Microsoft

DeepSpeed

DL Training Optimization Library
Towards Speed & Scale

Microsoft DeepSpeed Team

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, Arash Ashari, Elton Zheng, Jing Zhao, Minjia Zhang, Niranjan Uma Naresh, Reza Yazdani Aminabadi, Shaden Smith, ***Yuxiong He***

Outline

- Overview
  - Why & What
  - Highlights of results / techniques
  - Software architecture
- How to use DeepSpeed
- Example 1: Turing NLG 17B
  - Result summary, key techniques (ZeRO, flexible combination of parallelism)
- Example 2: RScan
  - Result summary, key techniques (sparse gradients, advanced HP tuning)
- Upcoming features

DL Training: Challenges and Capability

Challenges

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

Desired Capability of DeepSpeed

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

DL Training Optimization: DeepSpeed

DL Models

Training Optimization (DeepSpeed)

Training Framework (e.g., PyTorch, Tensorflow)

Training Infrastructure (e.g., AML, DL workspace, 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

DeepSpeed: https://github.com/microsoft/DeepSpeed

Scale
- 100B parameter
- 10X bigger

Speed
- Up to 5X faster

Cost
- Up to 5X cheaper

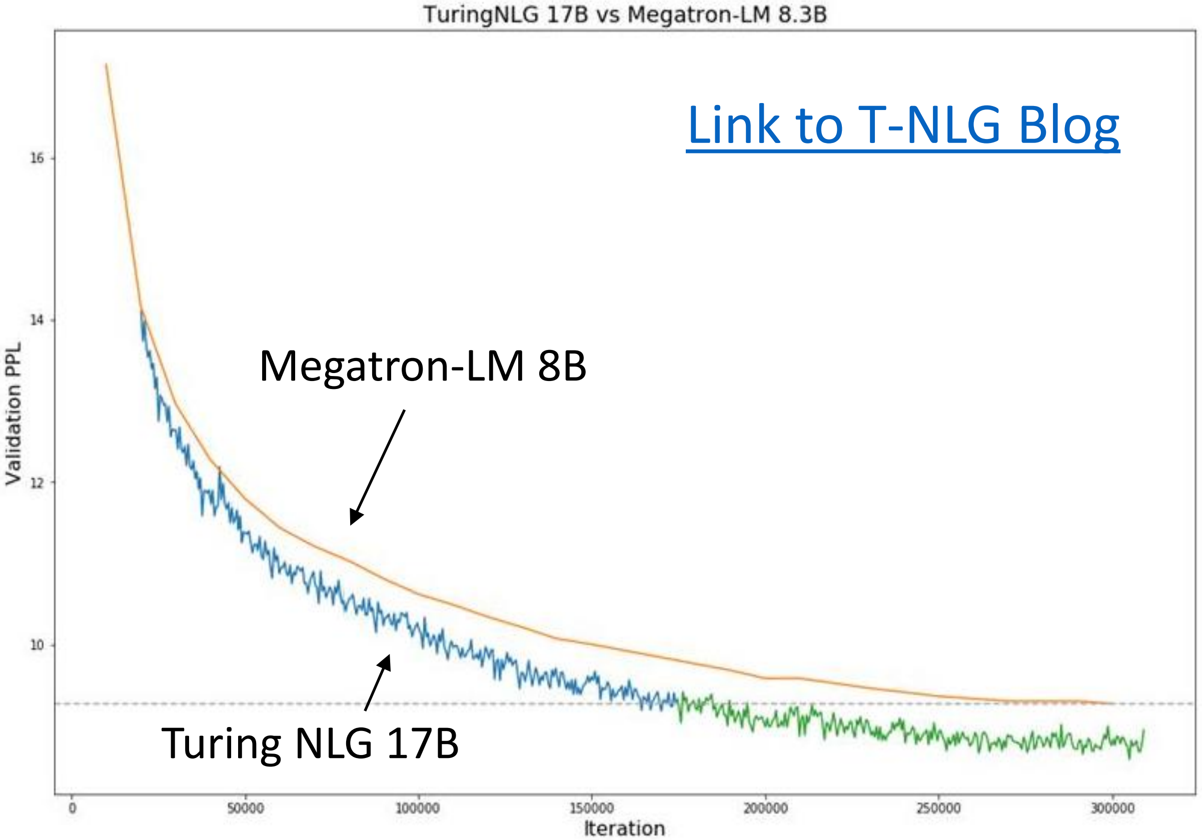
Usability
- Minimal code change

The State of Arts Models:

Nvidia Megatron 8B	Google T5 11B	**Microsoft Turing NLG 17B**

DeepSpeed System Capability:

ZeRO Stage 1 100B	**ZeRO Stage 2 200B**	**ZeRO Stage 3 1T**



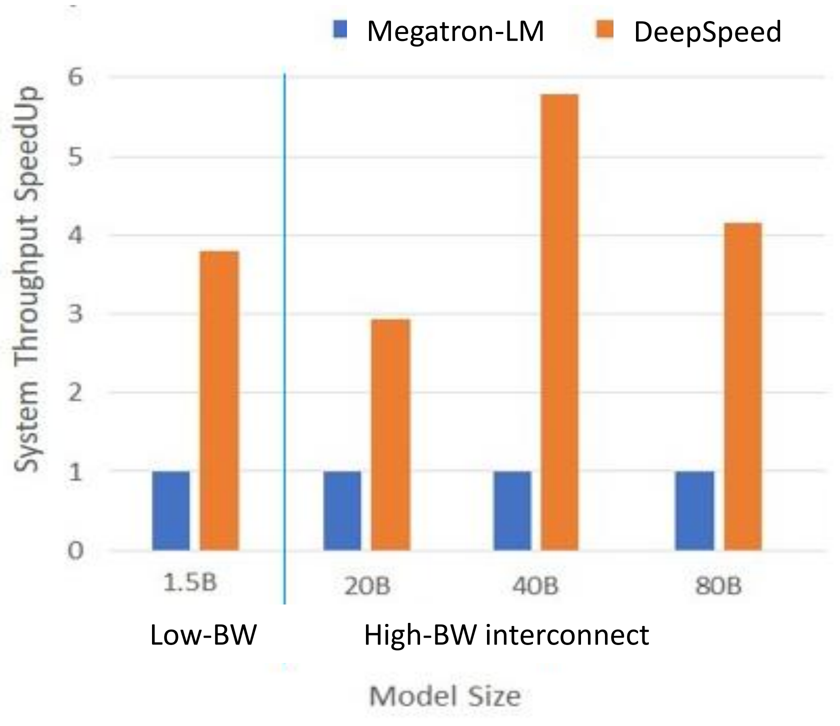
DeepSpeed: https://github.com/microsoft/DeepSpeed

Scale
- 100B parameter
- 10X bigger

Speed
- Up to 5X faster

Cost
- Up to 5X cheaper

Usability
- Minimal code change



Speed up of DeepSpeed over Megatron

Megatron-LM
- Tensor-slicing model parallelism

DeepSpeed
- ZeRO-powered data parallelism
- Tensor slicing of Megatron-LM

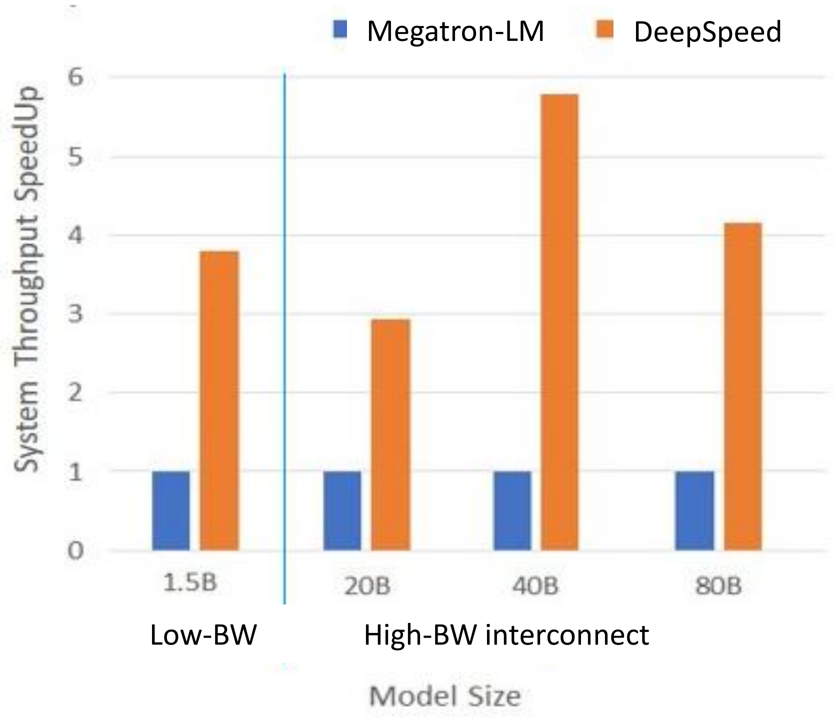
DeepSpeed: https://github.com/microsoft/DeepSpeed

Scale
- 100B parameter
- 10X bigger

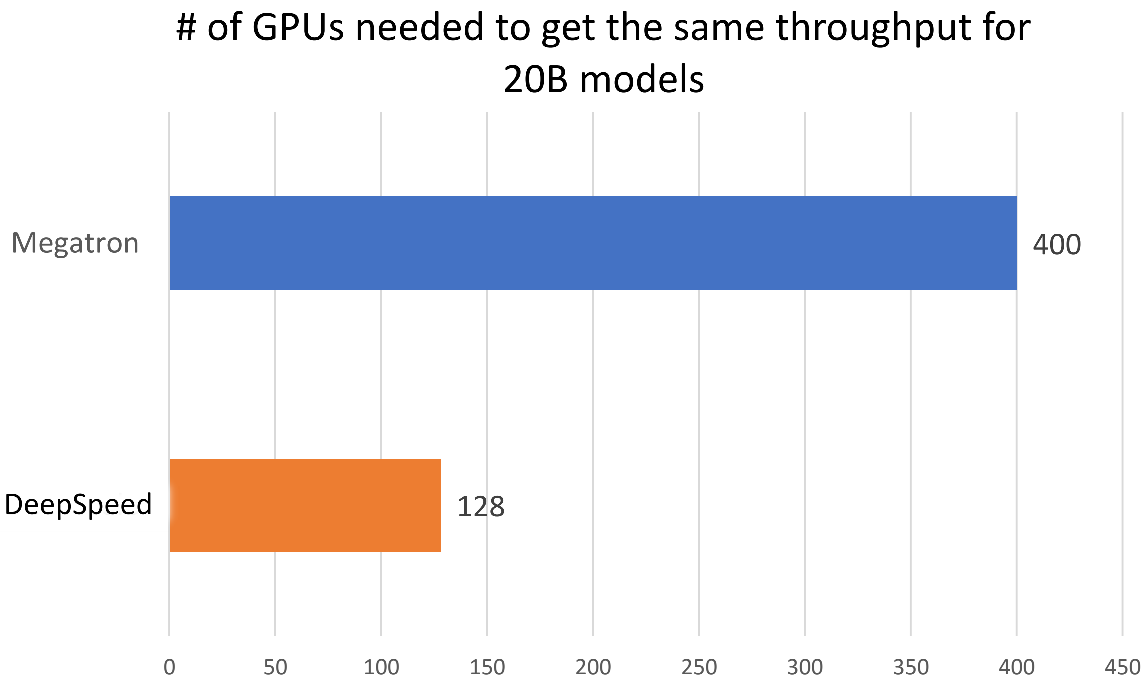
Speed
- Up to 5X faster

Cost
- Up to 5X cheaper

Usability
- Minimal code change



Speed up of DeepSpeed over Megatron



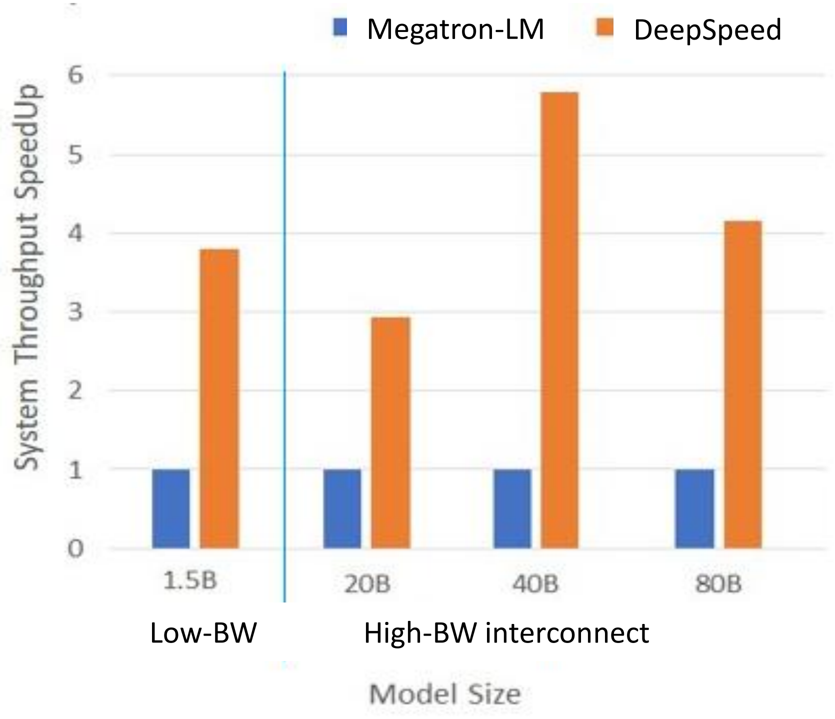
DeepSpeed: https://github.com/microsoft/DeepSpeed

