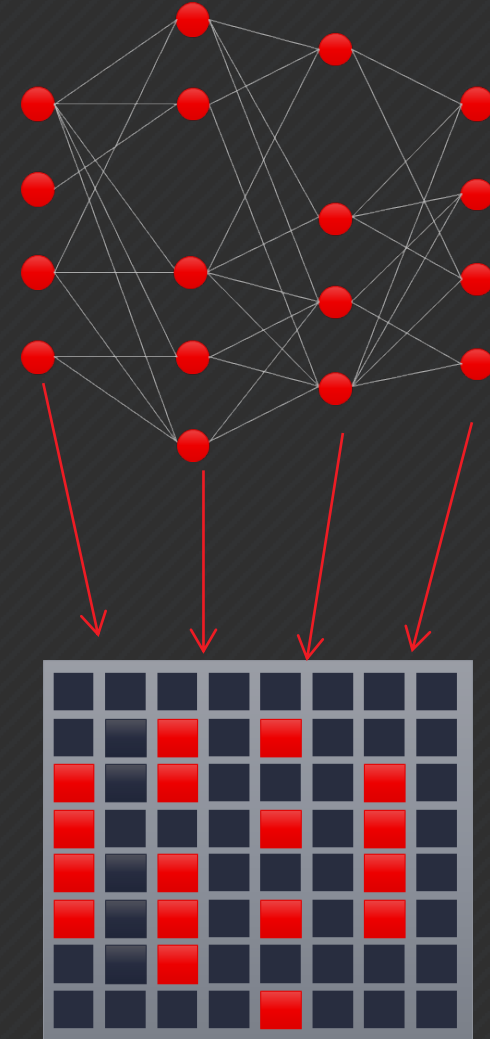




Co-Design for Efficient & Adaptive ML

CERN Seminar, 2024-06-26

Dr. Yaman Umuroğlu
Senior Member of Technical Staff
AMD Research & Advanced Development



AMD Research and Advanced Development (RAD)

- **Integrated Comms and AI Lab (RADICAL)**

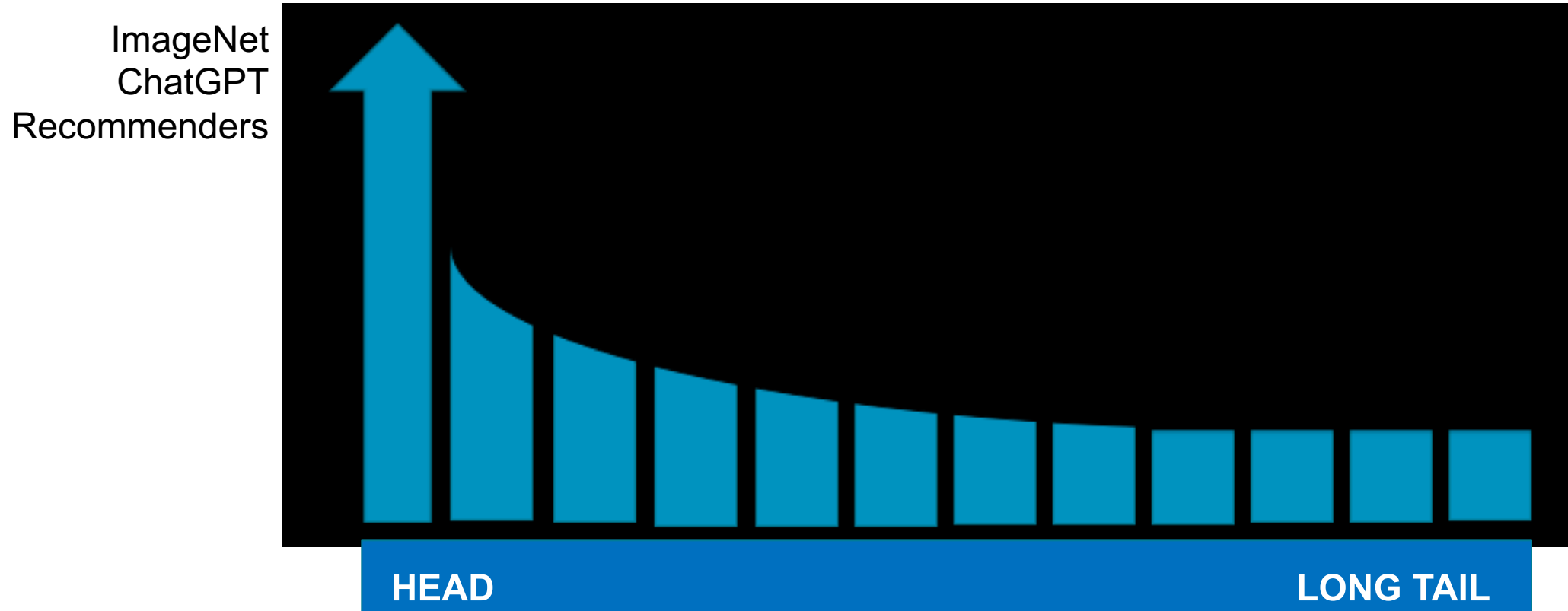
- Established 15 years ago
- ~20 researchers plus university program
 - 5 different locations
- Highly active internship program

- **Focus: Communications and AI**

- Building systems, architectural exploration, algorithmic optimizations, benchmarking
- In collaboration with partners, customers, and universities
 - ETH Zürich, Paderborn University, Imperial College, KIT, NTNU, Politecnico di Milano, NUS, University of Sydney

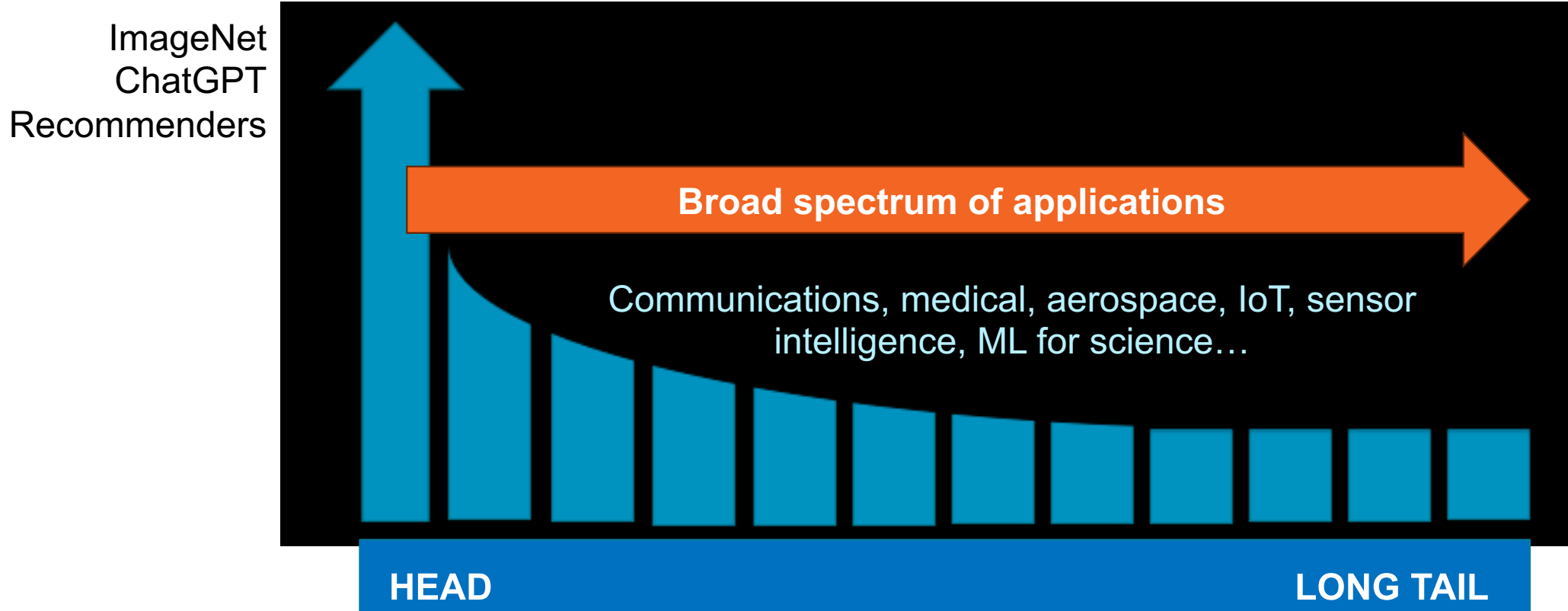


Pervasive AI



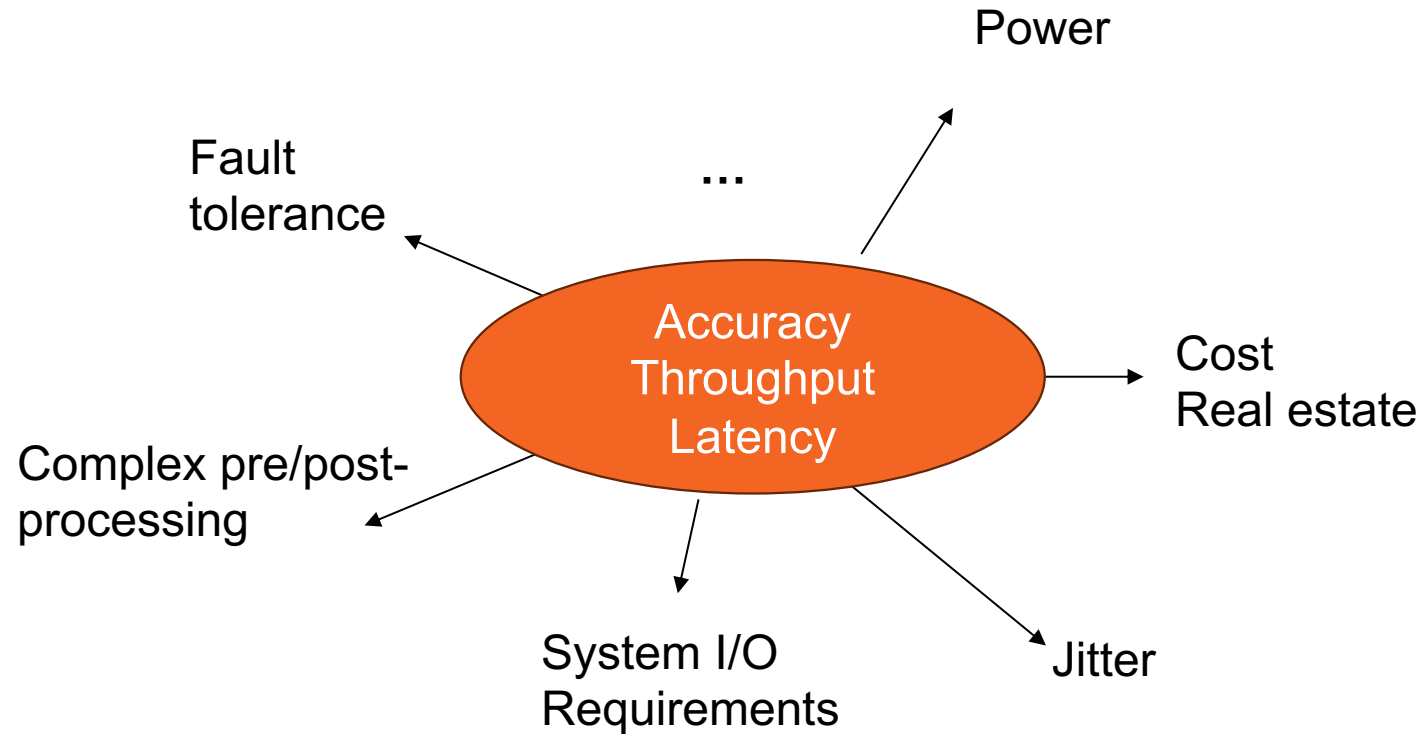
Adapted from TED Talk: Andrew Ng "How AI could empower any business"

Pervasive AI

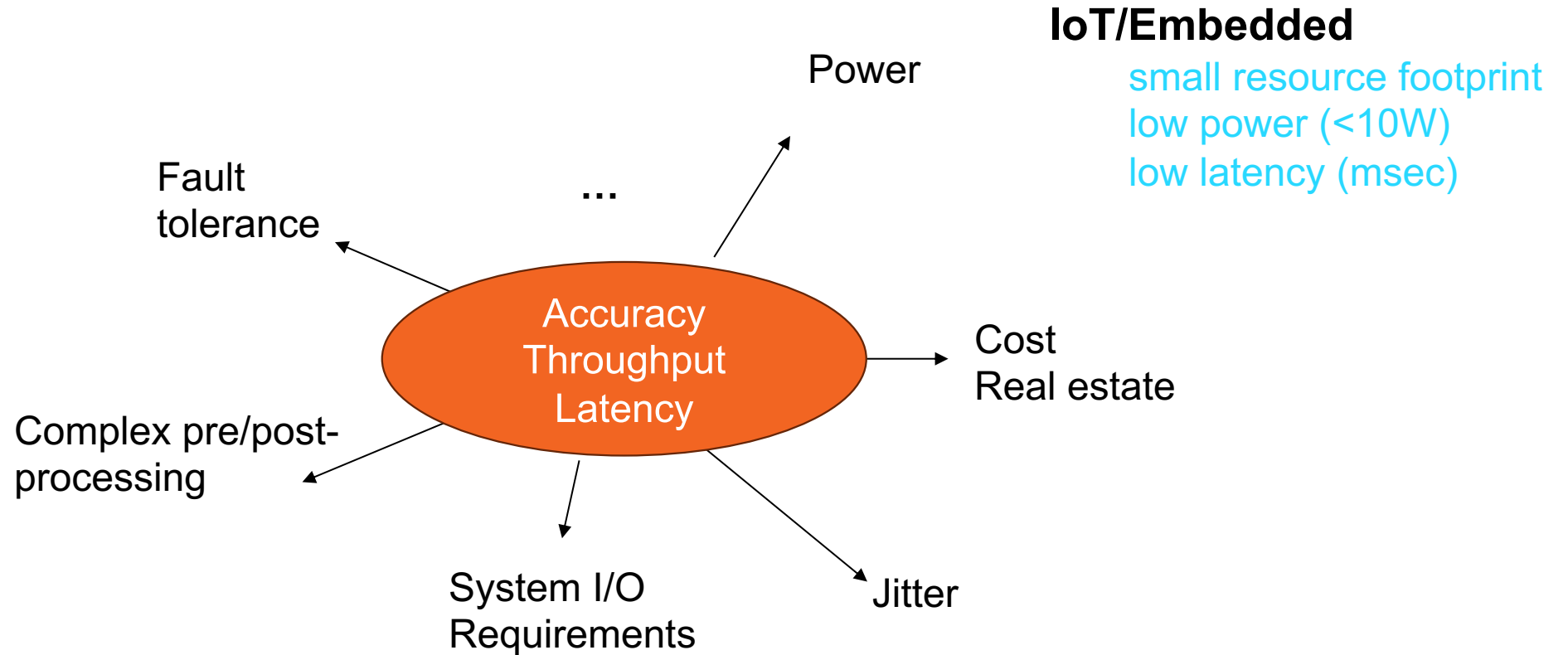


Adapted from TED Talk: Andrew Ng "How AI could empower any business"

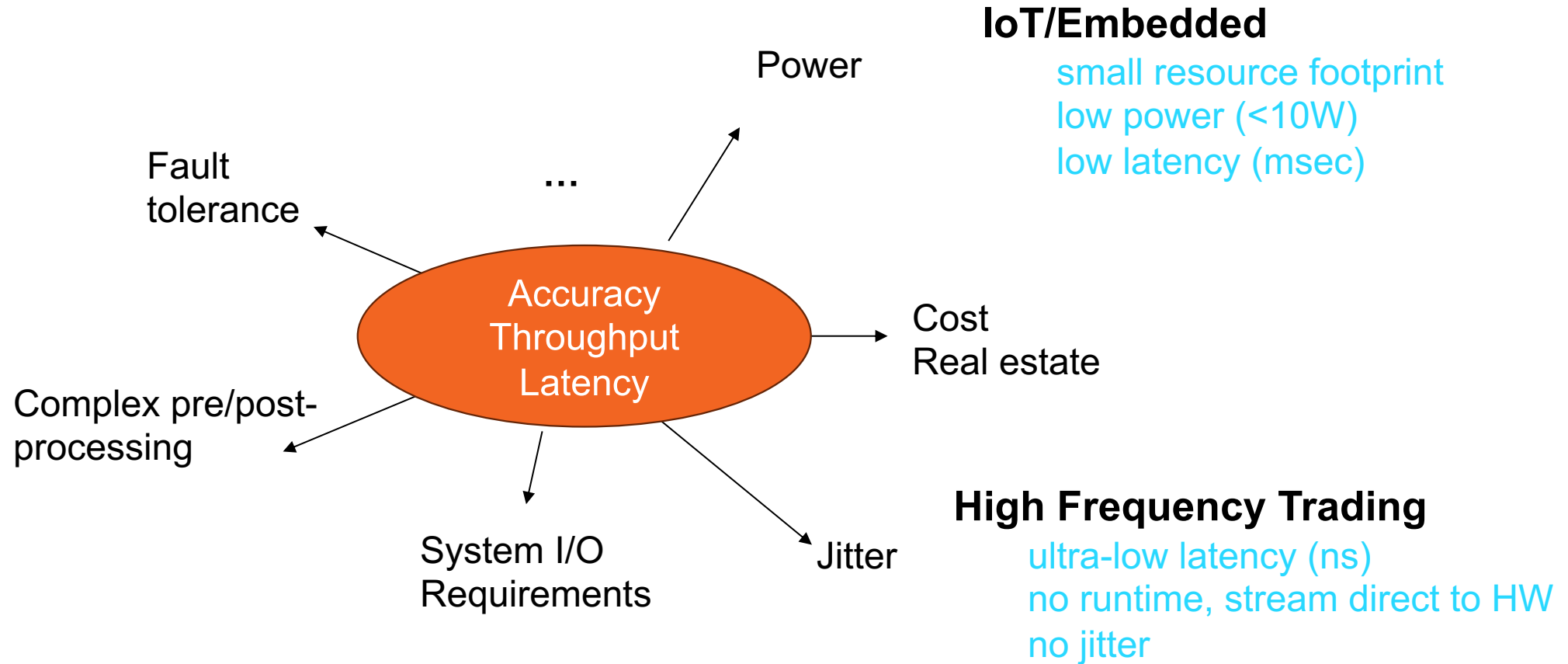
Pervasive AI: Diverse Requirements for Inference



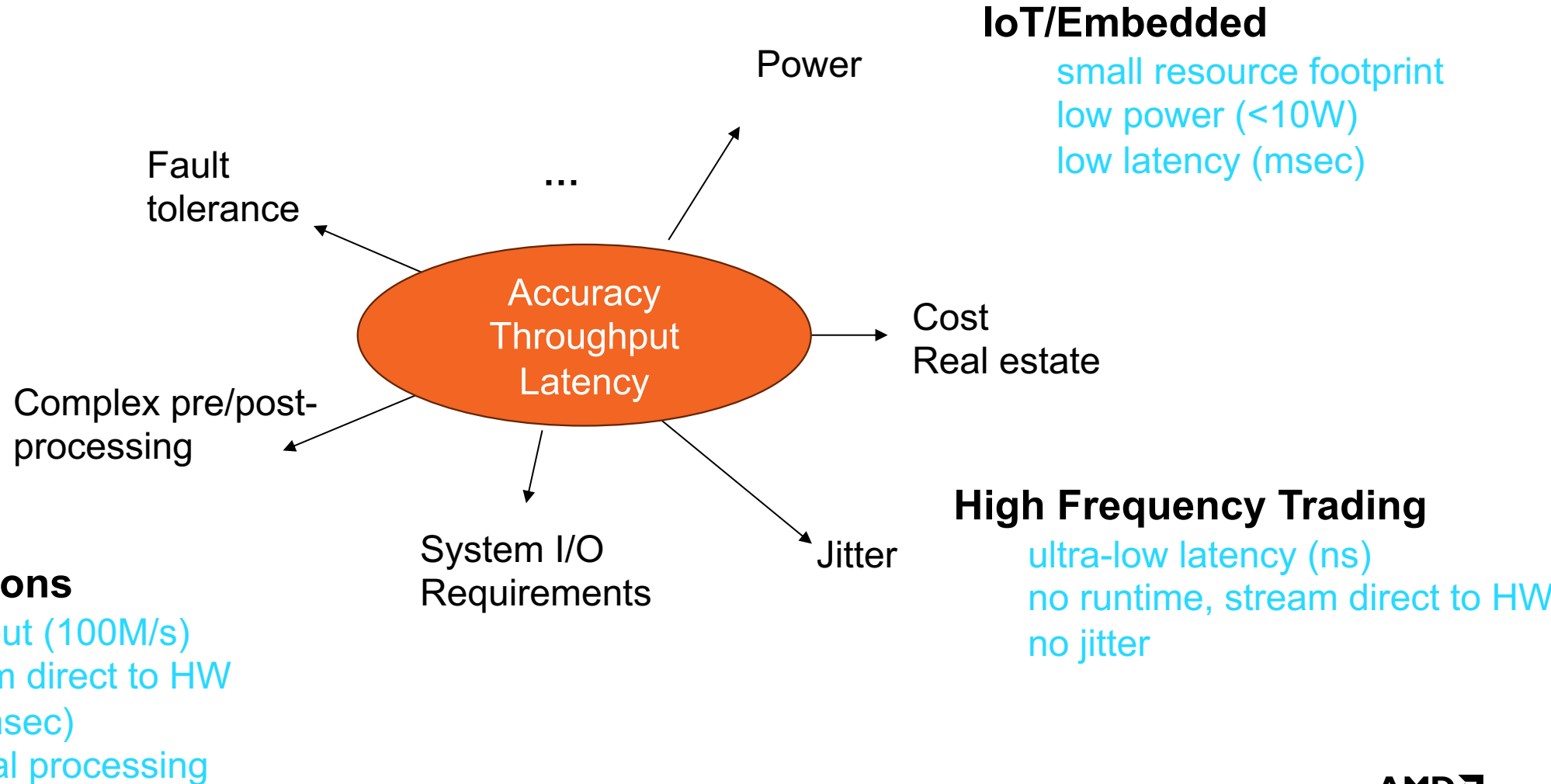
Pervasive AI: Diverse Requirements for Inference



Pervasive AI: Diverse Requirements for Inference



Pervasive AI: Diverse Requirements for Inference



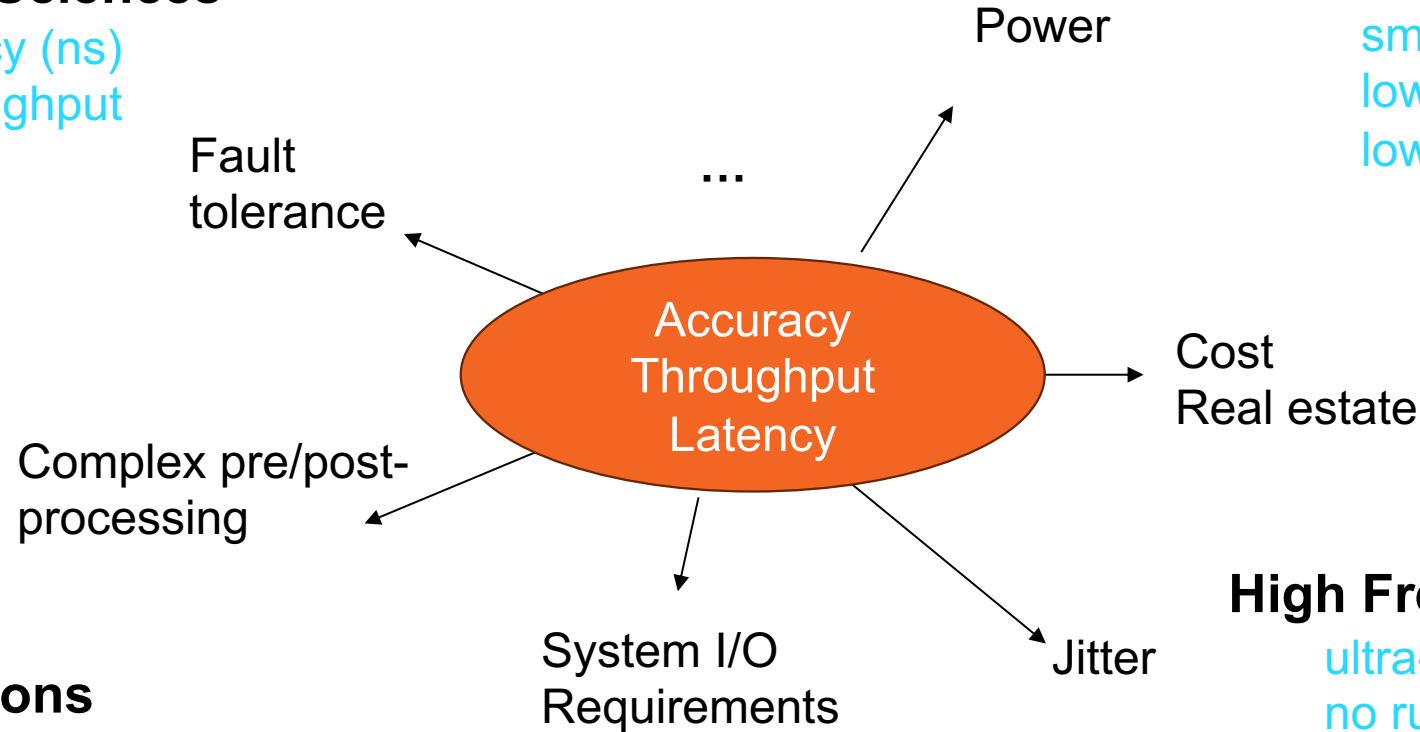
Pervasive AI: Diverse Requirements for Inference

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

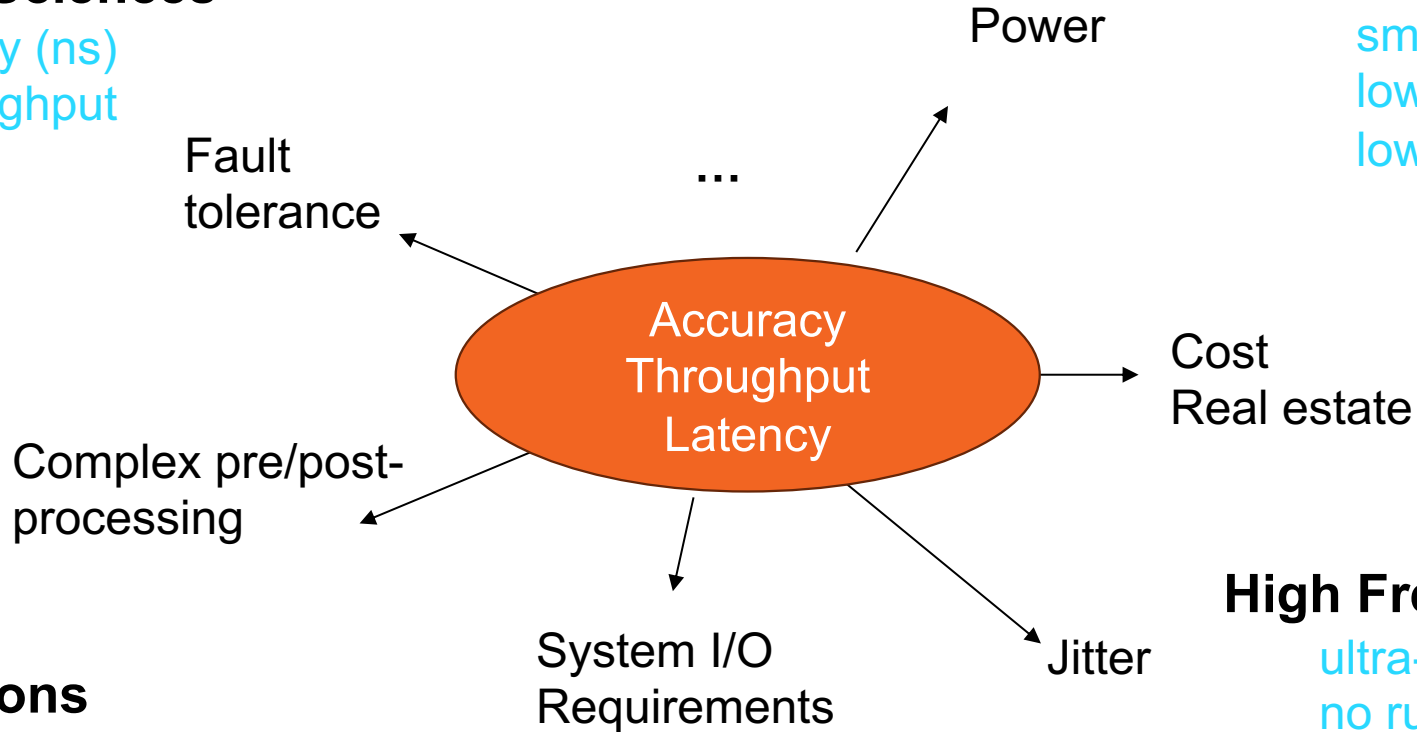
Pervasive AI: Diverse Requirements for Inference

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

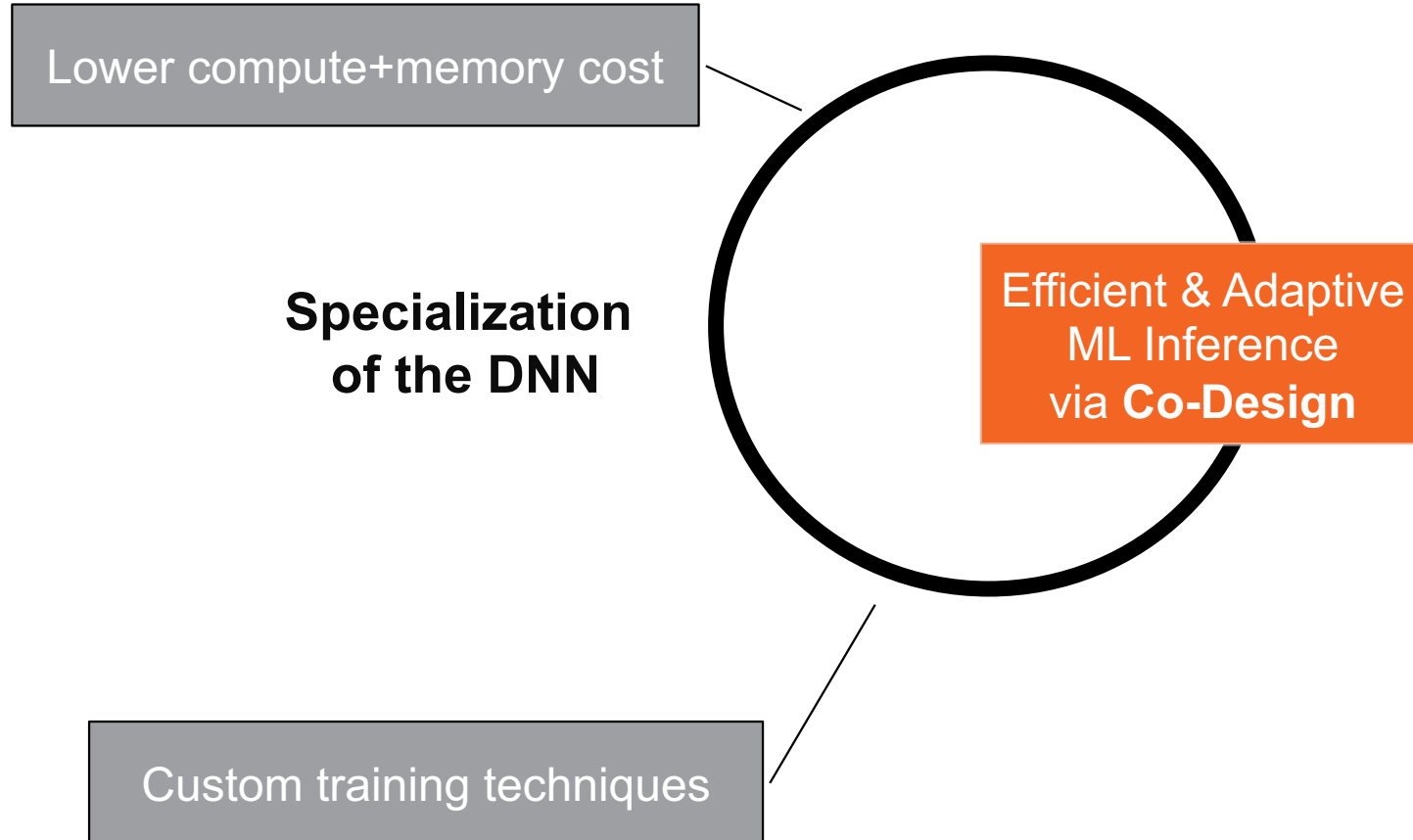
very high throughput (100M/s)
no run-time, stream direct to HW
low latency (sub-msec)
combine with signal processing

