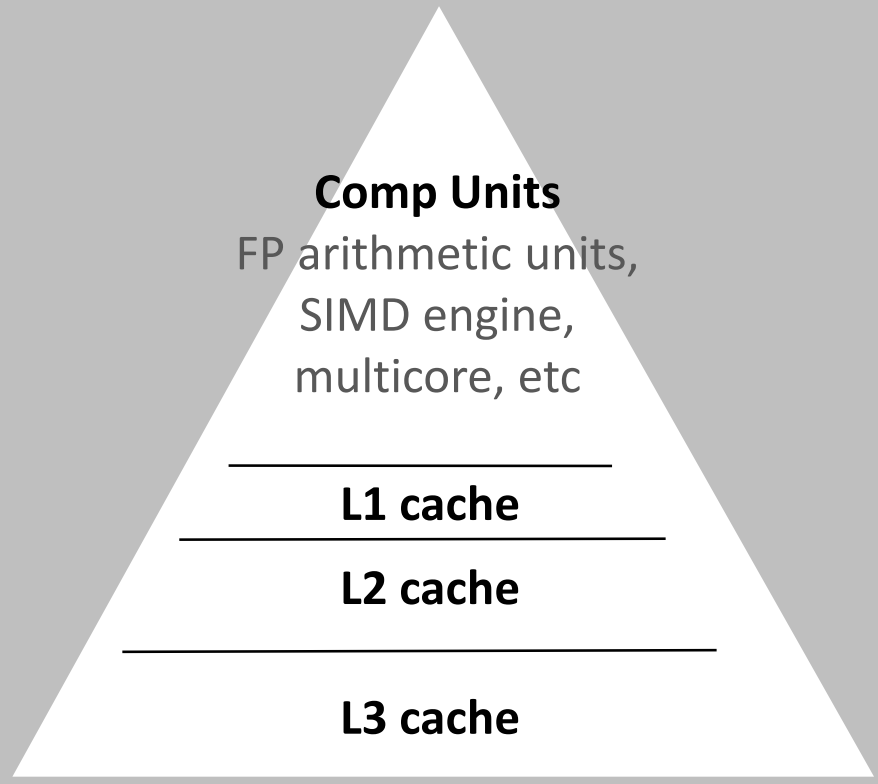
High-dimensional data tensor



Multi-level nested loops and various data access patterns

```
for i = 1 to C_i do
  for j = 1 to C_o do
    for k = 1 to W_o do
      for ℓ = 1 to H_o do
        for m = 1 to W_f do
          for n = 1 to H_f do
            O_{j,k,ℓ} += I_{i,k×s+m,ℓ×s+n} × F_{i,j,m,n}
```

