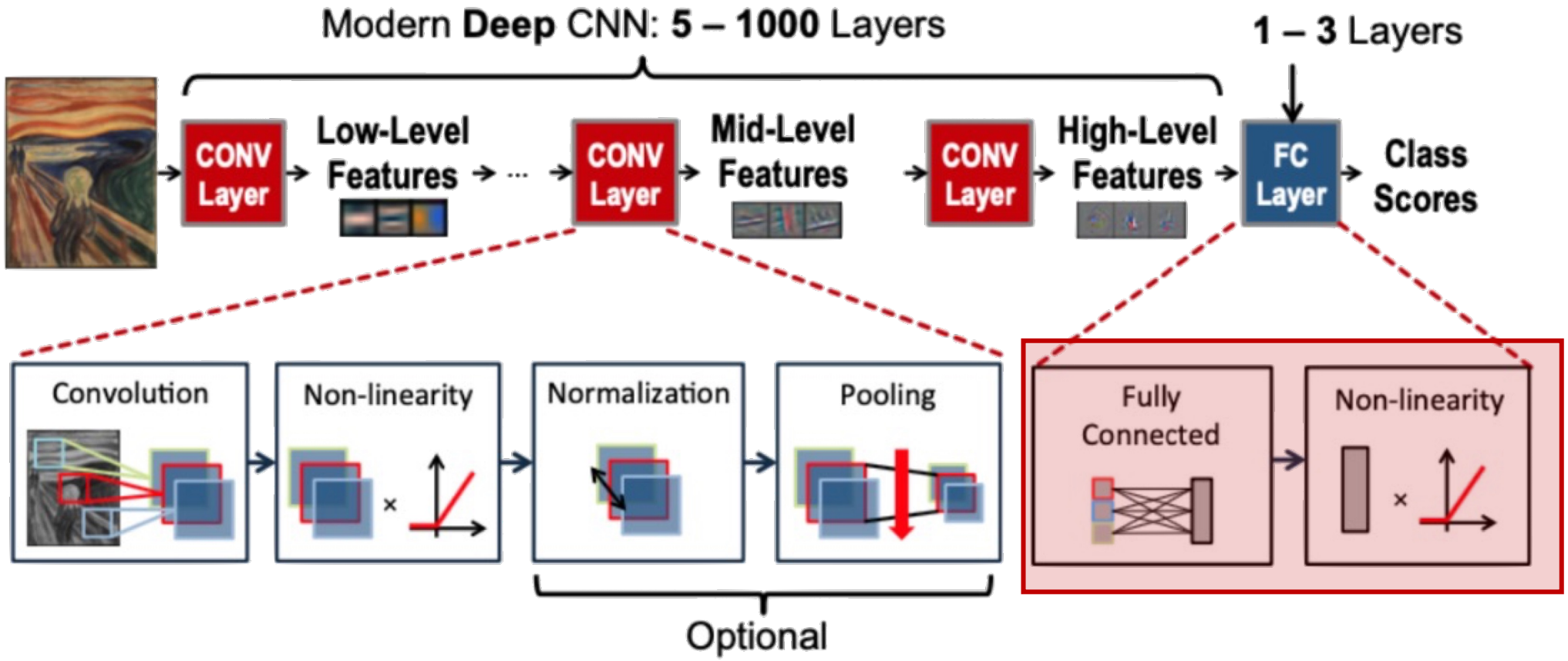
EE-508: Hardware Foundations for Machine Learning

Deeper DIVE into CDNNs (2)

University of Southern California

Ming Hsieh Department of Electrical and Computer Engineering

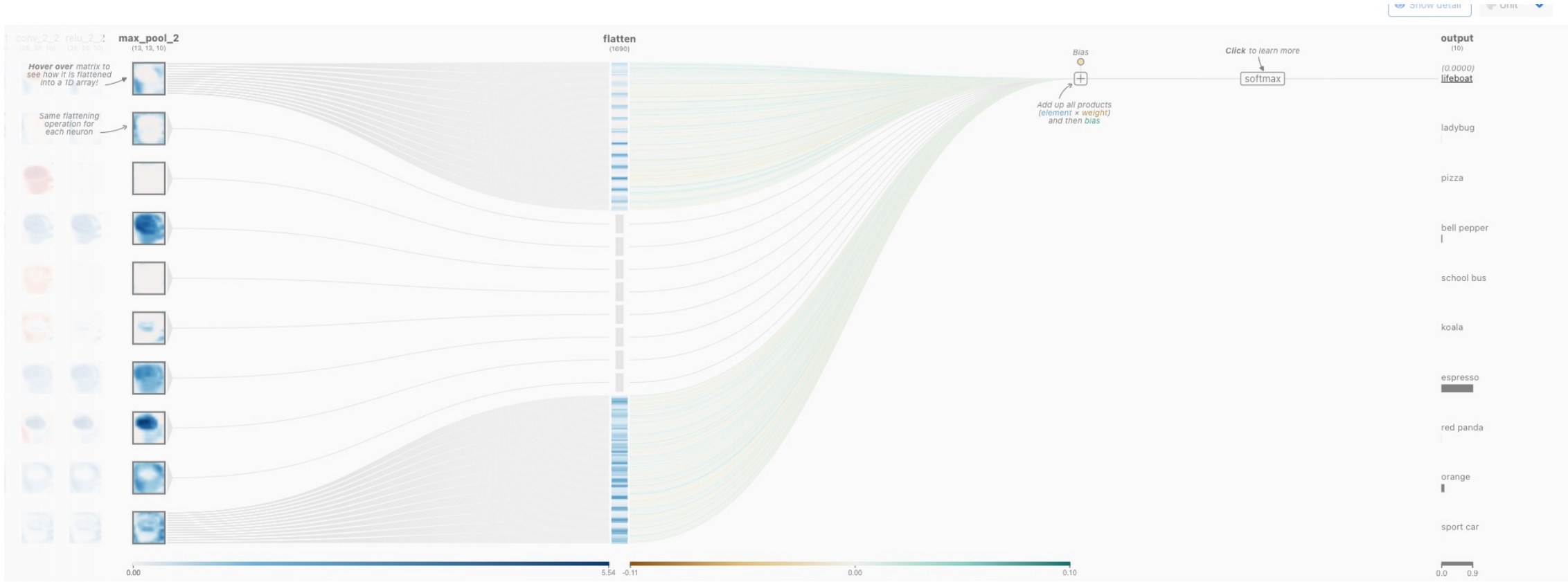
Instructor:
Arash Saifhashemi



Deep Convolutional Neural Networks

Source: eyeriss.mit.edu

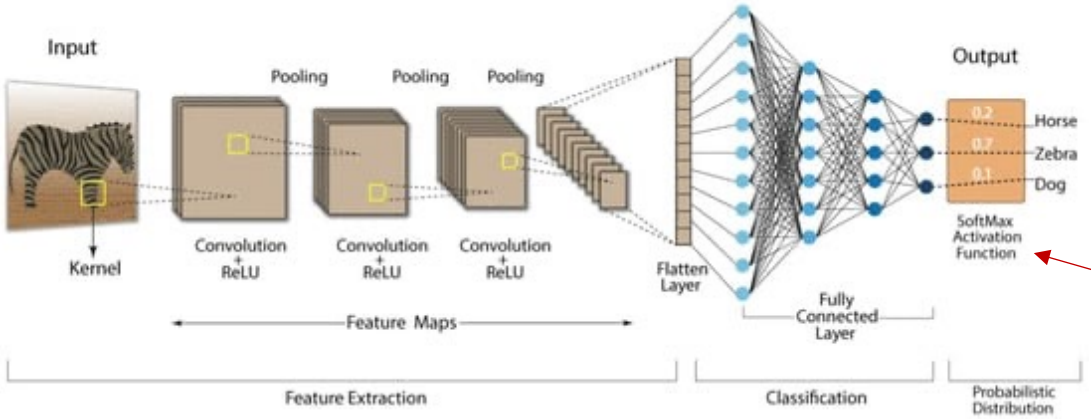
FCN for Multi-Class Classification



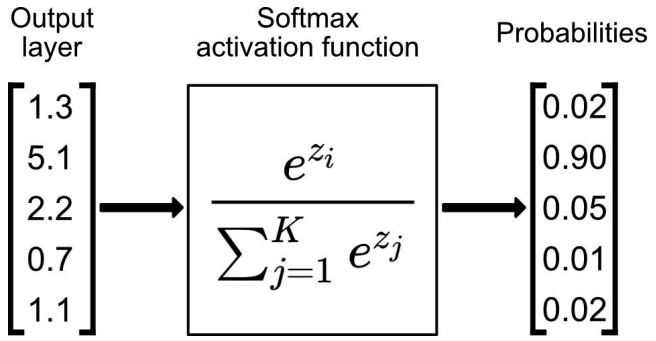
https://poloclub.github.io/cnn-explainer/

The SoftMax Function

SoftMax Function



Source: Ahmad Naebi et al.



Log-Odds

$$\text{Log-Odds} = \ln\left(\frac{p}{1-p}\right)$$

- For a fair coin ($p = 0.5$):

$$\text{Log-Odds} = \ln(1) = 0$$

  - Interpretation: The log-odds of getting heads are **0**, meaning equal chance.

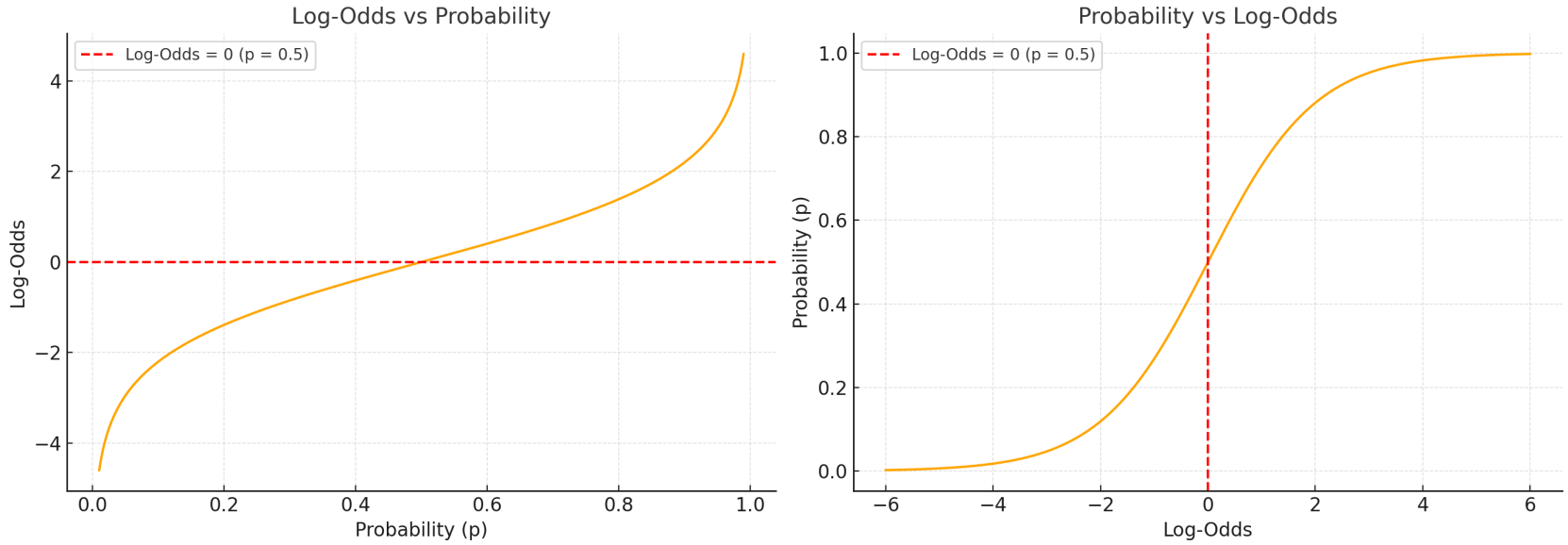
- For a biased coin ($p = 0.8$):

$$\text{Log-Odds} = \ln(4) \approx 1.386$$

  - Interpretation: Positive log-odds indicate heads are more likely than tails.

Probability (p)	**Odds**	**Log-Odds**
0.50	1	0
0.80	4	1.386
0.20	0.25	−1.386

Logistic Regression



1. **Model the Log-Odds**:
   - Fit a linear relationship between the features X and the log-odds of $y = 1$.

$$\text{Log-Odds} = \ln\left(\frac{p}{1-p}\right) = b_0 + b_1X_1 + b_2X_2 + \cdots + b_nX_n$$

2. **Convert Log-Odds to Probabilities**:
   - Apply the sigmoid function to the log-odds to get probabilities.

$$p = \frac{1}{1 + e^{-(b_0 + b_1X_1 + \cdots + b_nX_n)}}$$

3. **Classification**:
   - Assign class labels based on a probability threshold (e.g., $p > 0.5$).

SoftMax Function

- **What Are Logits?**
  - **Logits** are the raw, unnormalized scores output by a neural network before applying a probability function like **Softmax**.
  - They can take any real value $(-\infty, +\infty)$.
  - The Softmax function **converts logits into probabilities** for classification.

$$\text{logit}(p) = \log\left(\frac{p}{1-p}\right)$$

After applying Softmax:

$$\text{Logits} = [2.0, 1.0, -1.0]$$

$$\text{Probabilities} = [0.71, 0.26, 0.03]$$

•Instead of computing probabilities directly, neural networks output raw **logits**.

1. Logits **avoid numerical instability** (probabilities near 0 or 1 can cause computational issues).
2. They allow the model to **learn more flexibly**, since real-valued outputs are easier to optimize than constrained probability values.

•These logits can be **converted into probabilities** using Softmax in multi-class classification or the Sigmoid function for binary classification.

Resemblance Between Sigmoid and Softmax

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad \text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Sigmoid for binary classification. Softmax multi-class classification

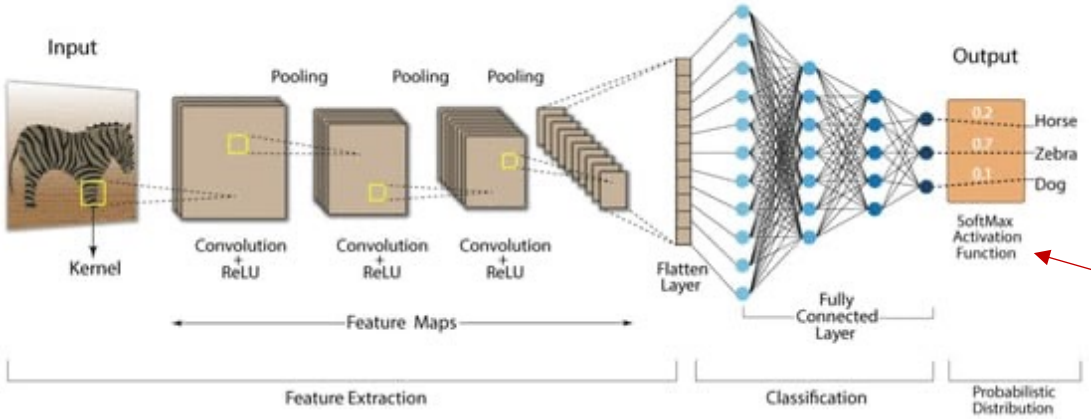
- **Sigmoid is a special case of Softmax** where $n = 2$ (binary classification with two classes).
- In Softmax, if there are only two classes, it behaves like **Sigmoid**, since:

$$\text{Softmax}([x_1, x_2]) = \left[\frac{e^{x_1}}{e^{x_1} + e^{x_2}}, \frac{e^{x_2}}{e^{x_1} + e^{x_2}}\right]$$

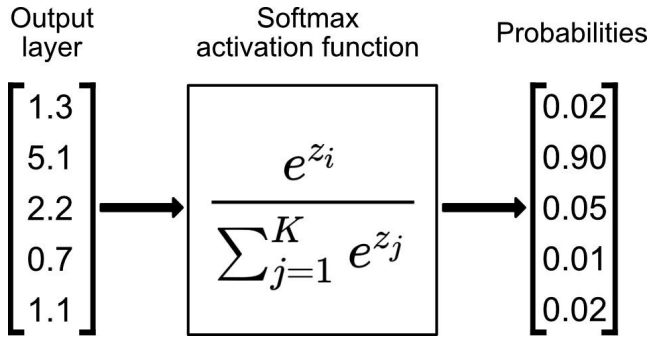
If $x_2 = 0$, the equation simplifies to the **Sigmoid function**.

$$\text{Softmax}(x_1) = \frac{e^{x_1}}{e^{x_1} + 1} \longrightarrow \sigma(x) = \frac{1}{1 + e^{-x}}$$

SoftMax Function

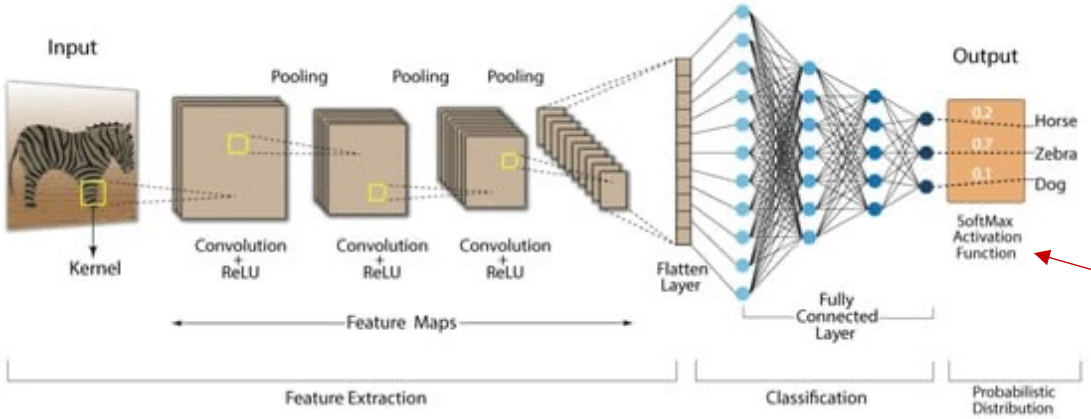


Source: Ahmad Naebi et al.

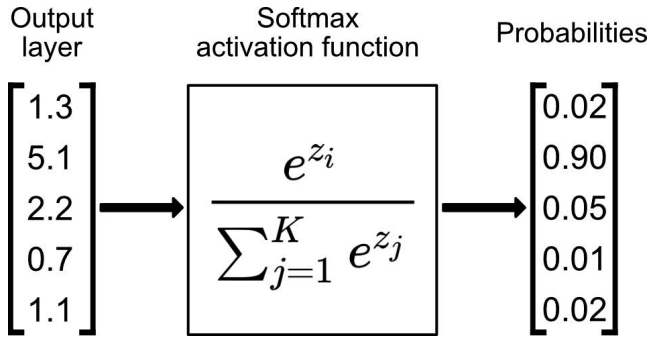


Question: Why use e^{z_i}? Why not just use z_i?

SoftMax Function



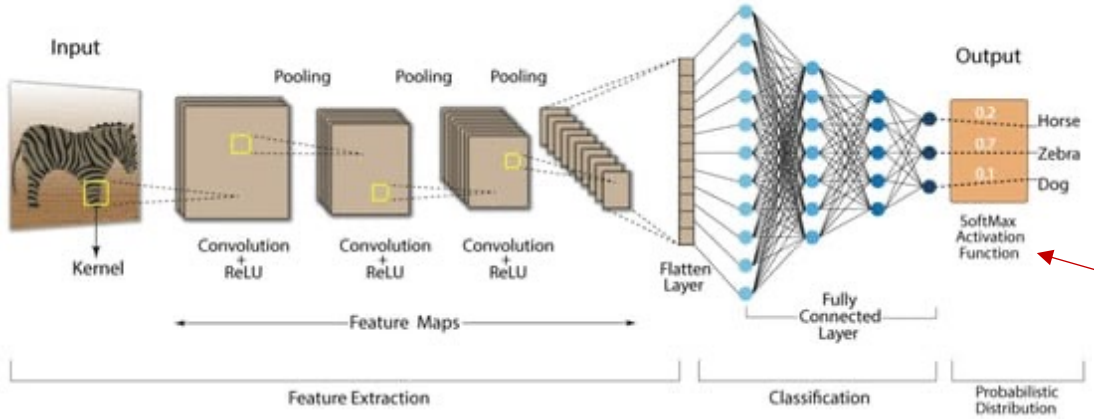
Source: Ahmad Naebi et al.



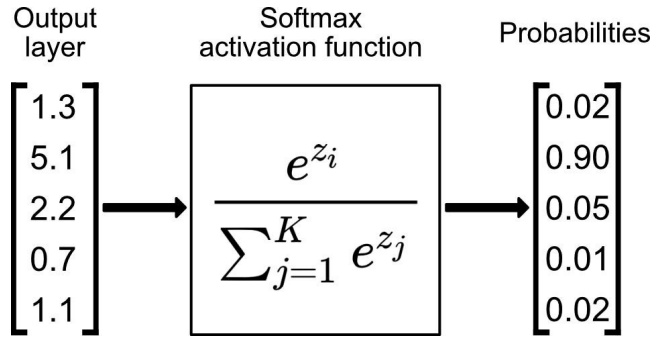
Question: Why use e^{z_i}? Why not just use z_i?

- Non negativity: since probabilities are non-negative

SoftMax Function



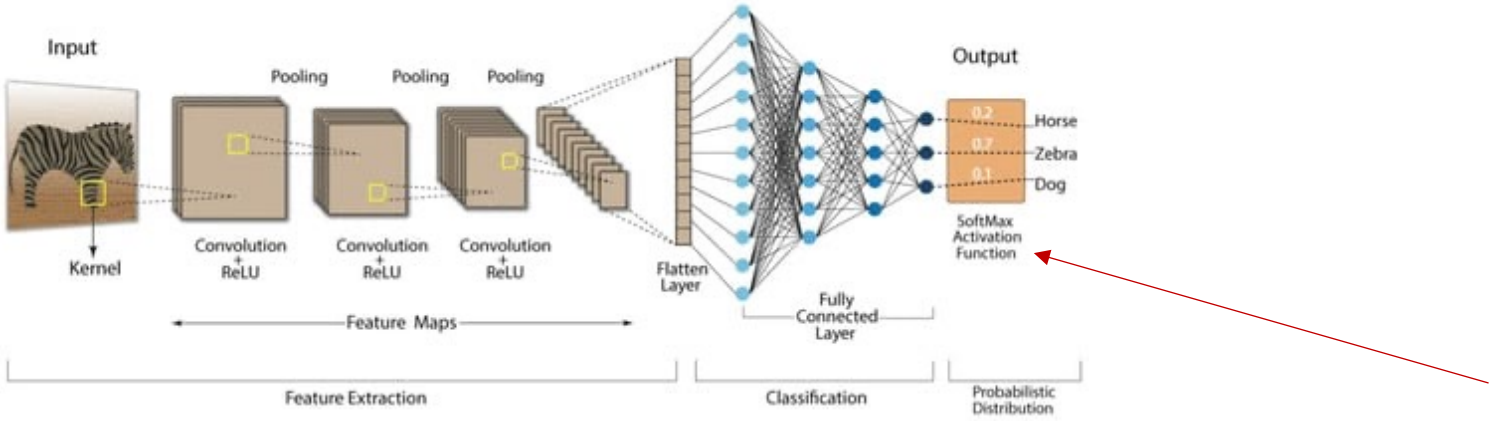
Source: Ahmad Naebi et al.



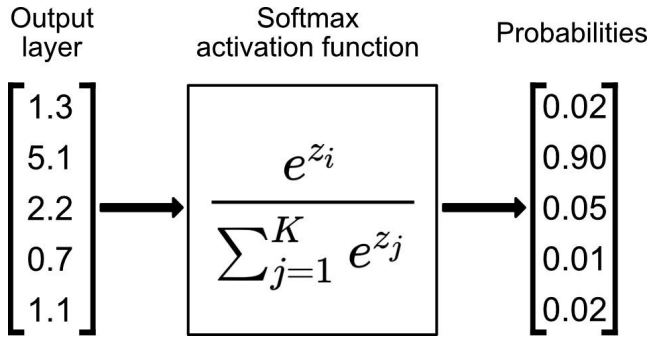
Question: Why use e^{z_i}? Why not just use z_i?

- Non negativity: since probabilities are non-negative
- Amplification of the difference: more discriminative outputs

SoftMax Function

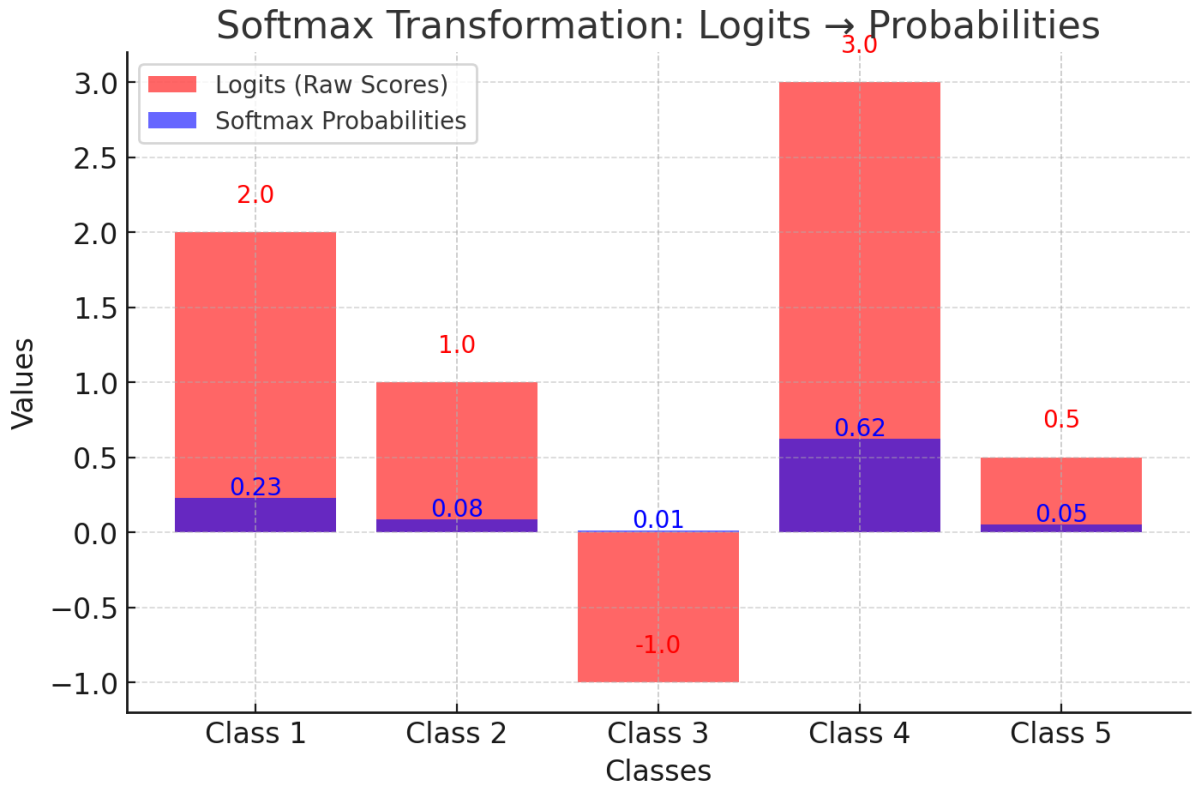


Source: Ahmad Naebi et al.



- **SoftMax as a "Soft" Maximum**: The softmax function can be seen as a "soft" version of the maximum function:
  - Instead of selecting the largest input value and assigning it a probability of 1 while assigning 0 to all others (as a hard maximum function would), softmax gives probabilities that are proportional to the exponentiated values.
  - This means that while the highest input value will have the highest probability, other values also have a chance proportional to their size, which is important for learning.

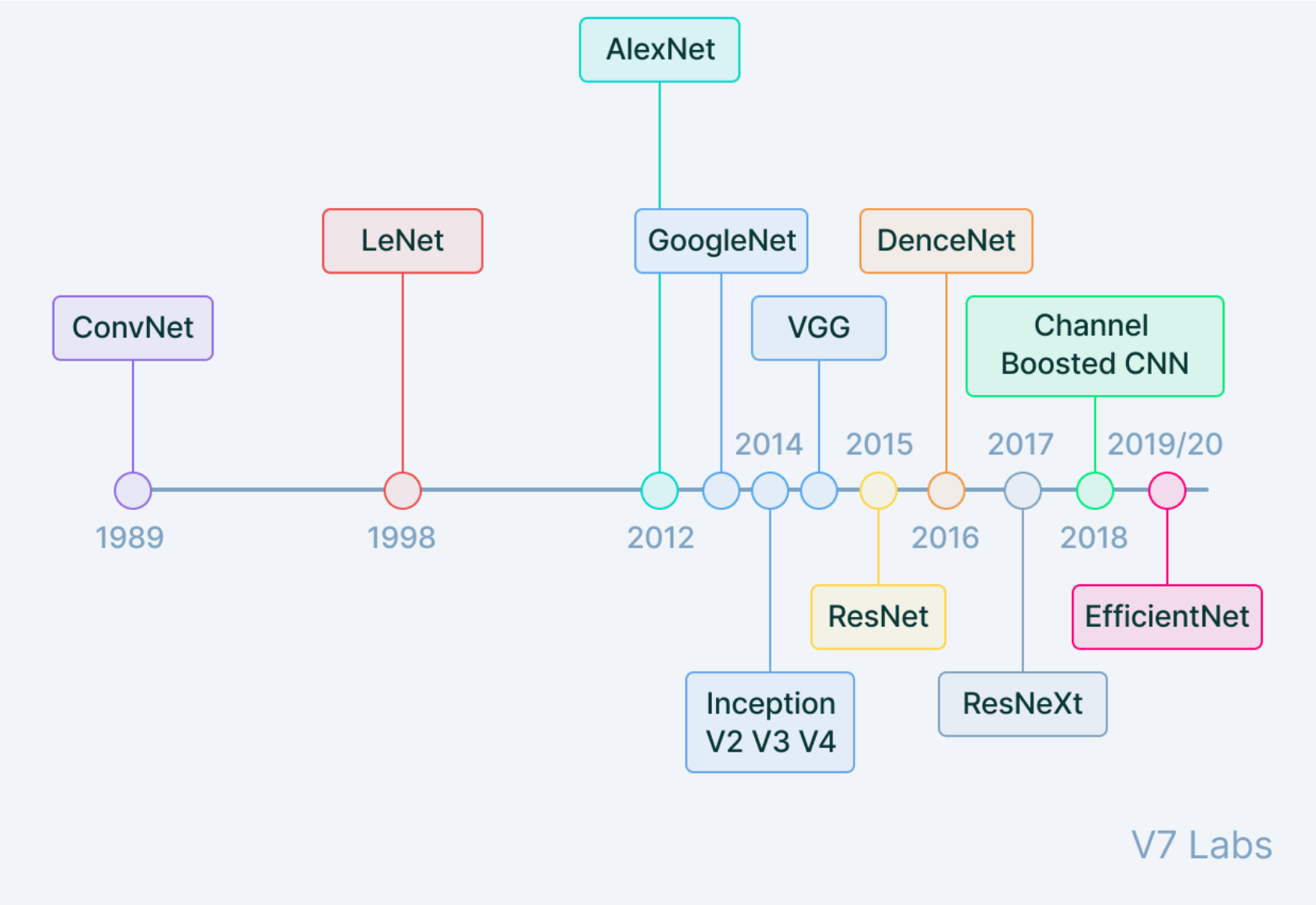
Softmax Function



- **Red Bars (Logits)**:
  - Raw output scores from the neural network.
  - Can be **negative or positive** and have no fixed range.
- **Blue Bars (Softmax Probabilities)**:
  - Transformed values where **all probabilities sum to 1**.
  - The highest logit gets the **highest probability**.

Famous CNN Models

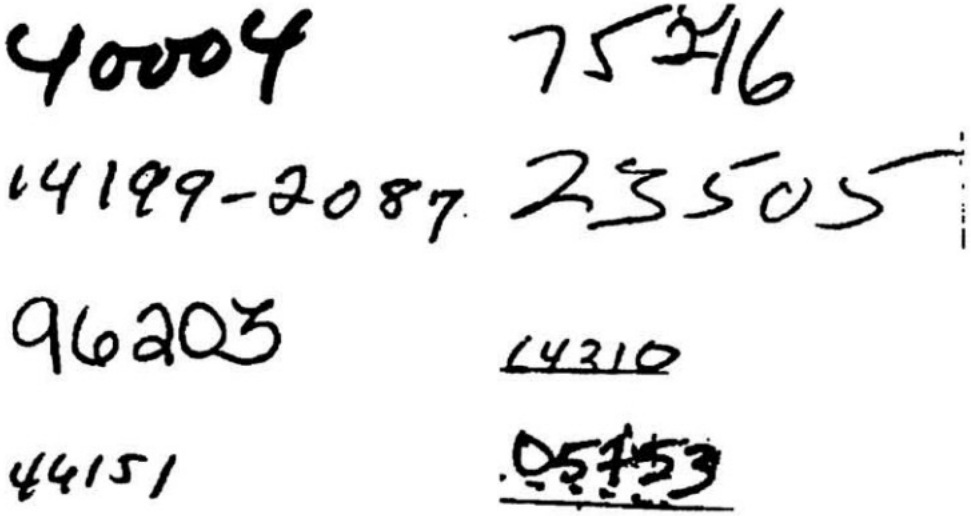
History of CNNs

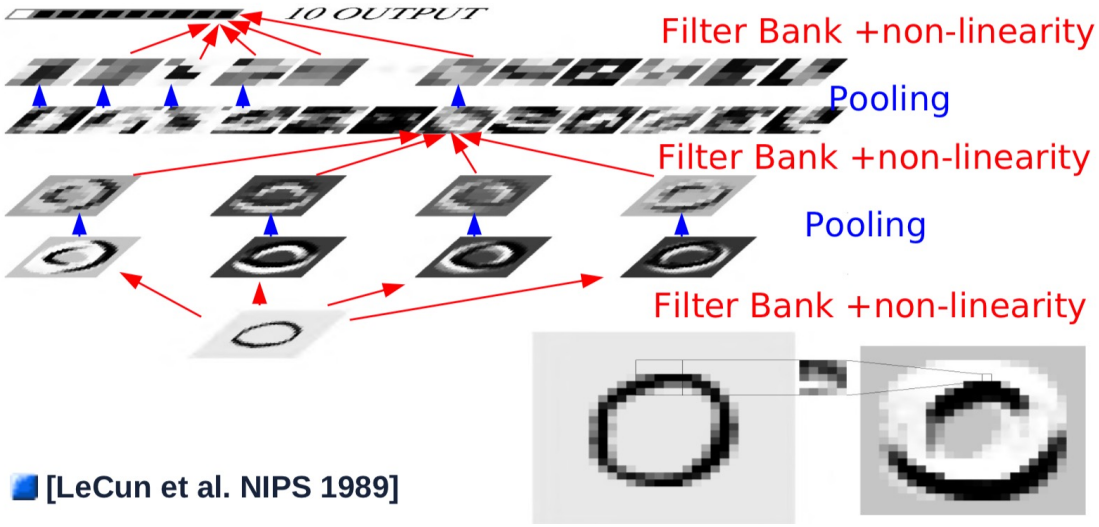


Source: v7labs.com

ConvNet

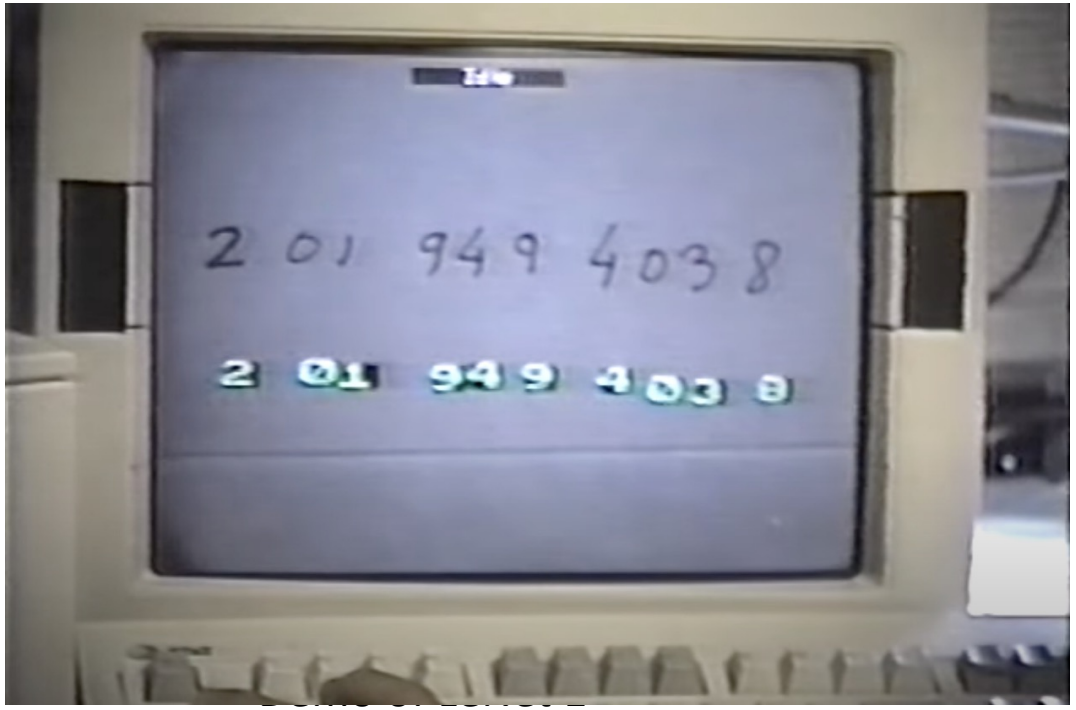
- First “real” ConvNet?
  - 1989
- Application:
  - Recognizing zip codes



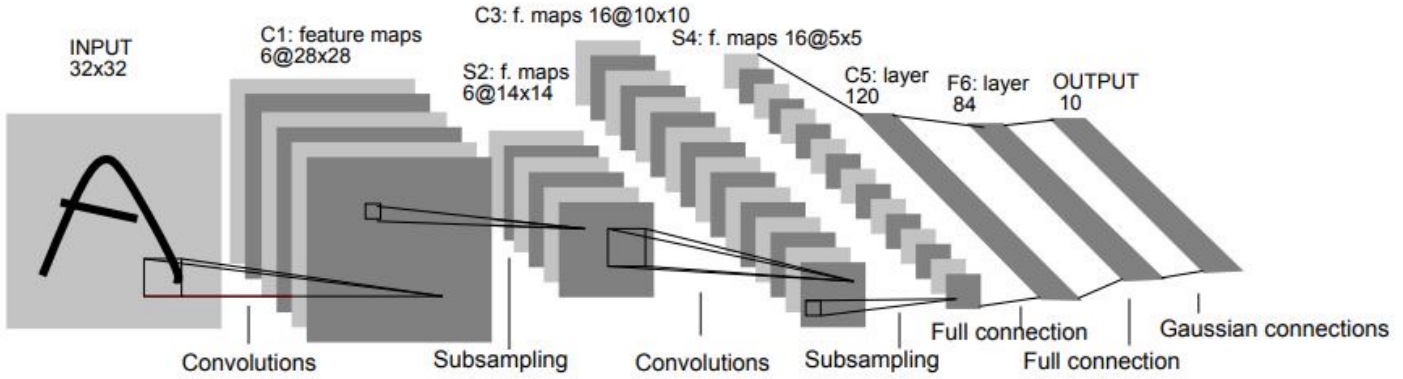


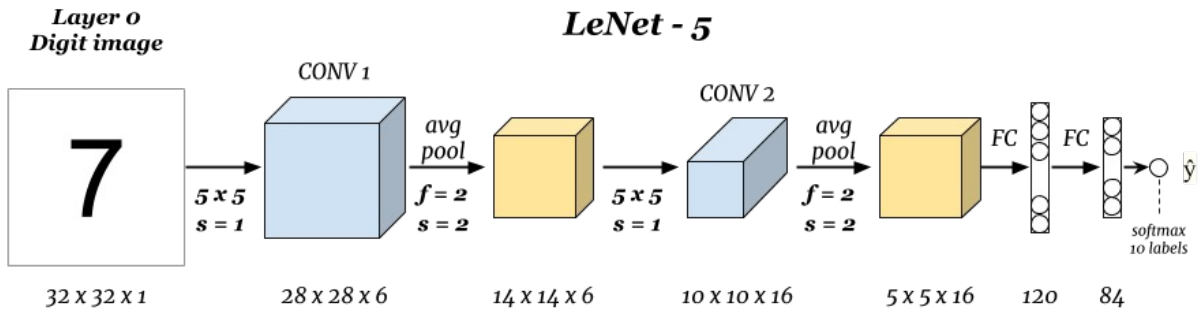
LeNet

- One of the first CNNs
  - 1998
- Handwritten Text recognition
- LeNet 5: ~60,000 parameters



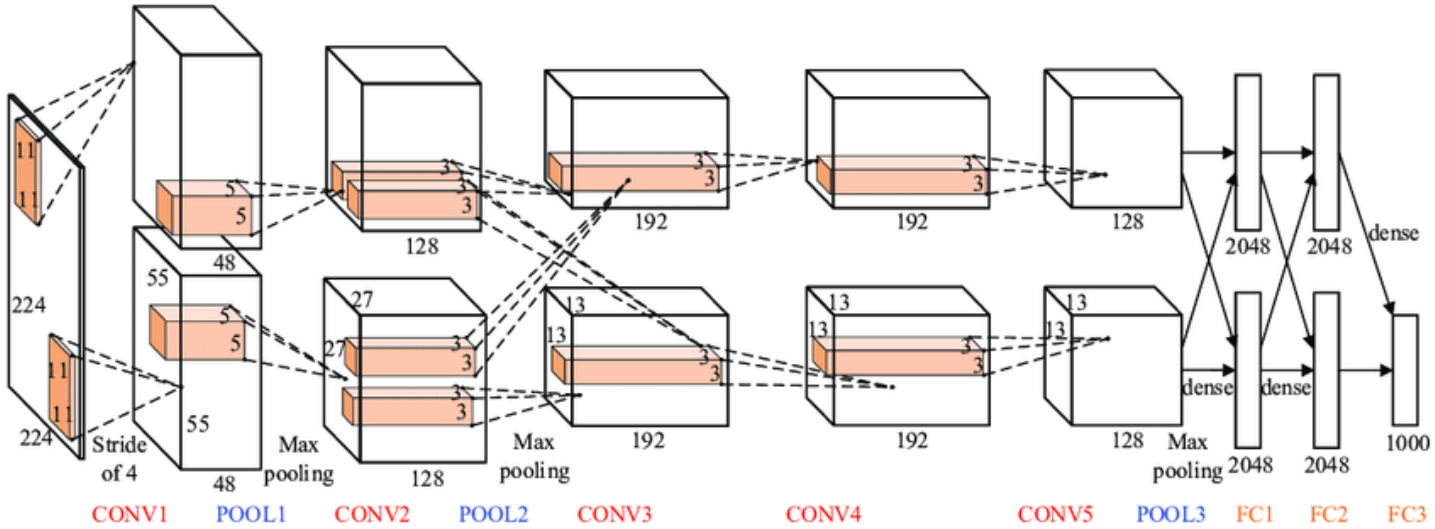
source: www.youtube.com/@YannLeCunPhD





AlexNet

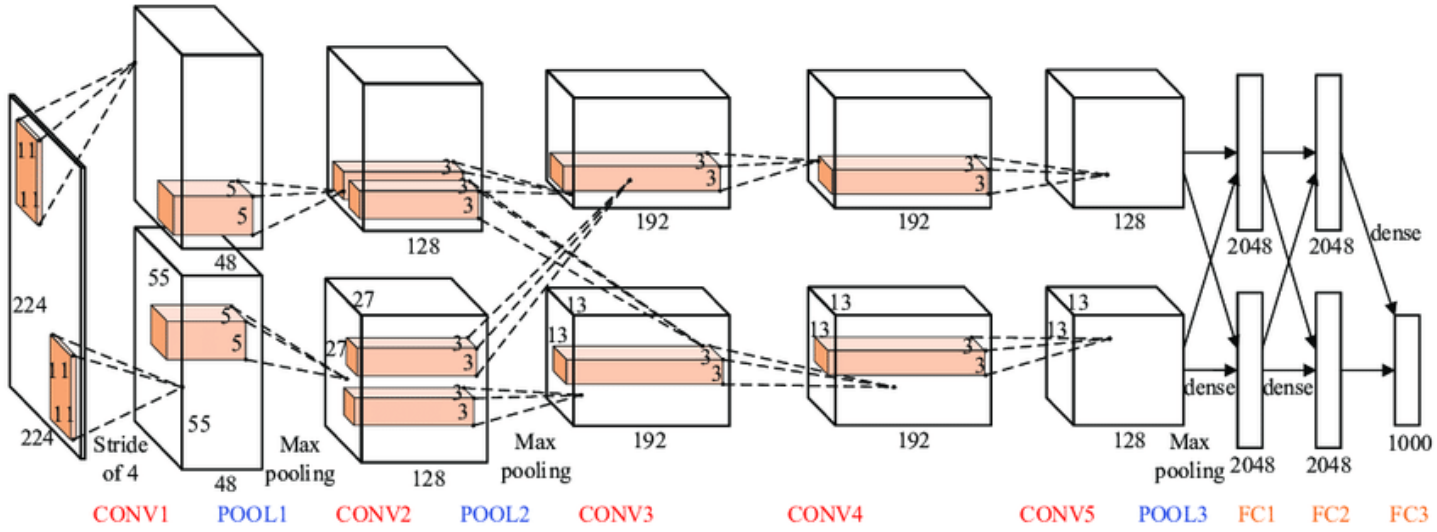
- **Year**: Introduced in 2012 at the ImageNet LSVRC.
- **Breakthrough**: Drastically reduced top-5 error rate by 10% compared to its predecessors.
- **Architecture**:
  - 5 convolutional layers,
  - max-pooling,
  - dropout,
  - 3 fully connected layers.
- **Activation**: Relu
- **Hardware**: NVIDIA's GTX 580 GPU, emphasizing the importance of GPU in deep learning.
- **Stats:** 61M weights and 724M MACs to process one 227x227 input image.



