Distributed Training (Contd.)

Arvind Krishnamurthy

Material adapted from slides by Tianqi Chen & Zhihao Jia (CMU), Minjia Zhang (UIUC)

How can we parallelize ML training?

Goals:

- Scale with training data size (ensure that compute efficiency is high)
- Scale with model size (ensure that memory efficiency is high)

Approaches:

- Data parallelism
- Model parallelism (tensor & pipeline)

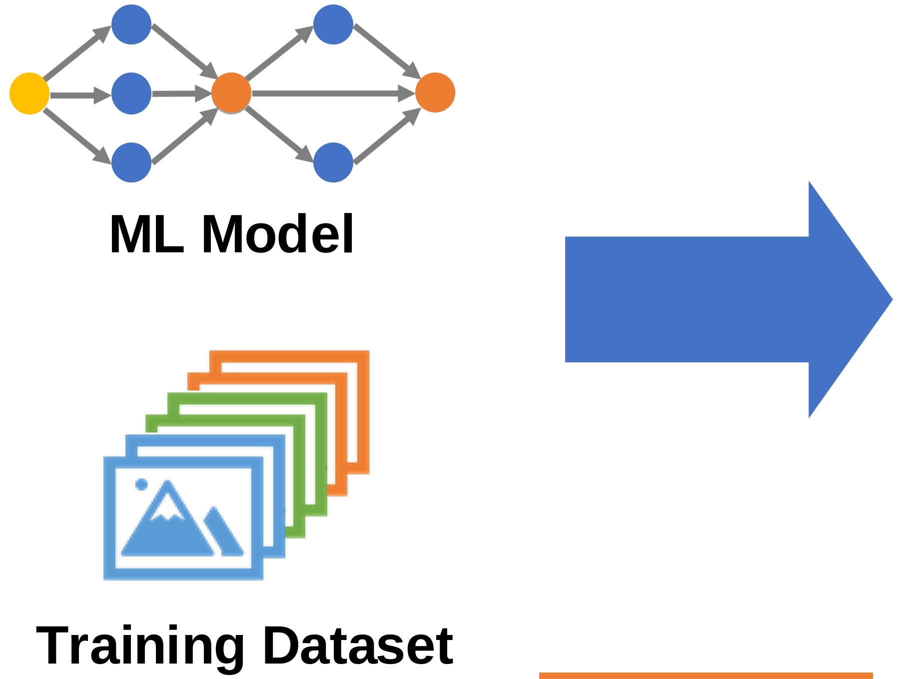
Training State

- **Model parameters:** eventually becomes the released model
- **Gradients:** how the loss function varies with small changes to parameters
- **Optimizer state:** maintains information about how parameters change over time
- **Activations:** intermediate results from the forward phase required for back-propagating gradients

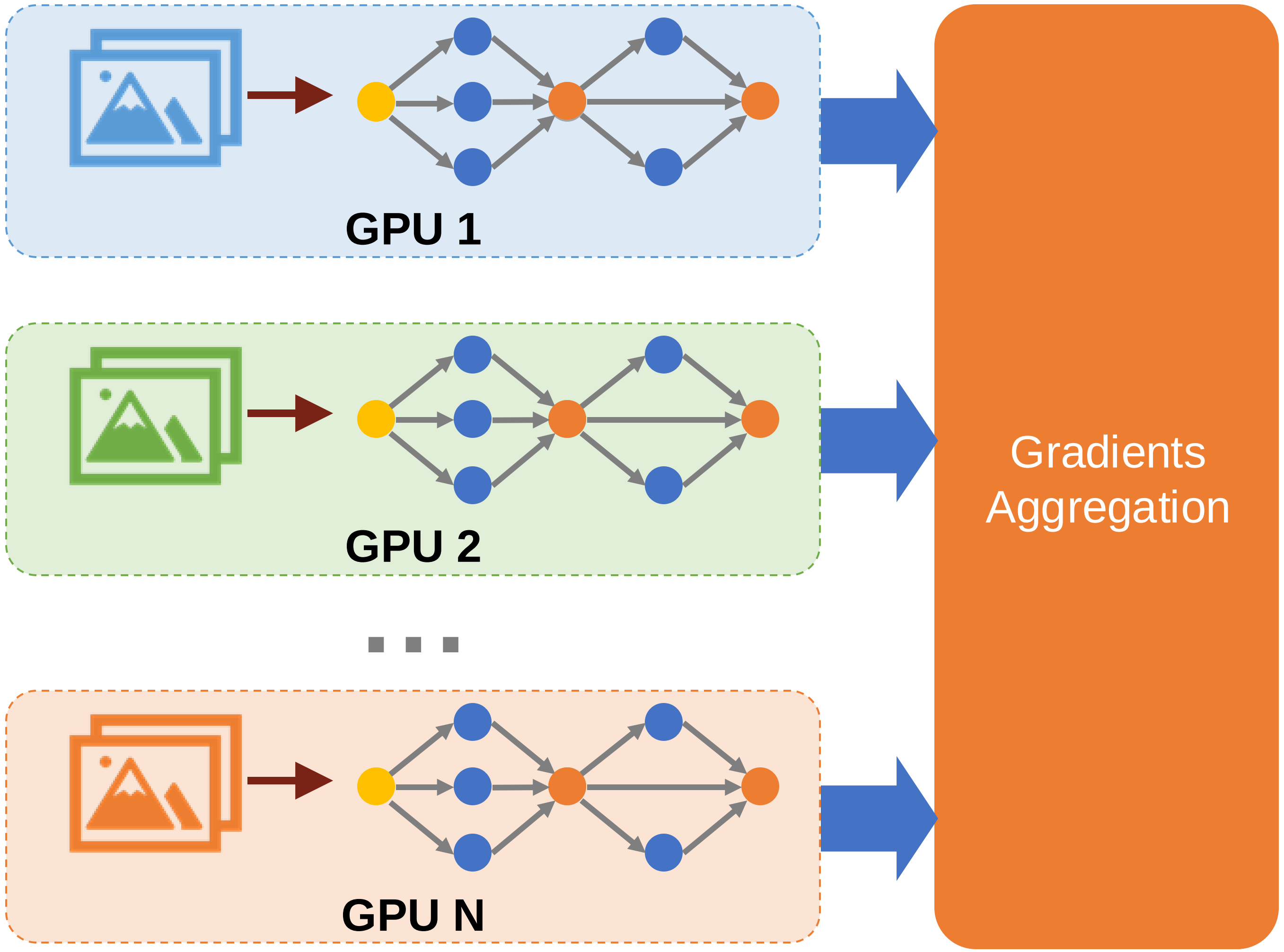
Data Parallelism

- Approach:
  - Split data into batches for each training iteration
  - Further split batch into mini-batches, each processed by a separate node
  - Perform forward/backwards pass to generate per-node gradients
  - Accumulate gradients across nodes
  - Update parameters based on averaged gradients

Data Parallelism



$$w_i := w_i - \gamma \nabla L(w_i) = w_i - \boxed{\frac{\gamma}{n} \sum_{j=1}^{n} \nabla L_j(w_i)}$$



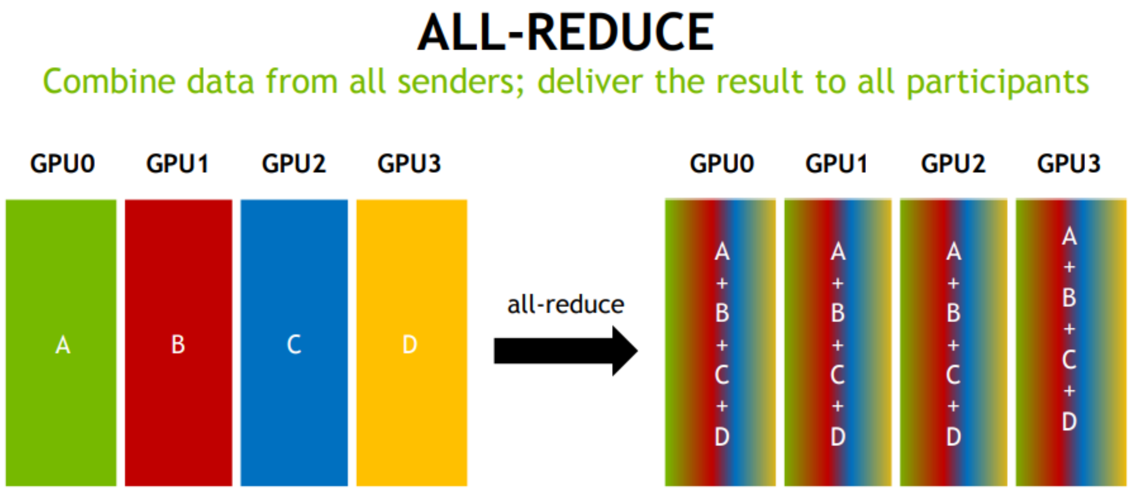
1. Partition training data into batches

2. Compute the gradients of each batch on a GPU

3. Aggregate gradients across GPUs

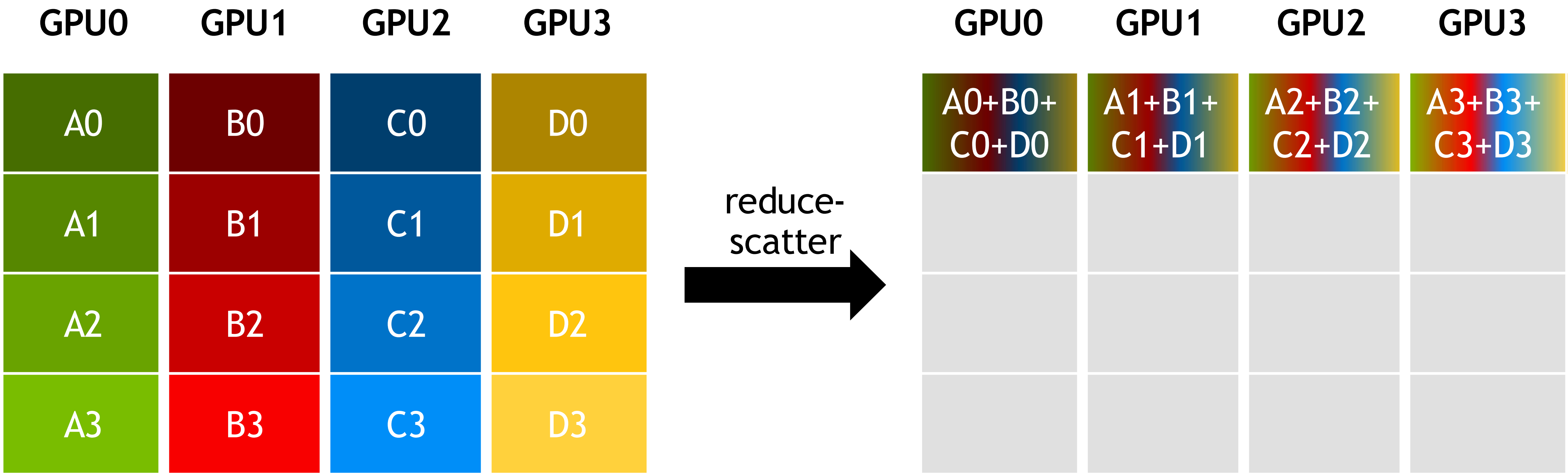
Collectives Synchronous - Communication Handling

Compute Nodes directly talk to each other to globally reduce their gradients and update the model through ***All-Reduce*** communication pattern.



