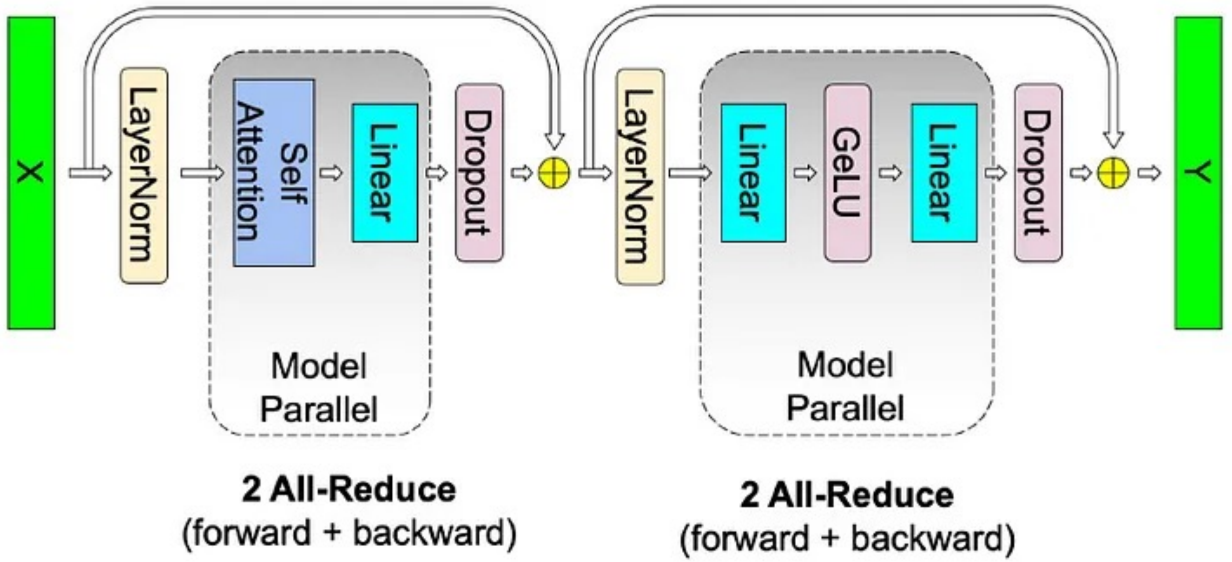
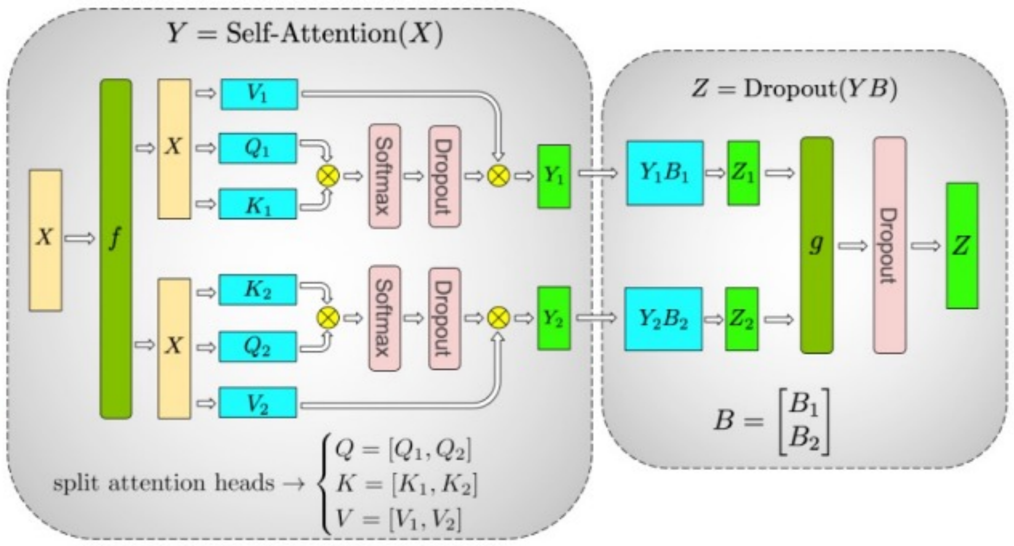
ZeRO: Memory Optimizations Toward Training Trillion Parameter Models

Zhe Su Yu-Chen Lin
Shuning Lin Kewen Zhao
CMU

Tensor Parallelism

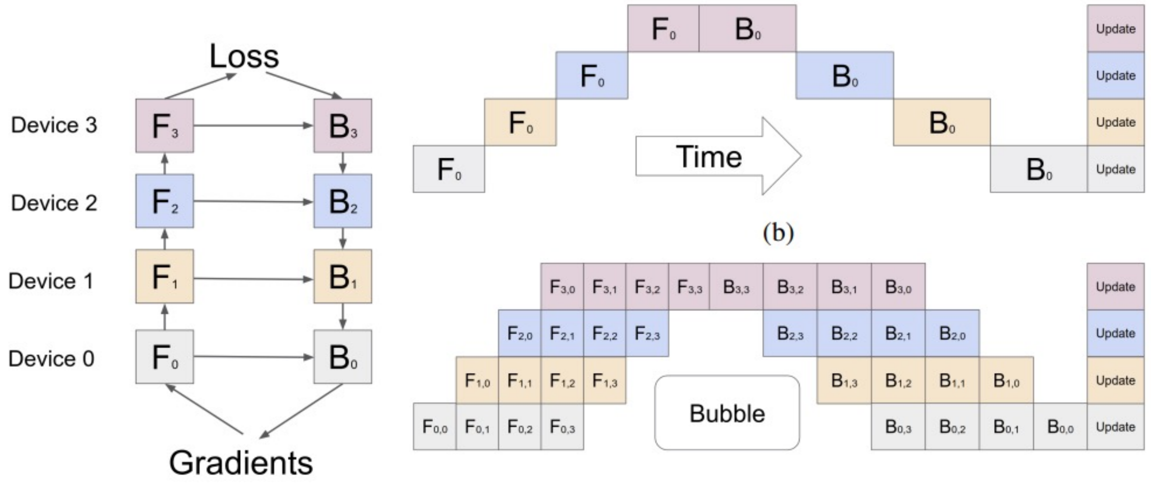
- Megatron-LM
  - Split the matrix into multiple parts and do matmul separately
  - No sync point within Linear and Self-attn



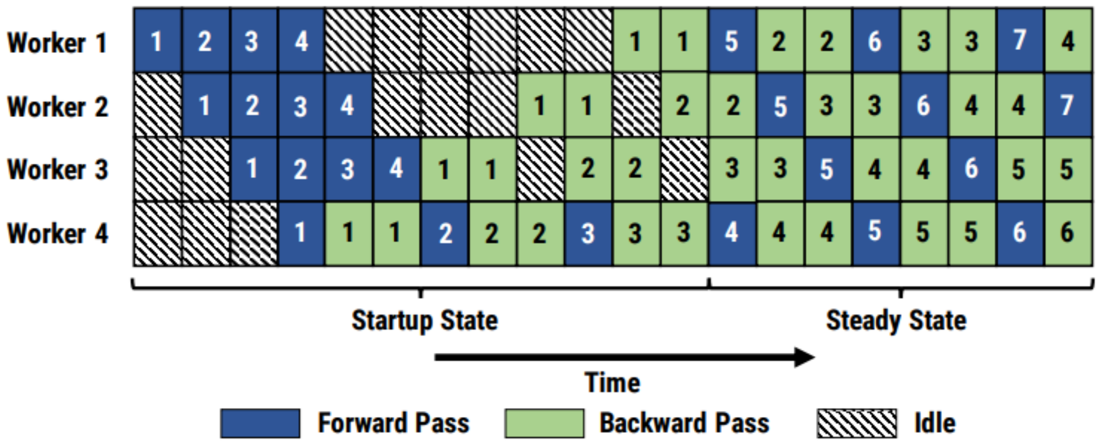


Pipeline Parallelism

- The model is distributed across multiple GPUs over layers.
- Devices can be idle while waiting for others
  - GPipe: divides data into smaller micro-batches. Has bubbles.
  - PipeDream: starts backward ASAP. Less bubbles.