AlexNet

Why it Matters?

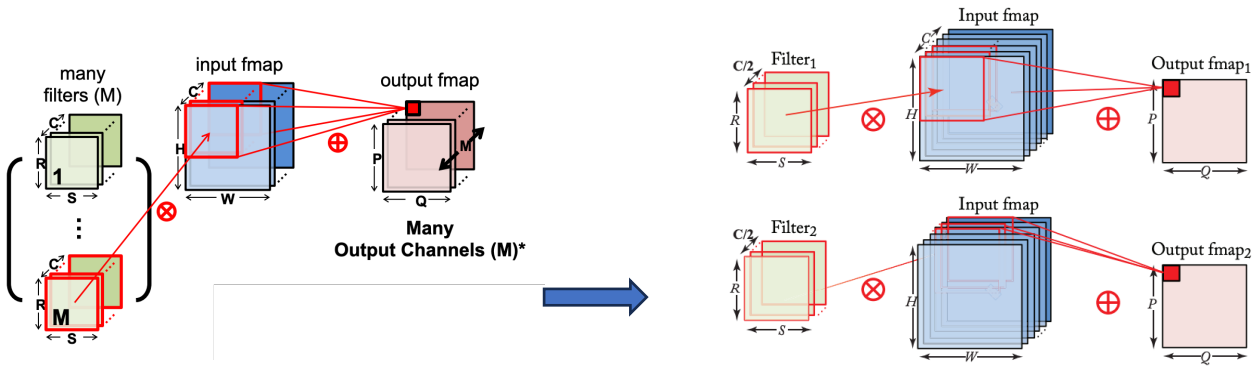
- Kick-started the modern Deep Learning era.
- Pioneered in large-scale image classification.
- Laid groundwork for future deep architectures.



AlexNet

Why it Matters?

- Kick-started the modern Deep Learning era.
- Pioneered in large-scale image classification.
- Laid groundwork for future deep architectures.

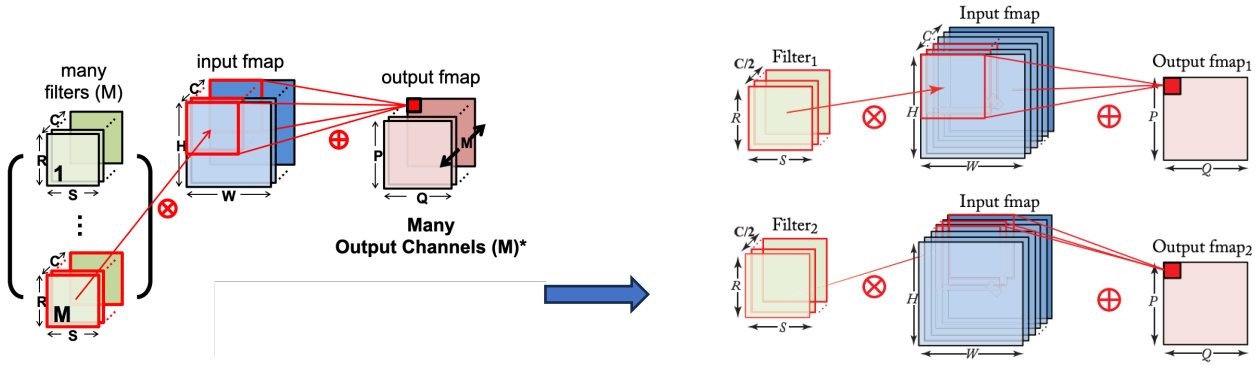


Group Convolution

Not all filters need to be convolved with all input channels.

- **Divide input feature maps into groups** (e.g., split 256 channels into two groups of 128).
- **Apply convolution separately to each group** instead of the entire feature map.
- **Merge outputs** at the end.

AlexNet



1. Standard Convolution (Without Grouping)

The total number of multiplications is:

$$R \times S \times C \times M \times W \times H$$

Where:

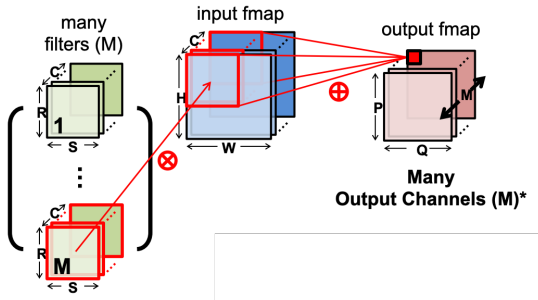
- R, S = Filter dimensions (kernel size, e.g., 5×5)
- C = Number of input channels (**96**)
- M = Number of filters (output channels, **256**)
- W, H = Spatial dimensions of the output feature map (**27 × 27**)

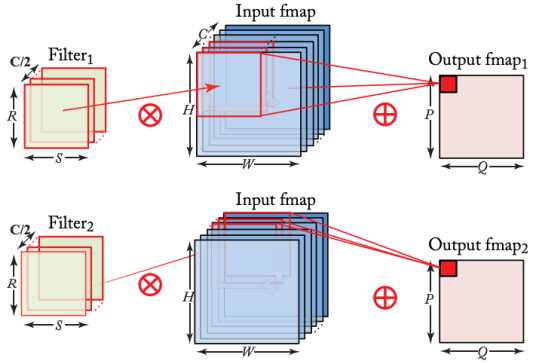
So, the number of operations:

$$5 \times 5 \times 96 \times 256 \times 27 \times 27$$

$$= 448,593,920 \text{ multiplications}$$

AlexNet





1. Standard Convolution (Without Grouping)

The total number of multiplications is:

$$R \times S \times C \times M \times W \times H$$

Where:

- R, S = Filter dimensions (kernel size, e.g., 5×5)
- C = Number of input channels (**96**)
- M = Number of filters (output channels, **256**)
- W, H = Spatial dimensions of the output feature map (**27 × 27**)

So, the number of operations:

$$5 \times 5 \times 96 \times 256 \times 27 \times 27$$

$$= 448,593,920 \text{ multiplications}$$

2. Group Convolution (With Grouping in AlexNet)

Since **AlexNet splits the 96 channels into 2 groups**, each group has:

- **Half the input channels:** $C/2 = 48$
- **Half the filters per group:** $M/2 = 128$

Total operations per group:

$$(R \times S \times \frac{C}{2} \times \frac{M}{2}) \times W \times H$$

Since there are **two groups**, multiply by **2**:

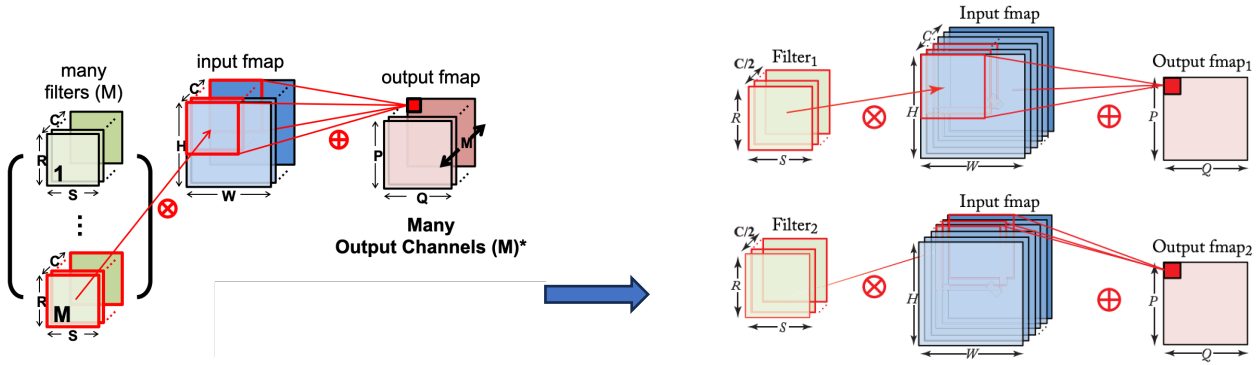
$$2 \times (R \times S \times \frac{C}{2} \times \frac{M}{2} \times W \times H)$$

Substituting values:

$$2 \times (5 \times 5 \times 48 \times 128 \times 27 \times 27)$$

$$= 224,296,960 \text{ multiplications}$$

AlexNet



1. Standard Convolution (Without Grouping)

The total number of multiplications is:

$$R \times S \times C \times M \times W \times H$$

Where:

- R, S = Filter dimensions (kernel size, e.g., 5×5)
- C = Number of input channels (**96**)
- M = Number of filters (output channels, **256**)
- W, H = Spatial dimensions of the output feature map (**27 × 27**)

So, the number of operations:

$$5 \times 5 \times 96 \times 256 \times 27 \times 27$$

$$= 448,593,920 \text{ multiplications}$$

2. Group Convolution (With Grouping in AlexNet)

Since **AlexNet splits the 96 channels into 2 groups**, each group has:

- **Half the input channels:** $C/2 = 48$
- **Half the filters per group:** $M/2 = 128$

Total operations per group:

$$(R \times S \times \frac{C}{2} \times \frac{M}{2}) \times W \times H$$

Since there are **two groups**, multiply by **2**:

$$2 \times (R \times S \times \frac{C}{2} \times \frac{M}{2} \times W \times H)$$

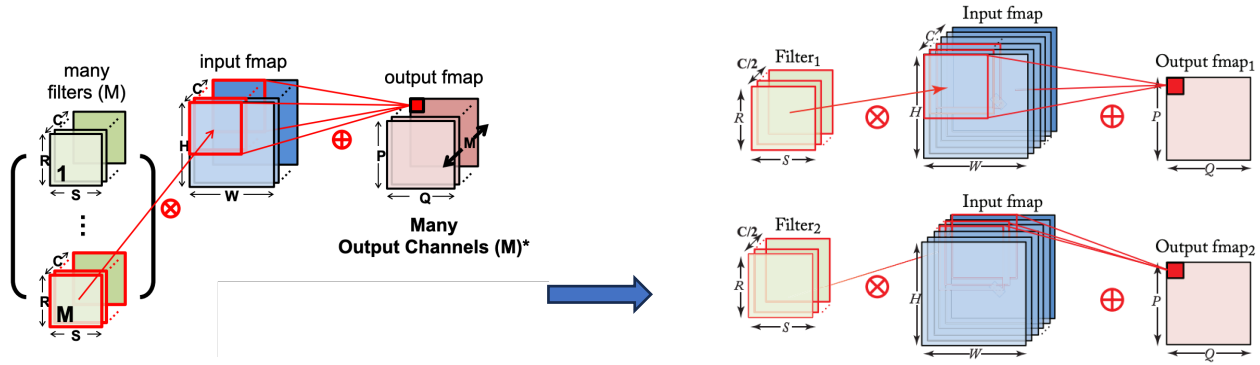
Substituting values:

$$2 \times (5 \times 5 \times 48 \times 128 \times 27 \times 27)$$

$$= 224,296,960 \text{ multiplications}$$

Group Convolution can act as an implicit form of **regularization** in CNNs by **reducing parameter redundancy and enforcing structured learning**.

AlexNet



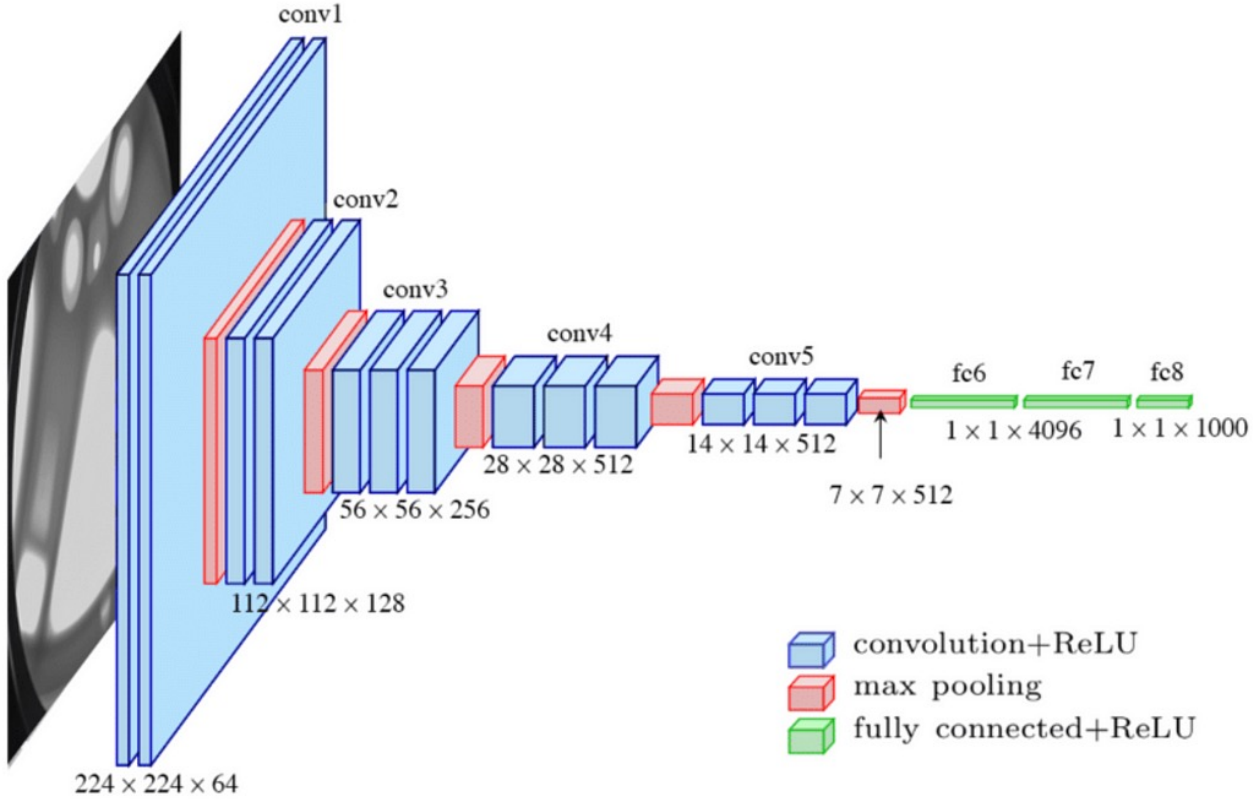
Approach	Formula	Total Multiplications
Without Grouping	$R \times S \times C \times M \times W \times H$	**448.6M**
With Grouping (AlexNet)	$2 \times (R \times S \times \frac{C}{2} \times \frac{M}{2} \times W \times H)$	**224.3M (50% Reduction!)**

Standard Convolution	**Group Convolution (AlexNet)**
All filters process all channels	**Filters process only a subset of channels**
High redundancy (filters learn similar features)	Less redundancy (filters specialize)
More computation	50% fewer multiplications
High overfitting risk	Lower overfitting risk
Works well on modern GPUs	Designed for AlexNet's **dual GPU setup**

AlexNet (2012) was **too large to fit on a single GPU**, so it was **split across two GPUs** to handle computations efficiently.

VGG-16

- **Year**: Introduced in 2014 by the Visual Geometry Group (VGG) at the University of Oxford.
- **Structure**: 16 layers (13 convolutional and 3 fully connected layers).
- **Depth**: Consistent use of 3x3 filters and deeper networks compared to predecessors.
- **Activation**: ReLU (Rectified Linear Units) used throughout for non-linearity.
- **Pooling**: Five max-pooling layers used to progressively reduce the spatial size.



Why it Matters?

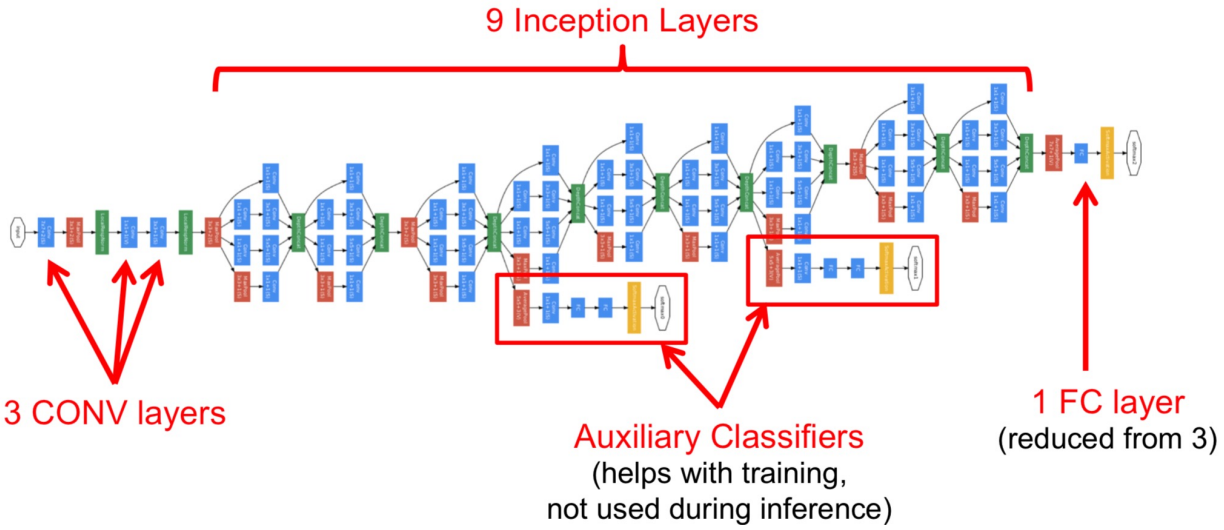
- A strong performance in the ImageNet Challenge.
- Simplicity: Uniform architecture
- Transfer Learning: Due to its depth and simplicity, VGG-16 became a popular model for transfer learning in various applications.
- Feature Extraction: Used as a base model in many computer vision tasks for extracting features from images.

GoogLeNet

- Developed by **Google** researchers in **2014**.
  - Parallel Convolutions
  - Captures features at various scales.
  - The weights and the activations could all fit into the GPU memory
  - Applications
    - Image Classification
    - Object Detection

CONV Layers: 21 (depth), 57 (total)
Fully Connected Layers: 1
Weights: 7.0M
MACs: 1.43G

Also, v2, v3 and v4
ILSVRC14 Winner



Why it Matters?

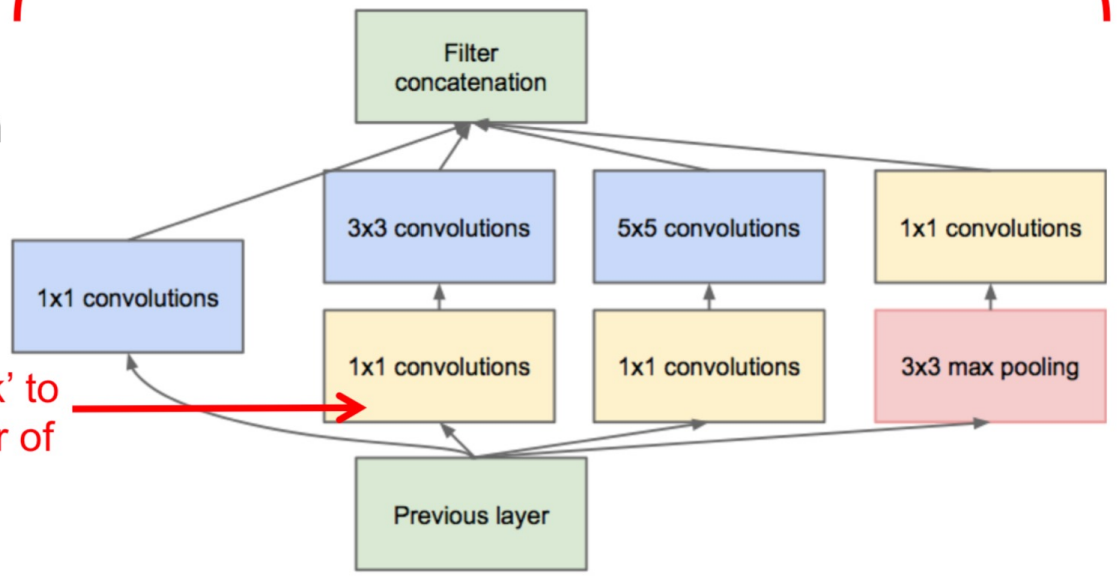
- Efficiency and Scalability (1x1 filters and global average pooling)
- Multiscale convolutions capture diverse features.
- Supports accurate classification and object detection.
- Inspires development of efficient neural architectures.
- Serves as a foundation for subsequent CNN designs.
- Achieves state-of-the-art accuracy in image classification.

Inception

CONV Layers: 21 (depth), 57 (total)
Fully Connected Layers: 1
Weights: 7.0M
MACs: 1.43G

Also, v2, v3 and v4
ILSVRC14 Winner

parallel filters of different size have the effect of processing image at different scales



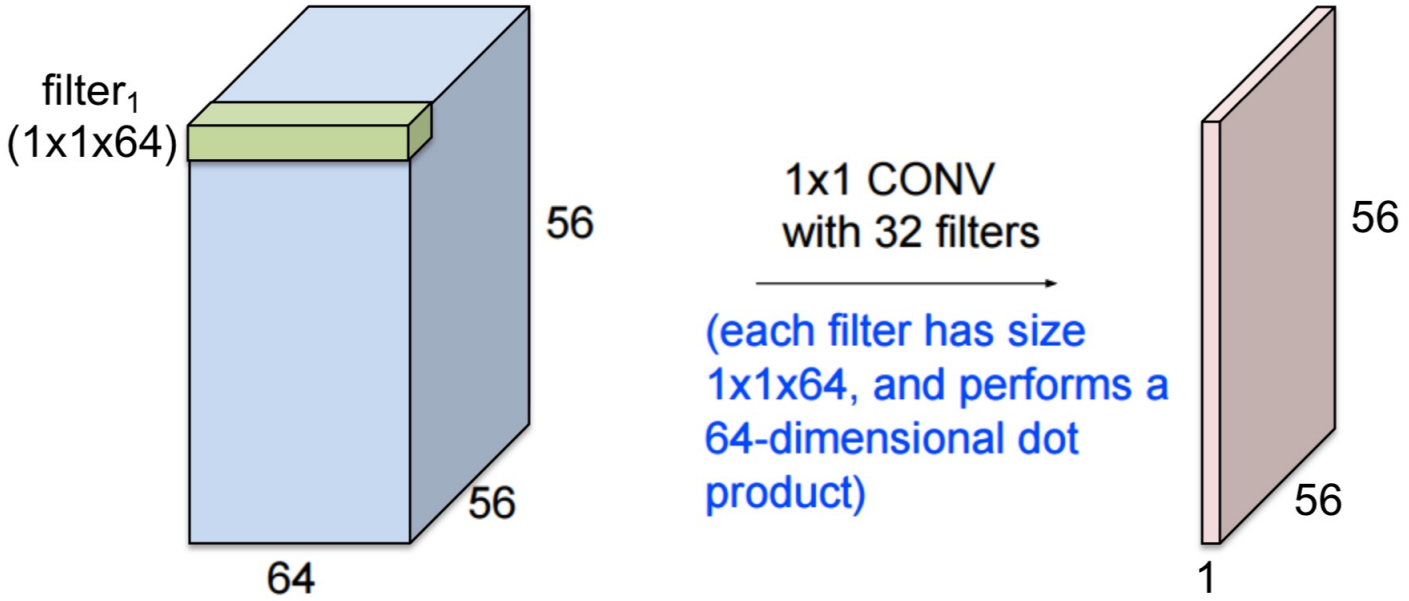
1x1 'bottleneck' to reduce number of weights and multiplications

What is 1×1 Convolution in Inception (GoogleNet)?

- In **GoogleNet (Inception)**, **1x1 convolution** is used for:
  1. **Dimensionality Reduction** → Reducing the number of input channels before expensive convolutions.
  2. **Efficient Computation** → Fewer parameters and faster processing.
  3. **Feature Mixing** → Combining information from different channels.

Bottleneck

Use **1x1 filter** to capture cross-channel correlation, but no spatial correlation.
Can be used to reduce the number of channels in next layer (**bottleneck**)

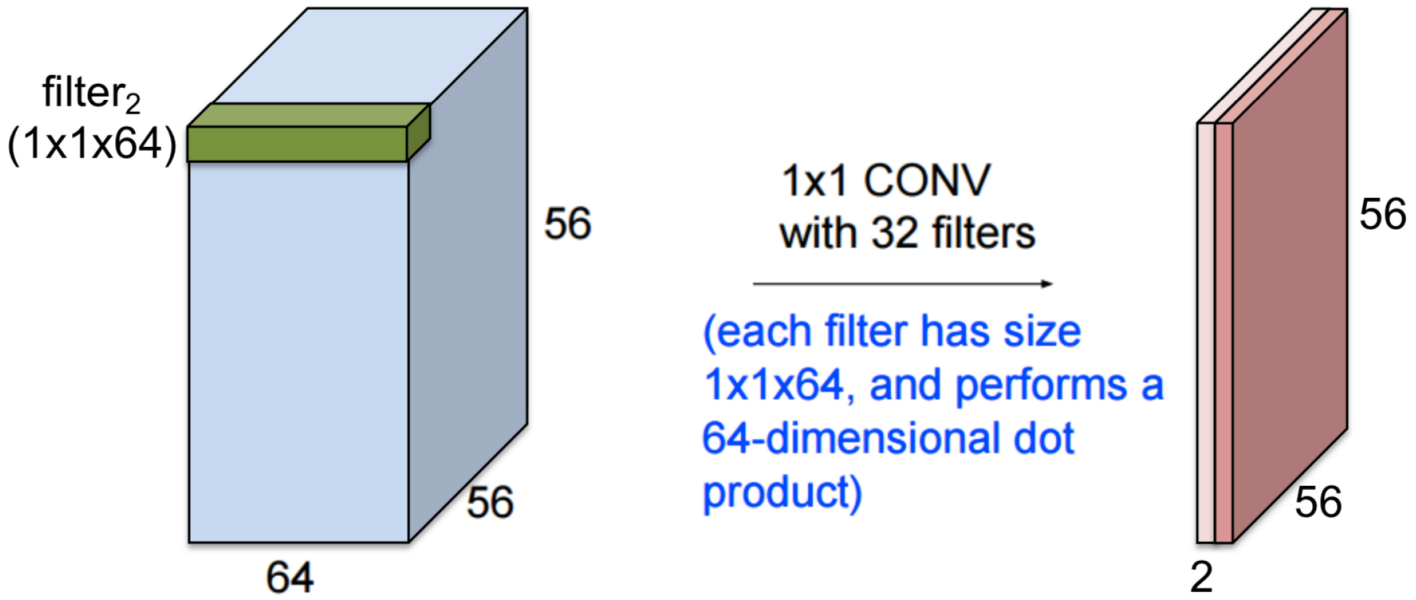


Modified image from source:
Stanford cs231n

Source: [Lin et al., Network in Network, Arxiv 2013, ICLR 2014]

Bottleneck

Use **1x1 filter** to capture cross-channel correlation, but no spatial correlation.
Can be used to reduce the number of channels in next layer (**bottleneck**)

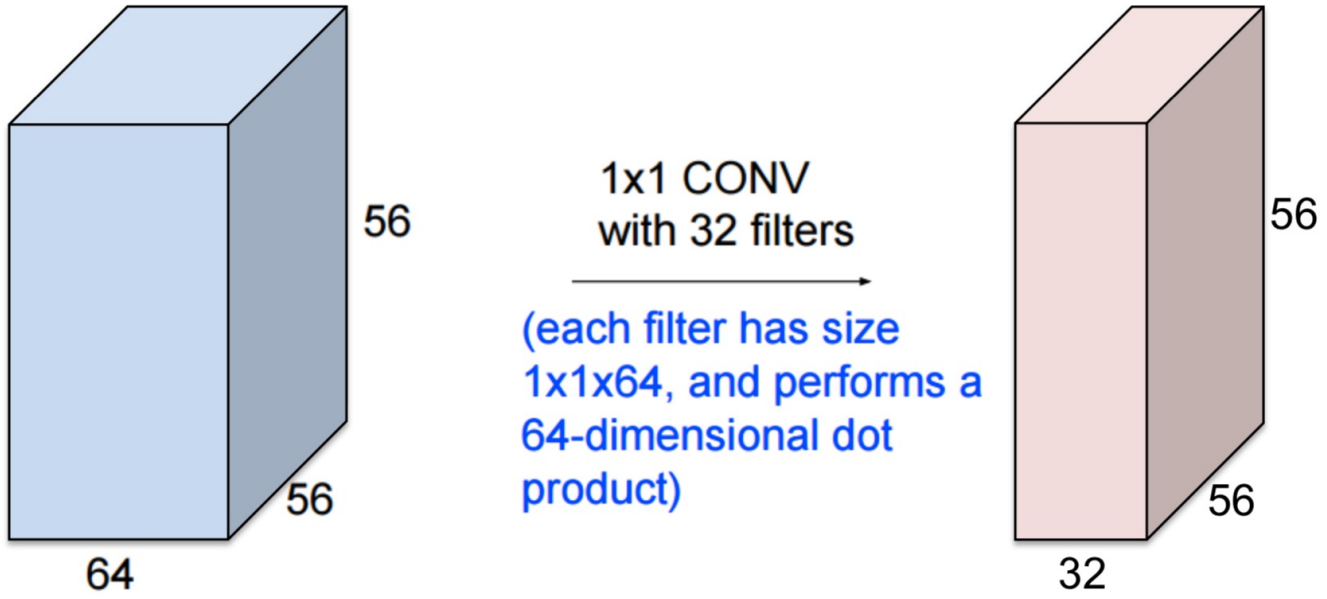


Modified image from source:
Stanford cs231n

Source: [Lin et al., Network in Network, Arxiv 2013, ICLR 2014]

Bottleneck

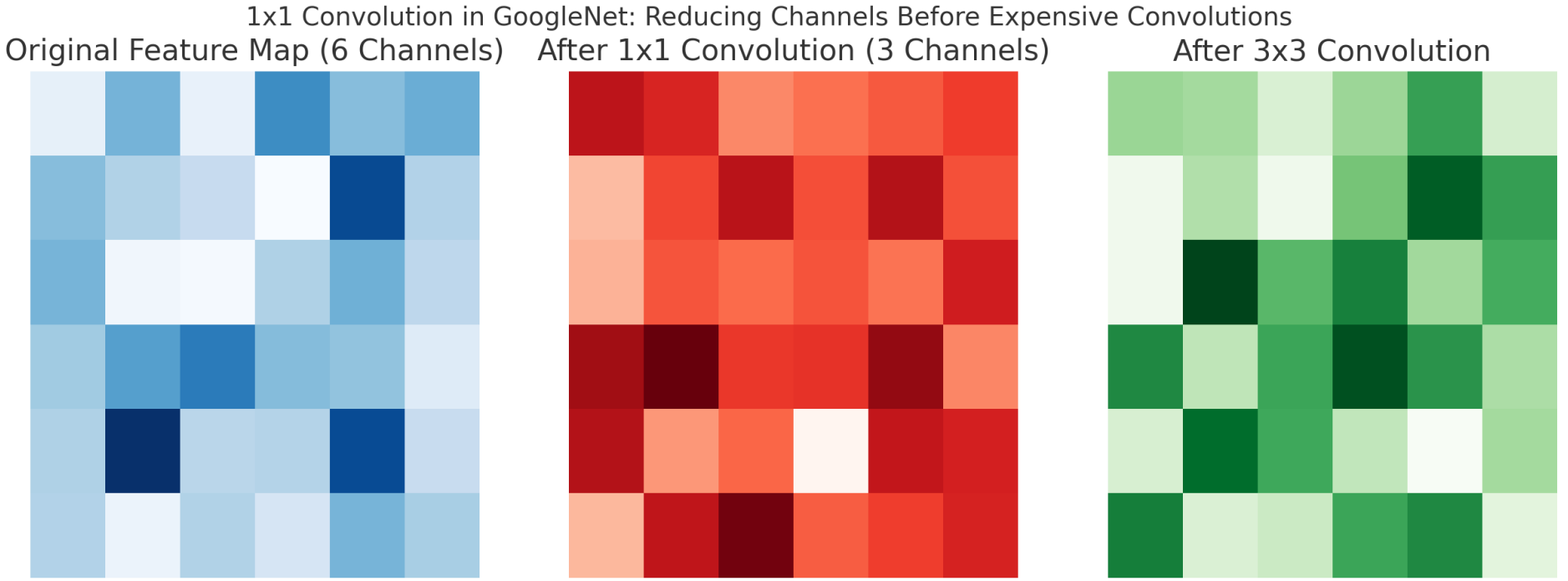
Use **1x1 filter** to capture cross-channel correlation, but no spatial correlation.
Can be used to reduce the number of channels in next layer (**bottleneck**)



Modified image from source:
Stanford cs231n

Source: [Lin et al., Network in Network, Arxiv 2013, ICLR 2014]

How 1×1 Convolution Works



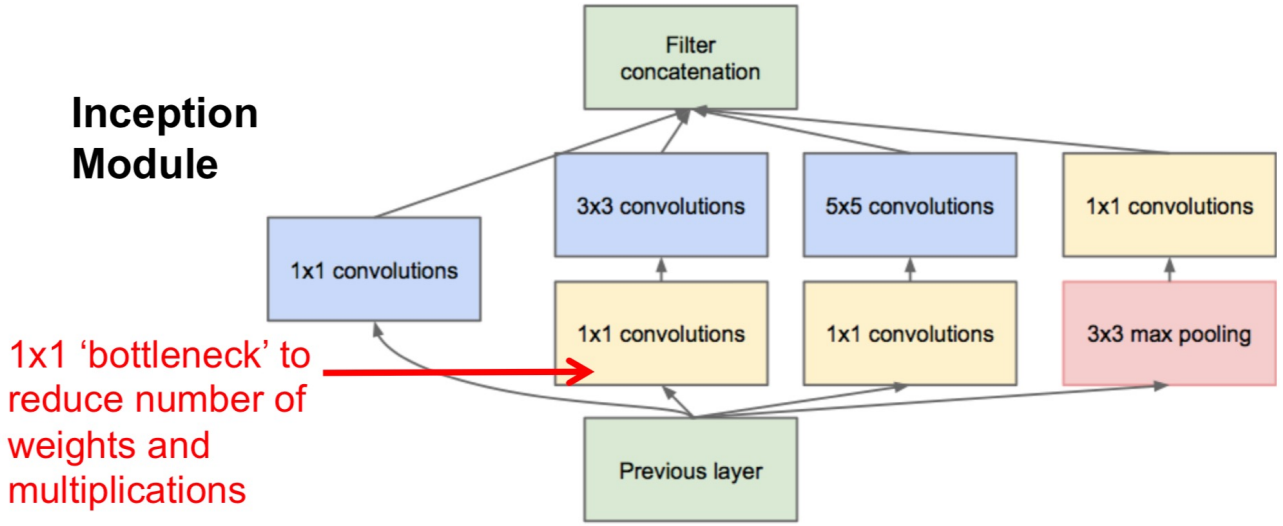
- **Original Feature Map - 6 Channels**
  - The full input feature map before applying **1×1 convolution**.
  - Contains a **high number of channels**, making standard convolutions expensive.

