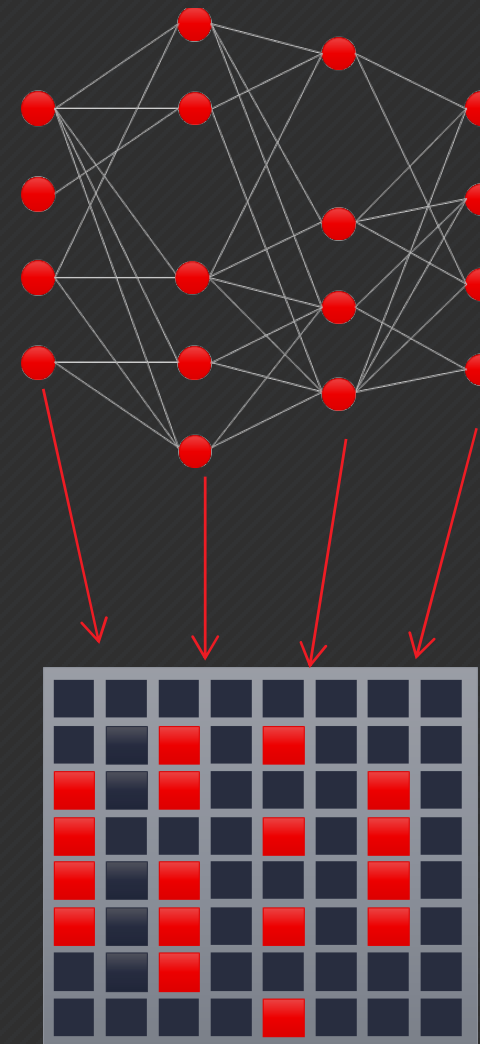




# Co-Design for Efficient & Adaptive ML

*Fast Machine Learning for Science, 2025-09-04*

Dr. Yaman Umuroğlu  
Principal Member of Technical Staff  
AMD Research & Advanced Development (RAD)



# AMD · RAD Integrated Communications & AI Lab (RADICAL)

## Team

- ~25 top-class researchers/PhDs
- Europe, Singapore, US
- With broad expertise in AI, compilers, compute architectures, and networking
- Highly active internship program ~10 across the different sites

## Subjects

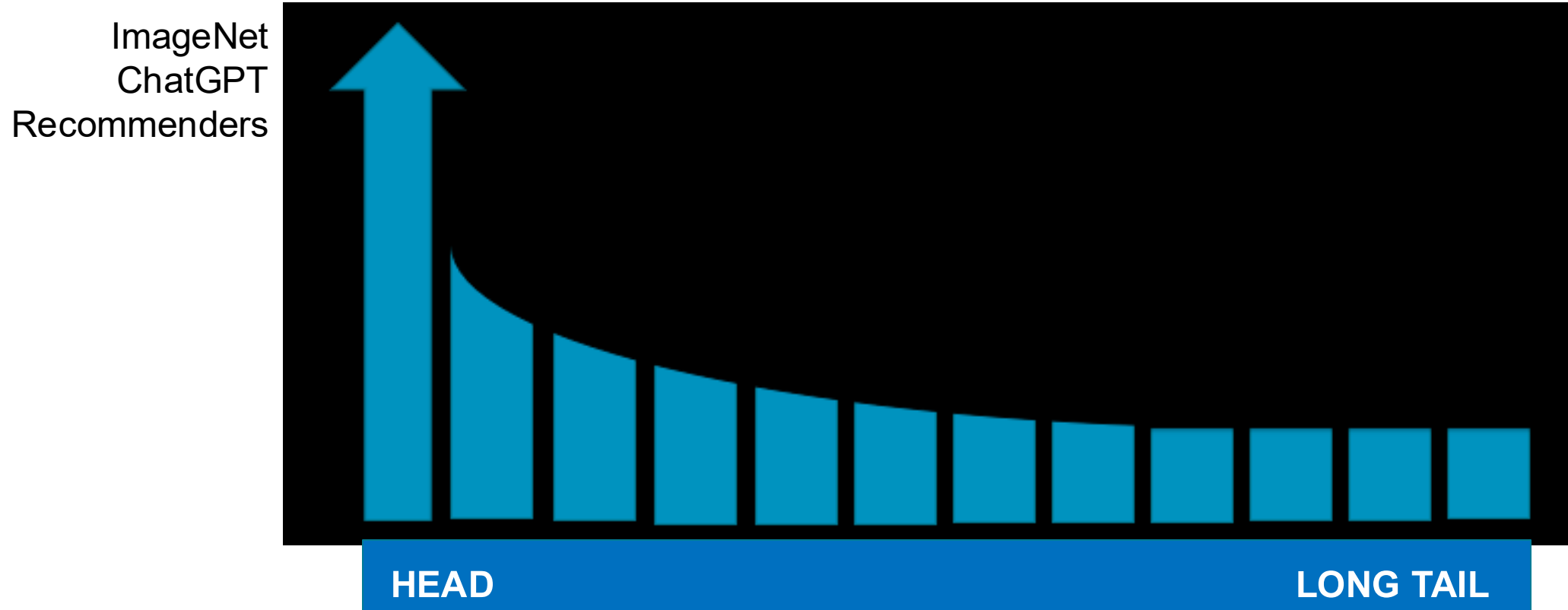
- Customized compute architectures for inference acceleration
- Model optimization
- Networking for distributed AI

## Close Collaboration

- In close collaboration with universities, partners and customers

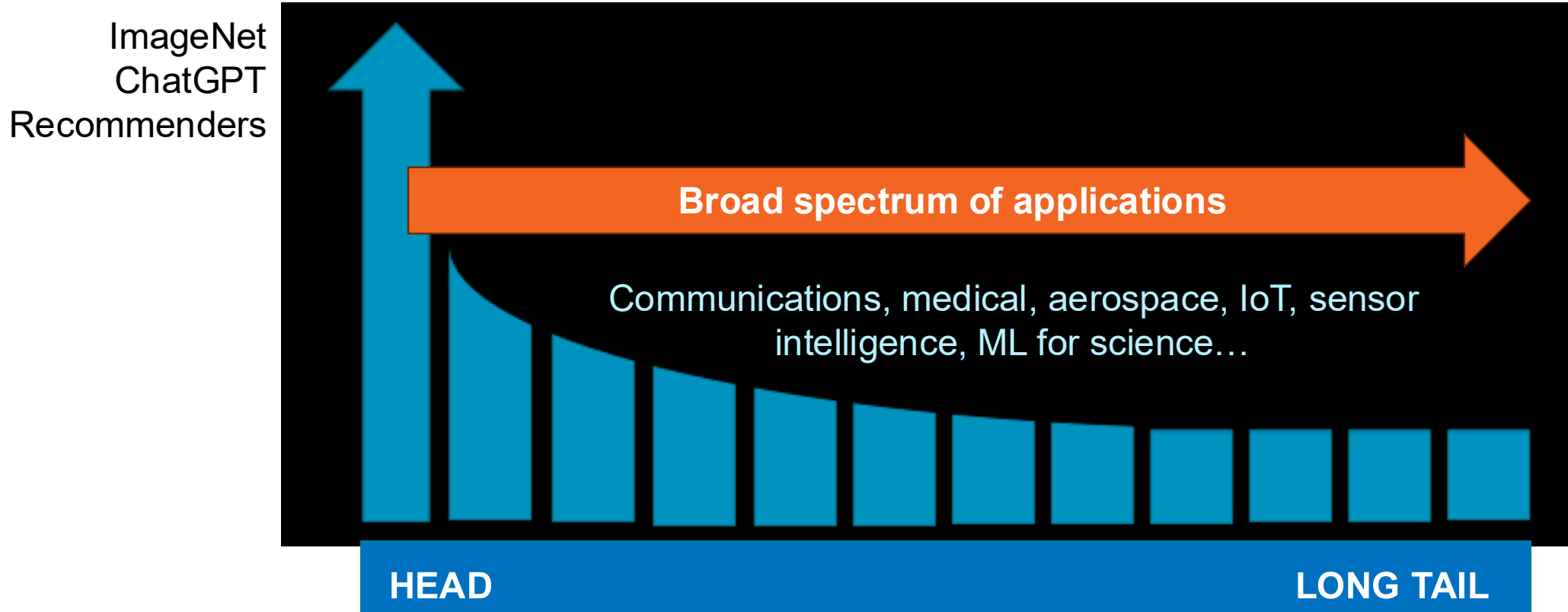


# Pervasive AI

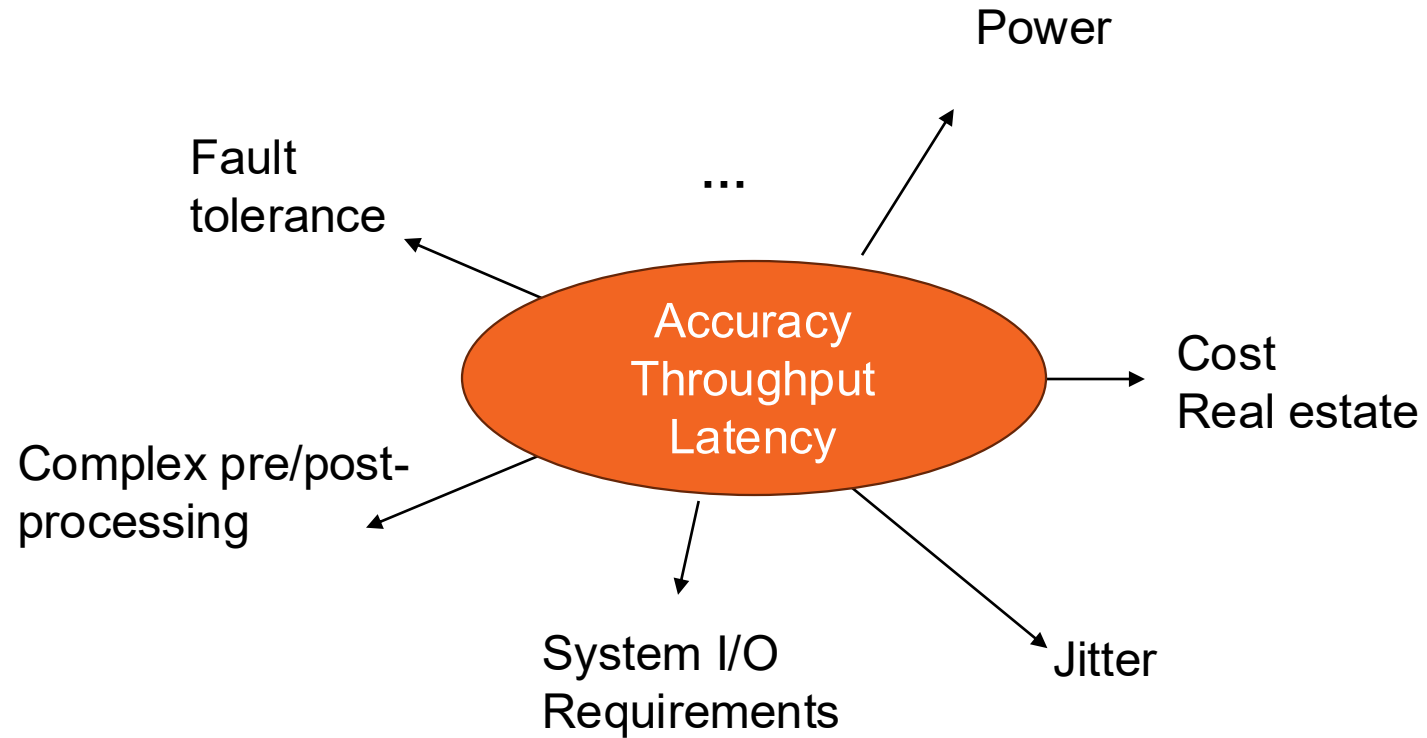


Adapted from TED Talk: Andrew Ng "How AI could empower any business"

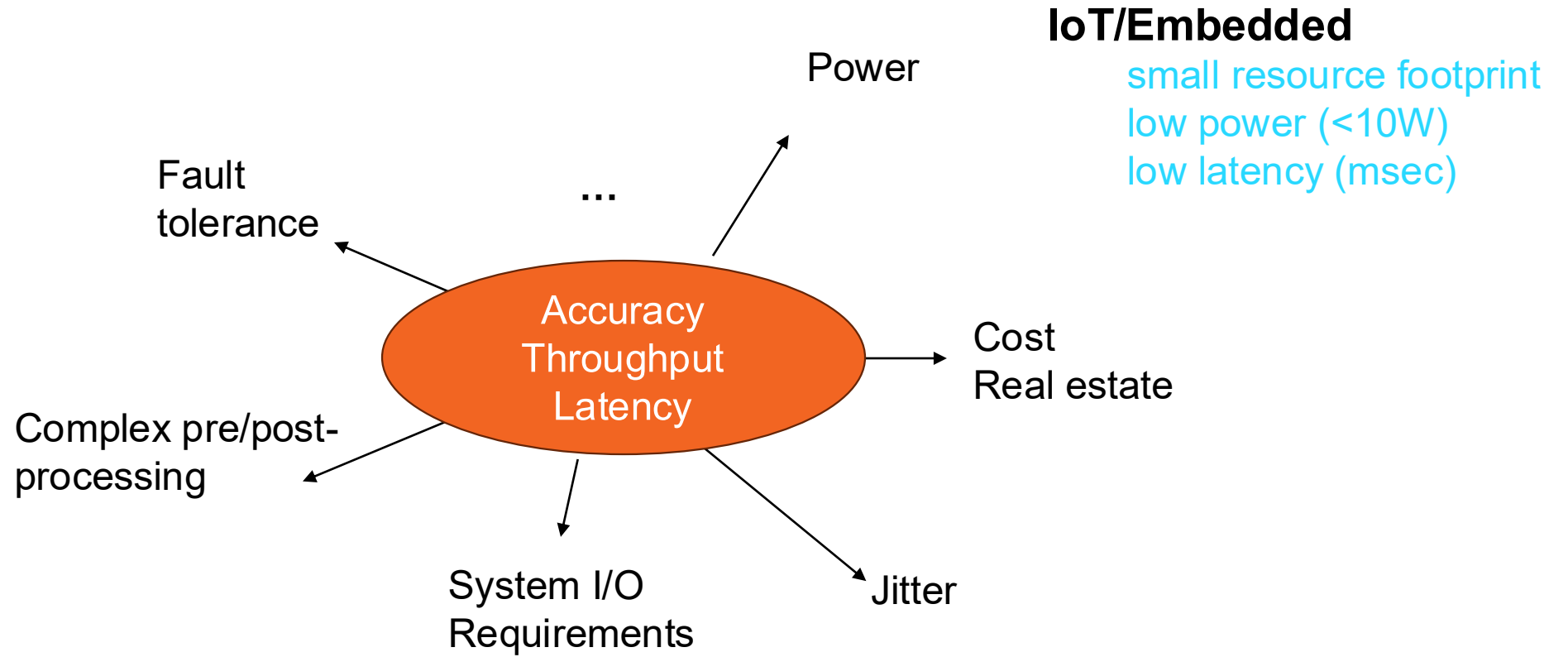
# Pervasive AI



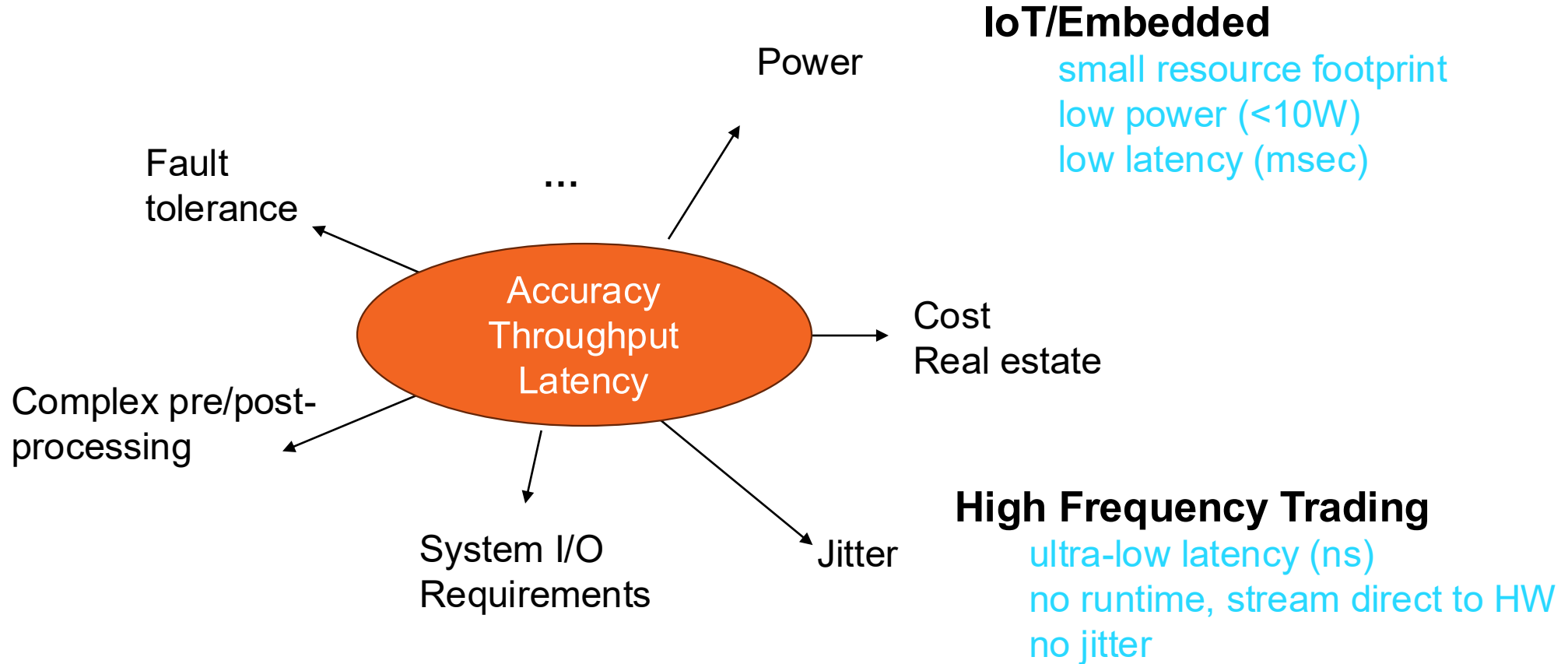
# Pervasive AI: Diverse Requirements for Inference



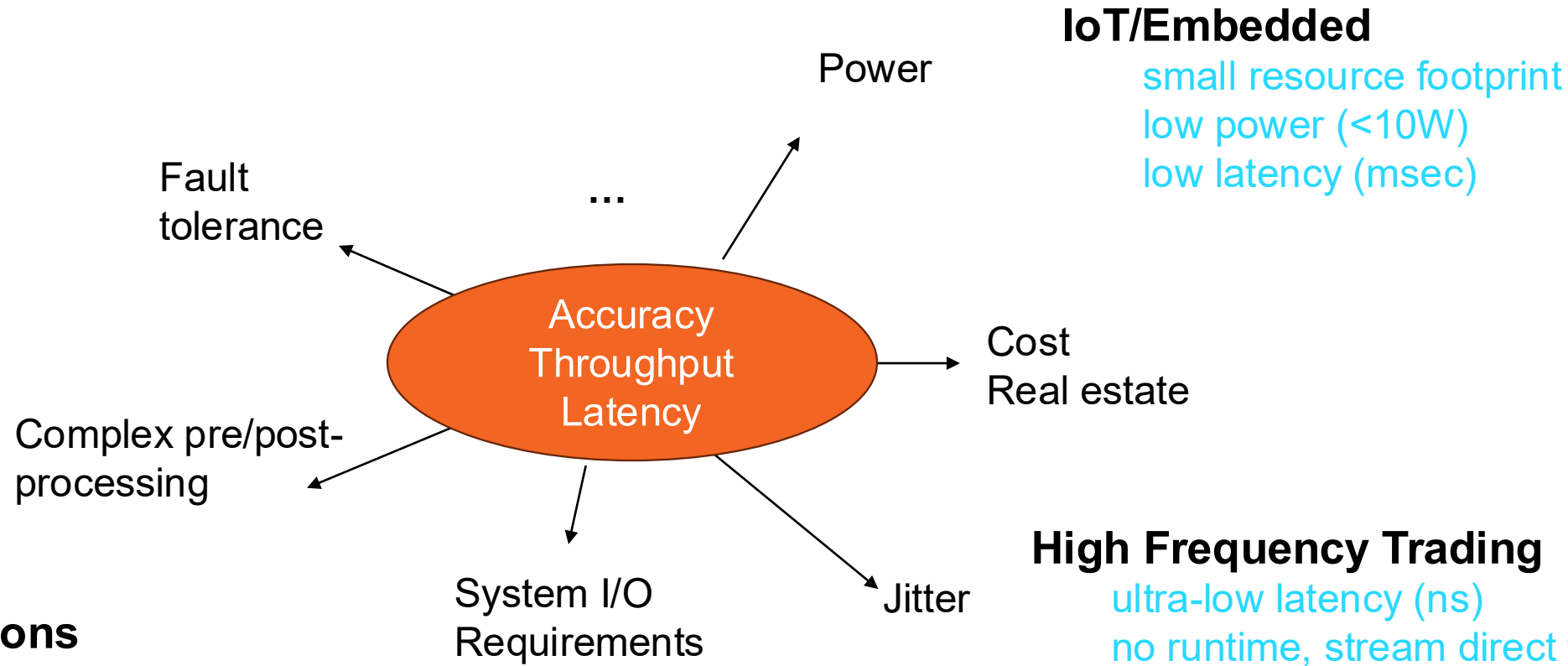
# Pervasive AI: Diverse Requirements for Inference



# Pervasive AI: Diverse Requirements for Inference



# Pervasive AI: Diverse Requirements for Inference



## IoT/Embedded

small resource footprint  
low power (<10W)  
low latency (msec)

## High Frequency Trading

ultra-low latency (ns)  
no runtime, stream direct to HW  
no jitter

## ML in Communications

very high throughput (100M/s)  
no run-time, stream direct to HW  
low latency (sub-msec)  
combine with signal processing

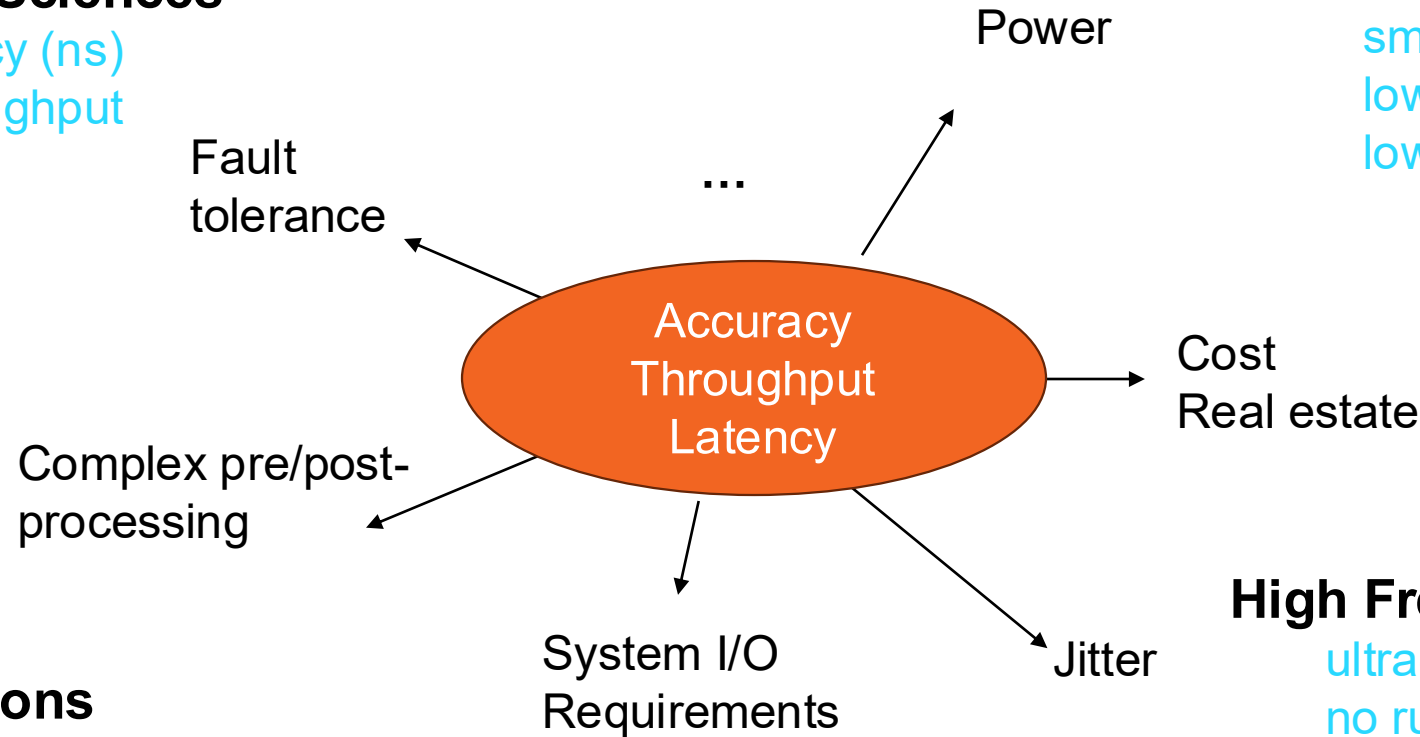
# Pervasive AI: Diverse Requirements for Inference

## ML for Physical Sciences

ultra-low latency (ns)  
ultra-high throughput  
fault tolerance

## IoT/Embedded

small resource footprint  
low power (<10W)  
low latency (msec)



## High Frequency Trading

ultra-low latency (ns)  
no runtime, stream direct to HW  
no jitter

## ML in Communications

very high throughput (100M/s)  
no run-time, stream direct to HW  
low latency (sub-msec)  
combine with signal processing

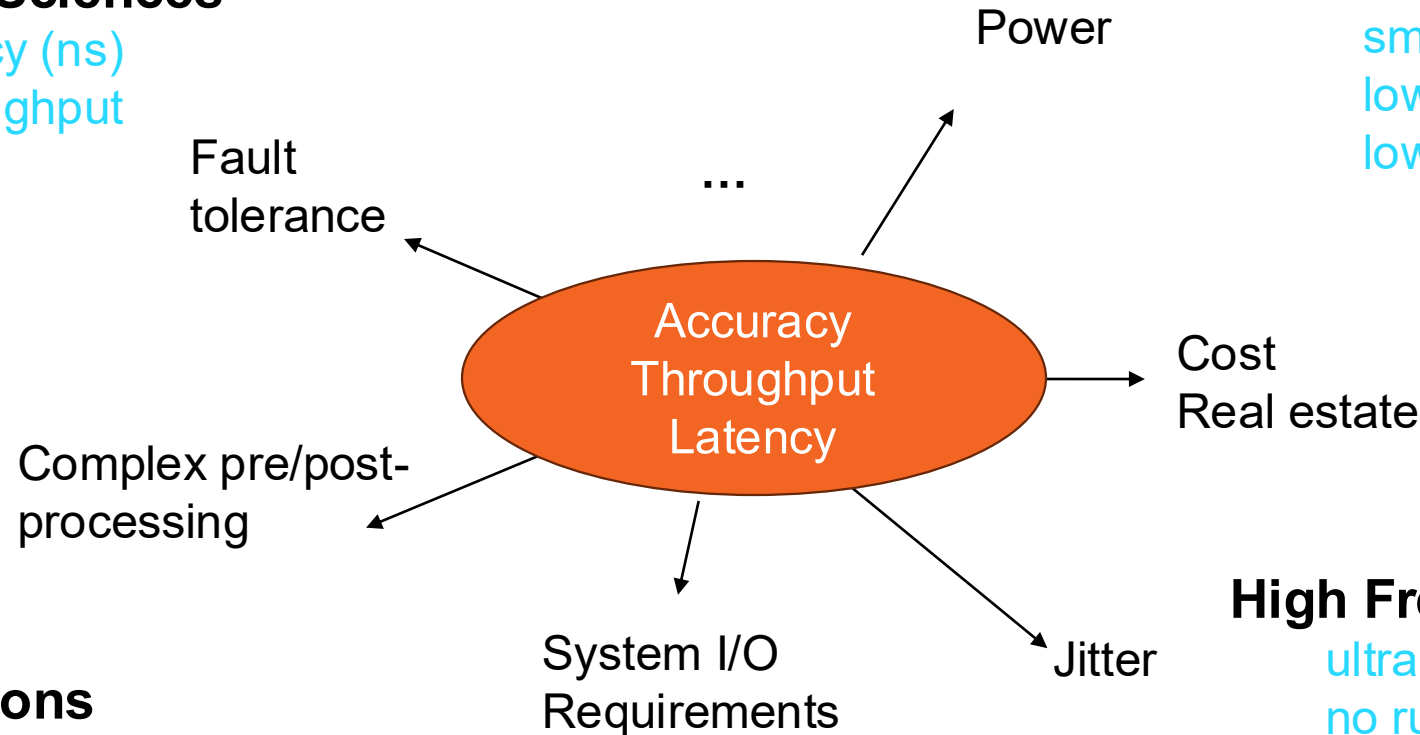
# Pervasive AI: Diverse Requirements for Inference

## ML for Physical Sciences

ultra-low latency (ns)  
ultra-high throughput  
fault tolerance

## IoT/Embedded

small resource footprint  
low power (<10W)  
low latency (msec)



## High Frequency Trading

ultra-low latency (ns)  
no runtime, stream direct to HW  
no jitter

## ML in Communications

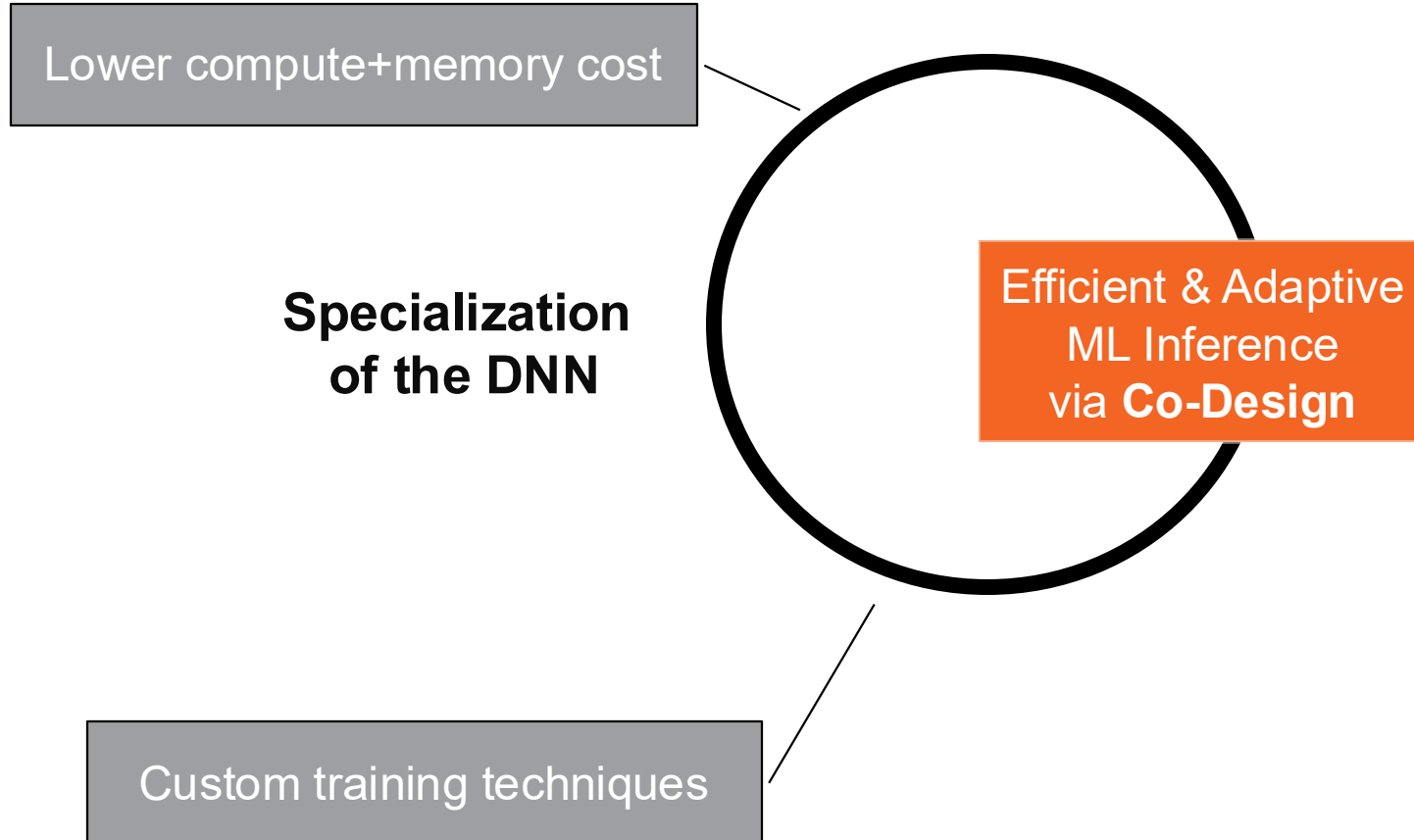
very high throughput (100M/s)  
no run-time, stream direct to HW  
low latency (sub-msec)  
combine with signal processing

