



UNIVERSITY OF  
**TORONTO**



VECTOR  
INSTITUTE

# Efficient DNN Training at Scale: from Algorithms to Hardware

**Gennady Pekhimenko, Assistant Professor**

**EcoSystem Group**

# Systems/Architecture Is a Servant for ML



ML Researcher

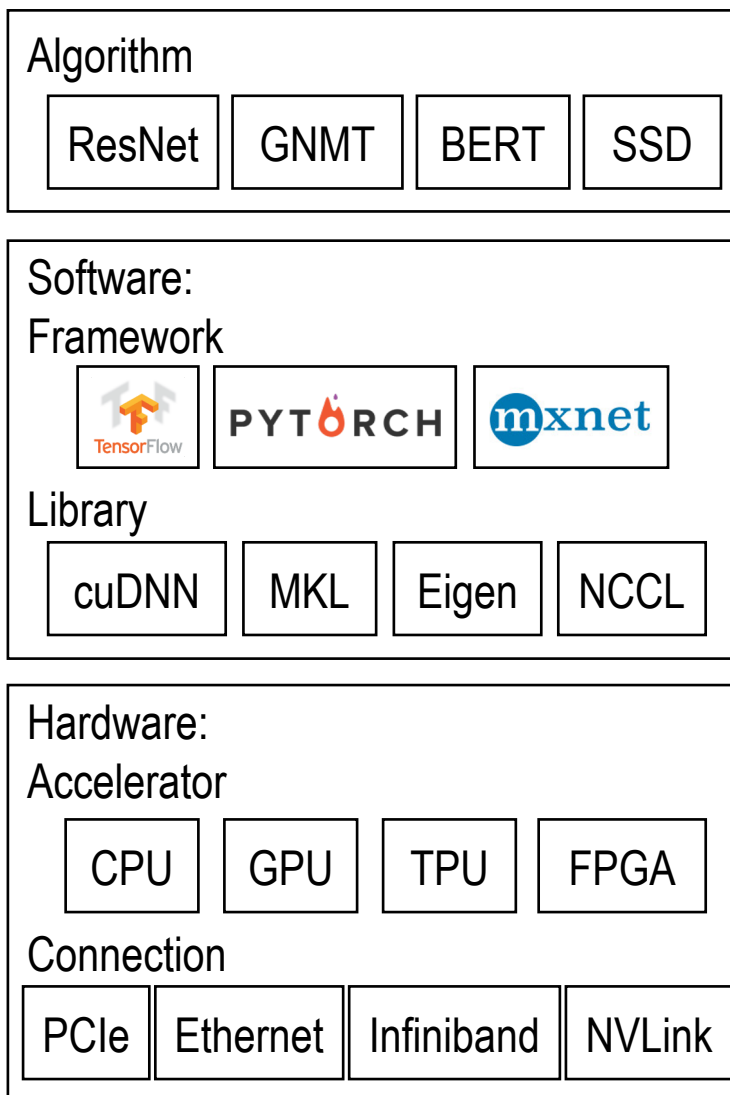


# System-level optimizations for DNNs

Researchers proposed many **system-level optimizations** for DNN computation, however, their performance largely depends on the entire stack

Given a full-stack configuration:

- How much better can we do to improve performance?
- How to identify future opportunities?





# Machine Learning Benchmarking and Analysis

*In collaboration with Project Fiddle (MSR)*





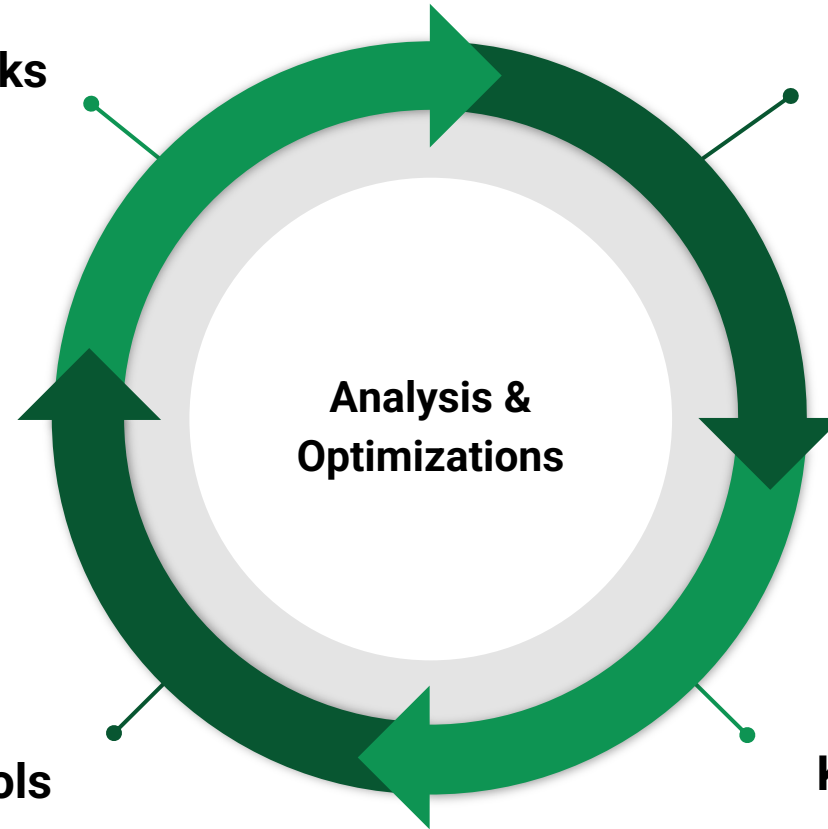
**Performance bottlenecks  
in DNN Training**

**Diverse benchmark suite with  
state-of-the-art models**

**Analysis &  
Optimizations**

**Key performance metrics**

**Tools**



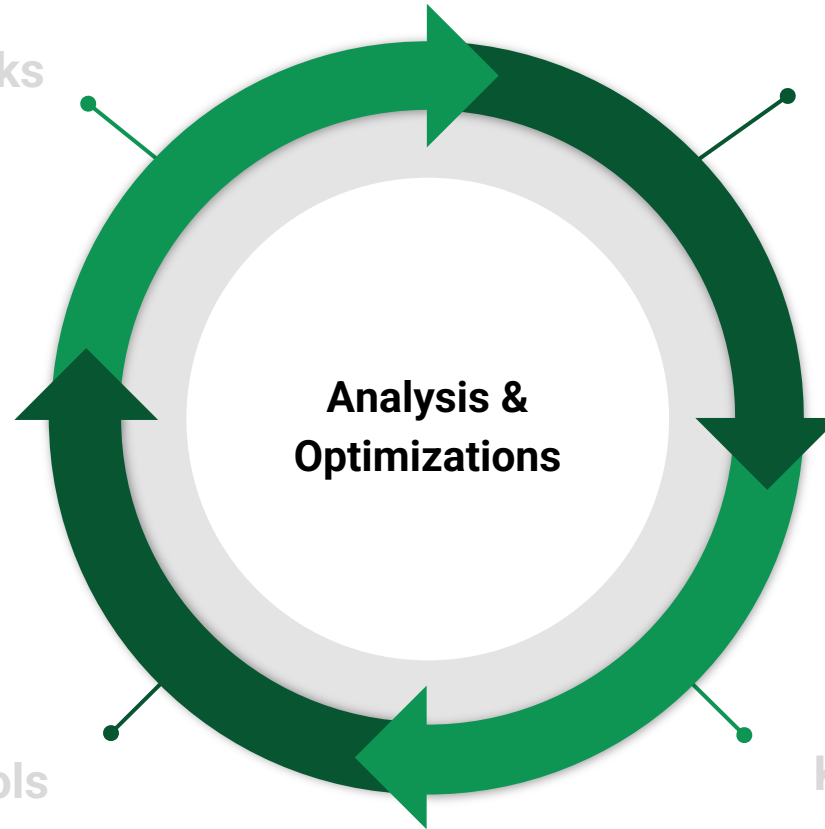
Performance bottlenecks  
in DNN Training

**Diverse benchmark suite with  
state-of-the-art models**

**Analysis &  
Optimizations**

Tools

Key performance metrics






# Training **B**enchmarks for **DNNs** (**TBD**), Jan. 2018

Applications	Models	Dataset	# of layers	Dominant layer	Maintainer
Image Classification	ResNet-50 $T,M,C$ Inception-v3 $T,M,C$	ImageNet	50 (152 max) 42	CONV	Hongyu Zhu
Machine Translation	Seq2Seq $T,M$ Transformer $T,M$	IWSLT15	5 12	LSTM Attention	Bojian Zheng Andrew Pelegris
Object Detection	Faster RCNN $T,M$ Mask RCNN $P$	Pascal VOC	101	CONV	Hongyu Zhu Zilun Zhang
Speech Recognition	Deep Speech 2 $P,M$	LibriSpeech	7 (9 max)	RNN	Kuei-Fang Hsueh Jiahuang Lin
Recommendation System	NCF $P$	MovieLens	4	GMF, MLP	Izaak Niksan
Adversarial Network	WGAN $T$	Downsampled ImageNet	14+14	CONV	Andrew Pelegris
Reinforcement Learning	A3C $T,M$	Atari 2600	4	CONV	Mohamed Akrouf

(Footnotes indicate available implementation:  $T$  for  ,  $M$  for  ,  $C$  for  ,  $P$  for )

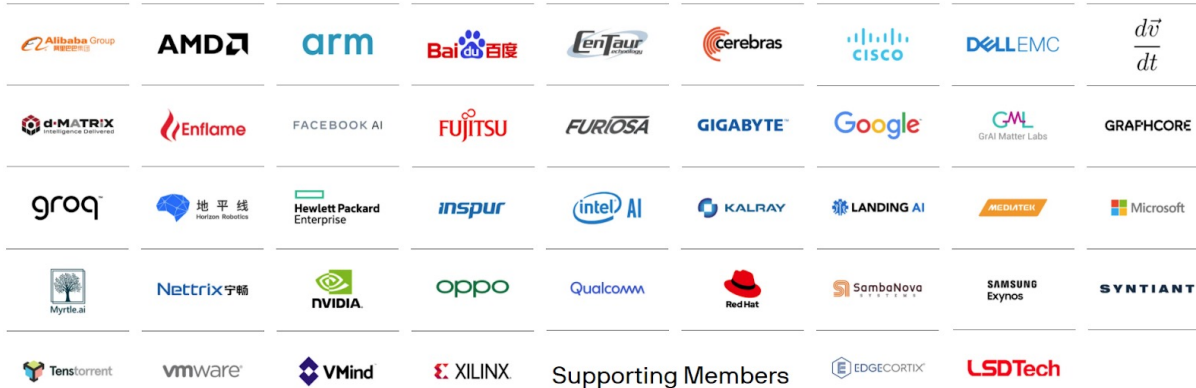
# TBD Benchmark Suite, Aug. 2020 update

Applications	Models	Dataset	# of layers	Dominant layer	Maintainer
Image Classification	ResNet-50 $T,M$ Inception-v3 $T,M$	ImageNet	50 (152 max) 42	CONV	<i>Xin Li</i>
Machine Translation	Seq2Seq $T,M$ Transformer $T,M$	IWSLT16	5 12	LSTM Attention	<i>Yu Bo Gao</i> <i>Yu Bo Gao</i>
Object Detection	Mask RCNN $T,P$ EfficientDet $T,P$	COCO	101	CONV	<i>Yu Bo Gao</i>
Speech Recognition	Deep Speech 2 $P$	LibriSpeech	7 (9 max)	RNN	<i>Cong Wei</i>
Language Modeling	BERT $P$	SQuAD	24	BERT block	<i>Xin Li</i>
Reinforcement Learning	MiniGo $T$		38	CONV	<i>Cong Wei</i>

(Footnotes indicate available implementation:  $T$  for ,  $M$  for ,  $P$  for )

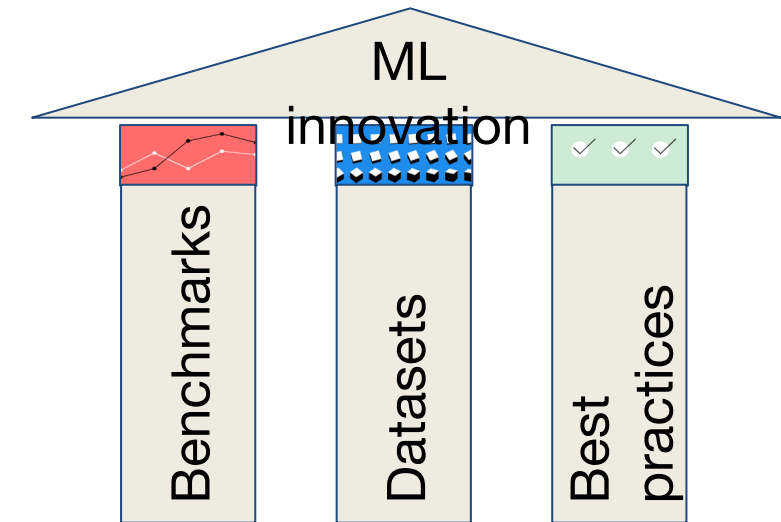
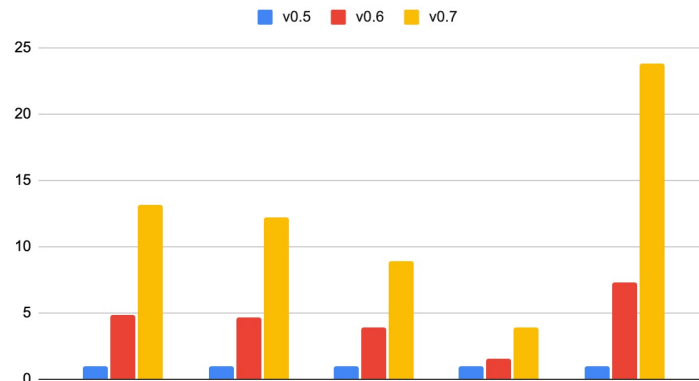
# MLPerf -> MLCommons

## Founding Members



## MLPerf Training, MLSys 2020

MLPerf best result speedup



## MLPerf Inference, ISCA 2020

The v0.7 ([datacenter](#), [edge](#), [mobile](#)) result highlights:

- 23 submitting organizations
- Over 1,200 peer-reviewed results - twice as many as the first round
- More than doubles the number of applications in the suite
- New dedicated set of MLPerf Mobile benchmarks
- Randomized third party audits for rules compliance

Read more in the [press release](#).

Performance bottlenecks  
in DNN Training

Diverse benchmark suite with  
state-of-the-art models

**Analysis &  
Optimizations**

Tools

**Key performance metrics**

# Performance Metrics

- Throughput  
*Number of data samples processed per second*
- Compute Utilization  
*GPU busy time over Elapsed time*
- FP32/FP16/Tensor Core Utilization  
*Average instructions executed per cycle over Maximum instructions per cycle*
- Memory Breakdown  
*Which data structures occupy how much memory*



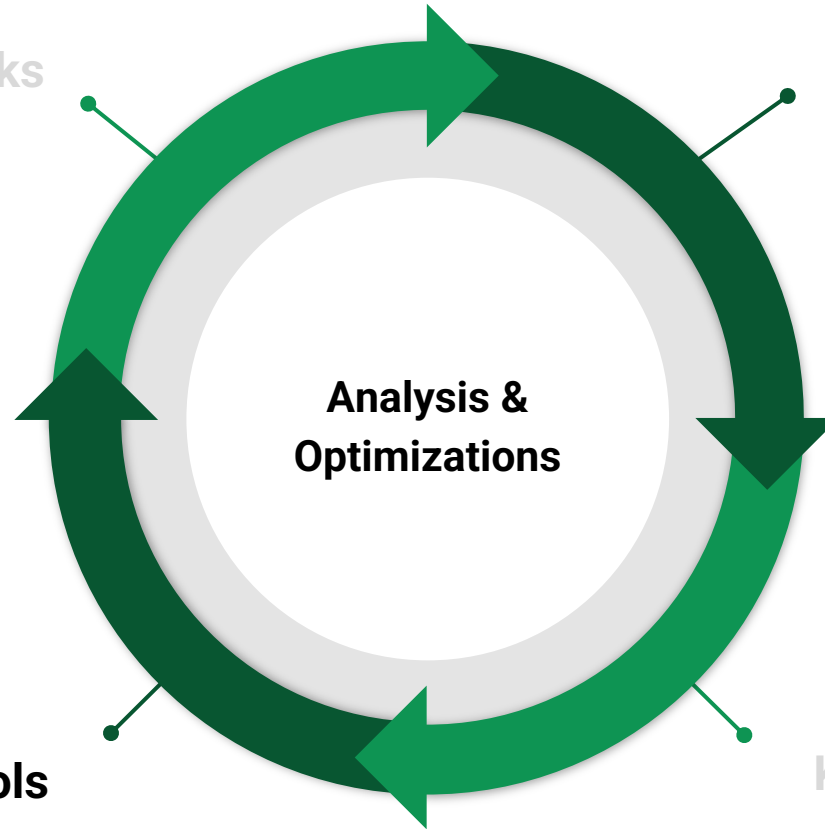
Performance bottlenecks  
in DNN Training

Diverse benchmark suite with  
state-of-the-art models

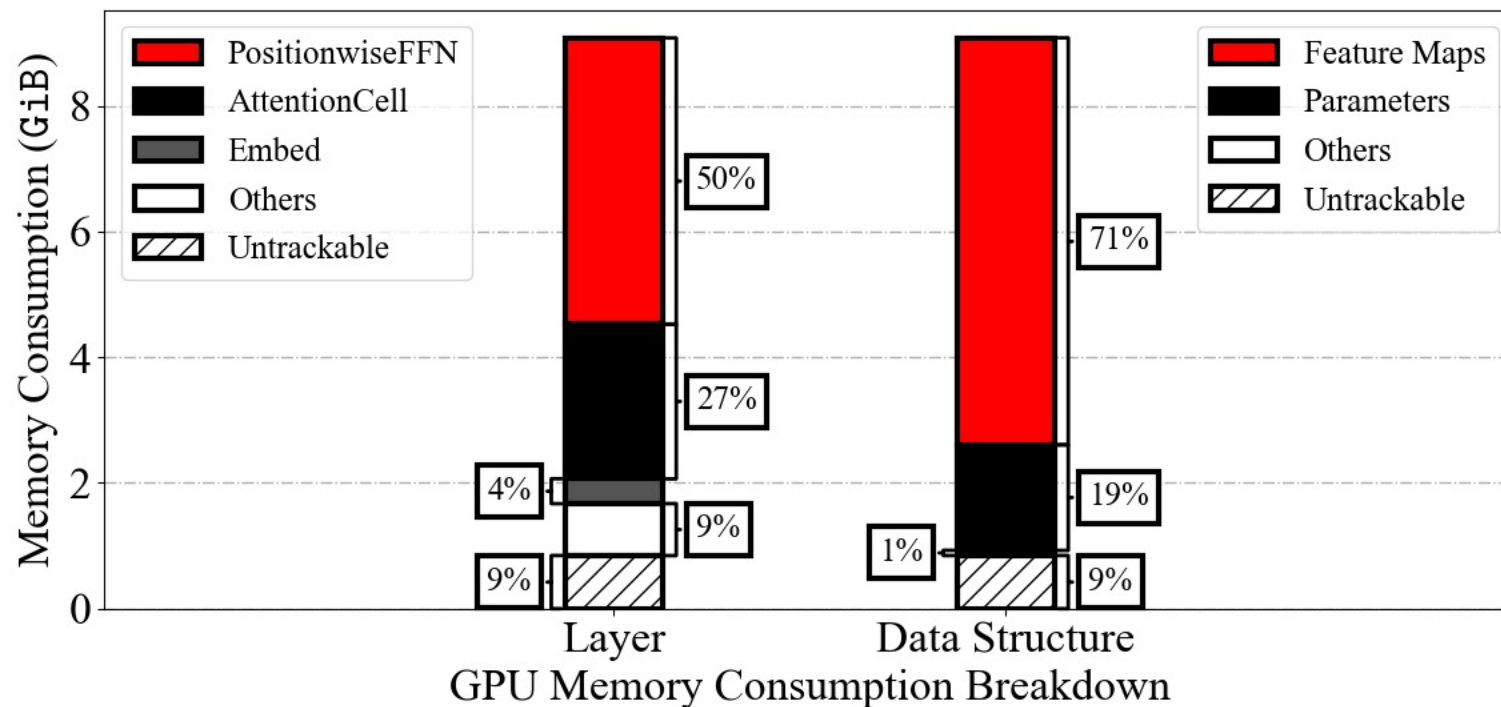
**Analysis &  
Optimizations**

Key performance metrics

**Tools**



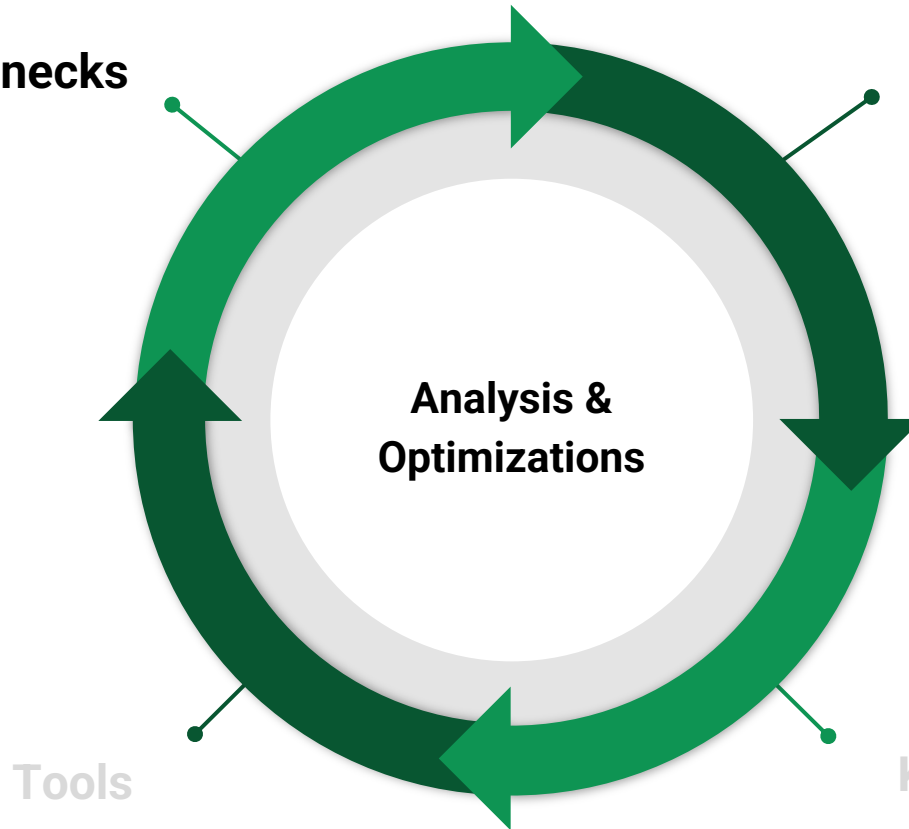
# Memory Profiler (BERT)



**Feature maps are still more important than weights for memory consumption**

**Performance bottlenecks  
in DNN Training**

Diverse benchmark suite with  
state-of-the-art models



**Analysis &  
Optimizations**

Tools

Key performance metrics



# Scaling **Back-P**ropagation by **P**arallel **S**can **A**lgorithm

Shang Wang<sup>1,2</sup>, Yifan Bai<sup>1</sup>, Gennady Pekhimenko<sup>1,2</sup>

1



Computer Science  
UNIVERSITY OF TORONTO

2



VECTOR  
INSTITUTE

MLSys 2020

# Executive Summary

The **back-propagation (BP)** algorithm is **popularly used** in training deep learning (DL) models and **implemented in many** DL frameworks (e.g., PyTorch and TensorFlow).

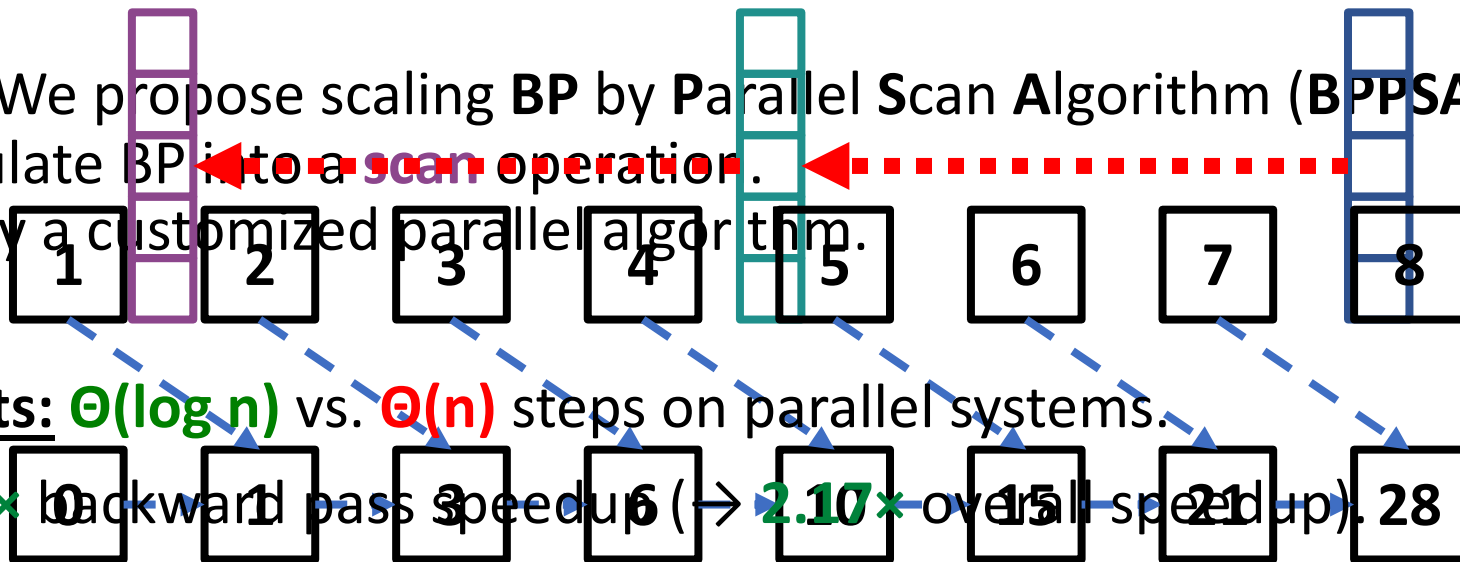
**Problem:** BP imposes a **strong sequential dependency** along layers during the gradient computations.

**Key idea:** We propose scaling BP by **Parallel Scan Algorithm (BPPSA)**:

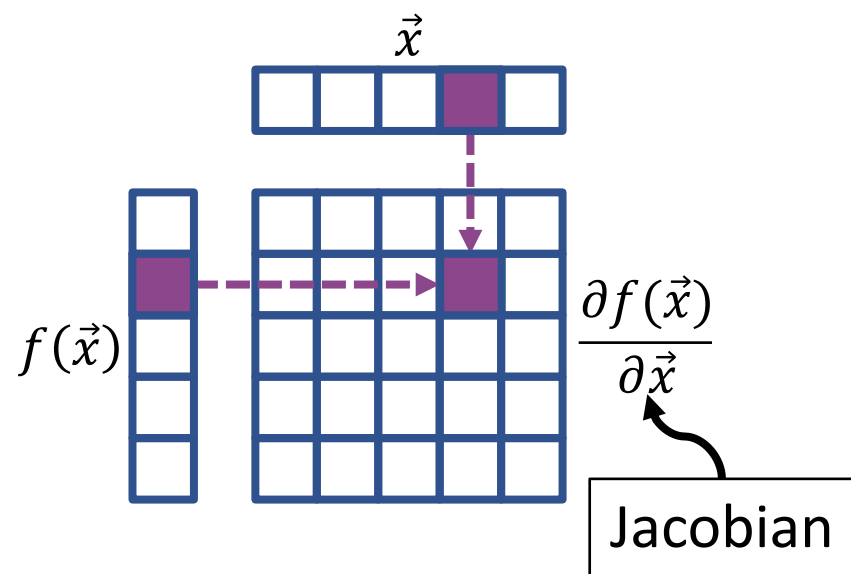
- Reformulate BP into a **scan** operation.
- Scaled by a customized parallel algorithm.

**Key Results:**  $\Theta(\log n)$  vs.  $\Theta(n)$  steps on parallel systems.

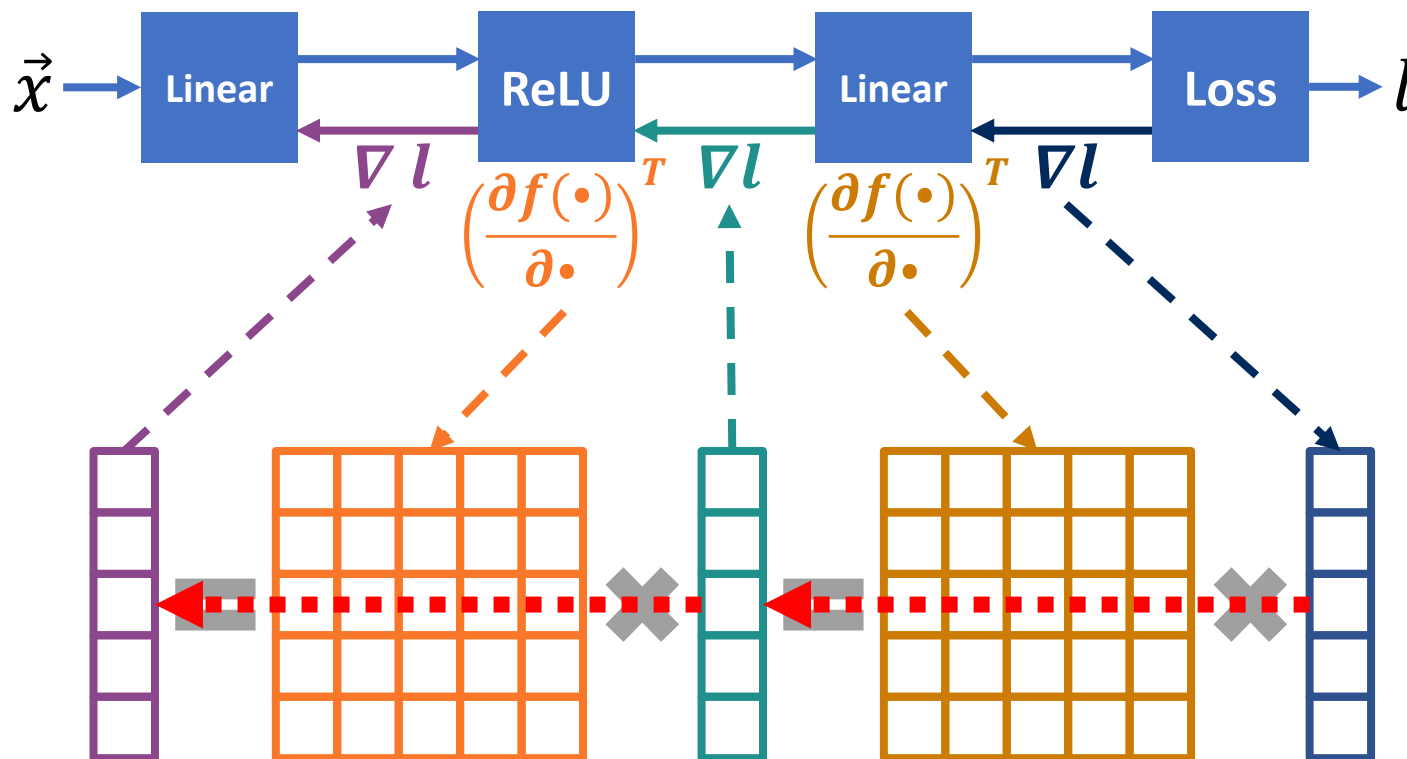
Up to **108x** backward pass speedup ( $\rightarrow$  **2.10x** overall speedup).



