Data Parallelism in LLM Training

Spring 2026

Lecturer: Yuedong (Steven) Xu

Fudan University

ydxu@fudan.edu.cn

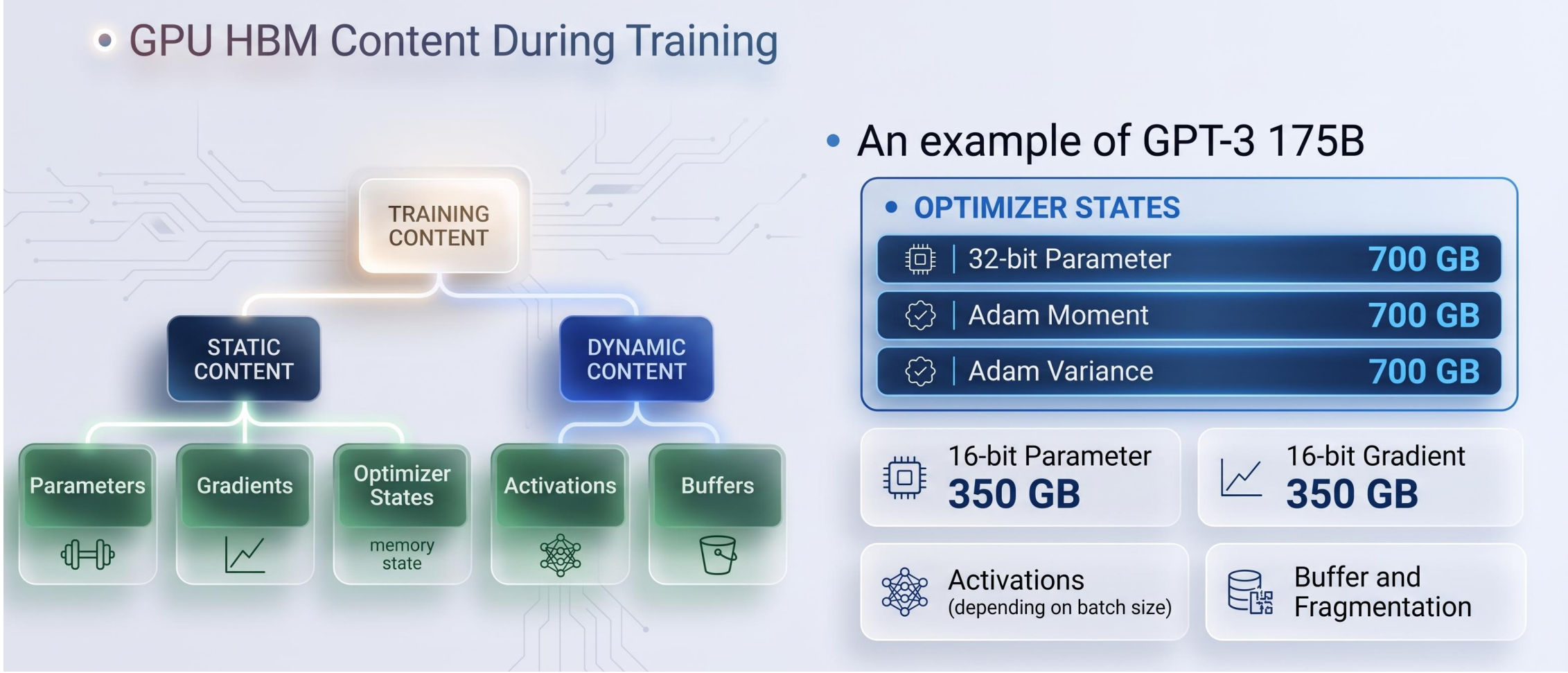
Disclaimer

Machine learning systems is a broad and rapidly evolving field. The course material has been developed using a broad spectrum of resources, including research papers, lecture slides, blogposts, research talks, tutorial videos, and other materials shared by the research community. We sincerely appreciate their efforts and assistance, and try our best to cite the sources of the materials used in this course.

Distributed LLM Training: Outline

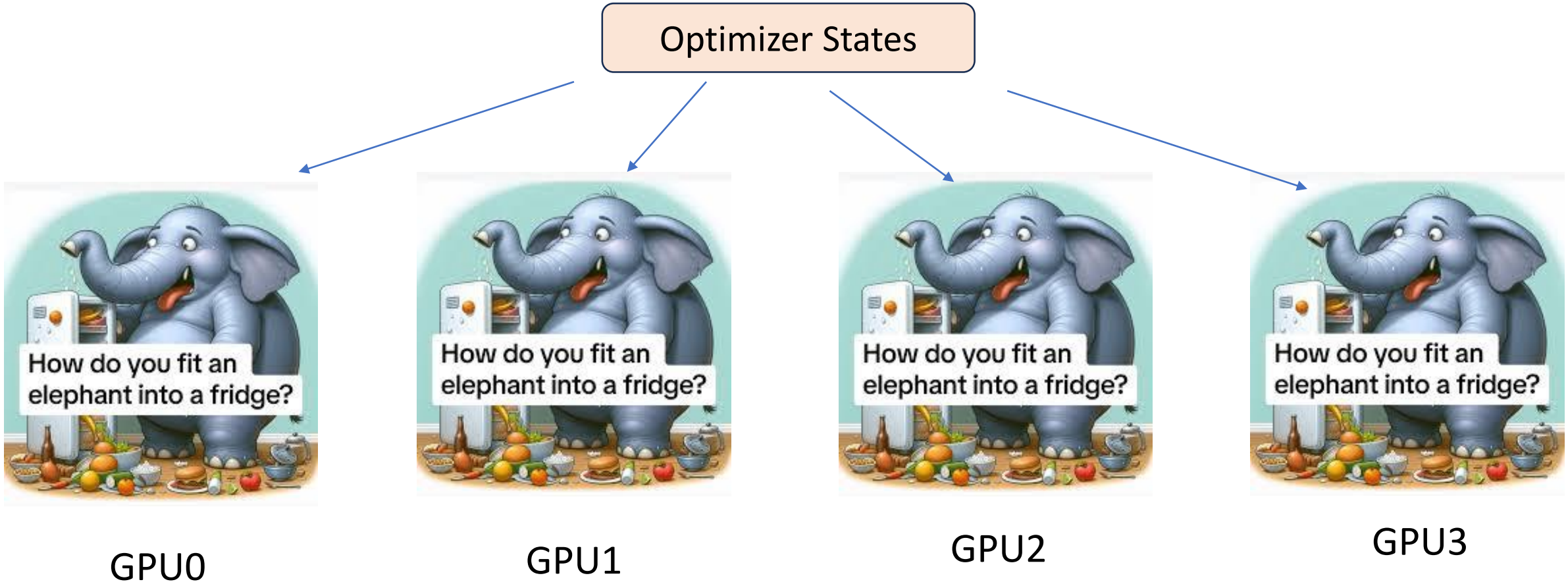
- Data Parallelism
  - Parameter-Server
  - All-Reduce
  - **Memory optimization**

LLM training: DP with Memory Optimization



LLM training: DP with Memory Optimization

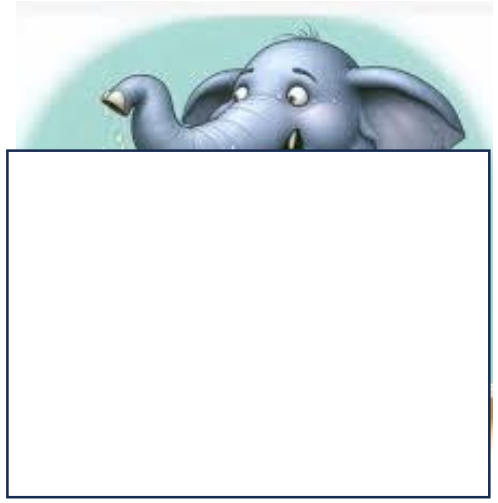
- How to put an elephant into a fridge?



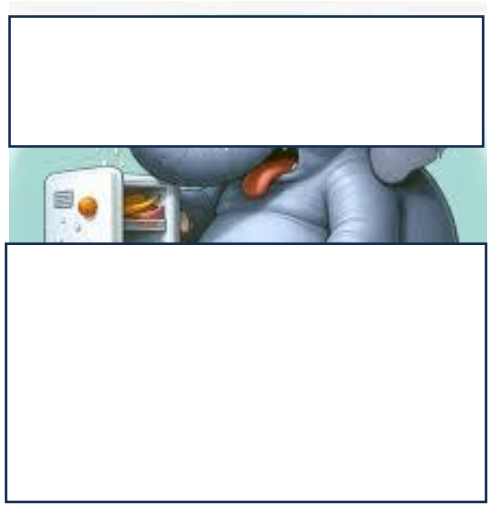
LLM training: DP with Memory Optimization

- How to put an elephant into a fridge?

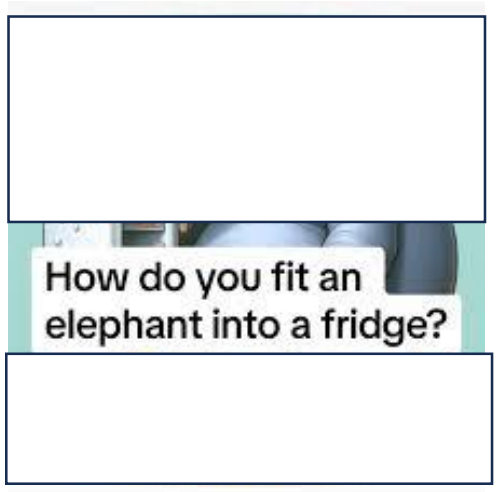
Sharding the elephant into four partitions, and each GPU HBM holds one chunk!



GPU0



GPU1

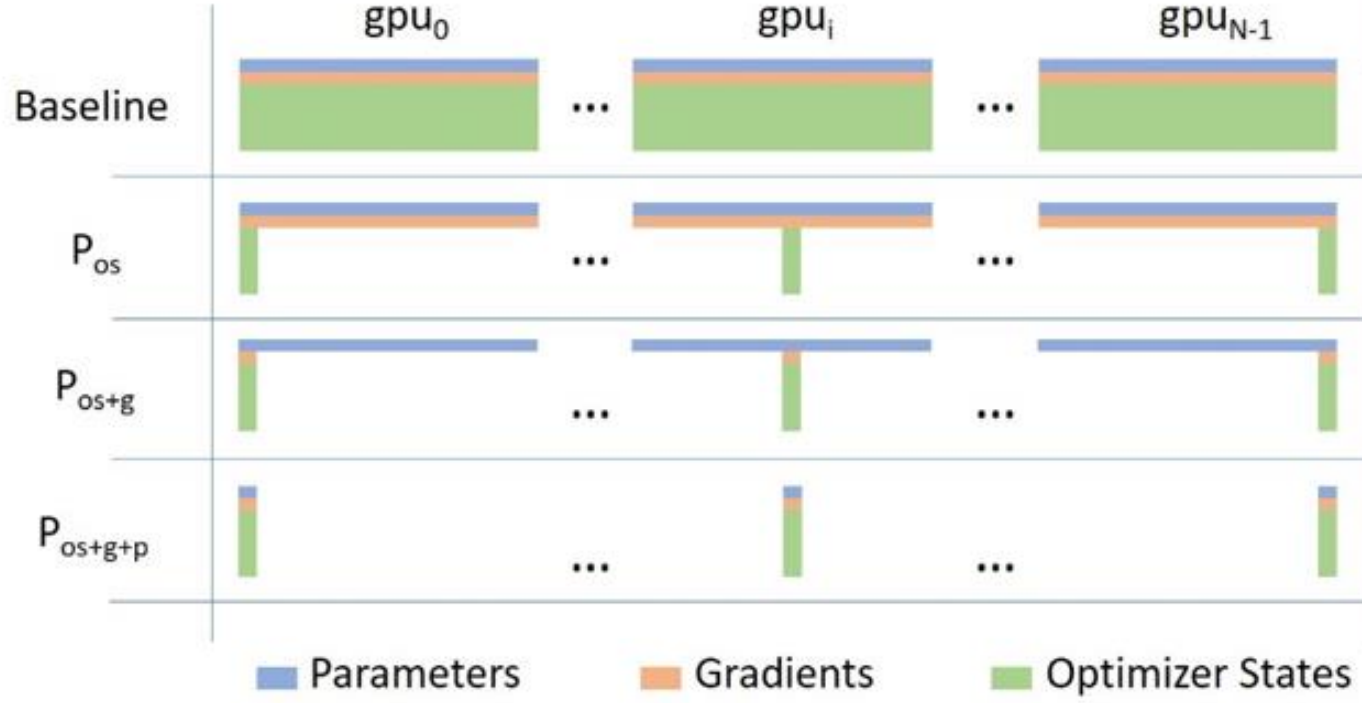


GPU2

GPU3

LLM training: DP with Memory Optimization

- ZeRO (Zero Redundancy) optimization: an overview



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

HBM usage with DP degree 64

《Fit More and Train Faster With ZeRO via DeepSpeed and FairScale》

LLM training: DP with Memory Optimization

- How to compute the storage stage
  - ZeRO-1 (P_{OS})

$$\text{Model States (bytes)}|P_{os+g} = \overbrace{2 \times \Psi}^{\text{Param}} + \overbrace{\frac{14 \times \Psi}{N_d}}^{\text{Gradient + Optimizer}}$$
$$\approx 2\Psi | N_d \to \infty$$

