EE290C - Fall 2018

Advanced Topics in Circuit Design

VLSI Signal Processing

Lecture 19: Neural Networks and Systolic Arrays

Announcements

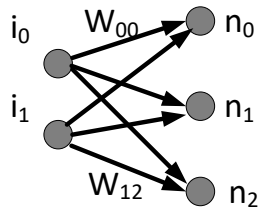
- **Chisel Community Conference**
  - Impressions?
- **Tape-in today**
  - We will review (briefly) on Tuesday
  - 5-min presentations by each group
  - 15-min meetings starting at 3:30
- **Final design due on Nov 30**
  - Final presentations on Nov 6

Outline

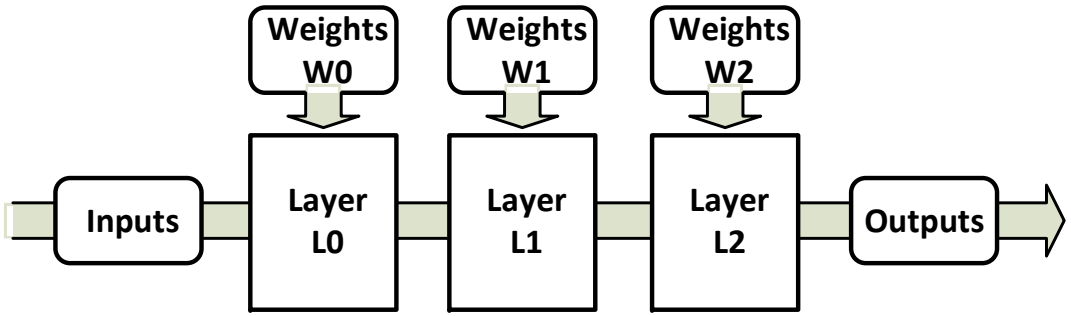
- **Neural networks**
  - Design of neural network accelerators
  - Systolic arrays
- **Reading:**
  - V. Sze, et al, Proc. IEEE, 12/17
  - M. Blott, HotChips 2018.
  - X. Yang, et al, arxiv 1809.04070

Convolutional Neural Networks

- **CNNs are typically feed-forward graphs**
  - One or more layers – up to thousands
- **Each layer has neurons n_i, interconnected with synapses, each with a weight w_{ij}**
- **Each neuron computes**
  - Linear transform (dot-product)
  - Followed by a non-linear activation function

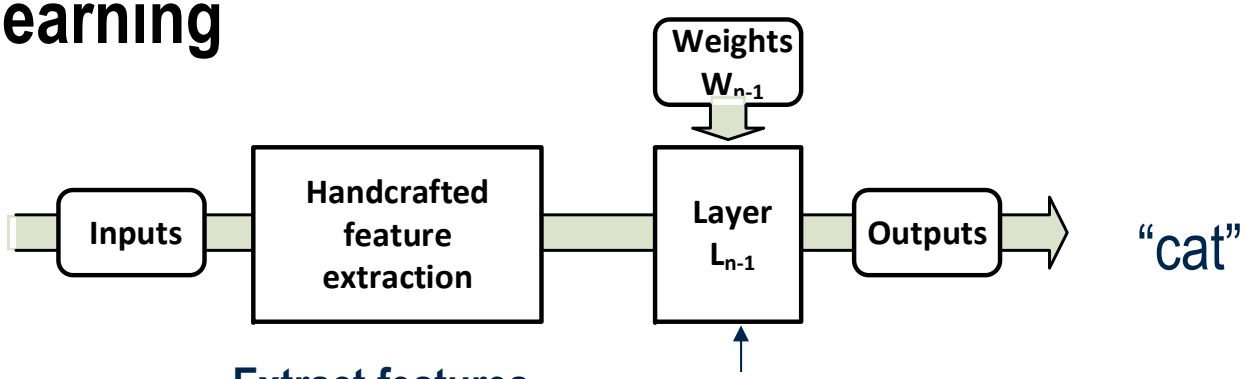


$$n_0 = Act(w_{00}*i_0 + w_{10}*i_1)$$

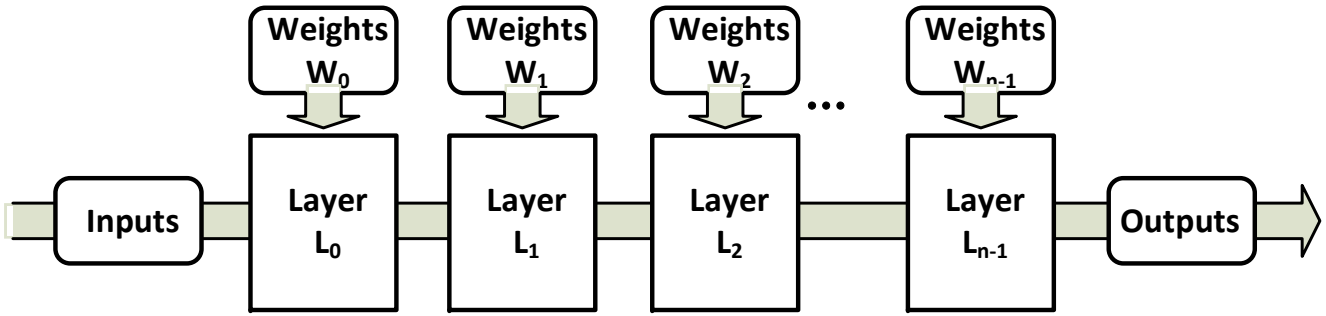


NN Evolution

- ‘Shallow’ learning



- ‘Deep’ learning



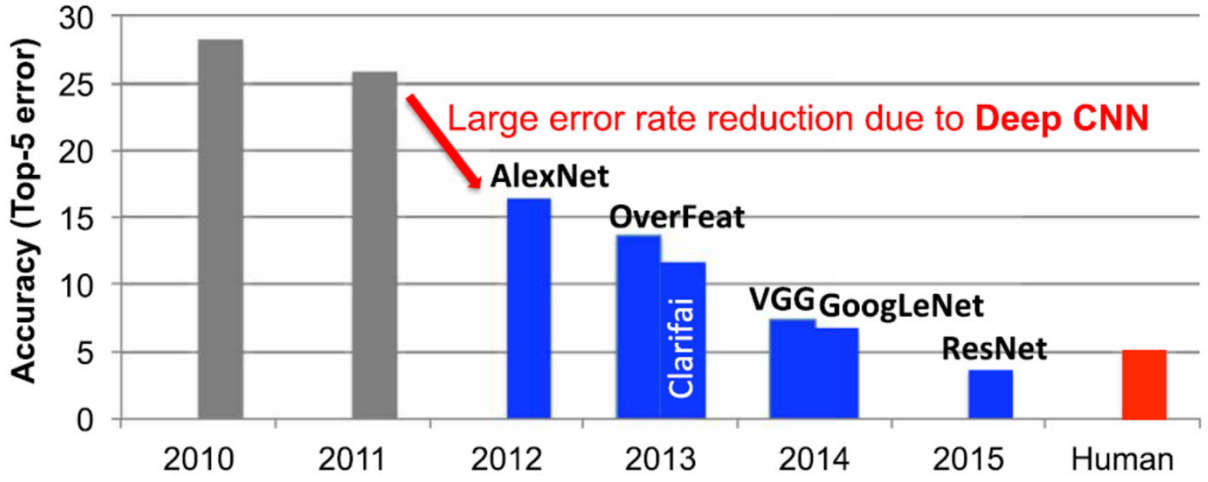
‘Learn’ low-level to high-level features

Convolutional Neural Networks

- **NNs are an "universal approximation function"**
- **If one make it big/deep enough and train it enough**
  - Can outperform humans on certain tasks
  - Outperforms traditional image classification/object detection algorithms
- **Requires little or no domain expertise**
  - Just train it and it works
- **'Swiss Army knife' of algorithms**

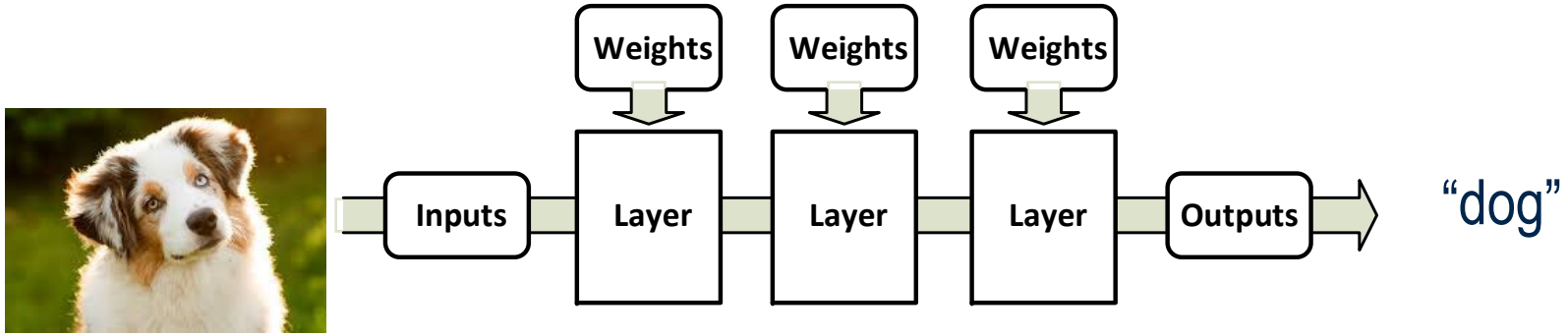
ImageNet Performance

- **Image classification among 1000 object categories**
  - Training set: 1.2M images
- **Step improvement after a series of incremental improvements**



