Big neural networks in small spaces

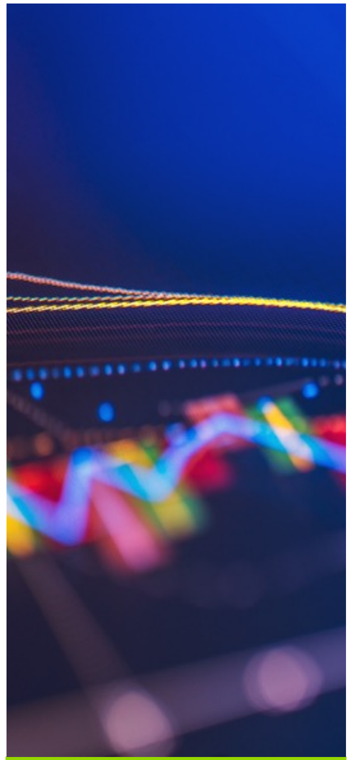
Towards end-to-end optimisation for ML at the edge

Rune Holm, Machine Learning Group, Arm

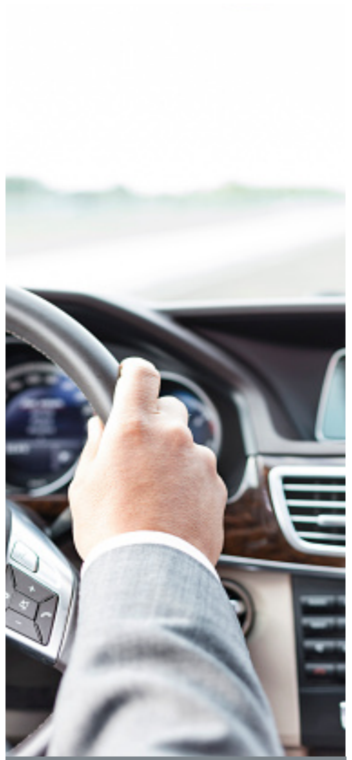
Why is ML Moving to the Edge?

Bandwidth

Power

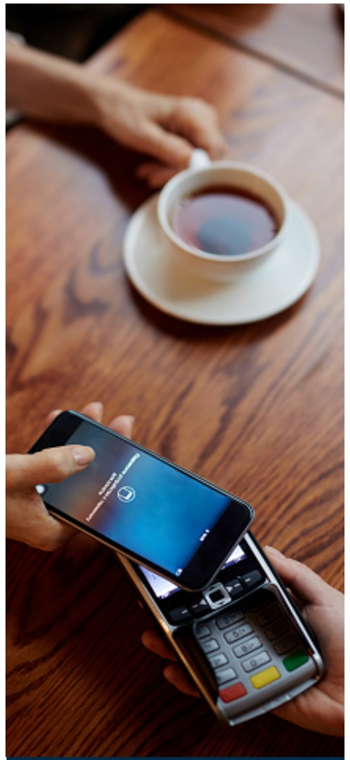


Cost



Latency

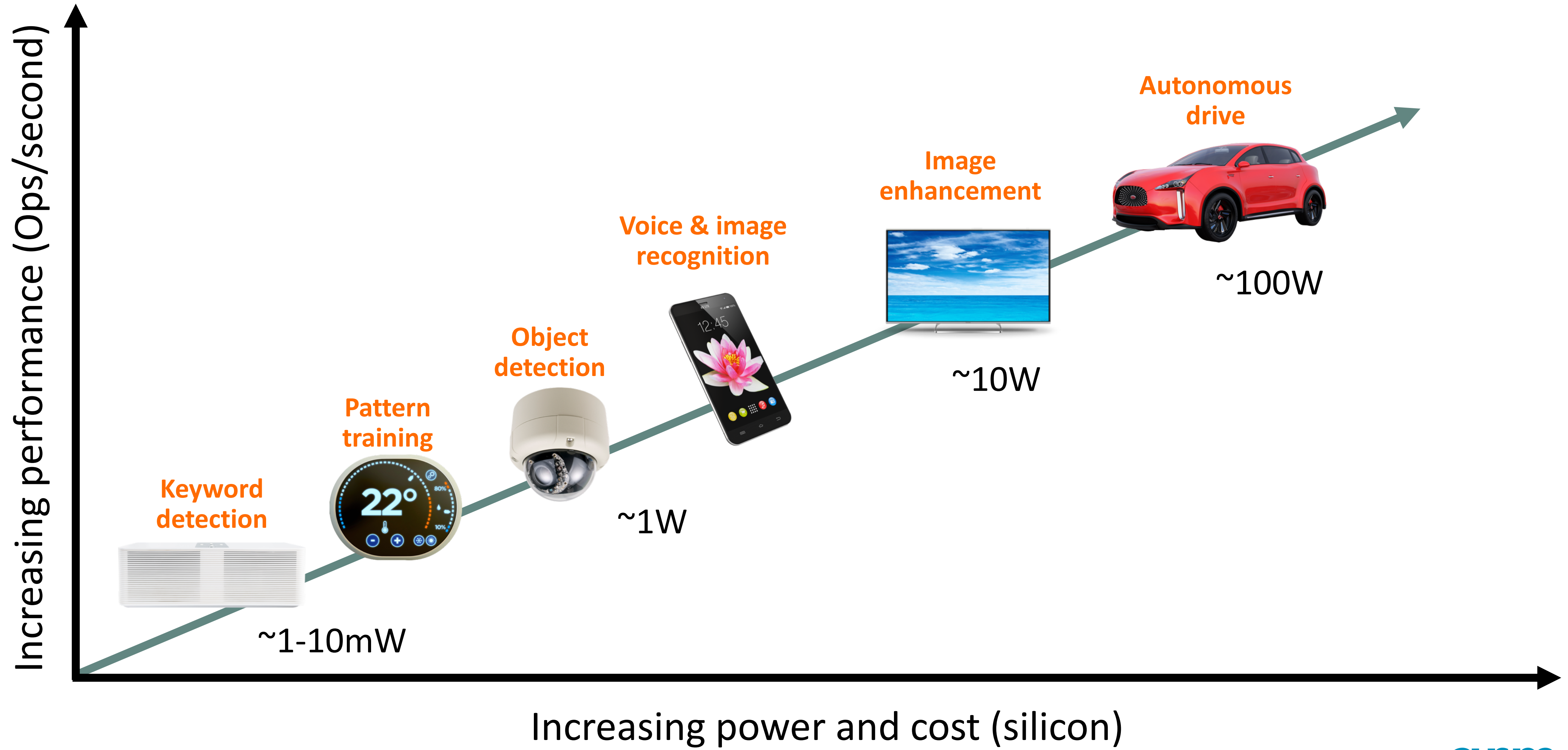
Reliability



Security

arm

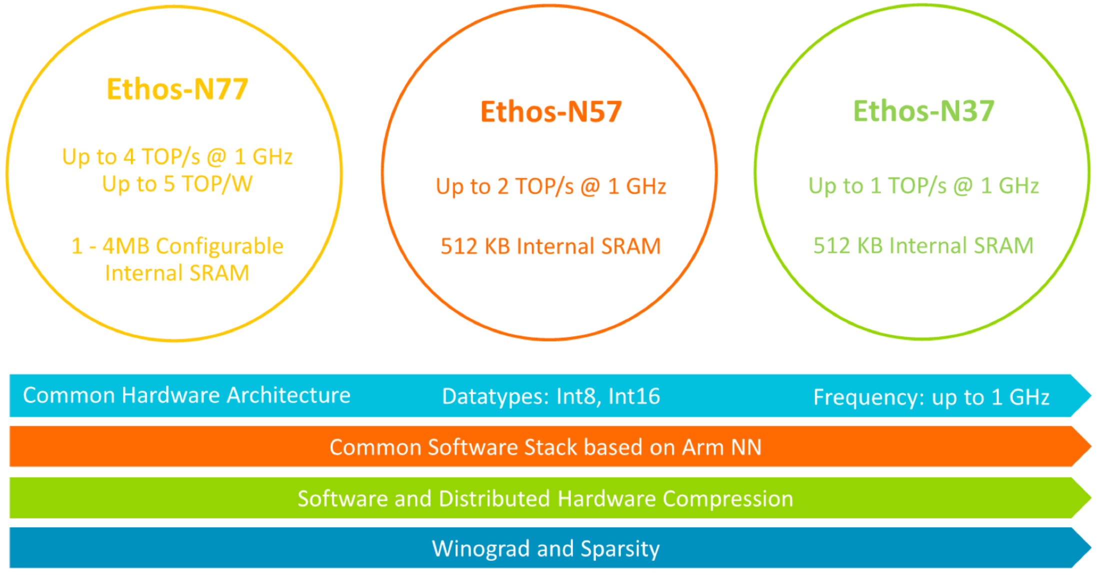
Wide Range of "Edge" Inference Applications



