DNPU: An 8.1TOPS/W Reconfigurable CNN-RNN Processor for General-Purpose Deep Neural Networks

Dongjoo Shin, **Jinmook Lee, Jinsu Lee, and Hoi-Jun Yoo**

Semiconductor System Laboratory
School of EE, KAIST

Outline

- Introduction
- Processor Architecture
- Key Features
  1. Heterogeneous architecture
  2. Mixed workload division method

  } Configurability

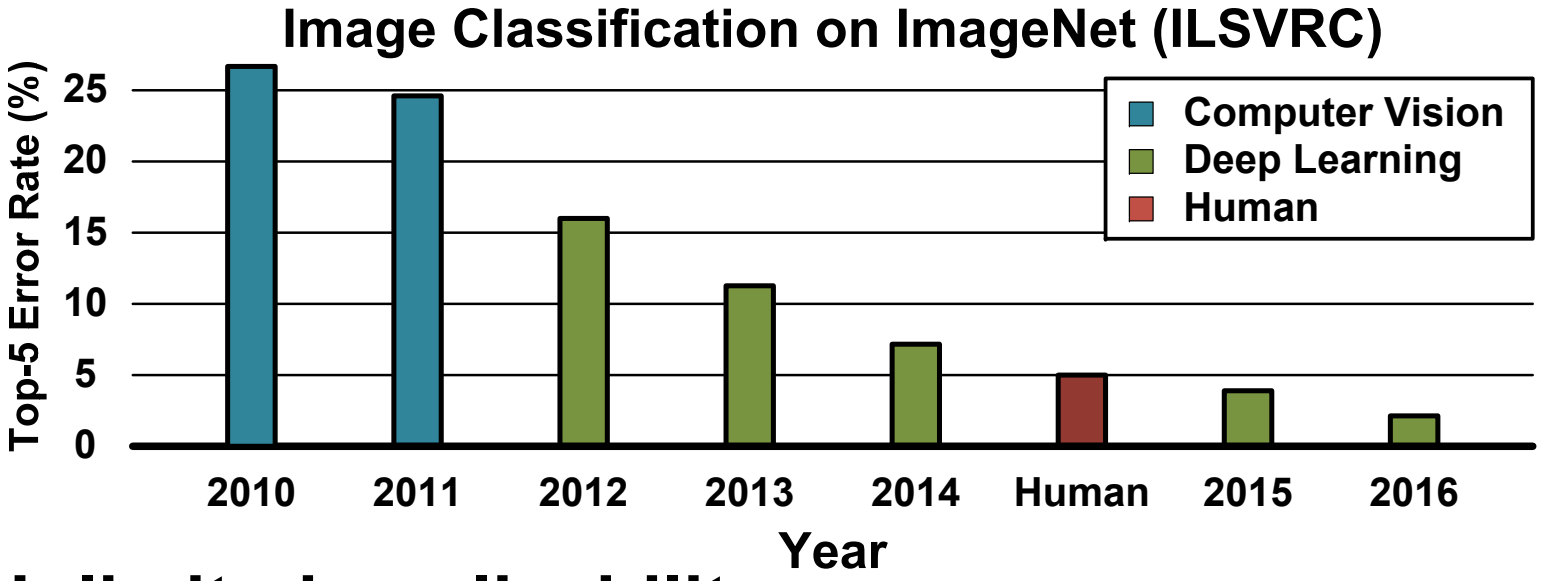
  3. Dynamic fixed-point with on-line adaptation
  4. Quantization-table-based multiplier

  } Low Power

- Implementation Results
- Conclusion

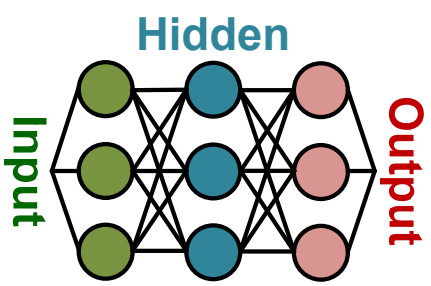
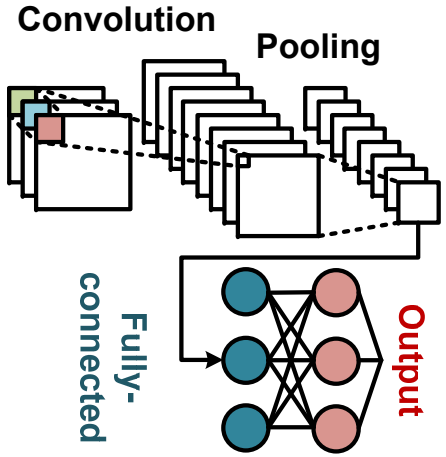
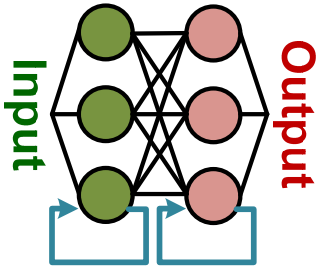
Why Deep Learning?

- **Overwhelming performance**



- **Unlimited applicability**
  - Image classification, action recognition, sign recognition…
  - Super resolution, depth map generation, …
  - Translation, natural language processing, …
  - Text generation, image captioning, composition, …

Types of Deep Learning

	MLP (multi-layer perceptron)	CNN (convolutional)	RNN (recurrent)
			
Chracteristic	**Fully-connected Layer**	**Convolutional layer**	**Feedback path, internal state**
Major Application	**Simple classification**, hand written letter recognition ...	**Vision processing**, face recognition, image classification, ...	**Sequential data processing**, translation, speech recognition, ...
Layers	3~10 layers	5~100s layers	3~5 layers

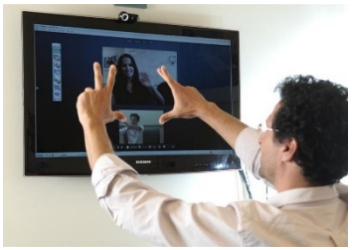
Why both CNN & RNN?

- CNN: Visual feature extraction and recognition
  - Face recognition, image classification…
- RNN: Sequential data recognition and generation
  - Translation, speech recognition…
- CNN + RNN: CNN-extracted features → RNN input

Image Captioning

CNN: Visual feature extraction
RNN: Sentence generation

Action Recognition

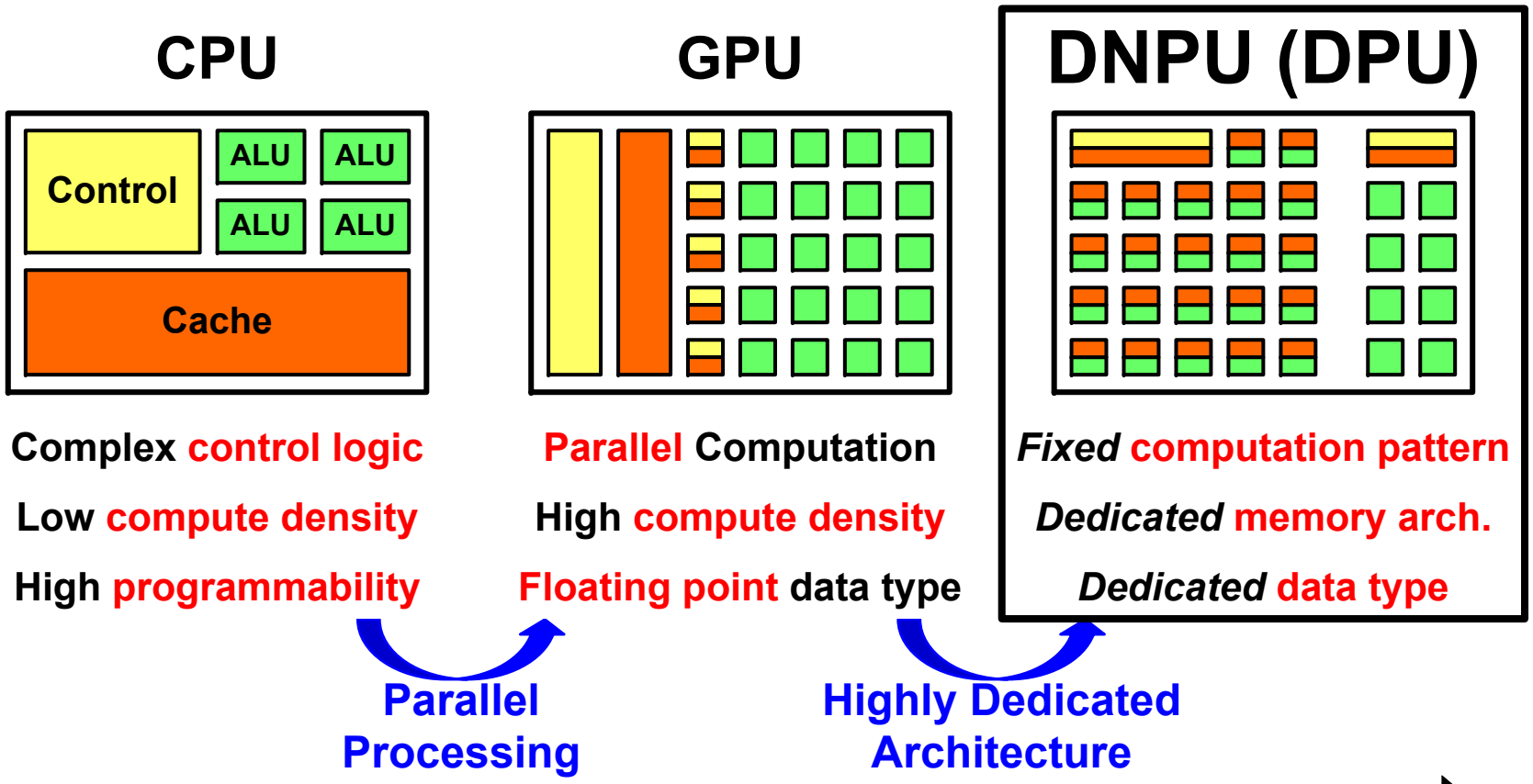


CNN: Visual feature extraction
RNN: Temporal information recognition

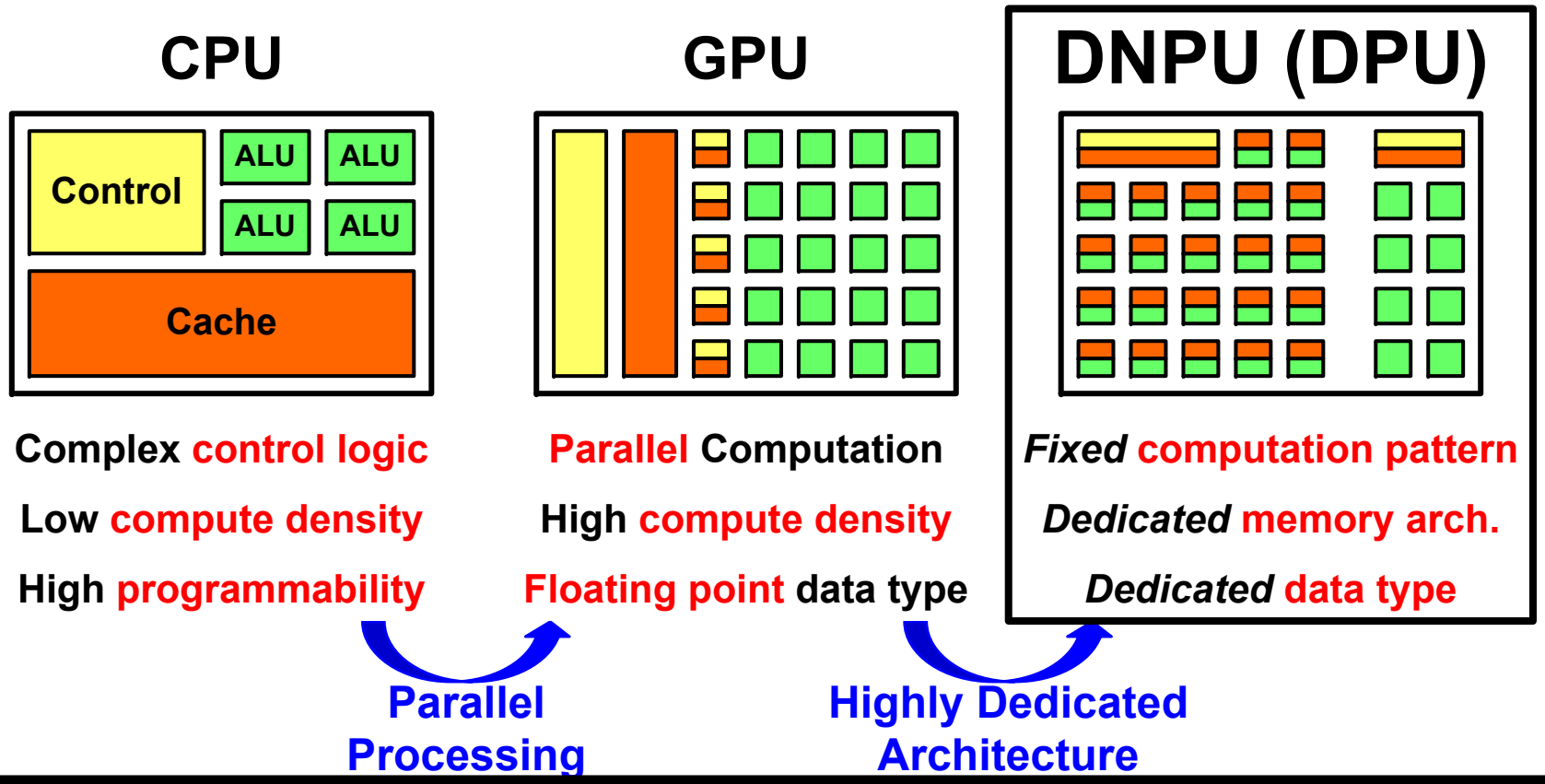
- Previous works
  - Optimized for convolution layer only: [1], [2], [3]
  - Optimized for FC layer and RNN only: [4]

