

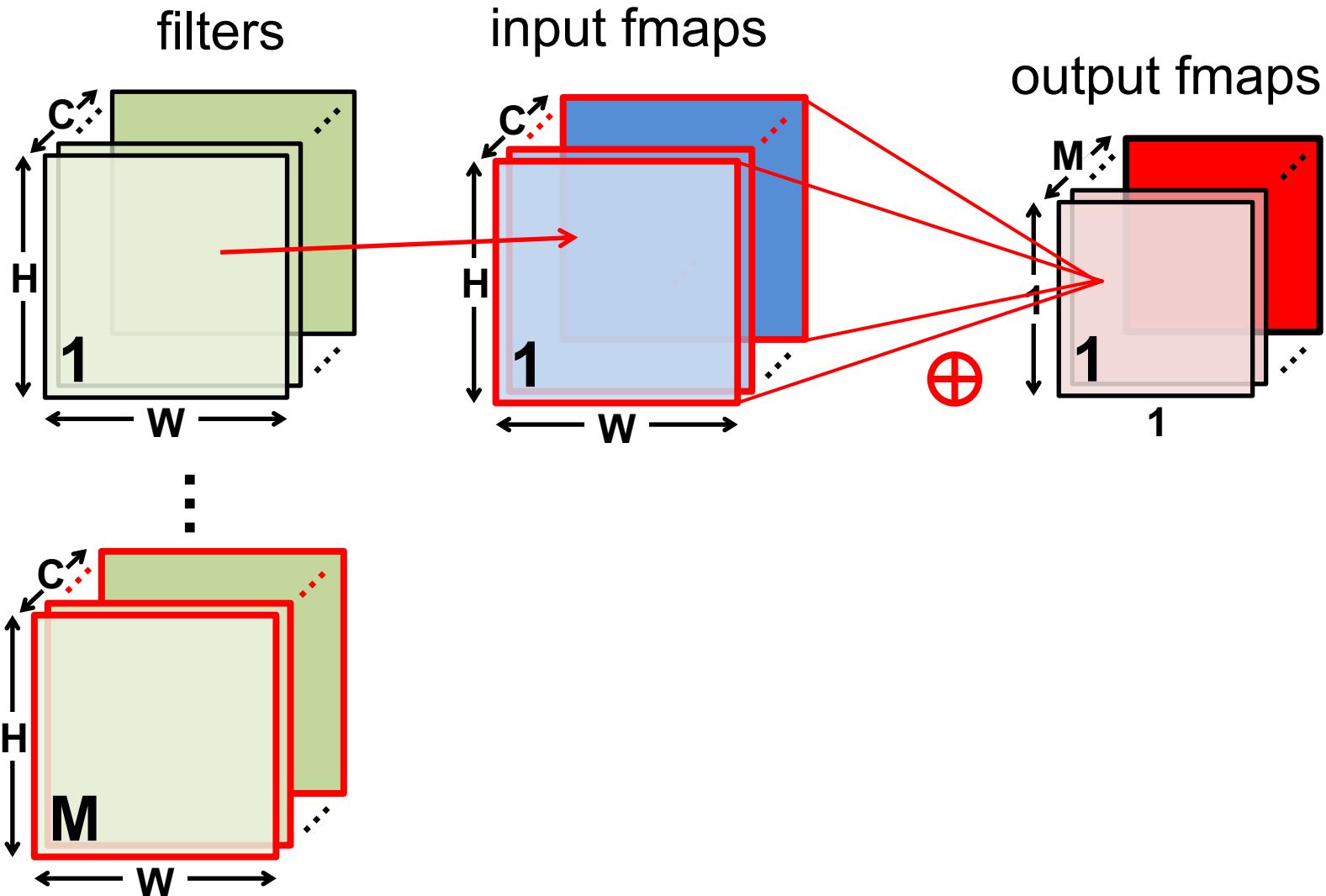
Kernel Computation

ISCA Tutorial (2019)

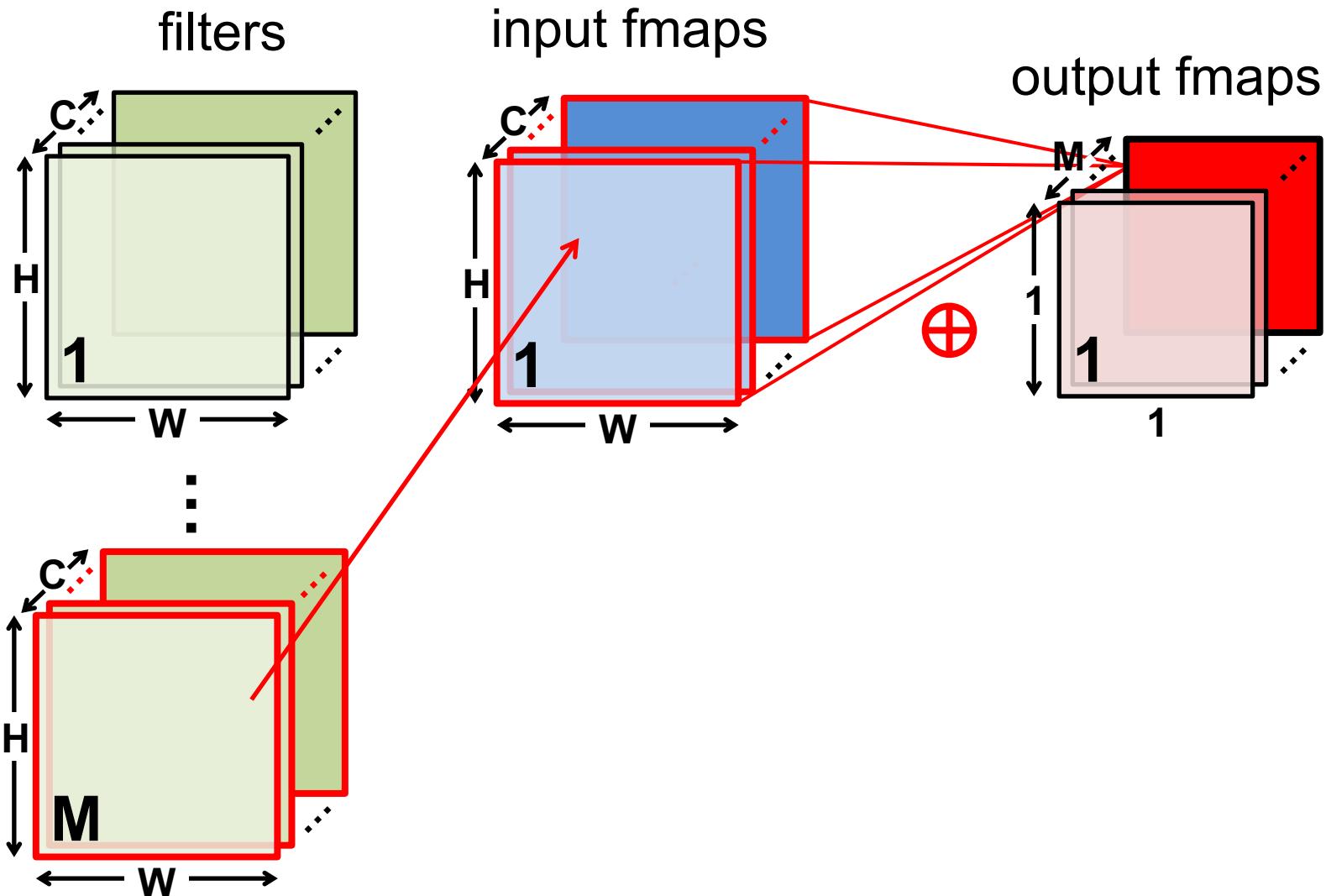
Website: <http://eyeriss.mit.edu/tutorial.html>

