Model Parallelism in LLM Training

Spring 2026

Lecturer: Yuedong (Steven) Xu

Fudan University

ydxu@fudan.edu.cn

Disclaimer

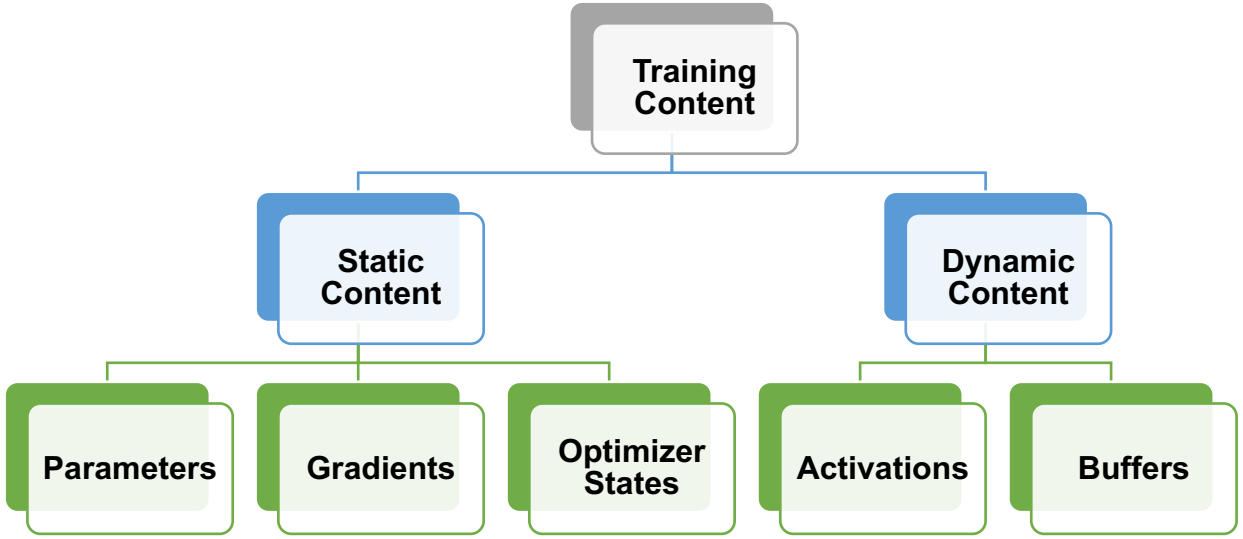
Machine learning systems is a broad and rapidly evolving field. The course material has been developed using a broad spectrum of resources, including research papers, lecture slides, blogposts, research talks, tutorial videos, and other materials shared by the research community.

Distributed LLM Training: Outline

- Data Parallelism
  - Parameter-Server
  - All-Reduce
  - Memory Optimization
- Model Parallelism
  - **Pipeline Parallelism**
  - Tensor Parallelism
  - Sequence Parallelism
- Mixture of Experts

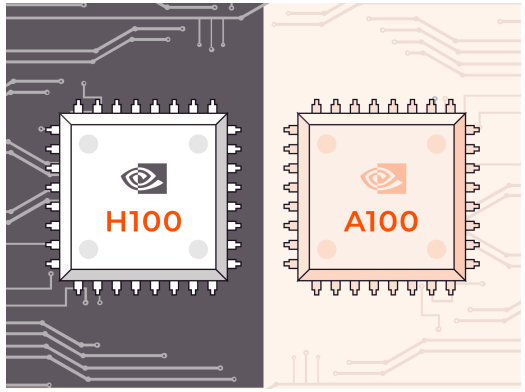
Pipeline Training

- Retrospect: GPU HBM Content During Training

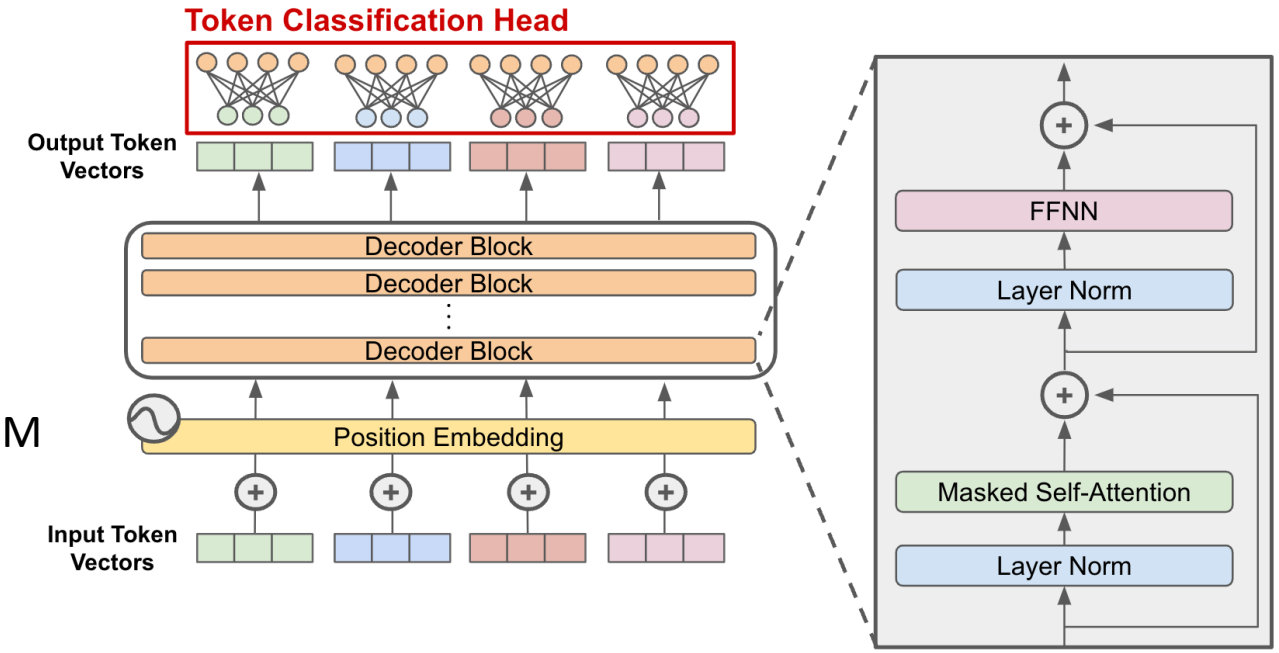


- An example of GPT-3 175B
  - **Optimizer States**
    - 32-bit Parameter (700 GB)
    - Adam Moment (700 GB)
    - Adam Variance (700 GB)
  - **16-bit Parameter (350 GB)**
  - **16-bit Gradient (350 GB)**
  - **Activations (depending on batch size)**
  - Buffer and Fragmentation

Pipeline Training



Insufficient HBM

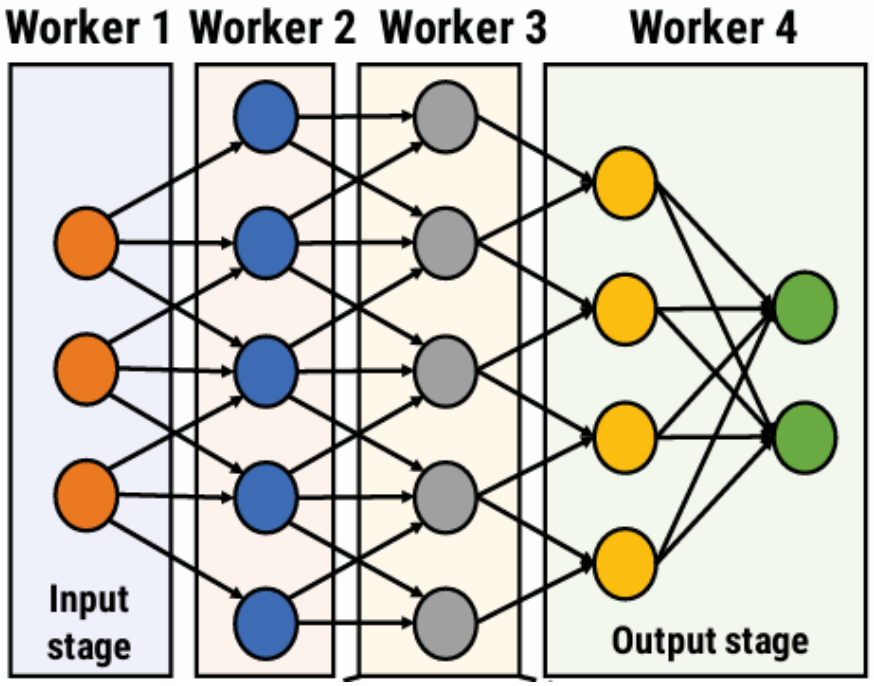


When LLM model is too LARGE!

Pipeline Training

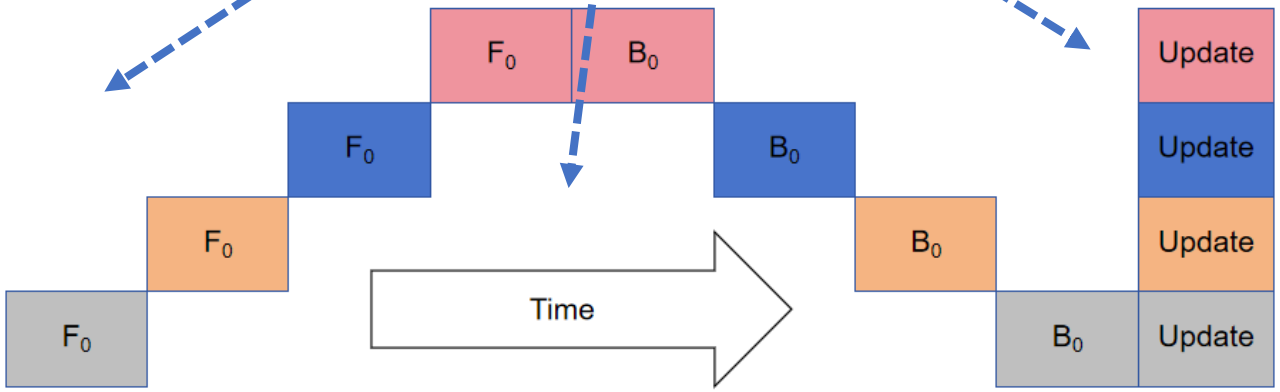
- Letting data parallelism be SUFFICIANTLY large
  - Linearly increasing network latency (αn)
  - Inefficient small data chunk transmission
  - Straggler effect caused by any communication pair