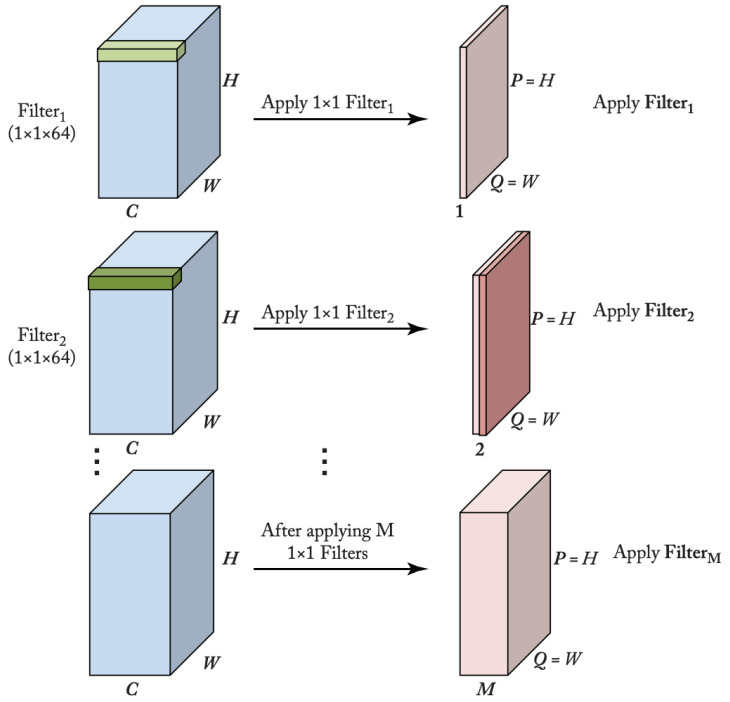
- **After 1×1 Convolution - Reduced to 3 Channels**
  - The **1×1 convolution reduces channel count** before applying a large kernel.
  - This saves **computational cost** while retaining important features.

- **After 3×3 Convolution**
  - A **standard 3×3 convolution** is applied **after channel reduction**.
  - This is much **cheaper** than applying 3×3 on the original feature map.

Bottleneck

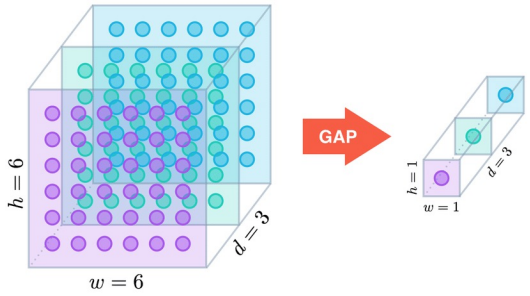
Apply bottleneck before ‘large’ convolution filters.
Reduce weights such that **entire CNN can be trained on one GPU.**
Number of multiplications reduced from 854M → 358M



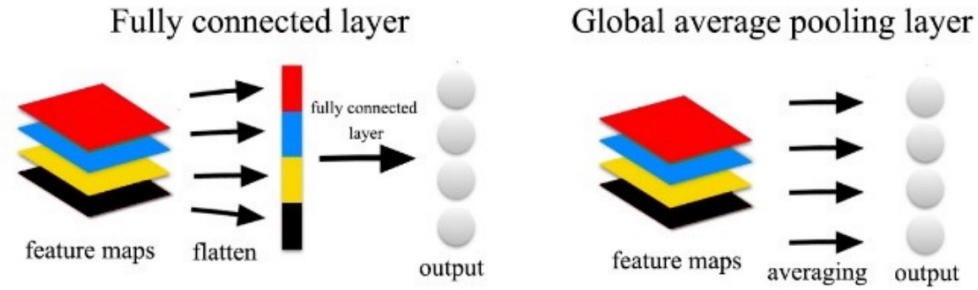


Global Pooling

- **Operation**: Reduces a *H*×*W*×*C* feature map to 1×1×*C*.
- **Purpose**:
  - **Dimensionality Reduction**: Facilitates connection to classification layers without FC layers.
  - **Robustness**: Minimizes overfitting by eliminating many parameters.



Source: Alena Selimović et al.

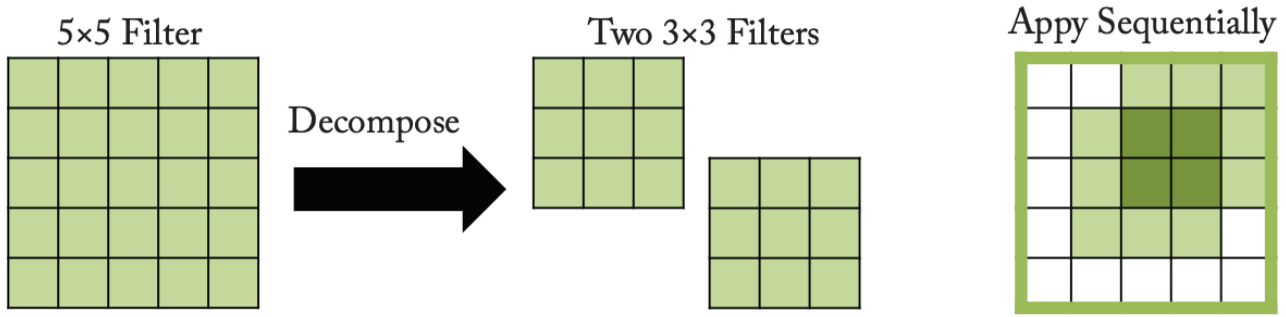


Source: Yuanyuan Guo et al.

In GAP, instead of using a pooling window that moves across the feature map, each entire feature map (per channel) is pooled individually to a single number.

It is often used as an alternative to Fully Connected layers at the end of the CNN for making predictions, effectively reducing the total number of weights in the model.

Decomposing Filters



(a) Constructing a 5×5 support from 3×3 filters. Used in VGG-16.

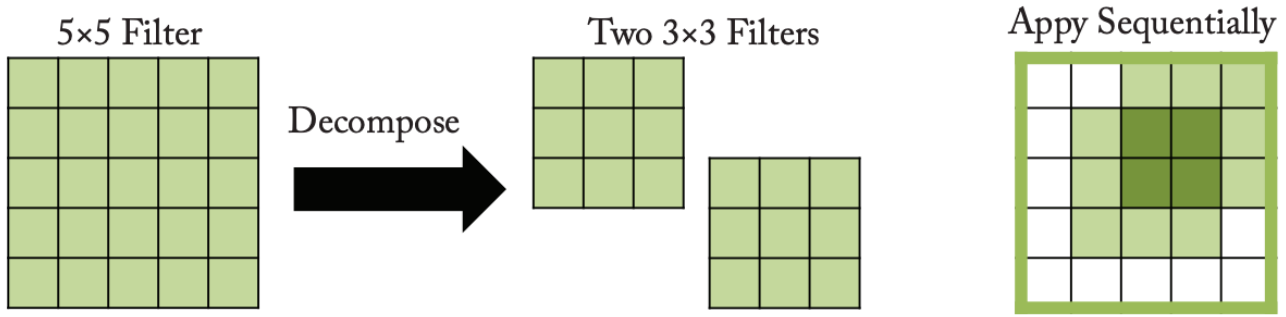
$$Output\ Feature\ Map\ = \frac{(Input\ Feature\ Map - \ Filter\ +\ Stride + 2 \times padding)}{Stride}$$

$$Output\ Feature\ Map\ = \frac{(N-5\ +1\ )}{1} = N-4 \qquad (W = H = N)$$ Input and filters are square

Decomposing a 5×5 filter into two 3×3 filters means that the effect (or receptive field) of applying a single 5×5 filter can be approximated by successively applying two 3×3 filters. It doesn't mean we can exactly represent the 5×5 matrix values with two 3×3 matrices.

The term **"support"** in this context refers to the **receptive field** of a filter, which is the size of the region in the input that the filter covers at any given time.

Decomposing Filters (After applying the first filter)



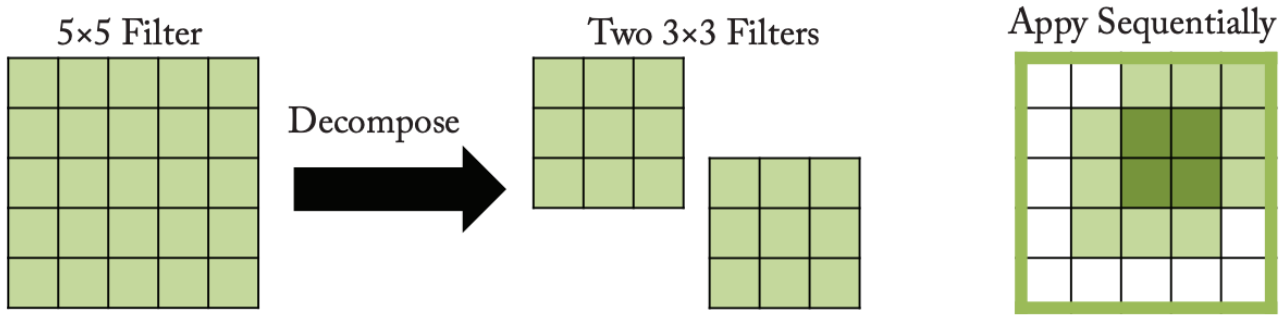
(a) Constructing a 5×5 support from 3×3 filters. Used in VGG-16.

$$Output\ Feature\ Map\ = \frac{(Input\ Feature\ Map - \ Filter\ +\ Stride + 2 \times padding)}{Stride}$$

$$Output\ Feature\ Map\ = \frac{(N-5\ +1\ )}{1} = N-4$$

$$Output\ Feature\ Map_1\ = \frac{(N-3\ +1\ )}{1} = N-2$$

Decomposing Filters (After applying the second filter)



(a) Constructing a 5×5 support from 3×3 filters. Used in VGG-16.

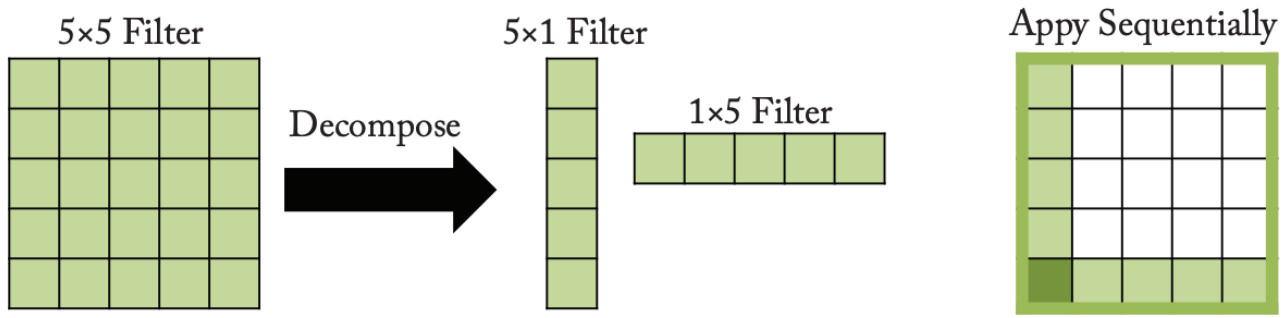
$$Output\ Feature\ Map\ = \frac{(Input\ Feature\ Map - \ Filter\ +\ Stride + 2 \times padding)}{Stride}$$

$$Output\ Feature\ Map\ = \frac{(N-5\ +1\ )}{1} = N-4$$

$$Output\ Feature\ Map_1\ = \frac{(N-3\ +1\ )}{1} = N-2$$

$$Output\ Feature\ Map_2\ = \frac{((N-2)-3+1\ )}{1} = N-4$$

Decomposing Filters



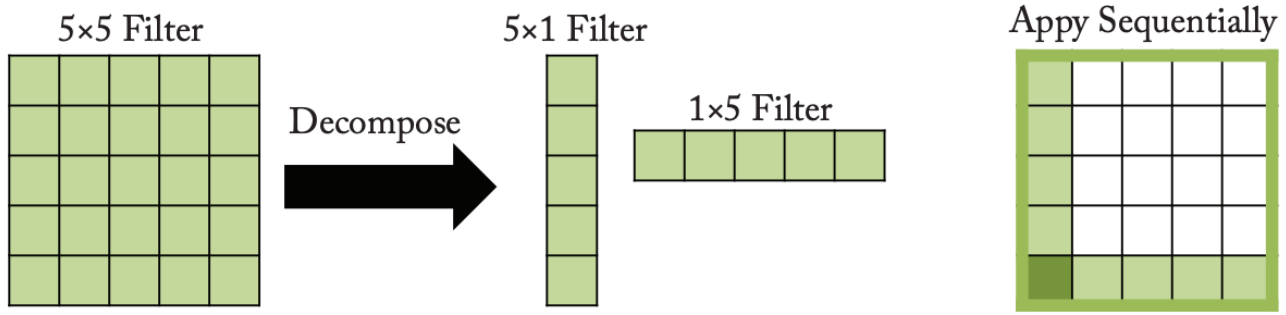
(b) Constructing a 5×5 support from 1×5 and 5×1 filter. Used in GoogLeNet/Inception v3 and v4.

$$Output\ Feature\ Map\ = \frac{(Input\ Feature\ Map - \ Filter\ +\ Stride + 2 \times padding)}{Stride}$$

$$Output\ Feature\ Map\ = \frac{(N-5\ +1\ )}{1} = N-4$$

The 5x1 filters are not square, but we can calculate the width and height of the OFM.

Decomposing Filters (After applying the first filter)



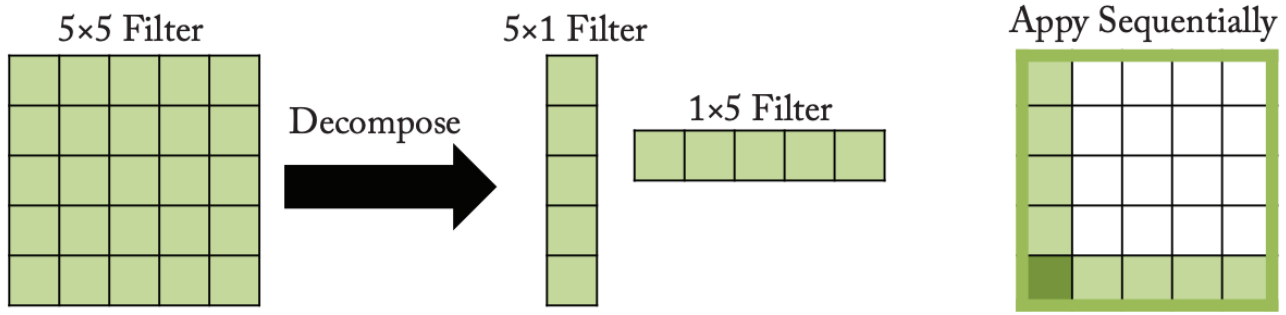
(b) Constructing a 5×5 support from 1×5 and 5×1 filter. Used in GoogLeNet/Inception v3 and v4.

$$Output\ Feature\ Map\ = \frac{(Input\ Feature\ Map - \ Filter\ +\ Stride + 2 \times padding)}{Stride}$$

$$Output\ Feature\ Map\ = \frac{(N-5\ +1\ )}{1} = N-4$$

$$Height(OFM) = \frac{(N-5\ +1\ )}{1} = N-4, Width(OFM) = Width(IFM)$$

Decomposing Filters (After applying the second filter)



(b) Constructing a 5×5 support from 1×5 and 5×1 filter. Used in GoogLeNet/Inception v3 and v4.

$$Output\ Feature\ Map\ = \frac{(Input\ Feature\ Map - \ Filter\ +\ Stride + 2 \times padding)}{Stride}$$

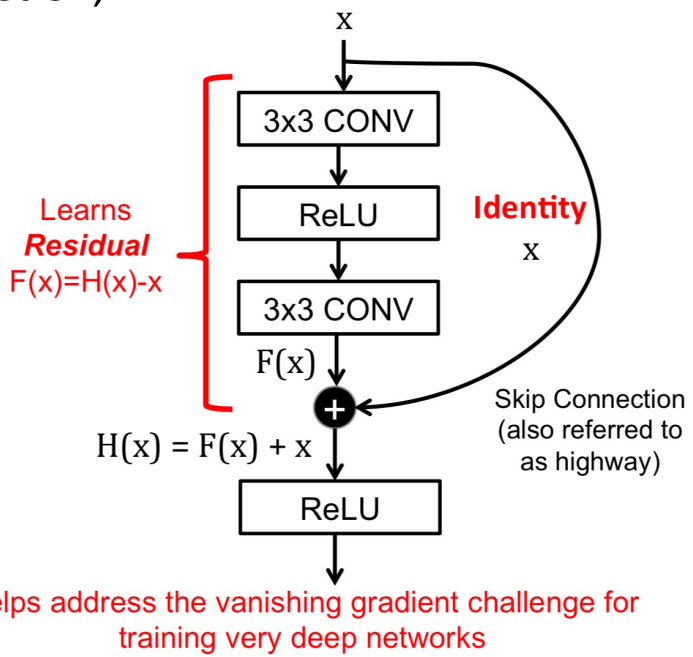
$$Output\ Feature\ Map\ = \frac{(N-5\ +1\ )}{1} = N-4$$

$$Height(OFM) = \frac{(N-5\ +1\ )}{1} = N-4, Width(OFM) = Width(IFM)$$

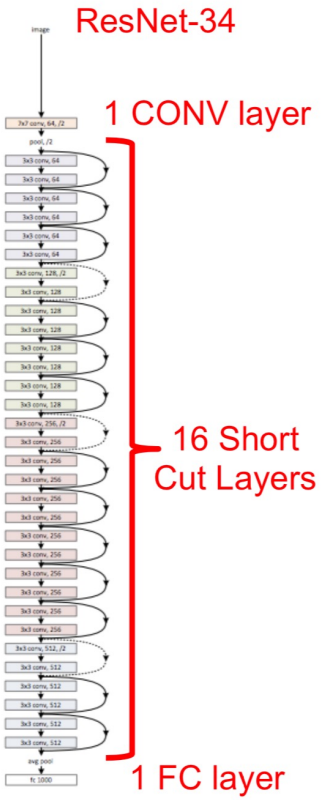
$$Width(OFM) = \frac{(N-5\ +1\ )}{1} = N-4$$

ResNet: Deep Residual Learning

- **Year**: Introduced in 2015 by Kaiming He and his team at Microsoft Research.
- **Deep Residual Learning**: Addresses the vanishing gradient problem by introducing skip (residual) connections that allow gradients to flow through the network.
- **Variants**: Comes in multiple depths like ResNet-18, ResNet-34, ResNet-50, ResNet-101, and ResNet-152.



[He et al., arXiv 2015, CVPR 2016]

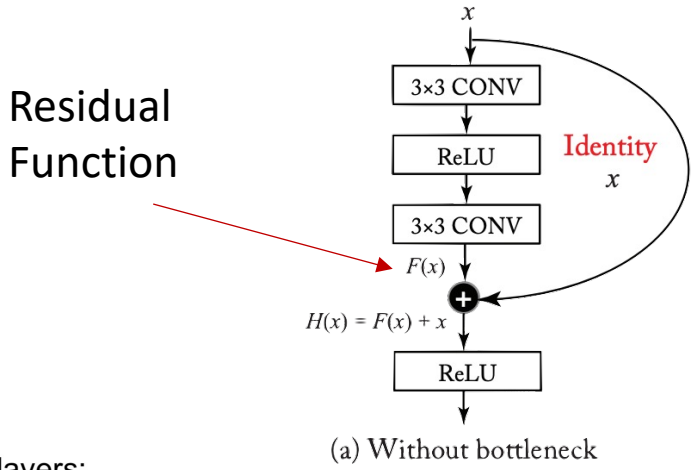


During backpropagation, because of the skip connections, the gradient can be directly propagated back to earlier layers without passing through all the intermediate layers. This reduces the risk of the gradient becoming infinitesimally small, combating the vanishing gradient problem.

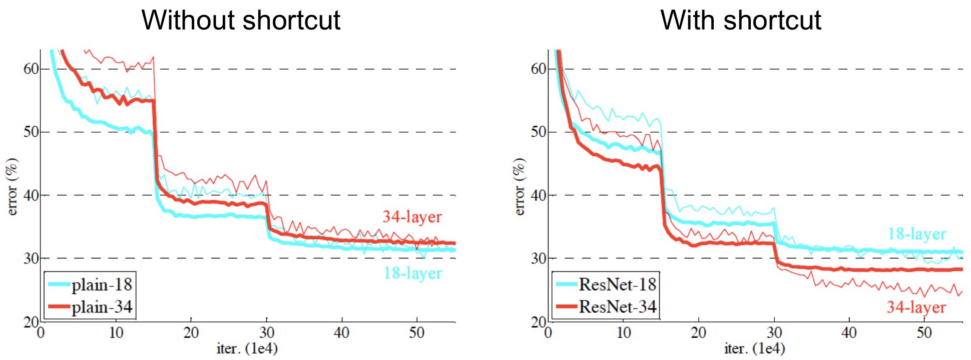
ResNet: Deep Residual Learning

Why it Matters?

- Winner of the ImageNet competition in 2015.
  - Exceeded human-level accuracy with a Top5 error rate below 5%.
- Residual blocks help combat the degradation problem, where accuracy plateaus and then degrades with the network depth.



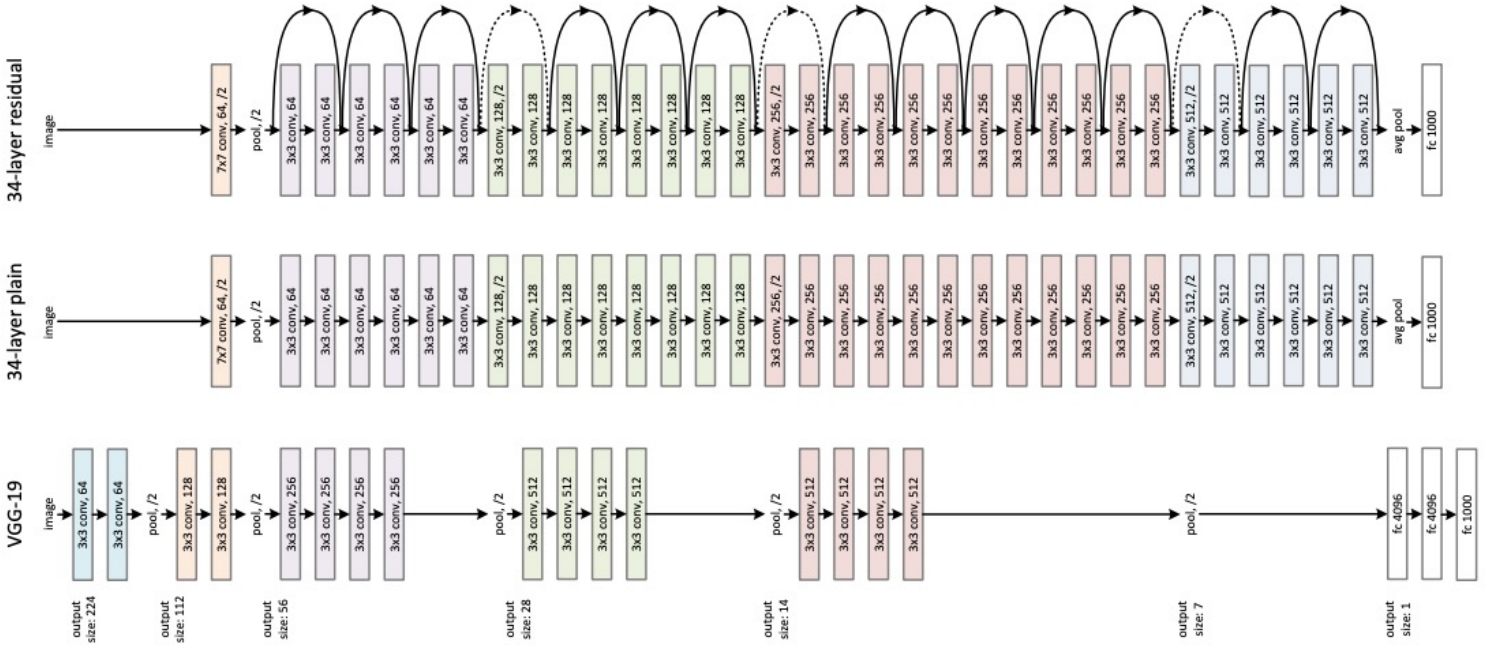
Training and validation error **increases** with more layers;
this is due to vanishing gradient, no overfitting.
Introduce **short cut module** to address this!



Thin curves denote training error, and bold curves denote validation error.

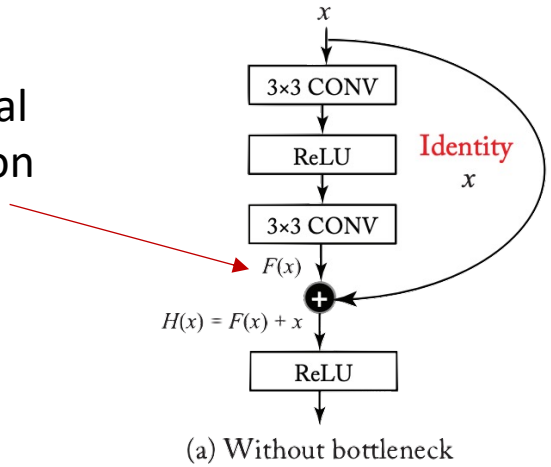
source: [He et al., Arxiv 2015, CVPR 2016]

ResNet: Deep Residual Learning



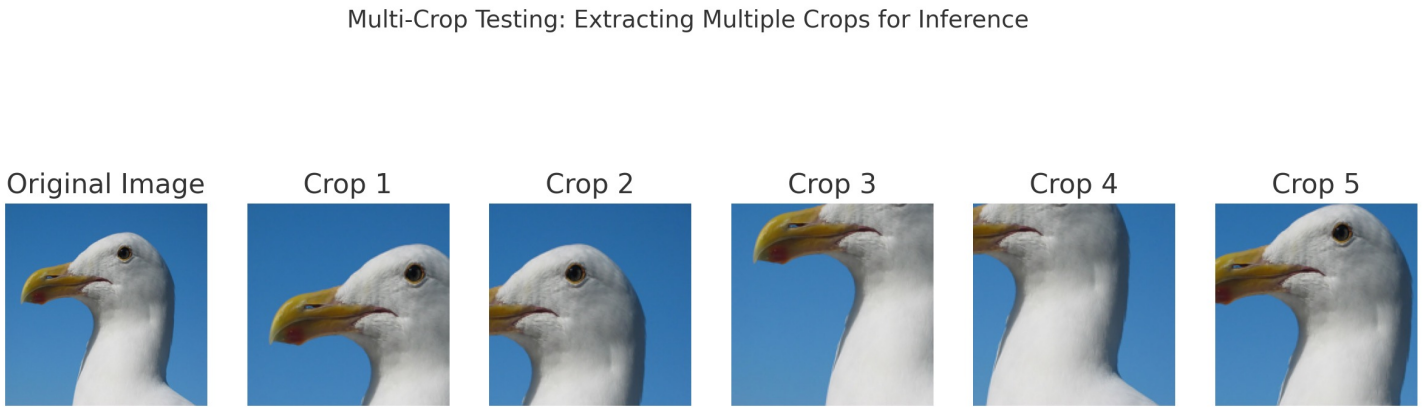
Source: Kaiming He et al.

Residual Function



(a) Without bottleneck

Single-Crop vs Multi-Crop



- **Single Crop** means evaluating the model on **one central crop** of the image.
- This is in contrast to **multi-crop testing**
- **Multi-Crop Testing** is a technique used to improve classification accuracy during inference by applying **multiple transformations (crops, flips, scales, rotations)** to the input image and averaging the predictions.

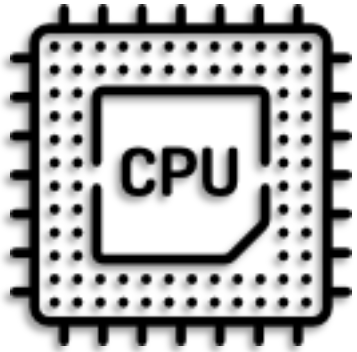
- **Depth vs. Accuracy:** Deeper networks generally yield higher accuracy.
- **Filter Flexibility:** There's a variation in filter shapes across layers, emphasizing the importance of flexibility.
- **Computation Allocation:** Most computational efforts are directed towards CONV (Convolutional) layers rather than FC (Fully Connected) layers.
- **Weight Distribution:**
  - Decrease in weights in FC layers over time.
  - In recent networks (e.g., since GoogLeNet), CONV layers also dominate in terms of weights.

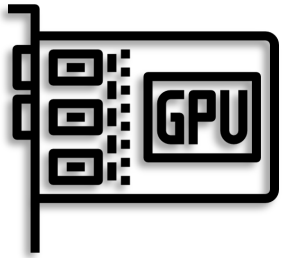
Metrics	LeNet 5	AlexNet	Overfeat Fast	VGG 16	GoogLeNet V1	ResNet 50
Top-5 error†	n/a	16.4	14.2	7.4	6.7	5.3
Top-5 error (single crop)†	n/a	19.8	17.0	8.8	10.7	7.0
Input size	28×28	227×227	231×231	224×224	224×224	224×224
Number of CONV layers	2	5	5	13	57	53
Depth in number of CONV layers	2	5	5	13	21	49
Filter sizes	5	3, 5, 11	2, 5, 11	3	1, 3, 5, 7	1, 3, 7
Number of channels	1, 20	3–256	3–1,024	3–512	3–832	3–2,048
Number of filters	20, 50	96–384	96–1,024	64–512	16–384	64–2,048
Stride	1	1, 4	1, 4	1	1, 2	1, 2
Weights	2.6 k	2.3 M	16 M	14.7 M	6.0 M	23.5 M
MACs	283 k	666 M	2.67 G	15.3 G	1.43 G	3.86 G
Number of FC layers	2	3	3	3	1	1
Filter sizes	1, 4	1, 6	1, 6, 12	1, 7	1	1
Number of channels	50, 500	256–4,096	1,024–4,096	512–4,096	1,024	2,048
Number of filters	10, 500	1,000–4,096	1,000–4,096	1,000–4,096	1,000	1,000
Weights	58 k	58.6 M	130 M	124 M	1 M	2 M
MACS	58 K	58.6 M	130 M	124 M	1 M	2 M
Total weights	60 k	61 M	146 M	138 M	7 M	25.5 M
Total MACs	341 k	724 M	2.8 G	15.5 G	1.43 G	3.9 G
Pretrained model website	[77]‡	[78, 79]	n/a	[78, 79, 80]	[78, 79, 80]	[78, 79, 80]

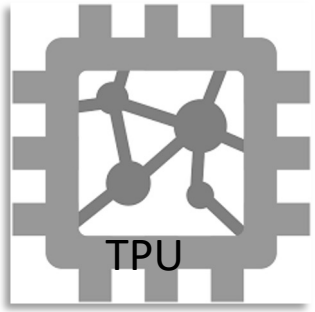
- **Hardware Implementation Focus:**
  - Emphasis should be on optimizing CONV layers.
  - CONV layers are becoming increasingly vital in various domains.

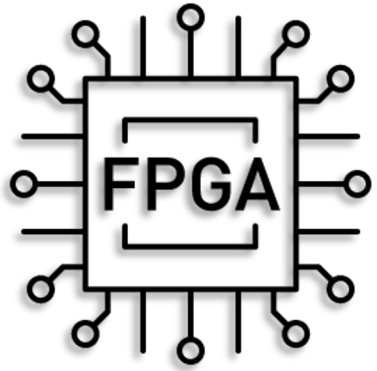
Accelerators for DNNs

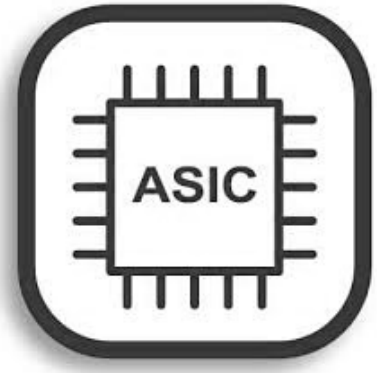
Hardware Types











More Flexible

More Efficient

How To Evaluate DNNs?

Metrics for Evaluation of DNNs

Performance

- Throughput
- Latency

Power

- At the edge: low battery consumption
- In the could: high cost of cooling

Accuracy

- How accurate is the result?

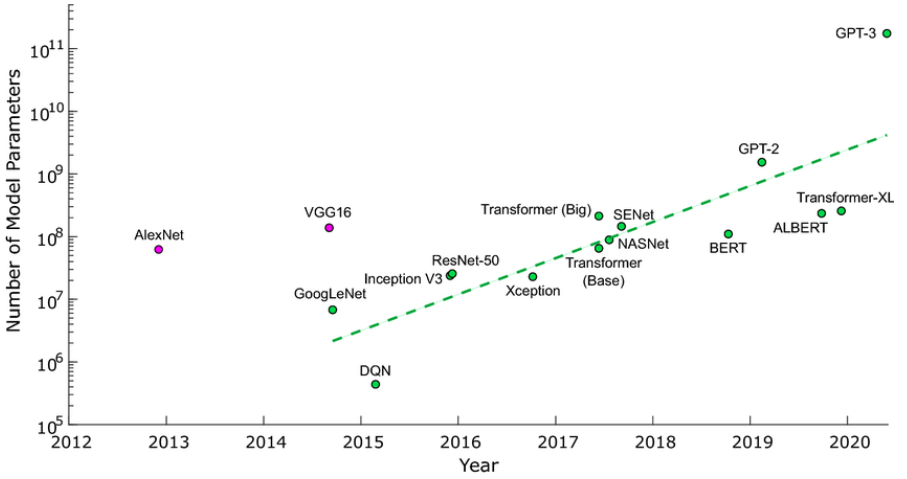
Flexibility

- That tasks can it do?

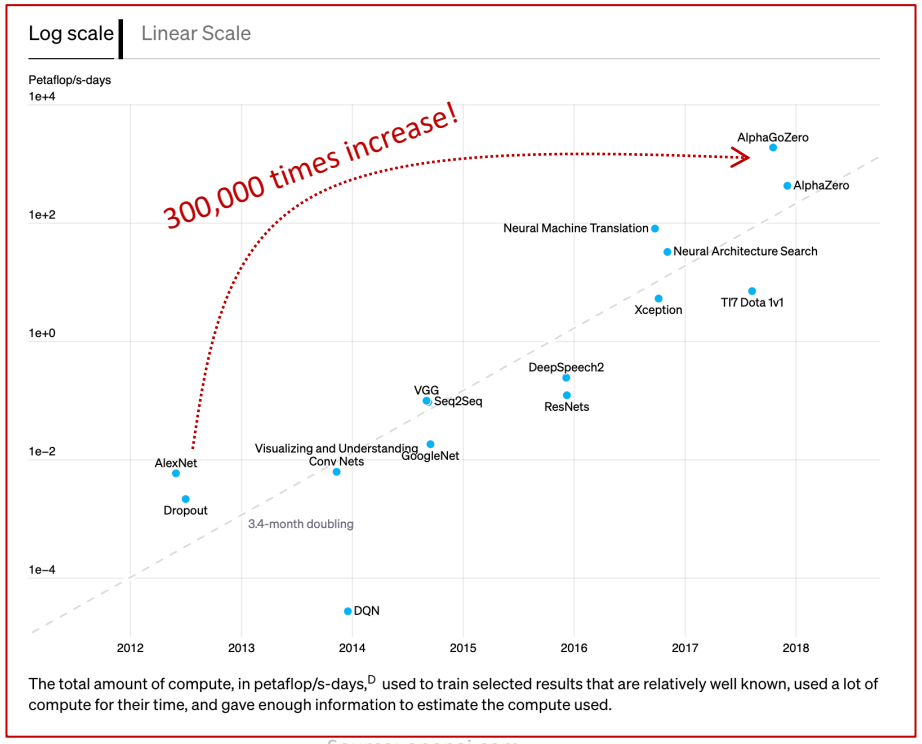
Scalability

DNN Metrics

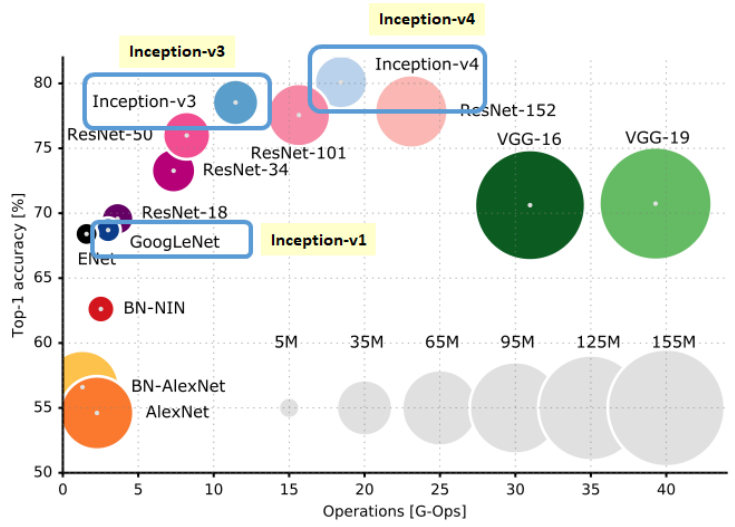
- Number of operations needed to run
  - Training
  - Inference
- Memory
  - Size of parameters
  - Size of activations
- Accuracy



Source: Liane Bernstein's et al.

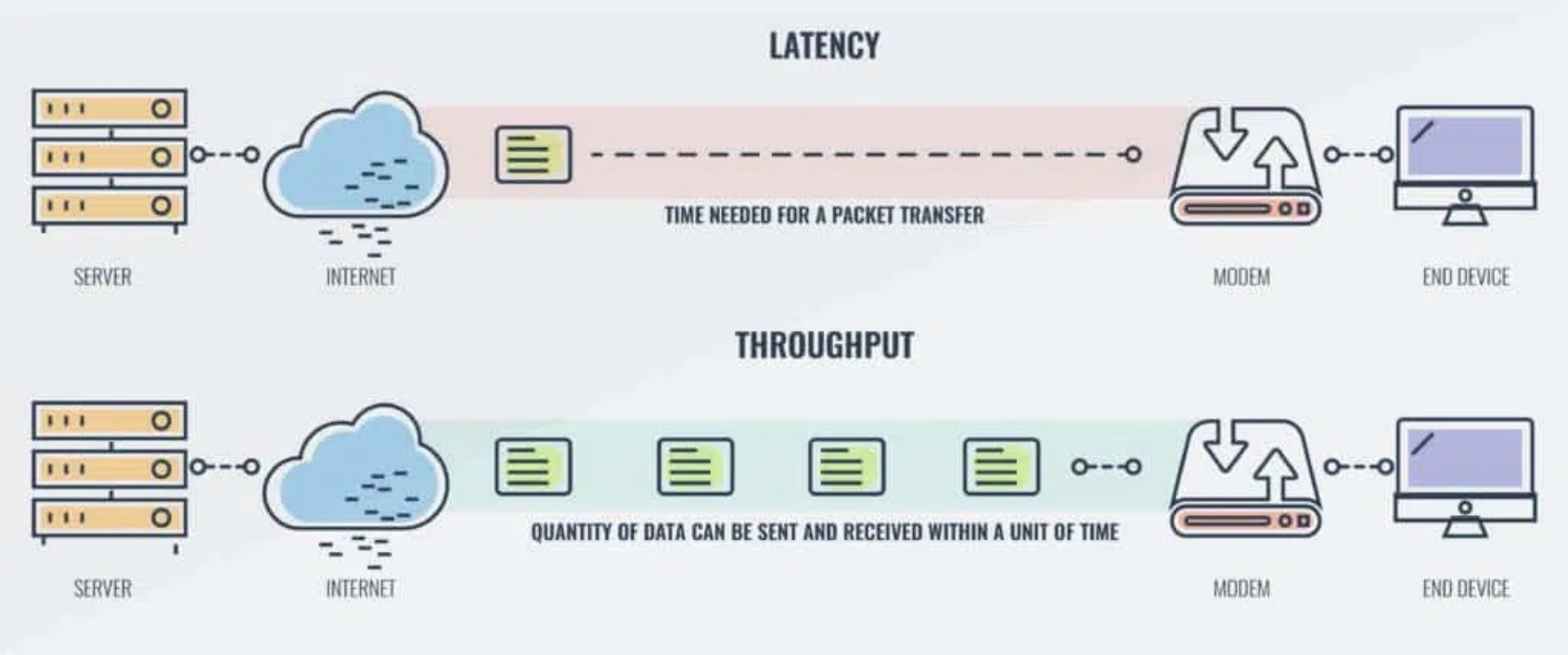


Source: openai.com



Source: Alfredo Canziani et al.

