deepspeed

The library to accelerate training and inference of DNN at scale

Talk @ CMU – 4/18/22

Presented by Minjia Zhang, Principal Researcher @Microsoft
(on behalf of the DeepSpeed team)

Model Scale
- 10+ Trillion parameters

Speed
- Fast & scalable training

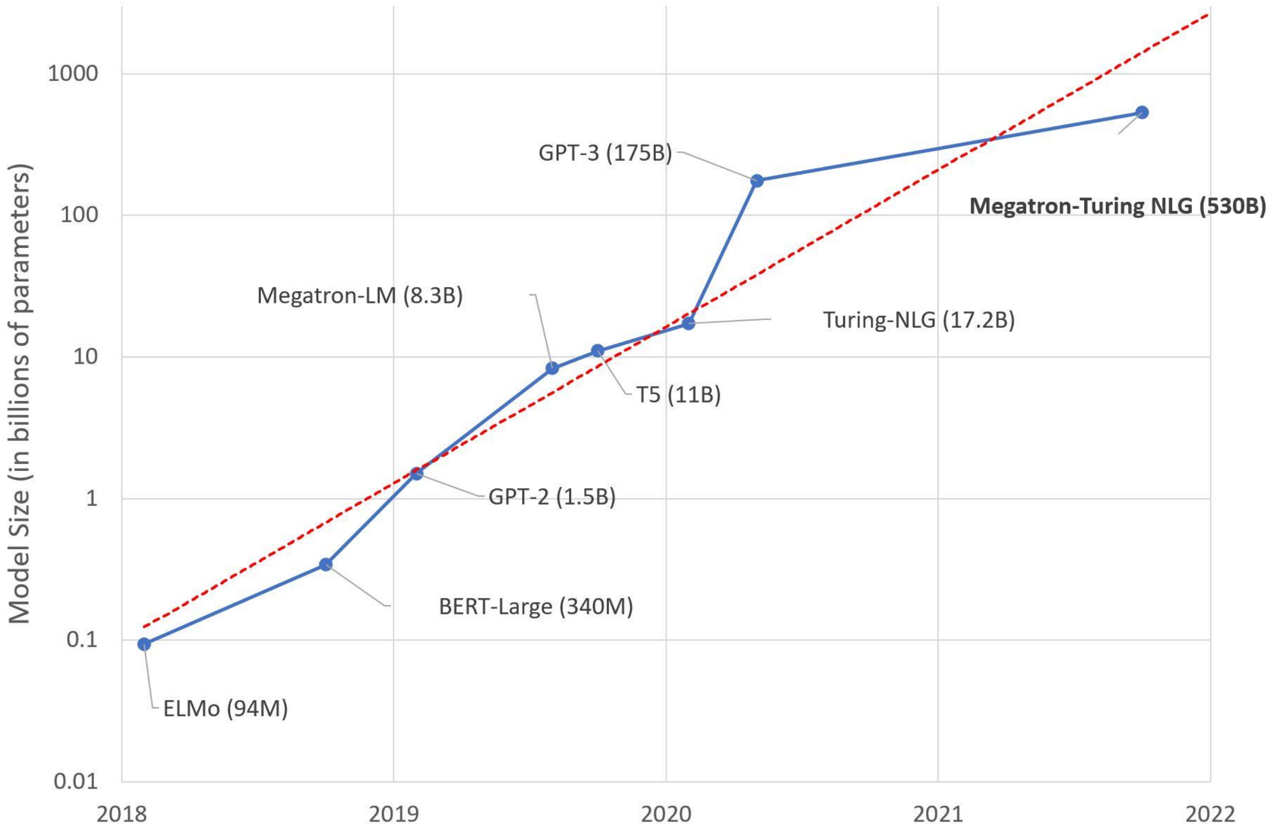
Democratize AI
- Bigger & faster for all

Compressed Training
- Boosted efficiency

Accelerated inference
- Up to 10x faster & cheaper

Usability
- Few lines of code changes

Evolution of DNN Models



Larger models → better accuracy

Model size is still growing

Not reached the accuracy limit yet

More compute-efficient to train larger models than smaller ones to same accuracy

What is a Language Model?

Text Summarization

COPYWRITING | SUMMARIZATION | PARSING UNSTRUCTURED TEXT | CLASSIFICATION | TRANSLATION

Summarize this email into a single sentence:

Dear Olivia,

The solar energy conference went great. New Horizon Manufacturing wants to meet with us to talk about our photovoltaic window system we're about to launch.
I think it would be great to talk this Tuesday.

Best,
Allison

Allison is excited to meet with New Horizon Manufacturing to discuss their photovoltaic window system.

Question and Answering

COPYWRITING SUMMARIZATION PARSING UNSTRUCTURED TEXT CLASSIFICATION TRANSLATION

Answer the following questions about this customer email:

I’m wondering if you could provide me with information about your cloud services for companies. I’m CTO for Moon Door, a movie production company and want to know if it can help us manage our different projects that are in production.

Questions:
1. What industry is the company
2. What is the name of the company
3. What are they interested in
4. What is the customer’s position

Answers:

```
1. The company is in the movie production industry.
2. The company is Moon Door.
3. The customer is interested in cloud services for companies.
4. The customer is the CTO.
```

Code Continuation and Generation

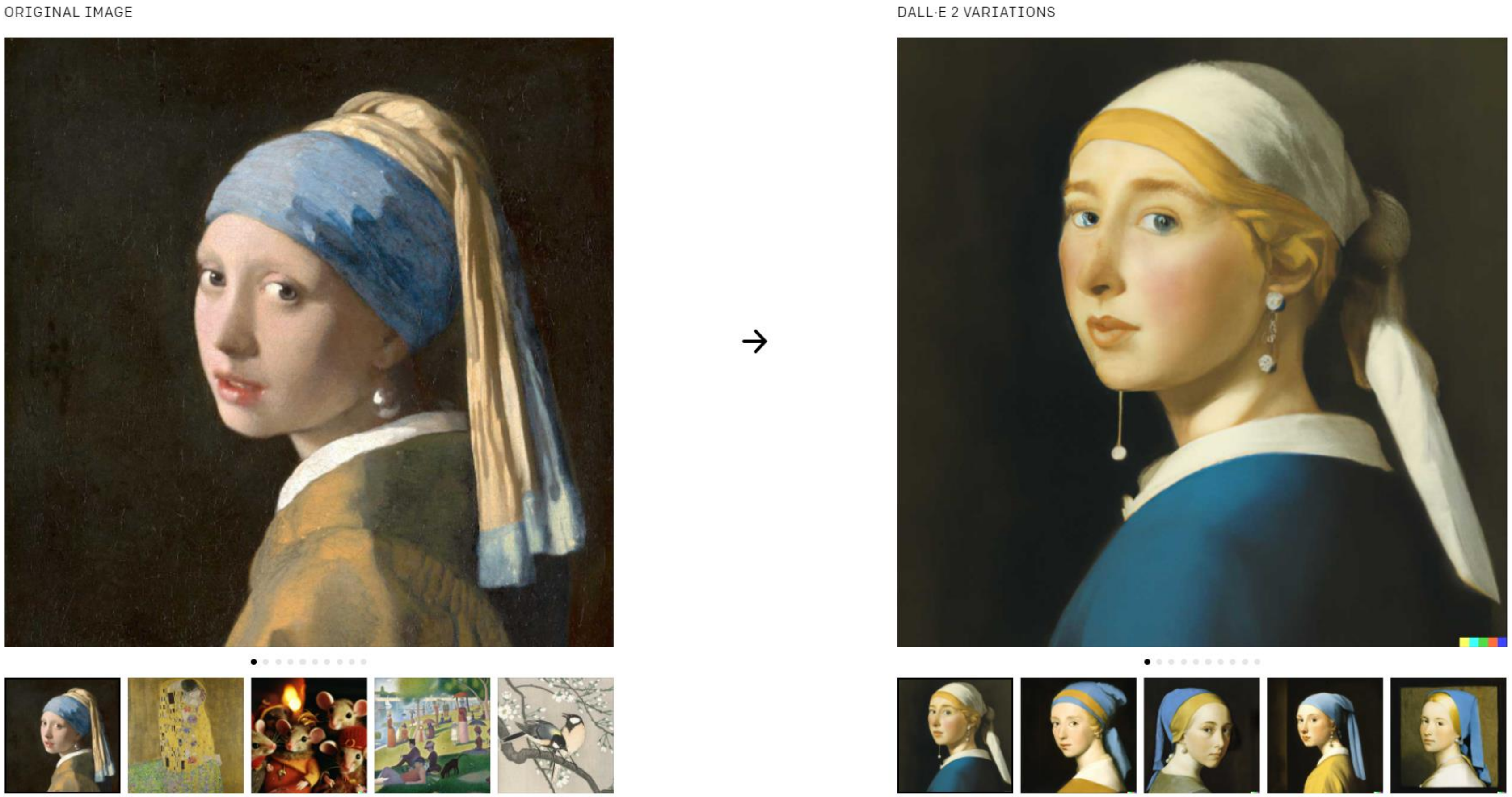
TEXT TO SQL TRANSLATION | CALLING AN API VIA NATURAL LANGUAGE | CODE CONTINUATION

```
"""
Python 3
Get the current value of a Bitcoin in US dollars using the bitcoincharts api
"""

import requests
import json

def get_bitcoin_price():
    url = 'http://api.bitcoincharts.com/v1/weighted_prices.json'
    response = requests.get(url)
    data = json.loads(response.text)
    return data['USD']['7d']

if __name__ == '__main__':
    print(get_bitcoin_price())
```

Many Other Examples

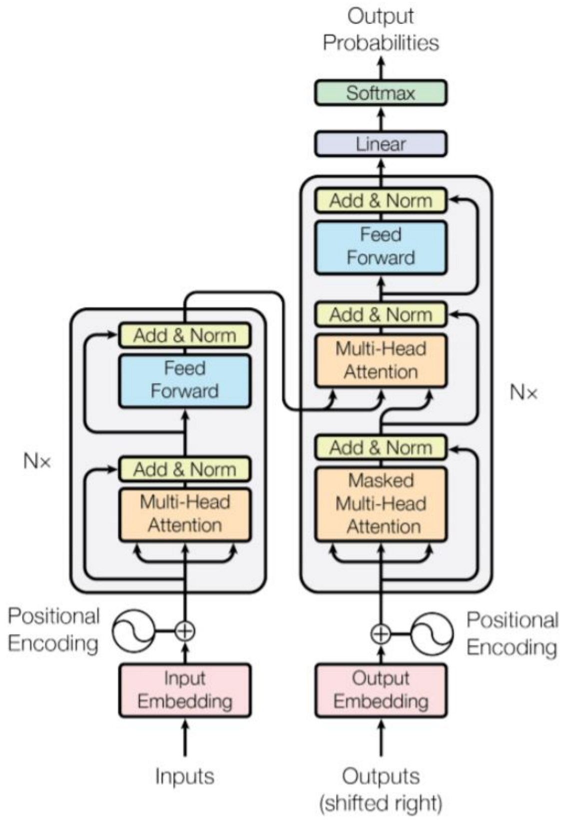
- Grammar correction
- English to other languages
- SQL translate
- Classification
- Query rewriting
- Conversation bot
- …

Image Generation from Text

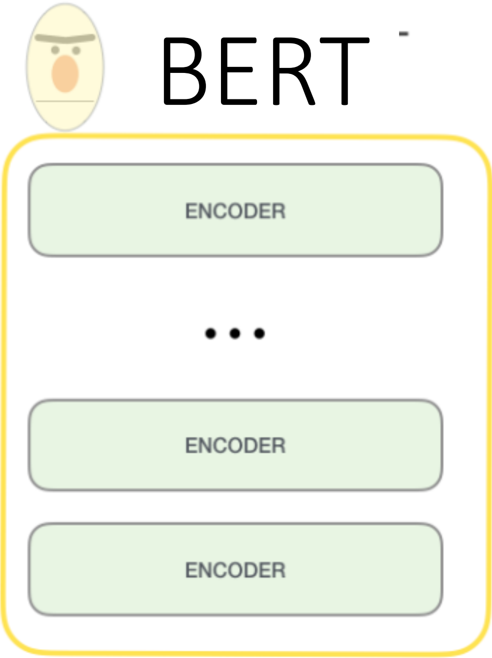


[DALL·E: Creating Images from Text - OpenAI]

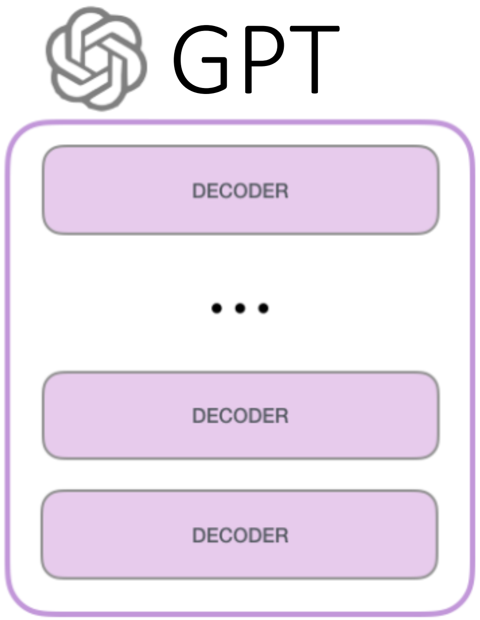
Transformers for Language Modeling



[1]



[2]



[3]

[1] Vaswani et al. "Attention Is All You Need", https://arxiv.org/abs/1706.03762, 2018
[2]Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", 2019, https://arxiv.org/abs/1810.04805
[3] Brown et al. "Language Models are Few-Shot Learners", 2020, https://arxiv.org/abs/2005.14165

DL System Challenges

- Too slow to **train** high-quality models on massive data
  - More hardware ≠ bigger model, higher throughput
  - Higher throughput ≠ better accuracy, faster convergence
  - Better techniques ≠ handy to use
- Slow and expensive to **deploy** the trained models

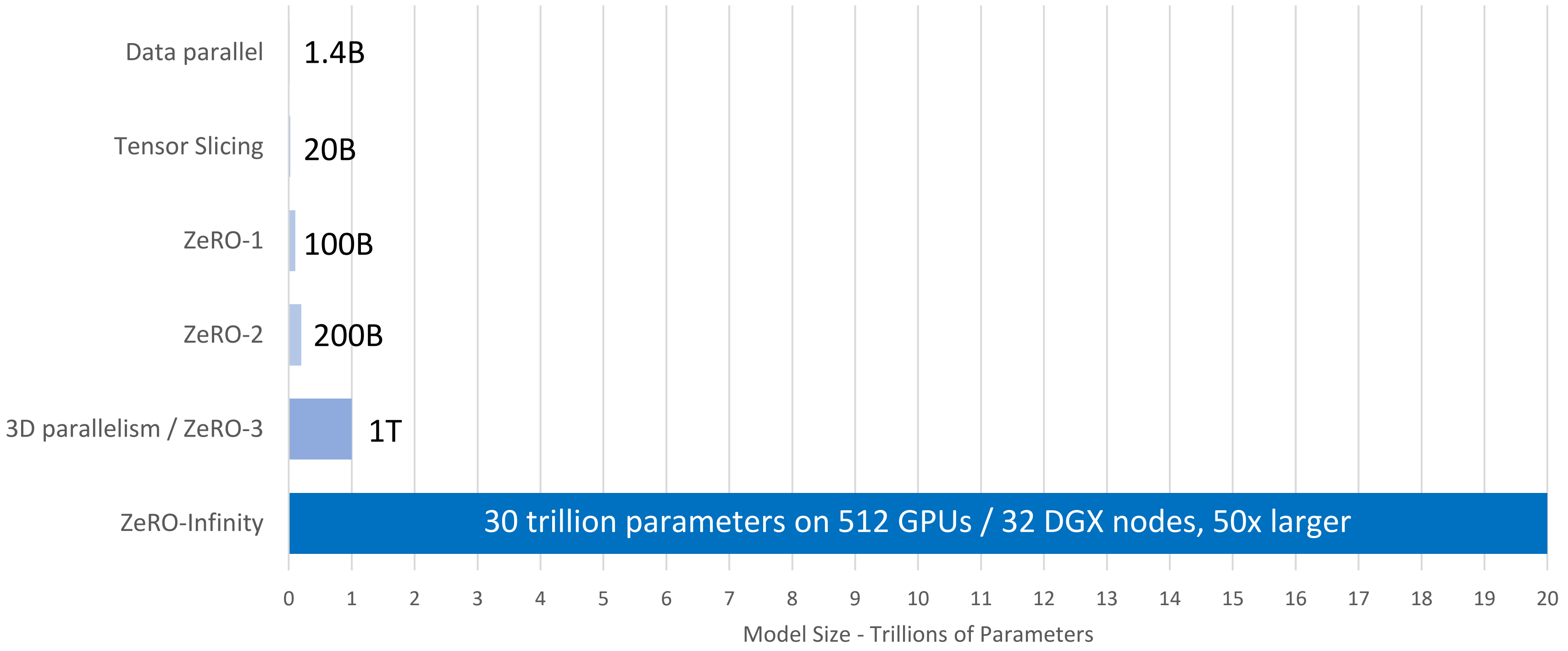
DL System Desired Capability (3Es)

Efficiency: Efficient use of hardware for high scalability and throughput
Effectiveness: High accuracy and fast convergence, lowering cost
Easy to use: Improve development productivity of model scientists

Outline

- DeepSpeed library overview
- ZeRO
  - Breaking the memory wall via memory efficient optimizer
- ZeRO-Offload
  - Democratizing DL training via heterogeneous memory
- Software and Usability

System capability to efficiently train models with over **10 trillion** parameters



DeepSpeed key technologies: Zero Redundancy Optimizer (ZeRO), 3D parallelism, ZeRO-Offload, ZeRO-Infinity

Large-scale models trained/in-training using DeepSpeed

- Active involvement of DS team: Z-code MoE 10B, Turing NLG 17B, Big Science LM 200B, MT-NLG 530B
- Independent efforts: GPT-NeoX 175B, Jurassic-1 178B

Model Scale
- 10+ Trillion parameters

Speed
- Fast & scalable training

Democratize AI
- Bigger & faster for all

Compressed Training
- Boosted efficiency

Accelerated inference
- Up to 6x faster & cheaper

Usability
- Few lines of code changes

Fastest Transformer Kernels

⇩

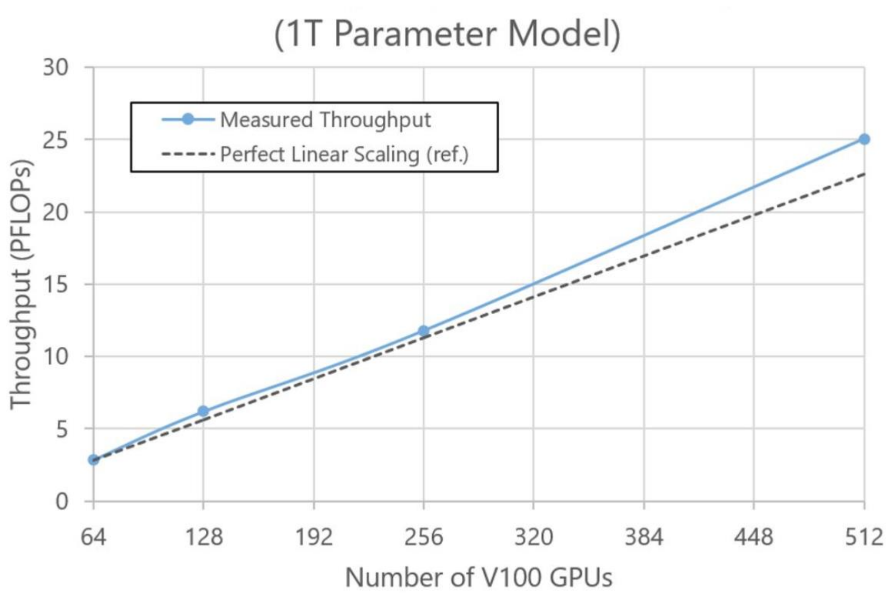
World Fastest BERT Training

#Devices	Source	Training Time
256 V100 GPUs	Nvidia	236 mins
256 V100 GPUs	DeepSpeed	144 mins
1024 TPU3 chips	Google	76 mins
1024 V100 GPUs	Nvidia	67 mins
1024 V100 GPUs	DeepSpeed	44 mins

Scalable distributed training through ZeRO-powered DP

⇩

Superlinear speedup with increasing #GPUs



DeepSpeed key technologies

- Efficiency: ZeRO, ultra-fast GPU kernels, IO/compute/communication overlapping
- Effectiveness: Advance HP tuning, large-batch scaling