Everything in flux (MLPs -> CNNs -> Transformers...)  
Pervasive AI needs **efficient** and **adaptive** solutions

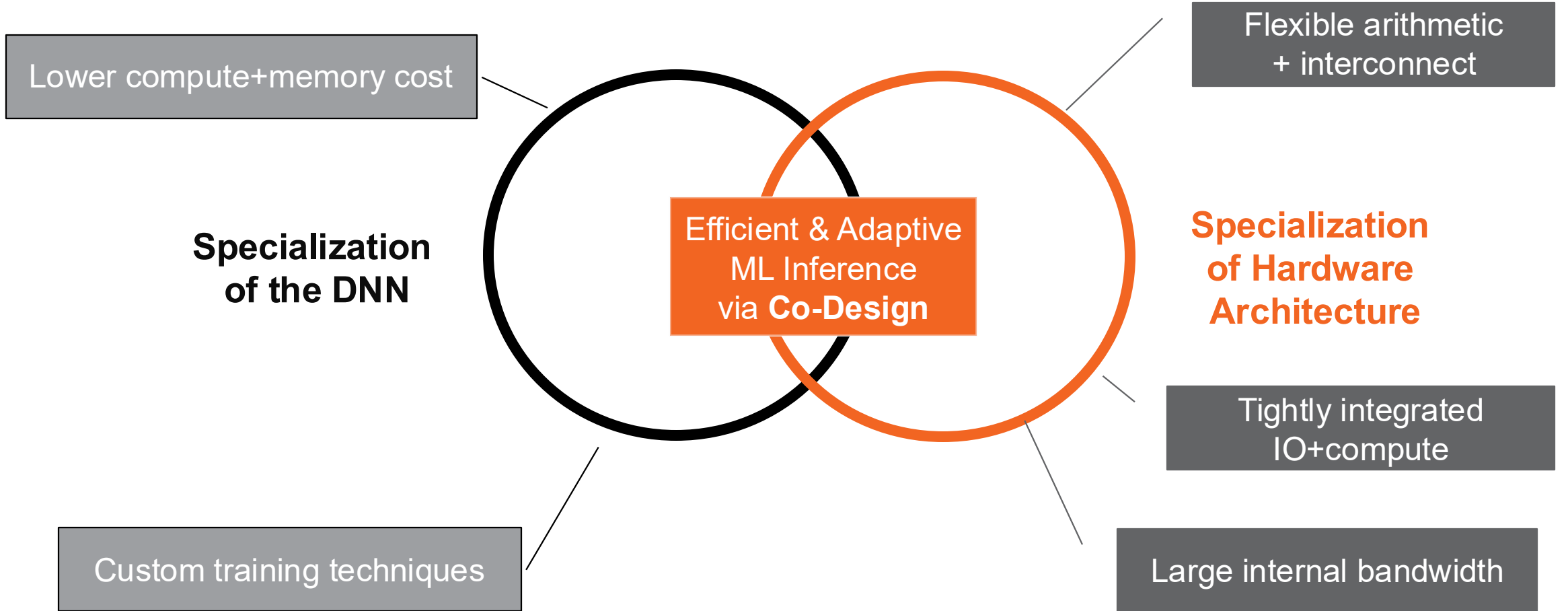
# Specialization is essential

Efficient & Adaptive  
ML Inference  
via **Co-Design**

# Specialization is essential

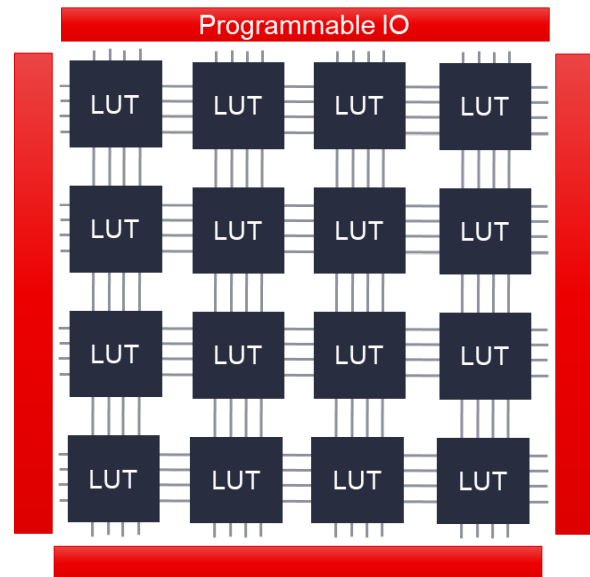


# Specialization is essential



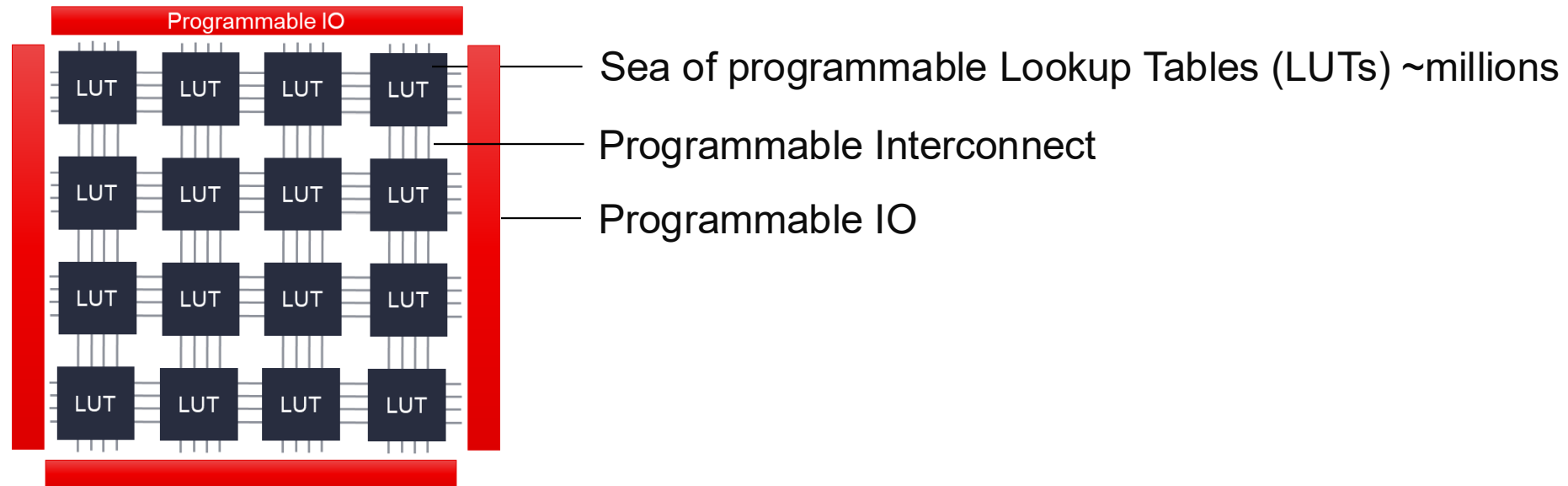
# Field-Programmable Gate Arrays (FPGAs)

- FPGAs: the **chameleon** amongst the semiconductors...
  - Customize IO interfaces
  - Customize functionality
  - Customize compute architectures & memory subsystems to meet performance or efficiency targets
- Flexible, adaptive, mostly homogeneous hardware architecture
  - Enable post-production customization at the architectural level



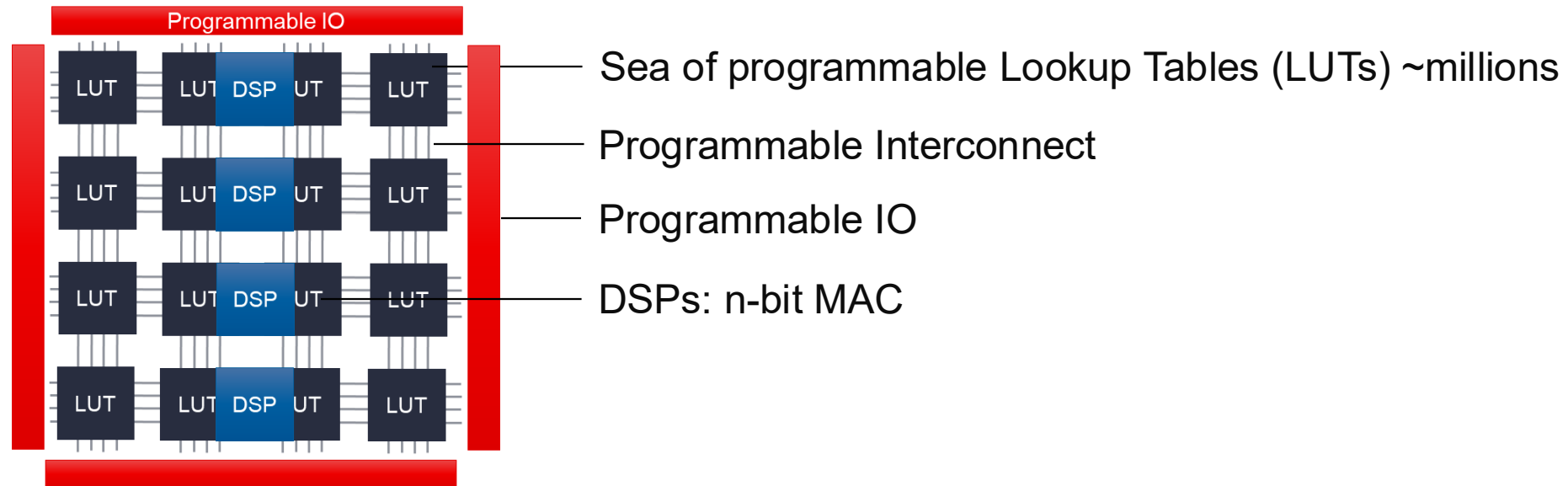
# Field-Programmable Gate Arrays (FPGAs)

- FPGAs: the **chameleon** amongst the semiconductors...
  - Customize IO interfaces
  - Customize functionality
  - Customize compute architectures & memory subsystems to meet performance or efficiency targets
- Flexible, adaptive, mostly homogeneous hardware architecture
  - Enable post-production customization at the architectural level



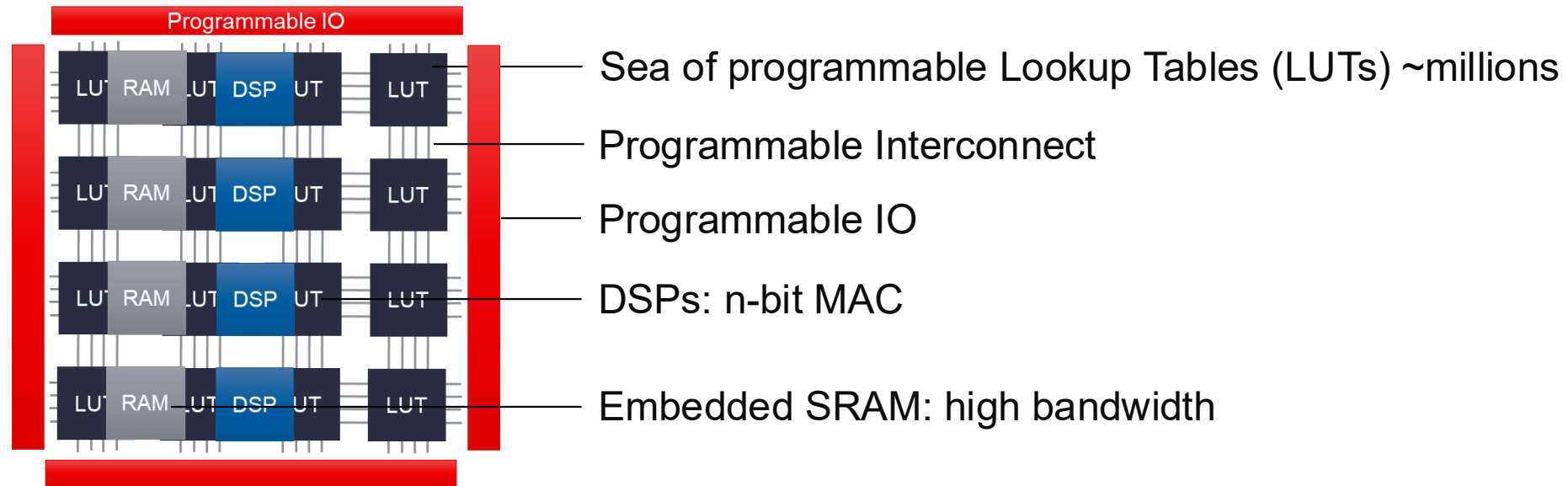
# Field-Programmable Gate Arrays (FPGAs)

- FPGAs: the **chameleon** amongst the semiconductors...
  - Customize IO interfaces
  - Customize functionality
  - Customize compute architectures & memory subsystems to meet performance or efficiency targets
- Flexible, adaptive, mostly homogeneous hardware architecture
  - Enable post-production customization at the architectural level

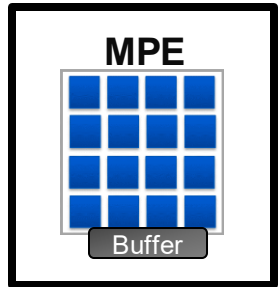
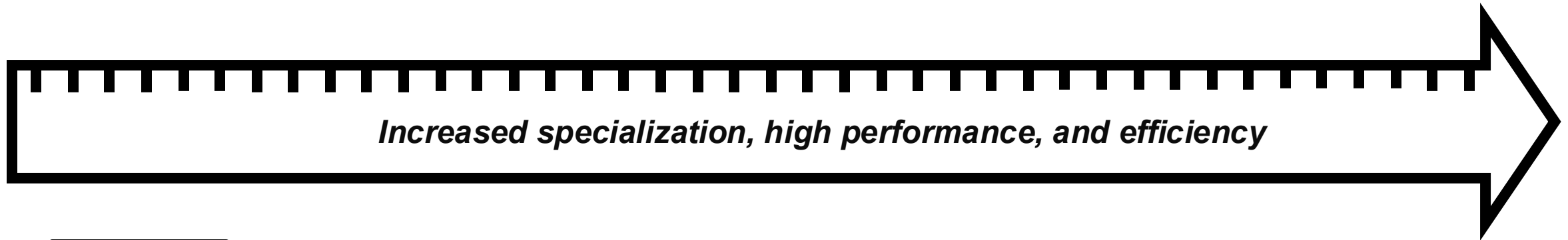


# Field-Programmable Gate Arrays (FPGAs)

- FPGAs: the **chameleon** amongst the semiconductors...
  - Customize IO interfaces
  - Customize functionality
  - Customize compute architectures & memory subsystems to meet performance or efficiency targets
- Flexible, adaptive, mostly homogeneous hardware architecture
  - Enable post-production customization at the architectural level



# Specialized FPGA Inference via Co-Design



Customized for  
ML in general

Customized for  
specific topologies

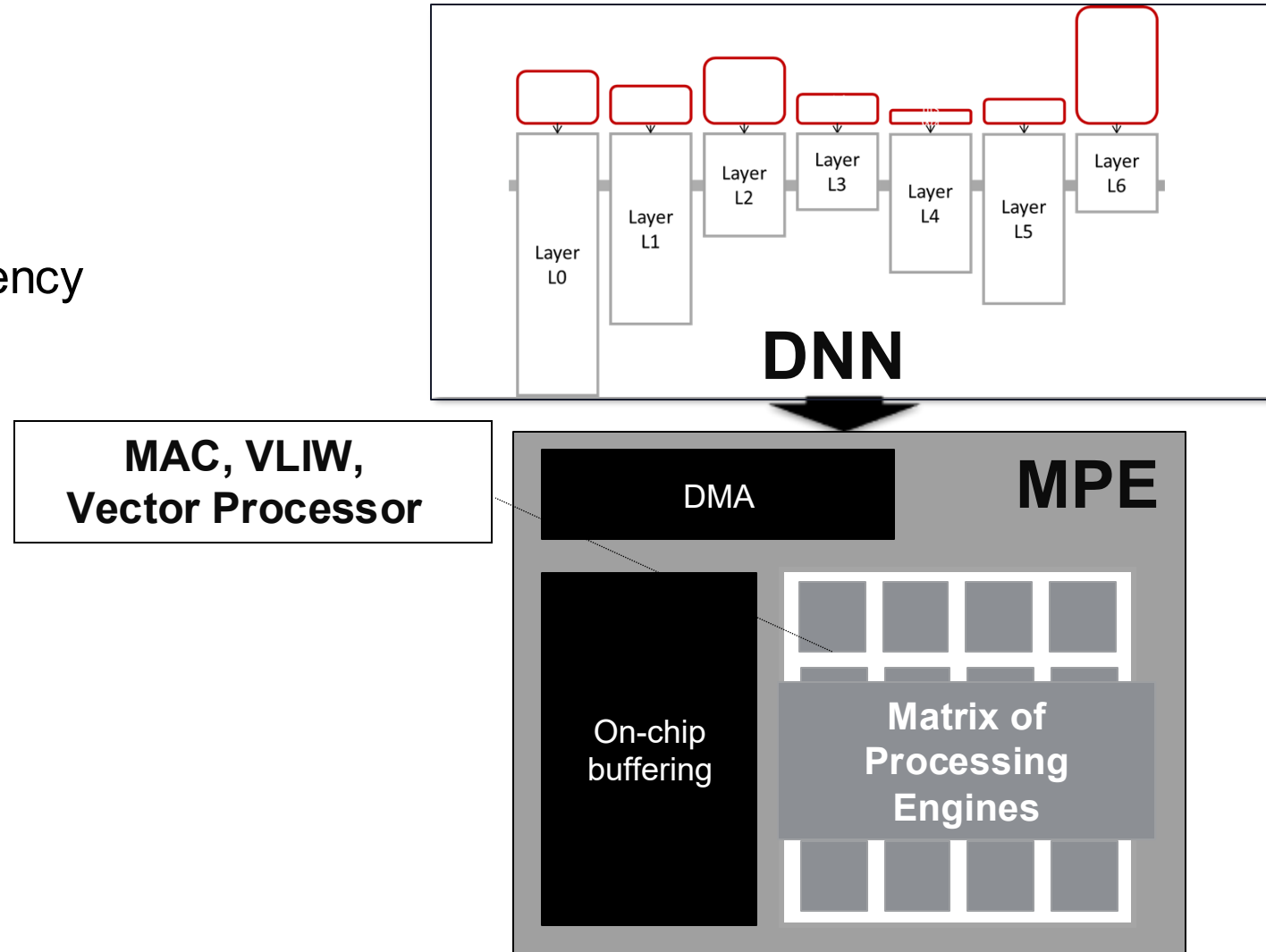
Customized in  
datatypes

Customized in  
connectivity

Ultimate:  
customize the DNN  
to fit the hardware

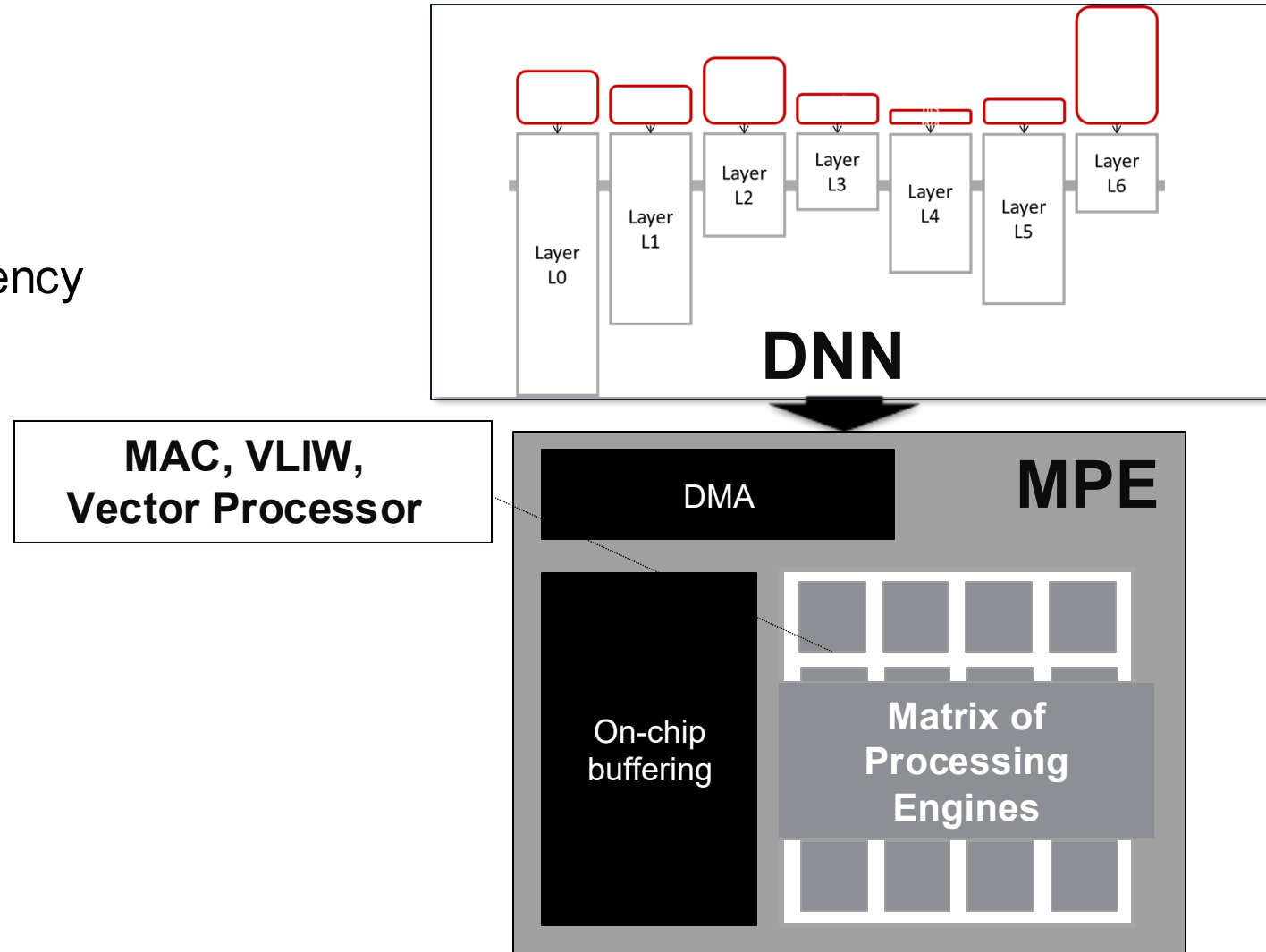
# MPE: Customizing for ML Workloads in General

- MPE: Matrix of Processing Engines
- Popular layer-by-layer compute
- Batching to achieve high compute efficiency
- Customized for “NNs in general”
- Specialized processing engines
  - Operators
  - ALU types
    - tensor-, matrix- or vector-based



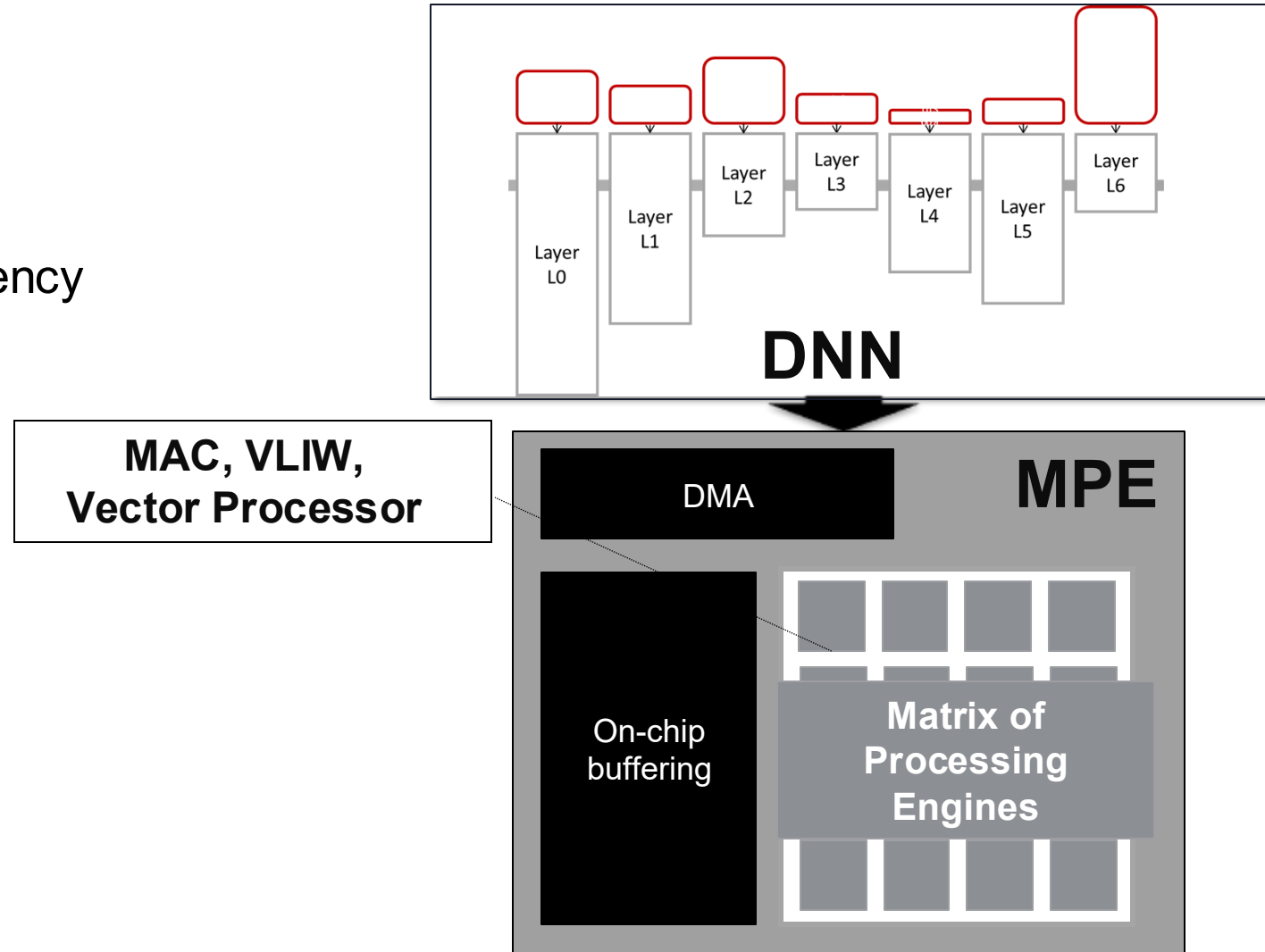
# MPE: Customizing for ML Workloads in General

- MPE: Matrix of Processing Engines
- Popular layer-by-layer compute
- Batching to achieve high compute efficiency
- Customized for “NNs in general”
- Specialized processing engines
  - Operators
  - ALU types
    - tensor-, matrix- or vector-based



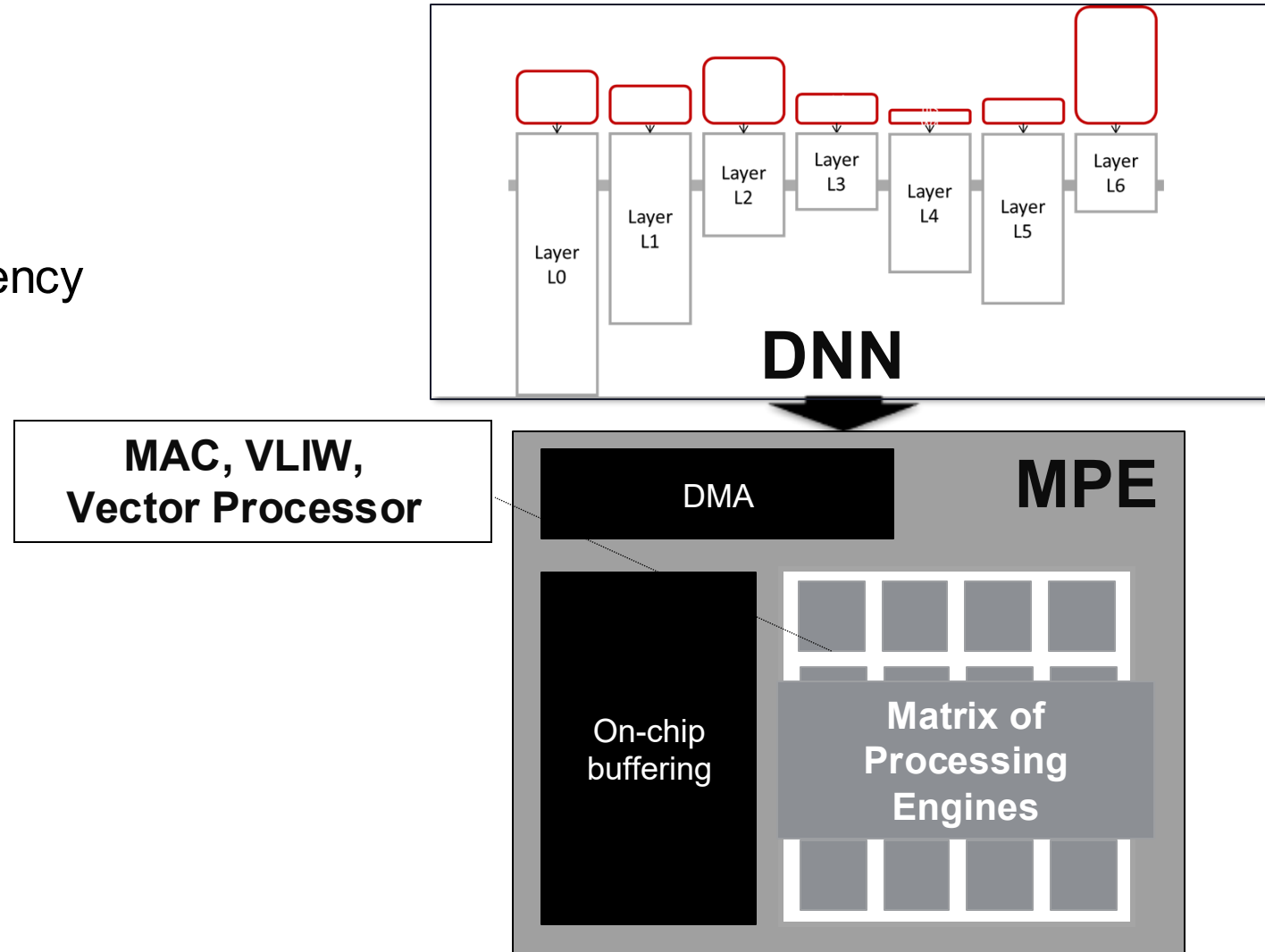
# MPE: Customizing for ML Workloads in General