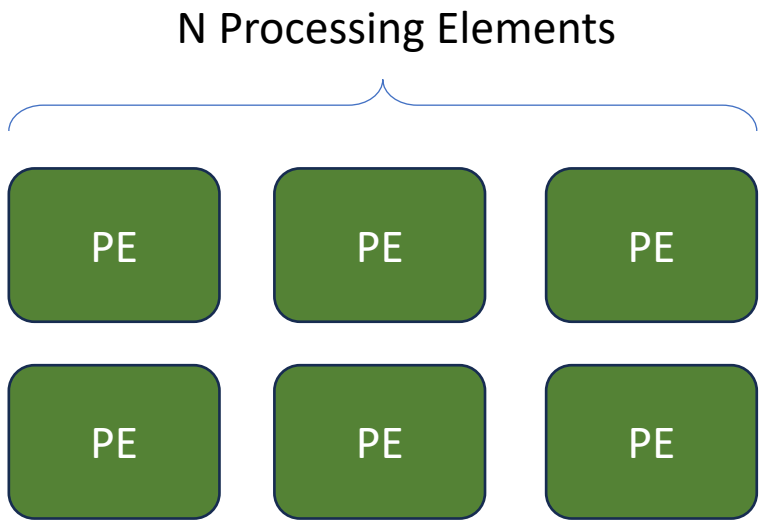
Performance



Source: comparitech.com

Computation Performance Metrics

- Throughput:
  - OP/S: Operations per second
    - GOP/S, TOP/S
    - Very inaccurate
  - FLOP/S
    - Floating Points Operations Per Second
    - Peak FLOPS
- Power Consumption
  - GOPs/Watt



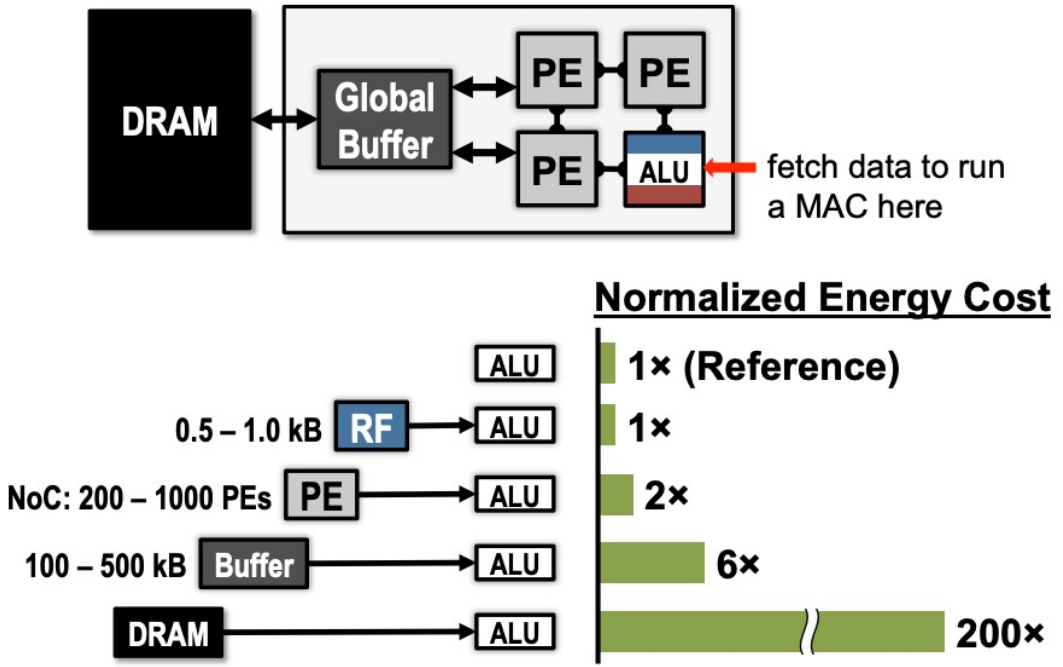
$$Peak\ OPS = OPS_{PE} \times N$$

Real-world application speed can differ due to factors like memory bandwidth, I/O operations, and specific DNN architectures.

Memory Performance Metrics

- Capacity: GB or MB
- Bandwidth: GB/s
  - READ/WRITE operations per second

- **Memory hierarchy**
  - **Large Global Buffer**:
    - Size: Several hundred kilobytes.
    - Connection: Directly to DRAM.
  - **Inter-PE Network**:
    - Function: Facilitates direct data passage between ALUs.
  - **Register File (RF) within each Processing Element (PE)**:
    - Size: A few kilobytes or less.

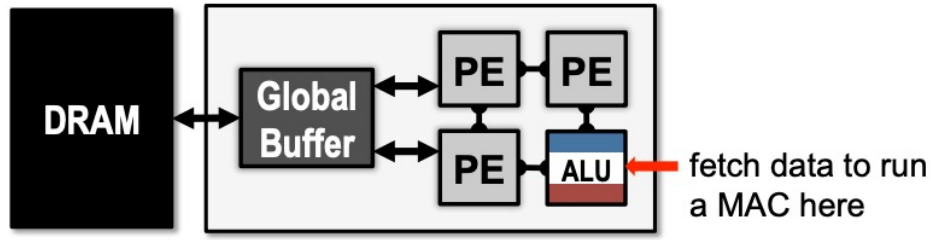


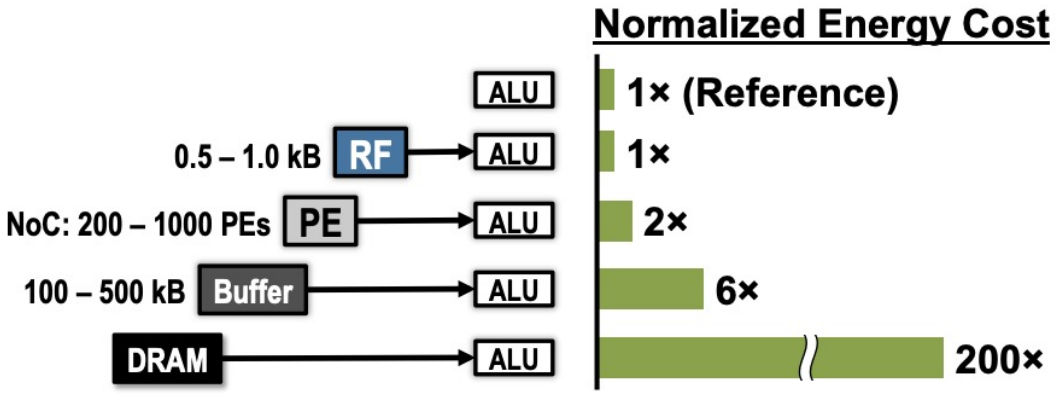
Memory hierarchy and data movement energy

Source: Vivienne Sze et al.

Memory Performance Metrics

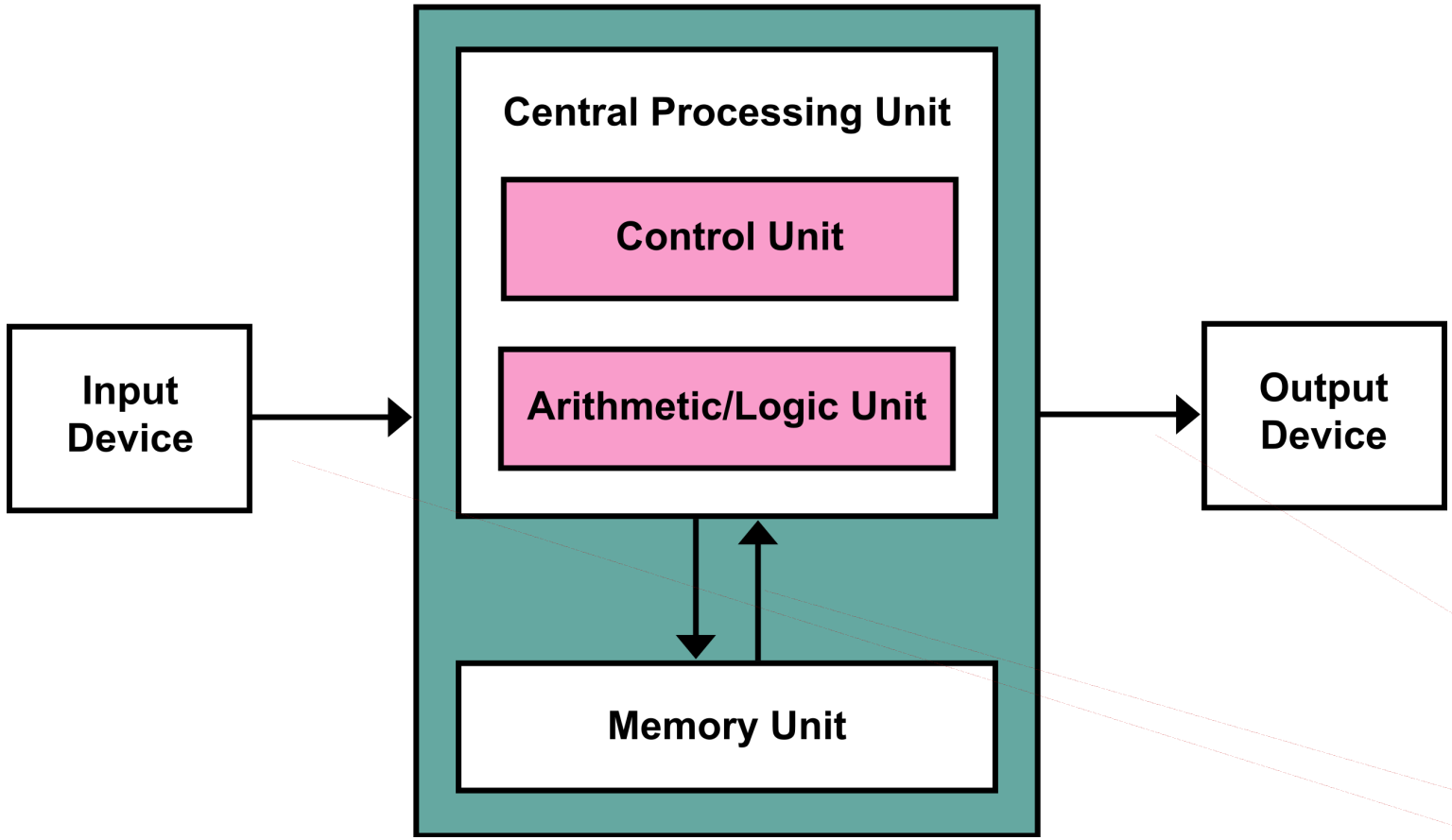
- **Memory Hierarchy Benefits**:
  - Enhances energy efficiency.
  - Provides low-cost data accesses.
- **Energy Cost Comparisons**:
  - Fetching from RF or neighbor PEs: Very energy efficient.
  - Cost is 1 or 2 orders of magnitude lower than fetching from DRAM.



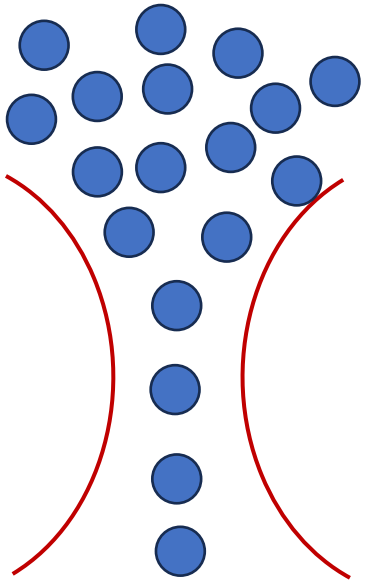


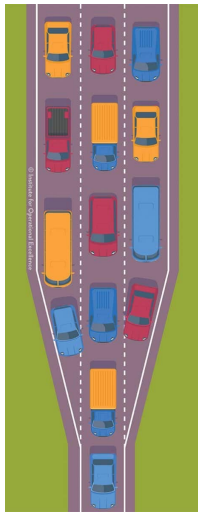
Source: Vivienne Sze et al.

Von Neumann Architecture



Source: Wikipedia

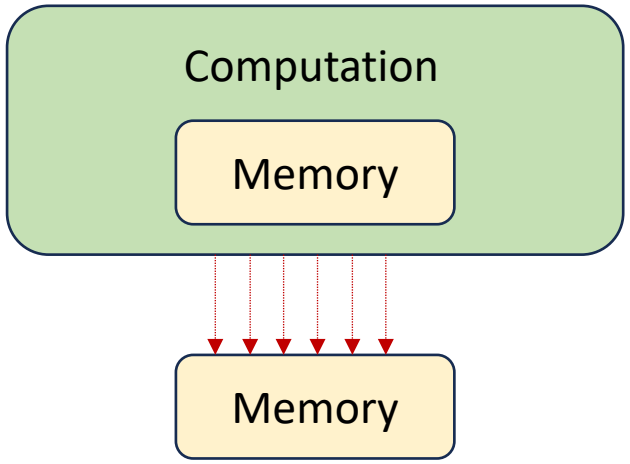




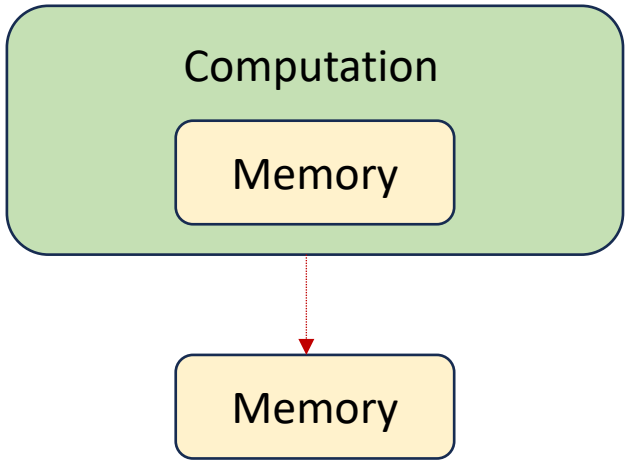
Bottleneck

Memory vs Computation Bound

- **Memory-Bound**: memory accesses time dominates computation time.
- **Characteristics**:
  - Data is not available in cache or local memory, leading to DRAM fetches.
  - Limited by memory bandwidth.
  - Seen in applications like database operations, stream processing.



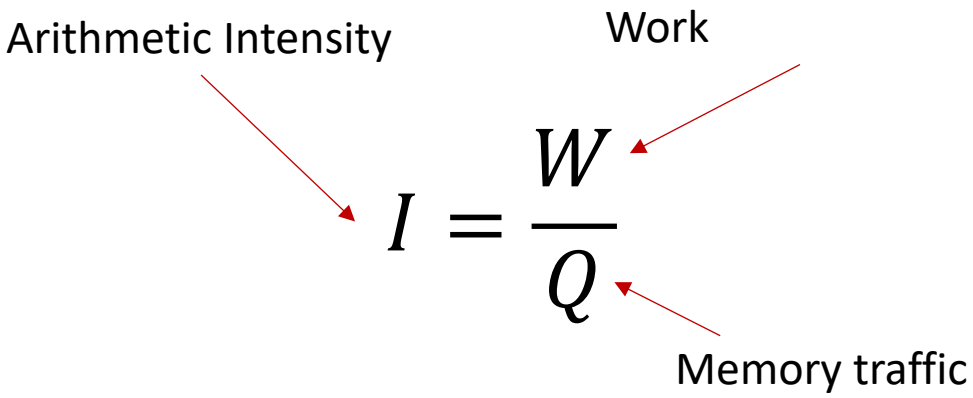
- **Compute-Bound**: computation time dominates memory access time.
- **Characteristics**:
  - Intensive arithmetic or logical operations.
  - Data is readily available, minimal waiting on memory accesses.
  - Limited by processing power.
- E.g. scientific simulations.



Arithmetic Intensity

- **W: Work (FLOPs)**
  - Number of operations performed. (Not to be confused with FLOP**S**)
- **Q: Memory Traffic (Bytes/s)**
  - Number of bytes of memory transfers during execution.
  - Directly impacted by platform properties, especially cache hierarchy.
- **I: Arithmetic Intensity (FLOPs/byte)**
  - Ratio of work (W) to memory traffic (Q).
  - Represents operations per byte of memory traffic.

Measure of data locality (reuse of data)



Roofline Model

Measure of data locality (reuse of data)

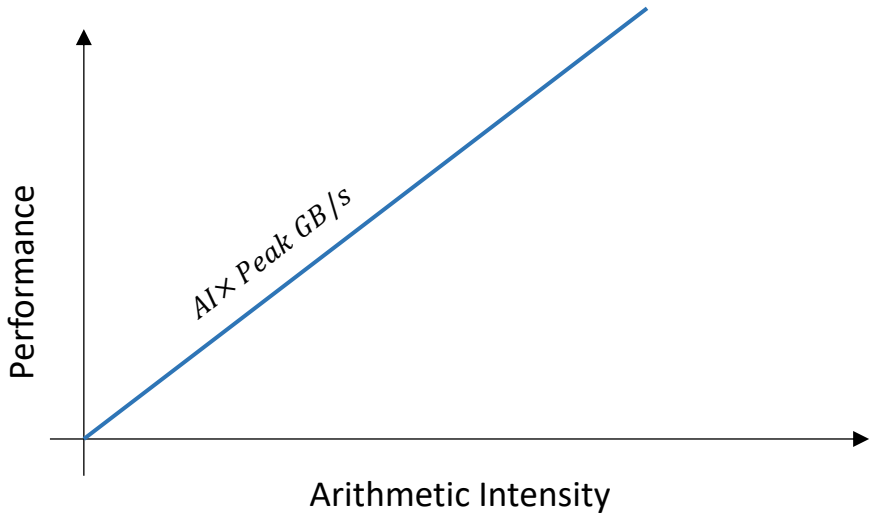
Peak Performance

Performance

$$GFLOP/s = \min \begin{cases} Peak\ GFLOP/s \\ AI \times Peak\ GB/s \end{cases}$$

Arithmetic Intensity

Peak memory bandwidth



Arithmetic Intensity

Work

$$I = \frac{W}{Q}$$

Memory traffic

Roofline Model

Measure of data locality (reuse of data)

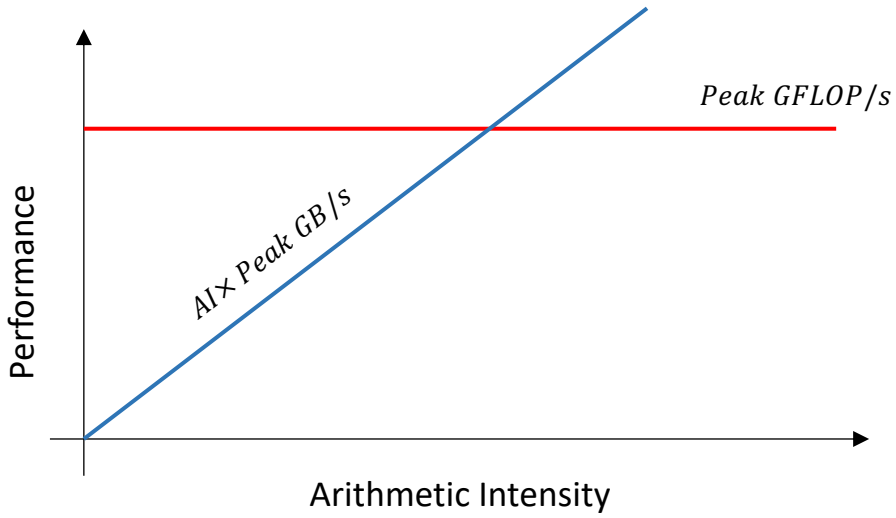
Peak Performance

Performance

$$GFLOP/s = \min \begin{cases} Peak\ GFLOP/s \\ AI \times Peak\ GB/s \end{cases}$$

Arithmetic Intensity

Peak memory bandwidth



Roofline Model

Measure of data locality (reuse of data)

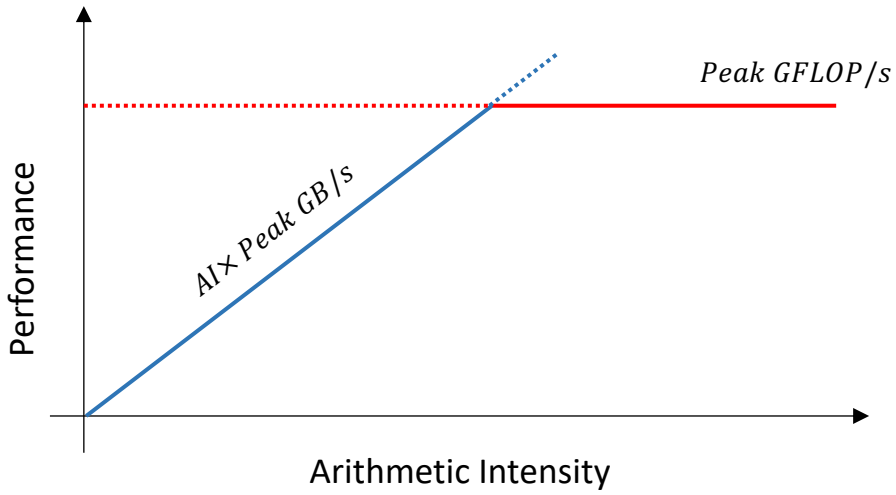
Peak Performance

Performance

$$GFLOP/s = \min \begin{cases} Peak\ GFLOP/s \\ AI \times Peak\ GB/s \end{cases}$$

Arithmetic Intensity

Peak memory bandwidth



Roofline Model

Measure of data locality (reuse of data)

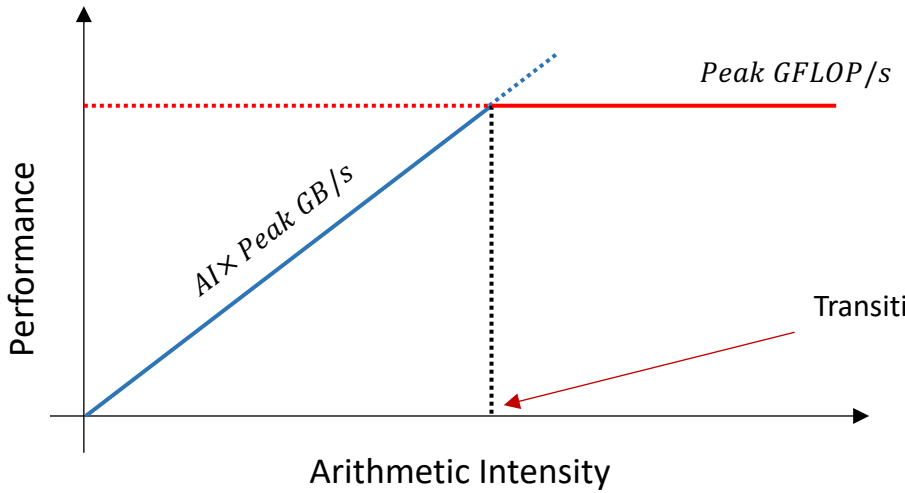
Peak Performance

Performance

$$GFLOP/s = \min \begin{cases} Peak\ GFLOP/s \\ AI \times Peak\ GB/s \end{cases}$$

Arithmetic Intensity

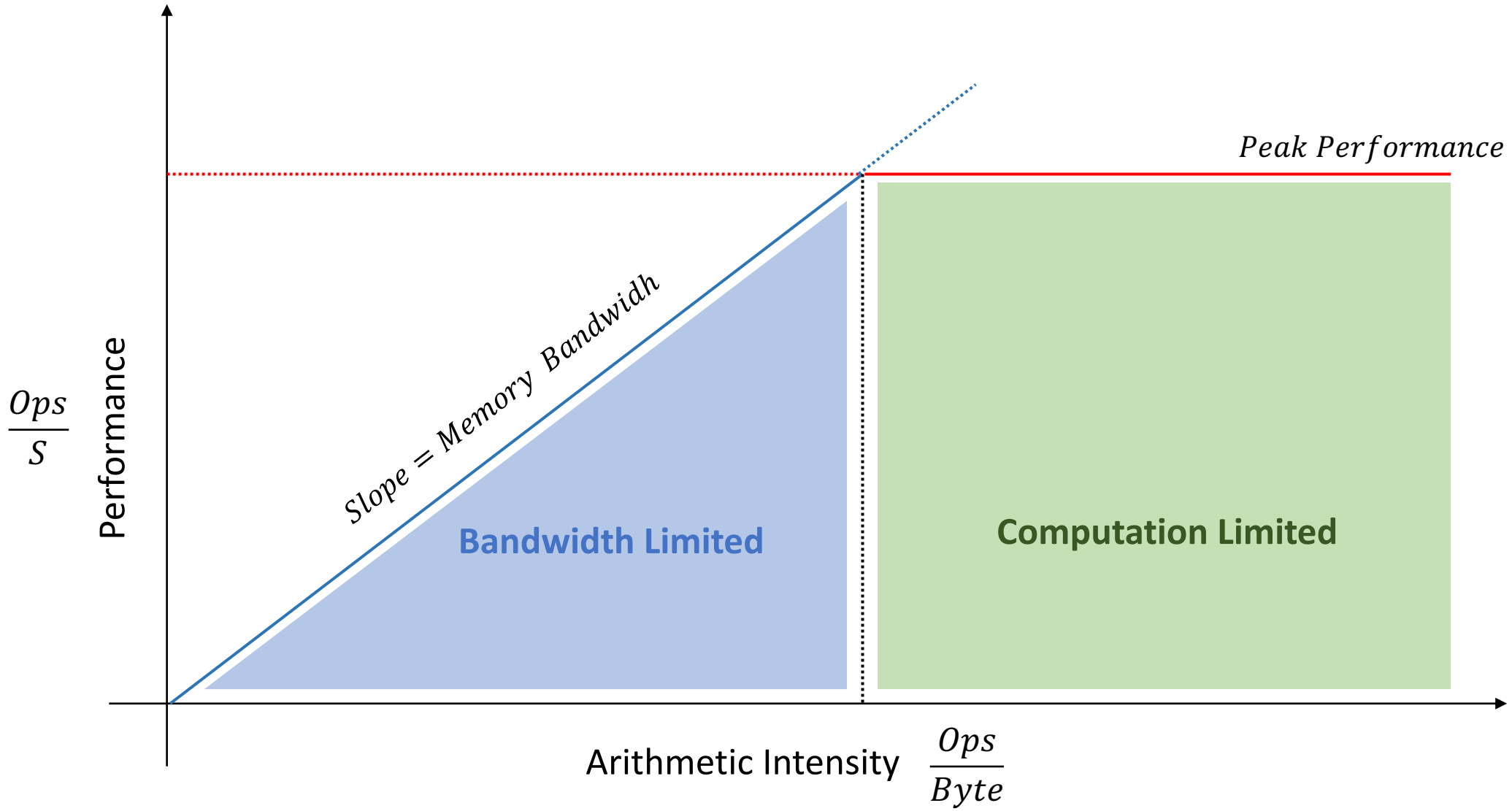
Peak memory bandwidth



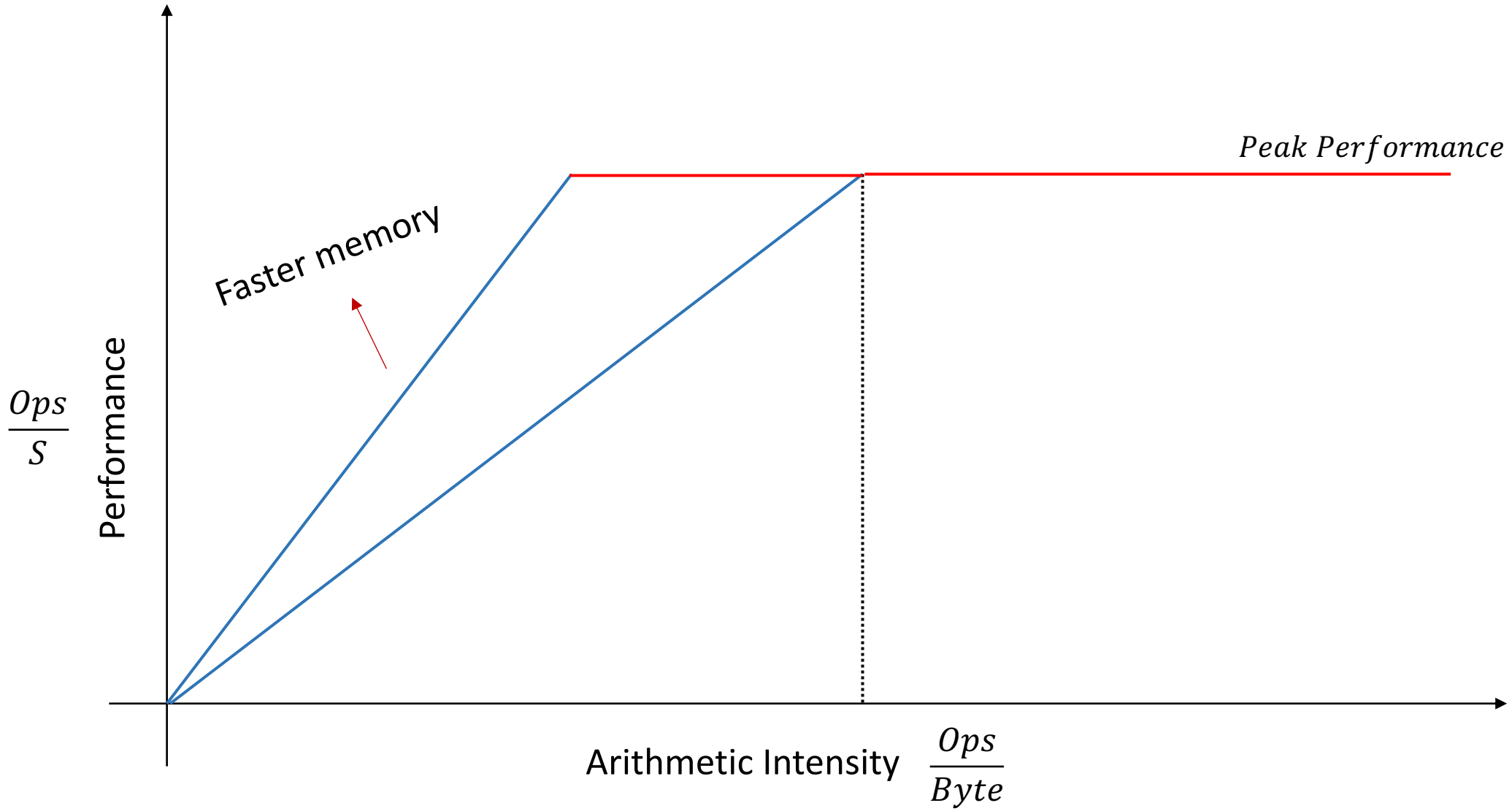
Transition Point $AI = \frac{Peak\ GFLOP/s}{Peak\ GB/s}$

Minimum AI to achieve Max performance

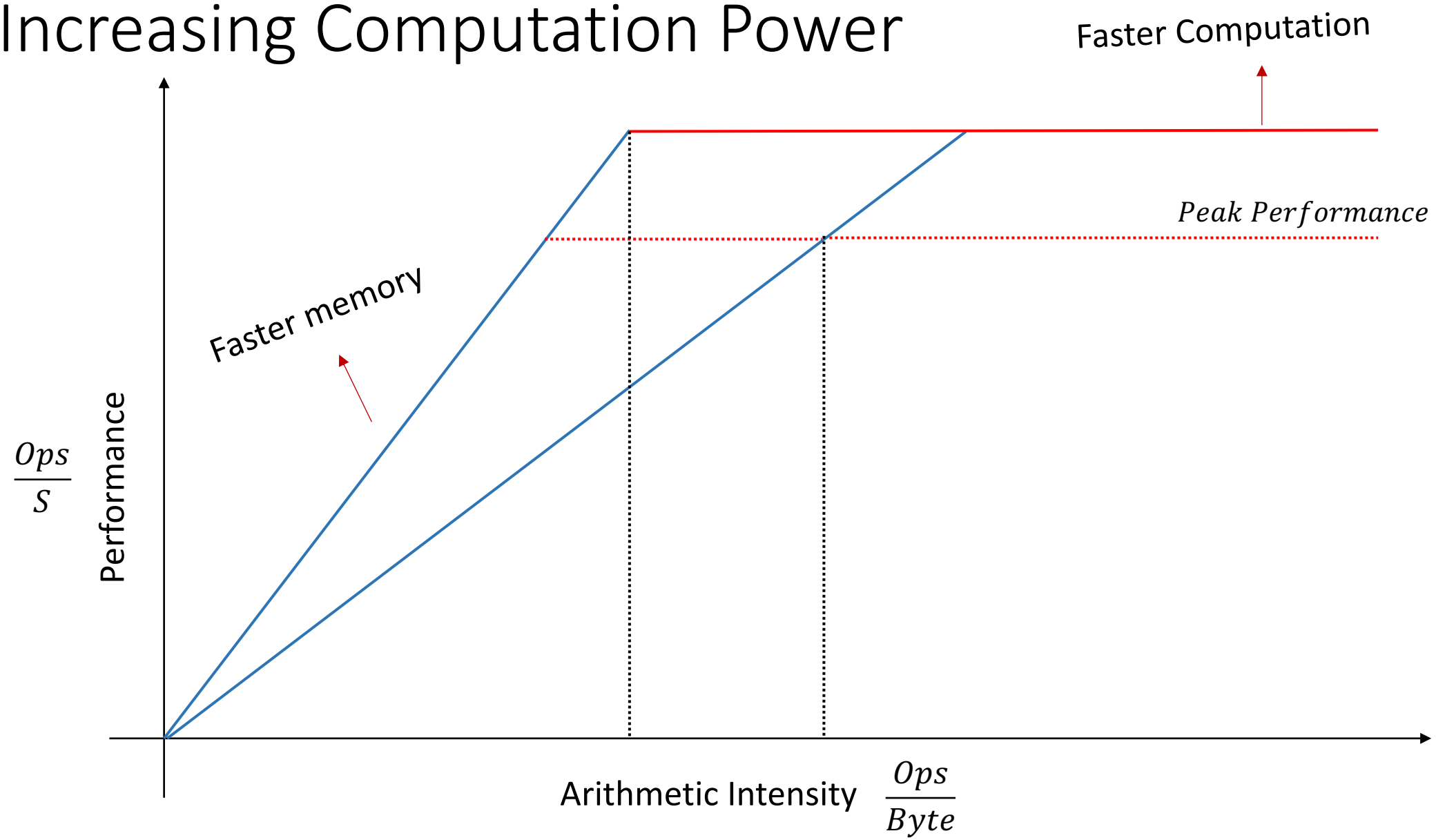
Roofline Model Operation Regions

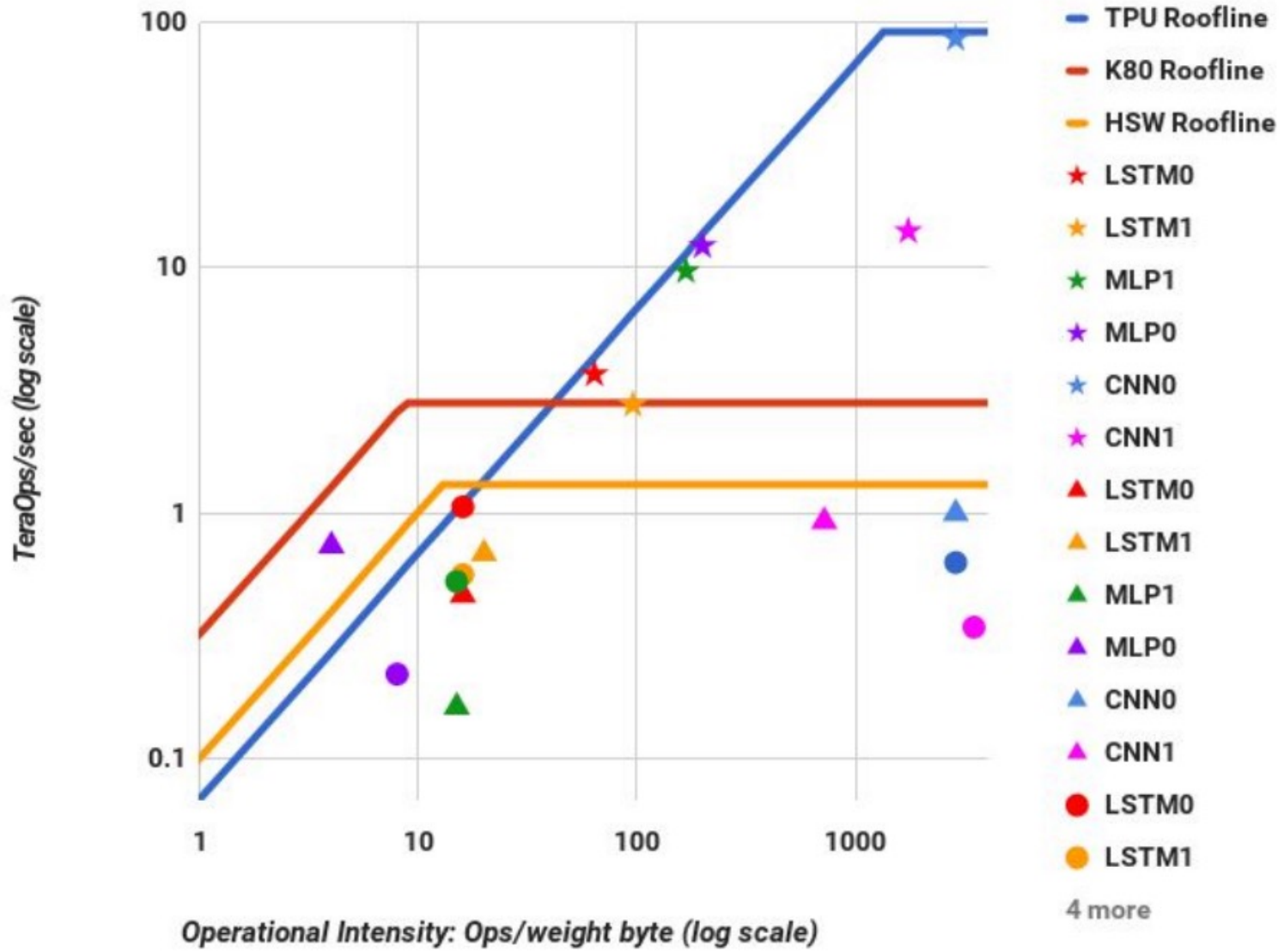


Increasing Memory Bandwidth



Increasing Computation Power



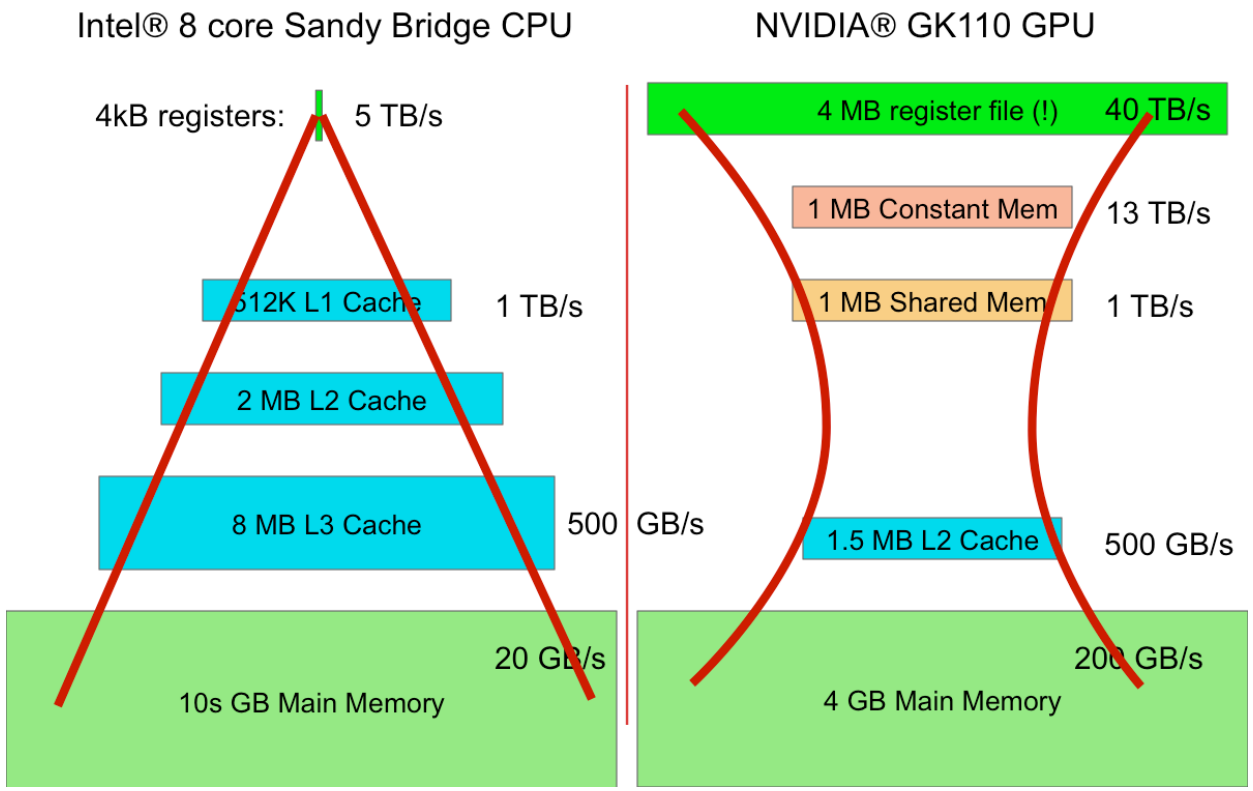


CPU vs. GPU vs. TPU Roofline

Source: Norman P.Jouppi et al. Google

Memory Hierarchy

Feature/Aspect	CPU	GPU
Memory Varieties	Memory mainly provisioned as **caches**.	**Multiple** forms of memory available.
	Not directly accessible by programmers.	Directly accessible, apt for diverse scenarios.

