Data Parallelism

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

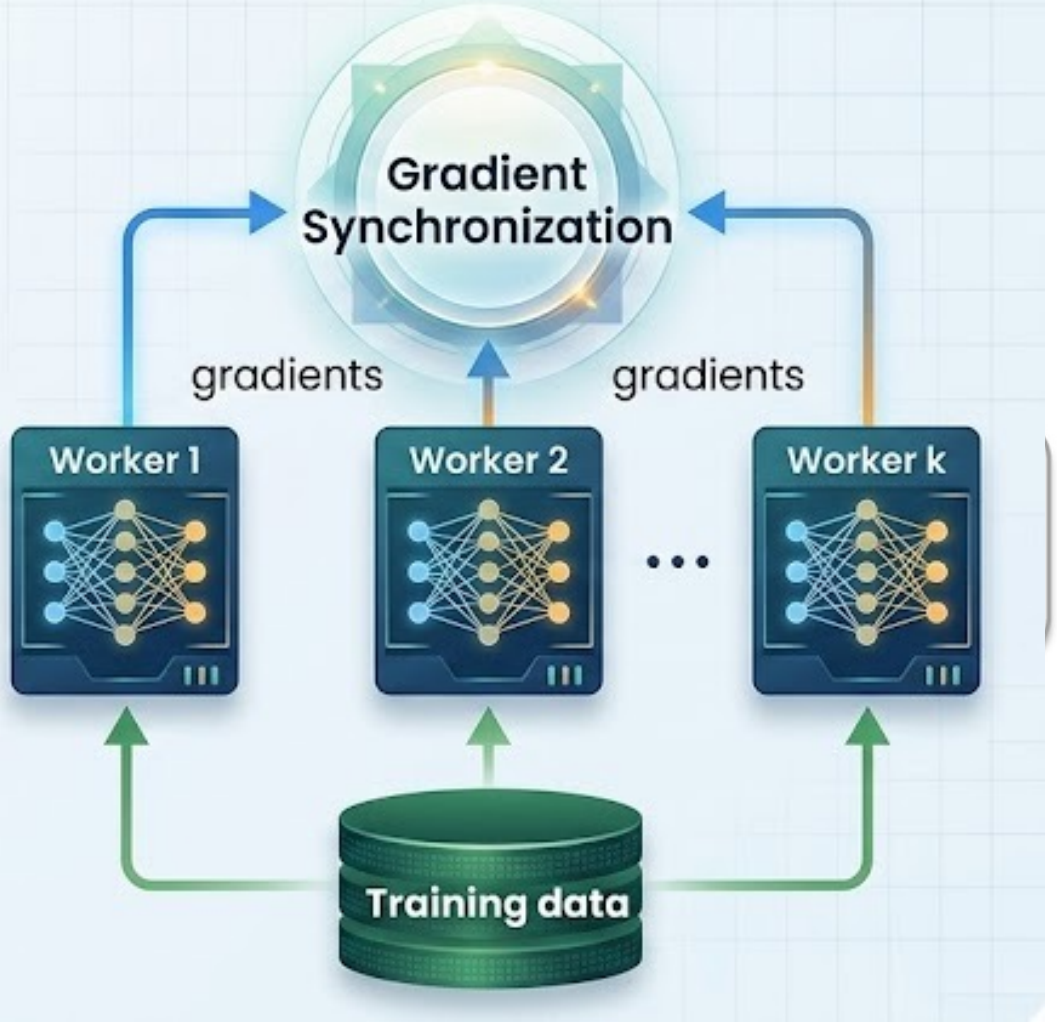
- Transformer: A Quick View
- Data Parallelism
  - **Parameter-Server**
  - All-Reduce
  - Memory Optimization

Distributed LLM Training: Outline

Data Parallelism

- Distribute data to workers
- Each worker work independently
- Synchronize gradients via different approaches
- Repeat the above procedures until model convergence

Aggregate compute power



Parameter Server Architecture

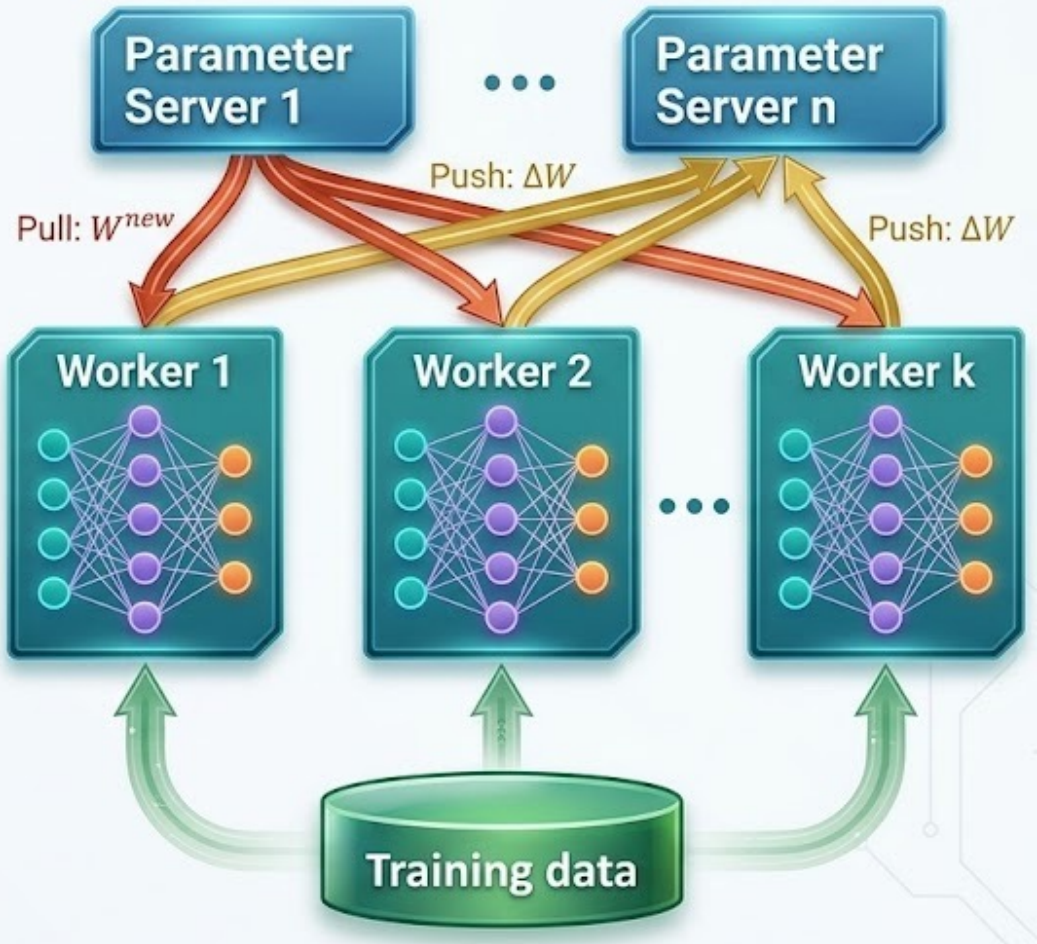
Update: $W^{new} = W - \gamma \Delta W$

5. Pull: W^{new}

3. Push: ΔW

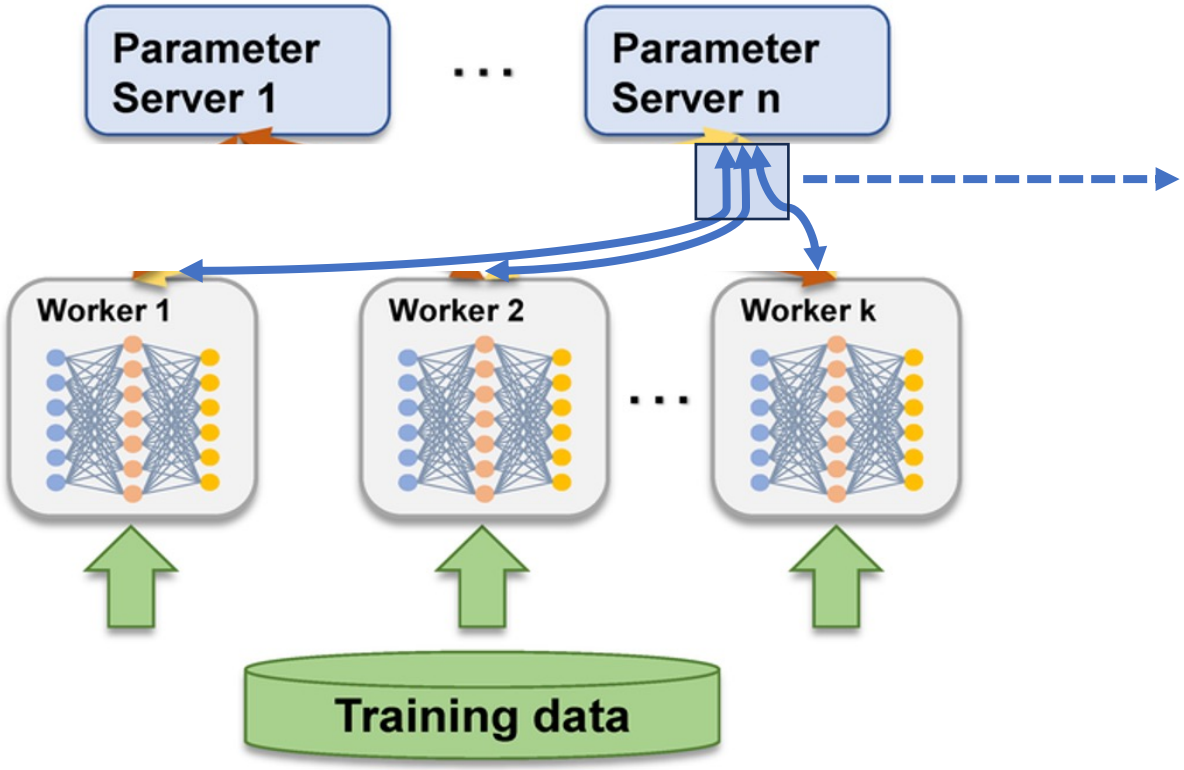
2. back propagation

1. forward propagation



Distributed Parameter Server Architecture: Augmenting Bandwidth

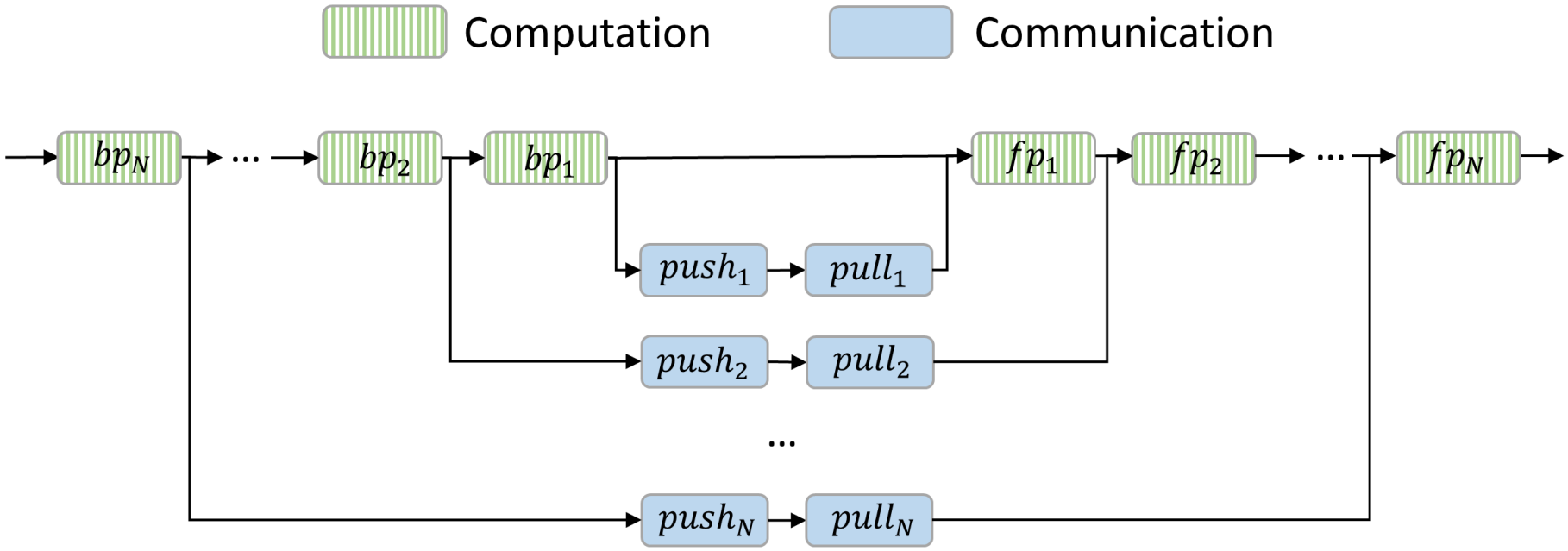
Parameter Server Architecture



- Shared bottleneck
  - Communication throttles computation
  - Reduced bandwidth efficiency due to multi-flow competition
  - Difficult to overlap comp. and comm., push and pull

Network Bottleneck

Optimizing PS Architecture via Scheduling

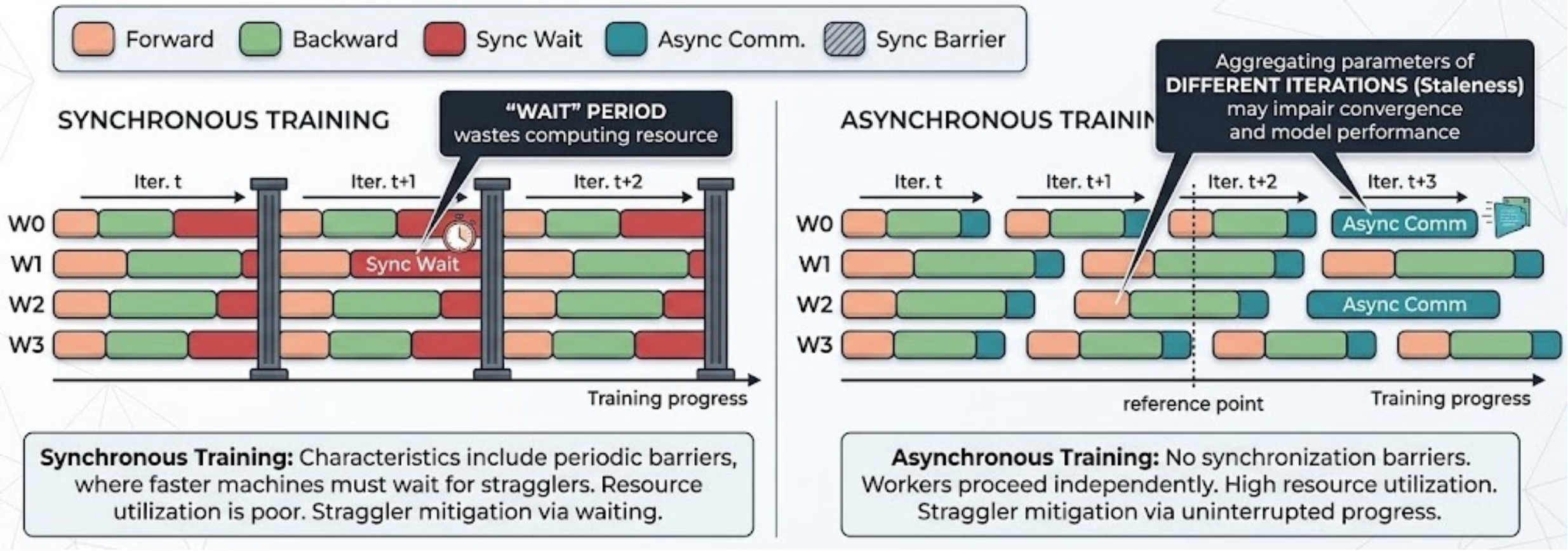


Computation order: bp_N → bp_{N-1} → … → bp_2 → bp_1 → fp_1 → fp_2 → … → fp_N

Data availability order: $gradient_N$ → $gradient_{N-1}$ → … → $gradient_2$ → $gradient_1$