Model Scale
- 10+ Trillion parameters

Speed
- Fast & scalable training

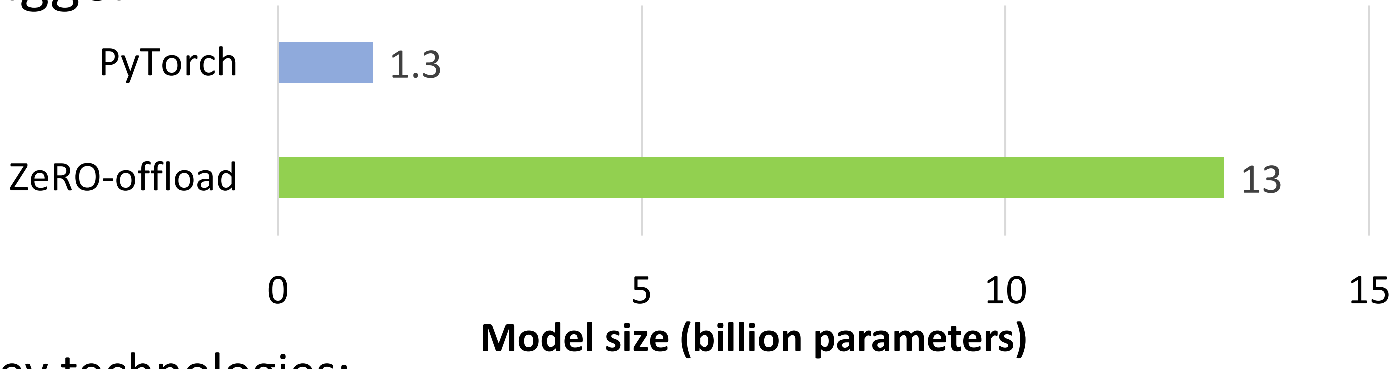
Democratize AI
- Bigger & faster for all

Compressed Training
- Boosted efficiency

Accelerated inference
- Up to 10x faster & cheaper

Usability
- Few lines of code changes

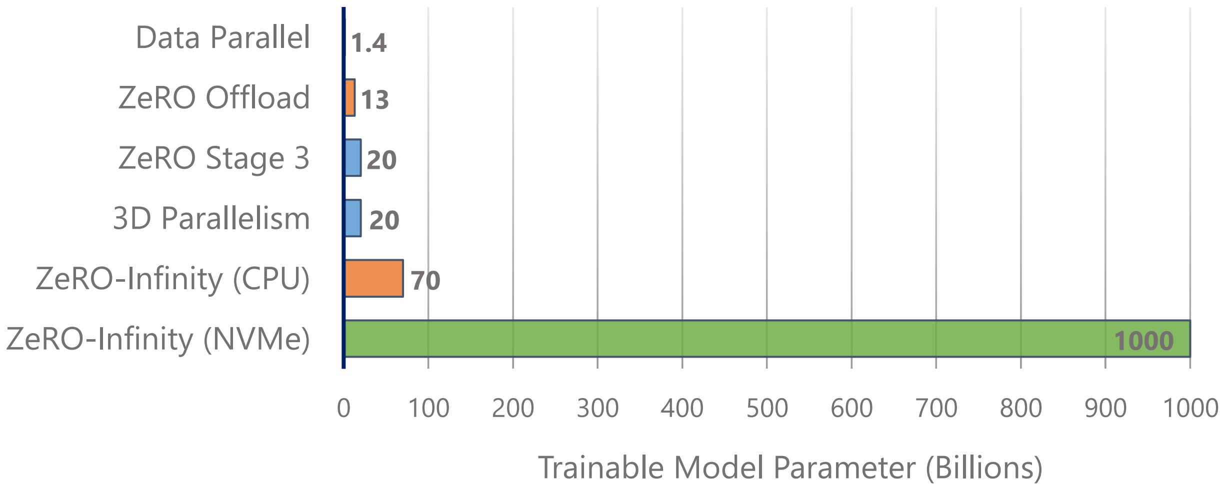
ZeRO-Offload (GPU + CPU): 13B model on single GPU, 10x bigger



Key technologies:

- Heterogeneous memory, ZeRO-style data parallelism, efficient tensor allocation and migration

ZeRO-Infinity (GPU + CPU + NVMe): 1 Trillion model on a single GPU, 700x bigger



Model Scale
- 10+ Trillion parameters

Speed
- Fast & scalable training

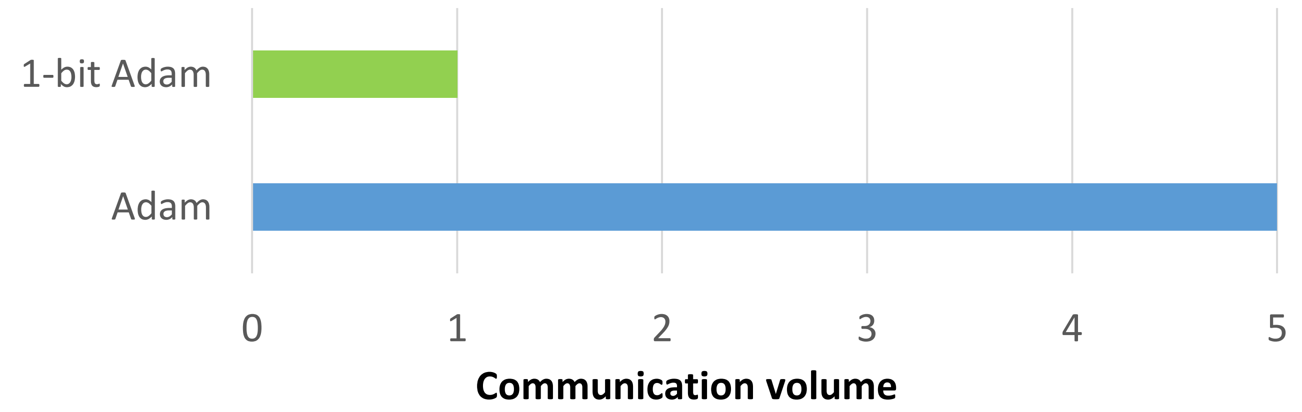
Democratize AI
- Bigger & faster for all

Compressed Training
- Boosted efficiency

Accelerated inference
- Up to 10x faster & cheaper

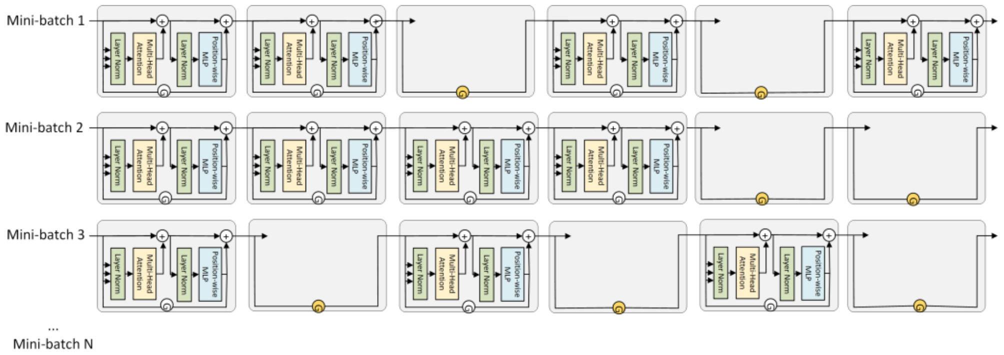
Usability
- Few lines of code changes

1-bit Adam: 5x less communication, 3.5x faster training



Key technologies: Gradient compression, error compensation

Progressive layer dropping: 2.5X faster pre-training speed to get similar accuracy on downstream tasks



Key technologies: Curriculum-based layer dropping, architecture change

Model Scale
- 10 Trillion parameters

Speed
- Fast & scalable training

Democratize AI
- Bigger & faster for all

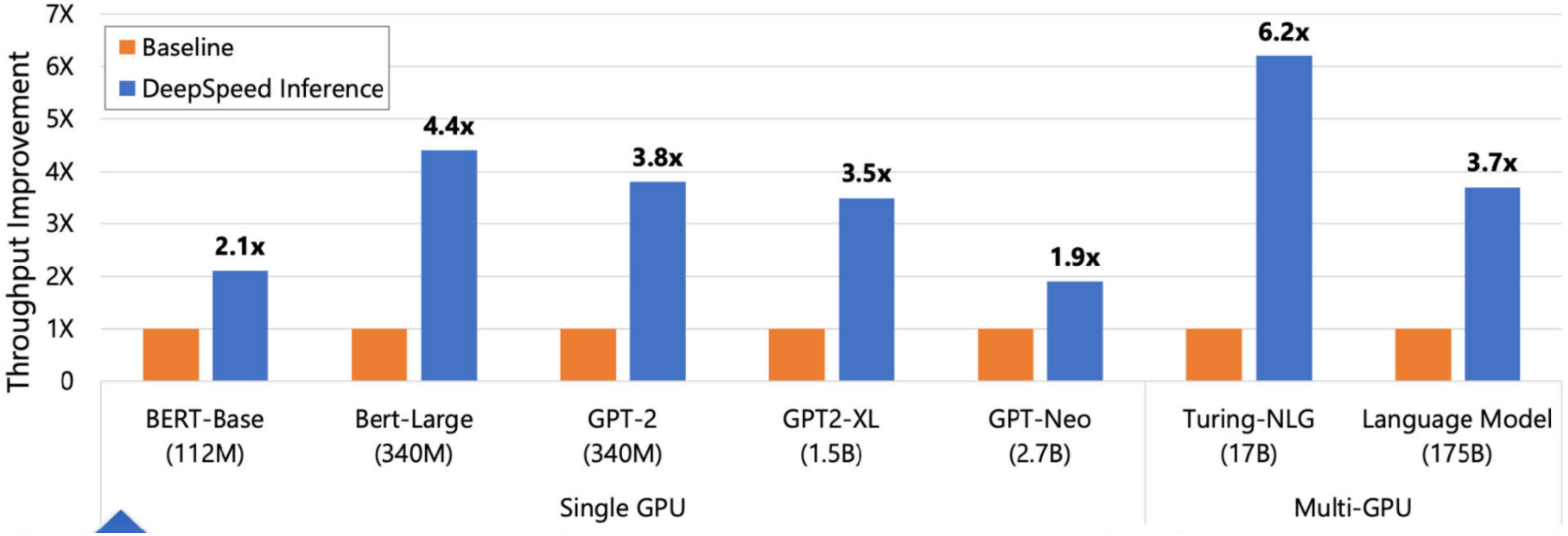
Compressed Training
- Boosted efficiency

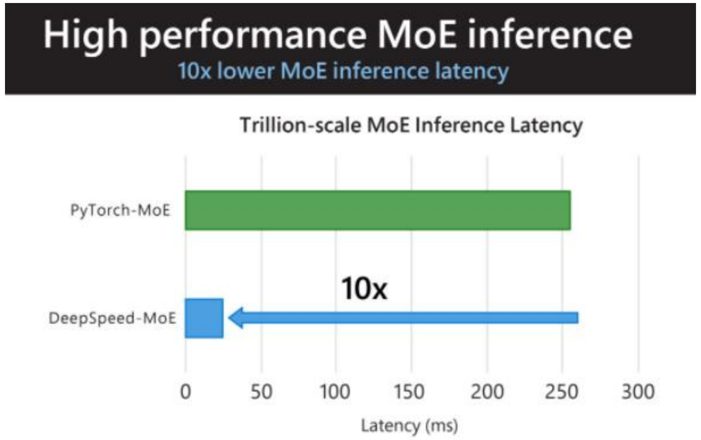
Accelerated inference
- Up to 6x faster & cheaper

Usability
- Few lines of code changes

Accelerated inference for large-scale transformer models

Up to 6x faster and cheaper





DeepSpeed key inference technologies:

- Inference optimized **kernels**
- Inference adapted **parallelism**
- Effective and flexible quantization for model **compression**
- **MoE-specific** optimizations

Model Scale
- 10+ Trillion parameters

Speed
- Fast & scalable training

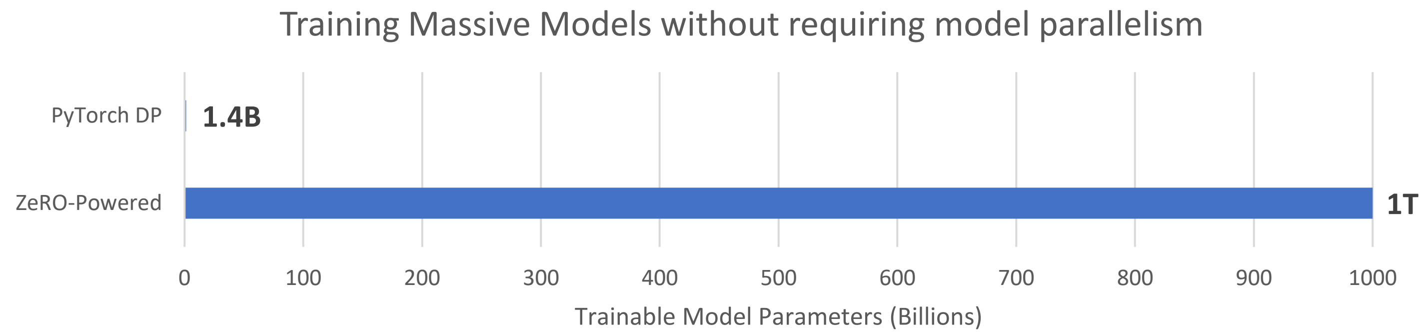
Democratize AI
- Bigger & faster for all

Compressed Training
- Boosted efficiency

Accelerated inference
- Up to 10x faster & cheaper

Usability
- Few lines of code changes

- Only few lines of code changes to enable DeepSpeed on PyTorch models
- Scalable and convenient data parallelism



- HuggingFace and PyTorch Lightning integrate DeepSpeed as a performance-optimized backend

```
deepspeed examples/pytorch/translation/run_translation.py \
--deepspeed tests/deepspeed/ds_config_zero3.json \
--model_name_or_path t5-small --per_device_train_batch_size 1 \
--output_dir output_dir --overwrite_output_dir --fp16 \
```

```
trainer = Trainer(gpus=4, plugins='deepspeed', precision=16)
```

deepspeed.py hosted with ❤ by GitHub view raw

- Infrastructure agnostic, supporting AzureML, Azure VMs, local-nodes

Model Scale
- 10+ Trillion parameters

Speed
- Fast & scalable training

Democratize AI
- Bigger & faster for all

Compressed Training
- Boosted efficiency

Accelerated inference
- Up to 10x faster & cheaper

Usability
- Few lines of code changes

ZeRO

Breaking the memory wall via memory efficient optimizer

ML/DL Training Problem Definition Recap

- Given model f, data set $\{xi, y_i\}_{i=1}^{N}$
- Minimize the loss between predicted labels and true labels:

  Min $\frac{1}{N}\sum_{i=1}^{N} loss(f(x_i, y_i))$

- Common loss function
  - Cross-entropy, MSE (mean squared error)
- Common way to solve the minimization problem
  - Stochastic gradient descent (SGD)
  - Adaptive learning rates optimizers (e.g., Adam)

Gradient Descent

- Model f_w is parameterized by weight w
- η > 0 is the learning rate

For t = 1 to T

$$\Delta w = \eta \times \frac{1}{N} \sum_{i=1}^{N} \underbrace{\boxed{\nabla}}_{\text{Backward pass}} \left(\underbrace{\boxed{loss\left(f_w(x_i, y_i)\right)}}_{\text{Forward pass}} \right)$$ // compute derivative and update

w -= Δw // apply update

End

Adaptive Learning Rates (Adam)

- Model f_w is parameterized by weight w
- η > 0 is the learning rate

For t = 1 to T

$$\Delta w = \eta \times \frac{1}{N} \sum_{i=1}^{N} \nabla \left(loss\left(f_w(x_i, y_i)\right)\right)$$

w -= $\boxed{\Delta w}$ // apply update

End

$$\nu_t = \beta_1 * \nu_{t-1} - (1 - \beta_1) * g_t$$
$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2$$
$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$

g_t : *Gradient at time t along* ω^j

ν_t : *Exponential Average of gradients along* ω_j

s_t : *Exponential Average of squares of gradients along* ω_j

β_1, β_2 : *Hyperparameters*

[1] Kingma and Ba, "Adam: A Method for Stochastic Optimization", 2014, https://arxiv.org/abs/1412.6980

Distributed Gradient Descent

- Model f_w is parameterized by weight w
- η > 0 is the learning rate

Can be parallelized

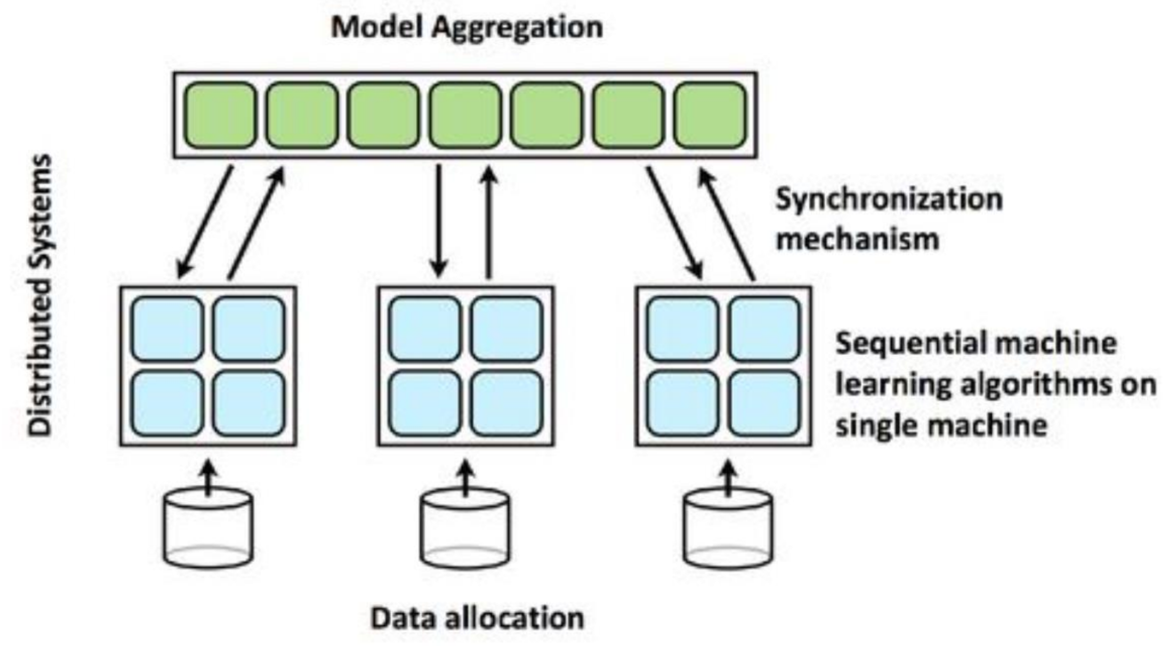
For t = 1 to T

$$\boxed{\Delta w = \eta \times \frac{1}{N}\sum_{i=1}^{N} \nabla \left(loss\left(f_w(x_i, y_i)\right)\right)}$$ // compute derivative and update

w -= Δw   // apply update

End

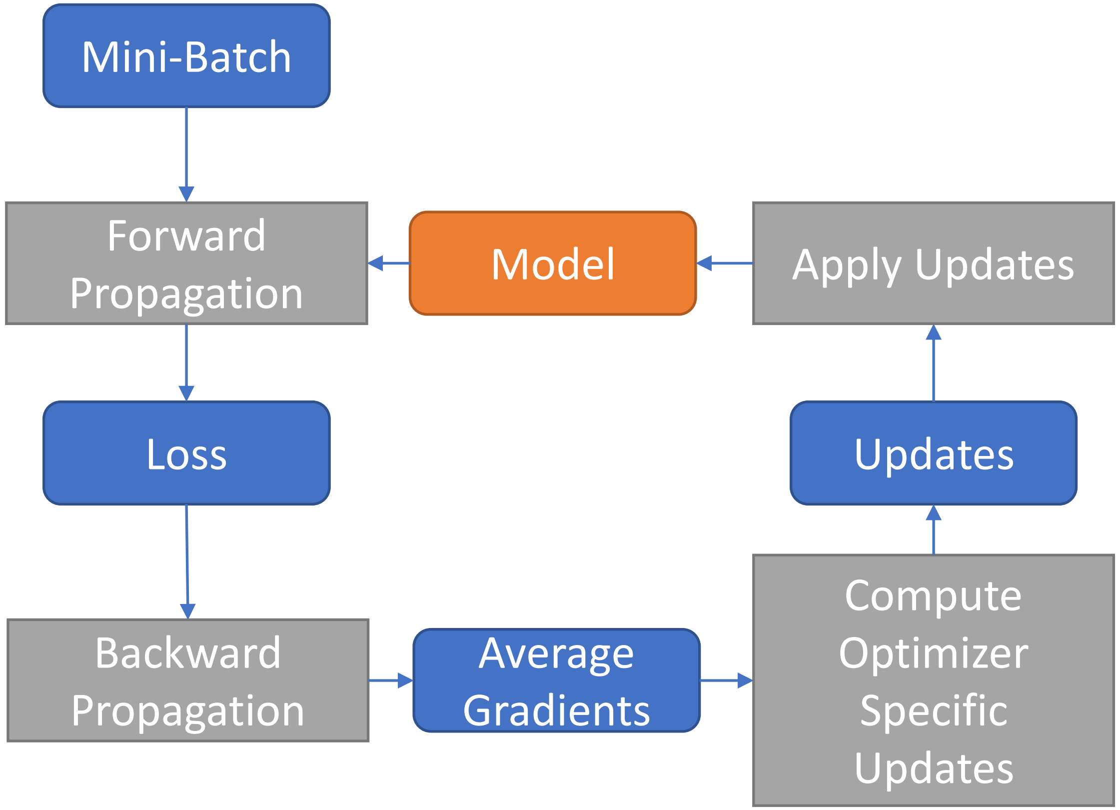
Data Parallelism (DP)