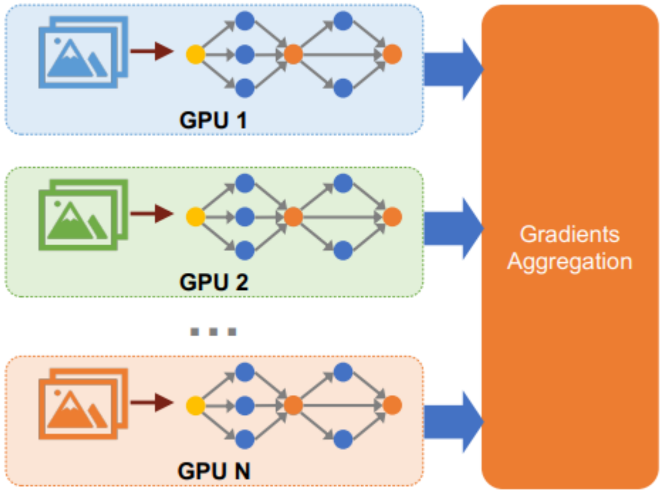
GPipe



PipeDream

Data Parallelism

- Each device has the same model and do forward and backward on a mini-batch separately. Quite easy and intuitive, but …
  - Cannot train LLM that cannot fit into one device
  - Each device has the whole replica of the model



Reduce

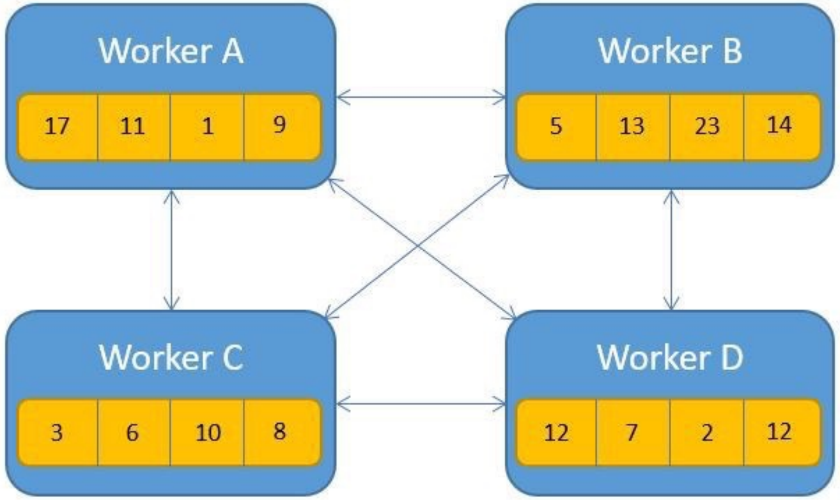
- Goal: In data parallelism, it is essential to ensure that each device is updated coherently, therefore we need to aggregate (reduce) gradients across different devices.

- Centralized Reduce: all workers communicate with parameter servers for weights update; cannot scale to large numbers of workers

- All Reduce
  - Naïve AllReduce
  - Ring AllReduce

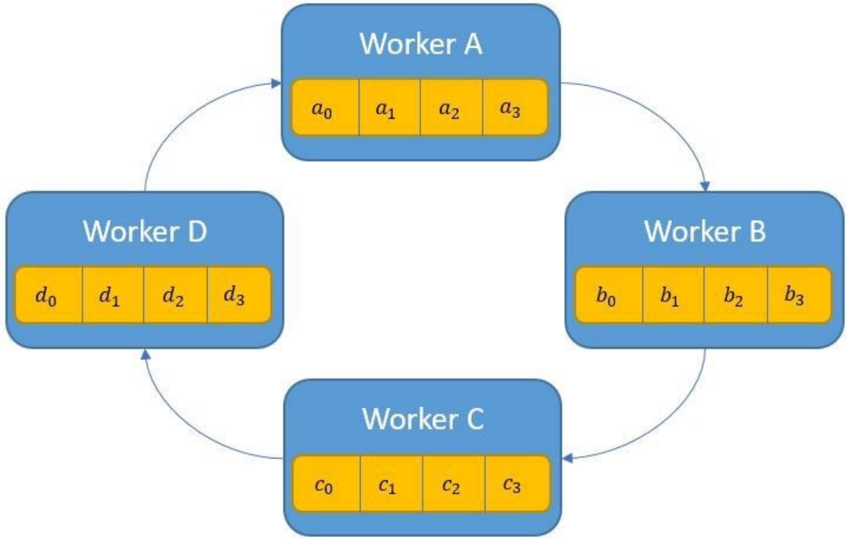
Naïve AllReduce

- Each worker can send its local gradients to other workers
- N workers, each M params, overall N * (N-1) * M params
- Issue: each worker communicates with all other workers; same scalability issue as parameter server



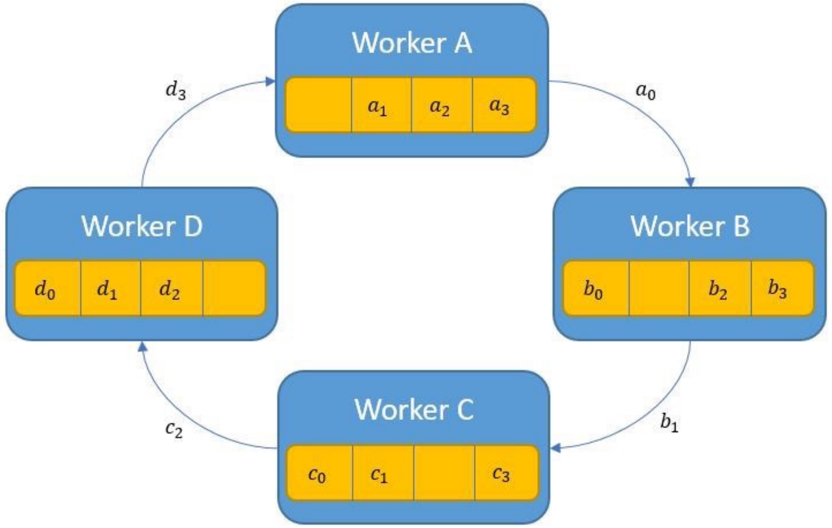
Ring AllReduce

- Construct a ring of N workers
- divide M parameters into N slices
- Step 1 (Aggregation): each worker send one slice (M/N parameters) to the next worker on the ring; repeat N times



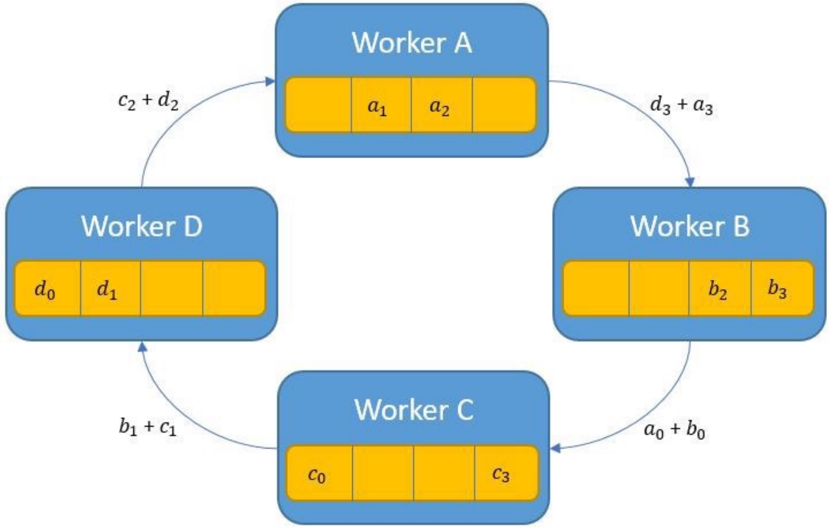
Ring AllReduce

- Construct a ring of N workers
- divide M parameters into N slices
- Step 1 (Aggregation): each worker send one slice (M/N parameters) to the next worker on the ring; repeat N times



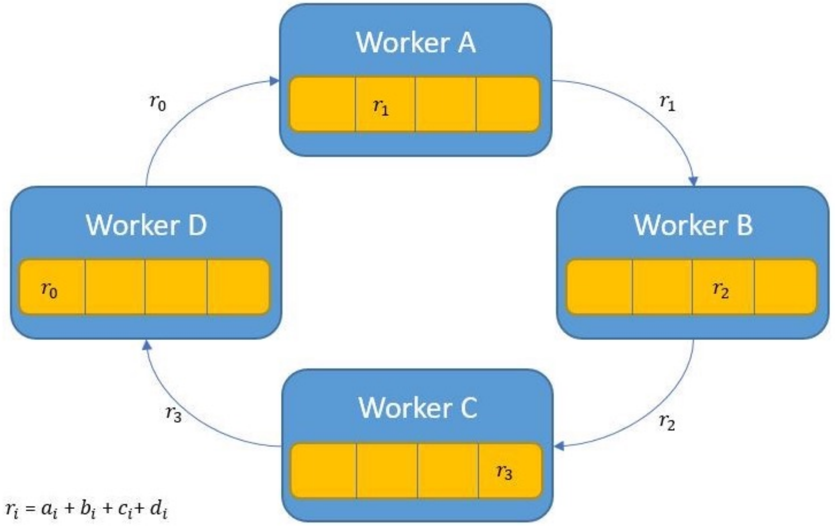
Ring AllReduce

- Construct a ring of N workers
- divide M parameters into N slices
- Step 1 (Aggregation): each worker send one slice (M/N parameters) to the next worker on the ring; repeat N times