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

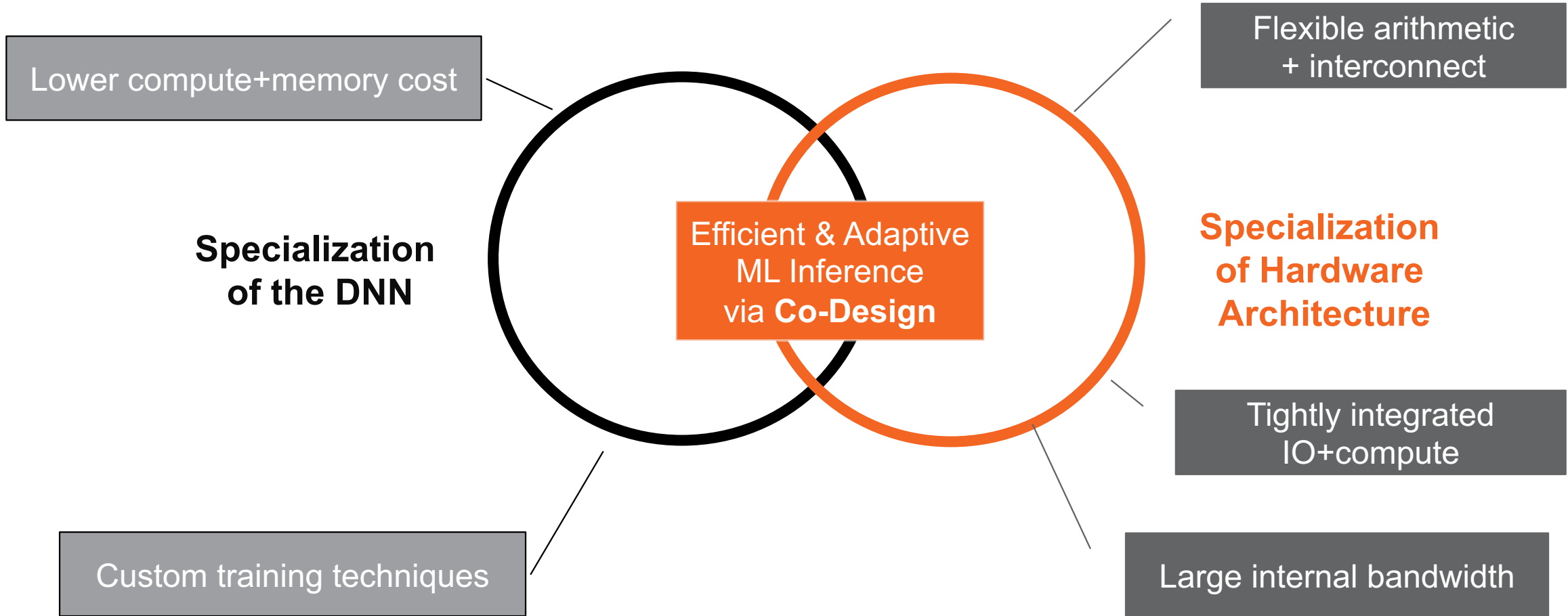
Specialization is essential

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

Specialization is essential

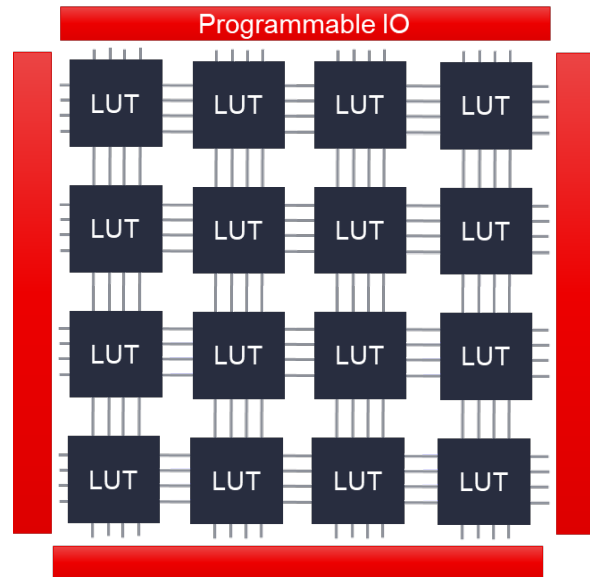


Specialization is essential



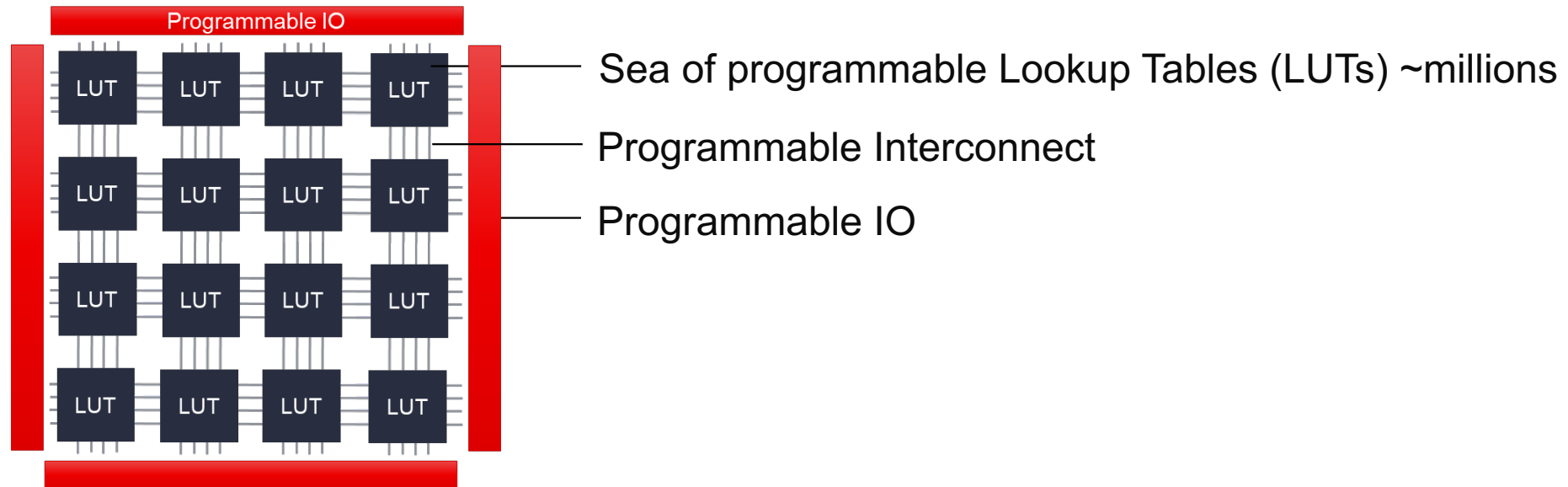
Field-Programmable Gate Arrays (FPGAs)

- 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



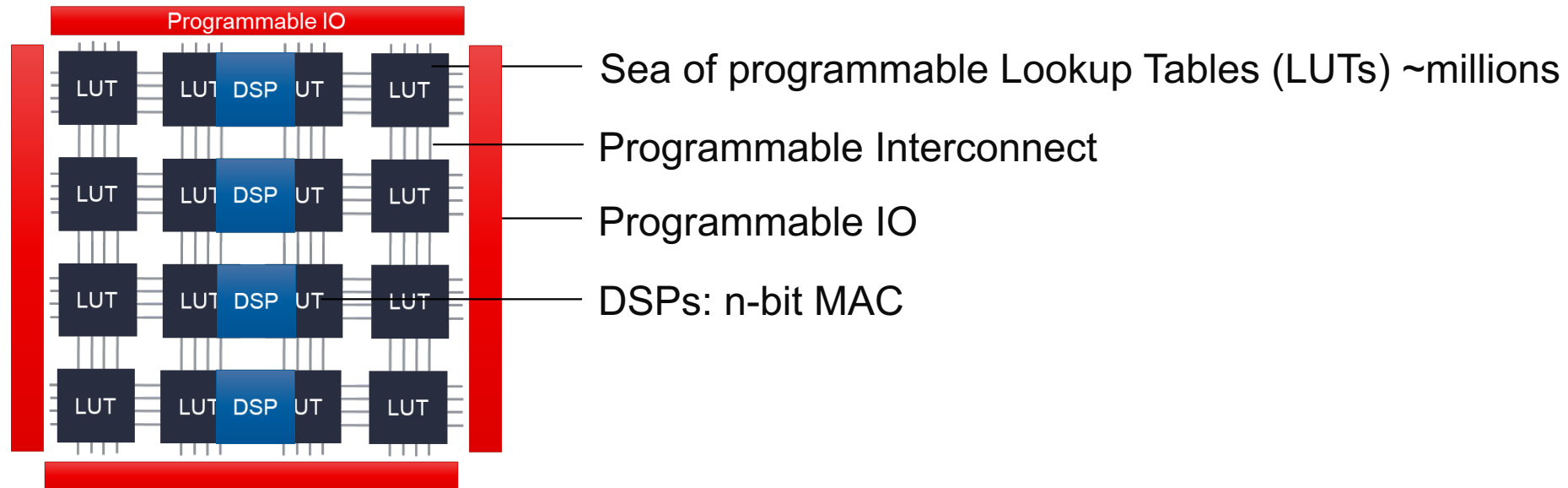
Field-Programmable Gate Arrays (FPGAs)

- 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



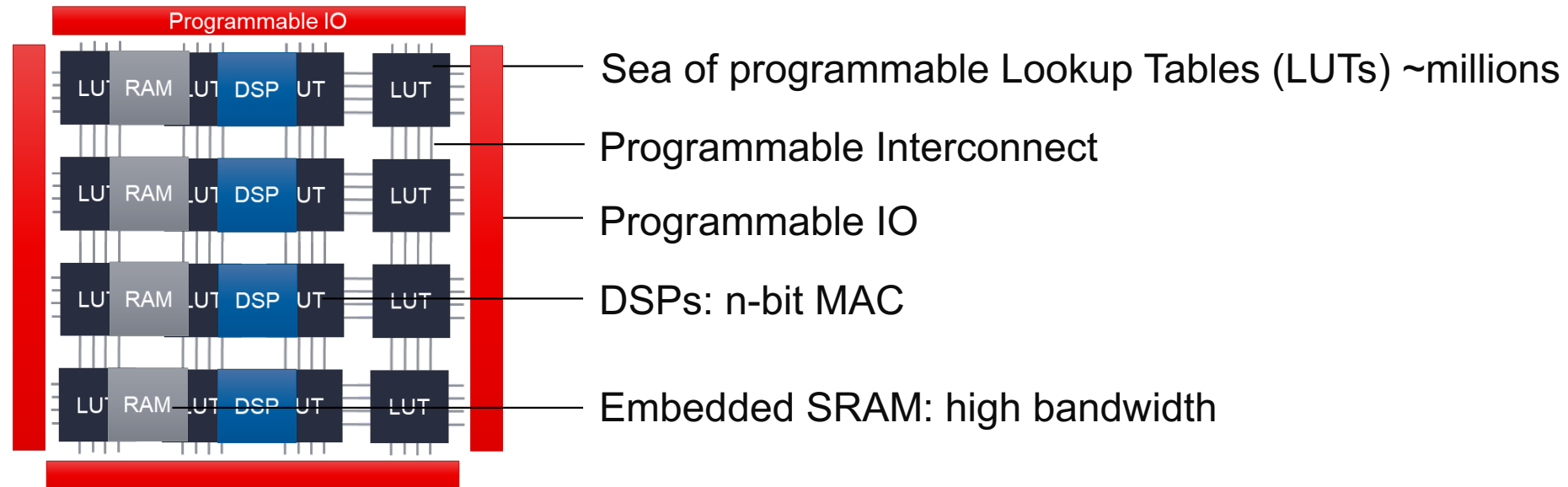
Field-Programmable Gate Arrays (FPGAs)

- 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

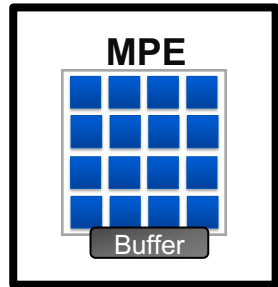


Field-Programmable Gate Arrays (FPGAs)

- 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

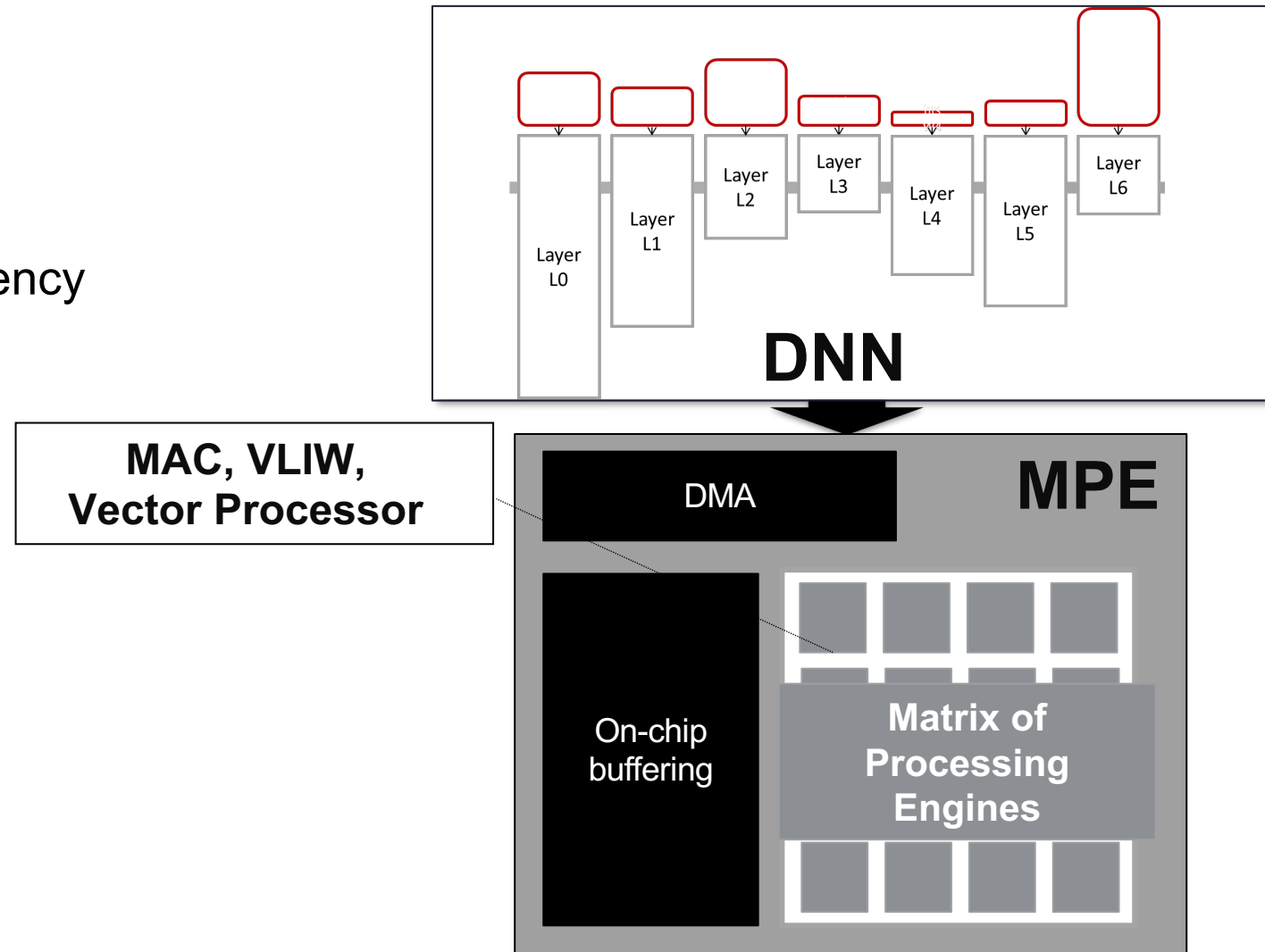
Customized in
datatypes

Customized in
connectivity

Ultimate:
customize the DNN
to fit the hardware

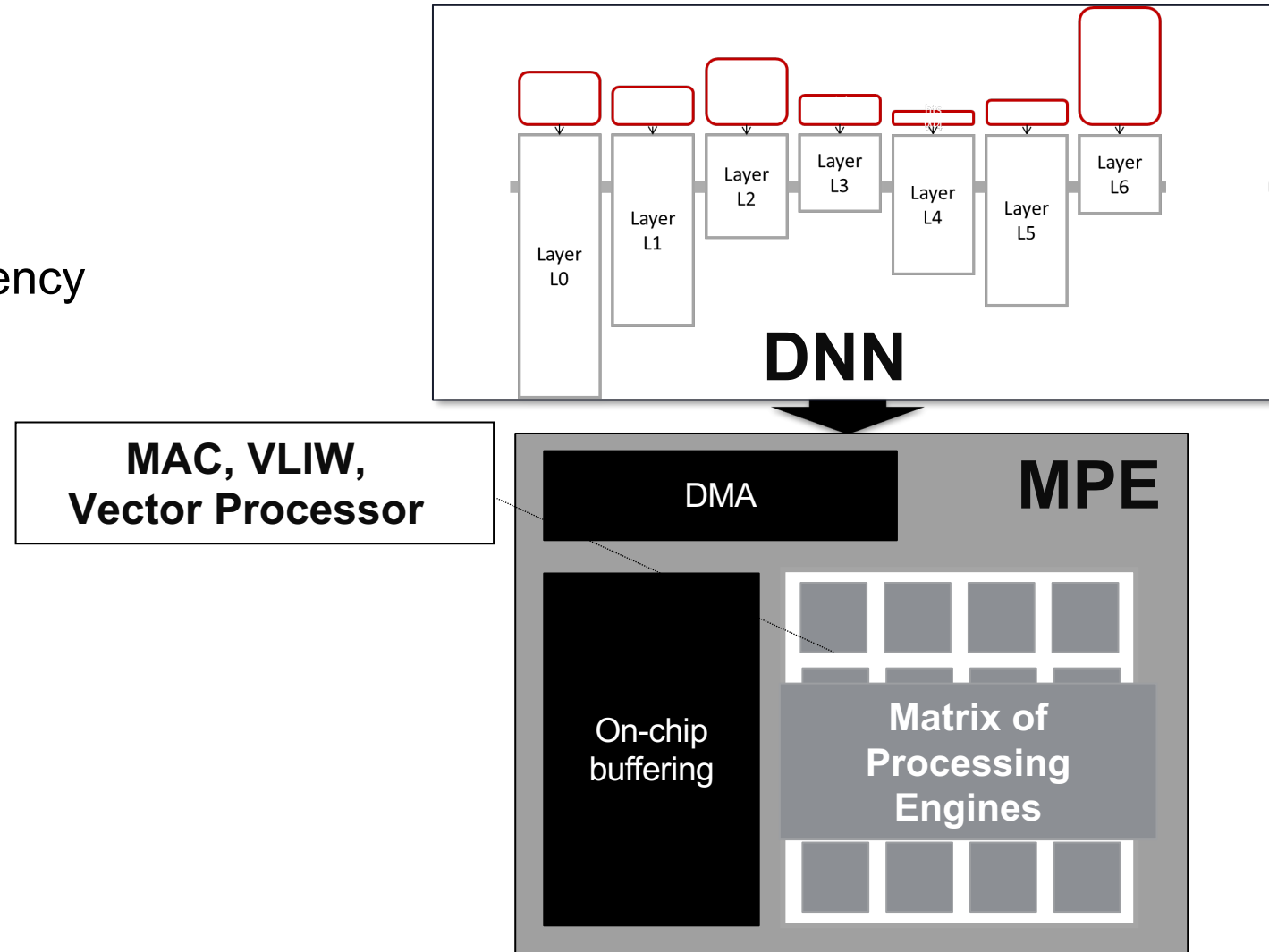
MPE: Customizing for ML Workloads in General

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



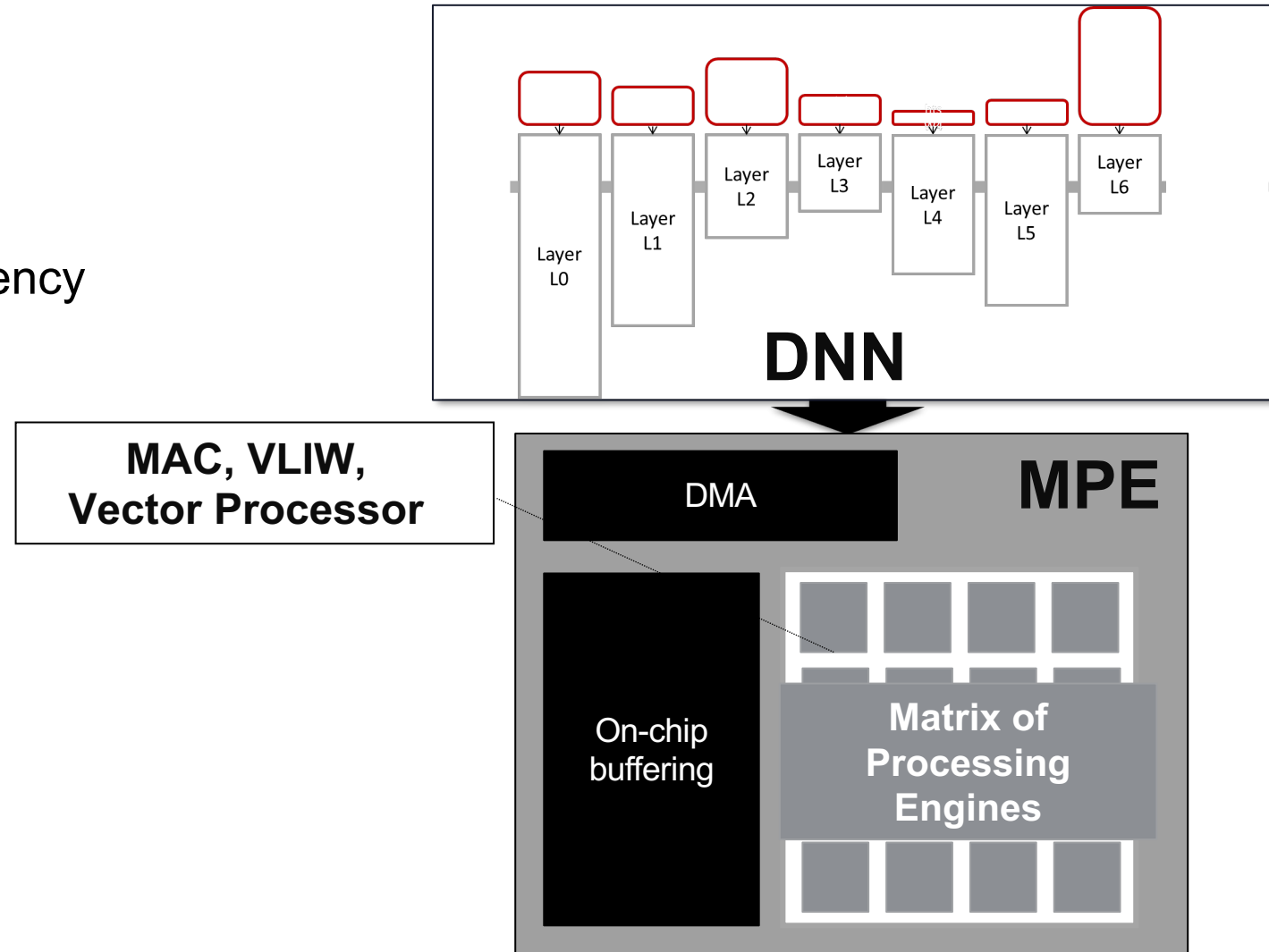
MPE: Customizing for ML Workloads in General

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



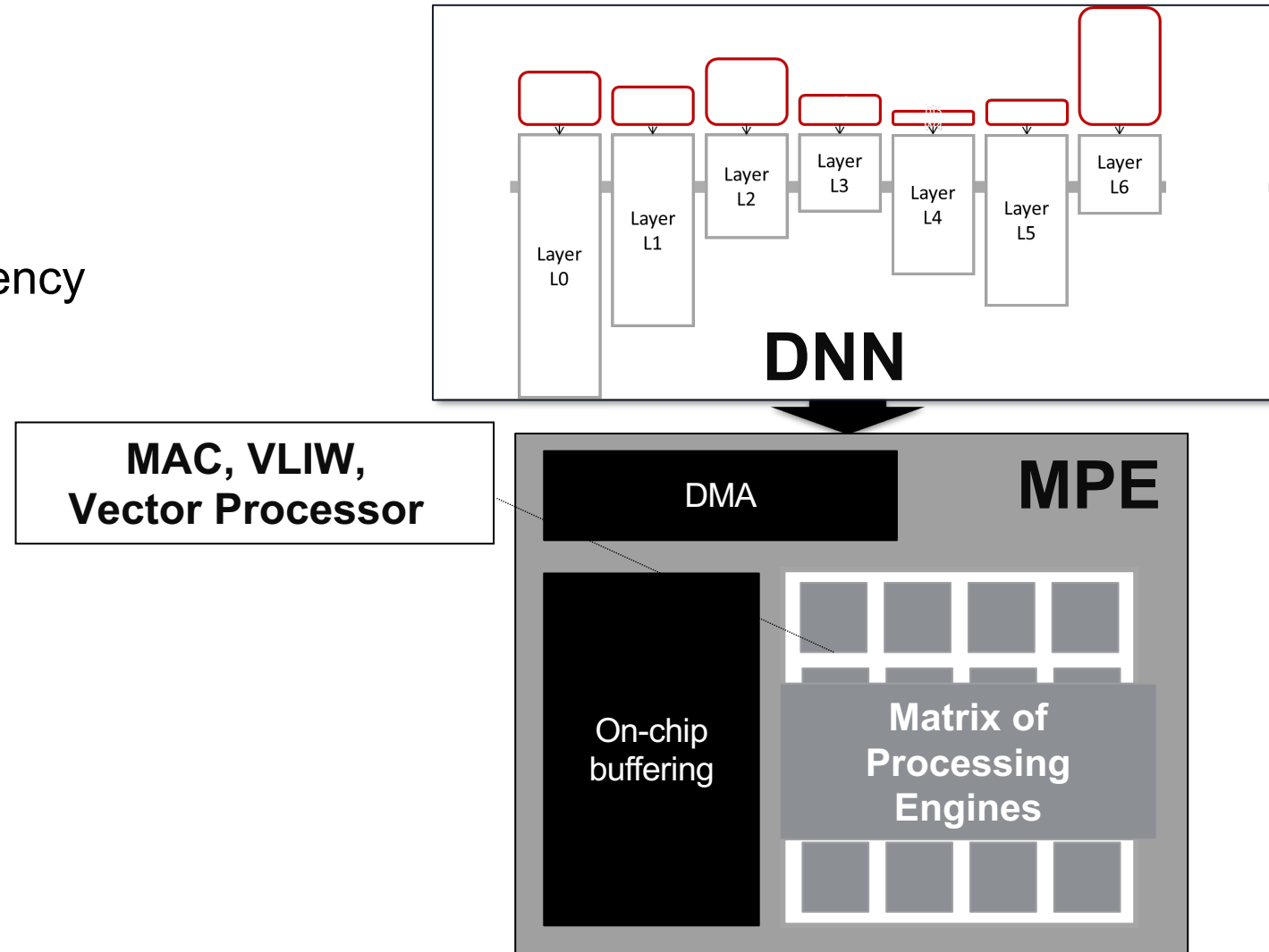
MPE: Customizing for ML Workloads in General

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



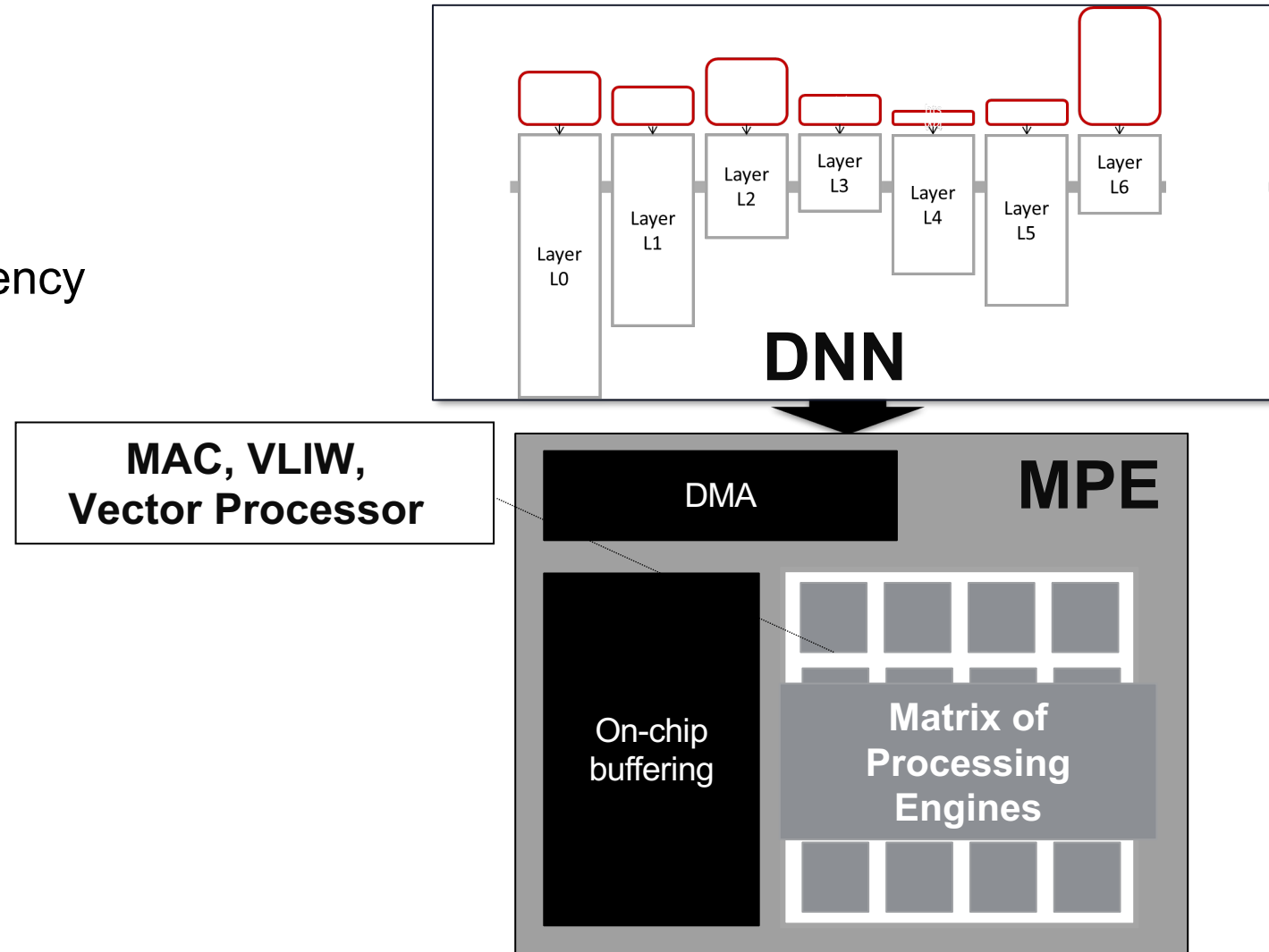
MPE: Customizing for ML Workloads in General

