

CS 250B: Modern Computer Systems

Hardware Acceleration Case Study Neural Network Accelerators



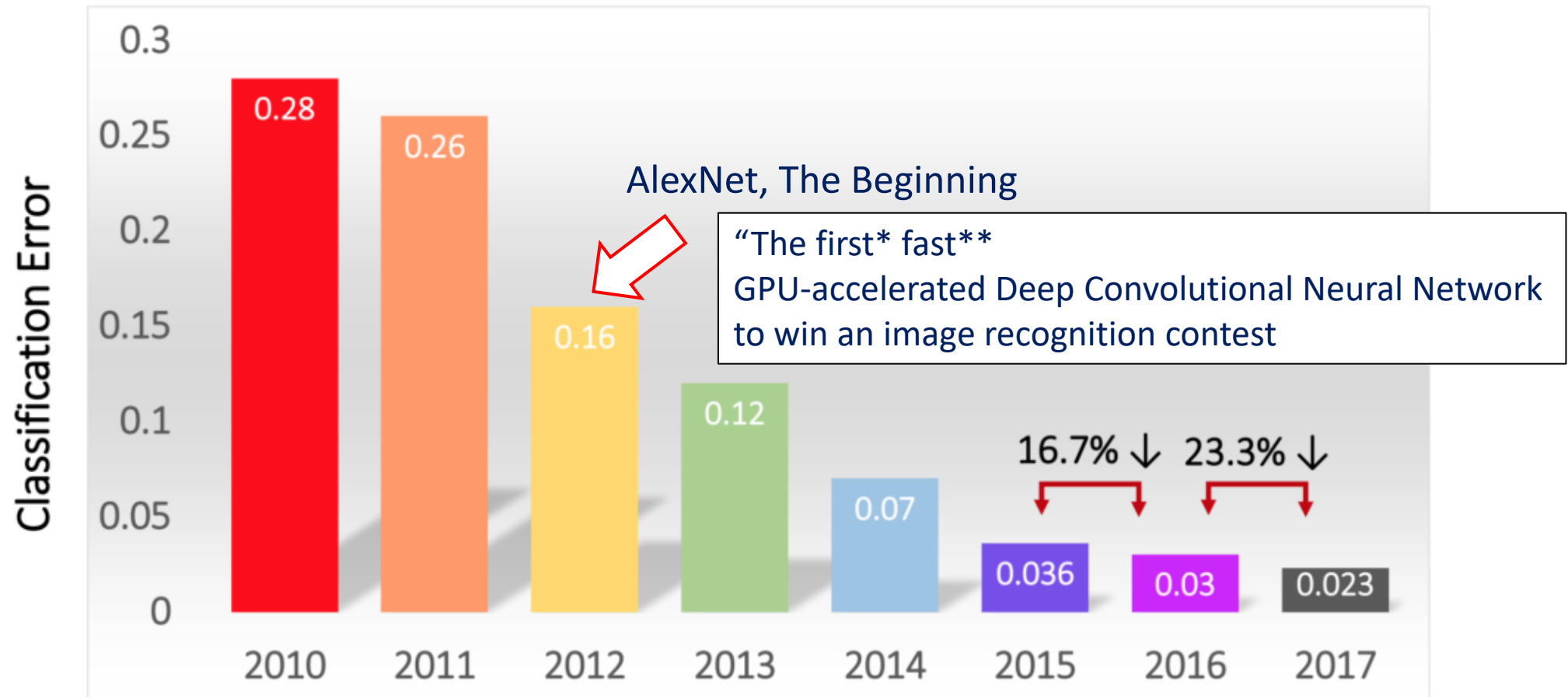
Sang-Woo Jun

Usefulness of Deep Neural Networks

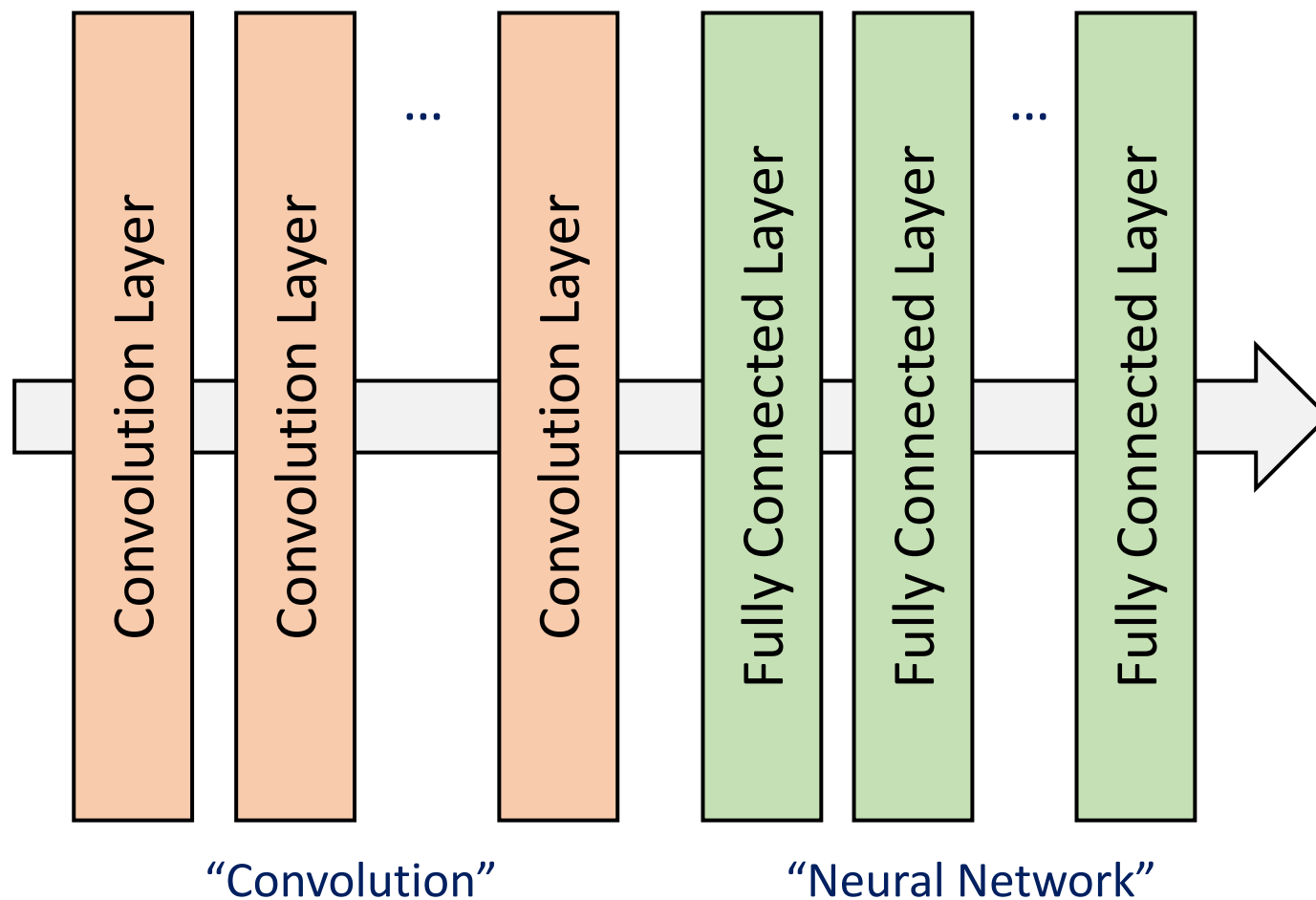
- ❑ No need to further emphasize the obvious

ImageNet Top-5 Classification Accuracy Over the Years

15 million images 1000 classes in the ImageNet challenge



Convolutional Neural Networks Overview



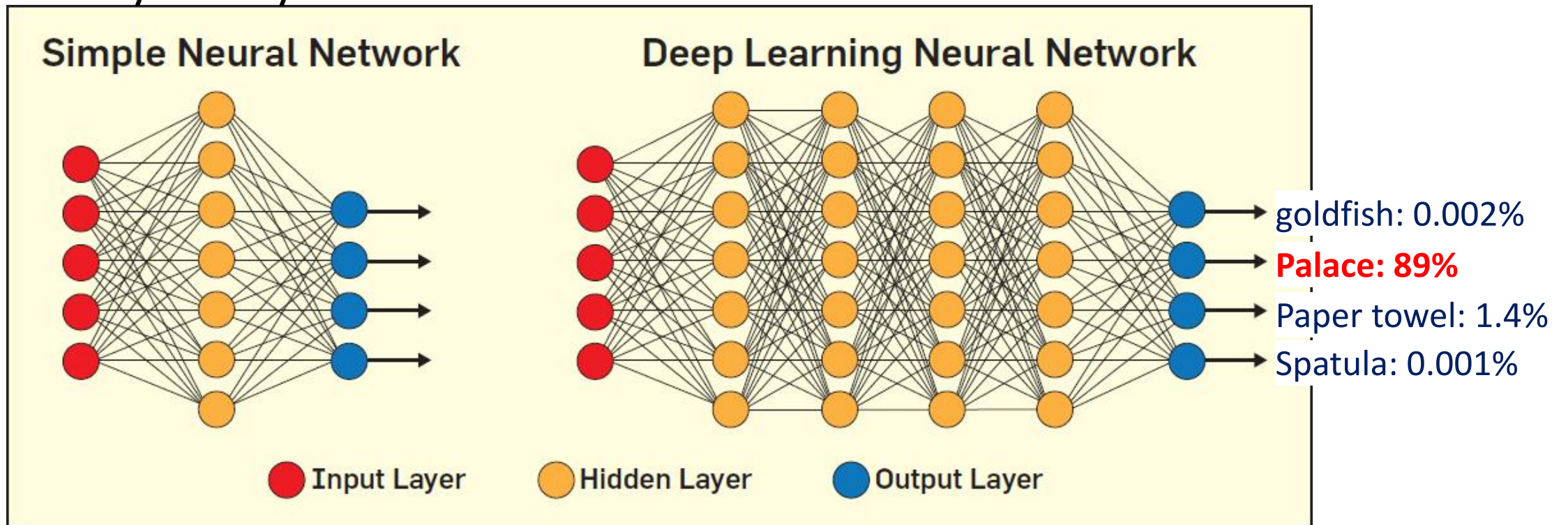
goldfish: 0.002%
shark: 0.08%
magpie: 0.02%
⋮
Palace: 89%
⋮
Paper towel: 1.4%
⋮
Spatula: 0.001%
⋮