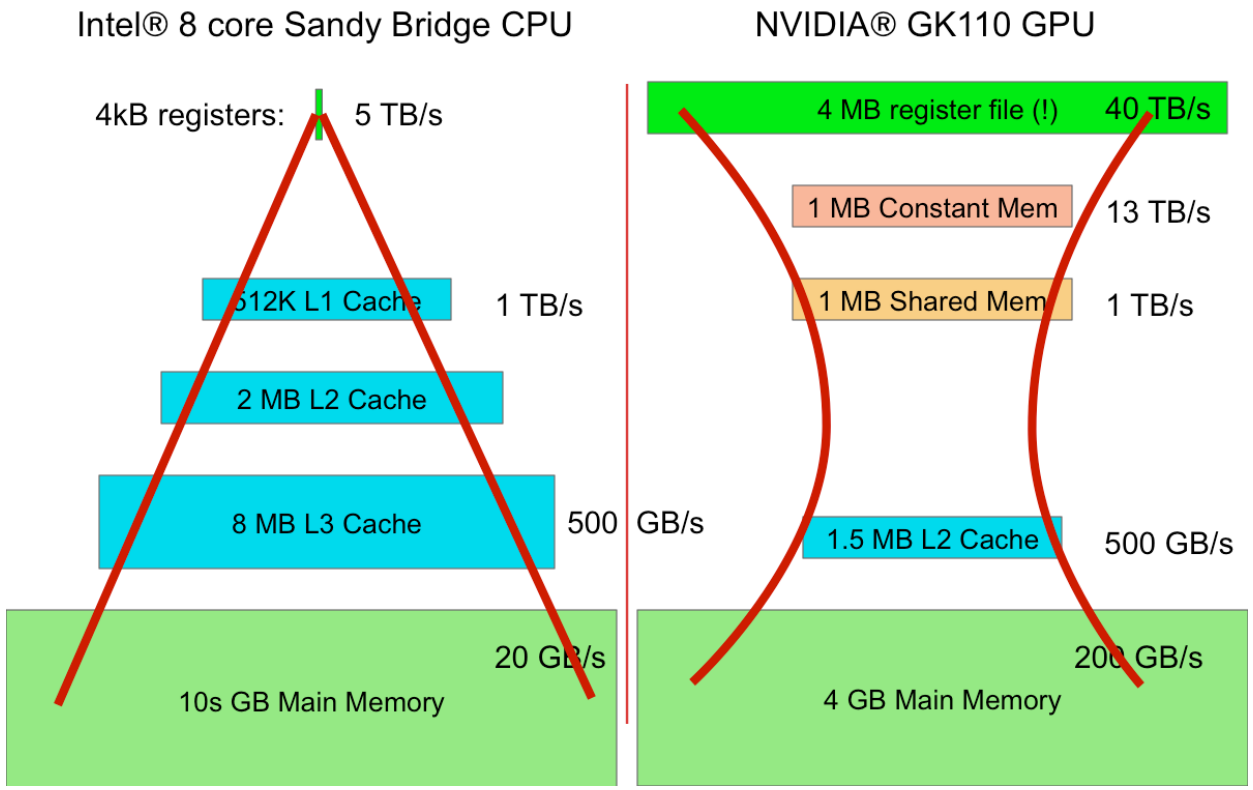


Source: NVIDIA

GPU's memory structure can drive remarkable speedups in specific operations when optimally utilized.

Memory Hierarchy

Feature/Aspect	CPU	GPU
Memory Varieties	Memory mainly provisioned as **caches**.	**Multiple** forms of memory available.
	Not directly accessible by programmers.	Directly accessible, apt for diverse scenarios.
Complexity of Access	Leads to complex code due to indirect memory access.	Simplified due to direct memory access.

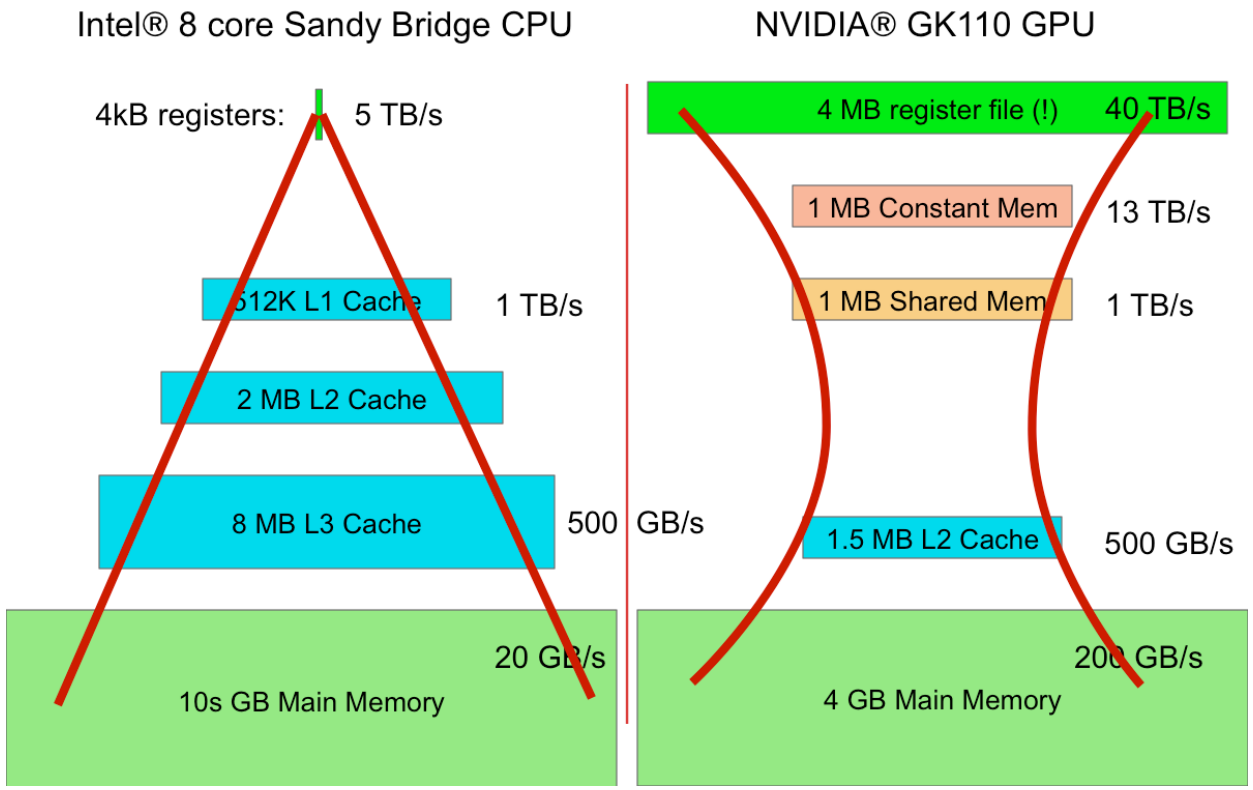


Source: NVIDIA

GPU's memory structure can drive remarkable speedups in specific operations when optimally utilized.

Memory Hierarchy

Feature/Aspect	CPU	GPU
Memory Varieties	Memory mainly provisioned as **caches**.	**Multiple** forms of memory available.
	Not directly accessible by programmers.	Directly accessible, apt for diverse scenarios.
Complexity of Access	Leads to complex code due to indirect memory access.	Simplified due to direct memory access.
Register Storage	KB	Significant amount (e.g., 4MB in NVIDIA GPUs).

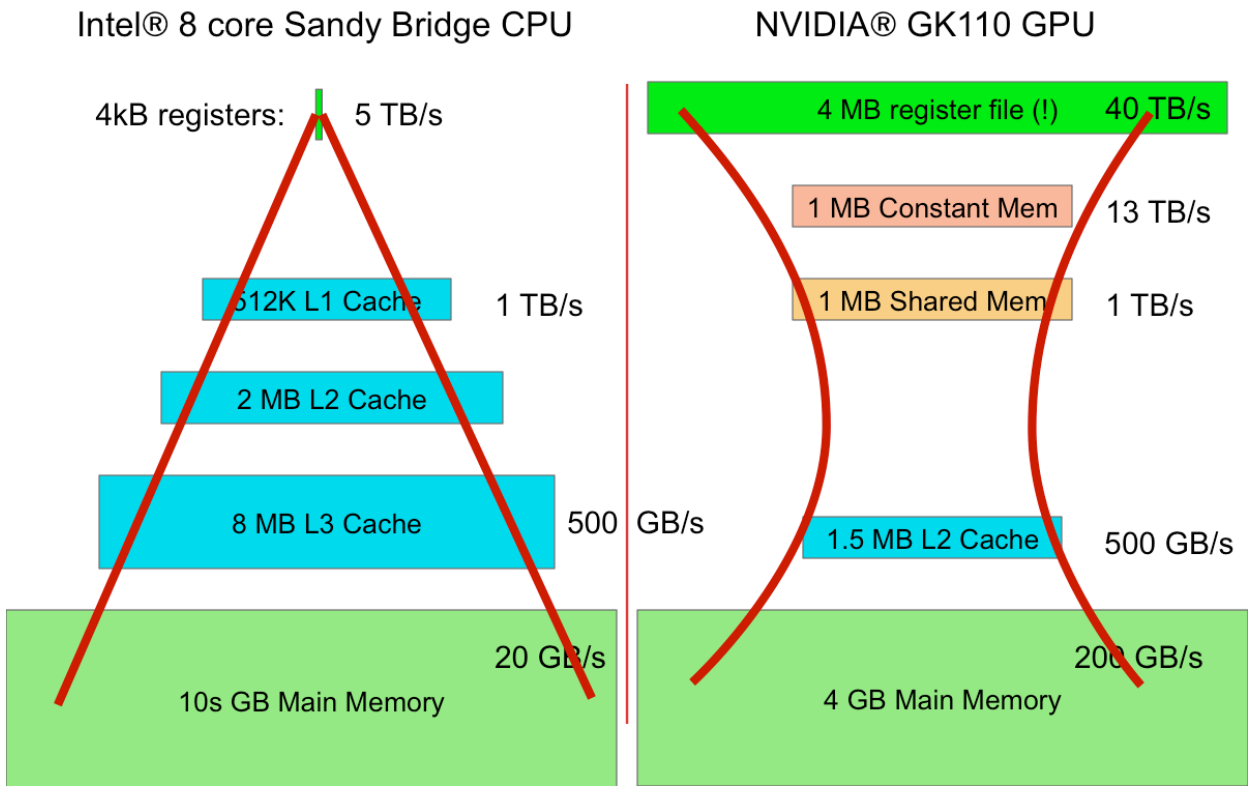


Source: NVIDIA

GPU's memory structure can drive remarkable speedups in specific operations when optimally utilized.

Memory Hierarchy

Feature/Aspect	CPU	GPU
Memory Varieties	Memory mainly provisioned as **caches**.	**Multiple** forms of memory available.
	Not directly accessible by programmers.	Directly accessible, apt for diverse scenarios.
Complexity of Access	Leads to complex code due to indirect memory access.	Simplified due to direct memory access.
Register Storage	KB	Significant amount (e.g., 4MB in NVIDIA GPUs).
Performance with Registers	High but not often controlled by the programmer	Calculations relying on registers achieve full machine speed.

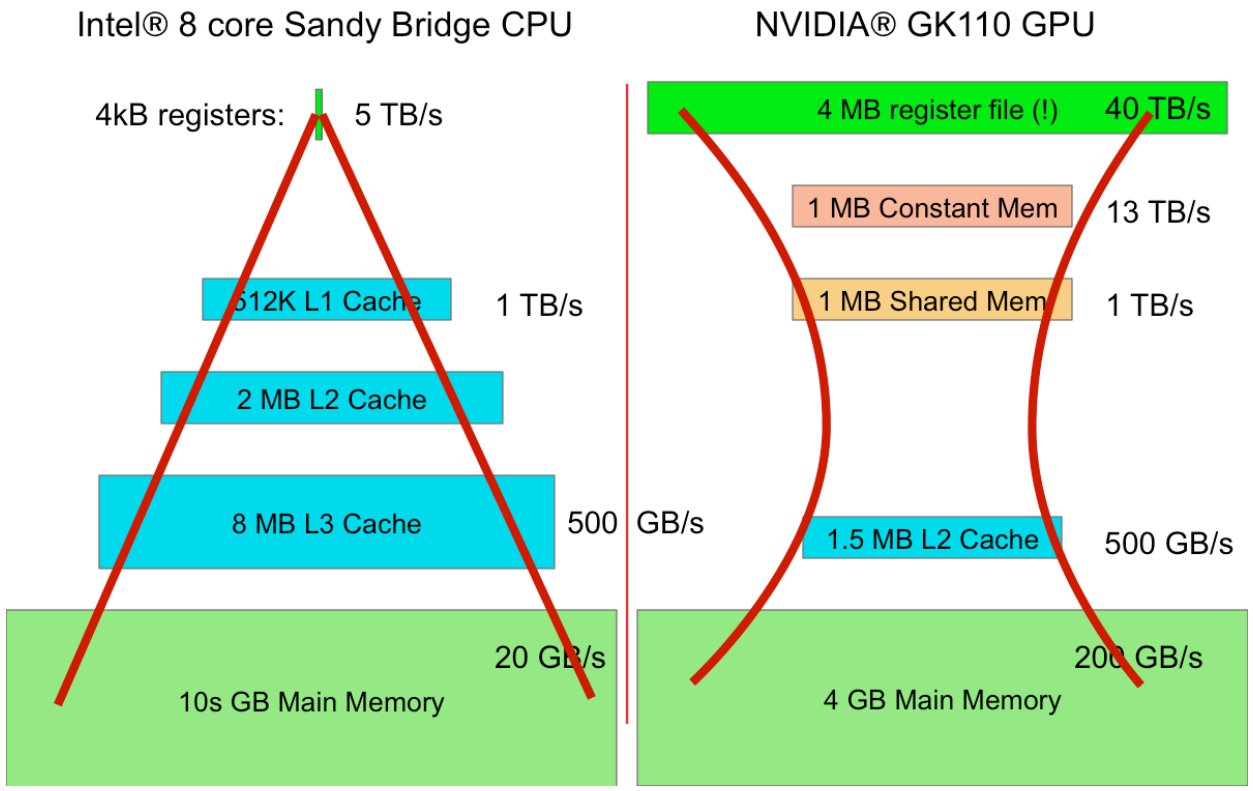


Source: NVIDIA

GPU's memory structure can drive remarkable speedups in specific operations when optimally utilized.

Memory Hierarchy

Feature/Aspect	CPU	GPU
Memory Varieties	Memory mainly provisioned as **caches**.	**Multiple** forms of memory available.
	Not directly accessible by programmers.	Directly accessible, apt for diverse scenarios.
Complexity of Access	Leads to complex code due to indirect memory access.	Simplified due to direct memory access.
Register Storage	KB	Significant amount (e.g., 4MB in NVIDIA GPUs).
Performance with Registers	High but not often controlled by the programmer	Calculations relying on registers achieve full machine speed.
Operational Benefits	Diverse operations	Vector operations



Source: NVIDIA

GPU's memory structure can drive remarkable speedups in specific operations when optimally utilized.

Metric	Hardware	DNN Model
Operations per inference		
Peak Performance		
Memory Bandwidth		
Operational Intensity		
Utilization of PEs		
Throughput		
Latency		

Metric	Hardware	DNN Model
Operations per inference		✓
Peak Performance		
Memory Bandwidth		
Operational Intensity		
Utilization of PEs		
Throughput		
Latency		

Metric	Hardware	DNN Model
Operations per inference		✓
Peak Performance	✓	
Memory Bandwidth		
Operational Intensity		
Utilization of PEs		
Throughput		
Latency		

Metric	Hardware	DNN Model
Operations per inference		✓
Peak Performance	✓	
Memory Bandwidth	✓	
Operational Intensity		
Utilization of PEs		
Throughput		
Latency		

Metric	Hardware	DNN Model
Operations per inference		✓
Peak Performance	✓	
Memory Bandwidth	✓	
Operational Intensity		✓
Utilization of PEs		
Throughput		
Latency		

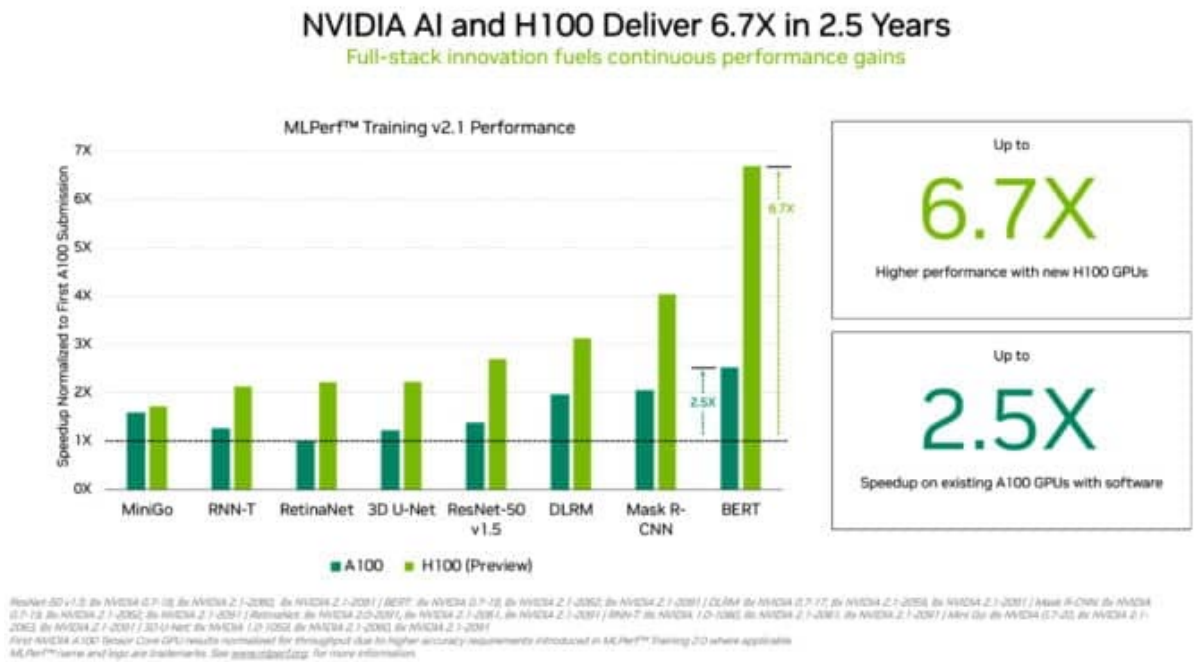
Metric	Hardware	DNN Model
Operations per inference		✓
Peak Performance	✓	
Memory Bandwidth	✓	
Operational Intensity		✓
Utilization of PEs	✓	✓
Throughput		
Latency		

Metric	Hardware	DNN Model
Operations per inference		✓
Peak Performance	✓	
Memory Bandwidth	✓	
Operational Intensity		✓
Utilization of PEs	✓	✓
Throughput	✓	✓
Latency		

Metric	Hardware	DNN Model
Operations per inference		✓
Peak Performance	✓	
Memory Bandwidth	✓	
Operational Intensity		✓
Utilization of PEs	✓	✓
Throughput	✓	✓
Latency	✓	✓

ML Pref

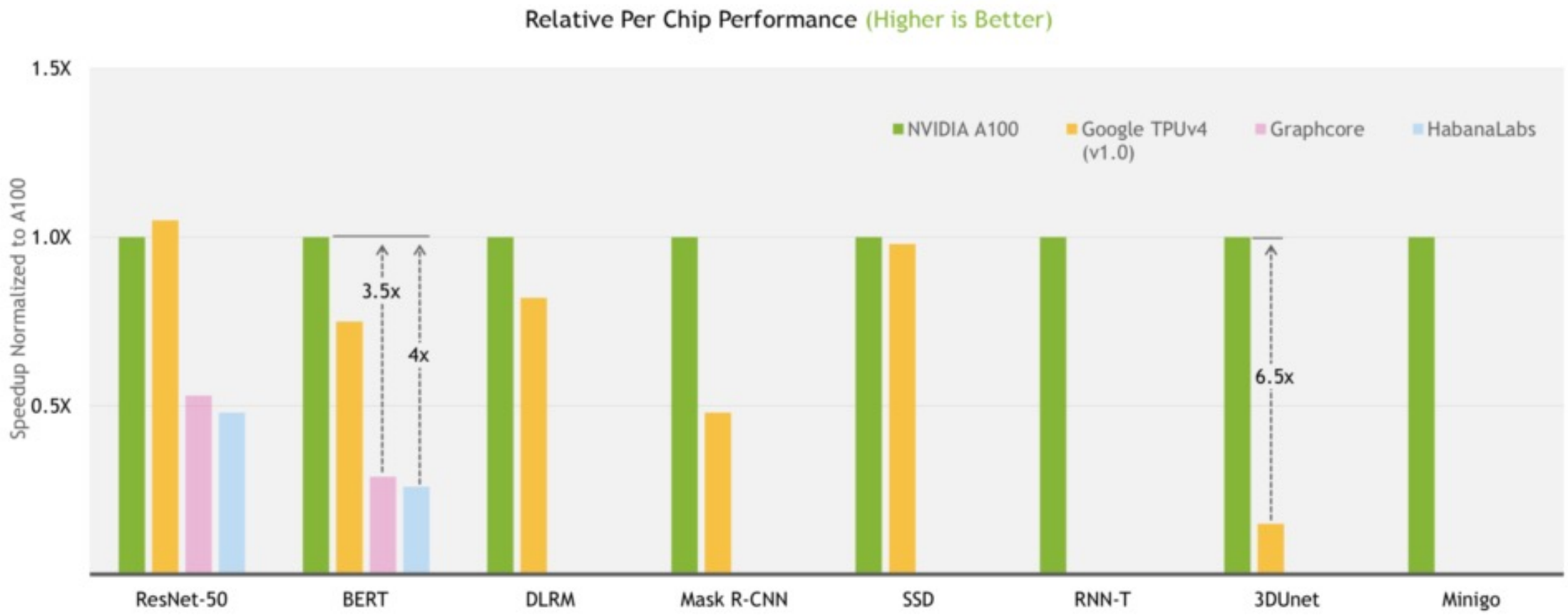
- MLPerf Benchmark Suite:
  - Collection of DNN models for fair evaluations across software frameworks, hardware accelerators, and platforms.
  - Inclusive of varied DNN types (CNN, RNN) and tasks (e.g., image classification, translation, sentiment analysis).



https://mlperf.org/

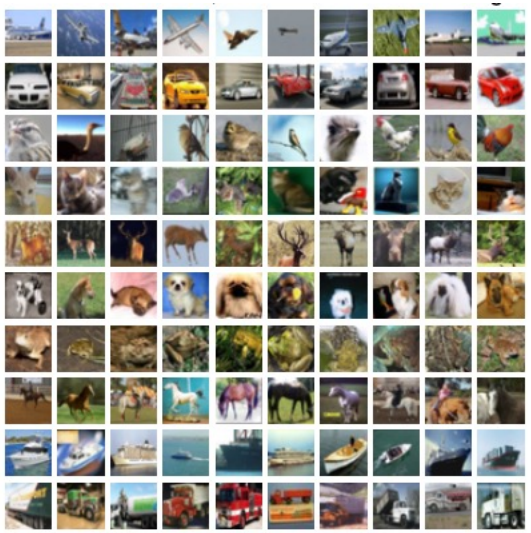
NVIDIA AI FASTEST PER-CHIP PERFORMANCE

Sets All Records and Only Platform to Submit Across All Benchmarks



Accuracy

- Accuracy indicates the **quality** of DNNs' **results**.
- Affected by **difficulty** of the task and **dataset**
  - Example: Classification on ImageNet is harder than on MNIST.
- Popularity of DNNs:
  - Achieving high accuracy on diverse tasks is a prime reason for DNNs' widespread use.



Interpret accuracy in its context for an authentic assessment

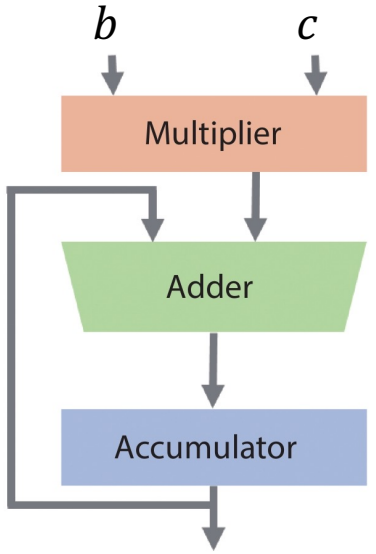
Comparison of Models

	MNIST	**CIFAR-10**	**IMAGENET**
Year	1998	2009	2012
Resolution	28x28	32x32	256x256
Classes	10	10	1000
Training	60k	50k	1.3M
Testing	10k	10k	100k
Accuracy	0.21% error (ICML 2013)	3.47% error (arXiv 2015)	2.25% top-5 error (2017 winner)

Kernel Design

MAC Operation

- **Key Computation**: Multiply-and-accumulate (MAC) operations.
- **Characteristics of MACs**:
  - Negligible dependencies between operations.



$$a = a + b \times c$$

Source: Jia Chen et al.

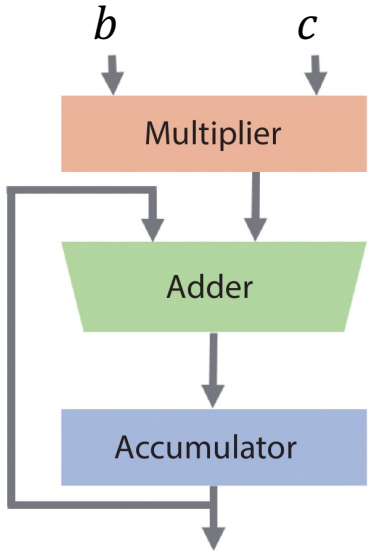
$$C = A \times B$$

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots$$

MAC Operation

MAC Operation

- **Parallelization**: Computations can be easily parallelized due to the independent nature of MACs.
- **High Performance Strategy**:
  - Use of highly parallel compute paradigms.
  - Architectural paradigms can be either **temporal** or **spatial**.



$$a = a + b \times c$$

Source: Jia Chen et al.

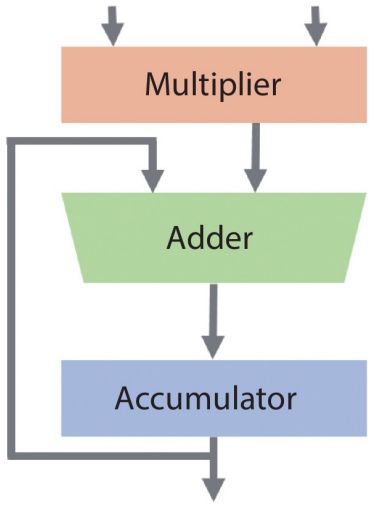
$$C = A \times B$$

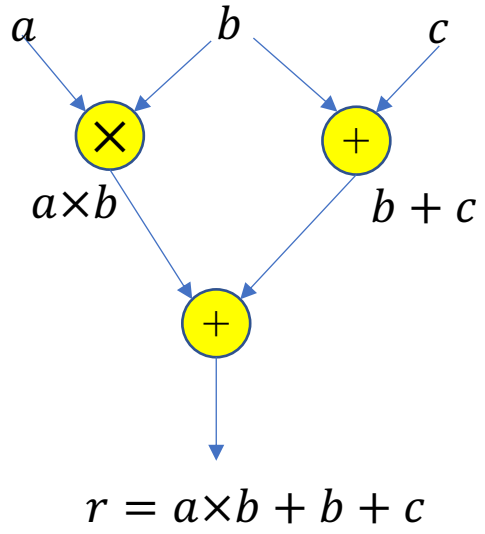
$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots$$

MAC Operation

Design Considerations for DNN Hardware

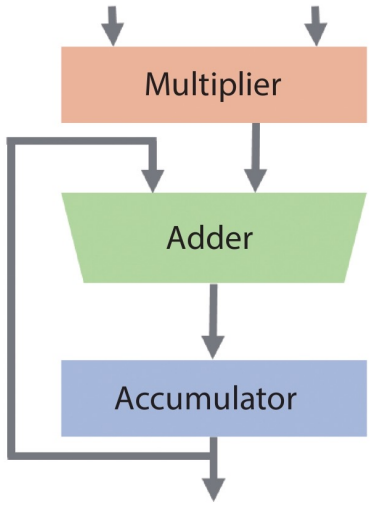
1. Multiple possible spatio-temporal orderings for each DNN layer.
   - Each layer involves performing numerous MAC operations.
   - For each layer, there can be different ways to order these MAC operations.
   - These orderings can be influenced by various factors, including the data dependencies between operations and the availability of computational resources.

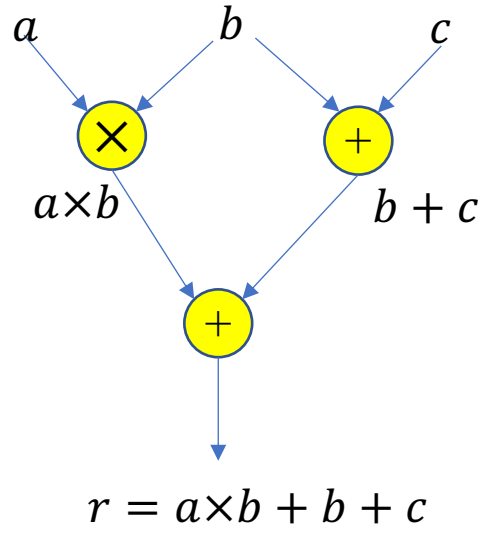




Design Considerations for DNN Hardware

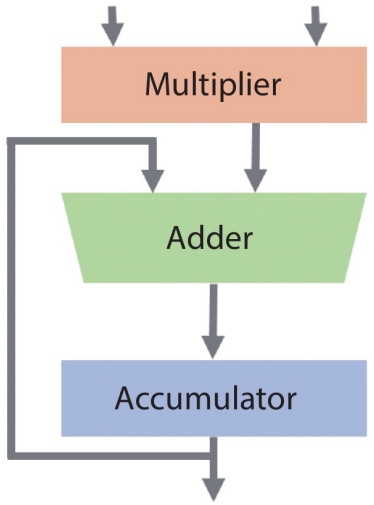
1. Multiple possible spatio-temporal orderings for each DNN layer.
   - Each layer involves performing numerous MAC operations.
   - For each layer, there can be different ways to order these MAC operations.
   - Orderings can be influenced by various factors:
     - the data dependencies between operations and the availability of computational resources.
   - Not feasible for a hardware architecture to support all possible orderings.
     - It is impractical for hardware architectures, such as GPUs or specialized AI accelerators, to support all the possible ways in which MAC operations can be ordered within each DNN layer.

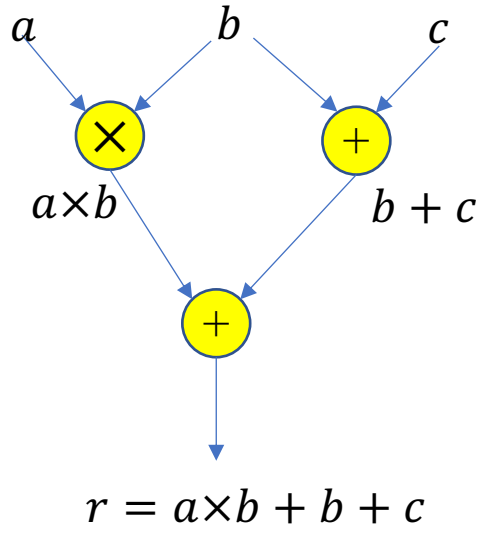




Design Considerations for DNN Hardware

1. Multiple possible spatio-temporal orderings for each DNN layer.
   - Each layer involves performing numerous MAC operations.
   - For each layer, there can be different ways to order these MAC operations.
   - Orderings can be influenced by various factors:
     - the data dependencies between operations and the availability of computational resources.
   - Not feasible for a hardware architecture to support all possible orderings.
     - It is impractical for hardware architectures, such as GPUs or specialized AI accelerators, to support all the possible ways in which MAC operations can be ordered within each DNN layer.
   - **Practical Approach**: Support a limited set of orderings, leading to fewer legal mappings.
     - This approach leads to "**legal mappings**," which are specific combinations of operations and orderings that the hardware can execute efficiently.





Example of Mapping

Task: Performing a convolution operation in a deep neural network layer.

- **Mapping 1 Steps**:
  1. **Input Data Loading**
  2. **Filter Loading**
     1. The weights (filters) associated with the convolution operation are loaded into GPU registers.
  3. **Convolution Computation**
     1. This computation can be done in parallel across multiple processing units within the GPU.
  4. **Activation Function**
  5. **Output Storing**

Example of Mapping

Task: Performing a convolution operation in a deep neural network layer.

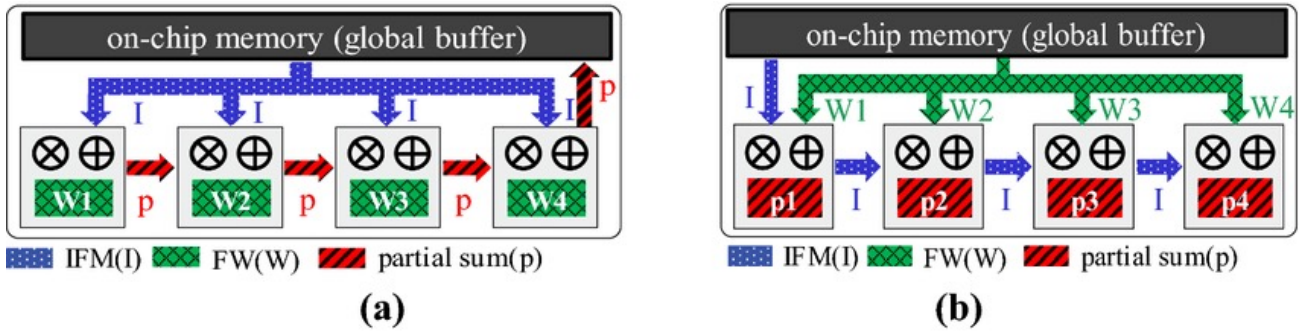
- **Mapping 1 Steps**:
  1. **Input Data Loading**
  2. **Filter Loading**
     1. The weights (filters) associated with the convolution operation are loaded into GPU registers.
  3. **Convolution Computation**
     1. This computation can be done in parallel across multiple processing units within the GPU.
  4. **Activation Function**
  5. **Output Storing**

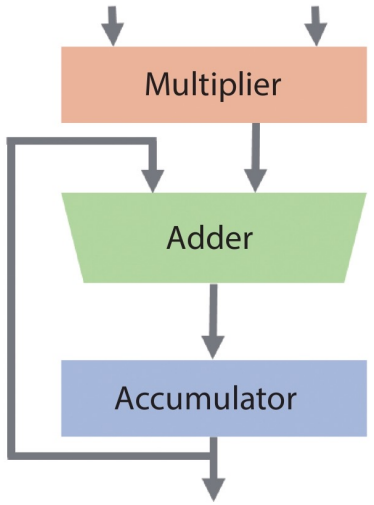
- **Mapping 2 Steps**:
  1. **Input Data Loading**
  2. **Filter Loading**
     1. The weights (filters) associated with the convolution operation are loaded into GPU registers.
  3. **Repeat:**
     1. **Partial Convolution**
        1. Instead of computing the entire convolution in one step, the GPU divides the operation into smaller sub-convolutions.
     2. **Accumulation:**
        1. After each partial convolution, the GPU accumulates the results separately for each segment or subset.
  4. **Activation Function**
  5. **Output Storing**

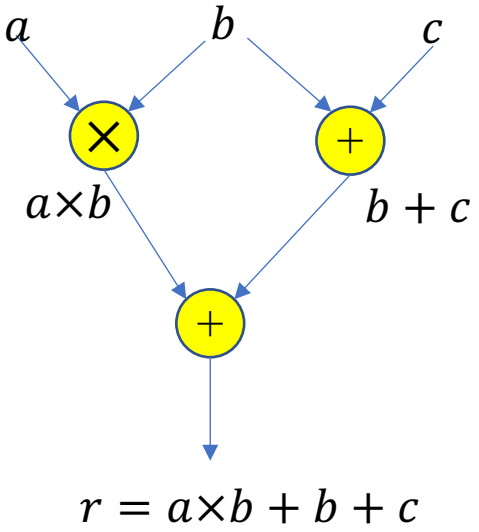
Design Considerations for DNN Hardware

2. **The Importance of Dataflow**:

1. Defines the subset of orderings a DNN hardware can support.
2. Directly impacts the optimization quality.





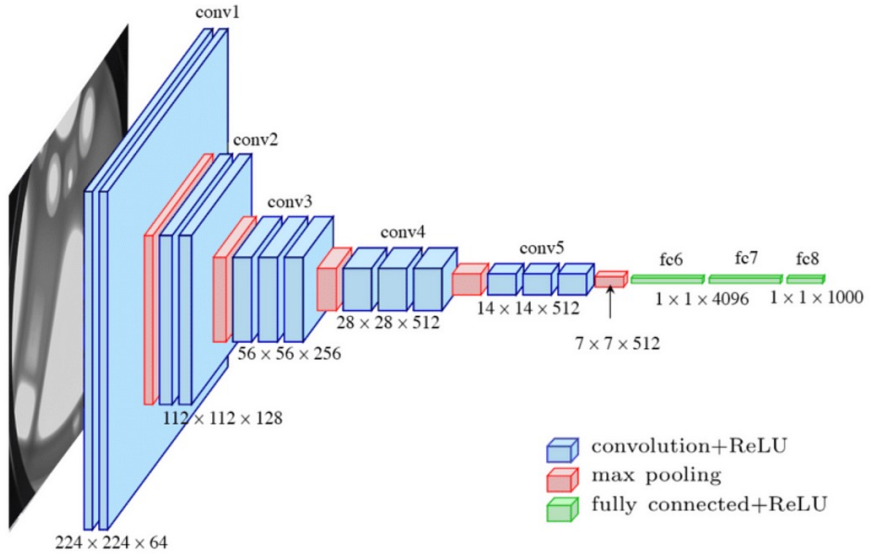


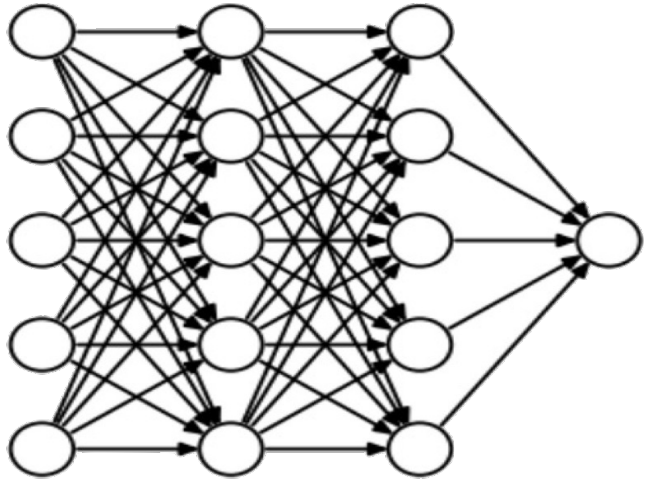
A good dataflow design can lead to better utilization of hardware resources, reduced latency, and lower power consumption, which are all important factors in the overall performance of DNN systems.