Example: ResNet50

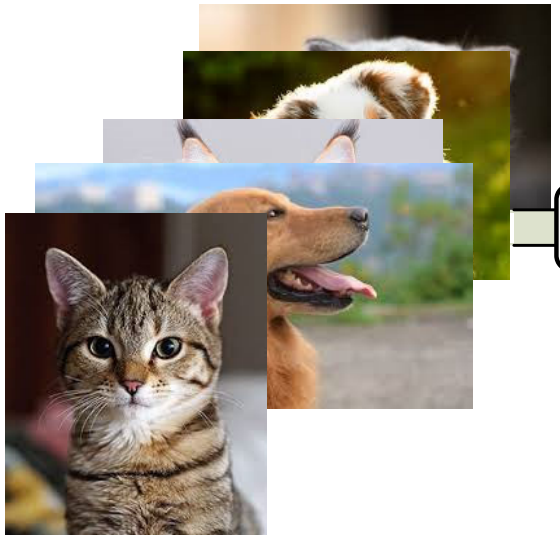
- Inference on ResNet50

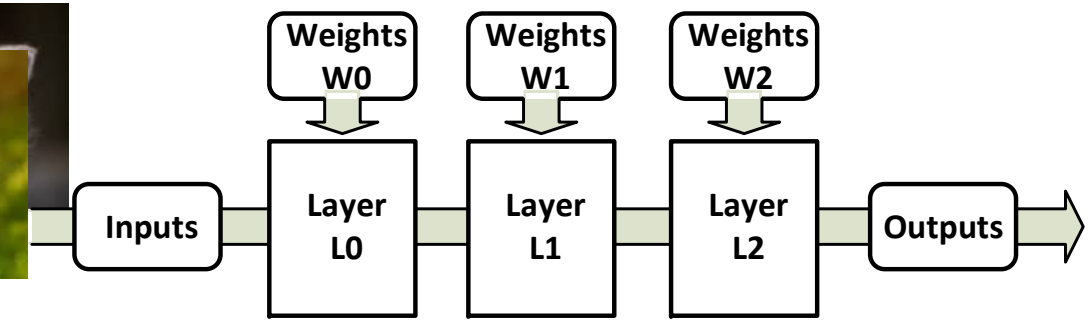


For ResNet50:
70 layers
7.7 billion operations
25.5Mbytes of weight storage*
10.1 Mbytes for activations*
Assuming int8

Training -> Inference

Training dataset

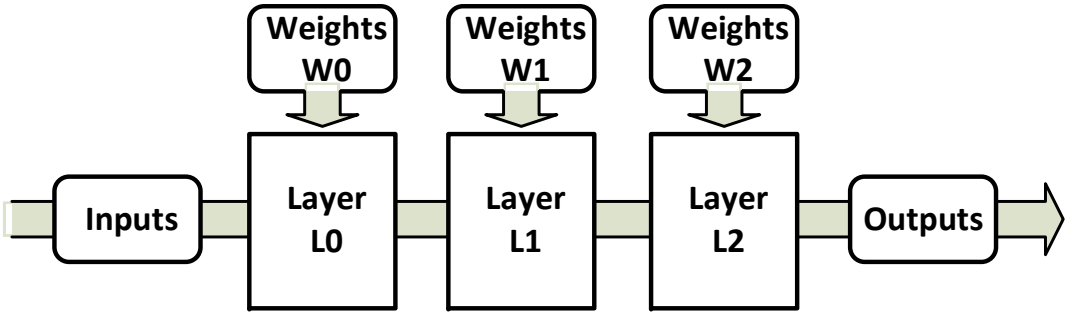




labels

"cat"
"dog"

Training: Machine 'learns' by optimizing weights from labeled data
In the **cloud**

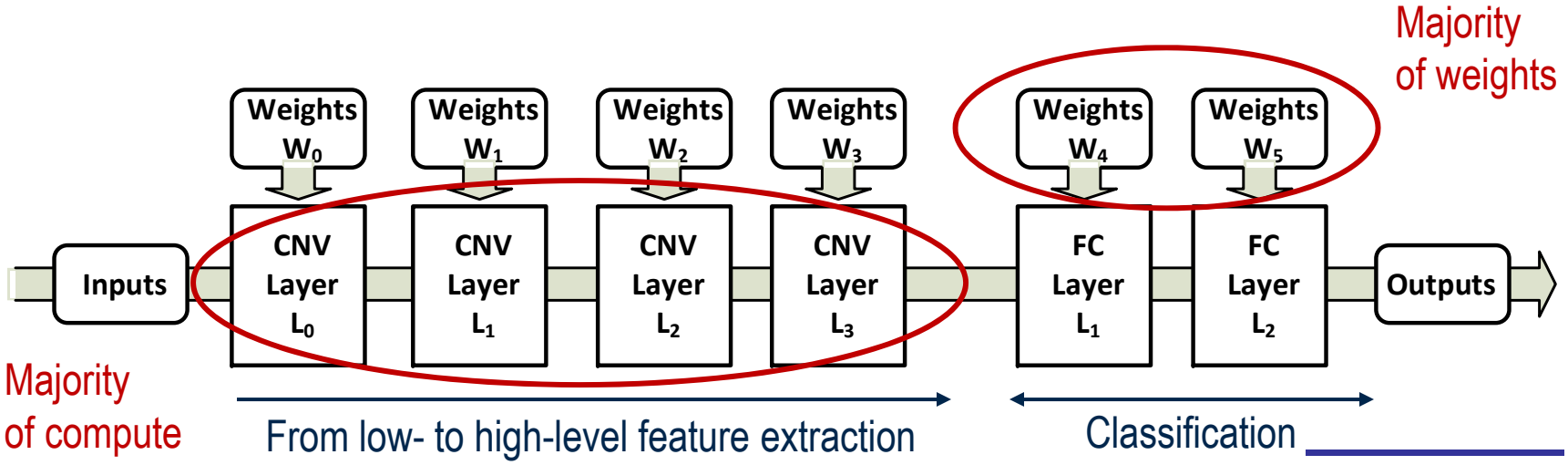


"cat"

Inference: Estimates outcomes from new data
In the **cloud** or at the **edge**

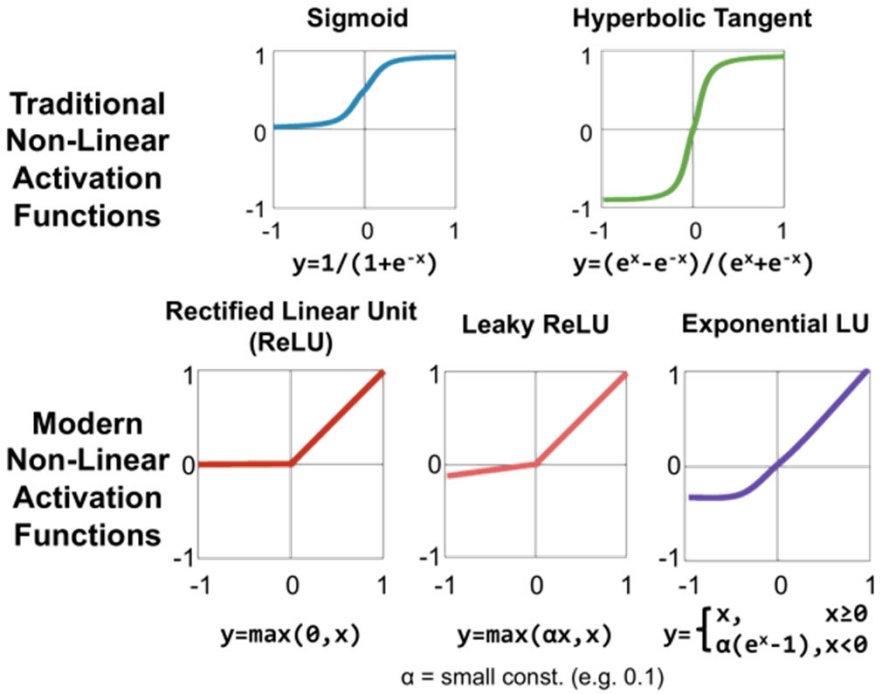
Neural Networks in More Detail

- **Basic components**
  - Activations
  - Fully connected layer (FC) (are increasingly replaced due to their memory intensive nature)
  - Convolutional layer (CNV)
  - Pooling (subsampling) layers (POOL)
  - Recurrent layers (RL)
  - Batch normalization
- **Popular meta layers**
  - Residual layers
  - Inception layers

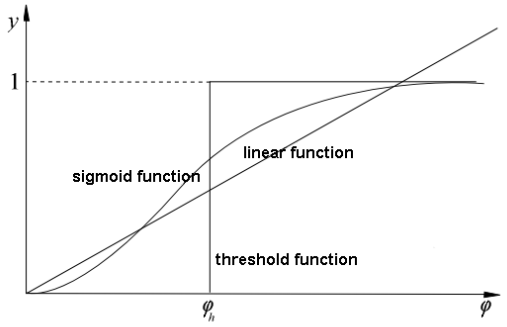


Activation Functions

- **Fires when input is greater than a threshold**
  - Non-linear so we can approximate more complex functions
- **Most popular for CNN: rectified linear unit (ReLU)**
  - Popular as it propagates gradients better than bounded
  - However, recent work says as long as there is the proper initialization, it'll be fine even with bounded act. function*
- **Other common ones include: tanh, leaky ReLU, sigmoid**
- **For quantized neural networks threshold functions are used**
  - Straight-through estimators for backpropagation