# BP's Strong Sequential Dependency



$$\nabla_{\vec{x}} l = \left( \frac{\partial f(\vec{x})}{\partial \vec{x}} \right)^T \nabla_{f(\vec{x})} l$$



**Strong Sequential Dependency** along layers.

# Rethinking BP from an Algorithm Perspective

- Problems with strong sequential dependency were (80'), but in a much simpler context.



- We propose scaling **Back-Propagation** by **Parallel Scan Algorithm (BPPSA)**:
  - Reformulate BP as a **scan** operation.
  - Scale BP by a **customized Blelloch Scan** algorithm.
  - Leverage **sparsity** in the Jacobians.

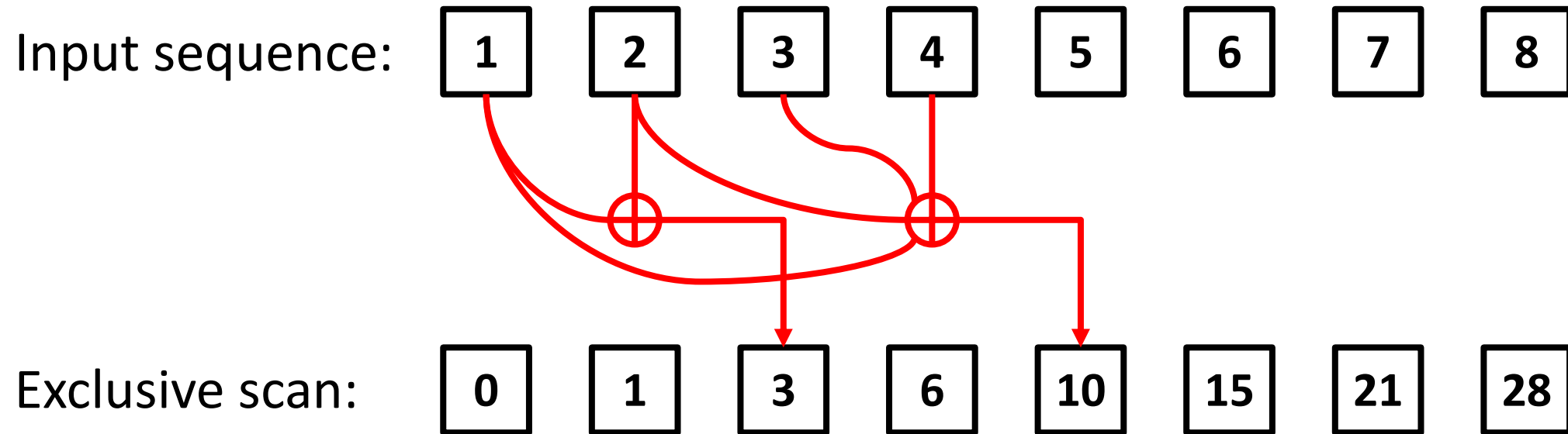




# What is a Scan<sup>1</sup> Operation?

Binary, associative operator:  $+$

Identity:  $0$



Compute partial reductions at each step of the sequence.

<sup>1</sup>Blelloch, Guy E. "Prefix sums and their applications". Technical Report (1990)

# Linear Scan

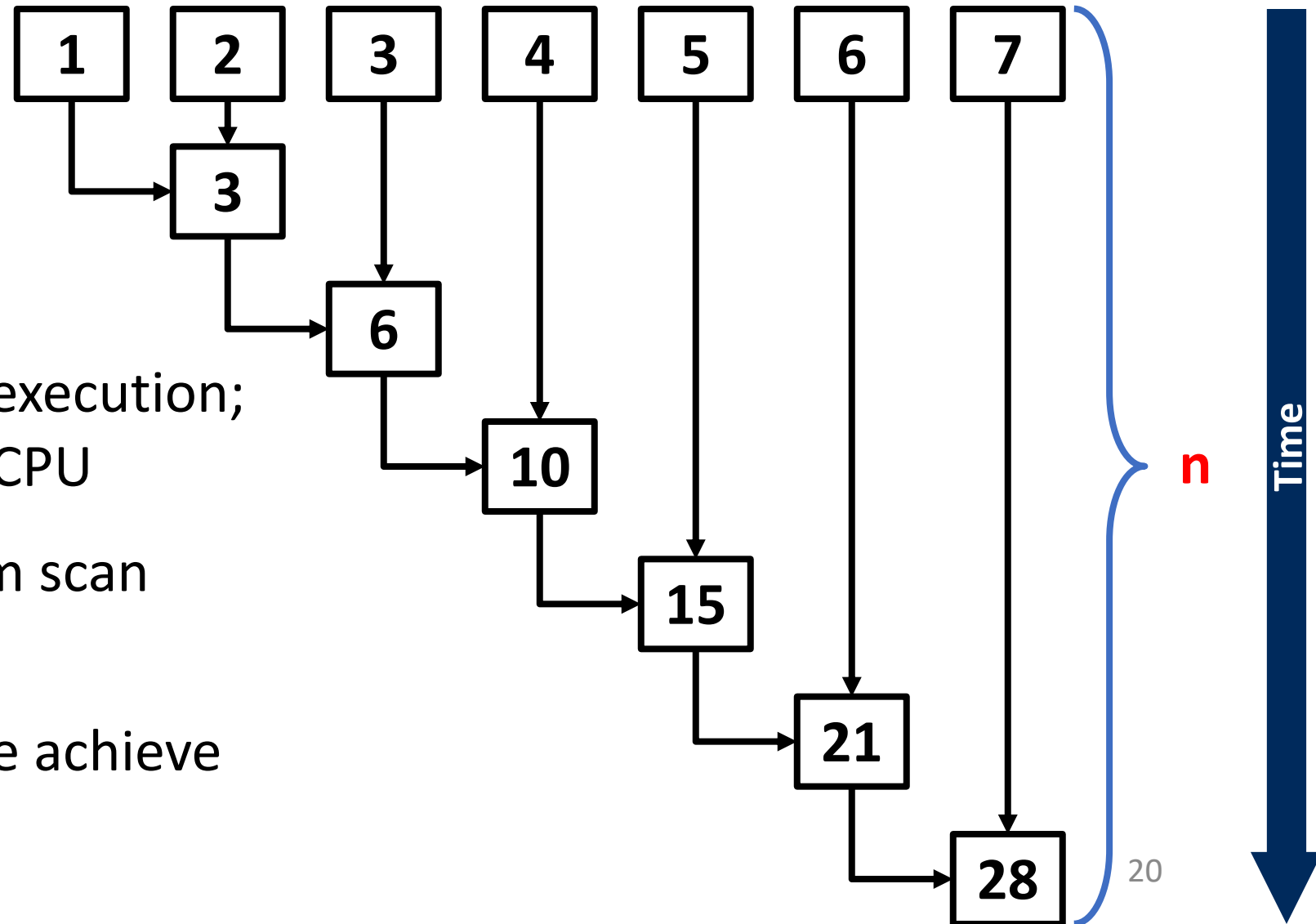
Step: executing the operator once.

## Number of Elements (n)

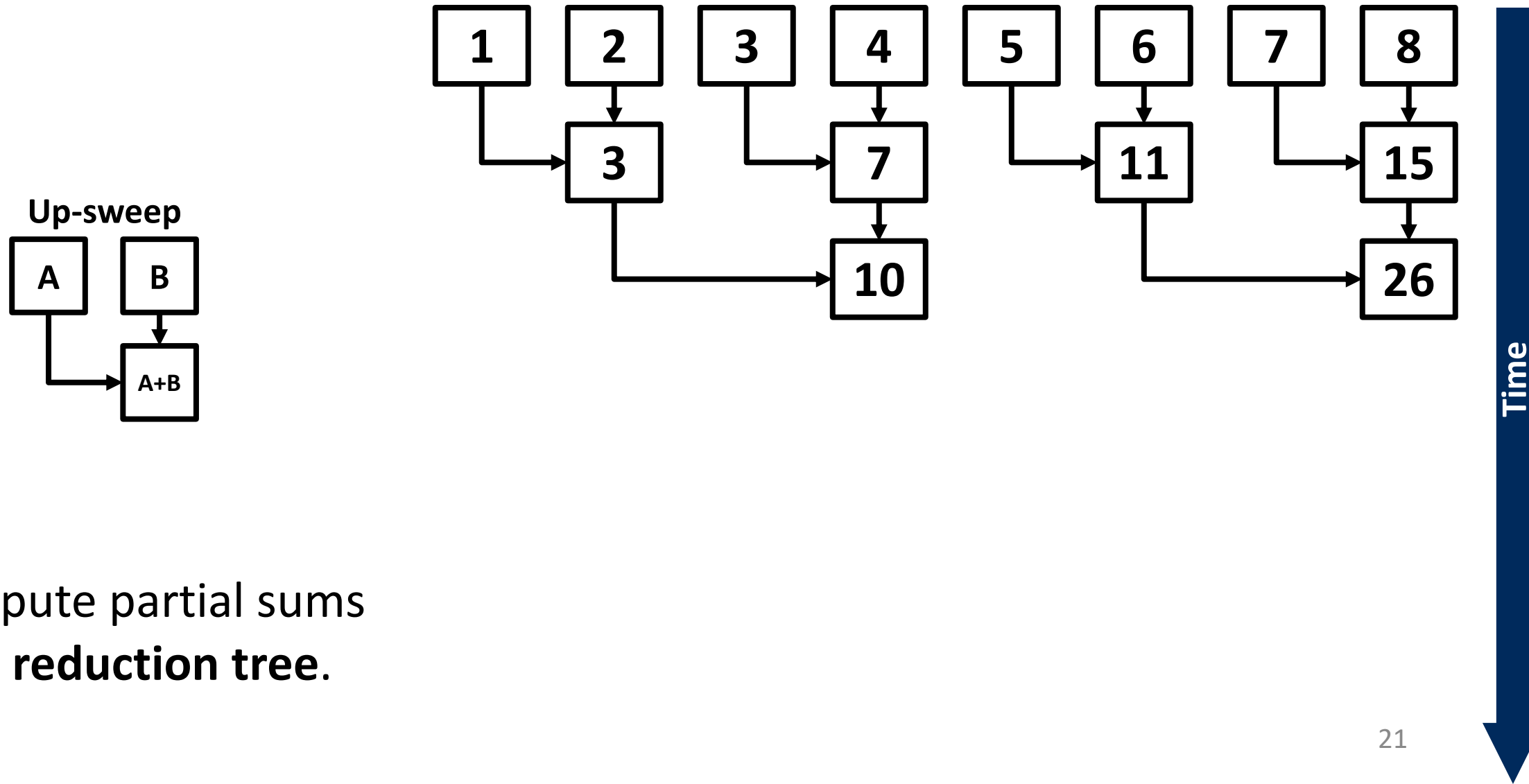
Worker (**p**): an instance of execution;  
e.g., a core in a multi-core CPU

On a single worker: perform scan linearly; takes **n** steps.

With more workers: Can we achieve **sublinear** steps?

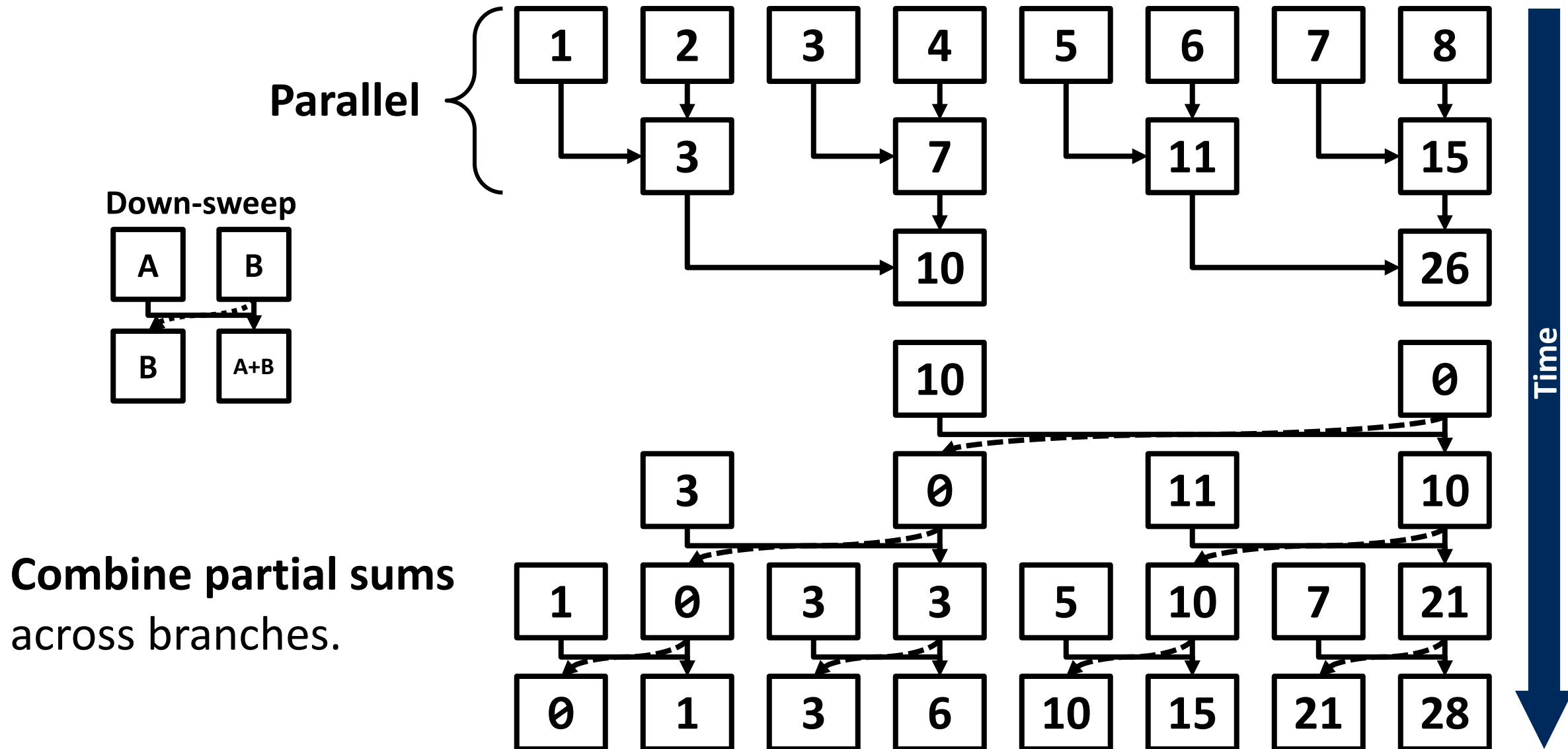


# Blelloch Scan: ① Up-sweep Phase

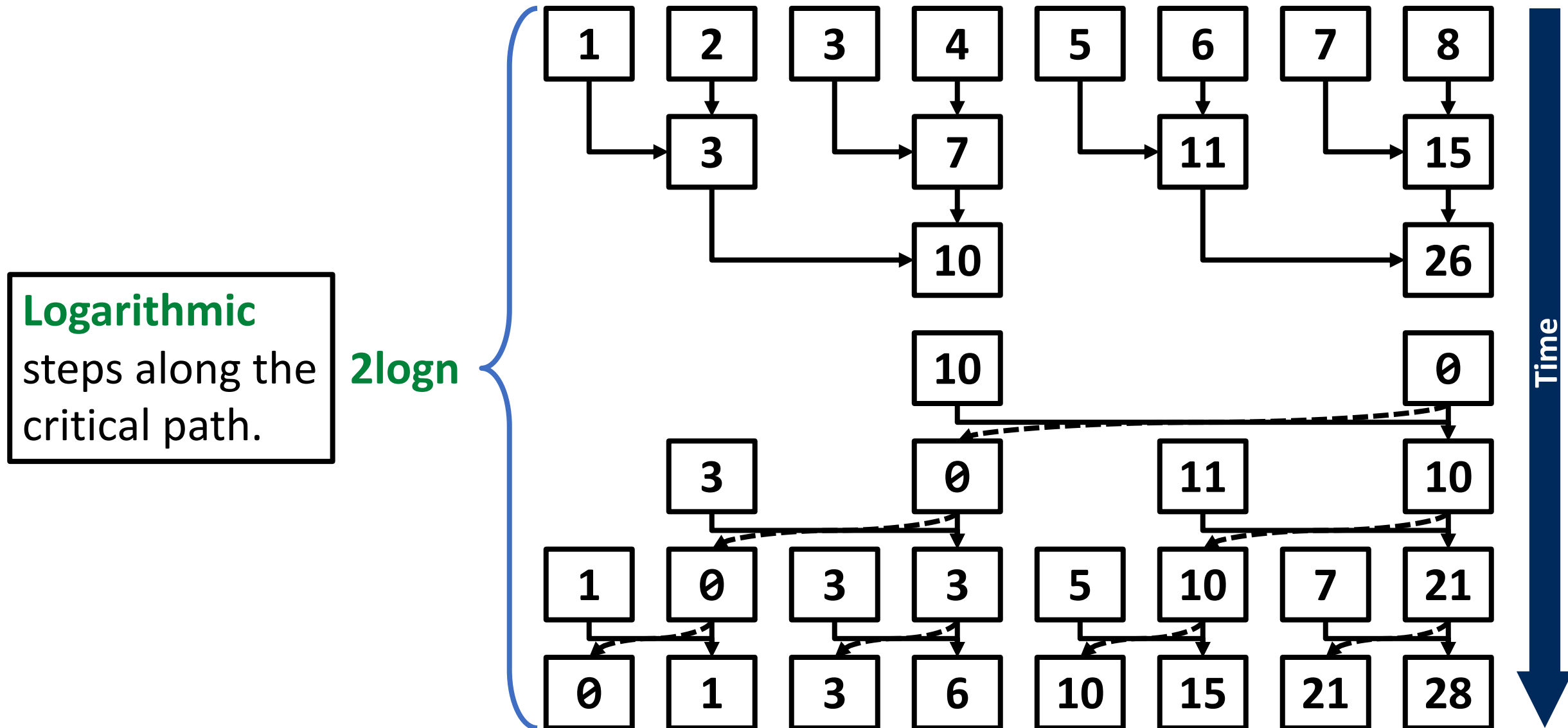


Compute partial sums  
via a **reduction tree**.

# Blelloch Scan: ② Down-sweep Phase



# Blelloch Scan: Efficiency



# Reformulate BP as a Scan Operation

$$G_i = \nabla_{\vec{x}_i} l$$

$$J_{i+1} = \left( \frac{\partial \vec{x}_{i+1}}{\partial \vec{x}_i} \right)^T$$

Binary, associative operator:  $\color{red}{+A} \diamond \color{red}{B} = \color{red}{BA}$

Identity:  $\color{red}{0}$

Input sequence: 

Exclusive scan: 

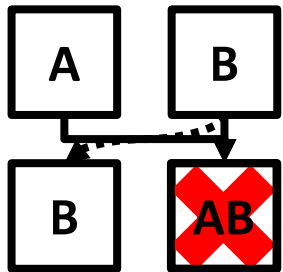
**Key Insight: matrix multiplication in BP is also binary & associative!**

# Scale BP by Blelloch Scan

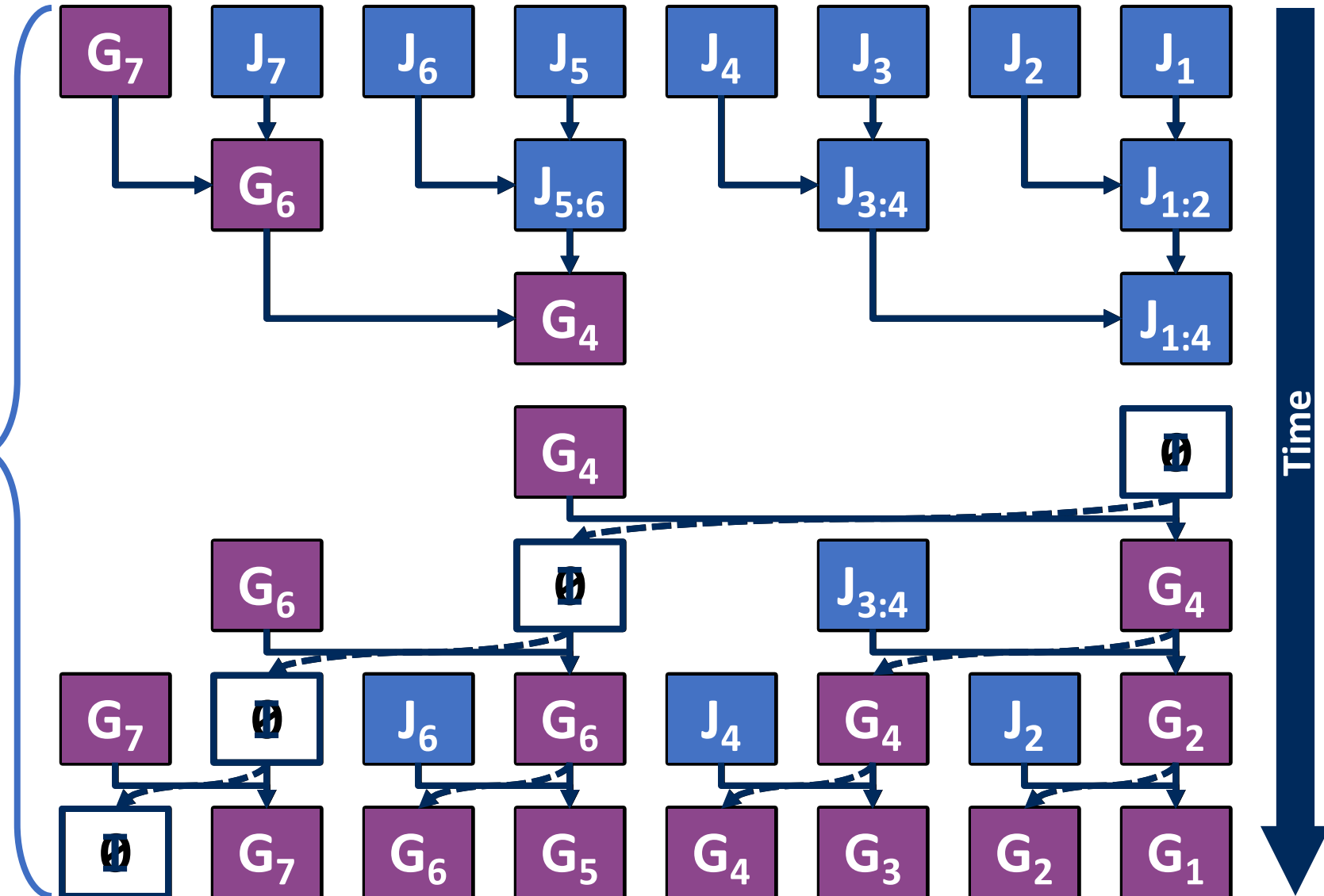
**Logarithmic**  
steps along the  
critical path!

**$2\log n$**

Down-sweep



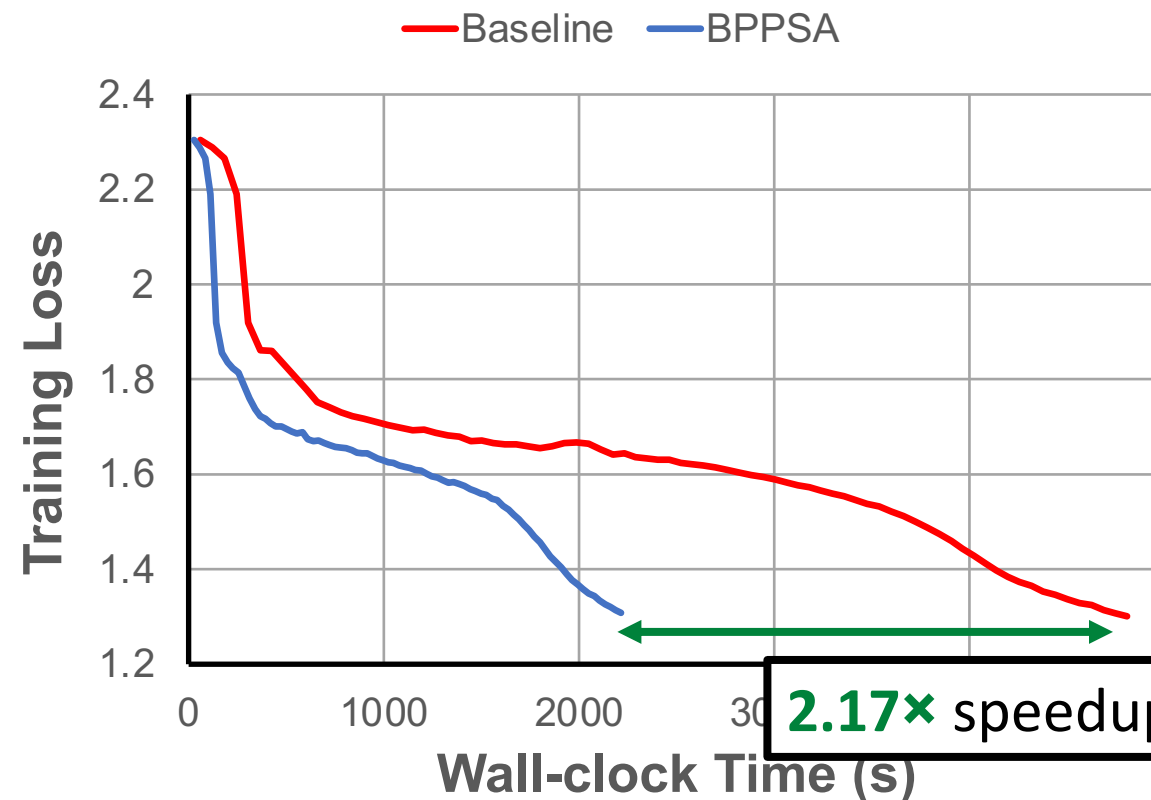
Matrix  
multiplications are  
**noncommutative**.





# End-to-end Training Speedup

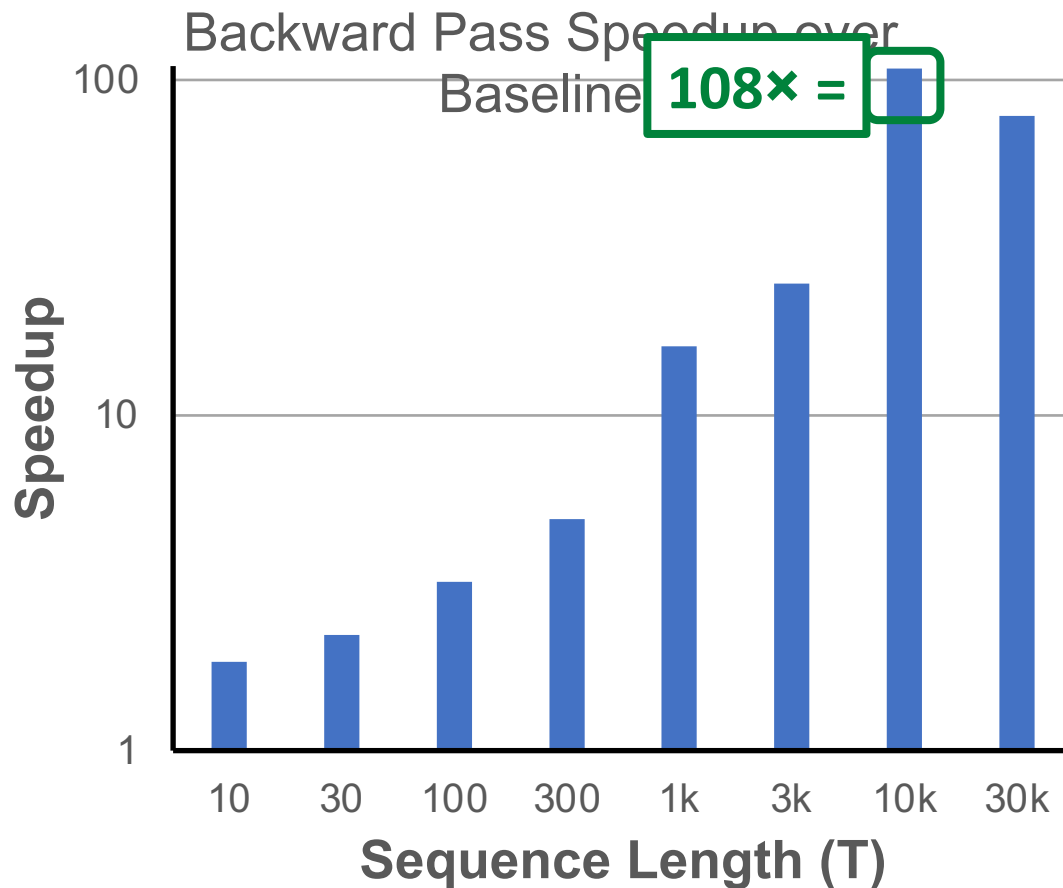
Training curve of BPPSA v.s. the baseline  
when batch size  $B=16$ , sequence length  $T=1000$ :



Numerical differences do **not** effect convergence.

**2.17×** speedup on the overall training time.

# Sensitivity Analysis: Model Length



Sequence length (**T**) reflects the model length **n**.

BPPSA **scales** with the model length (**n**);

until being bounded by the number of workers (**p**).