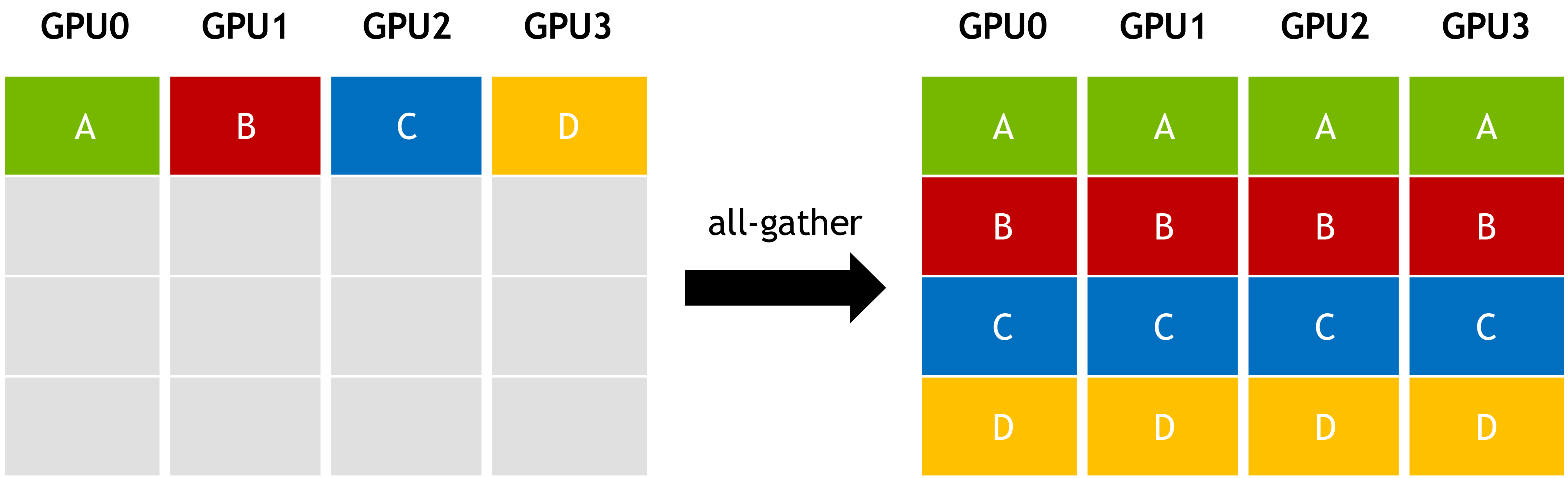
Other variants – ReduceScatter

Combine data from all senders; distribute result across participants



Other variants -- AllGather

Gather messages from all; deliver gathered data to all participants



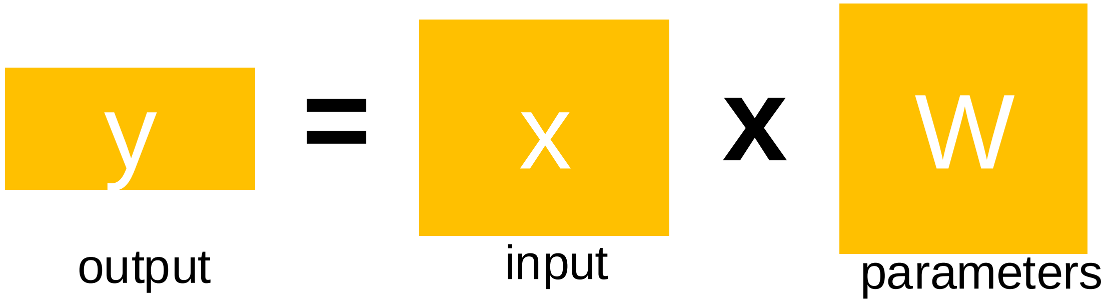
Data Parallelism

- Does it achieve high compute efficiency?
- Does it achieve high memory efficiency as in being able to run large models?

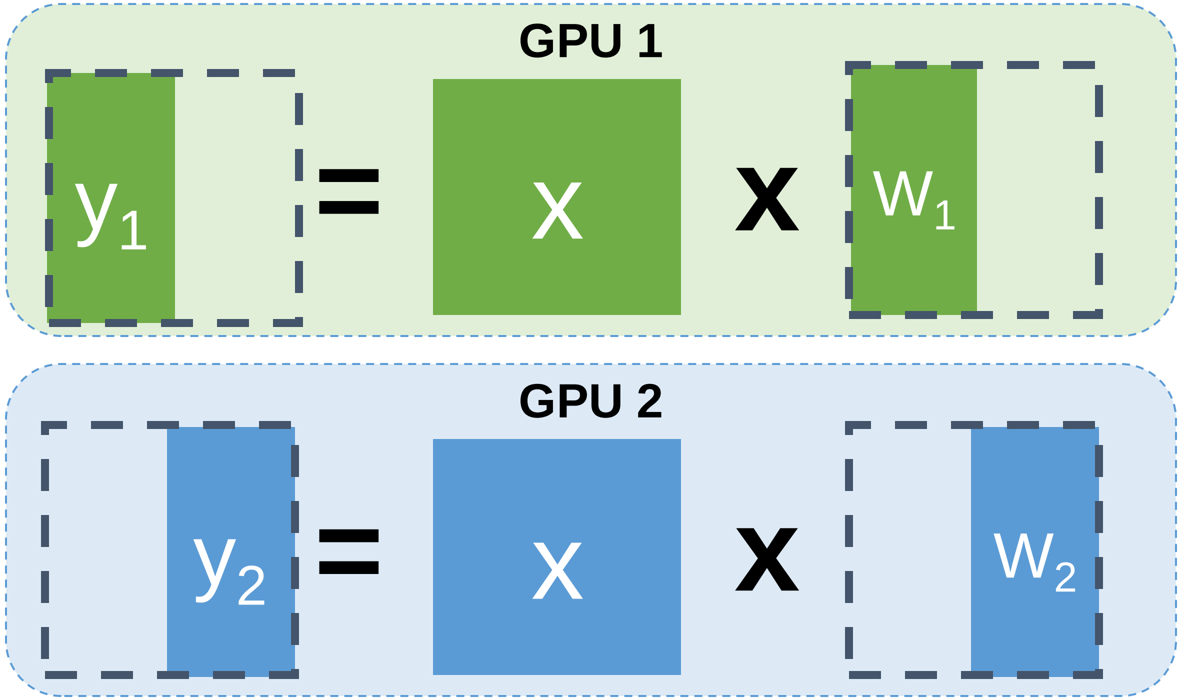
Model parallelism

- No node maintains the entire model
- Two forms:
  - Tensor parallelism
  - Pipeline parallelism

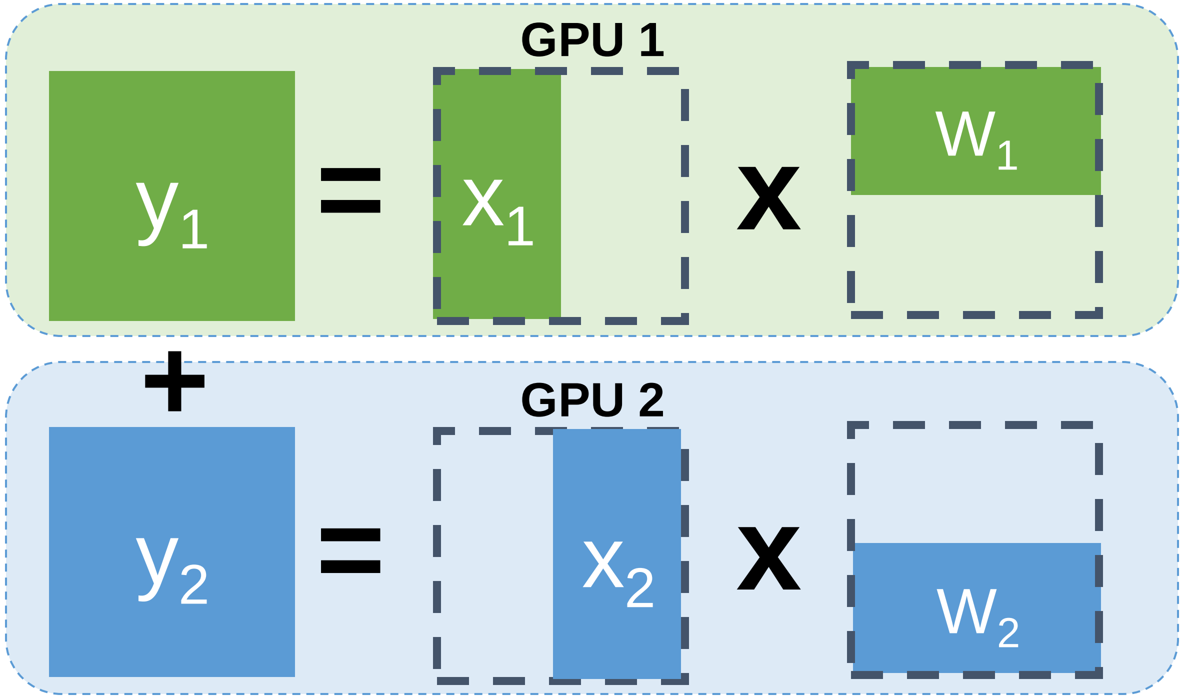
Tensor Model Parallelism



- Partition parameters/gradients ***within*** a layer



Tensor Model Parallelism (partition output)



Tensor Model Parallelism (reduce output)

$y = y1 + y2$

Tensor Parallelism

- Compute efficiency?
- Memory efficiency?

Pipeline Parallelism

- Divide a mini-batch into multiple micro-batches
- Pipeline the forward/backward computations across micro-batches
- Generally combined with model parallelism

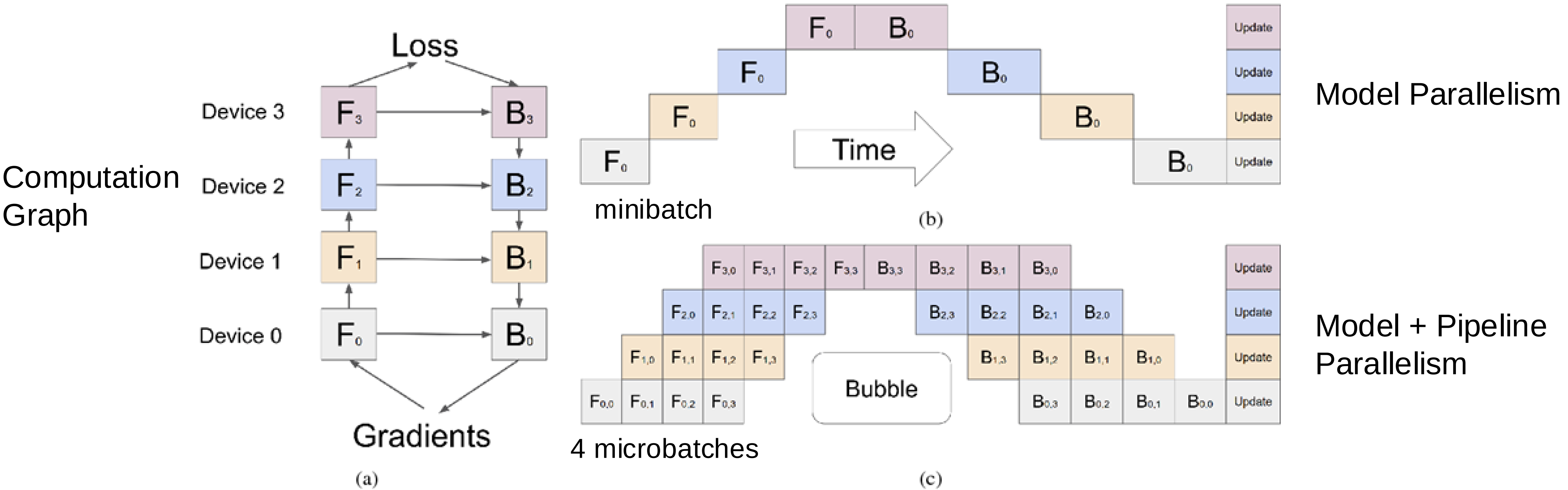


Figure from GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding

Pipeline Parallelism

- Compute efficiency?
- Memory efficiency?

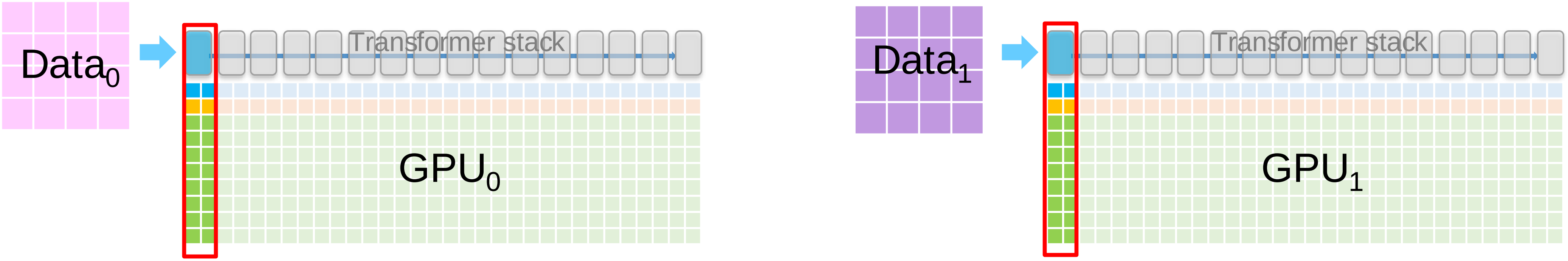
Revisiting Data Parallelism

- How much can we achieve with just simple modifications to data parallelism?
- Can we improve DP's memory efficiency?