Scale
- 100B parameter
- 10X bigger

Speed
- Up to 5X faster

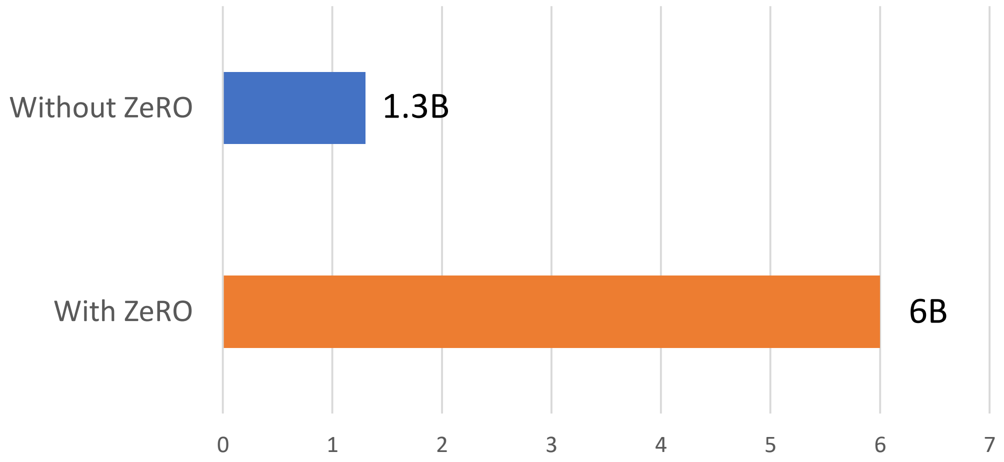
Cost
- Up to 5X cheaper

Usability
- Minimal code change



Speed up of DeepSpeed over Megatron

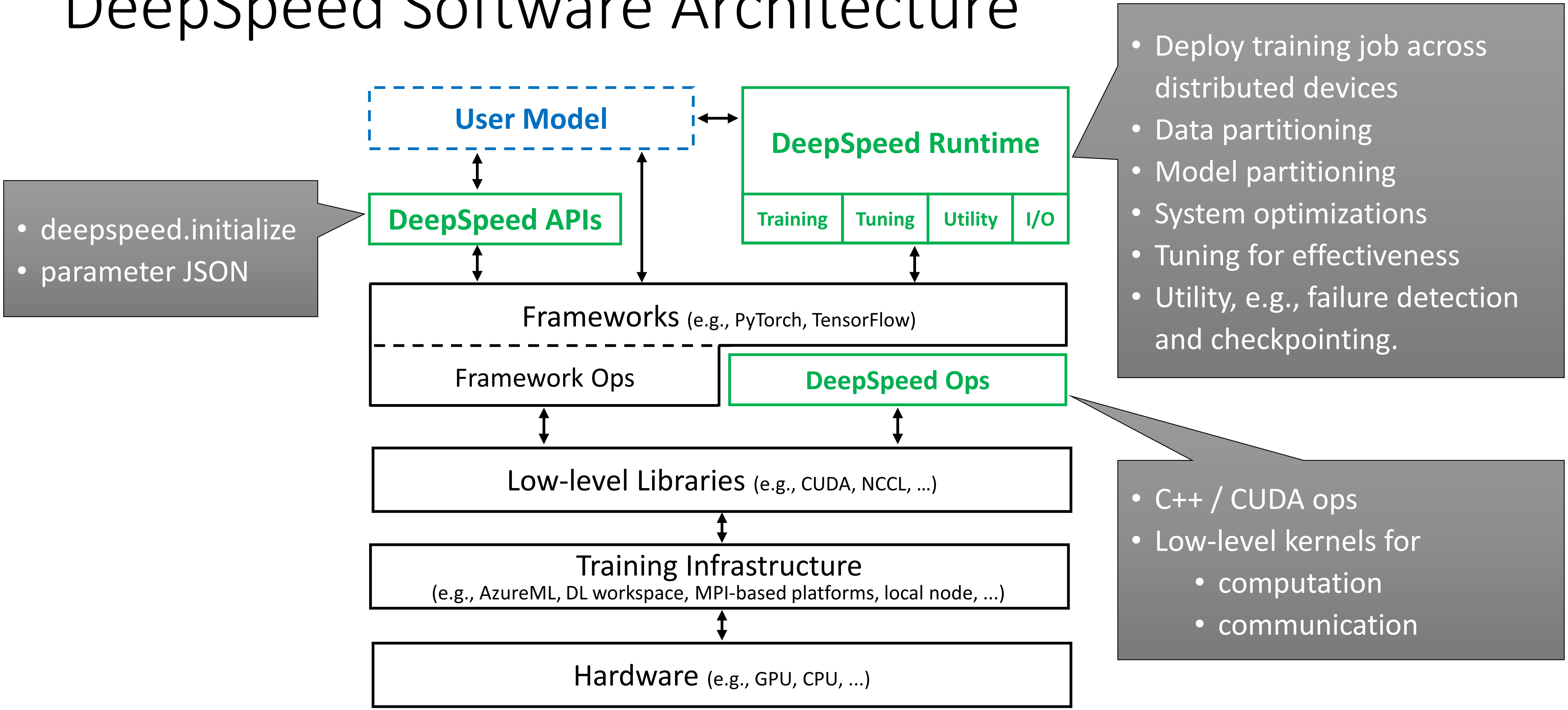
The largest model size (# of parameters) can be trained without model parallelism



Highlights of Techniques and Features

- Efficiency features
  - Memory: Zero Redundancy Optimizer (ZeRO)
  - Communication: sparse gradient
  - Compute: HPC kernels
  - Parallelism: ZeRO-powered data parallelism, flexible combination of data + model parallelism
- Effectiveness features
  - Adaptive hyperparameter tuning
  - Optimizers for large-batch training
- Usability features
  - Distributed training with mixed precision, gradient accumulation, etc.
  - IO: simplified data loader with auto batch creation
  - Training agnostic checkpoint / recompute
  - Performance profiling

DeepSpeed Software Architecture



1. Optimizations on top of training framework vs closely coupled
2. Agnostic of training infrastructure

Outline

- Overview
  - Why & What
  - Highlights of results / techniques
  - Software architecture
- How to use DeepSpeed
- Example 1: Turing NLG 17B
  - Result summary, key techniques (ZeRO, flexible combination of parallelism)
- Example 2: RScan
  - Result summary, key techniques (sparse gradients, advanced HP tuning)
- Upcoming features

DeepSpeed – PyTorch example

- Existing user code
  - Model written using torch.nn.Module
- User code modifications
  1. deepspeed.initialize(…) to wrap
     - Model (required)
     - Optimizer (optional)
     - LR scheduler (optional)
     - Dataset using torch.utils.data (optional)
  2. Add config_arguments to python argument parsing
     a)
     ```
     parser = deepspeed.add_config_arguments(parser)
     ```
  3. Use wrapped model for forward, backward, and parameter update

CLASS torch.nn.Module

Base class for all neural network modules.

Your models should also subclass this class.

Modules can also contain other Modules, allowing to nest them in a tree structure. Y
regular attributes:

```
import torch.nn as nn
import torch.nn.functional as F

class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5)
        self.conv2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        return F.relu(self.conv2(x))
```

DeepSpeed – PyTorch example

- Train model via:
  - *deepspeed*
    <client_entry.py>
    <client args>
    --deepspeed_config ds_config.json
  - Example
    - deepspeed cifar10.py
      --deepspeed_config ds_config.json

- Deploys training job across all distributed nodes
- Handles environment propagation, data partitioning
- Seamlessly incorporates all system efficiency optimizations

DeepSpeed Models: BERT

Before

```
# 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()
```

After

```
# 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()
```

For more details see: https://aka.ms/deepspeed-bert

DeepSpeed Models: BERT JSON config

- DeepSpeed parameters for BERT
  - Batch size config
  - Optimizer (e.g., Lamb)
  - Mixed precision

```
{
  "train_batch_size": 16384,
  "train_micro_batch_size_per_gpu": 64,

  "optimizer": {
    "type": "Lamb",
    "params": {
      "lr": 4e-3,
      "max_grad_norm": 1.0,
      "weight_decay": 0.01,
      "bias_correction": false,
      "max_coeff": 0.5,
      "min_coeff": 0.08
    }
  },

  "fp16": {
    "enabled": true,
    "loss_scale": 0
  }
}
```

For more details see: https://aka.ms/deepspeed-bert

DeepSpeed Models: GPT2

Before

```
# Construct FP16, distributed, GPT2 model
model = GPT2Model(num_layers=args.num_layers, ...)
model = FP16_Module(model)
model = DistributedDataParallel(model, ...)

...

# Construct FP16 Adam optimizer
optimizer = Adam(param_groups, ...)
optimizer = FP16_Optimizer(optimizer, ...)
```

```
# Forward pass
output = model(tokens, ...)

# Backward pass
optimizer.backward(loss)

# Parameter update
optimizer.step()
```

For more details see: https://aka.ms/deepspeed-gpt2

After

```
# Construct GPT2 model
model = GPT2Model(num_layers=args.num_layers, ...)

# Construct Adam optimizer
optimizer = Adam(param_groups, ...)

# Wrap model, optimizer, and lr scheduler
model, optimizer, lr_scheduler, _ = deepspeed.initialize(
    args=args,
    model=model,
    optimizer=optimizer,
    lr_scheduler=lr_scheduler,
    mpu=mpu
)
```

```
# Forward pass
output = model(tokens, ...)

# Backward pass
model.backward(loss)

# Parameter update
model.step()
```

DeepSpeed Models: GPT2 JSON config

- DeepSpeed parameters for BERT
  - Batch size config
  - Adam optimizer
  - Mixed precision
  - **Enable ZeRO optimization**

```json
{
  "train_batch_size": 512,
  "train_micro_batch_size_per_gpu": 16,

  "optimizer": {
    "type": "Adam",
    "params": {
      "lr": 0.00015,
      "max_grad_norm": 1.0
    }
  },

  "fp16": {
    "enabled": true,
    "loss_scale": 0,
    "loss_scale_window": 1000,
    "hysteresis": 2,
    "min_loss_scale": 1
  },

  "zero_optimization": true
}
```

For more details see: https://aka.ms/deepspeed-gpt2

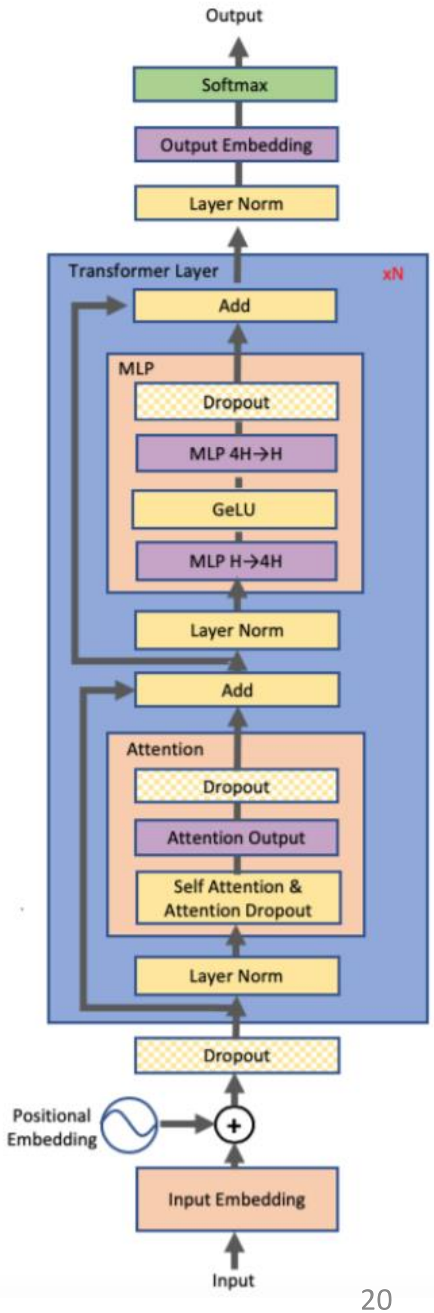
Outline

- Overview
  - Why & What
  - Highlights of results / techniques
  - Software architecture
- How to use DeepSpeed
- Example 1: Turing NLG 17B
  - Result summary, key techniques (ZeRO, flexible combination of parallelism)
- Example 2: RScan
  - Result summary, key techniques (sparse gradients, advanced HP tuning)
- Upcoming features

Example 1: Turing NLG 17B

- DeepSpeed + Megatron powered new state-of-the art LM
  - 10.21 perplexity
  - **Infeasible** to Train to **Practically Possible**
  - **Over 3x** throughput gain over Megatron alone

	Bert-Large	GPT-2	Turing 17.2 NLG
Parameters	0.32B	1.5B	17.2B
Layers	24	48	78
Hidden Dimension	1024	1600	4256
Relative Computation	1x	10x	**112x**
Memory Footprint	5.12GB	24GB	**275GB**