- MPE: Matrix of Processing Engines
- Popular layer-by-layer compute
- Batching to achieve high compute efficiency
- Customized for “NNs in general”
- Specialized processing engines
  - Operators
  - ALU types
    - tensor-, matrix- or vector-based



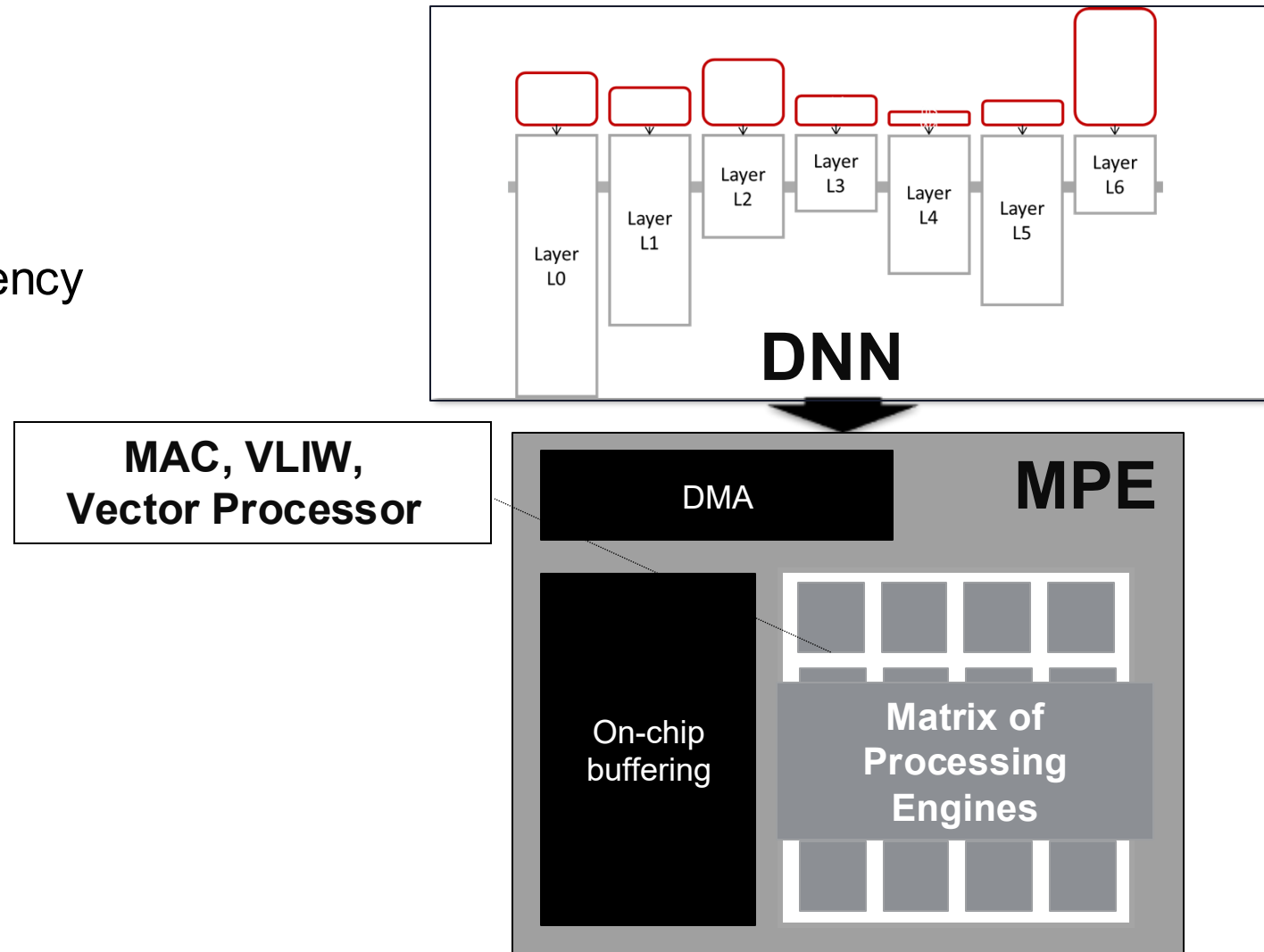
# MPE: Customizing for ML Workloads in General

- MPE: Matrix of Processing Engines
- Popular layer-by-layer compute
- Batching to achieve high compute efficiency
- Customized for “NNs in general”
- Specialized processing engines
  - Operators
  - ALU types
    - tensor-, matrix- or vector-based



# MPE: Customizing for ML Workloads in General

- MPE: Matrix of Processing Engines
- Popular layer-by-layer compute
- Batching to achieve high compute efficiency
- Customized for “NNs in general”
- Specialized processing engines
  - Operators
  - ALU types
    - tensor-, matrix- or vector-based

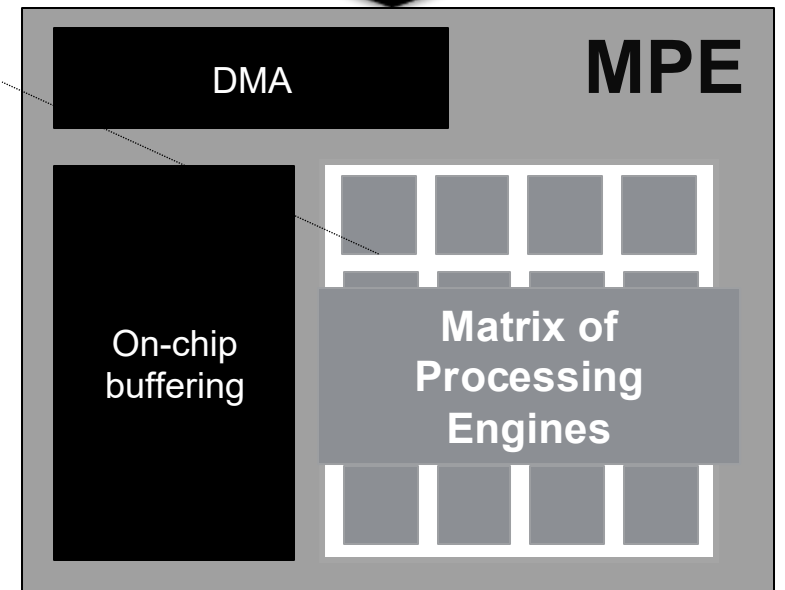
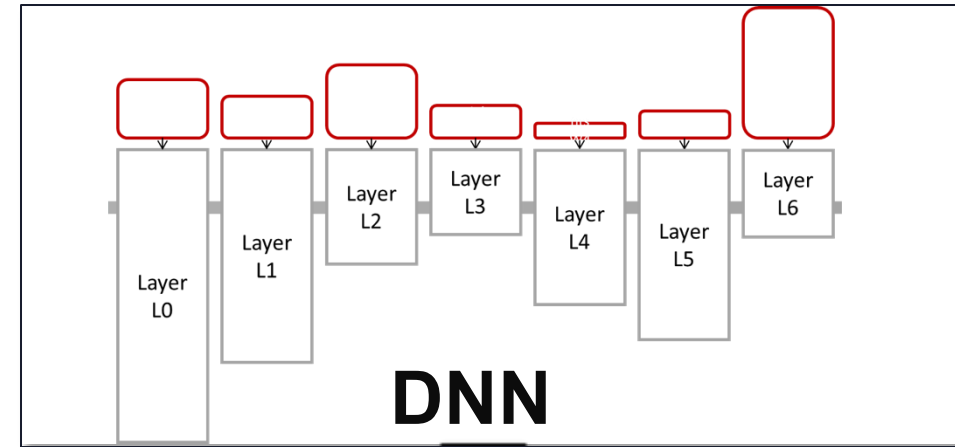


# MPE: Customizing for ML Workloads in General

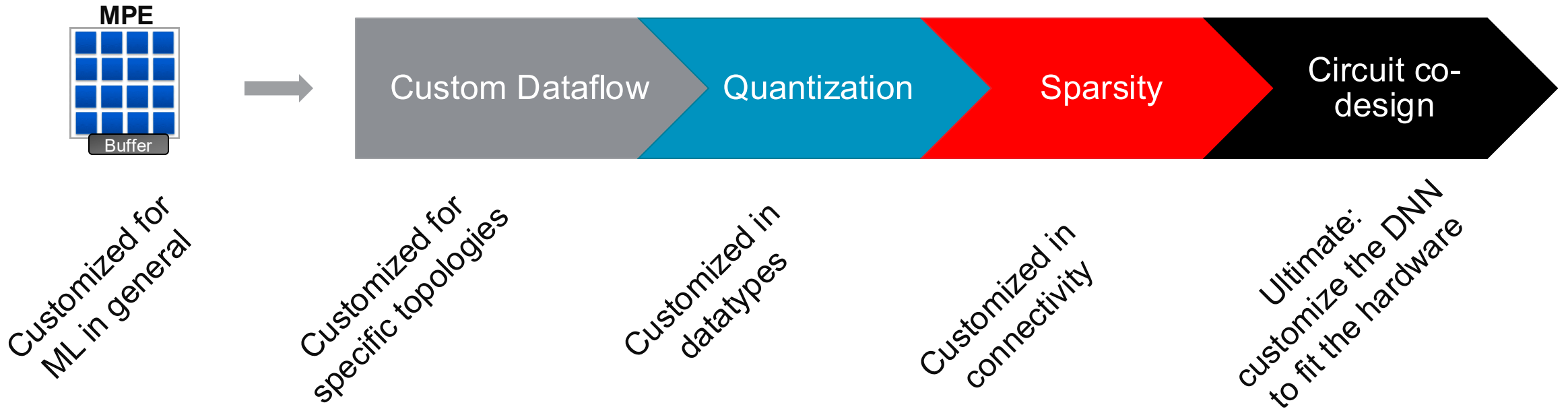
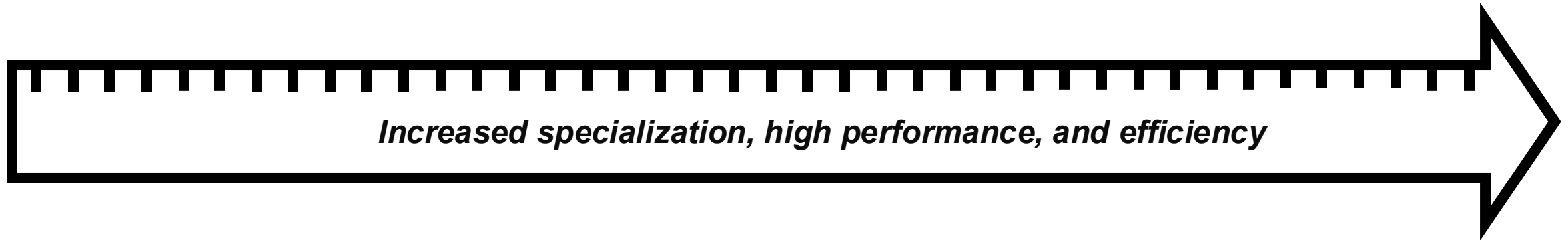
- MPE: Matrix of Processing Engines
- Popular layer-by-layer compute
- Batching to achieve high compute efficiency
- Customized for “NNs in general”
- Specialized processing engines
  - Operators
  - ALU types
    - tensor-, matrix- or vector-based

**MPEs can cater for a broad range of applications with one highly optimized architecture**

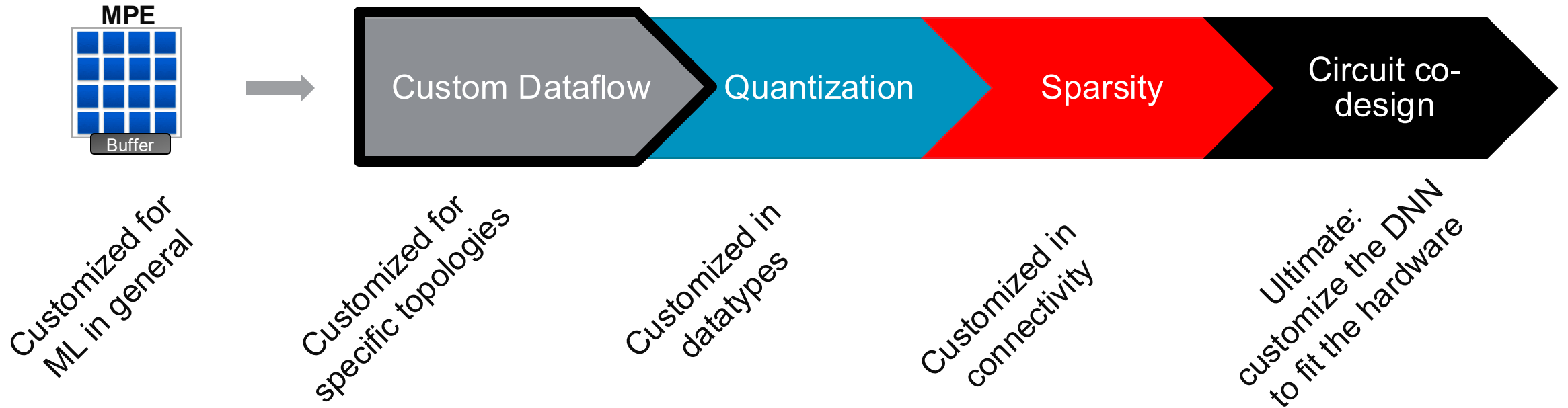
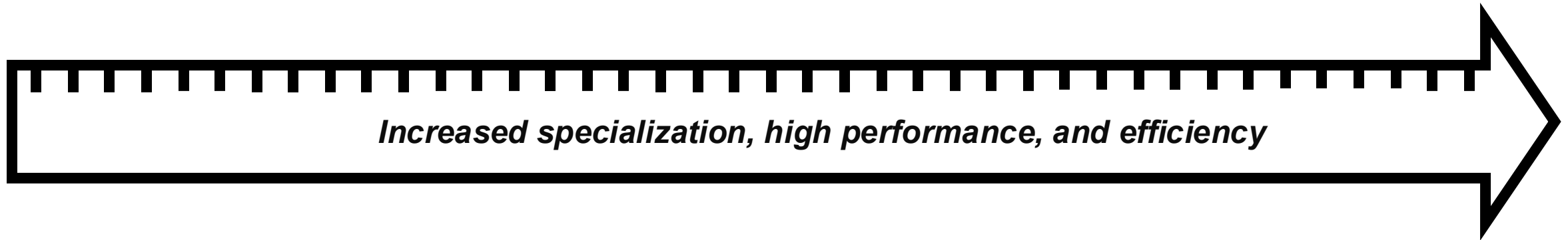
**MAC, VLIW,  
Vector Processor**



# Specialized FPGA Inference via Co-Design

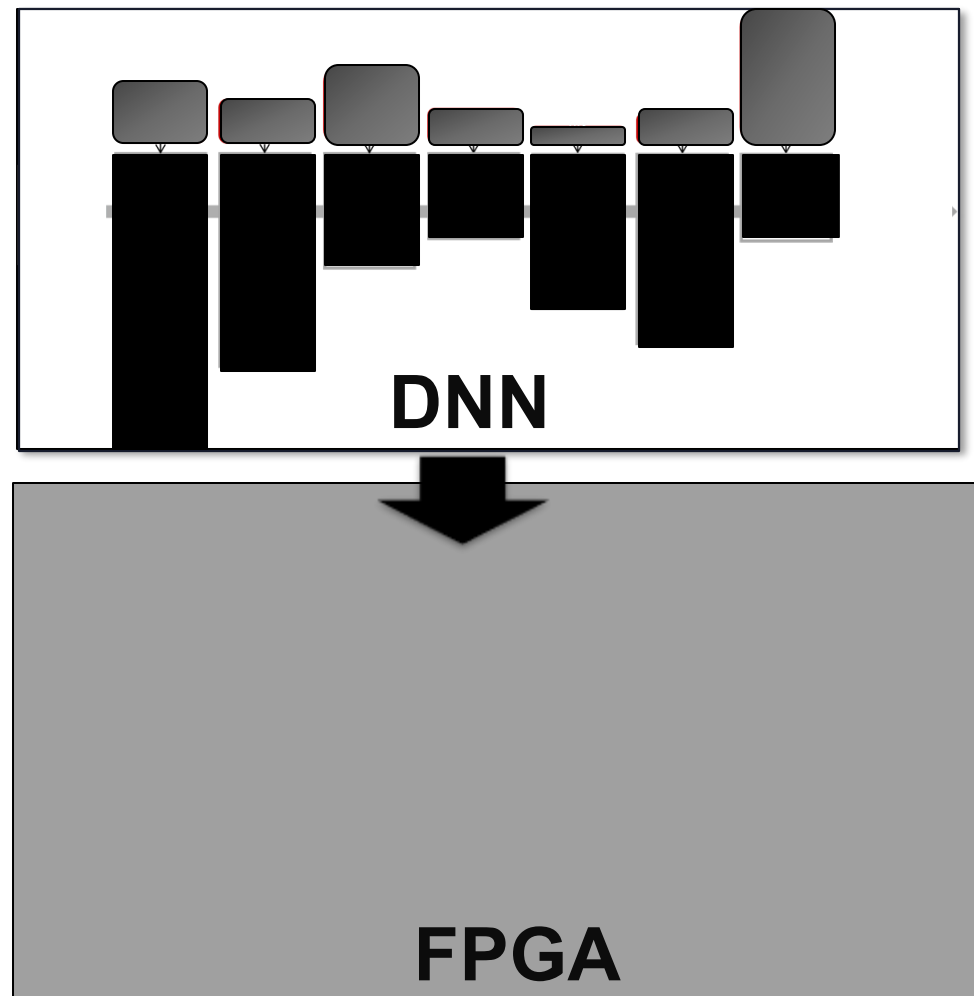


# Specialized FPGA Inference via Co-Design



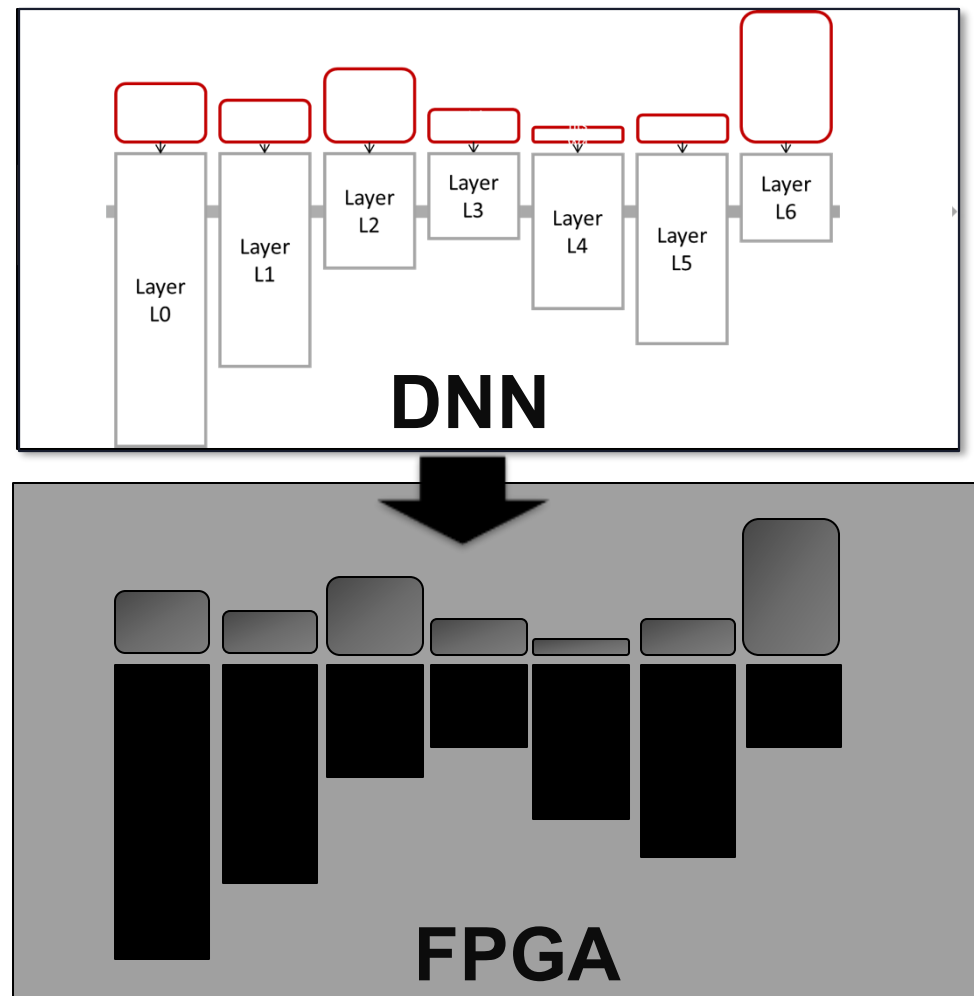
# Streaming Dataflow - Specializing for a Topology

- FPGA Dataflow NN Accelerator (FDNA)
- Hardware architecture mimics the topology
- Weights need to be accessible in parallel, but limited activation buffering needed
- Customize *everything* to the specifics of the DNN
- Benefits
  - Improved efficiency
  - Low fixed latency



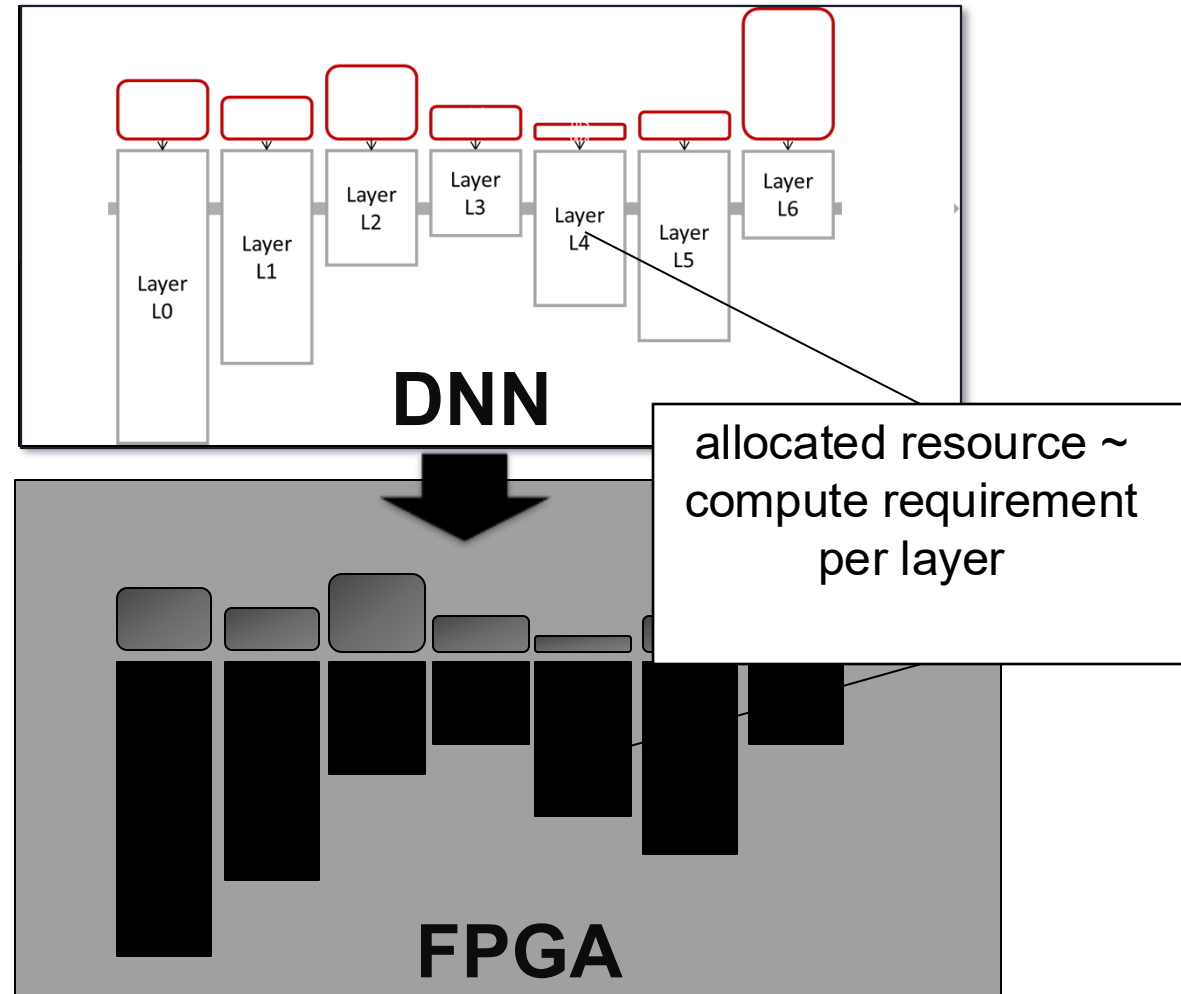
# Streaming Dataflow - Specializing for a Topology

- FPGA Dataflow NN Accelerator (FDNA)
- Hardware architecture mimics the topology
- Weights need to be accessible in parallel, but limited activation buffering needed
- Customize *everything* to the specifics of the DNN
- Benefits
  - Improved efficiency
  - Low fixed latency



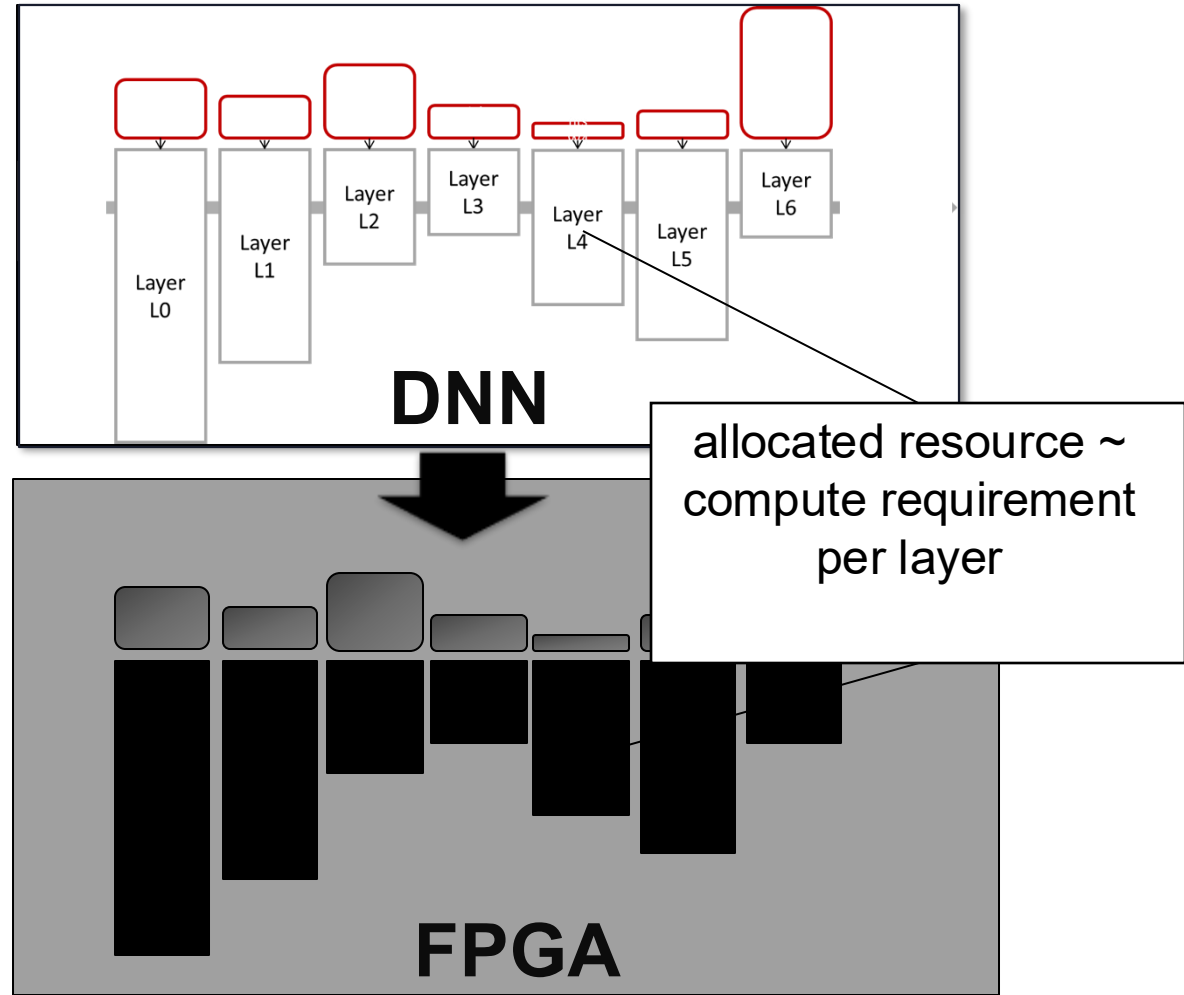
# Streaming Dataflow - Specializing for a Topology

- FPGA Dataflow NN Accelerator (FDNA)
- Hardware architecture mimics the topology
- Weights need to be accessible in parallel, but limited activation buffering needed
- Customize *everything* to the specifics of the DNN
- Benefits
  - Improved efficiency
  - Low fixed latency



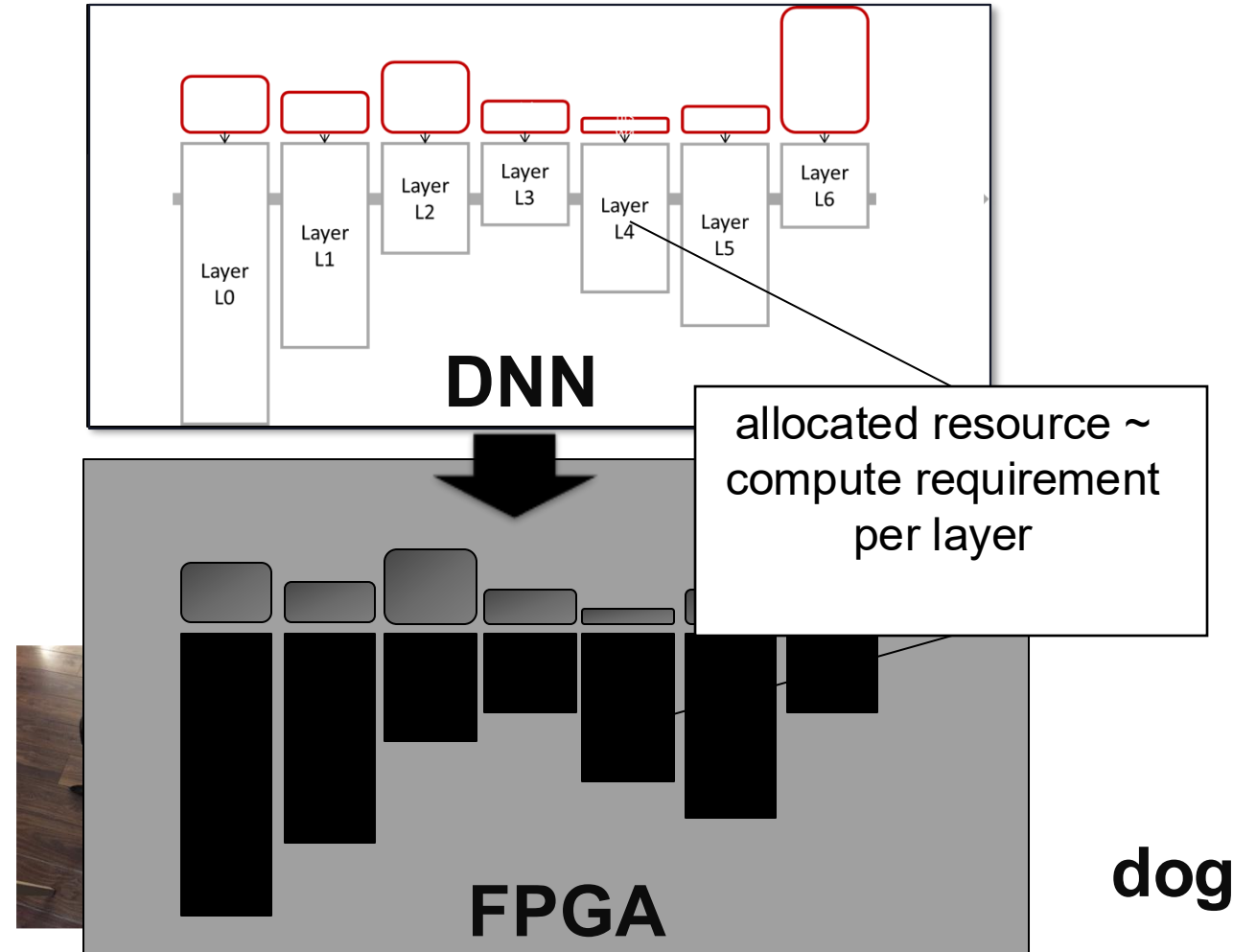
# Streaming Dataflow - Specializing for a Topology