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



MPE: Customizing for ML Workloads in General

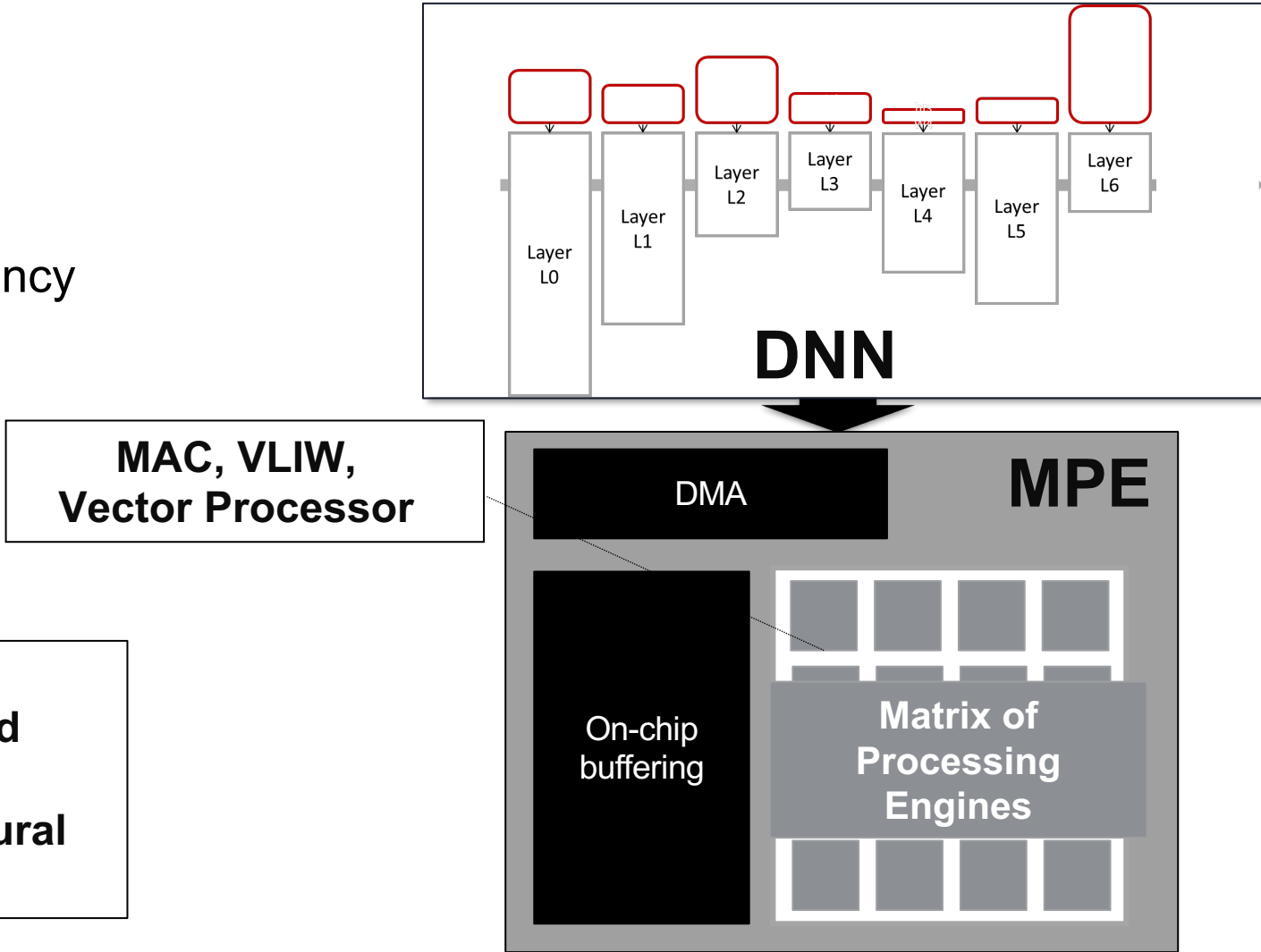
- MPE: Matrix of Processing Engines
- Popular layer-by-layer compute
- Batching to achieve high compute efficiency
- Customized for ML in general
- Specialized processing engines
 - Operators
 - ALU types
 - tensor-, matrix- or vector-based



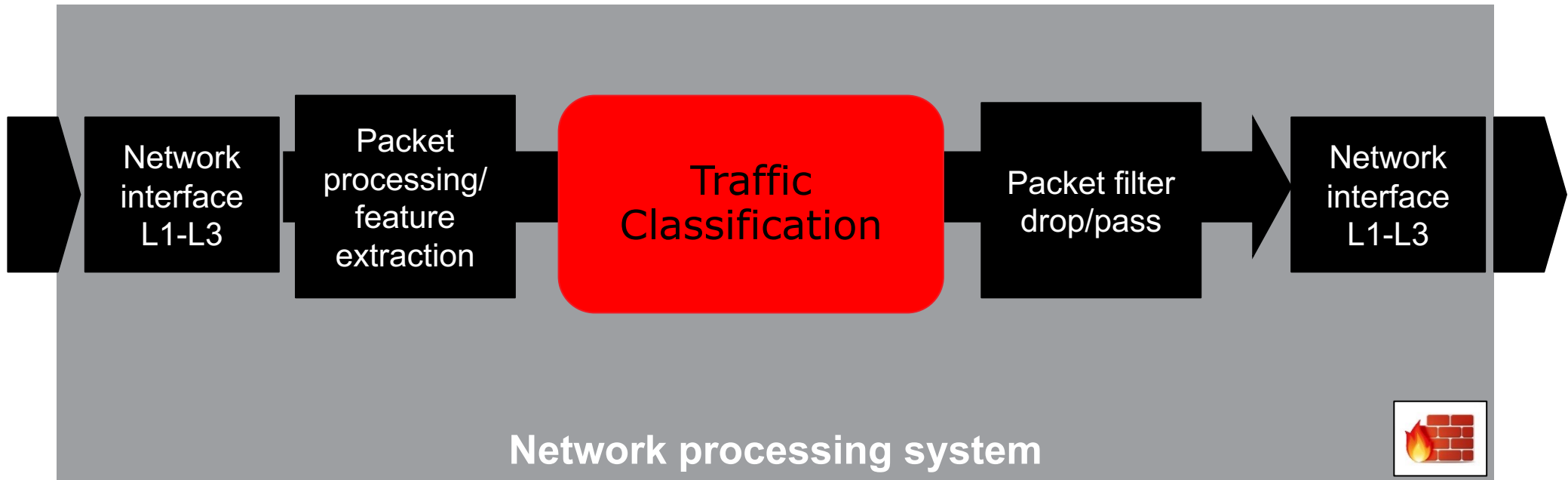
MPE: Customizing for ML Workloads in General

- MPE: Matrix of Processing Engines
- Popular layer-by-layer compute
- Batching to achieve high compute efficiency
- Customized for ML 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
Works well for computer vision and natural language processing



Running Example: Network Intrusion Detection System (NIDS)

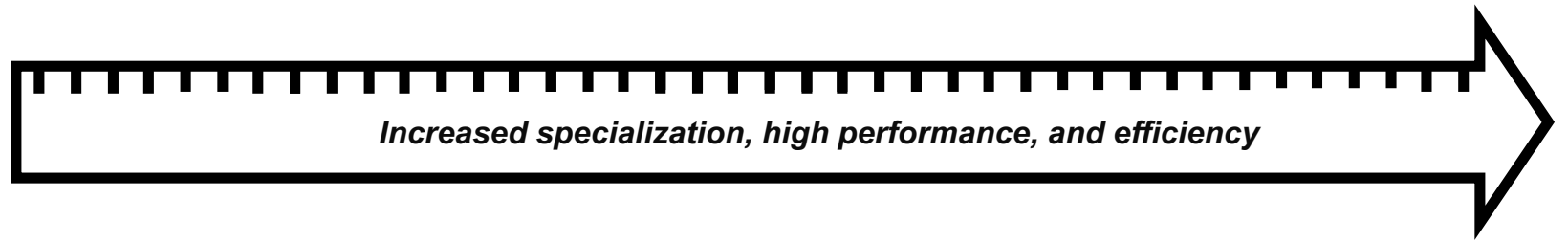


Evaluated with the UNSW-NB15 dataset [1]

	Throughput
5G (20Gbps)	30Minfps
100Gbps	150Minfps
400Gbps	600Minfps

Minfps: Million inferences per second
Assuming 64B/packet

NIDS Results



Matrix of Processing Engines

Topology / #layers / #OPs
Datatype
Accuracy

Vitis AI
MLP / 3 / 92kOPs
8b & 8b
92.3%

Mapped on UltraScale+, 16nm FPGA, all within the same SLR.

*DSPs: 8b or 16b Multiply Accumulates

**BRAMs: 36kb, URAM: 288kbit embedded SRAM blocks

NIDS Results



Matrix of Processing Engines

Topology / #layers / #OPs
Datatype
Accuracy

Vitis AI
MLP / 3 / 92kOPs
8b & 8b
92.3%

Performance
Throughput
Latency (compute only)

22 kinfps
26 us

Mapped on UltraScale+, 16nm FPGA, all within the same SLR.

*DSPs: 8b or 16b Multiply Accumulates

**BRAMs: 36kb, URAM: 288kbit embedded SRAM blocks

NIDS Results



Matrix of Processing Engines

Topology / #layers / #OPs
Datatype
Accuracy

Vitis AI
MLP / 3 / 92kOPs
8b & 8b
92.3%

Performance
Throughput
Latency (compute only)

22 kinfps
26 us

Resources
Compute (kLUTs, DSPs*)
Memory (BRAM, URAM**)
Clock

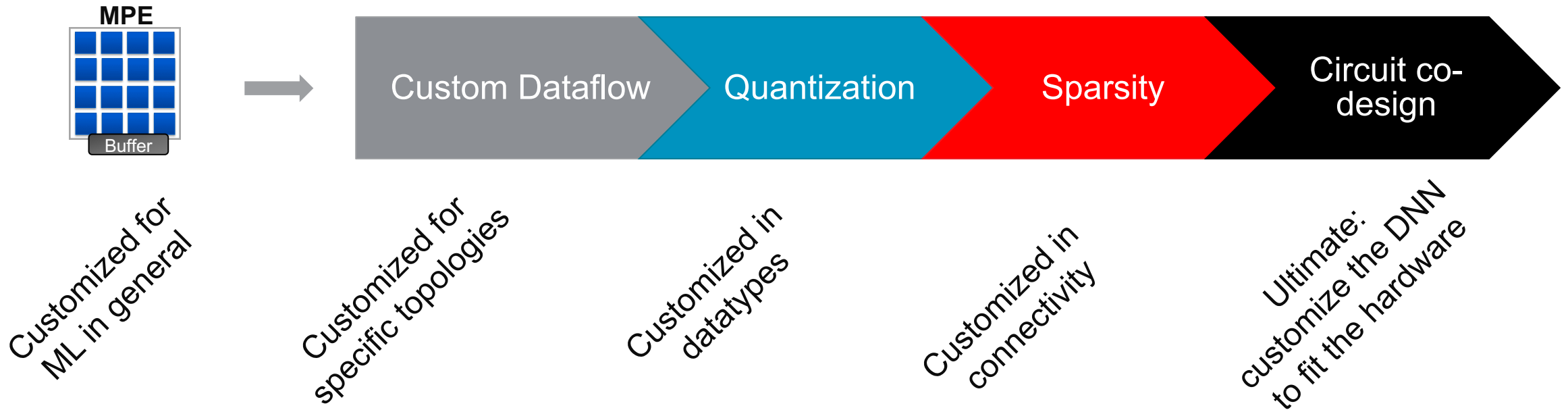
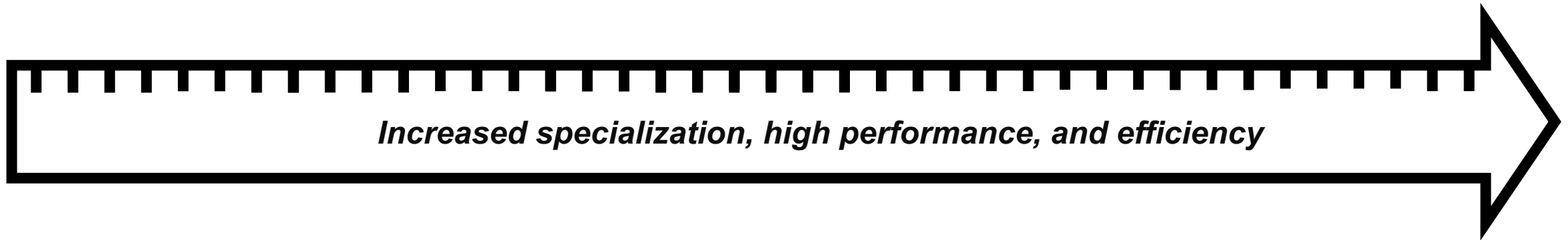
122,1124
290, 92
300/600 MHz

Mapped on UltraScale+, 16nm FPGA, all within the same SLR.

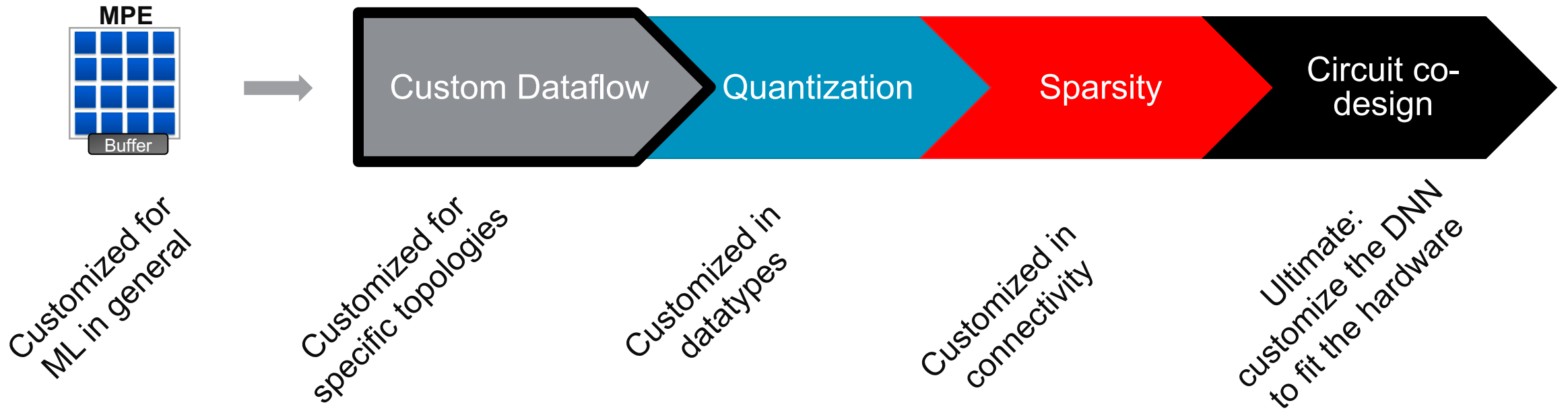
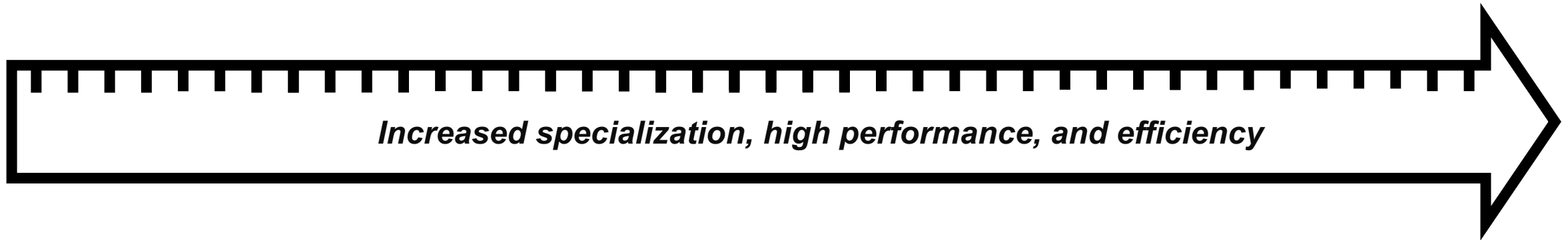
*DSPs: 8b or 16b Multiply Accumulates

**BRAMs: 36kb, URAM: 288kbit embedded SRAM blocks

Specialized FPGA Inference via Co-Design

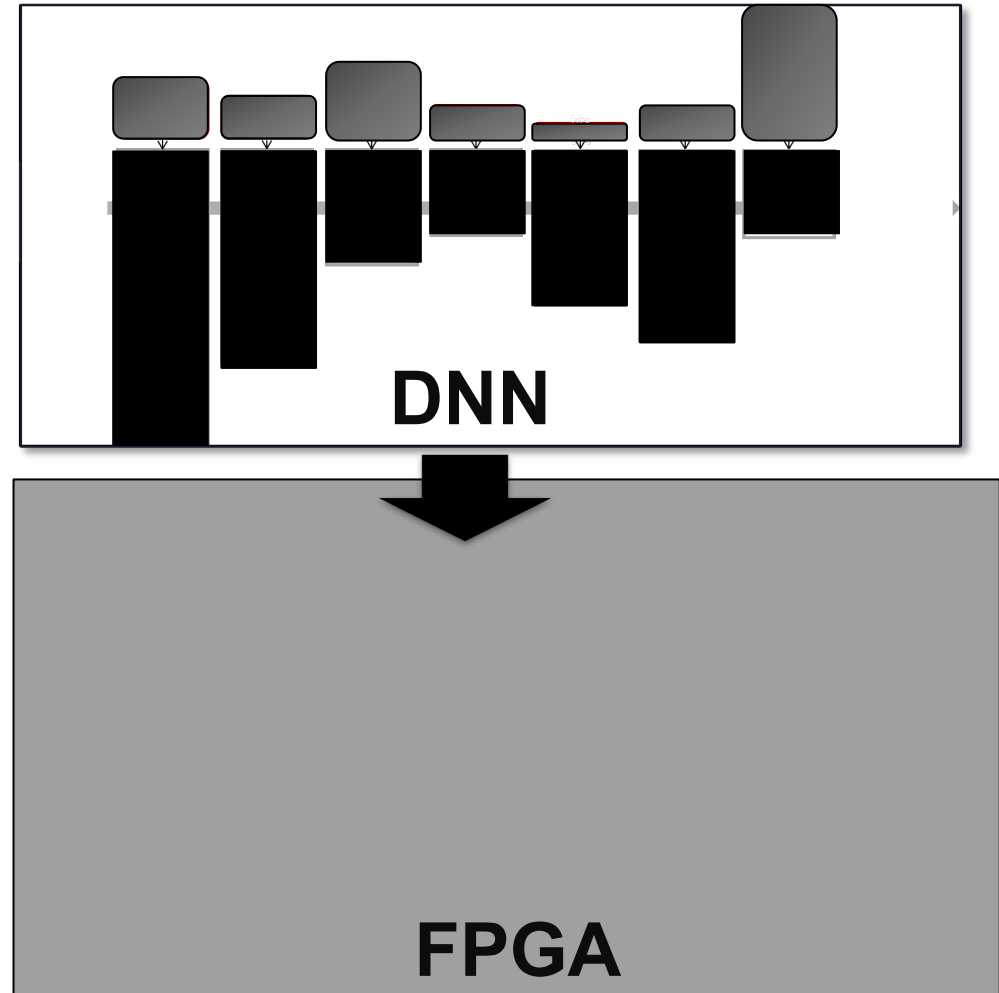


Specialized FPGA Inference via Co-Design



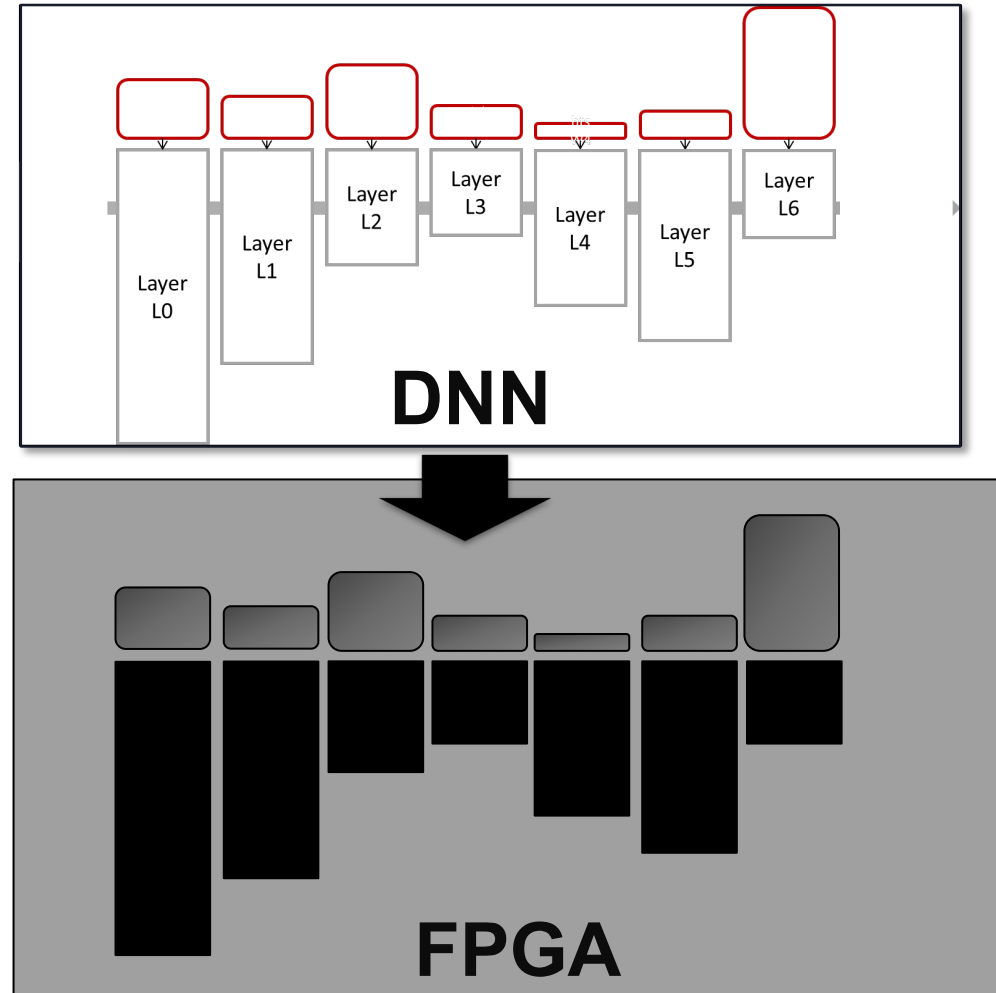
Streaming Dataflow - Specializing for a Topology

- Hardware architecture mimics the topology
- All 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



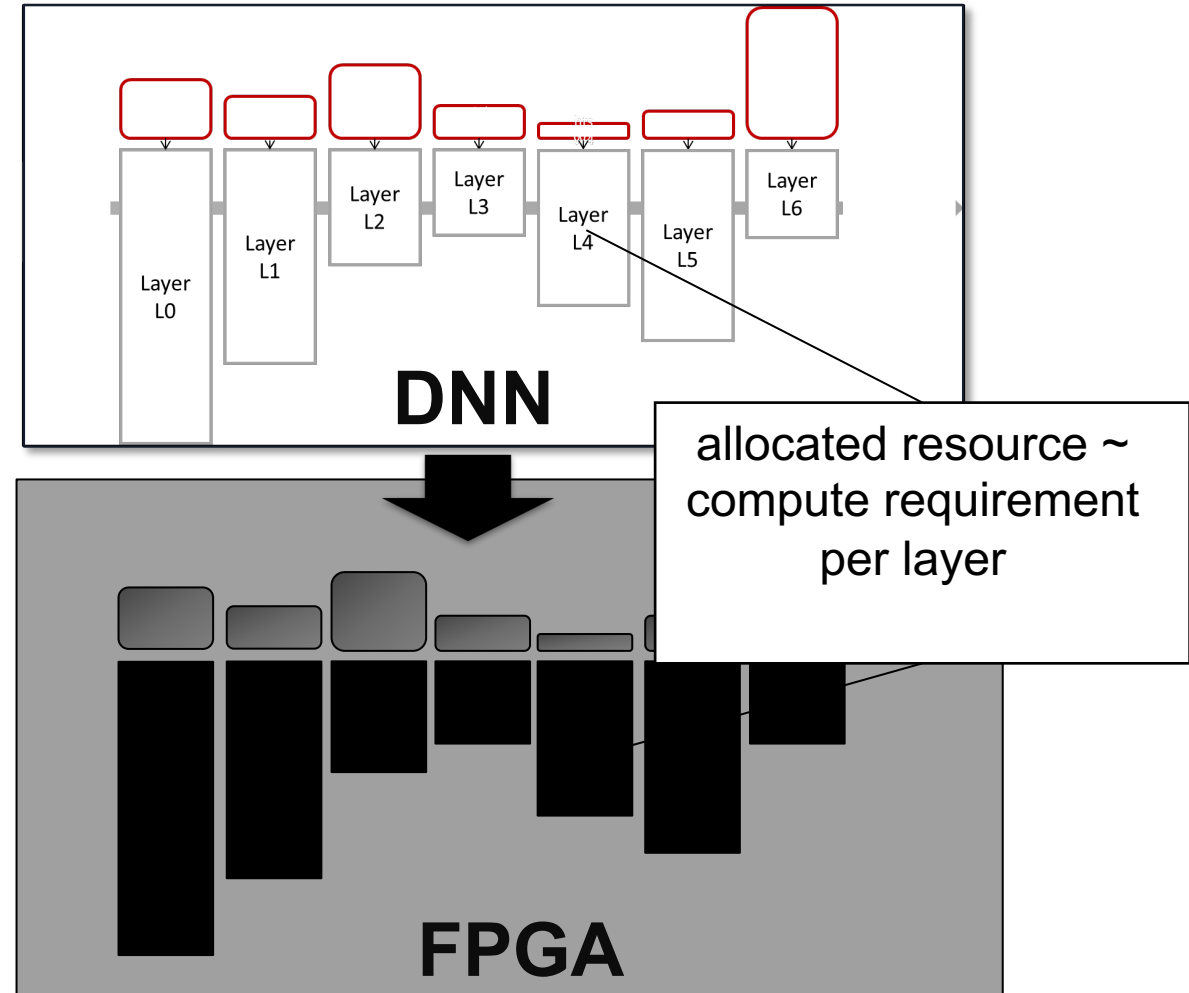
Streaming Dataflow - Specializing for a Topology

- Hardware architecture mimics the topology
- All 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



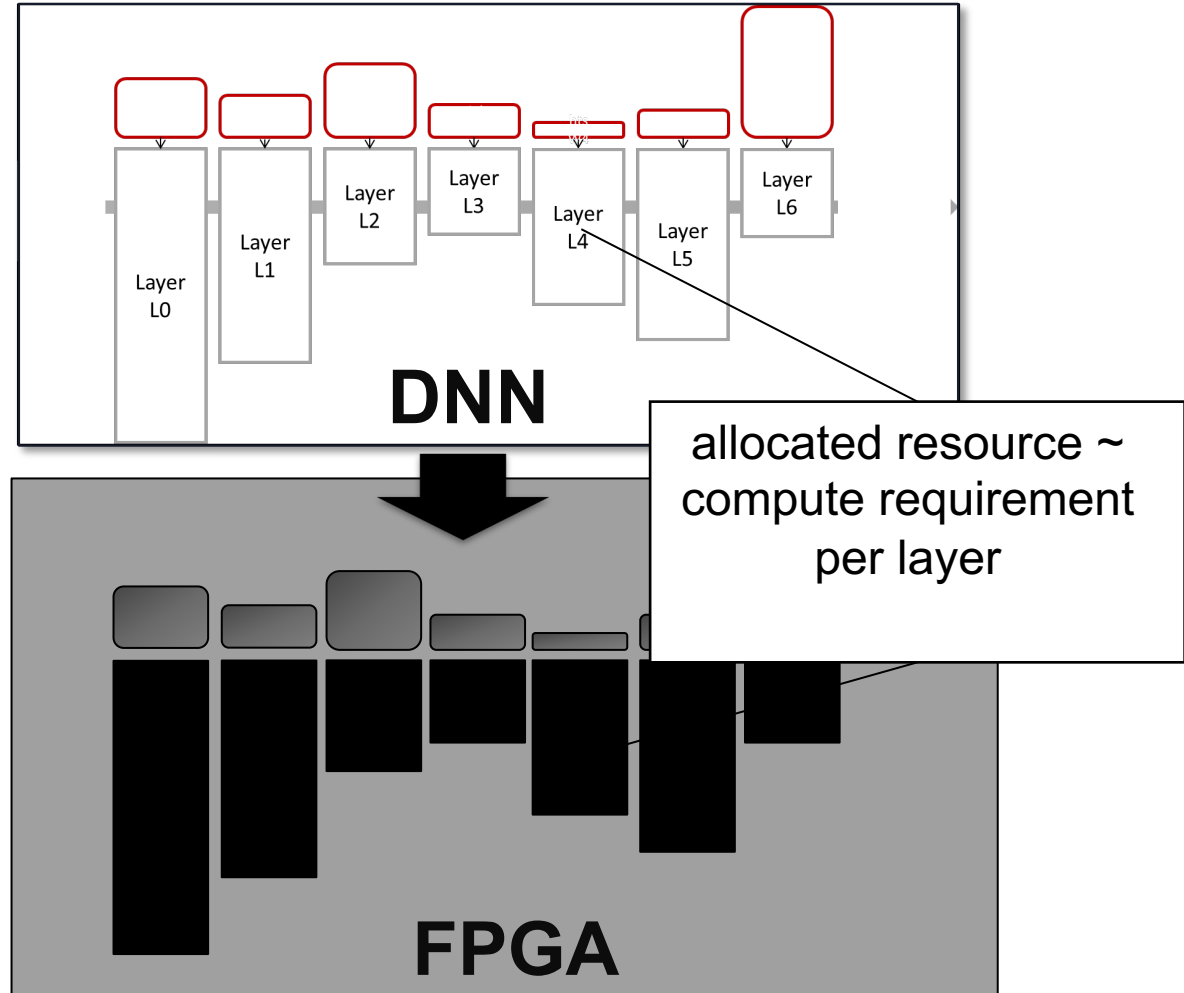
Streaming Dataflow - Specializing for a Topology