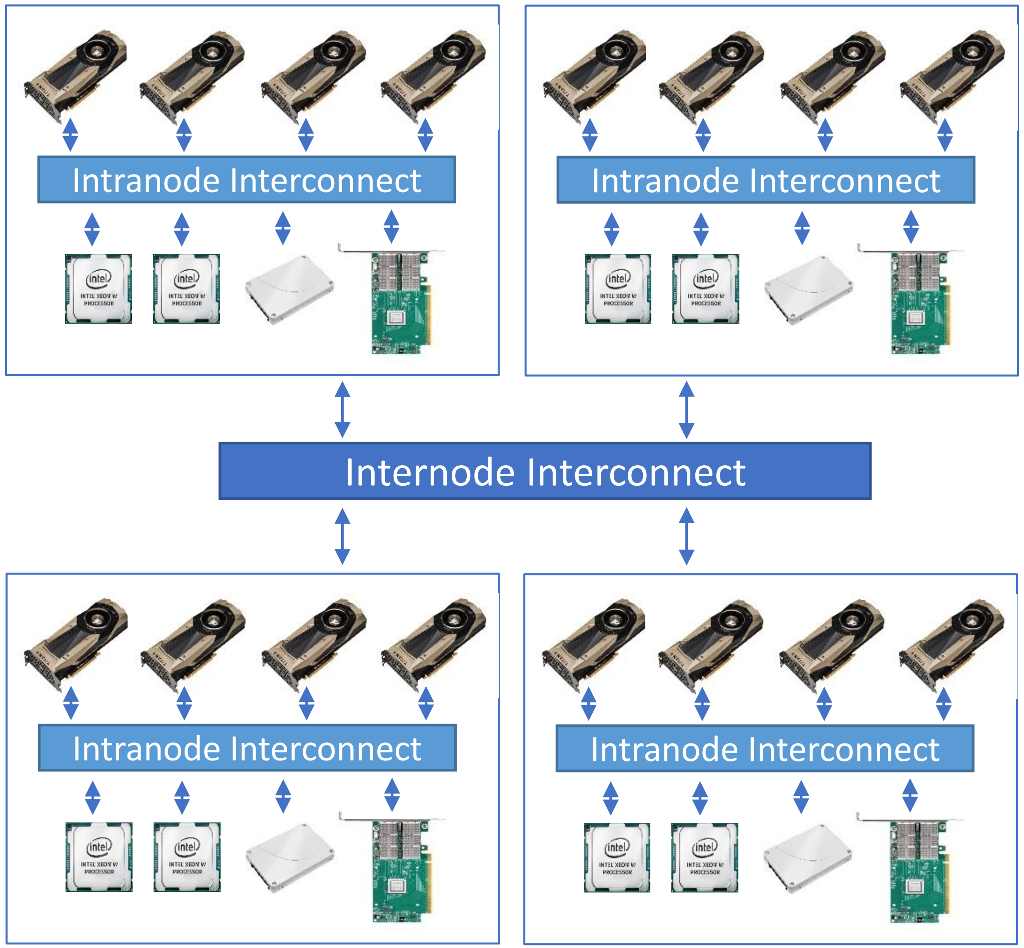
1. Partition the training data
2. Parallel training on different machines
3. Synchronize the local updates
4. Refresh local model with new parameters, then go to 2

Implemented as standard component in DL training frameworks, such as PyTorch DDP

Distributed Data Parallel Training in GPU Clusters



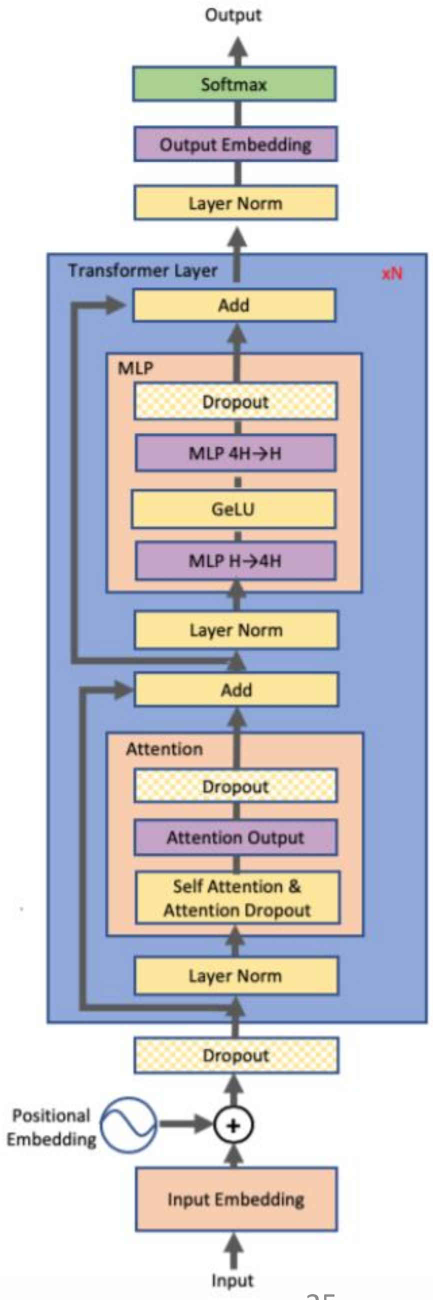
Data Parallel Training Loop



Distributed GPU Cluster

Large Model Training Challenges

	Bert-Large	GPT-2	Turing 17.2 NLG	GPT-3
Parameters	0.32B	1.5B	17.2B	175B
Layers	24	48	78	96
Hidden Dimension	1024	1600	4256	12288
Relative Computation	1x	4.7x	**54x**	**547x**
Memory Footprint	5.12GB	24GB	**275GB**	**2800GB**

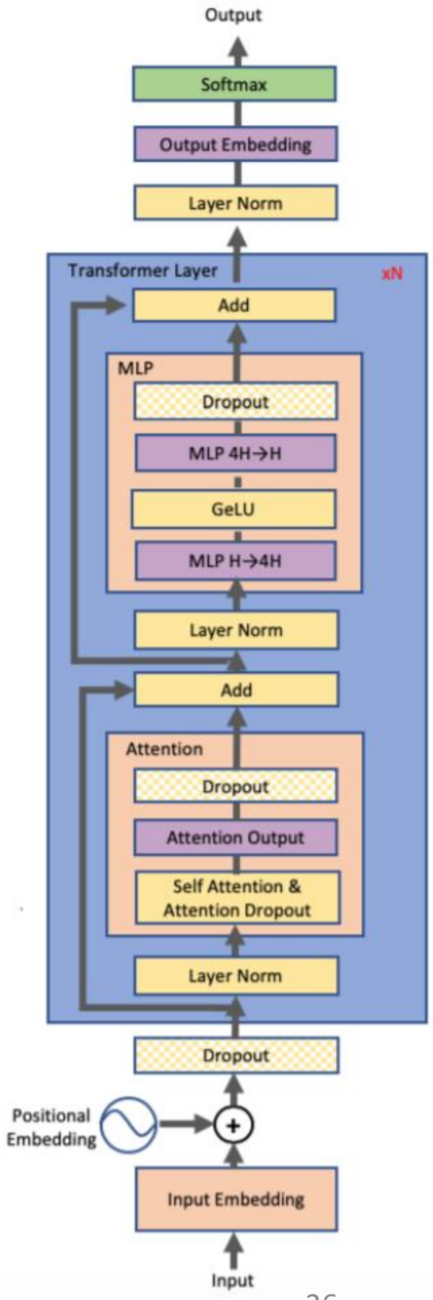


Large Model Training Challenges

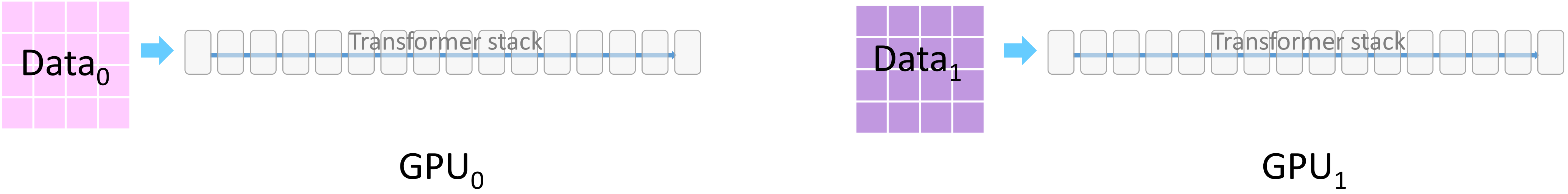
	Bert-Large	GPT-2	Turing 17.2 NLG	GPT-3
Parameters	0.32B	1.5B	17.2B	175B
Layers	24	48	78	96
Hidden Dimension	1024	1600	4256	12288
Relative Computation	1x	4.7x	**54x**	**547x**
Memory Footprint	5.12GB	24GB	**275GB**	**2800GB**

Out of Memory

NVIDIA V100 GPU memory capacity: 16G/32G
NVIDIA A100 GPU memory capacity: 40G/80G



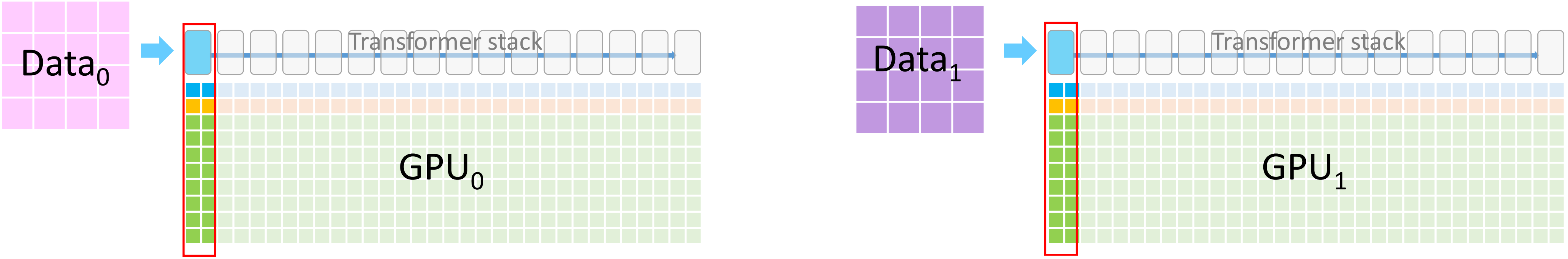
Understanding Memory Consumption



A 16-layer transformer model   = 1 layer

*[Mixed Precision Training](#) (ICLR ‘18) with Adam Optimizer

Understanding Memory Consumption



Each cell represents GPU memory used by its corresponding transformer layer

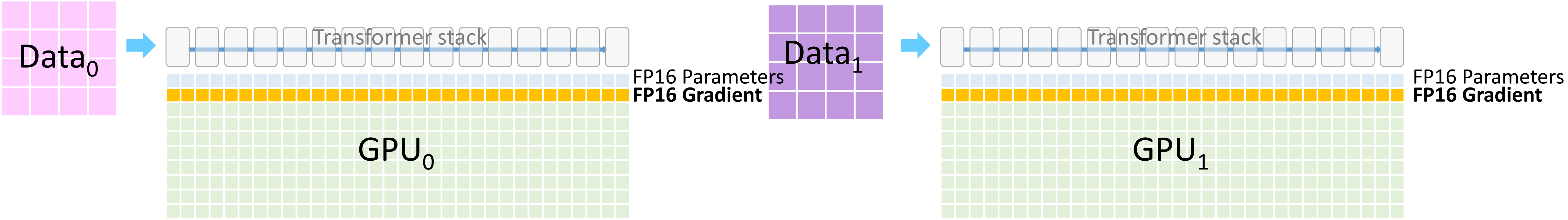
*Mixed Precision Training (ICLR '18) with Adam Optimizer

Understanding Memory Consumption

- FP16 parameter

*Mixed Precision Training (ICLR '18) with Adam Optimizer

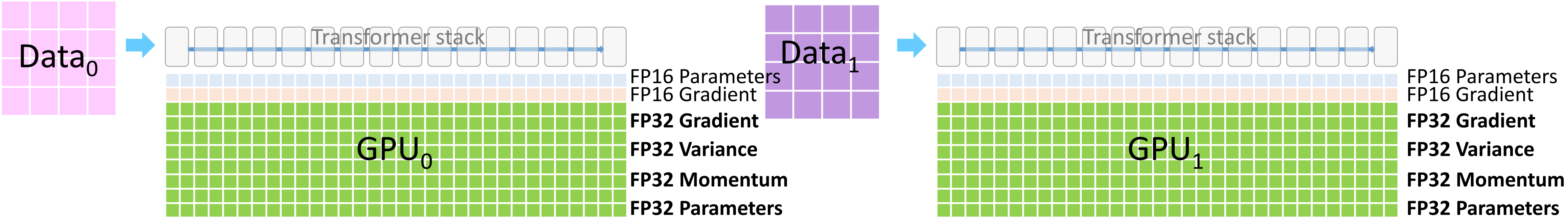
Understanding Memory Consumption



- FP16 parameter
- FP16 Gradients

*Mixed Precision Training (ICLR '18) with Adam Optimizer

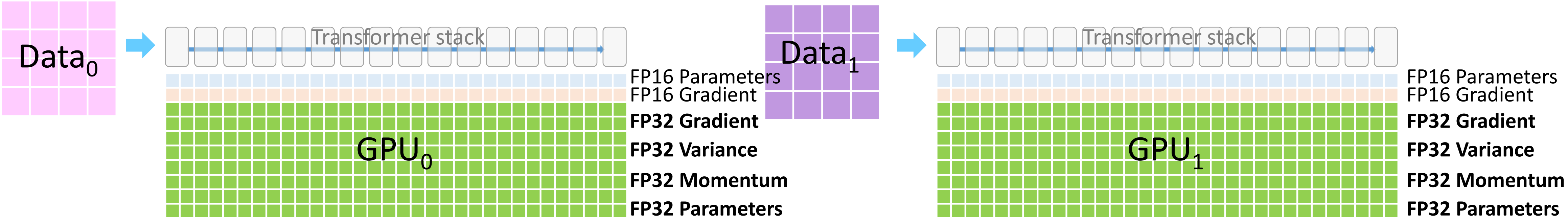
Understanding Memory Consumption



- FP16 parameter
- FP16 Gradients
- FP32 Optimizer States
  - Gradients, Variance, Momentum, Parameters

*Mixed Precision Training (ICLR '18) with Adam Optimizer

Understanding Memory Consumption



- FP16 parameter : **2M bytes**
- FP16 Gradients : **2M bytes**
- FP32 Optimizer States : **16M bytes**
  - Gradients, Variance, Momentum, Parameters

M = number of parameters in the model

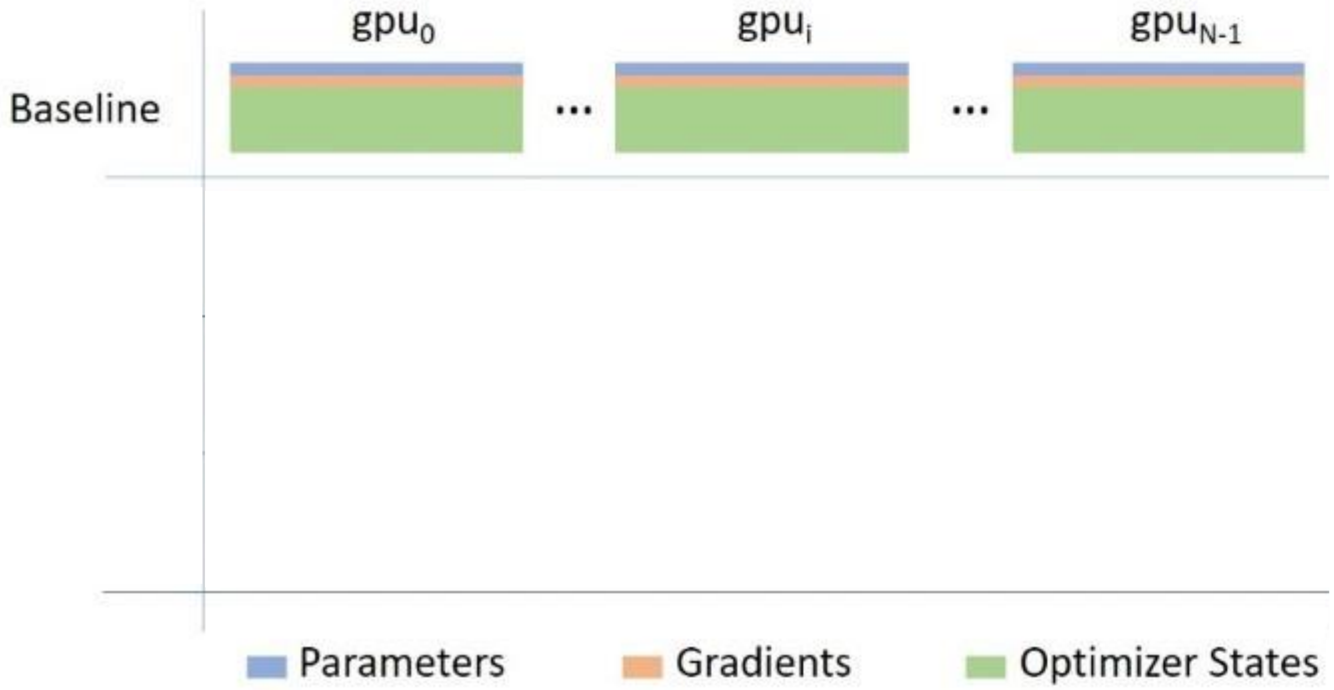
Example 1B parameter model -> 20GB/GPU

Memory consumption doesn't include:

- Input batch + activations

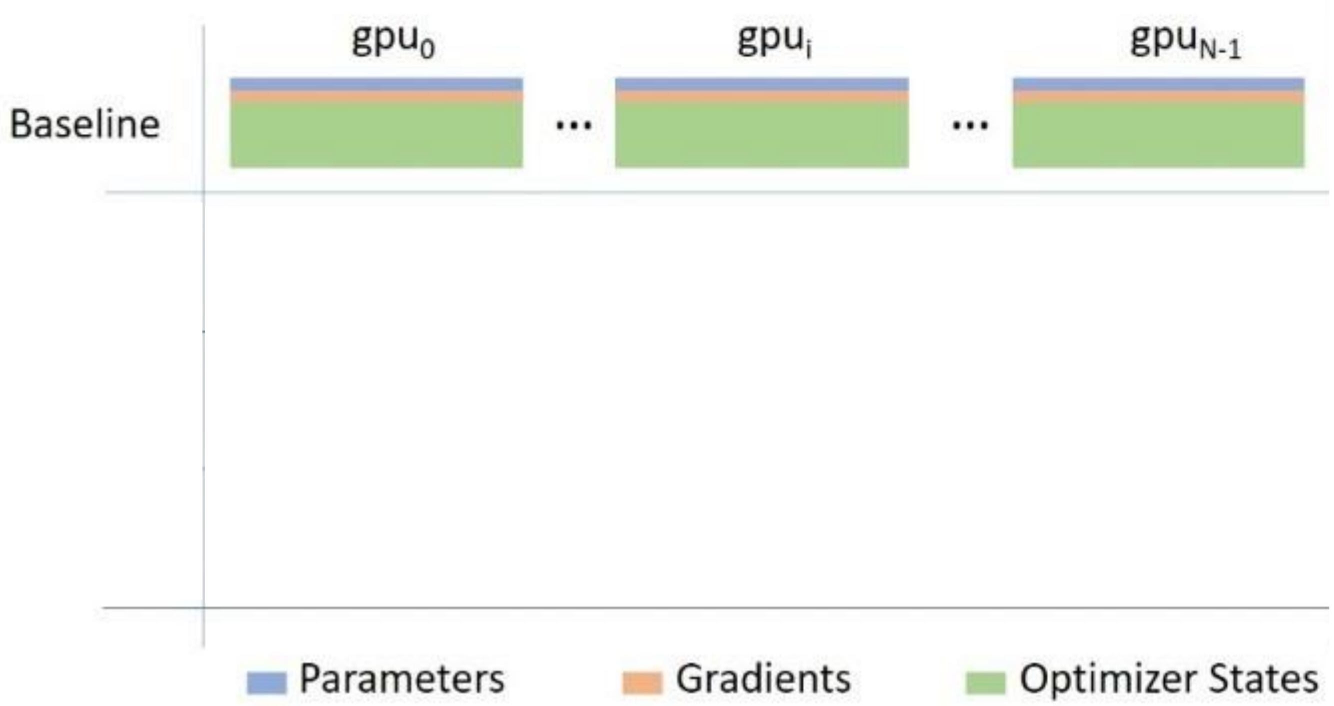
*Mixed Precision Training (ICLR '18) with Adam Optimizer

