

# Lecture17 Domain-Specific Architectures

Krste Asanovic

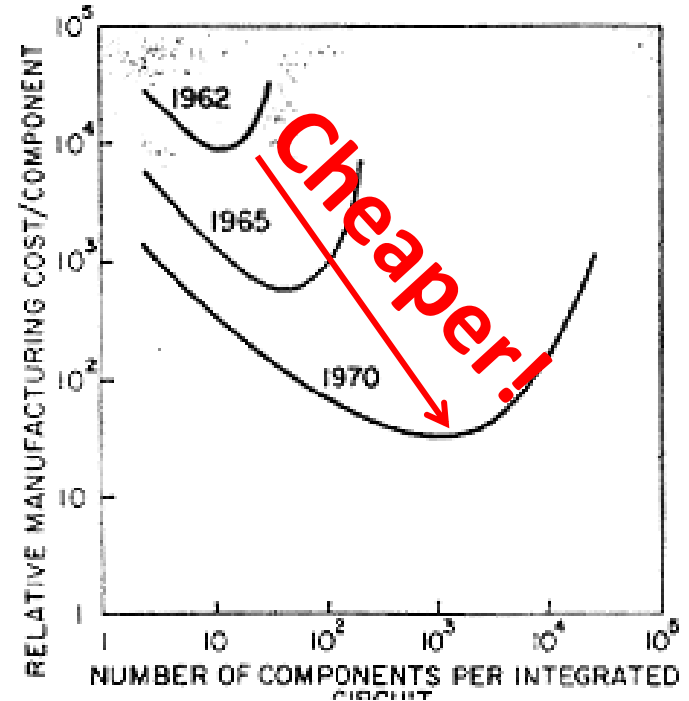
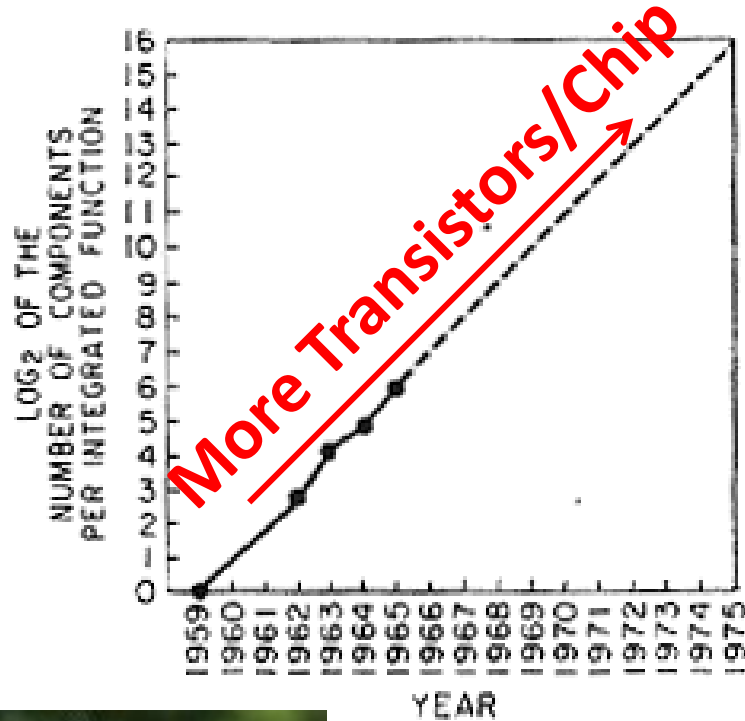
Electrical Engineering and Computer Sciences

University of California at Berkeley

`http://www.eecs.berkeley.edu/~krste`

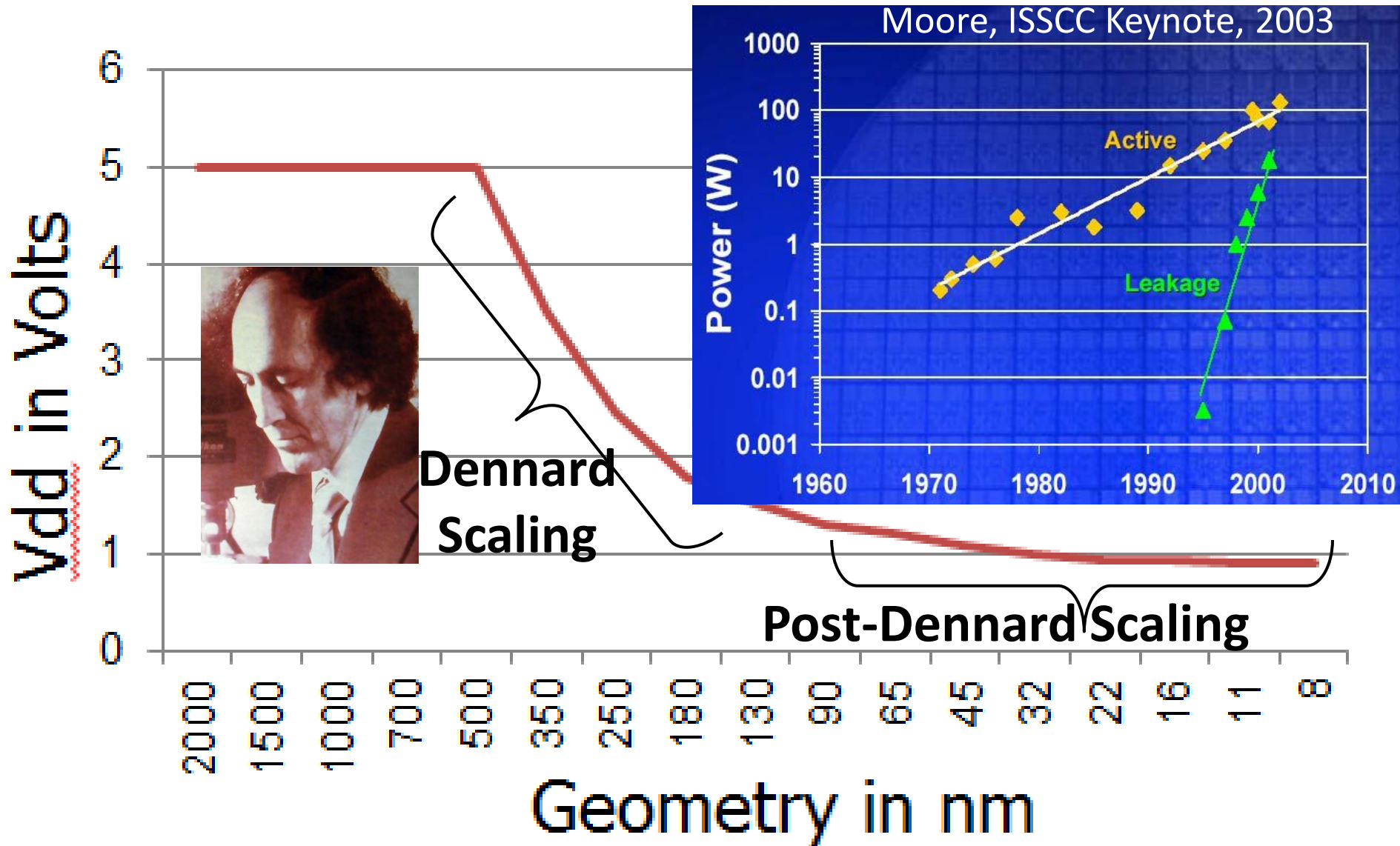
`http://inst.eecs.berkeley.edu/~cs152`

# Moore's Law



“Cramming more components onto integrated circuits”, Gordon E. Moore, Electronics, 1965

# End of Dennard (Voltage) Scaling

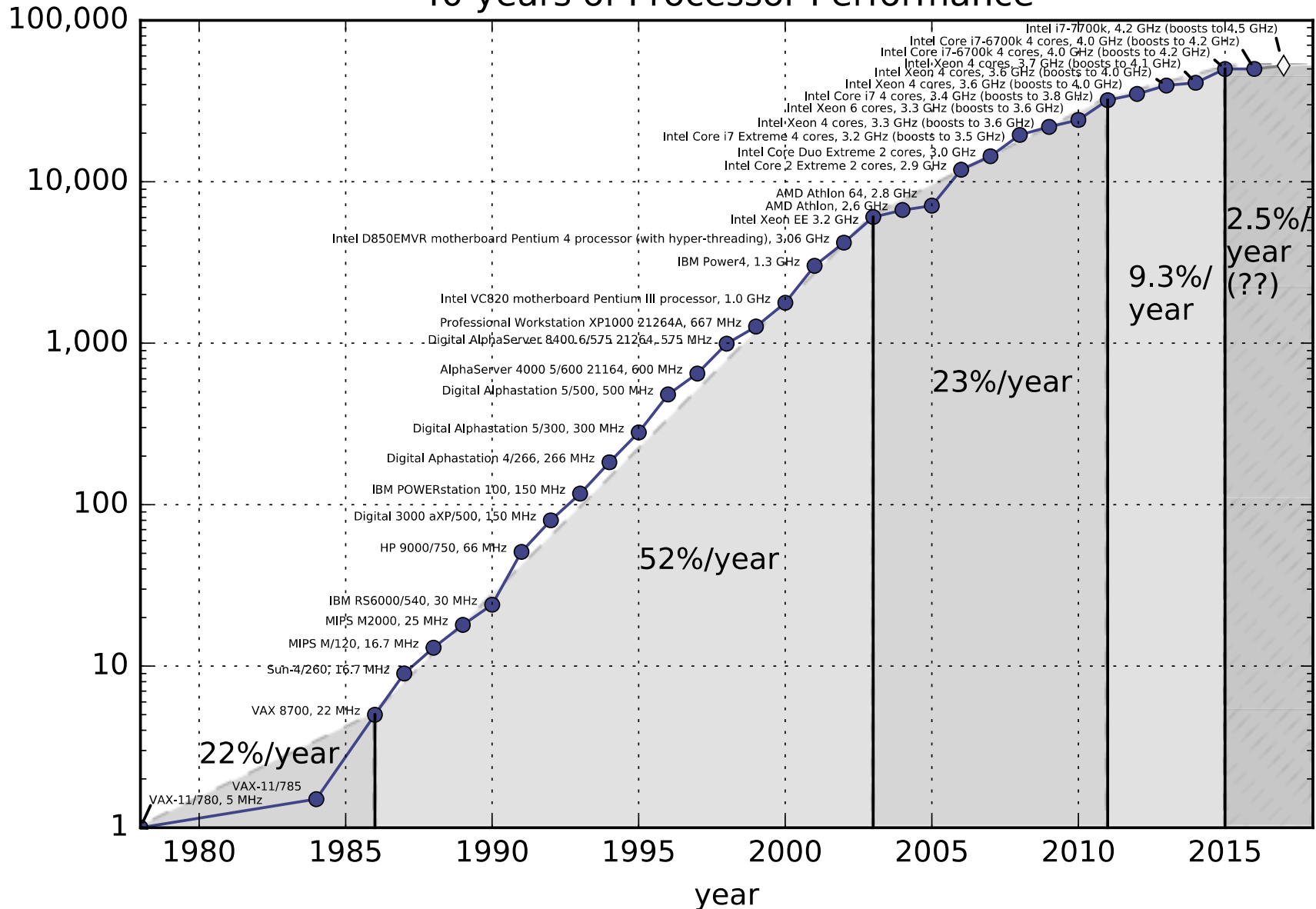


Data courtesy S. Borkar/Intel 2011

# Single-Thread Processor Performance

## 40 years of Processor Performance

Performance vs. VAX-11/780



[ Hennessy & Patterson, 2017 ]

# Domain-Specific Architectures

In early 2000s, microprocessors moved to manycore architecture (multiple general-purpose cores per die) to improve energy efficiency for parallel workloads.