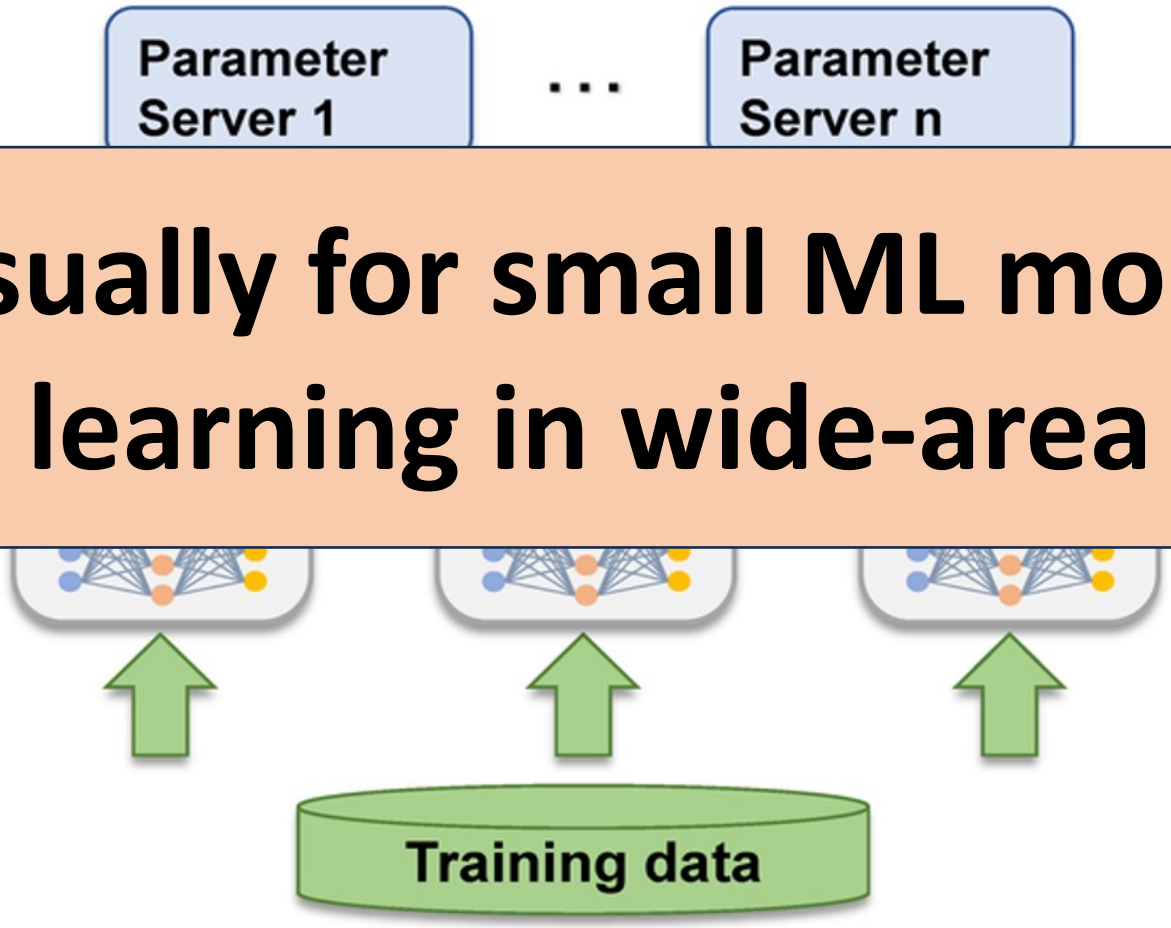
Layer-wise Computation and Communication for a DNN with three layers

Optimizing PS Architecture via Asynchronism



Asynchronous Training: Mitigating Stragglers

Parameter Server Architecture



Simple, usually for small ML models and distributed learning in wide-area networks

Distributed LLM Training: Outline

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

Collective Communication Basics

- Collective Communication

> *"**Collective communication** is communication that involves a group of processing elements (termed nodes in this entry) and affects a data transfer between all or some of these processing elements. Data transfer may include the application of a reduction operator or other transformation of the data." 《Encyclopedia of Parallel Computing》*

集合通信是指一个进程组的所有进程都参与的全局通信操作。

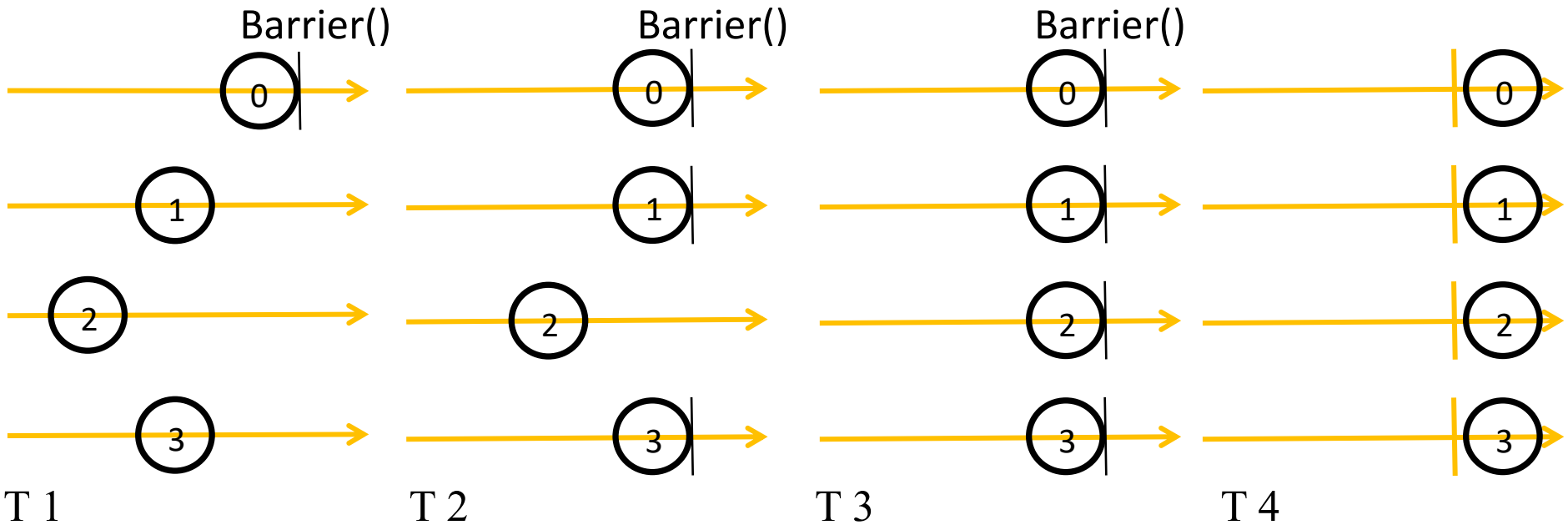
全部点对点通信完成才算集合通信完成

Collective Communication Basics

- Why Collective Communication
  - Simplified Programming Interface
    - Developers don't need to manually code complex synchronization and data distribution logic for every scenario.
  - Scalability for Large-Scale Systems
    - As systems grow to thousands of nodes or GPUs (e.g., in supercomputers or cloud clusters), managing communication becomes exponentially complex. These libraries support sparse data handling, fault tolerance, and observability features to ensure reliable operation at scale.
  - Decompose Compute and Communication
    - Allowing machine learning researchers and system engineers to work on their own.

Collective Communication Basics

- Collective Communication: Basic Operations
  - **SEND**, **RECEIVE**, COPY, BARRIER, SIGNAL+WAIL (in Message Passing Interface, i.e. MPI)

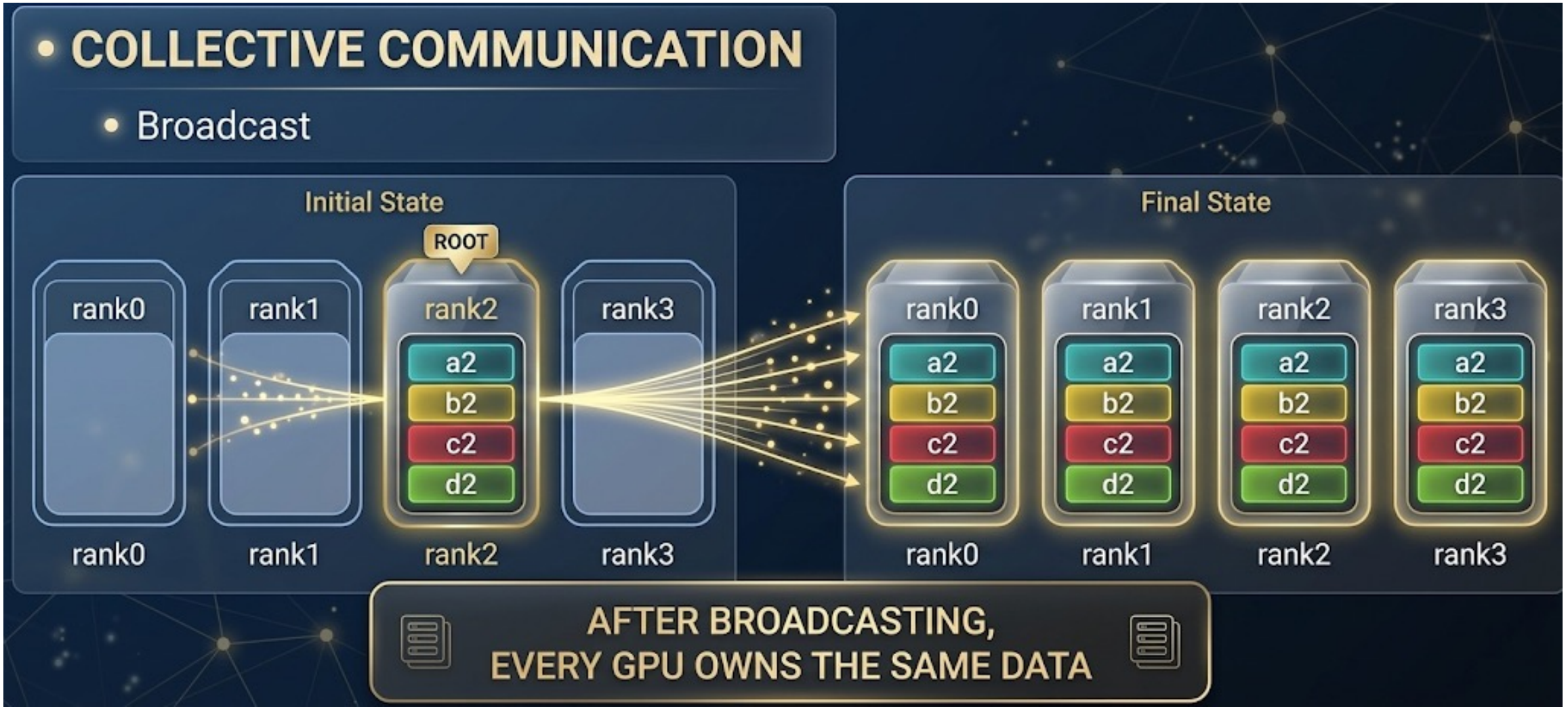


* Pictures replotted, but originated from Mengyuan @ Zhihu

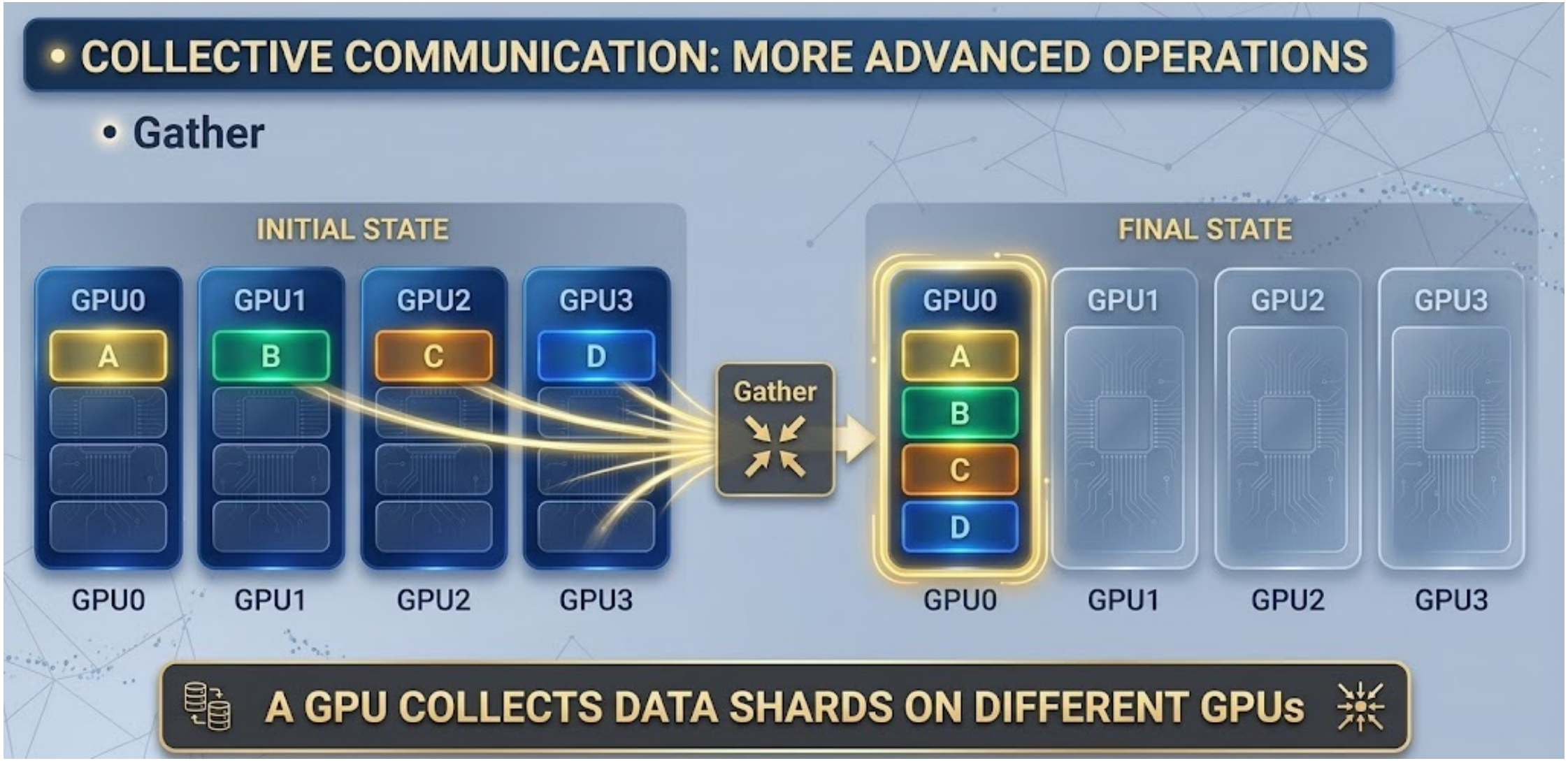
Collective Communication Basics

- Collective Communication: More Advanced Operations
  - Broadcast: one-to-many
  - Gather: many-to-one, and All-Gather: many-to-many
  - Scatter: one-to-many
  - Reduce: many-to-one, and All-Reduce: many-to-many
  - Reduce-Scatter: aggregate data and then transmit
  - All-to-All: many-to-many
  - AllReduce: ?

