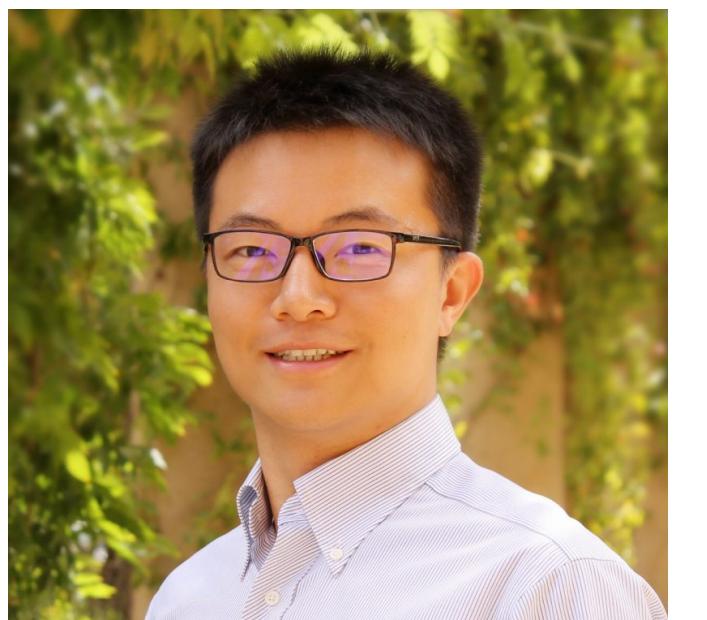


# EfficientML.ai Lecture 08

## Neural Architecture Search

Part II



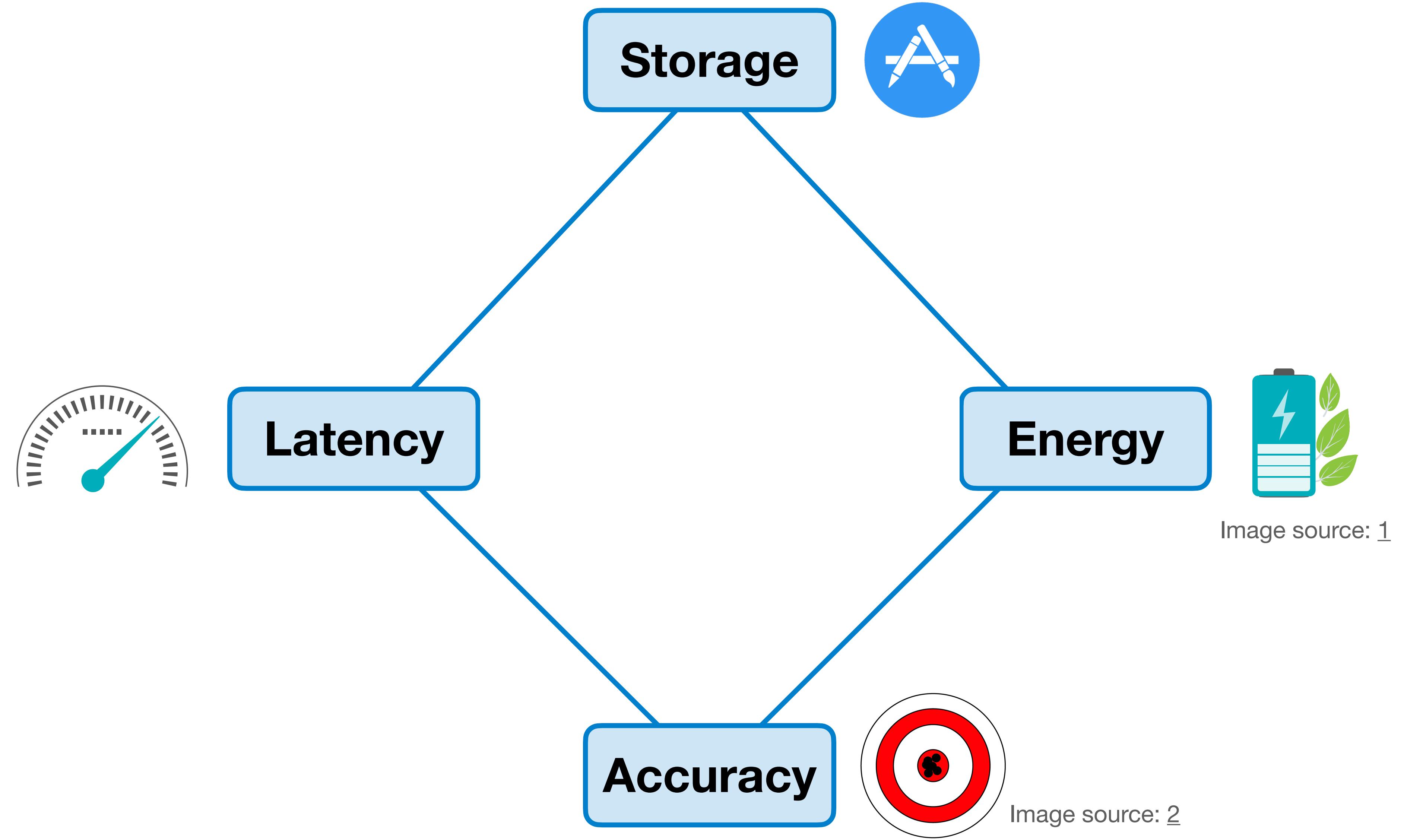
**Song Han**

Associate Professor, MIT  
Distinguished Scientist, NVIDIA

 @SongHan/MIT

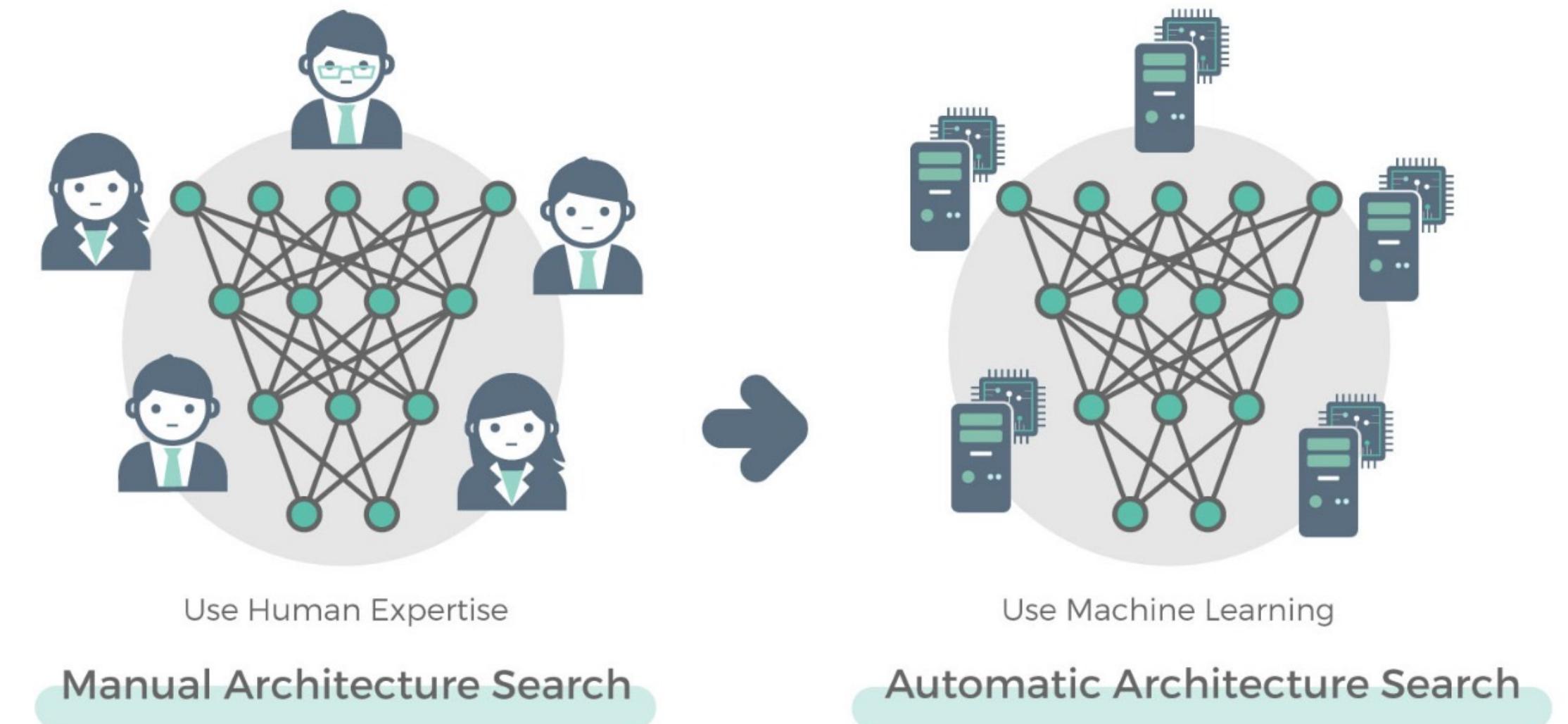


# Trade-off Between Efficiency and Accuracy



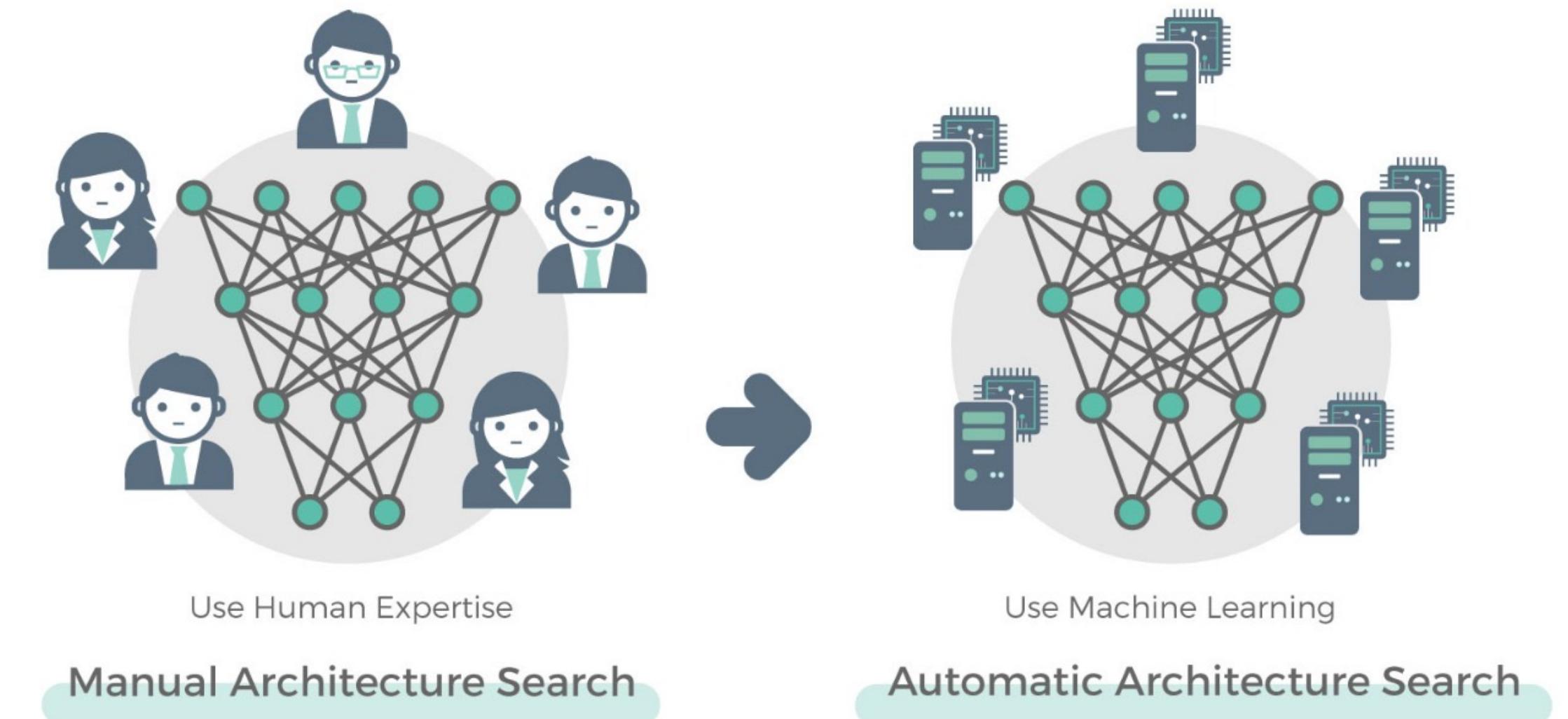
# Neural Architecture Search

- **Last Lecture:**
  - Primitive operations
  - Classic building blocks
  - **Introduction to neural architecture search (NAS)**
    - What is NAS?
    - Search space
    - Design the search space
    - Search strategy
- **This Lecture:**
  - **Efficient and hardware-aware NAS**
    - Accuracy estimation strategy
    - Zero-shot NAS
    - Hardware-aware NAS
    - Neural-hardware architecture co-search
  - **NAS applications**
    - NLP, GAN, point cloud, pose, etc



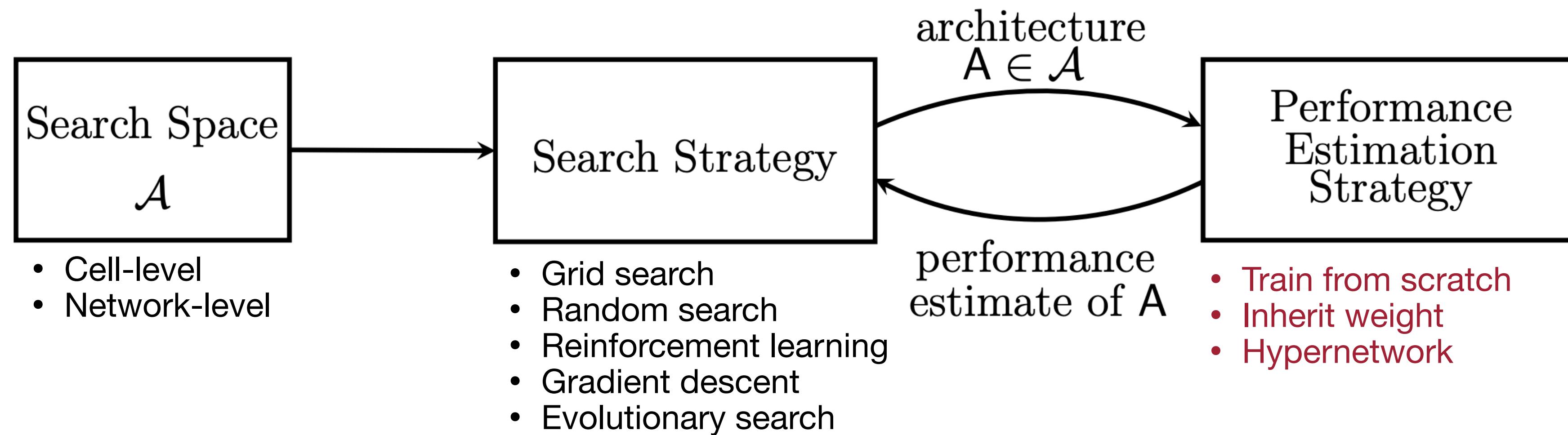
# Neural Architecture Search

- **Last Lecture:**
  - Primitive operations
  - Classic building blocks
  - **Introduction to neural architecture search (NAS)**
    - What is NAS?
    - Search space
    - Design the search space
    - Search strategy
- **This Lecture:**
  - **Efficient and hardware-aware NAS**
    - Accuracy estimation strategy
    - Zero-shot NAS
    - Hardware-aware NAS
    - Neural-hardware architecture co-search
  - **NAS applications**
    - NLP, GAN, point cloud, pose, etc



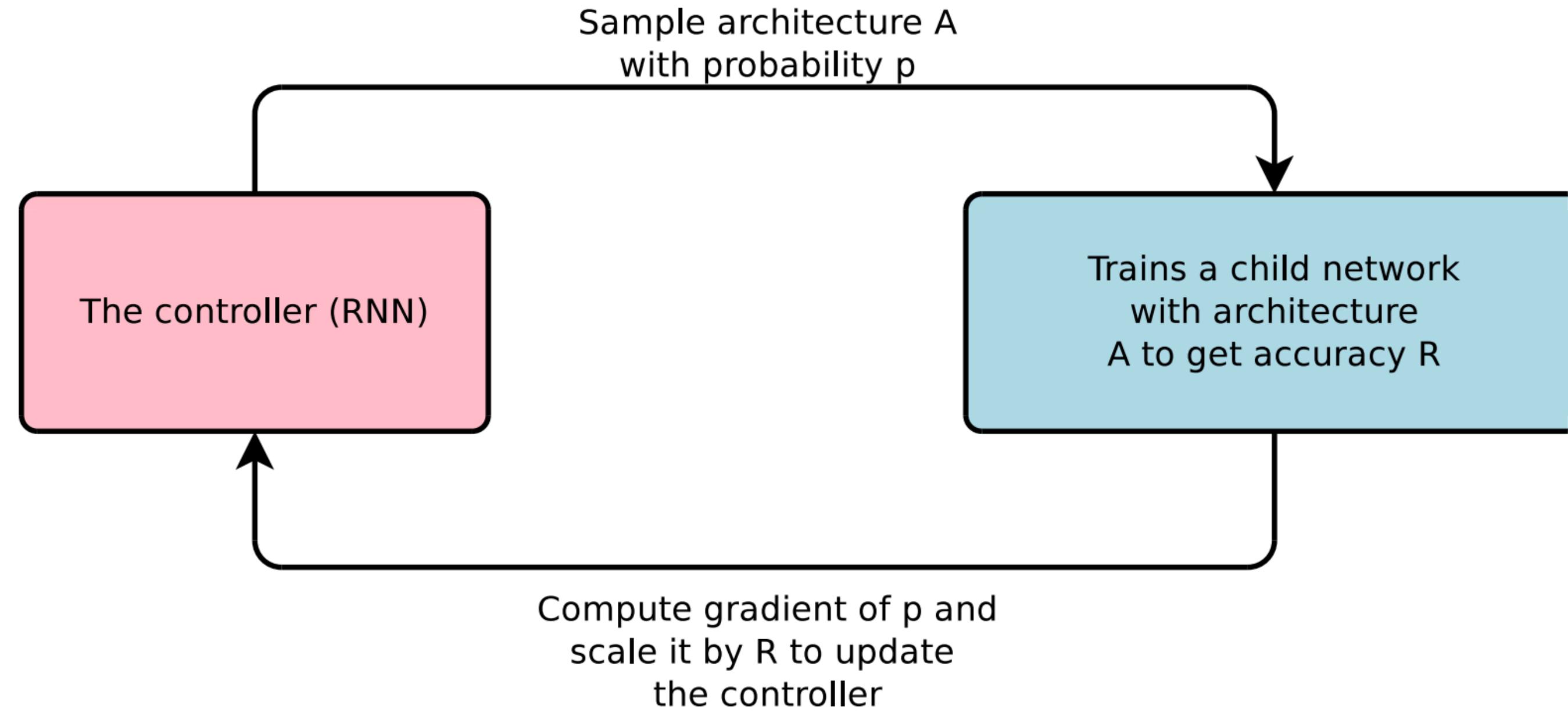
# Framework of NAS

- Search space is a set of candidate neural network architectures.
- Search strategy defines how to explore the search space.
- Accuracy estimation strategy defines how to estimate/predict the accuracy of a given neural network architecture in the design space.



# Accuracy Estimation Strategy

## Train from scratch

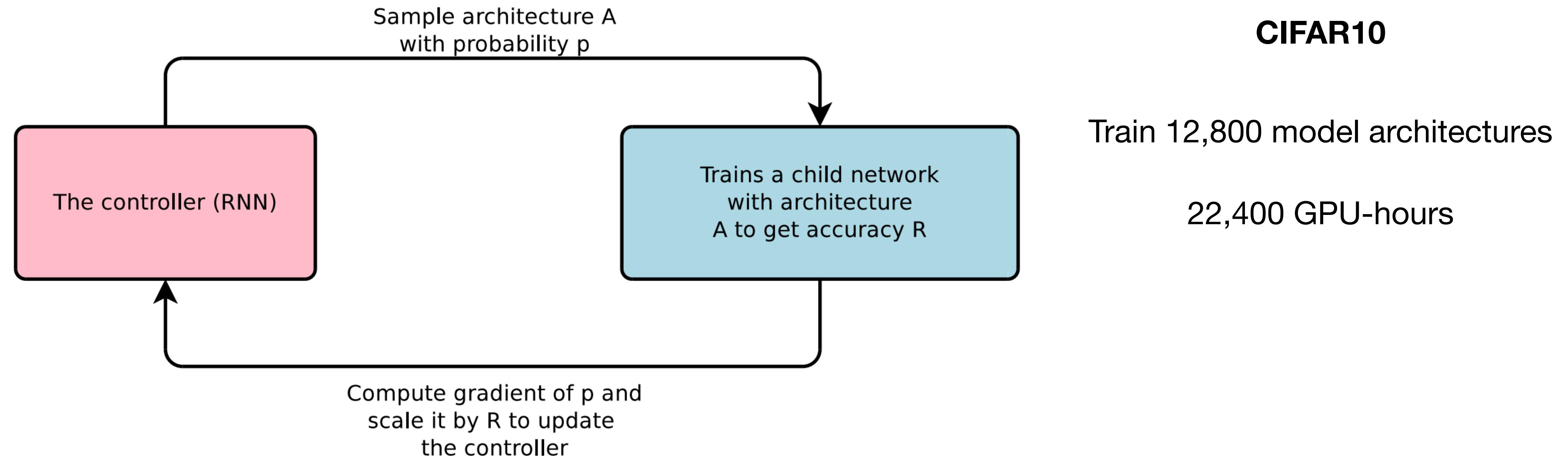


- Train the given model from scratch on the training set
- Evaluate the trained model on the validation set to get accuracy R
- Drawback: prohibitive training cost

Neural Architecture Search with Reinforcement Learning [Zoph et al., ICLR 2017]

# Accuracy Estimation Strategy

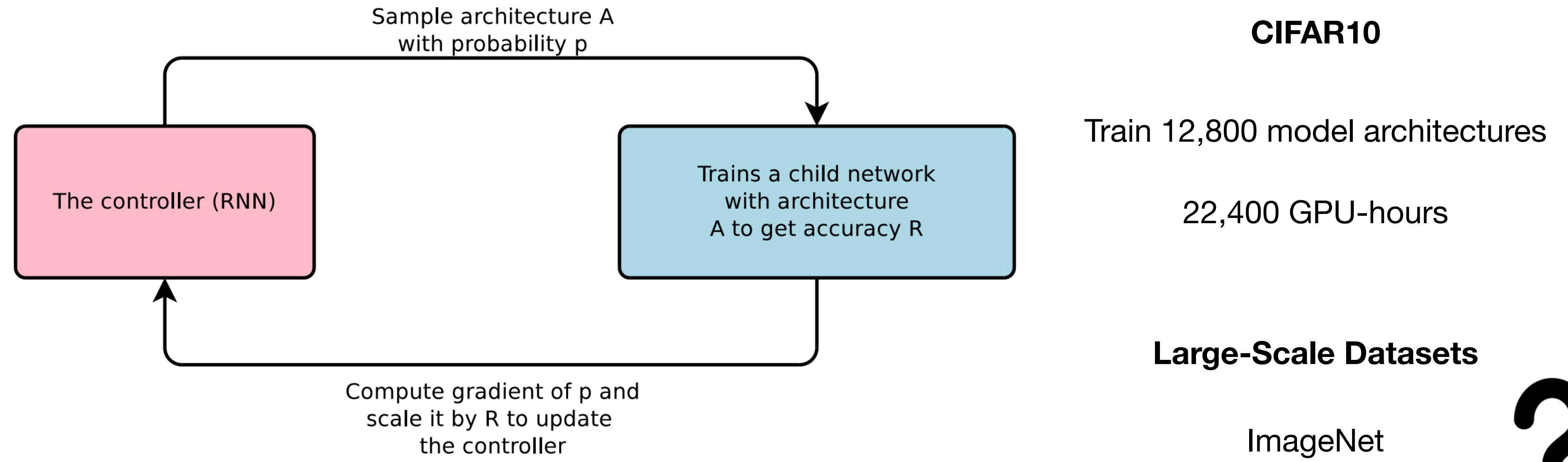
## Train from scratch



- Train the given model from scratch on the training set
- Evaluate the trained model on the validation set to get accuracy R
- Drawback: prohibitive training cost

# Accuracy Estimation Strategy

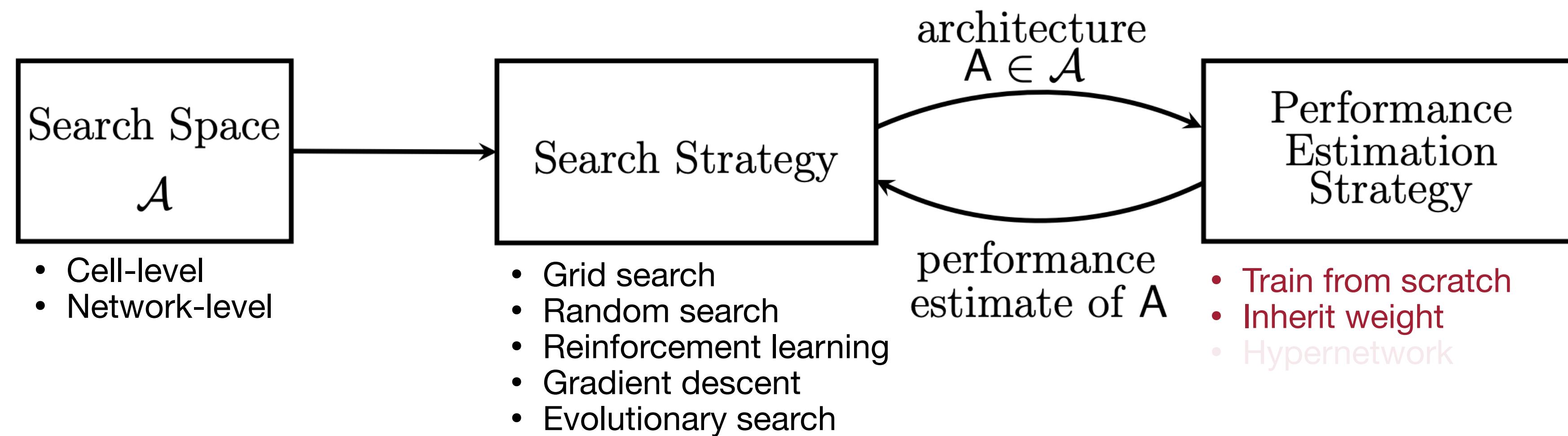
## Train from scratch



- Train the given model from scratch on the training set
- Evaluate the trained model on the validation set to get accuracy R
- Drawback: prohibitive training cost

# Framework of NAS

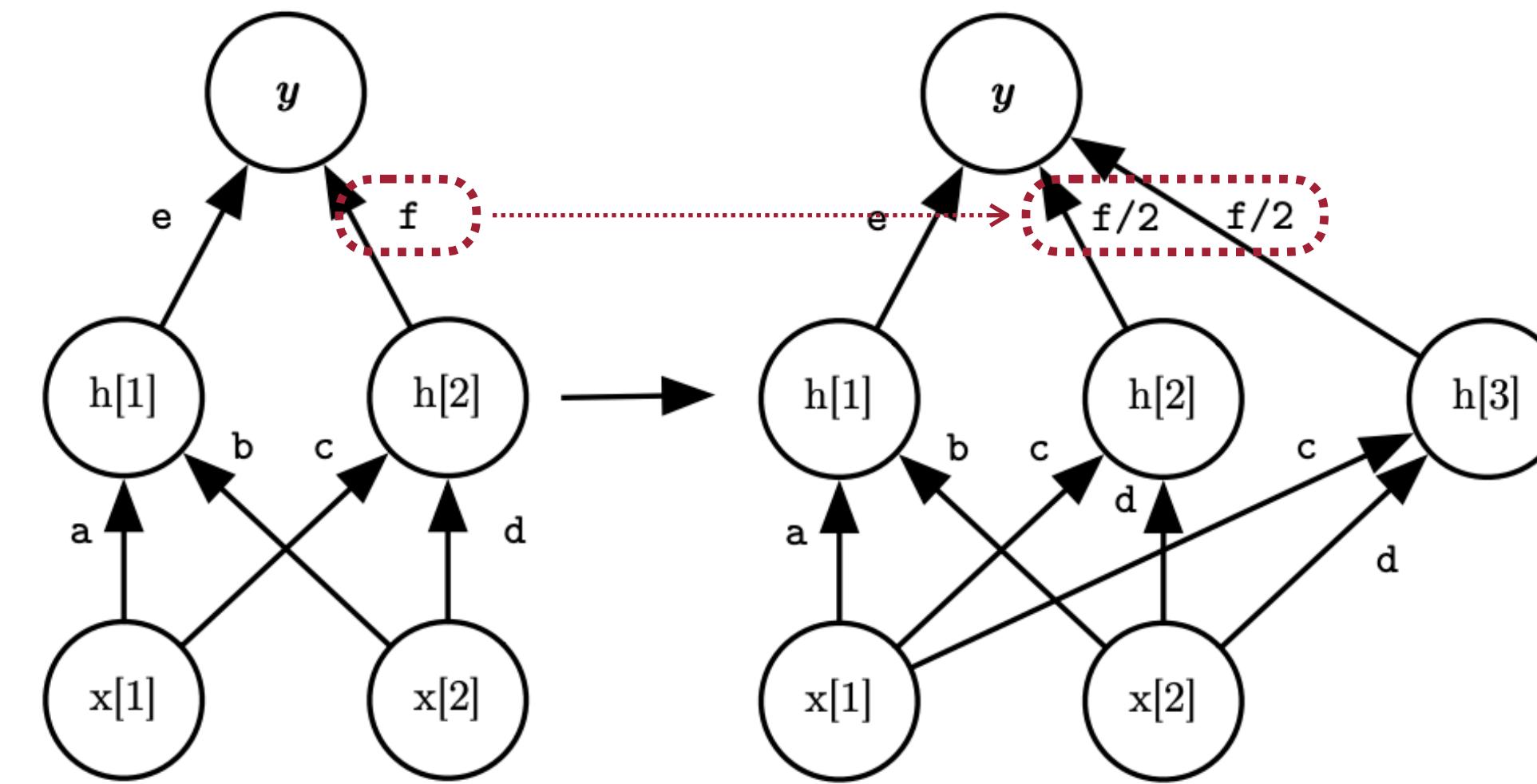
- Search space is a set of candidate neural network architectures.
- Search strategy defines how to explore the search space.
- Accuracy estimation strategy defines how to estimate/predict the accuracy of a given neural network architecture in the design space.



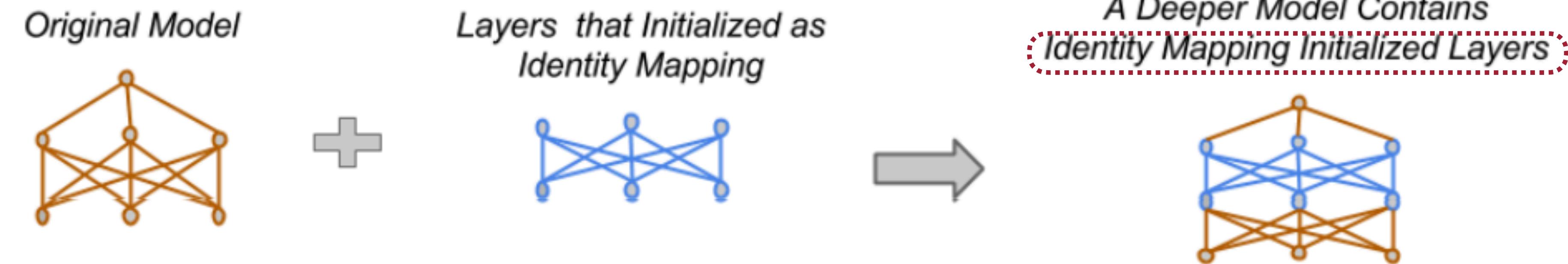
# Accuracy Estimation Strategy

## Inherit weight

Net2Wider



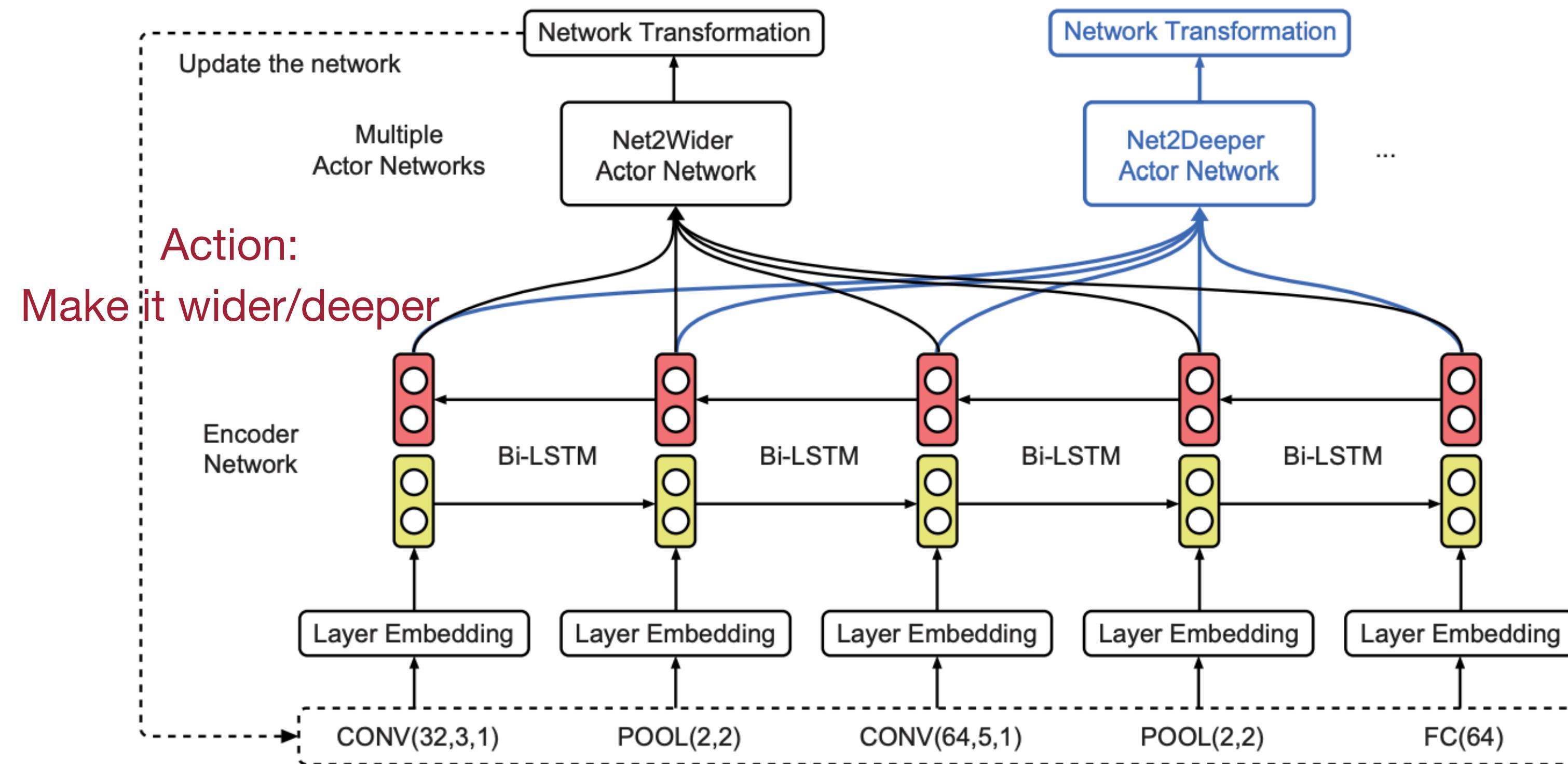
Net2Deeper



- Inherit weights from a parent model to reduce the training cost instead of training from scratch.

# Accuracy Estimation Strategy

## Inherit weight

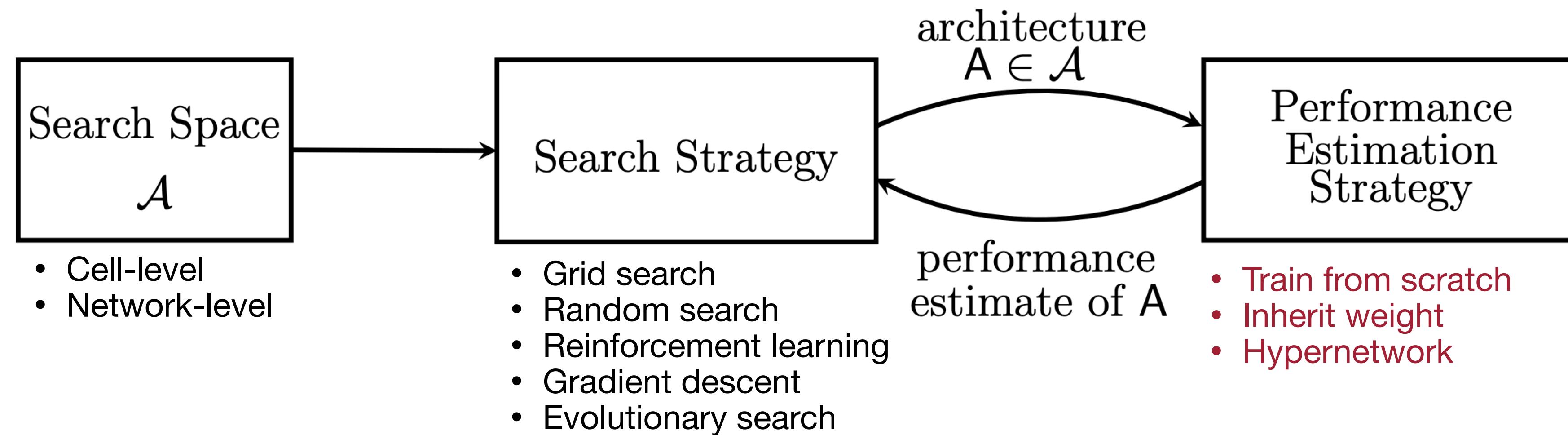


- Inherit weights from a parent model to reduce the training cost instead of training from scratch.
- Generate network transformation actions to update the model architecture instead of directly generating the model architecture.

Efficient Architecture Search by Network Transformation [Cai et al., AAAI 2018]

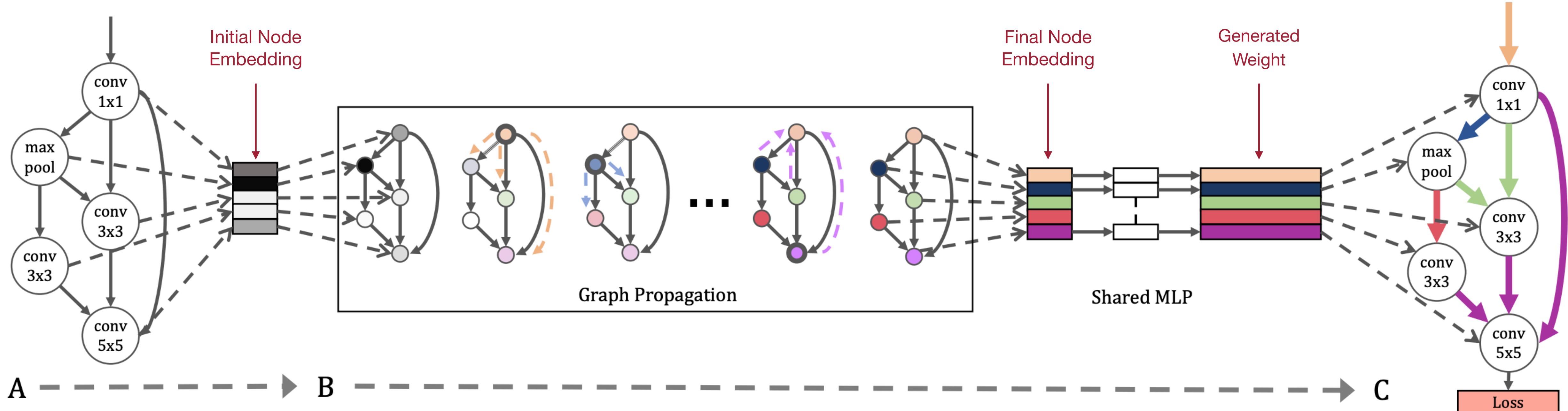
# Framework of NAS

- Search space is a set of candidate neural network architectures.
- Search strategy defines how to explore the search space.
- Accuracy estimation strategy defines how to estimate/predict the accuracy of a given neural network architecture in the design space.



# Accuracy Estimation Strategy

## Hypernet



- At each training step, a random model architecture is sampled from the search space.
- The hyper-network generates weights based on the model architecture.
- Using gradient descent to update the weights of the hyper-network.

SMASH: One-Shot Model Architecture Search through HyperNetworks [Brock et al., ICLR 2018]

image source: [link](#)

# Neural Architecture Search

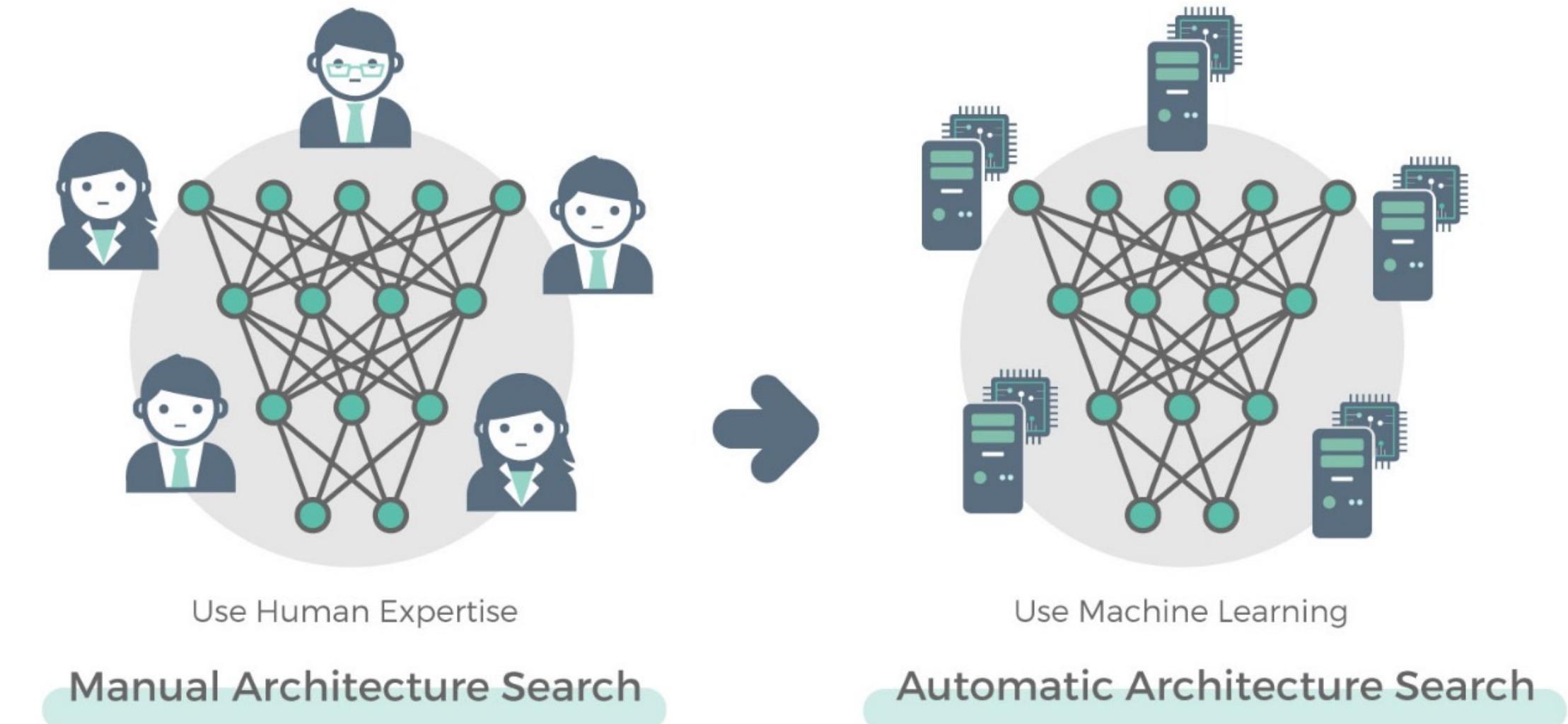
- **Last Lecture:**
  - Primitive operations
  - Classic building blocks
  - **Introduction to neural architecture search (NAS)**

- What is NAS?
- Search space
- Design the search space
- Search strategy

- **This Lecture:**

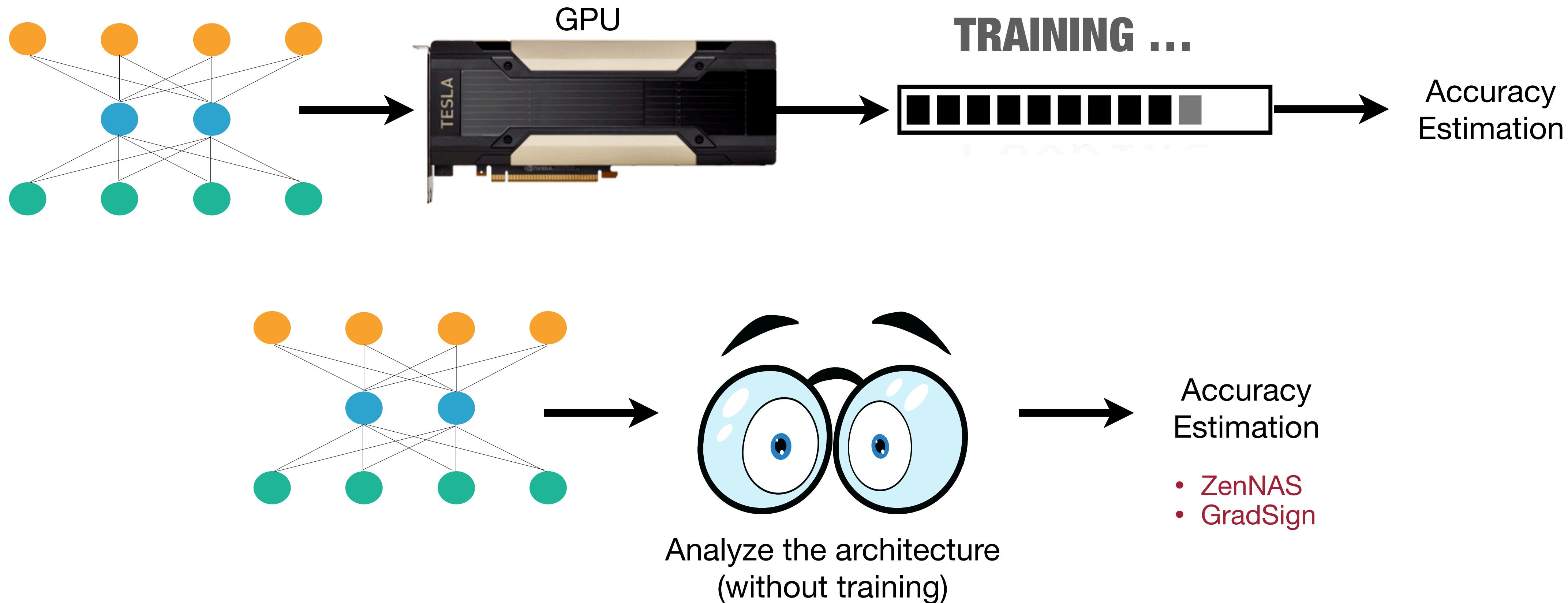
- **Efficient and hardware-aware NAS**

- Accuracy estimation strategy
- Zero-shot NAS
- Hardware-aware NAS
- Neural-hardware architecture co-search
- **NAS applications**
- NLP, GAN, point cloud, pose, etc



# Zero-Shot NAS

Estimate the accuracy without training it



# Zero-Shot NAS

## ZenNAS

Implementation:

1. Initialize the random input  $x \sim \mathcal{N}(0,1)$ ,
2. Add a small perturbation to  $x$  and obtain  $x' = x + \epsilon$ ,
3. Initialize all the weights of the neural network  $f$  to a normal distribution  $\mathcal{N}(0,1)$ ,
4. Calculate the log difference:  $z_1 = \log(f(x') - f(x))$ : **a good model should be sensitive to input perturbations**
5. + batch normalization variance:

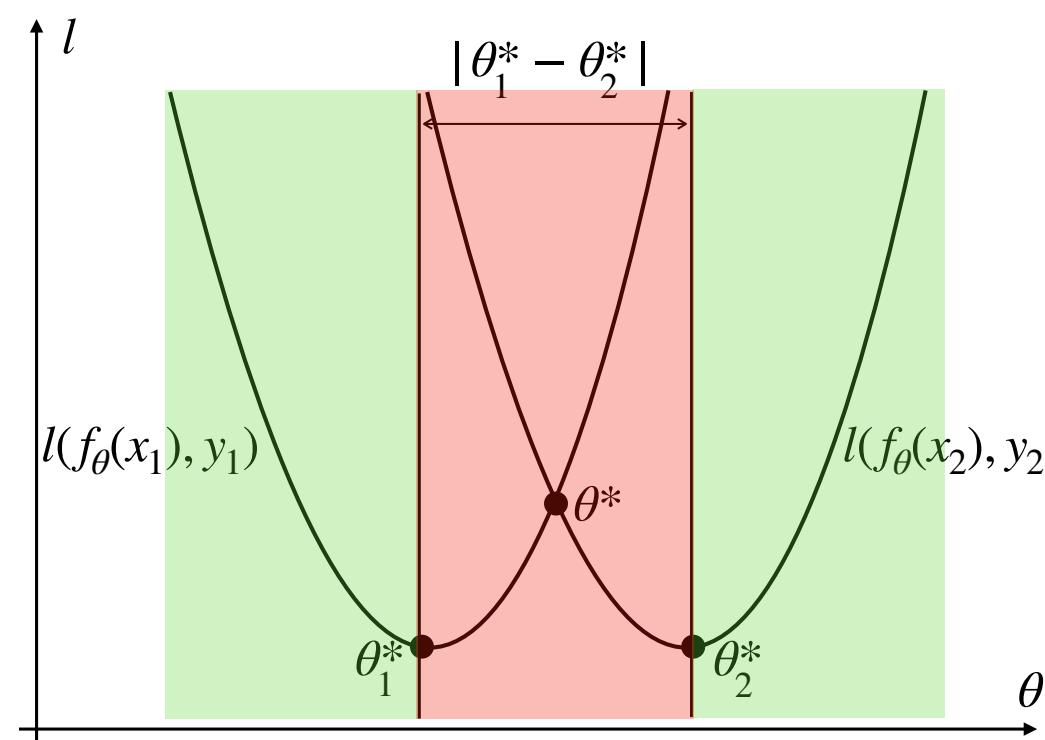
Consider BN for layer  $i$ :  $x_i = (x_i - \mu_i)/\sigma_i$ . We calculate the square mean  $\bar{\sigma}_i$  as  $\sqrt{\frac{\sum_j \sigma_{i,j}^2}{c_{\text{out}}}}$ .