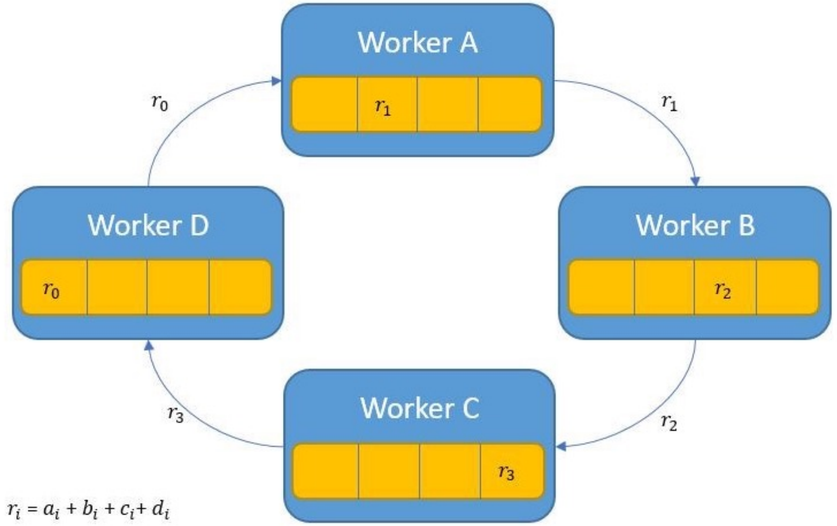
Ring AllReduce

- Construct a ring of N workers
- divide M parameters into N slices
- After step 1, each worker has the aggregated version of M/N parameters



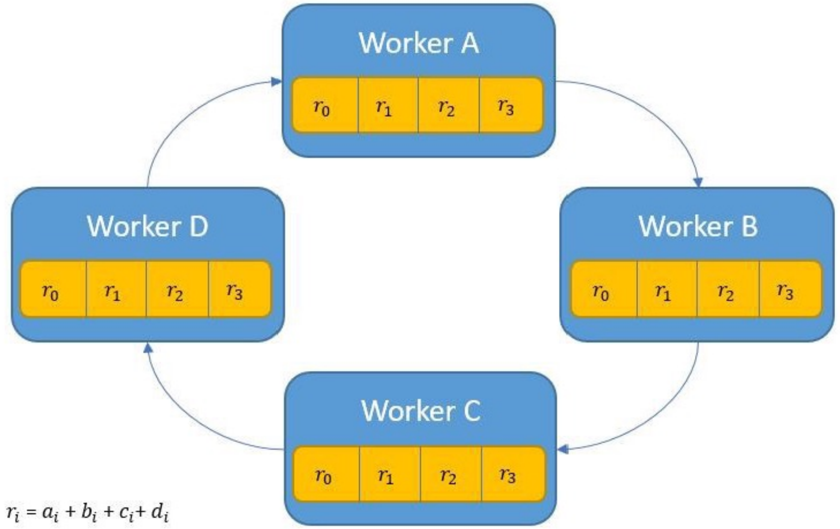
Ring AllReduce

- Construct a ring of N workers
- divide M parameters into N slices
- Step 2 (Broadcast): each worker send one slice of aggregated parameters to the next worker; repeat N times



Ring AllReduce

- Construct a ring of N workers
- divide M parameters into N slices
- Step 2 (Broadcast): each worker send one slice of aggregated parameters to the next worker; repeat N times



Ring AllReduce

- Overall communication: 2 * M * N parameters
  - Aggregation: M * N parameters
  - Broadcast: M * N parameters

Summary

- Model Parallelism
  - Pros: Good memory efficiency
  - Cons: Poor compute /communication efficiency (5% of peak perf in training 40B model with Megatron)
- Data parallelism
  - Pros: Good compute/communication efficiency
  - Cons: Poor memory efficiency (Every device has one copy of model)

Memory Usage

- The GPUs need to store model weights, forward activation, backward gradient, optimizer state

- Common methods in optimization: Adam + Mixed-precision
  - Optimizer States: Momentum + Variance
  - Model: Parameters and Gradients

while θ_t not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)

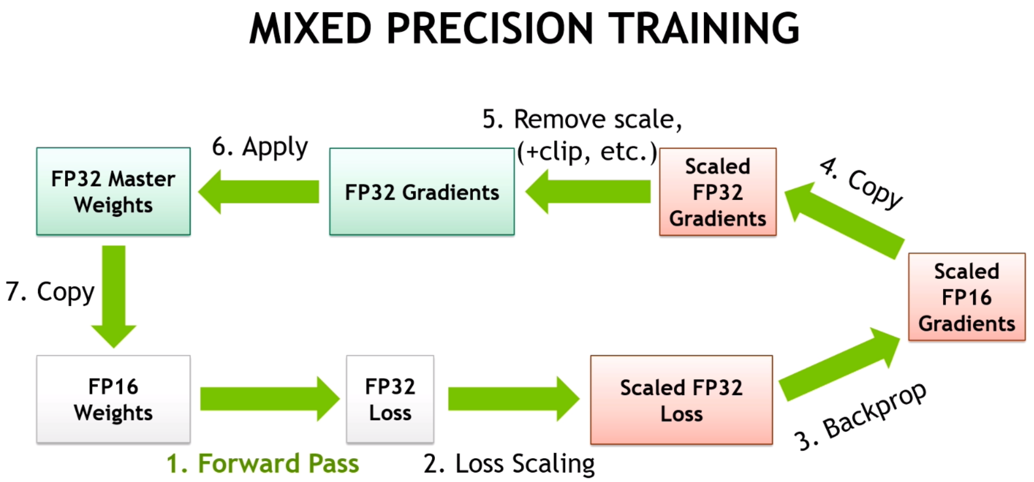
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)

$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)

end while

Adam Optimizer



Memory Usage

- adam optimizer, mixed-precision training, N params
  - FP32 master parameters: 4N Bytes
  - FP32 optimizer states: 4N * 2 Bytes (Momentum and Variance)
  - FP16 model parameters: 2N Bytes
  - FP16 optimizer states: 2N Bytes (Momentum only)
  - 16N Bytes in total
- For 1.5B GPT-2, 24GB vMem
- For 175B GPT-3, 2800GB vMem

Memory Usage

- Example: GPT-2 w/ 1.5B parameters
  - FP32 master parameters: 6G Bytes
  - FP32 optimizer states: 12G Bytes (Momentum and Variance)
  - FP16 model parameters: 3G Bytes
  - FP16 optimizer states: 3G Bytes (Momentum only)
  - 24G Bytes in total
- For 1.5B GPT-2, 24GB vMem
- For 175B GPT-3, 2800GB vMem

Memory Usage

- Example: GPT-3 w/ 175B parameters
  - FP32 master parameters: 700G Bytes
  - FP32 optimizer states: 1400G Bytes (Momentum and Variance)
  - FP16 model parameters: 350G Bytes
  - FP16 optimizer states: 350G Bytes (Momentum only)
  - 2800G Bytes in total

Other Memory Usages

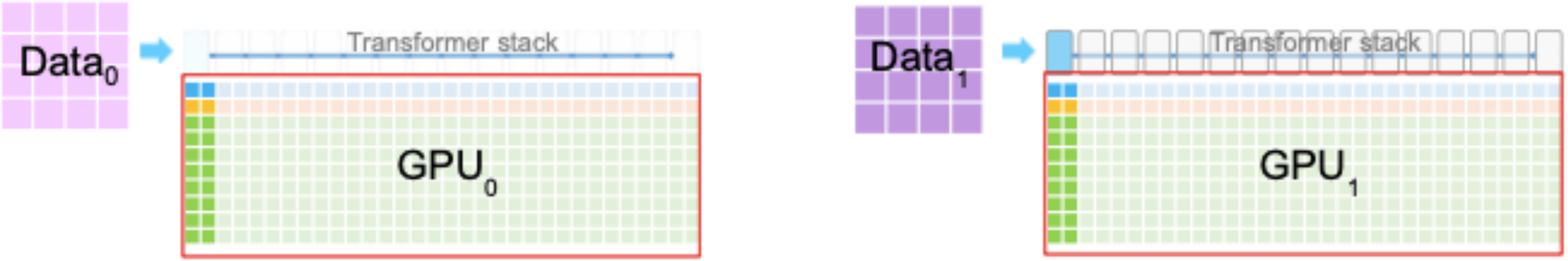
- Temporary Buffers:
  - Storing intermediate results.
  - Operations such as gradient norm computation tend to fuse all the gradients into a single flattened buffer before applying the operation in an effort to improve throughput.
- Memory Fragmentation
  - In extreme cases can be 30%.

