Basics on DNNs and Kernel Computation

Keypoints in Lecture 1

- Technology related issues
  - End of Moore's law, Dennard Scaling, ...
- Turing Tariff

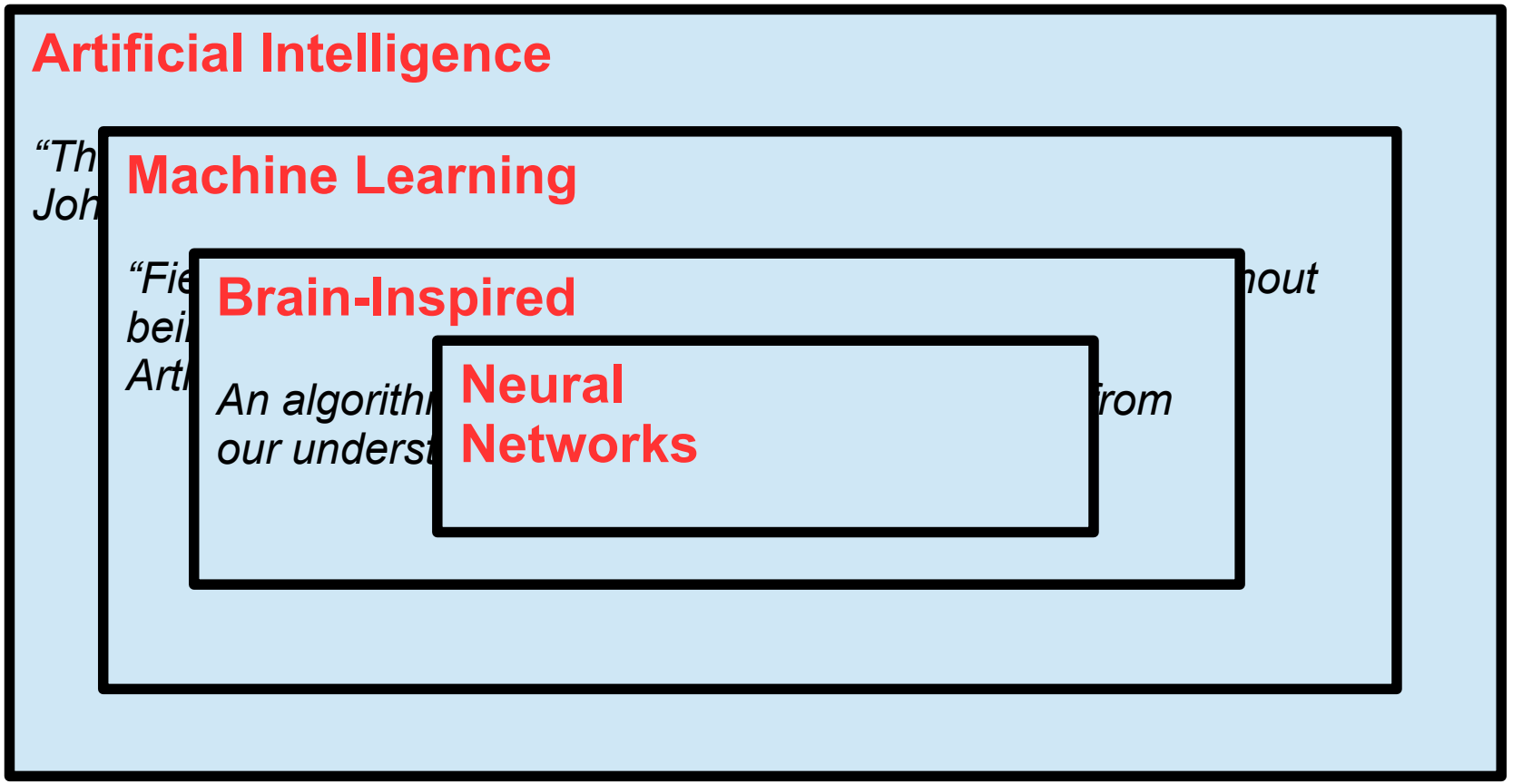
- Domain Specific Architectures
  - ASIP, GPU, FPGA, ASIC

Workshop Agenda

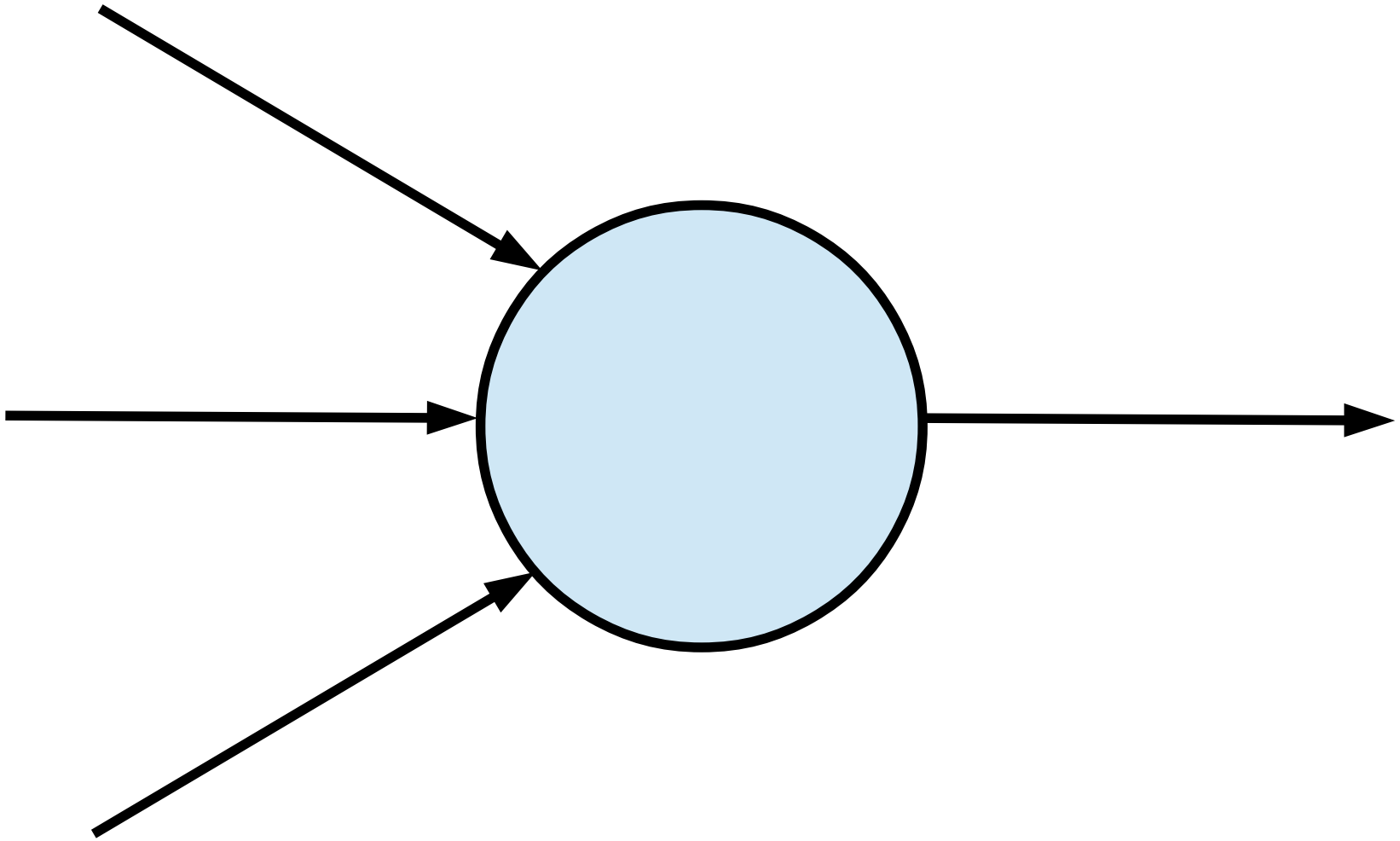
- Lecture 1: Domain Specific Architectures
- Lecture 2: Kernel computation
- Lecture 3: Data-flow techniques
- Lecture 4: DNN accelerators architectures

Agenda – Lecture 2

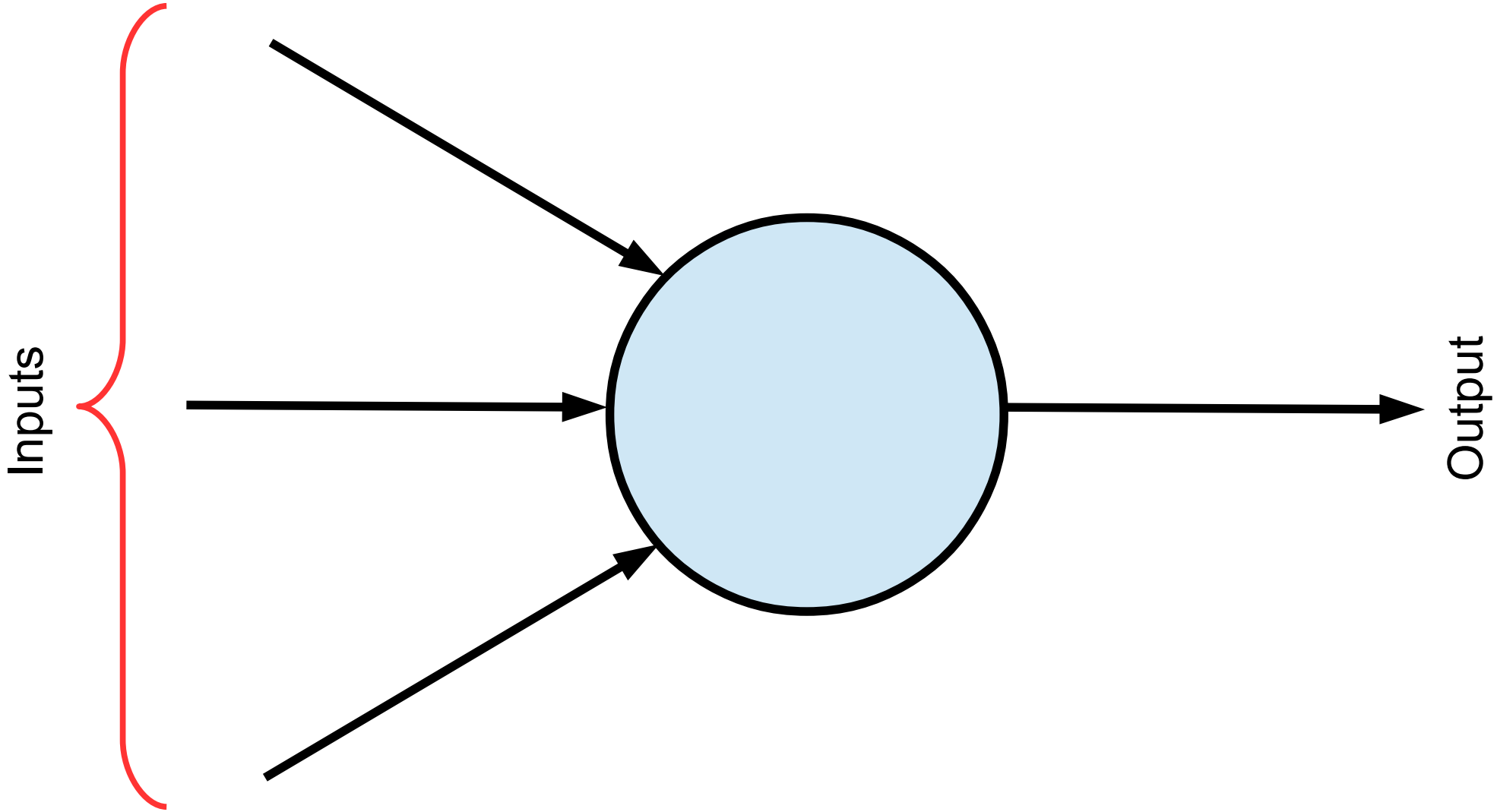
- Deep Neural Networks
- Types of layers
- Memory and communication traffic
- Kernel computation
- Quantitative analysis


Artificial Intelligence
Machine Learning
Brain-Inspired
Neural
Networks

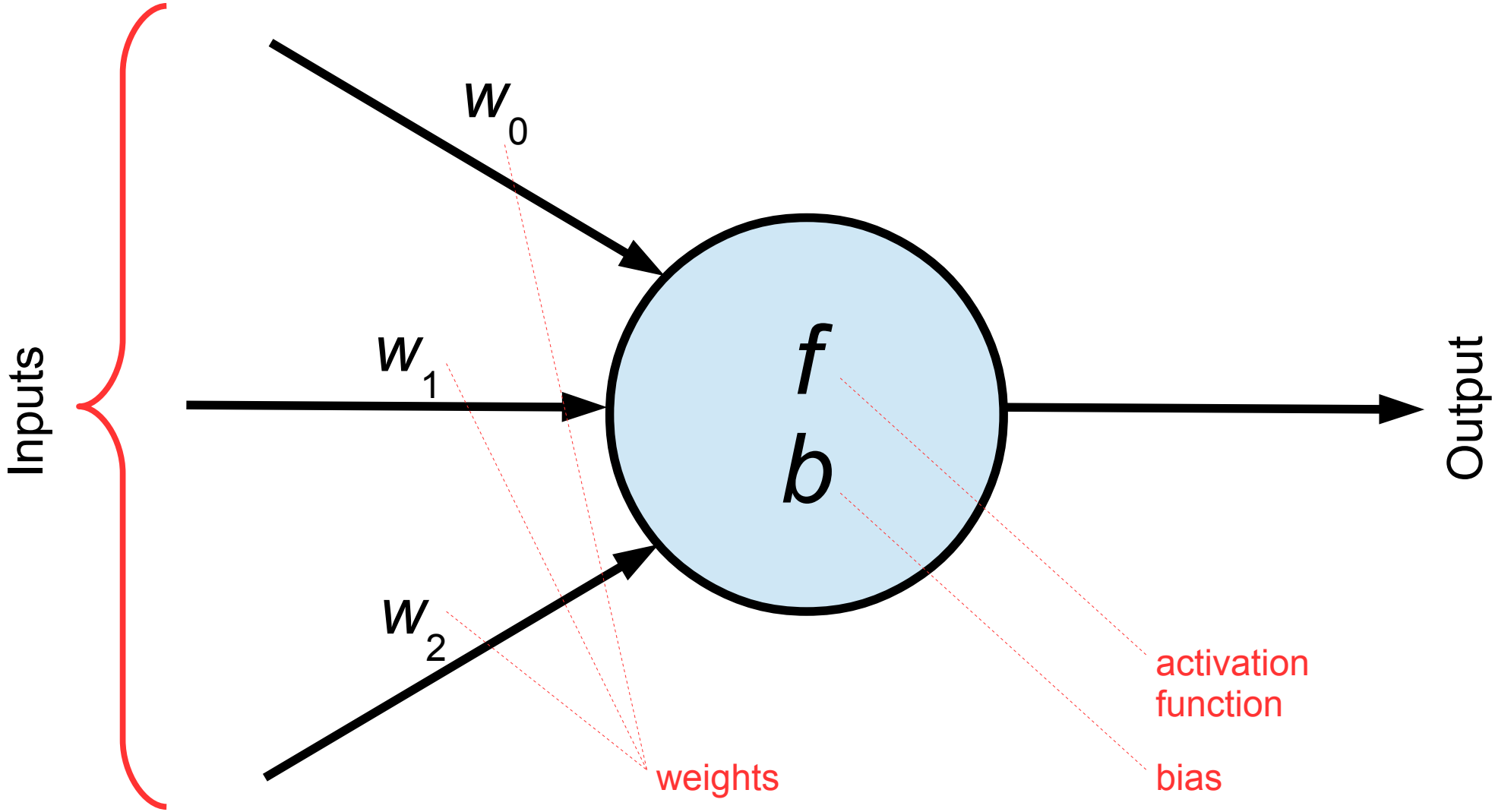
Neuron



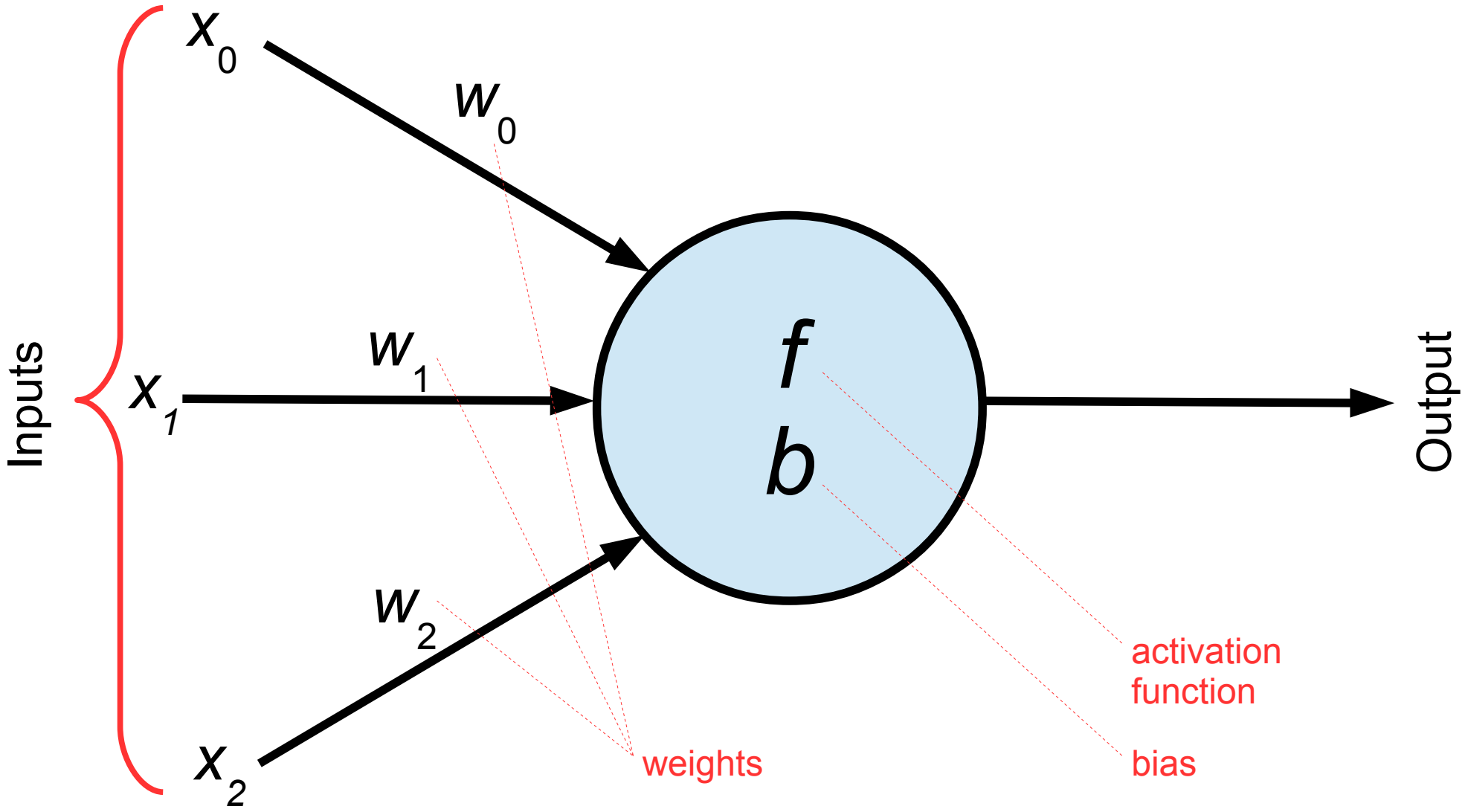
Neuron



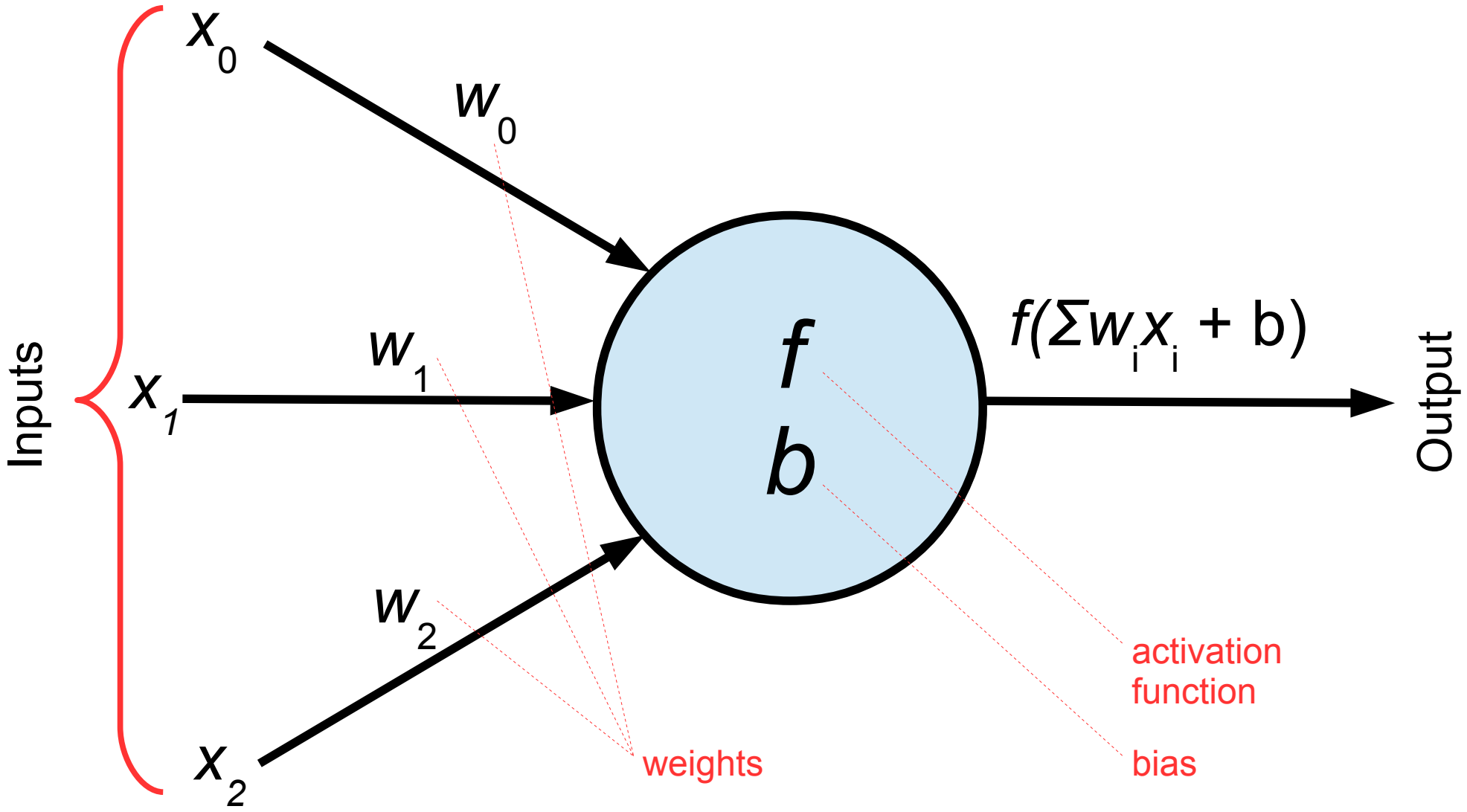
Neuron



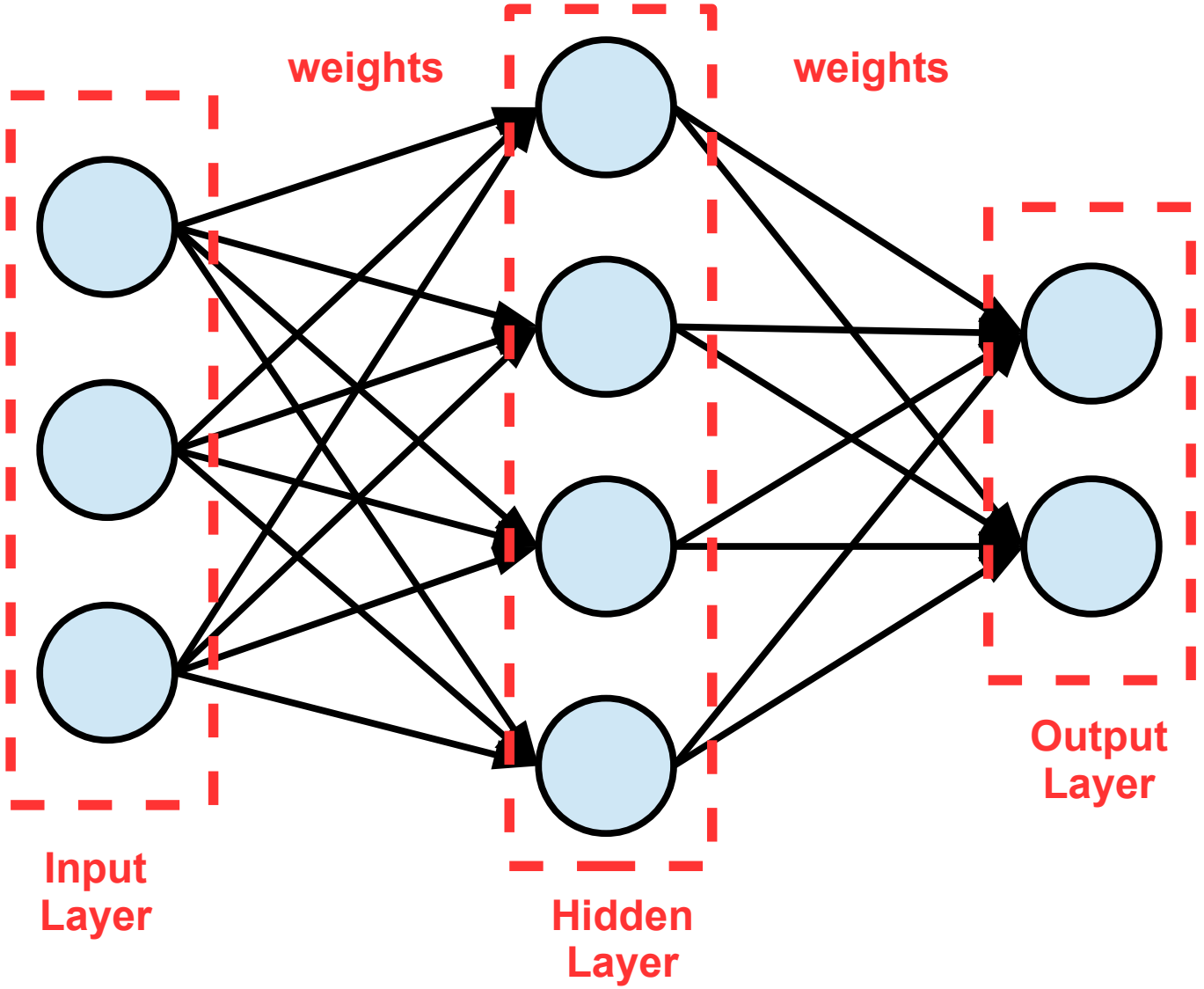
Neuron


x_0
w_0
x_1
w_1
w_2
x_2
Inputs
f
b
Output
activation function
weights
bias

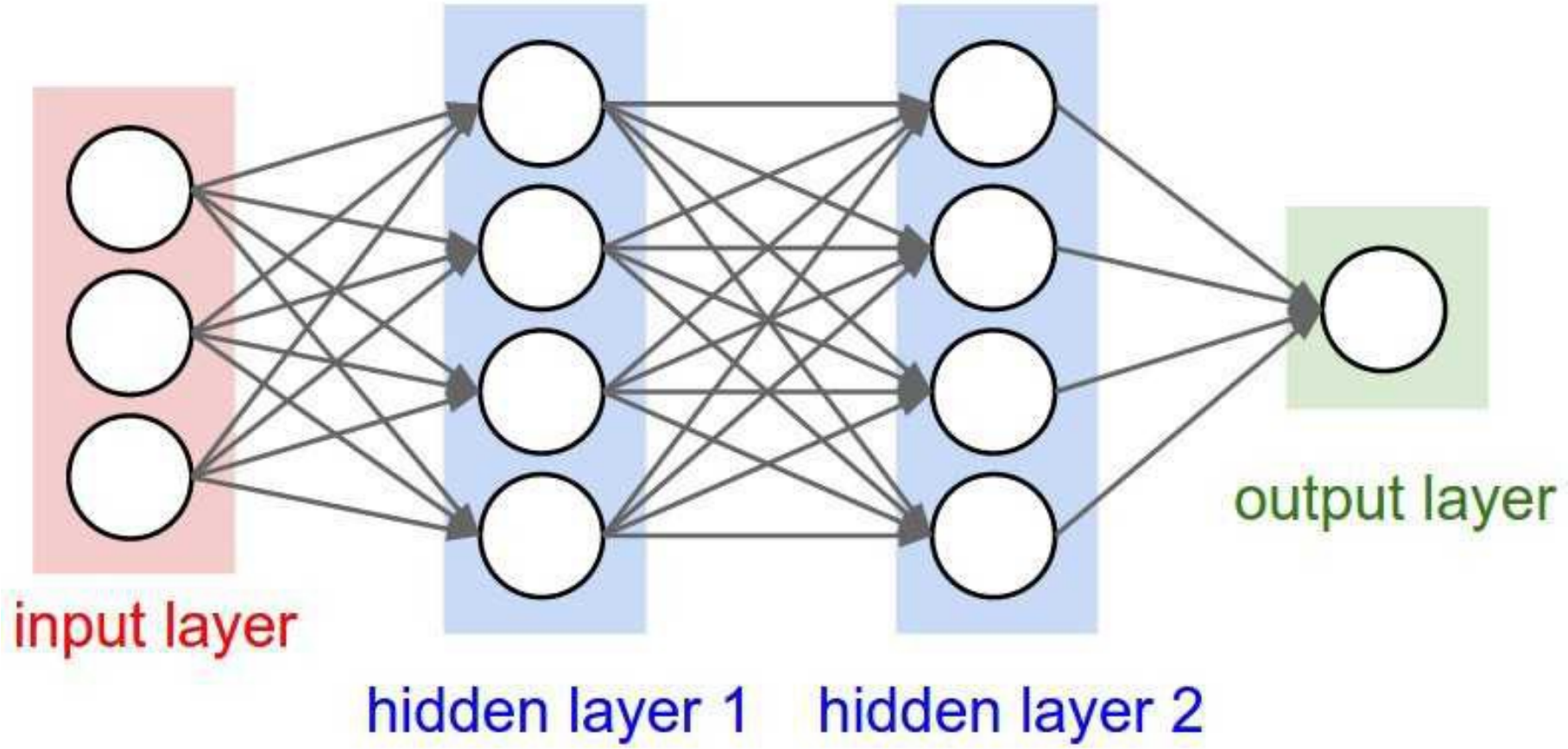
Neuron


Inputs
x_0
x_1
x_2
w_0
w_1
w_2
f
b
$f(\Sigma w_i x_i + b)$
Output
weights
activation function
bias

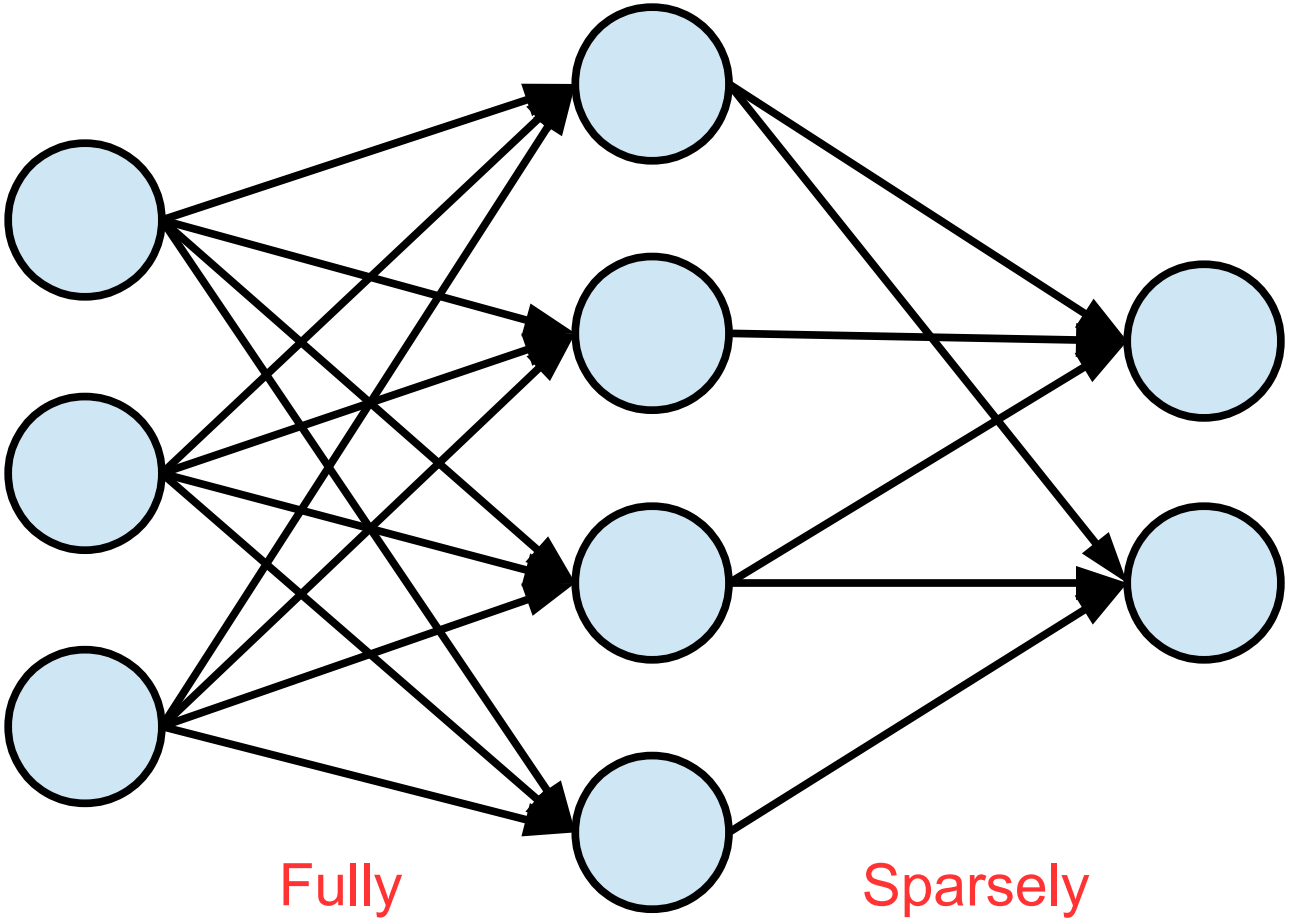
Terminology



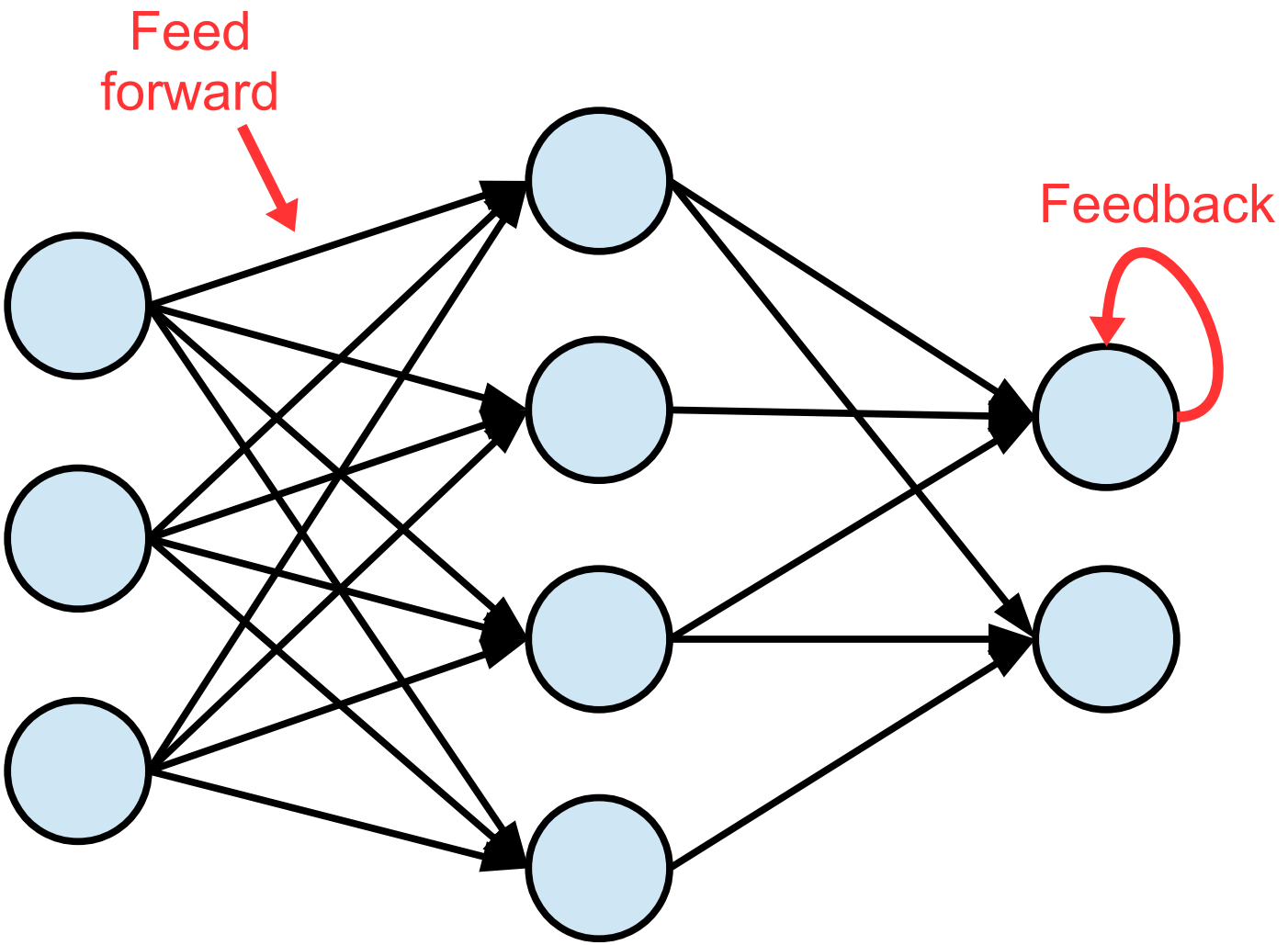
Terminology

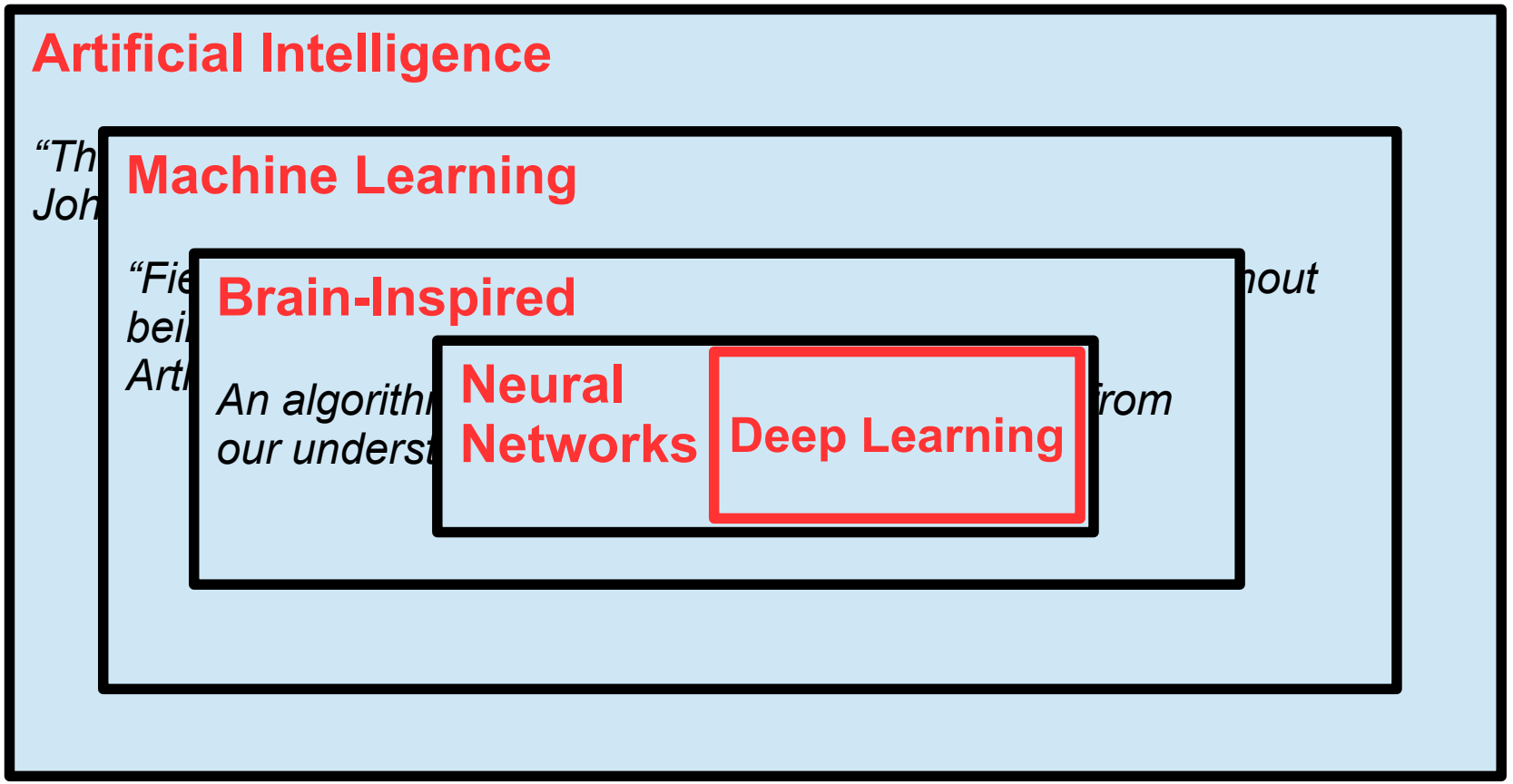


Terminology

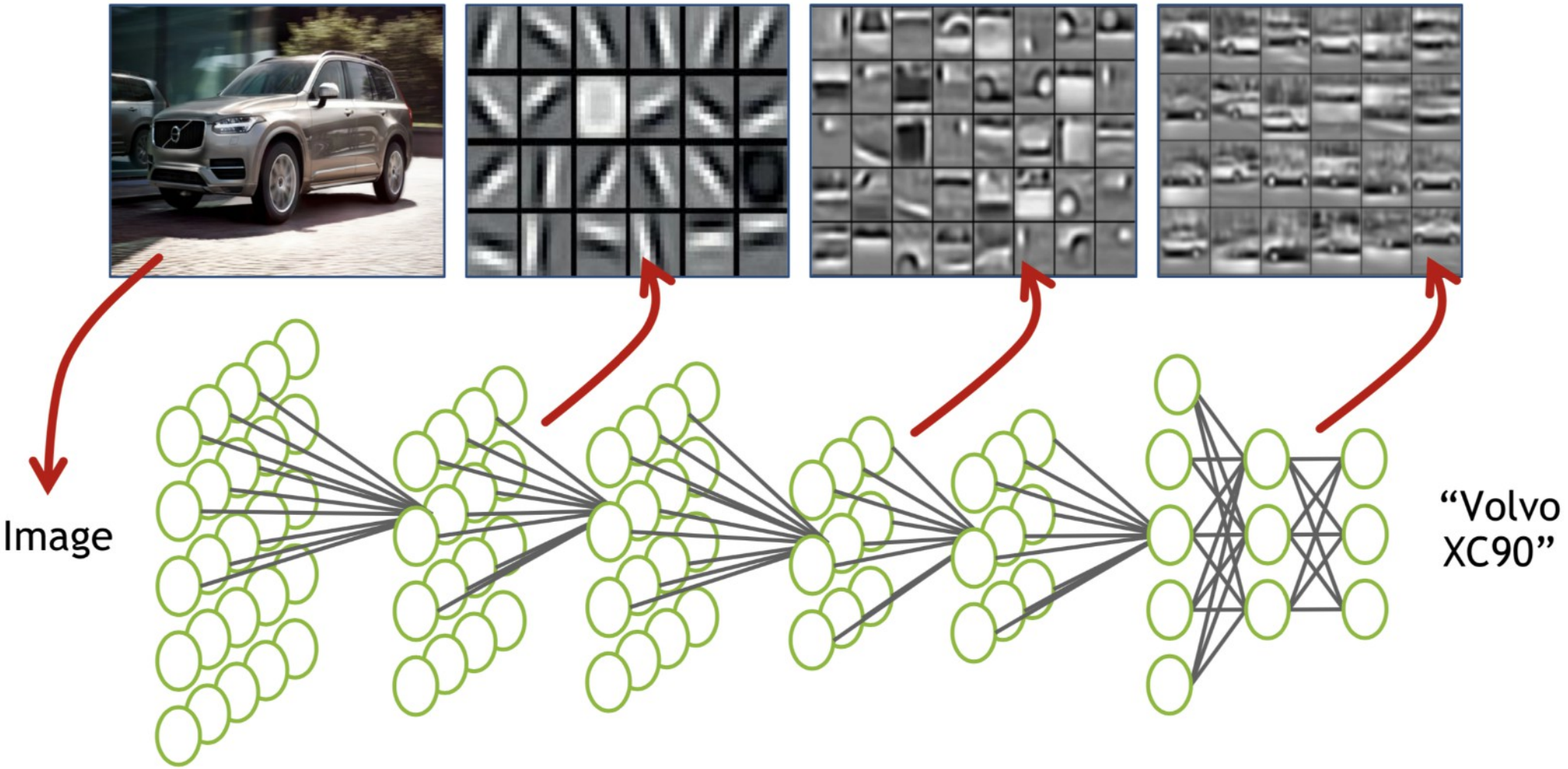


Terminology




Artificial Intelligence
Machine Learning
Brain-Inspired
Neural Networks
Deep Learning

Deep Neural Network



[H. Lee *et al.*, Unsupervised learning of hierarchical representations with convolutional deep belief networks. Commun. ACM 2011]

Types of DNNs

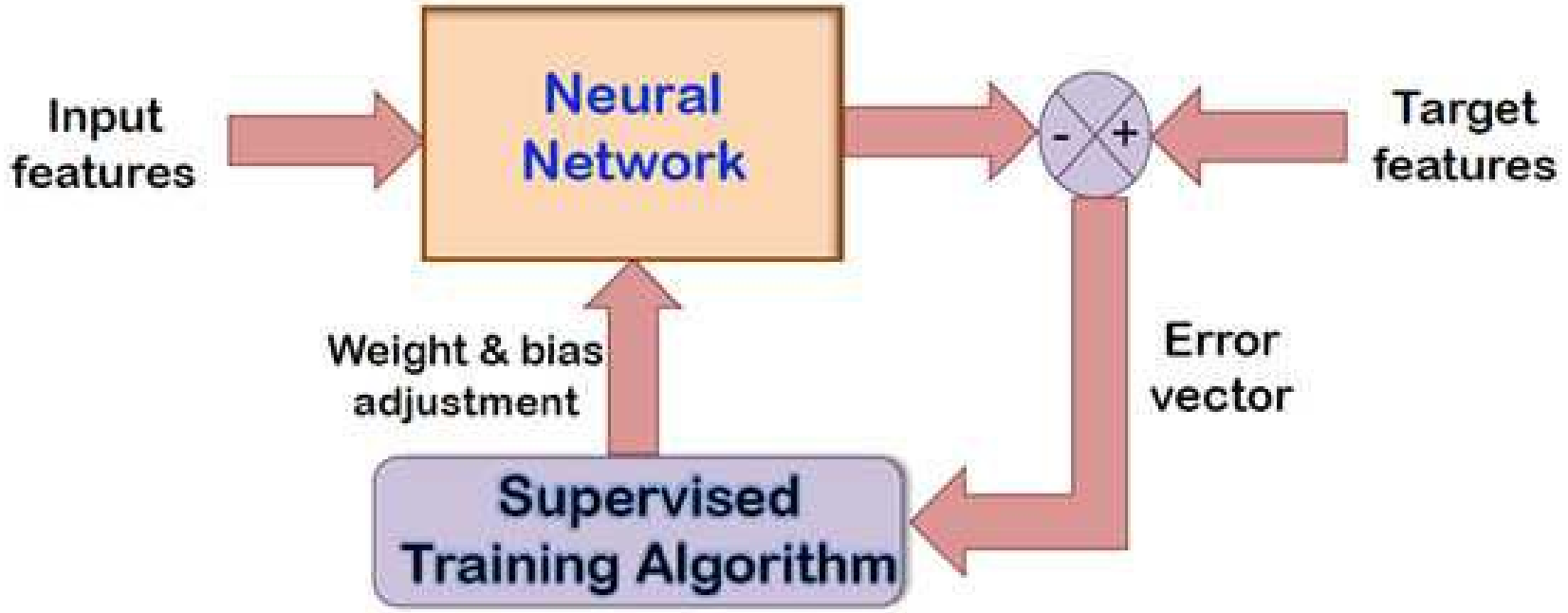
- Fully-Connected NN
  - feed forward, *a.k.a.* multilayer perceptron (MLP)
- Convolutional NN (CNN)
  - feed forward, sparsely-connected w/ weight sharing
- Recurrent NN (RNN)
  - feedback
- Long Short-Term Memory (LSTM)
  - feedback + storage

The Two Phases

- Training
- Inference

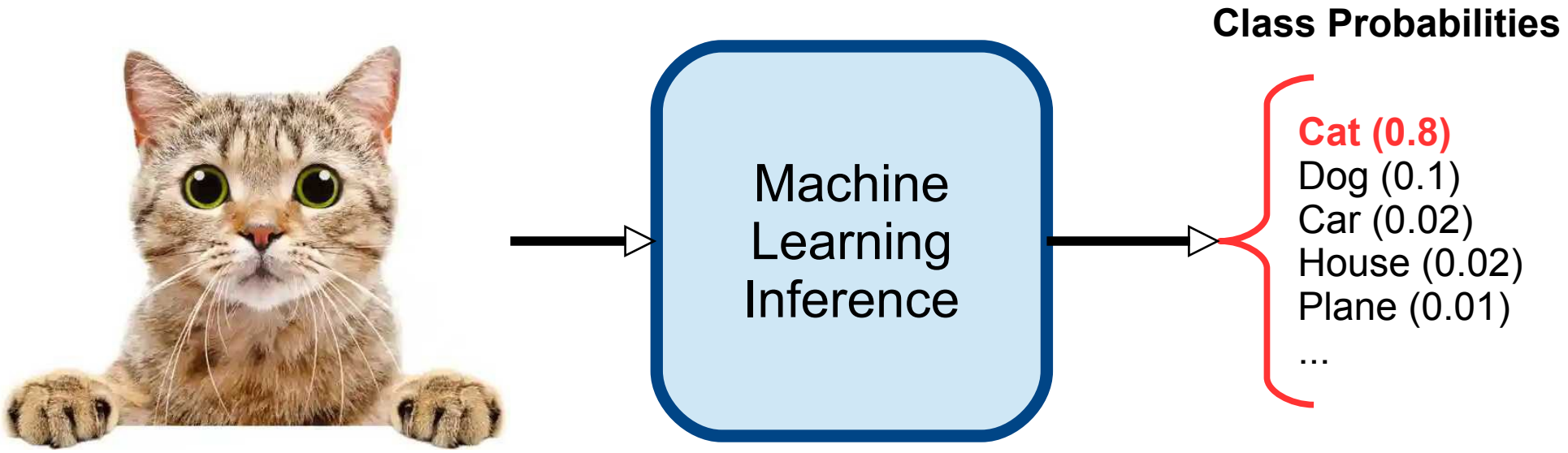
Training

- Determine weights (and biases)
  - Supervised
    - Training set has inputs and outputs, *i.e.*, labeled



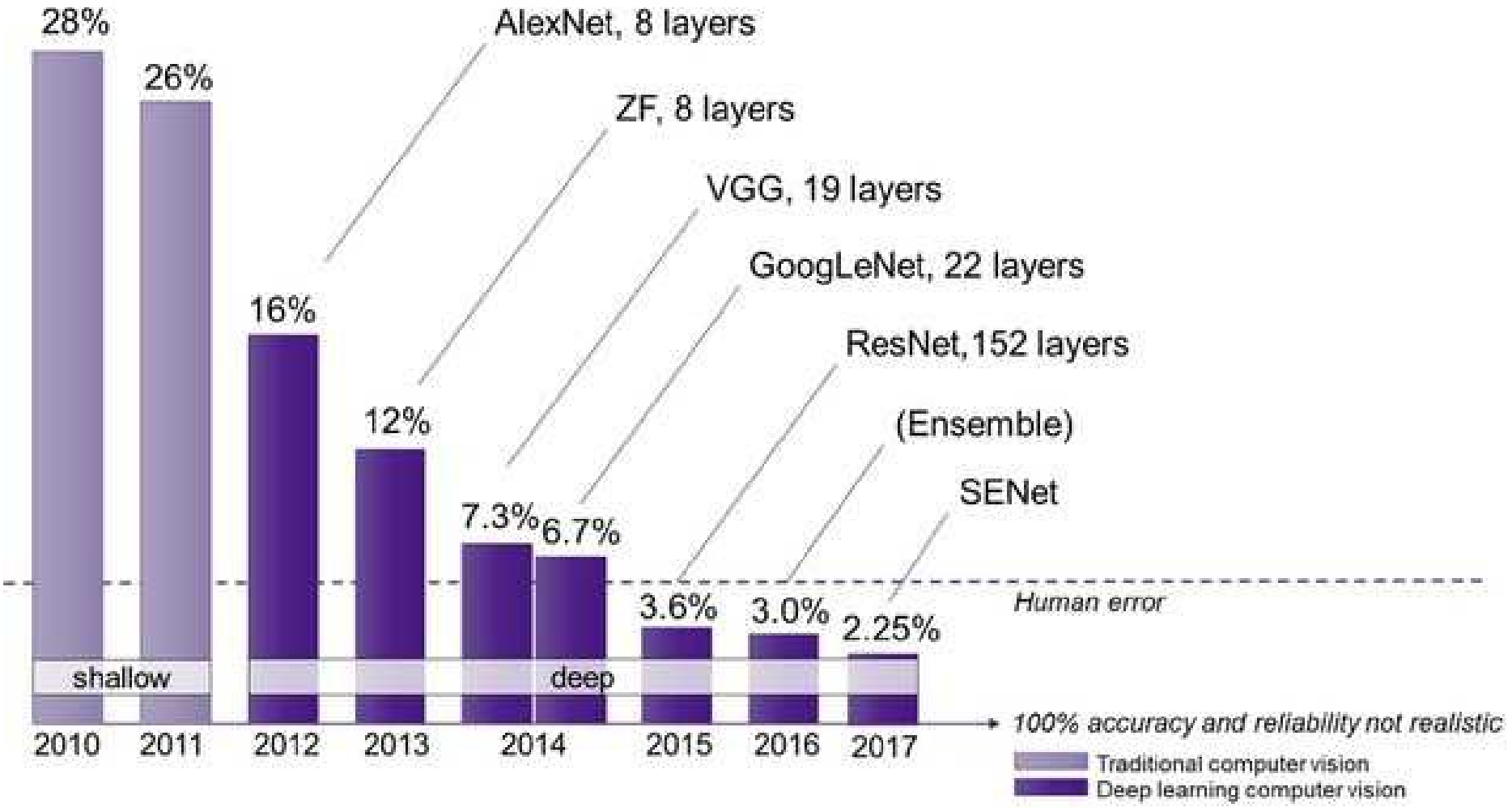
Inference

- Apply weights to determine output



ImageNet Challenge

- Training 1.2M images, 1K object categories



Applications of DNNs

- Image and Video
  - 70% to 80% of internet traffic is video (Cisco Visual Networking Index 2021)
  - 800 Mhours/day video surveillance in 2016
    - 7x increase in 2021 (VNI Complete Forecast Highlights)
  - DNN based CV to extract meaningful information
    - Image classification
    - Object localization and detection
    - Image segmentation
    - Action recognition

Applications of DNNs

- Speech and Language
  - Speech recognition
  - Machine translation
  - Natural language processing
  - Audio generation

Applications of DNNs

- Medicine and Health Care
- Game Play
- Robotics
- ...

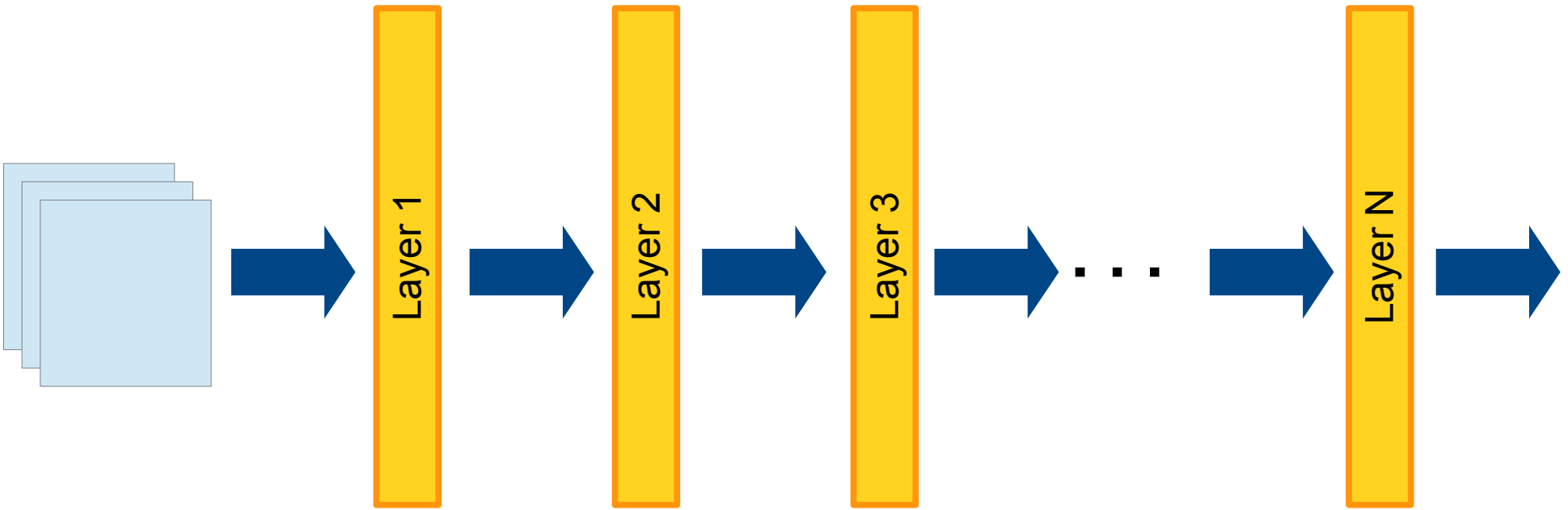
Embedded *vs.* Cloud

- Training
  - Typical performed in the cloud
- Inference
  - Trend is moving to the edge
    - Privacy/security
    - Latency
    - Communication energy

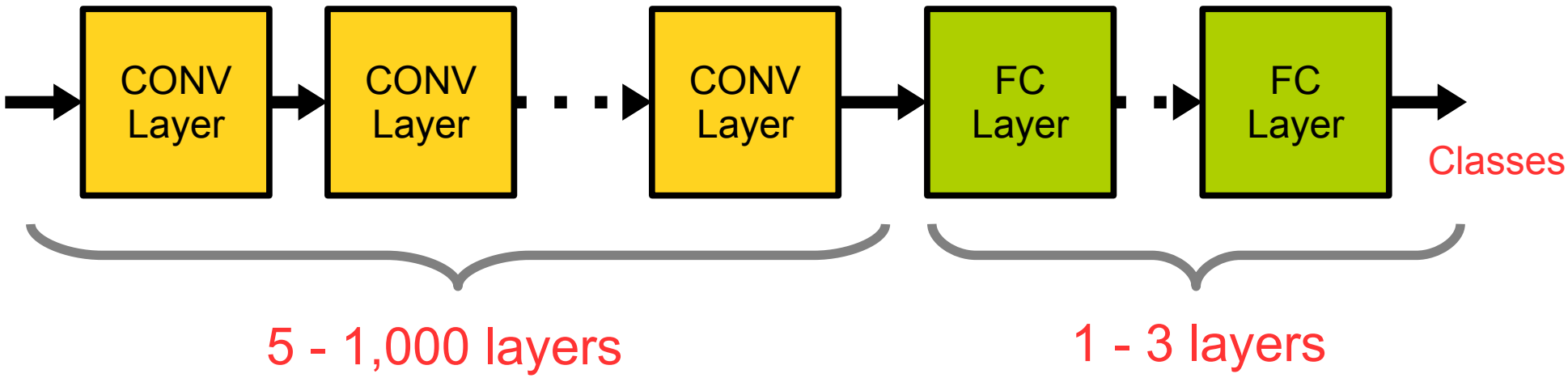
Agenda

- Deep Neural Networks
- Types of layers
- Memory and communication traffic
- Kernel computation
- Quantitative analysis

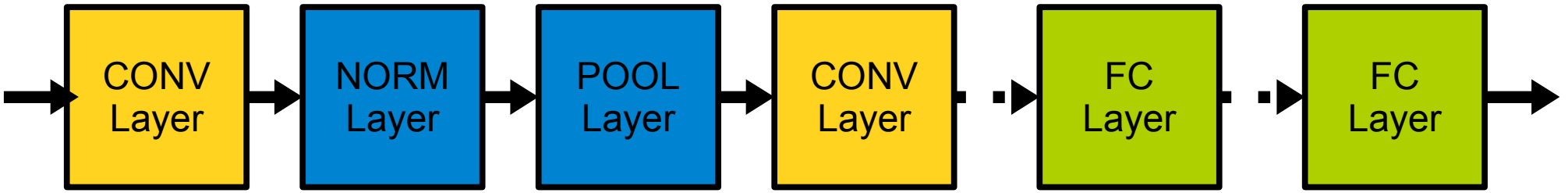
Deep Neural Network



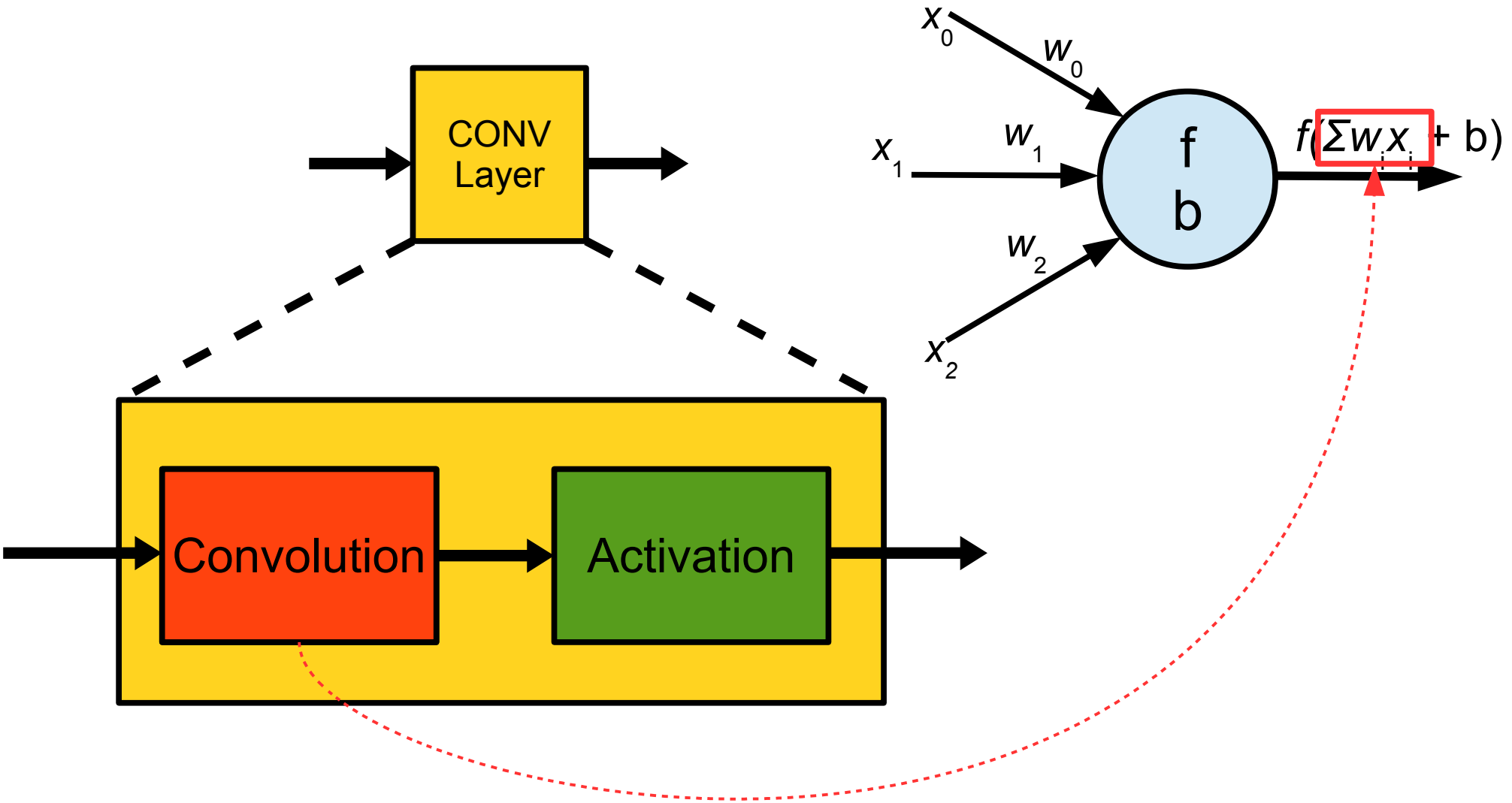
Deep Convolutional Neural Networks



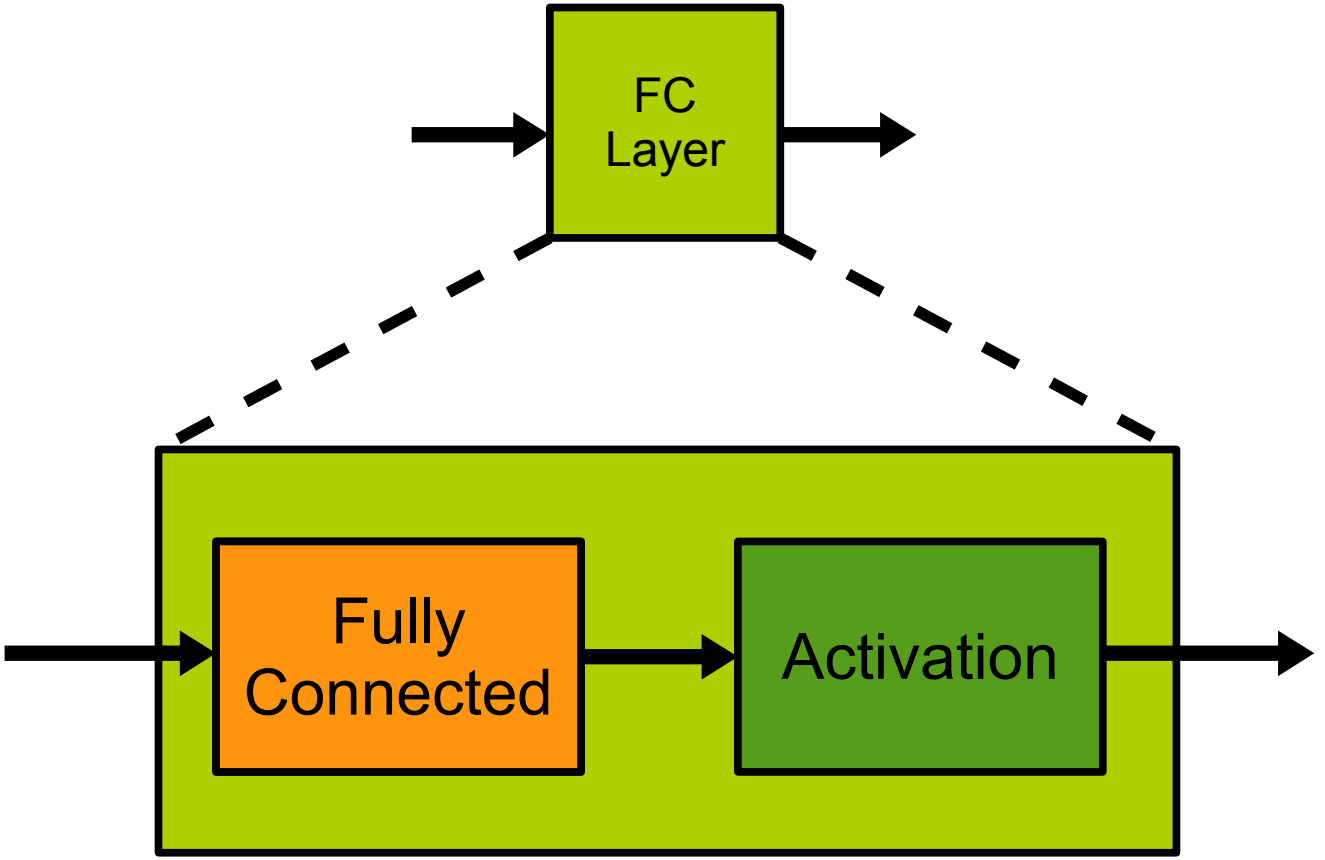
Deep Convolutional Neural Networks – Optional Layers