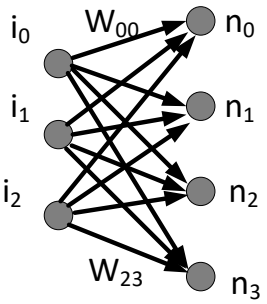


From Gabor Melli



Fully-Connected Layers

- **Each input activation is connected to every output activation**
  - Receptive field encompasses the full input
- **Can be written as a matrix-vector product with an element-wise non-linearity applied afterwards**
- **Implementation Challenges**
  - High weight memory requirement: #IN * #OUT * BITS
  - Low arithmetic intensity: 2* #IN* #OUT / #IN * #OUT * BITS/8



Inner product/Dense layers

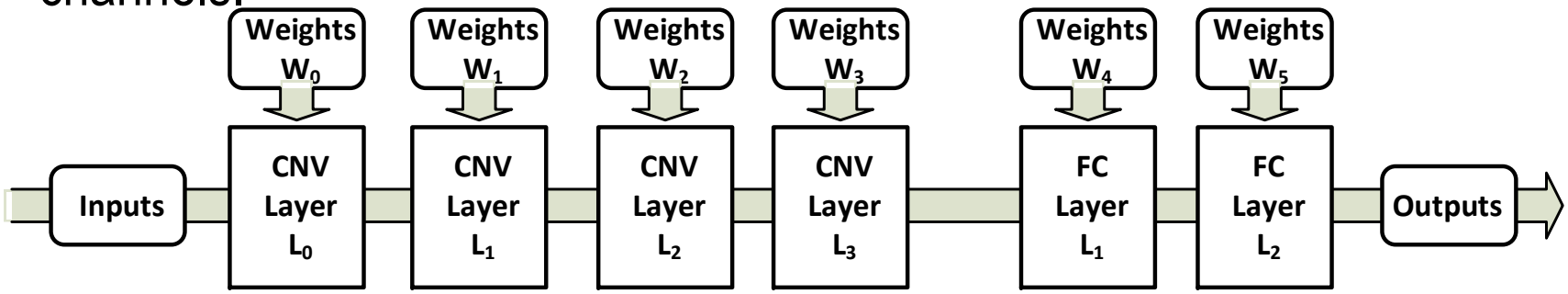
IN: number of input channels
OUT: number of output channels
BITS: bit precision in data types

$$(i_0 \quad i_1 \quad i_2) \times \begin{pmatrix} W_{00} & W_{01} & W_{02} & W_{03} \\ W_{10} & W_{11} & W_{12} & W_{13} \\ W_{20} & W_{21} & W_{22} & W_{23} \end{pmatrix} = (n_0' \quad n_1' \quad n_2' \quad n_3')$$

$$(n_0 \quad n_1 \quad n_2 \quad n_3) = Act(n_0' \quad n_1' \quad n_2' \quad n_3')$$

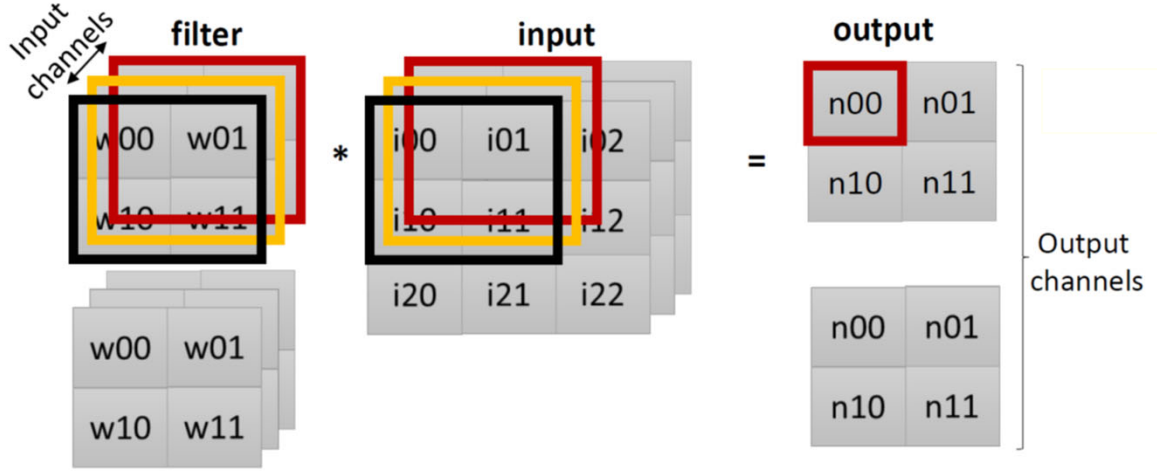
Convolutional Neural Networks

- Each of the CNV layers in CNNs is primarily composed of high-dimensional convolutions.
- Input activations of a layer are structured as a set of 2-D *input feature maps* (ifmaps), each of which is called a *channel*.
- Each channel is convolved with a distinct 2-D filter from the stack of filters, one for each channel;
- Stack of 2-D filters is often referred to as a single 3-D filter.
- The results of the convolution at each point are summed across all the channels.



Convolutional Layers

- **Example:**
  2D convolution



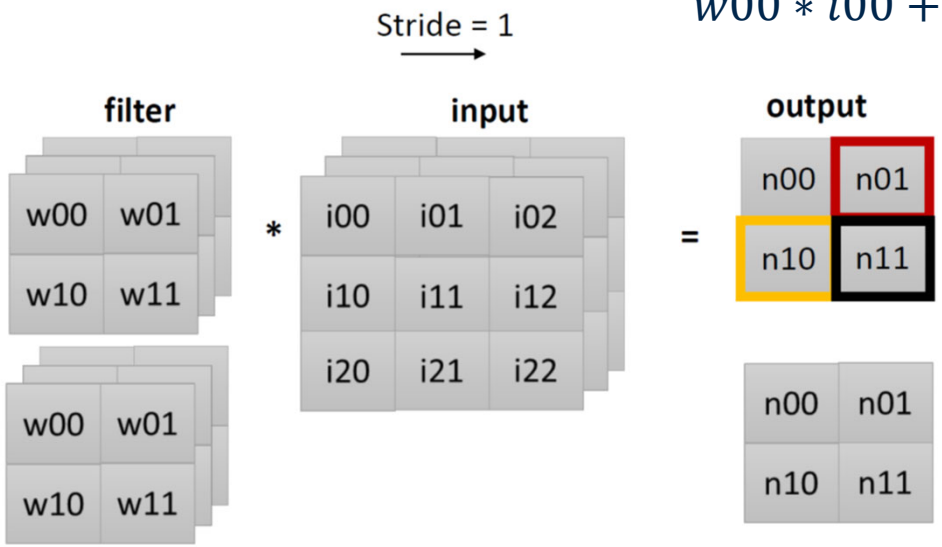
$$n00 = Act(w00 * i00 + w01 * i01 + w10 * i10 + w11 * i11 + w00 * i00 + w01 * i01 + w10 * i10 + w11 * i11 + w00 * i00 + w01 * i01 + w10 * i10 + w11 * i11)$$

- **Convolutions capture some kind of locality, spatial or temporal, in each domain**
- **Receptive field of each neuron reduced**
  - Applying convolution to all images in the previous layer
- **Weights represent the filters used for convolutions**

2D Convolutional Layers

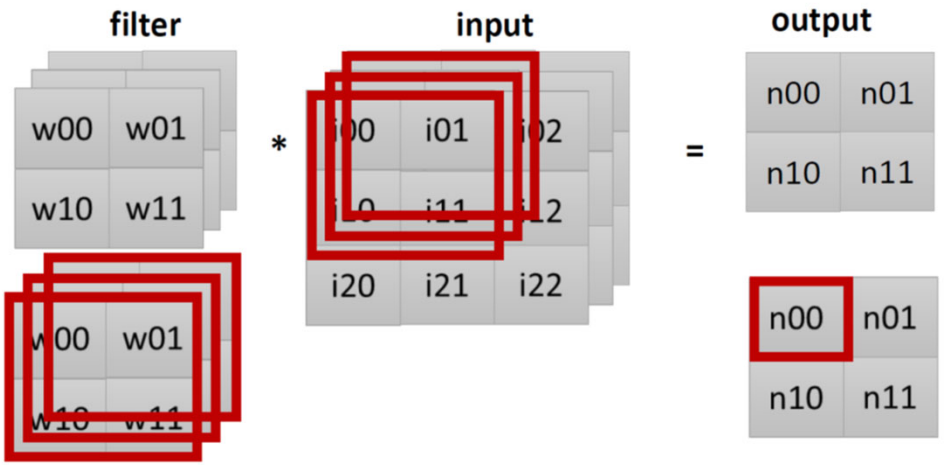
- **Slide the window till one feature map is complete**
  - With a given stride size

$$n01 = Act(w00 * i00 + w01 * i01 + w10 * i10 + w11 * i11 + w00 * i00 + w01 * i01 + w10 * i10 + w11 * i11 + w00 * i00 + w01 * i01 + w10 * i10 + w11 * i11)$$