Training vs. Inference

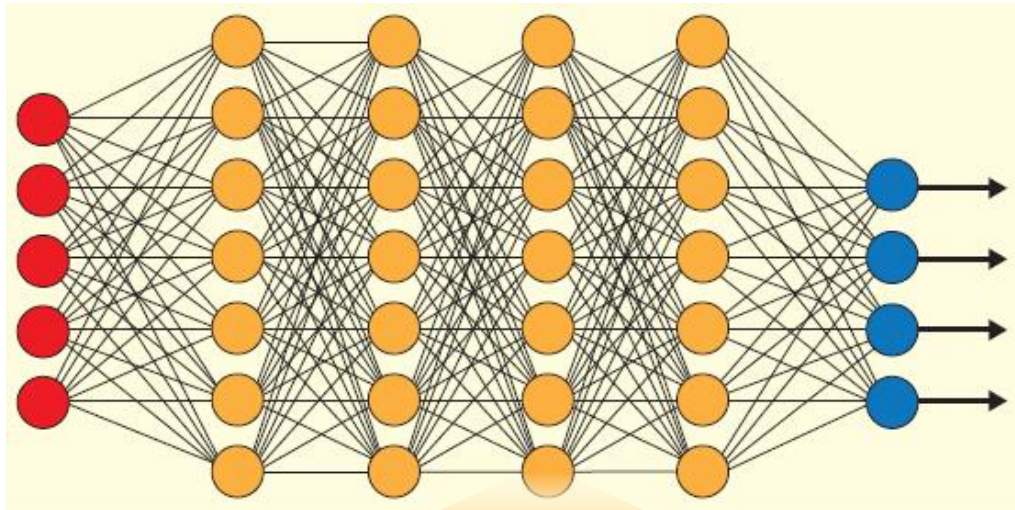
- ❑ Training: Tuning parameters using training data
 - Backpropagation using stochastic gradient descent is the most popular algorithm
 - Training in data centers and distributing trained data is a common model*
 - Because training algorithm changes rapidly, GPU cluster is the most popular hardware (**Low demand for application-specific accelerators**)
- ❑ Inference: Determining class of a new input data
 - Using a trained model, determine class of a new input data
 - Inference usually occurs close to clients
 - Low-latency and power-efficiency is required (**High demand for application specific accelerators**)

Deep Neural Networks (“Fully Connected”*)

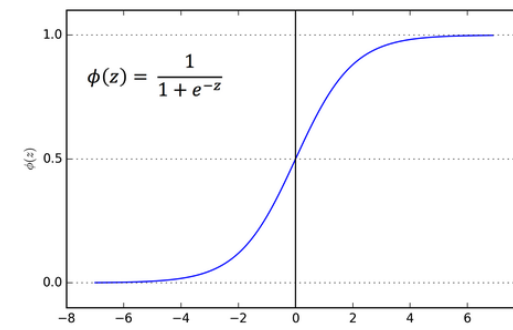
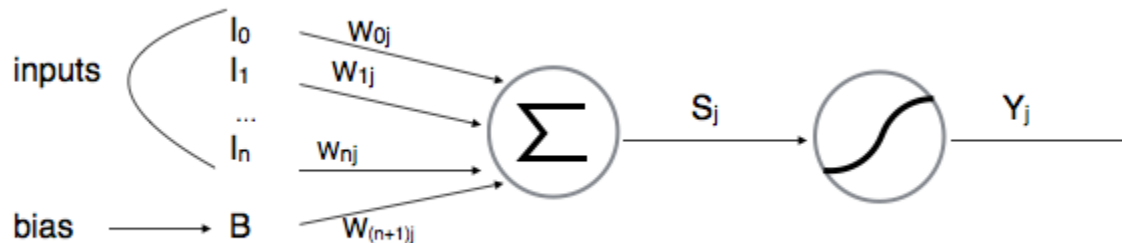
- Each layer may have a different number of neurons



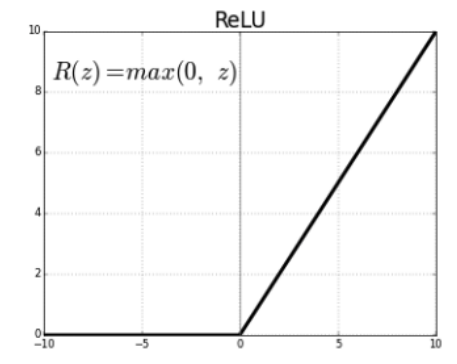
An Artificial Neuron



- ❑ Effectively weight vector multiplied by input vector to obtain a scalar
- ❑ May apply activation function to output
 - Adds non-linearity



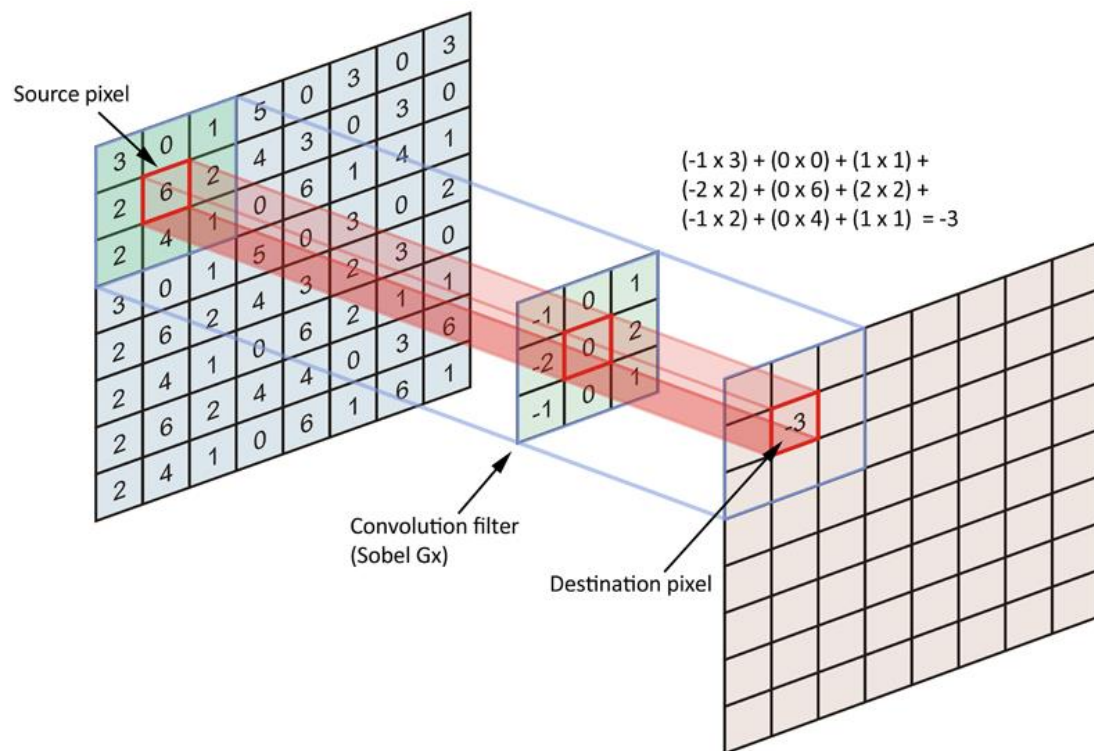
Sigmoid



Rectified Linear Unit (ReLU)

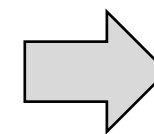
Convolution Layer

Convolution layer



Optional pooling layer

31	7	44	33
65	35	40	46
46	29	32	30
24	49	8	64



65	46
46	64

Convolution Example

Convolution
Filter

1	2	3
-2	0	-1
5	-2	4

×

Input map

0	1	0	1	0	1
2	4	3	1	0	0
5	2	7	2	1	5
4	1	8	4	2	8
5	0	1	5	8	3
0	0	0	5	2	6

=

Output map

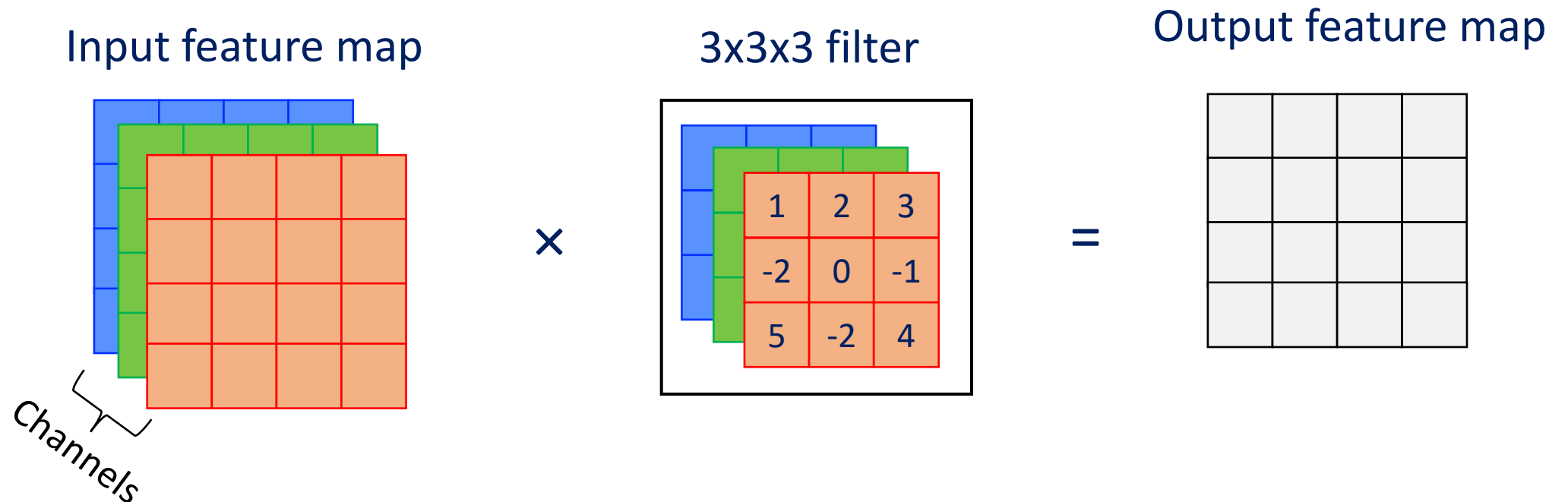
44	-1		

Typically adds zero padding to source matrix to maintain dimensions

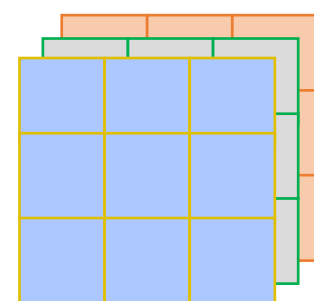
$$\begin{aligned} \text{Channel partial sum}[0][0] &= \\ & 1 \times 0 + 2 \times 1 + 3 \times 0 \\ & + (-2) \times 2 + 0 \times 4 + (-1) \times 3 \\ & + 5 \times 5 + (-2) \times 2 + 4 \times 7 \\ & = 44 \end{aligned}$$

Multidimensional Convolution

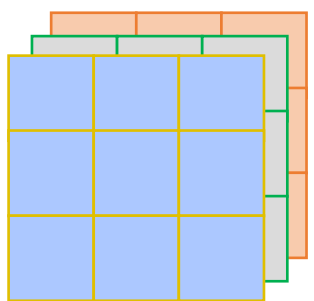
- ❑ “Feature Map” usually has multiple layers
 - An image has R, G, B layers, or “channels”
- ❑ One layer has many convolution filters, which create a multichannel output map