Now, great interest in more specialized architectures to further improve energy efficiency on certain workloads

Four examples:

- Google TPU

- Microsoft Catapult

- Intel Crest (now Nervana)

- Google Pixel Visual Core

# Guidelines for DSAs

- Use dedicated memories to minimize data movement
- Invest resources into more arithmetic units or bigger memories
- Use the easiest form of parallelism that matches the domain
- Reduce data size and type to the simplest needed for the domain
- Use a domain-specific programming language

# Guidelines for DSAs

Guideline	TPU	Catapult	Crest	Pixel Visual Core
Design target	Data center ASIC	Data center FPGA	Data center ASIC	PMD ASIC/SOC IP
1. Dedicated memories	24 MiB Unified Buffer, 4 MiB Accumulators	Varies	N.A.	Per core: 128 KiB line buffer, 64 KiB P.E. memory
2. Larger arithmetic unit	65,536 Multiply-accumulators	Varies	N.A.	Per core: 256 Multiply-accumulators (512 ALUs)
3. Easy parallelism	Single-threaded, SIMD, in-order	SIMD, MISD	N.A.	MPMD, SIMD, VLIW
4. Smaller data size	8-Bit, 16-bit integer	8-Bit, 16-bit integer 32-bit Fl. Pt.	21-bit Fl. Pt.	8-bit, 16-bit, 32-bit integer
5. Domain-specific lang.	TensorFlow	Verilog	TensorFlow	Halide/TensorFlow

# Example: Deep Neural Networks

- Inspired by neuron of the brain
- Computes non-linear “activation” function of the weighted sum of input values
- Neurons arranged in layers

Name	DNN layers	Weights	Operations/Weight
MLP0	5	20M	200
MLP1	4	5M	168
LSTM0	58	52M	64
LSTM1	56	34M	96
CNN0	16	8M	2888
CNN1	89	100M	1750



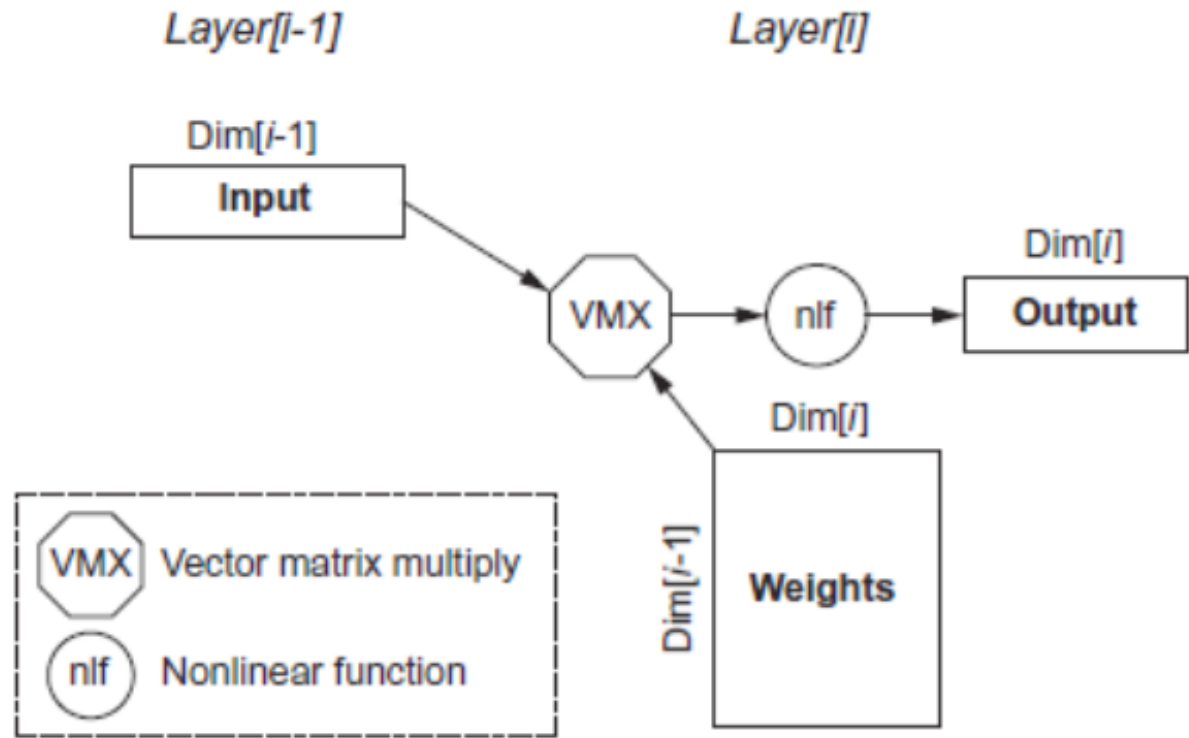
# Example: Deep Neural Networks

- Most practitioners will use existing design
  - Topology
  - Data type
- Training (learning):
  - Calculate weights using backpropagation algorithm
  - Supervised learning: stochastic gradient descent 随机梯度下降

Type of data	Problem area	Size of benchmark's training set	DNN architecture	Hardware	Training time
text [1]	Word prediction (word2vec)	100 billion words (Wikipedia)	2-layer skip gram	1 NVIDIA Titan X GPU	6.2 hours
audio [2]	Speech recognition	2000 hours (Fisher Corpus)	11-layer RNN	1 NVIDIA K1200 GPU	3.5 days
images [3]	Image classification	1 million images (ImageNet)	22-layer CNN	1 NVIDIA K20 GPU	3 weeks
video [4]	activity recognition	1 million videos (Sports-1M)	8-layer CNN	10 NVIDIA GPUs	1 month

- Inference: use neural network for classification

# Multi-Layer Perceptrons

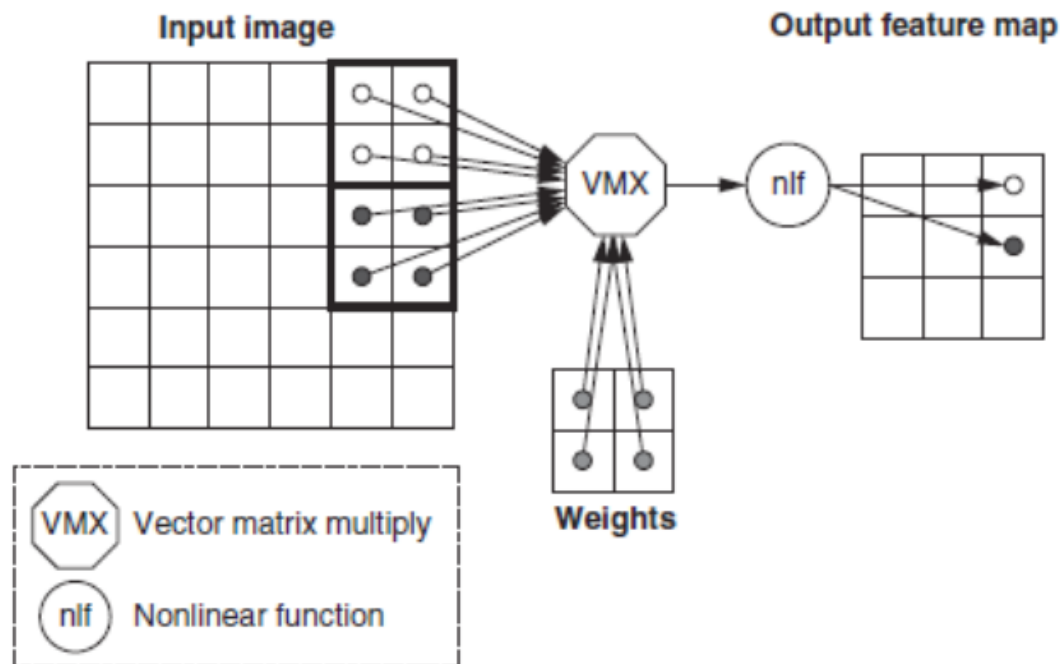


## Parameters:

- $\text{Dim}[i]$ : number of neurons
- $\text{Dim}[i-1]$ : dimension of input vector
- Number of weights:  $\text{Dim}[i-1] \times \text{Dim}[i]$
- Operations:  $2 \times \text{Dim}[i-1] \times \text{Dim}[i]$
- Operations/weight: 2

# Convolutional Neural Network

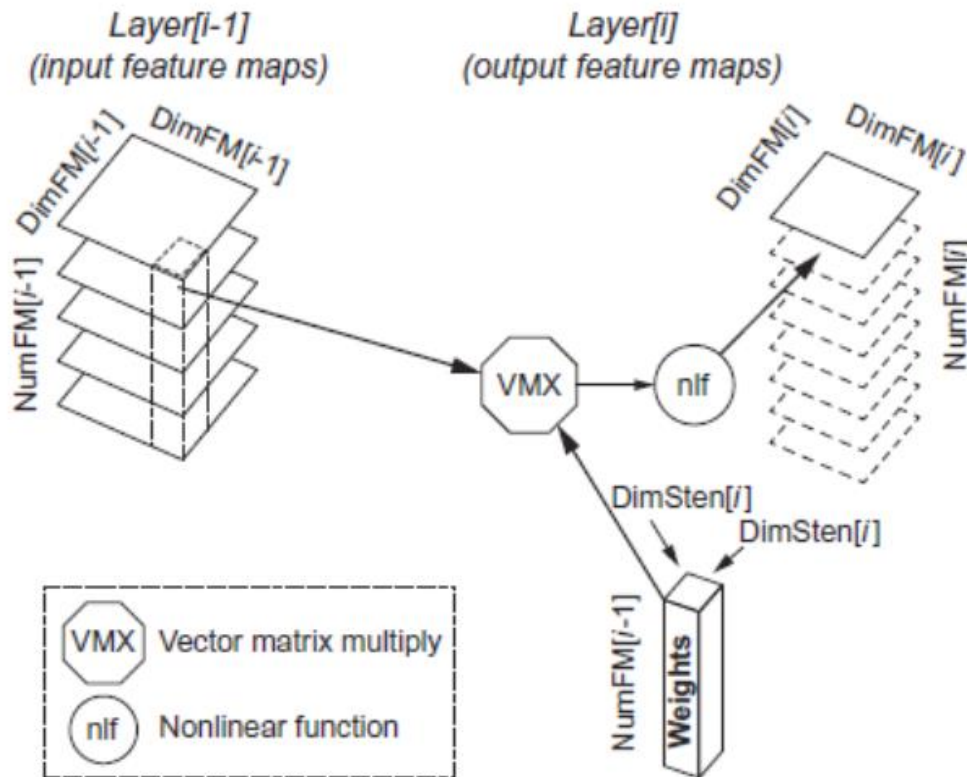
- Computer vision
- Each layer raises the level of abstraction
  - First layer recognizes horizontal and vertical lines
  - Second layer recognizes corners
  - Third layer recognizes shapes
  - Fourth layer recognizes features, such as ears of a dog
  - Higher layers recognizes different breeds of dogs