ZeRO-DP: ZeRO powered Data Parallelism



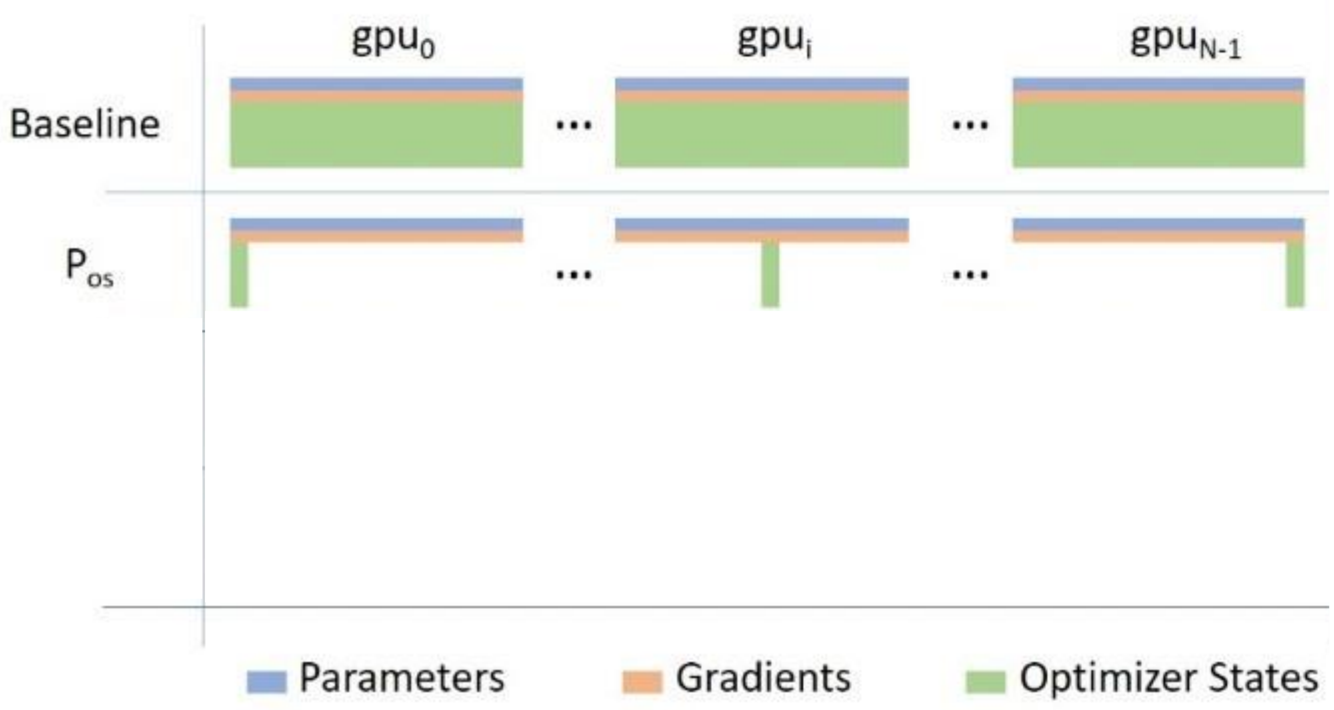
ZeRO-DP: ZeRO powered Data Parallelism

- ZeRO removes the redundancy across data parallel process
- Partitioning optimizer states, gradients and parameters (3 stages)



ZeRO-DP: ZeRO powered Data Parallelism

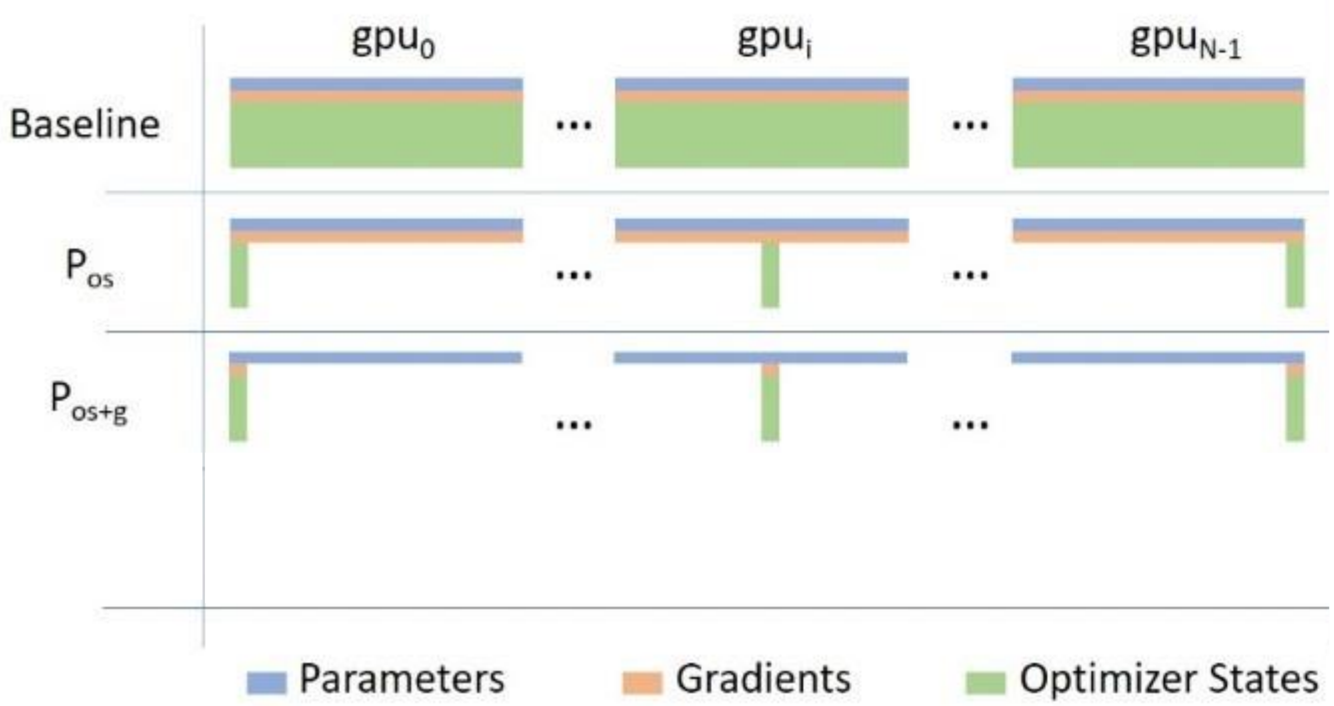
- ZeRO removes the redundancy across data parallel process
- Partitioning optimizer states, gradients and parameters (3 stages)



Stage 1 (P_{os})

ZeRO-DP: ZeRO powered Data Parallelism

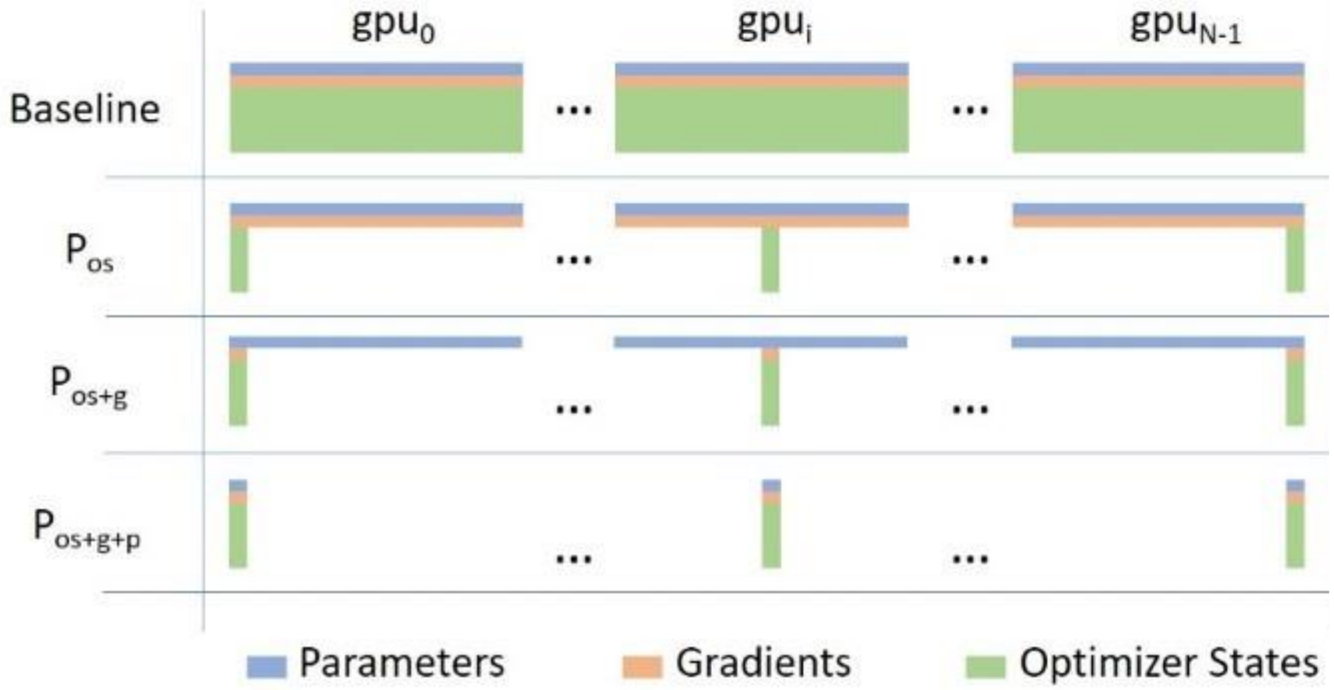
- ZeRO removes the redundancy across data parallel process
- Partitioning optimizer states, gradients and parameters (3 stages)



Stage 2 (P_{os+g})

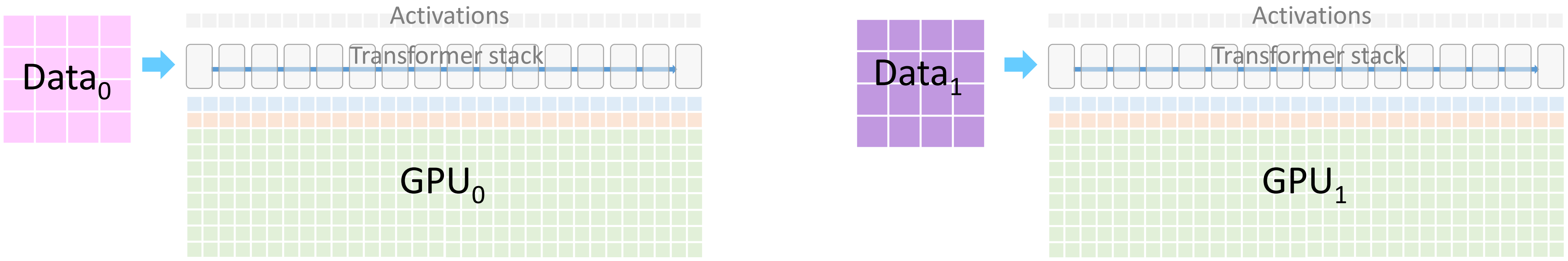
ZeRO-DP: ZeRO powered Data Parallelism

- ZeRO removes the redundancy across data parallel process
- Partitioning optimizer states, gradients and parameters (3 stages)



Stage 3 (P_{os+g+p})

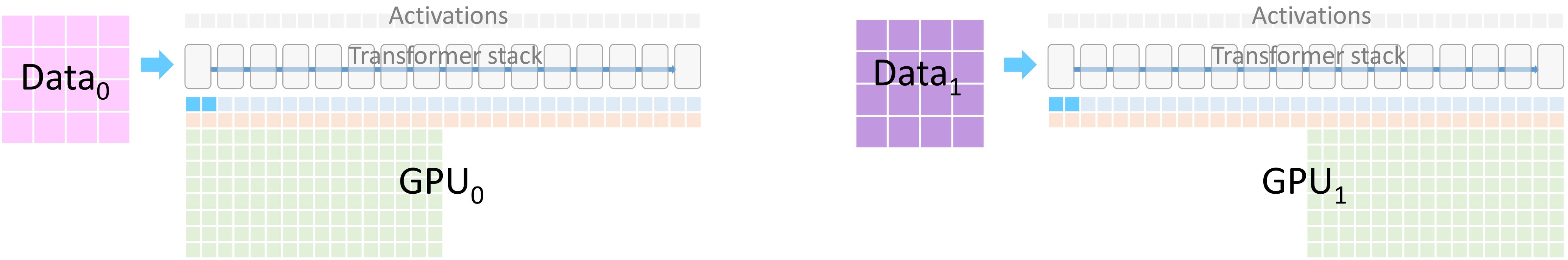
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1

Animation by Alec Berntson

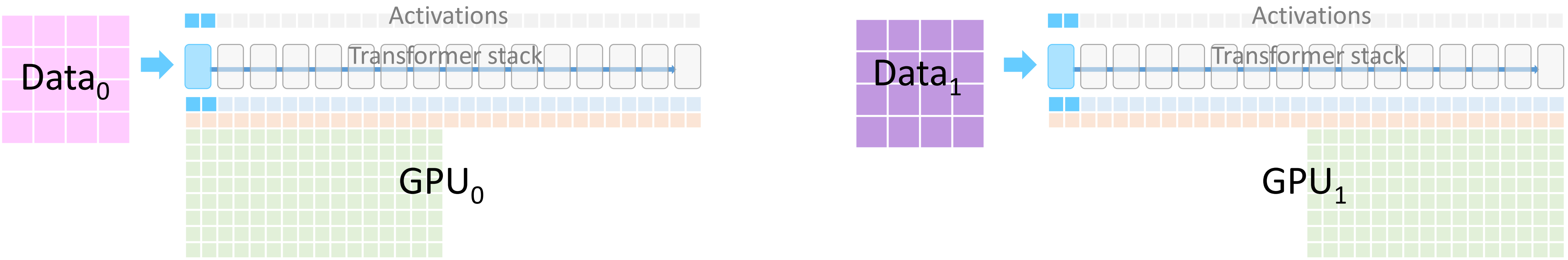
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs

Animation by Alec Berntson

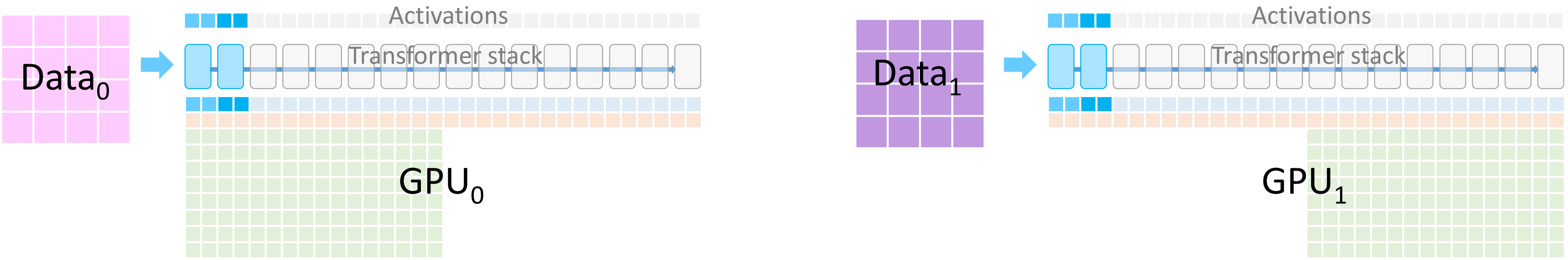
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks

Animation by Alec Berntson

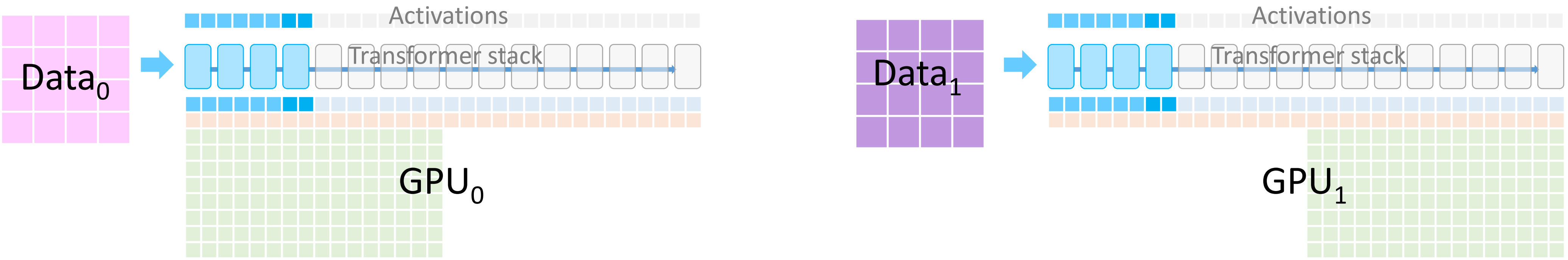
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks

Animation by Alec Berntson

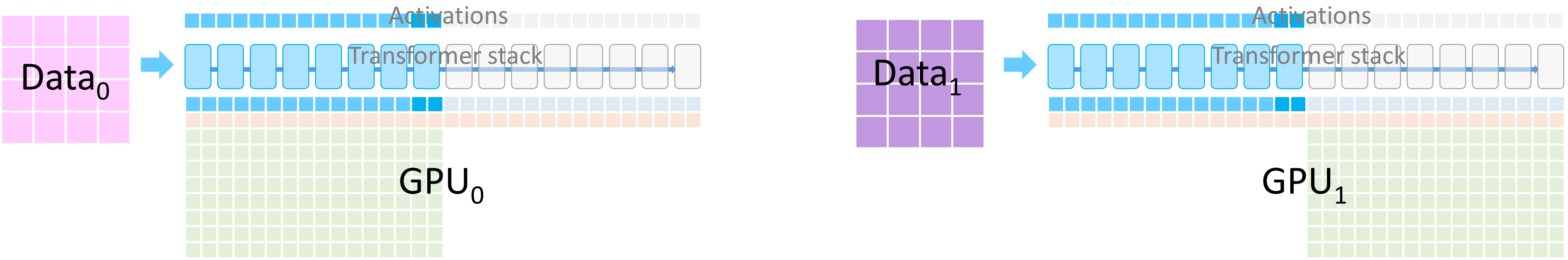
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks

Animation by Alec Berntson

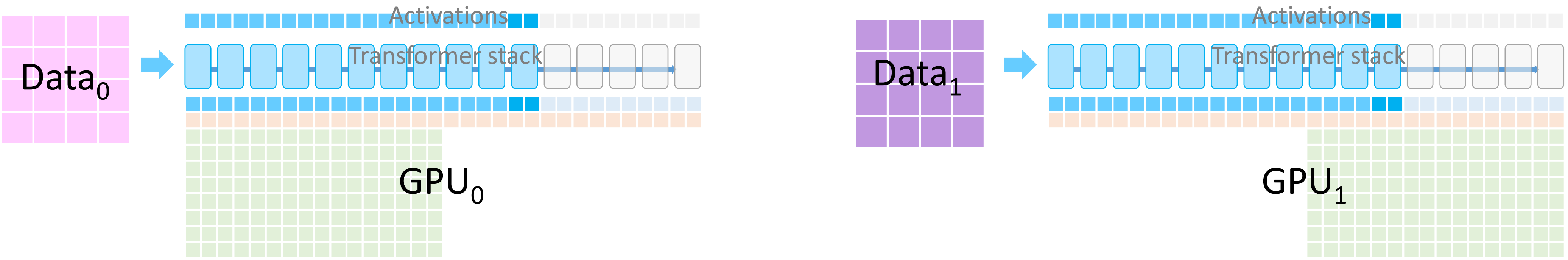
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks

Animation by Alec Berntson

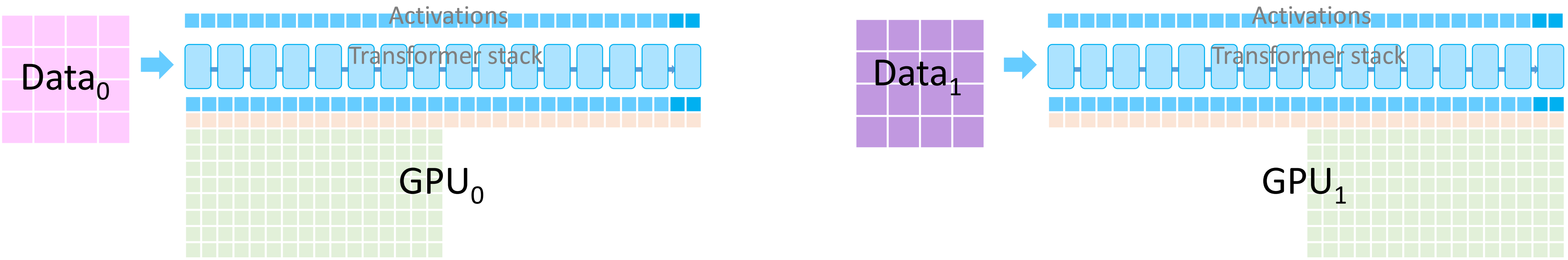
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks

Animation by Alec Berntson

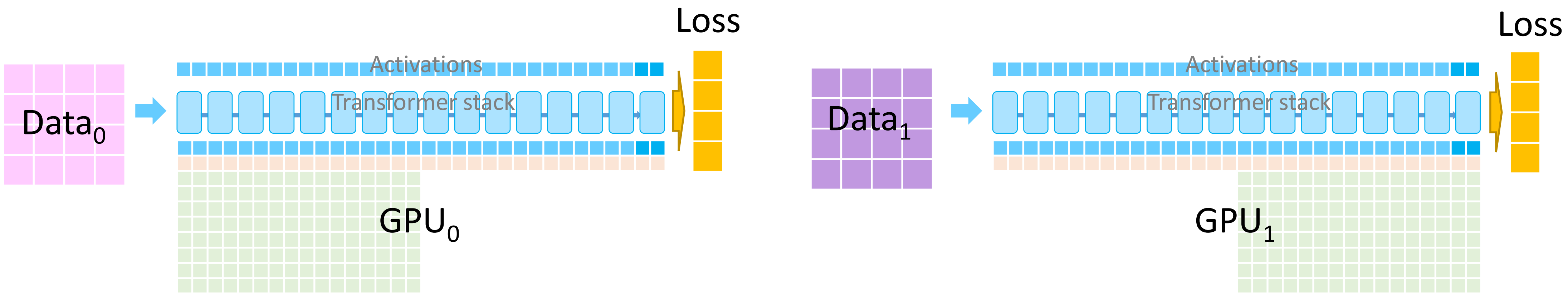
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks

Animation by Alec Berntson

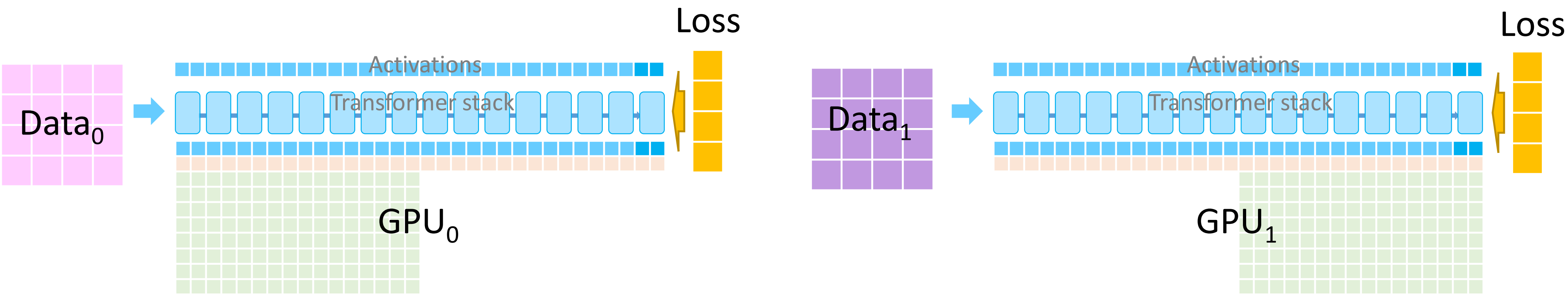
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks

Animation by Alec Berntson

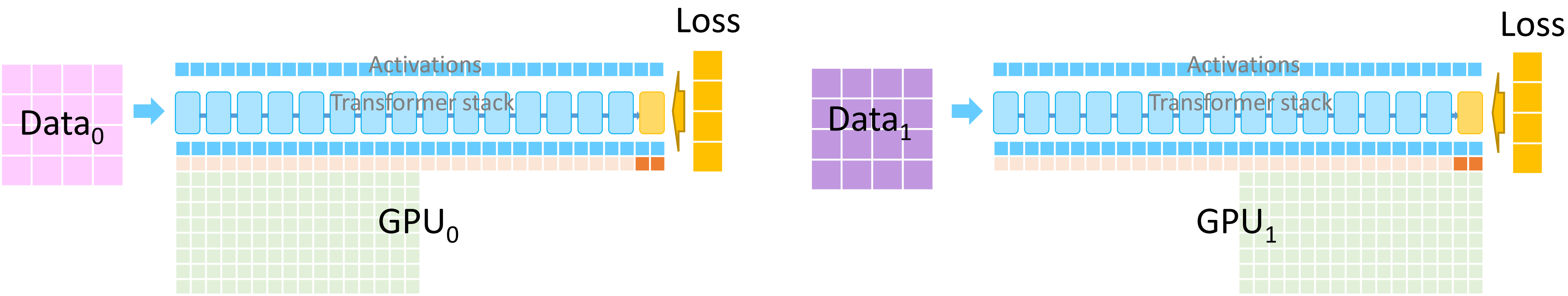
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients

Animation by Alec Berntson

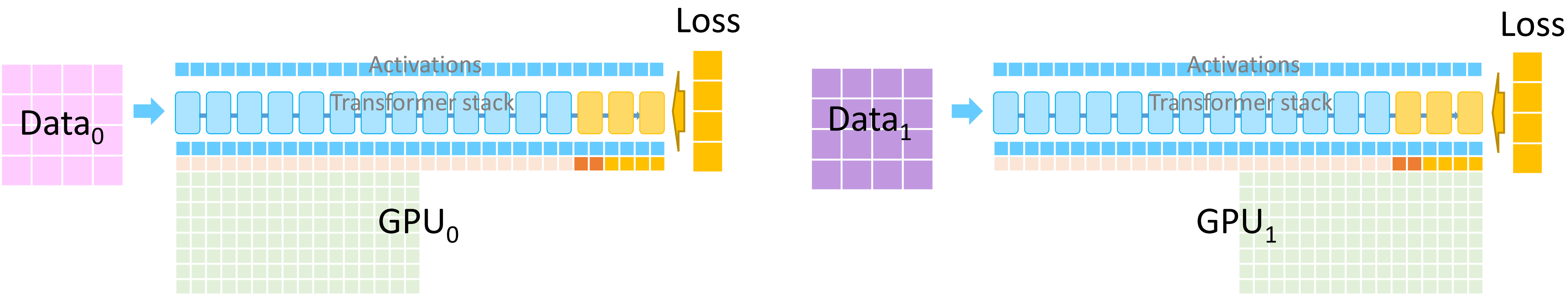
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients

Animation by Alec Berntson

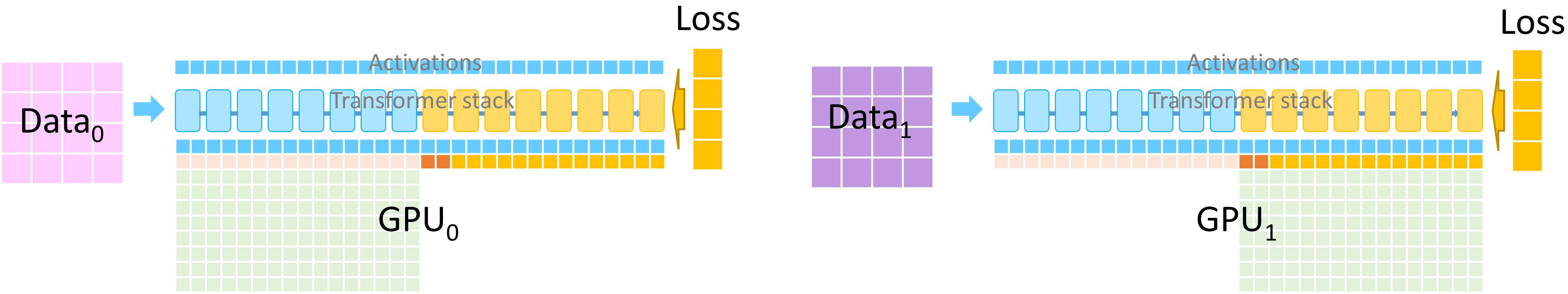
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients

Animation by Alec Berntson

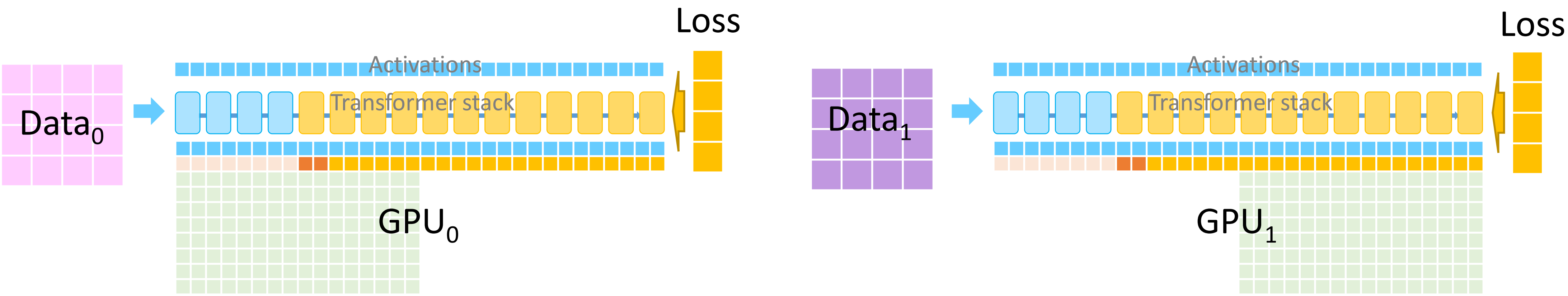
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients

Animation by Alec Berntson

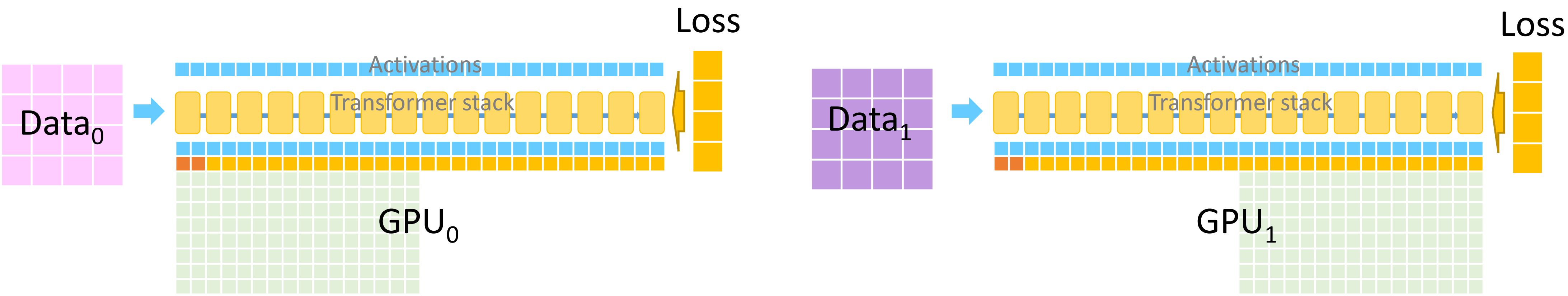
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients

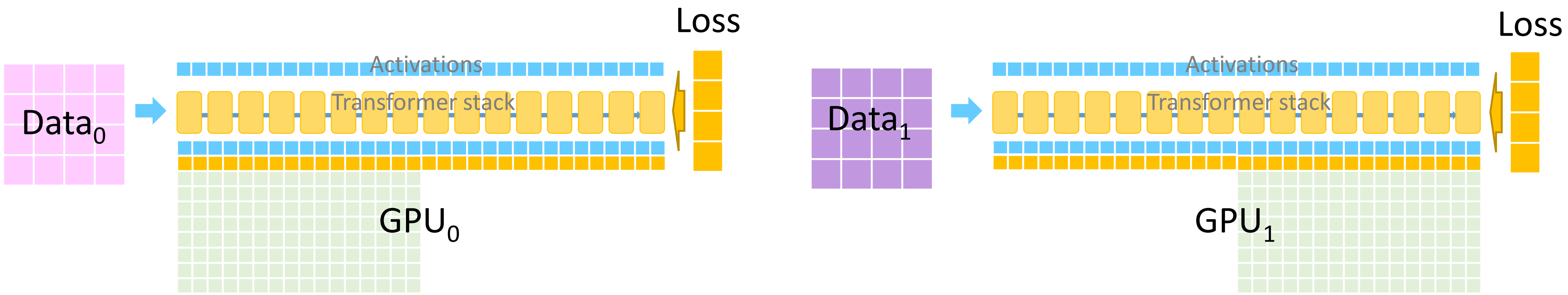
Animation by Alec Berntson

ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients

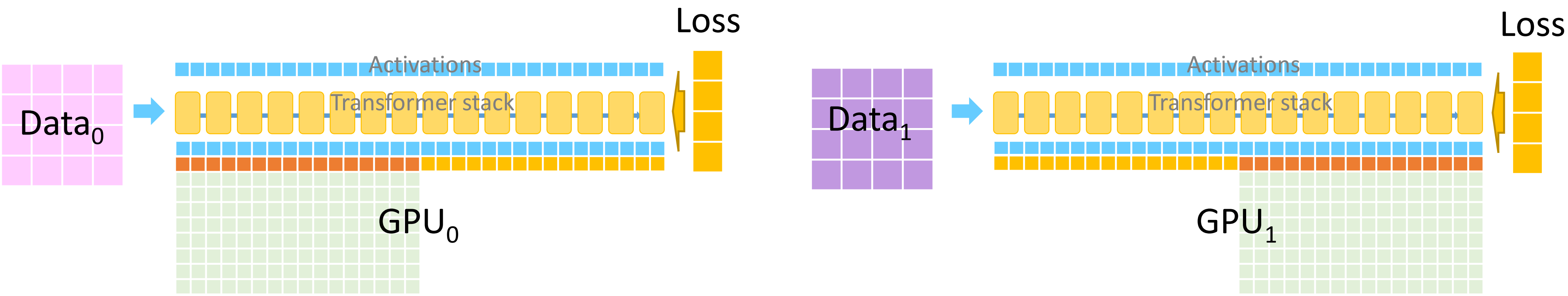
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and reduce scatter to average

Animation by Alec Berntson

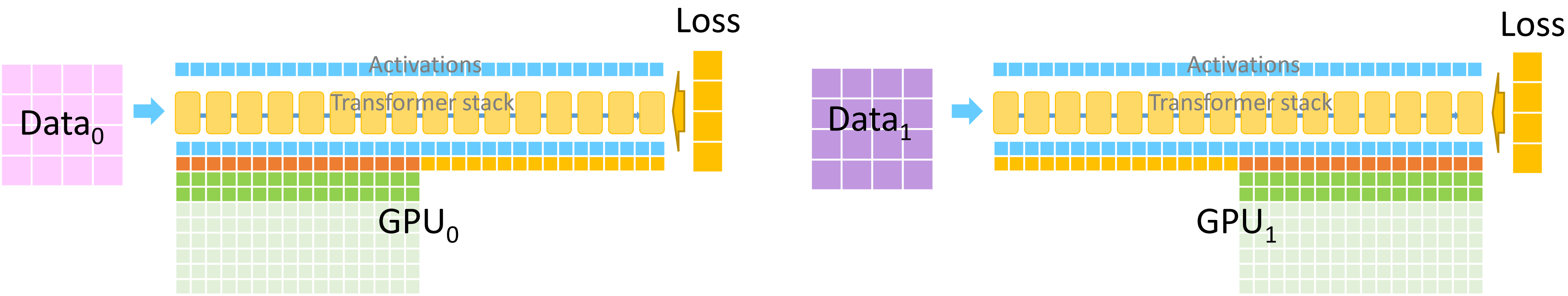
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and reduce scatter to average

Animation by Alec Berntson

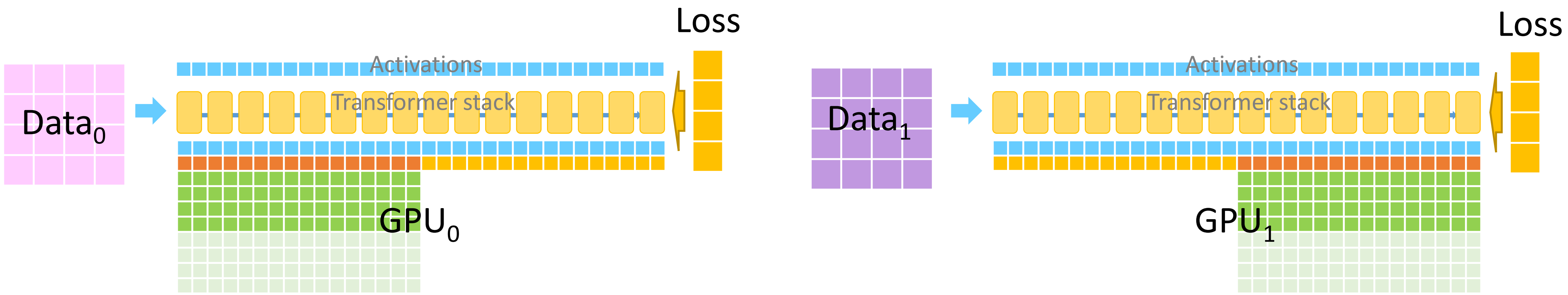
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and reduce scatter to average
- Update the FP32 weights with ADAM optimizer

Animation by Alec Berntson

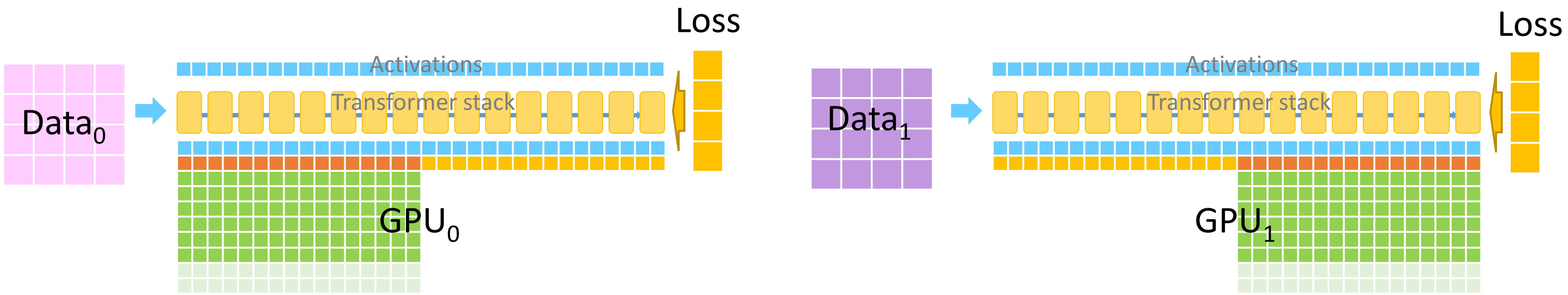
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and reduce scatter to average
- Update the FP32 weights with ADAM optimizer

Animation by Alec Berntson

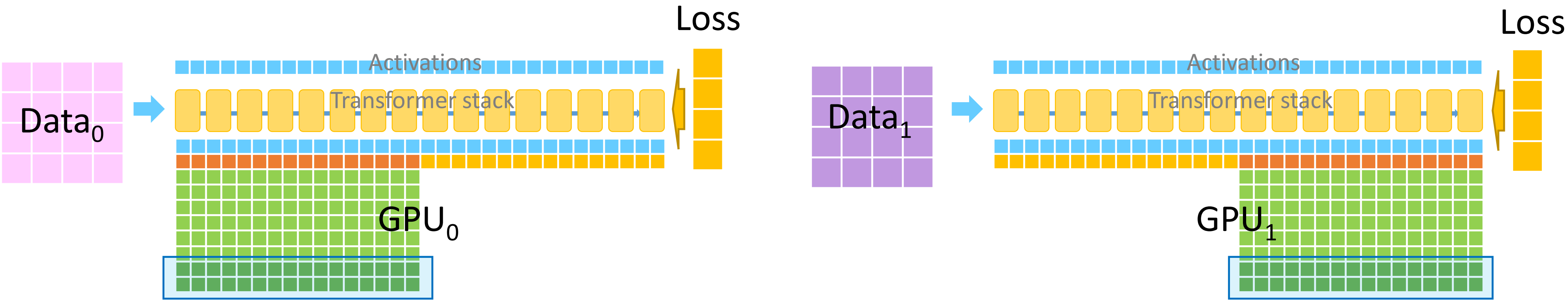
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and reduce scatter to average
- Update the FP32 weights with ADAM optimizer

Animation by Alec Berntson

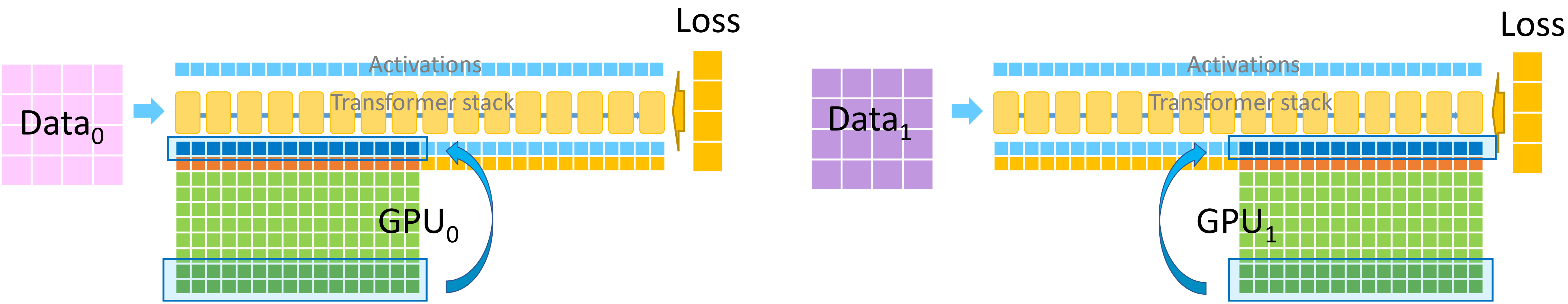
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and reduce scatter to average
- Update the FP32 weights with ADAM optimizer