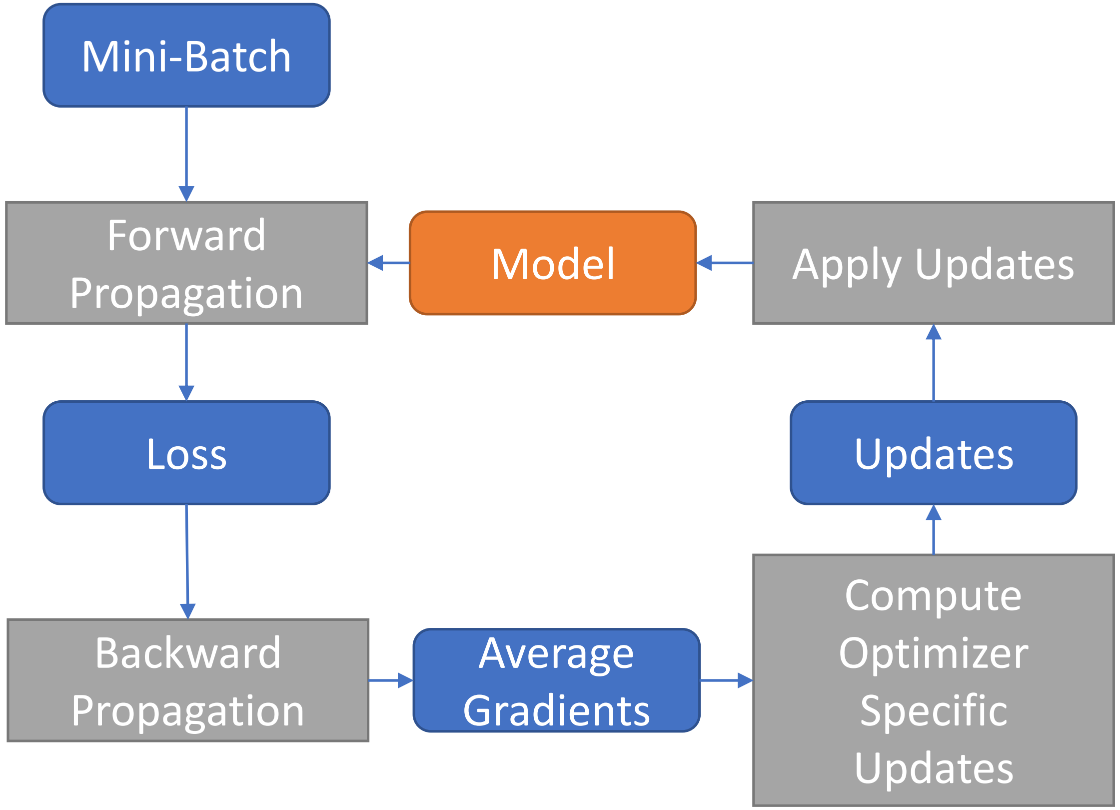


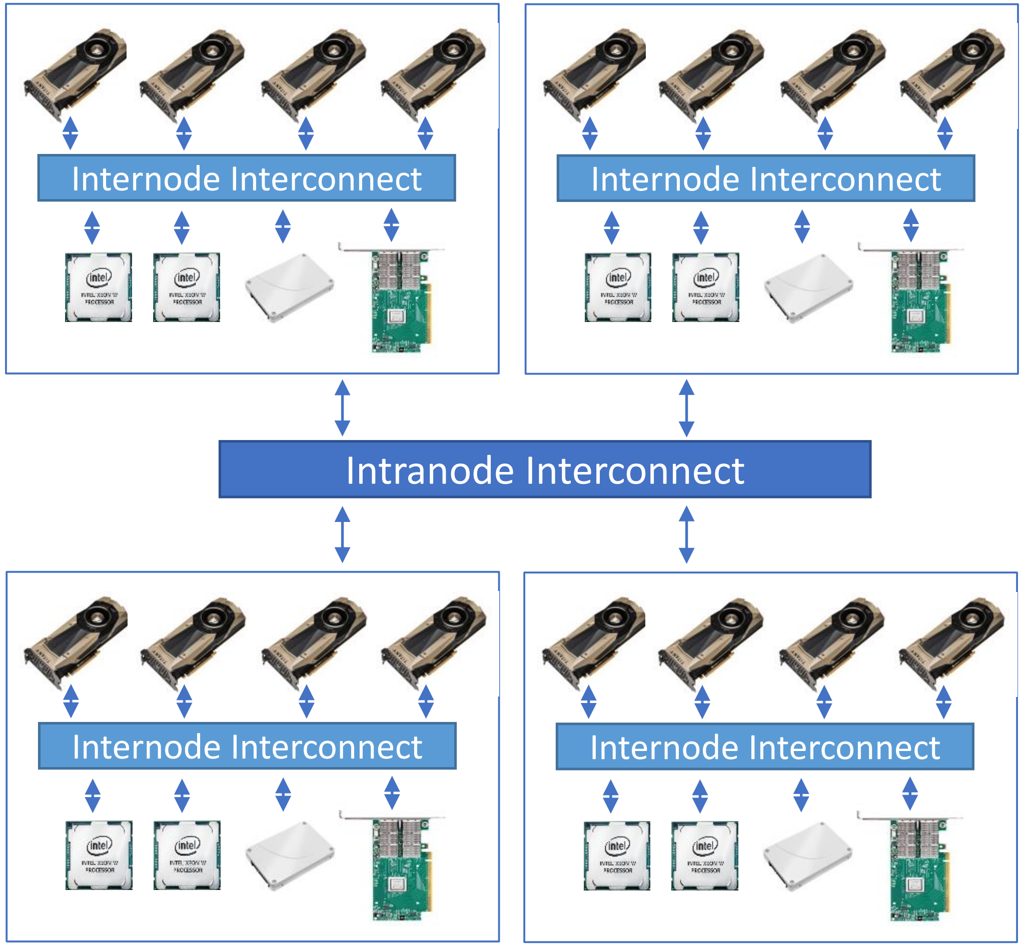
Example 1: Turing NLG 17B

- Zero Redundancy Optimizer (ZeRO) in DeepSpeed
- Less model parallelism and more batch size
- Over 3x throughput gain over Megatron alone on DGX-2 cluster

	With Megatron	With ZeRO and Megatron
Model Parallelism	16	4
Batch Size per Node	8	32
#GPUs (for batch 512)	1024	256
Throughput per GPU	9 Tflops	28 TFlops
Status	**Infeasible to Train**	**Possible to Train**

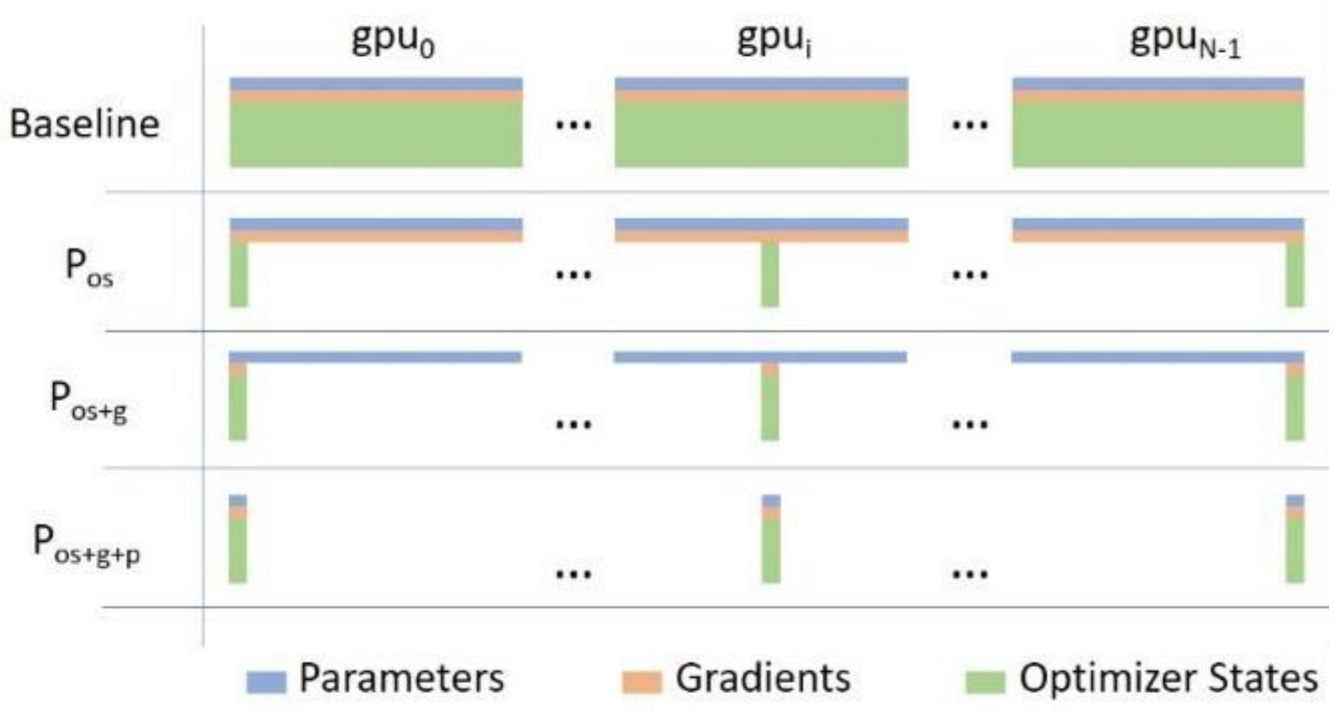
Distributed Data Parallel Training Overview





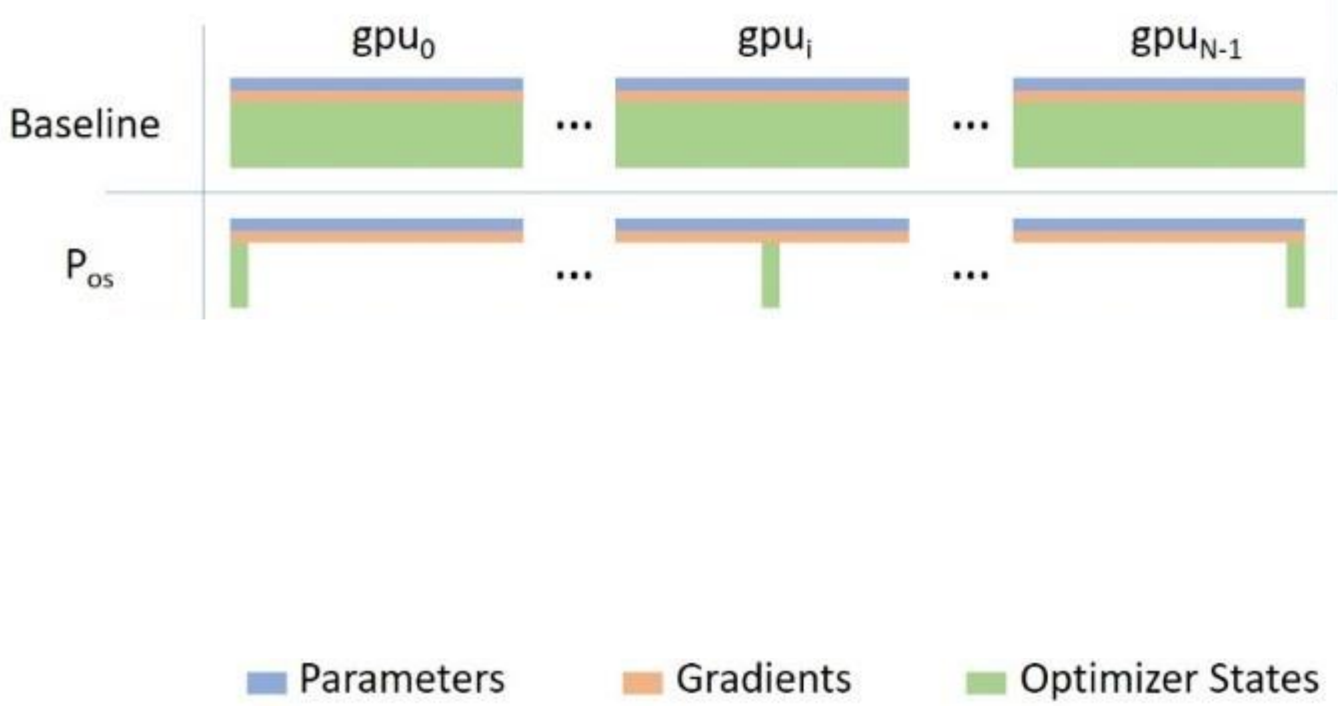
ZeRO: Zero Redundancy Optimizer

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



ZeRO: Zero Redundancy Optimizer

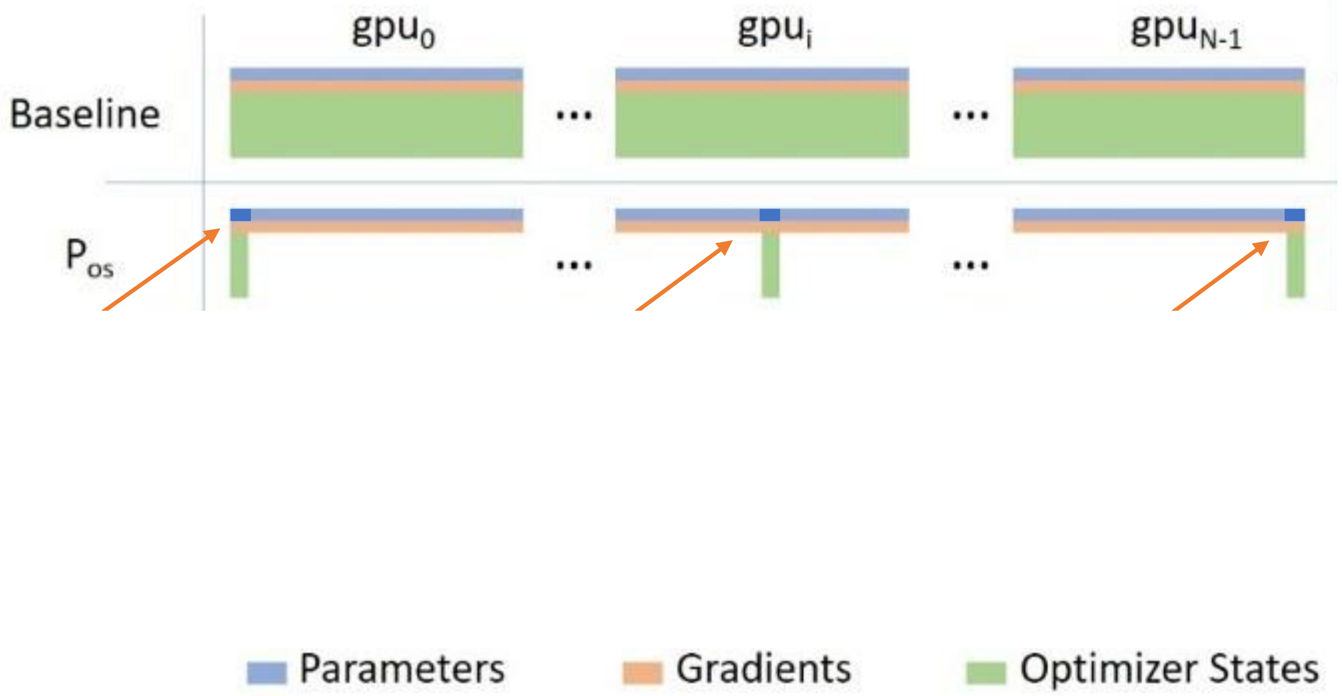
- ZeRO removes the redundancy across data parallel process
- Partitioning optimizer states, gradients and parameters (3 stages)