  - ZeRO-2 (P_{OS+g})

$$\text{Model States (bytes)}|P_{os+g} = \overbrace{2 \times \Psi}^{\text{Param}} + \overbrace{\frac{14 \times \Psi}{N_d}}^{\text{Gradient + Optimizer}}$$
$$\approx 2\Psi | N_d \to \infty$$

  - ZeRO-3 (P_{OS+g+p})

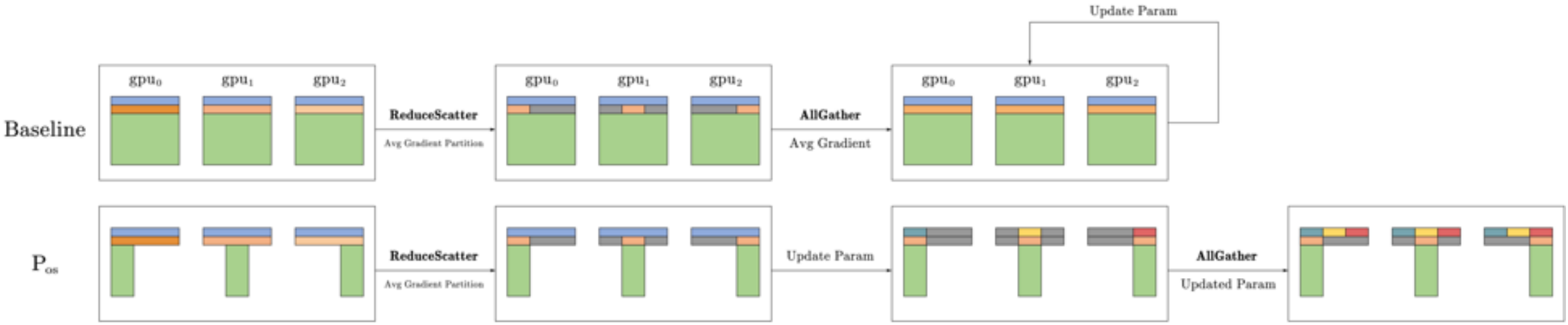
$$\text{Model States (bytes)}|P_{os+g+p} = \overbrace{\frac{16 \times \Psi}{N_d}}^{\text{Param + Gradient + Optimizer}}$$
$$\approx 0 | N_d \to \infty$$

LLM training: DP with Memory Optimization

- ZeRO-1&2
  - *Forward* is OK because each GPU holds the complete *parameter* ✓
  - *Backward* is OK because of the same reason ✓
  - *AllReduce* = ReduceScatter + AllGather
    - *ReduceScatter is OK* ✓
    - *Update optimizer state shards* ✓
      - *Incomplete optimizer states at every GPU*
    - *Update parameter shards* ✓
    - *AllGather remaining parameter shards to assembly the complete parameter* ✓

LLM training: DP with Memory Optimization

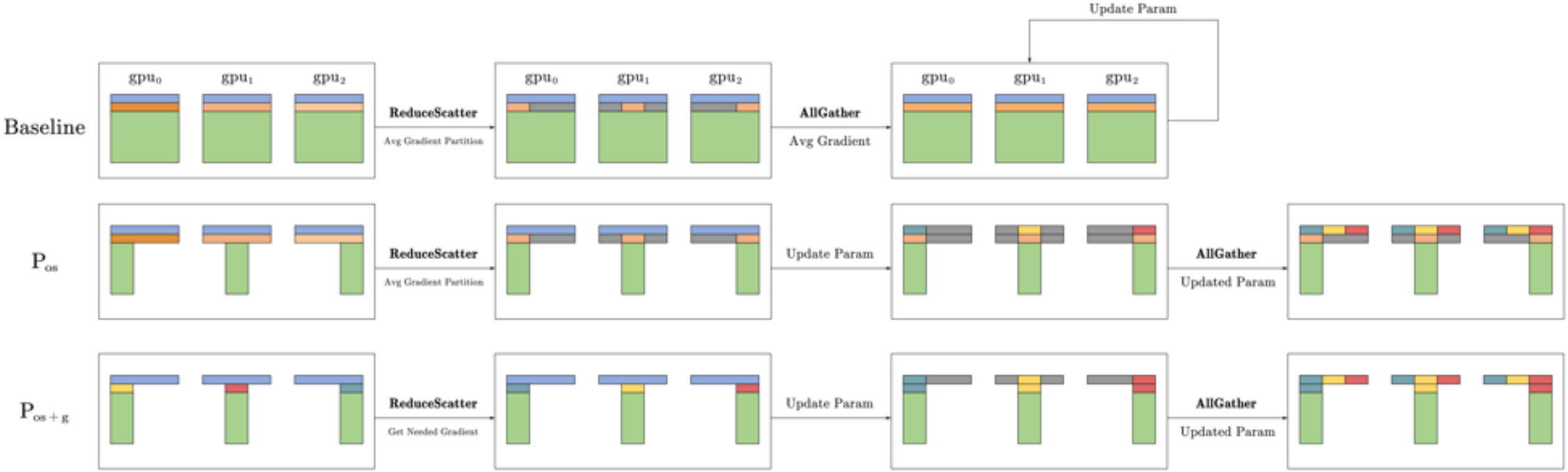
- ZeRO-1



- Using scatter-reduce and all-gather to obtain *global gradient shard*, and using global gradient shard to update the parameter and optimizer shard
- Using all-gather to obtain the complete global parameter

LLM training: DP with Memory Optimization

- ZeRO-2



- Similar to ZeRO-1, except not conserving all the global gradient shards

LLM training: DP with Memory Optimization

- ZeRO-3 (parameter, gradient and optimizer sharding)
  - *Forward* is ***NOT*** OK because of incomplete *parameter*
    - *Need to fetch parameter shards from other GPUs via BROADCAST*
  - *Backward* is ***NOT*** OK because of incomplete parameter
    - *Need to fetch parameter shards from other GPUs via BROADCAST again*
  - *Aggregating local gradient shards to obtain global gradient shards*
    - *Reduce Scatter is OK*
  - *Updating optimizer shards and parameter shards are OK*
    - *No "AllGather" operation afterwards*

LLM training: DP with Memory Optimization

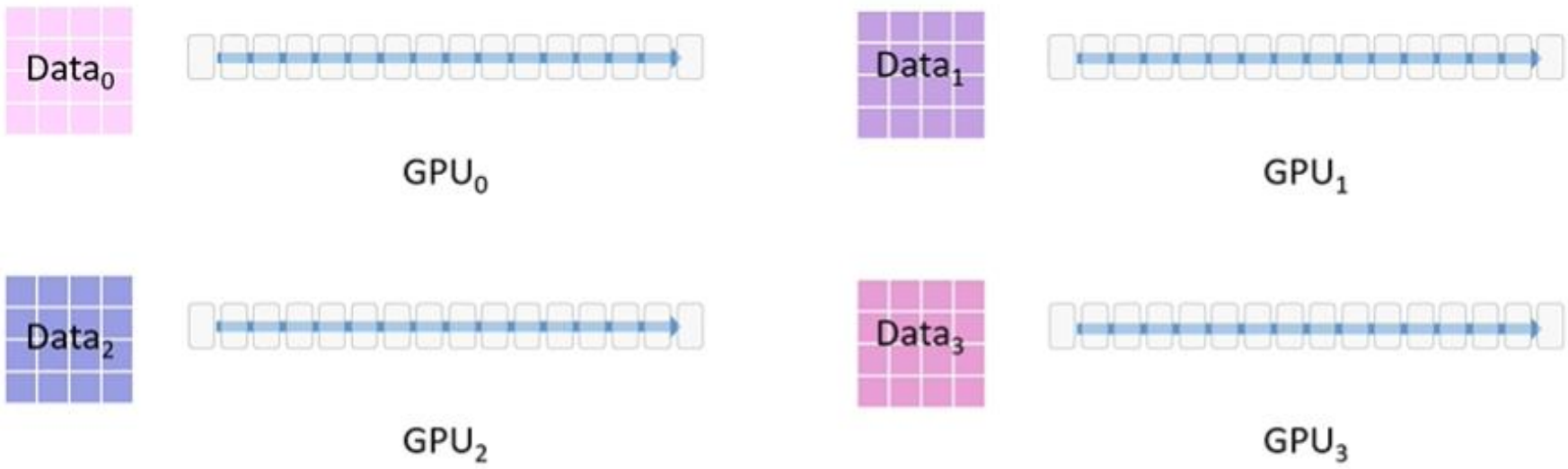
- ZeRO-3: An animation (Implementation on DeepSpeed could be somewhat different)

- All-Gather
  - Collecting parameters for forward propagation
- All-Gather
  - Collecting parameters for back propagation
- Reduce-Scatter
  - Obtaining global gradient
- Update OS and param.