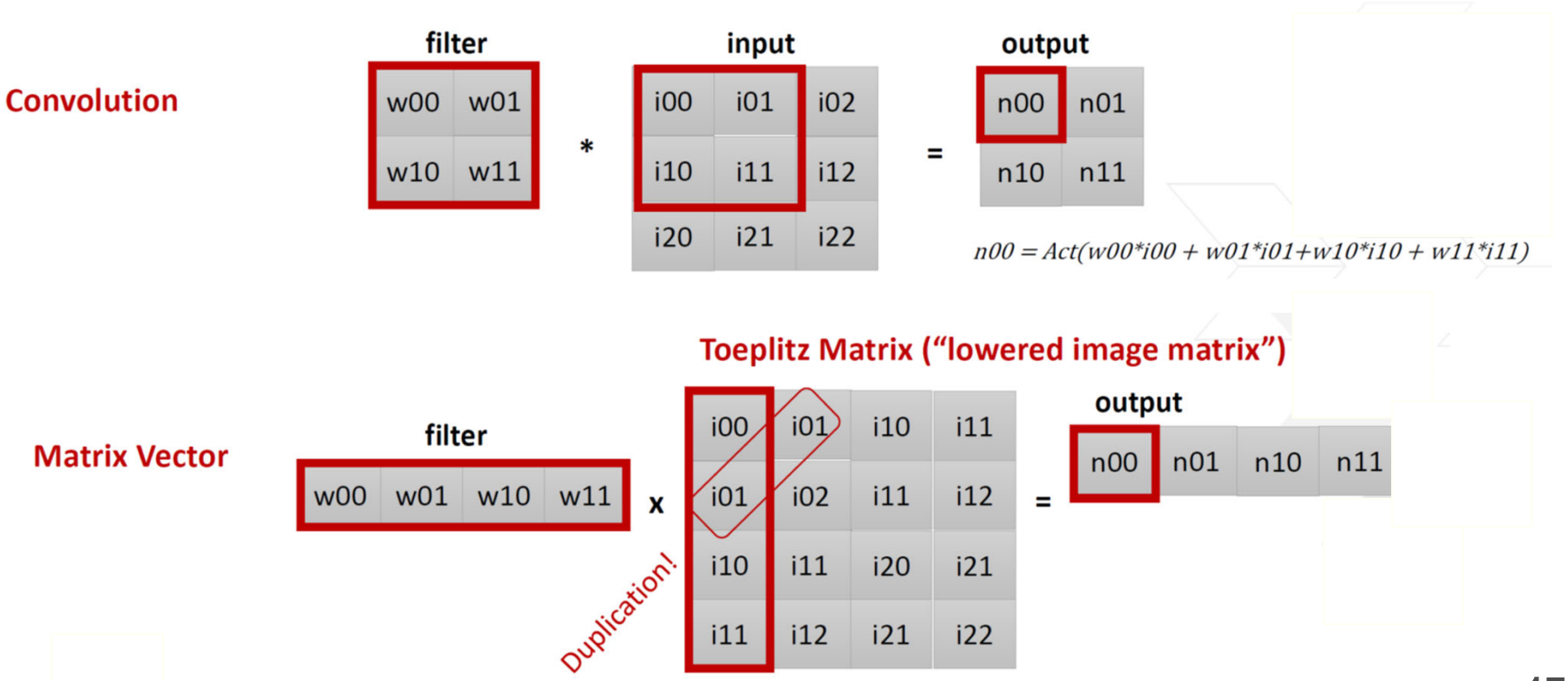
2D Convolutional Layers

- Compute next channel



2D Convolutional Layers

- **1 input and 1 output channel**
  - Lowered to a matrix-matrix multiply by using a Toeplitz matrix

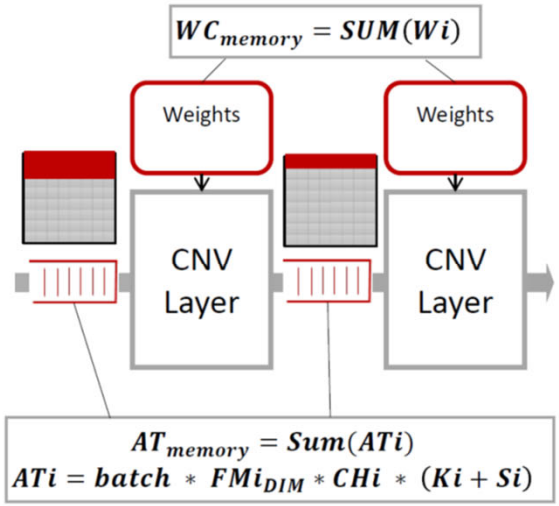


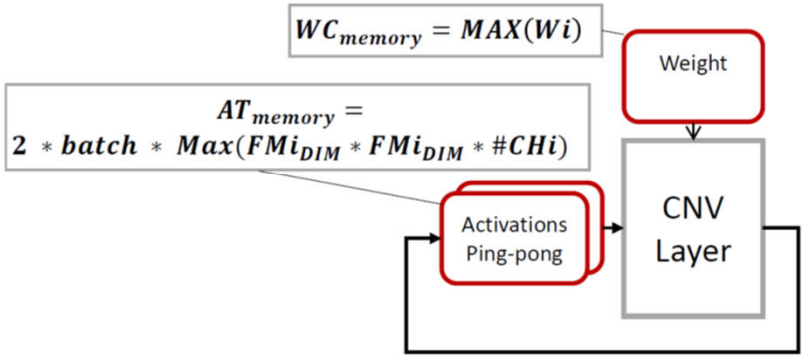
SDF vs. SA

- **Synchronous dataflow (SDF) vs. Systolic arrays (Matrices of processing elements, MPEs)**

Spectrum of options available

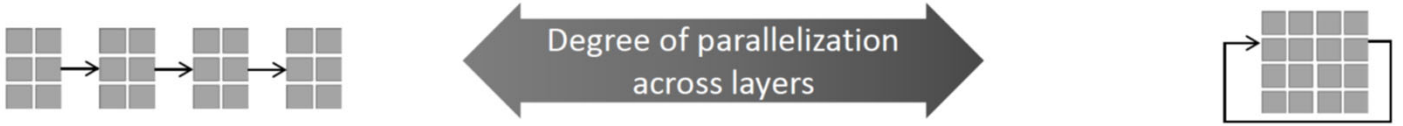
>> End points are pure layer-by-layer compute and feed-forward dataflow architecture, many intermediate options





SDF vs. SA

- **Tradeoffs between SDF and systolic arrays (MPEs)**



SDF uses task-level parallelism across layers

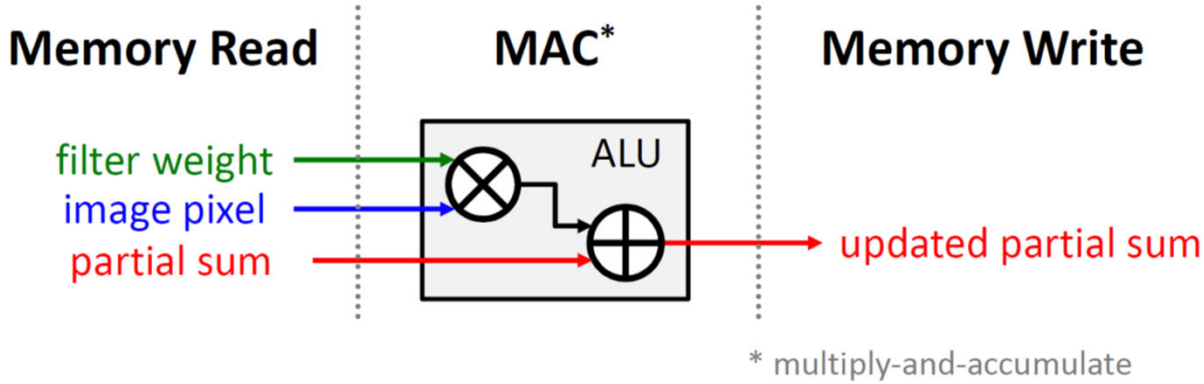
- Higher compute and memory efficiency due to custom-tailored hardware design
- Requires less activation buffering as compute can commence as soon as there is enough data for computing the next convolutional window
- Less latency (reduced buffering)
- No control flow (static schedule)

MPE (pure layer by layer compute)

- Compute efficiency is a scheduling problem
- Efficiency of memory for weights and activations depends on how well balanced the topology is
- Requires less on-chip weight memory, but more activation buffers
- Flexible hardware, which can scale to arbitrary large networks
- At the expense of generating sophisticated scheduling algorithms