- Hardware architecture mimics the topology
- All 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



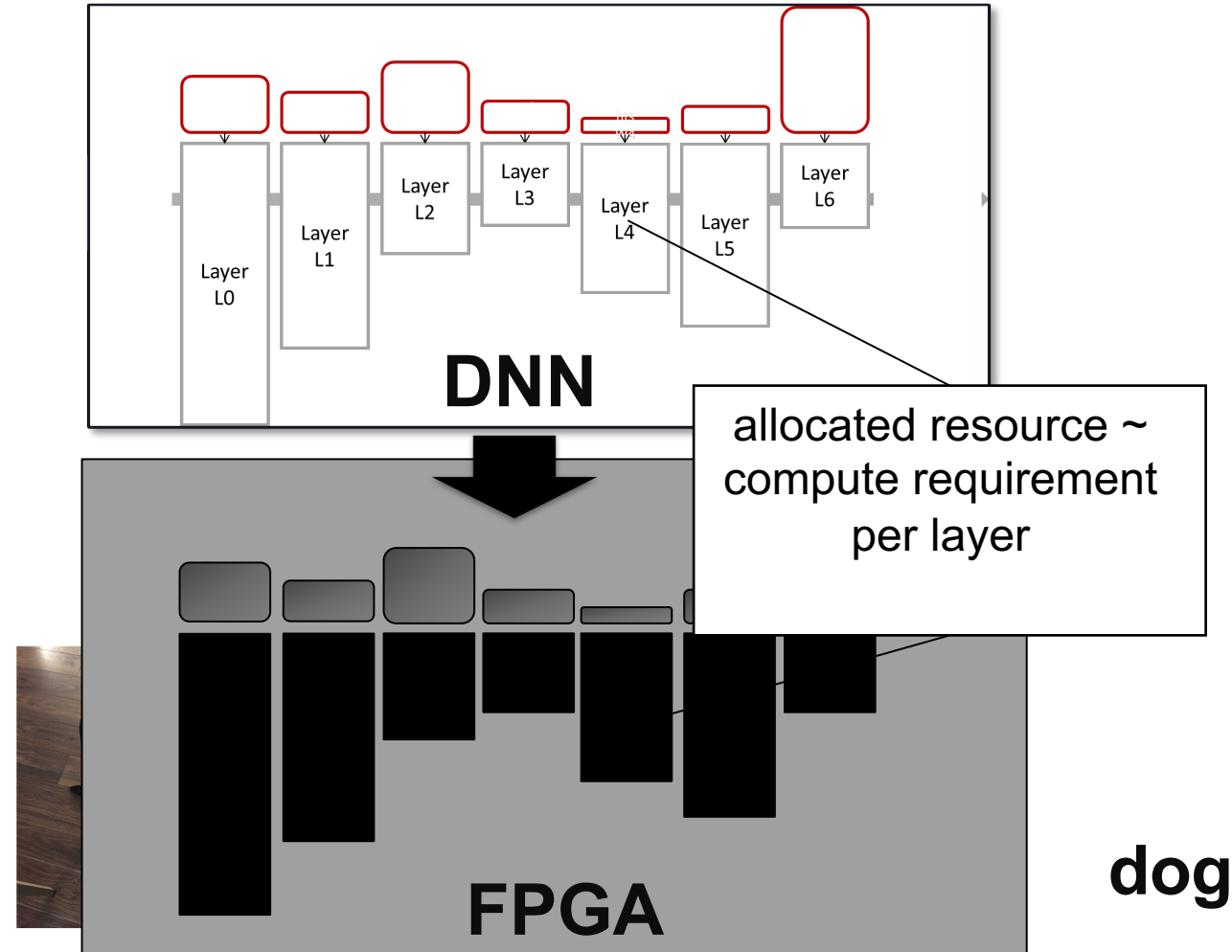
Streaming Dataflow - Specializing for a Topology

- Hardware architecture mimics the topology
- All 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

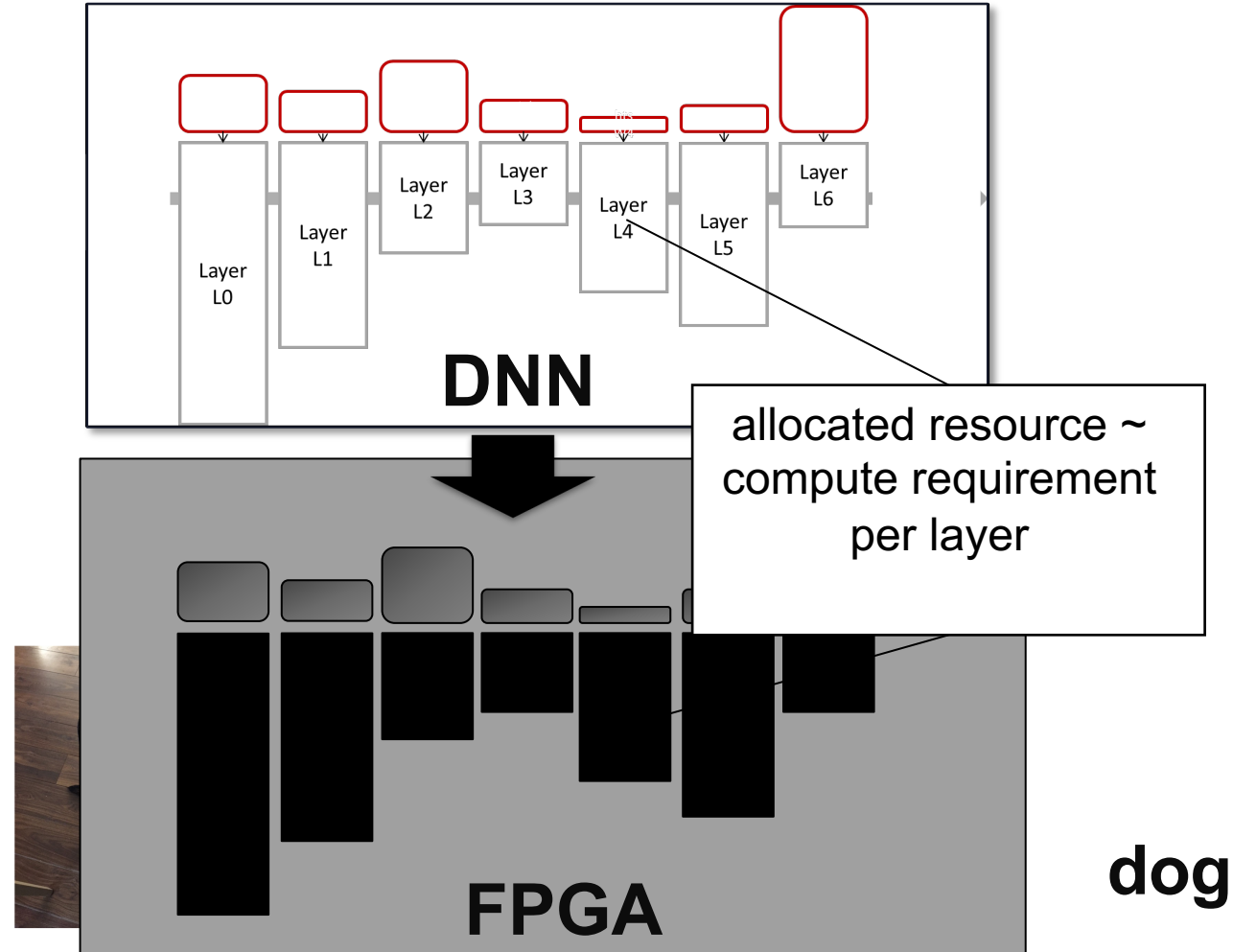
- Hardware architecture mimics the topology
- All weights need to be accessible in parallel, but limited activation buffering needed
- Customize *everything* to the specifics of the DNN
- Benefits
 - Improved efficiency
 - Low fixed latency



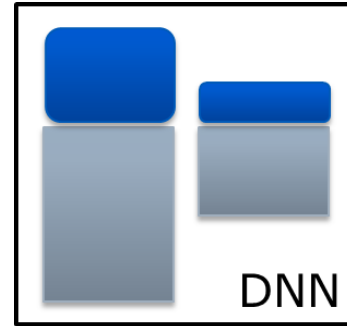
Streaming Dataflow - Specializing for a Topology

- Hardware architecture mimics the topology
- All 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

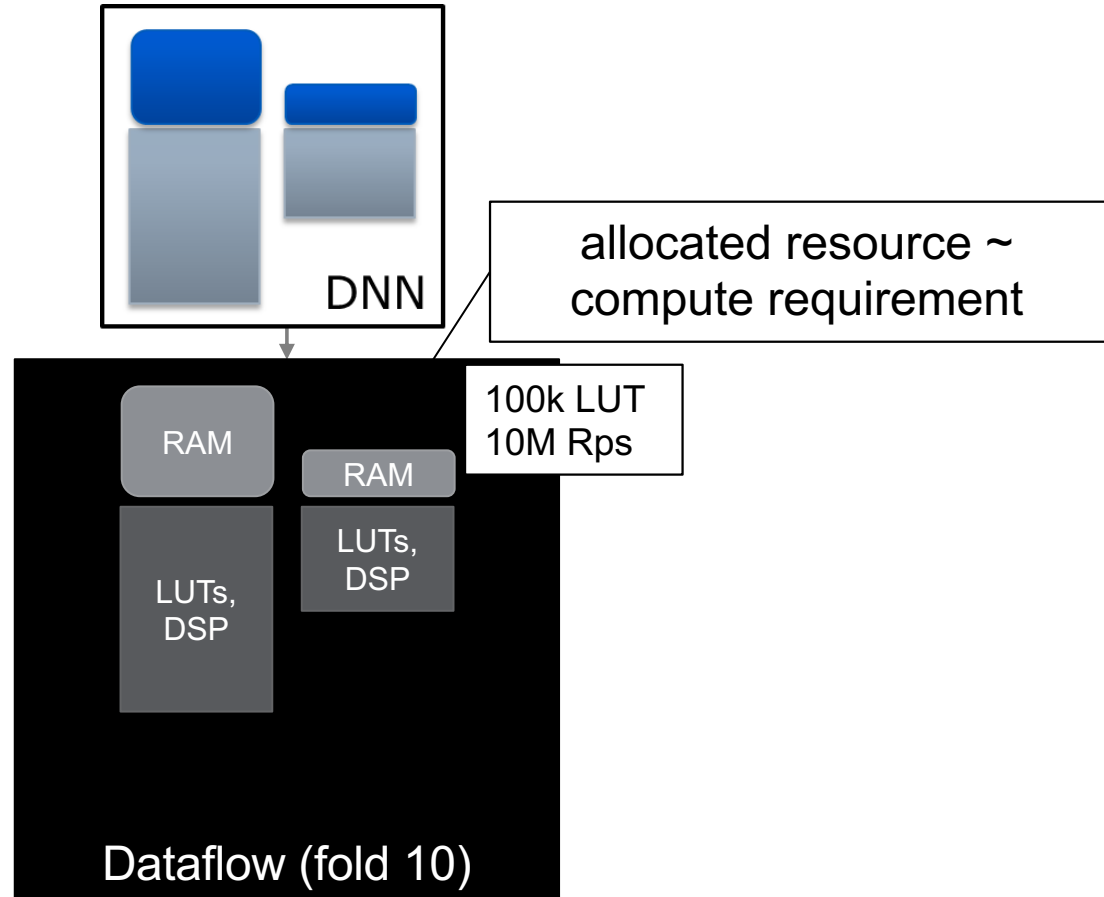
Dataflow can scale performance to meet the application requirements



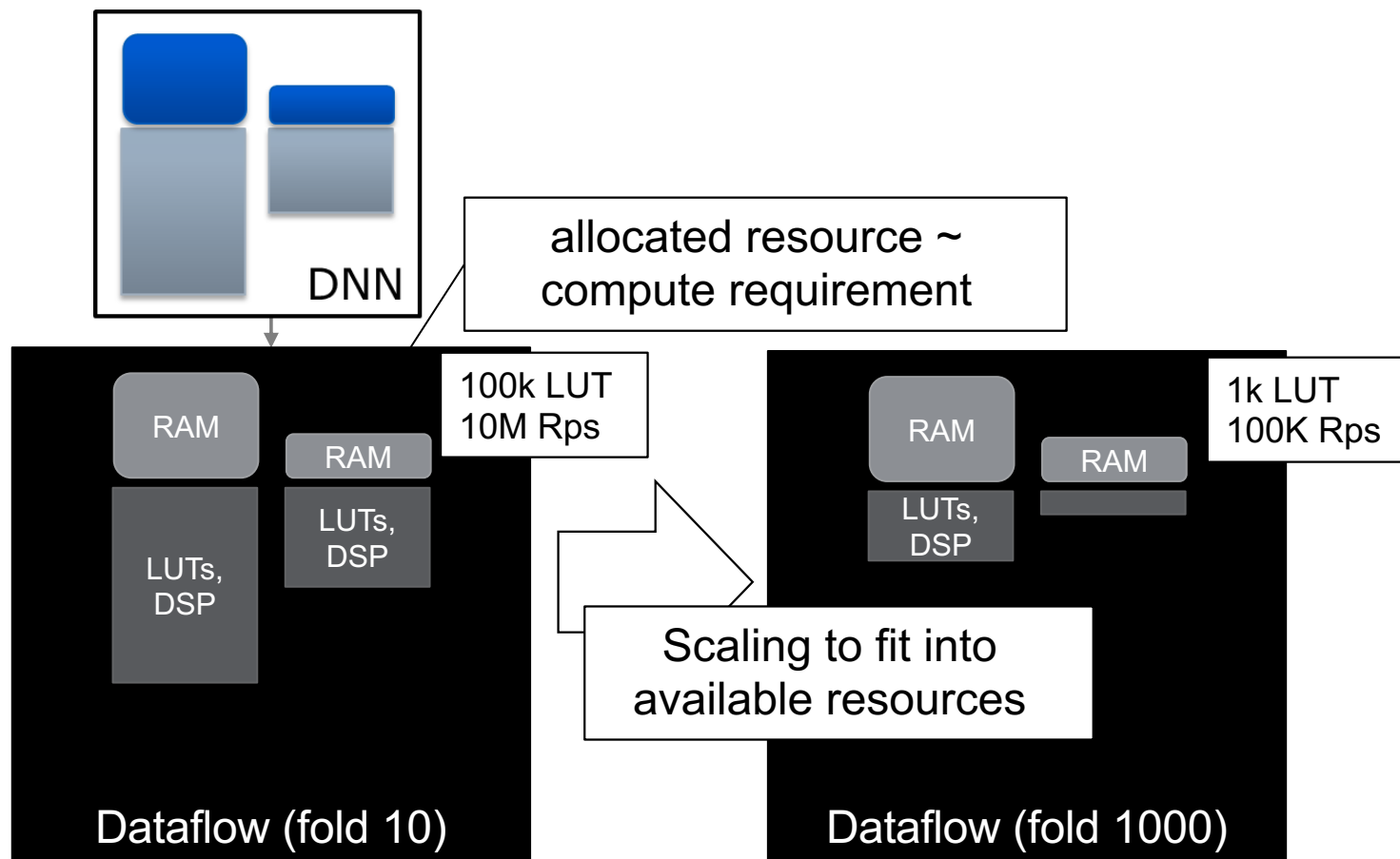
Scaling to Meet Performance & Resource Requirements



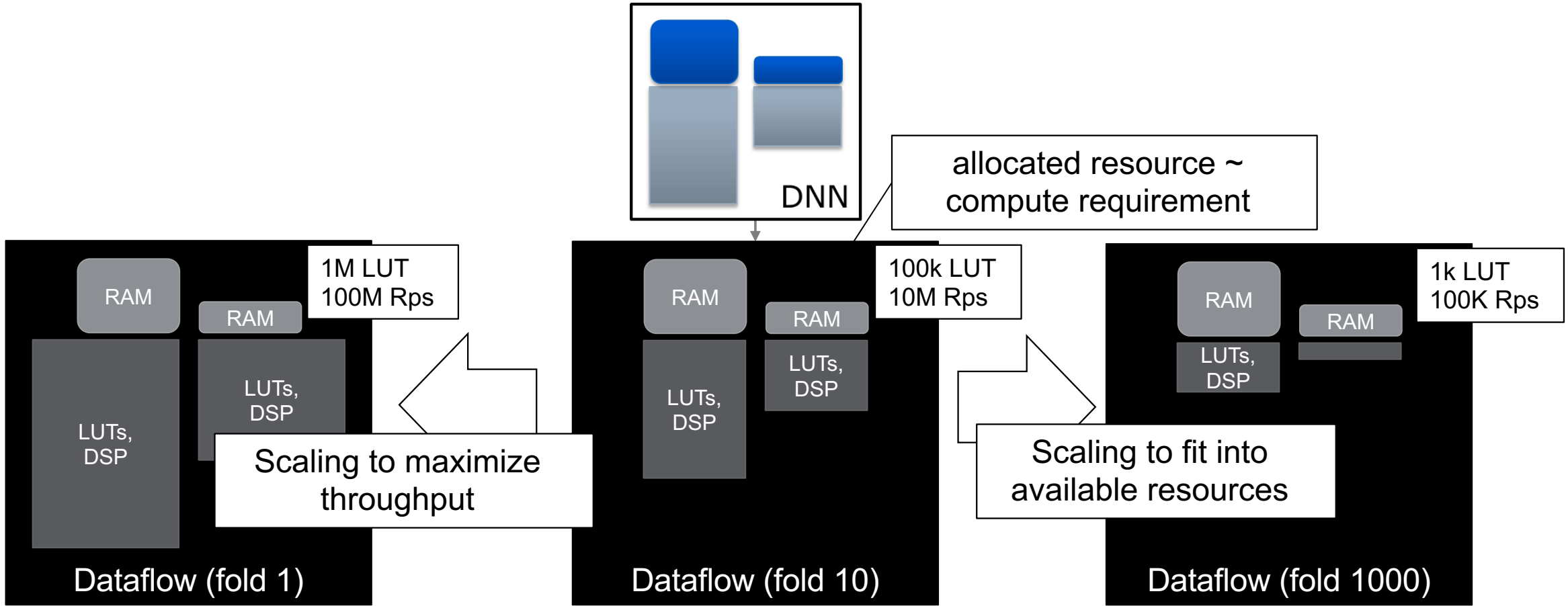
Scaling to Meet Performance & Resource Requirements



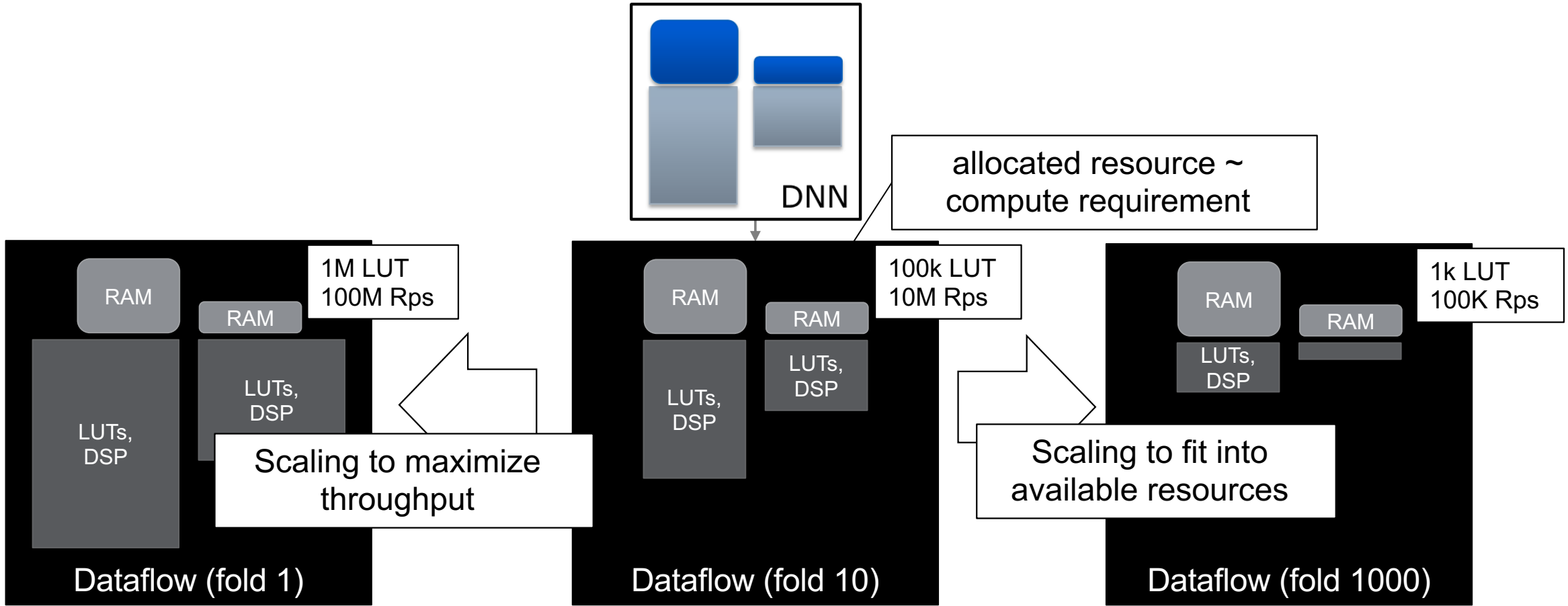
Scaling to Meet Performance & Resource Requirements



Scaling to Meet Performance & Resource Requirements

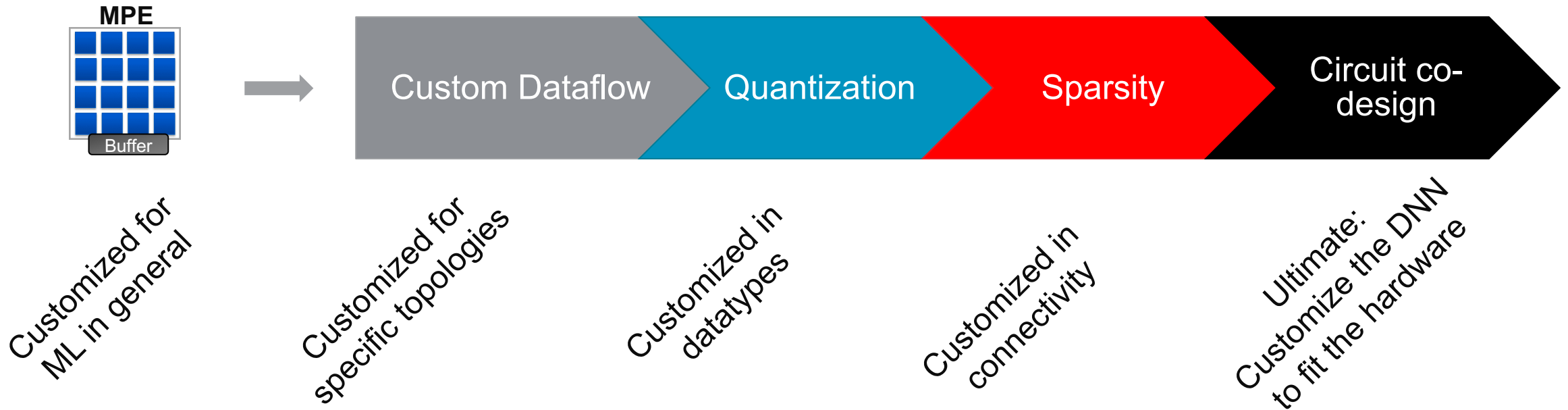
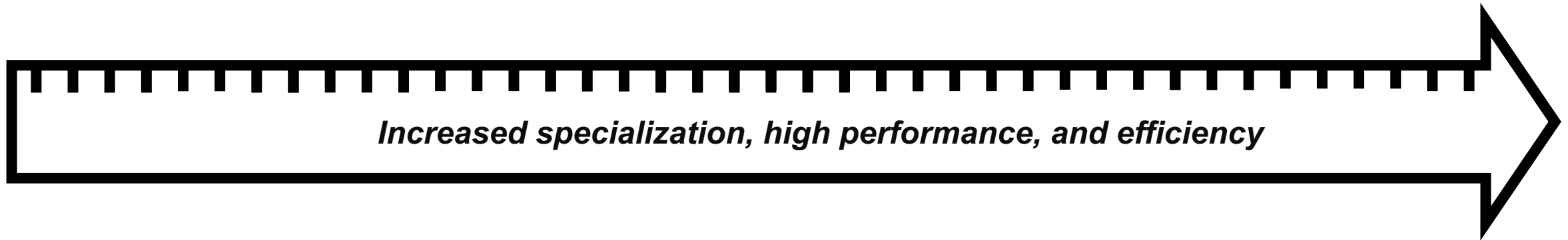


Scaling to Meet Performance & Resource Requirements

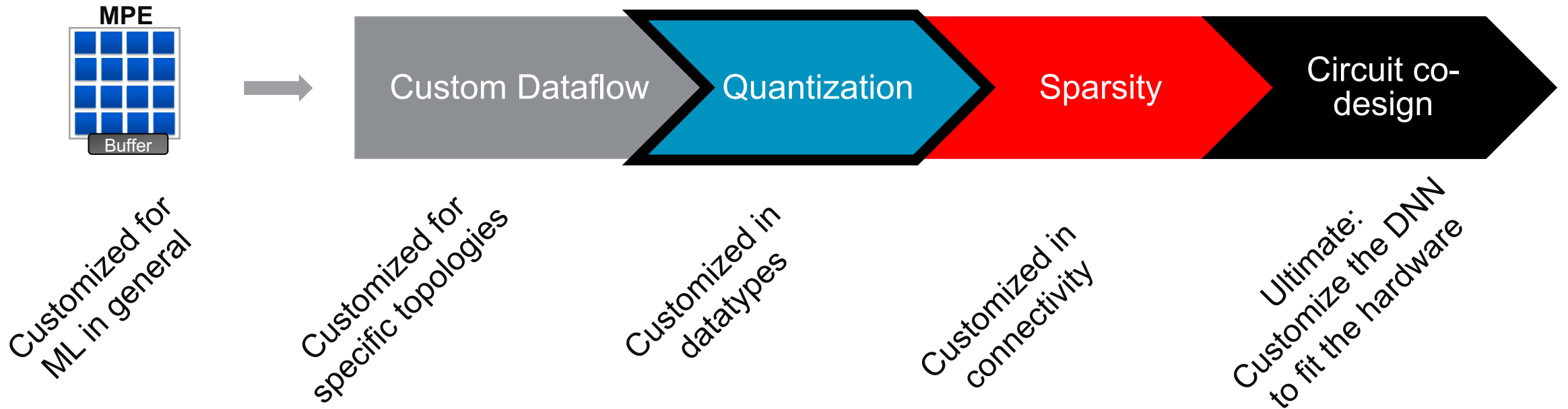


- Scale performance & resources to meet the application requirements
- If resources allow, we can **fully unfold the NN** to create a circuit that inferences at clock speed
 - Enables extra optimizations for fine-granular quantization and sparsity

Specialized FPGA Inference via Co-Design



Specialized FPGA Inference via Co-Design



Benefits of Quantization on FPGAs

AMD UltraScale+ MPSoC ZU19EG (conservative estimates)



Benefits of Quantization on FPGAs

AMD UltraScale+ MPSoC ZU19EG (conservative estimates)

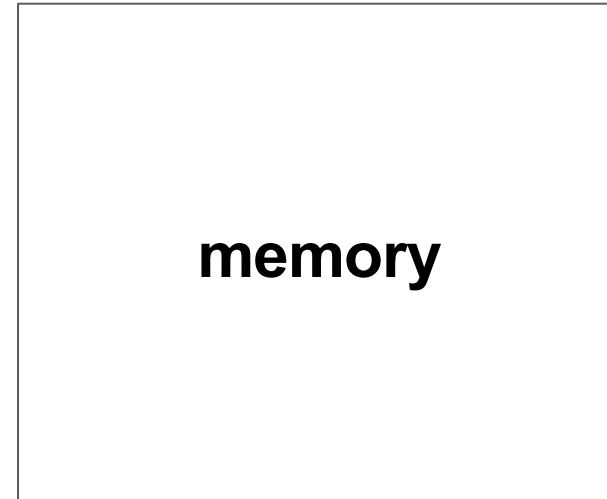
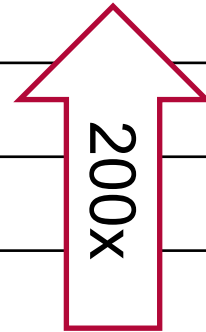
Precision	Approx. Peak GOPS
1b	64 000
4b	16 000
8b	4 000
32b	300

memory

Benefits of Quantization on FPGAs

AMD UltraScale+ MPSoC ZU19EG (conservative estimates)

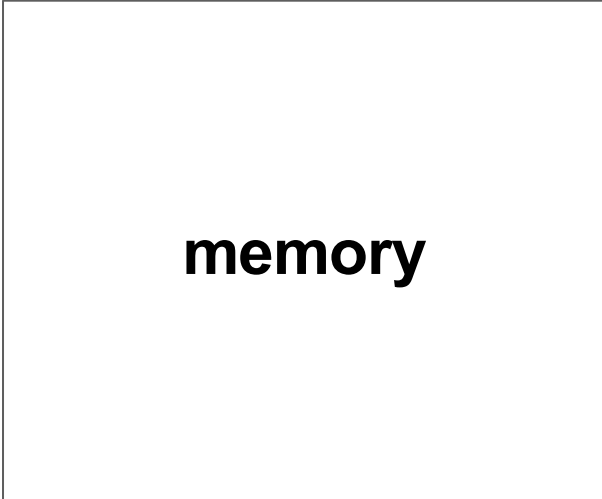
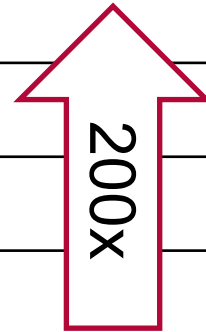
Precision	Approx. Peak GOPS
1b	64 000
4b	16 000
8b	4 000
32b	300