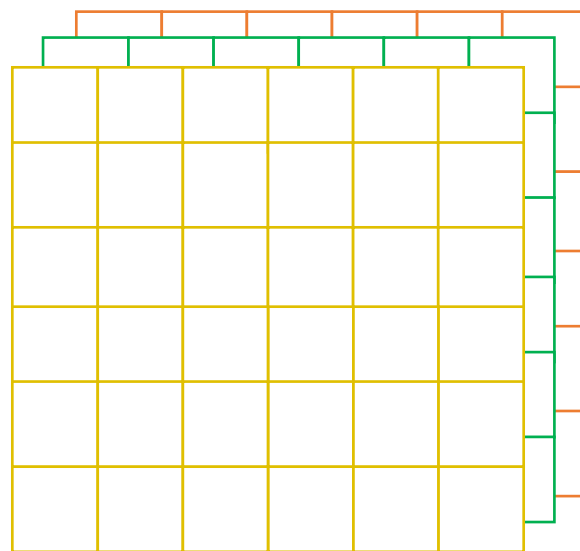
Multiple Convolutions



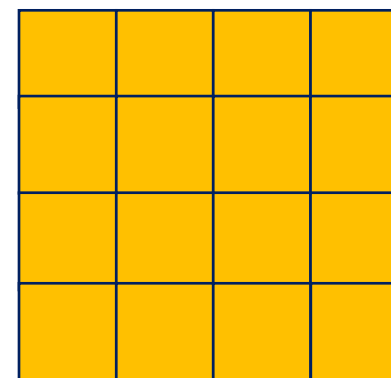
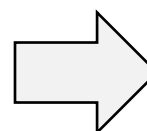
Filter 0



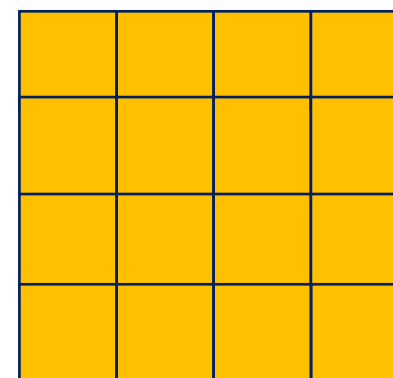
Filter 1



Input feature map



Output feature map 0

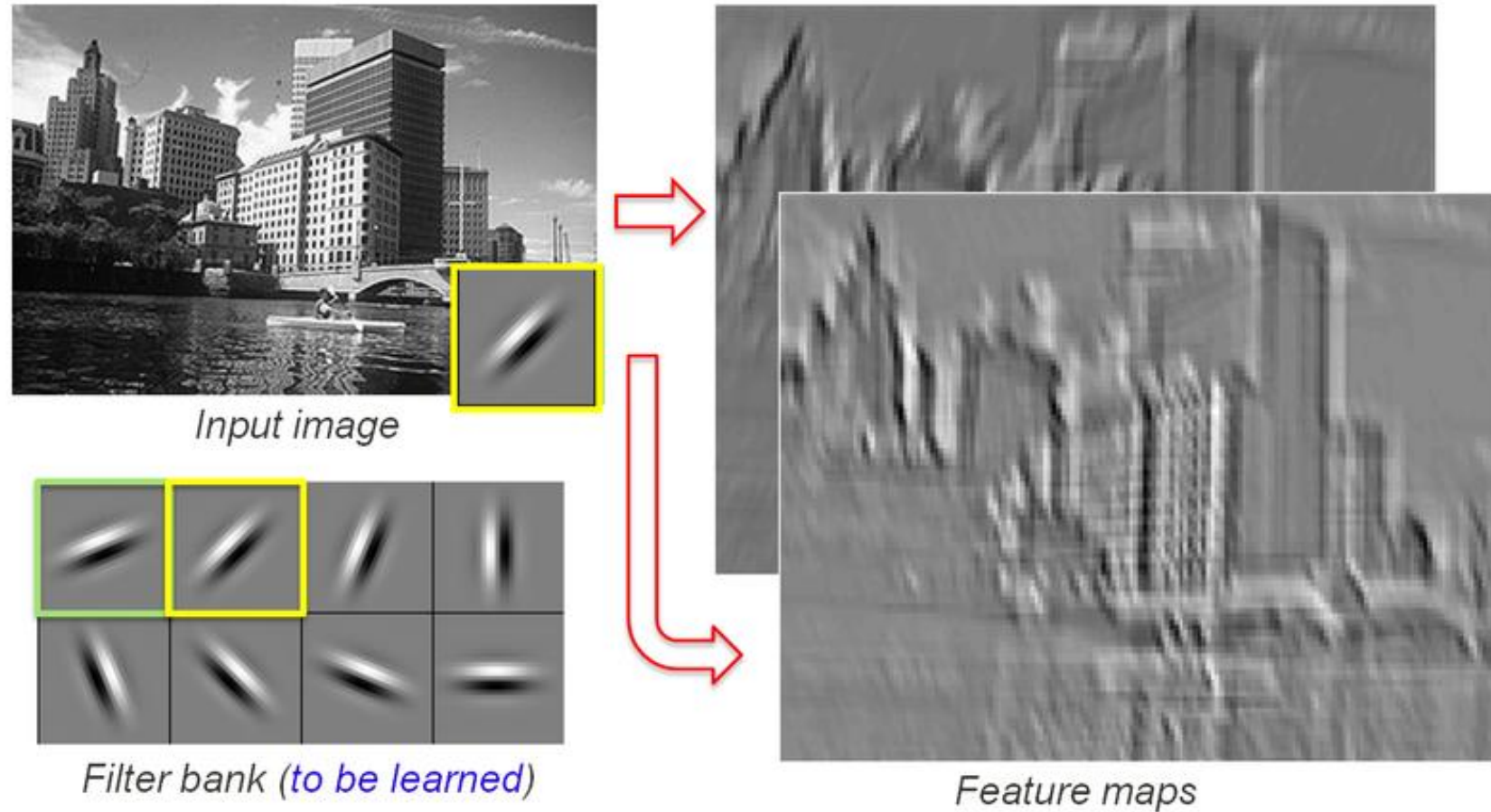


Output feature map 1

Example Learned Convolution Filters



Multidimensional Convolution



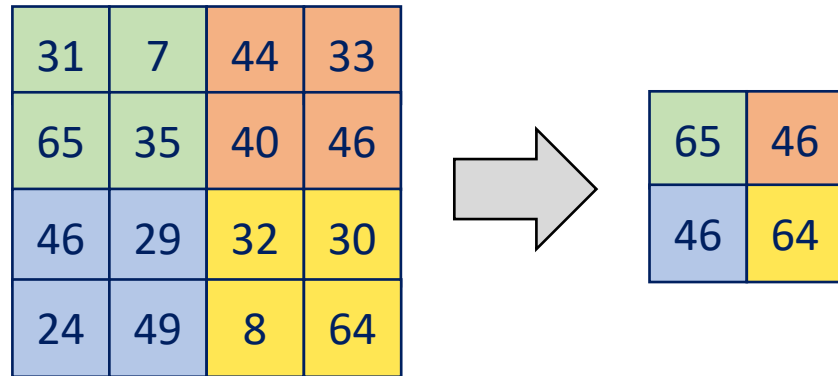
Computation in the Convolution Layer

```
for(n=0; n<N; n++) { // Input feature maps (IFMaps)
  for(m=0; m<M; m++) { // Weight Filters
    for(c=0; c<C; c++) { // IFMap/Weight Channels
      for(y=0; y<H; y++) { // Input feature map row
        for(x=0; x<H; x++) { // Input feature map column
          for(j=0; j<R; j++) { // Weight filter row
            for(i=0; i<R; i++) { // Weight filter column
              O[n][m][x][y] += W[m][c][i][j] * I[n][c][y+j][x+i]}}}}}}}
```

Pooling Layer

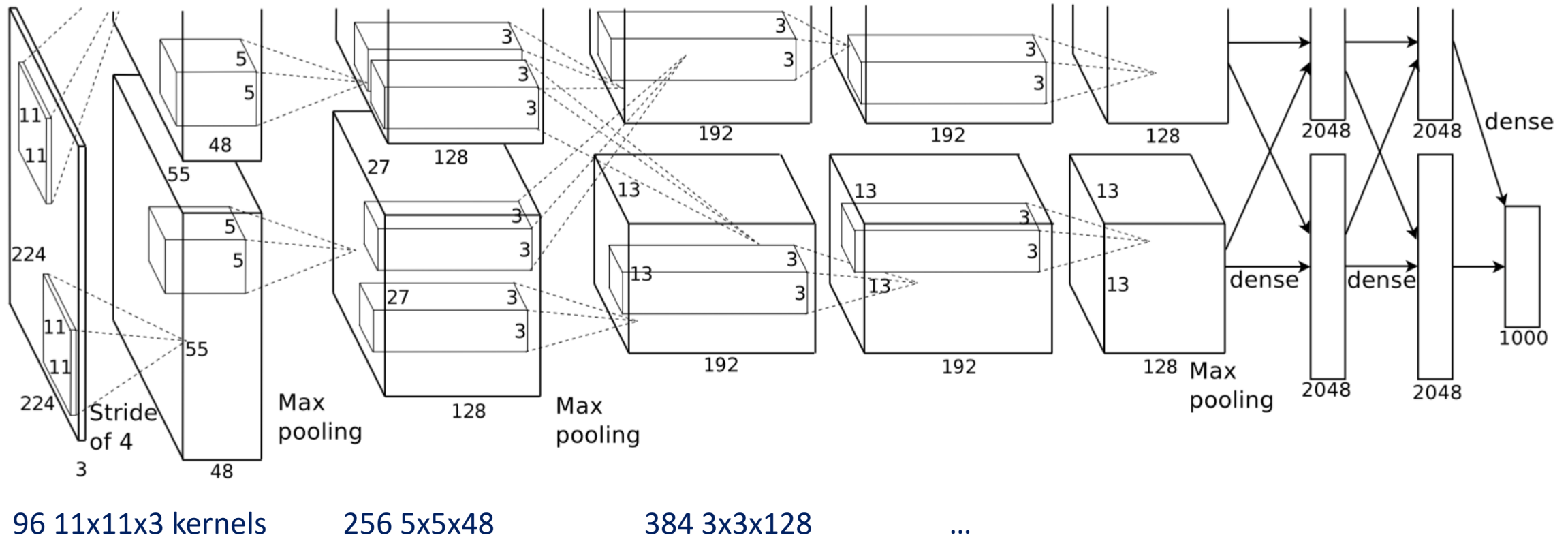
- ❑ Reduces size of the feature map
 - Max pooling, Average pooling, ...