Deep Convolutional Neural Networks

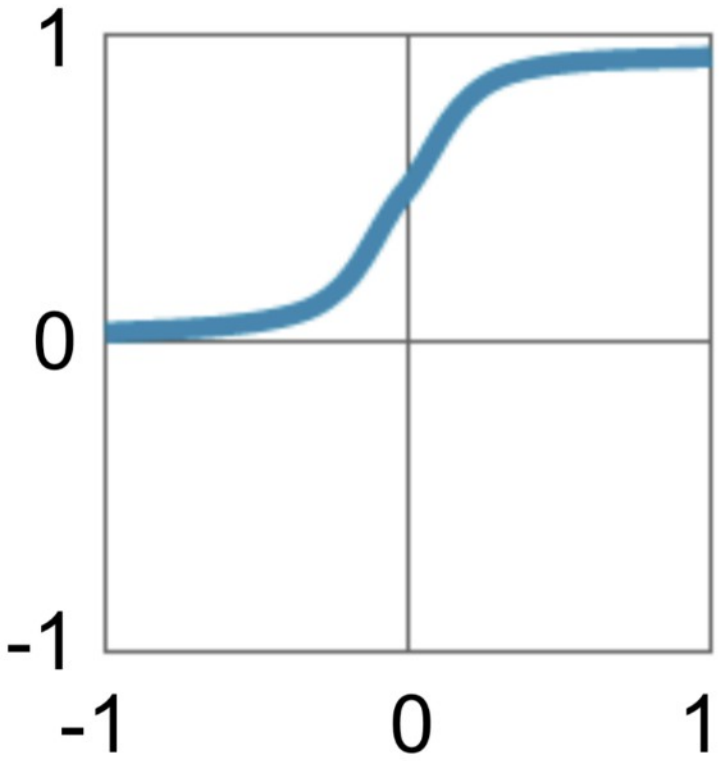


Deep Convolutional Neural Networks



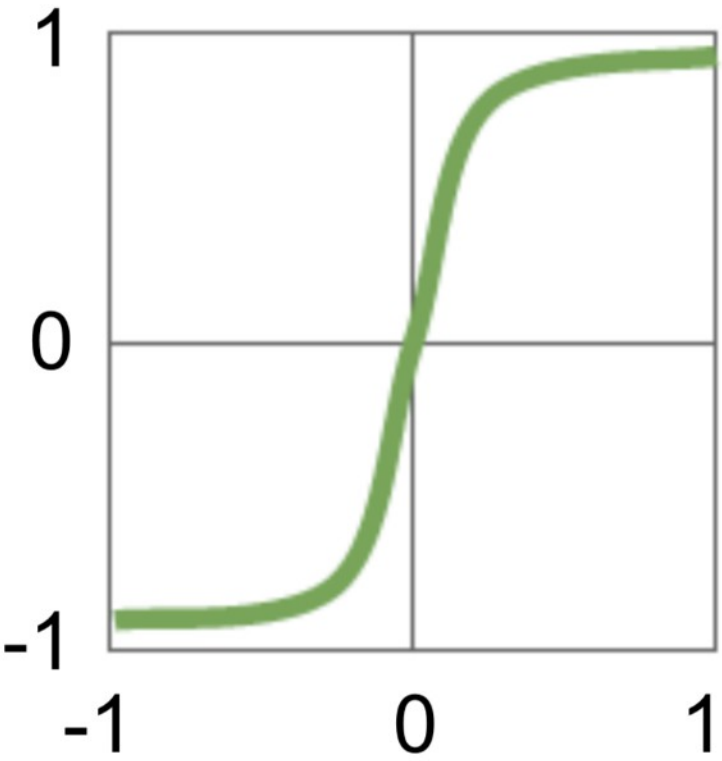
Activation -- Traditional

Sigmoid



$y=1/(1+e^{-x})$

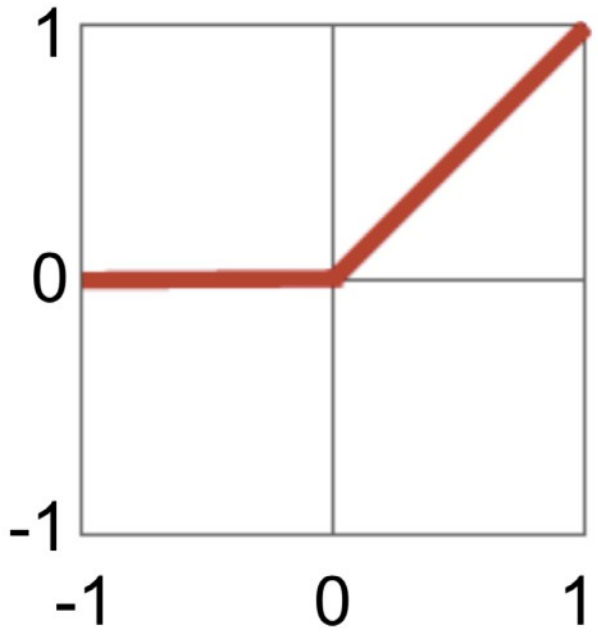
Hyperbolic Tangent



$y=(e^{x}-e^{-x})/(e^{x}+e^{-x})$

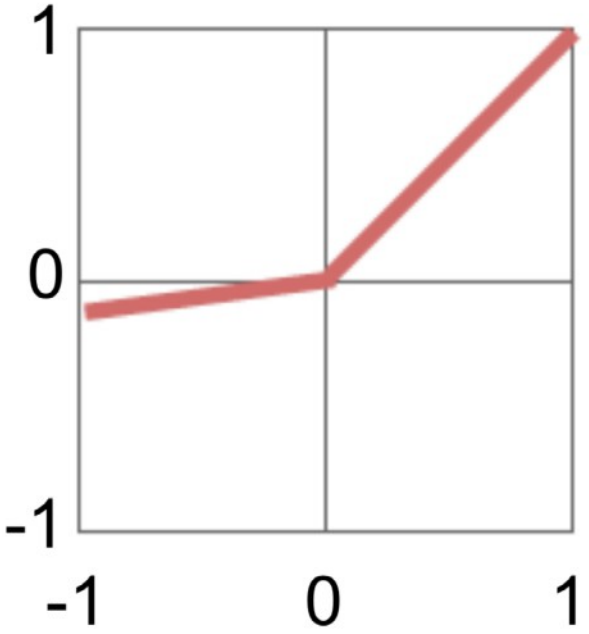
Activation Functions -- Modern

Rectified Linear Unit (ReLU)



$y=\max(0,x)$

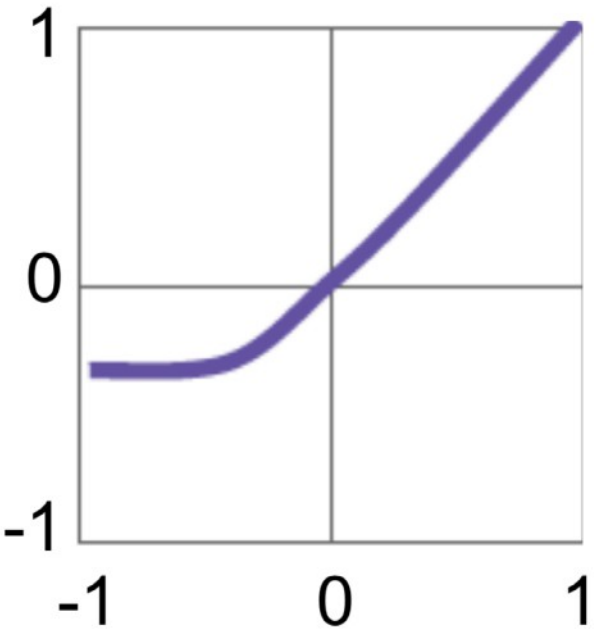
Leaky ReLU



$y=\max(\alpha x,x)$

α = small const. (e.g. 0.1)

Exponential LU

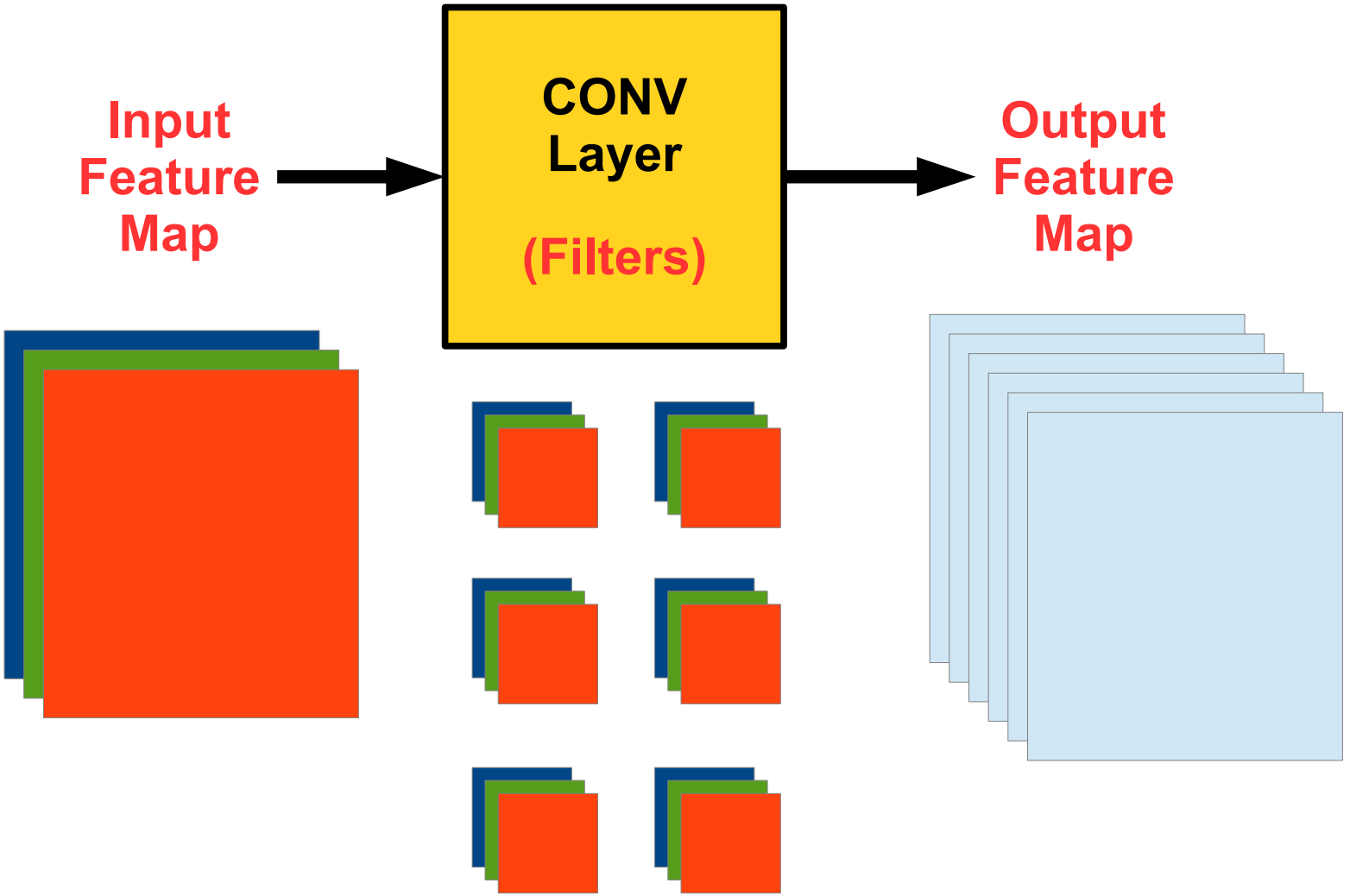


$$y=\begin{cases} x, & x\geq 0 \\ \alpha(e^{x}-1), & x<0 \end{cases}$$

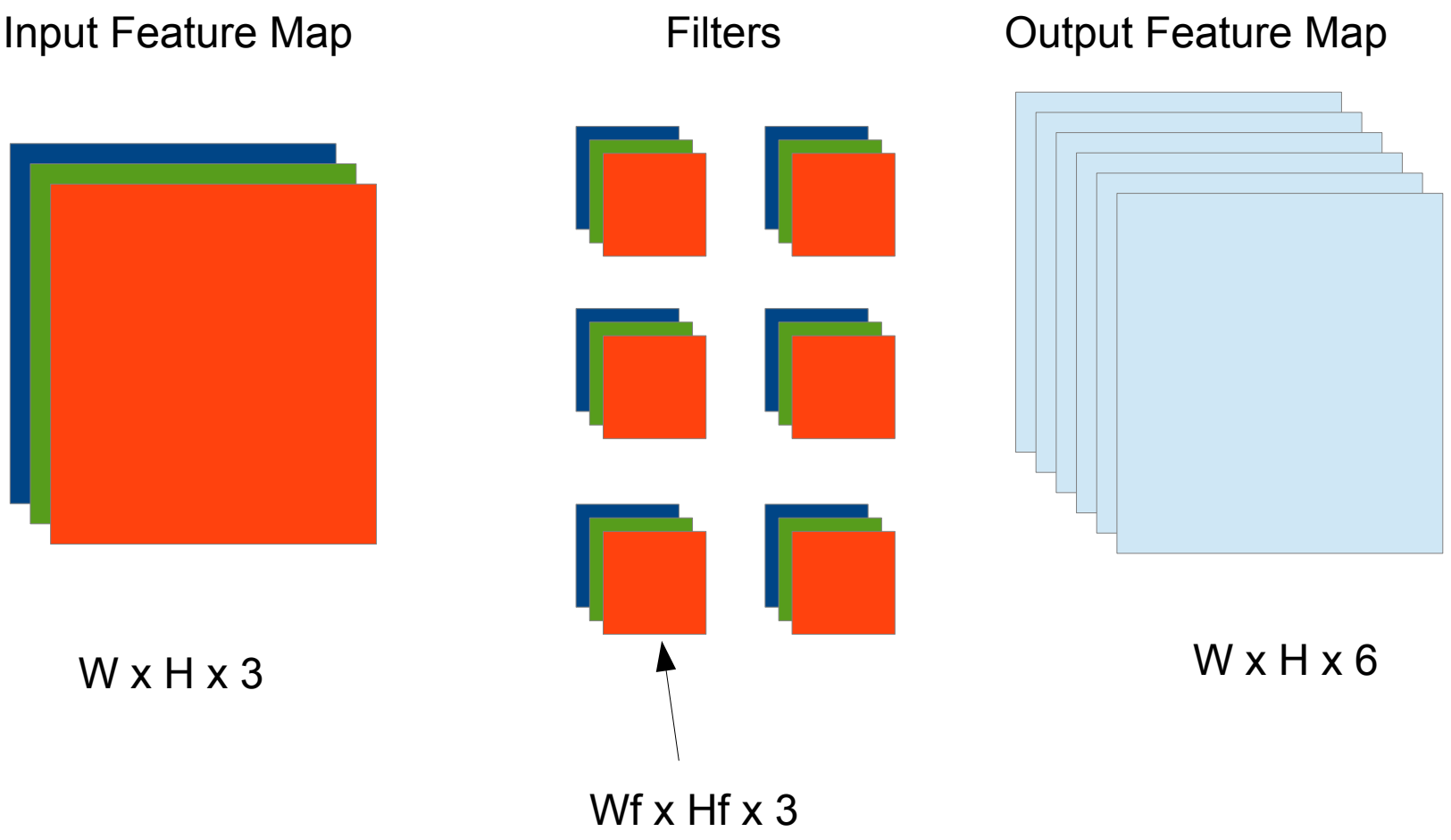
Convolution

- Convolutions account for more than 90% of overall computation
- Dominate runtime and energy consumption

Convolution

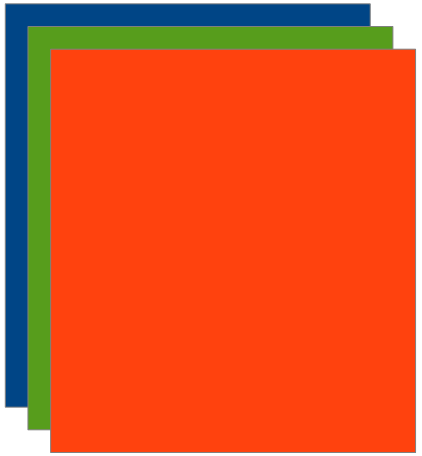


Convolution



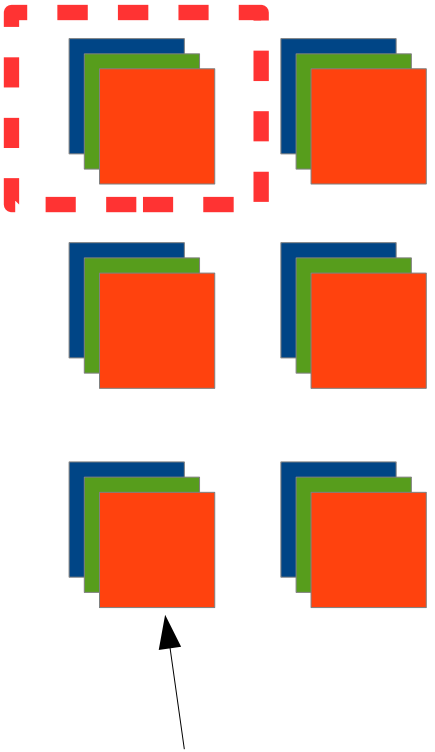
Convolution

Input Feature Map



W x H x 3

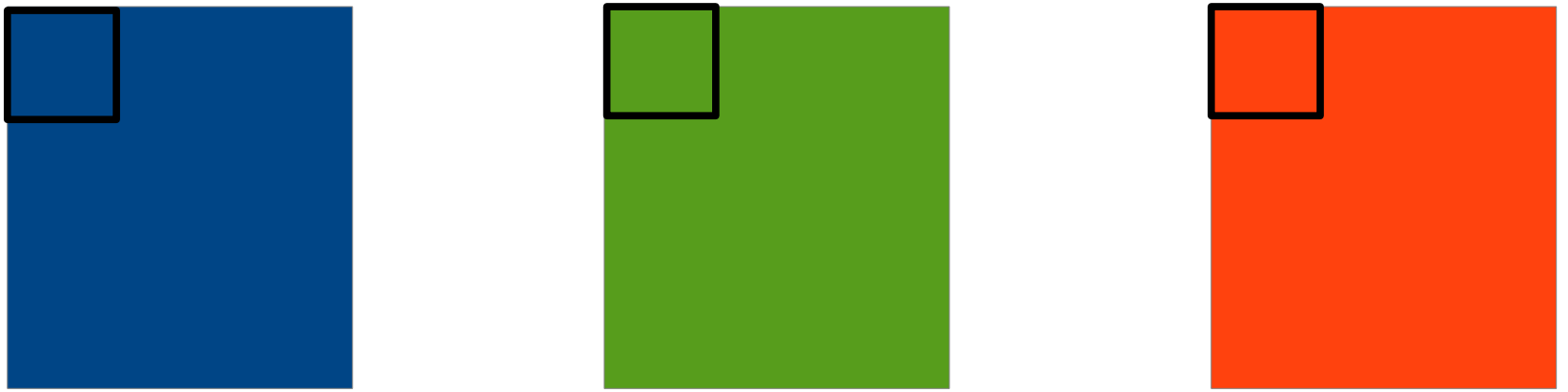
Filters



Wf x Hf x 3

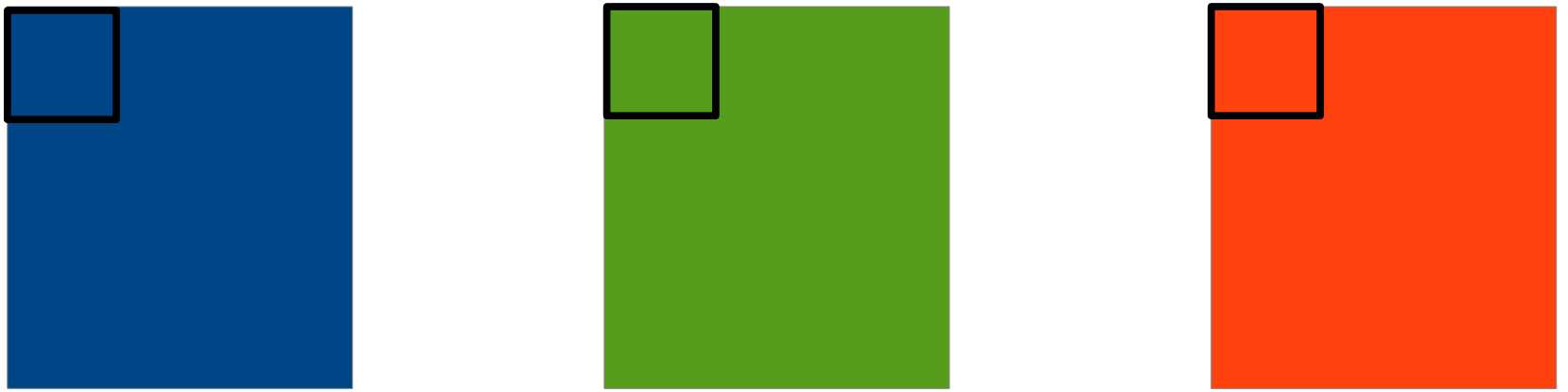
Convolution

Convolution



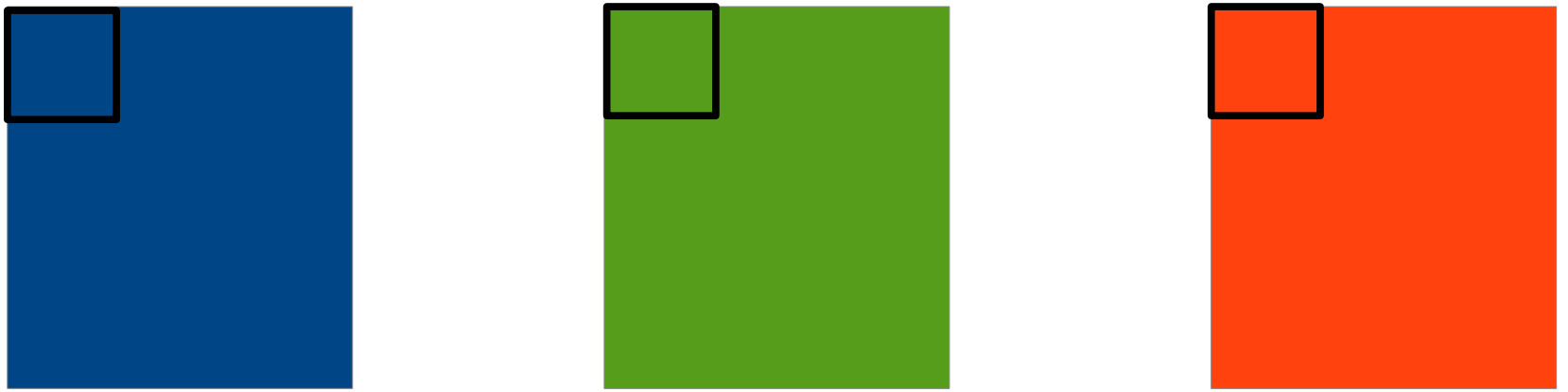
f1[0,0,■]*I[0,0,■] + f1[1,0,■]*I[1,0,■] + ... + f1[Wf-1,0,■]I[Wf-1,0,■]+
f1[0,1,■]*I[0,1,■] + f1[1,1,■]*I[1,1,■] + ... + f1[Wf-1,1,■]I[Wf-1,1,■]+
+ ... +
f1[0,Hf-1,■]*I[0,Hf-1,■] + f1[1,Hf-1,■]*I[1,Hf-1,■] + ... + f1[Wf-1,Hf-1,■]I[Wf-1,Hf-1,■]+

Convolution



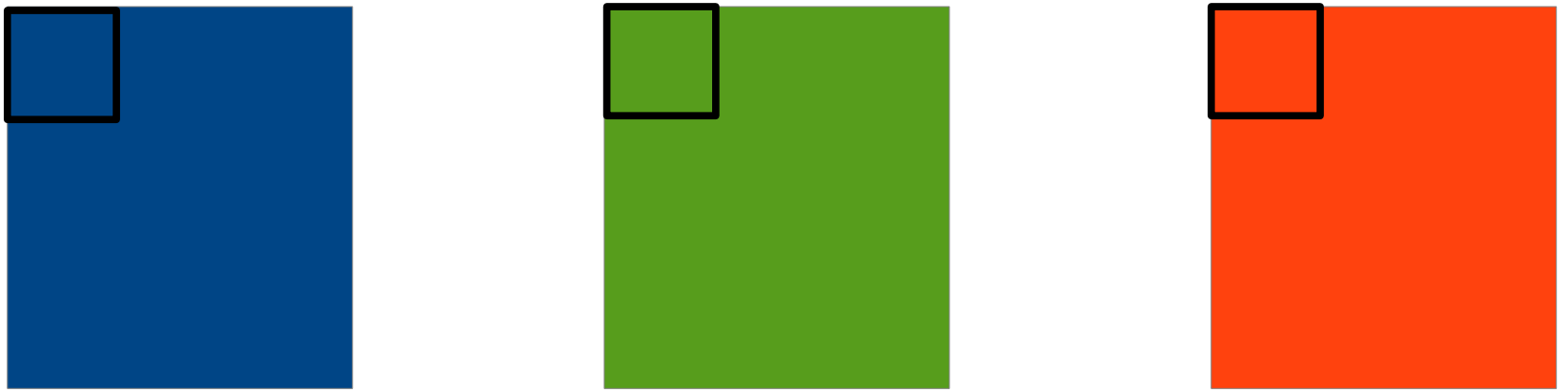
f1[0,0,■]*I[0,0,■] + f1[1,0,■]*I[1,0,■] + ... + f1[Wf-1,0,■]I[Wf-1,0,■]+
f1[0,1,■]*I[0,1,■] + f1[1,1,■]*I[1,1,■] + ... + f1[Wf-1,1,■]I[Wf-1,1,■]+
+ ... +
f1[0,Hf-1,■]*I[0,Hf-1,■] + f1[1,Hf-1,■]*I[1,Hf-1,■] + ... + f1[Wf-1,Hf-1,■]I[Wf-1,Hf-1,■]+
f1[0,0,■]*I[0,0,■] + f1[1,0,■]*I[1,0,■] + ... + f1[Wf-1,0,■]I[Wf-1,0,■]+
f1[0,1,■]*I[0,1,■] + f1[1,1,■]*I[1,1,■] + ... + f1[Wf-1,1,■]I[Wf-1,1,■]+
+ ... +
f1[0,Hf-1,■]*I[0,Hf-1,■] + f1[1,Hf-1,■]*I[1,Hf-1,■] + ... + f1[Wf-1,Hf-1,■]I[Wf-1,Hf-1,■]+

Convolution



f1[0,0,■]*I[0,0,■] + f1[1,0,■]*I[1,0,■] + ... + f1[Wf-1,0,■]I[Wf-1,0,■]+
f1[0,1,■]*I[0,1,■] + f1[1,1,■]*I[1,1,■] + ... + f1[Wf-1,1,■]I[Wf-1,1,■]+
+ ... +
f1[0,Hf-1,■]*I[0,Hf-1,■] + f1[1,Hf-1,■]*I[1,Hf-1,■] + ... + f1[Wf-1,Hf-1,■]I[Wf-1,Hf-1,■]+
f1[0,0,■]*I[0,0,■] + f1[1,0,■]*I[1,0,■] + ... + f1[Wf-1,0,■]I[Wf-1,0,■]+
f1[0,1,■]*I[0,1,■] + f1[1,1,■]*I[1,1,■] + ... + f1[Wf-1,1,■]I[Wf-1,1,■]+
+ ... +
f1[0,Hf-1,■]*I[0,Hf-1,■] + f1[1,Hf-1,■]*I[1,Hf-1,■] + ... + f1[Wf-1,Hf-1,■]I[Wf-1,Hf-1,■]+
f1[0,0,■]*I[0,0,■] + f1[1,0,■]*I[1,0,■] + ... + f1[Wf-1,0,■]I[Wf-1,0,■]+
f1[0,1,■]*I[0,1,■] + f1[1,1,■]*I[1,1,■] + ... + f1[Wf-1,1,■]I[Wf-1,1,■]+
+ ... +
f1[0,Hf-1,■]*I[0,Hf-1,■] + f1[1,Hf-1,■]*I[1,Hf-1,■] + ... + f1[Wf-1,Hf-1,■]I[Wf-1,Hf-1,■]+

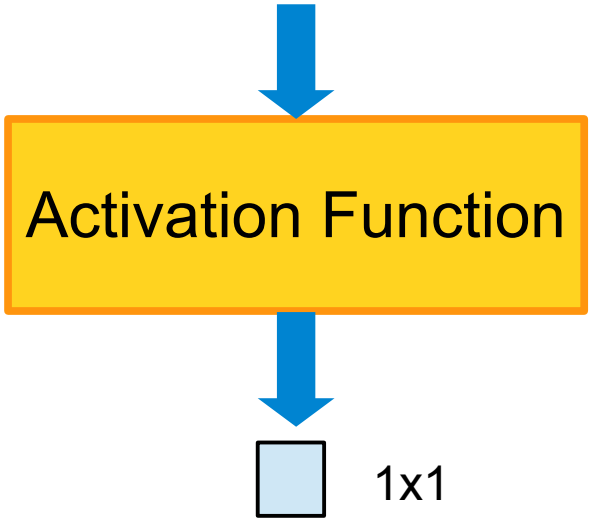
Convolution



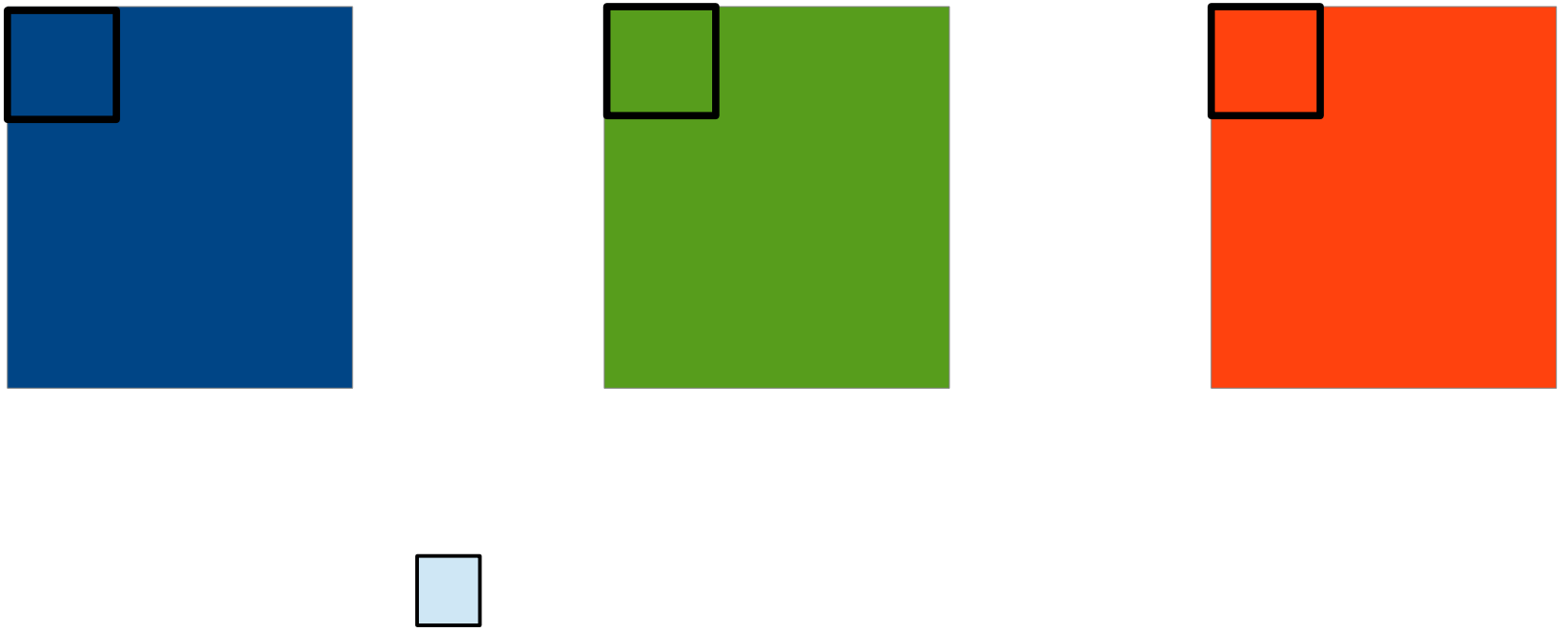
f1[0,0,■]*I[0,0,■] + f1[1,0,■]*I[1,0,■] + ... + f1[Wf-1,0,■]I[Wf-1,0,■]+
f1[0,1,■]*I[0,1,■] + f1[1,1,■]*I[1,1,■] + ... + f1[Wf-1,1,■]I[Wf-1,1,■]+
+ ... +
f1[0,Hf-1,■]*I[0,Hf-1,■] + f1[1,Hf-1,■]*I[1,Hf-1,■] + ... + f1[Wf-1,Hf-1,■]I[Wf-1,Hf-1,■]+
f1[0,0,■]*I[0,0,■] + f1[1,0,■]*I[1,0,■] + ... + f1[Wf-1,0,■]I[Wf-1,0,■]+
f1[0,1,■]*I[0,1,■] + f1[1,1,■]*I[1,1,■] + ... + f1[Wf-1,1,■]I[Wf-1,1,■]+
+ ... +
f1[0,Hf-1,■]*I[0,Hf-1,■] + f1[1,Hf-1,■]*I[1,Hf-1,■] + ... + f1[Wf-1,Hf-1,■]I[Wf-1,Hf-1,■]+
f1[0,0,■]*I[0,0,■] + f1[1,0,■]*I[1,0,■] + ... + f1[Wf-1,0,■]I[Wf-1,0,■]+
f1[0,1,■]*I[0,1,■] + f1[1,1,■]*I[1,1,■] + ... + f1[Wf-1,1,■]I[Wf-1,1,■]+
+ ... +
f1[0,Hf-1,■]*I[0,Hf-1,■] + f1[1,Hf-1,■]*I[1,Hf-1,■] + ... + f1[Wf-1,Hf-1,■]I[Wf-1,Hf-1,■]+ Bias1

Convolution

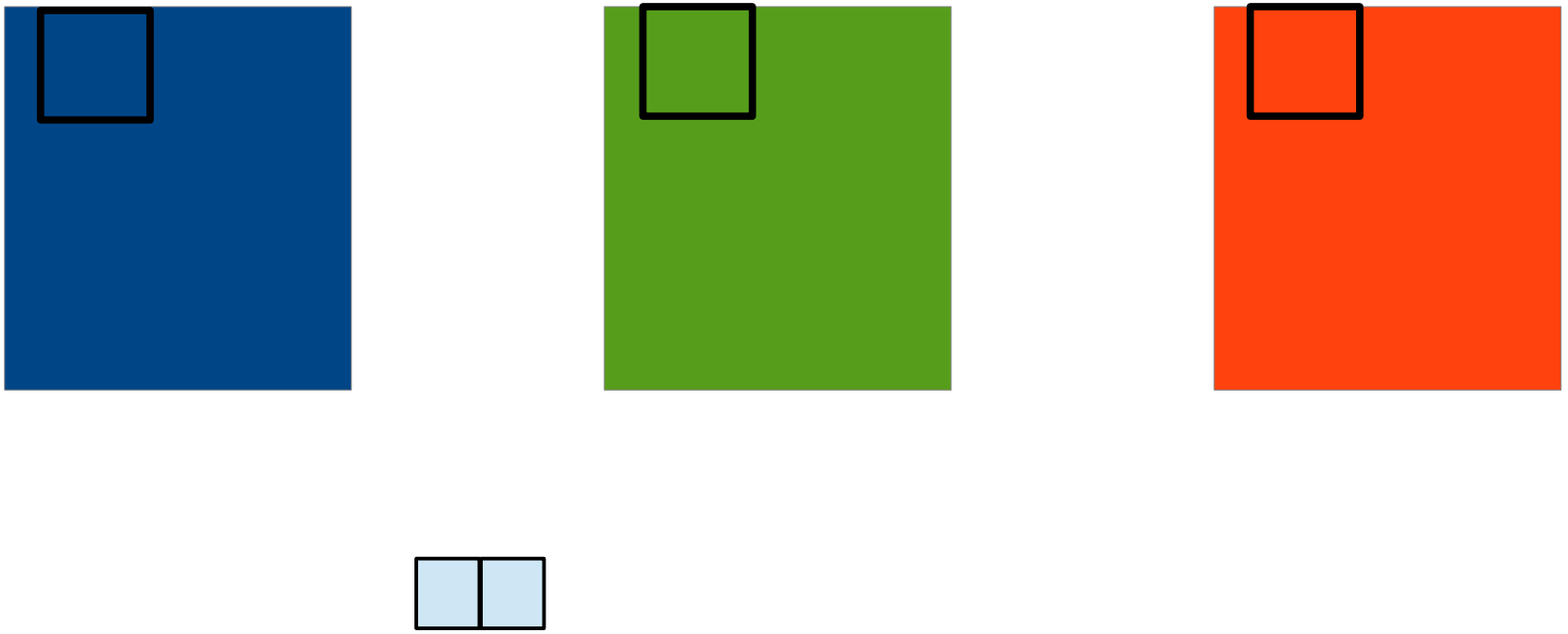
f1[0,0,■]*I[0,0,■] + f1[1,0,■]*I[1,0,■] + ... + f1[Wf-1,0,■]I[Wf-1,0,■]+
f1[0,1,■]*I[0,1,■] + f1[1,1,■]*I[1,1,■] + ... + f1[Wf-1,1,■]I[Wf-1,1,■]+
+ ... +
f1[0,Hf-1,■]*I[0,Hf-1,■] + f1[1,Hf-1,■]*I[1,Hf-1,■] + ... + f1[Wf-1,Hf-1,■]I[Wf-1,Hf-1,■]+
f1[0,0,■]*I[0,0,■] + f1[1,0,■]*I[1,0,■] + ... + f1[Wf-1,0,■]I[Wf-1,0,■]+
f1[0,1,■]*I[0,1,■] + f1[1,1,■]*I[1,1,■] + ... + f1[Wf-1,1,■]I[Wf-1,1,■]+
+ ... +
f1[0,Hf-1,■]*I[0,Hf-1,■] + f1[1,Hf-1,■]*I[1,Hf-1,■] + ... + f1[Wf-1,Hf-1,■]I[Wf-1,Hf-1,■]+
f1[0,0,■]*I[0,0,■] + f1[1,0,■]*I[1,0,■] + ... + f1[Wf-1,0,■]I[Wf-1,0,■]+
f1[0,1,■]*I[0,1,■] + f1[1,1,■]*I[1,1,■] + ... + f1[Wf-1,1,■]I[Wf-1,1,■]+
+ ... +
f1[0,Hf-1,■]*I[0,Hf-1,■] + f1[1,Hf-1,■]*I[1,Hf-1,■] + ... + f1[Wf-1,Hf-1,■]I[Wf-1,Hf-1,■]+ Bias1



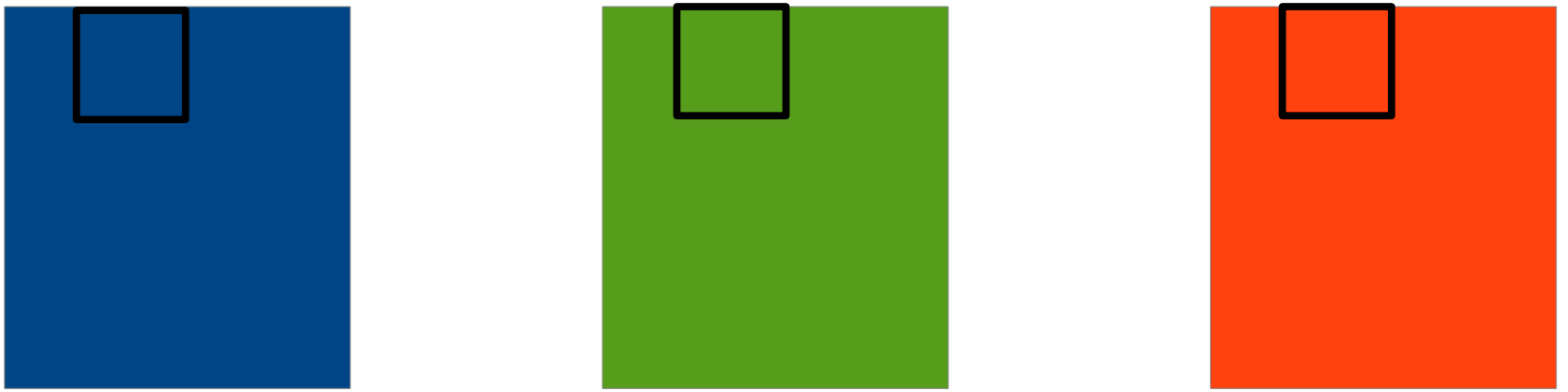
Convolution



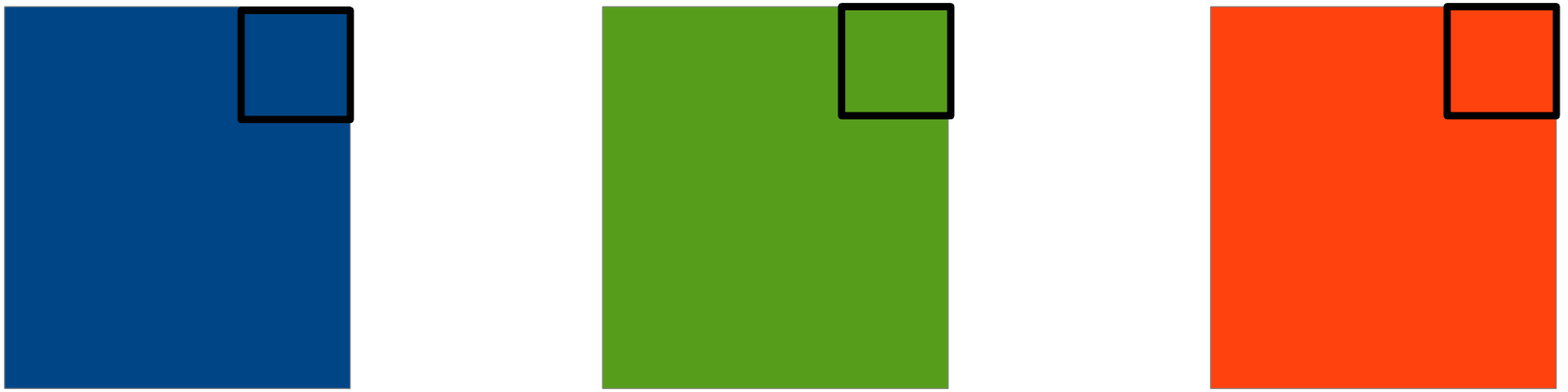
Convolution



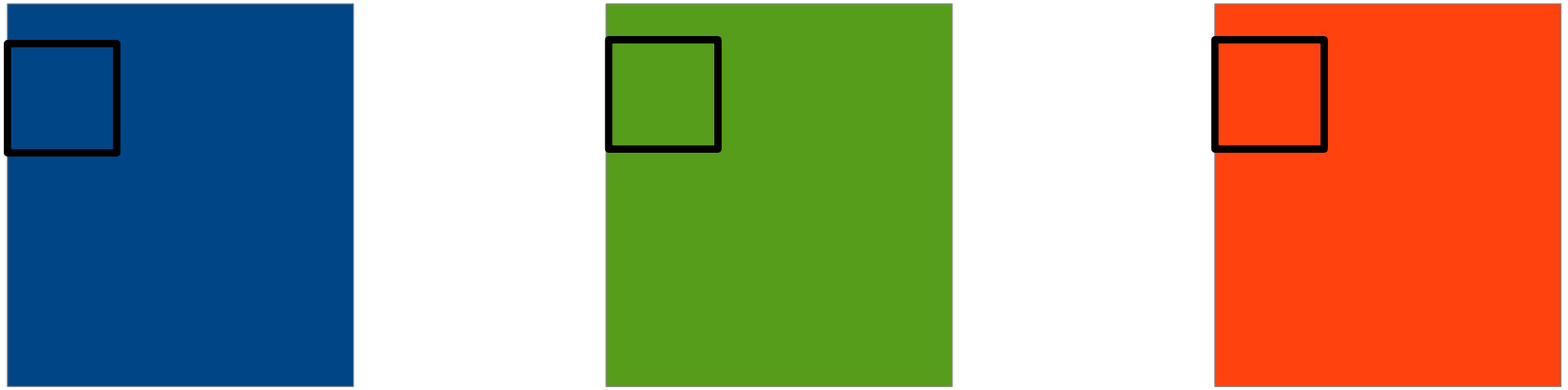
Convolution

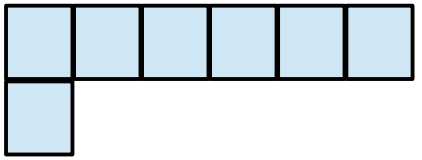


Convolution

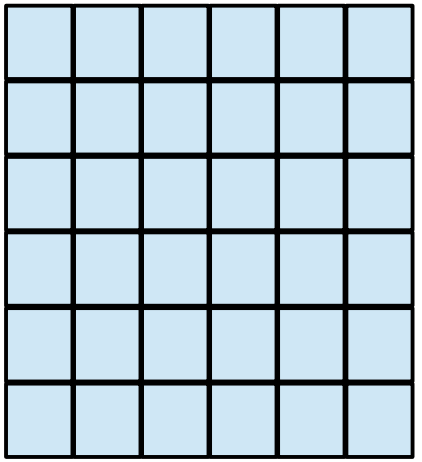


Convolution

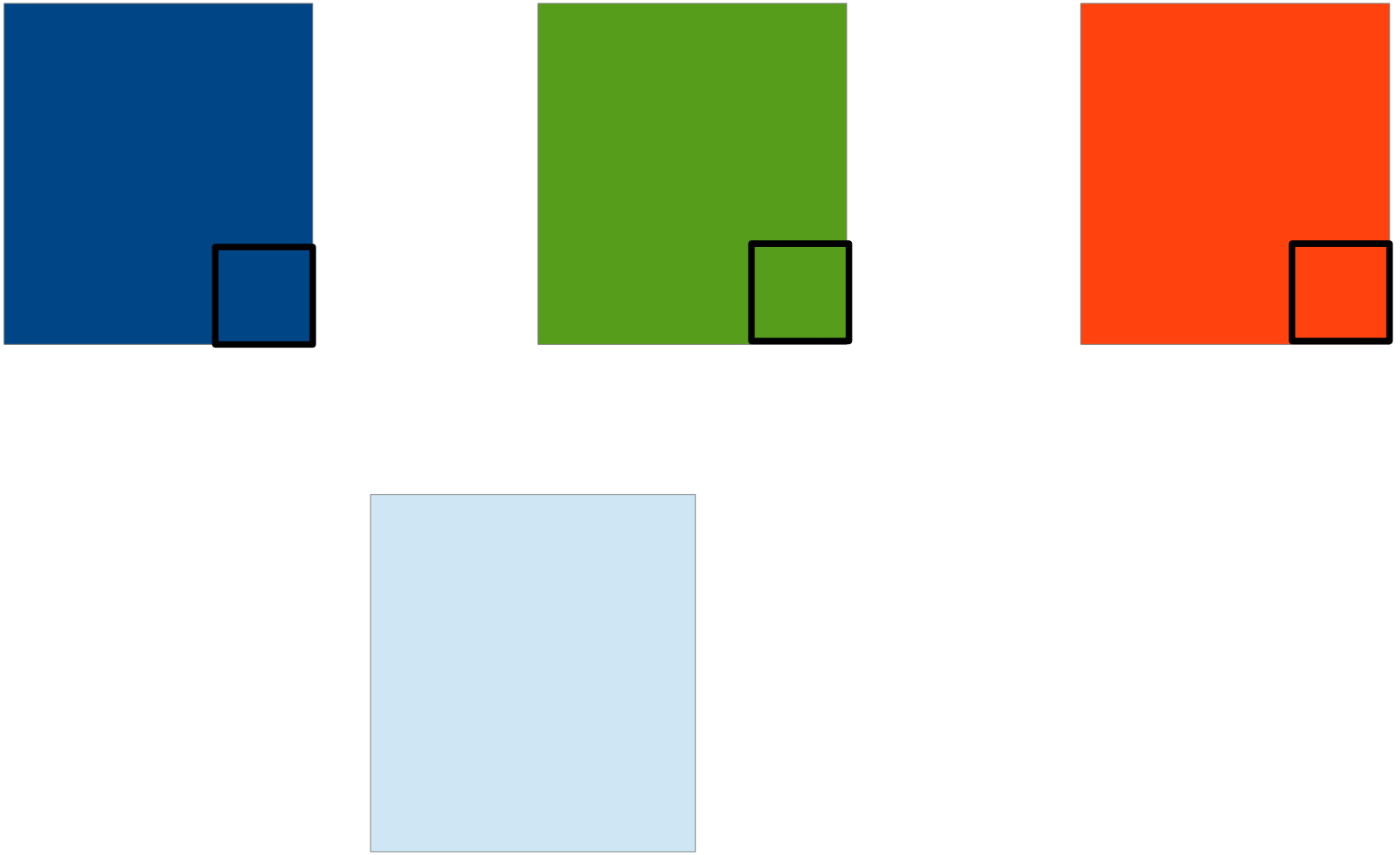




Convolution

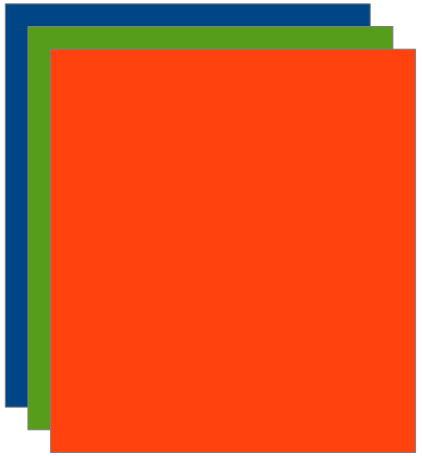


Convolution



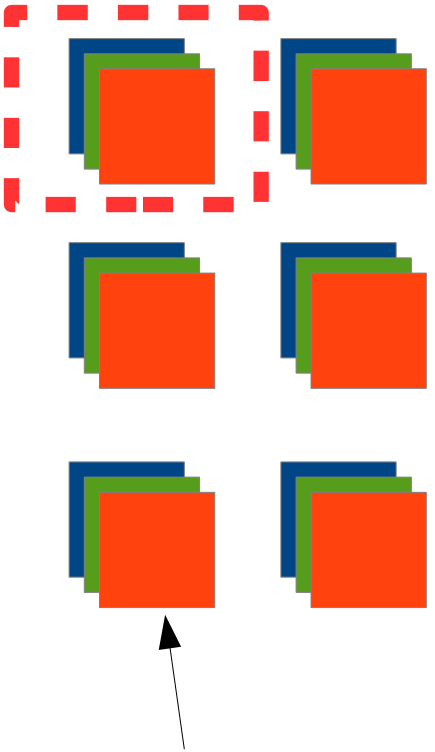
Convolution

Input Feature Map



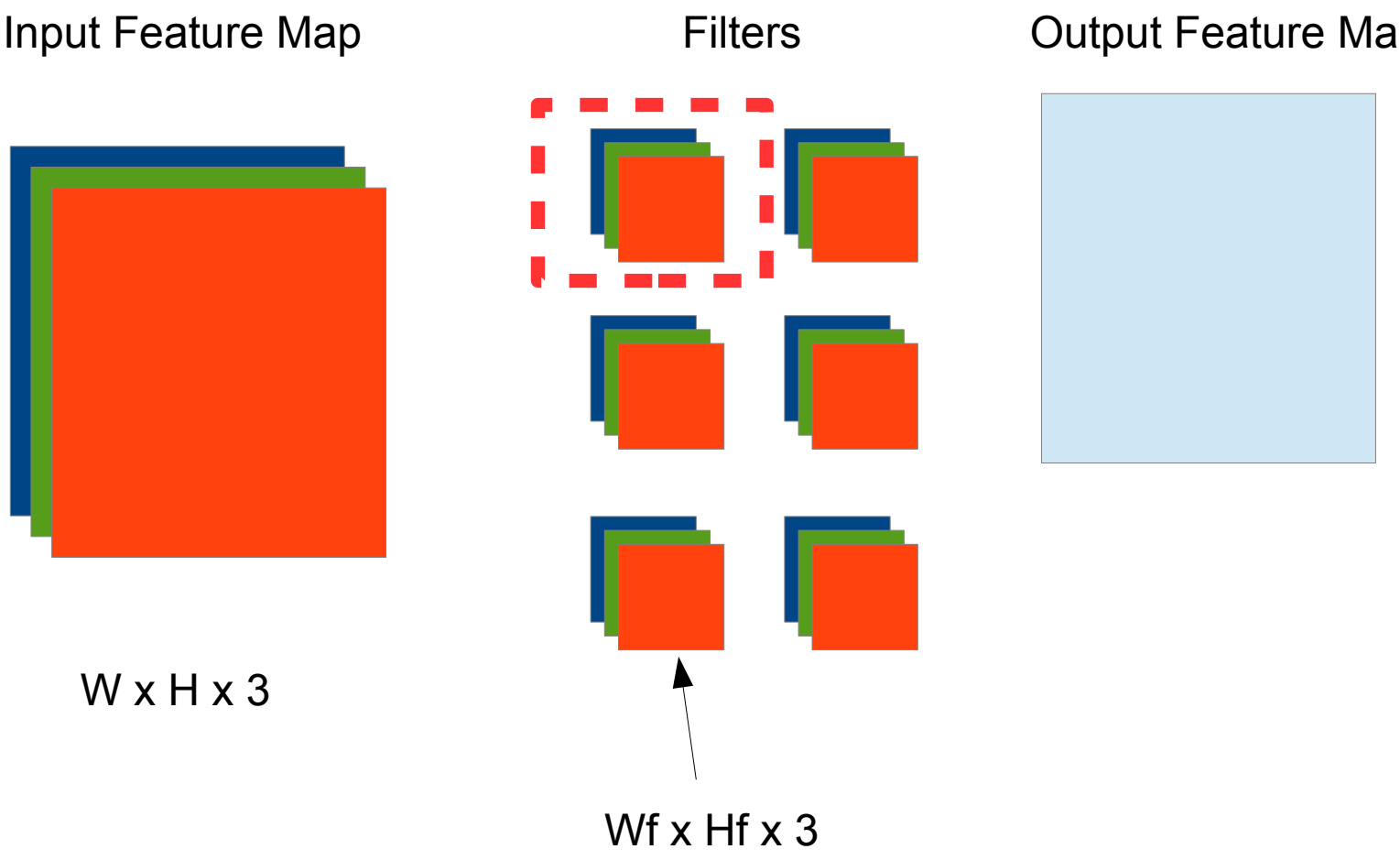
W x H x 3

Filters

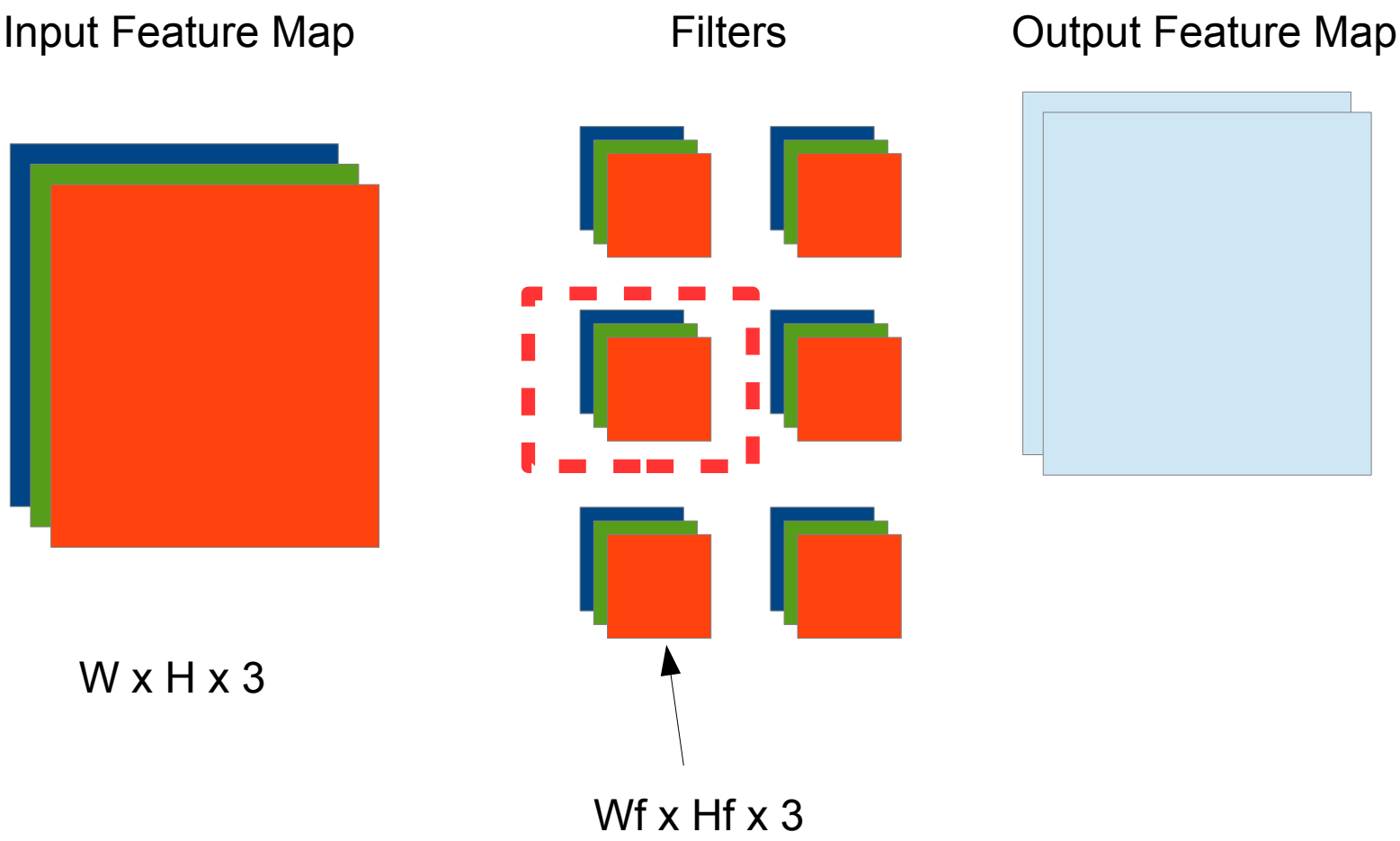


Wf x Hf x 3

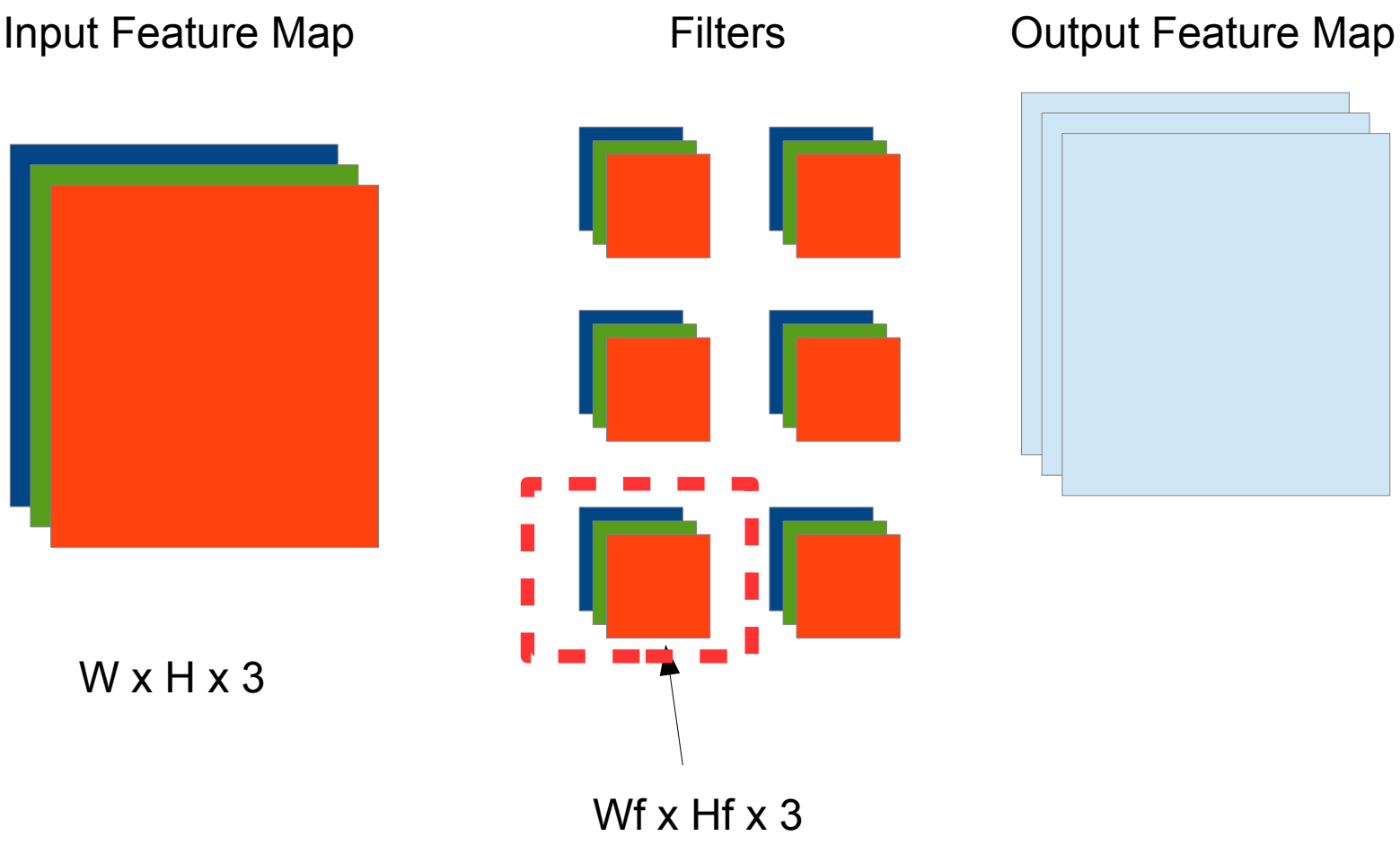
Convolution



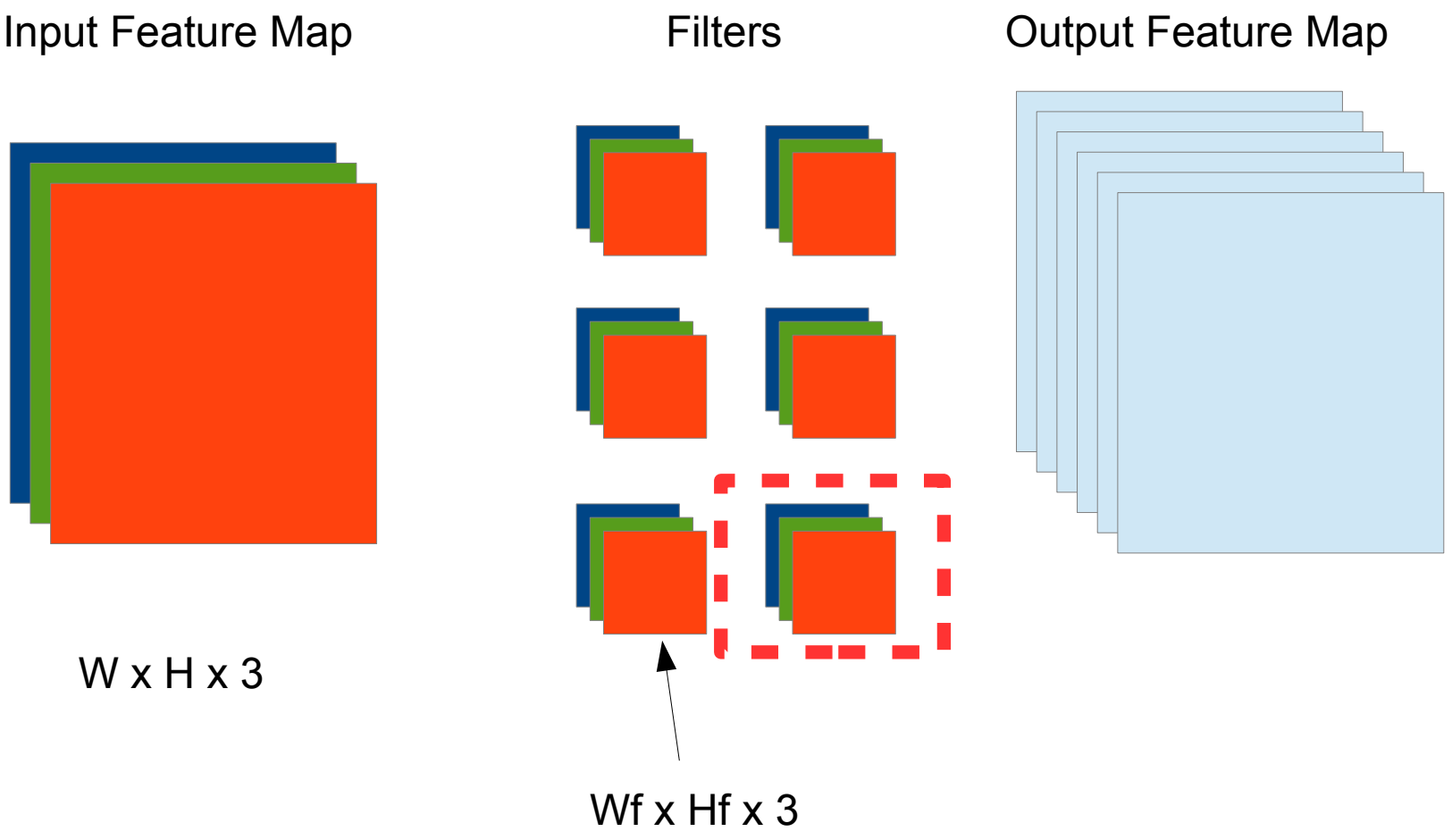
Convolution



Convolution

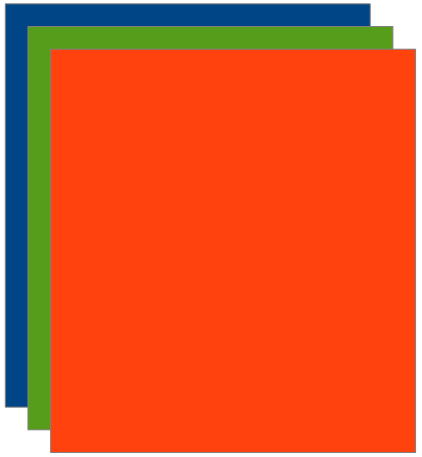


Convolution



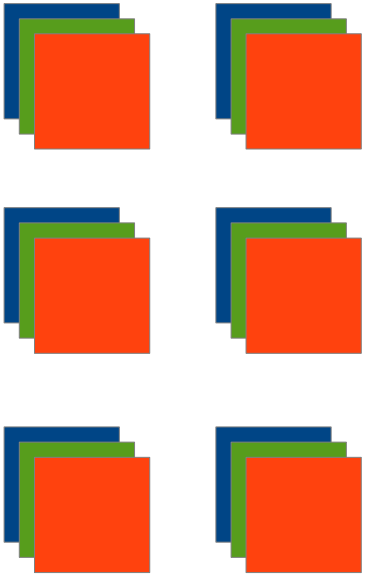
Convolution

Input Feature Map



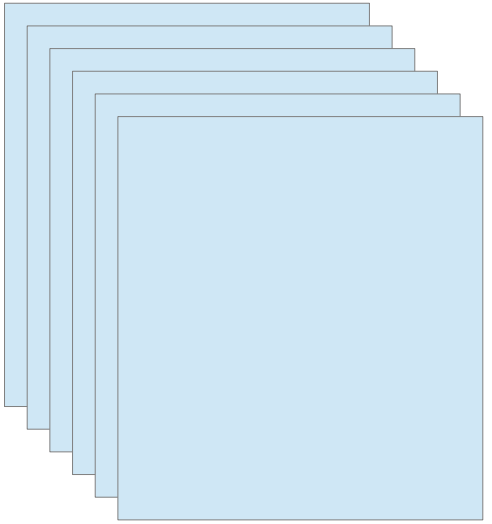
W x H x 3

Filters



Wf x Hf x 3

Output Feature Map



W x H x 6

CONV Layer Computation

Output fmaps (O)

Input fmaps (I)

Biases (B)

Filter weights (W)

$$\mathbf{O}[n][m][x][y] = \text{Activation}(\mathbf{B}[m] + \sum_{i=0}^{R-1}\sum_{j=0}^{S-1}\sum_{k=0}^{C-1} \mathbf{I}[n][k][Ux+i][Uy+j] \times \mathbf{W}[m][k][i][j]),$$

$$0 \le n < N, 0 \le m < M, 0 \le y < E, 0 \le x < F,$$

$$E = (H - R + U)/U, F = (W - S + U)/U.$$

Shape Parameter	Description
N	fmap batch size
M	# of filters / # of output fmap channels
C	# of input fmap/filter channels
H/W	input fmap height/width
R/S	filter height/width
E/F	output fmap height/width
U	convolution stride

CONV Layer Implementation

- Naïve 7-layer for-loop implementation

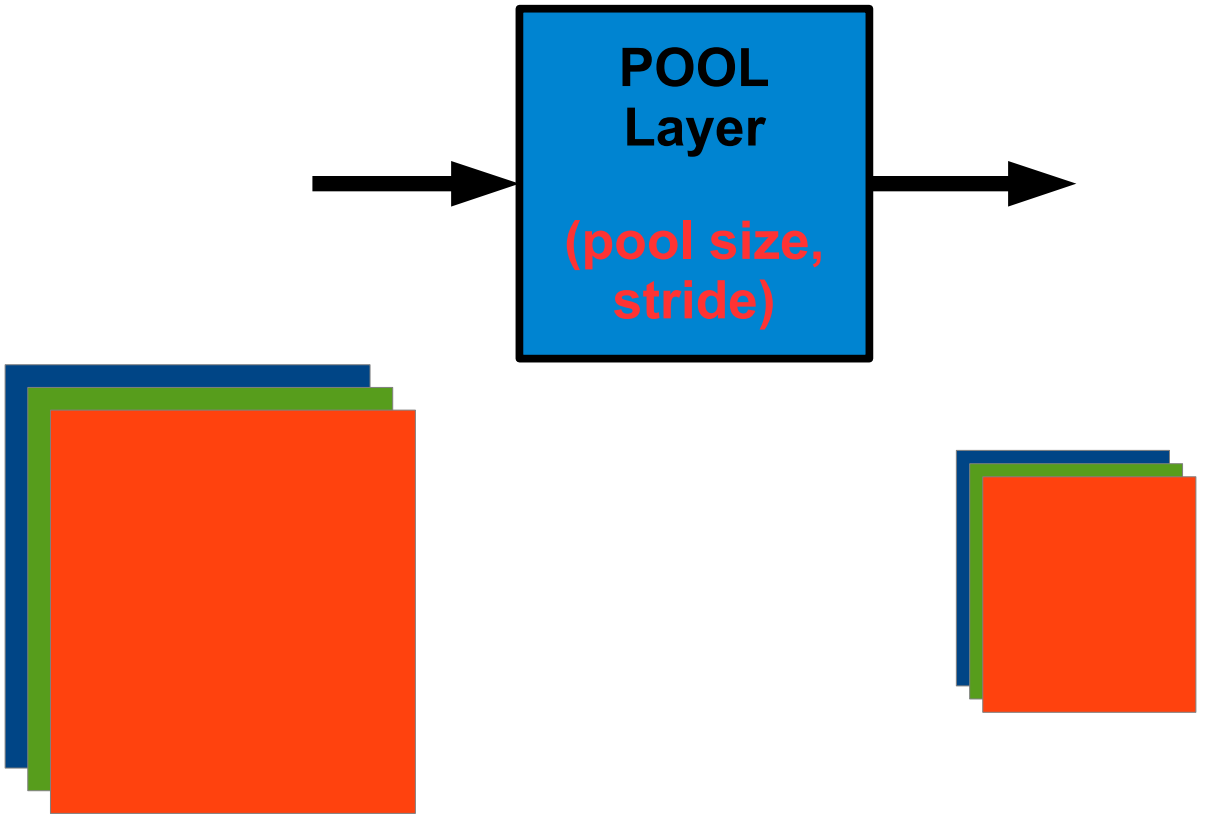
```
for (n=0; n<N; n++) {
    for (m=0; m<M; m++) {
        for (x=0; x<F; x++) {
            for (y=0; y<E; y++) {
                O[n][m][x][y] = B[m];
                for (i=0; i<R; i++) {
                    for (j=0; j<S; j++) {
                        for (k=0; k<C; k++) {
                            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]);
            }
        }
    }
}
```

for each output fmap value

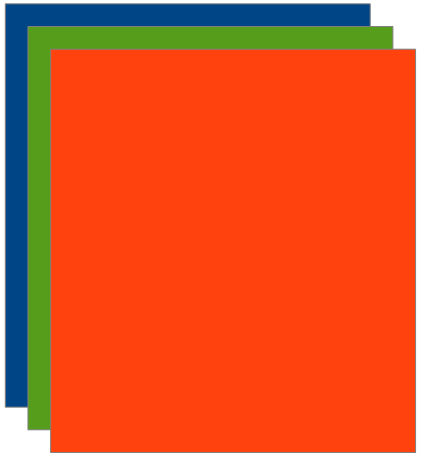
convolve a window and apply activation

Pooling



Pooling (Max/Avg)

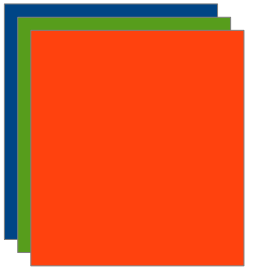
Input Feature Map



W x H x 3

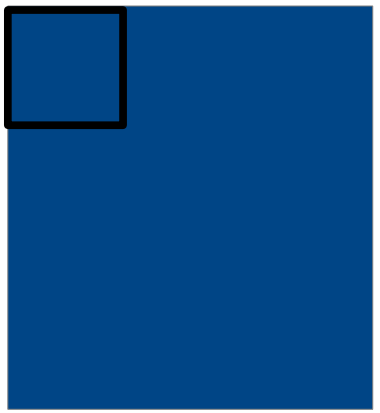
Pool size 2

Output Feature Map

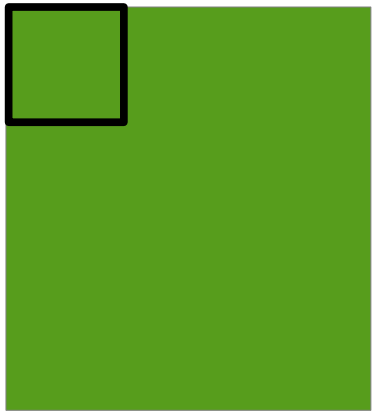


W/2 x H/2 x 3

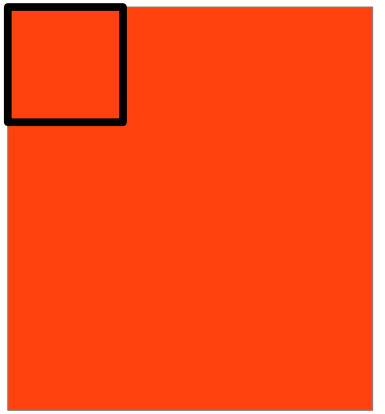
Max Pool



Max(I[0,0,■], I[1,0,■],
I[0,1,■], I[1,1,■])