Animation by Alec Berntson

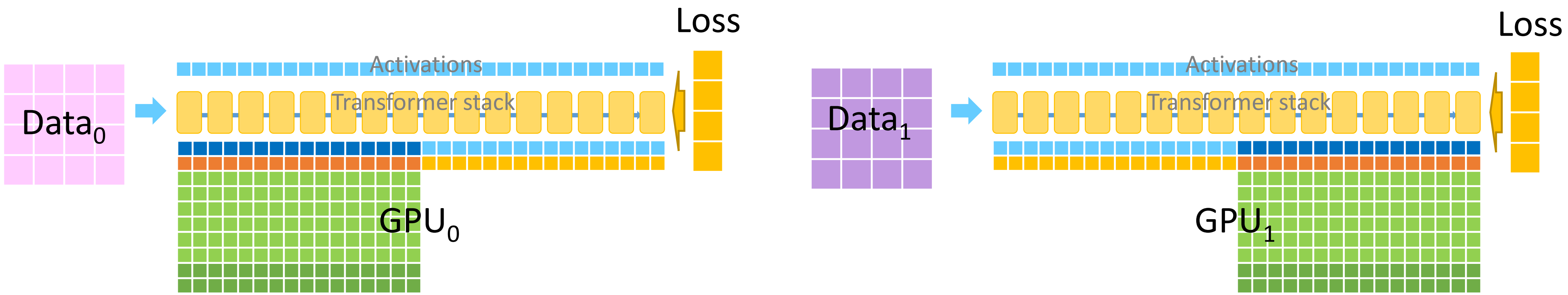
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and reduce scatter to average
- Update the FP32 weights with ADAM optimizer
- Update the FP16 weights

Animation by Alec Berntson

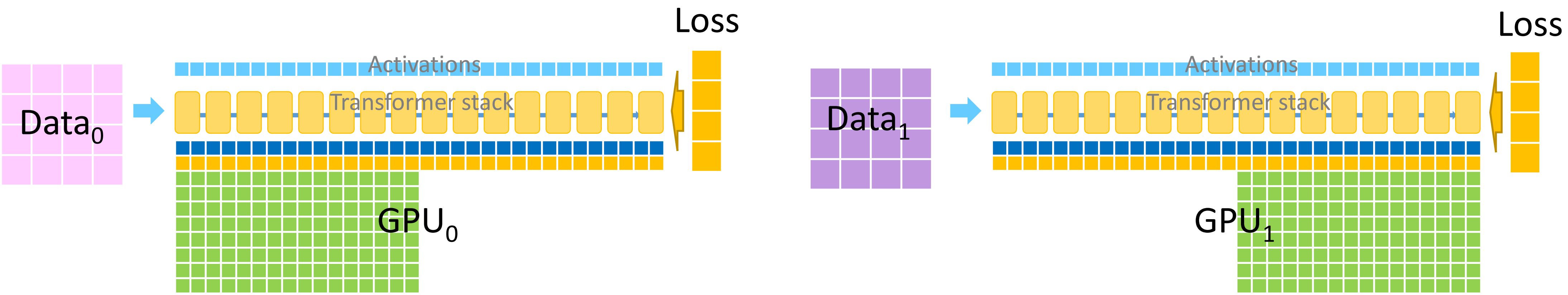
ZeRO: Zero Redundancy Optimizer



- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and reduce scatter to average
- Update the FP32 weights with ADAM optimizer
- Update the FP16 weights
- All Gather the FP16 weights to complete the iteration

Animation by Alec Berntson

ZeRO: Zero Redundancy Optimizer

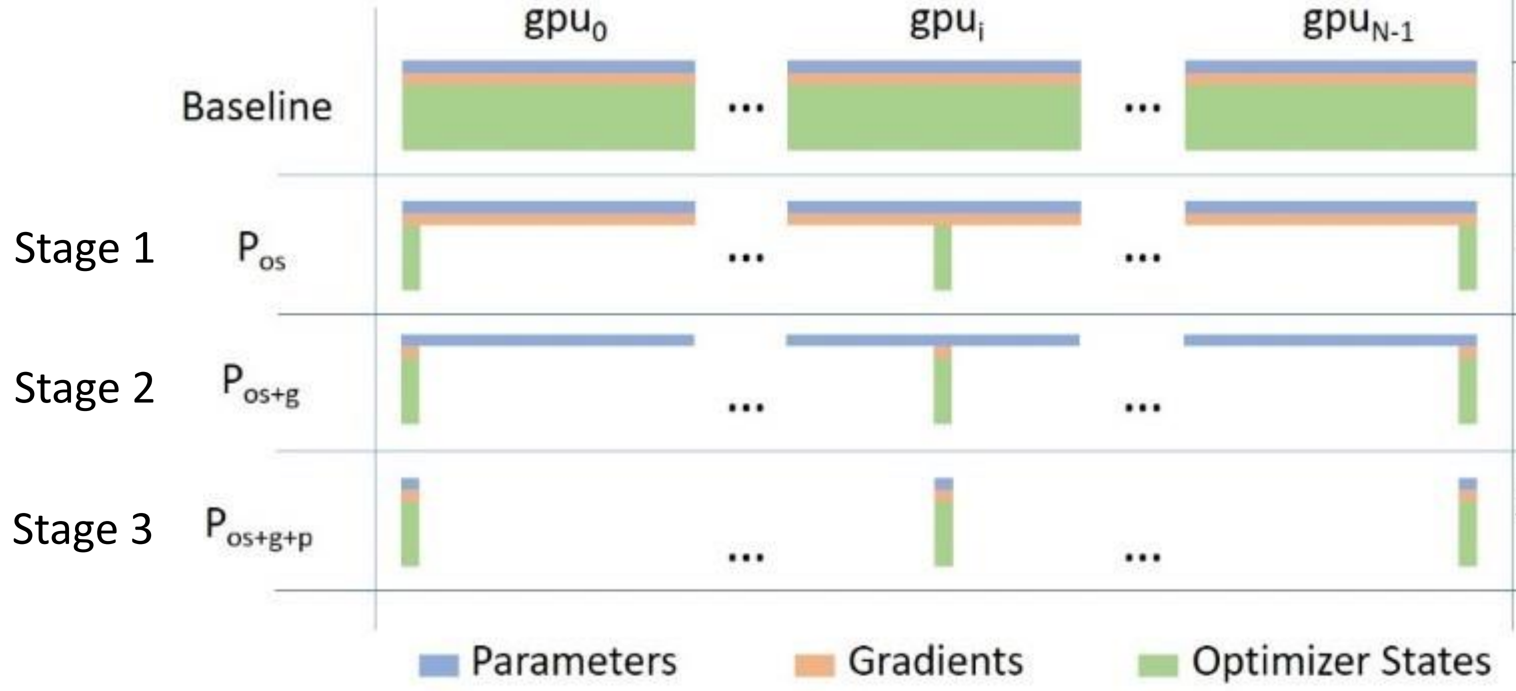


- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and reduce scatter to average
- Update the FP32 weights with ADAM optimizer
- Update the FP16 weights
- All Gather the FP16 weights to complete the iteration

Animation by Alec Berntson

ZeRO: Zero Redundancy Optimizer

- ZeRO has three different stages
- Progressive memory savings and communication volume
- Turning NLR 17.2B is powered by Stage 1 and Megatron



	Memory Consumption Reduction	Memory Consumption MaxSize	Comm Volume
Baseline	1x	1.2/20B	1x
Stage 1 (P_{os})	4x	6/100B	1x
Stage 2 (P_{os+g})	8x	12/200B	1x
Stage 3 (P_{os+g+p})	Nx	>1T	1.5x

ZeRO and Model/Pipeline Parallelism

- ZeRO is model parallelism agnostic
- Can work with any form of model parallelism
  - Tensor Slicing (Megatron[1])
  - Pipeline Parallelism (Gpipe[2], PipeDream[3])

[1] Shoeybi et al., Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism, 2019
[2] Huang et al. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism
[3] Harlap et al. PipeDream: Fast and Efficient Pipeline Parallel DNN Training

Large Models Need Parallelism

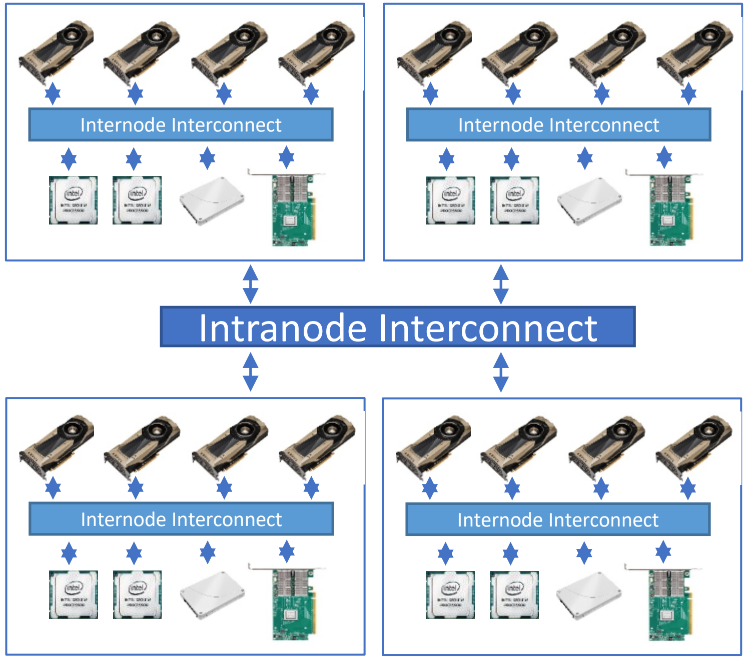
	Max Parameter (in billions)	Max Parallelism	Compute Efficiency	Usability (Model Rewrite)
Data Parallel (DP)	Approx. 1.2	>1000	Very Good	Great
Model Parallel (MP)	Approx. 20	Approx. 16	Good	Needs Model Rewrite
MP + DP	Approx. 20	> 1000	Good	Needs Model Rewrite
Pipeline Parallel (PP)	Approx. 100	Approx. 128	Very Good	Needs Model Rewrite
PP + DP	Approx. 100	> 1000	Very Good	Needs Model Rewrite
MP + PP + DP	> 1000	> 1000	Very Good	Needs Significant Model Rewrite
ZeRO	*> 1000*	*> 1000*	Very Good	Great

ZeRO-Offload

Democratizing DL training via heterogeneous memory

Billon-Scale Model Training - Scale Out Large Model Training

- Model parallelism (Megatron-LM)
  - Partition the model states vertically across multiple GPUs.
- Pipeline parallelism (PipeDream, SOSP'19)
  - Partition the model states horizontally across layers.
- ZeRO: Zero Redundancy Optimizer (ZeRO, SC'20)
  - Split the training batch across multiple GPUs without model states duplication.



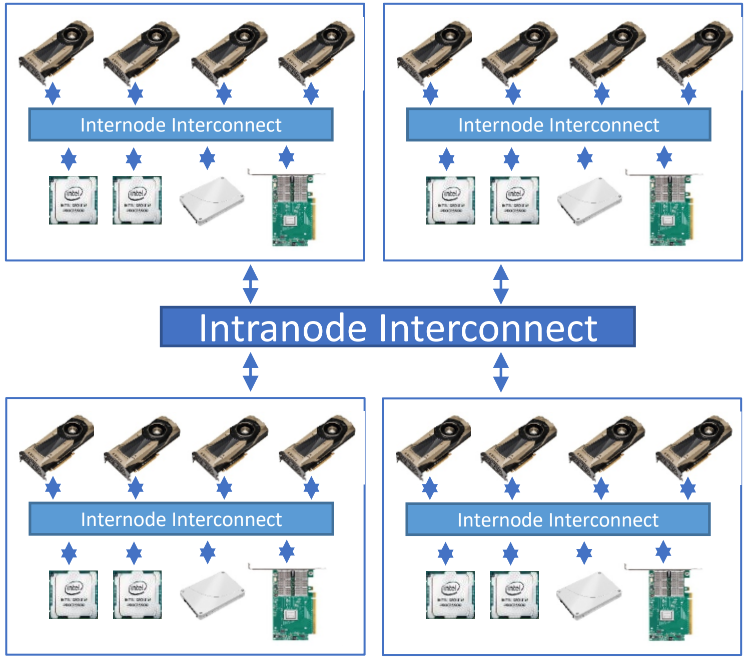
Distributed GPU Cluster

Billon-Scale Model Training - Scale Out Large Model Training

- Model parallelism (Megatron-LM)
  - Partition the model states vertically across multiple GPUs.

Require multiple GPUs resources

- ZeRO: Zero Redundancy Optimizer (ZeRO, SC'19)
  - Split the training batch across multiple GPUs without model states duplication.



Distributed GPU Cluster

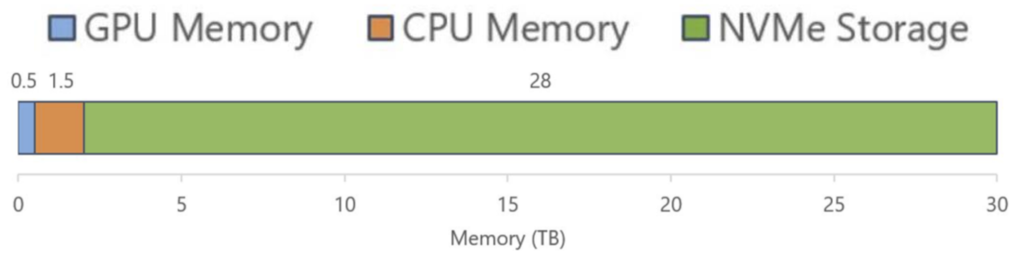
Beyond the GPU Memory

- Modern clusters have heterogeneous memory systems.

- GPU memory comprises a small fraction

- Can we extend an existing parallel training technology to use CPU/NVMe memory?

Memory available on a Single DGX-2 Node



ZeRO with CPU Offload

- Store optimizer states in CPU memory instead of GPU
- Send from CPU to GPU
- Broadcast or reduce as ZeRO

- Is CPU←→ GPU bandwidth sufficient?
  - Required bw for efficiency: 25-60 GB/s
  - PCIe peak bw on DGX-2: 32 GB/s

Offload Strategy

- ZeRO-Offload partitions the dataflow graph with:

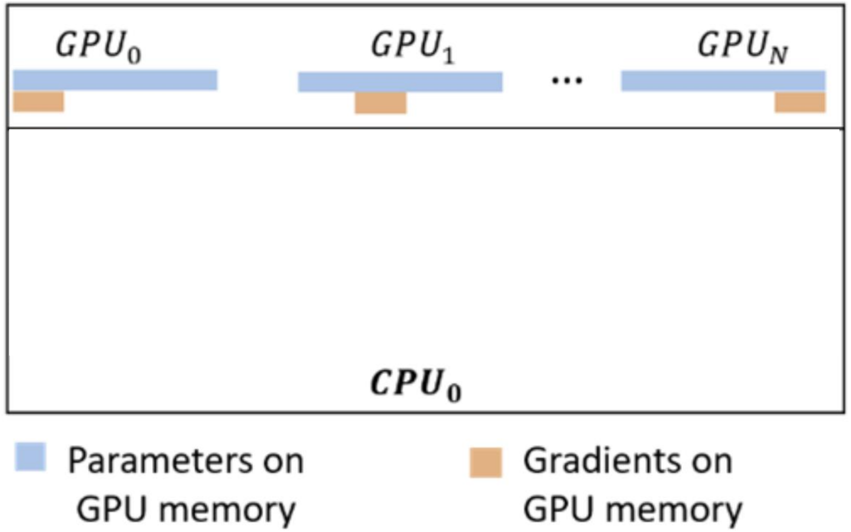
    i. Few computation on CPU

    ii. Minimize of communication volume

    iii. Maximize memory saving while achieving minimum communication volume

How Does ZeRO-Offload Work?

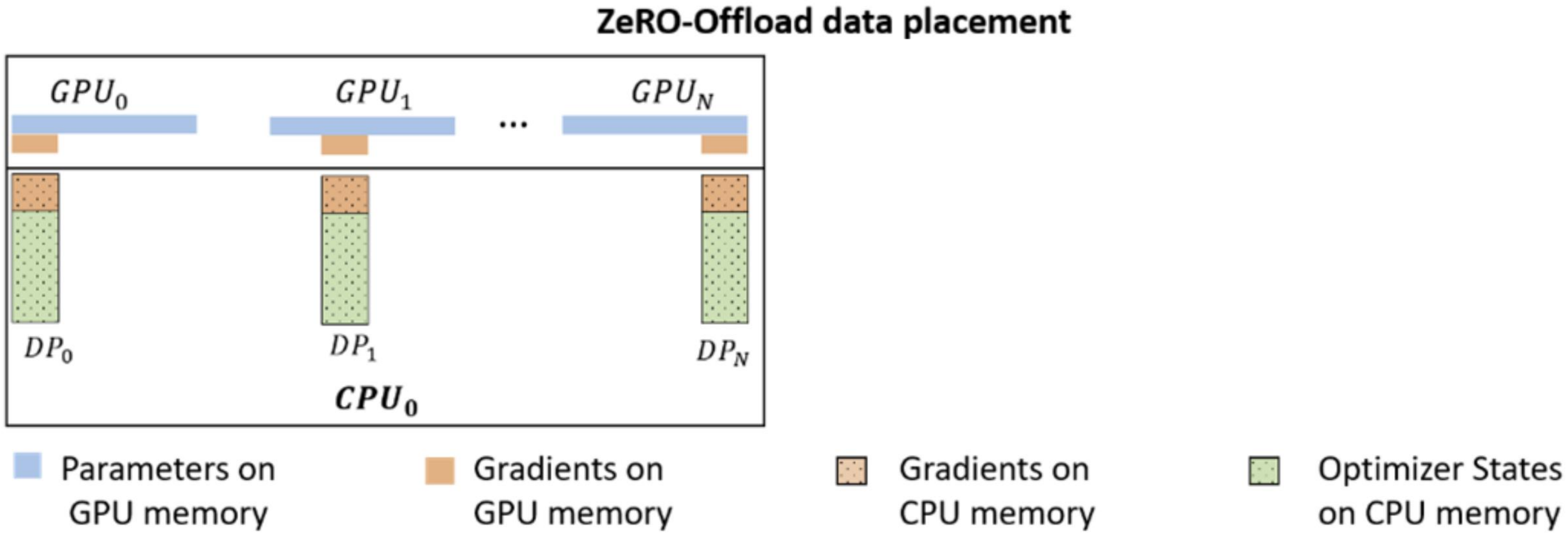
ZeRO-Offload data placement



GPU memory:

- FP16 weight parameters
- Partitioned gradients (ZeRO Stage 2)

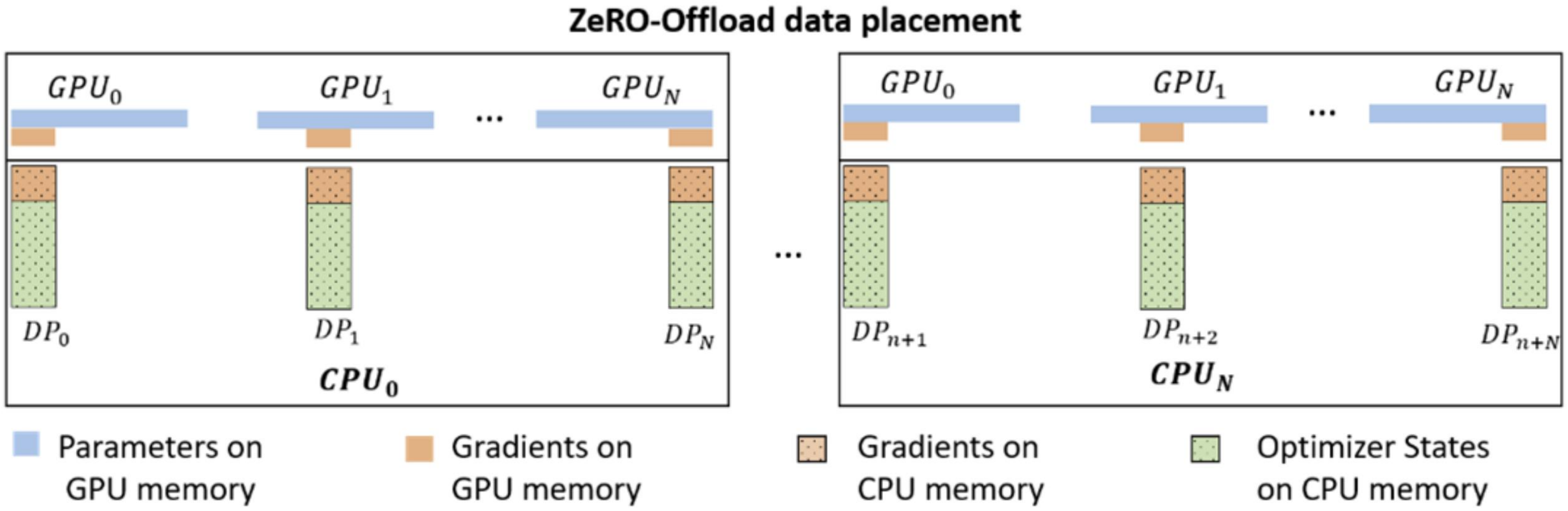
How Does ZeRO-Offload Work?