- **Stage 1**
- Compute gradients on different data
- Average Gradients

ZeRO: Zero Redundancy Optimizer

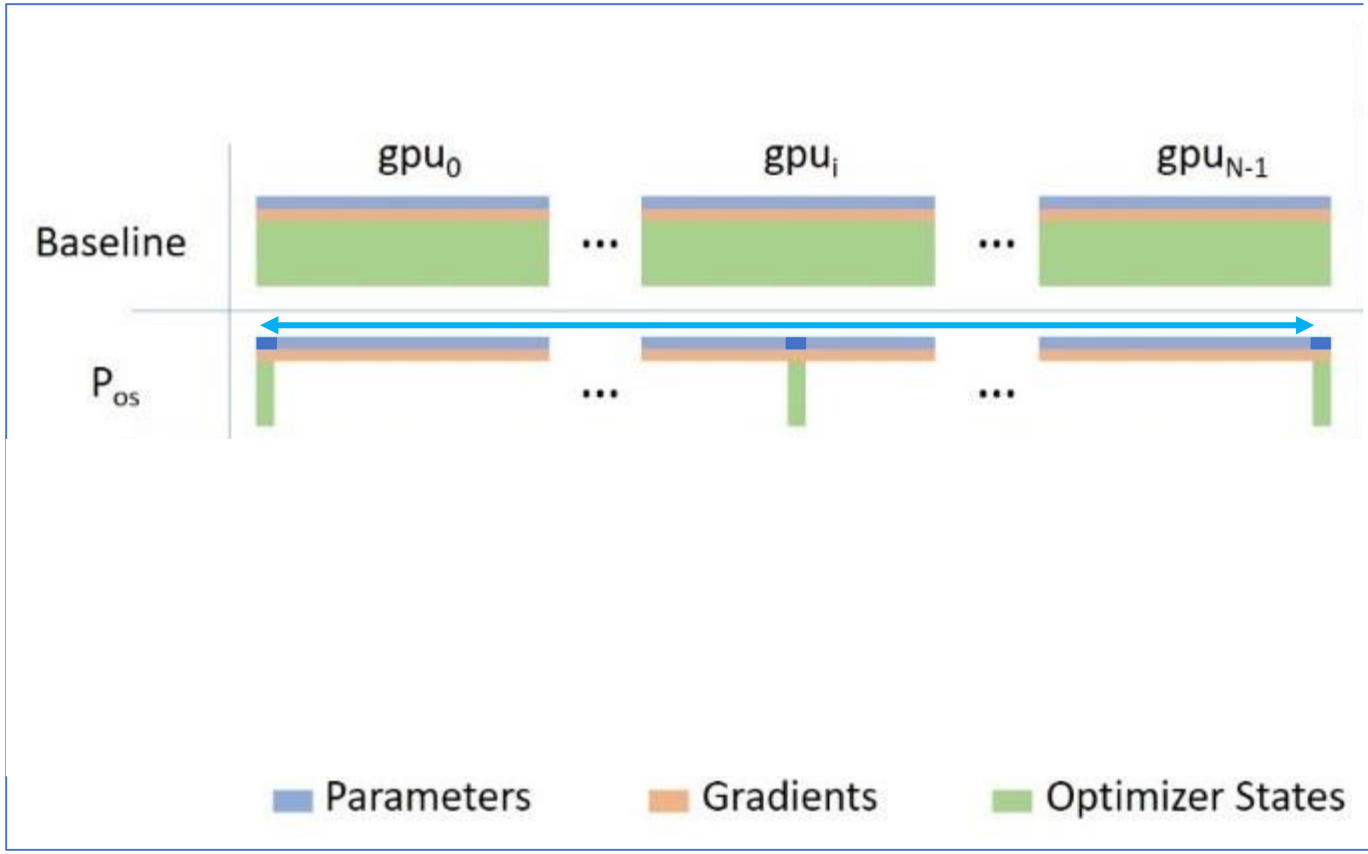
- ZeRO removes the redundancy across data parallel process
- Partitioning optimizer states, gradients and parameters (3 stages)



- **Stage 1**
- Compute gradients on different data
- Average Gradients
- Each GPU update only the parameter it owns

ZeRO: Zero Redundancy Optimizer

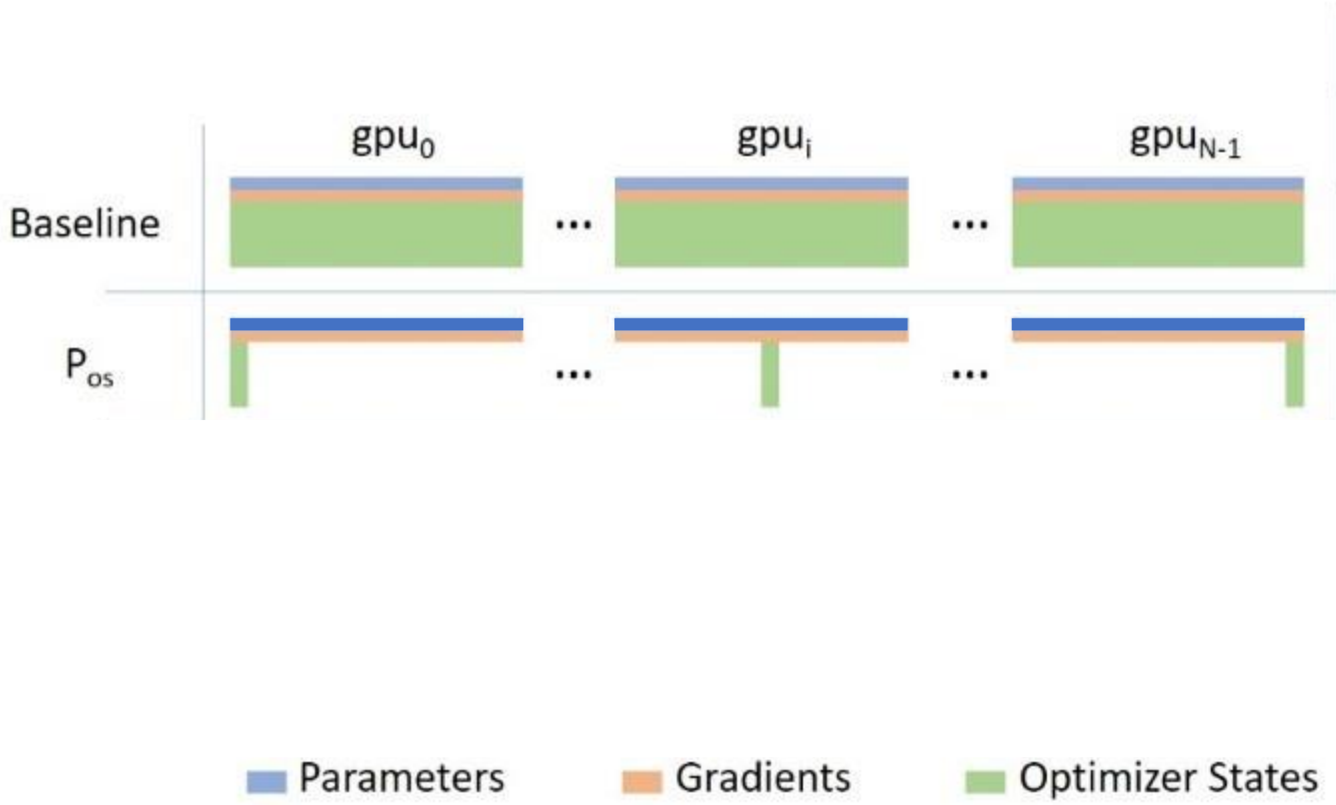
- ZeRO removes the redundancy across data parallel process
- Partitioning optimizer states, gradients and parameters (3 stages)



- **Stage 1**
- Compute gradients on different data
- Average Gradients
- Each GPU update only the parameter it owns
- All-gather the parameters

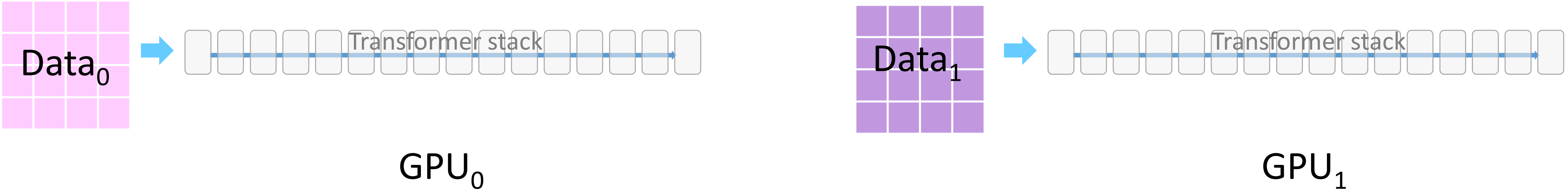
ZeRO: Zero Redundancy Optimizer

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



- **Stage 1**
- Compute gradients on different data
- Average Gradients
- Each GPU update only the parameter it owns
- All-gather the parameters

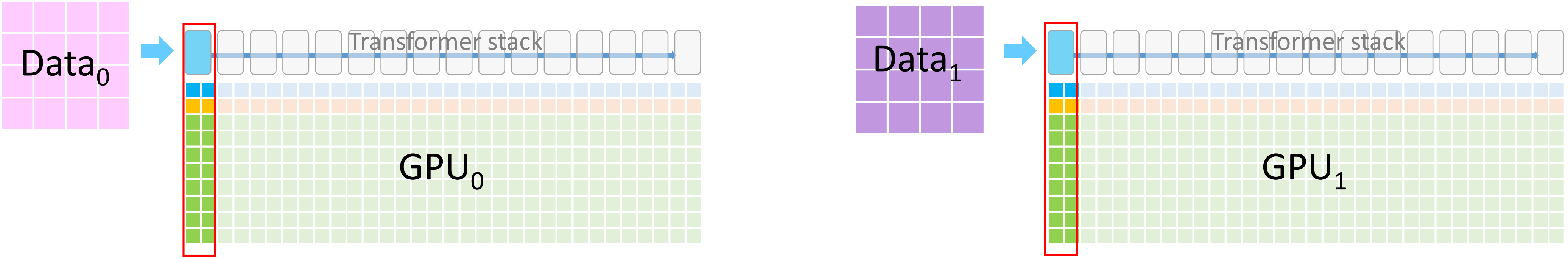
ZeRO: Zero Redundancy Optimizer



A 16-layer transformer model like or Turning NLR or $\text{BERT}_{\text{large}}$ is shown.

= 1 layer

ZeRO: Zero Redundancy Optimizer

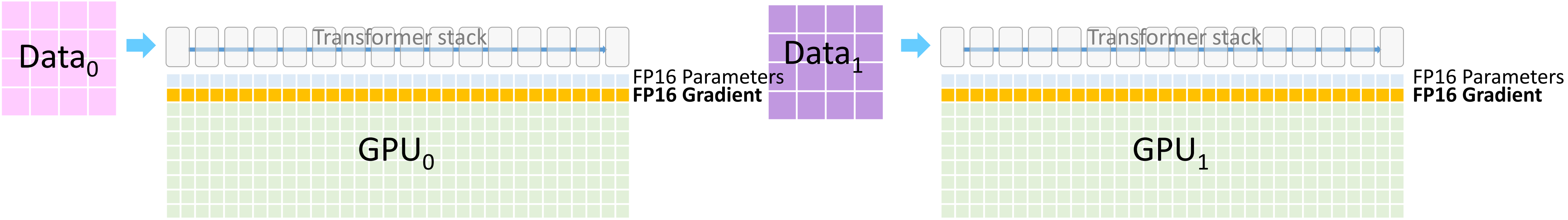


Each cell represents GPU memory used by its corresponding transformer layer

ZeRO: Zero Redundancy Optimizer

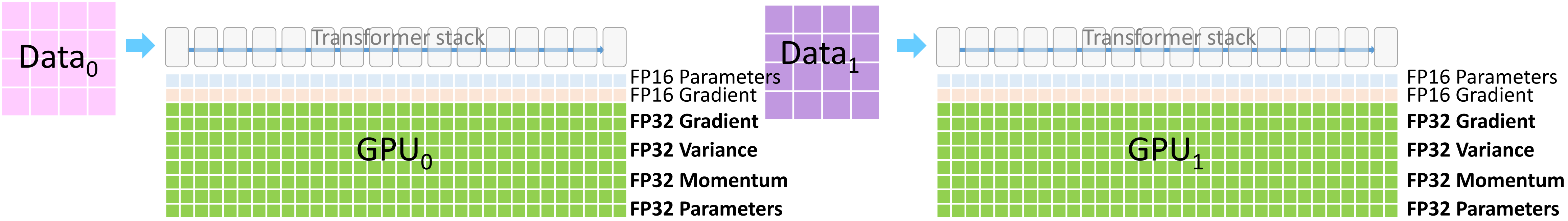
- FP16 parameter

ZeRO: Zero Redundancy Optimizer



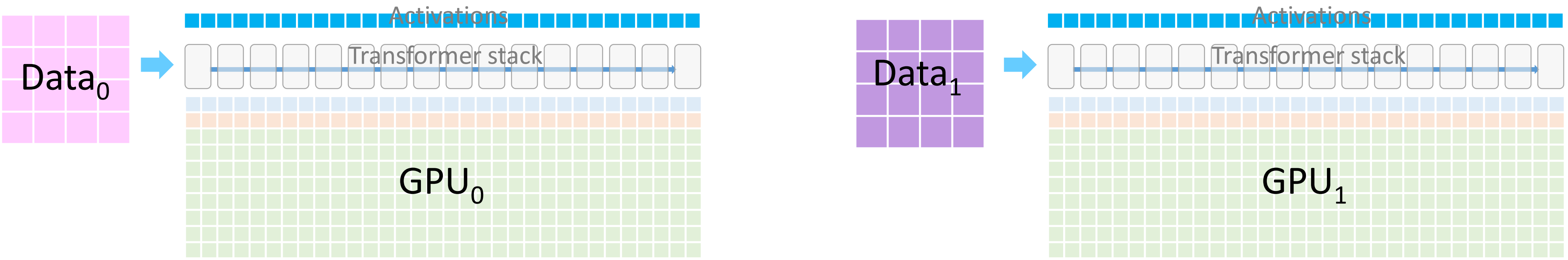
- FP16 parameter
- FP16 Gradients

ZeRO: Zero Redundancy Optimizer



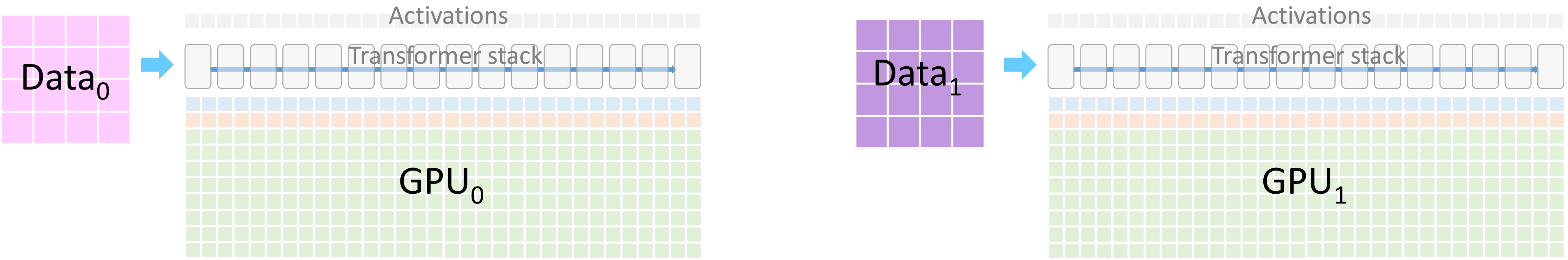
- FP16 parameter
- FP16 Gradients
- FP32 Optimizer States
    - Gradients, Variance, Momentum Parameters