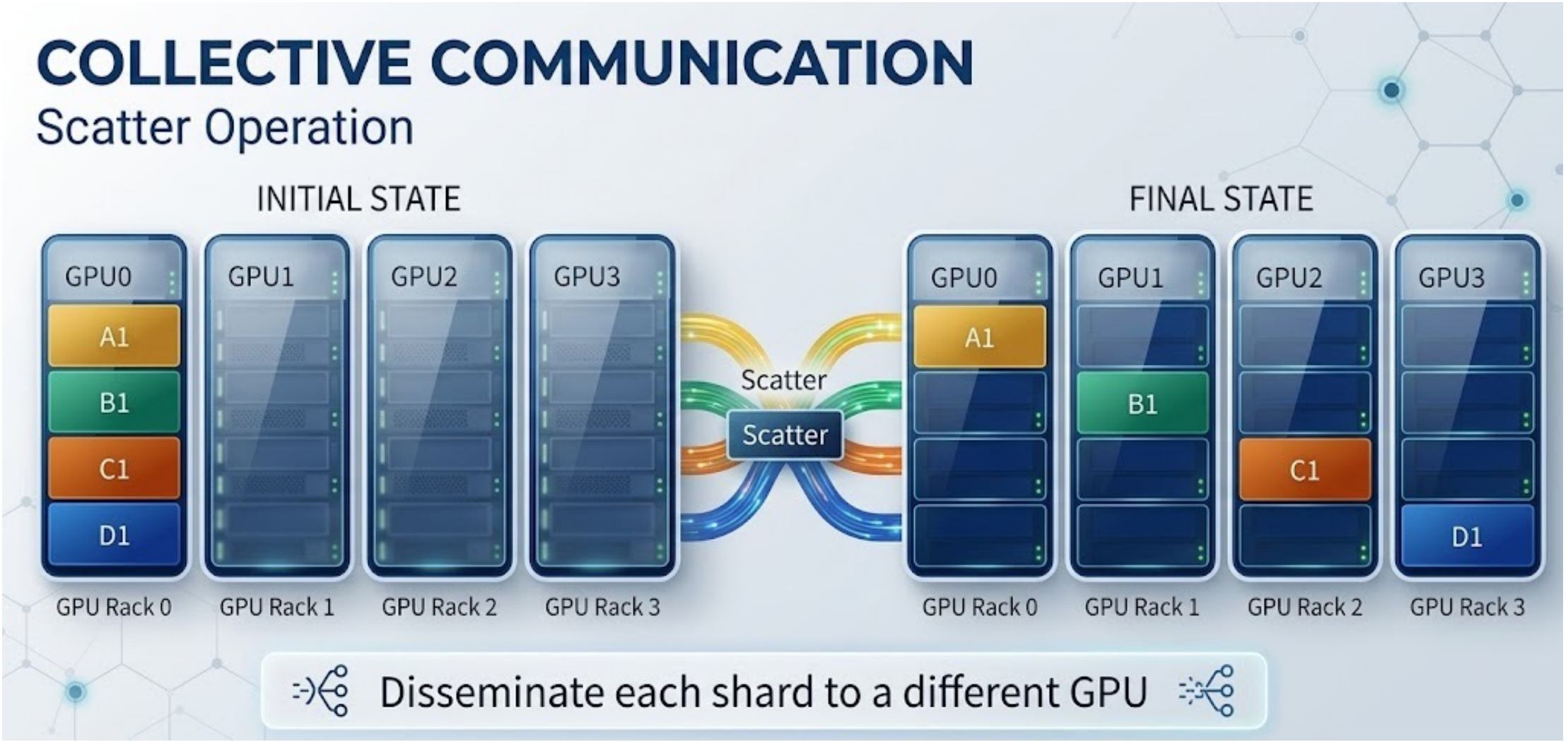
Collective Communication Basics



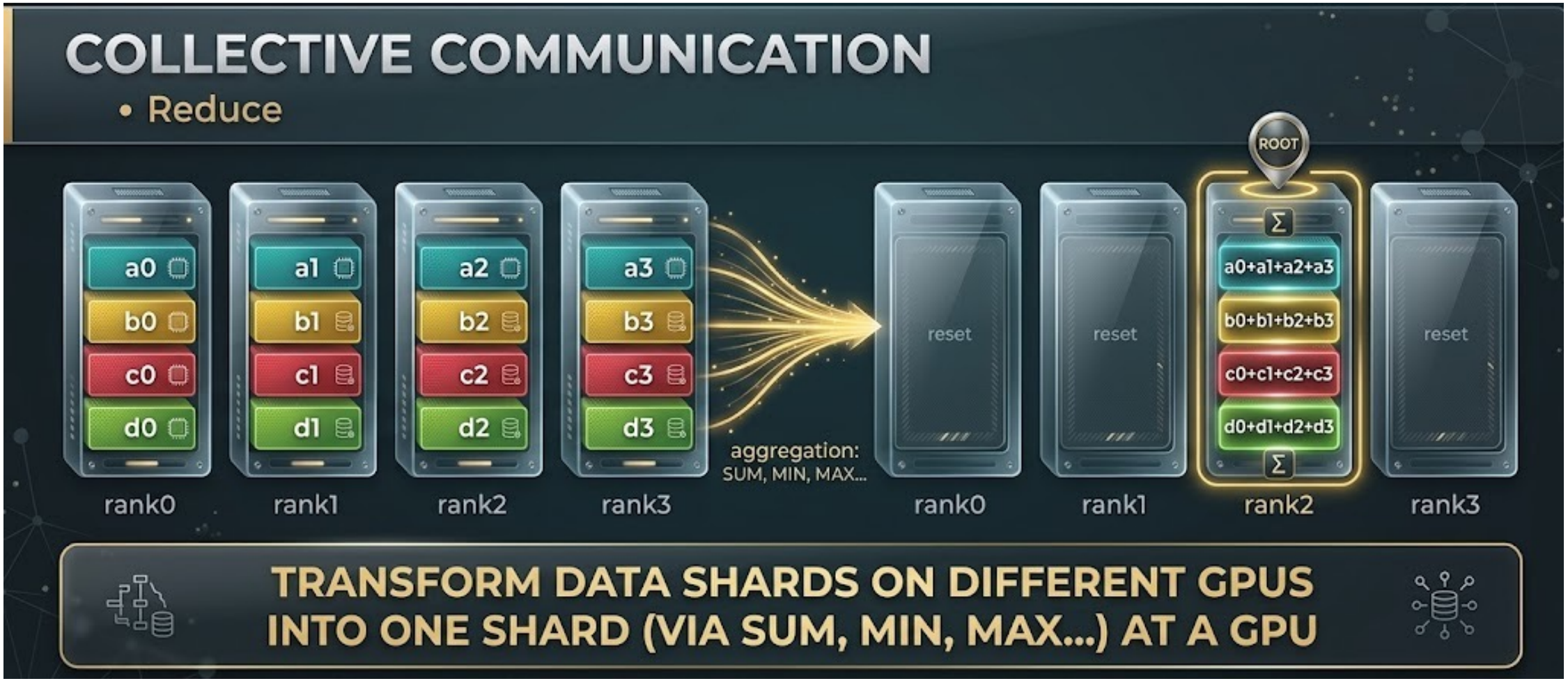
Collective Communication Basics



Collective Communication Basics



Collective Communication Basics



Collective Communication Basics

Collective Communication Basics

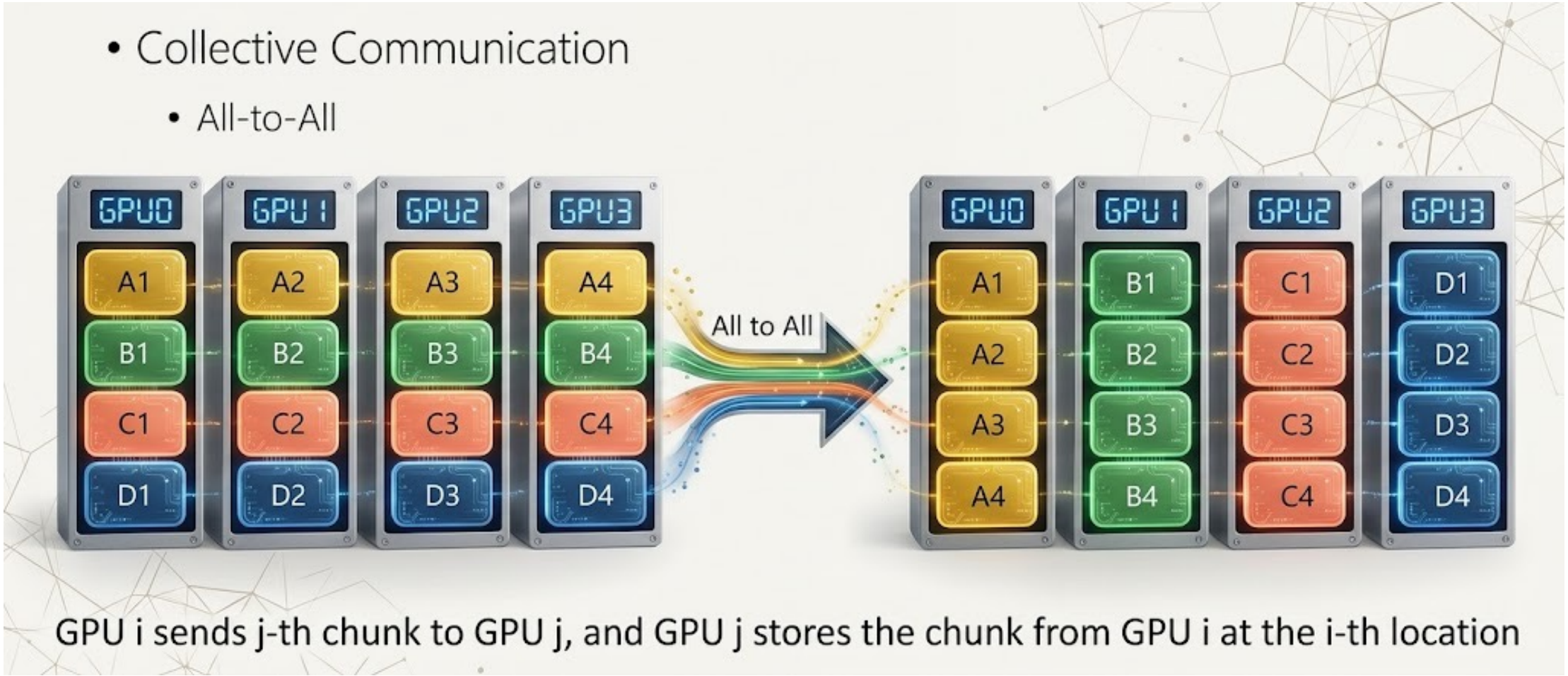
- Collective Communication
  - All-Gather

Collective Communication Basics

- **Collective Communication**
  - Reduce-Scatter

Aggregate data chunks and store one shard at a GPU

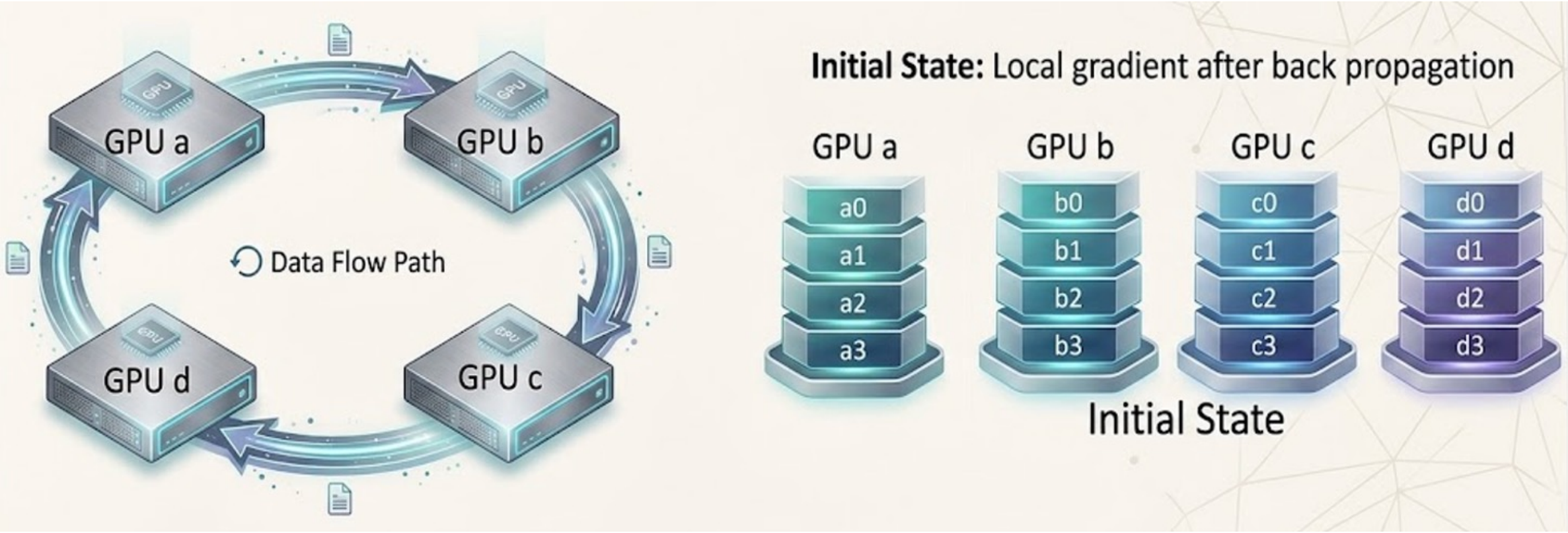
Collective Communication Basics



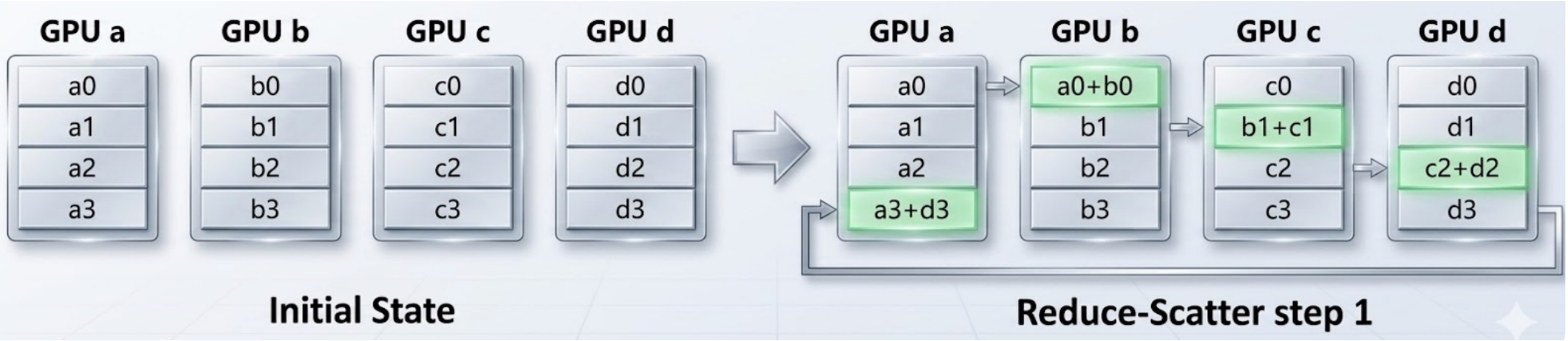
All-to-All is purely a data routing and memory transposition operation