# Convolutional Neural Network

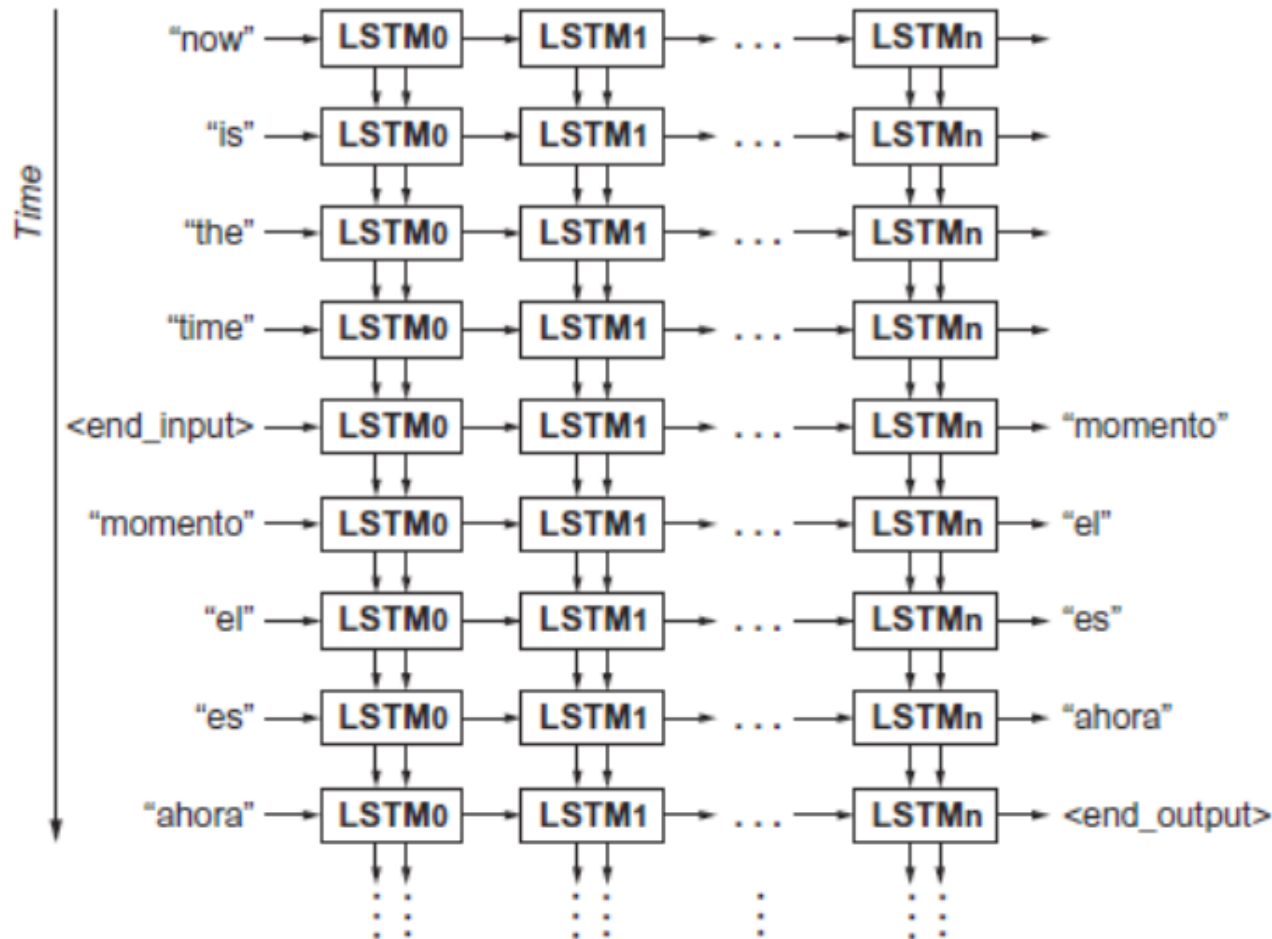
## Parameters:

- $\text{DimFM}[i-1]$ : Dimension of the (square) input Feature Map
- $\text{DimFM}[i]$ : Dimension of the (square) output Feature Map
- $\text{DimSten}[i]$ : Dimension of the (square) stencil
- $\text{NumFM}[i-1]$ : Number of input Feature Maps
- $\text{NumFM}[i]$ : Number of output Feature Maps
- Number of neurons:  $\text{NumFM}[i] \times \text{DimFM}[i]^2$
- Number of weights per output Feature Map:  $\text{NumFM}[i-1] \times \text{DimSten}[i]^2$
- Total number of weights per layer:  $\text{NumFM}[i] \times \text{Number of weights per output Feature Map}$
- Number of operations per output Feature Map:  $2 \times \text{DimFM}[i]^2 \times \text{Number of weights per output Feature Map}$
- Total number of operations per layer:  $\text{NumFM}[i] \times \text{Number of operations per output Feature Map} = 2 \times \text{DimFM}[i]^2 \times \text{NumFM}[i] \times \text{Number of weights per output Feature Map} = 2 \times \text{DimFM}[i]^2 \times \text{Total number of weights per layer}$
- Operations/Weight:  $2 \times \text{DimFM}[i]^2$

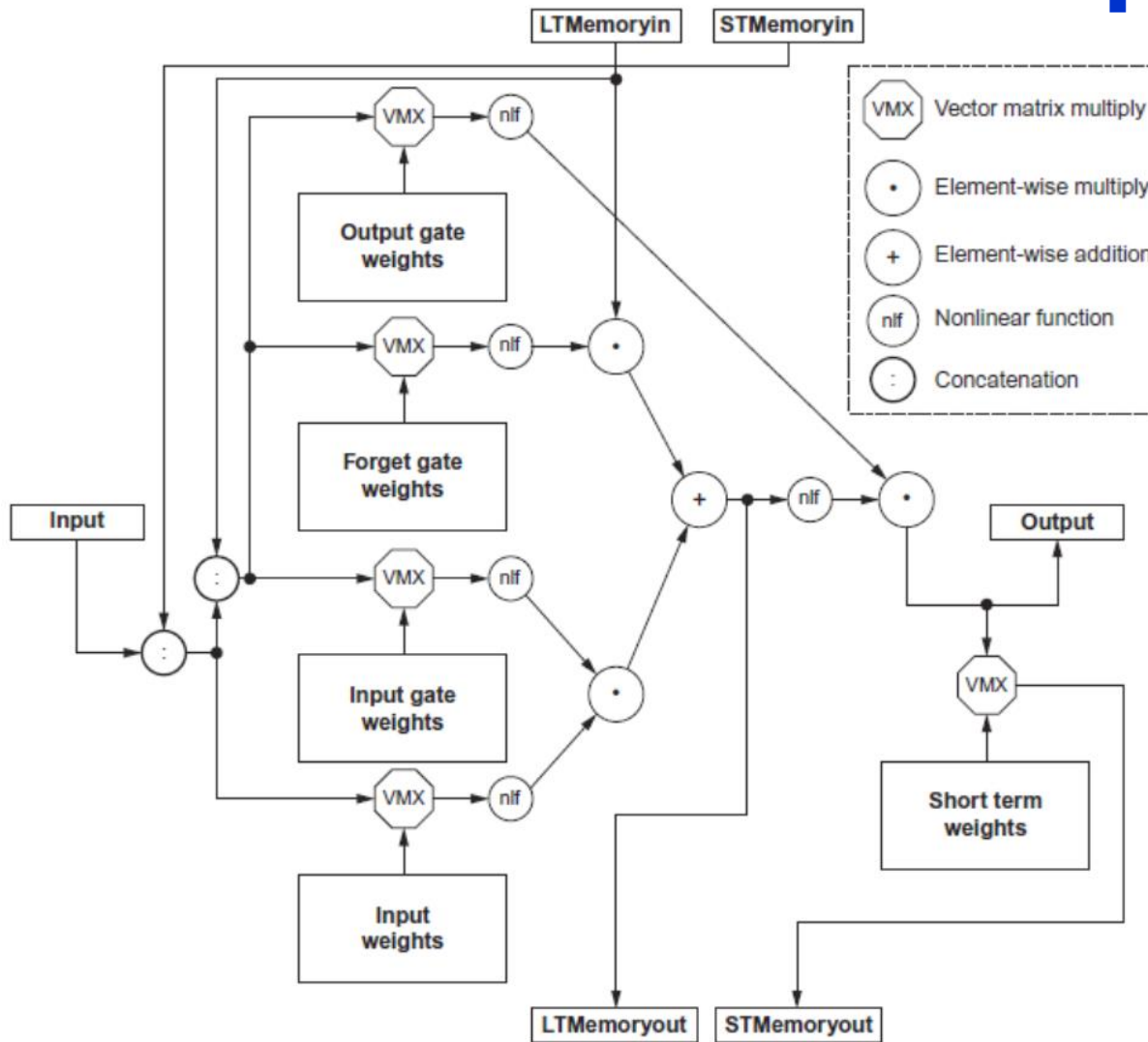


# Recurrent Neural Network

- Speech recognition and language translation
- Long short-term memory (LSTM) network



# Recurrent Neural Network



## Parameters:

- Number of weights per cell:  
 $3 \times (3 \times \text{Dim} \times \text{Dim}) + (2 \times \text{Dim} \times \text{Dim}) + (1 \times \text{Dim} \times \text{Dim}) = 12 \times \text{Dim}^2$
- Number of operations for the 5 vector-matrix multiplies per cell:  $2 \times \text{Number of weights per cell} = 24 \times \text{Dim}^2$
- Number of operations for the 3 element-wise multiplies and 1 addition (vectors are all the size of the output):  $4 \times \text{Dim}$
- Total number of operations per cell (5 vector-matrix multiplies and the 4 element-wise operations):  $24 \times \text{Dim}^2 + 4 \times \text{Dim}$
- Operations/Weight:  $\sim 2$