Source: microsoft official site

LLM training: DP with Memory Optimization

- ZeRO-3: An animation



Dataset is partitioned into four pieces, each of which stored at a GPU

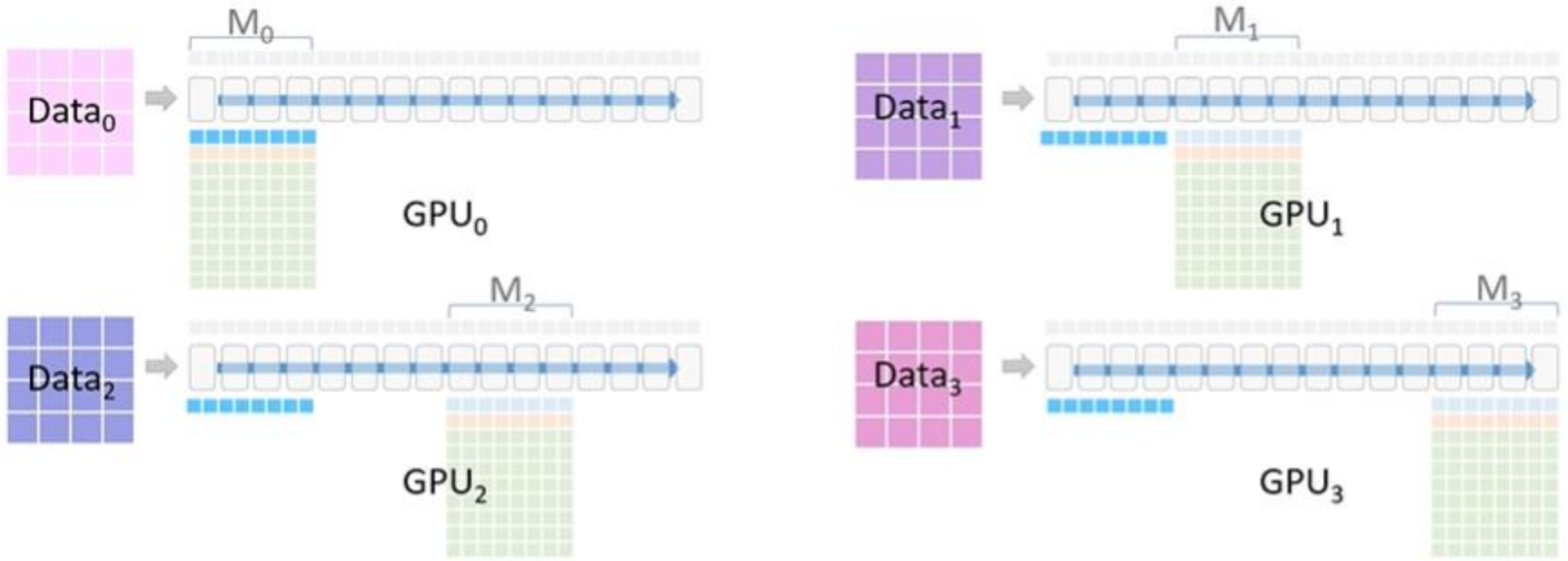
LLM training: DP with Memory Optimization

- ZeRO-3: An animation

Each GPU is responsible for one piece of the final model

LLM training: DP with Memory Optimization

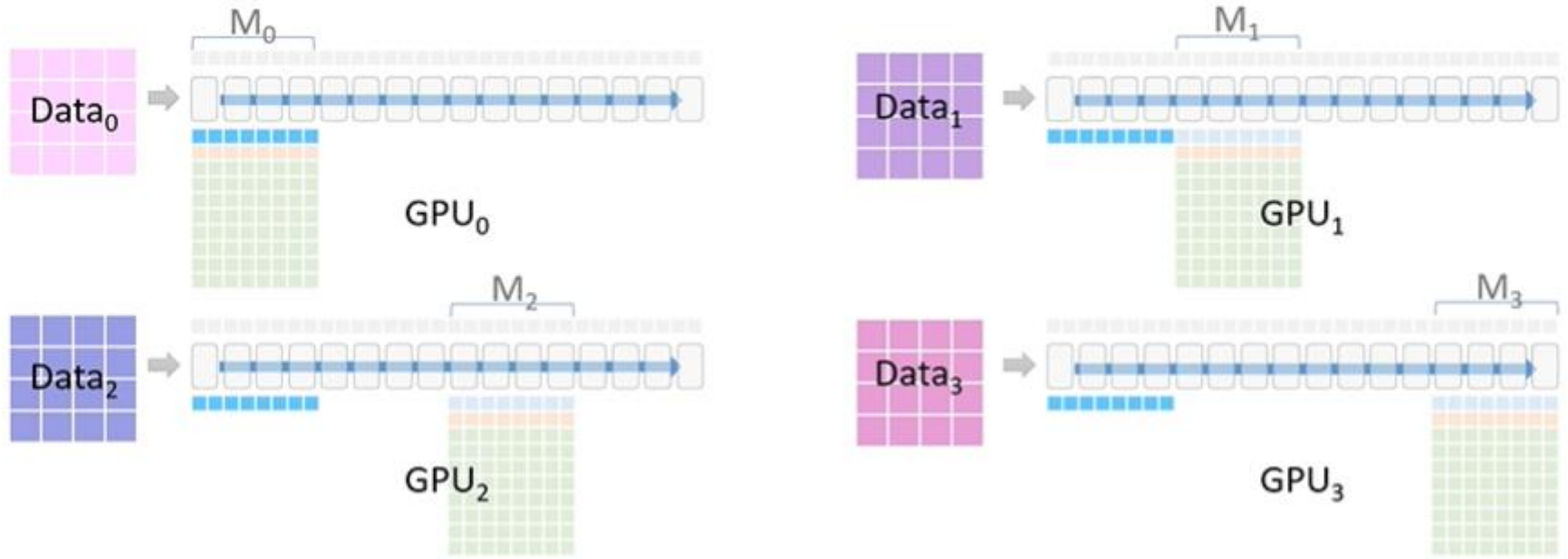
- ZeRO-3: An animation



Only GPU0 initially has the model parameters for M0, so it broadcasts them to GPU-1-2-3

LLM training: DP with Memory Optimization

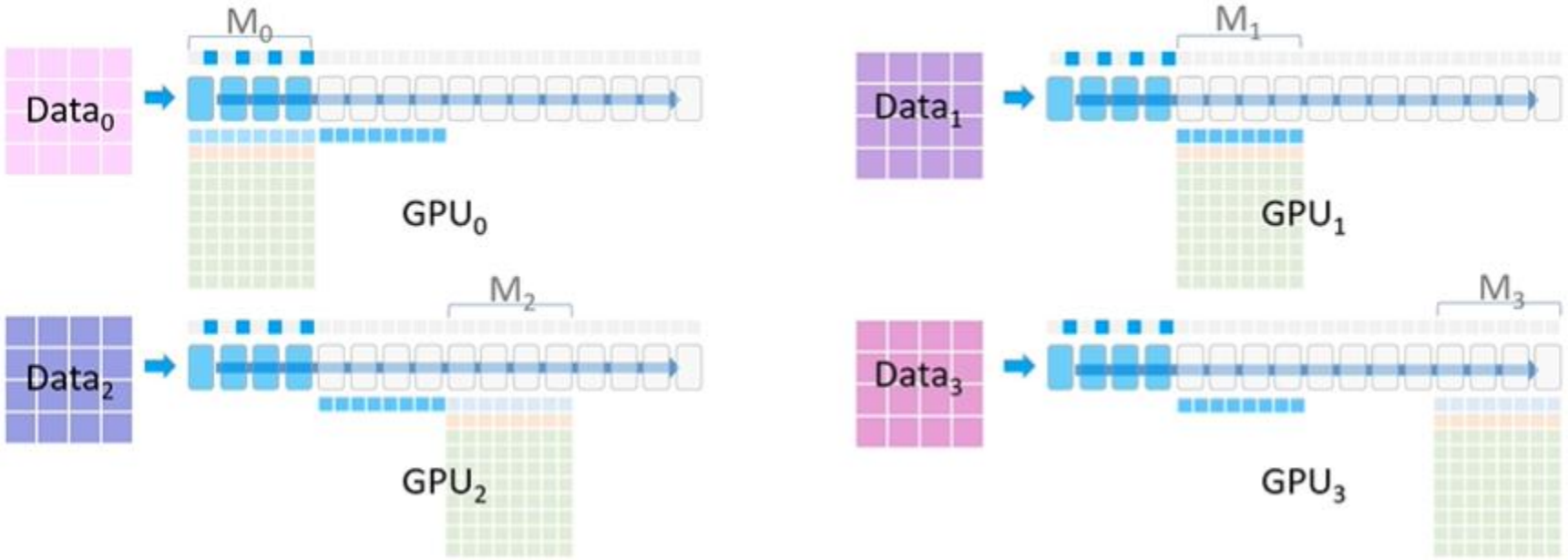
- ZeRO-3: An animation



GPU-1-2-3 store them in the temporary buffer and begin the forward propagation

LLM training: DP with Memory Optimization

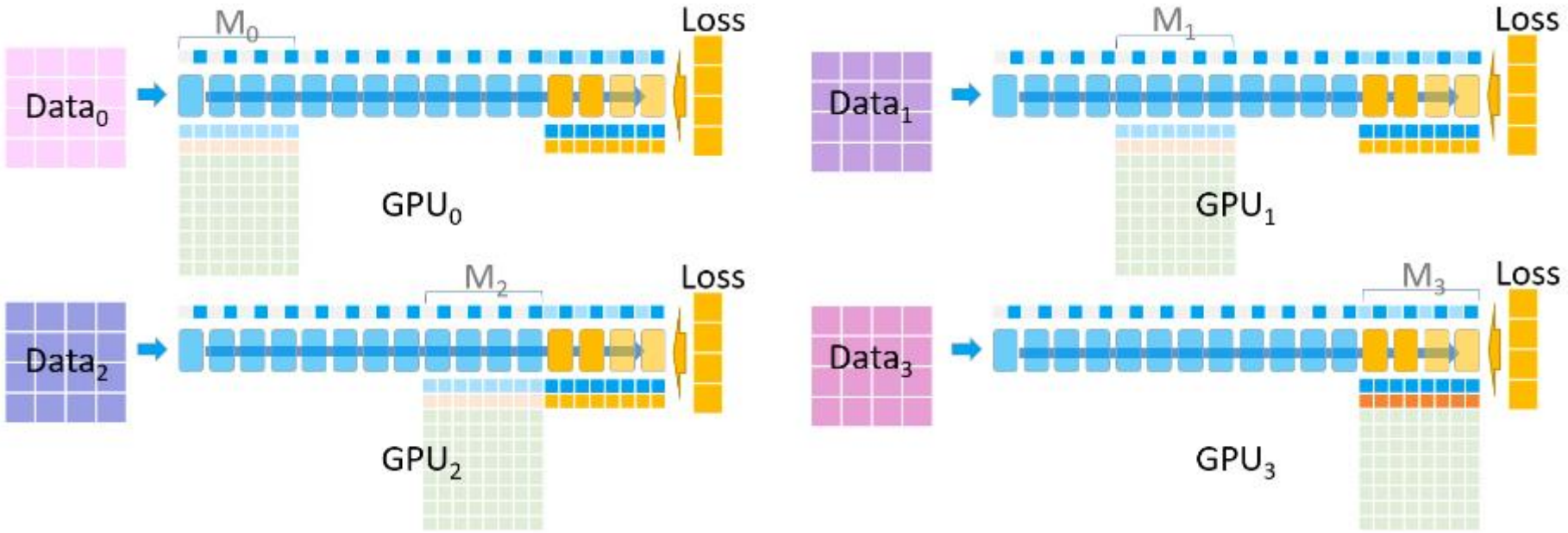
- ZeRO-3: An animation



GPU1 does the same thing, e.g. broadcasting M1 to GPU-0-2-3

LLM training: DP with Memory Optimization

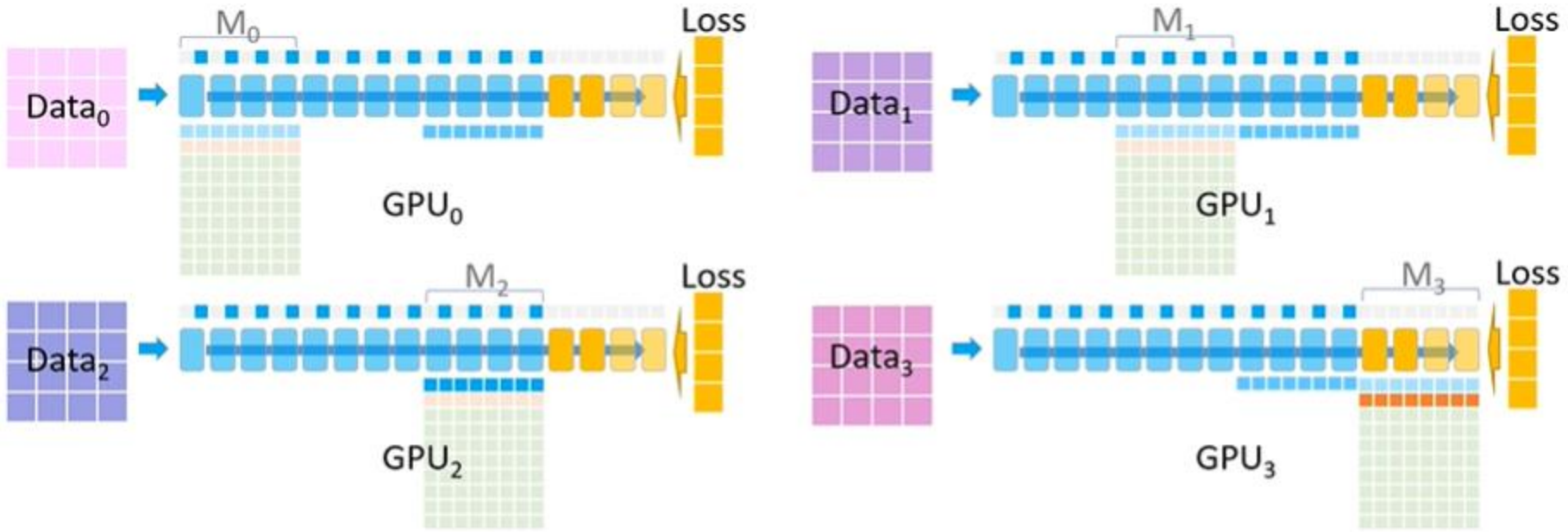
- ZeRO-3: An animation



Backward pass: GPU-0-1-2 pass their M3 gradient to GPU3, and GPU3 aggregate the gradients to obtain the global gradient shard so to generate the final M3 for all data