Starting from the most important “All-Reduce”

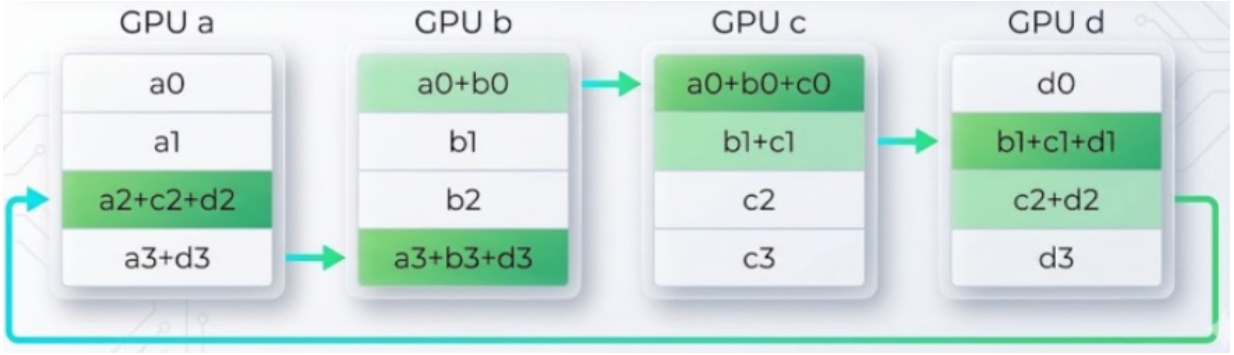
Ring All-Reduce



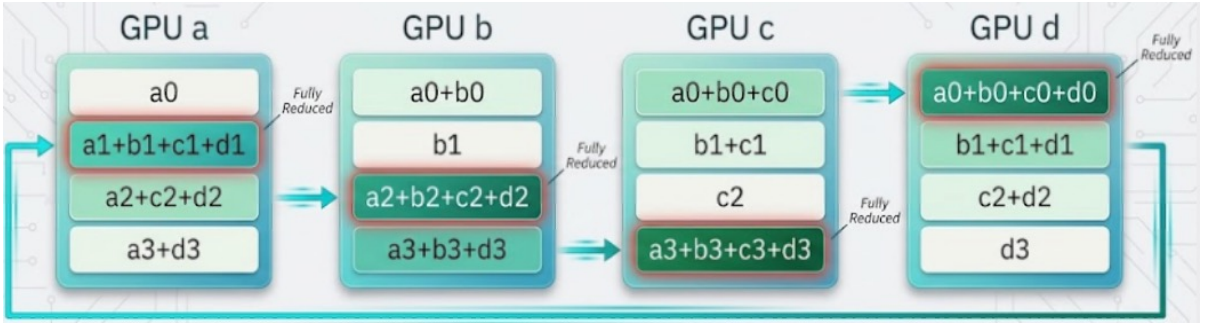
Ring All-Reduce



Ring All-Reduce



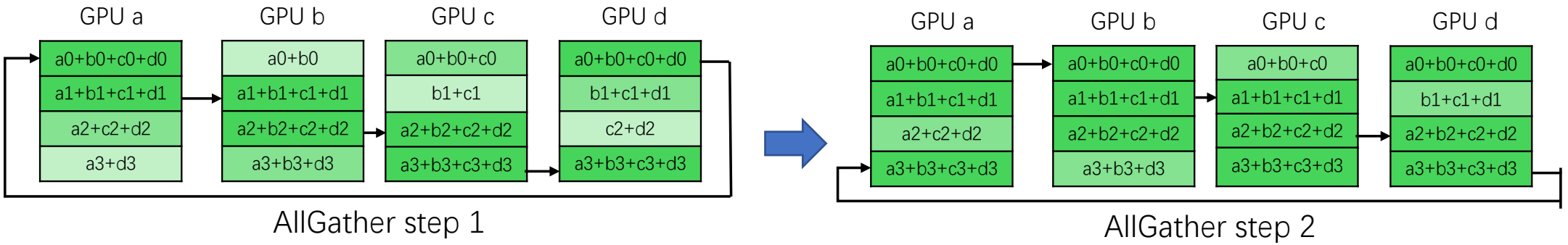
Reduce-Scatter Step 2



Reduce-Scatter Step 3

Ring All-Reduce

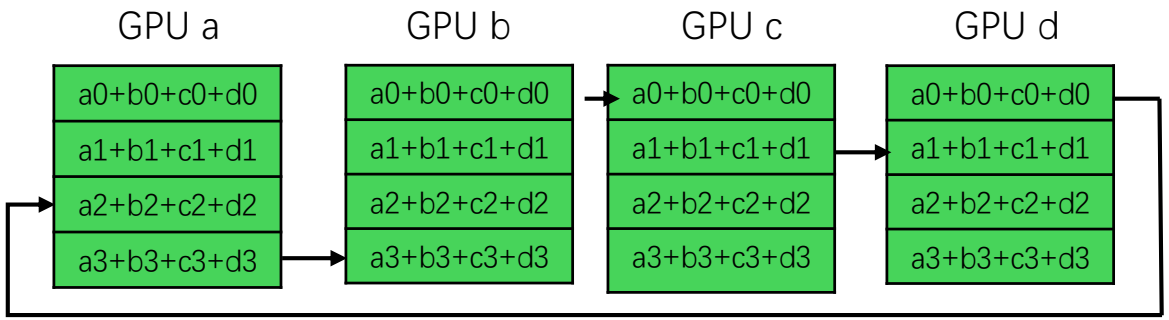
- Ring All-Reduce



All-Gather

Ring All-Reduce

- Ring All-Reduce



AllGather

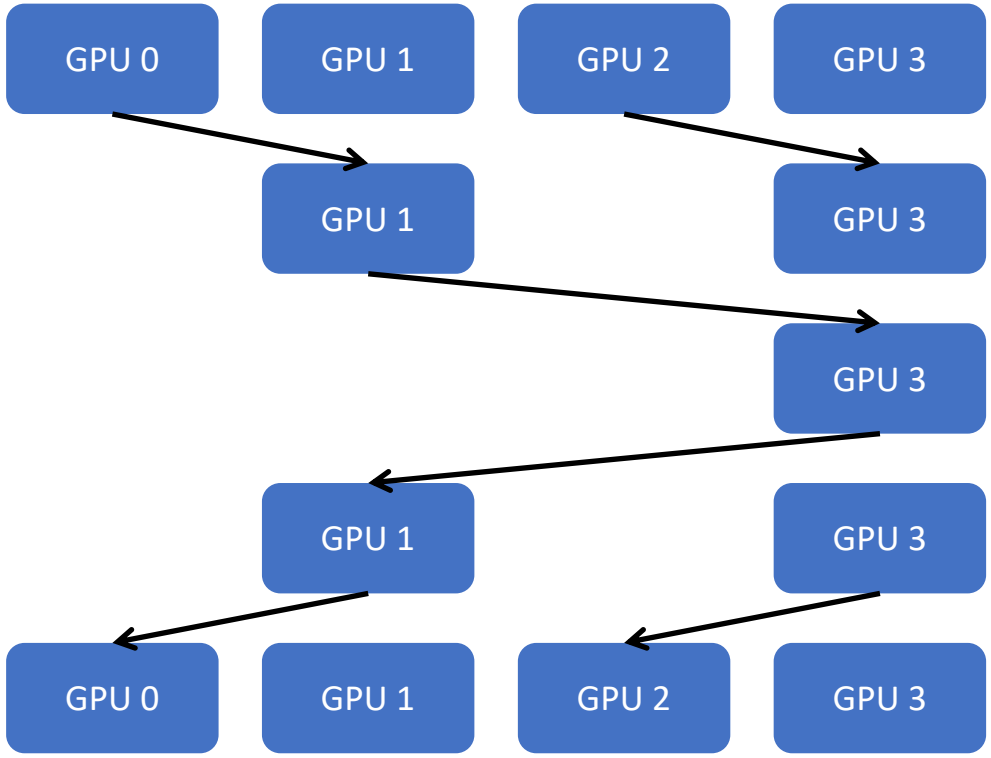
- Reduce-Scatter
  - N: # of GPUs, S: per-GPU data volume
  - $N-1$ rounds
  - S/N data transmission per round
- All-Gather
  - One round boradcasting or $N-1$ clockwise rounds
  - Data volume: $(N-1)*S/N$
- Total traffic per GPU
  - $\boldsymbol{2*(N-1)*S/N}$

Ring All-Reduce

“All-Reduce = Reduce-Scatter + All-Gather”

Tree All-Reduce

- Tree All-Reduce

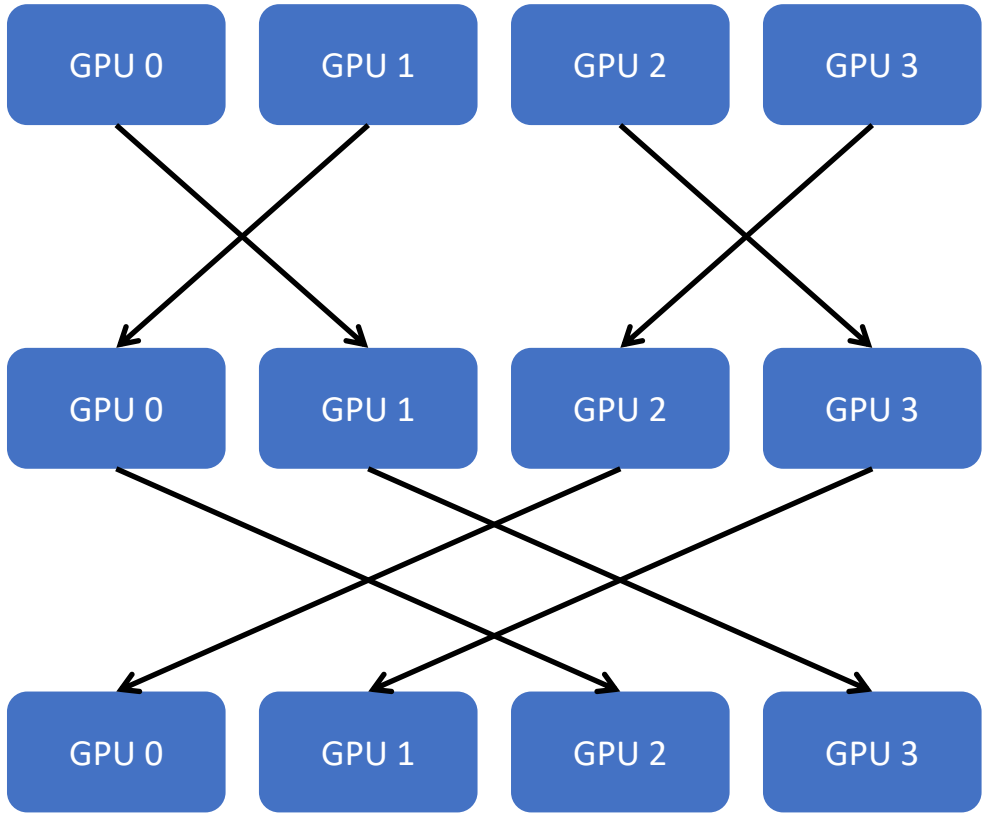


Data communication

- Upward
  - $\log_2 N$ rounds
  - At most S data per round
- Downward
  - $\log_2 N$ rounds
  - At most S data per round
- Total traffic per GPU
  - $2 * S * \log_2 N$ (naïve transmission without overlapping)