On-Device ML - Challenges

Tiny-edge device constraints for deploying ML algorithms

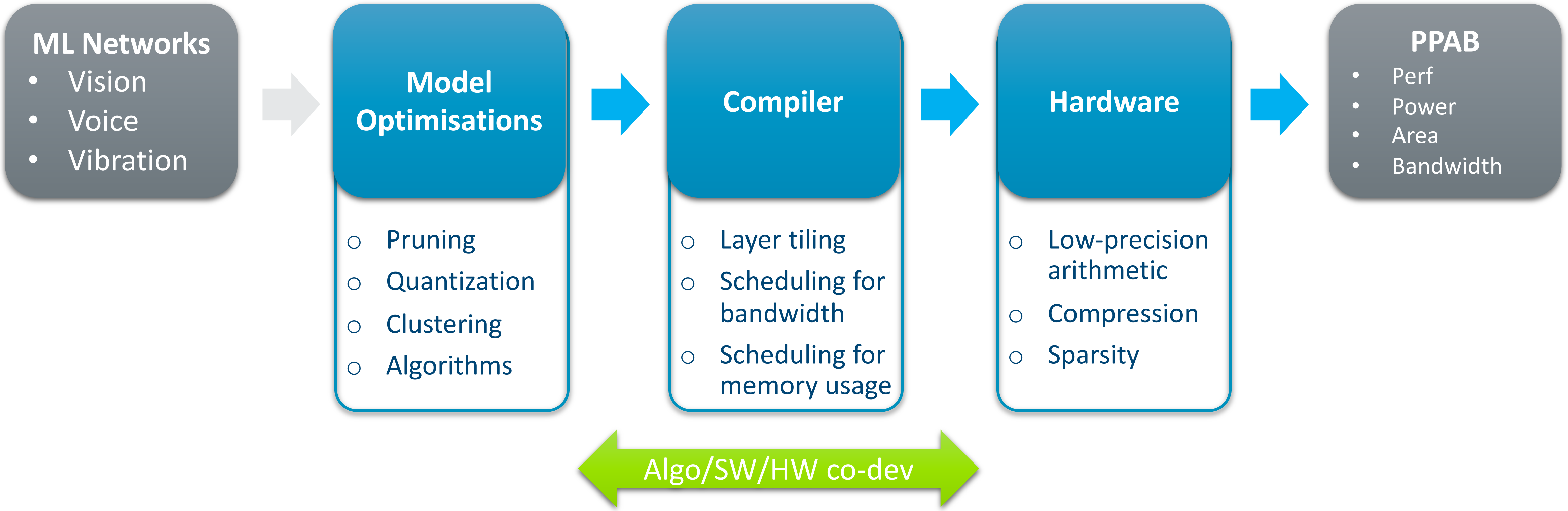
- Limited memory
  - SRAM (16 kB - 1024 kB)
- Limited compute capability (100 MHz - 1 GHz)
- Limited bandwidth
  - DRAM (2-16 GB/s)



- The better we optimise, the more interesting use cases we can run

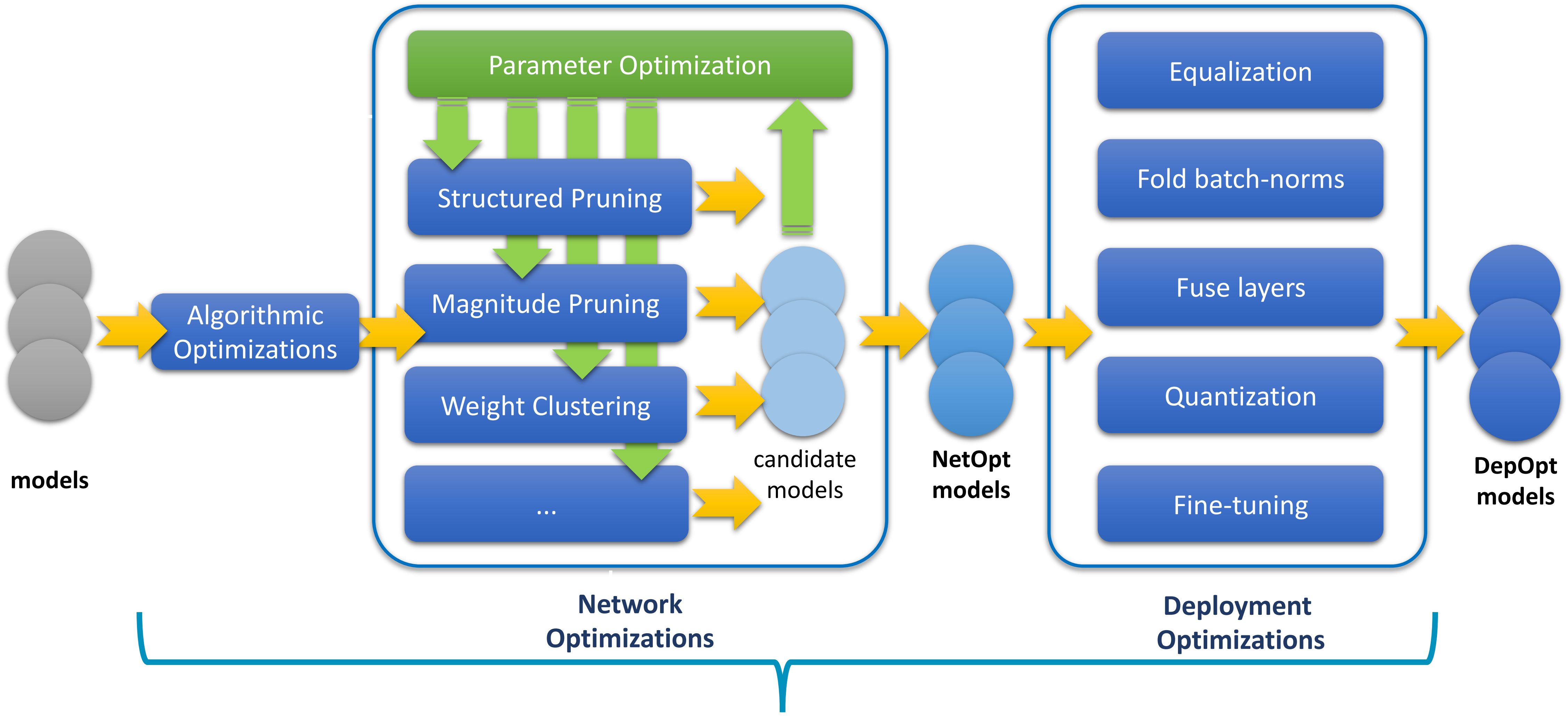
On-Device ML solutions = Model Optimization → Compiler → Hardware

End-to-end optimisation



Model Optimisations

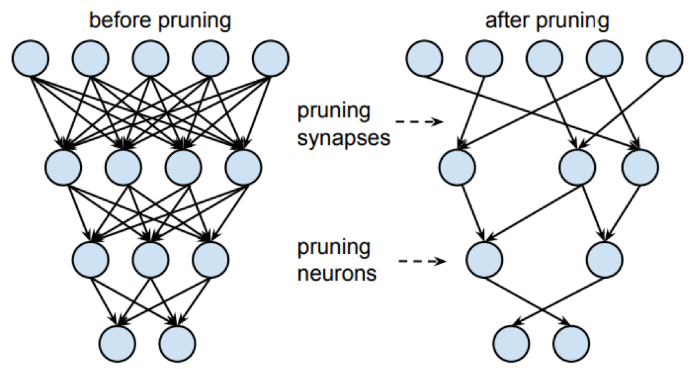
Overview of Model Optimizations

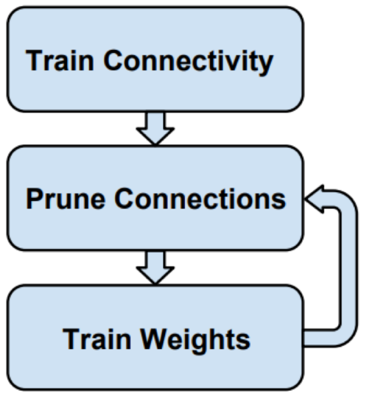


Collaborative Optimizations

Overview of Pruning Techniques

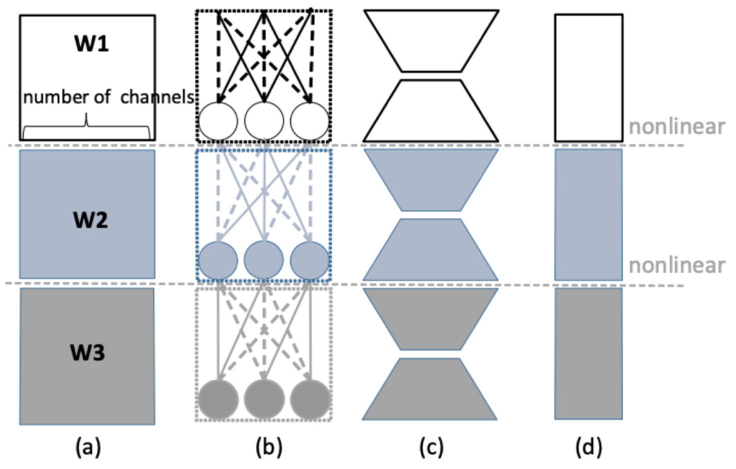
Magnitude Pruning

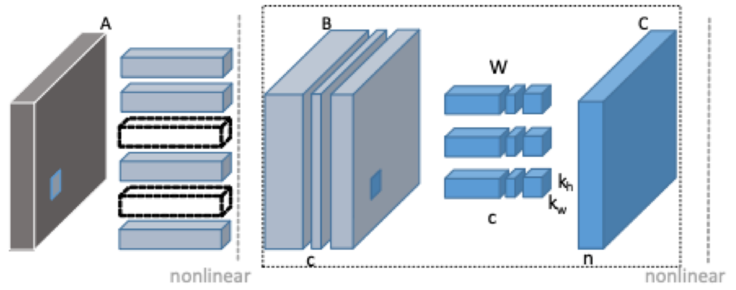




Song Han et al. “*Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*” arXiv: 1510.00149 (2015).