Memory Consumption

- Suppose there are
  - Two data splits: Data0 and Data1
  - Two GPUs: GPU0 and GPU1
  - 16 layer Transformer Model

Memory Consumption

- Each cell represents GPU memory used by the corresponding transformer layer

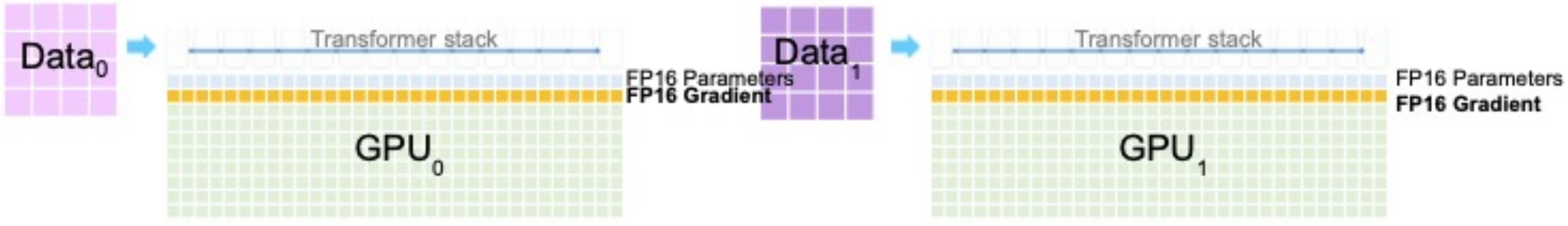


Memory Consumption

- Each cell represents GPU memory used by the corresponding transformer layer
  - FP16 parameters
  - FP16 Gradients
  - FP32 Optimizer States (Gradients, Variance, Momentum, Parameters)

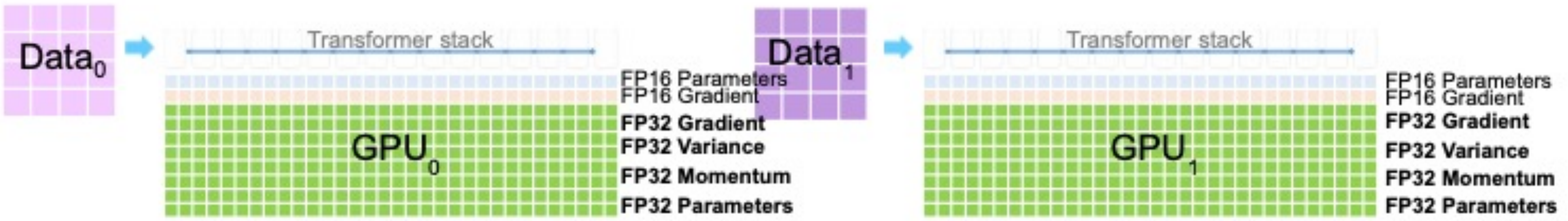
Memory Consumption

- Each cell represents GPU memory used by the corresponding transformer layer
  - FP16 parameters
  - **FP16 Gradients**
  - FP32 Optimizer States (Gradients, Variance, Momentum, Parameters)



Memory Consumption

- Each cell represents GPU memory used by the corresponding transformer layer
  - FP16 parameters
  - FP16 Gradients
  - FP32 Optimizer States (Gradients, Variance, Momentum, Parameters)



Common Approaches to Reduce Memory

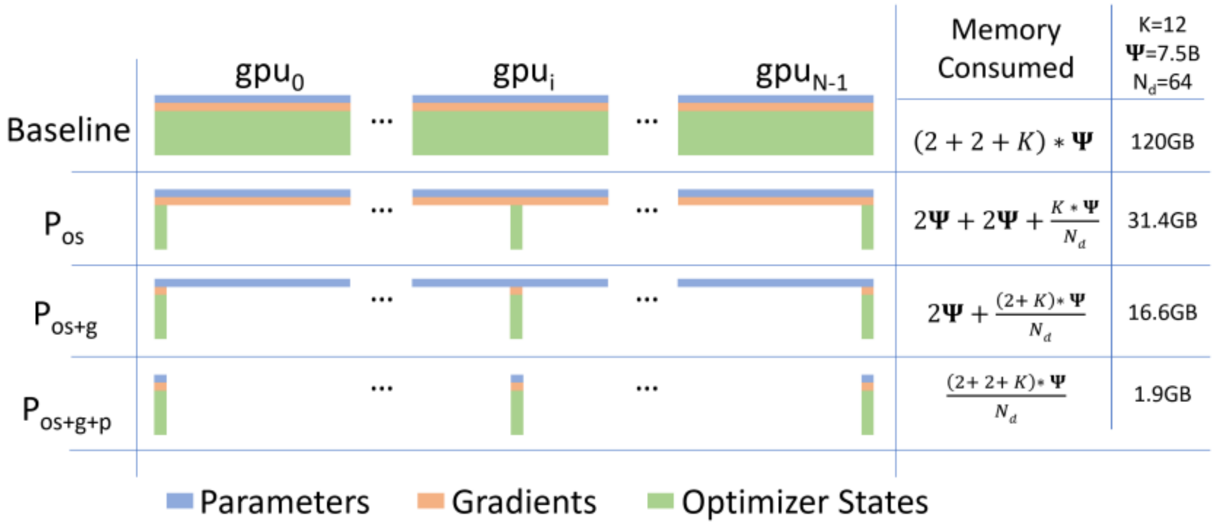
- Reducing Activation Memory
  - Activation Checkpoint, Compression
  - All Work in parallel with ZeRO

- CPU Offload
  - Requires CPU-GPU-CPU transfer, which can take 50% time

- Memory Efficient Optimizer
  - Maintaining coarser-grained stats of model params and gradients
  - Works in parallel with ZeRO

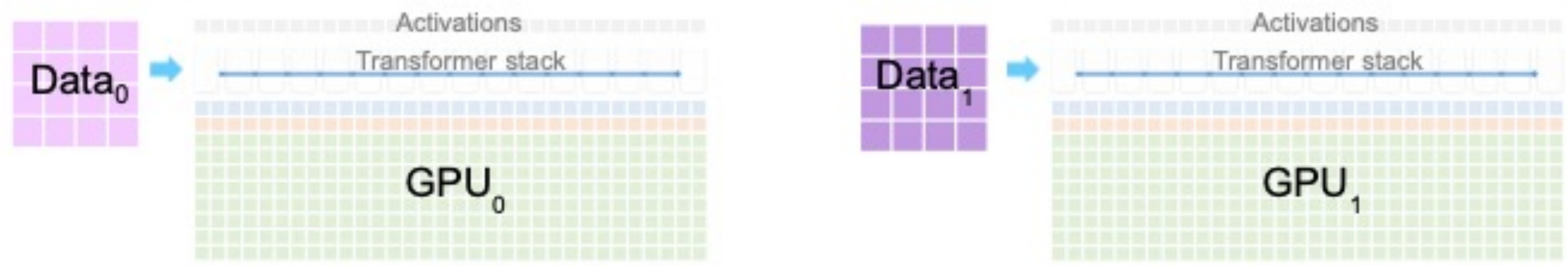
ZeRO - Zero Redundancy Optimizer

- Work done by Microsoft, implemented in Deepspeed.
- Features:
  - Eliminating data redundancy in data parallel training
  - Can be widely used in large language model training



ZeRO 1: Partitioning Optimizer States

- Question: How can we partition optimizer states?