Joel Emer, Vivienne Sze, Yu-Hsin Chen

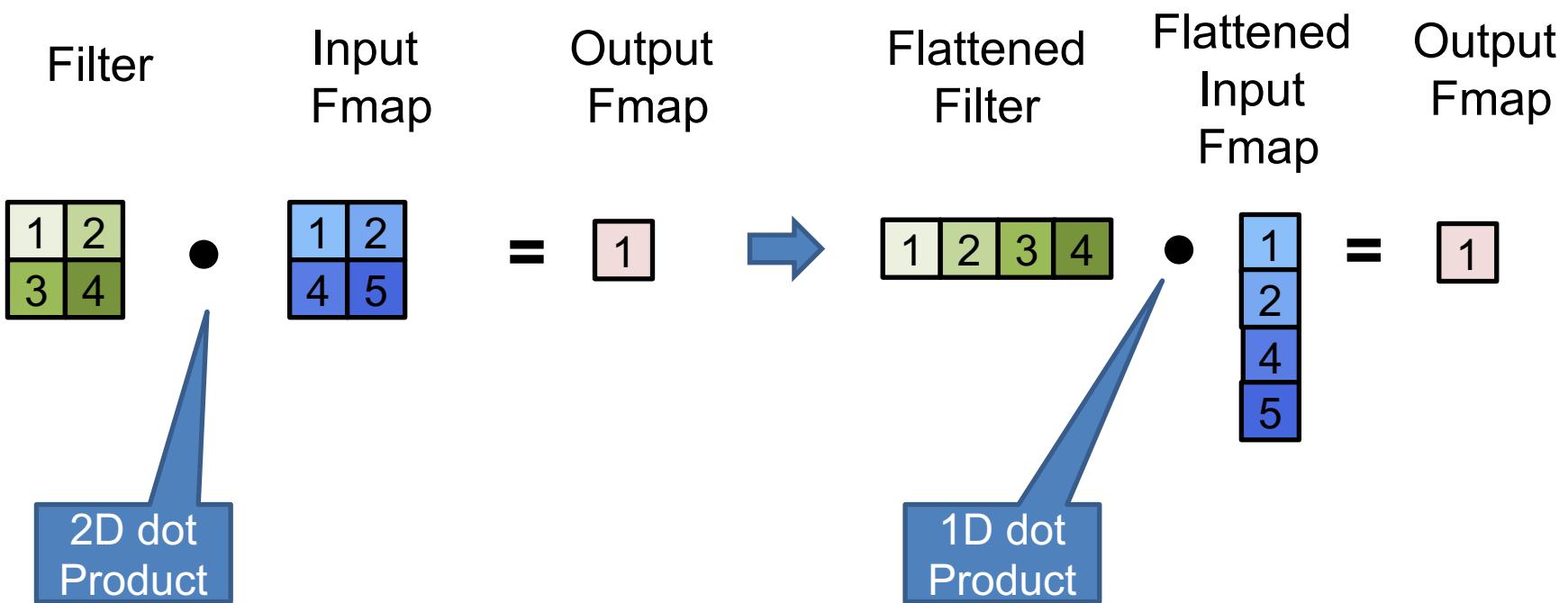
Fully-Connected (FC) Layer



Fully-Connected (FC) Layer

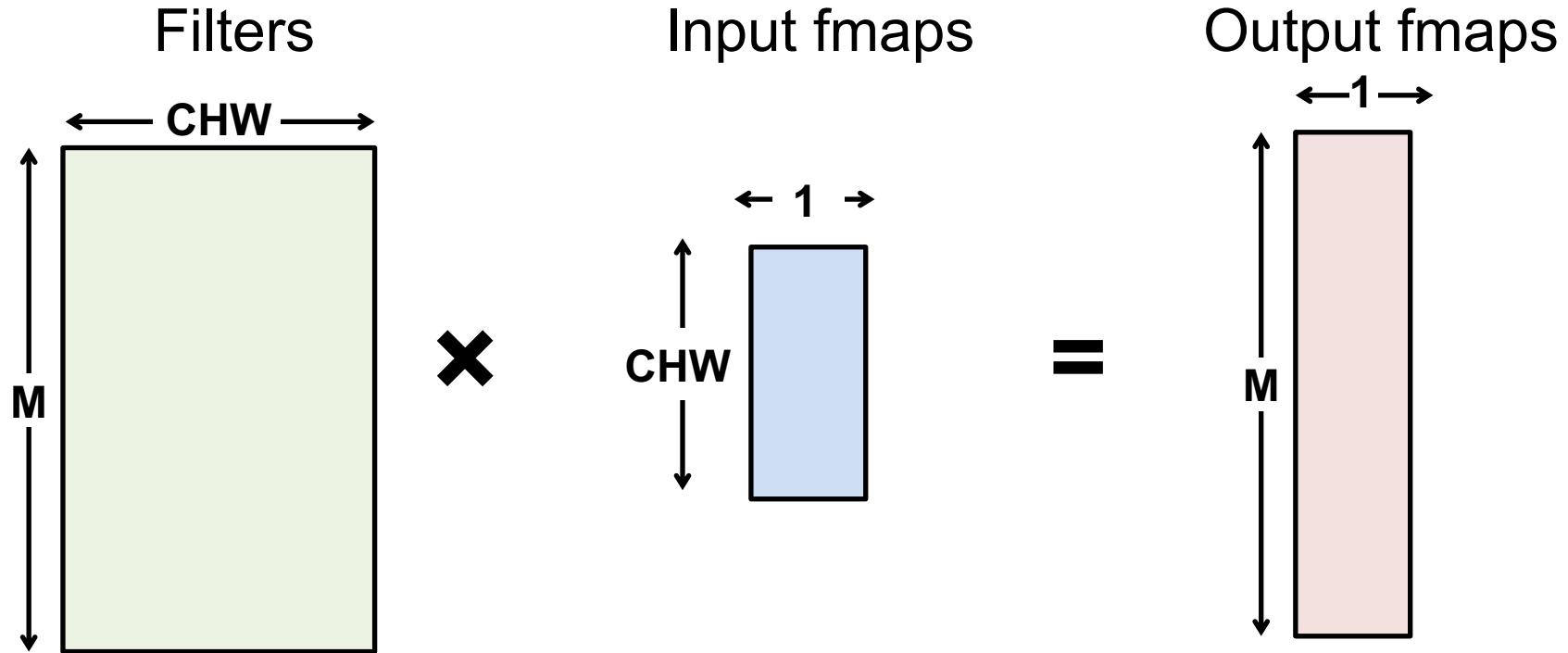


Flattened 2D Dot Product



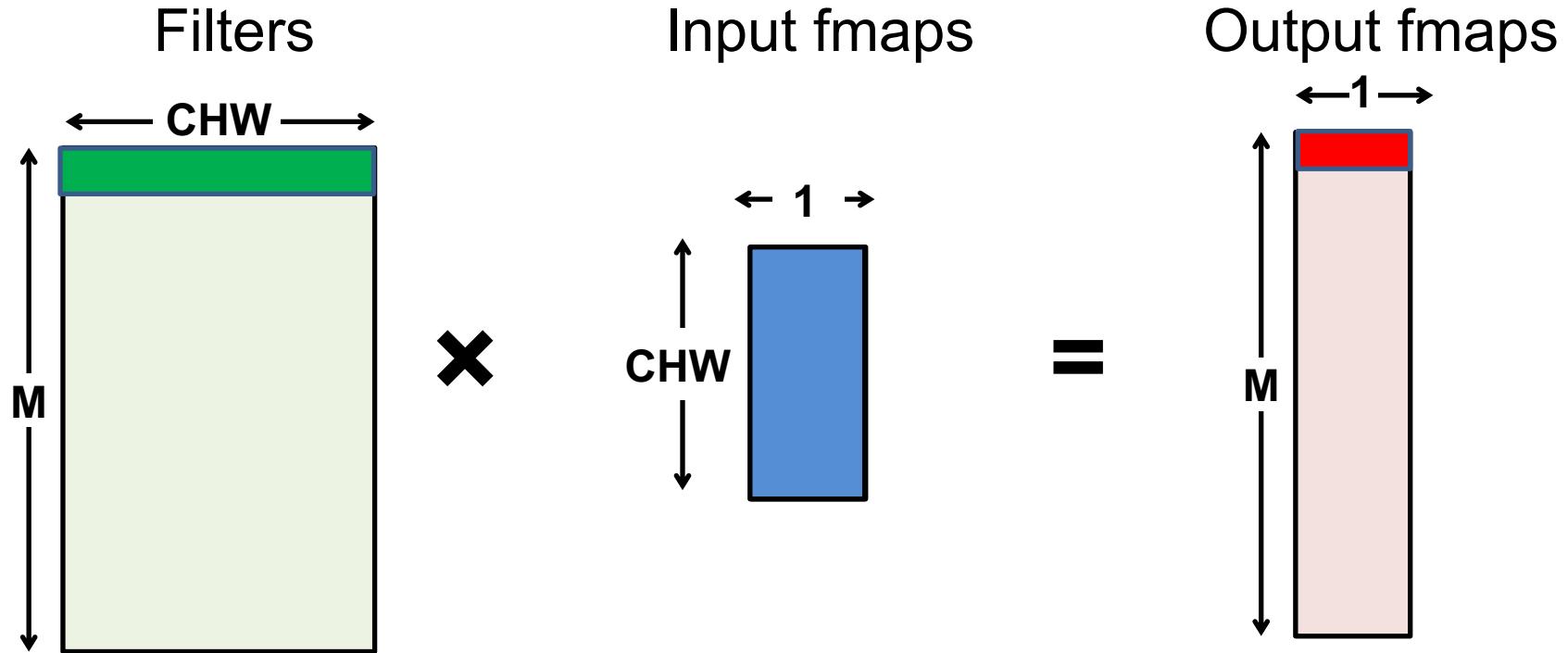
Fully-Connected (FC) Layer

- Matrix–Vector Multiply:
 - Multiply all inputs in all channels by a weight and sum



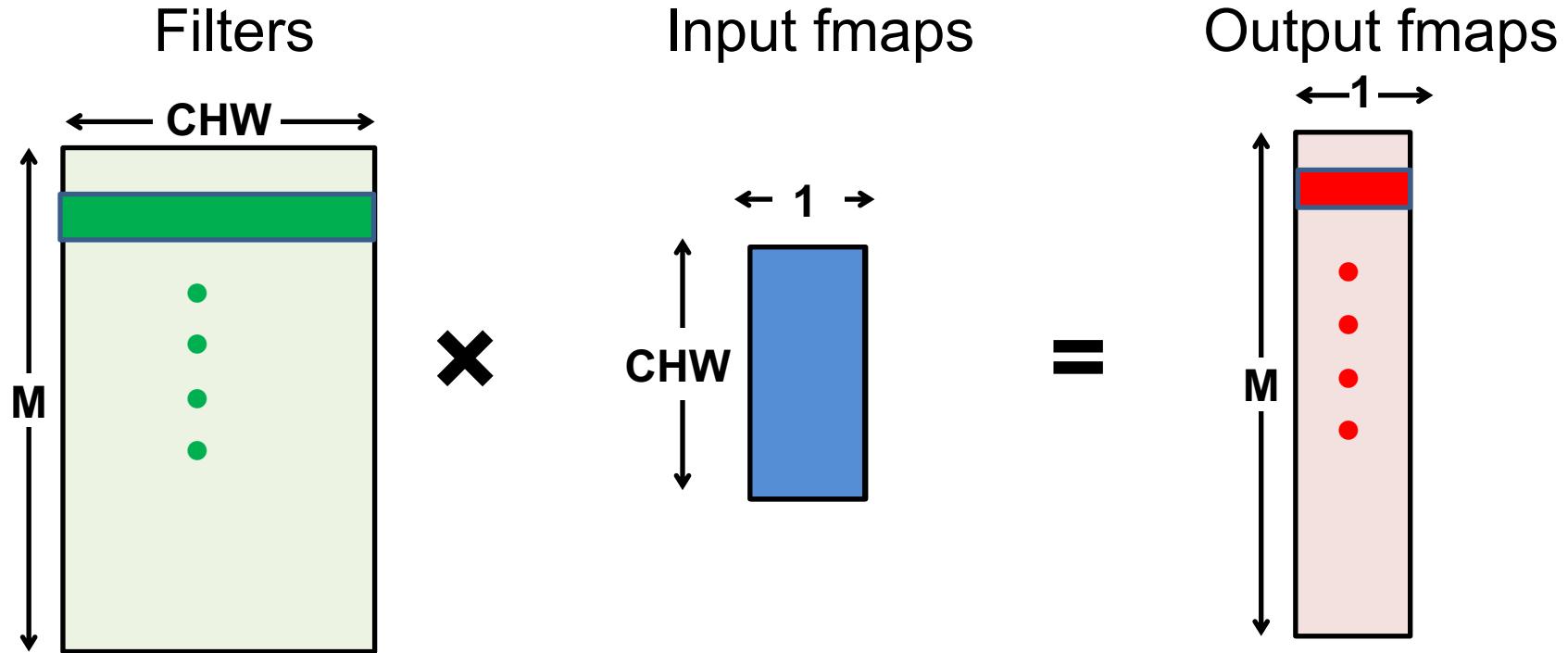
Fully-Connected (FC) Layer

- Matrix–Vector Multiply:
 - Multiply all inputs in all channels by a weight and sum