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

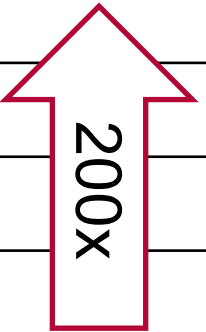


Trillions of
quantized
operations per
second

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

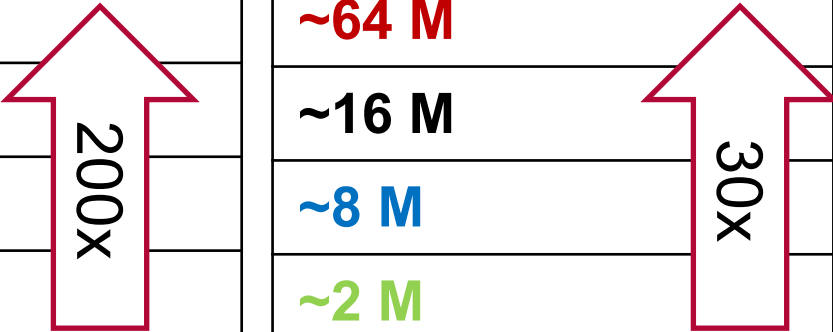


Trillions of
quantized
operations per
second

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

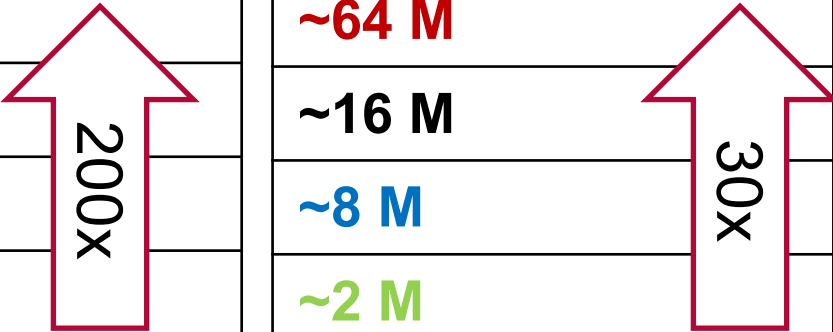


Trillions of
quantized
operations per
second

Benefits of Quantization on FPGAs

AMD UltraScale+ MPSoC ZU19EG (conservative estimates)

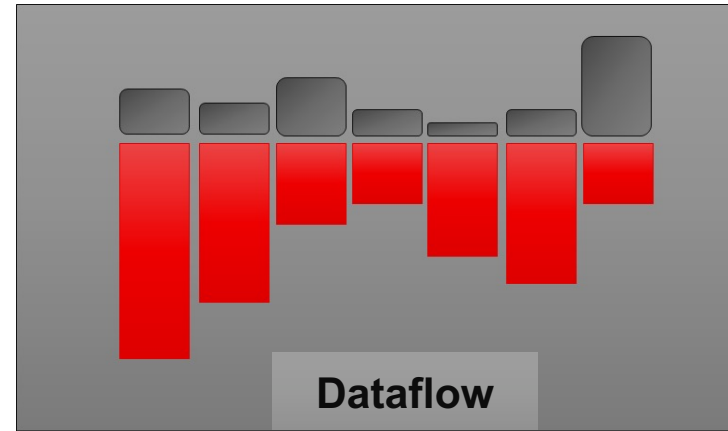
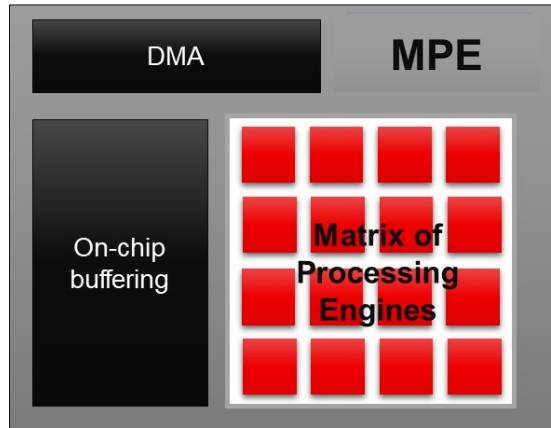
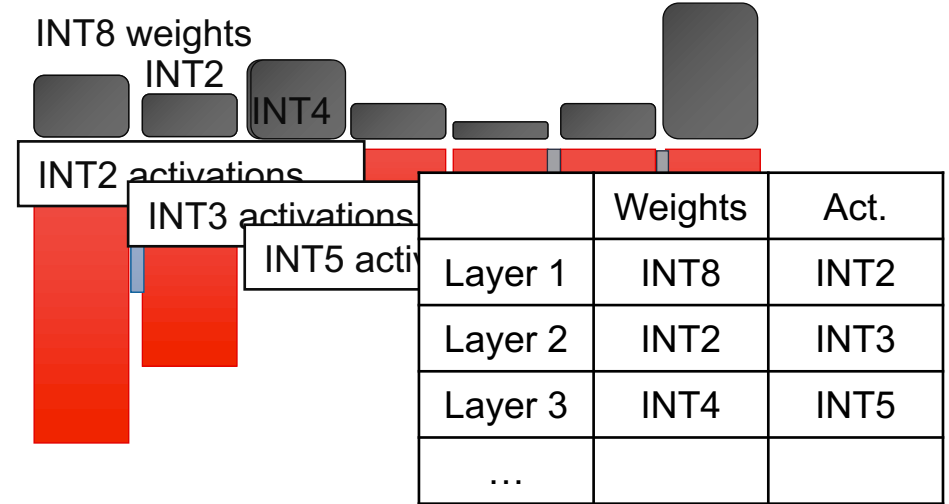
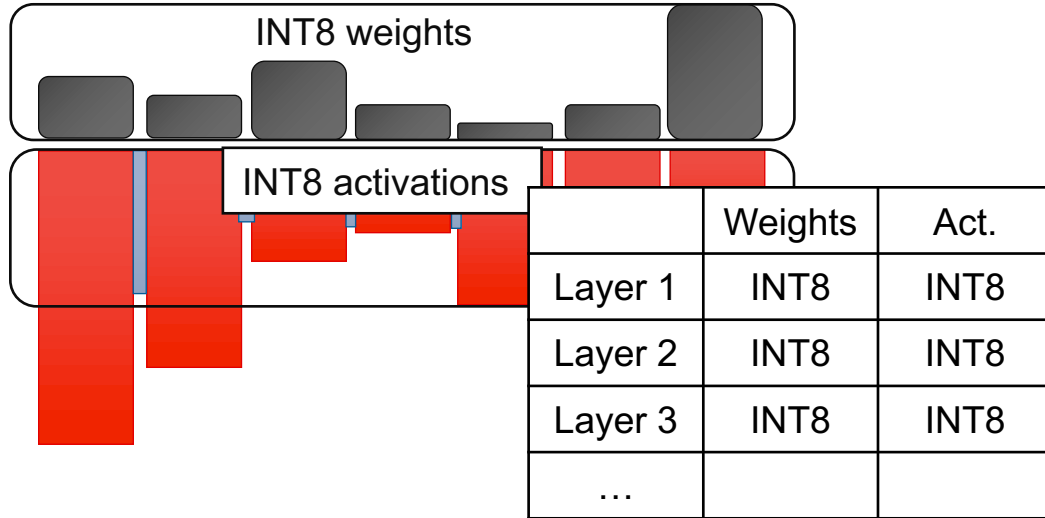
Precision	Approx. Peak GOPS	On-chip weights
1b	64 000	~64 M
4b	16 000	~16 M
8b	4 000	~8 M
32b	300	~2 M



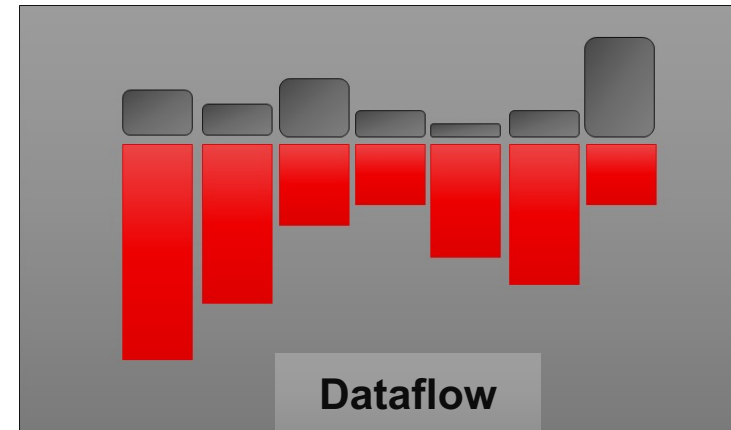
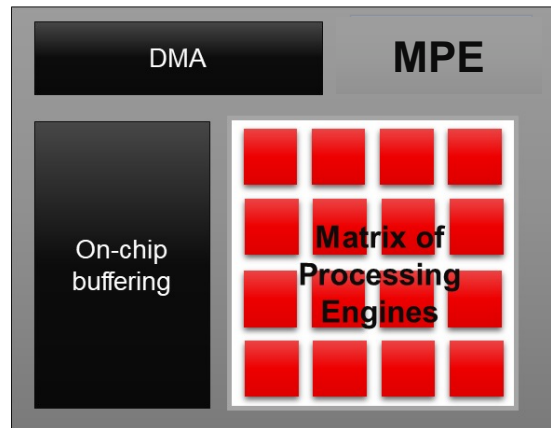
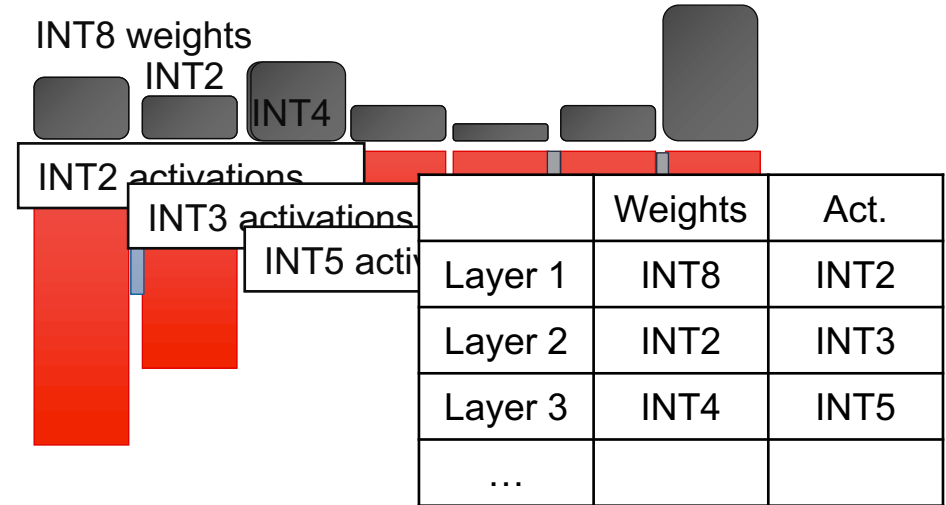
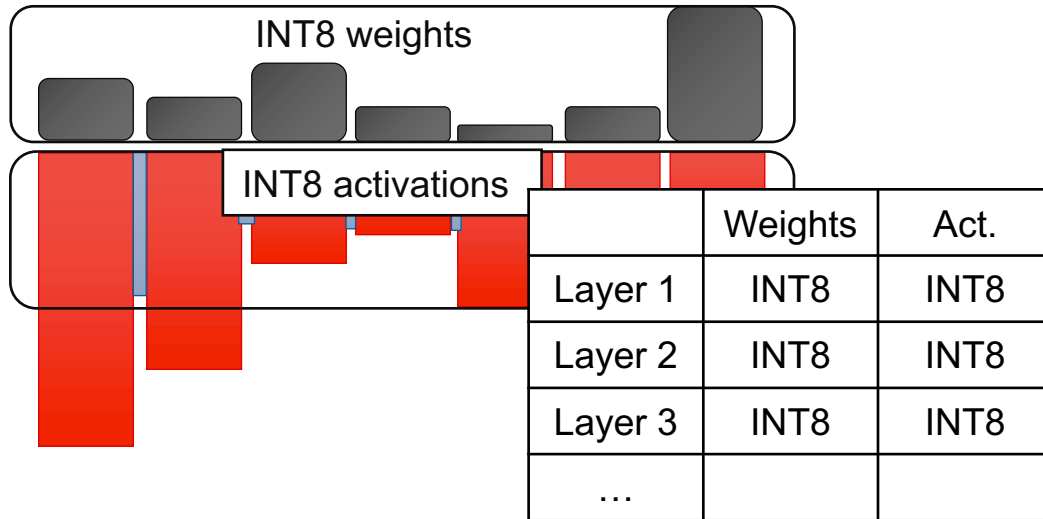
Trillions of quantized operations per second

Weights can stay **entirely on-chip**

Granularity of Customizing Arithmetic

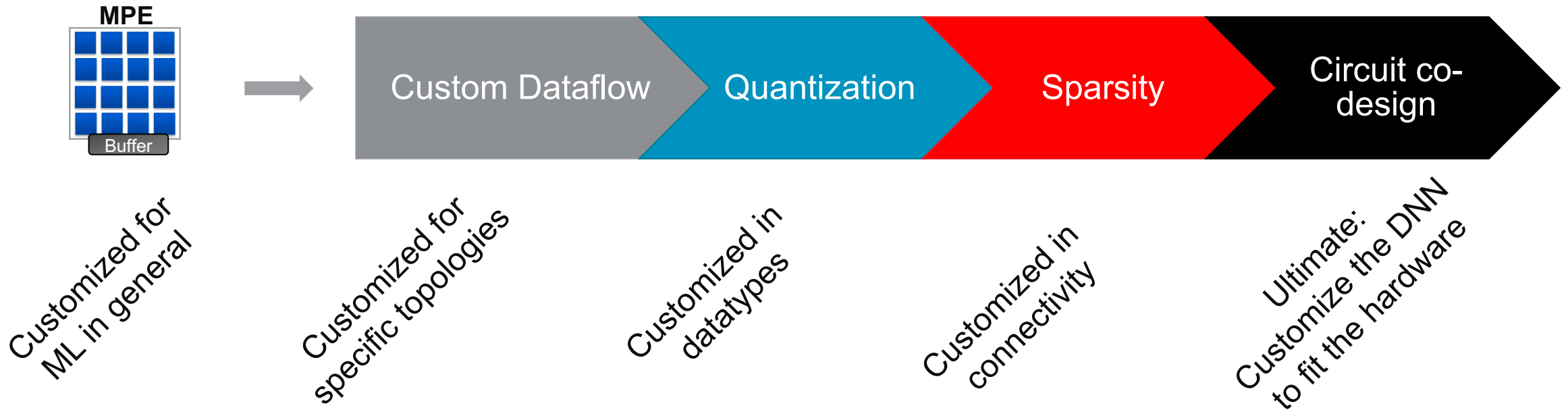
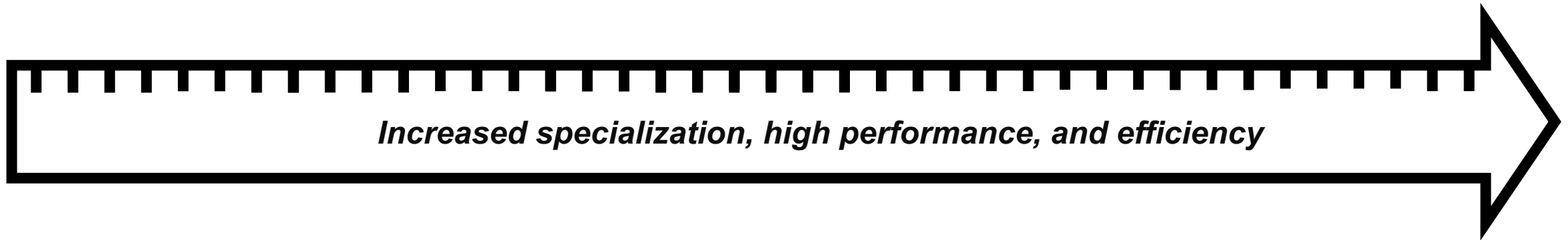


Granularity of Customizing Arithmetic

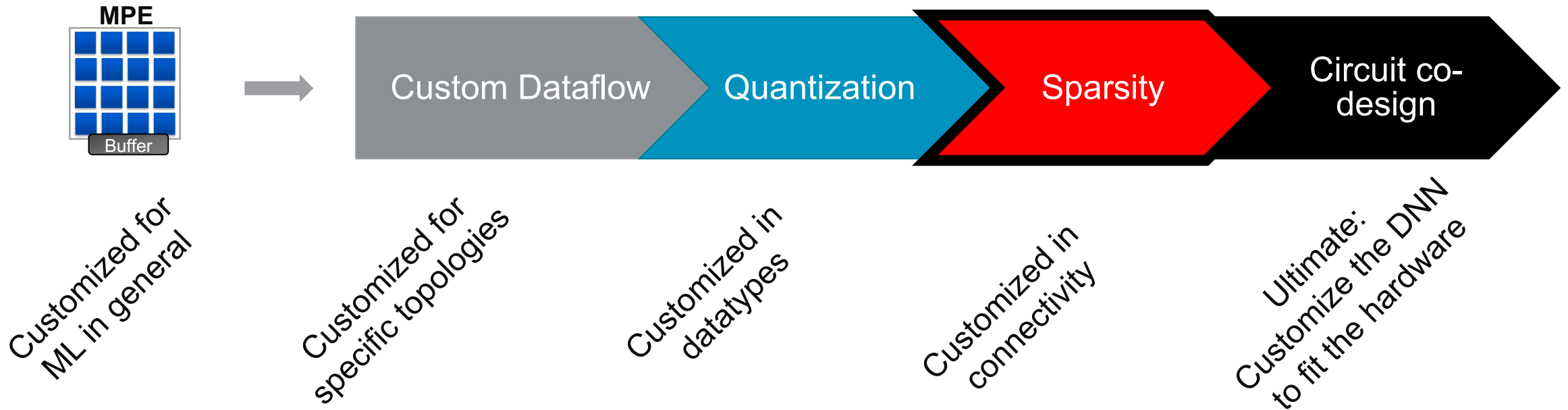
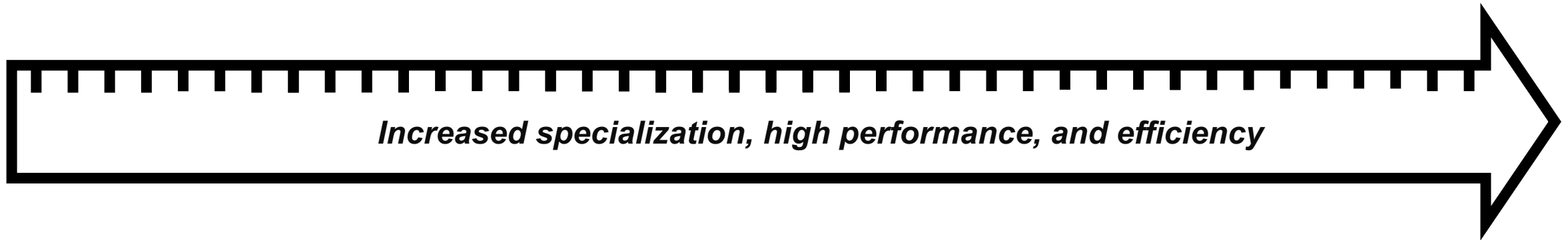


Dataflow architectures can exploit custom arithmetic at a finer granularity - even per-neuron and per-synapse custom arithmetic with full unfolding

Specialized FPGA Inference via Co-Design

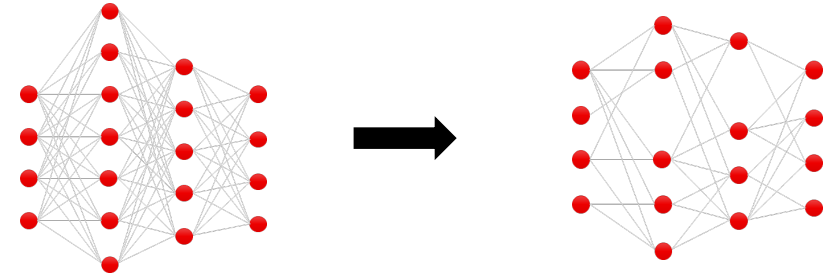


Specialized FPGA Inference via Co-Design



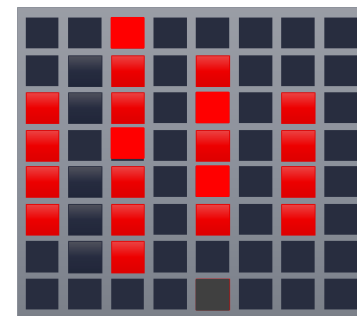
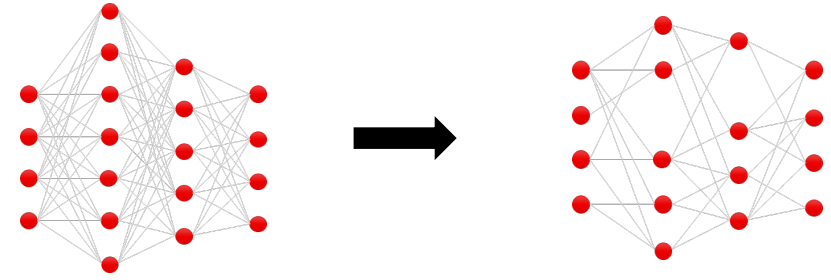
Sparsity