LLM training: DP with Memory Optimization

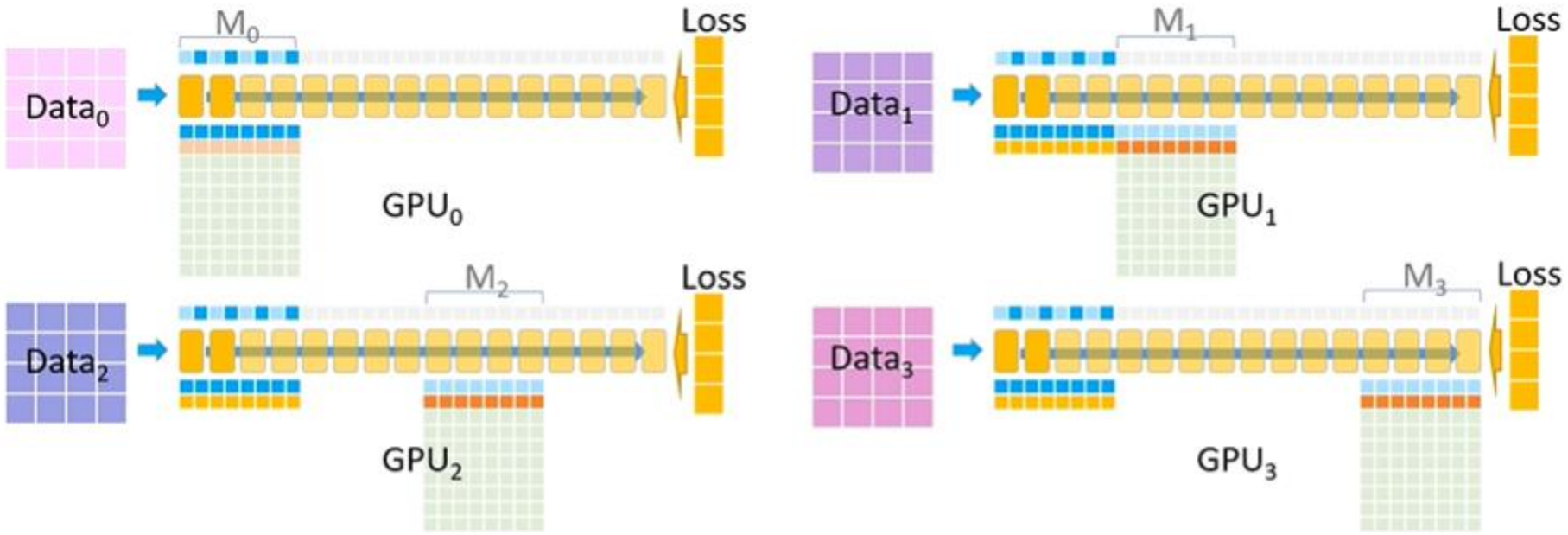
- ZeRO-3: An animation



Backward pass: GPU2 pass its M2 parameter to GPU-0-1-3, so that they can proceed backpropagation. GPU2 aggregates the gradient shards from GPU-0-1-3 and obtain the global gradient shard for M2.

LLM training: DP with Memory Optimization

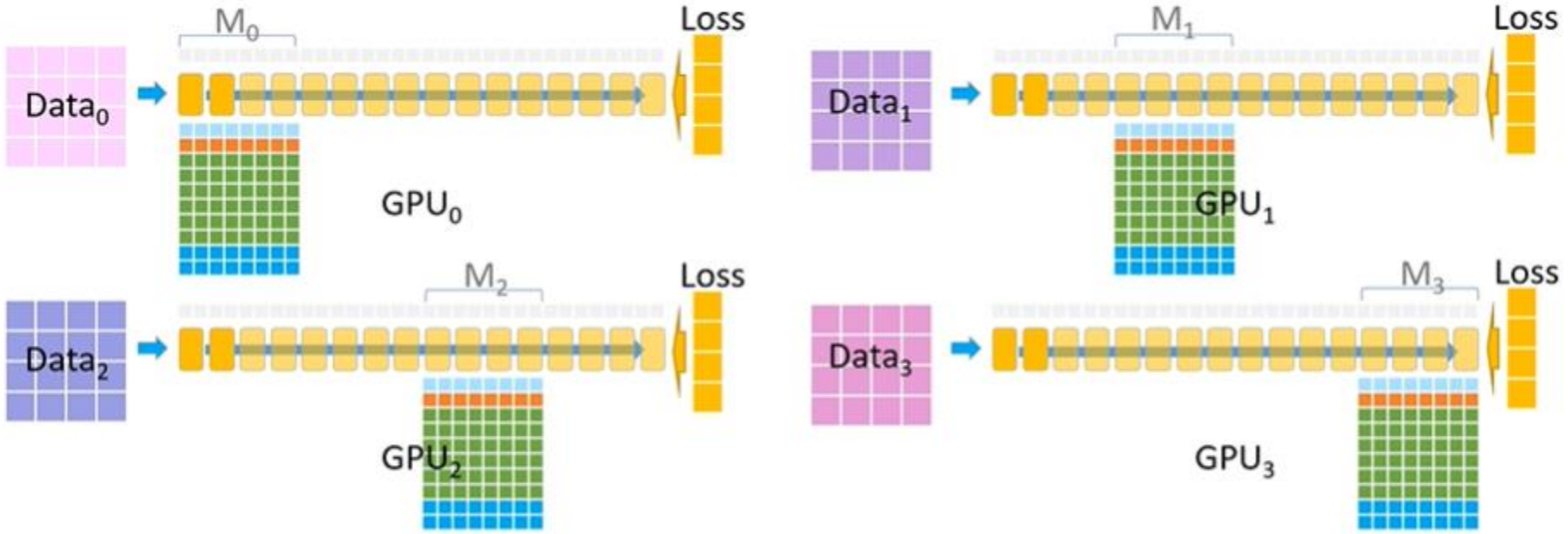
- ZeRO-3: An animation



Backward pass: The above steps repeat until GPU0 pass its M0 parameter to GPU-1-2-3 so that they can proceed the backpropagation. GPU0 aggregated the gradient shard for M0 parameter.

LLM training: DP with Memory Optimization

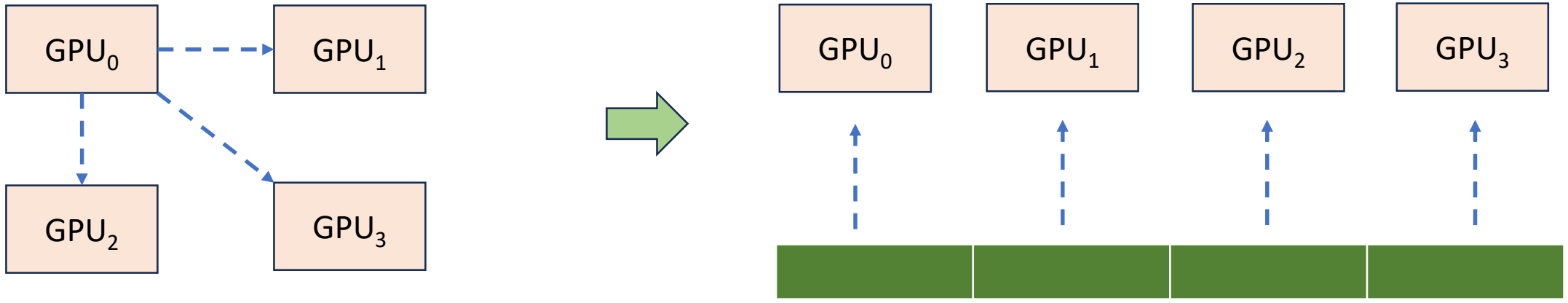
- ZeRO-3: An animation



Update phase: all the GPUs update their optimizer states and parameters in parallel.

LLM training: DP with Memory Optimization

- ZeRO-3: DeepSpeed Implementation
  - Replacing *broadcast* by *allrgather (why?)*
  - Intra-layer partitioning instead of inter-layer partitioning

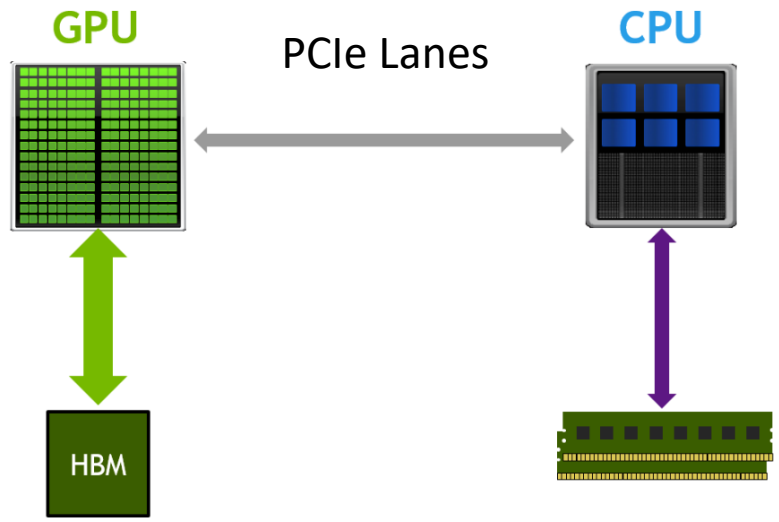


Broadcast

Intra-layer partitioning, and AllGather

LLM training: DP with Memory Optimization

- ZeRO Offload
  - When GPU HBM size is still insufficient to store partitions of optimizer states, parameter and gradients, can we still use it for training?

