CSC 2224: Parallel Computer Architecture and Programming DNN Training and Inference: Challenges, Trends, State-of-the-Art

Prof. Gennady Pekhimenko

University of Toronto

Fall 2022

Review #7

Horizontally Fused Training Array
Shang Wang et al., *MLSys 2021*

OR

In-Datacenter Performance Analysis of a Tensor Processing Unit, ISCA'17, Jouppi et al., https://dl.acm.org/doi/10.1145/3079856.3080246

Due Nov. 3rd

DNN Training and Inference : Challenges, Trends, State-of-the-Art

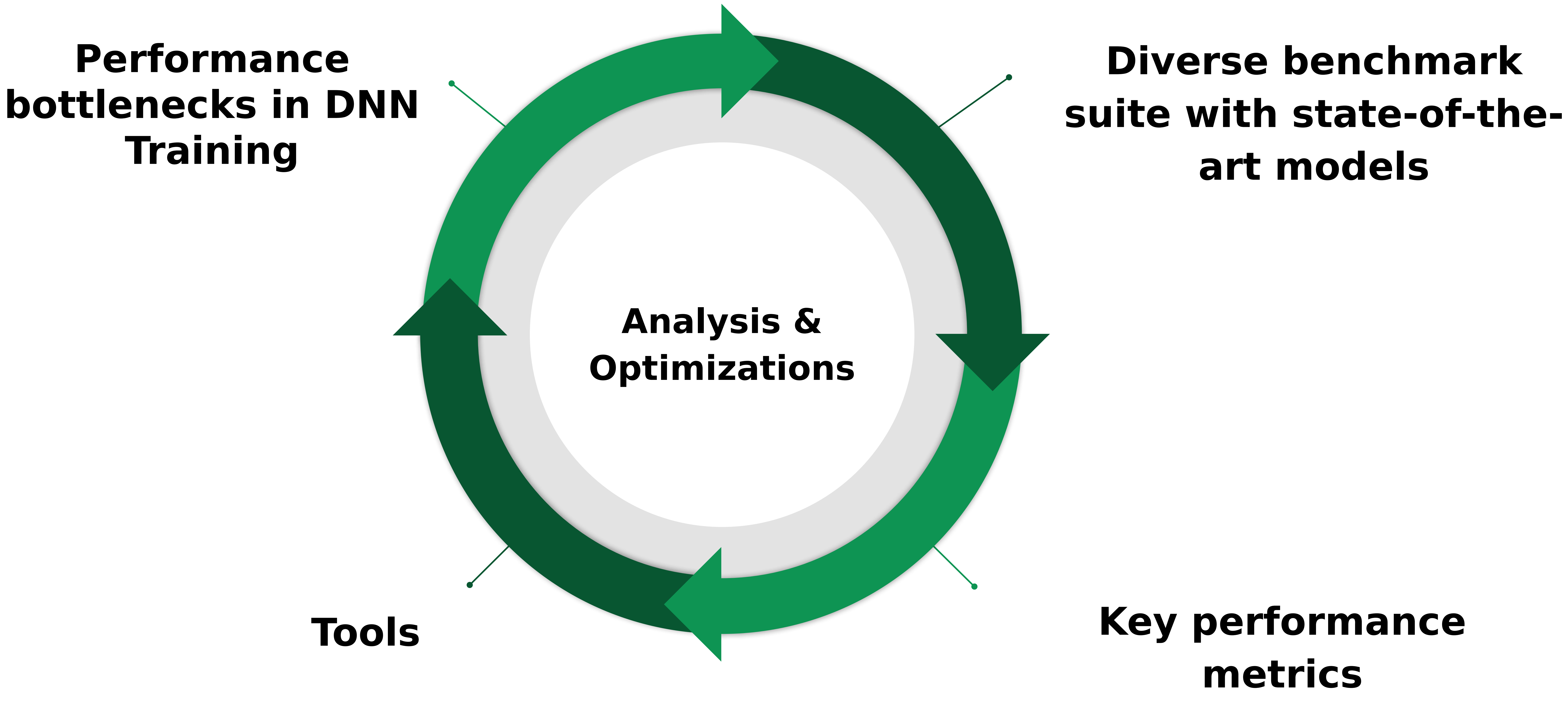
Gennady Pekhimenko, Assistant Professor

EcoSystem Group

Systems/Architecture Is a Servant for ML

ML Researcher

mxnet


Performance bottlenecks in DNN Training
Diverse benchmark suite with state-of-the-art models
Analysis & Optimizations
Key performance metrics
Tools

DNN Training and Inference : Challenges

1. Benchmarking

Machine Learning Benchmarking and Analysis

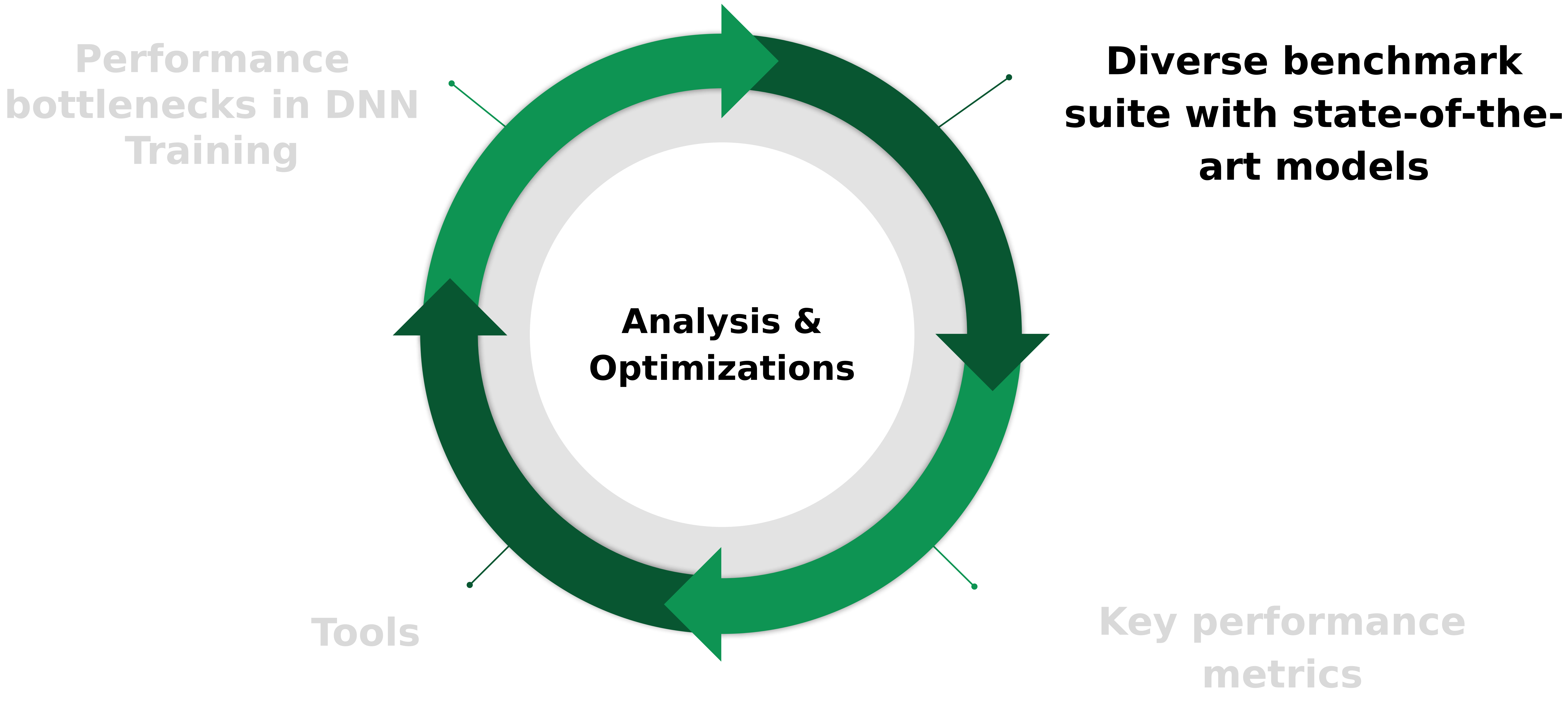
IISWC 2018

TO MEASURE IS TO KNOW

Lord Kelvin

MLSys 2020

ISCA 2020

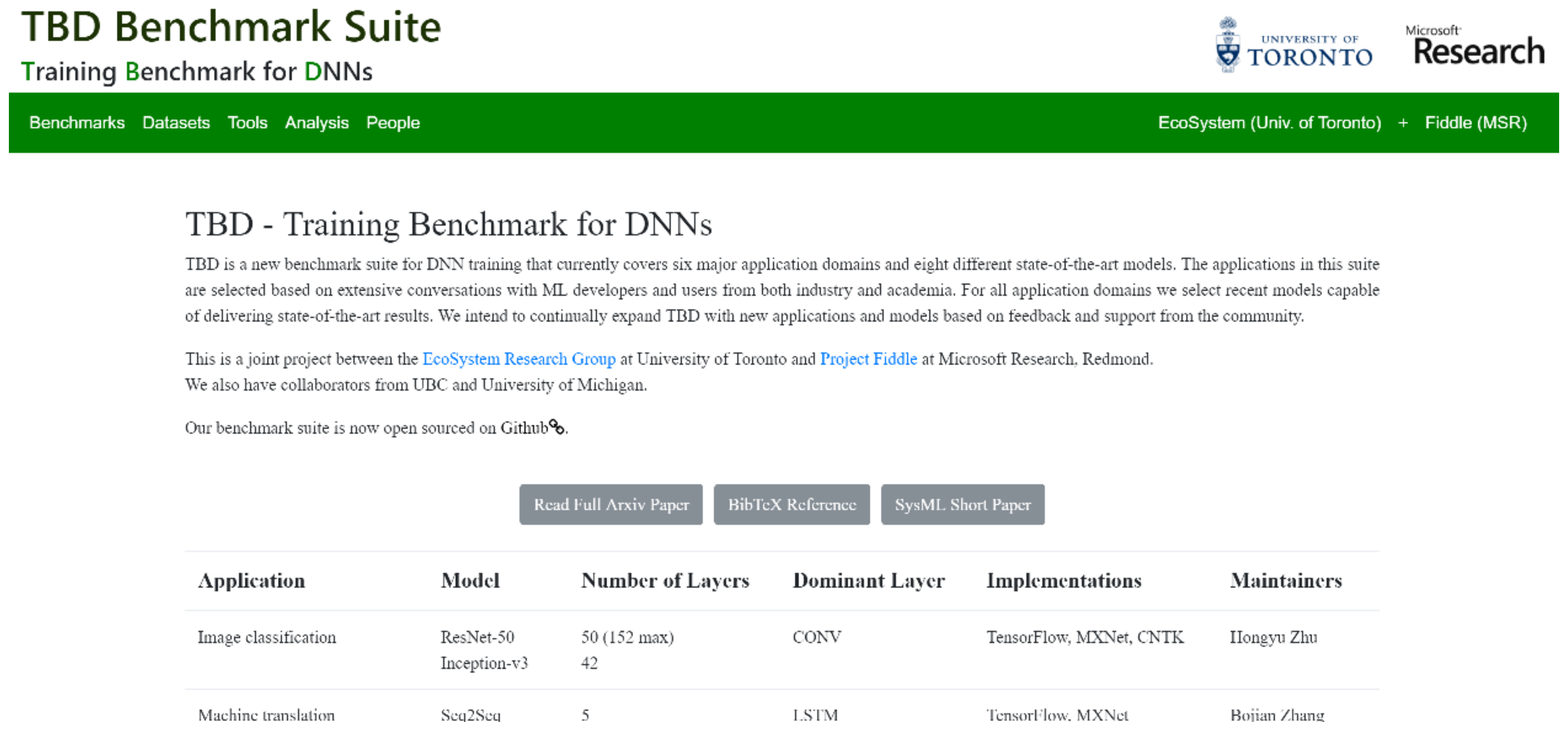

Performance bottlenecks in DNN Training
Diverse benchmark suite with state-of-the-art models
Analysis & Optimizations
Tools
Key performance metrics

Training Benchmarks for DNNs (TBD)

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 Akrout*

(Footnotes indicate available implementation: *T* for TensorFlow, *M* for mxnet, *C* for Microsoft CNTK, *P* for PYTORCH)

Our Focus: Benchmarking and Analysis


TBD Benchmark Suite
Training Benchmark for DNNs

UNIVERSITY OF TORONTO | Microsoft Research

Benchmarks Datasets Tools Analysis People | EcoSystem (Univ. of Toronto) + Fiddle (MSR)

TBD - Training Benchmark for DNNs

TBD is a new benchmark suite for DNN training that currently covers six major application domains and eight different state-of-the-art models. The applications in this suite are selected based on extensive conversations with ML developers and users from both industry and academia. For all application domains we select recent models capable of delivering state-of-the-art results. We intend to continually expand TBD with new applications and models based on feedback and support from the community.

This is a joint project between the EcoSystem Research Group at University of Toronto and Project Fiddle at Microsoft Research, Redmond.
We also have collaborators from UBC and University of Michigan.

Our benchmark suite is now open sourced on Github.

Read Full Arxiv Paper | BibTeX Reference | SysML Short Paper

Application	Model	Number of Layers	Dominant Layer	Implementations	Maintainers
Image classification	ResNet-50 Inception-v3	50 (152 max) 42	CONV	TensorFlow, MXNet, CNTK	Hongyu Zhu
Machine translation	Seq2Seq	5	LSTM	TensorFlow, MXNet	Bojian Zhang

Building tools to analyze ML performance/efficiency

Industry/Academia de-facto standard

http://tbd-suite.ai

https://mlperf.org/

MLPerf Training Results v0.6 (July 10th, 2019)

Closed Division Times

#	Submitter	System	Processor	#	Accelerator	#	Software	Benchmark results (minutes)							Details	Code
								Image classification	Object detection, light-weight	Object detection, heavy-wt.	Translation, recurrent	Translation, non-recur.	Recom-mendation	Reinforce-ment Learning		
								ImageNet	COCO	COCO	WMT E-G	WMT E-G	MovieLens-20M	Go		
								ResNet-50 v1.5	SSD w/ ResNet-34	Mask-R-CNN	NMT	Transformer	NCF	Mini Go		
Available in cloud																
0.6-1	Google	TPUv3.32			TPUv3	16	TensorFlow, TPU 1.14.1.dev	42.19	12.61	107.03	12.25	10.20	[1]		details	code
0.6-2	Google	TPUv3.128			TPUv3	64	TensorFlow, TPU 1.14.1.dev	11.22	3.89	57.46	4.62	3.85	[1]		details	code
0.6-3	Google	TPUv3.256			TPUv3	128	TensorFlow, TPU 1.14.1.dev	6.86	2.76	35.60	3.53	2.81	[1]		details	code
0.6-4	Google	TPUv3.512			TPUv3	256	TensorFlow, TPU 1.14.1.dev	3.85	1.79		2.51	1.58	[1]		details	code
0.6-5	Google	TPUv3.1024			TPUv3	512	TensorFlow, TPU 1.14.1.dev	2.27	1.34		2.11	1.05	[1]		details	code
0.6-6	Google	TPUv3.2048			TPUv3	1024	TensorFlow, TPU 1.14.1.dev	1.28	1.21			0.85	[1]		details	code
Available on-premise																
0.6-7	Intel	32x 2S CLX 8260L	CLX 8260L	64			TensorFlow						[1]	14.43	details	code
0.6-8	NVIDIA	DGX-1			Tesla V100	8	MXNet, NGC19.05	115.22					[1]		details	code
0.6-9	NVIDIA	DGX-1			Tesla V100	8	PyTorch, NGC19.05		22.36	207.48	20.55	20.34	[1]		details	code
0.6-10	NVIDIA	DGX-1			Tesla V100	8	TensorFlow, NGC19.05						[1]	27.39	details	code
0.6-11	NVIDIA	3x DGX-1			Tesla V100	24	TensorFlow, NGC19.05						[1]	13.57	details	code
0.6-12	NVIDIA	24x DGX-1			Tesla V100	192	PyTorch, NGC19.05			22.03			[1]		details	code
0.6-13	NVIDIA	30x DGX-1			Tesla V100	240	PyTorch, NGC19.05		2.67				[1]		details	code
0.6-14	NVIDIA	48x DGX-1			Tesla V100	384	PyTorch, NGC19.05				1.99		[1]		details	code
0.6-15	NVIDIA	60x DGX-1			Tesla V100	480	PyTorch, NGC19.05					2.05	[1]		details	code
0.6-16	NVIDIA	130x DGX-1			Tesla V100	1040	MXNet, NGC19.05	1.69					[1]		details	code
0.6-17	NVIDIA	DGX-2			Tesla V100	16	MXNet, NGC19.05	57.87					[1]		details	code
0.6-18	NVIDIA	DGX-2			Tesla V100	16	PyTorch, NGC19.05		12.21	101.00	10.94	11.04	[1]		details	code

MLPerf Inference Results v0.5 (Nov. 6, 2019)

Inf-0.5-14	dividiti	Firefly-RK3399 (firefly)	80.12				391.02			
Inf-0.5-15	Google	Cloud TPU v3							16,014.29	32,71
Inf-0.5-16	Google	2x Cloud TPU v3								65,43
Inf-0.5-17	Google	4x Cloud TPU v3								130,83
Inf-0.5-18	Google	8x Cloud TPU v3								261,58
Inf-0.5-19	Google	16x Cloud TPU v3								524,97
Inf-0.5-20	Google	32x Cloud TPU v3								1,038,51
Inf-0.5-21	Habana Labs	HL-102-Goya PCI-board					0.24	700.00		14,45
Inf-0.5-22	Intel	Intel® Xeon® Platinum 9200 processors								
Inf-0.5-23	Intel	Intel® Xeon® Platinum 9200 processors	0.49		27,244.81	29,203.30	1.37		4,850.62	5,96
Inf-0.5-24	Intel	DELL ICL i3 1005G1	3.55			507.71	13.58			10
Inf-0.5-25	NVIDIA	Supermicro 4029GP-TRT-OTO-28 8xT4 (T4x8)		6,320.00	135,073.00	141,807.00		1,920.00	41,546.64	44,97
Inf-0.5-26	NVIDIA	Supermicro 6049GP-TRT-OTO-29 20xT4 (T4x20)							103,532.10	113,59
Inf-0.5-27	NVIDIA	SCAN 3XS DBP T496X2 Fluid (TitanRTXx4)		8,704.00	199,098.30	222,388.00		2,560.00	60,030.57	66,25
Inf-0.5-28	NVIDIA	NVIDIA Jetson AGX Xavier (Xavier)	0.58	302.00		6,520.75	2.04	100.00		2,15
Inf-0.5-29	Qualcomm	SDM855 QRD	3.02				8.95			
CATEGORY: Preview										
Inf-0.5-31	Alibaba T-Head	Alibaba HanGuang					0.17	2,692.00	45,169.48	69,30
Inf-0.5-32	Centaur Technology	Centaur Technology Reference Design v1.0	0.33			6,042.34	1.05			1,21

MLPerf becomes de-facto standard

MLPerf Training Benchmark

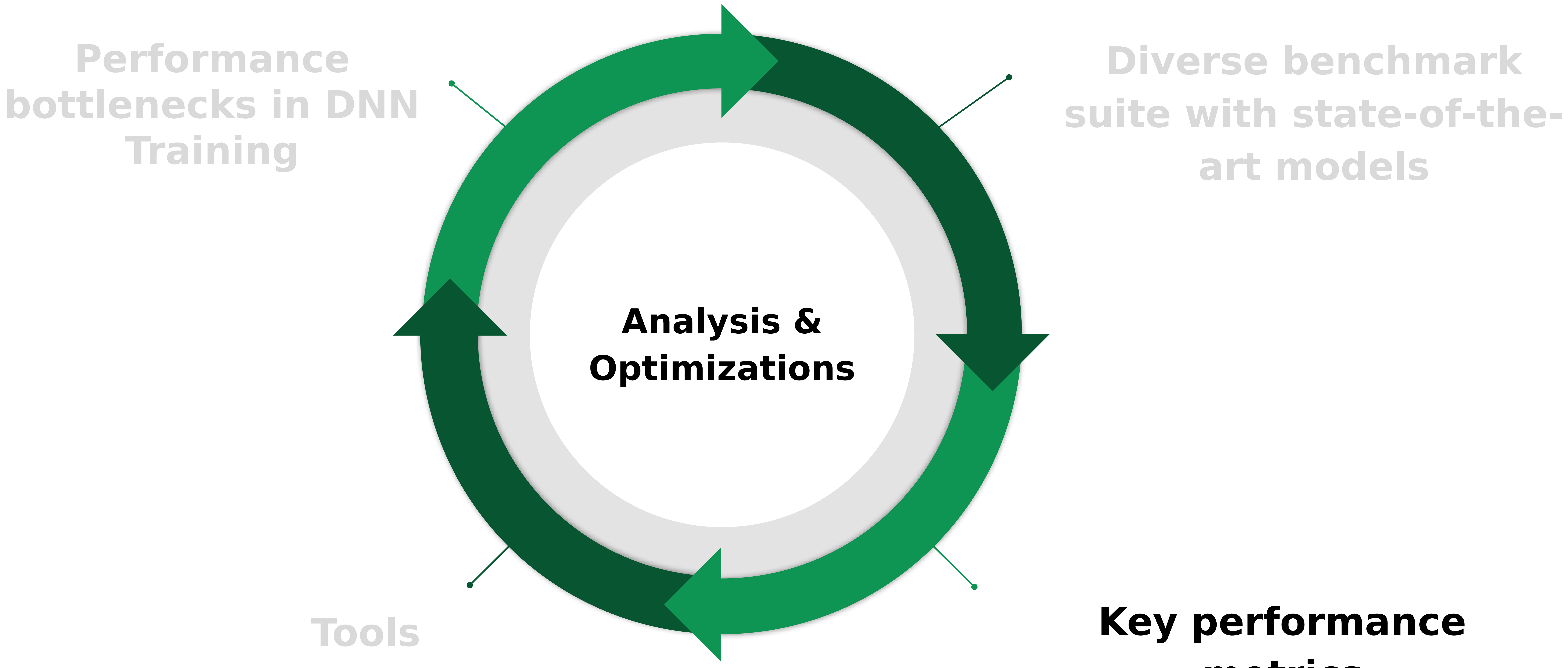
Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, David Brooks, Dehao Chen, Debojyoti Dutta, Udit Gupta, Kim Hazelwood, Andrew Hock, Xinyuan Huang, Atsushi Ike, Bill Jia, Daniel Kang, David Kanter, Naveen Kumar, Jeffery Liao, Guokai Ma, Deepak Narayanan, Tayo Oguntebi, *Gennady Pekhimenko*, Lillian Pentecost, Vijay Janapa Reddi, Taylor Robie, Tom St. John, Tsuguchika Tabaru, Carole-Jean Wu, Lingjie Xu, Masafumi Yamazaki, Cliff Young, and Matei Zaharia