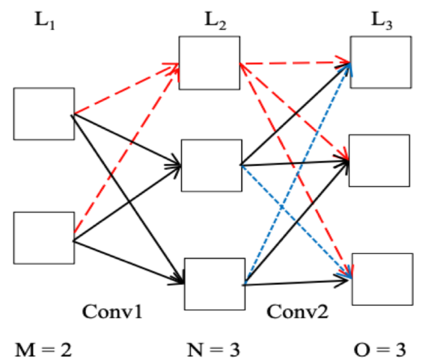
Channel Pruning





Yihui He et al. “*Channel Pruning for Accelerating Very Deep Neural Networks*” arXiv: 1707.06168 (2017).

Structured Pruning



$$\begin{array}{ccccc} 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{array} \qquad \begin{array}{ccccc} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{array}$$

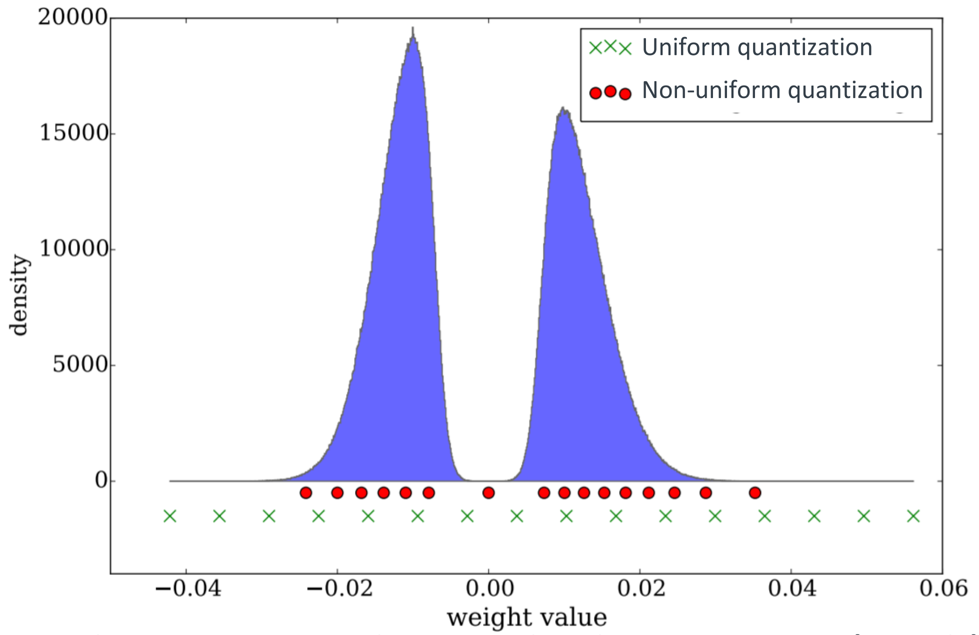
Sparsity = 50% ↓ Sparsity = 75% ↓

$$\begin{array}{ccccc} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{array} \qquad \begin{array}{ccccc} 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{array}$$

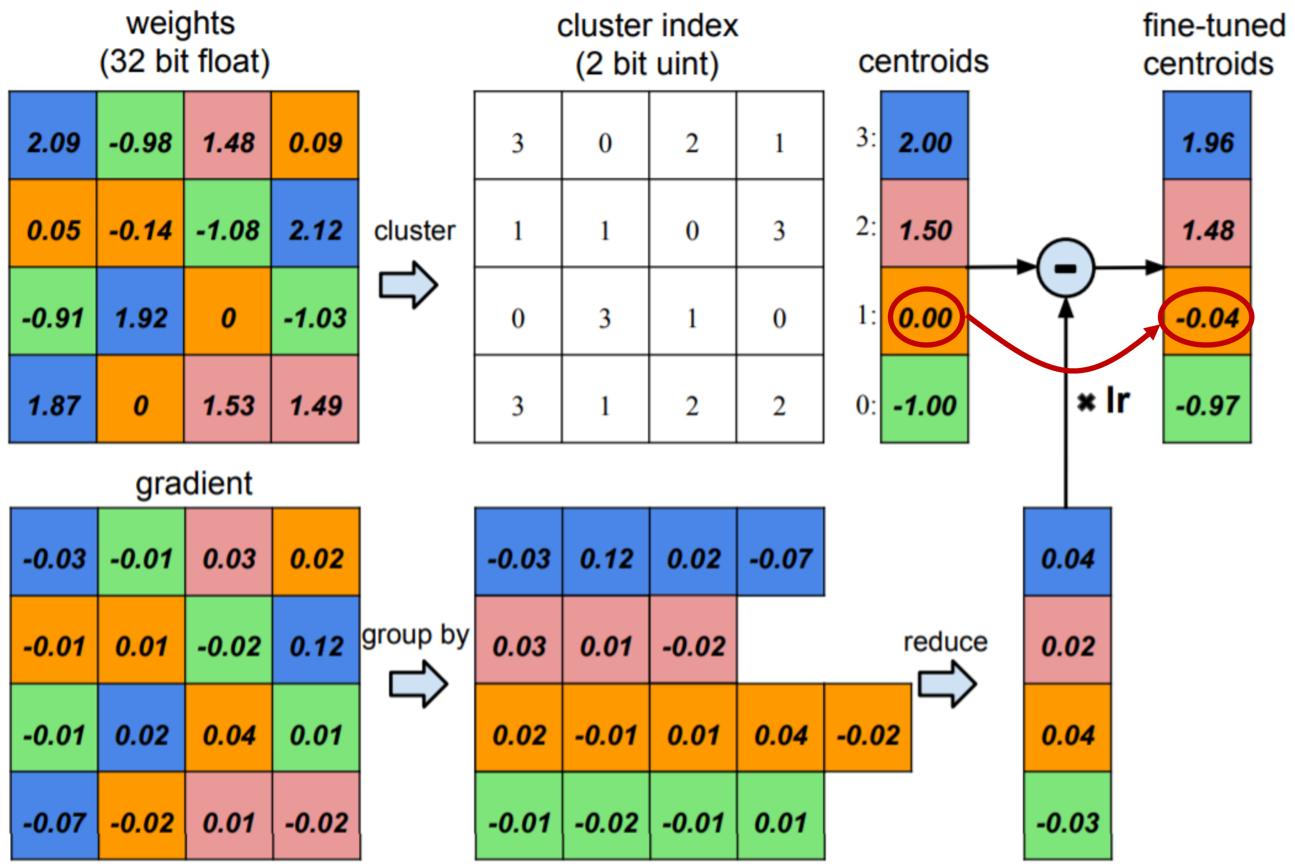
Stride = 2 *Stride = 4*

Sajid Anwar et al. “*Structured Pruning of Deep Convolutional Neural Networks*” arXiv: 1512.08571 (2015).

Clustering: Non-uniform Quantization



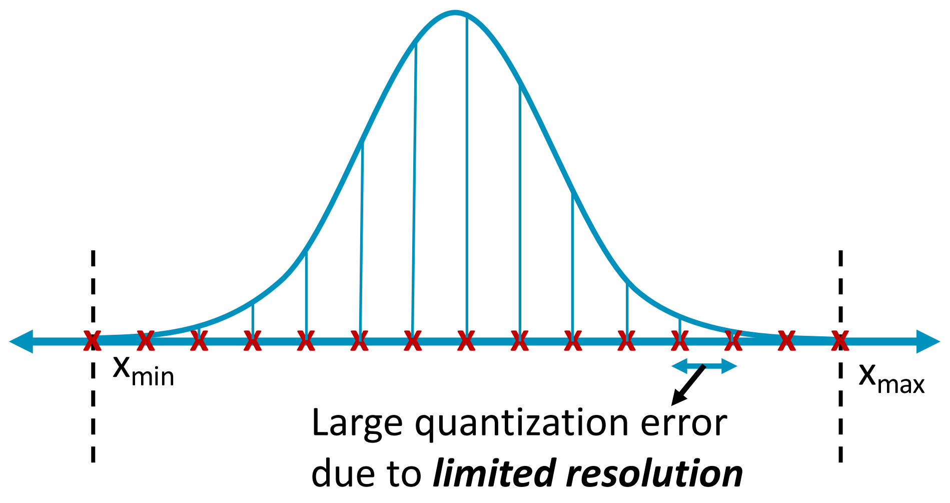
- Cluster n-weights to the k-centroids (n>>k).
  - Use K-Means for initial clustering
  - Enables weight compression
- Update centroids during retraining.
  - Sparsity preservation

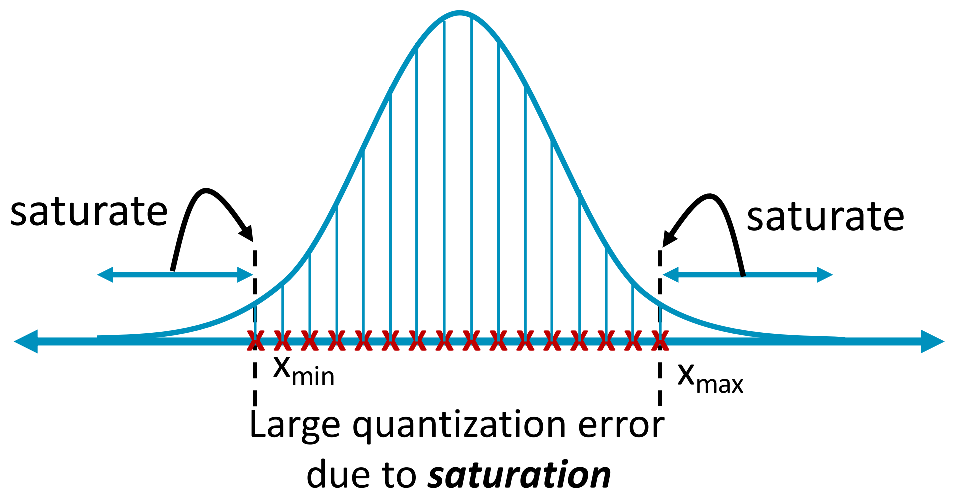


Song Han et al. "*Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*" arXiv: 1510.00149 (2015).