Understanding Memory Consumption

A 16-layer transformer model ▢ = 1 layer

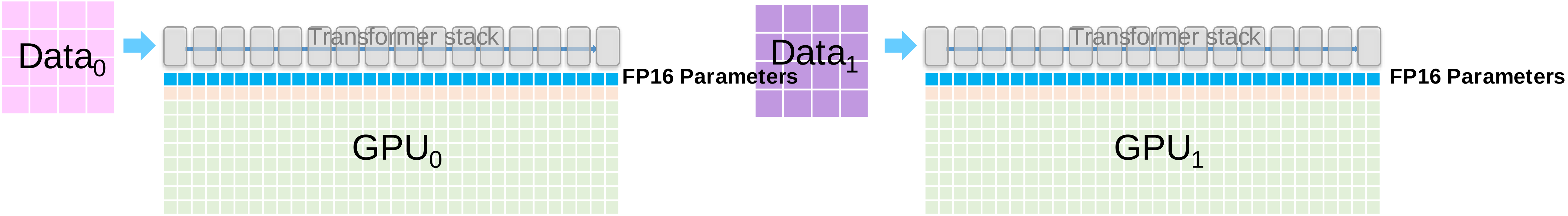
Understanding Memory Consumption



Each cell represents GPU memory used by its corresponding transformer layer

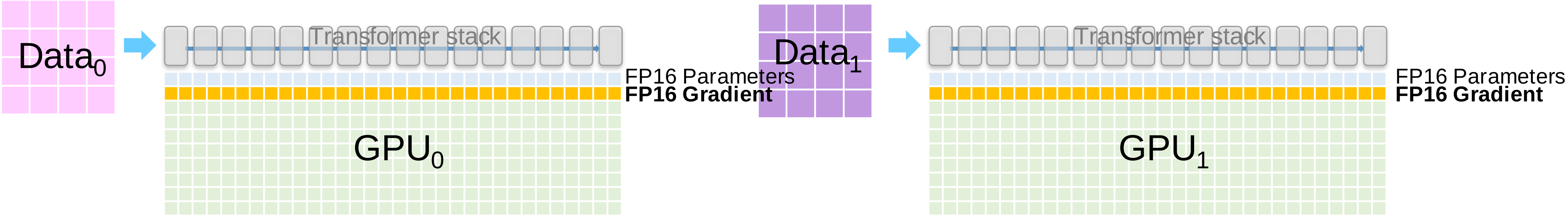
Adapted from Minjia Zhang, DeepSpeed Presentation

Understanding Memory Consumption



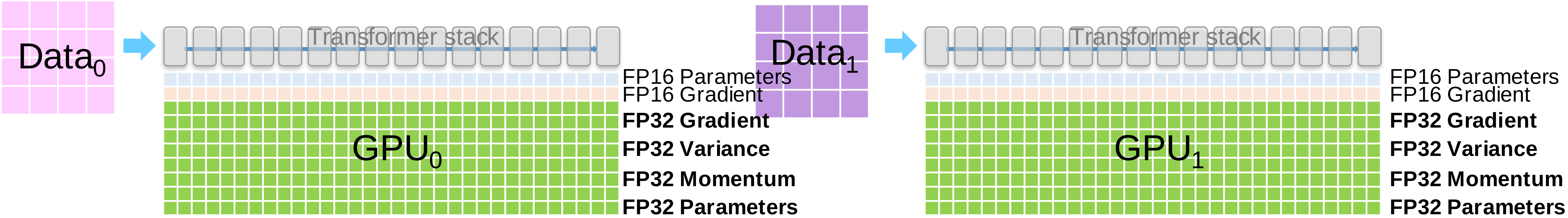
- FP16 parameter

Understanding Memory Consumption



- FP16 parameter
- FP16 Gradients

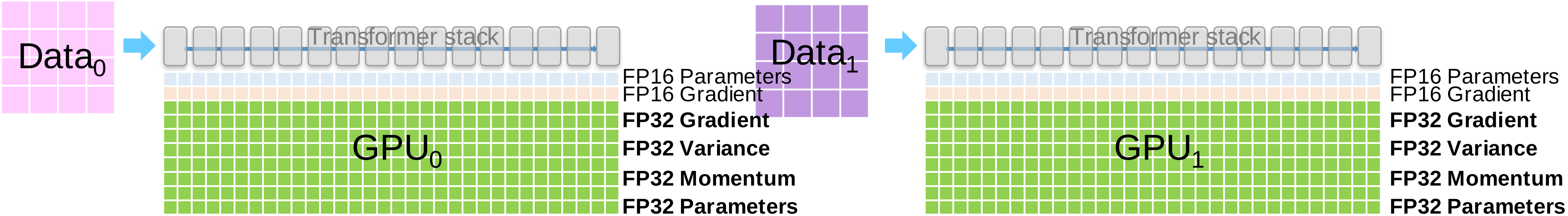
Understanding Memory Consumption



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

Adapted from Minjia Zhang, DeepSpeed Presentation

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

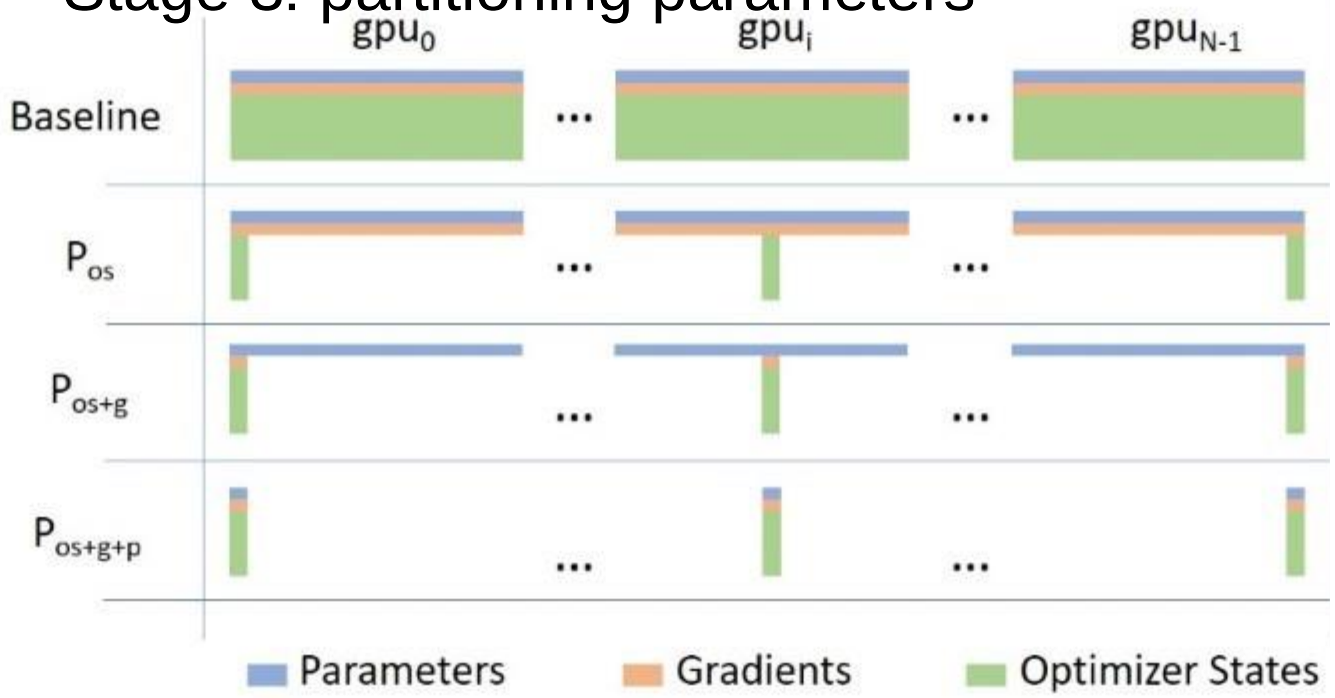
Example 1B parameter model -> 20GB/GPU

Memory consumption doesn’t include:
- Input batch + activations

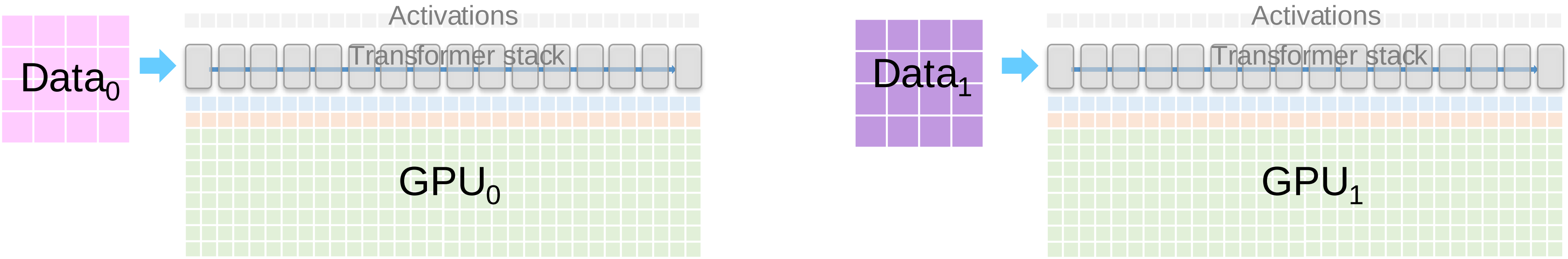
Adapted from Minjia Zhang, DeepSpeed Presentation

ZeRO-DP: ZeRO powered Data Parallelism

- ZeRO removes the redundancy across data parallel process
- Stage 1: partitioning optimizer states
- Stage 2: partitioning gradients
- Stage 3: partitioning parameters



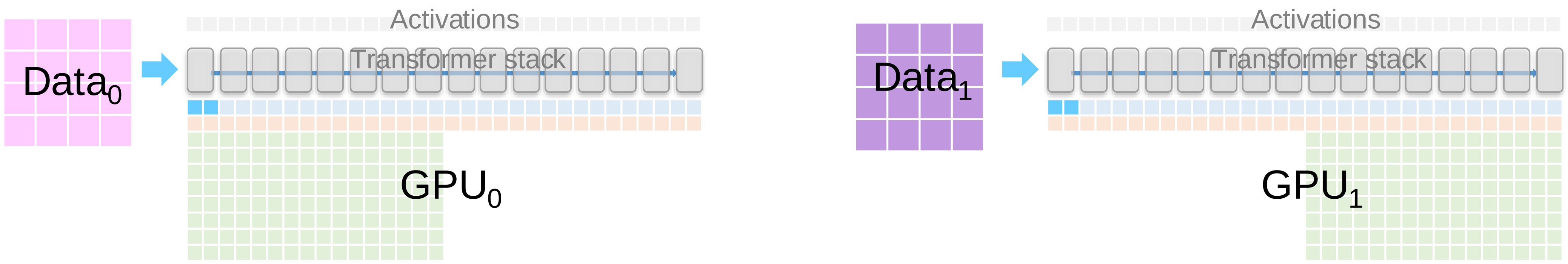
ZeRO Stage 1: Partitioning Optimizer States



- ZeRO Stage 1

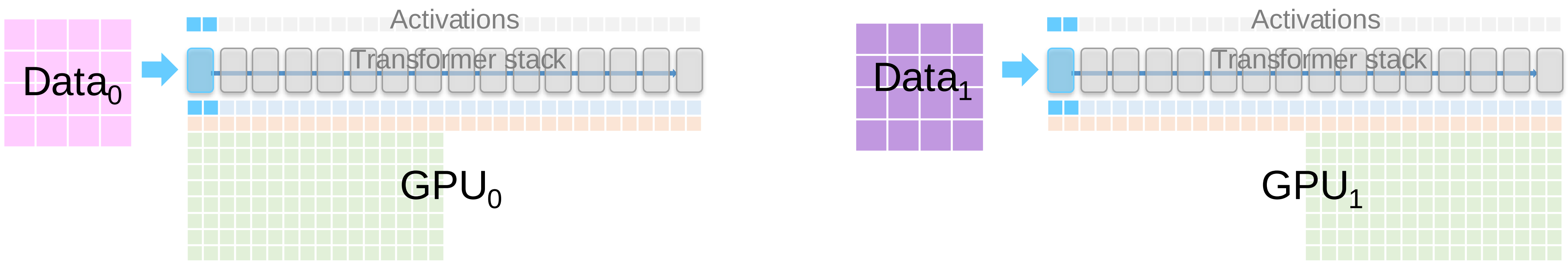
Adapted from Minjia Zhang, DeepSpeed Presentation

ZeRO Stage 1: Partitioning Optimizer States



- ZeRO Stage 1
- Partitions optimizer states across GPUs

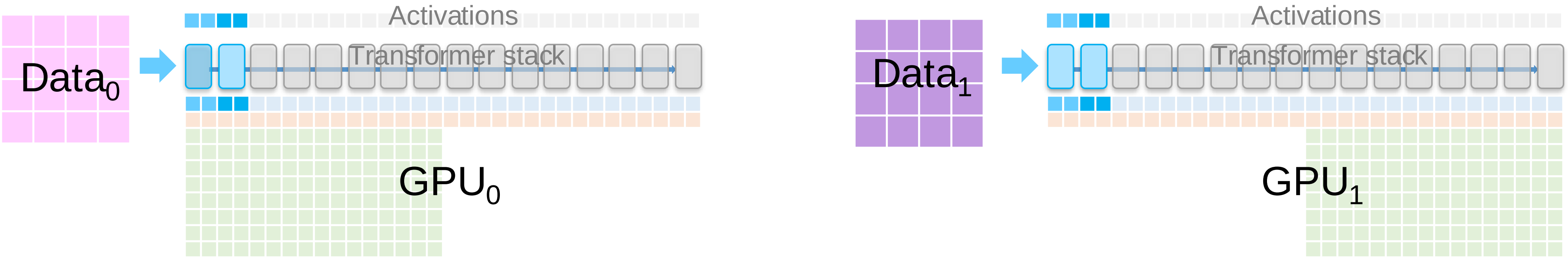
ZeRO Stage 1: Partitioning Optimizer States