[1] Y. Chen, ISSCC 2016 [2] B. Moons, SOVC 2016 [3] J. Sim, ISSCC 2016 [4] S. Han, ISCA 2016

Deep Learning-dedicated SoC: DNPU



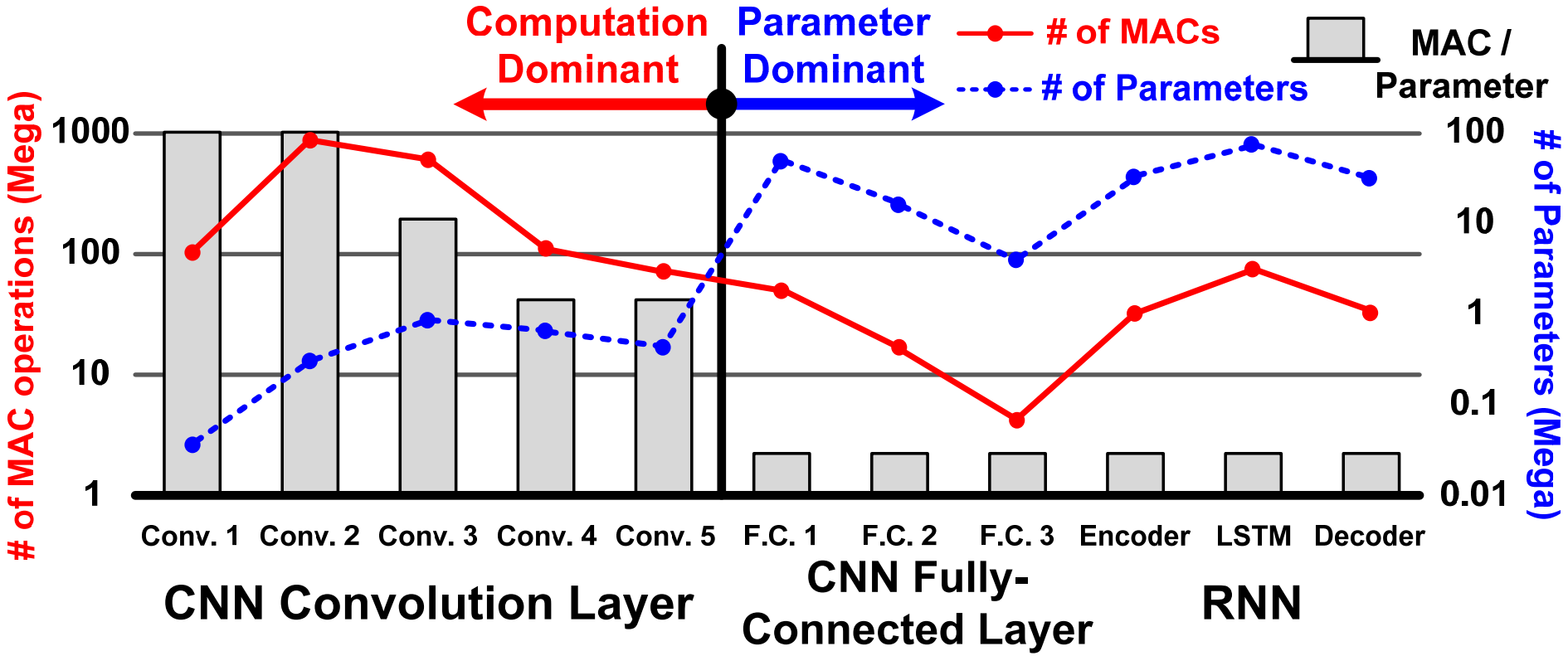
Deep Learning-dedicated SoC: DNPU



Deep learning **itself has high adaptability for various applications**

Design Challenges

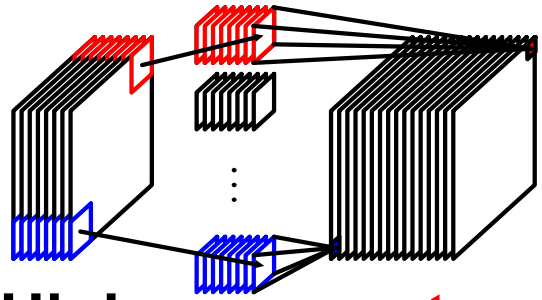
- **Heterogeneous characteristics**

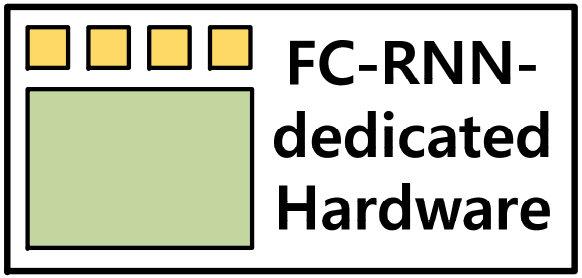


- Conv. Layer (CNN): Computation >> Parameter
- FC Layer (CNN), RNN: Computation ~= Parameter

Design Challenges

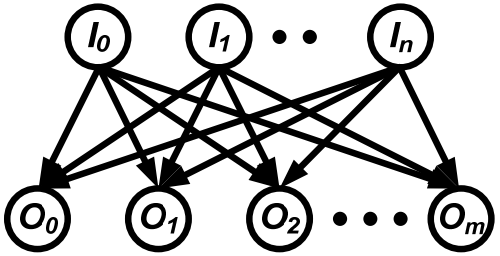
- **Acceleration of Conv. with FC-RNN dedicated HW**

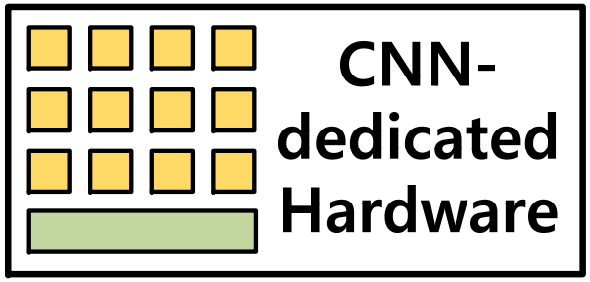




  - High **memory transaction**
  - Low **PE utilization**
  - Mismatch of the **computational patterns**

- **Acceleration of FC-RNNs with Conv. dedicated HW**

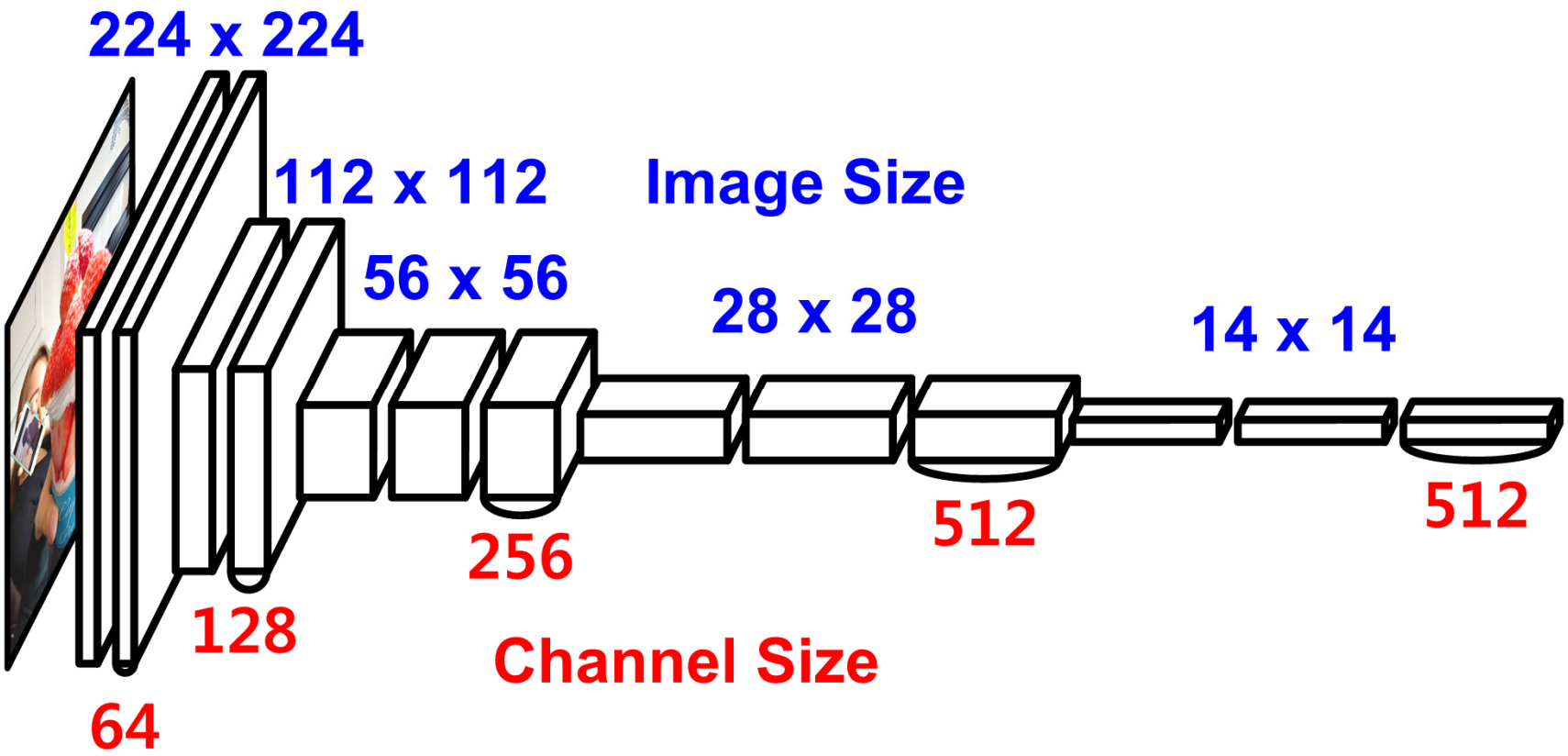




  - Cannot exploit **reusability** (image, kernel, convolution)
  - Cannot achieve required **throughput**

Design Challenges

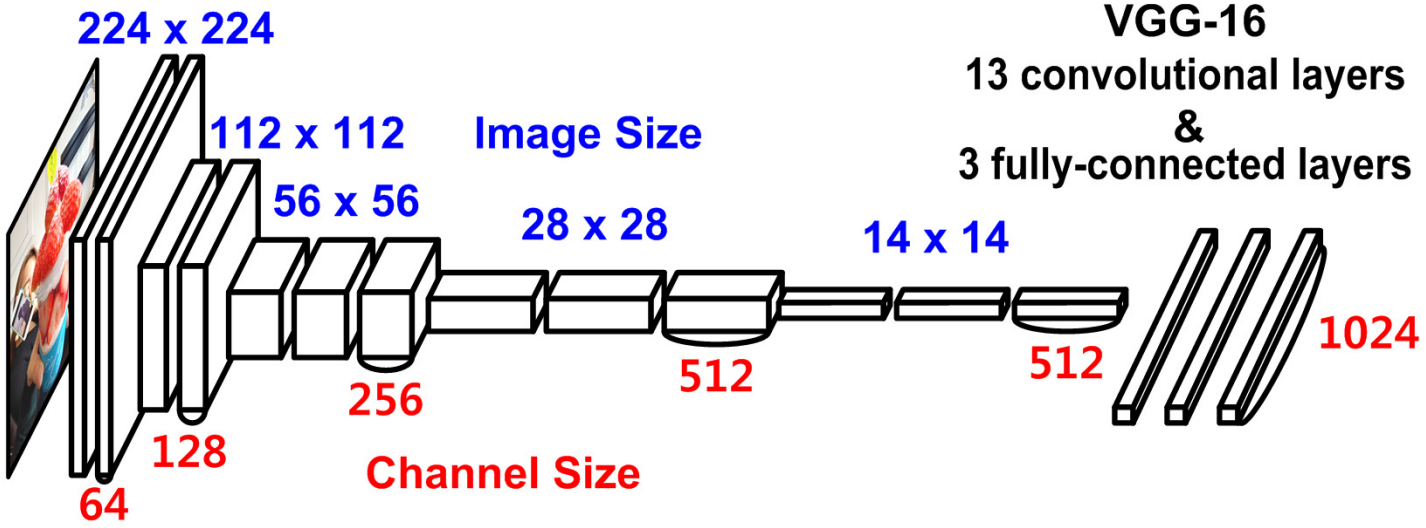
- Various configurations of layers
  - Image size, channel size, kernel (filter) size



Example: VGG-16 Convolution Layers

Design Challenges

- **Large amount of data & massive MACs**



	Convolution Layer													FC Layer		
	1	2	3	4	5	6	7	8	9	10	11	12	13	1	2	3
# of Kernel Weights (M)	0.01	0.04	0.07	0.14	0.28	0.56	0.56	1.1	2.2	2.2	2.2	2.2	2.2	49	8	2
# of MACs (Gops)	0.08	1.7	0.86	1.7	0.86	1.7	1.7	0.86	1.7	1.7	0.43	0.43	0.43	0.09	0.02	0.01

Total number of kernel weights: 70 M Total number of MACs: 14 G

for only one inference (@ VGG-16)