ZeRO 1: Partitioning Optimizer States

- forward pass to produce activations and loss (by fp16 parameters)

ZeRO 1: Partitioning Optimizer States

- forward pass to produce activations and loss (by fp16 parameters)

ZeRO 1: Partitioning Optimizer States

- forward pass to produce activations and loss (by fp16 parameters)

ZeRO 1: Partitioning Optimizer States

- forward pass to produce activations and loss (by fp16 parameters)

ZeRO 1: Partitioning Optimizer States

- loss backward to calculate fp16 gradients

ZeRO 1: Partitioning Optimizer States

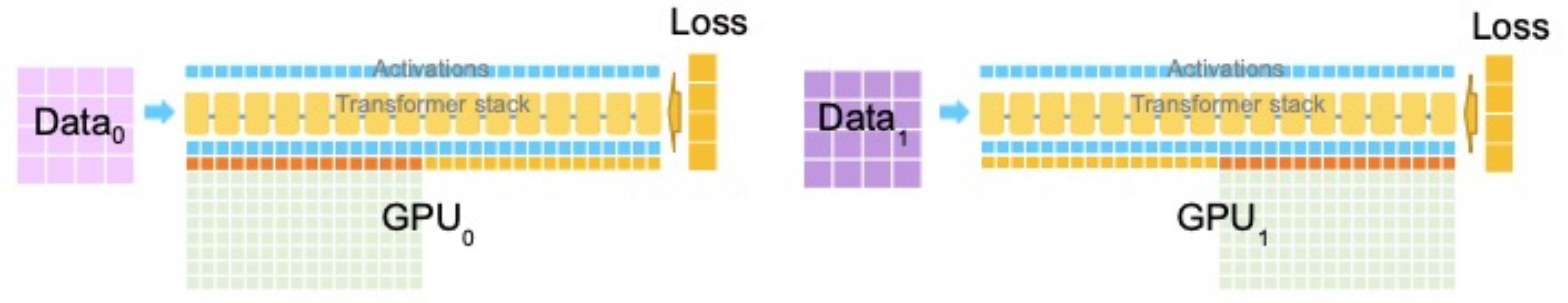
- loss backward to calculate fp16 gradients

ZeRO 1: Partitioning Optimizer States

- loss backward to calculate fp16 gradients

ZeRO 1: Partitioning Optimizer States

- gradient gathering from another GPU and average gradient calculation



ZeRO 1: Partitioning Optimizer States

- fp32 gradient update

ZeRO 1: Partitioning Optimizer States

- fp32 variance update

ZeRO 1: Partitioning Optimizer States

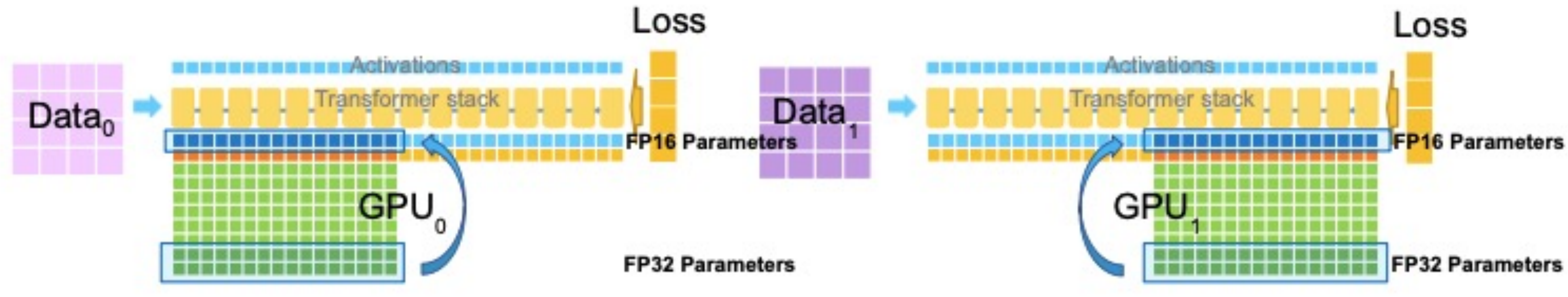
- fp32 momentum update

ZeRO 1: Partitioning Optimizer States

- fp32 parameters update

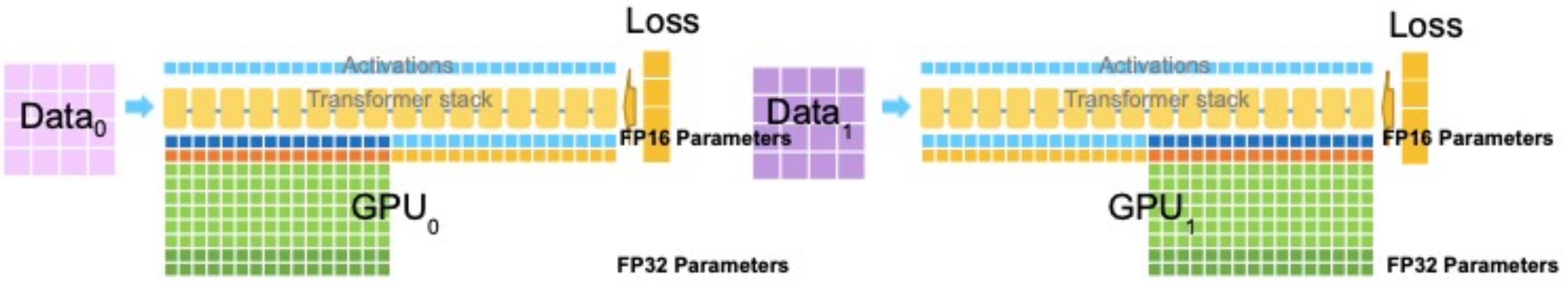
ZeRO 1: Partitioning Optimizer States

- copy fp32 parameters to fp16 parameters



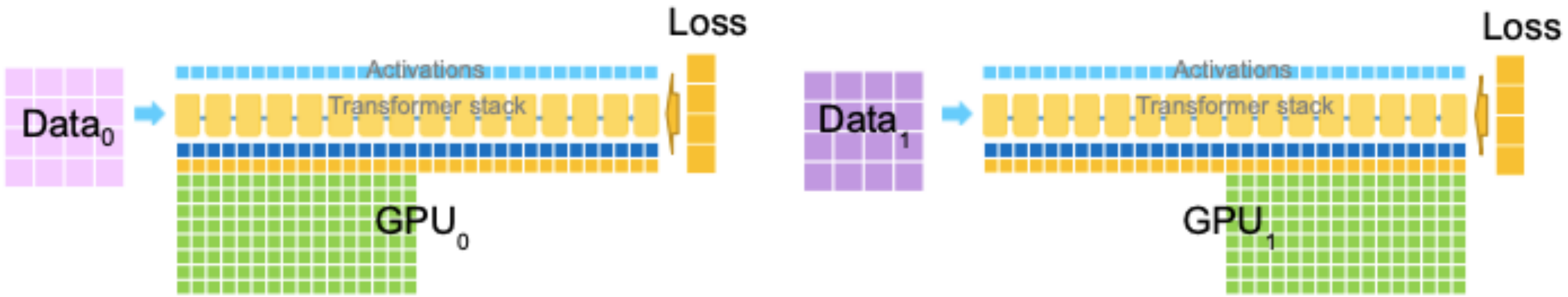
ZeRO 1: Partitioning Optimizer States

- fp16 parameters ready



ZeRO 1: Partitioning Optimizer States

- all gather the fp16 weights to complete the iteration



ZeRO 2: Partition Gradients

- Key idea:
  - Each GPU is only needs to store one partition of gradients instead of all gradients
  - However, each GPU is responsible for different data, meaning it still needs to compute all the gradients, although it only needs to store one partition

ZeRO 2: Partition Gradients

- Key idea:
  - For the gradients out of its responsibility, the GPU passes those gradients(computed with its own data) to the GPU responsible for those gradients.
  - The result is memory usage for gradients reduced by Nd times. (Nd = # of GPUs)

ZeRO 2: Partition Gradients

- The backward pass starts
- GPU 0,1,2 hold temporary buffers for the gradients that GPU 3 is responsible for (M3)

ZeRO 2: Partition Gradients

- GPU 0,1,2 pass the M3 gradients to GPU 3

ZeRO 2: Partition Gradients

- Then they delete M3 gradients, GPU 3 will keep M3 gradients

ZeRO 2: Partition Gradients

- GPU 0,2,3 hold temporary buffers for the gradients that GPU 2 is responsible for (M2)