- 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

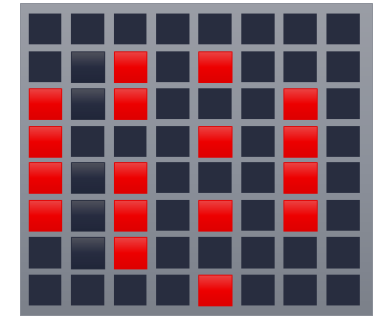


Sparsity

- 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



**Dense
Dataflow
on FPGA**



**Sparse
Dataflow
on FPGA**

Design Trade-offs with Quantization + Sparsity

Error vs Compute Cost

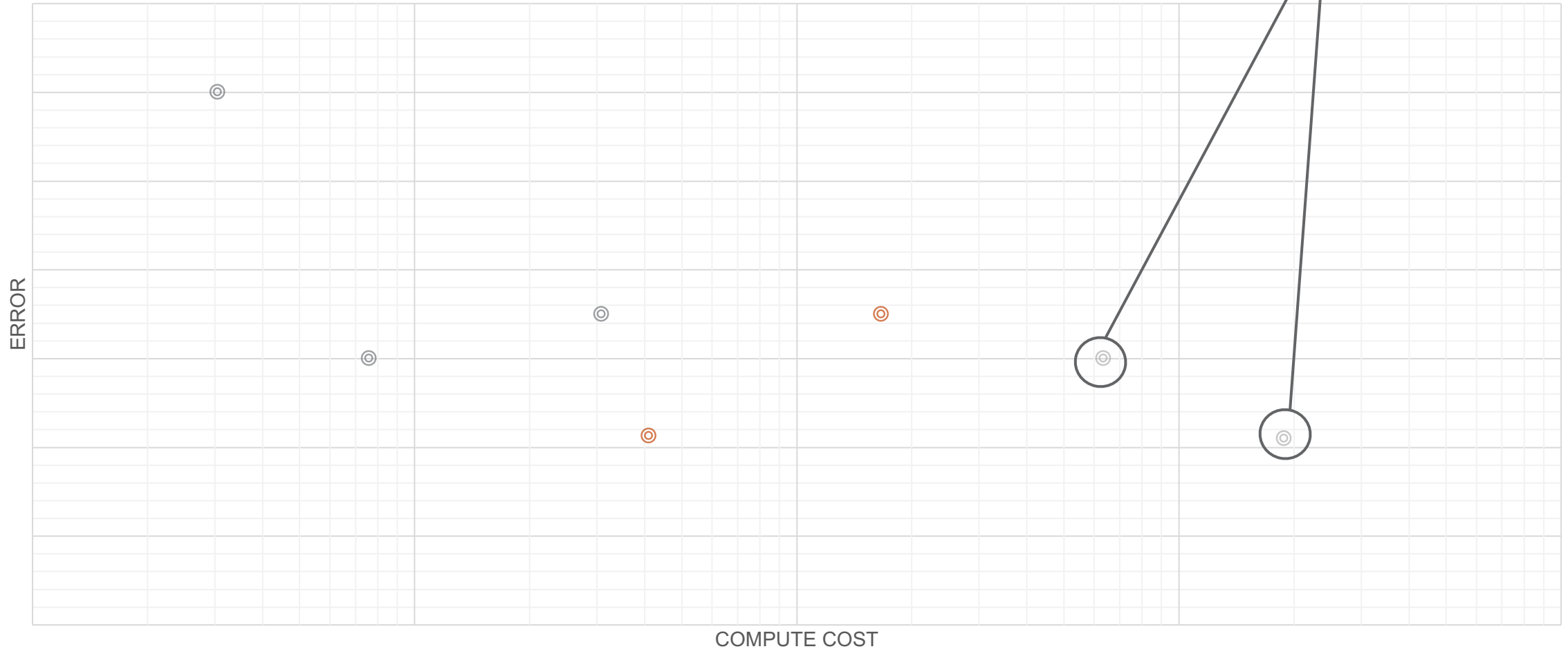
⊙ Float ⊙ 8-bit ⊙ Reduced Precision



Design Trade-offs with Quantization + Sparsity

Error vs Compute Cost

⊙ Float ⊙ 8-bit ⊙ Reduced Precision

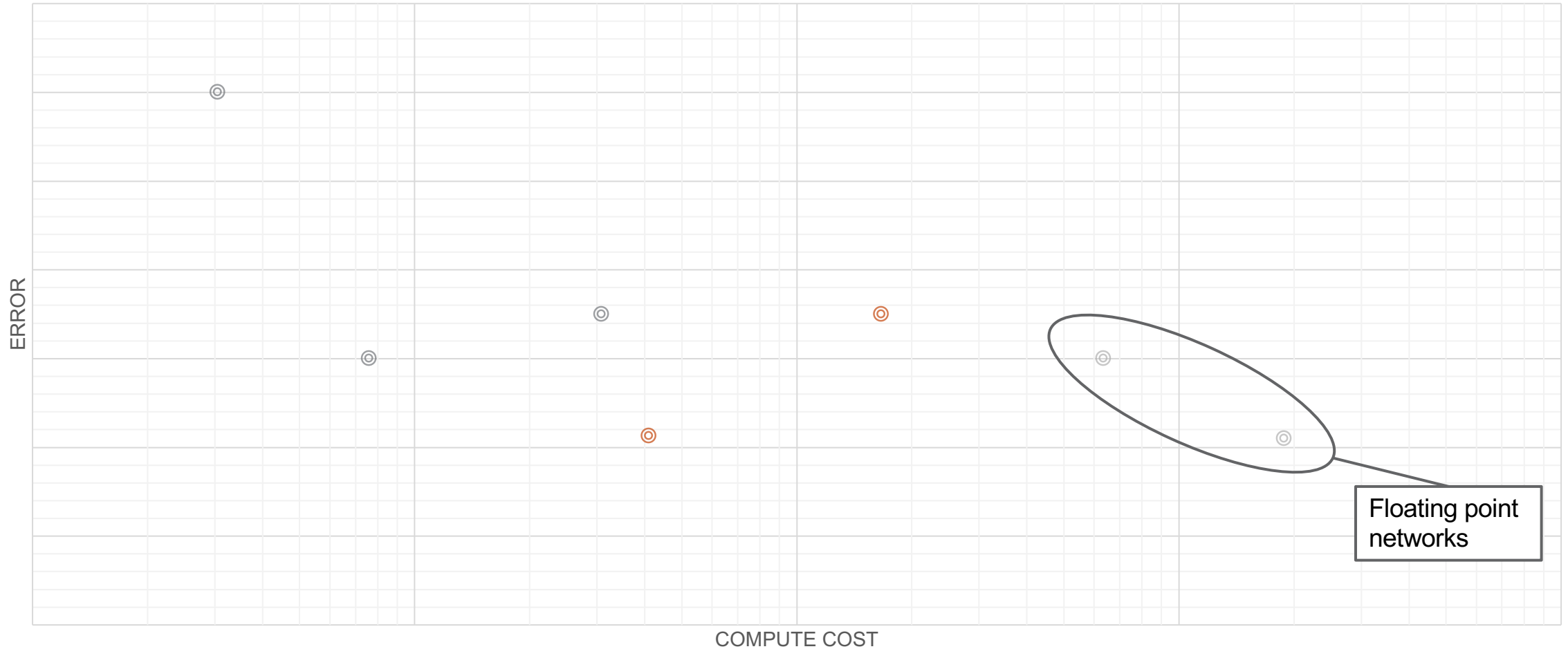


Different network topologies

Design Trade-offs with Quantization + Sparsity

Error vs Compute Cost

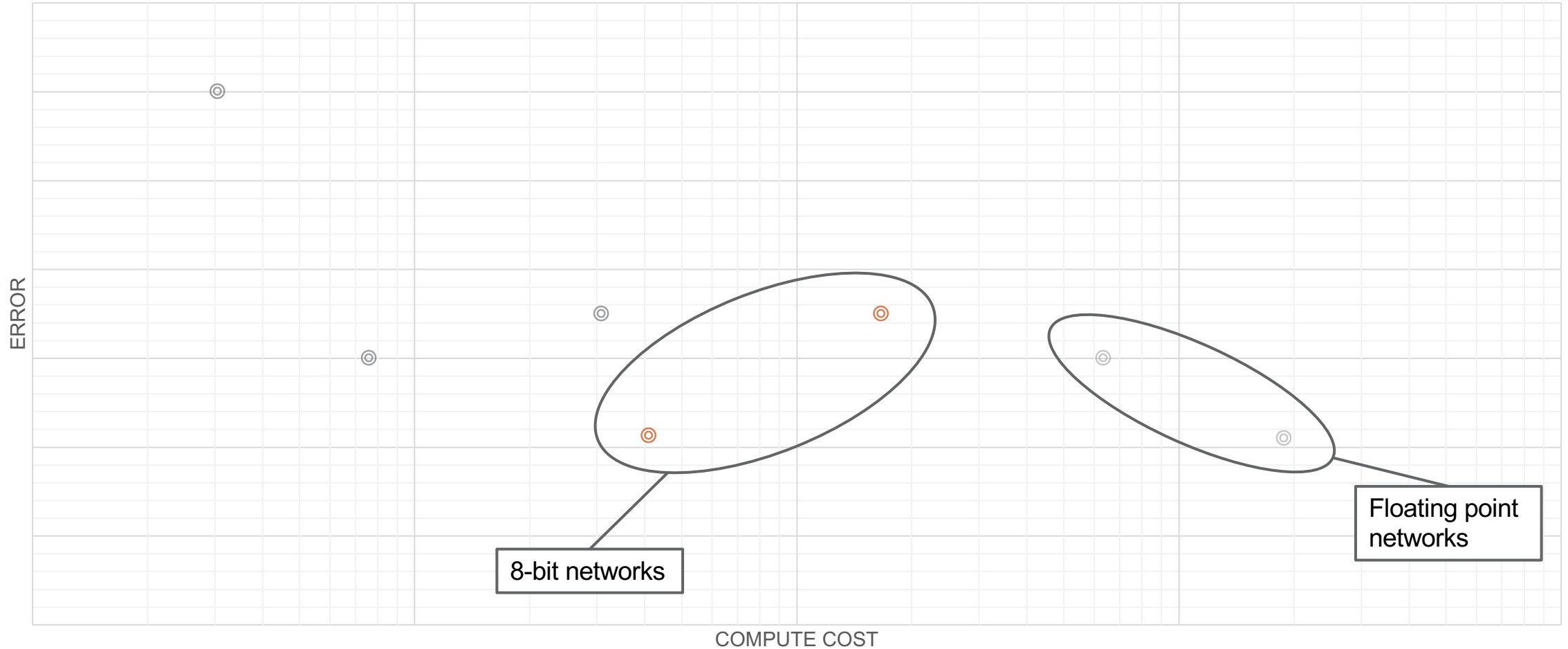
⊙ Float ⊙ 8-bit ⊙ Reduced Precision



Design Trade-offs with Quantization + Sparsity

Error vs Compute Cost

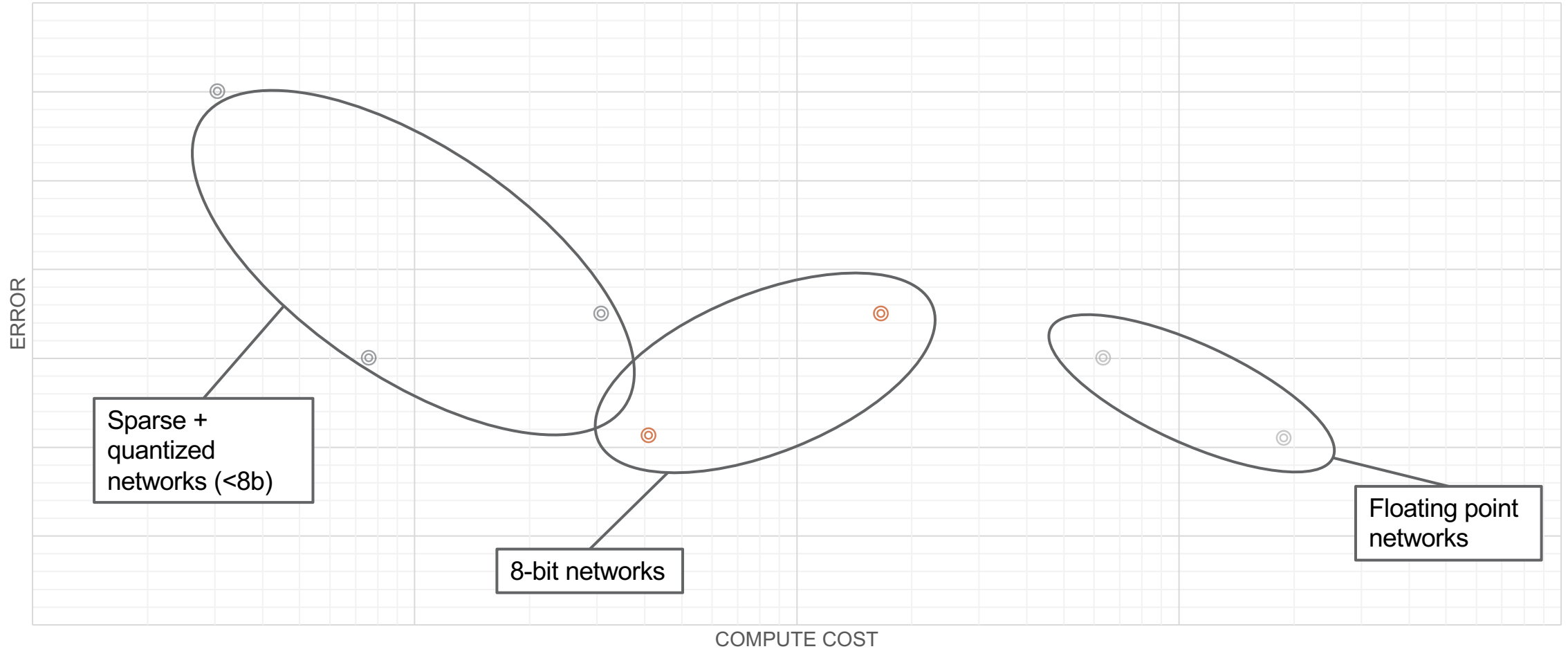
⊙ Float ⊙ 8-bit ⊙ Reduced Precision



Design Trade-offs with Quantization + Sparsity

Error vs Compute Cost

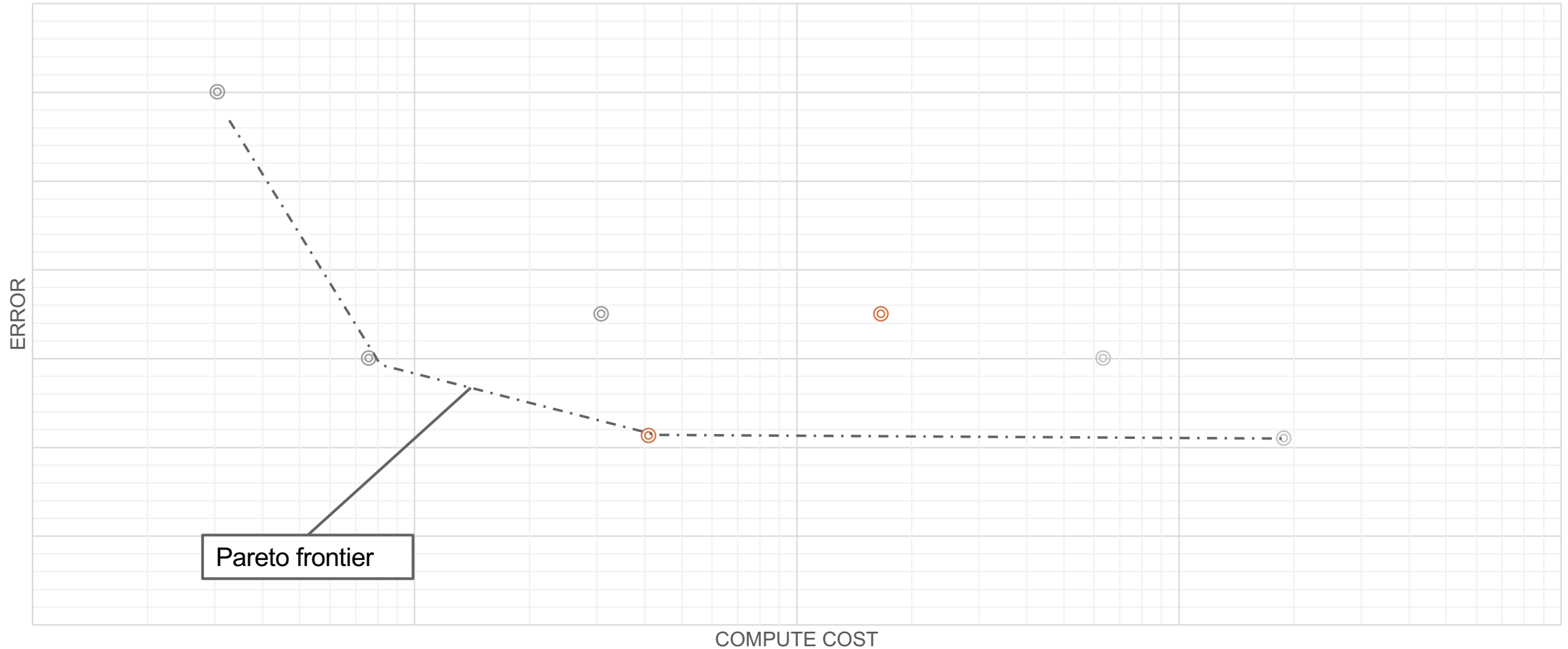
⊙ Float ⊙ 8-bit ⊙ Reduced Precision



Design Trade-offs with Quantization + Sparsity

Error vs Compute Cost

⊙ Float ⊙ 8-bit ⊙ Reduced Precision

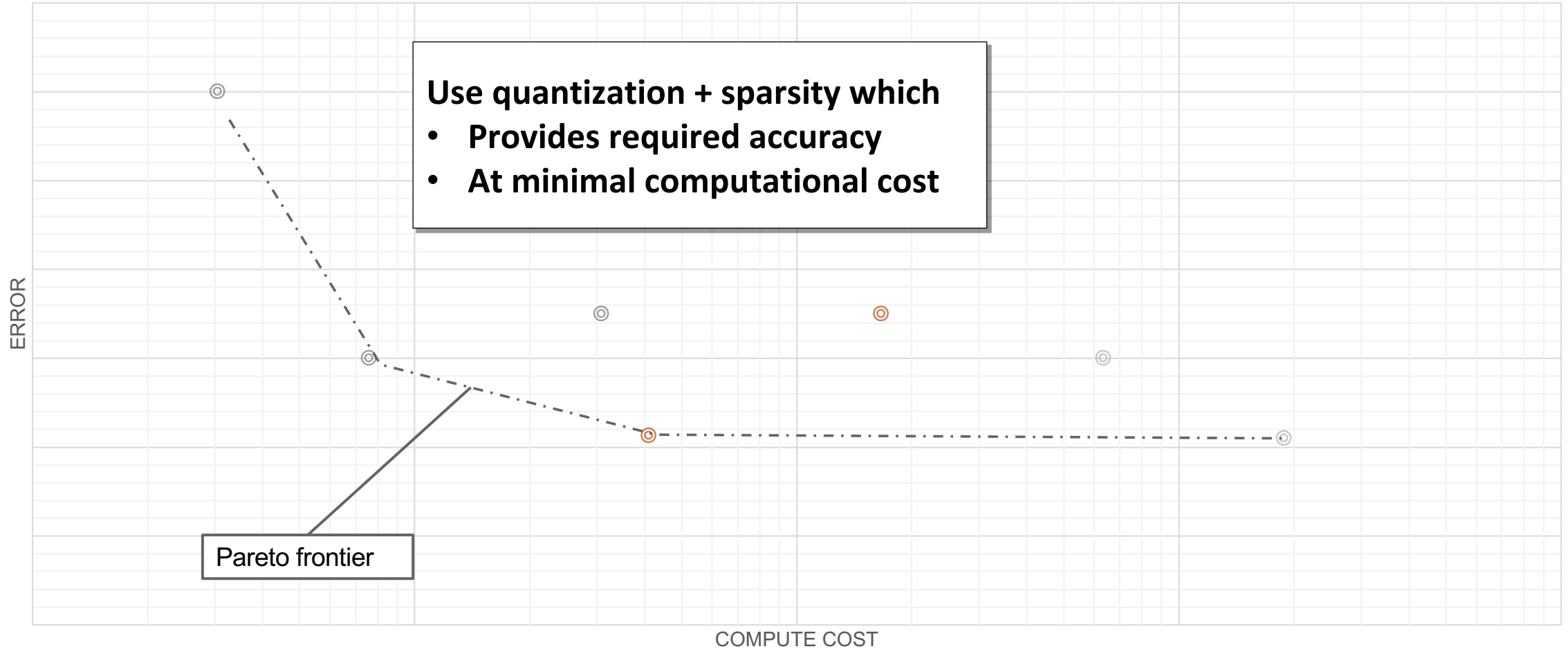


Pareto frontier

Design Trade-offs with Quantization + Sparsity

Error vs Compute Cost

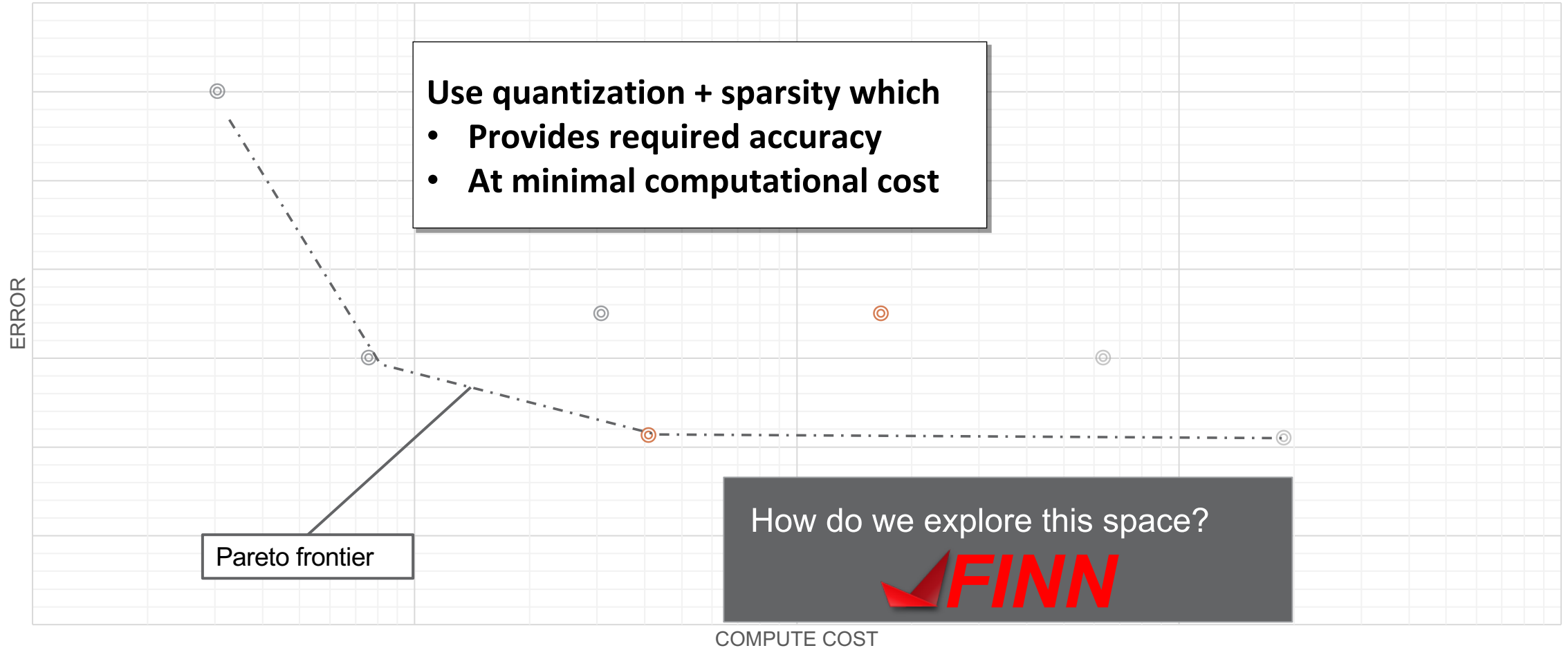
⊙ Float ⊙ 8-bit ⊙ Reduced Precision



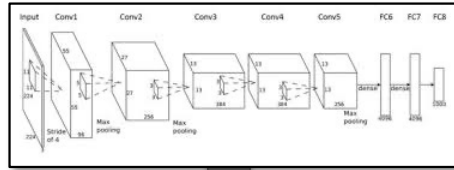
Design Trade-offs with Quantization + Sparsity

Error vs Compute Cost

⊙ Float ⊙ 8-bit ⊙ Reduced Precision



FINN Framework: From DNN to FPGA Deployment



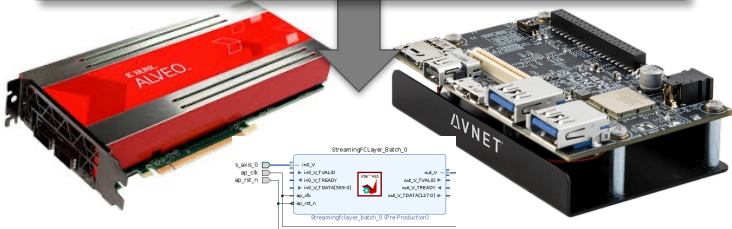
other quantizers
(QKeras, HAWQ)

Brevitas
Training in PyTorch
Algorithmic optimizations

QONNX

FINN Compiler
QNN-to-accelerator

Deployment
Verification, integration...

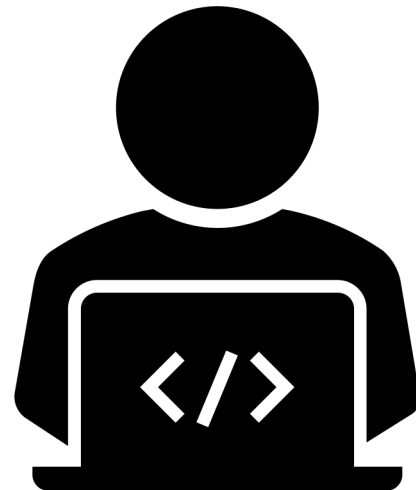


- Quantization-aware training for DNNs
- Library of common quantized layers
- Includes pre-trained examples
- Quantized NN exchange format + toolkit
- Perform optimizations
- Assemble parameterized HLS/RTL modules
- Generate a DNN hardware IP
- Run RTL testbenches to simulate IP
- System-level integration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

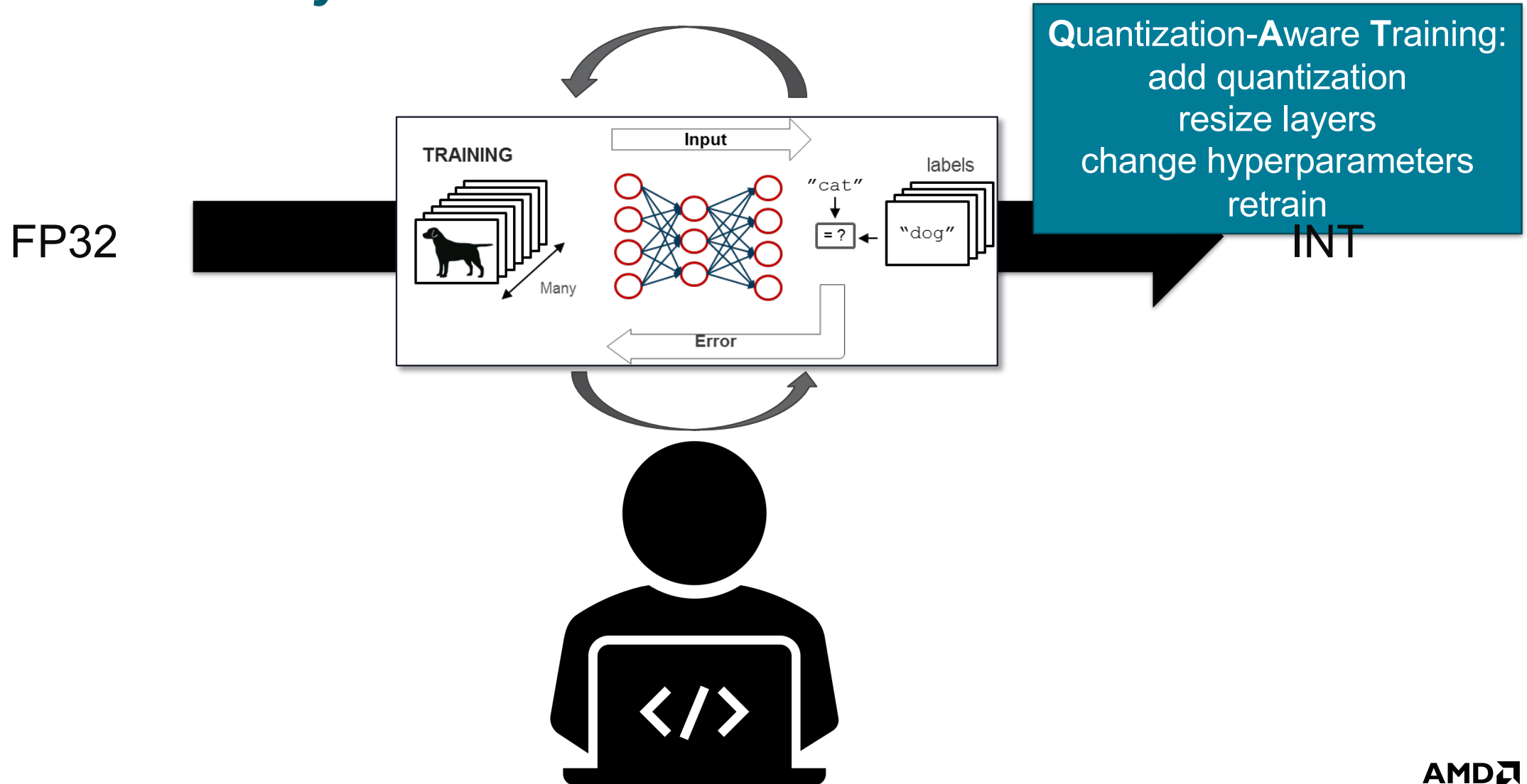
All open source & actively maintained

Brevitas:

A PyTorch Library for Neural Network Quantization

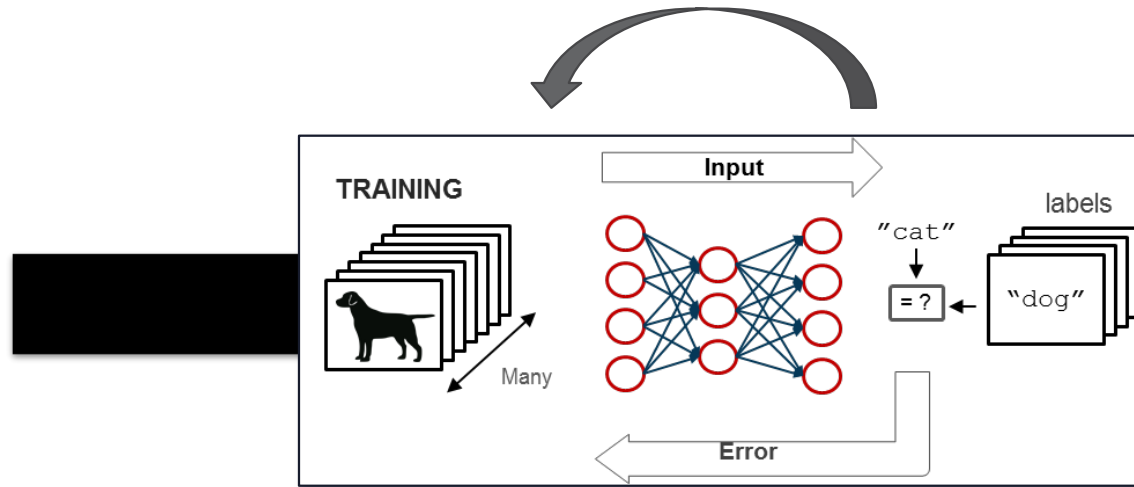


Brevitas: A PyTorch Library for Neural Network Quantization



Brevitas: A PyTorch Library for Neural Network Quantization

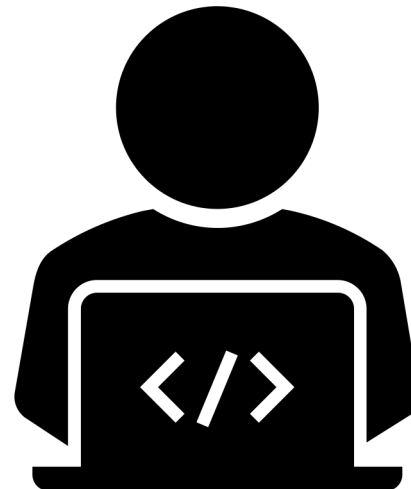
FP32



Quantization-Aware Training:
add quantization
resize layers
change hyperparameters
retrain

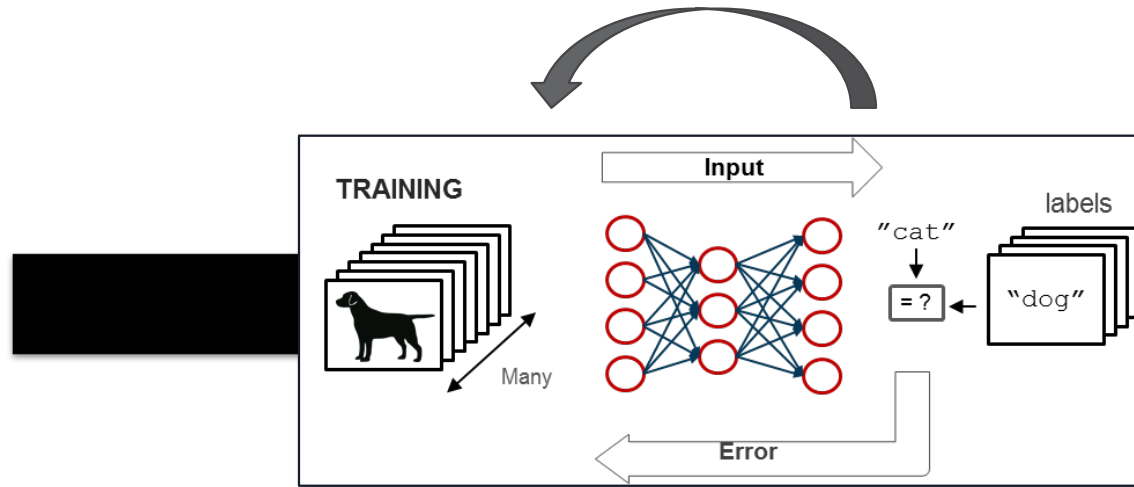
INT

Precision
Preset or
learned



Brevitas: A PyTorch Library for Neural Network Quantization

FP32

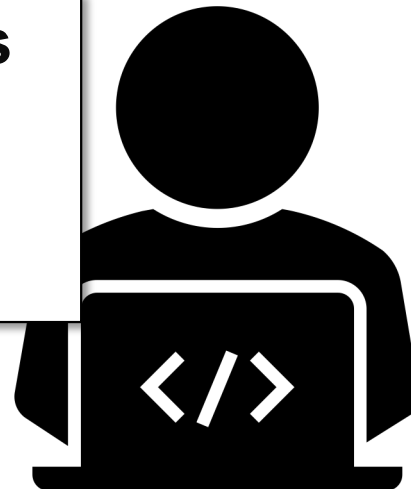


Quantization-Aware Training:
add quantization
resize layers
change hyperparameters
retrain

INT

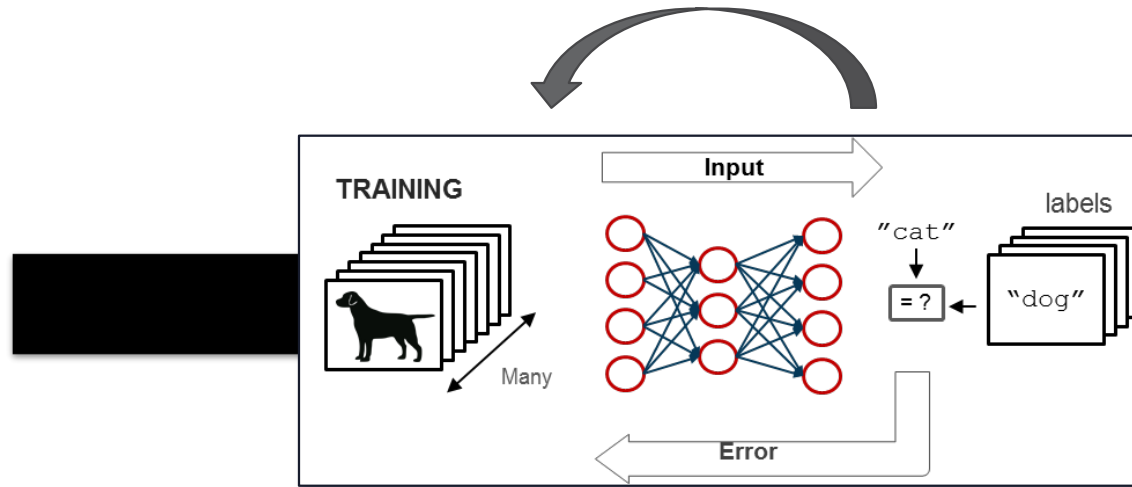
Precision
Preset or
learned

Scaling Factors
Granularities,
strategies and
constraints



Brevitas: A PyTorch Library for Neural Network Quantization

FP32



Quantization-Aware Training:
add quantization
resize layers
change hyperparameters
retrain

INT

Precision
Preset or learned

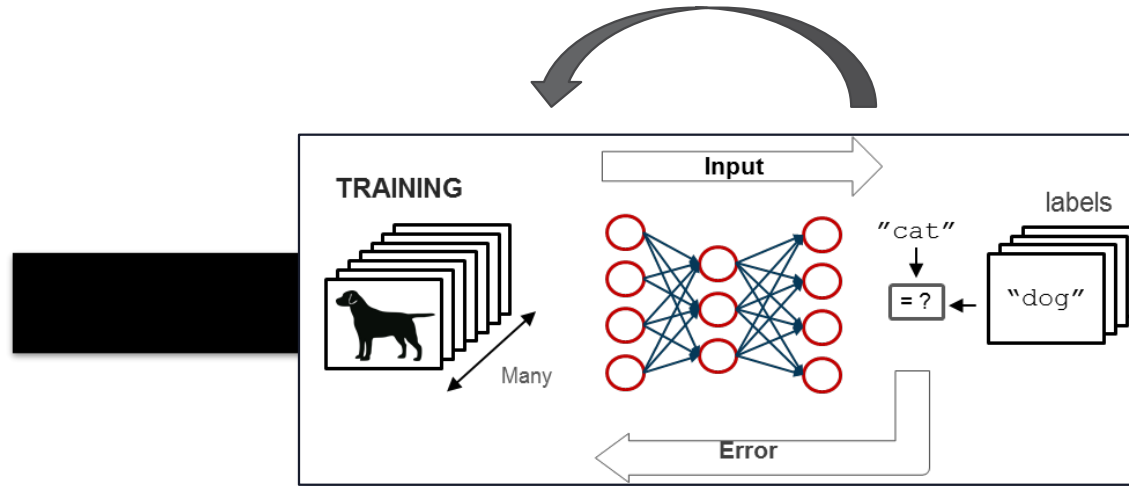
Scaling Factors
Granularities, strategies and constraints



Target Tensors
Weights, activations, **accumulators**

Brevitas: A PyTorch Library for Neural Network Quantization

FP32



Quantization-Aware Training:
 add quantization
 resize layers
 change hyperparameters
 retrain

INT

Precision
 Preset or
 learned

Scaling Factors
 Granularities,
 strategies and
 constraints

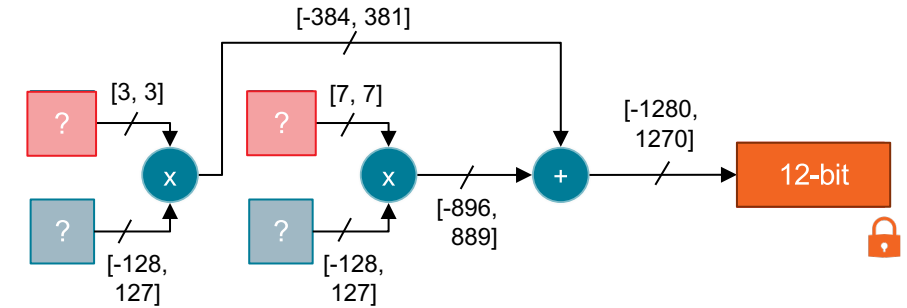


Target Tensors
 Weights,
 activations,
accumulators

Export to ONNX
 To import into the
 FINN compiler +
 various backends

A Brevitas showcase: Accumulator-Aware Quantization (A2Q)

- Cost of accumulators can be dominant for few-bit quantization
- Can we constrain weights to bound the max accumulator size?
 - Yes! Via Hölder’s inequality or zero-centered range analysis
 - See A2Q [7] and A2Q+ [8] for details 😊



• **A2Q and A2Q+ implementations are open-sourced as part of Brevitas**

- `>>> from brevitass.nn import QuantConv2d`
- `>>> from brevitass.quant import Int8AccumulatorAwareWeightQuant`
- `>>> conv = QuantConv2d(4, 4, 3, weight_quant=Int8AccumulatorAwareWeightQuant)`

Train from scratch!



Super Resolution
ESPCN on BSDS300

Train from checkpoint!