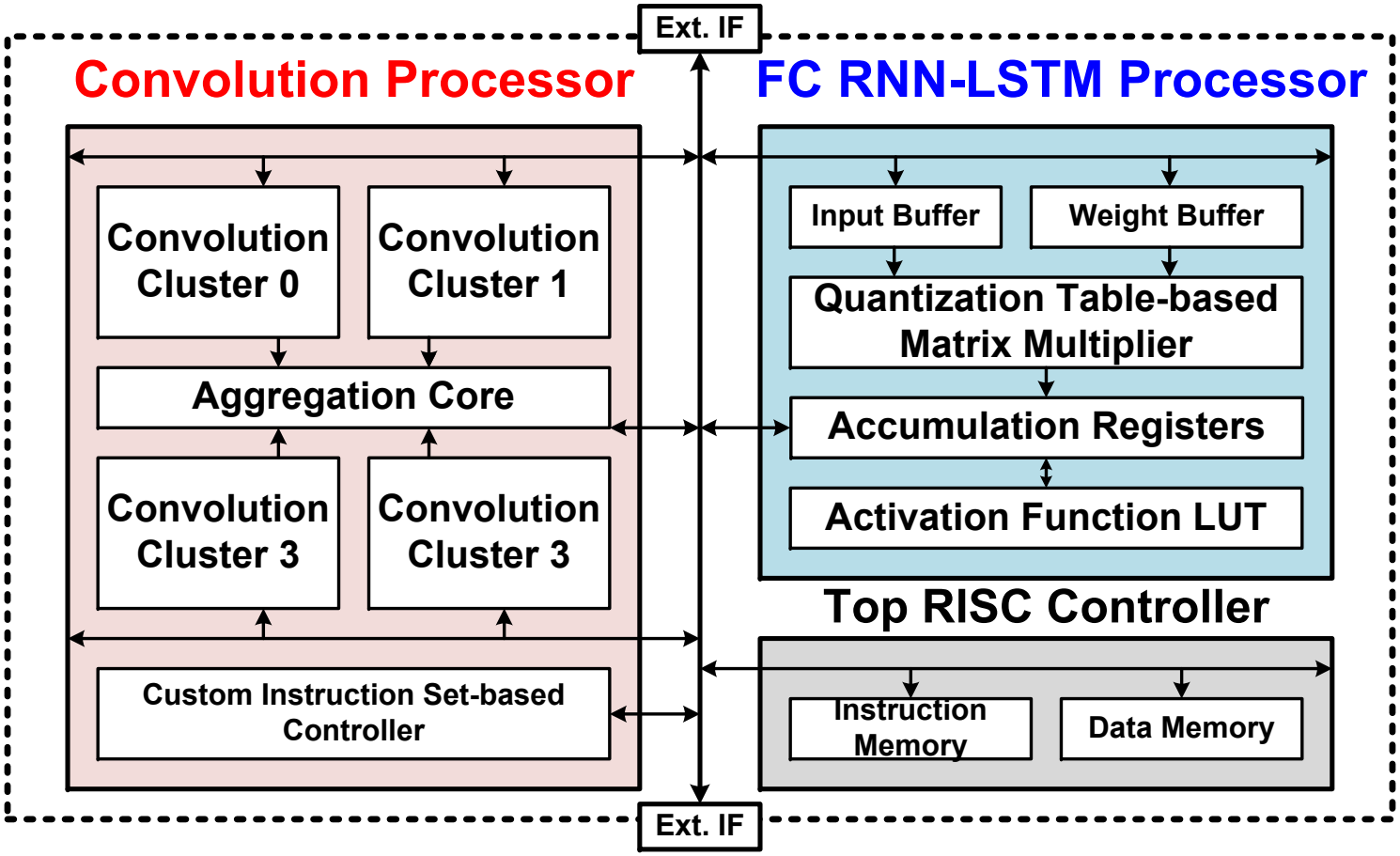
Proposed Processor: DNPU

- **For High Configurability,**
  1. Heterogeneous Architecture
  2. Mixed Workload Division

- **For Low-power Operation,**
  3. Dynamic Fixed-point with On-line Adaptation
  4. Quantization-table-based Multiplication

An 8.1TOPS/W CNN-RNN Processor for General-purpose DNNs

Overall Architecture



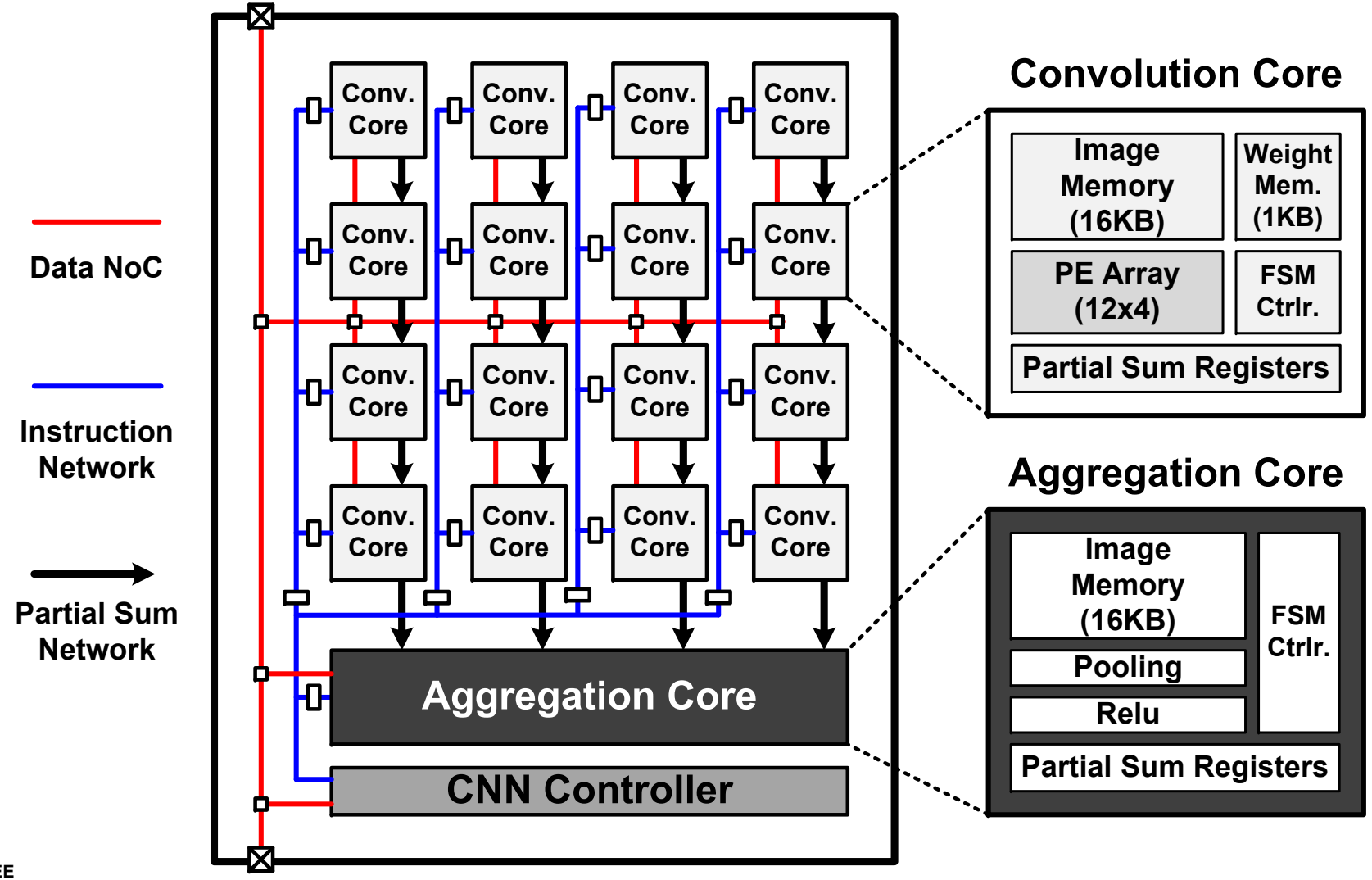
Convolution layer (CNN)

Heterogeneous Architecture

Fully-connected layer (CNN)
Recurrent neural network

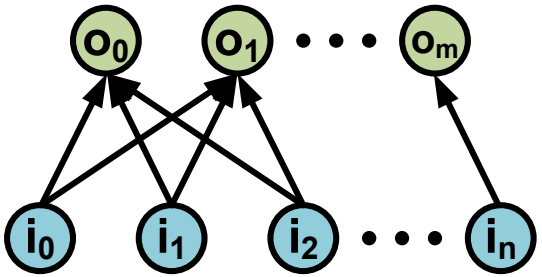
Convolution Processor

- **Distributed memory-based architecture**



FC Layer and RNN Hardware Sharing

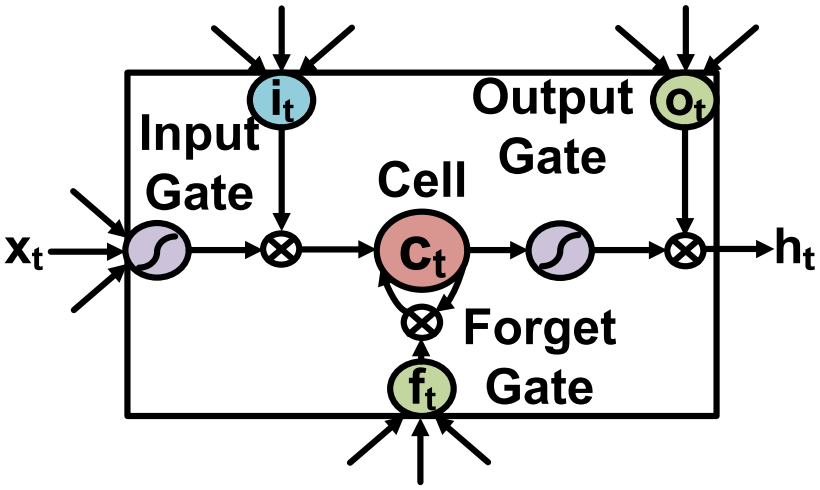
- **FC layer - Mapping to matrix multiplication**



$$o_m = W_{0m} i_0 + W_{1m} i_1 + \cdots + W_{nm} i_n + b_m$$

$$\begin{bmatrix} 1 & i_0 & i_1 & \cdots & i_n \end{bmatrix} \times \begin{bmatrix} b_0 & b_1 & \cdots & b_m \\ W_{10} & W_{11} & \cdots & W_{1m} \\ \vdots & \vdots & & \vdots \\ W_{n0} & W_{n1} & \cdots & W_{nm} \end{bmatrix}$$

- **RNN – Mapping to matrix multiplication**



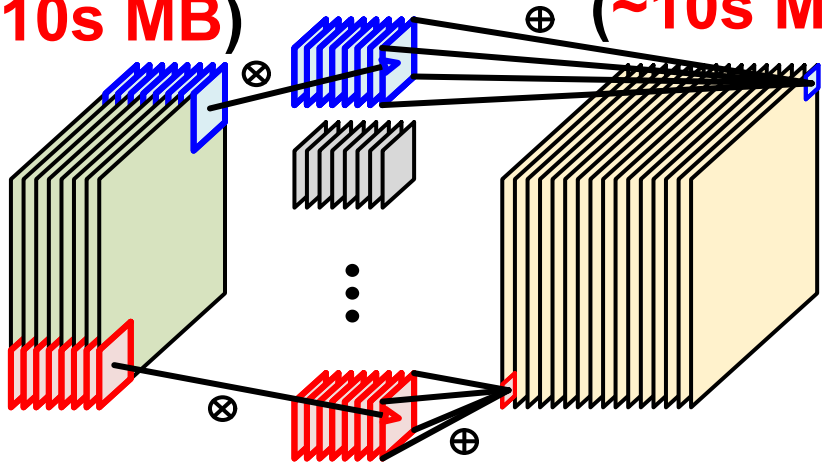
$$i_t = \sigma\left(W_{xi} x_t + W_{hi} h_{t-1} + b_i\right)$$
$$f_t = \sigma\left(W_{xf} x_t + W_{hf} h_{t-1} + b_f\right)$$
$$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc} x_t + W_{hc} h_{t-1} + b_c\right)$$
$$o_t = \sigma\left(W_{xo} x_t + W_{ho} h_{t-1} + b_o\right)$$

$$\begin{bmatrix} 1 & x_t & h_{t-1} \end{bmatrix} \times \begin{bmatrix} b_i & b_f & b_o & b_c \\ W_{xi} & W_{xf} & W_{xo} & W_{xc} \\ W_{hi} & W_{hf} & W_{ho} & W_{hc} \end{bmatrix}$$

Limited On-chip Memory Size

- Conv. operation with limited on-chip memory

❶ Input Layer Images (~10s MB) ⊗ Conv. Kernels (~10 MB) ⊕ Output Layer Images (~10s MB)