ZeRO 2: Partition Gradients

- GPU 0,2,3 pass the M2 gradients to GPU 2

ZeRO 2: Partition Gradients

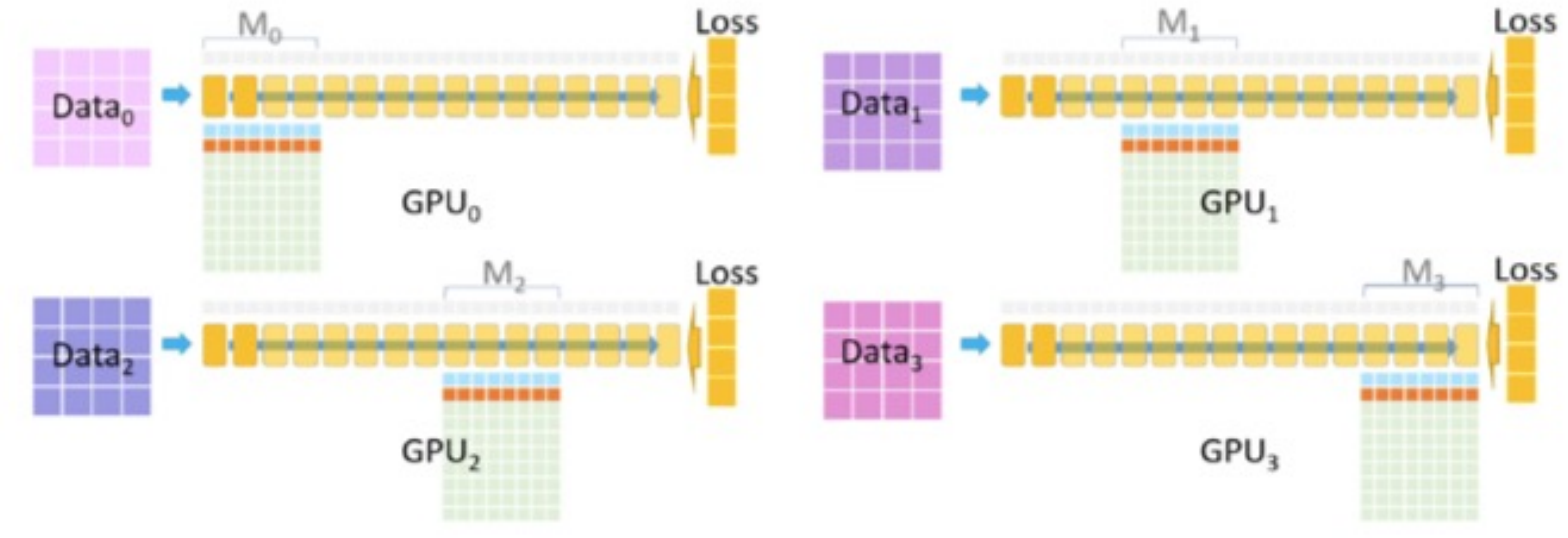
- Then they delete M2 gradients, GPU 2 will keep M2 gradients

ZeRO 2: Partition Gradients

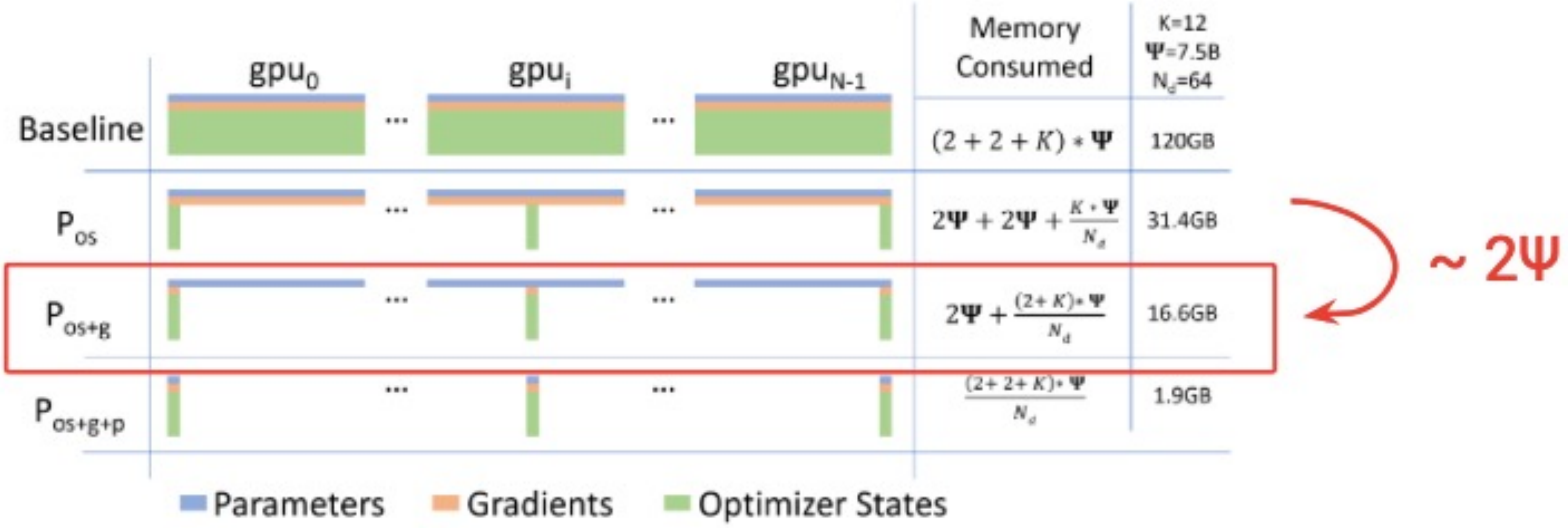
- Same thing for GPU1/M1

ZeRO 2: Partition Gradients

- Same thing for GPU0/M0



ZeRO 2: Partition Gradients

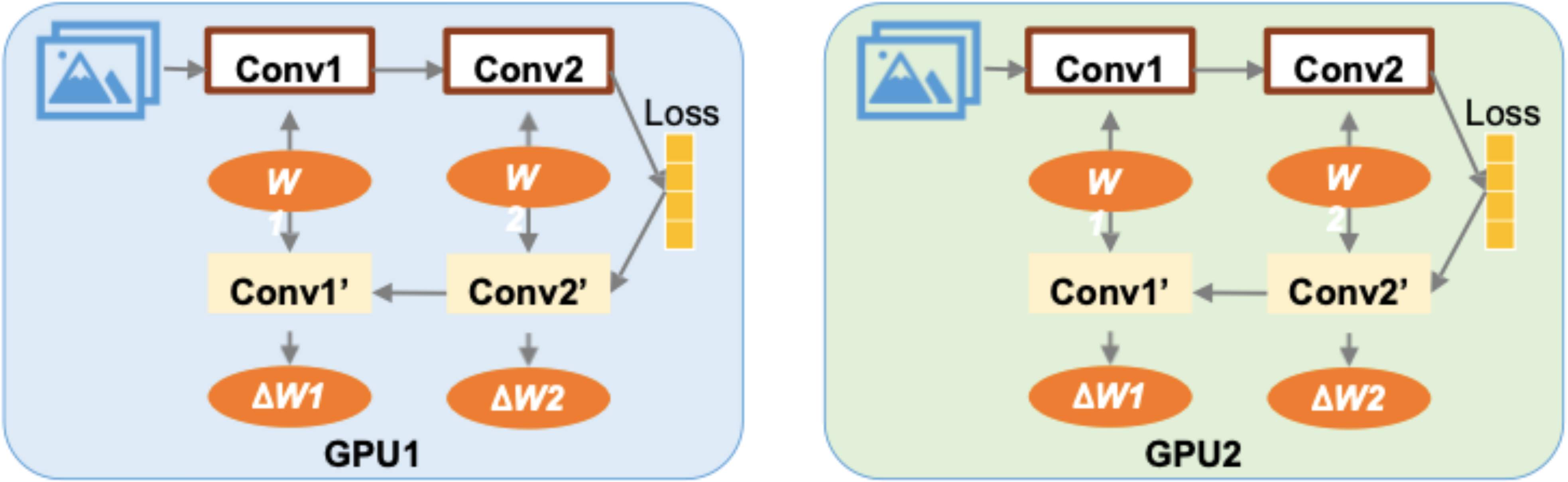


ZeRO 2: Partition Gradients

DP	7.5B Model (GB)			128B Model (GB)			1T Model (GB)		
	P_{os}	P_{os+g}	P_{os+g+p}	P_{os}	P_{os+g}	P_{os+g+p}	P_{os}	P_{os+g}	P_{os+g+p}
1	120	120	120	2048	2048	2048	16000	16000	16000
4	52.5	41.3	**30**	896	704	512	7000	5500	4000
16	35.6	**21.6**	7.5	608	368	128	4750	2875	1000
64	**31.4**	16.6	1.88	536	284	**32**	4187	2218	250
256	30.4	15.4	0.47	518	263	8	4046	2054	62.5
1024	30.1	15.1	0.12	513	257	2	4011	2013	**15.6**