Pipeline Training



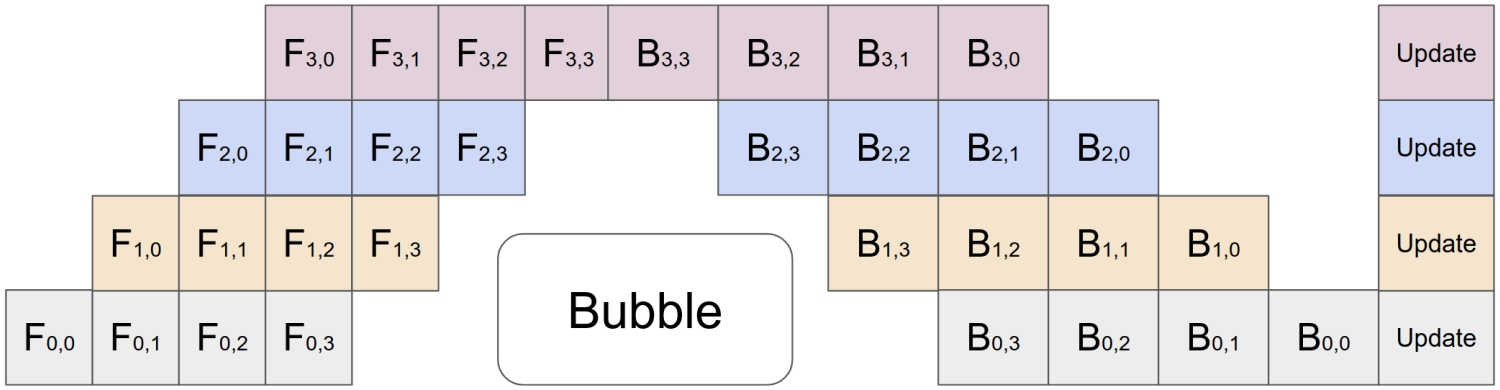
Layerwise partitioning on multiple GPUs

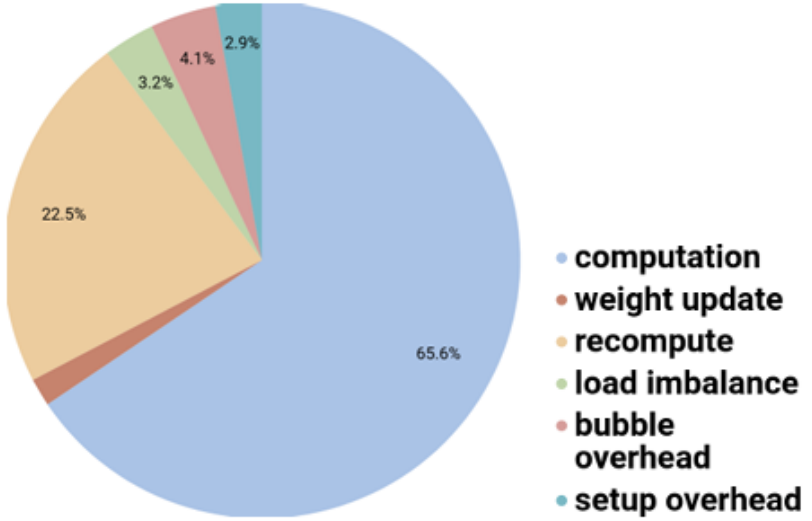
Large bubble $O(\frac{K-1}{K})$ → underutilized GPU



Naïve model parallelism
(F for forward； B for backward)

Pipeline Training





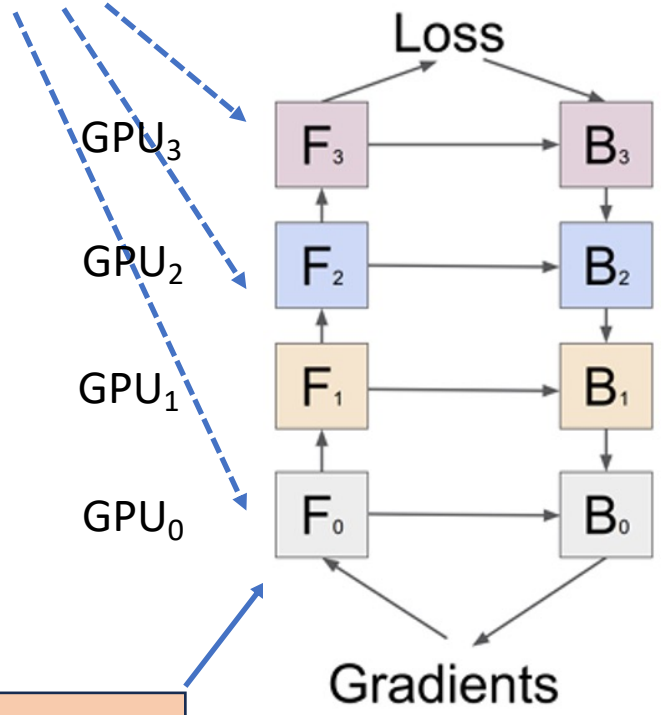
b: batch size
b_m : microbatch size
K: # pipeline stages
M: # microbatches, b = b_m *M

Bubble ratio: $O(\frac{K-1}{K+M-1})$

《GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism》

Pipeline Training

Cannot be discarded before BP



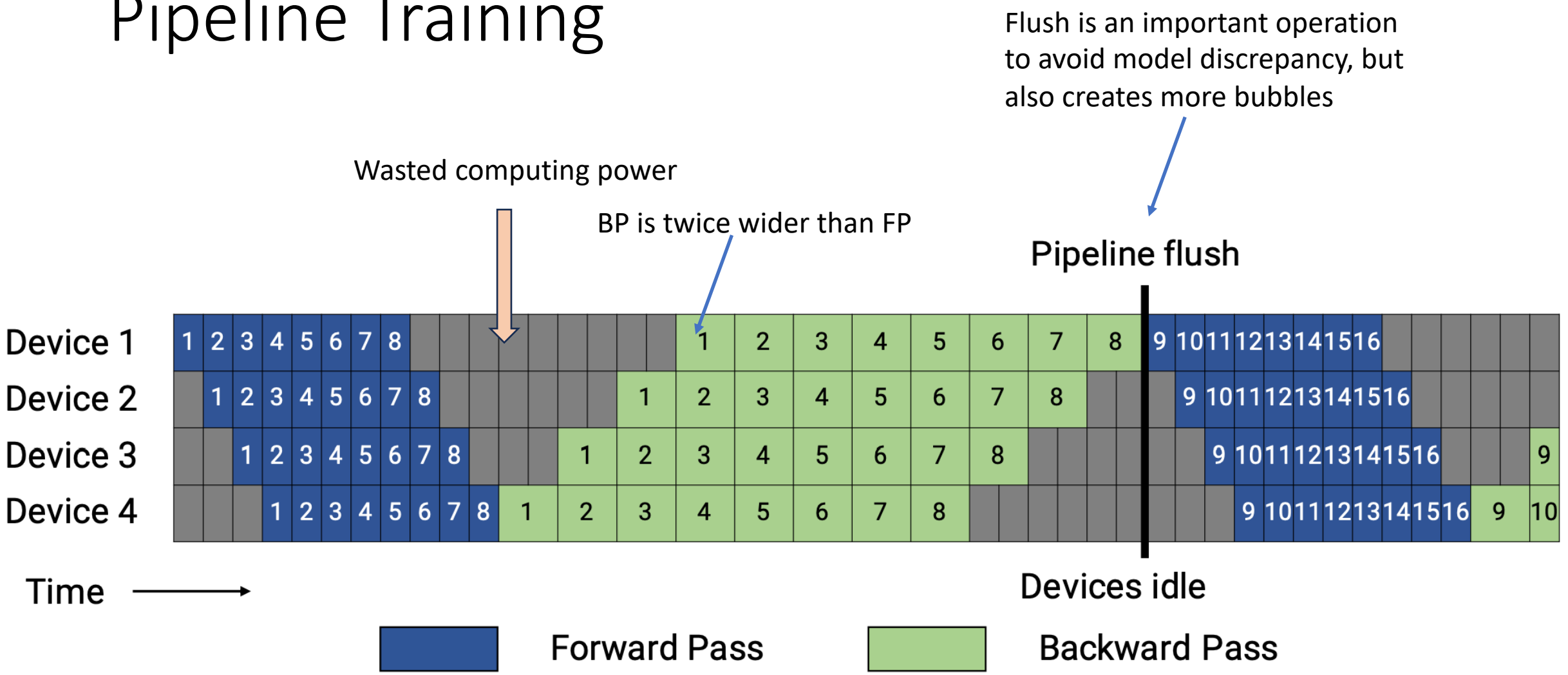
Keep F_0

- Computation time
  - Forward time is half of backward time (why?)
- Communication load
  - Intermediate activation between two partitions
- Memory footprint
  - First stage usually needs more memory: $K * M$
  - Re-materialization (re-computation) Further reducing memory consumption

F-then-B: Forward then Backward

《GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism》

Pipeline Training



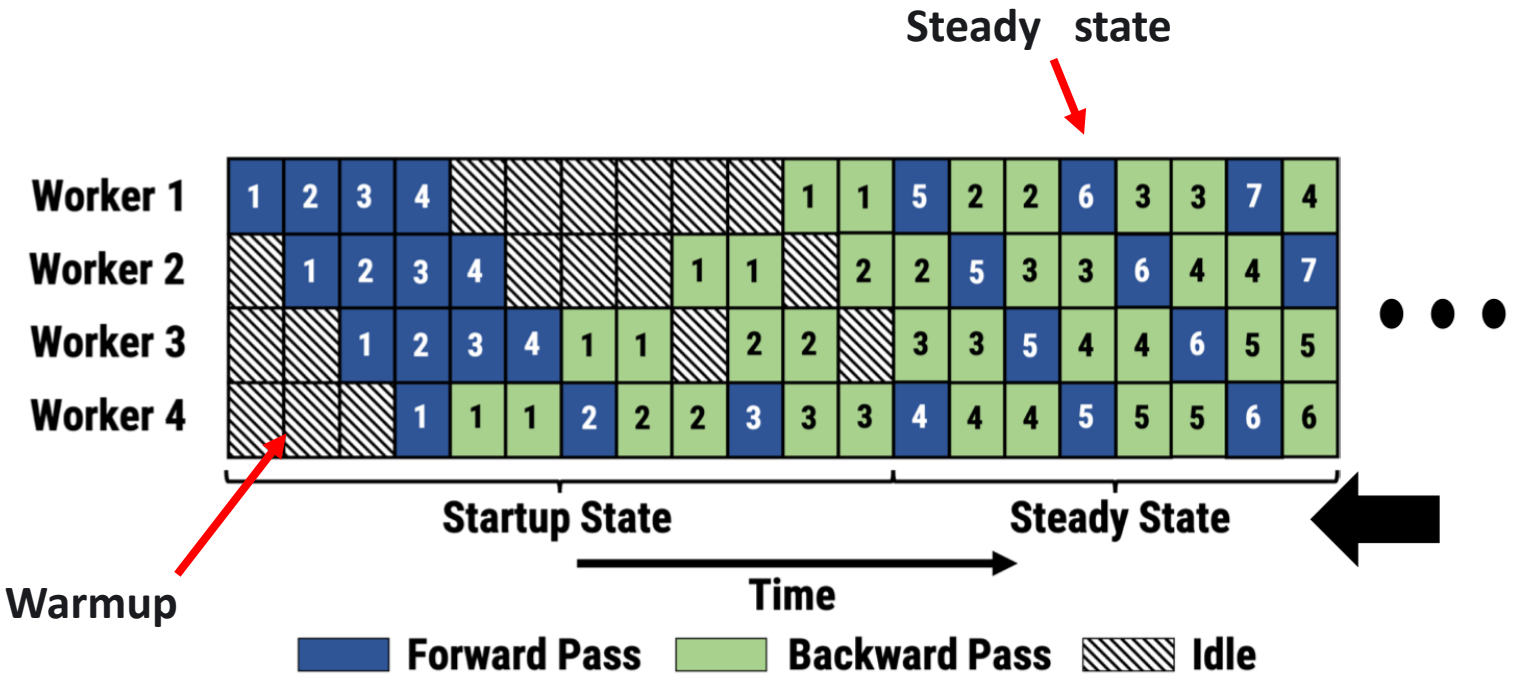
Frequent pipeline flushes lead to increased idle time

Pipeline Training

Orchestrating execution orders of forward and backward passes for microbatch so as to reduce bubbles and peak memory footprints?