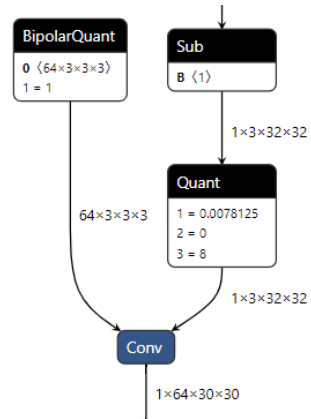
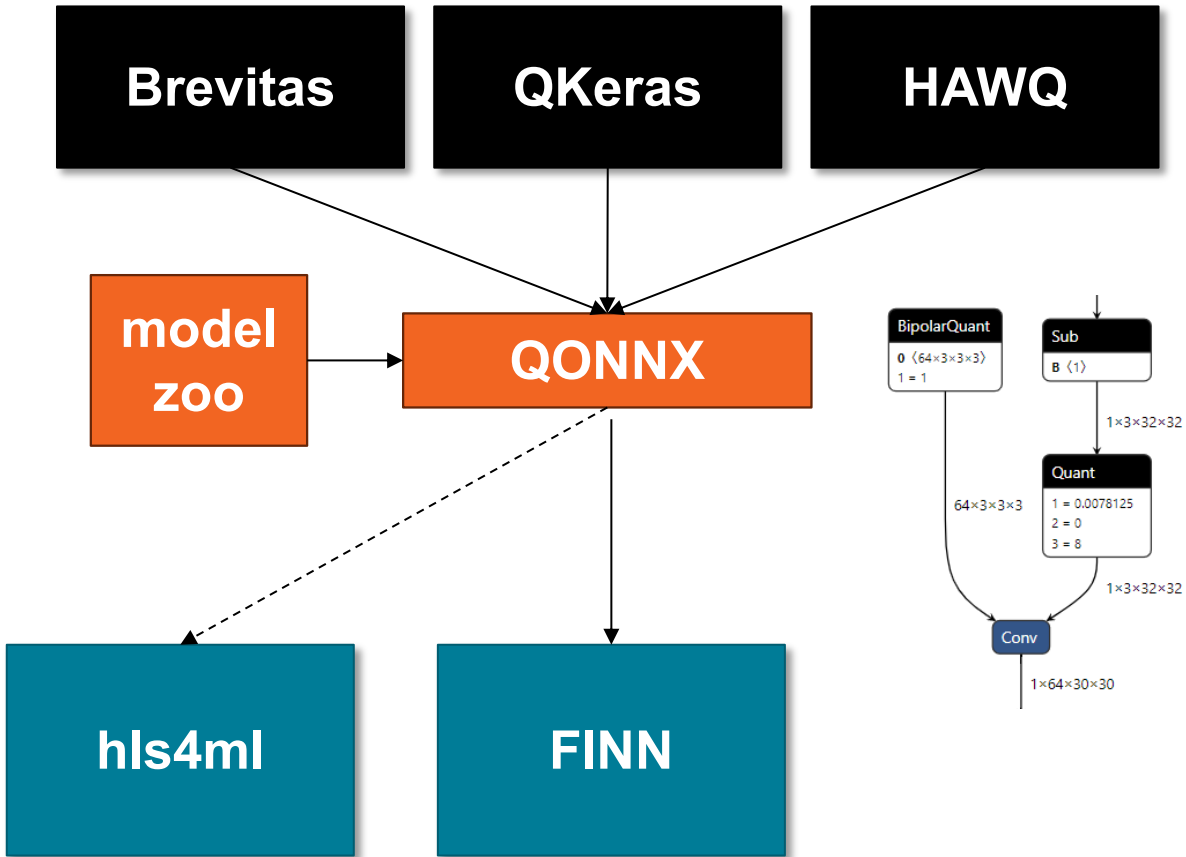
Image Classification
ResNet18 on CIFAR10

Network	Method	P	Top-1	Sparsity
ResNet50 (Float: 76.13%)	Base	32	75.9%	25.8%
	A2Q	16	76.0%	56.1%
		14	73.8%	77.2%
		12	55.0%	90.7%
	A2Q (w/ EP-init)	16	76.0%	56.1%
		14	74.5%	77.1%
		12	66.7%	88.6%
	A2Q+	16	76.0%	44.0%
		14	75.7%	67.7%
12		72.0%	84.4%	

4-bit weights and activations using 8-bit residuals and visible layers

QONNX: *Flexible quantized NNs in ONNX + related tools*

<https://github.com/fastmachinelearning/qonnx>

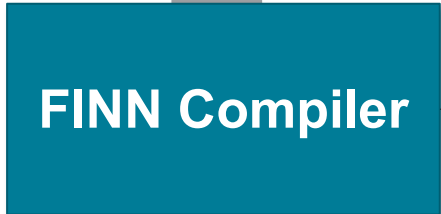


- Custom ONNX ops to represent arbitrary-bit uniform quantization
 - Standard ONNX only supports 8/16-bit
- ONNX-based common exchange format for QNNs
 - Meeting point between quantization frameworks and backends
- Infrastructure for manipulating + verifying custom ONNX graphs
- Including an own «model zoo» of quantized models
- Co-maintained by AMD RAD & FastML

FINN Compiler: From QONNX to hardware

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

Quantized NN (QONNX)



Vivado IP

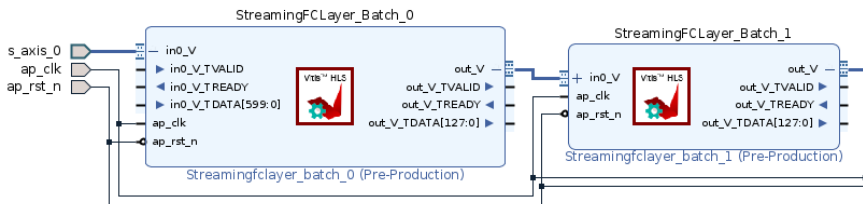
Build configuration

```

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",
  # ...
)

```

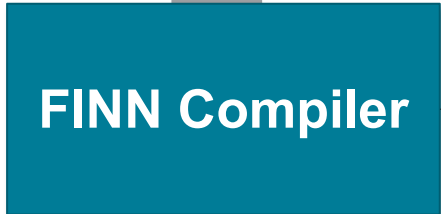
- ▶ Network optimizations: constant folding, streamlining
- ▶ Compute folding with respect to throughput and resource constraints
- ▶ Operator mapping and synthesis (via HLS and RTL op library)
- ▶ Assembly of pipelined dataflow IP with AXI stream interfaces



FINN Compiler: From QONNX to hardware

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

Quantized NN (QONNX)