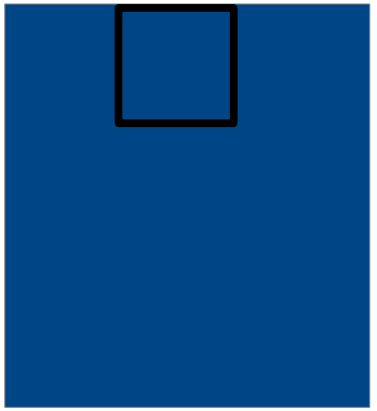


Max(I[0,0,■], I[1,0,■],
I[0,1,■], I[1,1,■])

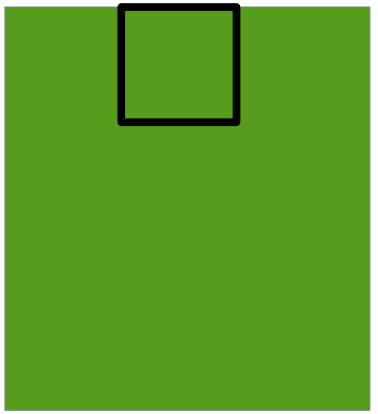


Max(I[0,0,■], I[1,0,■],
I[0,1,■], I[1,1,■])

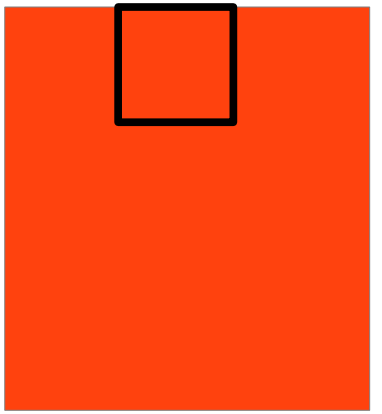
Max Pool



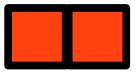
Max(I[2,0,■], I[3,0,■],
I[2,1,■], I[3,1,■])



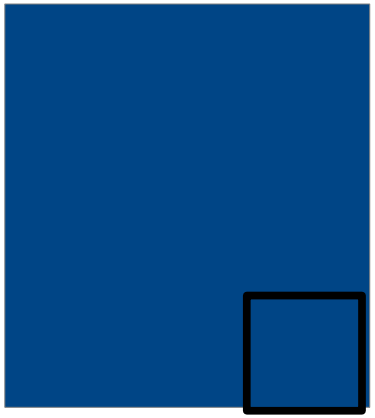
Max(I[2,0,■], I[3,0,■],
I[2,1,■], I[3,1,■])



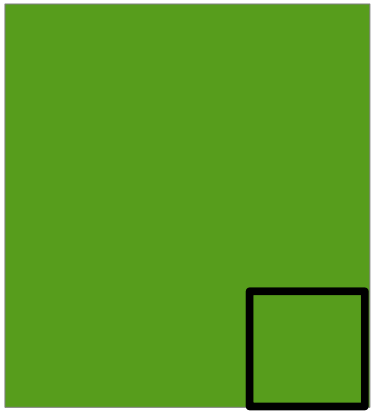
Max(I[2,0,■], I[3,0,■],
I[2,1,■], I[3,1,■])



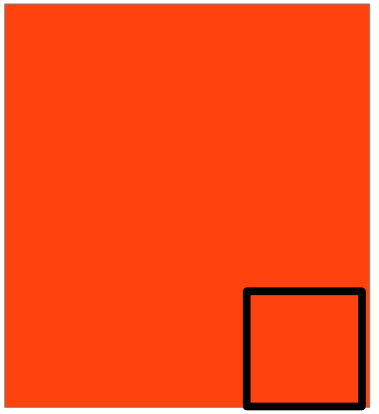
Max Pool



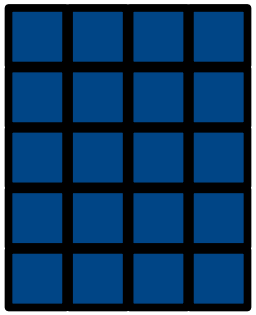
Max(I[2,0,■], I[3,0,■],
I[2,1,■], I[3,1,■])

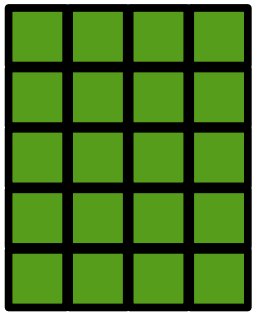


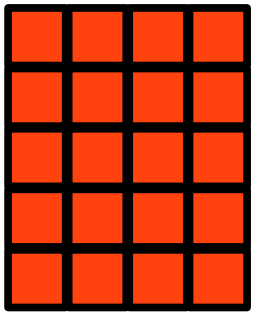
Max(I[2,0,■], I[3,0,■],
I[2,1,■], I[3,1,■])



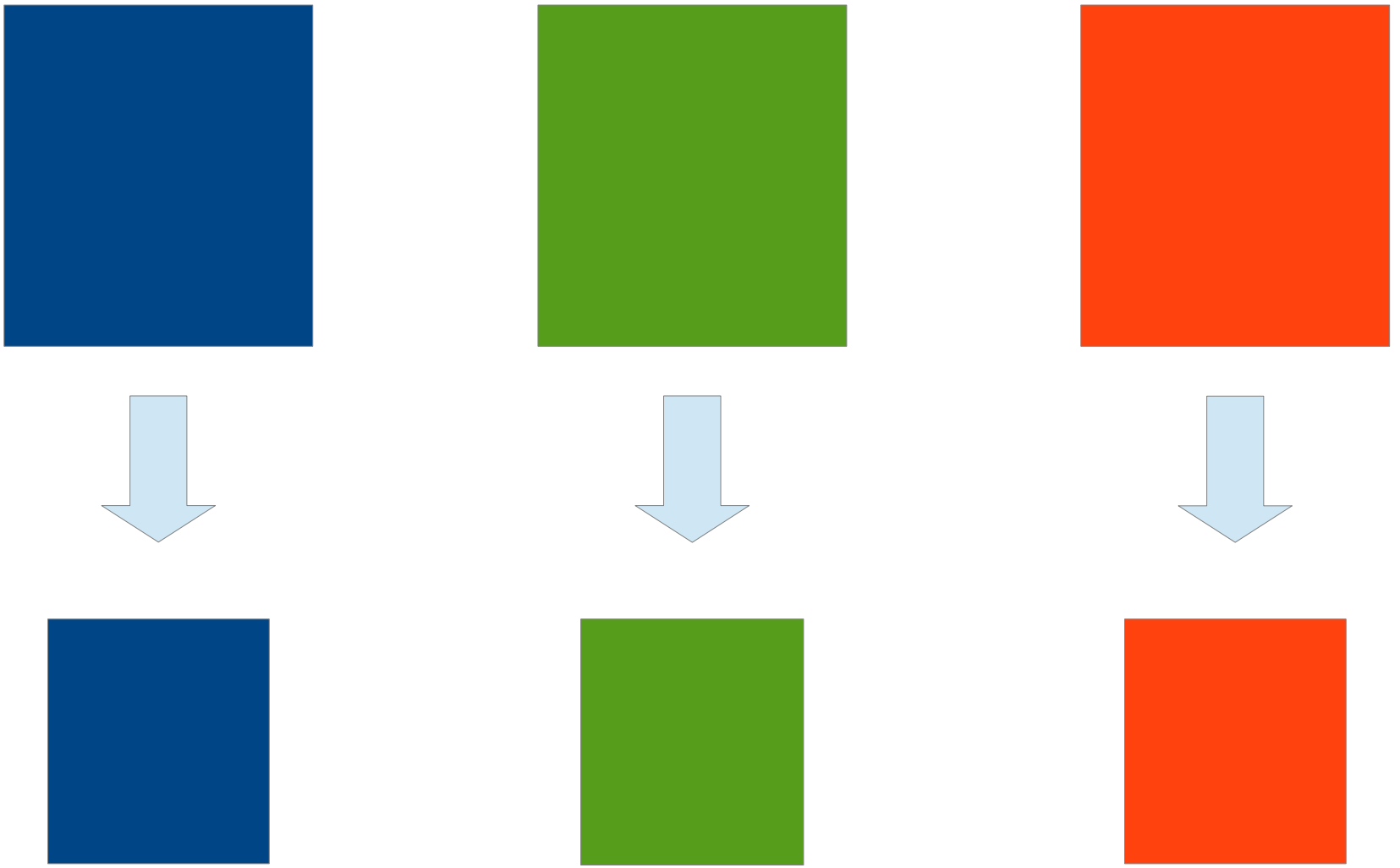
Max(I[2,0,■], I[3,0,■],
I[2,1,■], I[3,1,■])





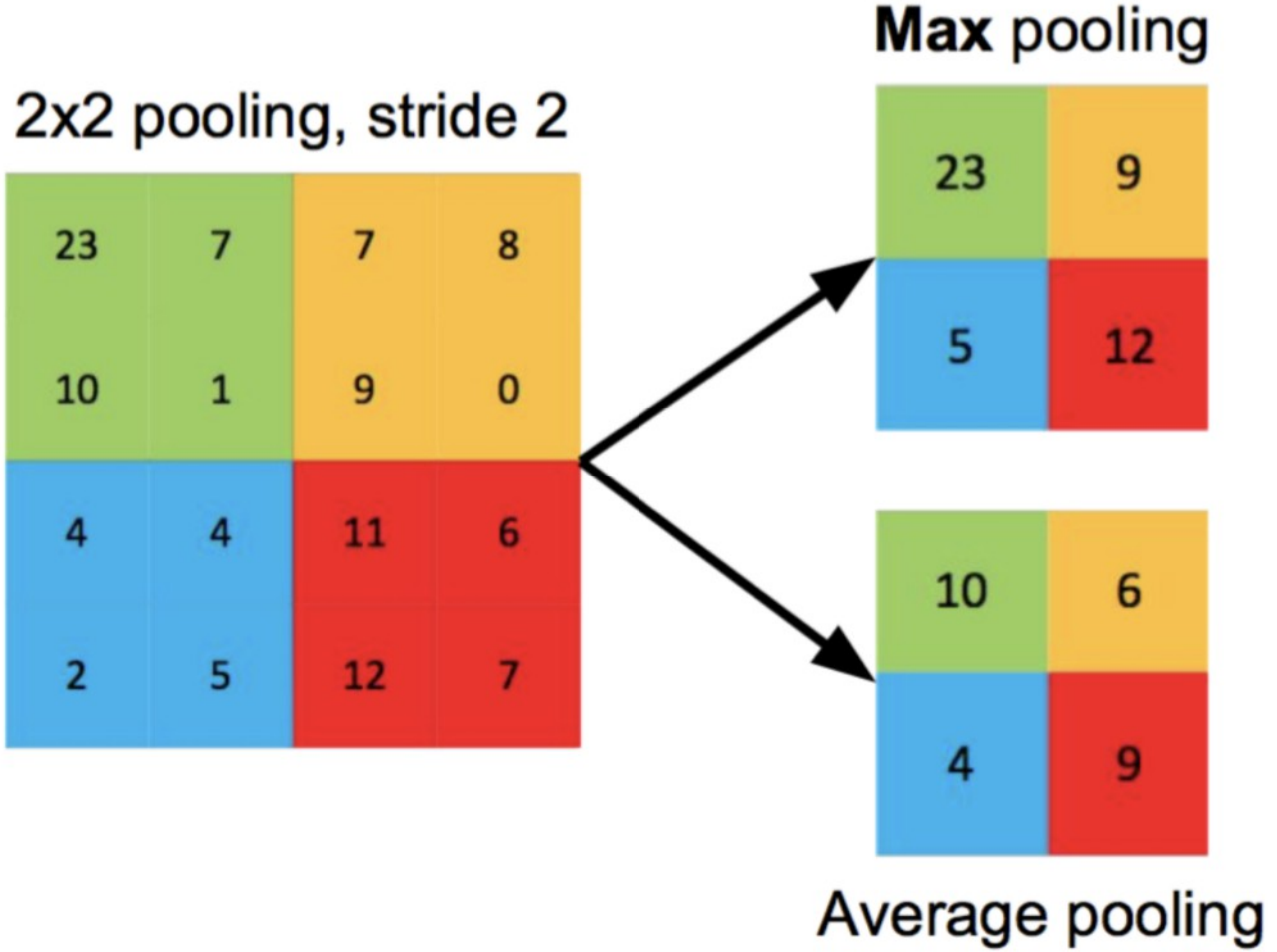


Max Pool

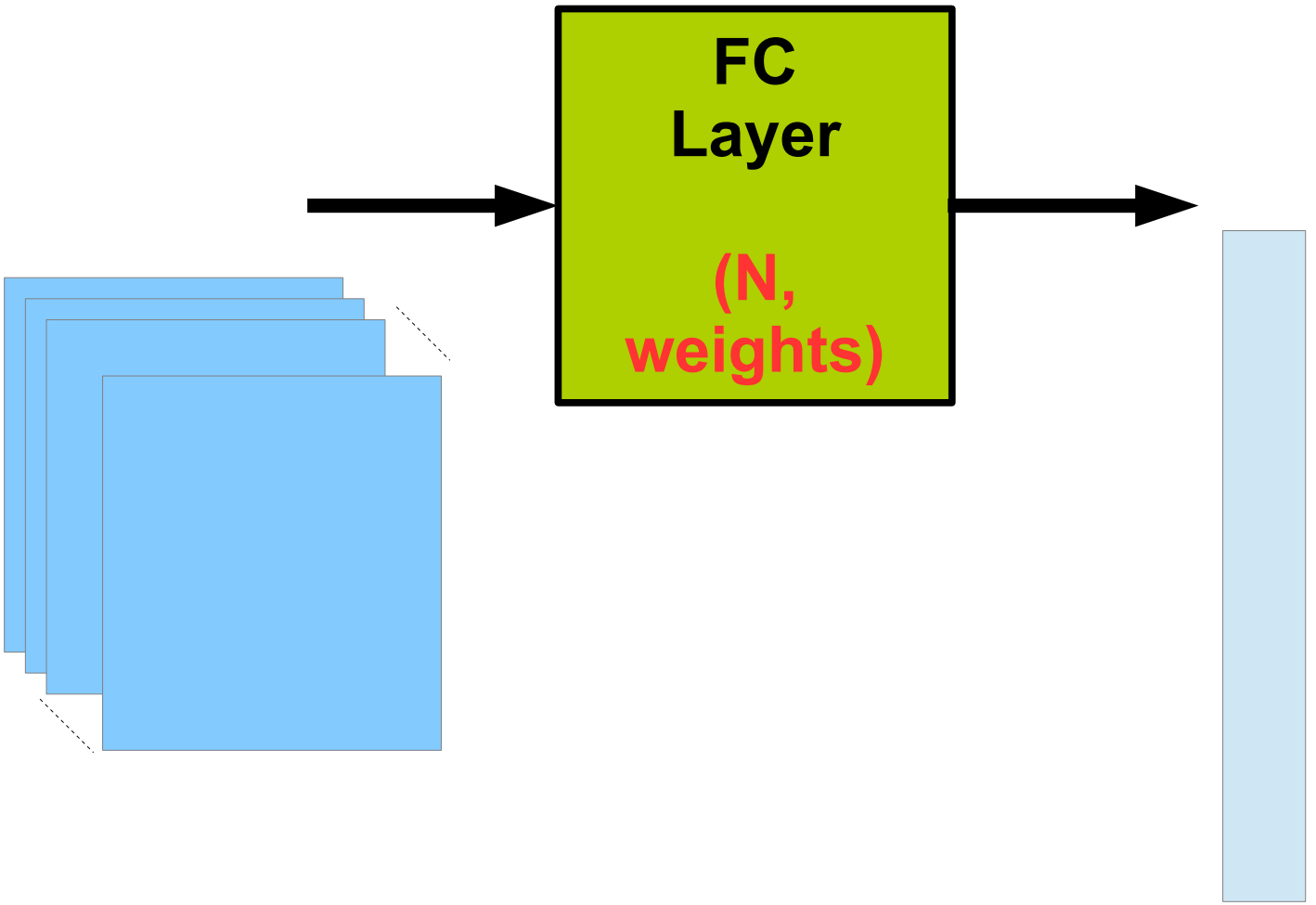


Pooling Layer

- Reduce resolution of each channel independently
- Overlapping or non-overlapping depending on stride

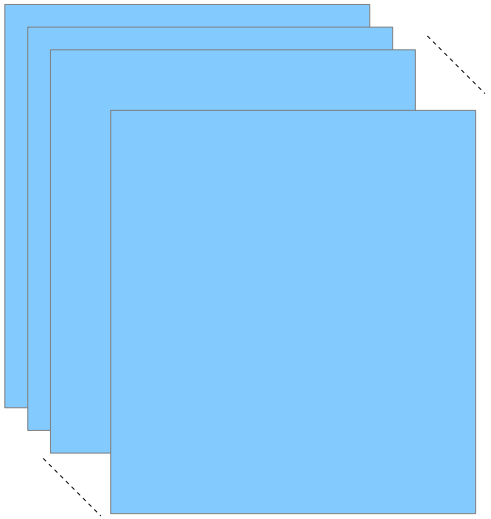


Fully Connected Layer



Fully Connected Layer

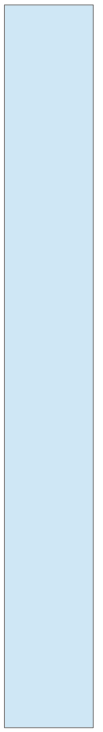
Input Feature Map



W × H × C

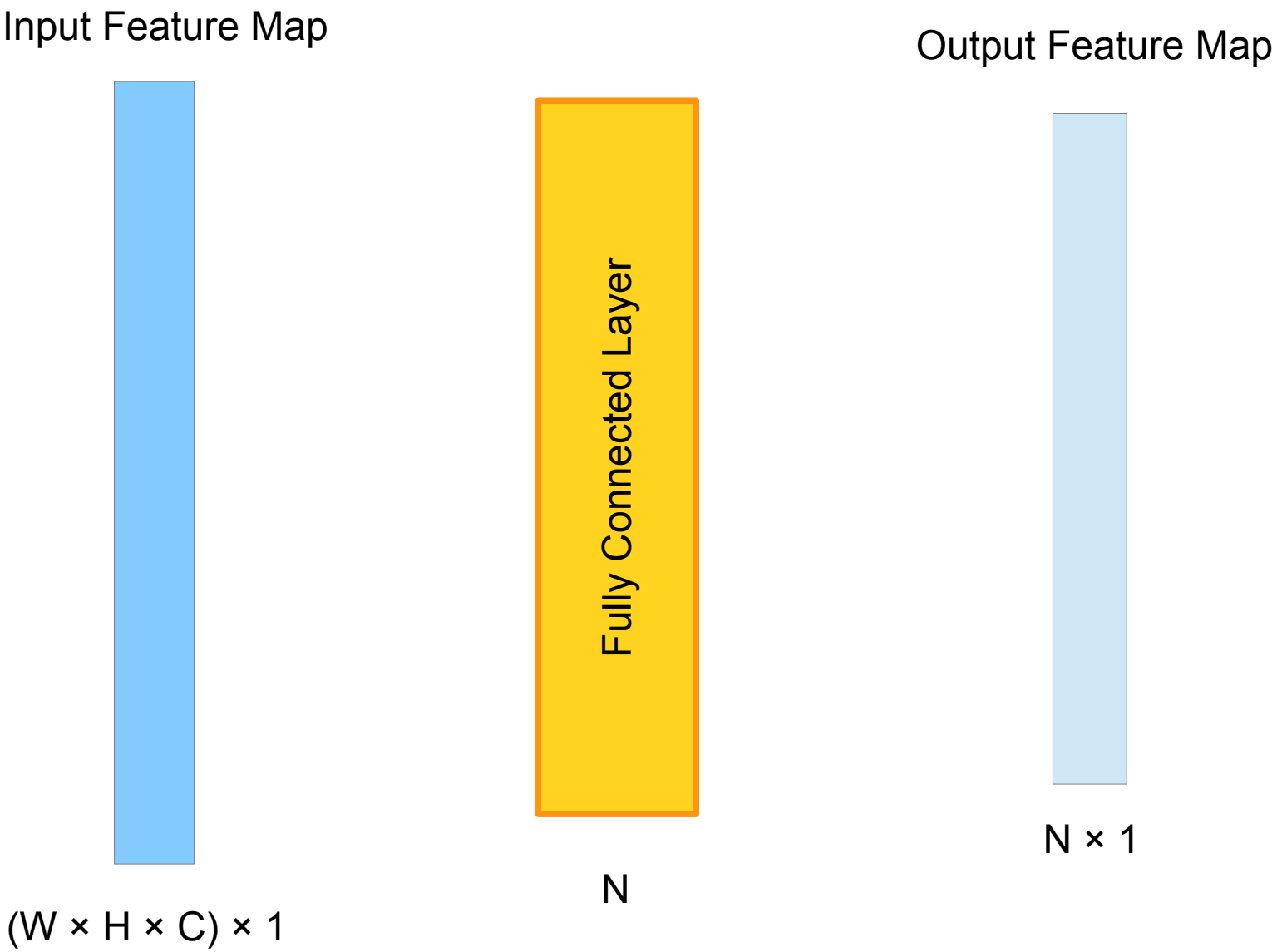
N

Output Feature Map

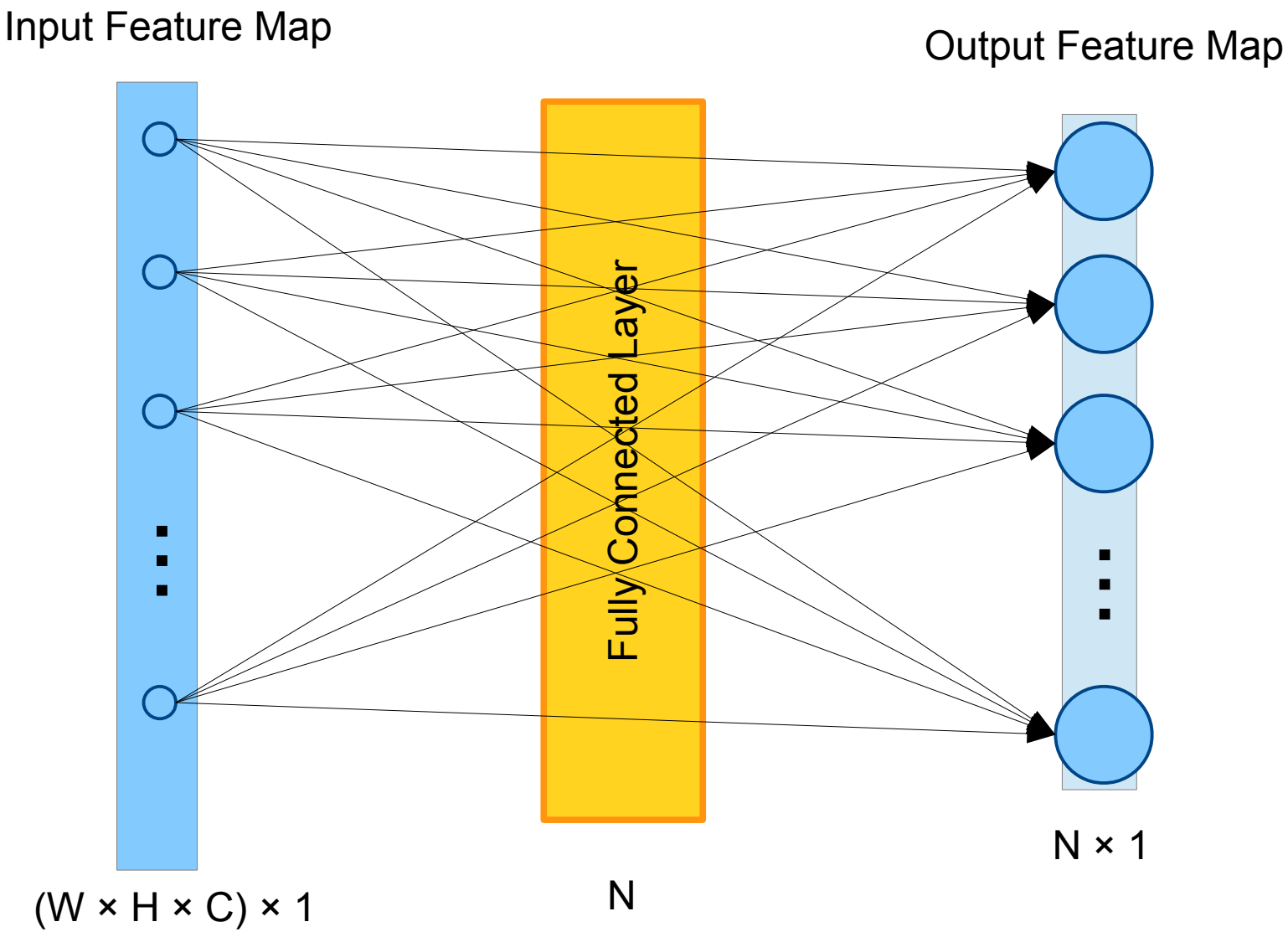


N × 1

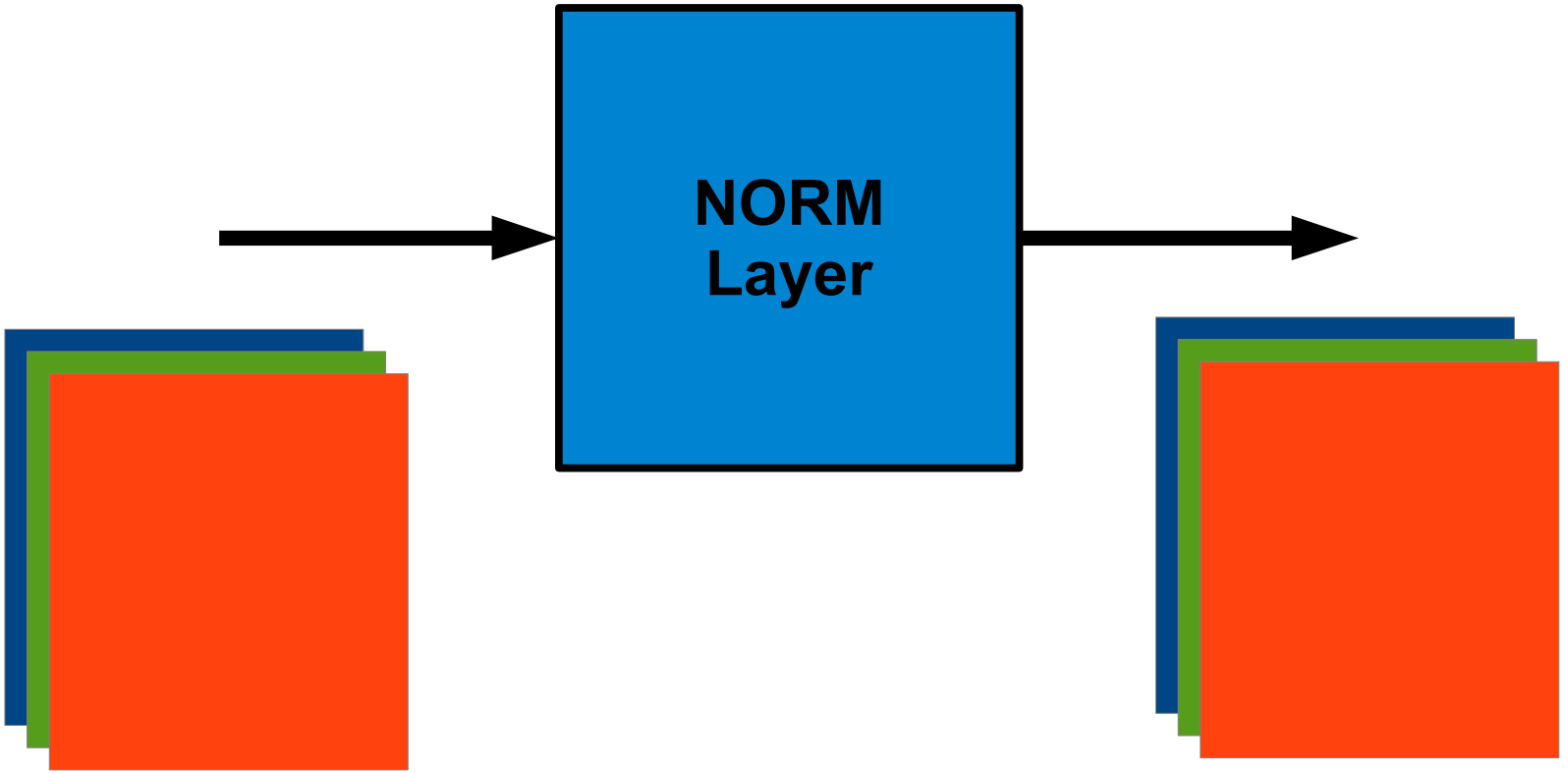
Fully Connected Layer



Fully Connected Layer



Normalization Layer



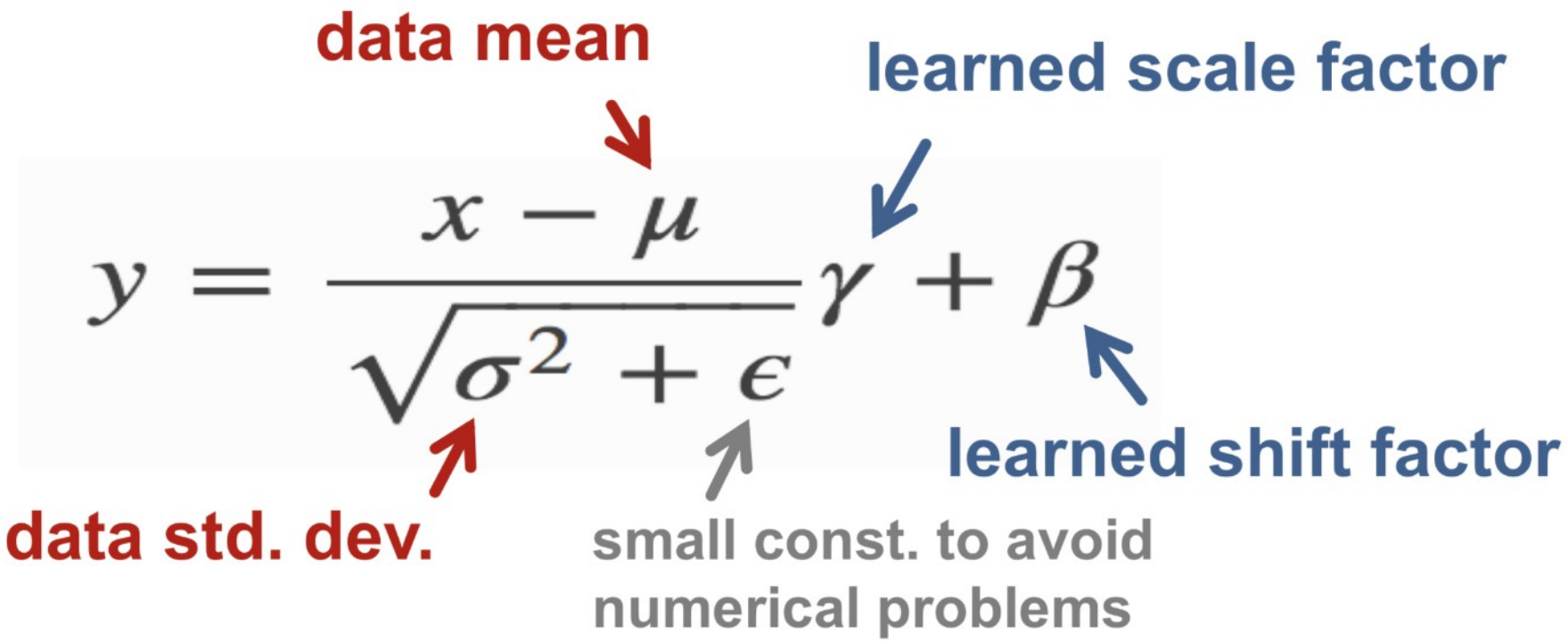
Normalization Layer

- Batch Normalization (BN)
  - Normalize activations towards mean 0 and stdev 1
  - Believed to be key to getting high accuracy and faster training on very deep neural networks

[S. Loffe & C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. ICML 2015]

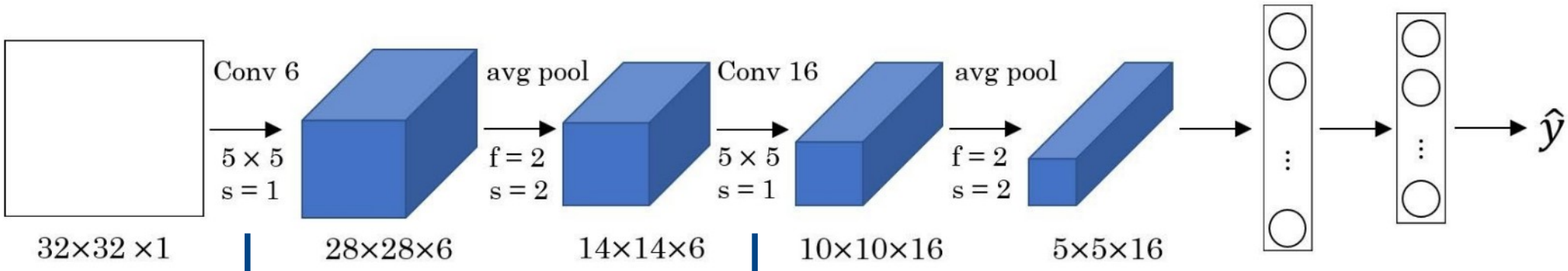
BN Layer Implementation

- The normalized value is further scaled and shifted, the parameters of which are learned from training



LeNet-5

LeCun et al., 1998. Gradient-based learning applied to document recognition



6 * (5 * 5 * 1) =
= **150 parameters**

16 * (5 * 5 * 6) =
= **2,400 parameters**

(5 * 5 * 16) * 120=
= **48,000 parameters**

120 * 84 =
= **10,080 parameters**

60 K parameters

LeNet-5

LeCun et al., 1998. Gradient-based learning applied to document recognition

- MNIST
  - 28x28 pixels (B&W)
  - 10 Classes
  - 60,000 Training
  - 10,000 Testing

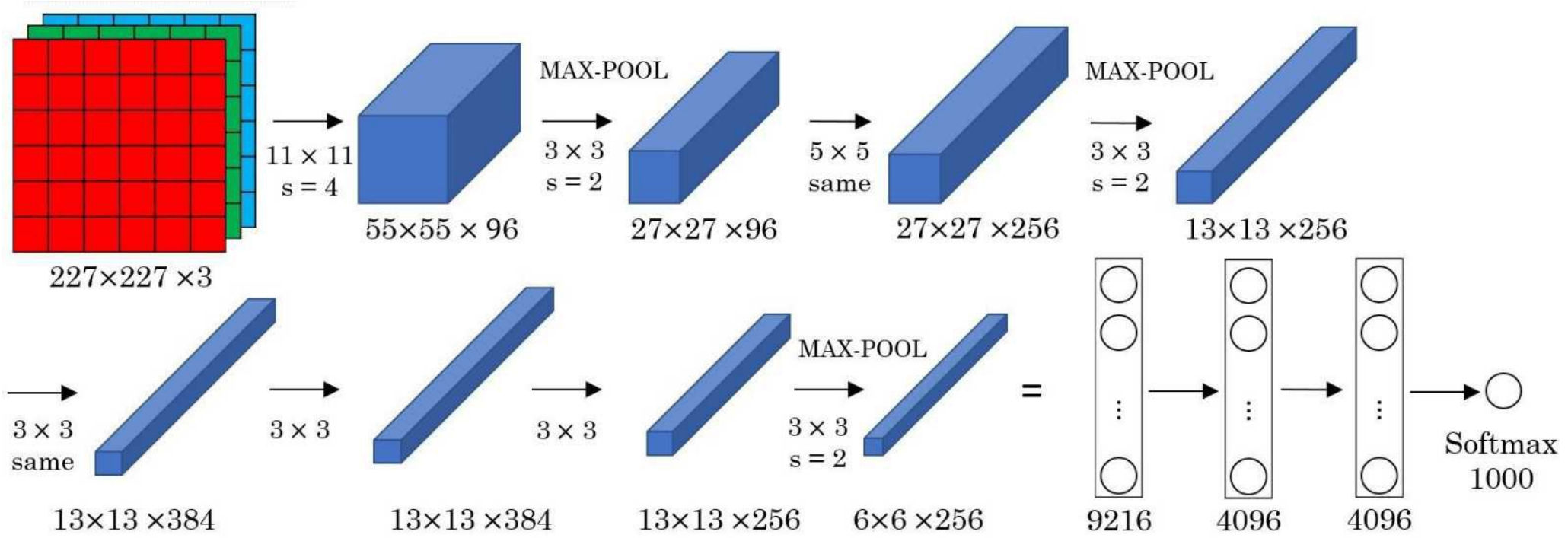
LeNet-5

LeCun et al., 1998. Gradient-based learning applied to document recognition

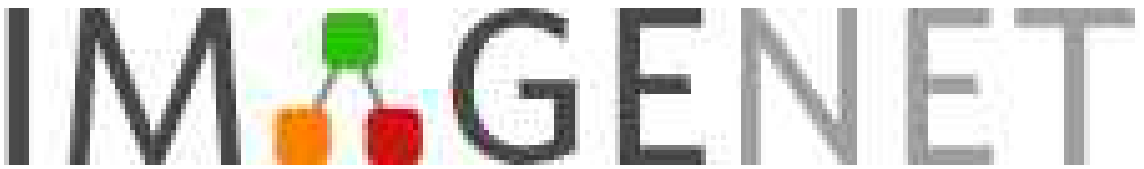
http://yann.lecun.com/exdb/lenet/

AlexNet

Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks



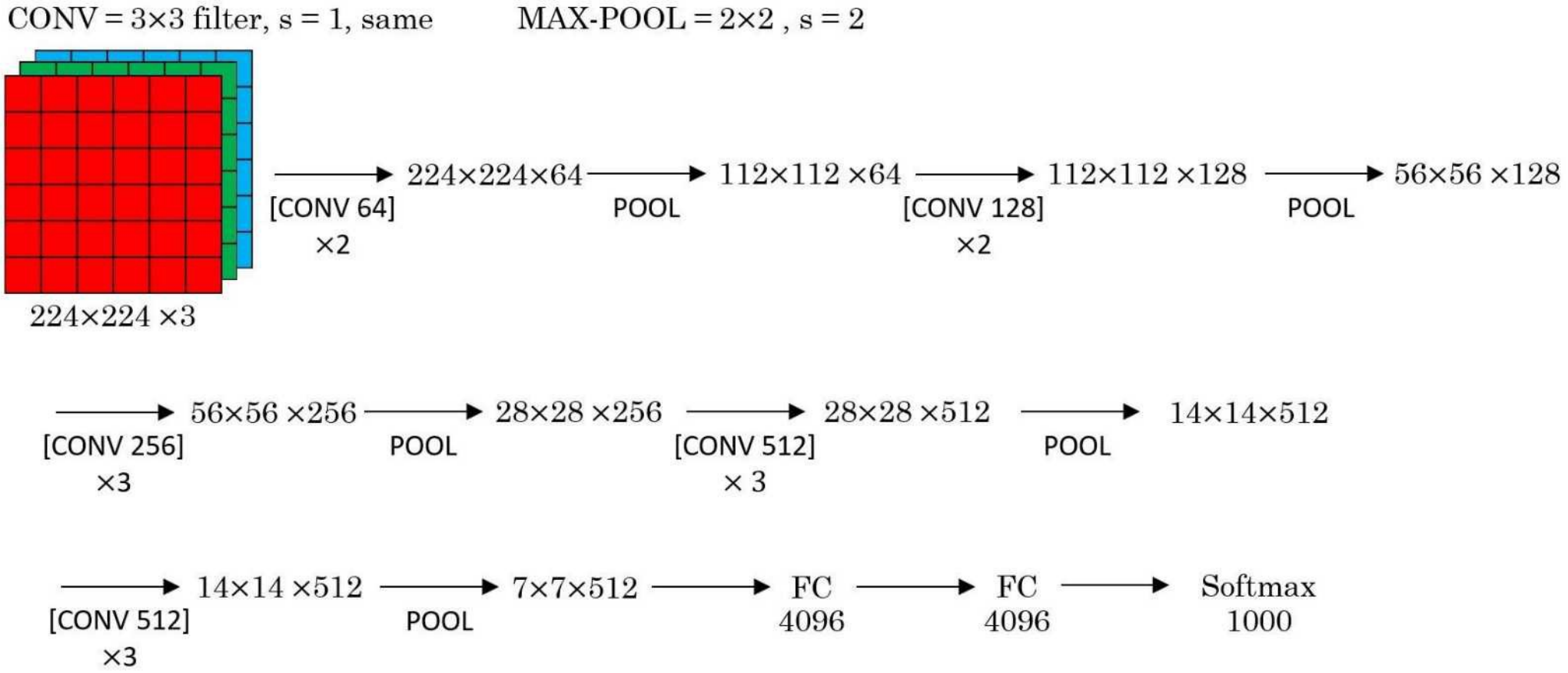
~60 M parameters



- http://www.image-net.org/challenges/LSVRC/
- ~256x256 pixels (color)
- 1000 Classes
- 1.3M Training
- 100,000 Testing

VGG-16

Simonyan & Zisserman 2015. Very deep convolutional networks for large-scale image recognition



~138 M parameters

Evolution of DNN Architectures

Year	Model name	Accuracy (%)	Size (MB)	MACs (G)	#Params (M)
2012	**AlexNet**	56.48	237.89	0.72	61.10
2014	**VGG19**	72.38	461.10	0.40	20.08
2014	**Inception_v3**	79.35	5212.60	1.53	6.25
2016	**ResNet18**	76.82	721.60	0.56	11.22
2016	**ResNet50**	78.07	4233.60	1.30	23.70
2016	**ResNet101**	77.35	6409.60	2.52	42.70
2016	**ResNet152**	78.28	9097.60	3.74	58.34
2018	**DenseNet121**	78.46	5025.60	0.90	7.05
2018	**DenseNet169**	75.56	6111.60	1.07	12.64
2018	**DenseNet201**	76.87	7947.60	1.38	18.28
2020	**COVID-Net-Large***	-na-	270.88	0.13	33.85

Variants and Expedients

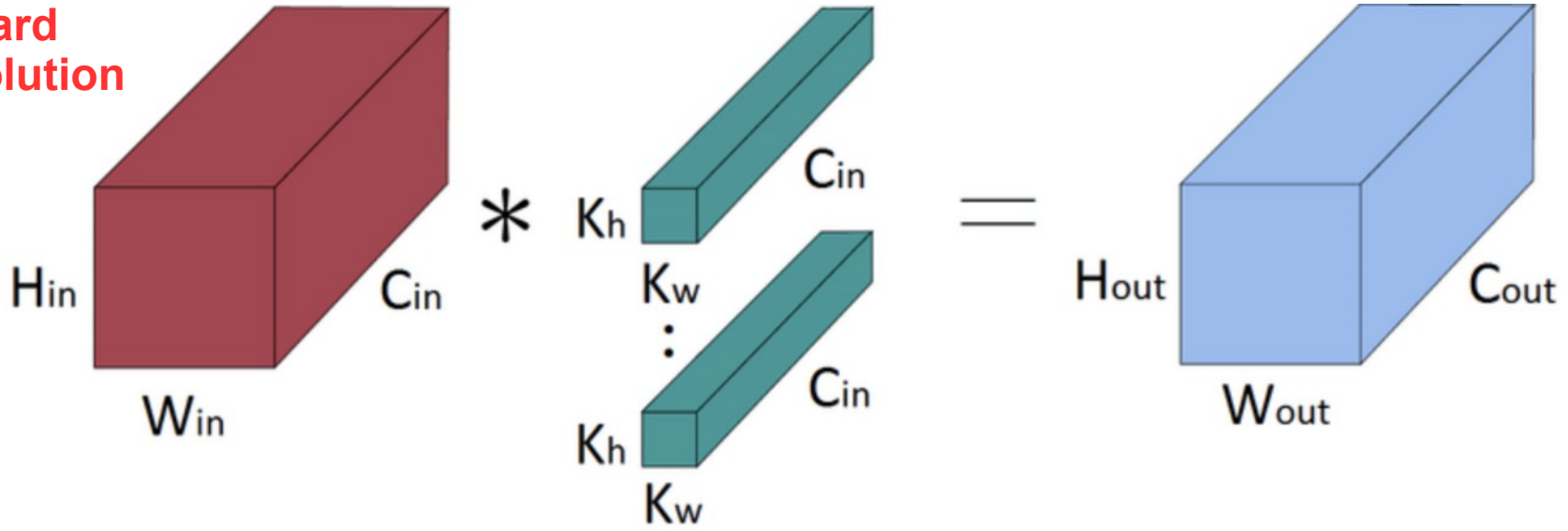
- Grouped Convolution
- Stacked Filter
- Inception Module
- 1x1 Filter
- Residual Connection

Grouped Convolution

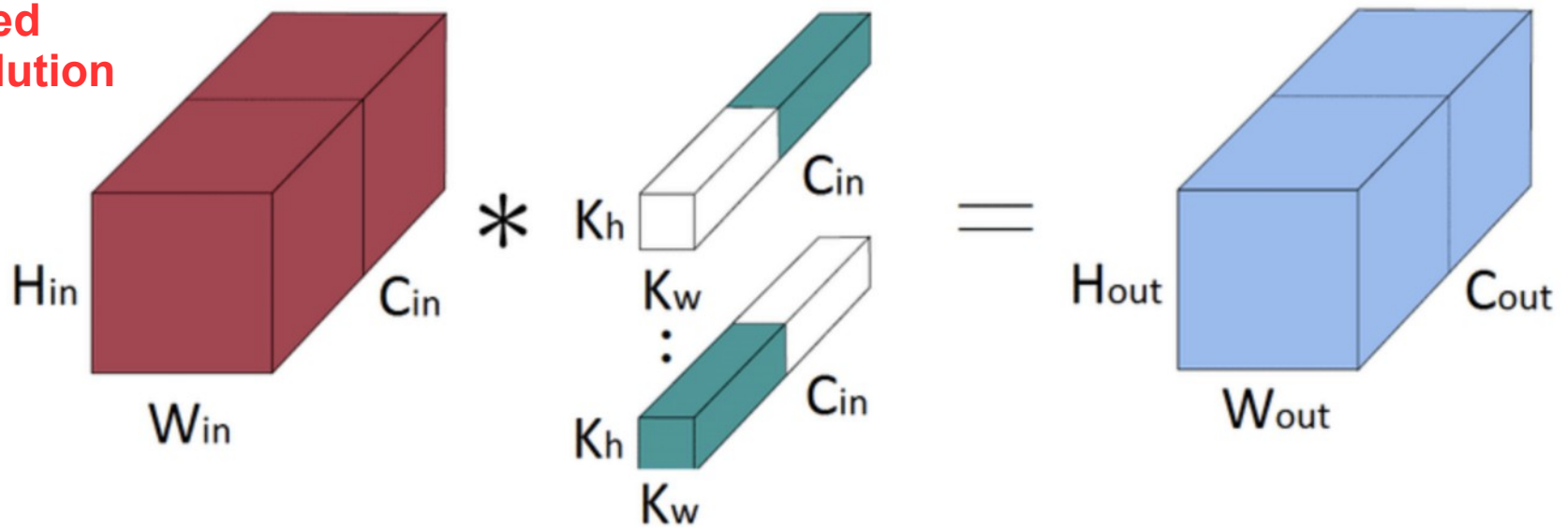
- To reduce the amount of weights
  - *e.g.*, used in AlexNet
- Channels of the Ifmap are split such that filters can have less channels