- FPGA Dataflow NN Accelerator (FDNA)
- Hardware architecture mimics the topology
- Weights need to be accessible in parallel, but limited activation buffering needed
- Customize *everything* to the specifics of the DNN
- Benefits
  - Improved efficiency
  - Low fixed latency

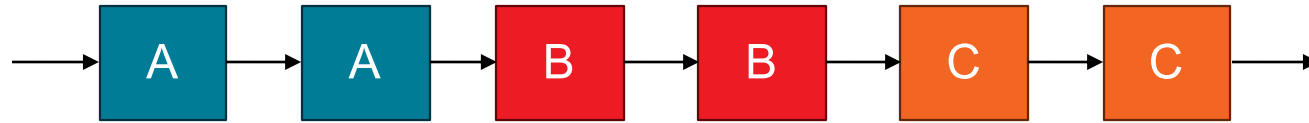


# Streaming Dataflow - Specializing for a Topology

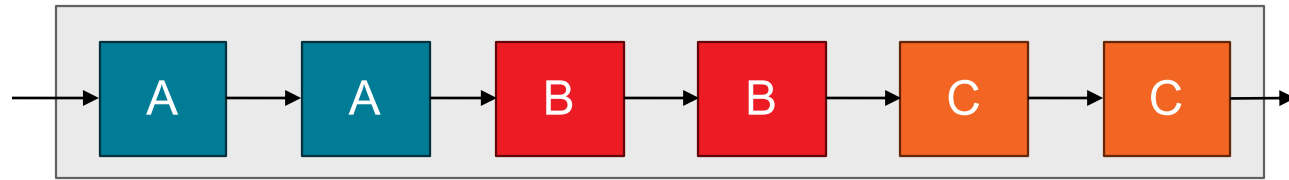
- FPGA Dataflow NN Accelerator (FDNA)
- Hardware architecture mimics the topology
- Weights need to be accessible in parallel, but limited activation buffering needed
- Customize *everything* to the specifics of the DNN
- Benefits
  - Improved efficiency
  - Low fixed latency



# Dataflow Architectures at Different Granularities

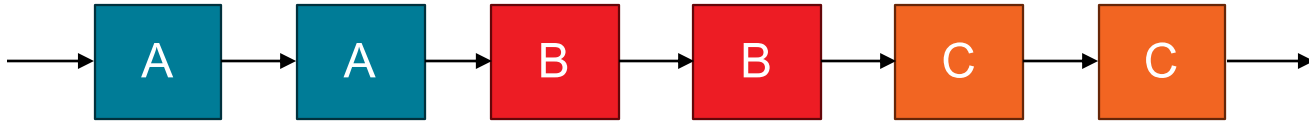


NN topology

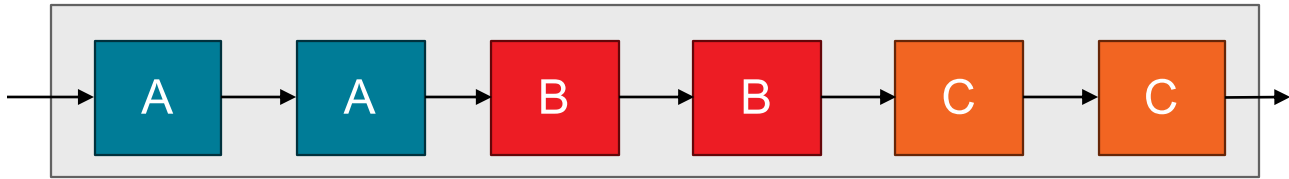


**full layer pipeline**  
each layer with dedicated hardware

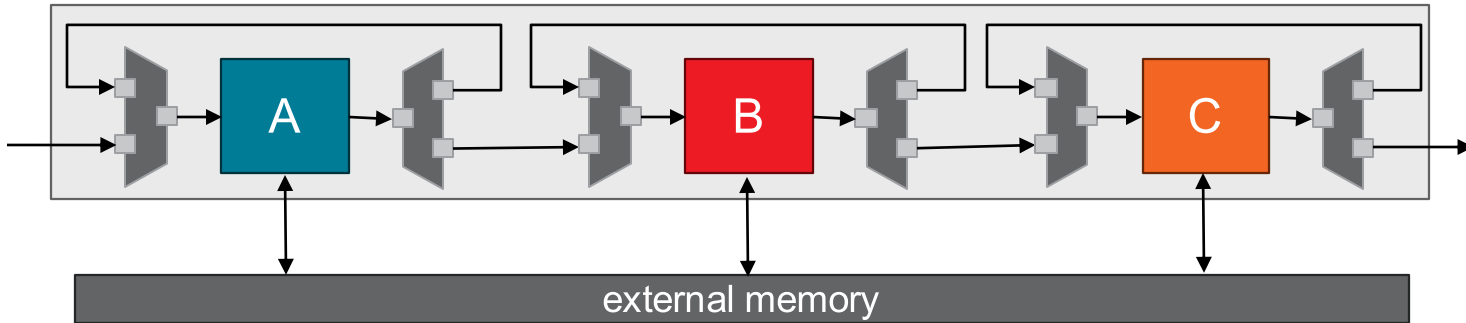
# Dataflow Architectures at Different Granularities



NN topology



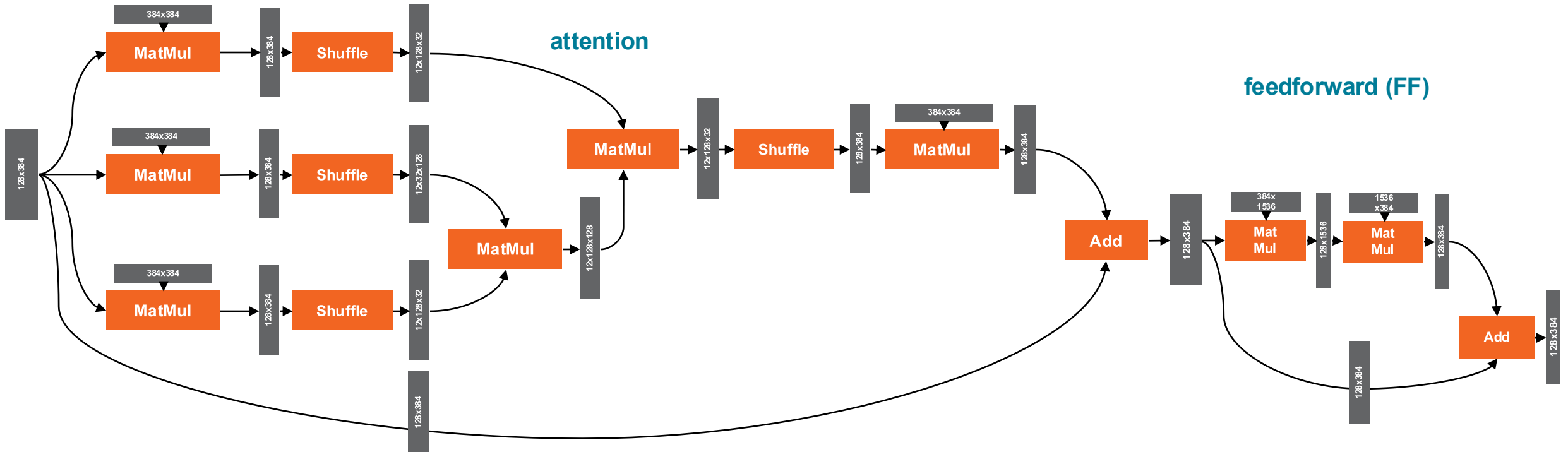
**full layer pipeline**  
each layer with dedicated hardware



**multi-layer offload (MLO)**  
identical blocks in same hardware

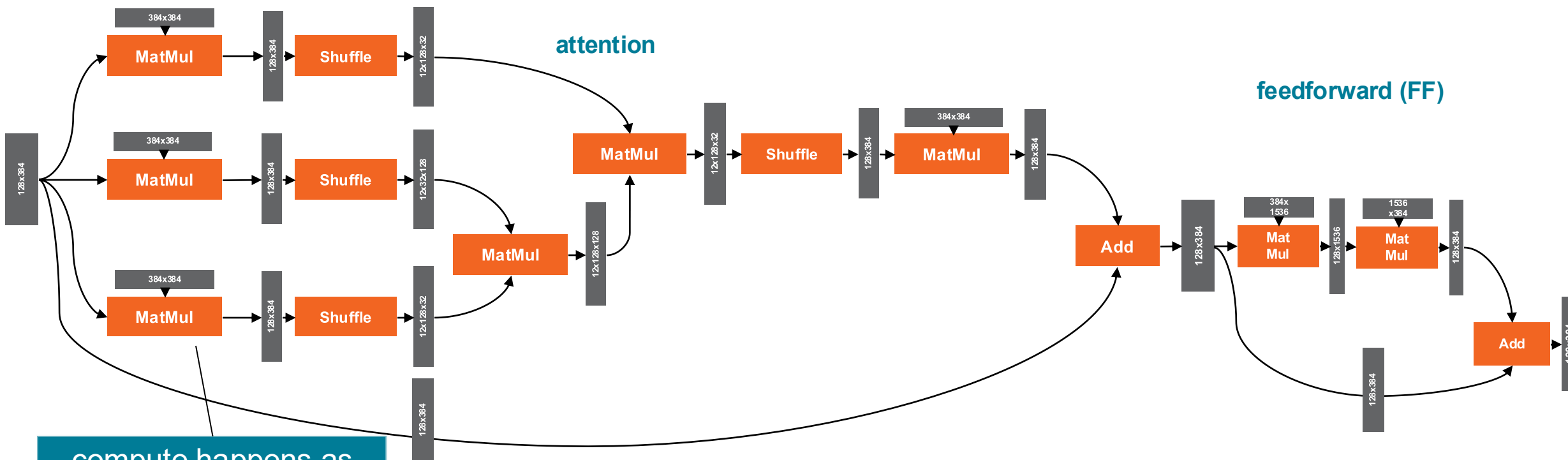


# Example: MLO Dataflow for a Transformer Block





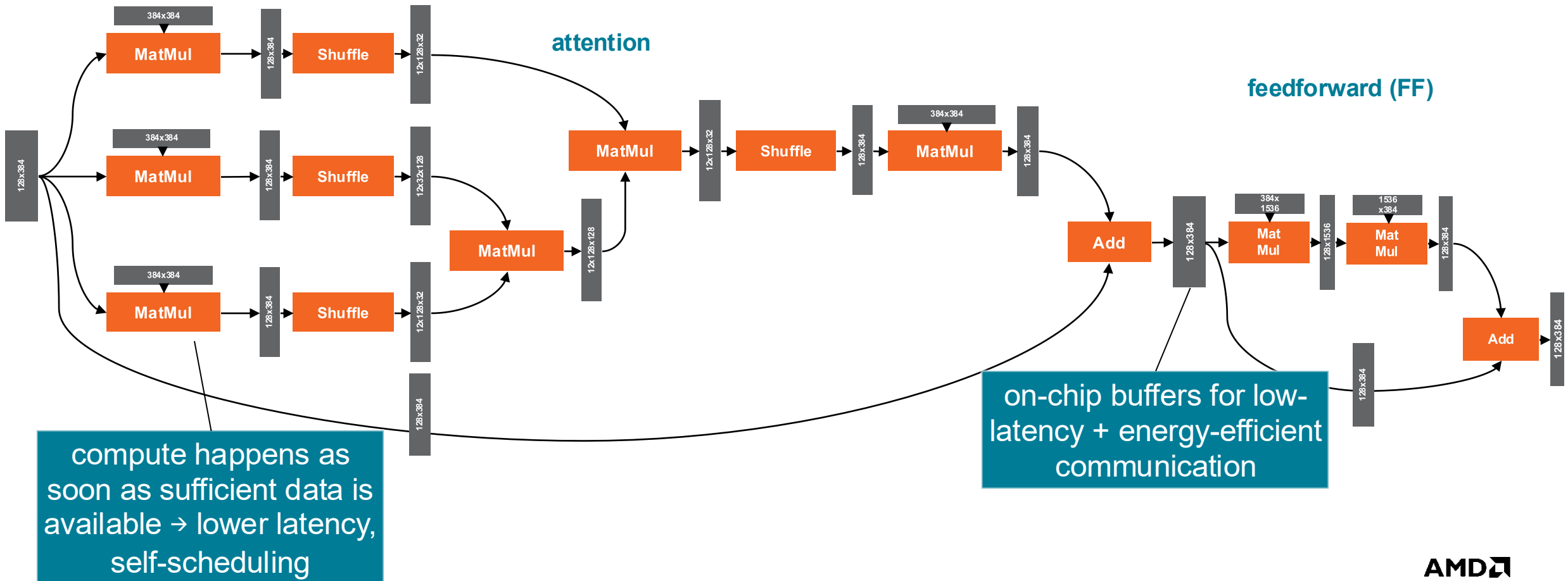
# Example: MLO Dataflow for a Transformer Block



compute happens as soon as sufficient data is available -> lower latency, self-scheduling

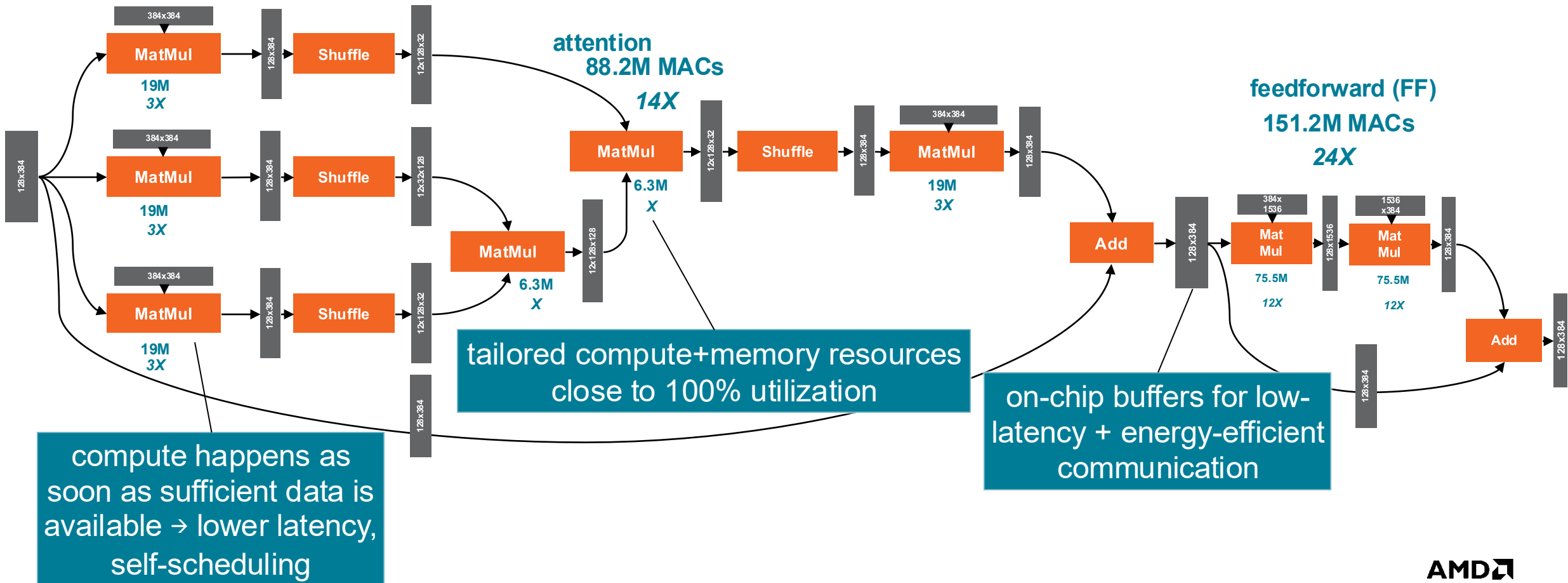


# Example: MLO Dataflow for a Transformer Block



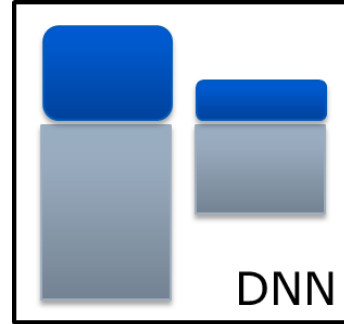


# Example: MLO Dataflow for a Transformer Block

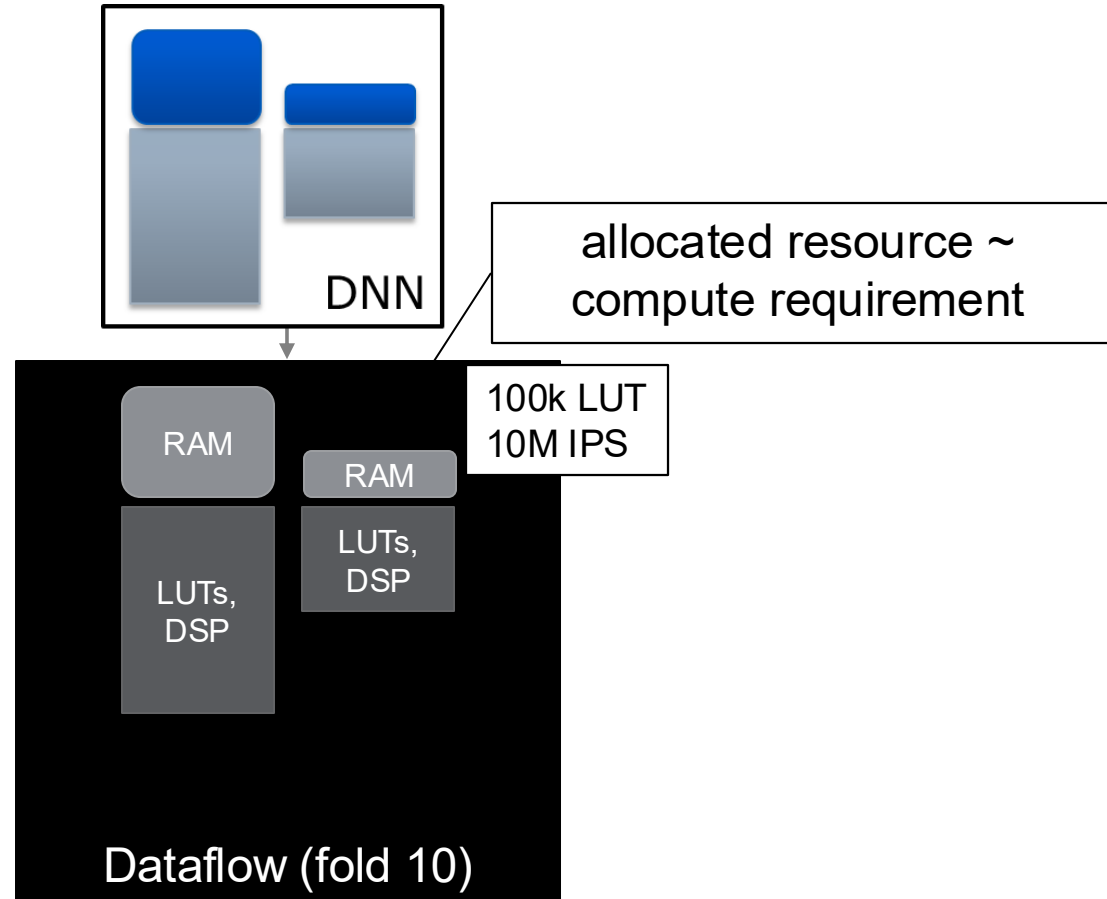




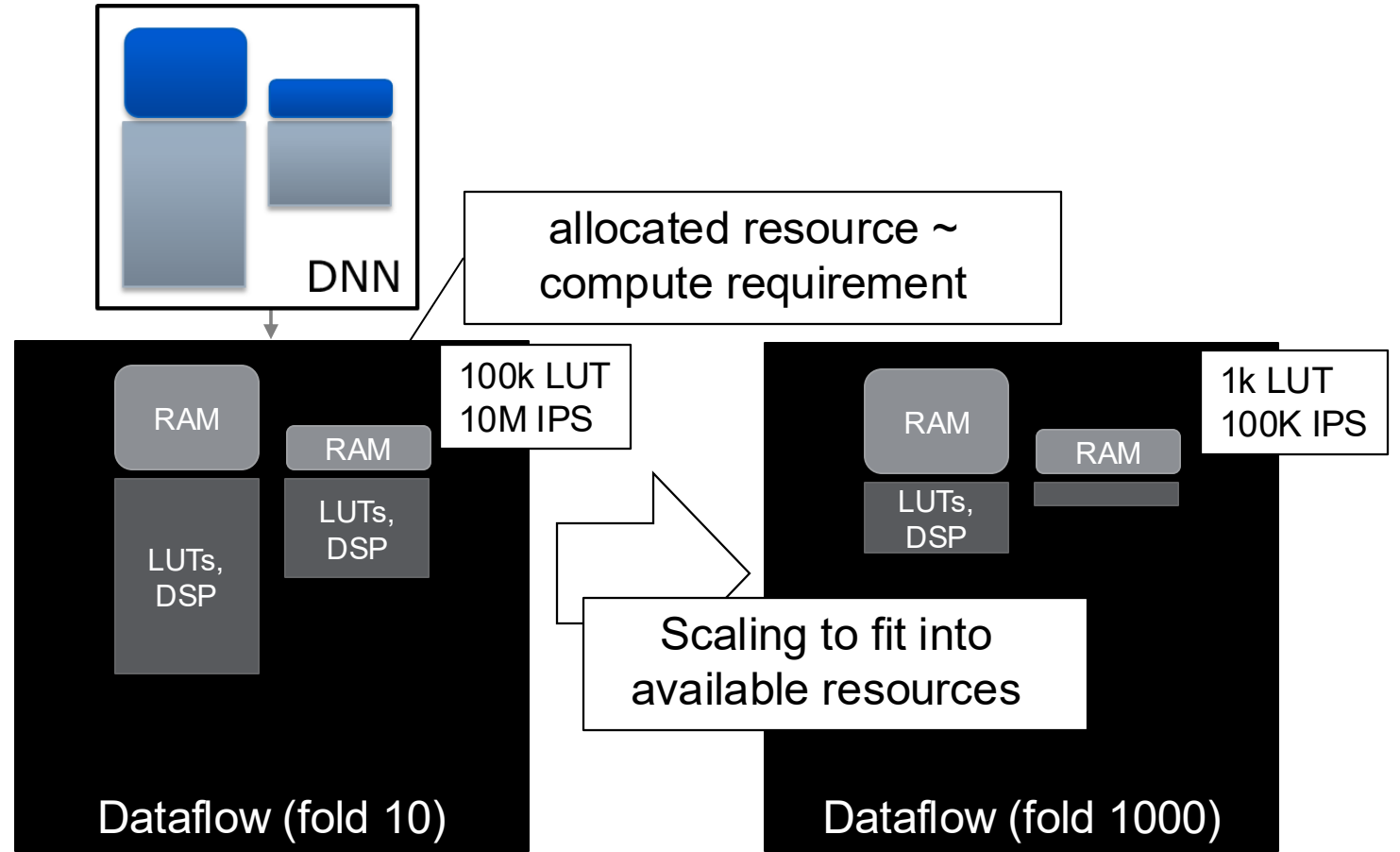
# Scaling to Meet Performance & Resource Requirements



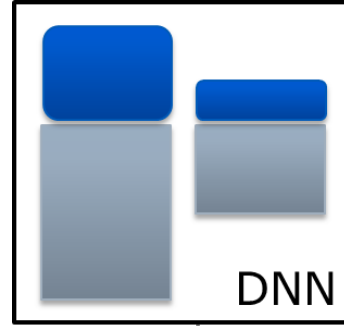
# Scaling to Meet Performance & Resource Requirements



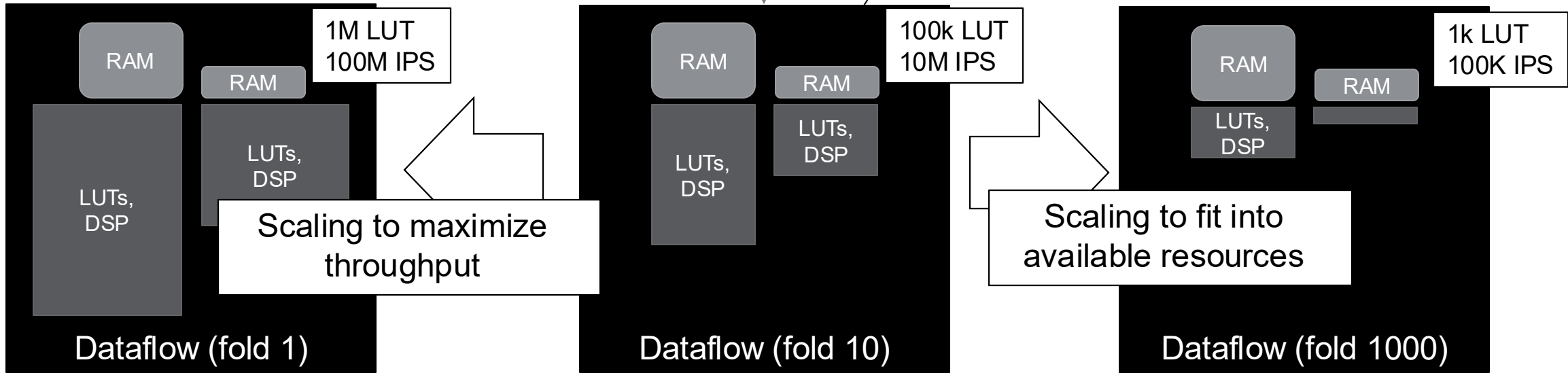
# Scaling to Meet Performance & Resource Requirements



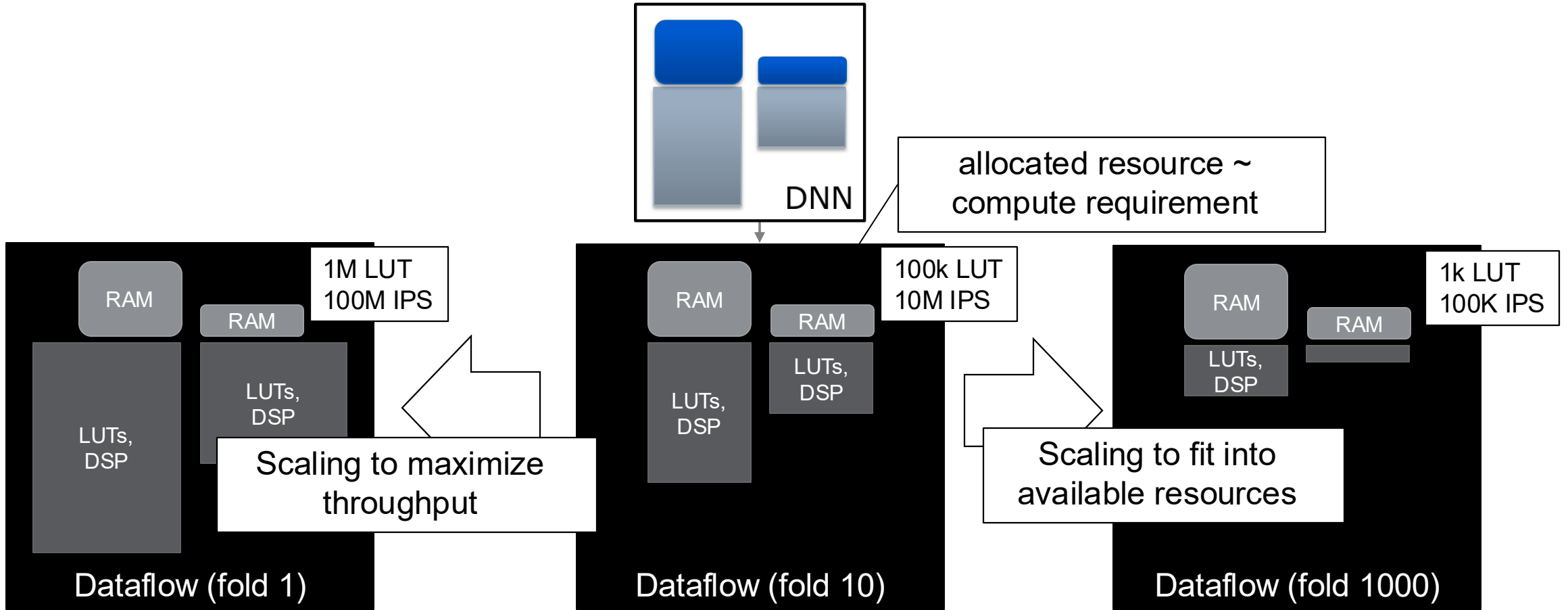
# Scaling to Meet Performance & Resource Requirements



allocated resource ~ compute requirement

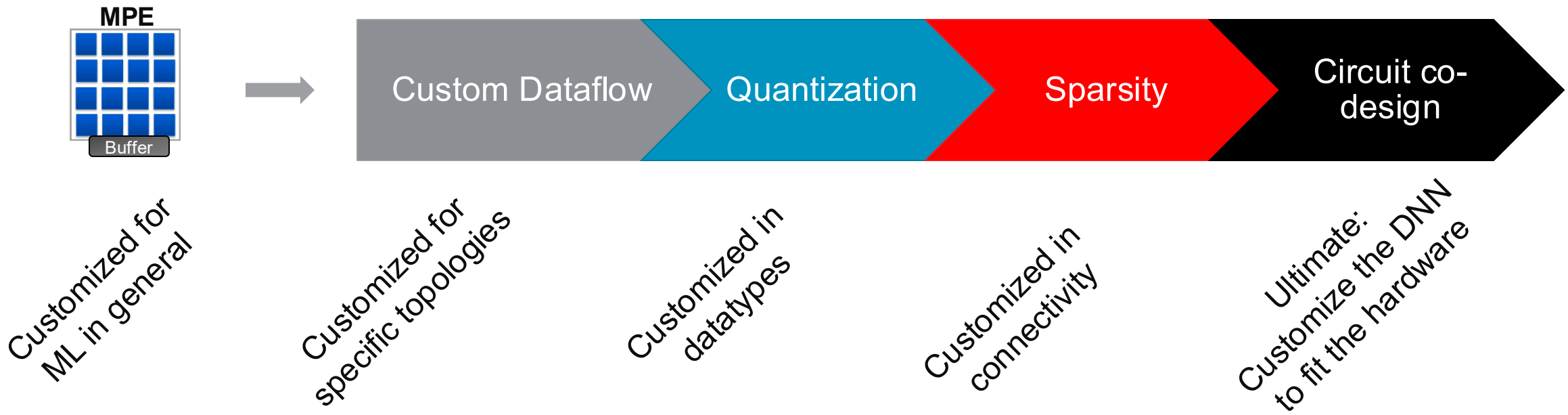


# Scaling to Meet Performance & Resource Requirements

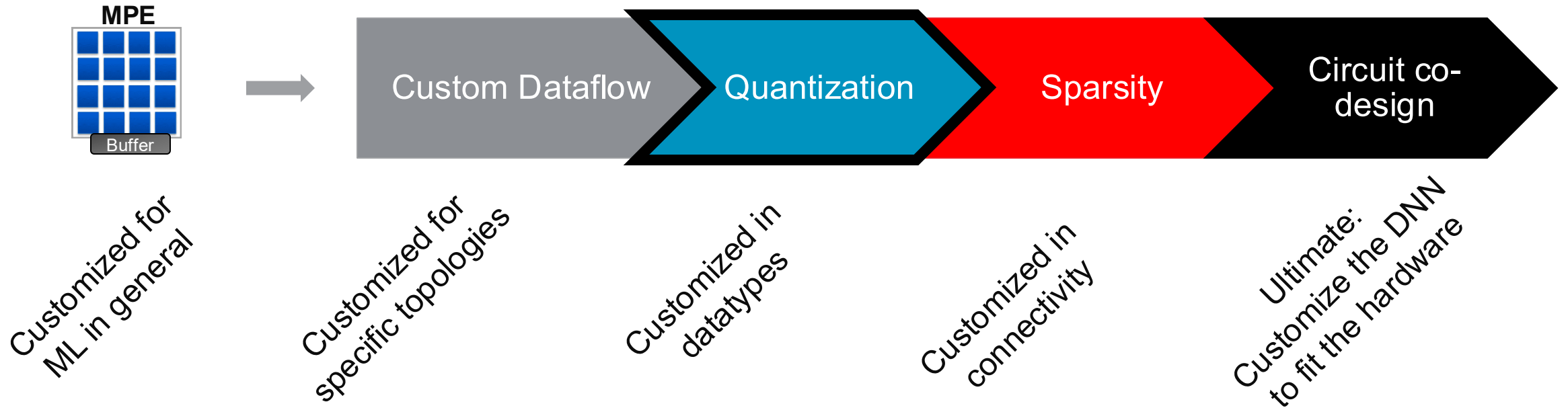
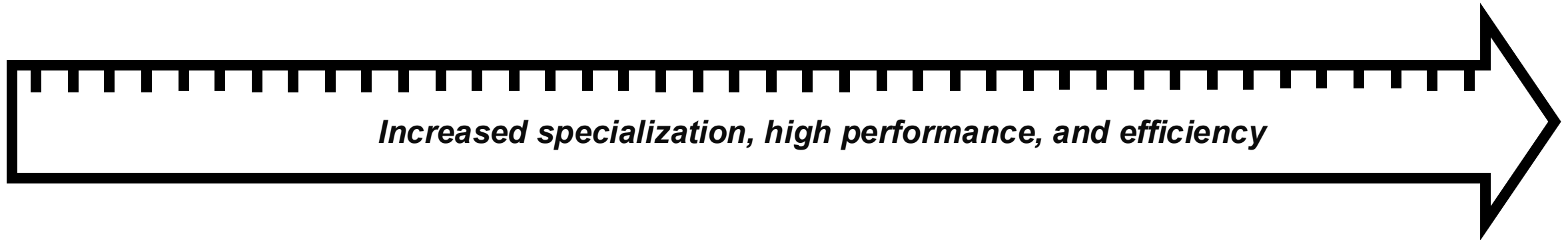