# Convolutional Neural Network

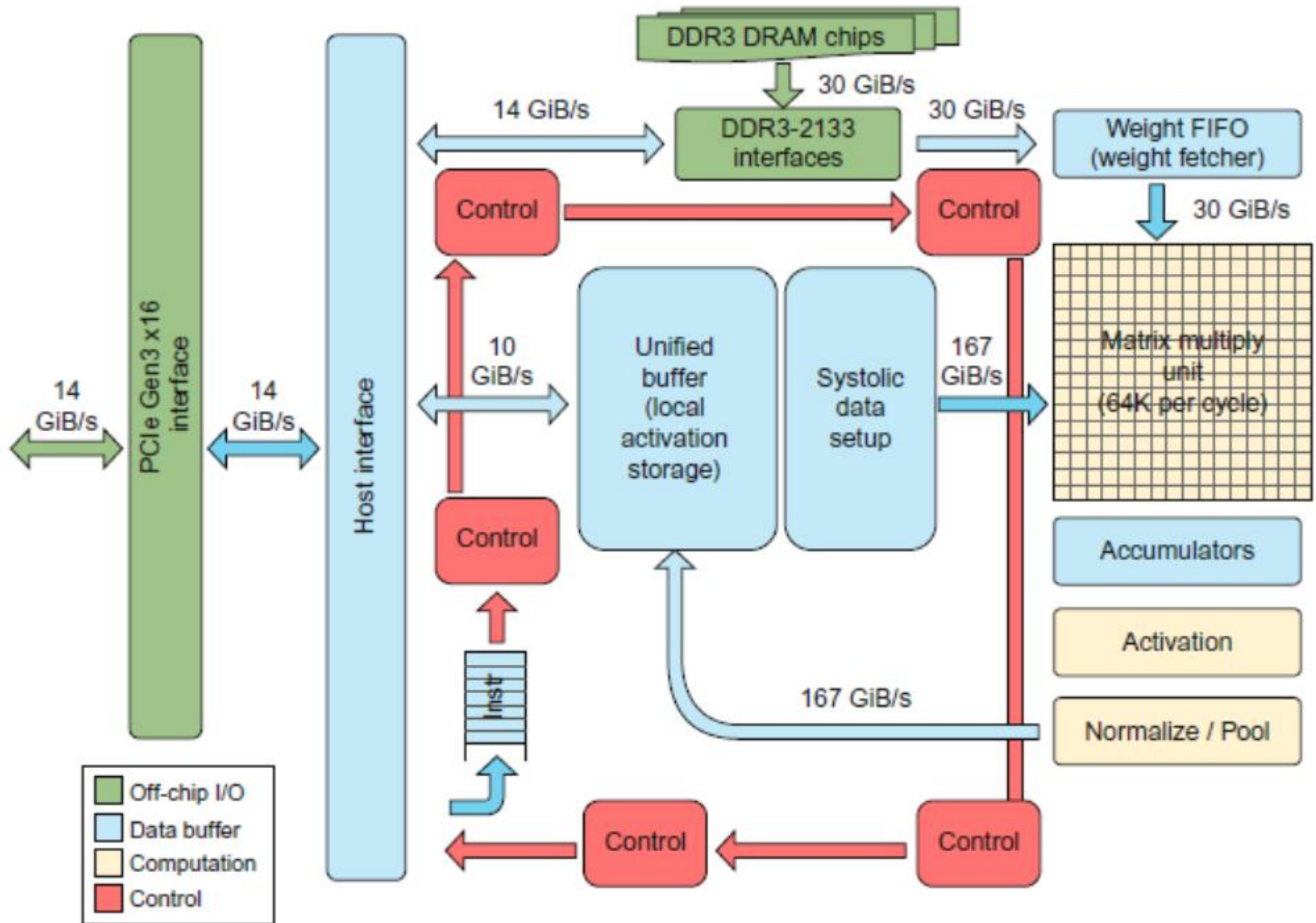
- Batches:
  - Reuse weights once fetched from memory across multiple inputs
  - Increases operational intensity
- Quantization
  - Use 8- or 16-bit fixed point
- Summary:
  - Need the following kernels:
    - Matrix-vector multiply
    - Matrix-matrix multiply
    - Stencil
    - ReLU (Rectified Linear Unit =  $\max(0, x)$ )
    - Sigmoid
    - Hyperbolic tangent

# Tensor Processing Unit

- Google's DNN ASIC
- 256 x 256 8-bit matrix-multiply unit
- Large software-managed scratchpad
- Coprocessor on the PCIe bus



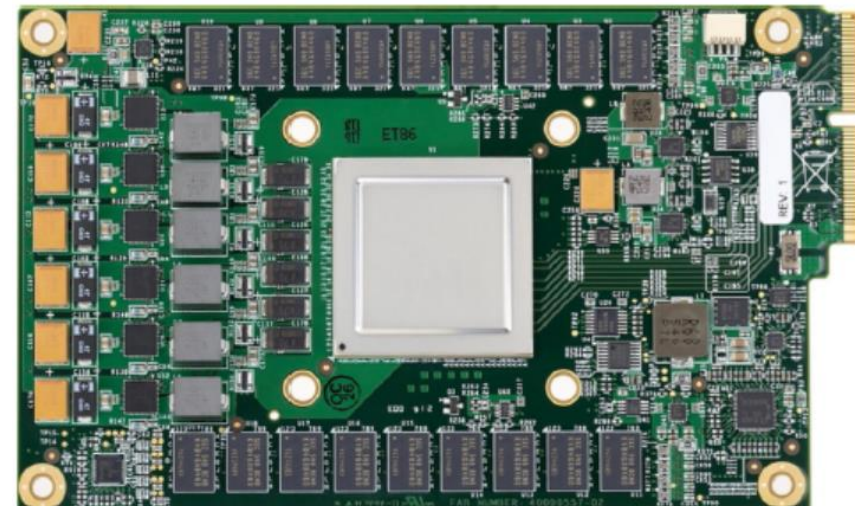
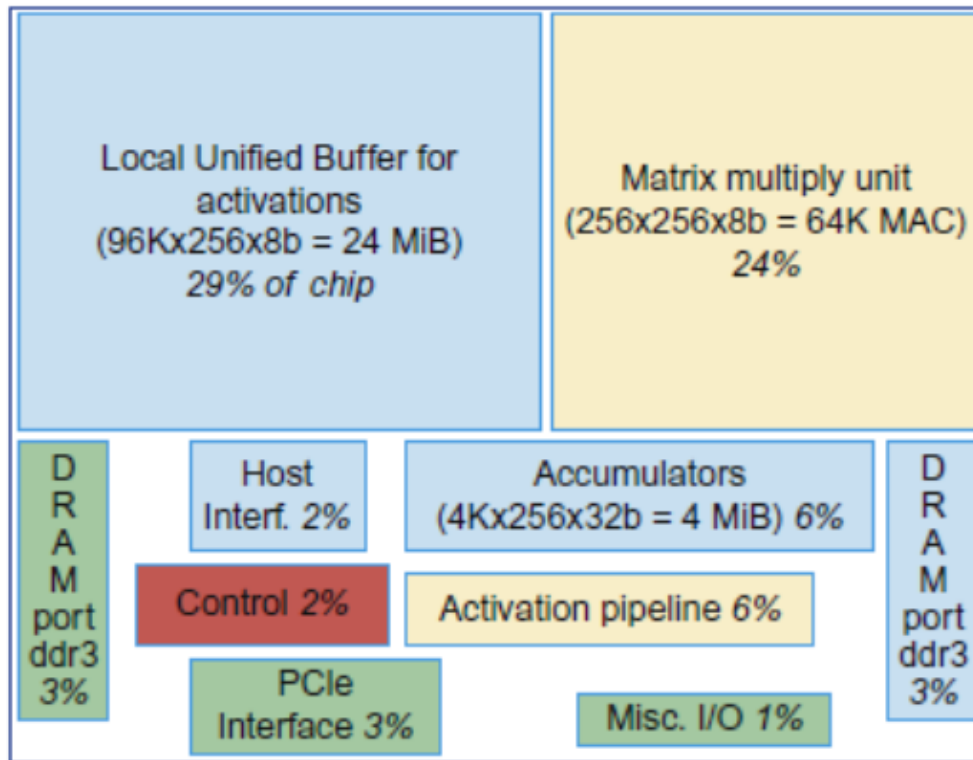
# Tensor Processing Unit



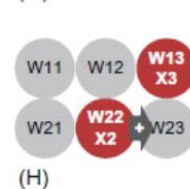
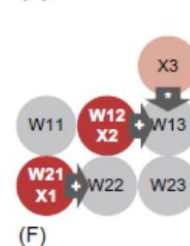
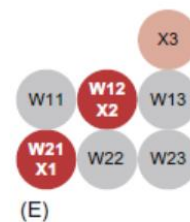
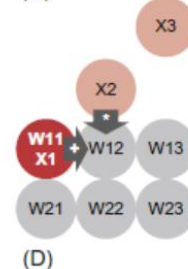
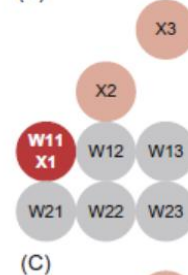
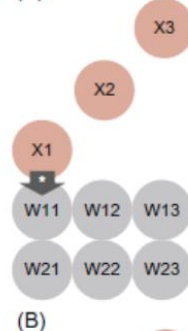
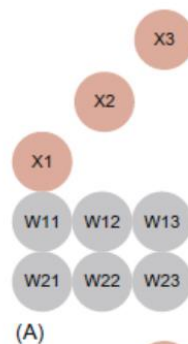
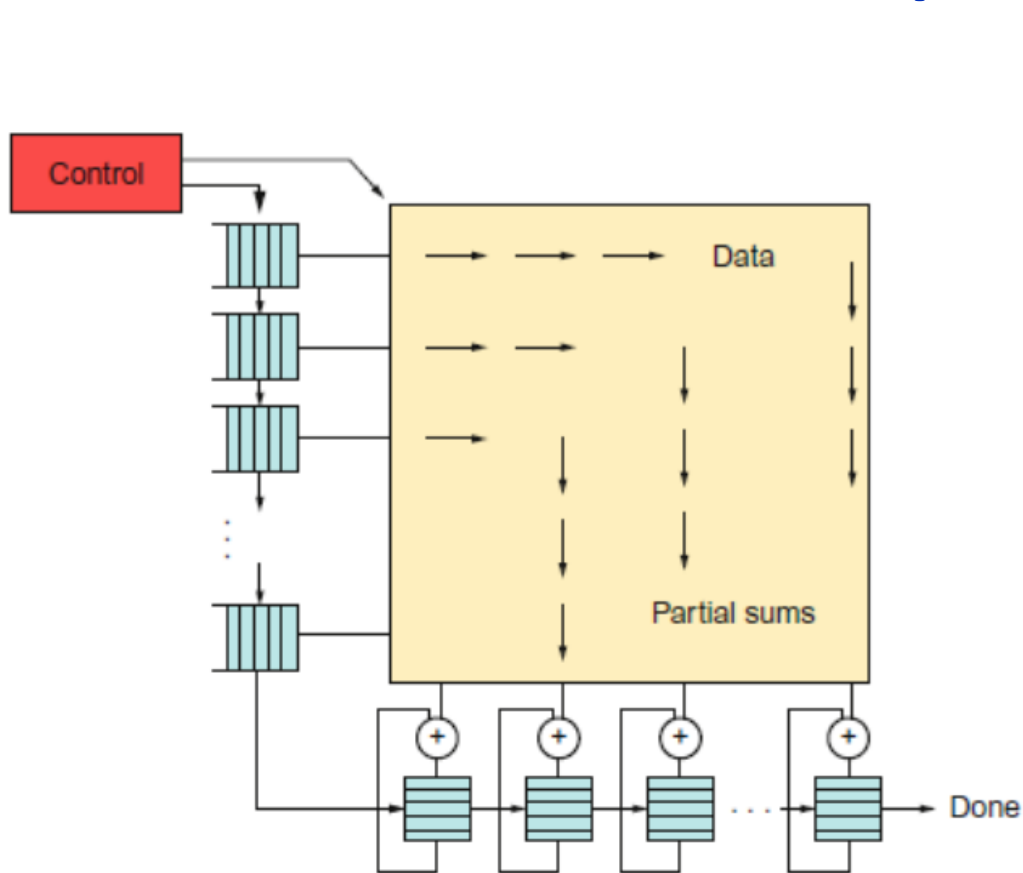
# TPU ISA