Grouped Convolution

Standard Convolution

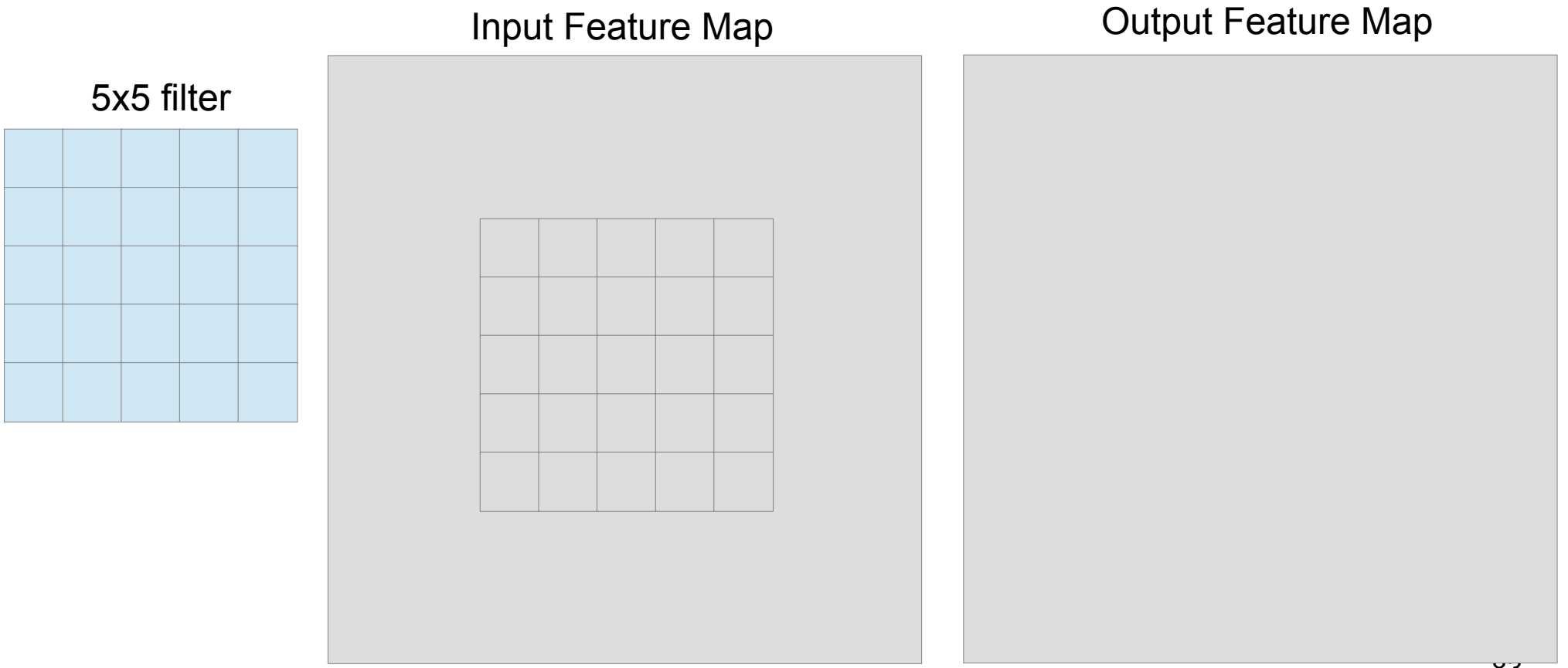


Grouped Convolution



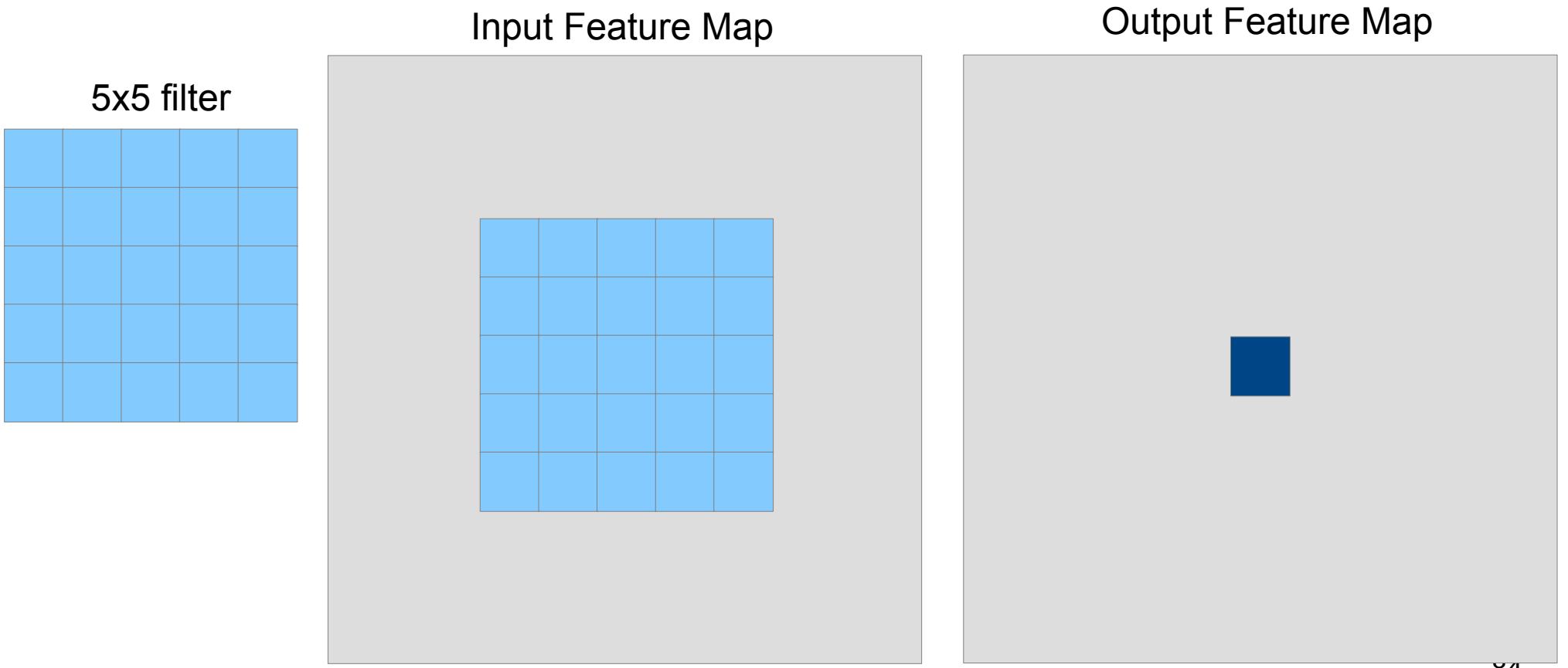
Stacked Filter

- Large filters built from multiple small filters
  - *e.g.*, used in VGG-16



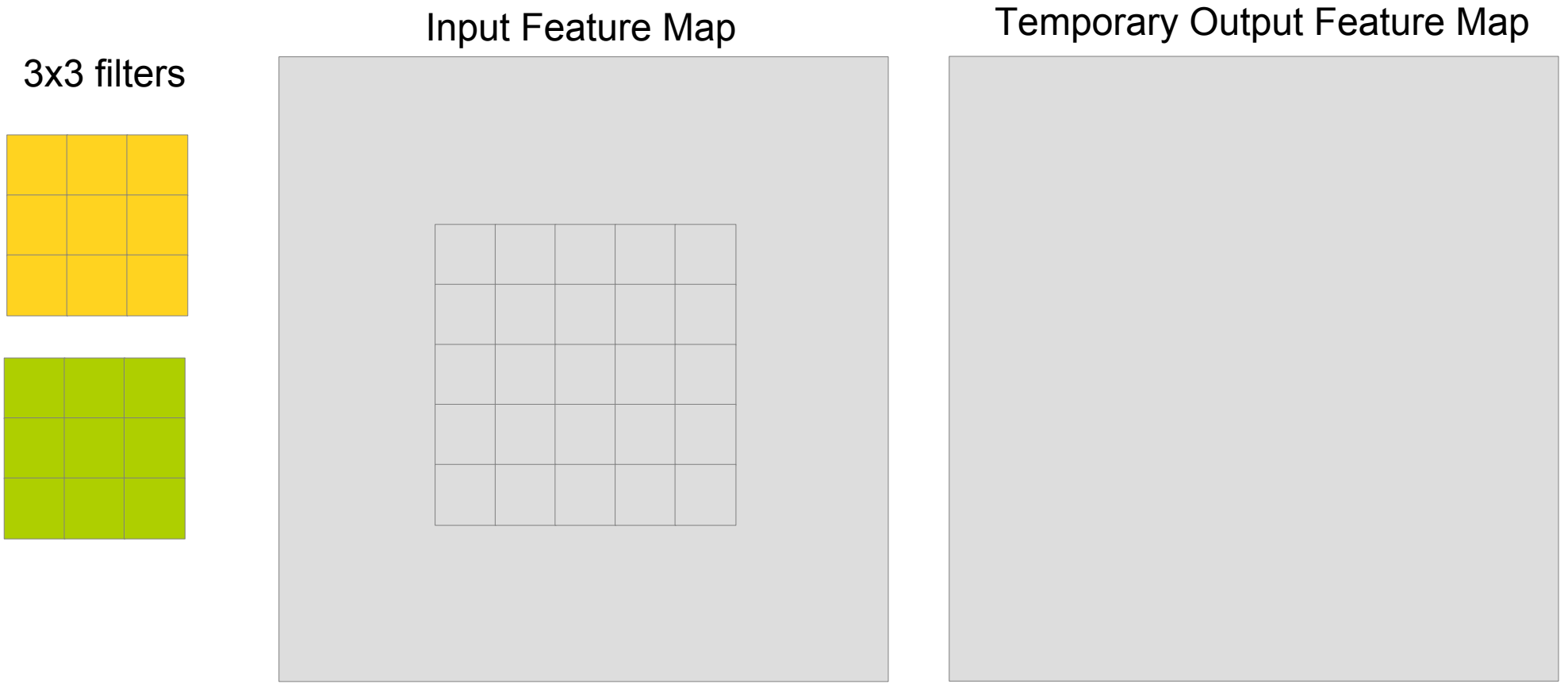
Stacked Filter

- Large filters built from multiple small filters
  - *e.g.*, used in VGG-16



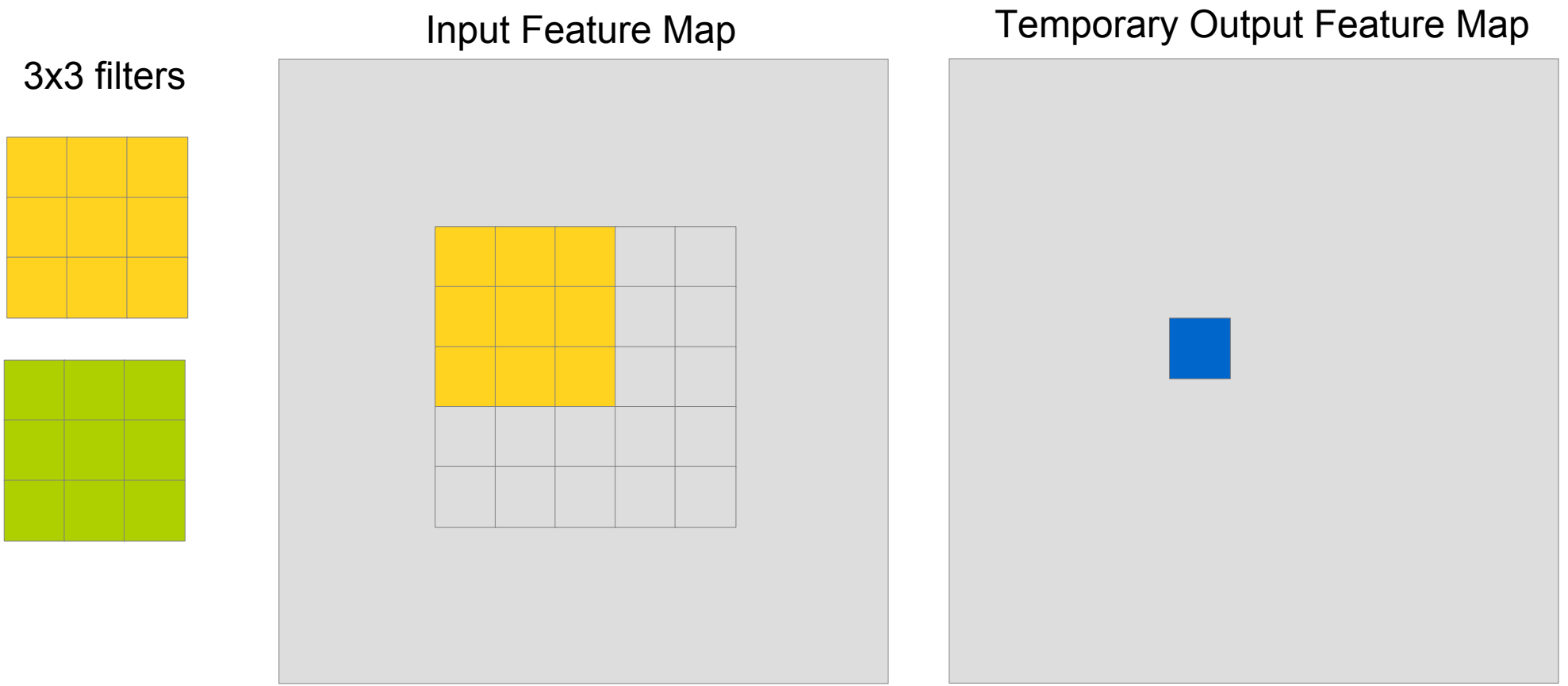
Stacked Filter

- Large filters built from multiple small filters
  - *e.g.*, used in VGG-16



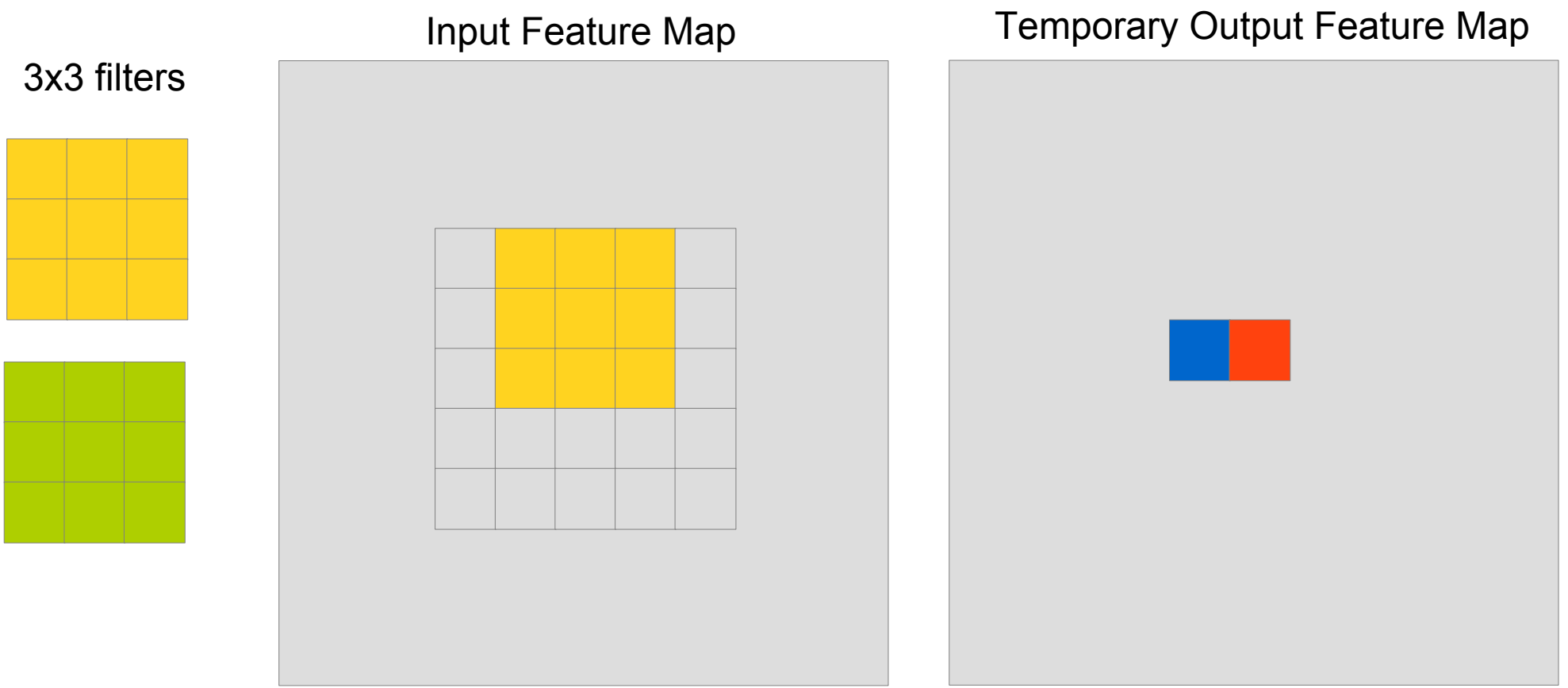
Stacked Filter

- Large filters built from multiple small filters
  - *e.g.*, used in VGG-16



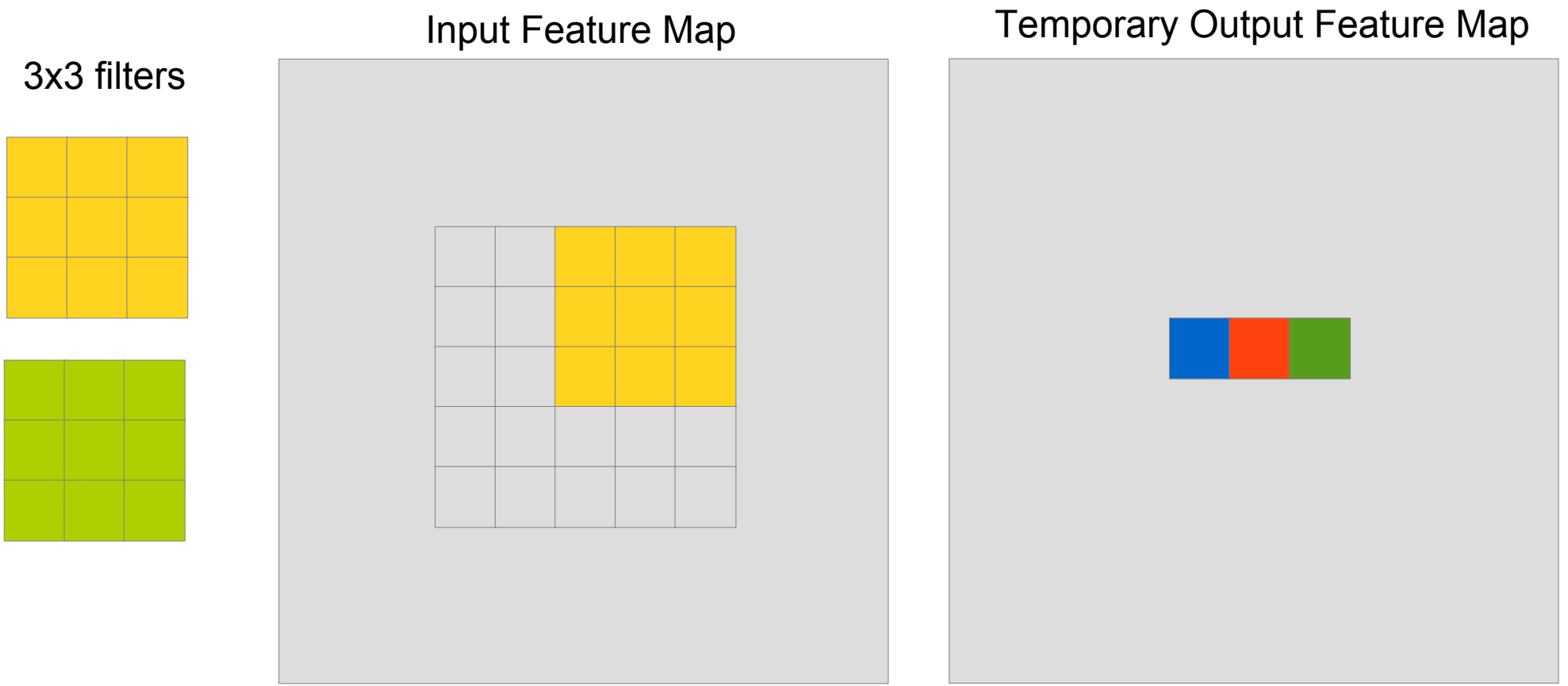
Stacked Filter

- Large filters built from multiple small filters
  - *e.g.*, used in VGG-16



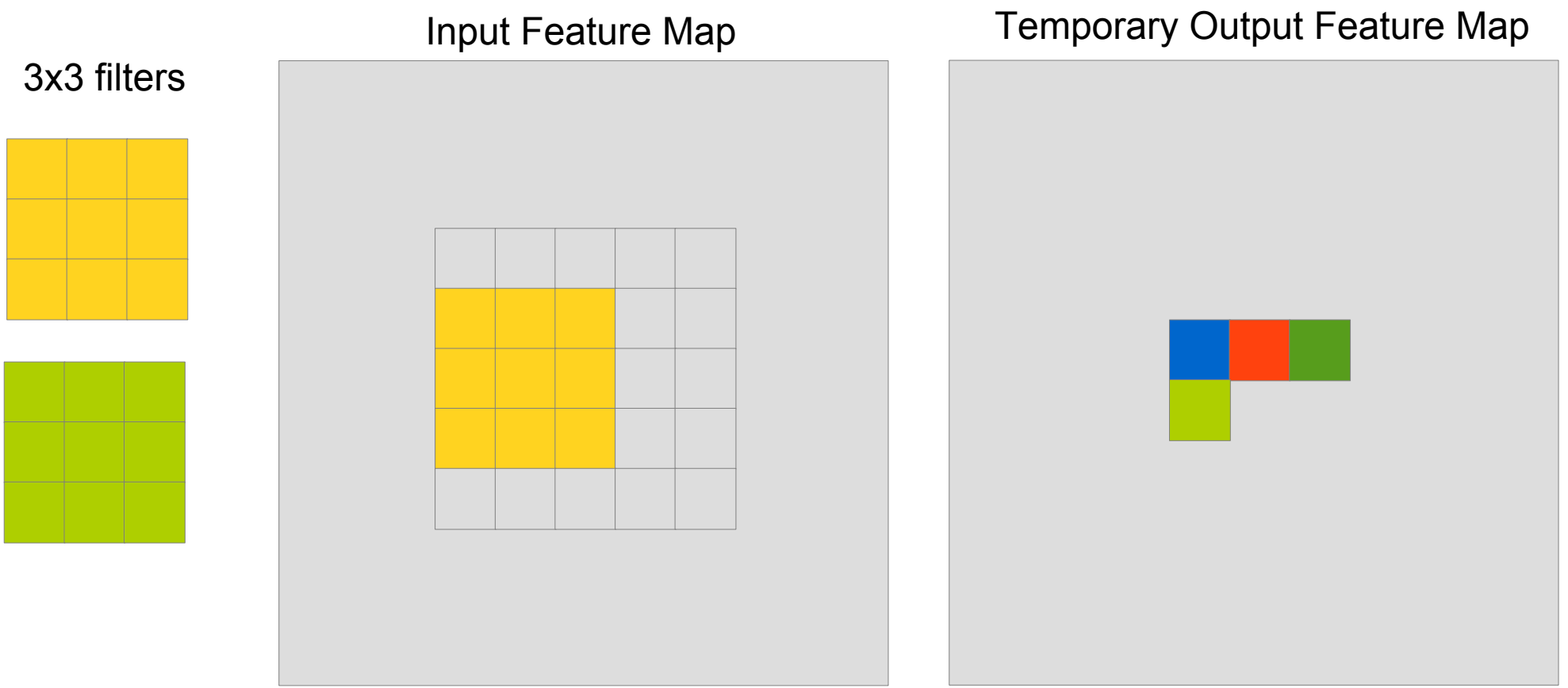
Stacked Filter

- Large filters built from multiple small filters
  - *e.g.*, used in VGG-16



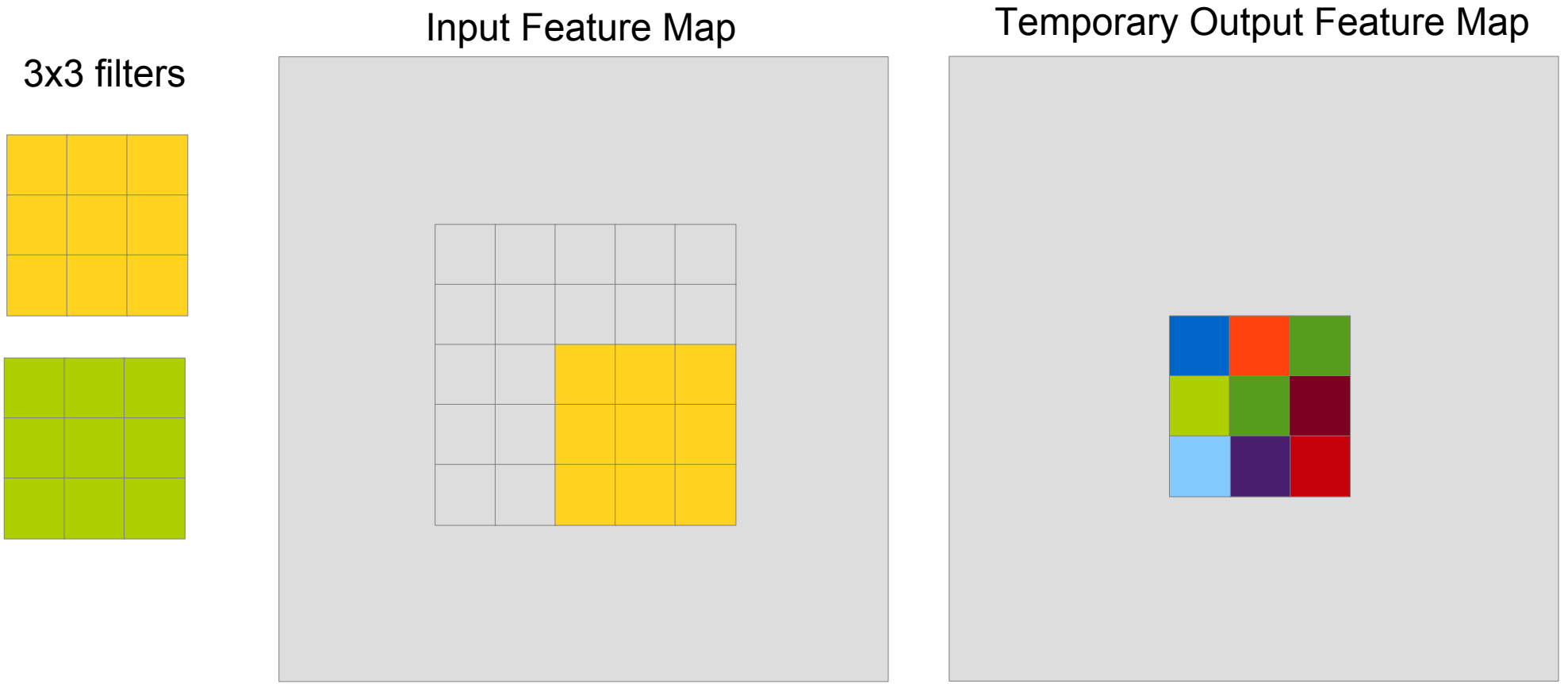
Stacked Filter

- Large filters built from multiple small filters
  - *e.g.*, used in VGG-16



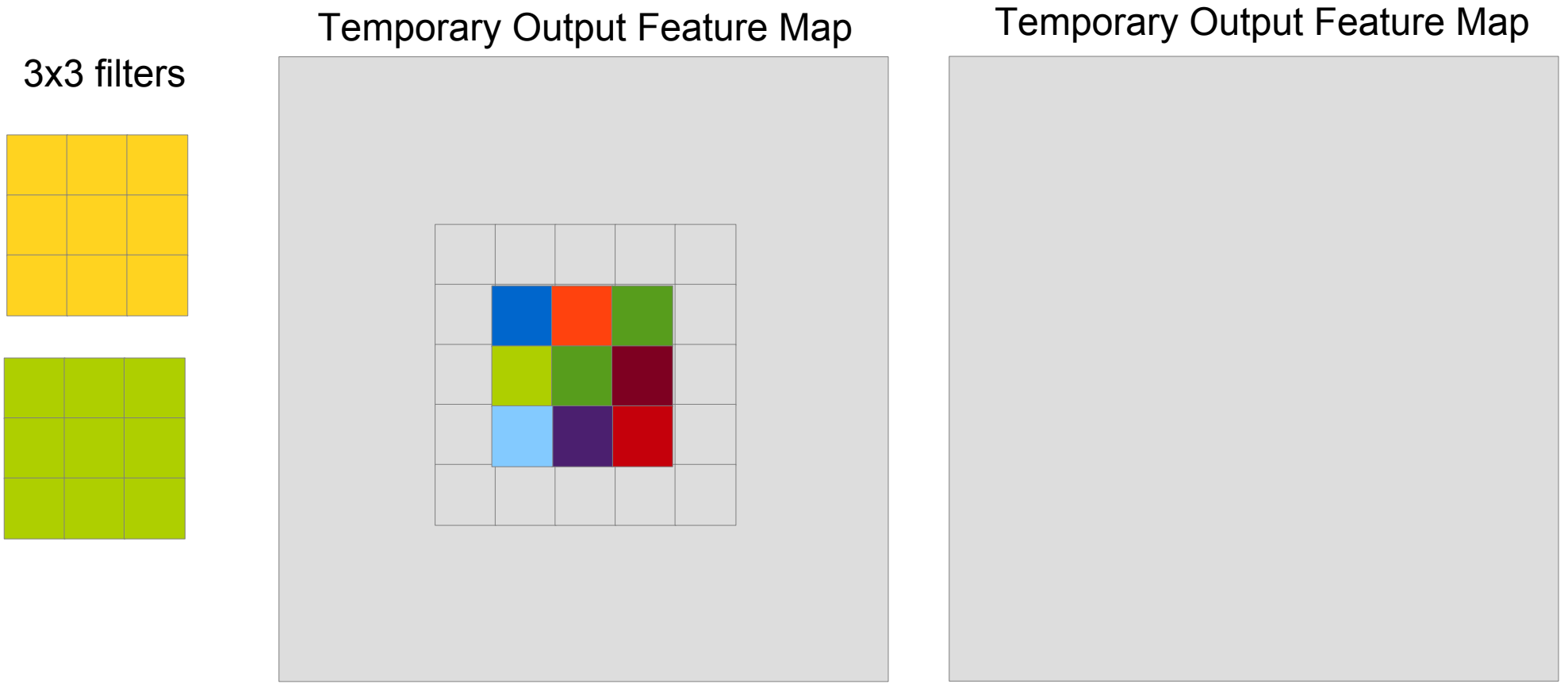
Stacked Filter

- Large filters built from multiple small filters
  - *e.g.*, used in VGG-16



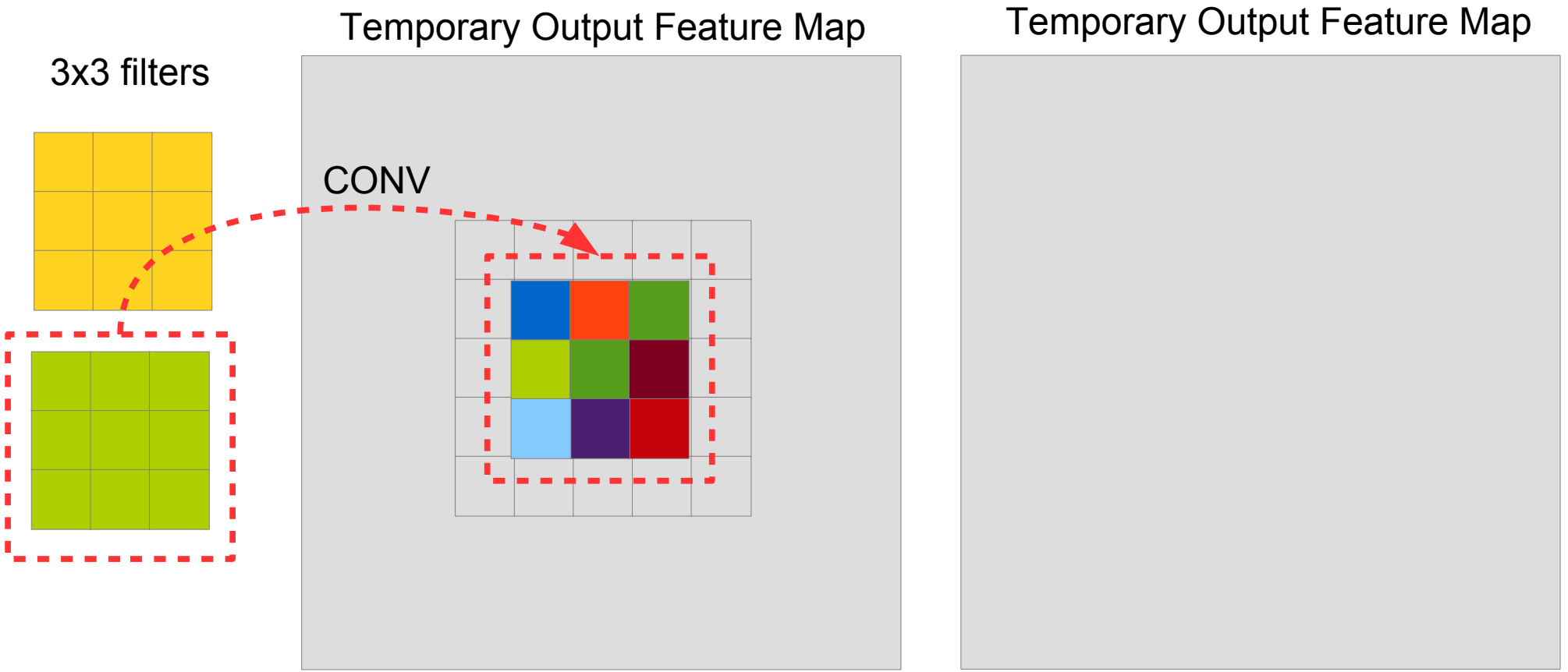
Stacked Filter

- Large filters built from multiple small filters
  - *e.g.*, used in VGG-16



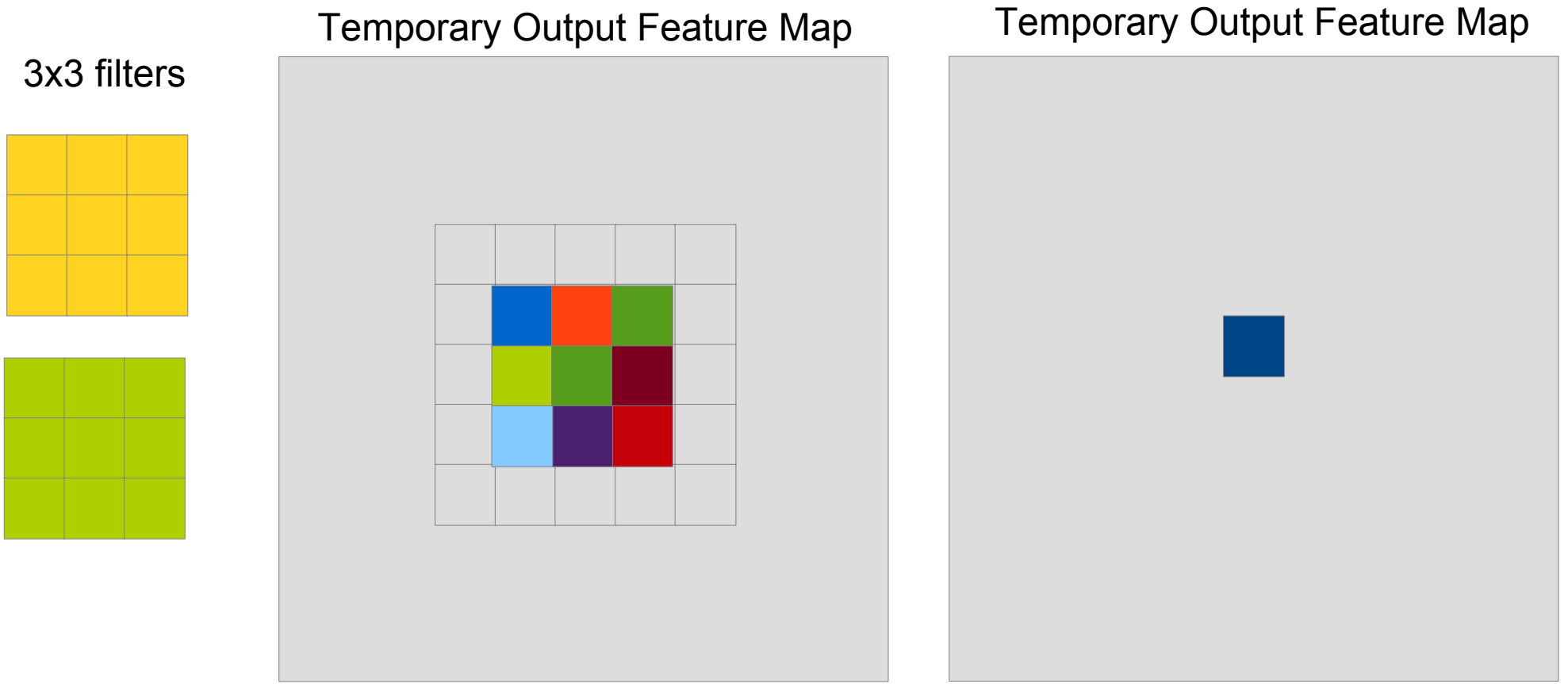
Stacked Filter

- Large filters built from multiple small filters
  - *e.g.*, used in VGG-16



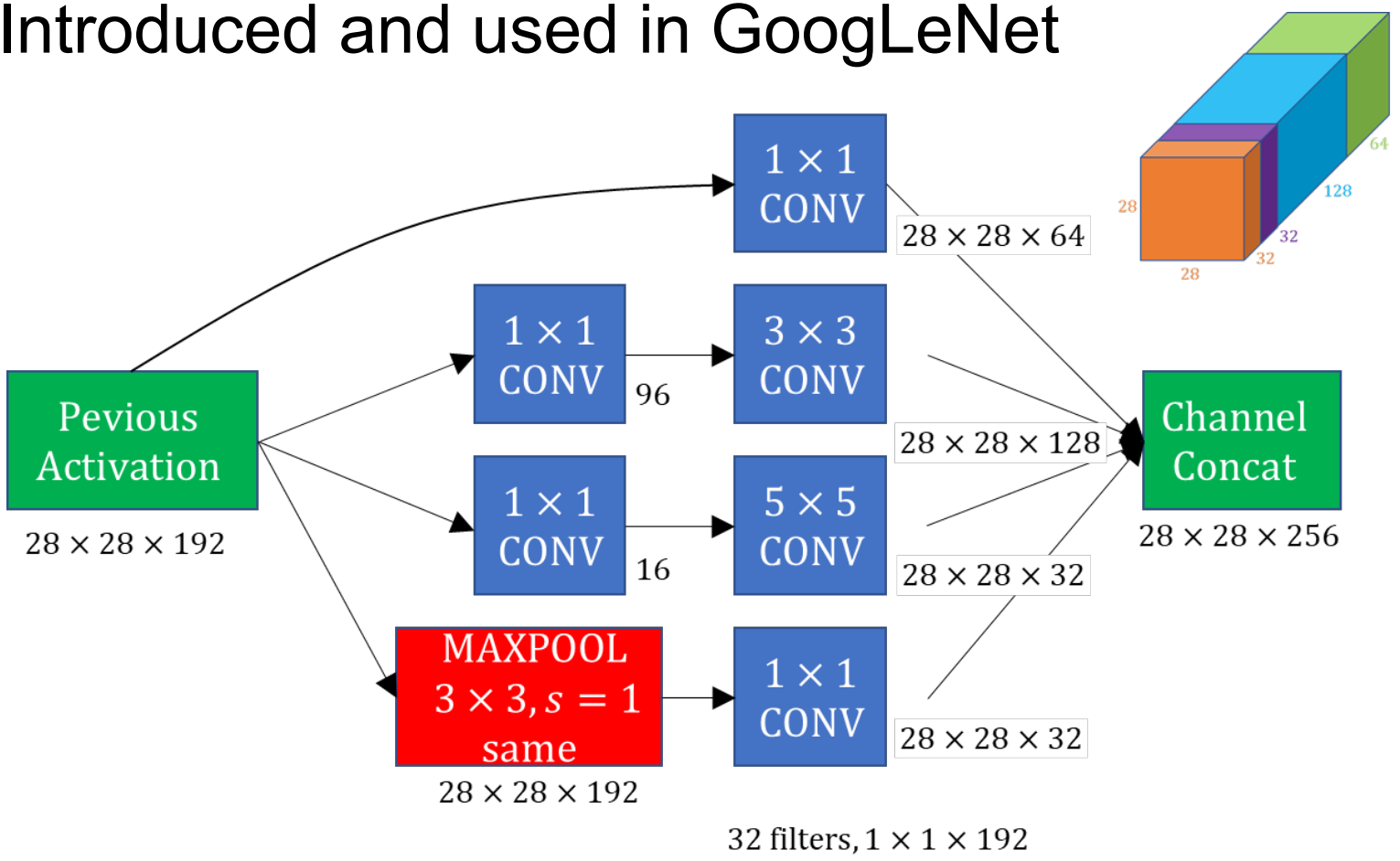
Stacked Filter

- Large filters built from multiple small filters
  - *e.g.*, used in VGG-16



Inception Module

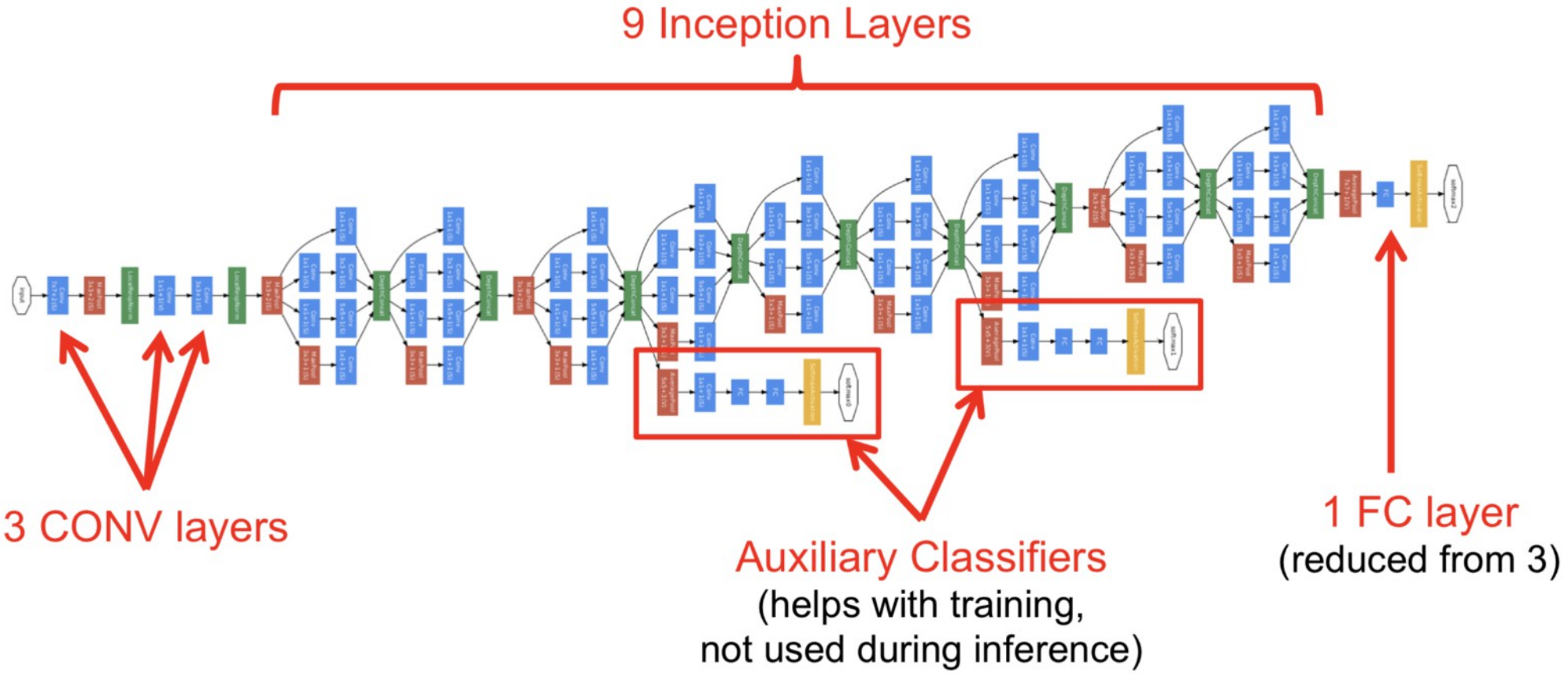
- Use multiple filter sizes for processing the input at multiple scales
  - Introduced and used in GoogLeNet



GoogLeNet/Inception (v1)

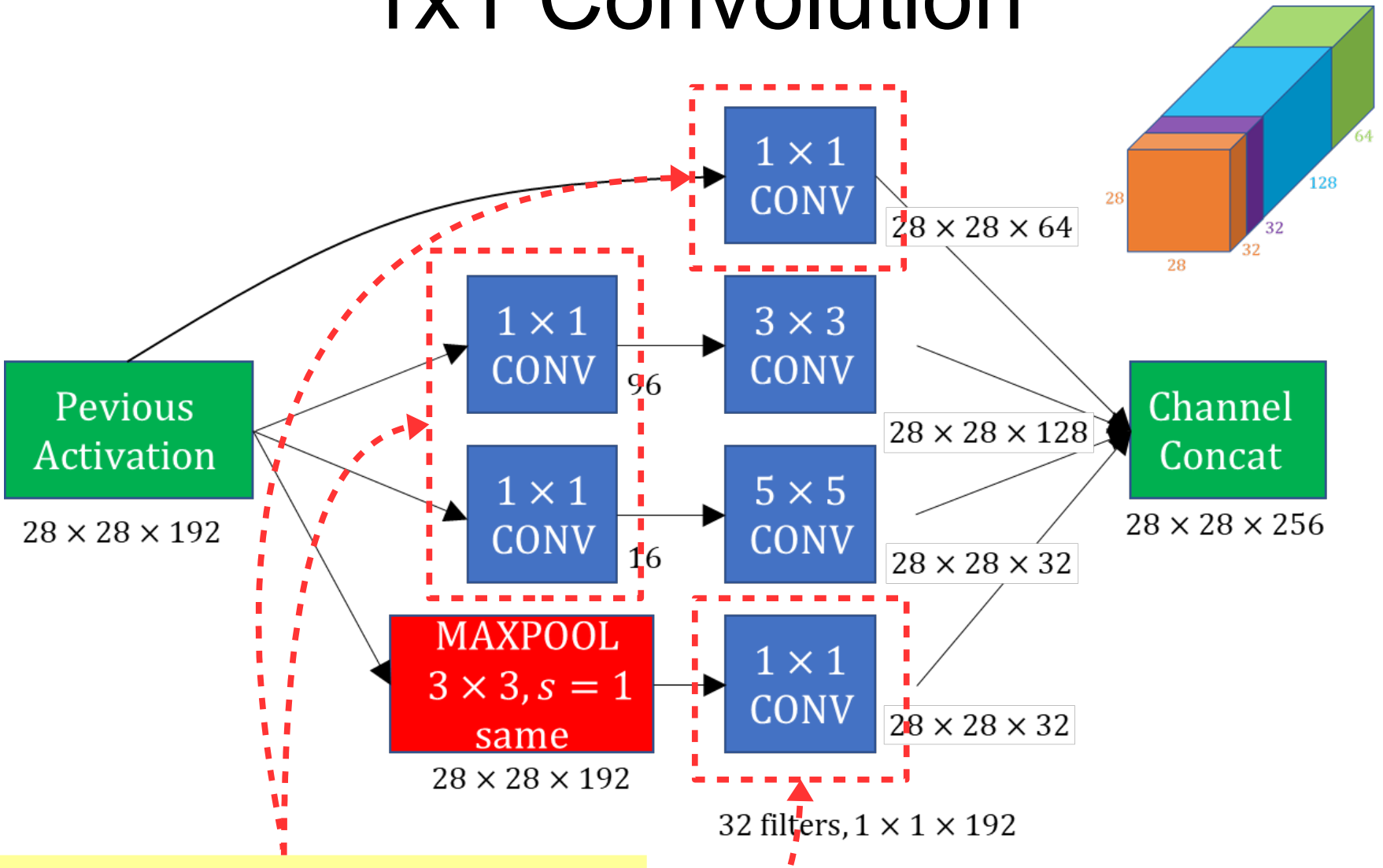
(Also, v2, v3 and v4)
ILSVRC14 Winner

- 57 layers (21 CONV, 1 FC)
- 7M parameters, 1.43G MACs



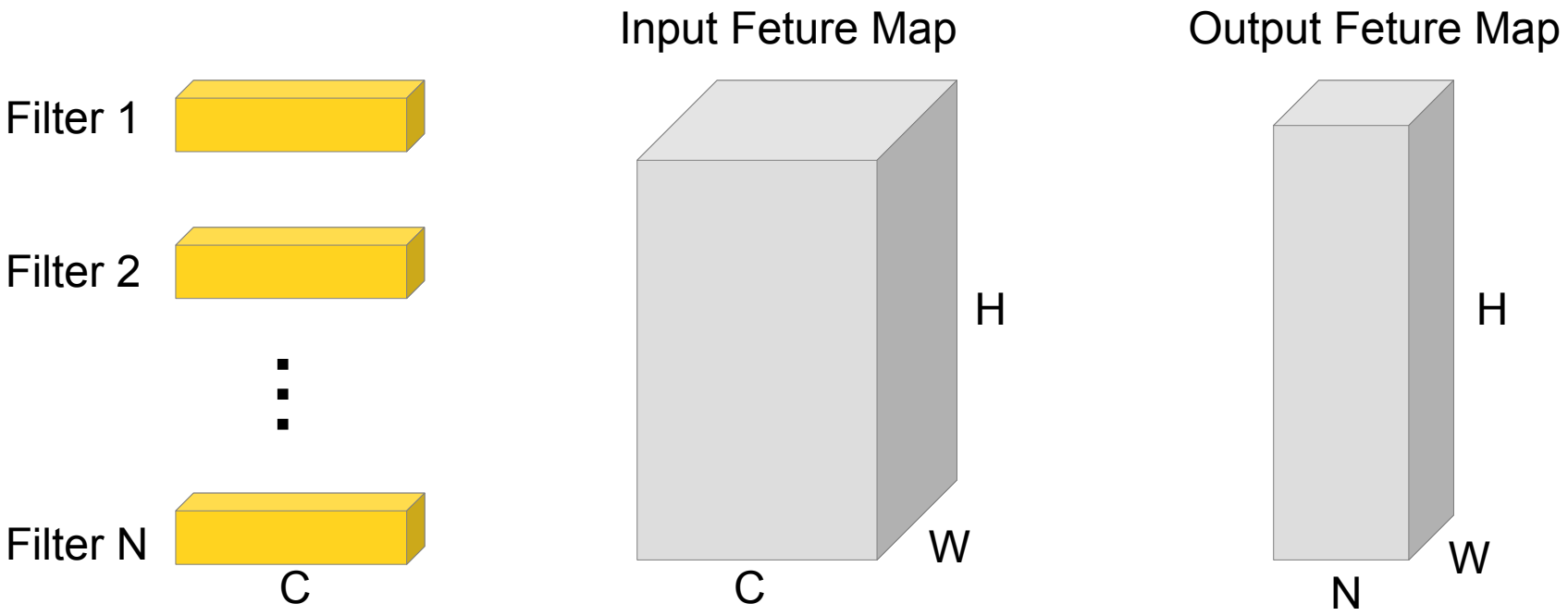
[C. Szegedy *et al.*, "Going deeper with convolutions," CVPR'15]

1x1 Convolution



1x1 CONV are used to reduce number of weights and multiplications

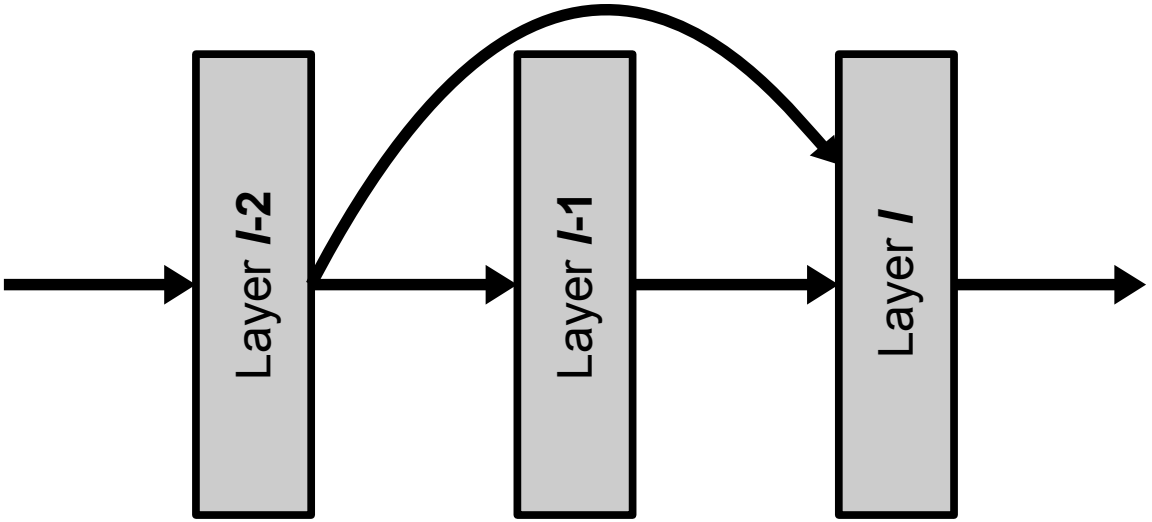
1X1 Convolution



- Capture cross-channel correlation
- Does not capture spatial correlation
- Reduces the number of channels in the next layer
  - If N < C

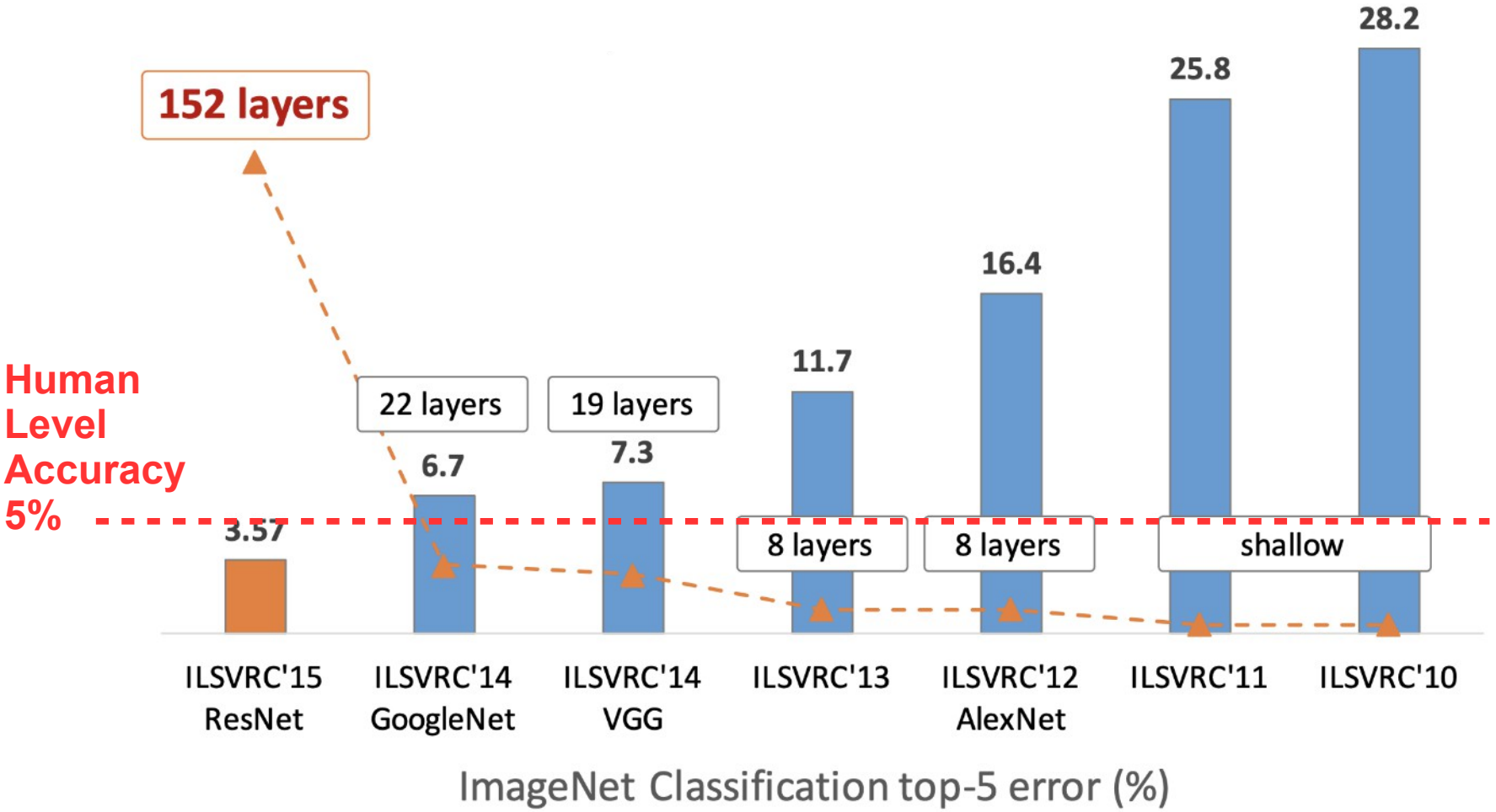
Residual Connection

- Feed-forward connection that connects to layers beyond the immediate next layer
- Improve the training of DNN with many layers
  - Address the *vanishing gradient* problem
- Introduced and used in ResNet



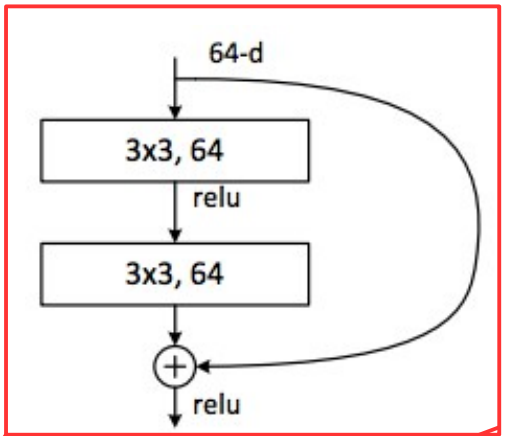
ResNet

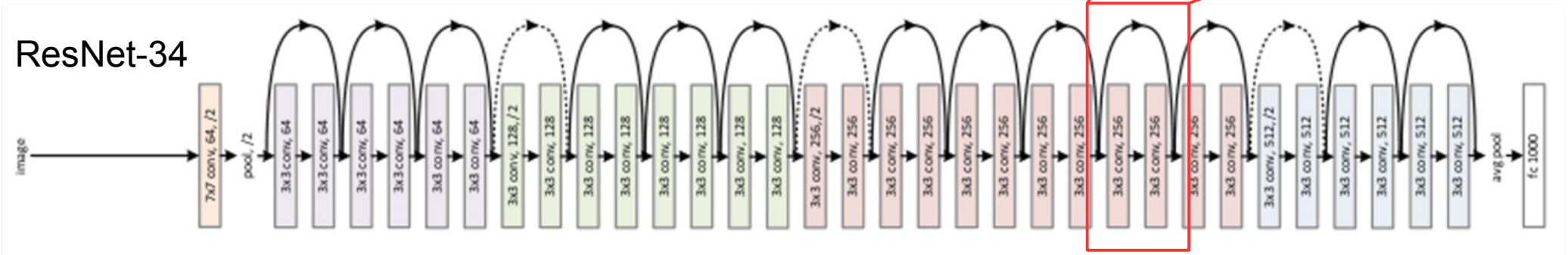
- ILSVRC15 Winner (ResNet-152)
  - Better than human level accuracy!



ResNet-50

- 50 layers (49 CONV, 1 FC)
  - Also, 34,152 and 1202 layer versions
- 25.5M parameters, 3.9G MACs





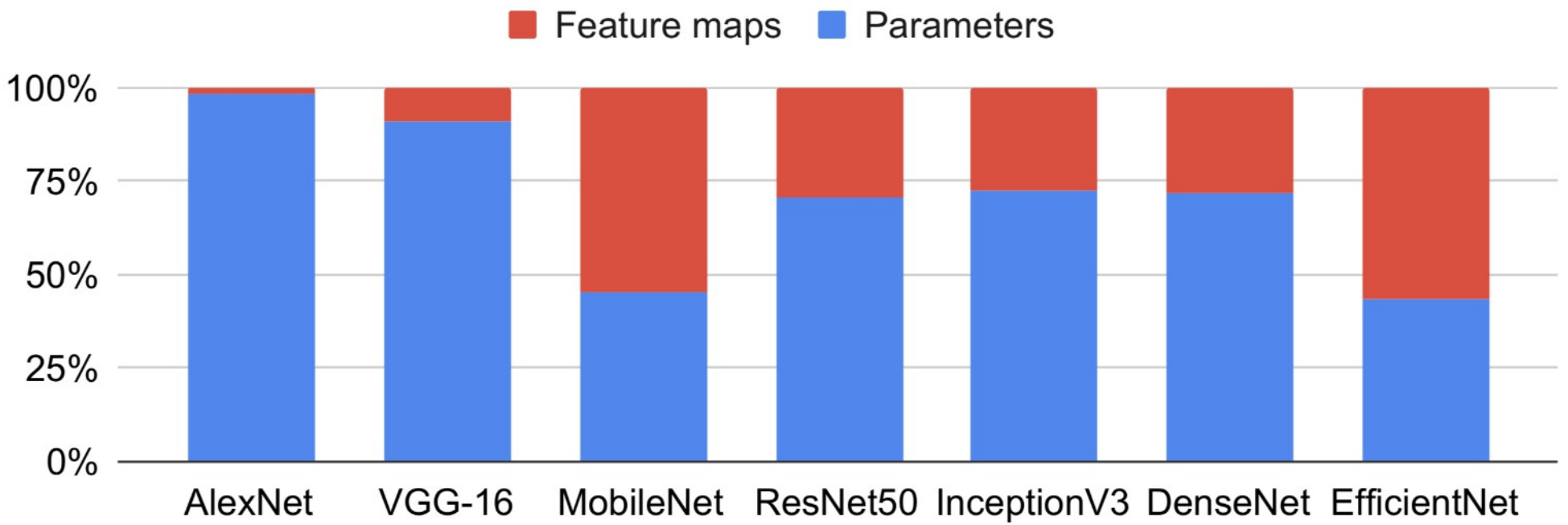
Trends

- Increase of the depth
- Increase in the number of filter shape
  - Flexibility is important
- Most of the computation on CONV layers
- Most of the parameters in FC layers
  - Even if FC layers have been replacing by CONV

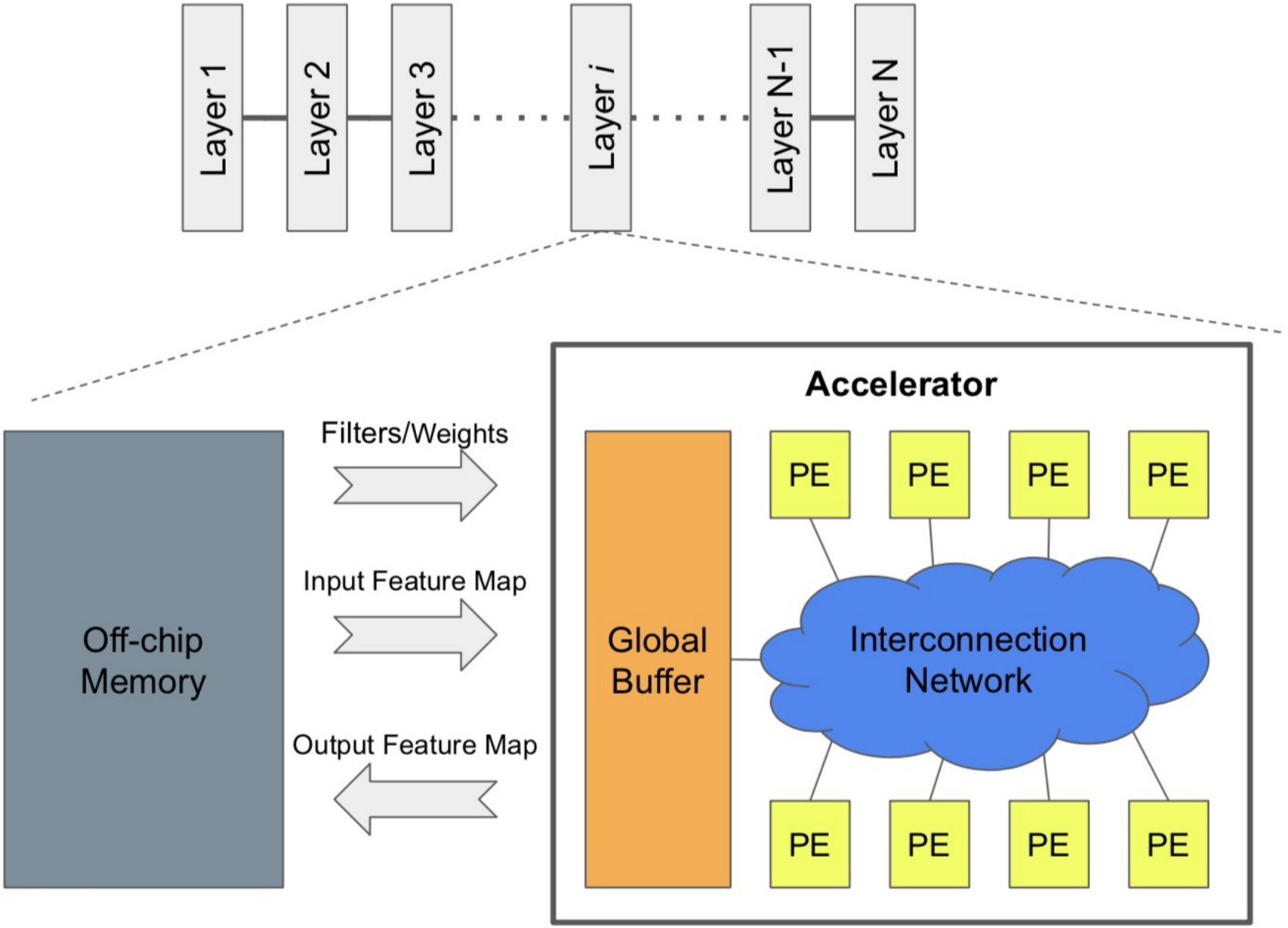
Agenda

- Deep Neural Networks
- Types of layers
- Memory and communication traffic
- Kernel computation
- Quantitative analysis

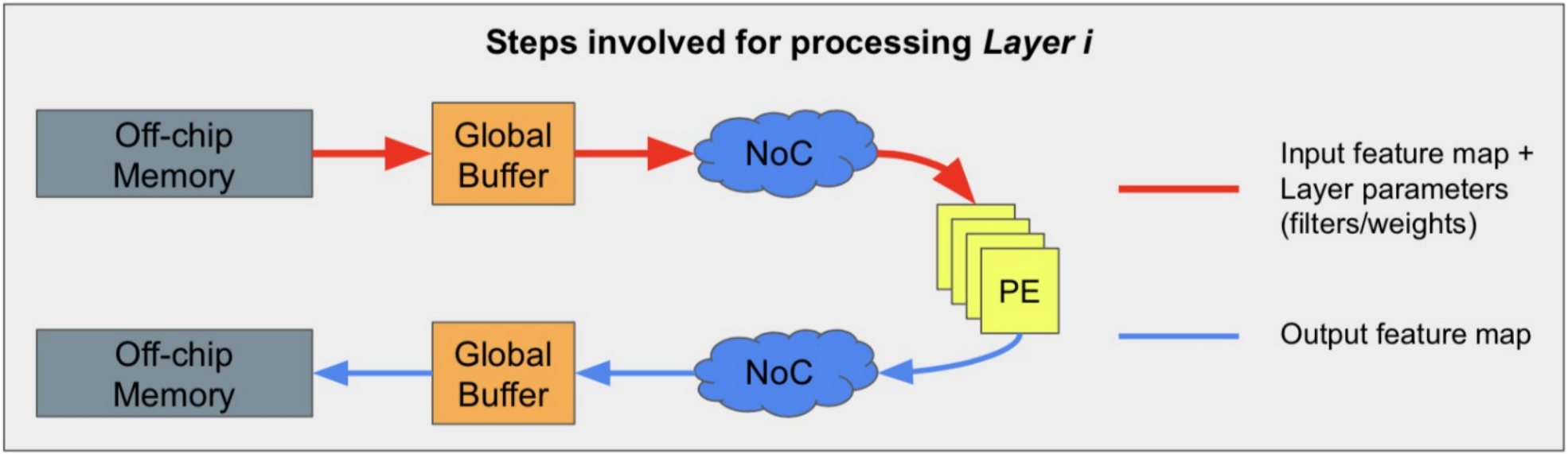
Feature Maps *vs.* Parameters



DNN Layer Computation



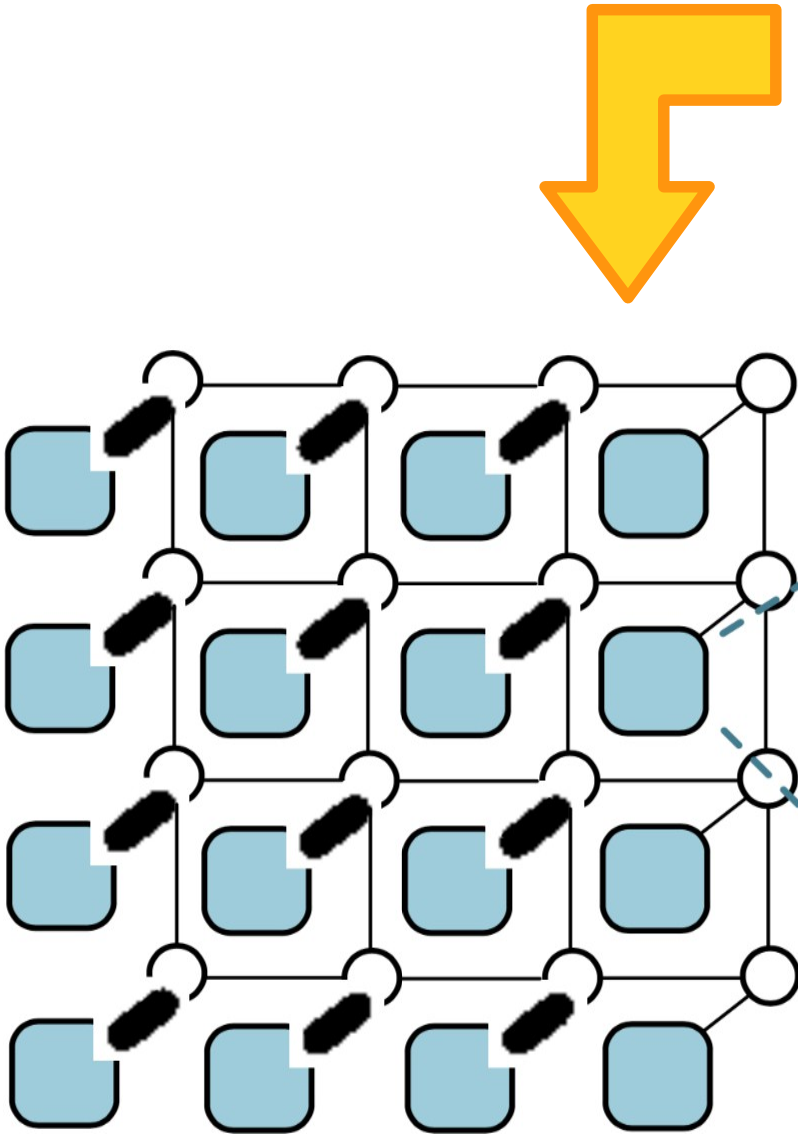
DNN Layer Computation

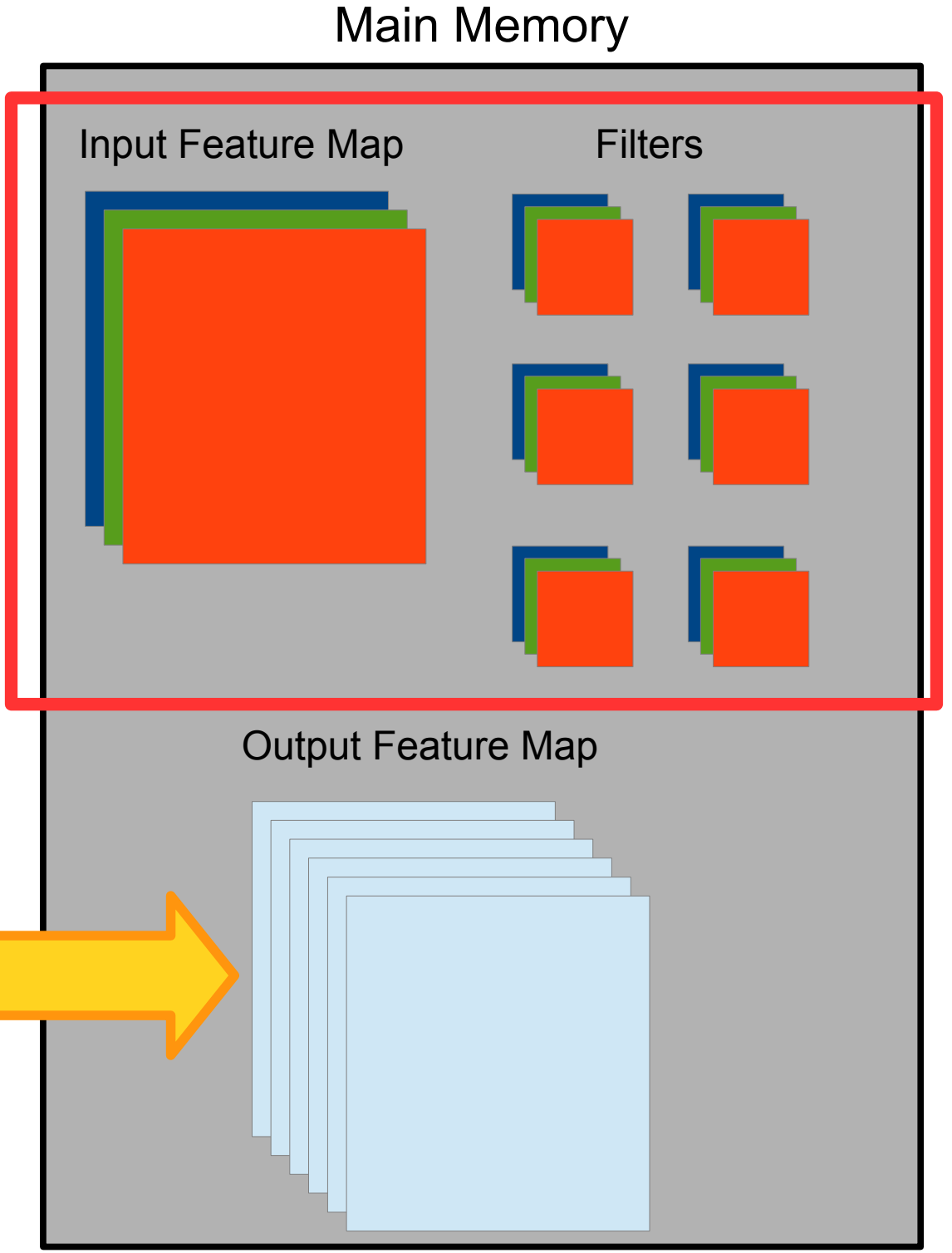


Induced Traffic

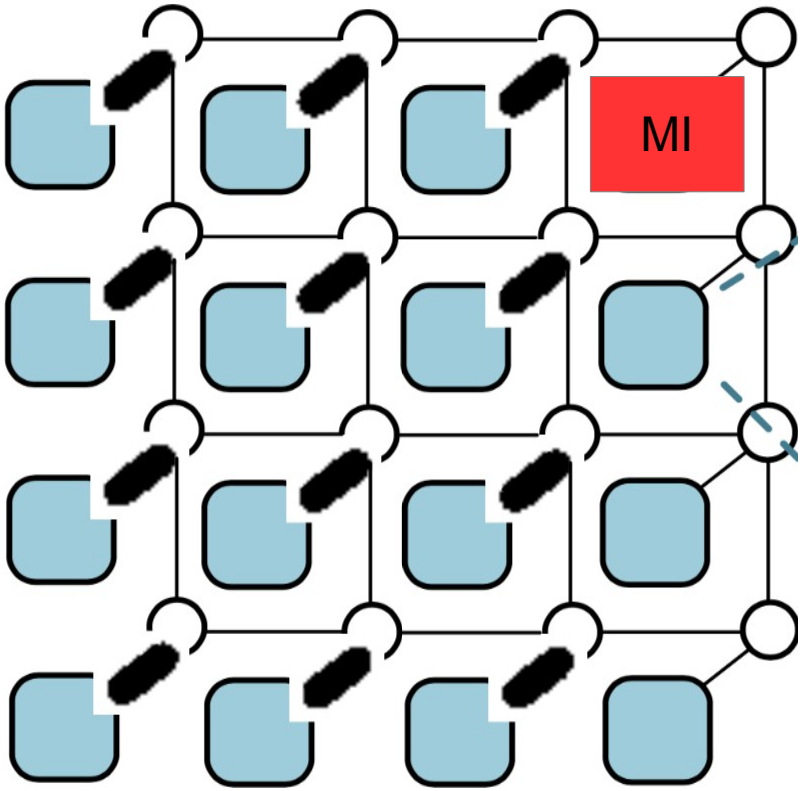
- From/to the memory
- Into the accelerator