Direct Convolution



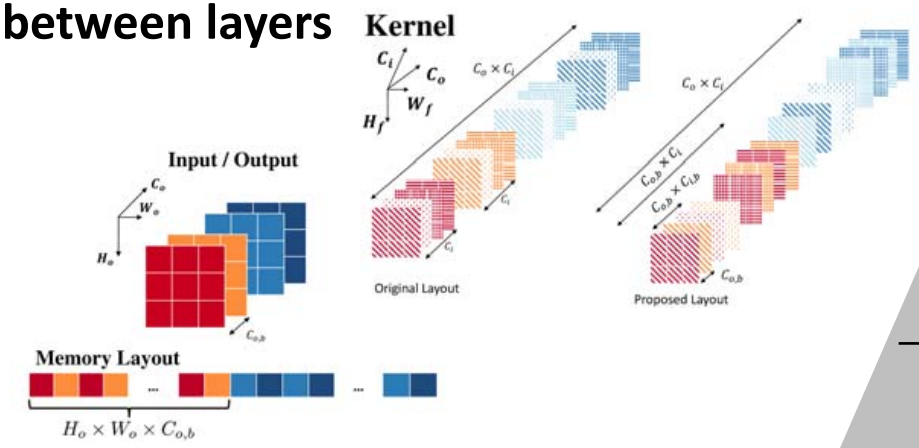
System Architecture

Source: J. Zhang et. al

High Performance Direct Convolution

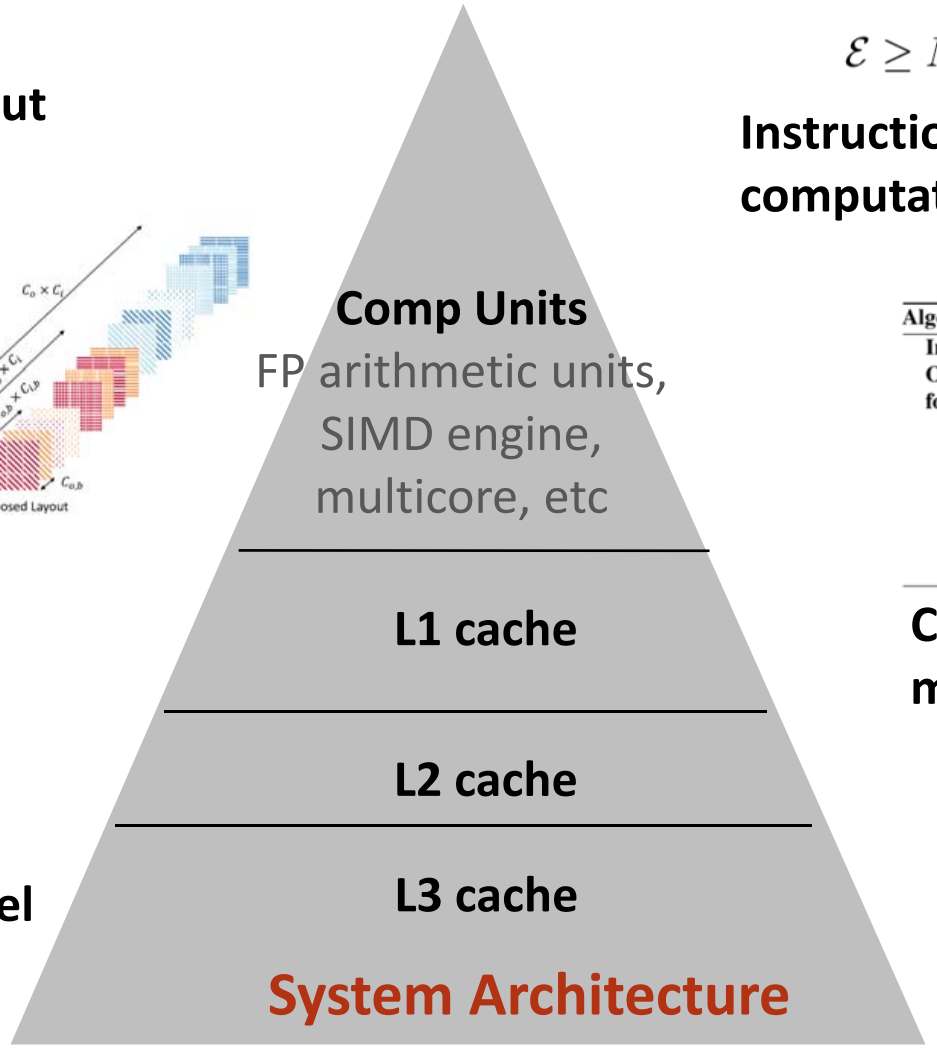
Analytical model to achieve system theoretical peak performance

Convolution-friendly data layout that avoids format reshaping between layers



$$\text{for } j' = 1 \text{ to } C_o/C_{o,b} \text{ in Parallel do}$$
$$\quad \text{for } i' = 1 \text{ to } C_i/C_{i,b} \text{ do}$$