MLSys 2020

MLPerf Inference accepted to ISCA 2020

DNN Training and Inference : Challenges

2. Tools and Metrics



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

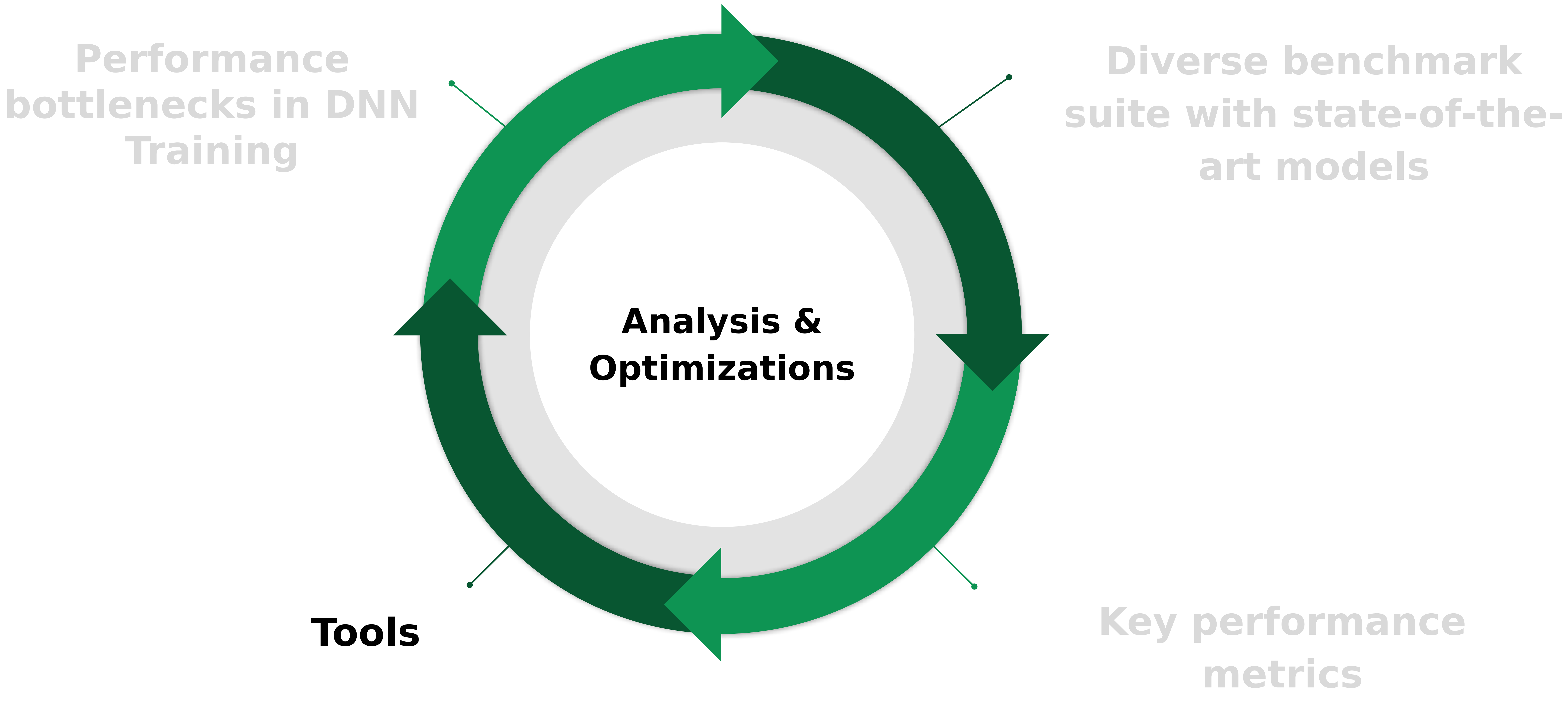
  GPU busy time over *Elapsed time*

- FP32/FP16/Tensor Core Utilization

  Average instructions executed per cycle over *Maximum instructions per cycle*

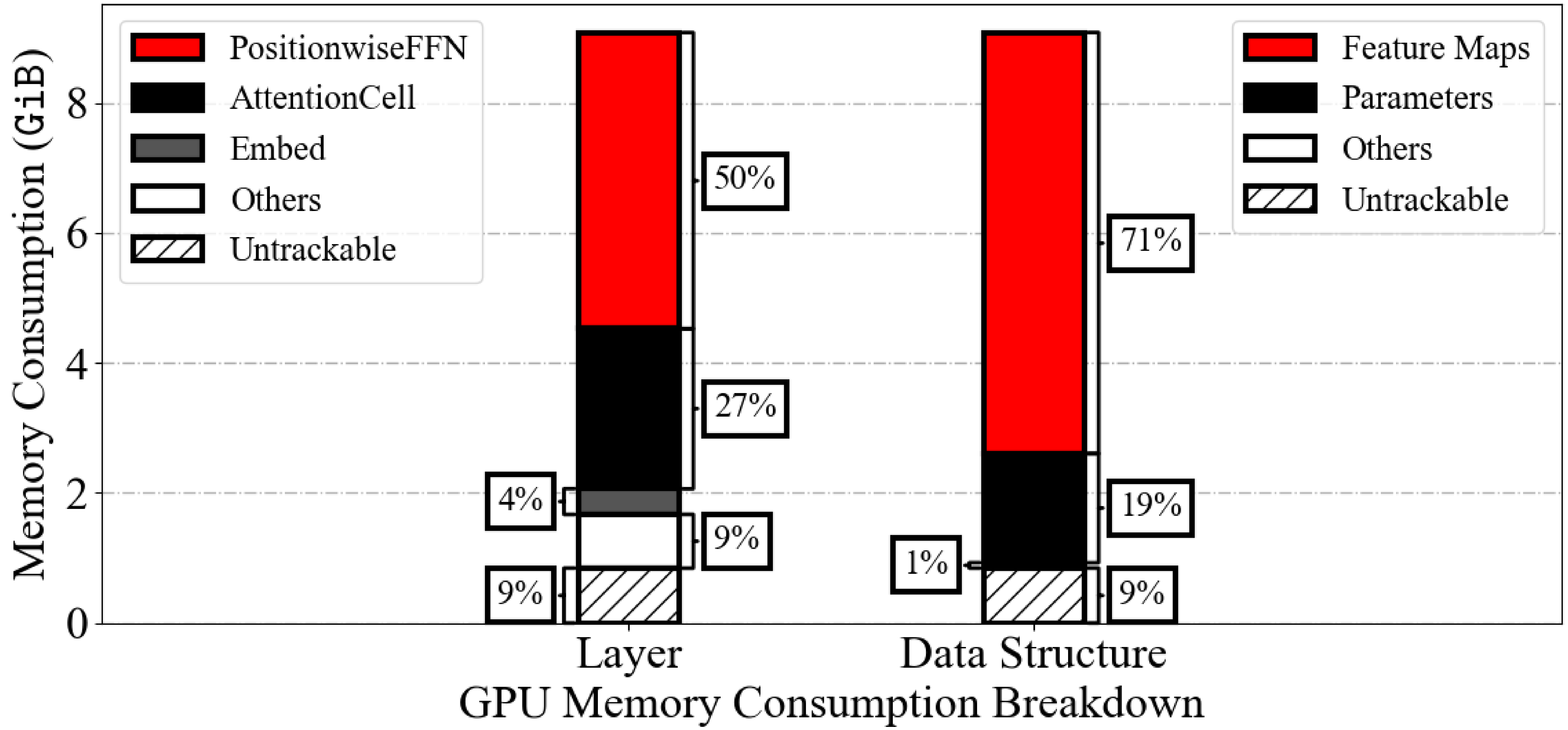
- Memory Breakdown

  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

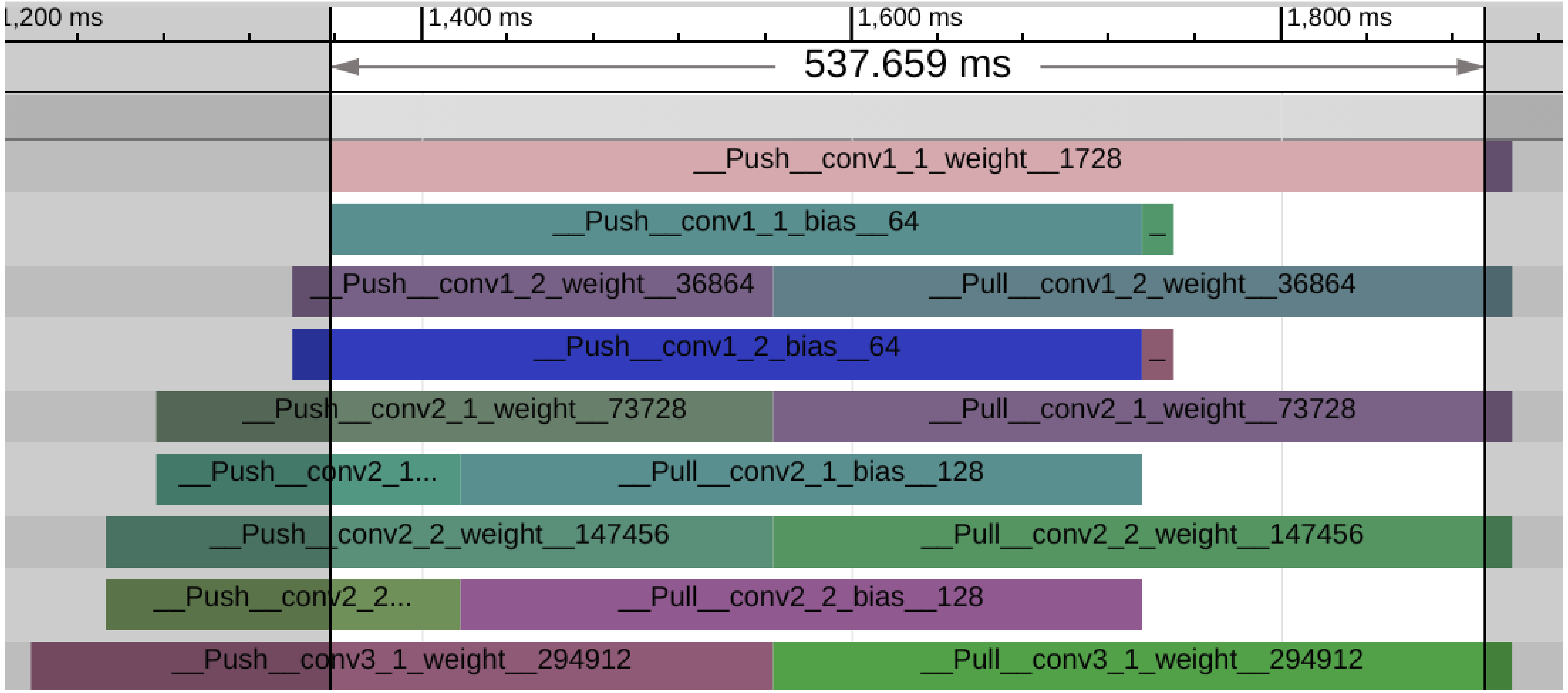
BERT: Memory Profile



Feature maps are still dominant in many new models

Network Profiling

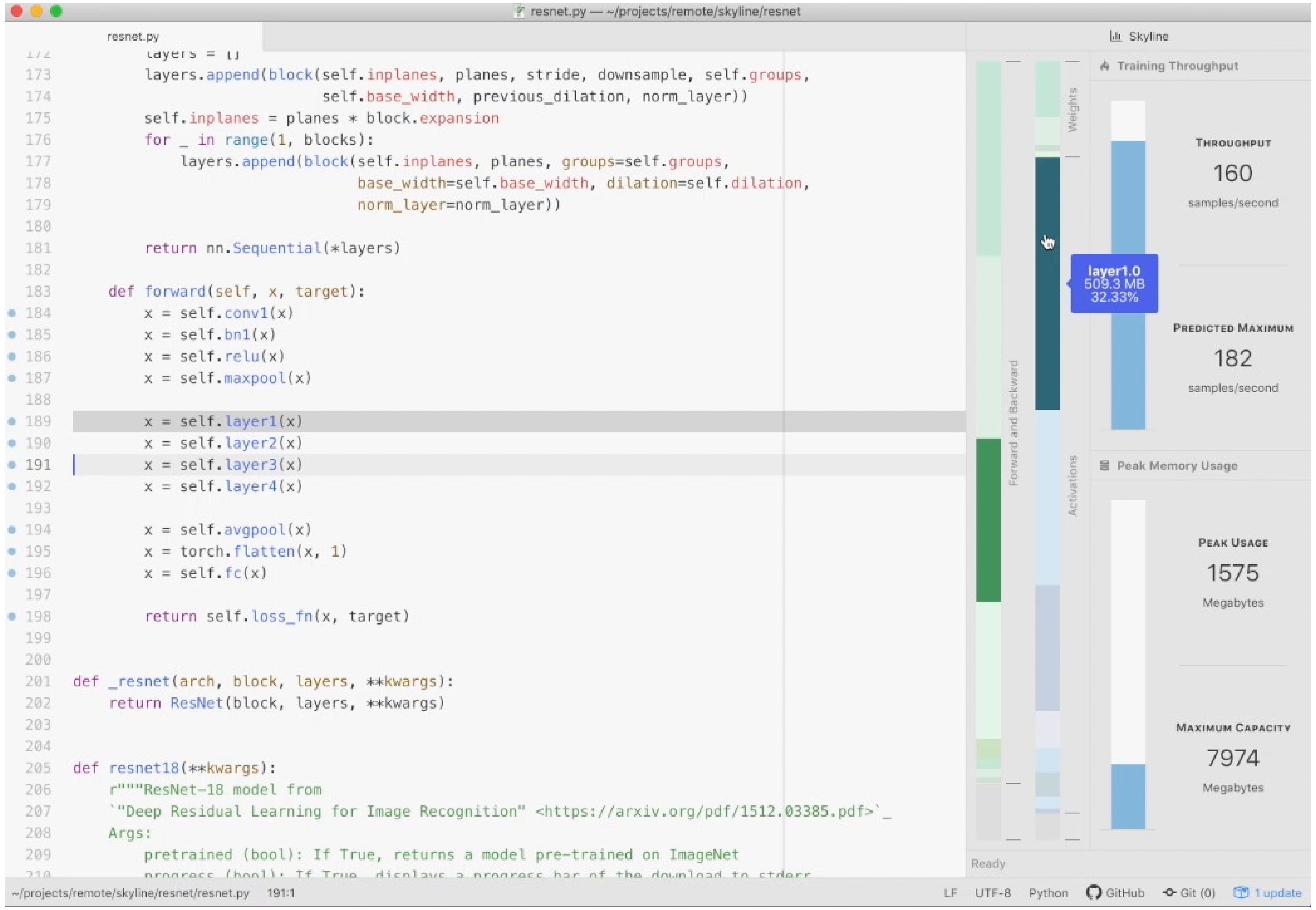
Our network profiler shows the communication traces



Skyline Demo at MLSys 2020

Skyline

Interactive In-editor Performance Visualizations and Debugging for DNN Training



Geoffrey X. Yu, Tovi Grossman,
Gennady Pekhimenko

UNIVERSITY OF TORONTO

VECTOR INSTITUTE

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

so. my pytorch code is slow. what do people us for profiling? cProfile just tells me run_backward is expensive, which is not so useful...

12 3:47 PM - May 7, 2017

See Hal Daumé III's other Tweets

Sam Bowman @sleepinyourhat · May 26, 2017
Any tips on identifying speed bottlenecks (profiling) with @PyTorch? Right now bumbling along with cProfile.

Zico Kolter
@zicokolter

rsperse torch.cuda.synchronize() liberally when debuggi
a code, to see where the bottlenecks acually are...

3:09 PM - May 27, 2017

See Zico Kolter's other Tweets

Sam Bowman @sleepinyourhat · May 26, 2017
Any tips on identifying speed bottlenecks (pr
@PyTorch? Right now bumbling along with c

Joachim Hagege
@JoachimHagege

Hi Sam. I'm struggling with same issue right
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

ely slow

day.

e of code and

her I know.

Profiling pytorch scripts?

hughperkins

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

I've tried the following:

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

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

- If it was a c++ script, that didnt use cuda, I might use eith
  debugger, stop it, and store the stacktrace. do this eg 5-1
  tend to me in man yof the stacktraces => this is the bottle
- if it was cltorch, or deepcl, well I pre-instrumented them w
- in pytorch cuda, I suppose I should use an nviida profiler?

Its not clear to me which bits of the program are taking the time,
at a higher level than nvidia profiler probably. Thoughts on idea
pytorch?

dvice for debugging slow backw... pass

mrdrozdov Andrew Drozdov Apr '17

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?

created Apr '17 | last reply Dec '17 | 4 replies | 1.3k views | 4 users | 1 like | 1 link

3 people are talking about this

ion running very slow?: I a
mount of training set, it is t
y code, I found the loss.ba
er, both score and target a

```
Model time on
Model time on
Model time on
Model time on
Model time on
Model time on
Model time on
```

dynamic attention
I use two for loops

2019

r Tweets

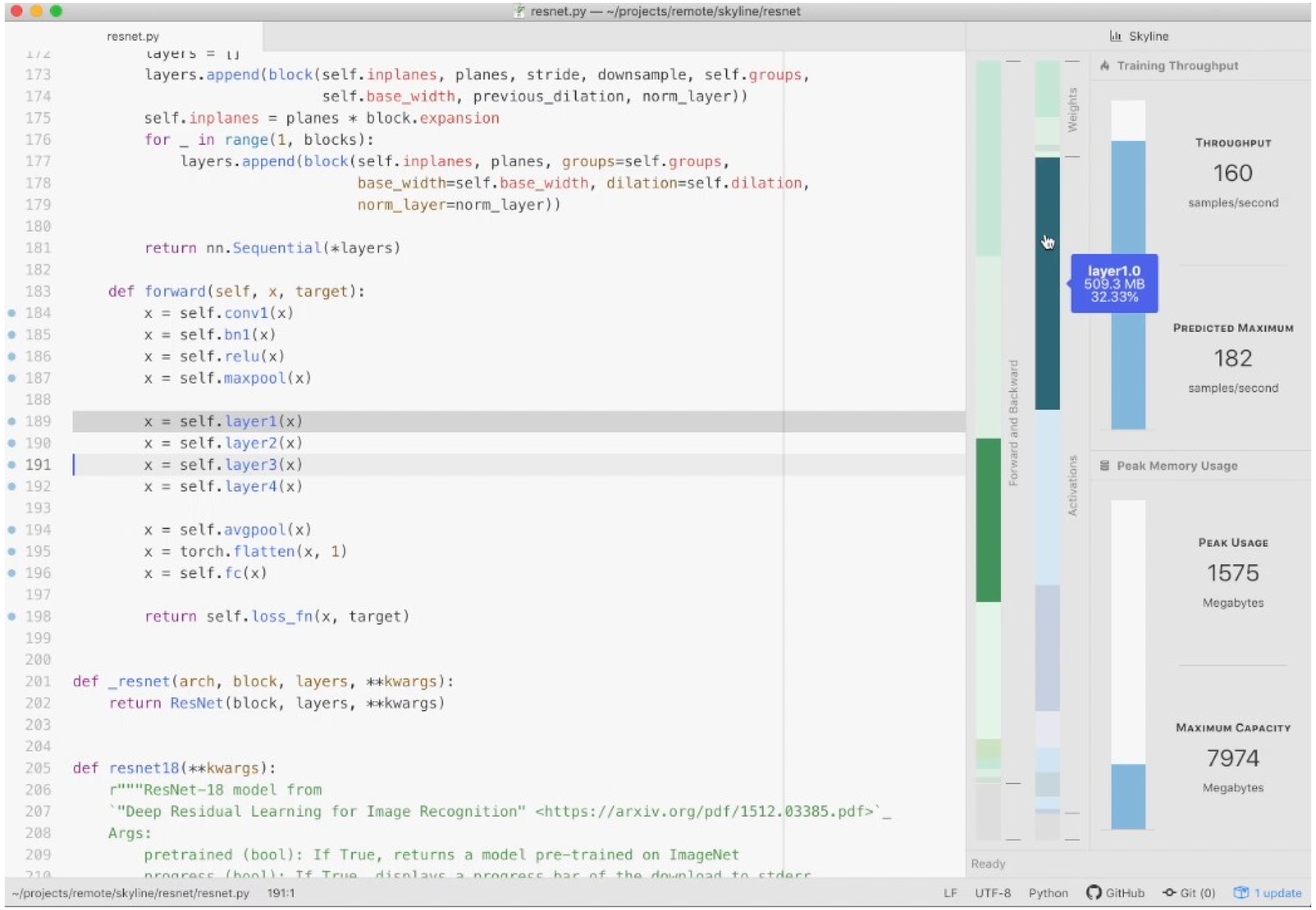
Skyline

Interactive In-editor Performance Visualizations and Debugging for DNN Training

UNIVERSITY OF TORONTO

VECTOR INSTITUTE

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



PyTorch ATOM


entry_point.py — ~/projects/remote/skyline/resnet
resnet.py
entry_point.py
Skyline
Interactive visualizations tied to the code!
import torch
import torch.nn as nn
import resnet
def skyline_model_provider():
    return resnet.resnet50().cuda()
def skyline_input_provider(batch_size=1):
    return (
        torch.randn((batch_size, 3, 224, 224)).cuda(),
        torch.randint(low=0, high=1000, size=(batch_size,)).cuda(),
)
def skyline_iteration_provider(model):
    optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)
    def iteration(*inputs):
        optimizer.step()
    return iteration
11 def skyline_input_provider(batch_size=1):
Training Throughput
Throughput
57
samples/second
Predicted Maximum
181
samples/second
Peak Memory Usage
Peak Usage
358
Megabytes
Maximum Capacity
7974
Megabytes
Weights
Forward and Backward
Activations
Untracked
Showing predicted performance
~/projects/remote/skyline/resnet/entry_point.py*
11:1
LF
UTF-8
Python
GitHub
Git (0)
1 update

Skyline

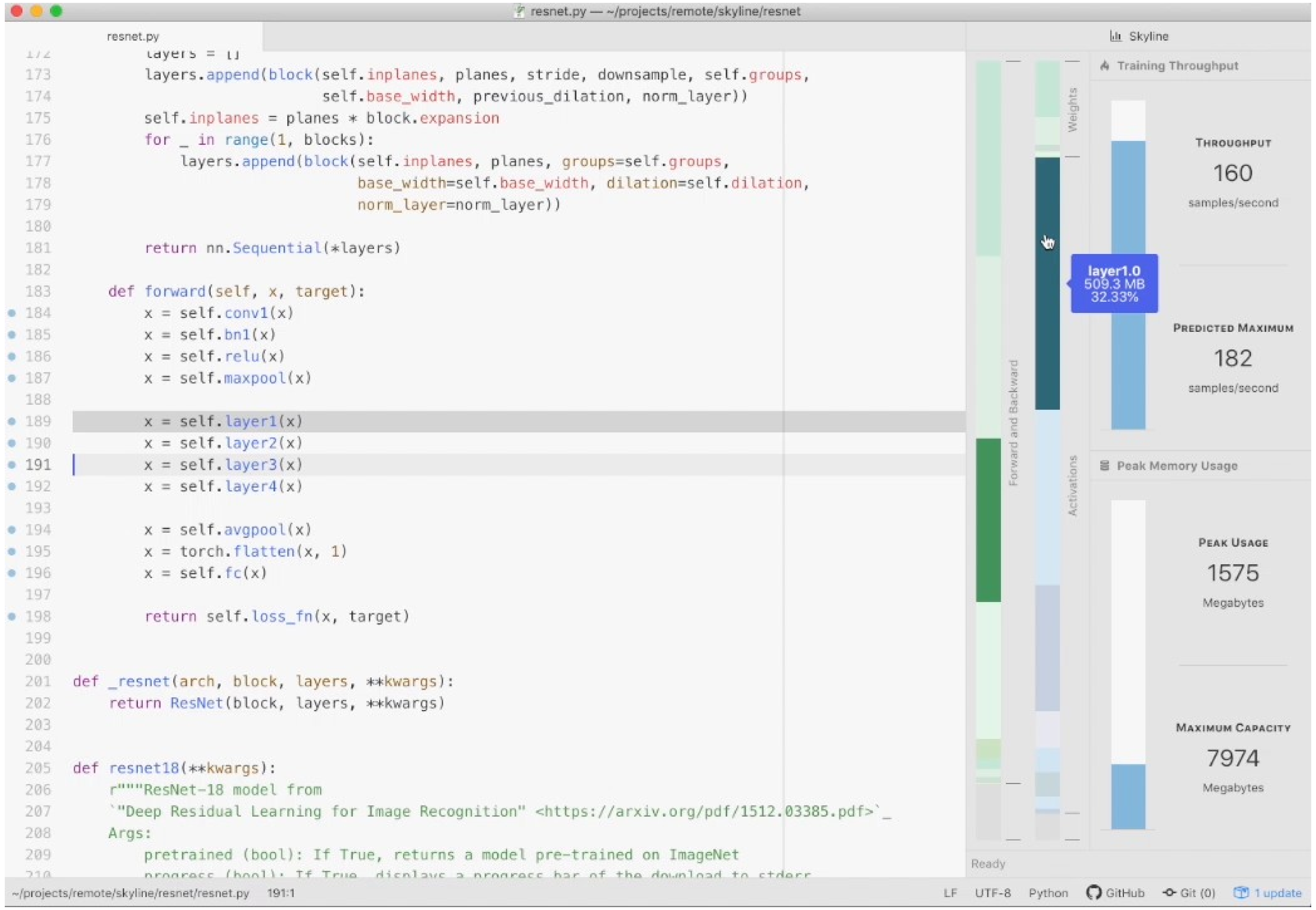
Interactive In-editor Performance Visualizations and Debugging for DNN Training

UNIVERSITY OF TORONTO

VECTOR INSTITUTE

Learn how to use Skyline to:

- ✓ *Identify* run time and memory bottlenecks
- ✓ *Tune* batch sizes during development
- ✓ *Proactively* design models with performance in mind



Skyline works with PyTorch models in Atom

```
pip install skyline-cli && \
    apm install skyline
```

PyTorch ATOM

DNN Training and Inference : Challenges

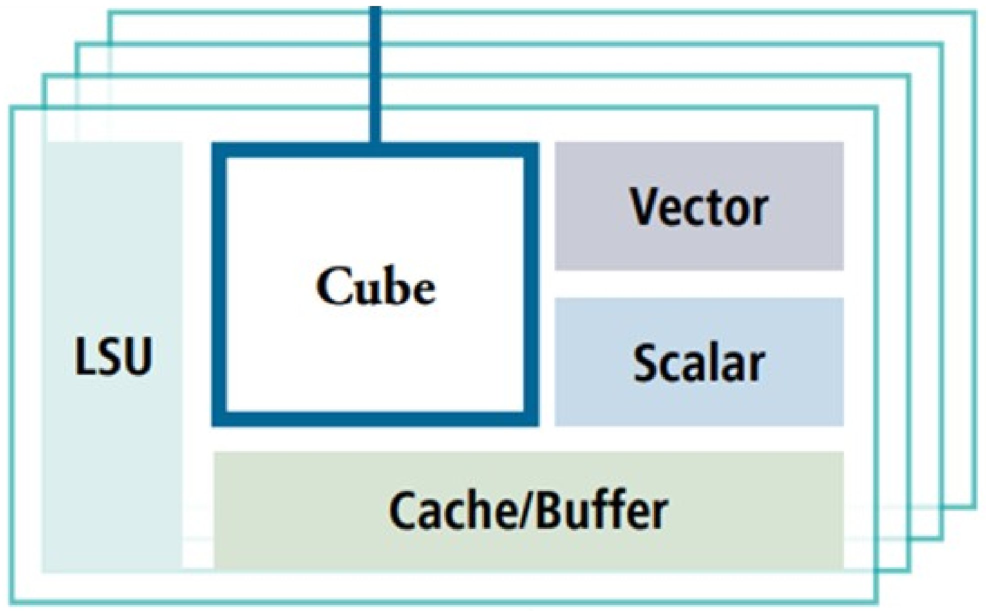
3. Methodology

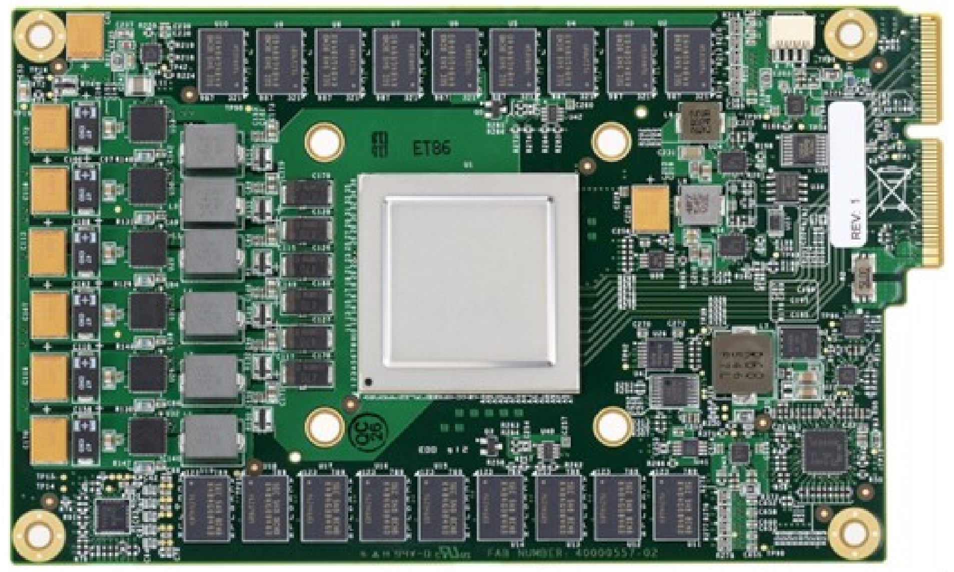
Challenges for Metrics & Profiling

Specialized hardware for DNN training is a hot research area

Nvidia
GPU

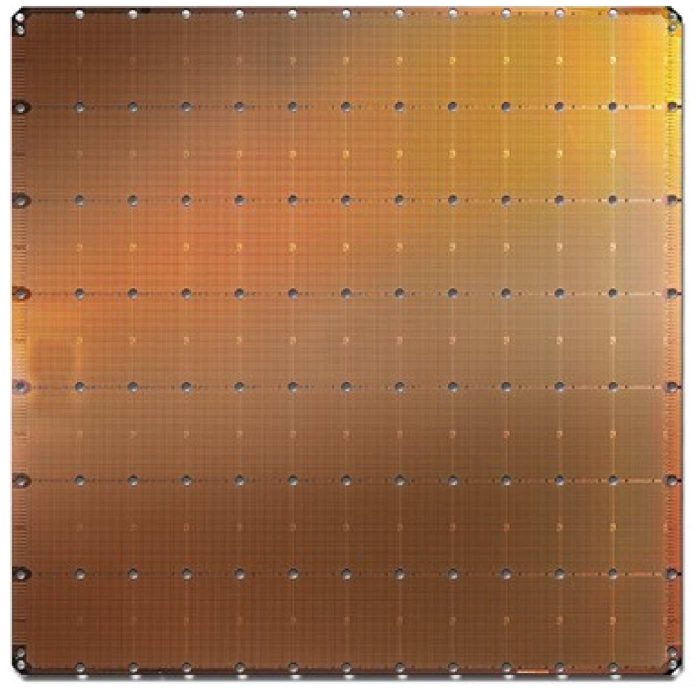
Huawei
Da Vinci
Core





Google
TPU

Cerebras
Wafer-Scale
Engine



Habana
Gaudi

Accelerators are specially optimized for DNN training

Challenges for Metrics & Profiling (2)

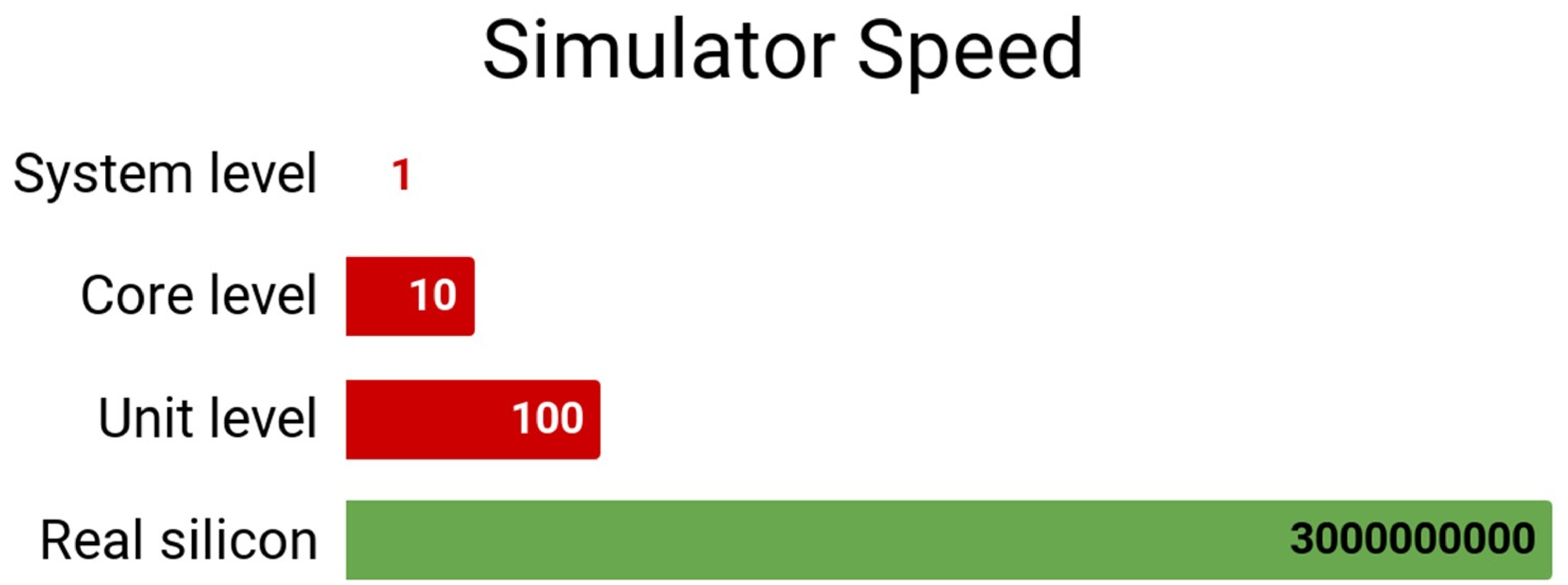
Measuring statistical efficiency require end-to-end training

MLPerf Benchmark	Training time on Nvidia P100 (Hours)
ResNet-50	147.2
Mask R-CNN	83.32
Transformer	31.16
MiniGo	73.14

Benchmarking could take many hours
even on powerful hardware

Challenges for Metrics & Profiling (3)

Option #1: On simulator

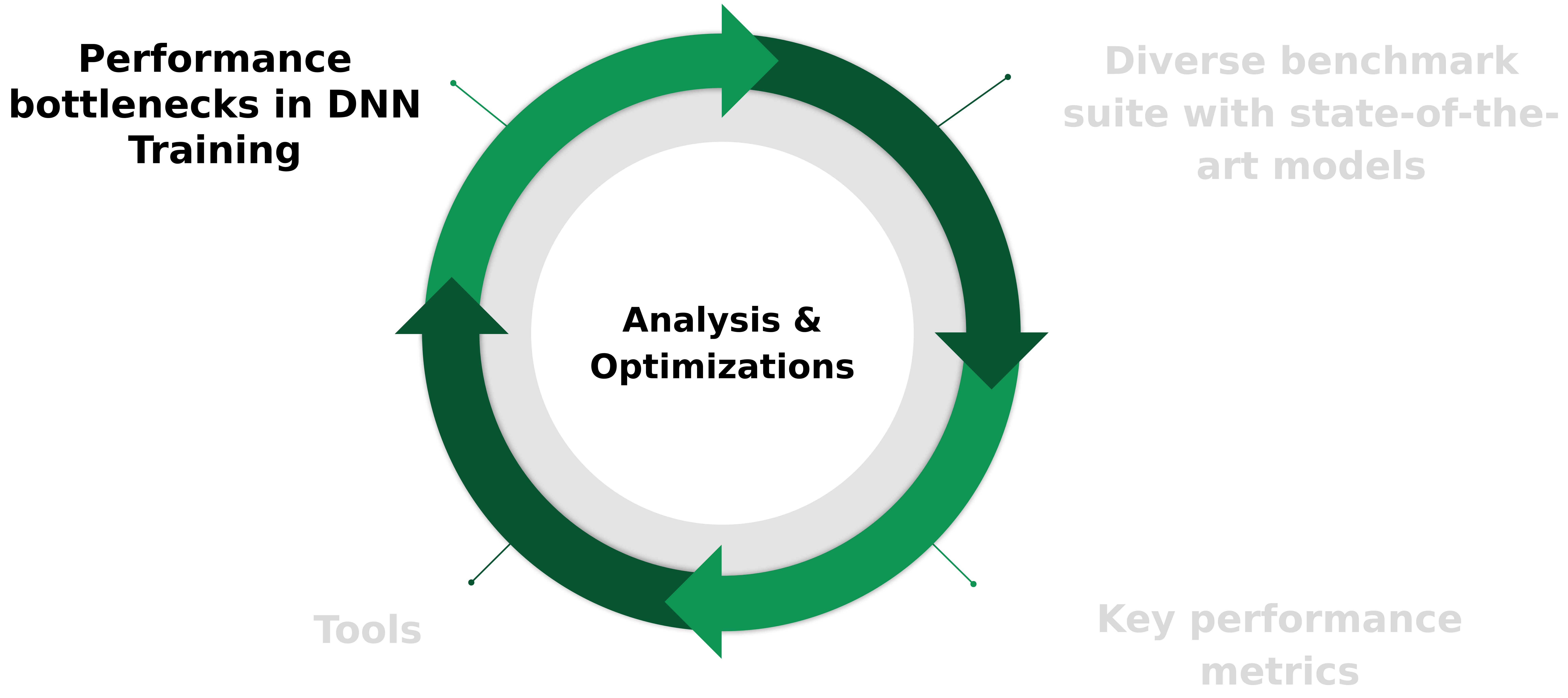


Source: David Kaplan, When hardware must just work

End-to-end training is prohibitively slow

Option #2: On FPGA/ASIC

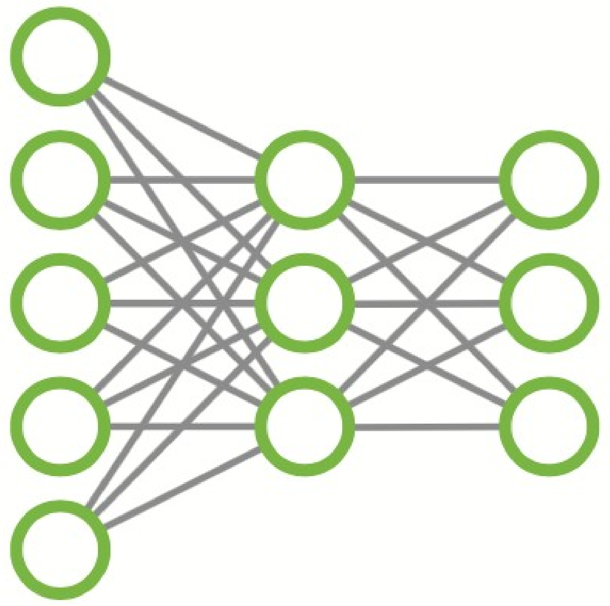
Expensive and require considerable effort