GPU memory:
- FP16 weight parameters
- Partitioned gradients (ZeRO Stage 2)

CPU memory:
- FP32 weight parameters
- Partitioned optimizer states
- Partitioned gradients

How Does ZeRO-Offload Work?



GPU memory:

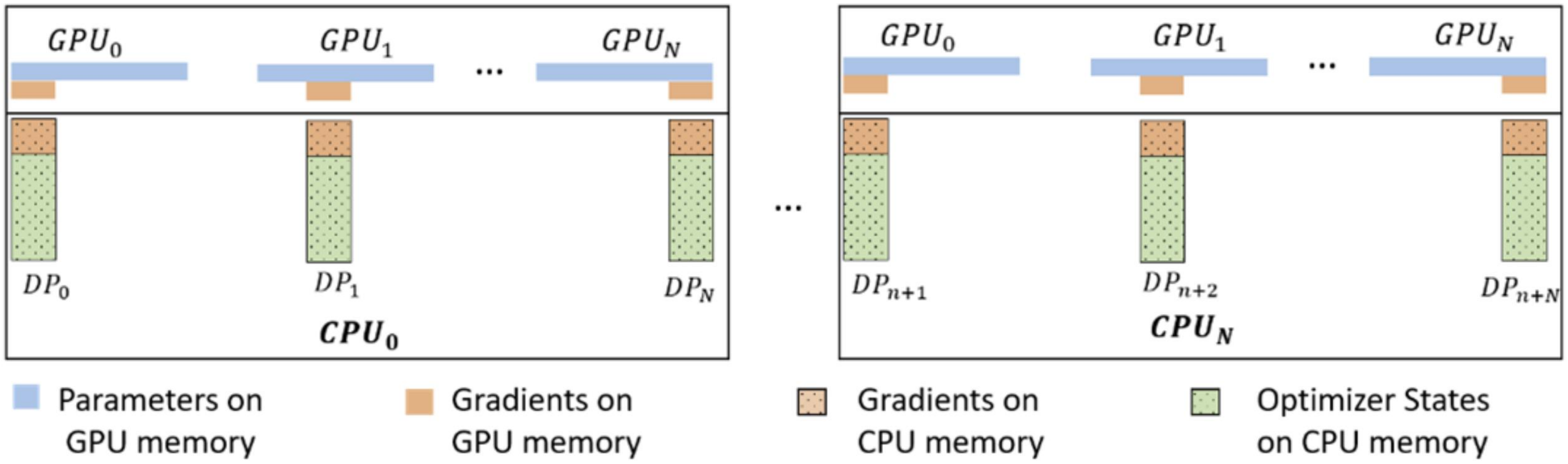
- FP16 weight parameters
- Partitioned gradients (ZeRO Stage 2)

CPU memory:

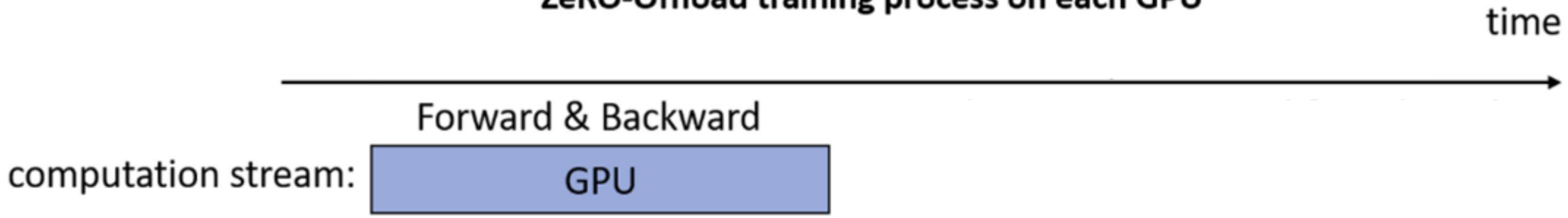
- FP32 weight parameters
- Partitioned optimizer states
- Partitioned gradients

How Does ZeRO-Offload Work?

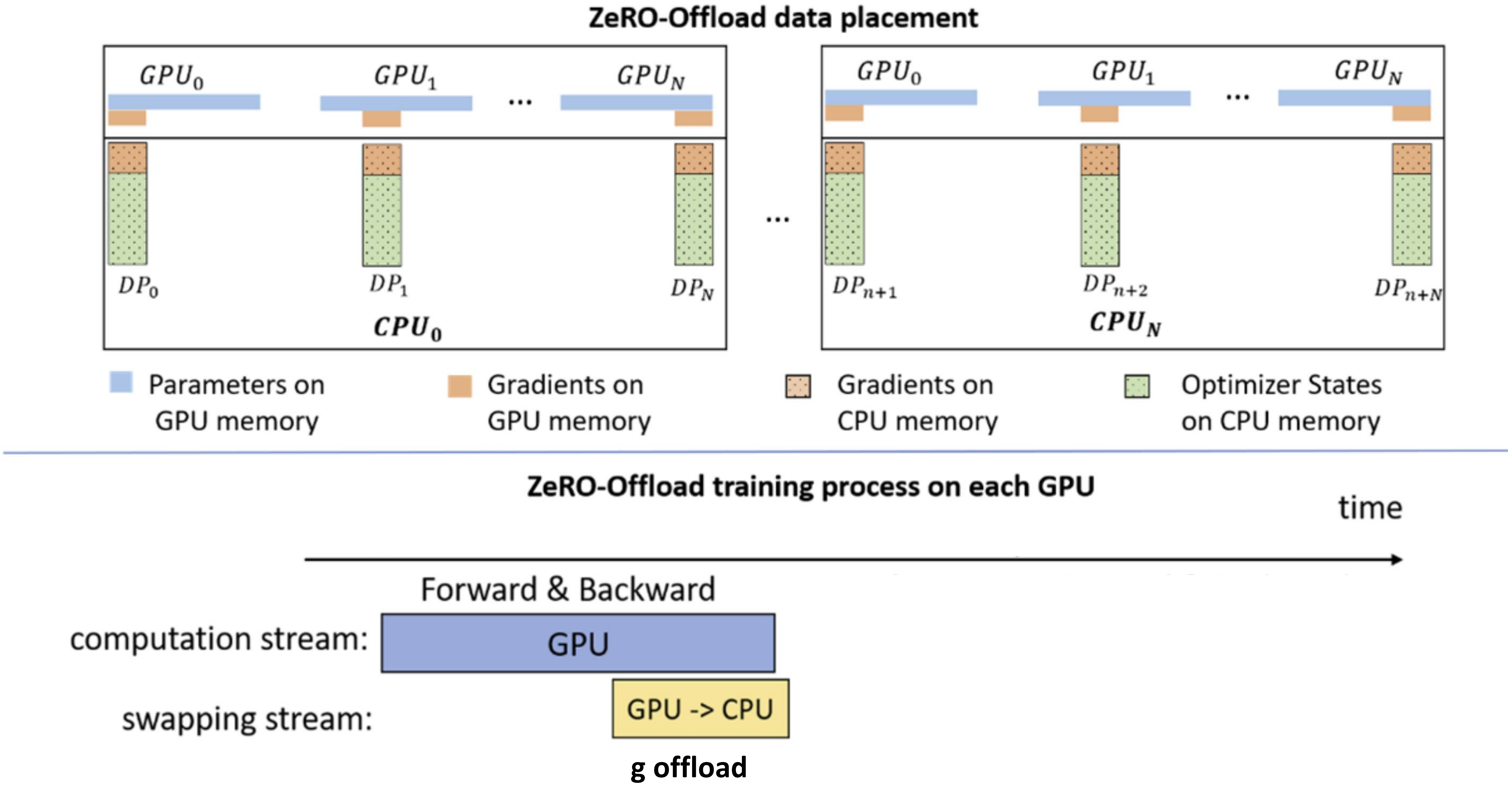
ZeRO-Offload data placement



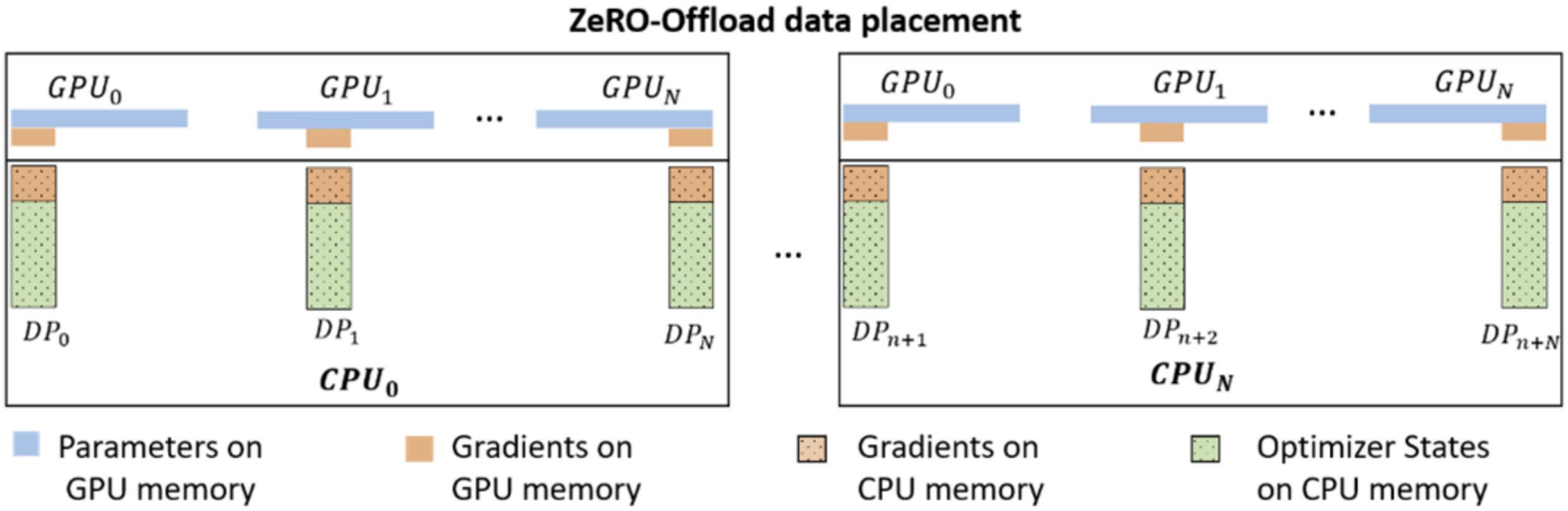
ZeRO-Offload training process on each GPU

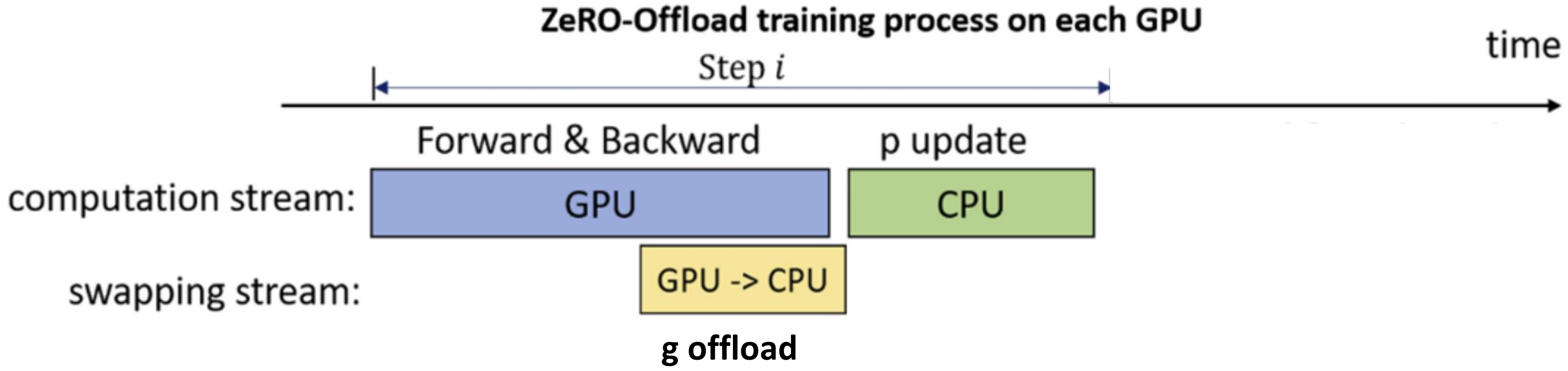


How Does ZeRO-Offload Work?

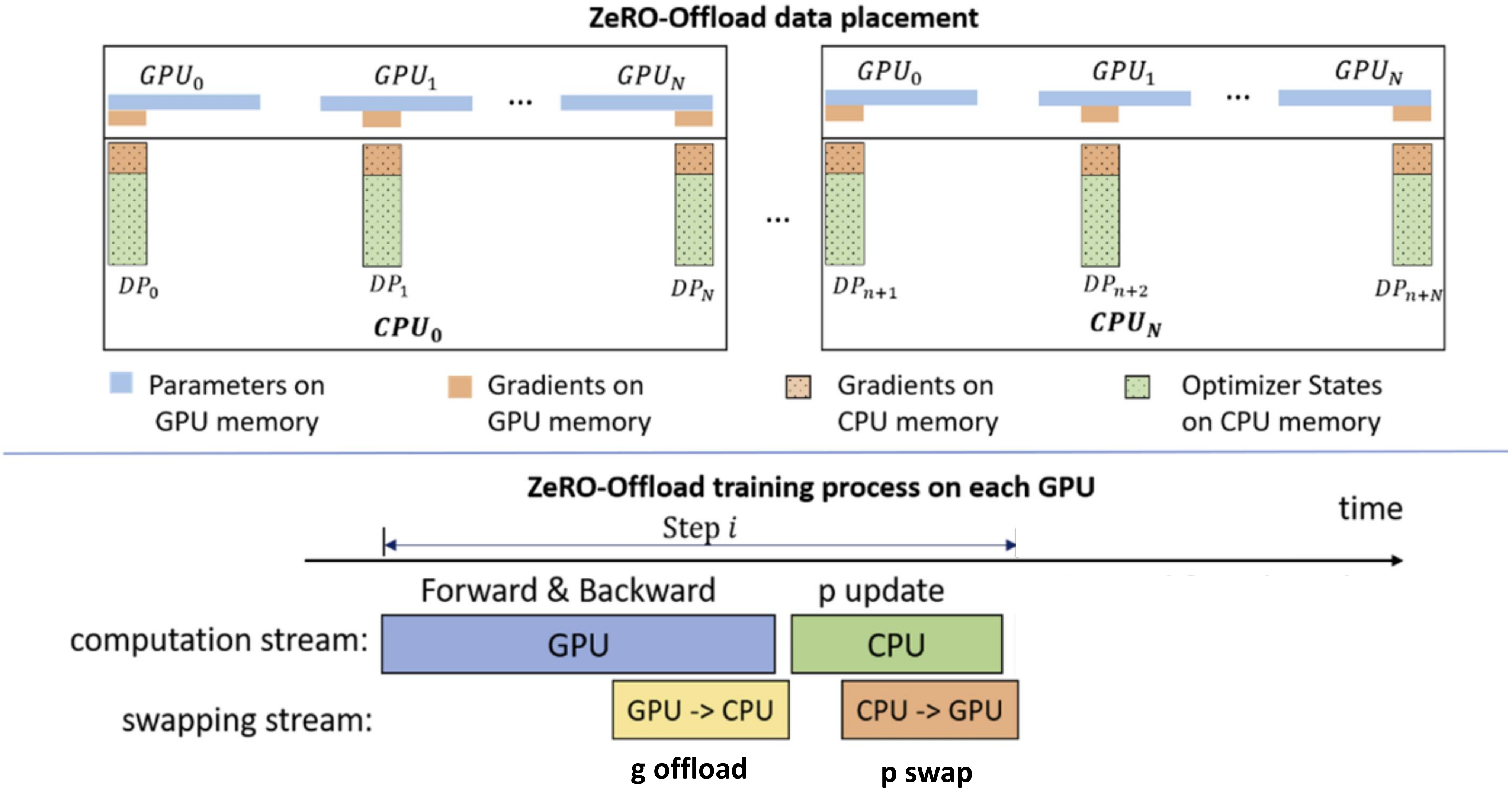


How Does ZeRO-Offload Work?

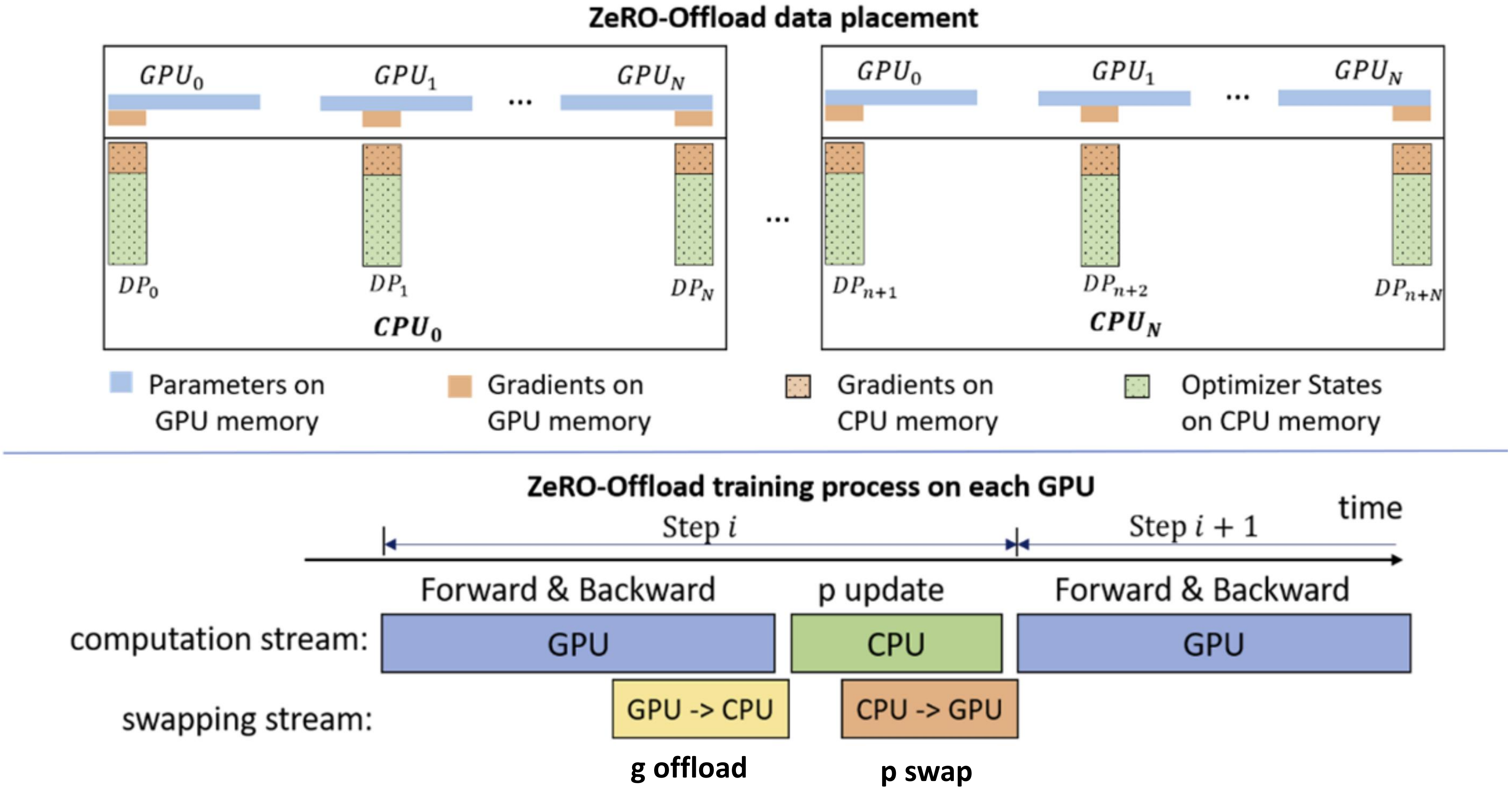




How Does ZeRO-Offload Work?



How Does ZeRO-Offload Work?



Optimized CPU Execution

- Highly parallelized CPU optimizer implementation

1) SIMD vector instruction for fully exploiting the hardware parallelism supported on CPU architectures.
2) Loop unrolling to increase instruction level parallelism.
3) OMP multithreading for effective utilization of multiple cores and threads on the CPU in parallel.