Finally, we have  $z_2 = \sum_i \log \bar{\sigma}_i$  and zen score  $z = z_1 + z_2$ .

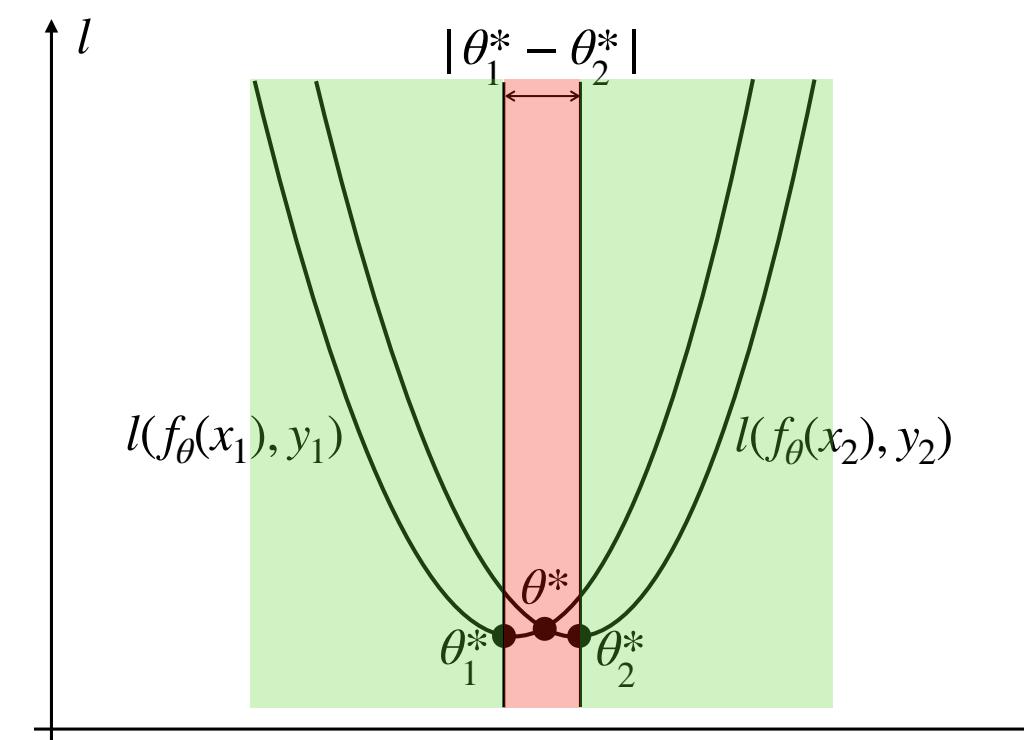
# Zero-Shot NAS

## GradSign

- **Intuition:** A good model has denser sample-wise local minima, i.e. the distance between local minima points  $\theta_1^*$  and  $\theta_2^*$  should be smaller. In this case, there is a higher probability in (b) that **the gradients of different samples have the same sign (green region).**



(a) Optimization landscape with **sparser** sample-wise local optima corresponding to **worse**  $J(\theta^*)$ .



(b) Optimization landscape with **denser** sample-wise local optima corresponding to **better**  $J(\theta^*)$ .

### Algorithm 1: GradSign

**Result:** GradSign score  $\tau_f$  for a function class  $f_\theta$   
Given  $\mathcal{S} = \{(x_i, y_i)\}_{i \in [n]}$ , randomly select initialization point  $\theta_0$ ;  
Initialize  $g[n, m]$ ;  
**for**  $i = 1, 2, \dots, n$  **do**  $i, n$  corresponds to samples  
    **for**  $k = 1, 2, \dots, m$  **do**  $k, m$  corresponds to layers  
         $| g[i, k] = \text{sign}([\nabla_\theta l(f_\theta(x_i), y_i)|_{\theta_0}]_k)$   
    **end**  
**end**  $| \sum_{i=1}^n g[i, k] | \leq n$ , “=” is achieved only when the gradients at all samples have the same sign.  
**return**  $\tau_f = \sum_k |\sum_i g[i, k]|$ ;

# Neural Architecture Search

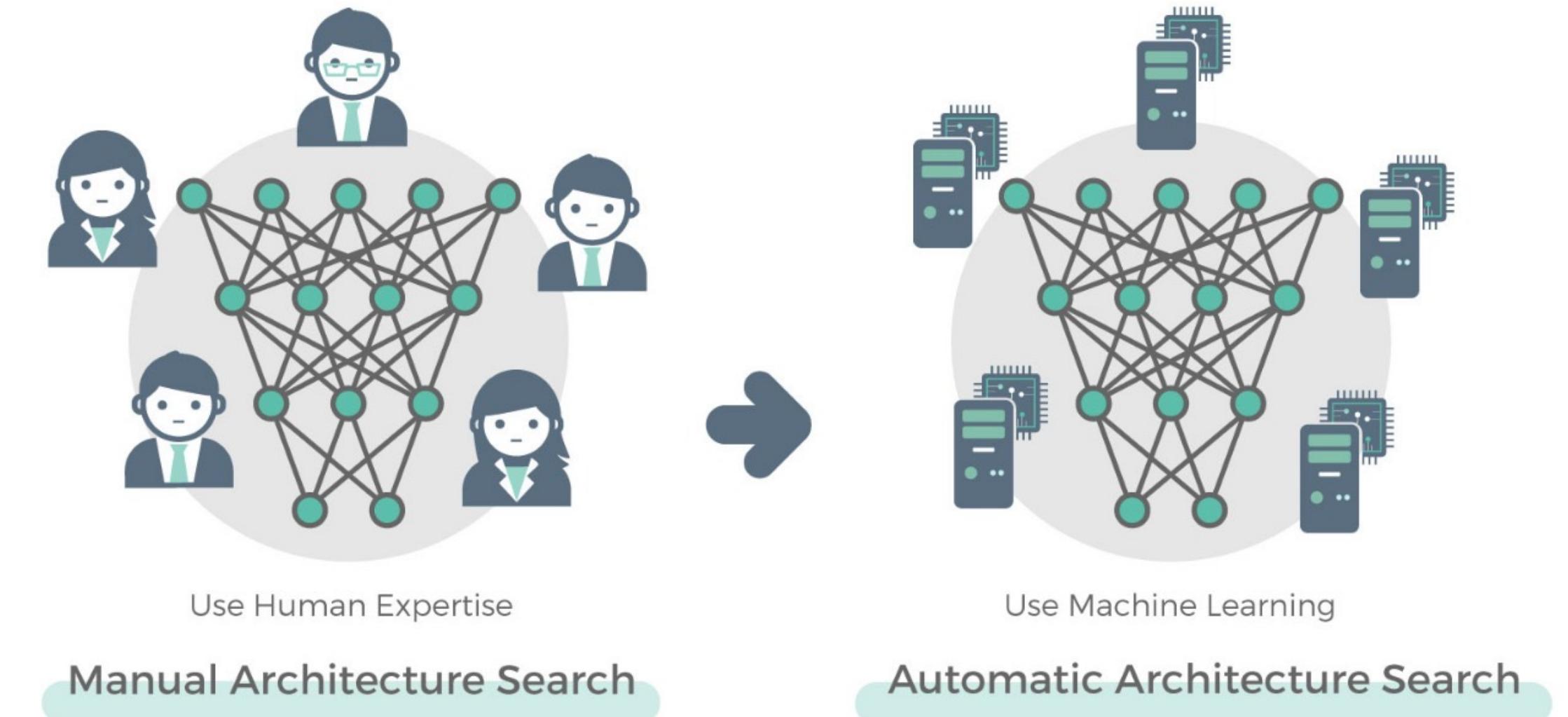
- **Last Lecture:**
  - Primitive operations
  - Classic building blocks
  - **Introduction to neural architecture search (NAS)**

- What is NAS?
- Search space
- Design the search space
- Search strategy

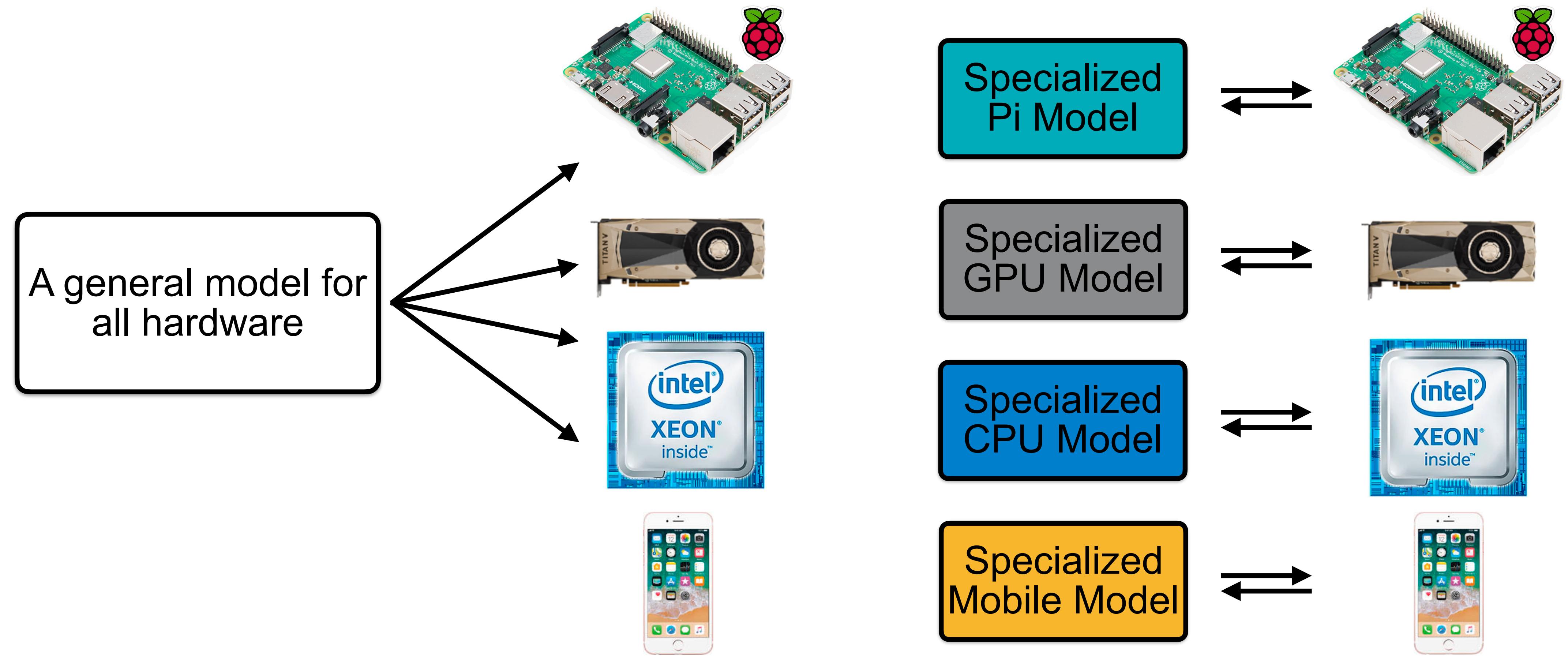
- **This Lecture:**

- **Efficient and hardware-aware NAS**

- Accuracy estimation strategy
- Zero-shot NAS
- Hardware-aware NAS
  - Neural-hardware architecture co-search
- **NAS applications**
  - NLP, GAN, point cloud, pose, etc

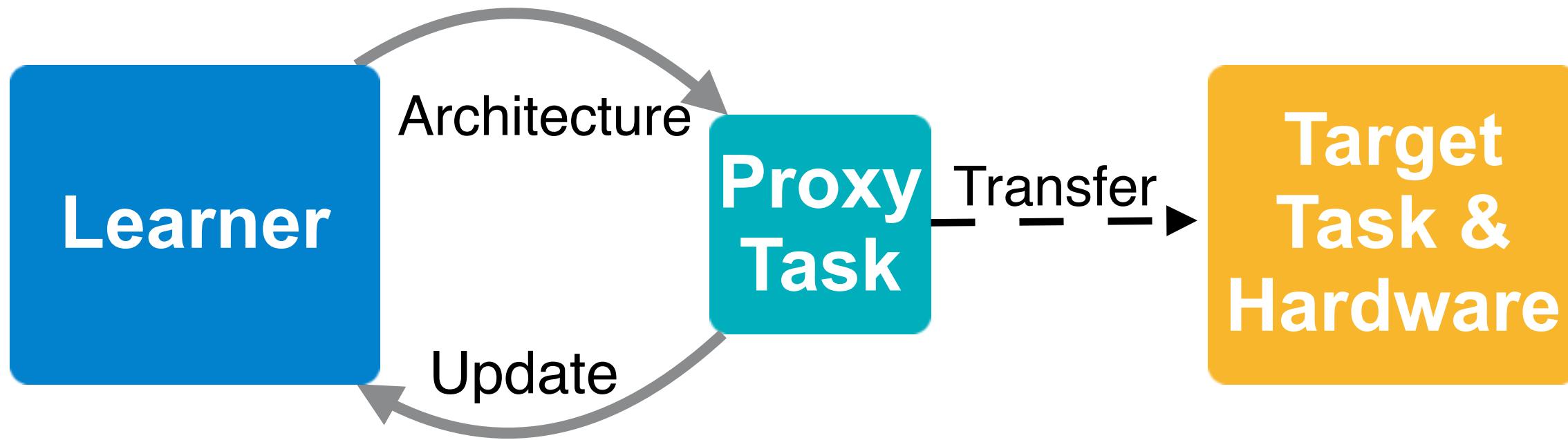


# From General Model to Specialized Models



# Efficiently Search on Target Task & Hardware

## Proxy task: sub-optimal



Previous neural architecture search (NAS) is **very expensive**.

- NASNet: 48,000 GPU hours, even on Cifar  $\approx$  5 years on single GPU
- DARTS: 100GB GPU memory if directly search on ImageNet

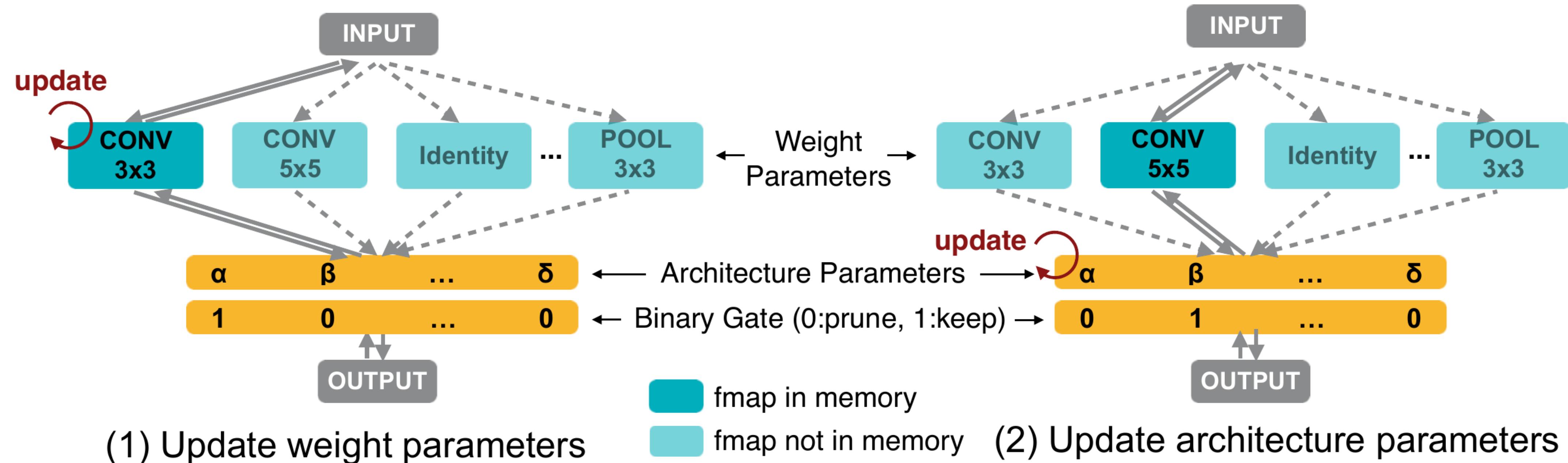
Therefore, previous work have to utilize **proxy tasks**:

- CIFAR-10
- Small architecture space (e.g. low depth)
- Fewer epochs training
- Flops and parameter counts

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai et al., ICLR 2019]

# Efficiently Search on Target Task & Hardware

## ProxylessNAS

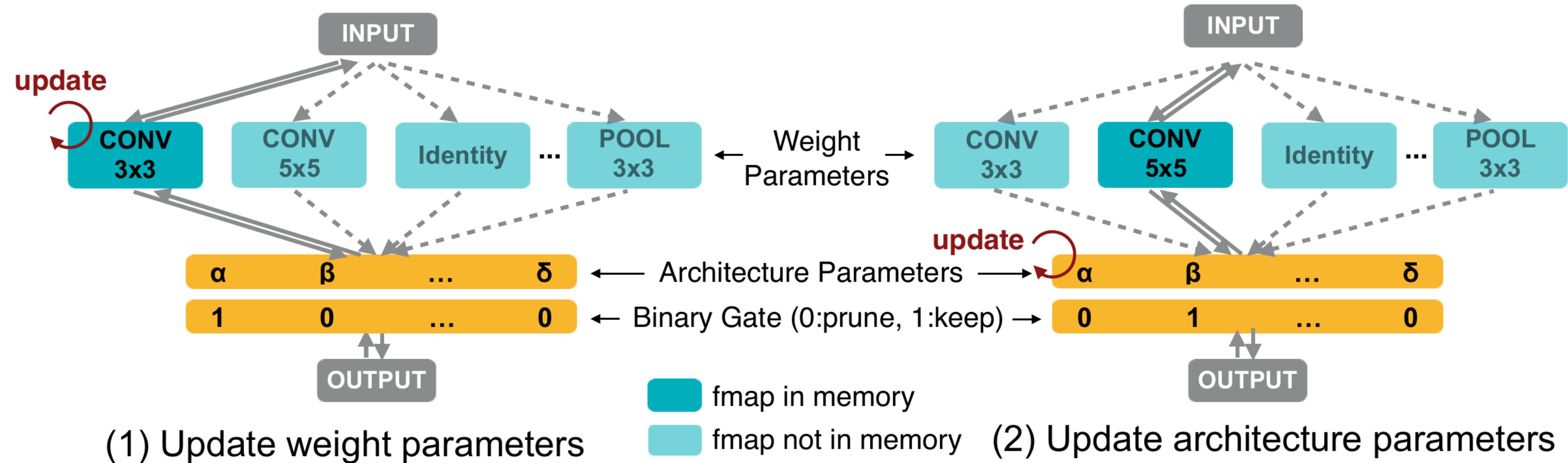


- Build the over-parameterized network **with all candidate paths**
- Simplify NAS to be a **single training process** of a over-parameterized network.
- **Pruning** redundant paths based on architecture parameters

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai et al., ICLR 2019]

# Efficiently Search on Target Task & Hardware

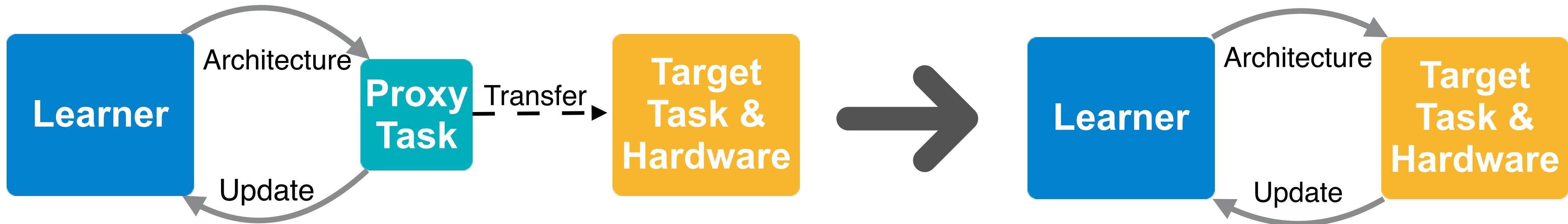
## ProxylessNAS



**Binarize the architecture parameters and allow only one path of activation to be active.**  
Thereby, the memory footprint reduces from  $O(N)$  to  $O(1)$ .

# Efficiently Search on Target Task & Hardware

## ProxylessNAS



Previous neural architecture search (NAS) is **very expensive**.

- NASNet: 48,000 GPU hours, even on Cifar  $\approx$  5 years on single GPU
- DARTS: 100GB GPU memory if directly search on ImageNet

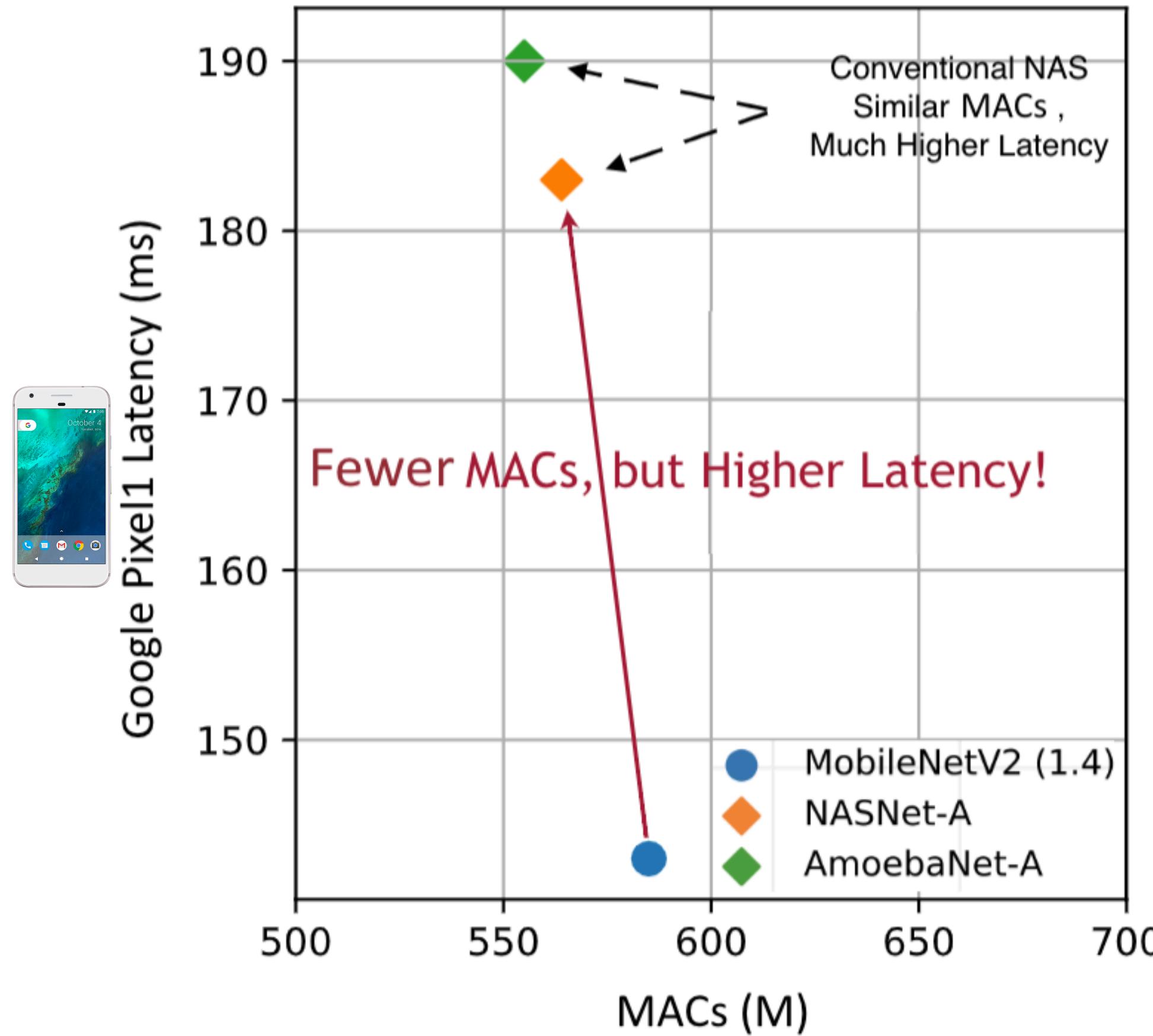
Therefore, previous work have to utilize **proxy tasks**:

- CIFAR-10
  - Small architecture space (e.g. low depth)
  - Fewer epochs training
  - Flops and parameter counts
- ImageNet
  - Large architecture space
  - Full training
  - Profiled Latency

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai et al., ICLR 2019]

# Search for Efficient Model

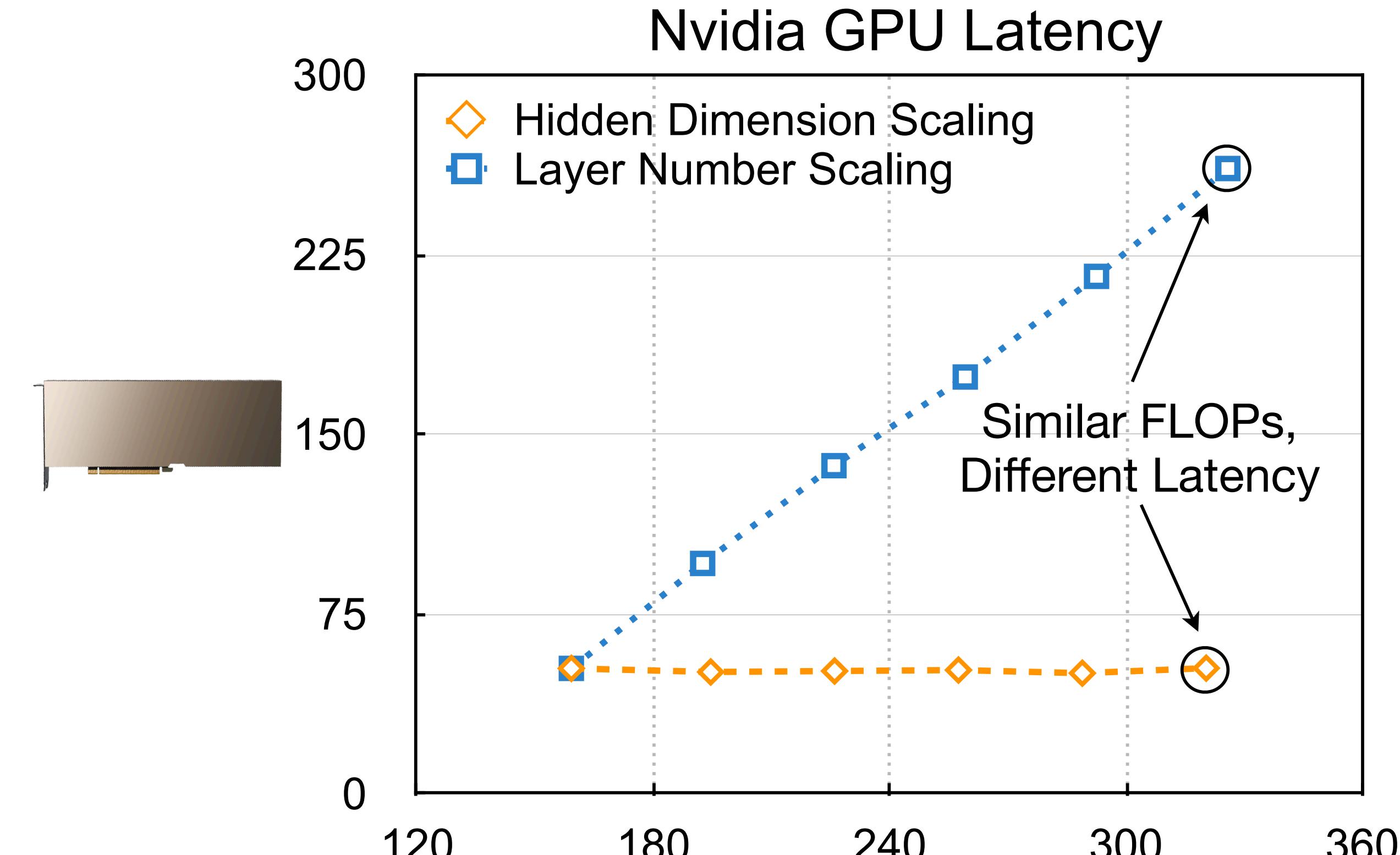
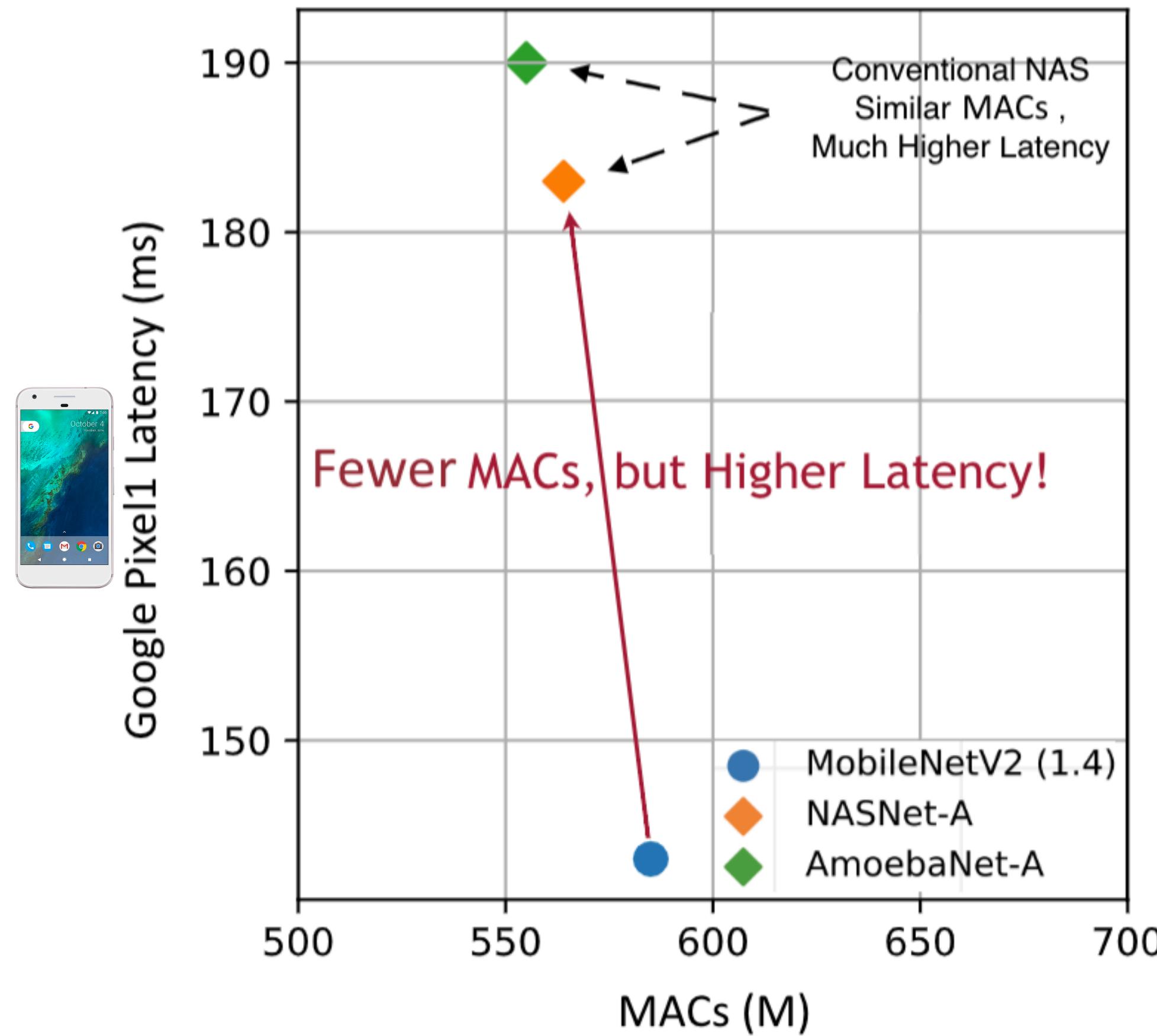
MACs != Real Hardware Efficiency



ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai et al., ICLR 2019]

# Search for Efficient Model

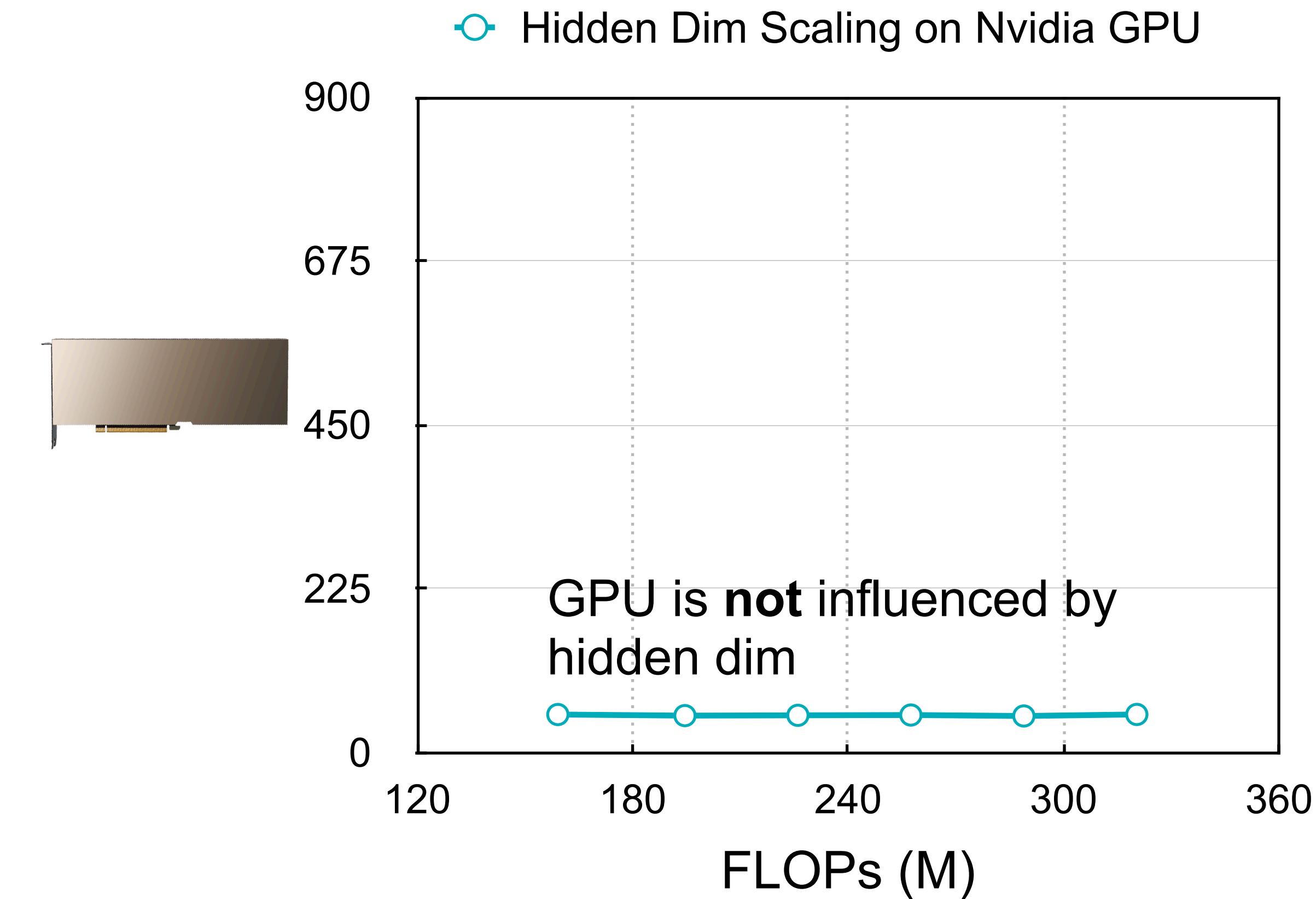
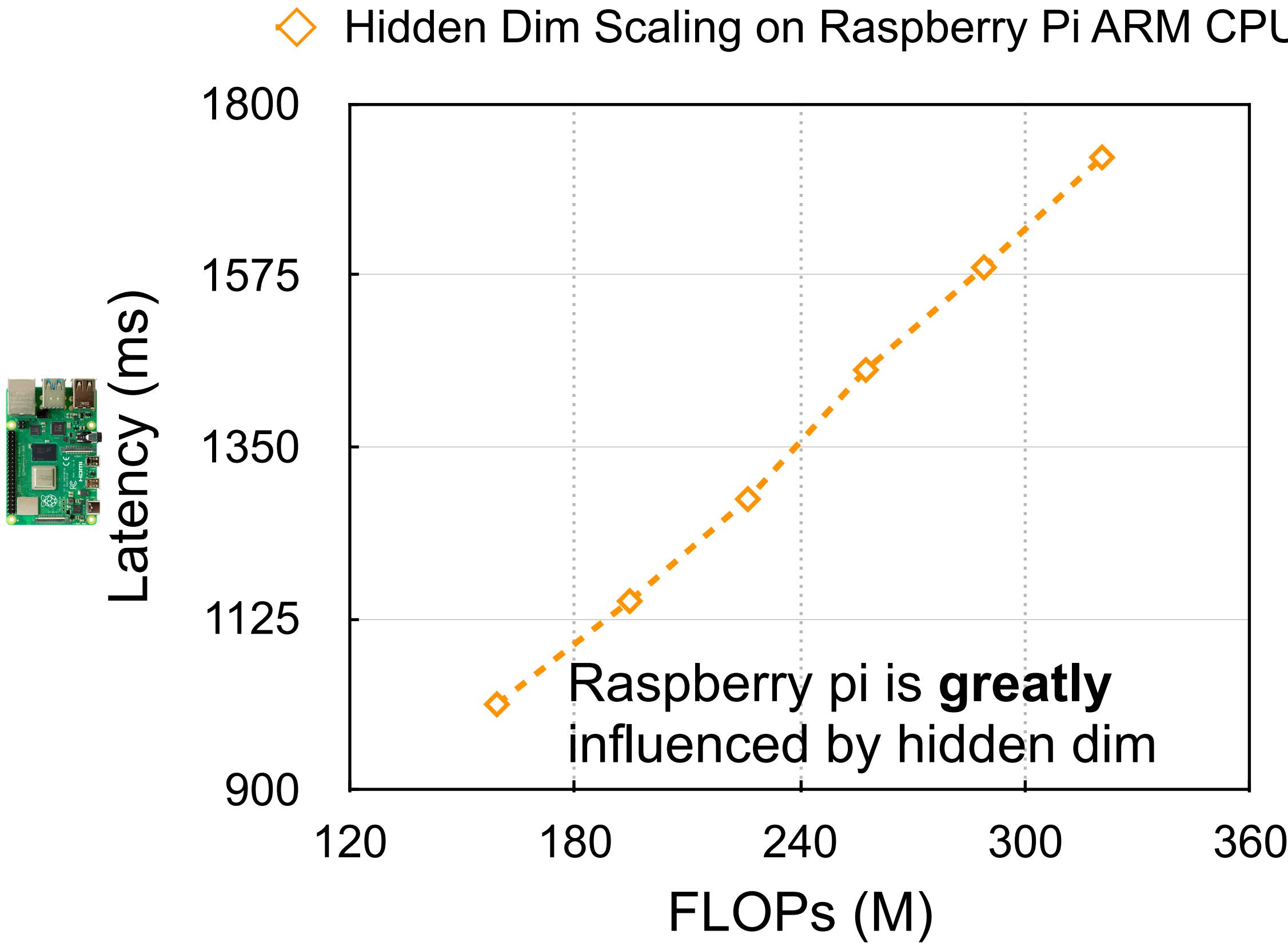
MACs != Real Hardware Efficiency



ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai et al., ICLR 2019]  
HAT: Hardware-Aware Transformers for Efficient Natural Language Processing [Wang et al., ACL 2020]

# Search for Efficient Model

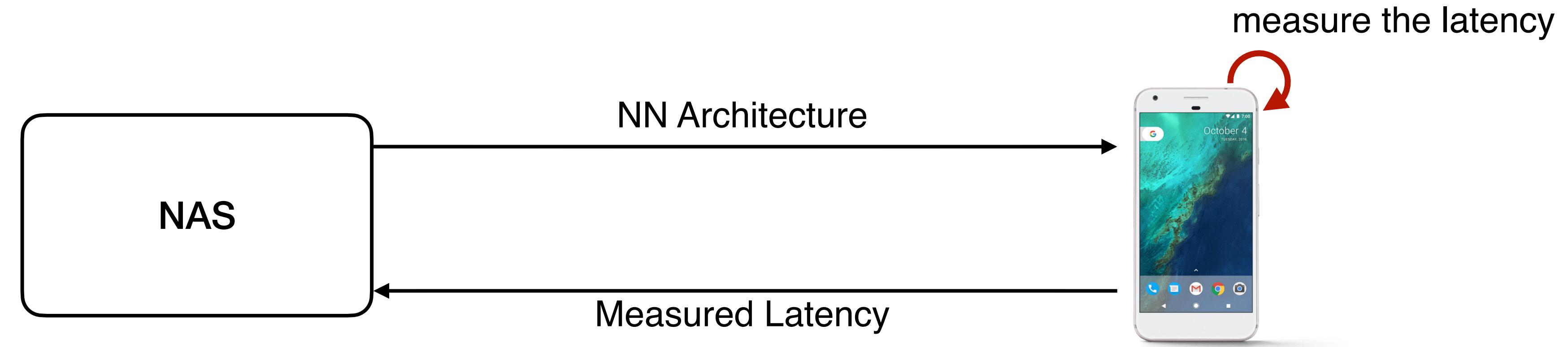
MACs != Real Hardware Efficiency



HAT: Hardware-Aware Transformers for Efficient Natural Language Processing [Wang et al., ACL 2020]

# Incorporate Hardware Feedback into NAS

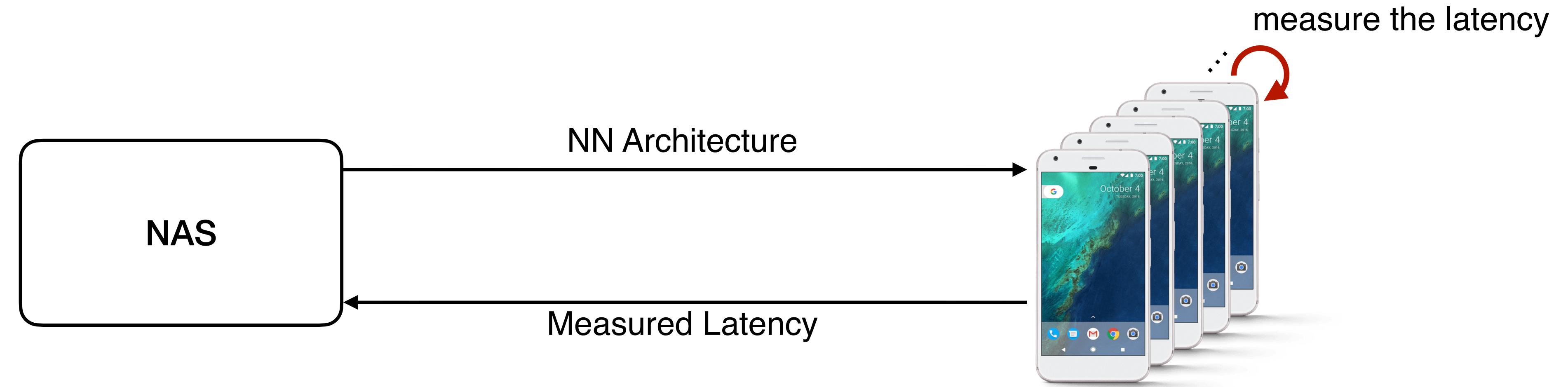
Measure the latency on device: slow



Mnasnet: Platform-Aware Neural Architecture Search for Mobile [Tang et al., CVPR 2019]

# Incorporate Hardware Feedback into NAS

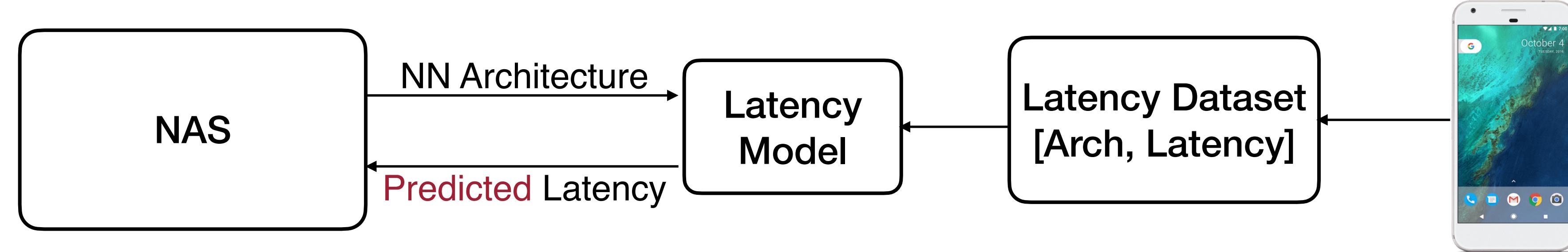
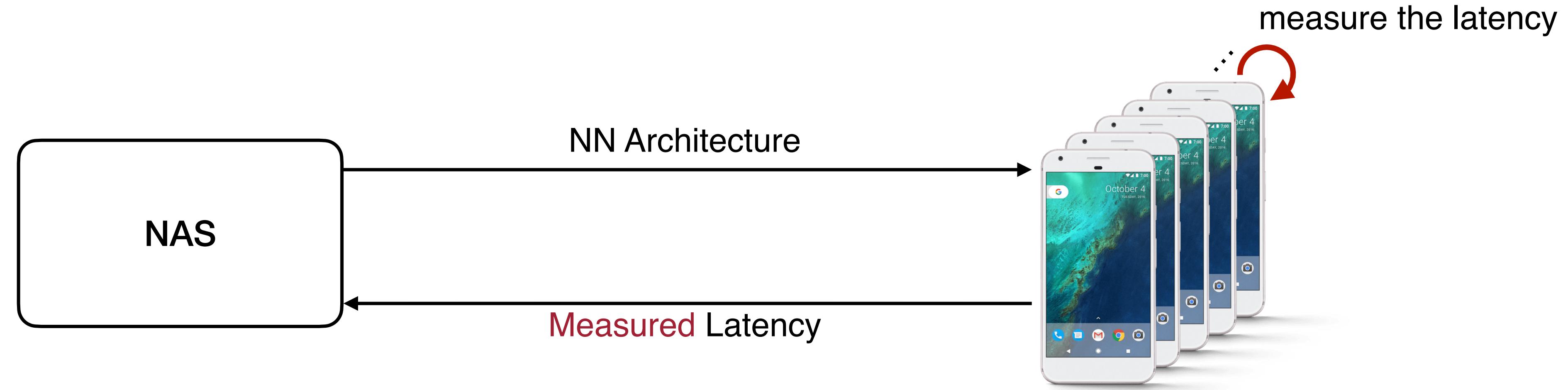
Measure the latency on device: expensive



Mnasnet: Platform-Aware Neural Architecture Search for Mobile [Tang et al., CVPR 2019]

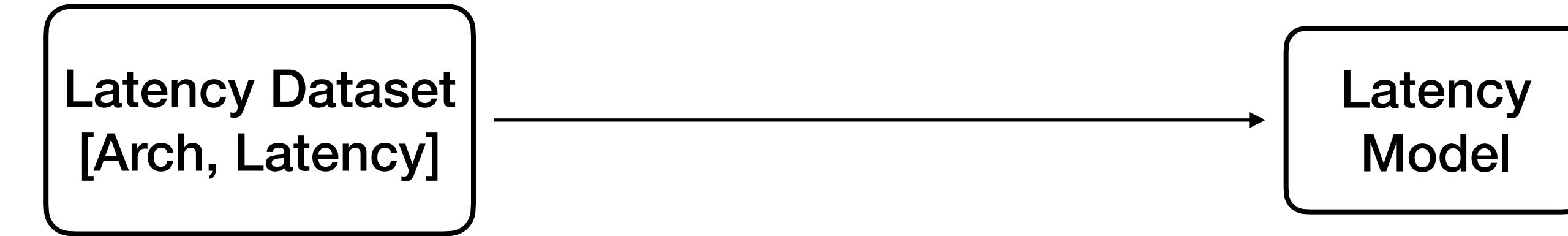
# Incorporate Hardware Feedback into NAS

Latency prediction: fast and low cost



ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai et al., ICLR 2019]