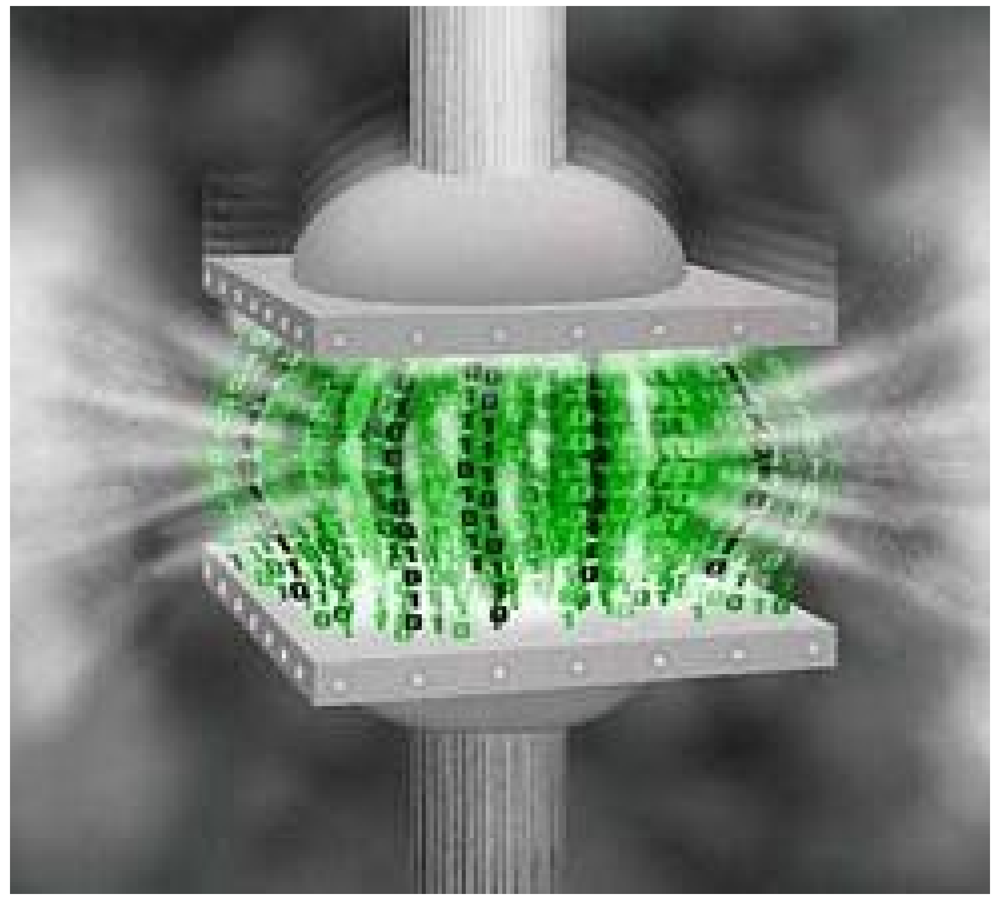

Performance bottlenecks in DNN Training
Diverse benchmark suite with state-of-the-art models
Analysis & Optimizations
Tools
Key performance metrics

DNN Training and Inference : Trends and State-of-the-Art

DNN Training and Inference : Trends and State-of-the-Art

1. Memory is still an Issue

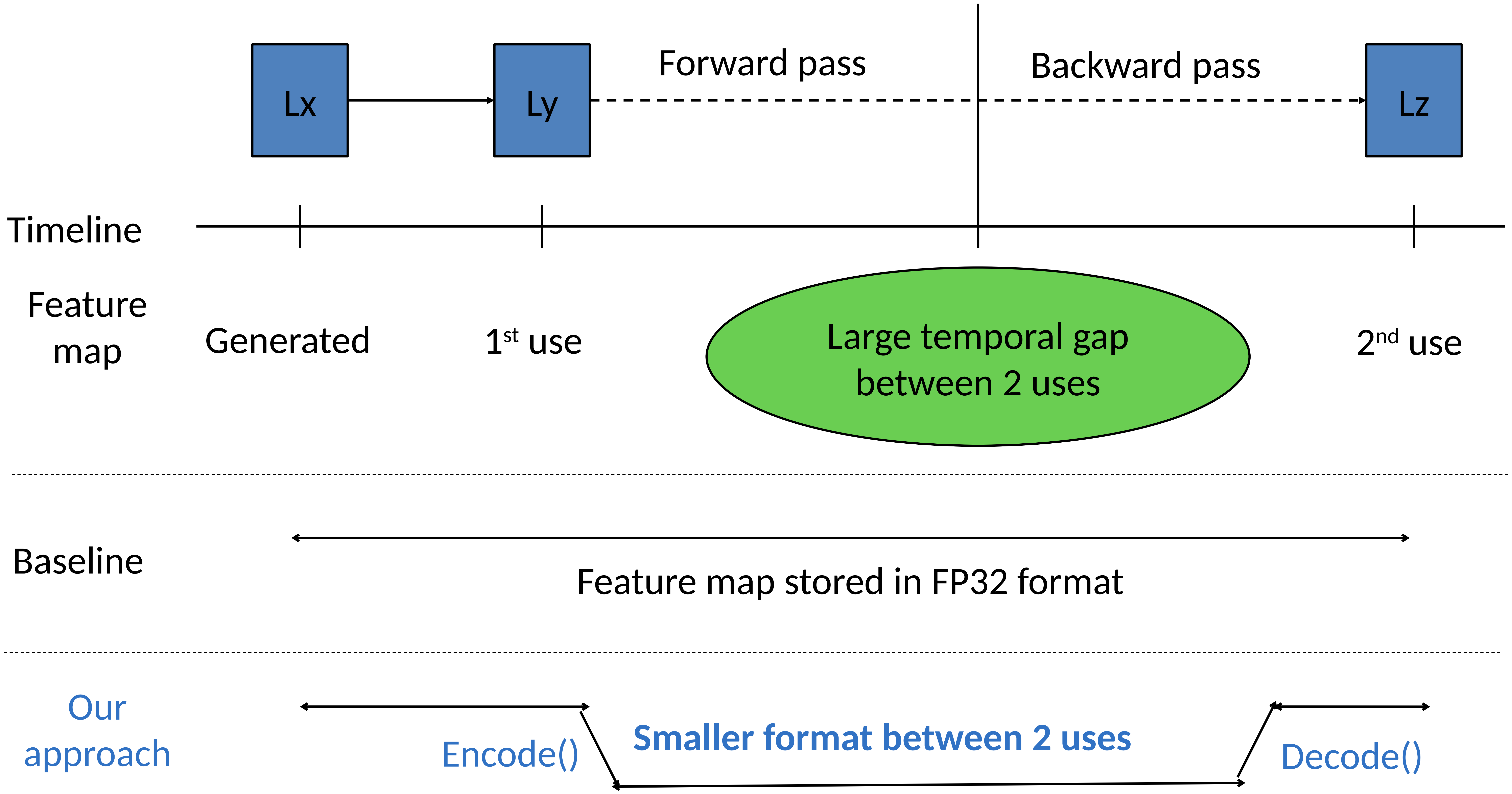




Gist: Efficient Data Encoding for Deep Neural Network Training

ISCA2018

Our Insight

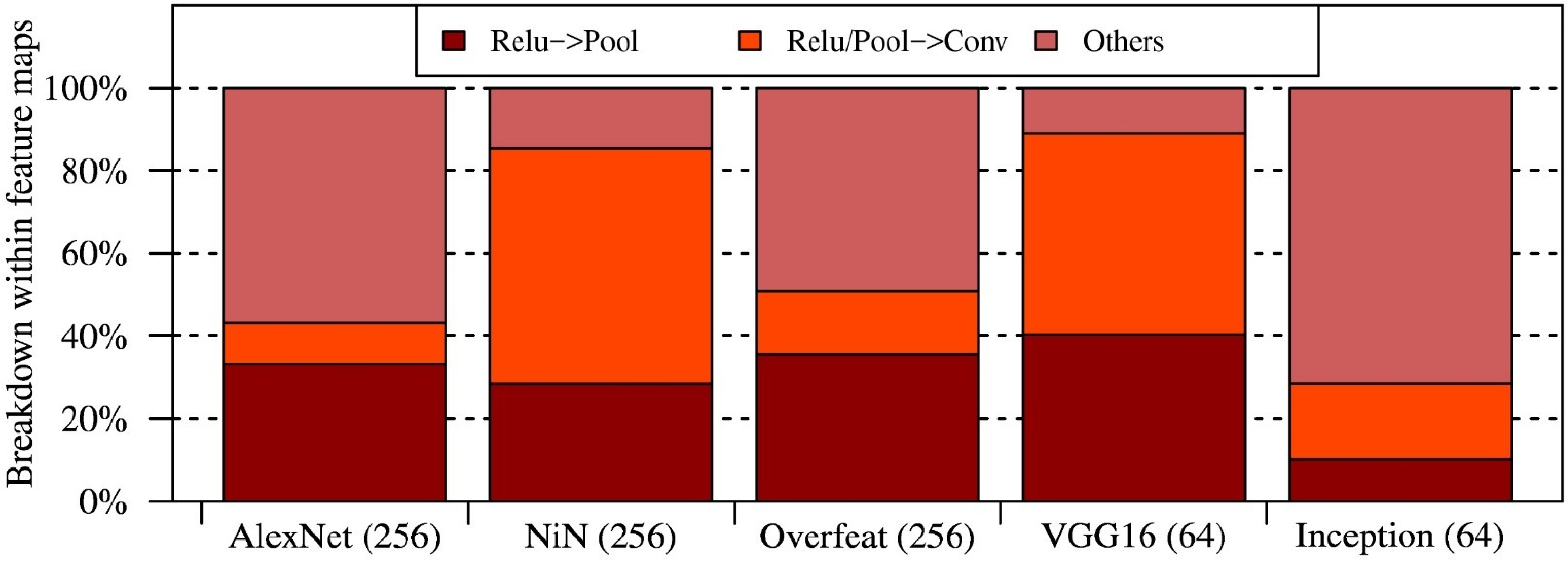


Lx
Ly
Forward pass
Backward pass
Lz
Timeline
Feature map
Generated
1st use
Large temporal gap between 2 uses
2nd use
Baseline
Feature map stored in FP32 format
Our approach
Encode()
Smaller format between 2 uses
Decode()

Layer-Specific Encodings

- Key Idea:
  - Use layer-specific compression
- Can be both fast and efficient
- Can be even lossless
  - Usually difficult for FP32

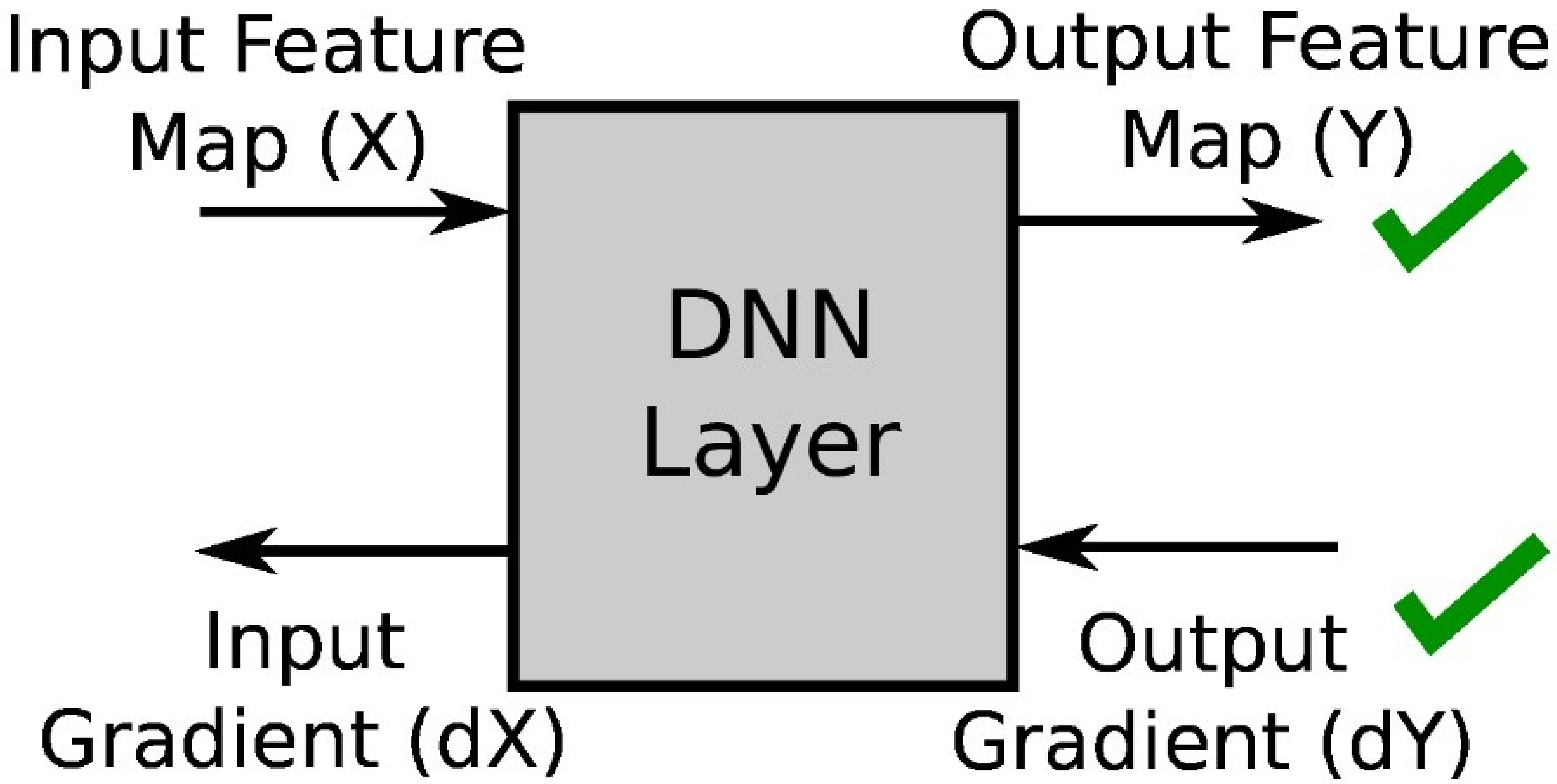
Relu Importance



Significant footprint is due to Relu layer
CNTK Profiling

Relu -> Pool

Relu Backward Propagation



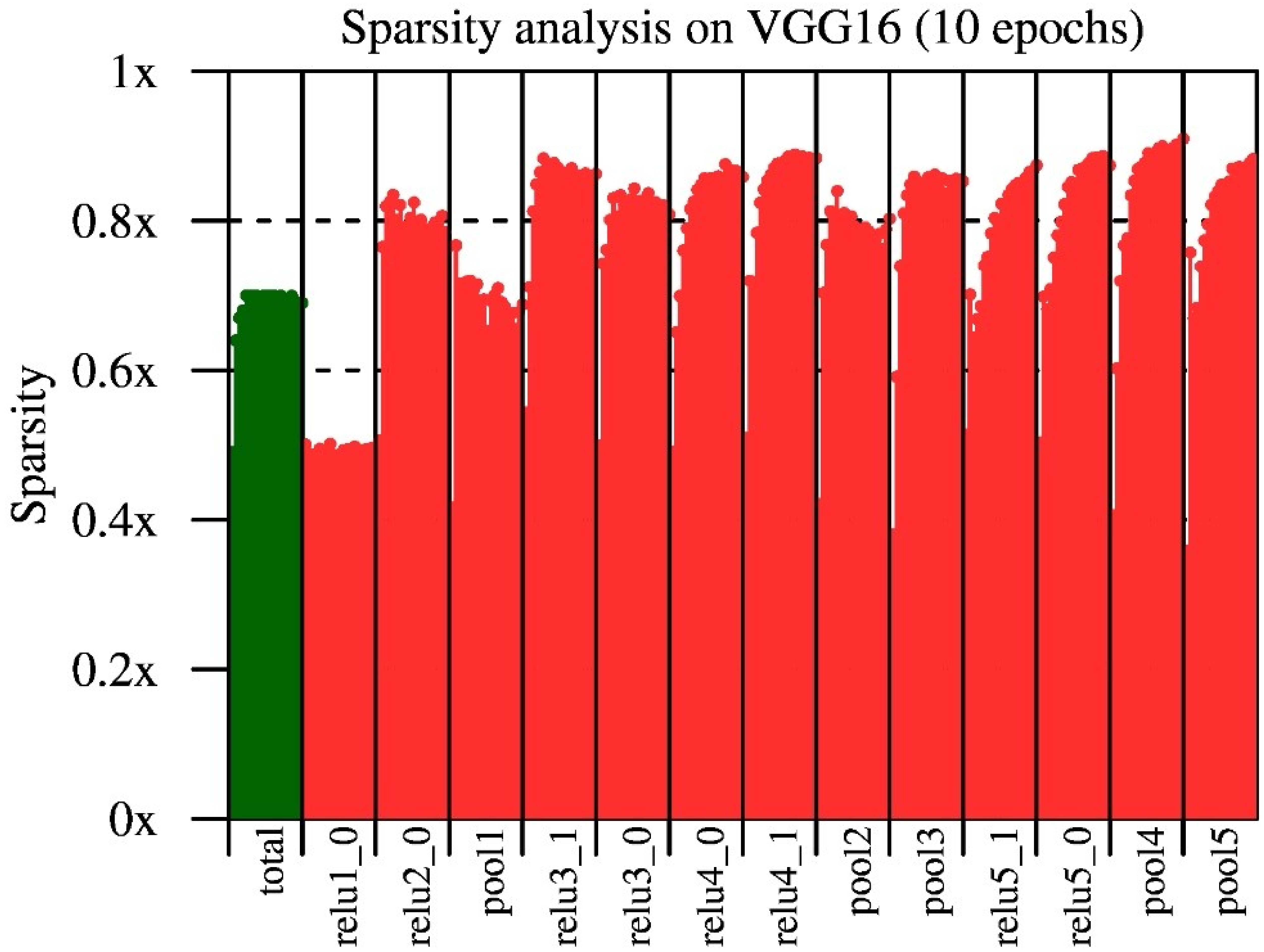
$$dX = f(Y, dY)$$

$$dx = y > 0 \ ?\ dy : 0;$$

Binarize **– 1 bit representation**

(Lossless)

Relu/Pool -> Conv



Sparse Storage Dense Compute

(Lossless)

Opportunity for Lossy Encoding

Precision Reduction Error

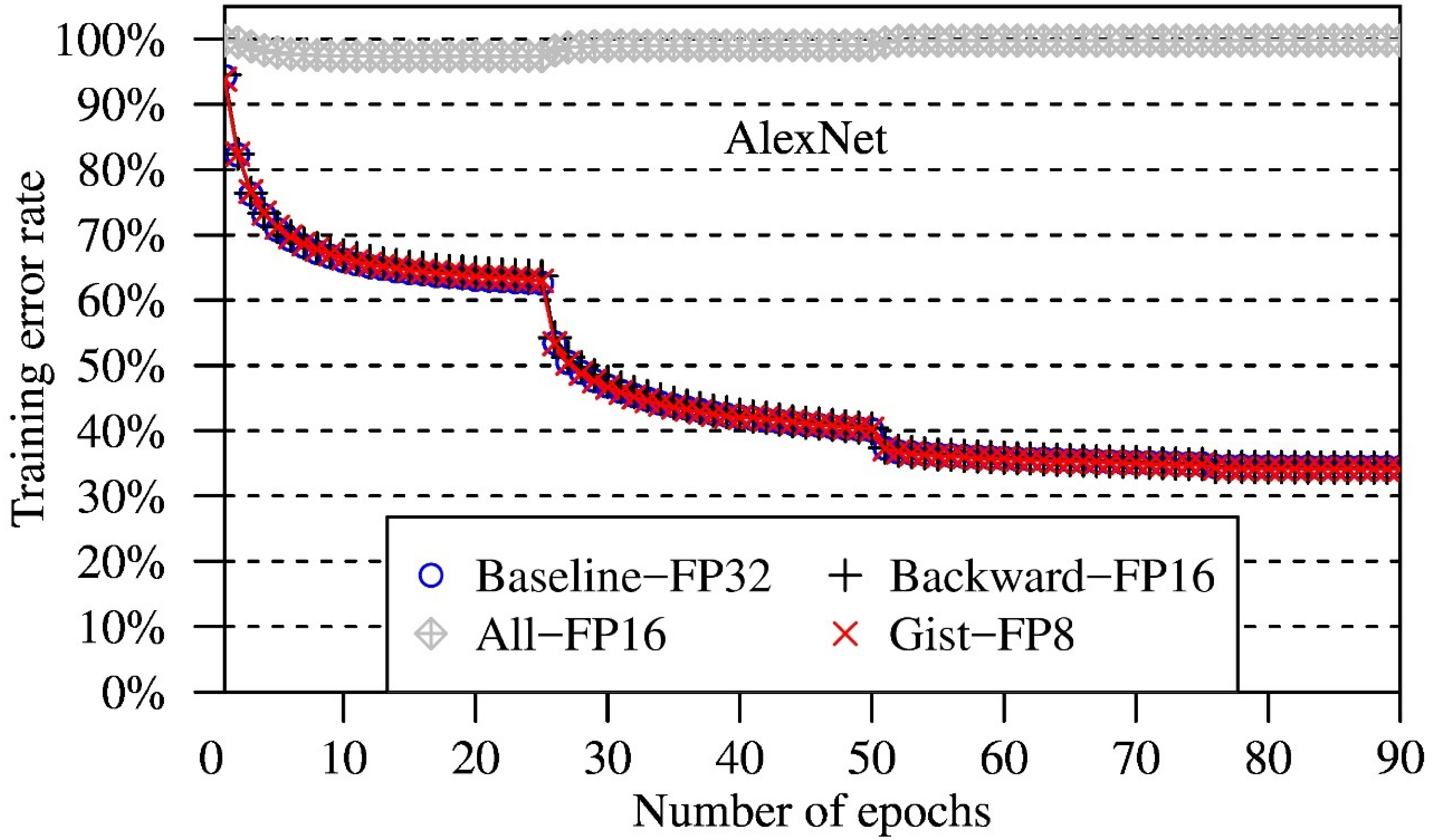
AlexNet : 16-bit doesn't train

Precision reduction in forward pass quickly degrades accuracy

Restricting precision reduction to the 2nd use results in aggressive bit savings with no effect on accuracy

Delayed Precision Reduction

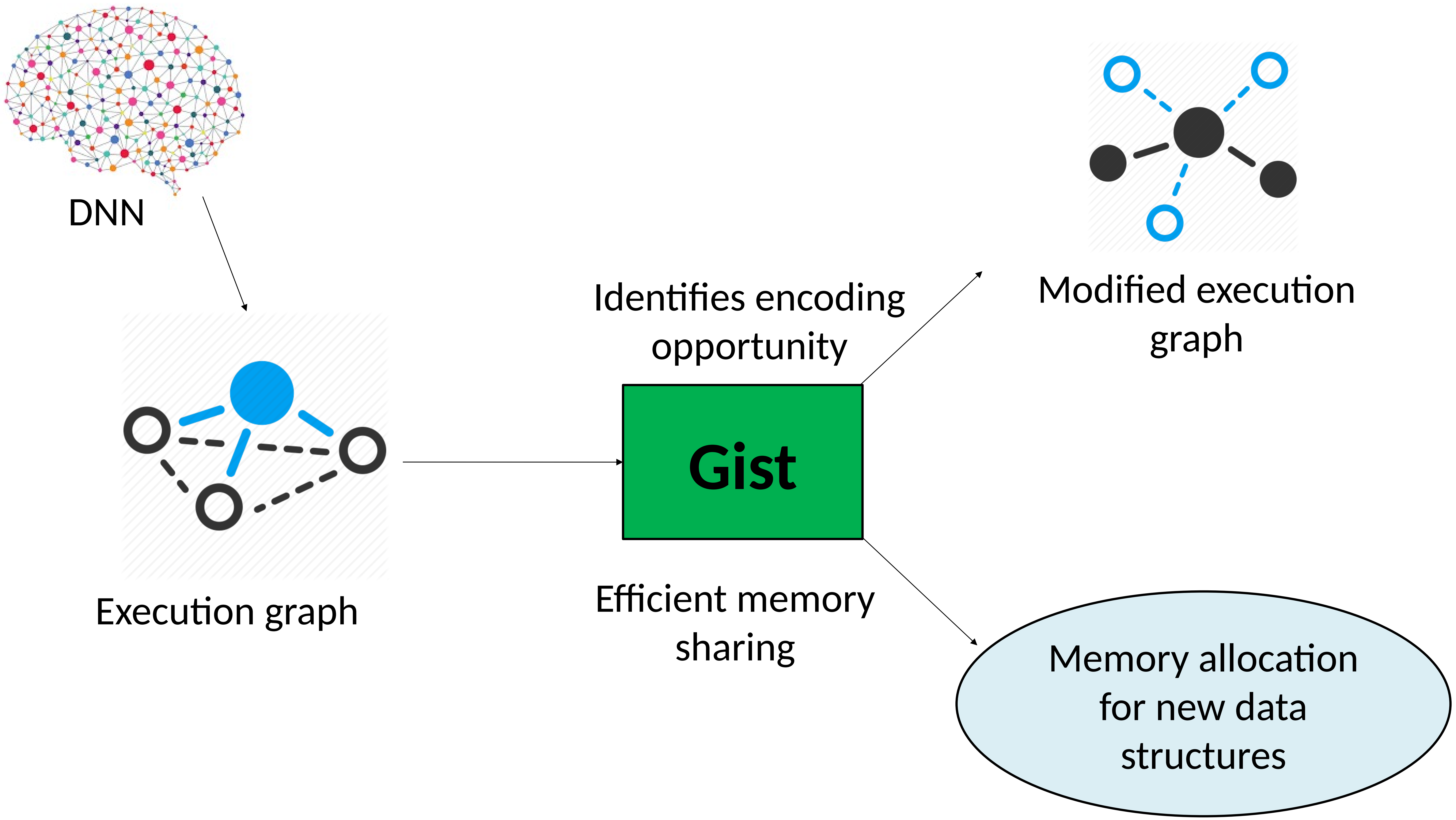
Training with Reduced Precision



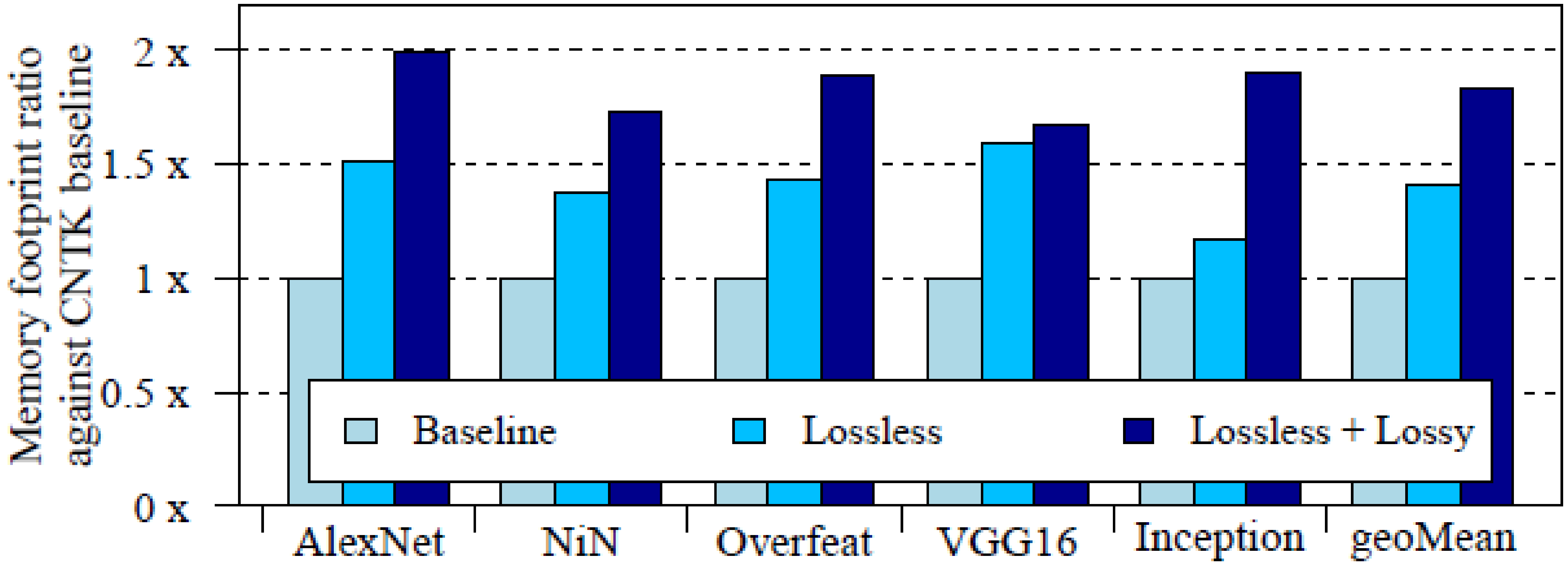
Delayed Precision Reduction

(Lossy)