- Scale performance & resources to meet the application requirements
- If resources allow, we can **fully unfold the NN** to create a circuit that inferences at clock speed
  - Enables extra optimizations for fine-granular quantization and sparsity

# Specialized FPGA Inference via Co-Design



# Specialized FPGA Inference via Co-Design



# Benefits of Quantization on FPGAs

AMD UltraScale+ MPSoC ZU19EG (conservative estimates)

Precision	compute	memory
1b		
4b		
8b		
32b		

# Benefits of Quantization on FPGAs

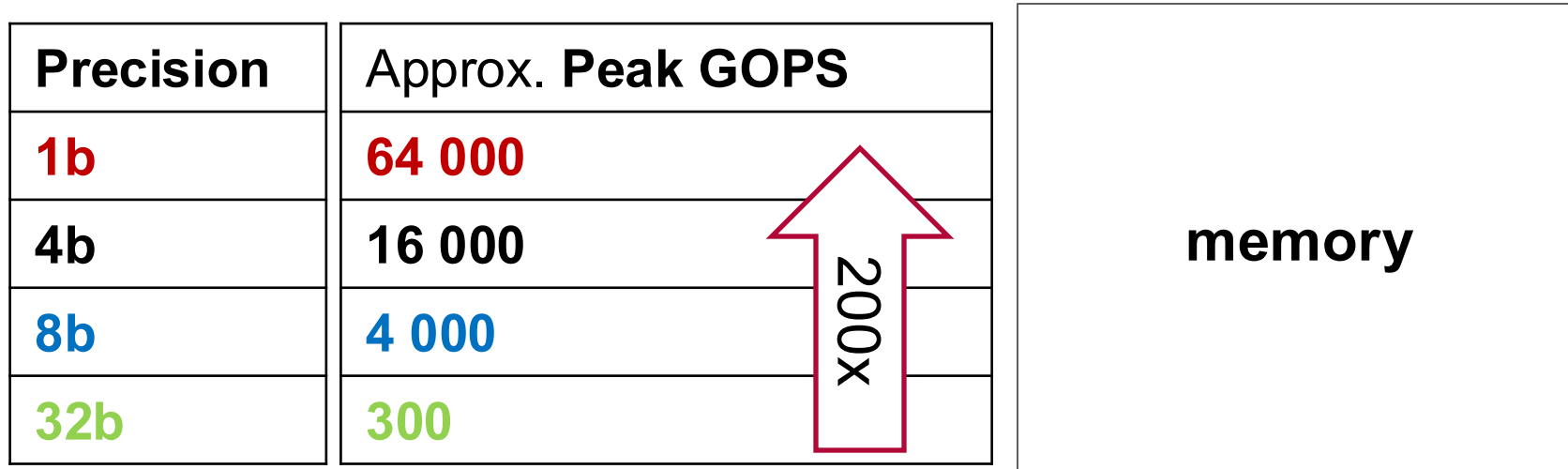
AMD UltraScale+ MPSoC ZU19EG (conservative estimates)

Precision	Approx. Peak GOPS
<b>1b</b>	<b>64 000</b>
<b>4b</b>	<b>16 000</b>
<b>8b</b>	<b>4 000</b>
<b>32b</b>	<b>300</b>

memory

# Benefits of Quantization on FPGAs

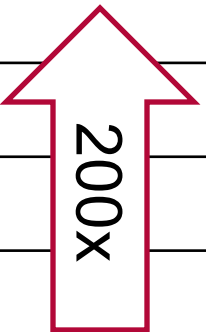
AMD UltraScale+ MPSoC ZU19EG (conservative estimates)



# Benefits of Quantization on FPGAs

AMD UltraScale+ MPSoC ZU19EG (conservative estimates)

Precision	Approx. Peak GOPS
1b	64 000
4b	16 000
8b	4 000
32b	300



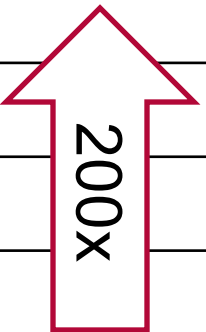
memory

**Trillions** of  
quantized  
operations per  
second

# Benefits of Quantization on FPGAs

AMD UltraScale+ MPSoC ZU19EG (conservative estimates)

Precision	Approx. Peak GOPS	On-chip weights
<b>1b</b>	<b>64 000</b>	<b>~64 M</b>
<b>4b</b>	<b>16 000</b>	<b>~16 M</b>
<b>8b</b>	<b>4 000</b>	<b>~8 M</b>
<b>32b</b>	<b>300</b>	<b>~2 M</b>

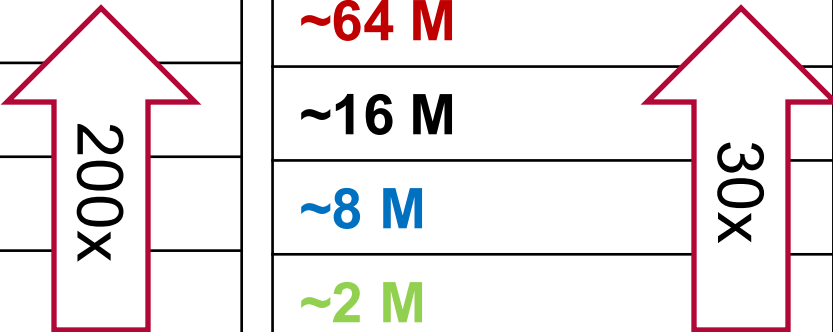


**Trillions** of  
quantized  
operations per  
second

# Benefits of Quantization on FPGAs

AMD UltraScale+ MPSoC ZU19EG (conservative estimates)

Precision	Approx. Peak GOPS	On-chip weights
1b	64 000	~64 M
4b	16 000	~16 M
8b	4 000	~8 M
32b	300	~2 M

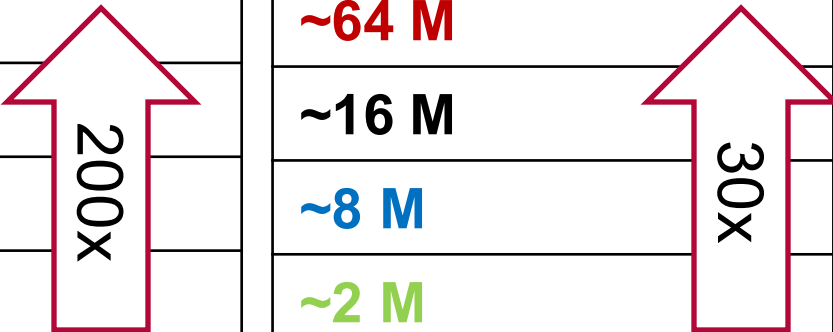


**Trillions** of  
quantized  
operations per  
second

# Benefits of Quantization on FPGAs

AMD UltraScale+ MPSoC ZU19EG (conservative estimates)

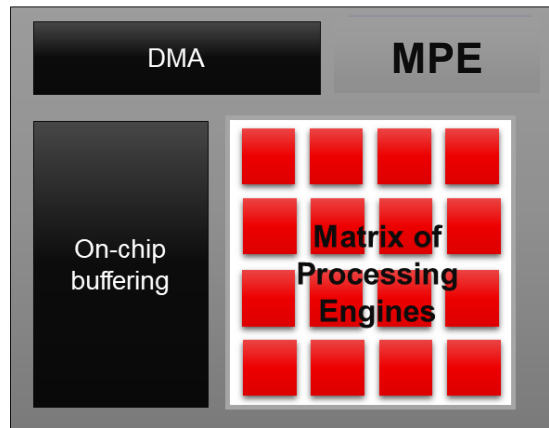
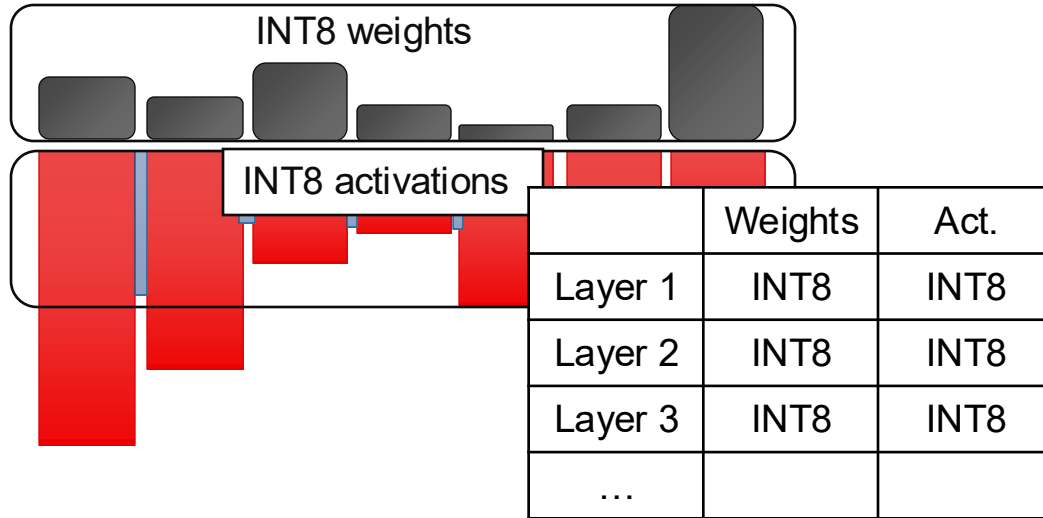
Precision	Approx. Peak GOPS	On-chip weights
1b	64 000	~64 M
4b	16 000	~16 M
8b	4 000	~8 M
32b	300	~2 M



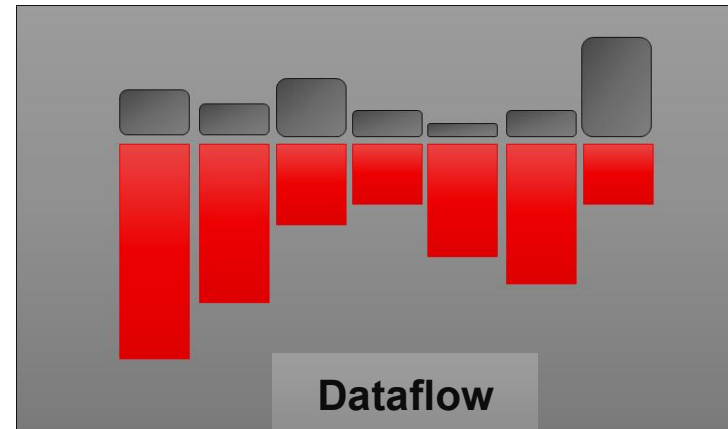
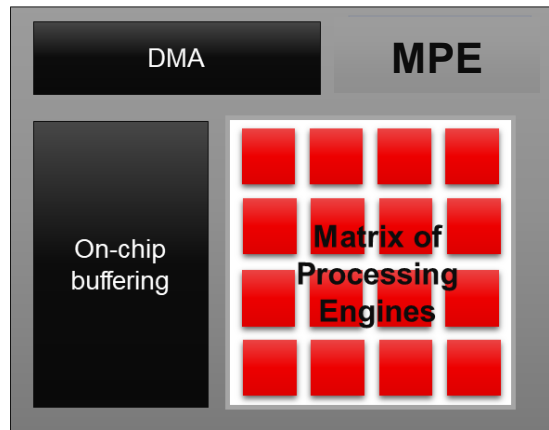
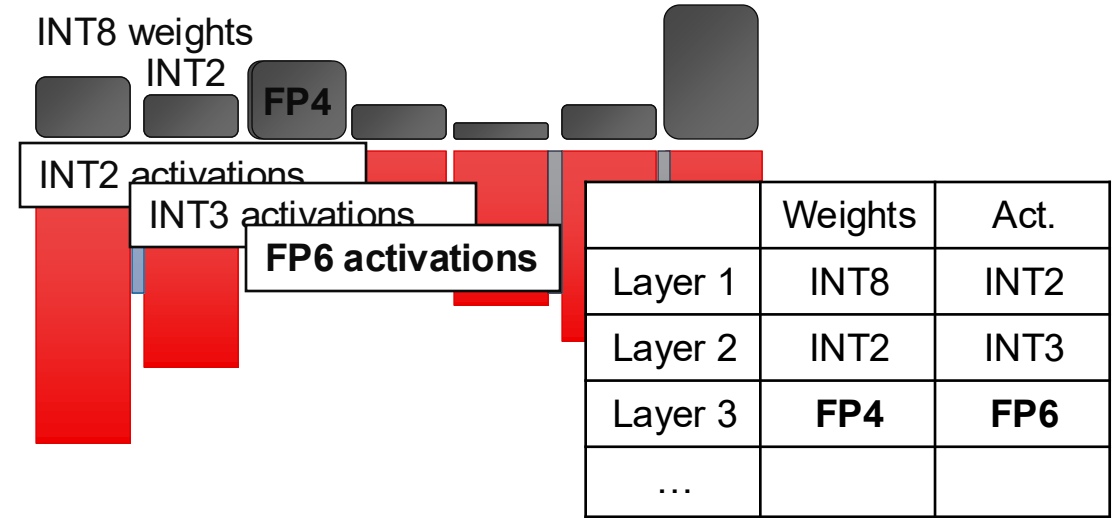
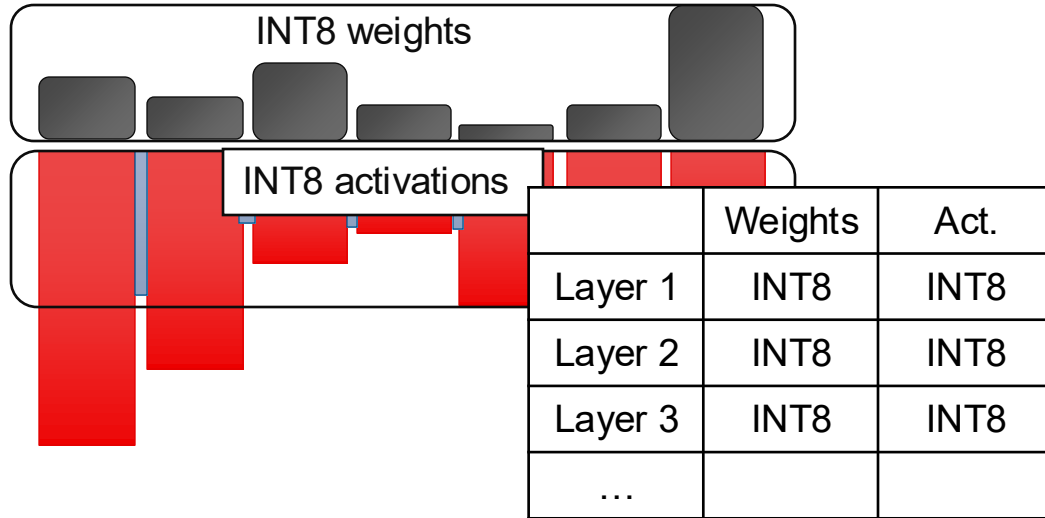
**Trillions** of quantized operations per second

Weights can stay **entirely on-chip**

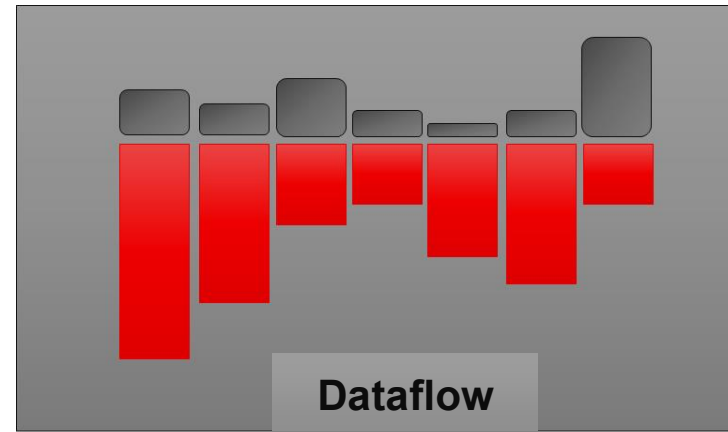
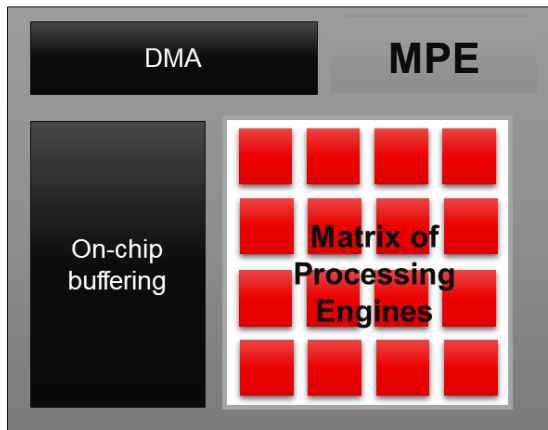
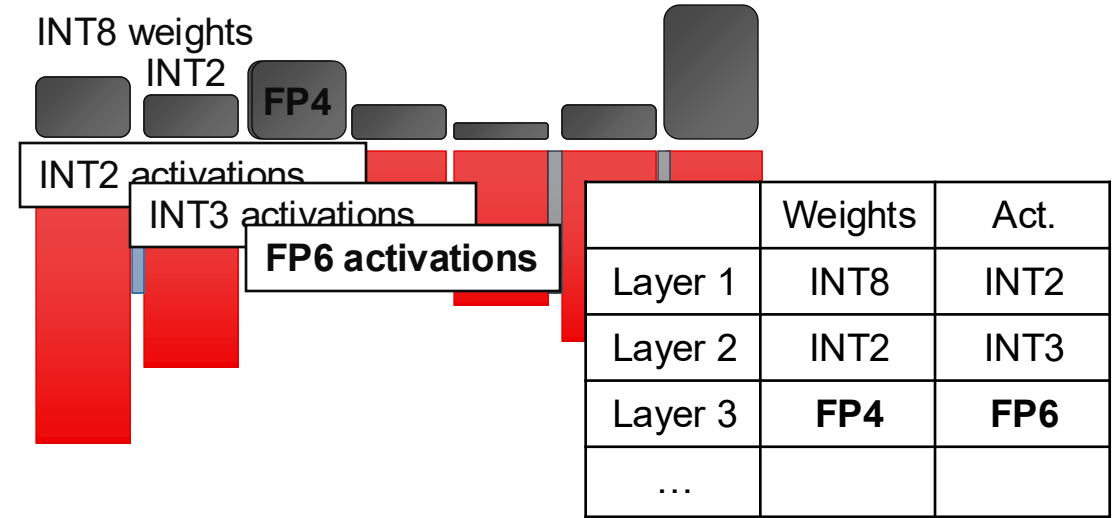
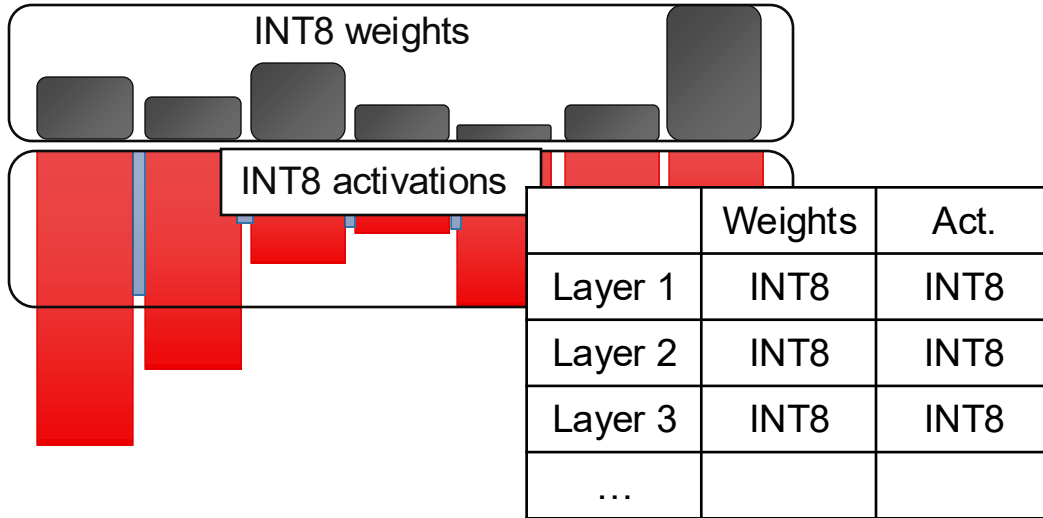
# Granularity of Customizing Arithmetic



# Granularity of Customizing Arithmetic

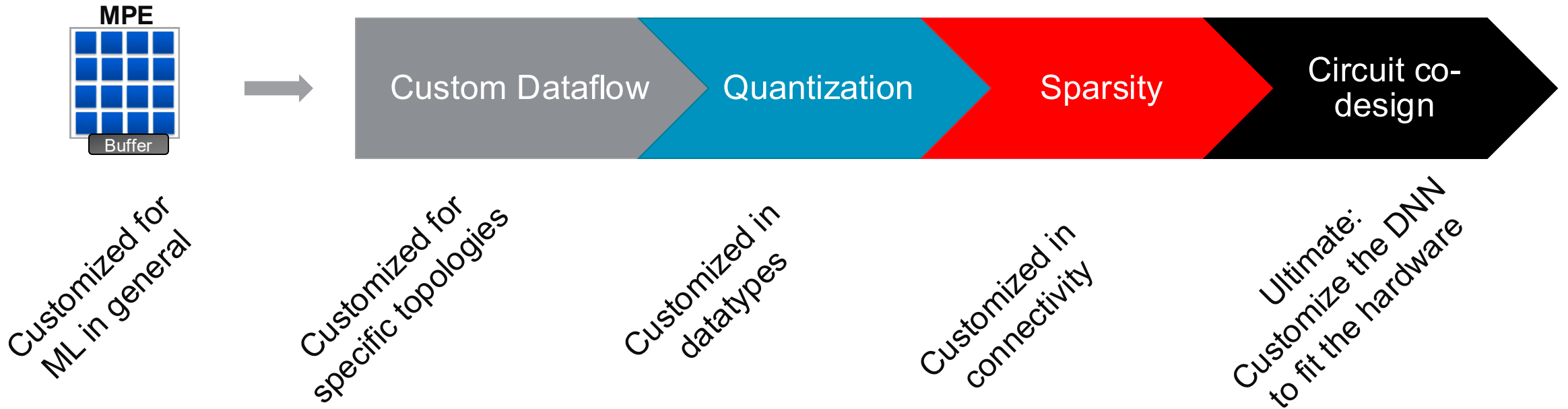
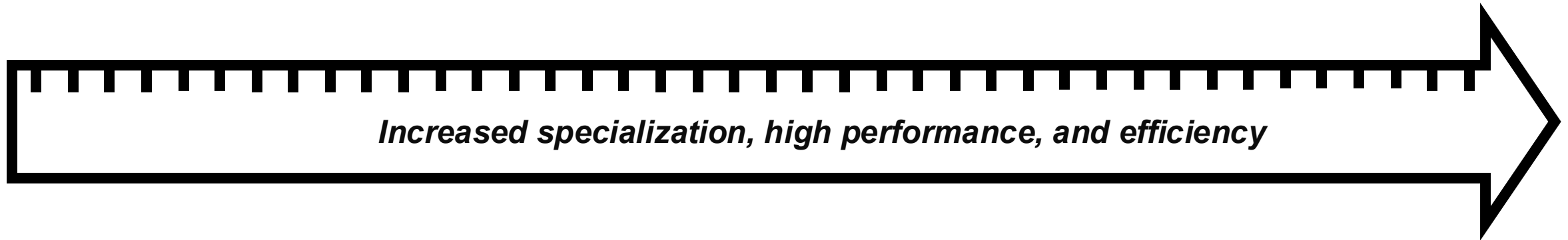


# Granularity of Customizing Arithmetic

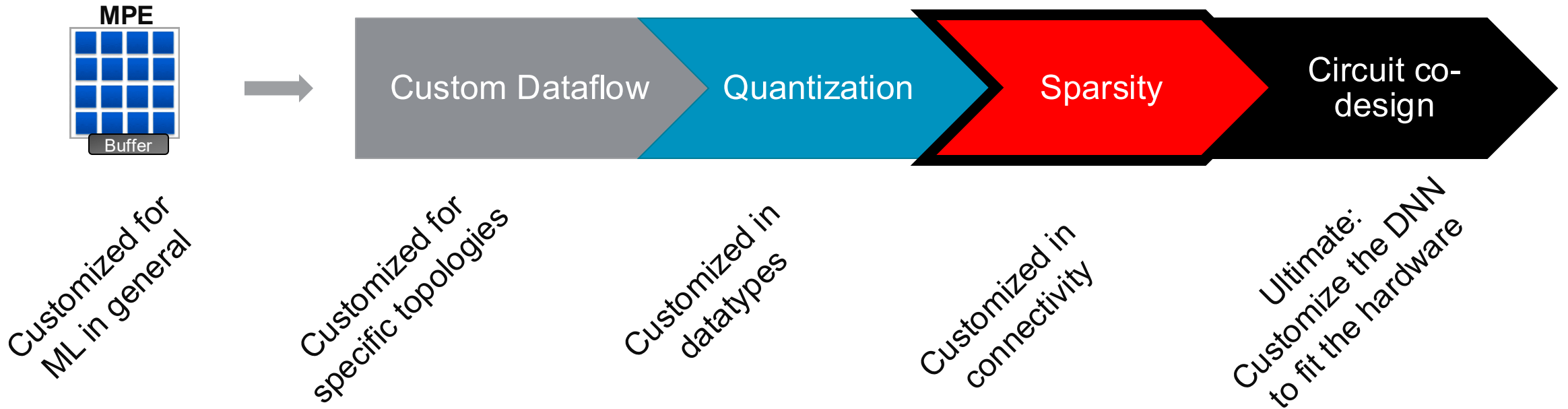
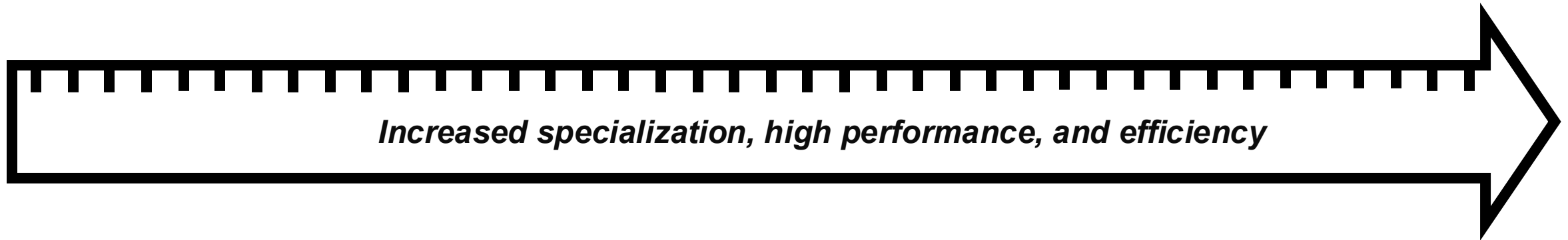


Dataflow architectures can exploit custom arithmetic at a finer granularity - even **per-neuron and per-synapse custom arithmetic** with full unfolding

# Specialized FPGA Inference via Co-Design

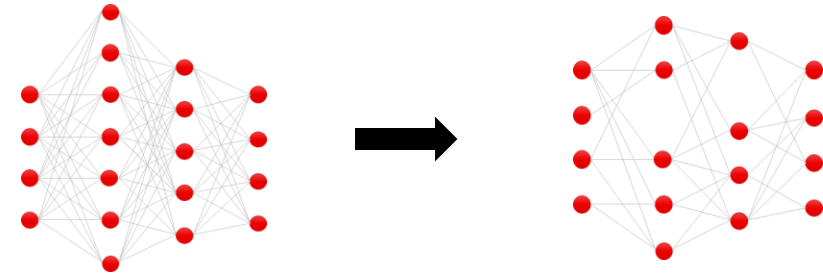


# Specialized FPGA Inference via Co-Design



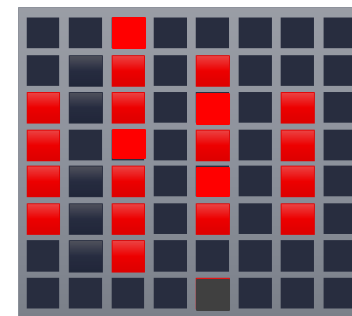
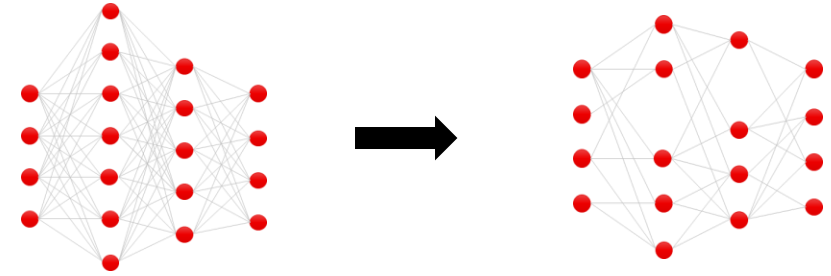
# Sparsity