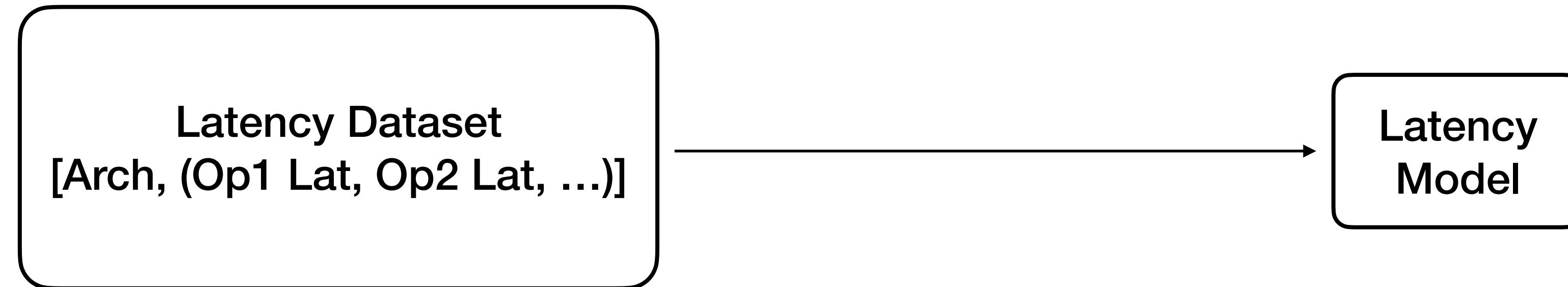
# Latency Prediction



- Layer-wise latency profiling: latency lookup table

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai et al., ICLR 2019]

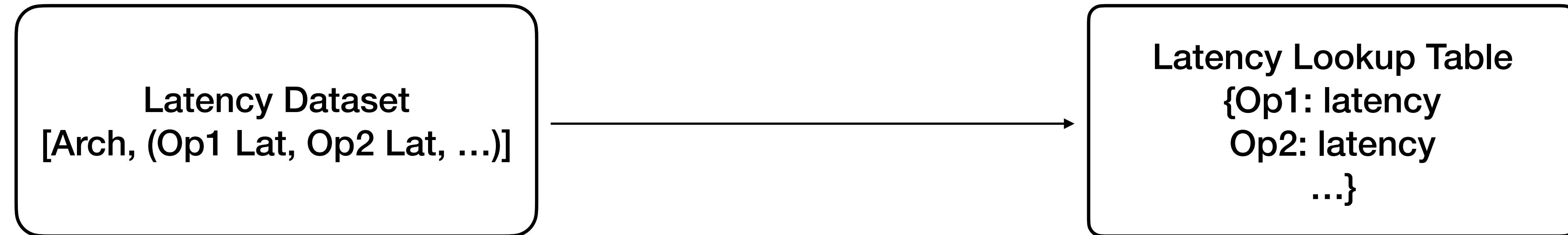
# Latency Prediction



- Layer-wise latency profiling: latency lookup table

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai et al., ICLR 2019]

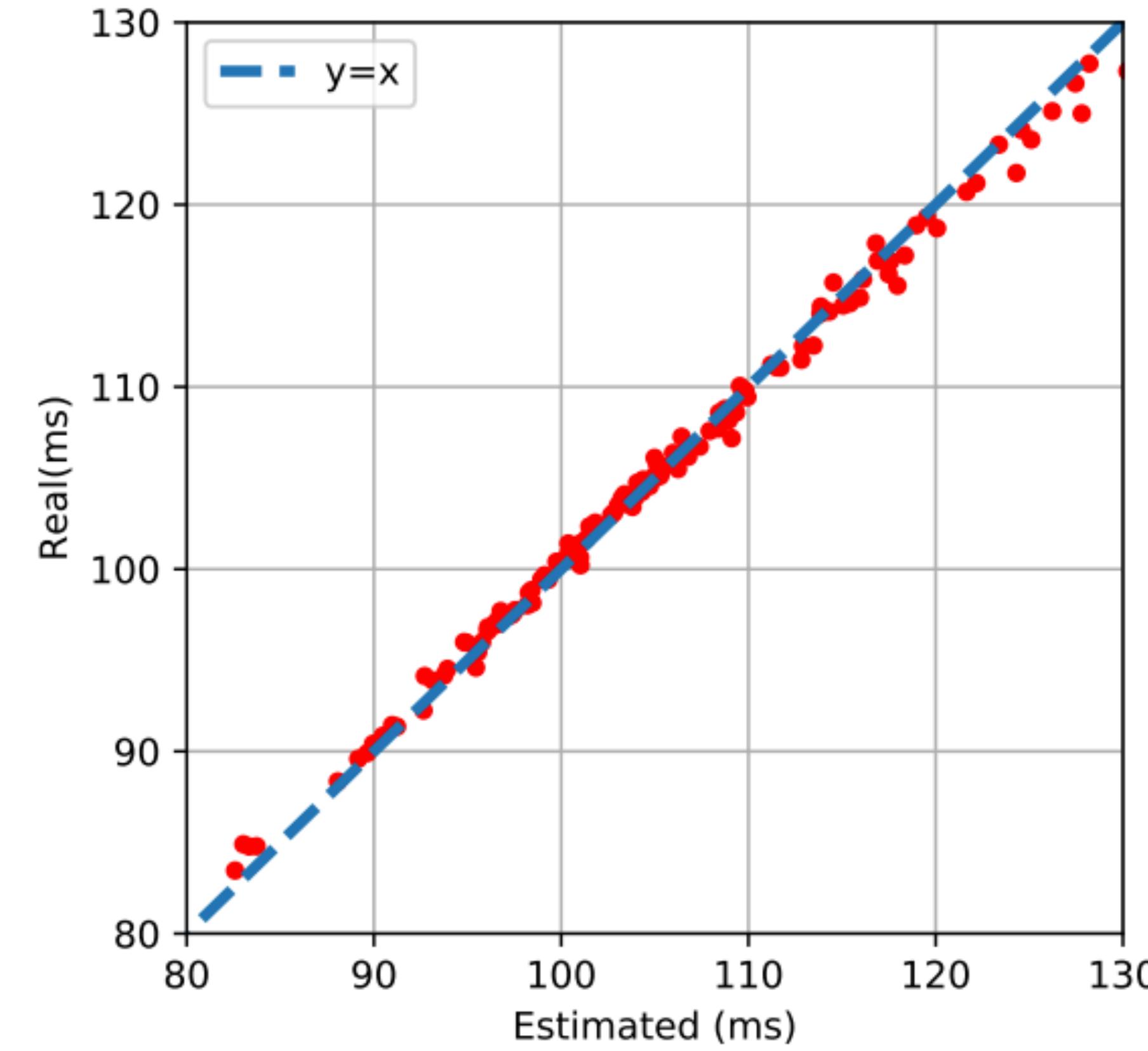
# Latency Prediction



- Layer-wise latency profiling: latency lookup table

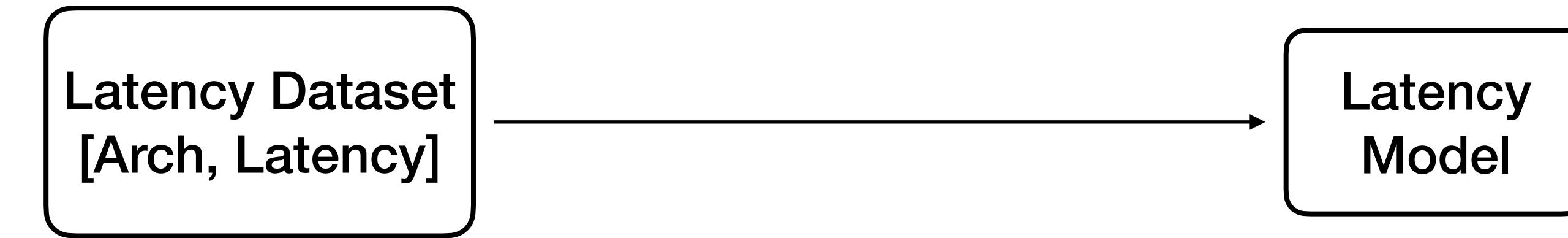
# Latency Prediction

## Latency Lookup Table



ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai et al., ICLR 2019]

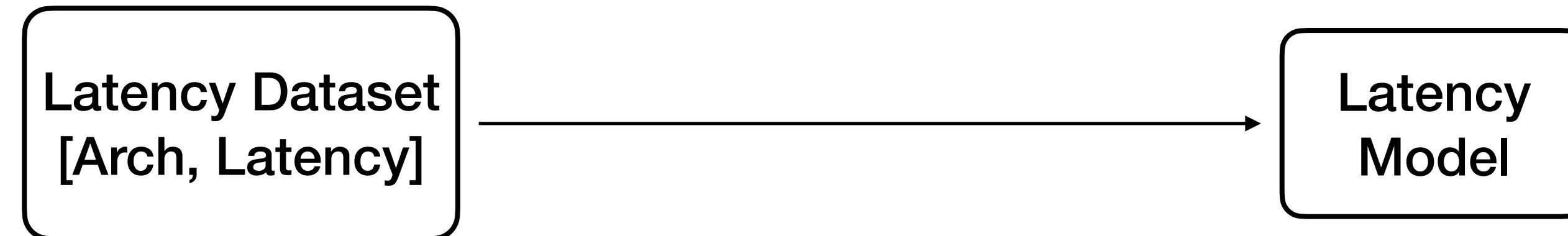
# Latency Prediction



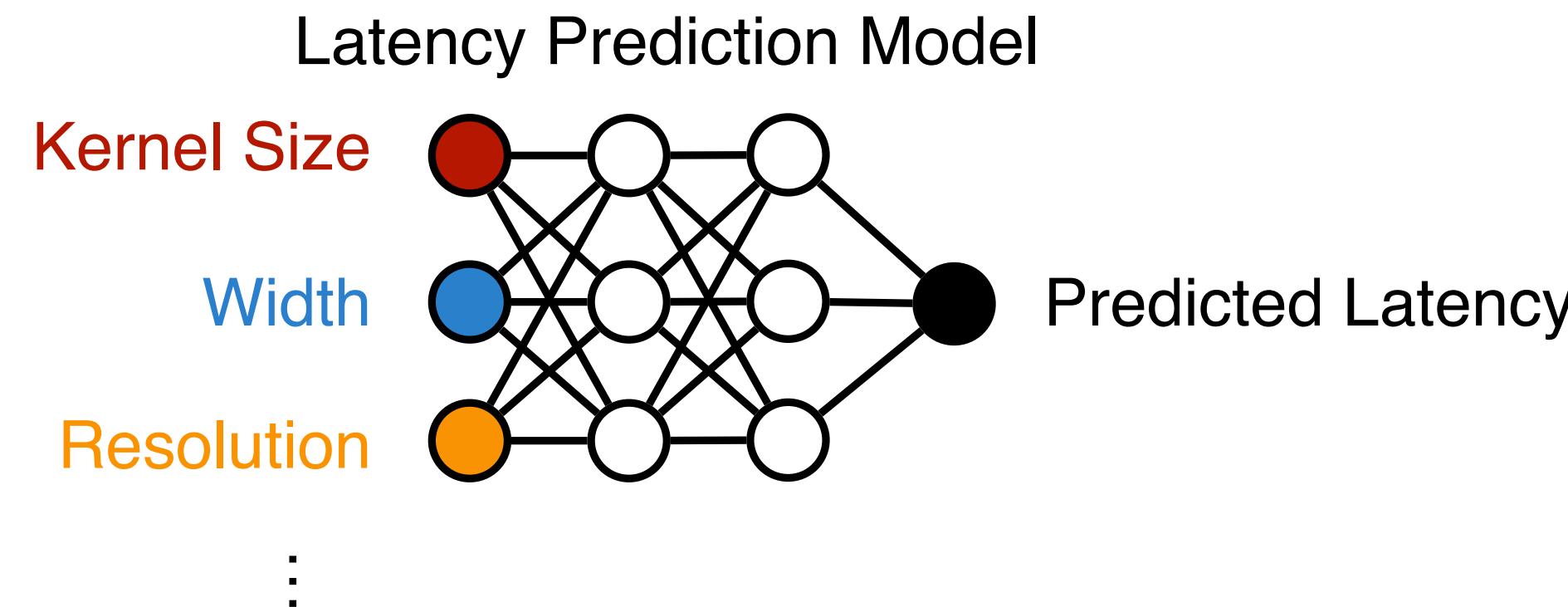
- Network-wise latency profiling: latency prediction model

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai et al., ICLR 2019]

# Latency Prediction

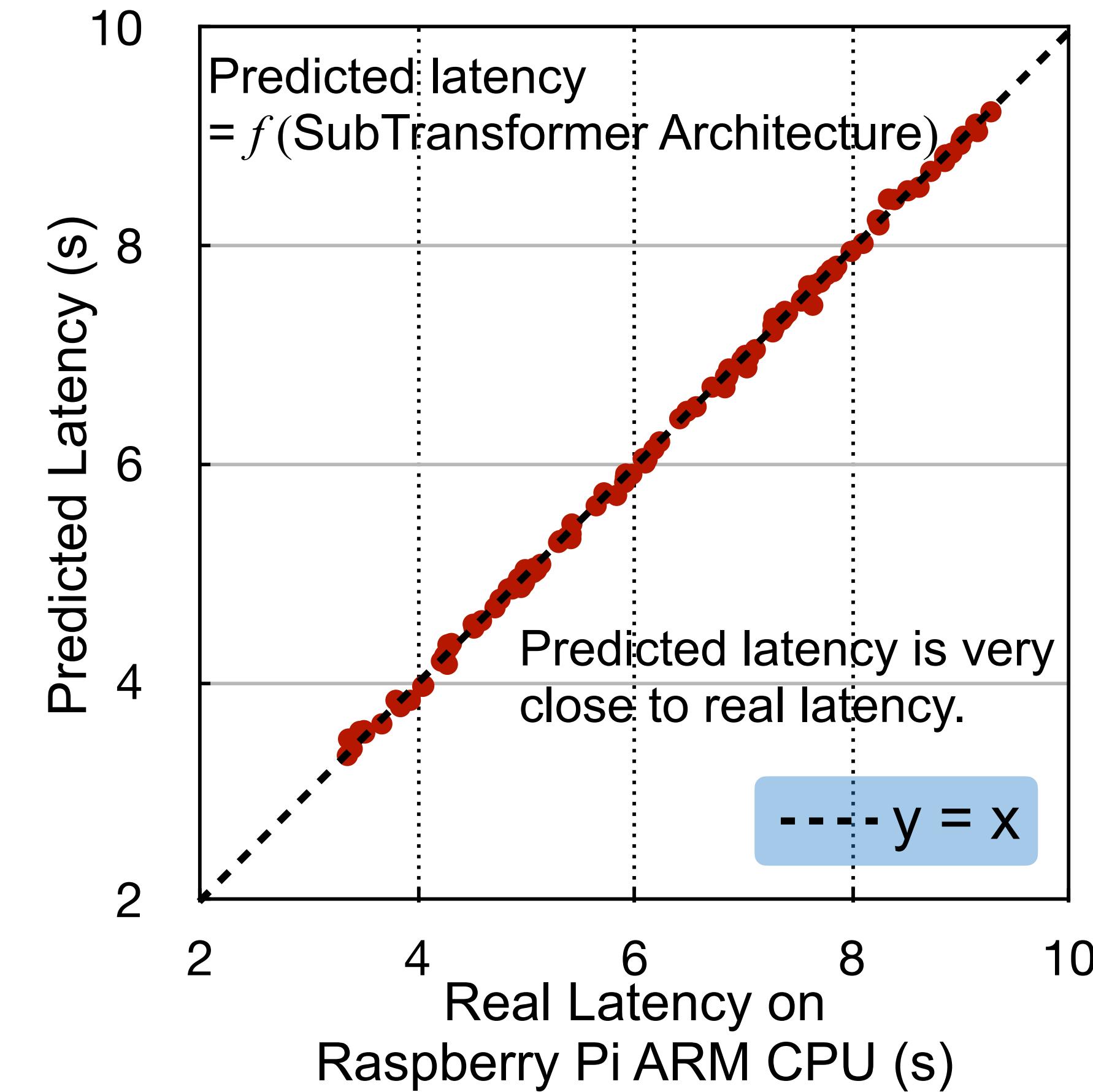


- Network-wise latency profiling: latency prediction model



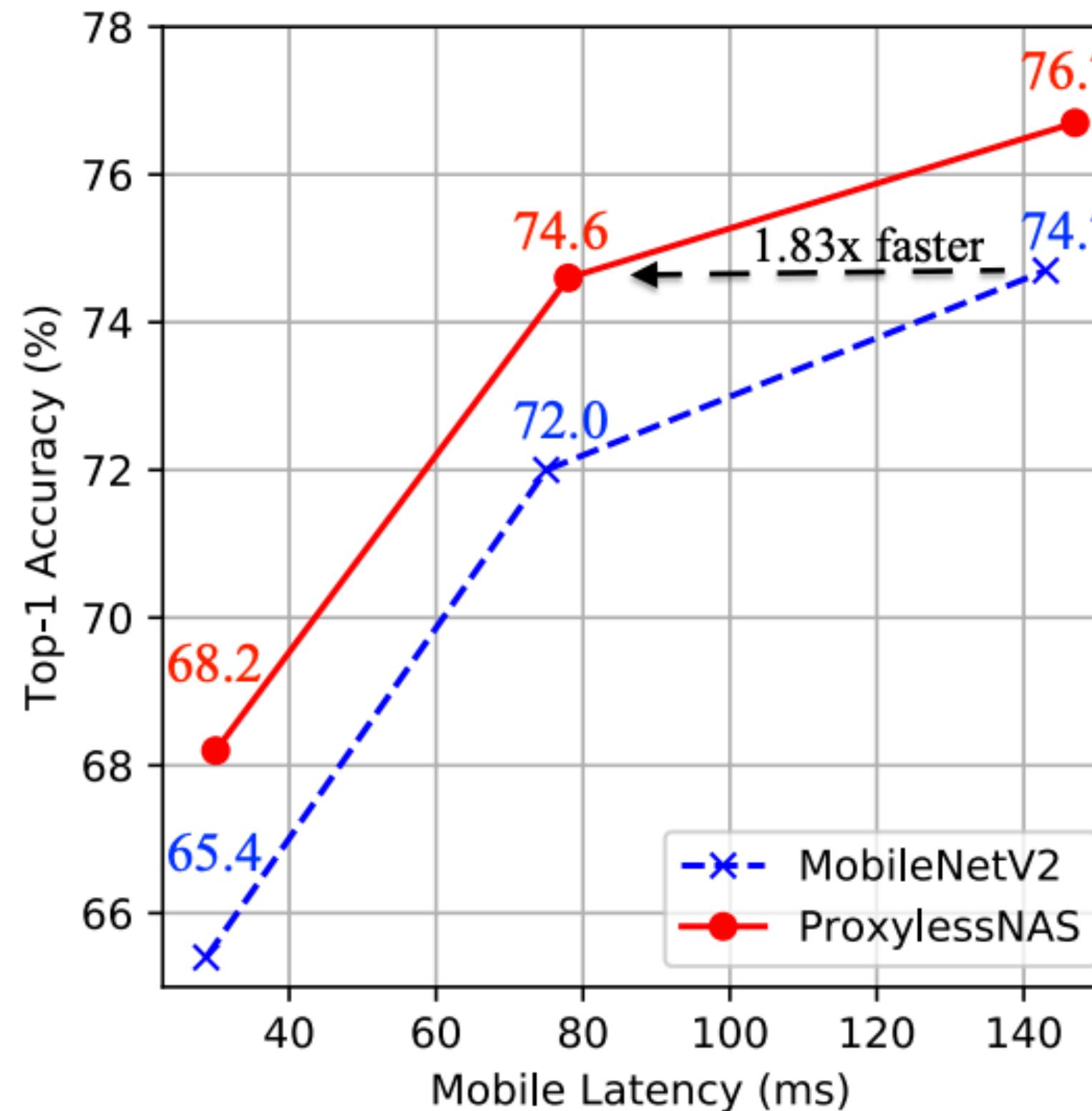
ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai et al., ICLR 2019]  
HAT: Hardware-Aware Transformers for Efficient Natural Language Processing [Wang et al., ACL 2020]

# Latency Prediction



ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai et al., ICLR 2019]  
HAT: Hardware-Aware Transformers for Efficient Natural Language Processing [Wang et al., ACL 2020]

# Specialized Models for Different Hardware



Model	Top-1	Top-5	GPU latency
MobileNetV2 (Sandler et al., 2018)	72.0	91.0	6.1ms
ShuffleNetV2 (1.5) (Ma et al., 2018)	72.6	-	7.3ms
ResNet-34 (He et al., 2016)	73.3	91.4	8.0ms
NASNet-A (Zoph et al., 2018)	74.0	91.3	38.3ms
DARTS (Liu et al., 2018c)	73.1	91.0	-
MnasNet (Tan et al., 2018)	74.0	91.8	6.1ms
Proxyless (GPU)	<b>75.1</b>	<b>92.5</b>	<b>5.1ms</b>

- The specialized model is 1.83x faster than the non-specialized counterpart on mobile.
- The improvement is larger on GPU.

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai et al., ICLR 2019]

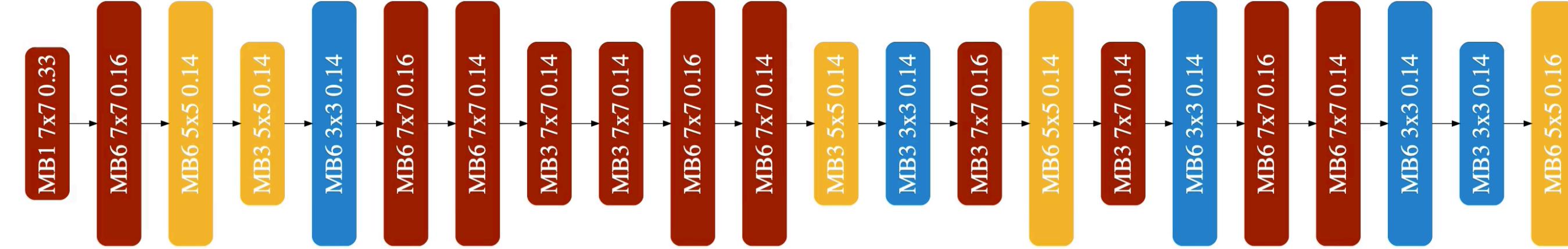
# Specialized Models for Different Hardware



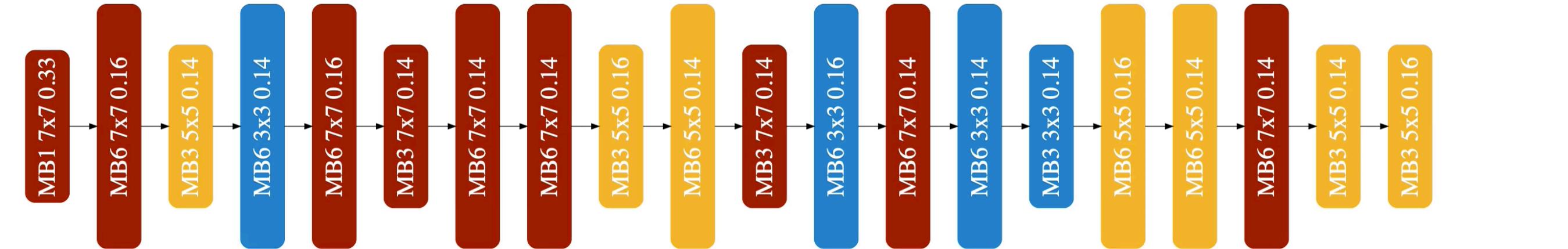
Model	Top-1 (%)	GPU latency	CPU latency	Mobile latency
Proxyless (GPU)	75.1	5.1ms	204.9ms	124ms
Proxyless (CPU)	75.3	7.4ms	138.7ms	116ms
Proxyless (mobile)	74.6	7.2ms	164.1ms	78ms

ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai et al., ICLR 2019]

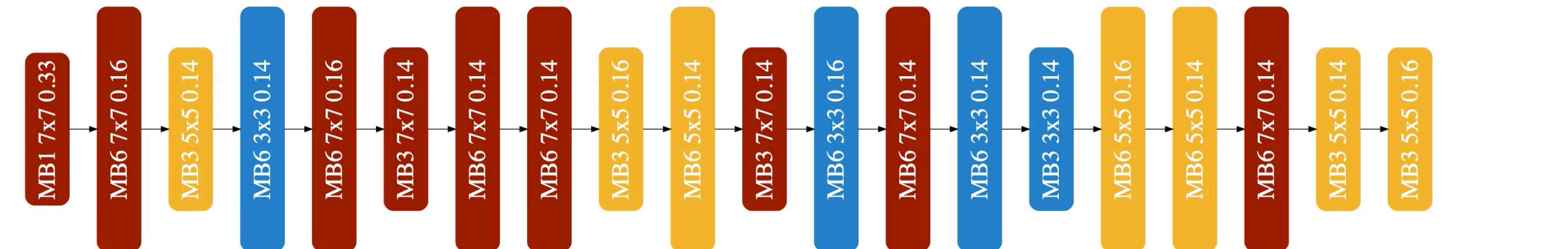
# Specialized Models for Different Hardware



(1) The history of finding efficient Mobile model



(2) The history of finding efficient CPU model



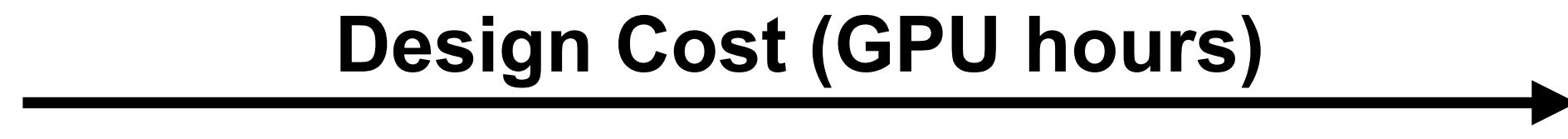
(3) The history of finding efficient GPU model

Epoch-00

# Diverse Platforms and Efficiency Constraints



2023



For search episodes:

40K

For training iterations:

forward-backward(); **Expensive**

If good\_model: **break**;

For post-search training iterations:

forward-backward(); **Expensive**

Mnasnet: Platform-Aware Neural Architecture Search for Mobile [Tang et al., CVPR 2019]

# Diverse Platforms and Efficiency Constraints

## Diverse Hardware Platforms



For devices:

For search episodes:

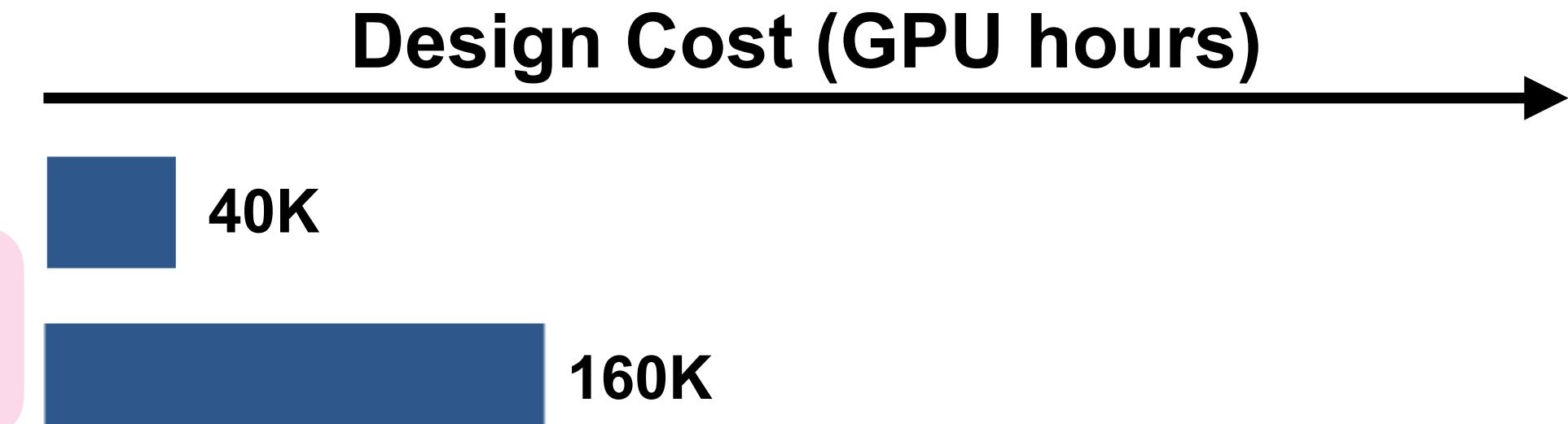
For training iterations:

forward-backward(); **Expensive!**

If good\_model: **break**;

For post-search training iterations:

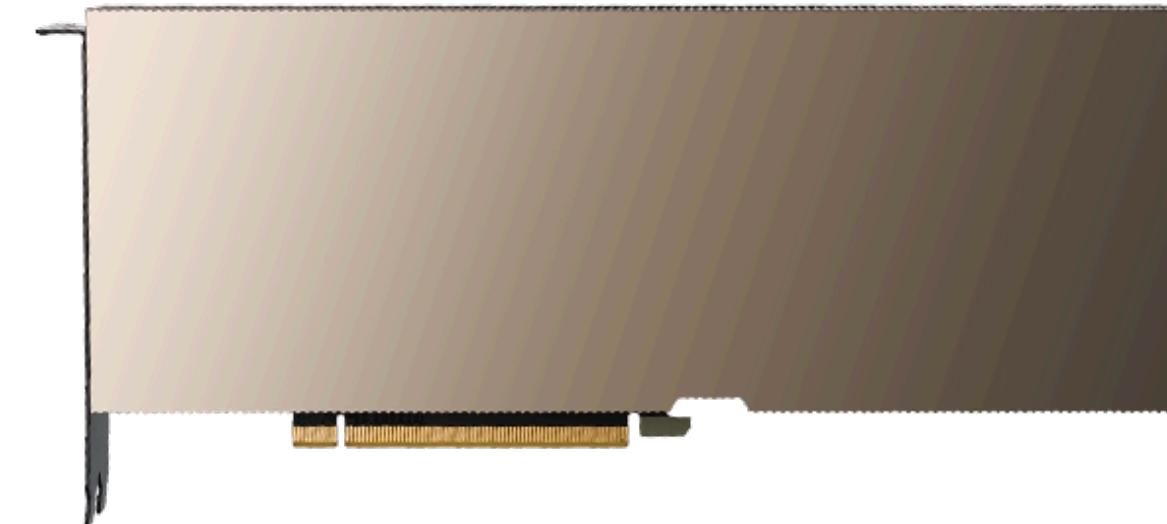
forward-backward(); **Expensive!**



Mnasnet: Platform-Aware Neural Architecture Search for Mobile [Tang et al., CVPR 2019]

# Diverse Platforms and Efficiency Constraints

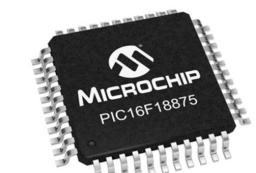
## Diverse Hardware Platforms



Cloud AI ( $10^{12}$  FLOPS)



Mobile AI ( $10^9$  FLOPS)



...

Tiny AI ( $10^6$  FLOPS)

For many devices:

For search episodes:

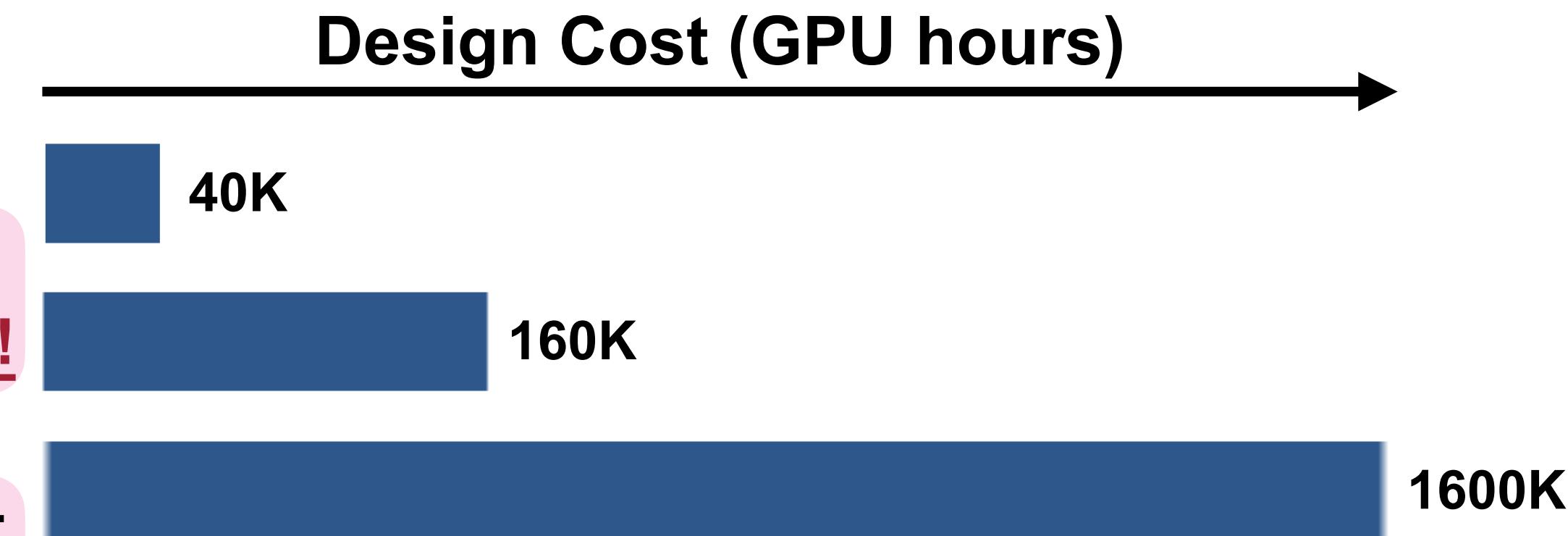
For training iterations:

forward-backward(); **Expensive!!**

If good\_model: **break**;

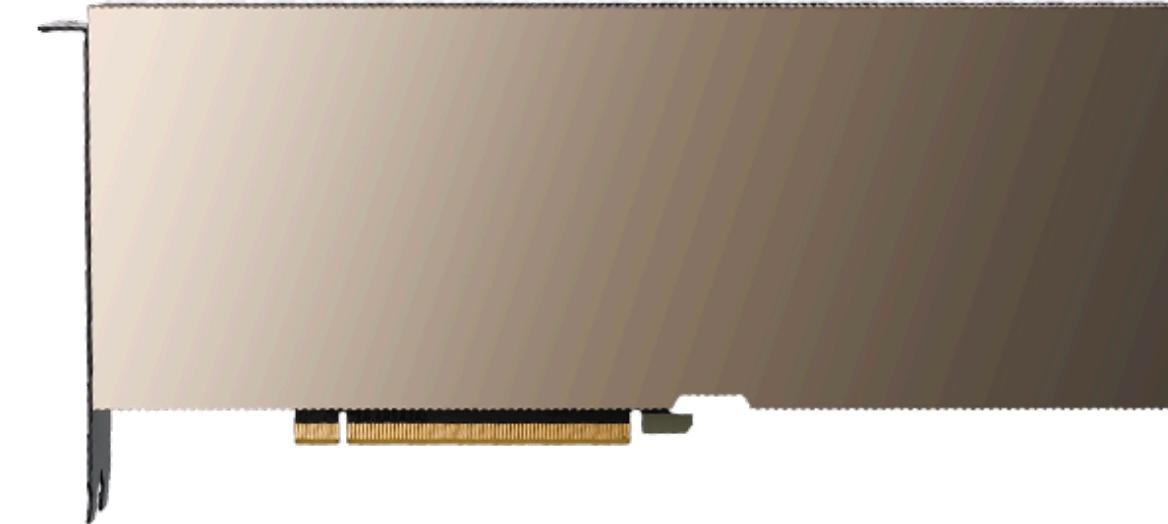
For post-search training iterations:

forward-backward(); **Expensive!!**

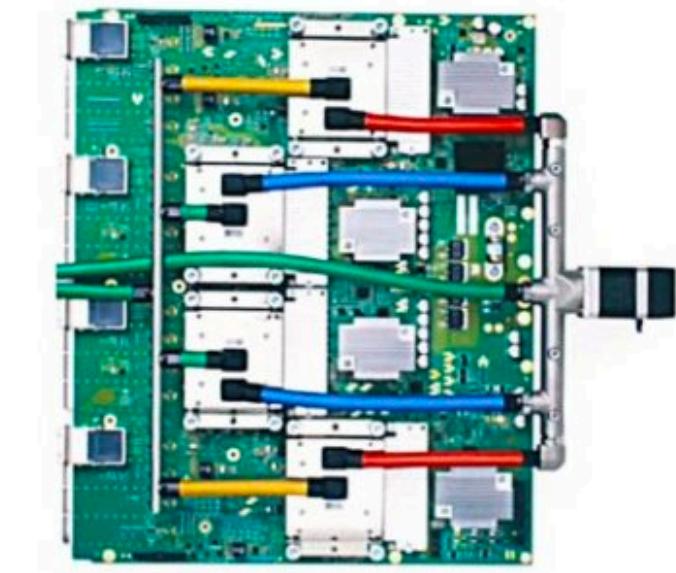


Mnasnet: Platform-Aware Neural Architecture Search for Mobile [Tang et al., CVPR 2019]

# Improve the productivity of designing models



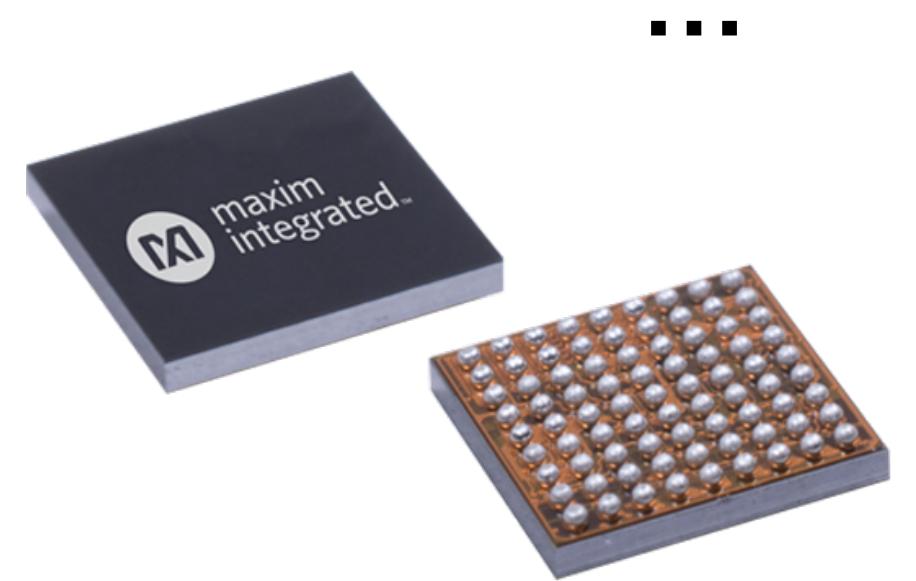
**GPU**



**TPU**



**DSP**



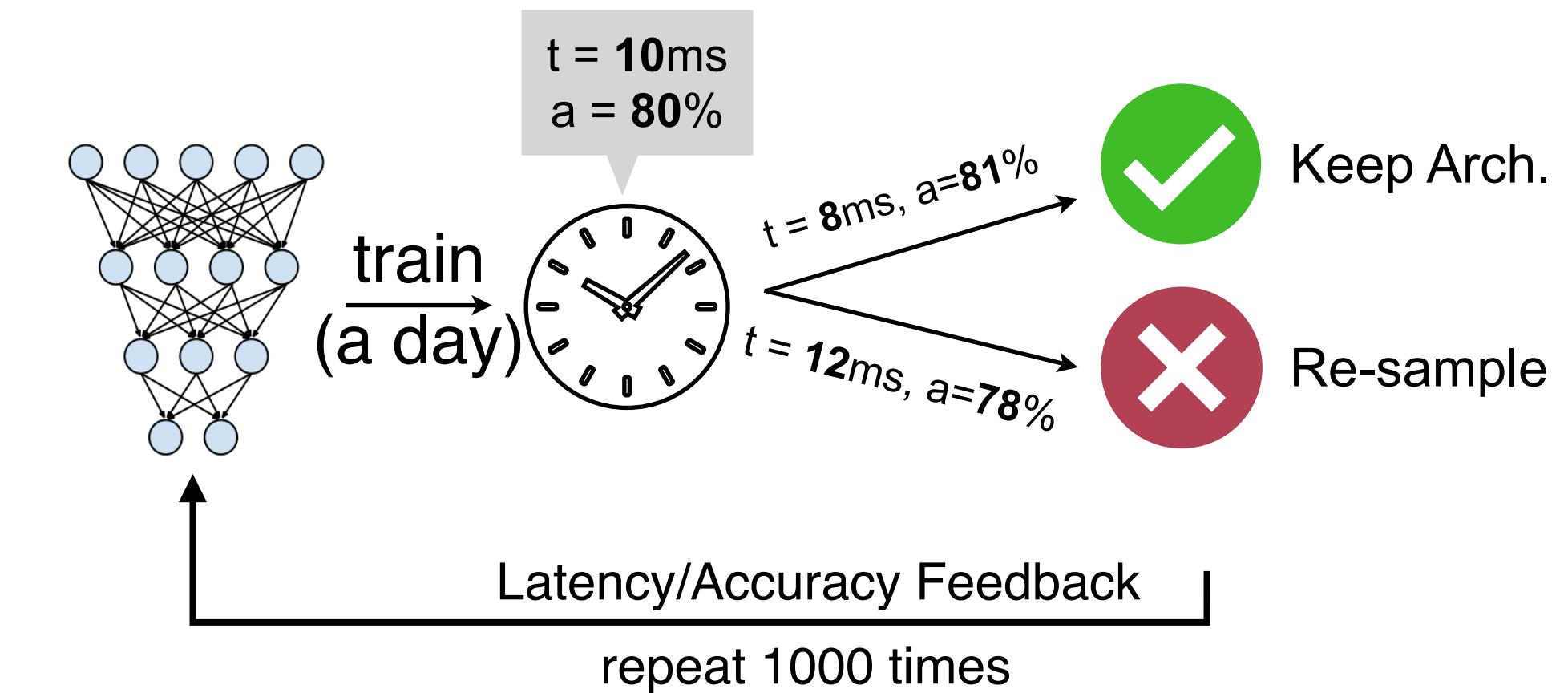
**Accelerator**

Existing models are heavily optimized for GPUs  
But doesn't fit well on other platforms

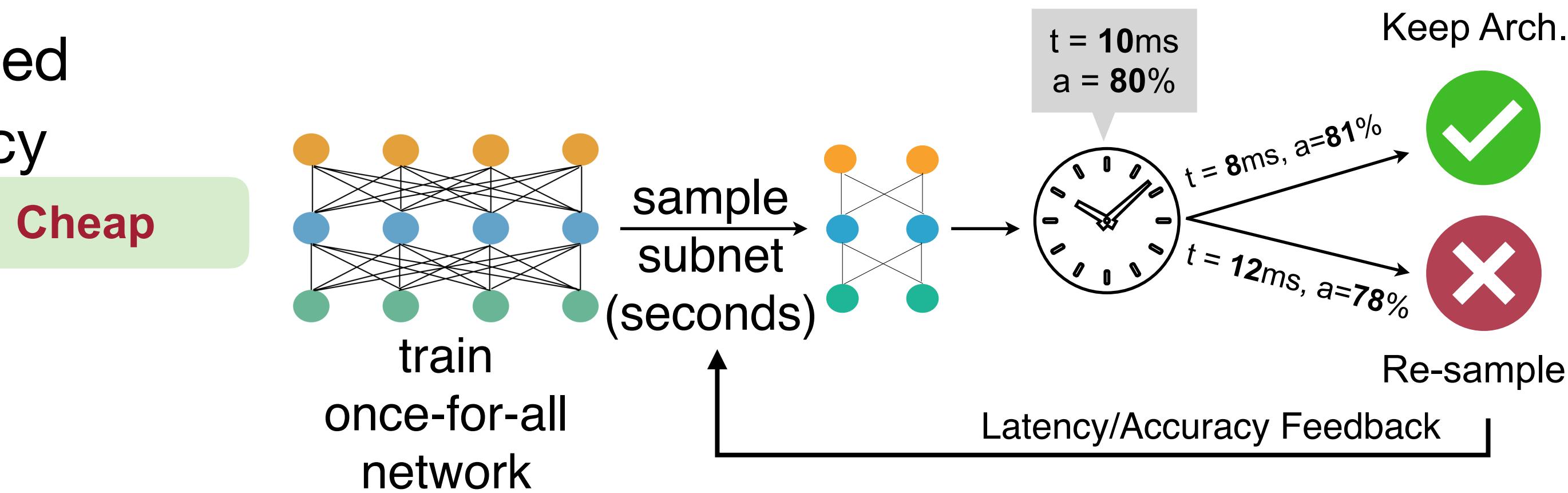
# Hardware-aware NAS with Once-for-All Network

Rather than training each network from scratch,  
can we train multiple models at the same time?