- Some KEY concepts
  - Synchronous pipeline versus asynchronous pipeline
  - Interleaved versus non-interleaved

Pipeline Training

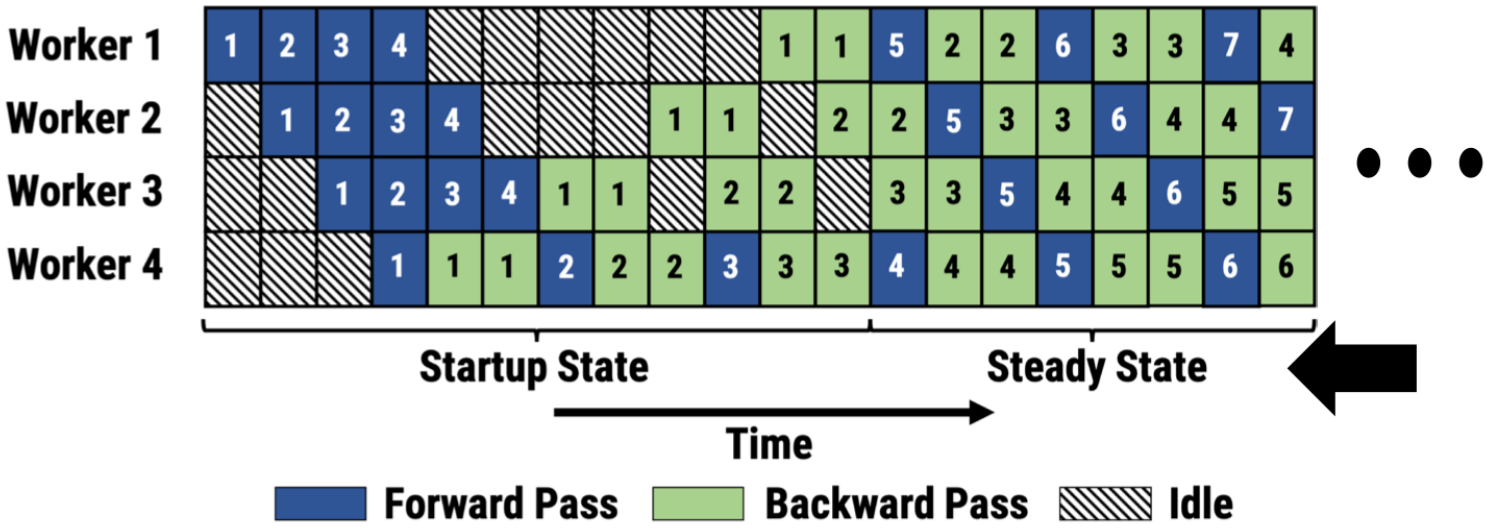


1F1B: One Forward then One Backward

- Minimizing lingering time of activations
  - Immediately executing BP after FP for the first microbatch at the output stage
- At steady state
  - Strictly alternative Forward and Backward operations
- Asynchronous weights

《PipeDream: generalized pipeline parallelism for DNN training》

Pipeline Training



1F1B: One Forward then One Backward

- Challenges
  - How to split a DNN model into different stages?

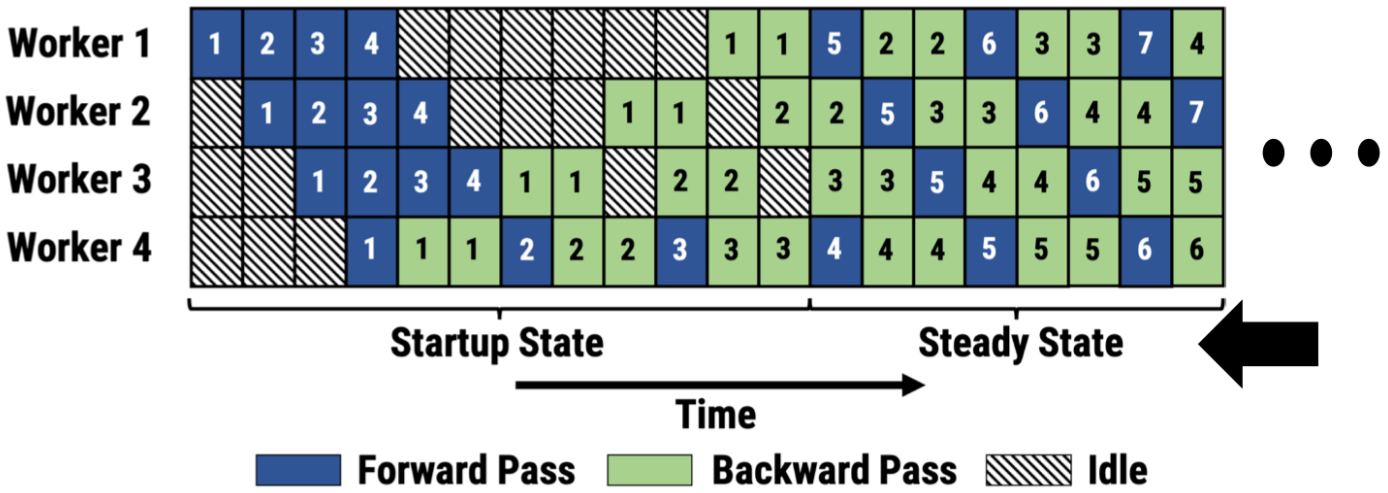
    LOAD BALANCING *across stages*

  - How to update parameters?

    After backward pass, the gradient can be used ***immediately to update the parameter***

《PipeDream: generalized pipeline parallelism for DNN training》

Pipeline Training

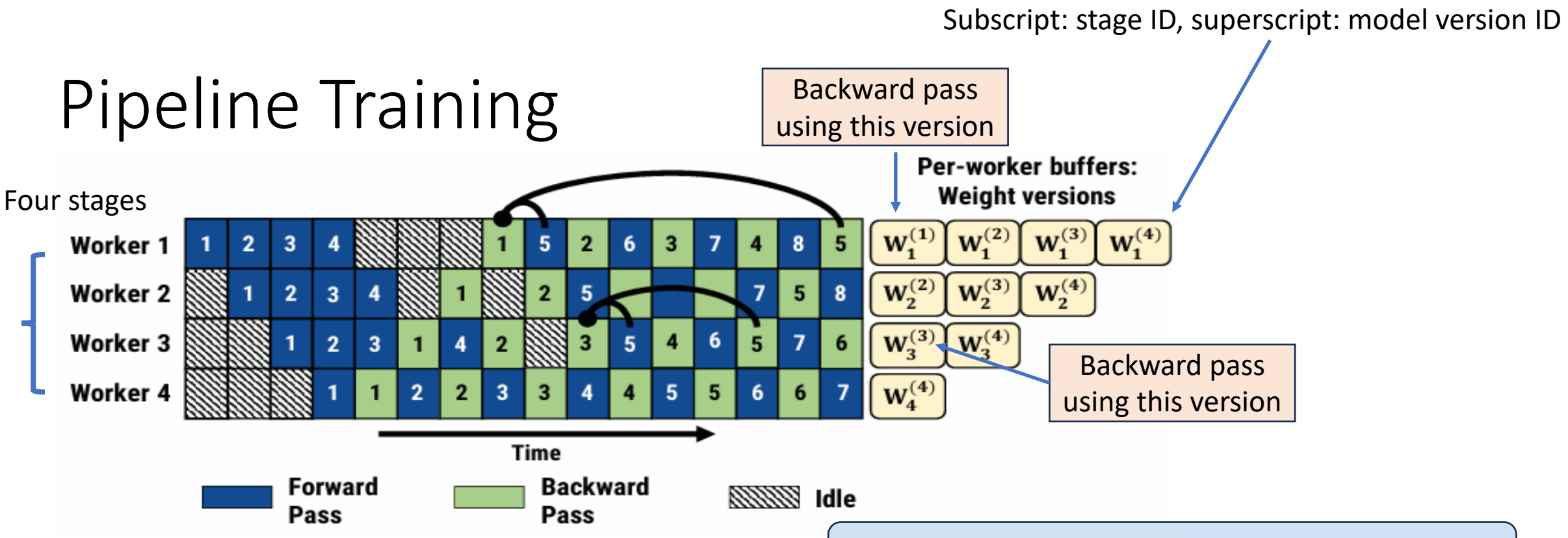


- Observations in stage 1
  - The forward pass of microbatch 5 is performed after the update of **microbatch 1**
  - The backward pass of microbatch 5 is performed after the updates of **microbatches 2, 3 and 4**

Forward and backward passes operate on different versions of parameters!

《PipeDream: generalized pipeline parallelism for DNN training》

Pipeline Training

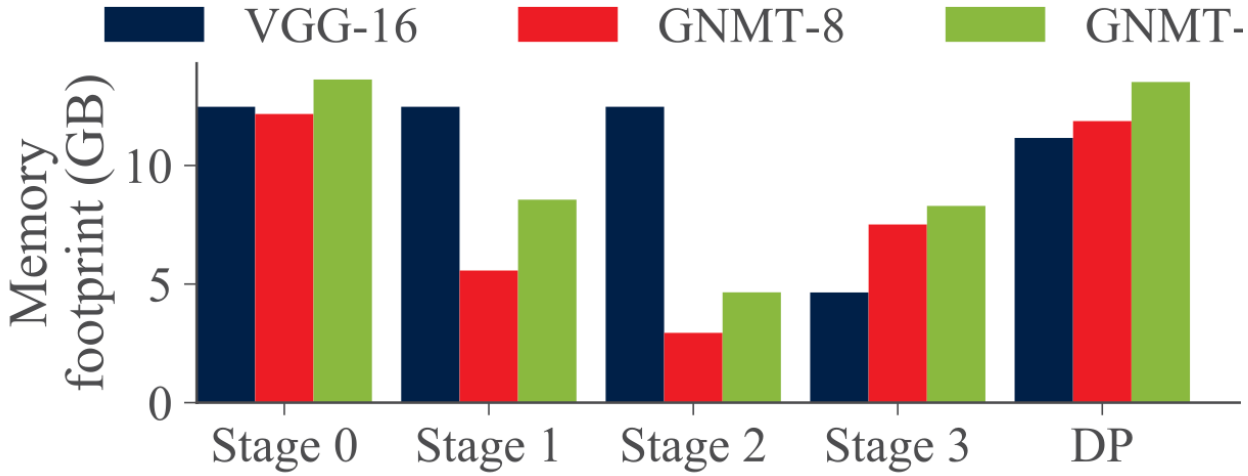


No guarantees across stages!

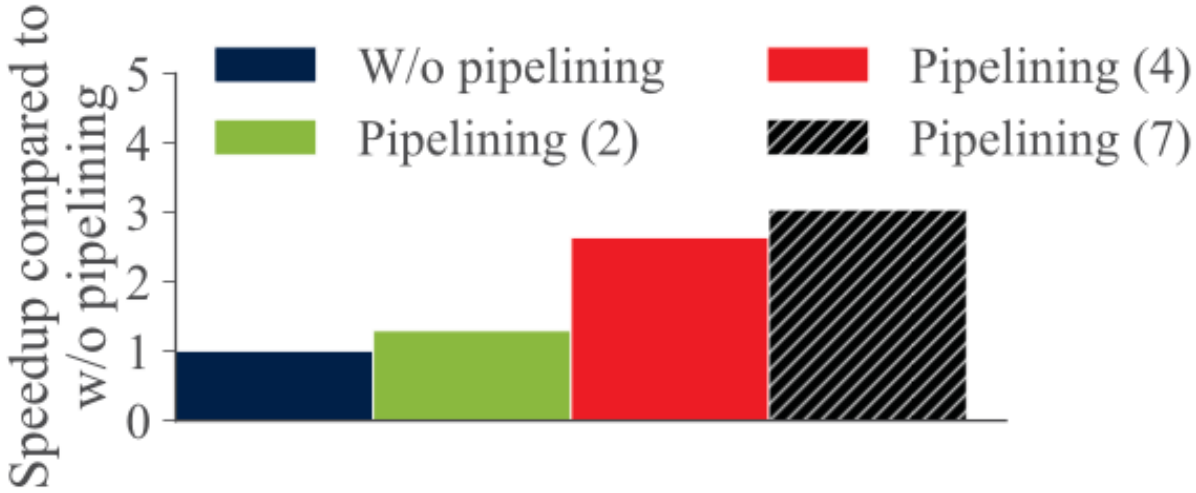
- ***Weight stashing*** of PipeDream
  - Keeps multiple versions of model weights—one for each active minibatch.
  - Forward pass: each pipeline stage uses the most recent weight version available. Once the forward pass for a minibatch is completed, the corresponding weight version is stored.
  - Backward pass: the same version is reused to compute gradients and update weights, ensuring that both passes for a given minibatch use identical parameters within a stage.