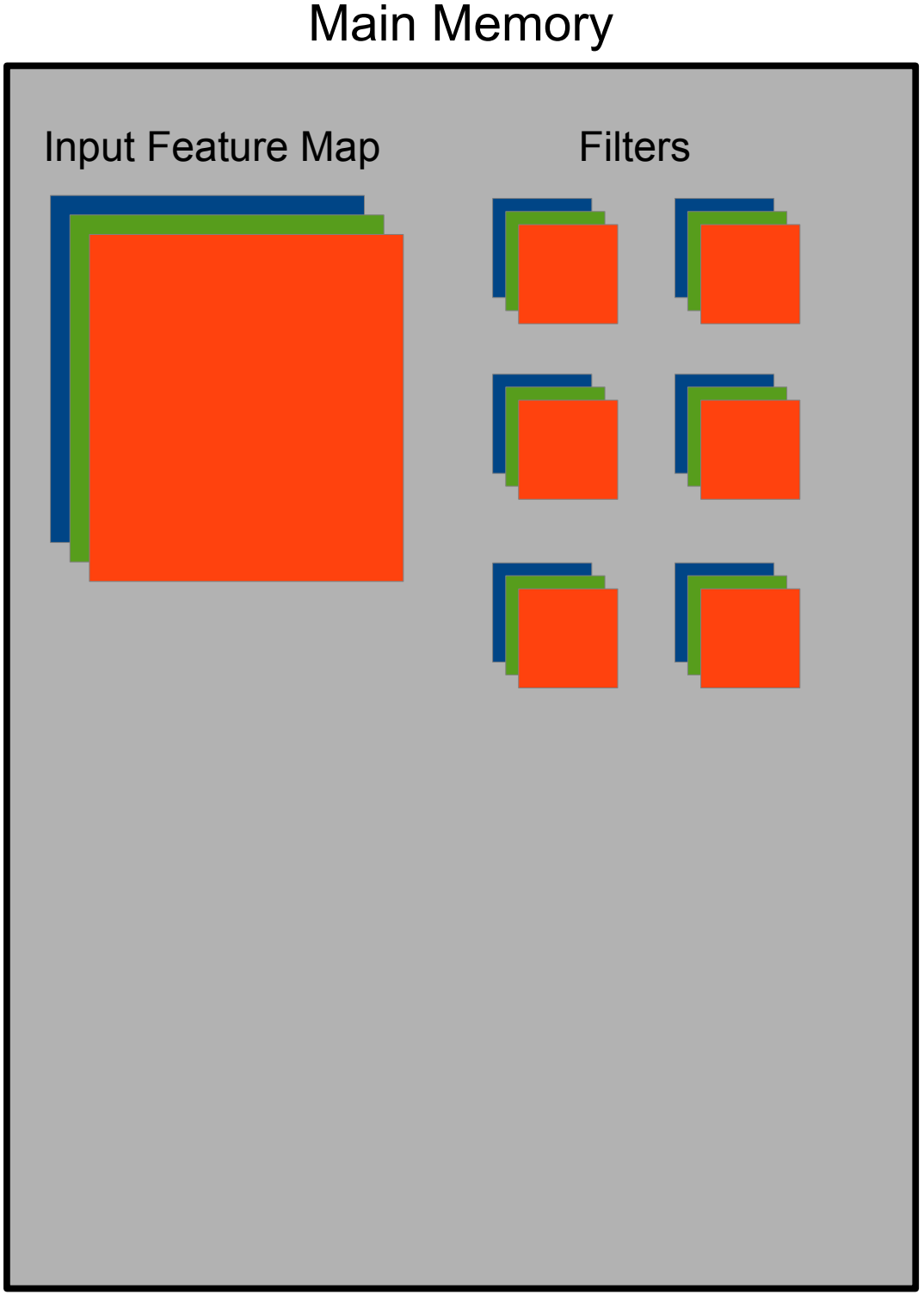
Convolution



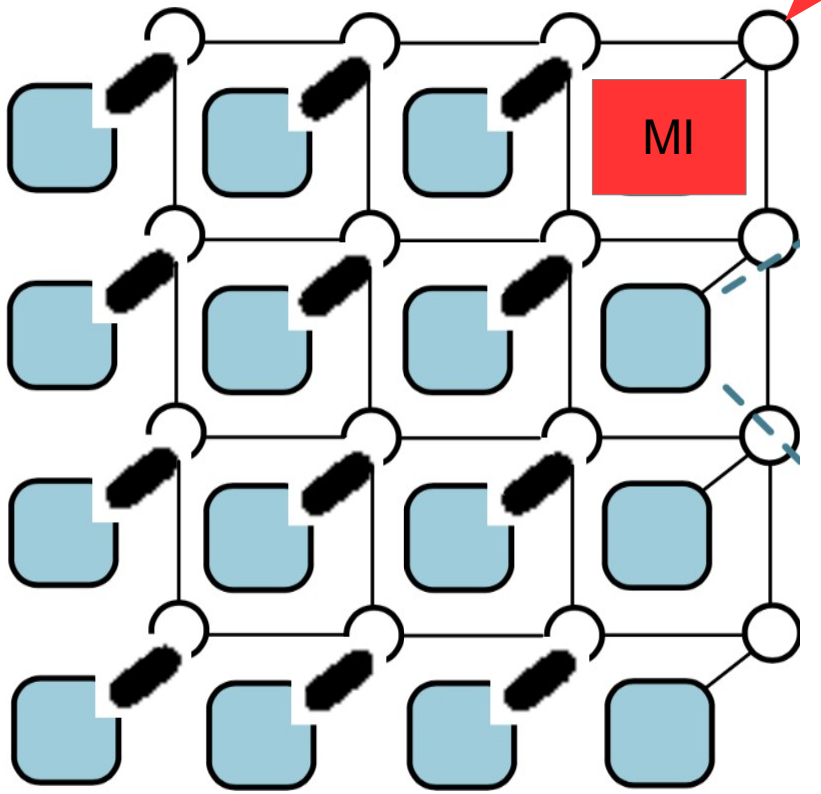


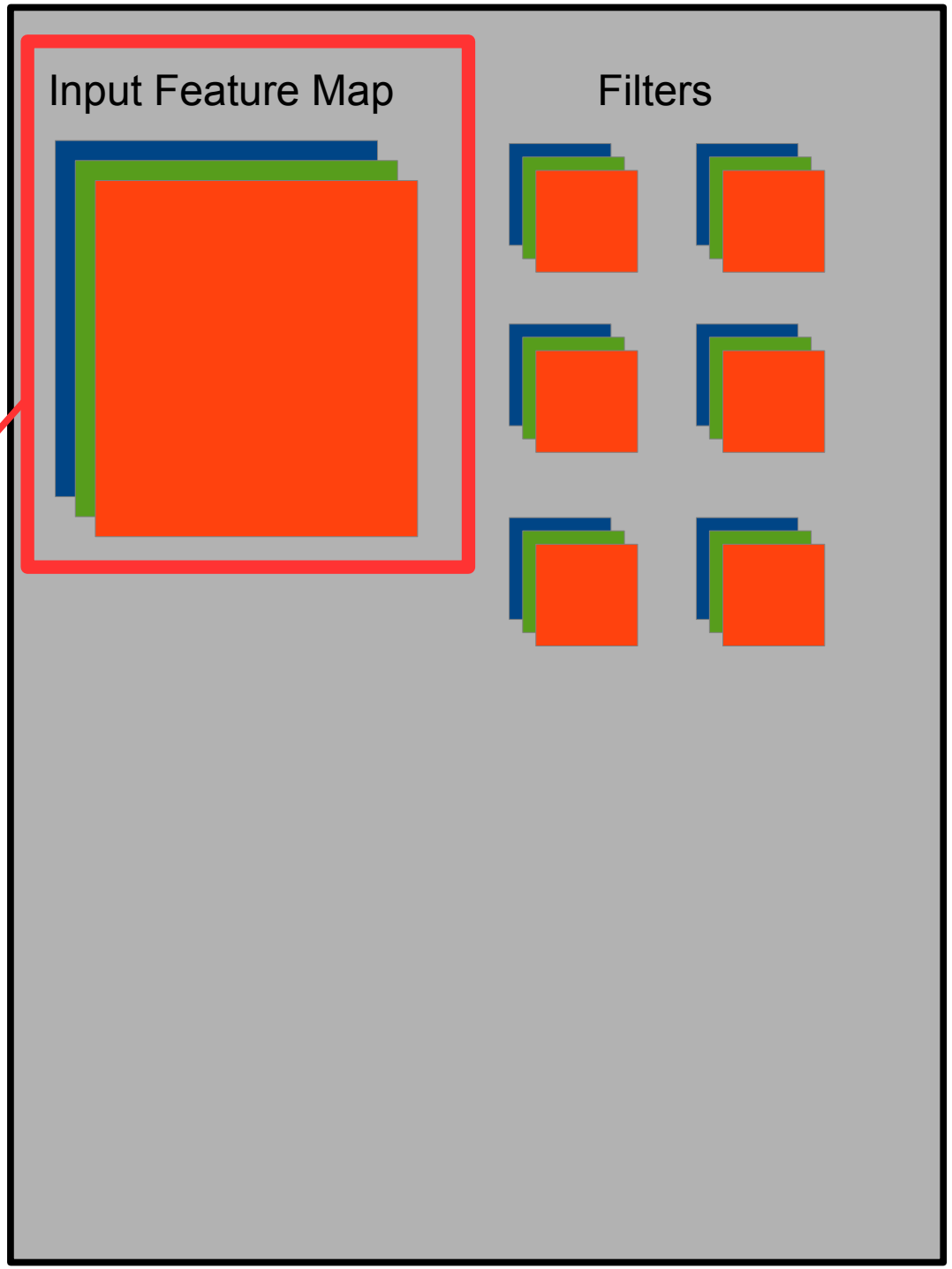
Convolution



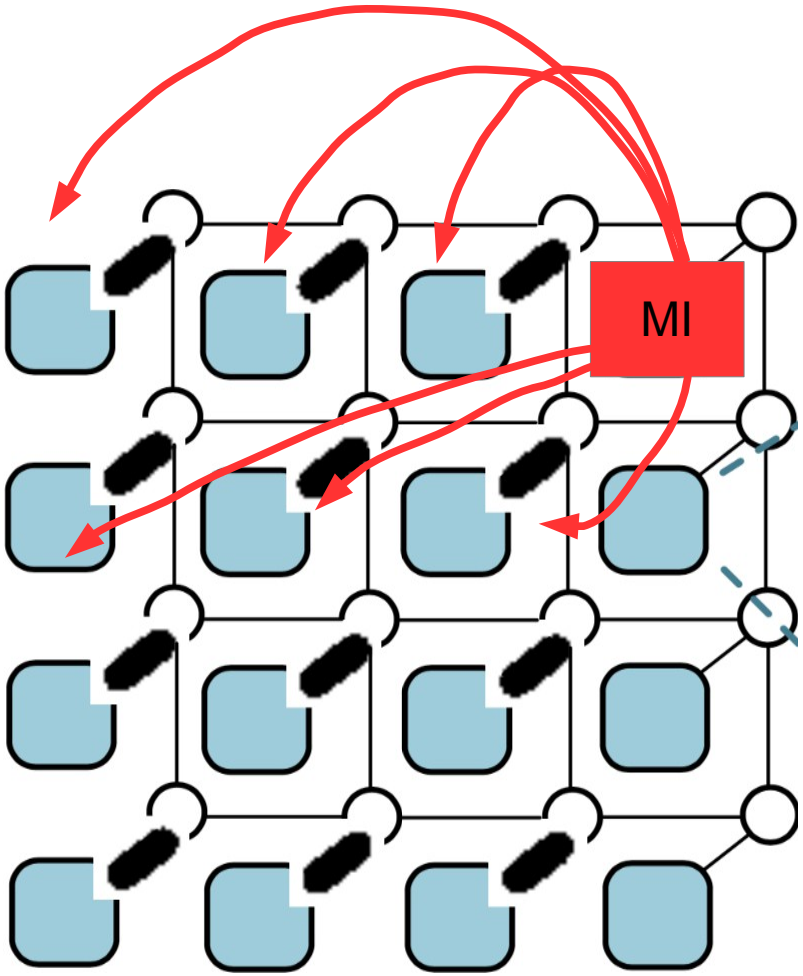


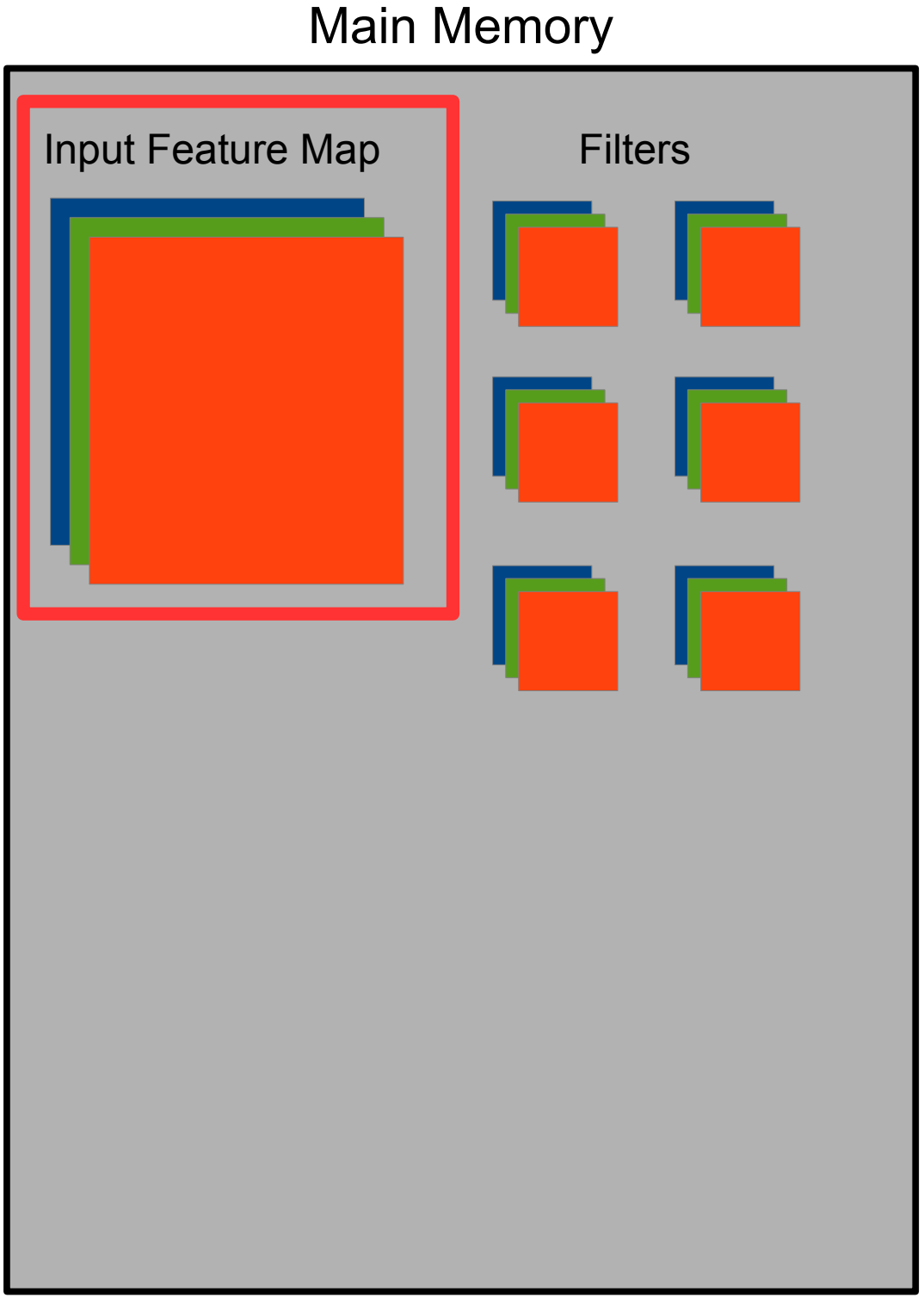
Convolution



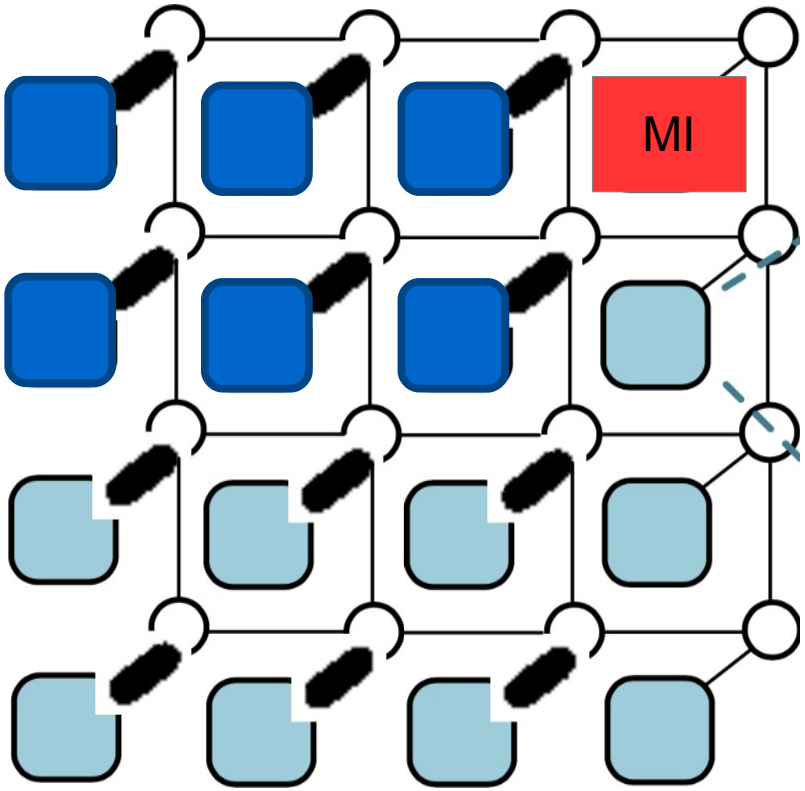


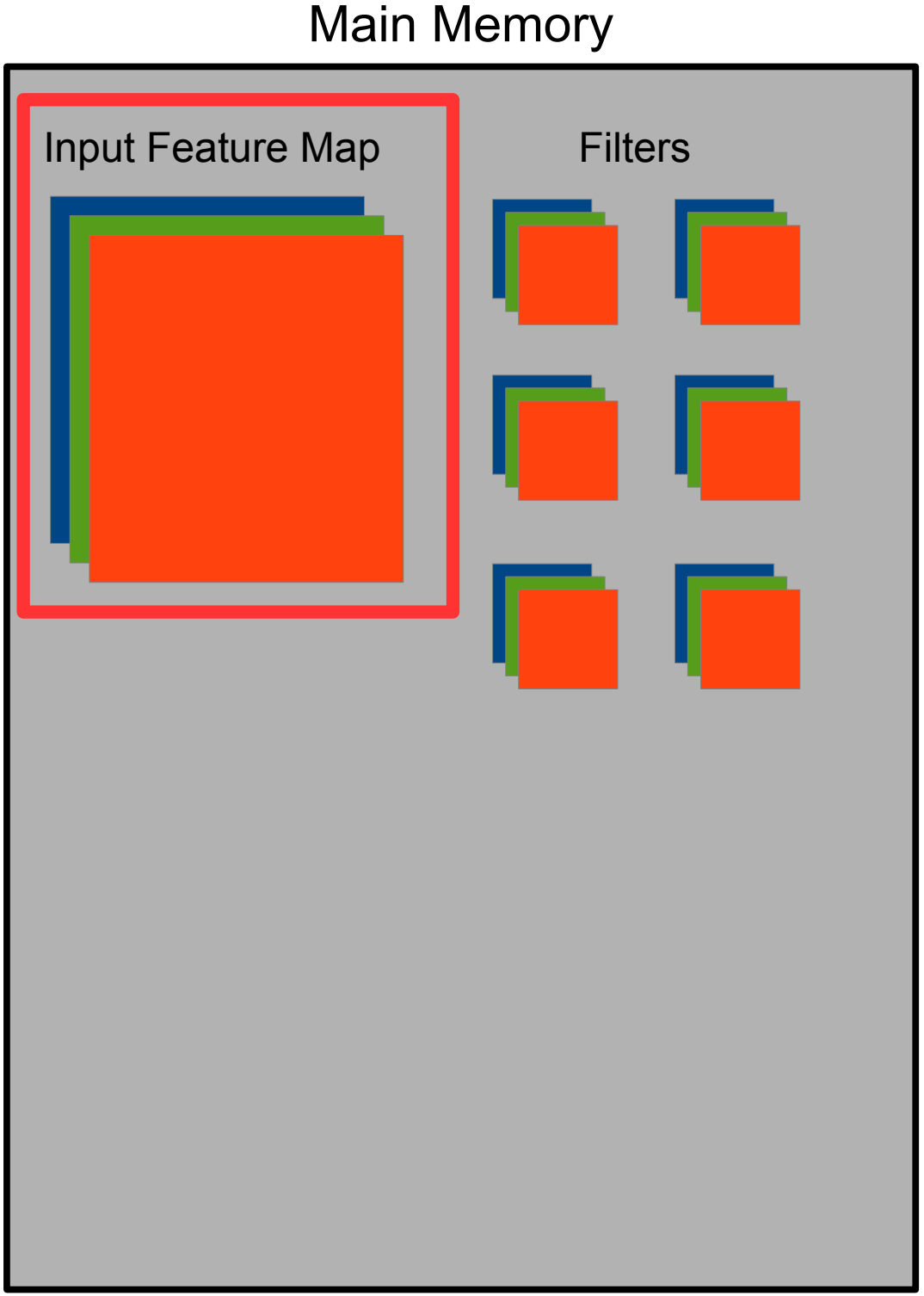
Convolution



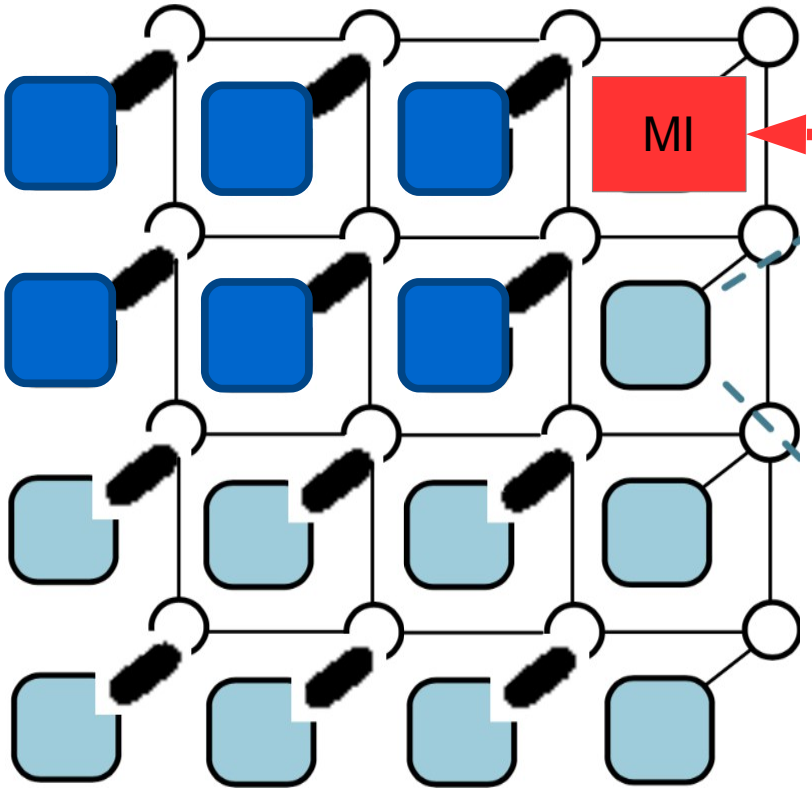


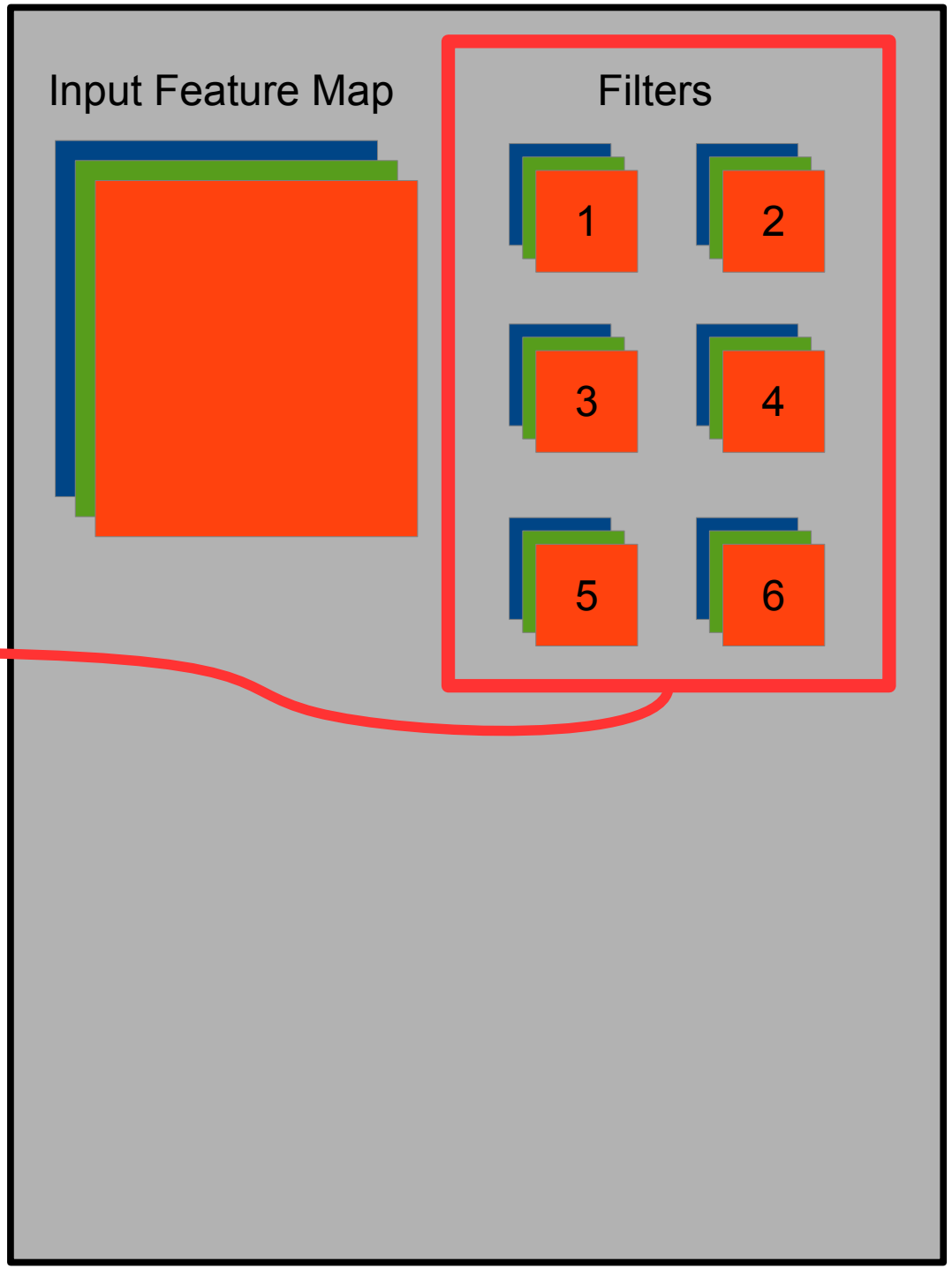
Convolution



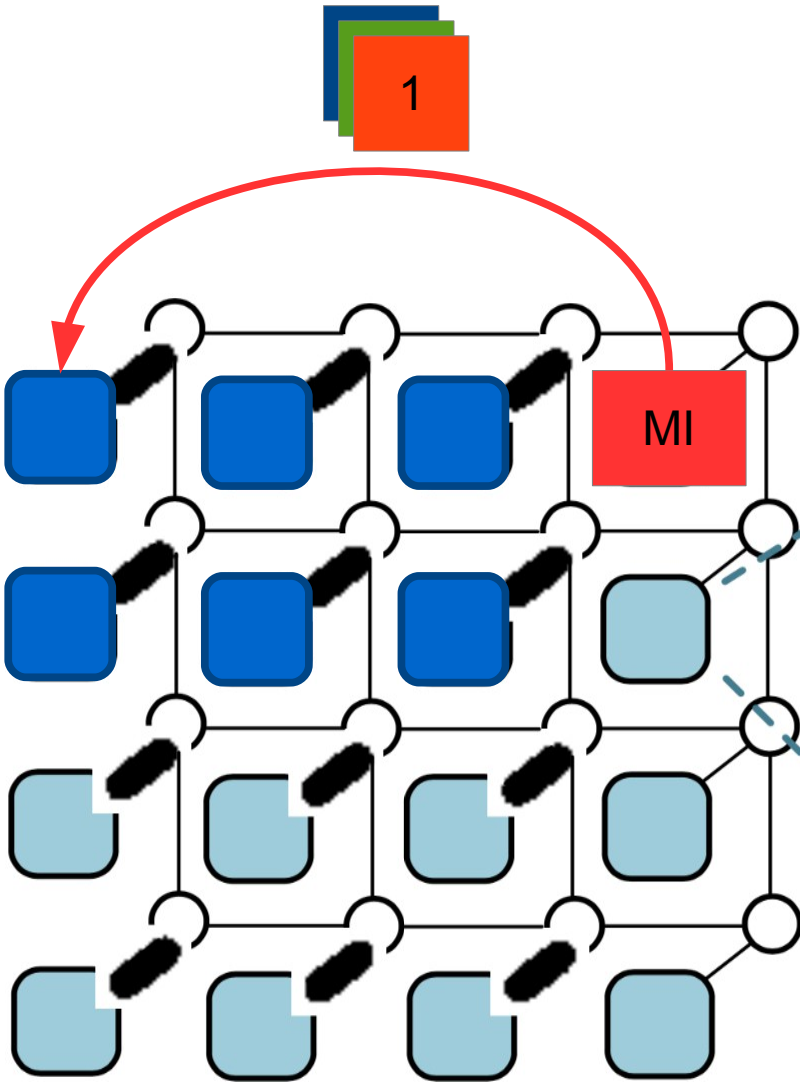


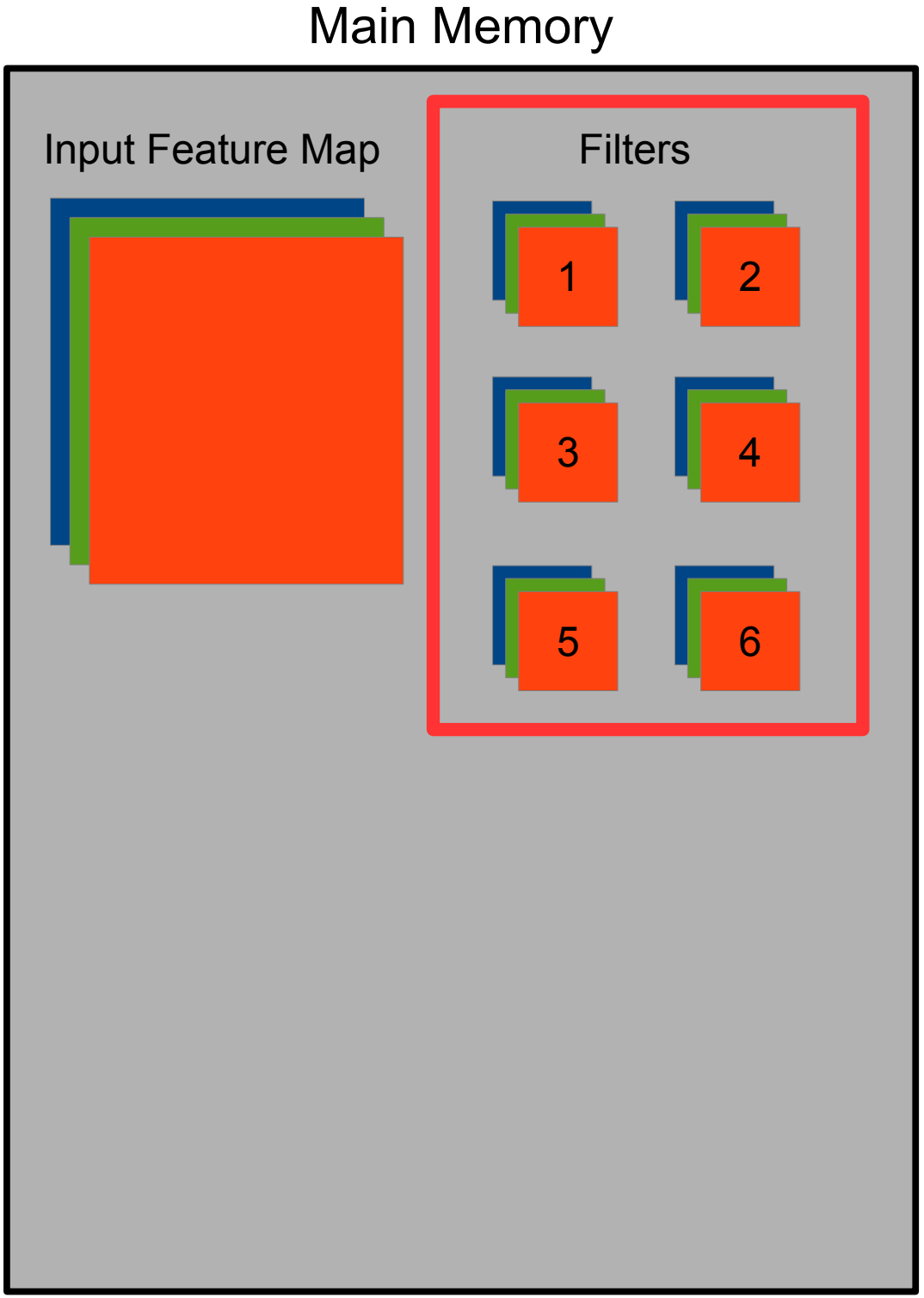
Convolution



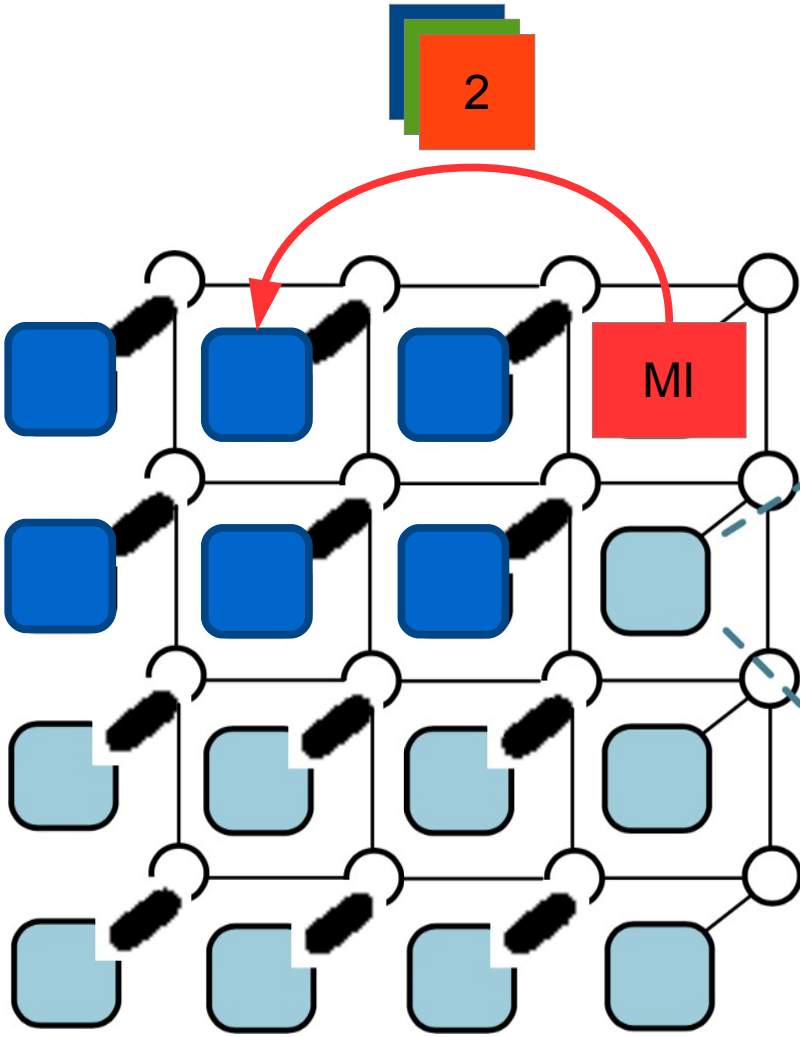


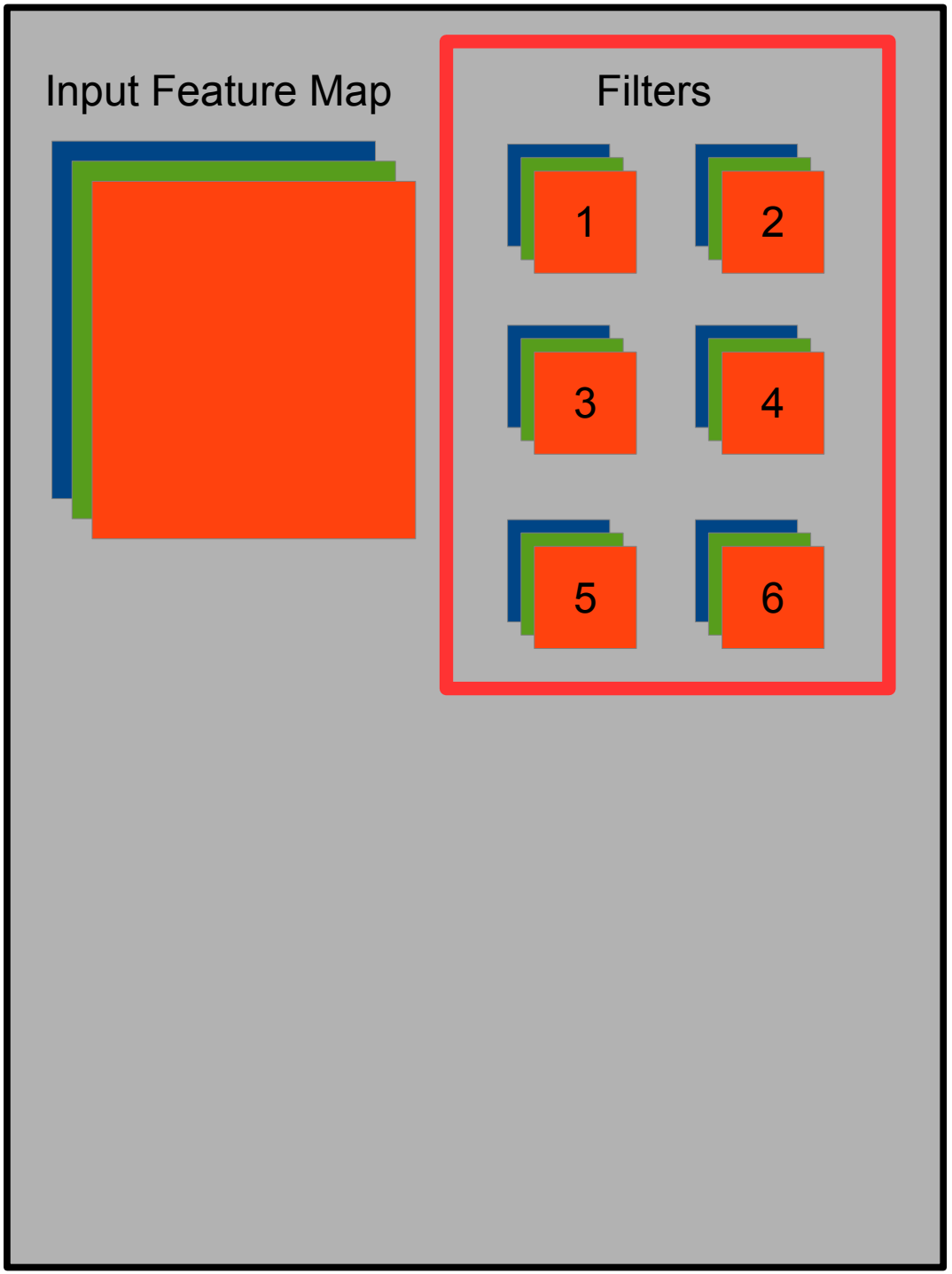
Convolution



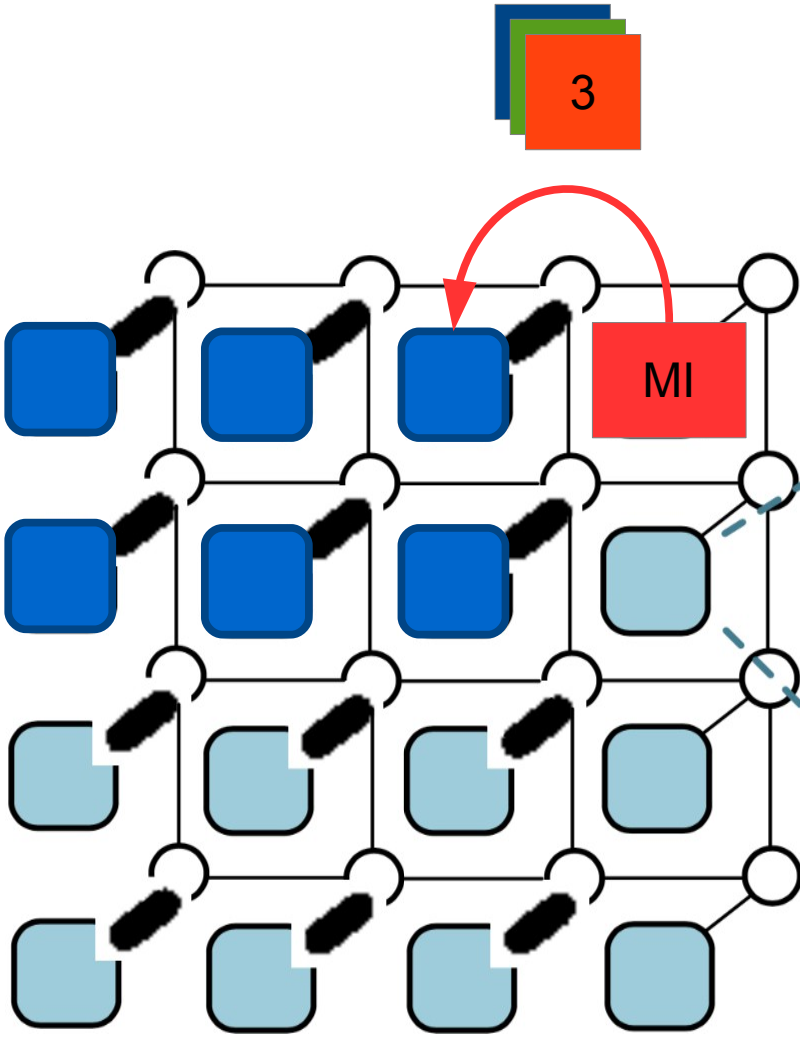


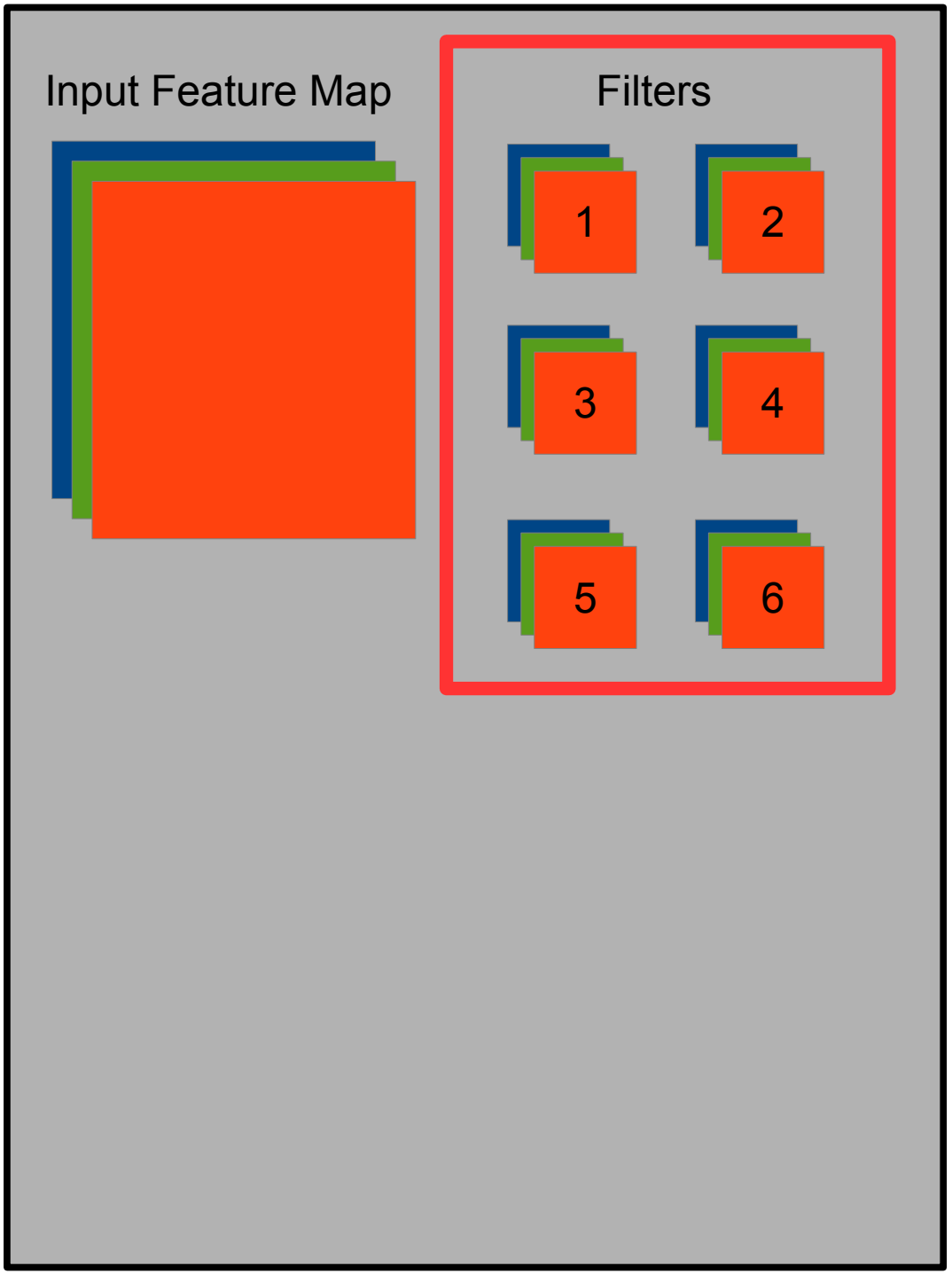
Convolution



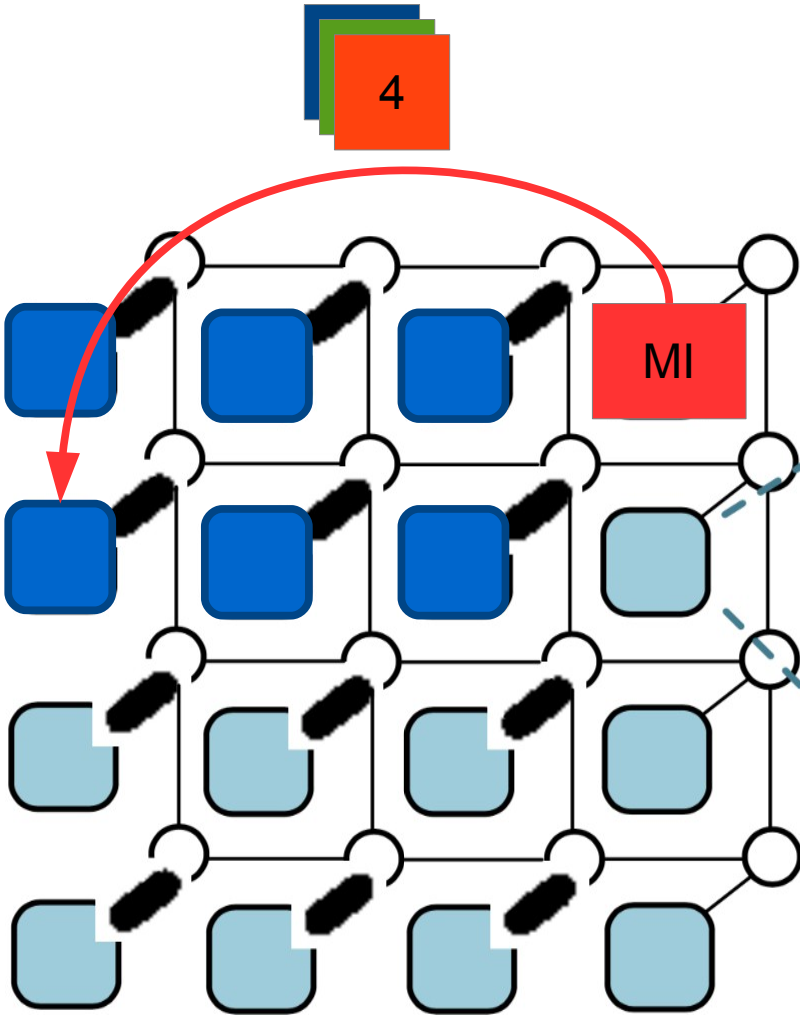


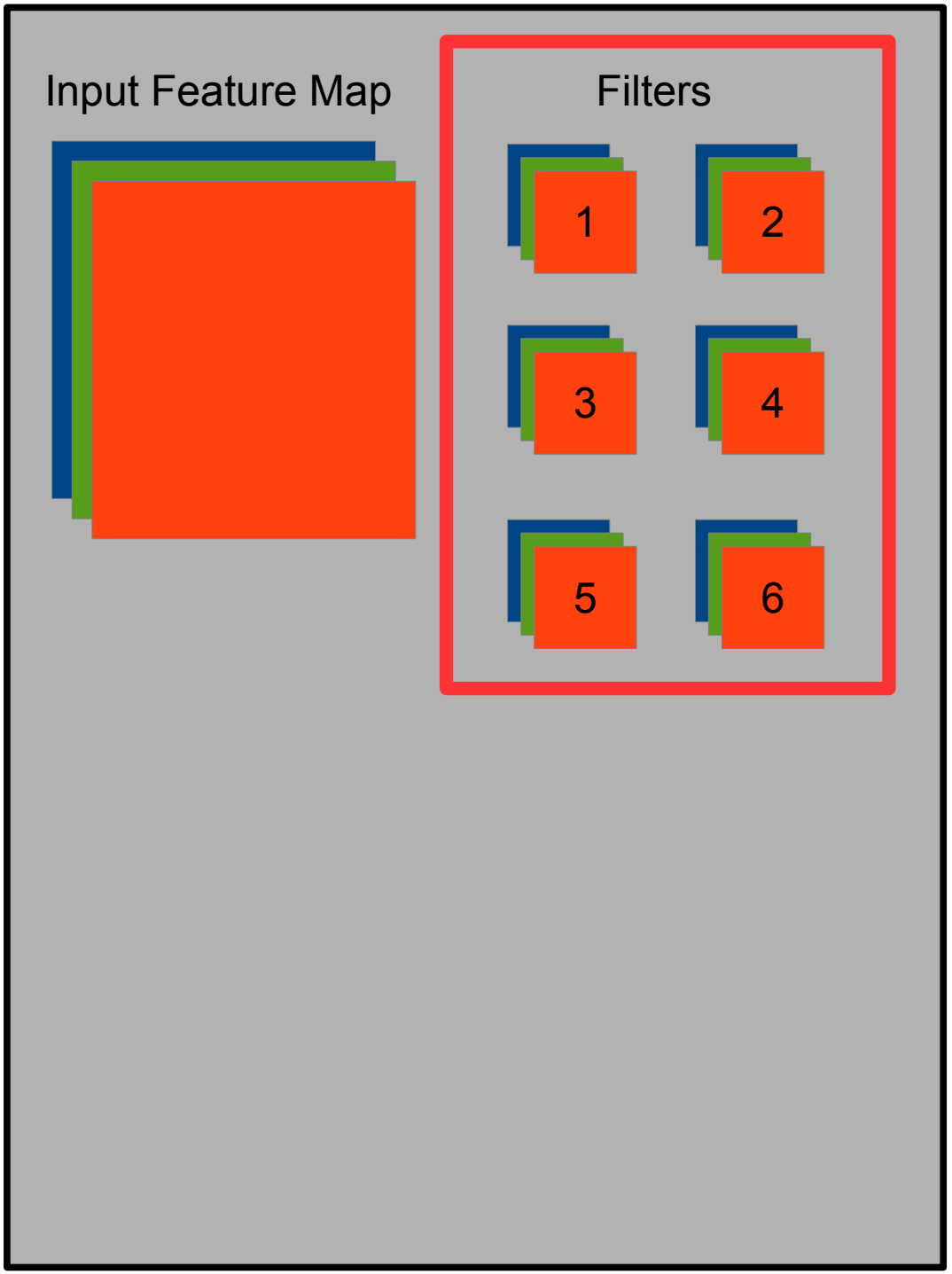
Convolution



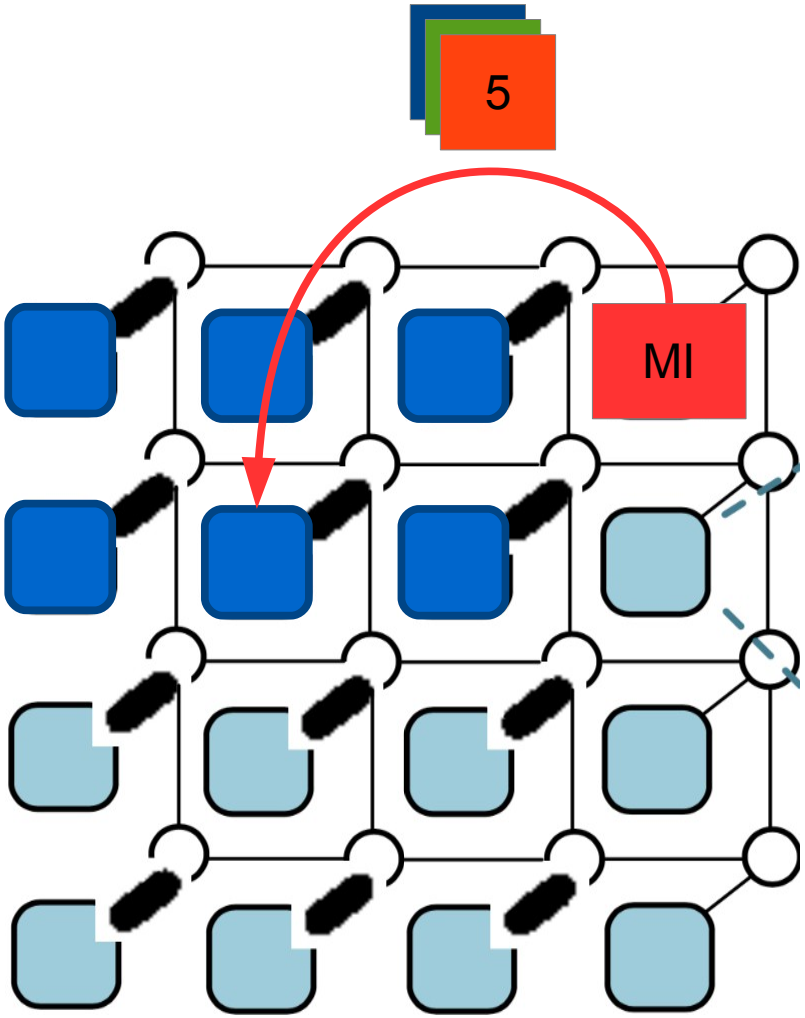


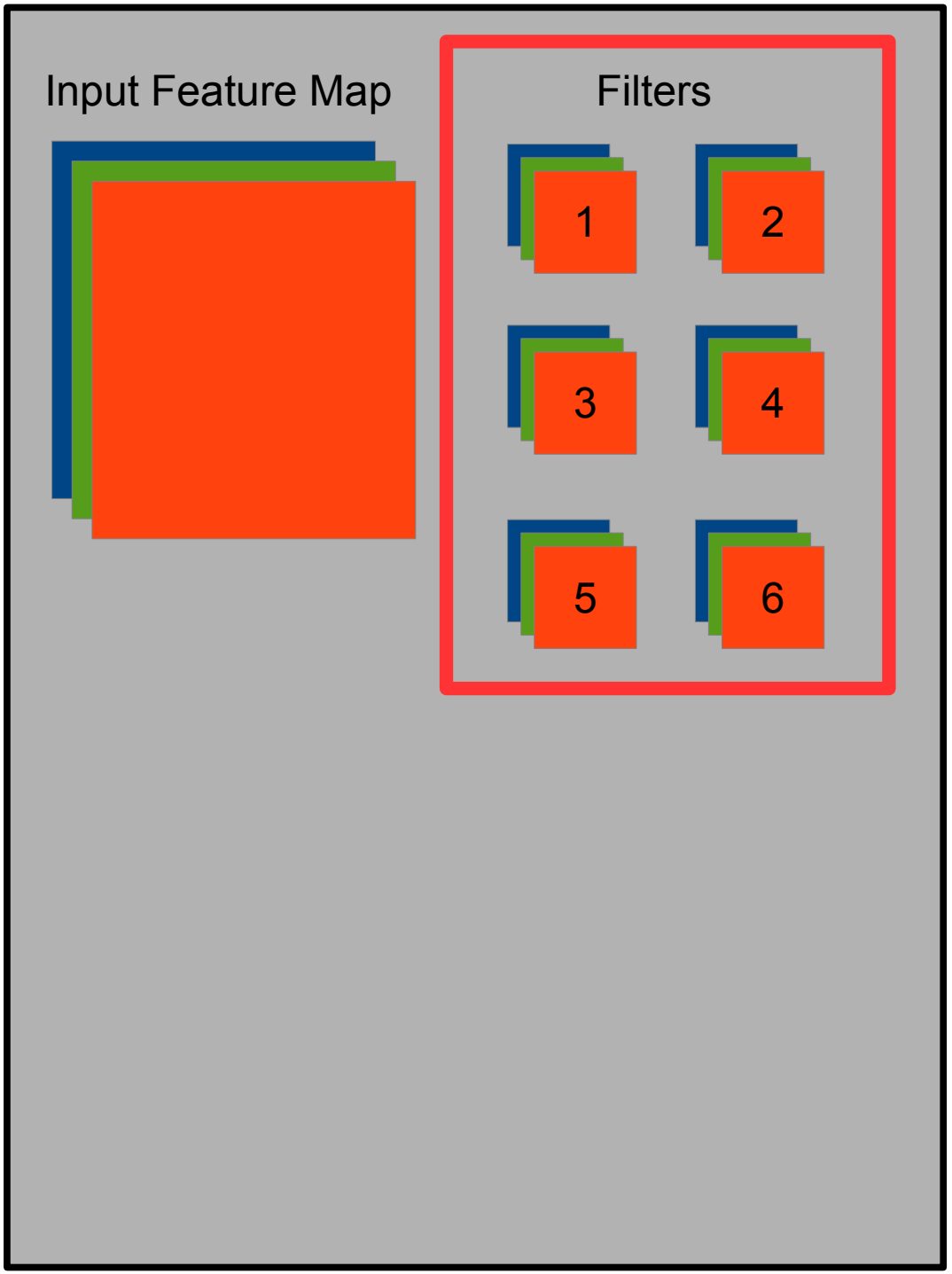
Convolution



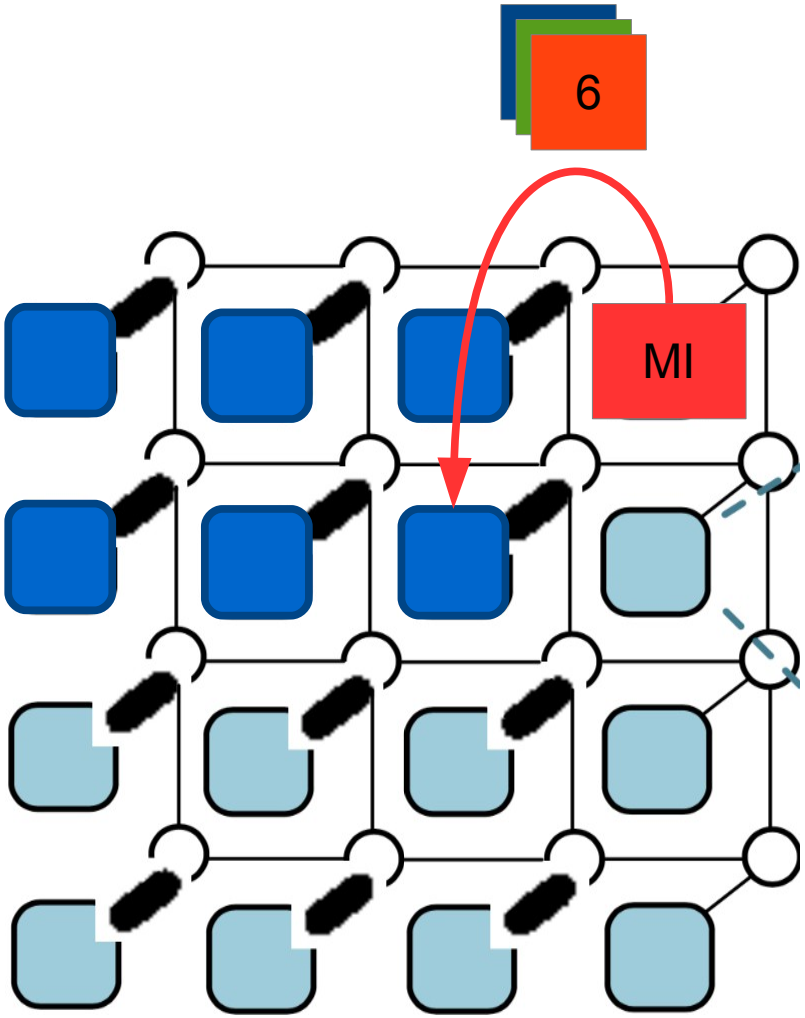


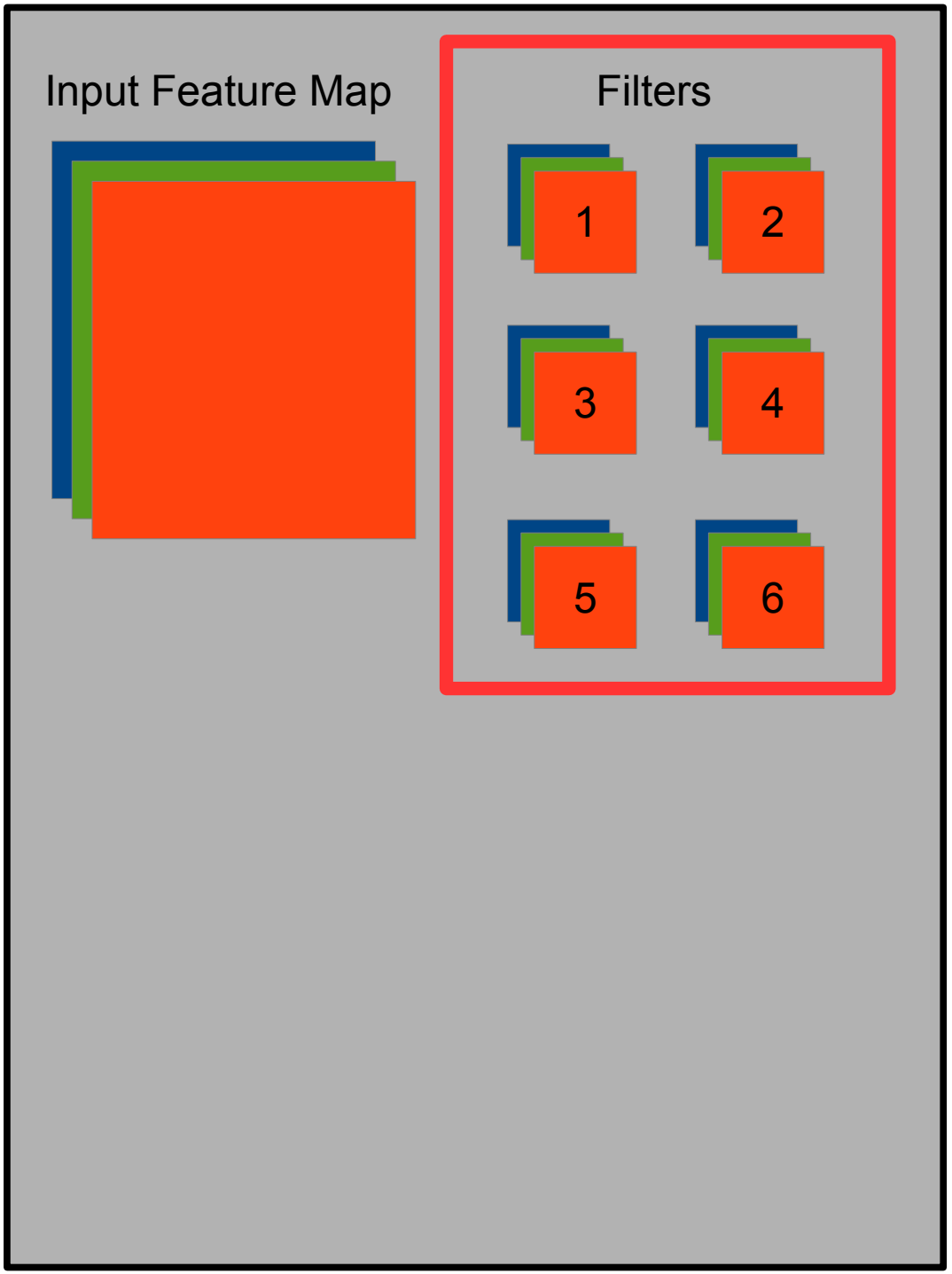
Convolution



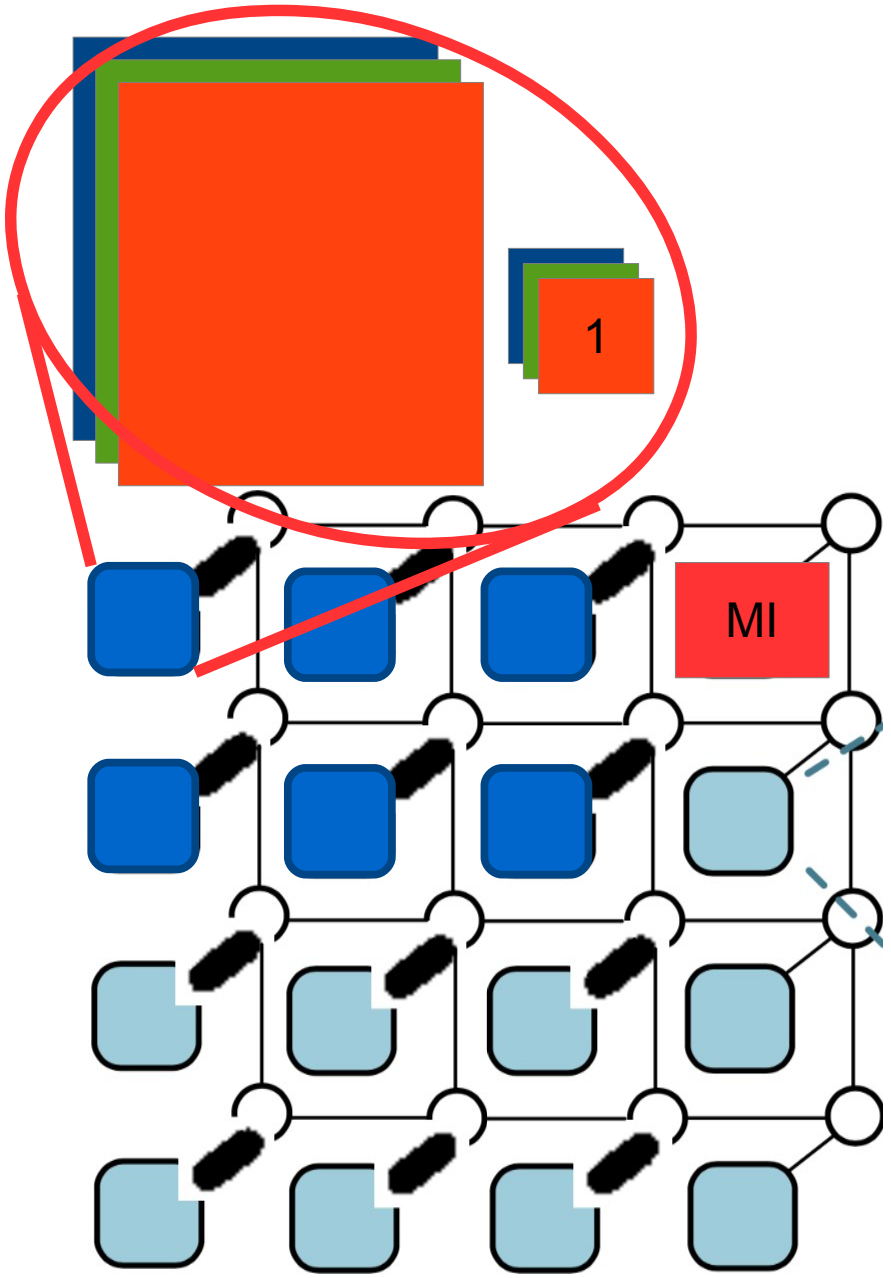


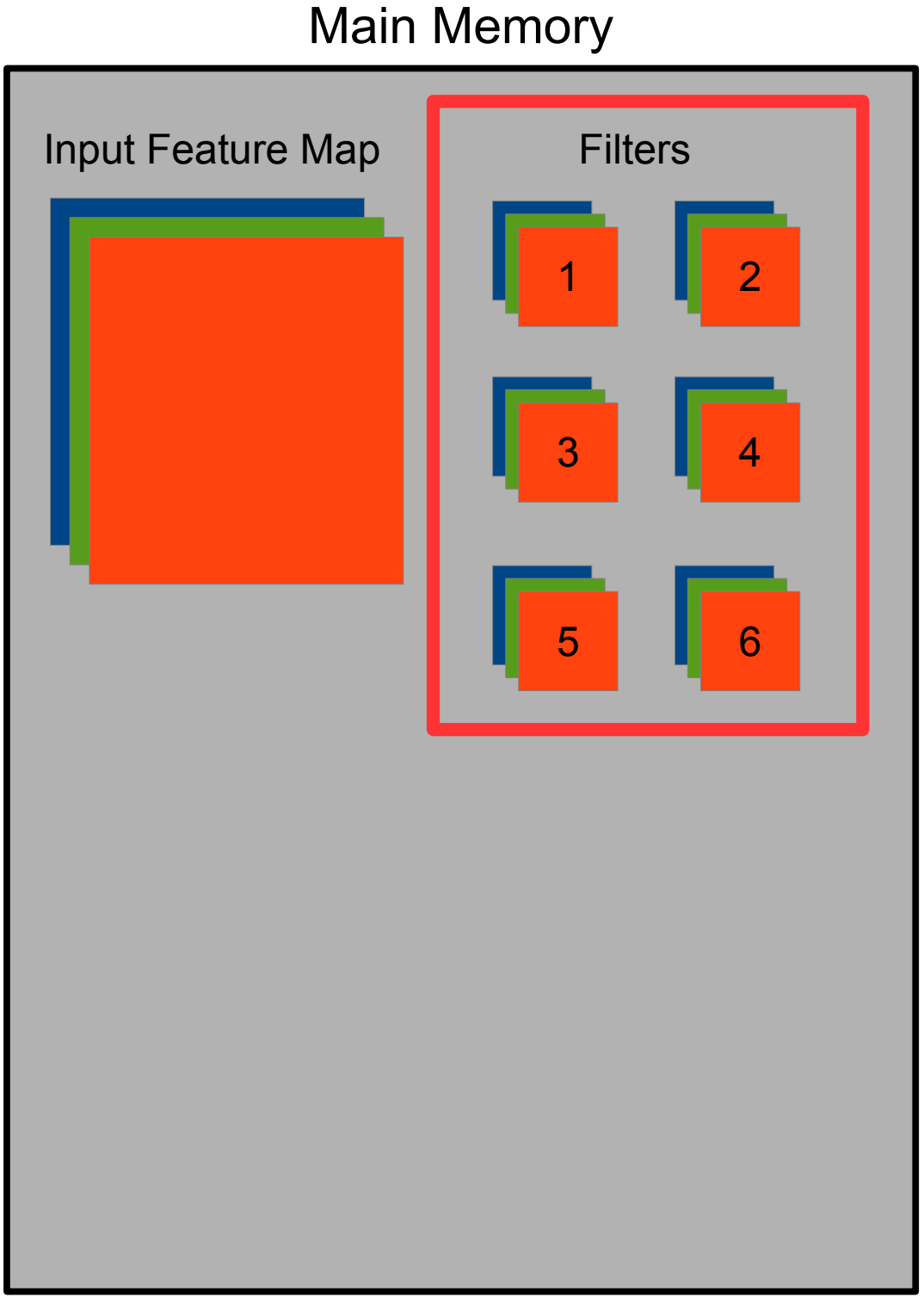
Convolution



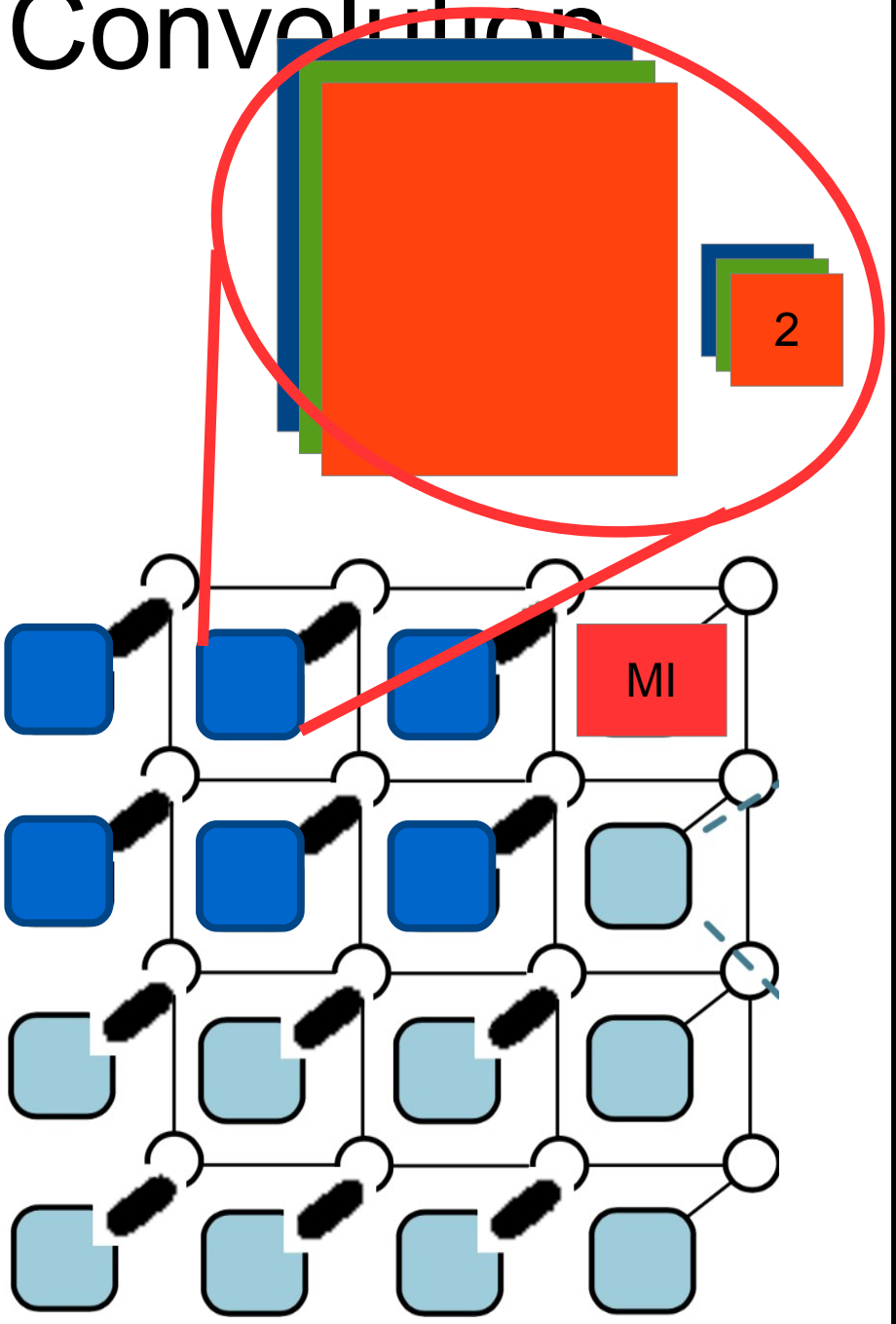


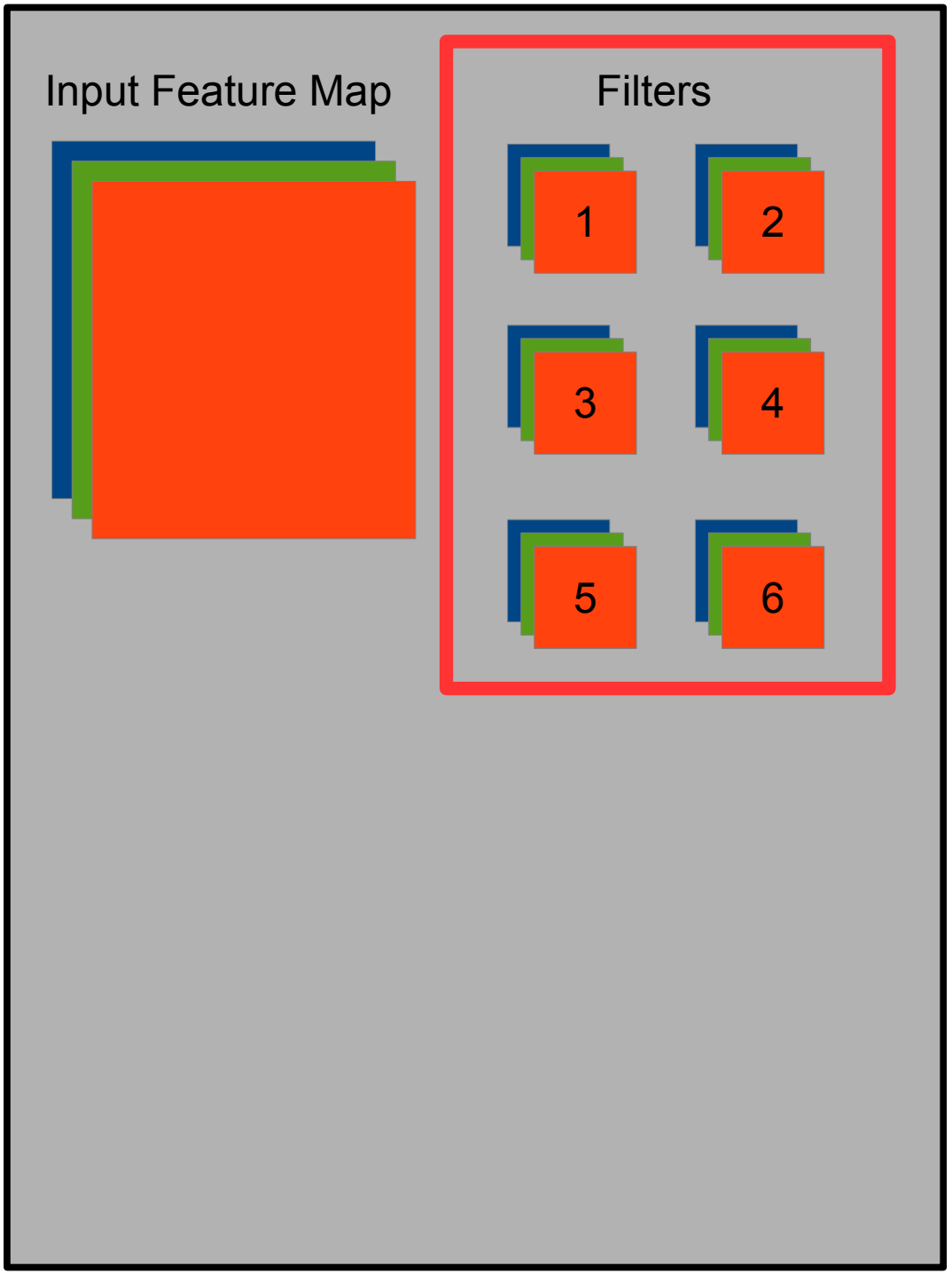
Convolution



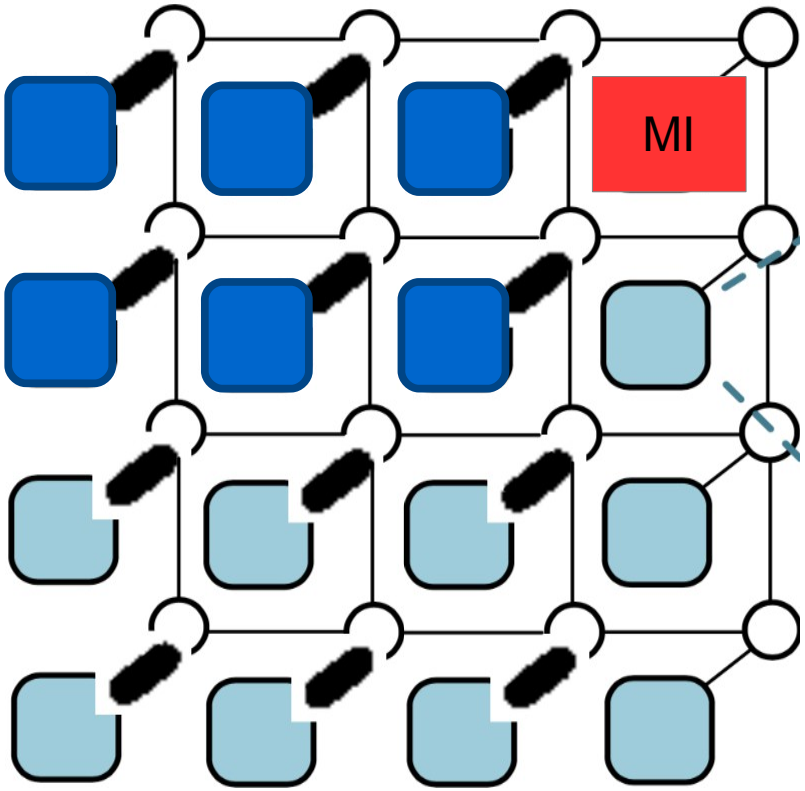


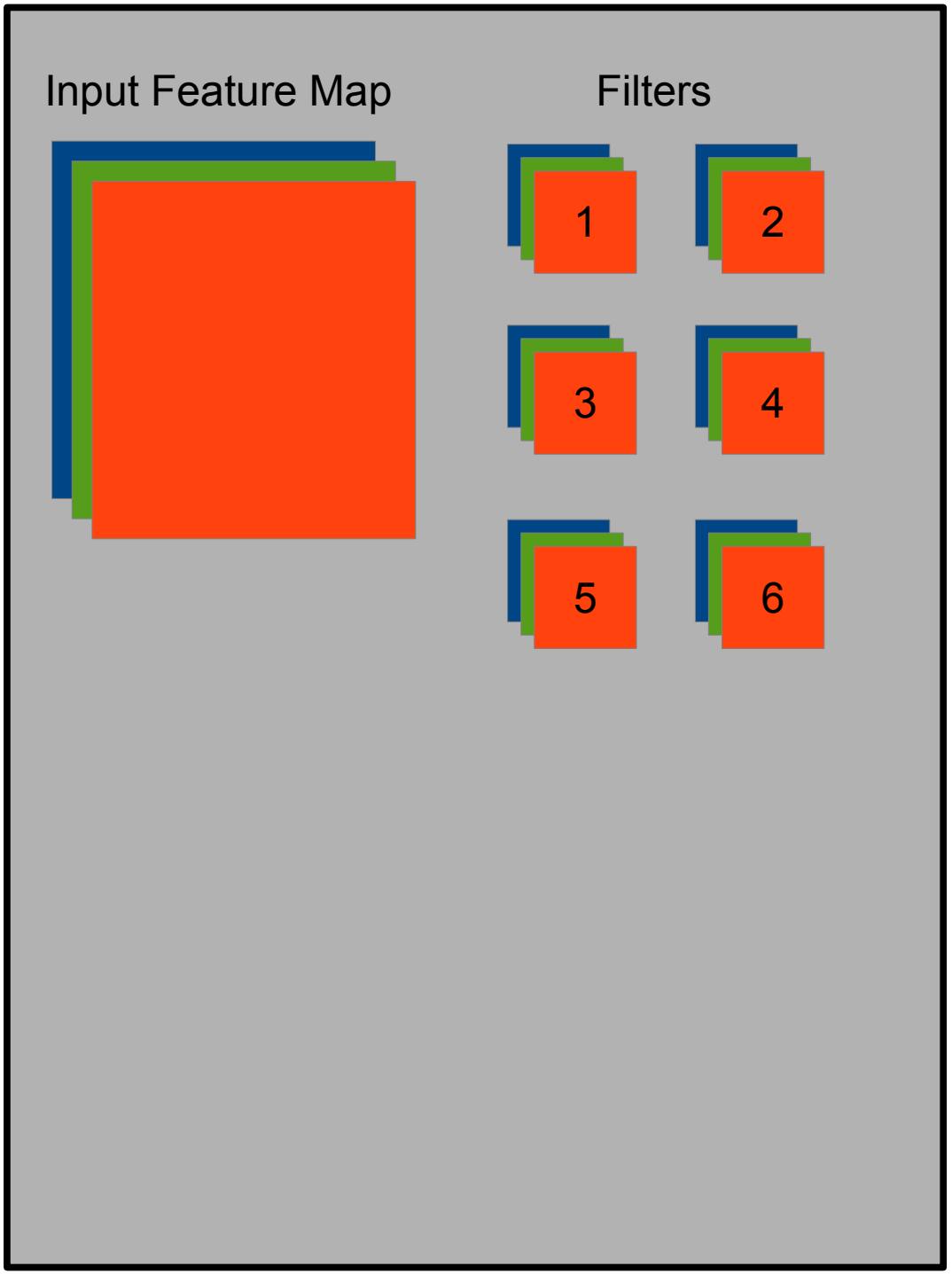
Convolution



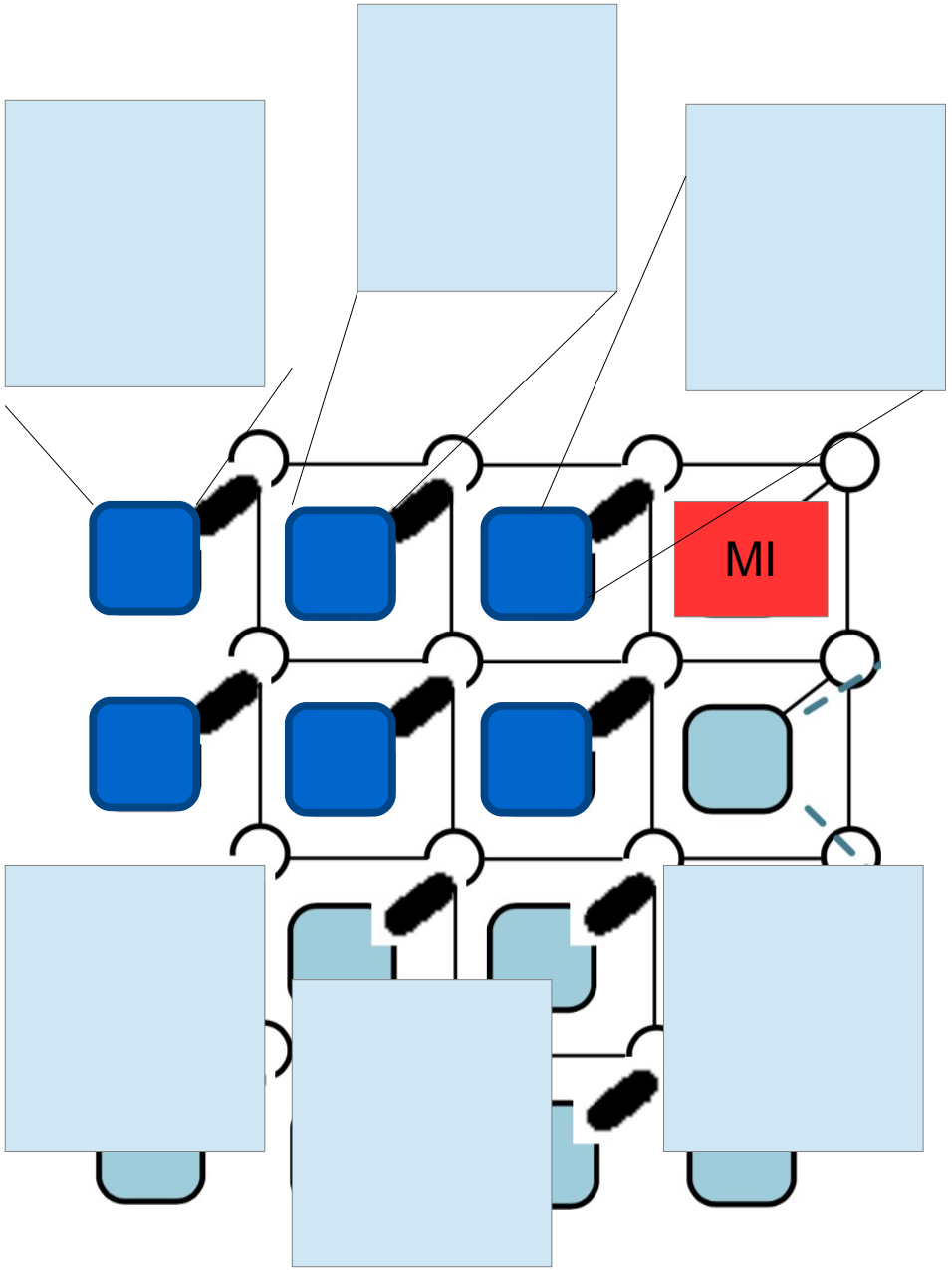


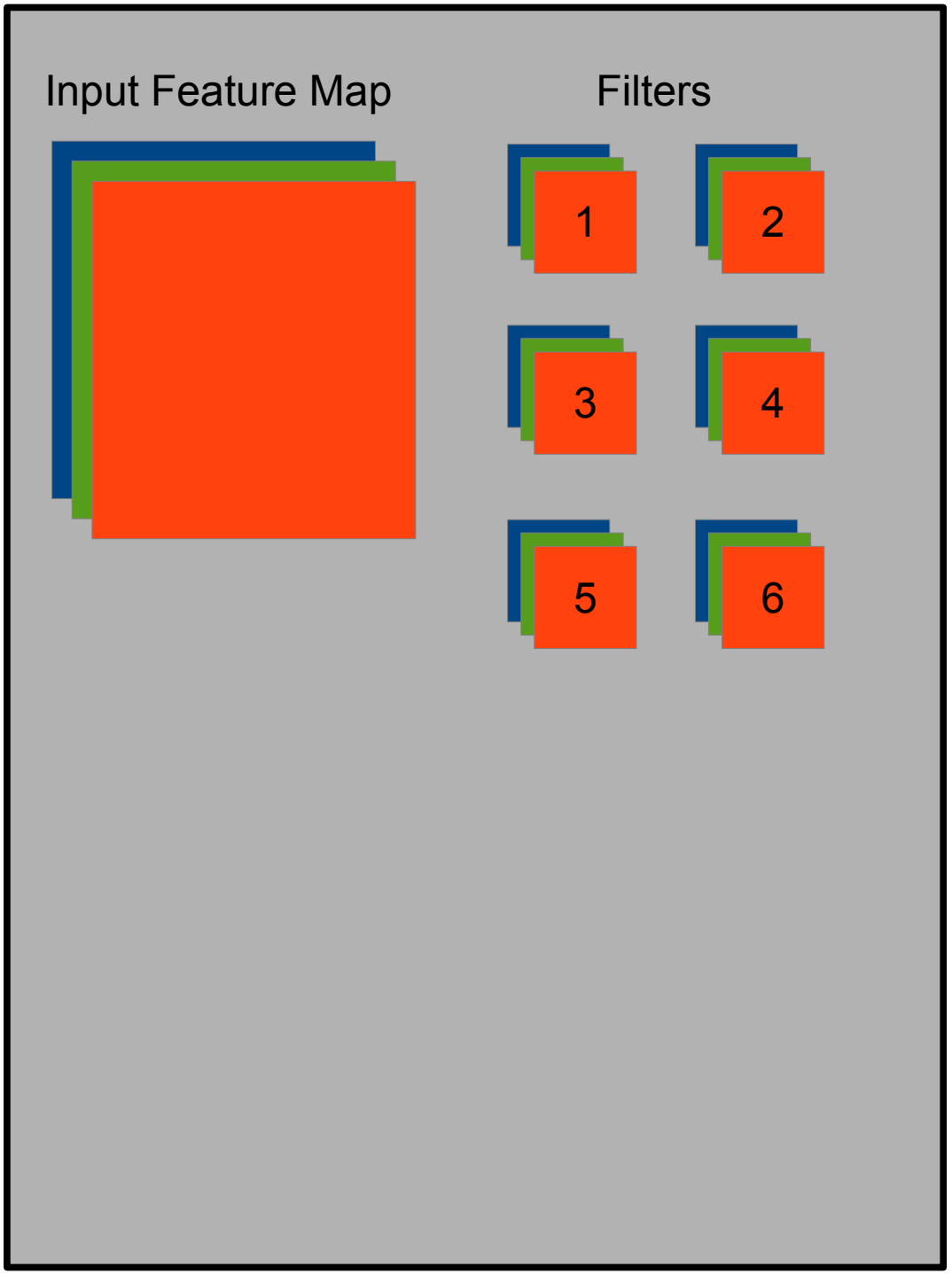
Convolution



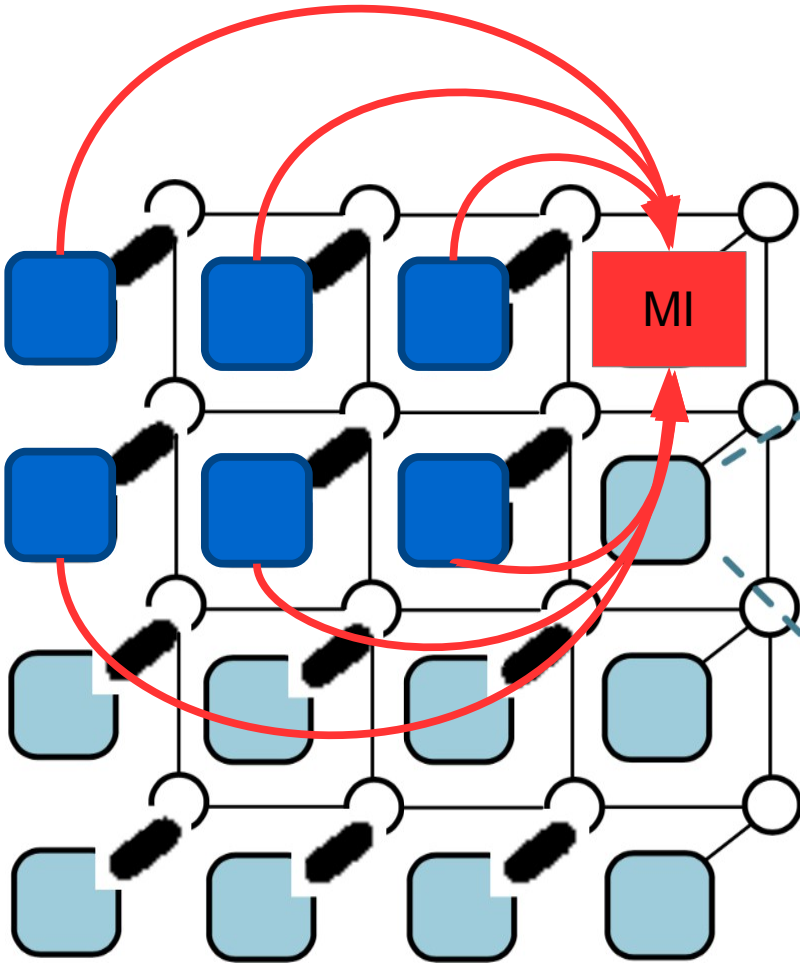


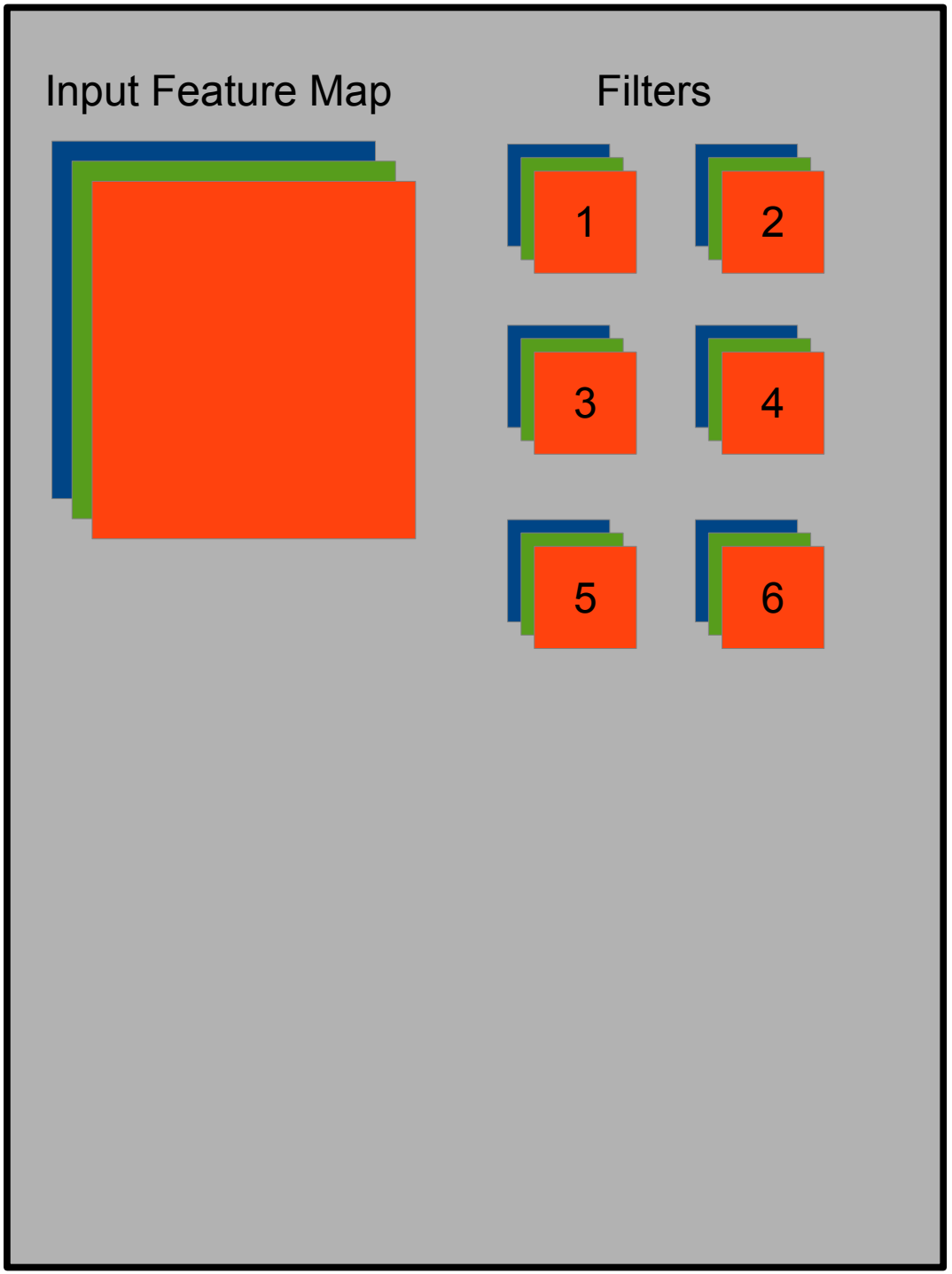
Convolution



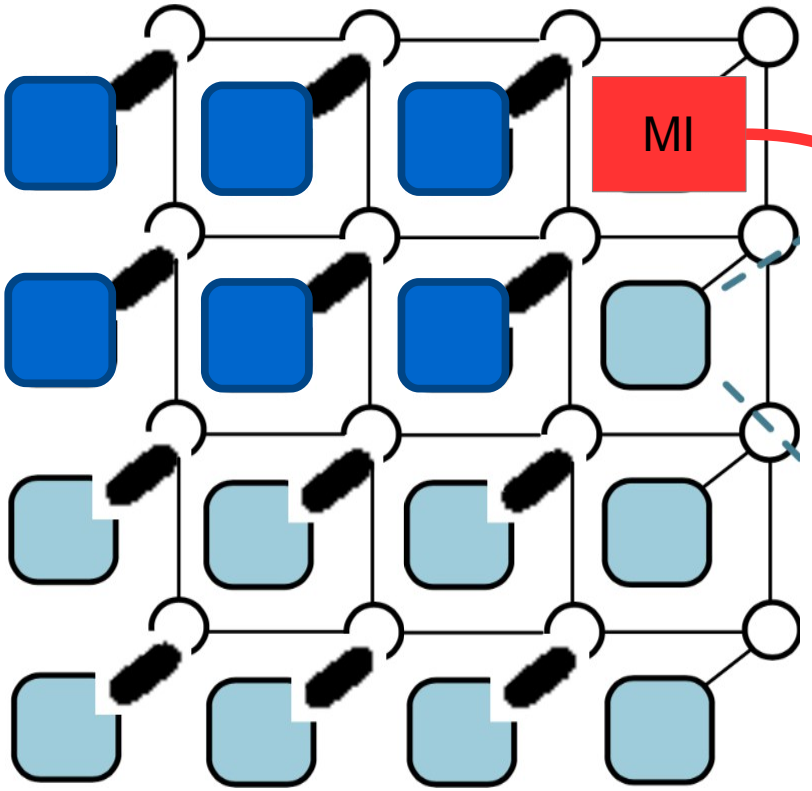


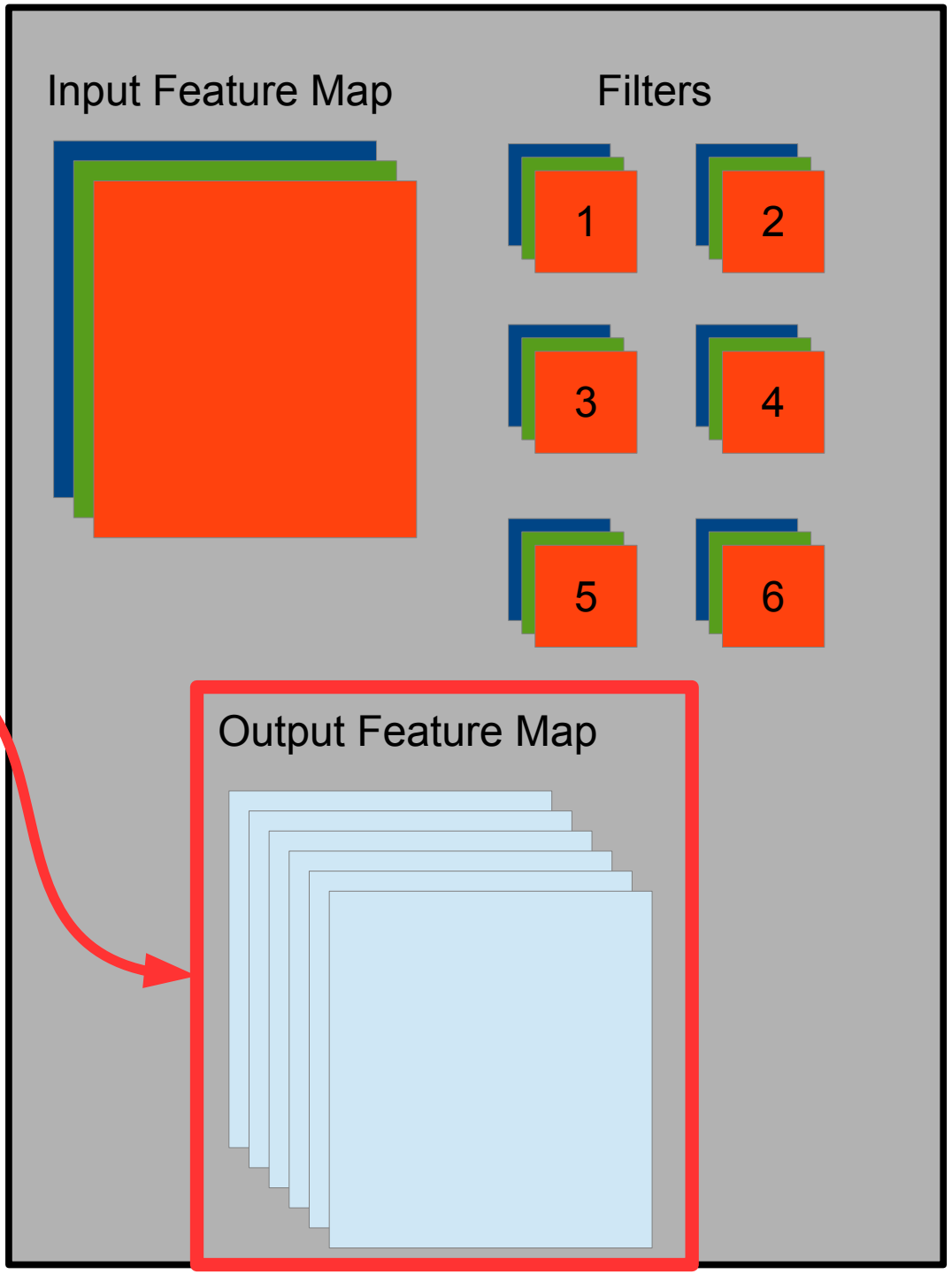
Convolution