Parallelism to exploit core-level concurrency



$$\mathcal{E} \geq N_{\text{vec}} N_{\text{fma}} L_{\text{fma}},$$

Instruction scheduling to saturate the computation pipeline

Algorithm 2 Reorder Convolution Algorithm

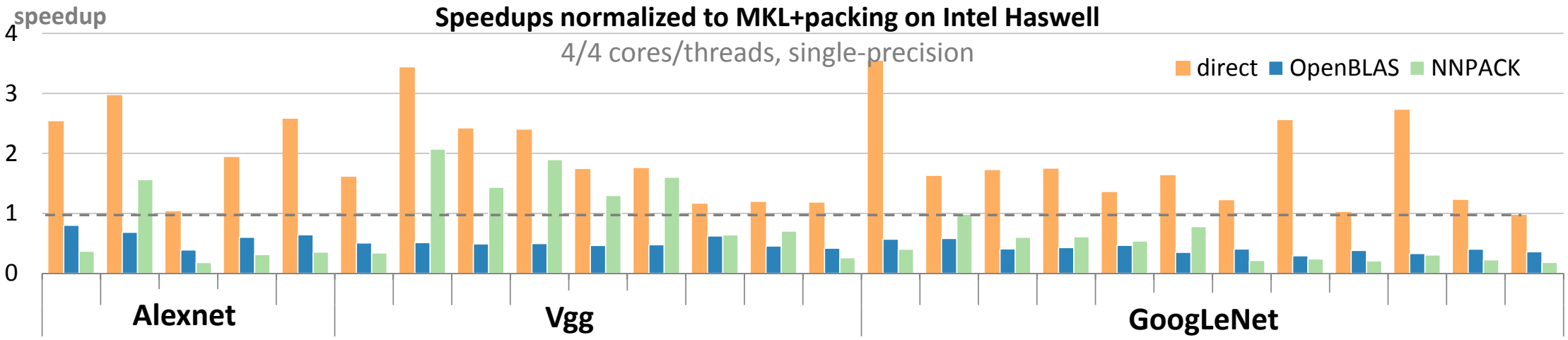
Input: Input $\mathcal{I}$, Kernel Weights $\mathcal{F}$, stride s;
Output: Output $\mathcal{O}$
for $\ell = 1$ **to** H_o **do**
  for $n = 1$ **to** H_f **do**
    for $m = 1$ **to** W_f **do**
      for $i = 1$ **to** C_i **do**
        for $k = 1$ **to** W_o **do**
          for $j = 1$ **to** C_o **do**
            $\mathcal{O}_{j,k,\ell} += \mathcal{I}_{i,k\times s+m,\ell\times s+n} \times \mathcal{F}_{i,j,m,n}$