Convolution Operation

❷ On-chip memory size

100s KB ~ 1s MB

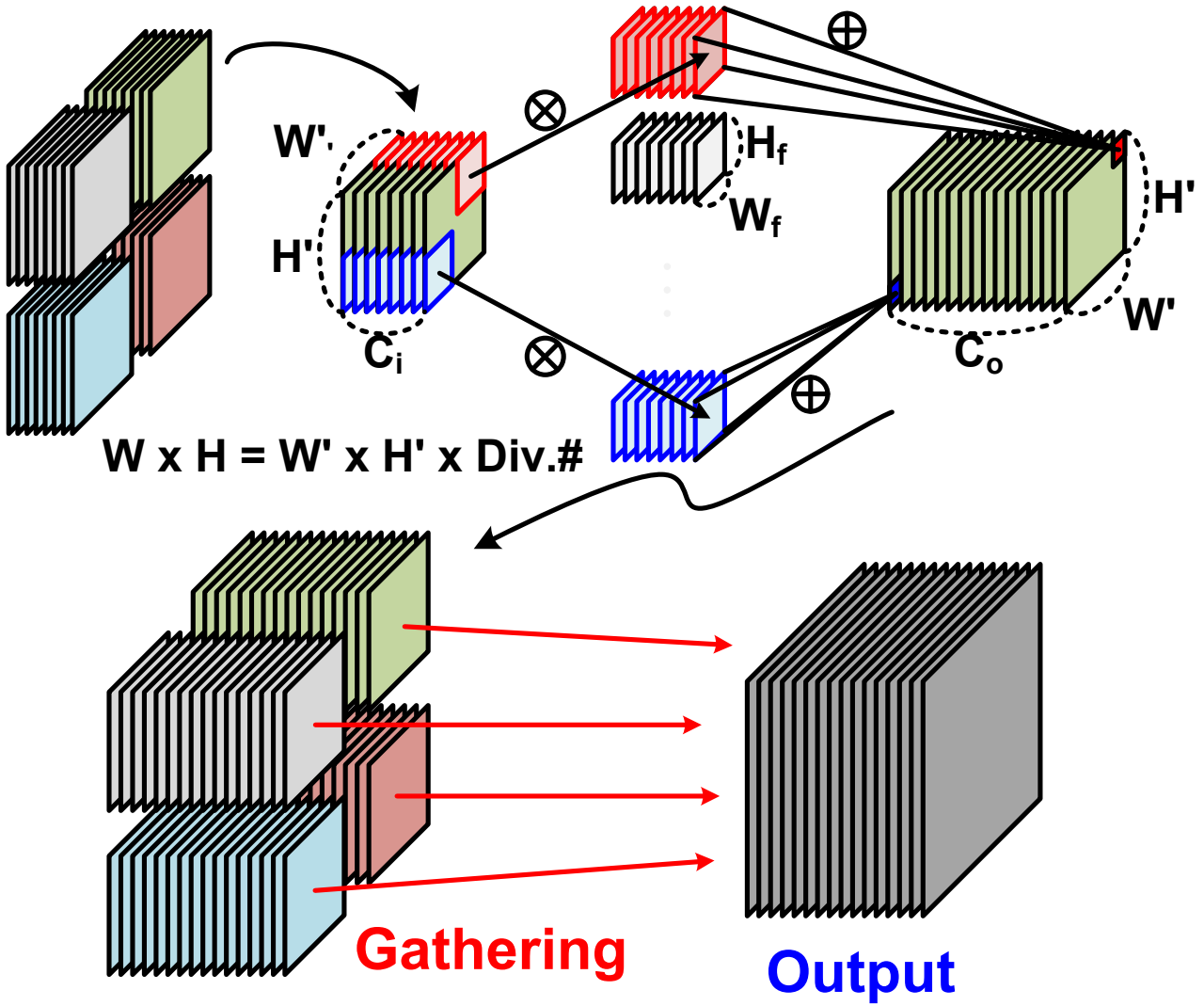
❸ On-chip memory ⟷ On-chip processing elements

Trade-off

→ Workload should be divided to reduce the required on-chip memory size

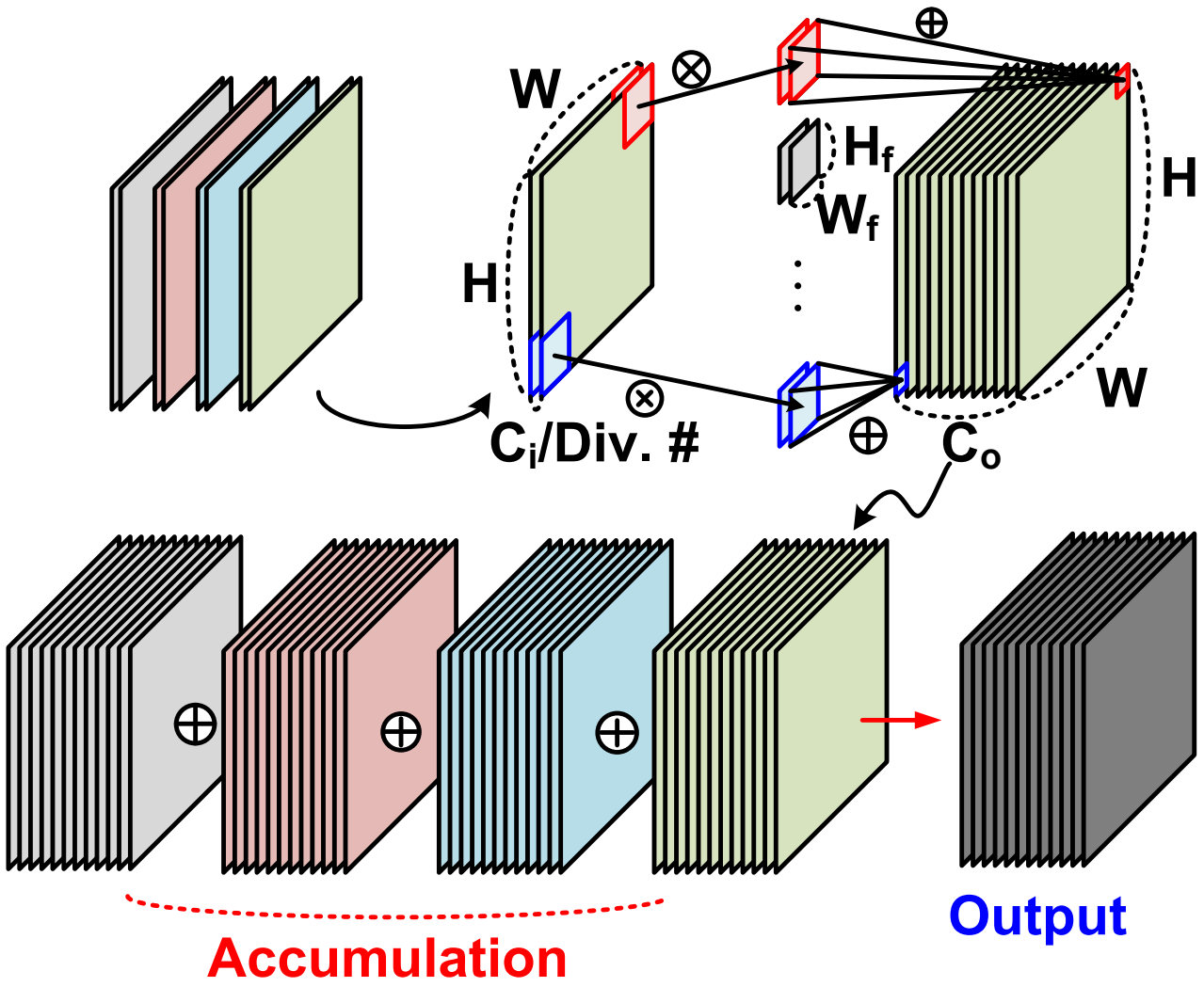
Image Division

- Workload division with *image direction*



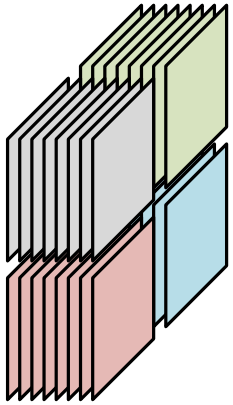
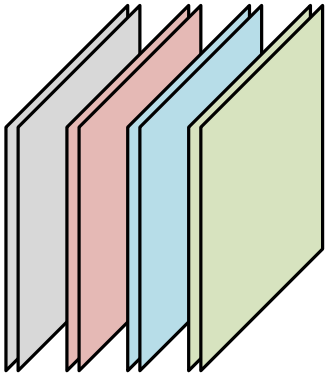
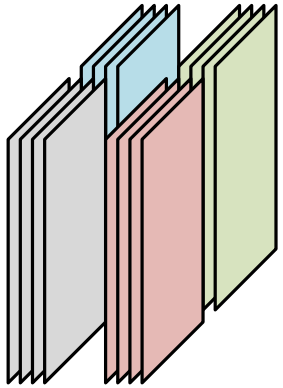
Channel Division

- Workload division with *channel direction*



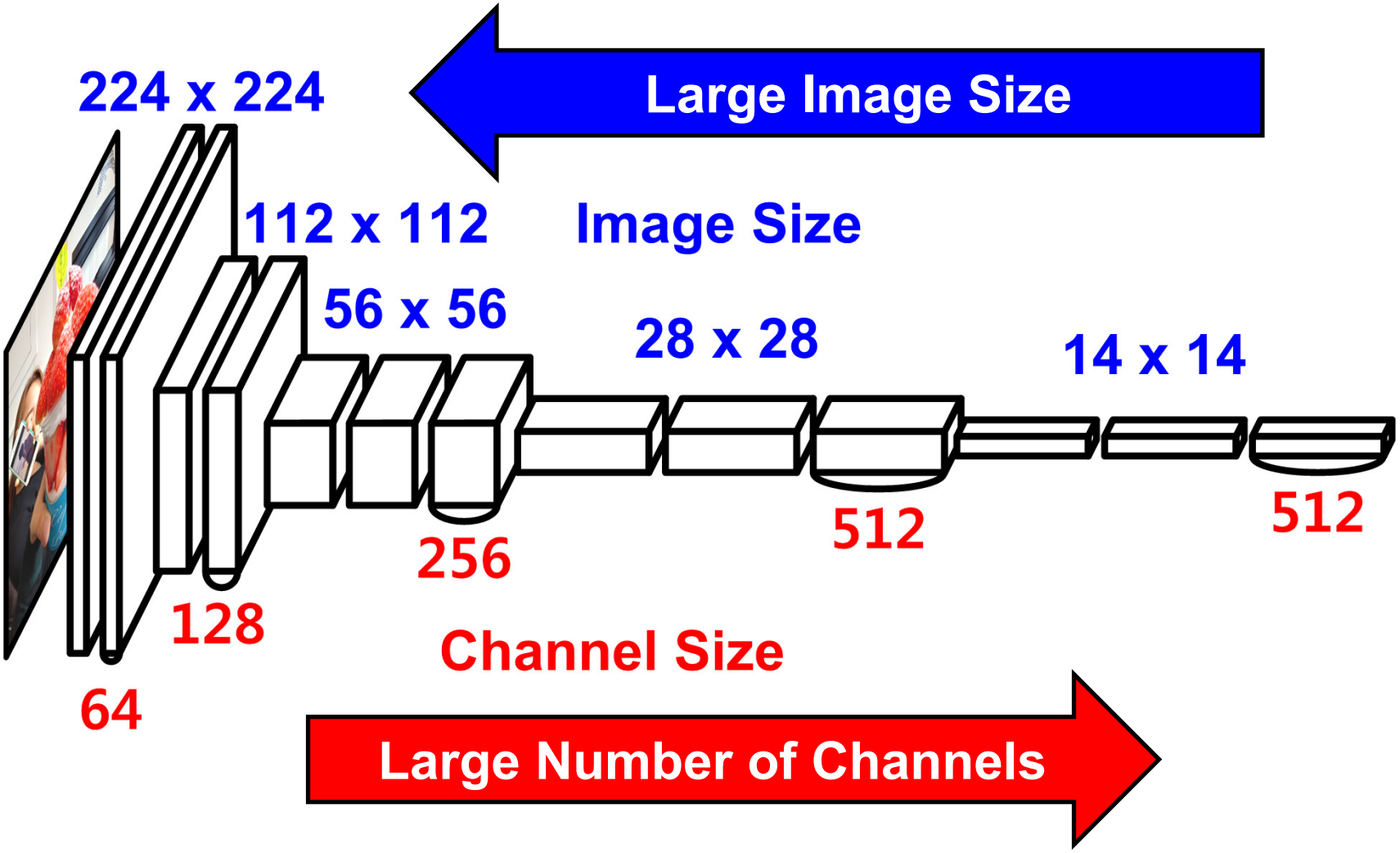
Mixed Division

- **Workload division with both directions**

Input Layer Division Method	Image Division	Channel Division	Mixed Division
			
Off-chip Access (W/O Compression Scheme)			
Input Image	$W_i \times H_i \times C_i$	$W_i \times H_i \times C_i$	$W_i \times H_i \times C_i$
Weight	$W_f \times H_f \times C_i \times C_o$ x Img. Div. #	$W_f \times H_f \times C_i \times C_o$	$W_f \times H_f \times C_i \times C_o$ x Img. Div. #
Output Image	$\frac{W_o \times H_o \times C_o}{\text{Pooling Size}}$	$W_o \times H_o \times C_o$ x Ch. Div. # x 2	$W_o \times H_o \times C_o$ x Ch. Div. #