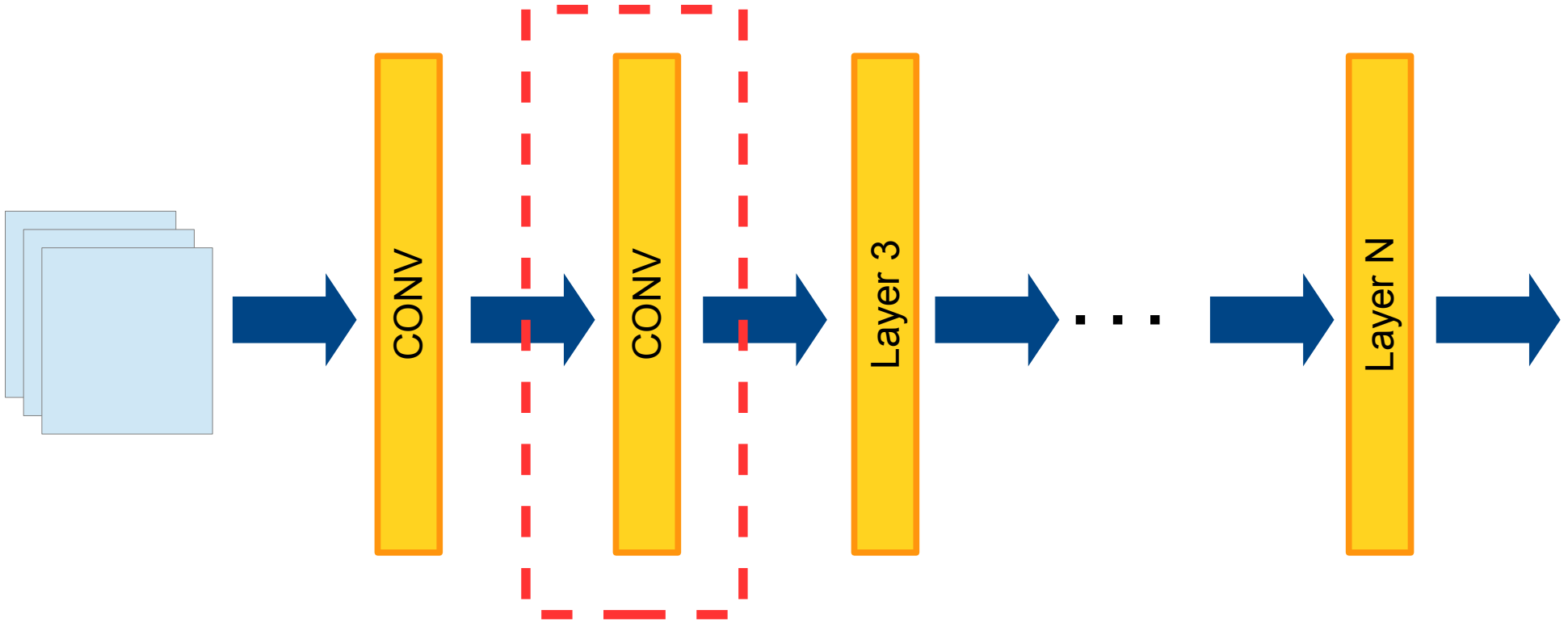


Convolution



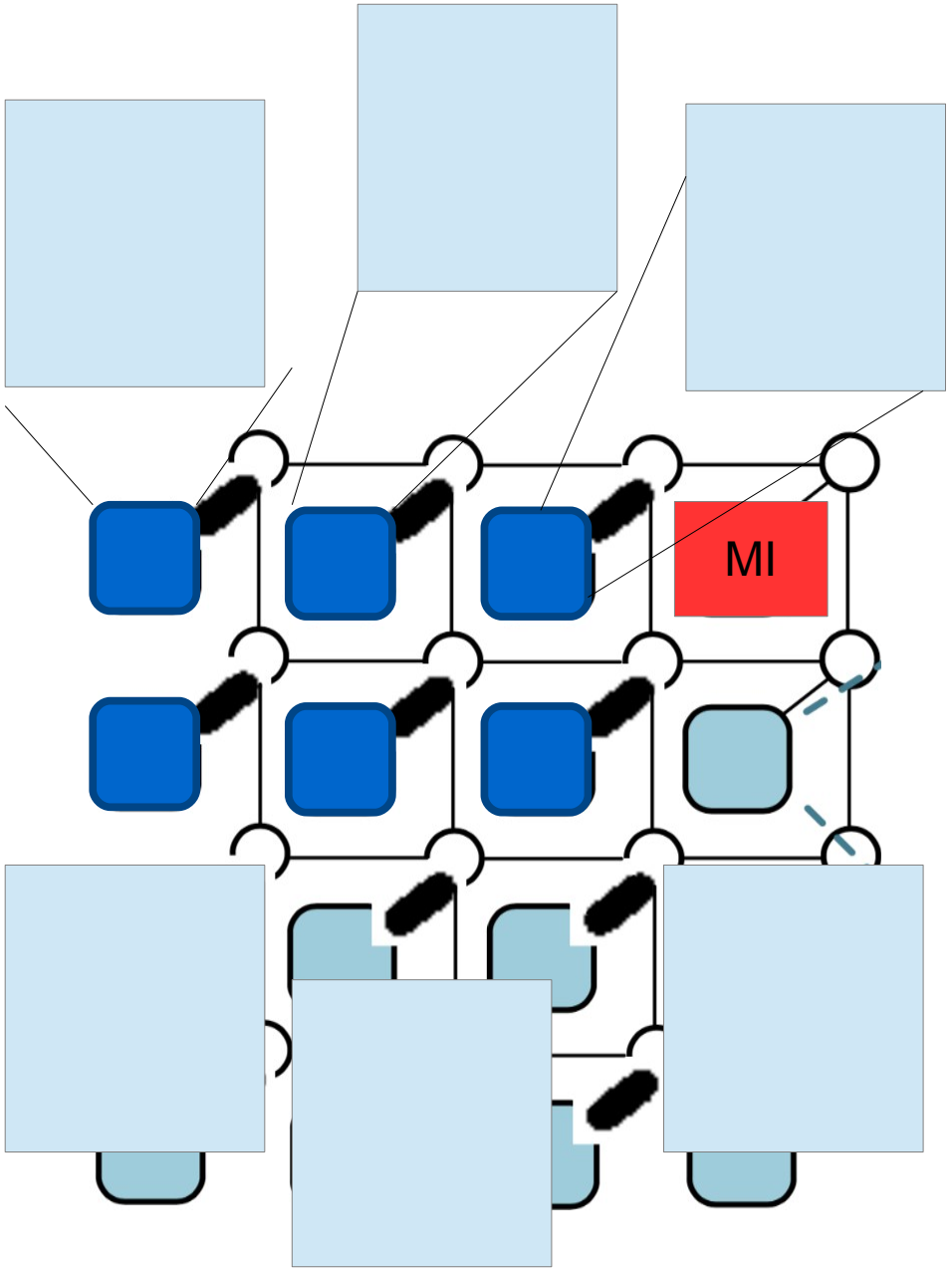


Example



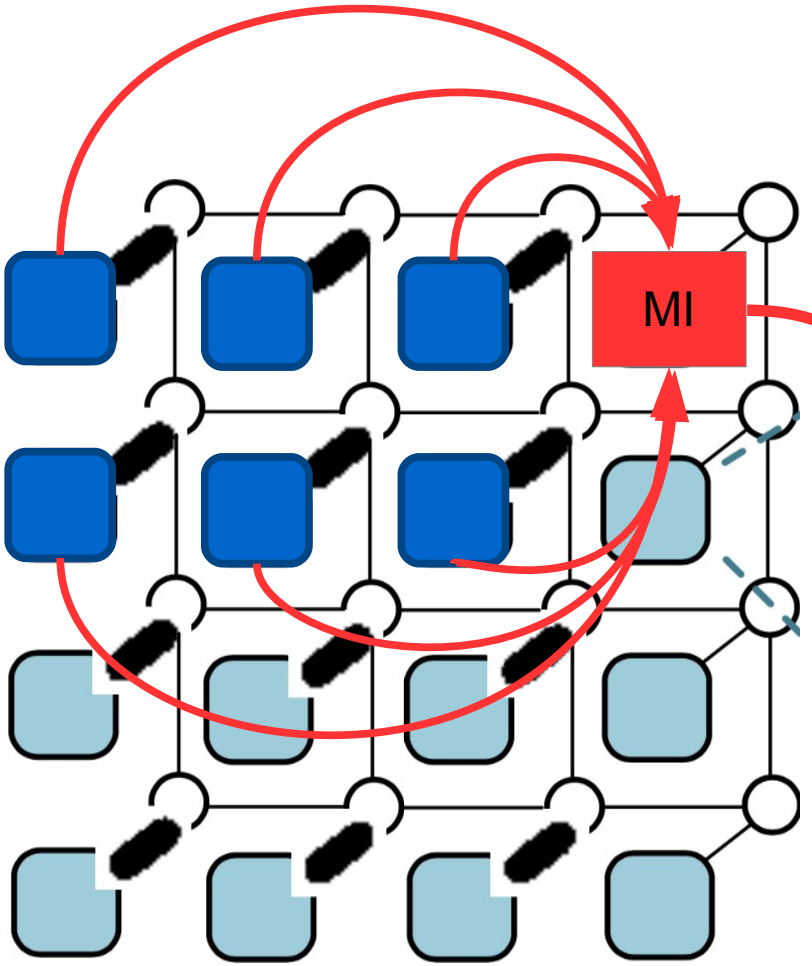
- The output feature map of layer 1 corresponds to the input feature map for layer2

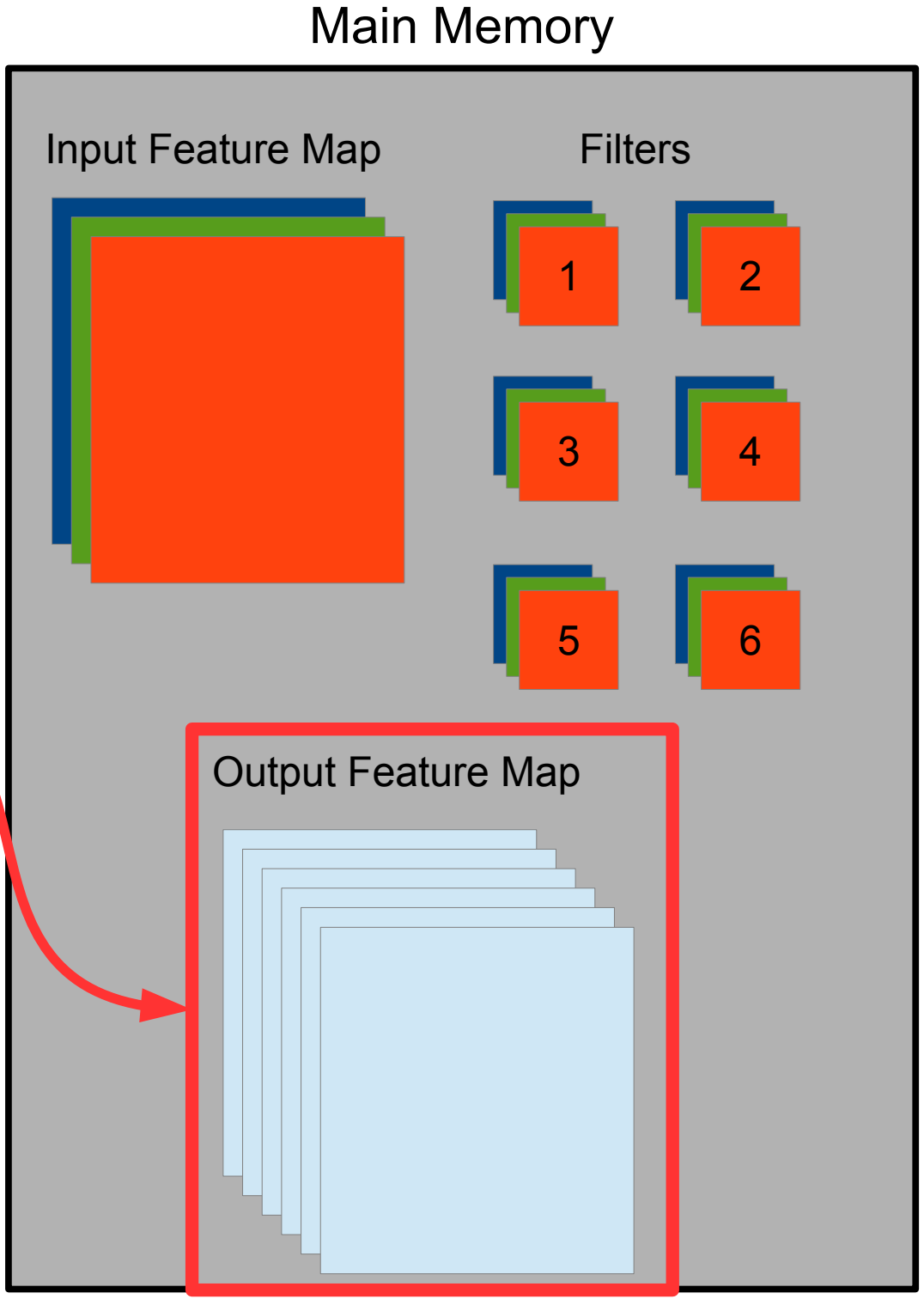
Convolution



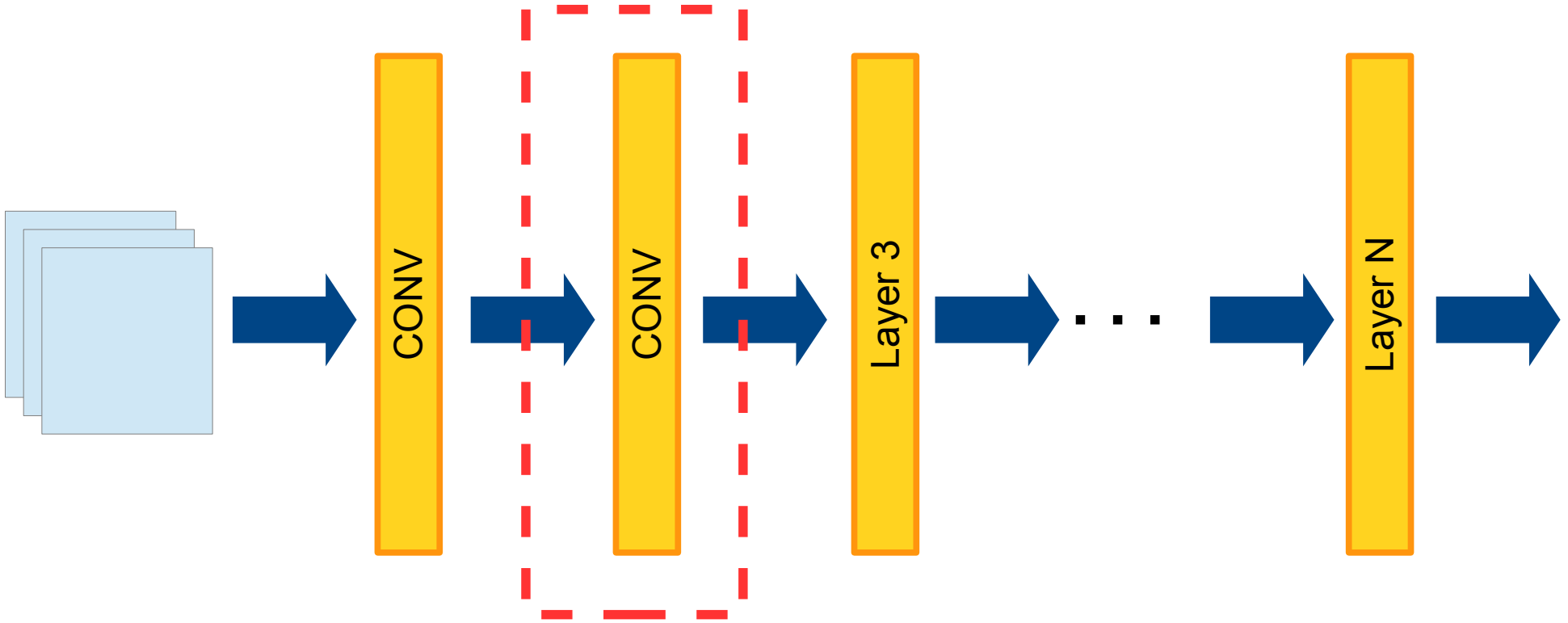
- The input feature map for the next layer is spread over the PE used in the previous layer

- No need to store back the outout feature map into main memory

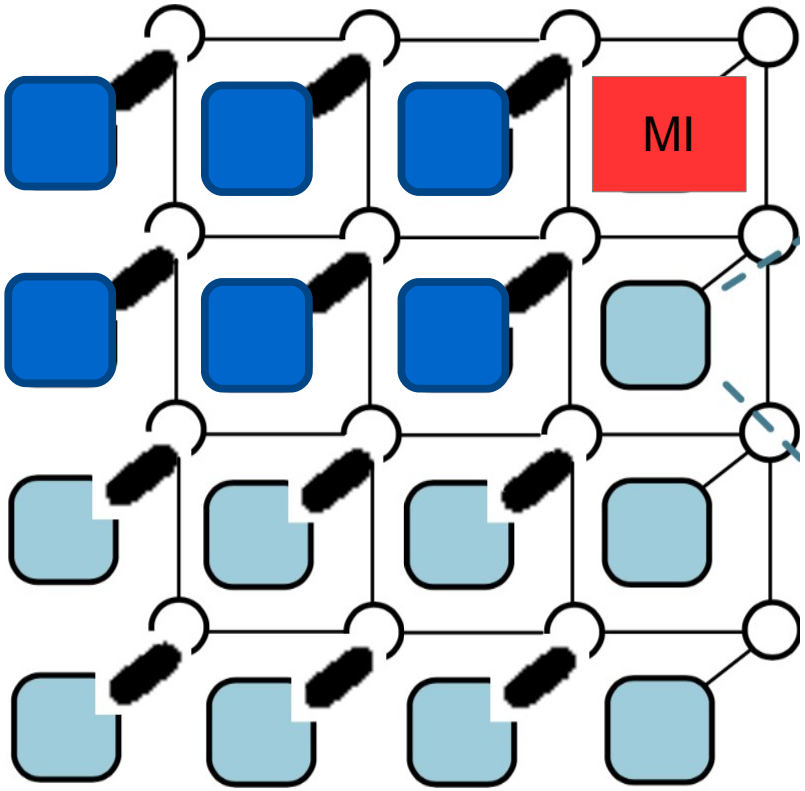


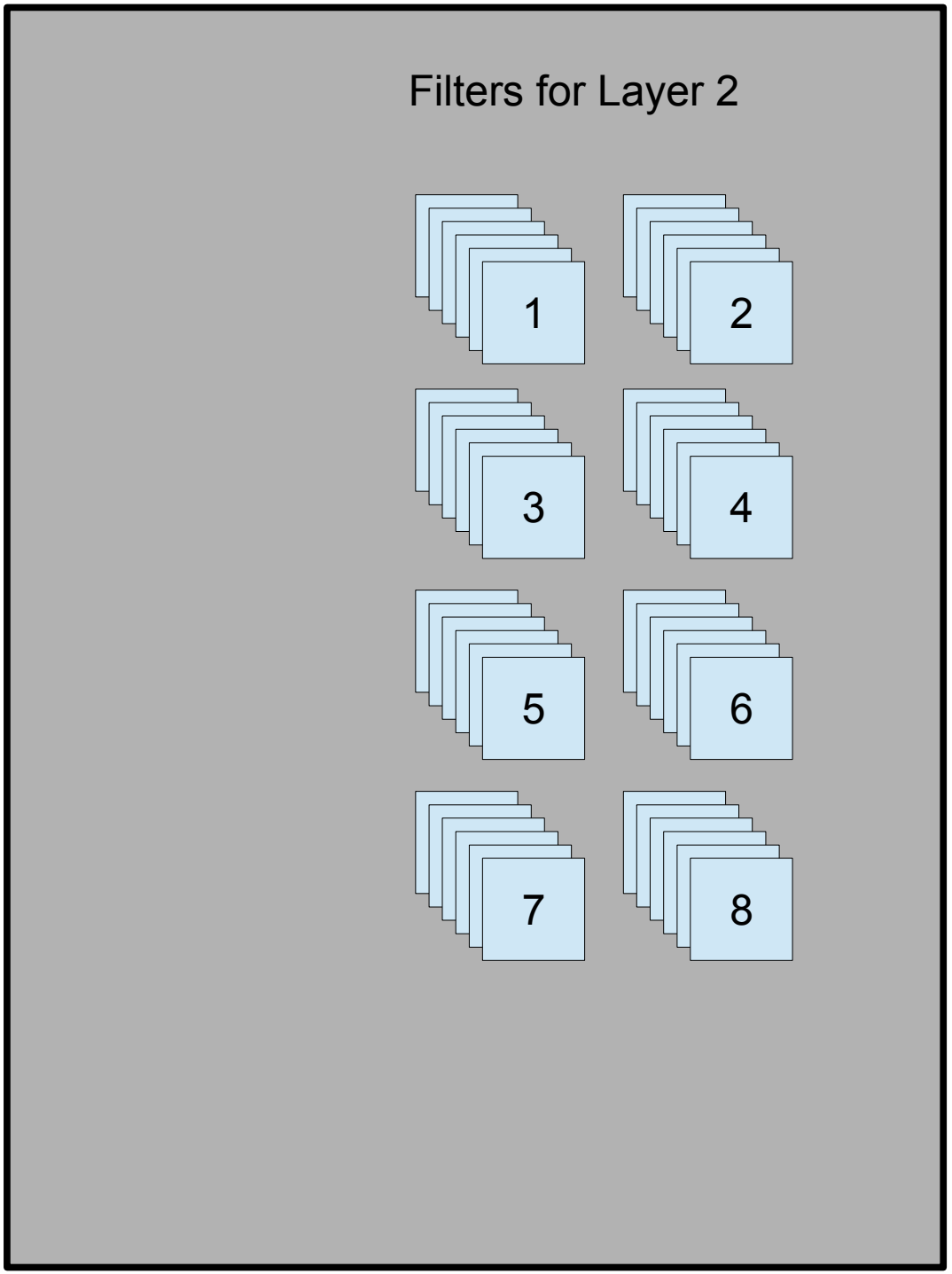


Convolution

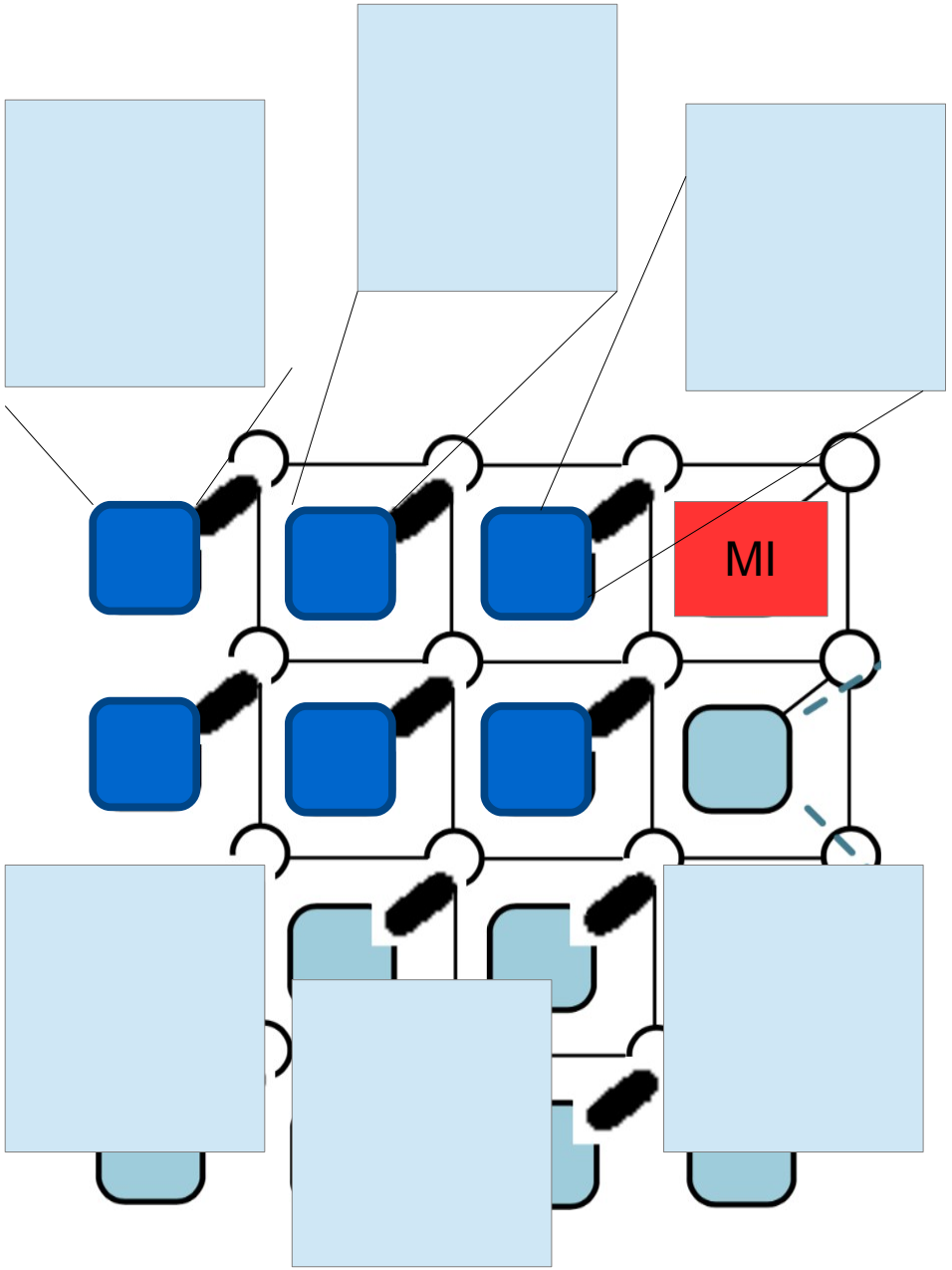


Convolution



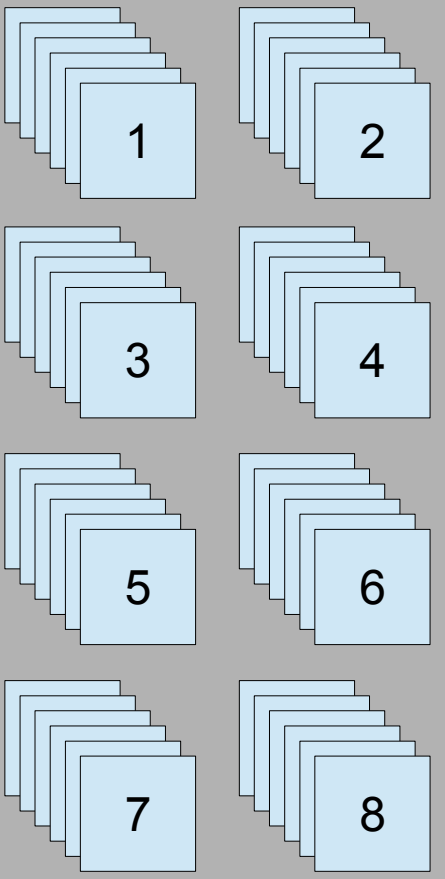


Convolution

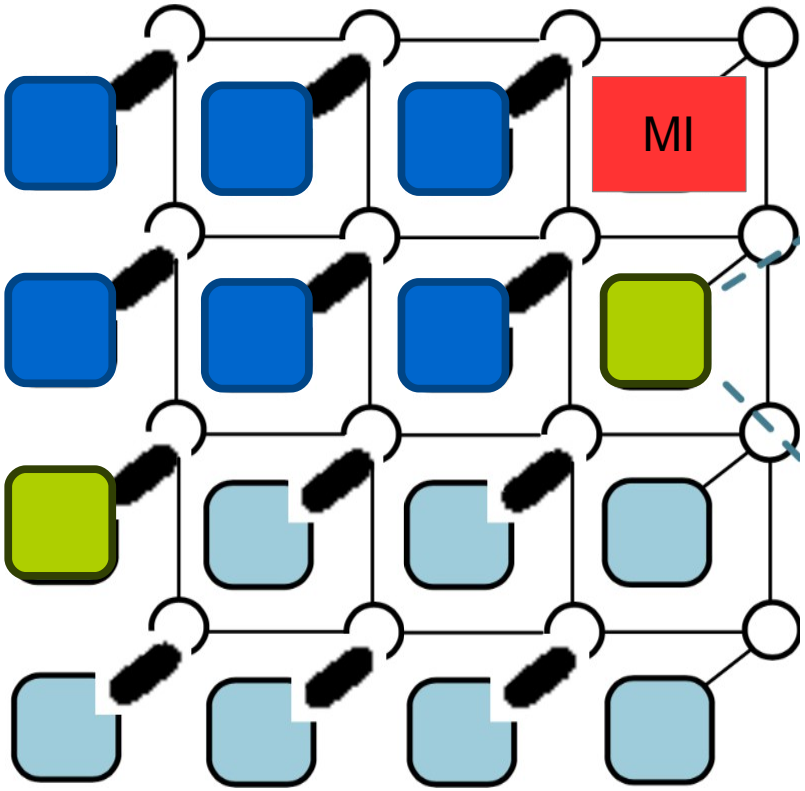


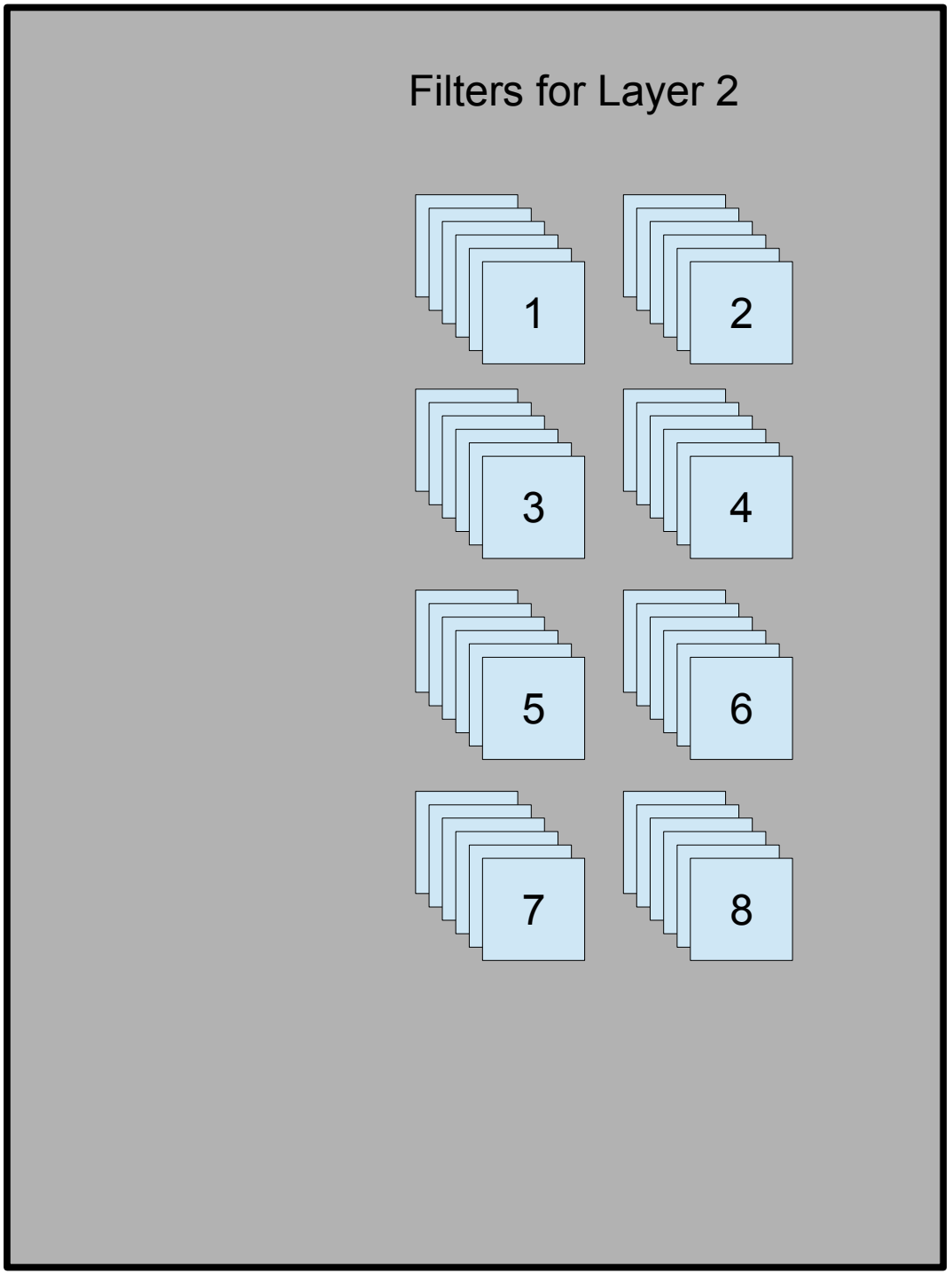
Main Memory

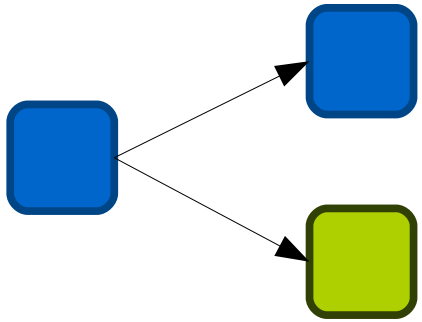
Filters for Layer 2

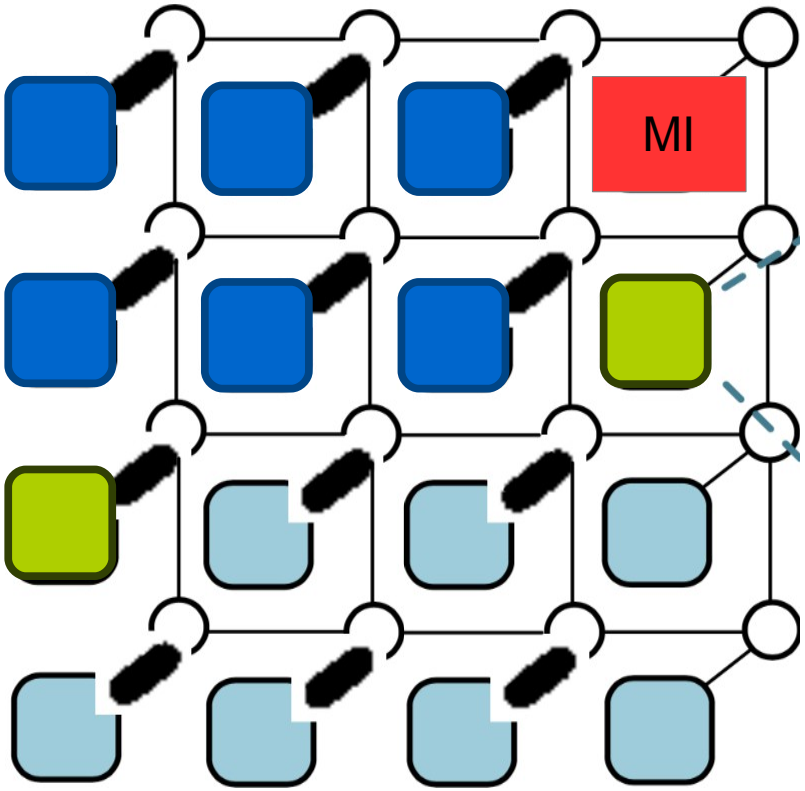


Convolution









Main Memory

Filters for Layer 2

1

2

3

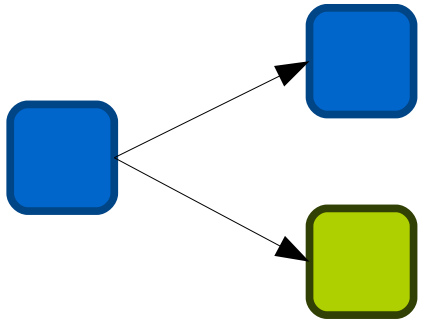
4

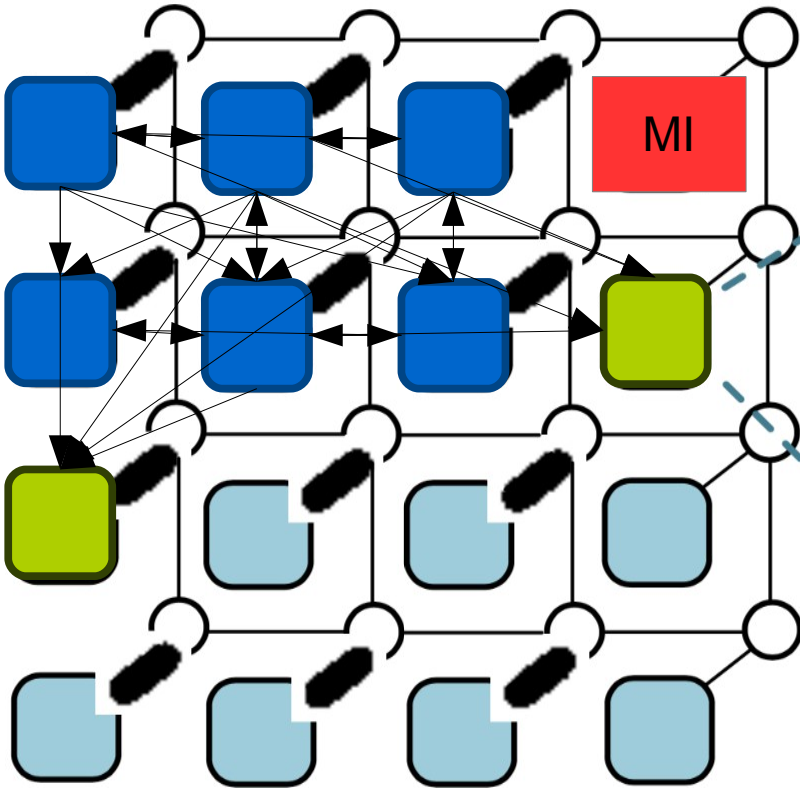
5

6

7

8





Main Memory

Filters for Layer 2

1

2

3

4

5

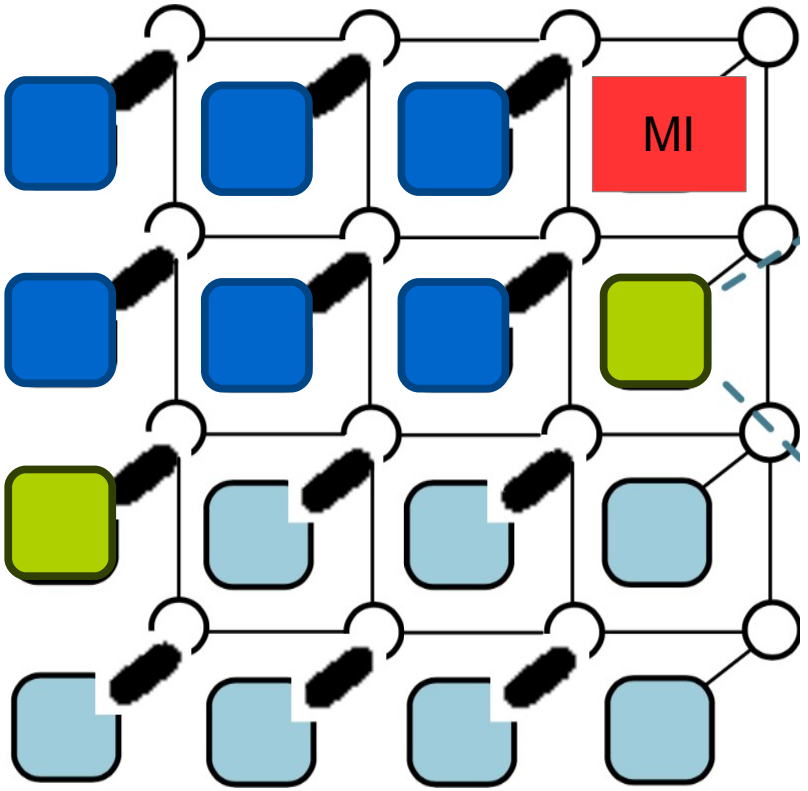
6

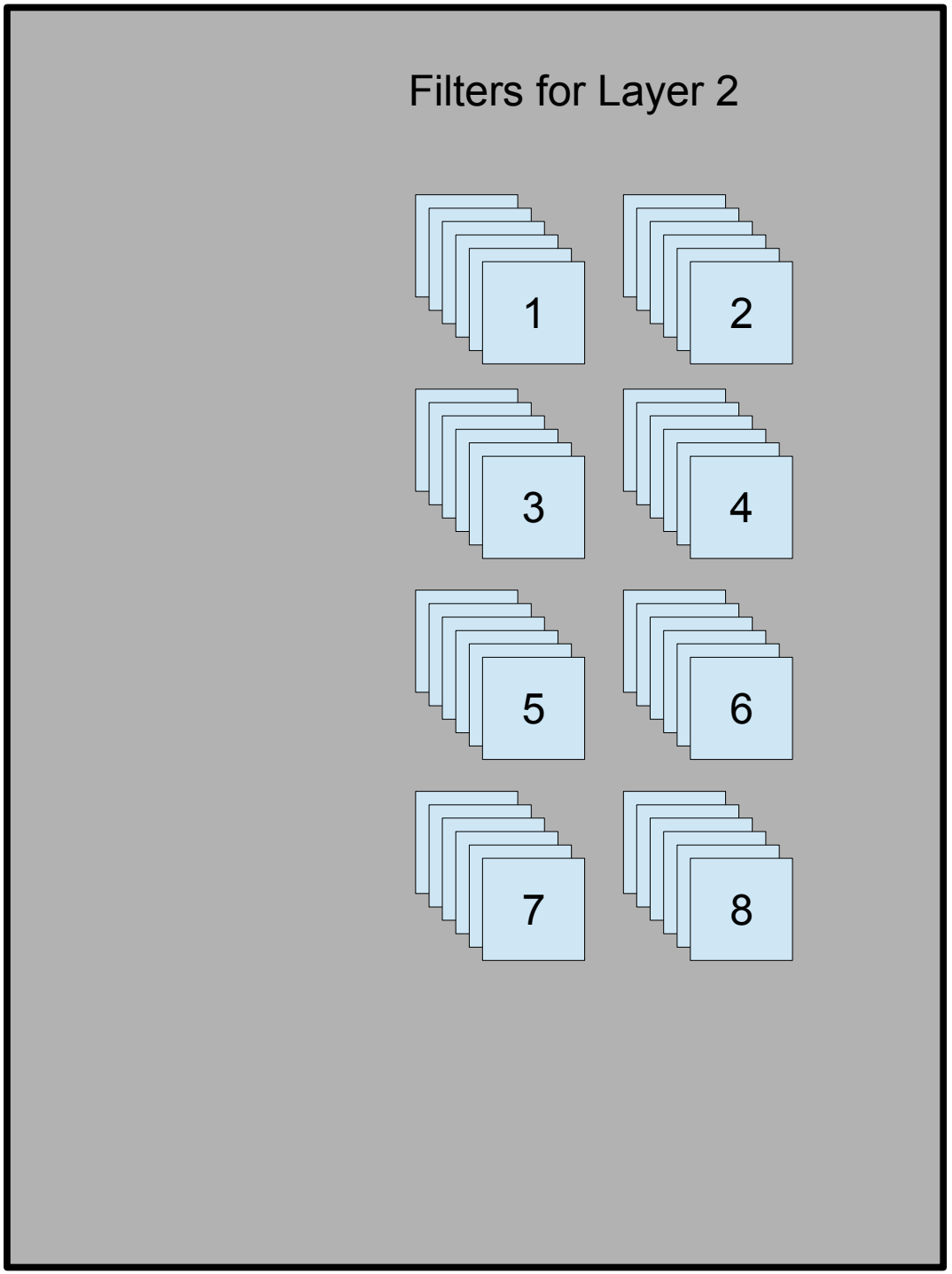
7

8

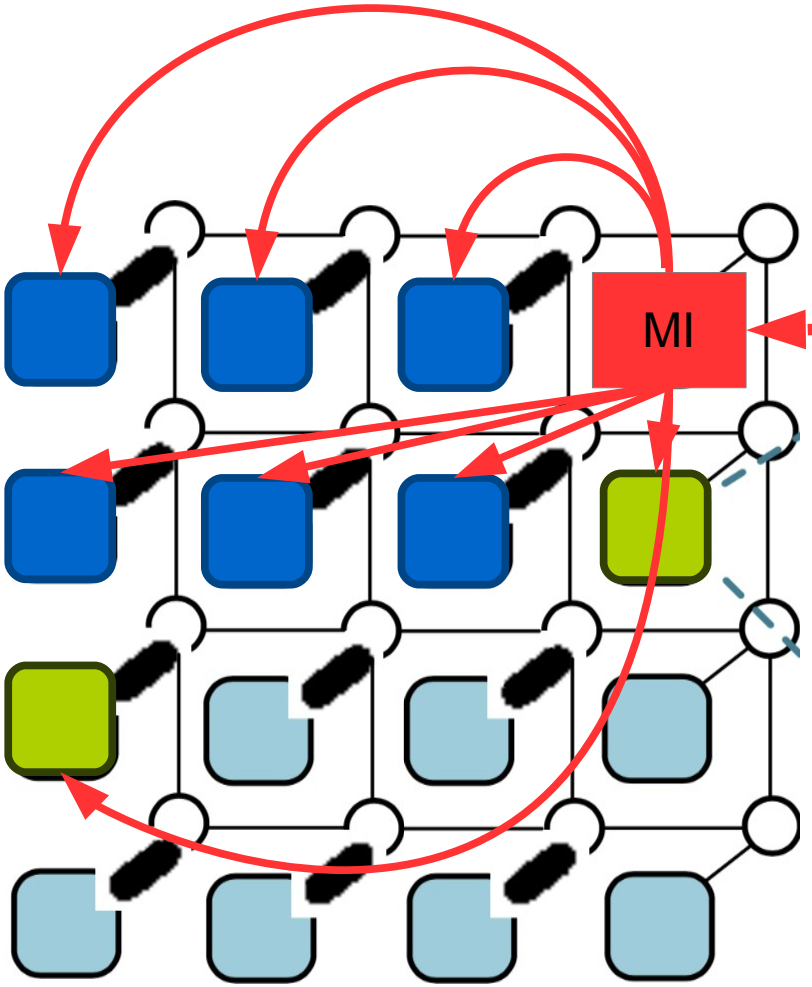
Convolution

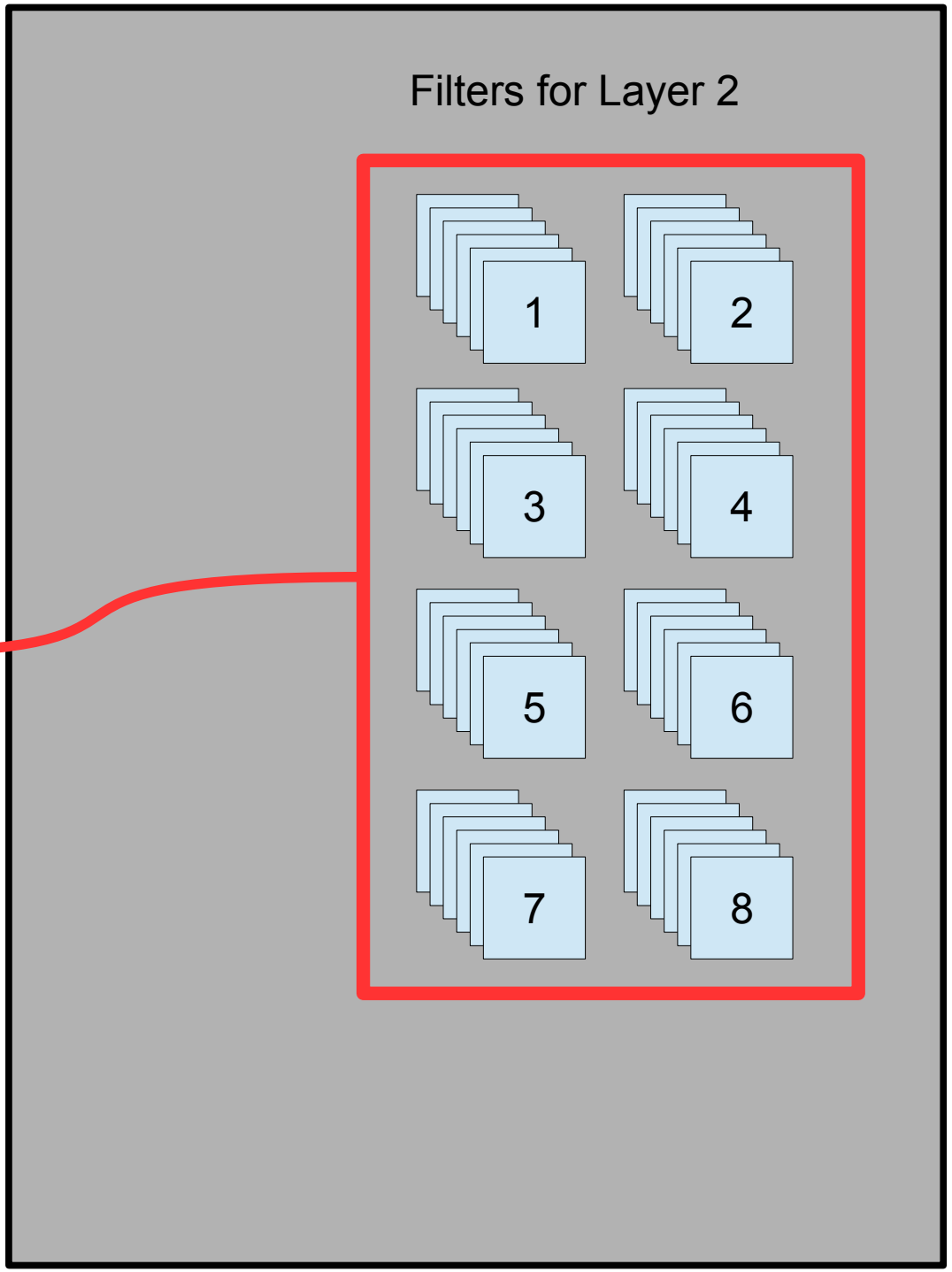
- All PE have the input feature map



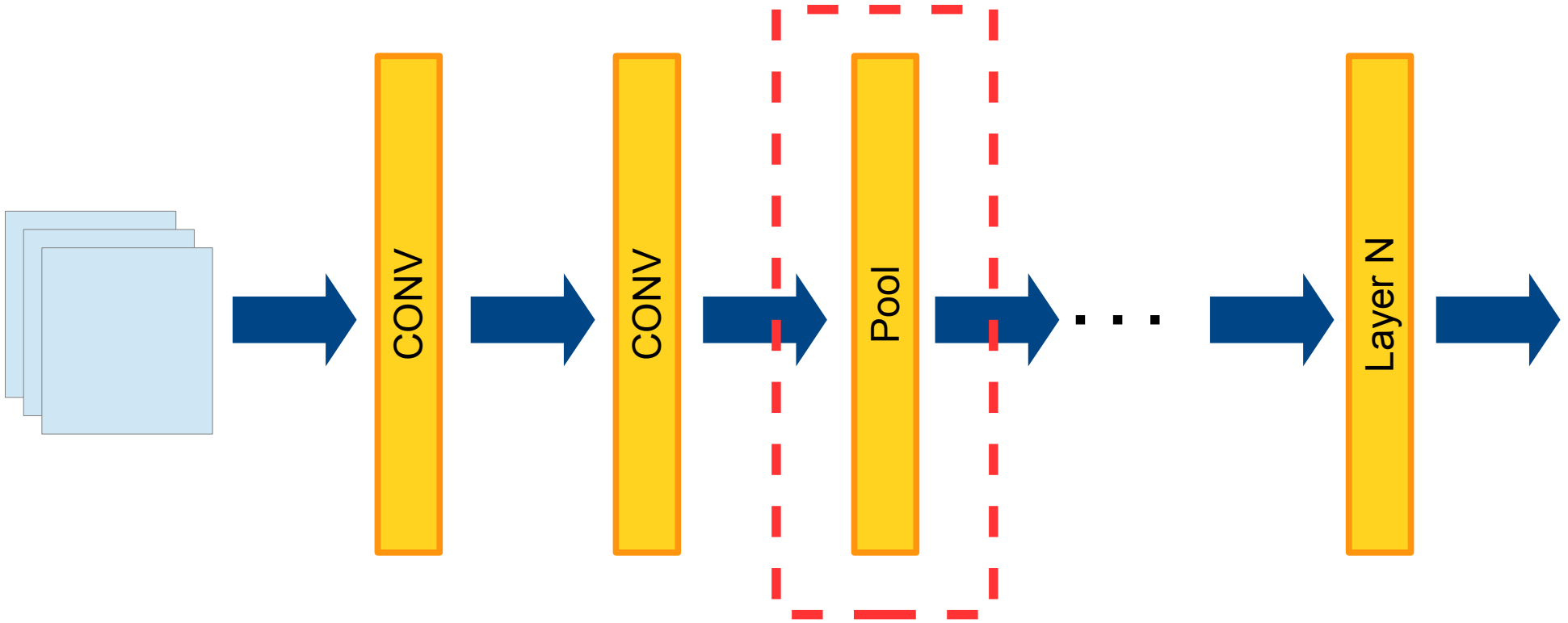


Convolution



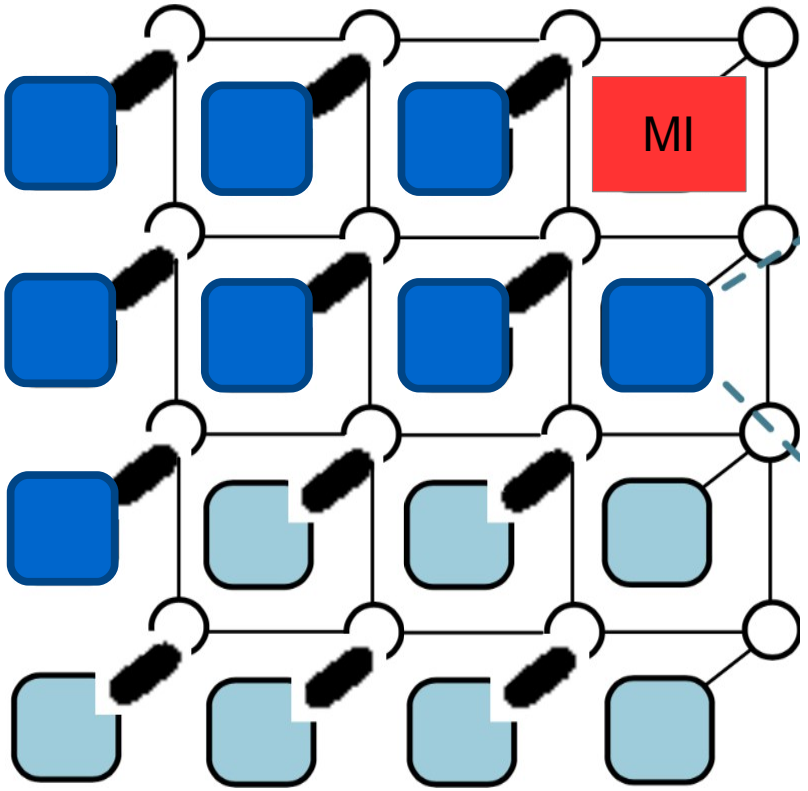


Max/Avg Pool



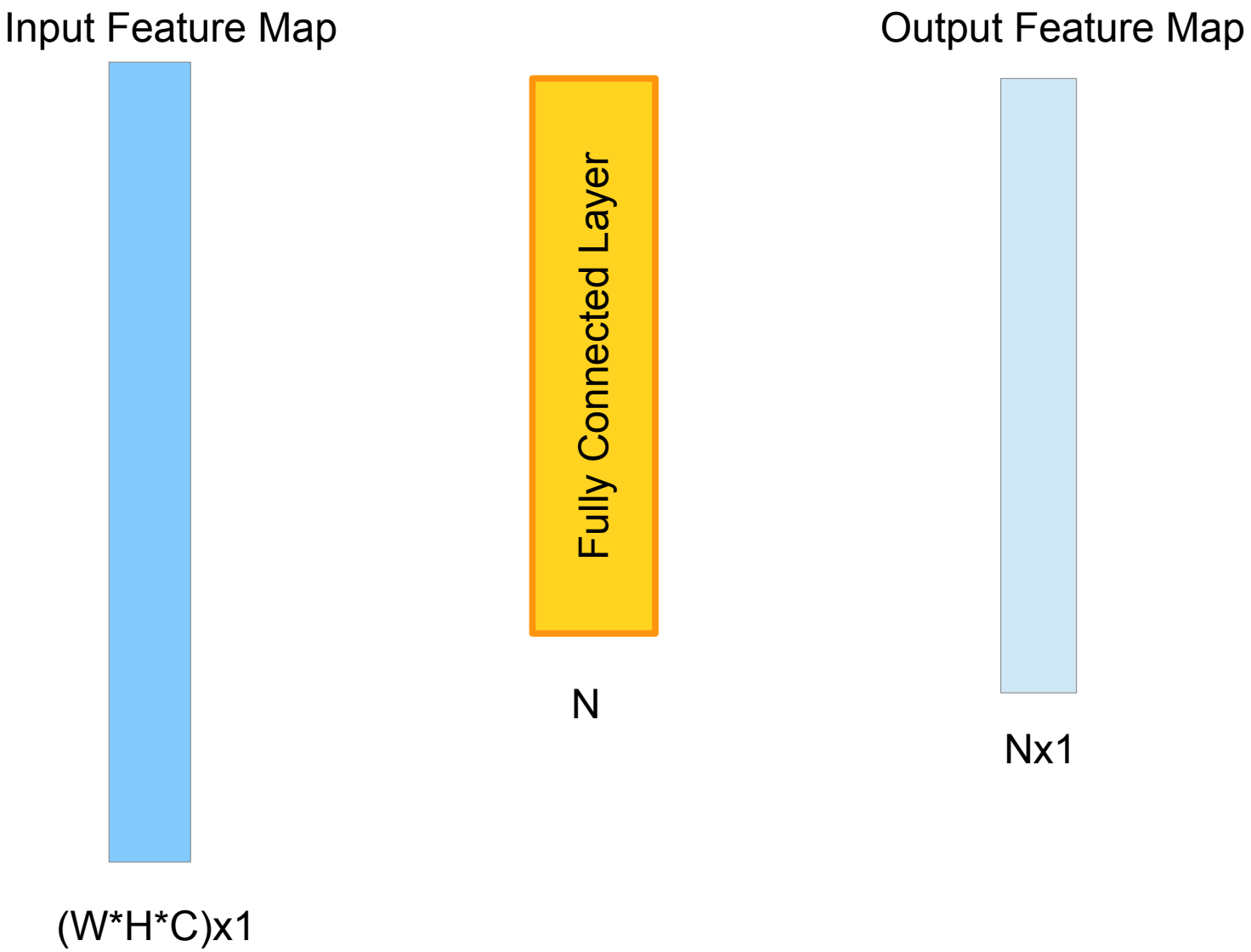
Avg/Max Pool

- Input feature map spread over the PE used in the previous layer
- Each PE computes its feature map channel