Max pooling example



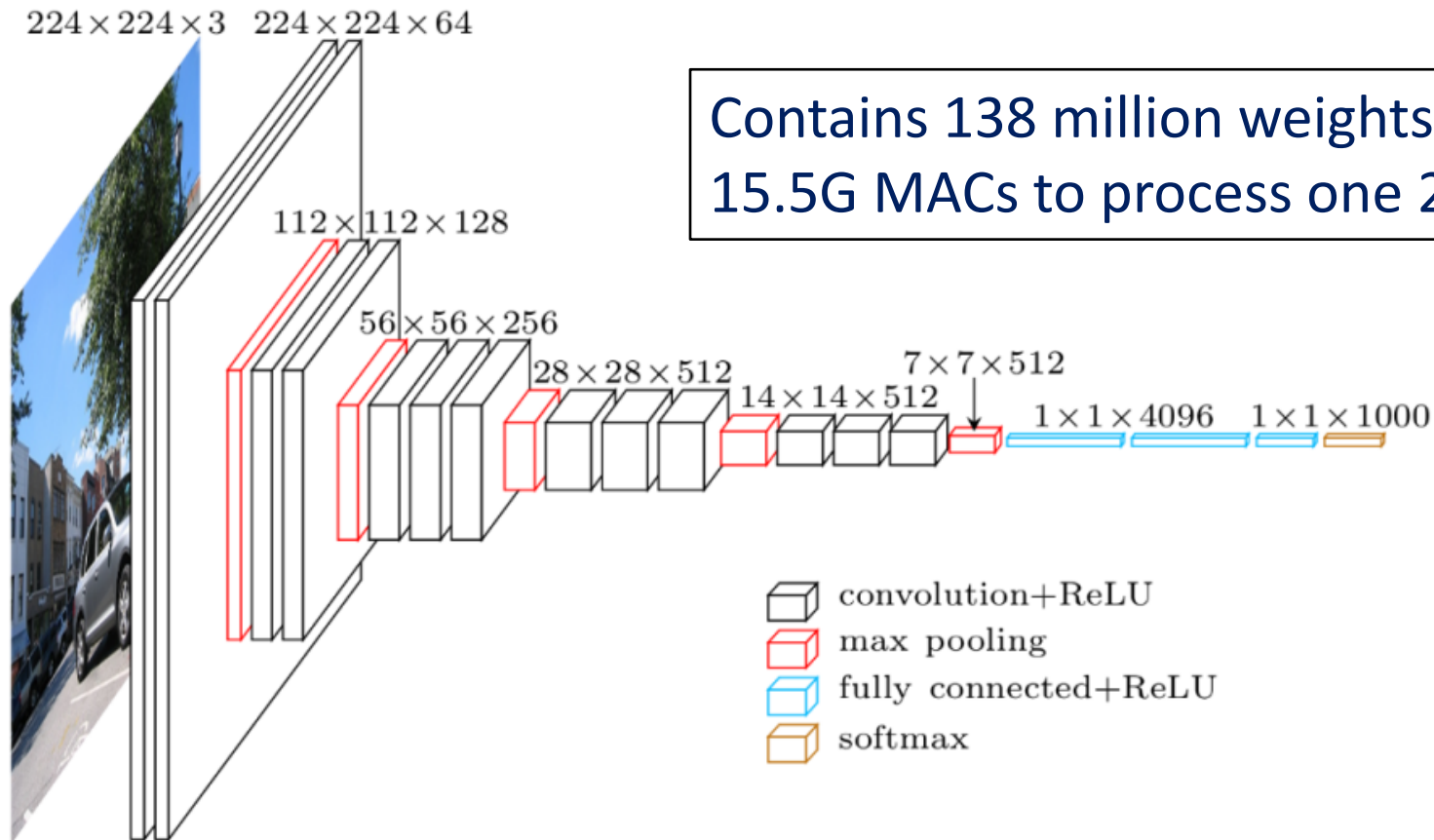
Real Convolutional Neural Network

-- AlexNet



Simplified intuition: Higher order information at later layer

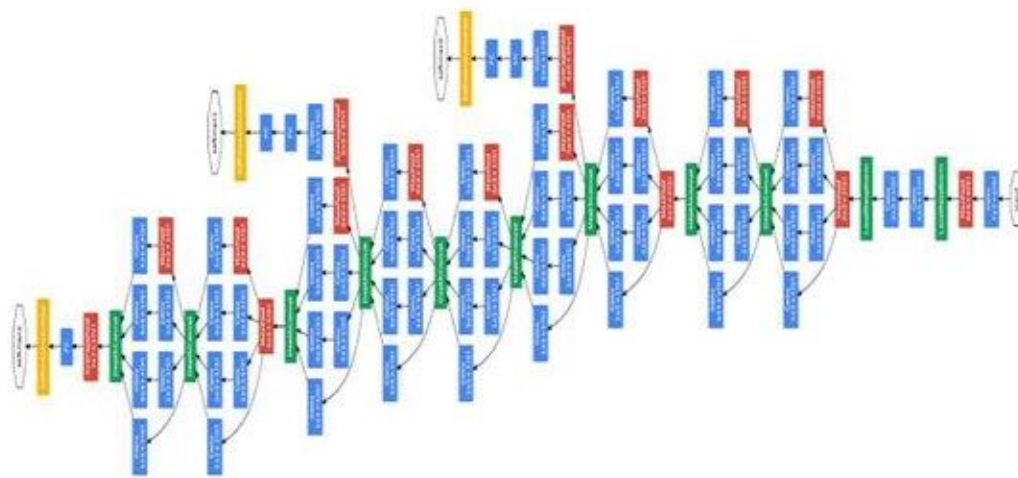
Real Convolutional Neural Network -- VGG 16



Contains 138 million weights and 15.5G MACs to process one 224 x 224 input image

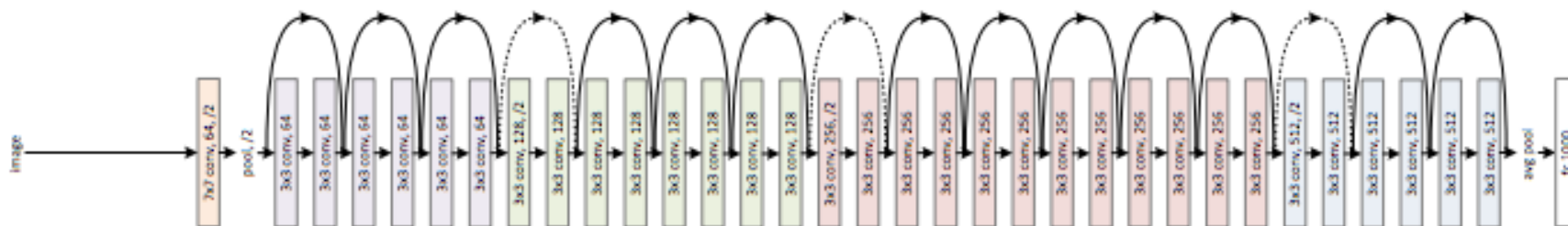
There are Many, Many Neural Networks

- GoogLeNet, ResNet, YOLO, ...
 - Share common building blocks, but look drastically different



GoogLeNet (ImageNet 2014 winner)

34-layer residual



ResNet
(ImageNet 2015 winner)

Beware/Disclaimer on Accelerators

- ❑ This field is advancing very quickly/messy right now
- ❑ Lots of papers/implementations always beating each other, with seemingly contradicting results
 - Eyes wide open!

The Need For Neural Network Accelerators

- ❑ Remember: “VGG-16 requires 138 million weights and 15.5G MACs to process one 224×224 input image”
 - CPU at 3 GHz, 1 IPC, (3 Giga Operations Per Second – GOPS): 5+ seconds per image
 - Also significant power consumption!
 - (Optimistically assuming 3 GOPS/thread at 8 threads using 100 W, 0.24 GOPS/W)

	CPU (Intel DuoCore, 2.7GHz)	GPU (GTX480)	mGPU (GT335m)	NeuFlow on Xilinx Virtex 6	NeuFlow on IBM 45 nm process
Real GOPs	1.1	294	54	147	1164
Power (W)	30	220	30	10	5
GOPs/W	0.04	1.34	1.8	14.7	230

* Old data (2011), and performance varies greatly by implementation, some reporting 3+ GOPS/thread on an i7
Trend is still mostly true!

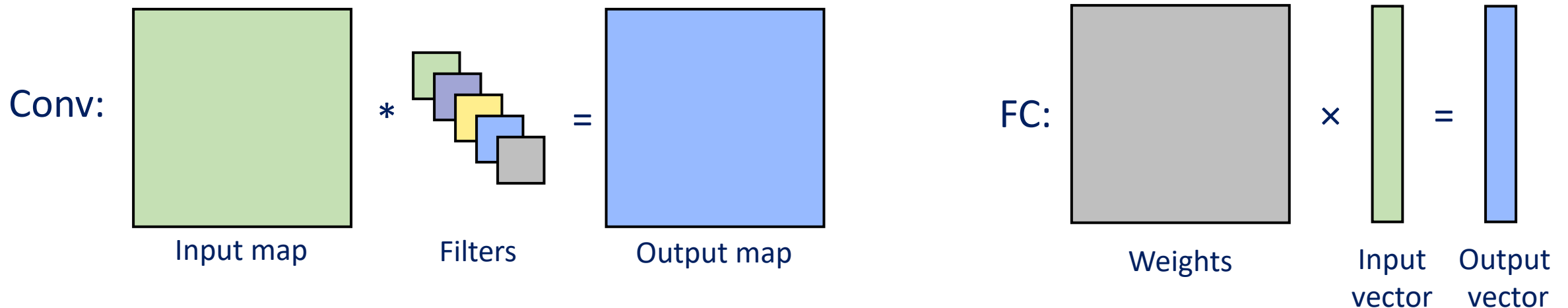
Two Major Layers

□ Convolution Layer

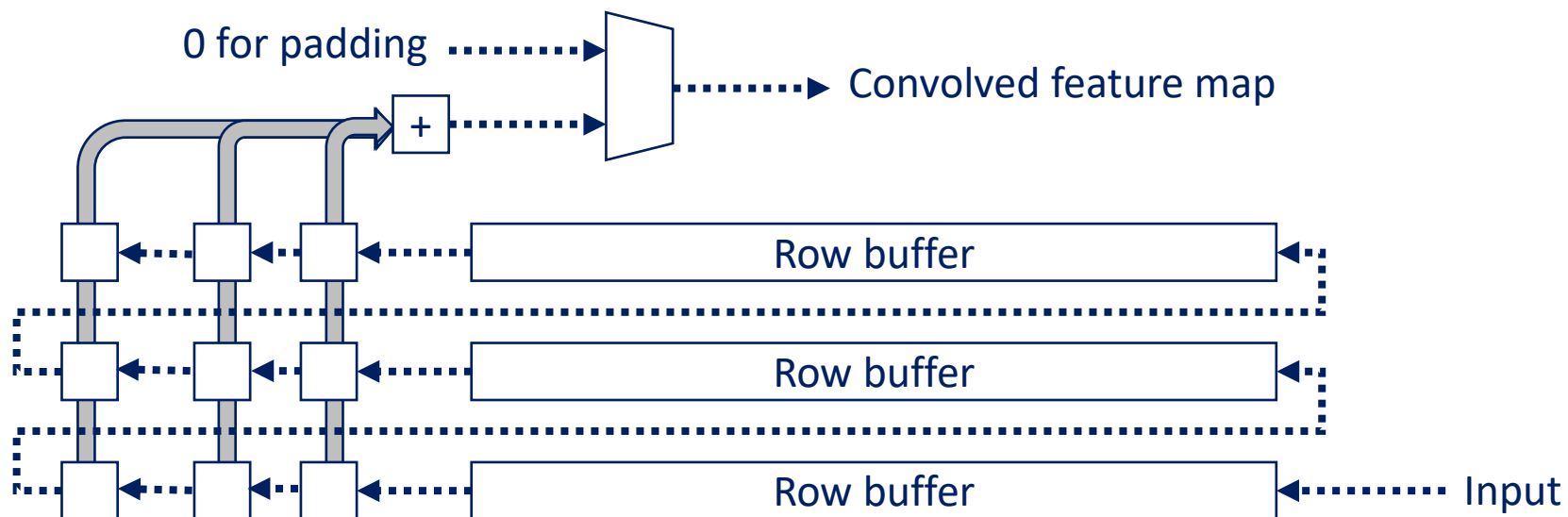
- Many small (1x1, 3x3, 11x11, ...) filters
 - Small number of weights per filter, relatively small number in total vs. FC
- Over 90% of the MAC operations in a typical model