Design Considerations for DNN Hardware

3.Layer Variability and Scalability:

- DNN layer shape and size vary across models and layers.
- Emerging DNNs present increasing numbers of layers.
- **Implication**: Need for hardware to be adaptable and scalable.

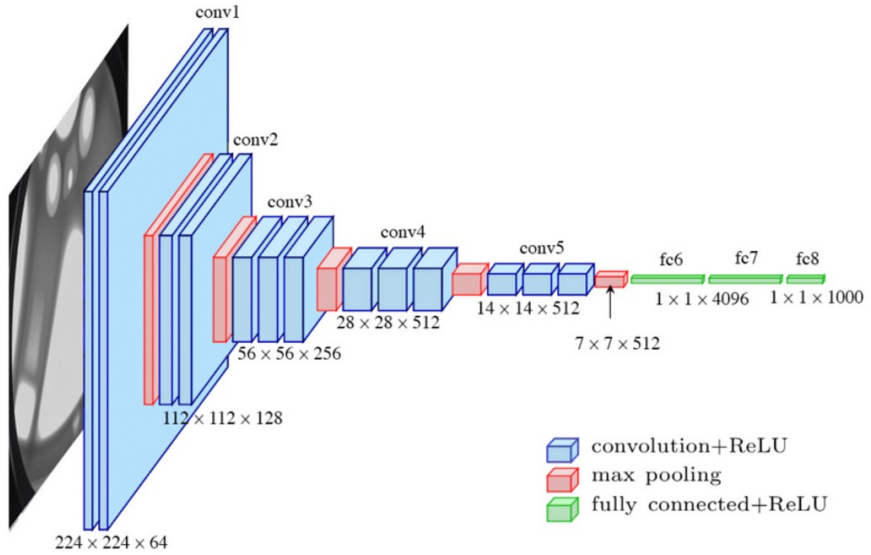


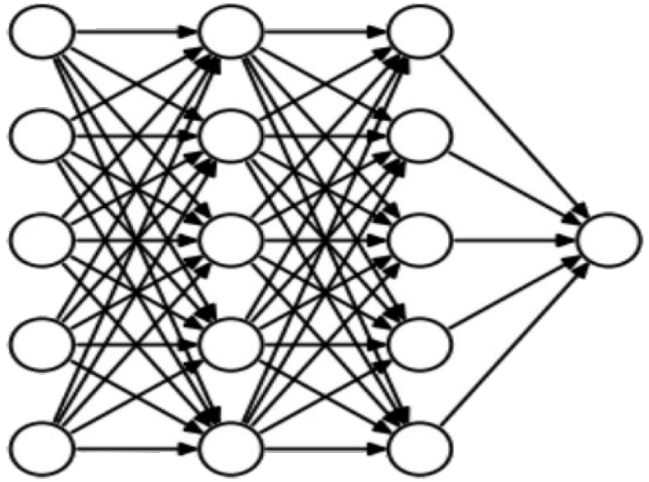


Design Considerations for DNN Hardware

4.Avoiding Assumptions:

- DNNs evolve, and their size tends to grow.
- **Avoid assuming** the entire model will fit on-chip.
- **Flexibility** is pivotal to support various DNNs efficiently, remains as a primary challenge in DNN accelerator design.





Challenges in Designing Specialized DNN Hardware

Design Flexibility vs. Efficiency:

- **Goal**: Flexible to support optimal performance and energy efficiency for different DNN layers.
- **Trade-off**: More flexibility often results **decreased efficiency**.

- **Example**:
  - A hardware accelerator designed to perform **convolution** operations efficiently might be very efficient in image recognition models, but not perform as well for **recurrent** layers used in natural language processing.
    - On the other hand, a more flexible and general-purpose hardware accelerator might handle both but might not achieve the same level of efficiency for either type compared to specialized accelerators.

Challenges in Designing Specialized DNN Hardware

Configuration for Efficiency:

- **Goal**: **Minimize energy consumption** and **maintain performance** for specific DNN models.
- Challenges:
  - **Mapping**: Defines the MAC operations' execution order (both temporally and spatially) and the data movement across memory levels.
  - **Variability**: Numerous possible mappings exist for a single DNN layer, making it crucial to pinpoint the best ones.

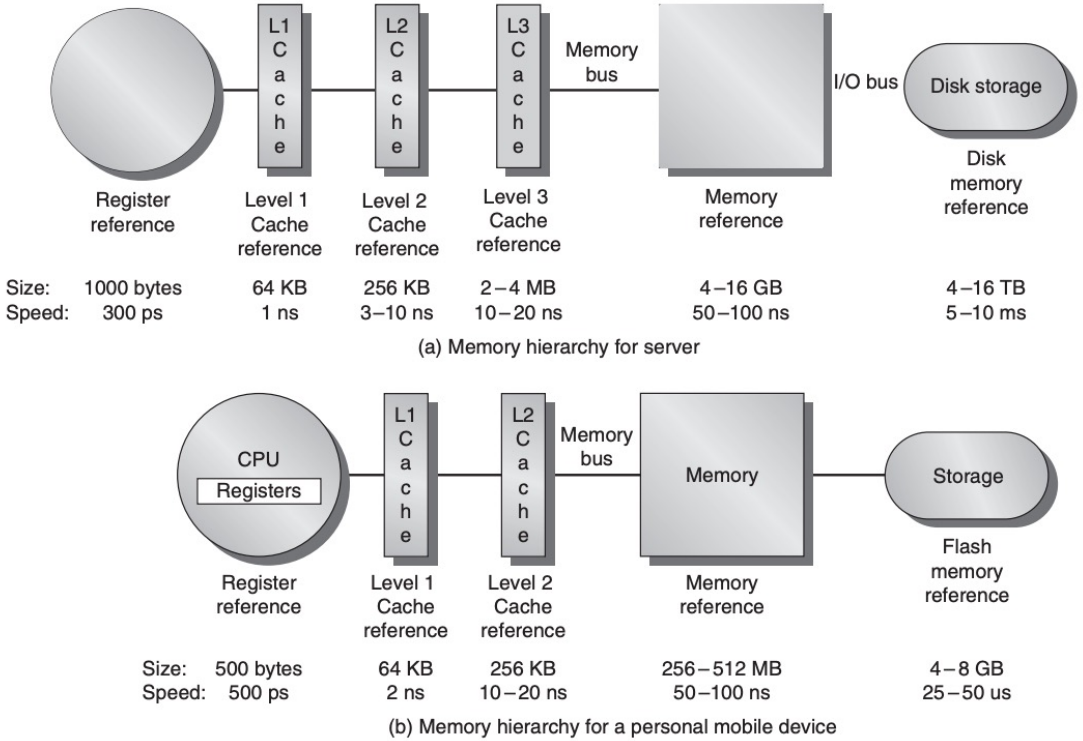
"Configuration for Efficiency" aims to reduce energy consumption in DNNs while preserving their performance.

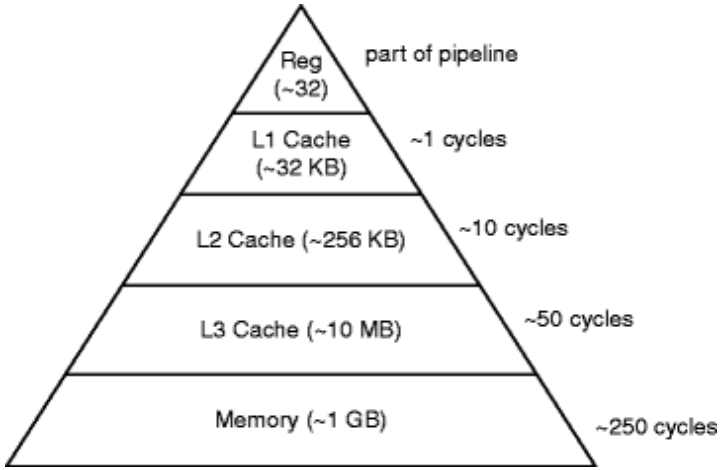
Optimization Techniques in DNN Hardware Execution

- 1. **Arithmetic** Enhancements:
  - **Specialized** Instructions: Reduces overhead for efficiency.
  - **Quantization**: Embrace lower precision for speed.

Optimization Techniques in DNN Hardware Execution

- 2. **Memory** Strategies:
  - **Locality**: Leverage on-chip memory to save costs.
  - **Reuse**: Reduce expensive memory fetches.

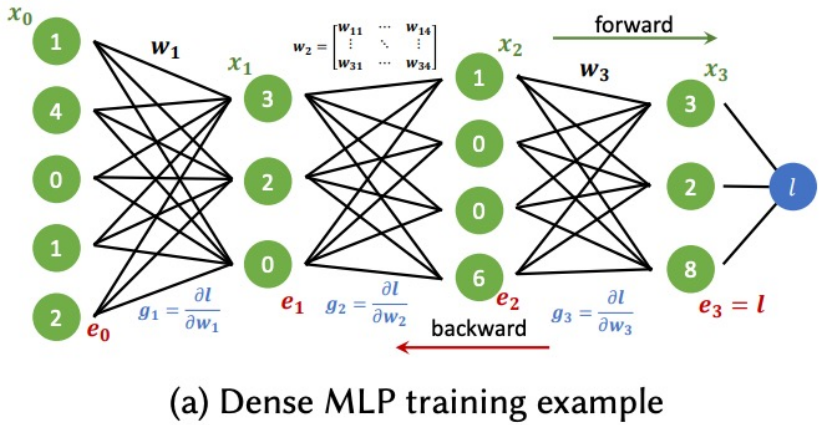




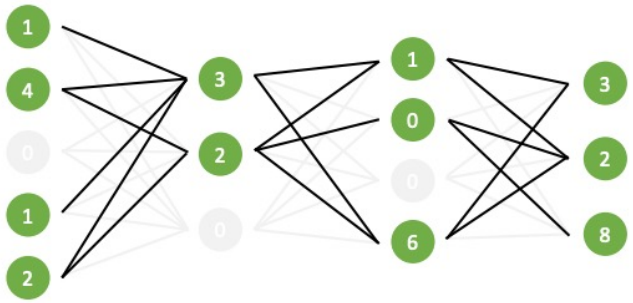
Source: Computer Organization And Design, J. Hennessy and D .Patterson

Optimization Techniques in DNN Hardware Execution

- 3. **Streamlining** Operations:
  - **Sparsity**: Bypass non-essential operations.



(a) Dense MLP training example

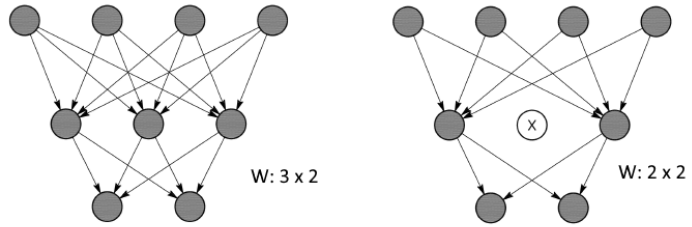


(b) Sparsified MLP

Source: Torsten Hoefler et al.

Optimization Techniques in DNN Hardware Execution

- 4. **Model-Level** Optimizations
  - Simplifying the DNN model
  - Examples:
    - **Network Pruning**: Simplifying the neural network by removing redundant or less important neurons, which can reduce the computational load without significantly affecting performance.

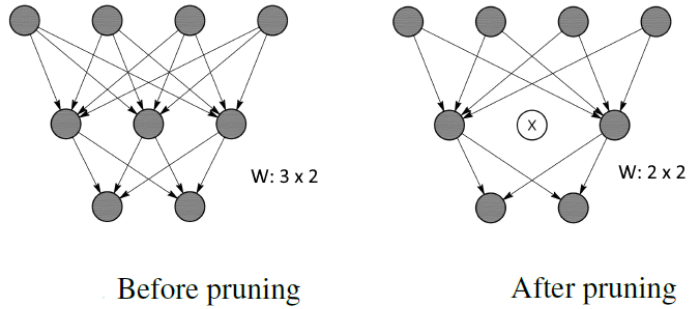


Source: Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures. Hengyuan Hu et al.

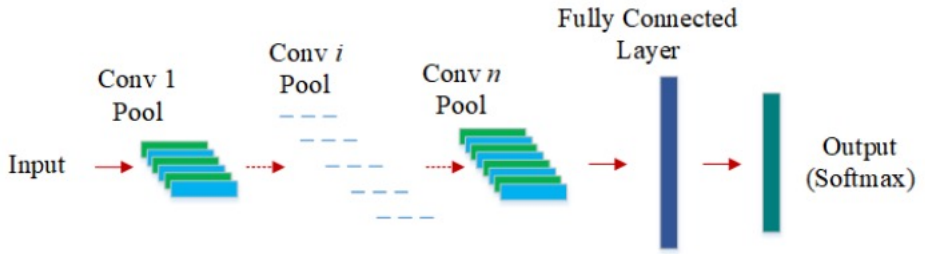
- **Weight Pruning**: Removing individual weights with low magnitudes to reduce model size.
- **Neuron Pruning**: Eliminating entire neurons with low impact on network output for compactness.
- **Filter Pruning** (CNNs): Removing convolutional filters with low importance for feature extraction.
- **Structured Pruning**: Removing entire structures or subnetworks like layers or neuron groups.

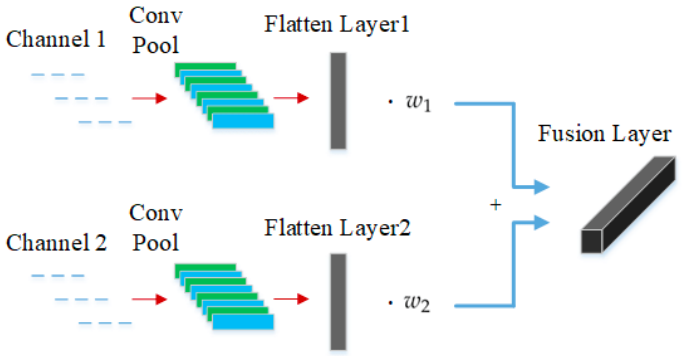
Optimization Techniques in DNN Hardware Execution

- 4. **Model-Level** Optimizations
  - Simplifying the DNN model
  - Examples:
    - **Network Pruning**: Simplifying the neural network by removing redundant or less important neurons, which can reduce the computational load without significantly affecting performance.
    - **Layer Fusion**: Combining multiple layers of the neural network into a single layer to reduce the overhead associated with moving data between layers.



Source: Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures. Hengyuan Hu et al.



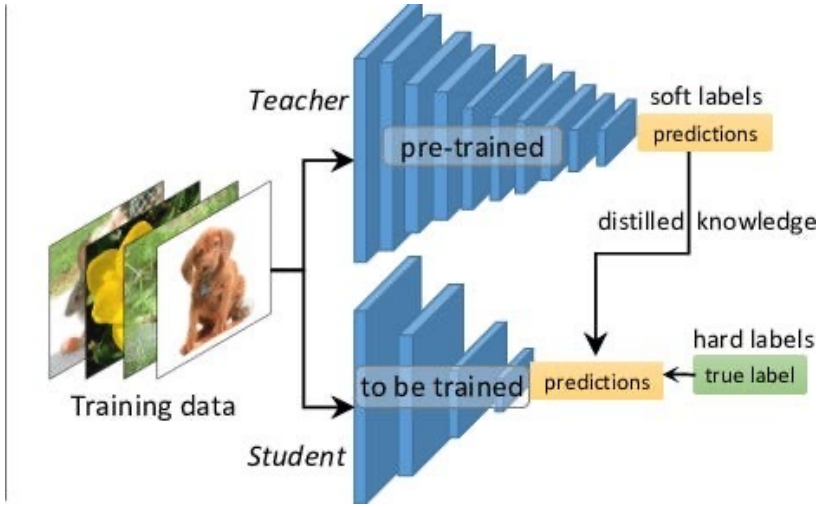


Fusion layer to combine 2 feature vectors

Source: Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures. Hengyuan Hu et al.

Optimization Techniques in DNN Hardware Execution

- 4. **Model-Level** Optimizations
  - Modifying the DNN model in such a way that it aligns well with the capabilities of the hardware.
  - Examples:
    - **Knowledge Distillation**: Training a smaller, more efficient model (the "student") to mimic the behavior of a larger, pre-trained model (the "teacher"), thus achieving similar performance with less computational demand.

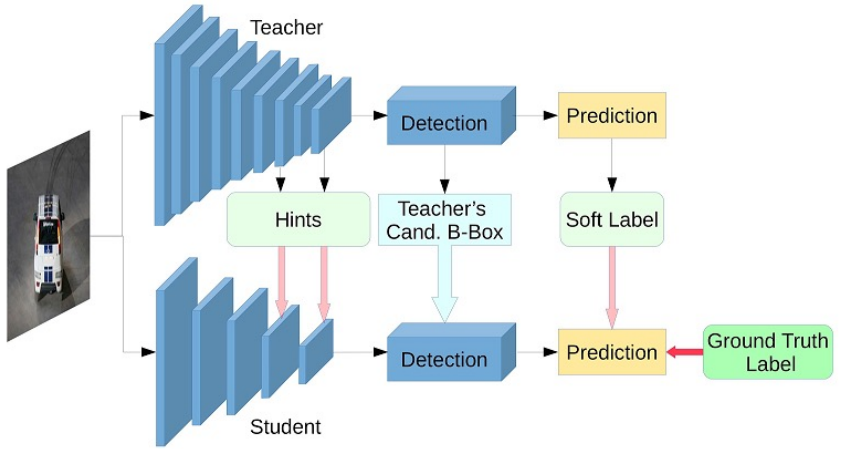


Source: towardsdatascience.com

- As the 'Teacher' model has already been trained, it can make predictions on the training data.
- These predictions, known as **'soft labels'**, carry additional information compared to **'hard labels'** because they include the probabilities (or confidence levels) the 'Teacher' model assigns to each possible class.
- Soft labels can provide a richer signal to the 'Student' model than hard labels because they reflect the 'Teacher' model's uncertainty.

Optimization Techniques in DNN Hardware Execution

- 4. **Model-Level** Optimizations
  - Modifying the DNN model in such a way that it aligns well with the capabilities of the hardware.
  - Examples:
    - **Knowledge Distillation**: Training a smaller, more efficient model (the "student") to mimic the behavior of a larger, pre-trained model (the "teacher"), thus achieving similar performance with less computational demand.



Source: Image Credits : Xiang Yu

- **Intermediate hints besides soft labels:**
  - The student is trained to replicate the feature activations of the teacher at certain intermediate layers
- **Teacher's Candidate B-Box**: The teacher's detected bounding boxes around the objects are used as part of the hints to the student network to improve its own detection accuracy.

Where Does the Energy Go?

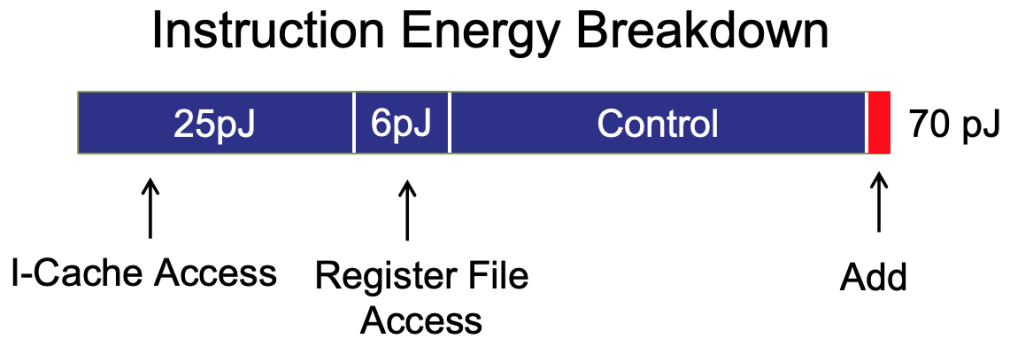
Instruction Energy breakdown

Rough Energy Numbers (45nm)

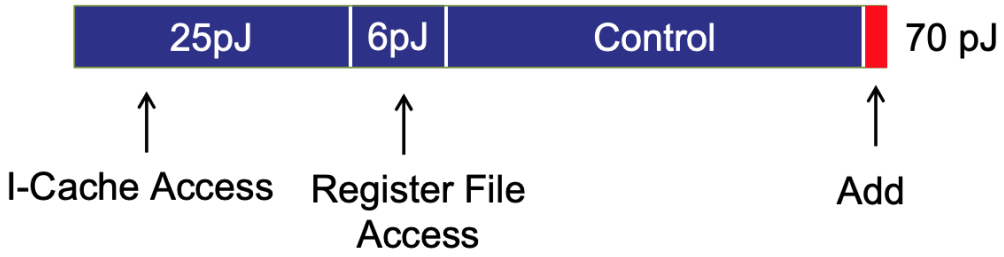
Integer	
Add	
8 bit	0.03pJ
32 bit	0.1pJ
Mult	
8 bit	0.2pJ
32 bit	3 pJ

FP	
FAdd	
16 bit	0.4pJ
32 bit	0.9pJ
FMult	
16 bit	1pJ
32 bit	4pJ

Memory	
Cache	(64bit)
8KB	10pJ
32KB	20pJ
1MB	100pJ
DRAM	1.3-2.6nJ

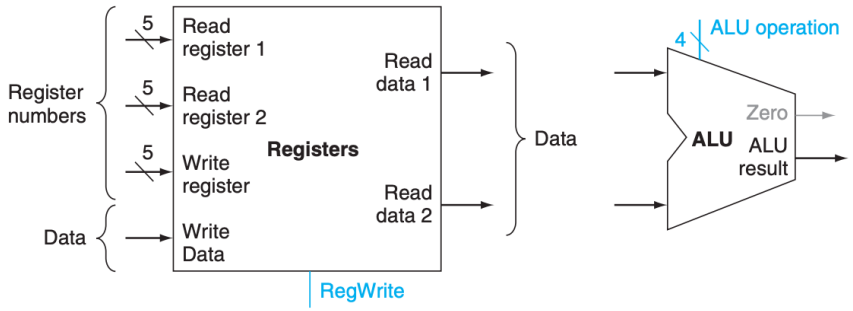


Simple Instructions



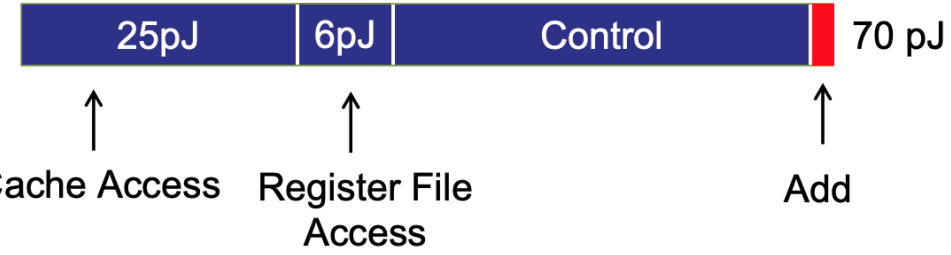
MIPS assembly language

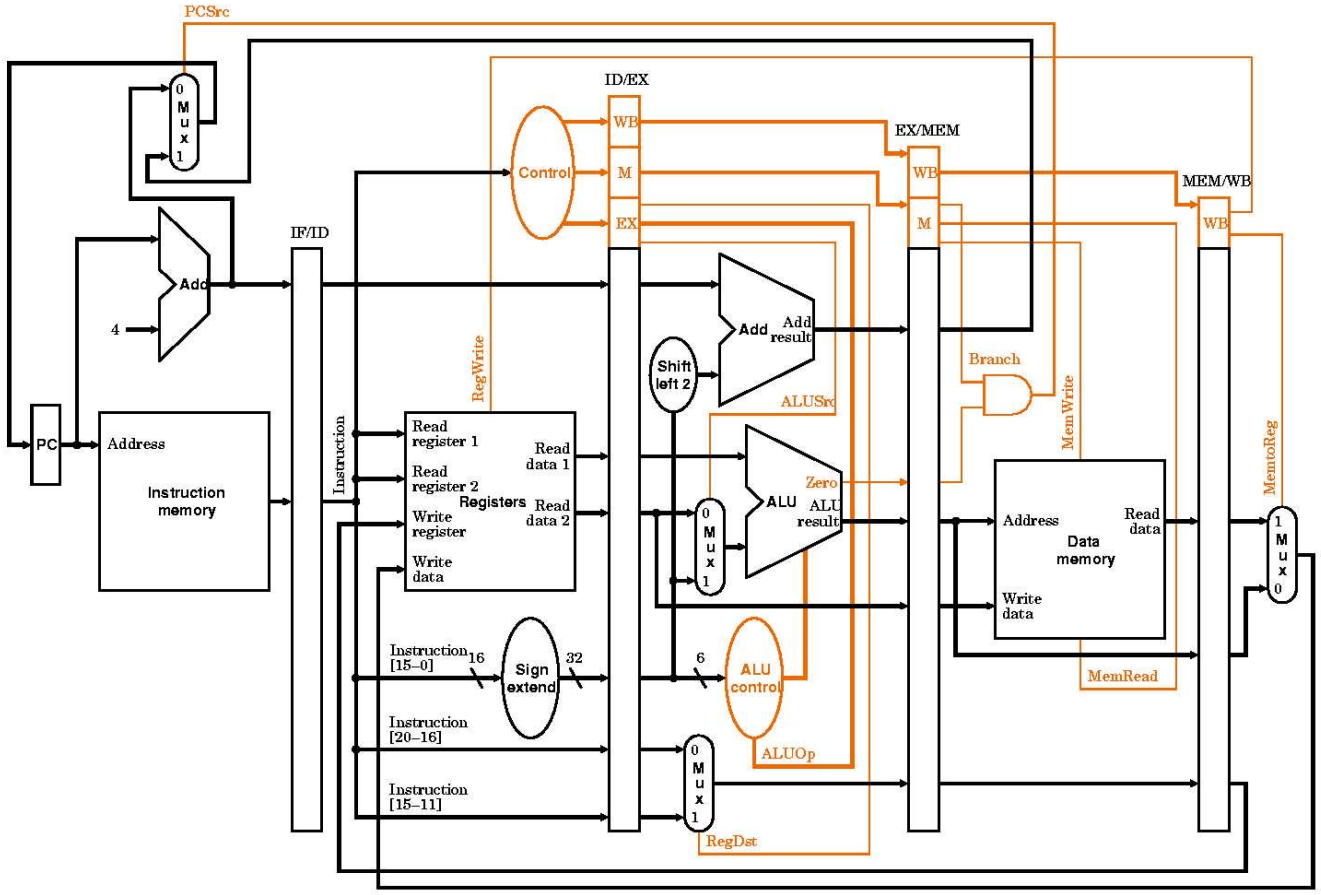
Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1,$s2,$s3`	$s1 = $s2 + $s3	Three operands; data in registers
	subtract	`sub $s1,$s2,$s3`	$s1 = $s2 – $s3	Three operands; data in registers
	add immediate	`addi $s1,$s2,100`	$s1 = $s2 + 100	Used to add constants
Data transfer	load word	`lw $s1,100($s2)`	$s1 = Memory[$s2 + 100]	Data from memory to register
	store word	`sw $s1,100($s2)`	Memory[$s2 + 100] = $s1	Data from register to memory



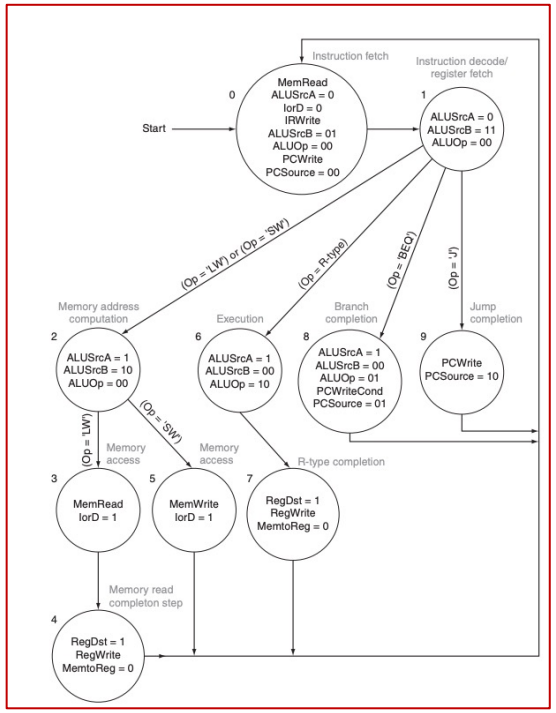
Source: Computer Organization And Design,
J. Hennessy and D .Patterson

Simple Instructions





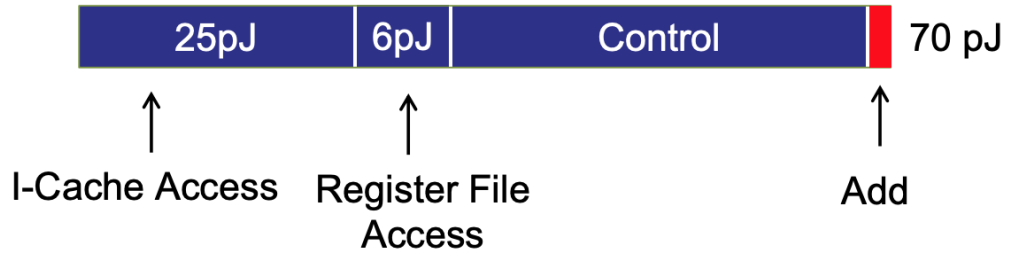
Datapath

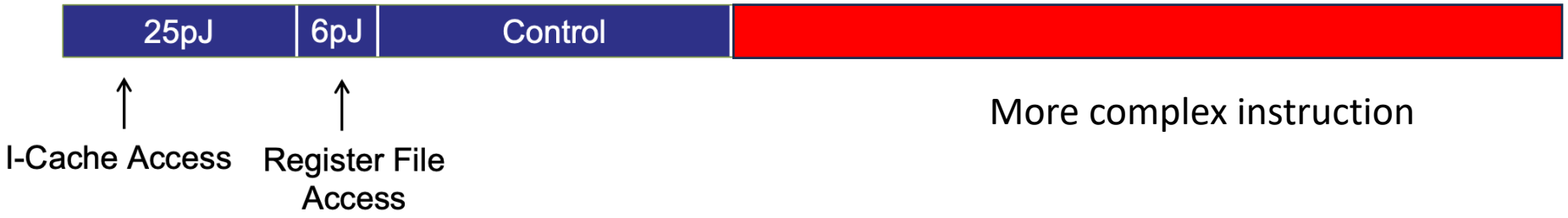


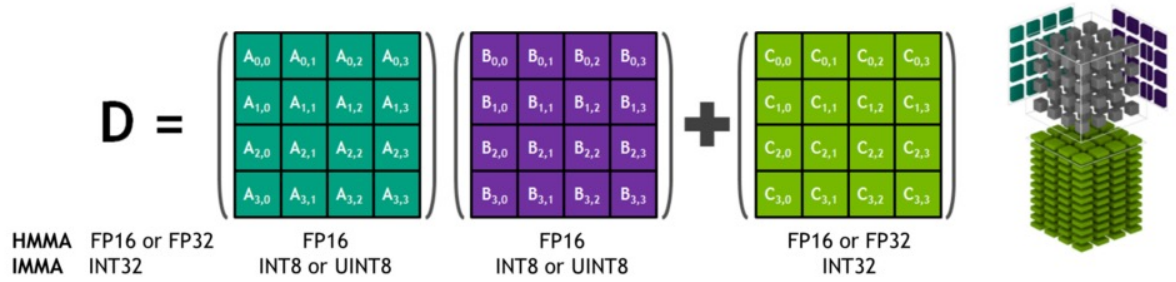
The control finite state machine

Source: Computer Organization And Design, J. Hennessy and D .Patterson

Complex Instructions

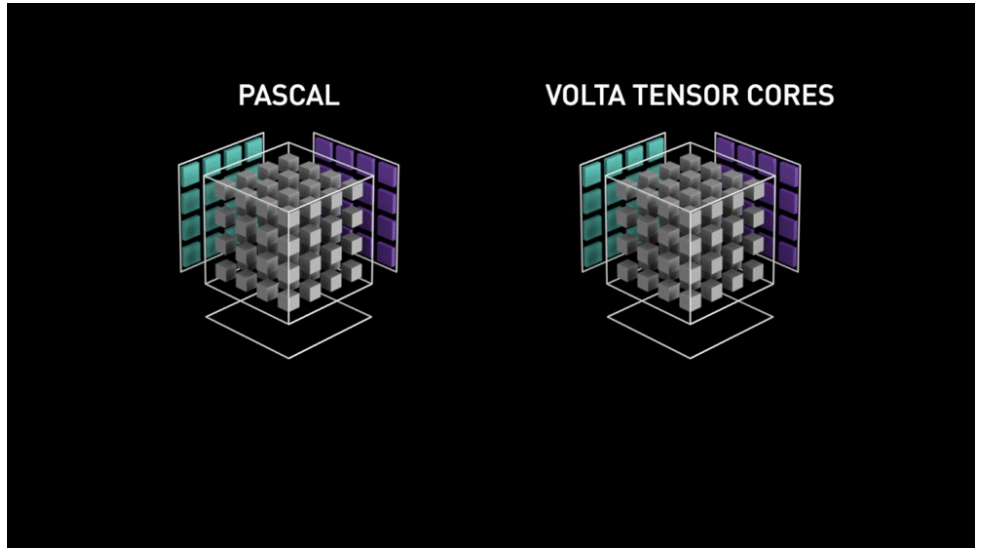






Tensor Core HMMA/IMMA 4x4x4 matrix multiply and accumulate

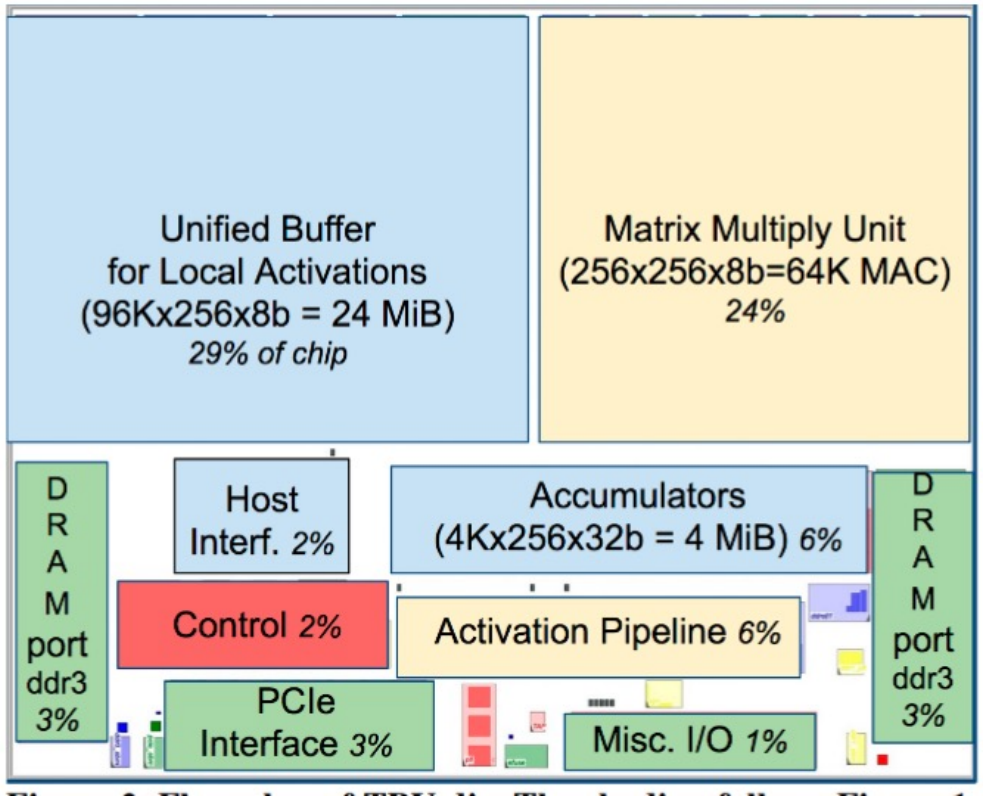
Complex Instructions



16x16 = 256* MAC/cycle

*~ *500 tensor cores per GPU*

Source: Nvidia

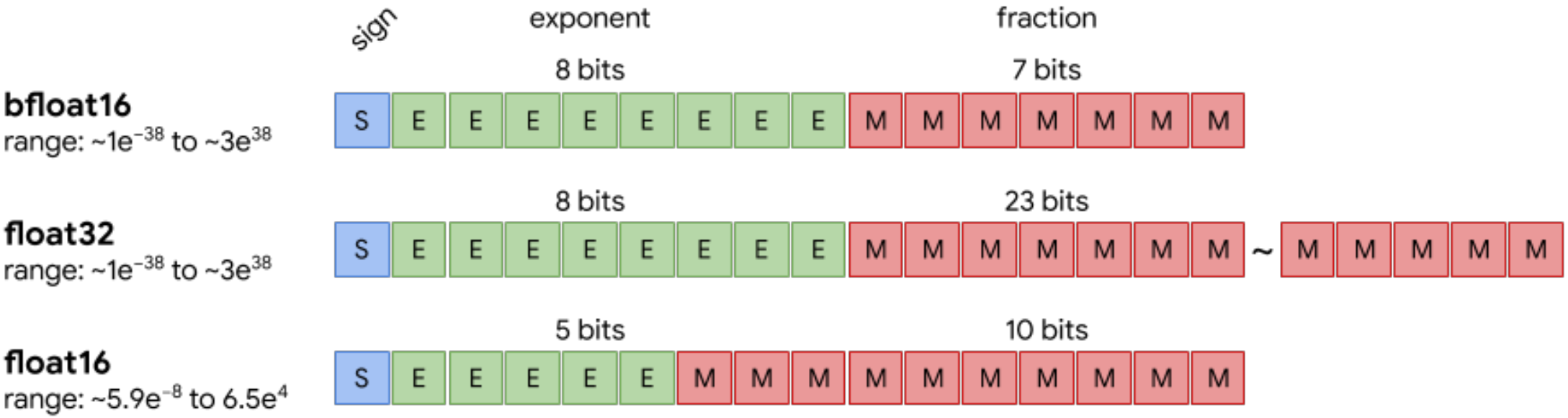


256x256 = 64 KMAC/cycle

Source: Google

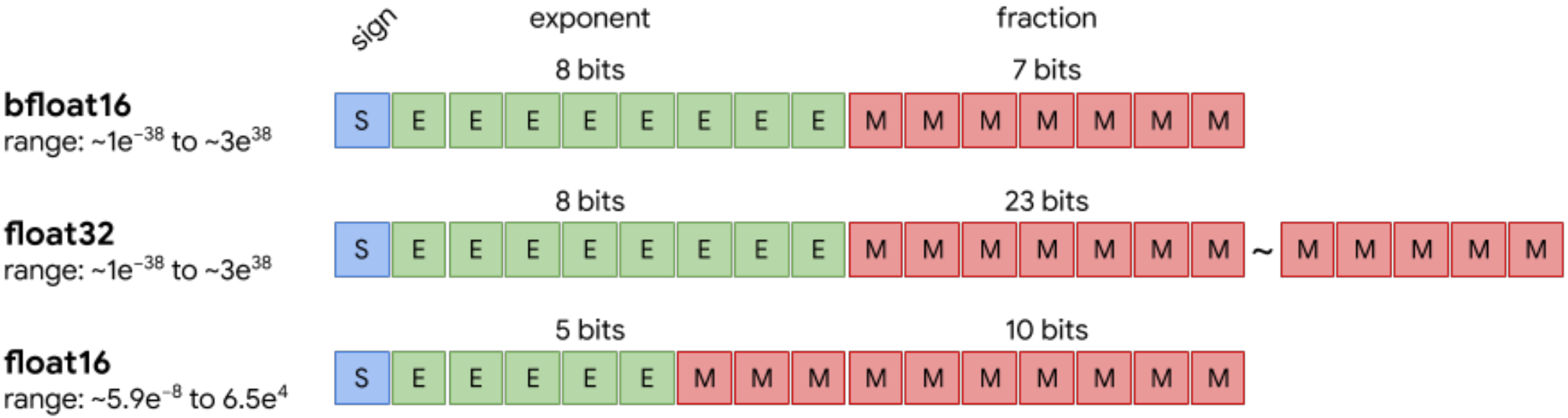
Numerical Precision

$$\text{value} = (-1)^{\text{sign}} \times (1.\text{fraction}) \times 2^{\text{exponent}}$$