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

Adapted from Minjia Zhang, DeepSpeed Presentation

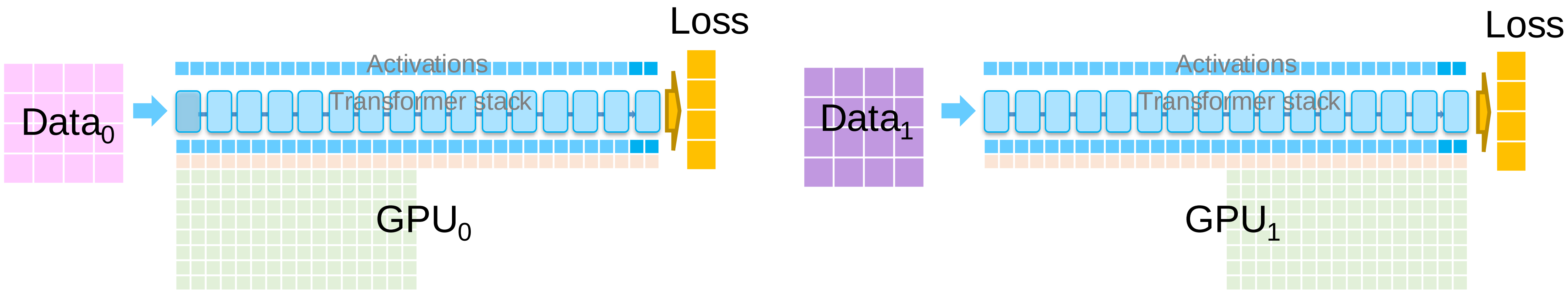
ZeRO Stage 1: Partitioning Optimizer States



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

Adapted from Minjia Zhang, DeepSpeed Presentation

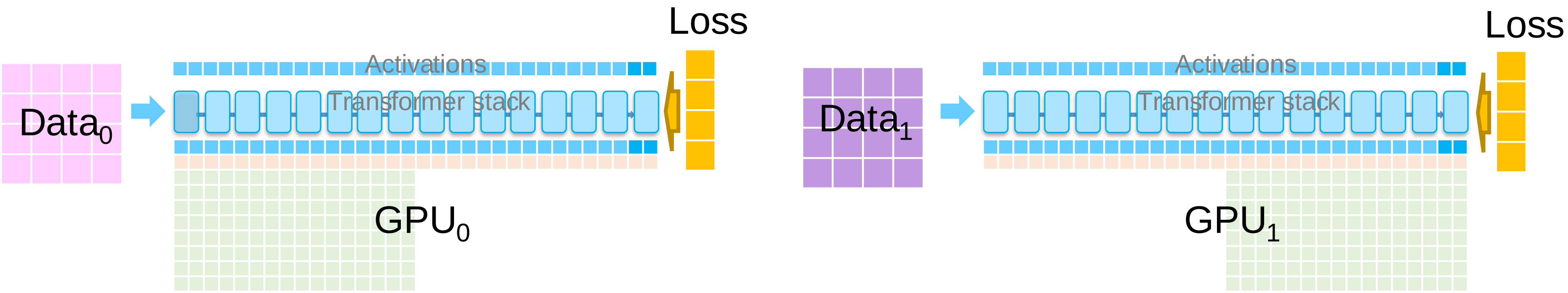
ZeRO Stage 1: Partitioning Optimizer States



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

Adapted from Minjia Zhang, DeepSpeed Presentation

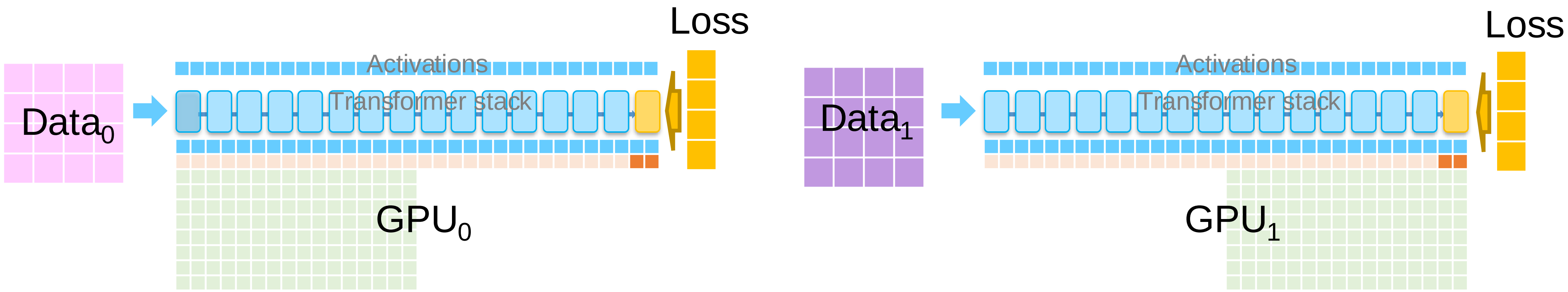
ZeRO Stage 1: Partitioning Optimizer States



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

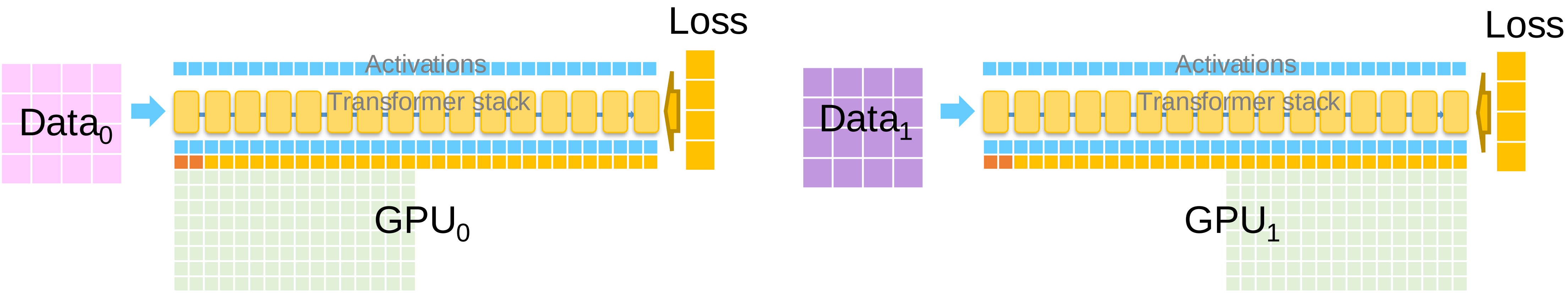
Adapted from Minjia Zhang, DeepSpeed Presentation

ZeRO Stage 1: Partitioning Optimizer States



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

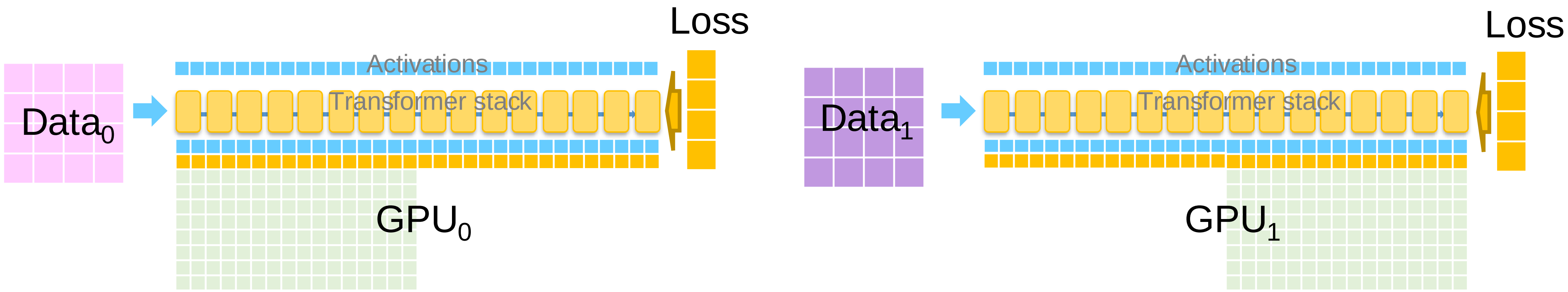
ZeRO Stage 1: Partitioning Optimizer States



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

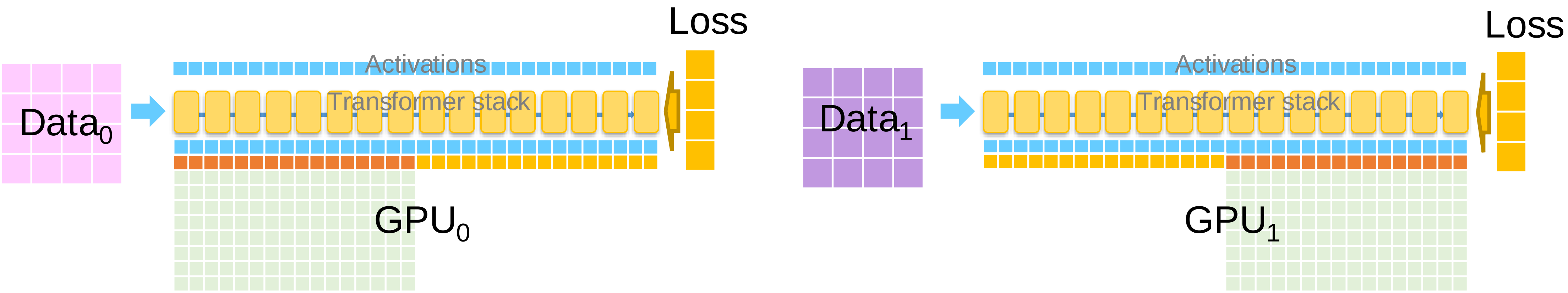
Adapted from Minjia Zhang, DeepSpeed Presentation

ZeRO Stage 1: Partitioning Optimizer States



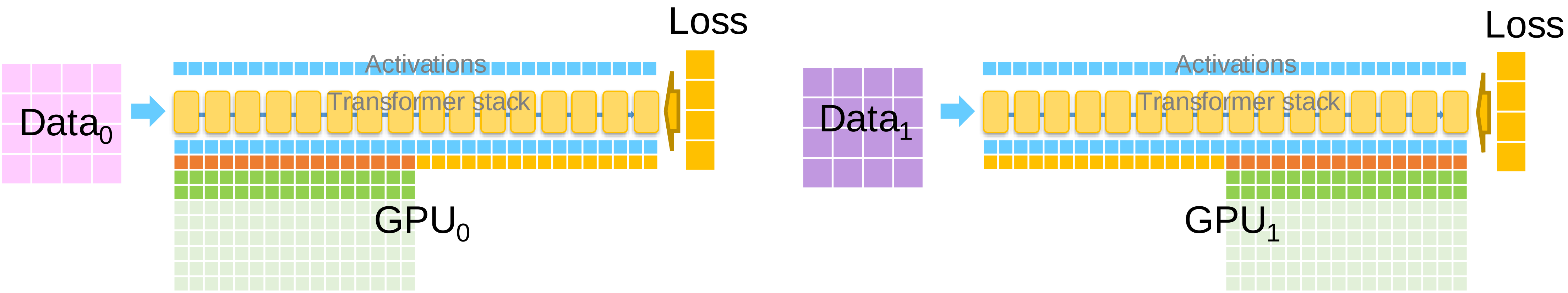
- ZeRO Stage 1
- Partitions optimizer states across GPUs
- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and ReduceScatter to average

ZeRO Stage 1: Partitioning Optimizer States



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

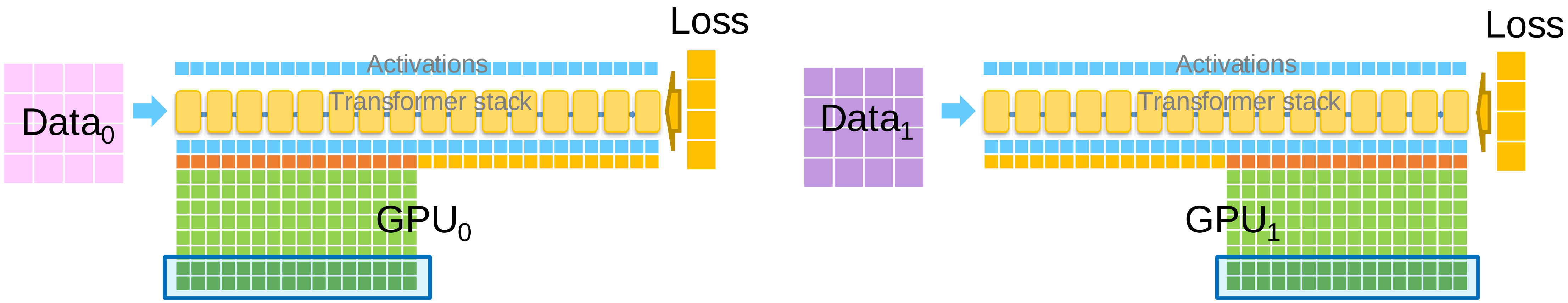
ZeRO Stage 1: Partitioning Optimizer States