- Read\_Host\_Memory
  - Reads memory from the CPU memory into the unified buffer
- Read\_Weights
  - Reads weights from the Weight Memory into the Weight FIFO as input to the Matrix Unit
- MatrixMatrixMultiply/Convolve
  - Perform a matrix-matrix multiply, a vector-matrix multiply, an element-wise matrix multiply, an element-wise vector multiply, or a convolution from the Unified Buffer into the accumulators
  - takes a variable-sized  $B \times 256$  input, multiplies it by a  $256 \times 256$  constant input, and produces a  $B \times 256$  output, taking  $B$  pipelined cycles to complete
- Activate
  - Computes activation function
- Write\_Host\_Memory
  - Writes data from unified buffer into host memory

# TPU Implementation



# TPU Operation

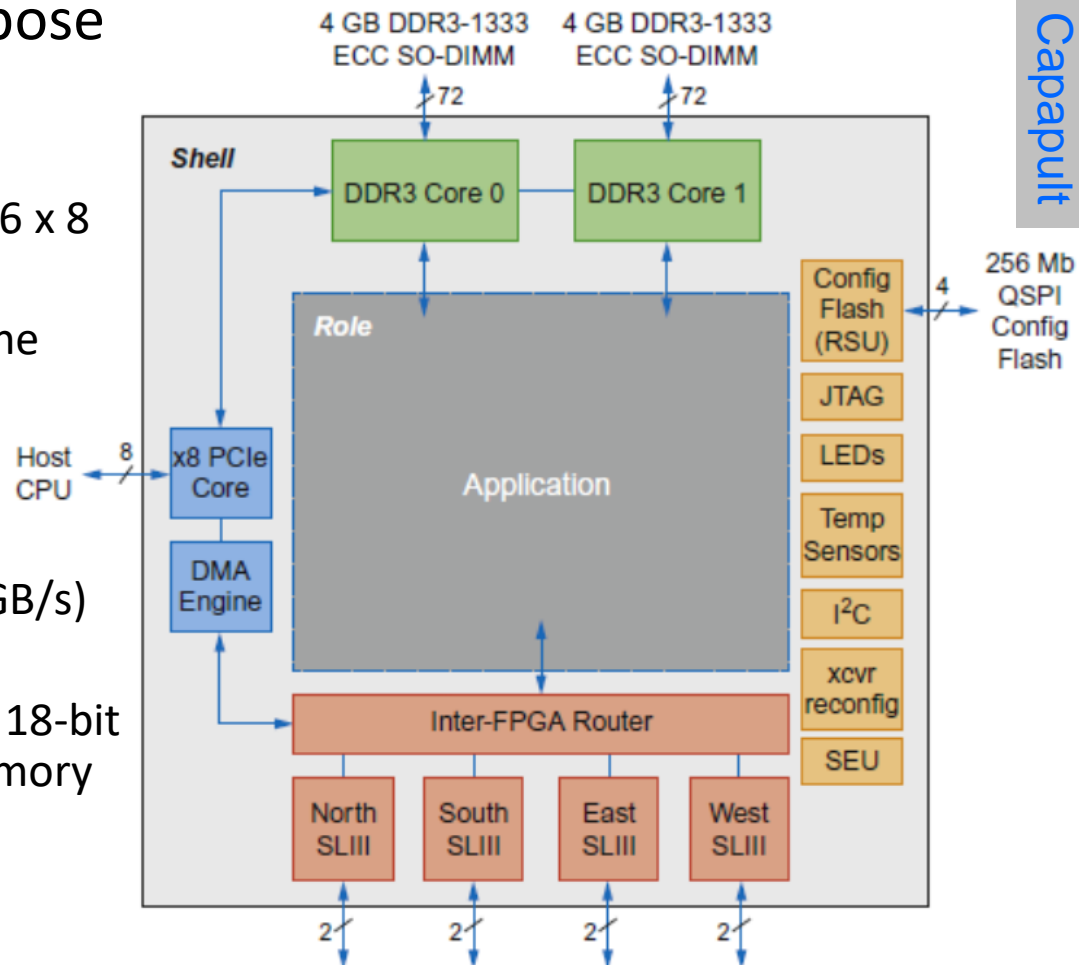


# The TPU and the Guidelines

- Use dedicated memories
  - 24 MiB dedicated buffer, 4 MiB accumulator buffers
- Invest resources in arithmetic units and dedicated memories
  - 60% of the memory and 250X the arithmetic units of a server-class CPU
- Use the easiest form of parallelism that matches the domain
  - Exploits 2D SIMD parallelism
- Reduce the data size and type needed for the domain
  - Primarily uses 8-bit integers
- Use a domain-specific programming language
  - Uses TensorFlow

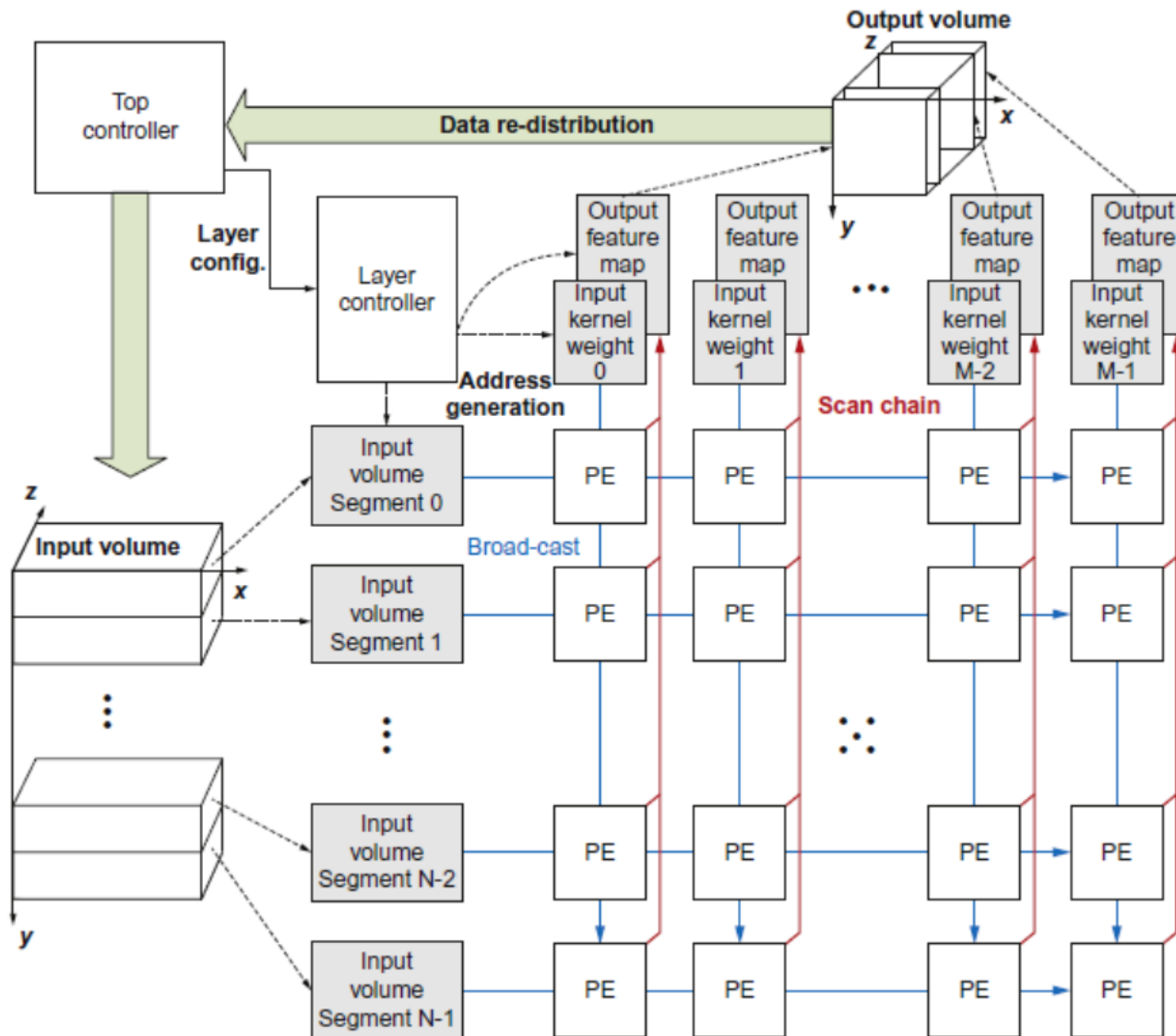
# Microsoft Catapult

- Needed to be general-purpose and power-efficient
  - Uses FPGA PCIe board with dedicated 20 Gbps network in 6 x 8 torus
  - Each of the 48 servers in half the rack has a Catapult board
  - Limited to 25 watts
  - 32 MiB Flash memory
  - Two banks of DDR3-1600 (11 GB/s) and 8 GiB DRAM
  - FPGA (unconfigured) has 3962 18-bit ALUs and 5 MiB of on-chip memory
  - Programmed in Verilog RTL
  - Shell is 23% of the FPGA