Numerical Precision

$$\text{value} = (-1)^{\text{sign}} \times (1.\text{fraction}) \times 2^{\text{exponent}}$$

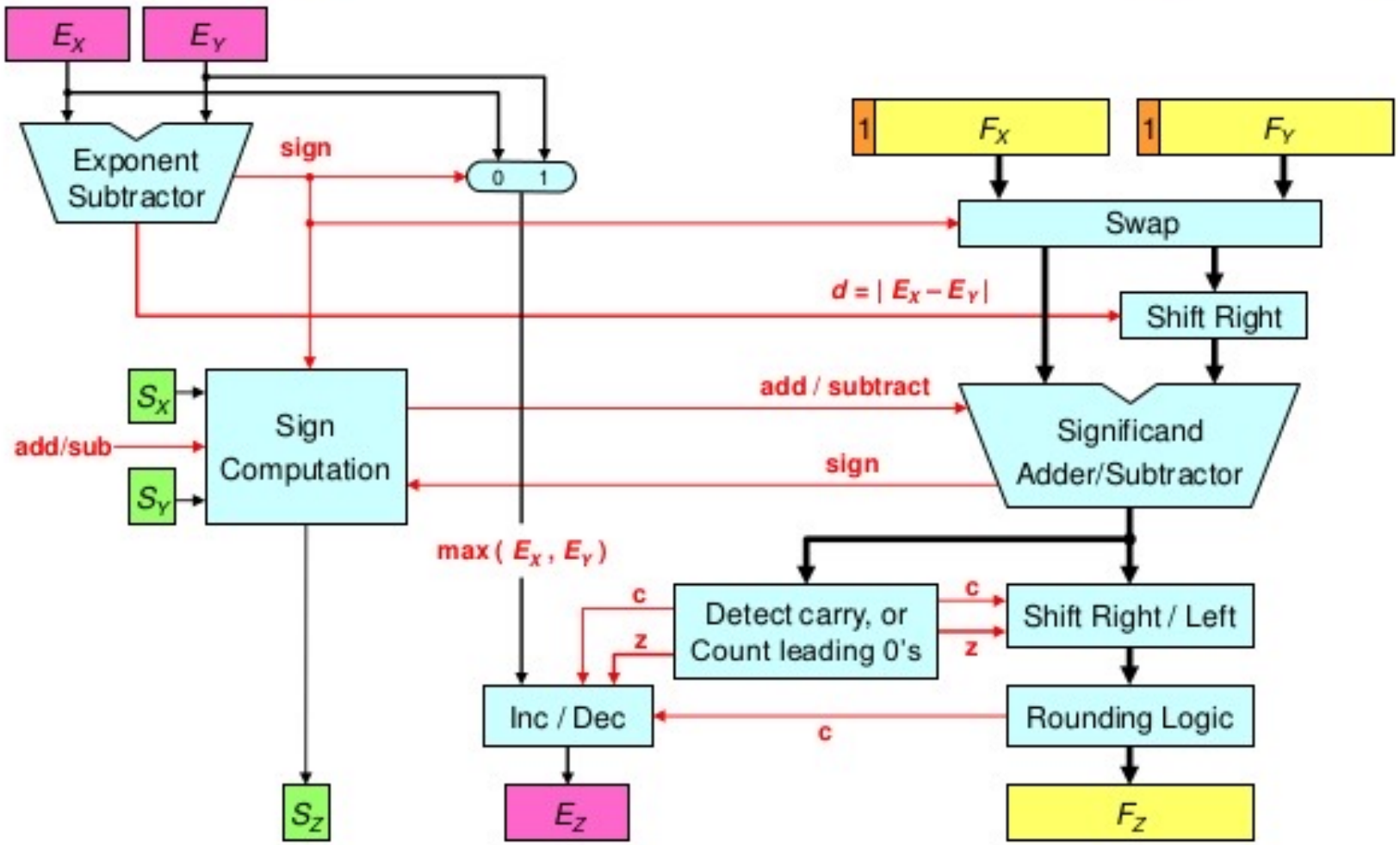


bfloat16:

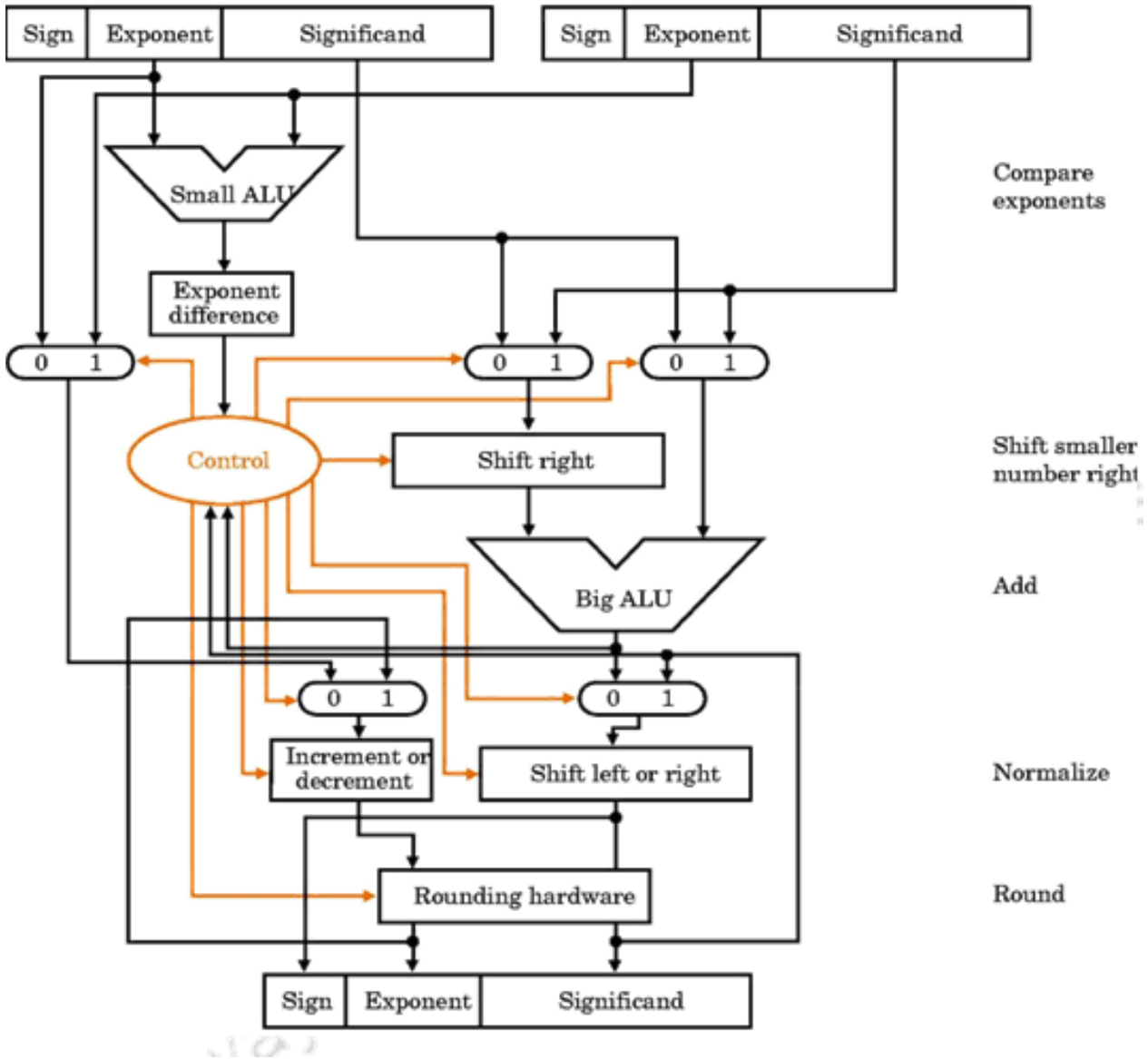
$$2^{-126} \times 2^{-7} \approx 1.18 \times 10^{-38}$$

$$(2 - 2^{-7}) \times 2^{127} \approx 3.39 \times 10^{38}$$

Floating Point Addition Block

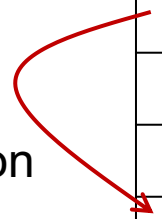


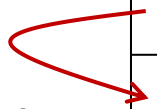
Floating Point Multiplication Block



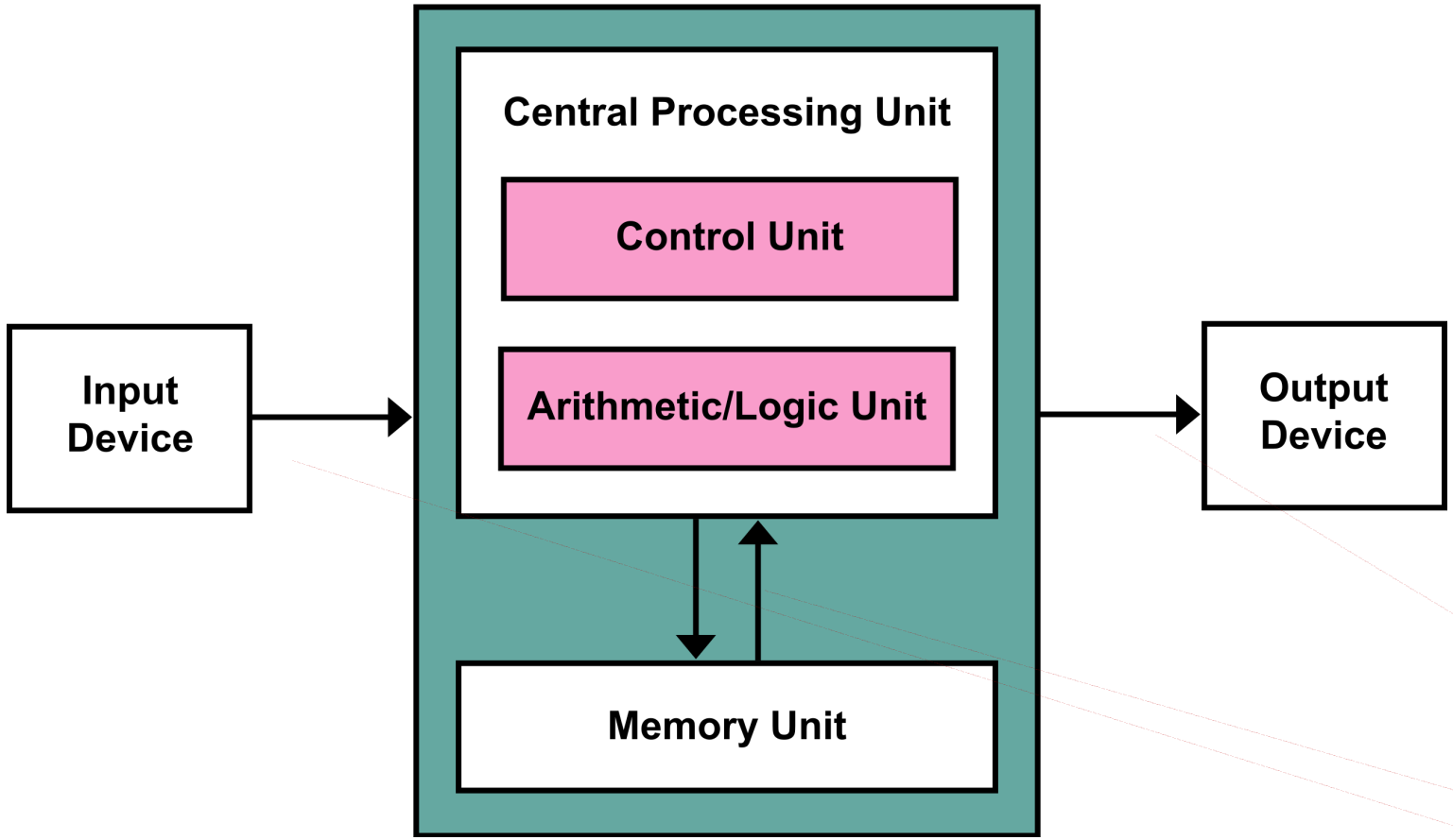
Cost of Arithmetic Operations

Operation	Precision	Energy (pJ)
Addition	INT8	0.03
	INT16	0.05
	INT32	0.1
	FP16	0.4
	FP32	0.9
Multiplication	INT8	0.2
	INT32	3.1
	FP16	1.1
	FP32	3.7
32-bit SRAM Read (8 kB)		5
32-bit SRAM Read (32 kB)		10
32-bit SRAM Read (1 MB)		100
32-bit DRAM Read		640