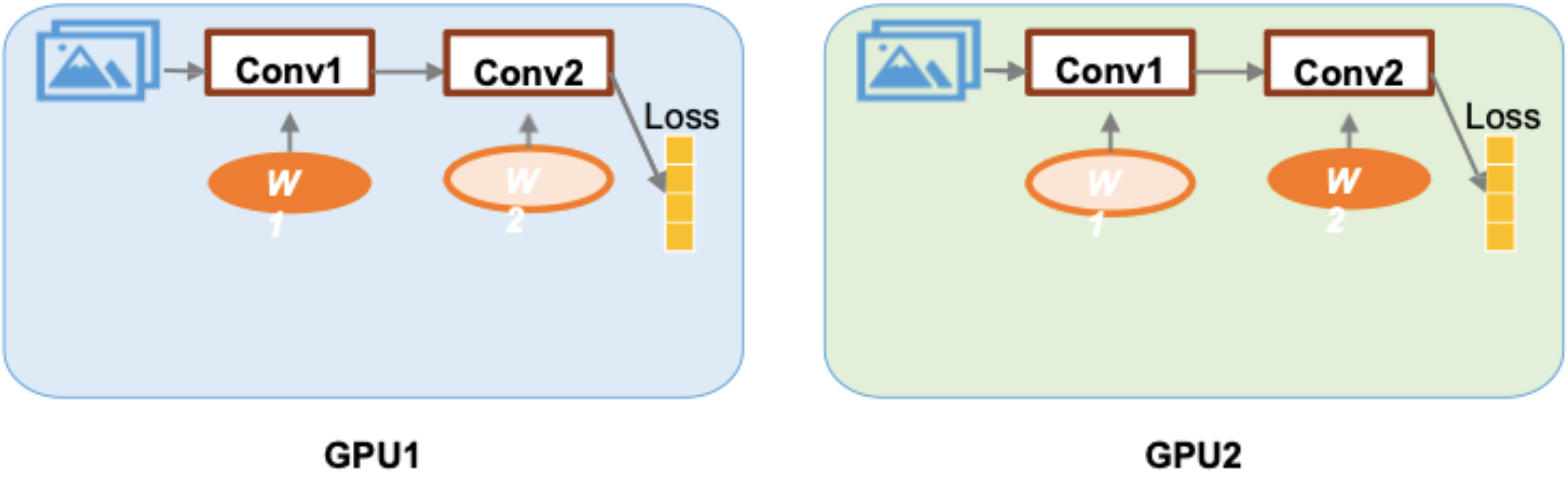
ZeRO 3: Partitioning Parameters

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



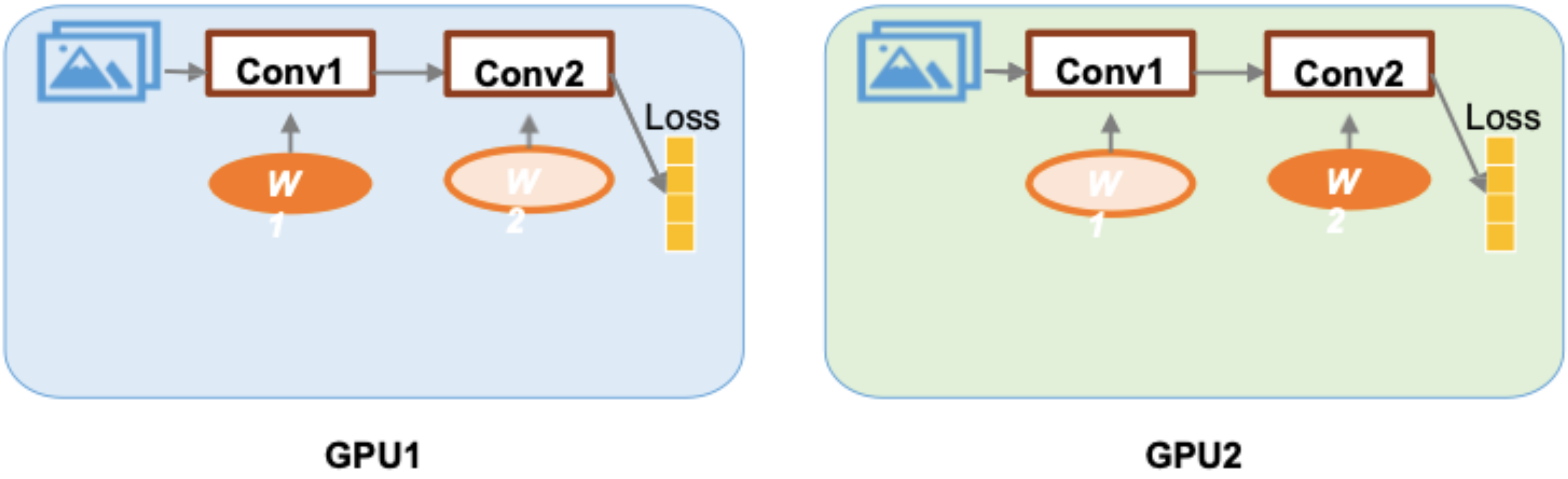
ZeRO 3: Partitioning Parameters

- In ZeRO, model parameters are partitioned across GPUs



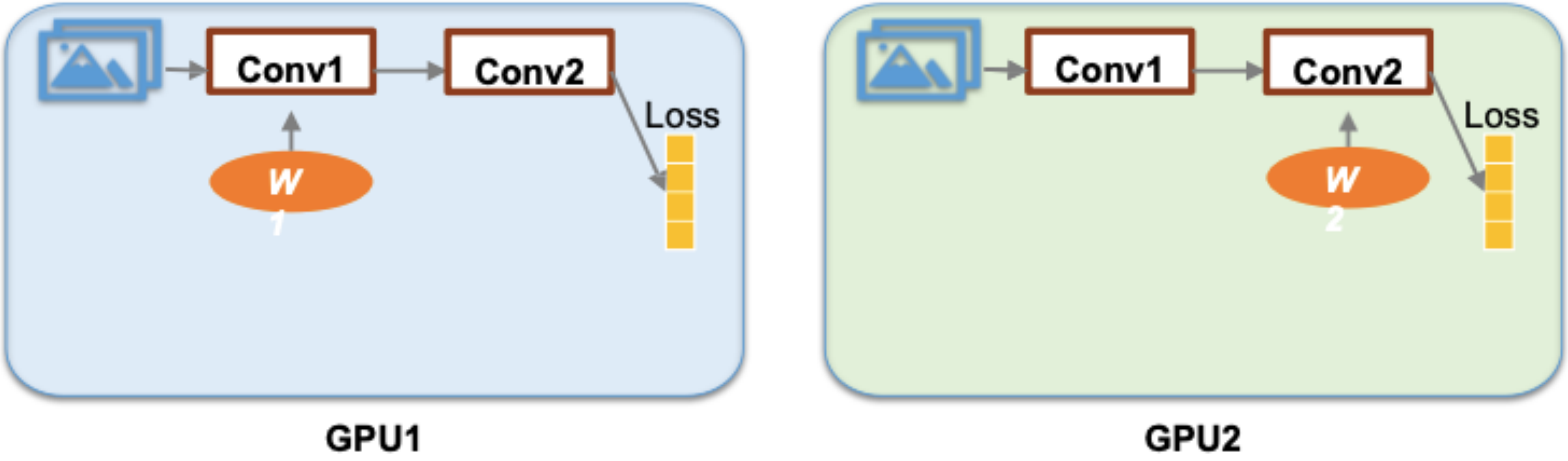
ZeRO 3: Partitioning Parameters

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



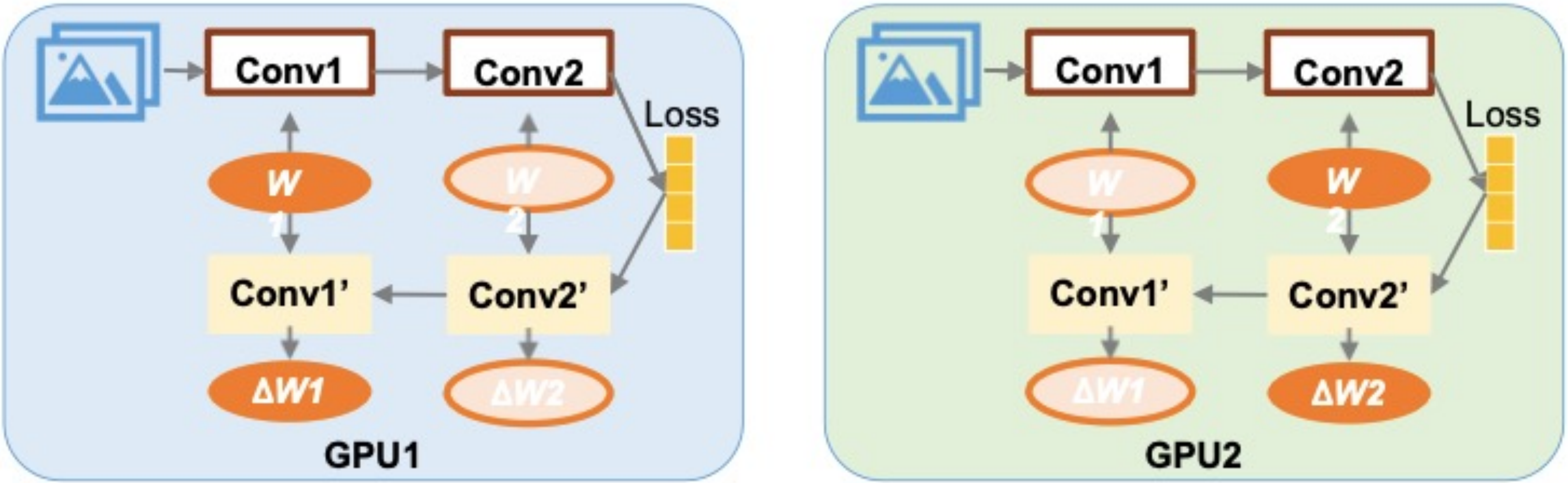
ZeRO 3: Partitioning Parameters

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



ZeRO 3: Partitioning Parameters

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



Zero-DP Summary

- Zero-DP stage 1 and 2 (optimizer state and gradient) doesn't introduce additional communication, while enabling up to 8x memory reduction

- Zero-DP stage 3 (parameter) incurs a maximum of 1.5x communication

ZeRO-R

- Partitioned Activation Checkpointing
  - Model Parallelism by design requires a replication of the activations
  - Split every activation to different devices
  - Gather them when needed

ZeRO-R

- Constant Size Buffers
    - Buffer is used in doing all-reduce to improve bandwidth
    - Modern implementations fuses all the parameters into a single buffer
    - ZeRO uses constant size buffers to be more efficient for a large model

ZeRO-R

- Memory Defragmentation
  - Long-lived memory (Model parameters, Optimizer state): Store together
  - Short-lived memory (Discarded activations)

ZeRO 3: Partitioning Parameters

Communication Analysis

- Model has N parameters
- Operation (scatter-reduce, all-gather) done on the (parameters, gradients) has the same amount of data transfer (C * N = M)
    - Baseline (Vanilla DP): One scatter-reduce to average gradients and one all-gather on averaged gradients, Total 2M
    - Zero-R: No precise numbers, depends on design choice and implementation

Communication Analysis

- Zero-1 (Partition optimizer state) One scatter-reduce to average gradients and one all-gather on collecting parameters, total 2M communication

Communication Analysis