<https://github.com/UofT-EcoSystem/hfta>

# Horizontally Fused Training Array:

An Effective Hardware Utilization Squeezer For Training Novel Deep Learning Models

**Shang Wang**<sup>4,1,2</sup>, Peiming Yang<sup>\*3,2</sup>, Yuxuan Zheng<sup>\*5</sup>, Xin Li<sup>\*2</sup>, Gennady Pekhimenko<sup>1,2</sup>

1



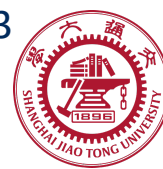
Computer Science  
UNIVERSITY OF TORONTO

2



VECTOR  
INSTITUTE

3



4

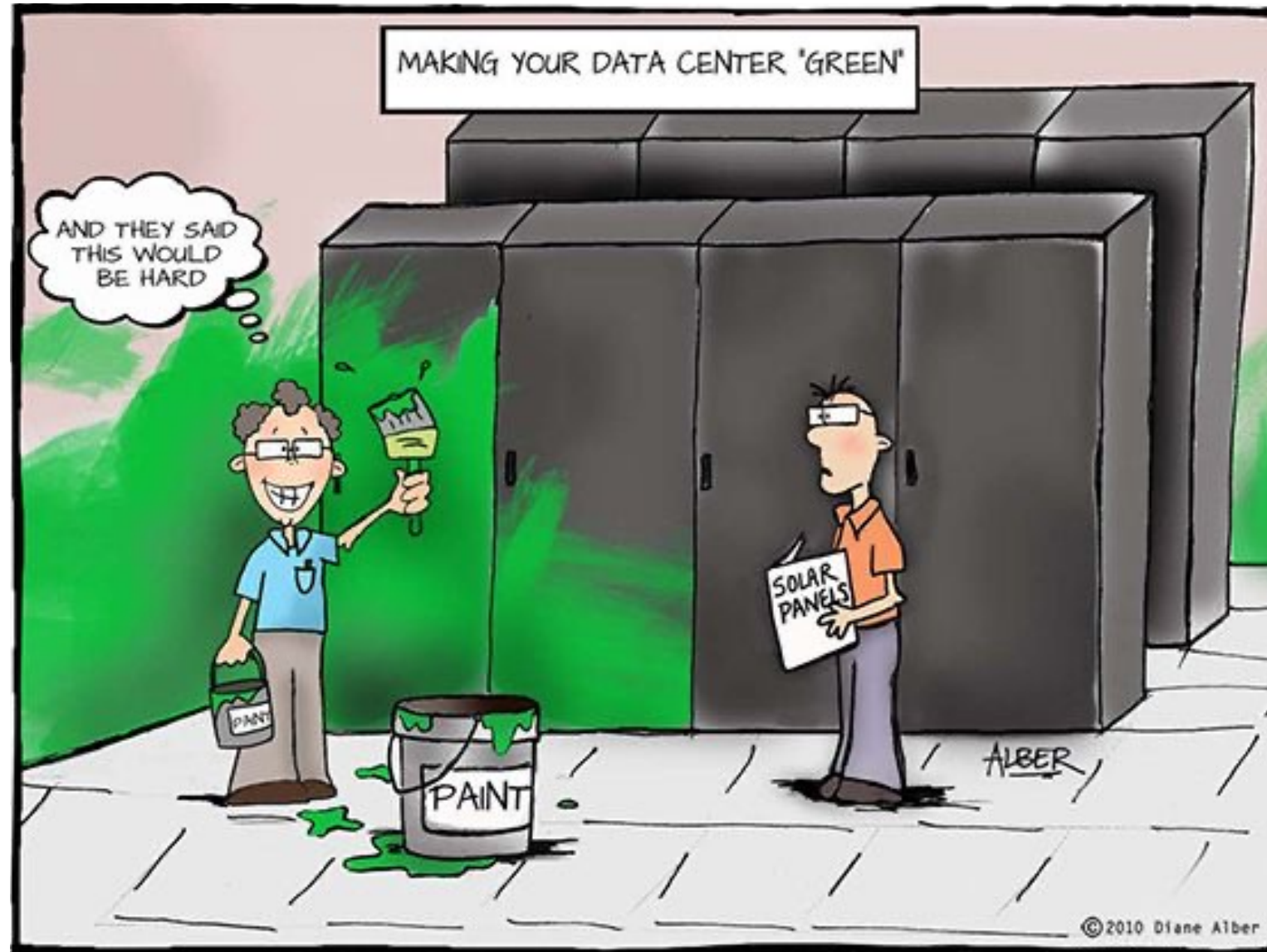


5

intel

MLSys 2021

# Does Training Utilize the Hardware Well?



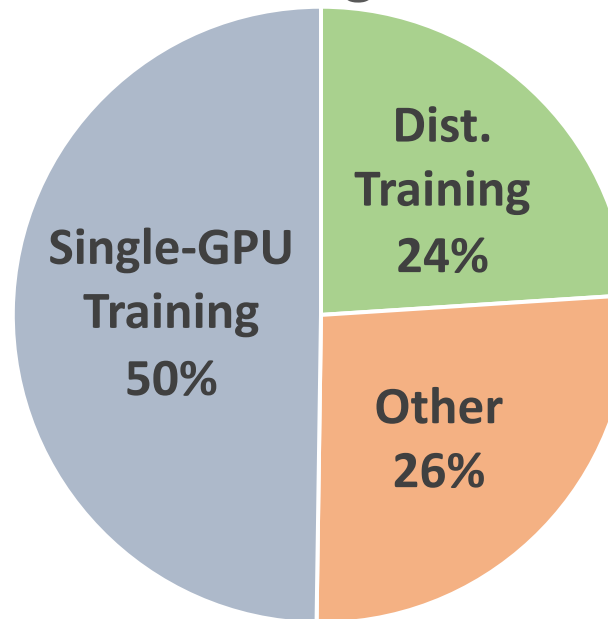
# Hardware Resource Usage @



VECTOR  
INSTITUTE

Monitored over 2 months: **51K** jobs, **472K** GPU hours.

GPU Hour Usage Breakdown



**Single-GPU** training:

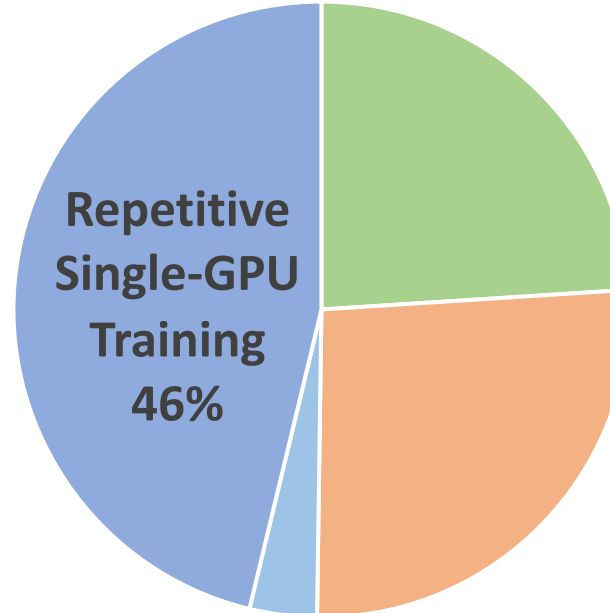
- **Dominates** the GPU hour usage.

# Hardware Resource Usage @



VECTOR  
INSTITUTE

## GPU Hour Usage Breakdown



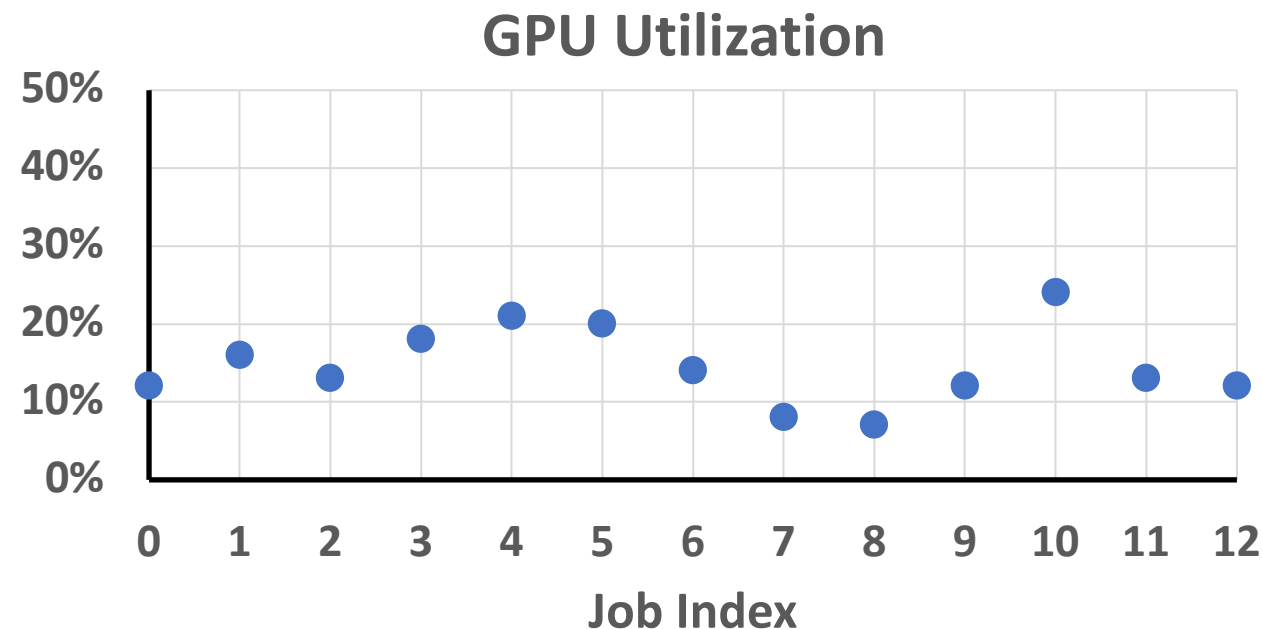
**Repetitive** single-GPU training:

- **Dominates** the GPU hour usage.
- **Concurrent** jobs; **same** program; **different** configs.
- For **hyper-param. tuning** or **convergence stability testing**.

# Hardware Resource Usage @



VECTOR  
INSTITUTE



**Repetitive** single-GPU training:

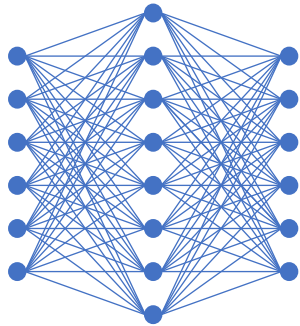
- Often have **low hardware utilization**.

Why? 🤔



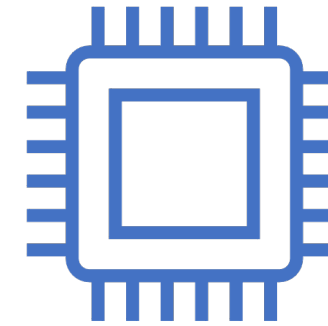
# Why Hardware Underutilization?

**DL**



?

**Hardware**

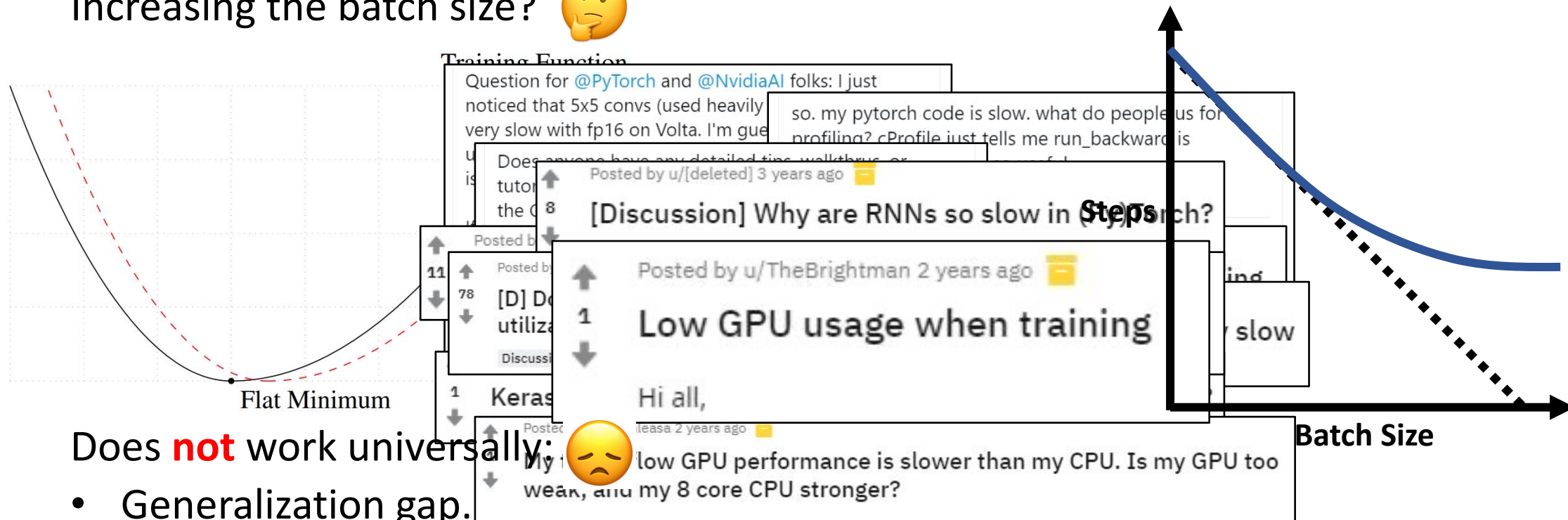


?

# Performance Optimization is **Hard**.

More so for **system & architecture “novices”**.

Increasing the batch size? 🤔



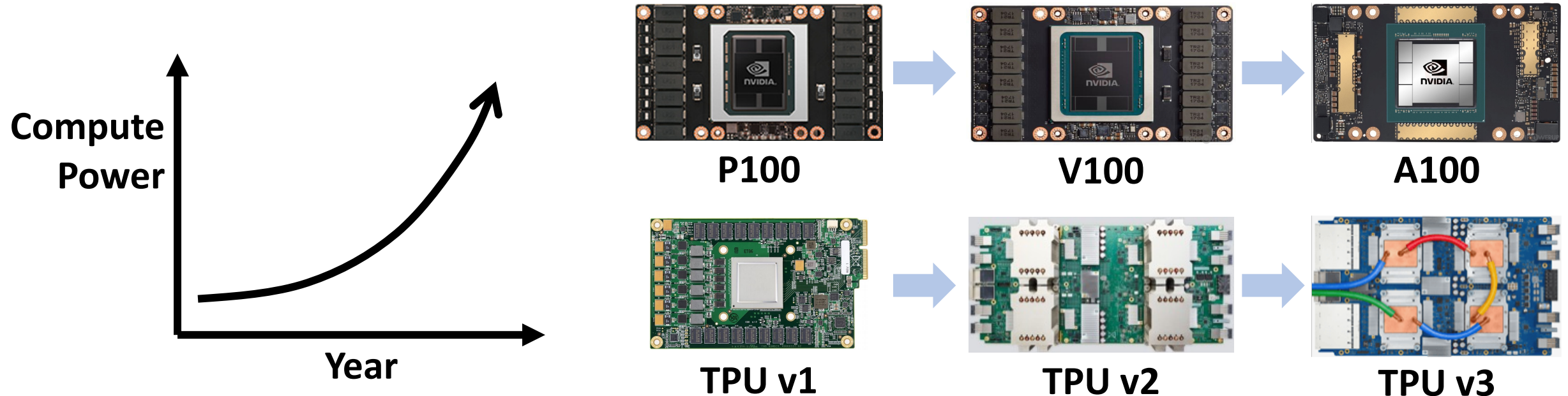
Does **not** work universally: 🙄

- Generalization gap.
- Batch size scaling limit.

<sup>3</sup>Keskar et al. On large-batch training for deep learning: Generalization gap and sharp minima. ICLR, 2017

<sup>4</sup>Shallue et al. Measuring the effects of data parallelism on neural network training. J. Mach. Learn. Res., 20:112:1–112:49, 2019

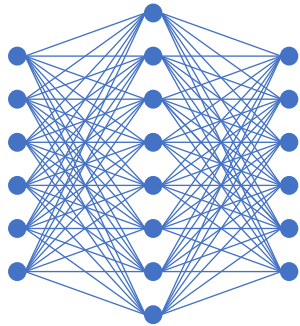
# Accelerators Get More Powerful.



**Unoptimized** workload → **Harder** to utilize well.

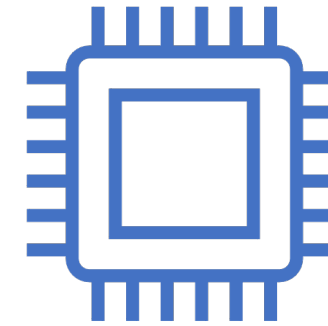
# Why Hardware Underutilization?

DL



Performance optimization is **hard**.

Hardware



Accelerators get more powerful.

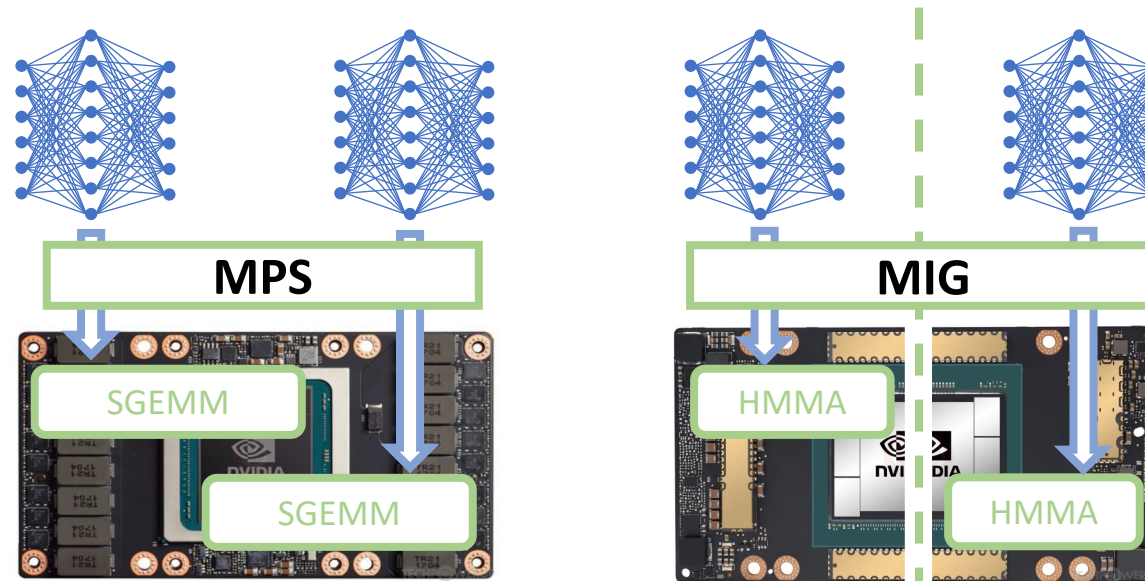
How to improve hardware utilization?



# Train $>1$ Models on 1 Accelerator Simultaneously?

**Special features** for sharing among **arbitrary processes**.

(e.g., MPS and MIG on NVIDIA GPUs)



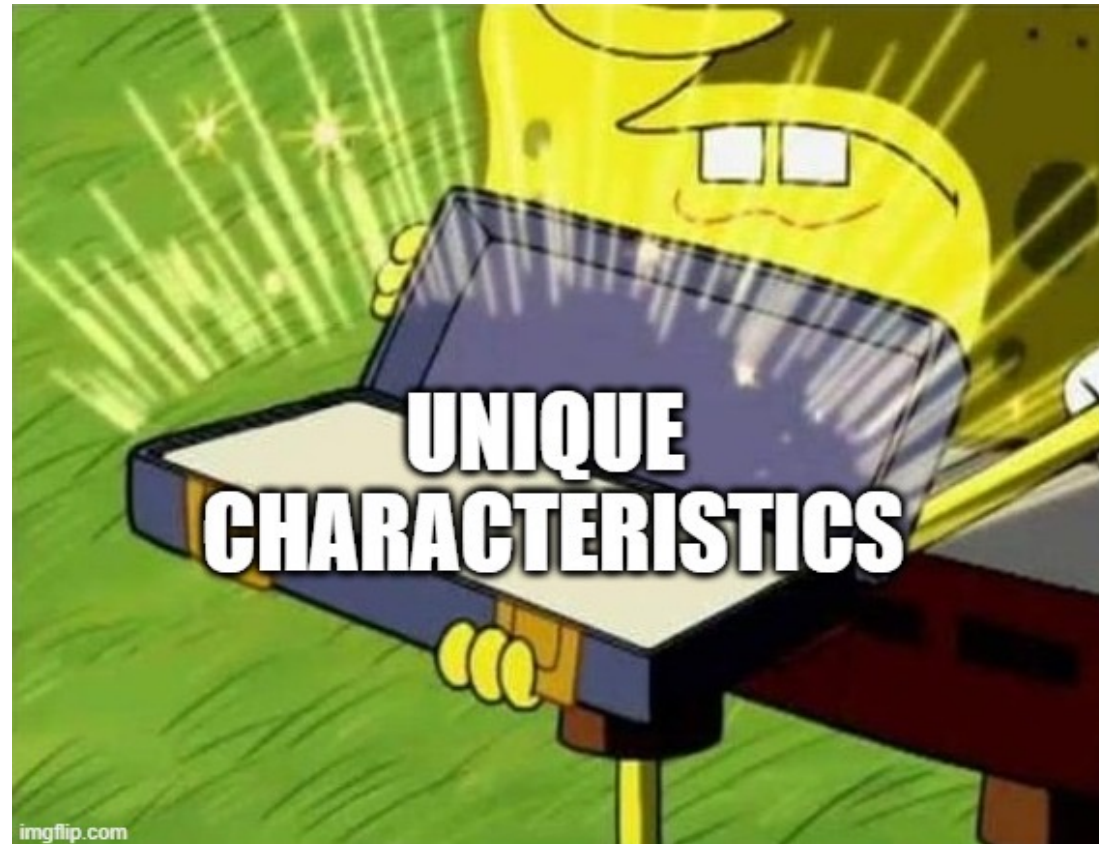
**Less effective** for repetitive training jobs.

Other accelerators (e.g., TPUs) do **not** possess such features.

What to do instead?

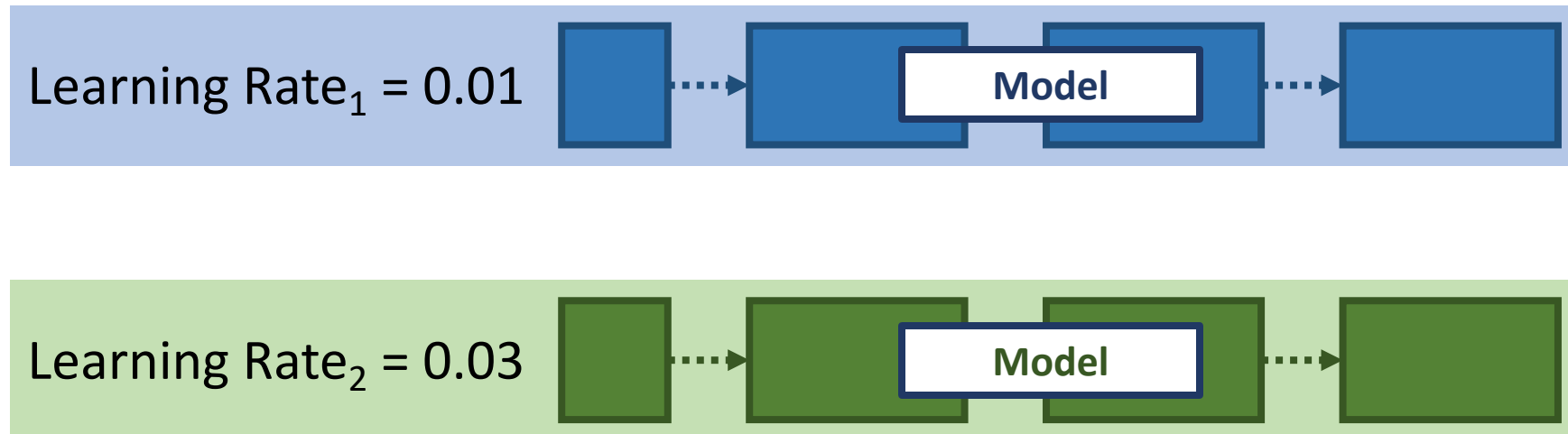


# Key Ideas



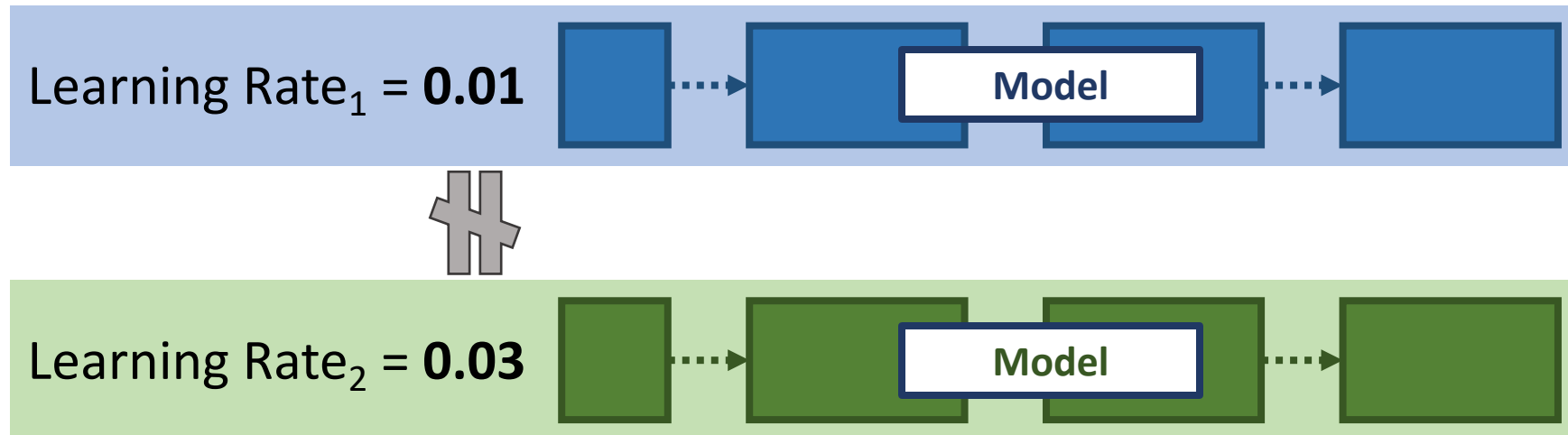
# Model Similarity

**Launched repetitively (e.g., hyper-parameter tuning)**



# Model Similarity

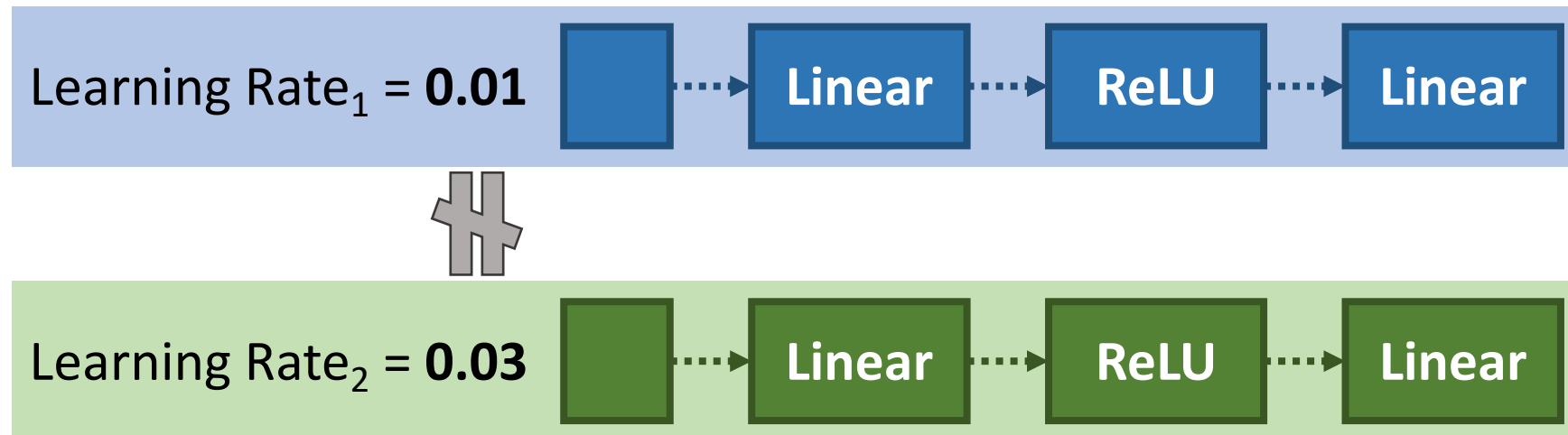
Launched repetitively (e.g., hyper-parameter tuning)





# Model Similarity

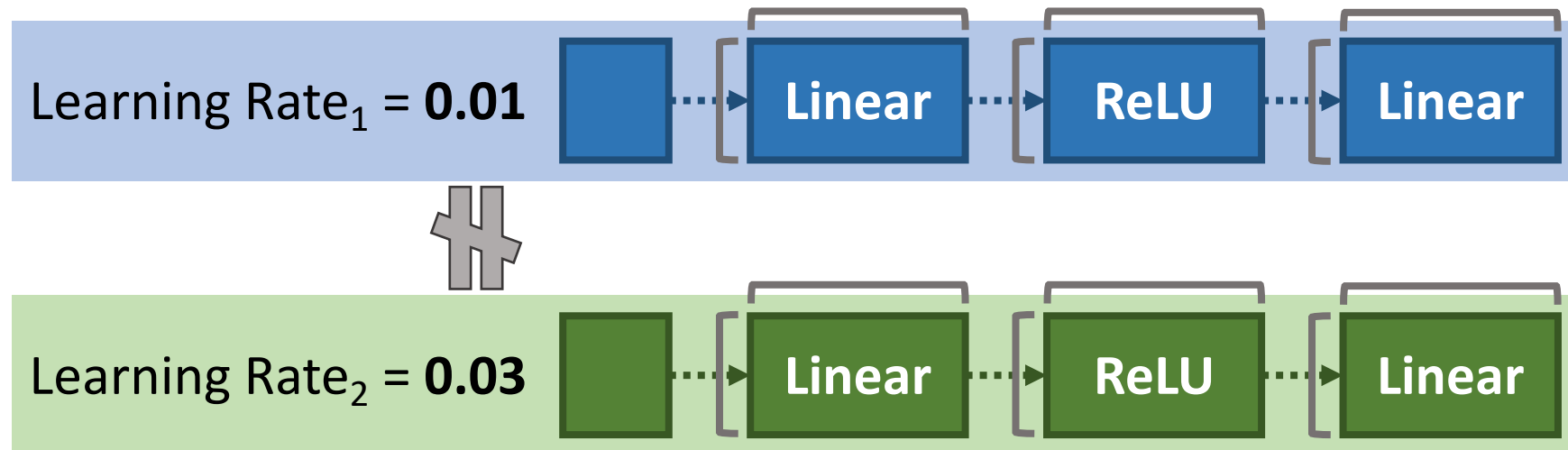
Launched repetitively (e.g., hyper-parameter tuning)



Same types of ops.

# Model Similarity

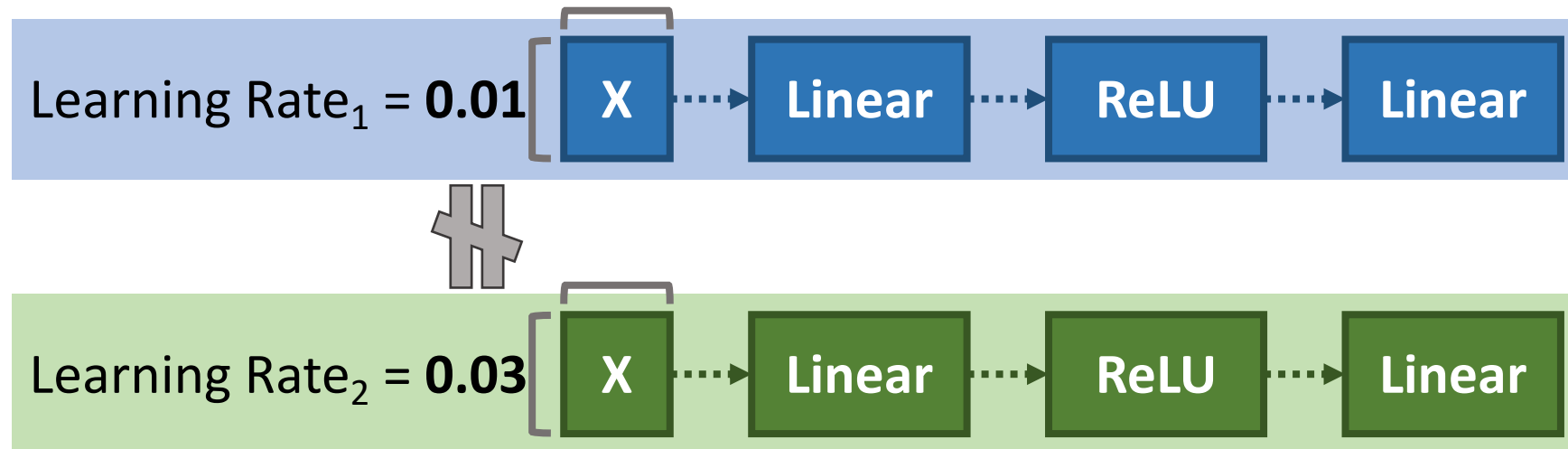
Launched repetitively (e.g., hyper-parameter tuning)



Same types of ops with the same shapes.