Pipeline Training

- Experiments



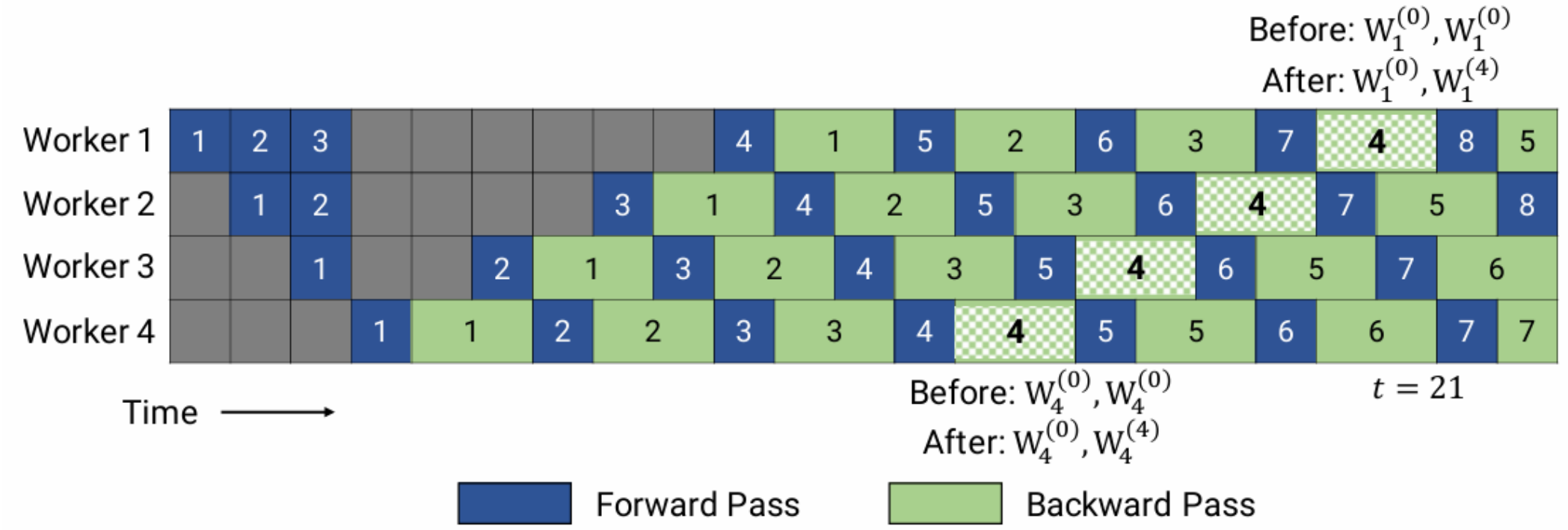
Memory footprint of PipeDream



Speedup ratios on 4 V100 GPUs

PipeDream still saves memory by trading multiple versions of parameters for large activations

Pipeline Training



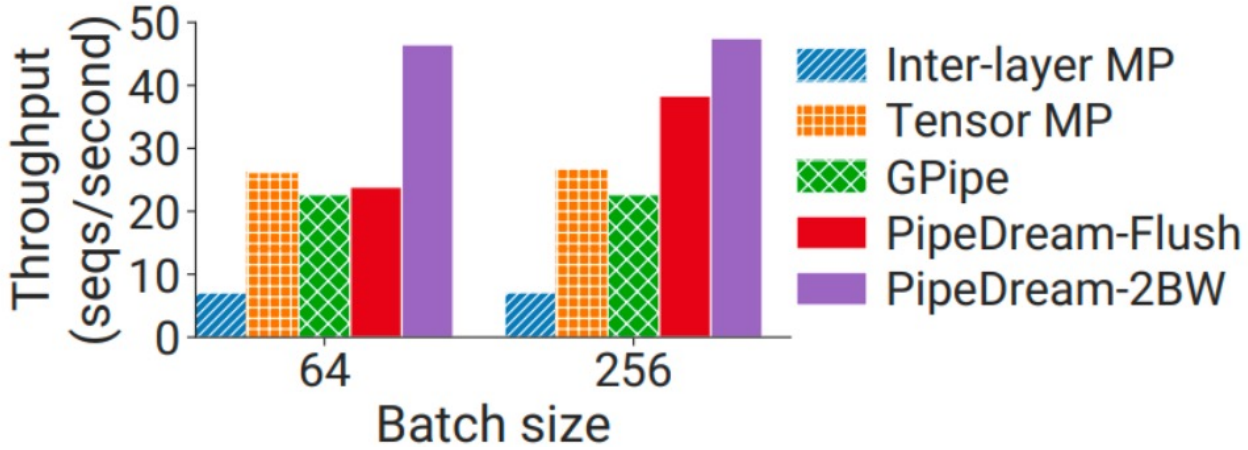
PipeDream-2BW:

- **less frequent flushing than Gpipe**
- **keeping no more than two model versions.**

Think why? How does it relate to pipeline depth K and number of microbranches M

《Memory-Efficient Pipeline-Parallel DNN Training》

Pipeline Training



- Flushing causes more bubbles
  - Warmup + Colddown
- Flushing needs less memory

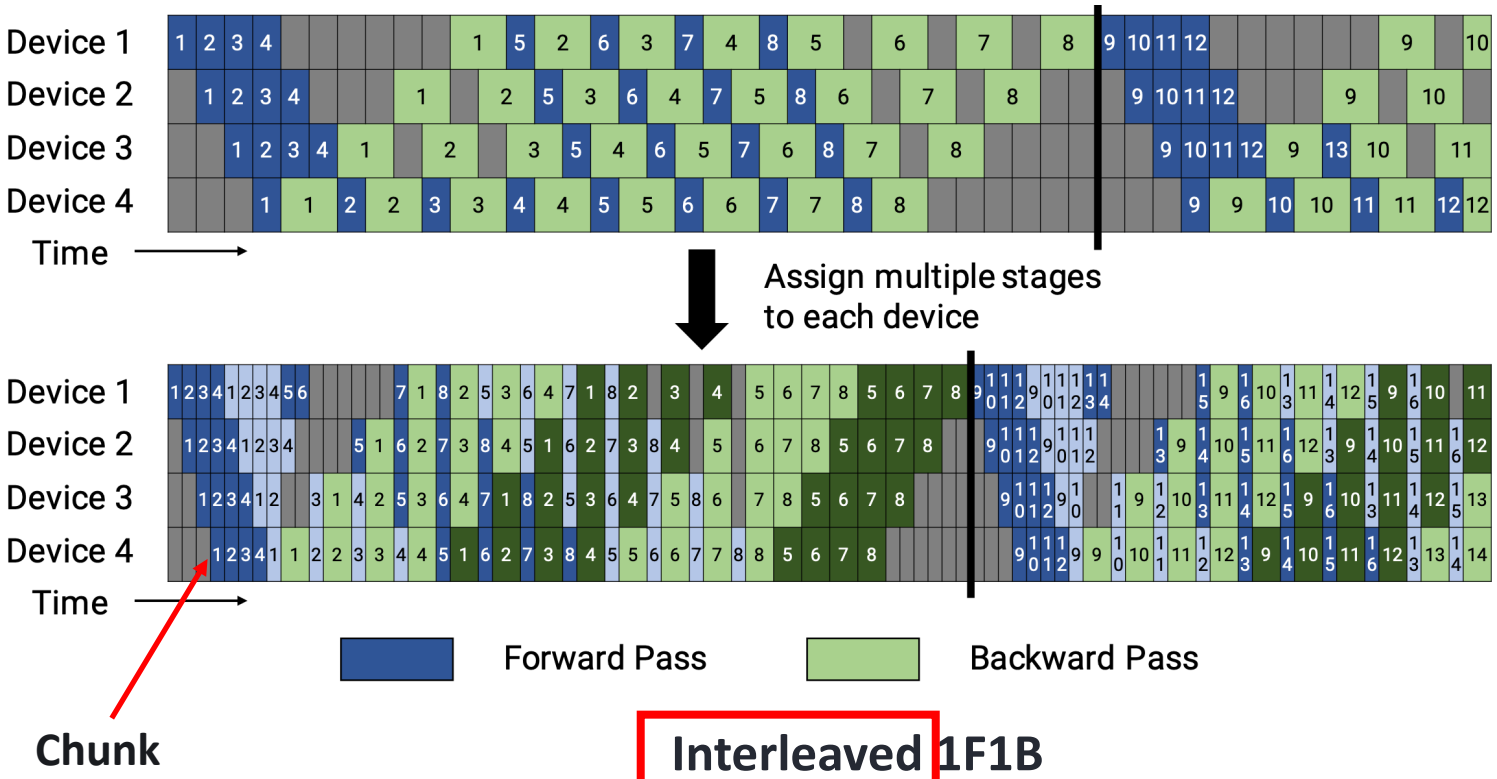
- Training BERT 2.2B with 8-way model parallelism (8 V100 GPU)
- PipeDream is much better than vannila GPipe

《Memory-Efficient Pipeline-Parallel DNN Training》

https://juejin.cn/post/7262274383287484476

Pipeline Training

- IFIB integrated in Megatron-LM later on



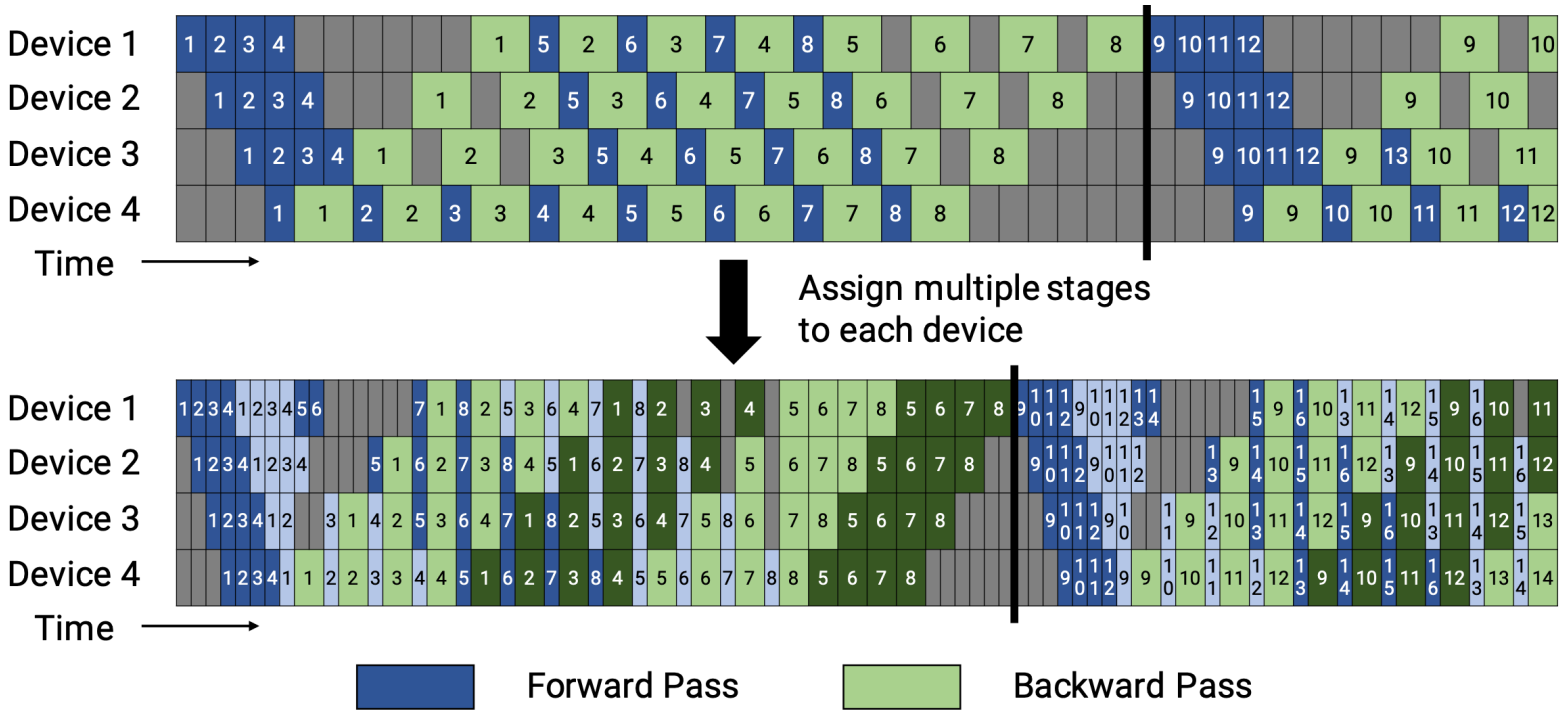
(Stages: 4, Layers: 16, each device assigned with 2 chunks. Layer 0-1, 8-9 on Device 1; Layer 2-3,10-11 on Device 2; Layer 4-5, 12-13 on Device 3 and Layer 6-7, 14-15 on Device4)