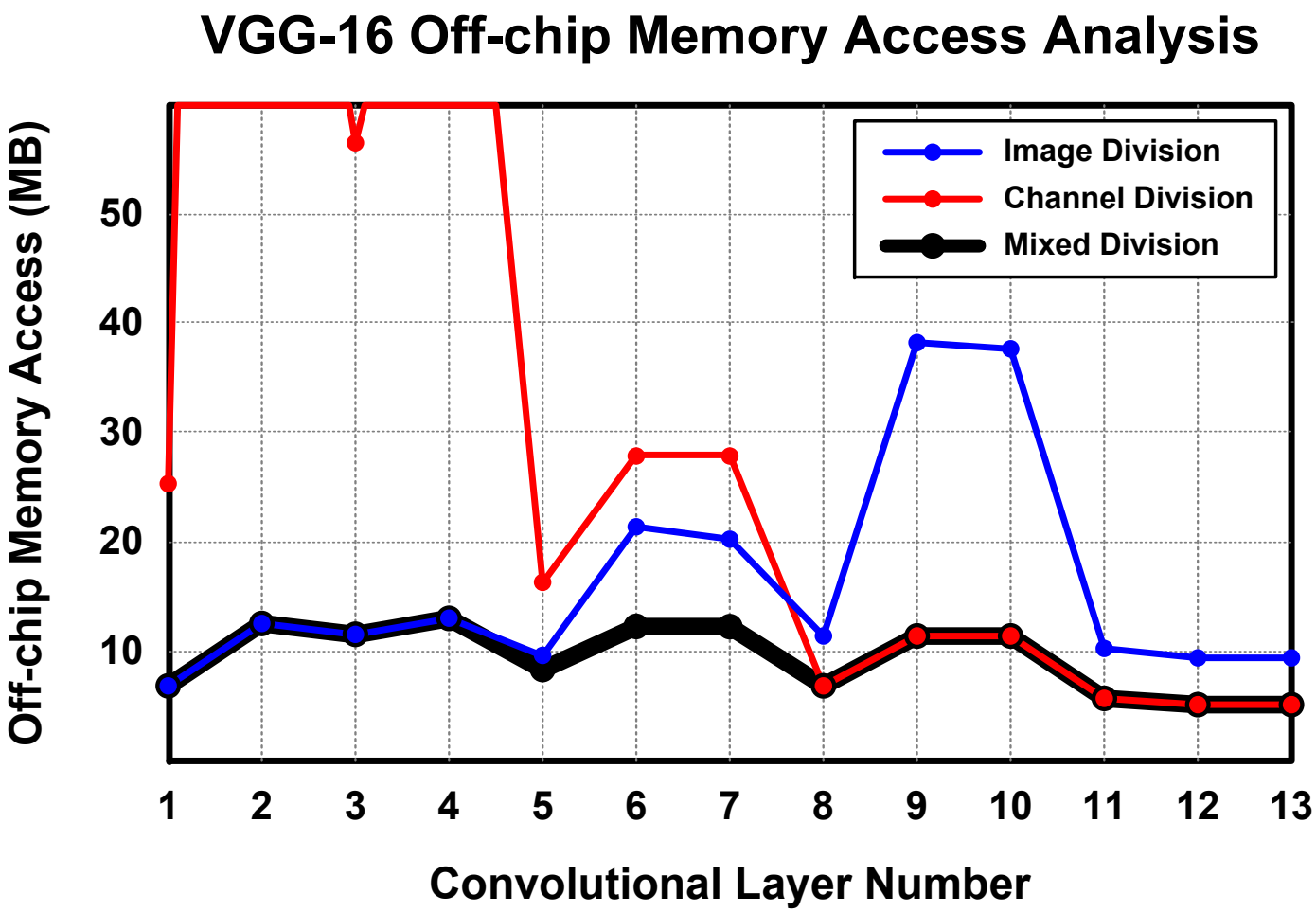
Variation Through Layers

- VGG-16 convolution layer example



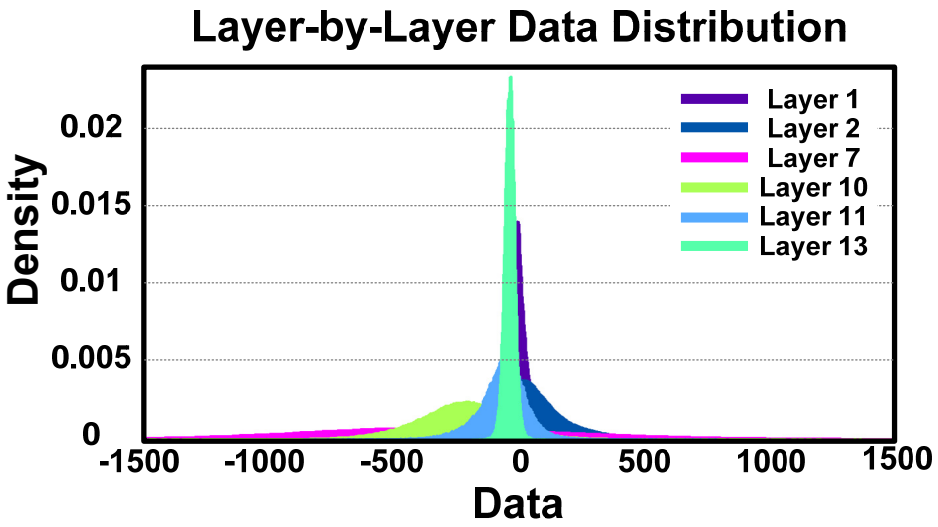
Off-chip Access Analysis

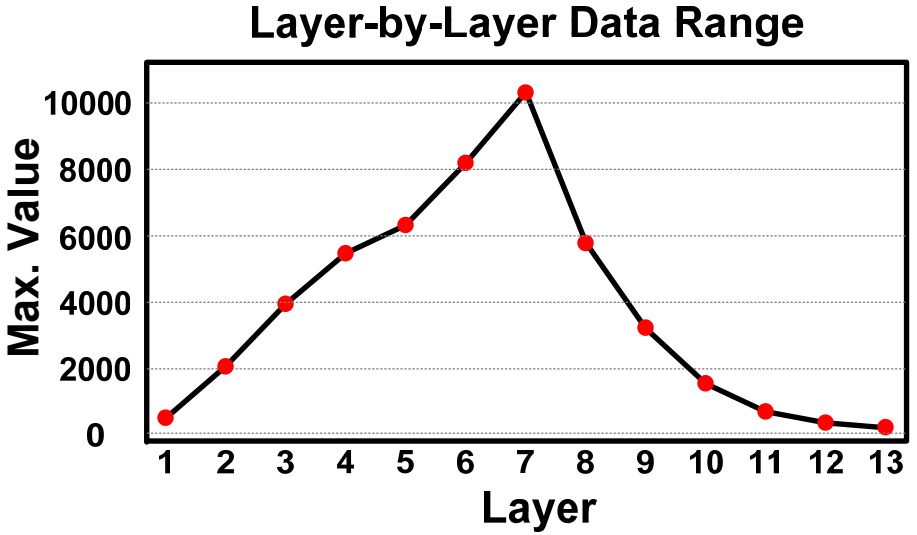
- **Mixed division can take lower points**



Layer-by-Layer Data Distribution

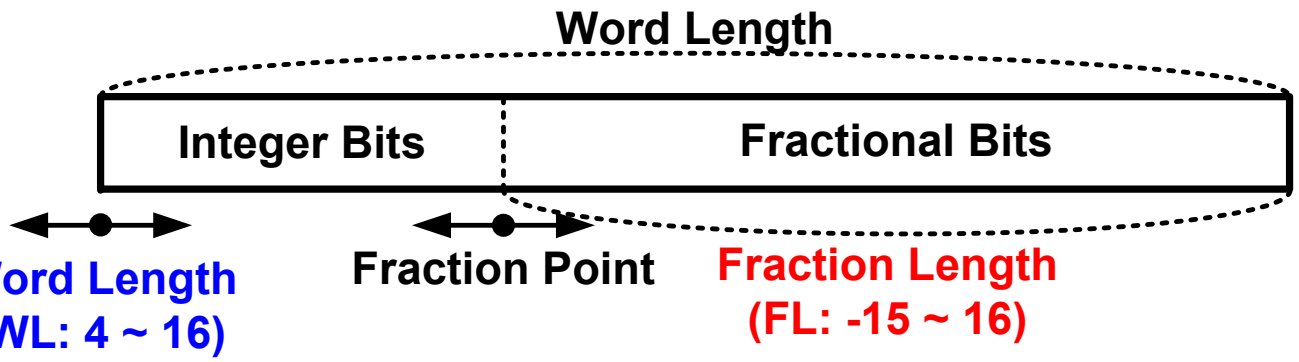
- Data distribution in each layer





- Floating-point implementation
  - Large data range, but high cost
- Fixed-point implementation
  - Low cost, but limited data range

Layer-by-Layer Dynamic Fixed-point



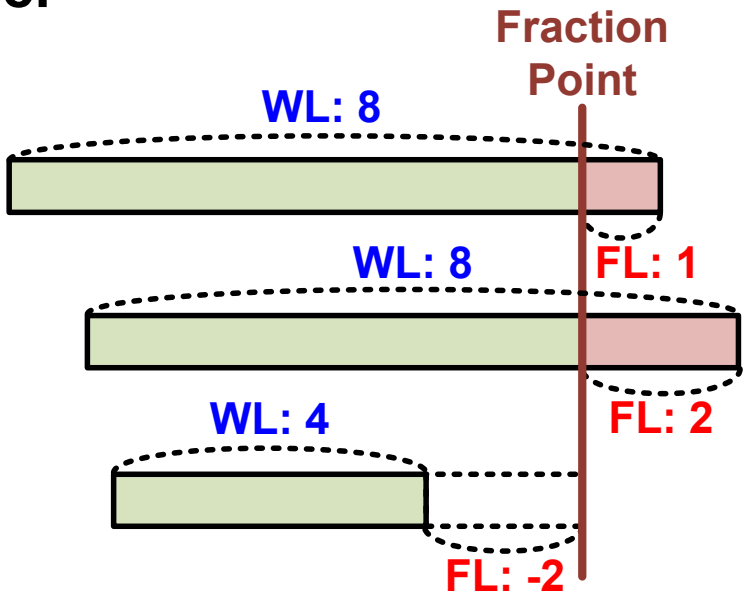
- **Each layer has different WL and FL**
  → Having floating-point characteristic via layers
- **WL and FL are fixed in a same layer**
  → Enabling fixed-point operation

Example)

Conv. Layer 1 – **WL: 8, FL: 1**

Conv. Layer 2 – **WL: 8, FL: 2**

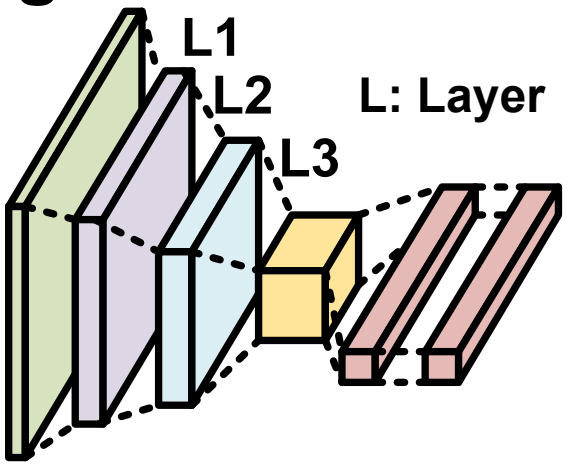
Conv. Layer 3 – **WL: 4, FL: -2**



Previous Approach

- **Off-line learning-based FL selection**

Image Data Set



Fraction Length (FL) Sets

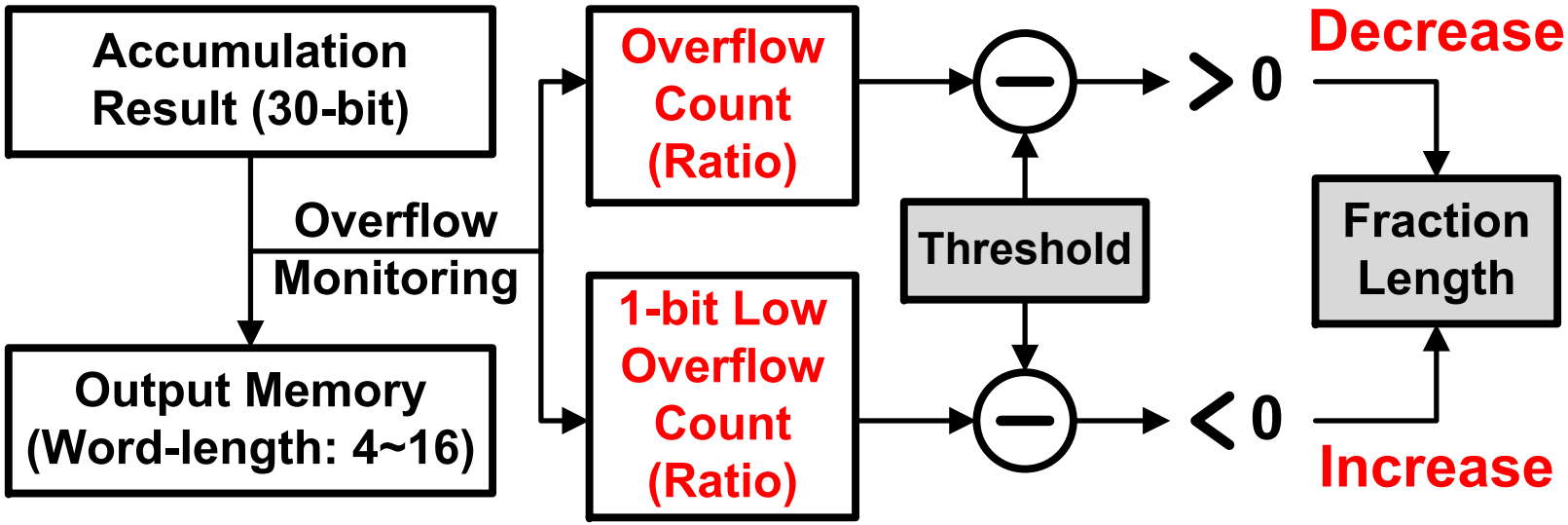
Set1 – L1: 0, L2: 0, L3:-4 …
Set2 – L1: 1, L2: -1, L3: -5 …
Set3 – L1: 1, L2: -3, L4: -6 …
⋮

Finding **FL set** which shows the **minimum error** with given **image data set**

- **Off-line (off-chip)** learning-based FL selection
- FL is ***trained to fit*** with ***given image data set***
- Selected FL is used for every image at run time