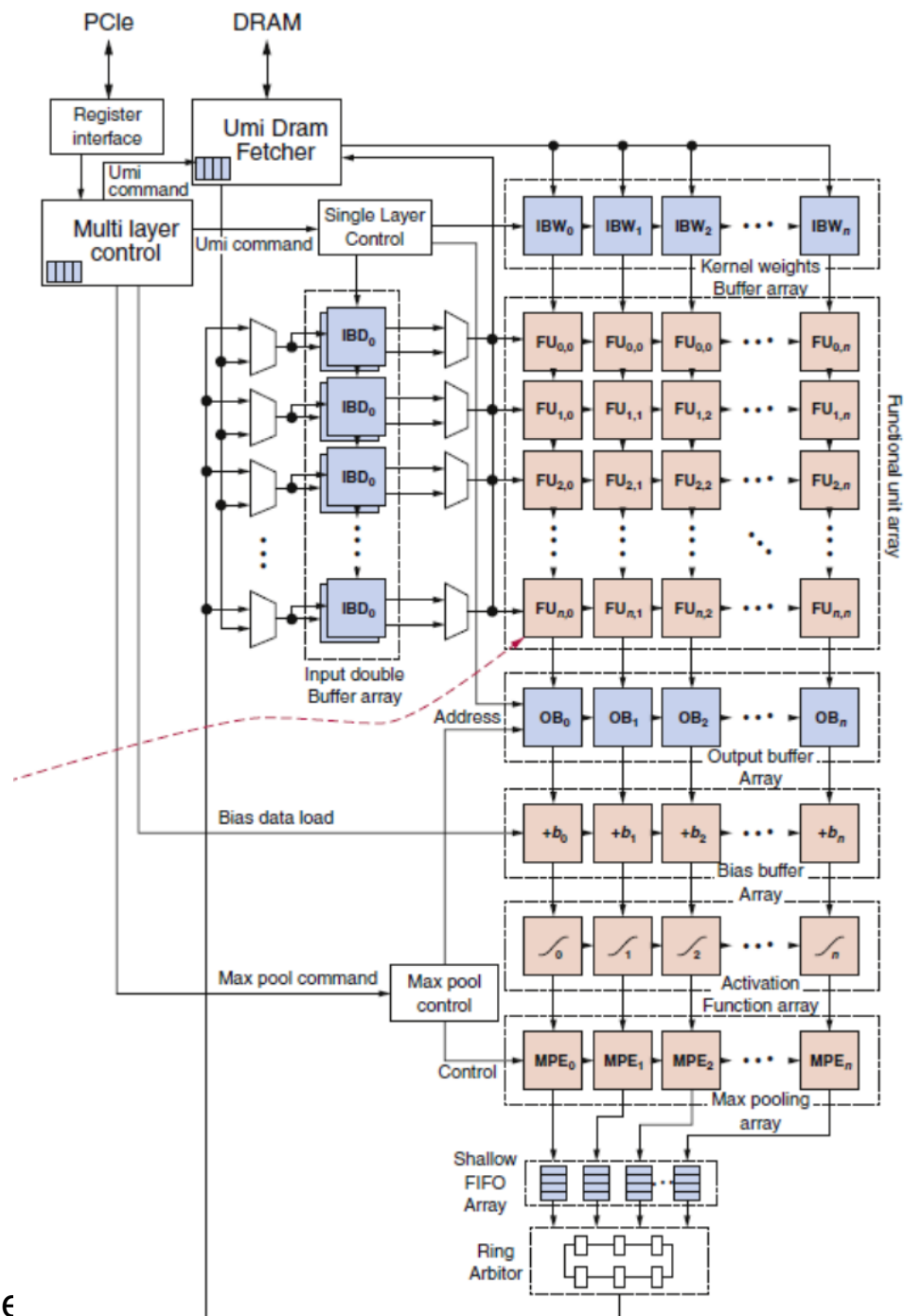
# Microsoft Catapult: CNN

- CNN accelerator, mapped across multiple FPGAs





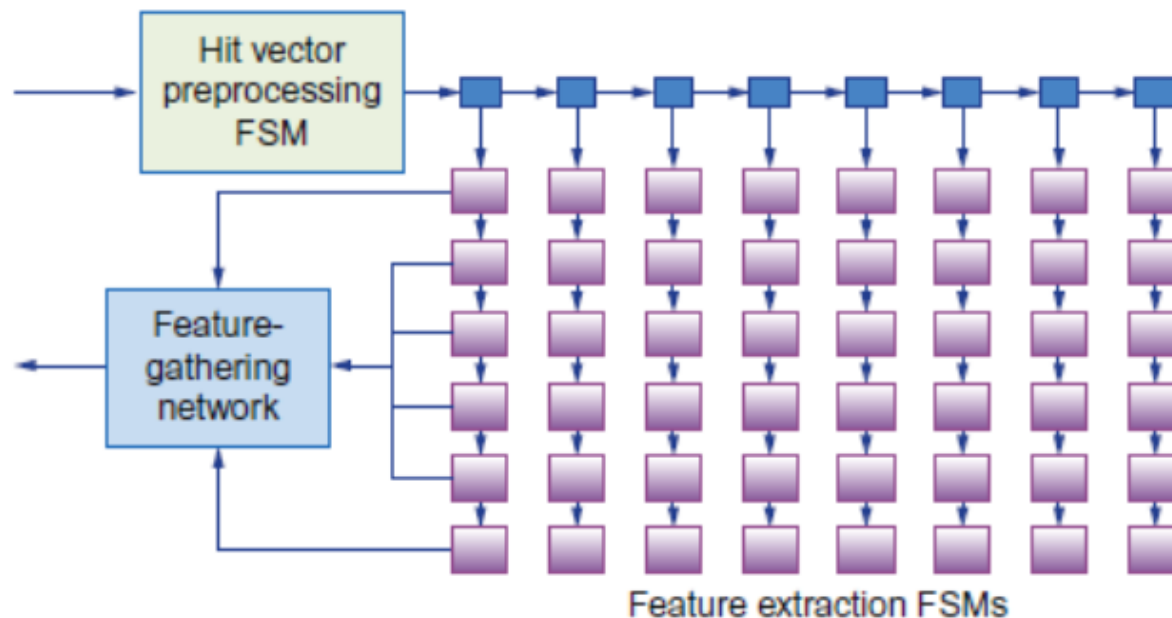
# Microsoft Catapult: CNN





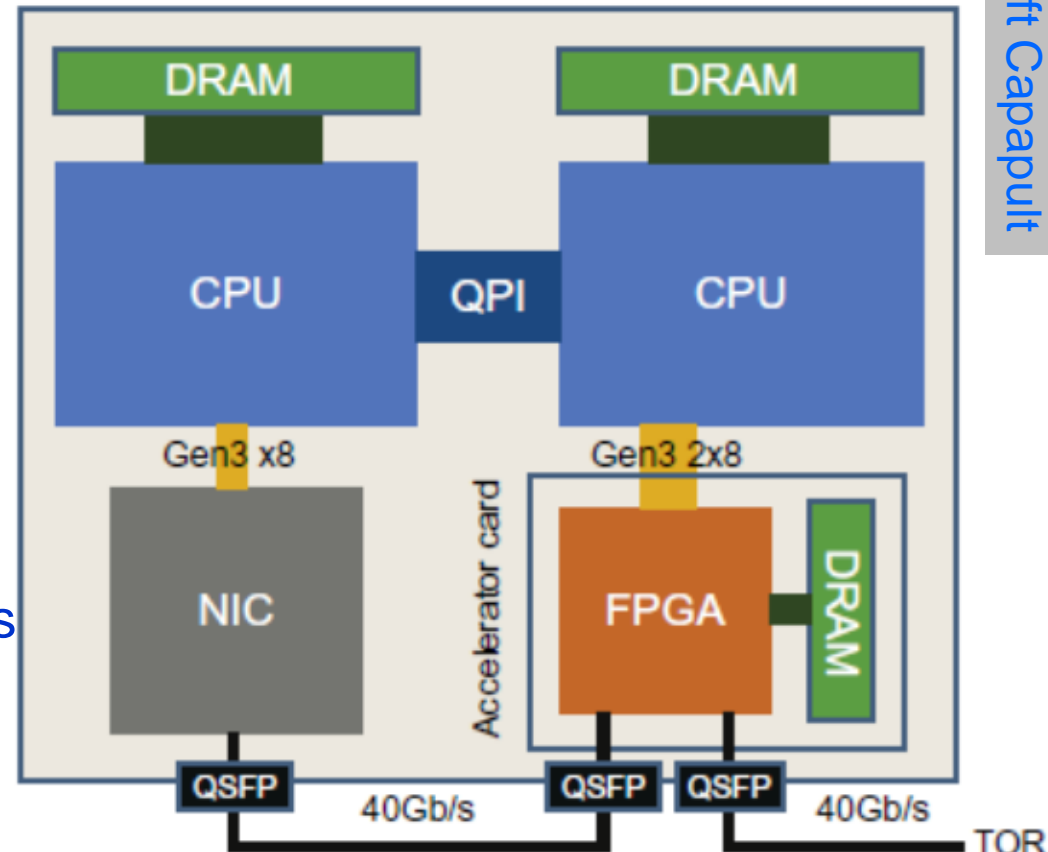
# Microsoft Catapult: Search Ranking

- Feature extraction (1 FPGA)
  - Extracts 4500 features for every document-query pair, e.g. frequency in which the query appears in the page
  - Systolic array of FSMs
- Free-form expressions (2 FPGAs)
  - Calculates feature combinations
- Machine-learned Scoring (1 FPGA for compression, 3 FPGAs calculate score)
  - Uses results of previous two stages to calculate floating-point score
- One FPGA allocated as a hot-spare



# Microsoft Catapult: Search Ranking

2-socket server blade



- Version 2 of Catapult
  - Placed the FPGA between the CPU and NIC
  - Increased network from 10 Gb/s to 40 Gb/s
  - Also performs network acceleration
  - Shell now consumes 44% of the FPGA
  - Now FPGA performs only feature extraction