□ Fully-Connected Layer

- N-to-N connection between all neurons, large number of weights



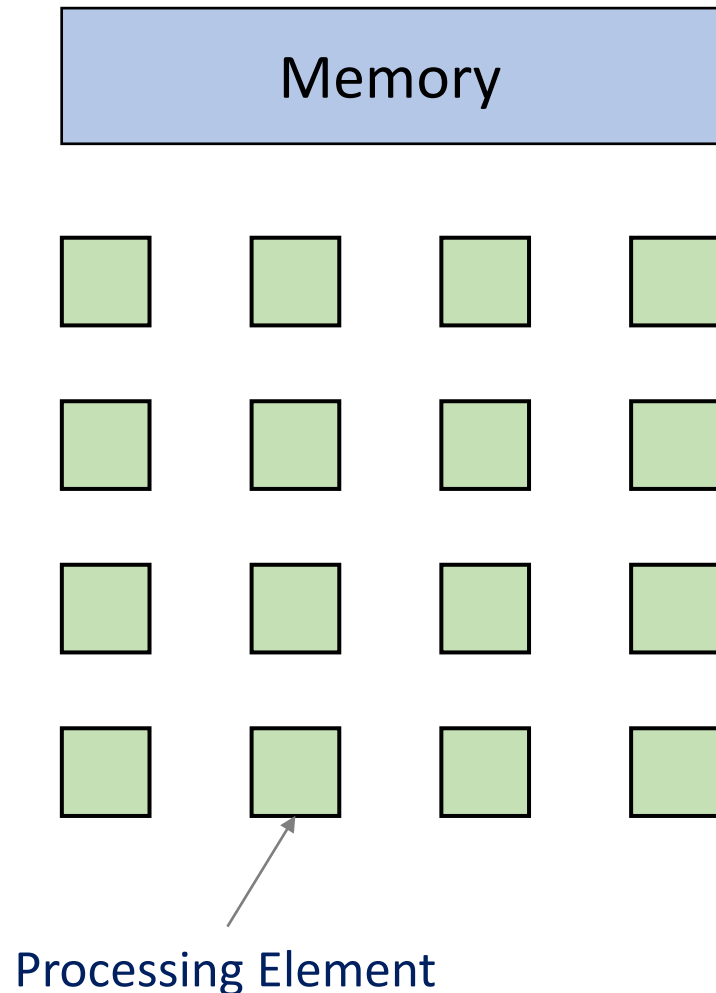
Systolic Array Design for Convolutions



Very efficient design!

BUT BRITTLE! Above design only works for 3x3 conv
and not for FC (Resource fragmentation!)

Spatial Mapping of General-Purpose Compute Units



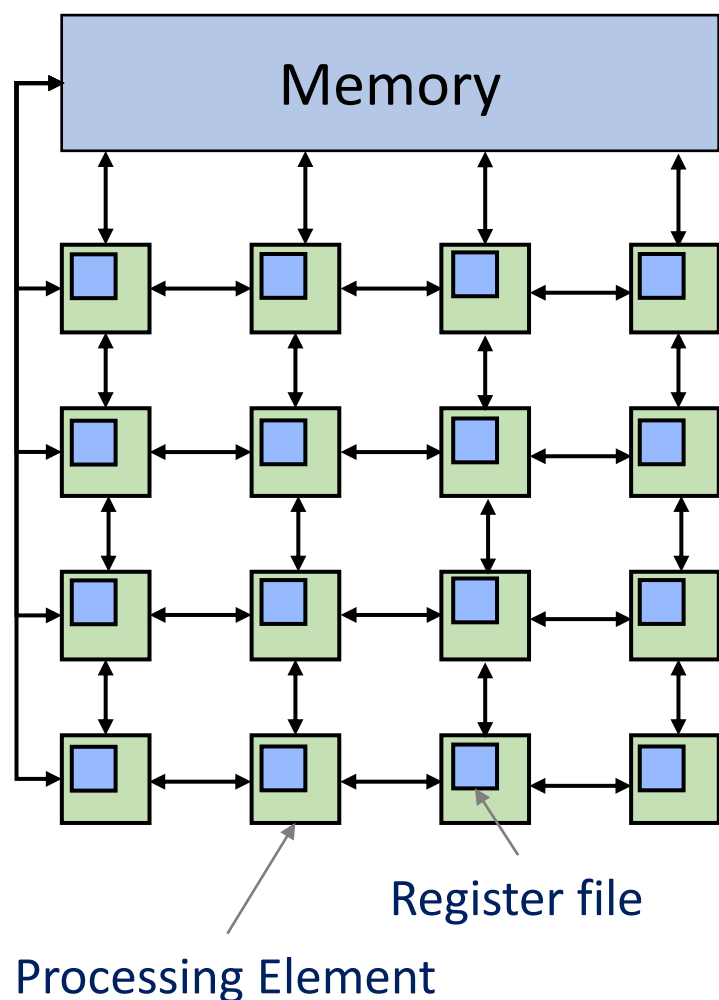
- ❑ Map both convolutions and FC to matrix multiplications
- ❑ Typically a 2D matrix of Processing Elements
 - Each PE is a simple multiply-accumulator
 - Extremely large number of PEs
 - Very high peak throughput!
- ❑ Is memory the bottleneck (Again)?

Memory Access is (Typically) the Bottleneck (Again)

- ❑ 100 GOPS requires over 300 Billion weight/activation accesses
 - Assuming 4 byte floats, 1.2 TB/s of memory accesses
- ❑ AlexNet requires 724 Million MACs to process a 227 x 227 image, over 2 Billion weight/activation accesses
 - Assuming 4 byte floats, that is over 8 GB of weight accesses per image
 - 240 GB/s to hit 30 frames per second
- ❑ An interesting question:
 - Can CPUs achieve this kind of performance?
 - With SIMD and good caching, YES!, but not at low power

“About 35% of cycles are spent waiting for weights to load from memory into the matrix unit ...” – Jouppi et. al., Google TPU

Spatial Mapping of Compute Units 2



- ❑ Optimization 1: On-chip network moves data (weights/activations/output) between PEs and memory for reuse
- ❑ Optimization 2: Small, local memory on each PE
 - Typically using a Register File, a special type of memory with zero-cycle latency, but at high spatial overhead
- ❑ Cache invalidation/work assignment... how?
 - Computation is very regular and predictable

A class of accelerators deal only with problems that fit entirely in on-chip memory. This distinction is important.

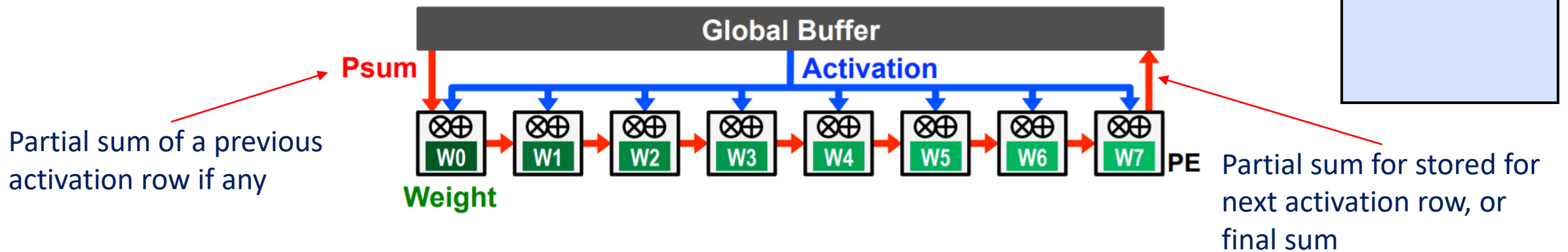
Different Strategies of Data Reuse

- ❑ Weight Stationary
 - Try to maximize local weight reuse
- ❑ Output Stationary
 - Try to maximize local partial sum reuse
- ❑ Row Stationary
 - Try to maximize inter-PE data reuse of all kinds
- ❑ No Local Reuse
 - Single/few global on-chip buffer, no per-PE register file and its space/power overhead

Weight Stationary

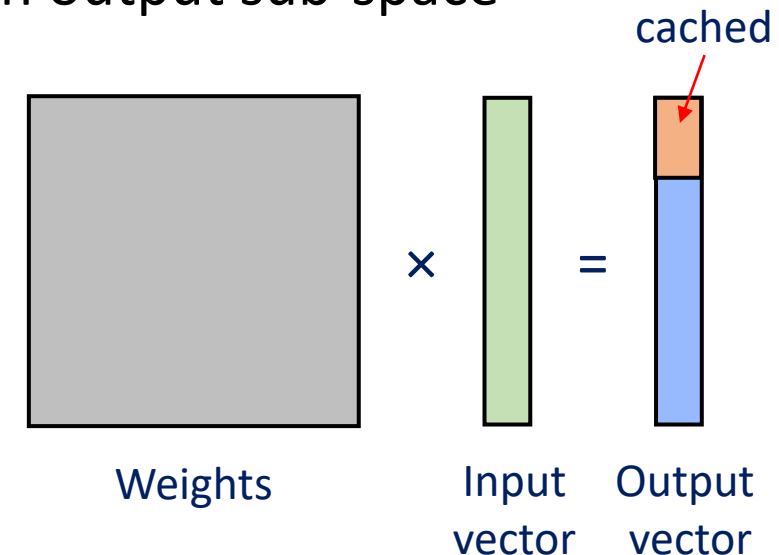
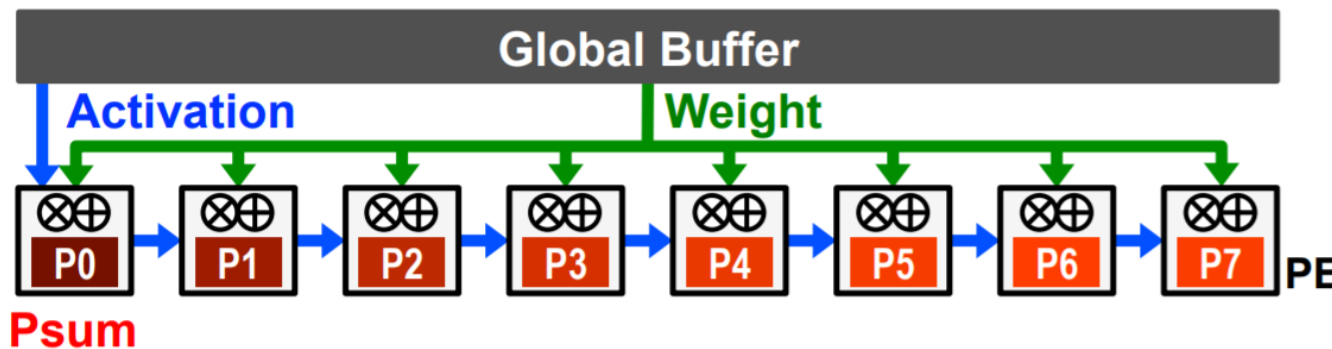
- ❑ Keep weights cached in PE register files
 - Effective for convolution especially if all weights can fit in PEs
- ❑ Each activation is broadcast to all PEs, and computed partial sum is forwarded to other PEs to complete computation
 - Intuition: Each PE is working on an adjacent position of an input row