Proposed System Architecture - Gist



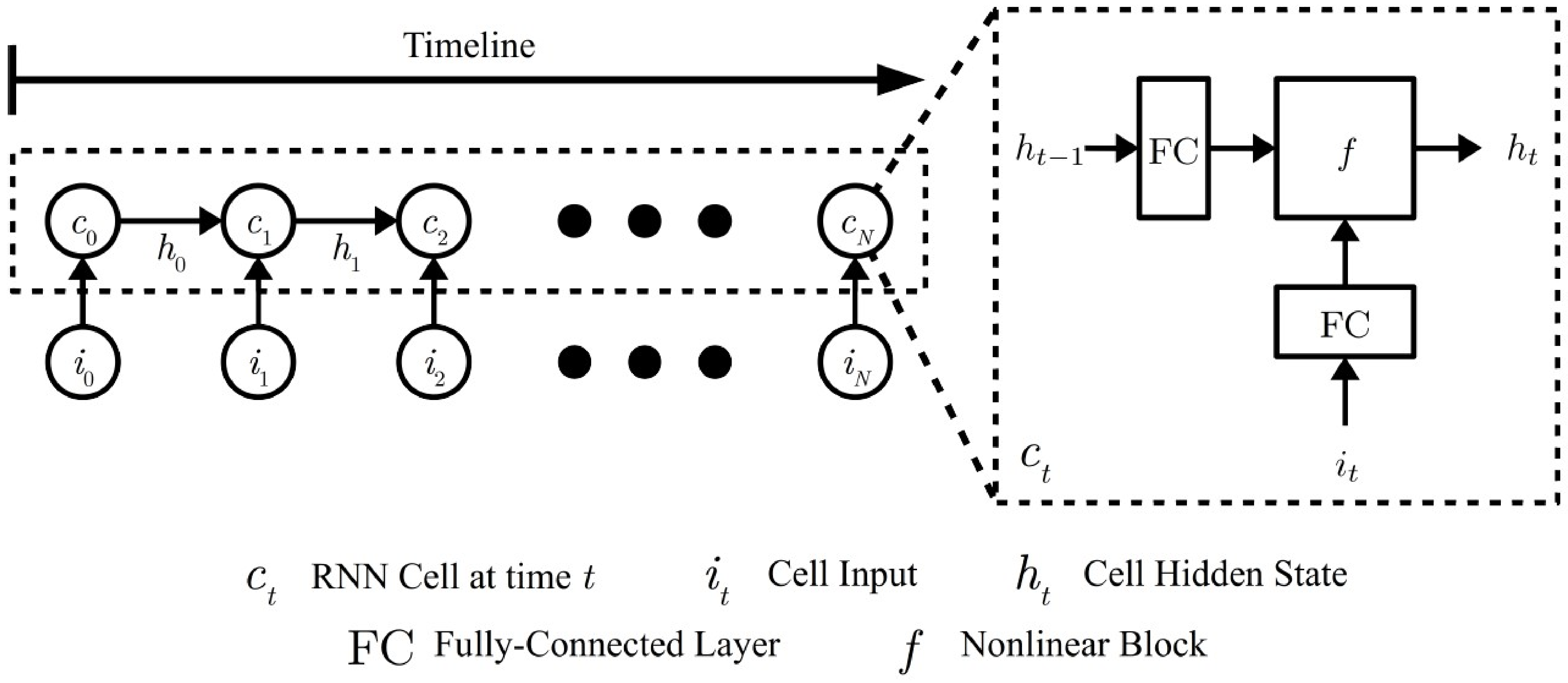
Compression Ratio

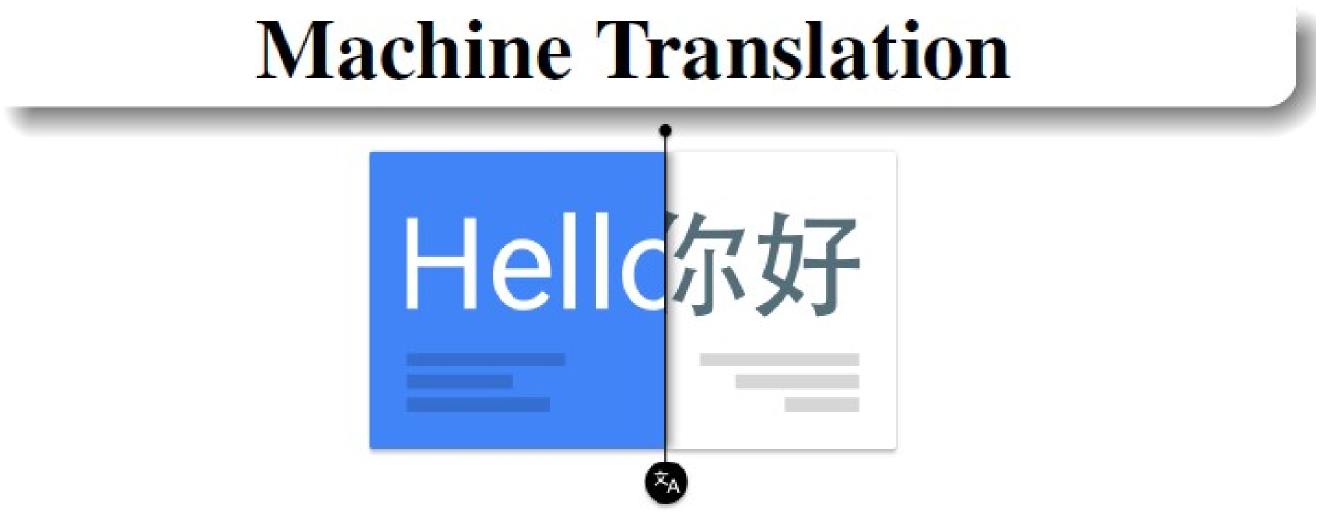


Up to 2X compression ratio
With minimal performance overhead

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

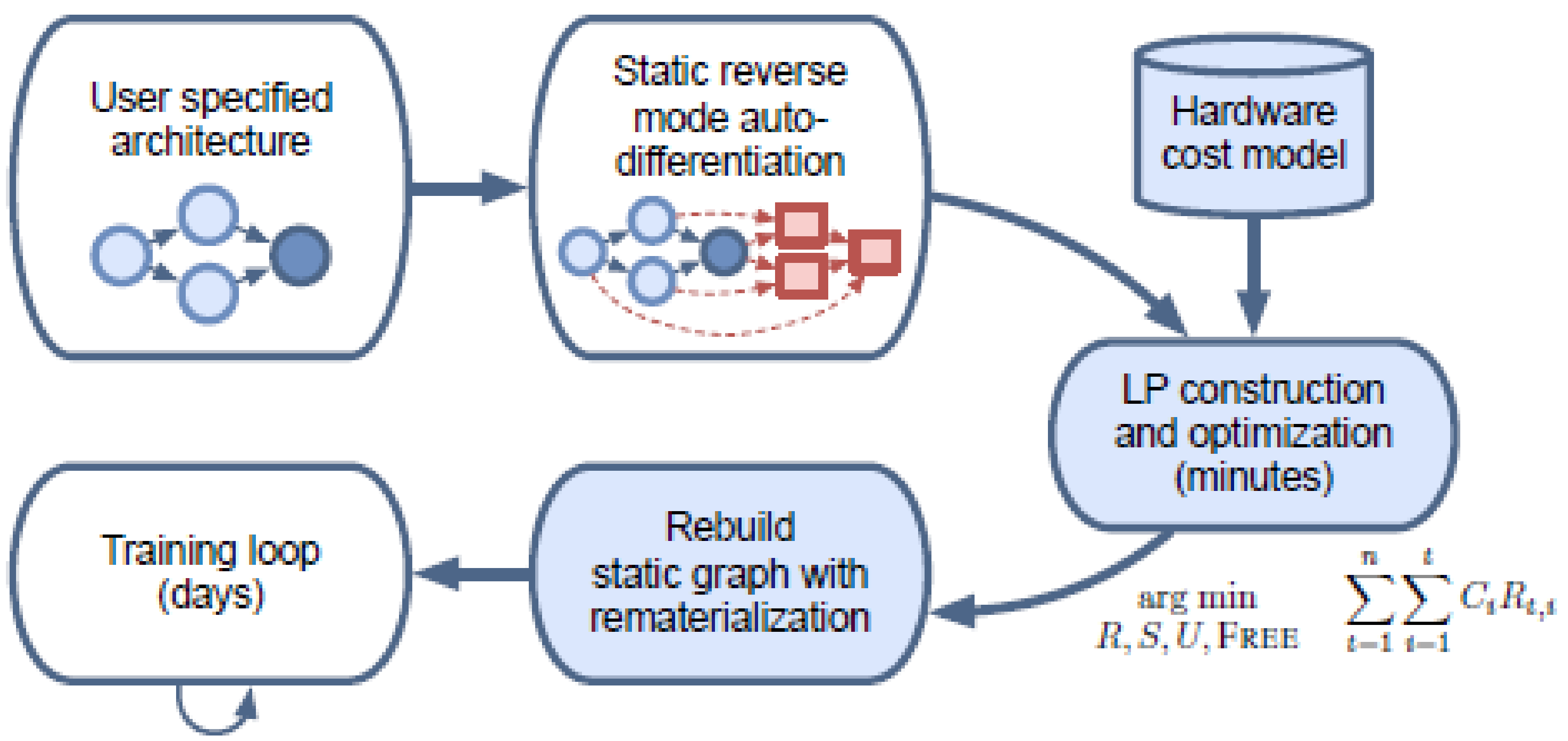


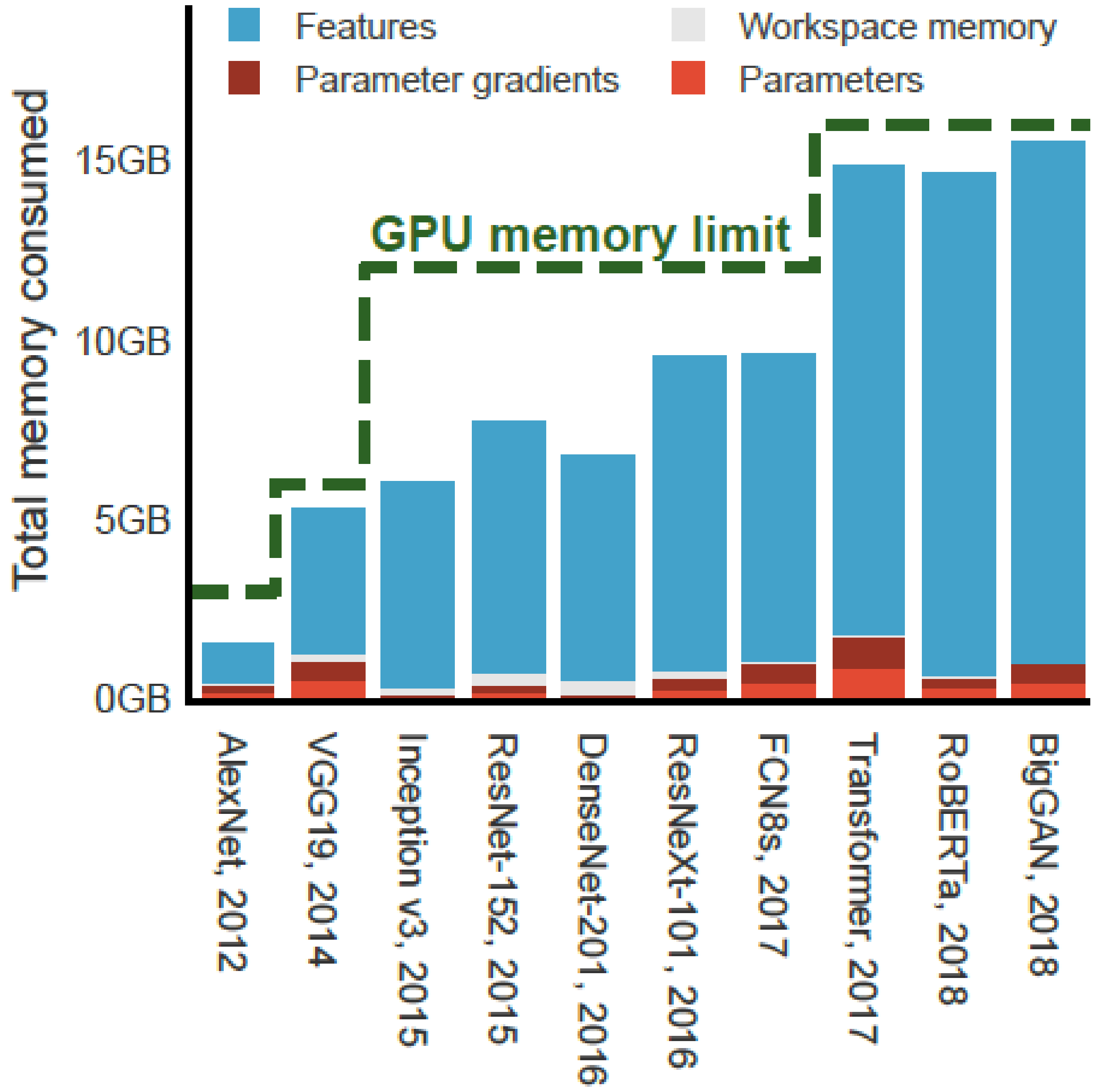


Echo: Compiler-based GPU Memory Footprint Reduction for LSTM RNN Training

Bojian Zheng et al.

ISCA 2020





CHECKMATE: BREAKING THE MEMORY WALL WITH OPTIMAL TENSOR REMATERIALIZATION

Paras Jain et al. (UC Berkeley)

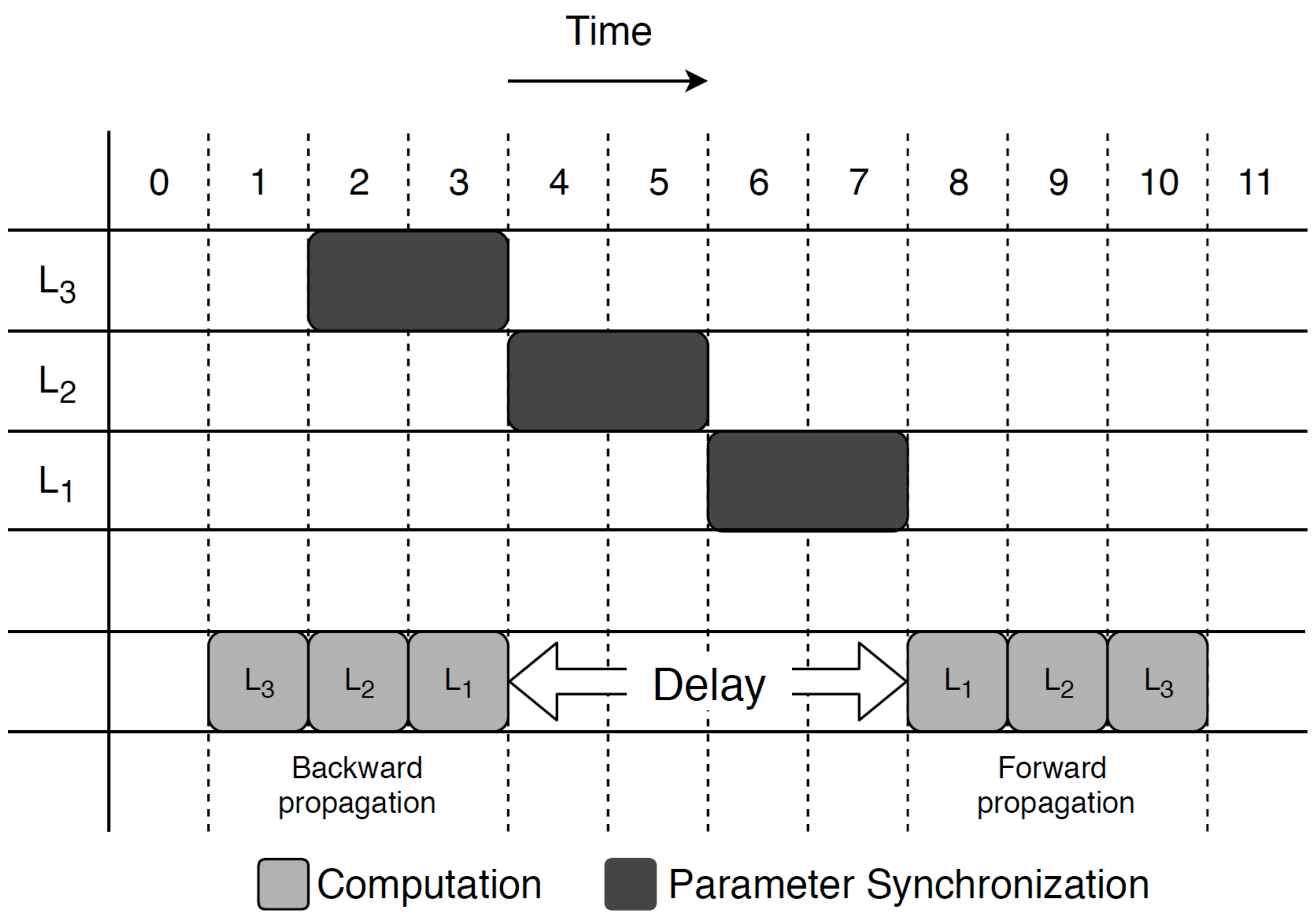
MLSys 2020

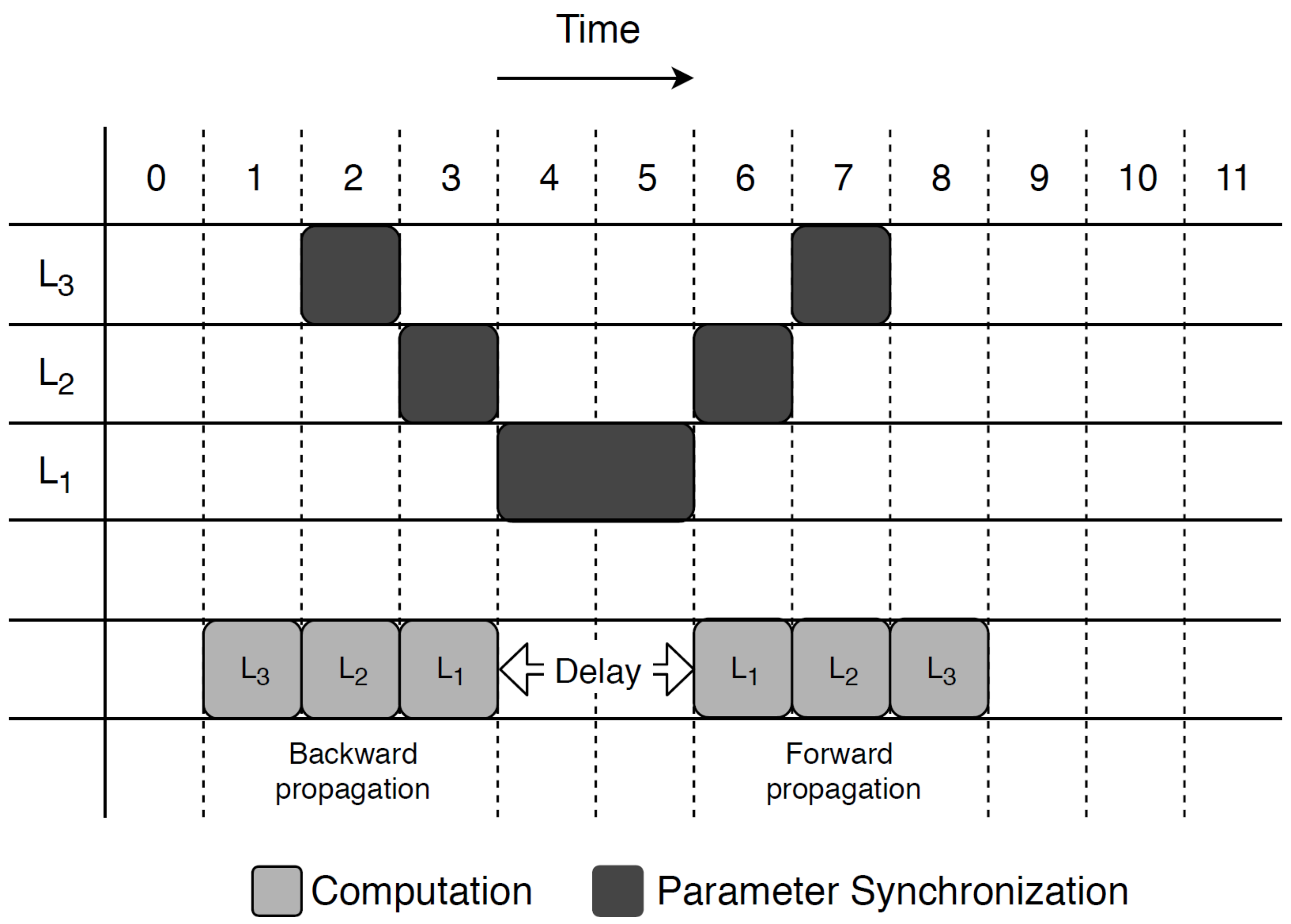
There are many more

- NeurIPS 2019
- Another paper at ISCA 2020 (jpeg encoding for CNNs)
- Tempo, NeurIPS 2022
- …

DNN Training and Inference : Trends and State-of-the-Art

2. Distributed Training: Algorithms and Networking





Priority-based Parameter Propagation (P3) for Distributed DNN Training

Anand Jayarajan et al.

SYSML 2019

P3 Followups

- TicTac (MLSys'19) from UIUC
- BytePS (SOSP'19) from ByteDance

PLink: Discovering and Exploiting Locality for Accelerated Distributed Training on the Public Cloud-based Distributed Systems

UW and Microsoft Research

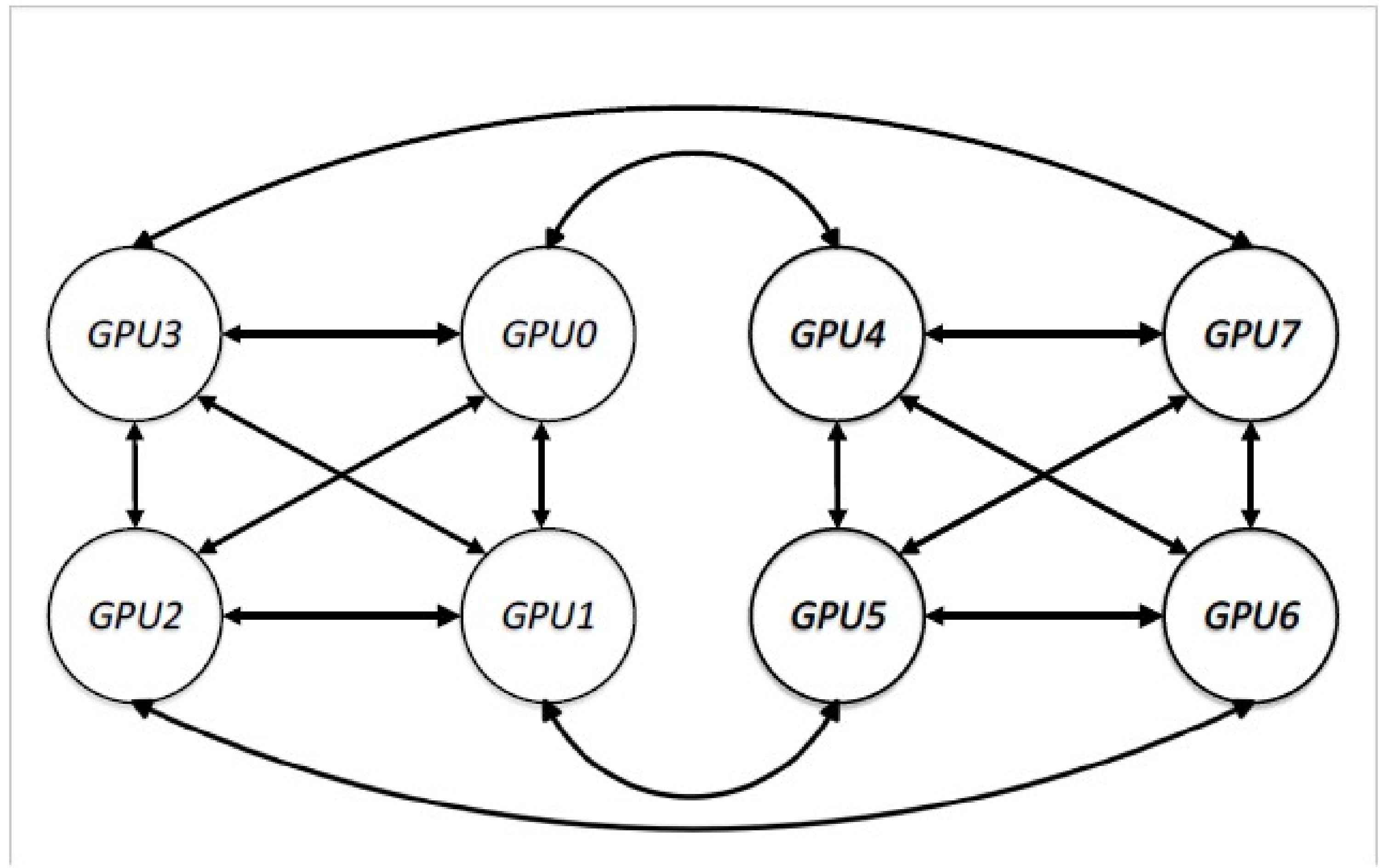
MLSys 2020

Blink: Fast and Generic Collectives for Distributed ML

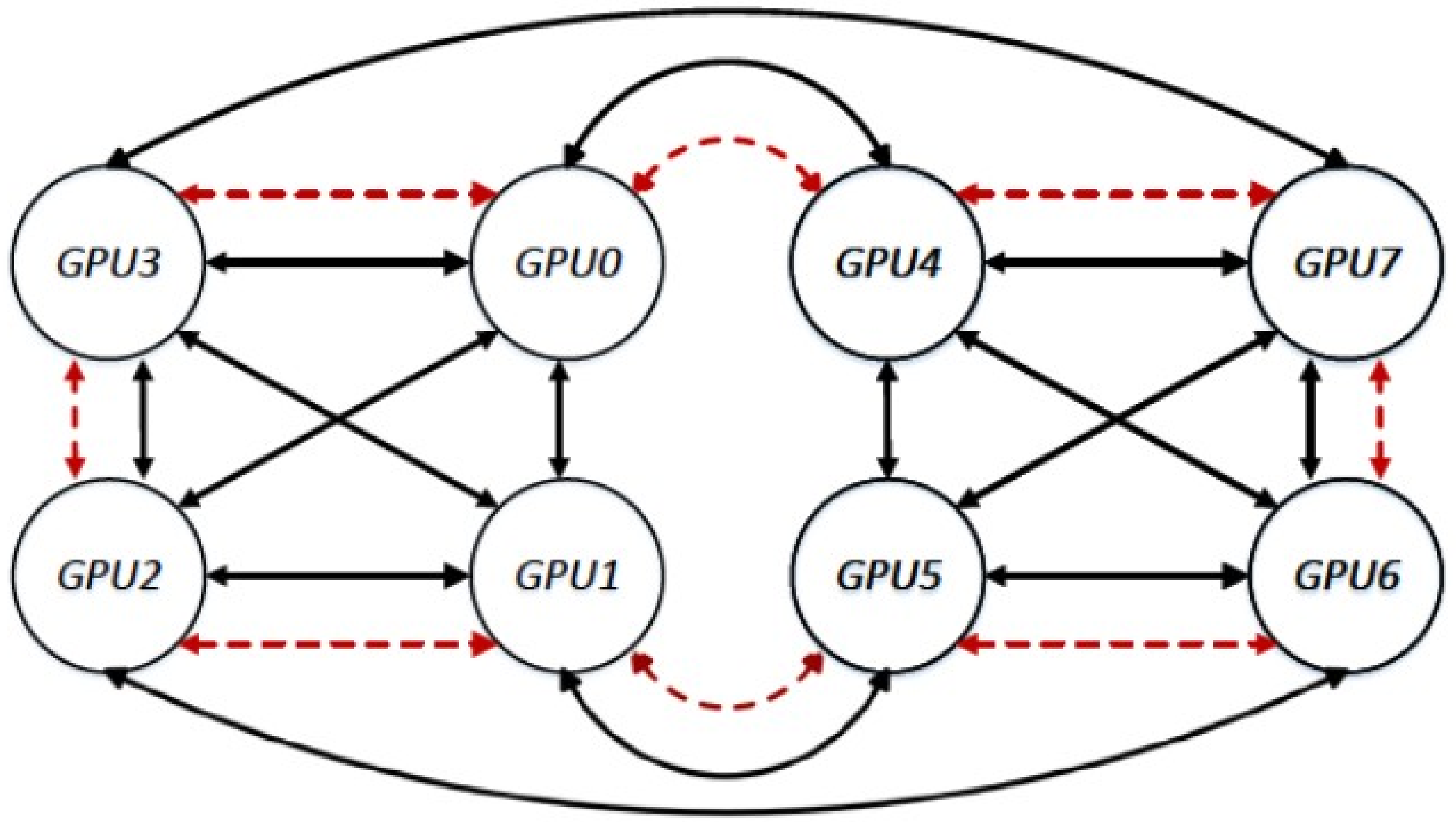
UC Berkeley, U of Wisconsin, and Microsoft Research

MLSys 2020

Challenge 1: Different server configurations



DGX1-P100 (NVLink 1st Gen, ~18GB/s)

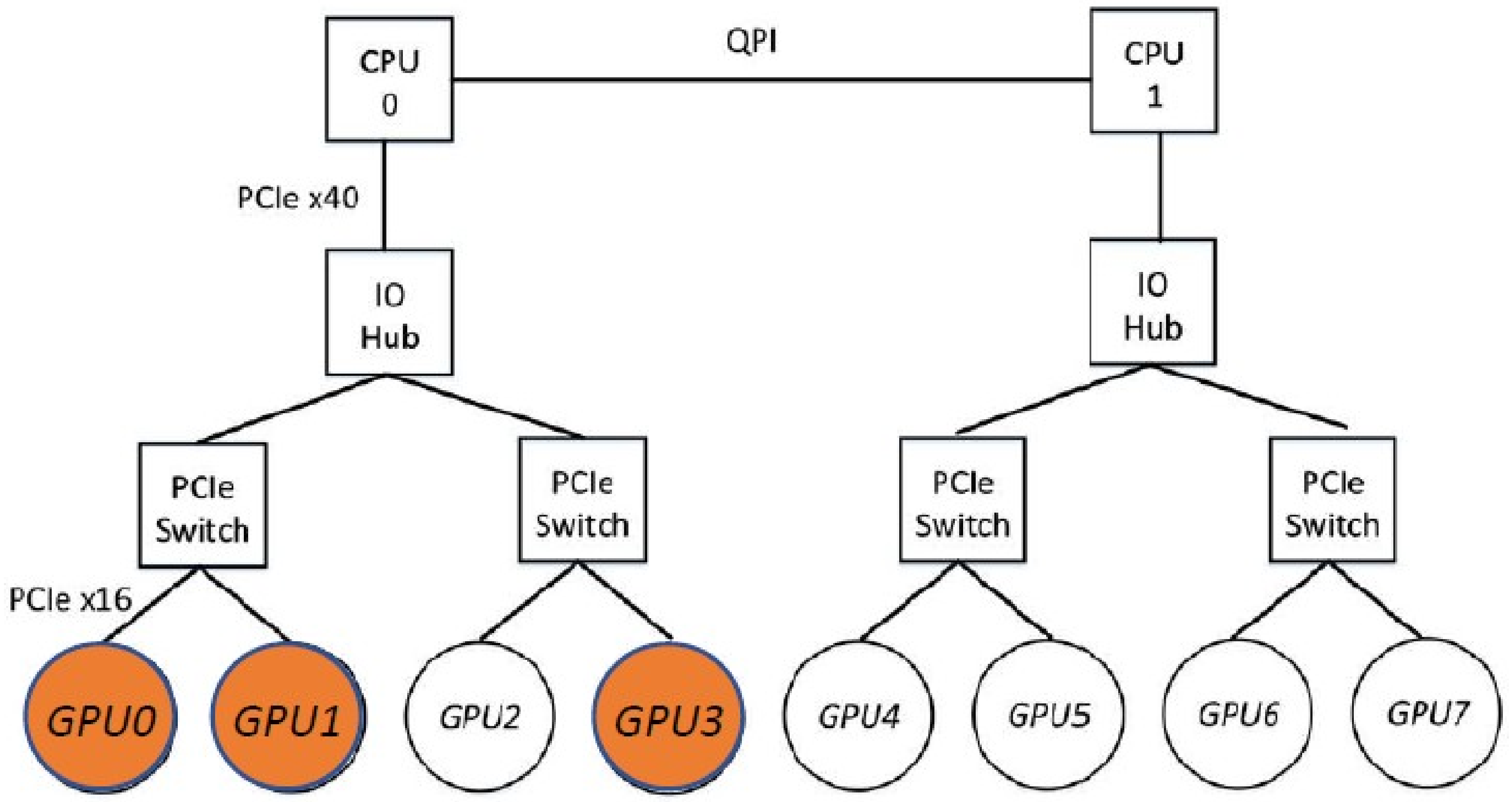


DGX1-V100 (NVLink 2nd Gen, ~23GB/s)

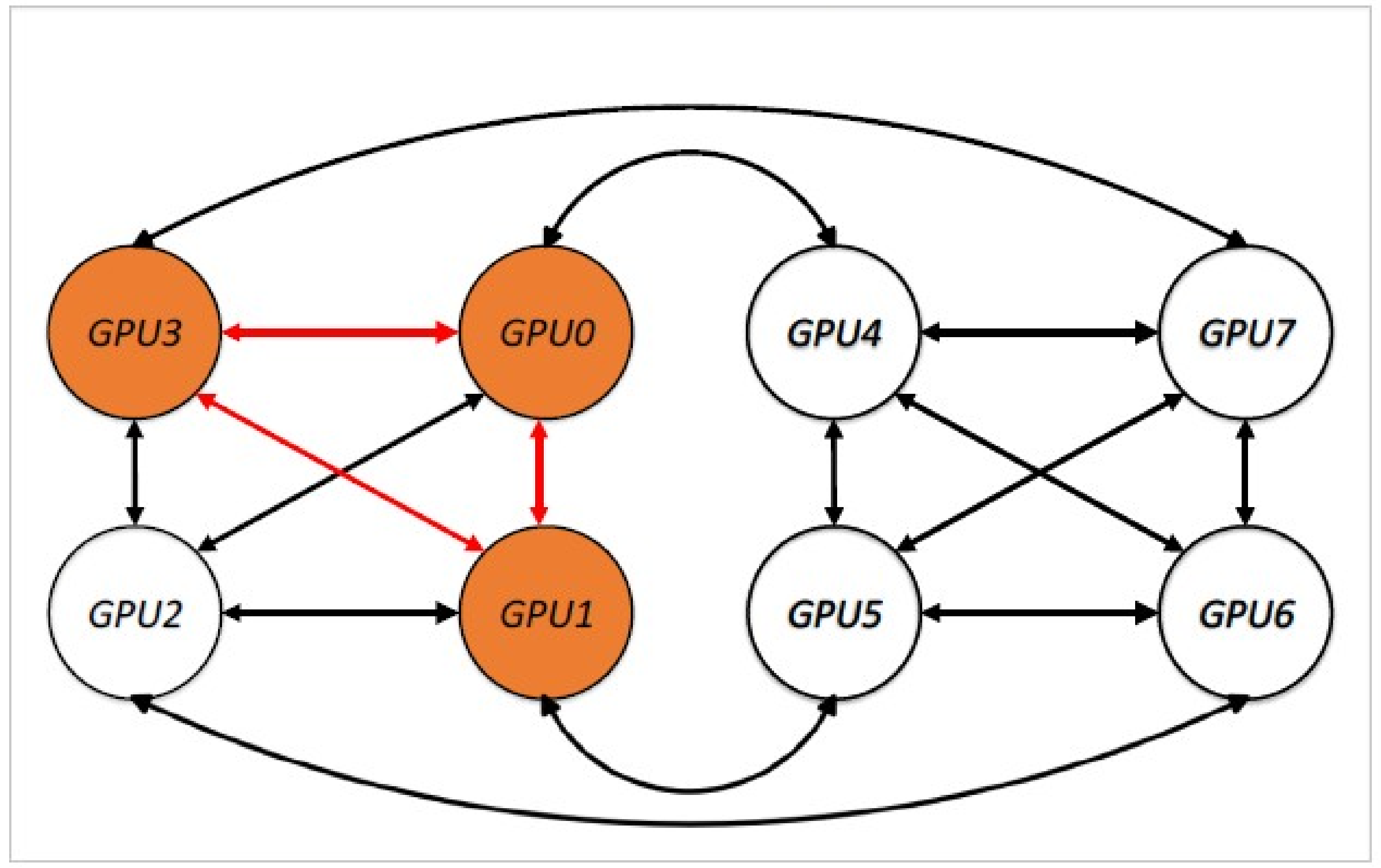
Protocols needs to be topology aware to effectively use hardware links.