- Key ides:
  - Further dividing each microbatch into even smaller chunks
  - Microbatch and small chunks execute alternatively
- Advantages/disadvantages
  - Smaller bubble
  - Higher inter-state traffic load

《Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM》

Pipeline Training

- IFIB integrated in Megatron-LM later on

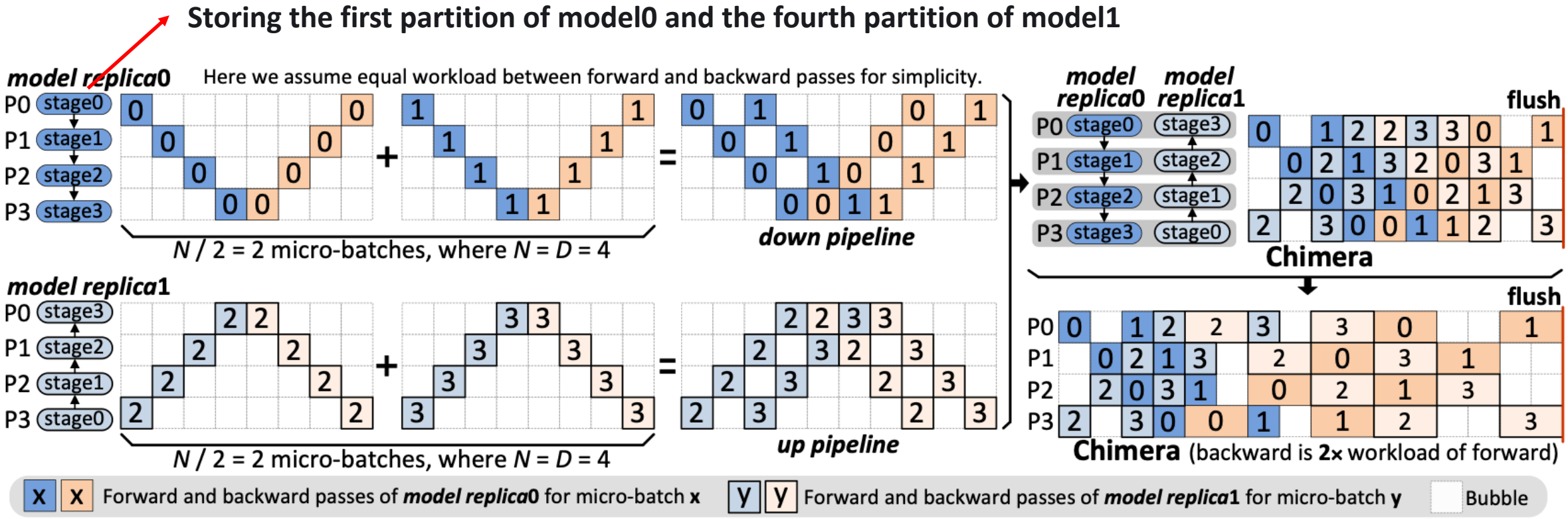


- Model placement:
  - Stages: 4, Layers: 16, each device assigned with 2 chunks
  - Layer 0-1, 8-9 on Device 1
  - Layer 2-3,10-11 on Device 2
  - Layer 4-5, 12-13 on Device 3
  - Layer 6-7, 14-15 on Device 4

Be aware of the communication scheduling!

《Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM》

Pipeline Training



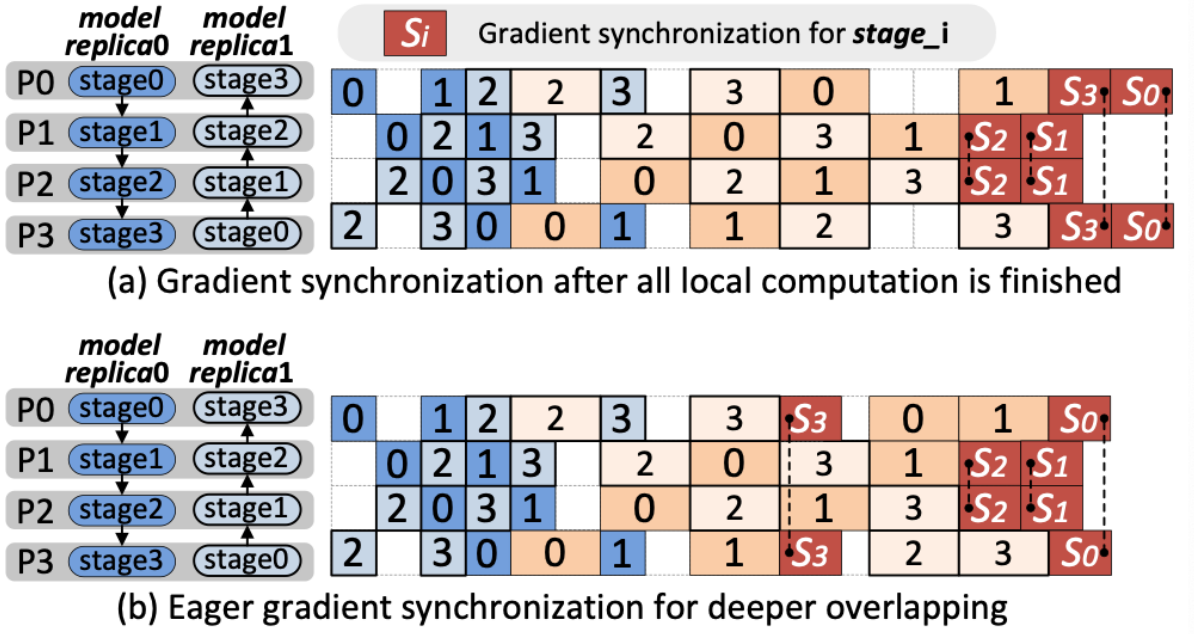
Chimera: double pipeline

- Smaller bubbles yet doubled memory occupation for model replicas and FP/BP activations

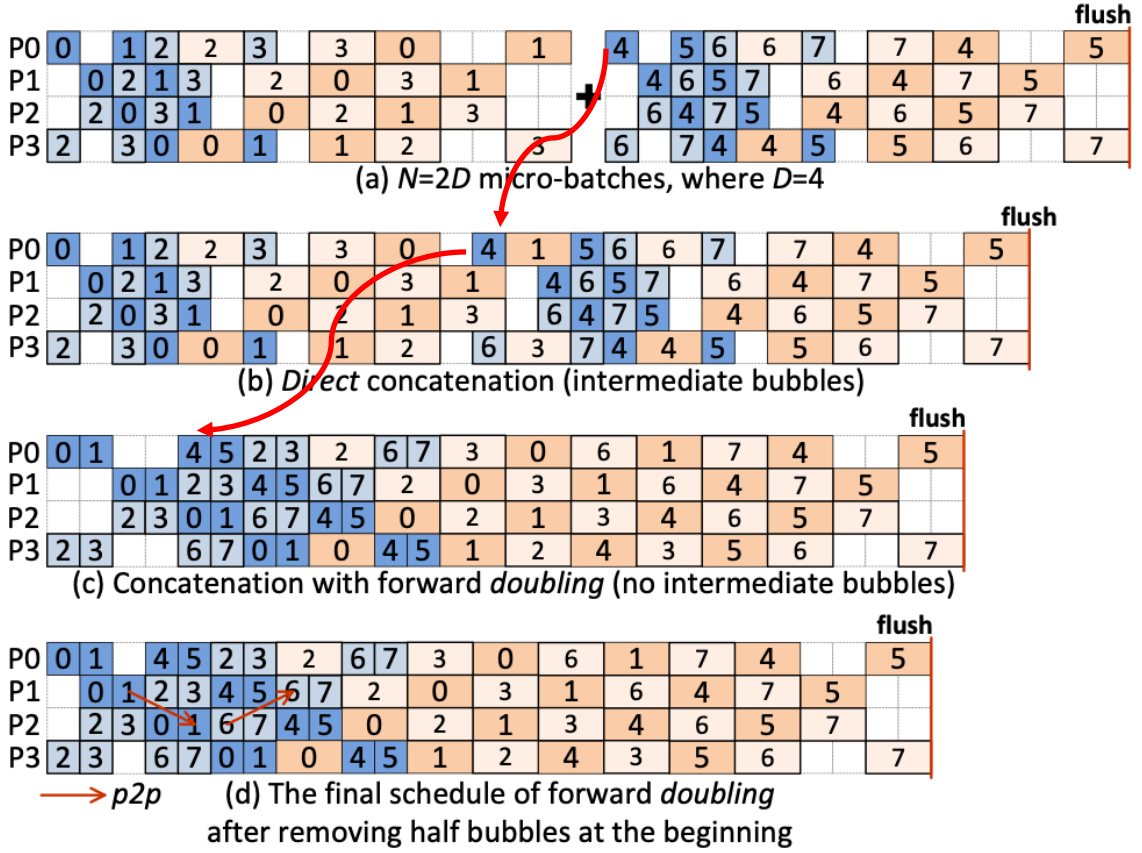
《Chimera: Efficiently Training Large-Scale Neural Networks with Bidirectional Pipelines》

Pipeline Training

- Orchestrating bubble mitigation and data-parallel synchronization



Hide the gradient synchronization overhead by overlapping them with computations, e.g. shifting S_3 to an empty slot in the left.

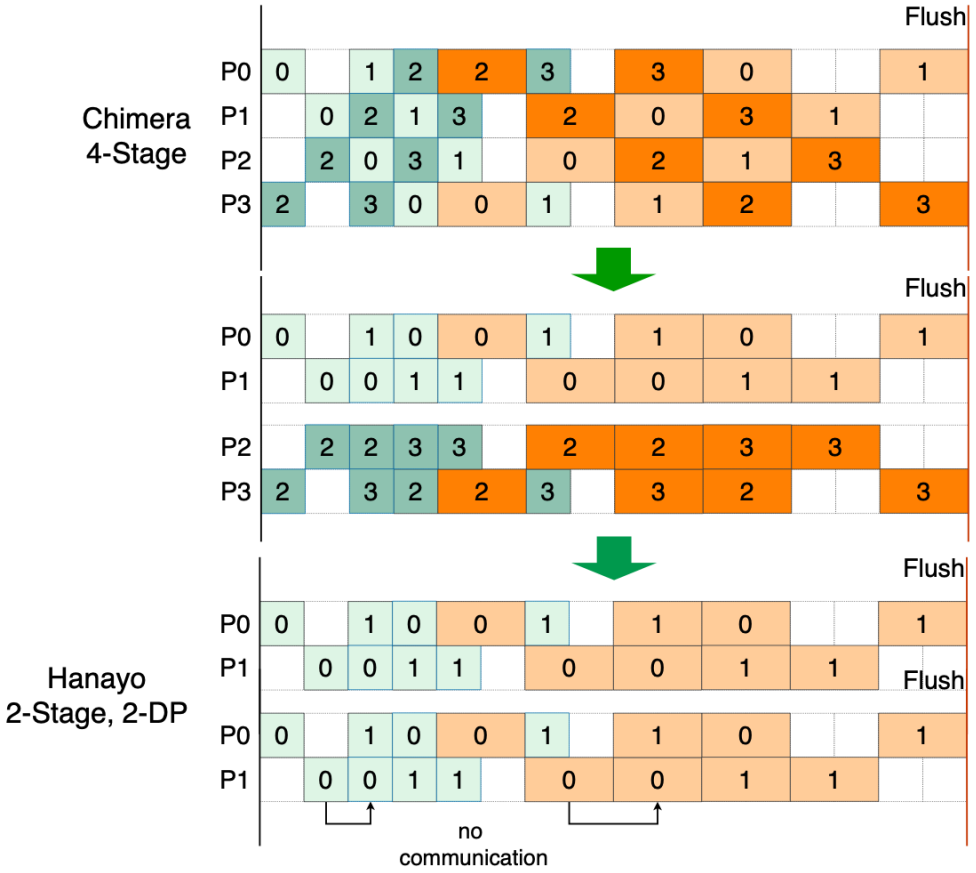


Select # replicas, # stages and Assemble blocks

《Chimera: Efficiently Training Large-Scale Neural Networks with Bidirectional Pipelines》