Weight stationary convolution for a row in the convolution



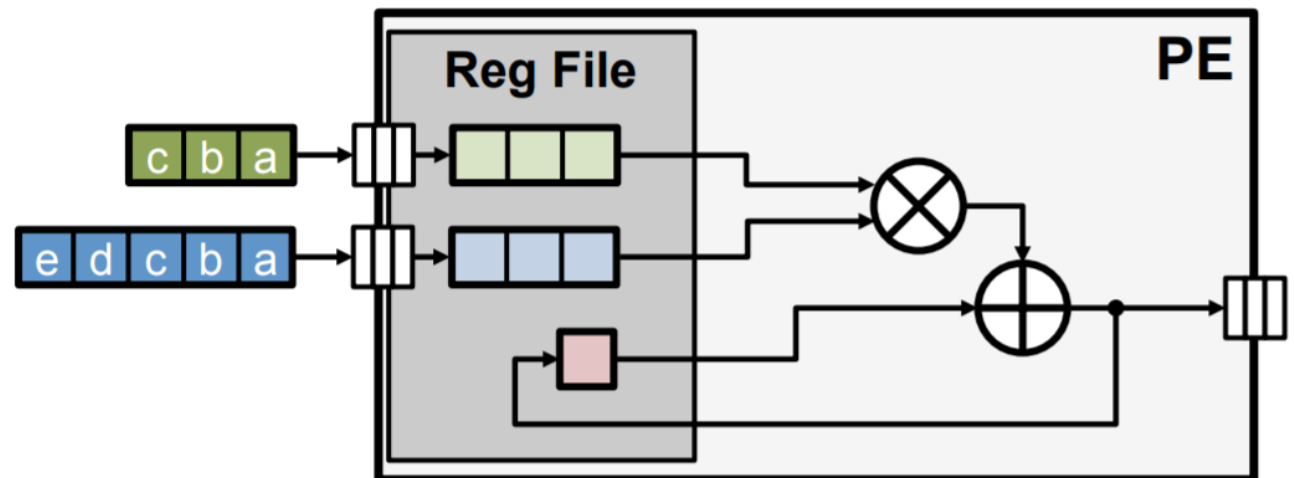
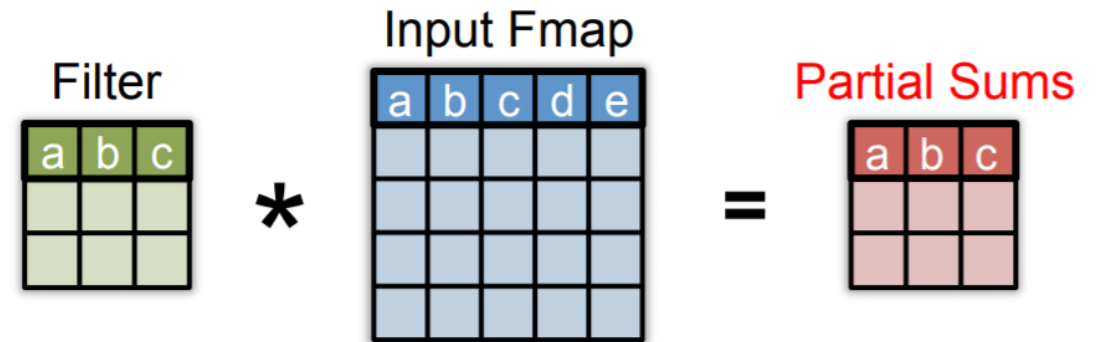
Output Stationary

- ❑ Keep partial sums cached on PEs – Work on subset of output at a time
 - Effective for FC layers, where each output depend on many input/weights
 - Also for convolution layers when it has too many layers
- ❑ Each weight is broadcast to all PEs, and input relayed to neighboring PEs
 - Intuition: Each PE is working on an adjacent position in an output sub-space



Row Stationary

- ❑ Keep as much related to the same filter row cached... Across PEs
 - Filter weights, input, output...
- ❑ Not much reuse in a PE
 - Weight stationary if filter row fits in register file

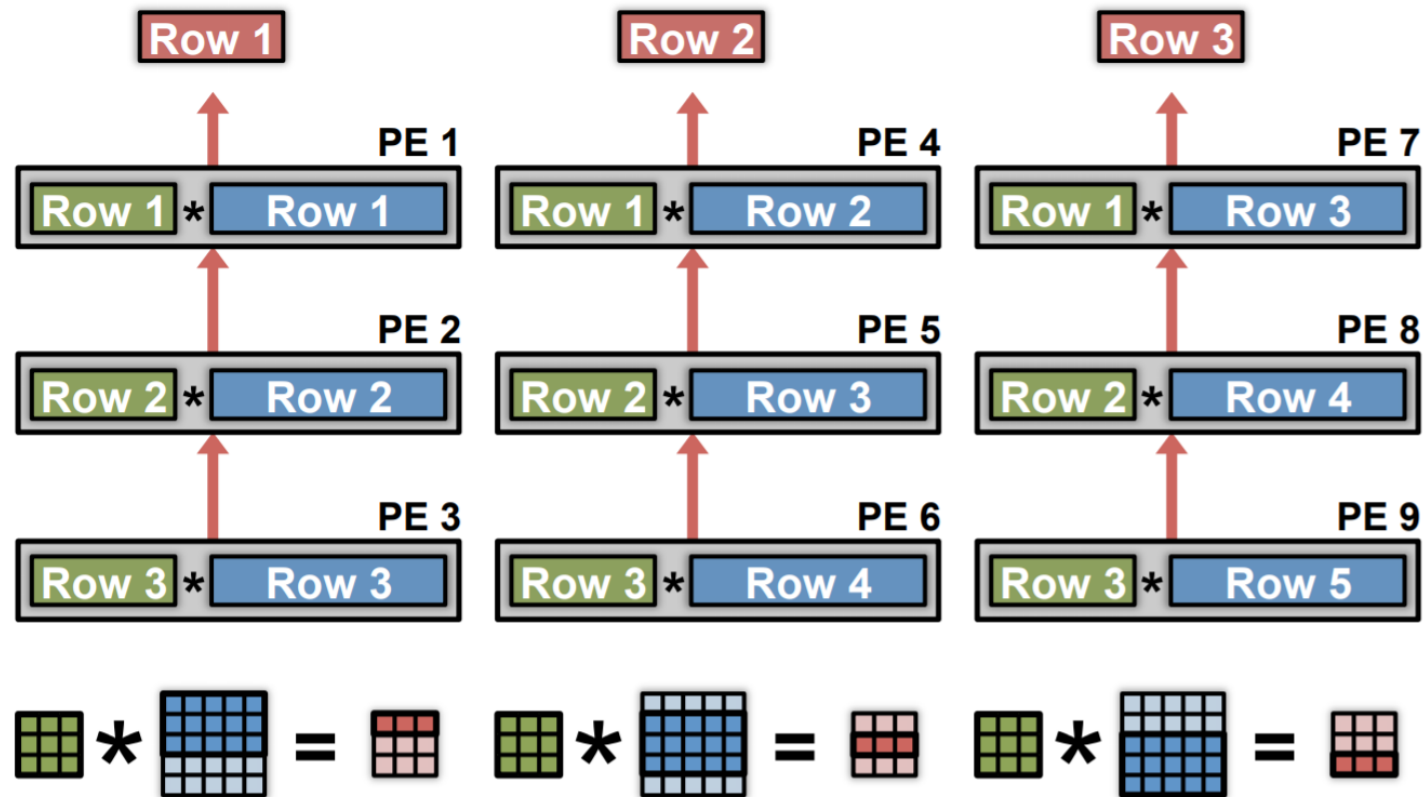


Row Stationary

- Lots of reuse across different PEs

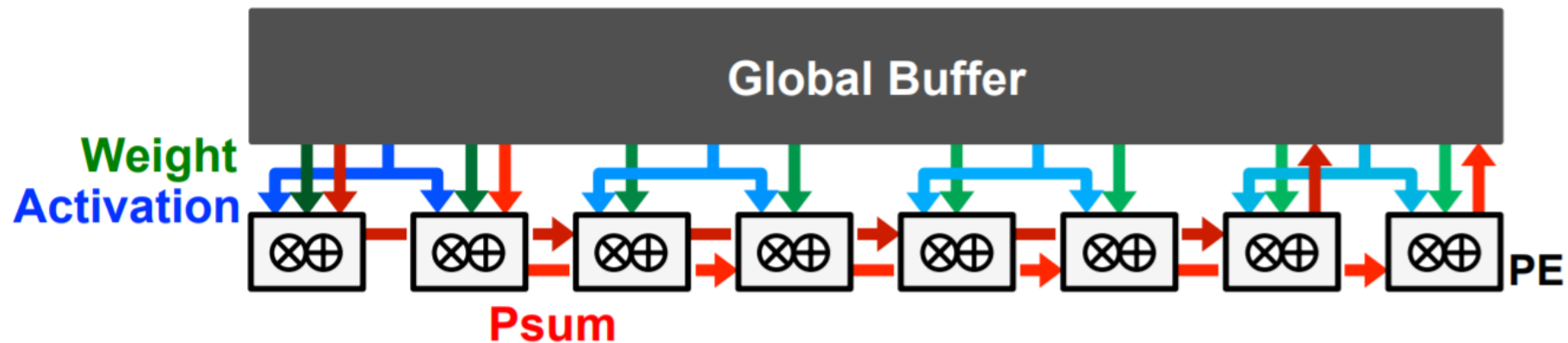
- Filter row reused horizontally
- Input row reused diagonally
- Partial sum reused vertically

- Even further reuse by interleaving multiple input channels and multiple filters