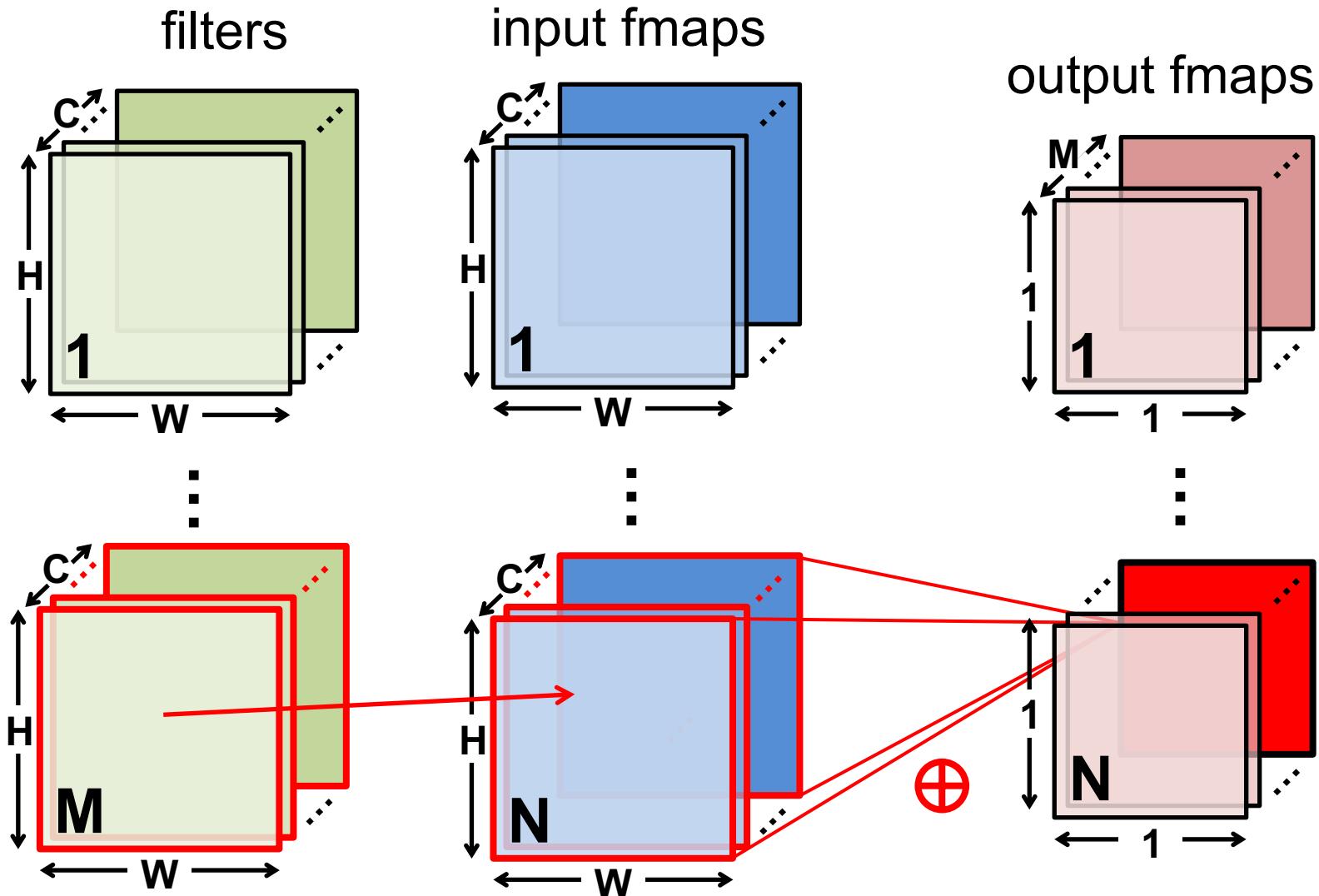


Fully-Connected (FC) Layer

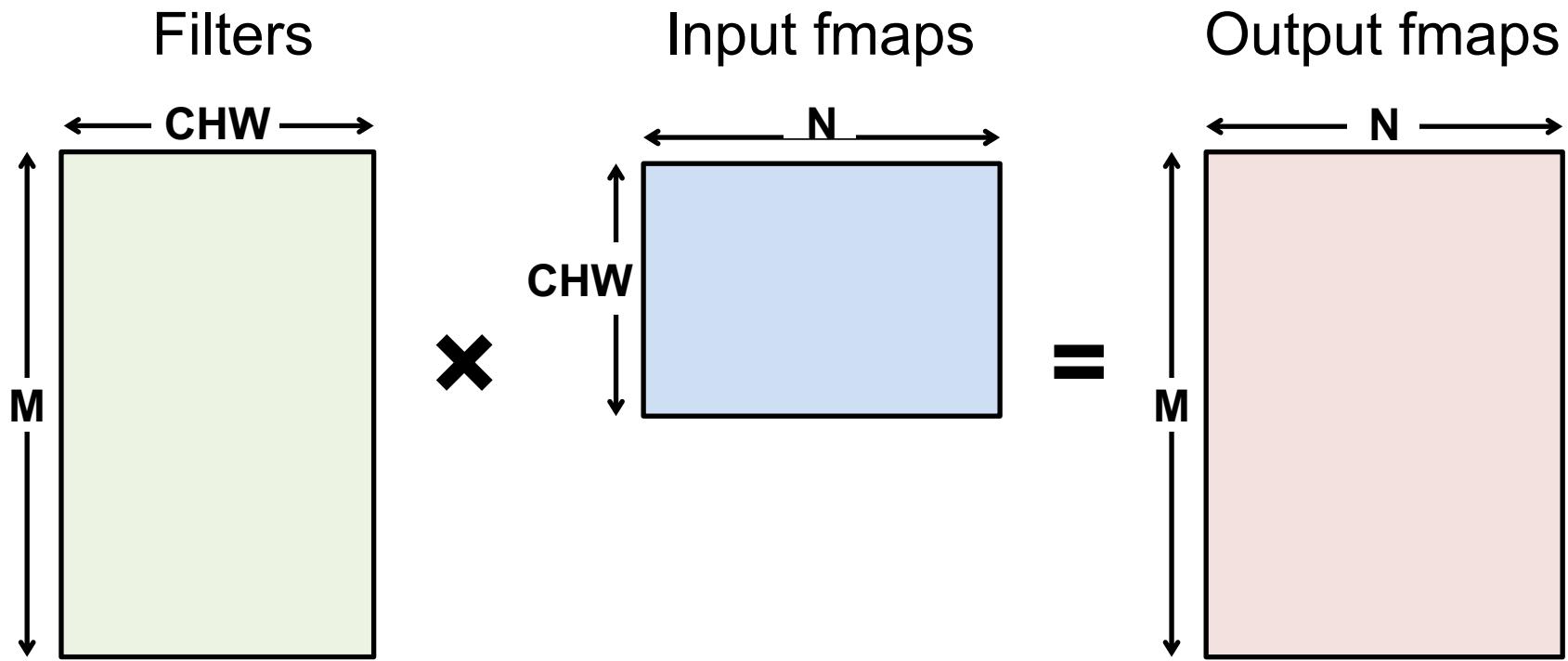
- Matrix–Vector Multiply:
 - Multiply all inputs in all channels by a weight and sum



Fully-Connected (FC) Layer

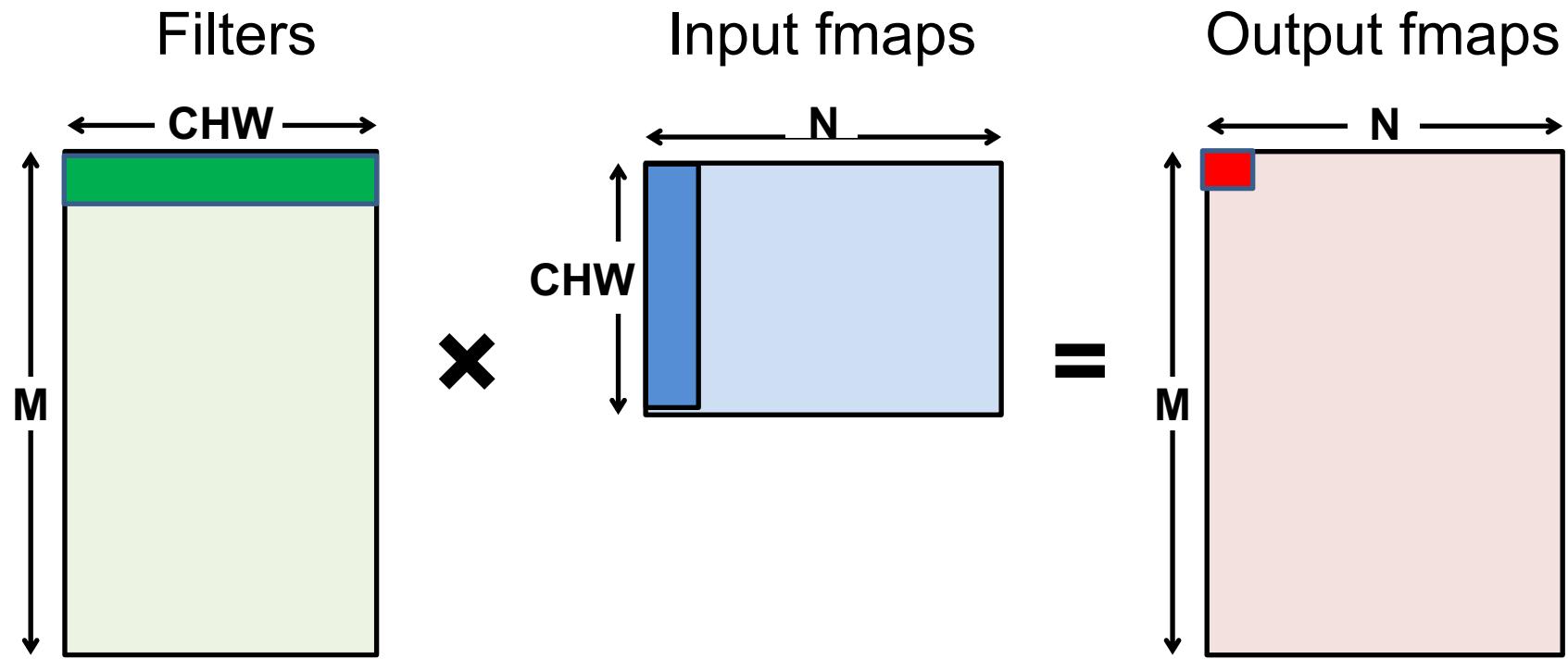


Flattened Fully-Connected Layer



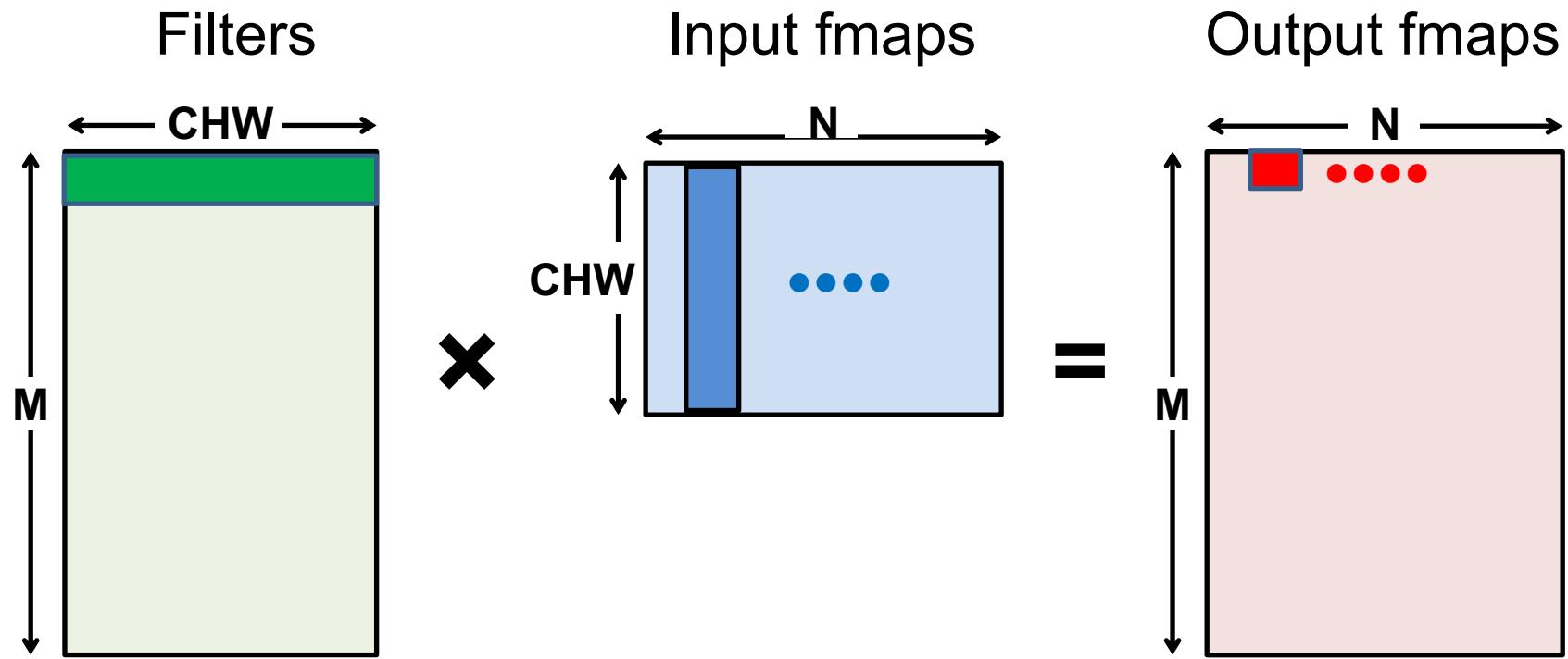
- After flattening, having a batch size of N turns the matrix-vector operation into a matrix-matrix multiply

Fully-Connected (FC) Layer



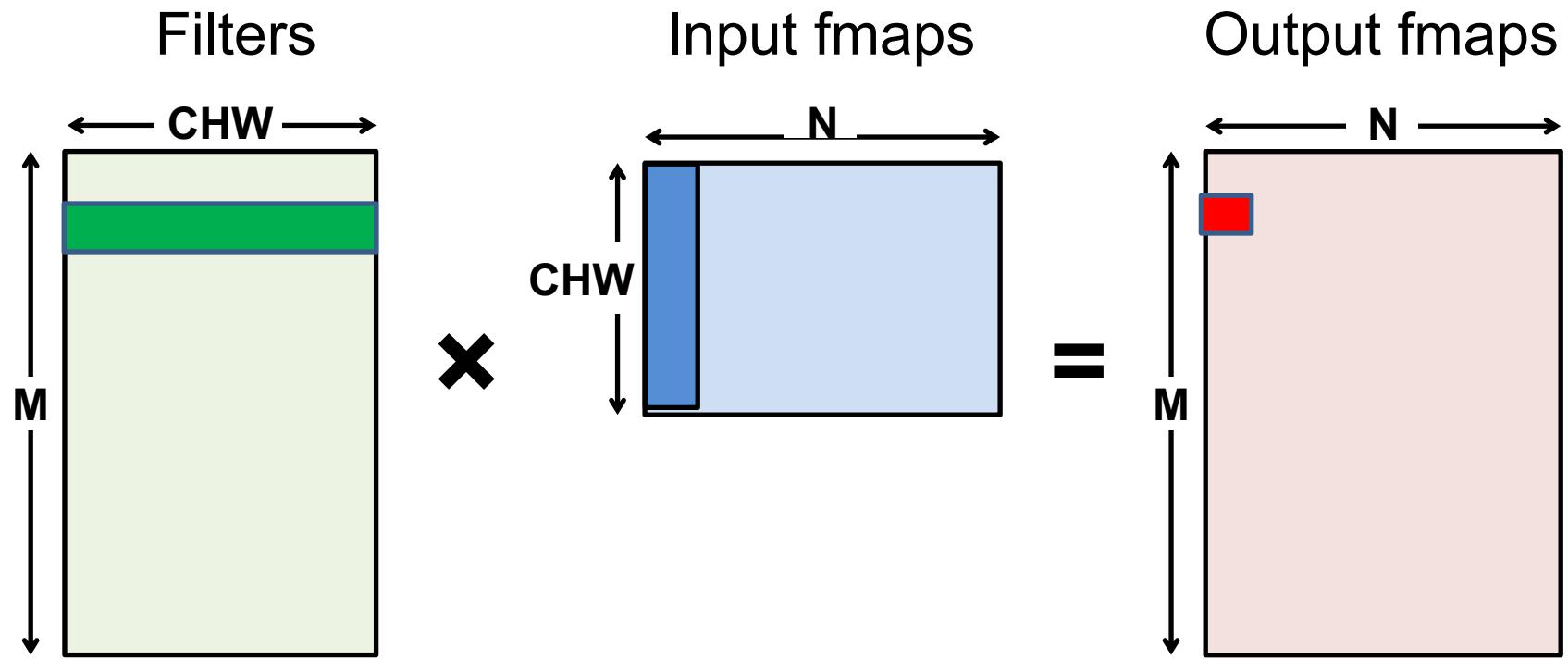
- After flattening, having a batch size of N turns the matrix-vector operation into a matrix-matrix multiply