Challenge 2: Link heterogeneity

PCIe topology

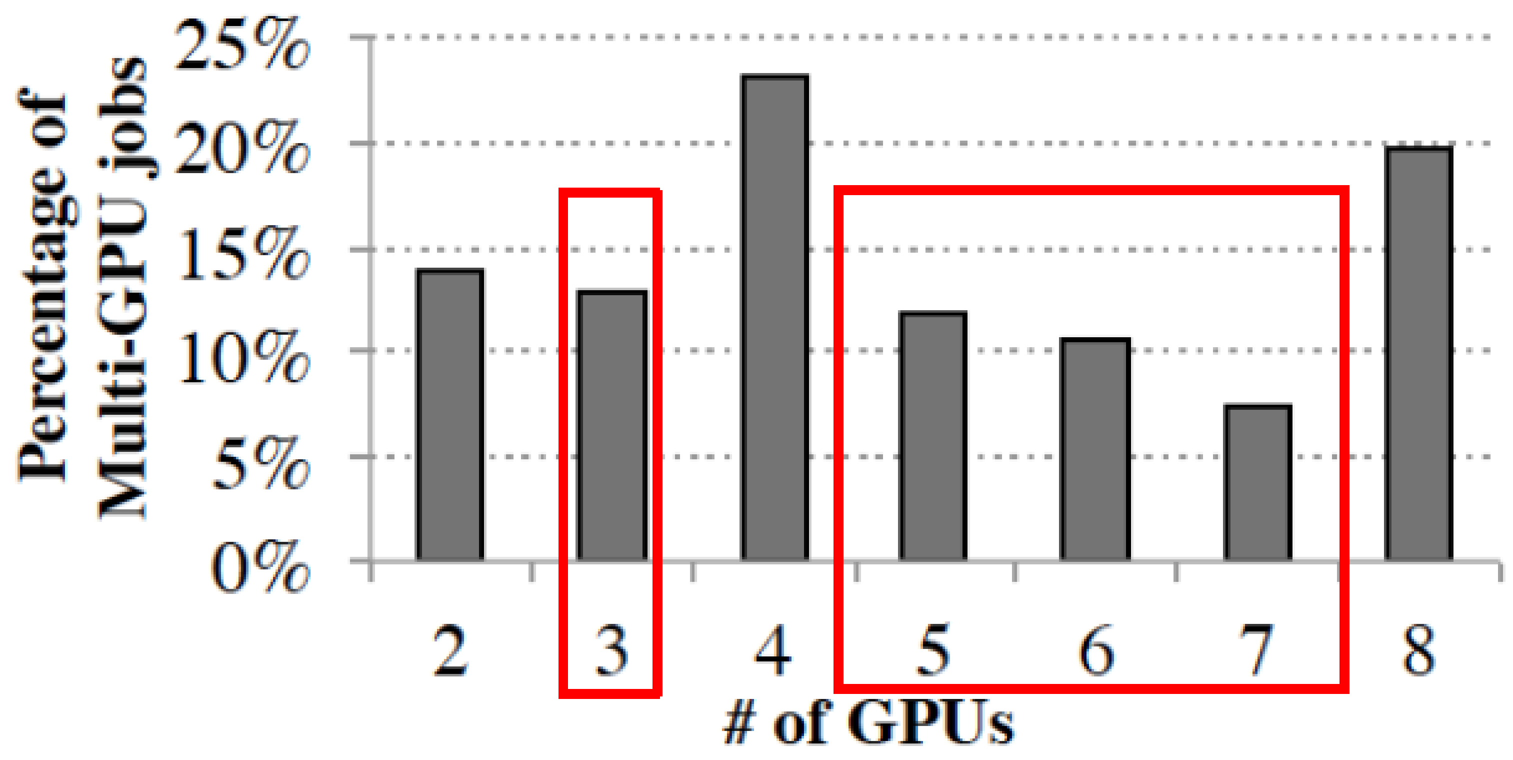


NVLink topology

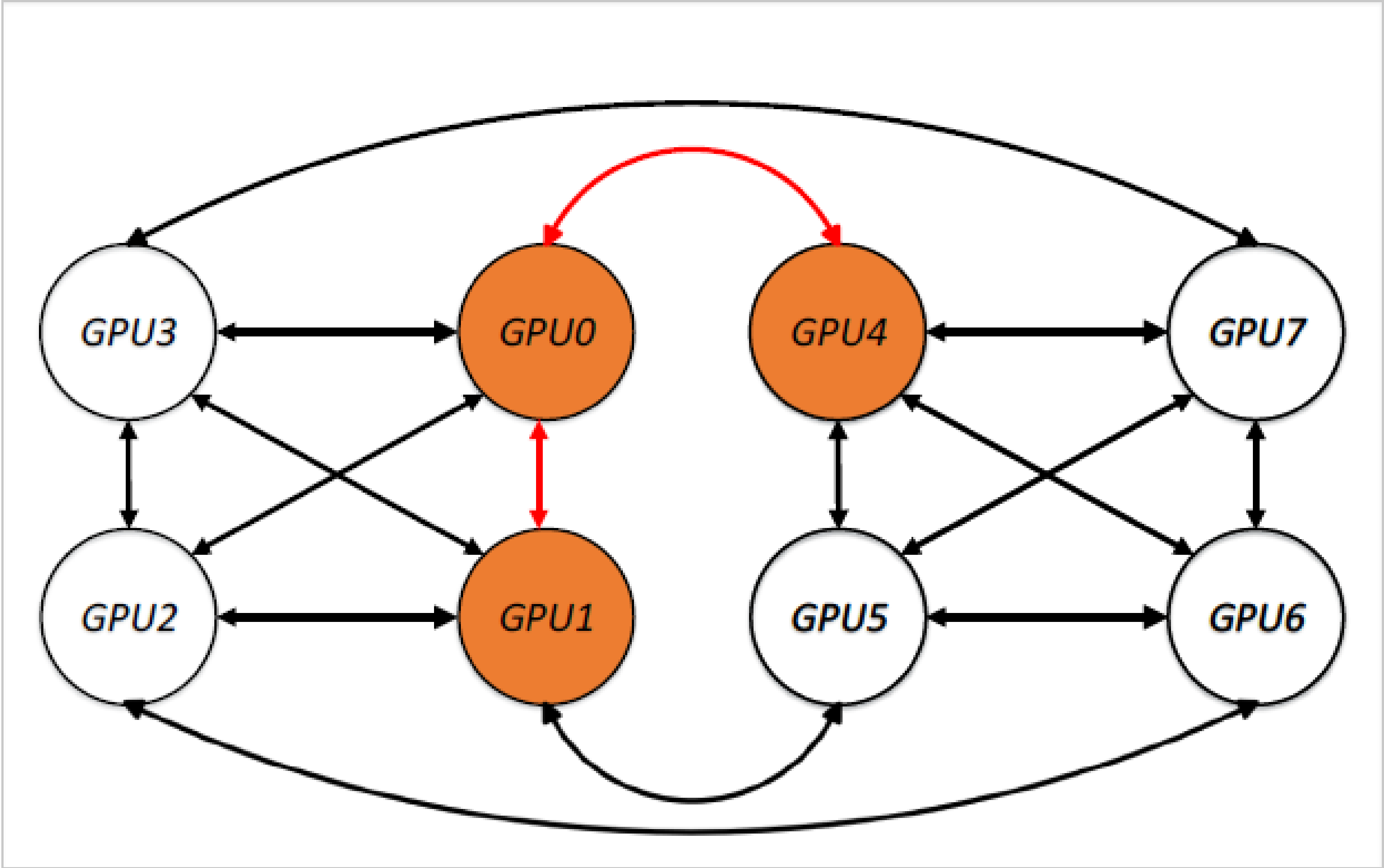


Ring-based collectives can only utilize homogeneous links.

Challenge 3: Fragmentation in multi-tenant clusters



Within each 8-GPU server, # of GPUs allocated to 40,000 multi-GPU jobs at Microsoft.



Why fragmentation?

Irregular topo. → no ring

Many cluster schedulers are not topology-aware.

Without support for efficient migration, DNN jobs must embrace fragmentation to avoid queuing delays.

Existing solutions (NCCL) fall back to PCIe if they cannot form a NVLink ring.

How Blink handles topology heterogeneity

Topology Heterogeneity		Blink
Different server configurations	→	Probe available links at job run time
Link heterogeneity	→	Concurrent data transfer over heterogenous links
Fragmentation in multi-tenant clusters (irregular topology)	→	Spanning trees (v.s. Rings) are more flexible and optimal.
	More →	NCCL-compatible API, seamless integration with TF, PyTorch, etc.

43

Scaling Back-Propagation by Parallel Scan Algorithm

Shang Wang[1,2], Yifan Bai[1], Gennady Pekhimenko[1,2]

1 Computer Science UNIVERSITY OF TORONTO

2 VECTOR INSTITUTE

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).

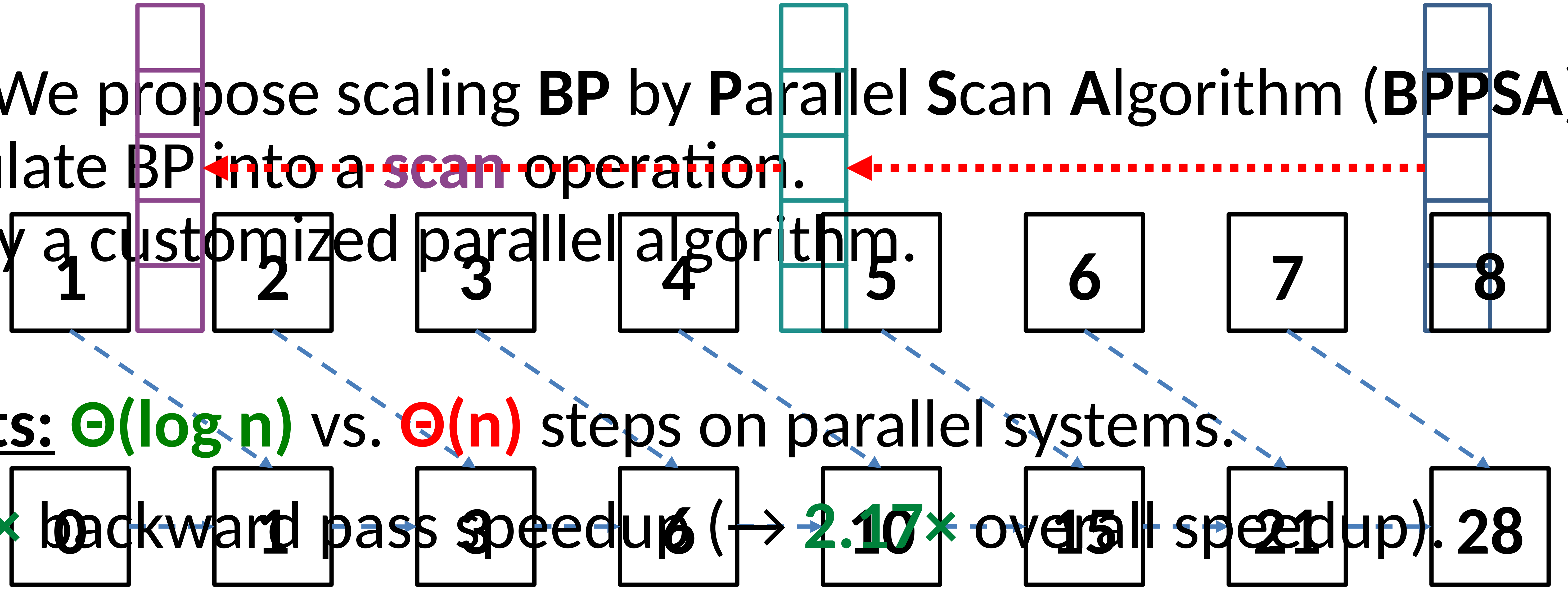
Problem: BP imposes a **strong sequential dependency** along layers during the gradient computations.

Key idea: We propose scaling **BP** by **P**arallel **S**can **A**lgorithm (**BPPSA**):

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

Key Results: **Θ(log n)** vs. **Θ(n)** steps on parallel systems.

Up to **108×** backward pass speedup (→ **2.17×** overall speedup).



Back-propagation[1] (BP) Everywhere

PyTorch

theano

Caffe

mxnet

Chainer

Microsoft Cognitive Toolkit

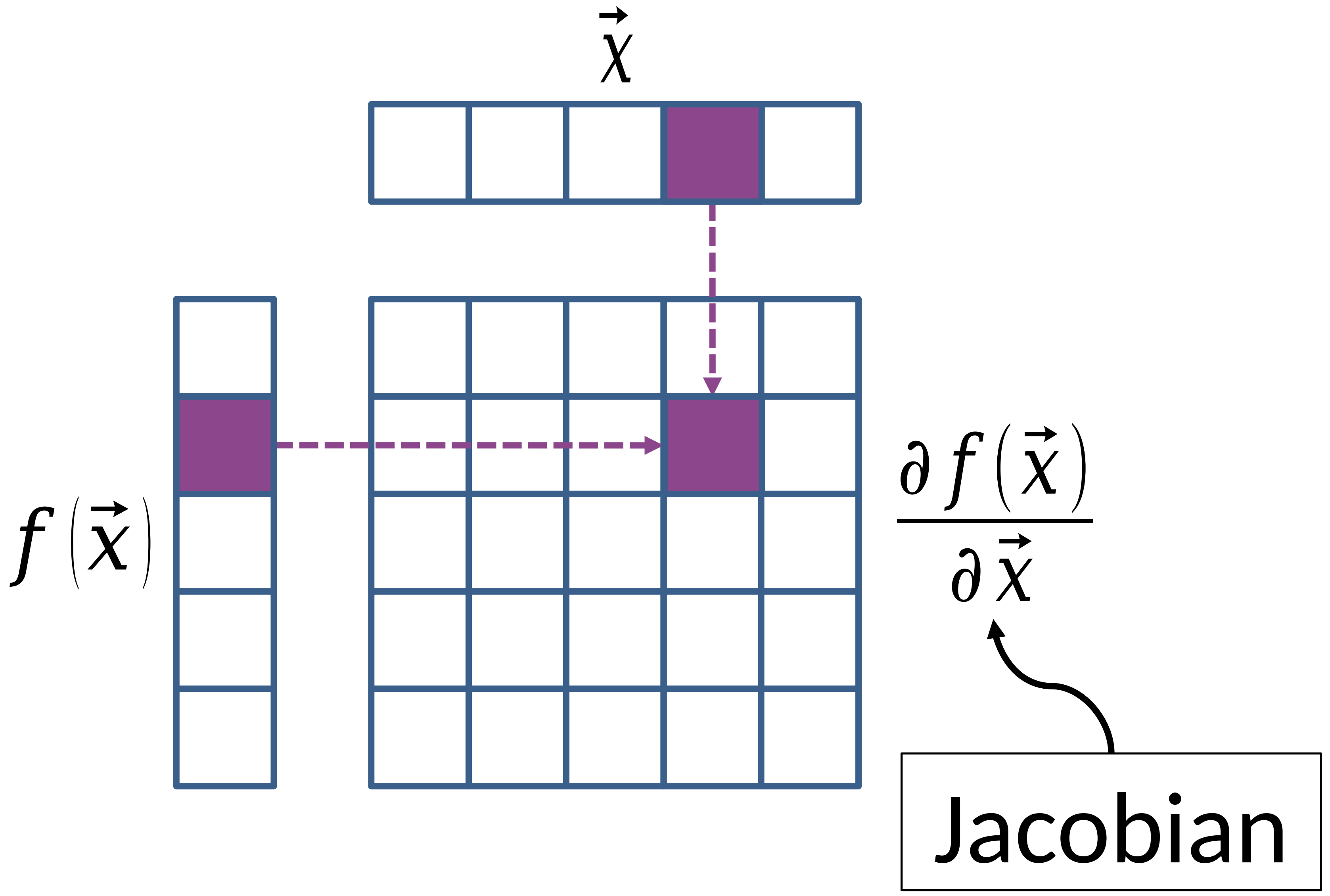
aka.ms/CognitiveToolkit

Microsoft

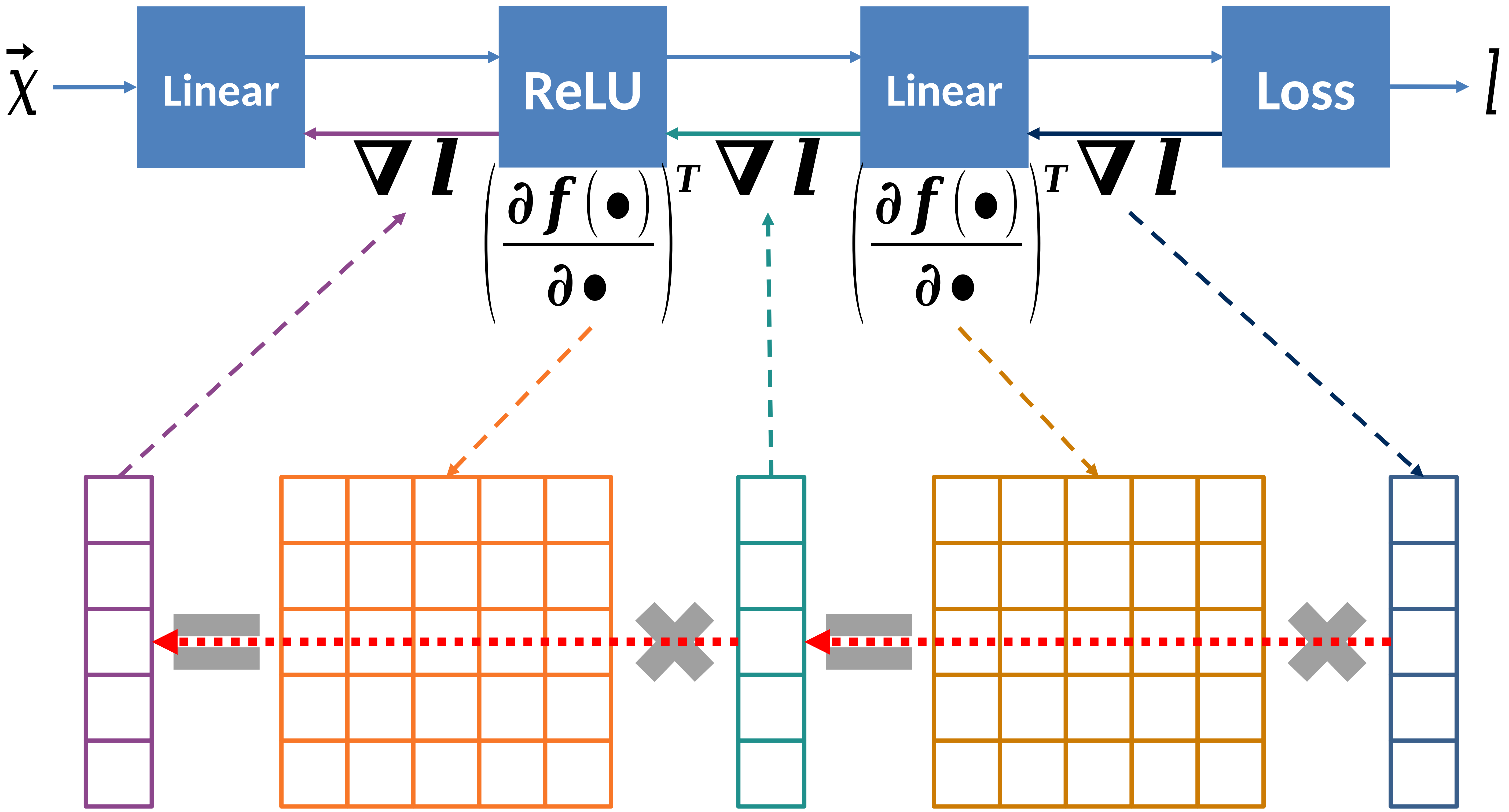
TensorFlow

[1]Rumelhart et al. "Learning representations by back-propagating errors.", Nature (1986)

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.

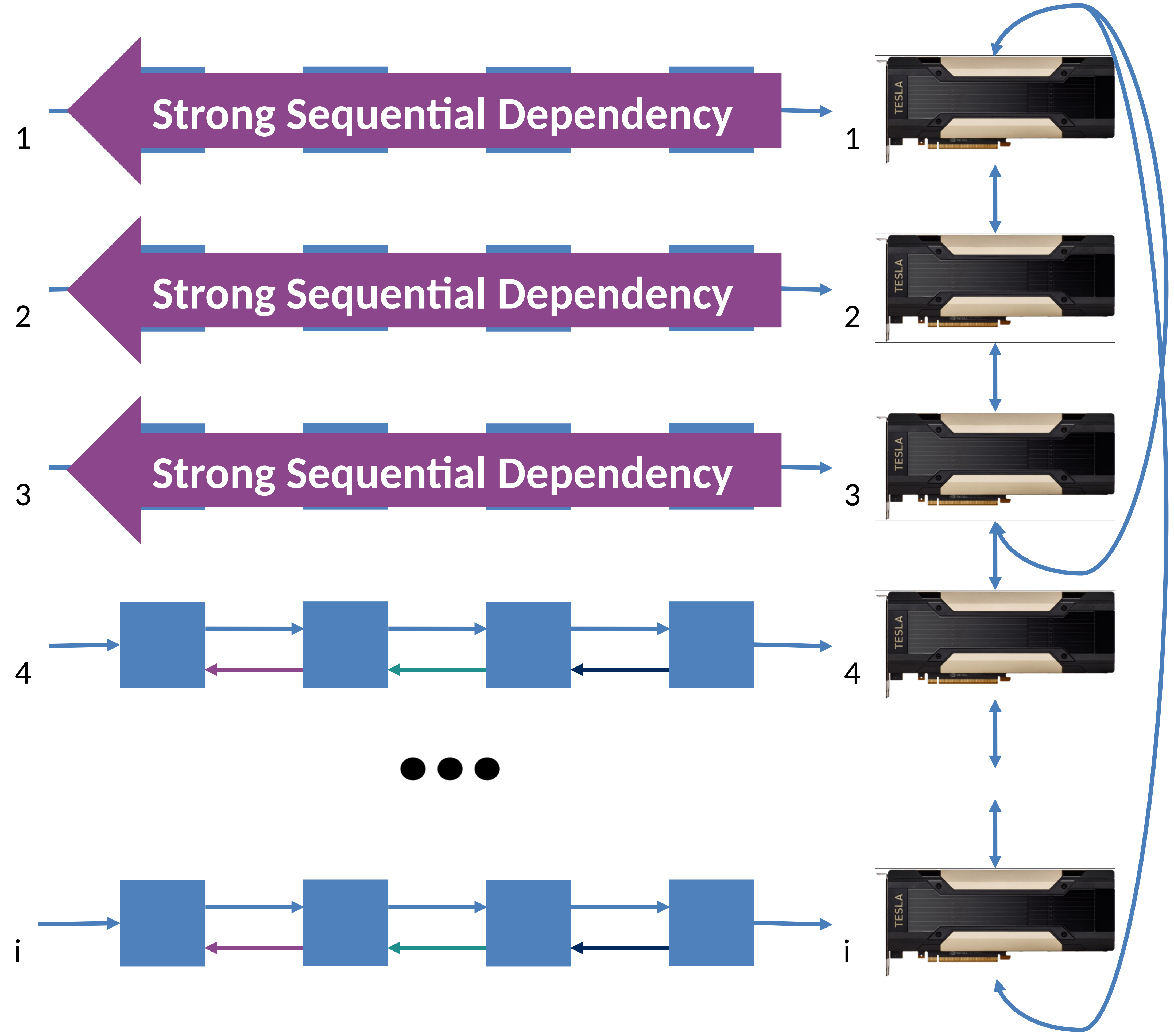
Data Parallel Training

Respects BP's strong sequential dependency.

Conceptually **simple**, **widely used**.

Effectively increases the batch size:
- **Generalization gap**[1]
- **Batch size scaling limit**[2]

Constraint: The model **must** fit in one device.



[1] Keskar, Nitish Shirish et al. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima." ICLR (2017)
[2] Shallue, Christopher J. et al. "Measuring the Effects of Data Parallelism on Neural Network Training." Journal of Machine Learning Research 20 (2019)

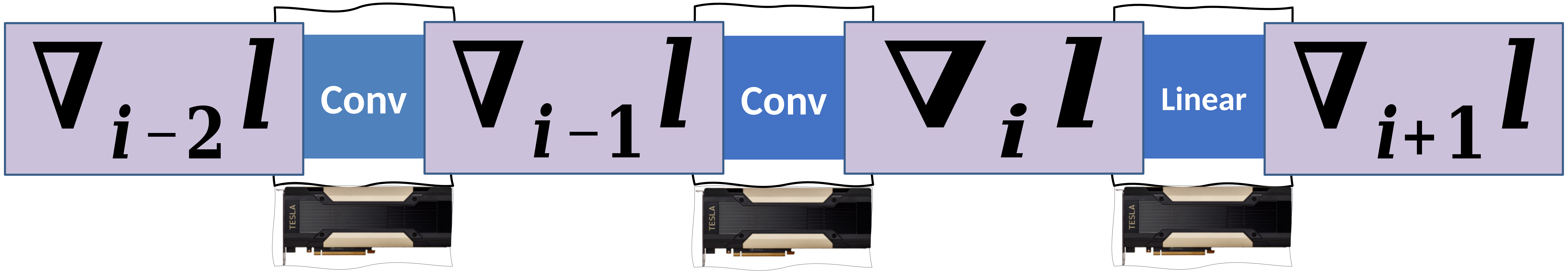
Model Parallel Training

Used when the model cannot fit in one device.

BP's strong sequential dependency **limits scalability**.

Prior works on **pipeline parallel training**[1,2] to mitigate such problem, but have their own limitations:

- **Linear** per-device space complexity.
- Trade-off between "**bubble of idleness**" vs. potential **convergence affect**.



[1]Harlap, Aaron et al. "PipeDream: Fast and Efficient Pipeline Parallel DNN Training." SOSP (2019)
[2]Huang, Yanping et al. "GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism." NeurIPS (2019)

Rethinking BP from an Algorithm Perspective

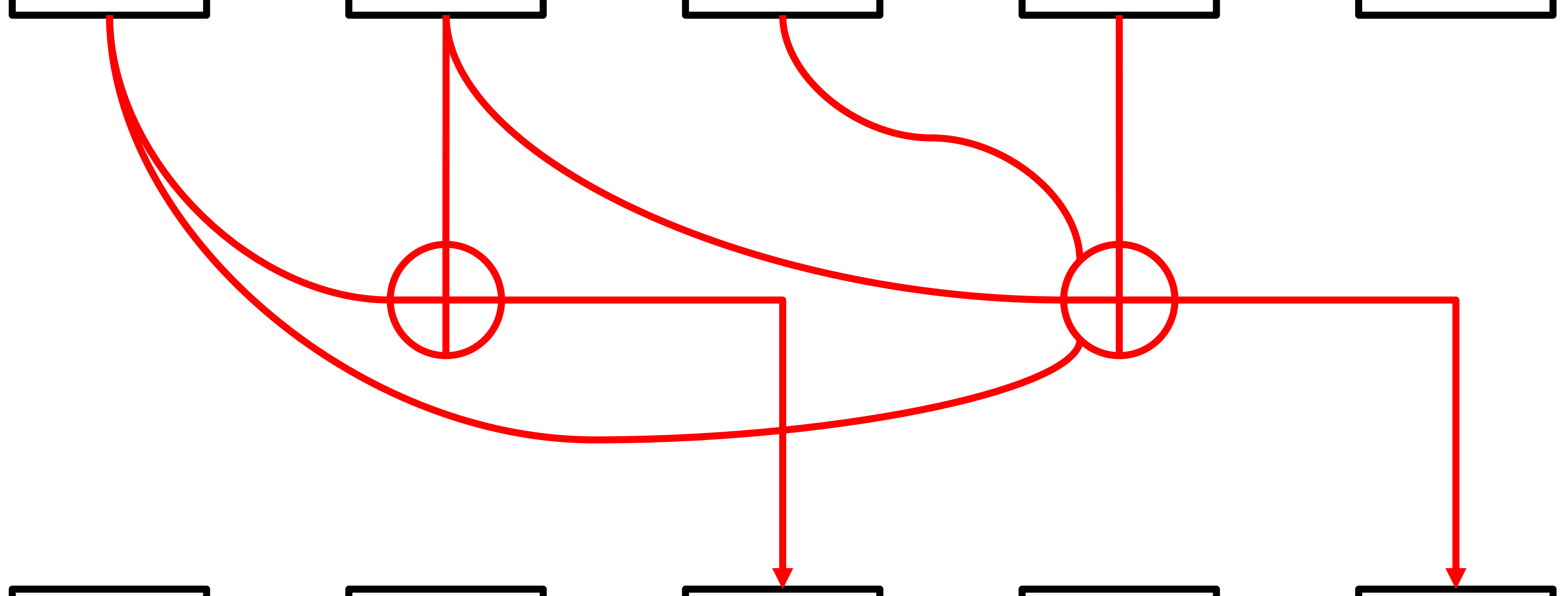
- Problems with strong sequential dependency were (80'), but in a much simpler context.

- We propose scaling **B**ack-**P**ropagation by **P**arallel **S**can **A**lgorithm (**BPPSA**):
  - Reformulate BP as a **scan** operation.
  - Scale BP by a **customized Blelloch Scan** algorithm.
  - Leverage **sparsity** in the Jacobians.

What is a Scan[1] Operation?

Binary, **associative** operator: **+** Identity: **0**

Input sequence: 1 2 3 4 5 6 7 8



Exclusive scan: 0 1 3 6 10 15 21 28

Compute partial reductions at each step of the sequence.