- Zero-2 (Partition gradient): Still one scatter-reduce to average gradients and one all-gather on collecting parameters, total 2M communication overhead

Communication Analysis

- Zero-3 (Partition model parameters): One more all-gather during forward, plus all in Zero-1,2, total 3M communication overhead

Results

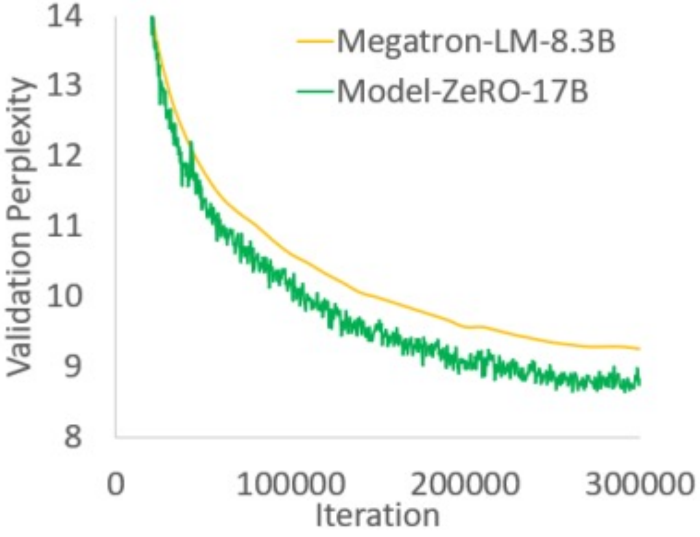
- Theoretical: On a 32GB V100 clusters (Up to 1024 V100)
  - Enable the training of a model with 1 Trillion (1000B) parameters using 1024 V100
  - There is no limit to the number of GPUs. (So probably more)

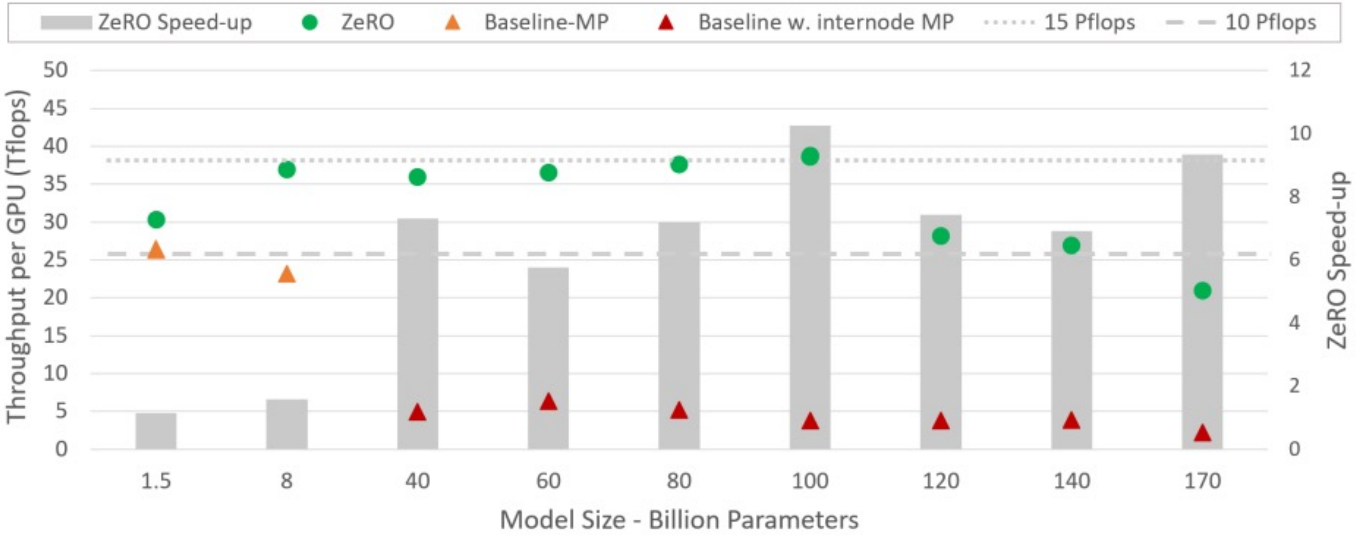
DP	7.5B Model (GB)			128B Model (GB)			1T Model (GB)		
	P_{os}	P_{os+g}	P_{os+g+p}	P_{os}	P_{os+g}	P_{os+g+p}	P_{os}	P_{os+g}	P_{os+g+p}
1	120	120	120	2048	2048	2048	16000	16000	16000
4	52.5	41.3	**30**	896	704	512	7000	5500	4000
16	35.6	**21.6**	7.5	608	368	128	4750	2875	1000
64	**31.4**	16.6	1.88	536	284	**32**	4187	2218	250
256	30.4	15.4	0.47	518	263	8	4046	2054	62.5
1024	30.1	15.1	0.12	513	257	2	4011	2013	**15.6**

Per-device memory consumption of different optimizations

Results

- Practical:
  - Train a 17B model (Turing-NLG. The largest as of 2020.1) and has SOTA perplexity in Webtext-103
  - Train a 100B model on 400 GPUs, achieving high throughput over baseline (~10x, 30% of the theoretical peak)





Summarization

- ZeRO is a distributed learning framework with data parallelization
- ZeRO partitions model states across devices
- ZeRO trains a new SOTA model with 17B models in 2019

Summarization

- Pros
  - Lower memory usages significantly
  - Scalable, flexible, easy-to-use
  - Can be applied to any type of model

Summarization

- Cons
  - Some stages introduce extra communication overhead
  - Performance may depend on infrastructure (PCI-E / NVLink)
  - No reduction on total computation needed

Future Directions

- Improve the memory utilization
- Lower communication overhead
- Better select training configurations with AutoML
- Heterogeneous hardware support to maximize performance