1. Train a network, get accuracy and latency **Expensive**
2. Repeat, select best



1. Train a once-for-all network, sparsely activated
2. Select sub-network, get accuracy and latency
3. Repeat 2, select best



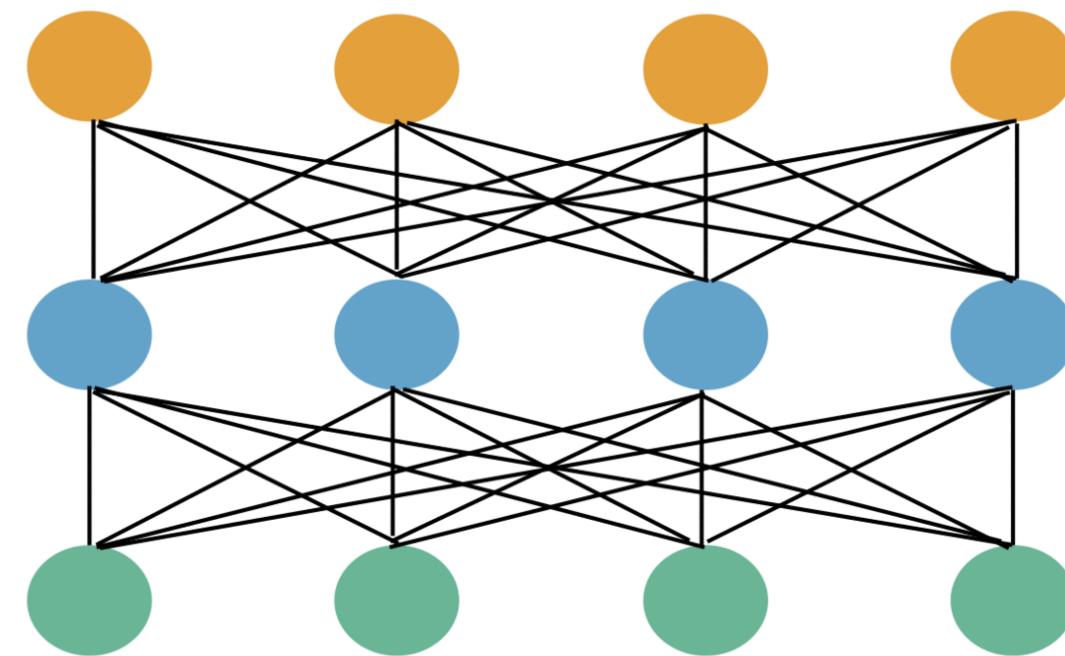
Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

**Train once, get many**

**Reduce the design cost**

**Fit diverse hardware constraints**



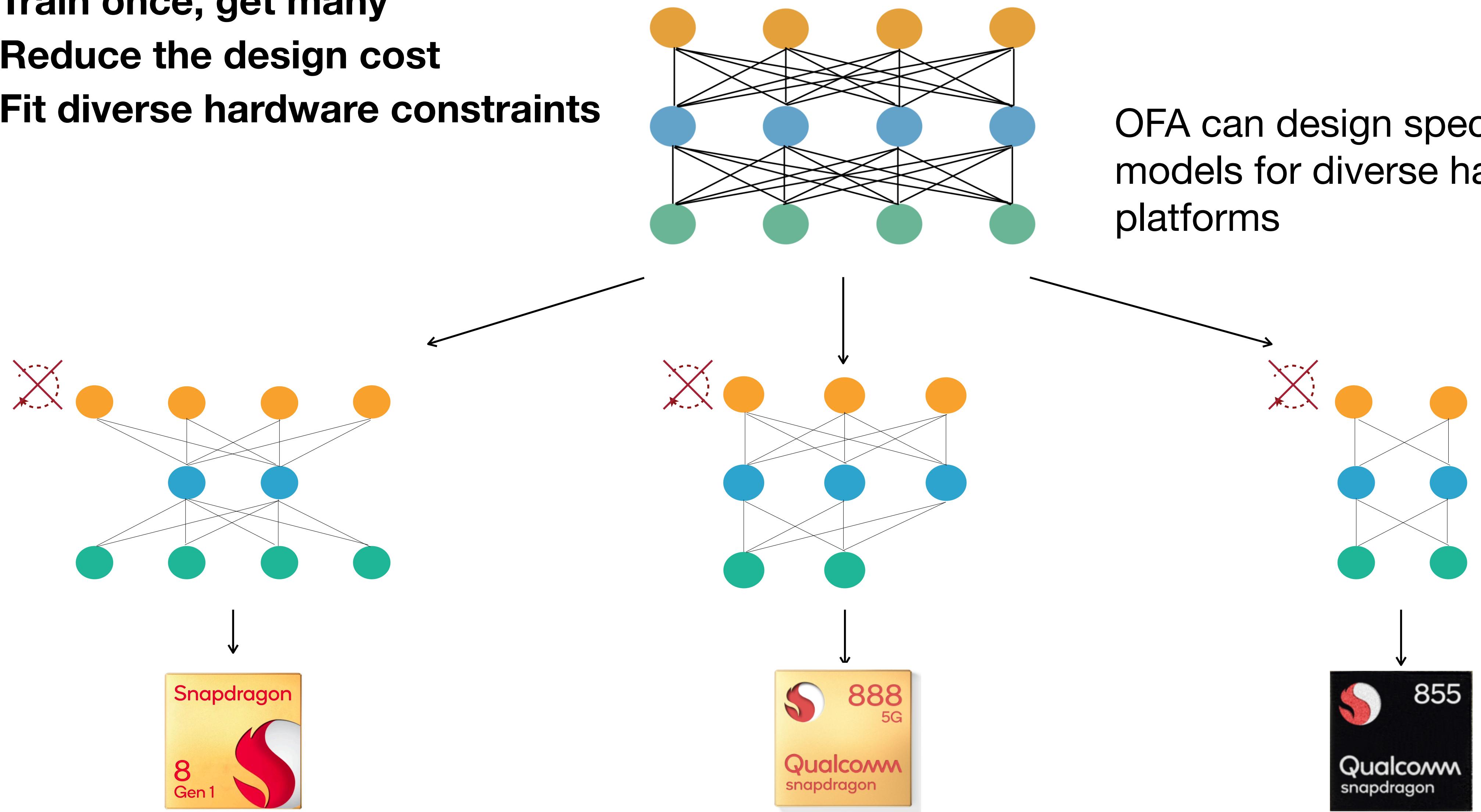
OFA network contains many child networks that are sparsely activated

# Once-for-All Network

Train once, get many

Reduce the design cost

Fit diverse hardware constraints



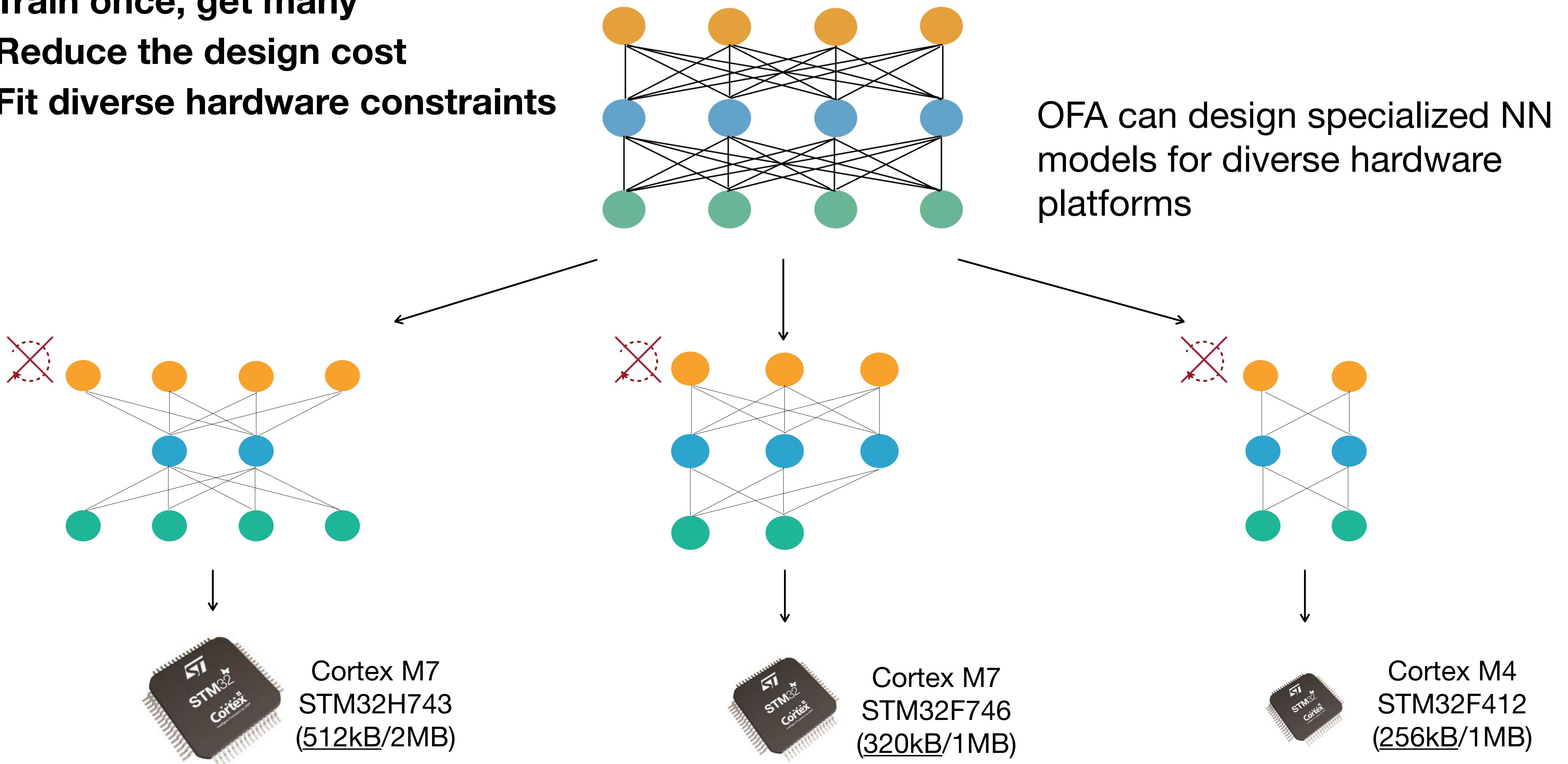
Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

Train once, get many

Reduce the design cost

Fit diverse hardware constraints



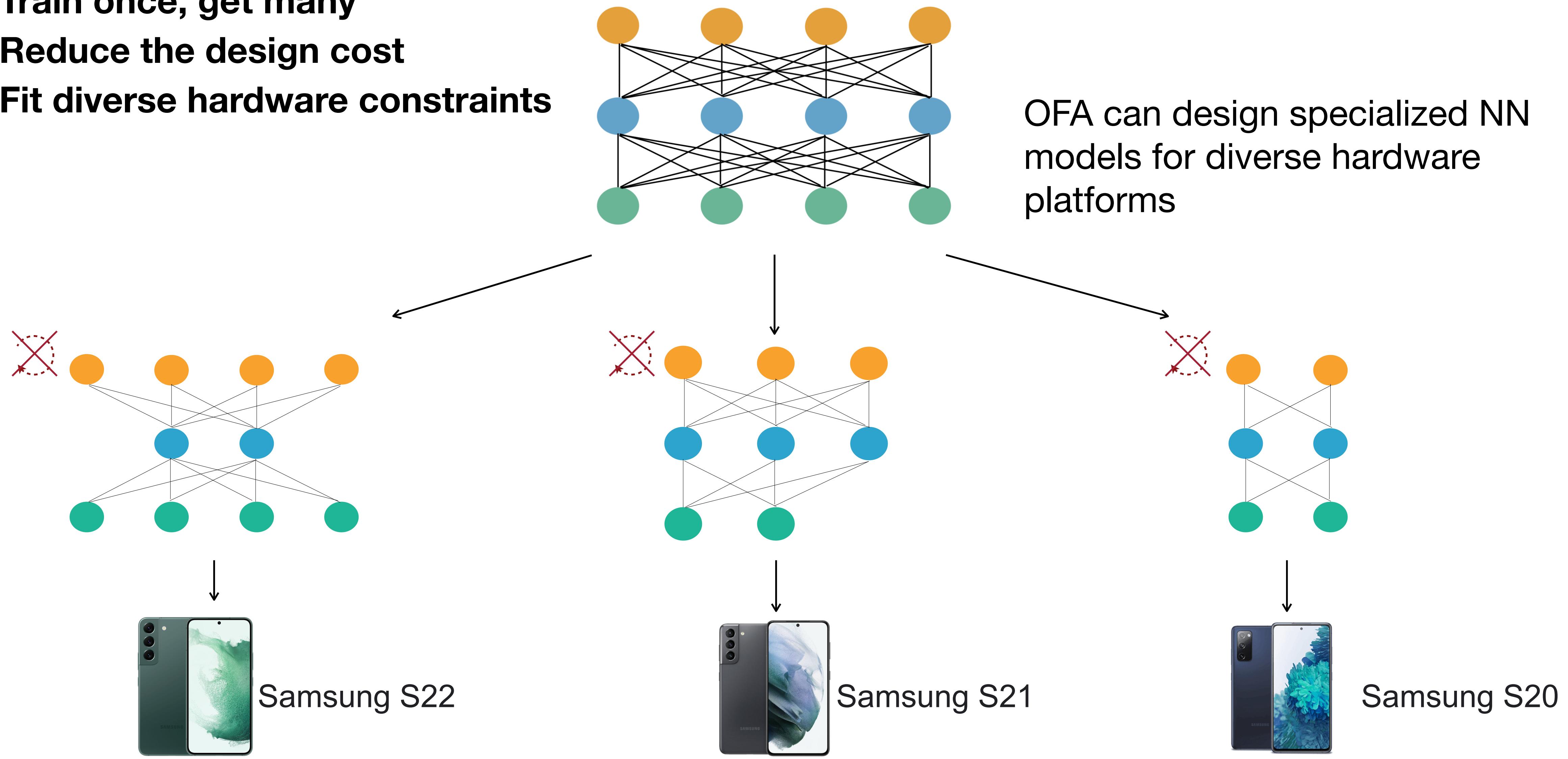
Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

**Train once, get many**

**Reduce the design cost**

**Fit diverse hardware constraints**



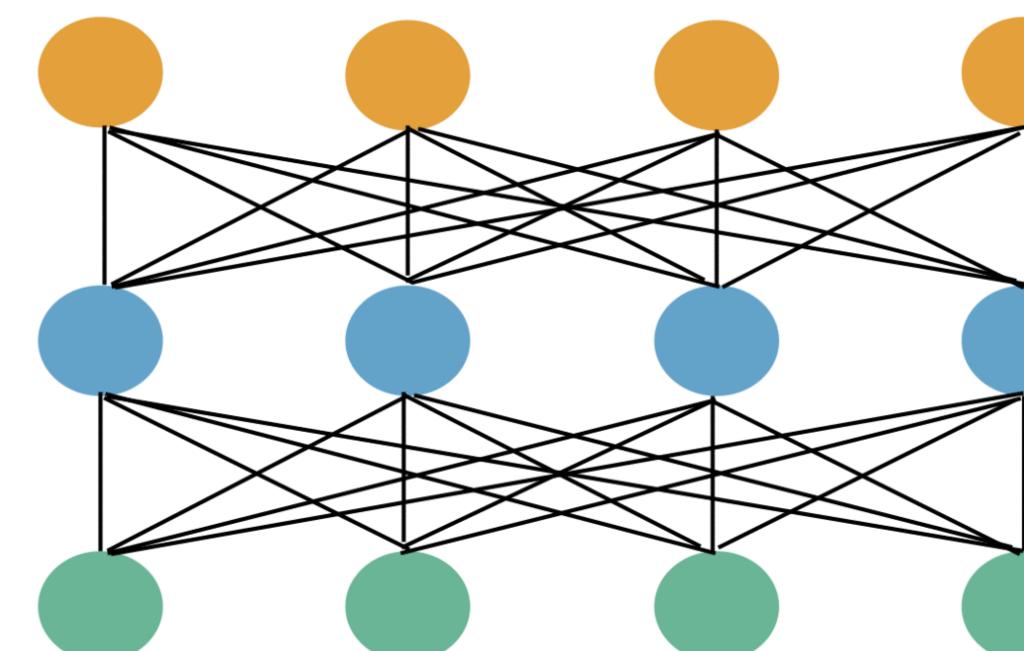
Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

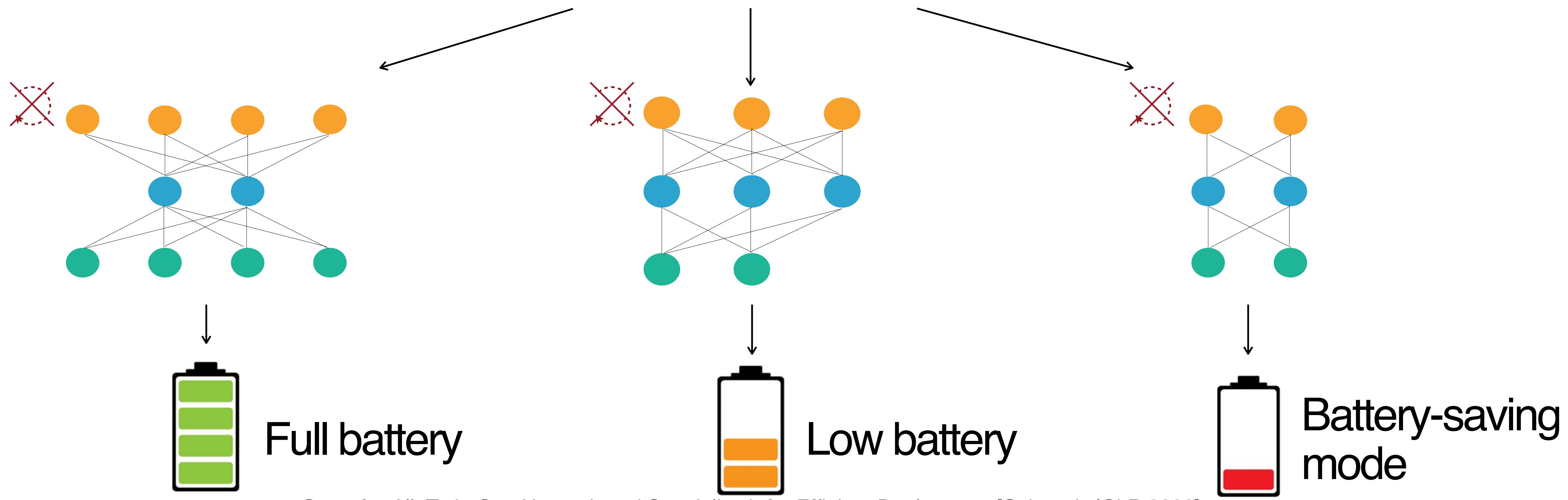
**Train once, get many**

**Reduce the design cost**

**Fit diverse hardware constraints**



OFA can design specialized NN models for diverse hardware platforms

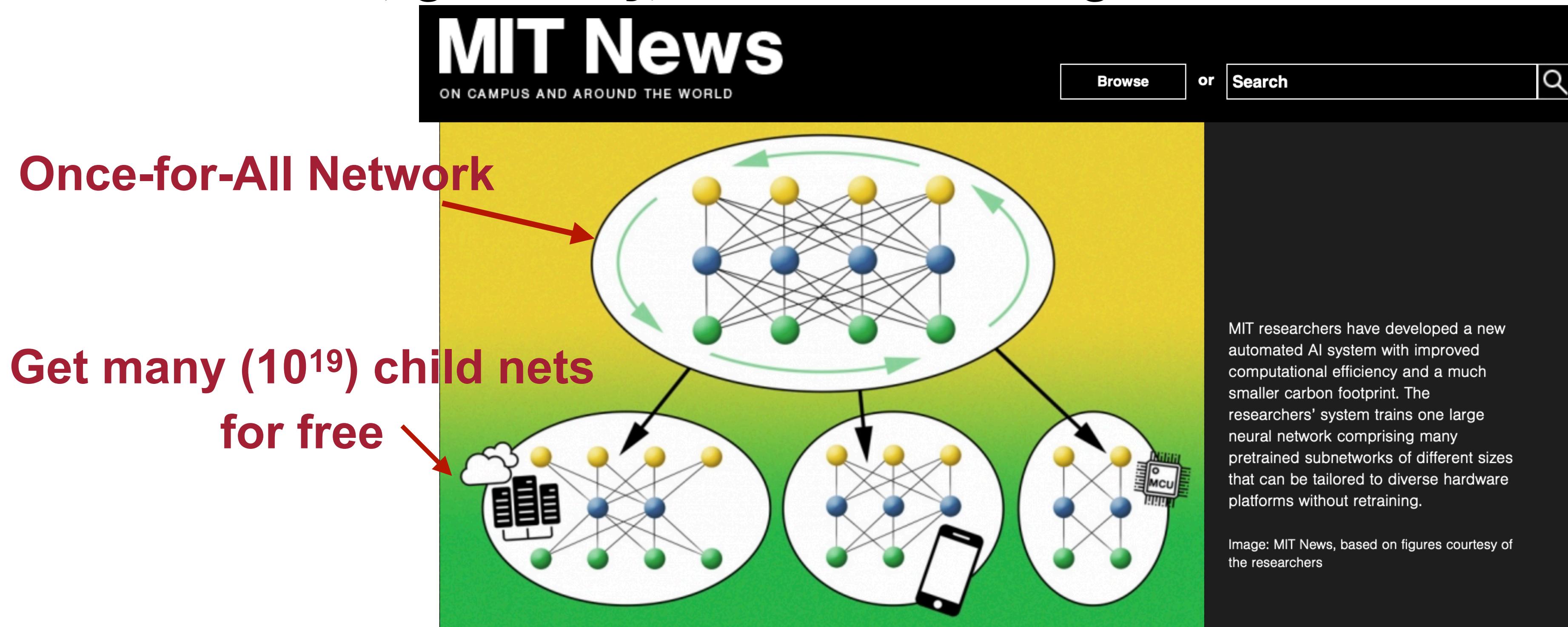


Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once for All Network: Train $10^{19}$ networks at the same time

Design new efficient models rather than compressing existing models

Train once, get many, reduce the design cost



OFA network contains many child networks that are sparsely activated

Child networks share the weights with the once-for-All network, trained jointly, amortize the training cost.

OFA can design specialized NN models for diverse hardware platforms

Reducing the carbon footprint of artificial intelligence  
MIT system cuts the energy required for training and running neural networks.

Rob Matheson | MIT News Office  
April 23, 2020

▼ Press Inquiries

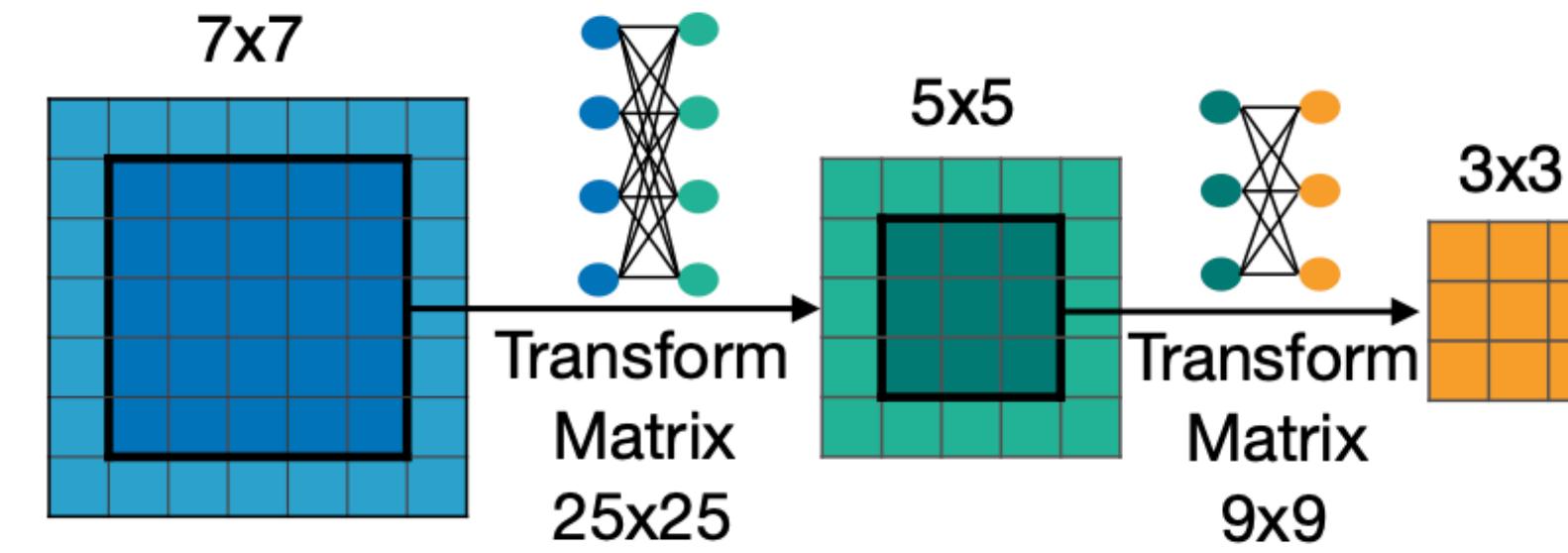
RELATED

Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

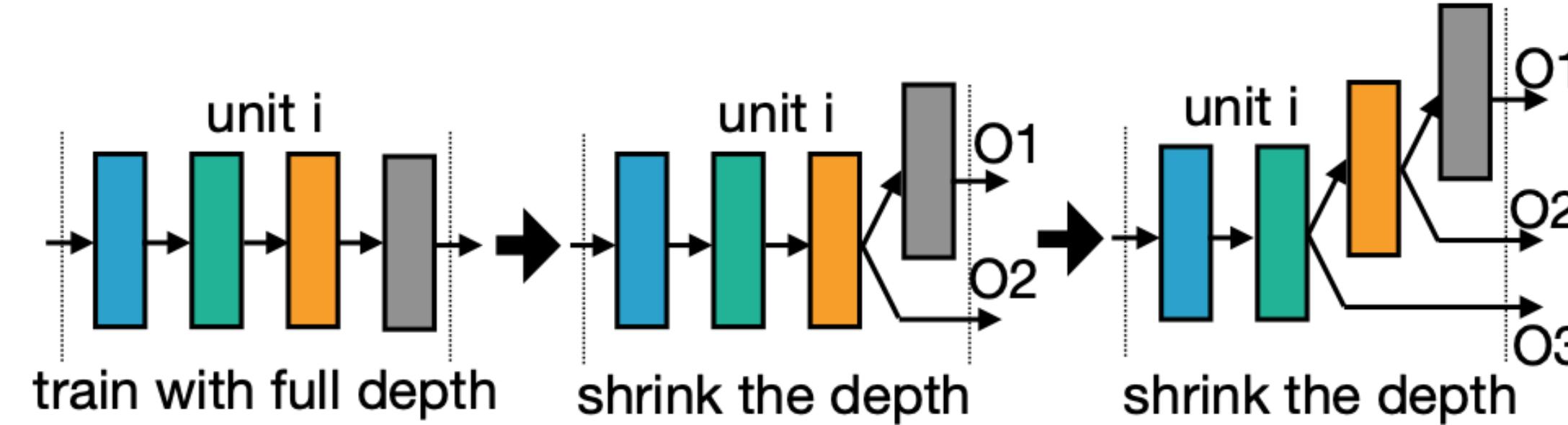
# Once-for-All Network

progressively prune the kernel size, depth, resolution

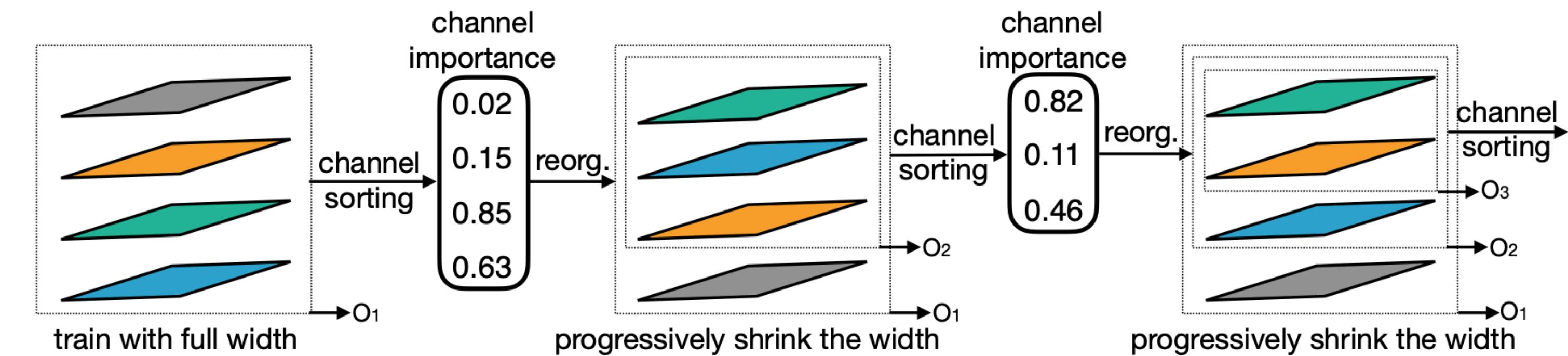
progressive shrinking:  
kernel size



progressive shrinking:  
# layers



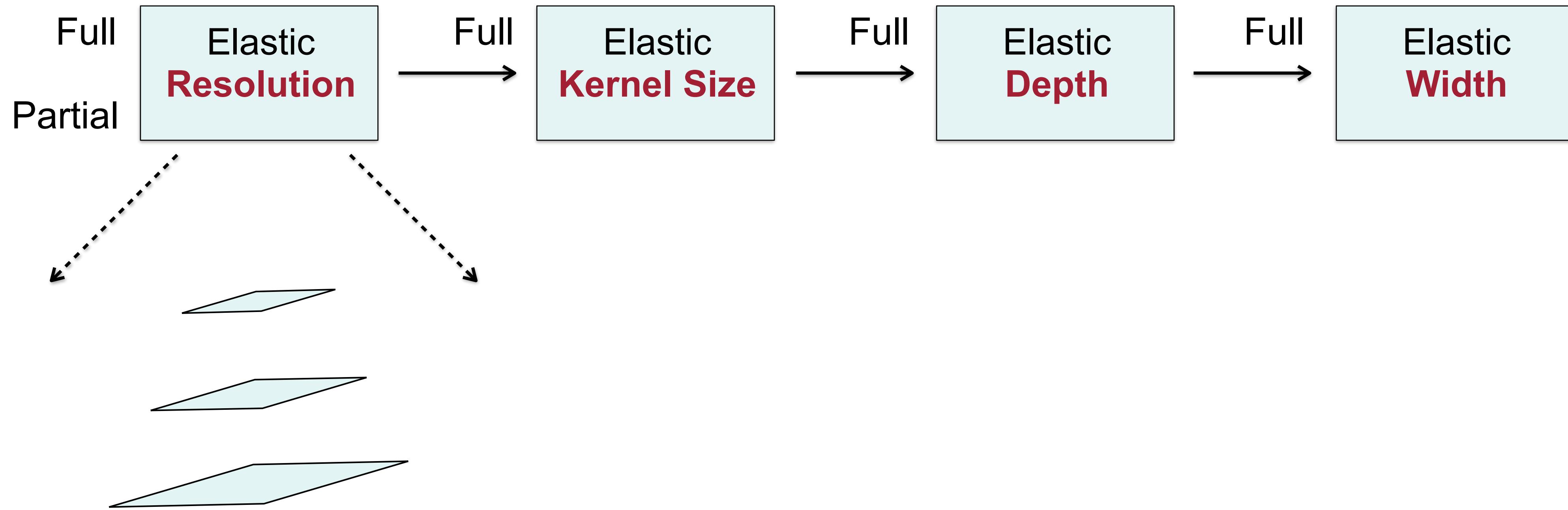
progressive shrinking:  
# channels



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

progressively prune the kernel size, depth, resolution

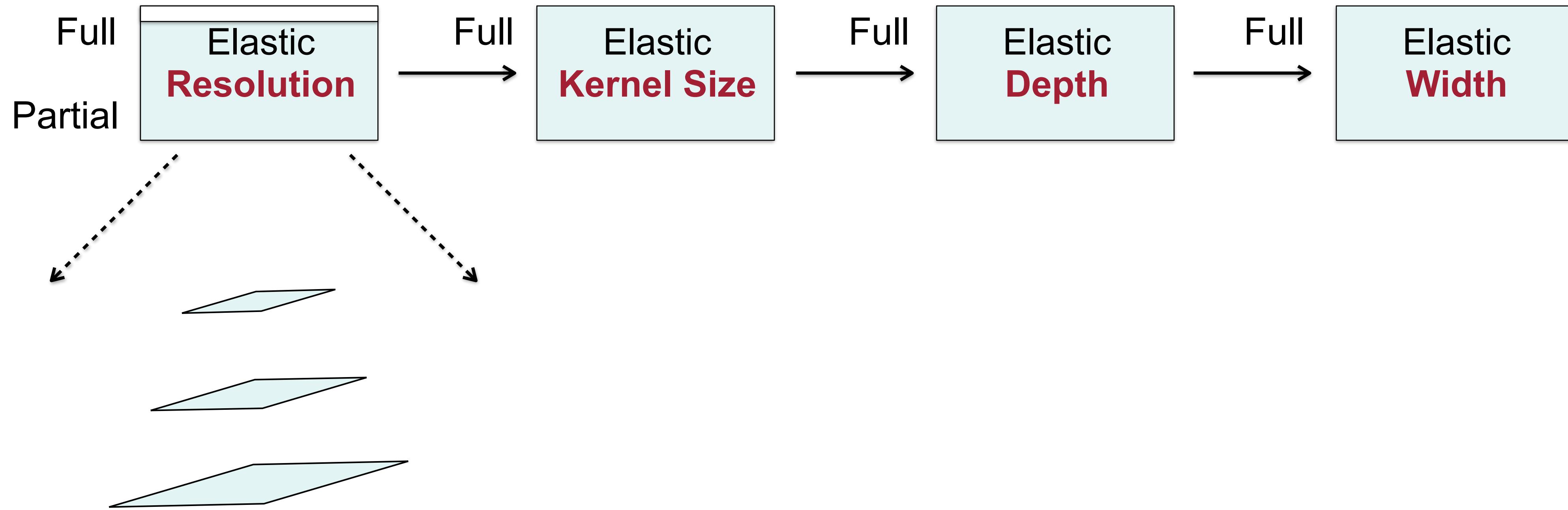


Randomly sample input image  
size for each batch

Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

progressively prune the kernel size, depth, resolution

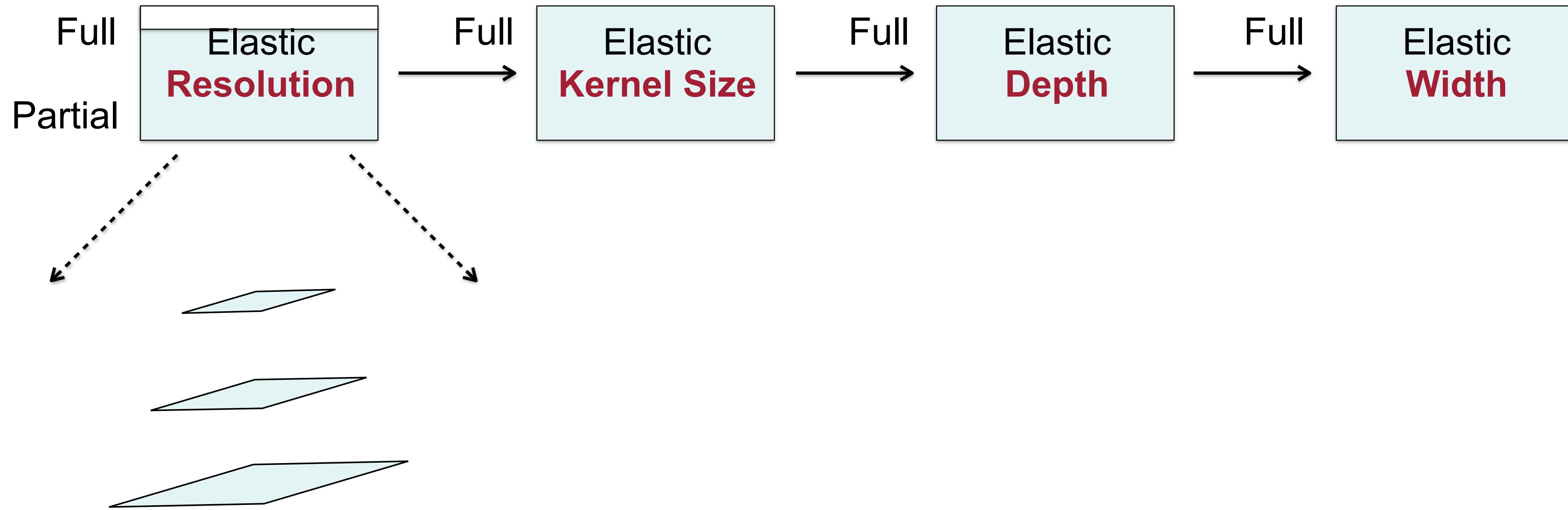


Randomly sample input image  
size for each batch

Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

progressively prune the kernel size, depth, resolution

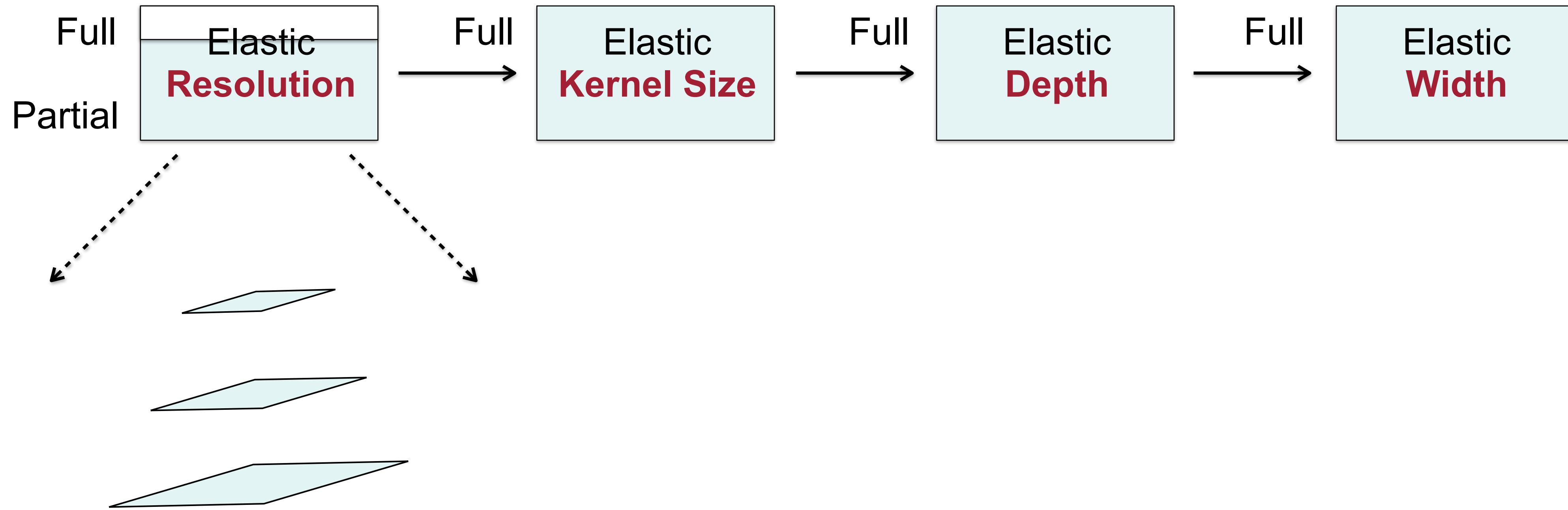


Randomly sample input image  
size for each batch

Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

progressively prune the kernel size, depth, resolution

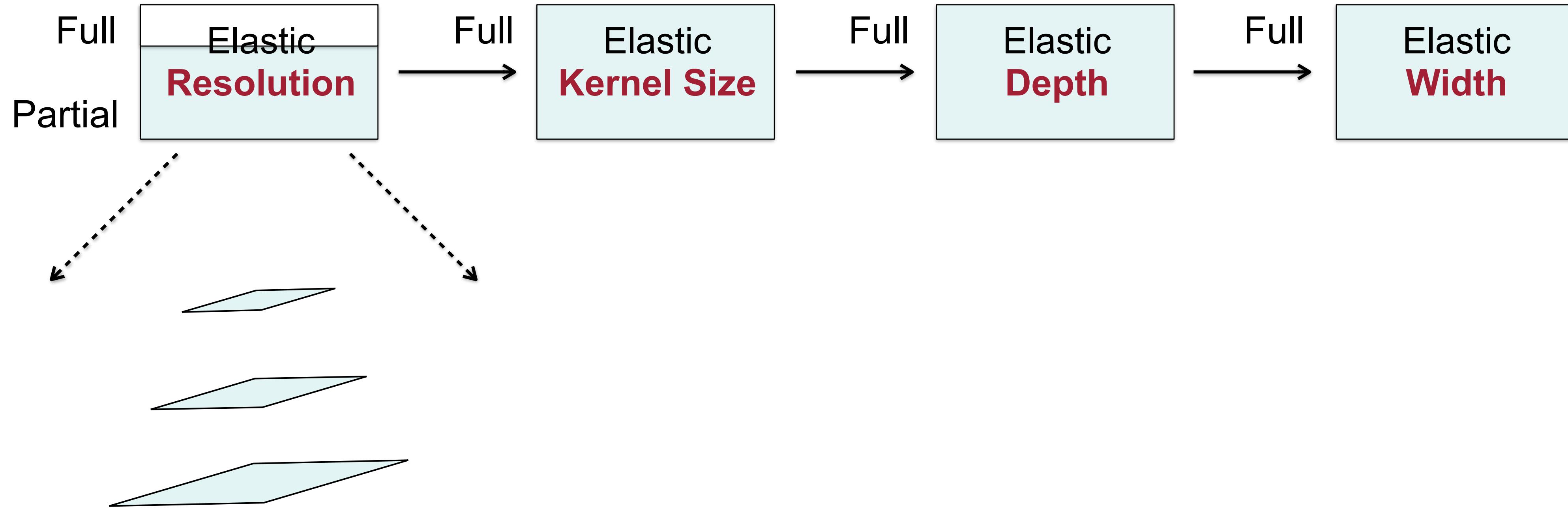


Randomly sample input image  
size for each batch

Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

progressively prune the kernel size, depth, resolution

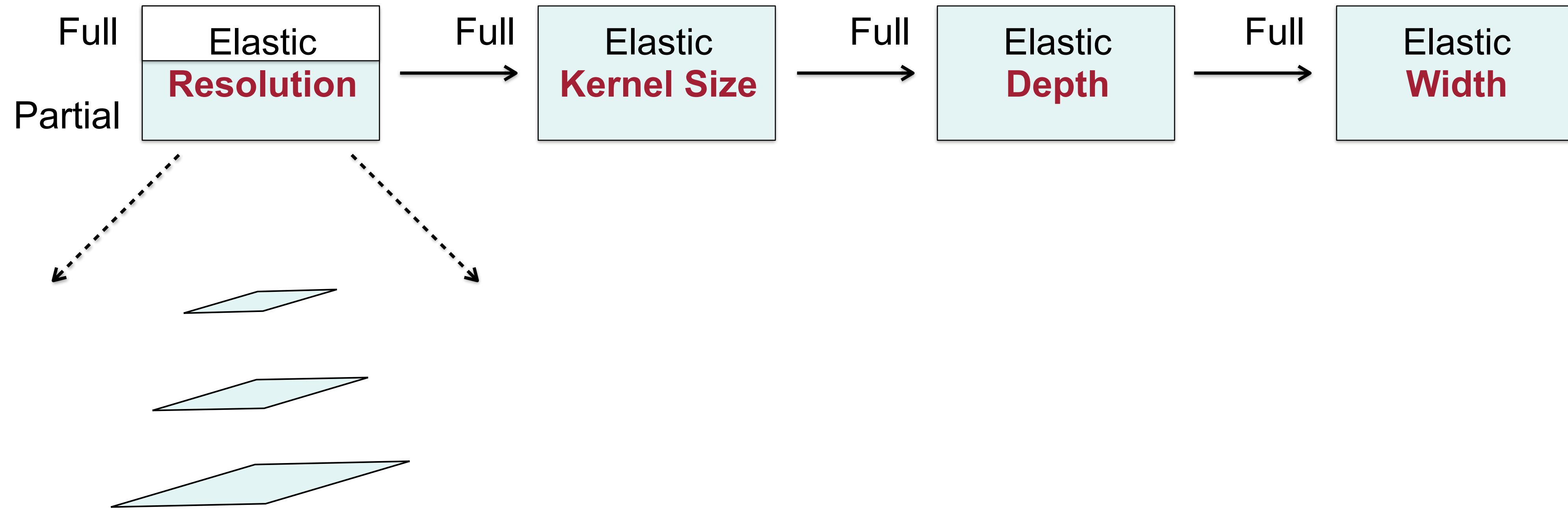


Randomly sample input image  
size for each batch

Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

progressively prune the kernel size, depth, resolution

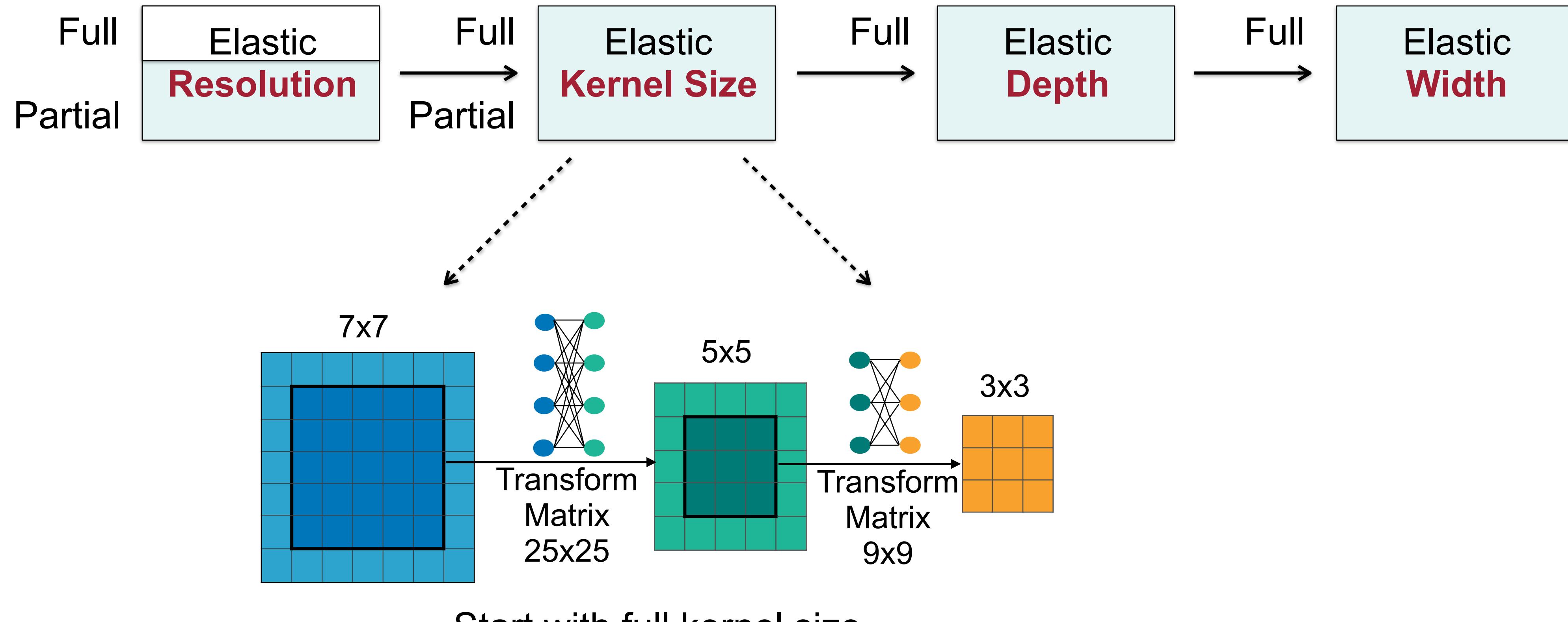


Randomly sample input image  
size for each batch

Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

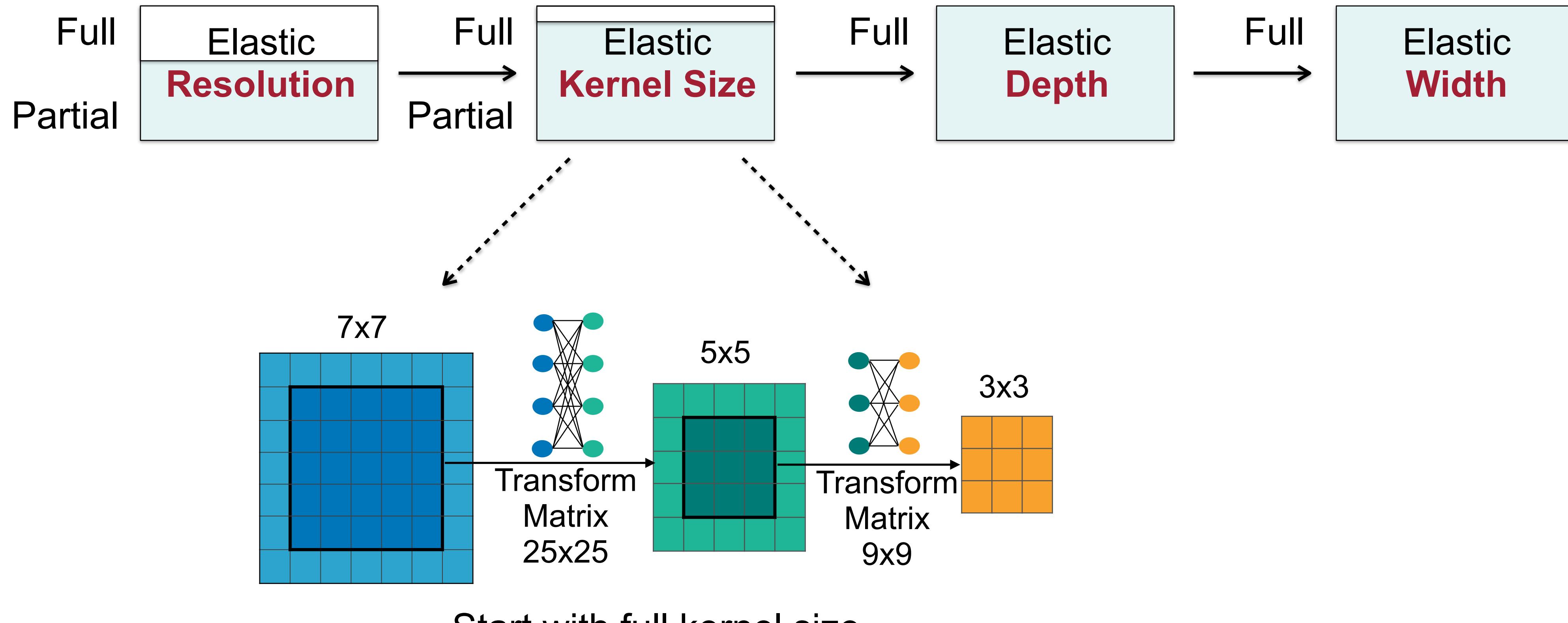
progressively prune the kernel size, depth, resolution



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

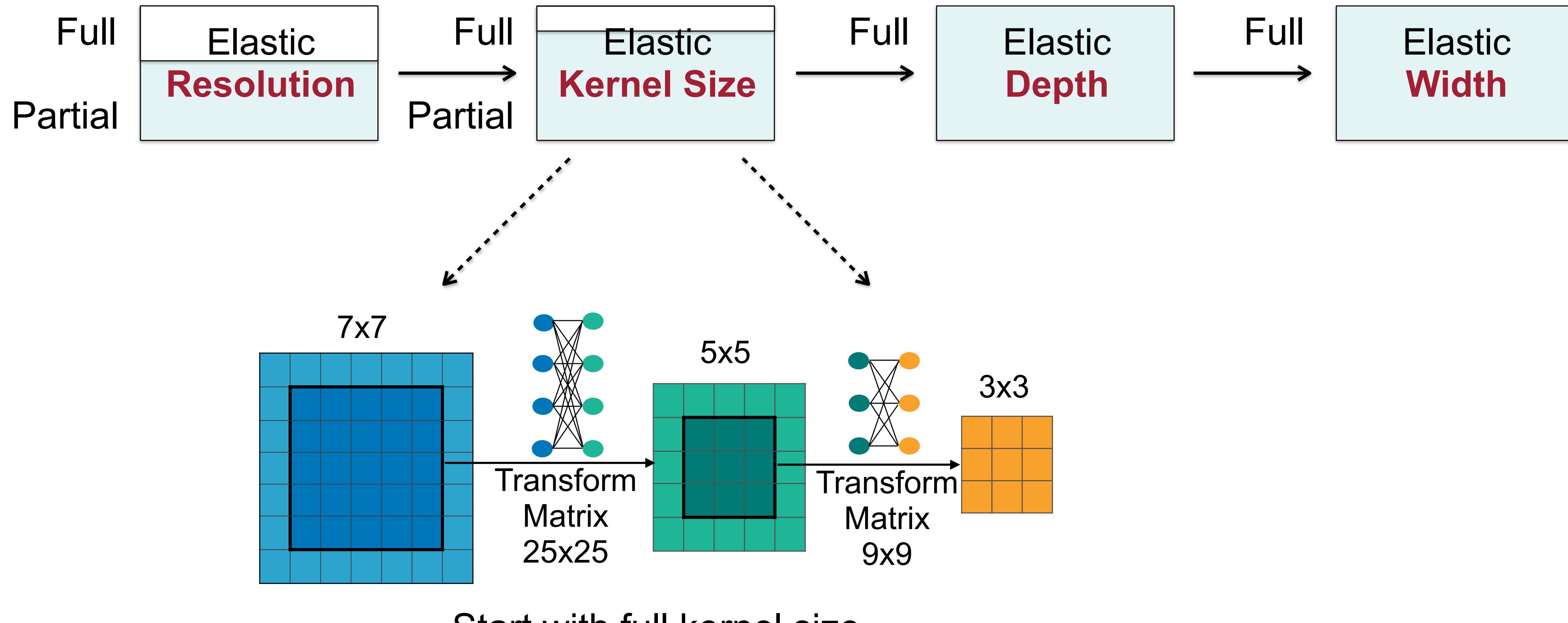
progressively prune the kernel size, depth, resolution