Fully-Connected (FC) Layer



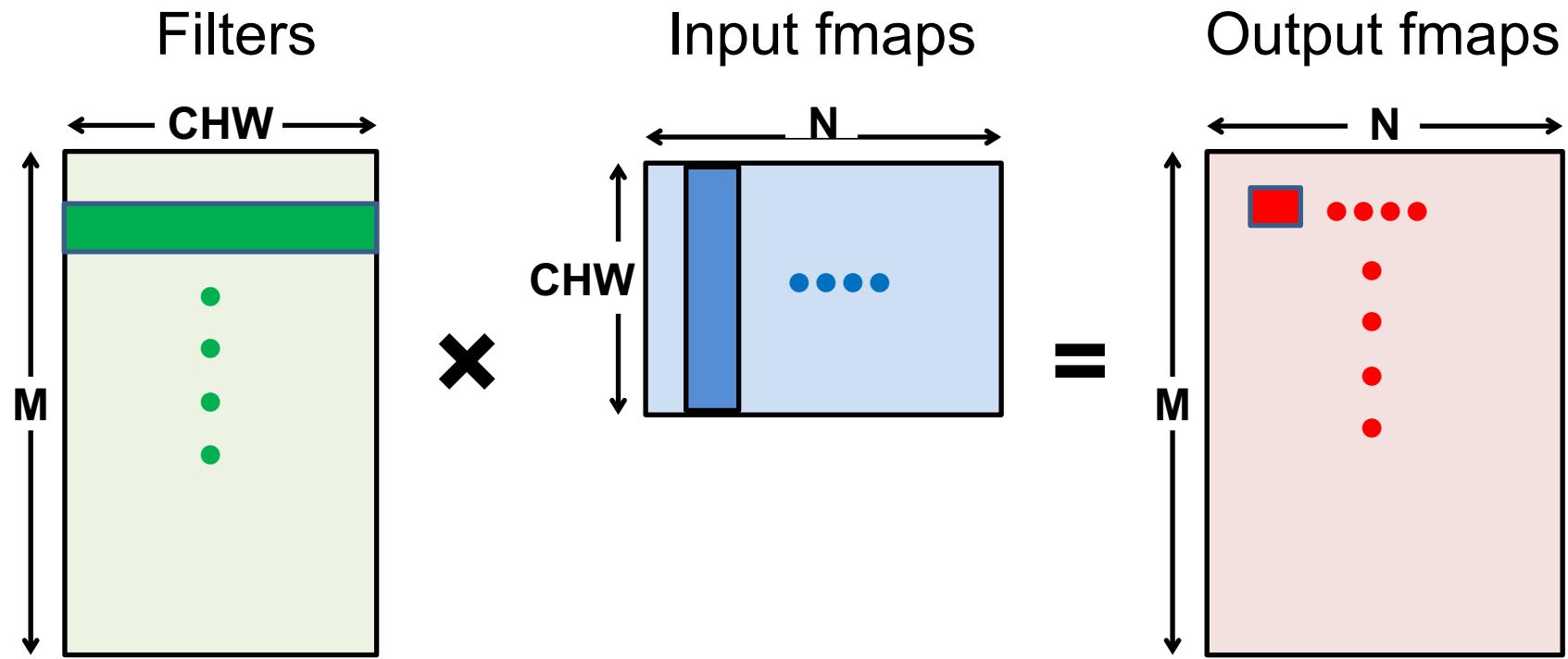
- After flattening, having a batch size of N turns the matrix-vector operation into a matrix-matrix multiply

Fully-Connected (FC) Layer



- After flattening, having a batch size of N turns the matrix-vector operation into a matrix-matrix multiply

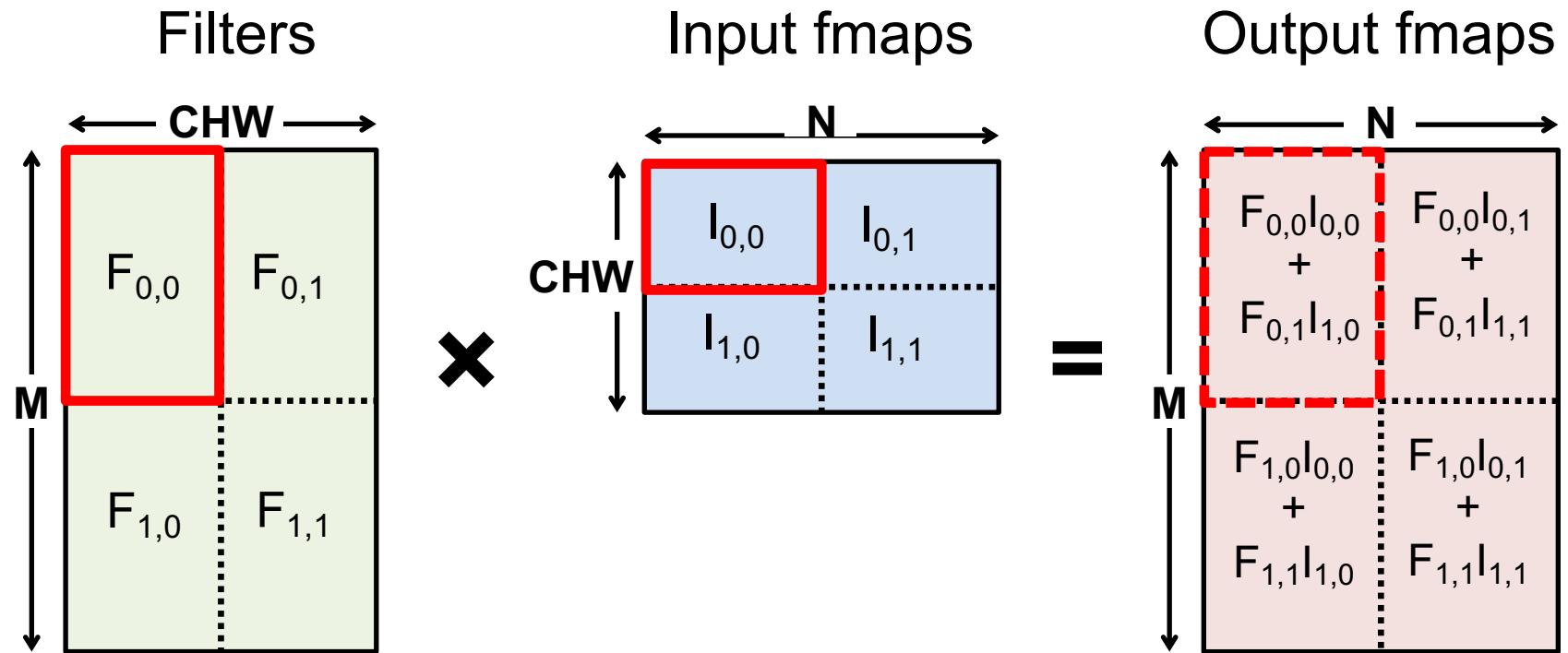
Fully-Connected (FC) Layer



- After flattening, having a batch size of N turns the matrix-vector operation into a matrix-matrix multiply

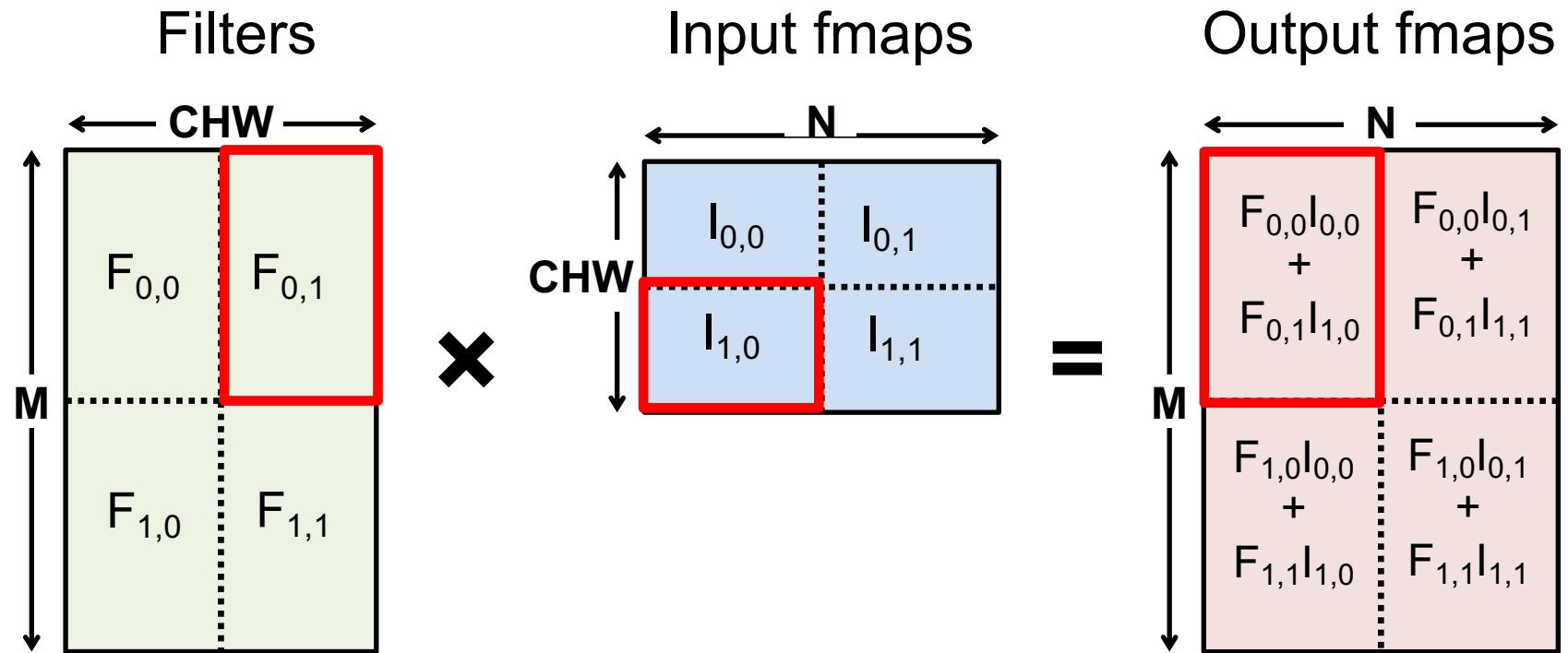
How much temporal locality for naïve implementation? **None**

Tiled Fully-Connected (FC) Layer



Matrix multiply tiled to fit in cache
and computation ordered to maximize reuse of data in cache

Tiled Fully-Connected (FC) Layer



Matrix multiply tiled to fit in cache
and computation ordered to maximize reuse of data in cache

Fully-Connected (FC) Layer

- Implementation: **Matrix Multiplication (GEMM)**
 - **CPU:** OpenBLAS, Intel MKL, etc
 - **GPU:** cuBLAS, cuDNN, etc
- Library will note shape of the matrix multiply and select implementation optimized for that shape.
- Optimization usually involves proper tiling to storage hierarchy

GV100 – “Tensor Core”

$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix}_{\text{FP16 or FP32}} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix}_{\text{FP16}} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}_{\text{FP16 or FP32}}$$

New opcodes – Matrix Multiply Accumulate (HMMA)

FP16 operands?

Inputs 48 / Outputs 16

Multiplies?

64

Adds?

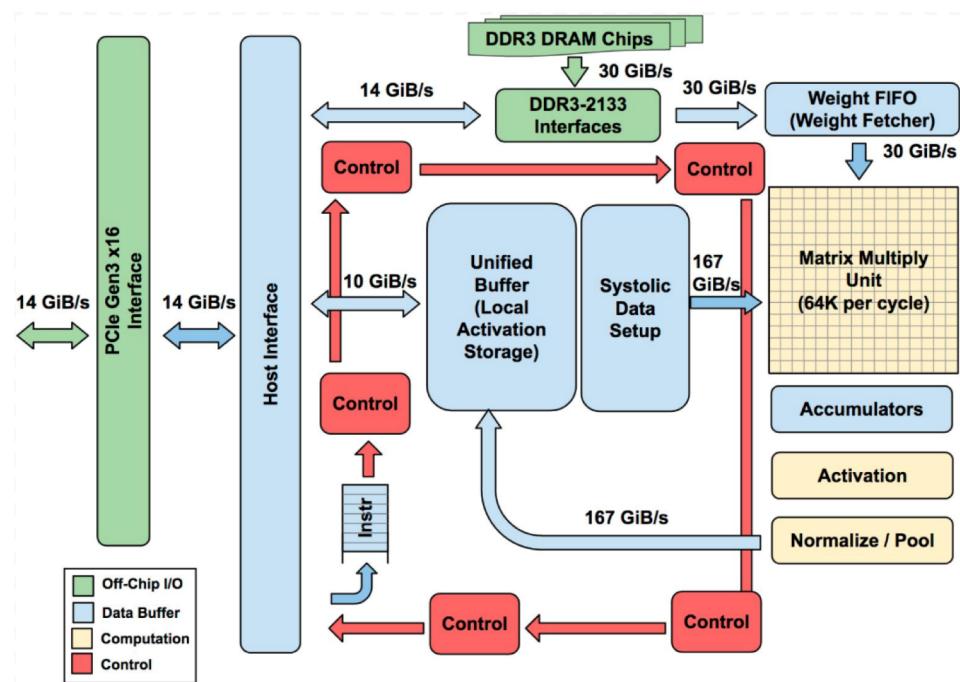
64

Tensor Core....