Butterfly All-Reduce

- Butterfly All-Reduce

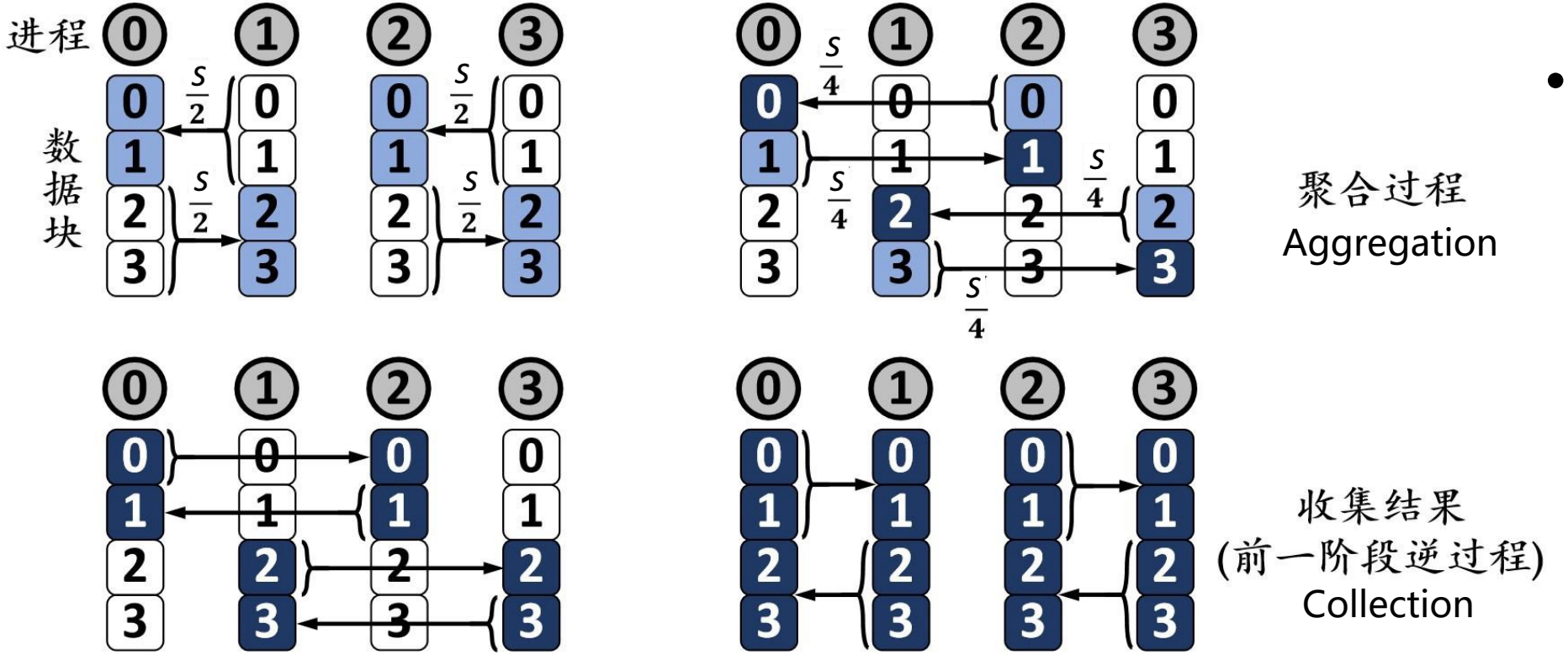


Butterfly Reduce: Utilizing bidirectional bandwidth

- Butterfly Reduce
  - $\log_2 N$ rounds
  - S data per round
- Total Traffic per-GPU
  - $\log_2 N * S$

Rabenseifner All-Reduce

- Rabenseifner Reduce



Rabenseifner Algorithm (~ recursive halving-doubling)

- Total Traffic per-GPU
  - $2\log_2 N$ rounds
  - Different traffic load at every round
  - Total traffic volume: $2\left(\frac{S}{2}+\frac{S}{4}+\cdots+\frac{S}{N}\right) \approx 2\frac{N-1}{N}S$

《Allreduce算法及其硬件加速方法介绍》

Cross-Comparison

- Communication Cost
  - Assumption: each GPU can send and receive data simultaneously
  - Classical α-β model: latency = $\alpha + \beta \cdot \frac{S}{B}$

"start-up" delay

transmission efficiency

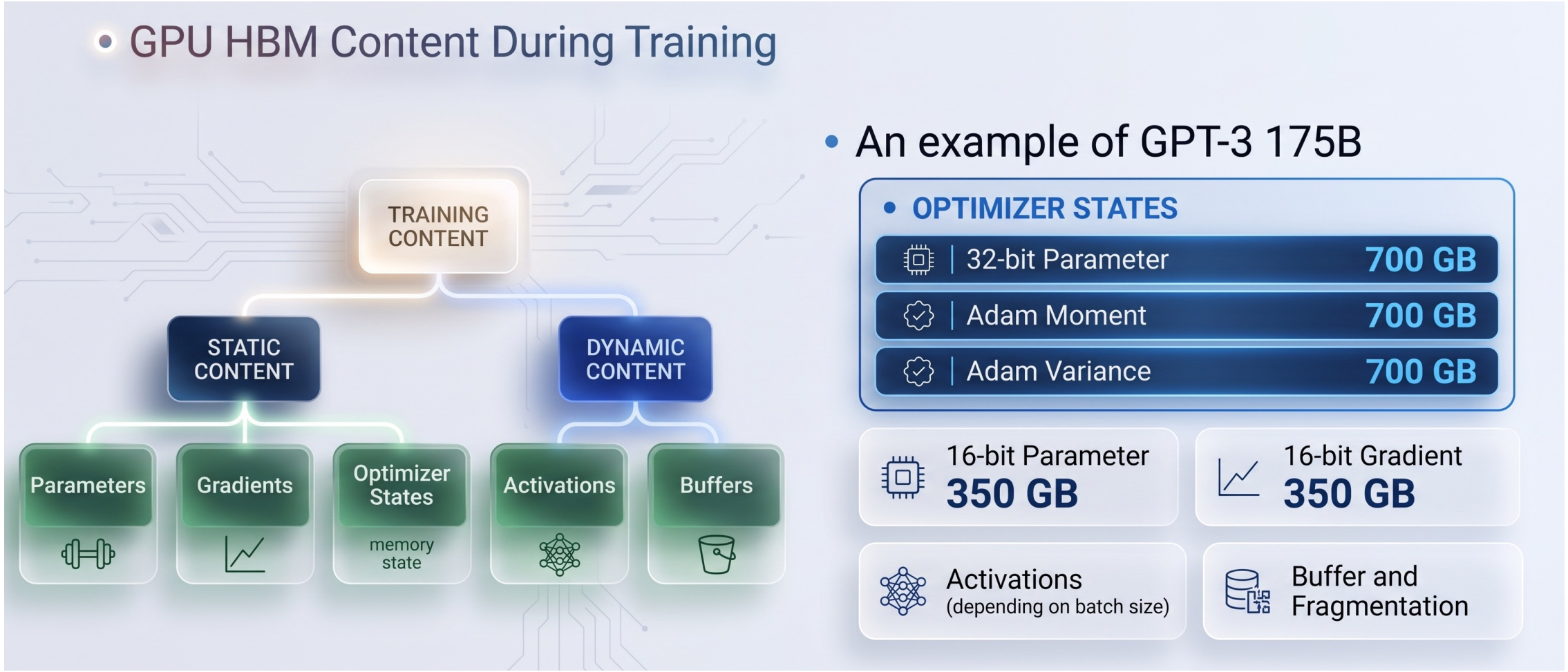
Message size/bandwidth

Algorithm	Rounds	Traffic Load	Cost	Pros and Cons
Ring	2(N-1)	2S(N-1)/N	*2(N-1)*(α + β∗S/N/B)*	▪ Large data chunk aggregation ▪ Relatively small # of GPUs ▪ Not suitable for short chunks ▪ Ease of implementation ▪ Utilizating bidirectional links
Rabenseifner	2 $\lceil \log N \rceil$	2S(N-1)/N	2$\lceil \log N \rceil\alpha$ + 2$(N-1)$ $\beta * S/N$/B	▪ Relatively smaller data chunk ▪ Large # of GPUs ▪ Periodically changing communication pairs

Distributed LLM Training: Outline

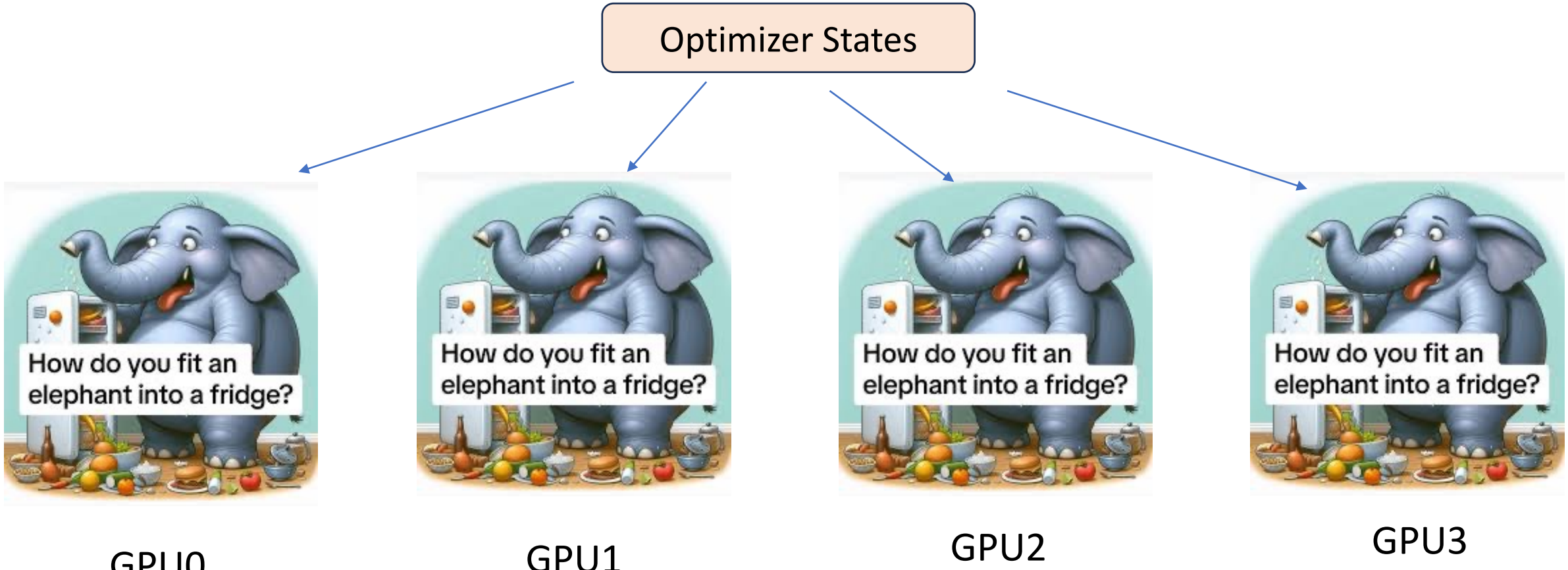
- Data Parallelism
  - Parameter-Server
  - All-Reduce
  - **Memory optimization**

LLM training: DP with Memory Optimization



LLM training: DP with Memory Optimization

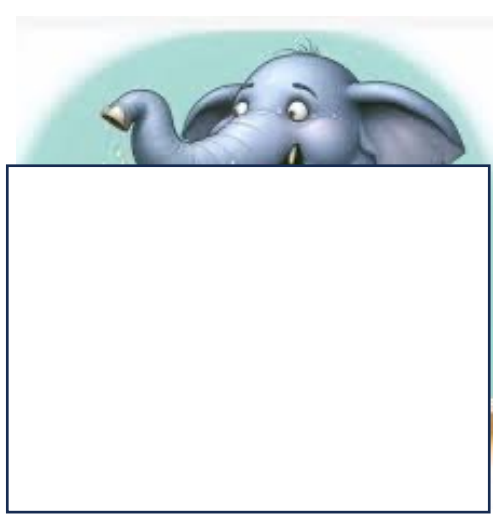
- How to put an elephant into a fridge?



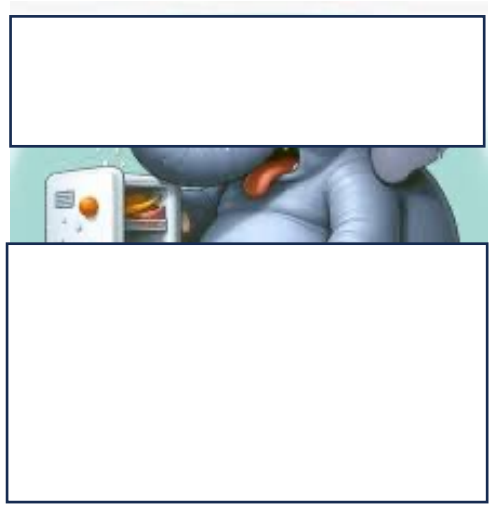
LLM training: DP with Memory Optimization

- How to put an elephant into a fridge?

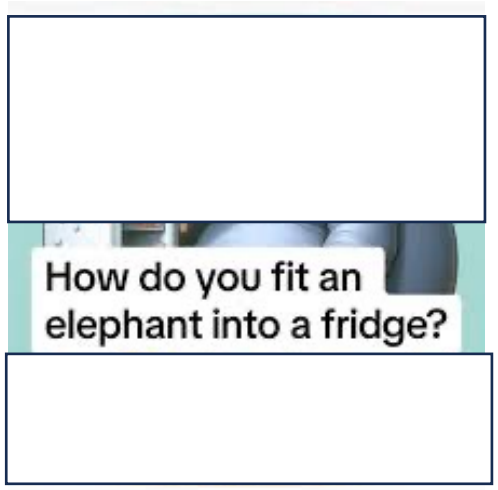
Sharding the elephant into four partitions, and each GPU HBM holds one chunk!



GPU0



GPU1

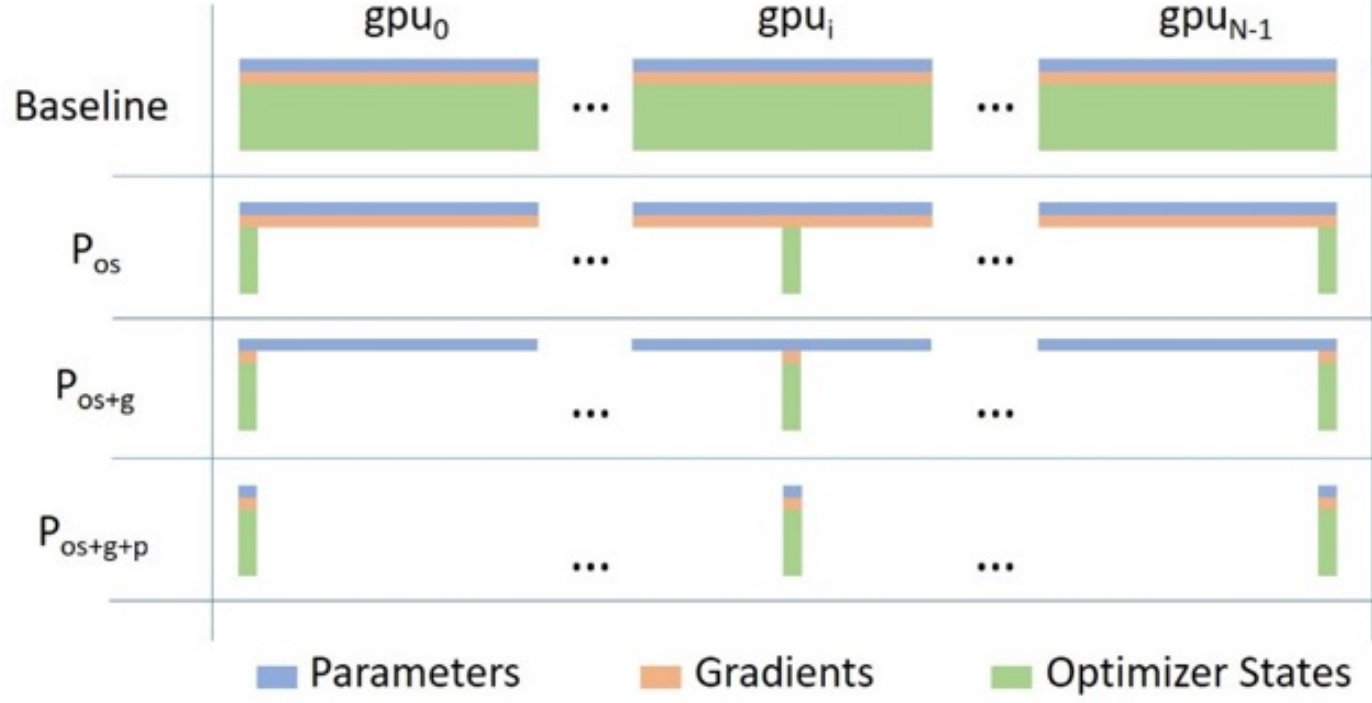


GPU2

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 (ψ for the number of parameters)