Computation reordering that maximizes data reuse

Loop tiling and blocking to utilize the memory hierarchies

Source: J. Zhang et. al

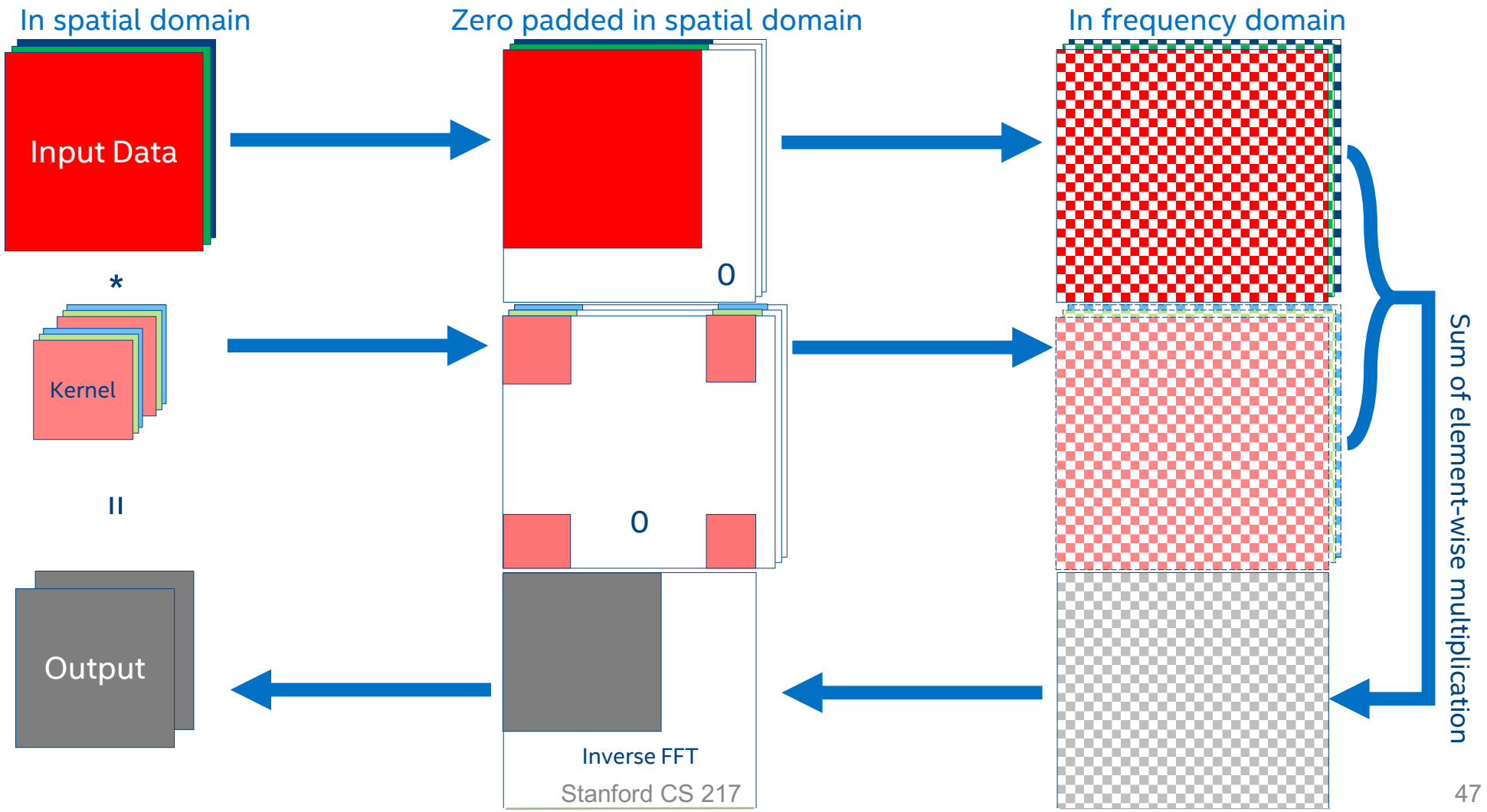
Better performance



Source: J. Zhang et. al

FFT based convolution

Convert input into Fourier Domain, apply element-wise multiplication to reduce complexity: $O(N^2K^2) \rightarrow O(N^2\log_2 N)$, where N is data size, K is kernel size



https://www.slideshare.net/IntelSoftware/clcaffe-unleashing-the-power-of-intel-graphics-for-deep-learning-acceleration

Winograd Method

- Fast algorithm for computing short convolutions

1. Polynomials
   a) Chinese remainder theorem
   b) Lagrange Interpolation
2. Toom-Cook Algorithm
3. Winograd method for FIR filters
4. Gray and Levin paper on CNNs for Winograd

https://artofproblemsolving.com/community/c1157h990758_the_chinese_remainder_theorem_and_lagrange_interpolation

Interpolation

- To find the value of *y* for an *x* between different x - values $x_0, x_1, \ldots, x_n$ is called problem of interpolation.
- To find the value of *y* for an *x* which falls outside the range of *x* ($x < x_0$ *or* $x > x_n$) is called the problem of extrapolation.
- Theorem by Weierstrass in 1885, “Every continuous function in an interval (a,b) can be represented in that interval to any desired accuracy by a polynomial. ”
- Let us assign polynomial P_n of degree *n* (or less) that assumes the given data values
- $P_n(x_0) = y_0, P_n(x_1) = y_1, \ldots, P_n(x_n) = y_n$
  This polynomial Pn is called interpolation polynomial.
- $x_0, x_1, \ldots, x_n$ is called the nodes (tabular points, pivotal points or arguments).