ZeRO: Zero Redundancy Optimizer



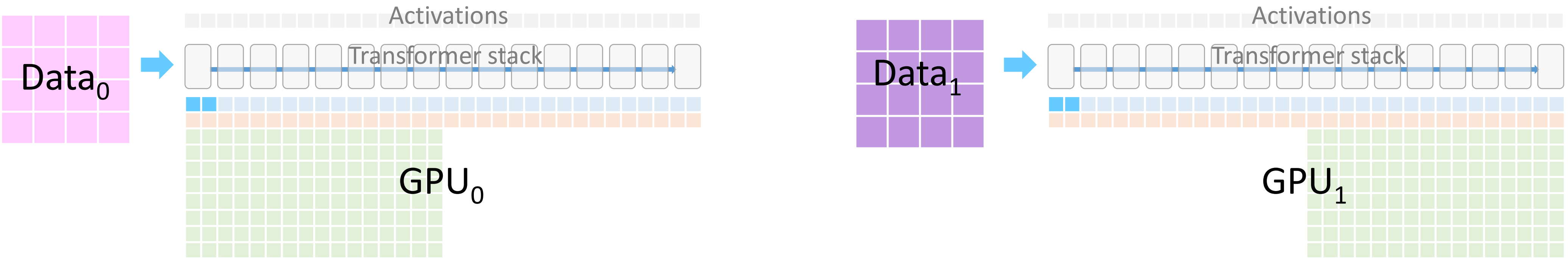
- FP16 parameter
- FP16 Gradients
- FP32 Optimizer States
  - Gradients, Variance, Momentum Parameters
- FP16 Activations

ZeRO: Zero Redundancy Optimizer



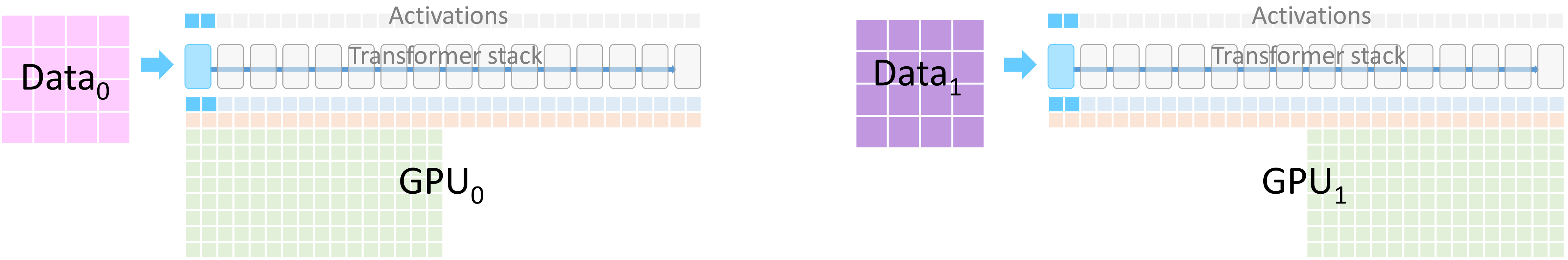
- ZeRO Stage 1

ZeRO: Zero Redundancy Optimizer



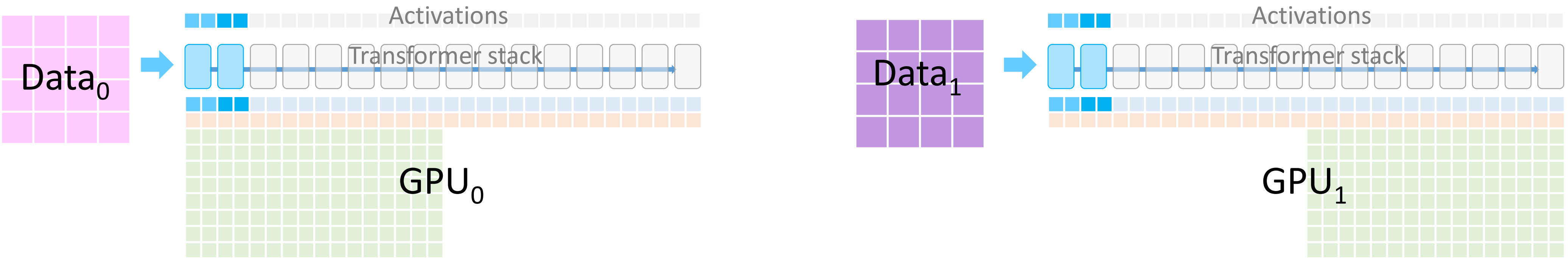
- ZeRO Stage 1
- Partitions optimizer states across GPUs

ZeRO: Zero Redundancy Optimizer



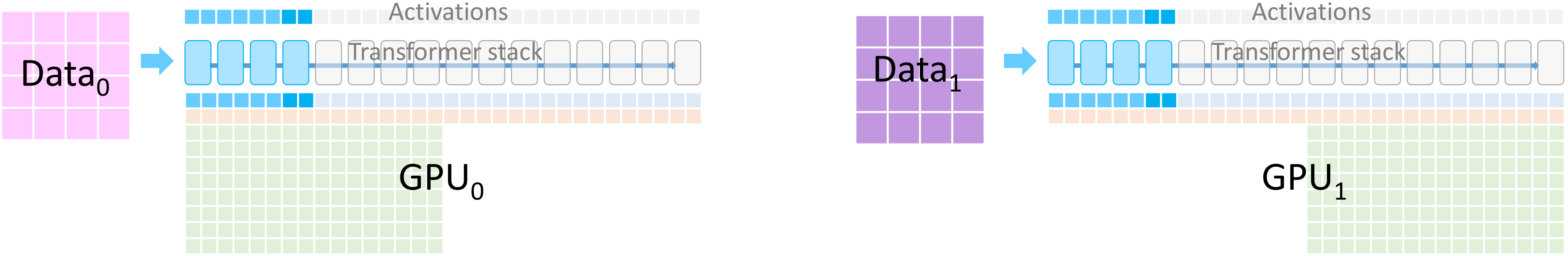
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks

ZeRO: Zero Redundancy Optimizer



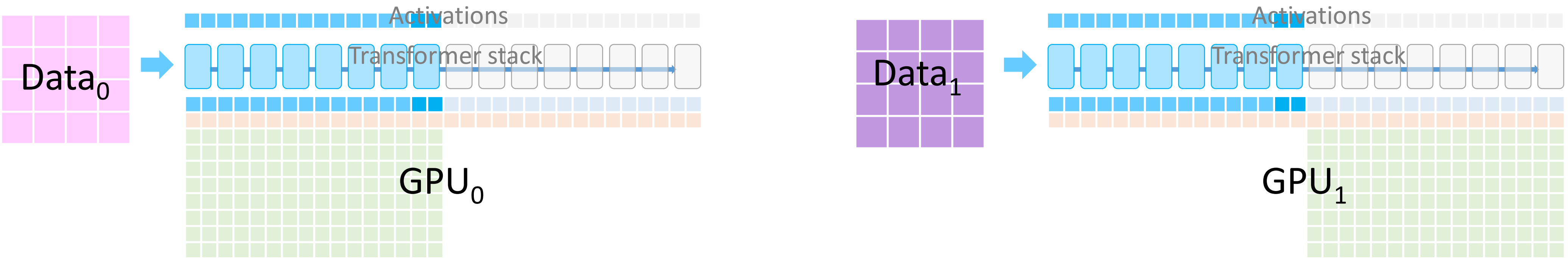
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks

ZeRO: Zero Redundancy Optimizer



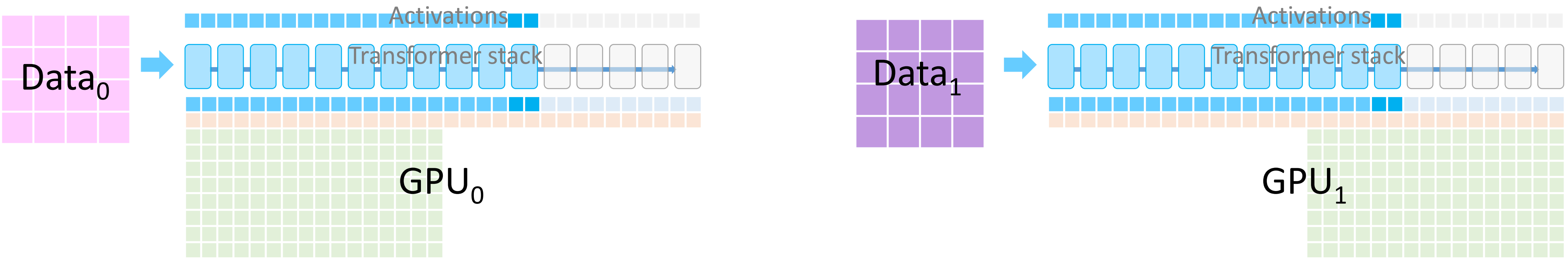
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks

ZeRO: Zero Redundancy Optimizer



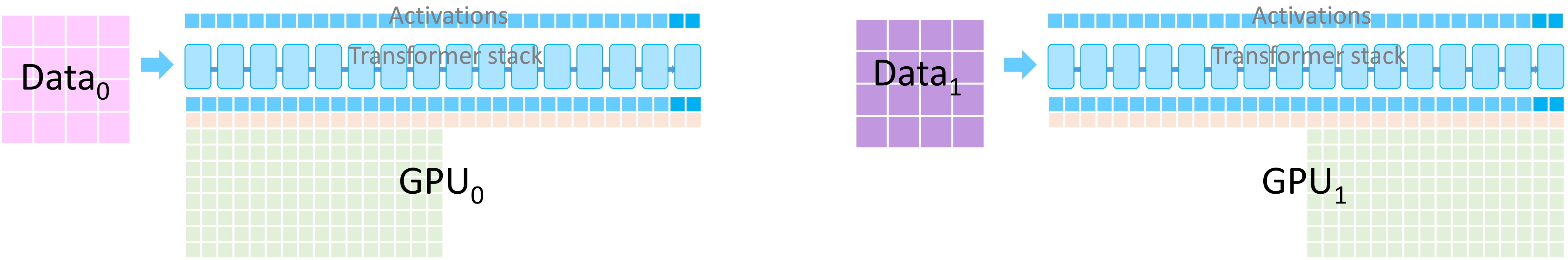
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks

ZeRO: Zero Redundancy Optimizer



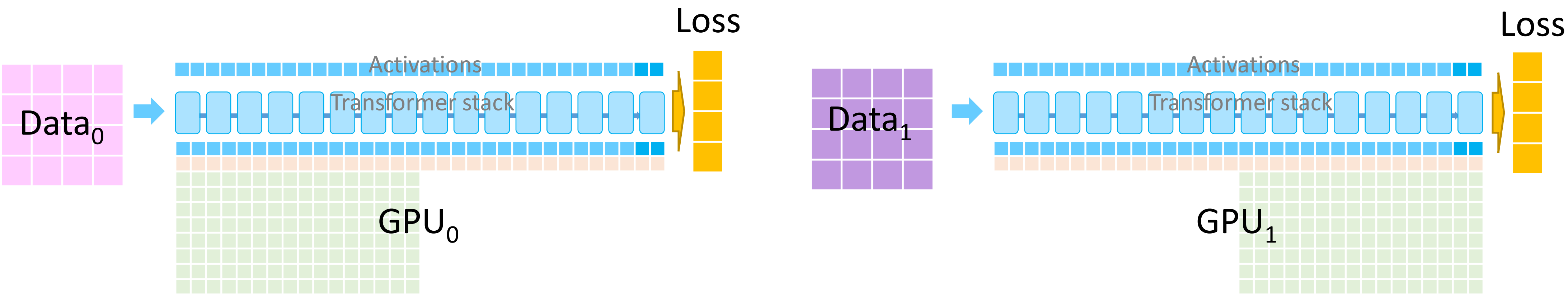
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks

ZeRO: Zero Redundancy Optimizer



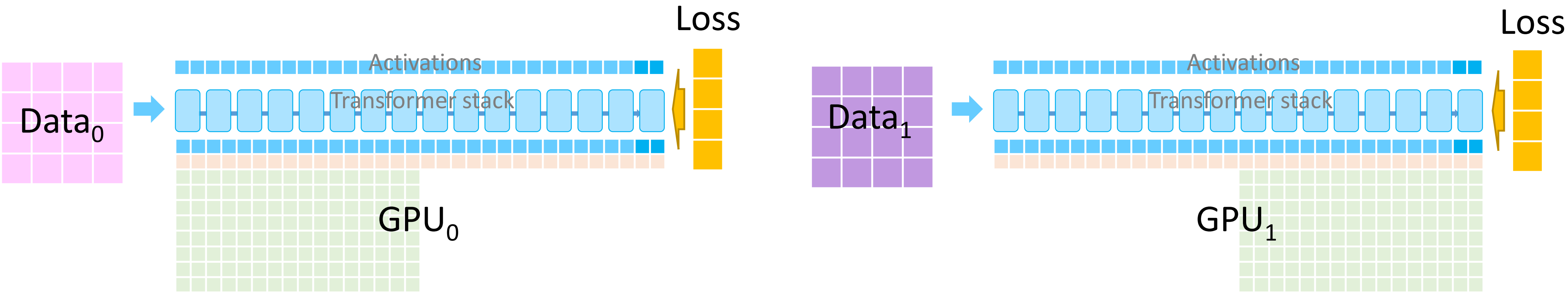
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks

ZeRO: Zero Redundancy Optimizer



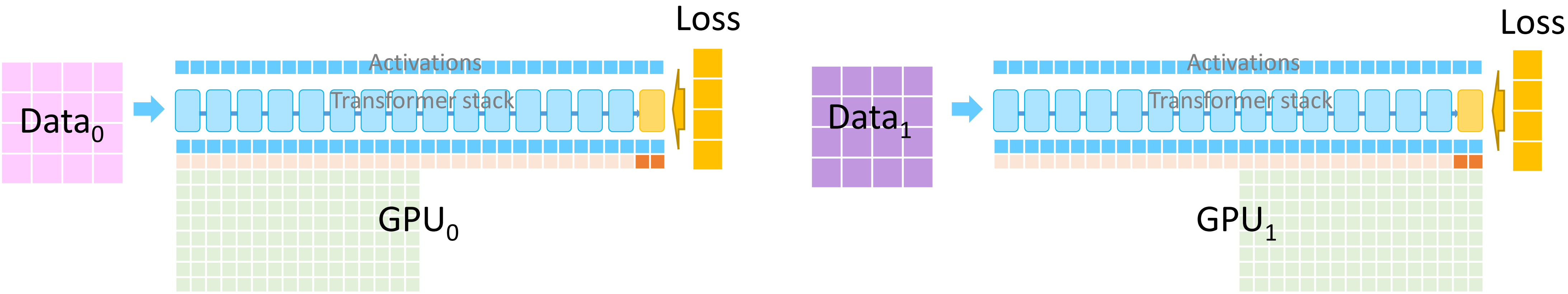
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks

ZeRO: Zero Redundancy Optimizer



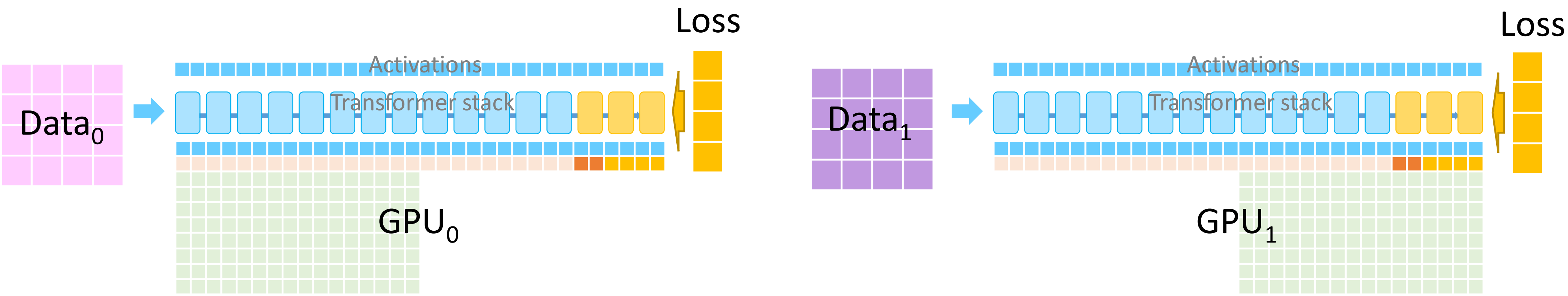
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients

ZeRO: Zero Redundancy Optimizer



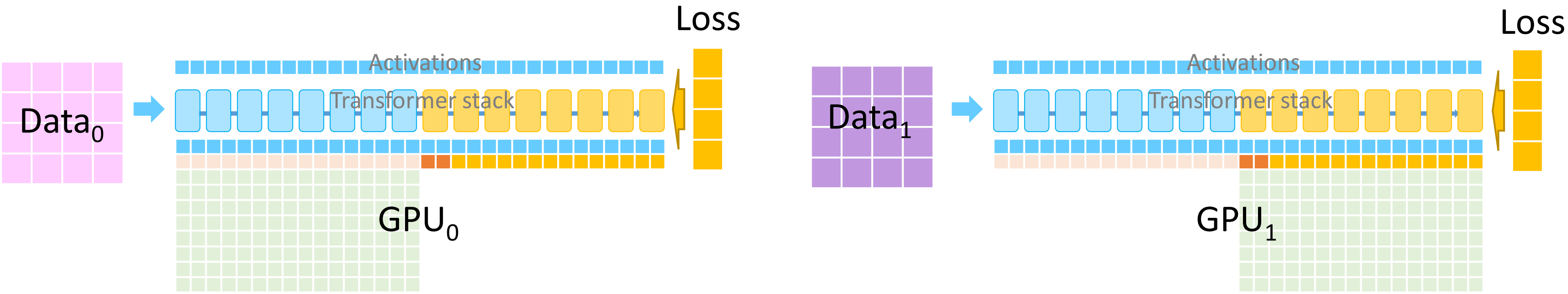
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients

ZeRO: Zero Redundancy Optimizer



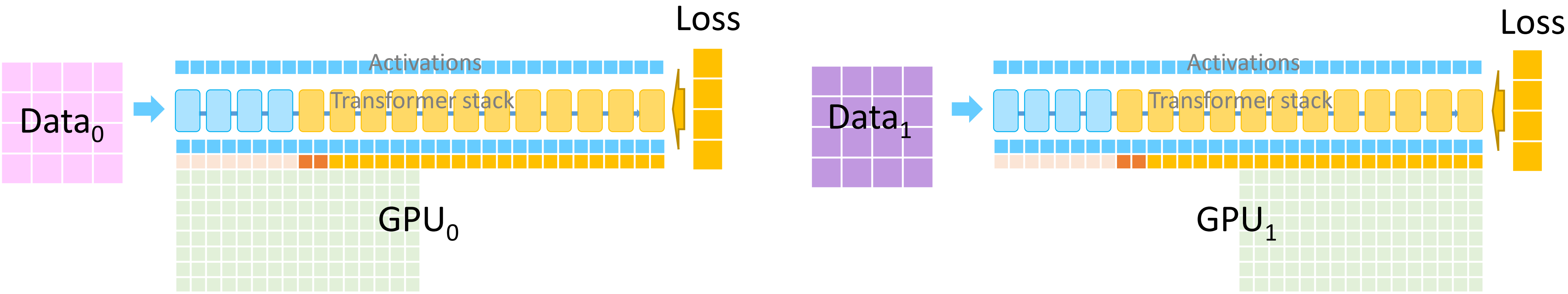
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients

ZeRO: Zero Redundancy Optimizer



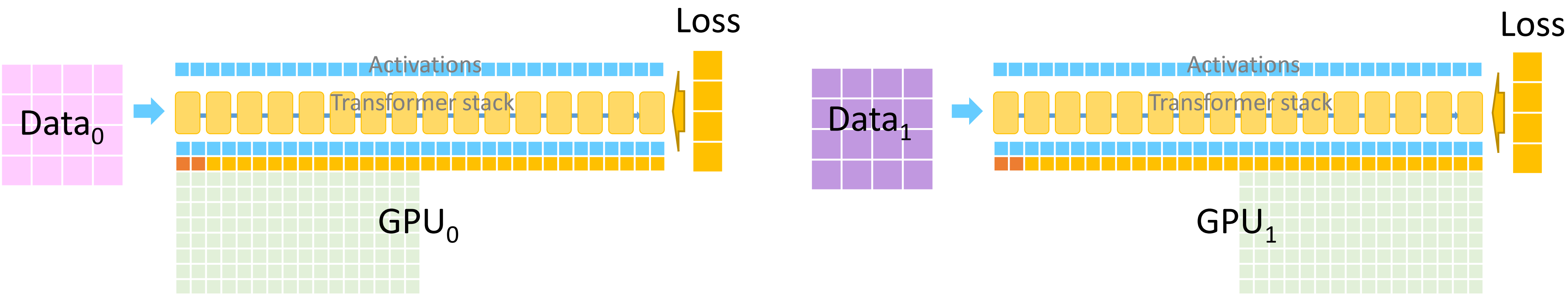
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients

ZeRO: Zero Redundancy Optimizer



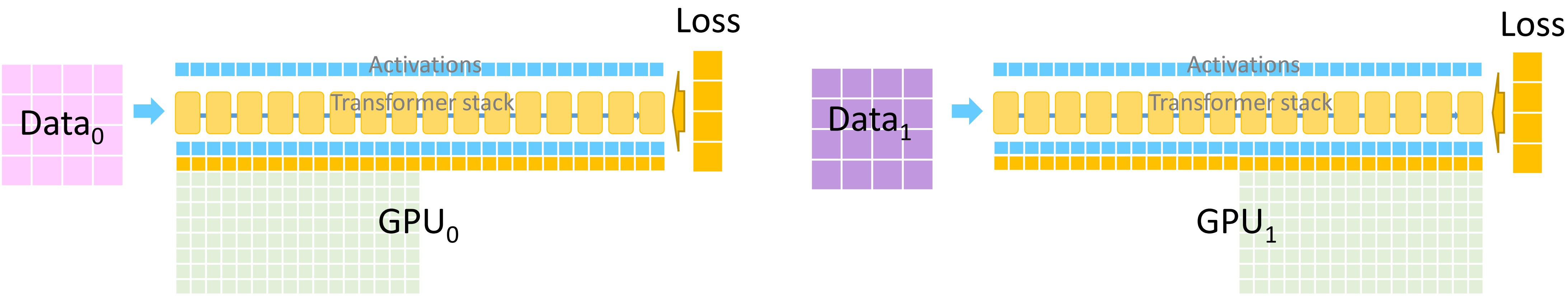
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients

ZeRO: Zero Redundancy Optimizer



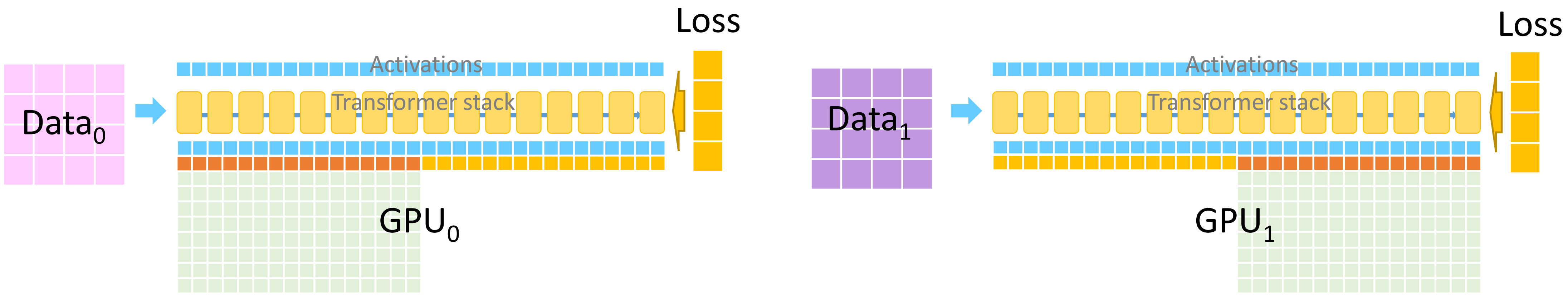
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients

ZeRO: Zero Redundancy Optimizer



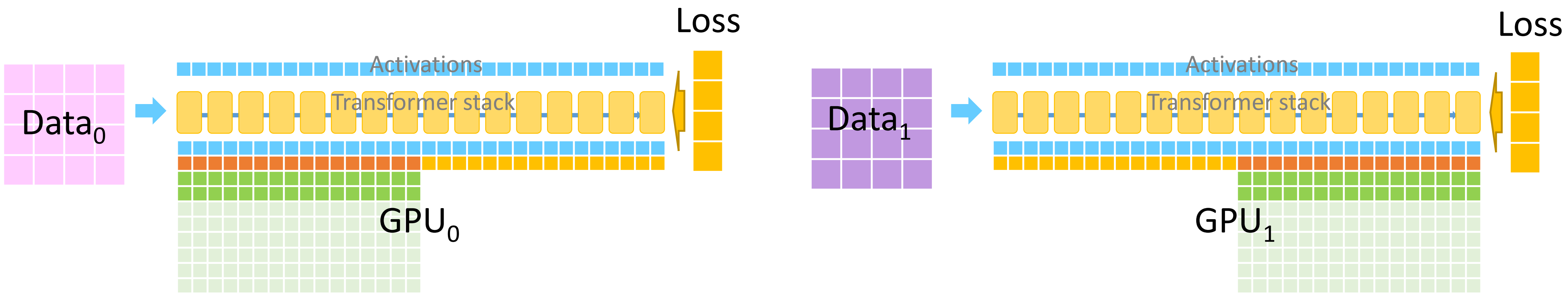
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and reduce scatter to average

ZeRO: Zero Redundancy Optimizer



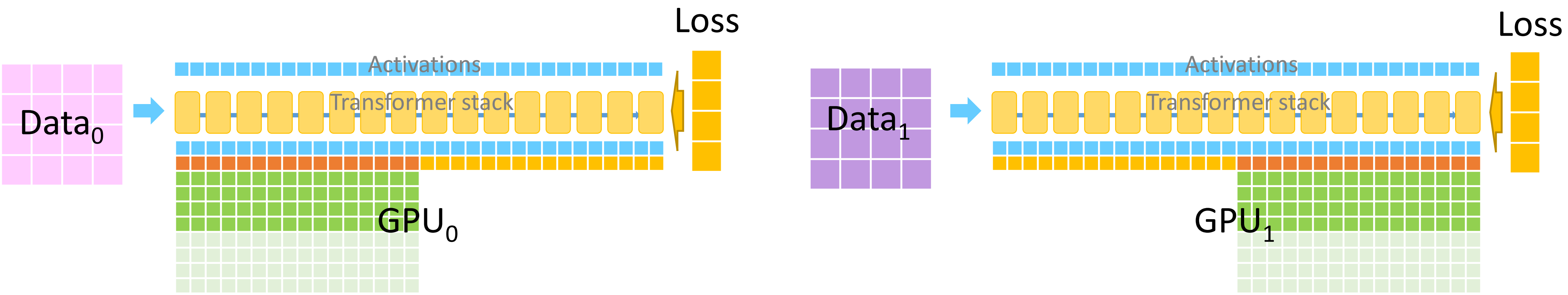
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and reduce scatter to average