Proposed On-line Adaptation

- **On-line adaptation-based FL selection**



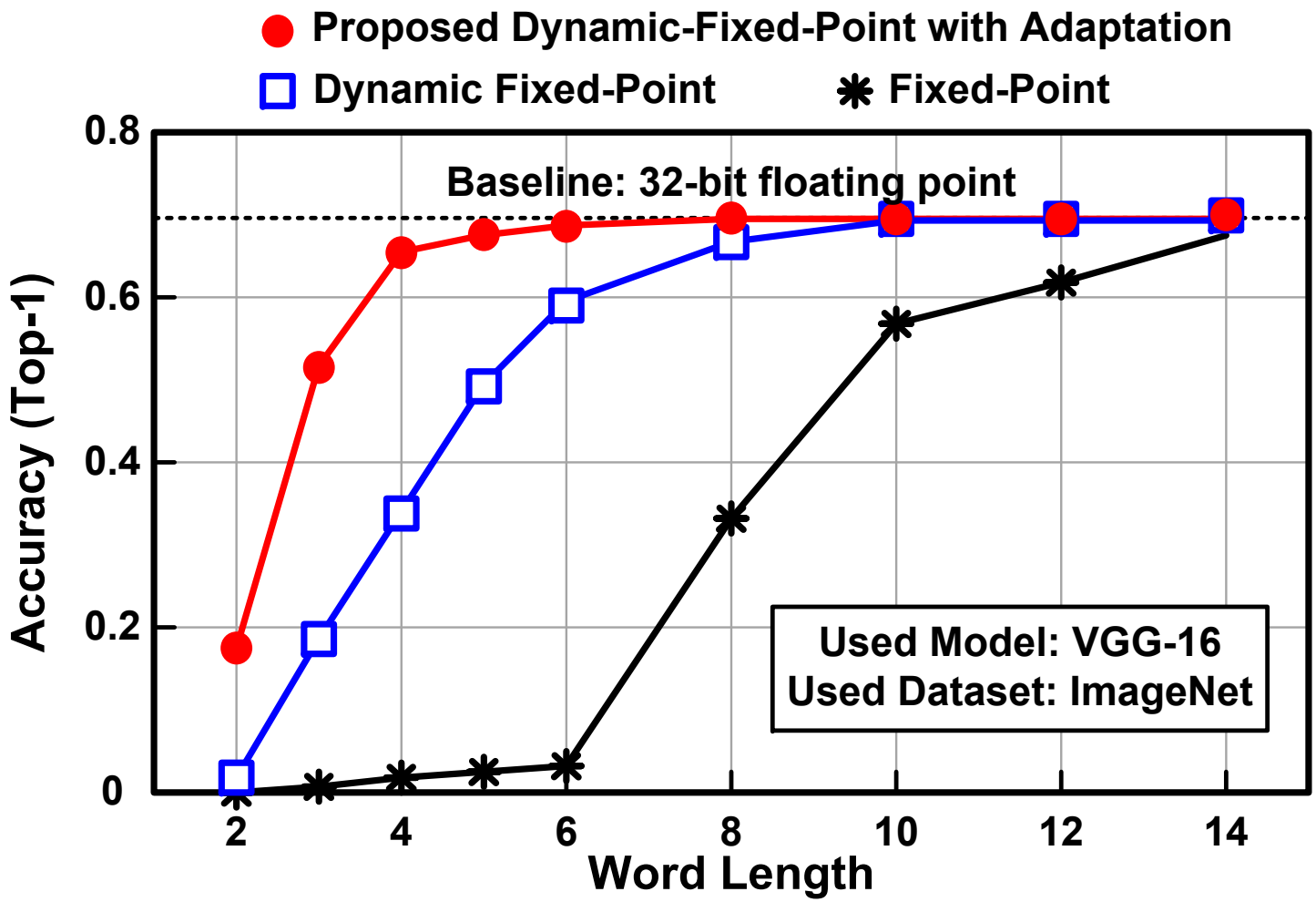
– **On-line (on-chip)** learning-based FL selection

– FL is ***dynamically fit*** to ***current input image***

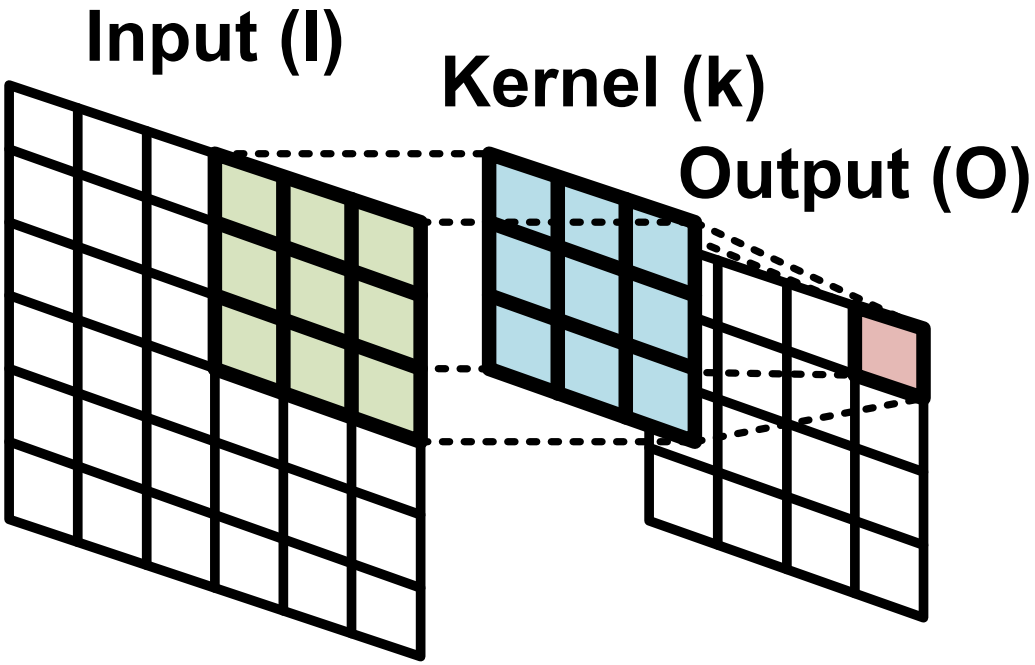
→ **No off-chip learning, lower required WL**

Performance Comparisons

- Image classification results



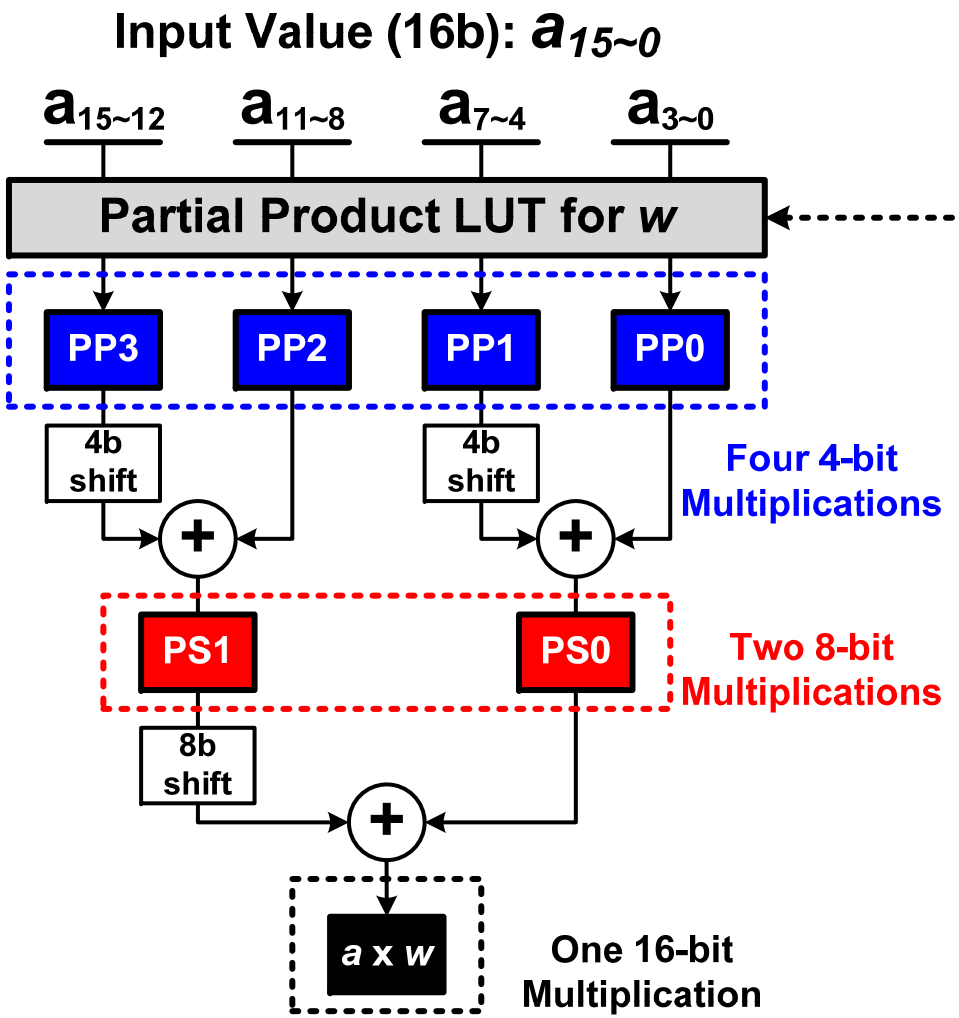
LUT-based Reconfigurable Multiplier



$$O(a,b) = \sum_{i=0}^{i=2} \sum_{j=0}^{j=2} \{I(a+i-1, b+j-1) \times \boldsymbol{k(i,j)}\}$$

50,176 times multiplications for each kernel weight @ Input image(I): 224 X 224, Kernel(k): 3 X 3

LUT-based Reconfigurable Multiplier



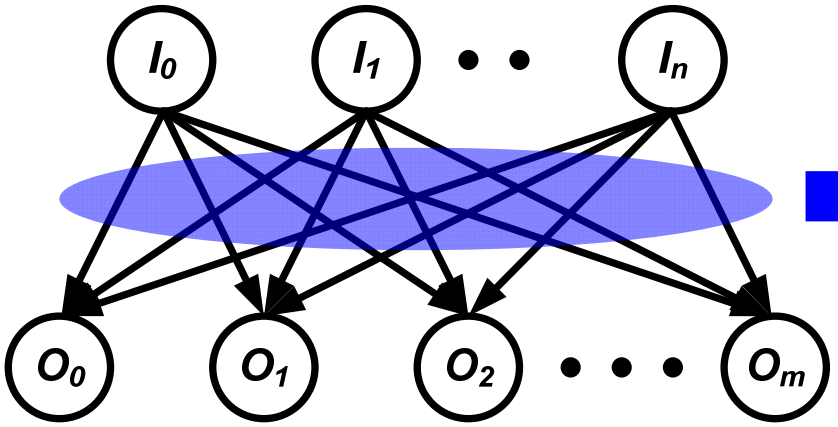
Physical LUT

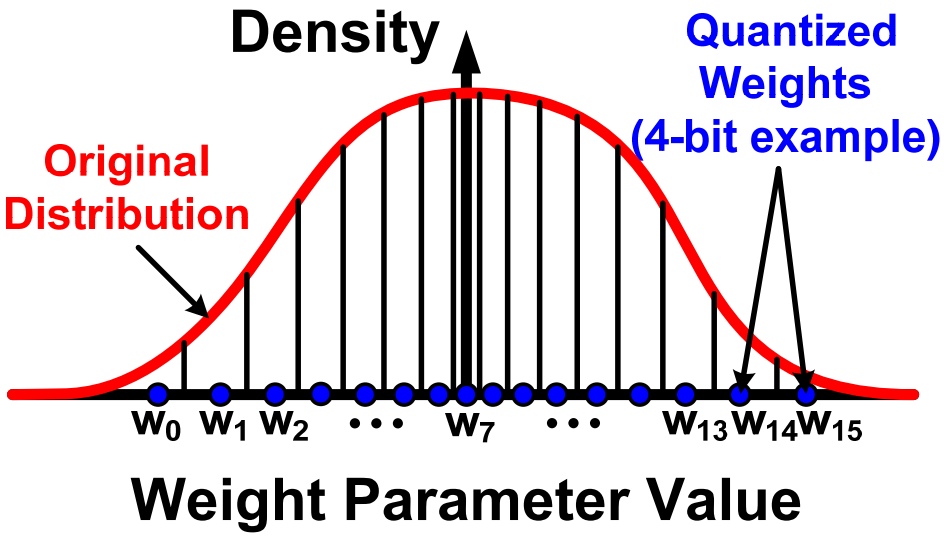
Part	0001	0011	0101	0111
Partial Product	w	3w	5w	7w
Part	1001	1011	1101	1111
Partial Product	9w	11w	13w	15w

Logical LUT

Part	0000	0010	0100	1000
Partial Product	0	w << 1	w << 2	w << 3
Part	0110	1100	1010	1110
Partial Product	3w << 1	3w << 2	5w << 1	7w << 1

Weight Quantization





FC Layer Weight Quantization

	Top-1 Error	Top-5 Error
32bits	42.78%	19.73%
4bits	42.79%	19.73%
2bits	44.77%	22.33%

ImageNet Classification Test

LSTM Layer Weight Quantization

	Perplexity (Lower is beter)	BLEU (Higher is better)
32bits	15.680	55.7 / 37.4 / 24 / 15.7
4bits	15.829	56.7 / 38 / 24.4 / 15.7
2bits	19.298	58.4 / 38.6 / 24 / 14.8