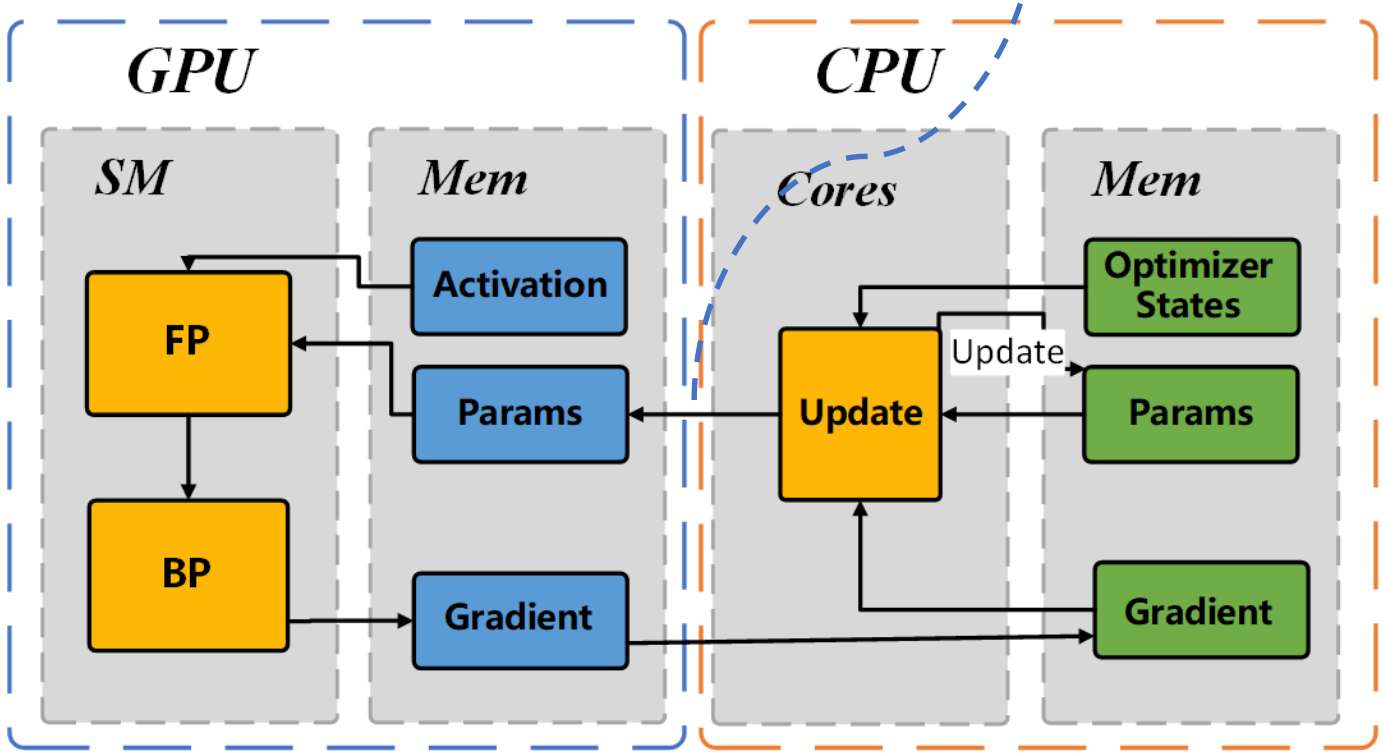


- CPU memory
  - ~1TB vs 80GB HBM
- Connection speed
  - PCIe 5.0 (16 lanes) ~ 64GB/s
  - NVIDIA H100 NVLink 4.0 ~ unidirectional 450GB/s
- GPU←→CPU bottleneck

LLM training: DP with Memory Optimization

- ZeRO Offload

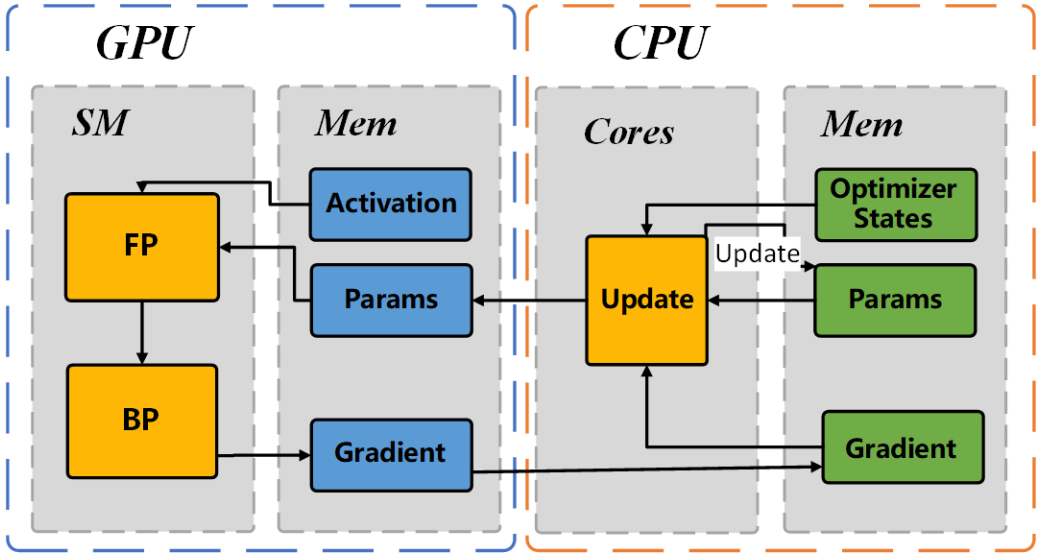
PCIe communications back and forth



- GPU HBM
  - Parameter
  - Gradient
  - Activations
- CPU Memory
  - Optimizer States
  - Parameter
  - Gradient

LLM training: DP with Memory Optimization

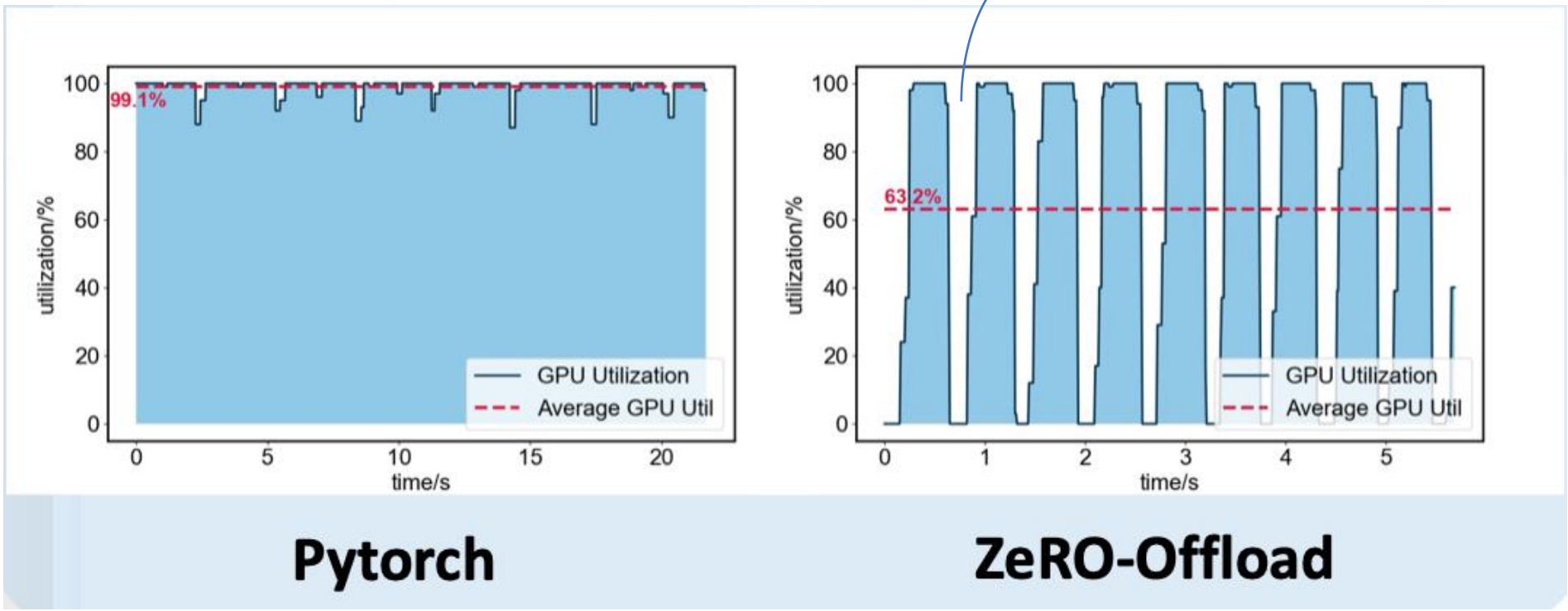
- ZeRO Offload: workflow
  - Forward propagation at **GPU HBM**
  - Backward propagation at **GPU HBM**
  - Optimizer state is at CPU main memory
  - Transmit gradient to from GPU to CPU
  - Update optimizer states at CPU memory (which is relatively slow)
  - Update parameter at CPU memory
  - Transmit new parameter from CPU to GPU, and repeat



LLM training: DP with Memory Optimization

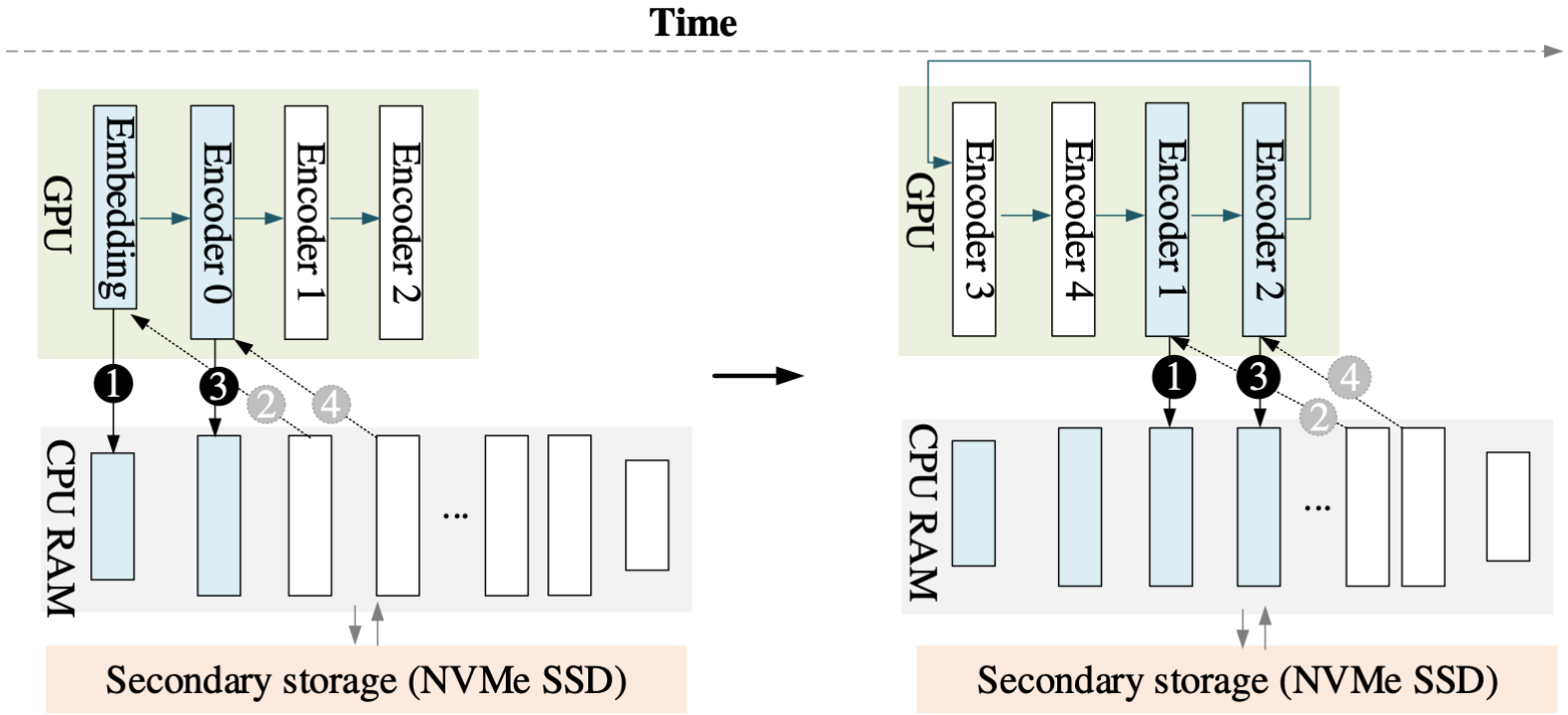
- Pros and Cons
  - Training GPT2 on A100: reduced GPU utilization

Where are these bubbles coming from?