- DNNs are naturally sparse
  - Zero- or near-zero weights, ReLU activations...
  - Multiplications with zero can be skipped => reduces compute load
- Sparse topologies result in irregular compute & memory access patterns
  - Hard to accelerate on vector- or matrix-based execution units
  - Structured sparsity better, but limits benefits

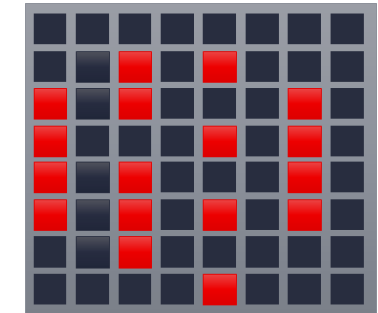


# Sparsity

- DNNs are naturally sparse
  - Zero- or near-zero weights, ReLU activations...
  - Multiplications with zero can be skipped => reduces compute load
- Sparse topologies result in irregular compute & memory access patterns
  - Hard to accelerate on vector- or matrix-based execution units
  - Structured sparsity better, but limits benefits
- Fully-unrolled streaming dataflow can also exploit **unstructured sparsity**
  - Each neuron & synapse has its own hardware
  - Becomes a synthesis P&R (non-)problem



**Dense  
Dataflow  
on FPGA**



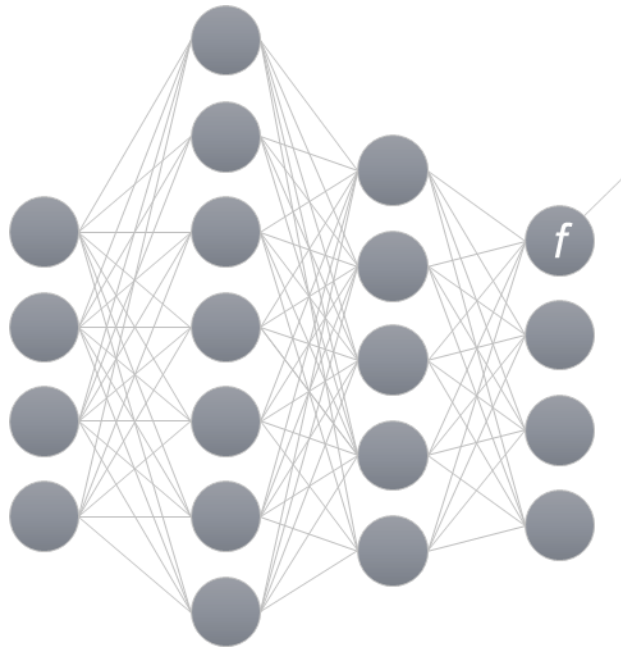
**Sparse  
Dataflow  
on FPGA**

# What does it cost to «run» a DNN?

- How much (time/energy/resources/...) is the **inference cost**?
  - With low cost, high throughput + low latency implementation is possible
  - With low cost, small footprint implementation is possible

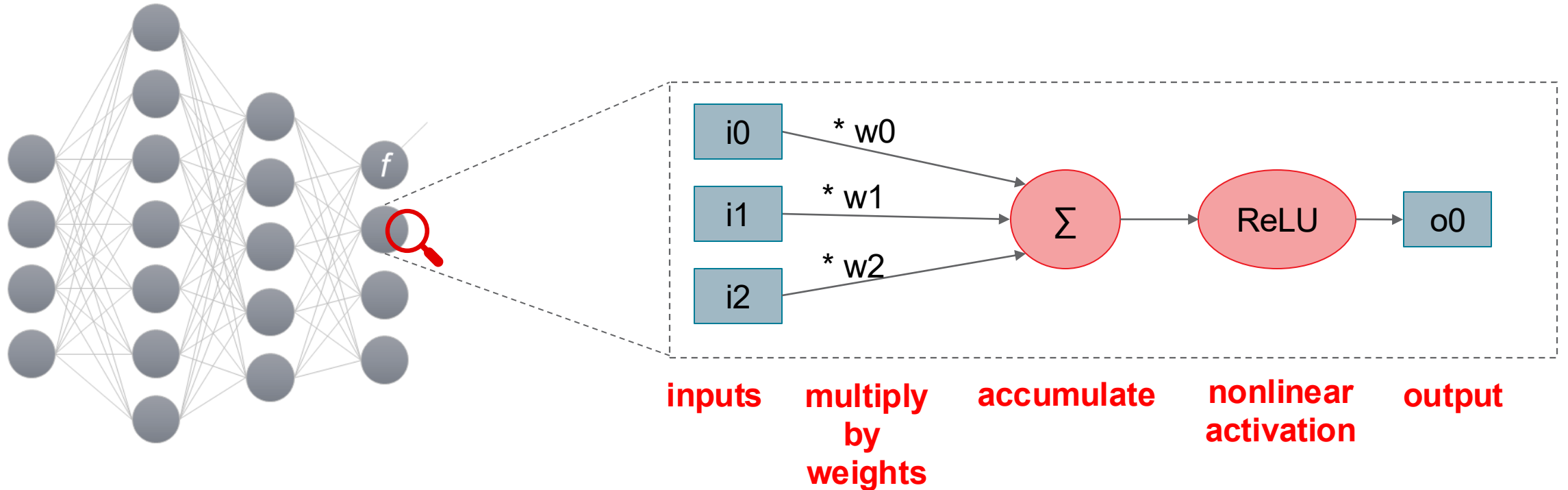
# What does it cost to «run» a DNN?

- How much (time/energy/resources/...) is the **inference cost**?
  - With low cost, high throughput + low latency implementation is possible
  - With low cost, small footprint implementation is possible
- Key determinants: cost of **multiply-accumulate (MAC) operations**



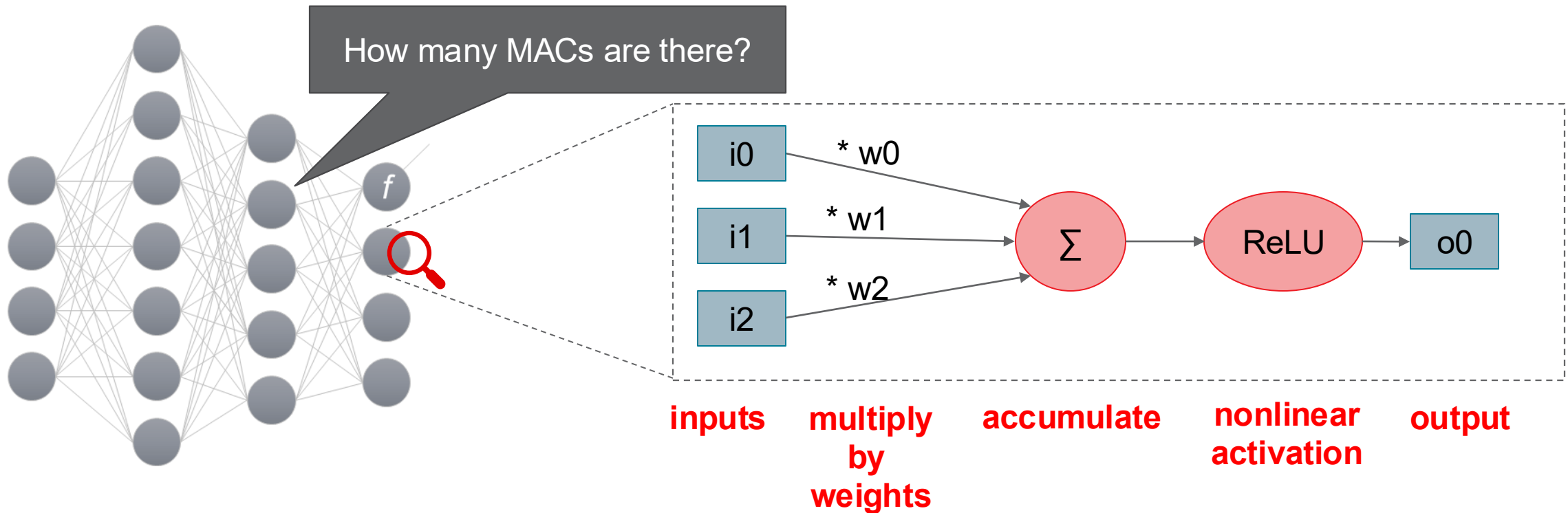
# What does it cost to «run» a DNN?

- How much (time/energy/resources/...) is the **inference cost**?
  - With low cost, high throughput + low latency implementation is possible
  - With low cost, small footprint implementation is possible
- Key determinants: cost of **multiply-accumulate (MAC) operations**



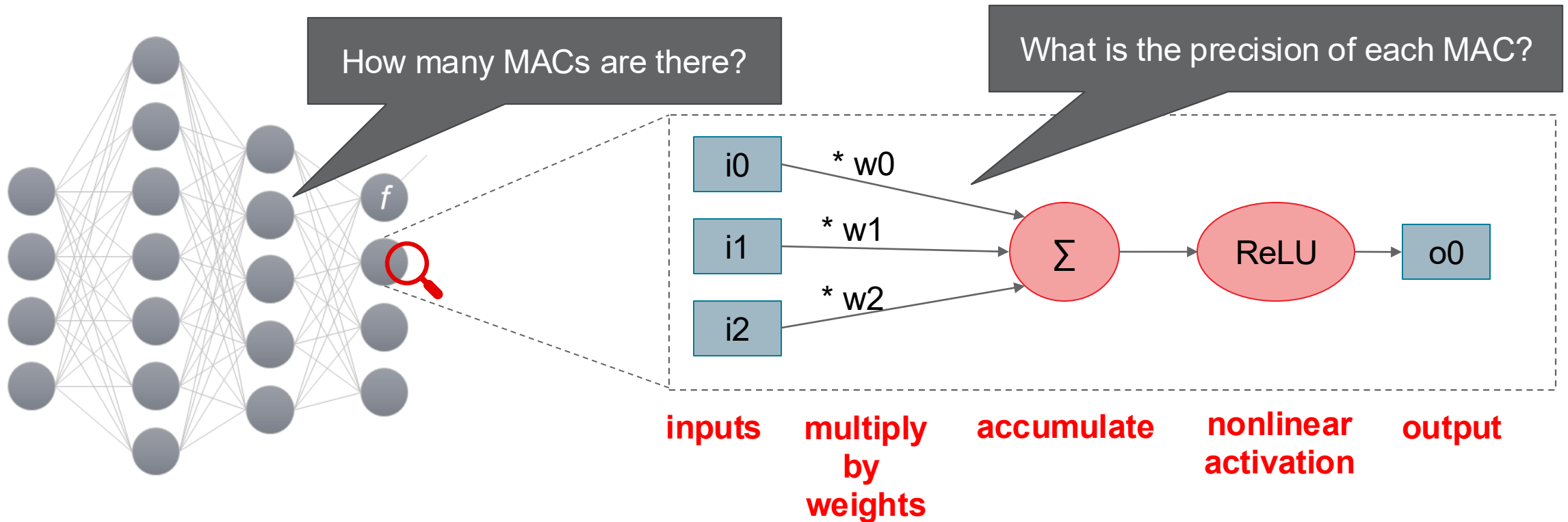
# What does it cost to «run» a DNN?

- How much (time/energy/resources/...) is the **inference cost**?
  - With low cost, high throughput + low latency implementation is possible
  - With low cost, small footprint implementation is possible
- Key determinants: cost of **multiply-accumulate (MAC) operations**



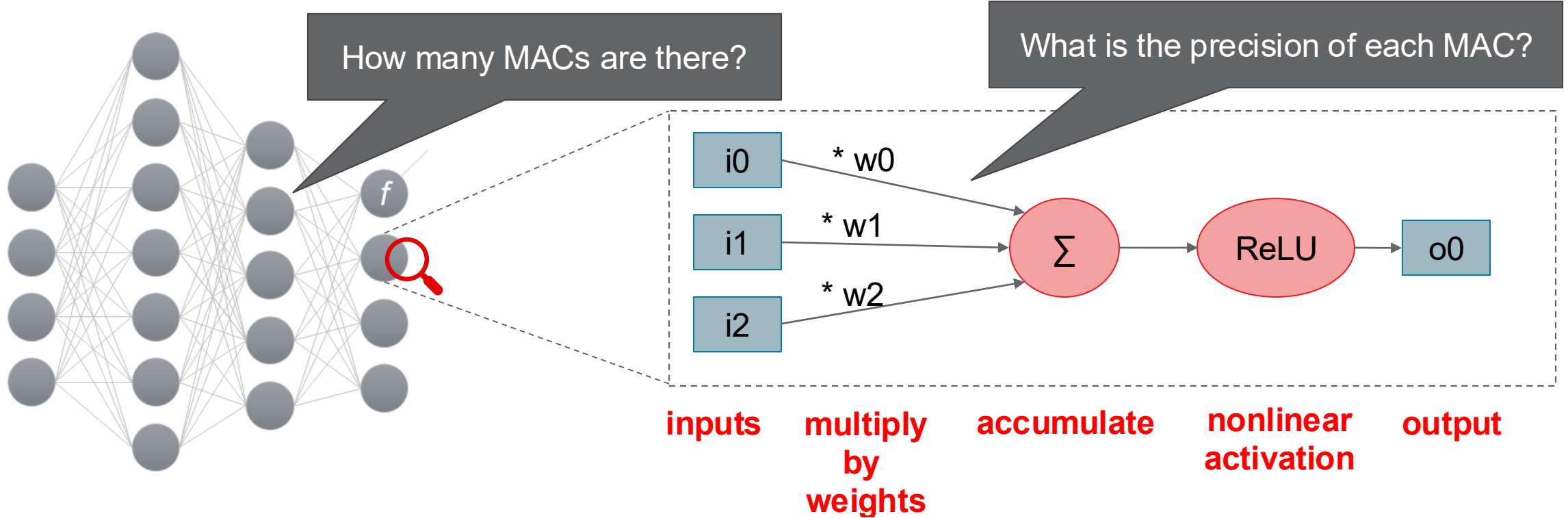
# What does it cost to «run» a DNN?

- How much (time/energy/resources/...) is the **inference cost**?
  - With low cost, high throughput + low latency implementation is possible
  - With low cost, small footprint implementation is possible
- Key determinants: cost of **multiply-accumulate (MAC) operations**



# What does it cost to «run» a DNN?

- How much (time/energy/resources/...) is the **inference cost**?
  - With low cost, high throughput + low latency implementation is possible
  - With low cost, small footprint implementation is possible
- Key determinants: cost of **multiply-accumulate (MAC) operations**
  - Various analytical model variants exist: bit operations [Hawks et al, Ps and Qs], LUT models [Blott et al, FINN-R]...

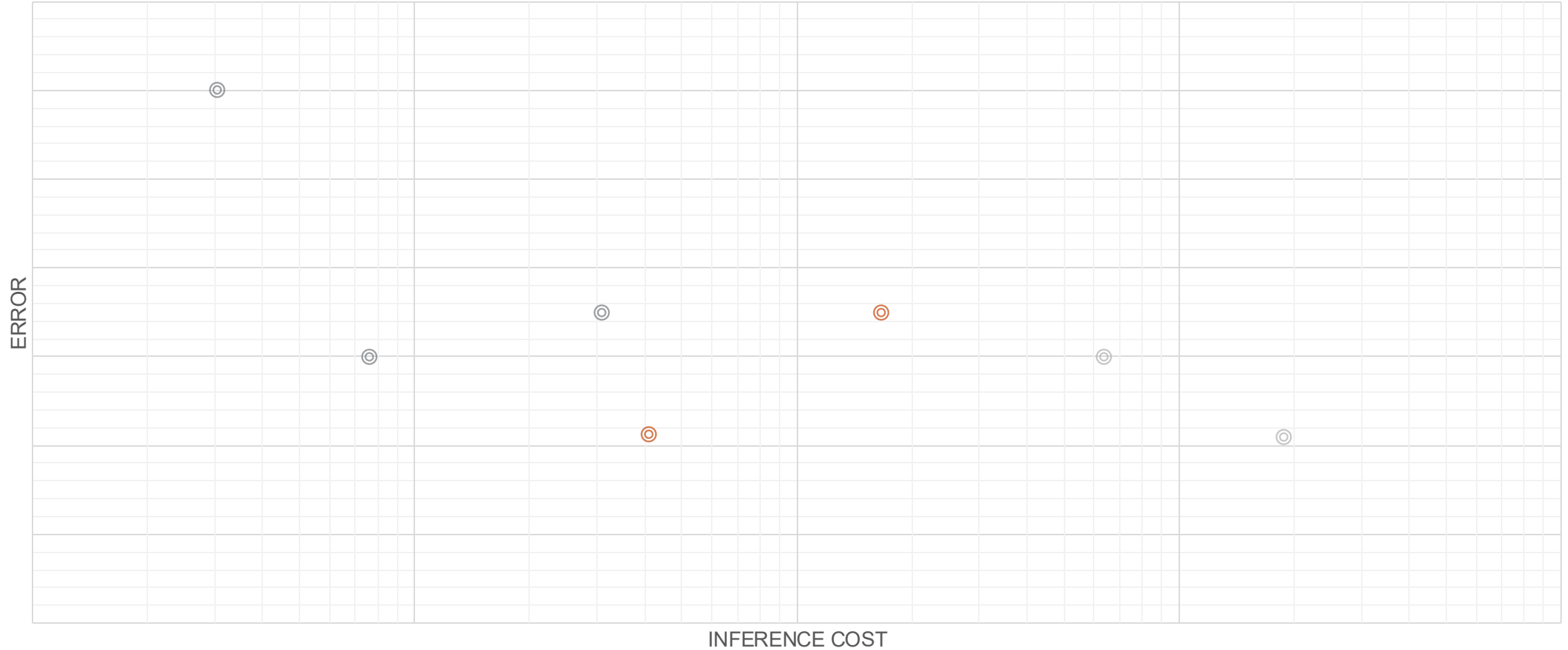


$$\text{BOPs} = mn \left[ (1 - f_p) b_a b_w + b_a + b_w + \log_2(n) \right]$$

# Design Trade-offs with Quantization + Sparsity

Error vs Compute Cost

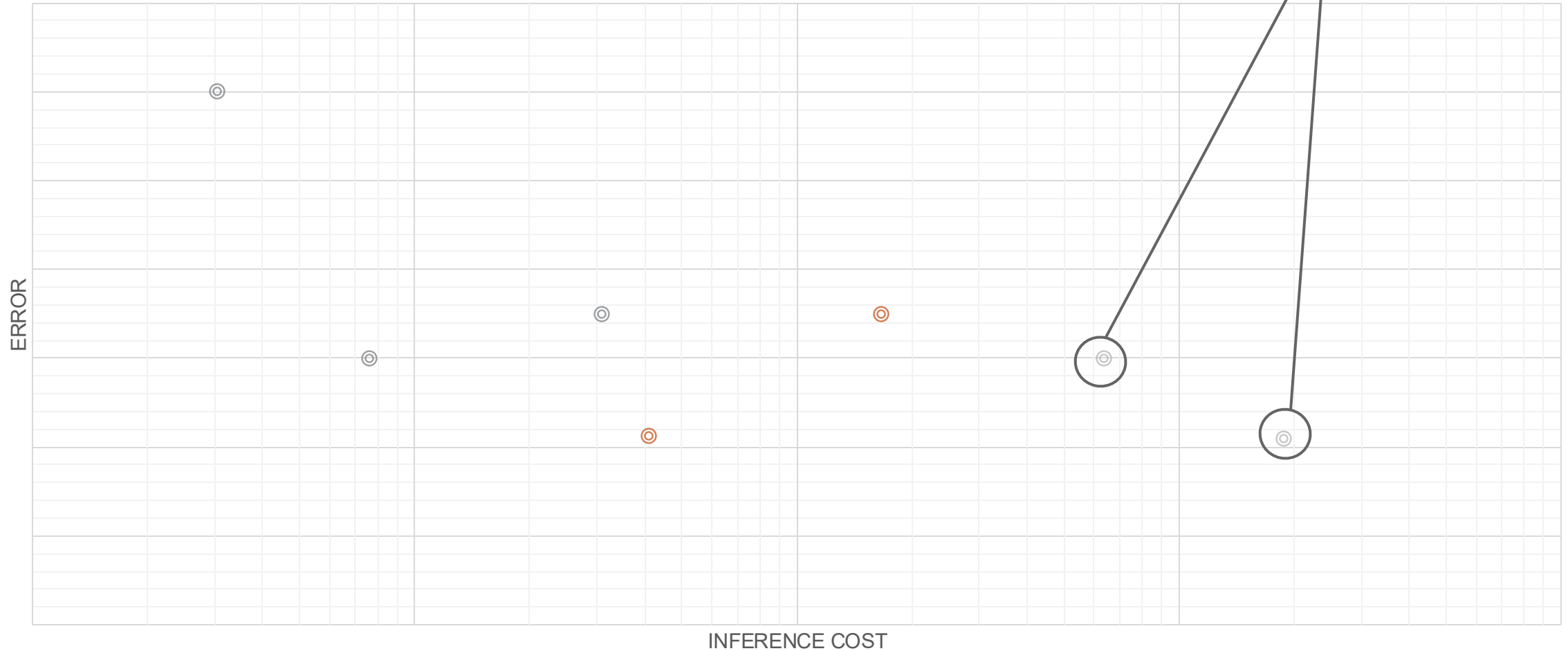
⊙ Float ⊙ 8-bit ⊙ Reduced Precision



# Design Trade-offs with Quantization + Sparsity

Error vs Compute Cost

○ Float   ○ 8-bit   ○ Reduced Precision



Different network topologies

# Design Trade-offs with Quantization + Sparsity

Error vs Compute Cost

⊙ Float ⊙ 8-bit ⊙ Reduced Precision



Floating point networks

# Design Trade-offs with Quantization + Sparsity

Error vs Compute Cost

⊙ Float ⊙ 8-bit ⊙ Reduced Precision



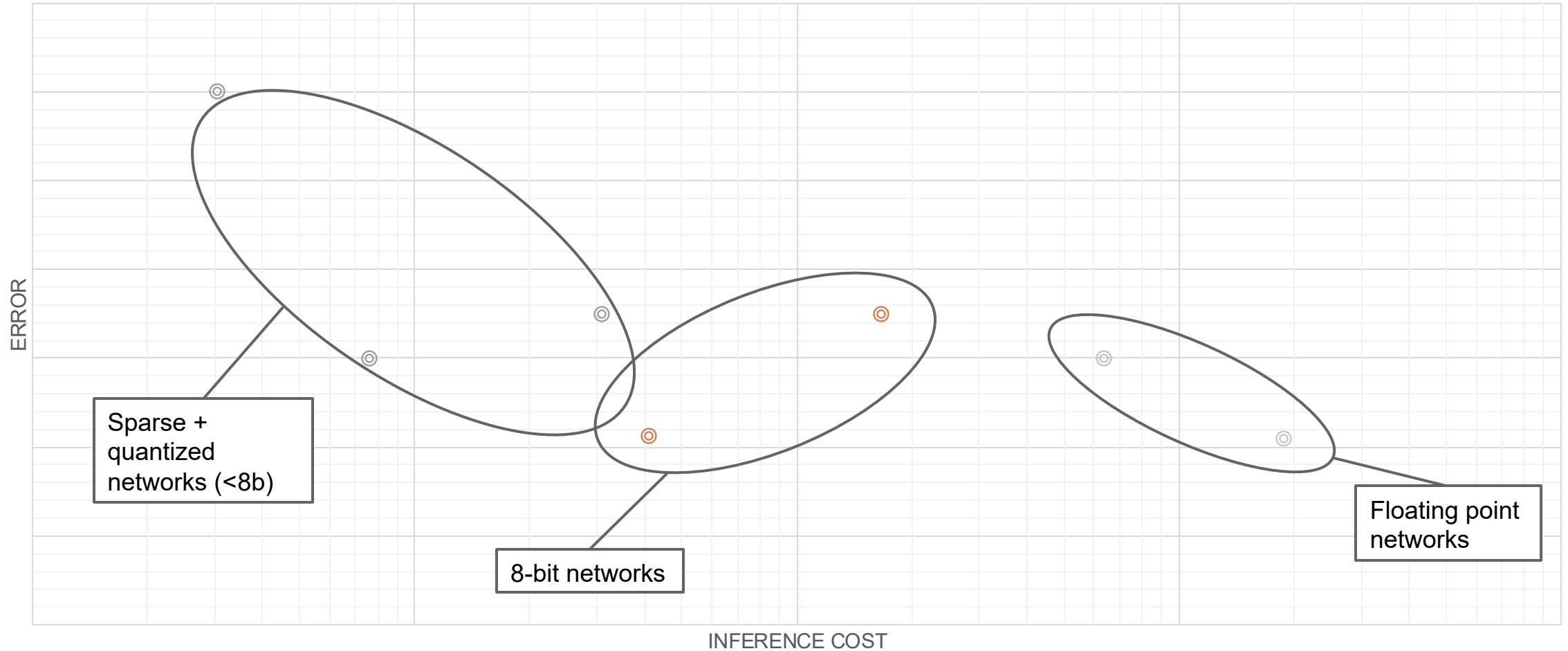
Floating point networks

8-bit networks

# Design Trade-offs with Quantization + Sparsity

Error vs Compute Cost

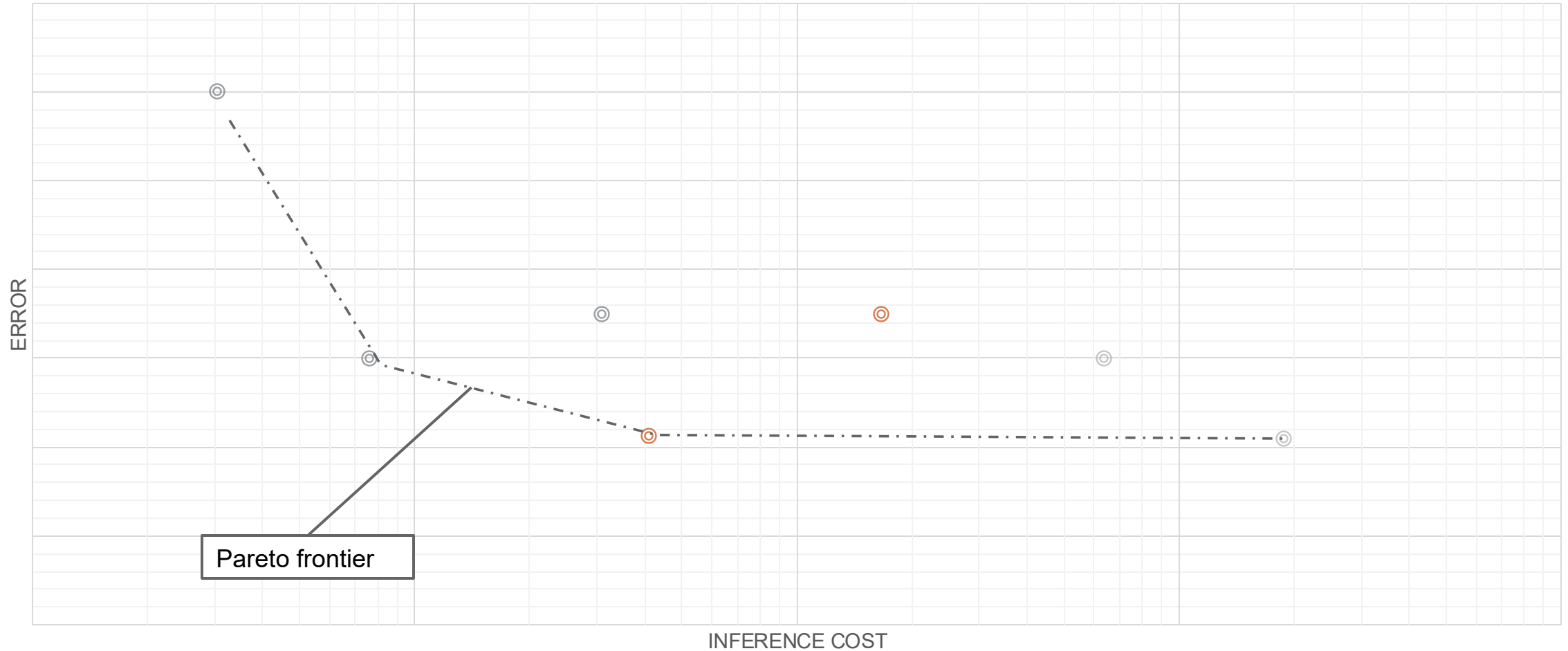
⊙ Float ⊙ 8-bit ⊙ Reduced Precision



# Design Trade-offs with Quantization + Sparsity

Error vs Compute Cost

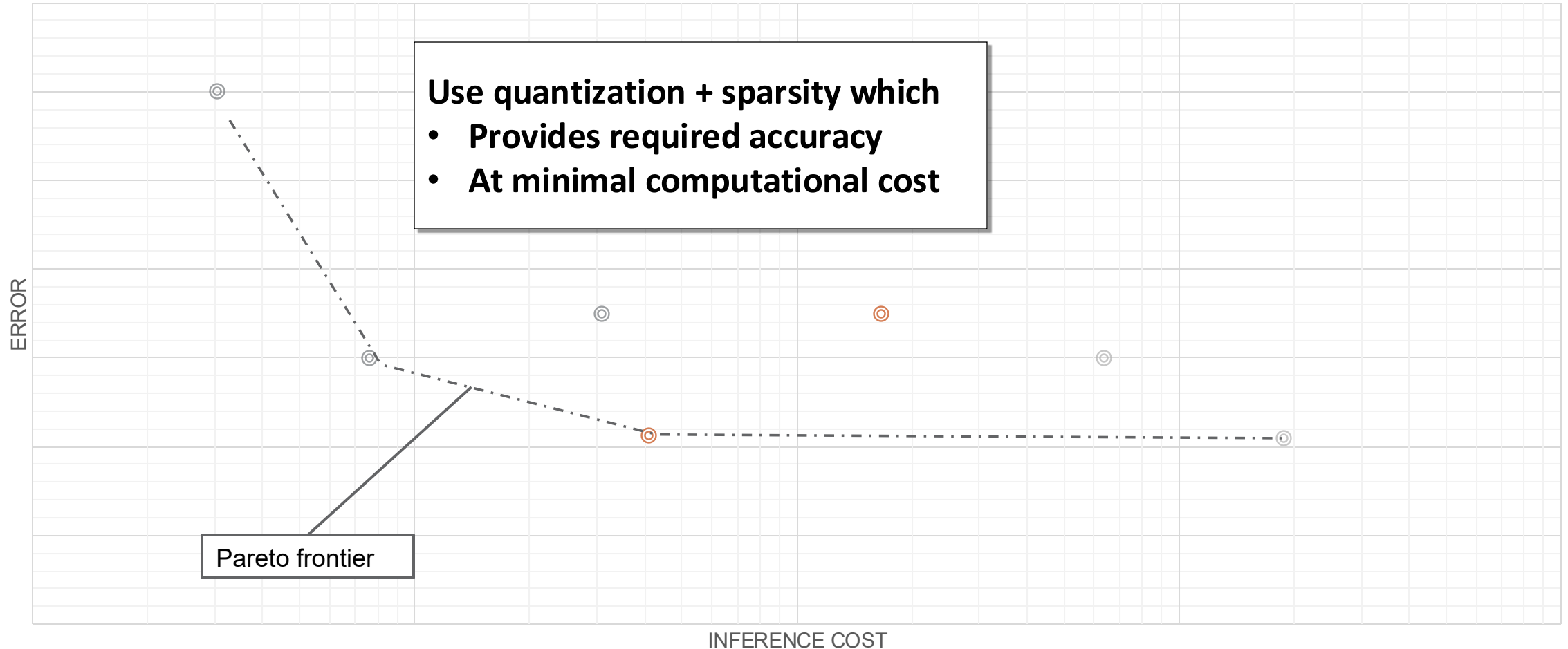
○ Float   ○ 8-bit   ○ Reduced Precision



# Design Trade-offs with Quantization + Sparsity

Error vs Compute Cost

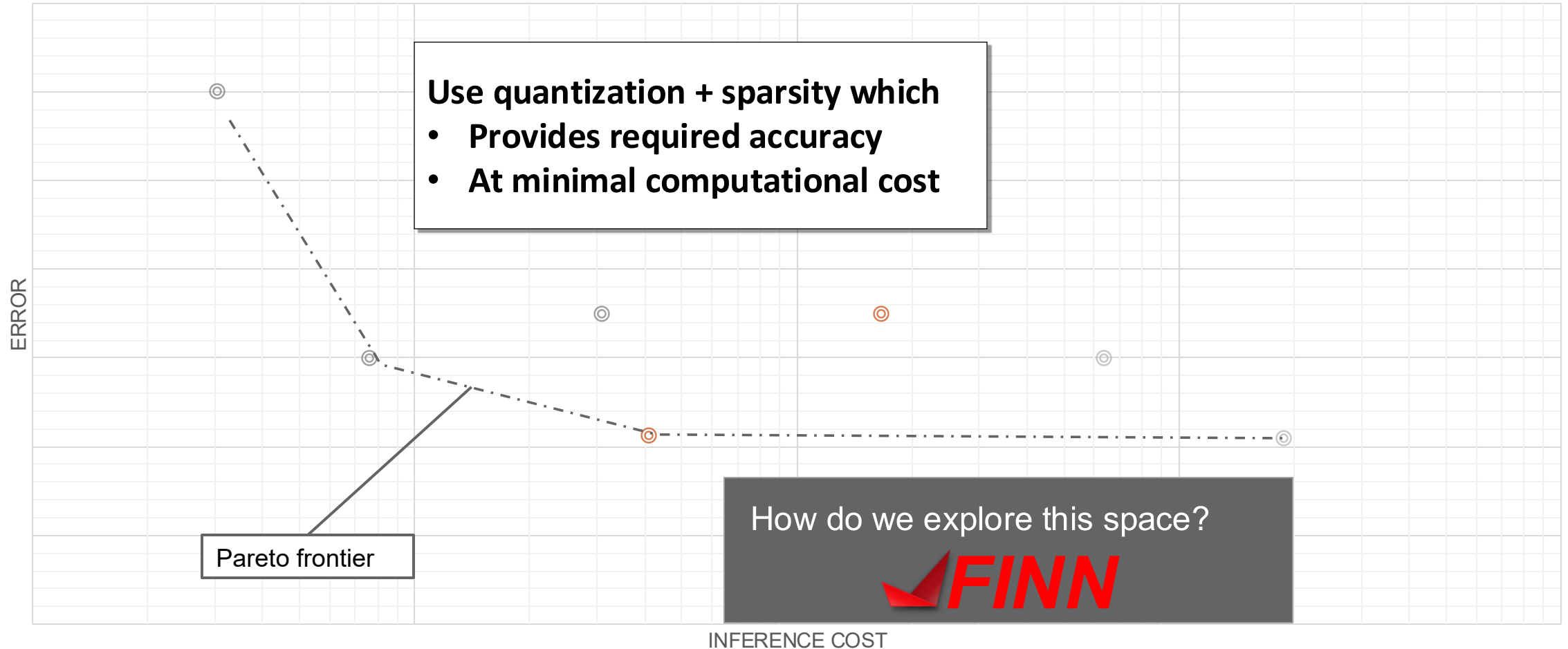
⊙ Float ⊙ 8-bit ⊙ Reduced Precision



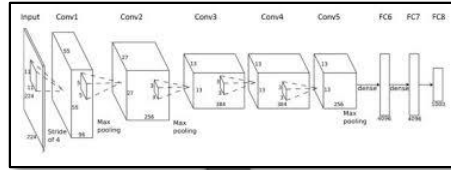
# Design Trade-offs with Quantization + Sparsity