[1]Blelloch, Guy E. "Prefix sums and their applications". Technical Report (1990)

Linear Scan

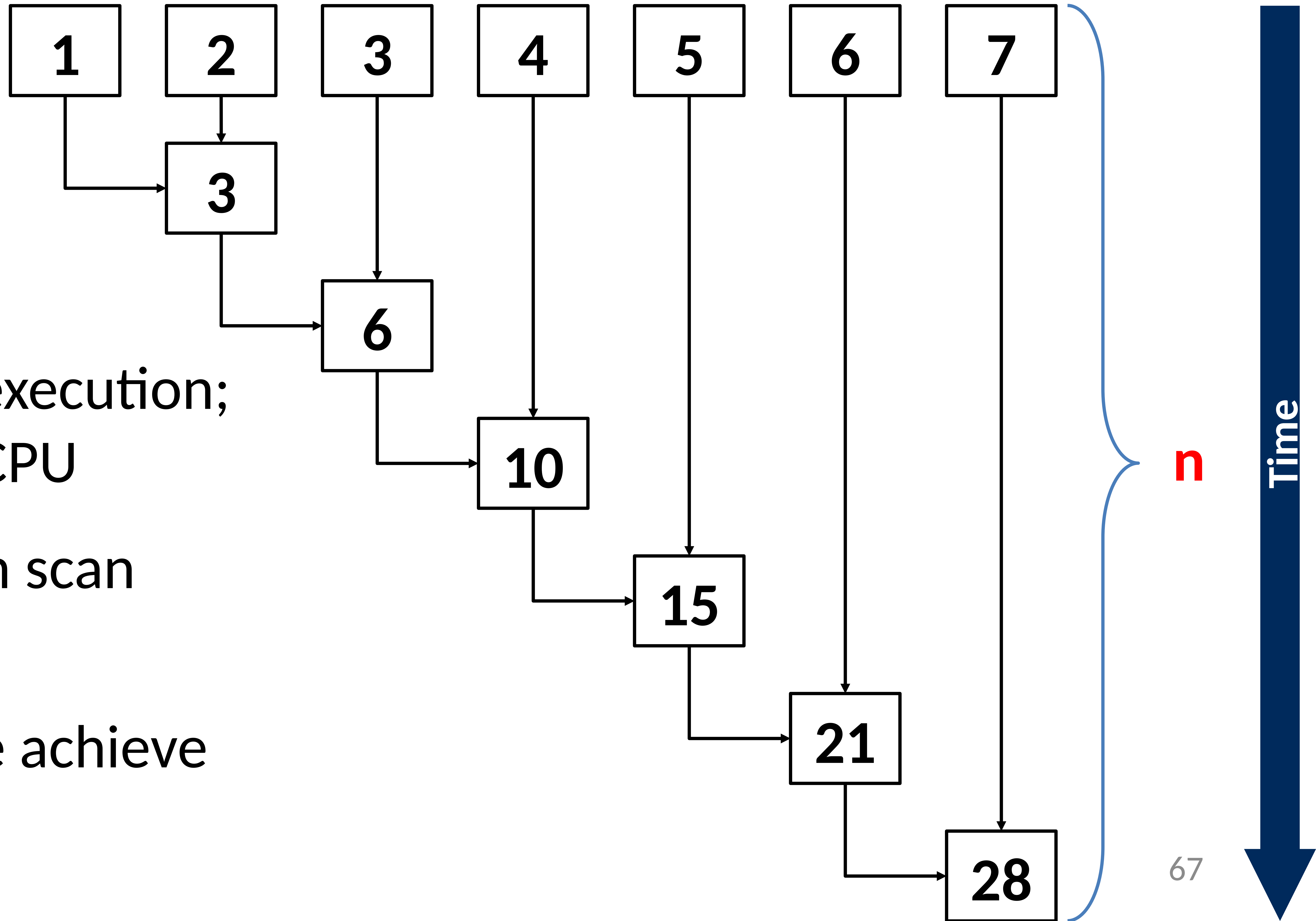
Step: executing the operator once.

Number of Elements (**n**)

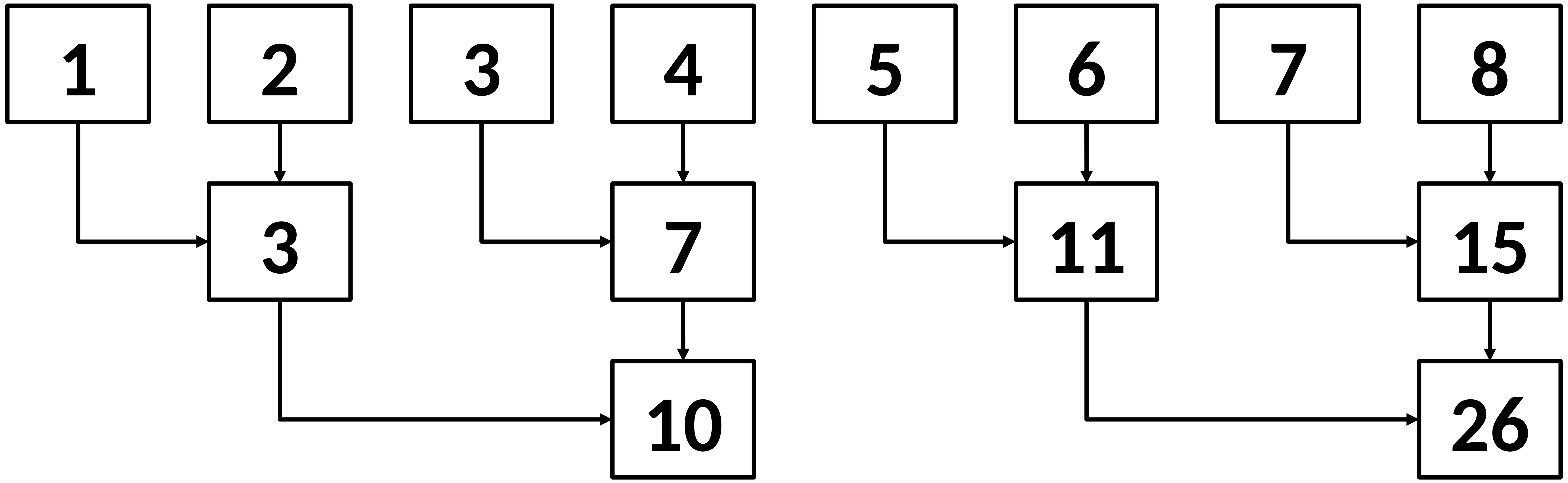
Worker (**p**): an instance of execution; e.g., a core in a multi-core CPU

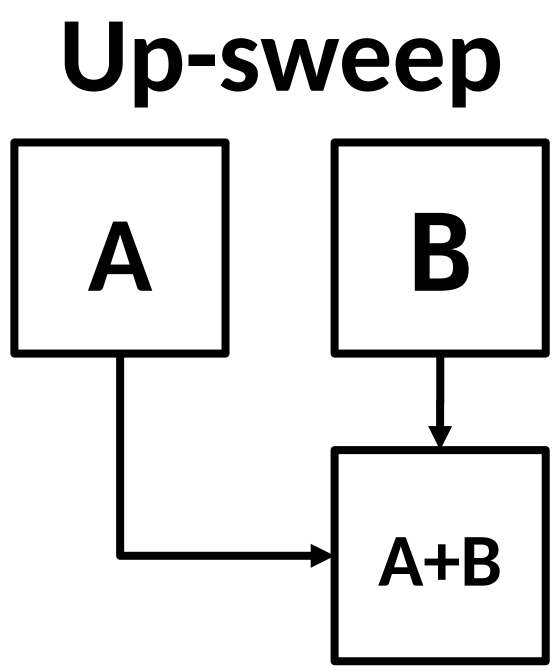
On a single worker: perform scan linearly; takes **n** steps.

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



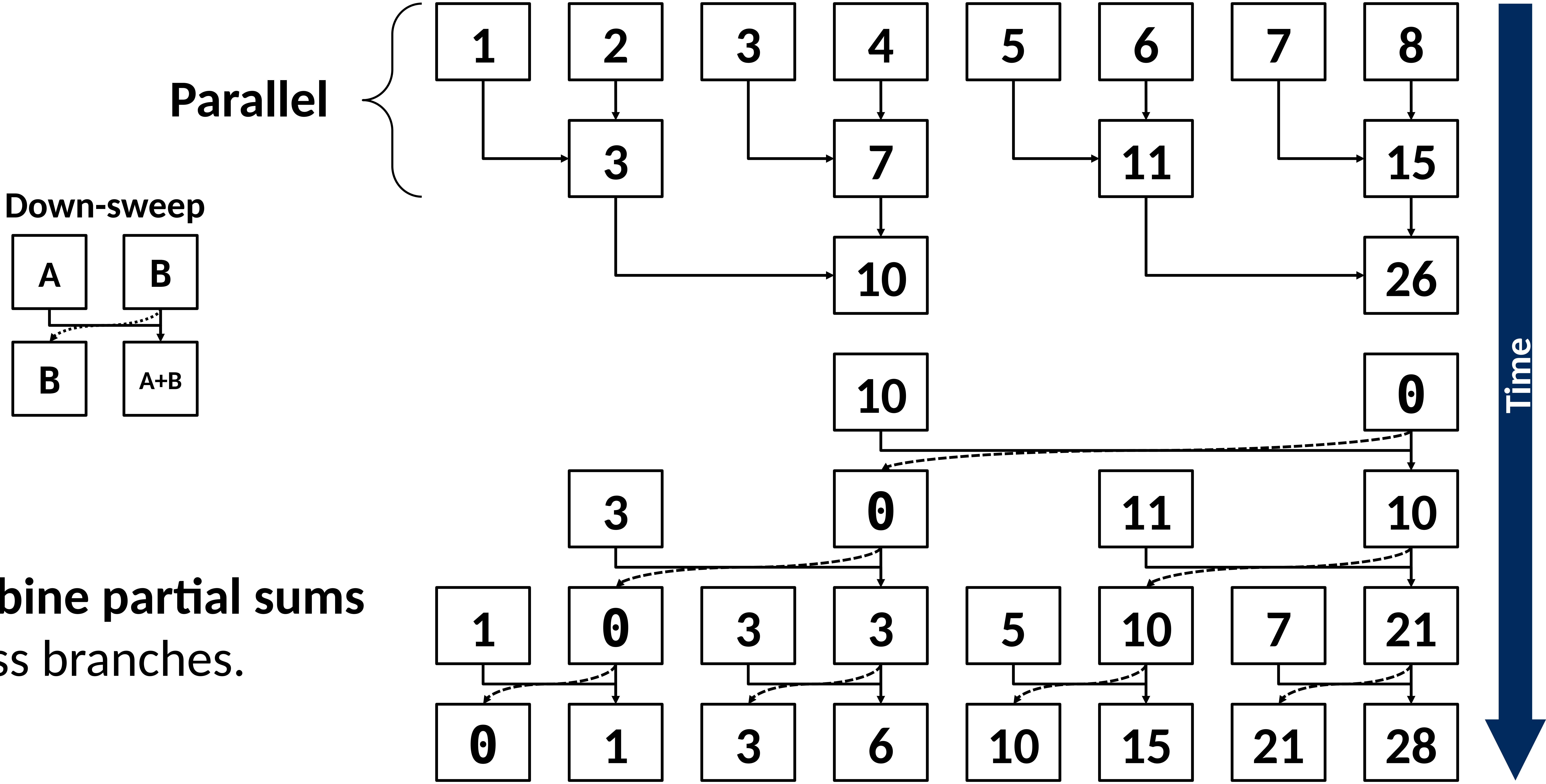
Blelloch Scan: ① Up-sweep Phase





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

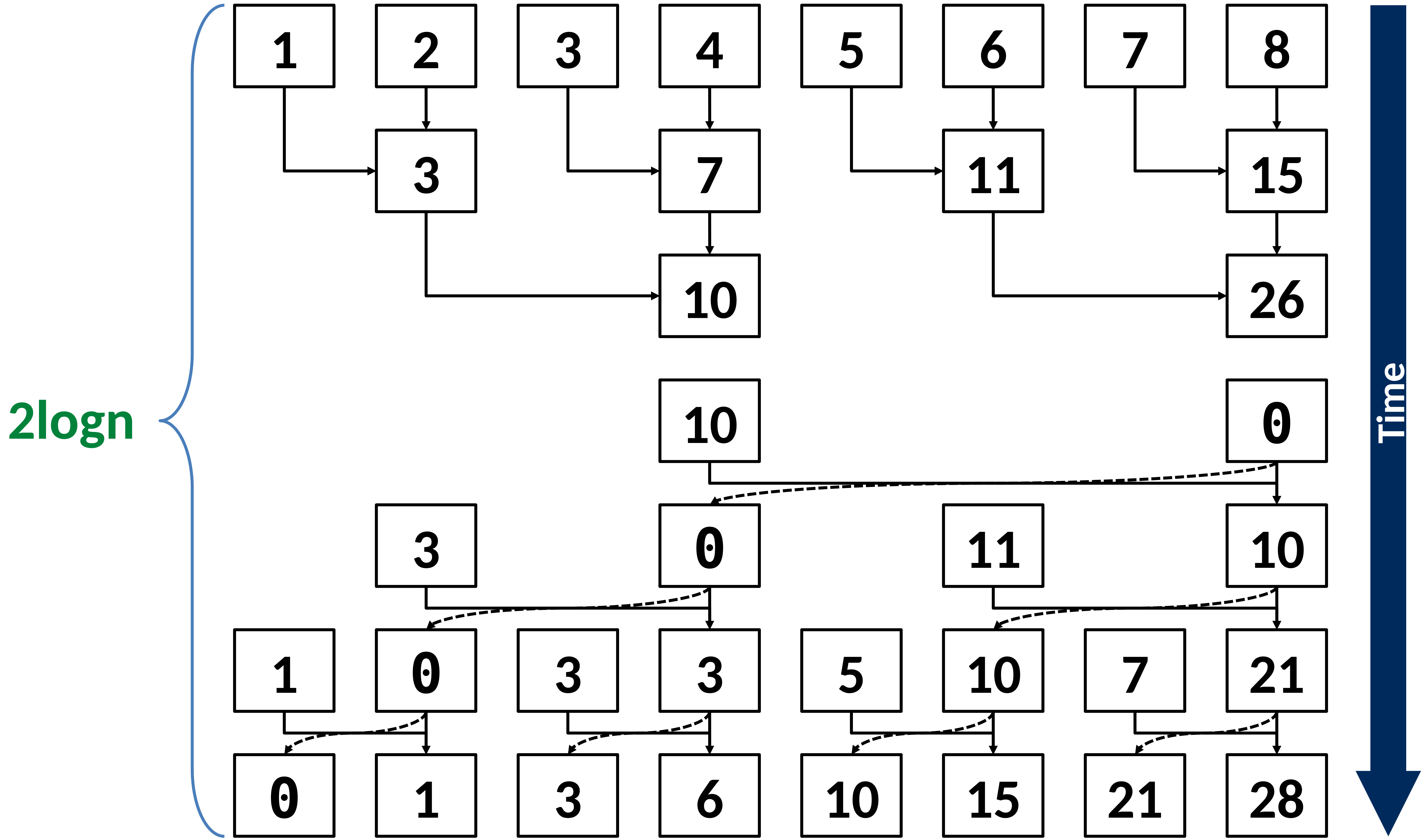
Blelloch Scan: ② Down-sweep Phase



Combine partial sums across branches.

Blelloch Scan: Efficiency

Logarithmic steps along the critical path.



Reformulate BP as a Scan Operation

$G_i =$

$J_{i+1} =$

Binary, **associative** operator: **+A ◊ B = BA** Identity: **0**

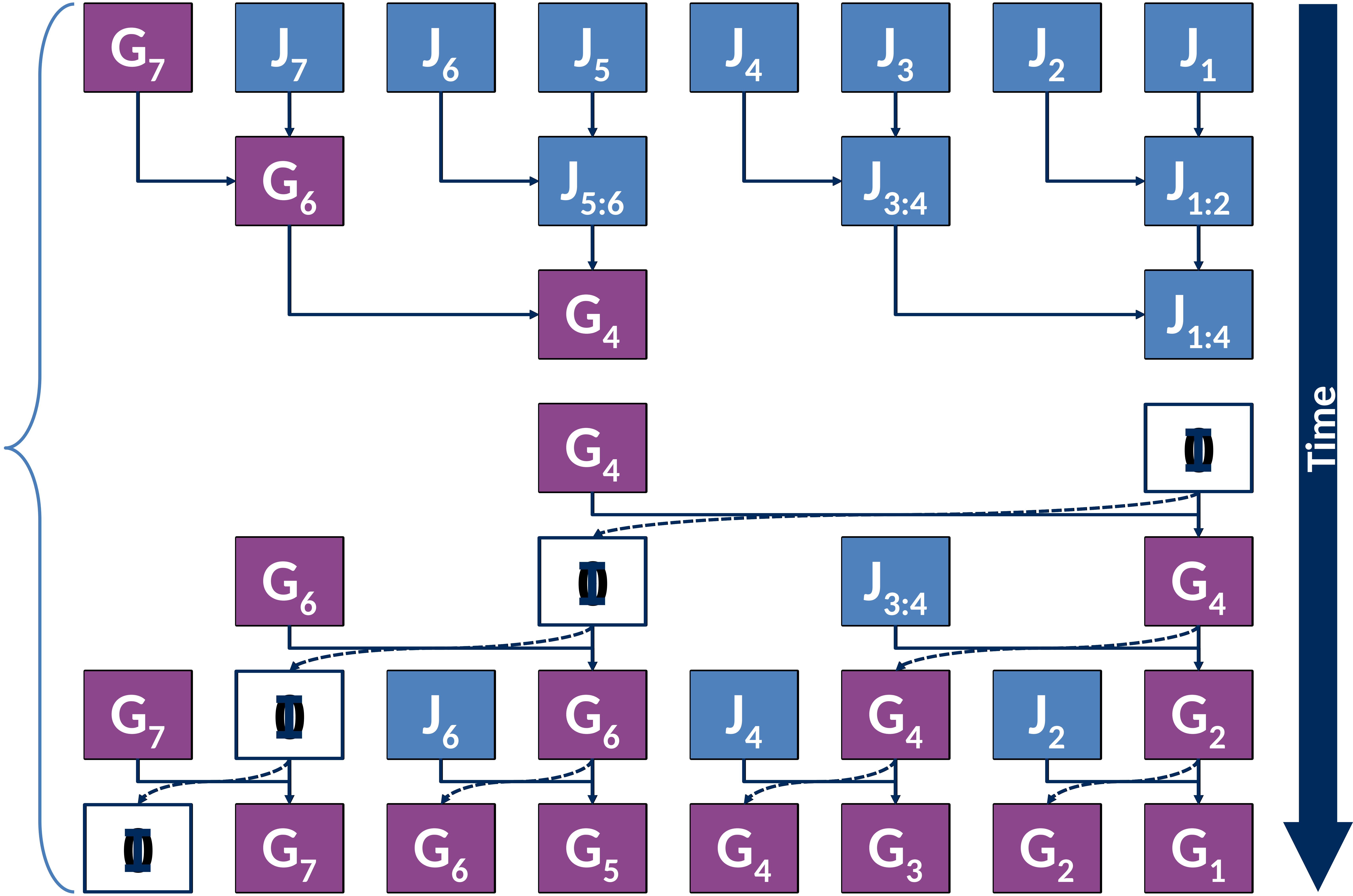
Input sequence:	1	2	3	4	5	6	7	8
Exclusive scan:	0	1	3	6	10	15	21	28

<u>Key Insight</u>: matrix multiplication in **BP** is also **binary** & **associative**!

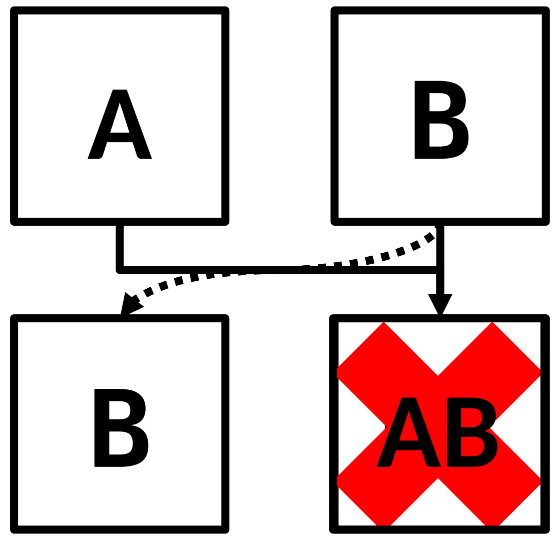
Scale BP by Blelloch Scan

Logarithmic steps along the critical path!

2logn



Down-sweep



Matrix multiplications are **noncommutative**.

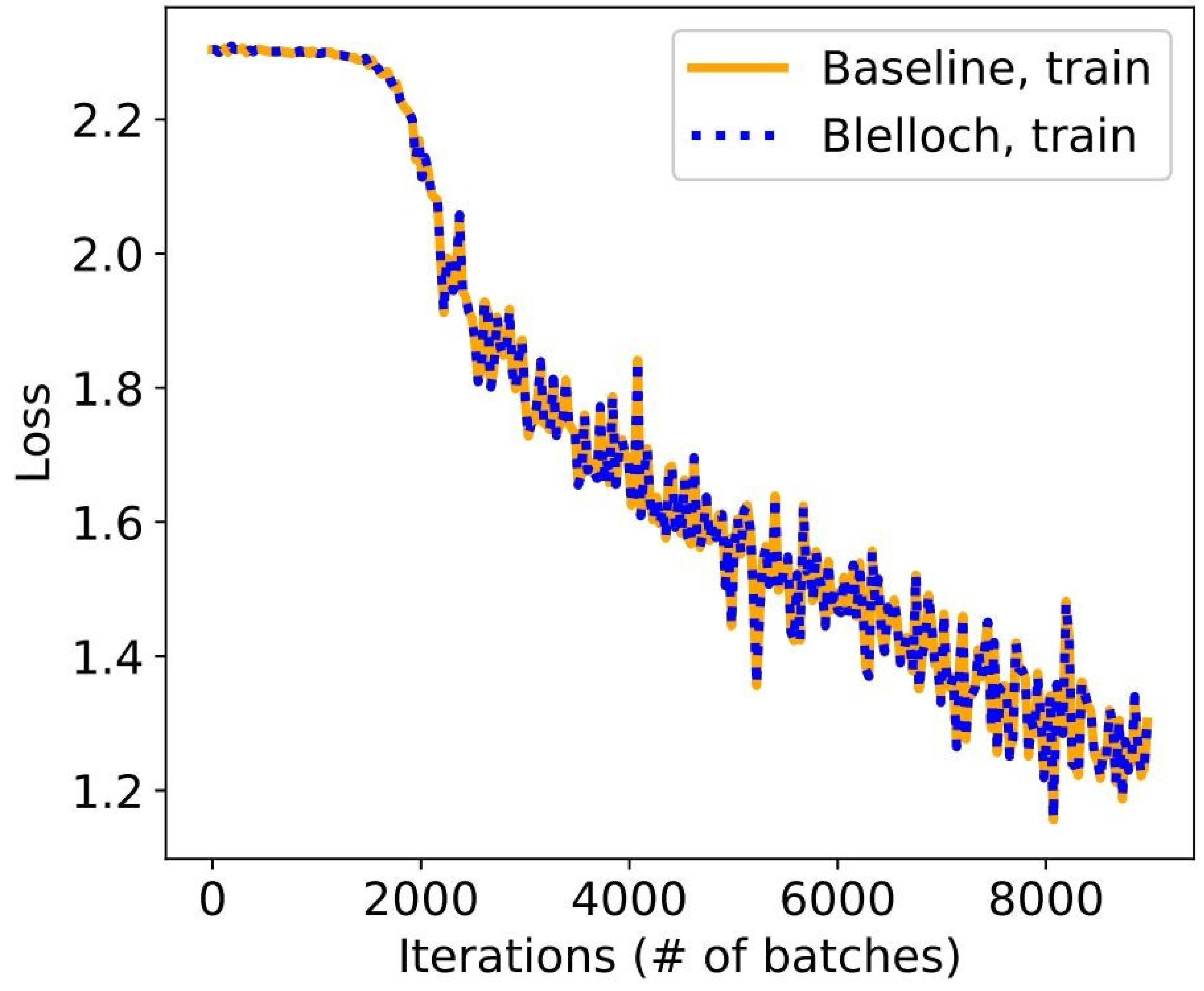
Reconstructs the Original BP Exactly

Our method produces gradients **mathematically equivalent** to BP.

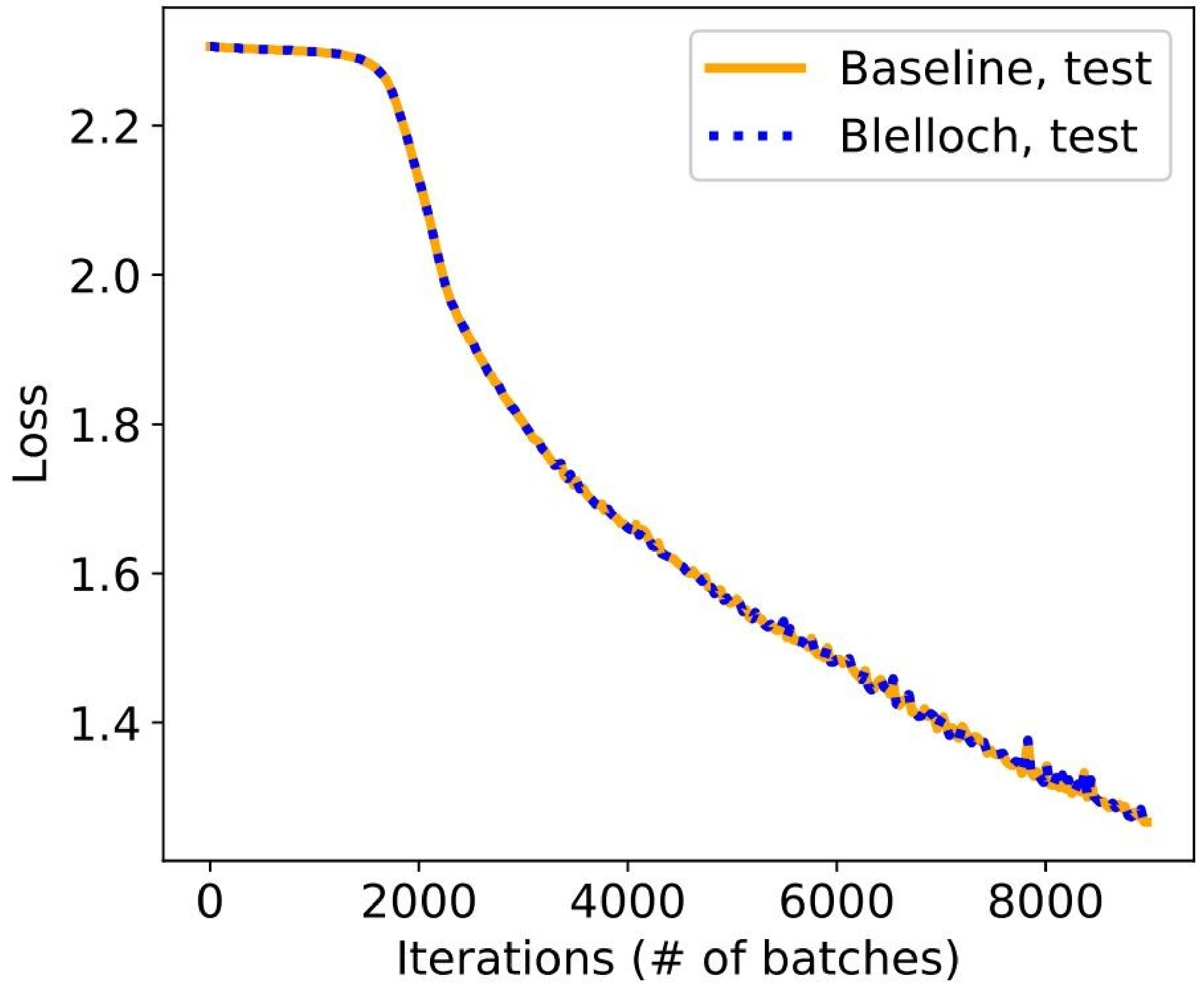
The Jacobians are multiplied in a different order → numerical differences.

Empirically show that such differences do not effect convergence.

Training LeNet-5 on CIFAR-10 (baseline: PyTorch Autograd)



(a) Training loss per iteration.



(b) Test loss per iteration.

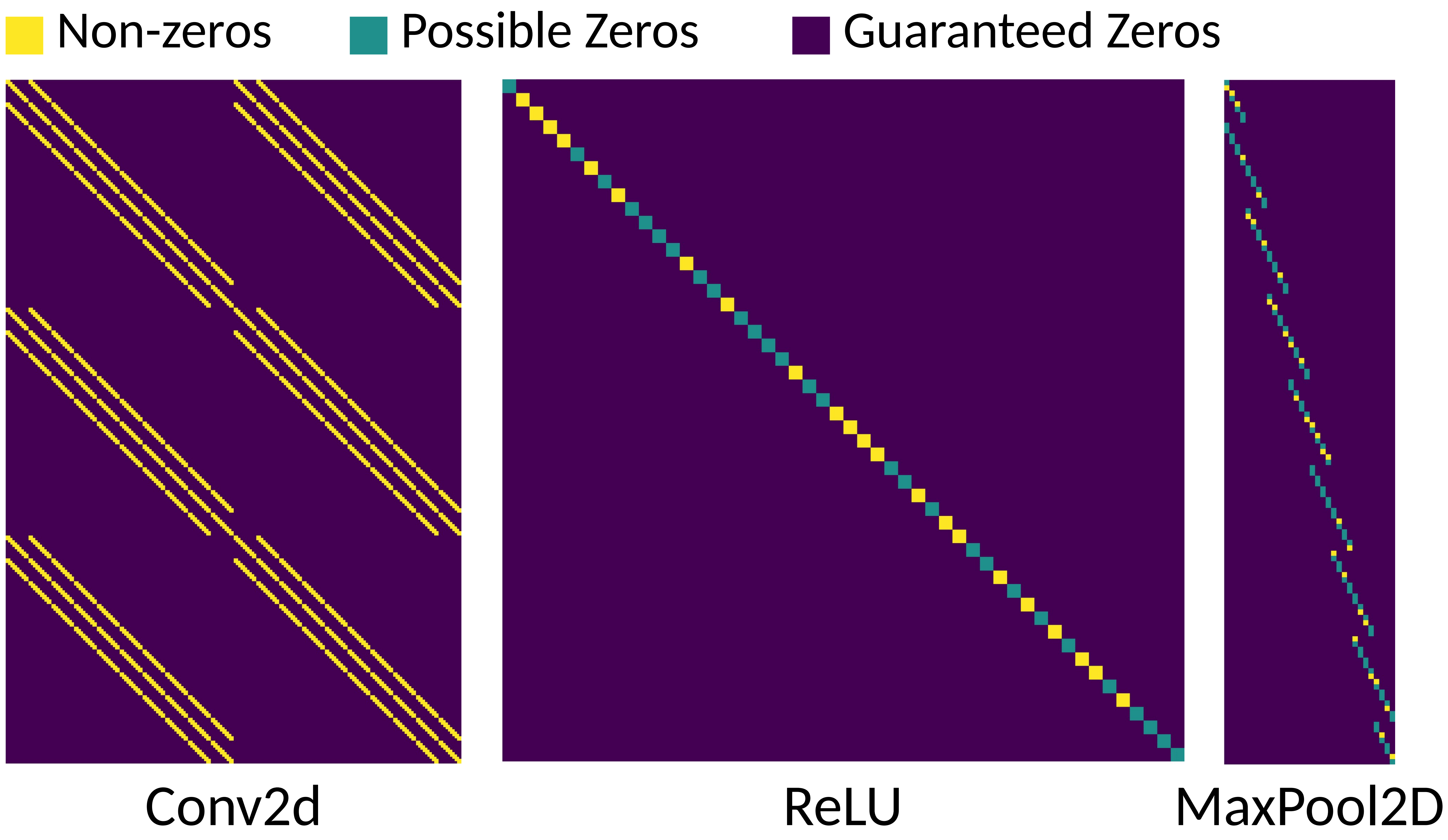
Jacobians are Memory & Compute Hungry

A full Jacobian can be prohibitively expensive to handle.

- e.g., 1st convolution in VGG-11 on CIFAR-10 images occupy **768 MB** of memory.
- Generated one row at a time by passing basis vectors into `Op_Grad()` (the VJP function).

Conventional ML algorithms avoid using Jacobians directly (including BP).