# Model Similarity

Launched repetitively (e.g., hyper-parameter tuning)

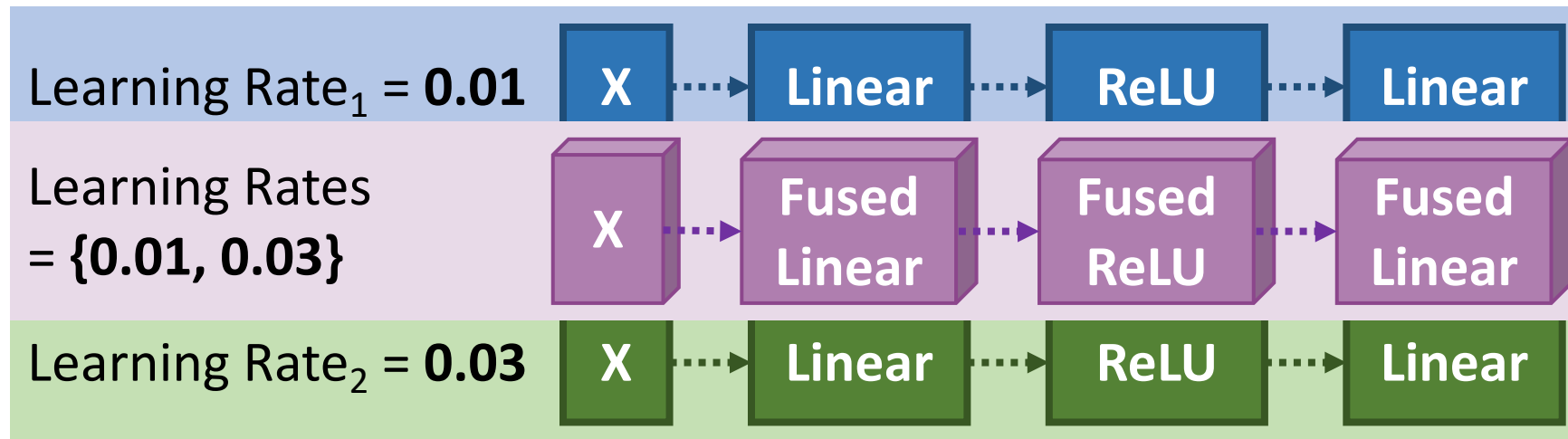


Same types of ops with the same shapes.

What to do with it? 🤔

# Inter-model Horizontal Operator Fusion

We propose:



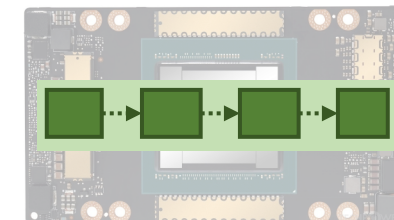
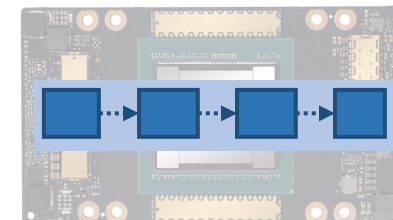
But, DL stack → training **single** models on **separate** accelerators.

PyTorch

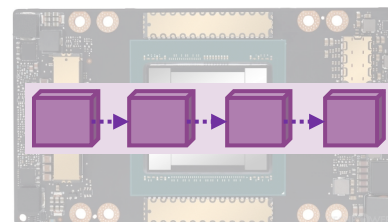
TensorFlow

mxnet

cuDNN



How to

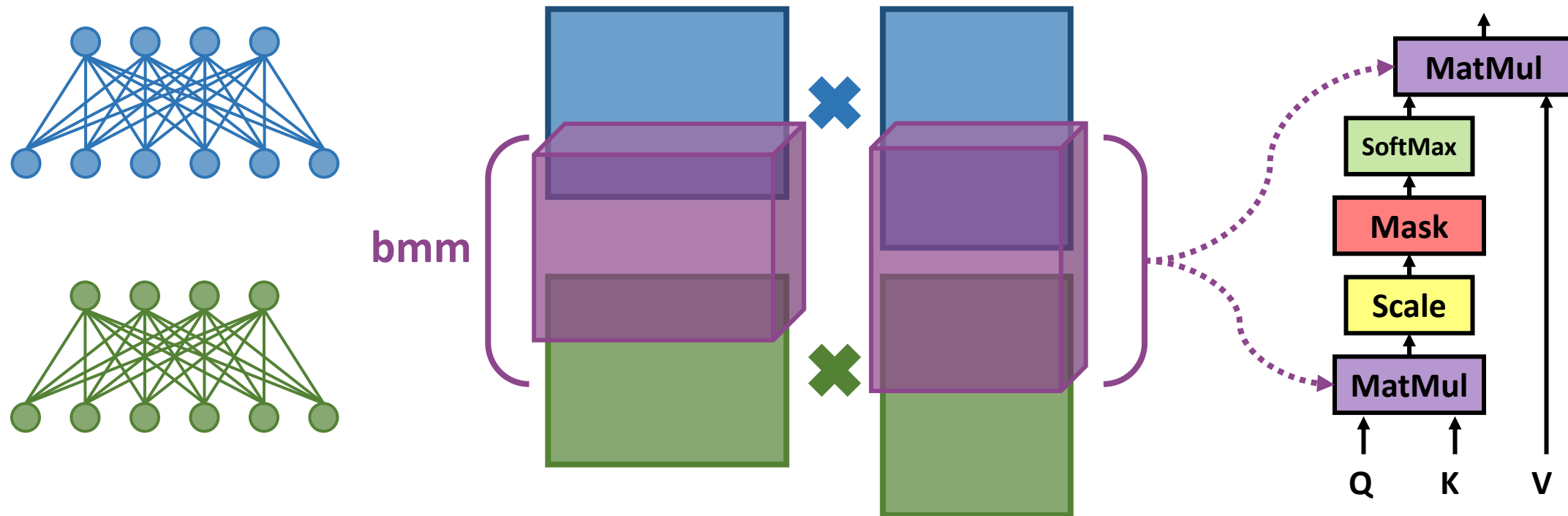


?



# Implementation Reuse

**Horizontally fused ops** → other existing **mathematically equivalent ops**.

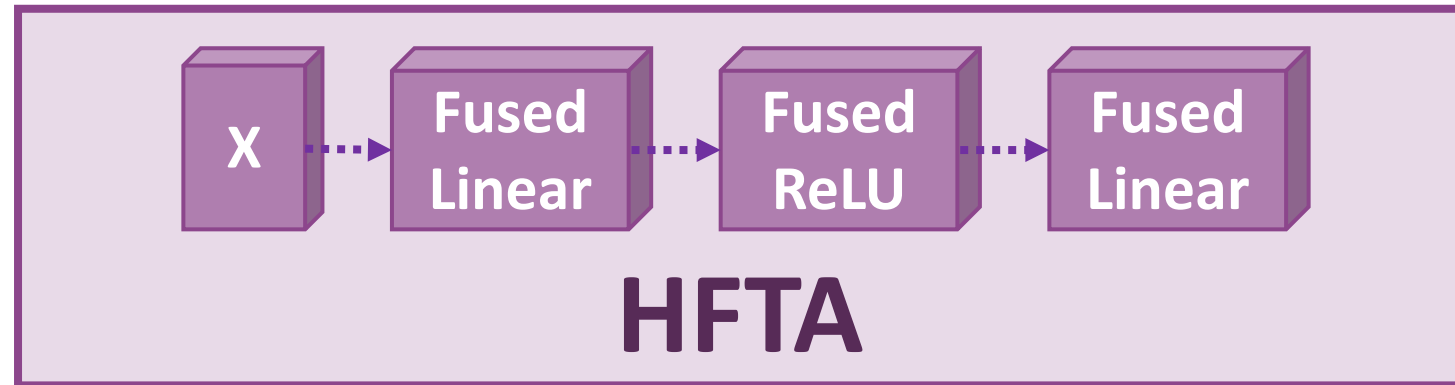


What about other ops?



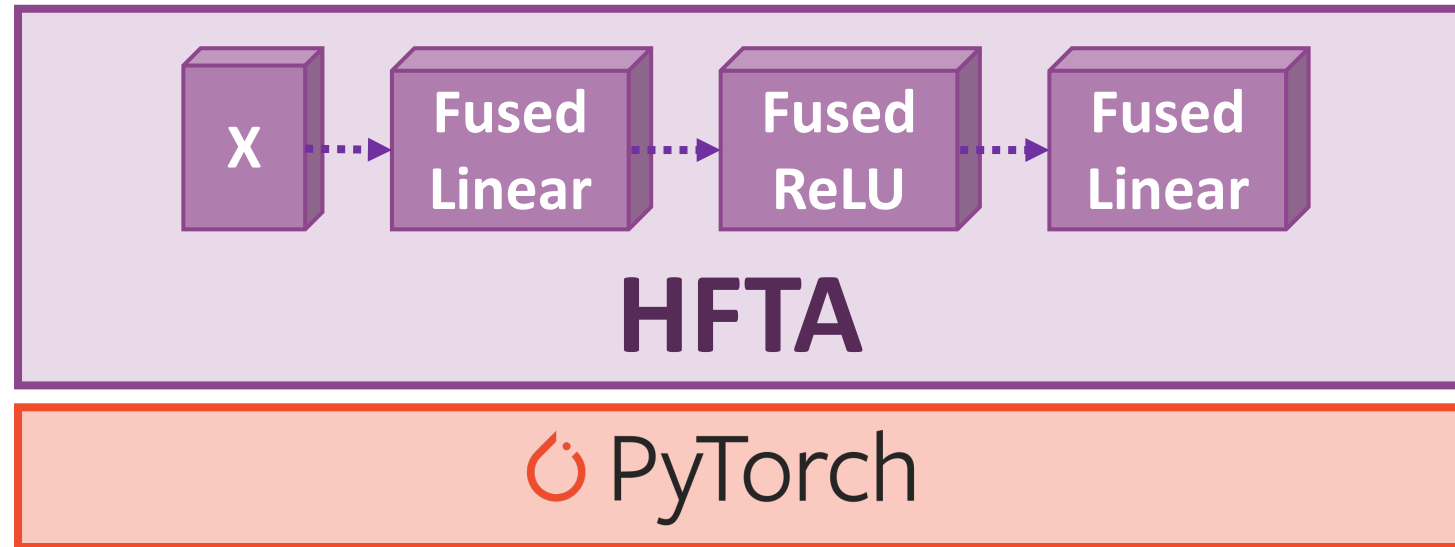
# Horizontally Fused Training Array (HFTA)

Different ops  $\rightarrow$  different rules  $\rightarrow$  tools required.



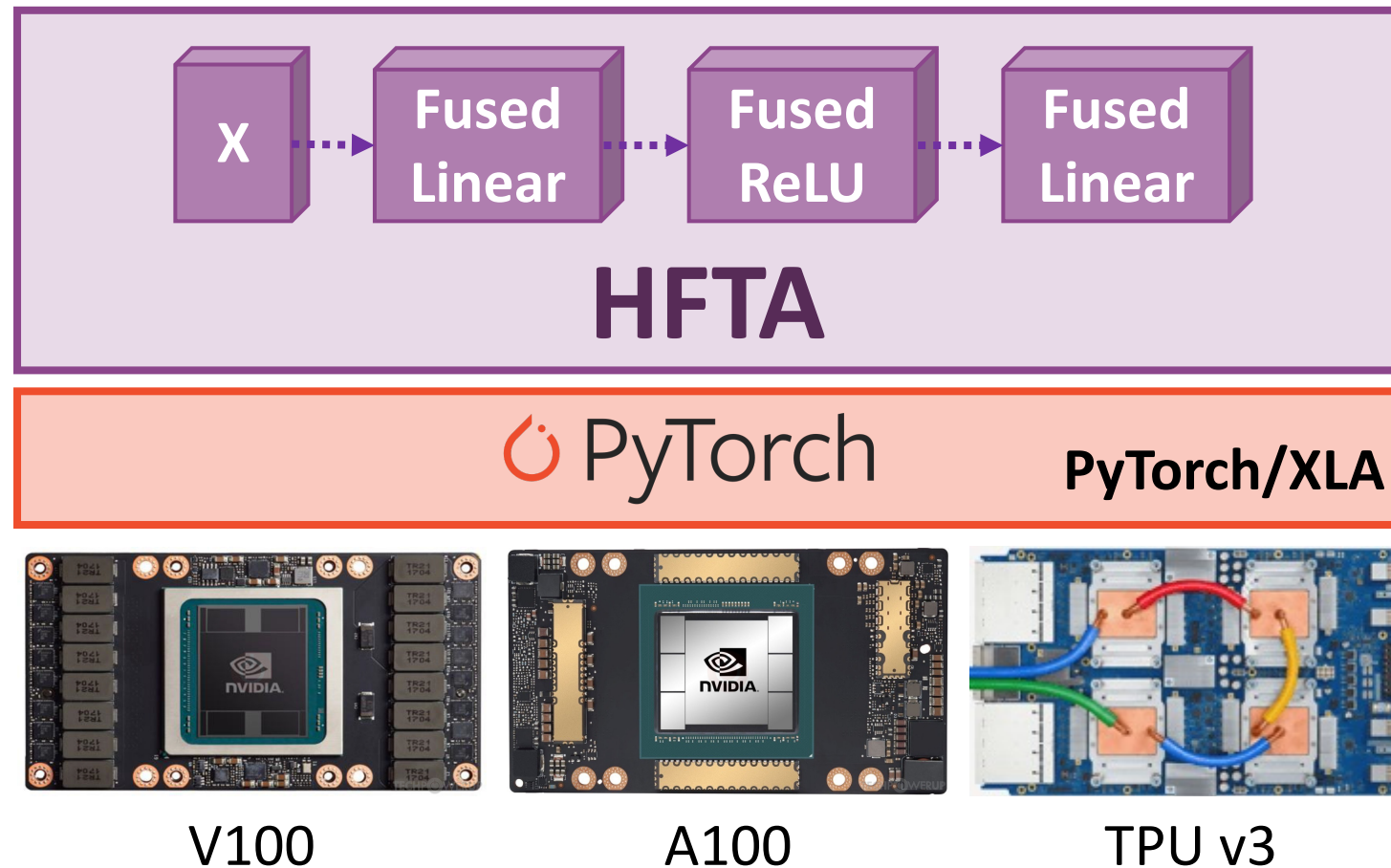
# Horizontally Fused Training Array (HFTA)

We choose **PyTorch** for its **popularity**, but the idea is **general**.



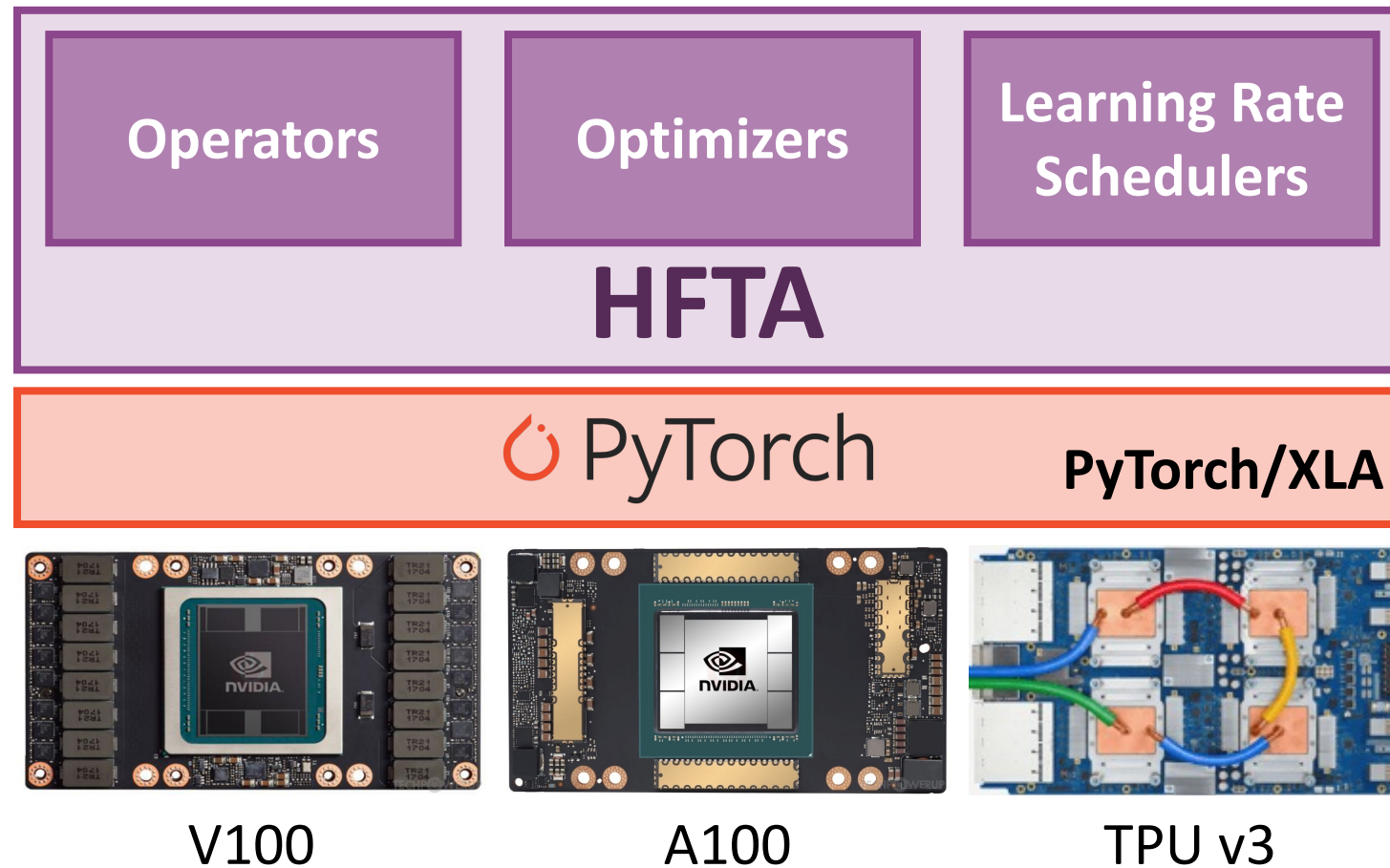
# Horizontally Fused Training Array (HFTA)

Support **all** DL framework's hardware backends.





# HFTA Components



# HFTA: Fused Operators

Conv1d, Conv2d, ConvTranspose2d

Linear

MaxPool2d, AdaptiveAvgPool2d

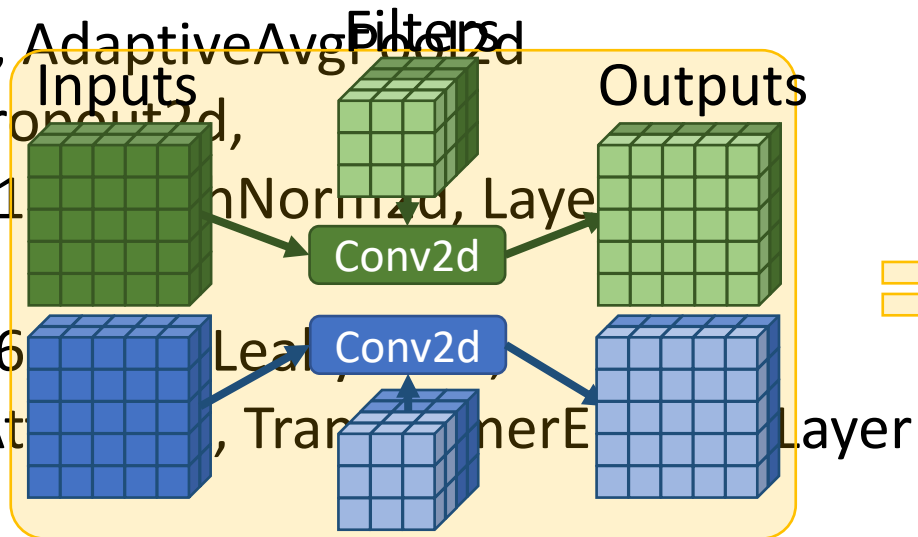
Dropout, Dropout2d,

BatchNorm1d

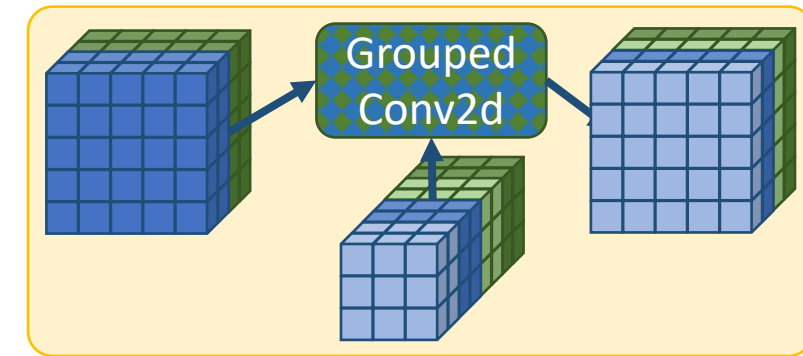
Embedding

ReLU, ReLU6

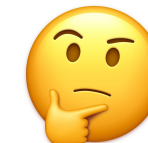
MultiheadAttn



=

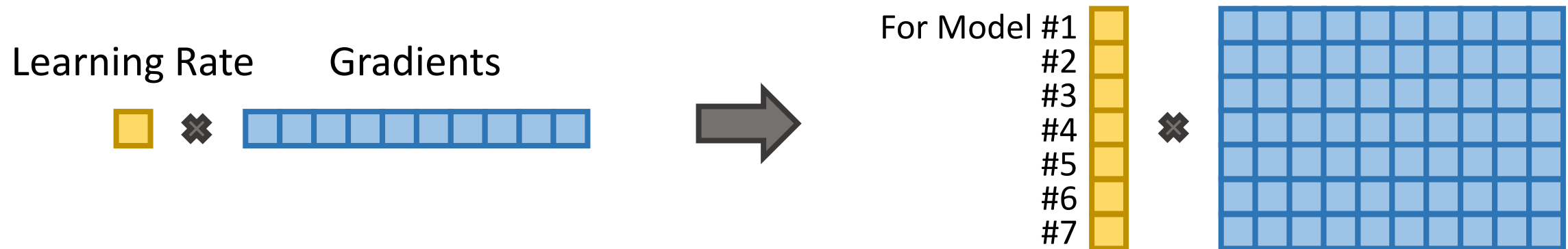


What else can we fuse?



# HFTA: Fused Optimizers and LR Schedulers

Adadelta, Adam  
StepLR

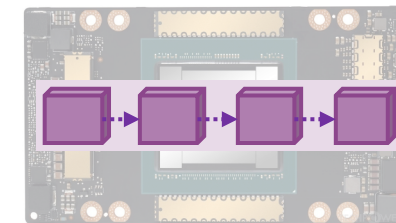
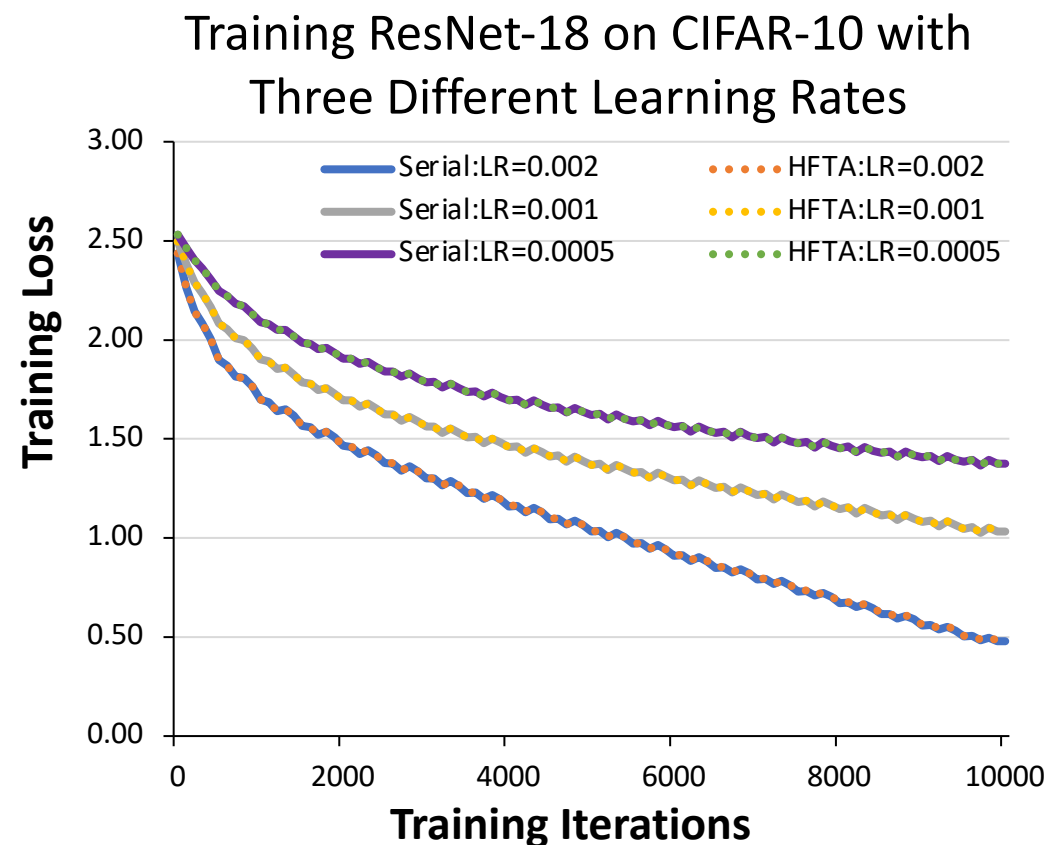
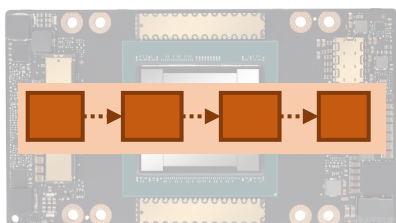
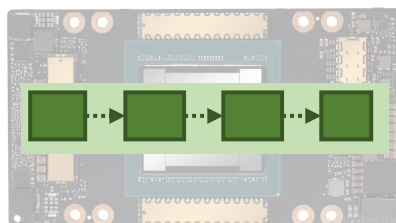
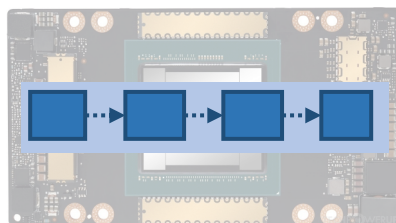


What else can we fuse?



# No Impact on Convergence

## Mathematically Equivalent Transformations



What about training throughputs?



# Methodology: Environment

Accelerators:

From:

Versions:

 PyTorch



**Initial Experimental results**

**Repeating Experiments**

**TPU v3**

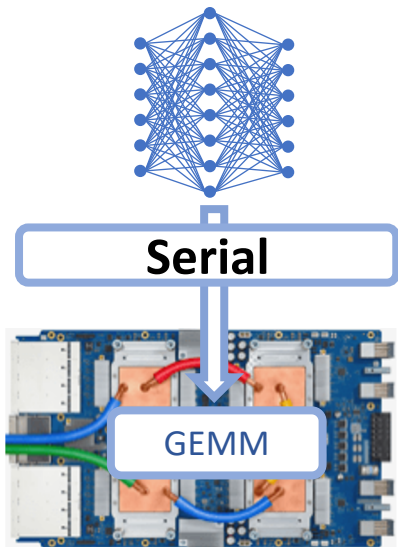


Google Cloud

1.7

# Methodology: Baselines

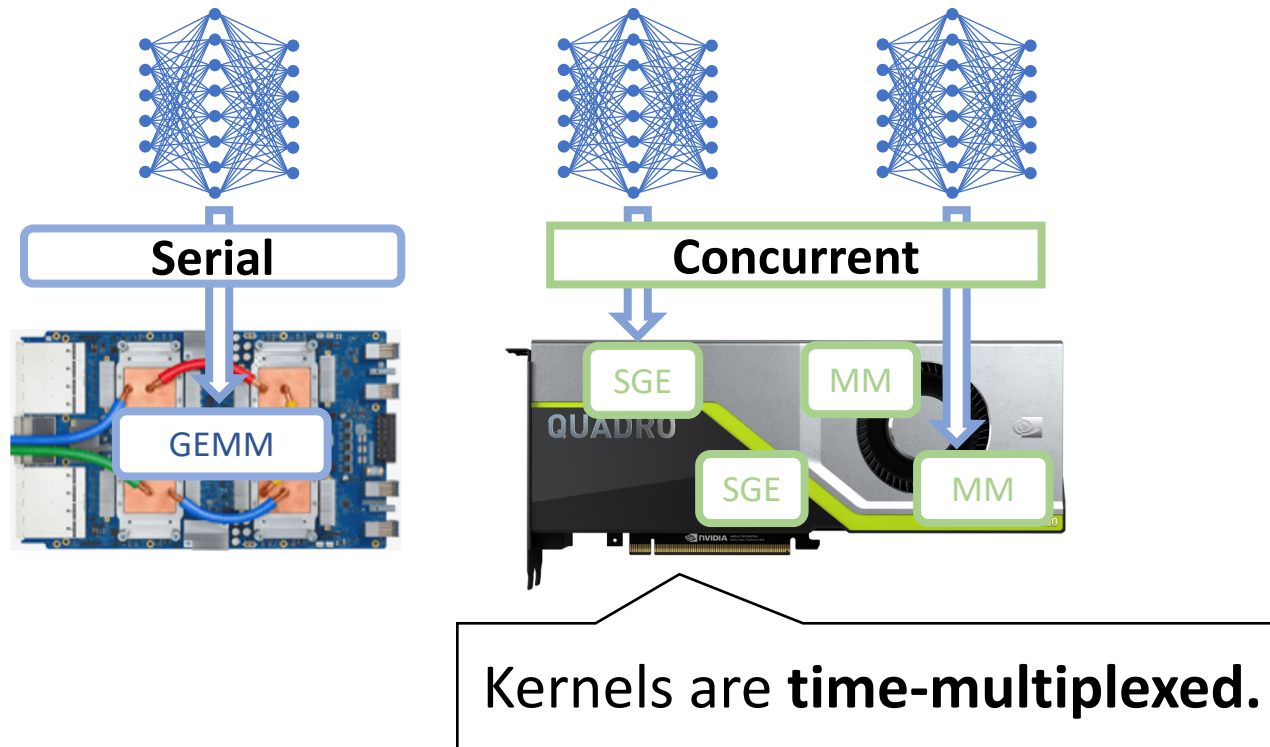
One model per accelerator.



The **common practice** in hyper-param. tuning frameworks.

# Methodology: Baselines

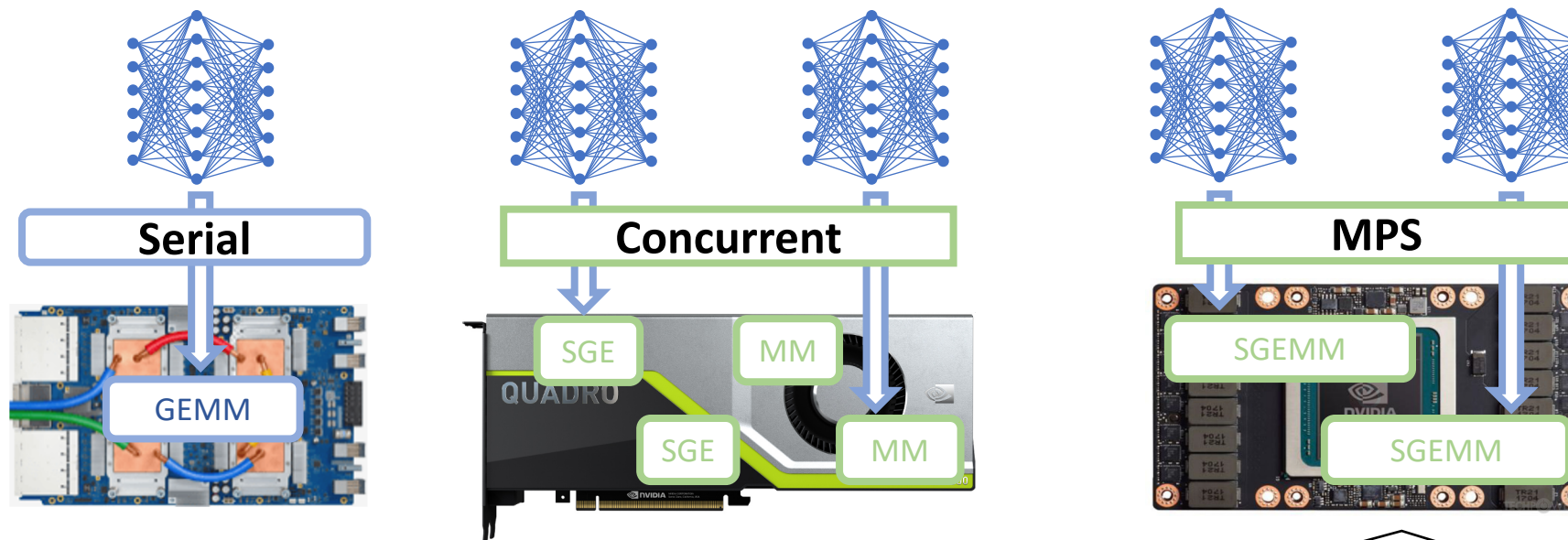
Some accelerators (e.g. NVIDIA GPUs) support running  $>1$  processes.