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

Adapted from Minjia Zhang, DeepSpeed Presentation

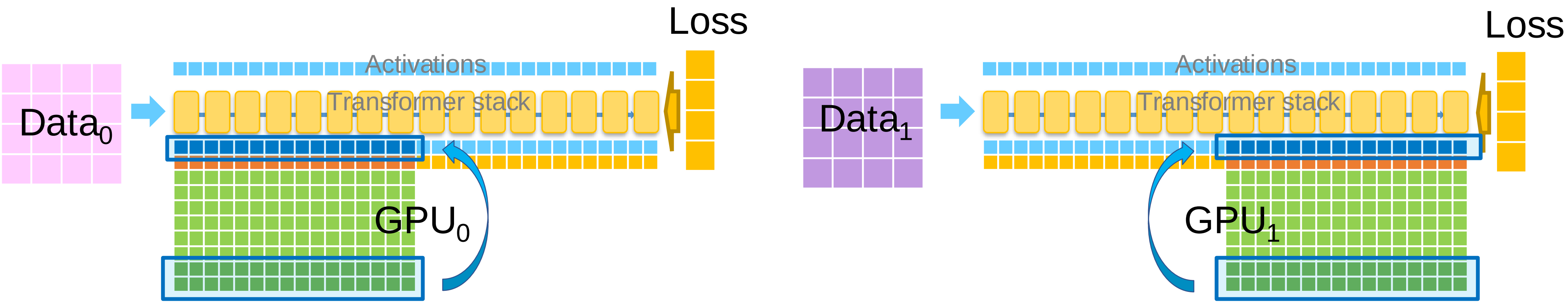
ZeRO Stage 1: Partitioning Optimizer States



- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and ReduceScatter to average
- Update the FP32 weights with ADAM optimizer

Adapted from Minjia Zhang, DeepSpeed Presentation

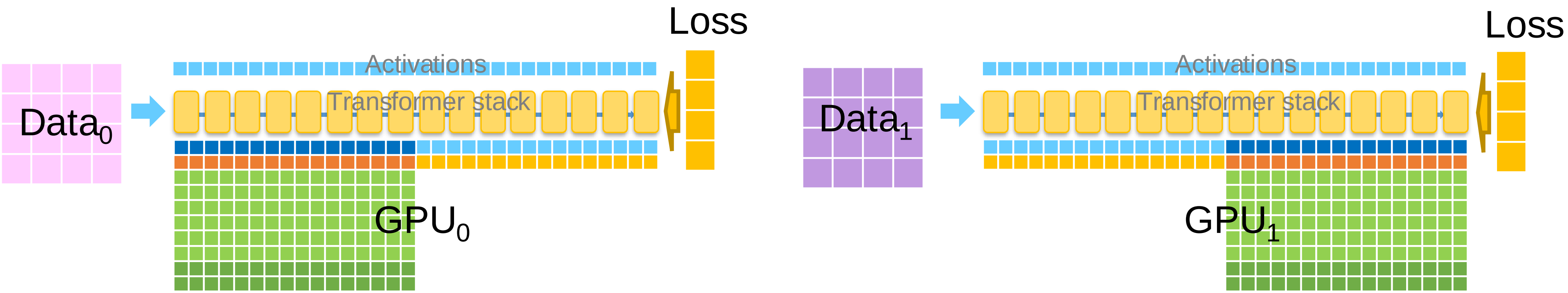
ZeRO Stage 1: Partitioning Optimizer States



- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and ReduceScatter to average
- Update the FP32 weights with ADAM optimizer
- Update the FP16 weights

Adapted from Minjia Zhang, DeepSpeed Presentation

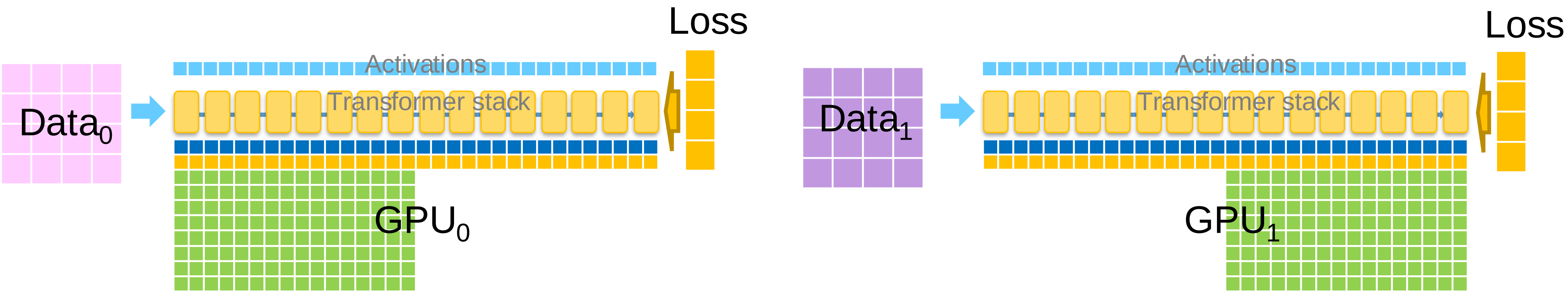
ZeRO Stage 1: Partitioning Optimizer States



- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and ReduceScatter to average
- Update the FP32 weights with ADAM optimizer
- Update the FP16 weights
- All Gather the FP16 weights to complete the iteration

Adapted from Minjia Zhang, DeepSpeed Presentation

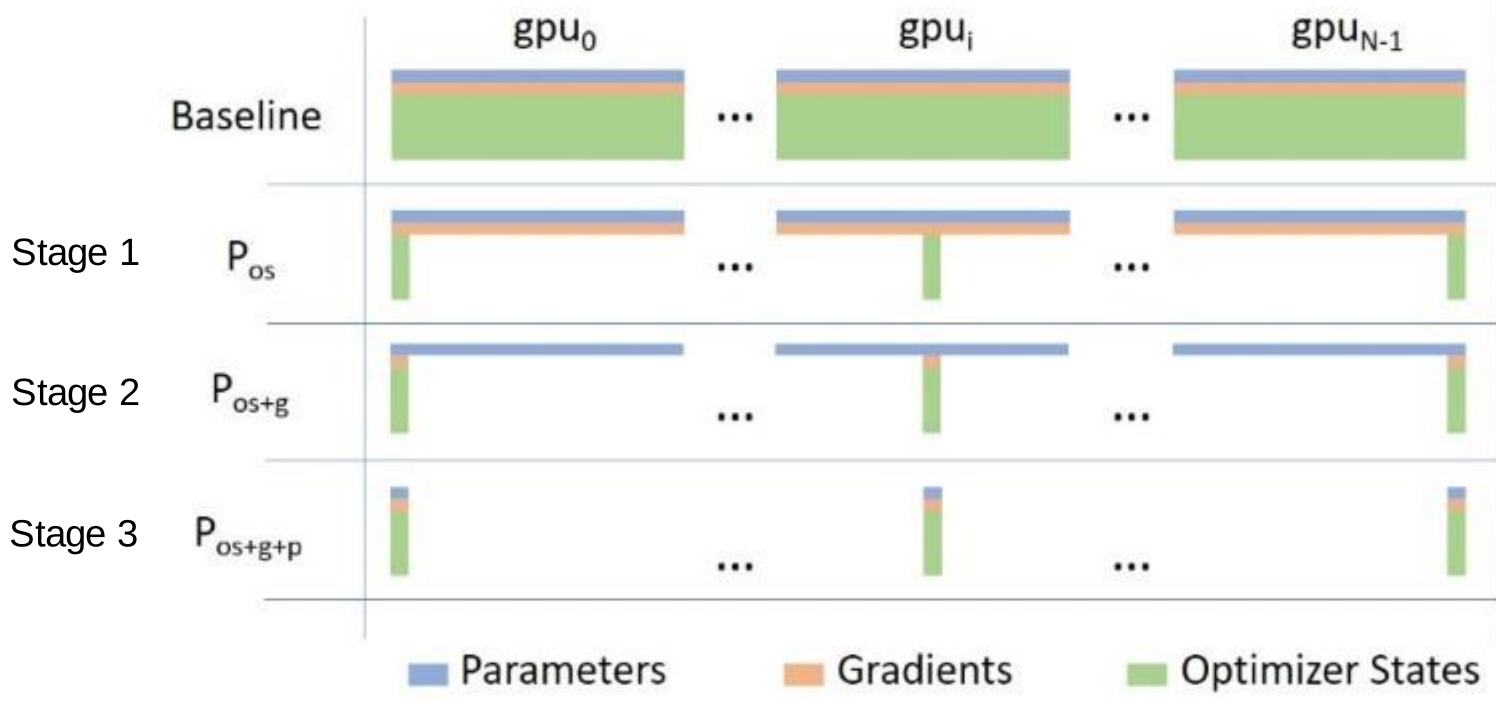
ZeRO Stage 1: Partitioning Optimizer States



- Run Forward across the transformer blocks
- Backward propagation to generate FP16 gradients and AllReduce to average
- Update the FP32 weights with ADAM optimizer
- Update the FP16 weights
- All Gather the FP16 weights to complete the iteration

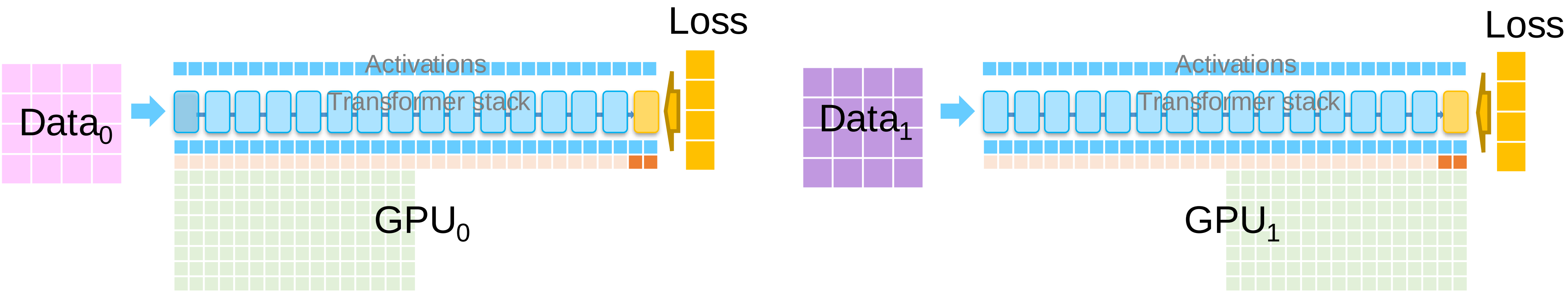
ZeRO: Zero Redundancy Optimizer

- Progressive memory savings and communication volume
- Turning NLR 17.2B is powered by Stage 1 and Megatron



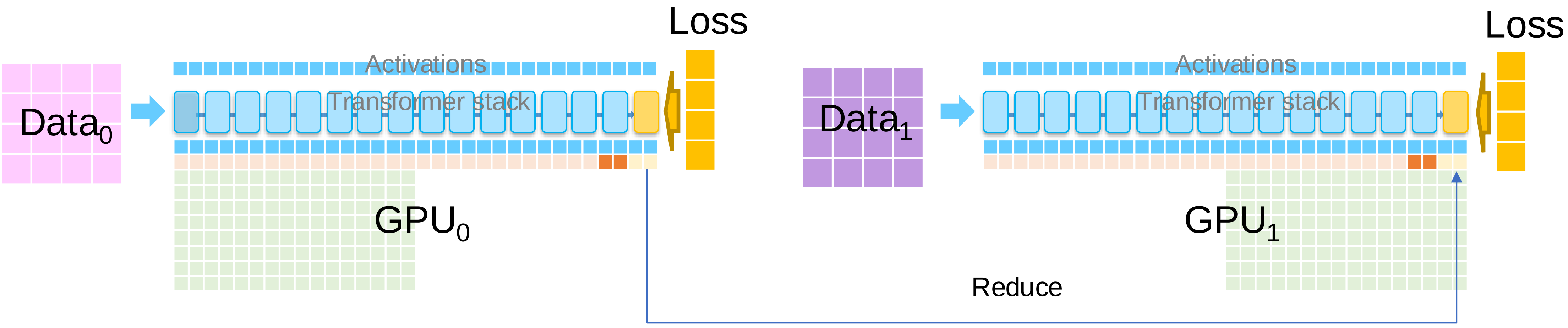
	Memory Consumption		Comm Volume
	Formulation	Specific Example $K=12$ $\Psi=7.5B$ $N_d=64$	
Baseline	$(2+2+K)*\Psi$	120GB	1x
Stage 1 P_{os}	$2\Psi+2\Psi+\frac{K*\Psi}{N_d}$	31.4GB	1x
Stage 2 P_{os+g}	$2\Psi+\frac{(2+K)*\Psi}{N_d}$	16.6GB	1x
Stage 3 P_{os+g+p}	$\frac{(2+2+K)*\Psi}{N_d}$	1.9GB	1.5x