ZeRO: Zero Redundancy Optimizer



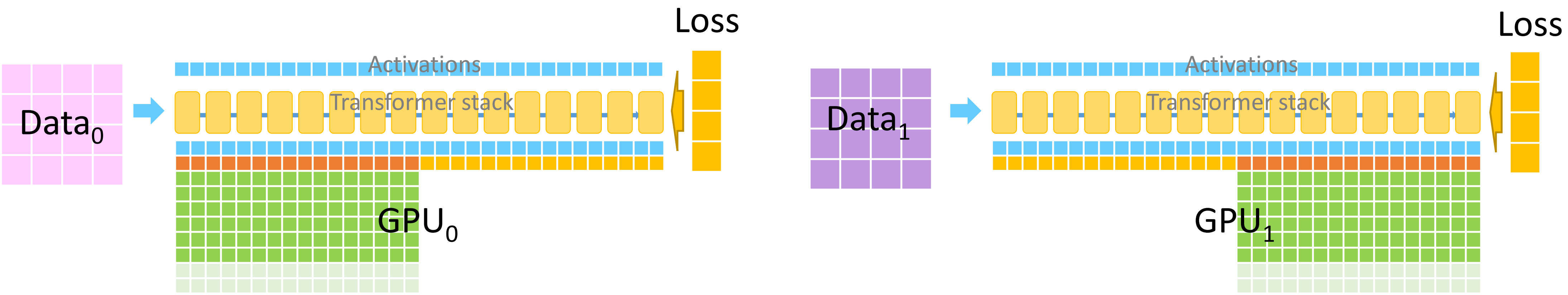
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and reduce scatter to average
- Update the FP32 weights with ADAM optimizer

ZeRO: Zero Redundancy Optimizer



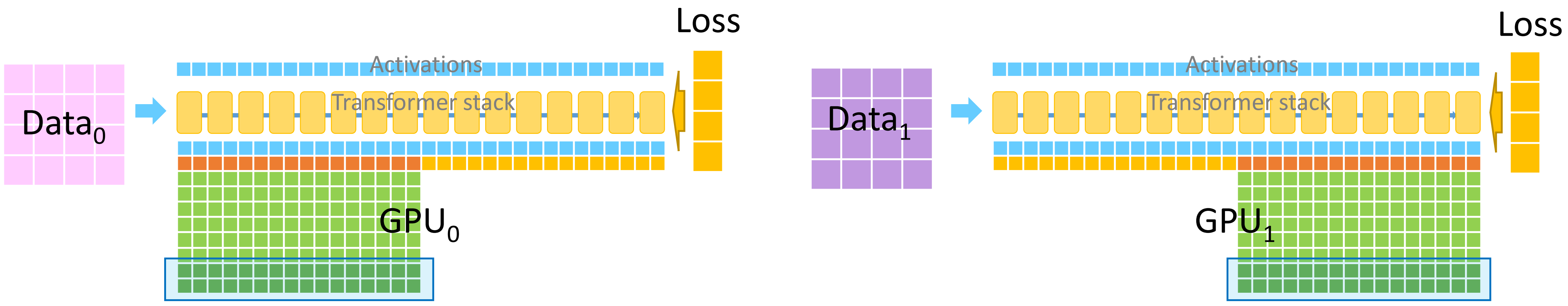
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and reduce scatter to average
- Update the FP32 weights with ADAM optimizer

ZeRO: Zero Redundancy Optimizer



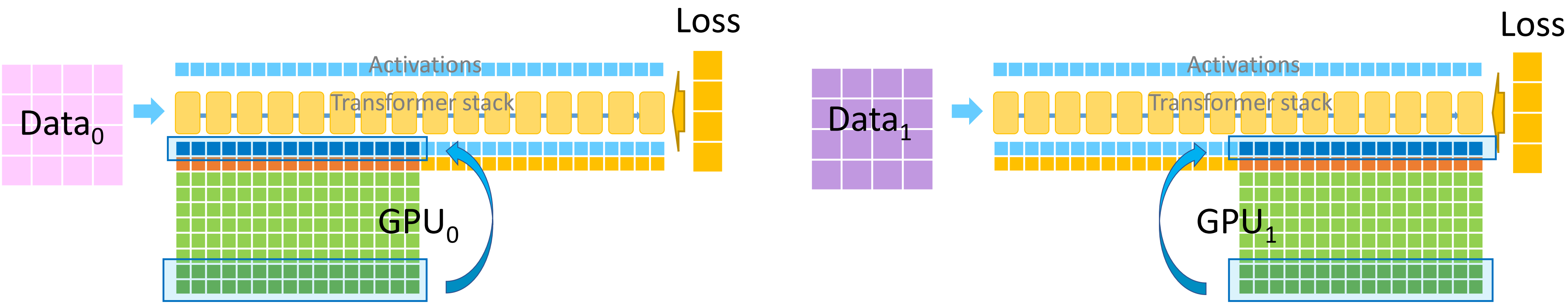
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and reduce scatter to average
- Update the FP32 weights with ADAM optimizer

ZeRO: Zero Redundancy Optimizer



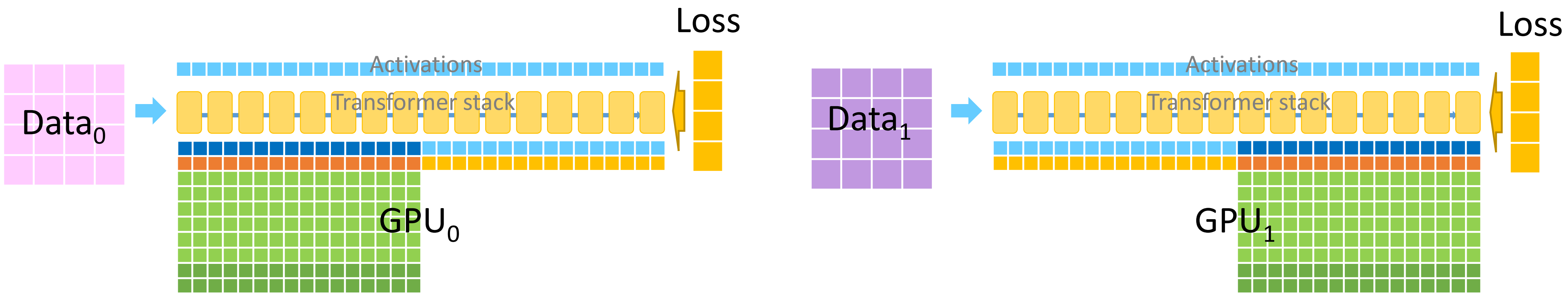
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and reduce scatter to average
- Update the FP32 weights with ADAM optimizer

ZeRO: Zero Redundancy Optimizer



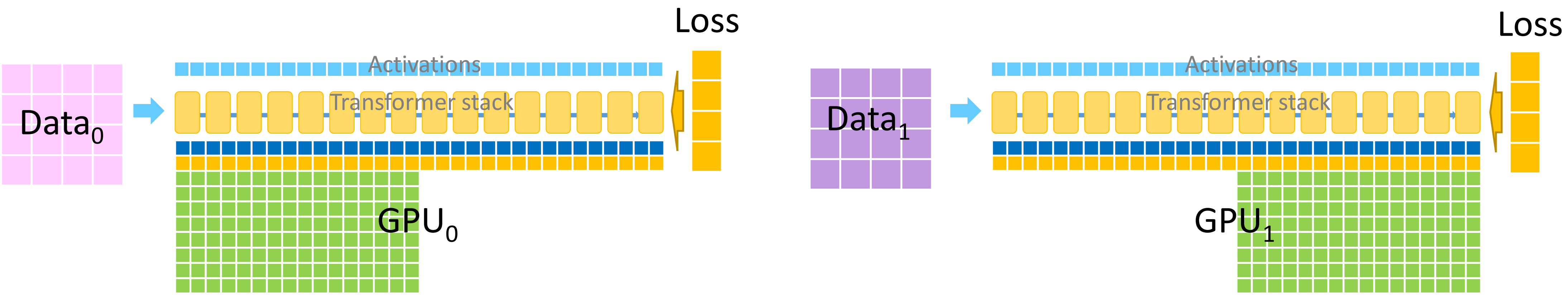
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and reduce scatter to average
- Update the FP32 weights with ADAM optimizer
- Update the FP16 weights

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

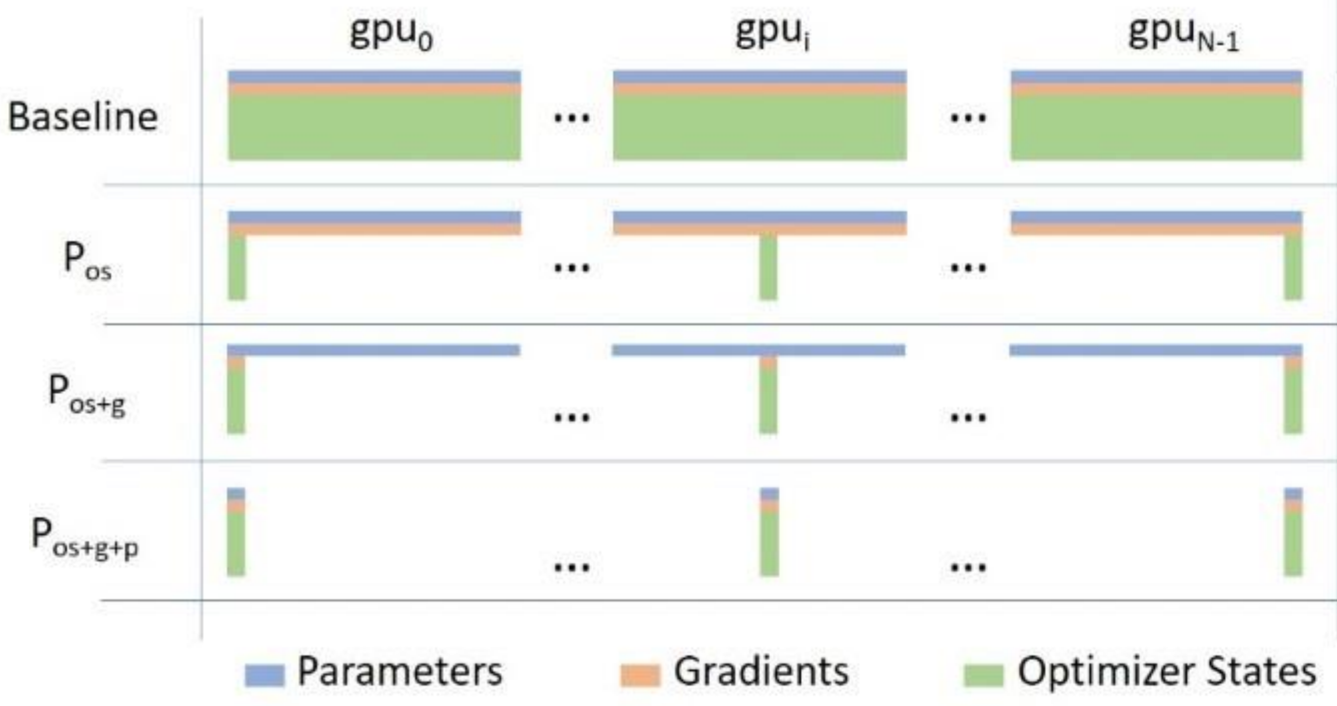
ZeRO: Zero Redundancy Optimizer



- 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

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

DeepSpeed, ZeRO and Model Parallelism

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

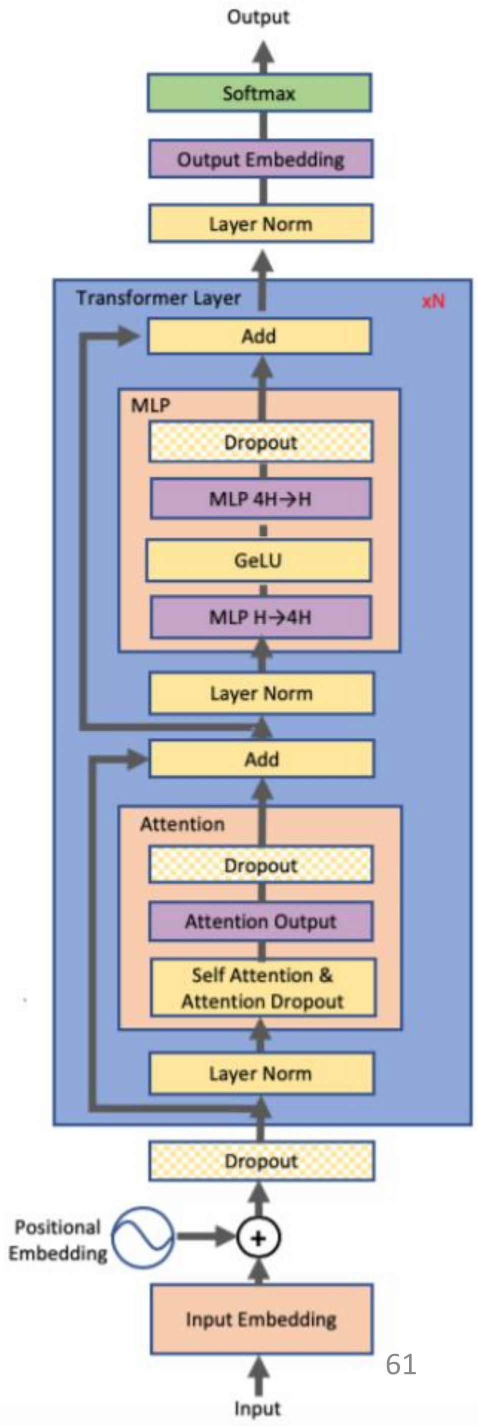
DeepSpeed, ZeRO and Model Parallelism

- ZeRO is model parallelism agnostic
- Can work with any form of model parallelism
  - Tensor Slicing (Megatron)
  - Pipeline Parallelism (Gpipe, PipeDream)
- Model Parallel Unit (mpu)
  - get_data_parallel_group()
  - get_model_parallel_group()

```
# Wrap model, optimizer, and lr scheduler
model, optimizer, lr_scheduler, _ = deepspeed.initialize(
    args=args,
    model=model,
    optimizer=optimizer,
    lr_scheduler=lr_scheduler,
    mpu=mpu
)
```

Example 1: Turing NLG 17B

- DeepSpeed powered new state-of-the art LM
  - 10.21 perplexity
  - **Infeasible** to Train to **Practically Possible**
  - **Over 3x** throughput gain over Megatron alone