Pipeline Training

- Linking Chimera with Hanavo



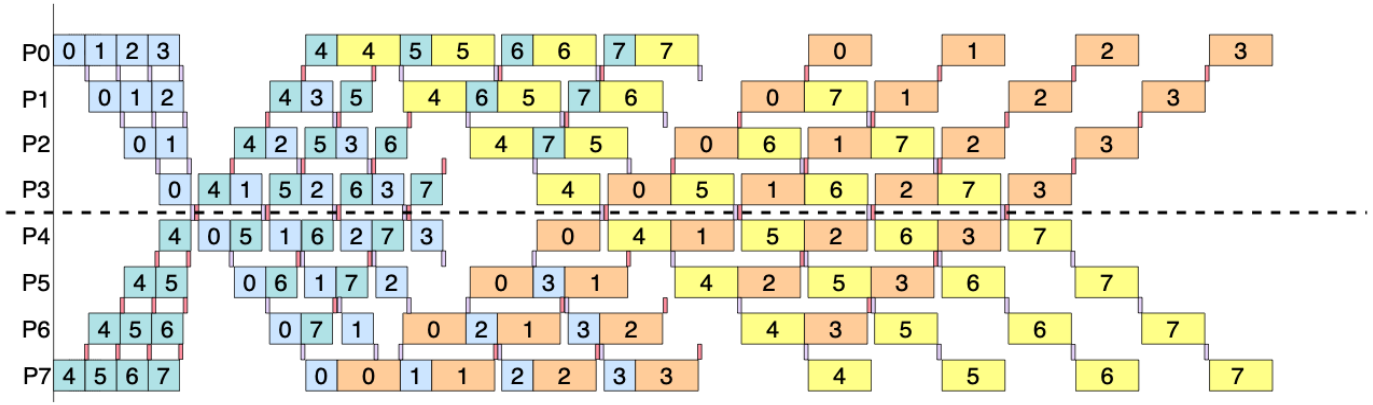
- Forward propagation marked green and Backward propagation marked orange.
- Back propagation illustrated twice as long as forward propagation.
- Purple and pink blocks to represent P2P communication that goes downward and upward.
- Numbers on the blocks show which of the micro-batches is being processed.

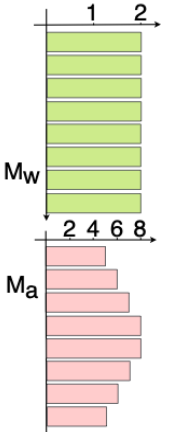
《Hanayo: Harnessing Wave-like Pipeline Parallelism for Enhanced Large Model Training Efficiency》

Pipeline Training

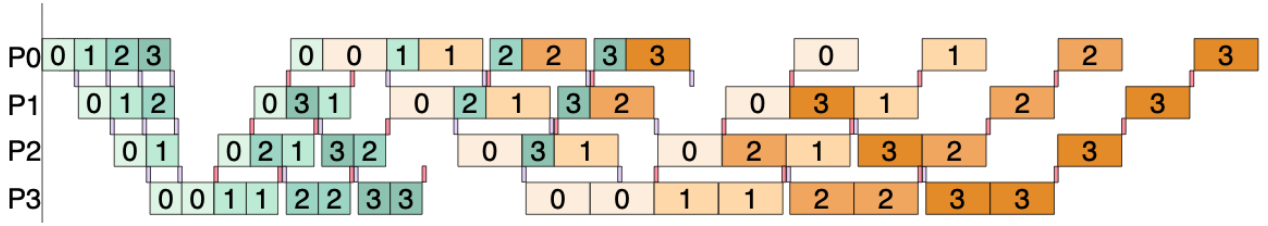
- Linking Chimera with Hanavo

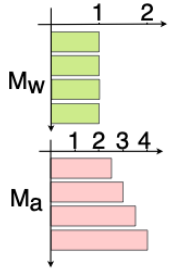
Chimera



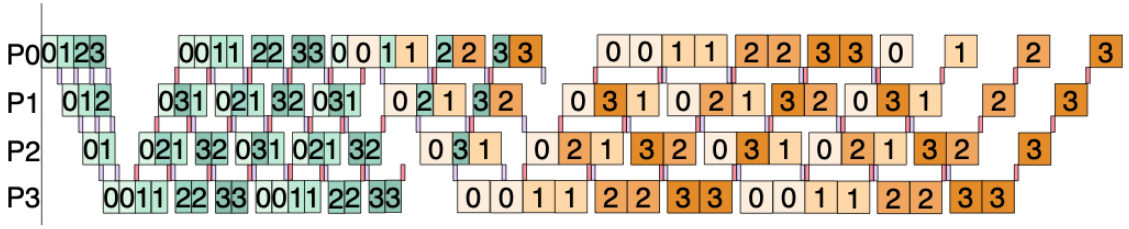


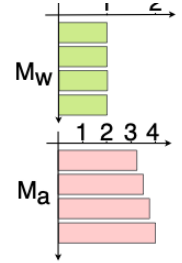
Hanavo with one wave (after folding)