The Jacobians of Many Operators are Sparse



Guaranteed zeros:

Known **ahead of training time.**

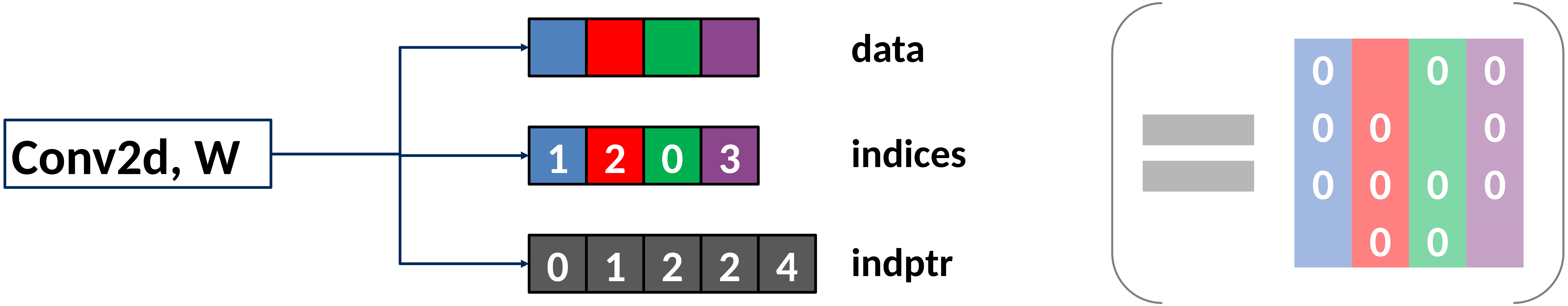
Deterministic pattern.

Potentially **better** SpGEMM performance.

First three ops of VGG-11 on CIFAR-10	Convolution	ReLU	Max Pooling
Sparsity	0.99157	0.99998	0.99994

Fast Sparse Jacobians Generation

Therefore, instead of calculating the Jacobians row-wise, generate **directly** into **Compressed Sparse Row (CSR)**:



First three ops of VGG-11 on CIFAR-10	Convolution	ReLU	Max Pooling

Complexity Analysis

Per-step Complexity (**C**): runtime of each step.

Runtime:

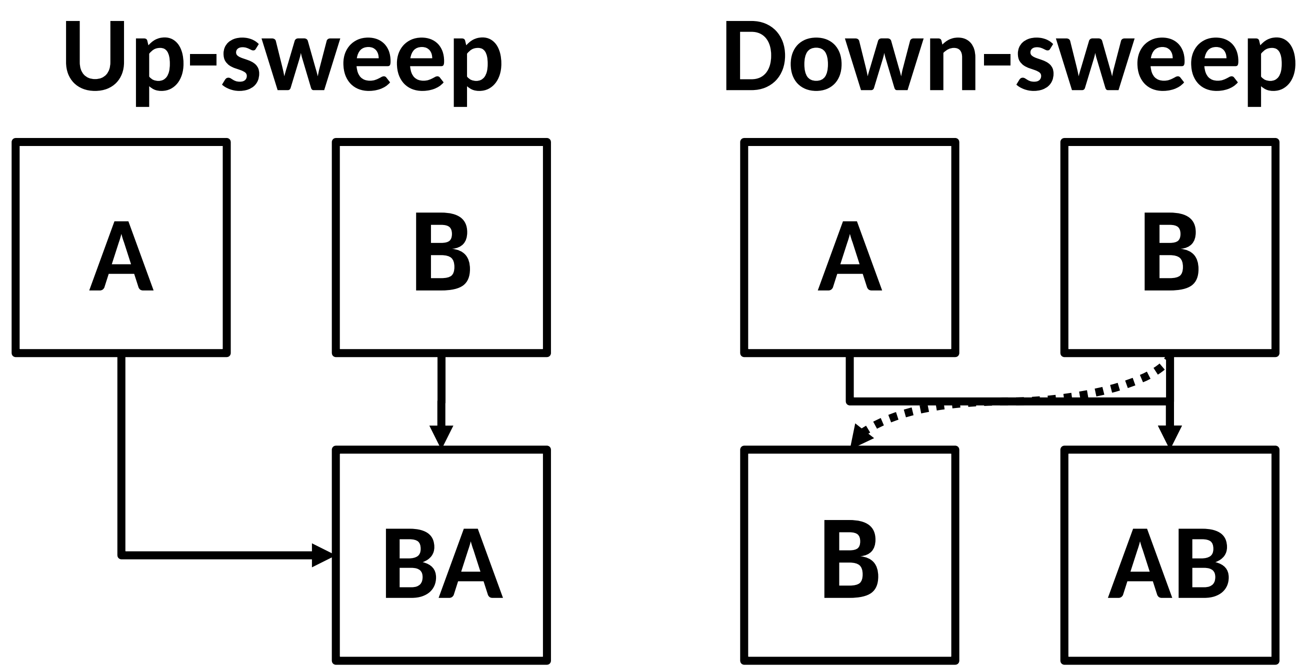
BPPSA		BP
$C_{BPPSA}\ \Theta(\log n)$	vs.	$C_{BP}\ \Theta(n)$

Performance benefits:

1. Large **n**: deep network, long sequential dependency.
2. Reducing per-step complexity: SpGEMM.

Constant per-device space complexity!

In-place

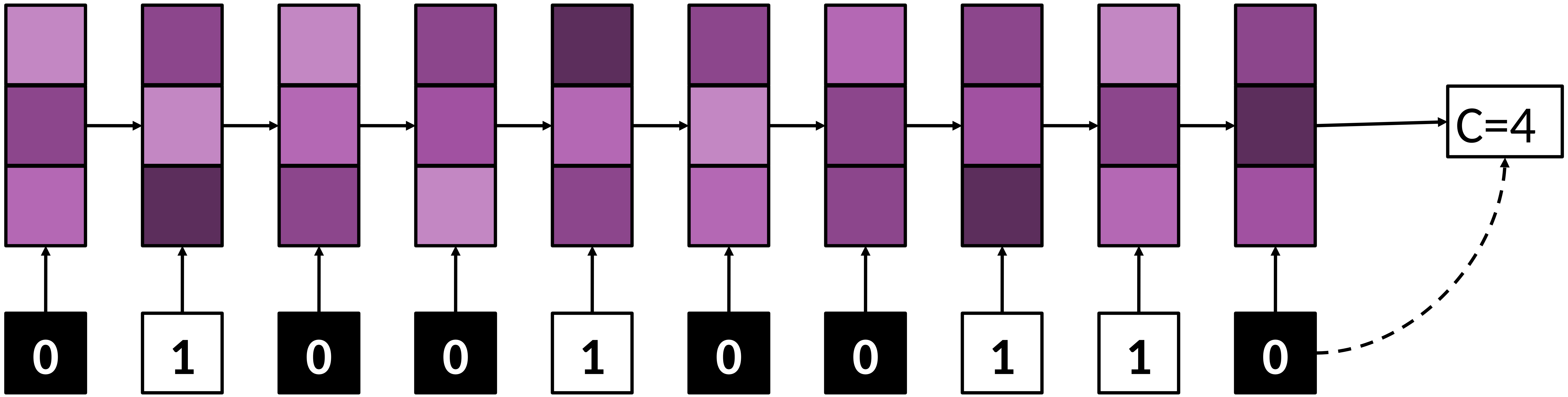


Methodology: Benchmark

<u>Model:</u> RNN

<u>Task:</u> Bitstream Classification

$$\vec{h}_t^{(k)} = \tanh\left(W_{ih} x_t^{(k)} + \vec{b}_{ih} + W_{hh} \vec{h}_{t-1}^{(k)} + \vec{b}_{hh}\right)$$



$$x_t^{(k)} \quad Bernoulli(0.05 + C¿¿(k) \times 0.1)¿$$

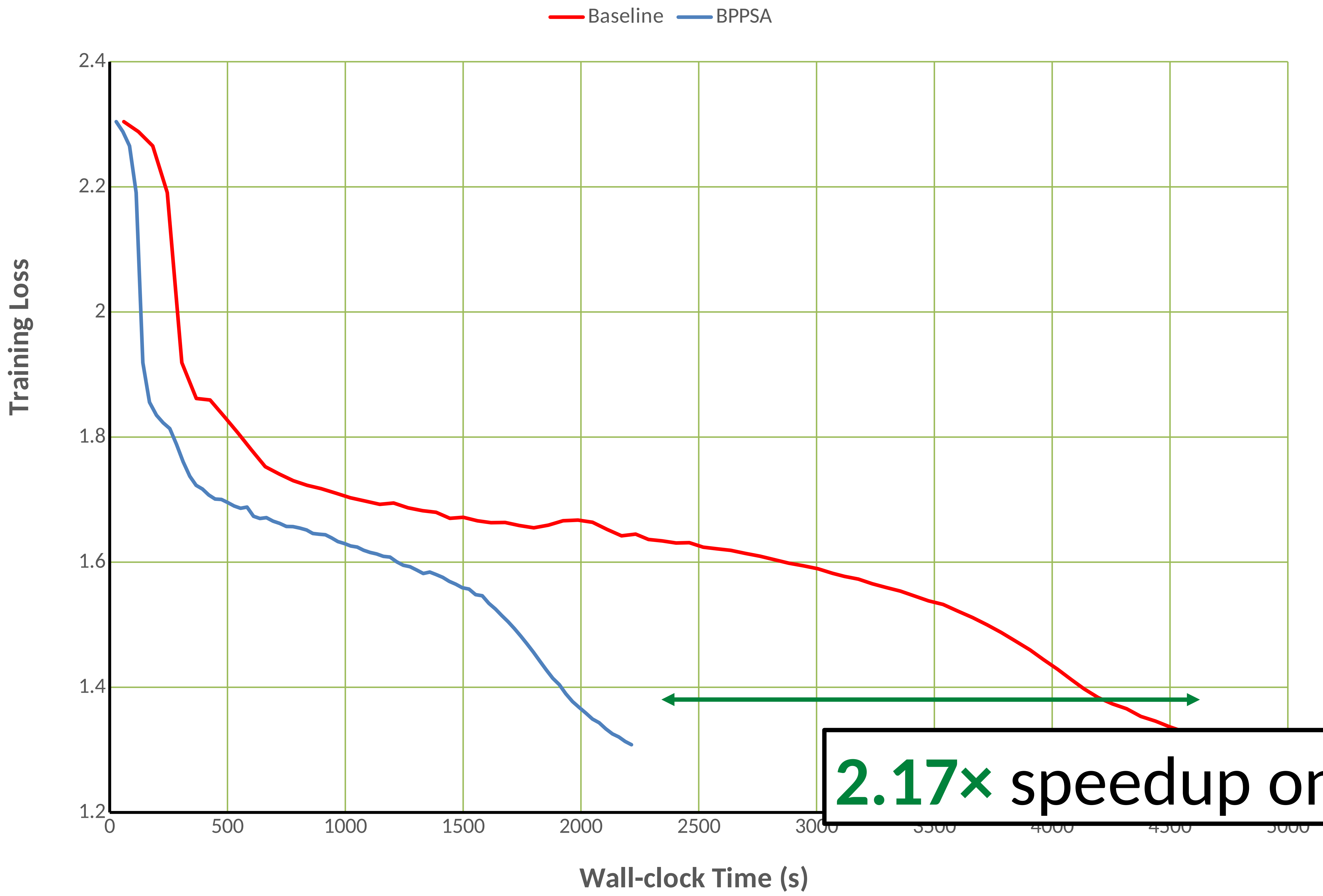
Methodology: Environment

		RTX 2070	RTX 2080 Ti
Hardware:		RTX 2070	RTX 2080 Ti
Baseline:	cuDNN	7.5.1	7.6.2
	PyTorch	1.1	1.2

Implementation: custom CUDA 10 kernels.

End-to-end Training Speedup

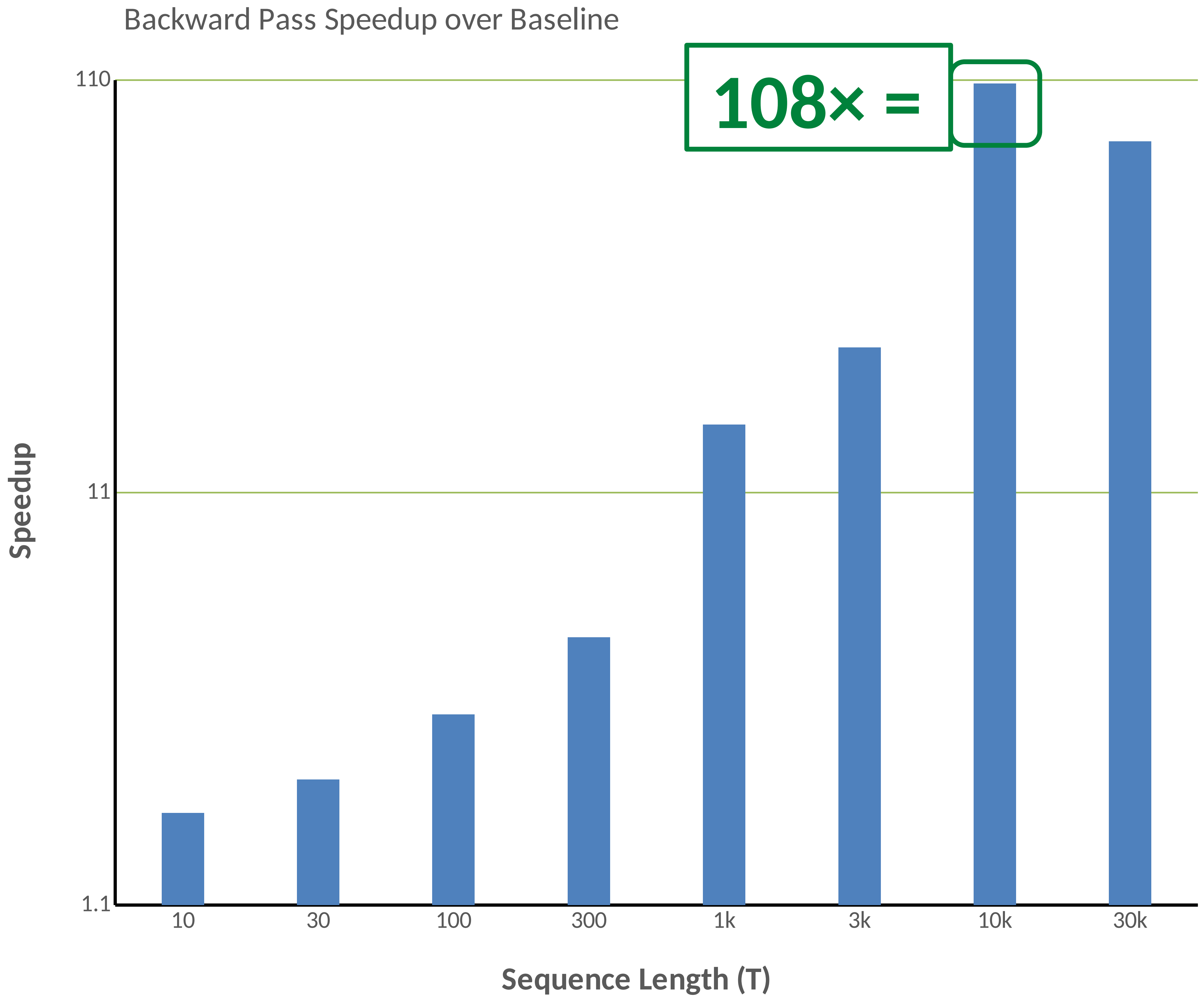
Training curve of BPPSA v.s. the baseline
when batch size **B**=16, sequence length **T**=1000:



Numerical differences do **not** effect convergence.

2.17× speedup on the overall training time.

Sensitivity Analysis: Model Length

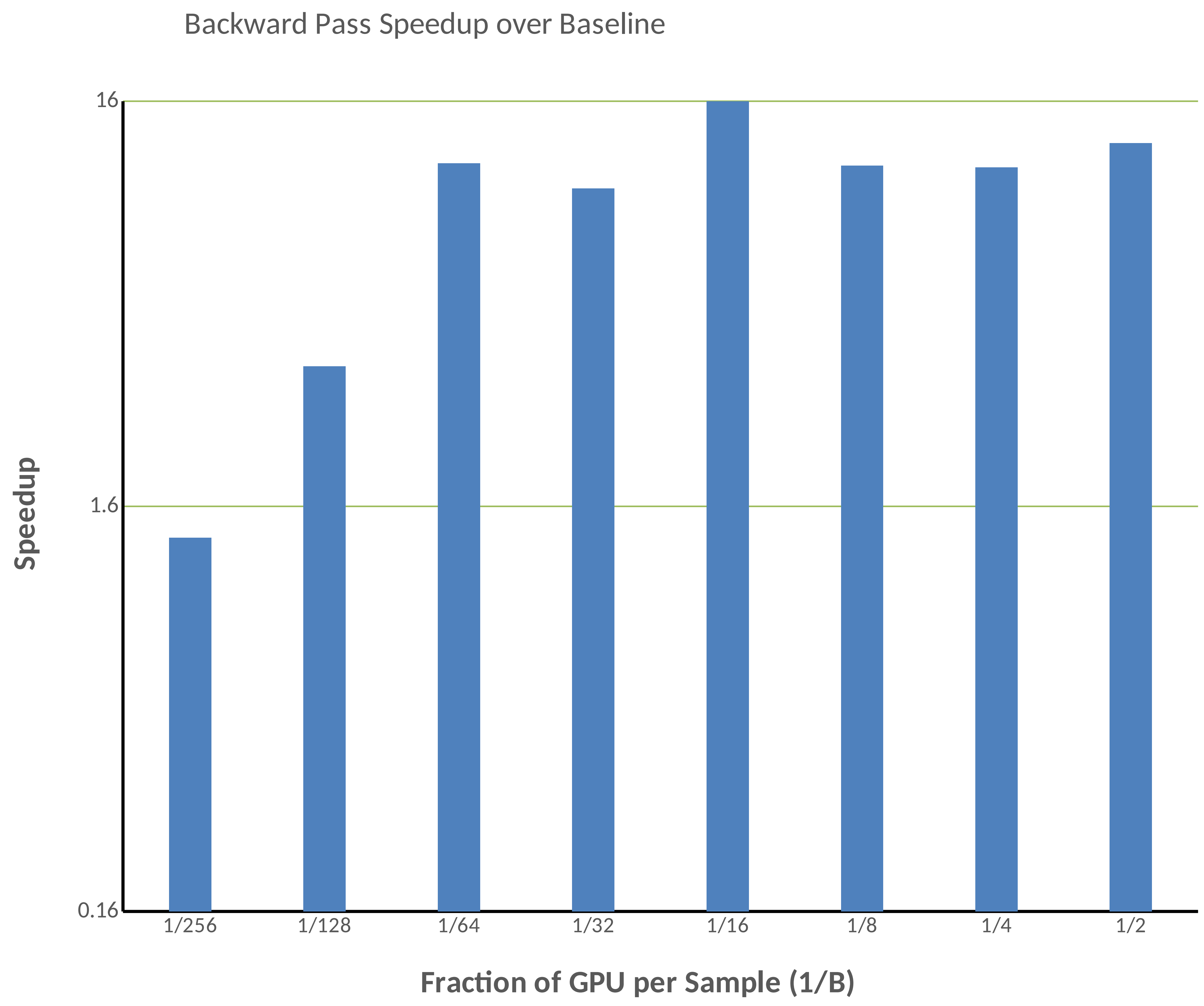


Sequence length (**T**) reflects the model length **n**.

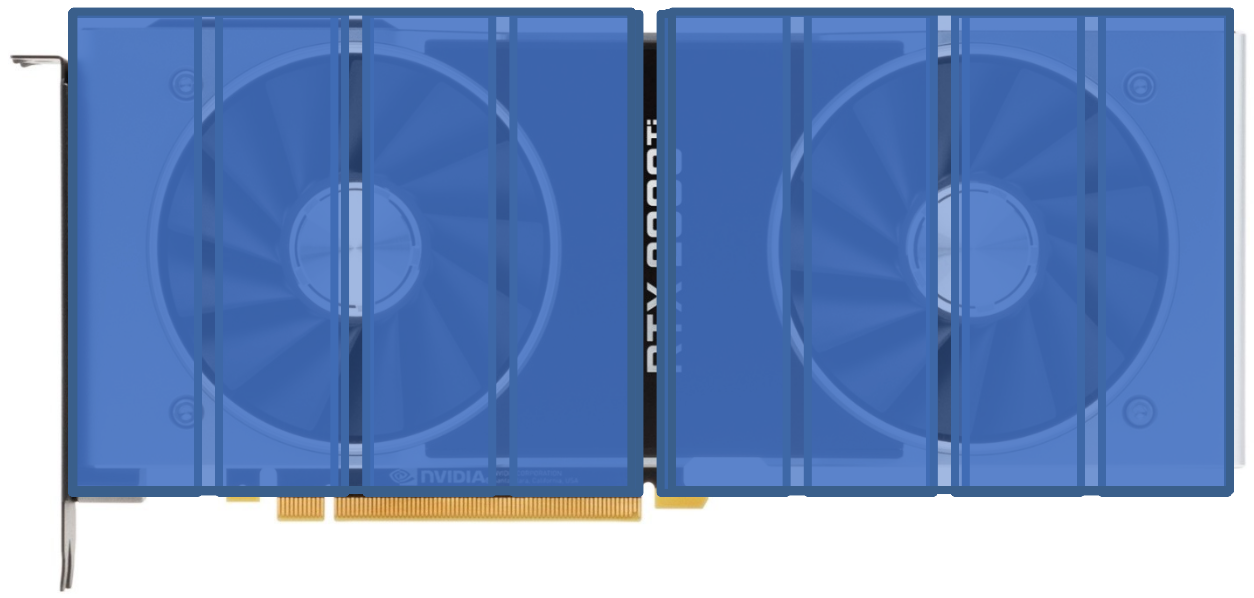
BPPSA **scales** with the model length (**n**);

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

Sensitivity Analysis: Number of Workers



Fraction of GPU per sample (**1/B**) reflects the number of workers **p**.

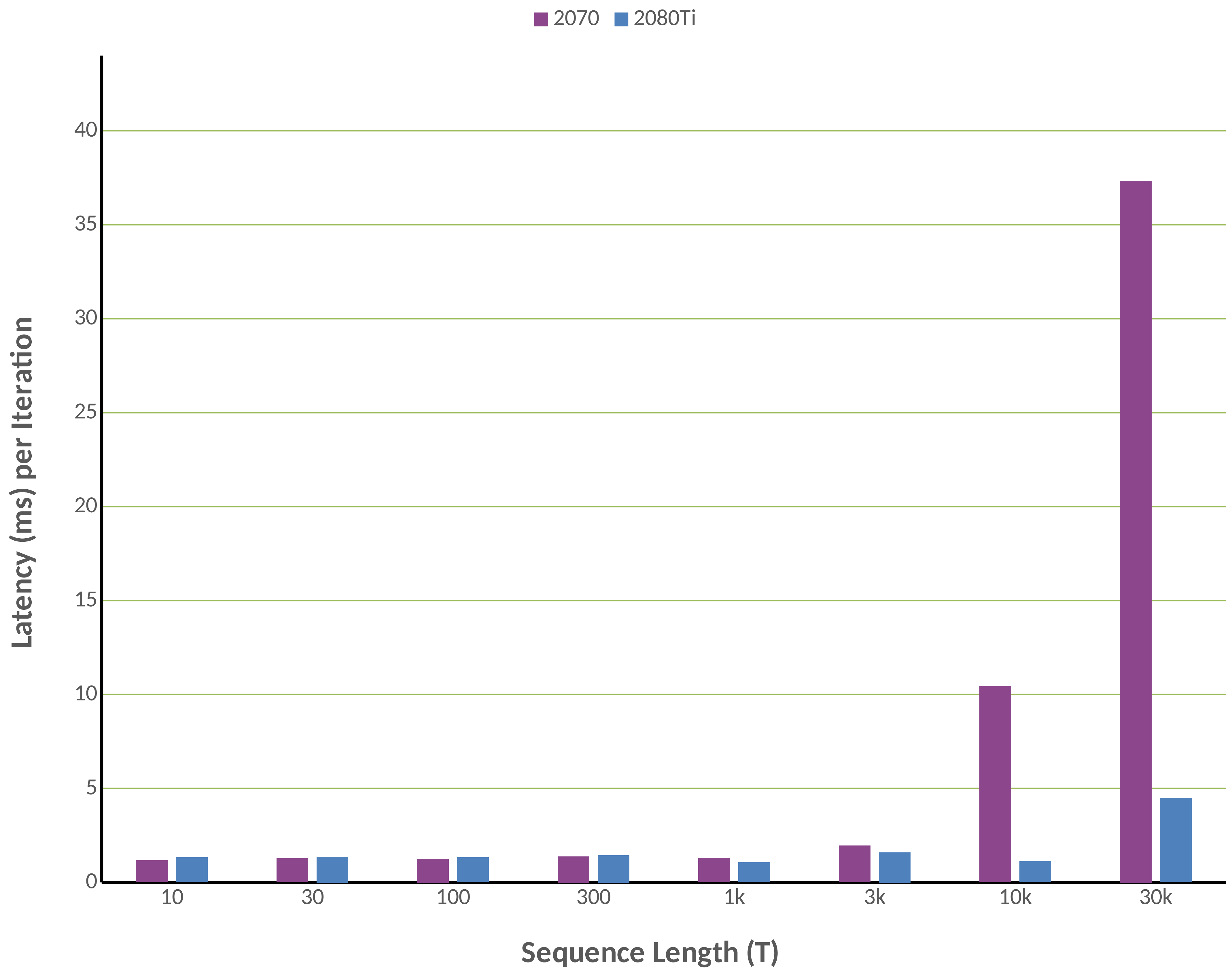


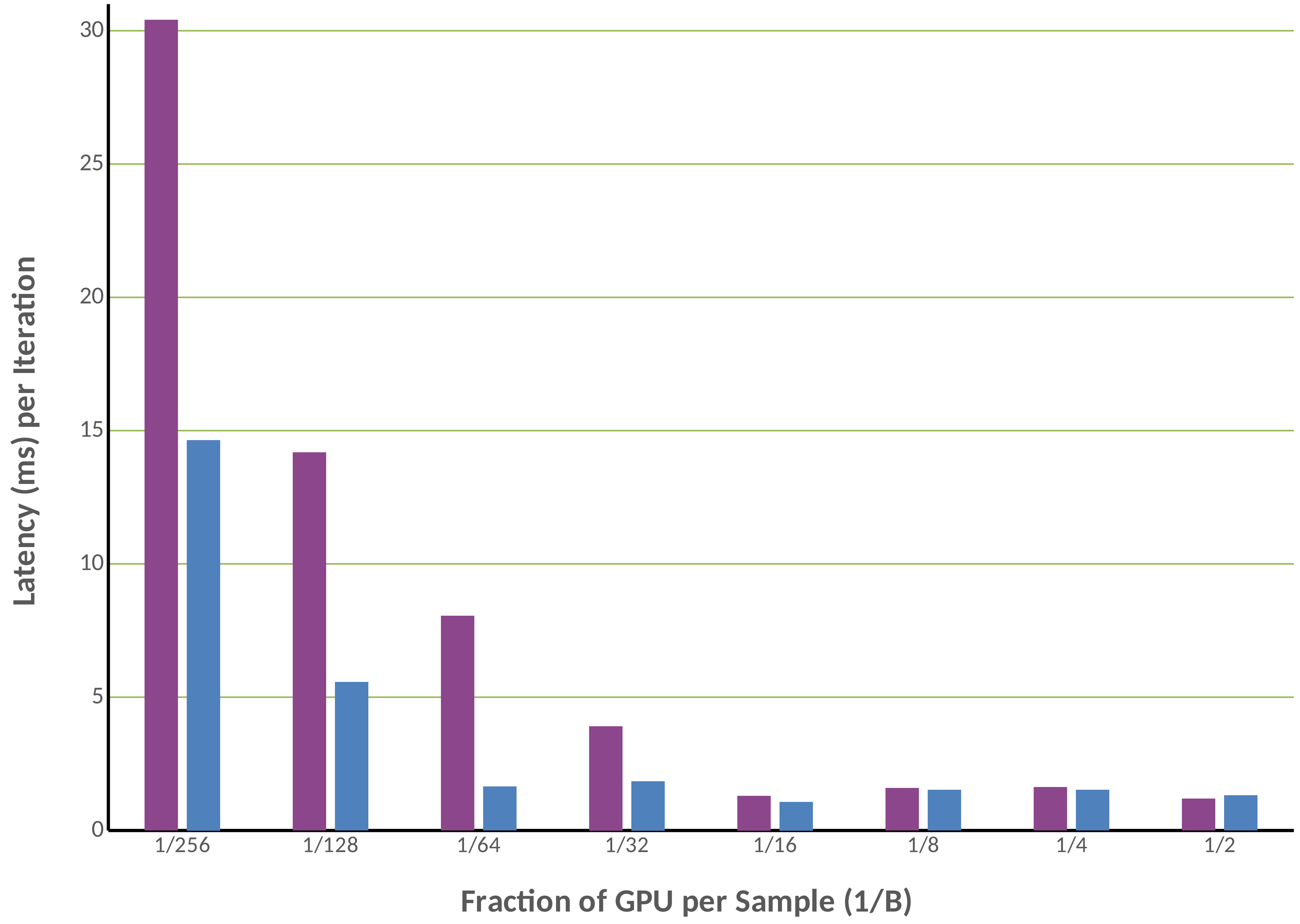
BPPSA **scales** with the number of workers (**p**).

Sensitivity Analysis: 2070 v.s. 2080Ti

#SMs(2070) < #SMs(2080Ti)
→ Latency(2070) > Latency(2080Ti)

<u>SM</u>: Streaming Multiprocessor;
i.e., “Parallel Cores”.





More Results in the Paper

- End-to-end benchmarks of GRU training on IRMAS.
  - A more realistic version of the RNN results.
- Pruned VGG-11 retraining on CIFAR-10.
  - Microbenchmark via FLOP measurements.
  - Evaluate the effectiveness of leveraging the Jacobians' sparsity in CNNs.

Conclusion

BP imposes a **strong sequential dependency** among layers during the gradient computations, limiting its scalability on parallel systems.