Uniform Quantization: Balancing Range vs. Resolution

Finding Optimal (min, max) for Quantization





Goal: Find (x_{min_opt}, x_{max_opt}) that minimizes quantization error

Solution: Signal-to-Quantization Noise Ratio (SQNR) as a metric to choose optimal quantization ranges.

Optimized Models

Networks	Optimization	Accuracy Loss/Increase
Inception V3	pruned (50%), clustered (5-bit), quantized (8-bit)	1% loss
Resnet 50	pruned (50%), clustered (5-bit), quantized (8-bit)	1.1% loss
VGG16	pruned (50%), quantized (8-bit), clustered (3 clusters for last 3 layers)	0.3% increase

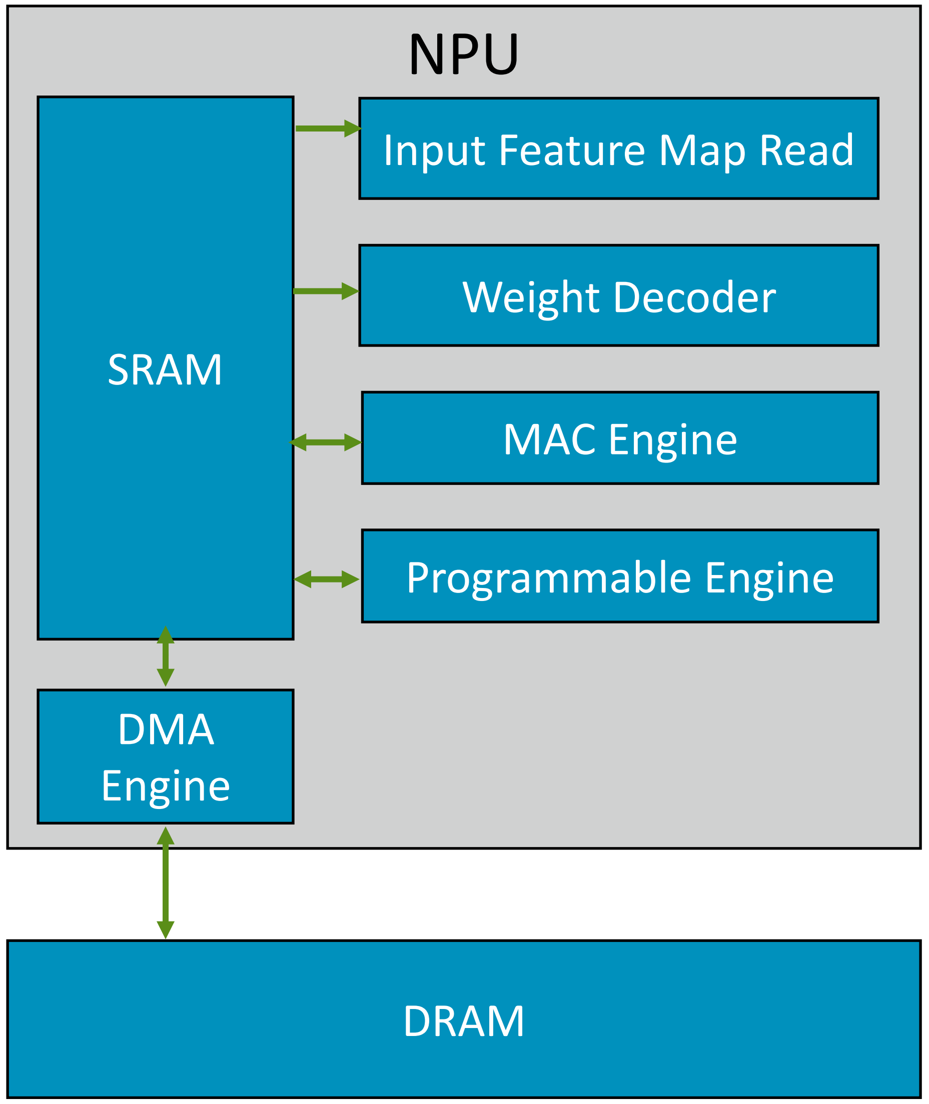
* Post-training quantization applied. Accuracy further improves with fine-tuning.

- Application domains
  - image classification, object detection, speech recognition, etc.
- Reduce model size and improve compressibility
- Enable efficient on-device computation

Neural Processor Unit hardware

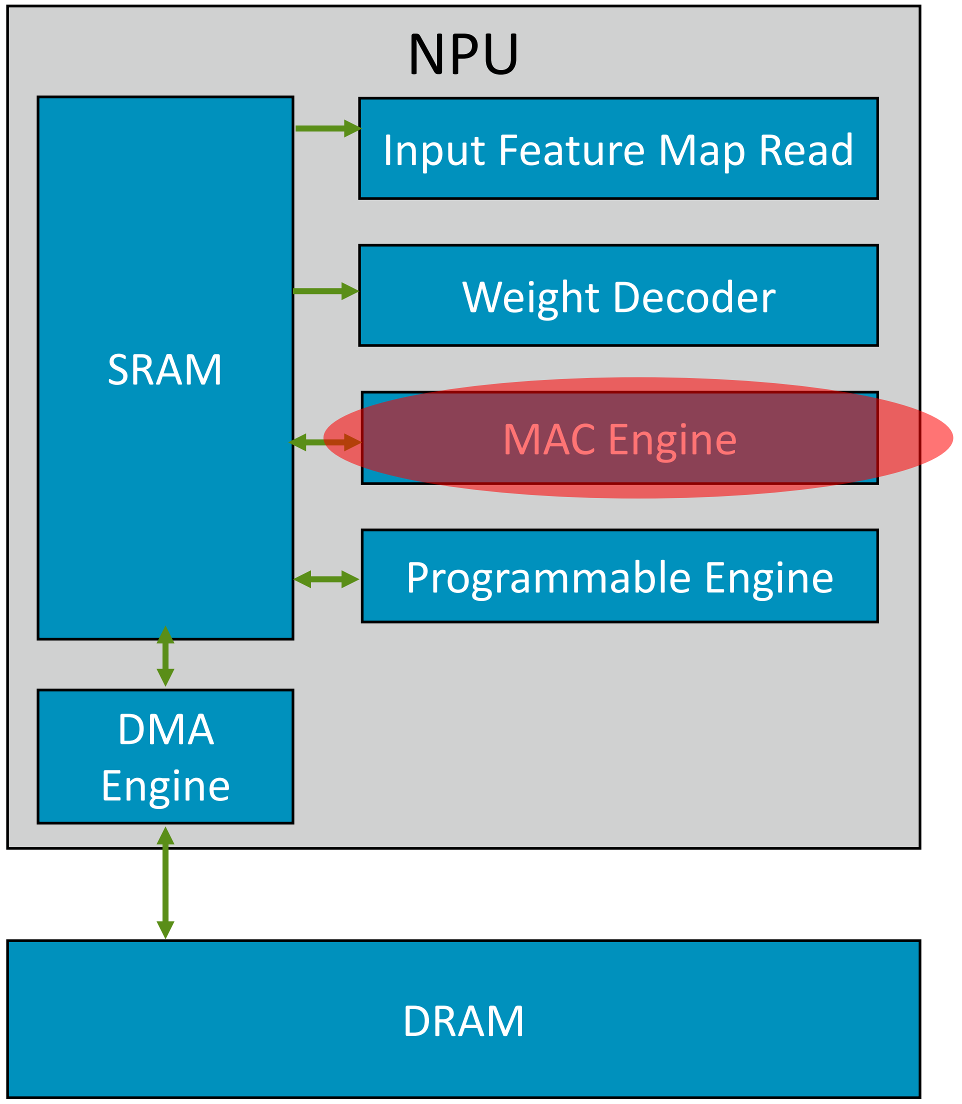
Key Ingredients for a Neural Processor Unit