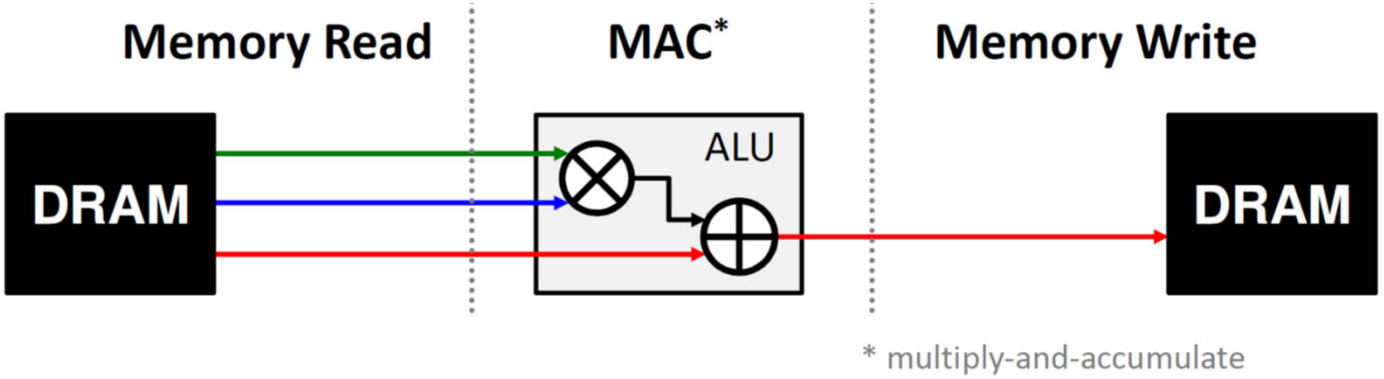
Computation and Memory

- Memory accesses are a bottleneck



Computation and Memory

- **Memory accesses are bottleneck**

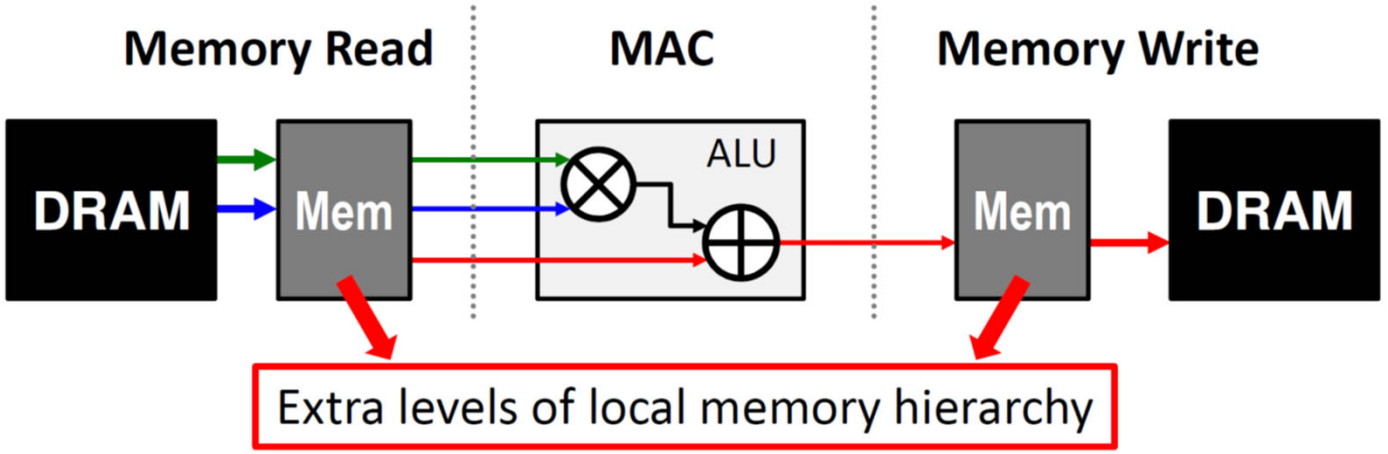


Worst Case: all memory R/W are **DRAM** accesses

- Example: AlexNet [NIPS 2012] has **724M** MACs
  → **2896M** DRAM accesses required

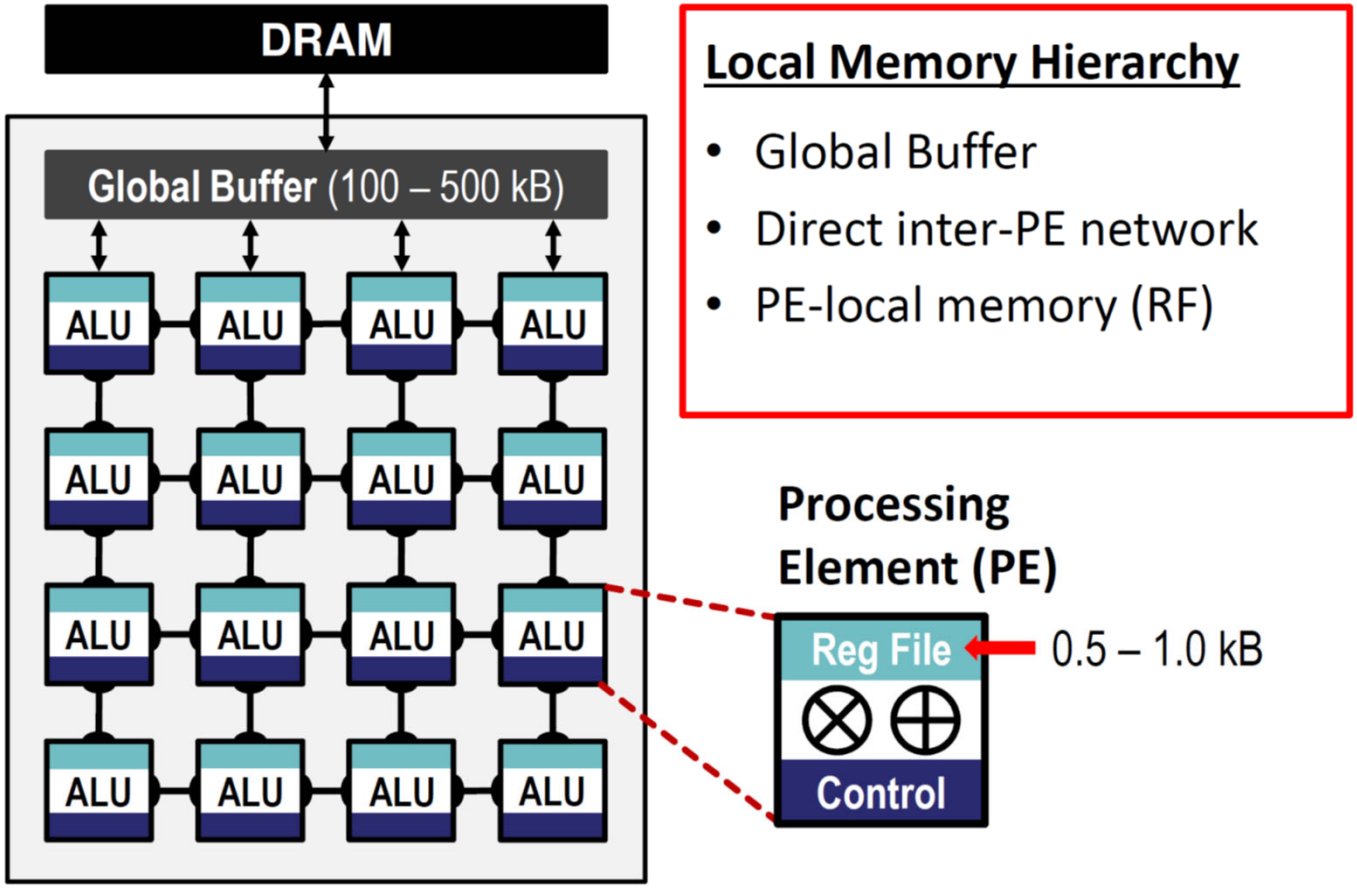
Computation and Memory

- **Add additional levels of memory**
  - Ping-pong buffers



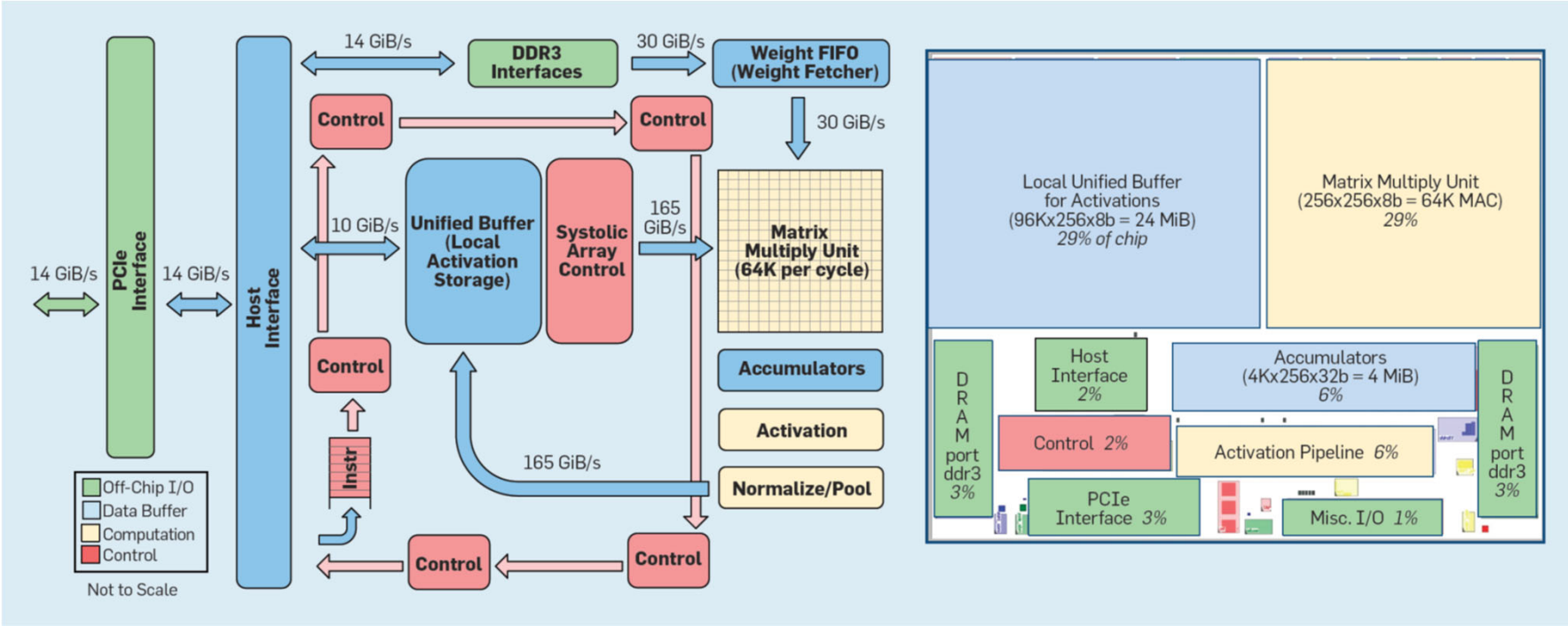
Systolic Array Architecture

- aka 'Matrix of PEs'