	Adam Optimizer: Parameter Update Latency (seconds)		
Model Size (B)	PyTorch-CPU	DeepSpeed-CPU	PyTorch-GPU
1	1.39	0.22	0.10
2	2.75	0.51	0.26
4	5.71	1.03	0.64
8	11.93	2.41	0.87
10	14.00	2.57	1.00

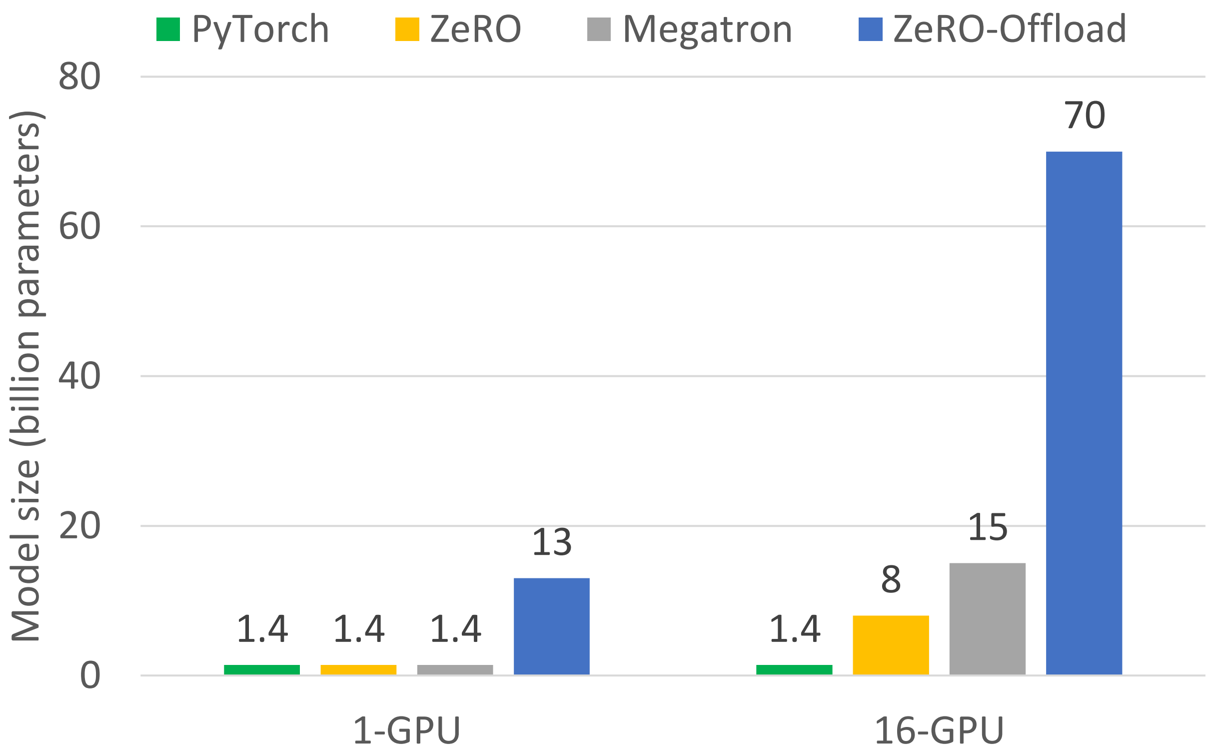
Evaluation

- Testbed

DGX-2 node	
GPU	16 NVIDIA Tesla V100 Tensor Core GPUs
GPU Memory	32GB HBM2 on each GPU
CPU	2 Intel Xeon Platinum 8168 Processors
CPU Memory	1.5TB 2666MHz DDR4
CPU cache	L1, L2, and L3 are 32K, 1M, and 33M, respectively
PCIe	bidirectional 32 GBps PCIe

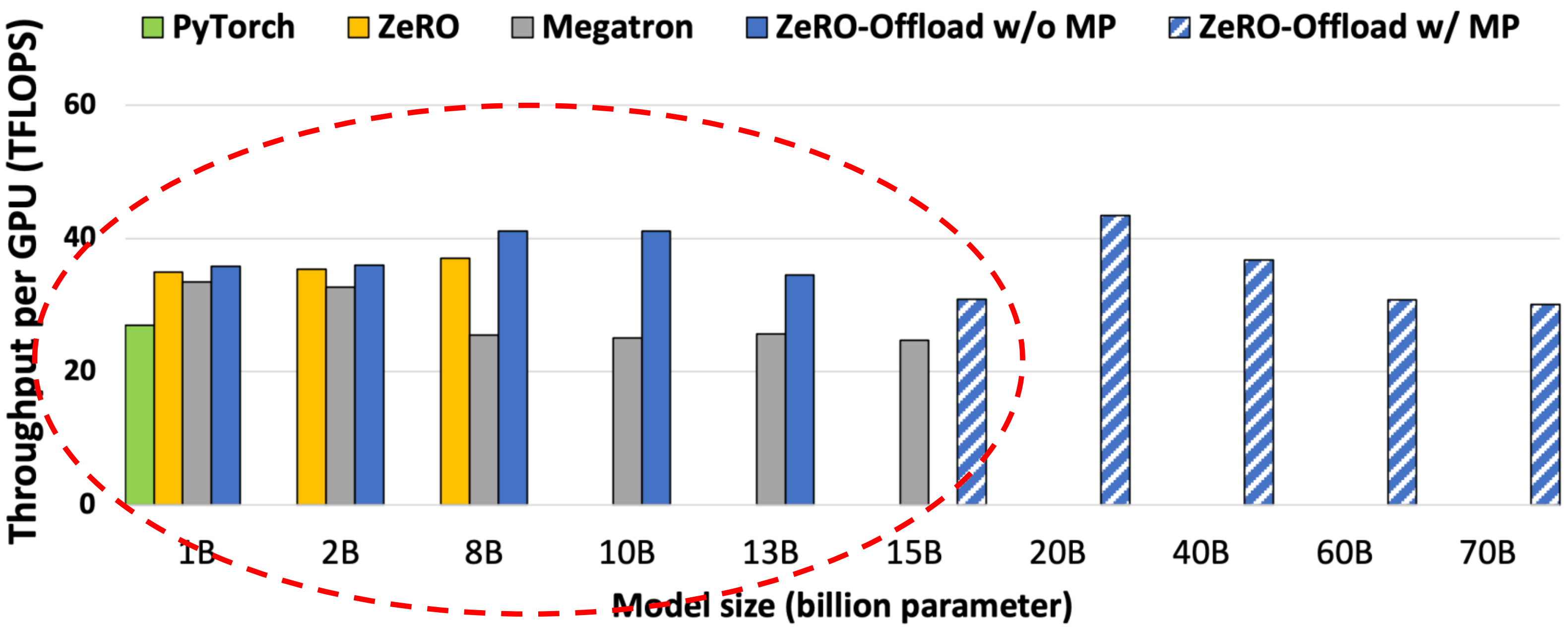
- Baselines
  1) Pytorch DDP: distributed data parallelism
  2) Megatron: model parallelism
  3) ZeRO: extended data parallelism by eliminating memory redundancies across multiple GPUs

Model Scale



ZeRO-Offload enables 13B model training on a single GPU, and easily enables training of up to 70B parameter with 16 GPUs.

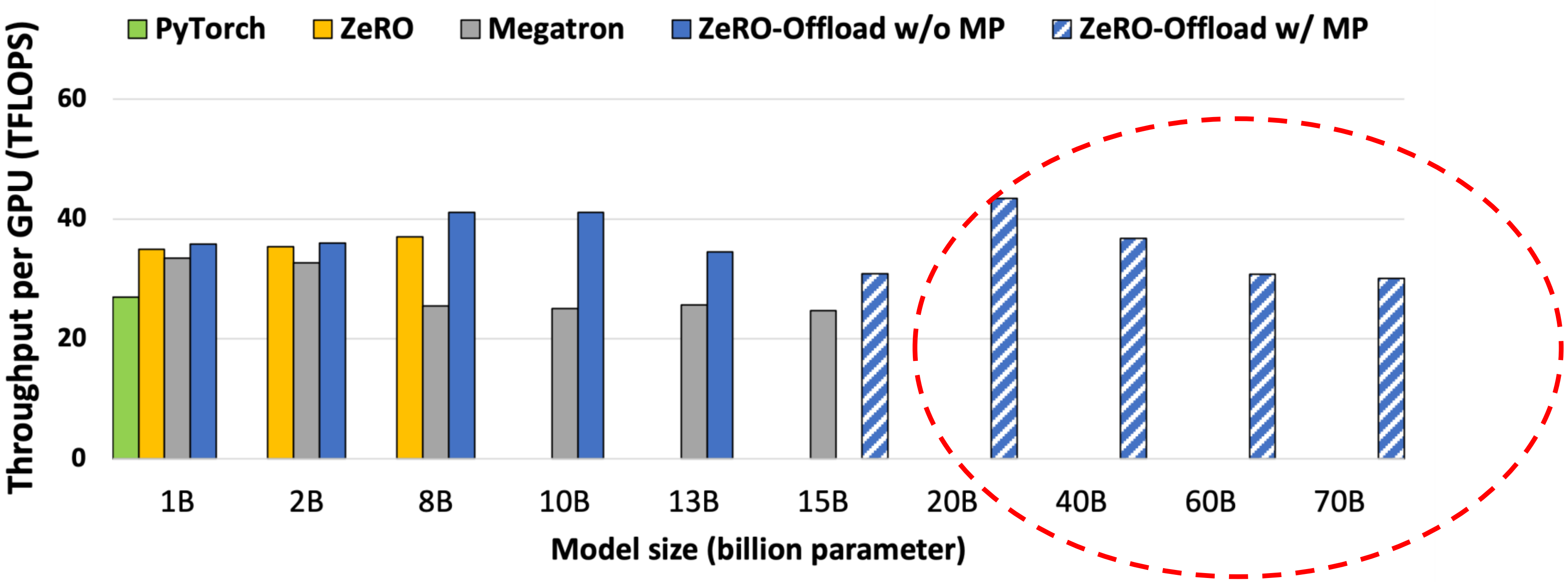
Training Throughput – Multiple GPUs



For 1B to 15B models, ZeRO-Offload achieves the highest throughput compared with PyTorch, ZeRO, and Megatron.

Single DGX-2 node (x16 V100-32GB)

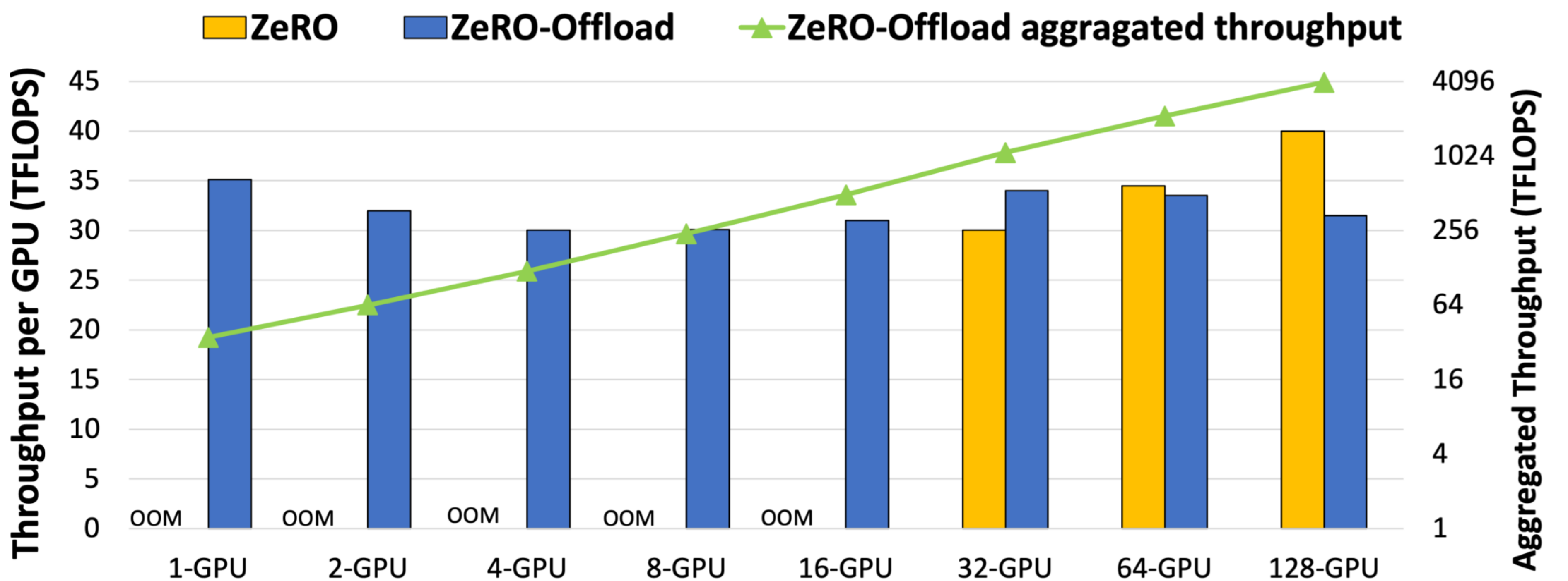
Training Throughput – Multiple GPUs



Combined with model parallelism, ZeRO-Offload enables training up to 70B parameter models with more than 30 TFLOPS throughput per GPU.

Single DGX-2 node (x16 V100-32GB)

Throughput Scalability



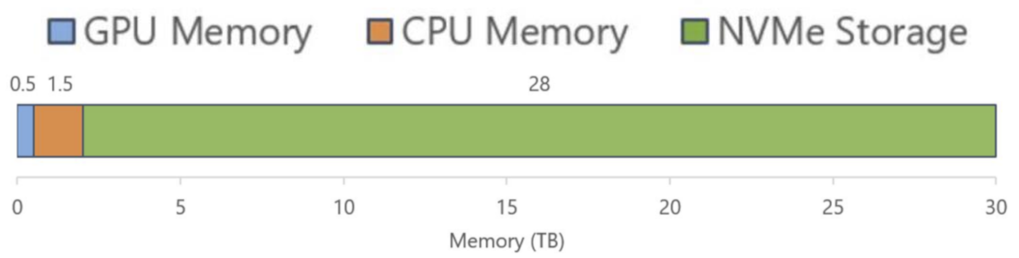
ZeRO-Offload achieves near perfect linear speedup in terms of aggregated throughput running at over 30 TFlops per GPU.

Leveraging NVMe

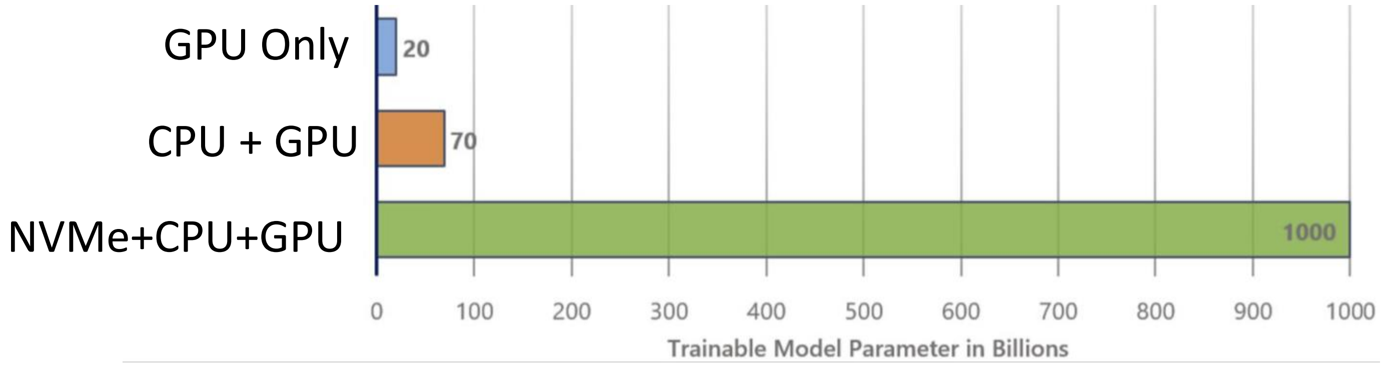
- Leverages GPU/CPU/NVMe memory (ZeRO-Infinity)
  - 1T params on a single node

- GPT-3 can be fine-tuned on a single node

Memory available on a Single DGX-2 Node



Model Size on a Single DGX-2 Node

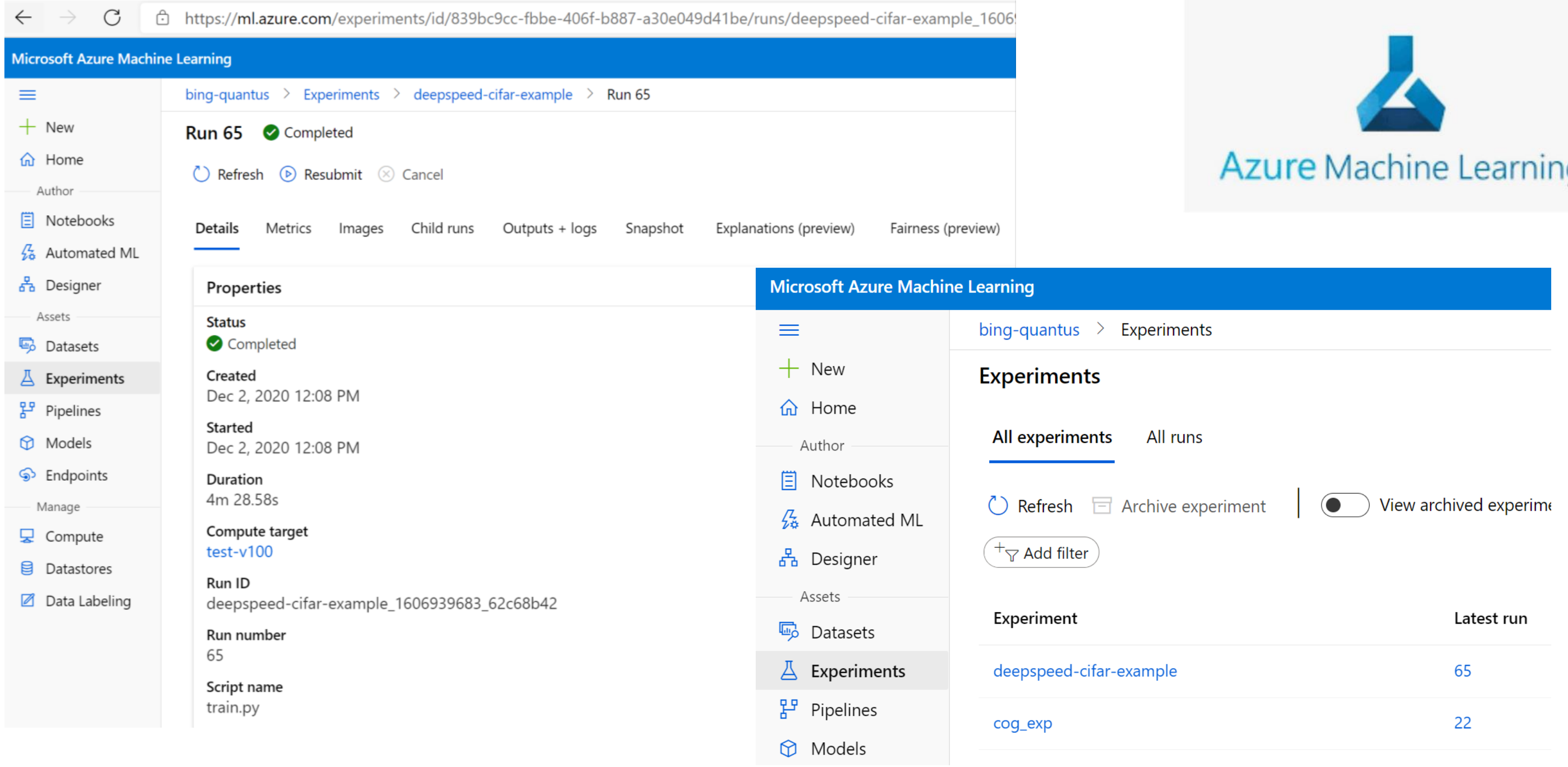


Software and Usability

How to install, extend, and use the DeepSpeed library?

DeepSpeed runs everywhere!

- AzureML, Azure VMs, …
- On premises hardware including support for both AMD and NVIDIA GPUs.



Azure Machine Learning

DeepSpeed is simple to use

DL Models

DL Optimizations (DeepSpeed)

DL Framework
(e.g., PyTorch, TensorFlow)

DL Infrastructure
(e.g., AML, Singularity, ITP, MPI-based platforms)

Hardware
(e.g., GPU/CPU clusters)

Bert - Original

```
# Construct distributed model
model = BertMultiTask(...)
model = DistributedDataParallel(model)

...

# Construct FP16 optimizer
optimizer = FusedAdam(model_parameters, ...)
optimizer = FP16_Optimizer(optimizer, ...)
```

```
# Forward pass
loss = model(batch)

# Backward pass
optimizer.backward(loss)

# Parameter update
optimizer.step()
```

Bert – w. DeepSpeed

```
# Construct Bert model
model = BertMultiTask(...)

# Wrap to get distributed model and FP16 optimizer
model, optimizer, _, _ = deepspeed.initialize(
    args=args,
    model=model,
    model_parameters=model_parameters,
    ...
)
```

```
# Forward pass
loss = model(batch)

# Backward pass
model.backward(loss)

# Paramter update
model.step()
```

Minimal code change

⬇

Efficiency + Effectiveness

⬇

Speed + Scale

Thank You!

deepspeed

We welcome contributions! Make your first pull request ☺

https://github.com/microsoft/DeepSpeed

www.deepspeed.ai

The DeepSpeed Library and Team Members

- An open-source library to optimize training and inference of DL models at scale
  - https://github.com/microsoft/DeepSpeed

- DeepSpeed team is comprised of researchers and engineers excited about large-scale ML/DL models and large-scale systems

Team members: Minjia Zhang, Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, Shaden Smith, Cheng Li, Conglong Li, Du Li, Elton Zheng, Ammar Ahmad Awan, Jefferey Zhu, Michael Wyatt, Zhewei Yao, Reza Yazdani Aminabadi, Xiaoxia Wu, and Yuxiong He