《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})

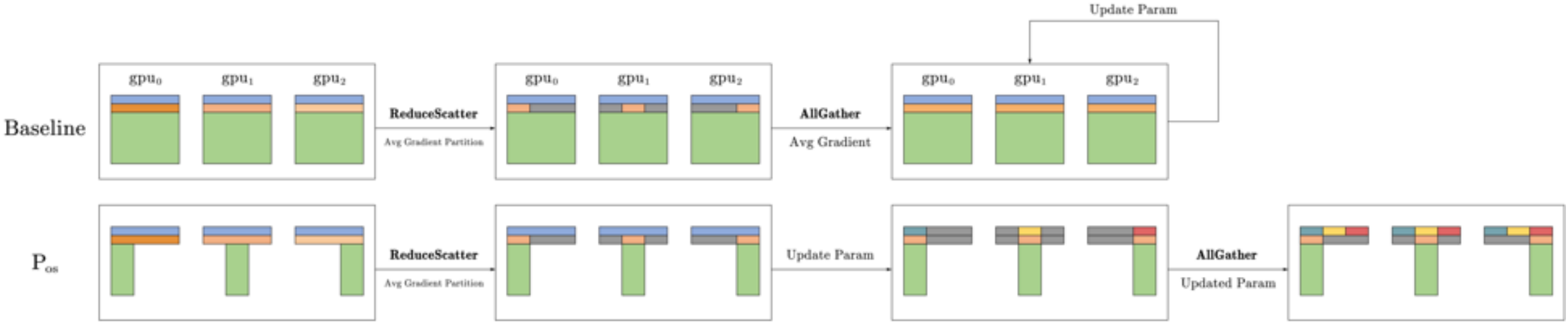
$$\text{Model States (bytes)}|P_{os+g+p} = \overbrace{\frac{16 \times \Psi}{N_d}}^{\text{Param + Gradient + Optimizer}}$$
$$\approx 0 | N_d \to \infty$$

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

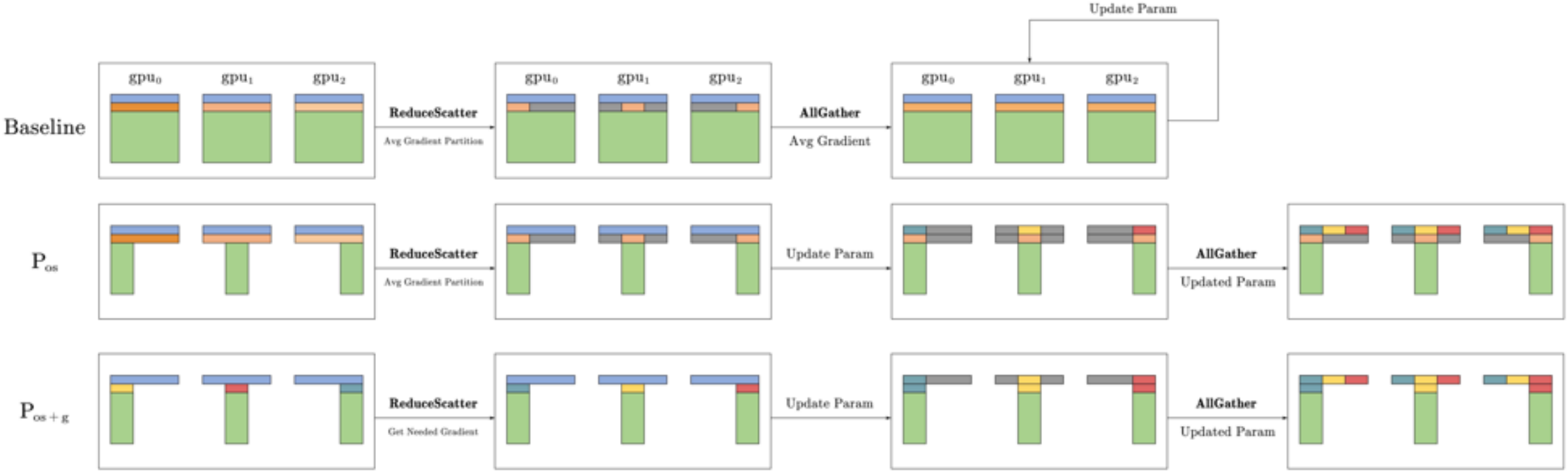
- ZeRO-1



- 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

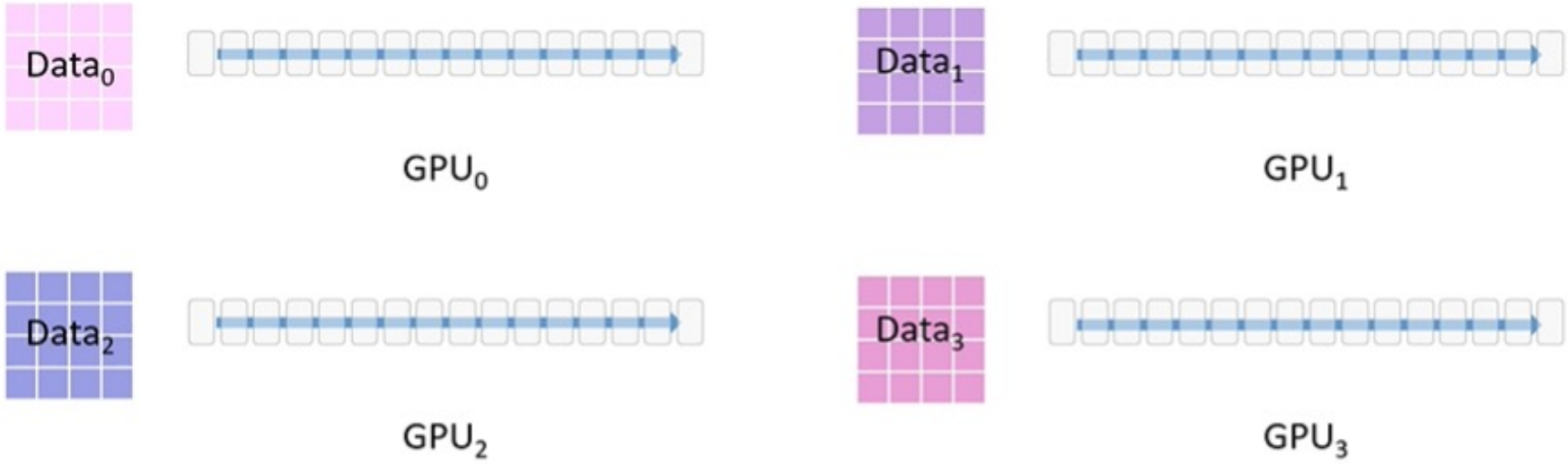
- ZeRO-3: An animation (Implementation on DeepSpeed could be somewhat different)