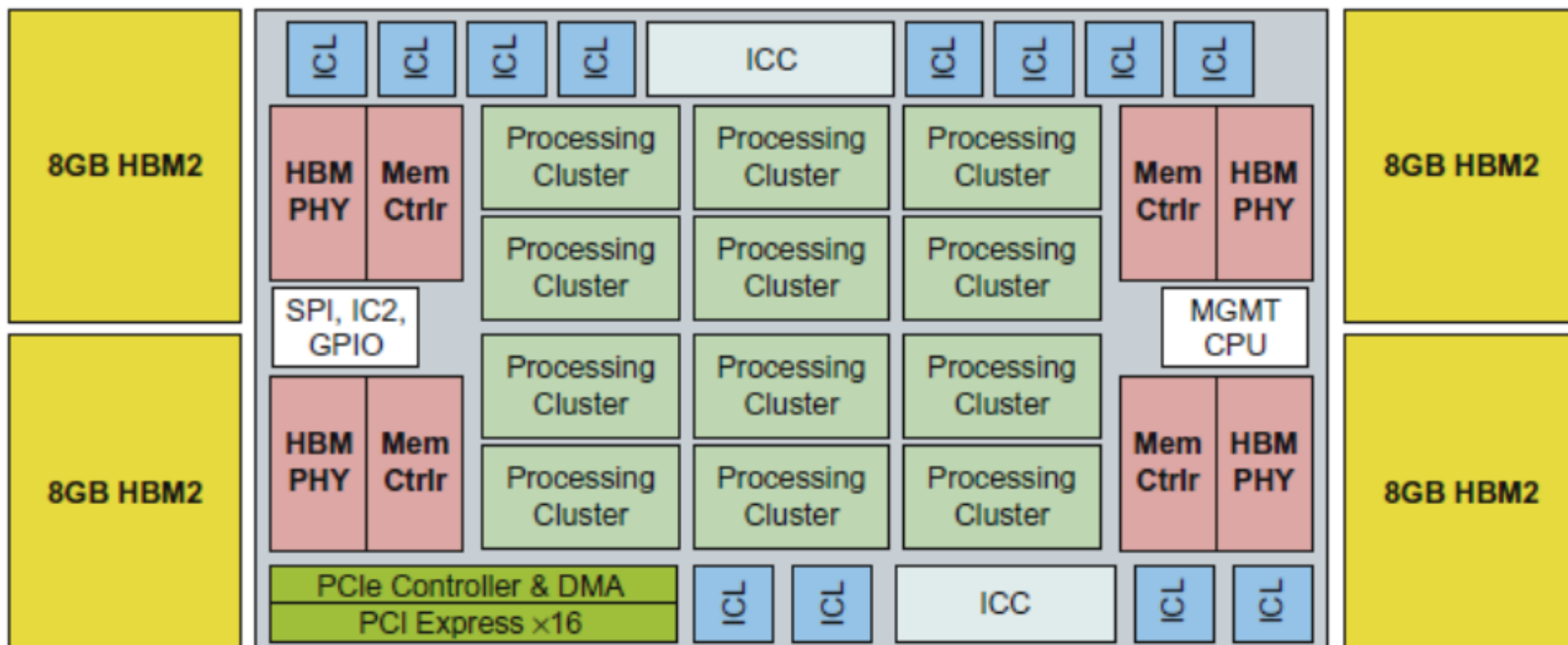
# Catapult and the Guidelines

- Use dedicated memories
  - 5 MiB dedicated memory
- Invest resources in arithmetic units and dedicated memories
  - 3926 ALUs
- Use the easiest form of parallelism that matches the domain
  - 2D SIMD for CNN, MISD parallelism for search scoring
- Reduce the data size and type needed for the domain
  - Uses mixture of 8-bit integers and 64-bit floating-point
- Use a domain-specific programming language
  - Uses Verilog RTL; Microsoft did not follow this guideline

# Intel Crest (now “Nervana”)

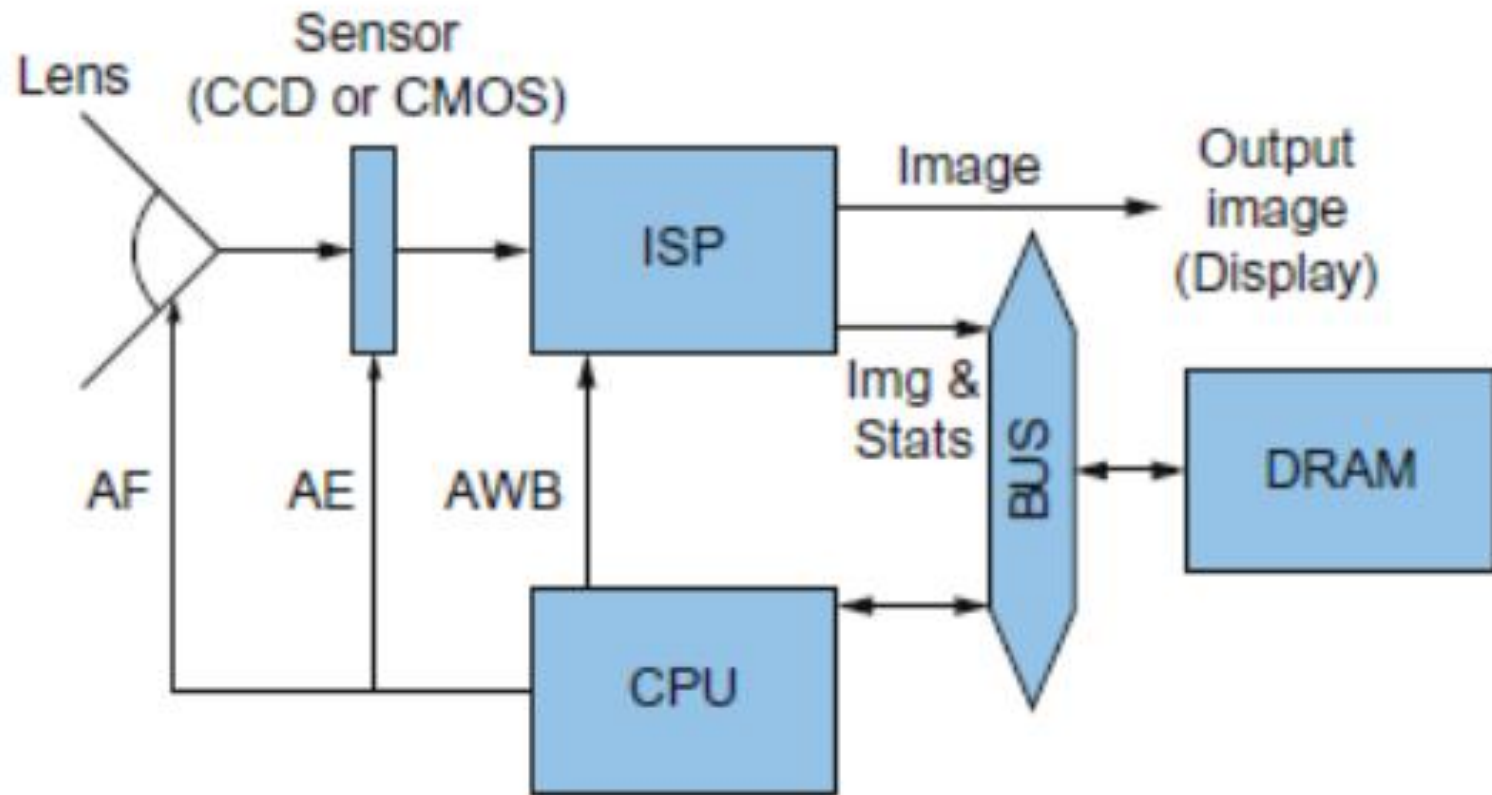
- DNN training
- 16-bit fixed point
- Operates on blocks of 32x32 matrices
- SRAM + HBM2

Interposer



# Google Pixel Visual Core

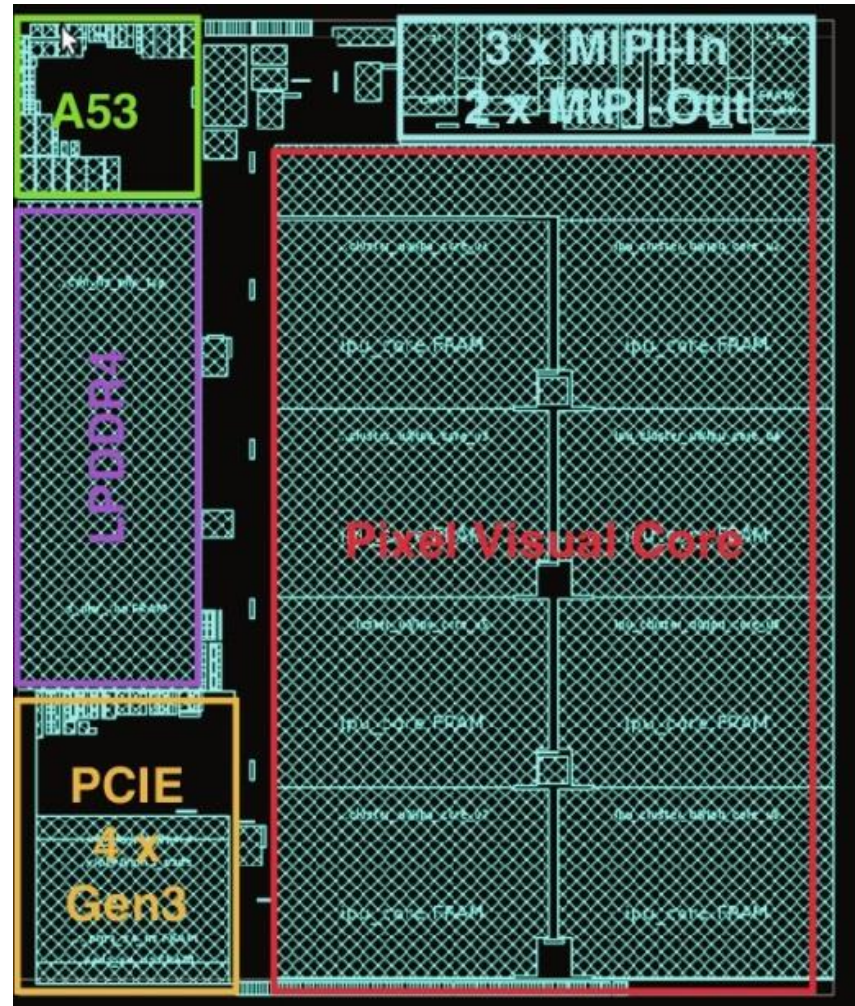
- Pixel Visual Core
  - Image Processing Unit
  - Performs stencil operations
  - Descended from Image Signal processor



# Pixel Visual Core

- Software written in Halide, a DSL
  - Compiled to virtual ISA
  - vISA is lowered to physical ISA using application-specific parameters
  - pISA is VLSI
- Optimized for energy
  - Power Budget is 6 to 8 W for bursts of 10-20 seconds, dropping to tens of milliwatts when not in use
  - 8-bit DRAM access equivalent energy as 12,500 8-bit integer operations or 7 to 100 8-bit SRAM accesses
  - IEEE-754 floating-point operations require 22X to 150X of the cost of 8-bit integer operations
- Optimized for 2D access
  - 2D SIMD unit
  - On-chip SRAM structured using a square geometry

# Pixel Core Layout



**Figure 7.38 Floor plan of the 8-core Pixel Visual Core chip.** A53 is an ARMv7 core. LPDDR4 is a DRAM controller. PCIE and MIPI are I/O buses.