ZeRO Stage 2: Partitioning Gradients



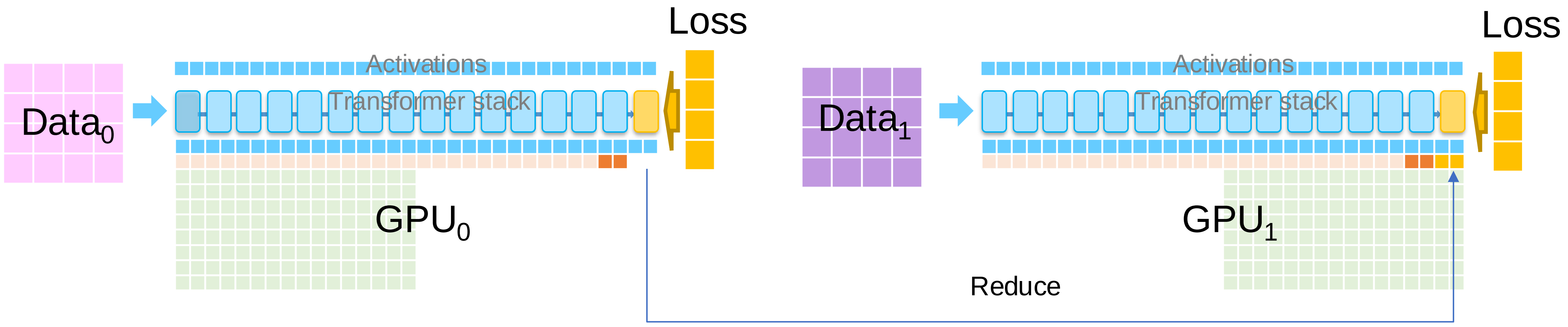
- Partitioning gradients across GPUs
- The forward process remains the same as stage 1

ZeRO Stage 2: Partitioning Gradients



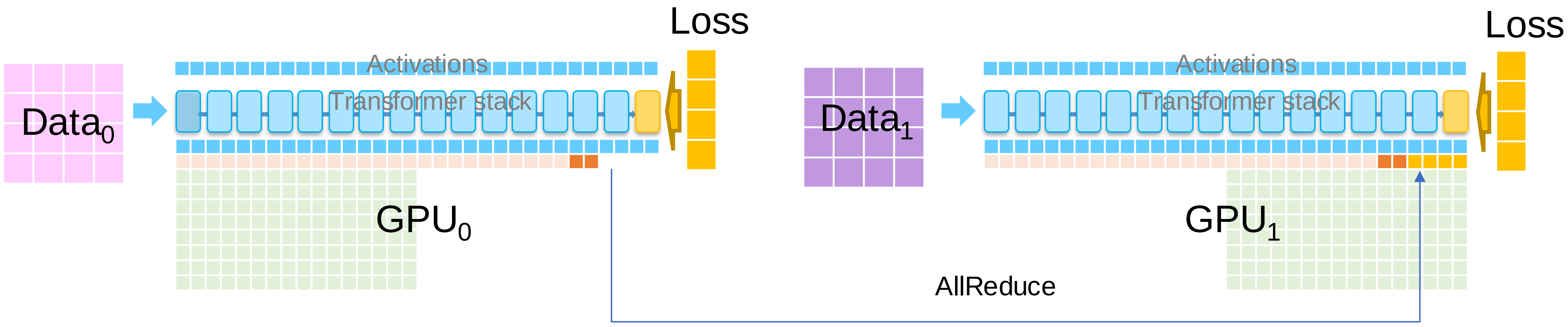
- Partitioning gradients across GPUs
- Perform Reduce right after back propagation of each layer

ZeRO Stage 2: Partitioning Gradients



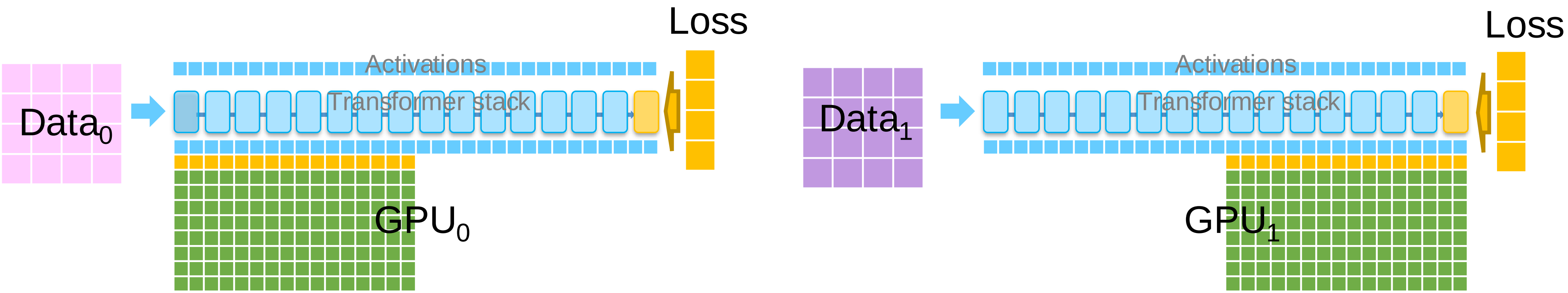
- Partitioning gradients across GPUs
- Only one GPU keeps the gradients after Reduce

ZeRO Stage 2: Partitioning Gradients



- Partitioning gradients across GPUs
- Reduce gradients on GPUs responsible for updating parameters

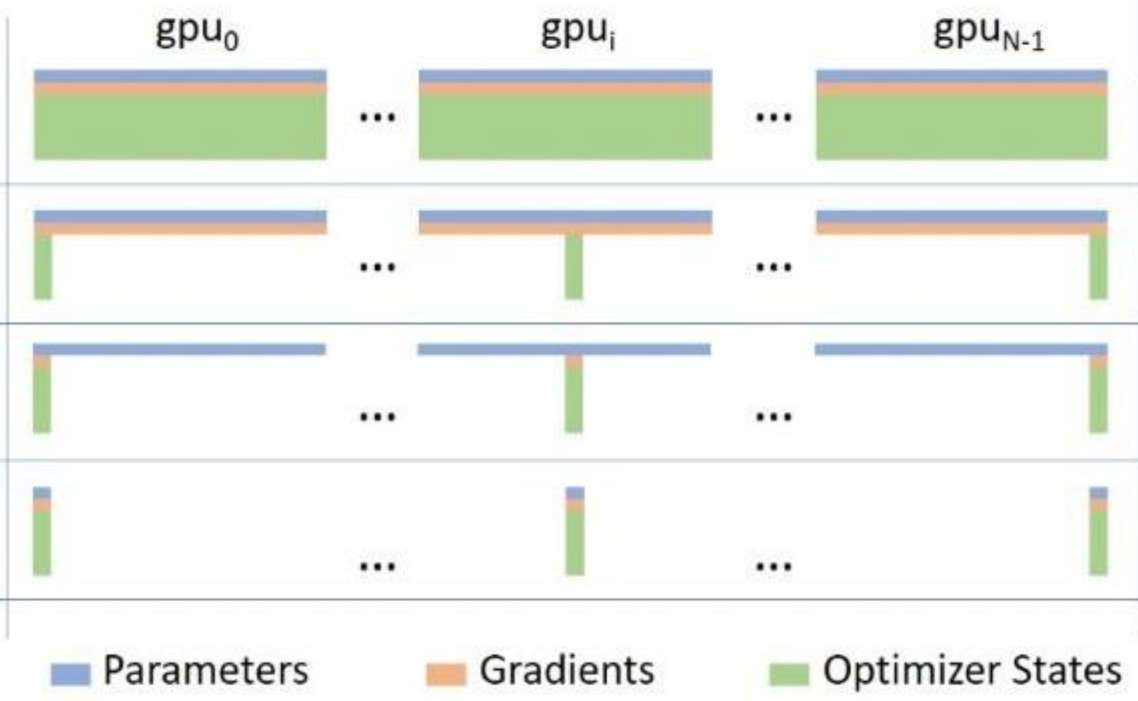
ZeRO Stage 2: Partitioning Gradients



- Partitioning gradients across GPUs
- Reduce gradients on GPUs responsible for updating parameters

ZeRO: Zero Redundancy Optimizer

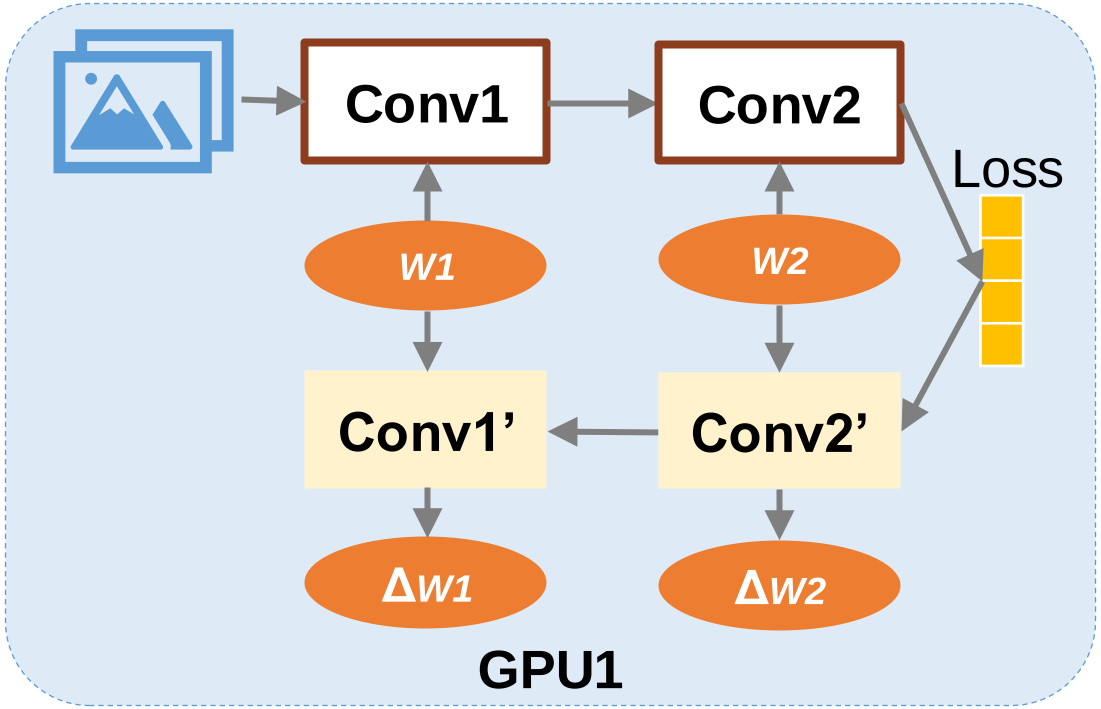
- Progressive memory savings and communication volume
- Turning NLR 17.2B is powered by Stage 1 and Megatron

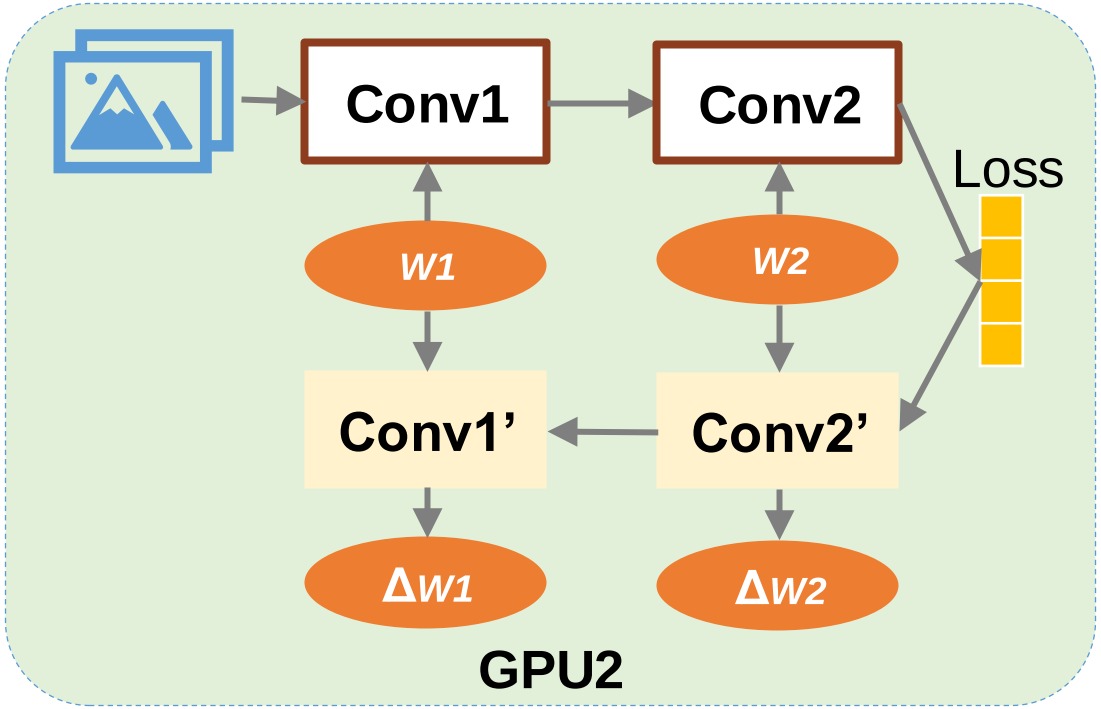


		Memory Consumption: Formulation	Memory Consumption: Specific Example ($K=12$ $\Psi=7.5B$ $N_d=64$)	Comm Volume
	Baseline	$(2+2+K)*\Psi$	120GB	1x
Stage 1	P_{os}	$2\Psi+2\Psi+\frac{K*\Psi}{N_d}$	31.4GB	1x
Stage 2	P_{os+g}	$2\Psi+\frac{(2+K)*\Psi}{N_d}$	16.6GB	1x
Stage 3	P_{os+g+p}	$\frac{(2+2+K)*\Psi}{N_d}$	1.9GB	1.5x