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

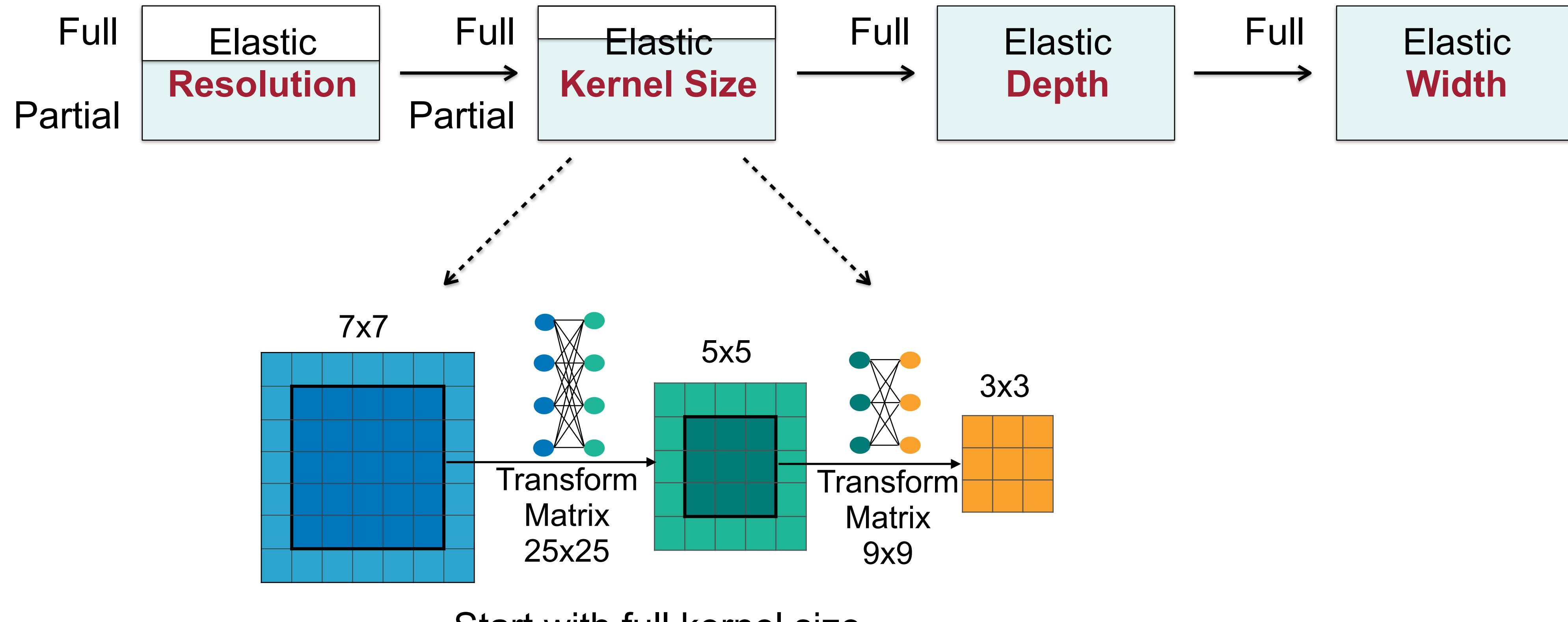
progressively prune the kernel size, depth, resolution



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

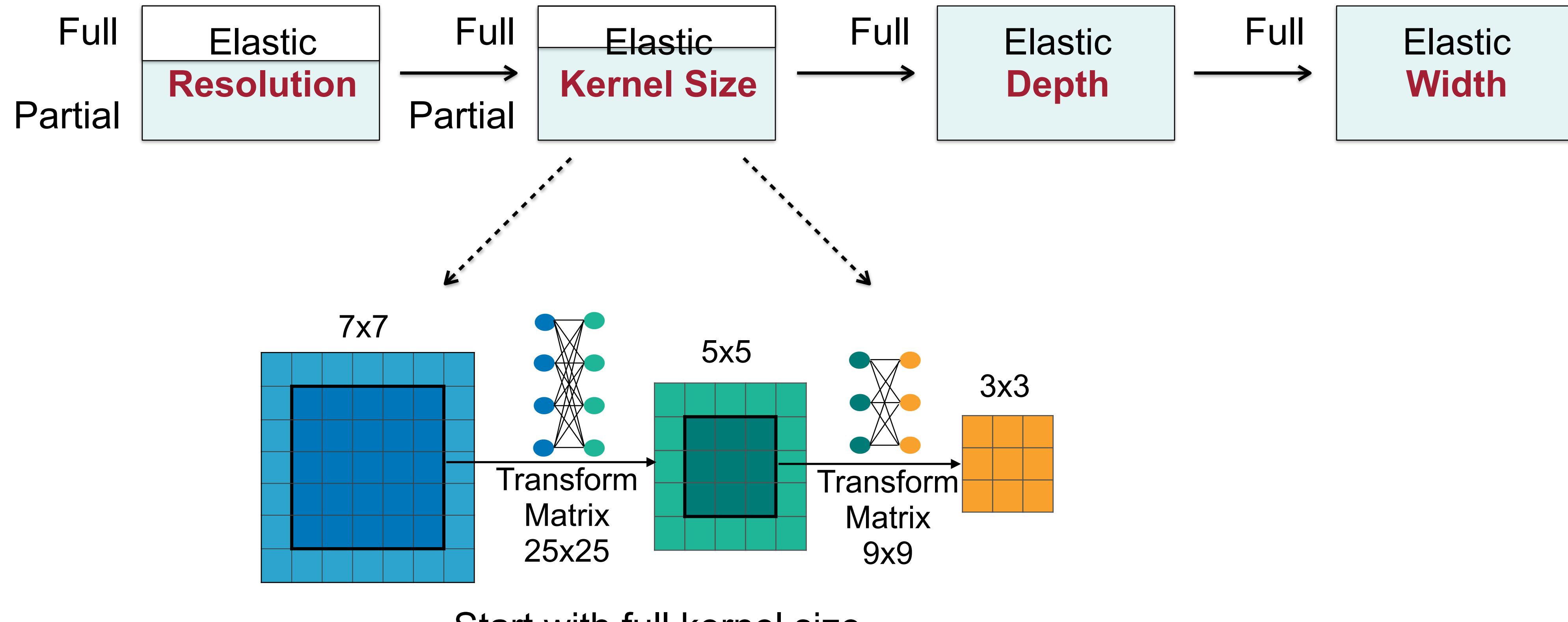
progressively prune the kernel size, depth, resolution



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

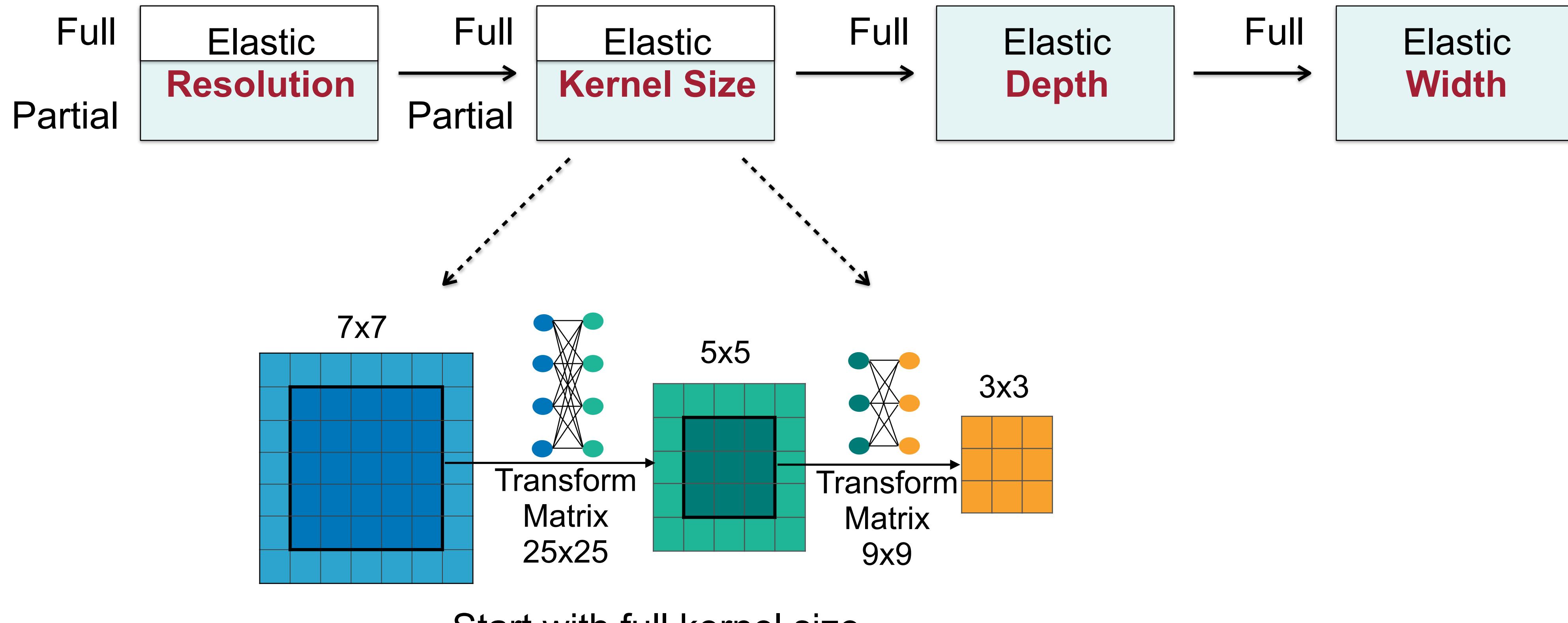
progressively prune the kernel size, depth, resolution



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

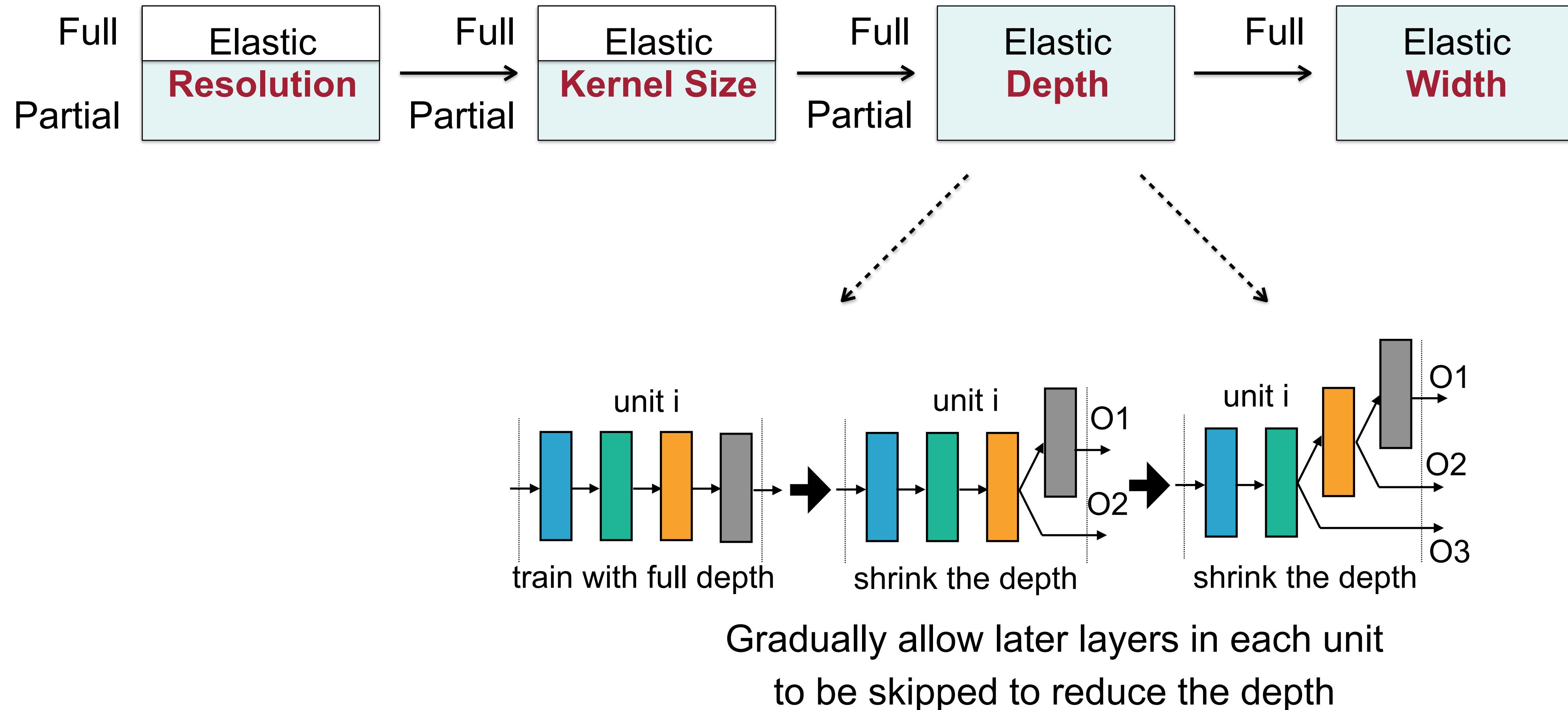
progressively prune the kernel size, depth, resolution



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

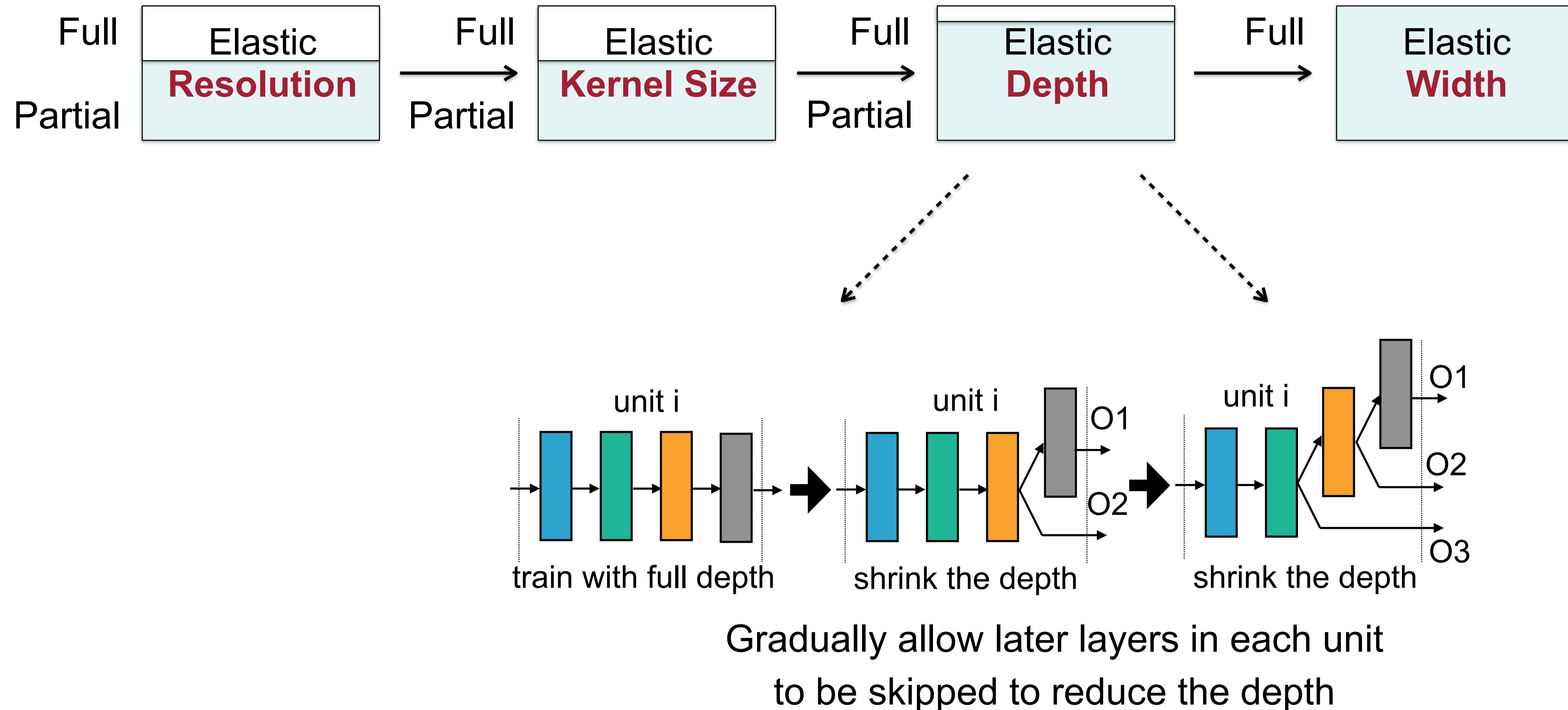
progressively prune the kernel size, depth, resolution



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

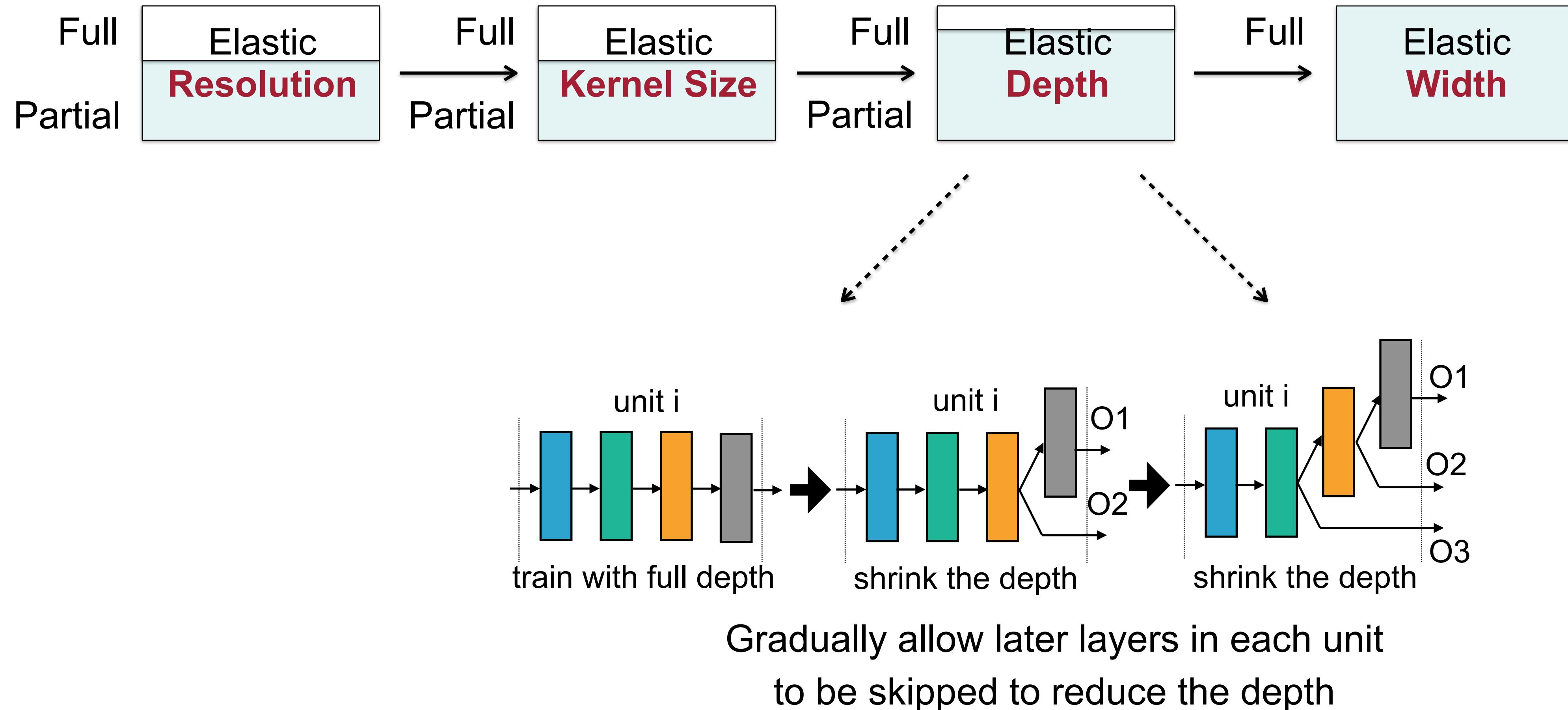
progressively prune the kernel size, depth, resolution



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

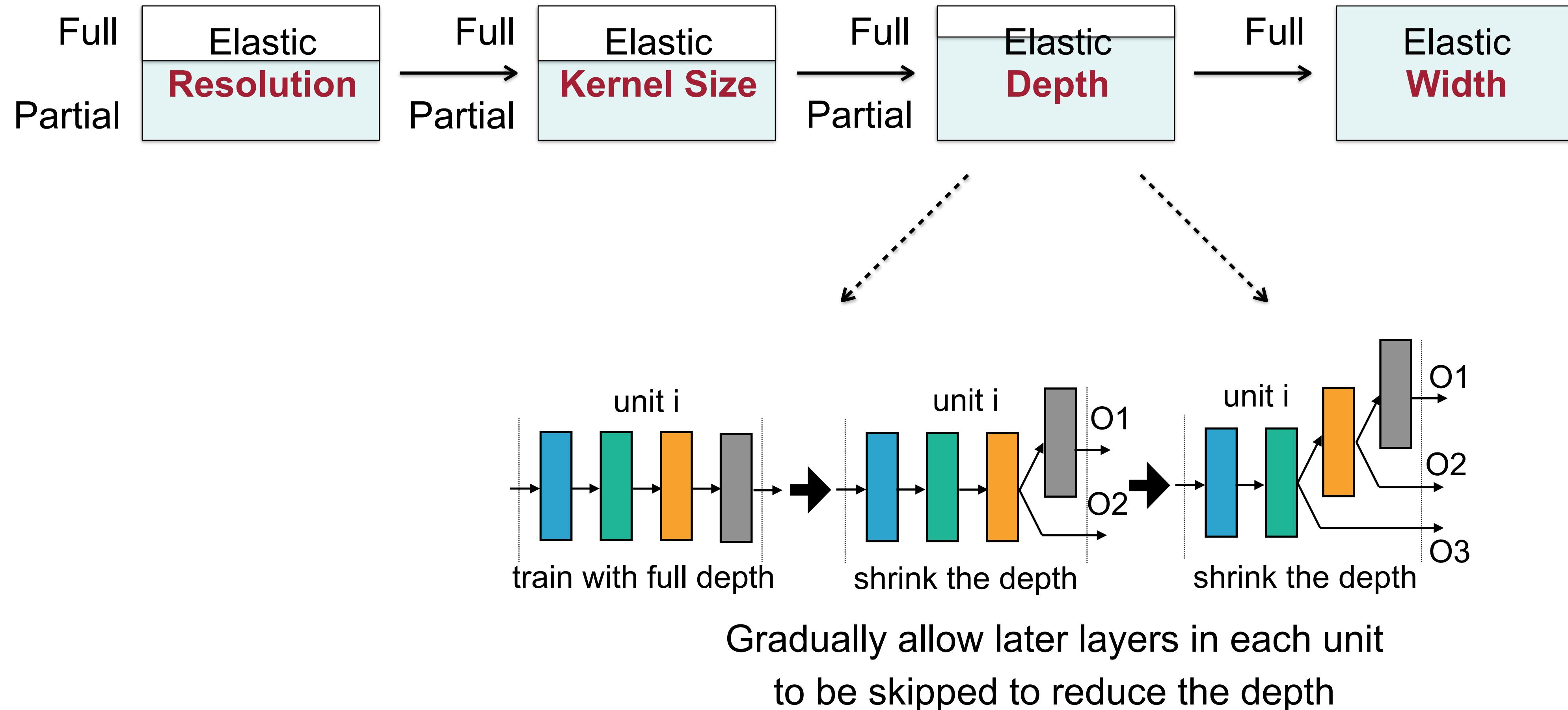
progressively prune the kernel size, depth, resolution



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

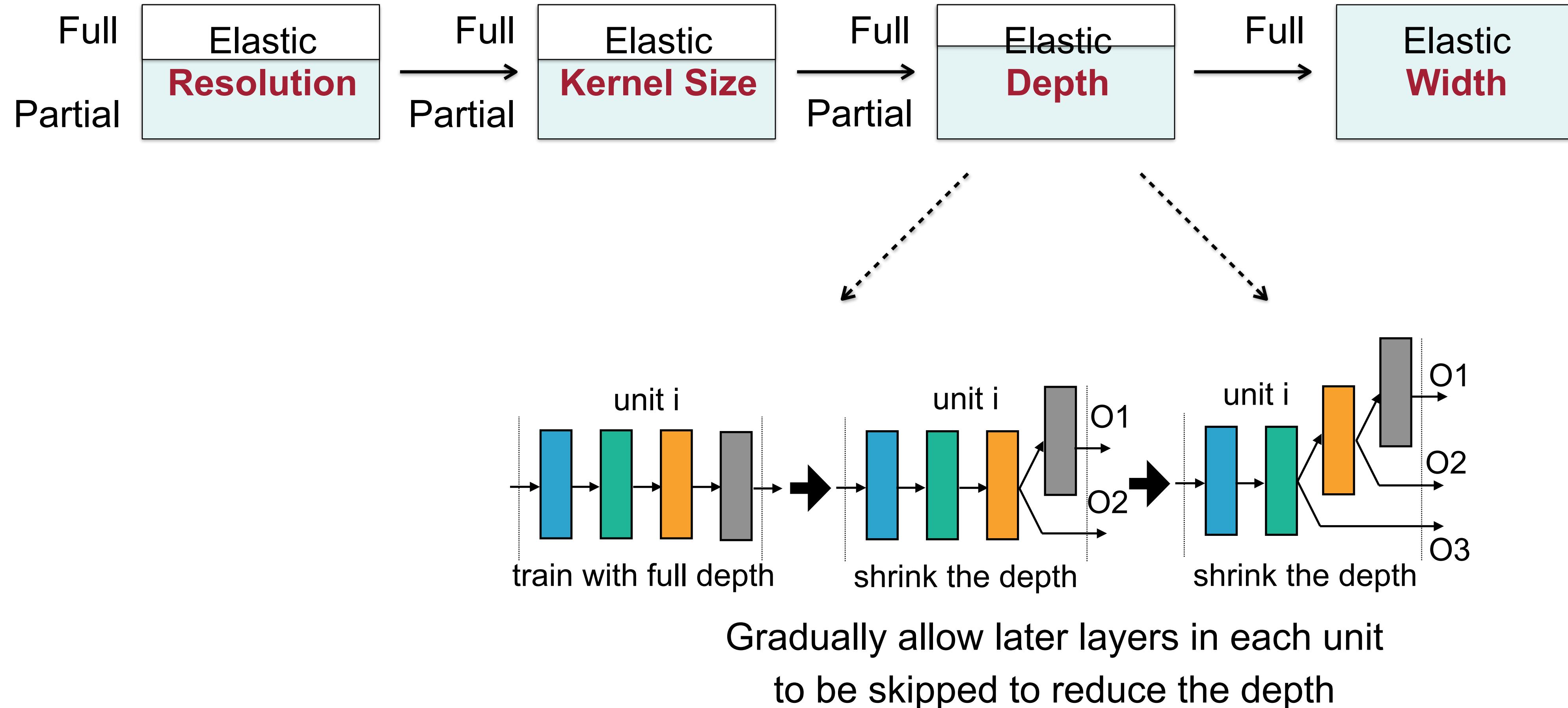
progressively prune the kernel size, depth, resolution



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

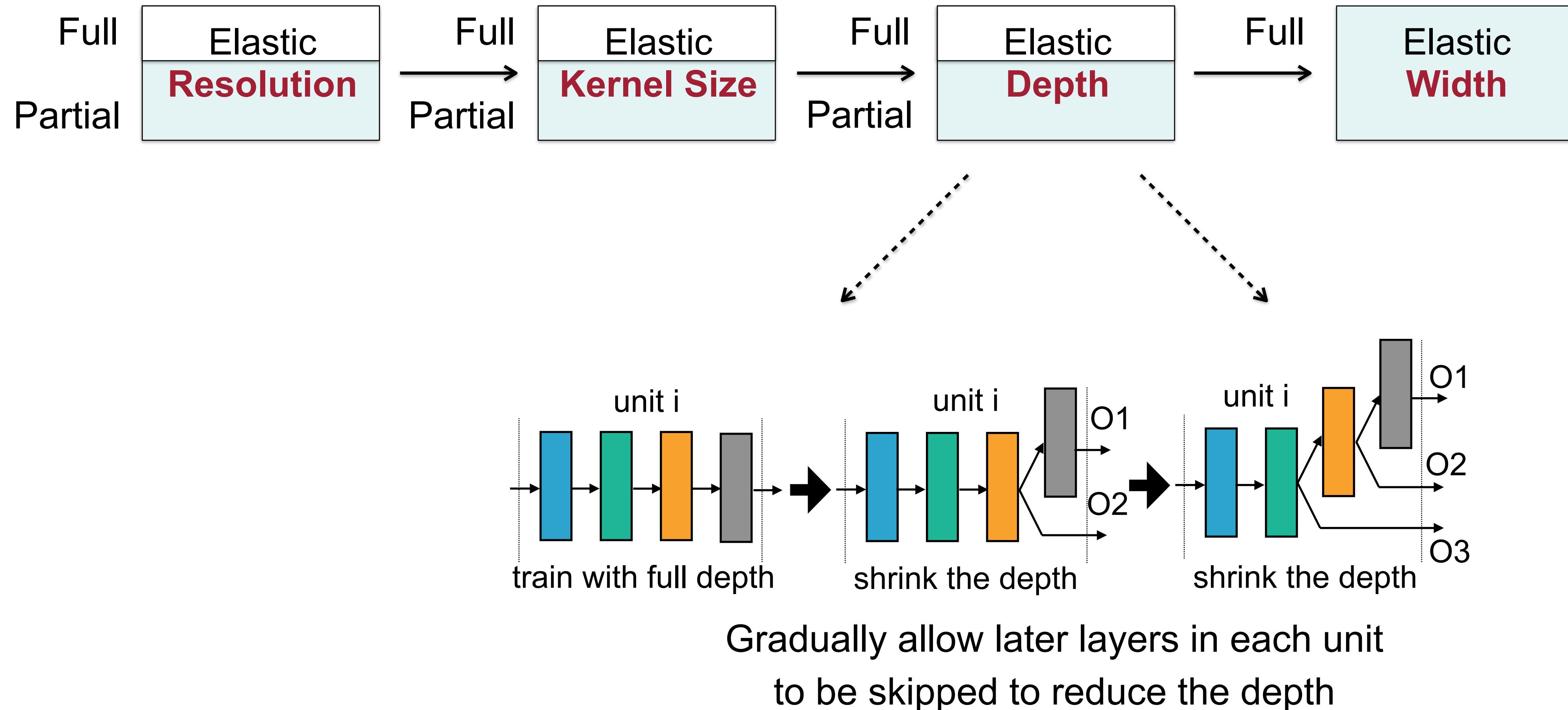
progressively prune the kernel size, depth, resolution



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

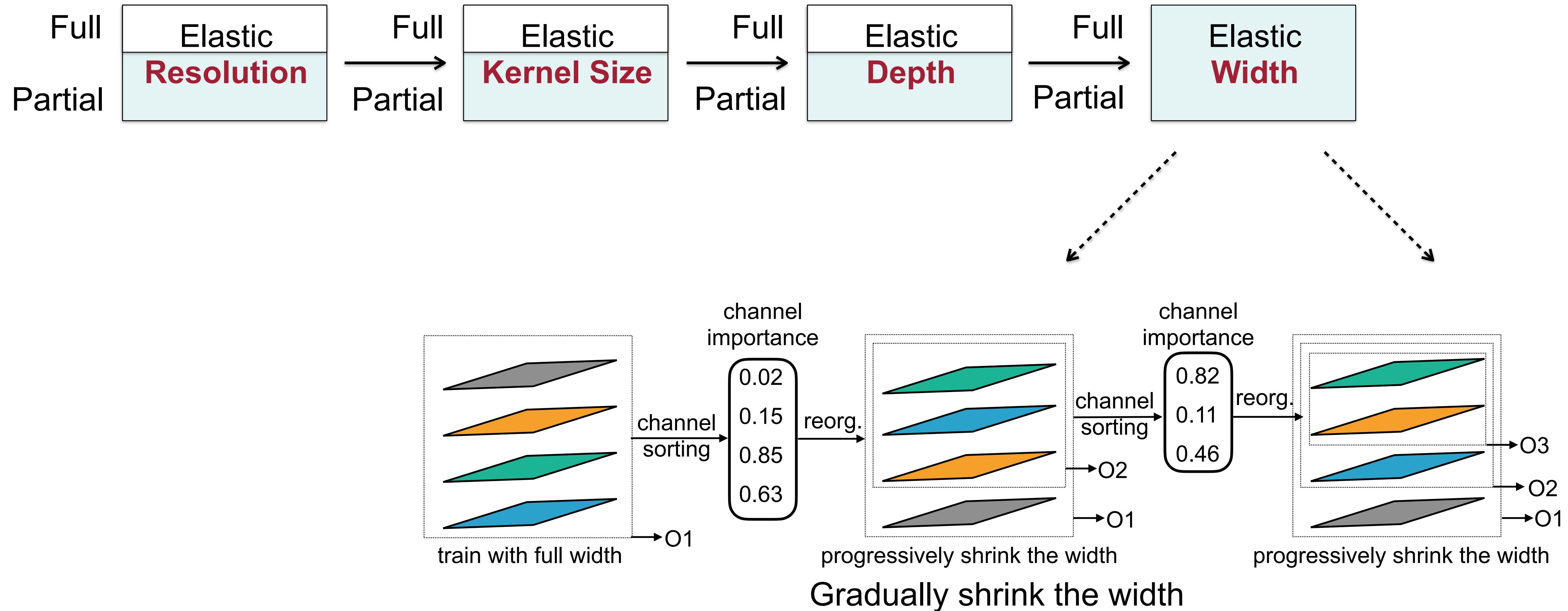
progressively prune the kernel size, depth, resolution



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

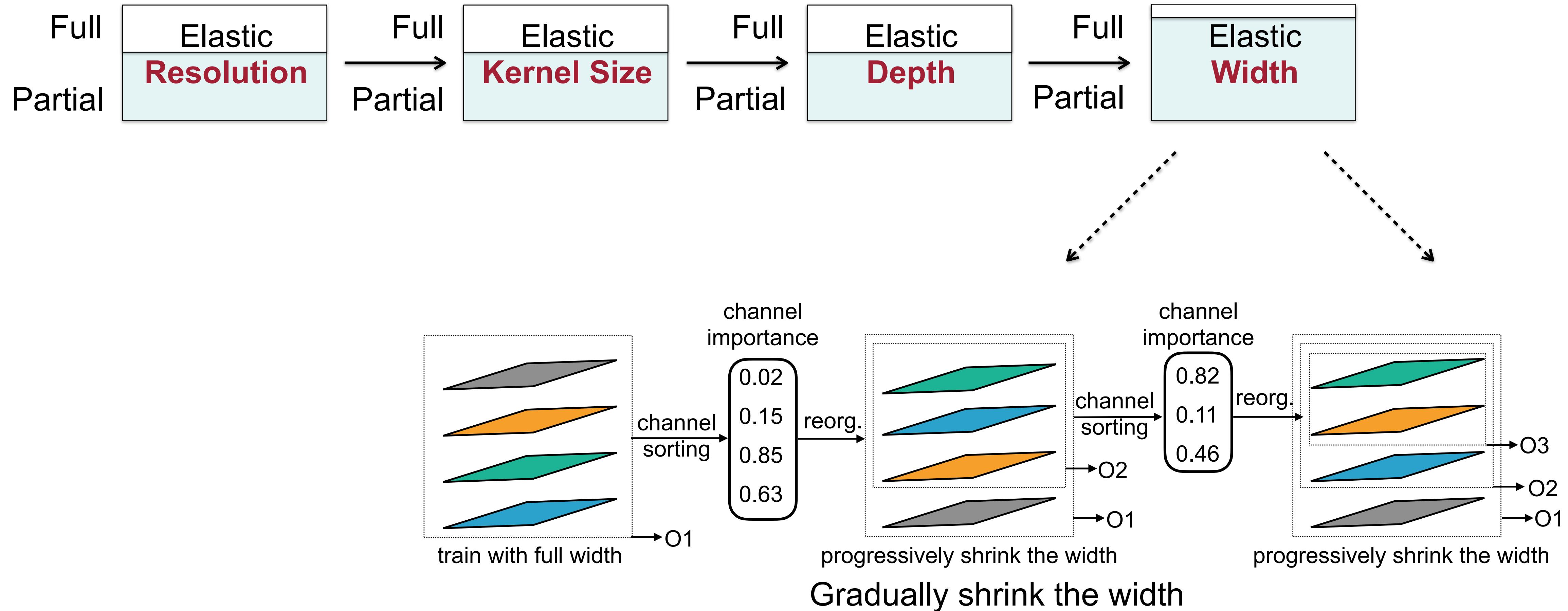
progressively prune the kernel size, depth, resolution



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

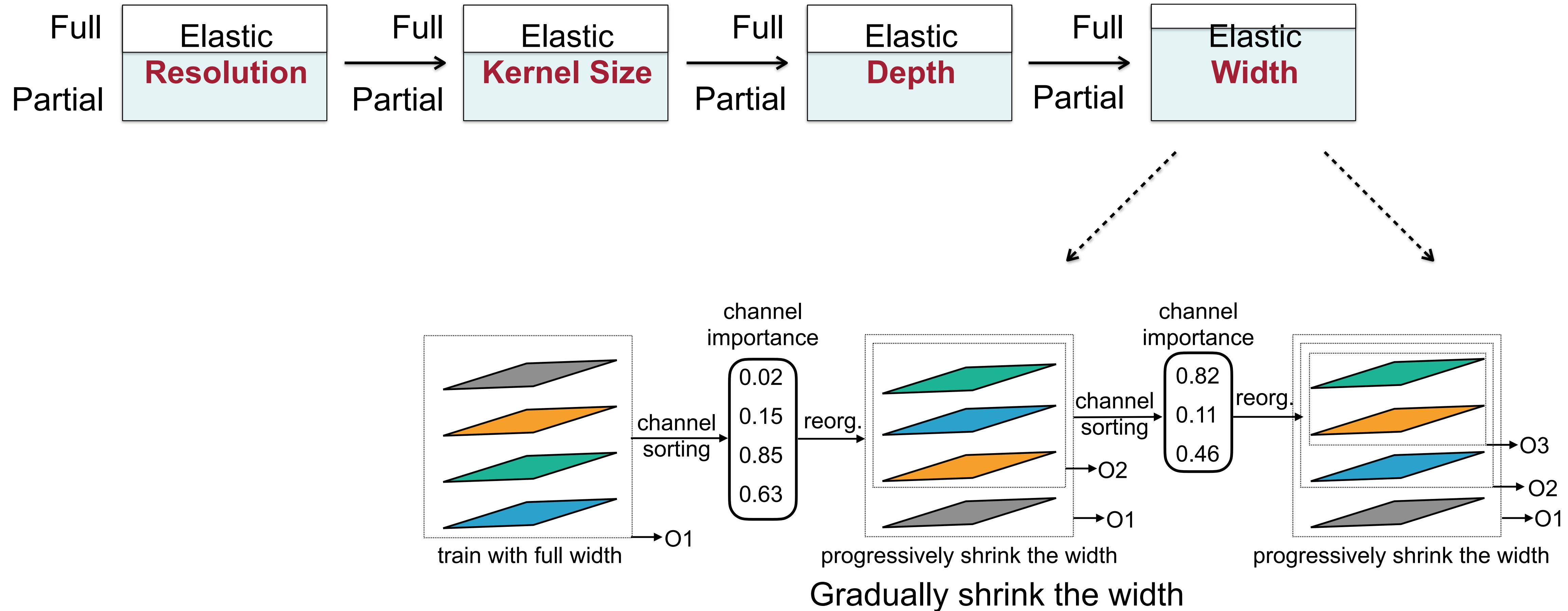
progressively prune the kernel size, depth, resolution



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

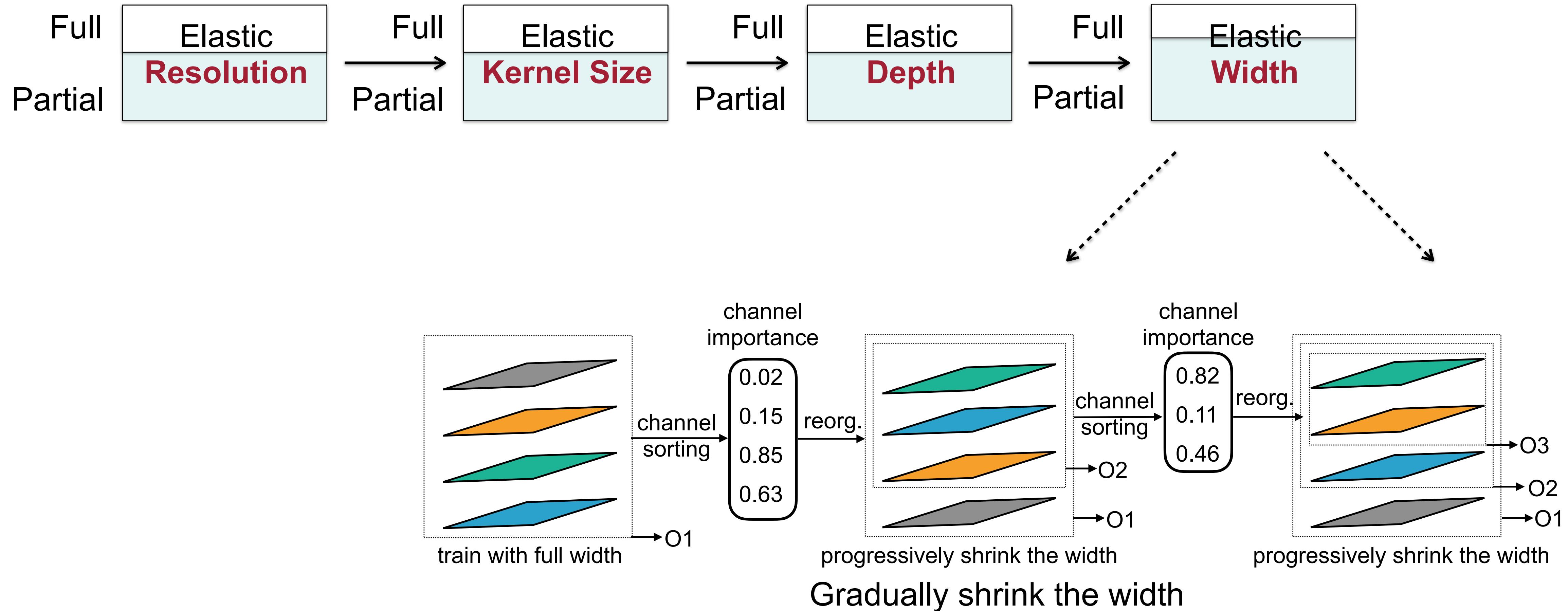
progressively prune the kernel size, depth, resolution



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

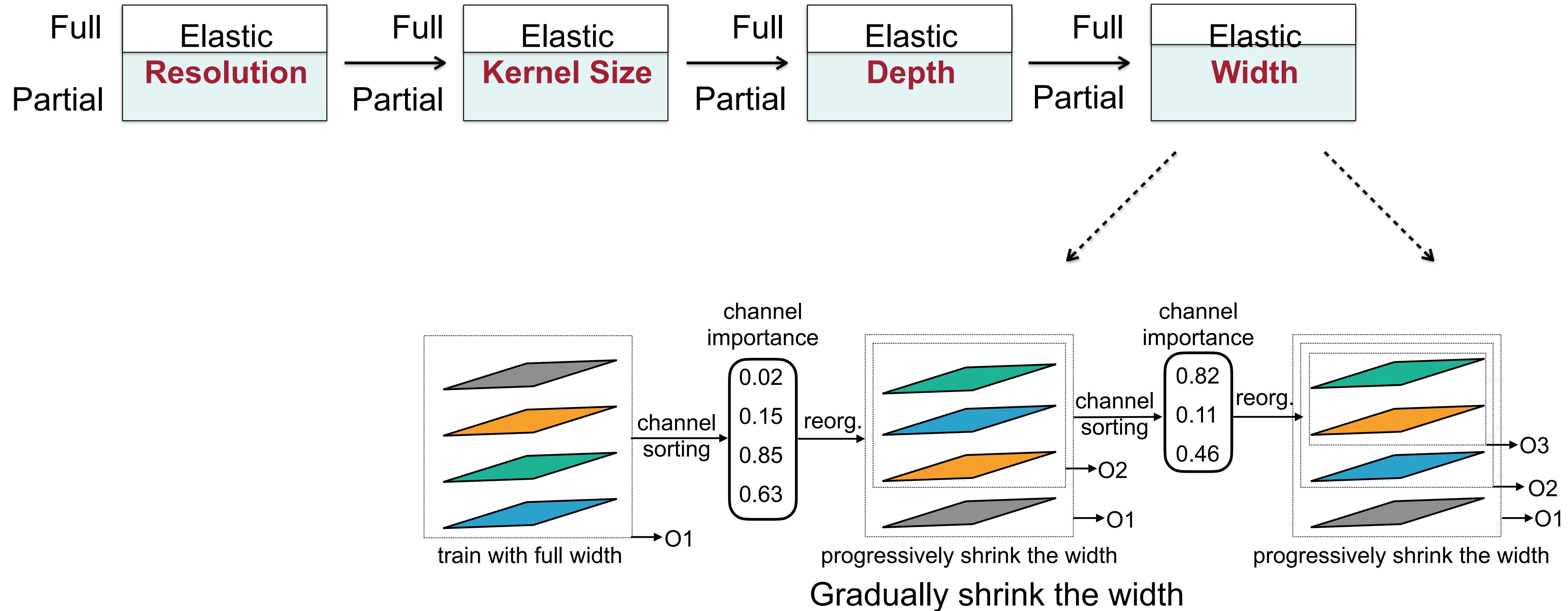
progressively prune the kernel size, depth, resolution



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

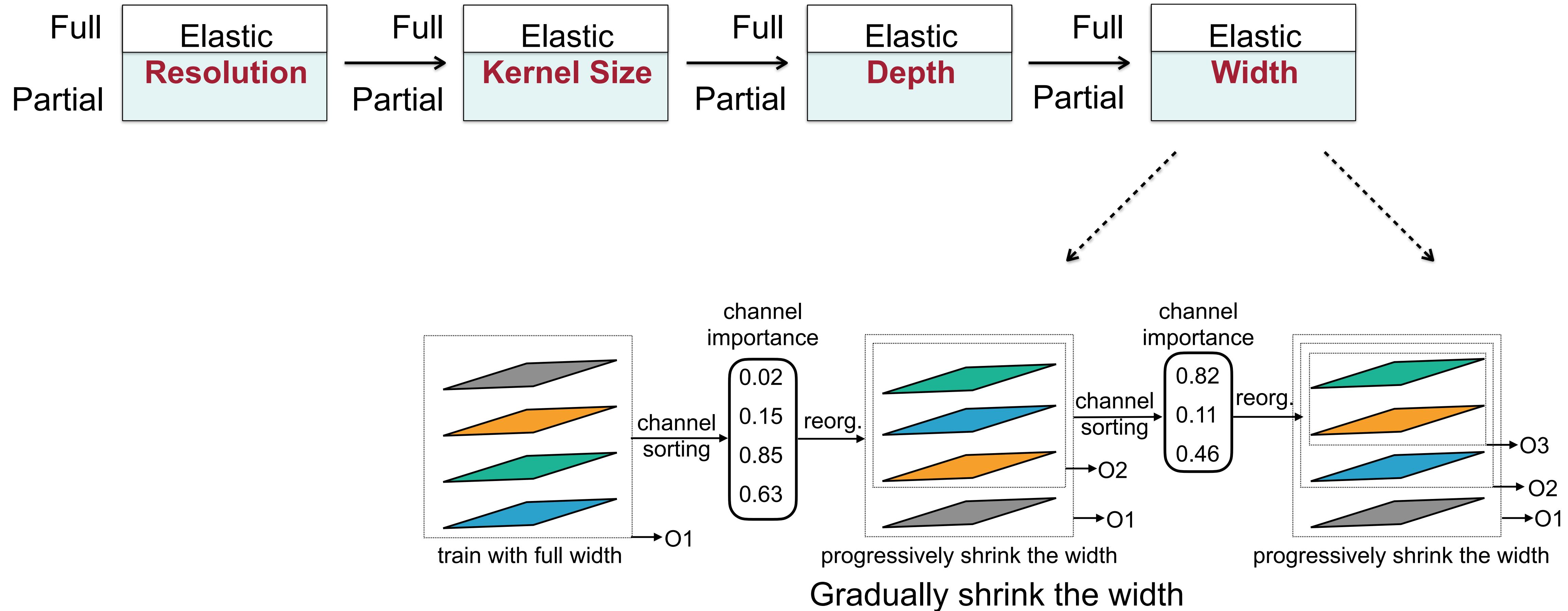
progressively prune the kernel size, depth, resolution



Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network

progressively prune the kernel size, depth, resolution

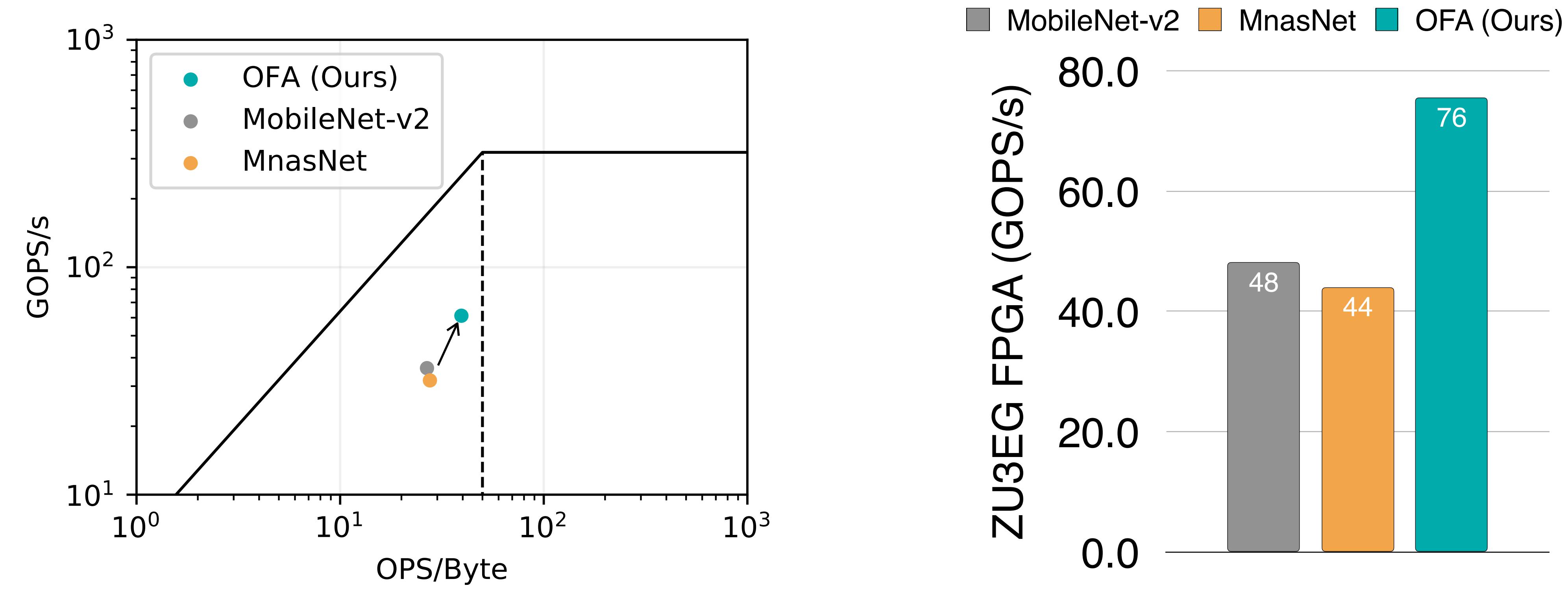


Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All Network: Roofline Analysis

Computation is cheap; memory is expensive.