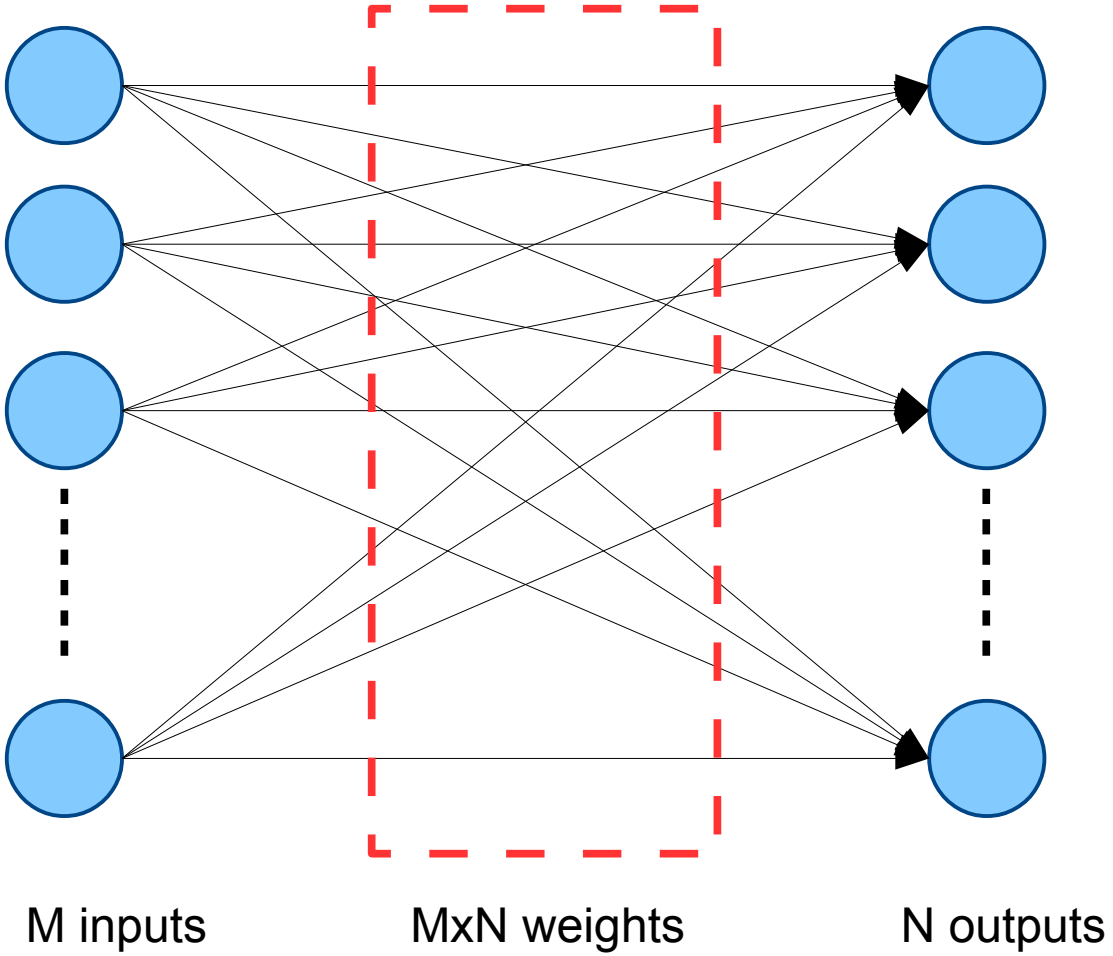


No memory traffic

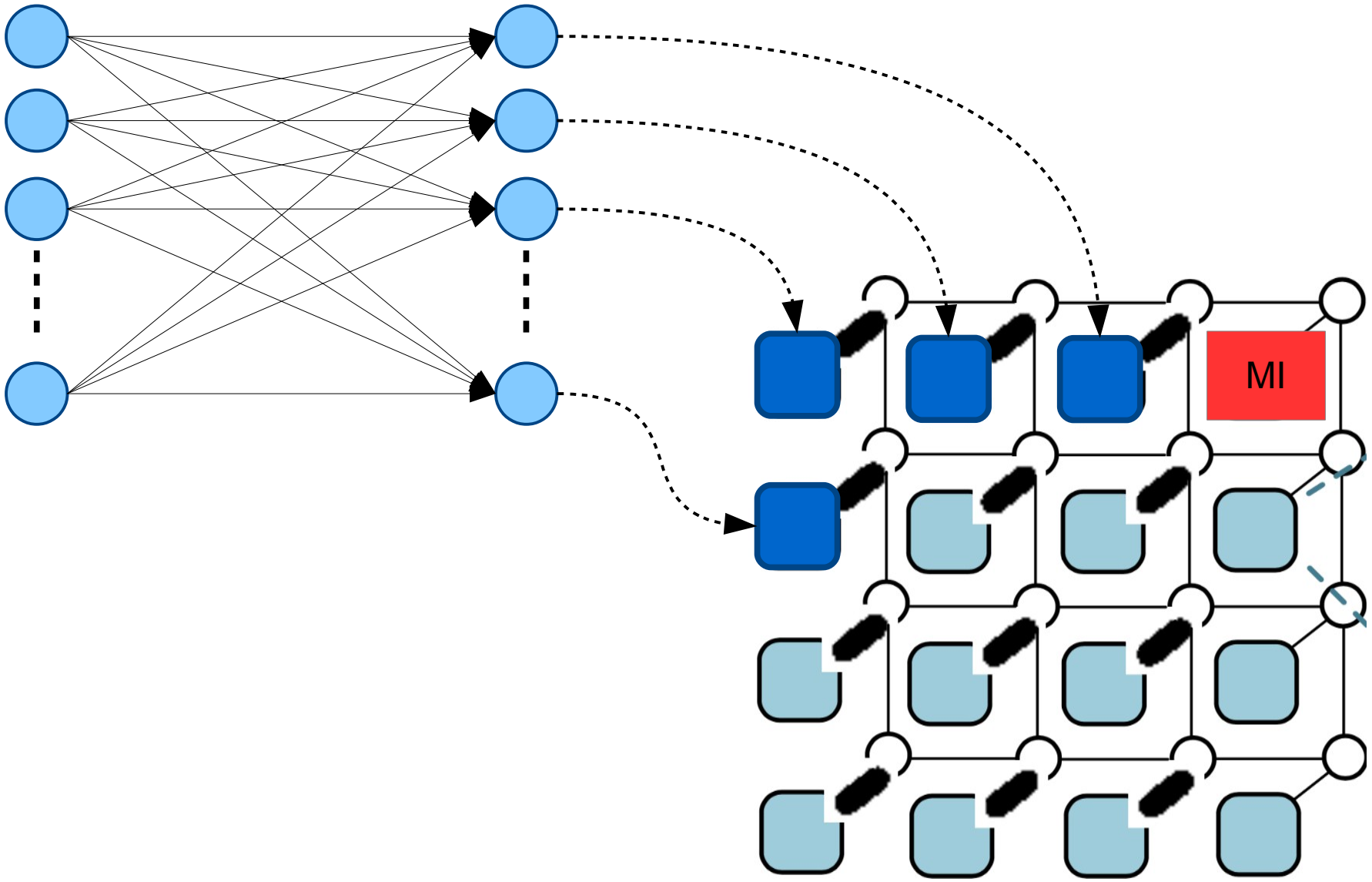
Fully Connection



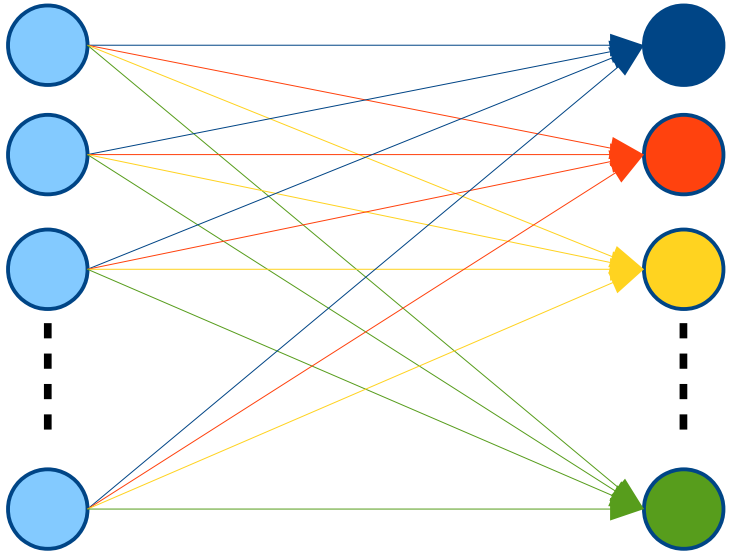
Fully Connection

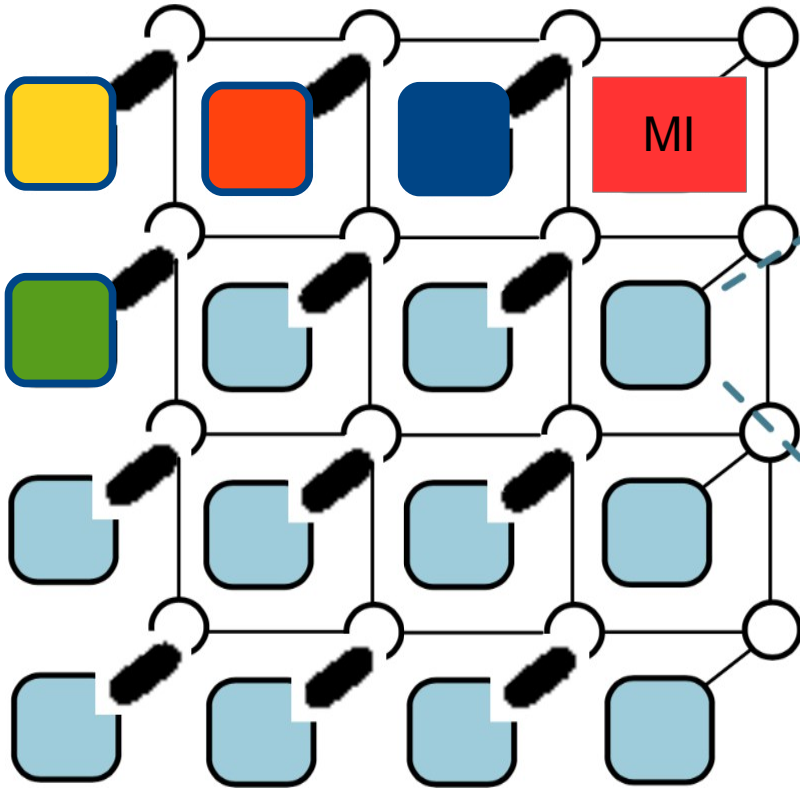


Fully Connection

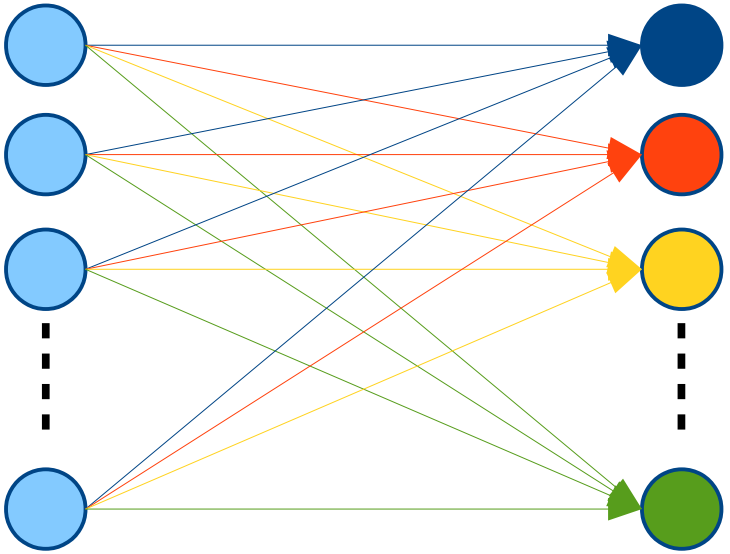


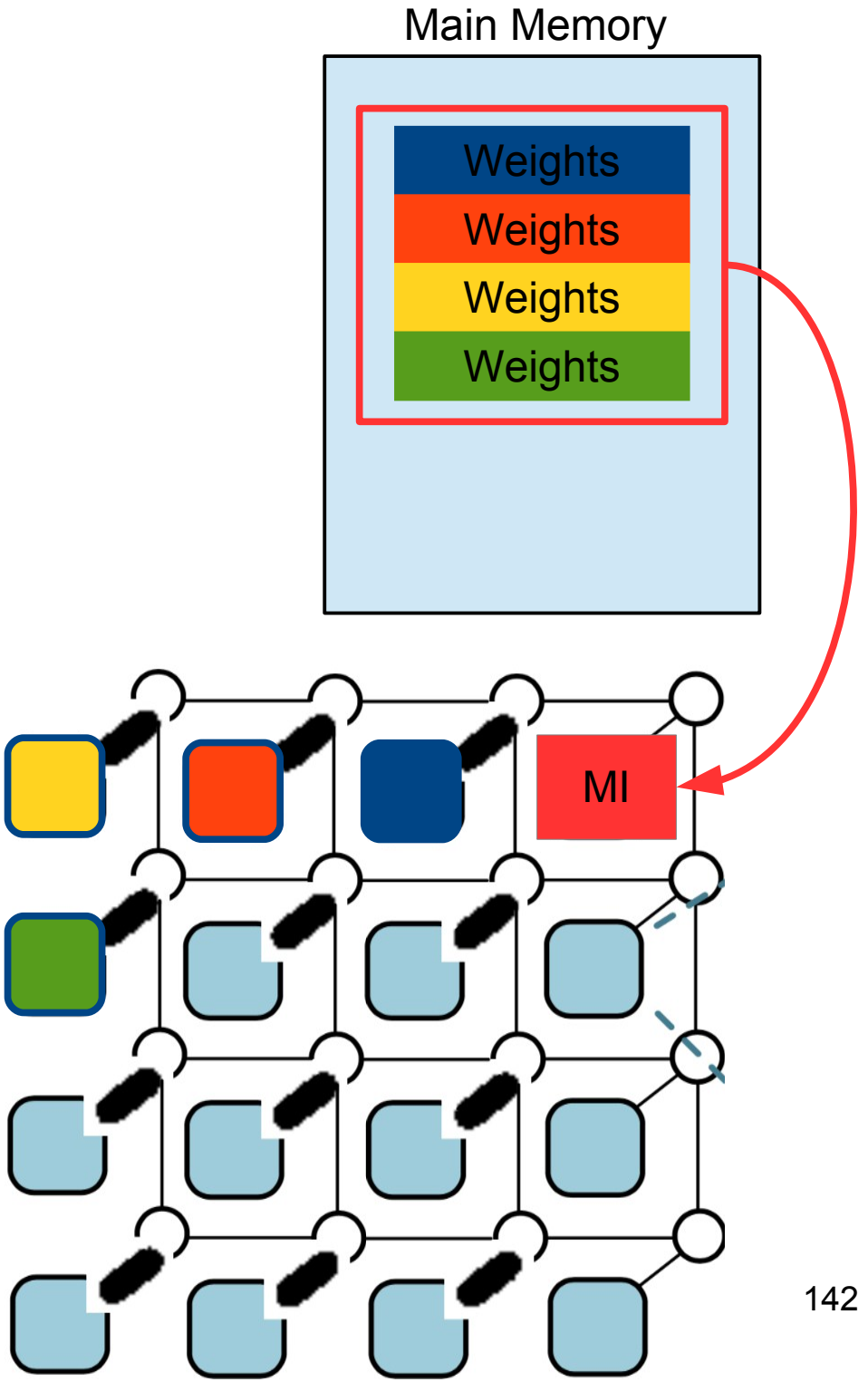
Fully Connection



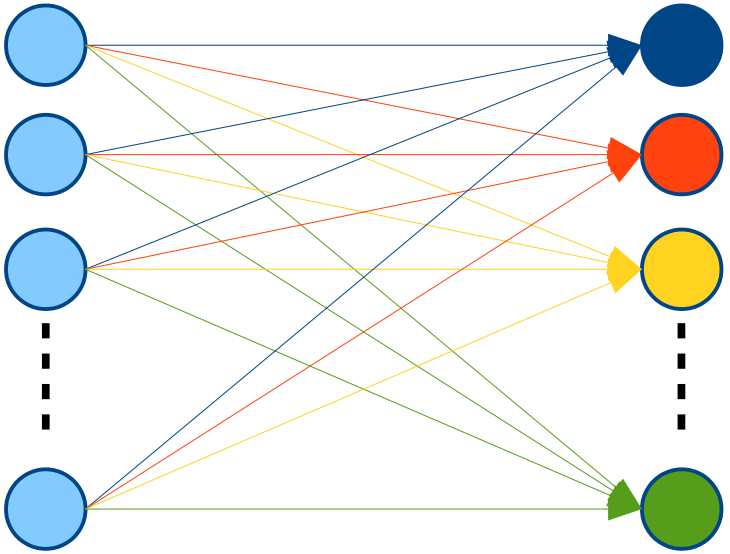


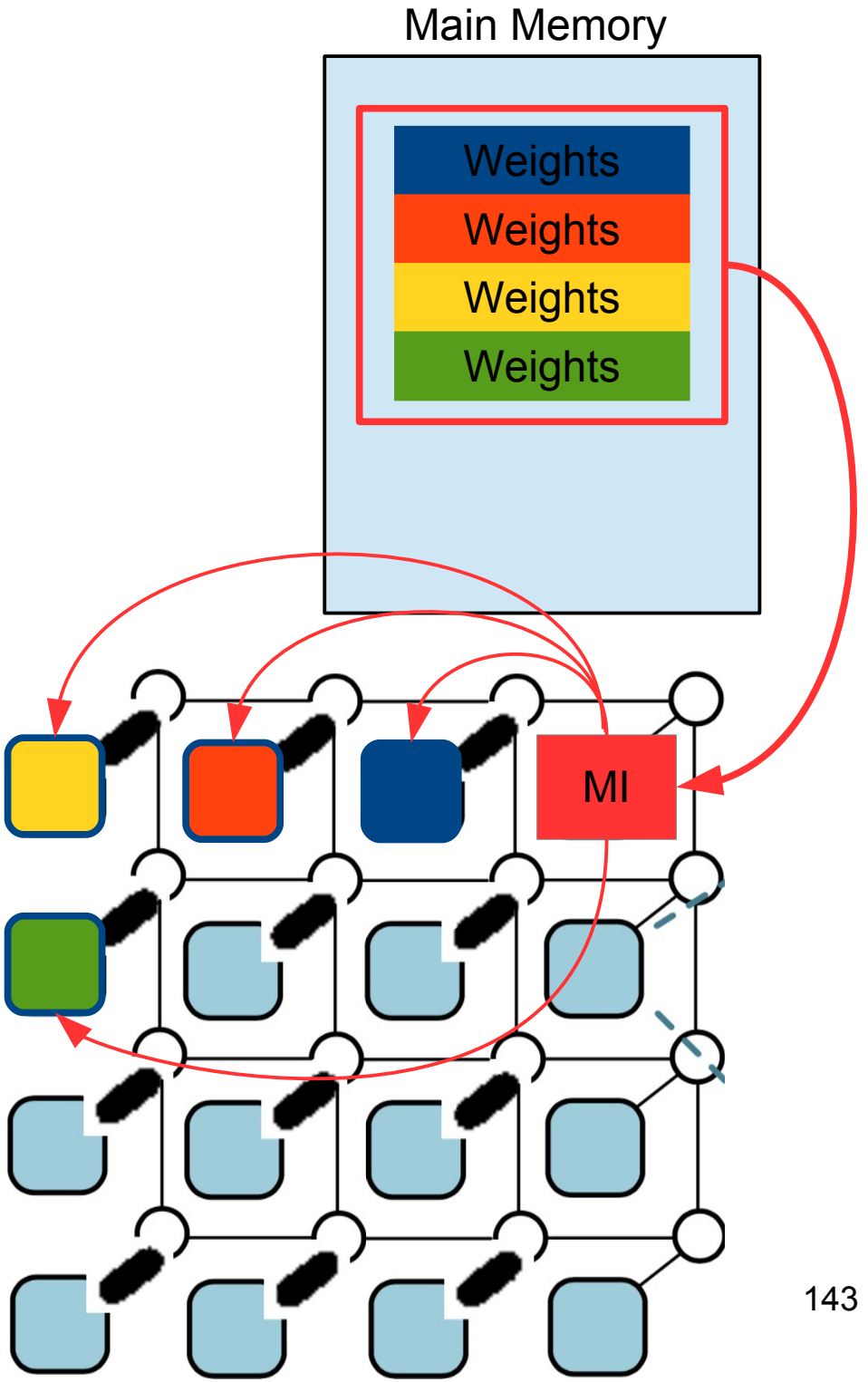
Fully Connection





Fully Connection





Memory/Comm Traffic Summary

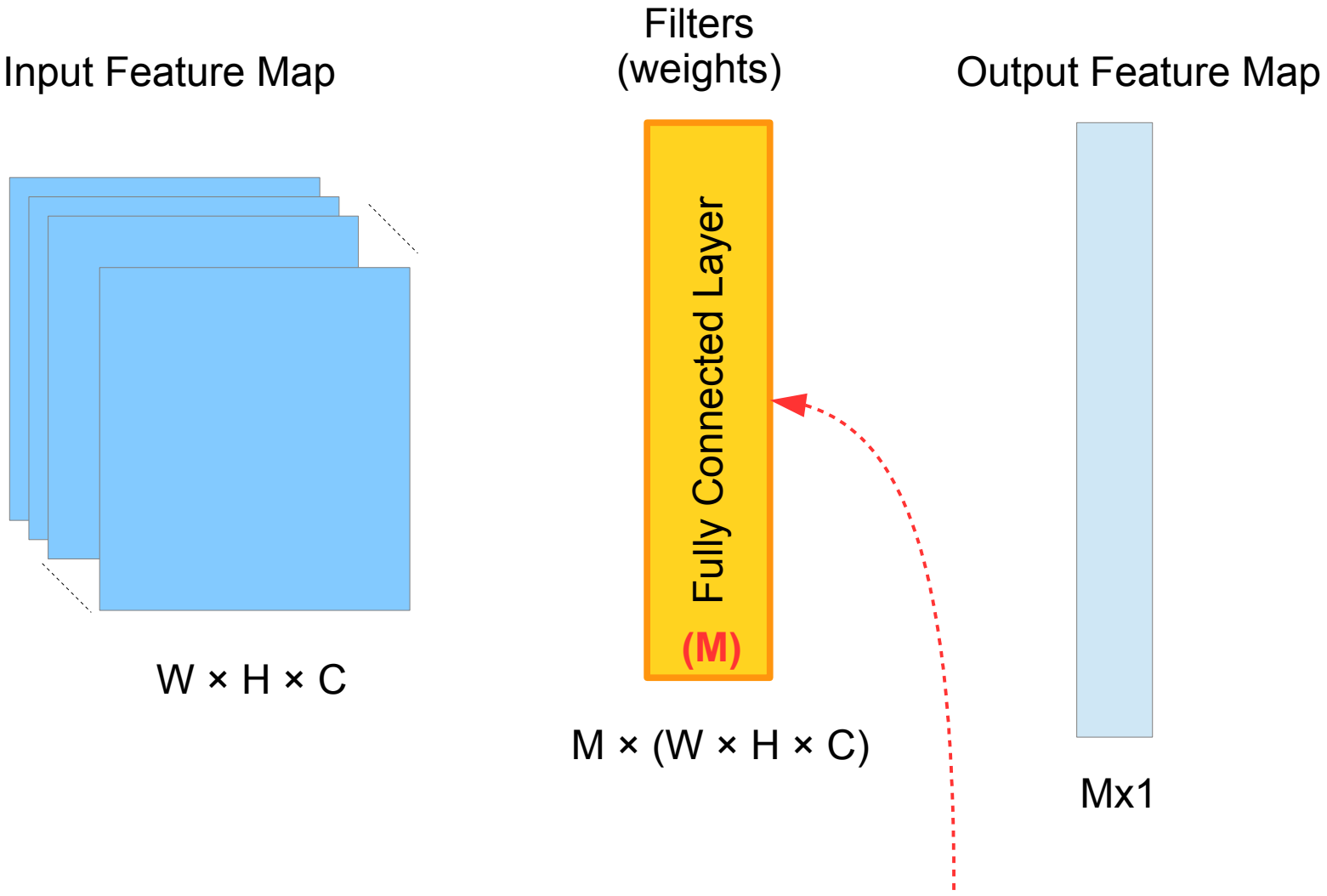
- High traffic volume to/from memory
- MI is an hot-spot point
- Different traffic types
  - Unicast
  - Multicast
  - Broadcast
- Communication network plays an important role

[S. M. Nabavinejad, *et al.* “An Overview of Efficient Interconnection Networks for Deep Neural Network Accelerators”. IEEE J. Emerg. Sel. Topics Circuits Syst. 2020]

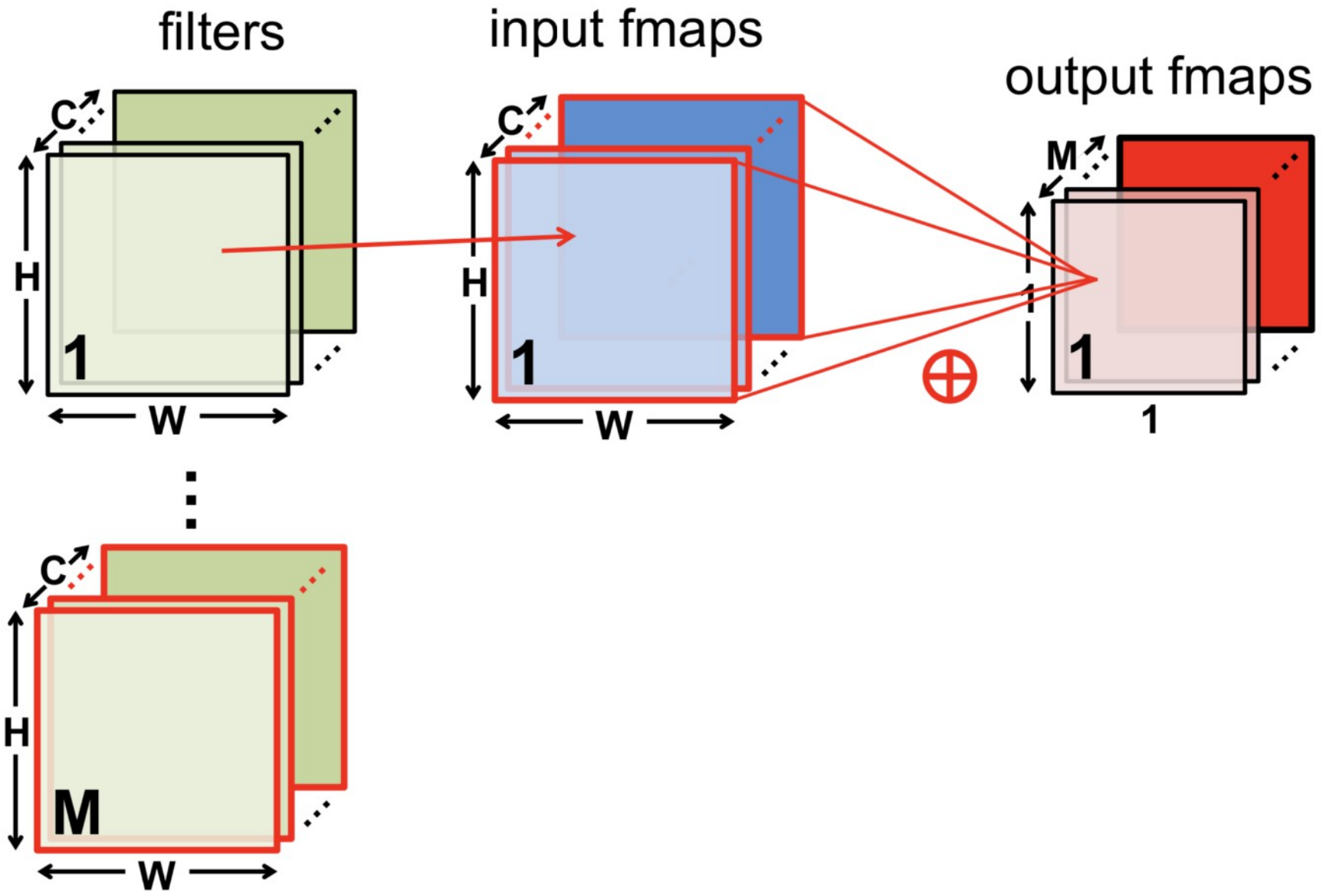
Agenda

- Deep Neural Networks
- Types of layers
- Memory and communication traffic
- Kernel computation
- Quantitative analysis

Fully Connected Layer

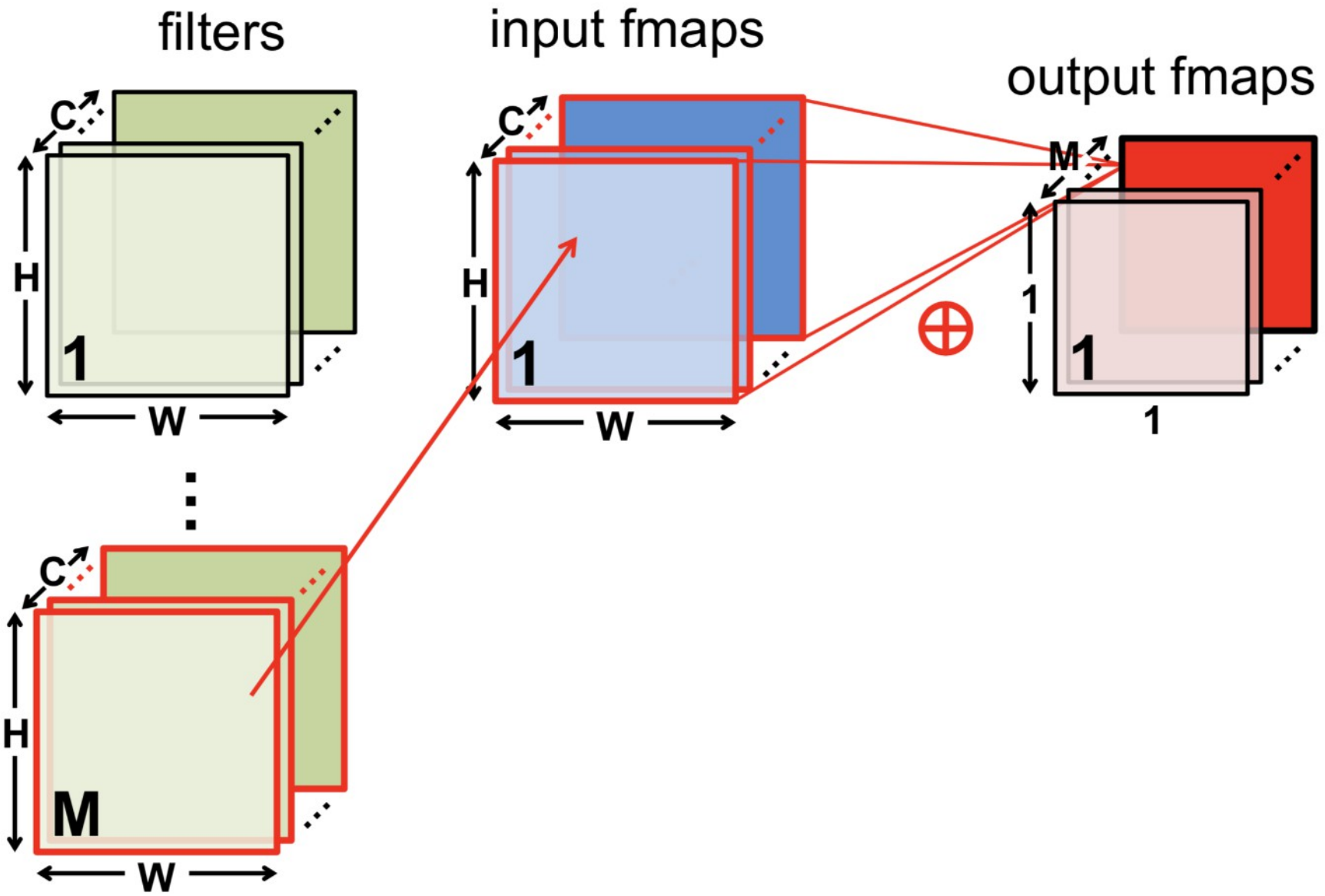


Fully Connected Layer



[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

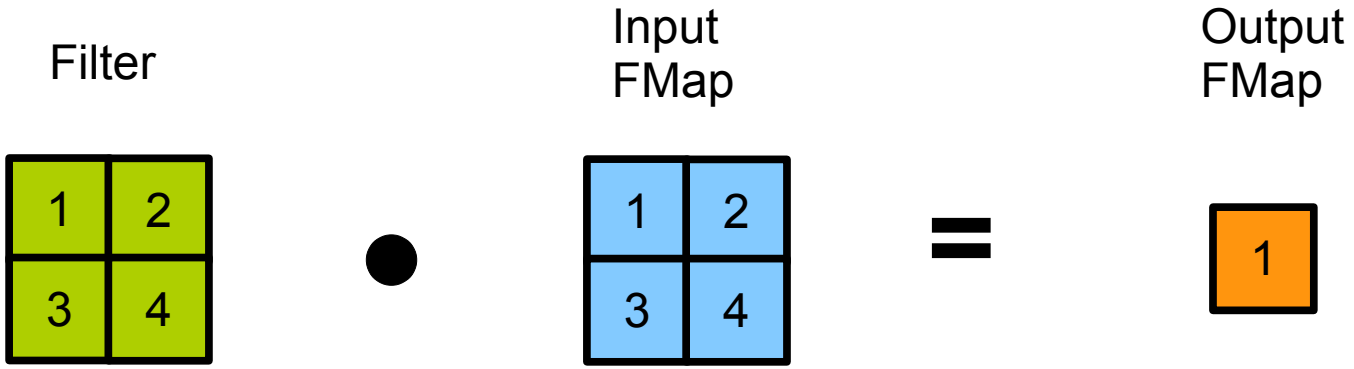
Fully Connected Layer



[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

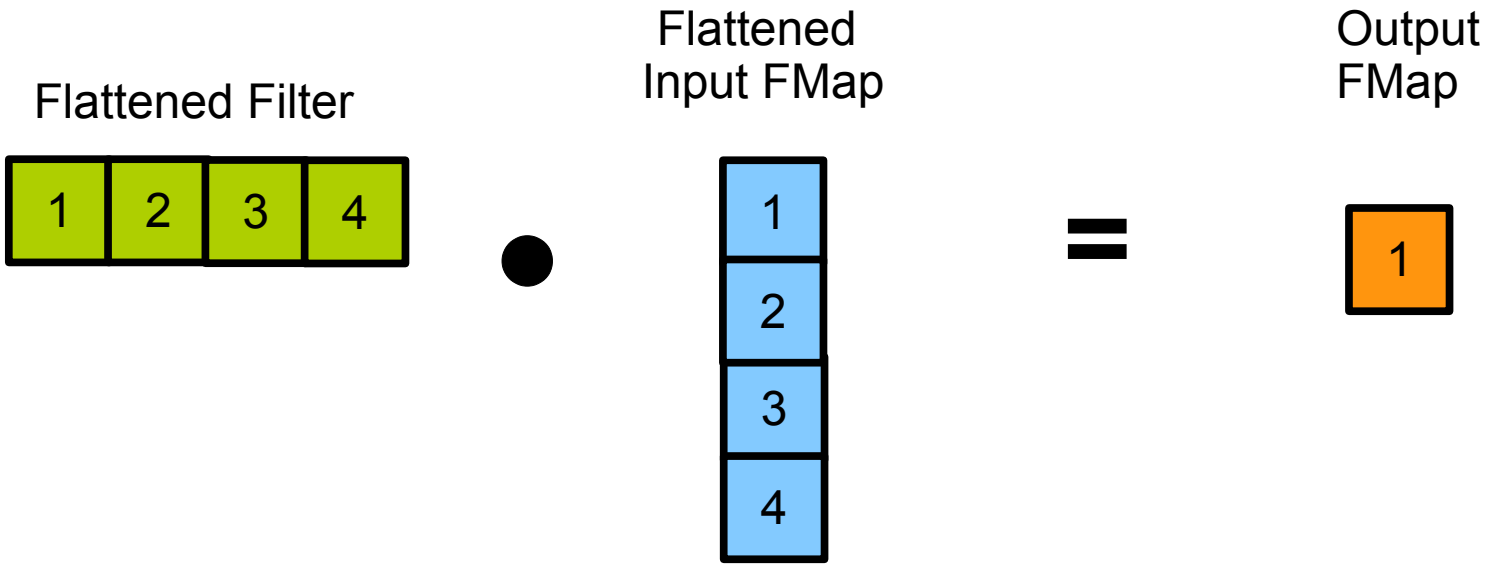
Kernel Operation

- Computing the Output Fmap point-by-point
  - 2D Dot Product



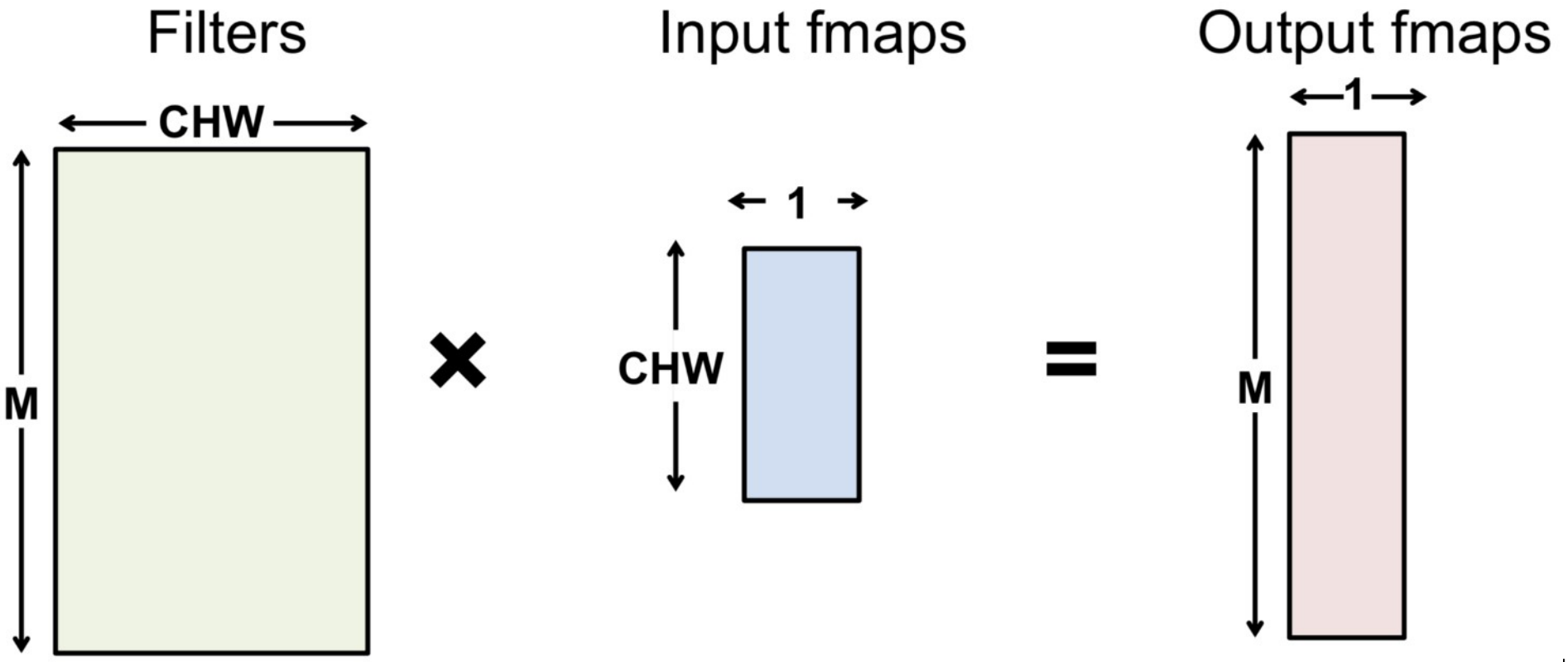
Kernel Operation

- 2D Dot Product can be converted to 1D Dot Product



Fully Connected Layer

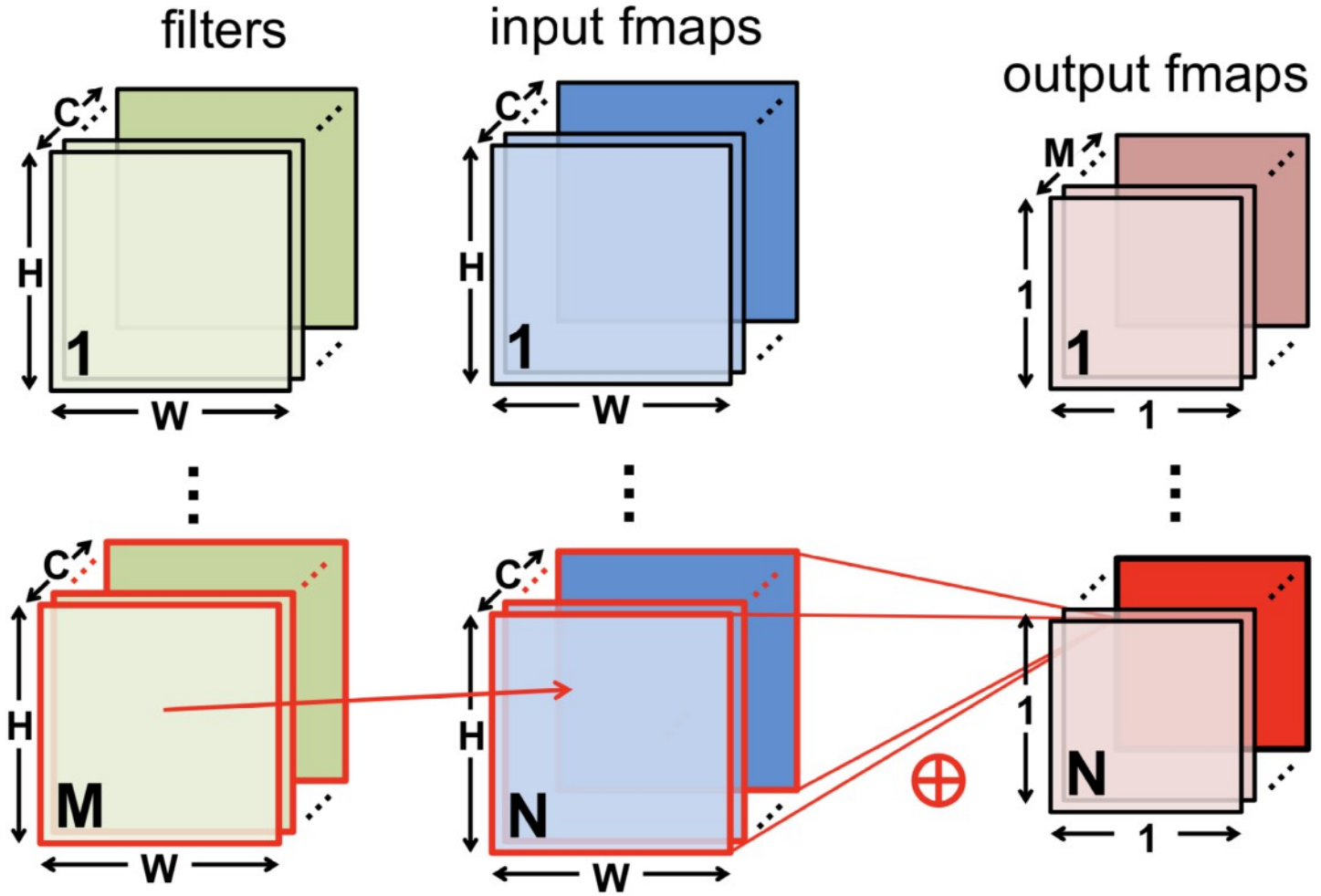
- Compute the Output fmap all at once
  - Matrix-Vector Multiply



[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Fully Connected Layer

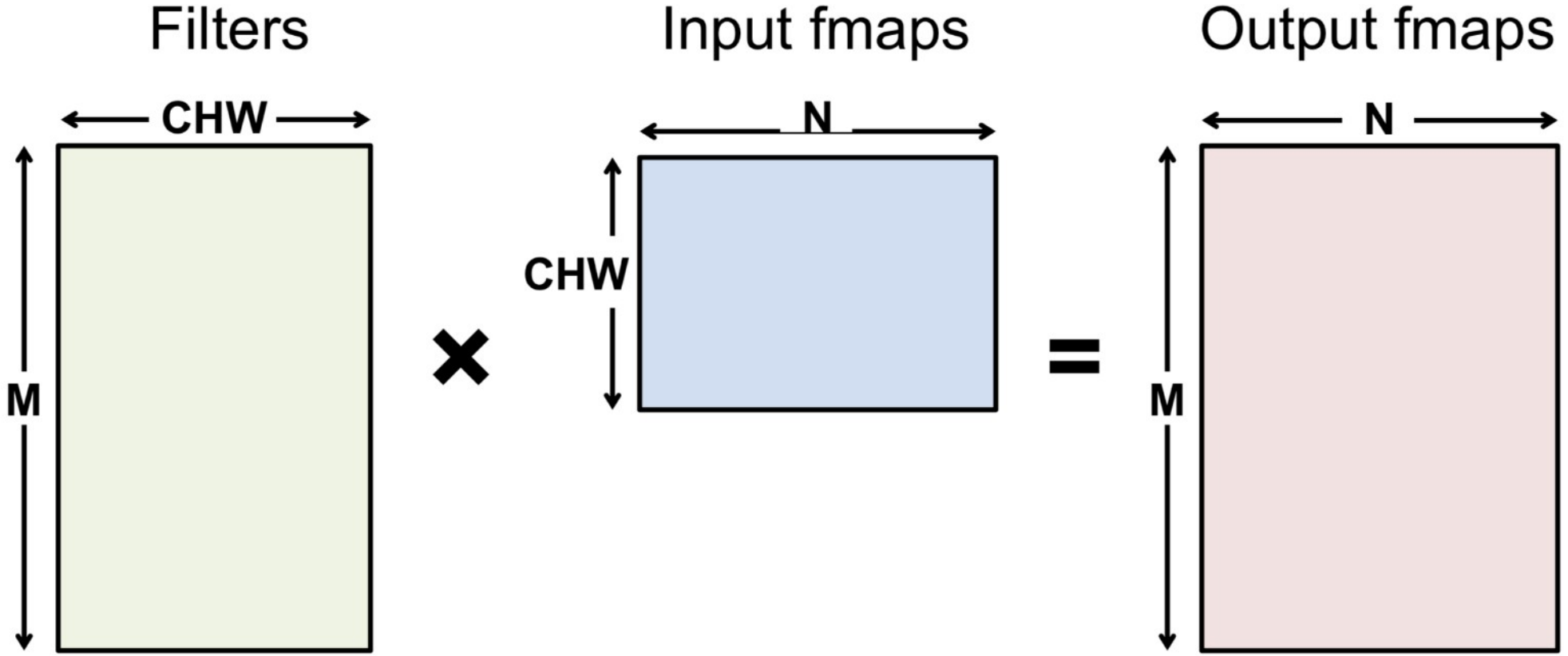
- Batch size > 1



[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Fully Connected Layer

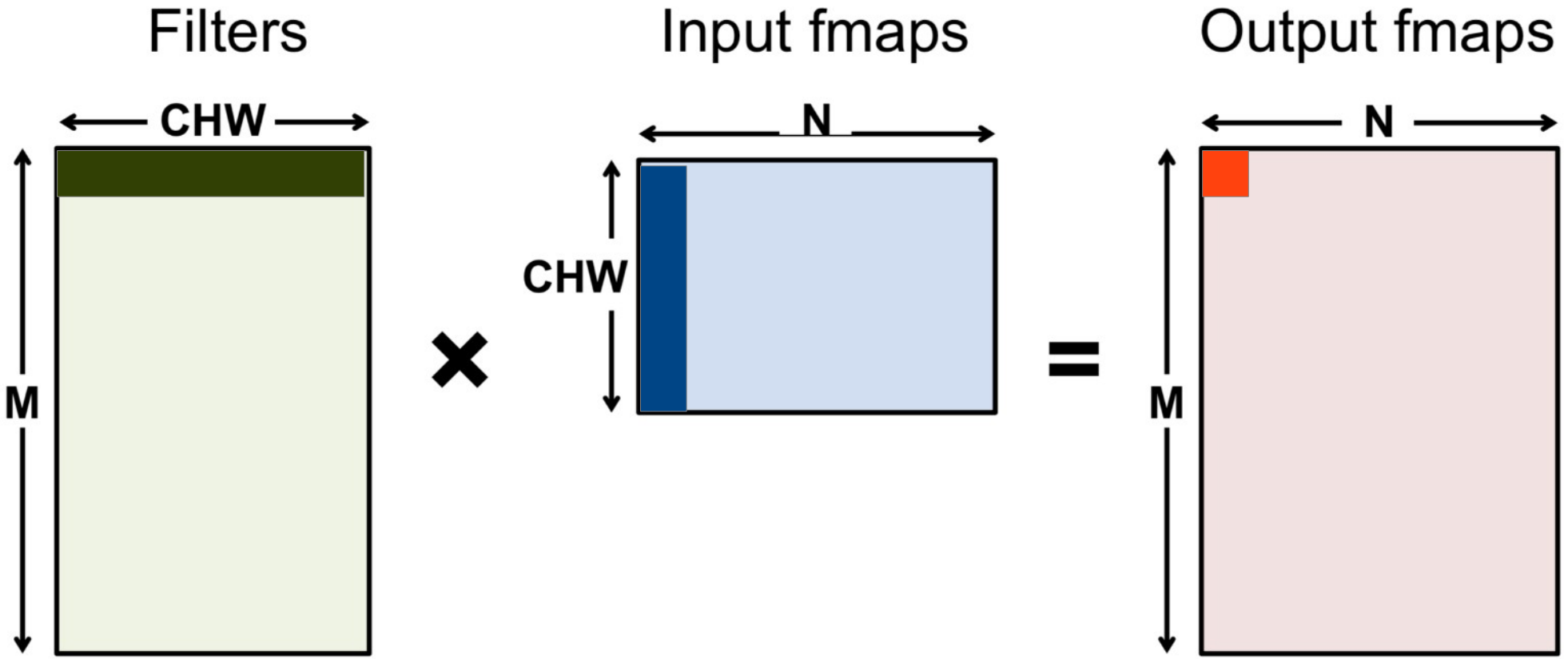
- Batch size > 1



[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Fully Connected Layer

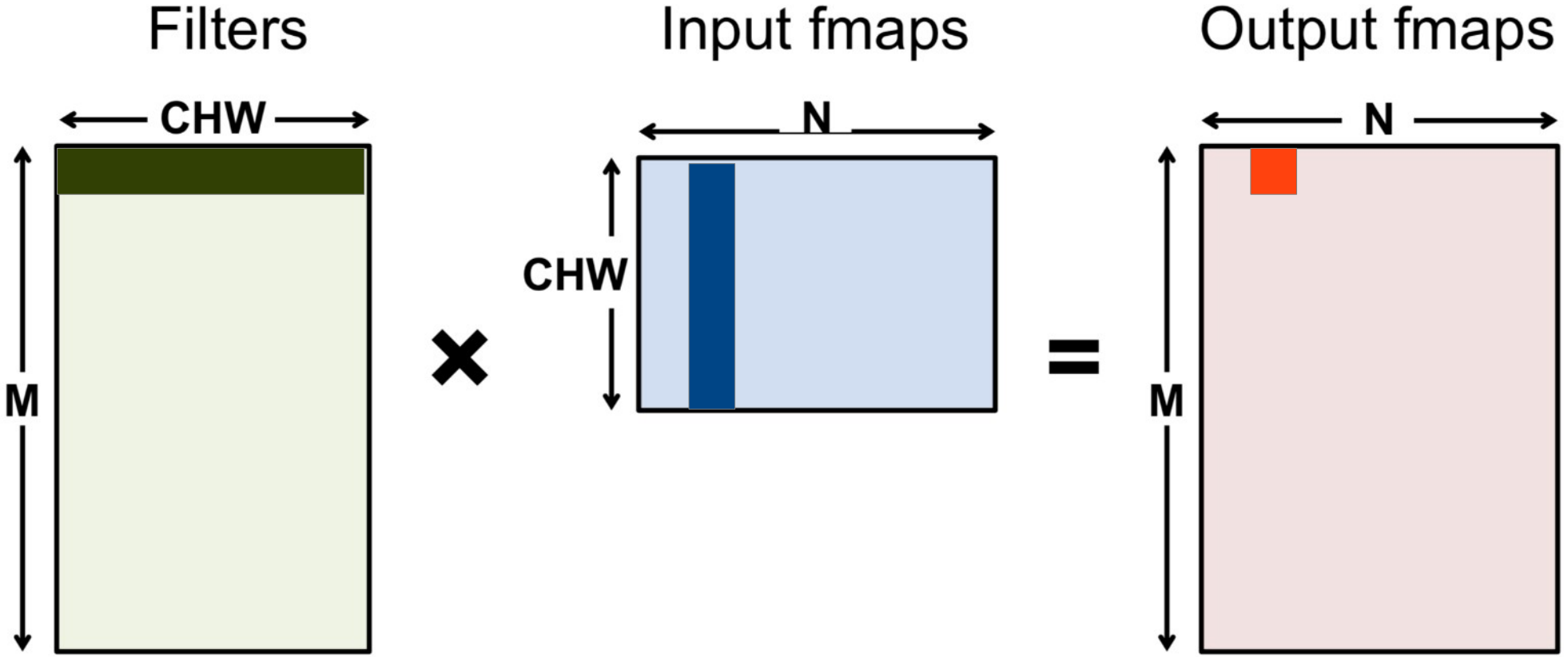
- Batch size > 1



[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Fully Connected Layer

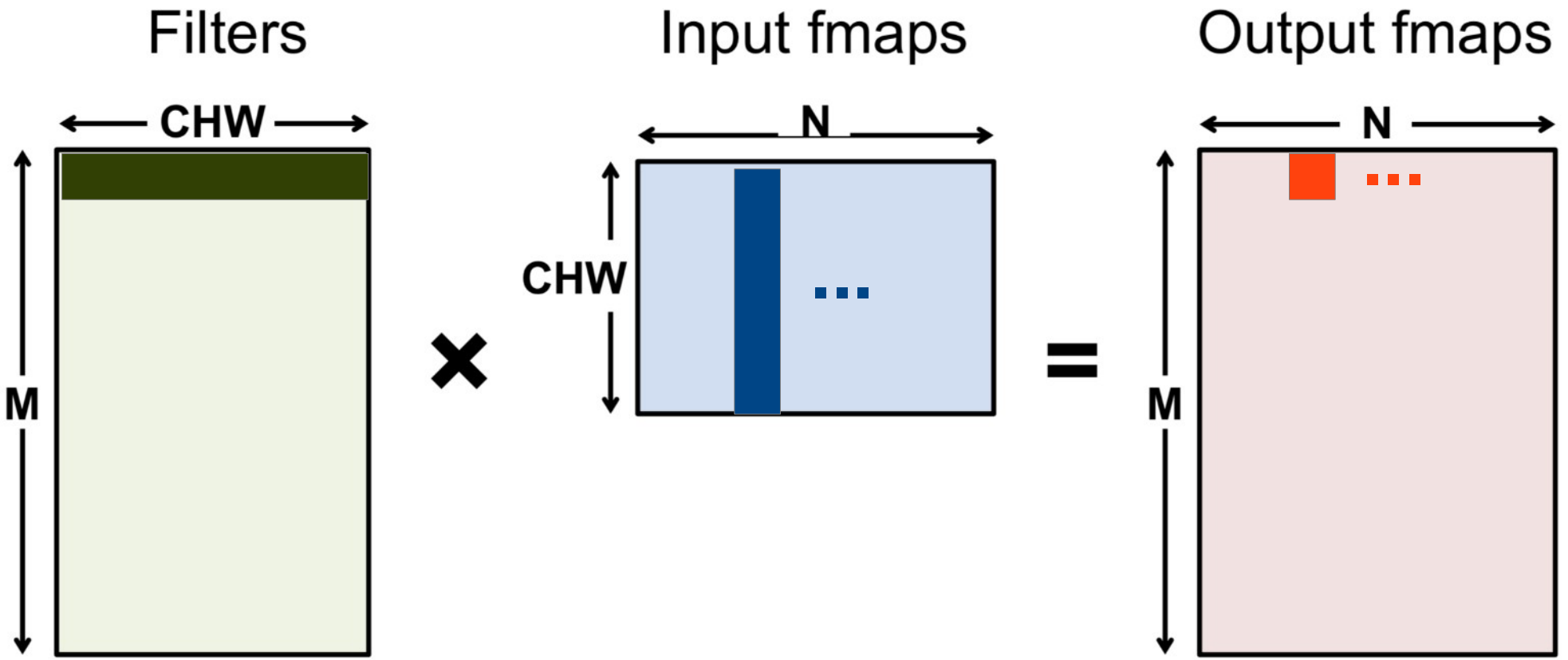
- Batch size > 1



[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Fully Connected Layer

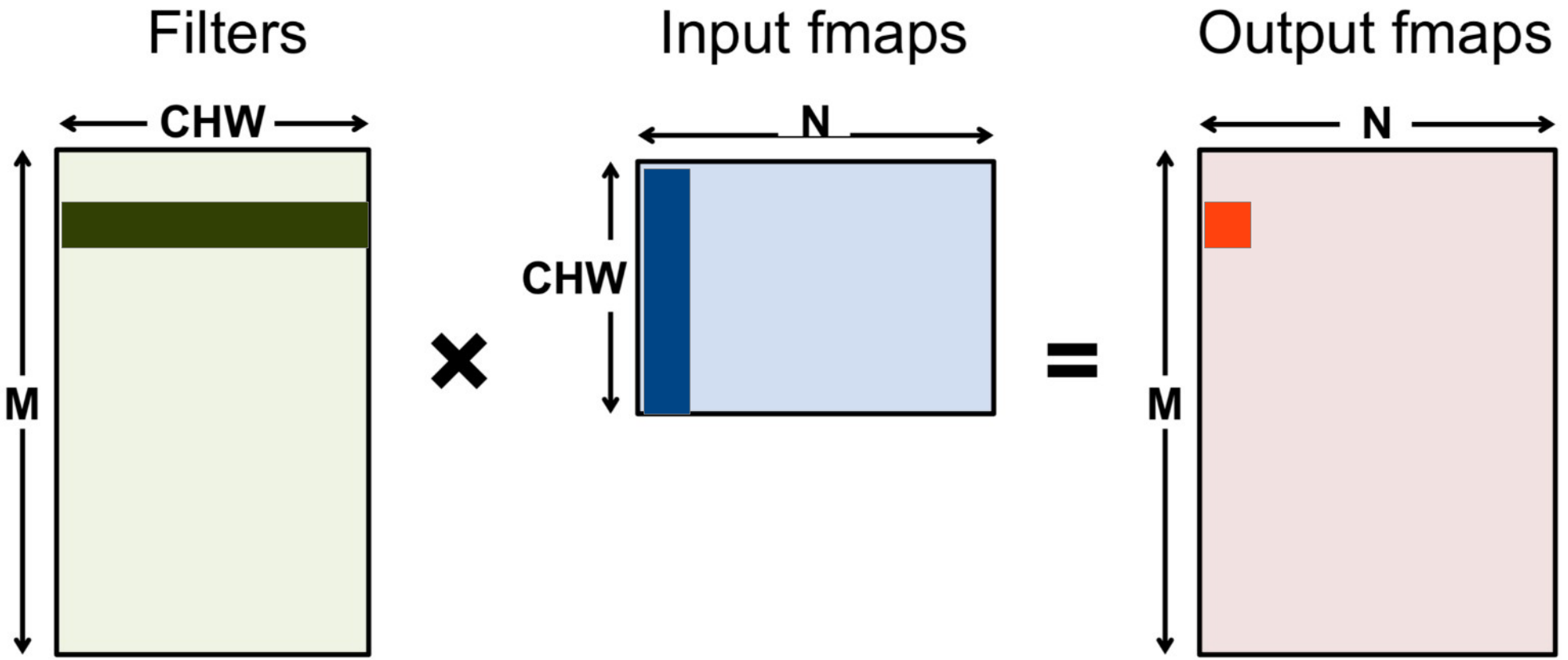
- Batch size > 1



[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Fully Connected Layer

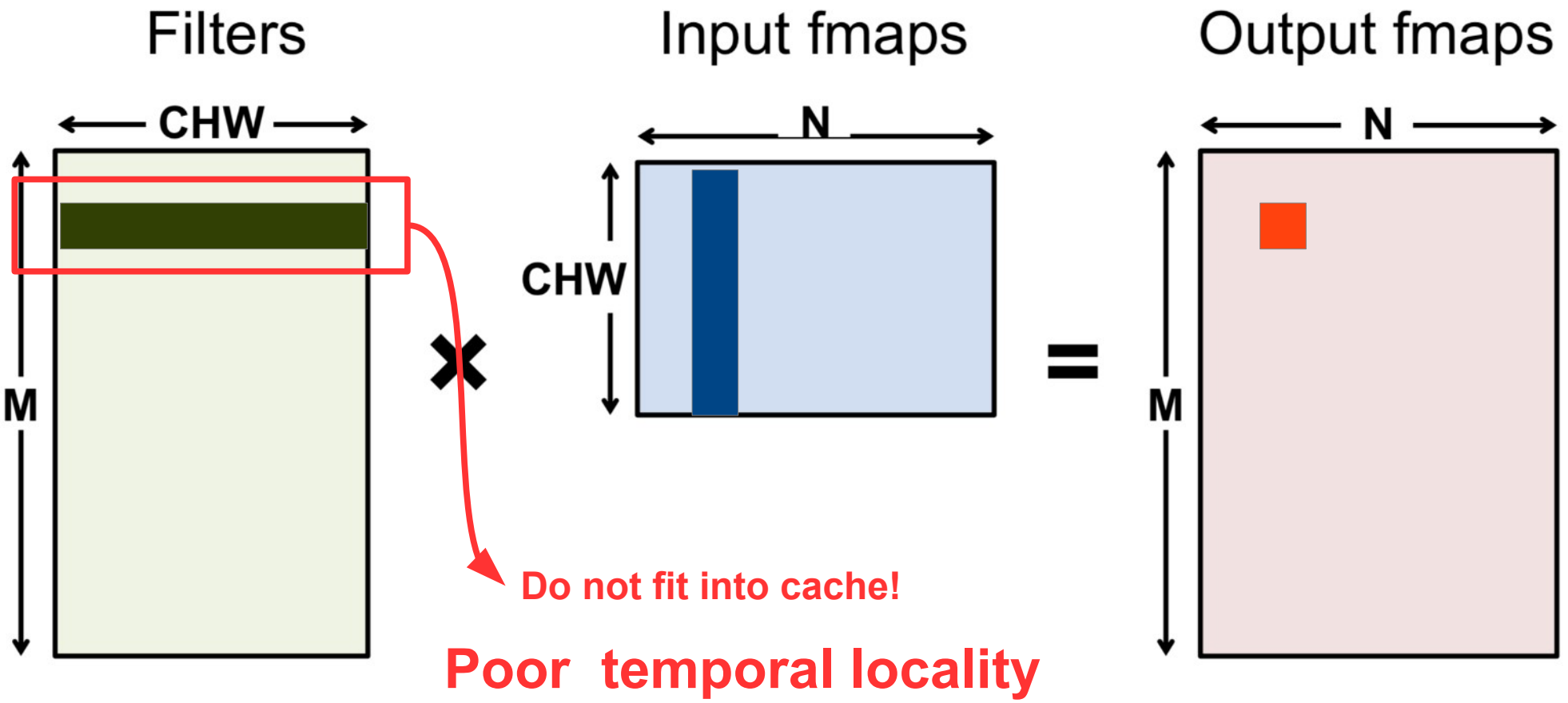
- Batch size > 1



[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Fully Connected Layer

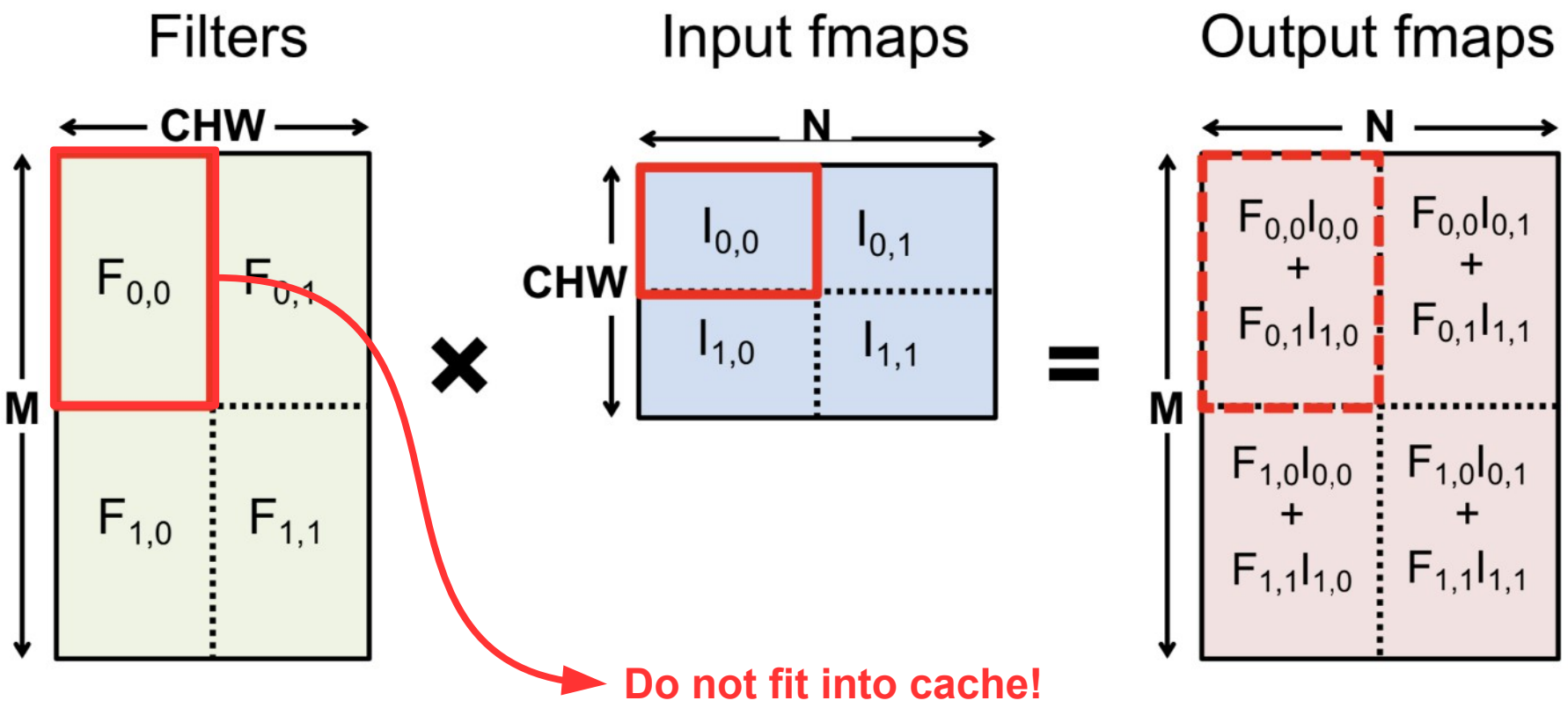
- Batch size > 1



[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Tiling

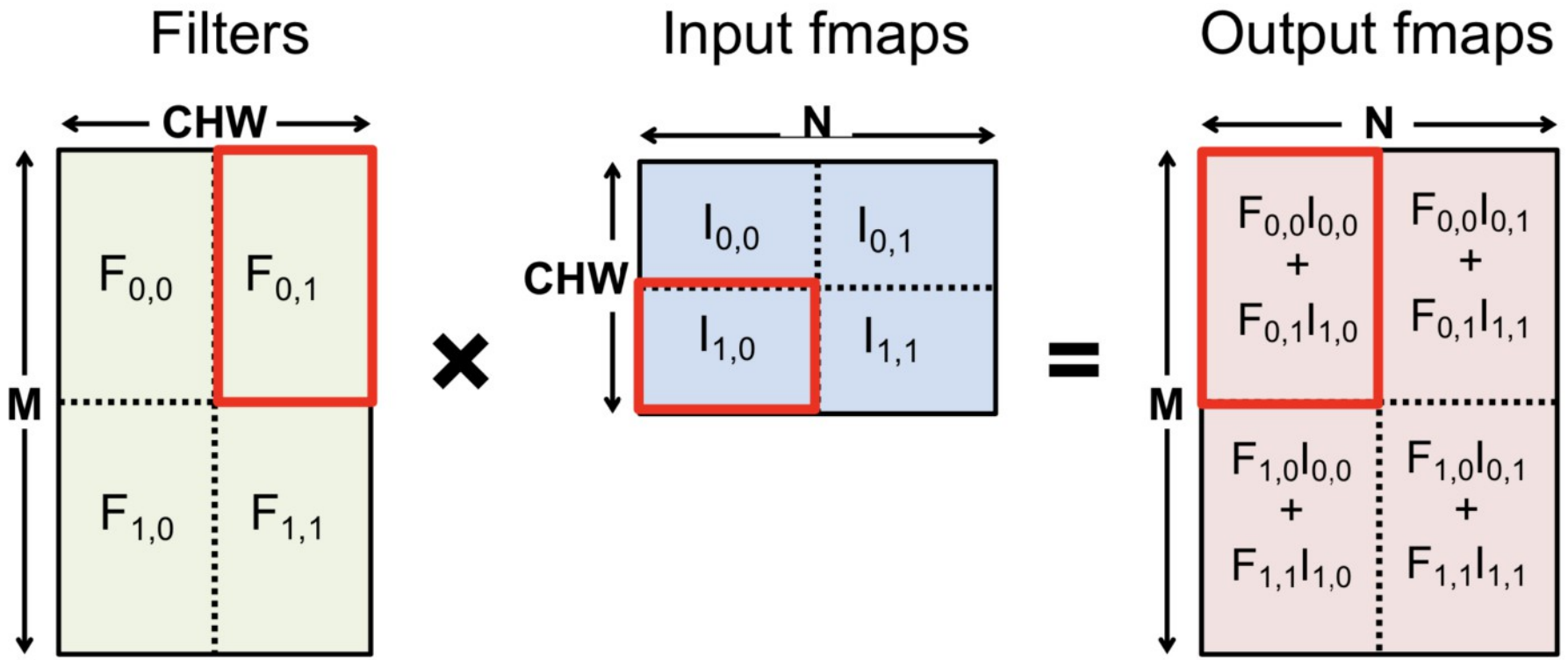
- Matrix multiply tiled to fit in cache
- Computation ordered to maximize data reuse



[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Tiling

- Matrix multiply tiled to fit in cache
- Computation ordered to maximize data reuse