We propose scaling **B**ack-**P**ropagation by **P**arallel **S**can **A**lgorithm (**BPPSA**):

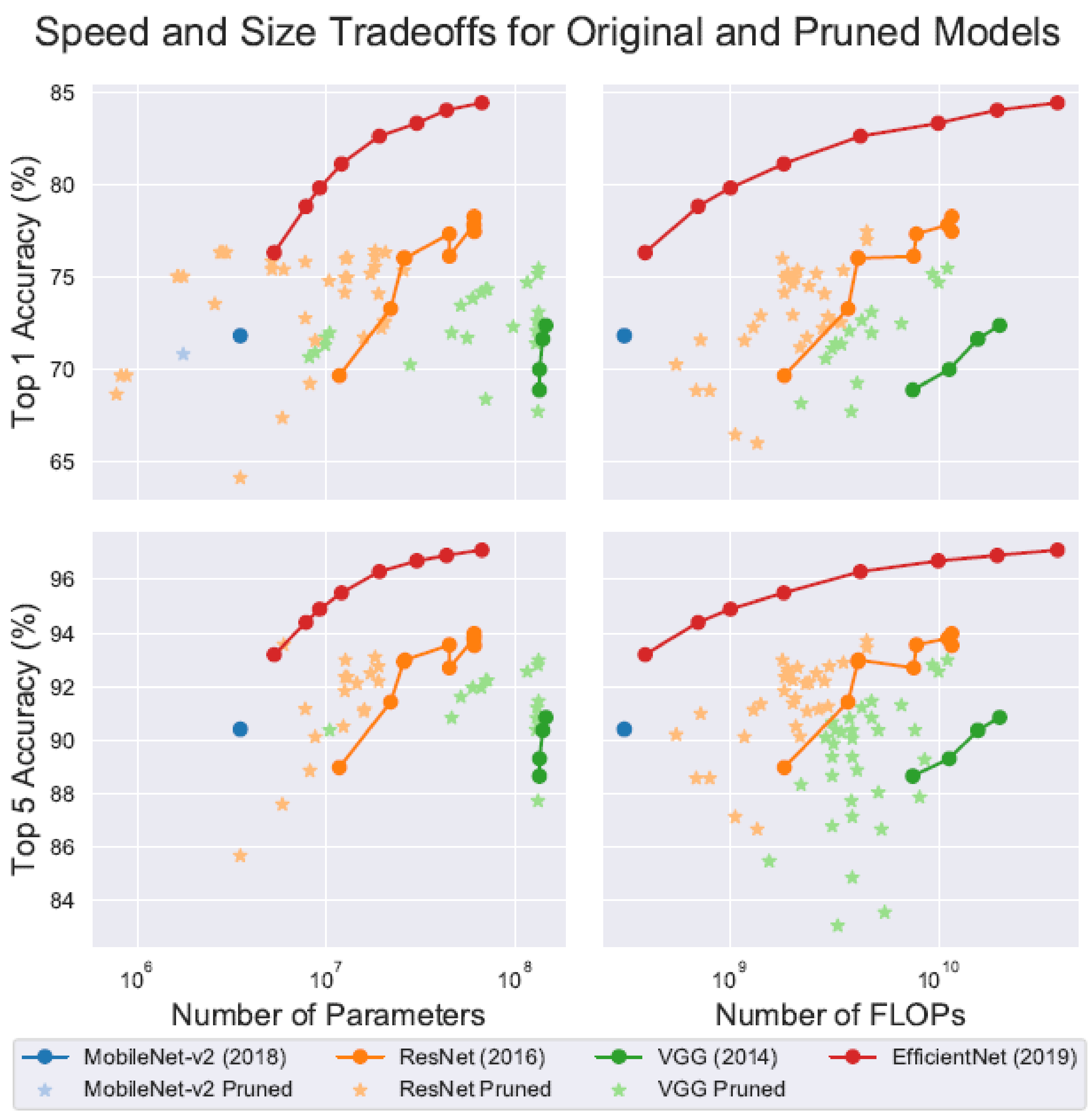
- Reformulate BP as a **scan** operation.
- Scale by a **customized Blelloch scan** algorithm.
- Leverage **sparsity** in the Jacobians.

Key Results: **Θ(log n)** vs. **Θ(n)** steps on parallel systems.

Up to **108×** speedup on the backward pass (→ **2.17×** overall speedup).

DNN Training and Inference : Trends and State-of-the-Art

3. Inference: More Solid Quantization and Pruning



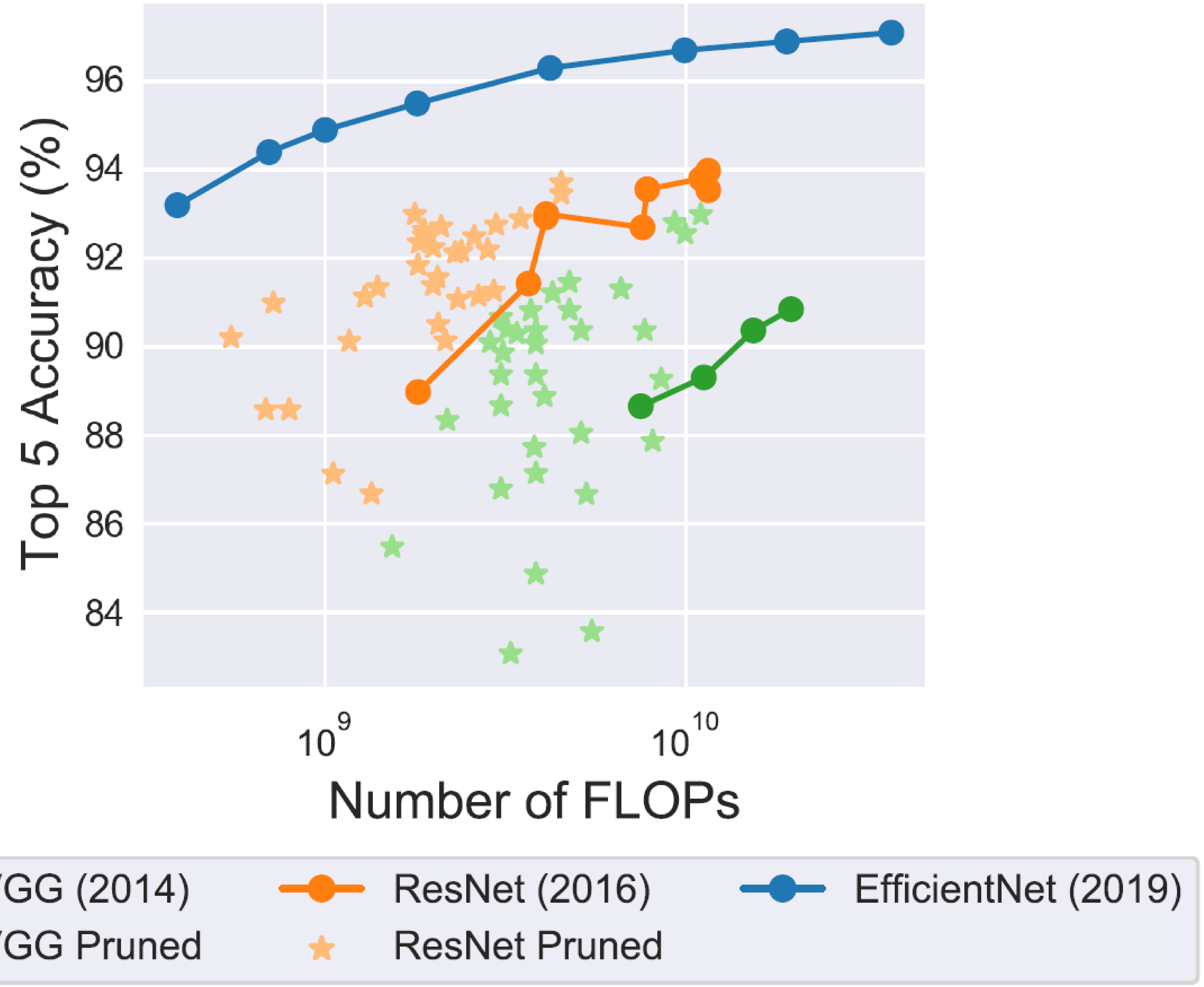
What is the State of Neural Network Pruning?

MIT

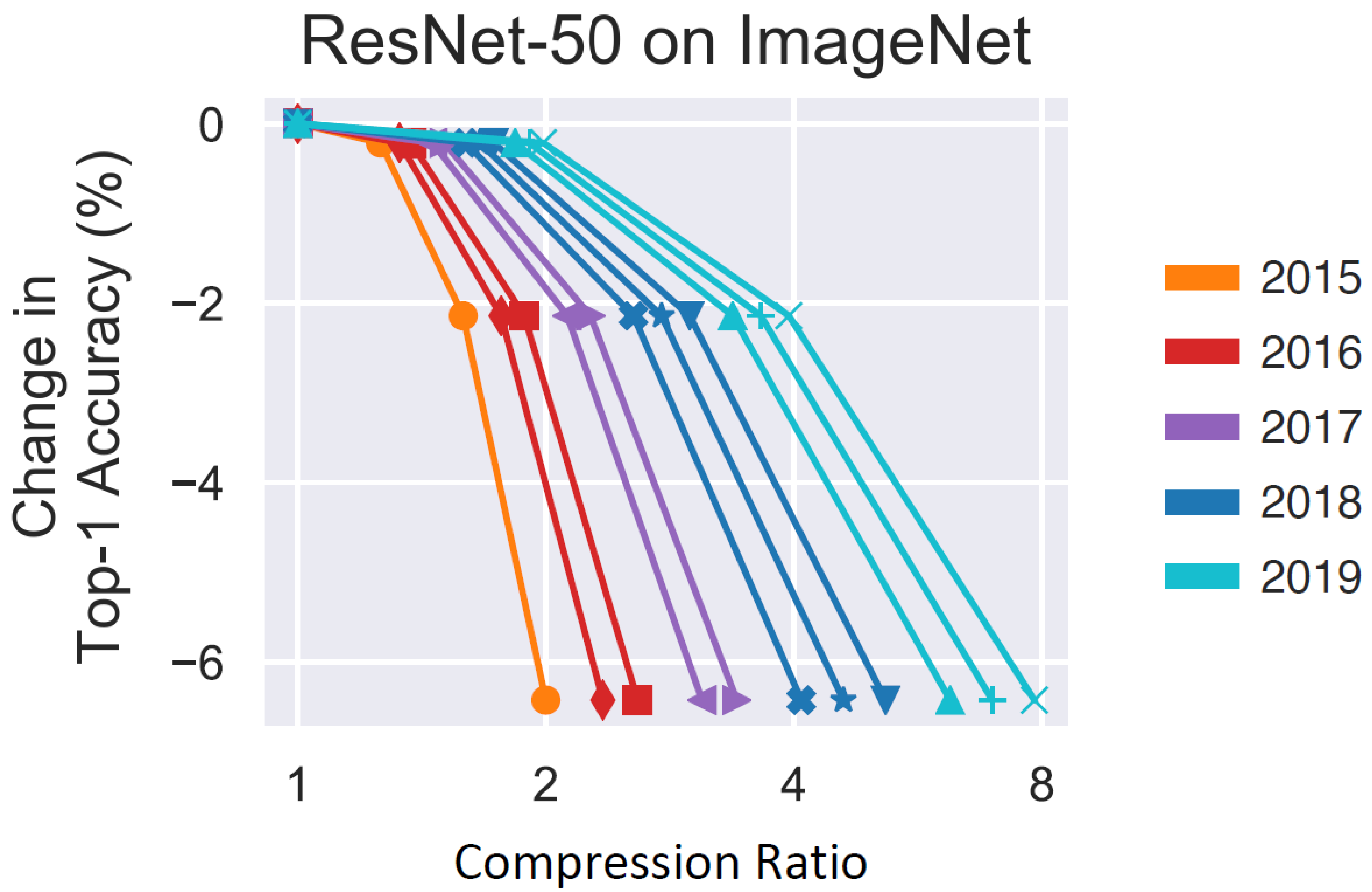
MLSys 2020

- We aggregated results across 81 pruning papers
- Mostly published in top venues
- Corpus closed under experimental comparison

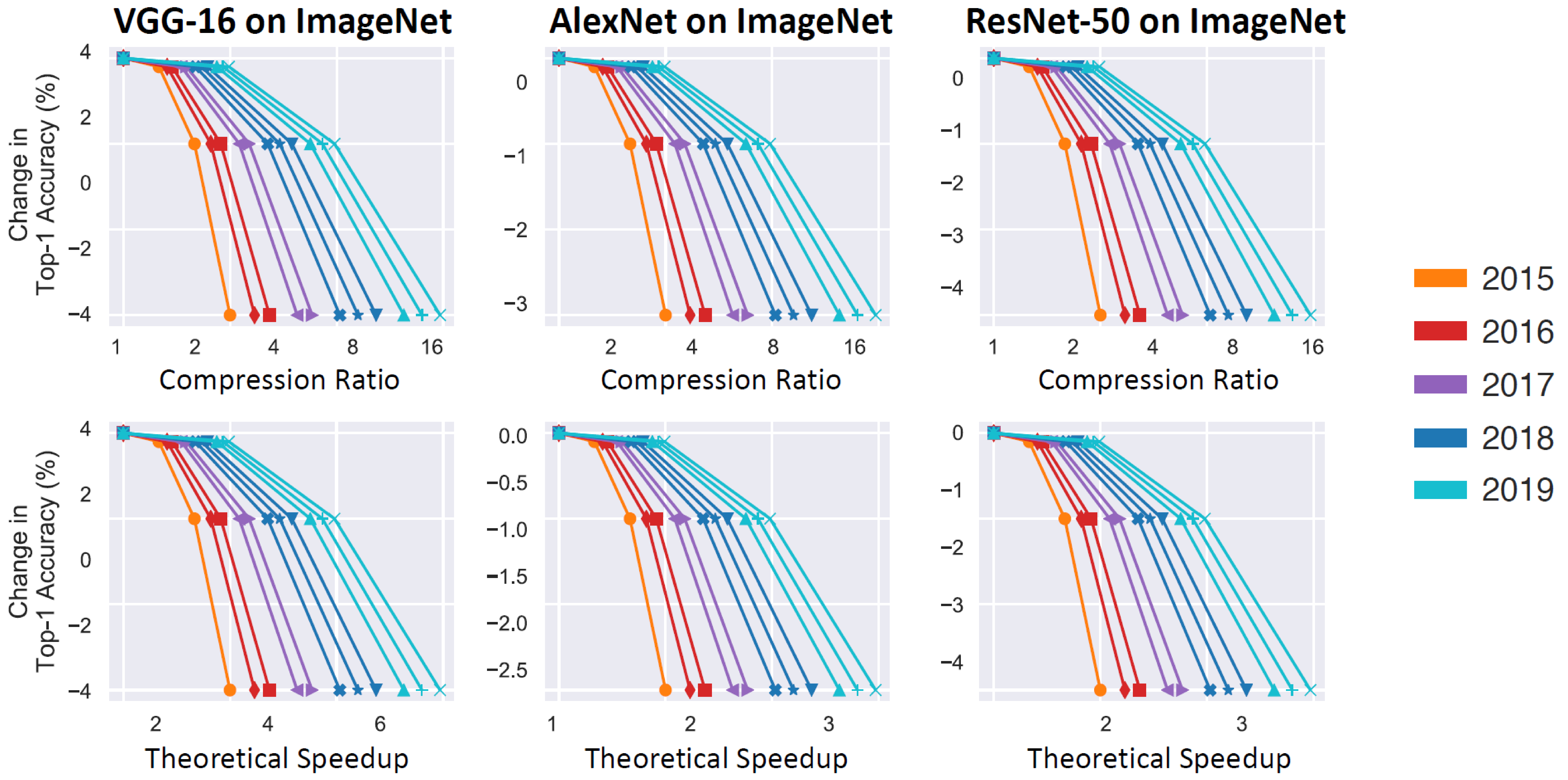
Venue	# of Papers
arXiv only	22
NeurIPS	16
ICLR	11
CVPR	9
ICML	4
ECCV	4
BMVC	3
IEEE Access	2
Other	10



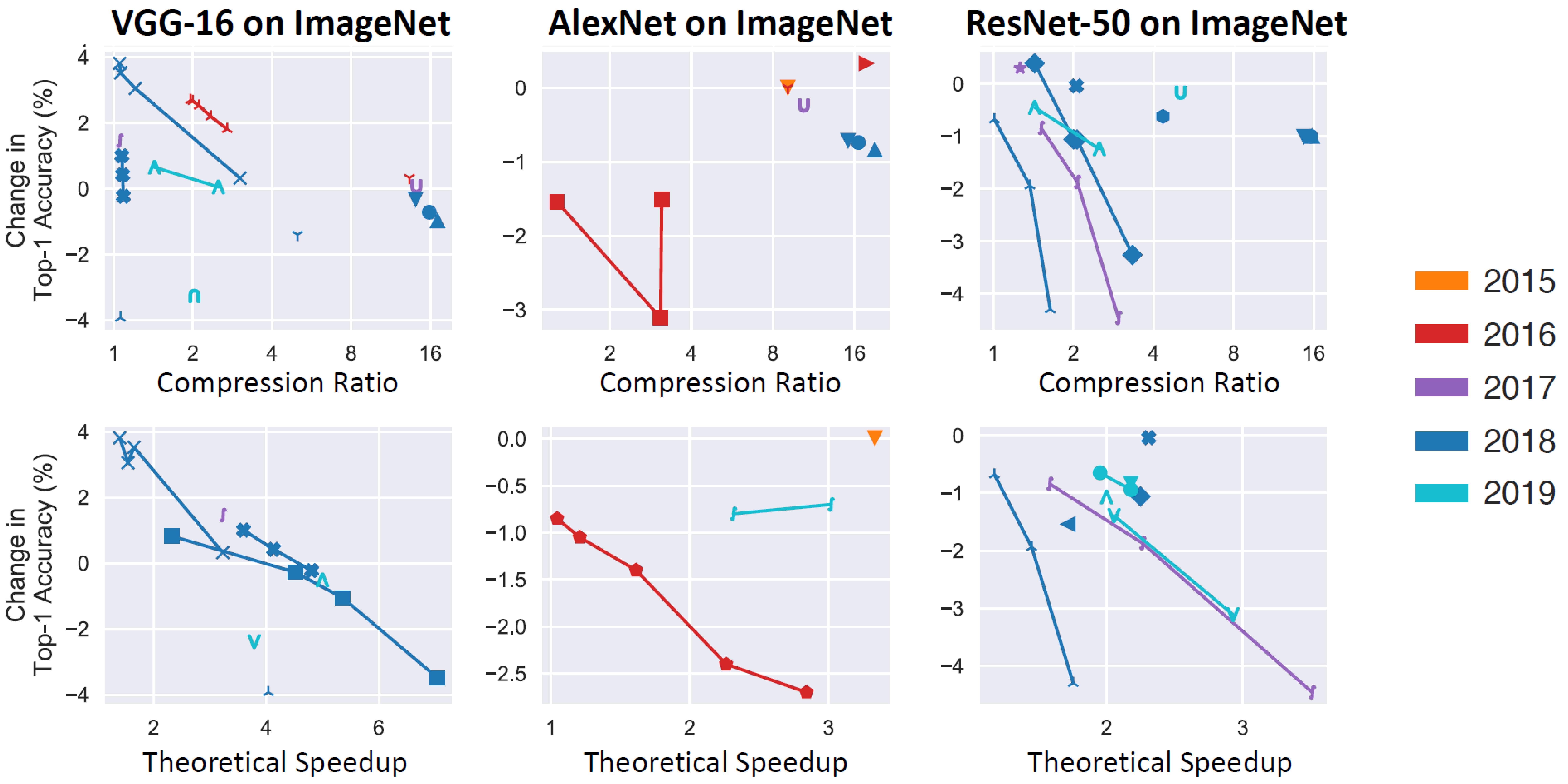
Top 5 Accuracy (%)
96
94
92
90
88
86
84
10^9
10^{10}
Number of FLOPs
VGG (2014)
ResNet (2016)
EfficientNet (2019)
VGG Pruned
ResNet Pruned



(Dataset, Architecture, X metric, Y metric, Hyperparameters) → Curve

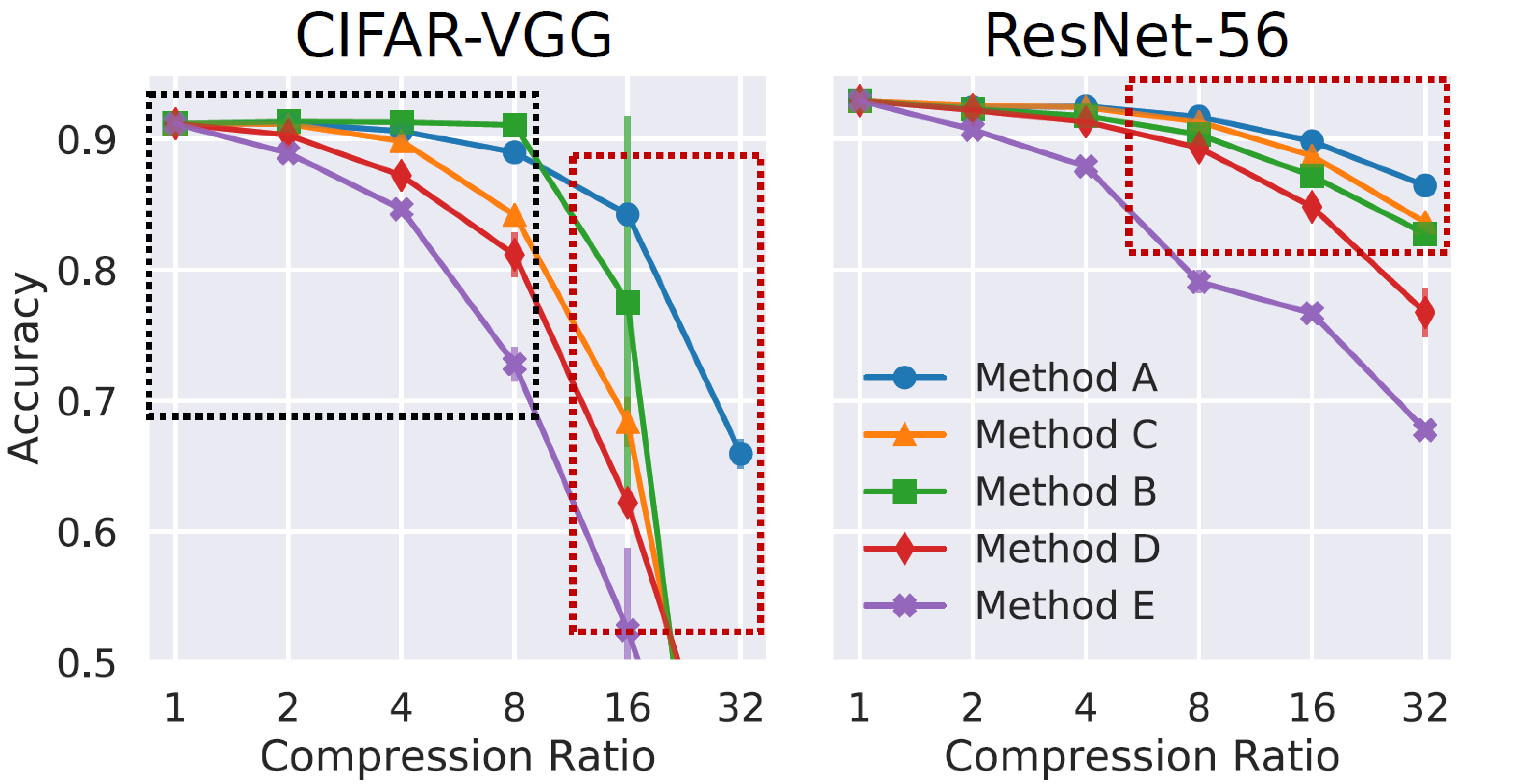


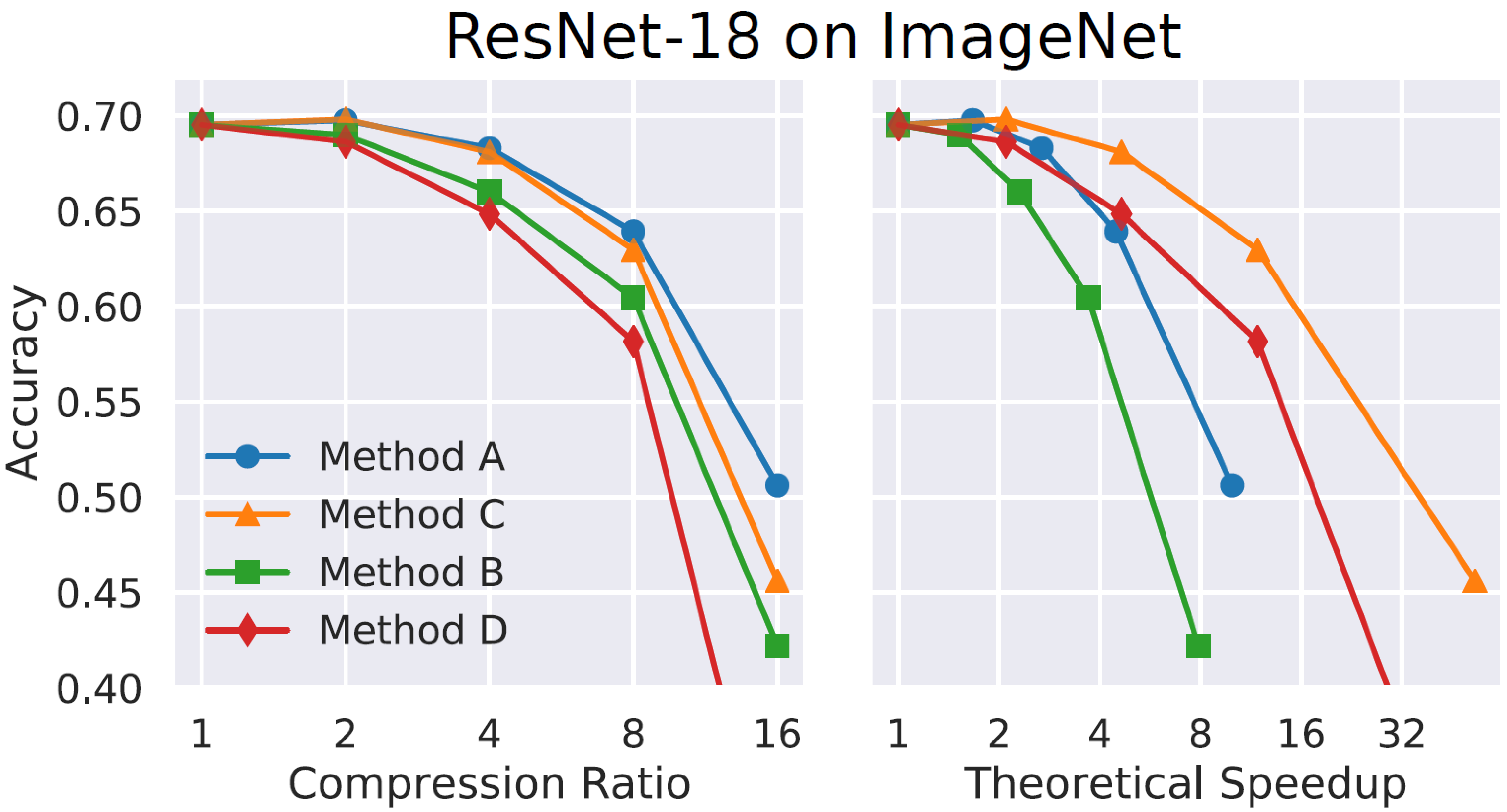
VGG-16 on ImageNet
AlexNet on ImageNet
ResNet-50 on ImageNet
Change in Top-1 Accuracy (%)
Compression Ratio
Theoretical Speedup
2015
2016
2017
2018
2019

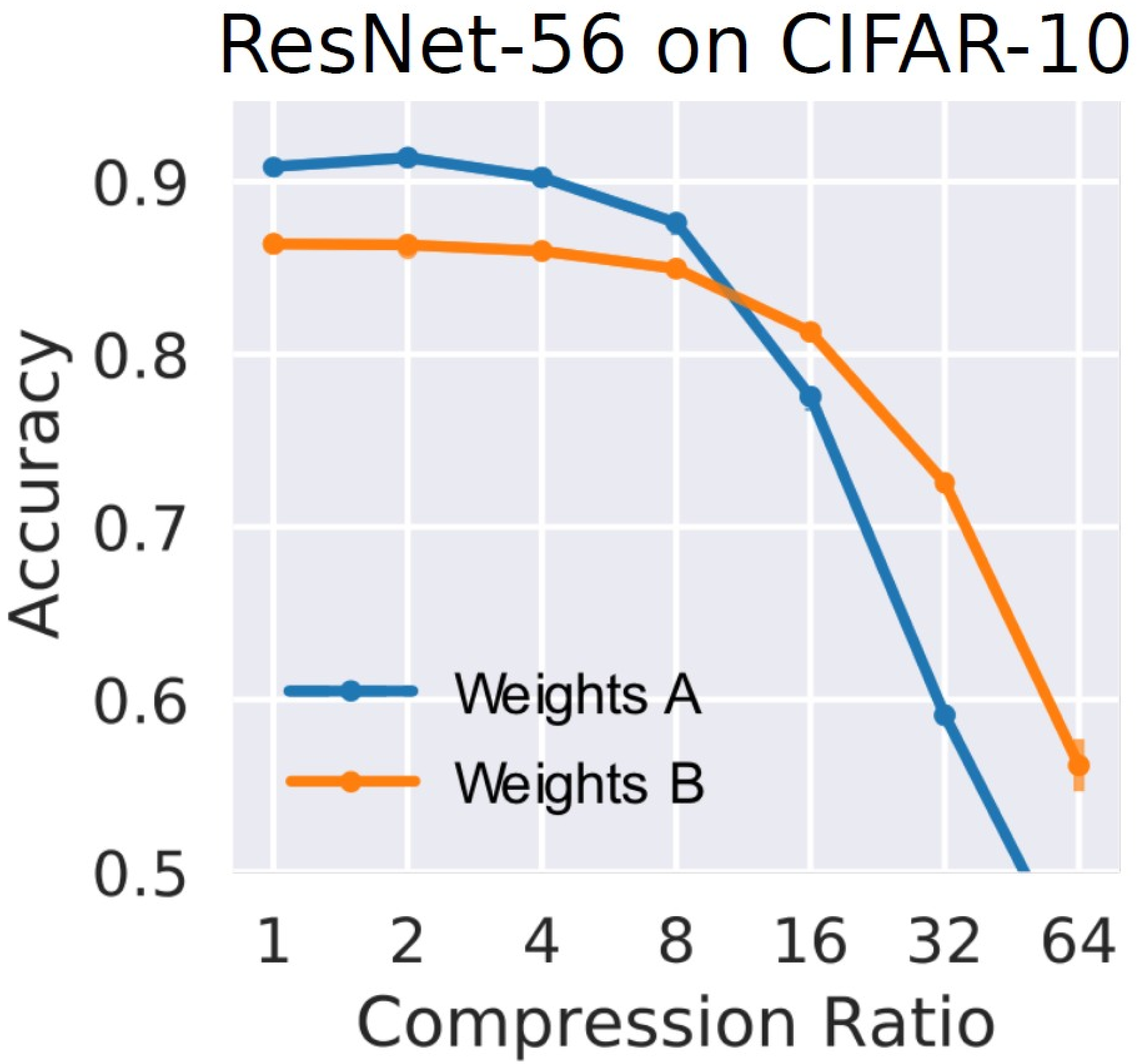


VGG-16 on ImageNet
AlexNet on ImageNet
ResNet-50 on ImageNet
Change in Top-1 Accuracy (%)
Compression Ratio
Theoretical Speedup
2015
2016
2017
2018
2019

- **Presence of comparisons:**
  - Most papers compare to at most 1 other method
  - 40% papers have never been compared to
  - Pre-2010s methods almost completely ignored
- **Reinventing the wheel:**
  - Magnitude-based pruning: *Janowsky (1989)*
  - Gradient times magnitude: *Mozer & Smolensky (1989)*
  - “Reviving” pruned weights: *Tresp et al. (1997)*


CIFAR-VGG
ResNet-56
Accuracy
0.9
0.8
0.7
0.6
0.5
1
2
4
8
16
32
Compression Ratio
Method A
Method C
Method B
Method D
Method E


ResNet-18 on ImageNet
Accuracy
0.70
0.65
0.60
0.55
0.50
0.45
0.40
Method A
Method C
Method B
Method D
1
2
4
8
16
Compression Ratio
1
2
4
8
16
32
Theoretical Speedup


ResNet-56 on CIFAR-10
Accuracy
0.9
0.8
0.7
0.6
0.5
1
2
4
8
16
32
64
Compression Ratio
Weights A
Weights B

Memory-Driven Mixed Low Precision Quantization for Enabling Deep Network Inference on Microcontrollers

Universita’ di Bologna, Bologna, Italy

MLSys 2020

DNN Training and Inference : Trends and State-of-the-Art

4. ML Compilers

Existing Efforts: Pros and cons

- TVM, XLA, Glow, PlaidML
  - Don’t perform well for training
  - TVM can be 2-3 orders of magnitude worse on important kernels

- We need a new ML compiler with representative IR:
  - Recent papers from our group: DietCode (MLSys’22), Roller (OSDI’22), Hidet (ASPLOS’23)

CSC 2224: Parallel Computer Architecture and Programming
DNN Training and Inference : Challenges, Trends, State-of-the-Art

Prof. Gennady Pekhimenko

University of Toronto

Fall 2022