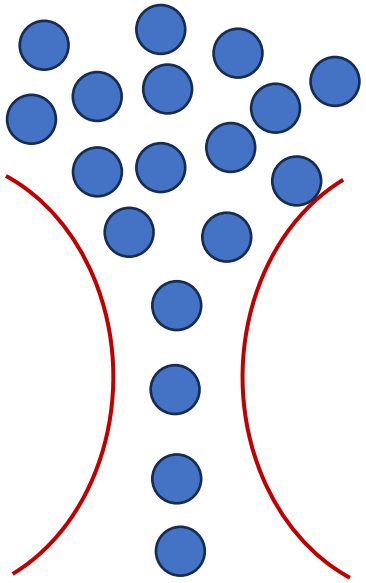




Von Neumann Architecture



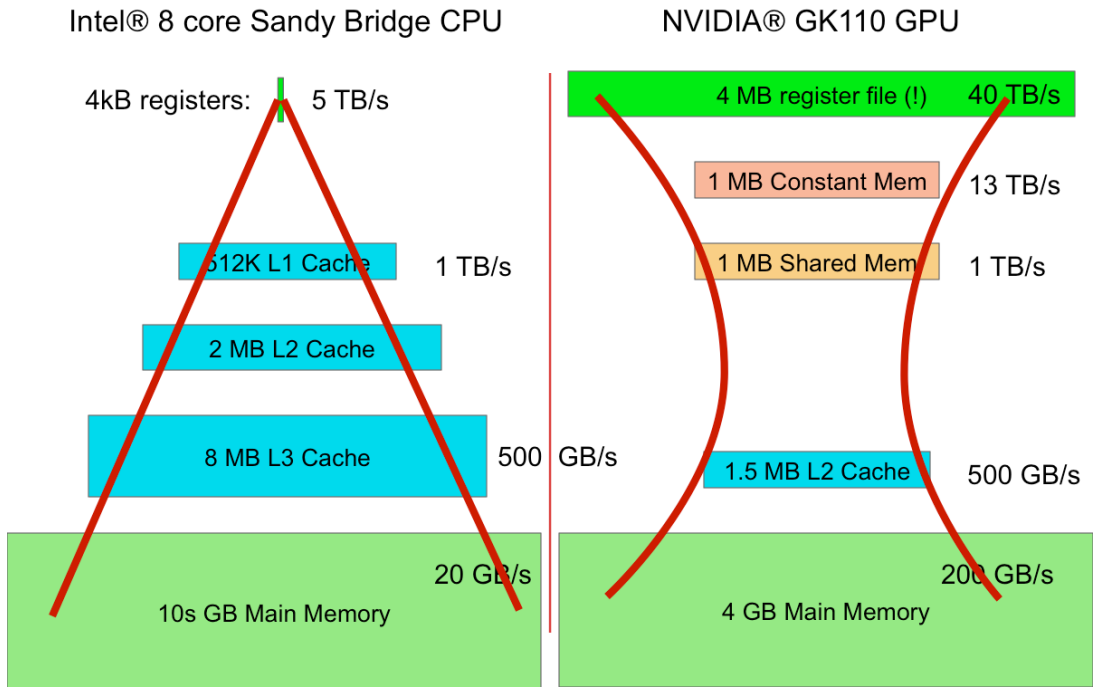
Source: Wikipedia

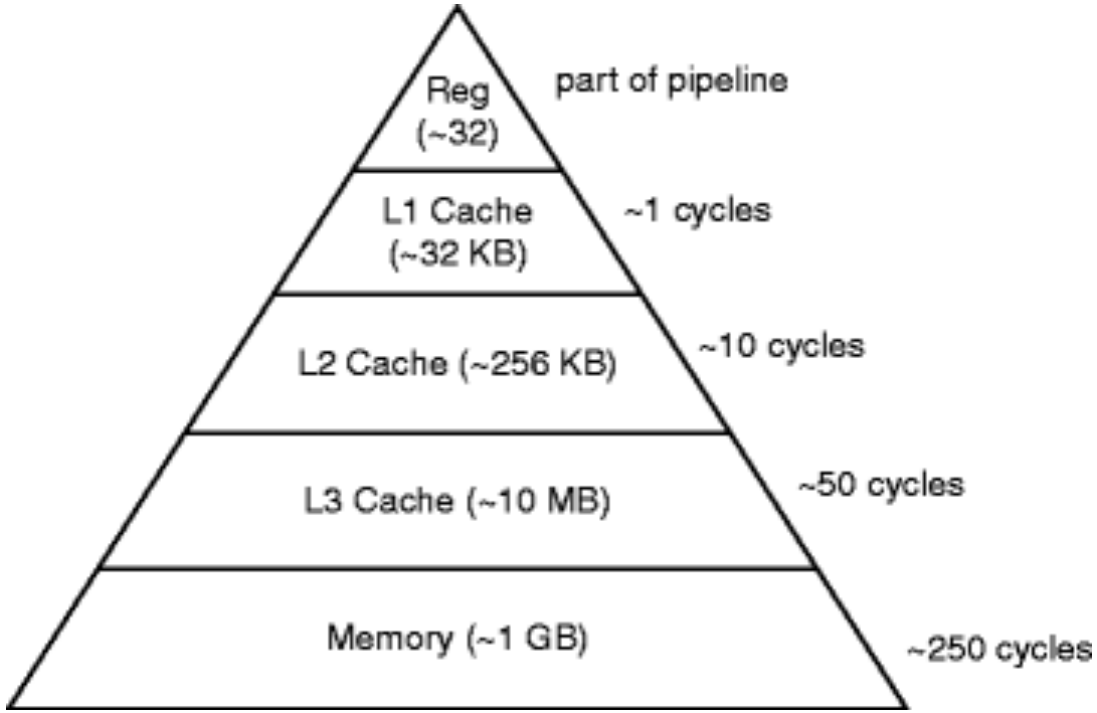


Bottleneck

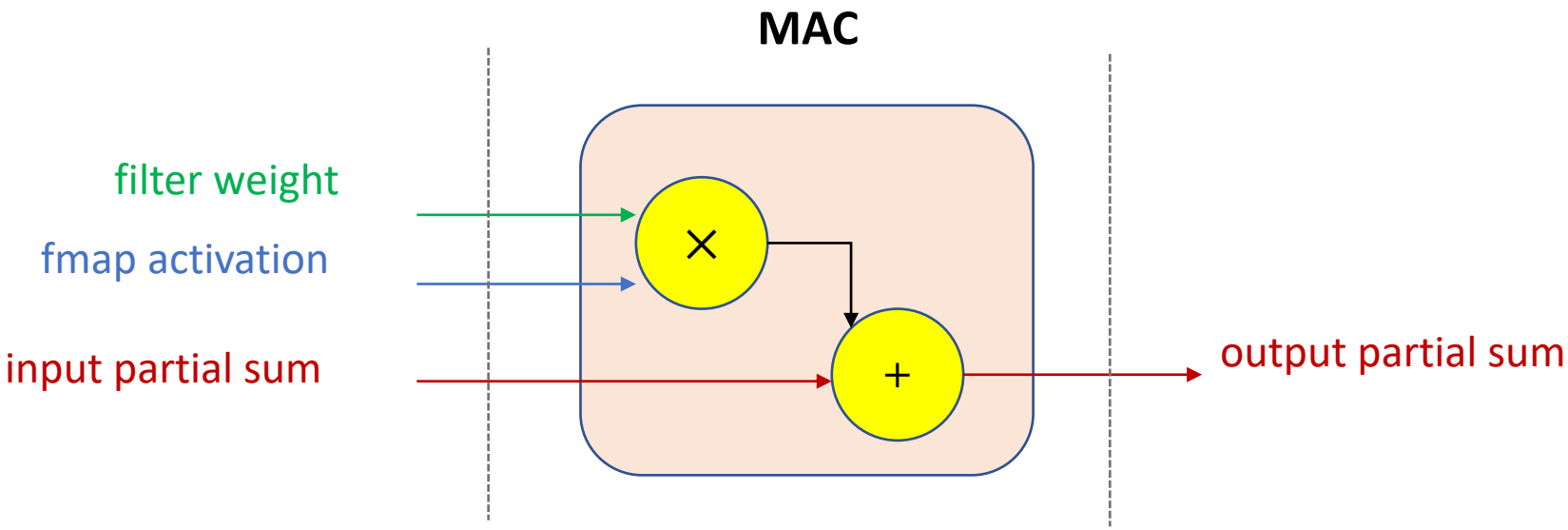
Memory Cost

- Keep Data close (Locality)
- Reuse Data as much as possible

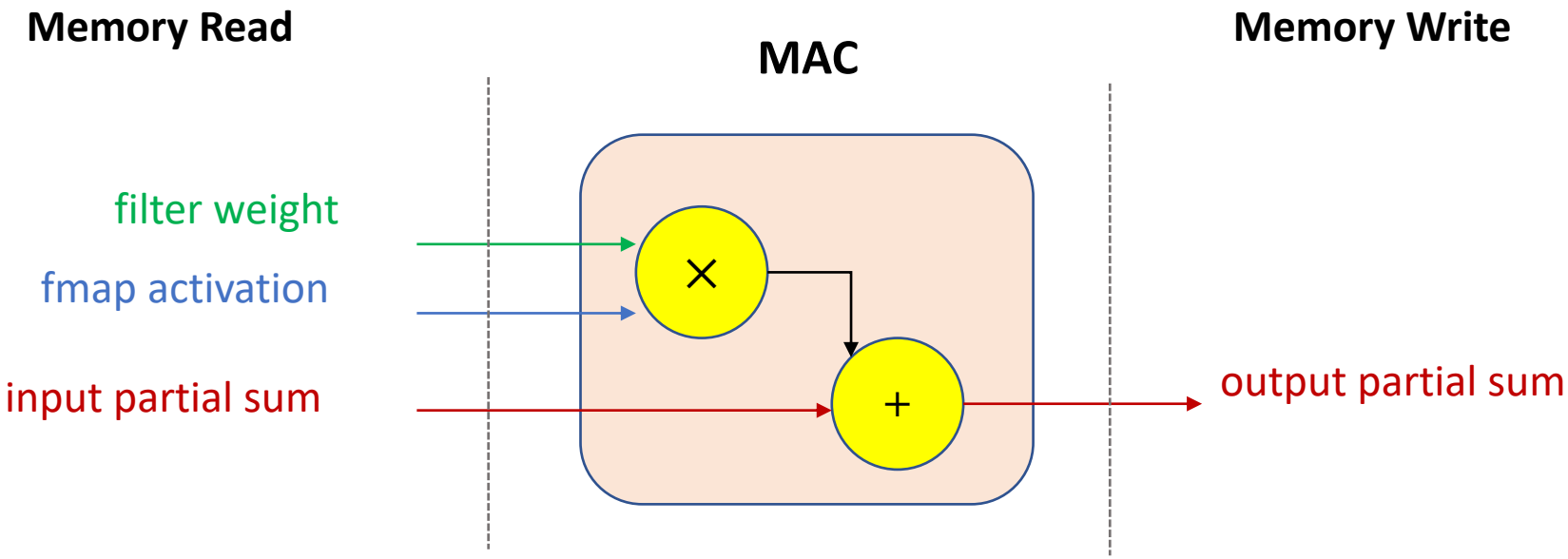




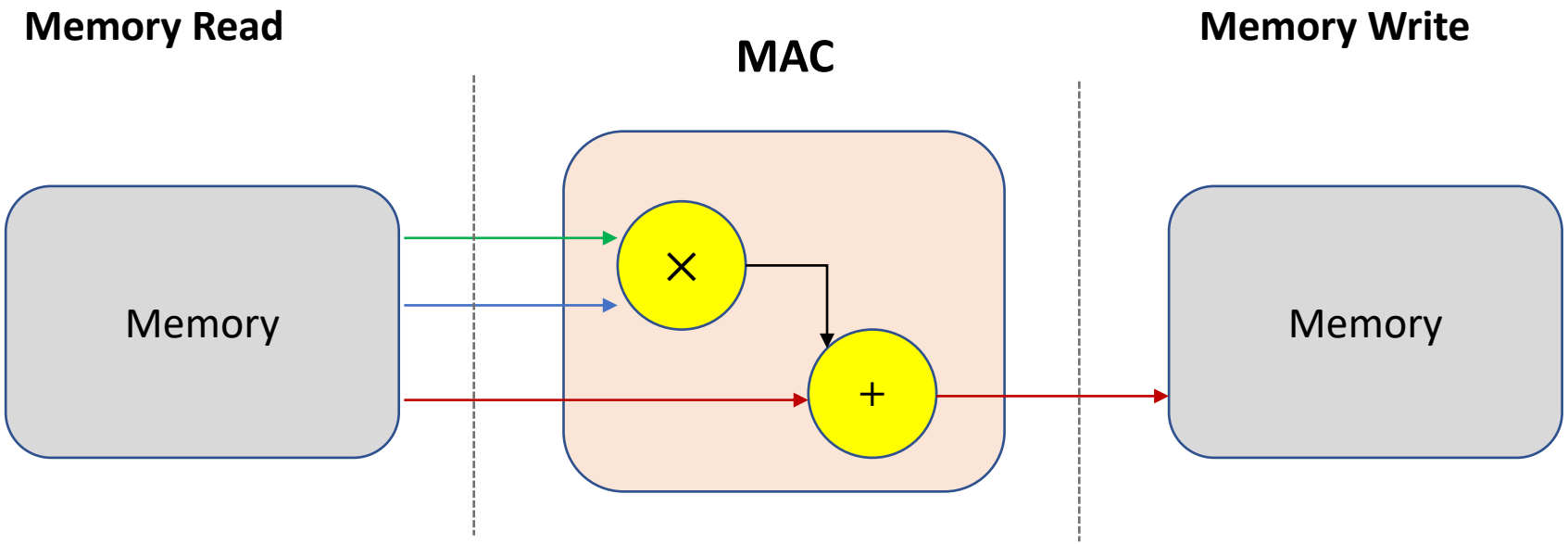
Memory Access is Expensive



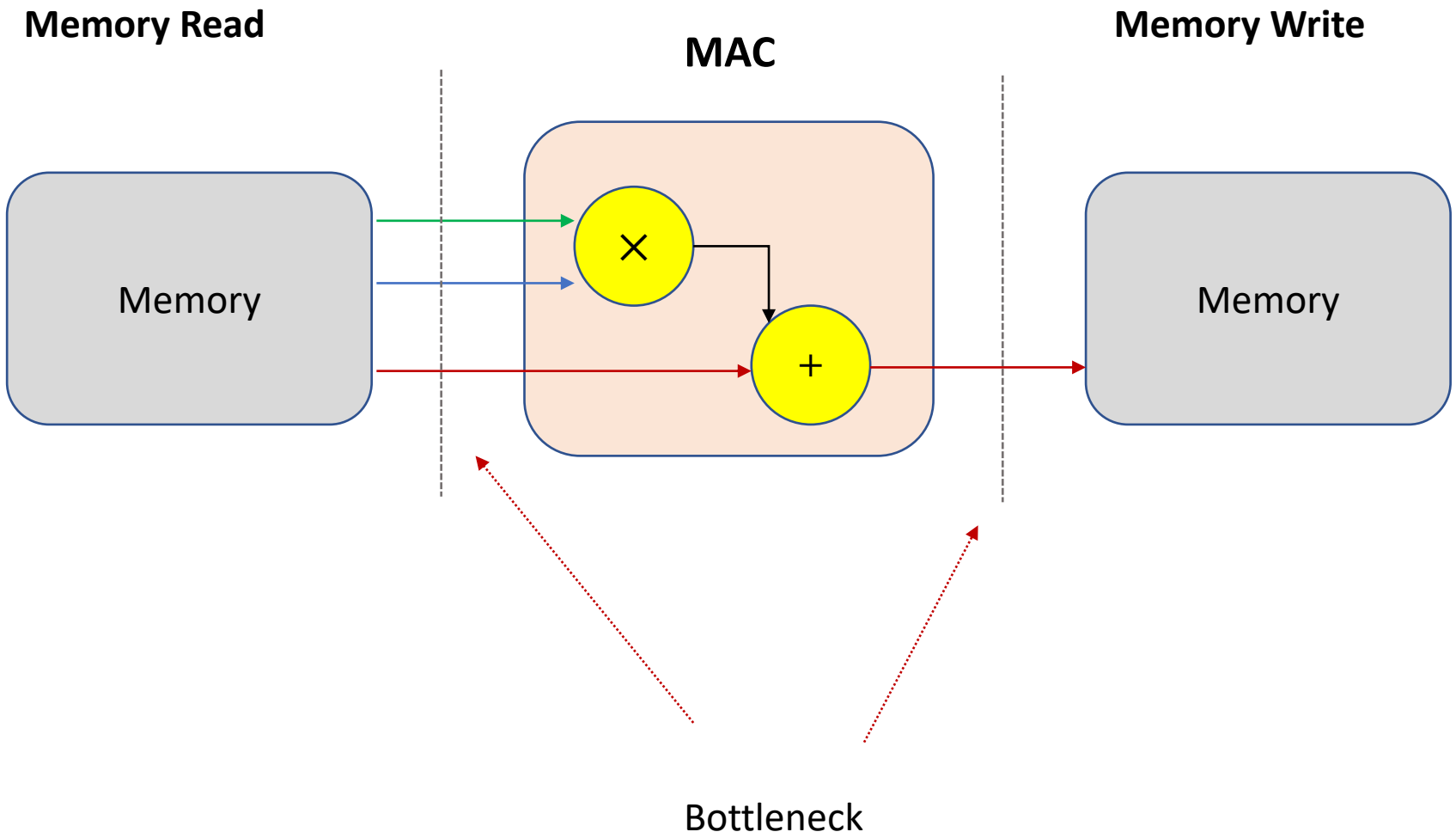
Memory Access is Expensive



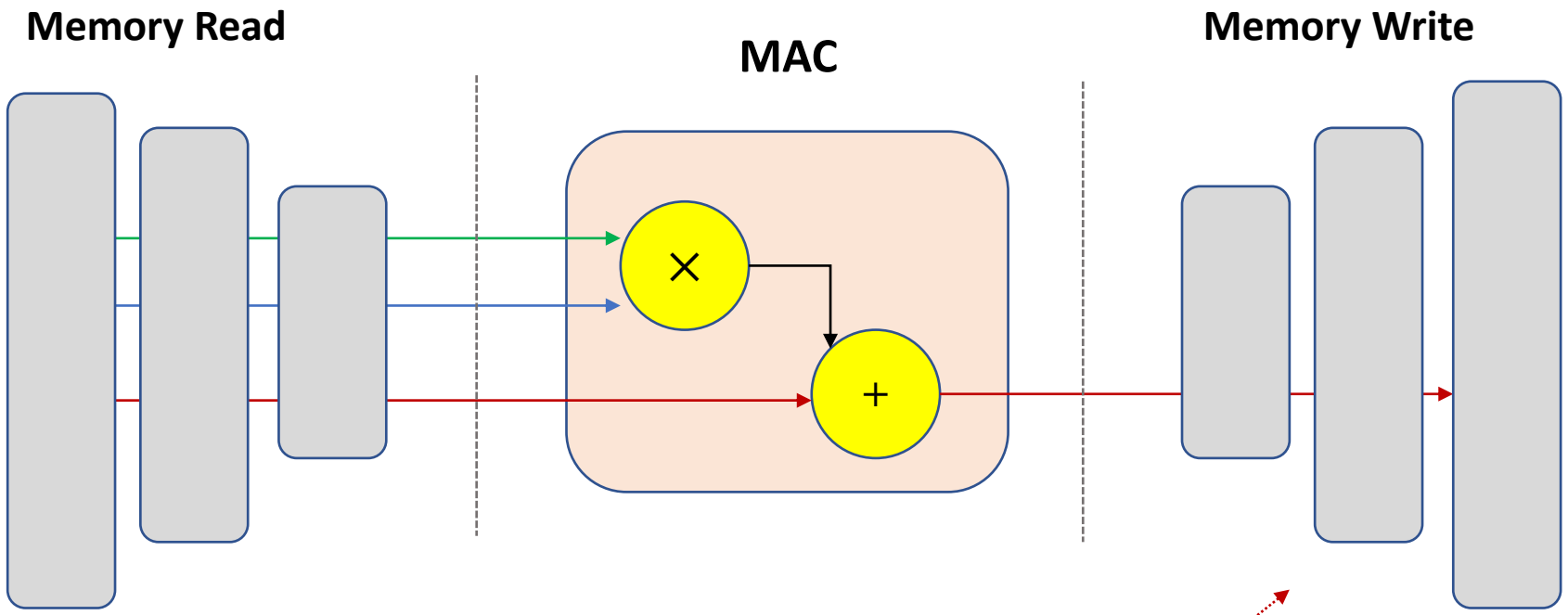
Memory Access is Expensive



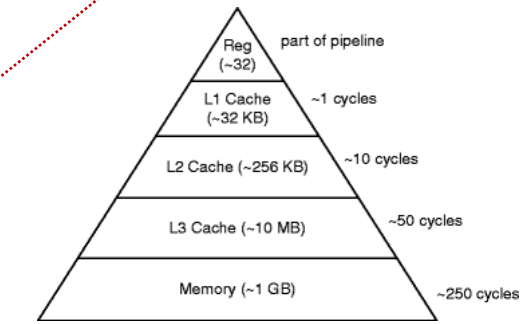
Memory Access is Expensive



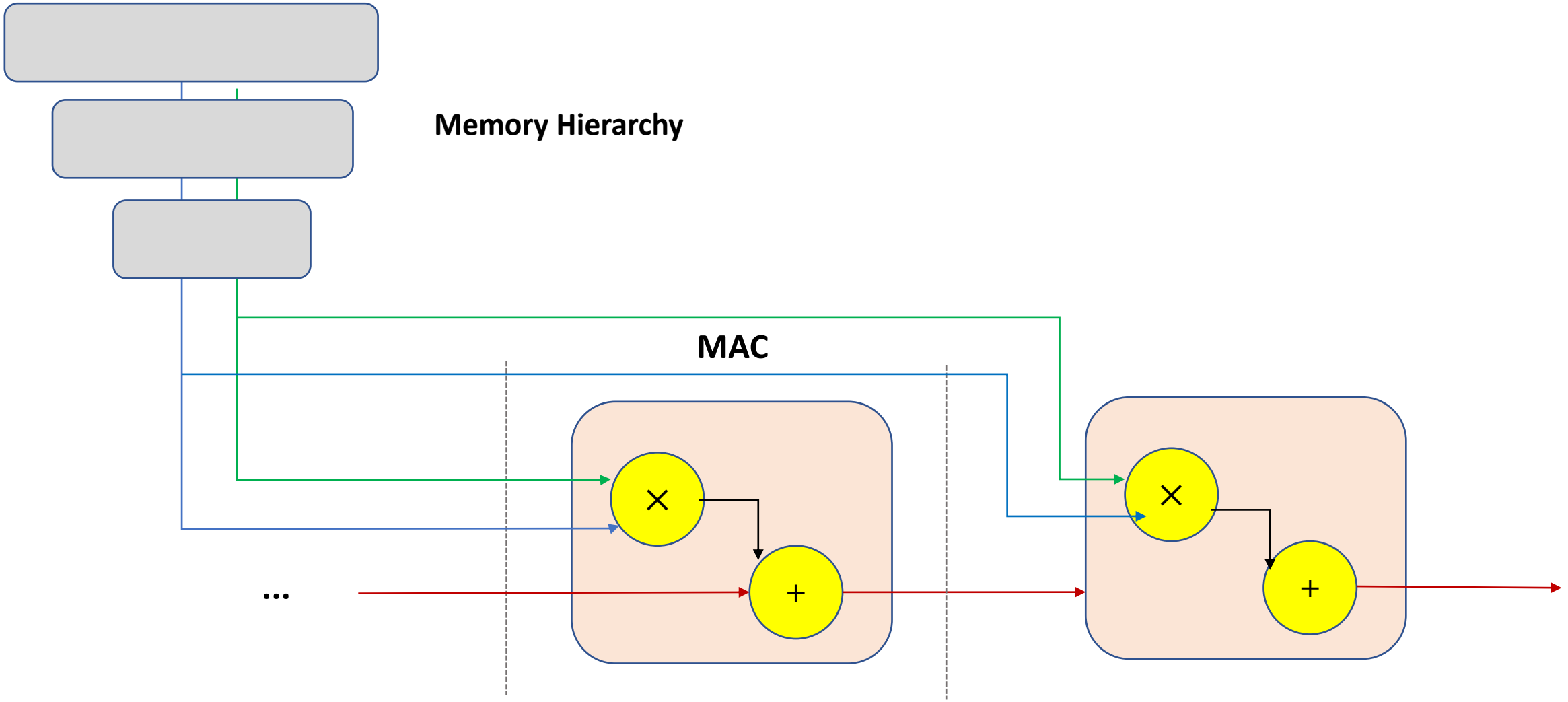
Locality: Memory Hierarchy



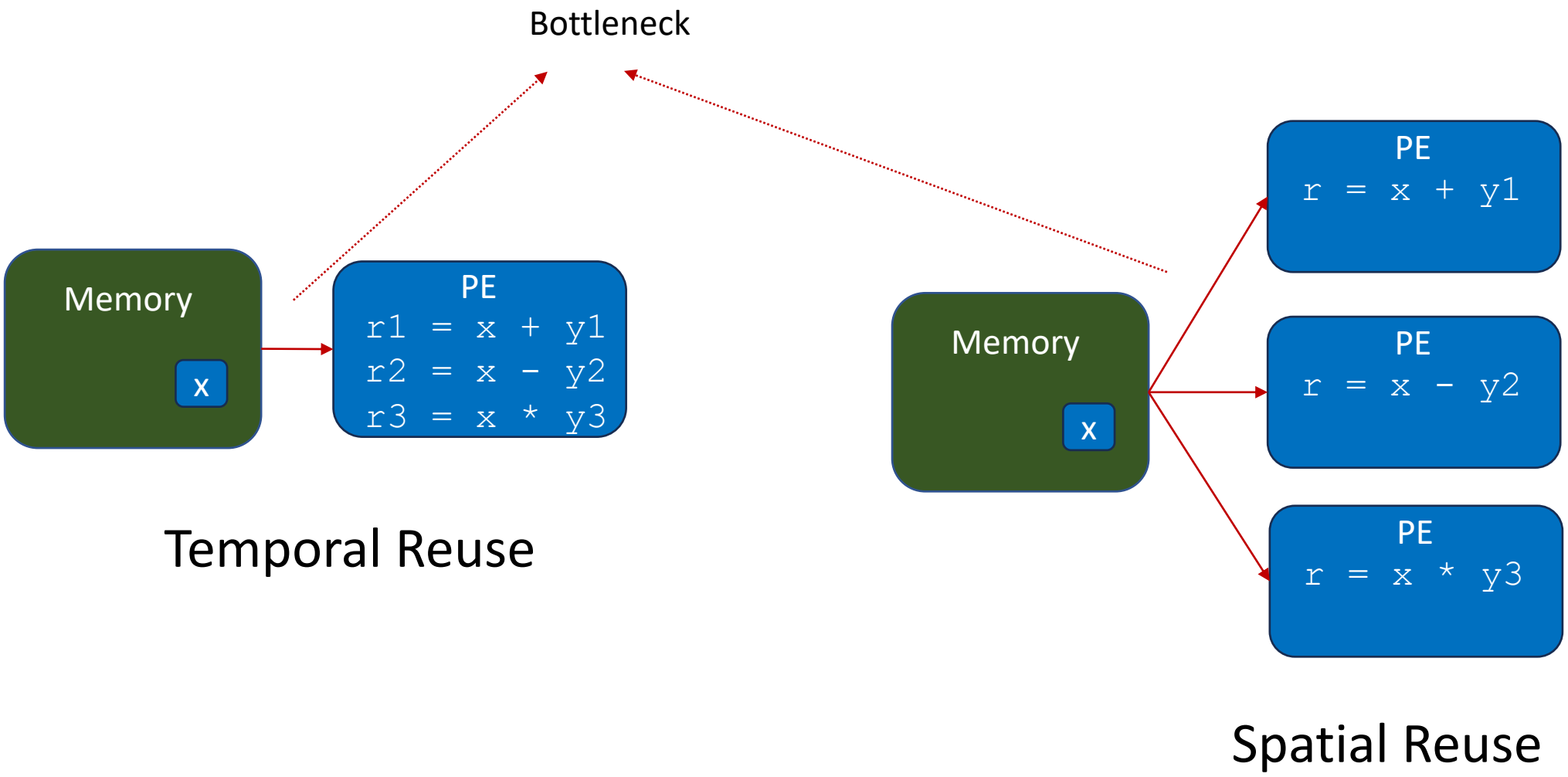
Memory Hierarchy



Reuse

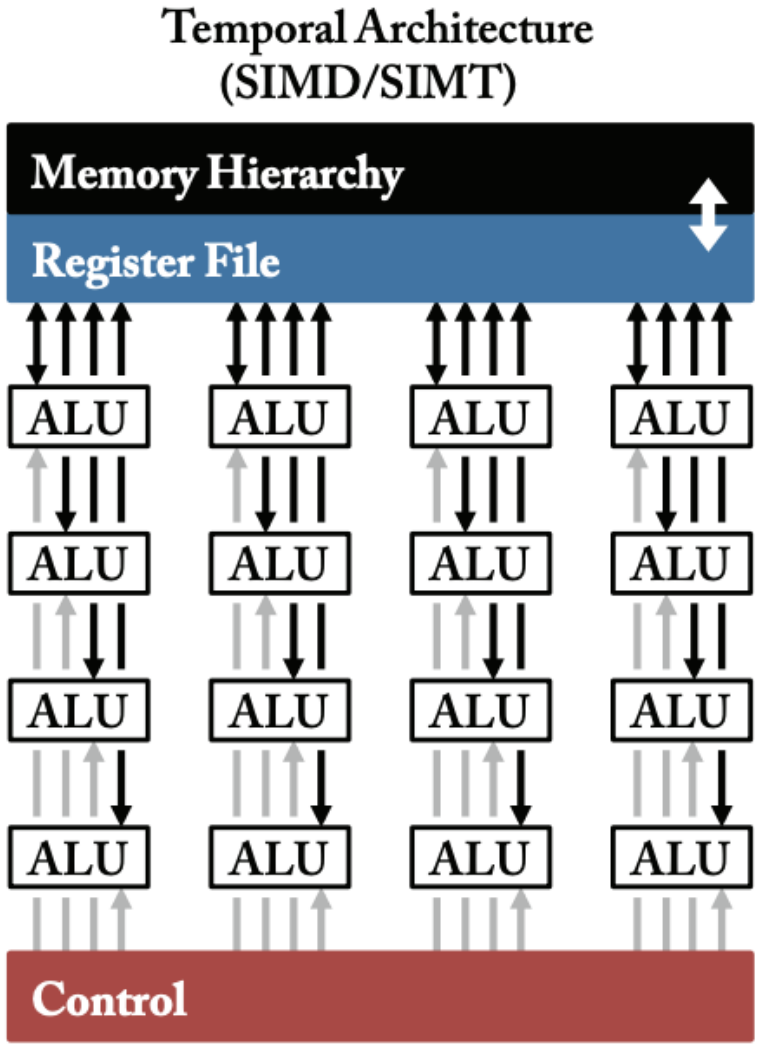


Reducing Memory Cost



Temporal vs. Spatial Architectures

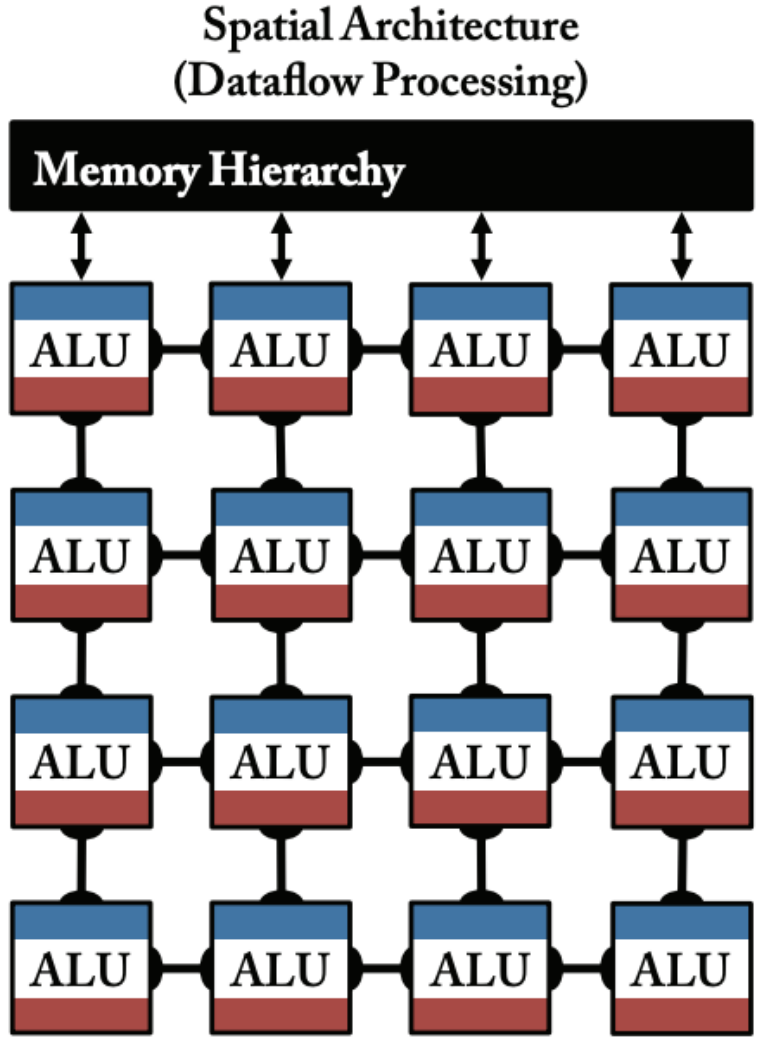
- **Temporal Architectures**:
  - **Centralized** control for **numerous** ALUs.
  - ALUs fetch data mainly from the memory hierarchy
  - **Limited** inter-ALU communication.
  - Typical in CPUs or GPUs.
  - Enhance parallelism through:
    - **Vector** instructions (e.g., **SIMD**).
    - Parallel **threads** (e.g., **SIMT** architectures).



Source: Sze et al.

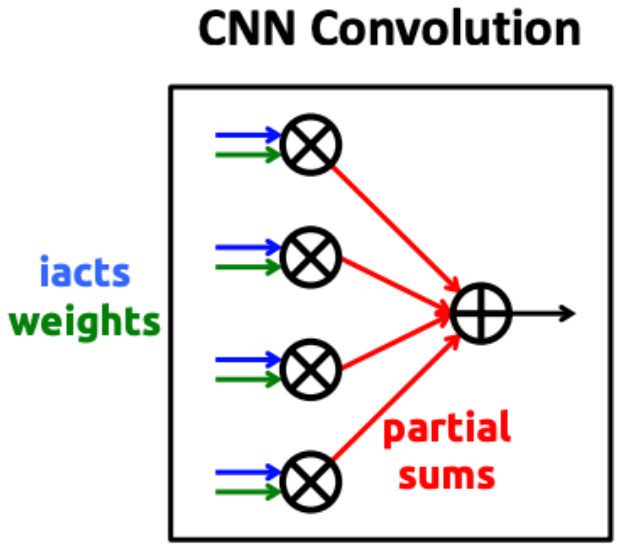
Temporal vs. Spatial Architectures

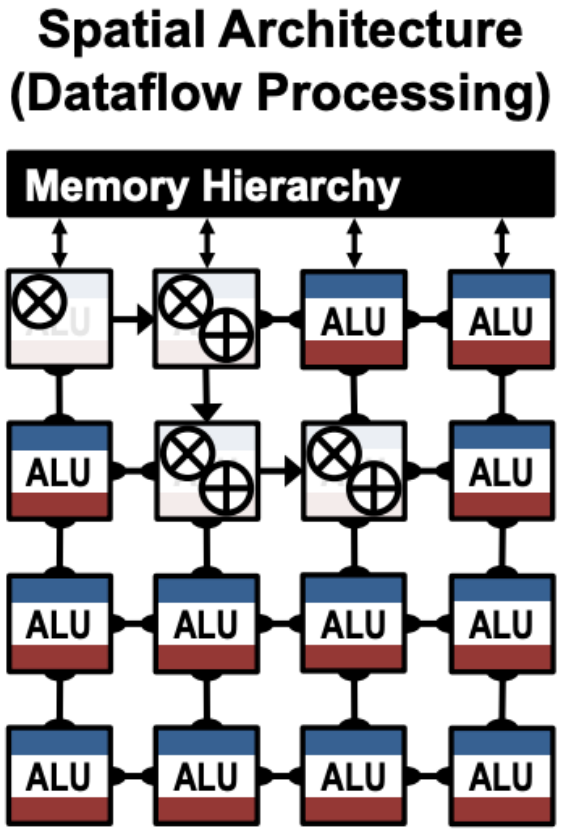
- **Spatial Architectures**:
  - Enables direct **ALU-to-ALU** communication.
  - Embraces **dataflow** processing.
  - ALUs can have **independent control logic** and **local memory**.
  - ALU + local memory = **Processing Engine (PE)**.
  - Common in ASIC- and FPGA-based DNN designs.



Source: Sze et al.

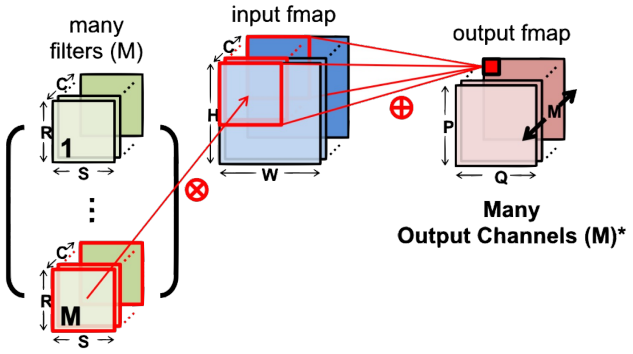
Spatial Architectures

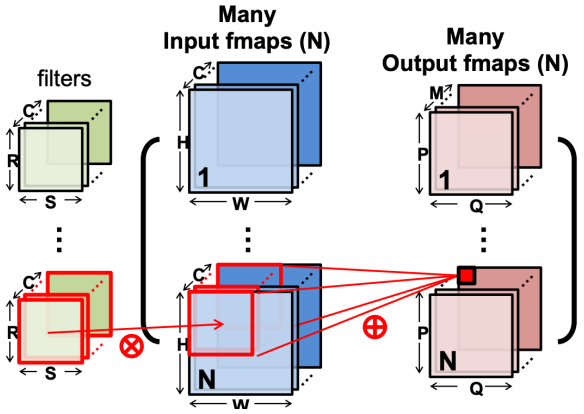


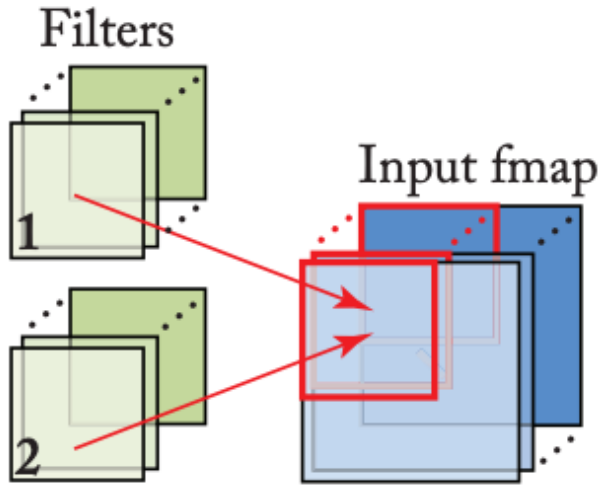


Source: Sze et al.

Input Feature Map Reuse

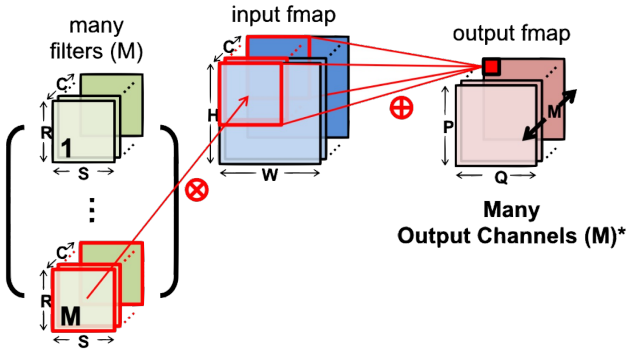


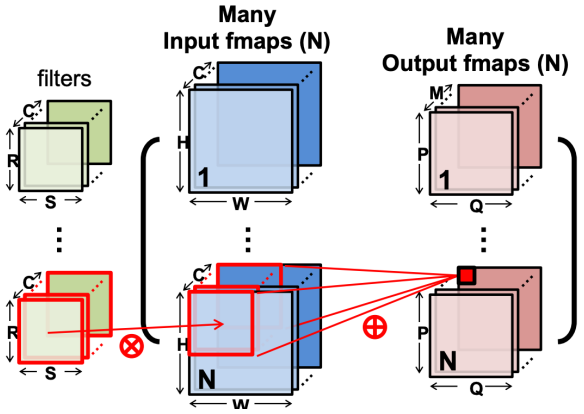


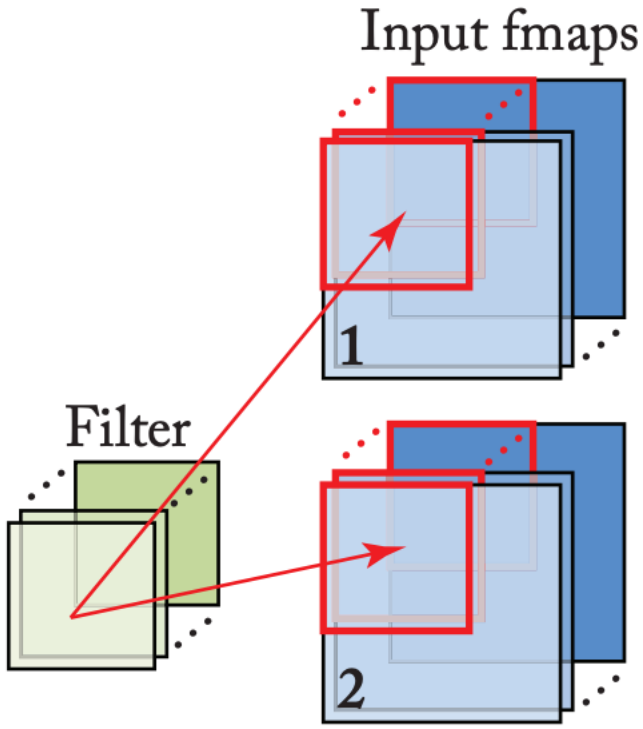


Multiple filters applied on the same IFM

Filter Reuse

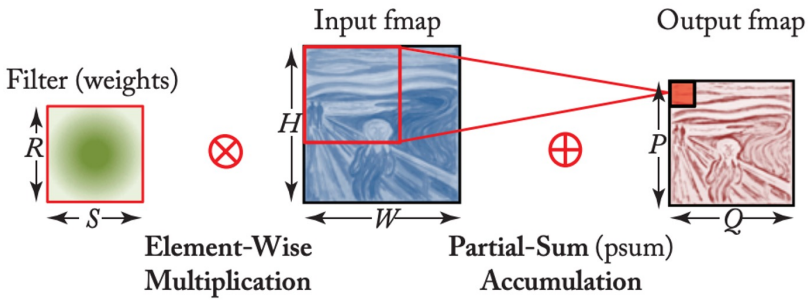


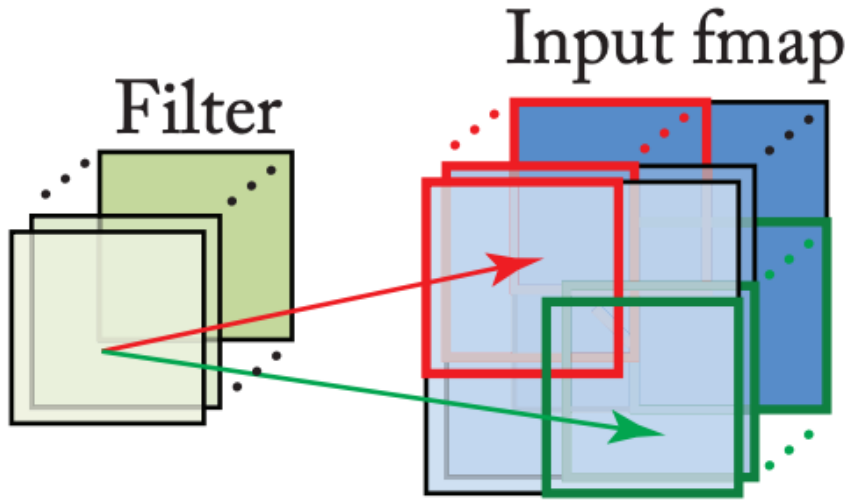




The same filter applied on multiple IFM

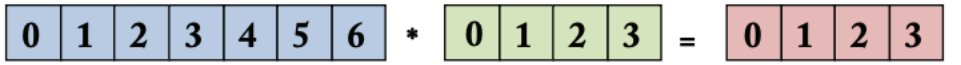
Convolution Reuse



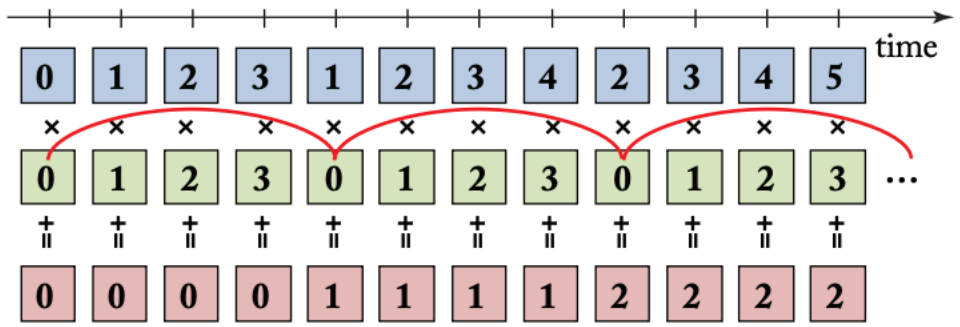


- Reuse both activations and filter weights
  - Each weight in the filter is used $H \times W$ times
  - Each element in the IFM is used $R \times S$ times

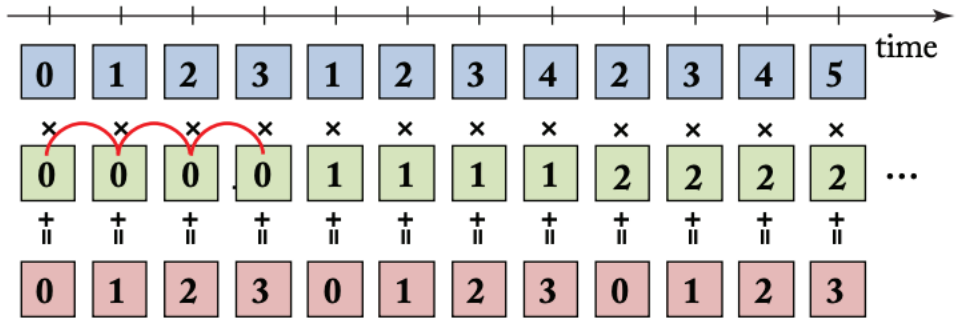
Reuse Distance



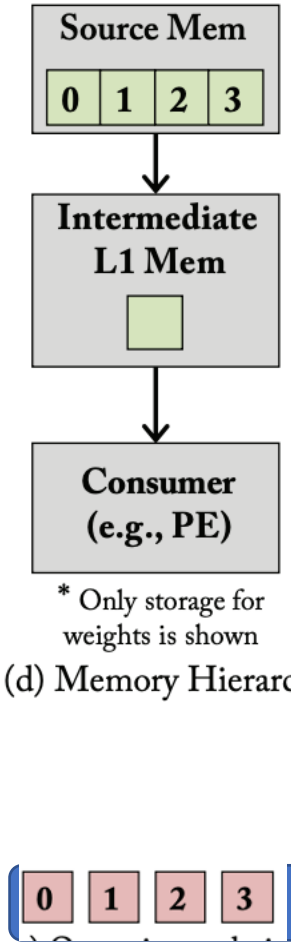
(a) Example 1D convolution



(b) Operation ordering 1: weight reuse distance = 4



(c) Operation ordering 2: weight reuse distance = 1

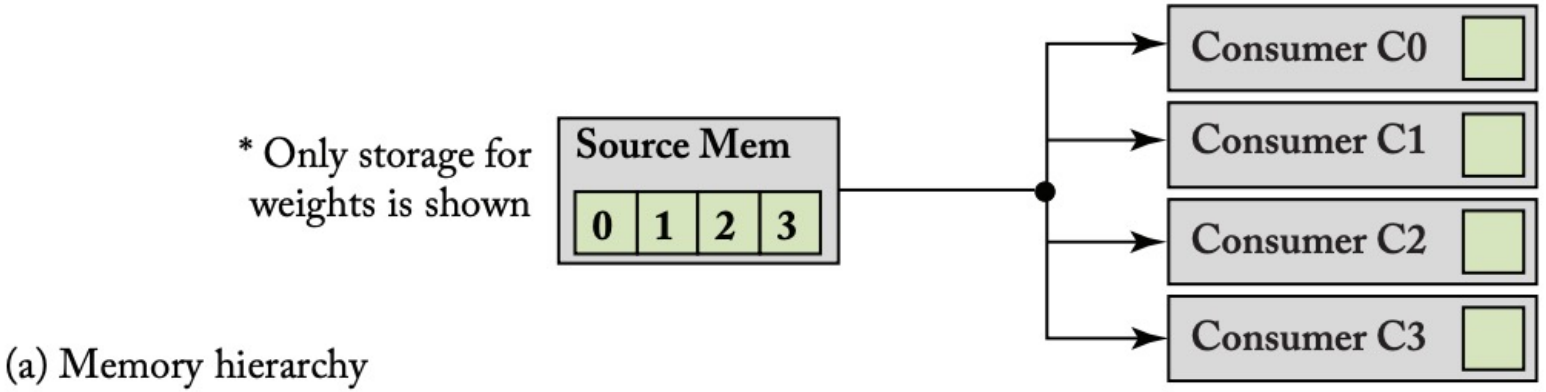


(d) Memory Hierarchy

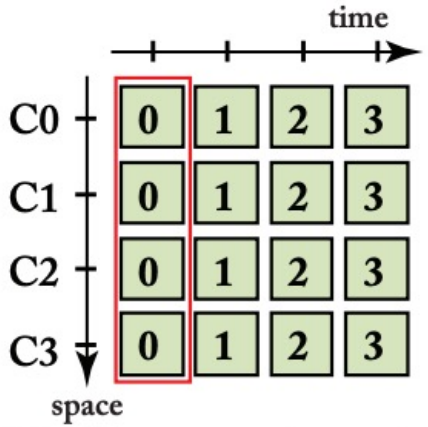
Reuse distance: the number of data accesses required by the consumer in between the accesses to the same data value, which is a function of the ordering of operations.

Spatial Reuse

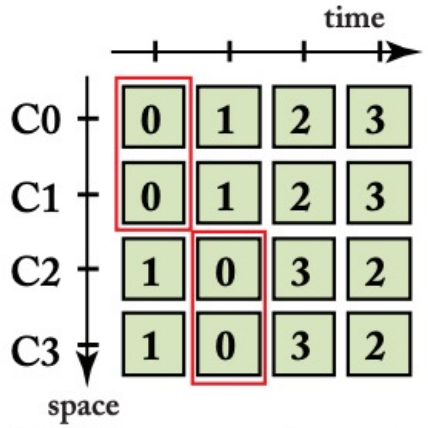
(a) Example 1D convolution



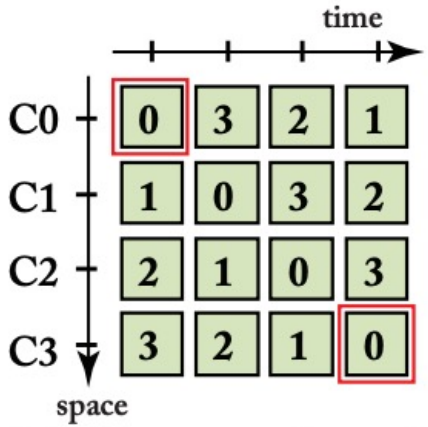
(a) Memory hierarchy



(b) Operation ordering 1

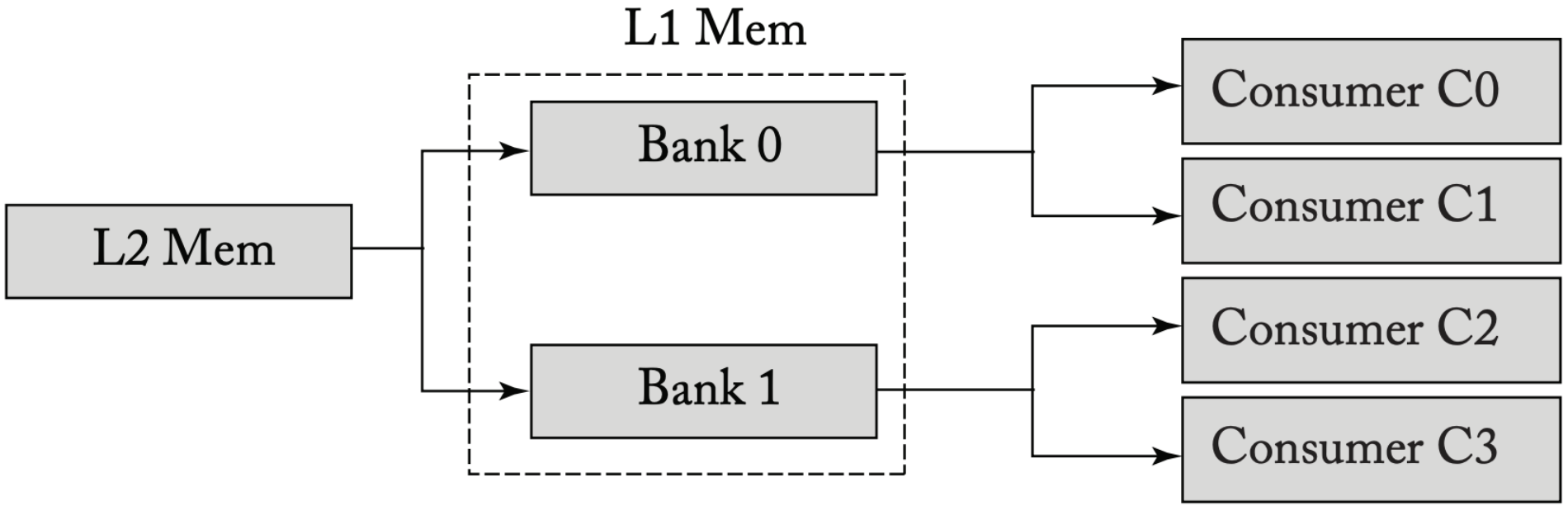


(c) Operation ordering 2



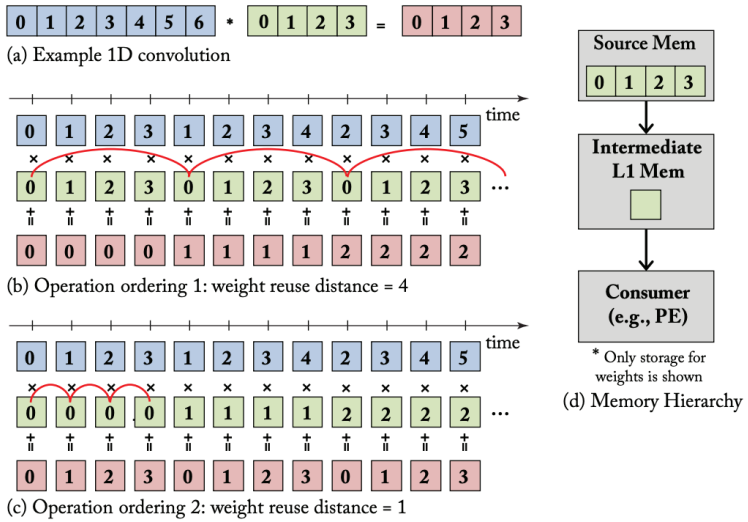
(d) Operation ordering 3

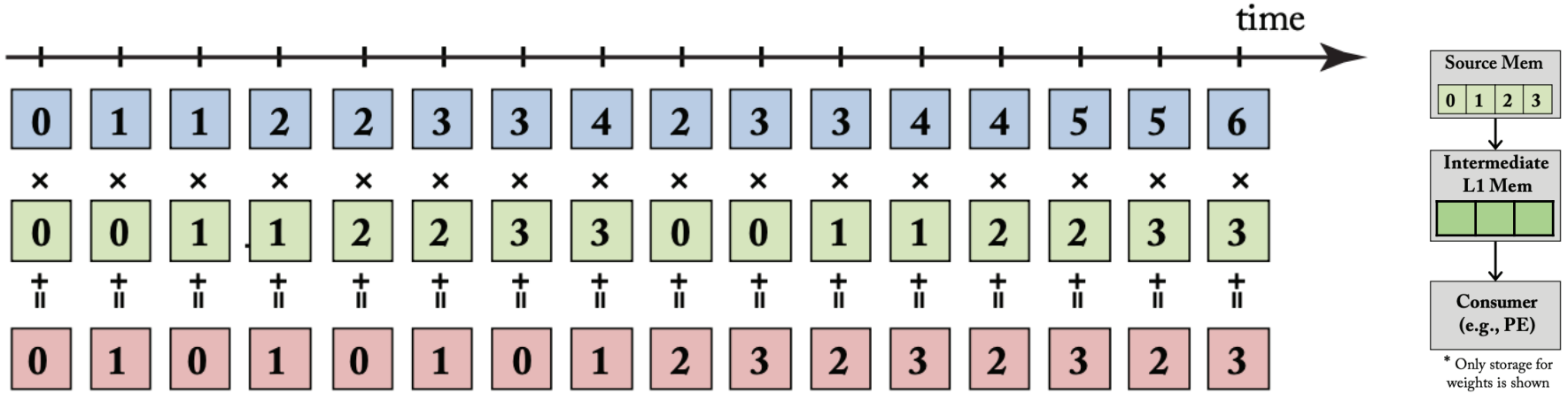
Spatial Reuse In NoC



To multicast from L2, we have to first multicast to L1.
This means the same data is written in both banks of L1, which is not efficient.

Reducing Reuse Distance with Temporal Tiling





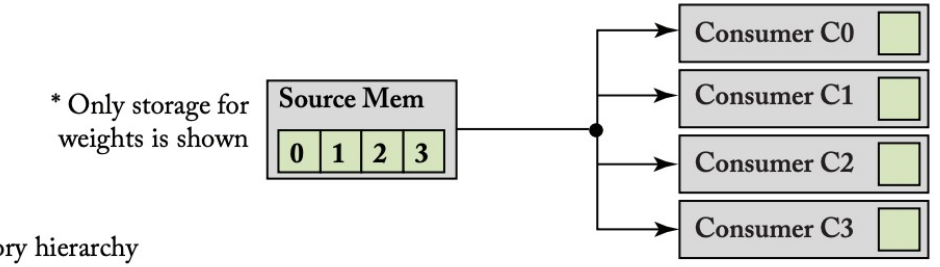
Temporal tiling: reducing the reuse distance to make it smaller than the storage capacity of a certain memory level

one weight value and a tile of two partial sum values can fit into L1 to exploit temporal reuse

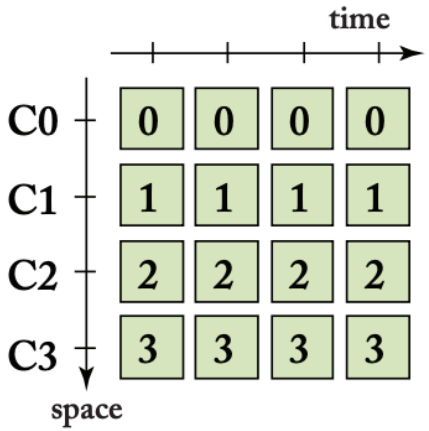
Spatial Reuse and Tiling

- **Reusing** the same data value by as many **consumers** as possible
- **Reducing the reuse distance** so that one multicast can serve as many consumers as possible.

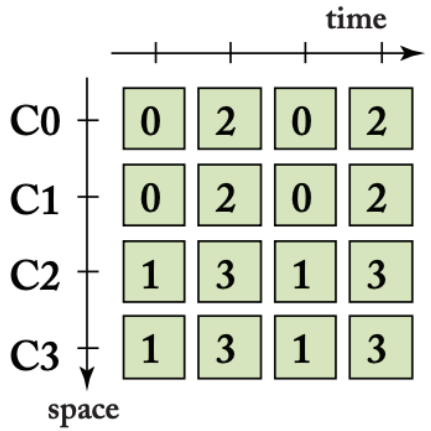
(a) Example 1D convolution



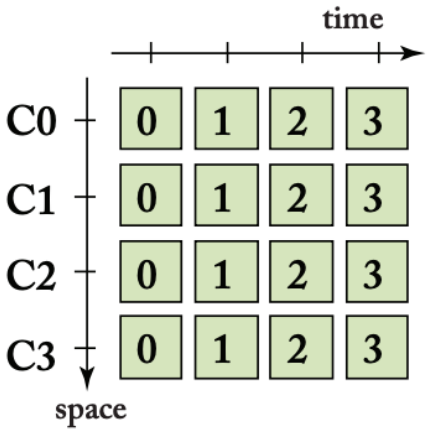
(a) Memory hierarchy



(a) No spatial reuse



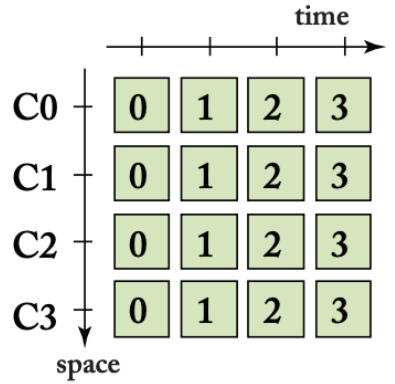
(b) Medium degree of spatial reuse

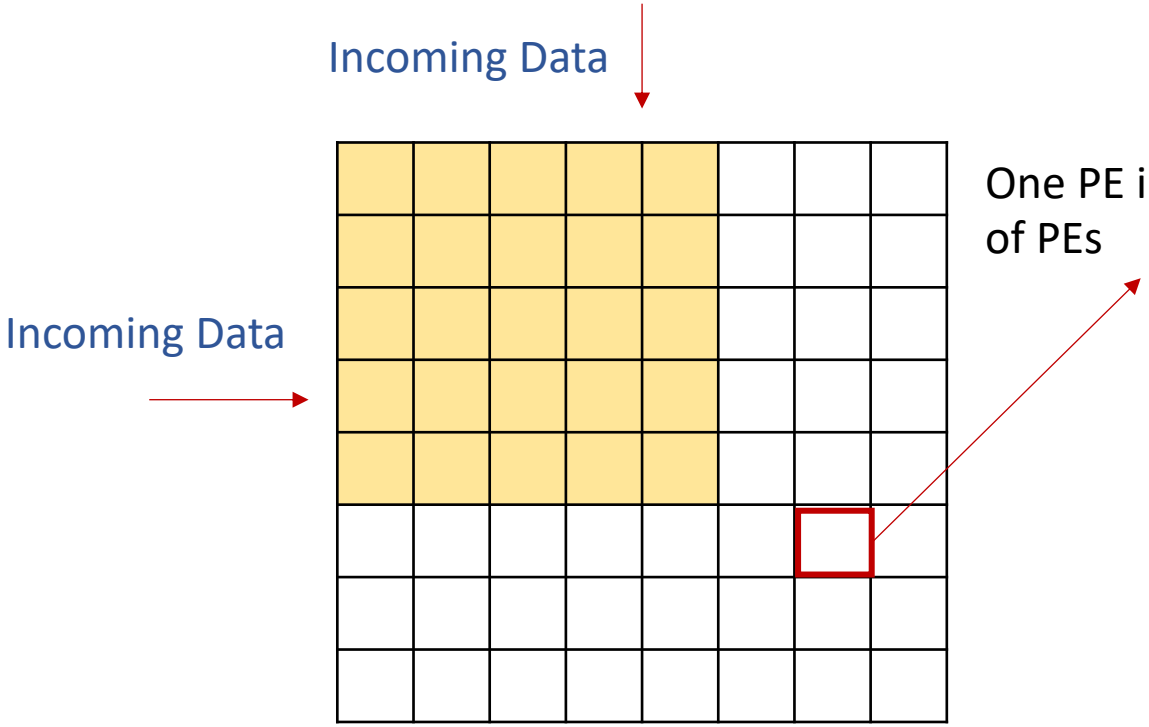


(c) High degree of spatial reuse

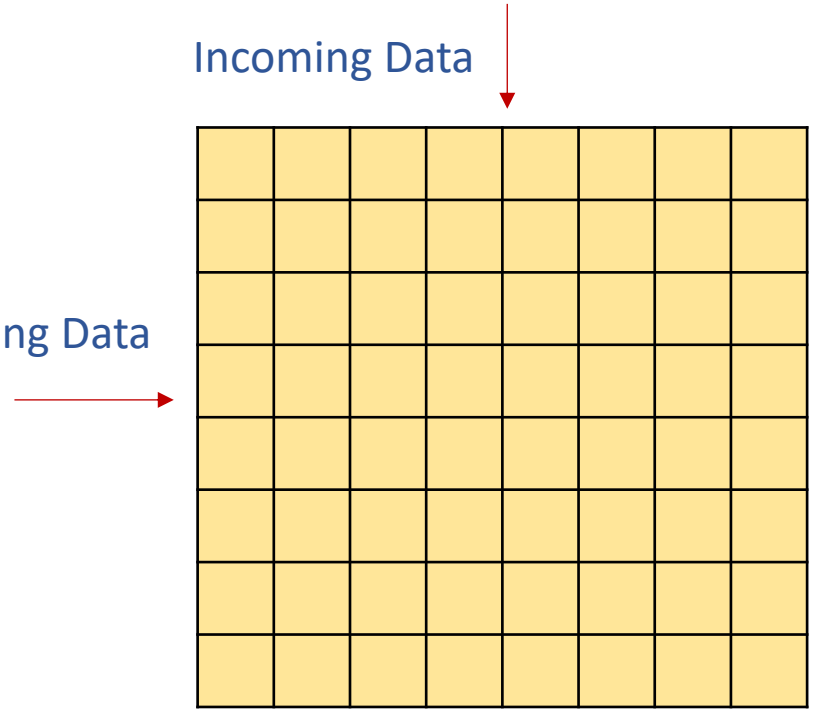
Spatial Tiling Issues

- **Mismatched Tile Sizes**: Risk of inactive PEs if tile dimensions don't align with DNN layers.
- **Boundary Conditions**: Potential underutilization at feature map edges.
- **Memory Bottlenecks**: Idle PEs due to data fetch/store delays.
- **Implementation Overheads**: Control complexities can offset tiling benefits.





Only some of PEs are utilized



All PEs are utilized