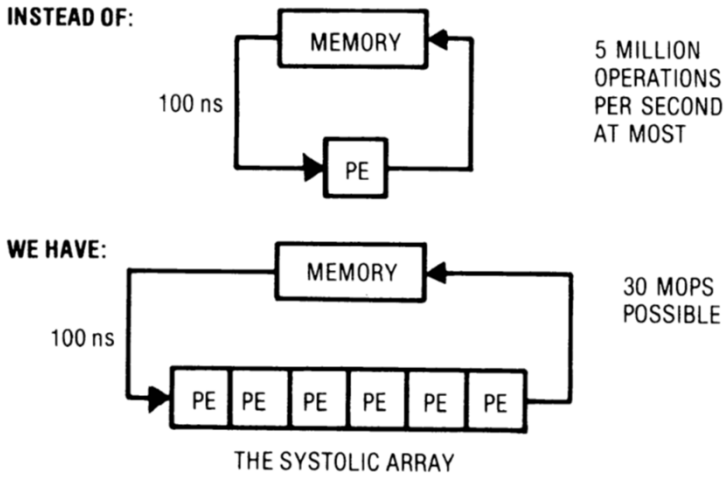
Google TPU

- A systolic array (matrix) is the heart of it



Systolic Arrays

- H.T. Kung, C. Leiserson, 1978

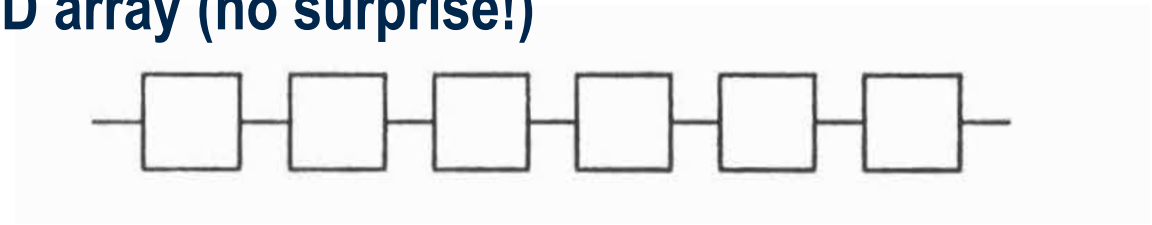


Kung, 82

Systolic Arrays

- 1-D, 2-D, hypercube arrays of processing elements
- Various algorithms can be mapped onto systolic arrays

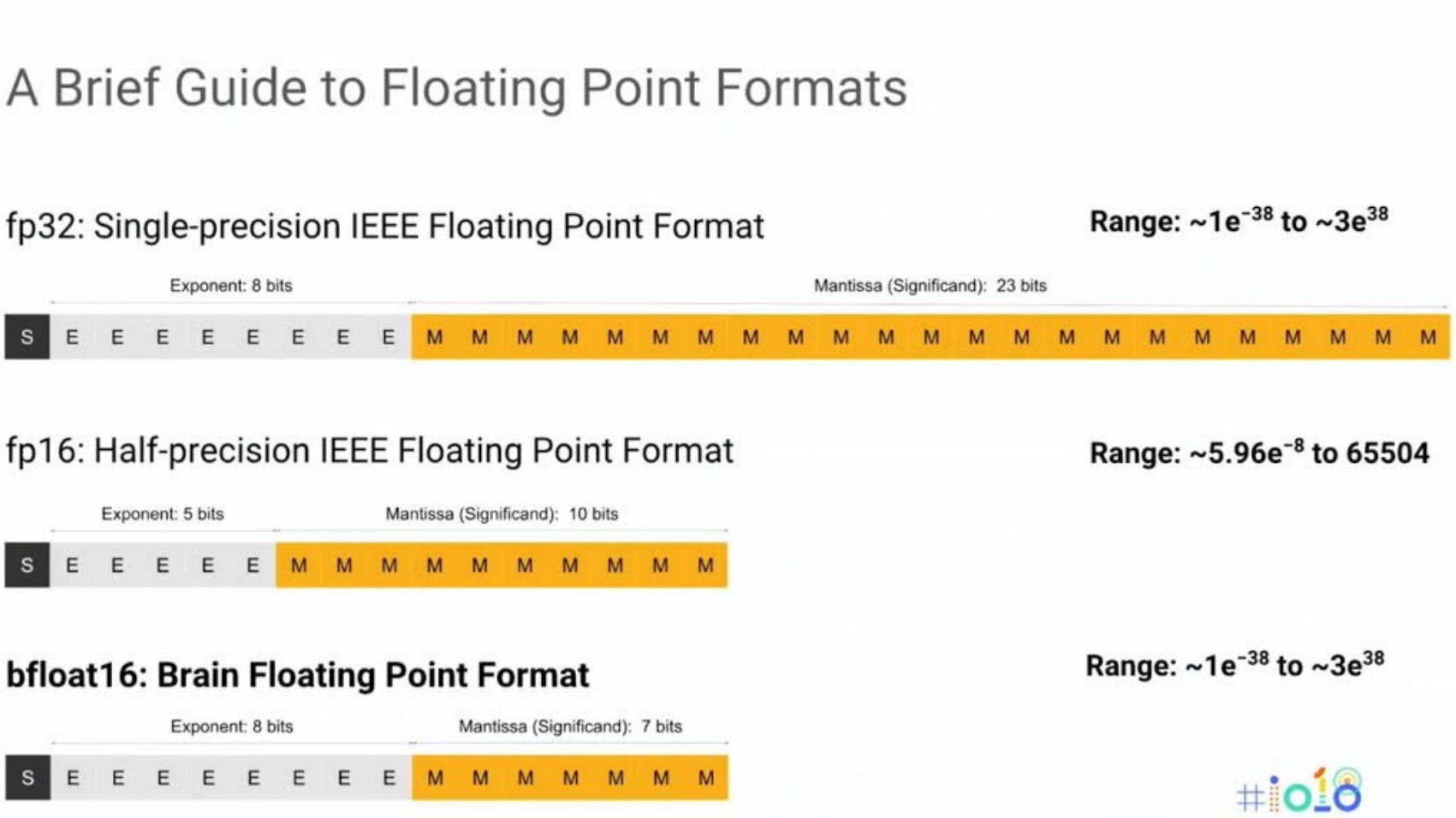
1-D array (no surprise!)



The original idea was to have PEs optimized for algorithms/applications
Current view is to have ALUs/MACs as Pes (integer or FP16 or BFLOAT16)

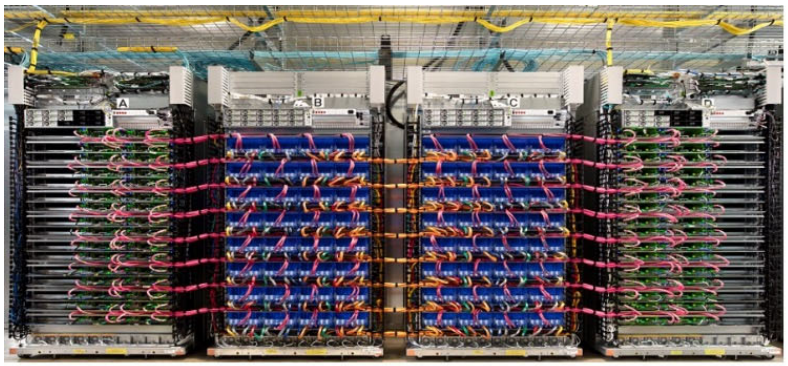
Aside on Floating Point

- TPU 1 used INT8 or INT16; TPU2 use BFLOAT16

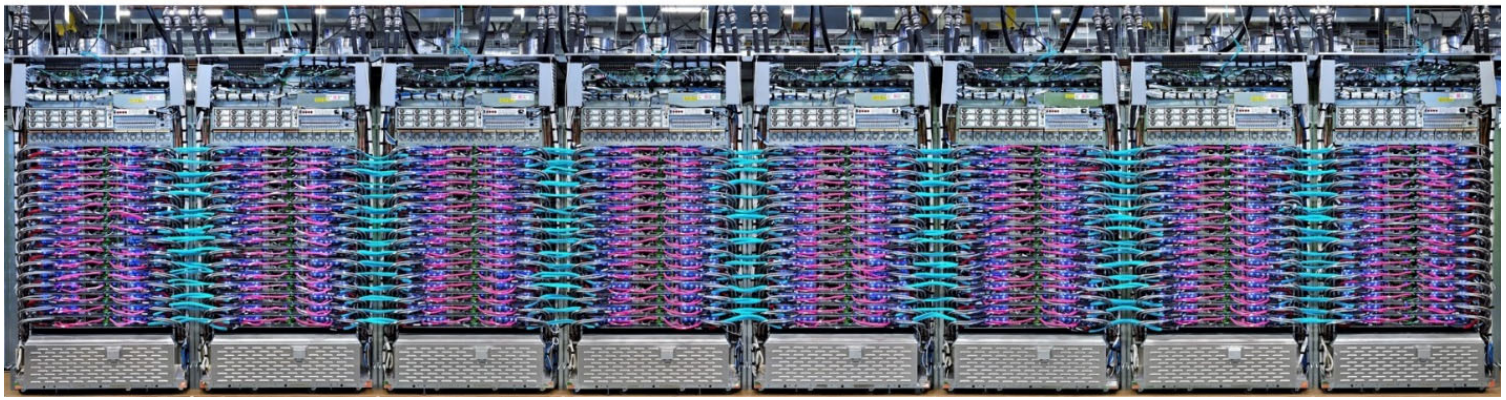


TPUs in the Cloud

- TPU2 pod vs. TPU3 pod



TPU2 pod



TPU3 pod

Systolic Arrays and Convolution

- 1-D array
- Convolution of input x_i with vector w_i

Given the sequence of weights $\{w_1, w_2, \ldots, w_k\}$ and the input sequence $\{x_1, x_2, \ldots, x_n\}$,