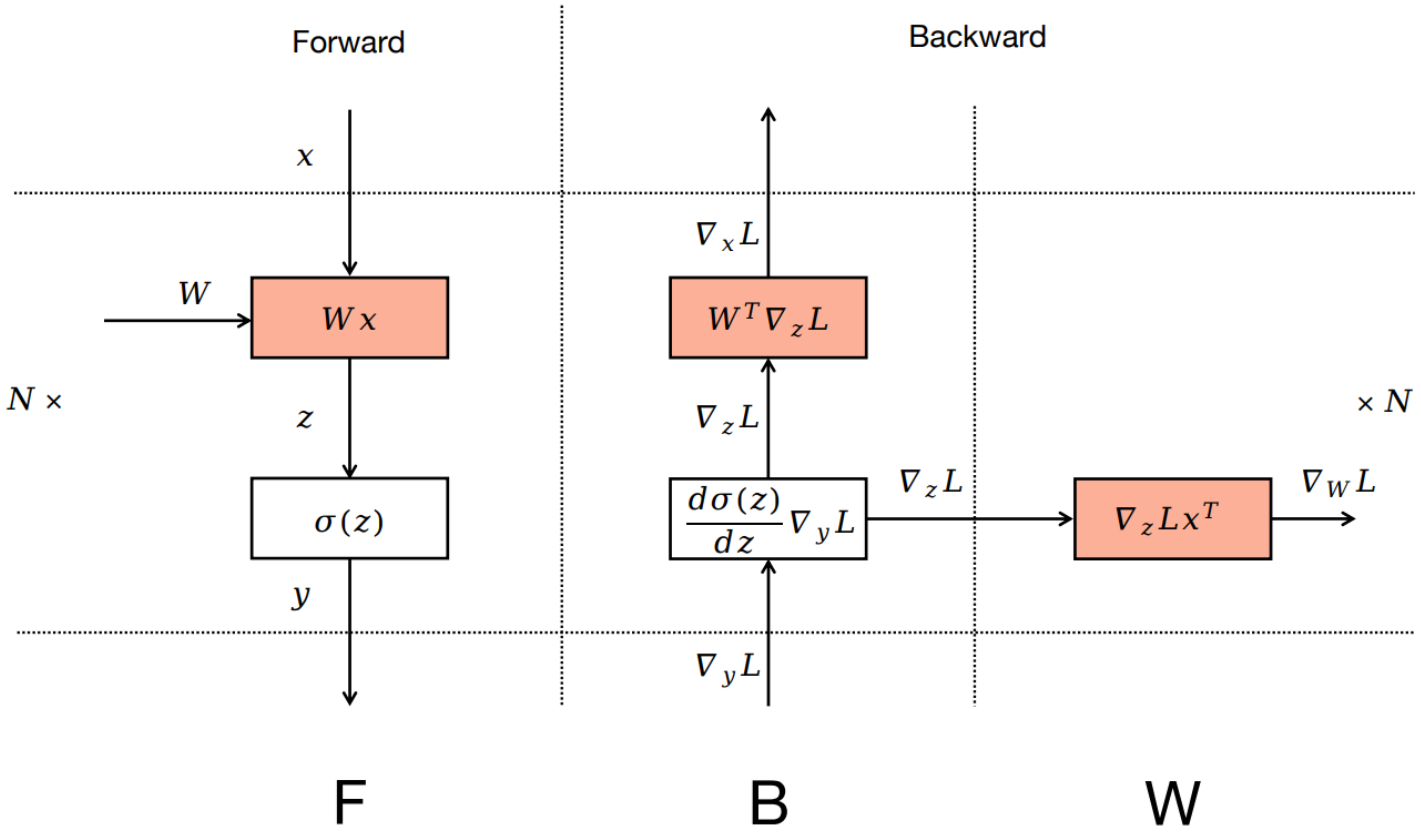
Hanavo with two waves





Pipeline Training

- A microscopic view of BP

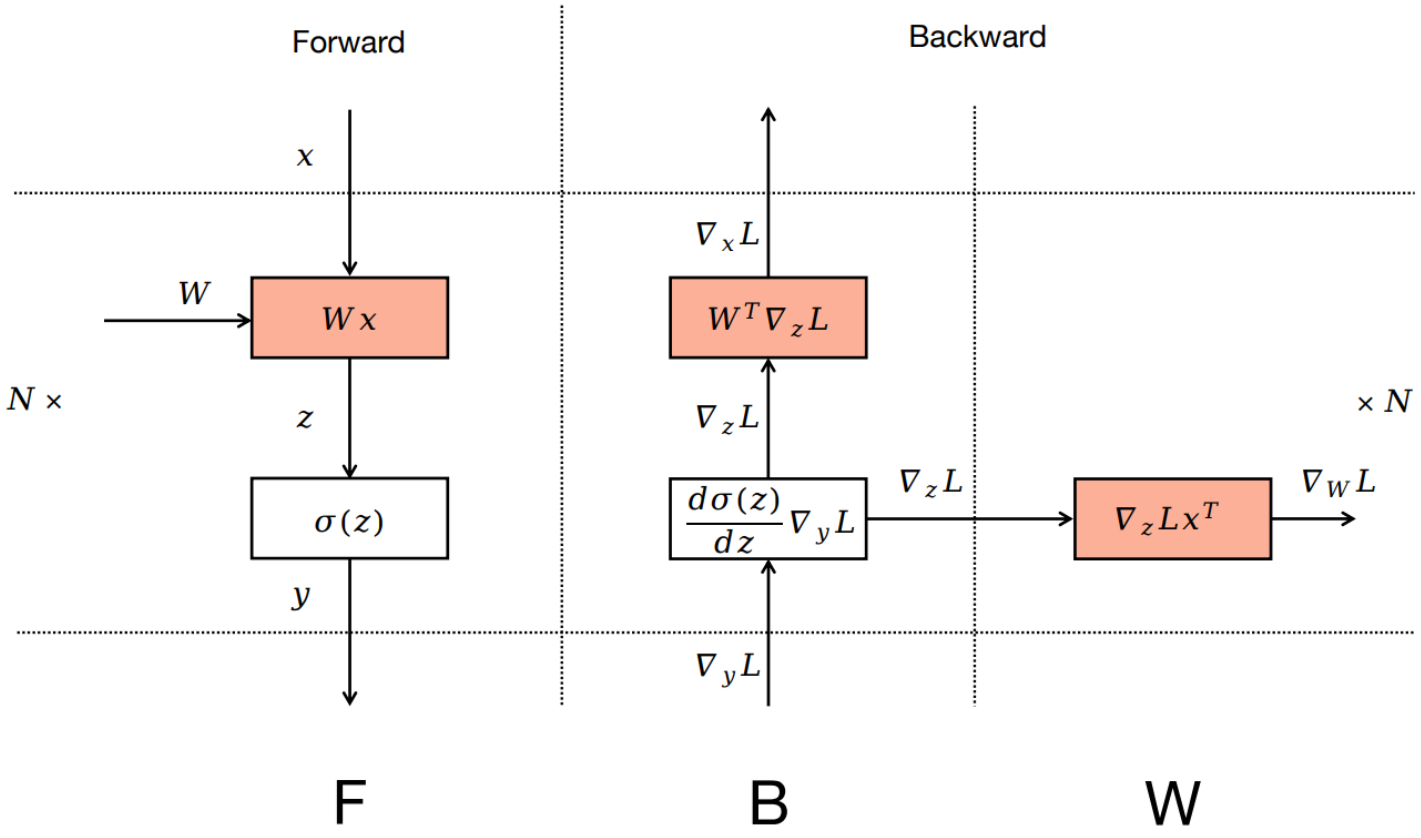


- Derivative over input
  - for backward propagation
- Derivative over weight
  - for updating model directly
- $\nabla_{\boldsymbol{W}} f(\boldsymbol{x}, \boldsymbol{W})^\top \frac{d\ell}{d\boldsymbol{y}}$ computed immediately after $\nabla_{\boldsymbol{x}} f(\boldsymbol{x}, \boldsymbol{W})^\top \frac{d\ell}{d\boldsymbol{y}}$

《Zero Bubble Pipeline Parallelism》

Pipeline Training

- A microscopic view of BP



- Question
  - Which one should be computed first in the pipeline?

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}, \boldsymbol{W})^\top \frac{d\ell}{d\boldsymbol{y}}$$

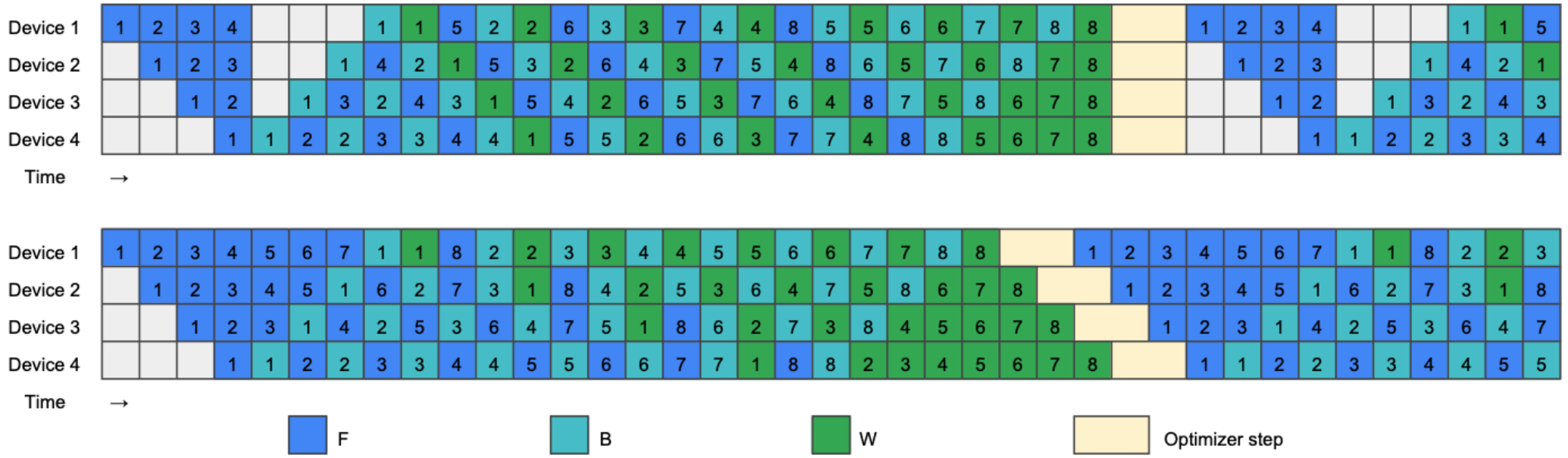
or

$$\nabla_{\boldsymbol{W}} f(\boldsymbol{x}, \boldsymbol{W})^\top \frac{d\ell}{d\boldsymbol{y}}$$

《Zero Bubble Pipeline Parallelism》

Pipeline Training

- Dividing BP into BP for input (urgent) and BP for weight (not urgent)

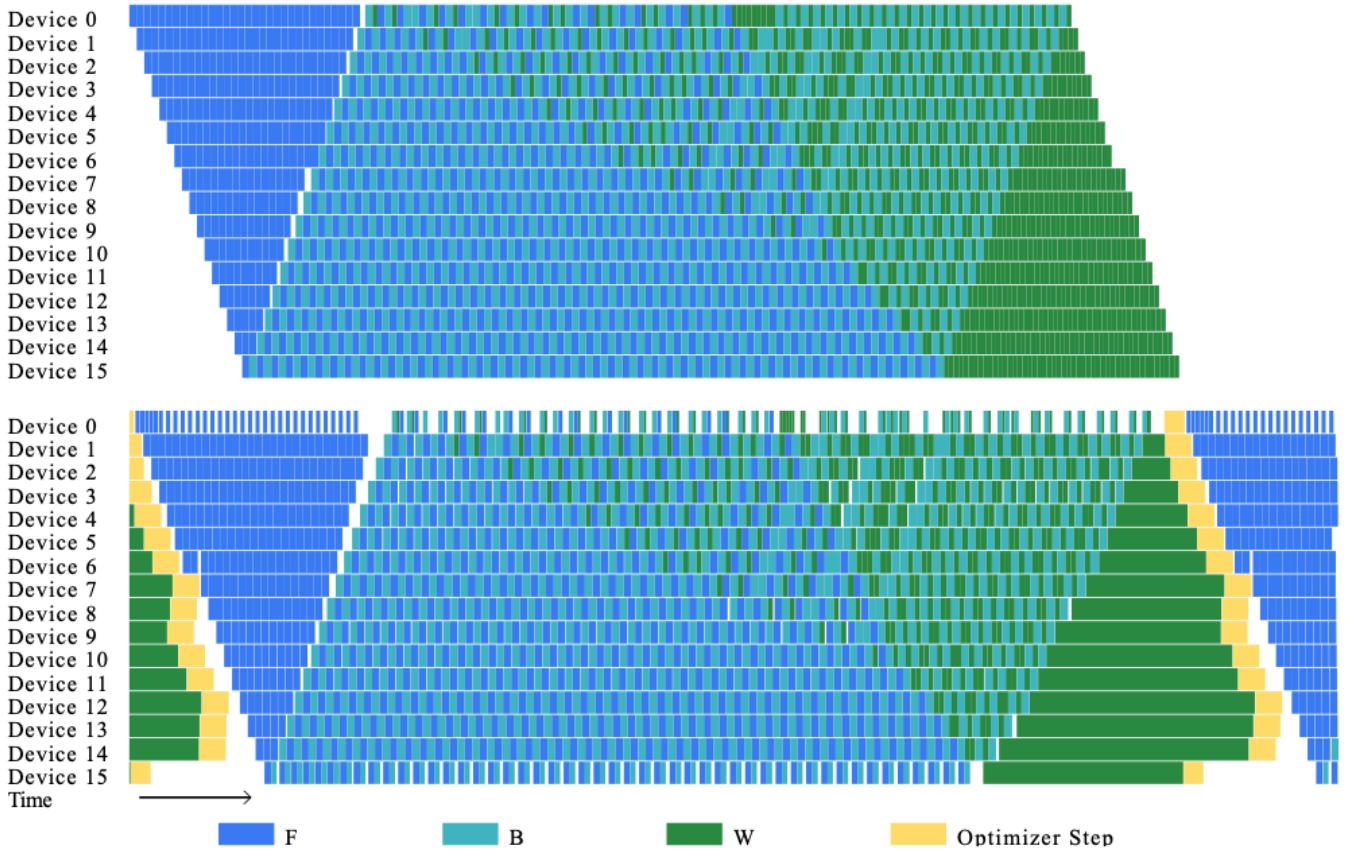


- Upper (ZB-H1): strictly follow 1F1B, and fill W to ensure all workers maintain the same number of in-flight microbatches.
- Lower (ZB-H2): theoretically zero bubble as # of microbatches is sufficiently large; large memory occupation over ZB-H1.

《Zero Bubble Pipeline Parallelism》

Pipeline Training

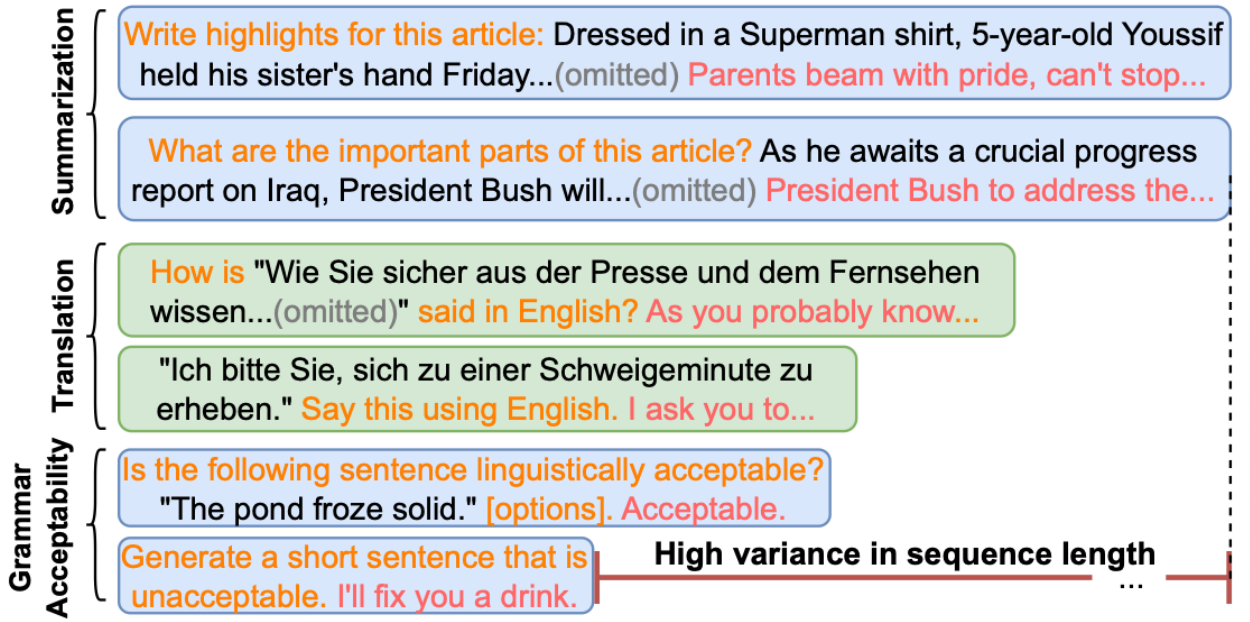
- An illustration of zero bubble with 16 stages: inserting W ops strategically