- OFA designed model has higher arithmetic intensity (Ops/Byte)
  - => less memory bounded
  - => higher utilization and performance without changing the RTL

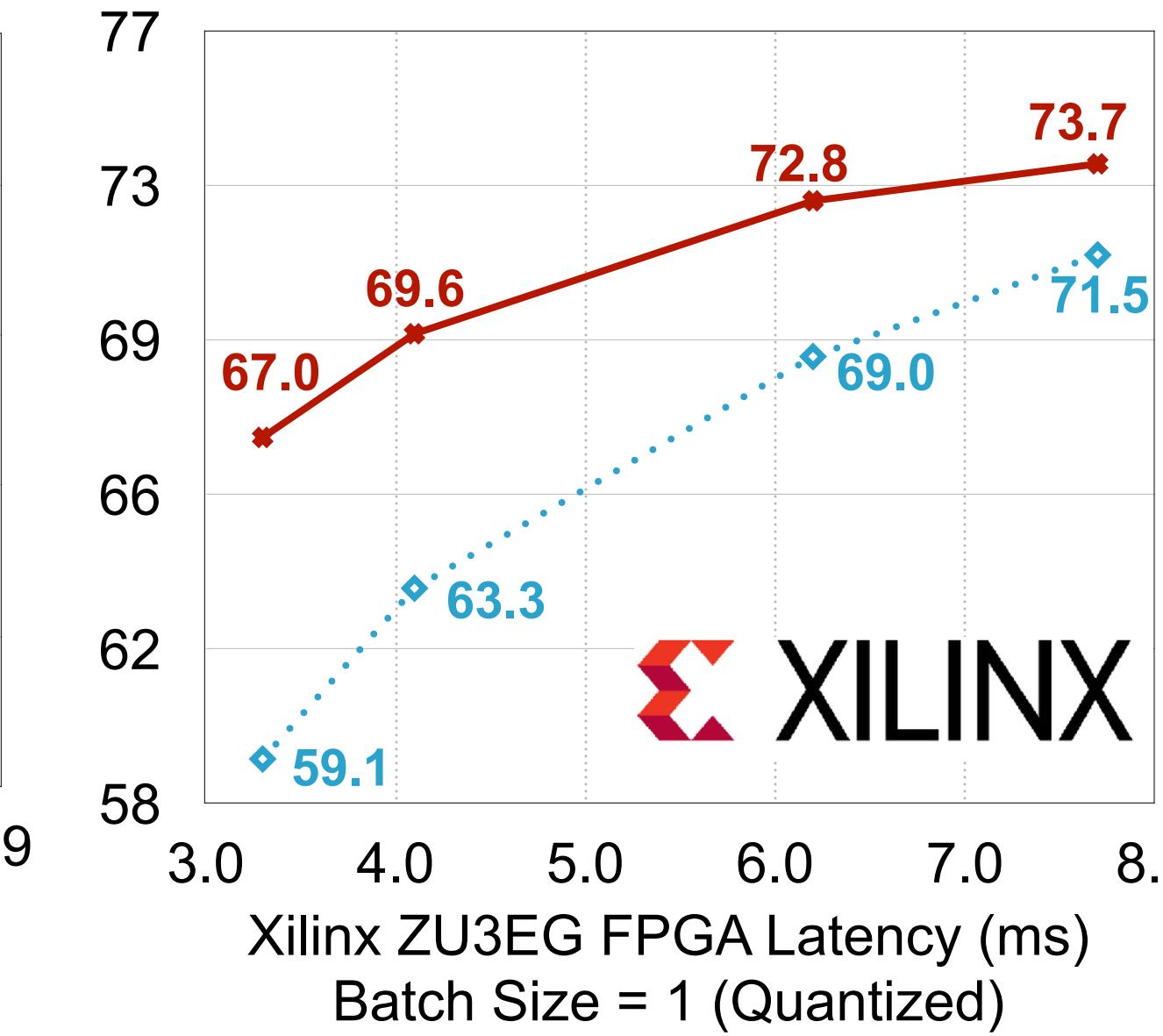
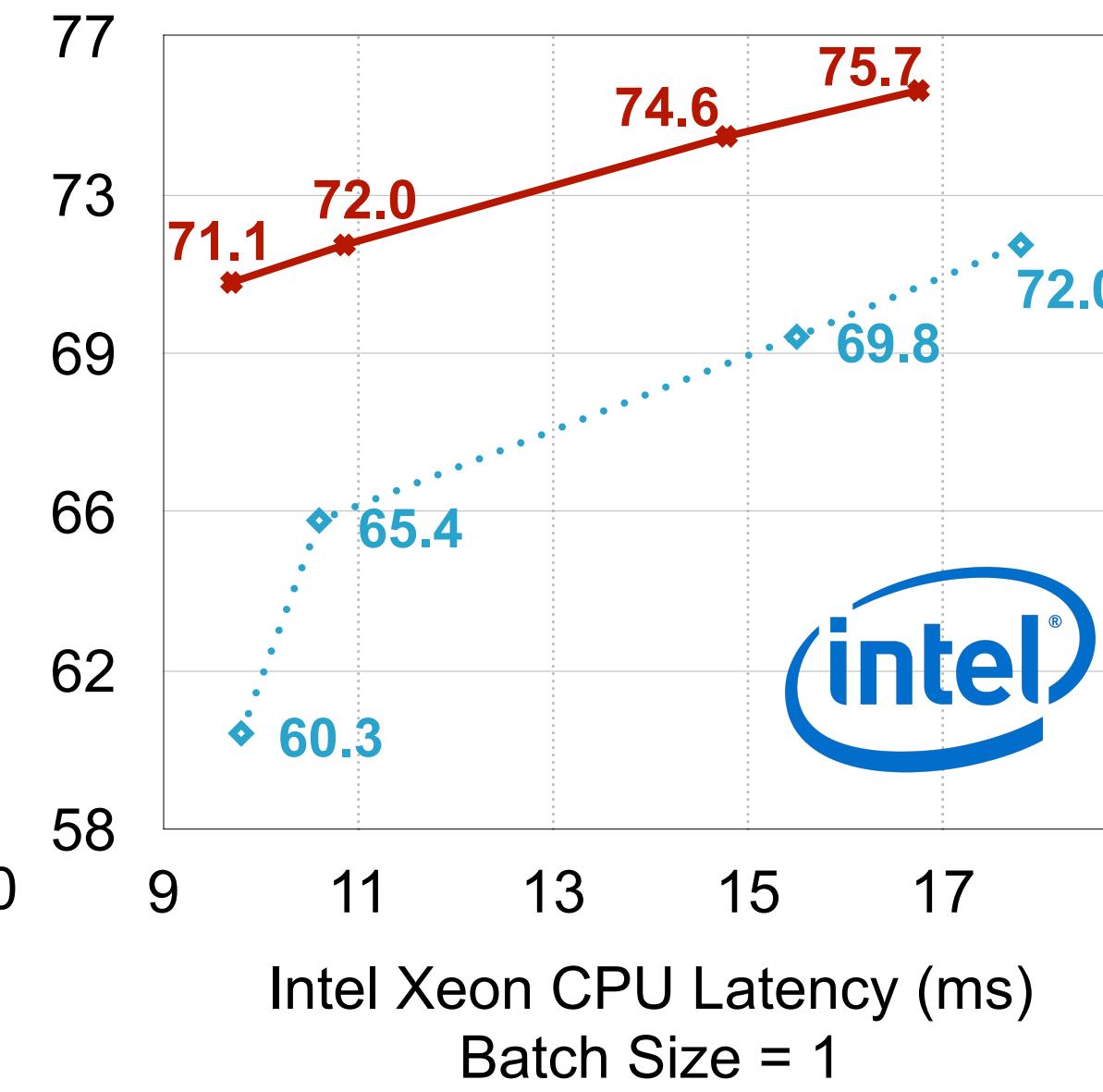
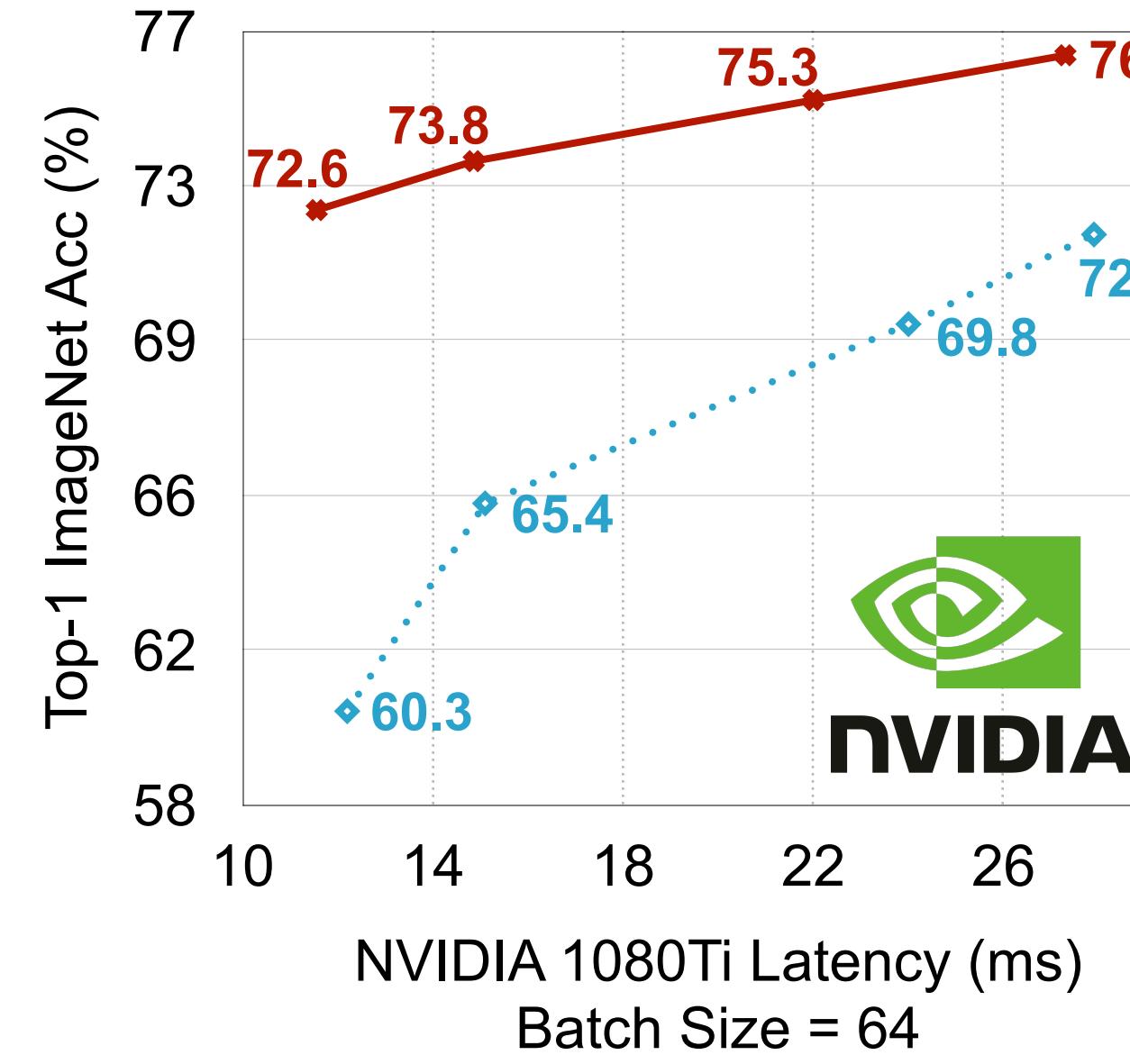
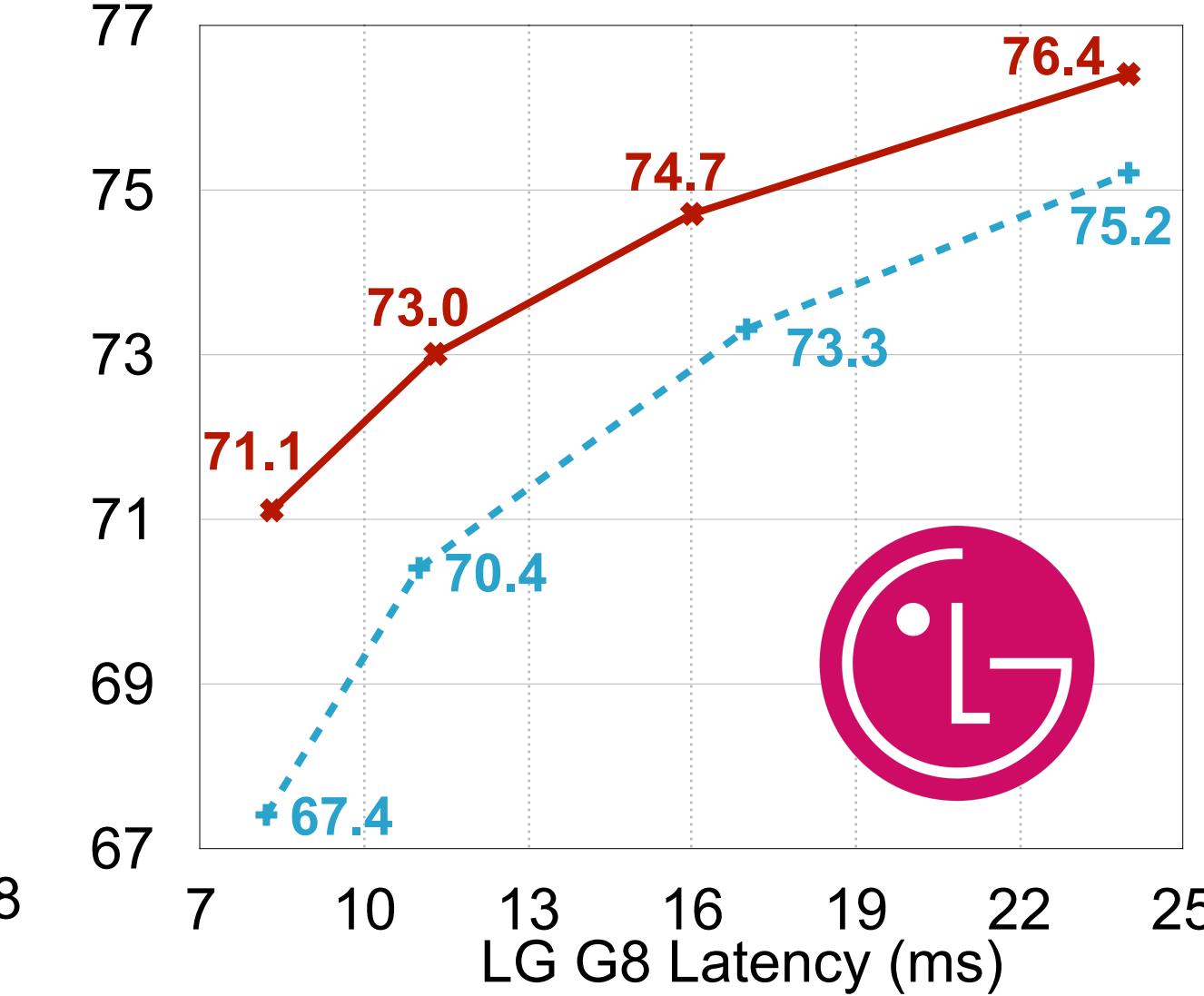
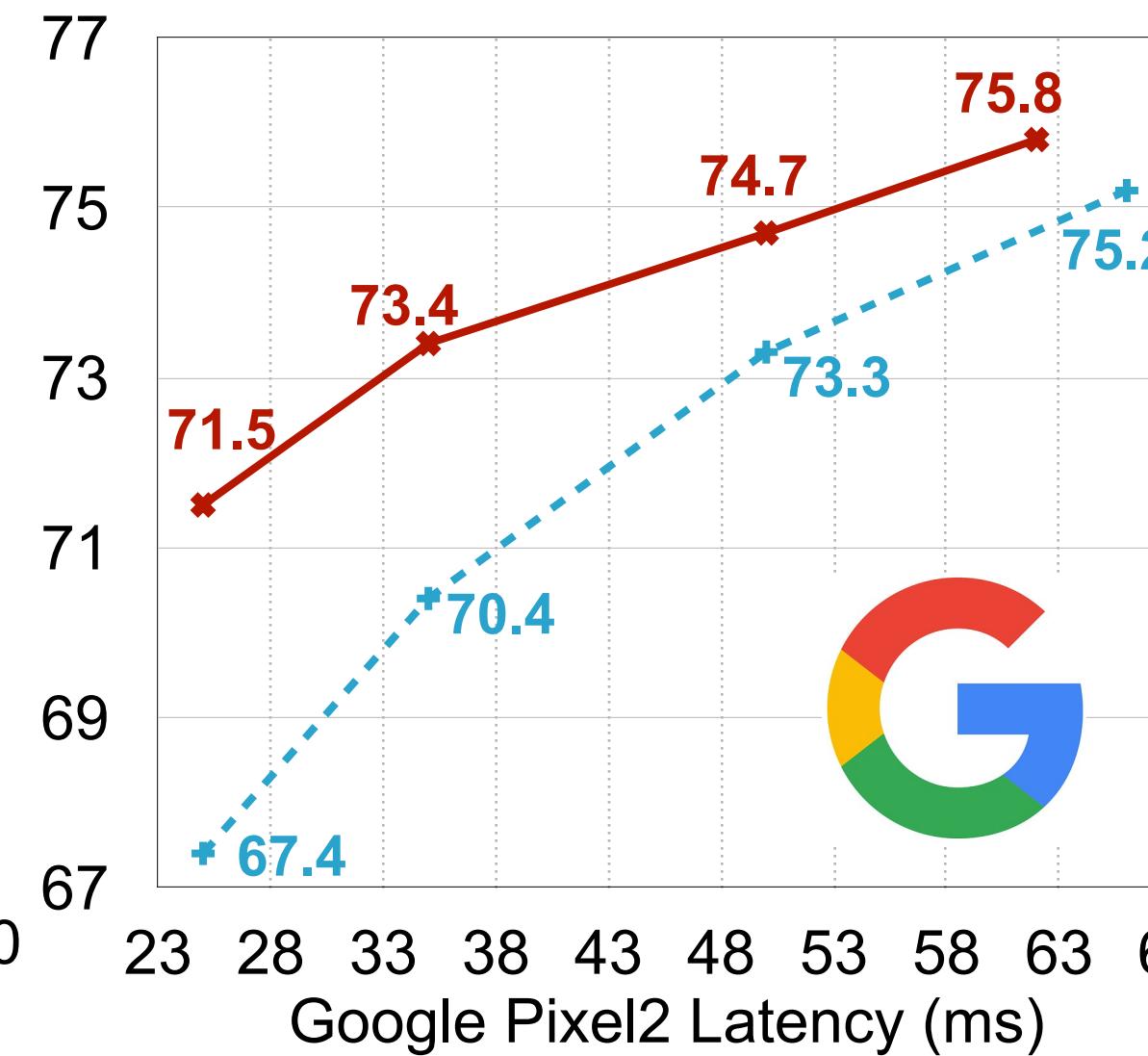
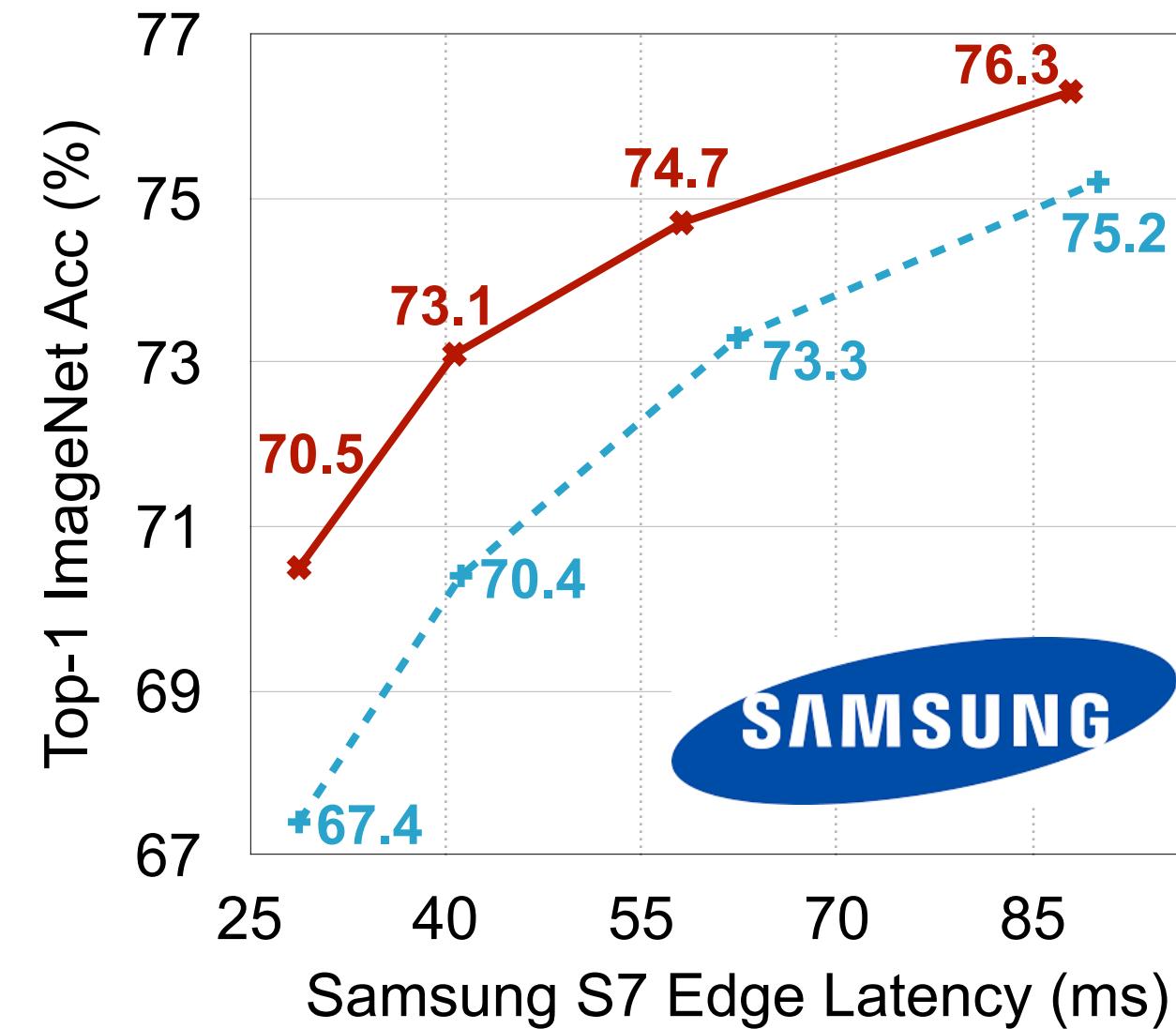


(b) on Xilinx ZU3EG FPGA

Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

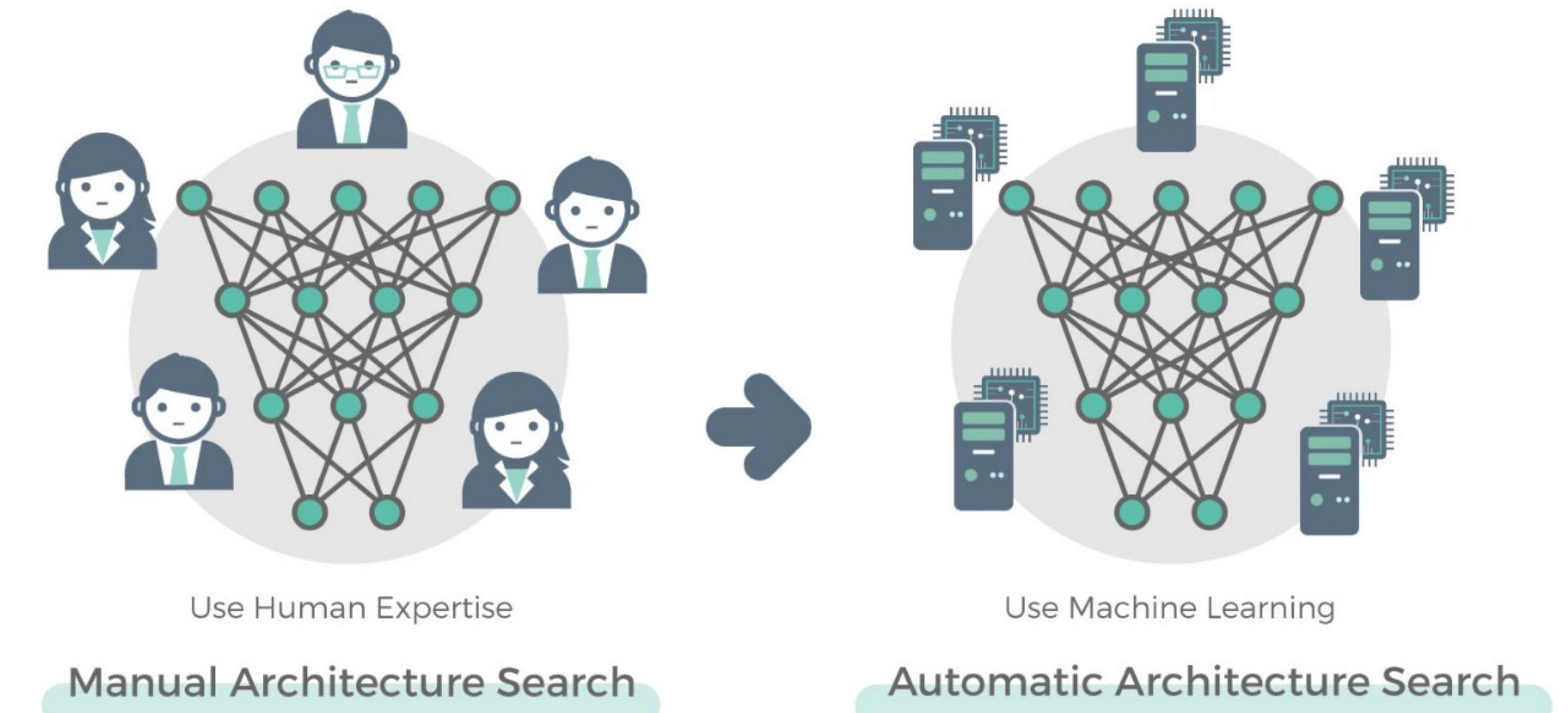
# OFA on Diverse HW Platforms

\* OFA + MobileNetV3  $\diamond$  MobileNetV2



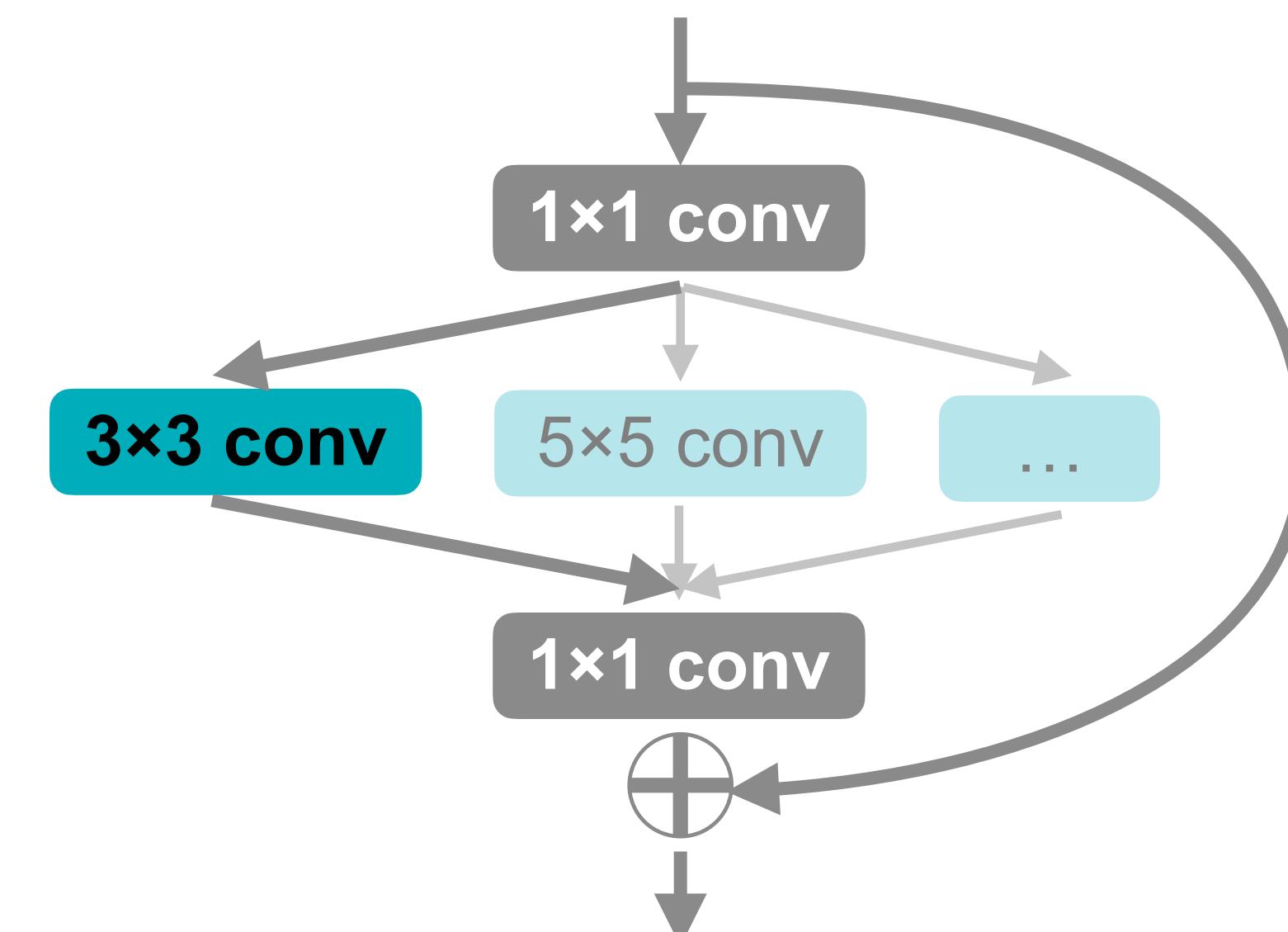
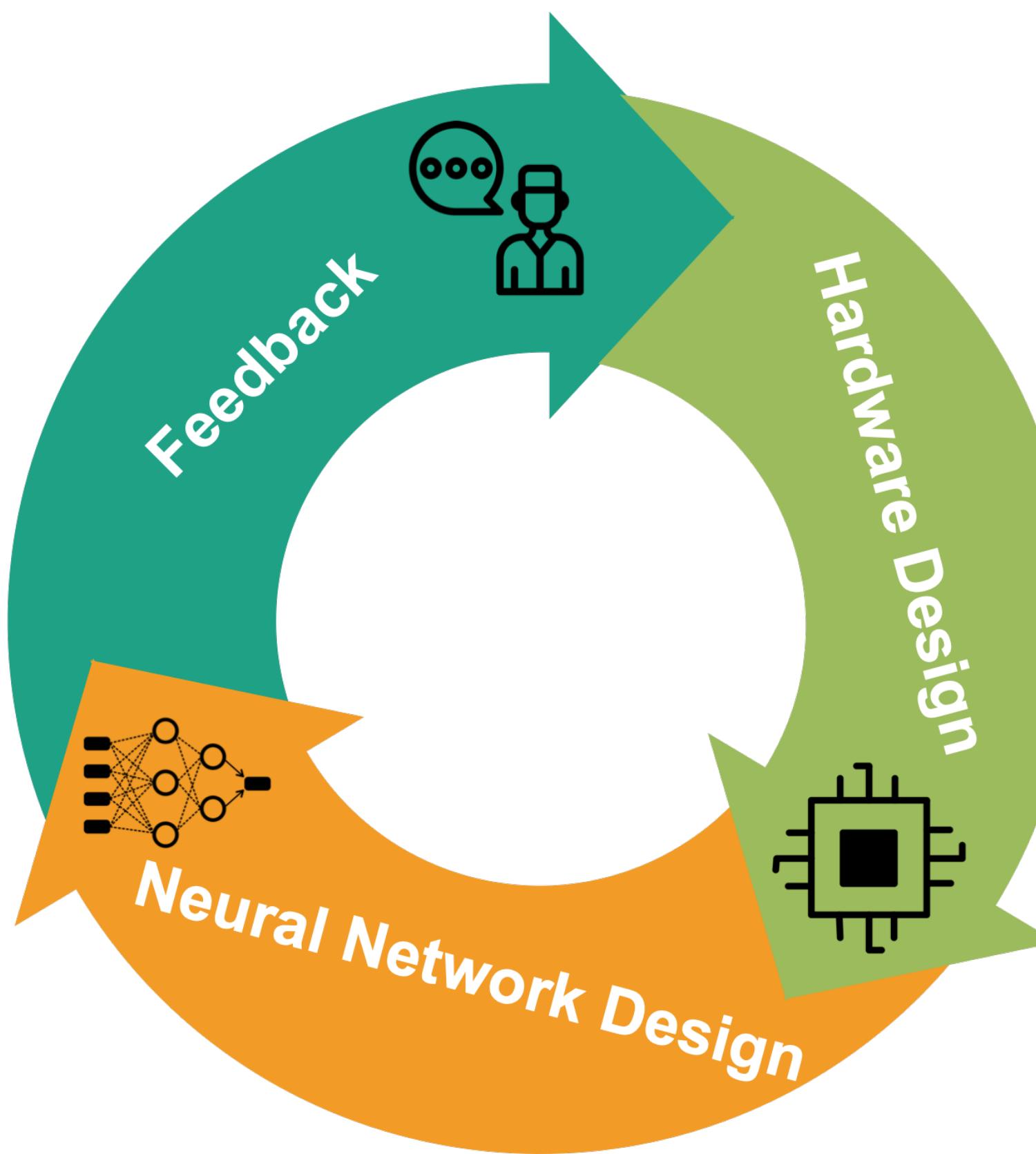
# Neural Architecture Search

- **Last Lecture:**
  - Primitive operations
  - Classic building blocks
  - **Introduction to neural architecture search (NAS)**
    - What is NAS?
    - Search space
    - Design the search space
    - Search strategy
- **This Lecture:**
  - **Efficient and hardware-aware NAS**
    - Accuracy estimation strategy
    - Zero-shot NAS
    - Hardware-aware NAS
    - Neural-hardware architecture co-search
  - **NAS applications**
    - NLP, GAN, point cloud, pose, etc

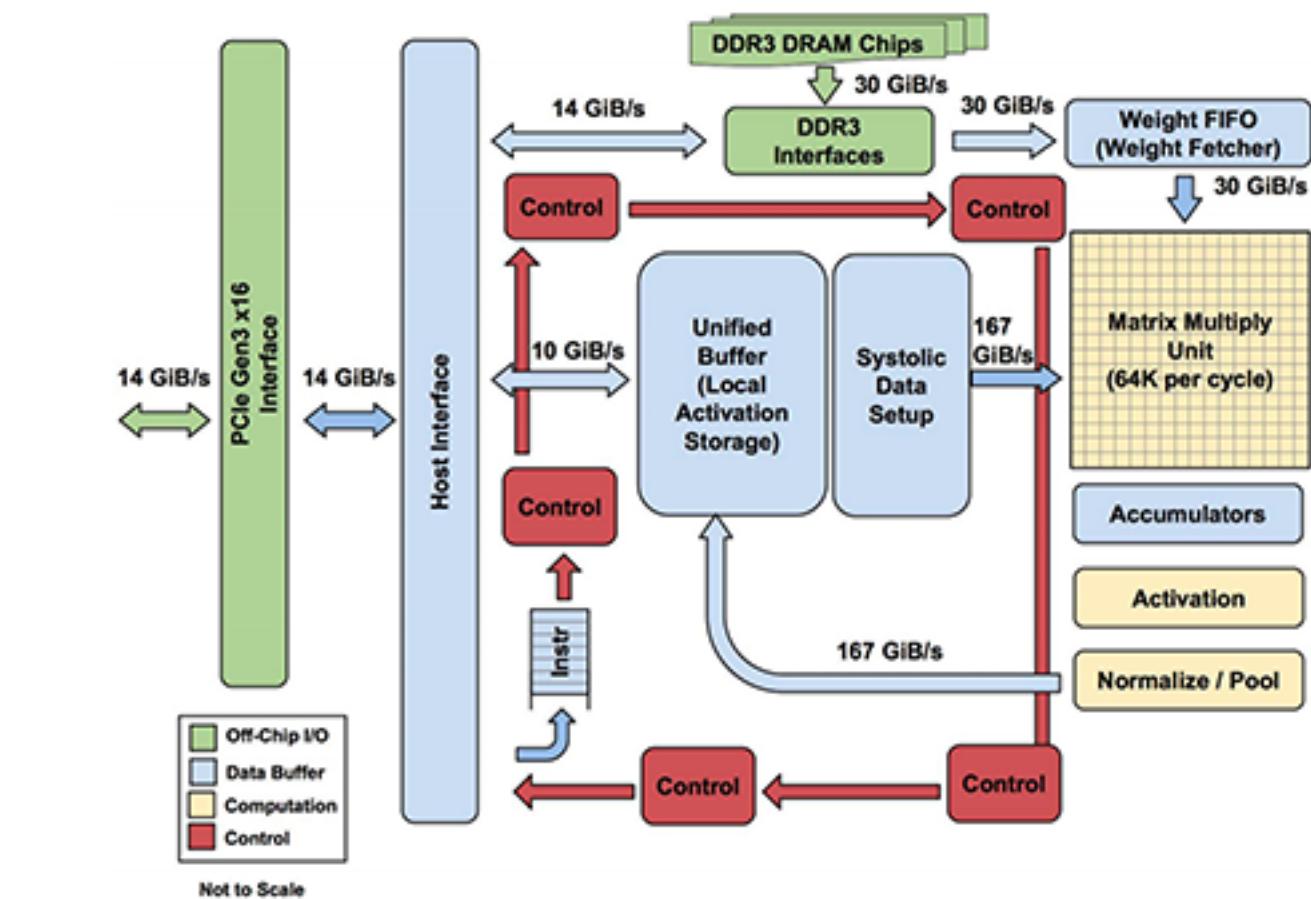


# Accelerating Deep Learning Computing

Both **neural architecture** and **accelerator architecture** design are important to enable specialization and acceleration.



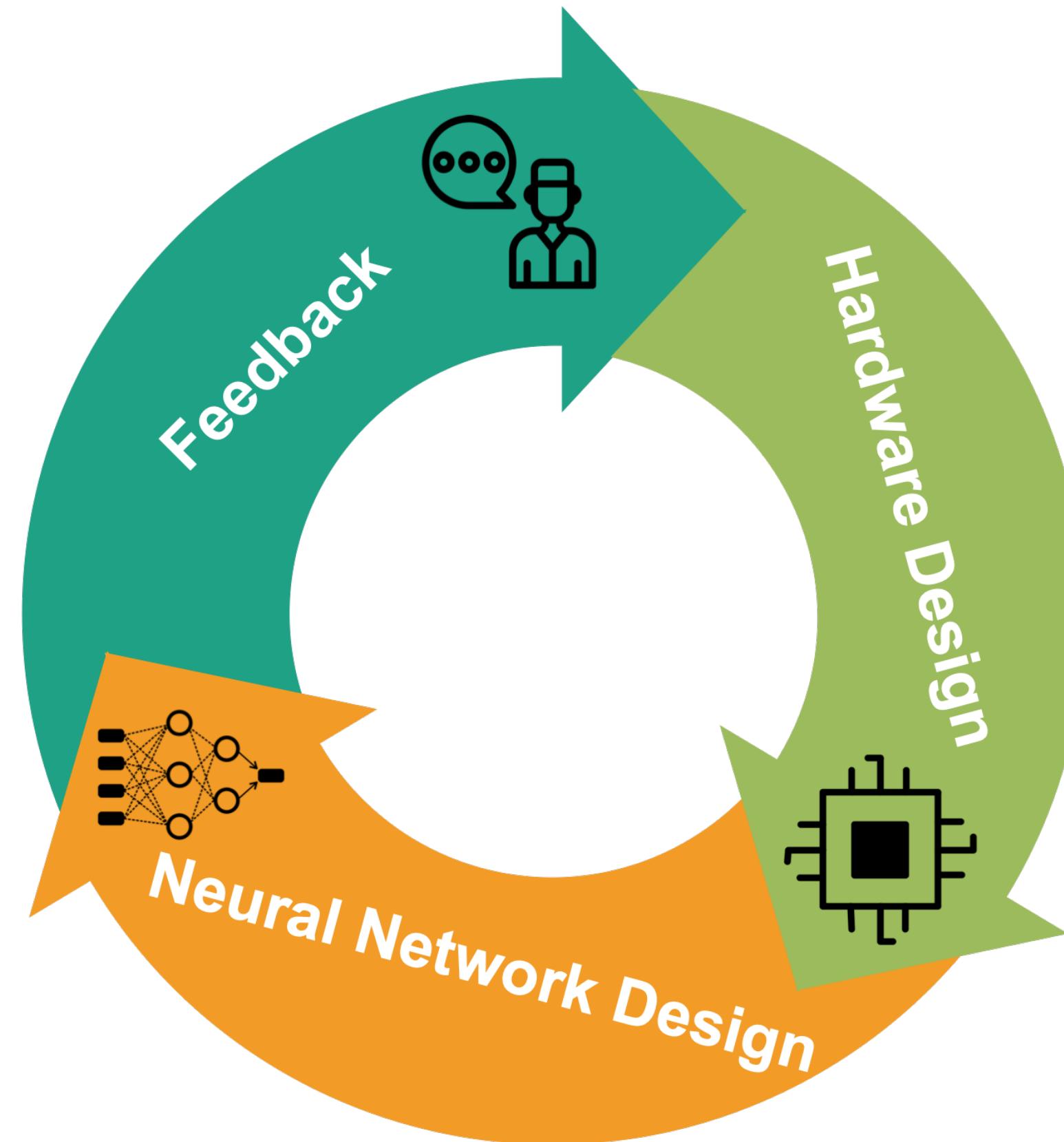
Neural Architecture



Accelerator Architecture

NAAS: Neural Accelerator Architecture Search [Lin et al., DAC 2021]  
In-Datacenter Performance Analysis of a Tensor Processing Unit [Jouppi et al., ISCA 2017]

# Design Spaces

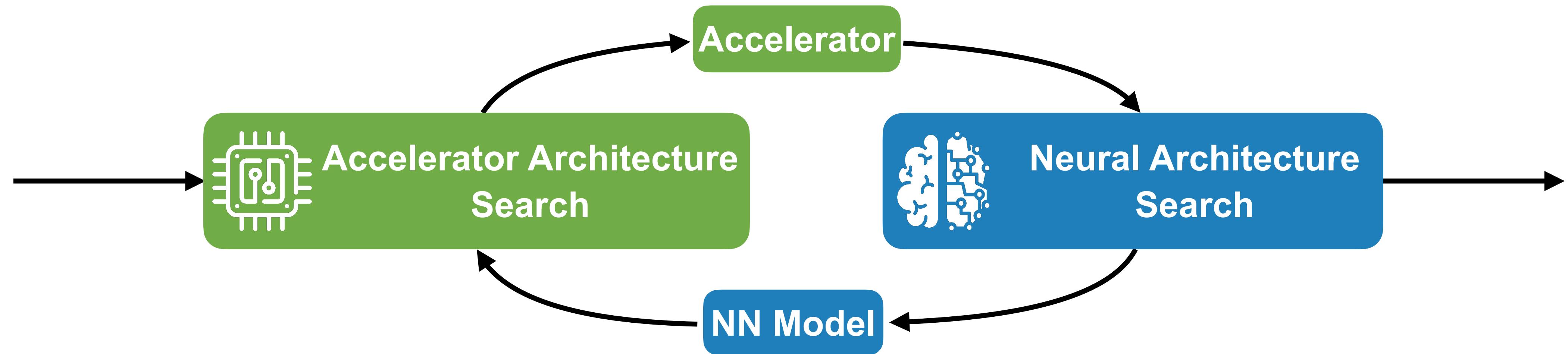


Key Dimensions	
Accelerator	Local Buffer Size, Global Buffer Size, #PEs
Compiler	Compute Array Size, PE Connectivity
Neural Network	Loop Orders, Loop Tiling Size, Dataflow
	#Layers, #Channels, Kernel Size, Bypass
	(Input / Weight) Quantization Precision

NAAS: Neural Accelerator Architecture Search [Lin et al., DAC 2021]

# Jointly Search Accelerator and Neural Network

- Searching accelerator and neural architecture *in one optimization loop* offers highly matched solutions



NAAS: Neural Accelerator Architecture Search [Lin et al., DAC 2021]

# Architecture Design Space

Key Dimensions		
Accelerator	Local Buffer Size, Global Buffer Size, #PEs	Architectural Sizing
	Compute Array Size, PE Connectivity	Connectivity Parameters
Compiler	Loop Orders, Loop Tiling Size, Dataflow	
	#Layers, #Channels, Kernel Size, Bypass	
	(Input / Weight) Quantization Precision	
Neural Network		

NAAS: Neural Accelerator Architecture Search [Lin et al., DAC 2021]

# Convolution Loop Nests

- Convolution loop nests can be divided into two parts: temporal mapping and spatial parallelism

Tensor Dimension	Notation
Batch	N
Output Channel	K
Input Channel	C
Input (Output) Row	Y (Y')
Input (Output) Column	X (X')
Kernel Row	R
Kernel Column	S

```
For _R in range(R / R):
    For _S in range(S / S):
        For _C in range(C / T_C):
            For _Y' in range(Y' / T_Y'):
                For _X' in range(X' / T_X'):
                    For r in range(R):
                        For s in range(S):
                            For _k in range(K / 16):
                                For _y' in range(T_Y'):
                                    For _x' in range(T_X'):
                                        For _c in range(T_C / 16):
                                            Parallel-For _m in range(16):
                                            Parallel-For _n in range(16):
                                                c = _C * T_C + _c * 16;
                                                k = _k * 16;
                                                y' = _Y' * T_Y' + _y';
                                                x' = _X' * T_X' + _x';
                                                y = y' + r - R;
                                                x = x' + s - S;
                                                psum[b,k,y',x'] += acts[b,x,y,x]
                                                * wgts[k,c,y,x];

```

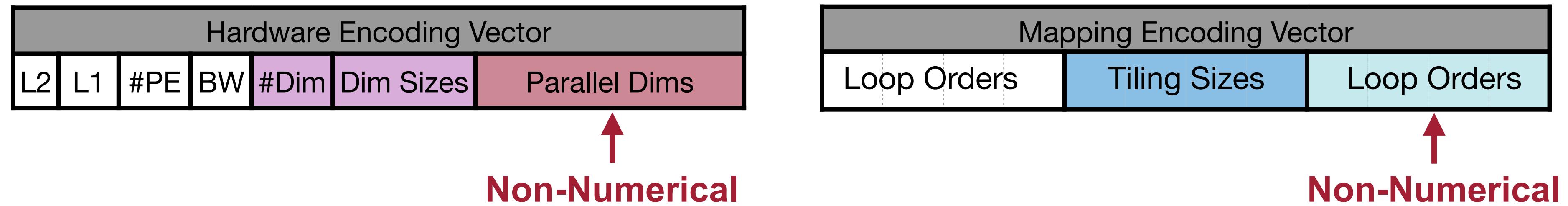


Loop Tiling

Loop Order

Hardware Parallelism

# Encoding Non-numerical Parameters

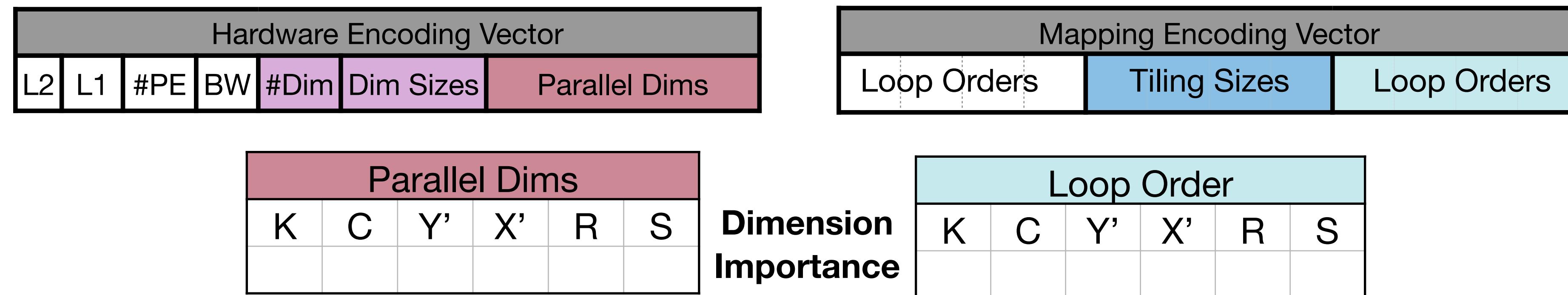


- Index-based Encoding
  - Increment/Decrement of index value does not convey any physical information

Non-Numerical Parameter Loop Orders	Numerical Encoding Value Index
CRXKYS	0
CXYRSK	1
...	...