- 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

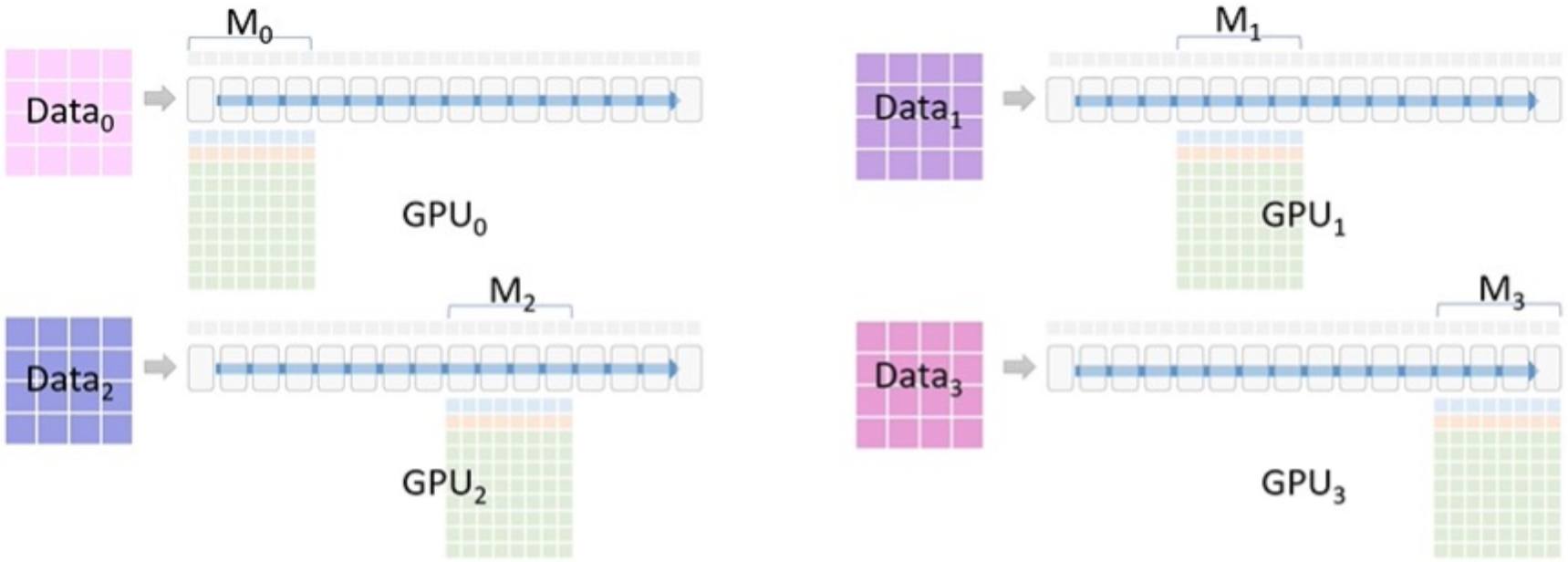
- ZeRO-3: An animation



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

LLM training: DP with Memory Optimization

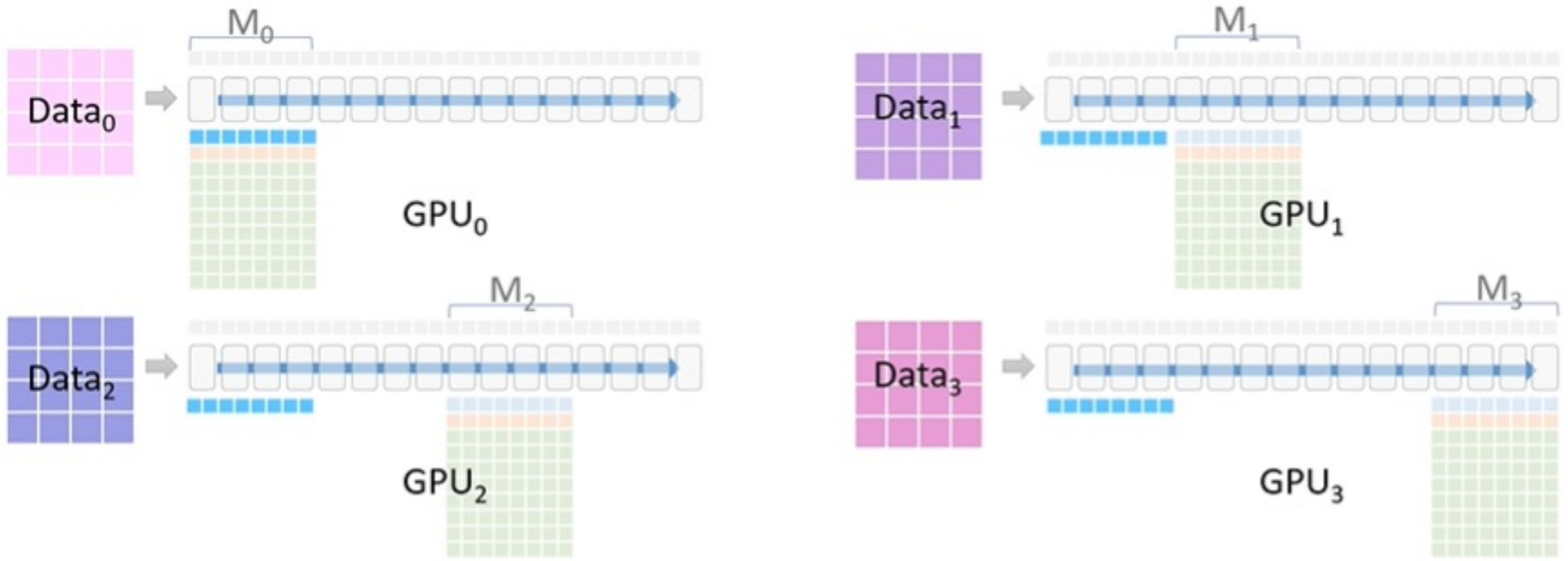
- ZeRO-3: An animation



Each GPU is responsible for one piece of the final model

LLM training: DP with Memory Optimization

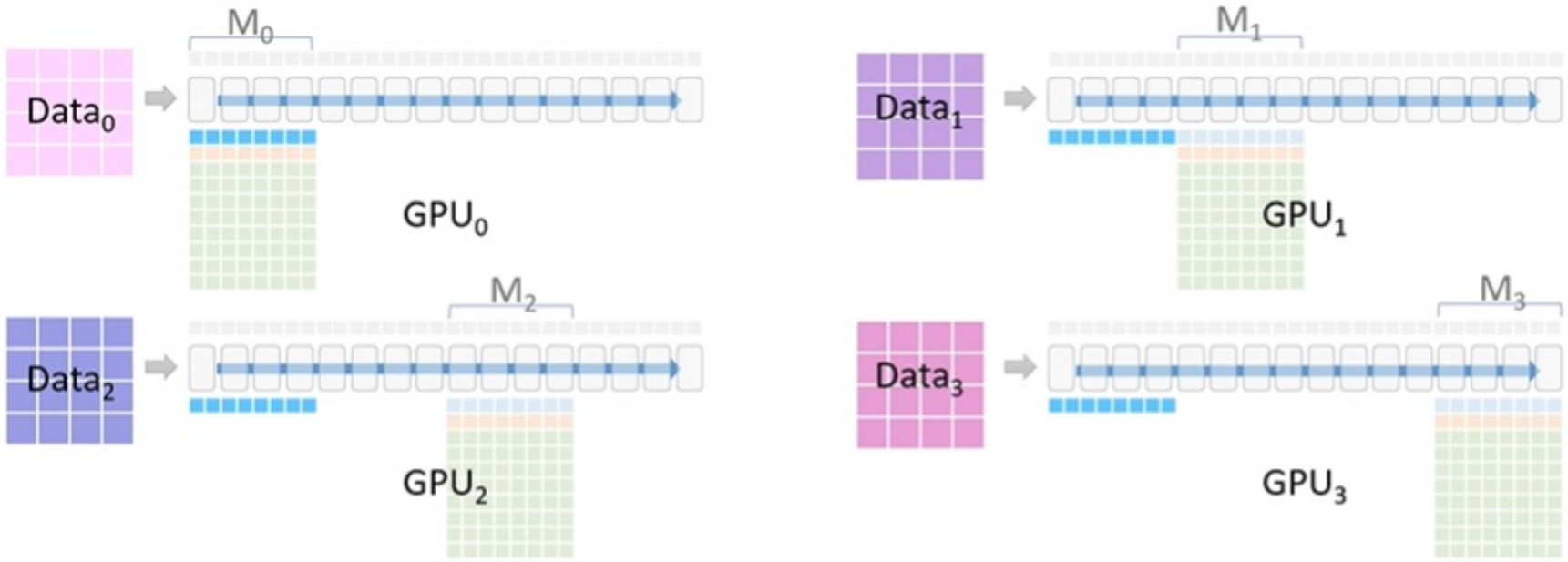
- ZeRO-3: An animation



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

LLM training: DP with Memory Optimization

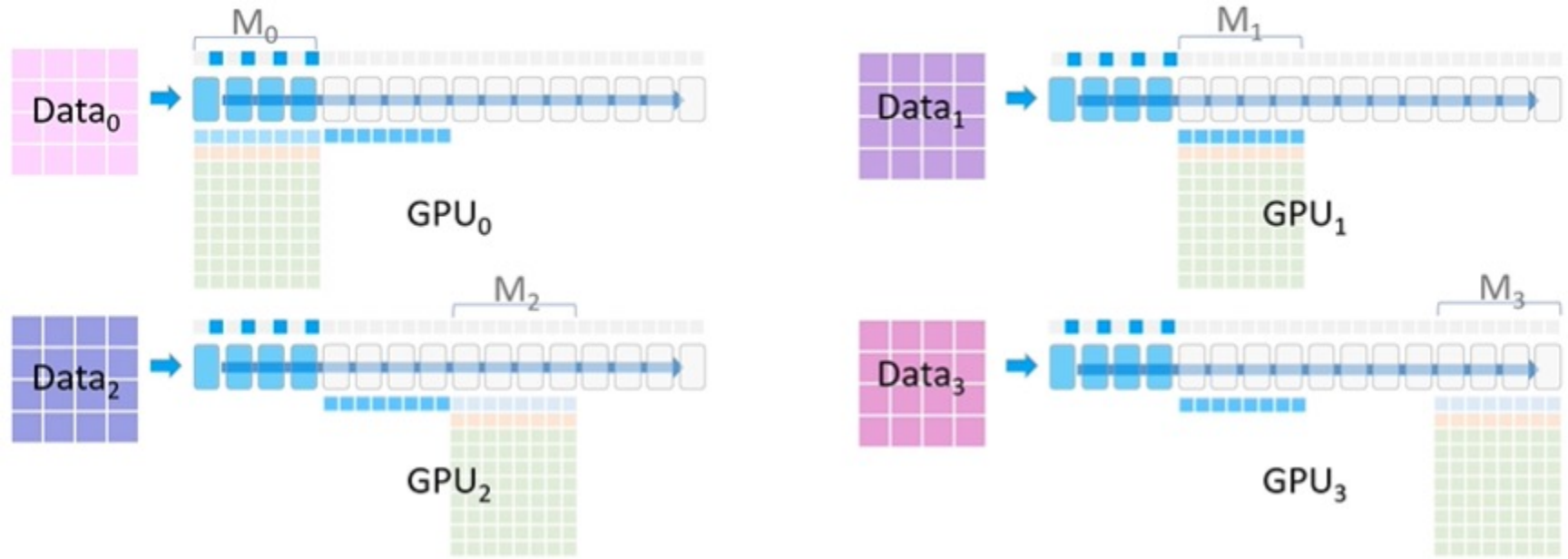
- ZeRO-3: An animation



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

LLM training: DP with Memory Optimization

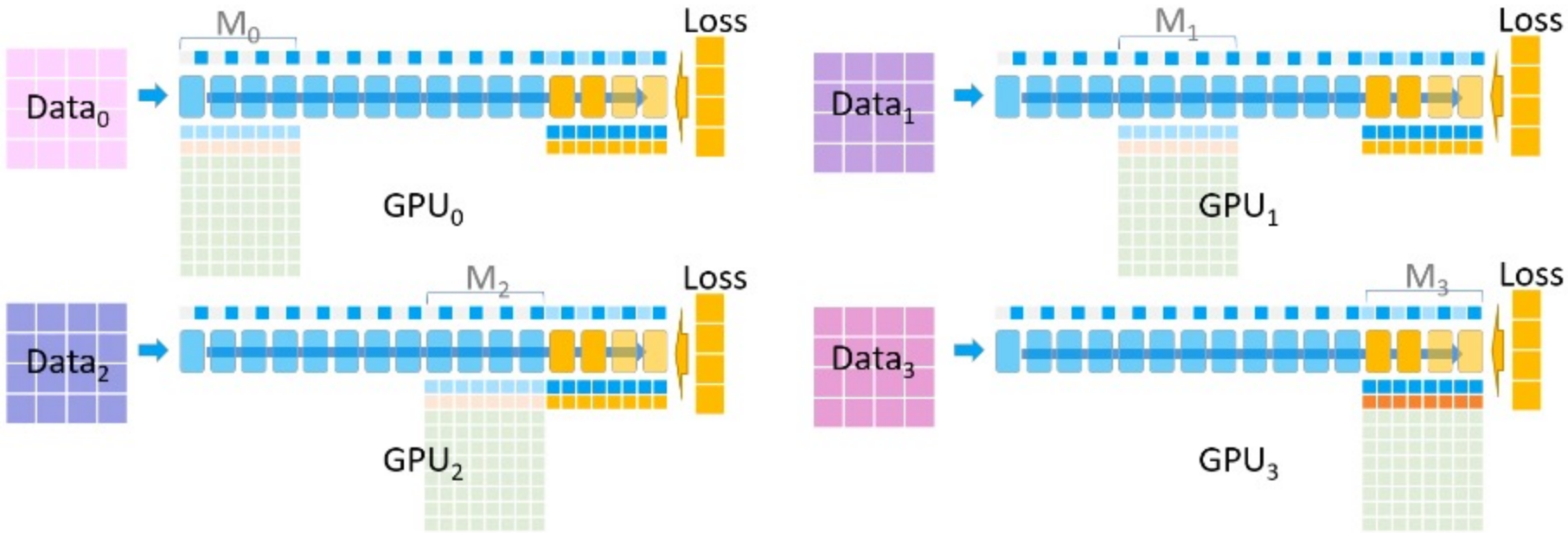
- ZeRO-3: An animation



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

LLM training: DP with Memory Optimization

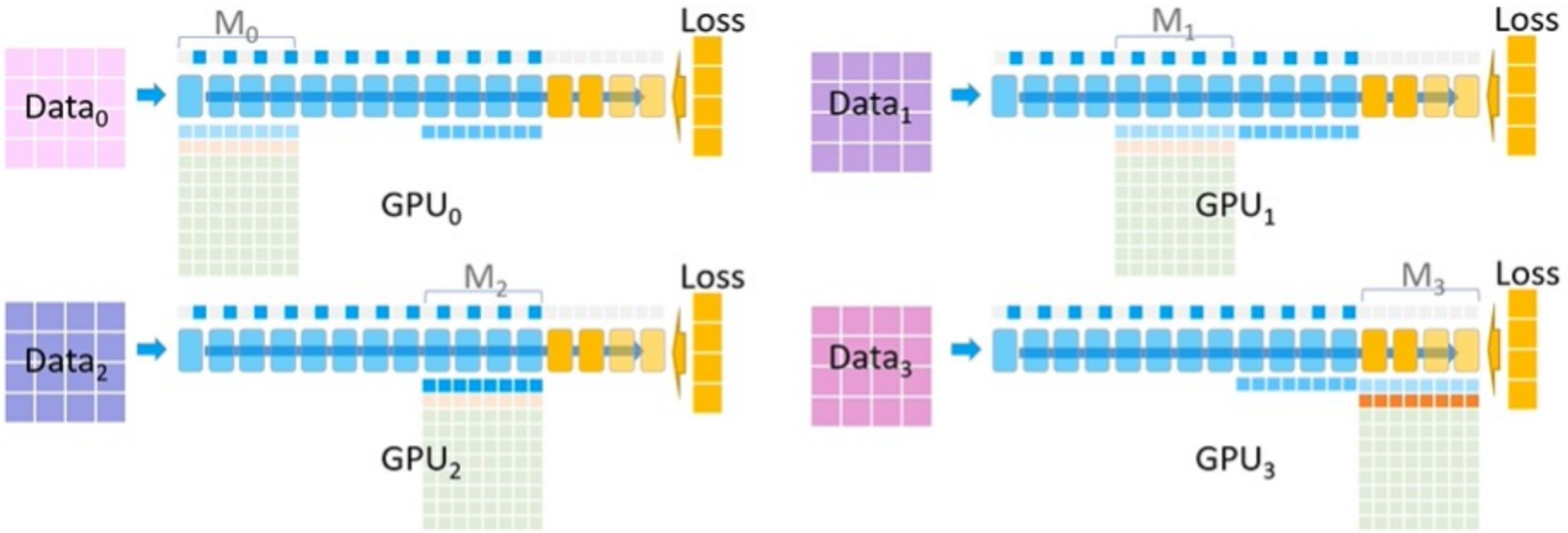
- ZeRO-3: An animation



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

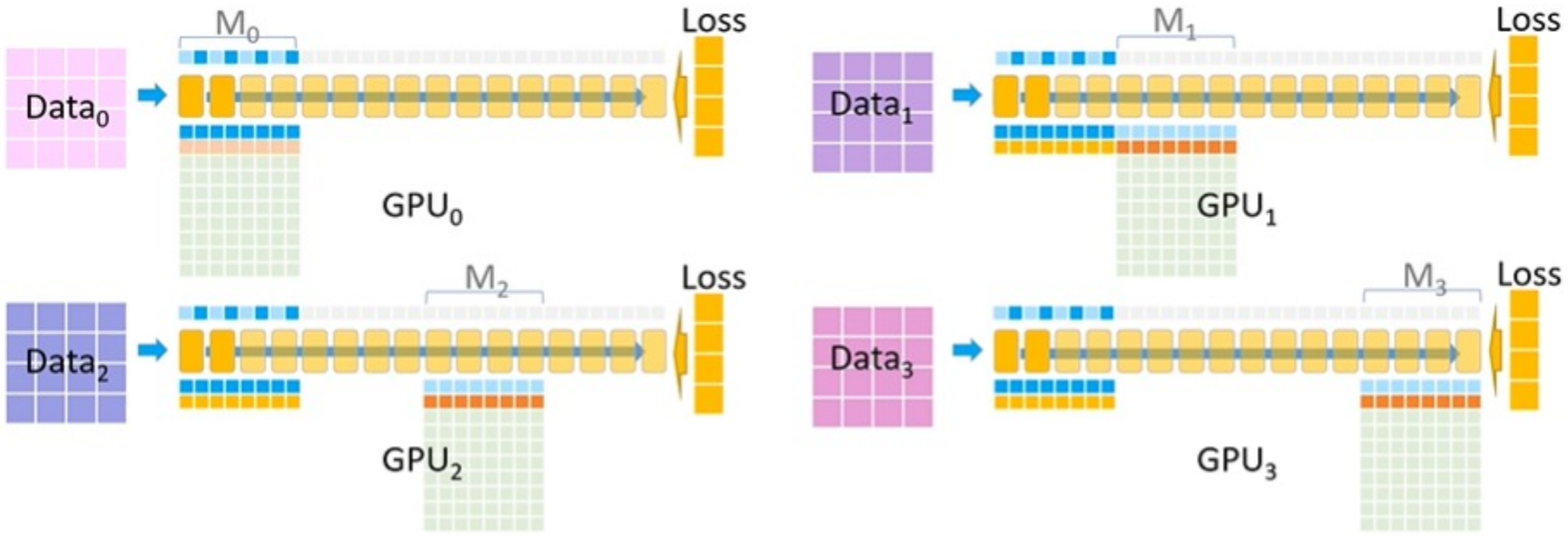
- ZeRO-3: An animation



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

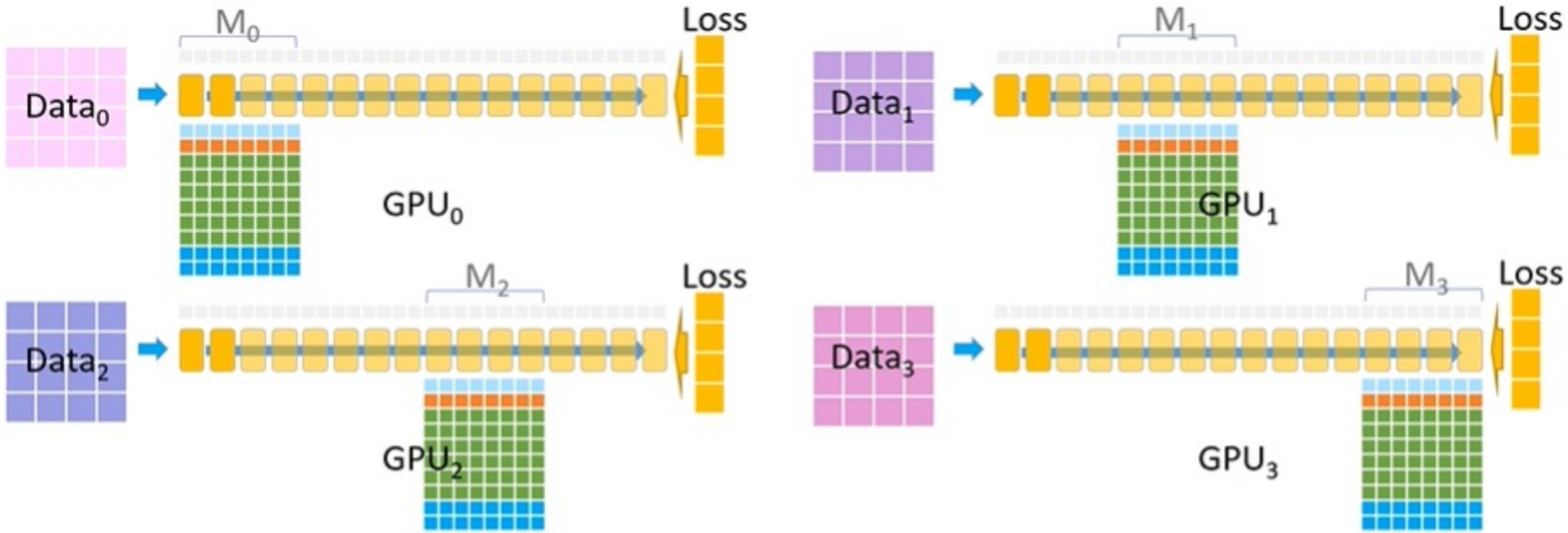
- ZeRO-3: An animation



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