compute the result sequence $\{y_1, y_2, \ldots, y_{n+1-k}\}$ defined by

$$y_i = w_1 x_i + w_2 x_{i+1} + \ldots + w_k x_{i+k-1}$$

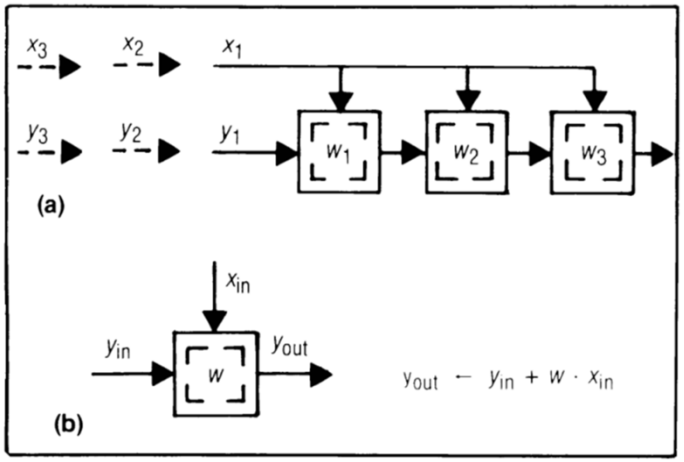


Figure 3. Design B1: systolic convolution array (a) and cell (b) where x_i's are broadcast, w_i's stay, and y_i's move systolically.

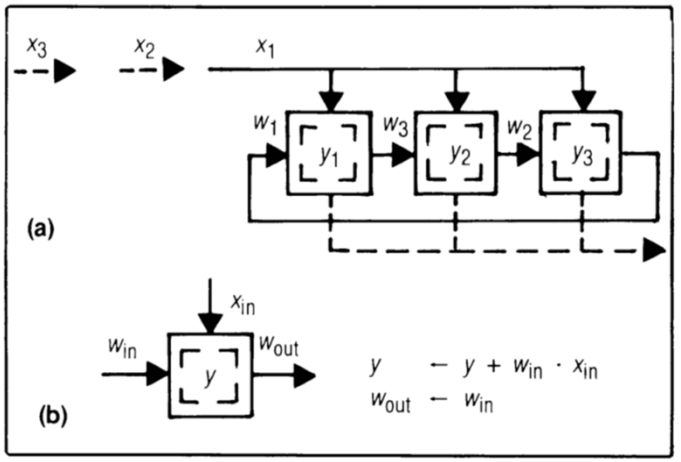


Figure 4. Design B2: systolic convolution array (a) and cell (b) where x_i's are broadcast, y_i's stay, and w_i's move systolically.

PE is a MAC

Systolic Arrays and Convolution

- F: Weights stationary
- R1: Results stationary, inputs and weights move in opposite directions (every two cycles; can be interleaved)

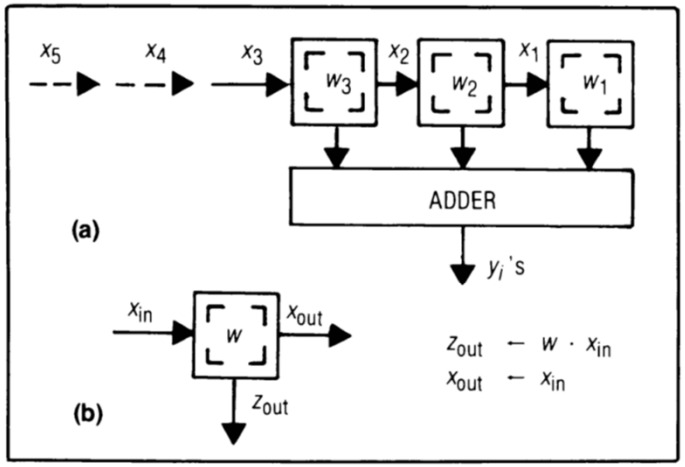


Figure 5. Design F: systolic convolution array (a) and cell (b) where w_i's stay, x_i's move systolically, and y_i's are formed through the fan-in of results from all the cells.

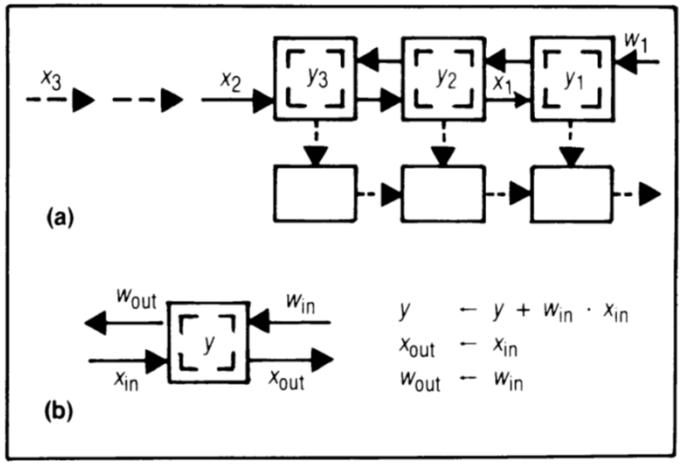


Figure 6. Design R1: systolic convolution array (a) and cell (b) where y_i's stay and x_i's and y_i's move in opposite directions systolically.

Systolic Arrays and Convolution

- R2: x's move 2x as fast as w's (w stays for two cycles)
- W1, W2: Weights stay

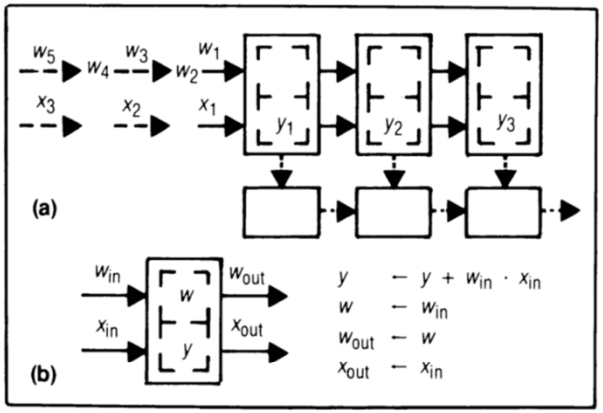


Figure 7. Design R2: systolic convolution array (a) and cell (b) where y_i's stay and x_i's and w_i's both move in the same direction but at different speeds.

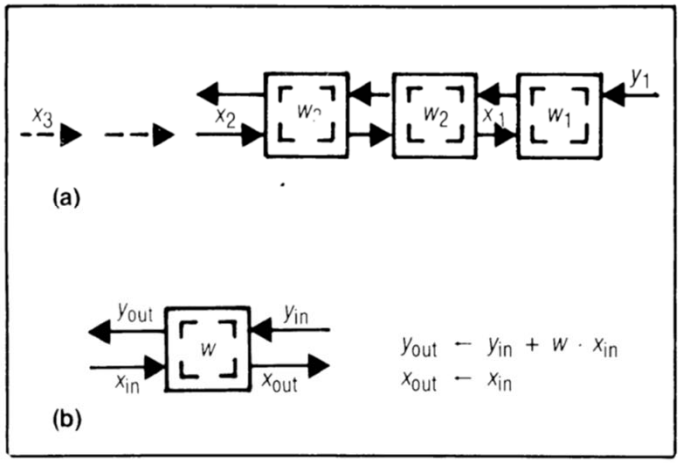


Figure 8. Design W1: systolic convolution array (a) and cell (b) where w_i's stay and x_i's and y_i's move systolically in opposite directions.