NAAS: Neural Accelerator Architecture Search [Lin et al., DAC 2021]

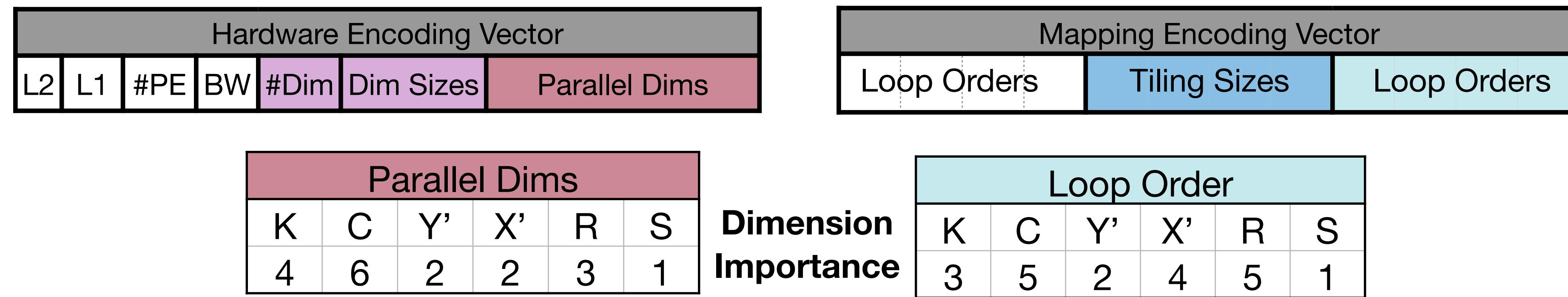
# Encoding Non-numerical Parameters



- **Importance-based Encoding**
  - Step 1: Fix the dimension position in the encoding vectors

NAAS: Neural Accelerator Architecture Search [Lin et al., DAC 2021]

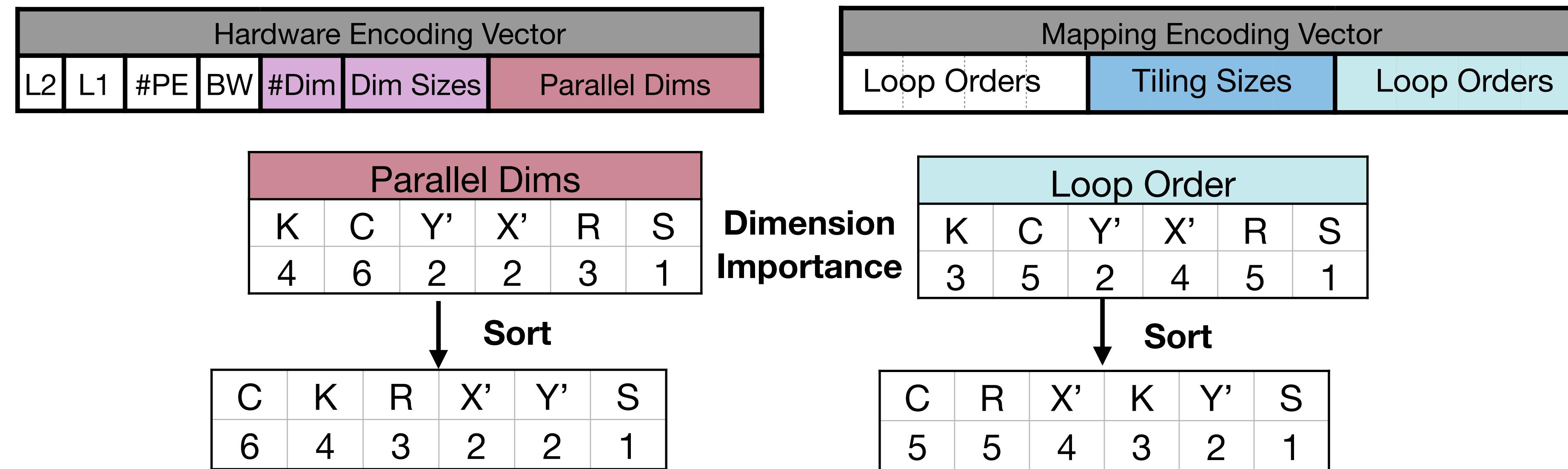
# Encoding Non-numerical Parameters



- **Importance-based Encoding**
  - Step 1: Fix the dimension position in the encoding vectors
  - Step 2: Optimizer assigns numerical importance to these dimensions
    - sampling based on multivariate normal distribution

NAAS: Neural Accelerator Architecture Search [Lin et al., DAC 2021]

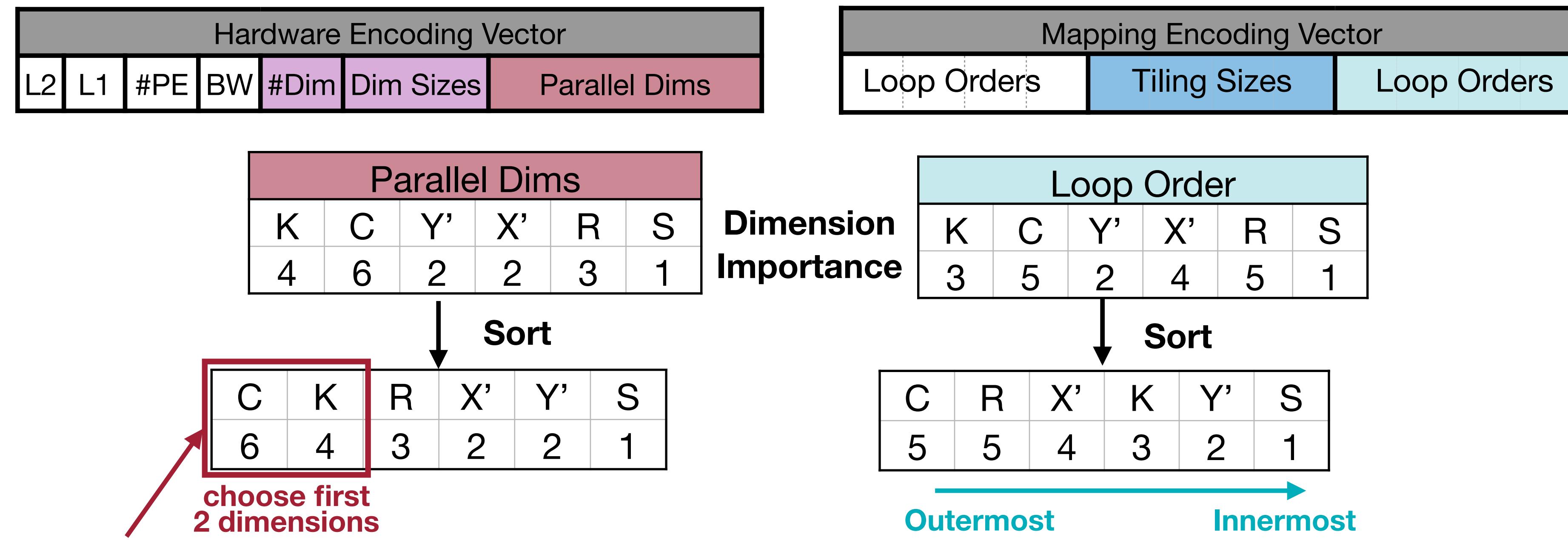
# Encoding Non-numerical Parameters



- **Importance-based Encoding**
  - Step 1: Fix the dimension position in the encoding vectors
  - Step 2: Optimizer assigns numerical importance to these dimensions
  - Step 3: Sort the dimensions by the importance value in decreasing order

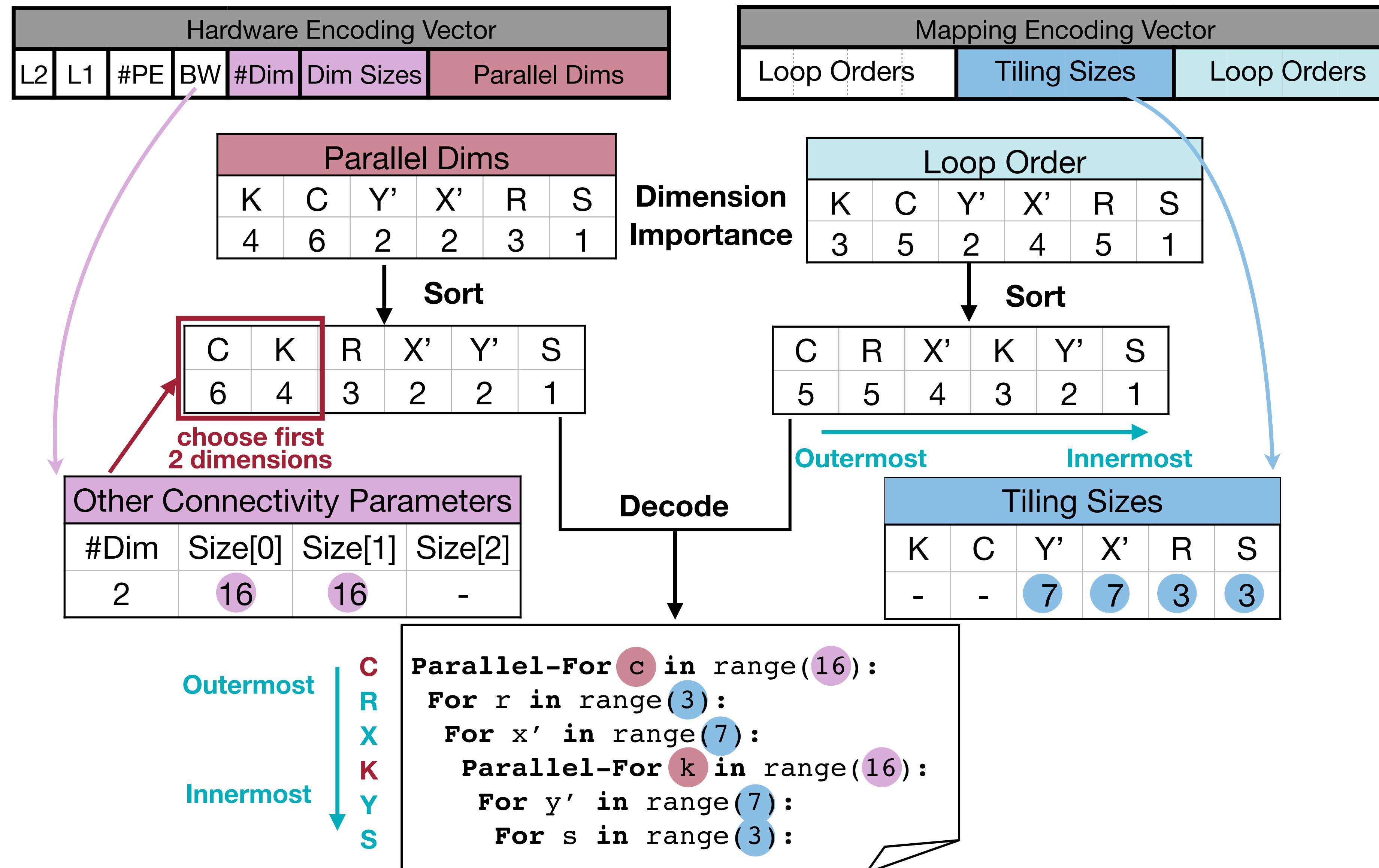
NAAS: Neural Accelerator Architecture Search [Lin et al., DAC 2021]

# Encoding Non-numerical Parameters

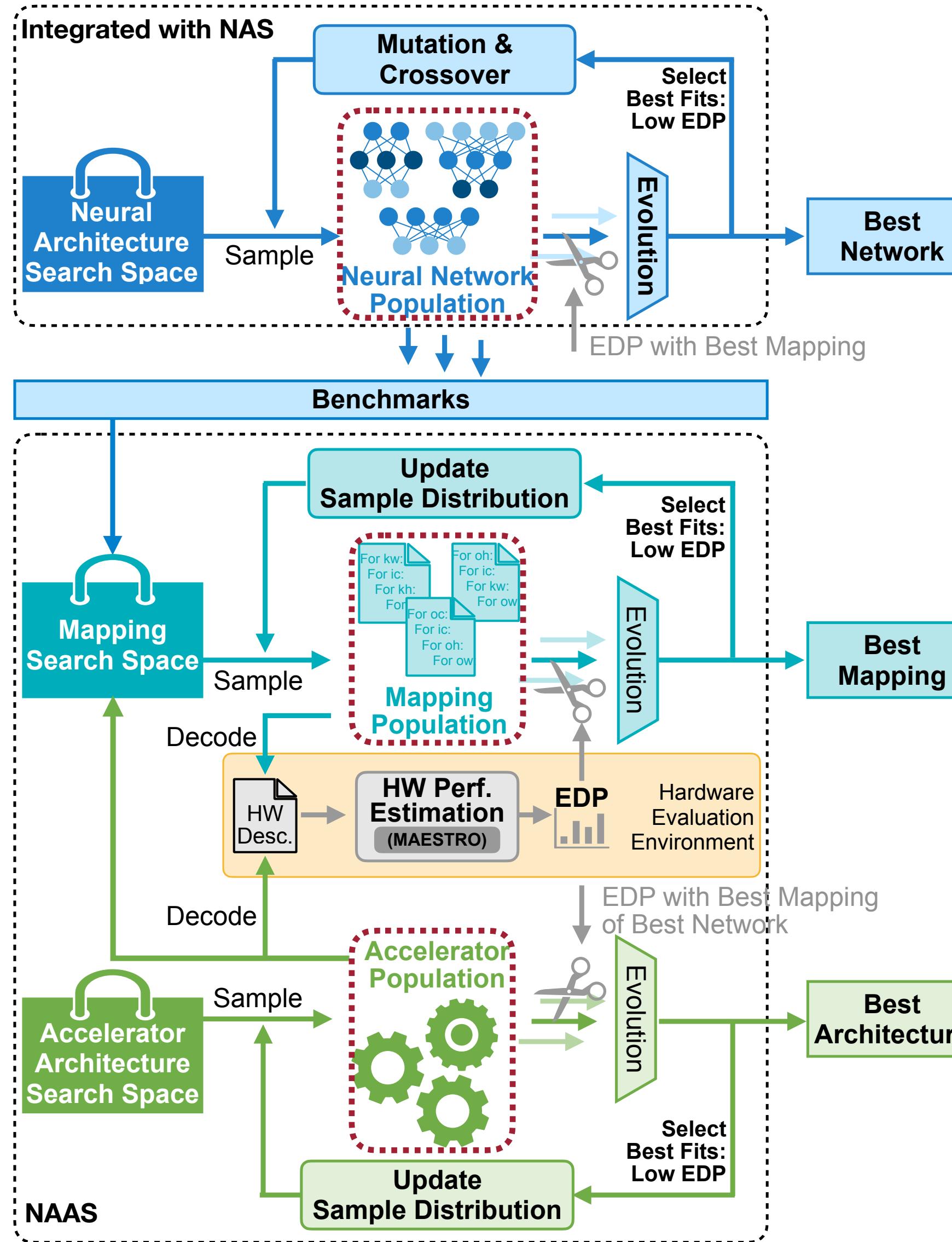


- **Importance-based Encoding**
  - Step 1: Fix the dimension position in the encoding vectors
  - Step 2: Optimizer assigns numerical importance to these dimensions
  - Step 3: Sort the dimensions by the importance value in decreasing order

# Encoding Non-numerical Parameters



# Jointly Optimize NN, Mapping, Accelerator



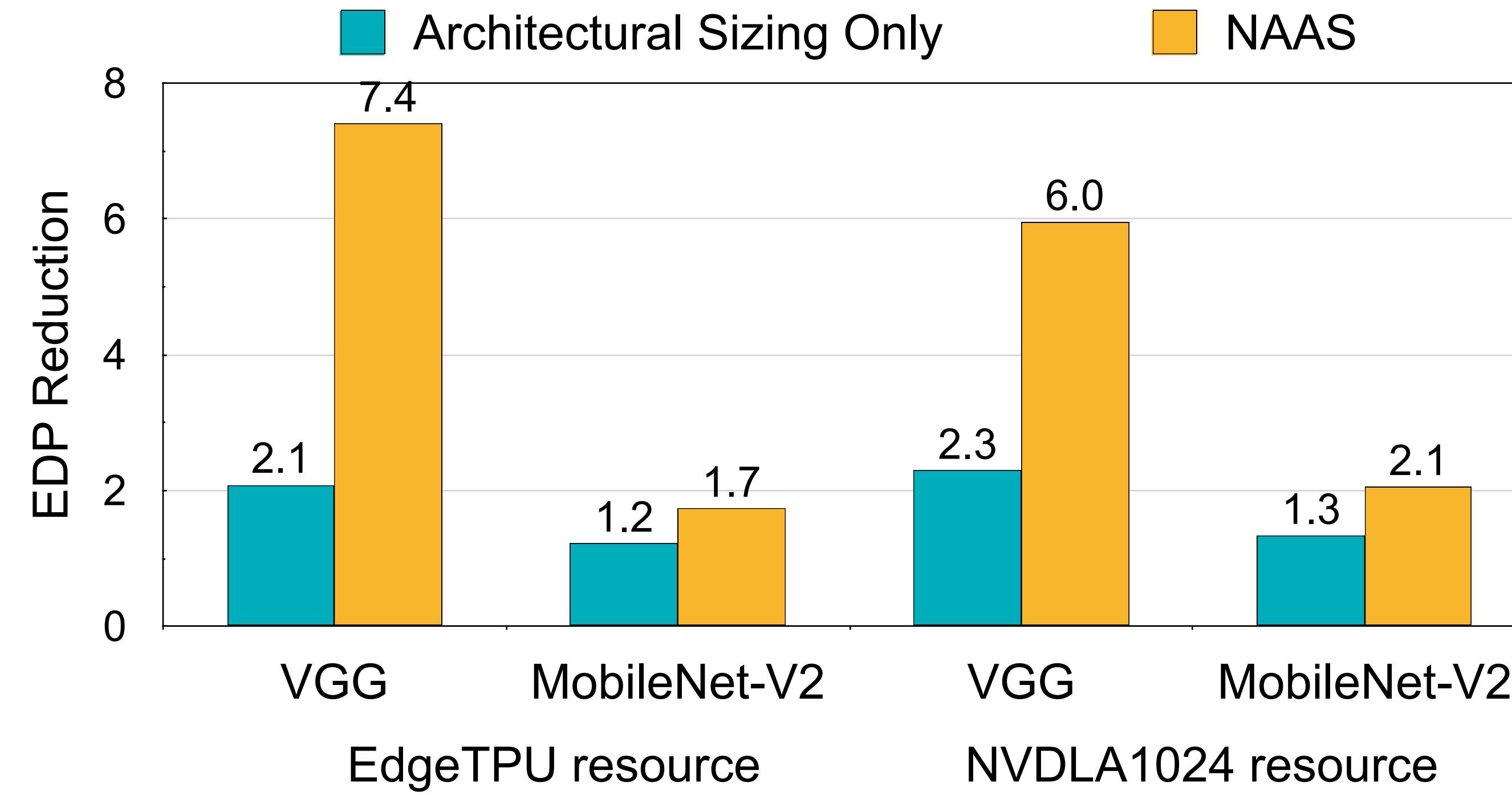
```

For epoch_naas in range(max_naas_epochs):
    accelerators = NAAS_generate_hardware()
    For hw in accelerators:
        For epoch_ofa in range(max_ofa_epochs):
            networks = OFA_generate_networks(accuracy)
            For nn in networks:
                map = NAAS_optimize_mappings(hw, nn)
                edp = NAAS_get_edp(hw, nn, map)
                OFA_update_optimizer(nn, edp)
                best_nn, best_map, best_edp = OFA_update_best(nn, map, edp)
            NAAS_update_optimizer(hw, best_nn, best_map, best_edp)

```

NAAS: Neural Accelerator Architecture Search [Lin et al., DAC 2021]

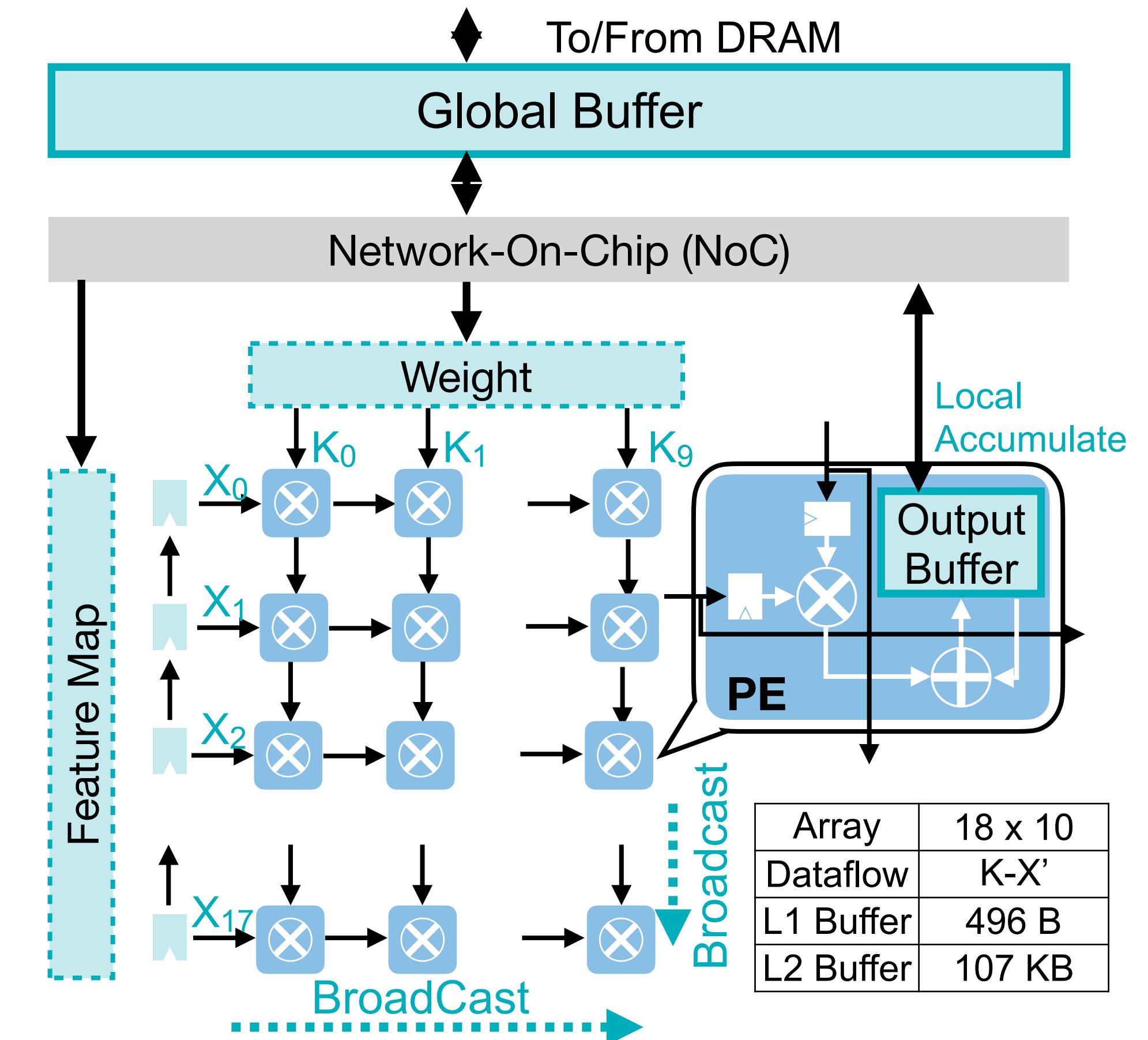
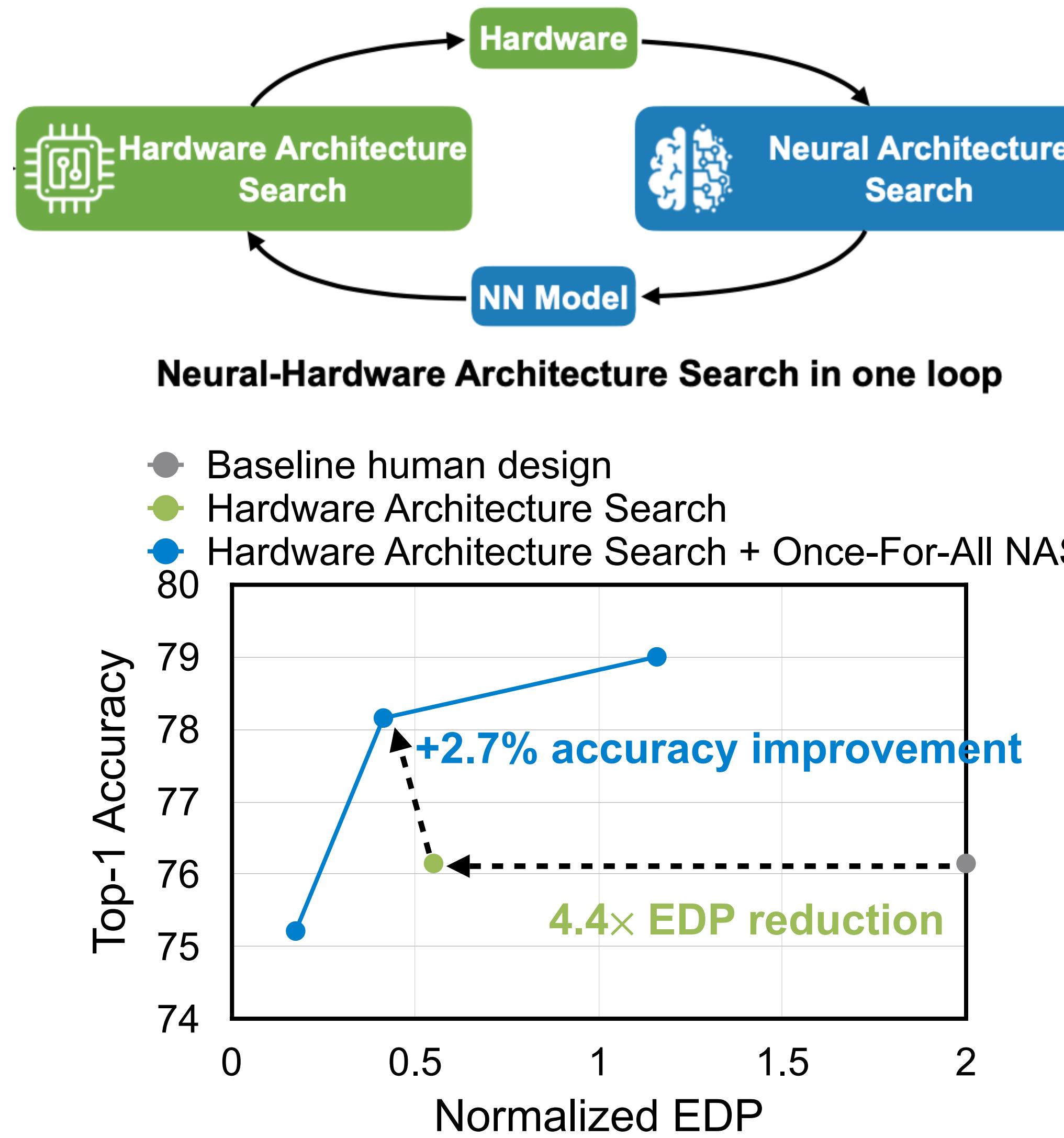
# Search Beyond Architecture Sizing



- Compared to searching the architectural sizing only, searching the connectivity parameters and mapping strategies as well achieves considerable EDP reduction.

NAAS: Neural Accelerator Architecture Search [Lin et al., DAC 2021]

# NAAS outperforms human design

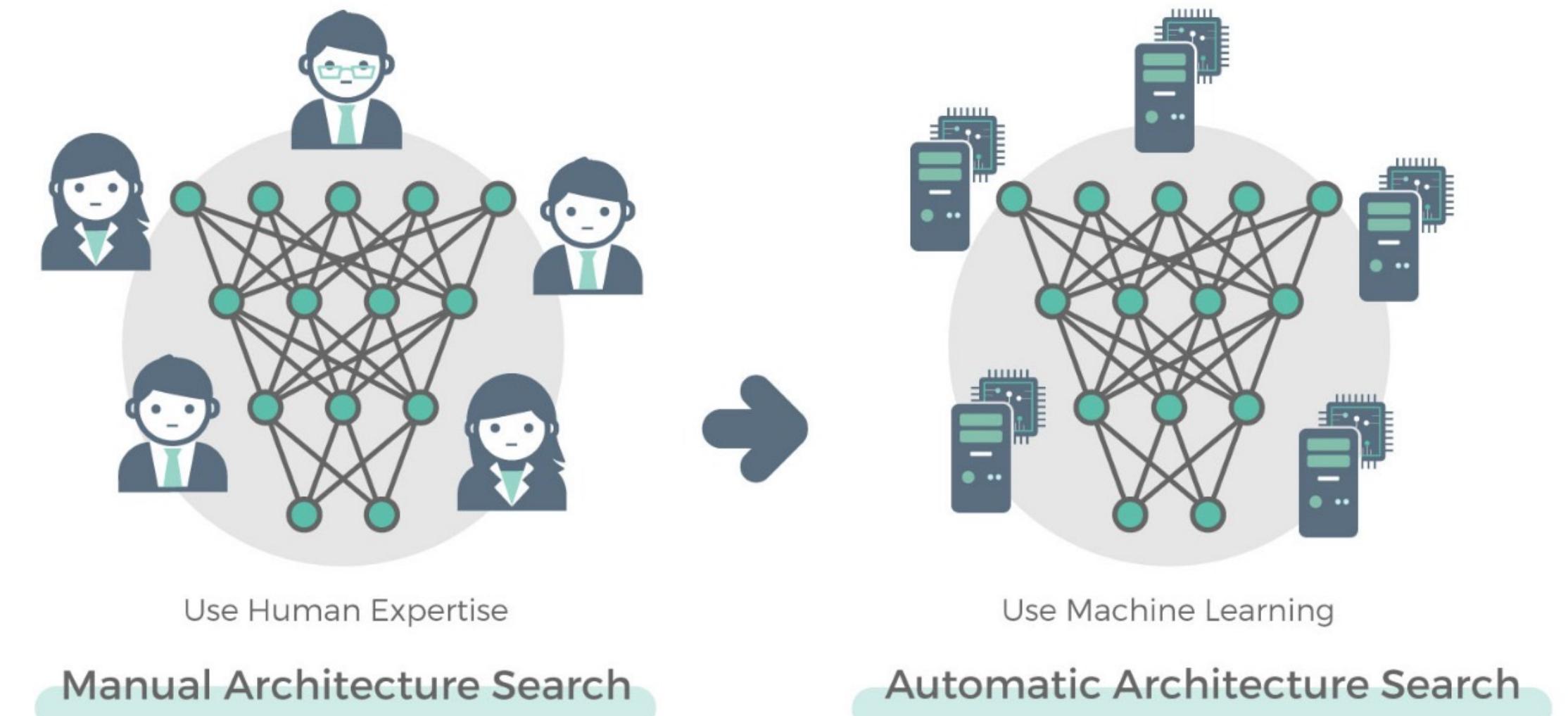


NAAS designed data flow parallelizes the output height and output channel dimension, which is very different from the human design

NAAS: Neural Accelerator Architecture Search [Lin et al., DAC 2021]

# Neural Architecture Search

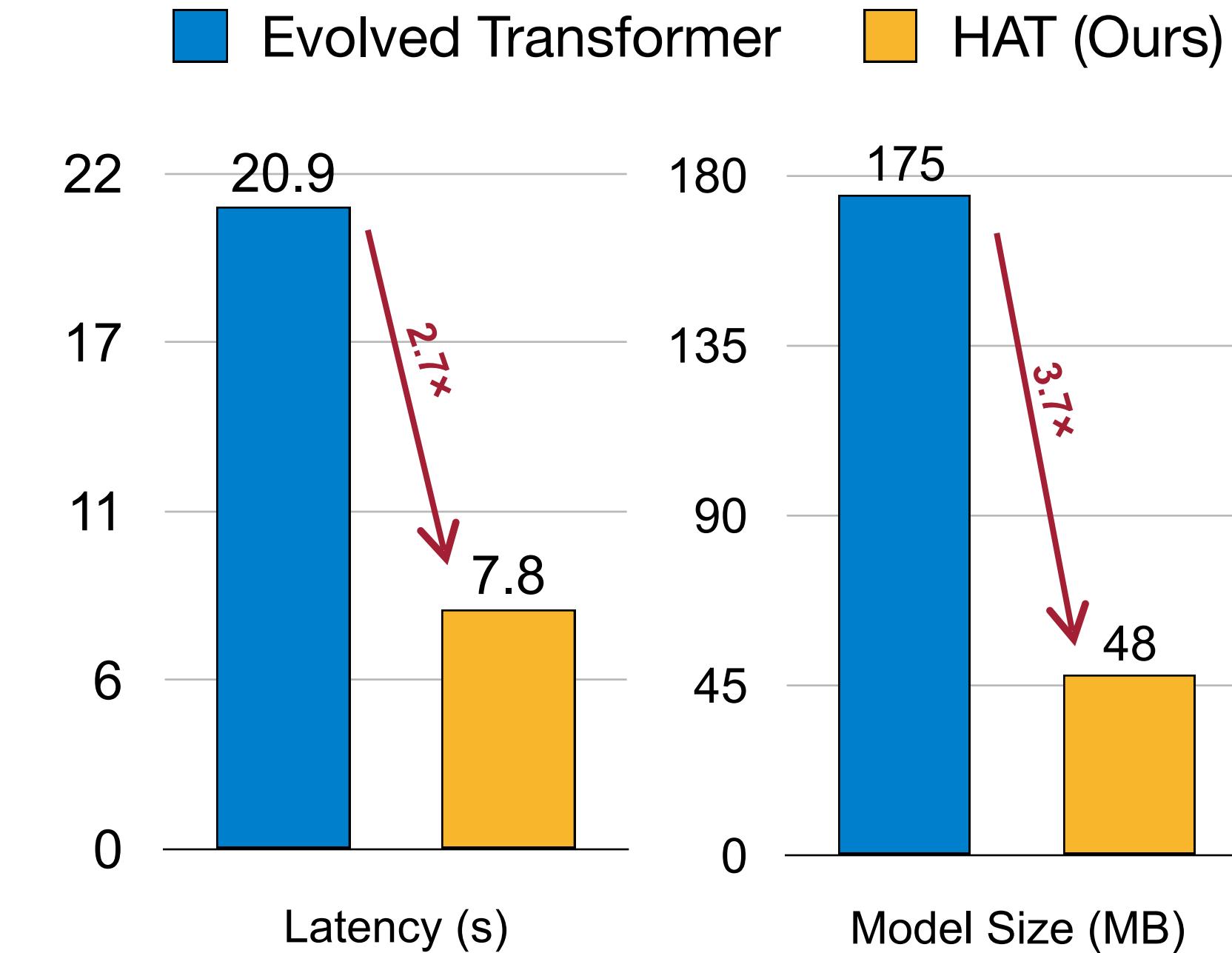
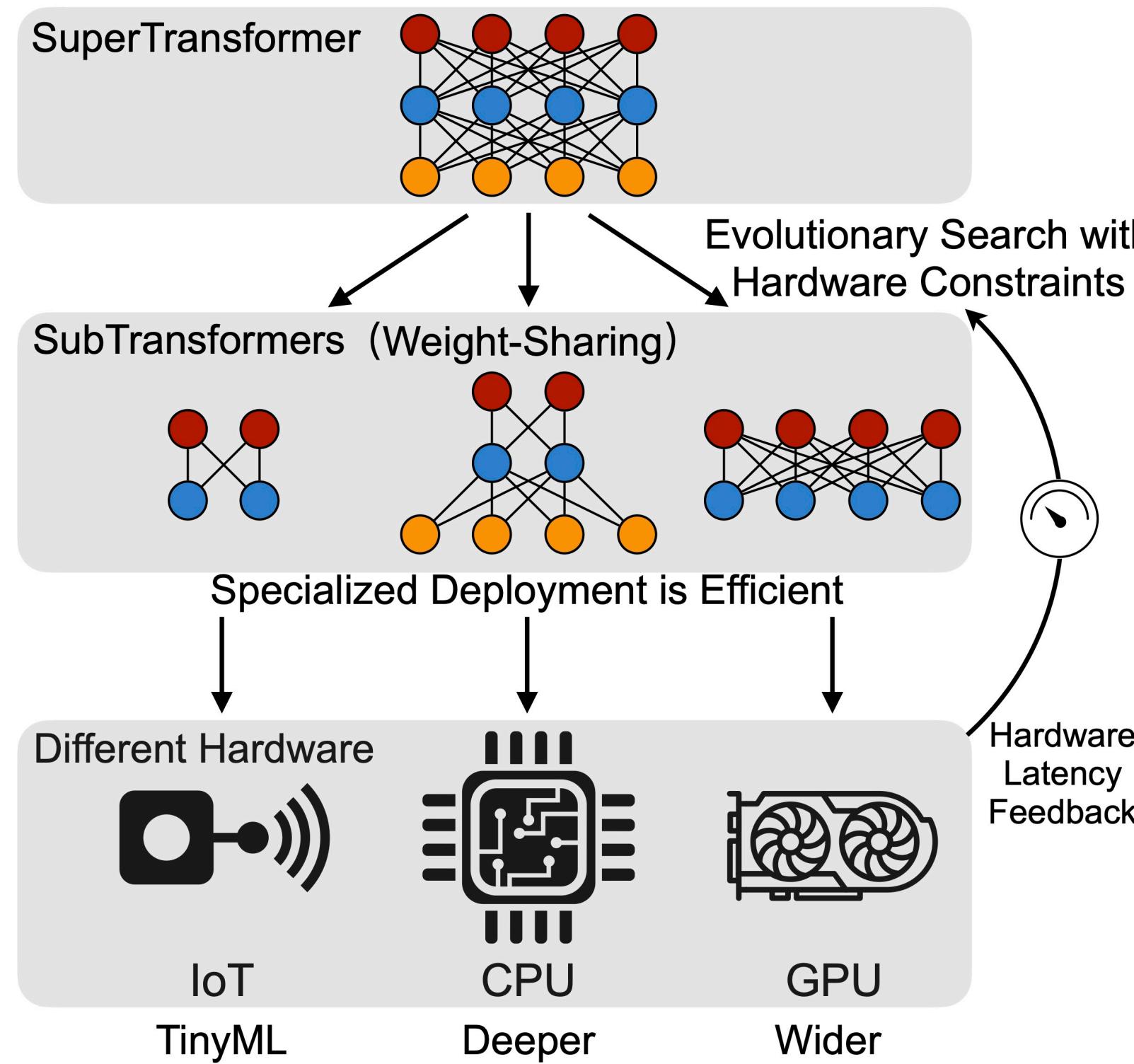
- **Last Lecture:**
  - Primitive operations
  - Classic building blocks
  - **Introduction to neural architecture search (NAS)**
    - What is NAS?
    - Search space
    - Design the search space
    - Search strategy
- **This Lecture:**
  - **Efficient and hardware-aware NAS**
    - Accuracy estimation strategy
    - Zero-shot NAS
    - Hardware-aware NAS
    - Neural-hardware architecture co-search
  - **NAS applications**
    - NLP, GAN, point cloud, pose, etc



# Once-for-All for Transformers and NLP

HAT

## Once-for-all Transformer

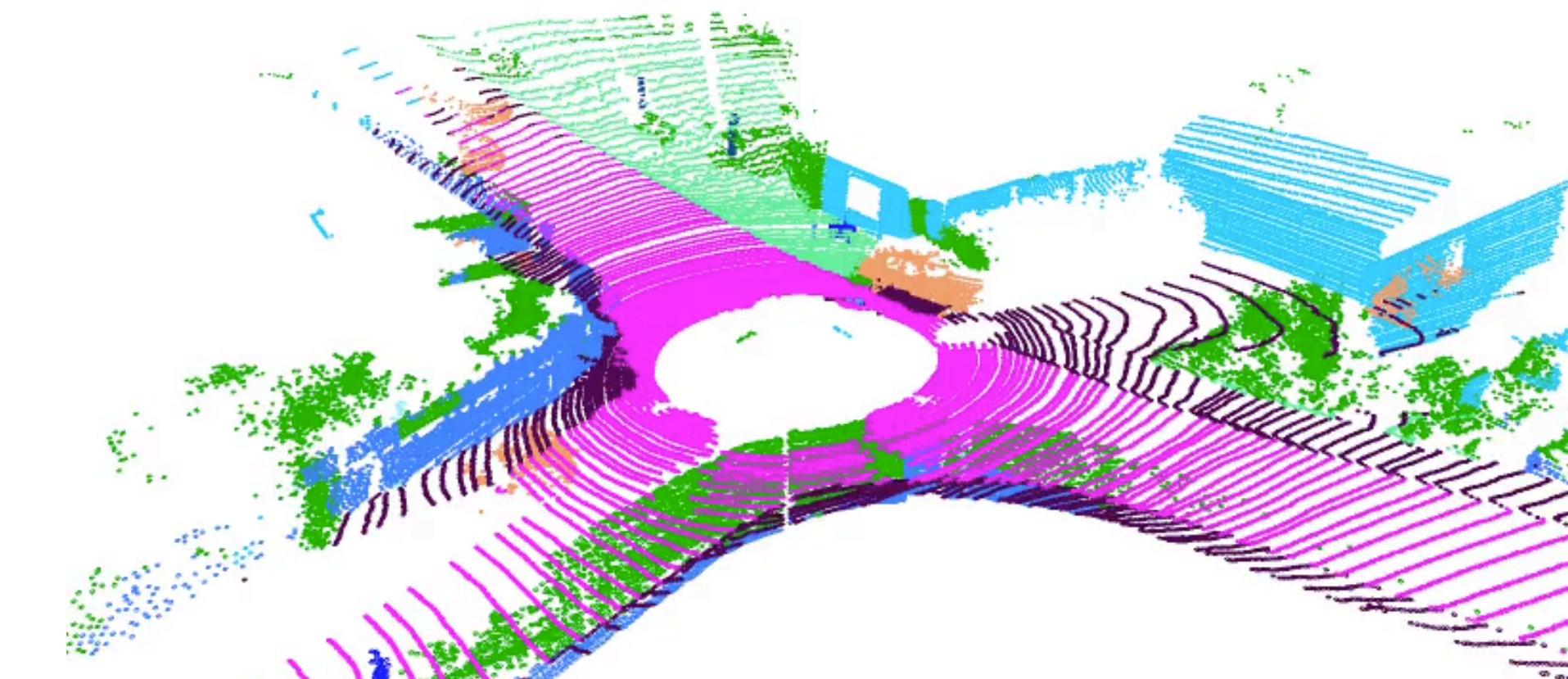
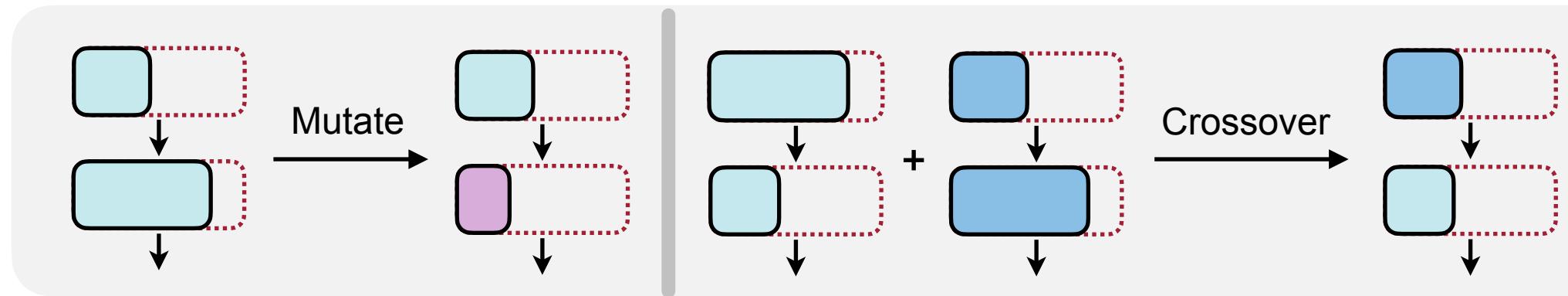
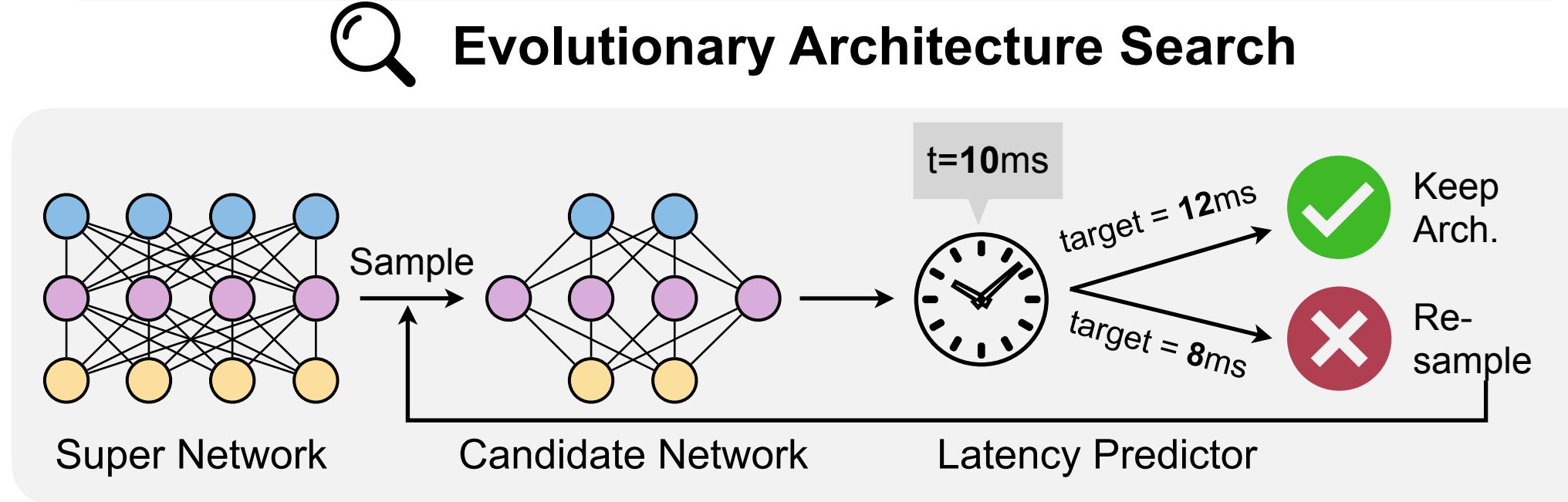
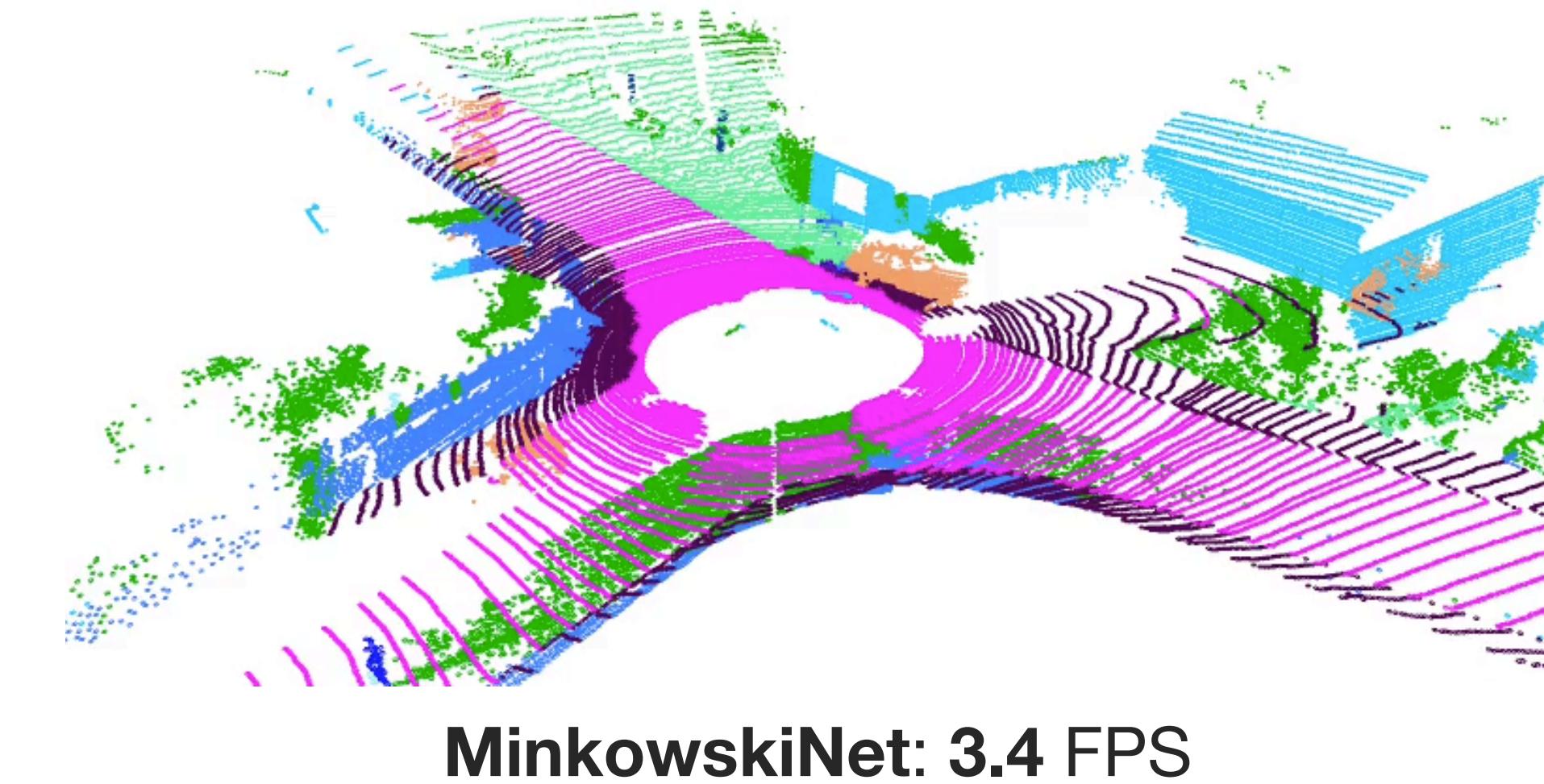
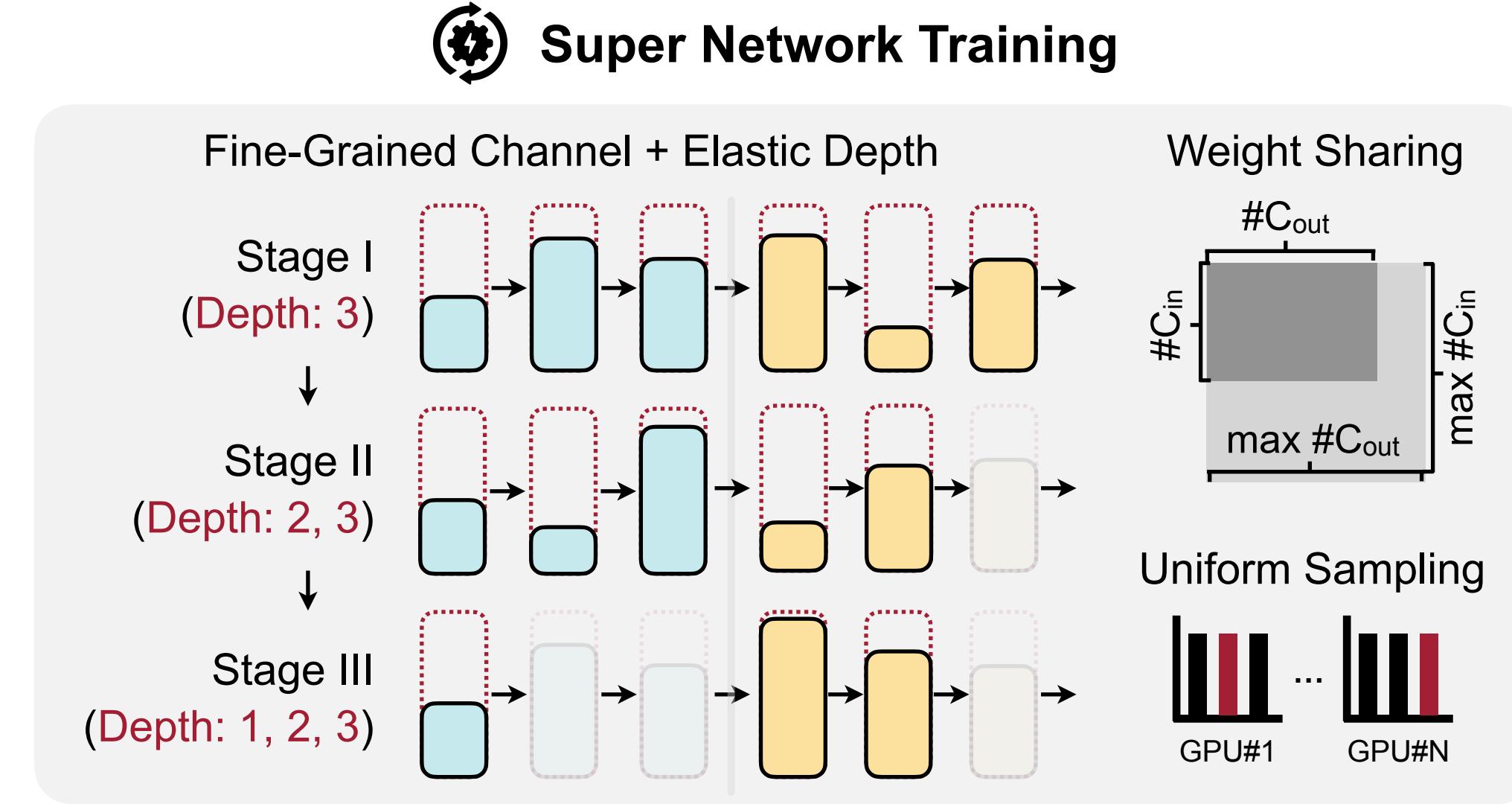


- For WMT'14 En-Fr running on Raspberry Pi, HAT is **2.7x** faster, **3.7x** smaller model size, **3.2x** fewer FLOPs, and **10,148x** less search cost, even achieves **0.1 higher BLEU score**.