LLM training: DP with Memory Optimization

- ZeRO Offload Variants

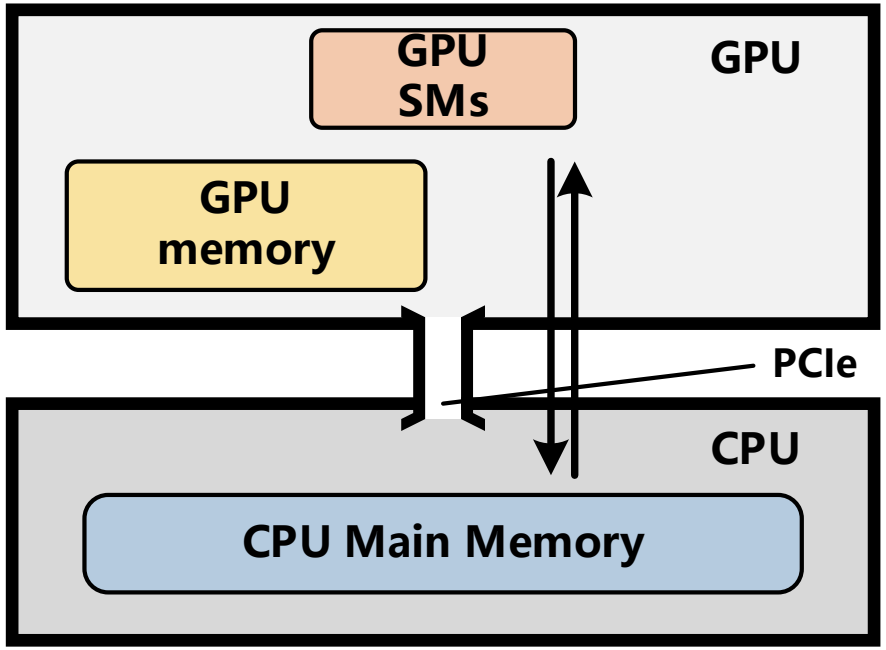


STRONGHOLD stores some DNN layers in the GPU memory and swapping out the finished layer states to the CPU RAM.

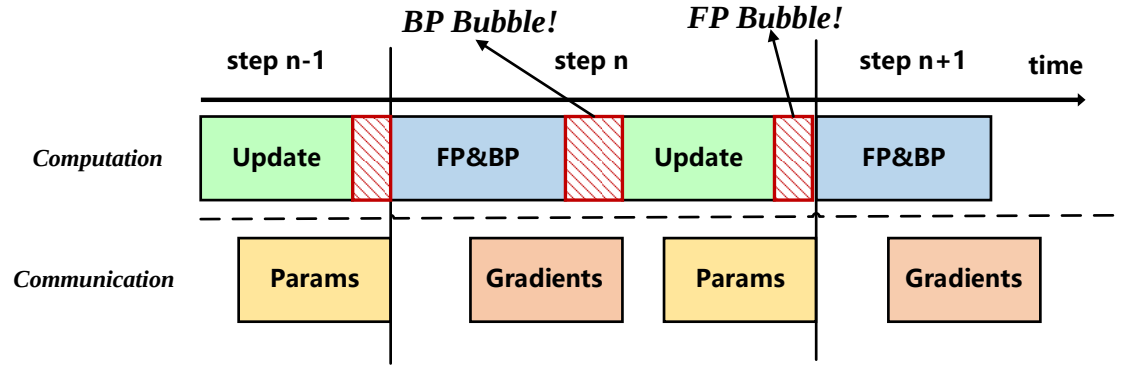
《STRONGHOLD: Fast and Affordable Billion-Scale Deep Learning Model Training》

LLM training: DP with Memory Optimization

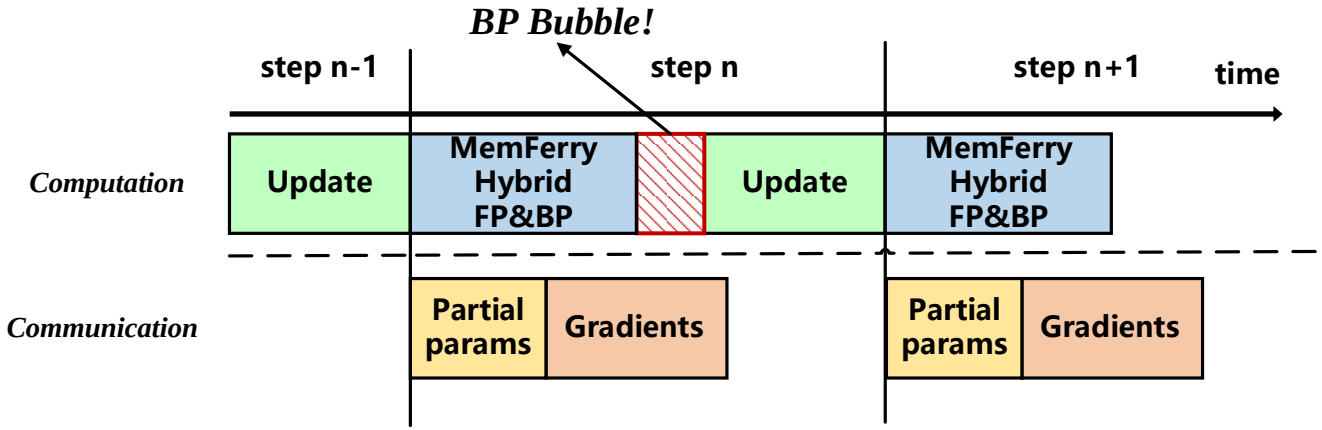
- ZeRO Offload Variants



DHA mode: GPU SMs compute tensors stored in CPU memory



Eliminating bubbles

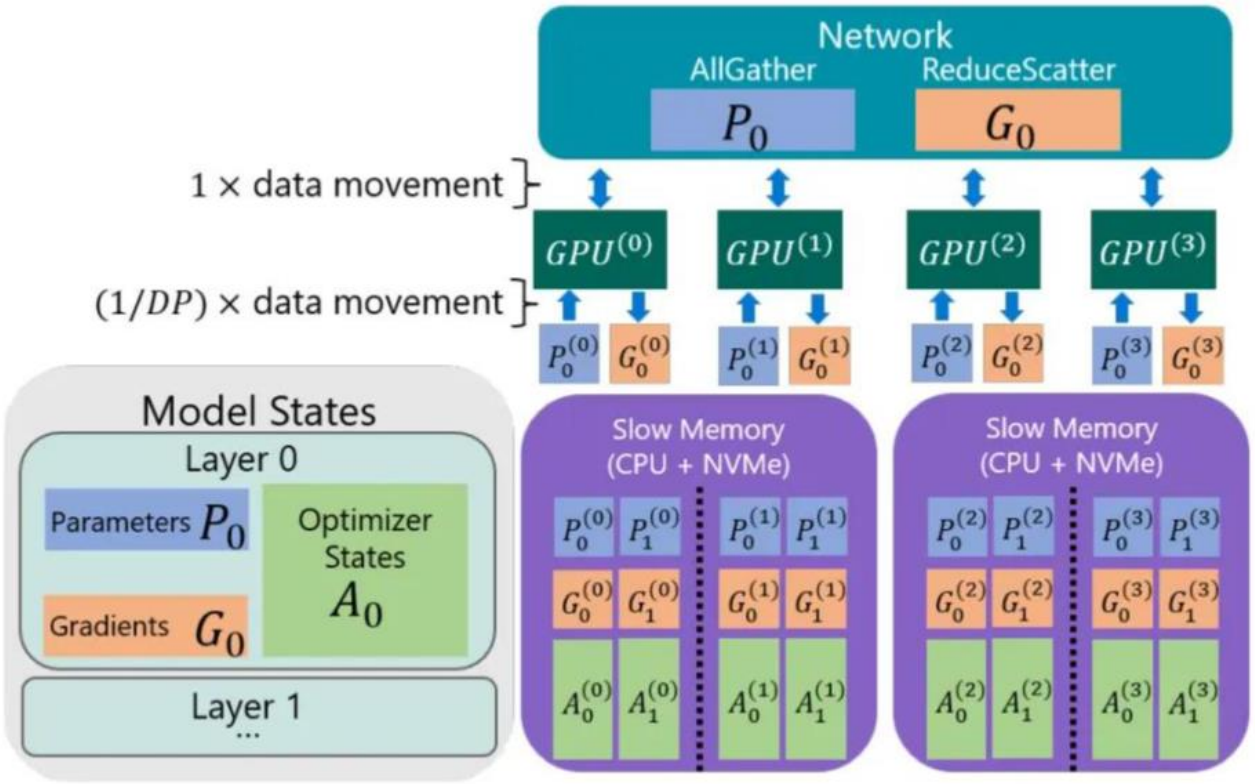


Eliminating bubbles

《MemFerry: A Fast and Memory Efficient Offload Training Framework with Hybrid GPU Computation》

LLM training: DP with Memory Optimization

- ZeRO Infinity (for your reference)

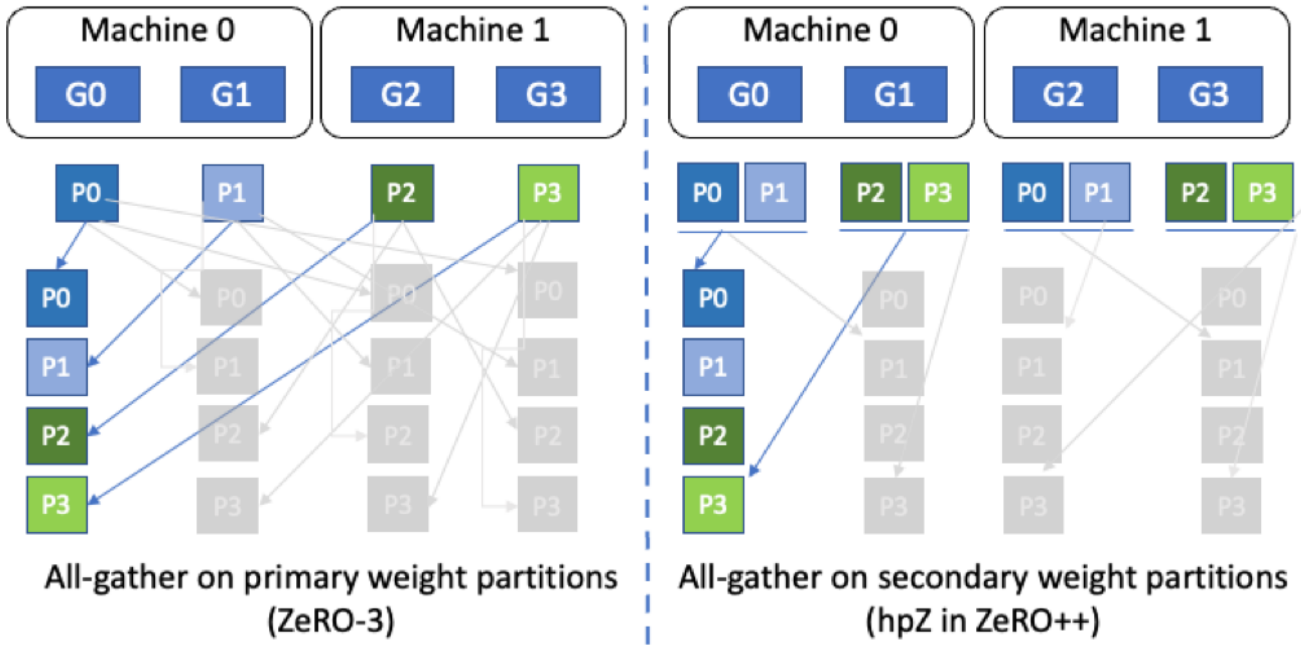


- Bandwidth
  - GPU > CPU > NVMe
- Read Data
  - Read parameter using All-Gather
- Data Path
  - CPU → GPU
  - NVMe → CPU → GPU

《ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning》

LLM training: DP with Memory Optimization

- ZeRO++: Low-bandwidth Scenario (for your reference)



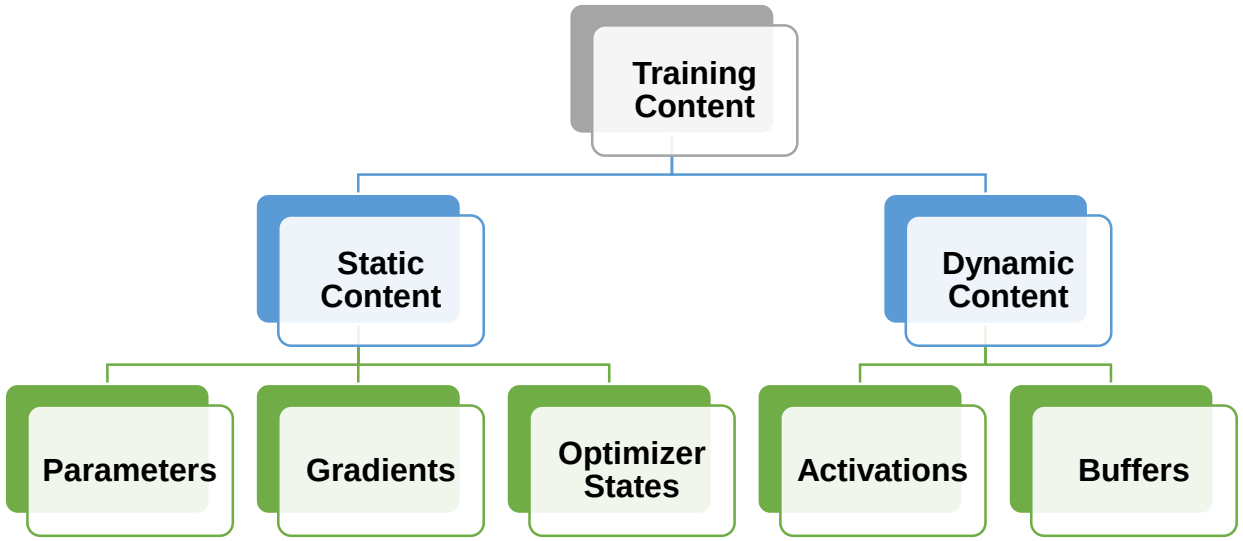
Hierarchical partitioning (not diving into details)

- Quantized Weight Communication
  - FP16 weight -> INT8 weight before All-Gather
  - block-quantization based all-gather in FP
  - INT8 -> FP16 after All-Gather
- Hierarchical Partitioning
  - (trying to) eliminate the inter-node all-gather
- Quantized Gradients Communication
  - Standard Reduce-Scatter involves a lot of quantization and dequantization steps
  - leverages all-to-all collectives to implement quantized reduce- scatter

《ZeRO++: Extremely Efficient Collective Communication for Giant Model Training 》

LLM training: DP with Memory Optimization

- Retrospect: GPU HBM Content During Training



- An example of GPT-3 175B
  - Optimizer States
    - 32-bit Parameter (700 GB)
    - Adam Moment (700 GB)
    - Adam Variance (700 GB)
  - 16-bit Parameter (350 GB)
  - 16-bit Gradient (350 GB)
  - **Activations (depending on batch size)**
  - Buffer and Fragmentation

LLM training: DP with Memory Optimization

- Activation is non-negligible
  - Forward propagation
  - Backward propagation
  - Size of activation
    - Standard transformer: b - batch size, s - sequence length, h - hidden layer dimension , a - head number

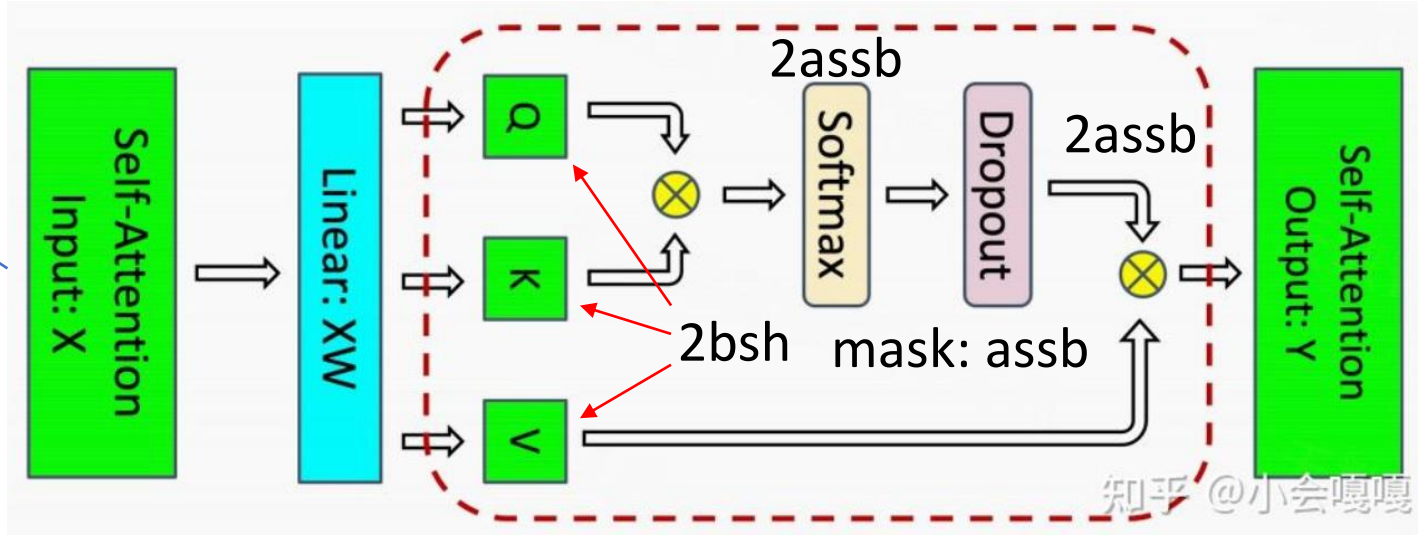
Output of a previous layer, intermediate variable

$$y = Wx + b$$

$$\frac{dL}{dW} = \frac{dL}{dy}x$$

Cannot be discarded after FP

bsh * 2 Bytes



In-total: 5bss + 8sbh

LLM training: DP with Memory Optimization

- Activation is non-negligible
  - Forward propagation
  - Backward propagation
  - Size of activation
    - Standard transformer: b - batch size, s - sequence length, h - hidden layer dimension, a - head number

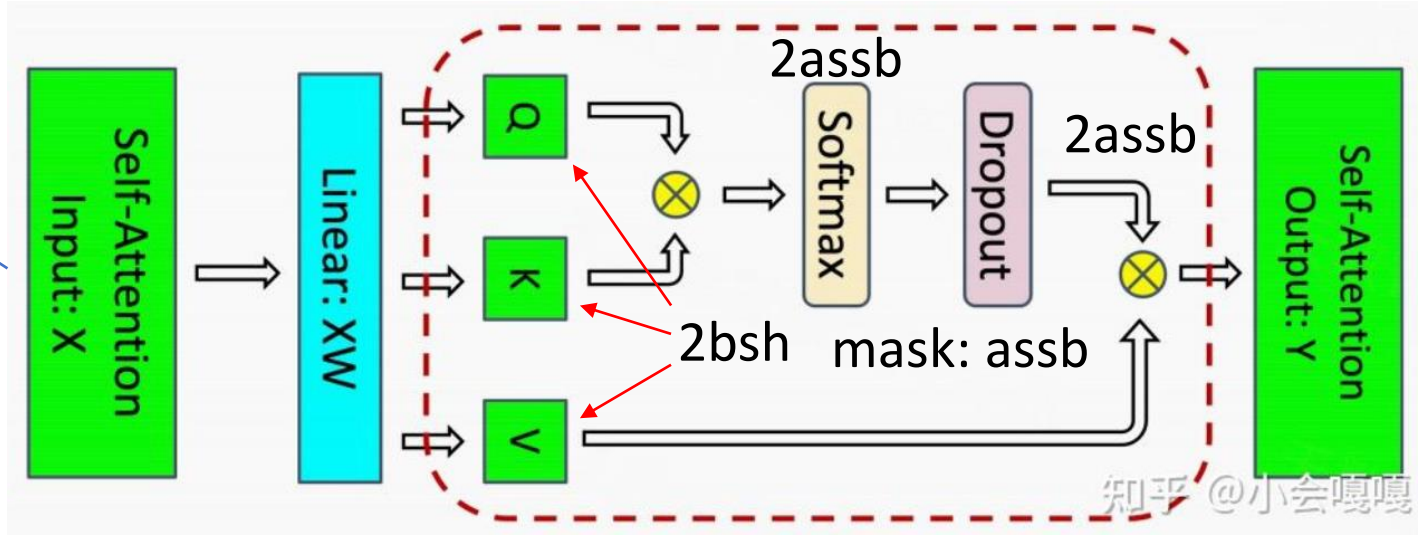
Output of a previous layer, intermediate variable