Error vs Compute Cost

⊙ Float ⊙ 8-bit ⊙ Reduced Precision



# FINN Framework: From DNN to FPGA Deployment



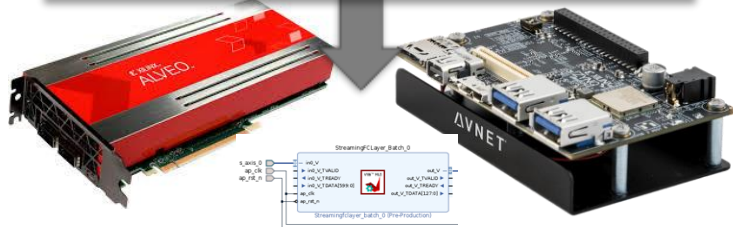
other quantizers  
(QKeras, HAWQ)

**Brevitas**  
Training in PyTorch  
Algorithmic optimizations

QONNX

**FINN Compiler**  
QNN-to-accelerator

**Deployment**  
Verification, integration...



- Quantization-aware training for DNNs
- Library of common quantized layers
- Includes pre-trained examples
  
- Quantized NN exchange format + toolkit
  
- Perform optimizations
- Assemble parameterized HLS/RTL modules
- Generate a DNN hardware IP
  
- Run RTL testbenches to simulate IP
- System-level integration in Vivado IPI
- Rapid prototyping with PYNQ

All open source & actively maintained

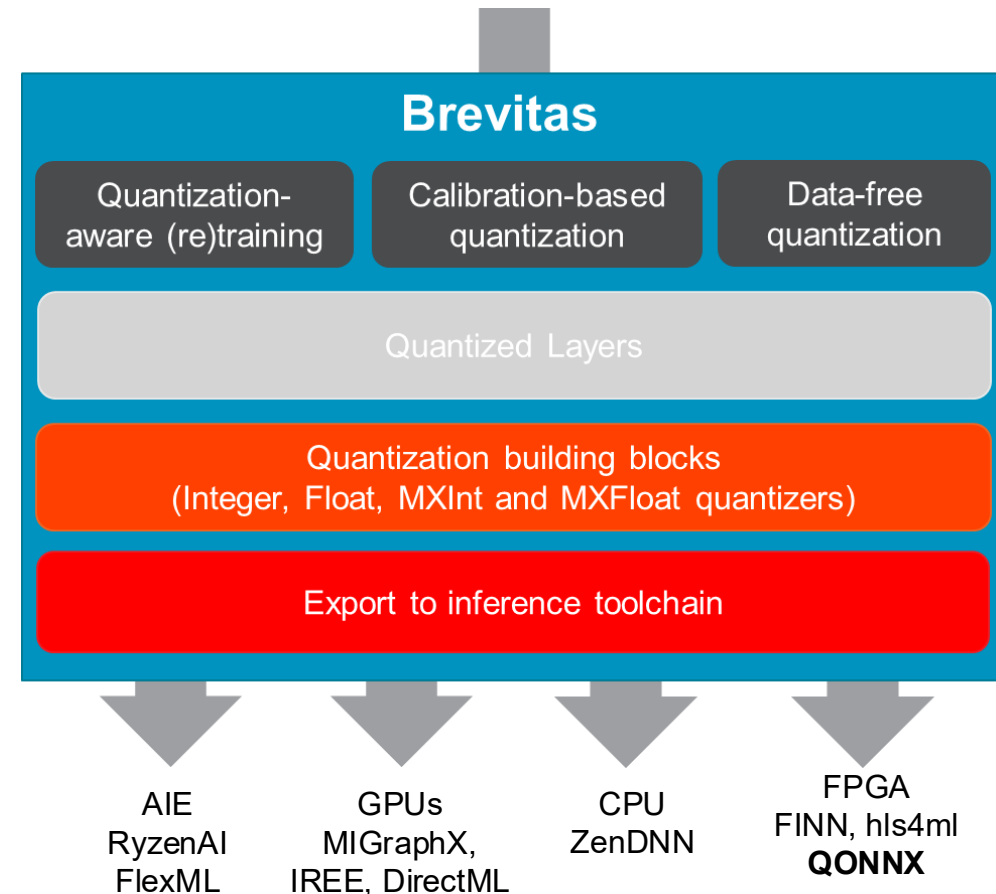
# Brevitas - PyTorch Library for Agile Quantization

- **First class support for custom datatypes and operators at ML framework level**
  - Arbitrary precision integer, float, incl. FP8, block-style (MX)
  - Extendible to user-defined datatypes, operators and support at training time
- **Composable building blocks at multiple abstraction levels that can be arbitrarily combined**
  - Easy to broaden/scale model scope
  - Supports PTQ & QAT
    - GPxQ, SmoothQuant, Bias Correction, Weight Equalization,...
- **Enabling advanced quantization research**
  - Recent highlight: Post-Training Model Expansion [[arXiv:2503.17513](https://arxiv.org/abs/2503.17513)]
- **Hardware independent through diverse export flows**

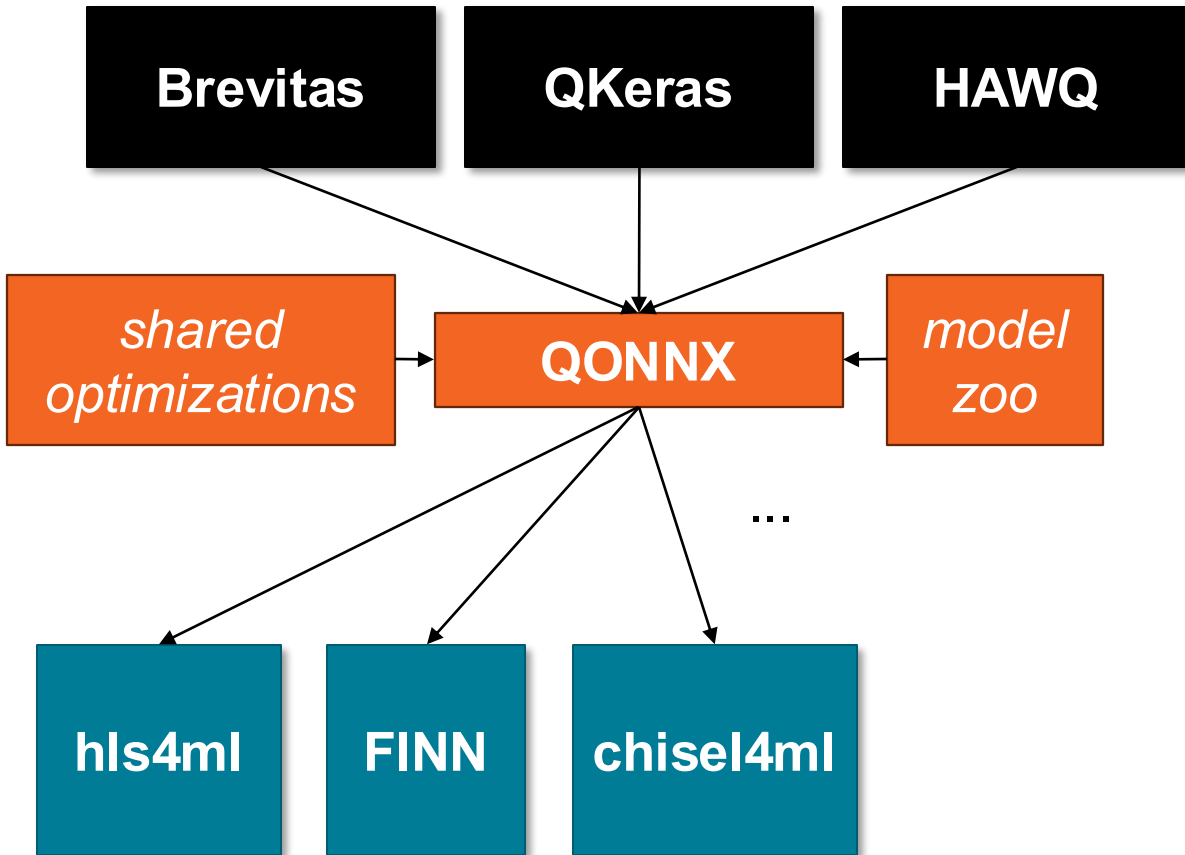
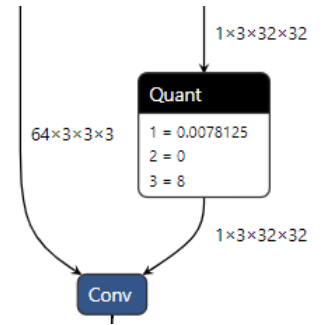
<https://github.com/Xilinx/brevitas>



Hugging Face

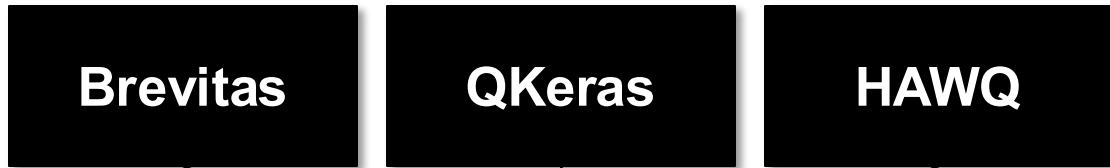
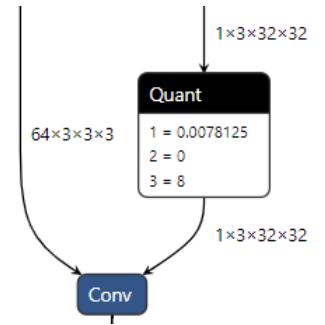


# QONNX: Flexible quantized NNs in ONNX + related tools

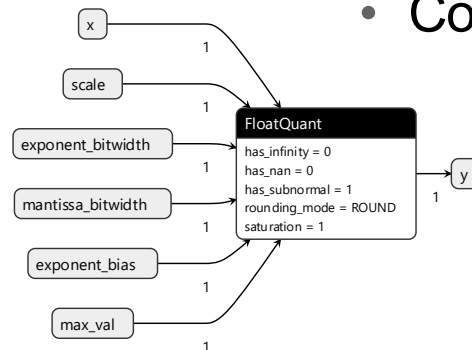


- Custom ONNX ops to represent **arbitrary-bit integer and float quantization**
- ONNX-based common exchange format for QNNs
  - Meeting point between quantization frameworks and backends
  - Own «model zoo» of quantized models
  - Set of shared optimizations
- Infrastructure for manipulating + verifying custom ONNX graphs
- Co-maintained by AMD RAD & FastML

# QONNX: Flexible quantized NNs in ONNX + related tools



- Custom ONNX ops to represent **arbitrary-bit integer and float quantization**
- ONNX-based common exchange format for QNNs
  - Meeting point between quantization frameworks and backends
  - Own «model zoo» of quantized models
  - Set of shared optimizations
- Infrastructure for manipulating + verifying custom ONNX graphs
- Co-maintained by AMD RAD & FastML

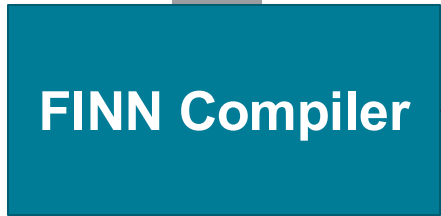


New: flexible minifloat quantization  
See our poster on FloatQuant 😊

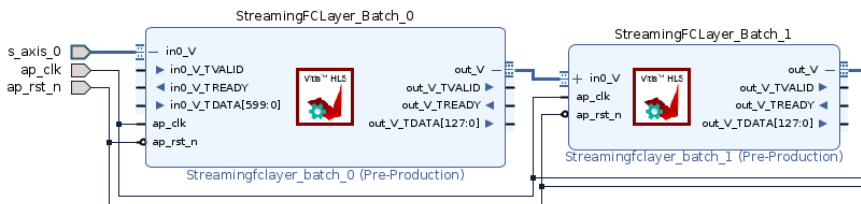
# FINN Compiler: From QONNX to FPGA

<https://github.com/Xilinx/finn>

Quantized NN (QONNX)



Vivado IP



Build configuration

```

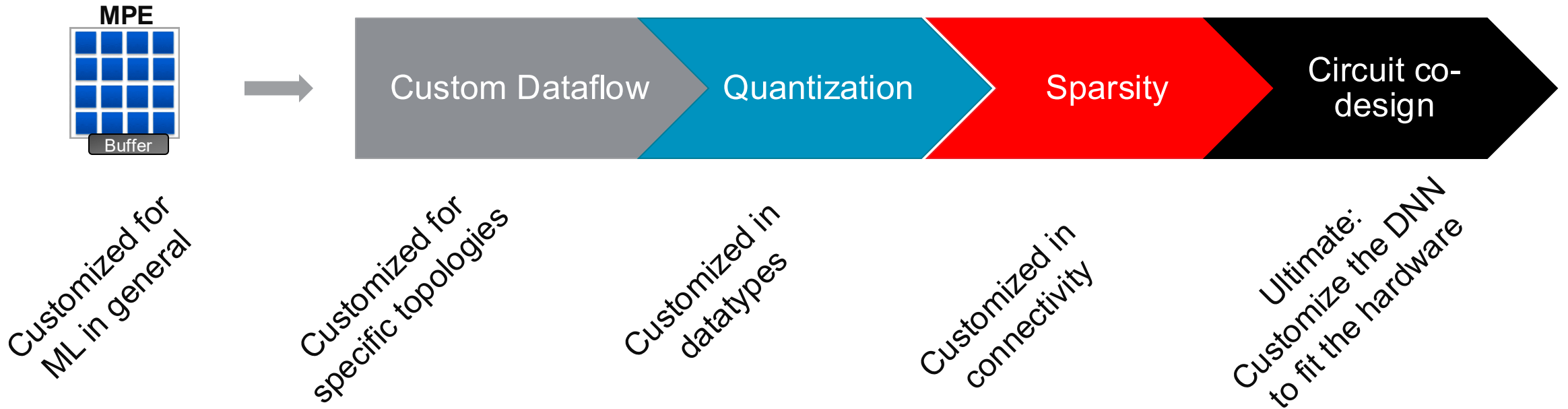
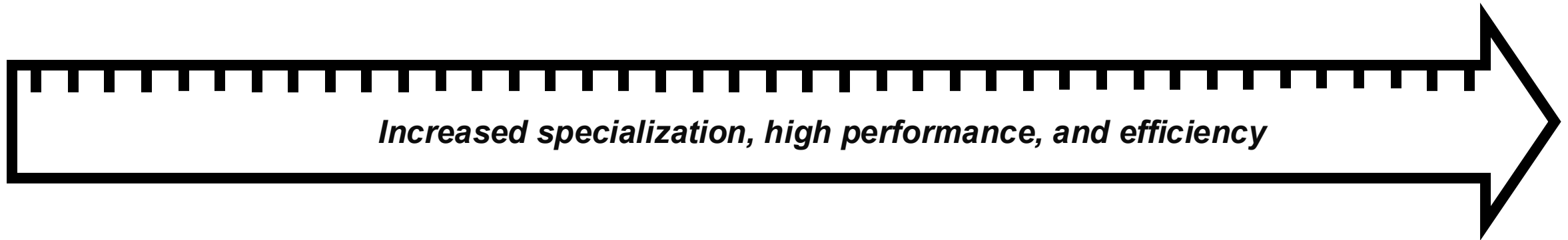
build.DataflowBuildConfig (
  # target performance and clock frequency
  target_fps           = 100 000 000,
  synth_clk_period_ns = 5.0,
  # target FPGA part number (e.g. for ZCU104)
  fpga_part            = "xczu7ev-ffvc1156-2-e",
  # ...
)

```

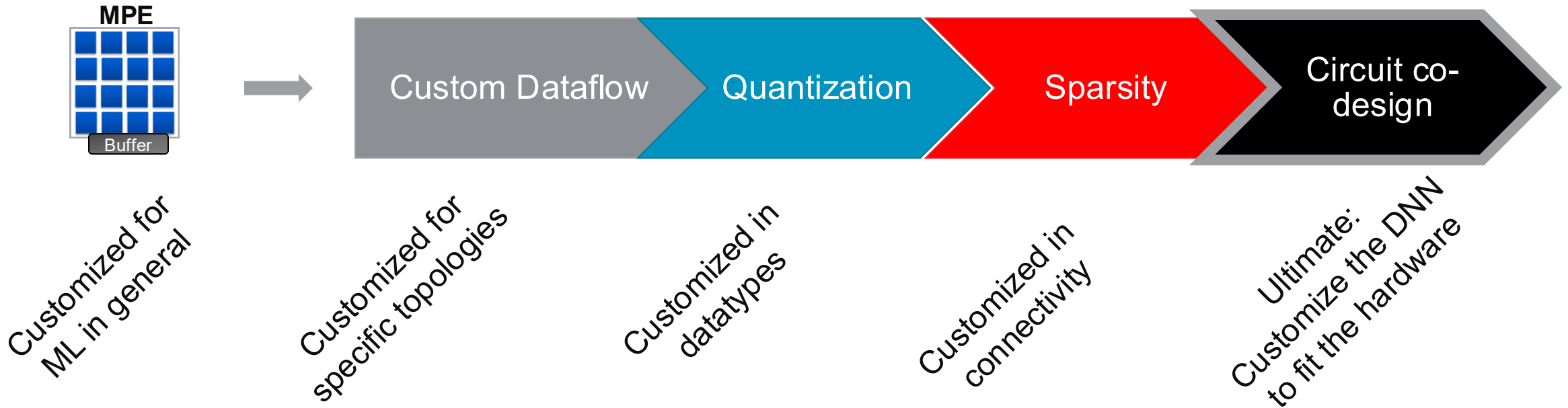
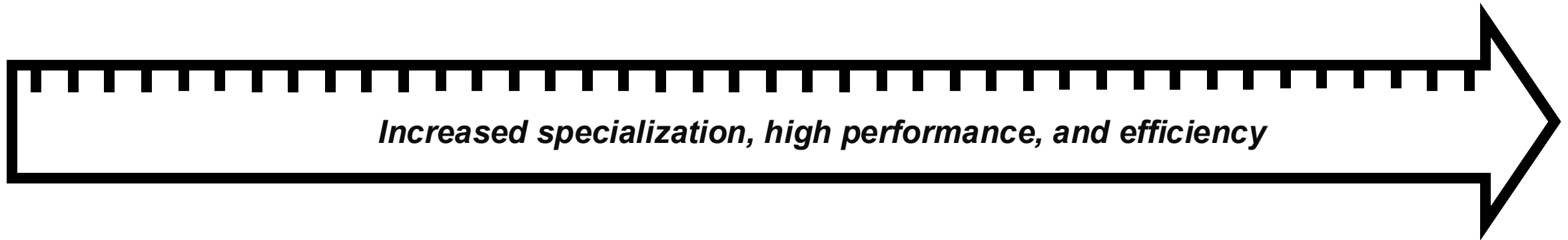
- ▶ Network optimizations: constant folding, streamlining
- ▶ Compute folding with respect to throughput and resource constraints
- ▶ Operator mapping and synthesis (via HLS and RTL op library)
- ▶ Assembly of pipelined dataflow IP with AXI stream interfaces

Key difference to hls4ml:  
performance & efficiency focus -> larger NNs

# Specialized FPGA Inference via Co-Design



# Specialized FPGA Inference via Co-Design



# Bottom-Up: What maps to a 6:1 LUT?



PyTorch

FPGA

# Quantized Neurons as Truth Tables

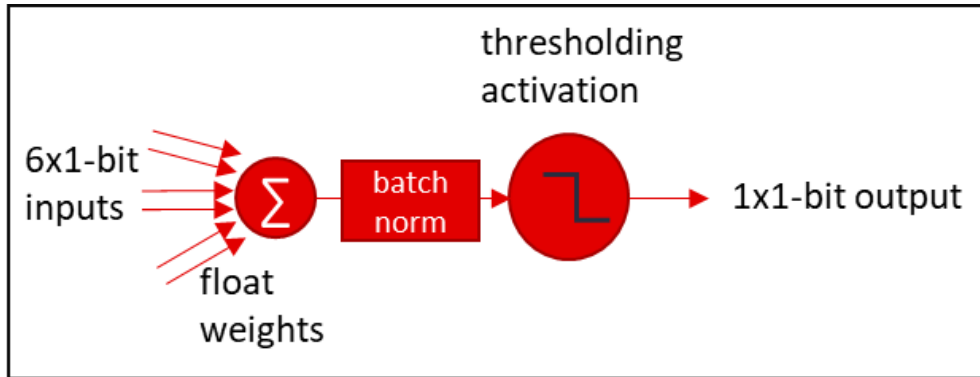
PyTorch

FPGA



Total input: 6 bits  
Total output: 1 bit

# Quantized Neurons as Truth Tables



Total dynamic input *in\_bits*: 6 bits  
Total output *out\_bits*: 1 bit

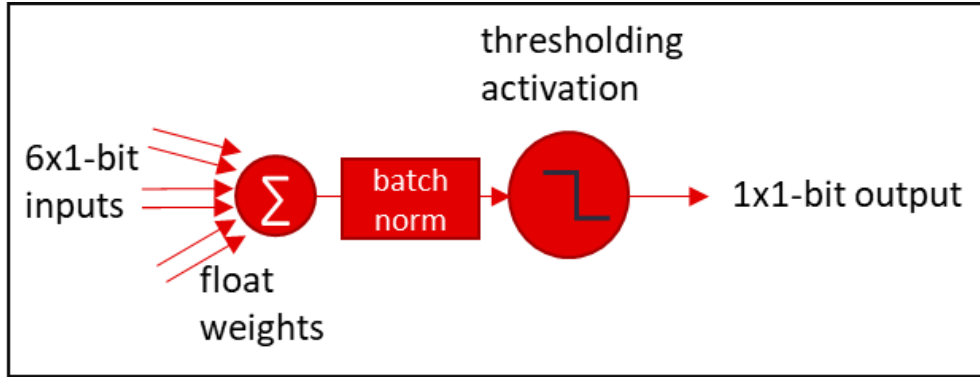
**PyTorch**

**FPGA**



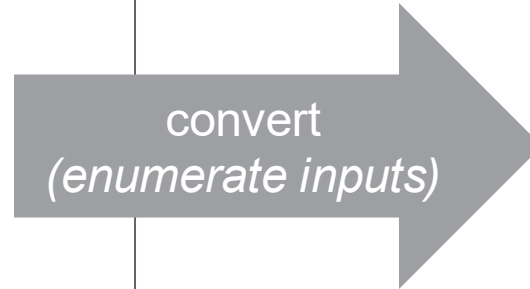
Total input: 6 bits  
Total output: 1 bit

# Quantized Neurons as Truth Tables



Total dynamic input *in\_bits*: 6 bits  
Total output *out\_bits*: 1 bit

**PyTorch**

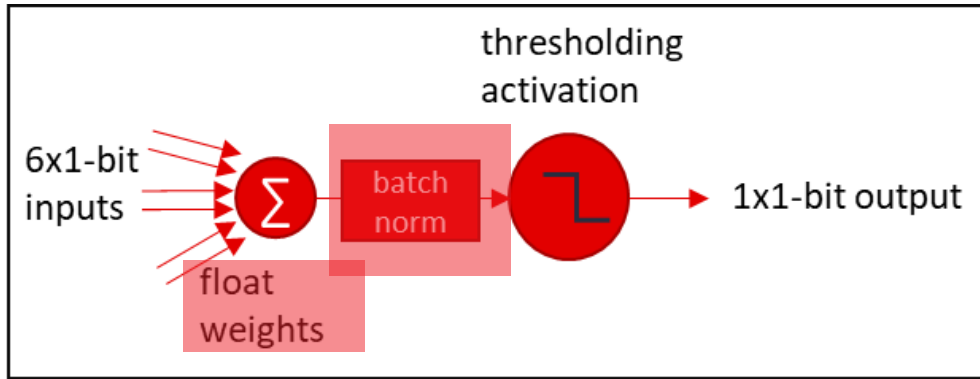


**FPGA**



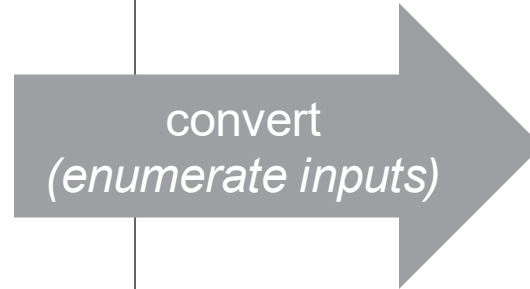
Total input: 6 bits  
Total output: 1 bit

# Quantized Neurons as Truth Tables



Total dynamic input *in\_bits*: 6 bits  
Total output *out\_bits*: 1 bit

**PyTorch**



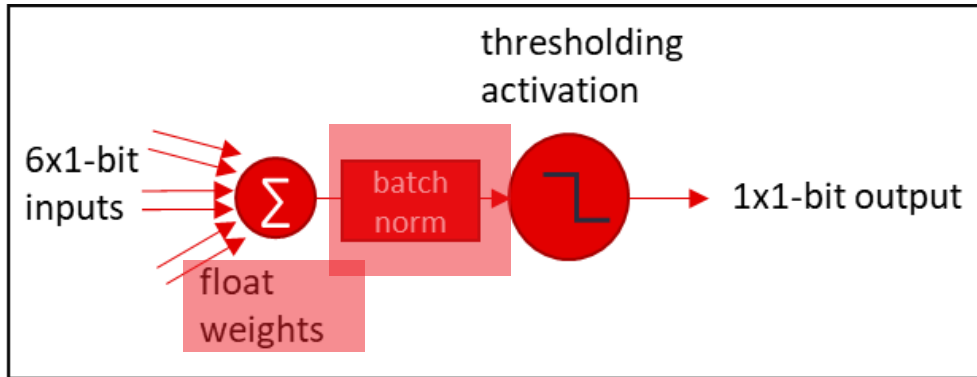
**FPGA**



Total input: 6 bits  
Total output: 1 bit

# Quantized Neurons as Truth Tables

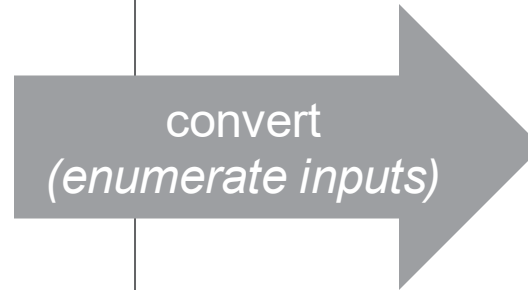
## Neuron Equivalent (NEQ)



Total dynamic input *in\_bits*: 6 bits  
Total output *out\_bits*: 1 bit

**PyTorch**

## Hardware Building Block (HBB)

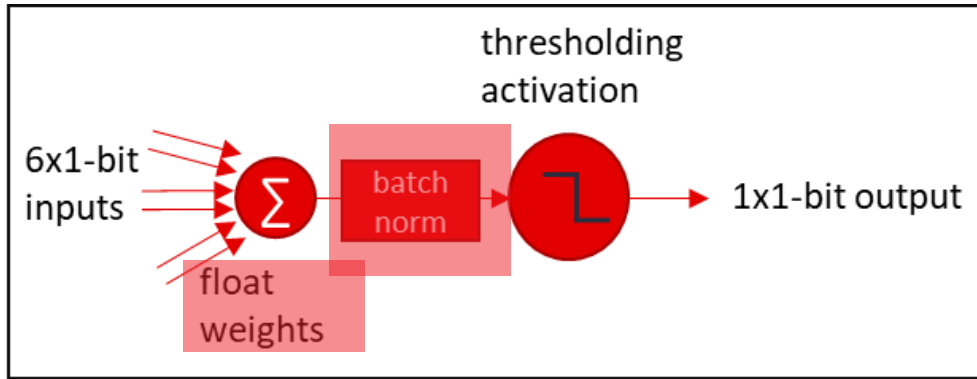


Total input: 6 bits  
Total output: 1 bit

**FPGA**

# Quantized Neurons as Truth Tables

## Neuron Equivalent (NEQ)



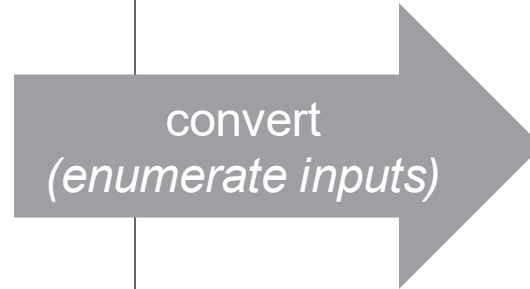
Total dynamic input  $in\_bits$ : 6 bits  
Total output  $out\_bits$ : 1 bit