- Efficient convolutions
- Bandwidth reduction mechanisms
- Static scheduling



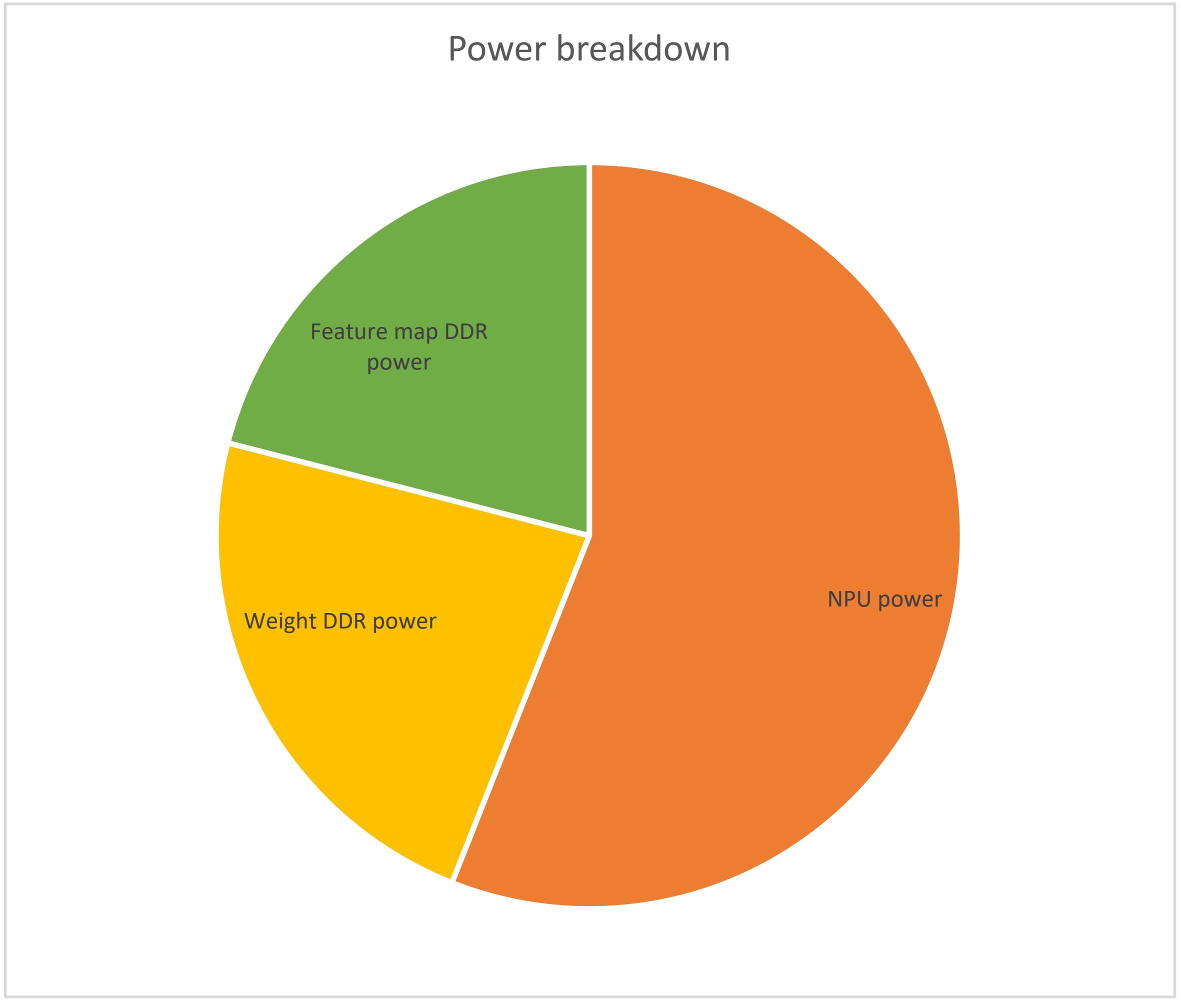
Efficient convolutions

- Large amount of MAC units – utilize the 100+:1 ALU:LS ratio of typical convolutions
- Quantisation
  - 8 bit integer operations for CNNs
  - More bits for RNNs
  - Fewer bits are possible for some layers
- Reuse of SRAM reads between MAC units, otherwise SRAM read power dominates
- Significant number of zeros (ReLU: >50% feature map zeros)
  - Opportunities for clock gating
  - Or even zero-skipping units

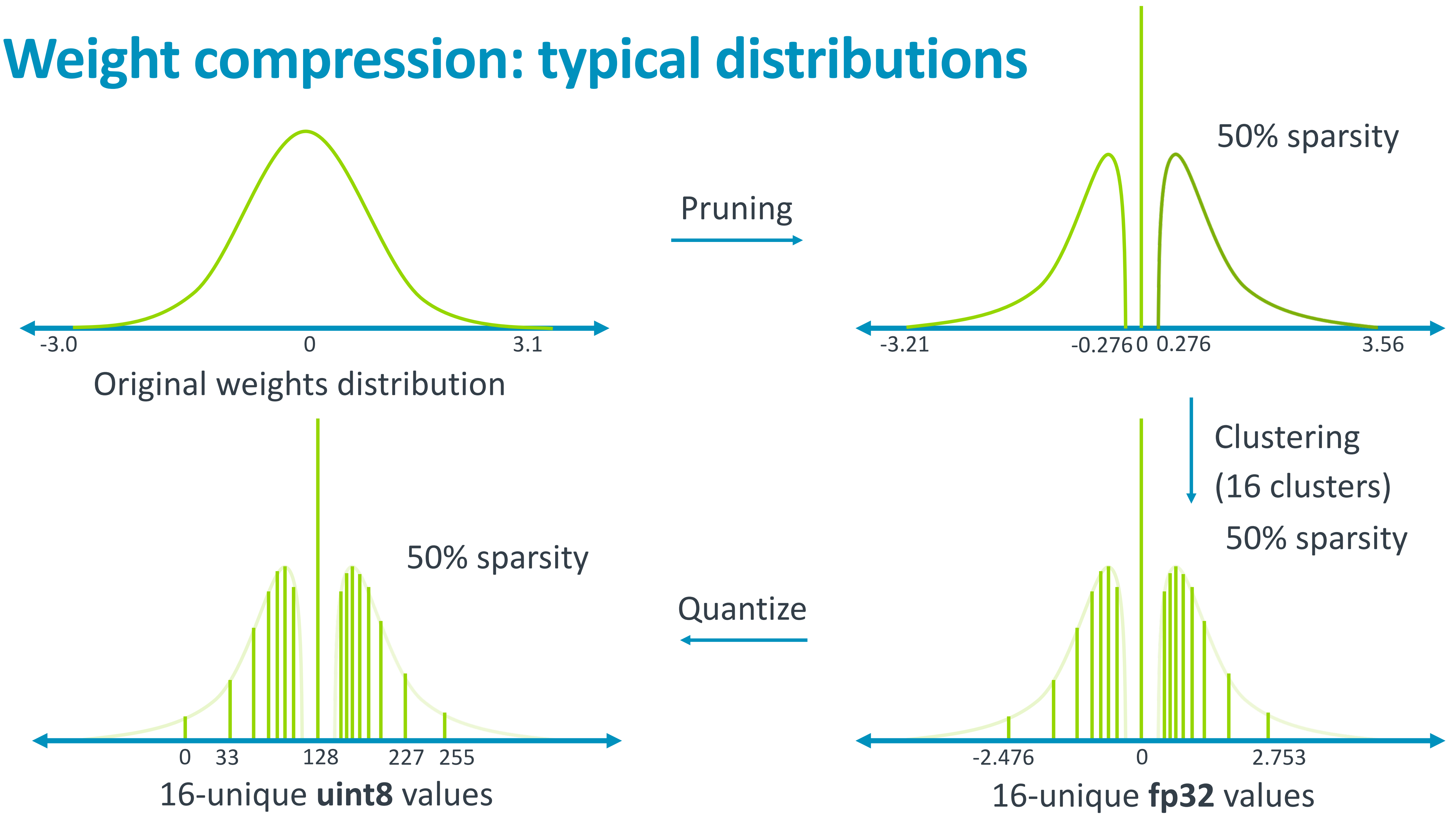


Importance of Weight and Feature Map Compression

- DRAM power can be nearly as high as the processor power itself
- Bandwidth reduction techniques important
  - Weight compression
  - Activation compression
  - Tiling



Weight compression: typical distributions



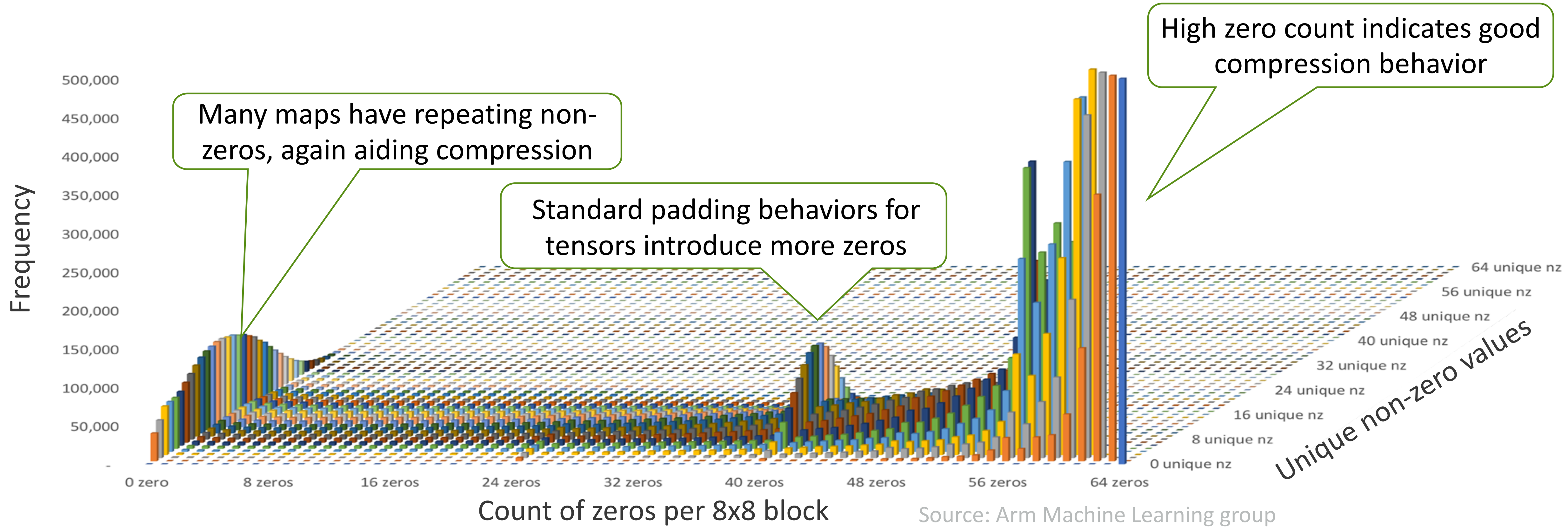
Lossless weight compression

- Unequal distribution provides compression opportunities, straight out of TF/PyTorch
- Pruning and clustering provide additional possibilities