# Methodology: Baselines

Some accelerators (e.g. NVIDIA GPUs) support running  $>1$  processes.

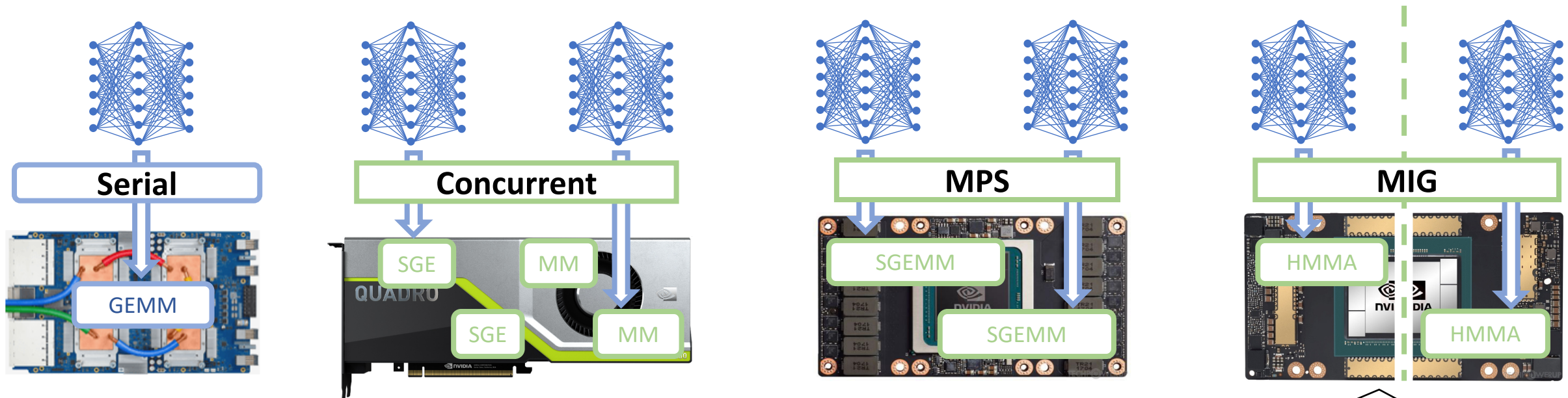


**Co-run**  $>1$  kernels if a **single** kernel underutilizes the GPU.



# Methodology: Baselines

Some accelerators (e.g. NVIDIA GPUs) support running  $>1$  processes.



Slice (**only**) **A100** into ( $\leq 7$ ) partitions.

# Methodology: Workloads

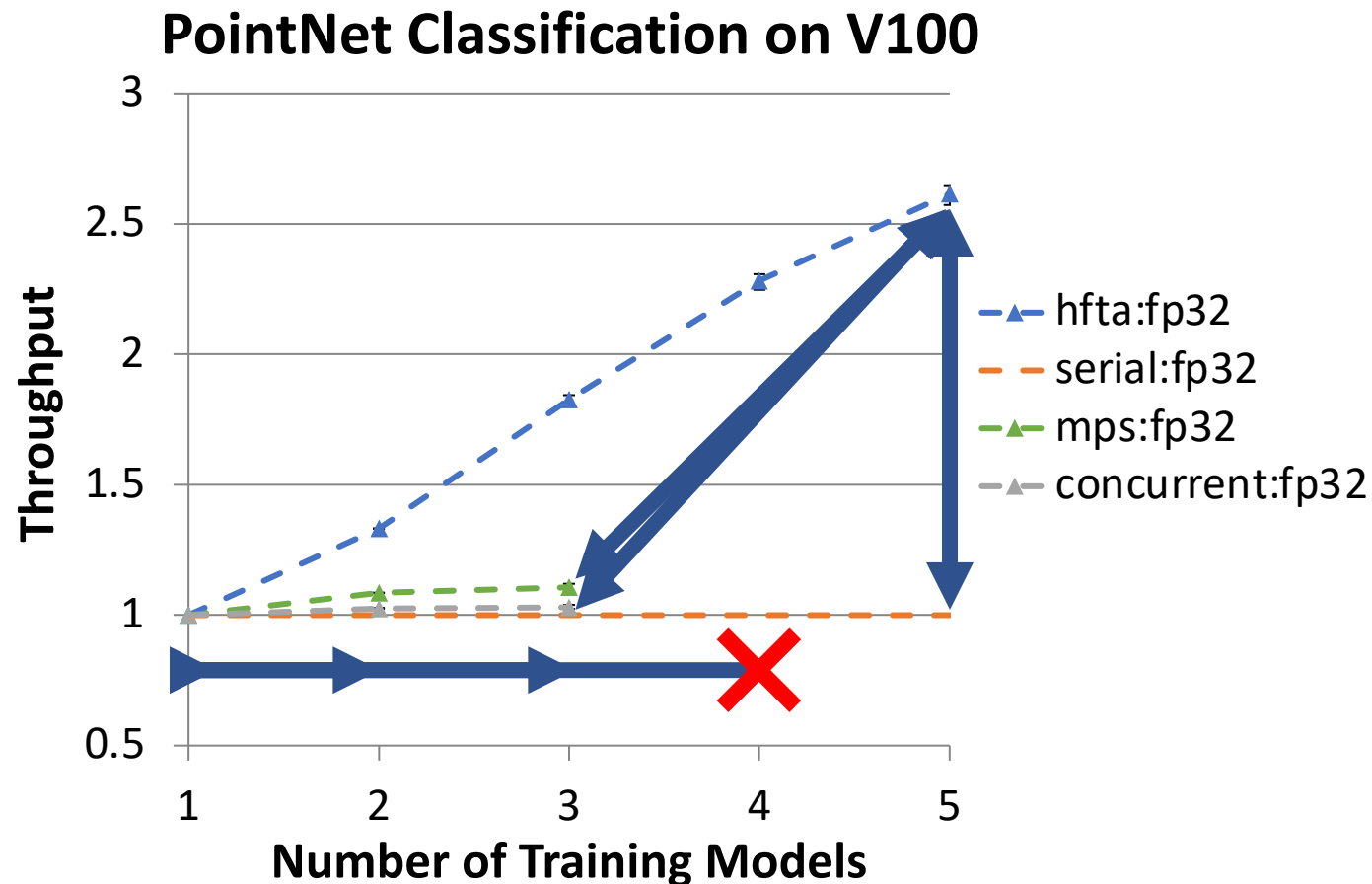
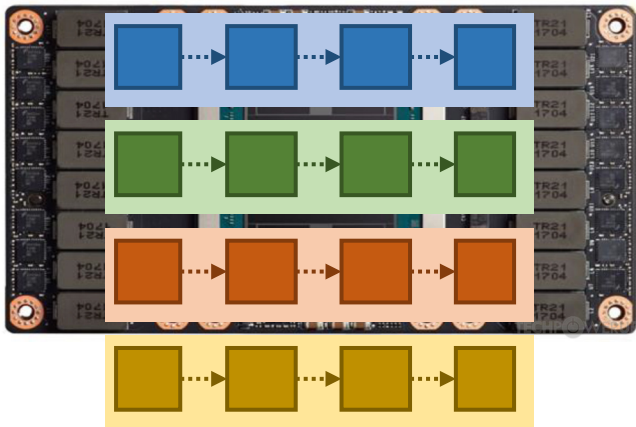
Model	PointNet		DCGAN	ResNet <sub>18</sub>	MobileNet <sub>v3L</sub>	Transformer	BERT <sub>Medium</sub>
Task	Point Cloud Classification	Point Cloud Segmentation	Images Generation	Image Classification		Language Modeling (LM)	Mask LM
Dataset	ShapeNet part		LSUN	CIFAR-10		WikiText-2	

**Not** intensively optimized → realistically reflect novel DL research workloads.

# V100 Results

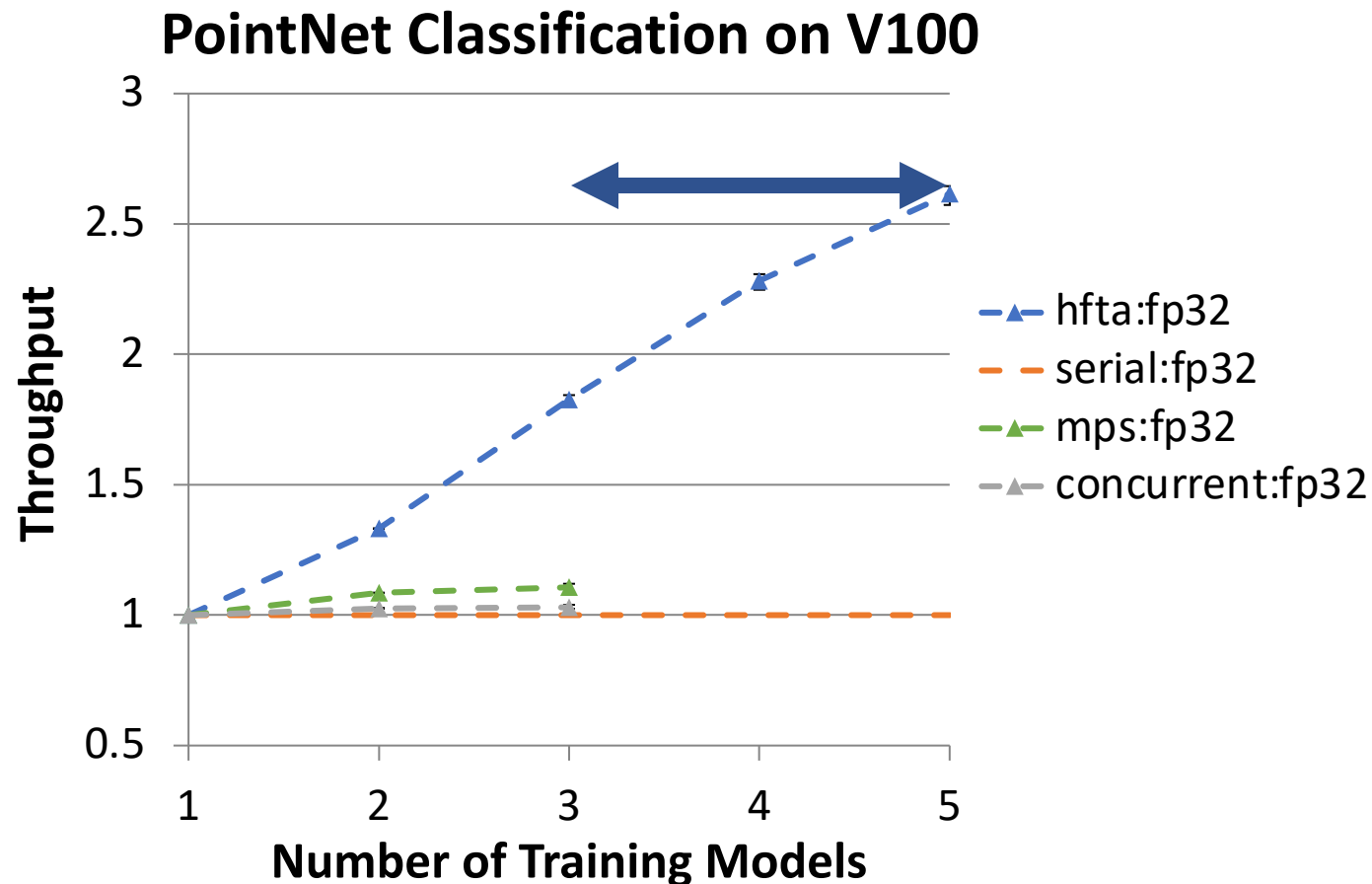
Keep sharing with **more** models until **OOM**.

- Serial
- Concurrent
- MPS



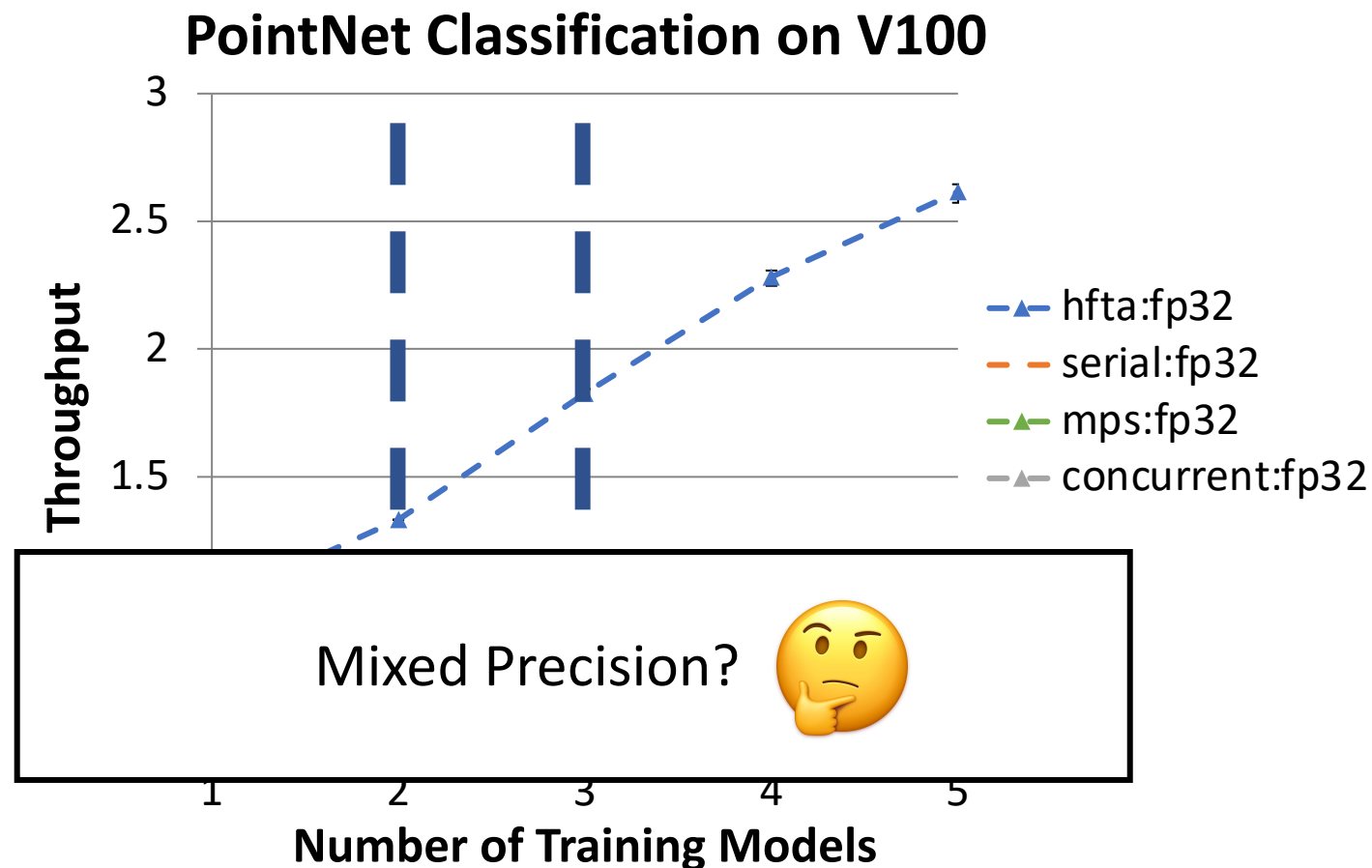
# V100 Results

**Fixed** memory budget, HFTA co-trains **more** models than **MPS** and **concurrent**.



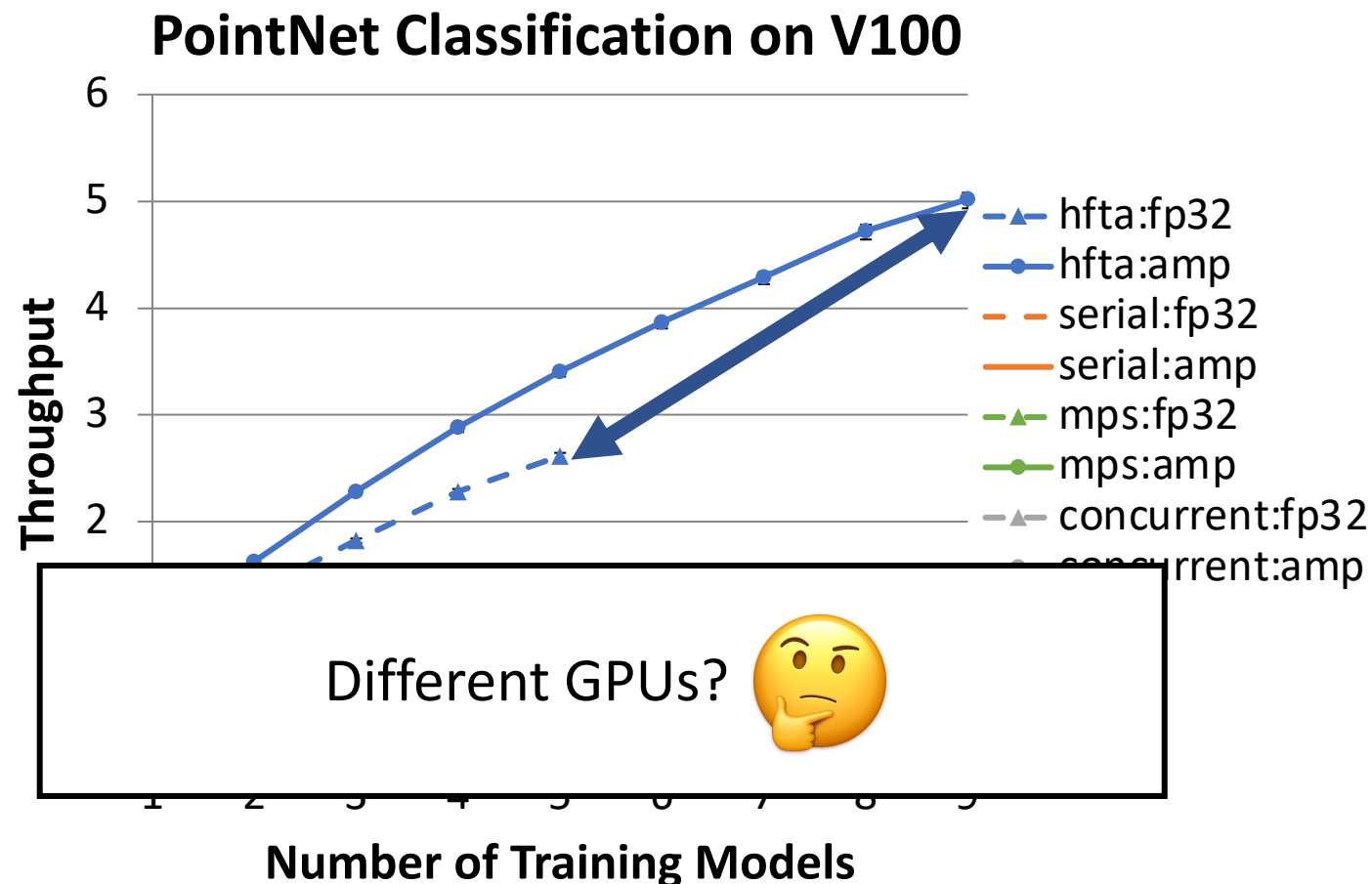
# V100 Results

Same # of models & same GPU, HFTA achieves **higher throughput** than all baselines.



# How About Mixed Precision?

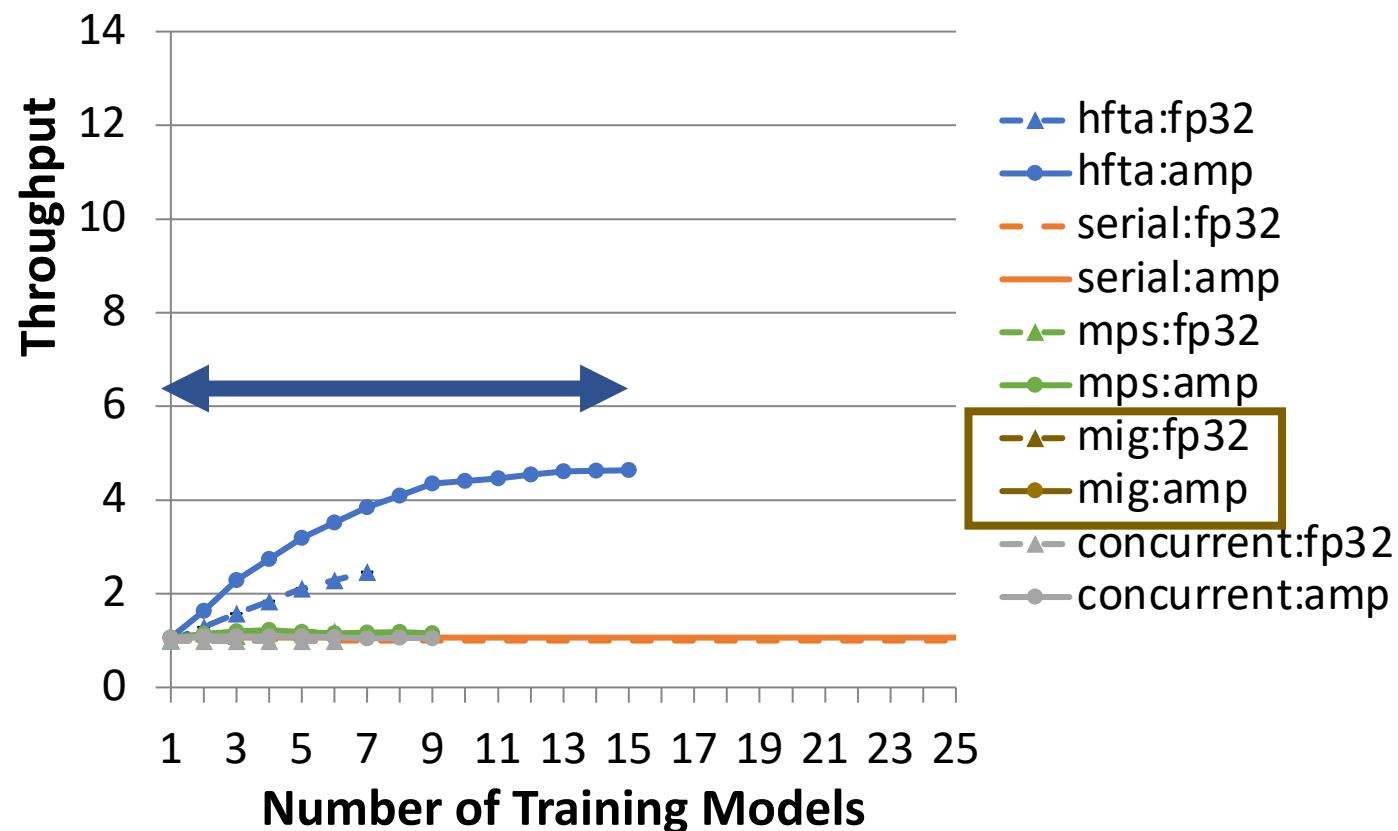
HFTA can **better exploit** tensor cores during AMP training than all baselines.



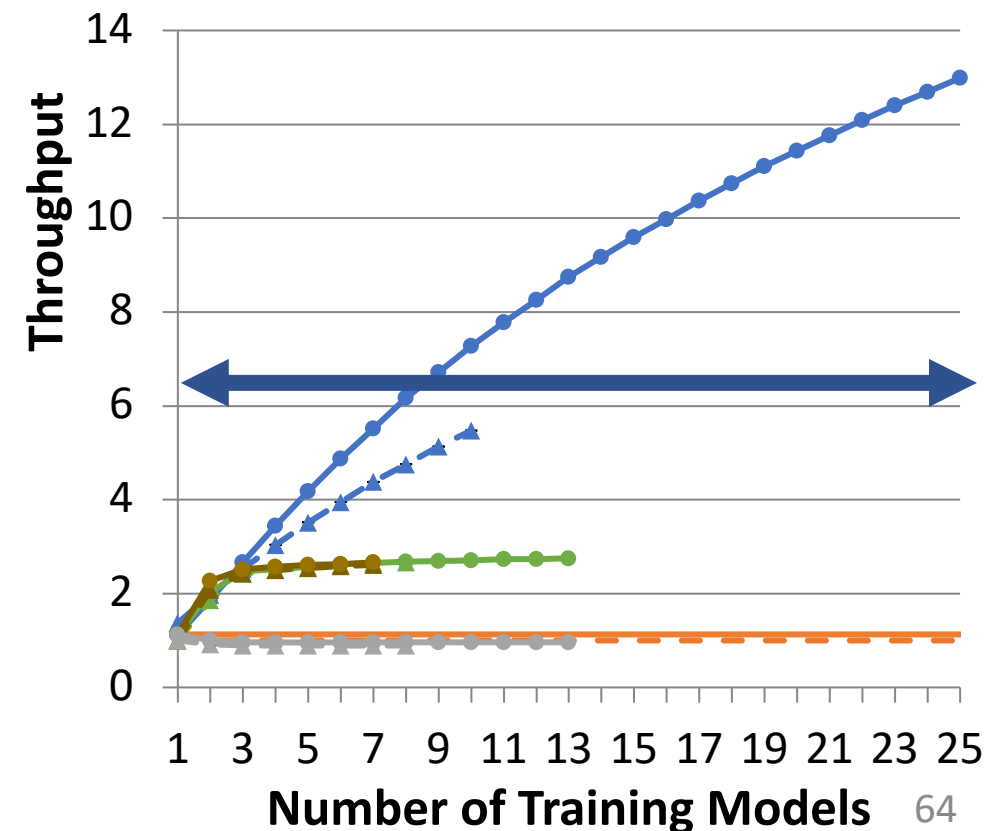
# How About Fancier GPUs?

Since  $\text{Mem}(\text{A100}) > \text{Mem}(\text{RTX6000})$ , HFTA can fit **more** models on A100.

## PointNet Classification on RTX6000

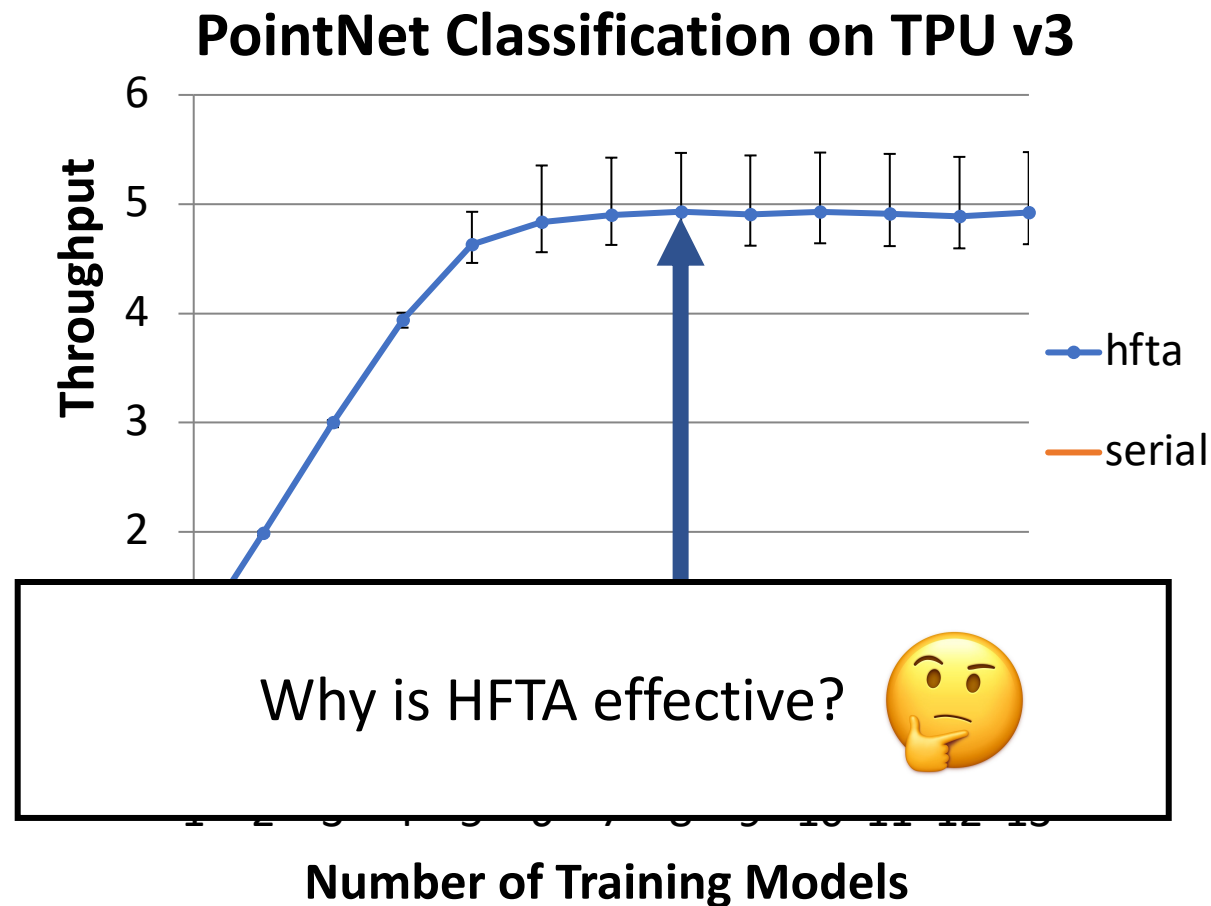


## PointNet Classification on A100



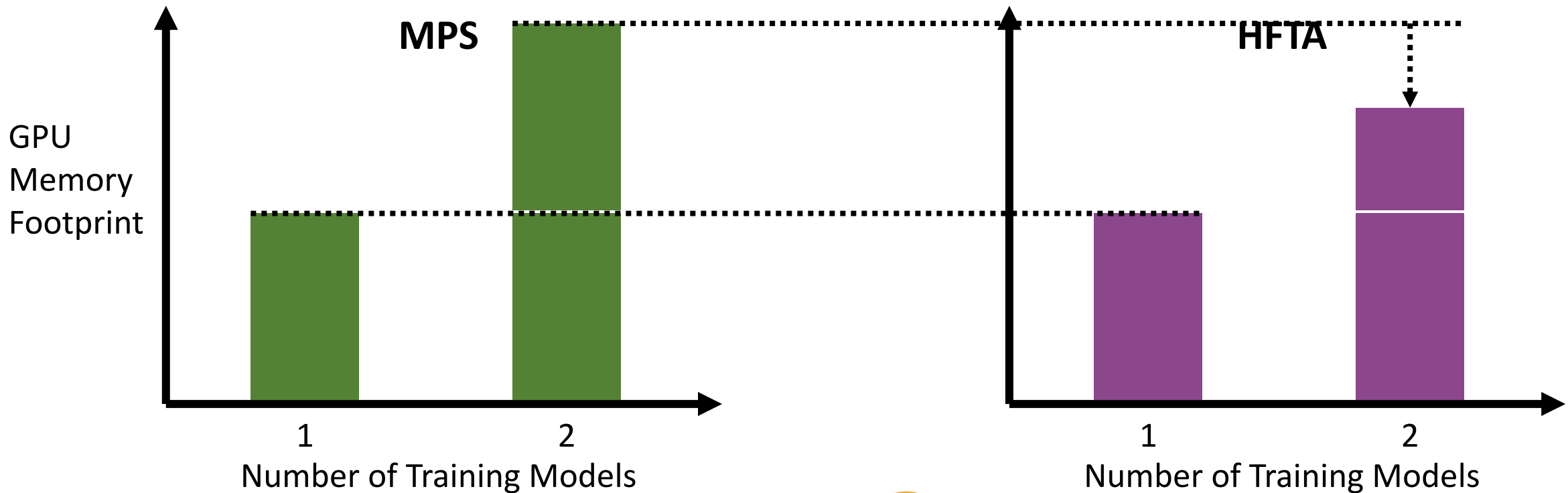
# How About TPUs?