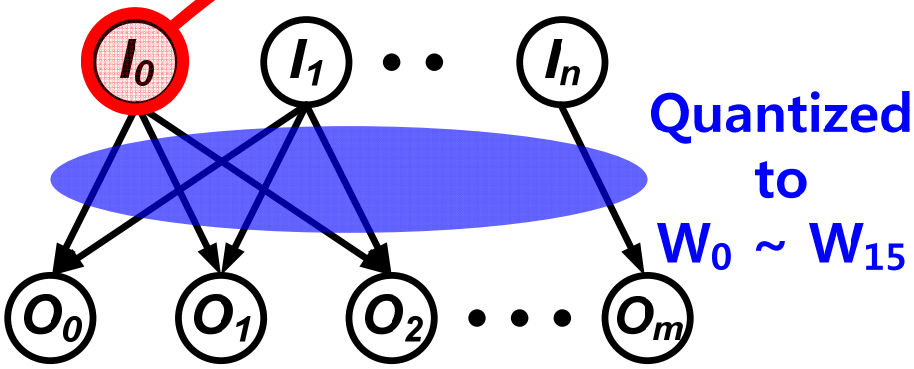
Flickr 8K Image Captioning Test

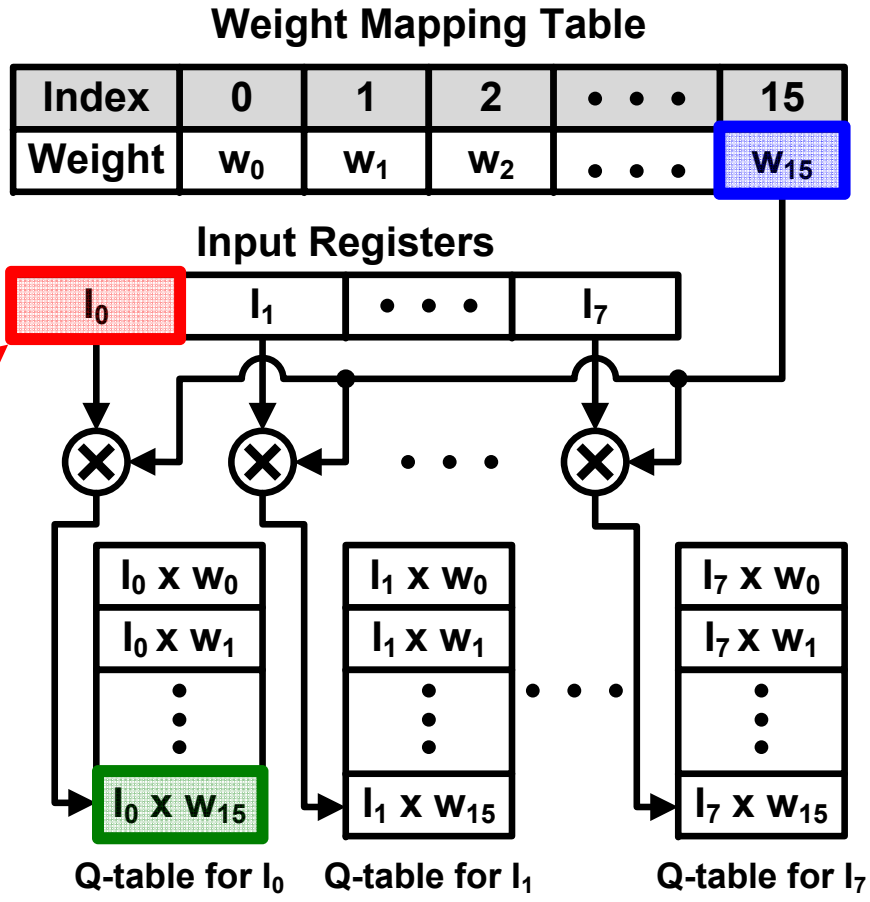
Quantization Table Construction

- **Pre-computation with each quantized weight**

Multiplications between *input* and quantized *weights*

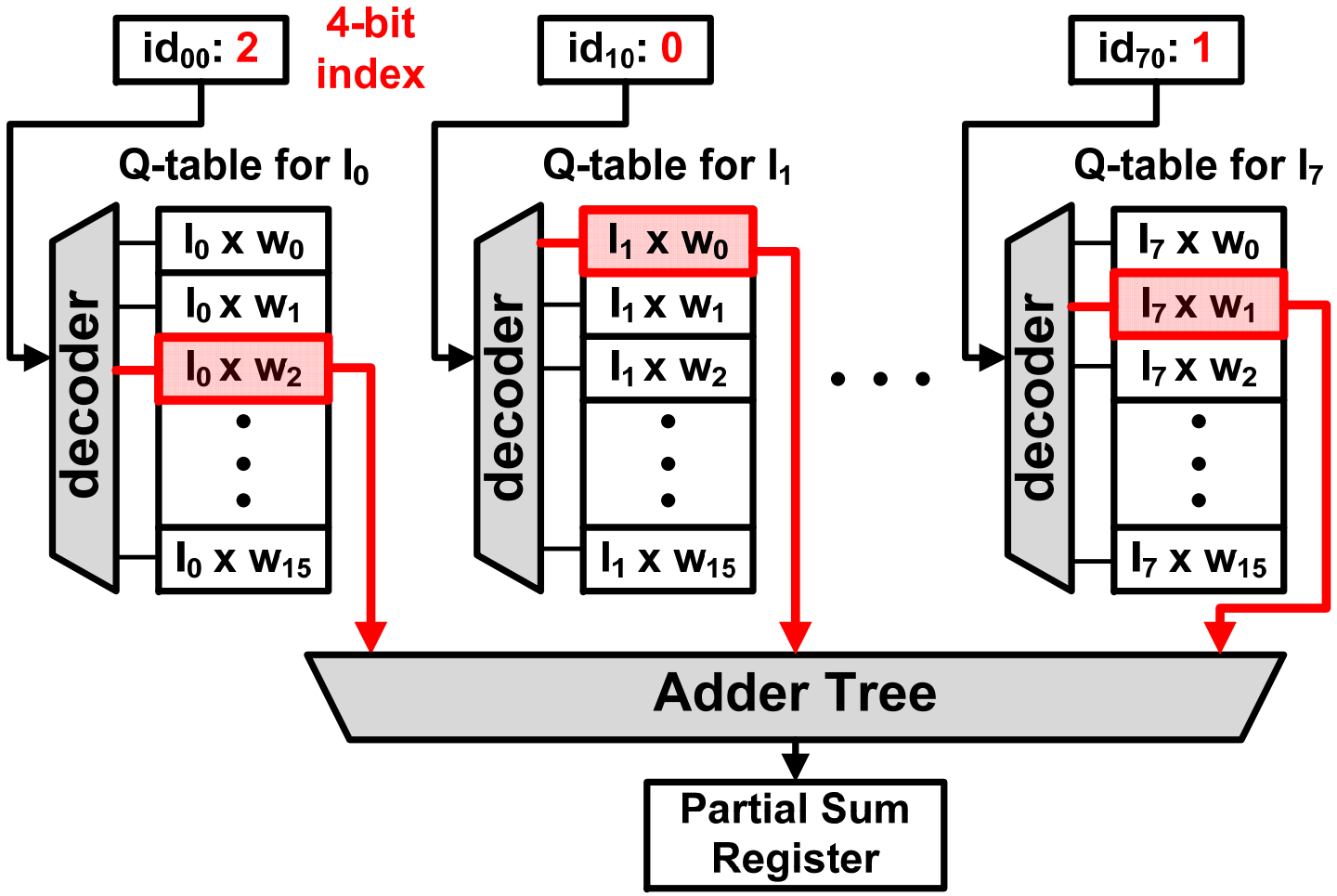
Multiplication results are also quantized to *16 values.*





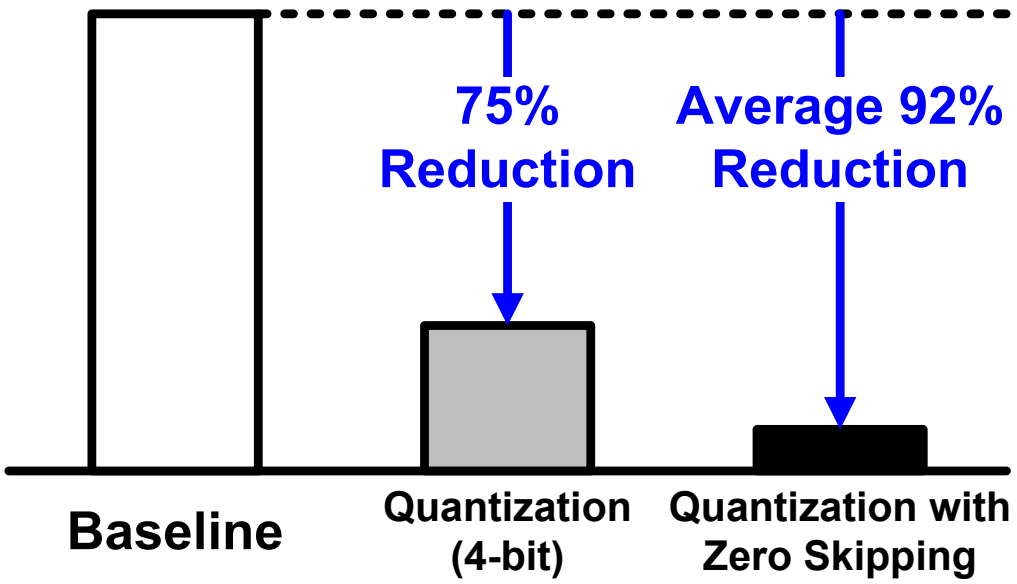
Multiplication with Quantization Table

- Decode index to load the pre-computed result



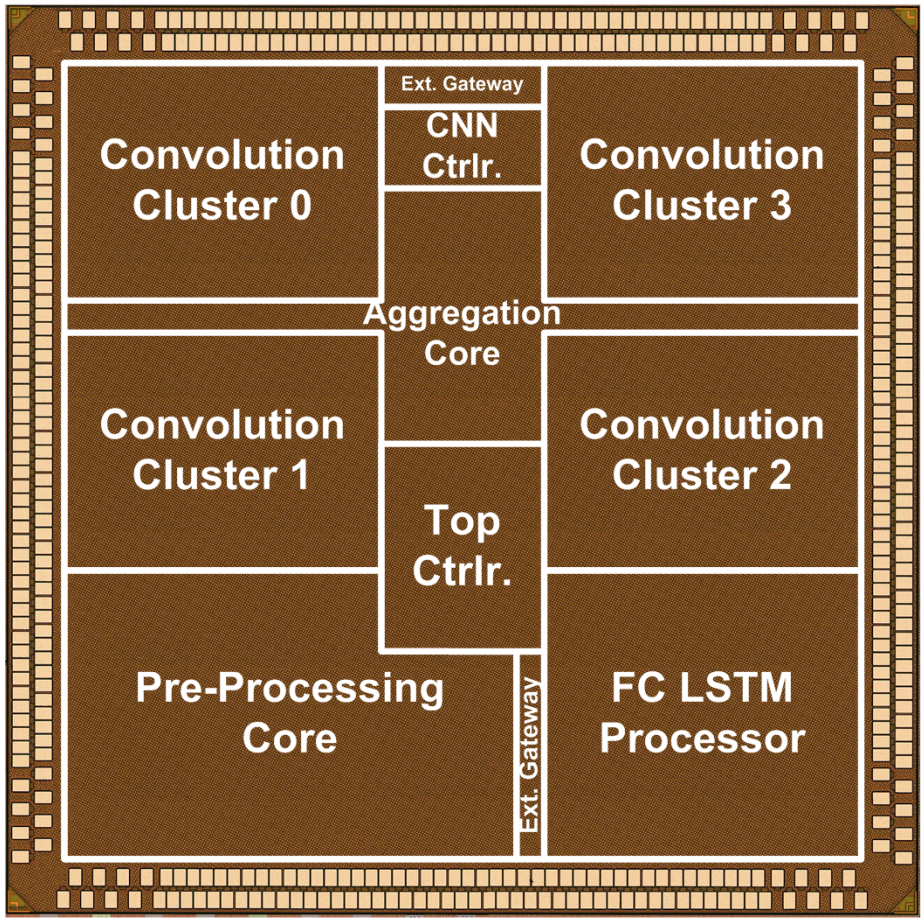
Quantization Table-based Multiplier

- **Off-chip memory BW**
  - Quantization: 16-bit weight → 4-bit index
  - Zero skipping: Skip weight load for zero input



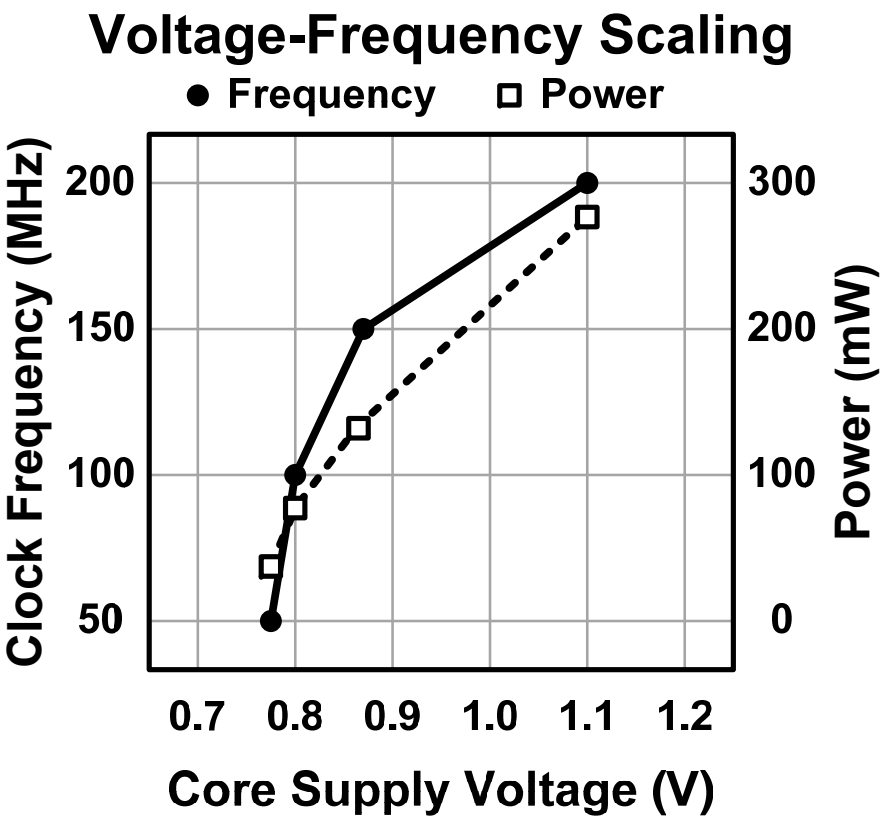
- **Multiplications**
  - 99% of multiplications → Table look-up