Hardware cost: 1 x LUT6  
Generalized cost:  $out\_bits \cdot 2^{in\_bits}$

**PyTorch**

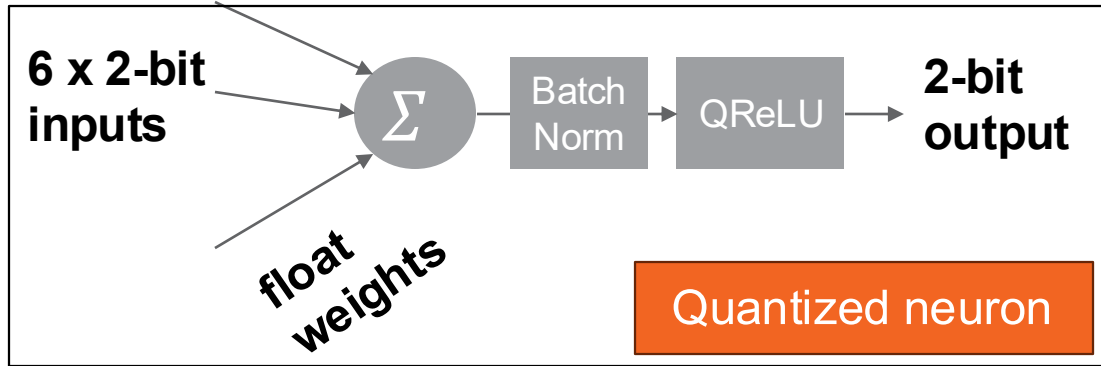
**FPGA**

## Hardware Building Block (HBB)



Total input: 6 bits  
Total output: 1 bit

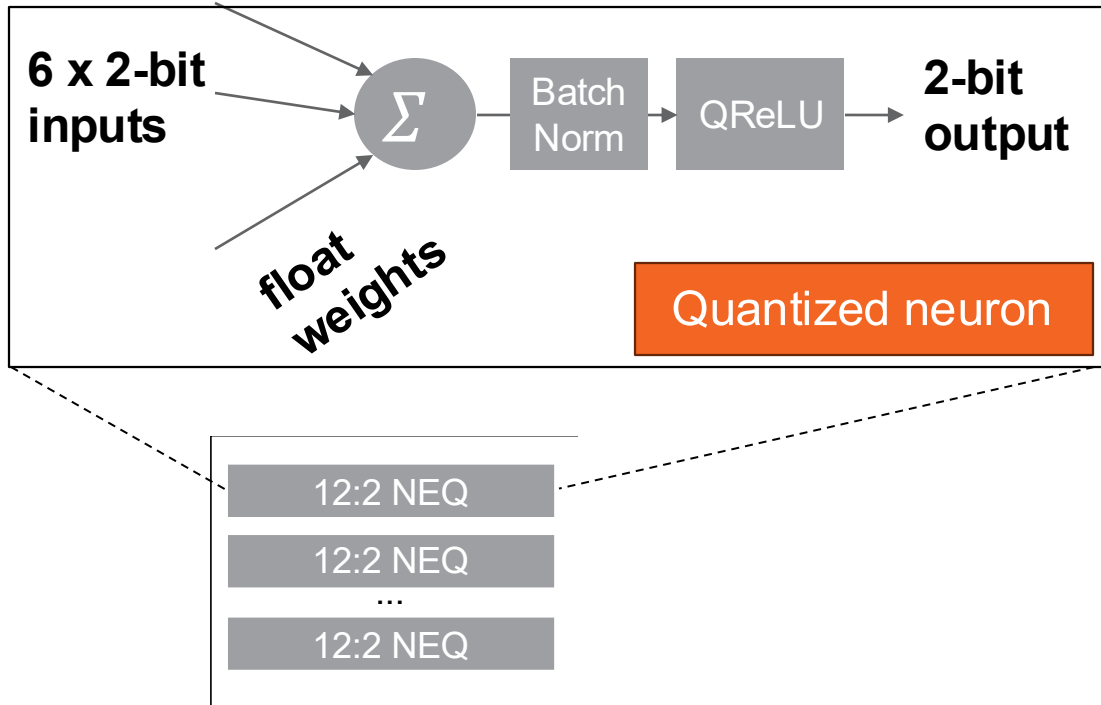
# LogicNets at a Glance



PyTorch

FPGA

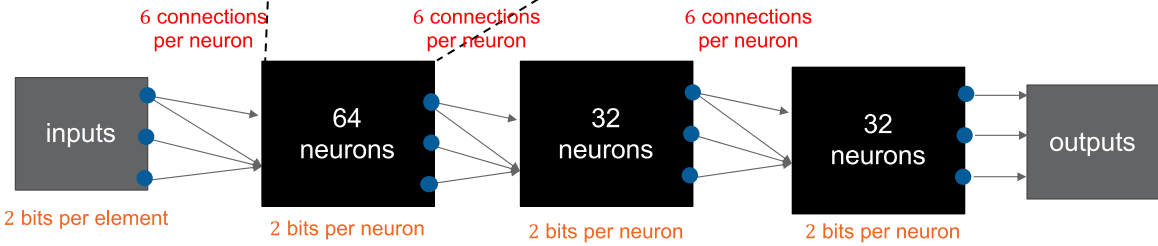
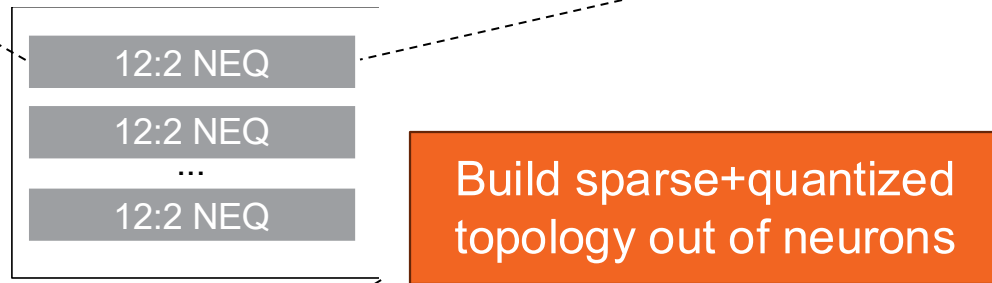
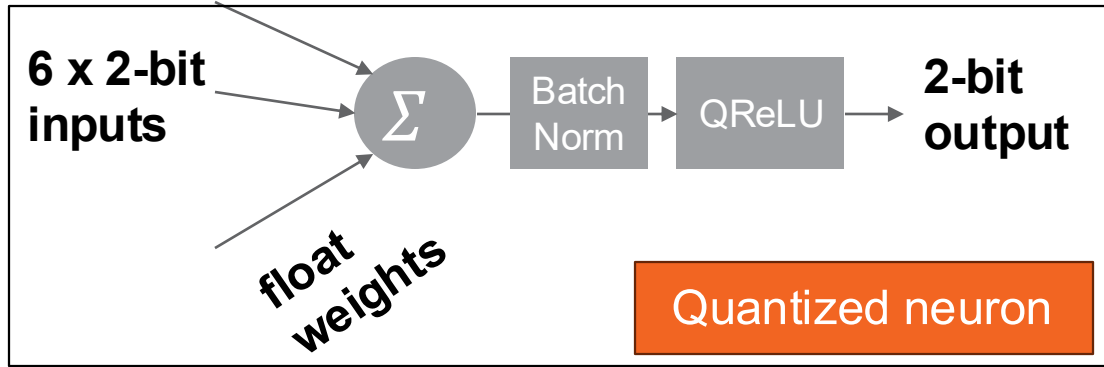
# LogicNets at a Glance



PyTorch

FPGA

# LogicNets at a Glance

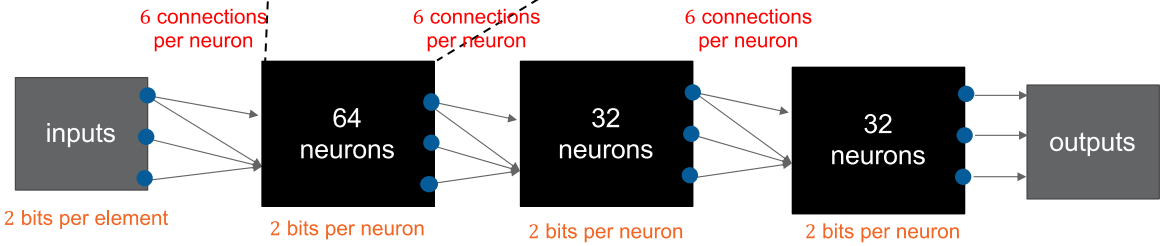
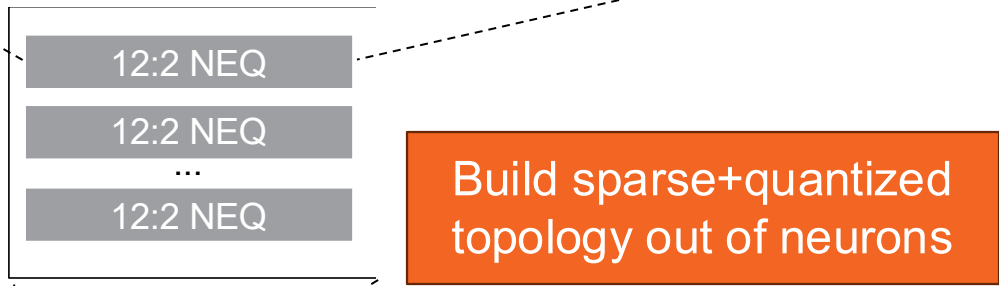
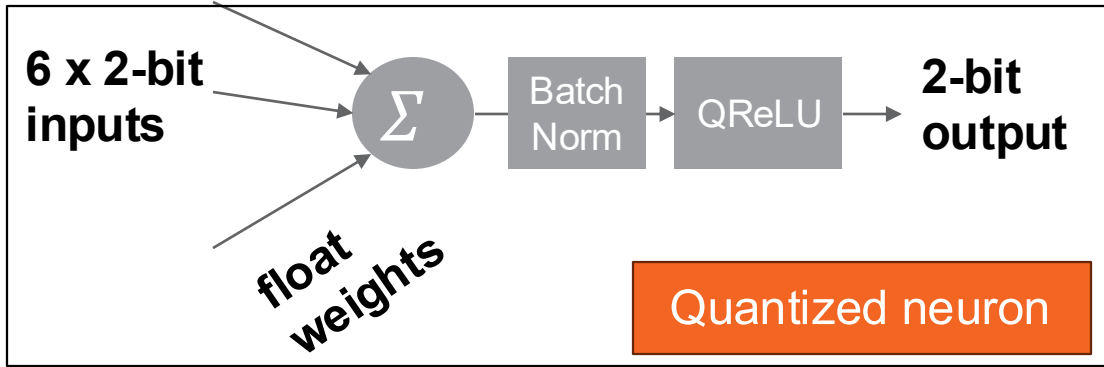


Train in ML framework

PyTorch

FPGA

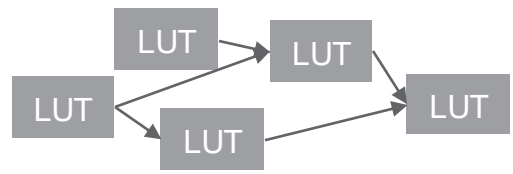
# LogicNets at a Glance



Train in ML framework

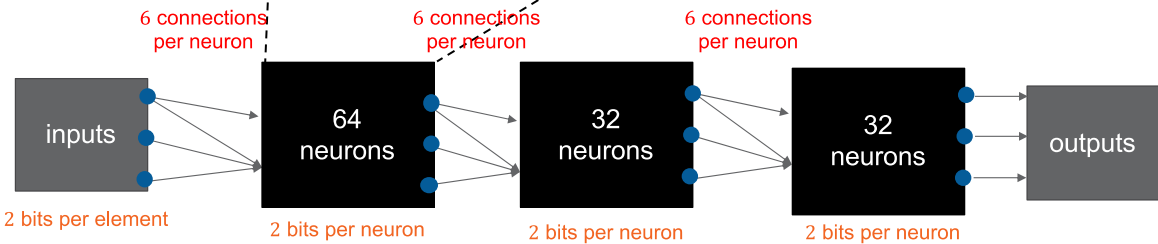
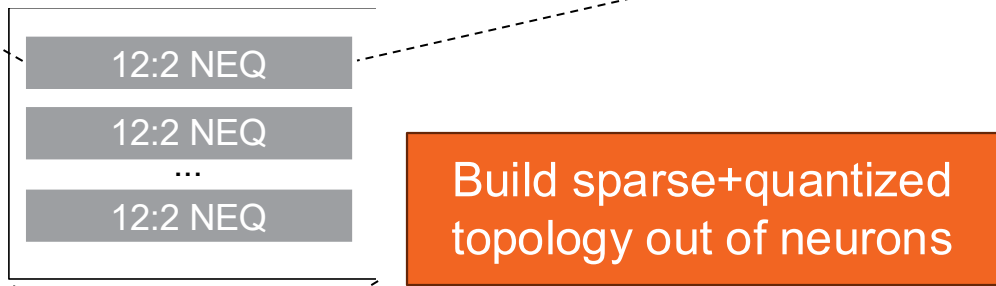
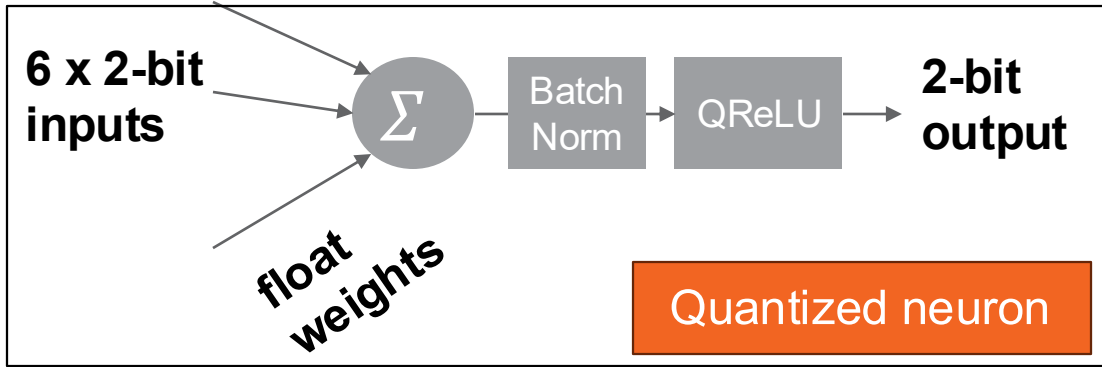
PyTorch

FPGA



Convert to sparse LUT circuit

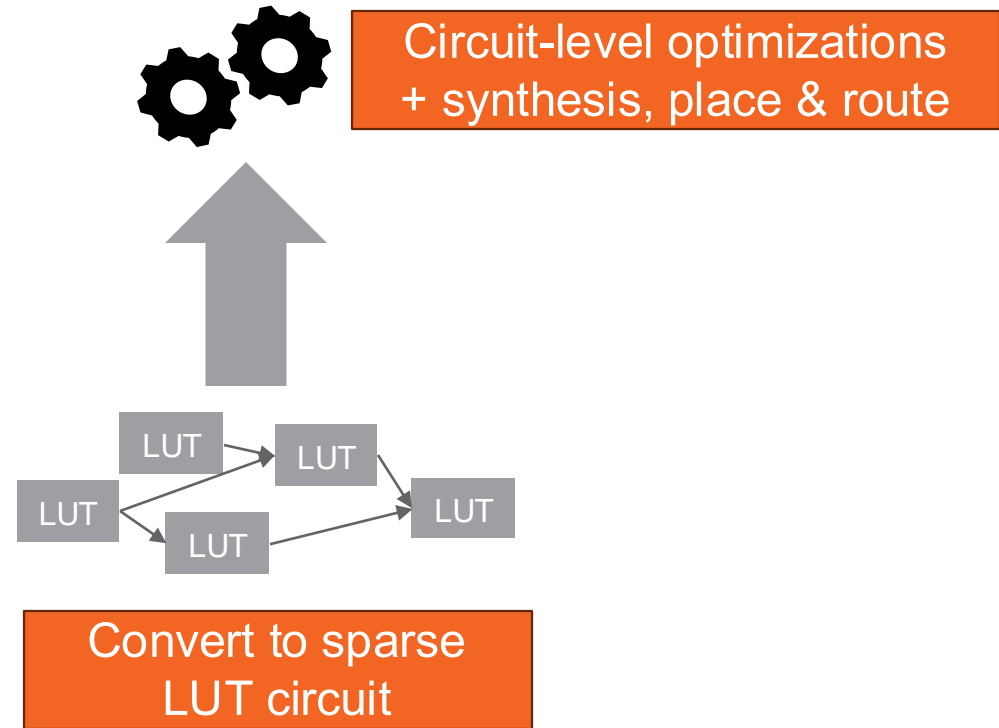
# LogicNets at a Glance



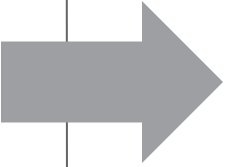
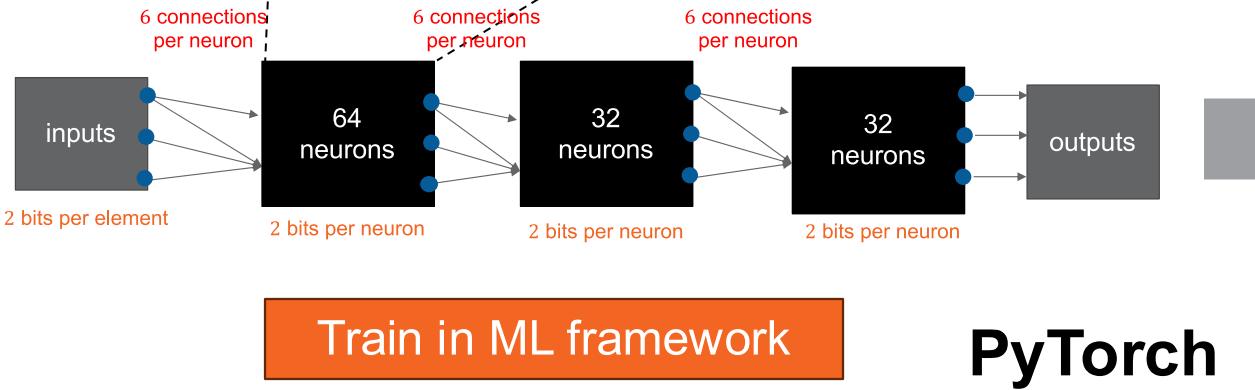
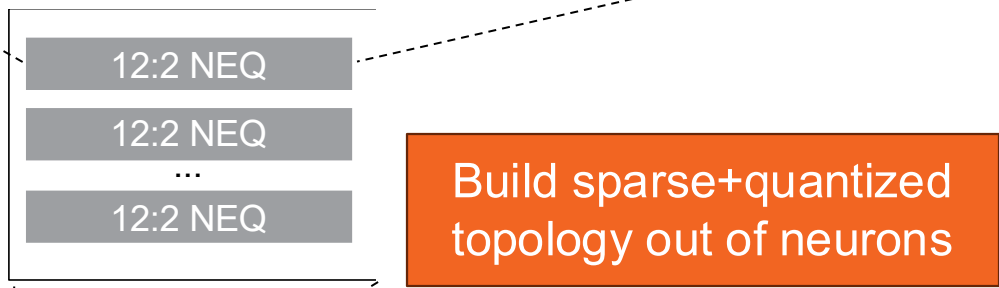
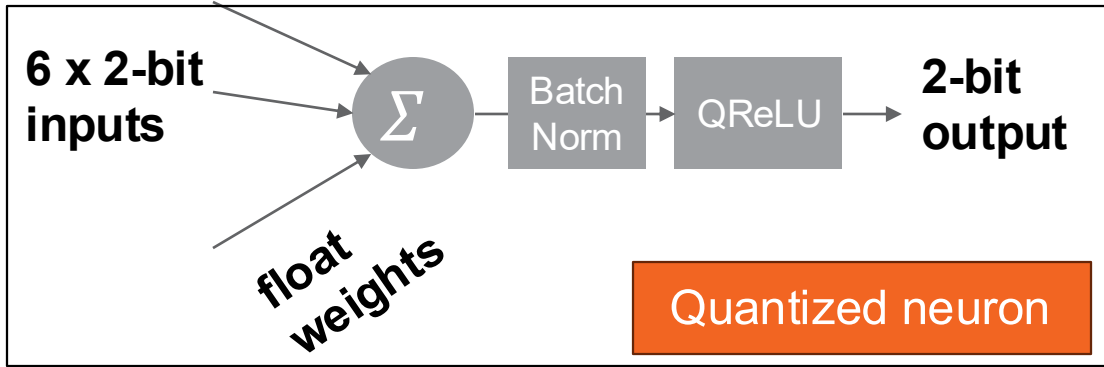
Train in ML framework

PyTorch

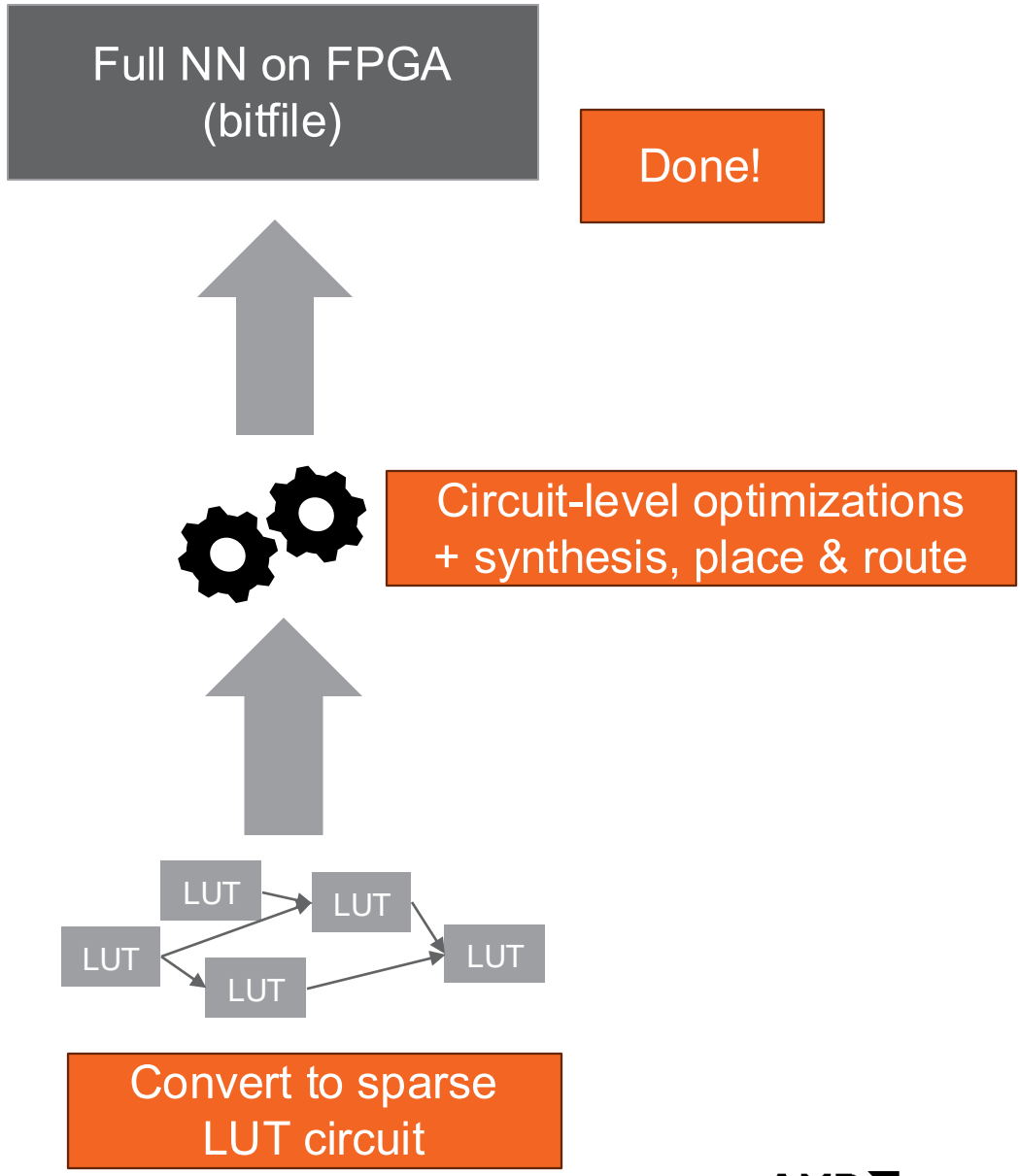
FPGA



# LogicNets at a Glance



FPGA

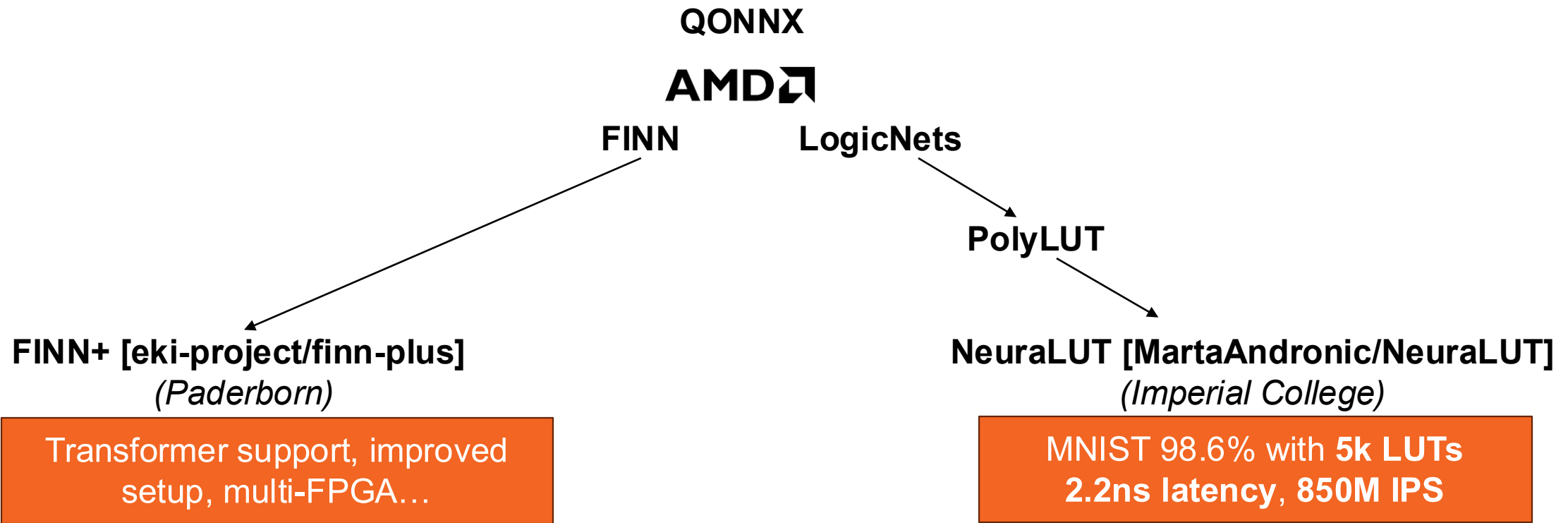


# The Power of the Open Ecosystem: Recent Examples

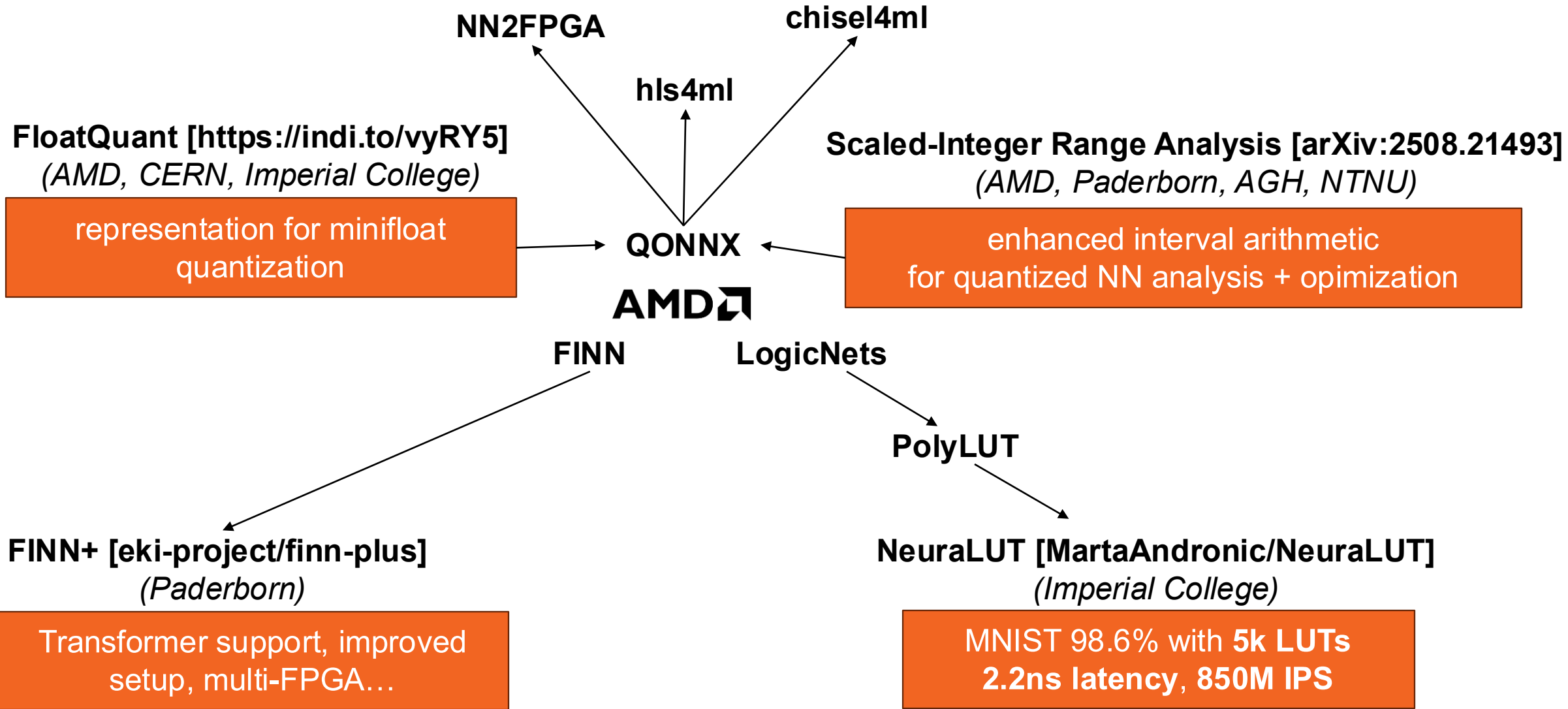




# The Power of the Open Ecosystem: Recent Examples



# The Power of the Open Ecosystem: Recent Examples



# Conclusion



- Co-design of NNs and FPGA HW can yield **orders of magnitude** more efficient inference
  - Combination of streaming dataflow, quantization and sparsity
  - Essential ingredients for the “long tail” of Pervasive AI
- Two key ingredients make NN/FPGA co-design technology accessible
  - **Open-source tools** like Brevitas, FINN, QONNX, hls4ml and LogicNets
  - **Open ecosystem** to build & share the technical expertise

Internships available at AMD RADICAL!  
Talk to me or e-mail your CV: [yamanu@amd.com](mailto:yamanu@amd.com)

# COPYRIGHT AND DISCLAIMER

©2025 Advanced Micro Devices, Inc. All rights reserved.

AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS.' AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

**AMD** 