ZeRO Stage 3: Partitioning Parameters

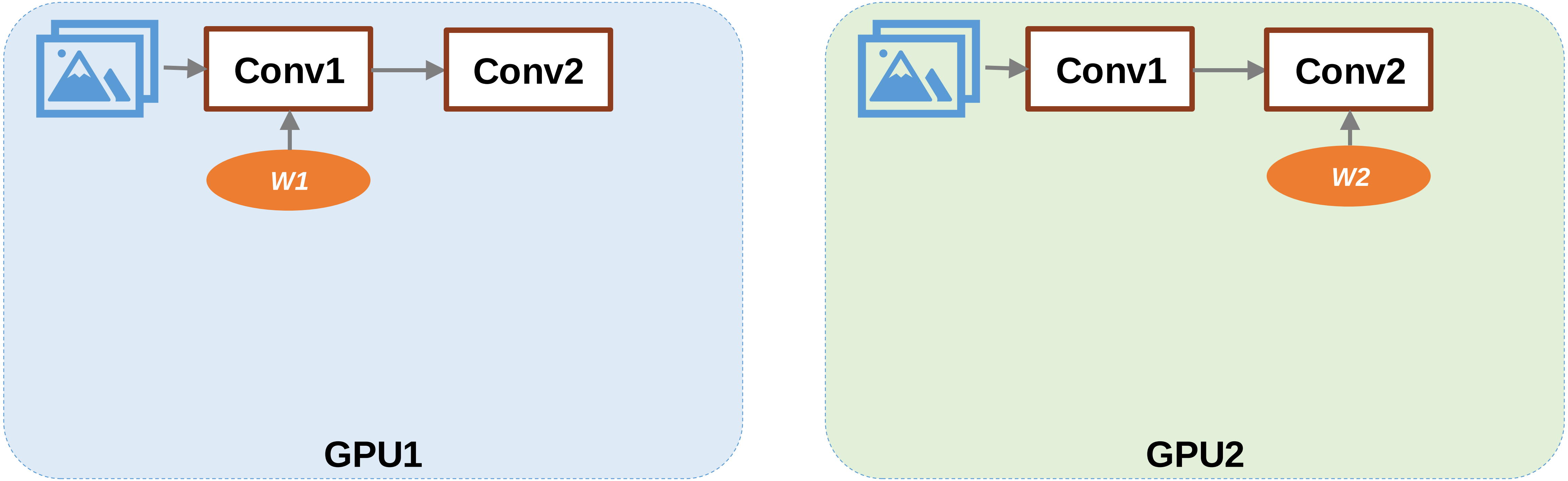
- In data parallel training, all GPUs keep **all** parameters during training





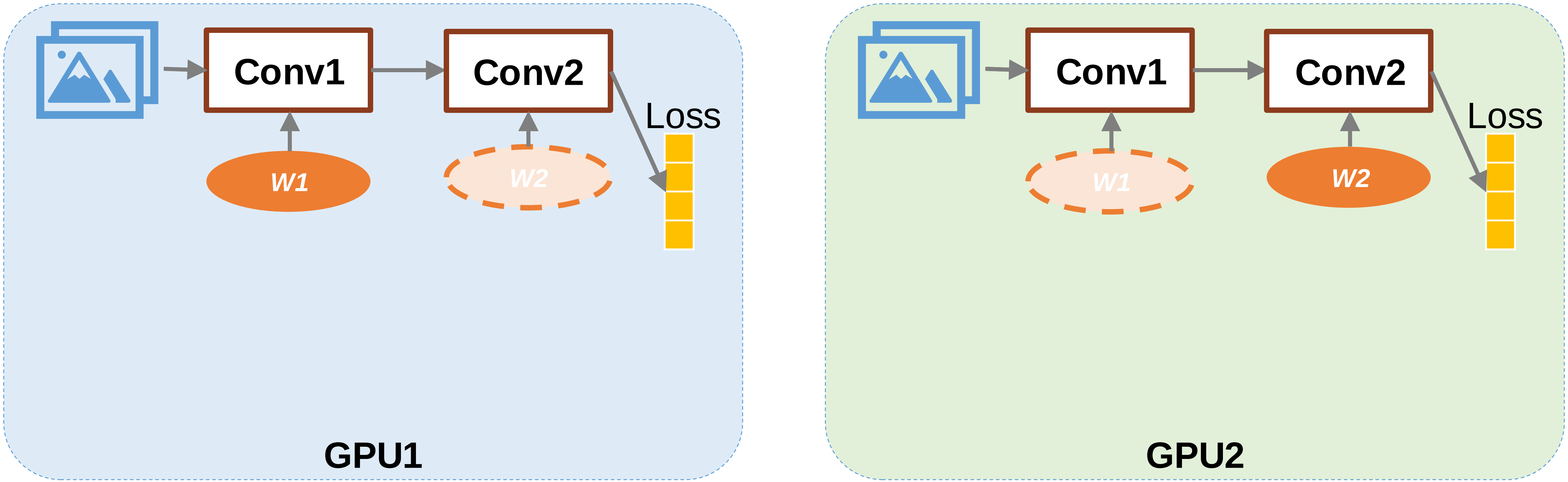
ZeRO Stage 3: Partitioning Parameters

- In ZeRO, model parameters are partitioned across GPUs



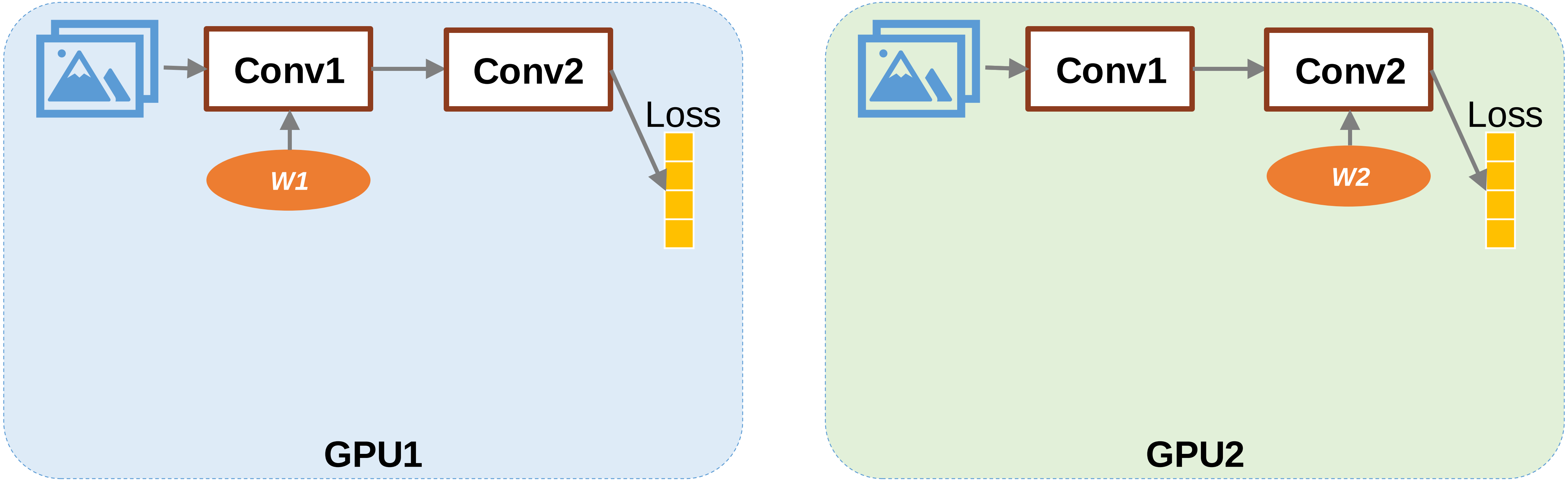
ZeRO Stage 3: Partitioning Parameters

- In ZeRO, model parameters are partitioned across GPUs
- GPUs broadcast their parameters during forward (allgather)



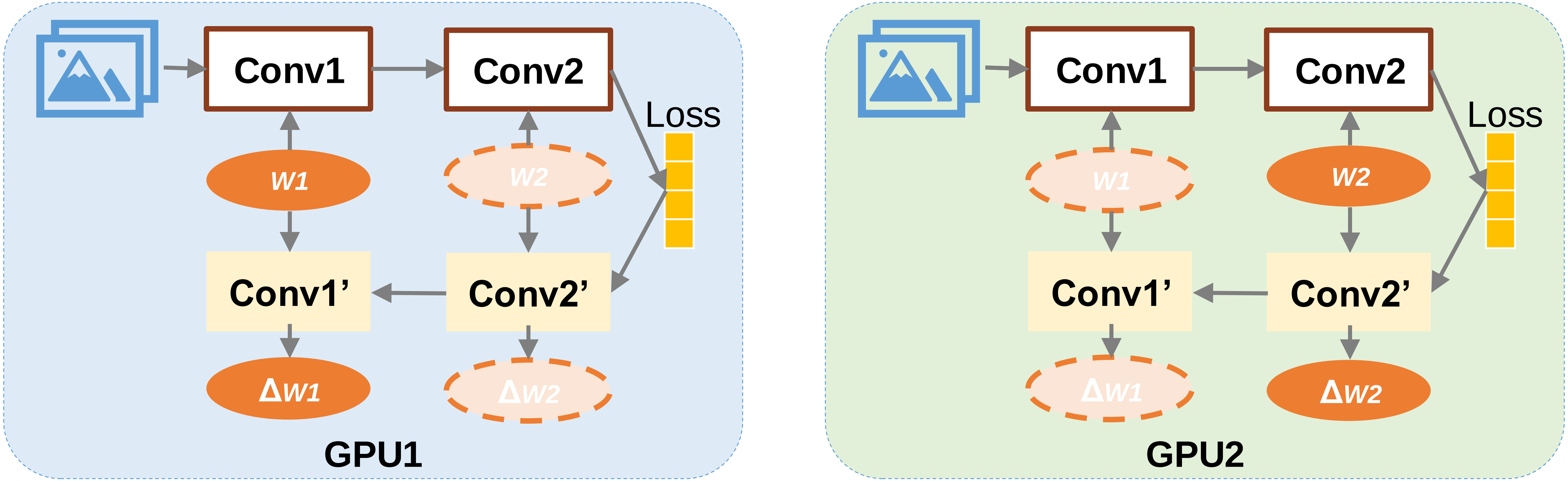
ZeRO Stage 3: Partitioning Parameters

- In ZeRO, model parameters are partitioned across GPUs
- Parameters are discarded right after use



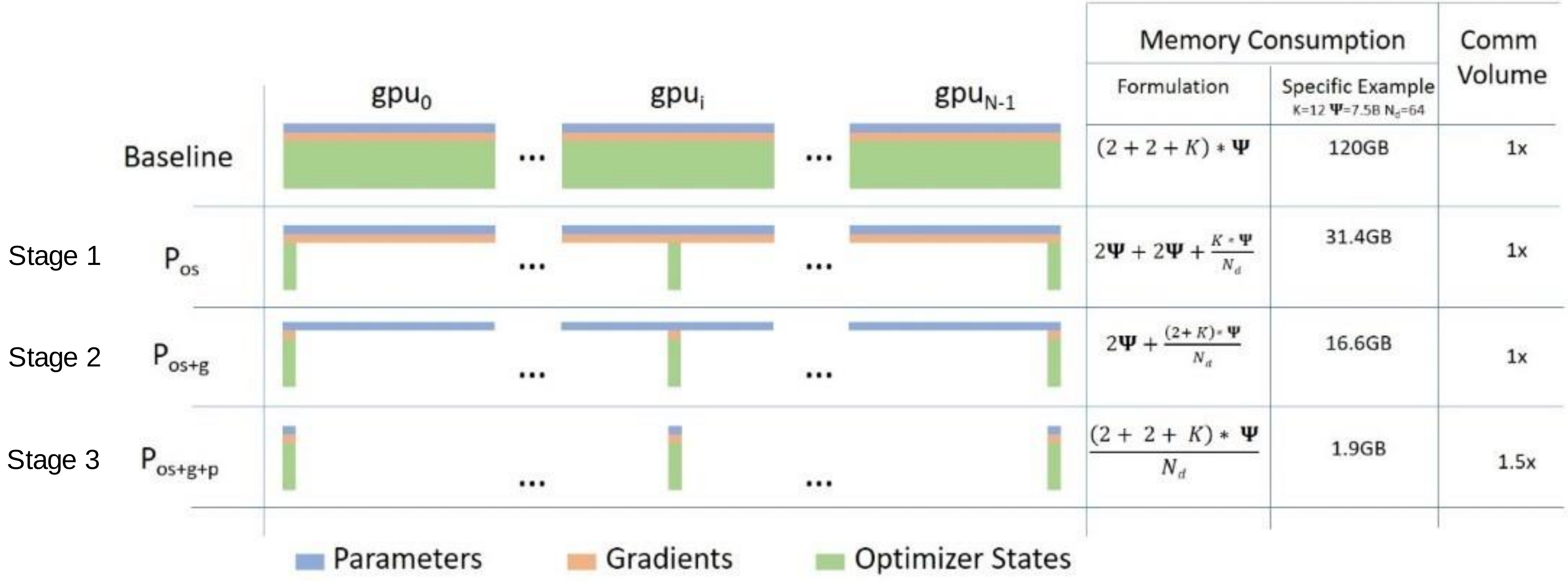
ZeRO Stage 3: Partitioning Parameters

- In ZeRO, model parameters are partitioned across GPUs
- GPUs broadcast their parameters again during backward