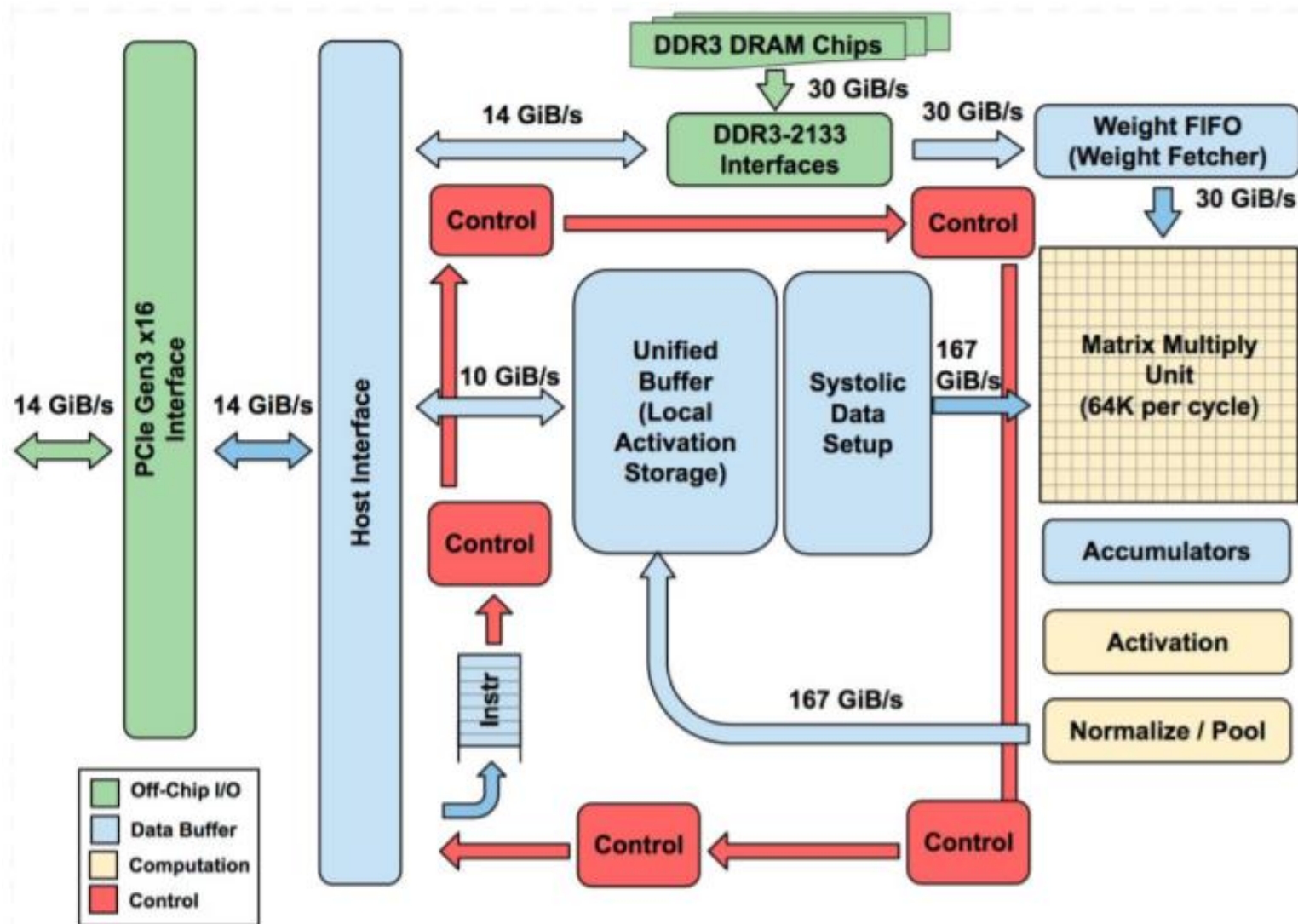


No Local Reuse

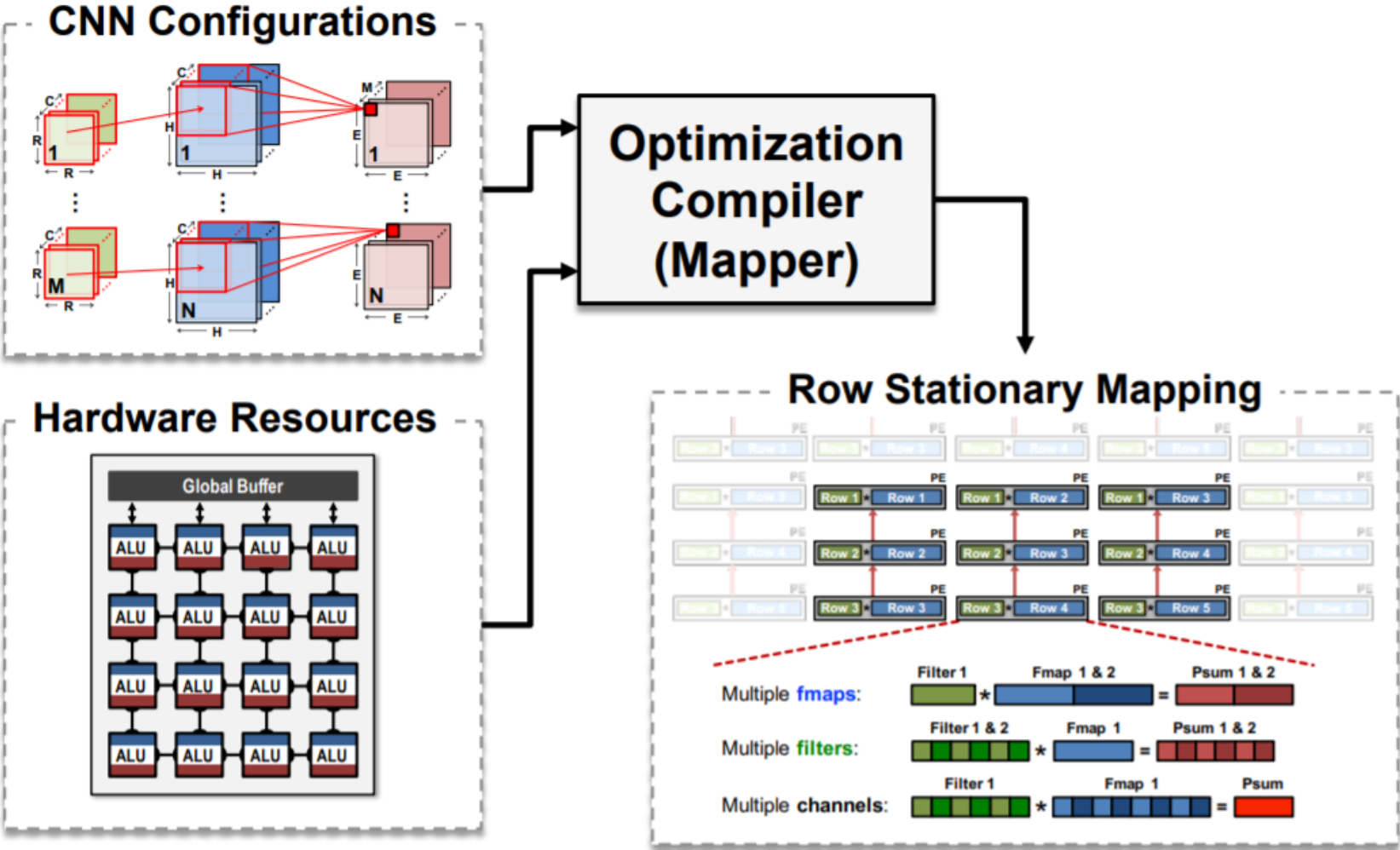
- ❑ While in-PE register files are fast and power-efficient, they are not space efficient
- ❑ Instead of distributed register files, use the space to build a much larger global buffer, and read/write everything from there



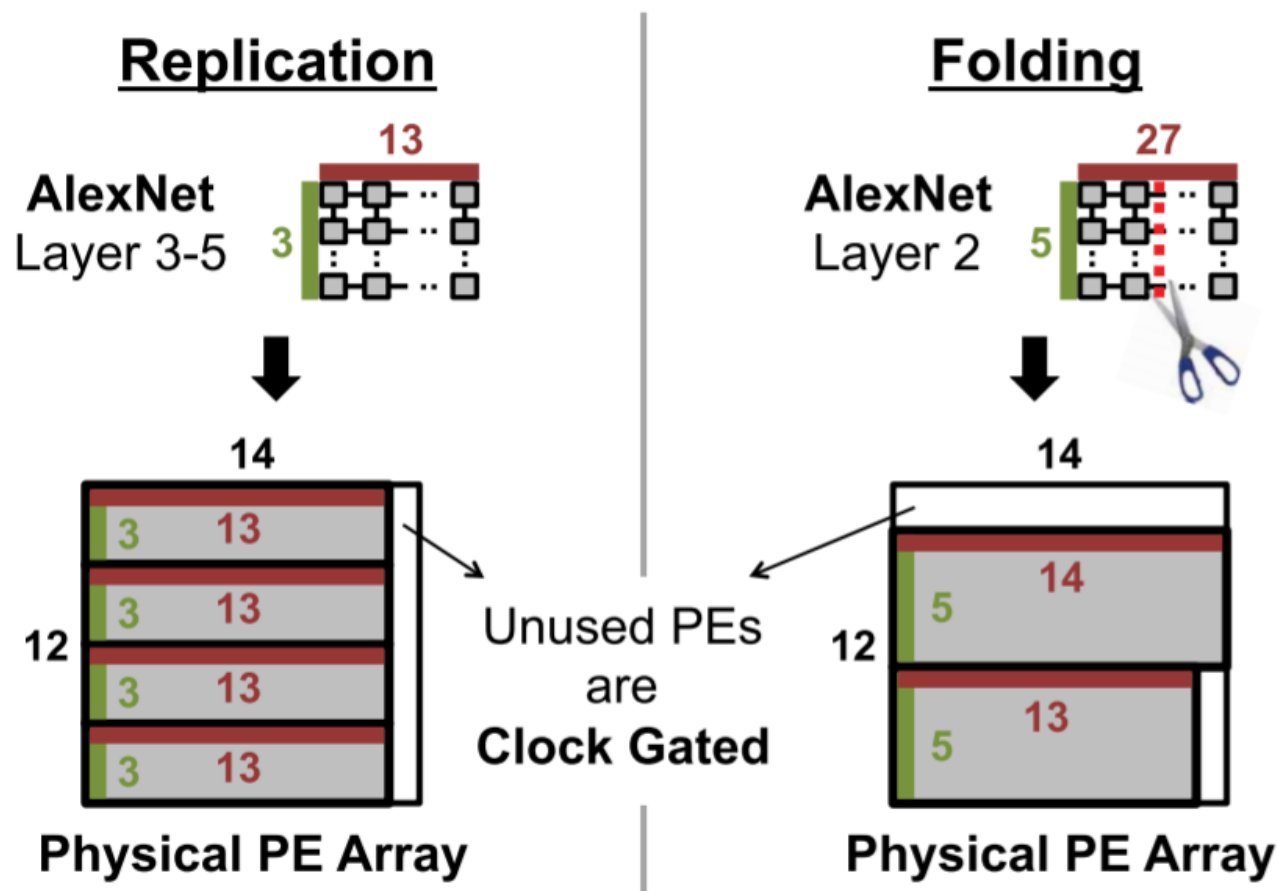
Google TPU Architecture (v1 for simplicity)