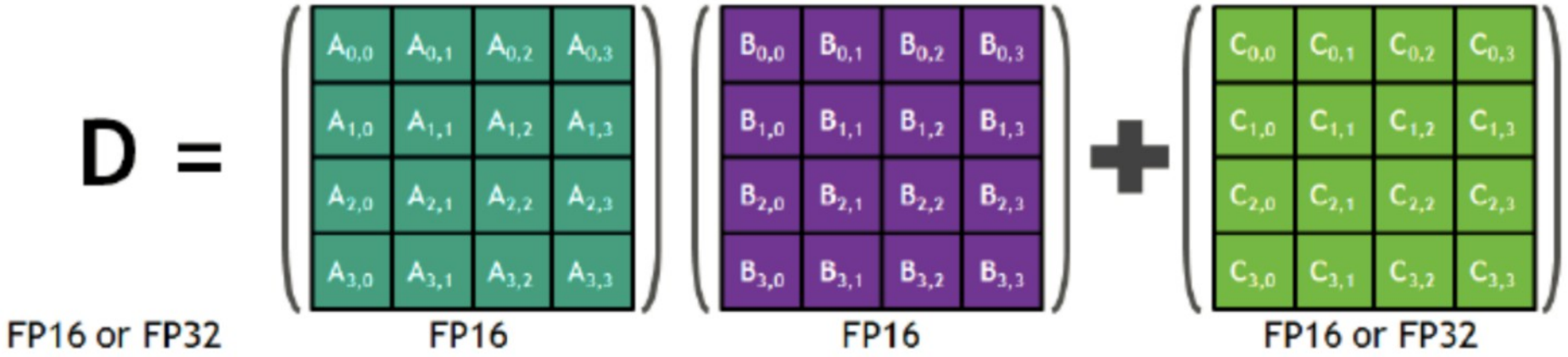


[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Implementation

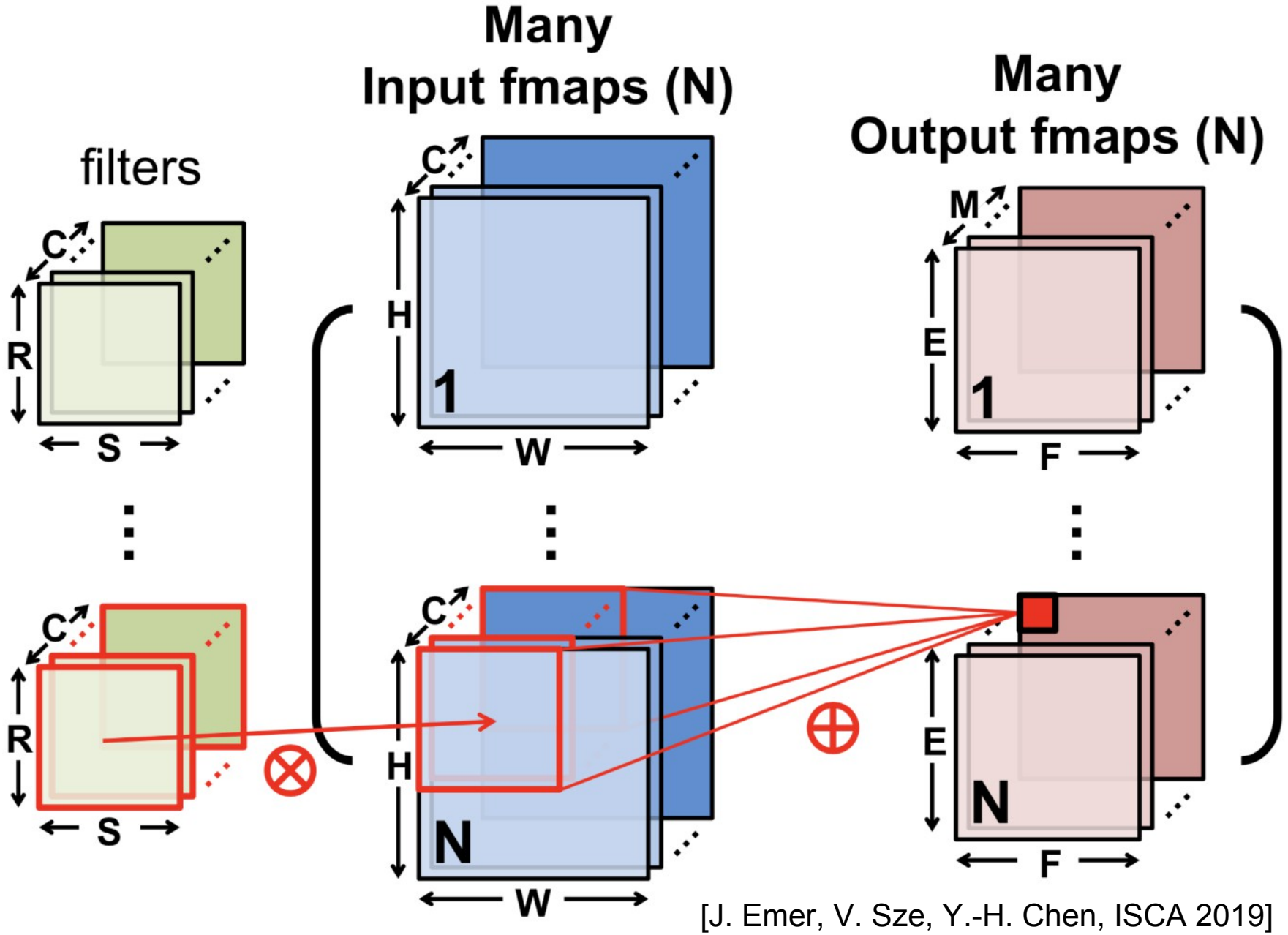
- Matrix Multiplication
  - CPU: OpenBLAS, Intel MKL, ...
  - GPU: cuBLAS, cuDNN, ...
- Library will note shape of the matrix multiply and select implementation optimized for that shape
- Optimization involves proper tiling to storage hierarchy

GV100 – Tensor Core



- Matrix Multiply Accumulate (HMMA)
  - FP16 operands → Inputs 48 / Outputs 16
  - Multiplies → 64
  - Adds → 64

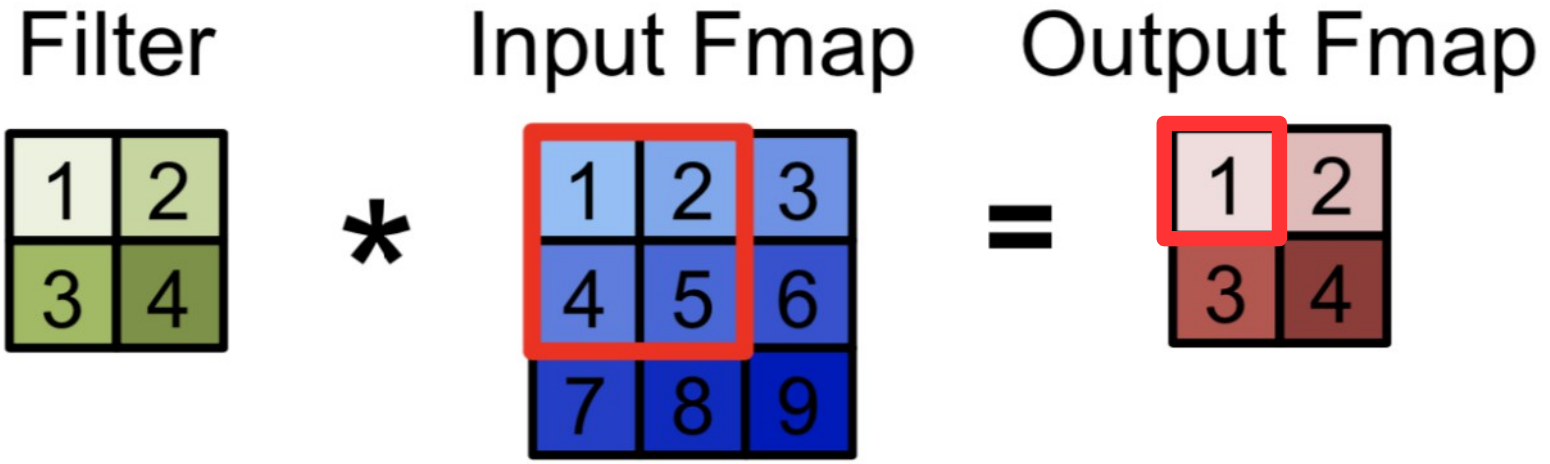
Convolution Layer



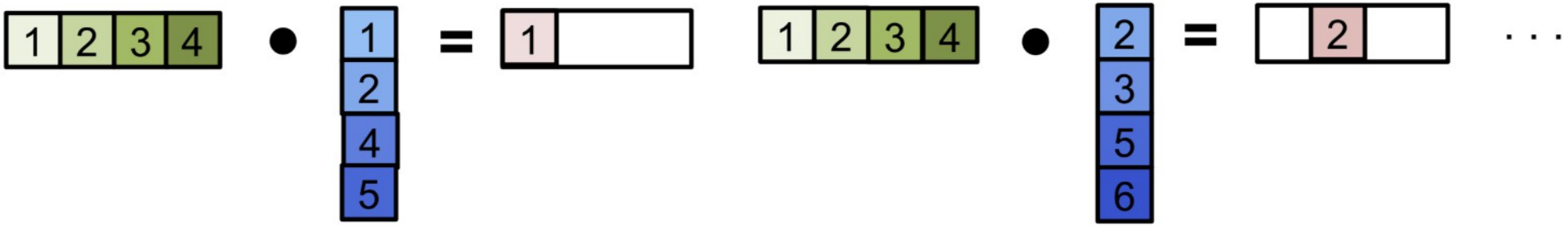
[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Convolution by 1D Products

Convolution

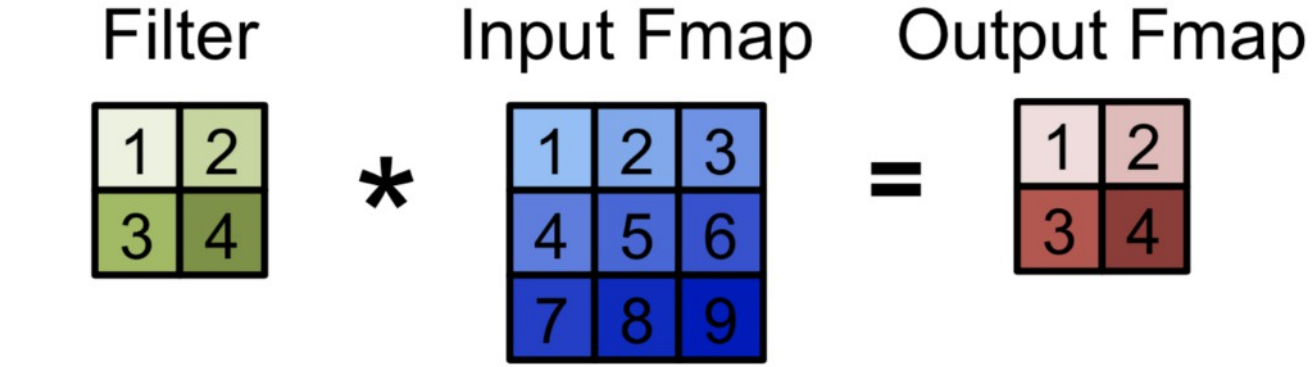


1D Product (flattened)



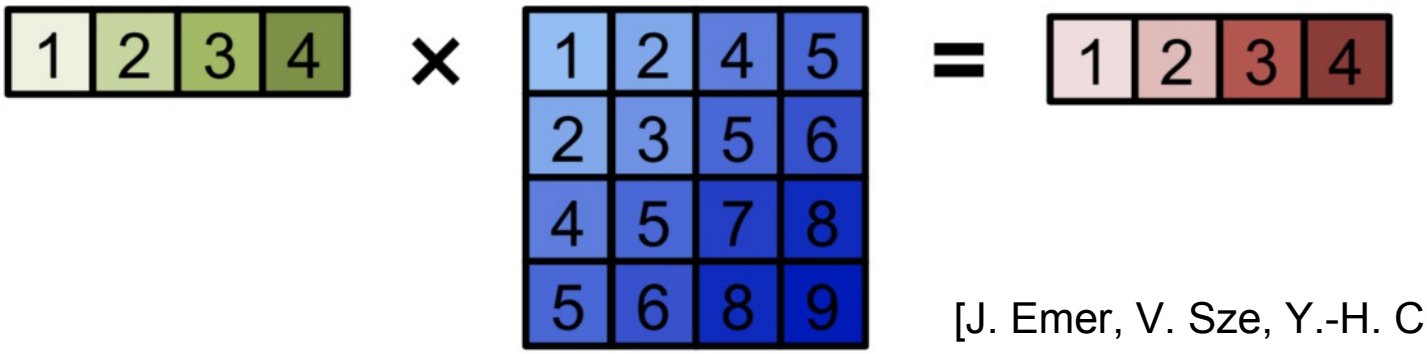
[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Convolution by Matrix Multiplication



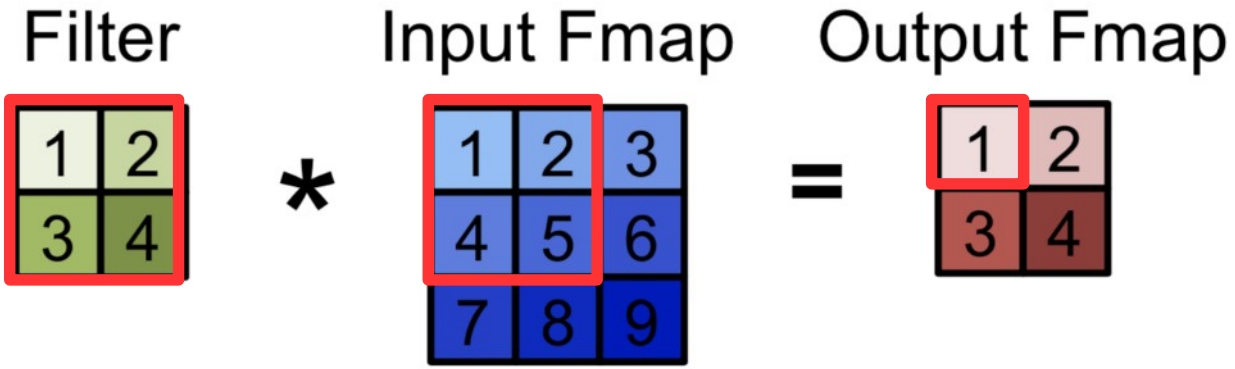
Convolution:

Matrix Multiply (by Toeplitz Matrix)



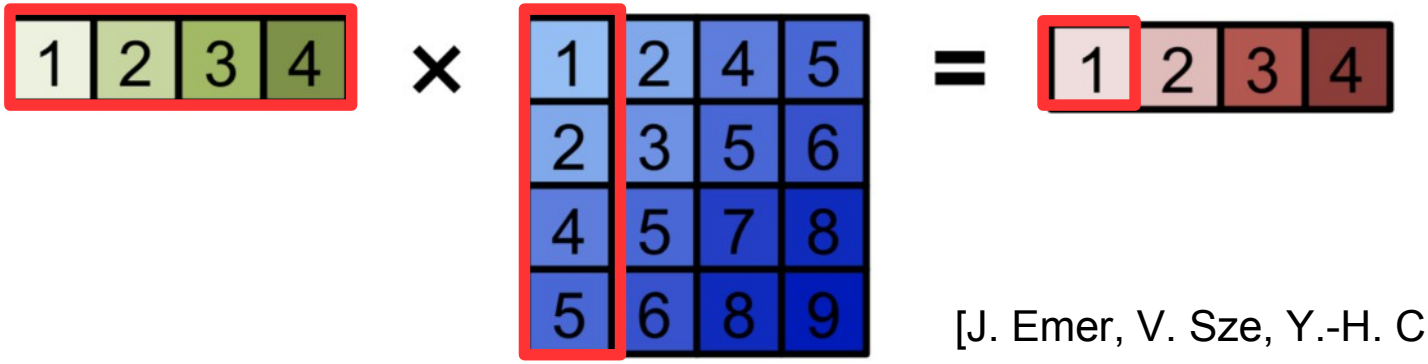
[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Convolution by Matrix Multiplication



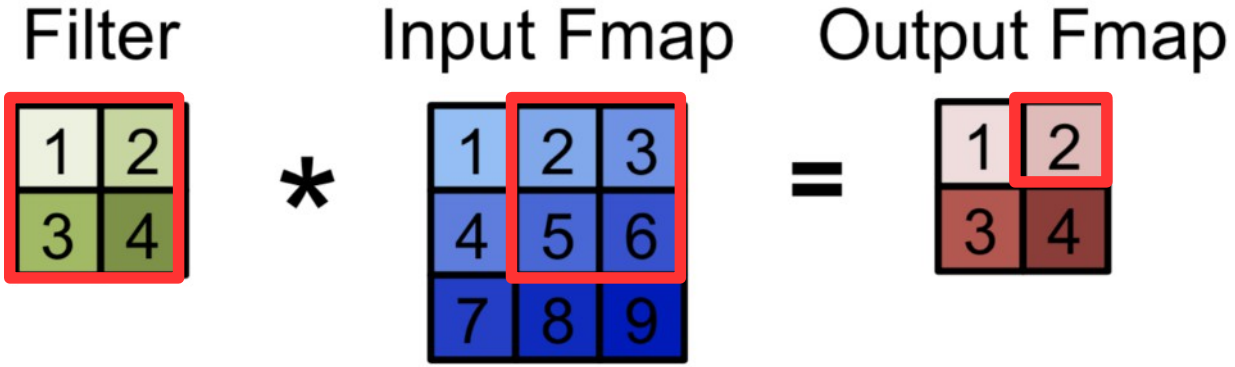
Convolution:

Matrix Multiply (by Toeplitz Matrix)



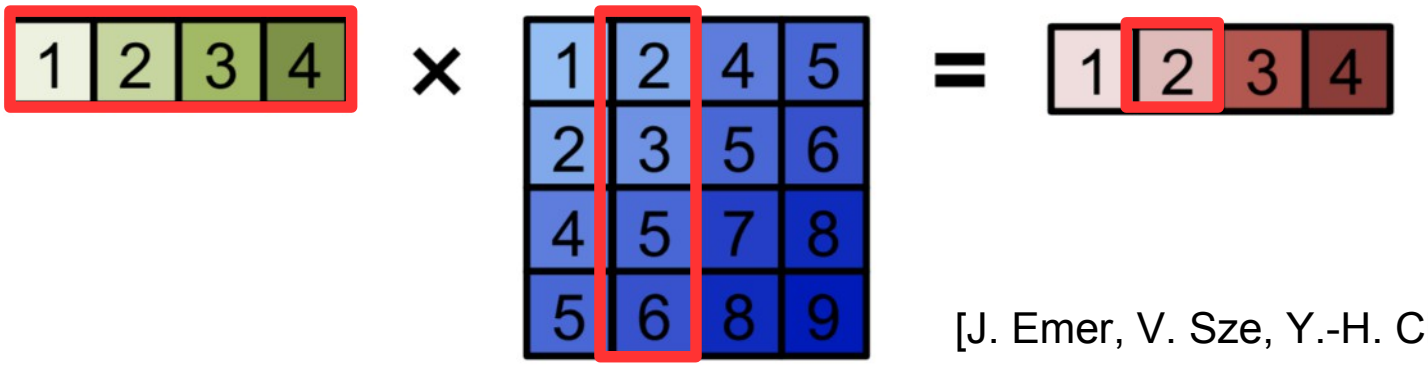
[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Convolution by Matrix Multiplication



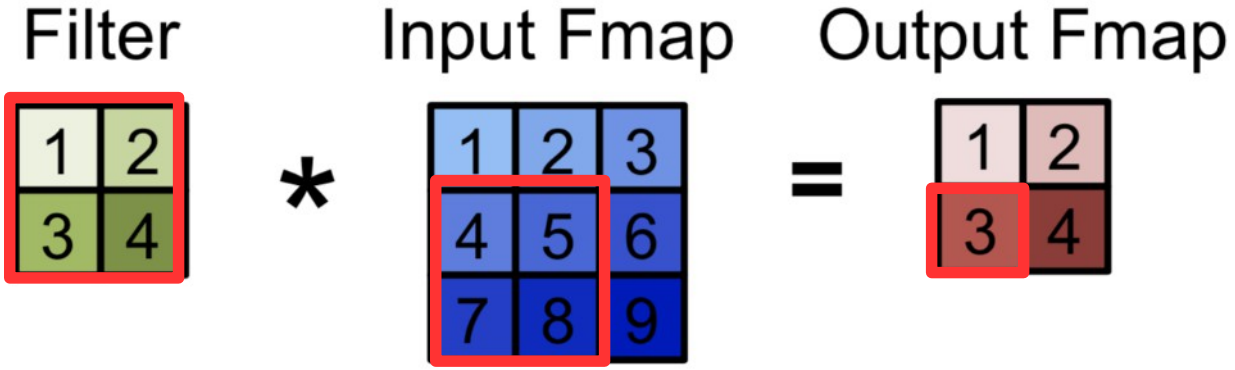
Convolution:

Matrix Multiply (by Toeplitz Matrix)



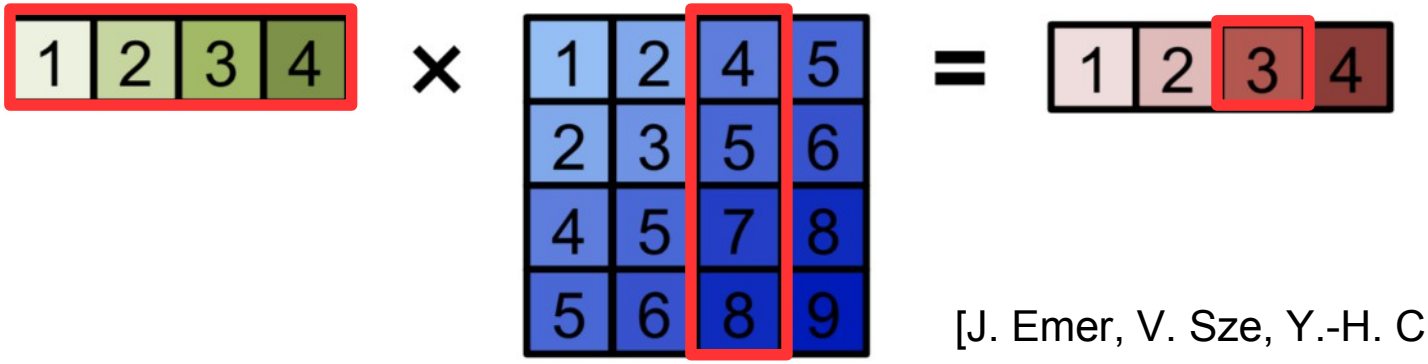
[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Convolution by Matrix Multiplication



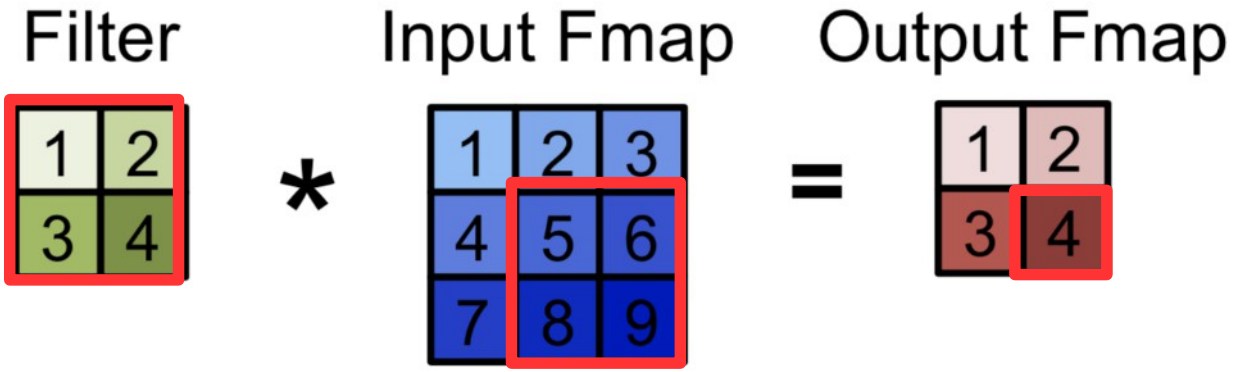
Convolution:

Matrix Multiply (by Toeplitz Matrix)



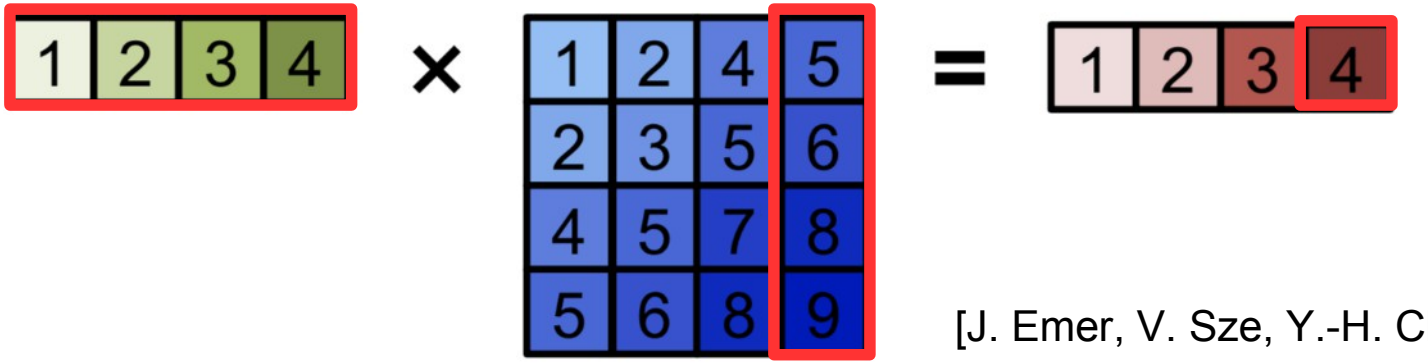
[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Convolution by Matrix Multiplication



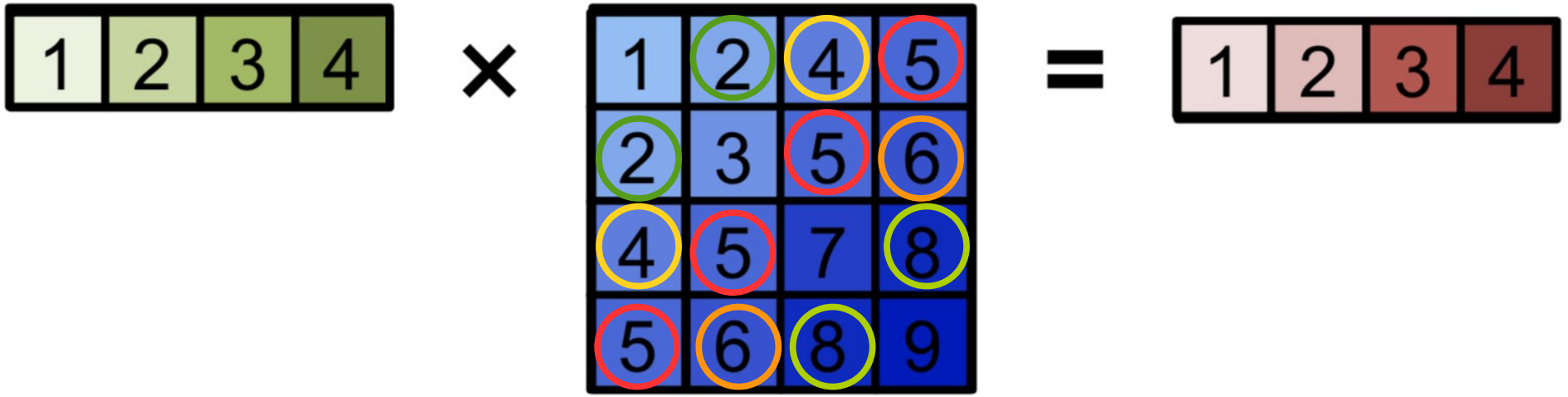
Convolution:

Matrix Multiply (by Toeplitz Matrix)



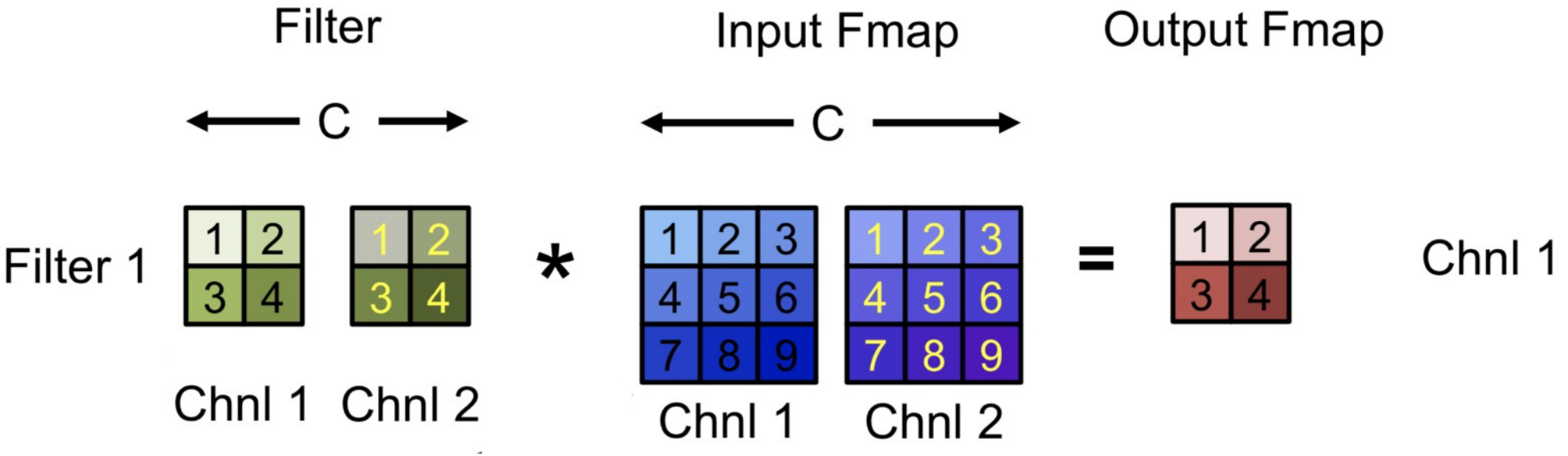
[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Convolution by Matrix Multiplication



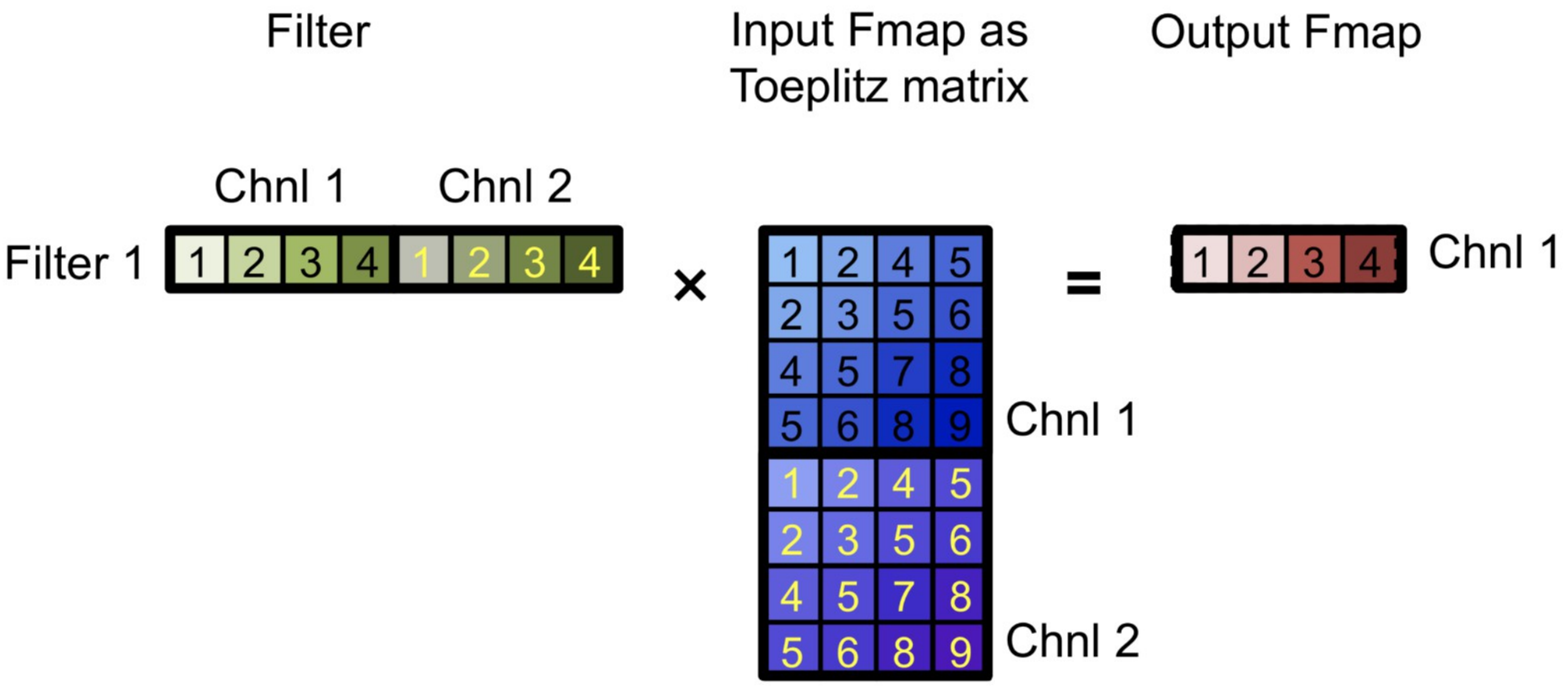
- Data is repeated
  - Memory *vs.* Computation trade-off

Convolution with Multiple Channels



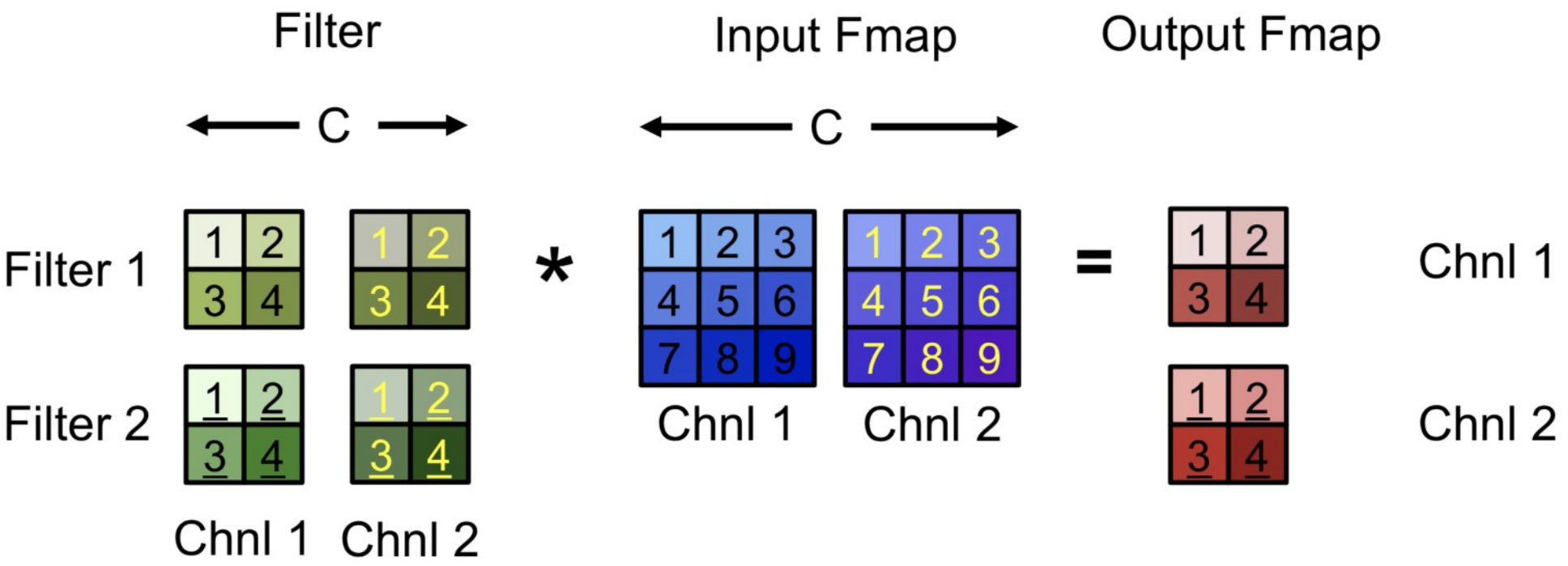
[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Convolution by Matrix Multiplication



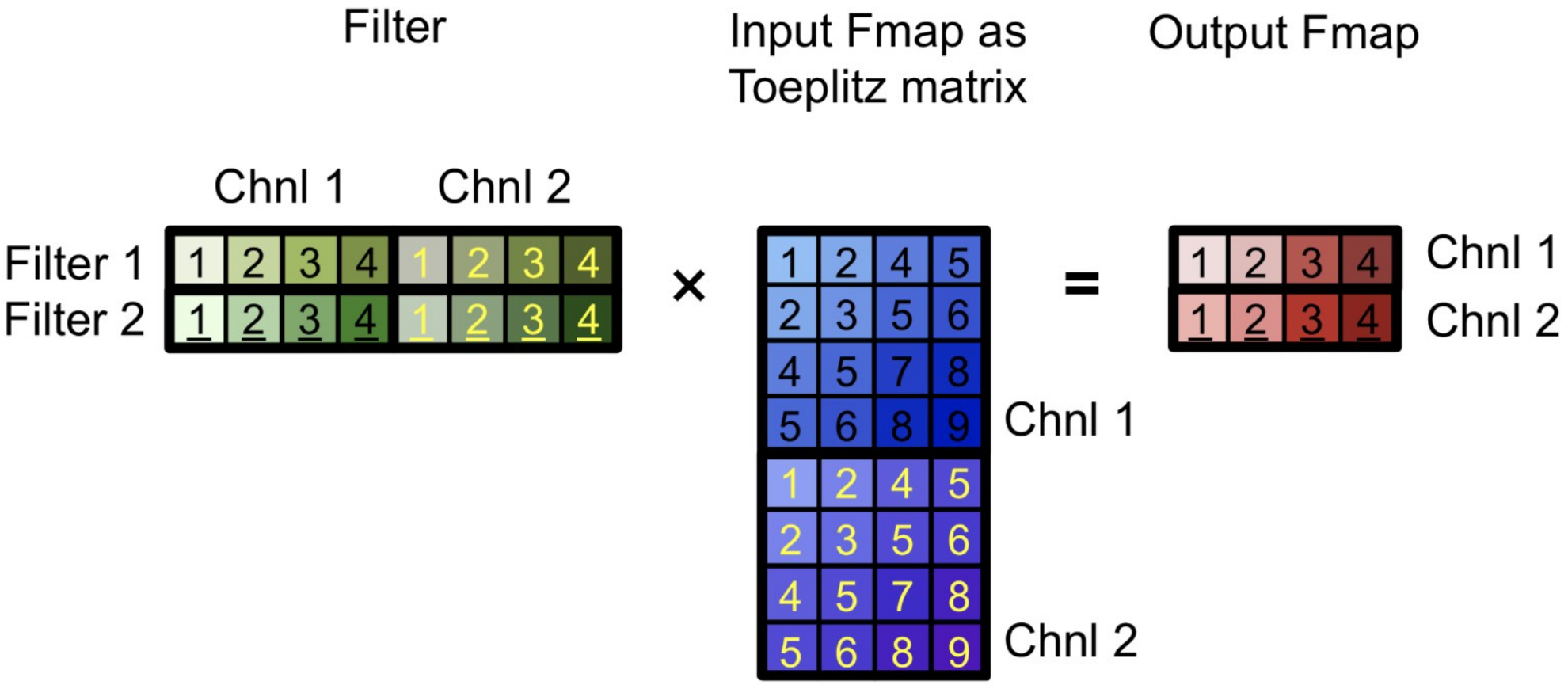
[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Convolution with Multiple Filters



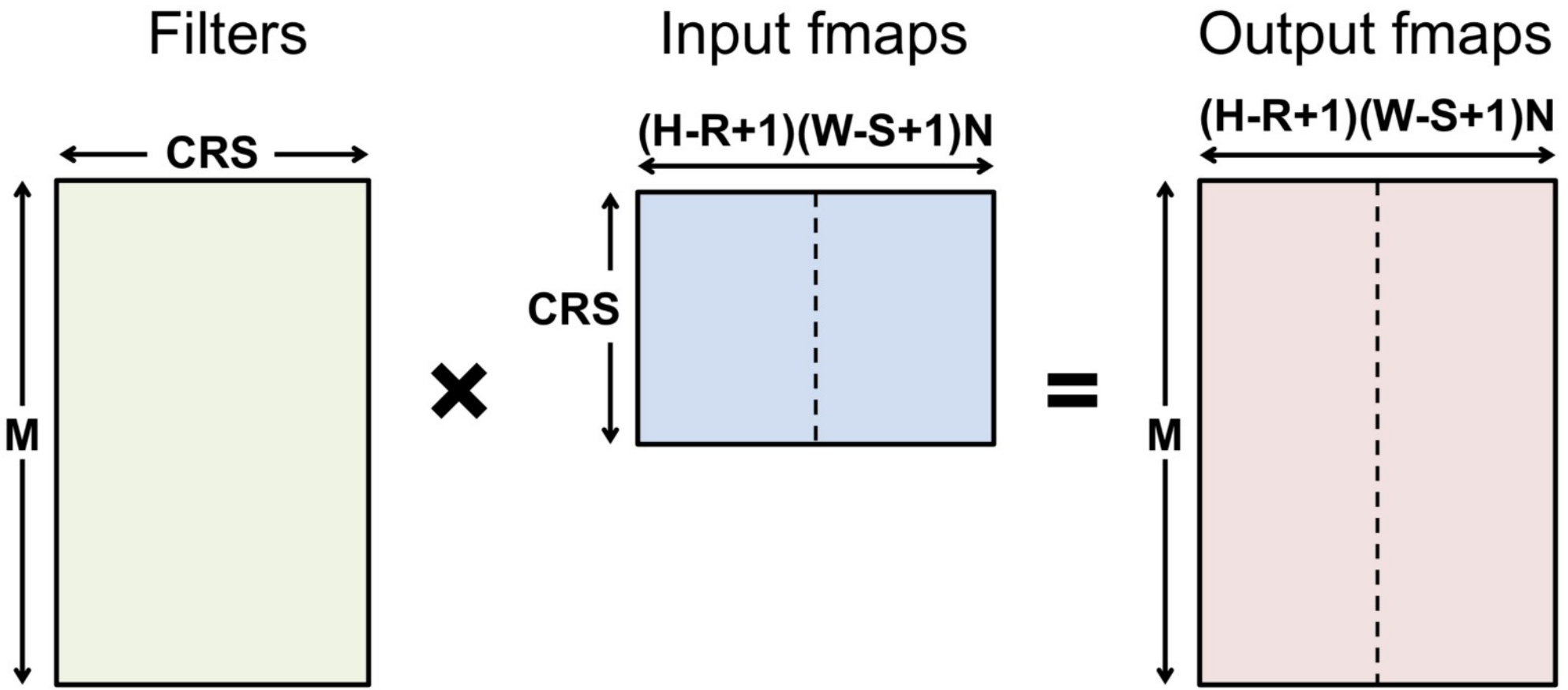
[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Convolution by Matrix Multiplication



[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Convolution with Batch size > 1



[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Computation Transformations

- Obtaining the same results with less computation
  - Gauss's multiplication
  - Strassen
  - Winograd
  - FFT

Gauss's Multiplication

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

4 multiplications + 3 additions

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

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

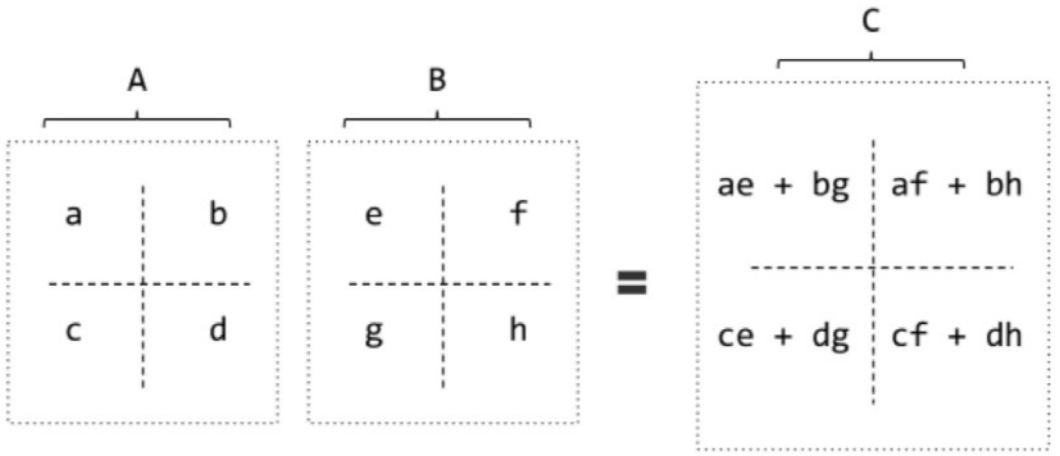
$k_3 = b \cdot (c + d)$

3 multiplications + 5 additions

Real part = $k_1 - k_3$

Imaginary part = $k_1 + k_2$.

Strassen



8 multiplications + 4 additions

$$P1 = a(f - h)$$
$$P2 = (a + b)h$$
$$P3 = (c + d)e$$
$$P4 = d(g - e)$$
$$P5 = (a + d)(e + h)$$
$$P6 = (b - d)(g + h)$$
$$P7 = (a - c)(e + f)$$

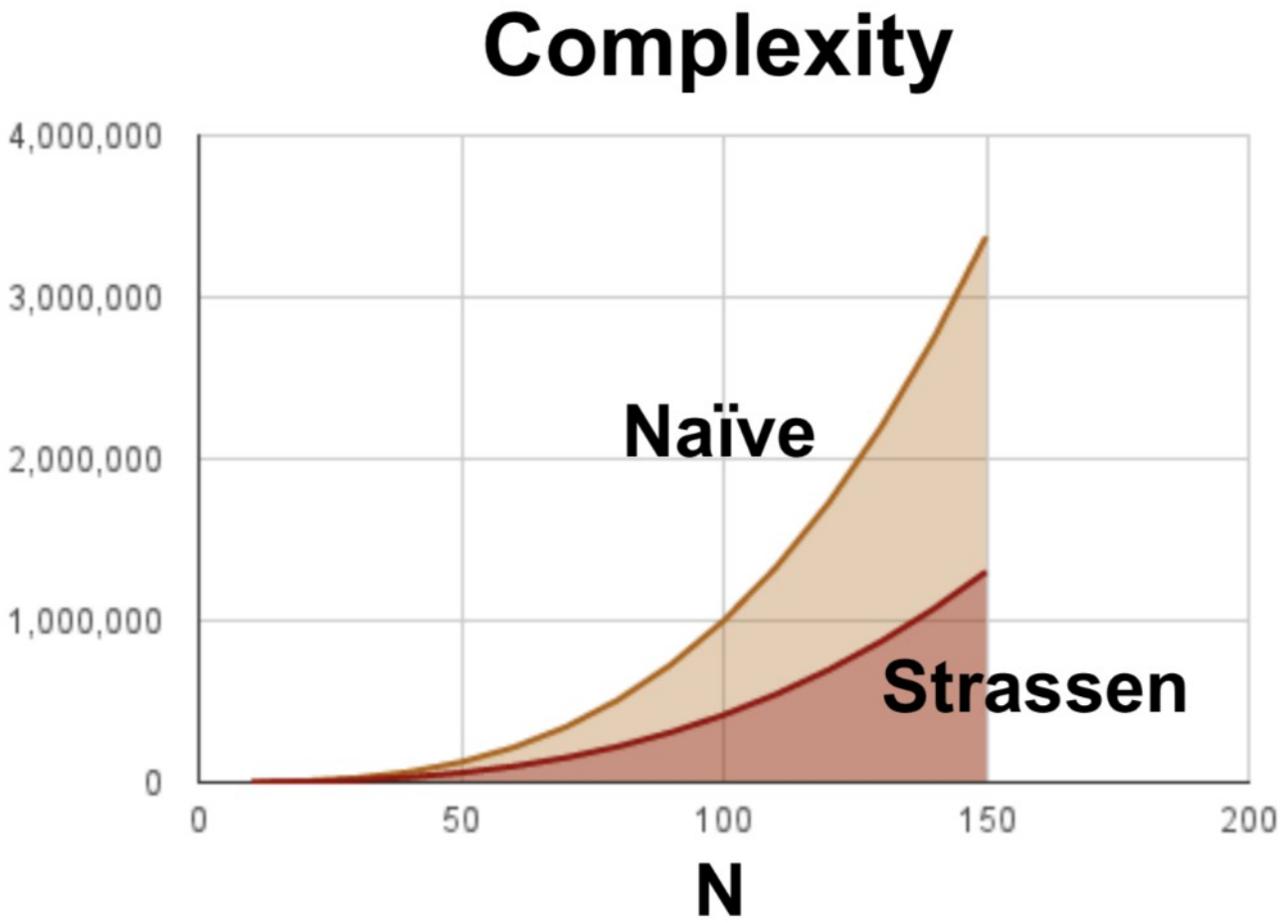
$$AB = \begin{bmatrix} P5 + P4 - P2 + P6 & P1 + P2 \\ P3 + P4 & P1 + P5 - P3 - P7 \end{bmatrix}$$

7 multiplications + 18 additions

[J. Cong and Bi. Xiao, Minimizing Computation in Convolutional Neural Networks. ICANN 2014]

Strassen

- Reduce the complexity of matrix multiplication from $O(N^3)$ to $O(N^{2.807})$



...but

- Reduced numerical stability
- Requires significantly more memory

Winograd 1D

- Used for convolutions

$$input = [d_0 d_1 d_2 d_3] \qquad filter = [g_0 g_1 g_2]$$

$$output = \begin{bmatrix} d_0 \times g_0 + d_1 \times g_1 + d_2 \times g_2 \\ d_1 \times g_0 + d_2 \times g_1 + d_3 \times g_2 \end{bmatrix}$$

Original:
6 multiplications, 4 additions

[Lavin & Gray, Fast Algorithms for Convolutional Neural Networks, CVPR 2016]

Winograd 1D

- Used for convolutions

input

$$\begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix}$$

filter

$$\begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix}$$

$$m_1 = (d_0 - d_2) g_0$$

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

$$m_4 = (d_1 - d_3) g_2$$

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

Winograd:
4 multiplications, 12 additions, 2 shifts

Filter weights are constant. Only need to be performed once!

[Lavin & Gray, Fast Algorithms for Convolutional Neural Networks, CVPR 2016]

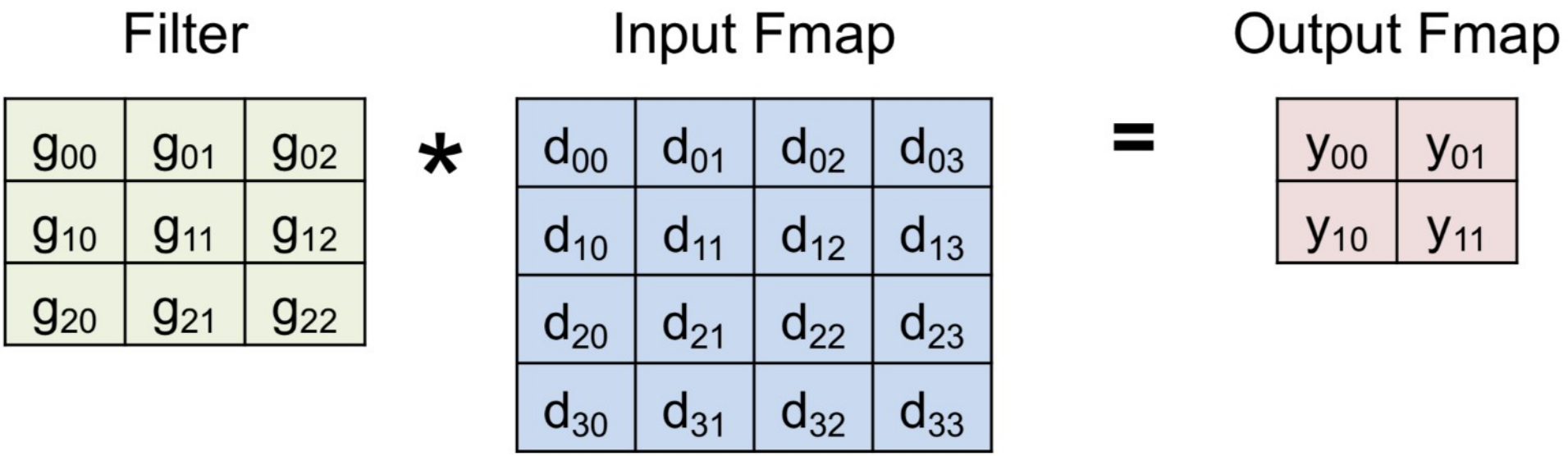
Winograd 1D – Observation

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

- Works on a small part of input
  - 4 elements in this example
- An intere convolution requires the application of Winograd on a tile-by-tile basis
- Reduction of multiplications depends on filter/tile size

[Lavin & Gray, Fast Algorithms for Convolutional Neural Networks, CVPR 2016]

Winograd 2D



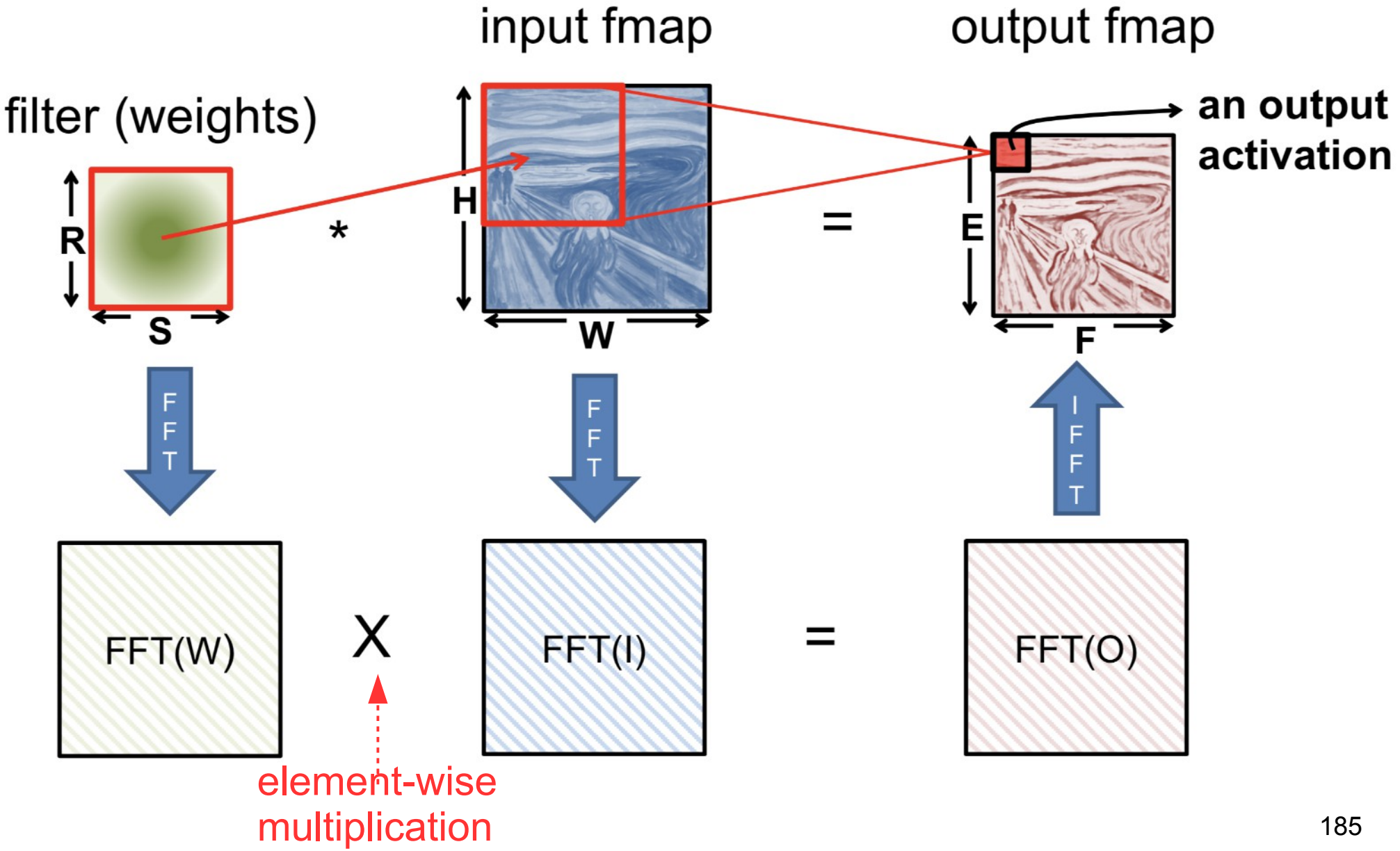
Original: 36 multiplications
Winograd: 16 multiplications → Speedup 2.25

[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

Winograd Summary

- Pros
  - Optimized computation for convolutions
  - Significantly reduce multiplies
- Cons
  - Each filter size (and output size) is a different computation

Fast Fourier Transform



[J. Emer, V. Sze, Y.-H. Chen, ISCA 2019]

FFT Cost

- Complexity
  - Convolution **O(*RSEF*)**
  - Multiplication **O($EF \log_2 EF$)**
- Computational benefit of FFT decreases with decreasing size of filter
- Large storage capacity and bandwidth
- Coefficients in the frequency domain are complex

FFT Optimization and Trade-offs

- FFT of real matrix is symmetric
  - Save 1/2 of the computations
- Filters can be pre-computed and stored
  - In frequency domain they are much larger than in space domain

Agenda

- Deep Neural Networks
- Types of layers
- Memory and communication traffic
- Kernel computation
- Quantitative analysis

Key Metrics and Design Objectives

- Accuracy
- Throughput and Latency
- Energy
- ...

Throughput

- Number of execution of a task that can be completed in a given time period
  - *e.g*, infereces per second

Latency

- Time between the beginning of a task and its completion
  - *e.g.*, inference latency

Fallacies

- If throughput increases latency decreases and viceversa
- Example
  - Want throughput of 30 fps
  - With a bach size of 100 frames → Amortize overhead due to load weights → 100 frames in 1s (more than 30 fps required)
  - What about latency for a single frame?
    - Results after filling up the batch
    - Batch filled up in 3.3 sec
    - Latency > 3.3 sec! Not acceptable for real-time applications (*e.g.*, high-speed navigation)

Throughput – Quantitative Analysis

$$\frac{inferences}{second} = \frac{operations}{second} \times \frac{1}{\frac{operations}{inference}}$$

Depends on
- DNN hardware
- DNN model

Depends on
- DNN model

Throughput – Quantitative Analysis

$$\frac{operations}{second} = \frac{1}{\frac{cycles}{operation}} \times \frac{cycles}{second}$$

Peak throughput
(aggregate for **all** PEs)

Throughput – Quantitative Analysis

$$\frac{operations}{second} = \left(\frac{1}{\frac{cycles}{operation}} \times \frac{cycles}{second} \right) \times number\ of\ PEs$$

Peak throughput
(aggregate for **single** PEs)

Throughput – Quantitative Analysis

$$\frac{operations}{second} = \left(\frac{1}{\frac{cycles}{operation}} \times \frac{cycles}{second} \right) \times number\ of\ PEs \times utilization\ of\ PEs$$

Peak throughput
(aggregate for **single** PEs)

Throughput – Quantitative Analysis

$$\frac{operations}{second} = \left(\frac{1}{\frac{cycles}{operation}} \times \frac{cycles}{second} \right) \times number\ of\ PEs \times utilization\ of\ PEs$$

Peak throughput of a PE

Parallelism

Degradation due to the inability to utilize the PE

Throughput – Quantitative Analysis

$$\frac{operations}{second} = \left(\frac{1}{\frac{cycles}{operation}} \times \frac{cycles}{second} \right) \times number\ of\ PEs \times utilization\ of\ PEs$$

Decrease the cycles for performing an operation (*e.g.*, non-pipelined multi-cycle MAC vs. pipelined MAC)

Increase the clock frequency (*e.g.*, micro-architectural improvement)

If area is fixed
- Reducing PE area
- Trade-off with storage area → impact on PEs utilization

Mapping and data-flow technique

Throughput – Quantitative Analysis

$$utilization\ of\ PEs = \underbrace{\frac{number\ of\ active\ PEs}{number\ of\ PEs}}_{\text{Ability to distribute the workload among PEs}} \times \underbrace{utilization\ of\ active\ PEs}_{\text{How efficiently active PEs process the workload}}$$

Throughput – Quantitative Analysis

$$utilization\ of\ PEs = \frac{number\ of\ active\ PEs}{number\ of\ PEs} \times utilization\ of\ active\ PEs$$

- Number of PEs that receive work
- Flexibility of the architecture plays a key role
  - On-chip network able to support different layer shapes
- Mapping
  - Placement and scheduling in place and time of operations
  - Delivery of the data to PEs

Throughput – Quantitative Analysis

$$utilization\ of\ PEs = \frac{number\ of\ active\ PEs}{number\ of\ PEs} \times utilization\ of\ active\ PEs$$

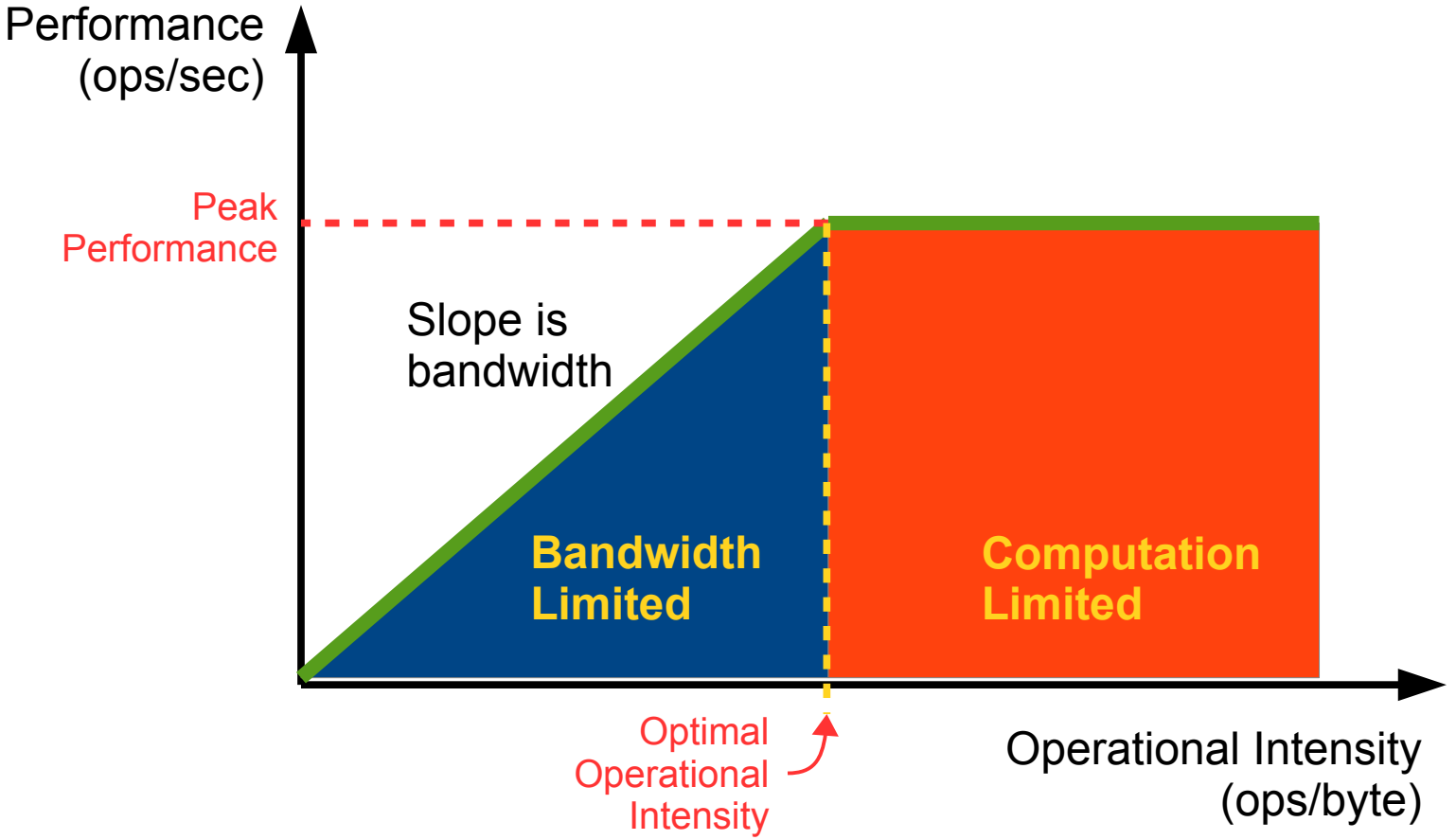
- Timely arrival of operands to PEs
  - PEs don't become idle while waiting for data to arrive
- Causes that affect utilization
  - Bandwidth and latency of memory and communication
    - Affected by the amount of data reuse
      - Data-flow techniques
      - Use of large batch size
  - Imbalance of work allocated across PEs
    - *E.g.*, due to sparsity

Throughput – Quantitative Analysis

$$utilization\ of\ PEs = \frac{number\ of\ active\ PEs}{number\ of\ PEs} \times utilization\ of\ active\ PEs$$

- Interplay PEs utilization *vs.* number of PEs
  - Use local memory as buffer for avoid PEs waiting for data → increase PE size → reduce the number of PEs

Roofline Model



Throughput – Other Observations

$$\frac{inferences}{second} = \frac{operations}{second} \times \frac{1}{\frac{operations}{inference}}$$

Depends on
- DNN hardware
- DNN model

Depends on
- DNN model

- Efficient Network Architectures
  - DNN models with efficient layer shapes
  - Reduce MAC operations → improve ops/inference
  - **...but** wide range of layer shapes → might result in poor utilization of PEs → reduce ops/second

Throughput – Other Observations

$$\frac{inferences}{second} = \frac{operations}{second} \times \frac{1}{\frac{operations}{inference}}$$

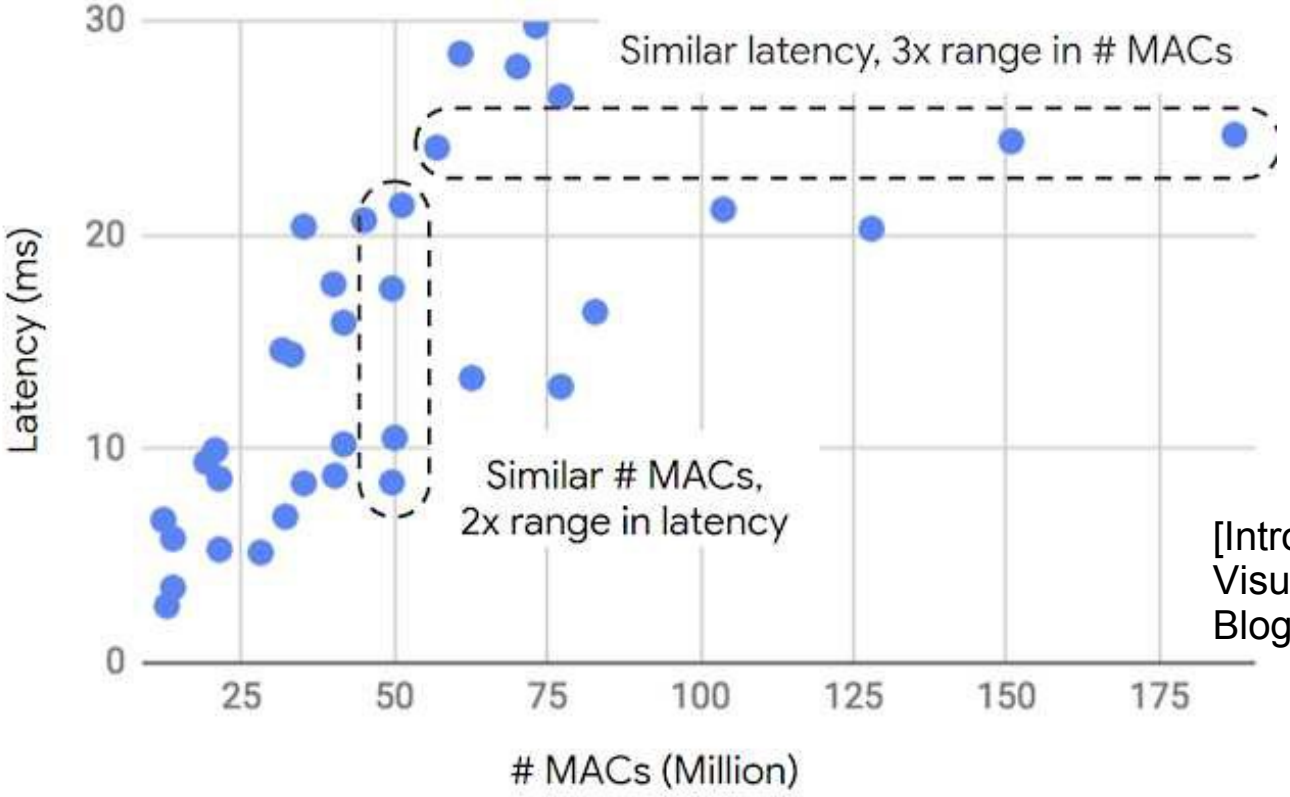
- Not all the operations are the same
  - *e.g.*, multiply by 0 → *ineffectual operations*
  - HW exploiting ineffectual operations → increase ops/sec
  - **...but** overhead in HW → increase of the critical path and/or decrease the number of PEs if area fixed

Throughput – Other Observations

- Designing HW and DNN models that support reduced precision
  - Increase the number of operations per second
  - Required memory bandwidth decreases → Increase PE utilization
  - PE area decreases → Increase the number of PEs
  - **...but**, if multiple level of precision need to be supported → HW overhead → Increase of the critical path, *etc.*

Throughput – Summary

- Number of MAC operations alone **is not sufficient** for evaluating throughput (and latency)



[Introducing the CVPR 2018 On-Device Visual Intelligence Challenge, Google AI Blog 2018]

- Need for designing DNN models with HW in the loop

Energy

- Importance of energy in DNN processing
  - Edge *vs.* Cloud processing
    - Privacy, latency, comm. BW limitation...
- Energy efficiency
  - Amount of work that can be complited with for unit of energy

Power *vs.* Energy

- Power consumption
  - Energy consumed per unit time
  - Related to heat dissipation
  - Important when DNN processed in the cloud
    - Data centers have stringent cooling costs
  - Important in wereable devices
    - Form factor limited by cooling mechanisms

Power *vs.* Energy

- Energy consumption
  - Integral of the power
  - Important in mobile terminals
    - Battery has finite energy
    - Determines life-time of a device before recharging
    - Affects form factor/weights of the device

Energy – Quantitative Analysis

$$\frac{inferences}{second} \leq max\left(\frac{joules}{second}\right) \times \frac{inferences}{joule}$$

Thermal Design Power (TPD)
Maximum power consumption

If inference per joule increases, inference per second can be increased

Energy – Quantitative Analysis

$$\frac{inferences}{joule} = \frac{operations}{joule} \times \frac{1}{\frac{operations}{inference}}$$

Depends on

- DNN hardware
- DNN model

Energy – Quantitative Analysis

- Energy per operation (in general)

$$\frac{joules}{operation} = \alpha \times C \times V_{dd}^2$$

- Two components

$$Energy_{total} = Energy_{data} + Energy_{MAC}$$

Dominant contribution

Energy – Quantitative Analysis

Operation	Energy (pJ)	Operation	Energy (pJ)
32 bit int ADD	0.1	32 bit float MULT	3.7
32 bit float ADD	0.9	32 bit SRAM memory	5.0
32 bit register file	1.0	32 bit NoC hop[a]	13.4
32 bit int MULT	3.1	32 bit DRAM memory	640.0

45 nm CMOS process

[M. Horowitz, “Energy table for 45nm process,” Stanford VLSI Wiki.]

Energy – Quantitative Analysis

$$Energy_{total} = Energy_{data} + Energy_{MAC}$$

- Reducing $Energy_{data}$
  - Improving memory system
  - Improving communication system
  - Improving dataflow technique

Energy – Quantitative Analysis

$$Energy_{total} = Energy_{data} + Energy_{MAC}$$

- Reducing $Energy_{MAC}$
  - Reducing precision
    - *e.g.*, 8-bit add requires 10× less energy than 32-bit add
    - Energy *vs.* Accuracy trade-off
  - For instruction based systems (GP, ASIP, GPU)
    - Use large aggregate instructions (SIMD/Vector)
    - *e.g.*, HMMA, IMMA Nvidia Volta V100 GPU

Summary

- Common layers in DNN
  - CONV, FC, Pooling, expedients
- Kernel computation
  - Layer computation led to matrix multiplication
- Induced traffic
  - Communication/Memory play a key role
- Quantitative analysis
  - MACs/Parameters not a good metric for performance/energy
  - Need of designing with HW in the loop