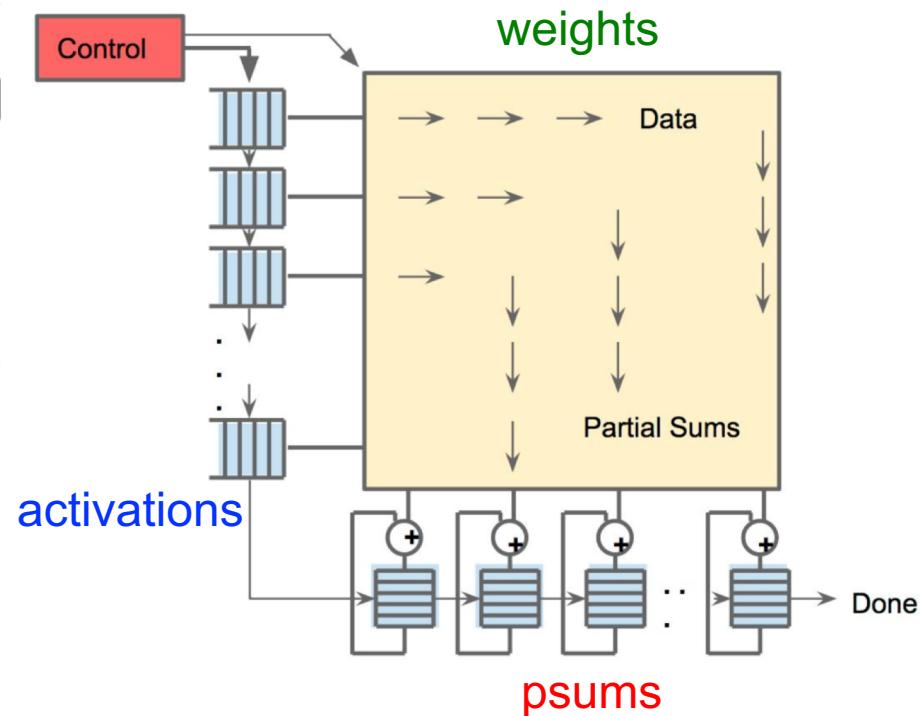
- 120 TFLOPS (FP16)
- 400 GFLOPS/W (FP16)

TPU

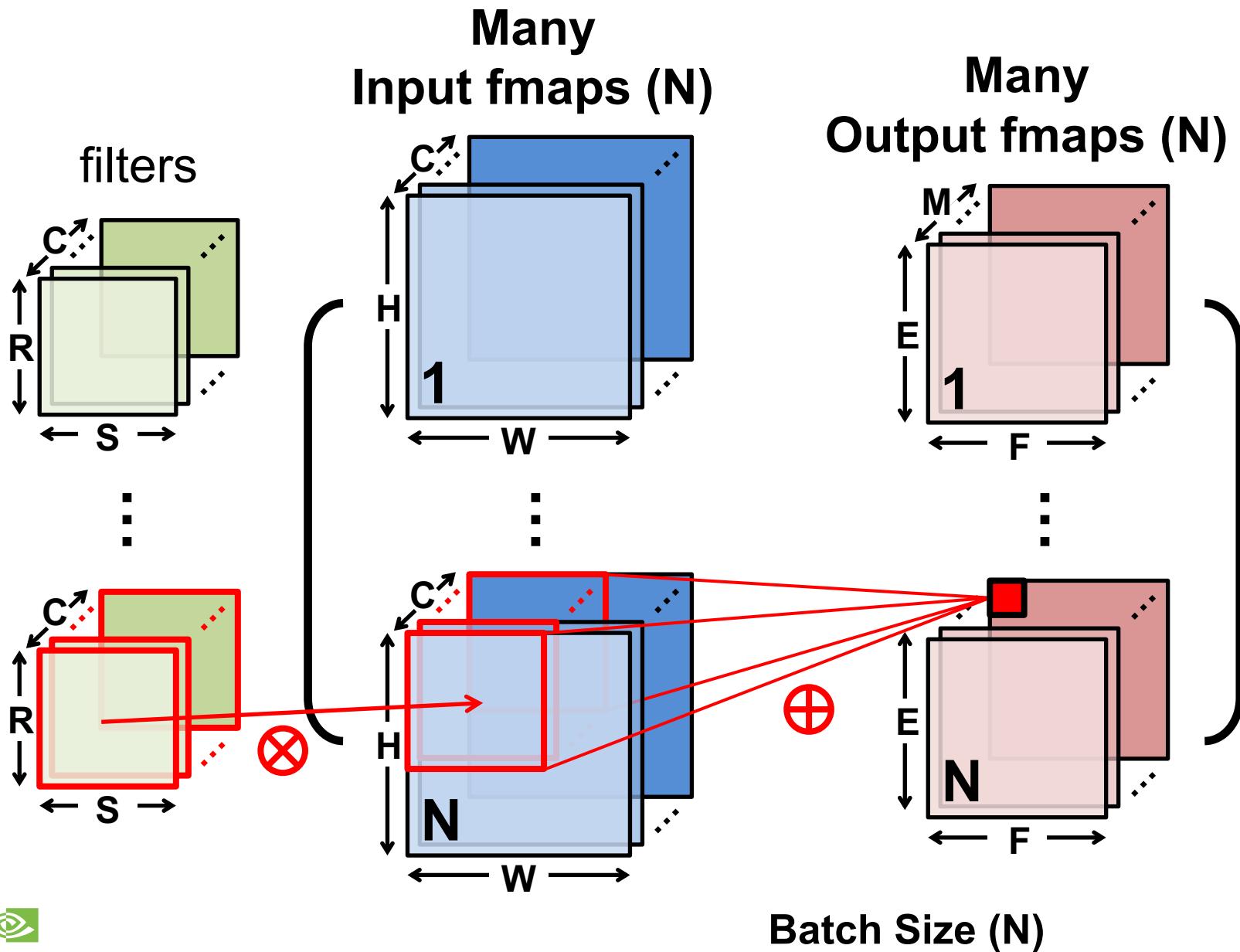
Top-Level Architecture



Matrix Multiply Unit



Convolution (CONV) Layer



CONV Layer Implementation

Naïve 7-layer for-loop implementation:

```
for (n=0; n<N; n++) {  
    for (m=0; m<M; m++) {  
        for (x=0; x<F; x++) {  
            for (y=0; y<E; y++) {  
  
                o[n][m][x][y] = B[m];  
                for (i=0; i<R; i++) {  
                    for (j=0; j<S; j++) {  
                        for (k=0; k<C; k++) {  
                            o[n][m][x][y] += I[n][k][Ux+i][Uy+j] × W[m][k][i][j];  
                        }  
                    }  
                }  
                o[n][m][x][y] = Activation(o[n][m][x][y]);  
            }  
        }  
    }  
}
```

convolve
a window
and apply
activation

} for each output fmap value

Convolution (CONV) Layer

Filter

1	2
3	4

2D dot
Product

Input Fmap

1	2	3
4	5	6
7	8	9

Output Fmap

1	2
3	4

*

=

1D dot
Product

Filtering steps

1	2
3	4

1	2
4	5

=

1



1	2	3	4
---	---	---	---

Flattened

1	
2	
4	
5	

1	
---	--

1	2
3	4

2	3
5	6

=

	2



1	2	3	4
---	---	---	---

1	2
3	4

4	5
7	8

=

		3



1	2
3	4

5	6
8	9

=

			4
--	--	--	---

Convolution (CONV) Layer

Filter	Input Fmap	Output Fmap													
<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td></tr></table>	1	2	3	4	$*$	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr><tr><td>7</td><td>8</td><td>9</td></tr></table>	1	2	3	4	5	6	7	8	9
1	2														
3	4														
1	2	3													
4	5	6													
7	8	9													
=		<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td></tr></table>	1	2	3	4									
1	2														
3	4														

Convolution:



Flattened

$$\begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \bullet \begin{matrix} 1 \\ 2 \\ 4 \\ 5 \end{matrix} = \begin{matrix} 1 & \boxed{} \end{matrix} \quad \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \bullet \begin{matrix} 2 \\ 3 \\ 5 \\ 6 \end{matrix} = \begin{matrix} \boxed{} & 2 & \boxed{} \end{matrix} \dots$$

Convolution (CONV) Layer

$$\begin{array}{c} \text{Filter} \\ \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} \end{array} * \begin{array}{c} \text{Input Fmap} \\ \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline 7 & 8 & 9 \\ \hline \end{array} \end{array} = \begin{array}{c} \text{Output Fmap} \\ \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} \end{array}$$

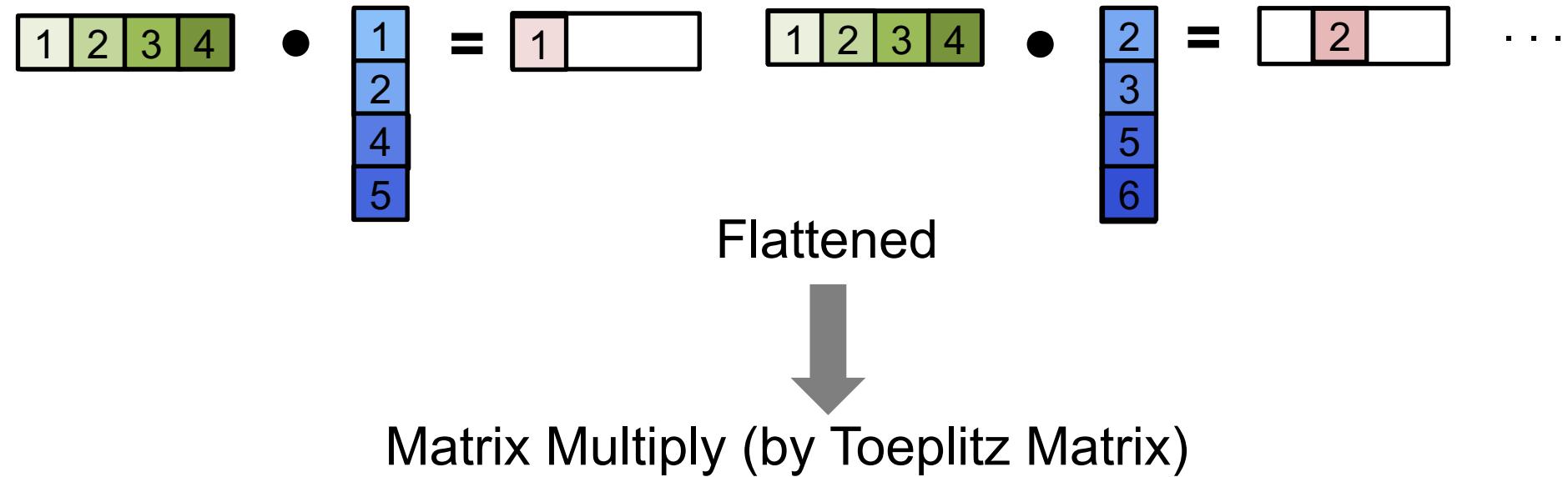
Convolution:



Flattened

$$\begin{array}{c} \begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 4 \\ \hline \end{array} \bullet \begin{array}{|c|c|c|} \hline 1 \\ \hline 2 \\ \hline 4 \\ \hline 5 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 1 & \boxed{} \\ \hline \end{array} \quad \begin{array}{|c|c|c|c|} \hline 1 & 2 & 3 & 4 \\ \hline \end{array} \bullet \begin{array}{|c|c|c|} \hline 2 \\ \hline 3 \\ \hline 5 \\ \hline 6 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline \boxed{} & 2 & \boxed{} \\ \hline \end{array} \dots \end{array}$$

Convolution (CONV) Layer



$$\begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 4 & 5 \\ 2 & 3 & 5 & 6 \\ 4 & 5 & 7 & 8 \\ 5 & 6 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}$$

Convolution (CONV) Layer

Filter	Input Fmap	Output Fmap													
<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td></tr></table>	1	2	3	4	$*$	$=$									
1	2														
3	4														
	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr><tr><td>7</td><td>8</td><td>9</td></tr></table>	1	2	3	4	5	6	7	8	9	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td></tr></table>	1	2	3	4
1	2	3													
4	5	6													
7	8	9													
1	2														
3	4														

Convolution:



Matrix Multiply (by Toeplitz Matrix)

<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>	1	2	3	4	\times	$=$																
1	2	3	4																			
	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>2</td><td>4</td><td>5</td></tr><tr><td>2</td><td>3</td><td>5</td><td>6</td></tr><tr><td>4</td><td>5</td><td>7</td><td>8</td></tr><tr><td>5</td><td>6</td><td>8</td><td>9</td></tr></table>	1	2	4	5	2	3	5	6	4	5	7	8	5	6	8	9	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>	1	2	3	4
1	2	4	5																			
2	3	5	6																			
4	5	7	8																			
5	6	8	9																			
1	2	3	4																			