HFTA achieves **4.93×** over **Serial**.





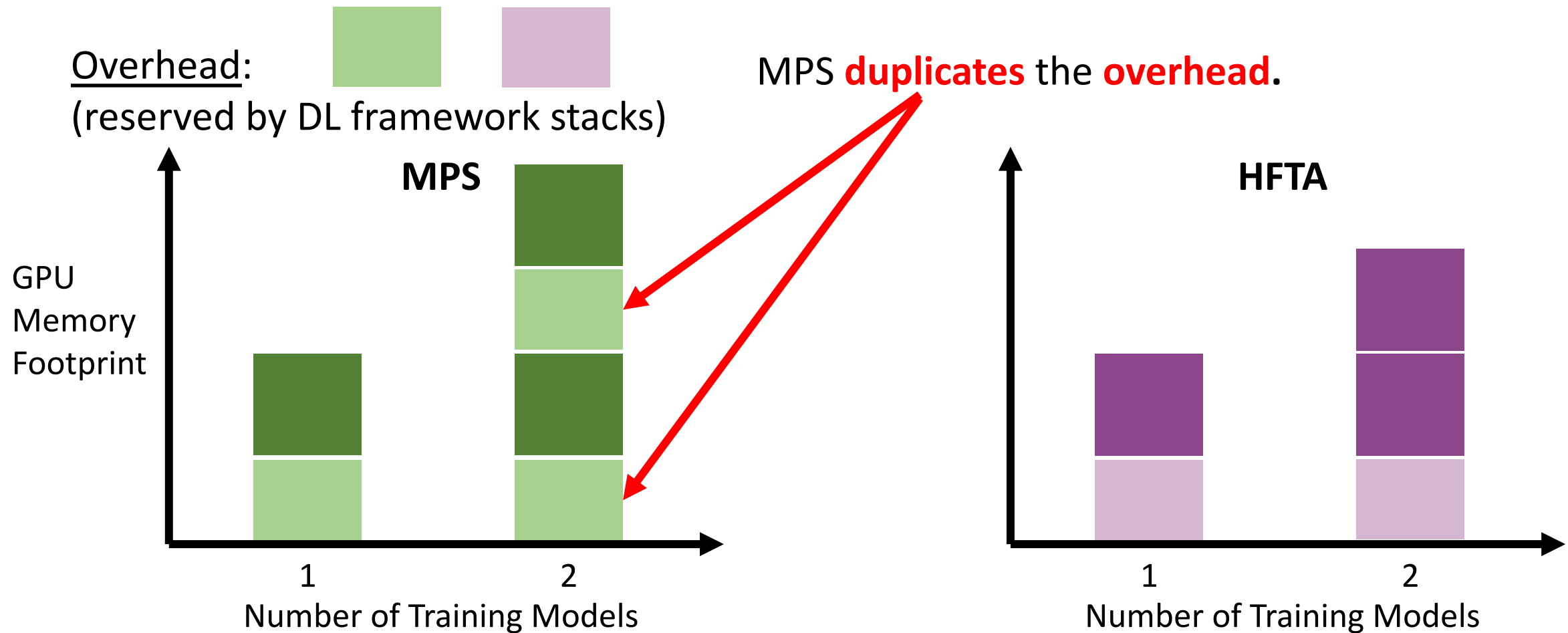
# Performance Analysis: Memory



Why?



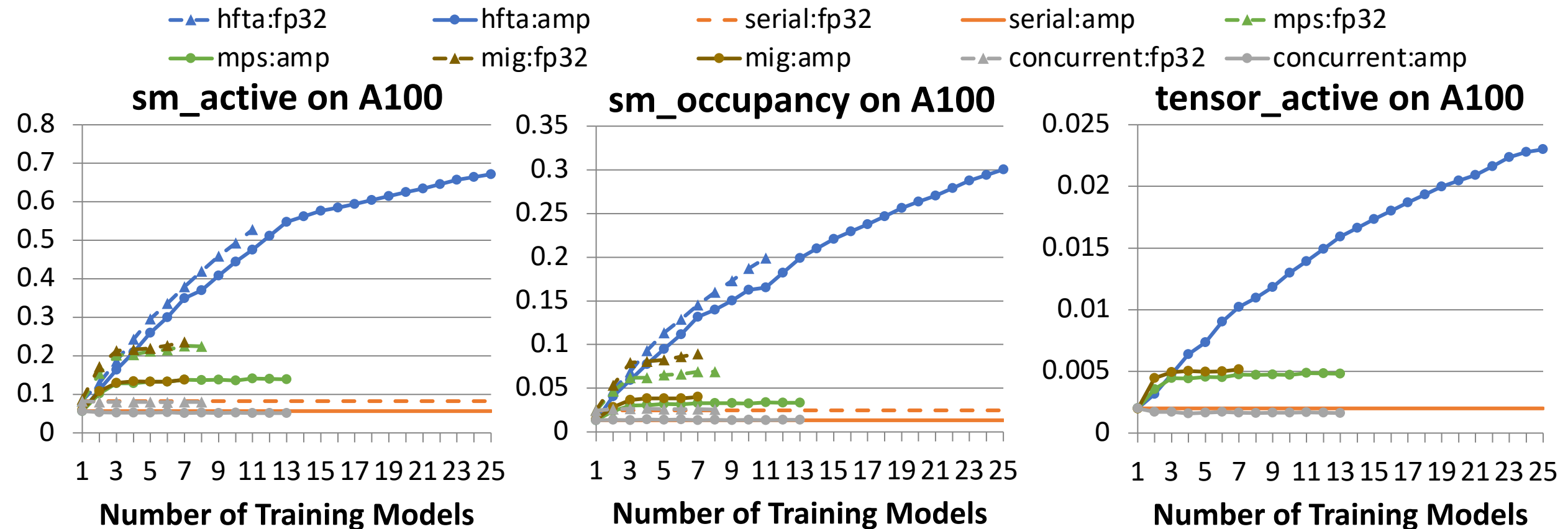
# Performance Analysis: Memory



Any other reason why HFTA is effective?



# Performance Analysis: Compute



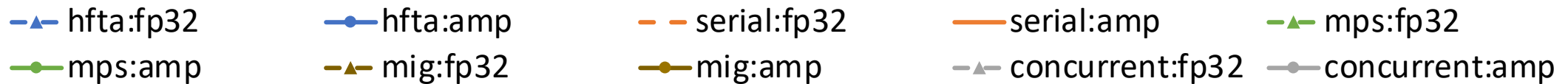
# Performance Analysis: Compute

sm\_active: Fraction of cycles when SMs have resident warps.

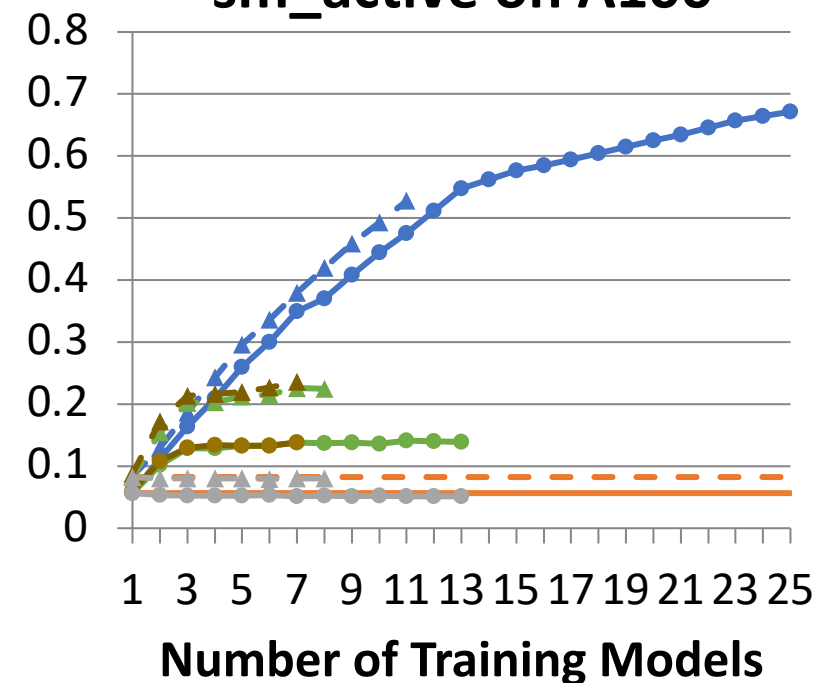
sm\_occupancy: Ratio of # resident warps over SM's max. # warps.

tensor\_active: Fraction of cycles when tensor cores are active.

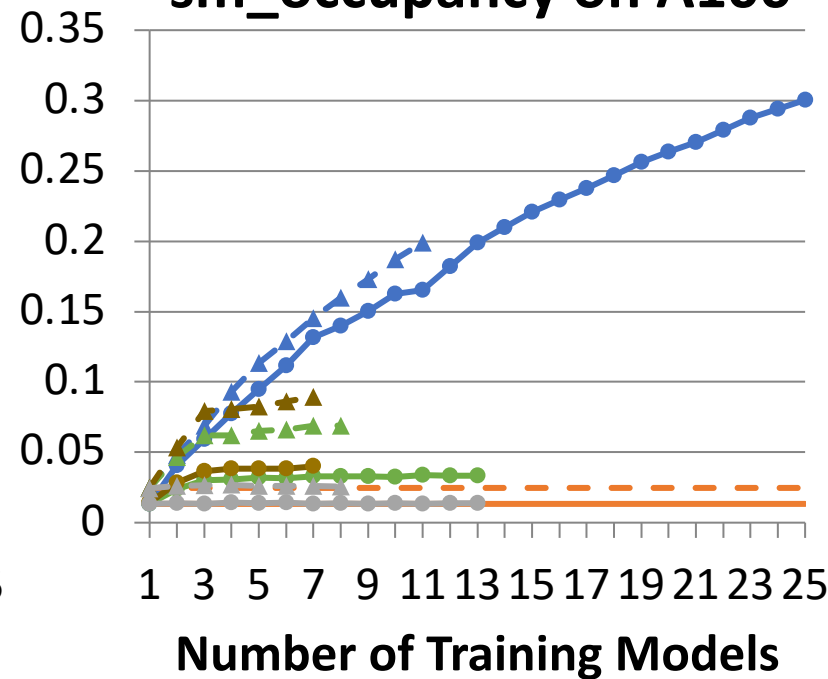
Proxy metrics for different aspects of GPU utilization.



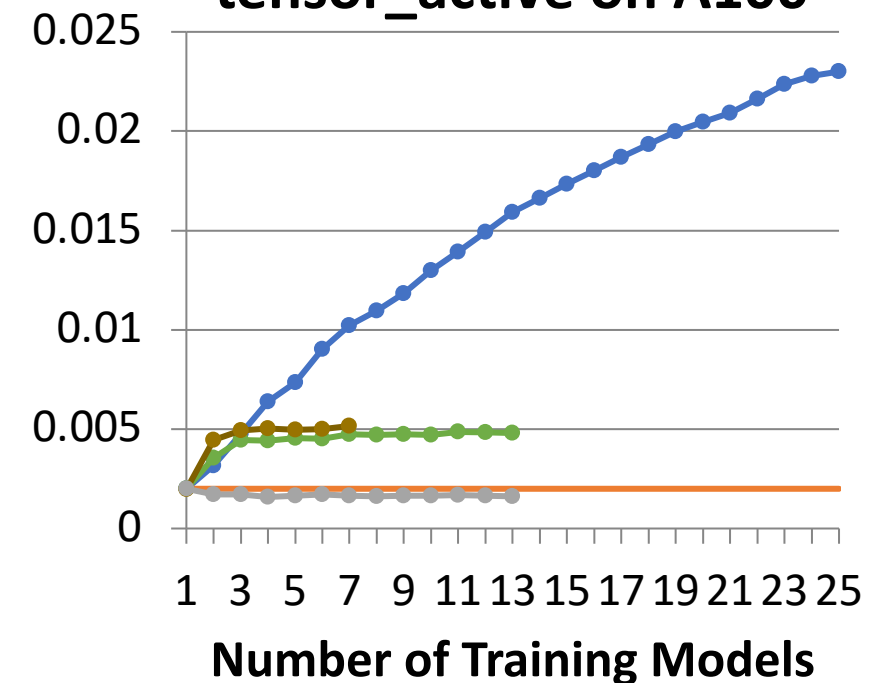
### sm\_active on A100



### sm\_occupancy on A100

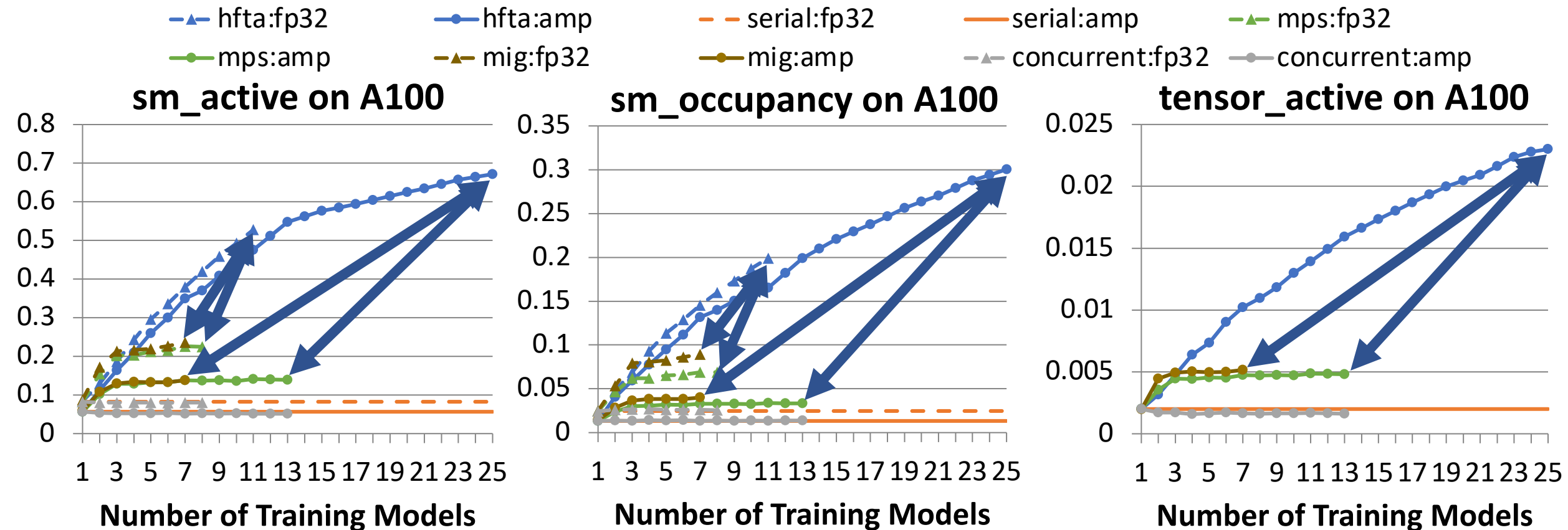


### tensor\_active on A100



# Performance Analysis: Compute

While MPS & MIG **does** improve utilization, HFTA is **more effective**!



# More Results in the Paper

PointNet Segmentation, DCGAN, ResNet-18, MobileNet<sub>v3Large</sub>, Transformer, BERT<sub>Medium</sub>

- On GPUs, HFTA achieves:
  - **2.42×** to **11.50×** over **Serial**.
  - **1.25×** to **4.72×** over **MPS**.
  - **1.33×** to **4.88×** over **MIG**.
- On TPUs, HFTA achieves **2.98×** to **15.13×** over **Serial**.

HFTA's Integration with hyper-parameter tuning algorithms.

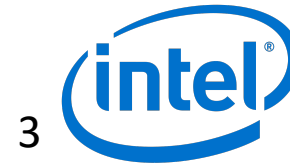
- Reduce total GPU hour cost by up to **5.10×**.

Performance sensitivity study on partially fused ResNet-18.



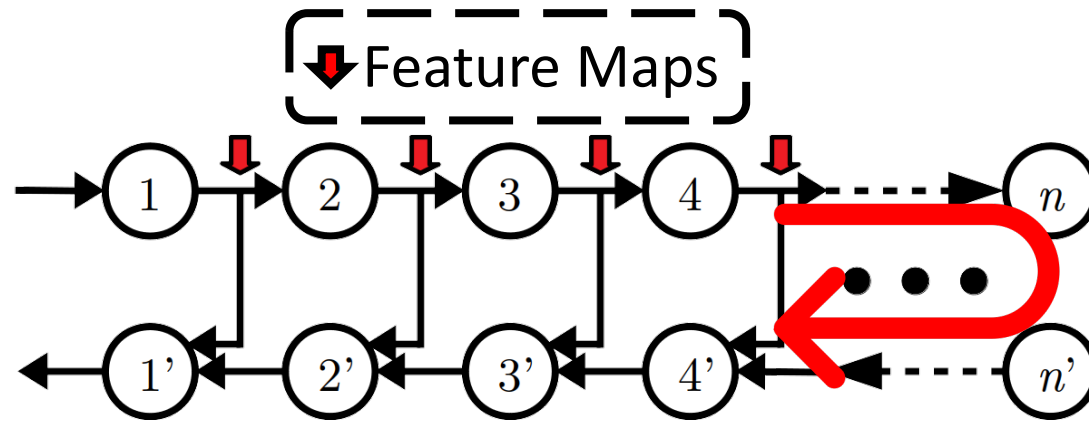
# ECHO: Compiler-based GPU Memory Footprint Reduction for LSTM RNN Training

Bojian Zheng<sup>1,2</sup>, Nandita Vijaykumar<sup>1,3</sup>, Gennady Pekhimenko<sup>1,2</sup>



# Background: Feature Maps

- Stashed data by the forward pass to compute the backward gradients



Large **Temporal Gap** between Usage

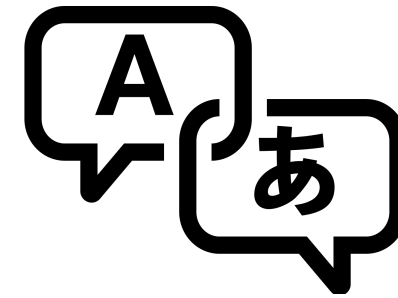
- The cause of high memory footprint in Convolutional Neural Networks (CNNs).<sup>[1, 2]</sup>

[1] M. Rhu et al. *vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design*. MICRO 2016

[2] A. Jain et al. *Gist: Efficient Data Encoding for Deep Neural Network Training*. ISCA 2018



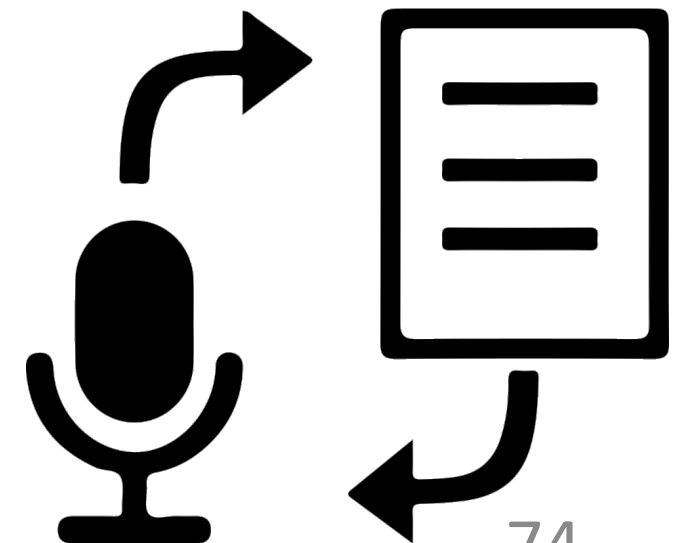
# Background: LSTM RNN



Neural Machine Translation (NMT)

- Long-Short-Term-Memory Recurrent Neural Network (LSTM RNN)
- Heavily adopted in sequence analysis (e.g., machine translation (NMT) & speech recognition (DeepSpeech2)).
- Its **training** is **inefficient** on the **GPUs**, especially when compared with CNN.<sup>[1, 2]</sup>

DeepSpeech2

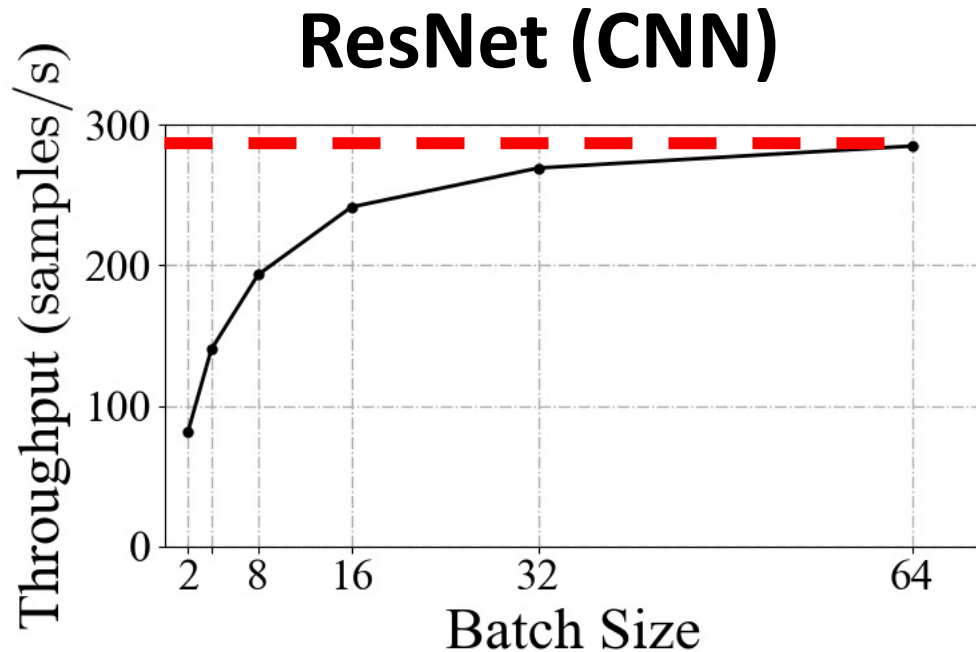


[1] J. Bradbury et al. *Quasi-Recurrent Neural Networks*. ICLR 2016

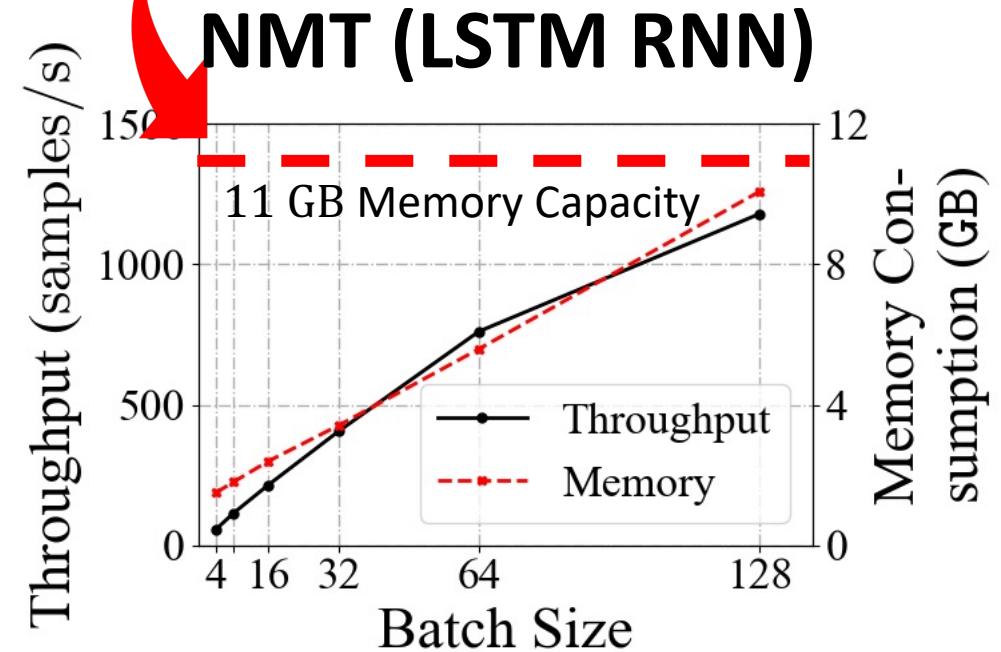
[2] T. Lei et al. *Simple Recurrent Units for Highly Parallelizable Recurrence*. EMNLP 2018

# Why LSTM RNN Training is Inefficient?

Training throughput **saturates** as batch size increases.



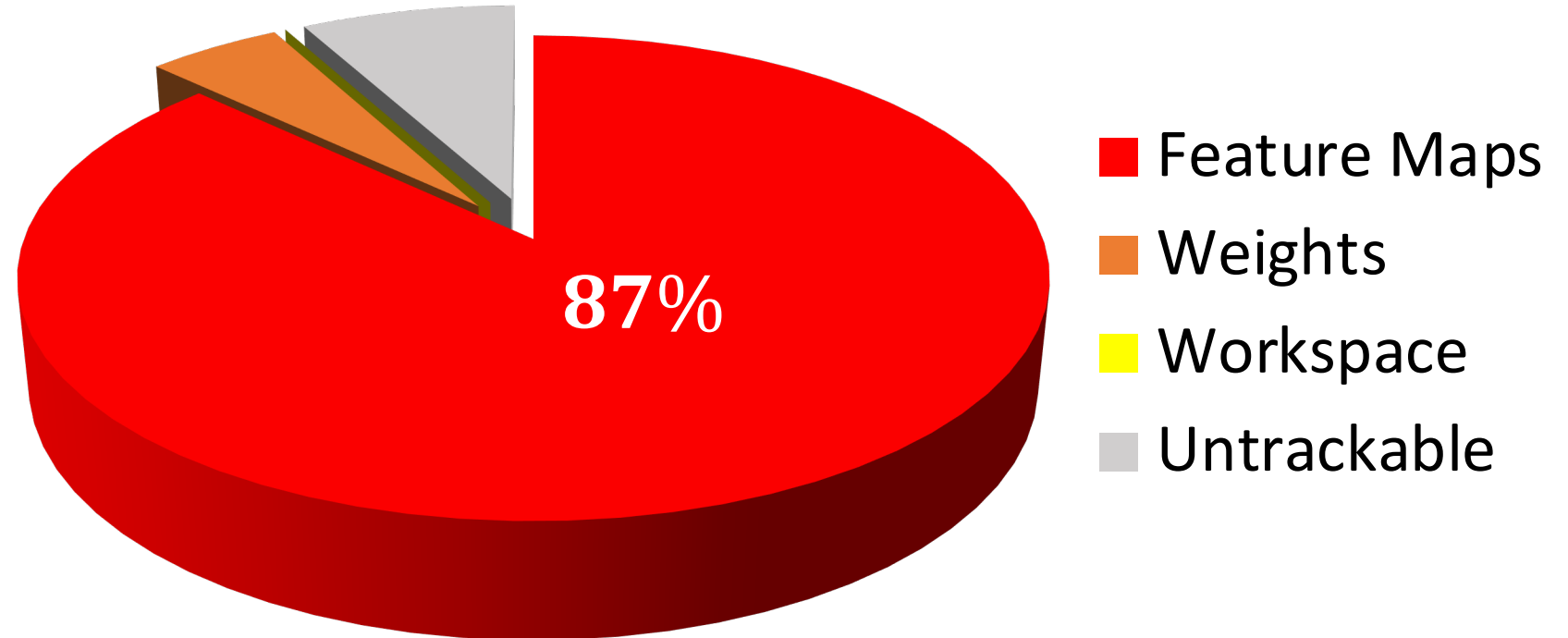
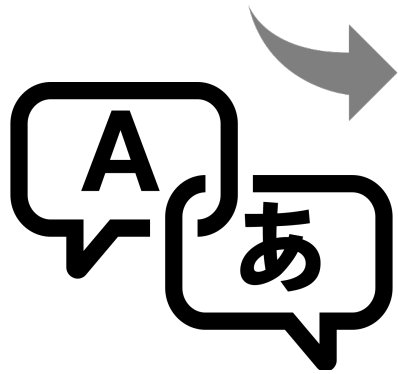
Training throughput is limited by the **memory capacity**.



**Memory capacity** limits the NMT training throughput.

# GPU Memory Profiling Results

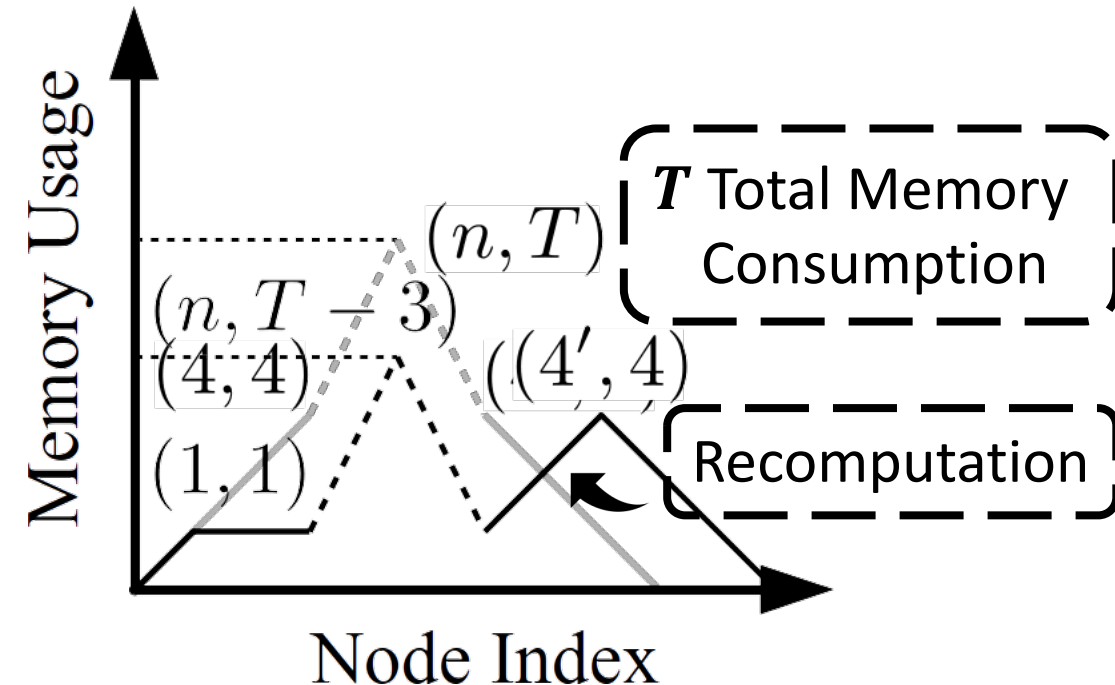
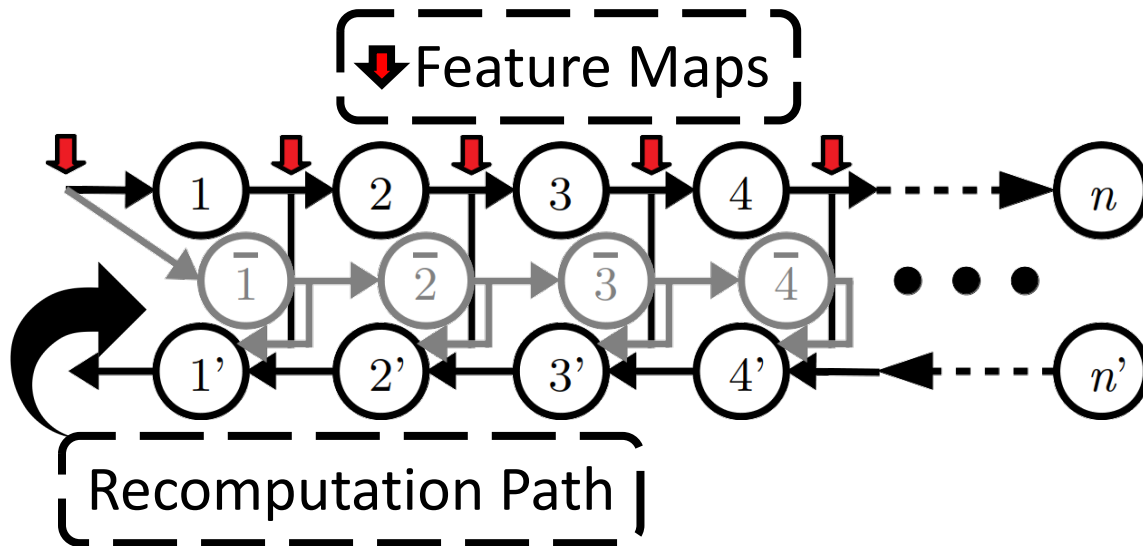
MXNet GPU Memory Profiler



**Feature maps** dominate the GPU memory footprint.

# Selective Recomputation

- Key Idea: Trade **runtime** with **memory**.