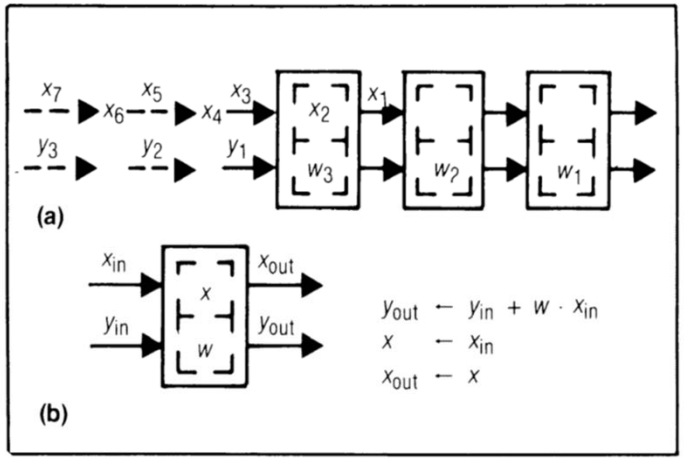
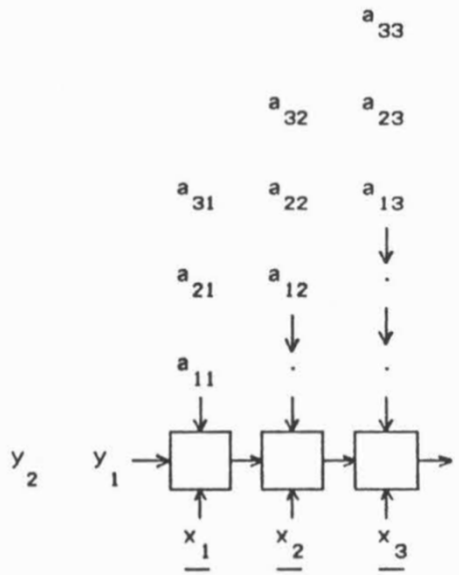


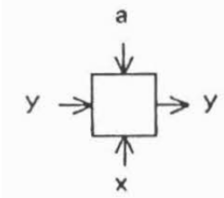
Figure 9. Design W2: systolic convolution array (a) and cell (b) where w_i's stay and x_i's and y_i's both move systolically in the same direction but at different speeds.

Systolic Arrays

- Matrix-vector multiplication on 1-D array

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \\ a_{51} & a_{52} & a_{53} \\ . & . & . \\ . & . & . \\ . & . & . \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ . \\ . \\ . \end{bmatrix}$$





$y \leftarrow y + ax$

Systolic Arrays

- **Band matrix-vector multiplication**

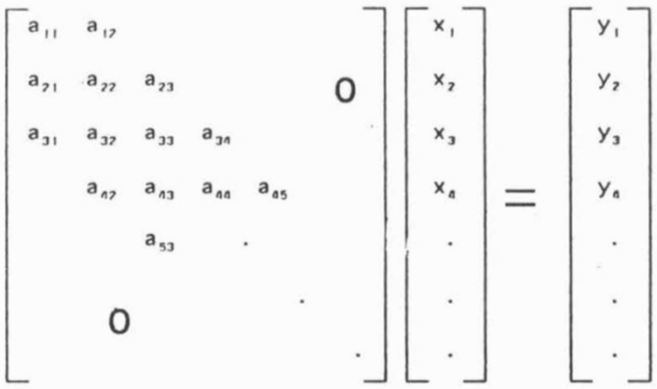
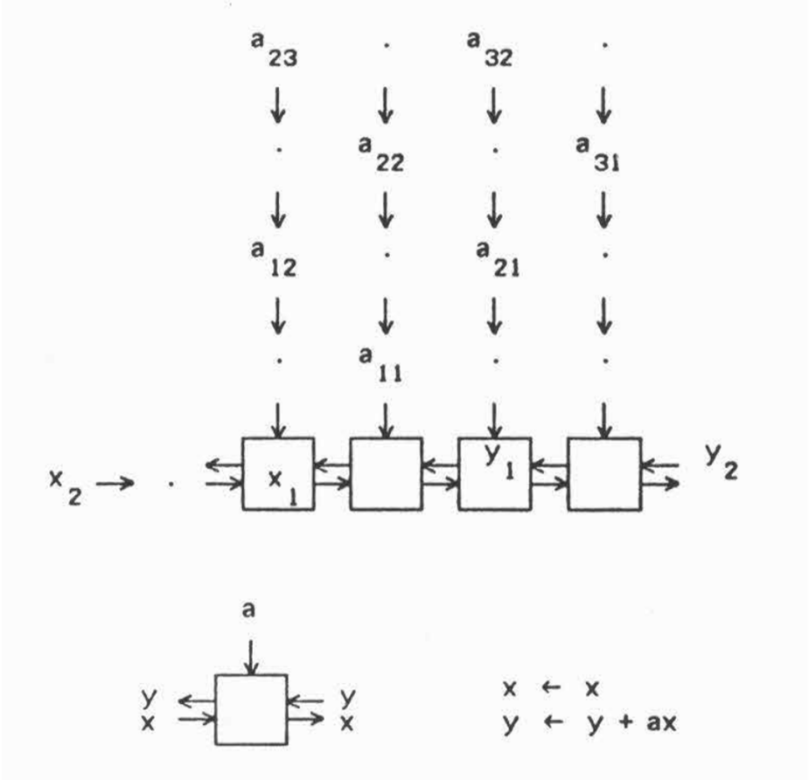
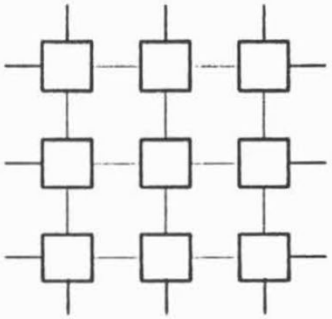


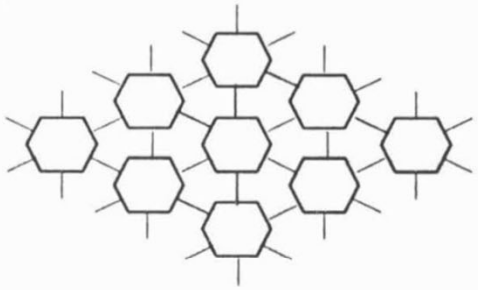
Figure 7: Band matrix-vector multiplication.

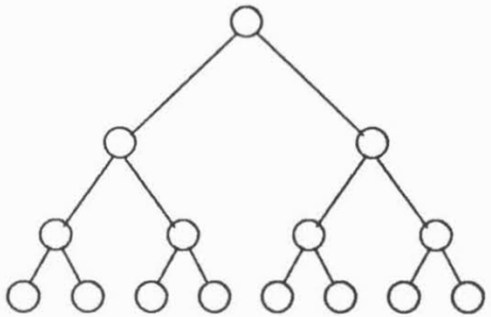


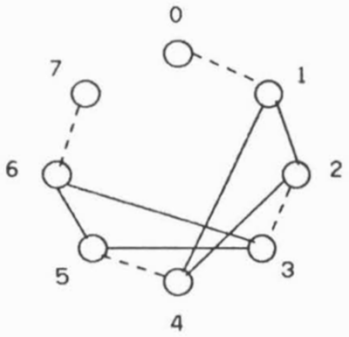
Systolic Arrays

- 2-D arrays









shuffle ——

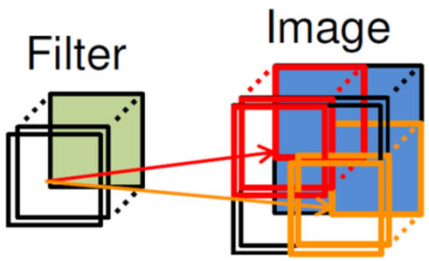
exchange - - - -

Communication Geometry	Examples
1-dim linear arrays	Matrix-vector multiplication FIR filter Convolution DFT Carry pipelining Pipeline arithmetic units Real-time recurrence evaluation Solution of triangular linear systems Constant-time priority queue, on-line sort Cartesian product Odd-even transposition sort
2-dim square arrays	Dynamic programming for optimal parenthesization Numerical relaxation for PDE Merge sort FFT Graph algorithms using adjacency matrices
2-dim hexagonal arrays	Matrix multiplication Transitive closure LU-decomposition by Gaussian elimination without pivoting
Trees	Searching algorithms Queries on nearest neighbor, rank, etc. NP-complete problems systolic search tree Parallel function evaluation Recurrence evaluation
Shuffle-exchange networks	FFT Bitonic sort

Data Reuse in CNNs

Convolutional Reuse

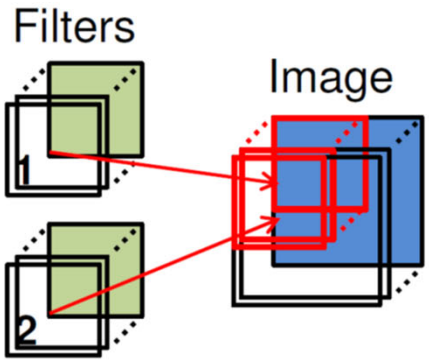
CONV layers only
(sliding window)



Reuse: Image pixels
Filter weights

Image Reuse

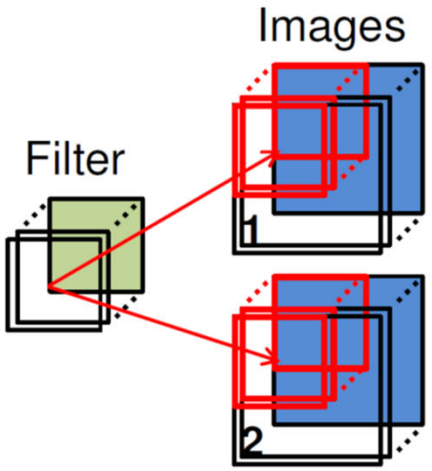
CONV and FC layers



Reuse: Image pixels

Filter Reuse

CONV and FC layers
(batch size > 1)



Reuse: Filter weights

Next Lecture

- **Tape-in presentations**
- **Systolic arrays for ML**
  - And other accelerators