$$y = Wx + b$$

$$\frac{dL}{dW} = \frac{dL}{dy}x$$

Cannot be discarded after FP

bsh * 2 Bytes

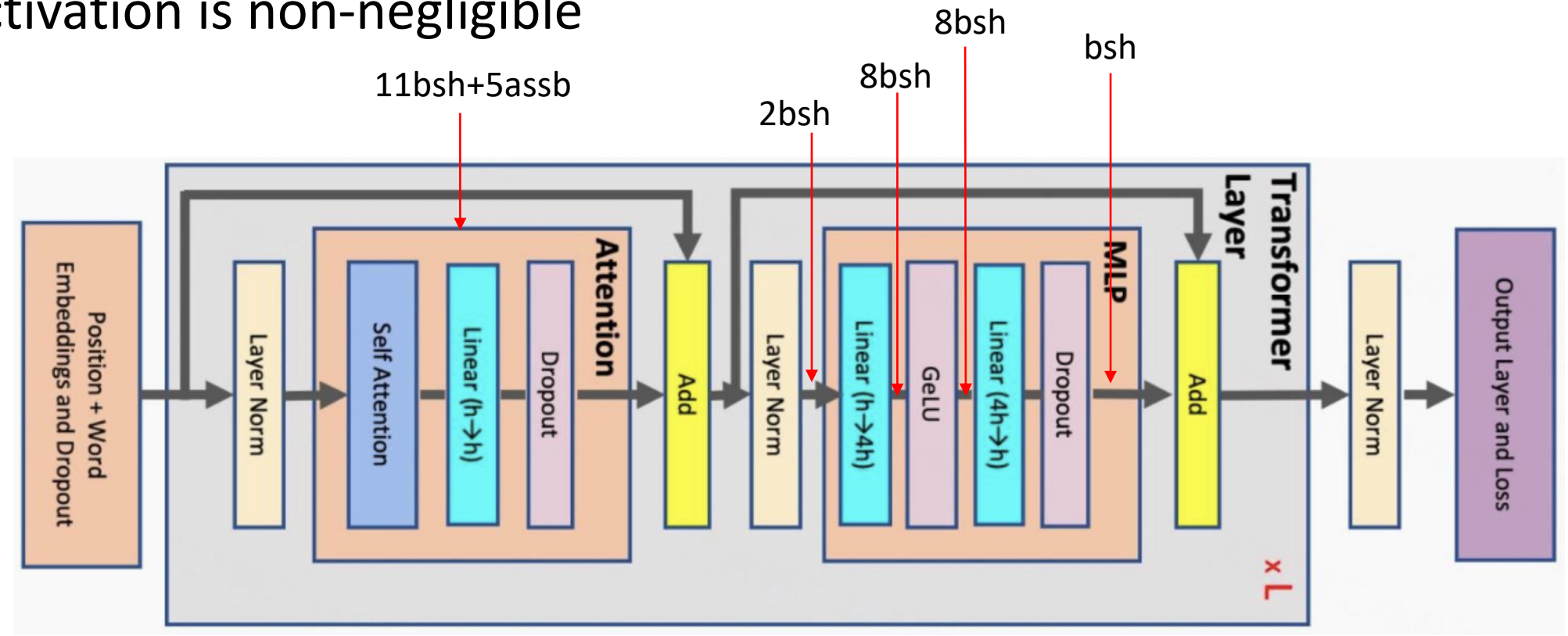


3sbh: Linear layer + dropout layer

In-total: 5abss + 11sbh

LLM training: DP with Memory Optimization

- Activation is non-negligible



In-total: 5abss + 34sbh

From Zhihu blogger and《Reducing Activation Recomputation in LLMs》

LLM training: DP with Memory Optimization

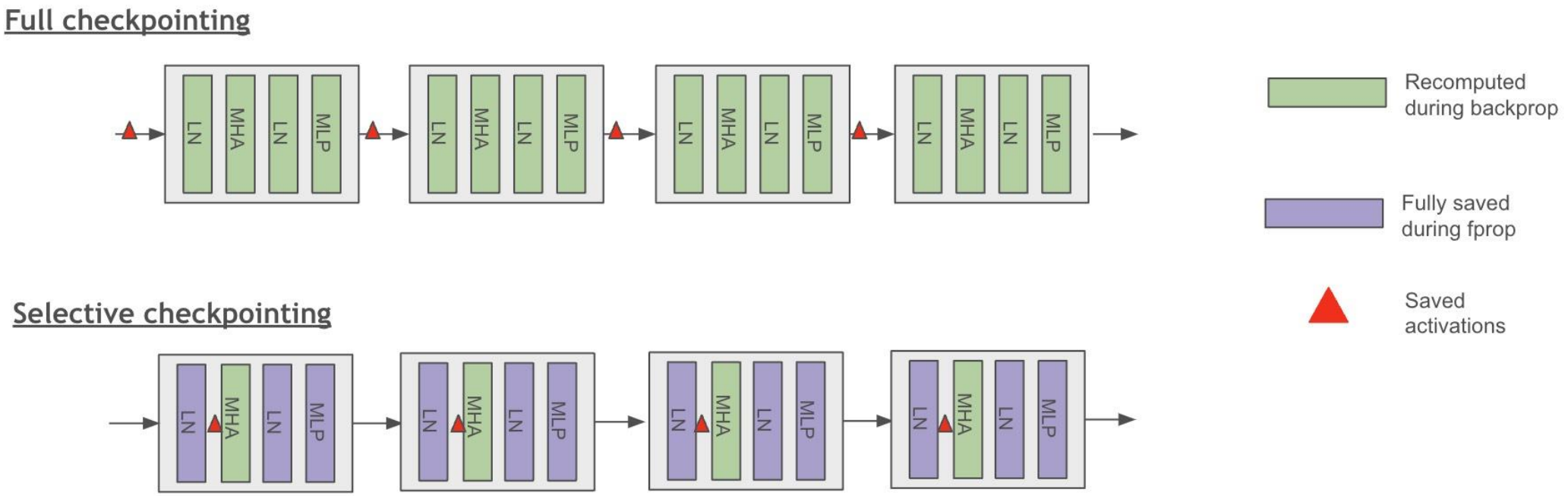
- Activation is non-negligible
  - b: 3.2M tokens/sequence length, s: 2048 tokens, h: 12288, l: 96 layers
  - Activation size ?
  - Full activation re-computation: only keeping the initial input and recompute everything
    - Minimal memory occupation
    - Prolonged training time (doing forward propagation once again) by 30%~40%
  - Goal of strategic recomputation (not the scope of this class)
    - Significantly reducing memory occupation while slightly increasing training time
    - Softmax and Softmax dropout are more suitable to be re-computed

https://lambda.ai/blog/demystifying-gpt-3

《Reducing Activation Recomputation in LLMs》

LLM training: DP with Memory Optimization

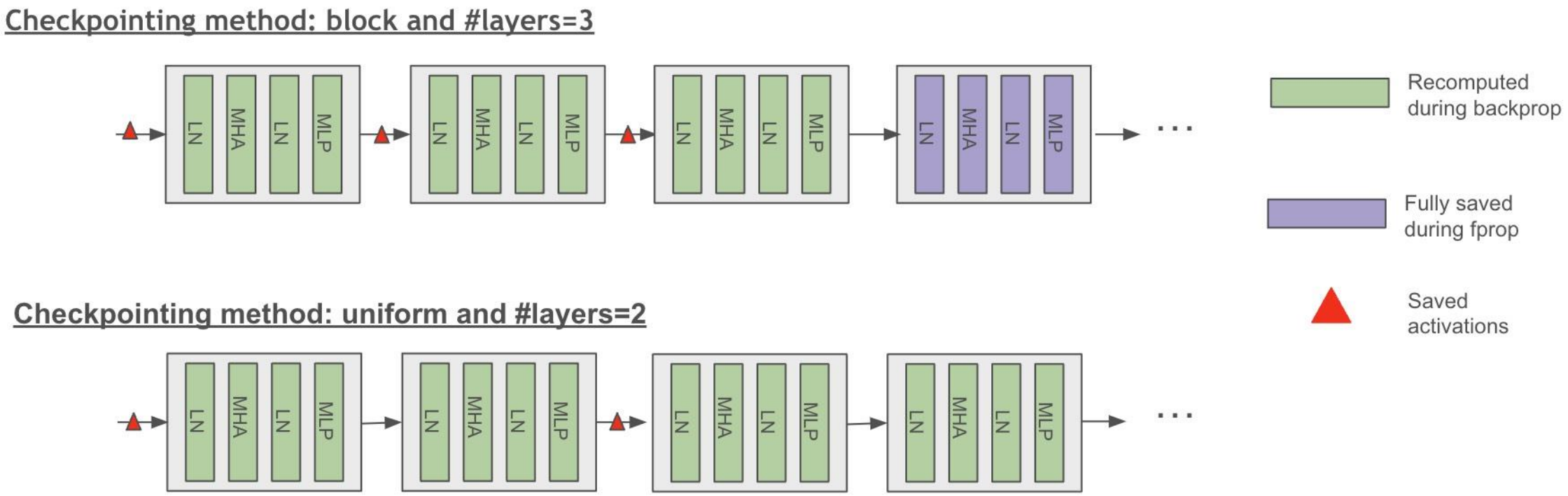
- Activation recomputation (simple)



setting *activations_checkpoint_granularity = selective or full*

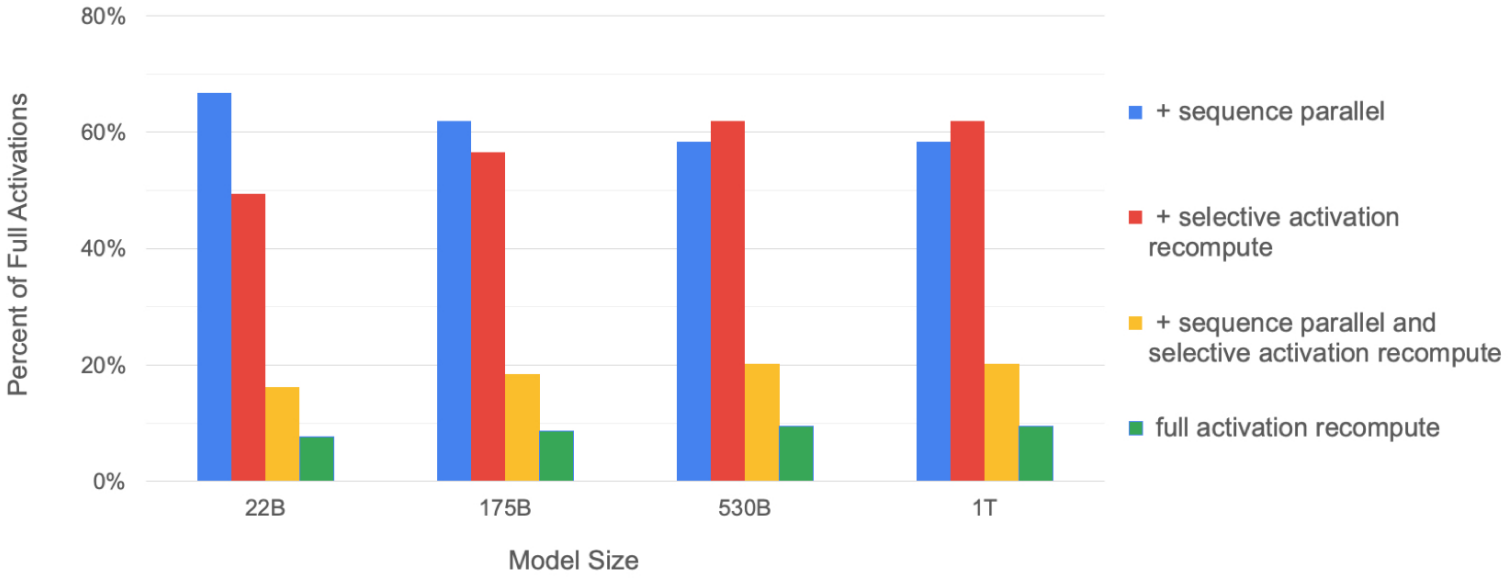
LLM training: DP with Memory Optimization

- Activation recomputation (simple)

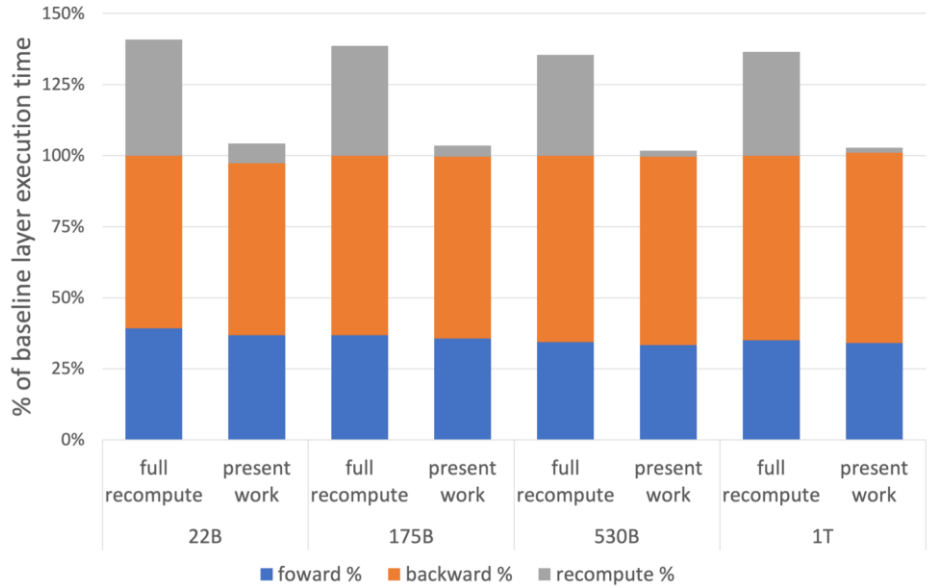


LLM training: DP with Memory Optimization

- Advantage of Re-computation



Recomputation saves memory footprint



Recomputation brings more compute loads

《Reducing Activation Recomputation in Large Transformer Models》

Thanks!