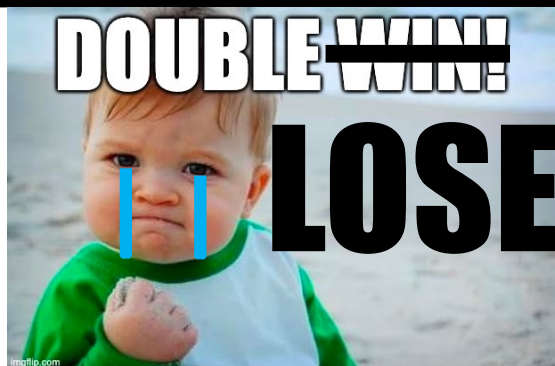
- The recomputation path should only involve **lightweight** operators.

# 1 Accurate Footprint Estimation

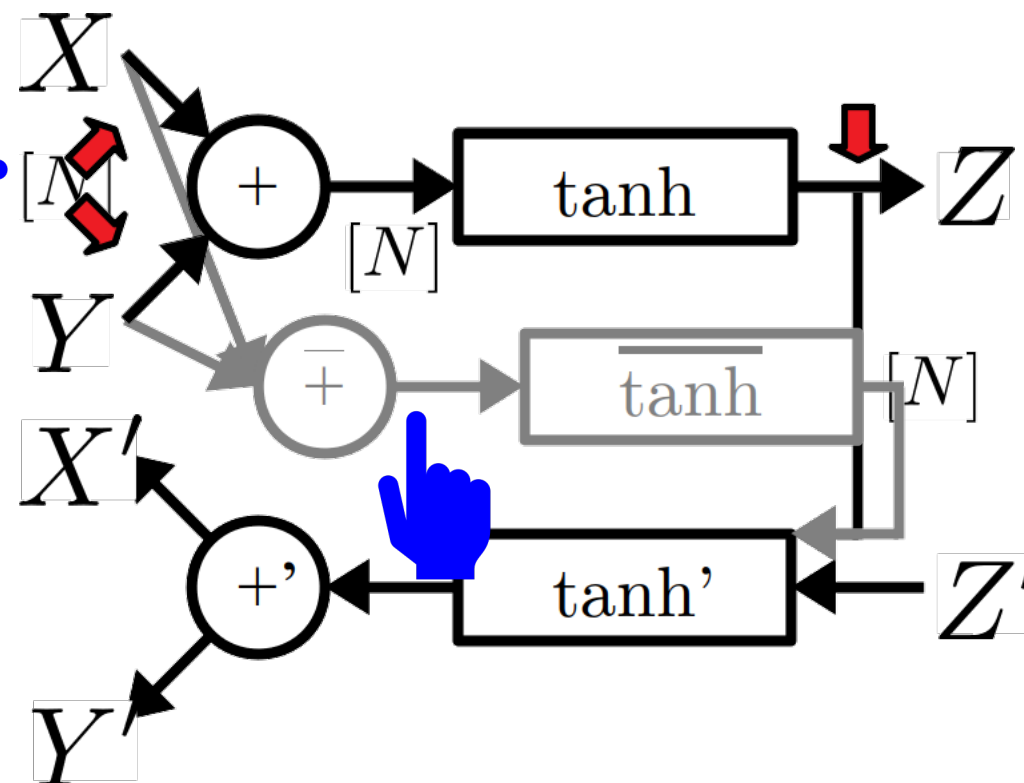
For each recomputation to be efficient, need to estimate its effect on the **global footprint**.

Selective Recomputation causes:

- (-) increased memory footprint &
- (-) performance degradation!



Example:  $Z = \tanh(X + Y)$



# 1 Accurate Footprint Estimation

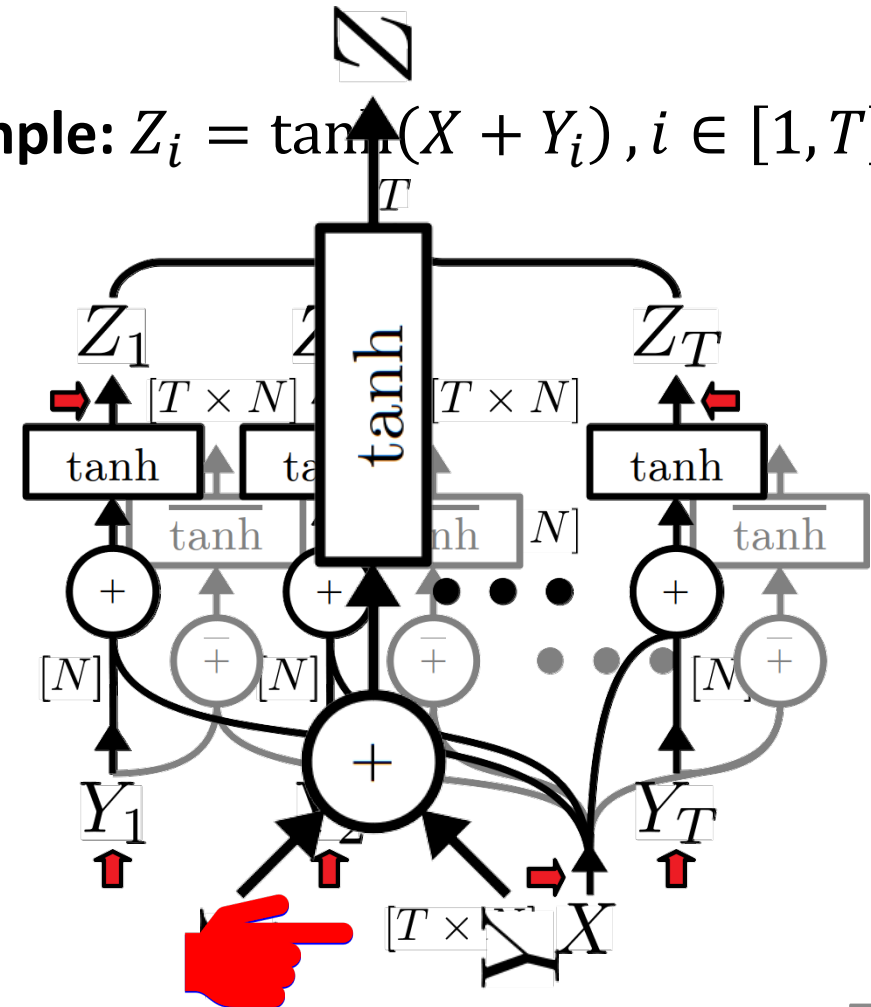
For each recomputation to be efficient, need to estimate its effect on the **global footprint**.

Selective Recomputation causes:  
(+) feature maps:  $T^2 N \rightarrow 2TN$

Global Footprint Analysis:

1. shapes and types
2. reuse **Challenging!**

Example:  $Z_i = \tanh(X + Y_i), i \in [1, T]$



## ② Non-Conservative Overhead Estimation

For each recomputation to be efficient, need to estimate its effect on the **runtime overhead**.

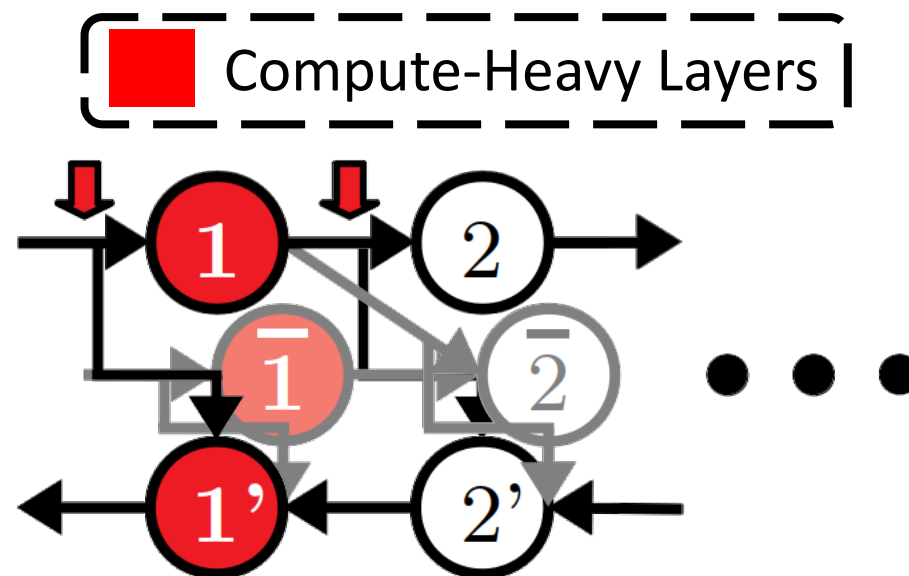
**Layer-Specific Property:**

$$\frac{dE}{dX} = \frac{dE}{dY} W \quad \& \quad \frac{dE}{dW} = \frac{dE^T}{dY} X$$

(NO Dependency on  $Y$ )

Example:  $Y = XW^T$

- **Compute-Heavy**
  - 50% of the NMT training time
- Excluded in prior works



# ECHO: A Graph Compiler Pass

- Integrated in the MXNet NNVM<sup>[1]</sup> module
- Fully **Automatic & Transparent**
  - Requires NO changes in the training source code.
- Addresses the 2 key challenges of Selective Recomputation:
  - 1 Accurate Footprint Estimation
    - 👉 *Bidirectional Dataflow Analysis*
  - 2 Non-Conservative Overhead Estimation
    - 👉 *Layer Specific Optimizations*

[1] <https://github.com/apache/incubator-mxnet/tree/master/src/nnvm>



# ECHO: Bidirectional Dataflow Analysis

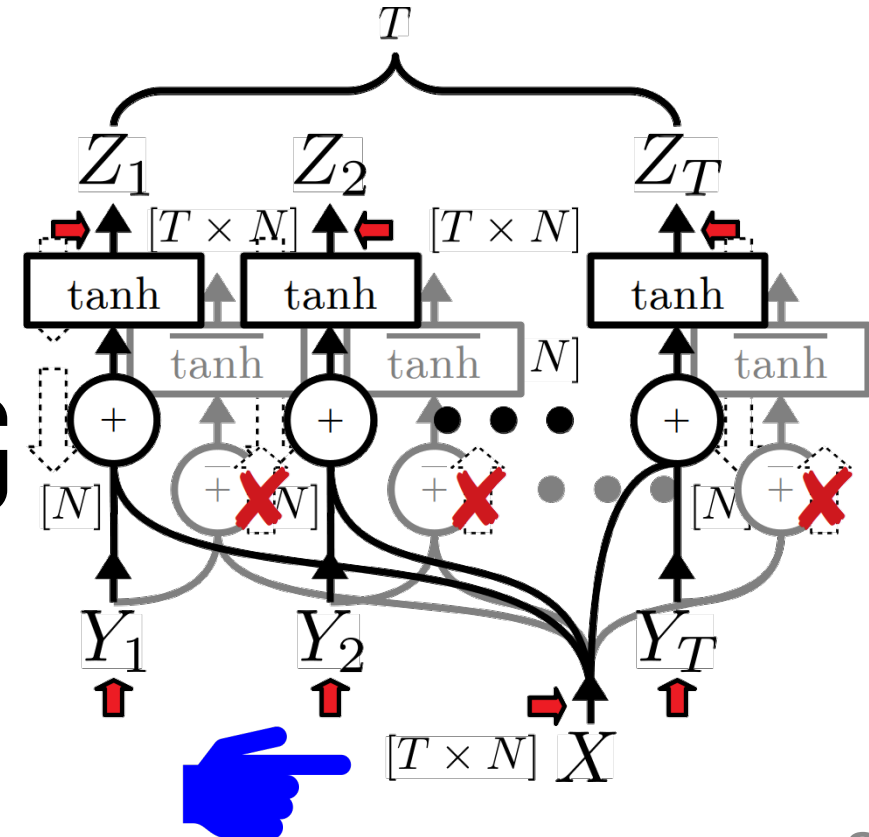
- **Storage Reuse**

Causes ALL correlated operators to forward propagate simultaneously.

$$\text{sizeof} \left( \sum \text{FeatureMaps}_{\text{new}} \right) \leq \text{sizeof} \left( \sum \text{FeatureMaps}_{\text{old}} \right)$$

$$[T^2 N \not\leq 2TN]$$

Example:  $Z_i = \tanh(X + Y_i), i \in [1, T]$



# Evaluation: Benchmarks

## Sockeye<sup>[1]</sup>

[1] F. Hieber et al. *Sockeye: A Toolkit for Neural Machine Translation*.  
Arxiv Preprint 2017



- State-of-the-Art Neural Machine Translation Toolkit under MXNet
- Datasets:
  - IWSLT'15 English-Vietnamese (Small)
  - WMT'16 English-German (Large)
- Key Metrics:
  - Training Throughput
  - GPU Memory Consumption
  - Training Time to Validation BLEU Score

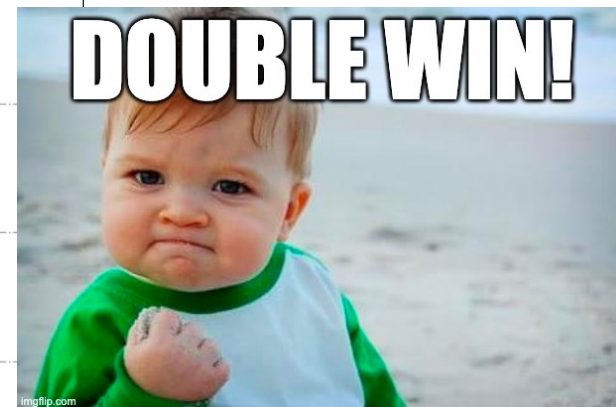
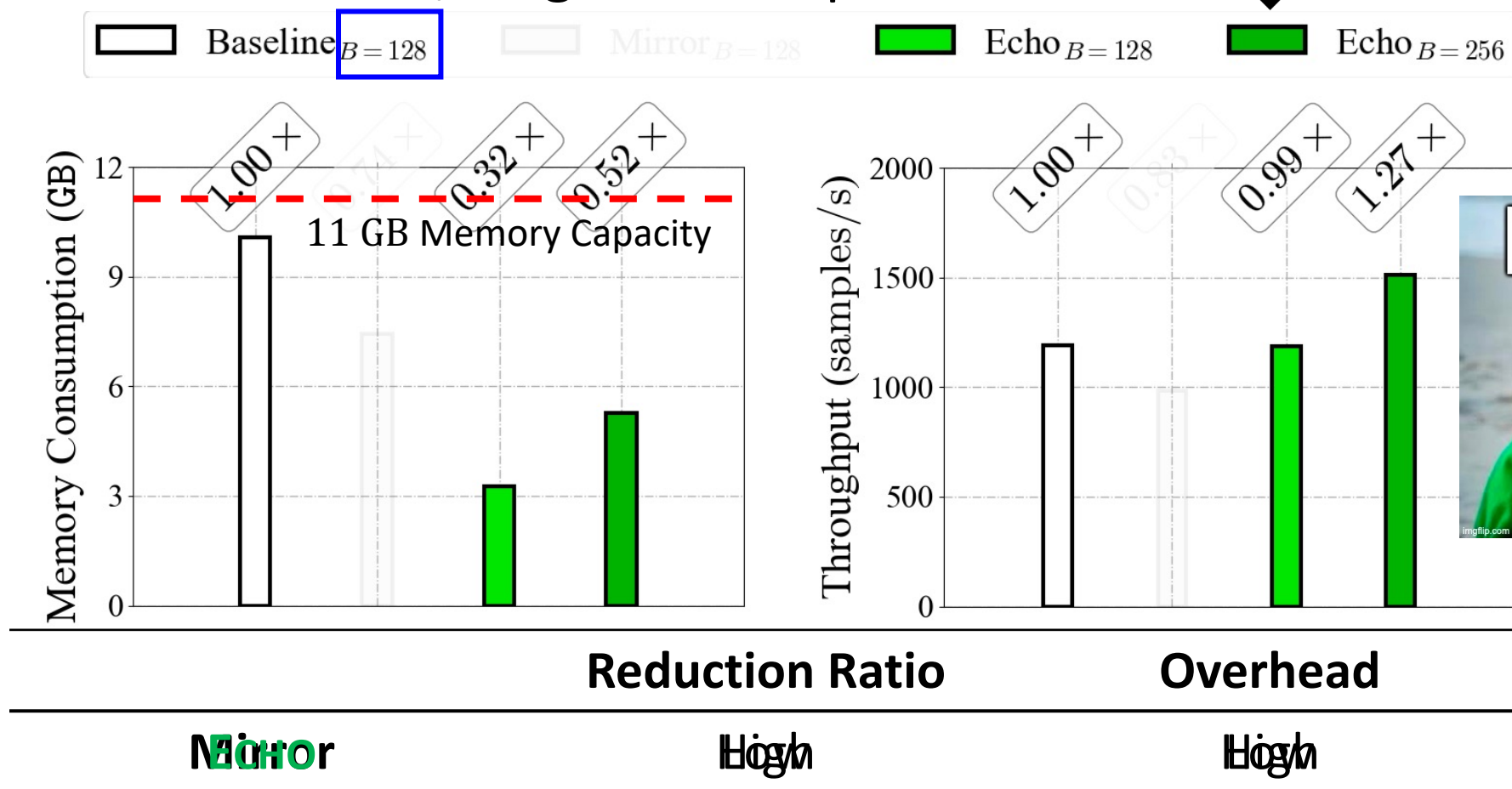
# Evaluation: Systems

<b>Baseline</b>	Baseline System without Selective Recomputation
<b>Mirror</b>	T. Chen et al. <sup>[1]</sup> <small>[1] T. Chen et al. <i>Training Deep Nets with Sublinear Memory Cost</i>. Arxiv Preprint 2016</small>
<b>ECHO</b>	Compiler-based Automatic and Transparent Optimizations

# ECHO's Effect on Memory and Performance

Small Dataset, Single-GPU Experiment

2× Training Batch Size

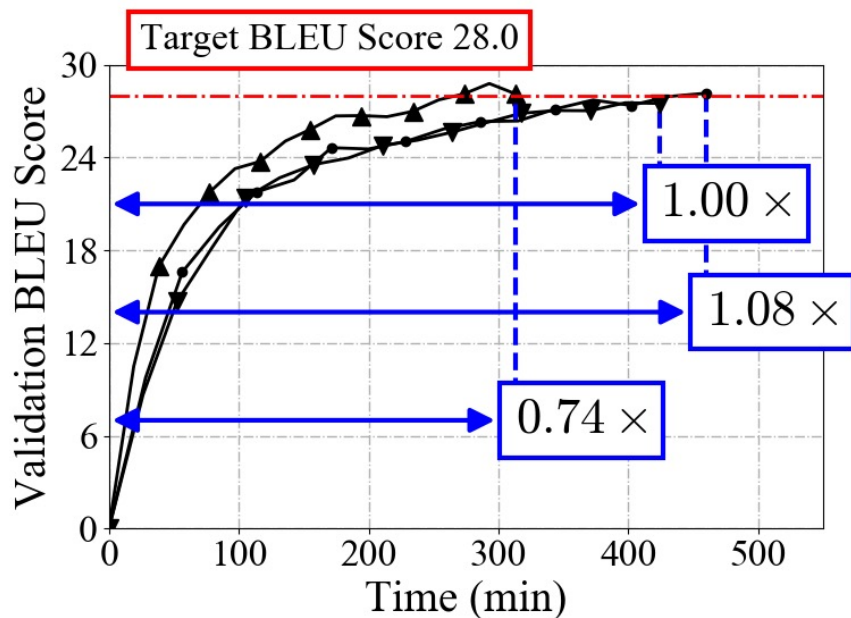
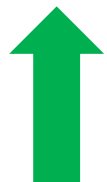


# ECHO's Effect on Training Convergence

Large Dataset, Multi-GPU Experiment,  
Same Number of Training Steps

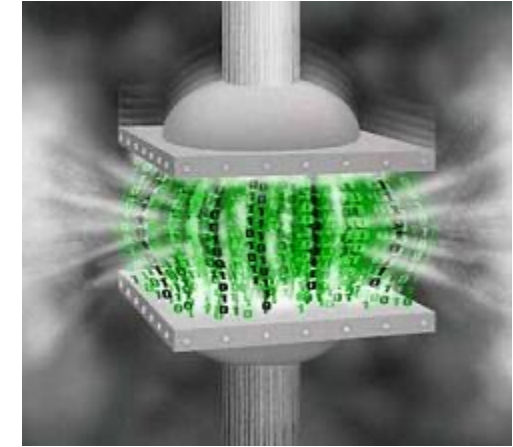
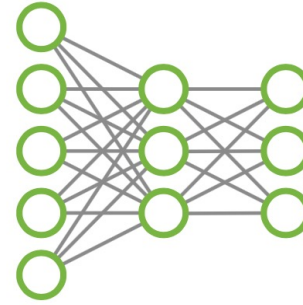


Better



**ECHO** achieves:

- + Same Validation BLEU Score
- + Faster Convergence
- + Fewer Compute Devices



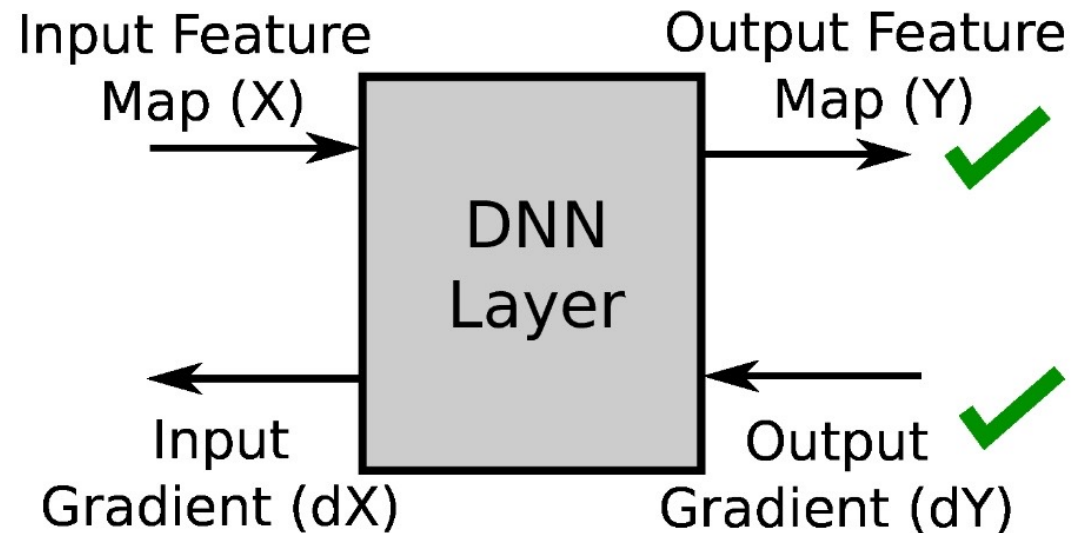
# Gist: Efficient Data Encoding for Deep Neural Network Training

*In collaboration with Project Fiddle (MSR)*



# Relu -> Pool

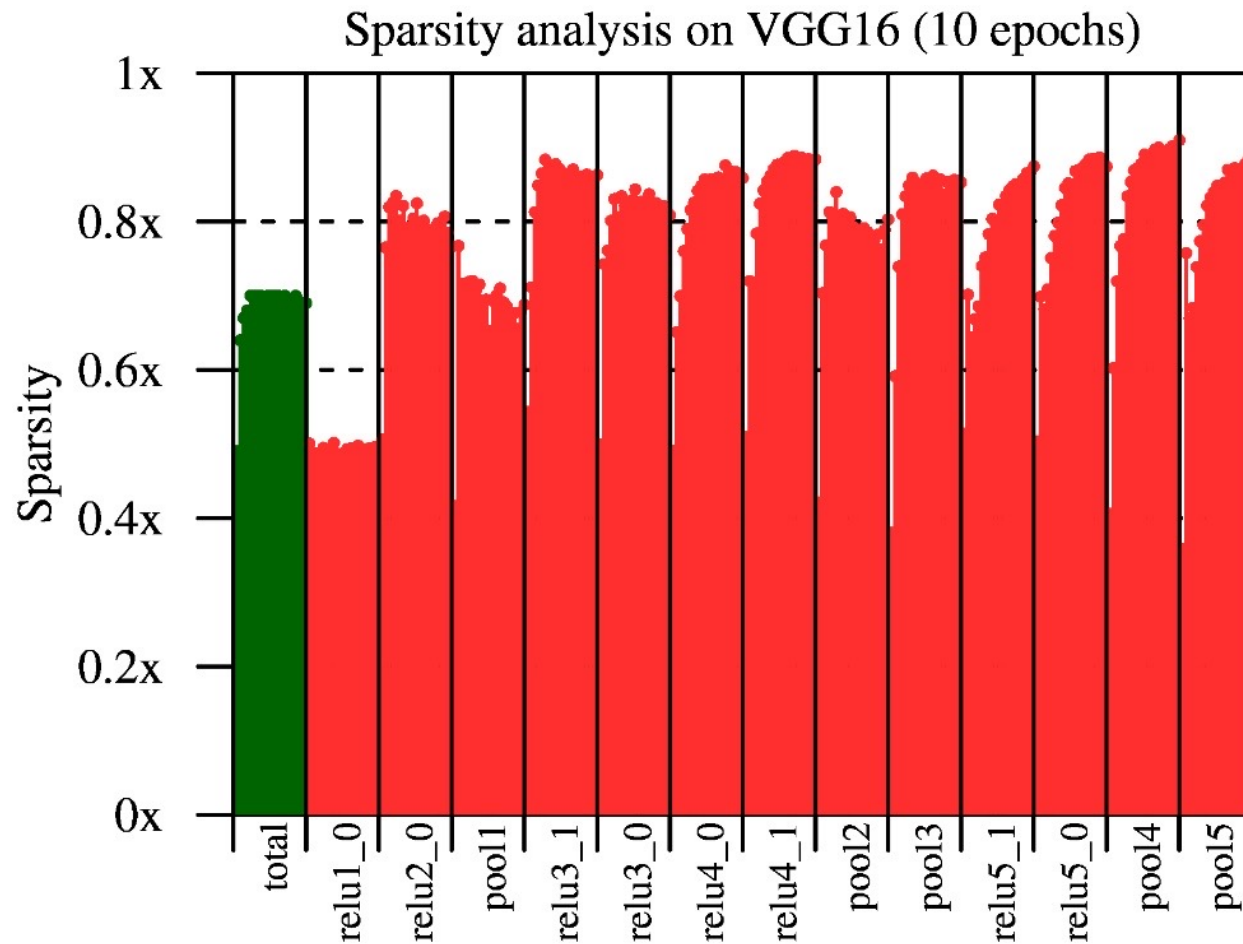
## Relu Backward Propagation



$$dX = f(Y, dY)$$
$$dx = y > 0 ? dy : 0;$$

**Binarize – 1 bit representation**  
**(Lossless)**

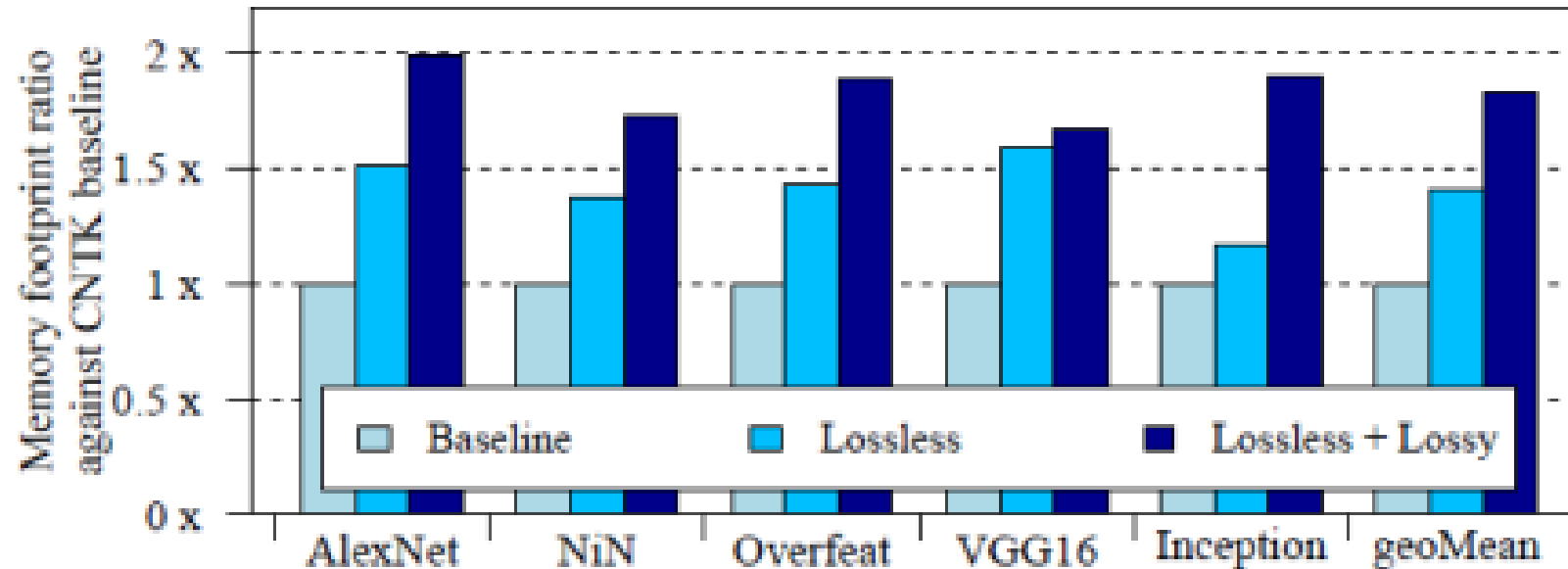
# Relu/Pool -> Conv



**Sparse Storage Dense Compute**  
**(Lossless)**



# Compression Ratio



**Up to 2X compression ratio**  
**With minimal performance overhead**

# Gist Summary

- Systematic **memory breakdown** analysis for image classification
- Layer-specific **lossless** encodings
  - **Binarization** and sparse storage/dense compute
- Aggressive lossy encodings
  - With **delayed precision reduction**
- Footprint reduction measured on real systems:
  - Up to **2X** reduction with only 4% performance overhead
  - Further optimizations – more than **4X** reduction

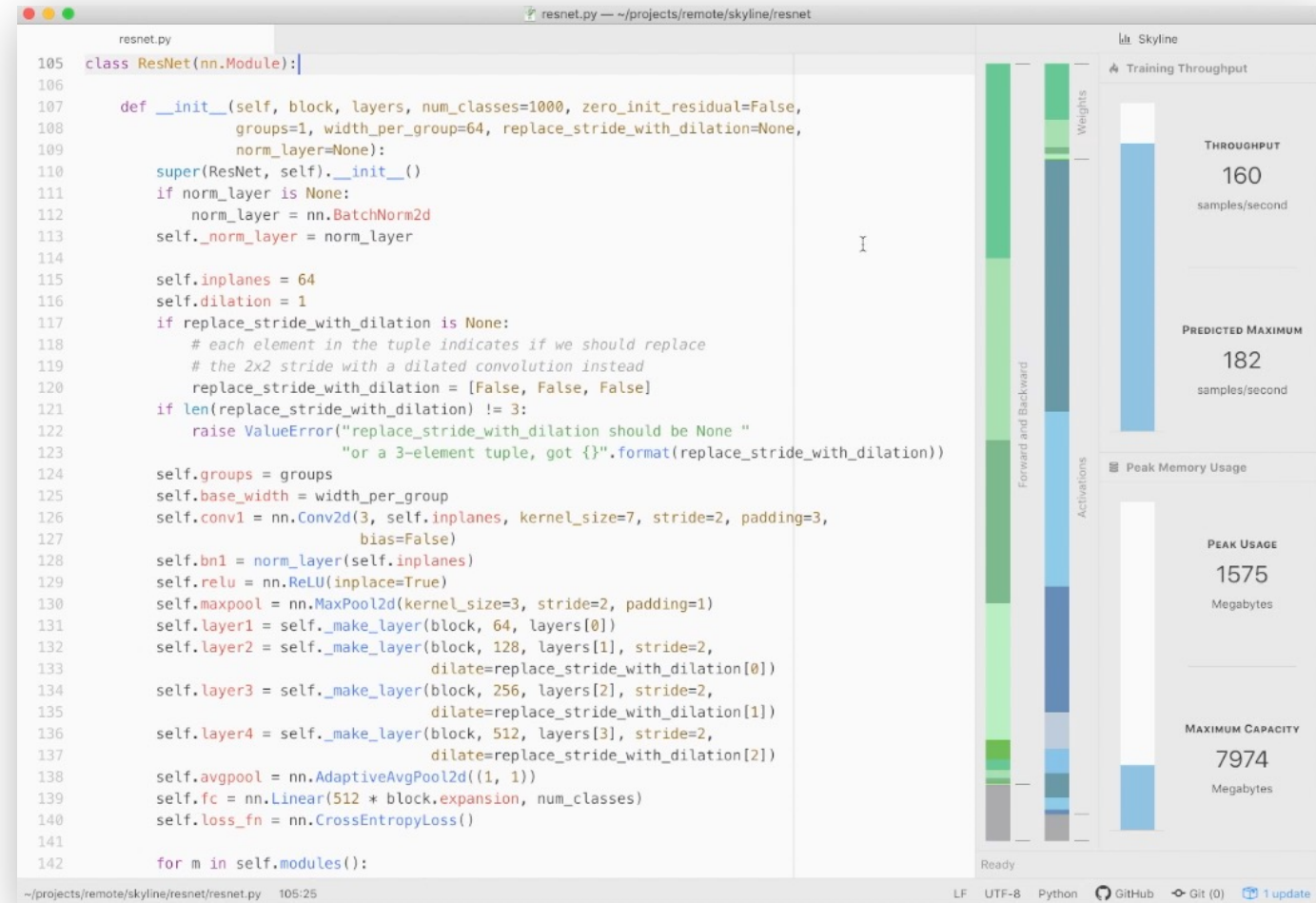
# New Generation of Debugging/Prediction Tools

- **Daydream**: Accurately Estimating the Efficacy of Performance Optimizations for DNN Training (**USENIX ATC'20**)
- **Skyline**: Interactive In-editor Performance Visualizations and Debugging for DNN Training (**UIST'20**)
- **Habitat**: Prediction-guided Hardware Selection for Deep Neural Network Training (**USENIX ATC'21**)



# Interactive In-editor Performance Visualizations and Debugging for DNN Training

Geoffrey X. Yu, Tovi Grossman,  
Gennady Pekhimenko



Tired of not knowing why your model is  
**slow** and/or **uses up so much memory**?