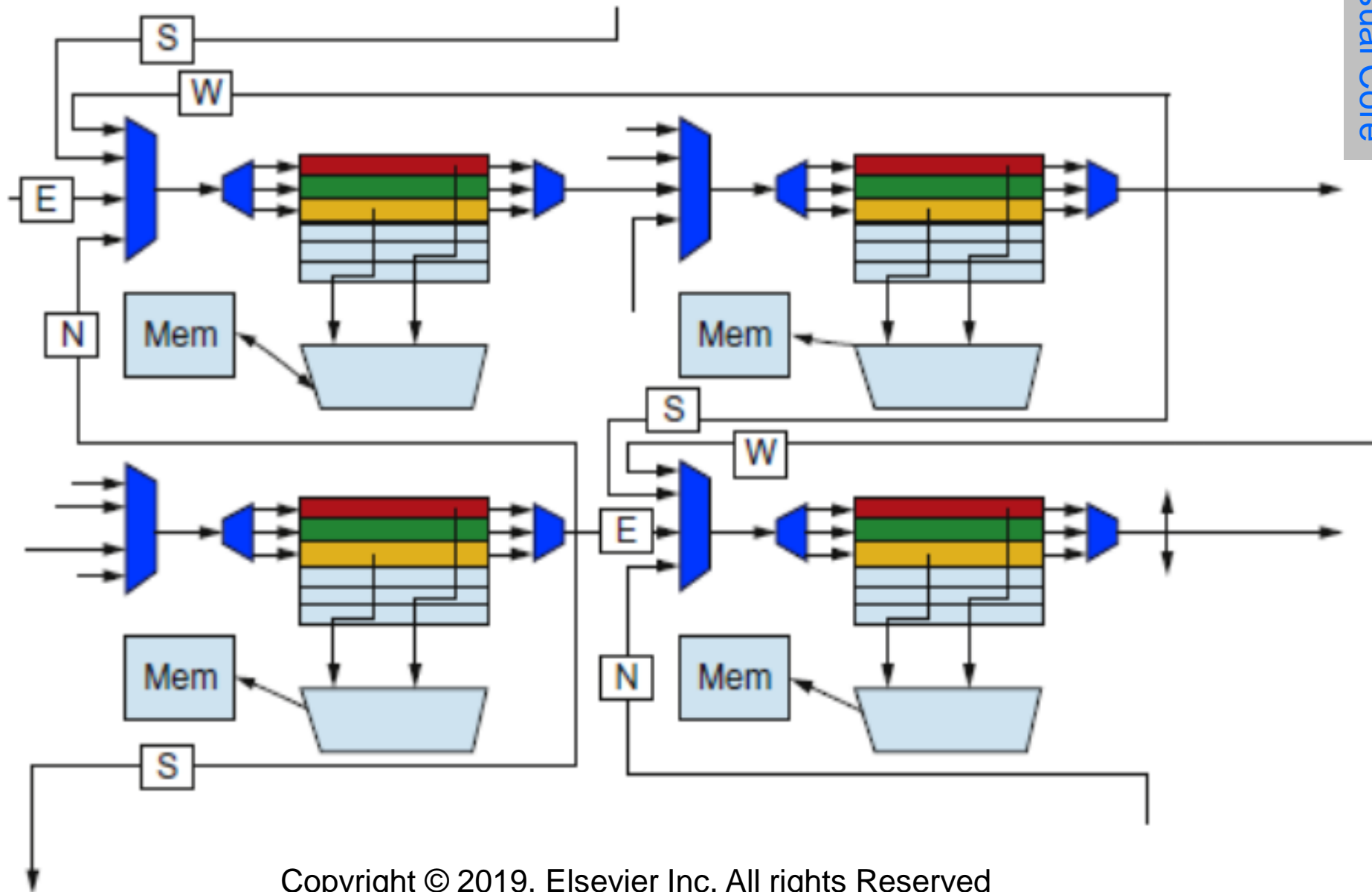
# Pixel Core VLIW Format

Field	Scalar	Math	Memory	Imm	MemImm
# Bits	43	38	12	16	10

**Figure 7.35 VLIW format of the 119-bit pISA instruction.**

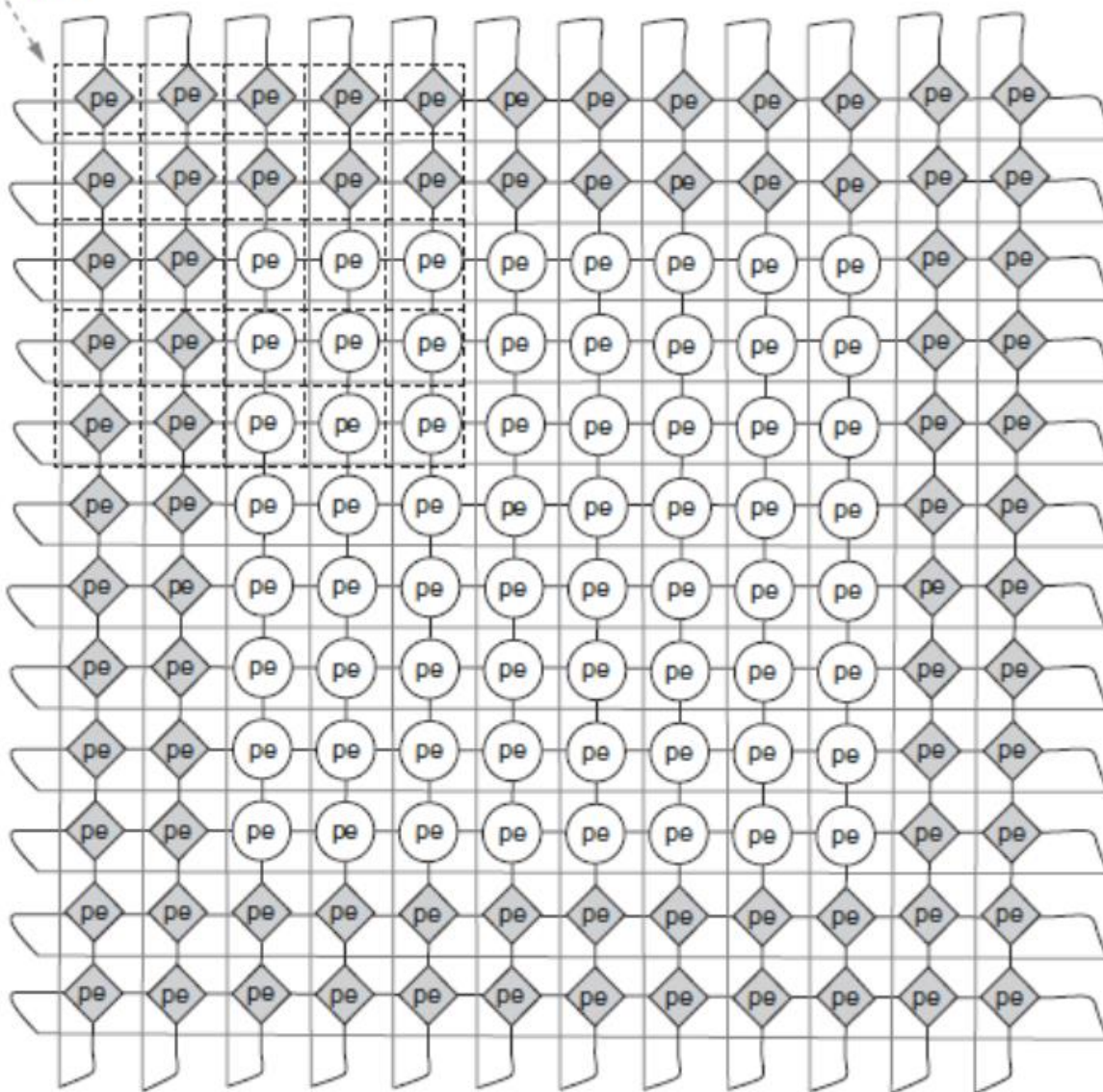


# Pixel Visual Core

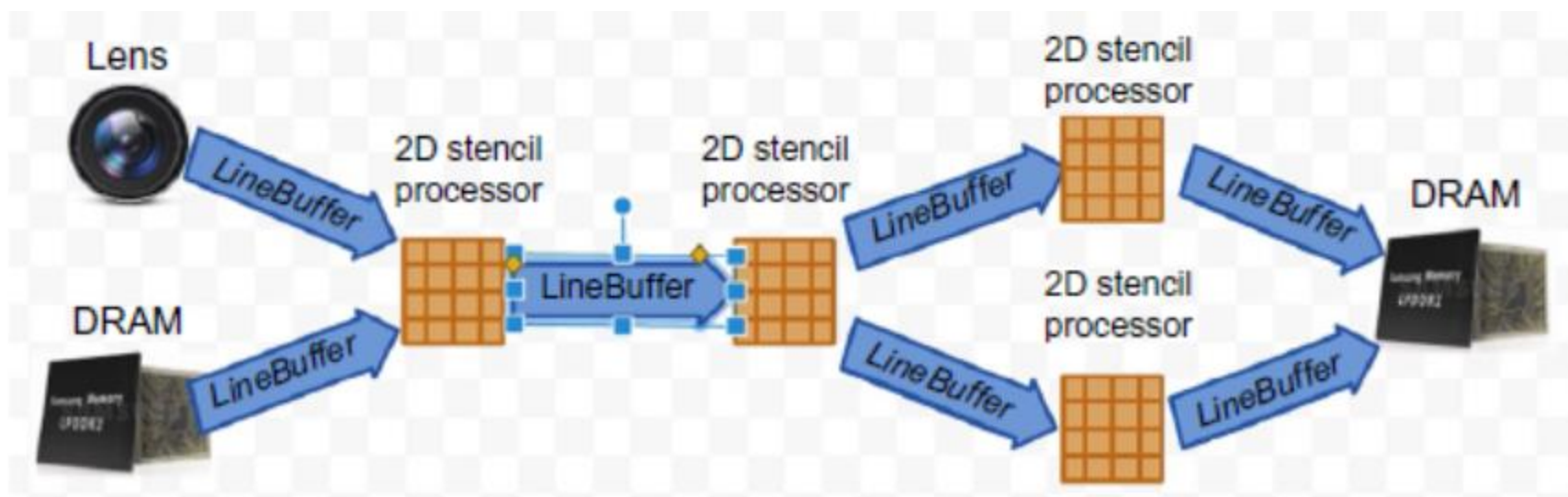


# Pixel Visual Core

5 x 5 stencil



# Pixel Visual Core



# Visual Core and the Guidelines

- Use dedicated memories
  - 128 + 64 MiB dedicated memory per core
- Invest resources in arithmetic units and dedicated memories
  - 16x16 2D array of processing elements per core and 2D shifting network per core
- Use the easiest form of parallelism that matches the domain
  - 2D SIMD and VLIW
- Reduce the data size and type needed for the domain
  - Uses mixture of 8-bit and 16-bit integers
- Use a domain-specific programming language
  - Halide for image processing and TensorFlow for CNNs

# Computer Architecture in the future

- Explosion of interest in custom architectures due to end of transistor scaling
  - Full employment for computer architects!
- But need to learn about application domains
  - Cannot just work with precompiled binaries anymore!
- Get involved in research projects,
  - ADEPT – microprocessor architecture and chip design
  - RISE – machine learning, datacenter software, and security
- Research experience is the most important part of application to top grad schools!