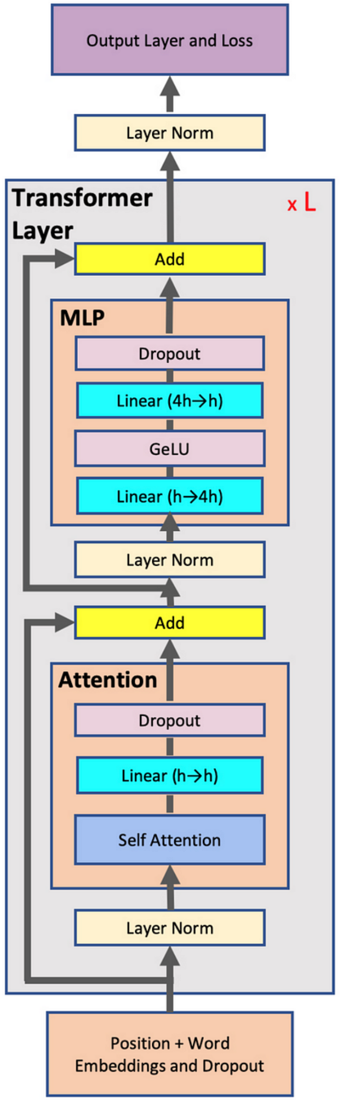


ZeRO: Zero Redundancy Optimizer

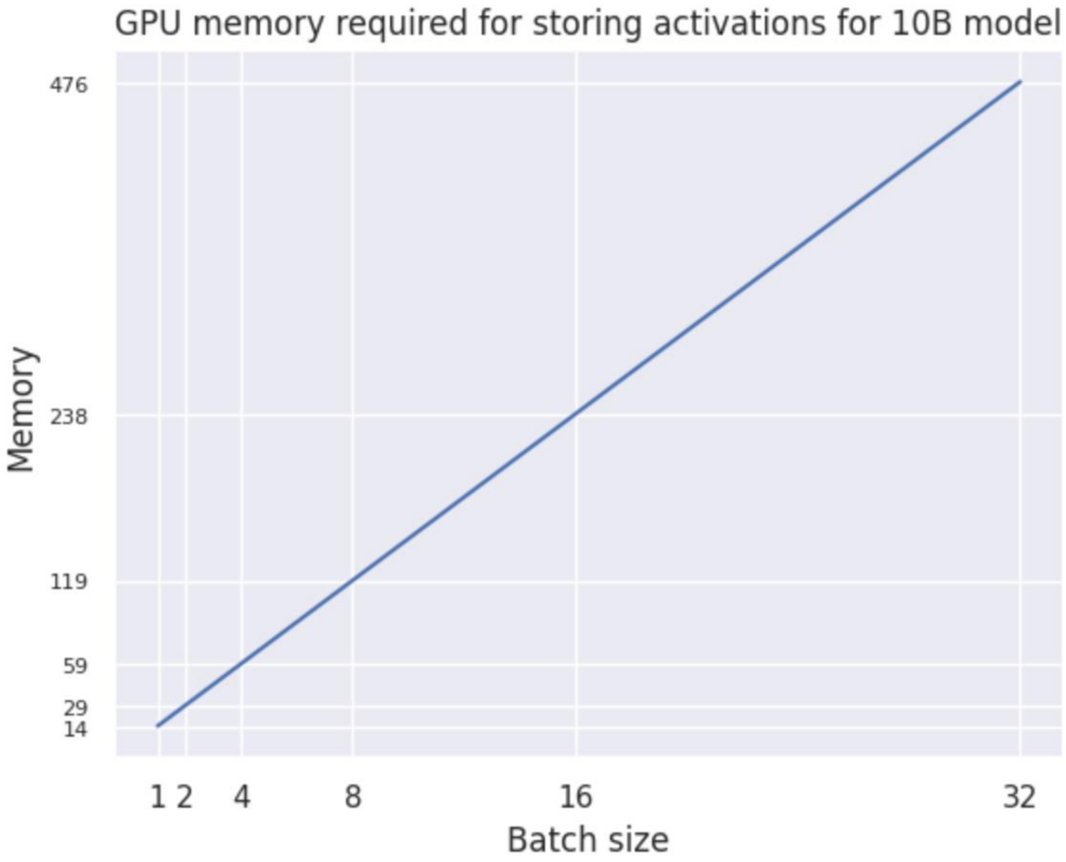
- ZeRO has three different stages
- Progressive memory savings and communication volume



Memory Footprint of Large Models – *Activations*



L - number of transformer layers
s - sequence length
b - batch size
h - hidden dimension size
a - number of attention heads
p - precision

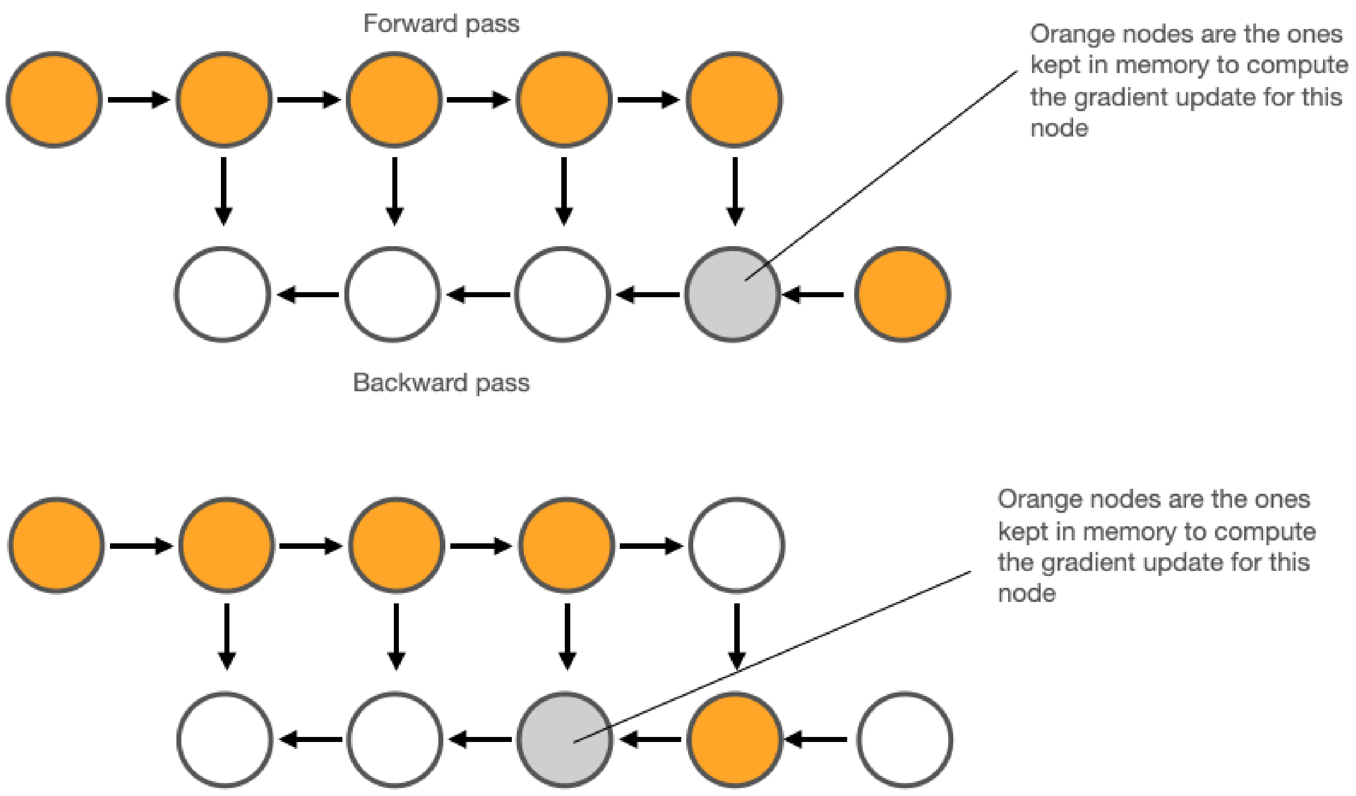


https://arxiv.org/pdf/2205.05198.pdf

Activation Stashing vs Checkpointing

- Stashing of activations is the baseline strategy here

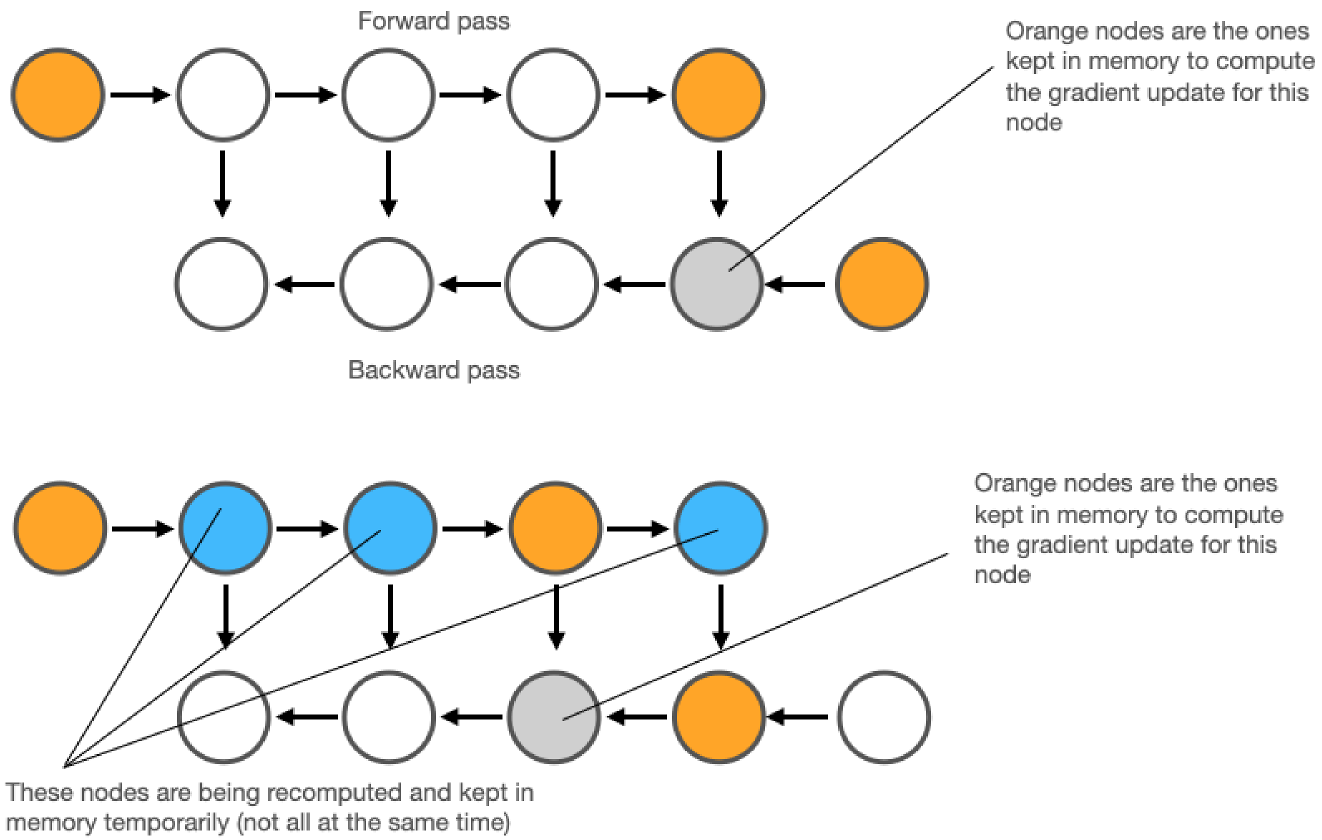
Drawing inspired by https://github.com/cybertronai/gradient-checkpointing



Activation Stashing vs Checkpointing

- Stashing of activations is the baseline strategy here
- Activation checkpointing is a technique used for reducing the memory footprint at the cost of more compute

Drawing inspired by https://github.com/cybertronai/gradient-checkpointing



Summary

- Forms of parallelism
  - Data parallelism
  - Model parallelism

- Memory optimizations in data parallelism
  - ZeRO: zero redundancy optimizer
  - Reduce activation storage by activation checkpointing