**Sam Bowman**  
@sleepinyourhat

Any tips on identifying speed bottlenecks (profiling) with @PyTorch? Right now bumbling along with cProfile.

28 12:16 PM - May 26, 2017

[See Sam Bowman's other Tweets](#)



**Hal Daumé III**  
@haldaume3


pytorch code is slow. what do people use for profiling? It just tells me run\_backward is expensive, which is not s...

3:47 PM - May 7, 2017



**Sam Bowman** @sleepinyourhat · May 26, 2017

Any tips on identifying speed bottlenecks (profiling) with @PyTorch? Right now bumbling along with cProfile.



**Zico Kolter**  
@zicokolter

Interperse torch.cuda.synchronize() liberally when debugging cuda code, to see where the bottlenecks actually are...


4 3:09 PM - May 27, 2017

[See Zico Kolter's other Tweets](#)



**Sam Bowman** @sleepinyourhat · May 26, 2017

Any tips on identifying speed bottlenecks (profiling) with @PyTorch? Right now bumbling along with cProfile.




**Joachim Hagege**  
@JoachimHagege

Hi Sam. I'm struggling with same issue right now. Did you identify best practices since posting? Thanks!

10:32 AM - Nov 11, 2018

[See Joachim Hagege's other Tweets](#)



**Jeremy Howard**  
@jeremyphoward


Does anyone have any detailed tips, walkthrus, or tutorials on how to profile @PyTorch code running on the GPU?

I'm trying to optimize efficientnet and want to see exactly where the time is spent.

312 10:29 AM - Oct 25, 2019

62 people are talking about this

## Advice for debugging slow backward pass



**mrdroz dov** Andrew Drozdov

I am working with a recursive neural network where the forward pass takes roughly 2s on average, and the backward pass closer to 7 or 8s. Does this sound like normal behavior? I wonder what I could be doing which is causing such a slowdown.

I have a lot of narrow/chunk/cat in the model. Could this be a factor?

Apr '17

13 people are talking about this

ion running very slow?: I have a large amount of training set, it is tiny pytorch code, I found the loss.backward() is slow, both score and target are slow.

Model time on GPU  
Model time on GPU  
Model time on GPU  
Model time on GPU  
Model time on GPU  
Model time on GPU  
Model time on GPU


dynamic attention  
I use two for loops

2019

[See other Tweets](#)

## Why is my script so slow?

### Profiling pytorch scripts?



**hughperkins**

I've written a pytorch script, and I'm looking to speed it up.

I've tried the following:

- use a c4.4xlarge, in cpu mode, instead of Mac OS X, in cpu mode on Mac 😞
- use an aws g2, in cuda mode => twice as fast as Mac laptop
- use an aws p2, in cuda mode => another 50% as fast as g2

Now at this point, I'm not sure which bits are slow

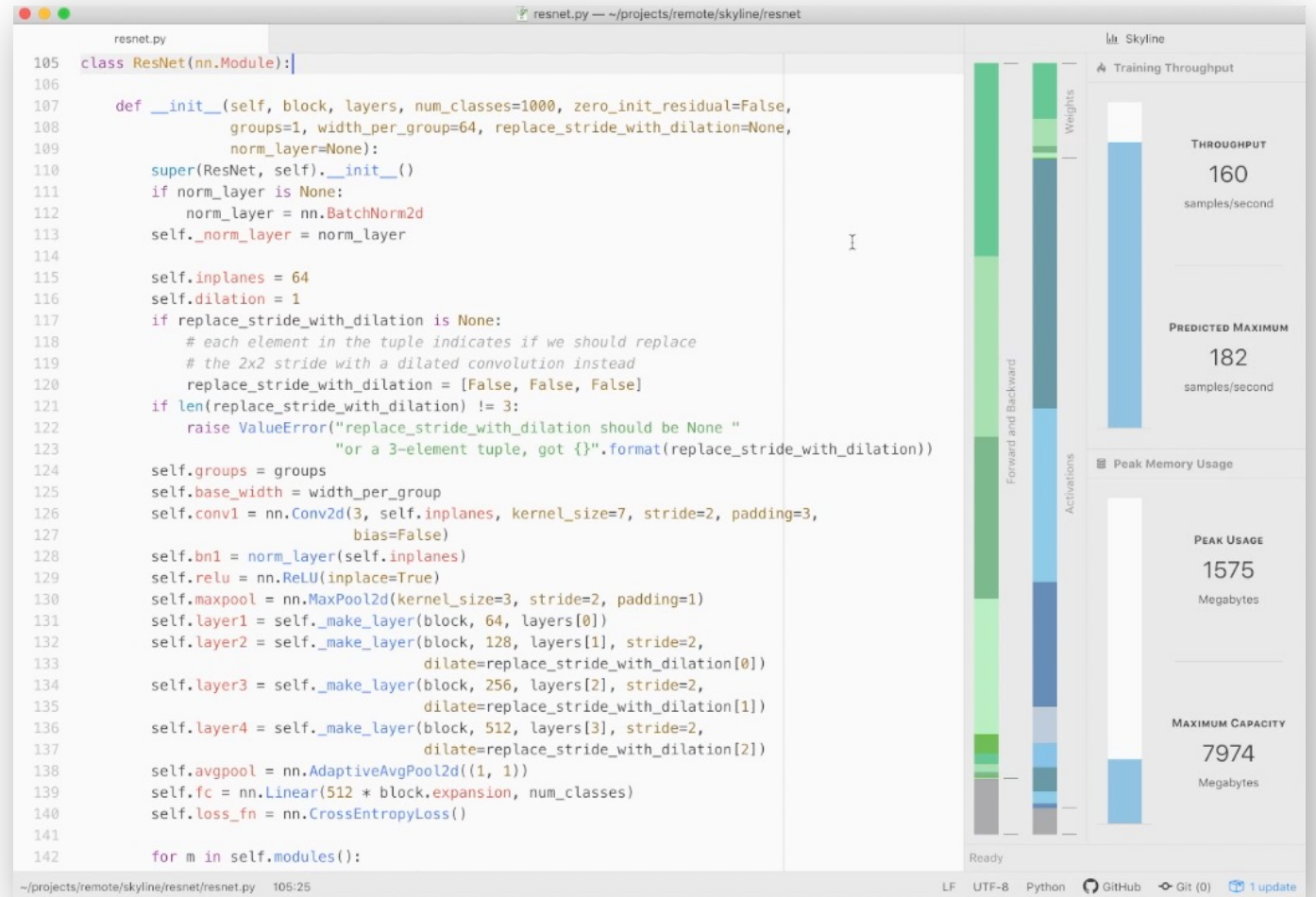
- If it was a c++ script, that didn't use cuda, I might use either gprof or valgrind, stop it, and store the stacktrace. do this eg 5-10 times, then look at the stacktraces => this is the bottleneck
- if it was cltorch, or deepcl, well I pre-instrumented them with timers
- in pytorch cuda, I suppose I should use an nvidia profiler?

It's not clear to me which bits of the program are taking the time, at a higher level than nvidia profiler probably. Thoughts on ideas for profiling pytorch?



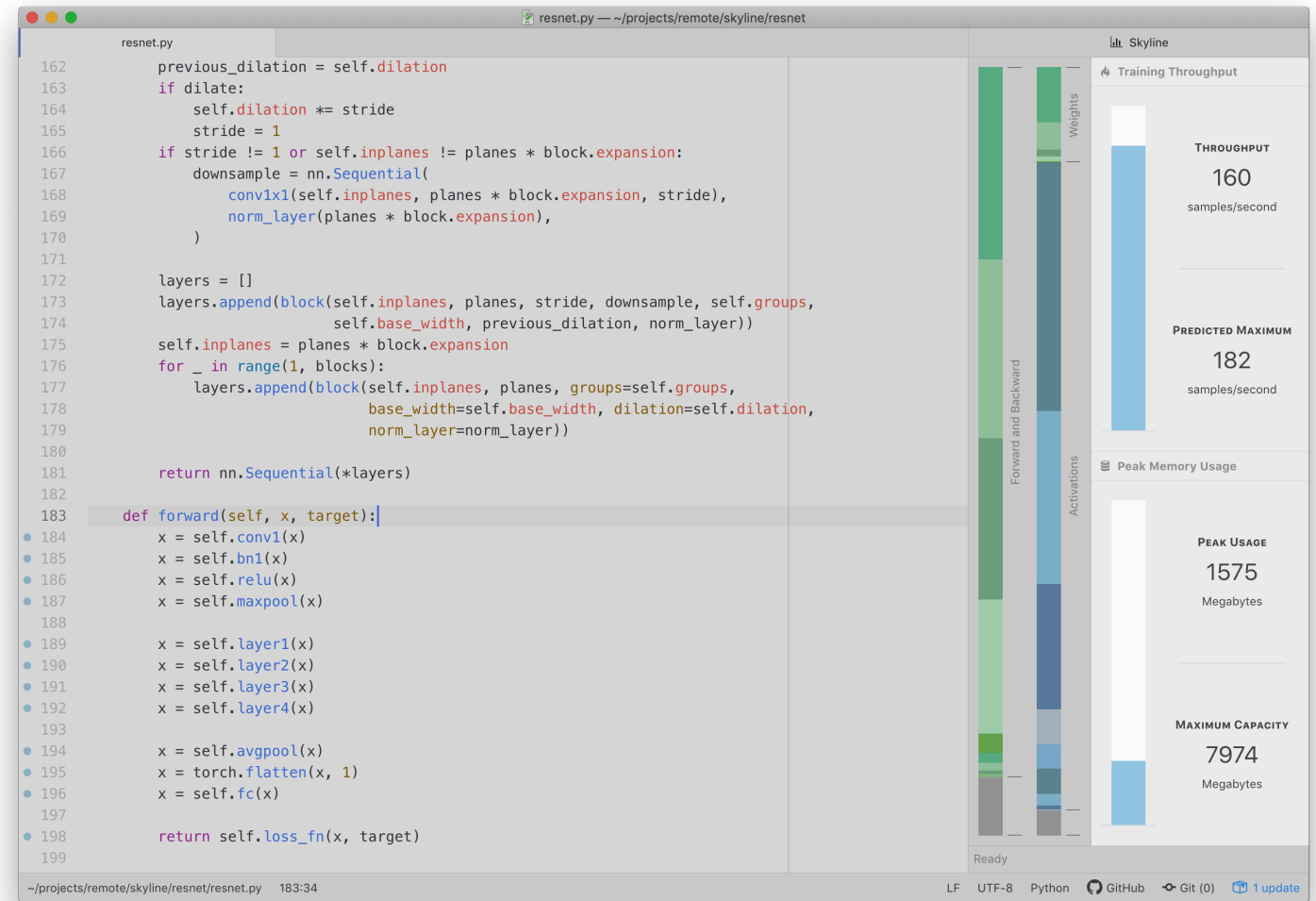
# Skyline: Interactive In-editor Performance Debugging

- Key performance metrics (throughput, memory usage)
- Iteration run time and memory footprint **breakdowns**
- **Interactive** visualizations linked to batch size predictions
- **Live and proactive** performance debugging during development



# Skyline: Interactive In-editor Performance Debugging

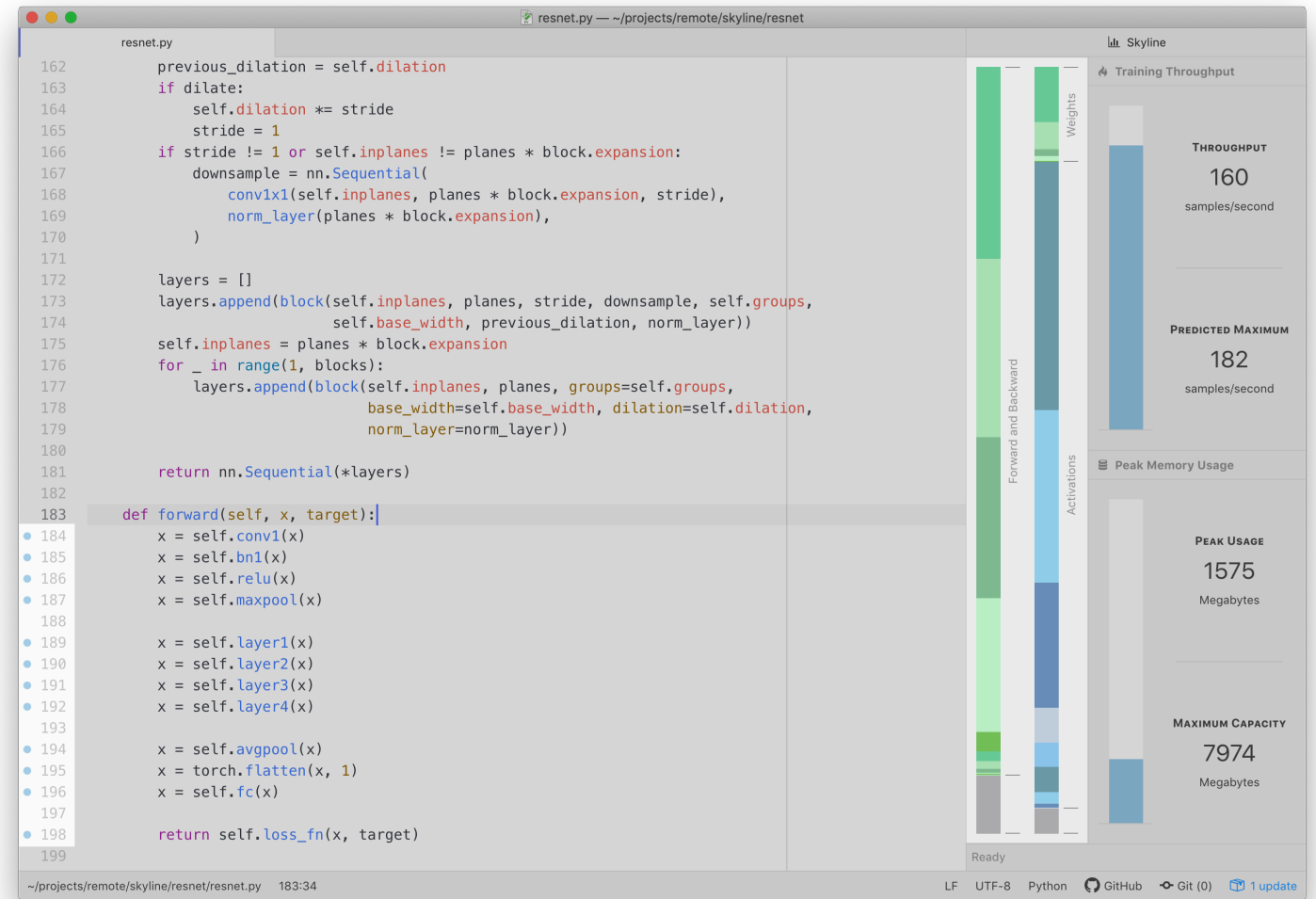
- Key performance metrics (throughput, memory usage)
- Iteration run time and memory footprint breakdowns
- Interactive visualizations linked to batch size predictions
- Live and proactive performance debugging during development





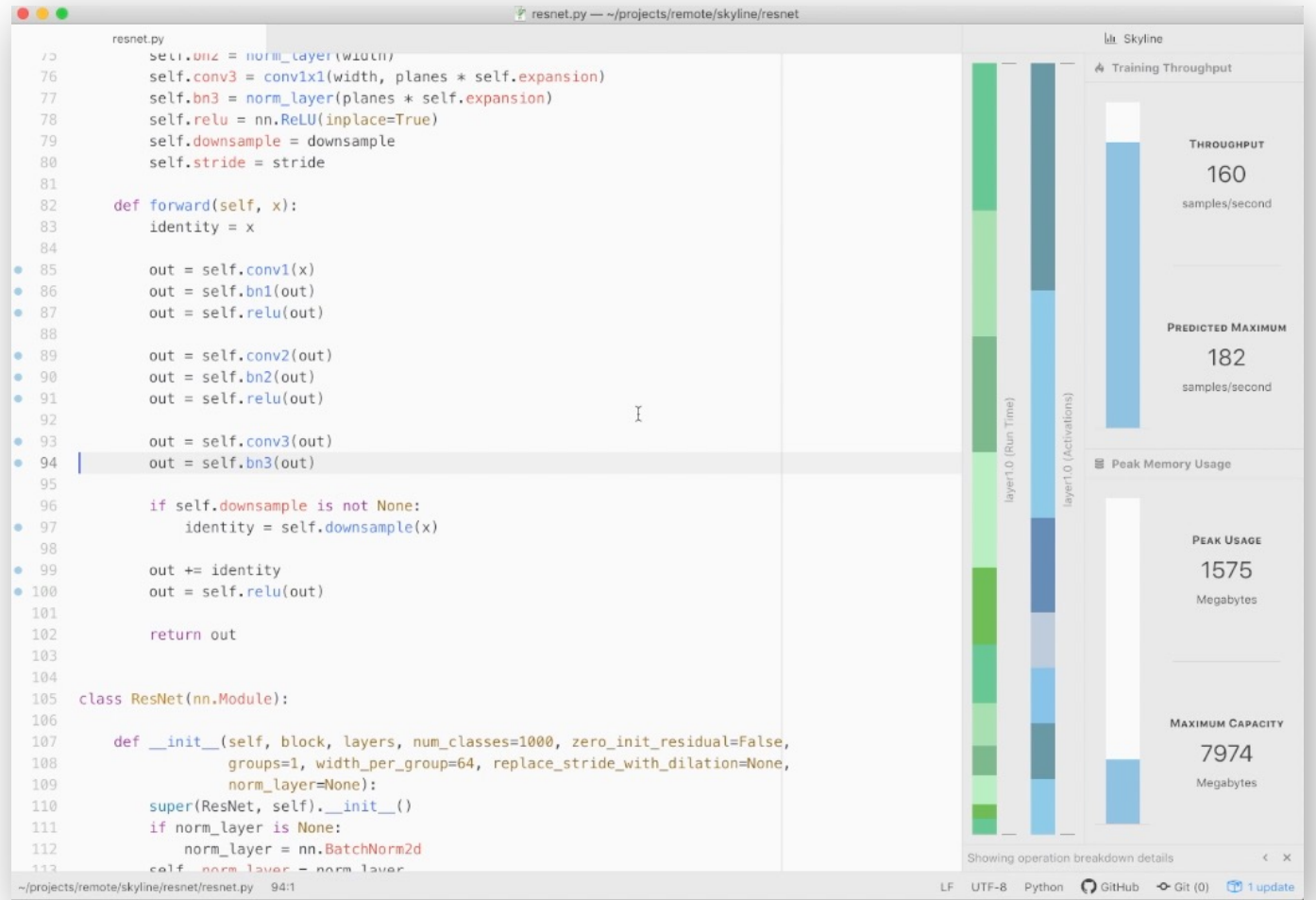
# Skyline: Interactive In-editor Performance Debugging

- Key performance metrics (throughput, memory usage)
- Iteration run time and memory footprint **breakdowns**
- Interactive visualizations linked to batch size predictions
- Live and proactive performance debugging during development



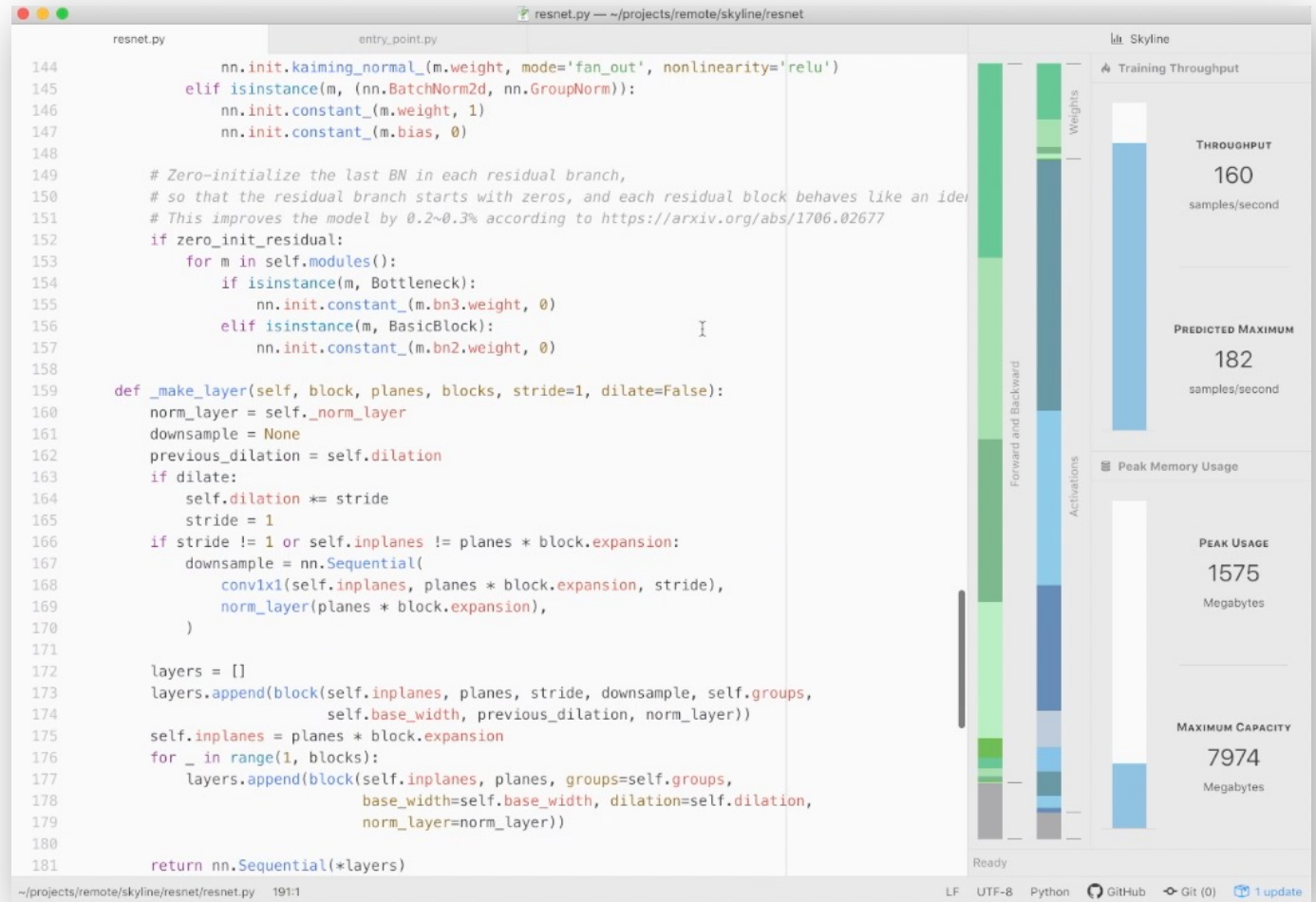
# Skyline: Interactive In-editor Performance Debugging

- Key performance metrics (throughput, memory usage)
- Iteration run time and memory footprint **breakdowns**
- **Interactive** visualizations linked to batch size predictions
- Live and proactive performance debugging during development



# Skyline: Interactive In-editor Performance Debugging

- Key performance metrics (throughput, memory usage)
- Iteration run time and memory footprint **breakdowns**
- **Interactive** visualizations linked to batch size predictions
- **Live and proactive** performance debugging during development

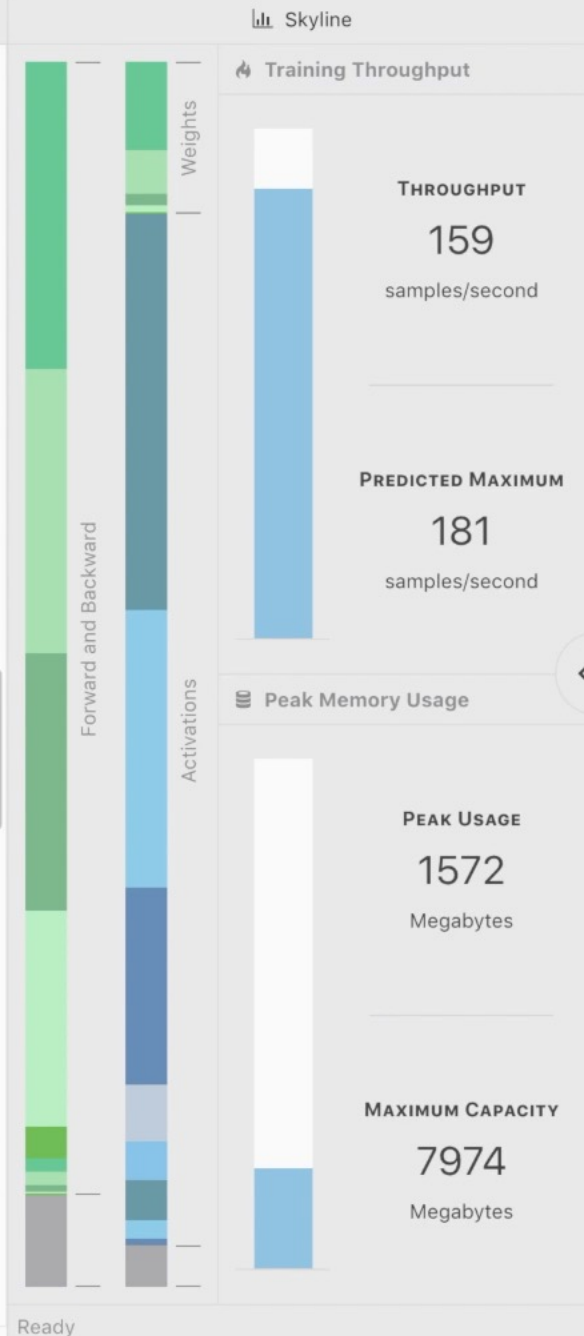


```

105 class ResNet(nn.Module):
106     def __init__(self, block, layers, num_classes=1000, zero_init_residual=False,
107                 groups=1, width_per_group=64, replace_stride_with_dilation=None,
108                 norm_layer=None):
109         super(ResNet, self).__init__()
110         if norm_layer is None:
111             norm_layer = nn.BatchNorm2d
112         self._norm_layer = norm_layer
113
114         self.inplanes = 64
115         self.dilation = 1
116         if replace_stride_with_dilation is None:
117             # each element in the tuple indicates if we should replace
118             # the 2x2 stride with a dilated convolution instead
119             replace_stride_with_dilation = [False, False, False]
120         if len(replace_stride_with_dilation) != 3:
121             raise ValueError("replace_stride_with_dilation should be None "
122                             "or a 3-element tuple, got {}".format(replace_stride_with_dilation))
123         self.groups = groups
124         self.base_width = width_per_group
125         self.conv1 = nn.Conv2d(3, self.inplanes, kernel_size=7, stride=2, padding=3,
126                                bias=False)
127         self.bn1 = norm_layer(self.inplanes)
128         self.relu = nn.ReLU(inplace=True)
129         self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
130         self.layer1 = self._make_layer(block, 64, layers[0])
131         self.layer2 = self._make_layer(block, 128, layers[1], stride=2,
132                                       dilate=replace_stride_with_dilation[0])
133

```

Interactive visualizations tied to the code!





# My Students: EcoSystem Research Group



- Hongyu Zhu (PhD)
  - Bojian Zheng (PhD)
  - Alexandra Tsvetkova (PhD)
  - James Gleeson (PhD, co-advised)
  - Anand Jayarajan (PhD)
  - Mustafa Quraish (PhD)
  - Shang (Sam) Wang (MSc)
  - Jiacheng Yang (MASc)
  - Pavel Golikov (MSc)
  - Yaoyao Ding (MASc)
  - Daniel Snider (MSc)
  - Kevin Song (MASc)
- 
- Yu Bo Gao (BSc)
  - Kimberly Hau (BASc)
  - Qingyuan Qie (BSc)
  - Chenhao Jiang (BSc)
  - Murali Andoorvedu (BASc)