- Multiple off-ramps for different levels of developer effort

Networks	FP32		Quantized		Quantized + compressed		Pruned, clustered, quantized, compressed		Savings
	Size	Bits/elem	Size	Bits/elem	Size	Bits/elem	Size	Bits/elem	
Inception V3	92 MB	32	23 MB	8	16 MB	5.6	12 MB	4.2	**7.7x**
Resnet 50	100 MB	32	25 MB	8	15 MB	4.8	12 MB	3.8	**8.3x**
VGG16	540 MB	32	135 MB	8	96 MB	5.7	32 MB	1.9	**16.9x**

Lossless feature map compression

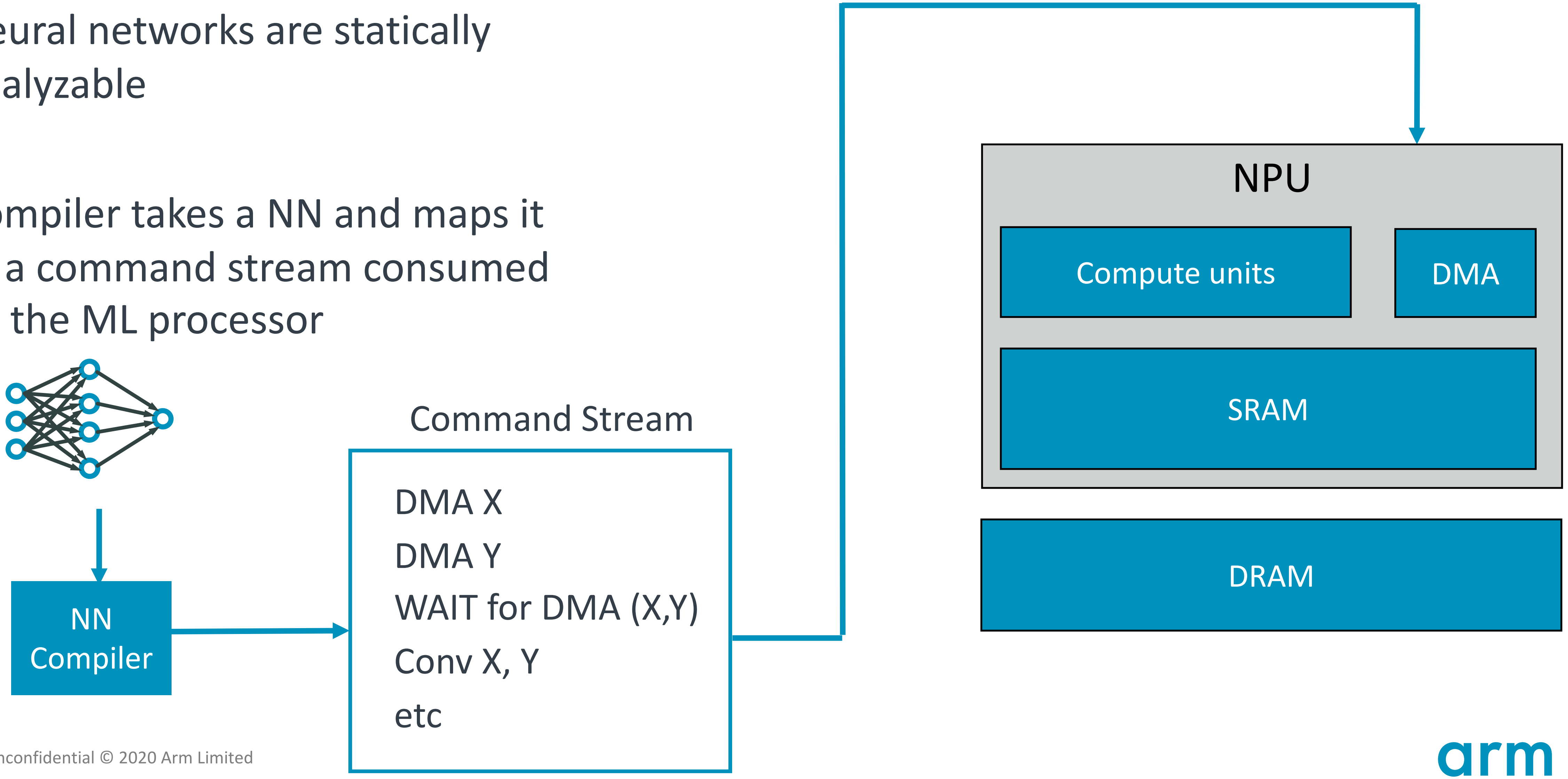


- Compression per 8x8 block
- 3.3x compression for Inception V3

Static Scheduling

- Neural networks are statically analyzable

- Compiler takes a NN and maps it to a command stream consumed by the ML processor



DMA X
DMA Y
WAIT for DMA (X,Y)
Conv X, Y
etc

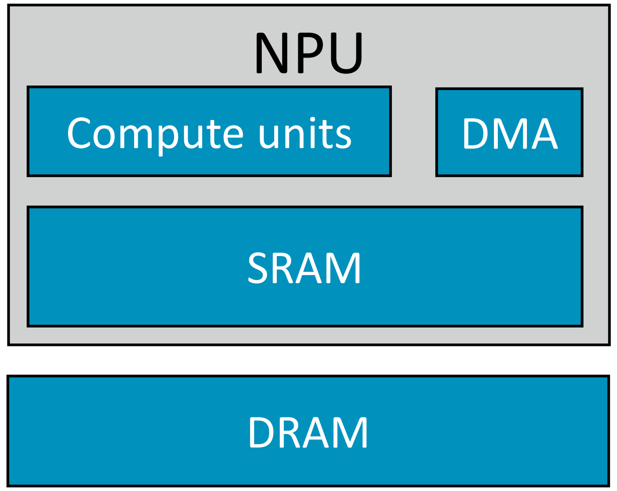
NPU Compiler

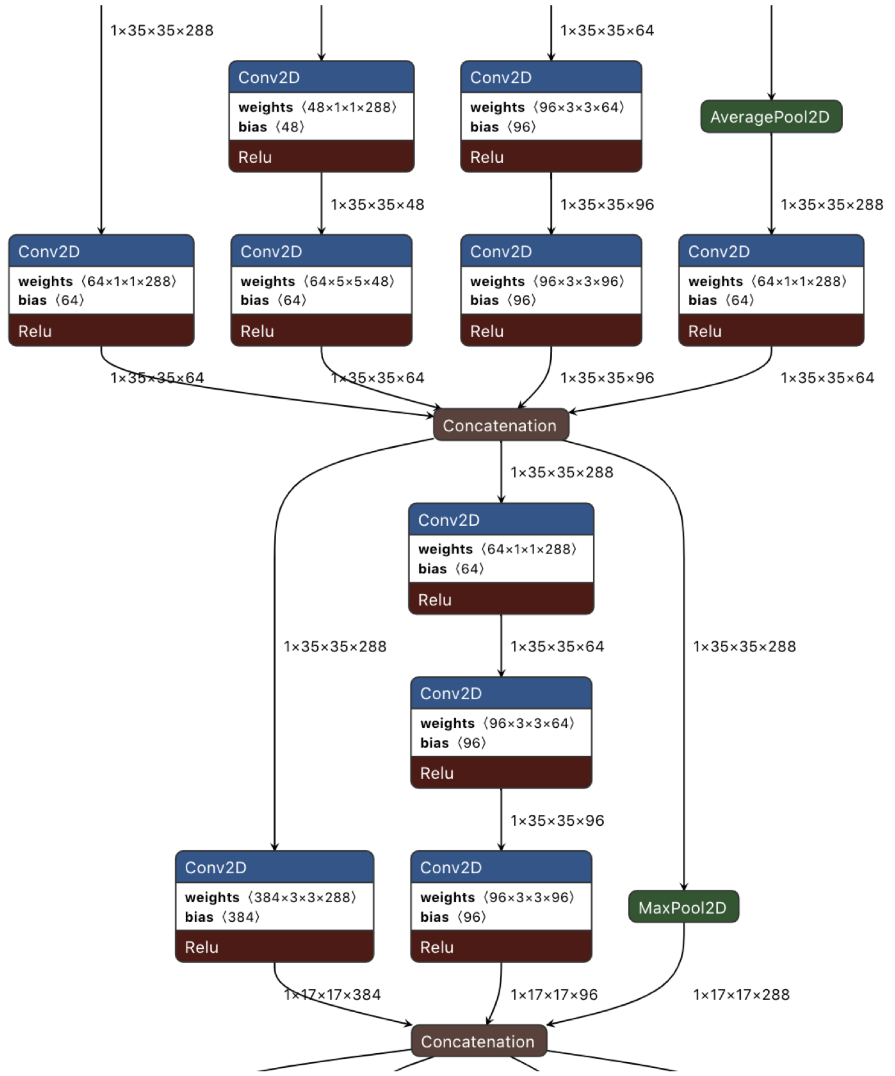
Mapping neural networks onto NPU hardware

NPU: compute units paired with compiler-managed SRAM storage, with DMA units to move data in and out of limited-bandwidth DRAM

Neural network: operations and tensors, in a graph that can have complex connectivity

How do we decide what operations to schedule when, and which tensors or parts of tensors to keep in SRAM?