Lagrange Interpolation

- Lagrange's interpolation formula with unequal intervals:
- Let $y = f(x)$ be continuous and differentiable in the interval (a, b).
- Given the set of n+1 values $(x_0,y_0),(x_1,y_1),\ldots,(x_n,y_n)$ of x and y, where the values of x need not necessarily be equally spaced.
- It is required to find $P_n(x)$, a polynomial of degree n such that y and $P_n(x)$ agree at the tabulated points.

Lagrange Interpolation

- This polynomial is given by the following formula:

$$y = f(x) \approx P_n(x) = \frac{(x - x_1)(x - x_2)\ldots(x - x_n)}{(x_0 - x_1)(x_0 - x_2)\ldots(x_0 - x_n)} y_0$$
$$+ \frac{(x - x_0)(x - x_2)\ldots(x - x_n)}{(x_1 - x_0)(x_1 - x_2)\ldots(x_1 - x_n)} y_1 + \ldots$$
$$+ \frac{(x - x_0)(x - x_1)\ldots(x - x_{n-1})}{(x_n - x_0)(x_n - x_1)\ldots(x_n - x_{n-1})} y_n$$

- In the Lagrange's interpolation formula *y* is treated as dependent variable and expressed as function of independent variable *x*.
- Instead if *x* is treated as dependent variable and expressed as the function of independent variable *y*, then Lagrange's interpolation formula becomes

$$x = g(y) \approx P_n(y) = \frac{(y-y_1)(y-y_2)\dots(y-y_n)}{(y_0-y_1)(y_0-y_2)\dots(y_0-y_n)}x_0$$
$$+\frac{(y-y_0)(y-y_2)\dots(y-y_n)}{(y_1-y_0)(y_1-y_2)\dots(y_1-y_n)}x_1+\dots$$
$$+\frac{(y-y_0)(y-y_1)\dots(y-y_{n-1})}{(y_n-y_0)(y_n-y_1)\dots(y_n-y_{n-1})}x_n$$

- This relation is referred as Lagrange's inverse interpolation formula.

Chinese Remainder Theorem

- It is possible to uniquely determine a nonnegative integer given only its moduli with respect to each of several integers, provided that the integer is known to be smaller than the product of the moduli.

CRT Example

- Choose the moduli m_0=3, m_1=4,and m_2=5,and let $M = m_0 m_1 m_2 = 60$.
- Given the integer c satisfying $0 \le c < 60$, let $c_i = Rm_i[c]$.
- The Chinese remainder theorem says that there is a one-to-one map between the sixty values that c is allowed to take on and the sixty values that the vector of residues (c_0, c_1, c_2) can take on.
- Suppose that $c_0 = 2$, $c_1 = 1$, and $c_2 = 2$. The Unique Solution is ..

$$c \in \{2, 5, 8, 11, 14, 17, 20, 23, 26, 29, \ldots\},$$

$$c \in \{1, 5, 9, 13, 17, 21, 25, 29, 33, \ldots\},$$

$$c \in \{2, 7, 12, 17, 22, 27, 32, 37, \ldots\}.$$

Relation between CRT and Lagrange

- Chinese Remaining Theorem for polynomials is related to Lagrange interpolation
- Lagrange Interpolation is used in Toom-Cook Fast Convolution
- CRT is used in Winograd Fast Convolution
- Read More about the relation:

Saints and Scoundrels and Two Theorems That Are Really the Same by Ezra Brown