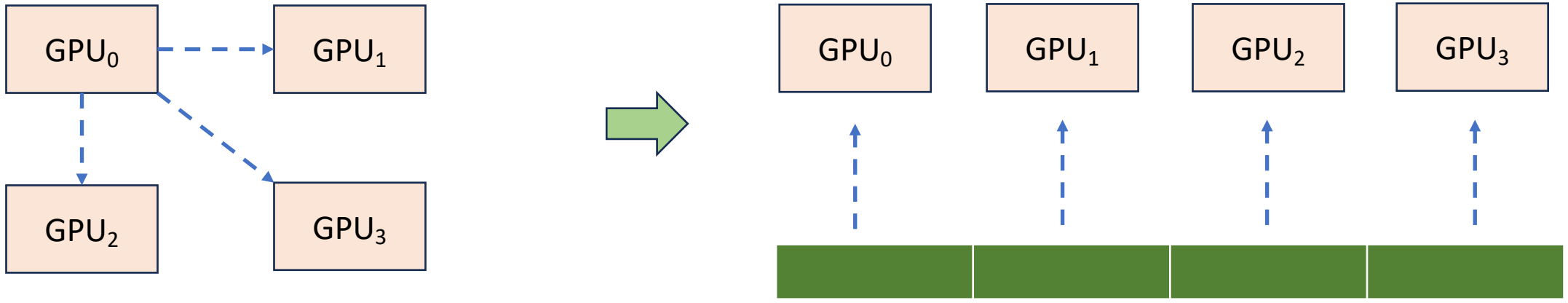
- ZeRO-3: An animation



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

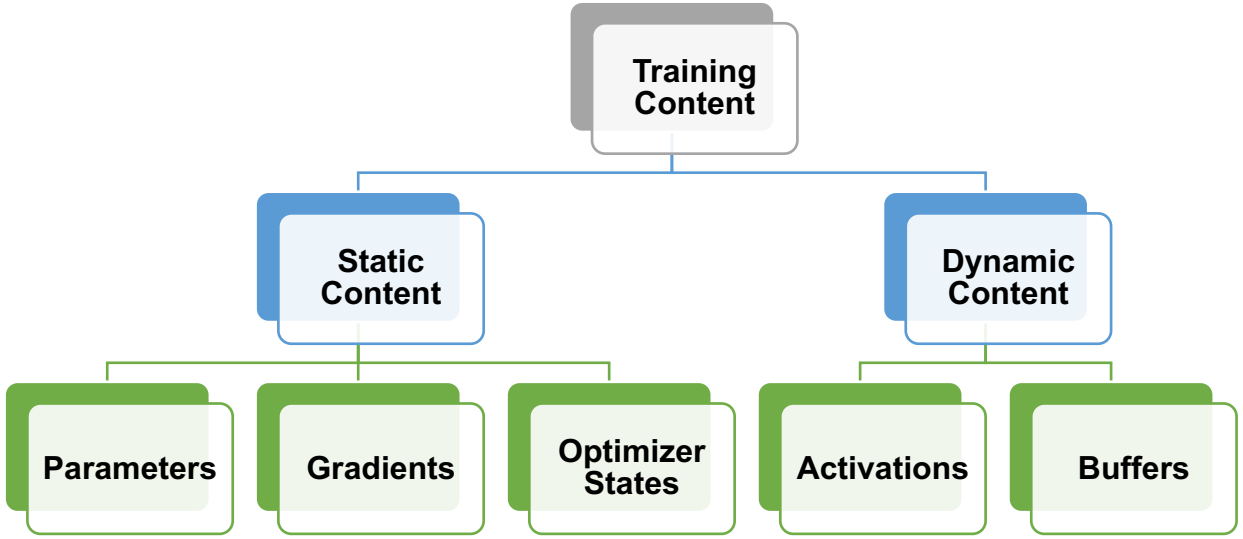


Broadcast

Intra-layer partitioning, and AllGather

LLM training: DP with Memory Optimization

- 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

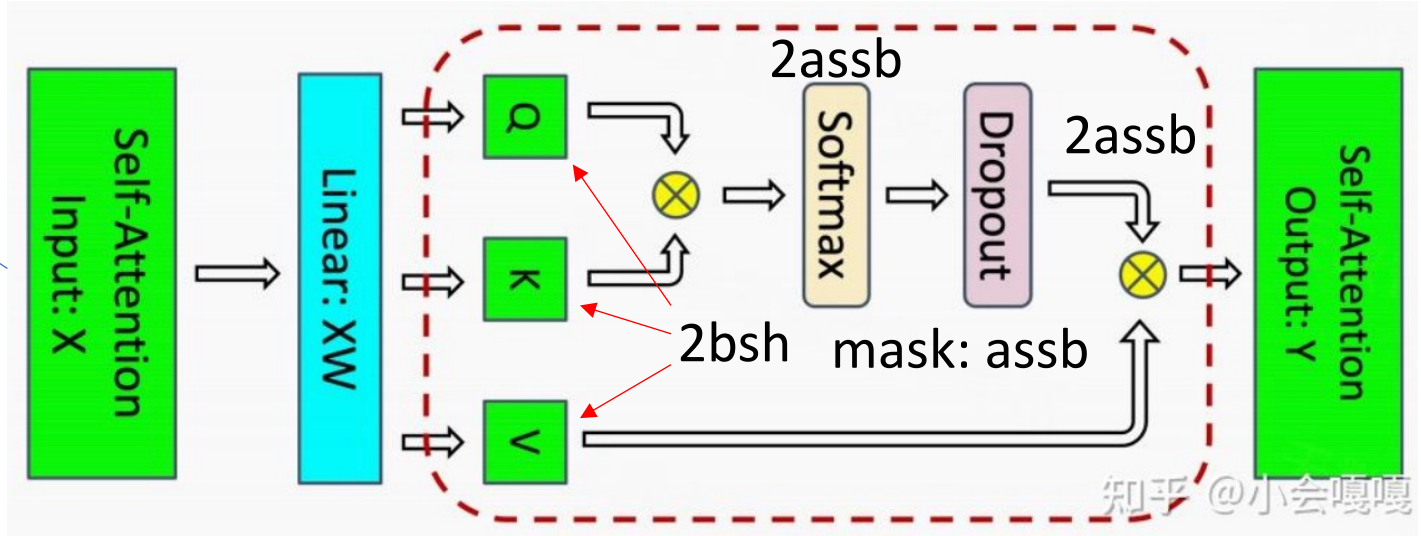
Output of a previous layer, intermediate variable

$$y = Wx + b$$

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

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

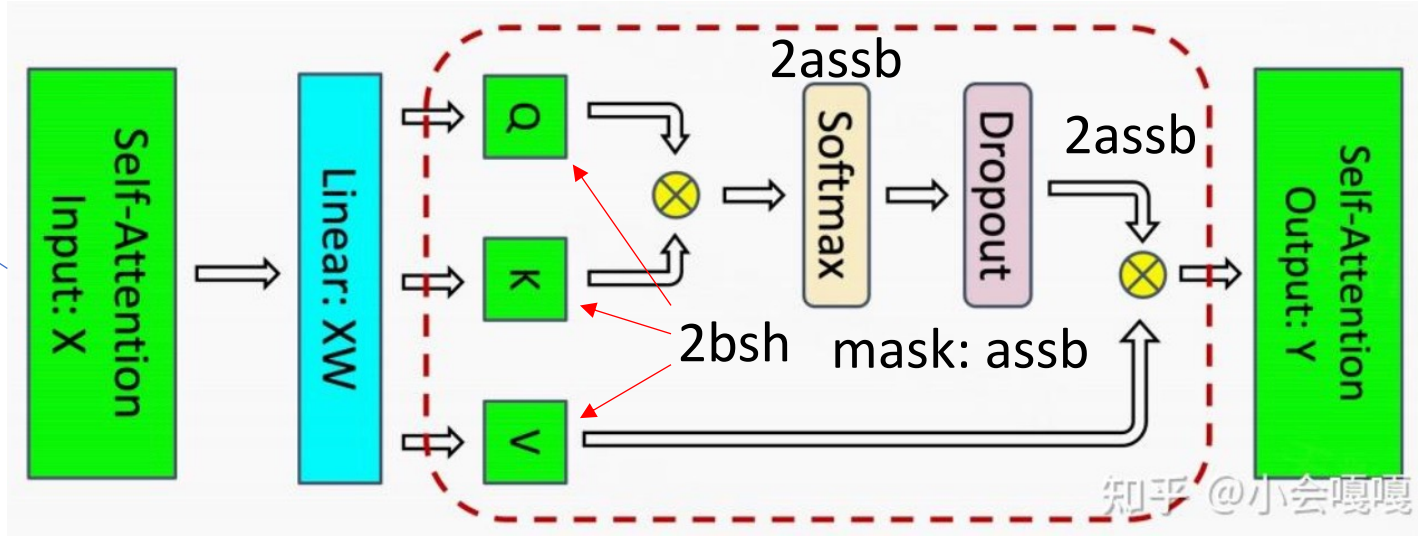
Output of a previous layer, intermediate variable

$$y = Wx + b$$

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

Cannot be discarded after FP

bsh * 2 Bytes

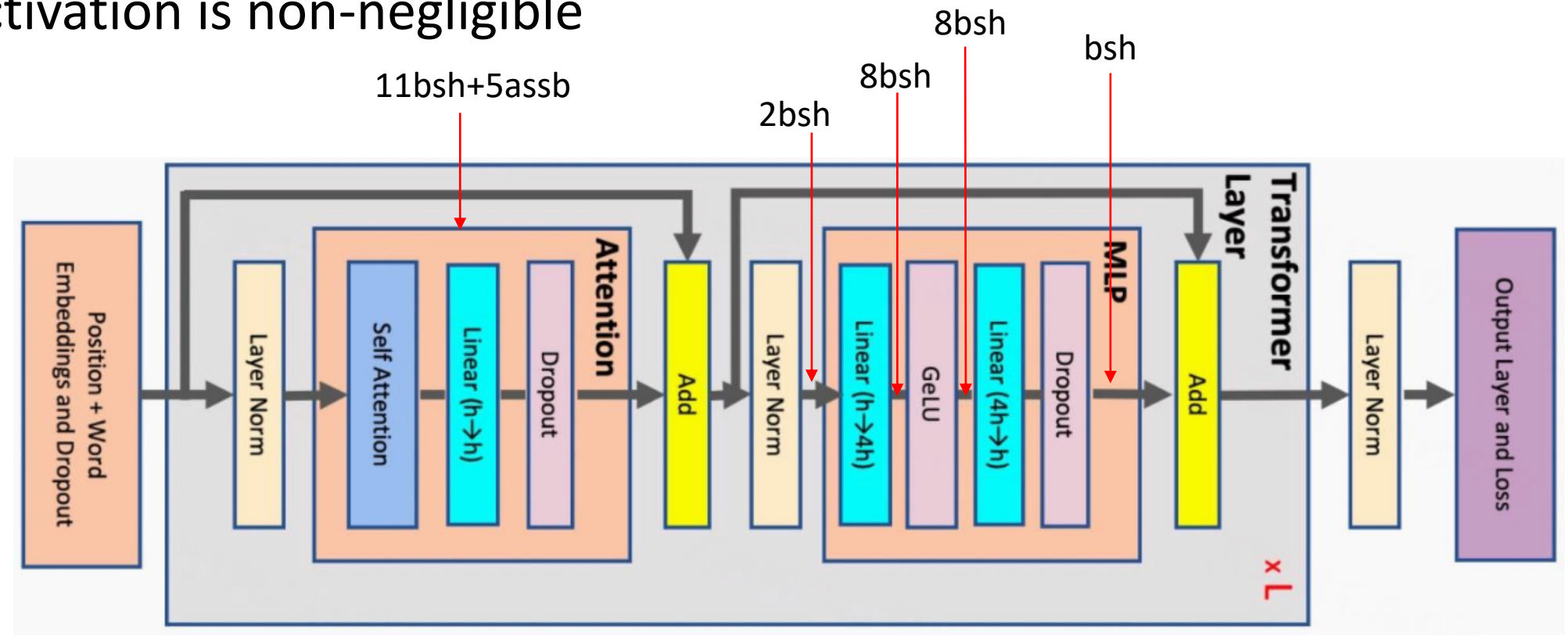


3sbh: Linear layer + dropout layer

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

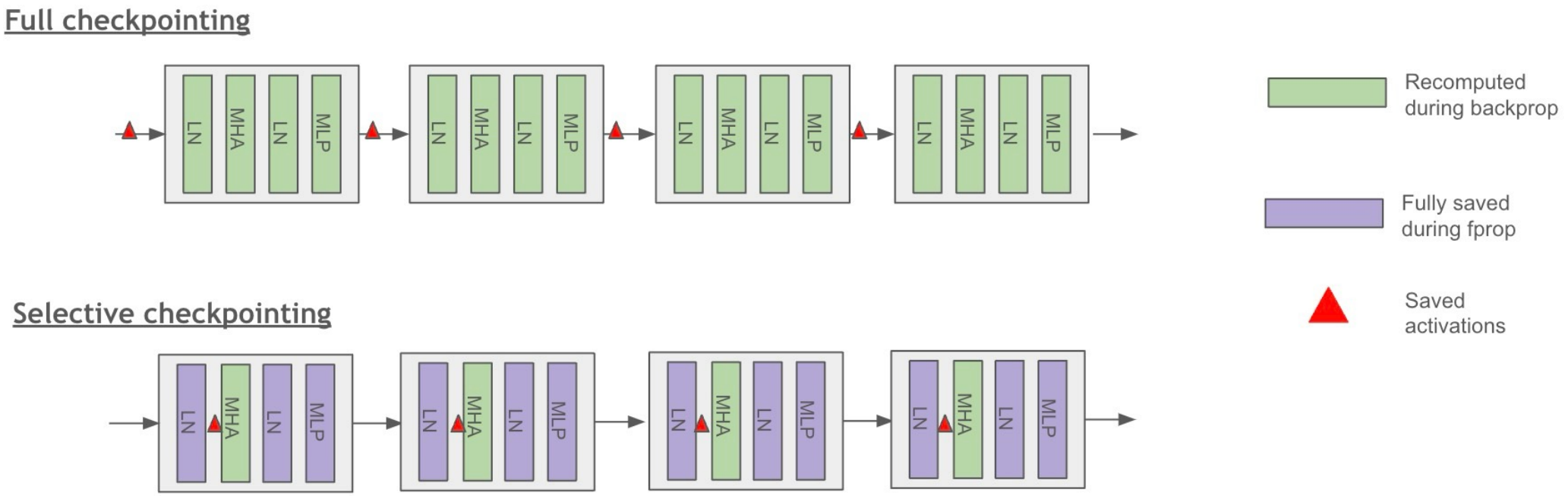
- Activation is non-negligible
  - b: 3.2M tokens/sequence length, s: 2048 tokens, h: 12288, l: 96 layers
  - Activation size ?
  - Full activation re-computation: only keeping the initial input and recompute everything
    - Minimal memory occupation
    - Prolonged training time (doing forward propagation 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*

Thanks!