Static Resource Mapping

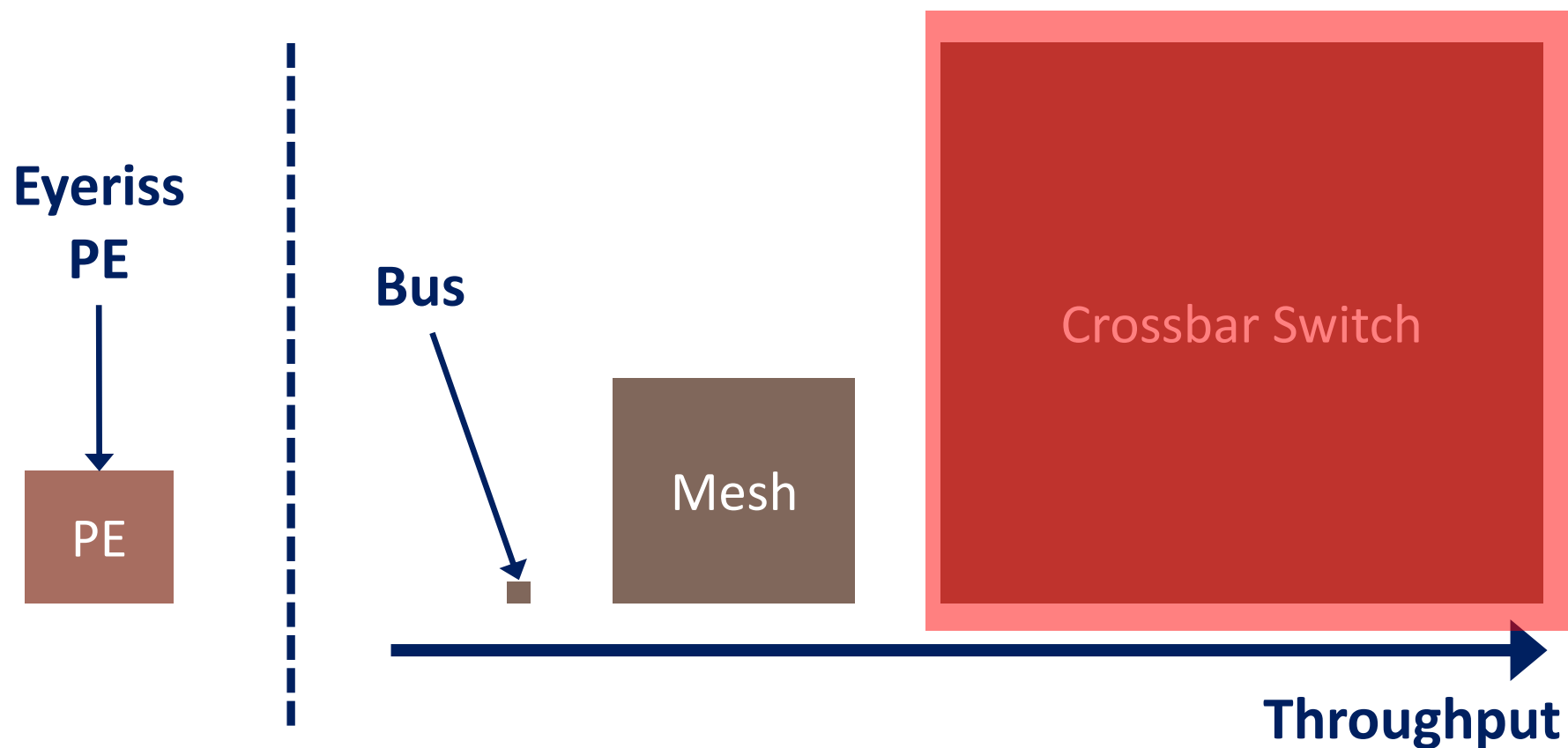


Map And Fold For Efficient Use of Hardware



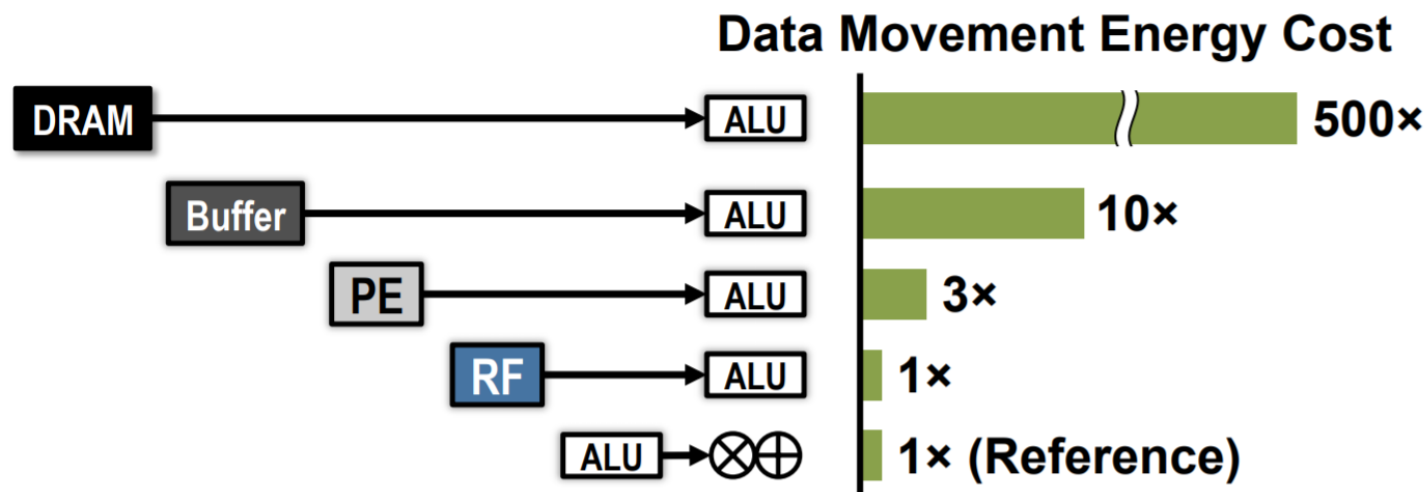
Requires a flexible on-chip network

Overhead of Network-on-Chip Architectures



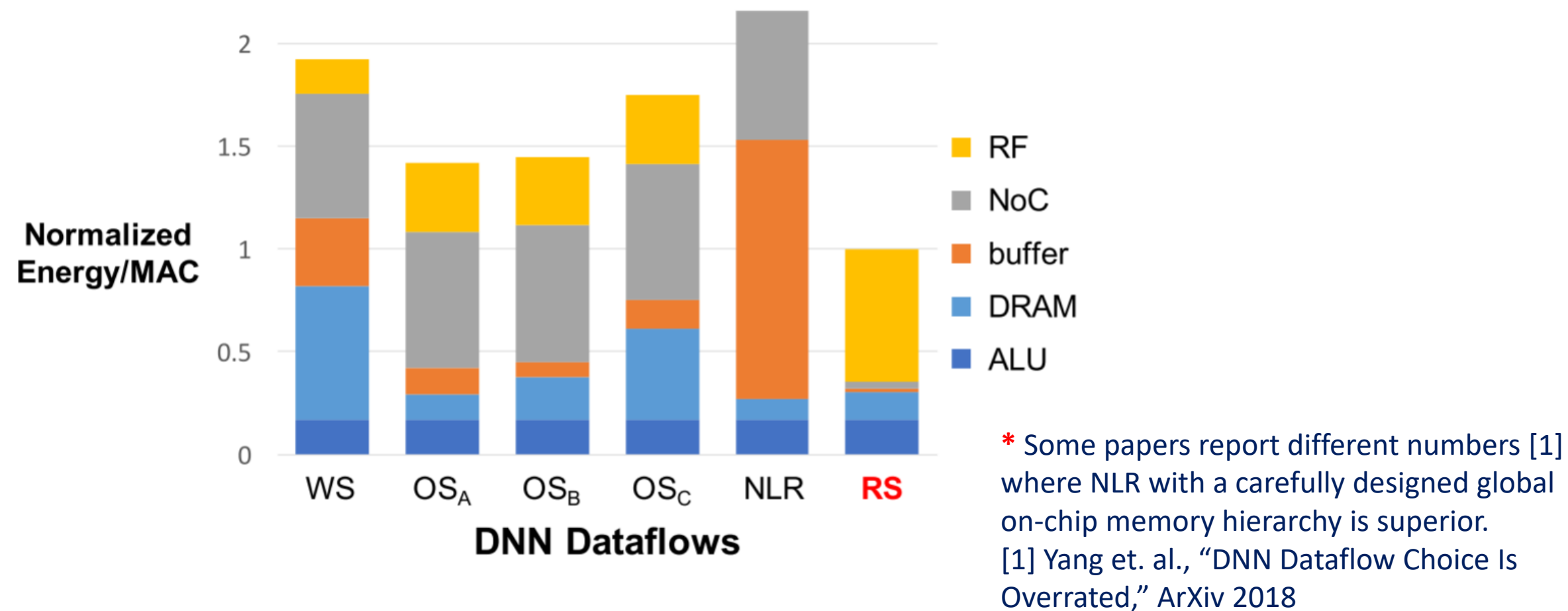
Power Efficiency Comparisons

- Any of the presented architectures reduce memory pressure enough that memory access is no longer the dominant bottleneck
 - Now what's important is the power efficiency

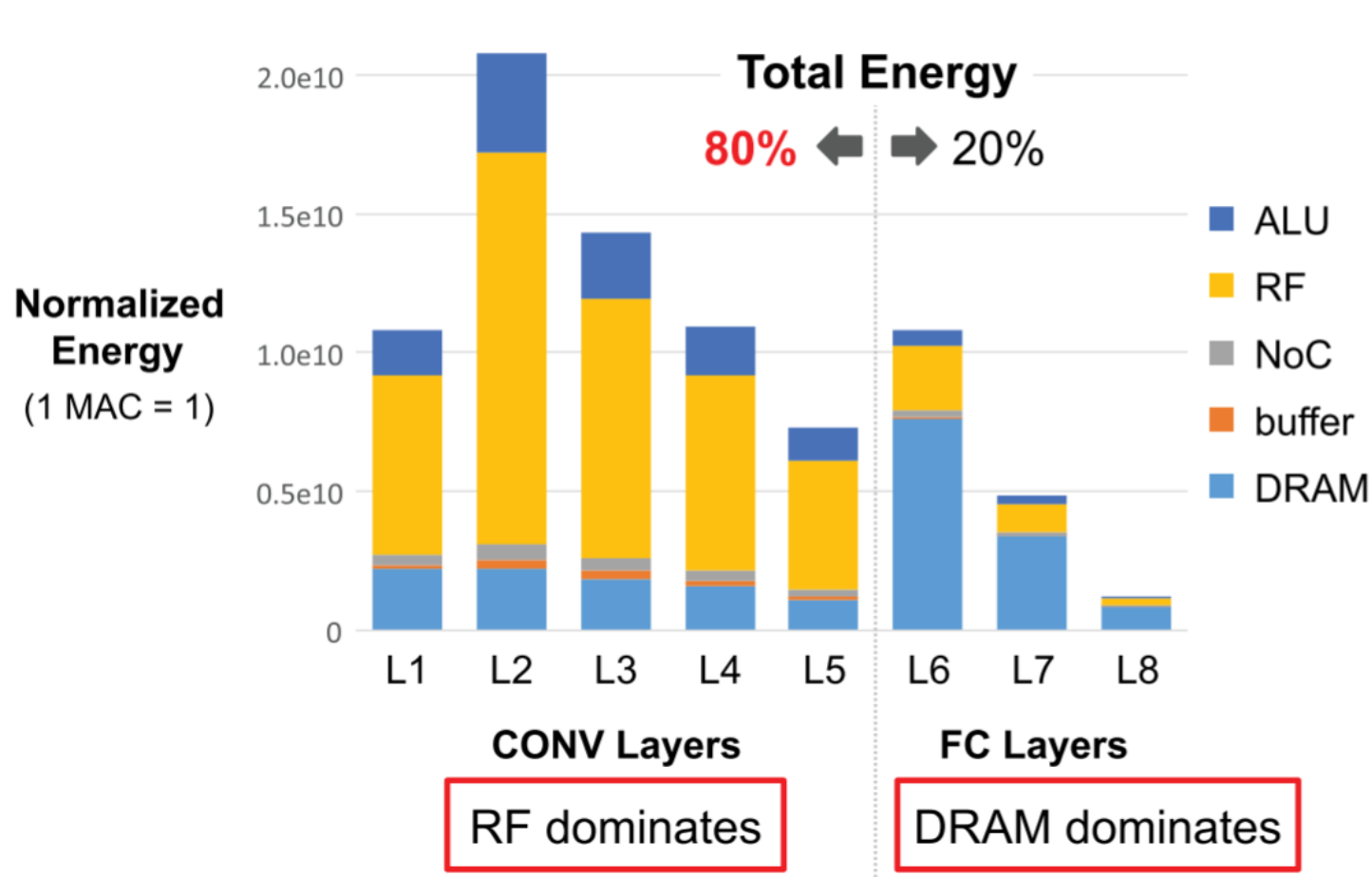


Goal becomes to reduce as much DRAM access as possible!

Power Efficiency Comparisons



Power Consumption Comparison Between Convolution and FC Layers



□ Data reuse in FC is inherently low

- Unless we have enough on-chip buffers to keep all weights, systems methods are not going to be enough

Next: Model Compression