Convert to matrix multiply using the **Toeplitz Matrix**

Convolution (CONV) Layer

Filter	Input Fmap	Output Fmap													
<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td></tr></table>	1	2	3	4	$*$	$=$									
1	2														
3	4														
	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr><tr><td>7</td><td>8</td><td>9</td></tr></table>	1	2	3	4	5	6	7	8	9	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td></tr></table>	1	2	3	4
1	2	3													
4	5	6													
7	8	9													
1	2														
3	4														

Convolution:



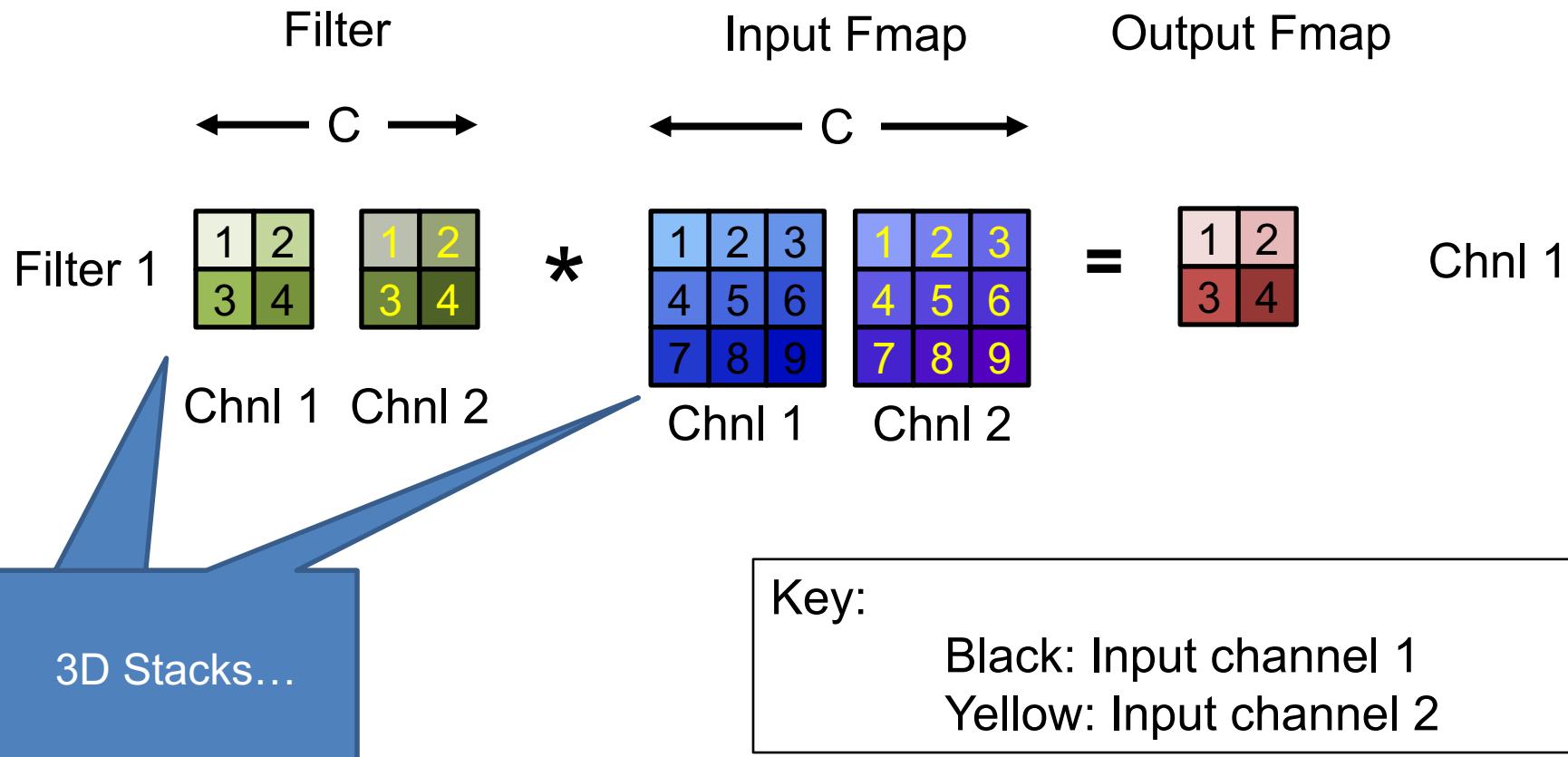
Matrix Multiply (by Toeplitz Matrix)

<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>	1	2	3	4	\times	$=$																
1	2	3	4																			
	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>2</td><td>4</td><td>5</td></tr><tr><td>2</td><td>3</td><td>5</td><td>6</td></tr><tr><td>4</td><td>5</td><td>7</td><td>8</td></tr><tr><td>5</td><td>6</td><td>8</td><td>9</td></tr></table>	1	2	4	5	2	3	5	6	4	5	7	8	5	6	8	9	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>	1	2	3	4
1	2	4	5																			
2	3	5	6																			
4	5	7	8																			
5	6	8	9																			
1	2	3	4																			

Data is repeated

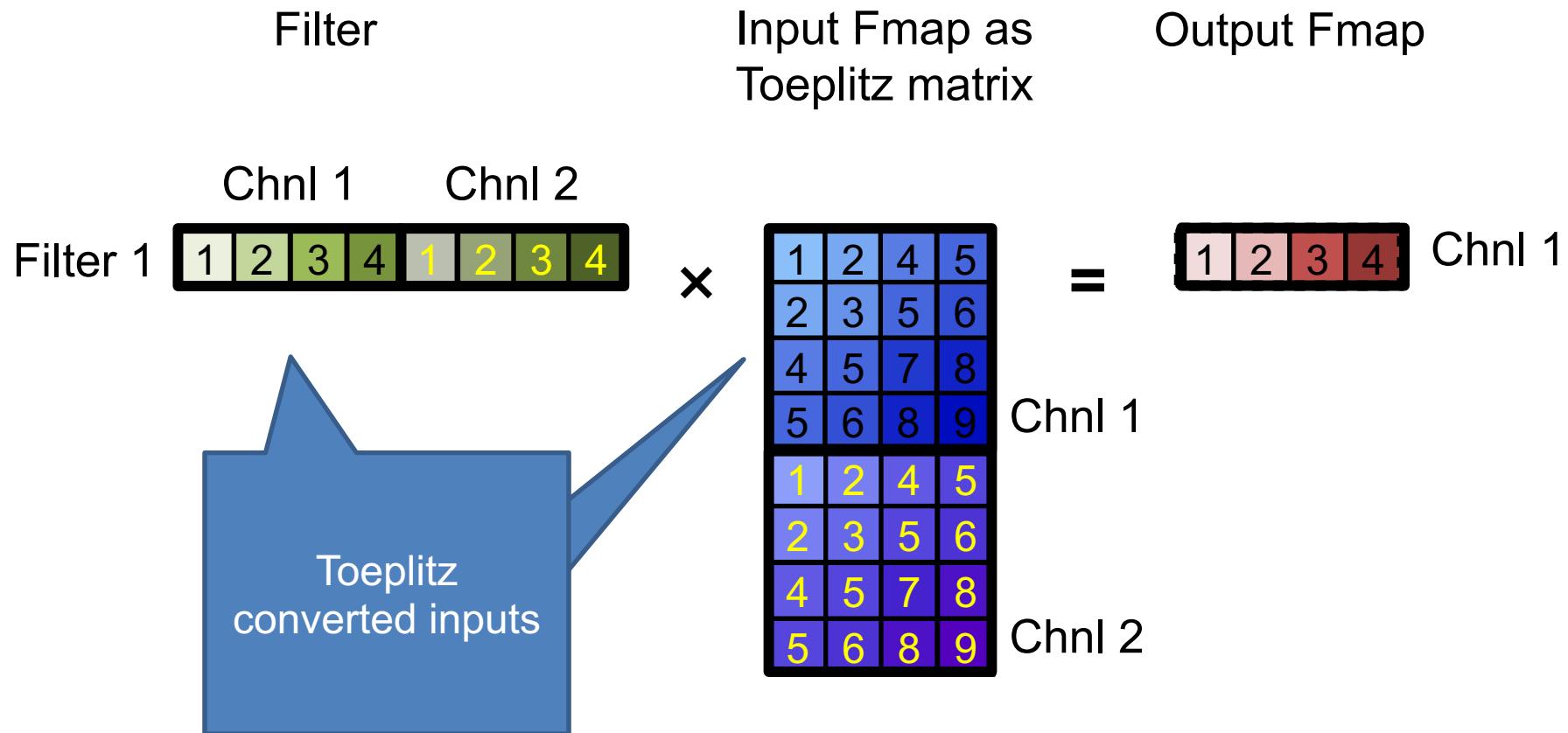
Convolution (CONV) Layer

- Multiple Input Channels



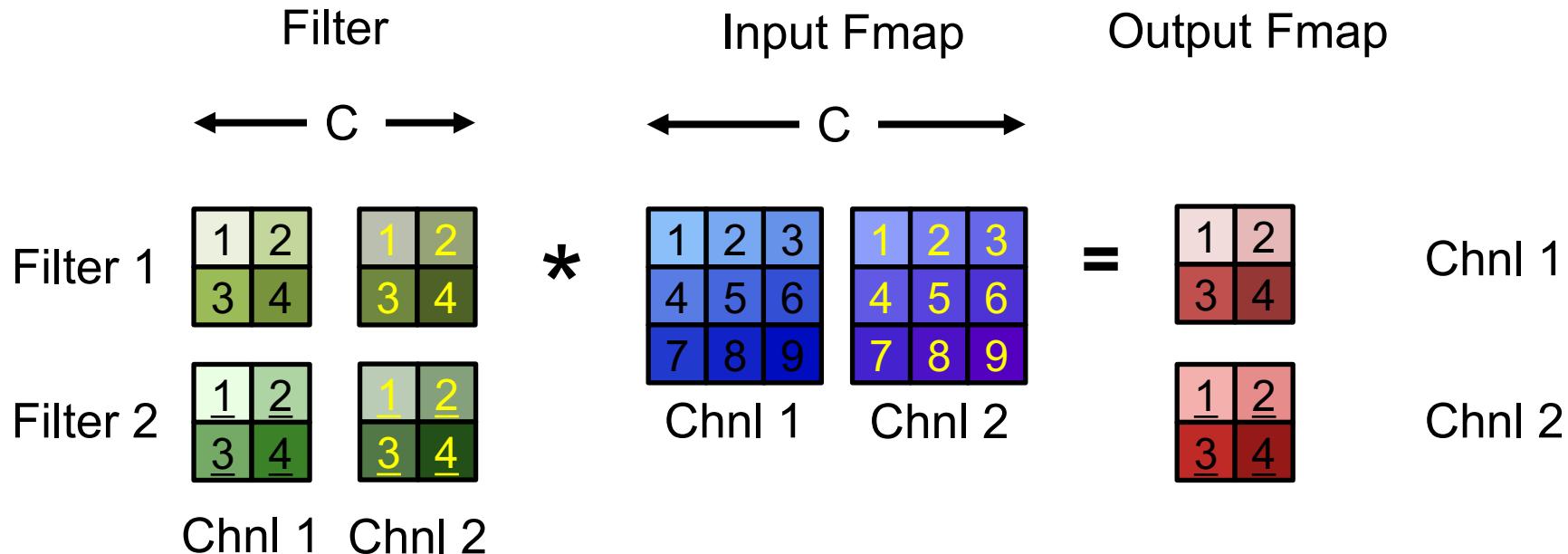
Convolution (CONV) Layer

- Multiple Input Channels



Convolution (CONV) Layer

- Multiple Input Channels and Output Channels



Key:

Black: Input channel 1
Yellow: Input channel 2
Underlined: Output channel 2

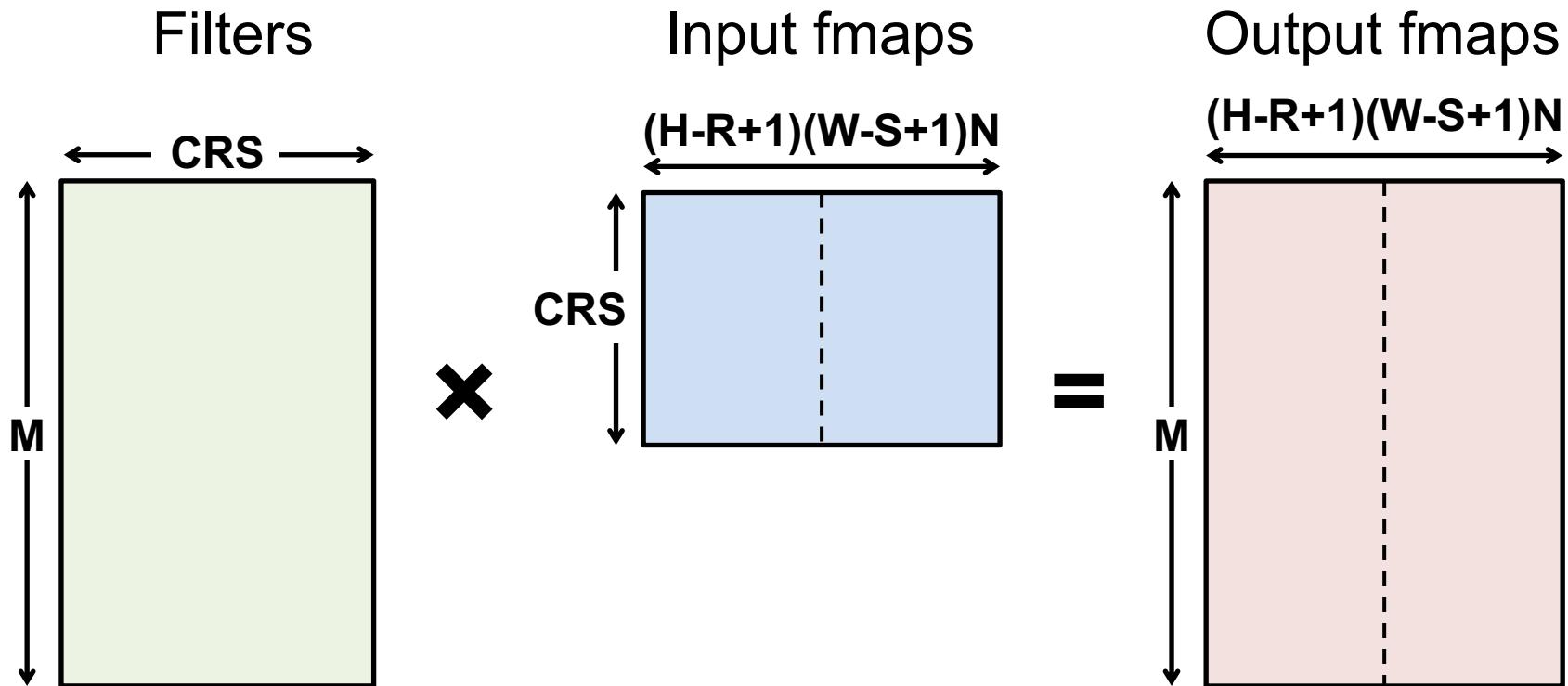
Convolution (CONV) Layer

- Multiple Input Channels and Output Channels

	Filter	Input Fmap as Toeplitz matrix	Output Fmap																																								
Chnl 1																																											
Filter 1	<table border="1" style="border-collapse: collapse; text-align: center;"><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	<table border="1" style="border-collapse: collapse; text-align: center;"><tr><td>1</td><td>2</td><td>4</td><td>5</td></tr><tr><td>2</td><td>3</td><td>5</td><td>6</td></tr><tr><td>4</td><td>5</td><td>7</td><td>8</td></tr><tr><td>5</td><td>6</td><td>8</td><td>9</td></tr></table>	1	2	4	5	2	3	5	6	4	5	7	8	5	6	8	9	<table border="1" style="border-collapse: collapse; text-align: center;"><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>	1	2	3	4	1	2	3	4
1	2	3	4	1	2	3	4																																				
1	2	3	4	1	2	3	4																																				
1	2	4	5																																								
2	3	5	6																																								
4	5	7	8																																								
5	6	8	9																																								
1	2	3	4																																								
1	2	3	4																																								
Chnl 2																																											
Filter 2	<table border="1" style="border-collapse: collapse; text-align: center;"><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	<table border="1" style="border-collapse: collapse; text-align: center;"><tr><td>1</td><td>2</td><td>4</td><td>5</td></tr><tr><td>2</td><td>3</td><td>5</td><td>6</td></tr><tr><td>4</td><td>5</td><td>7</td><td>8</td></tr><tr><td>5</td><td>6</td><td>8</td><td>9</td></tr></table>	1	2	4	5	2	3	5	6	4	5	7	8	5	6	8	9	<table border="1" style="border-collapse: collapse; text-align: center;"><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>	1	2	3	4	1	2	3	4
1	2	3	4	1	2	3	4																																				
1	2	3	4	1	2	3	4																																				
1	2	4	5																																								
2	3	5	6																																								
4	5	7	8																																								
5	6	8	9																																								
1	2	3	4																																								
1	2	3	4																																								
	×	=																																									
			Chnl 1																																								
			Chnl 2																																								

Convolution (CONV) Layer

- Dimensions of matrices for matrix multiply in convolution layers with batch size N



N=2 in example

Computational Transforms

Computation Transformations

- Goal: Bitwise same result, but reduce number of operations
- Focuses mostly on compute

Gauss's Multiplication Algorithm

$$(a + bi)(c + di) = (ac - bd) + (bc + ad)i.$$

4 multiplications + 3 additions

$$k_1 = c \cdot (a + b)$$

$$k_2 = a \cdot (d - c)$$

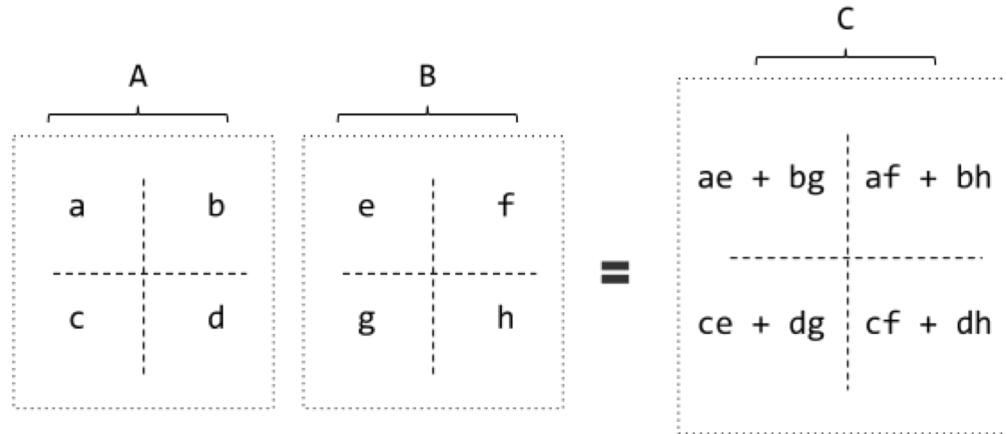
$$k_3 = b \cdot (c + d)$$

$$\text{Real part} = k_1 - k_3$$

$$\text{Imaginary part} = k_1 + k_2.$$

3 multiplications + 5 additions

Strassen



8 multiplications + 4 additions

$$\begin{aligned} P1 &= a(f - h) \\ P2 &= (a + b)h \\ P3 &= (c + d)e \\ P4 &= d(g - e) \end{aligned}$$

$$\begin{aligned} P5 &= (a + d)(e + h) \\ P6 &= (b - d)(g + h) \\ P7 &= (a - c)(e + f) \end{aligned}$$

$$AB = \begin{bmatrix} P5 + P4 - P2 + P6 & P1 + P2 \\ P3 + P4 & P1 + P5 - P3 - P7 \end{bmatrix}$$

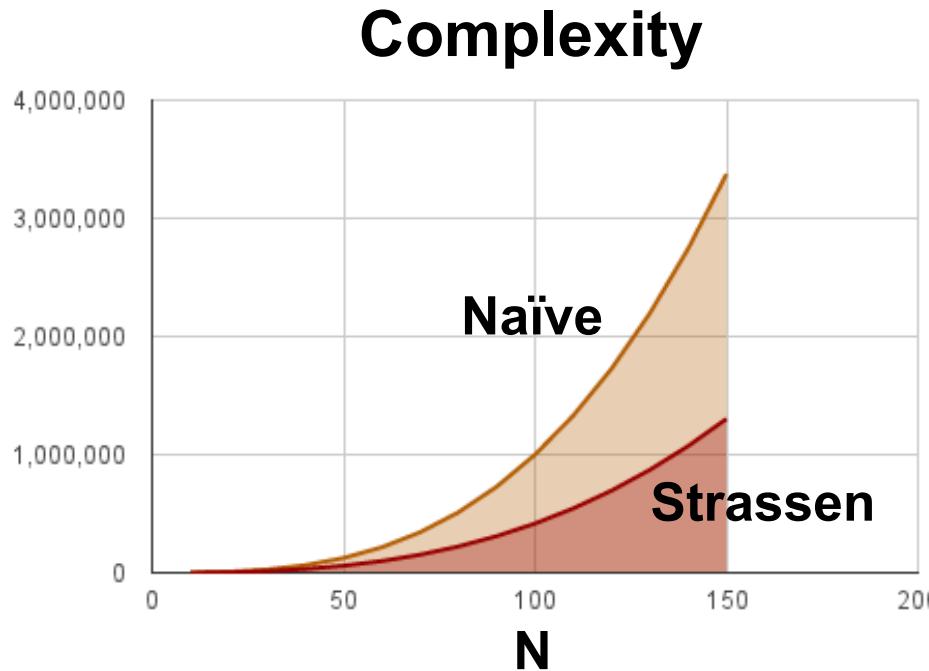
7 multiplications + 18 additions

7 multiplications + 13 additions (for constant B matrix – weights)

[Cong et al., ICANN, 2014]

Strassen

- Reduce the complexity of matrix multiplication from $\Theta(N^3)$ to $\Theta(N^{2.807})$ by reducing multiplications



Comes at the price of reduced numerical stability and requires significantly more memory

Image Source: <http://www.stoimen.com/blog/2012/11/26/computer-algorithms-strassens-matrix-multiplication/>

Winograd 1D – F(2,3)

- Targeting convolutions instead of matrix multiply
- Notation: F(size of output, filter size)

$$F(2, 3) = \begin{matrix} \text{input} \\ \left[\begin{matrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{matrix} \right] \end{matrix} \begin{matrix} \text{filter} \\ \left[\begin{matrix} g_0 \\ g_1 \\ g_2 \end{matrix} \right] \end{matrix} = \begin{bmatrix} y_0 \\ y_1 \end{bmatrix}$$

6 multiplications + 4 additions

[Lavin et al., CVPR 2016]

Winograd 1D – F(2,3)

- Targeting convolutions instead of matrix multiply
- Notation: F(size of output, filter size)

$$F(2,3) = \begin{matrix} & \text{input} & \text{filter} \\ \begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} & \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} & = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix} \end{matrix}$$

$$m_1 = (d_0 - d_2)g_0 \quad m_2 = (d_1 + d_2)\frac{g_0 + g_1 + g_2}{2}$$

$$m_4 = (d_1 - d_3)g_2 \quad m_3 = (d_2 - d_1)\frac{g_0 - g_1 + g_2}{2}$$

4 multiplications + 12 additions + 2 shifts

4 multiplications + 8 additions (for constant weights)

[Lavin et al., CVPR 2016]

Winograd 2D - F(2x2, 3x3)

- 1D Winograd is nested to make 2D Winograd

Filter	Input Fmap	Output Fmap
g ₀₀ g ₀₁ g ₀₂	d ₀₀ d ₀₁ d ₀₂ d ₀₃	y ₀₀ y ₀₁
g ₁₀ g ₁₁ g ₁₂	d ₁₀ d ₁₁ d ₁₂ d ₁₃	y ₁₀ y ₁₁
g ₂₀ g ₂₁ g ₂₂	d ₂₀ d ₂₁ d ₂₂ d ₂₃	
	d ₃₀ d ₃₁ d ₃₂ d ₃₃	