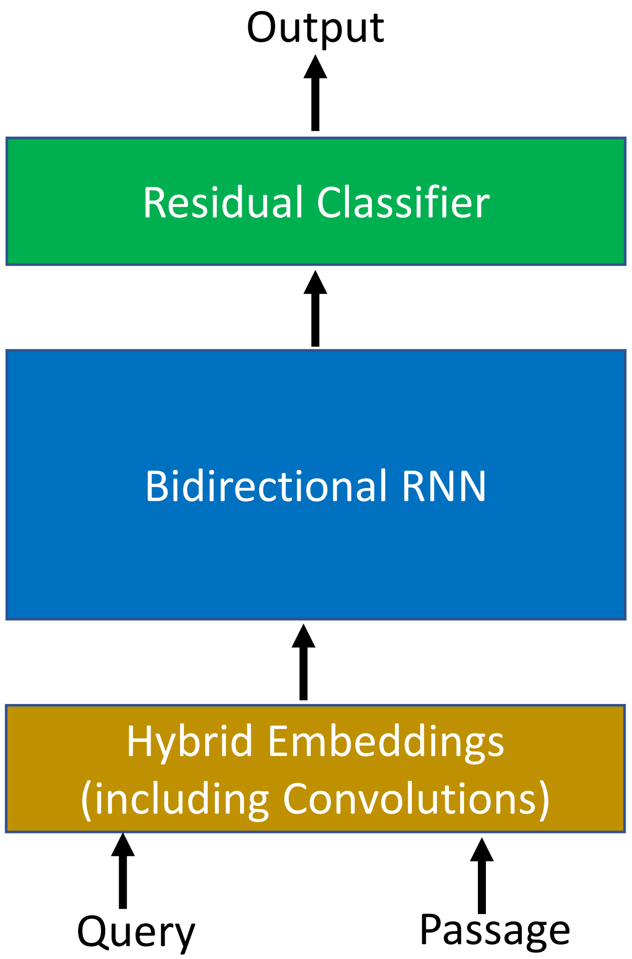


Outline

- Overview
  - Why & What
  - Highlights of results / techniques
  - Software architecture
- How to use DeepSpeed
- Example 1: Turing NLG 17B
  - Result summary, key techniques (ZeRO, flexible combination of parallelism)
- Example 2: RScan
  - Result summary, key techniques (sparse gradients, advanced HP tuning)
- Upcoming features

Example 2: RScan

- Used in Bing production to measure query/passage similarity
- Based on Bidirectional RNNs



Example 2: RScan

- DeepSpeed enables faster convergence to target AUC score
  - 15.6X speedup on 8 GPUs relative to 1 GPU baseline

	Speedup Relative to 1-GPU baseline		
	Efficiency (Samples/Second)	Effectiveness (AUC/Samples)	Wall clock (AUC/Second)
Baseline (1-GPU)	1x	1x	1x
Baseline (8-GPU)	5.8X	0x	0x
DeepSpeed (8-GPU)	8.7x	1.8x	15.6x

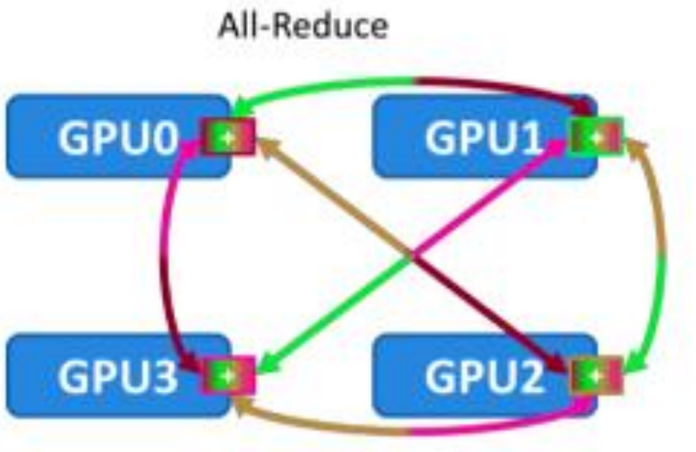
Target AUC not achieved

RScan Breakdown: 15.6X faster convergence

Efficiency: 8.7x

- **Communication**: 6.77x, Sparse gradients + Custom AllReduce
- **Compute**: 1.29x, CuDNN on 1-GPU

More efficient communication



Dense Representation

0.24	2.19
0.98	0.45
0.00	0.00
2.48	-3.03
0.00	0.00
3.97	-6.51
0.00	0.00
5.47	-9.99
0.00	0.00
0.00	0.00
0.00	0.00
8.45	-16.95
0.00	0.00
9.95	-20.43

→

Index Slices

Index	Values	
0	0.24	2.19
1	0.98	0.45
3	2.48	-3.03
5	3.97	-6.51
7	5.47	-9.99
11	8.45	-16.95
13	9.95	-20.43

Index Slices with Duplicates

Index	Values	
0	0.24	2.19
1	0.98	0.45
0	2.48	-3.03
5	3.97	-6.51
1	5.47	-9.99
1	8.45	-16.95
0	9.95	-20.43

→

Unique Index Slices

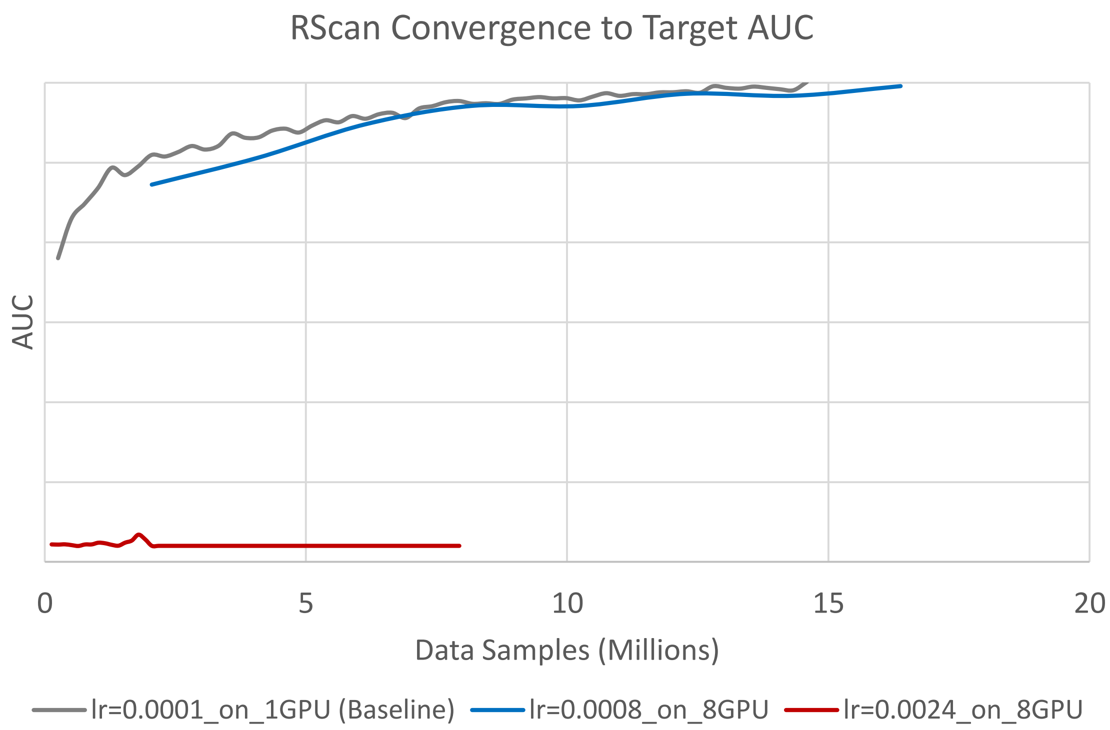
Index	Values	
0	12.67	-21.27
1	14.9	-26.49
5	3.97	-6.51

RScan Breakdown: 15.6X faster convergence

Effectiveness: 1.8x

Challenge: Slow Convergence of Batch Scaling

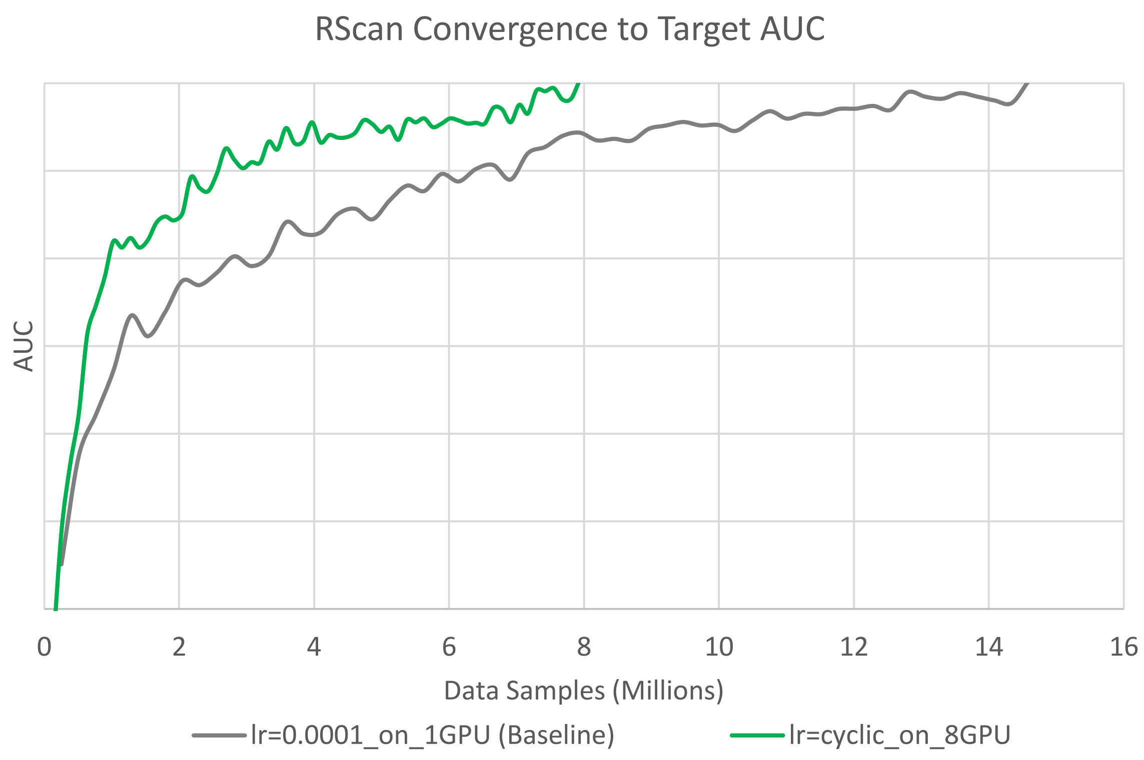
- Less frequent model updates
  - 8-GPU updates $= \frac{1}{8}$ 1-GPU updates
- Small batch hyperparameters not optimal for large batch
- LR linear scaling does not work
  - Slow convergence, or
  - Model divergence

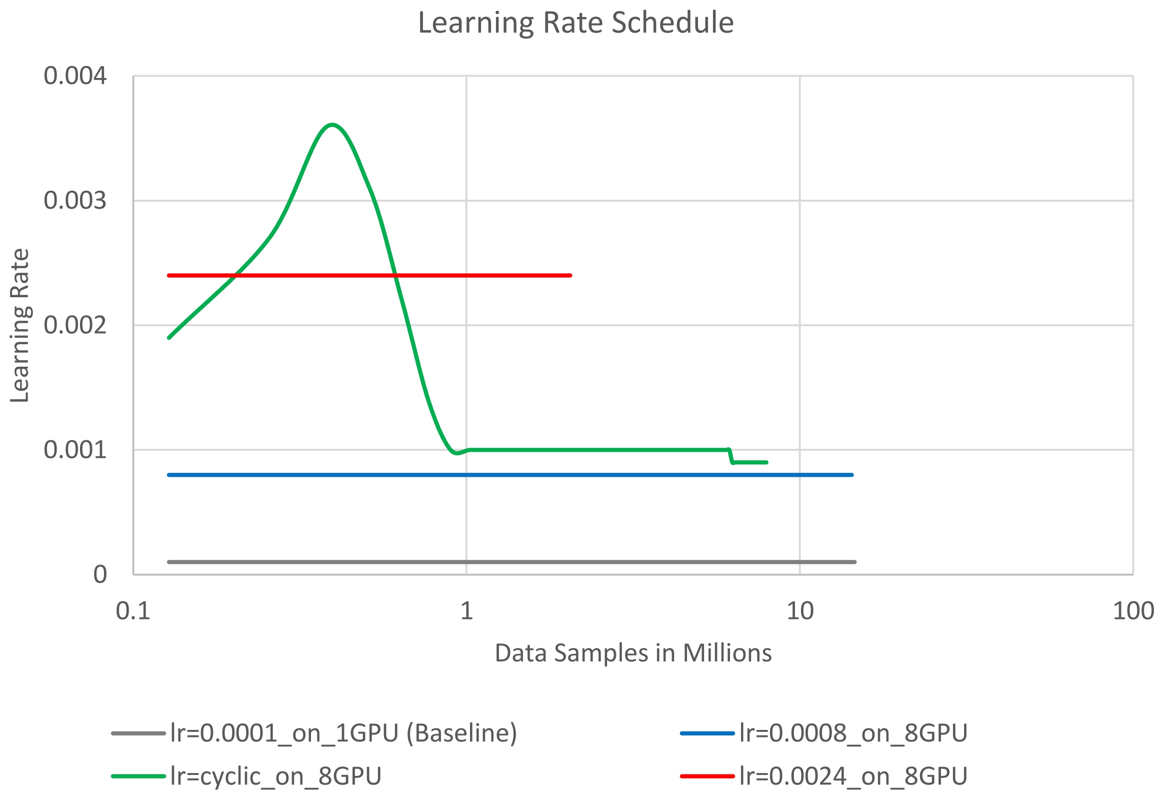


RScan Breakdown: 15.6X faster convergence

Effectiveness: 1.8x

- **Adaptive Hyperparameter tuning:** 1.8x
  - 1Cycle & Decay schedules: LR, momentum, ..., etc.





RScan Breakdown: 15.6x faster convergence

Efficiency: 8.7x

Communication: 6.77x

Compute: 1.29x

Effectiveness: 1.8x

Hyperparameter tuning

Upcoming Features

- Reduce Scatter for ZeRO stage 1
  - Partition-aware approach instead of global collective (all-reduce)
  - Total communication volume reduction 1.5x → 1x of data parallelism
  - Up to 2x reduction in communication time compared to all-reduce
- Zero Stage 2
  - Reduce memory footprint of gradients
  - Train larger models: e.g., 10B parameters on 32GPUs without model parallelism
  - Train larger batch sizes
- For more new and exciting features
  - Repo: https://github.com/microsoft/DeepSpeed
  - Breaking news page and documentation: https://www.deepspeed.ai/

DeepSpeed + ZeRO

Scale
- 100B parameter
- 10X bigger

Speed
- Up to 5X faster

Cost
- Up to 5X cheaper

Usability
- Minimal code change

https://github.com/microsoft/DeepSpeed