Styles of compilation

C compilers

```
int max
 (int a, int b)
{
  return b>a?b:a;
}
```

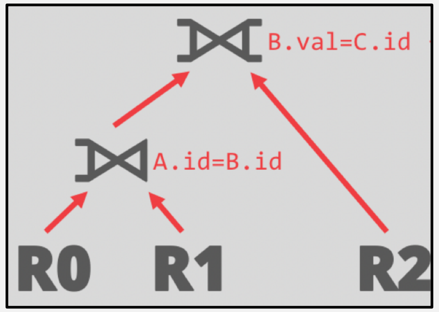
```
max(int, int):
cmp w1, w0
csel w0, w1, w0, ge
ret
```

- Optimising low level flow(instruction scheduling)
- Fixed high level flow (memory layout, access order)

Database query planners

```
SELECT * FROM A
  INNER JOIN B
    ON A.id = B.id
  INNER JOIN C
    ON B.val = C.id
```

- Optimising high level flow (layout, access order)
- Fixed low level flow (pre-implemented routines)



Try to do both at the same time? Infeasible compilation times

The NN compilation problem looks more like query planning than C compilation

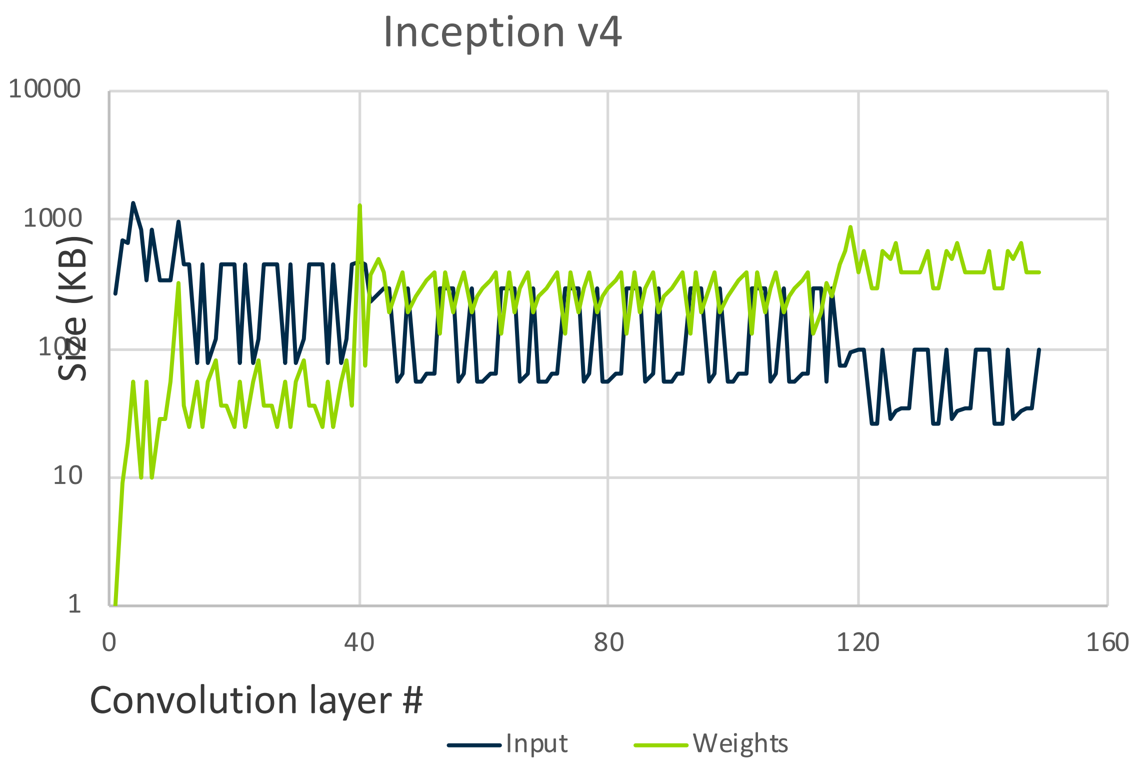
A neural network compiler needs to match that

Scheduling to reduce bandwidth

Choose traversal order to minimize resident memory and bandwidth of a pass.

Inputs large and weights small: Outermost loop index – Output Y

Inputs small and weights large: Outermost loop index - Output Channel



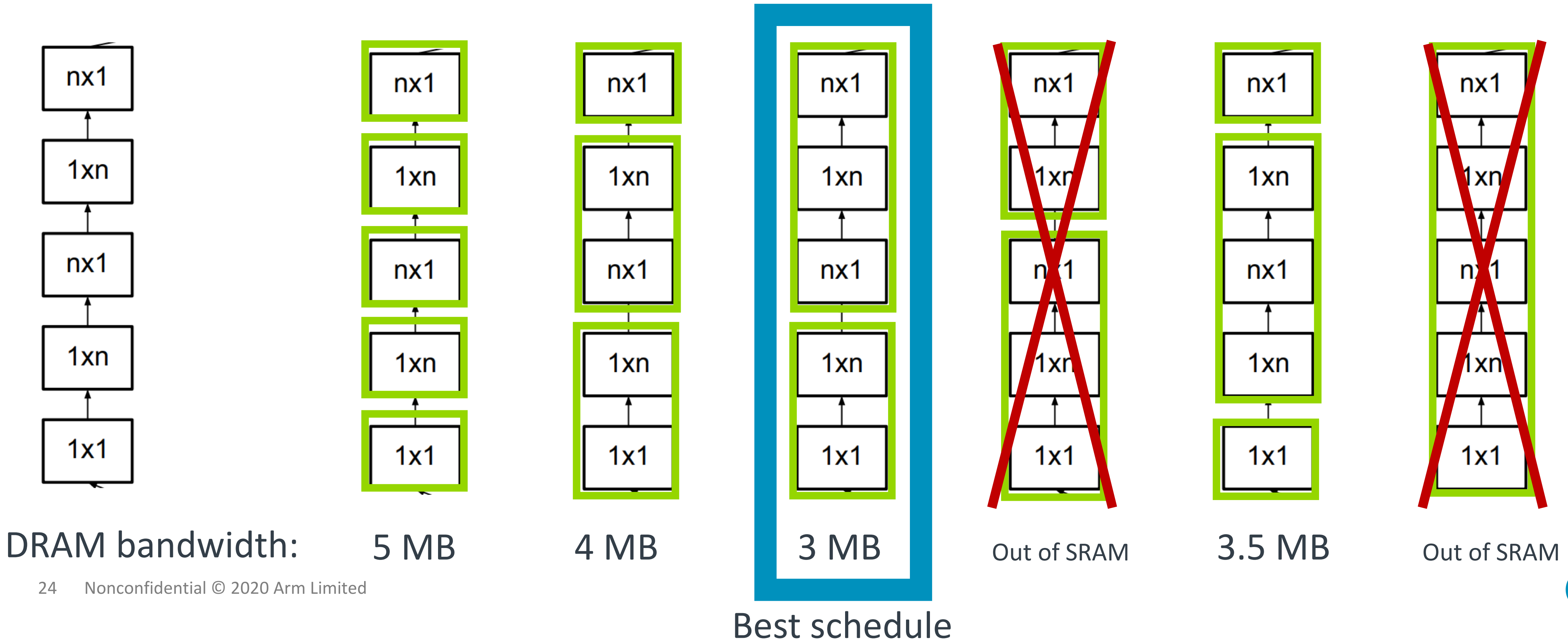
```
conv2d_inputs_large(input, output, weights):
for(output Y)
    for(output channel)
        for(output X)
            for(input channel)
                for(kernel XY)
                    MAC
            write accumulator
```

```
conv2d_weights_large(input, output, weights):
for(output channel)
    for(output Y)
        for(output X)
            for(input channel)
                for(kernel XY)
                    MAC
            write accumulator
```

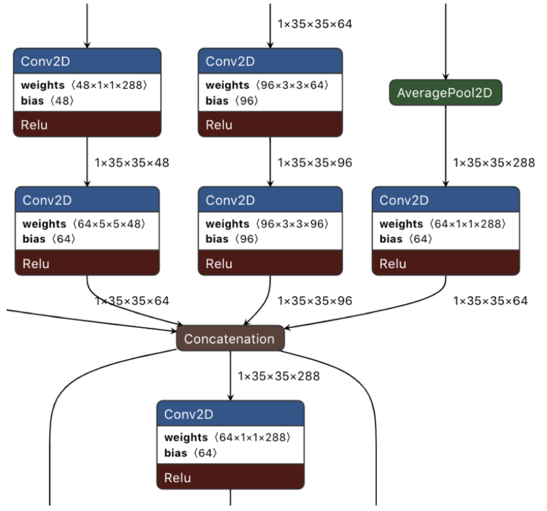
Tiling together passes for better schedule

Tile together passes to avoid writing full intermediate feature maps when possible.