HAT: Hardware-Aware Transformers for Efficient Natural Language Processing [Wang et al., ACL 2020]

# Once-for-All for Point Cloud Understanding

## SPVNAS

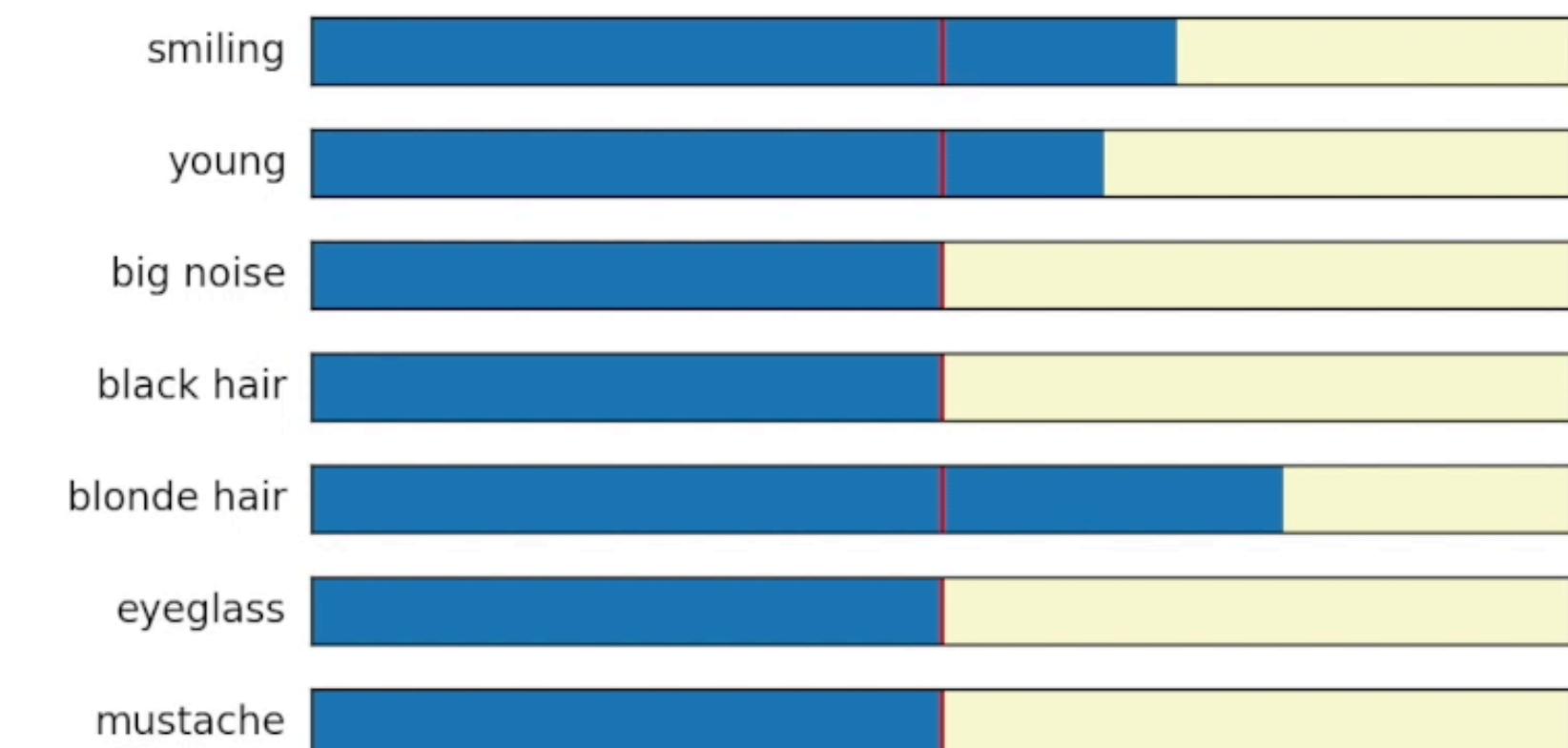
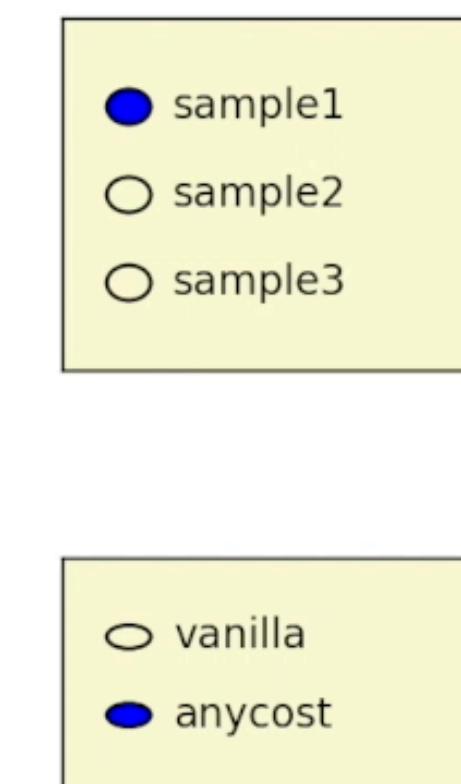
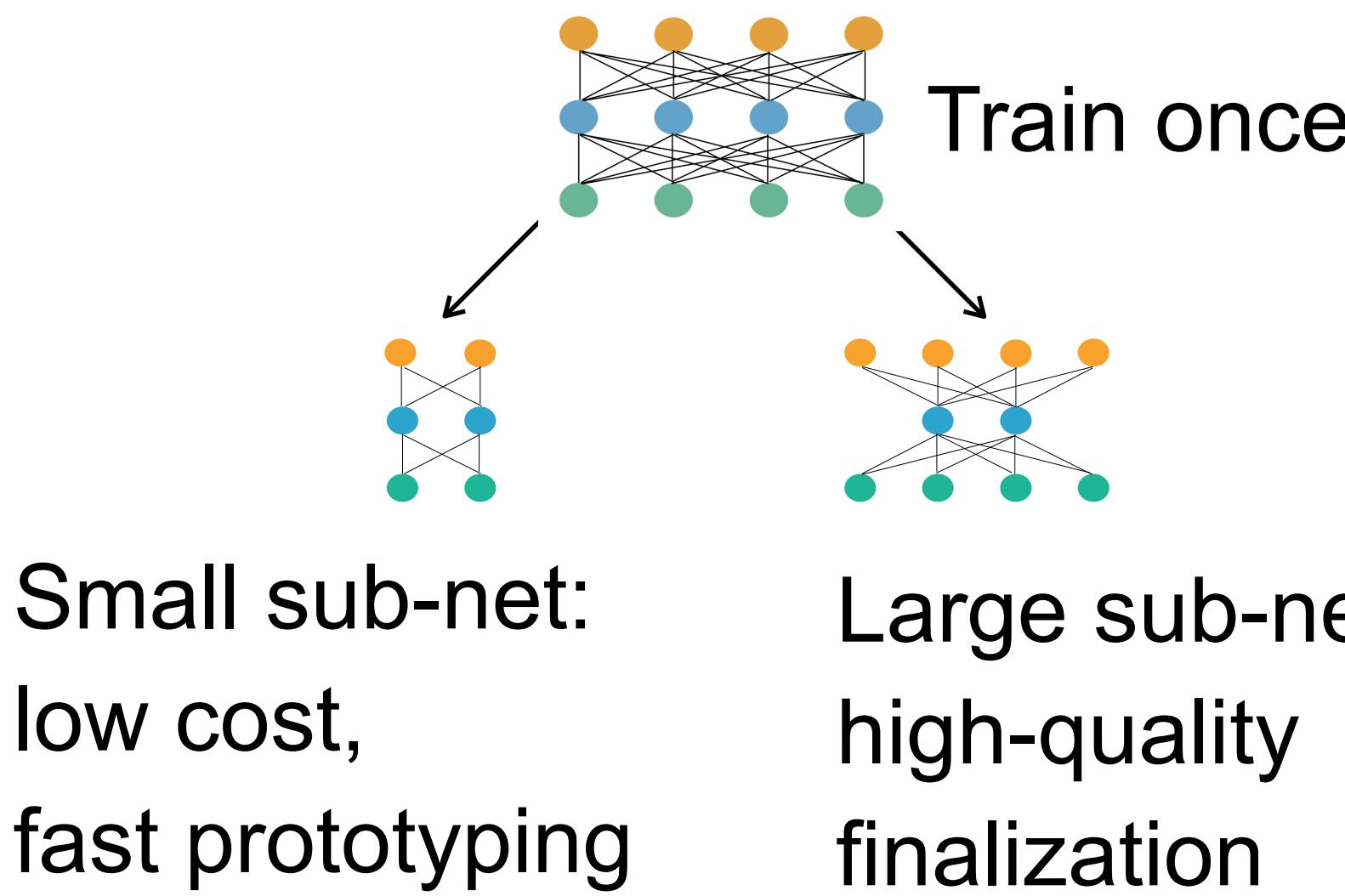
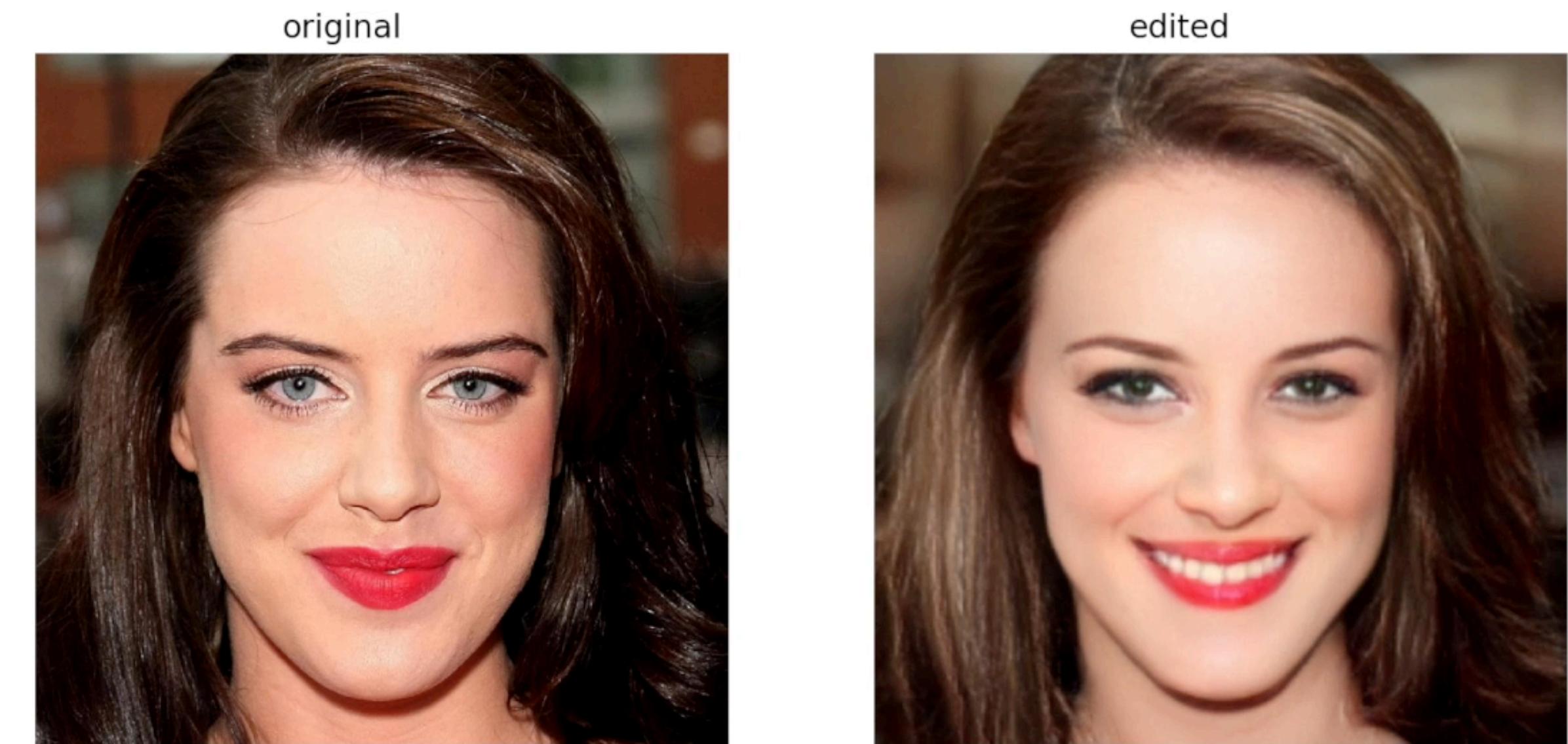


Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution [Tang et al., ECCV 2020]

# Once-for-All for GAN

## Anycost GAN

- Generative Adversarial Network (GAN) is computationally heavy and slow
- Difficult for interactive photo editing on mobile device (iPad)
- Anycost GAN with once-for-all network:



\* Status: done (0.40s)

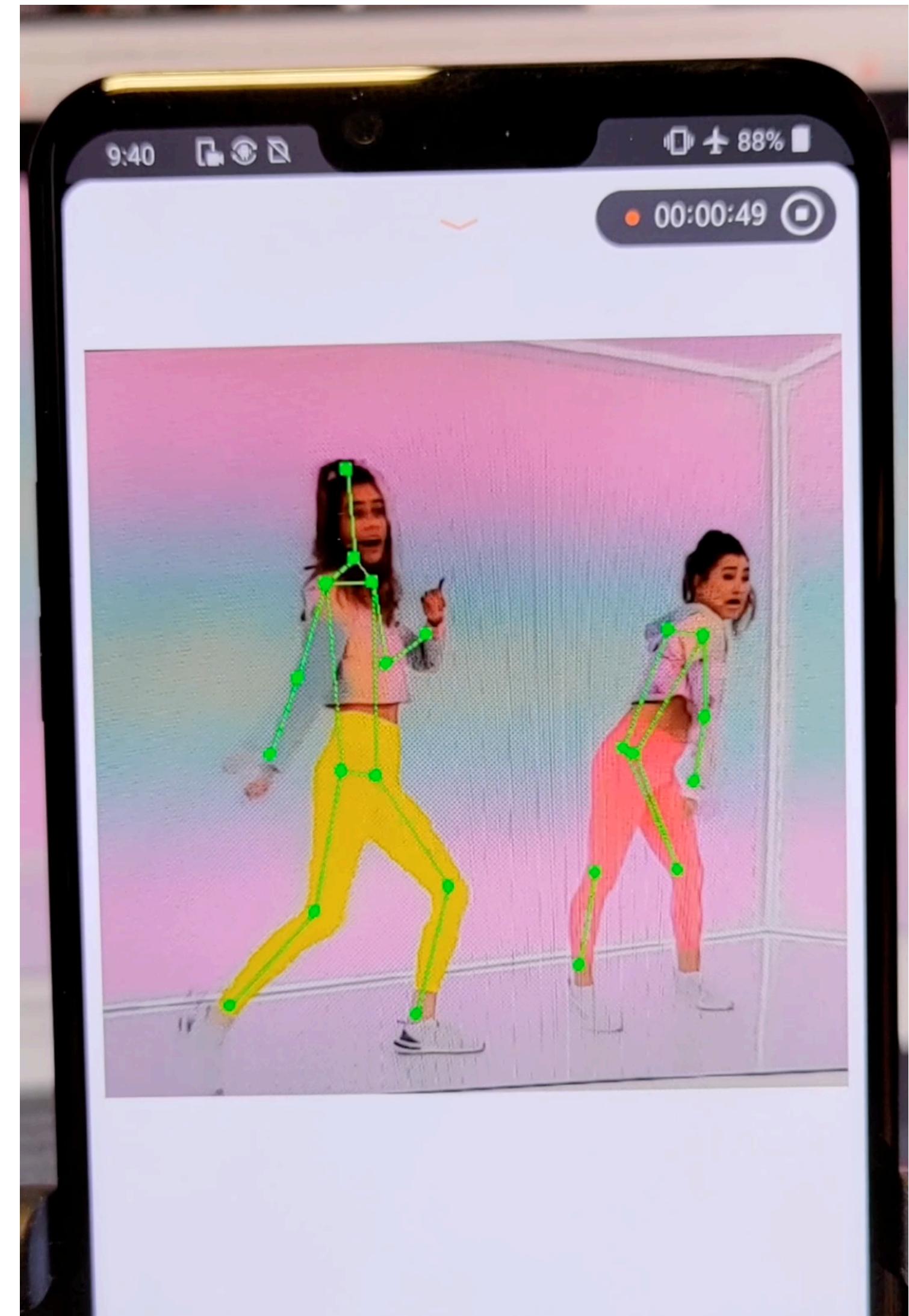
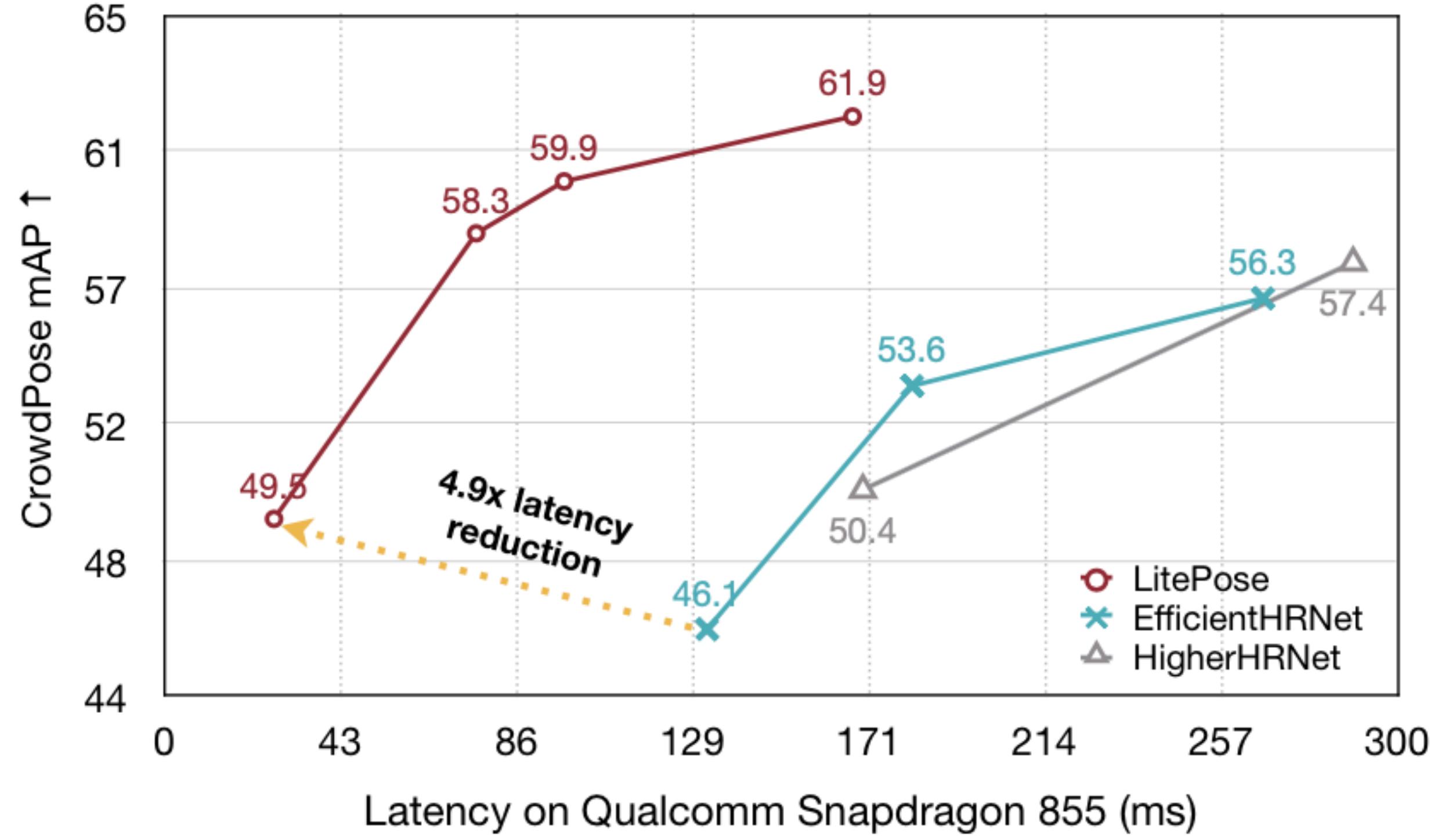
Finalize

Reset

Anycost GANs for Interactive Image Synthesis and Editing [Lin et al., CVPR 2021]

# Once-for-All for Pose Estimation

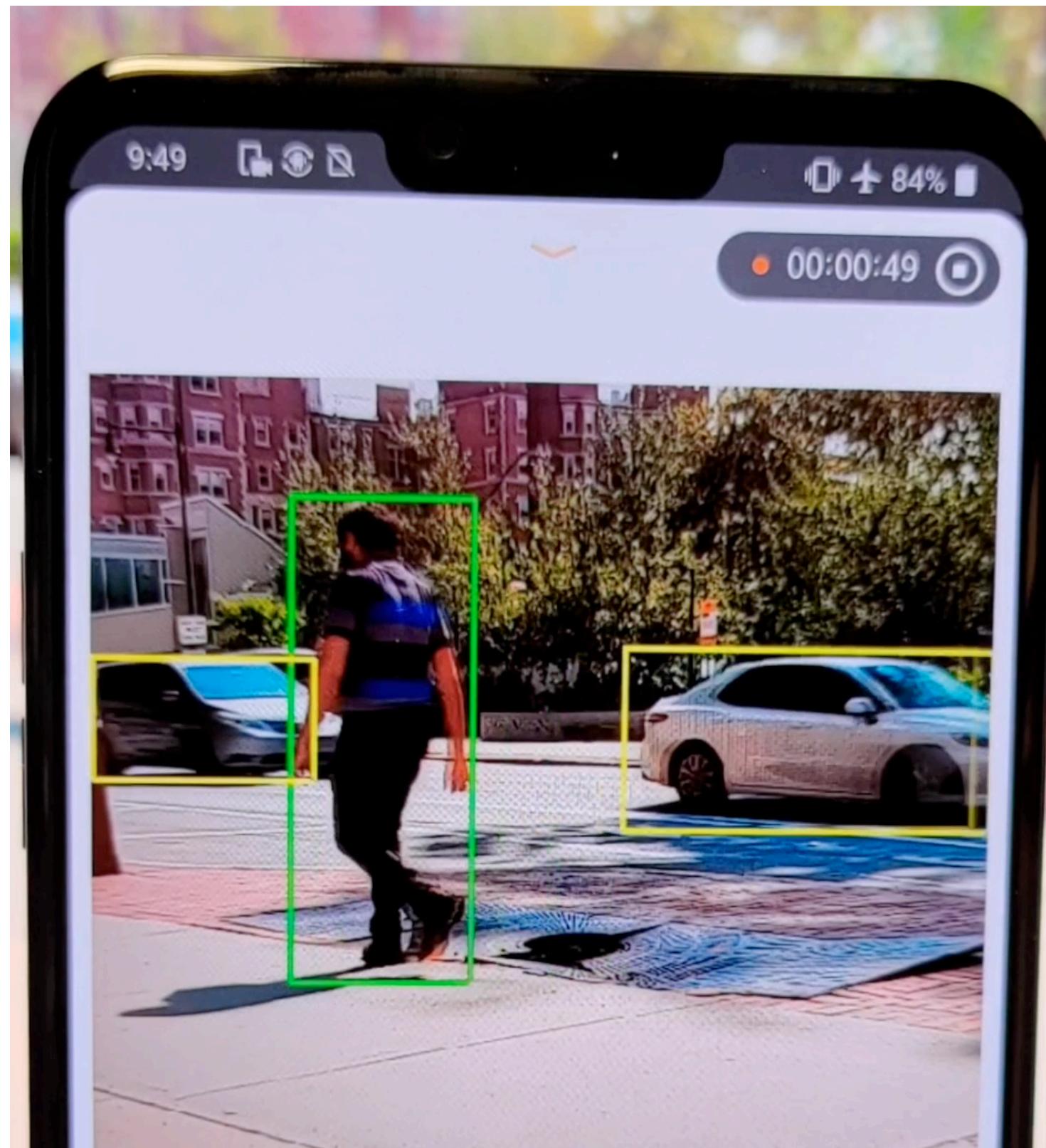
## LitePose



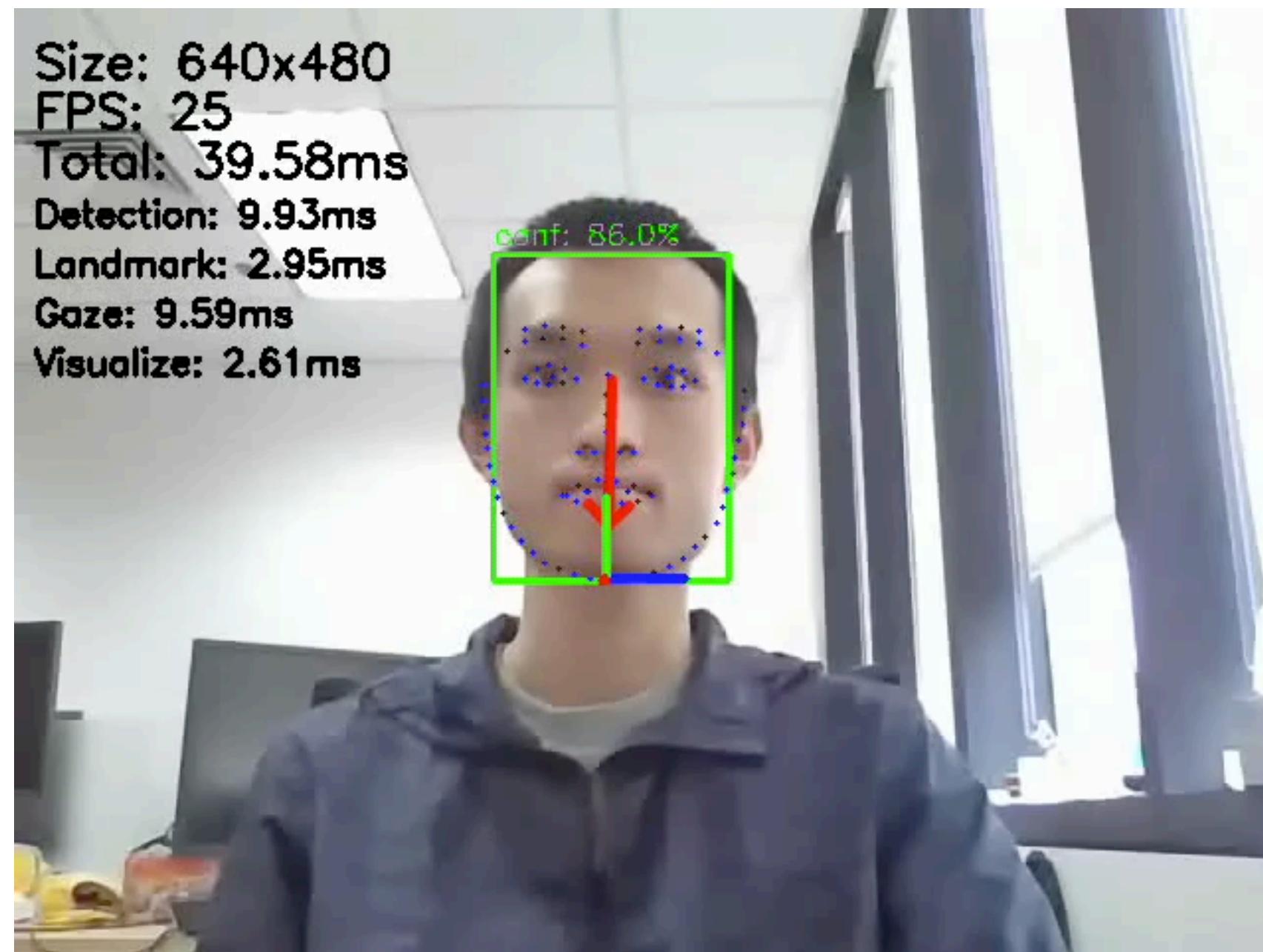
On-device pose estimation by:  
Designing new design space + HW-aware NAS

Lite Pose: Efficient Architecture Design for 2D Human Pose Estimation [Wang et al., CVPR 2022]

# Once-for-All Designs Light-weight Model, Bring AI to Mobile Devices



on-device car/person detection



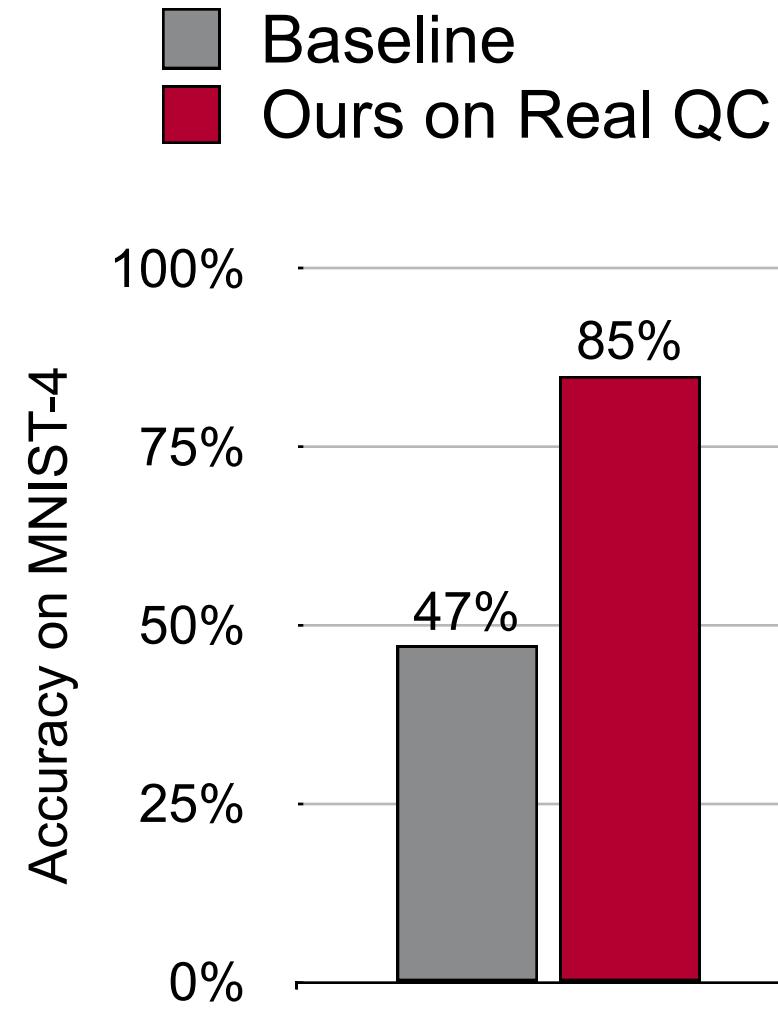
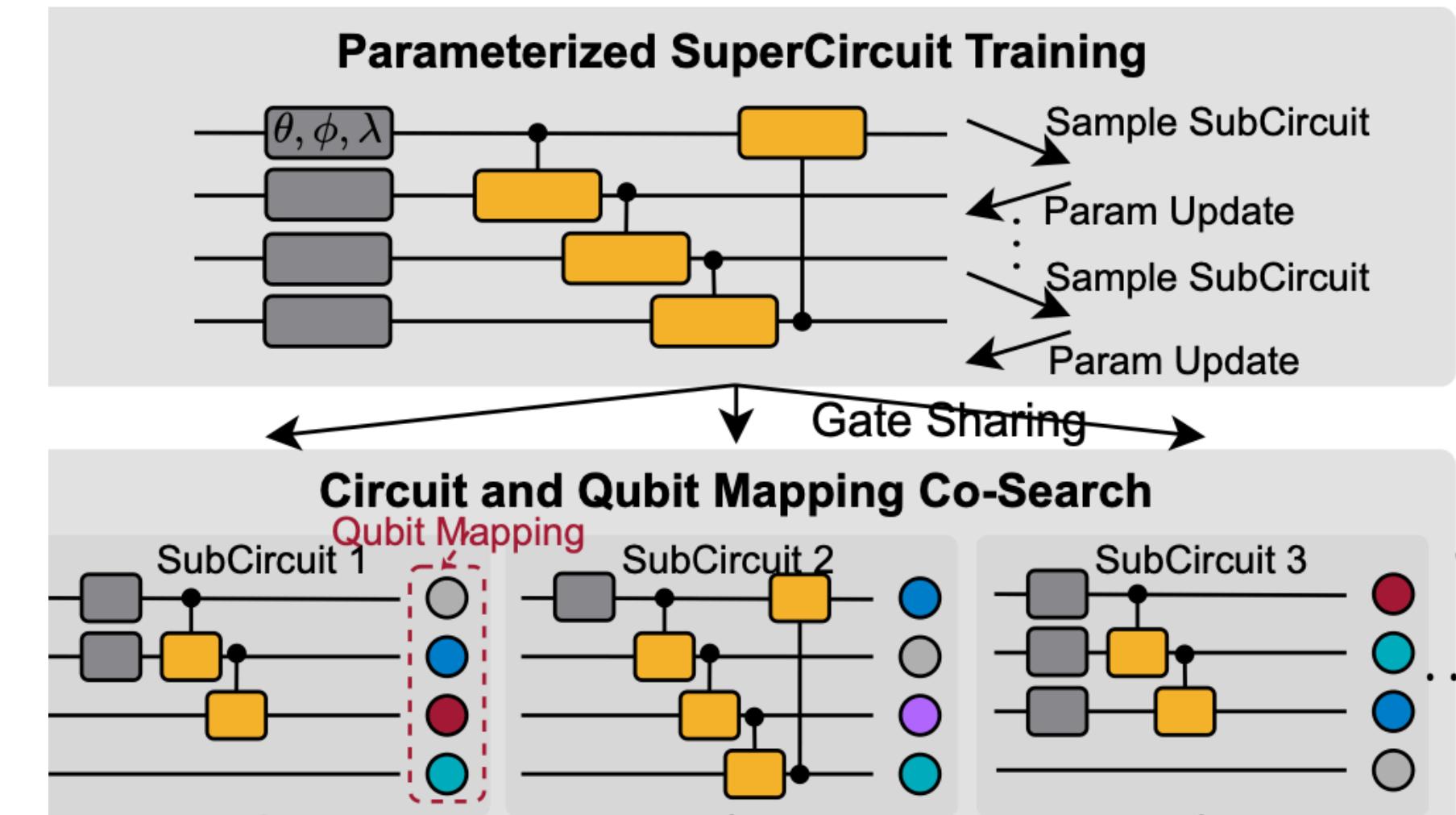
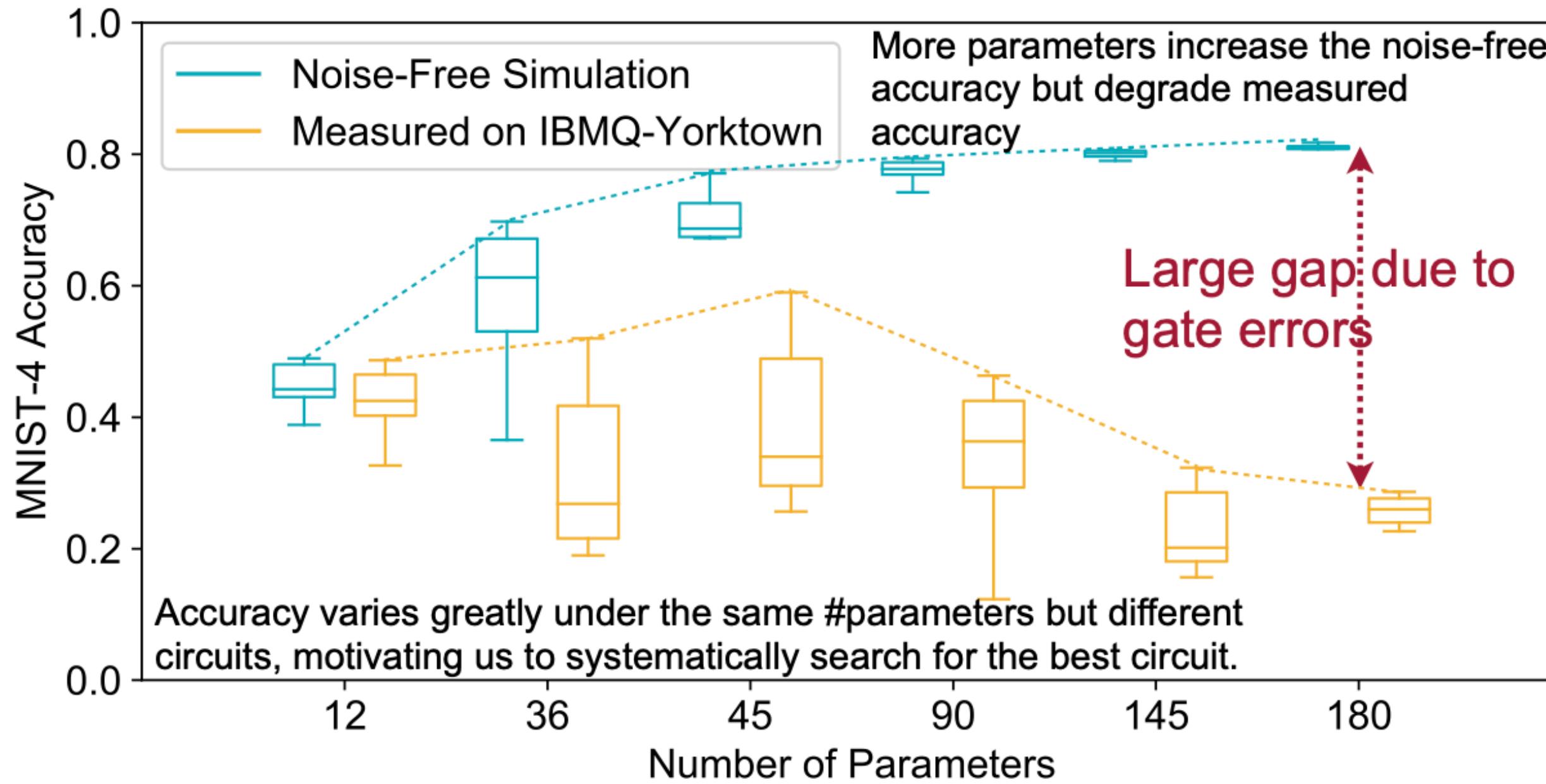
on-device gaze estimation  
(Rasp Pi)



on-device segmentation

Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai et al., ICLR 2020]

# Once-for-All for Quantum AI



- Quantum noise is the bottleneck for quantum neural nets, accuracy: 87%  $\rightarrow$  47%
- QuantumNAS: Noise-Aware Search** for robust circuit architecture by training a “super circuit”, then search a “sub-circuit” that is robust to noise; **prune** small-magnitude quantum gates
- TorchQuantum**: open-source library used by UT Austin, Cornell, Notre Dame, USC, U Chicago, Tsinghua U

QuantumNAS: Noise-Adaptive Search for Robust Quantum Circuits [Wang et al., HPCA 2022]

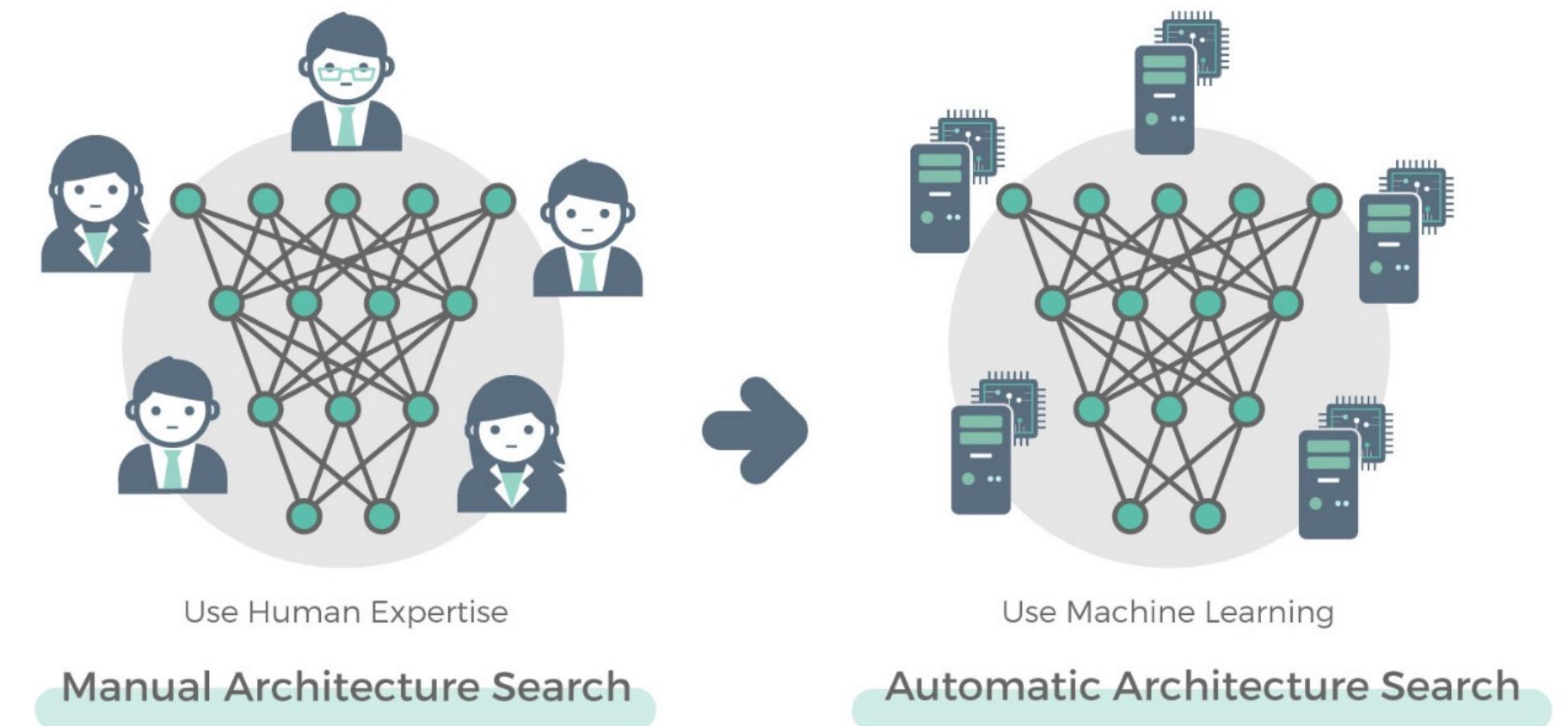
QuantumNAT: Quantum Noise-Aware Training with Noise Injection, Quantization and Normalization [Wang et al., DAC 2022]

QOC: Quantum On-Chip Training with Parameter Shift and Gradient Pruning [Wang et al., DAC 2022]

[qmlsys.mit.edu](http://qmlsys.mit.edu)

# Summary of Today's Lecture

- Performance estimation strategy in NAS
- Hardware-aware NAS
- Zero-shot NAS
- Neural-hardware architecture search
- NAS applications
- We will cover in the next lecture:
  - Knowledge Distillation



# References

1. Neural Architecture Search with Reinforcement Learning [Zoph *et al.*, ICLR 2017]
2. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware [Cai *et al.*, ICLR 2019]
3. Net2Net: Accelerating Learning via Knowledge Transfer [Chen *et al.*, ICLR 2016]
4. Efficient Architecture Search by Network Transformation [Cai *et al.*, AAAI 2018]
5. Simple And Efficient Architecture Search for Convolutional Neural Networks [Elsken *et al.*, arXiv 2017]
6. Path-Level Network Transformation for Efficient Architecture Search [Cai *et al.*, ICML 2018]
7. SMASH: One-Shot Model Architecture Search through HyperNetworks [Brock *et al.*, ICLR 2018]
8. Efficient Neural Architecture Search via Parameter Sharing [Pham *et al.*, ICML 2018]
9. HAT: Hardware-Aware Transformers for Efficient Natural Language Processing [Wang *et al.*, ACL 2020]
10. Mnasnet: Platform-Aware Neural Architecture Search for Mobile [Tan *et al.*, CVPR 2019]
11. Once-for-All: Train One Network and Specialize it for Efficient Deployment [Cai *et al.*, ICLR 2020]
12. SNIP: Single-Shot Network Pruning based on Connection Sensitivity [Lee *et al.*, ICLR 2019]

# References

- 13. Picking Winning Tickets before Training by Preserving Gradient Flow [Wang et al., ICLR 2020]
- 14. Pruning Neural Networks without Any Data by Iteratively Conserving Synaptic Flow [Tanaka et al., NeurIPS 2020]
- 15. GradSign: Model Performance Inference with Theoretical Insights [Zhang and Jia, ICLR 2022]
- 16. Zen-NAS: A Zero-Shot NAS for High-Performance Image Recognition [Lin et al., ICCV 2021]
- 17. MAE-DET: Maximum Entropy Principle for Efficient Object Detection [Sun et al., ICML 2022]
- 18. NAAS: Neural Accelerator Architecture Search [Lin et al., DAC 2021]
- 19. Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution [Tang et al., ECCV 2020]
- 20. Anycost GANs for Interactive Image Synthesis and Editing [Lin et al., CVPR 2021]
- 21. Lite Pose: Efficient Architecture Design for 2D Human Pose Estimation [Wang et al., CVPR 2022]
- 22. QuantumNAS: Noise-Adaptive Search for Robust Quantum Circuits [Wang et al., HPCA 2022]

# References

- 23. QuantumNAT: Quantum Noise-Aware Training with Noise Injection, Quantization and Normalization [Wang et al., DAC 2022]
- 24. QOC: Quantum On-Chip Training with Parameter Shift and Gradient Pruning [Wang et al., DAC 2022]