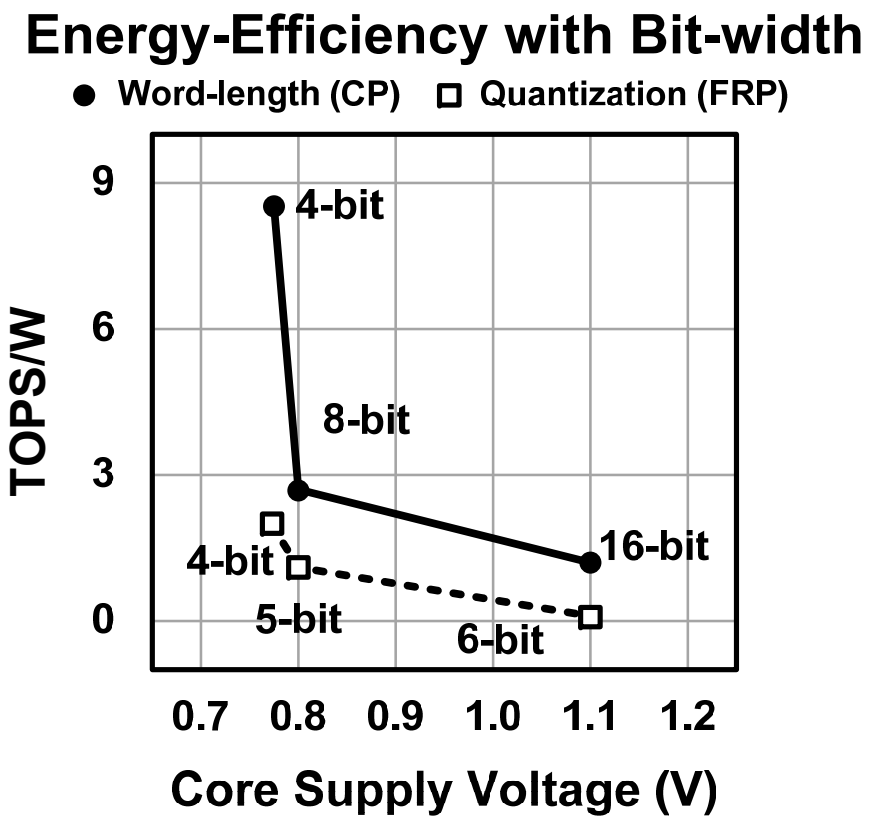
Chip Photograph and Summary



	Specifications		
Process	65nm 1P8M CMOS		
Die Area	4mm x 4mm (16mm^2)		
SRAM	CLP	FCRLP	
	280 KB	10 KB	
Supply	0.77V ~ 1.1V		
Frequency	50 ~ 200MHz		
Power	50MHz @ 0.77V	34.6mW	
	200MHz @ 1.1V	279mW	
Energy Efficiency		CLP Word Length: 16b	CLP Word Length: 4b
	50MHz @ 0.77V	2.1 TOPS/W	8.1 TOPS/W
	200MHz @ 1.1V	1.0 TOPS/W	3.9 TOPS/W

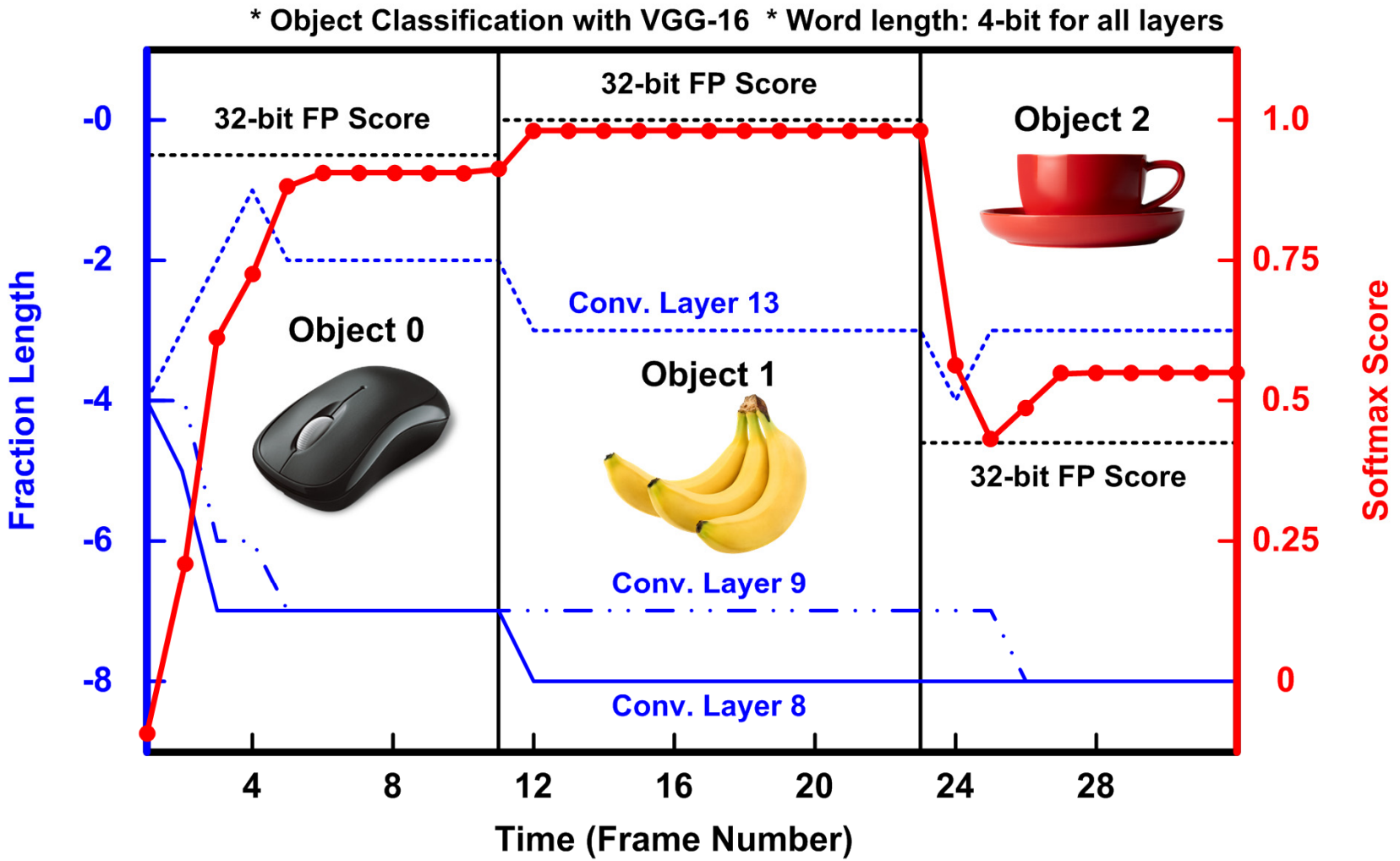
Measurement Results



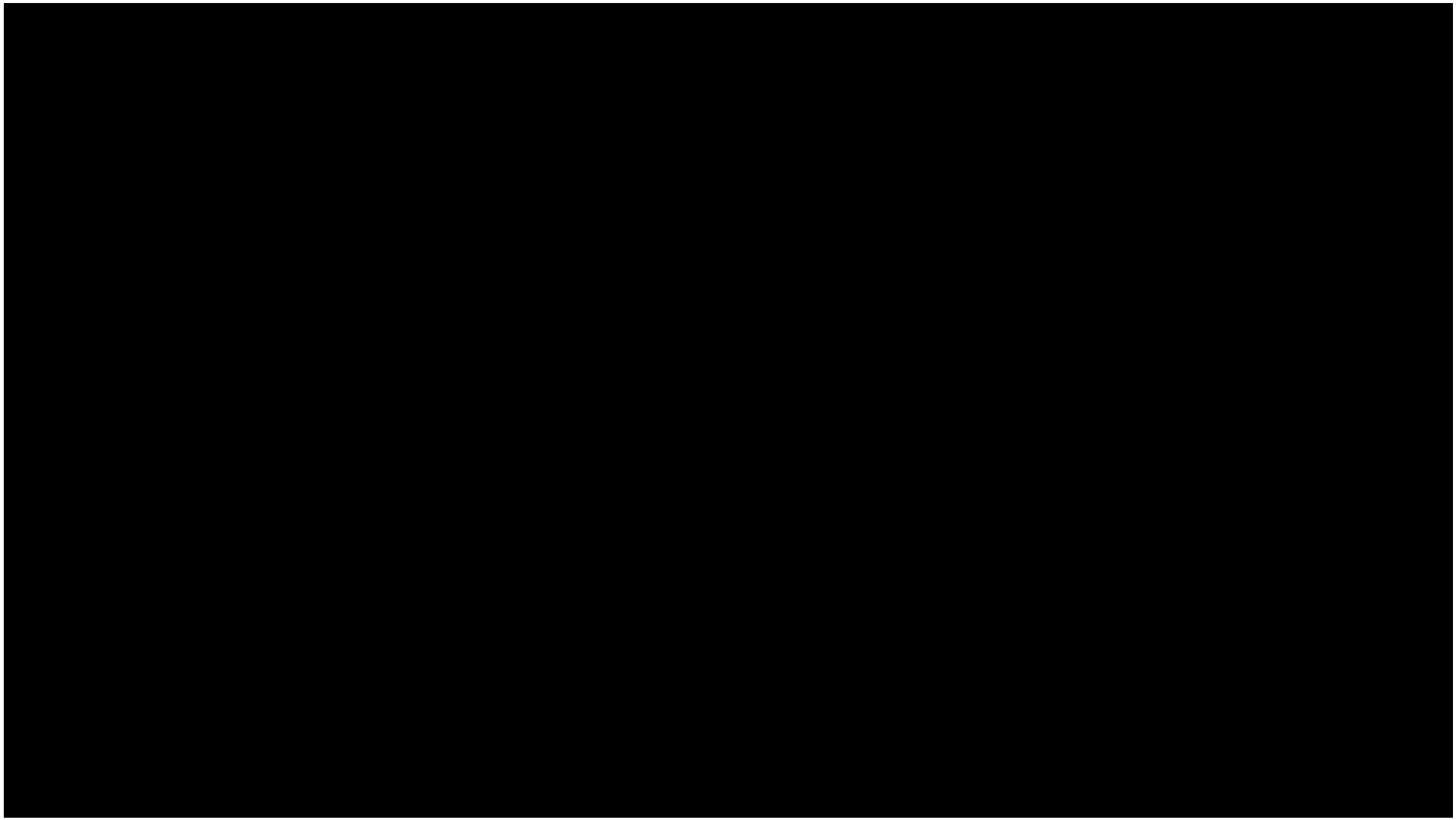


Measurement Results

- **Dynamic fixed-point with on-line adaptation**



Demonstration Video



Performance Comparison

- **Comparison with convolution processors**

	General					Convolution Layer Performance		
						AlexNet		
	Tech. (nm)	Clock Freq. (MHz)	Supply Voltage (V)	CNN Support	RNN Support	Power (mW)	Energy Efficiency	Frame Rate
ISSCC 2016 [1]	65	100 – 250	0.82 – 1.17	o	x	278 (200MHz)	0.21 TOPS/W	34.7 fps
SoVC 2016 [2]	40	12 – 204	0.55 – 1.1	o	x	76 (~100MHz)	0.94 TOPS/W	47 fps
Proposed	65	50 – 200	0.77 – 1.1	o	o	63 (100MHz)	4.2 TOPS/W	177 fps

[1] Y. Chen, et al., ISSCC 2016 [2] B. Moons, et al., SOVC 2016

Performance Comparison

- **Comparison with FC layer processor**

	General					Fully-connected, LSTM Layer Performance			
	Tech. (nm)	Clock Freq. (MHz)	Supply Voltage (V)	CNN Support	RNN Support	Power (mW)	Energy Efficiency (TOPS/W)	AlexNet	
								Power (mW)	Frame Rate
ISCA 2016 [4]	45	800	100 – 250	x	o	590	0.17	600	18.8k fps
Proposed	65	50 – 200	0.77 – 1.1	o	o	21	1.1	3.5	1.2k fps

→ **Proposed work is the only one which can support both CNN and RNN**

[4] S. Han, et al., ISCA 2016

Conclusion

- An Energy Efficient CNN-RNN Processor SoC is proposed for General-purpose DNNs
- Key Features:
  - Support both CNN and RNN
  - Mixed workload division method
  - Dynamic fixed-point with on-line adaptation
  - Quantization table-based multiplier

DNPU: An 8.1TOPS/W reconfigurable CNN-RNN Processor SoC