Search for best schedule realizable within the amount of SRAM available.



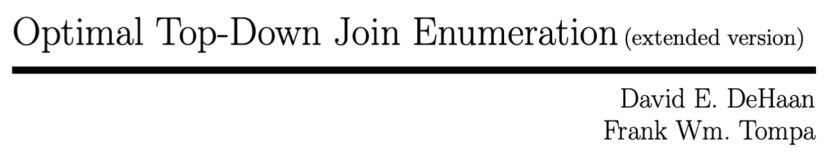
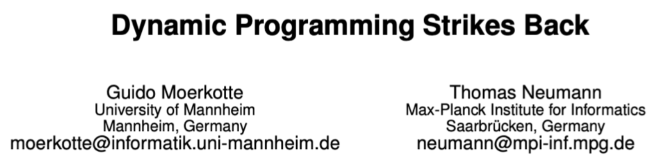
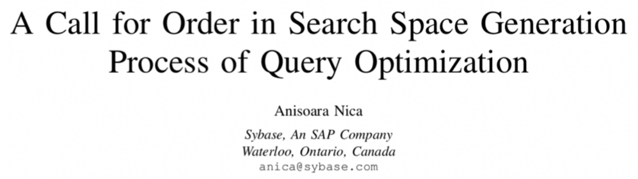
Schedule search



NNs can have complex topology

- a locally optimal choice not necessarily globally optimal

Search can be formulated as a dynamic programming problem, as long as you can use cost functions satisfying the Bellman equation.

Style	Database query planning paper
Top-down search	 Optimal Top-Down Join Enumeration (extended version) David E. DeHaan Frank Wm. Tompa
Bottom-up search	 **Dynamic Programming Strikes Back** Guido Moerkotte University of Mannheim Mannheim, Germany moerkotte@informatik.uni-mannheim.de Thomas Neumann Max-Planck Institute for Informatics Saarbrücken, Germany neumann@mpi-inf.mpg.de
Top-down/bottom-up hybrid	 A Call for Order in Search Space Generation Process of Query Optimization Anisoara Nica *Sybase, An SAP Company* *Waterloo, Ontario, Canada* anica@sybase.com

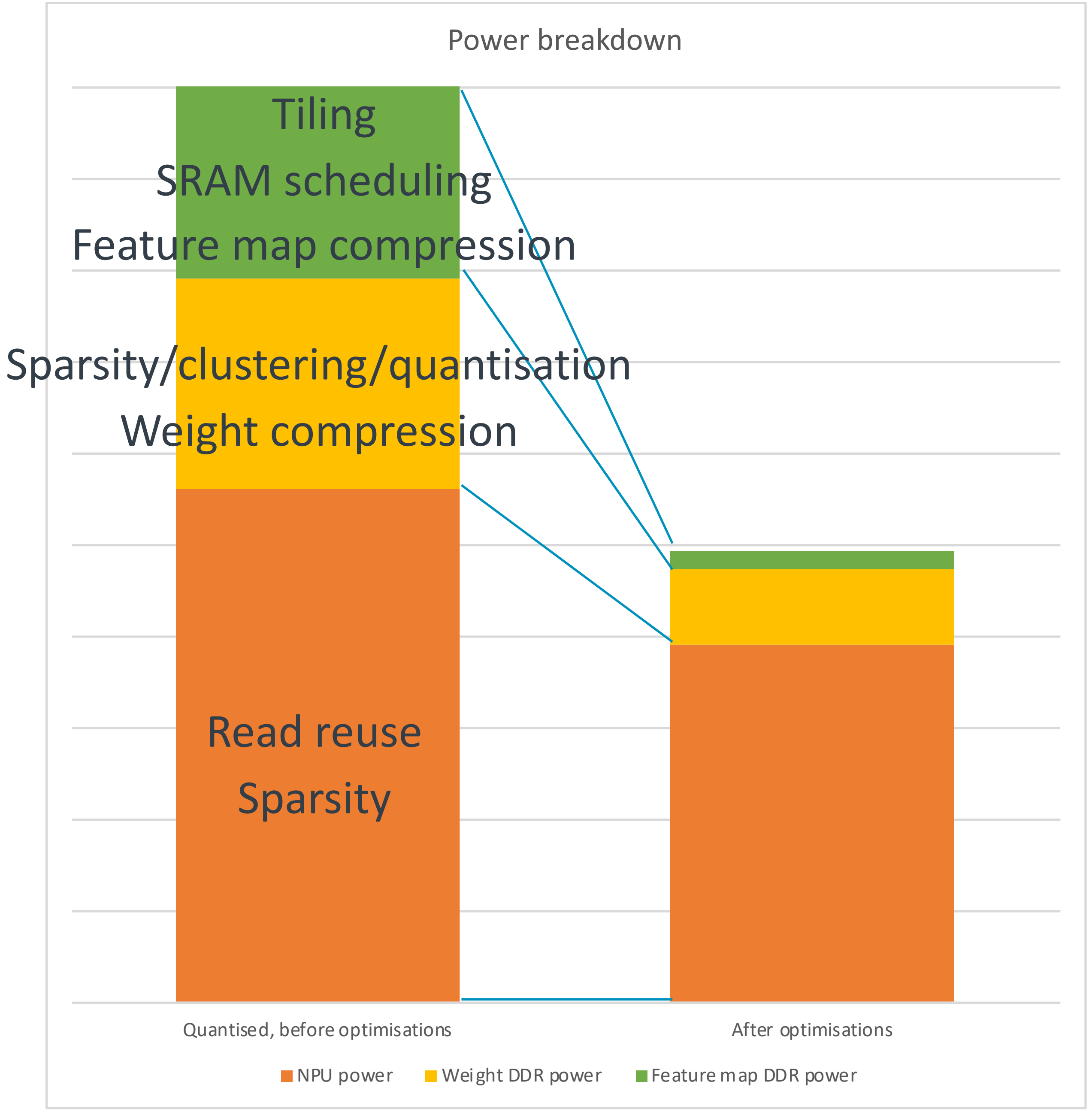
Bringing it all together

Done well, we can eliminate 95%+ of intermediate data traffic to DRAM

(CNNs, 1 MB SRAM, 299x299 input resolution)

Leaving us with:

- NN input read bandwidth
- NN output write bandwidth
- Compressed weight read bandwidth

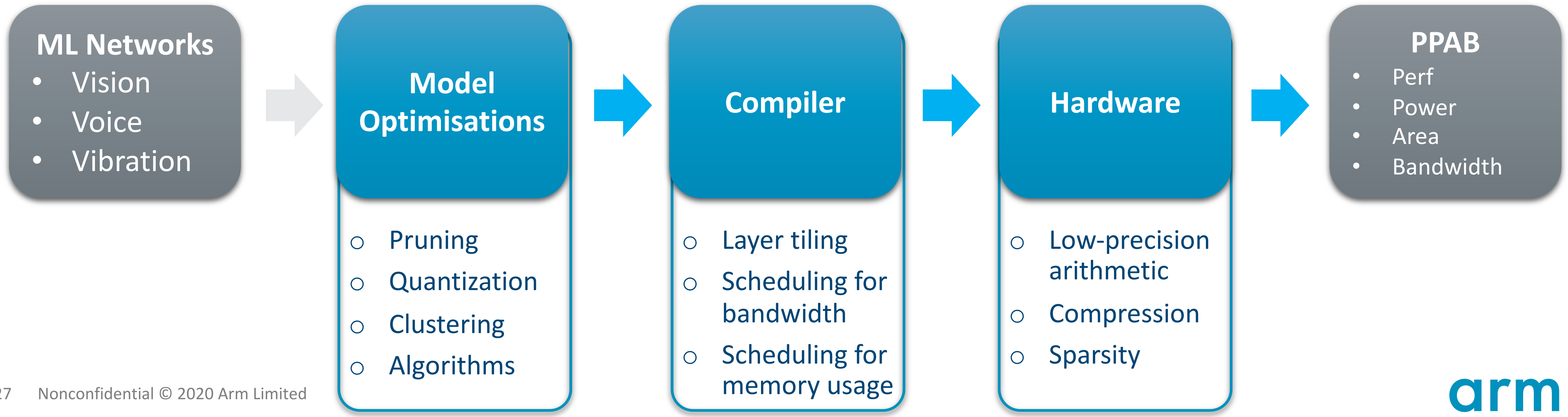


Conclusion

We can enable big neural networks in small spaces

No ”one weird trick” to solve it all at once

Rather, lots of painstaking engineering required: model optimisation, compiler, hardware



ML Networks
Vision
Voice
Vibration
Model Optimisations
Pruning
Quantization
Clustering
Algorithms
Compiler
Layer tiling
Scheduling for bandwidth
Scheduling for memory usage
Hardware
Low-precision arithmetic
Compression
Sparsity
PPAB
Perf
Power
Area
Bandwidth