*

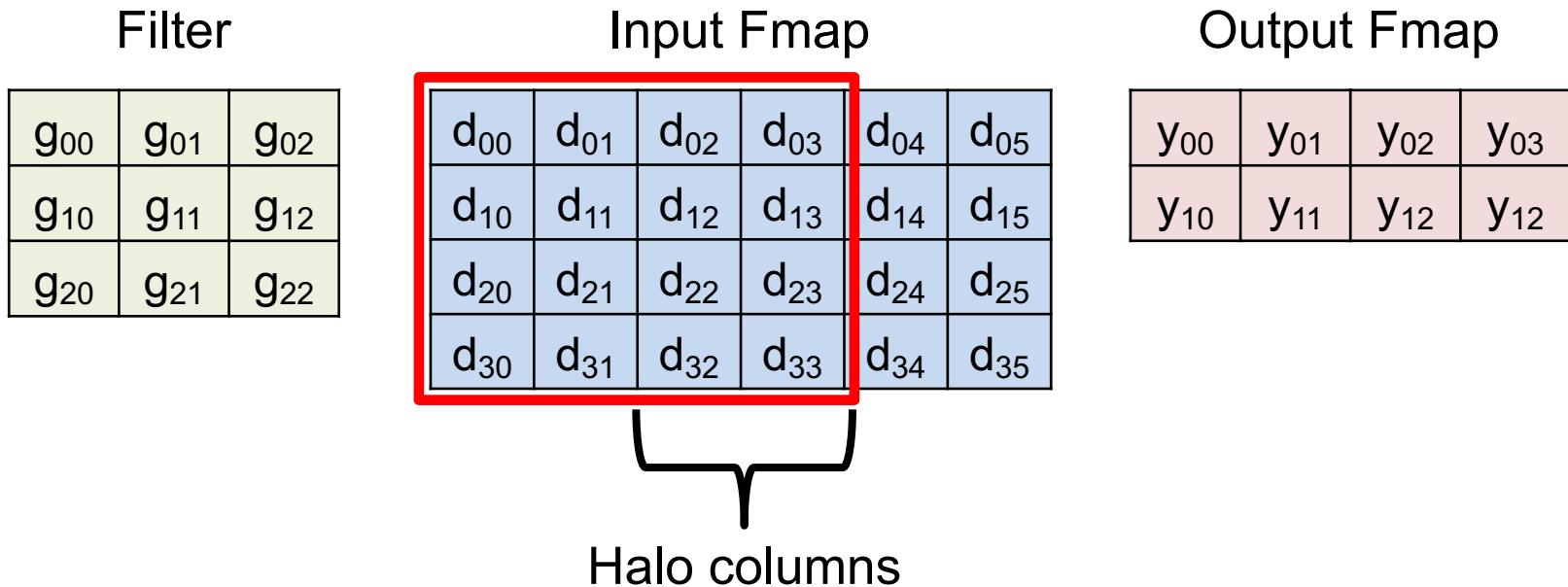
=

Original: 36 multiplications

Winograd: 16 multiplications → 2.25 times reduction

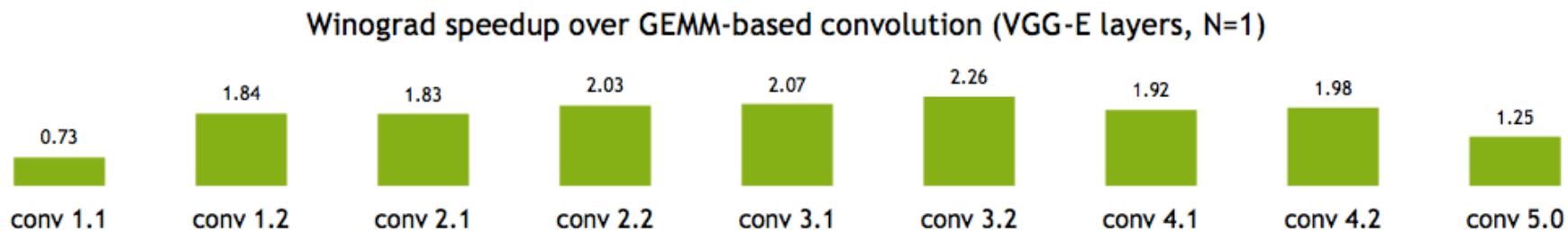
Winograd Halos

- Winograd works on a small region of output at a time, and therefore uses inputs repeatedly



Winograd Performance Varies

Optimal convolution algorithm depends on convolution layer dimensions



Meta-parameters (data layouts, texture memory) afford higher performance

Using texture memory for convolutions: **13% inference speedup**

(GoogLeNet, batch size 1)



Source: Nvidia

Winograd Summary

- Winograd is an optimized computation for convolutions
- It can significantly reduce multiplies
 - For example, for 3x3 filter by 2.25X
- But, each filter size (and output size) is a different computation.

Winograd as a Transform

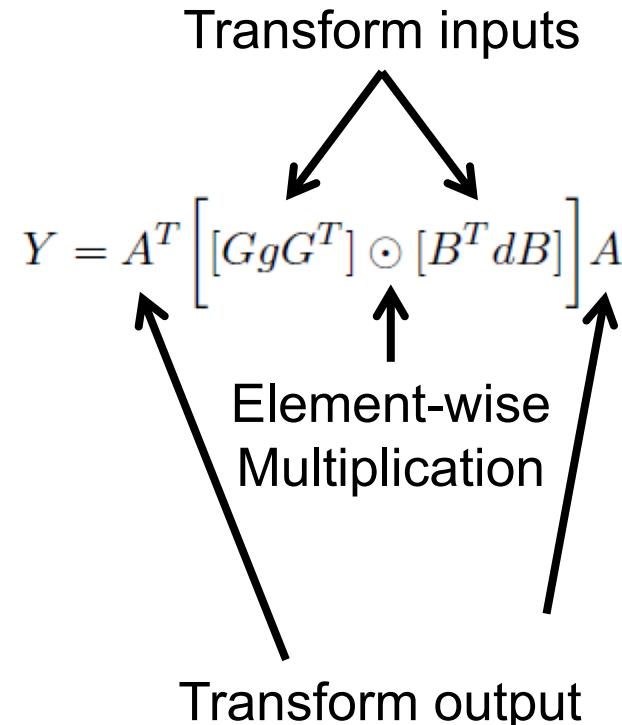
$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & 1 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

filter $g = [g_0 \ g_1 \ g_2]^T$

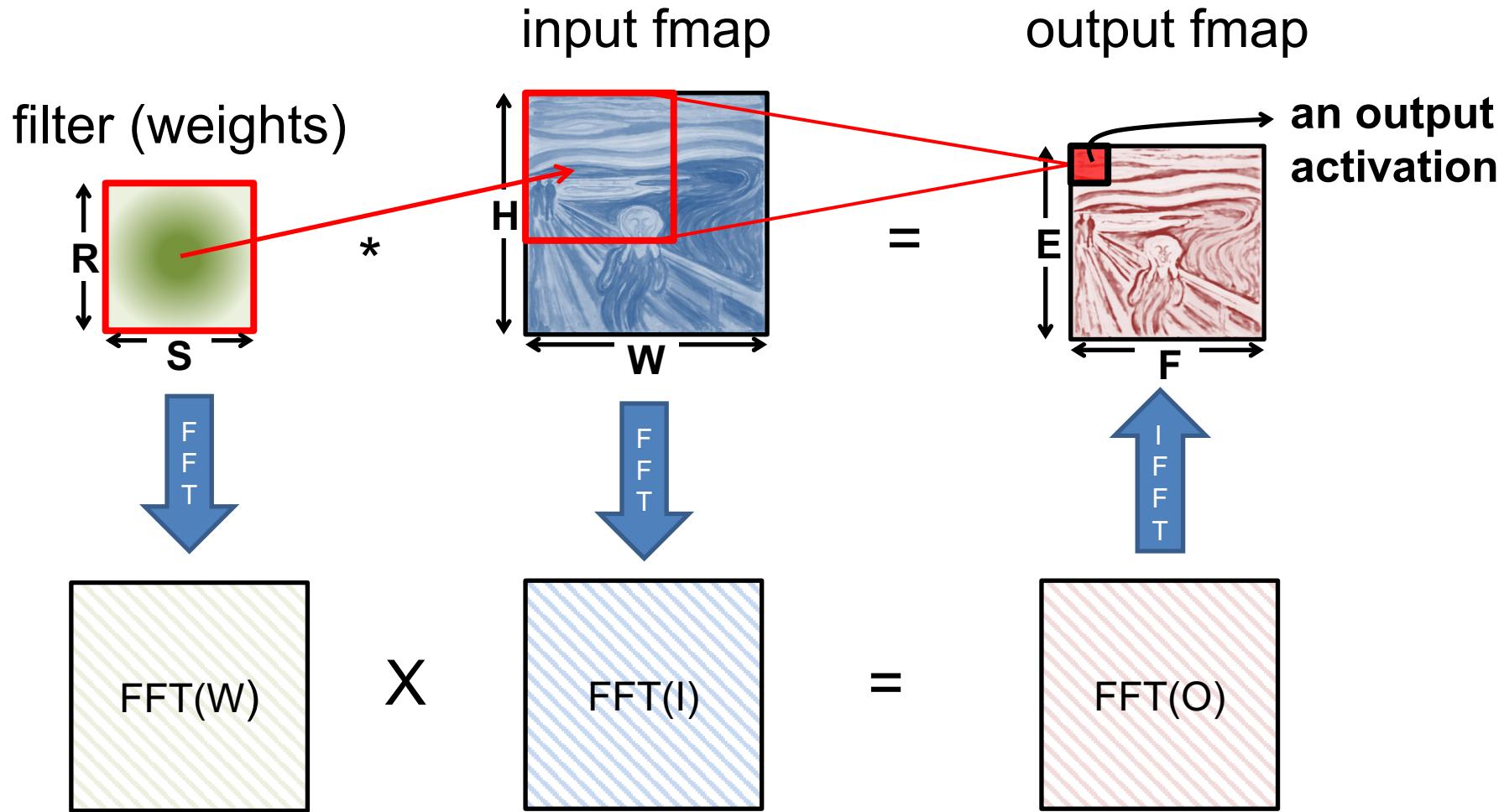
input $d = [d_0 \ d_1 \ d_2 \ d_3]^T$



GgG^T can be precomputed

[Lavin et al., CVPR 2016]

Fast Fourier Transform (FFT) Flow



FFT Overview

- Convert filter and input to frequency domain to make convolution a simple multiply then convert back to space domain.
- Convert direct convolution $O(N_o^2 N_f^2)$ computation to $O(N_o^2 \log_2 N_o)$
- So note that computational benefit of FFT decreases with decreasing size of filter

[Mathieu et al., ArXiv 2013, Vasilache et al., ArXiv 2014]

FFT Costs

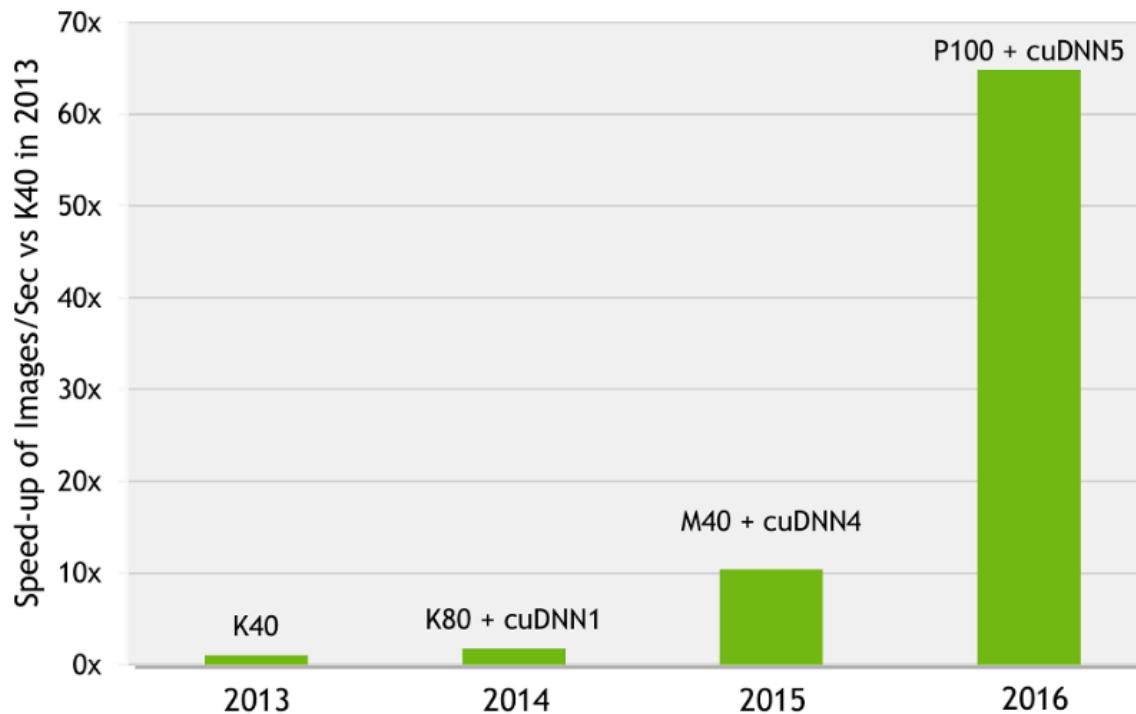
- Input and Filter matrices are ‘0-completed’,
 - i.e., expanded to size $E+R-1 \times F+S-1$
- Frequency domain matrices are same dimensions as input, but complex.
- FFT often reduces computation, but requires much more memory space and bandwidth

Optimization opportunities

- FFT of real matrix is symmetric allowing one to save $\frac{1}{2}$ the computes
- Filters can be pre-computed and stored, but convolutional filter in frequency domain is much larger than in space domain
- Can reuse frequency domain version of input for creating different output channels to avoid FFT re-computations
- Can accumulate across channels before performing inverse transform to reduce number of IFFT

cuDNN: Speed up with Transformations

60x Faster Training in 3 Years



AlexNet training throughput on:

CPU: 1x E5-2680v3 12 Core 2.5GHz. 128GB System Memory, Ubuntu 14.04

M40 bar: 8x M40 GPUs in a node, P100: 8x P100 NVLink-enabled

Source: Nvidia

Optimization Tools

- Halide
- TVM
- Timeloop
-