Vivado IP

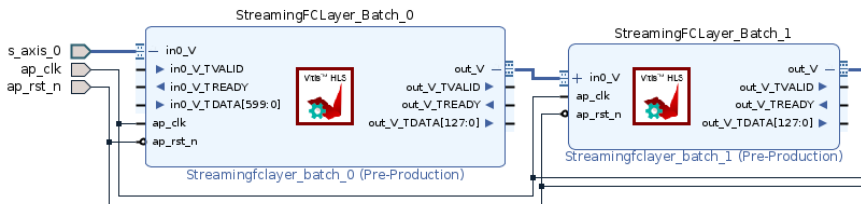
Build configuration

```

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



Many similarities and differences versus hls4ml
 Ongoing collaboration around common frontend (QONNX), knowledge sharing and joint publications since 2020

NIDS Results



Matrix of Processing Engines

Dataflow + Quantization + Sparsity

Topology / #layers / #OPs
Datatype
Accuracy

Vitis AI
MLP / 3 / 92 kOPs
8b & 8b
92.3%

Performance
Throughput
Latency (compute only)

22 kinfps
26 us

Resources
Compute (kLUTs, DSPs*)
Memory (BRAM, URAM**)
Clock

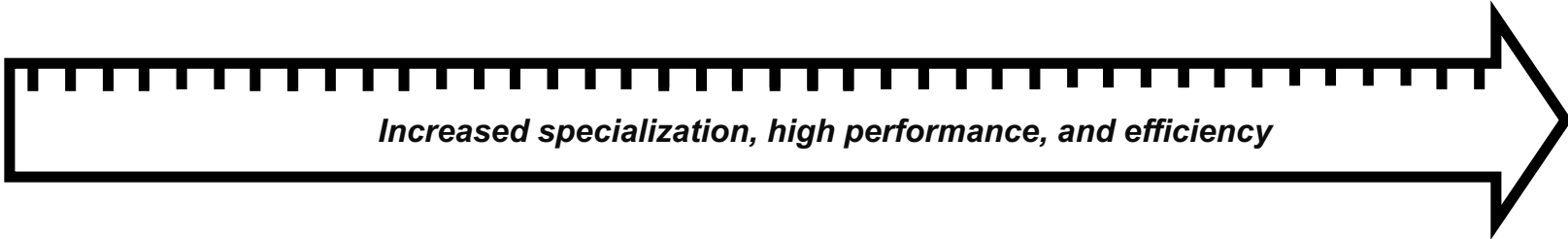
122,1124
290, 92
300/600 MHz

Mapped on UltraScale+, 16nm FPGA, all within the same SLR.

*DSPs: 8b or 16b Multiply Accumulates

**BRAMs: 36kb, URAM: 288kbit embedded SRAM blocks

NIDS Results




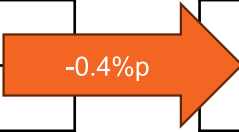
Matrix of Processing Engines

Dataflow + Quantization + Sparsity

Topology / #layers / #OPs
Datatype
Accuracy

Vitis AI
MLP / 3 / 92 kOPs
8b & 8b
92.3%

 FINN
MLP / 3 / 92 kOPs
2b & 2b
91.9%



Performance
Throughput
Latency (compute only)

22 kinfps
26 us

Resources
Compute (kLUTs, DSPs*)
Memory (BRAM, URAM**)
Clock

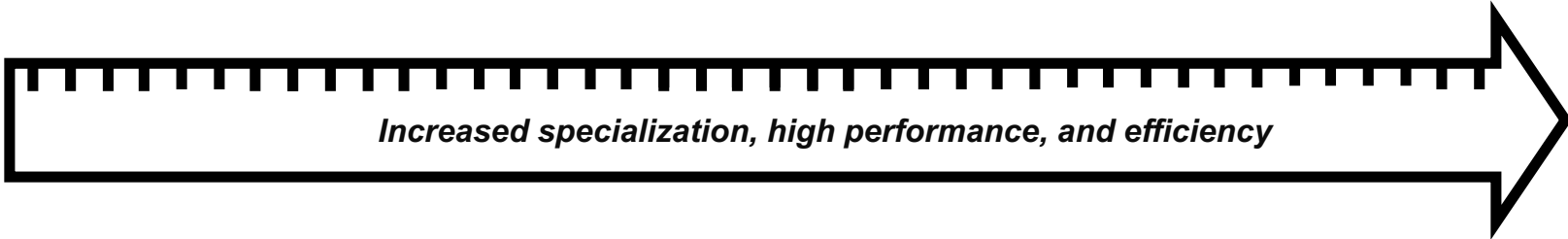
122,1124
290, 92
300/600 MHz

Mapped on UltraScale+, 16nm FPGA, all within the same SLR.

*DSPs: 8b or 16b Multiply Accumulates

**BRAMs: 36kb, URAM: 288kbit embedded SRAM blocks

NIDS Results




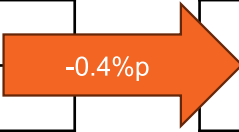
Matrix of Processing Engines

Dataflow + Quantization + Sparsity

Topology / #layers / #OPs
Datatype
Accuracy

Vitis AI
MLP / 3 / 92 kOPs
8b & 8b
92.3%

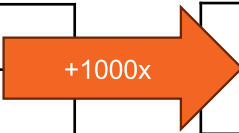
 FINN
MLP / 3 / 92 kOPs
2b & 2b
91.9%



Performance
Throughput
Latency (compute only)

22 kinfps
26 us

Fold 8
25.3 Minfps
160 ns



Resources
Compute (kLUTs, DSPs*)
Memory (BRAM, URAM**)
Clock

122,1124
290, 92
300/600 MHz

Mapped on UltraScale+, 16nm FPGA, all within the same SLR.

*DSPs: 8b or 16b Multiply Accumulates

**BRAMs: 36kb, URAM: 288kbit embedded SRAM blocks

NIDS Results



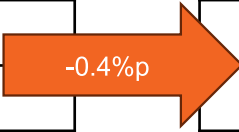
Matrix of Processing Engines

Dataflow + Quantization + Sparsity

Topology / #layers / #OPs
Datatype
Accuracy

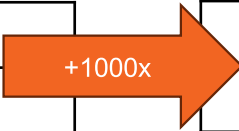
Vitis AI
MLP / 3 / 92 kOPs
8b & 8b
92.3%

FINN
MLP / 3 / 92 kOPs
2b & 2b
91.9%



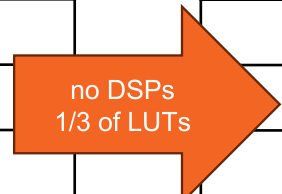
Performance
Throughput
Latency (compute only)

22 kinfps	Fold 8
26 us	25.3 Minfps
	160 ns



Resources
Compute (kLUTs, DSPs*)
Memory (BRAM, URAM**)
Clock

122,1124	44, 0
290, 92	166, 0
300/600 MHz	203 MHz

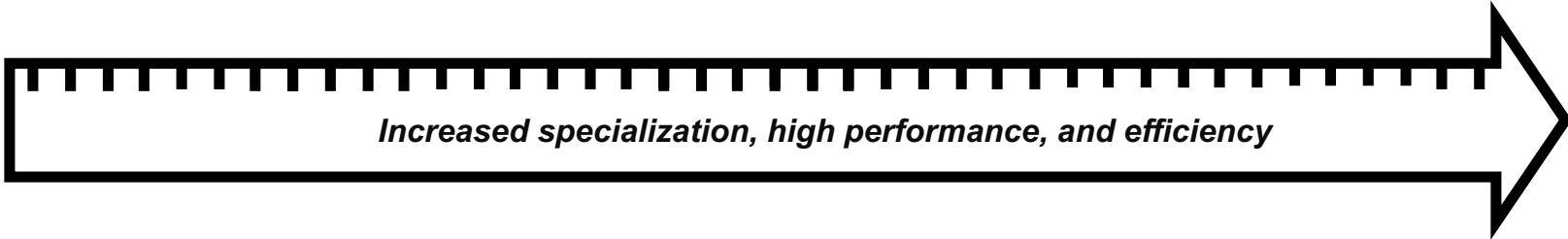


Mapped on UltraScale+, 16nm FPGA, all within the same SLR.

*DSPs: 8b or 16b Multiply Accumulates

**BRAMs: 36kb, URAM: 288kbit embedded SRAM blocks

NIDS Results



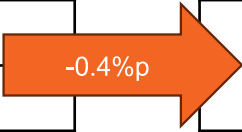
Matrix of Processing Engines

Dataflow + Quantization + Sparsity

Topology / #layers / #OPs
Datatype
Accuracy

Vitis AI
MLP / 3 / 92 kOPs
8b & 8b
92.3%

FINN
MLP / 3 / 92 kOPs
2b & 2b
91.9%

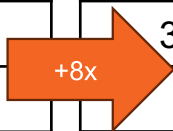
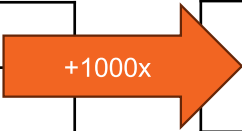


Performance
Throughput
Latency (compute only)

22 kinfps
26 us

Fold 8
25.3 Minfps
160 ns

Unfolded
300 Minfps
18 ns

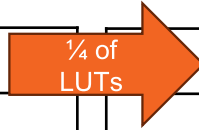
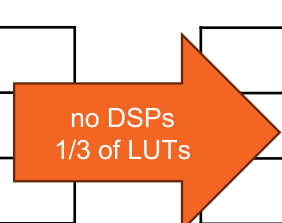


Resources
Compute (kLUTs, DSPs*)
Memory (BRAM, URAM**)
Clock

122,1124
290, 92
300/600 MHz

44, 0
166, 0
203 MHz

10, 0
0, 0
300 MHz

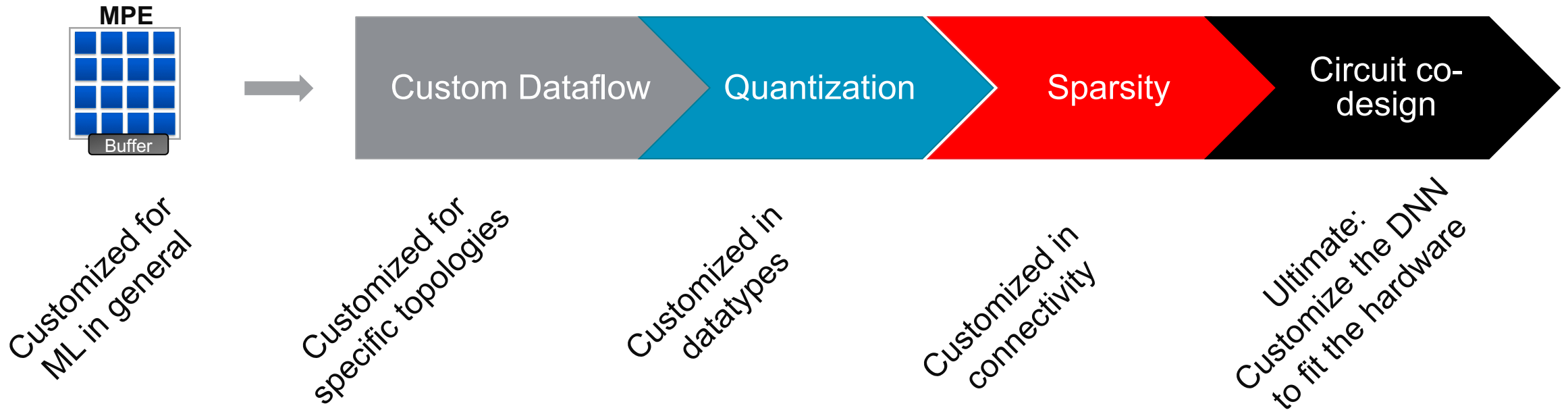
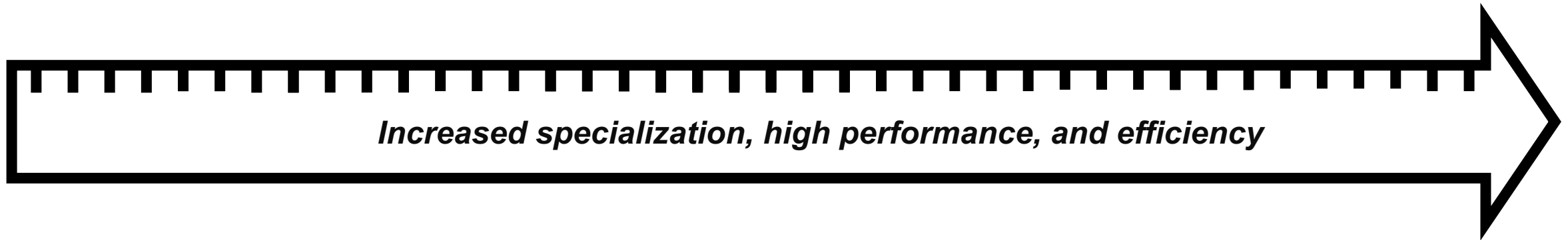


Mapped on UltraScale+, 16nm FPGA, all within the same SLR.

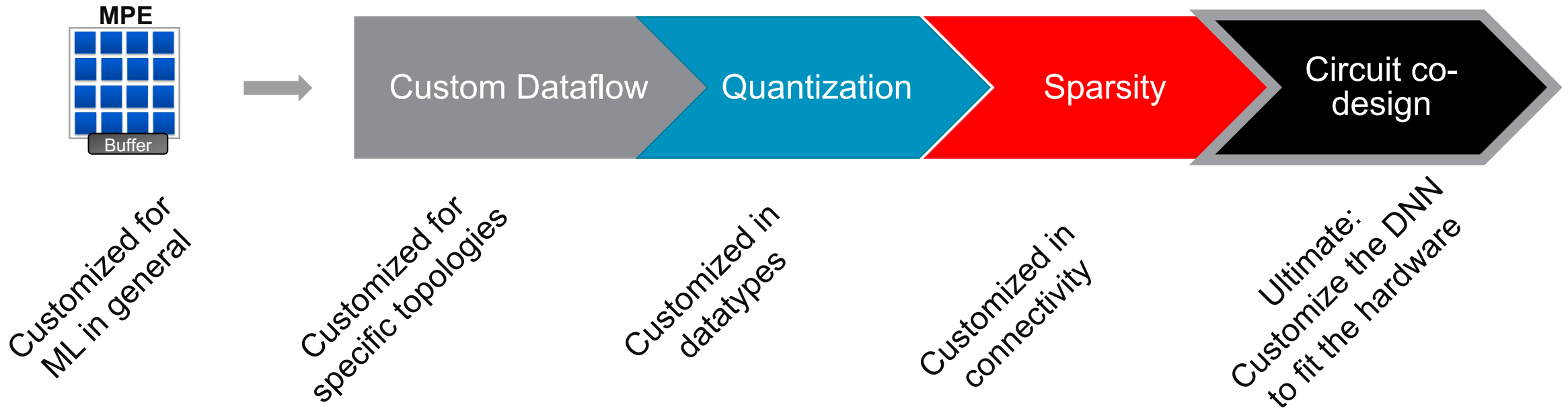
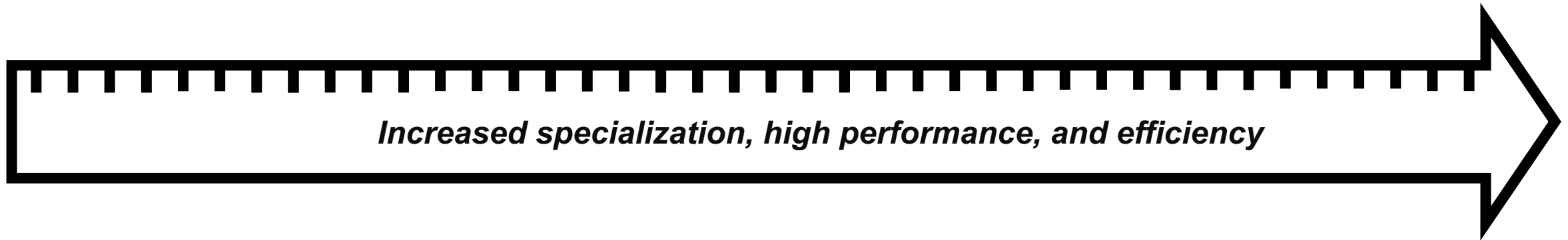
*DSPs: 8b or 16b Multiply Accumulates

**BRAMs: 36kb, URAM: 288kbit embedded SRAM blocks

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

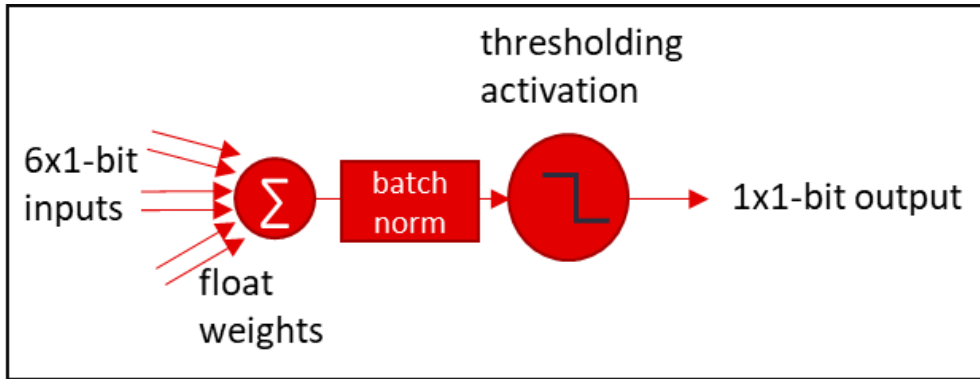
PyTorch

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

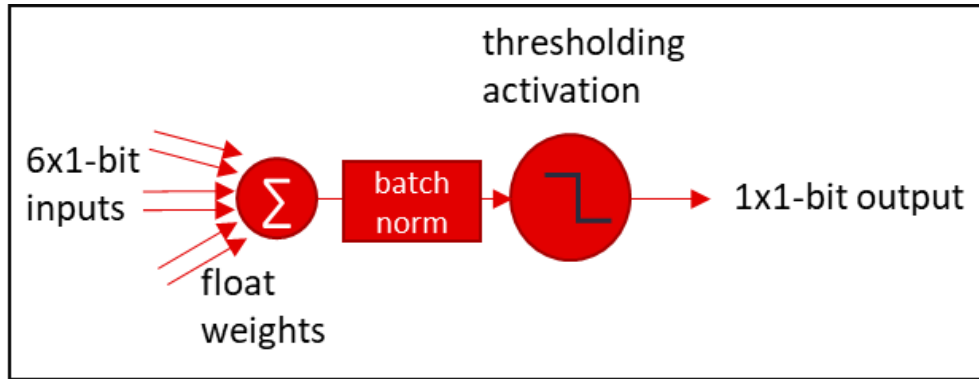
PyTorch

FPGA



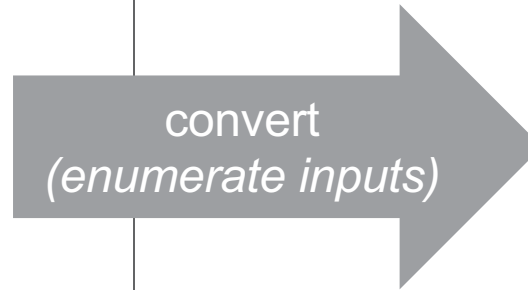
Total input: 6 bits
Total output: 1 bit

Quantized Neurons as Truth Tables



Total dynamic input *in_bits*: 6 bits
Total output *out_bits*: 1 bit

PyTorch

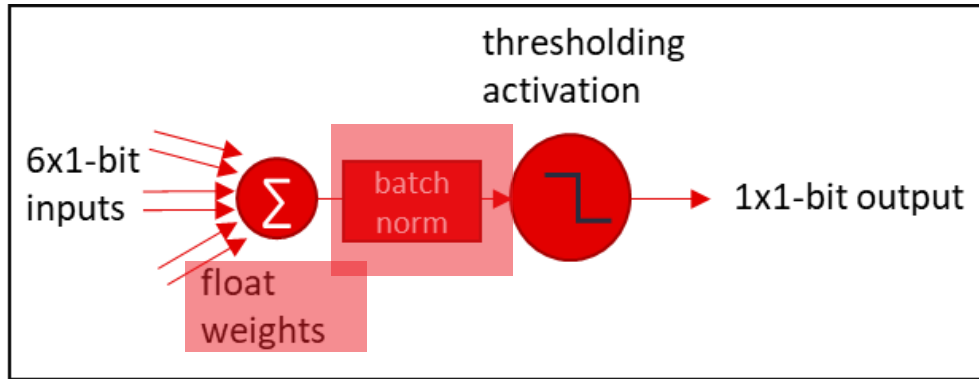


FPGA



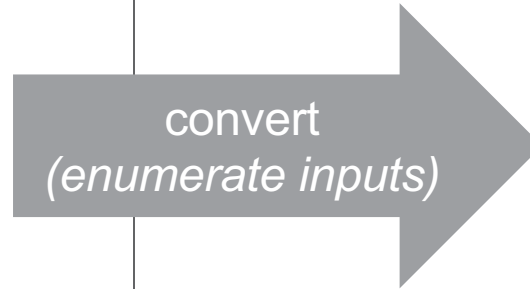
Total input: 6 bits
Total output: 1 bit

Quantized Neurons as Truth Tables



Total dynamic input *in_bits*: 6 bits
Total output *out_bits*: 1 bit

PyTorch

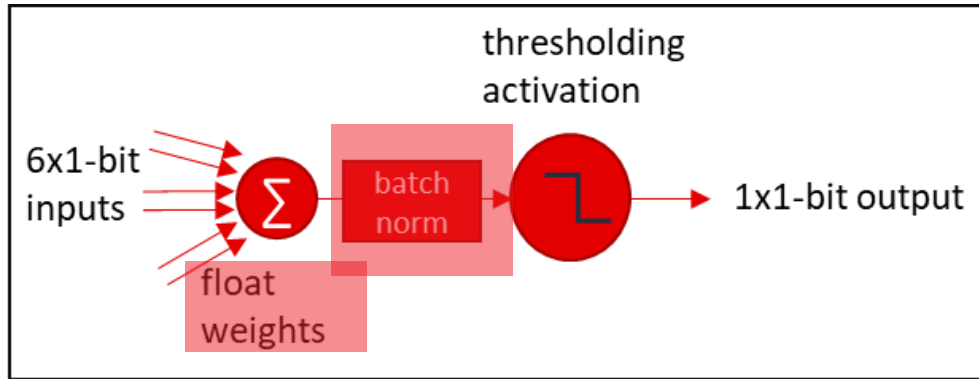


Total input: 6 bits
Total output: 1 bit

FPGA

Quantized Neurons as Truth Tables

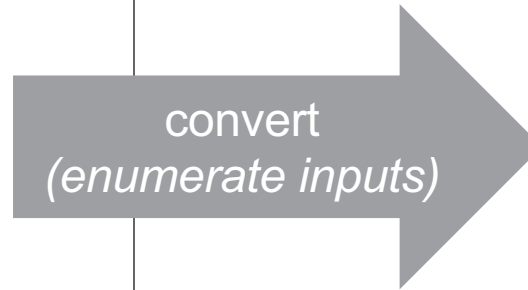
Neuron Equivalent (NEQ)



Total dynamic input *in_bits*: 6 bits
Total output *out_bits*: 1 bit

PyTorch

Hardware Building Block (HBB)

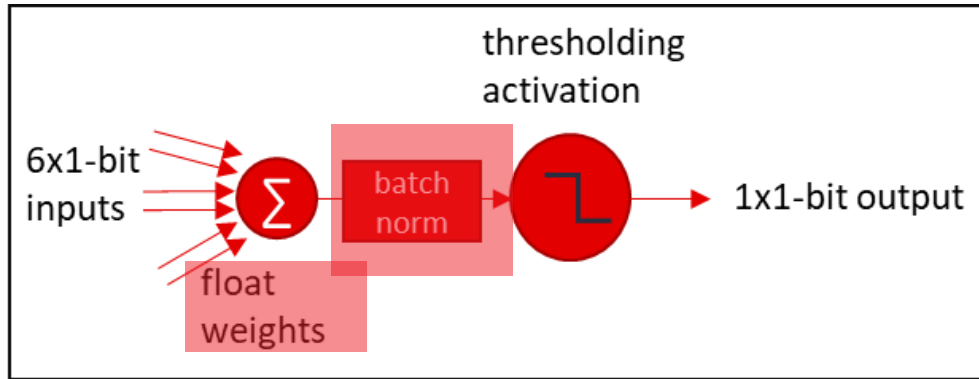


Total input: 6 bits
Total output: 1 bit

FPGA

Quantized Neurons as Truth Tables

Neuron Equivalent (NEQ)



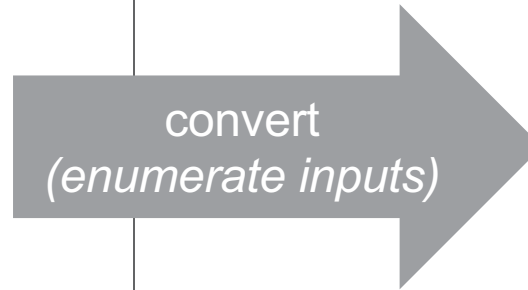
Total dynamic input in_bits : 6 bits
Total output out_bits : 1 bit

Hardware cost: 1 x LUT6
Generalized cost: $out_bits \cdot 2^{in_bits}$

PyTorch

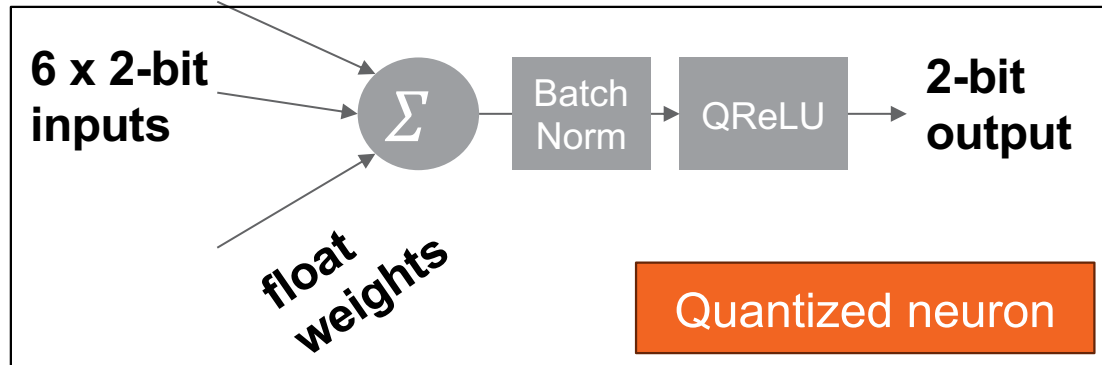
FPGA

Hardware Building Block (HBB)



Total input: 6 bits
Total output: 1 bit

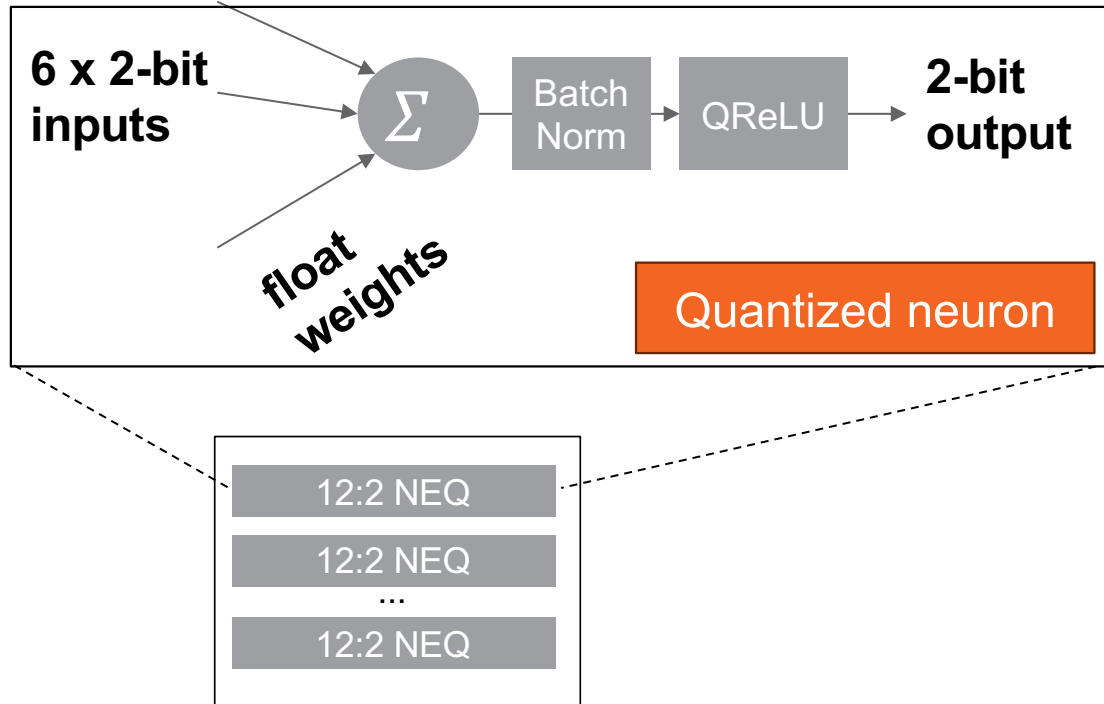
LogicNets at a Glance



PyTorch

FPGA

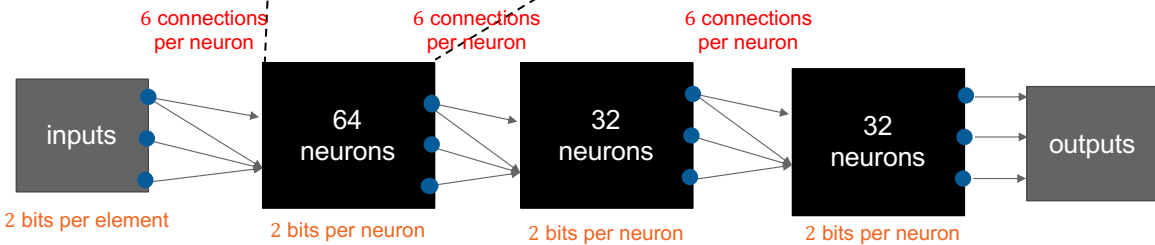
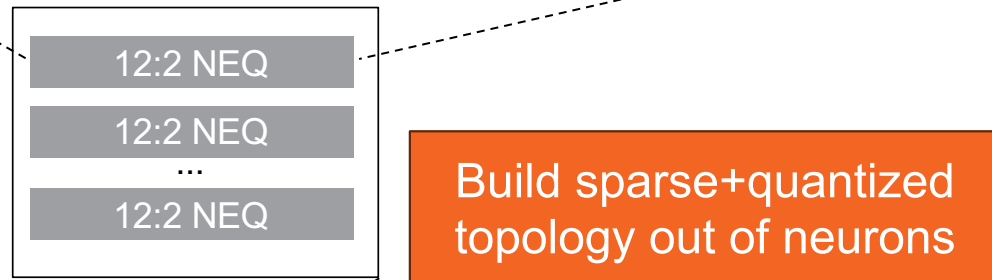
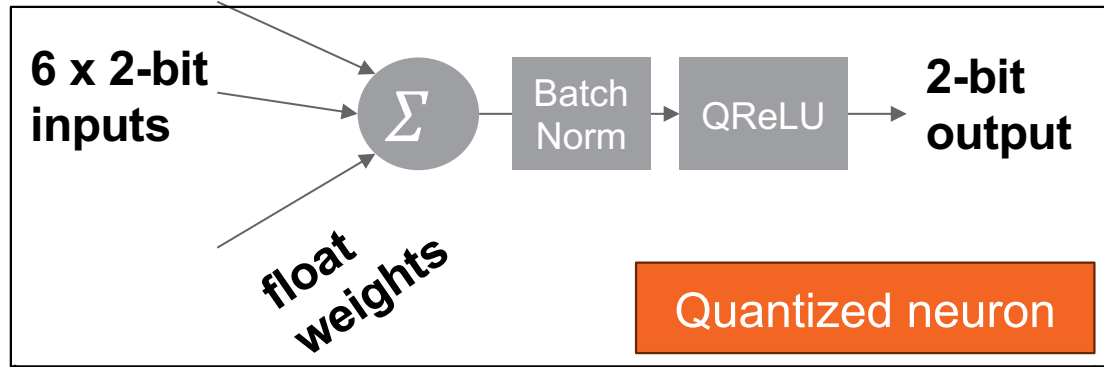
LogicNets at a Glance



PyTorch

FPGA

LogicNets at a Glance

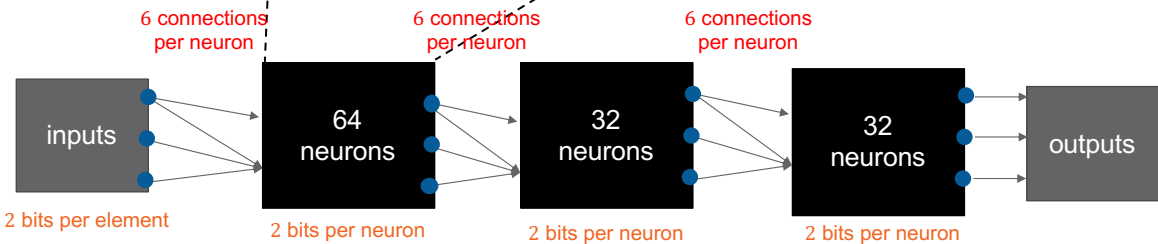
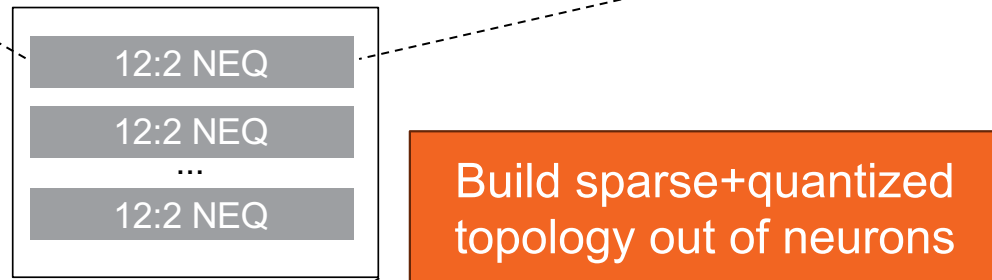
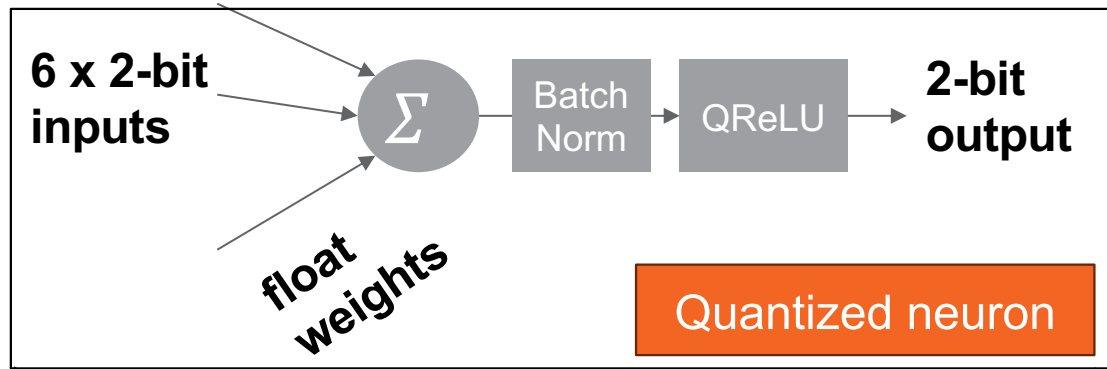


Train in ML framework

PyTorch

FPGA

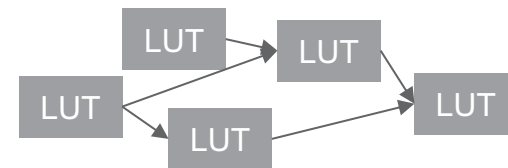
LogicNets at a Glance



Train in ML framework

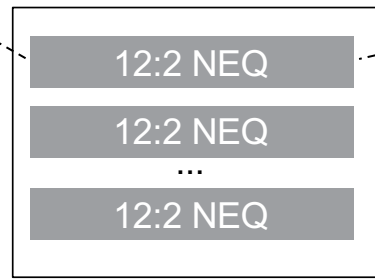
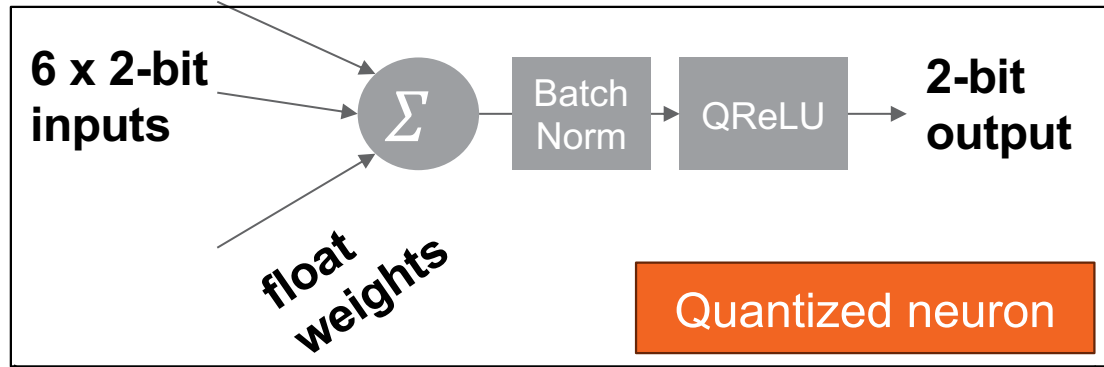
PyTorch

FPGA

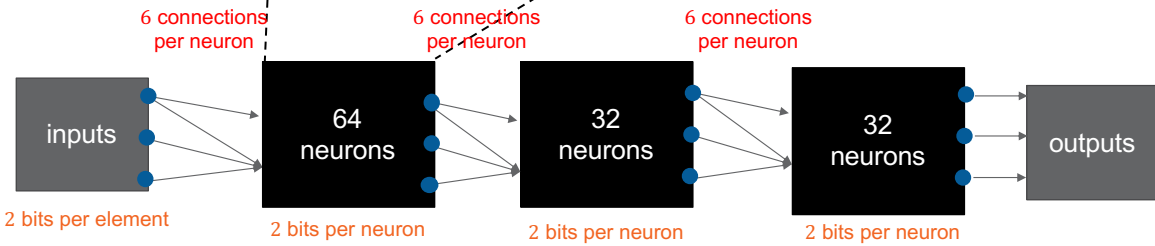


Convert to sparse LUT circuit

LogicNets at a Glance



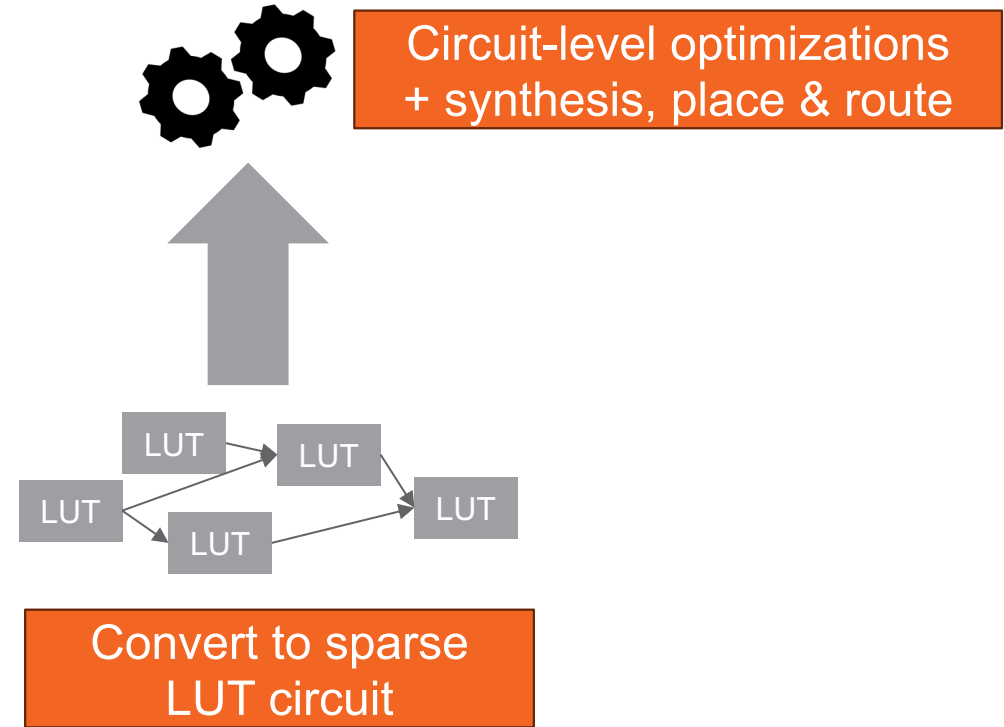
Build sparse+quantized topology out of neurons



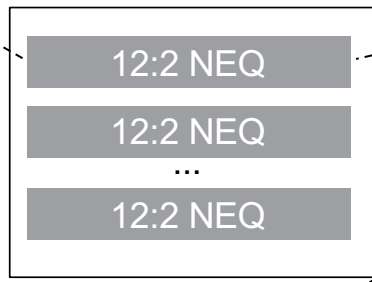
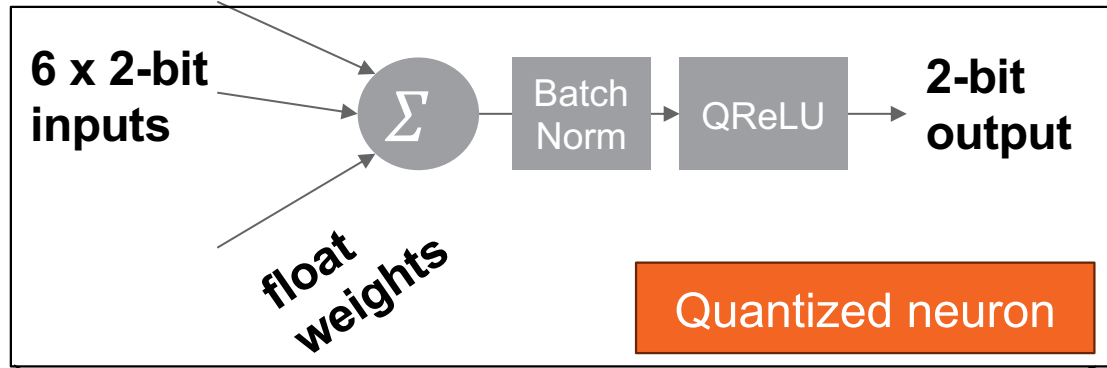
Train in ML framework

PyTorch

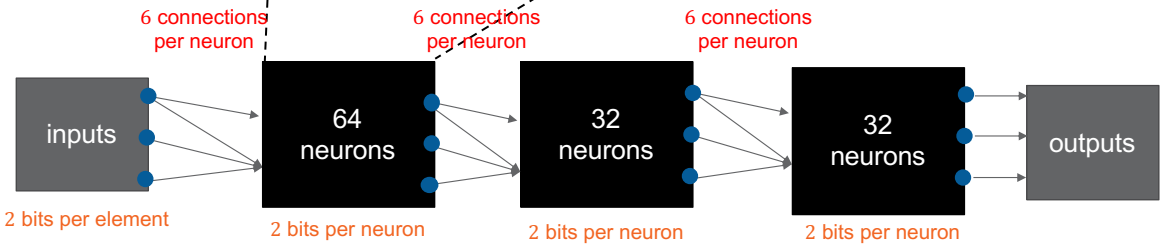
FPGA



LogicNets at a Glance

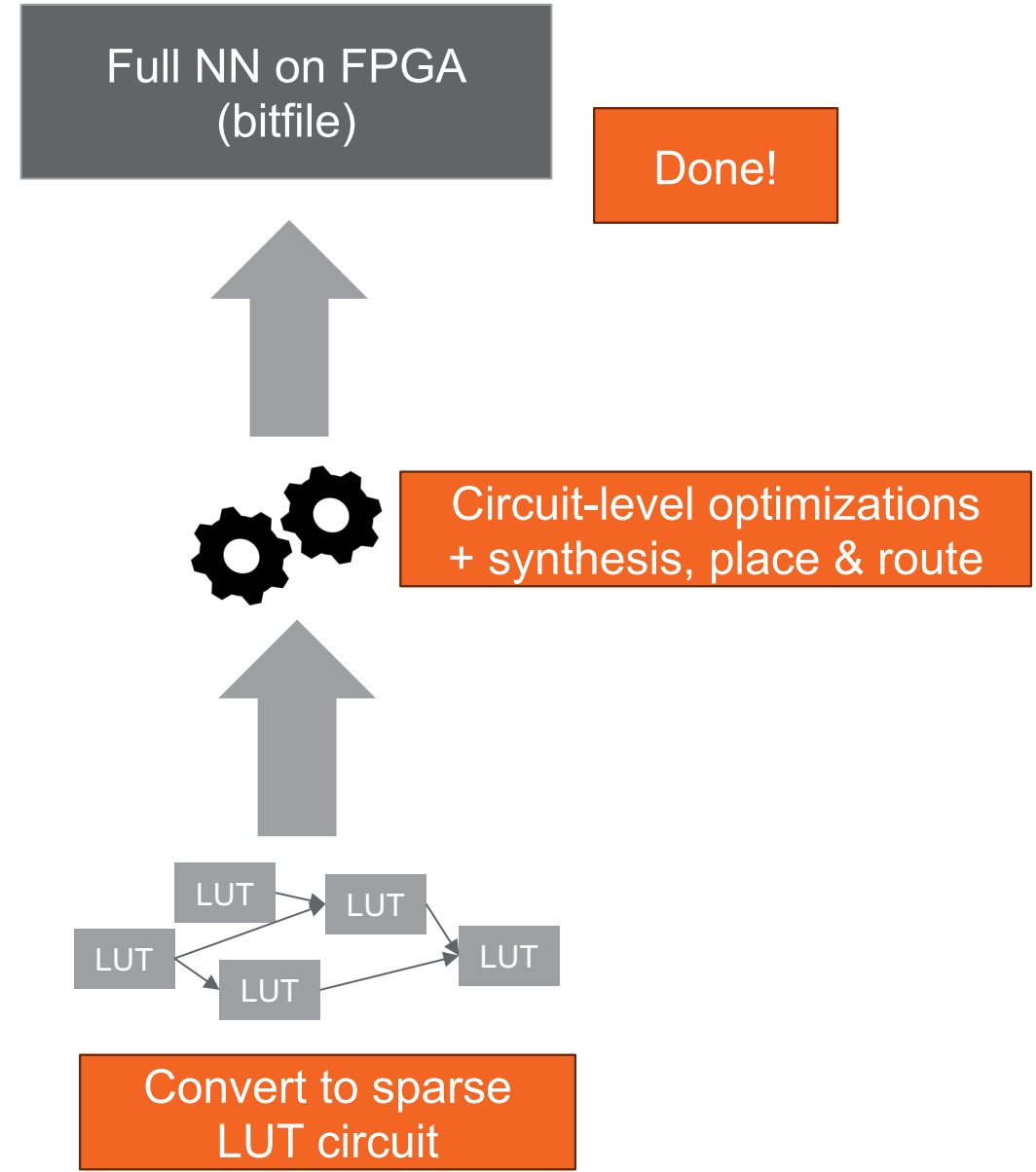


Build sparse+quantized topology out of neurons

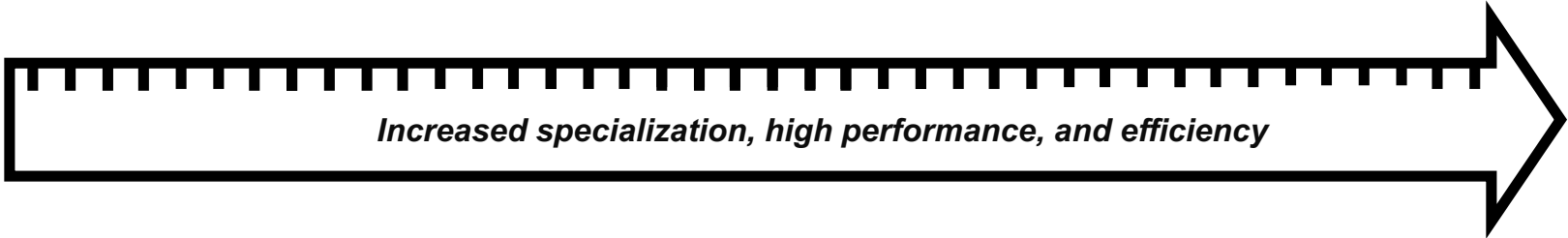


Train in ML framework

PyTorch FPGA



NIDS Results




Matrix of Processing Engines

Dataflow + Quantization + Sparsity

Co-Designed Sparse LUT Circuit

Topology / #layers / #OPs
Datatype
Accuracy

Vitis AI
MLP / 3 / 92kOPs
8b & 8b
92.3%


MLP / 3 / 92 kOPs
2b & 2b
91.9%

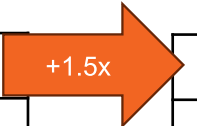


LogicNets
Circuit / 4 / 15.4kOPs
32b & 2b
91.3%

Performance
Throughput
Latency (compute only)

22 kinfps
26 us

Fold 8	Unfolded
25.3 Minfps	300 Minfps
160 ns	18 ns

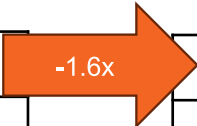


471 Minfps
9 ns

Resources
Compute (kLUTs, DSPs*)
Memory (BRAM, URAM**)
Clock

122,1124
290, 92
300/600 MHz

44, 0	10, 0
166, 0	0, 0
203 MHz	300 MHz



16, 0
0, 0
471 MHz

Mapped on UltraScale+, 16nm FPGA, all within the same SLR.

*DSPs: 8b or 16b Multiply Accumulates

**BRAMs: 36kb, URAM: 288kbit embedded SRAM blocks

LogicNets for Science: Jet Substructure Classification [2]

Related Work

Config.	Acc. [%]	LUT	F _{max} [MHz]	Latency [ns]
High accuracy $\geq 73\%$				
Duarte et al. [2]	75	88k +1k DSPs	200	50
FINN W8A8	75.5	581k		115
FINN W4A4	73.6	47k		85
NullaNet-L [3]	73.4	11.8k	436	-
Medium accuracy $\geq 71\%$				
FINN W2A2	71.0	3k	200	75
NullaNet-M [3]	72.2	1.6k	841	-
Low accuracy $< 71\%$				
NullaNet-S [3]	69.7	39	2,079	-

LogicNets

LogicNets for Science: Jet Substructure Classification [2]

Related Work

Config.	Acc. [%]	LUT	F _{max} [MHz]	Latency [ns]
High accuracy $\geq 73\%$				
Duarte et al. [2]	75	88k +1k DSPs	200	50
FINN W8A8	75.5	581k		115
FINN W4A4	73.6	47k		85
NullaNet-L [3]	73.4	11.8k	436	-
Medium accuracy $\geq 71\%$				
FINN W2A2	71.0	3k	200	75
NullaNet-M [3]	72.2	1.6k	841	-
Low accuracy $< 71\%$				
NullaNet-S [3]	69.7	39	2,079	-

7x - 363x smaller
1.7x - 3.7x faster

LogicNets

Config.	Acc. [%]	LUT	F _{max} [MHz]	Latency [ns]
High accuracy $\geq 73\%$				
M2.6	73.0	1.6k	666 (735)	6

LogicNets for Science: Jet Substructure Classification [2]

Related Work

Config.	Acc. [%]	LUT	F _{max} [MHz]	Latency [ns]
High accuracy ≥73%				
Duarte et al. [2]	75	88k +1k DSPs	200	50
FINN W8A8	75.5	581k		115
FINN W4A4	73.6	47k		85
NullaNet-L [3]	73.4	11.8k	436	-
Medium accuracy ≥71%				
FINN W2A2	71.0	3k	200	75
NullaNet-M [3]	72.2	1.6k	841	-
Low accuracy <71%				
NullaNet-S [3]	69.7	39	2,079	-

LogicNets

Config.	Acc. [%]	LUT	F _{max} [MHz]	Latency [ns]
High accuracy ≥73%				
M2.6	73.0	1.6k	666 (735)	6
Medium accuracy ≥71%				
S2.8	72.1	315	666 (882)	3
S2.1	71.3	86	666 (1,350)	3

7x - 363x smaller
1.7x - 3.7x faster

5x - 35x smaller
1x - 6.8x faster

LogicNets for Science: Jet Substructure Classification [2]

Related Work

Config.	Acc. [%]	LUT	F _{max} [MHz]	Latency [ns]
High accuracy ≥73%				
Duarte et al. [2]	75	88k +1k DSPs	200	50
FINN W8A8	75.5	581k		115
FINN W4A4	73.6	47k		85
NullaNet-L [3]	73.4	11.8k	436	-
Medium accuracy ≥71%				
FINN W2A2	71.0	3k	200	75
NullaNet-M [3]	72.2	1.6k	841	-
Low accuracy <71%				
NullaNet-S [3]	69.7	39	2,079	-

LogicNets

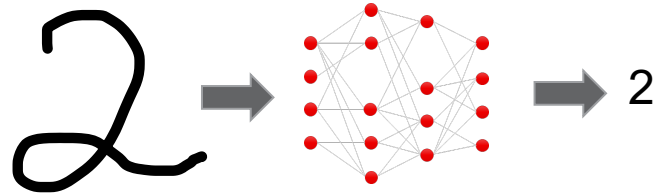
Config.	Acc. [%]	LUT	F _{max} [MHz]	Latency [ns]
High accuracy ≥73%				
M2.6	73.0	1.6k	666 (735)	6
Medium accuracy ≥71%				
S2.8	72.1	315	666 (882)	3
S2.1	71.3	86	666 (1,350)	3
Low accuracy <71%				
S2.0	70.6	30	666 (1,876)	3
S	69.5	24	666 (1,984)	3

7x - 363x smaller
1.7x - 3.7x faster

5x - 35x smaller
1x - 6.8x faster

1.3x smaller

LogicNets for Vision: MNIST



Related Work

Config.	Acc. [%]	LUT	F _{Max} [MHz]	Latency [ns]	FPS
FINN [4] LFC-max	98.4	83k	200	2,440	1.6M
FINN [4] SFC-max	95.8	91k		310	12.4M
LUTNet [5]	97.9	58k		-	200M
Logic-shrunk [5]	97.8	55k		-	200M

LogicNets

Config.	Acc. [%]	LUT	F _{Max} [MHz]	Latency [ns]	FPS
M	97.7	45k	517	38	517M
S	95.8	12k	458	9	458M

Work in progress – already over 2x faster and 20% smaller, at similar accuracy

“FINN [...] the **fastest method** for classifying MNIST at an accuracy of 98.4%,”
Petersen et al., NeurIPS’22 [6]

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, hls4ml and LogicNets
 - **Ecosystem** to build & share the technical expertise
- Fruitful AMD-FastML collaboration strengthens the ecosystem
 - QONNX – **active with Thea Aarestad, Sioni Summers ++**
 - MLPerf Tiny joint submission
 - Multiple joint papers
 - *...more to come!*

Internships available at AMD RADICAL Dublin!
Talk to me or e-mail your CV: yamanu@amd.com

References

1. Moustafa, Nour, and Jill Slay. "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)." 2015 military communications and information systems conference (MilCIS). IEEE, 2015.
2. Duarte et al., "Fast inference of deep neural networks in FPGAs for particle physics," Journal of Instrumentation, vol. 13, no. 07, 2018.
3. Nazemi et al. "NullaNet Tiny: Ultra-low-latency DNN inference through fixed-function combinational logic." FCCM, 2021.
4. Wang et al. "Logic Shrinkage: Learned Connectivity Sparsification for LUT-Based Neural Networks." ACM TRETS, 2023.
5. Umuroglu, Yaman, et al. "FINN: A framework for fast, scalable binarized neural network inference." FPGA. 2017.
6. Petersen et al. "Deep Differentiable Logic Gate Networks." NeurIPS, 2022.
7. Colbert et al. "A2Q: Accumulator-Aware Quantization with Guaranteed Overflow Avoidance", ICCV, 2023.
8. Colbert et al. "A2Q+: Improving Accumulator-Aware Weight Quantization", ICML, 2024 (to appear).

COPYRIGHT AND DISCLAIMER

©2024 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 