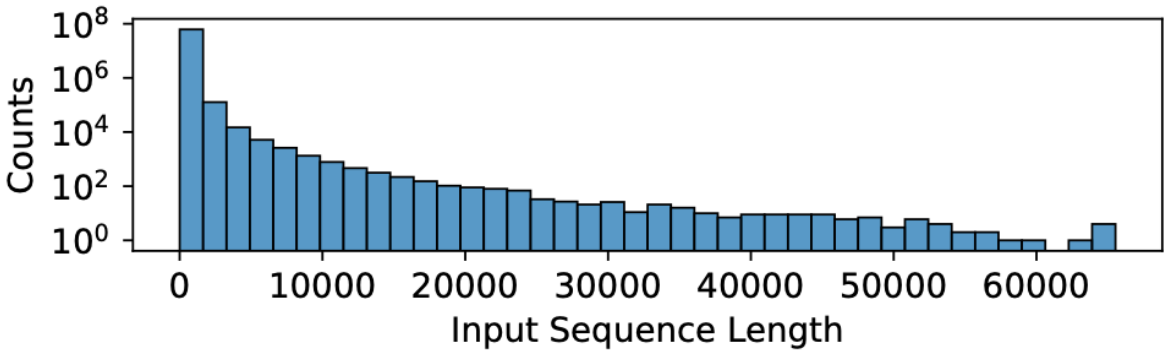


《Zero Bubble Pipeline Parallelism》

Pipeline Training

- Multi-task model pretraining
  - Input sequence lengths are non-uniform





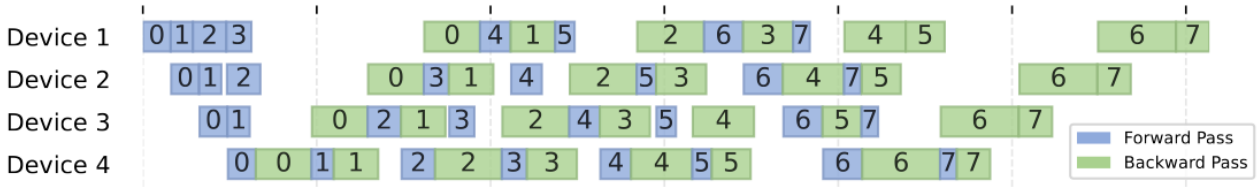
- (left) Orange texts are instructions to the model. Inputs to process are colored black. Expected responses are in red
- (upper) The sequence length distribution in dataset

《DynaPipe: Optimizing Multi-task Training through Dynamic Pipelines》

Pipeline Training

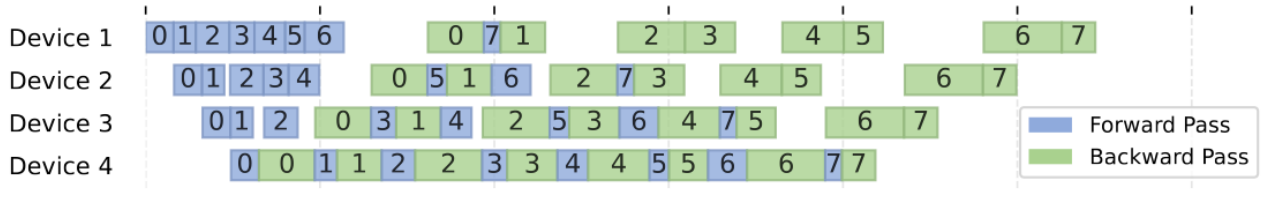
- Multi-task training pipeline: packing and scheduling

(a) 1F1B (uniform micro-batches)

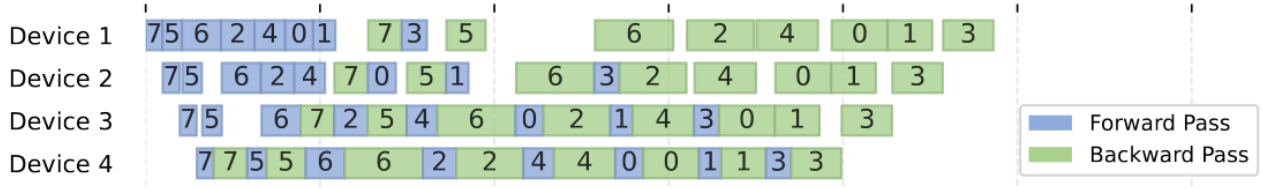


(b) 1F1B (dynamic micro-batches)

Misaligned pipeline with irregular bubbles



(c) Adaptive Schedule



(d) Adaptive Schedule & Micro-batch Reordering

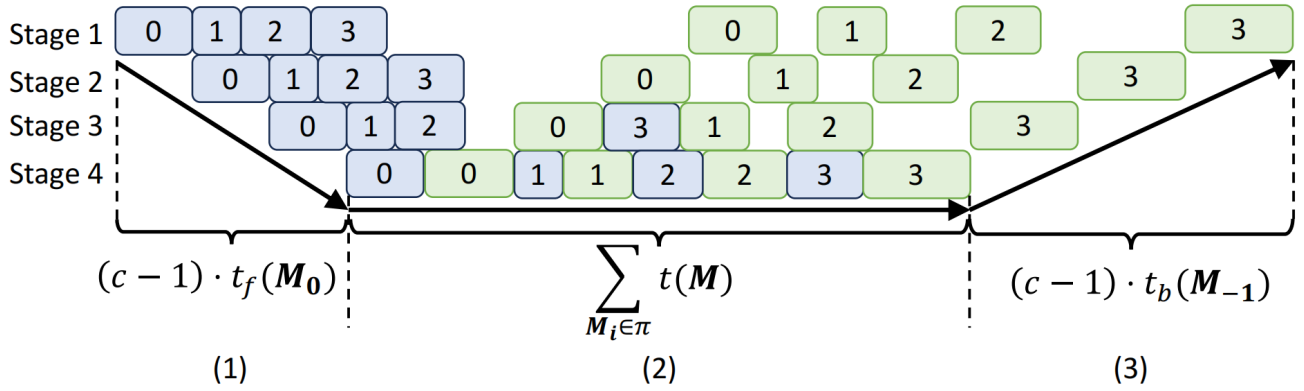
Adaptive scheduling controls the injection time of micro-batches into the pipeline; reordering adjusts the sequence of computations for different microbatches

Pipeline Training

- Multi-task model pretraining (for reference)
  - Seek the best micro-batch assignment $\boldsymbol{\pi}$ to minimize iteration time

$$\min_{\boldsymbol{\pi}} \left\{ (c-1) \cdot \max\{t(\mathrm{M_i}) | \mathrm{M_i} \in \boldsymbol{\pi}\} + \sum_{\mathrm{M_i} \in \boldsymbol{\pi}} t(\mathrm{M_i}) \right\}$$

  - Construct a dynamic programming algorithm to optimally partition samples



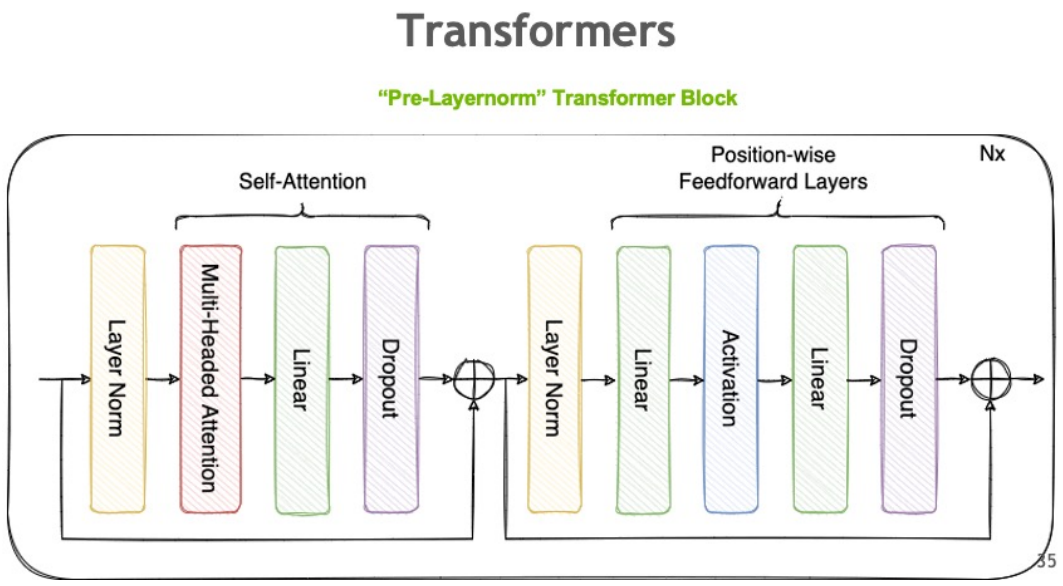
  - Other related works: FlexPipe (ATC 2025), WLB-LLM (OSDI 2025)

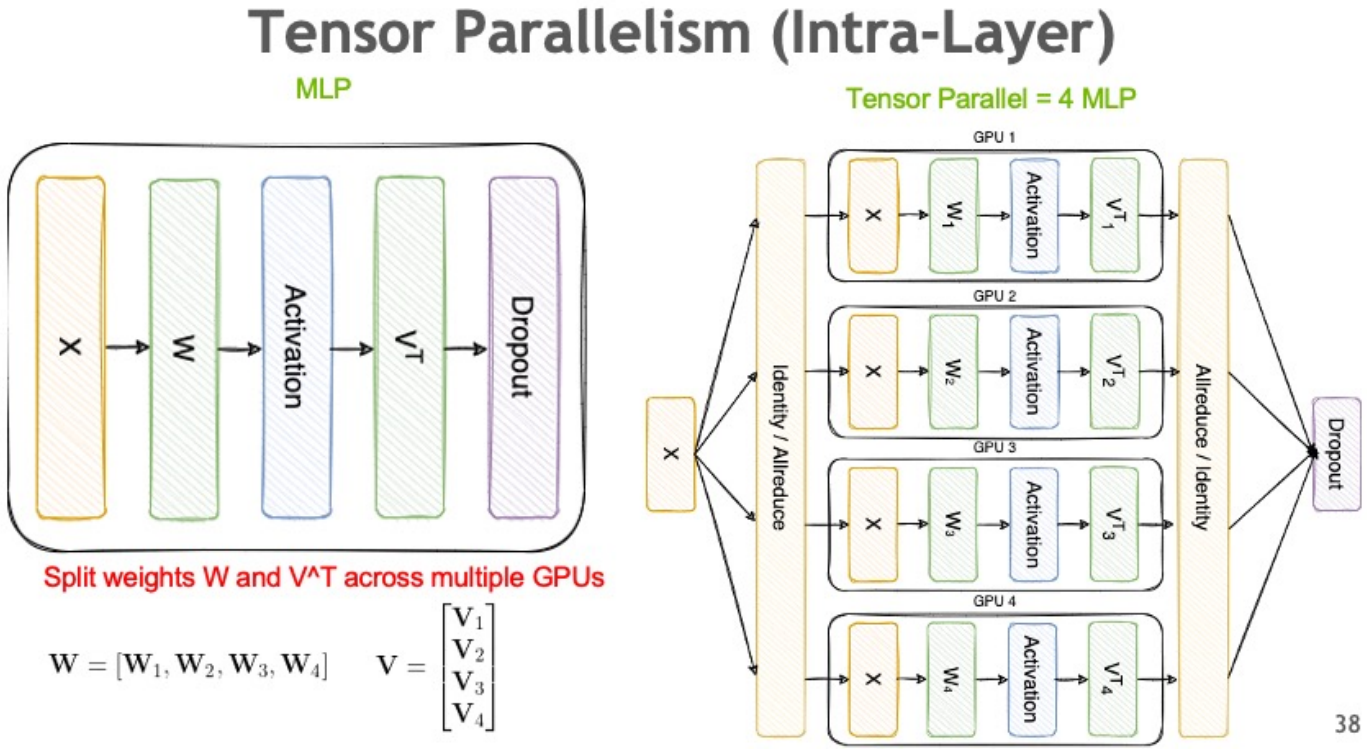
Distributed LLM Training: Outline

- Data Parallelism
  - Parameter-Server
  - All-Reduce
  - Memory Optimization
- Model Parallelism
  - Pipeline Parallelism
  - **Tensor Parallelism**
  - Sequence Parallelism
- Mixture of Experts

Tensor Parallelism

- A visual perception of TP



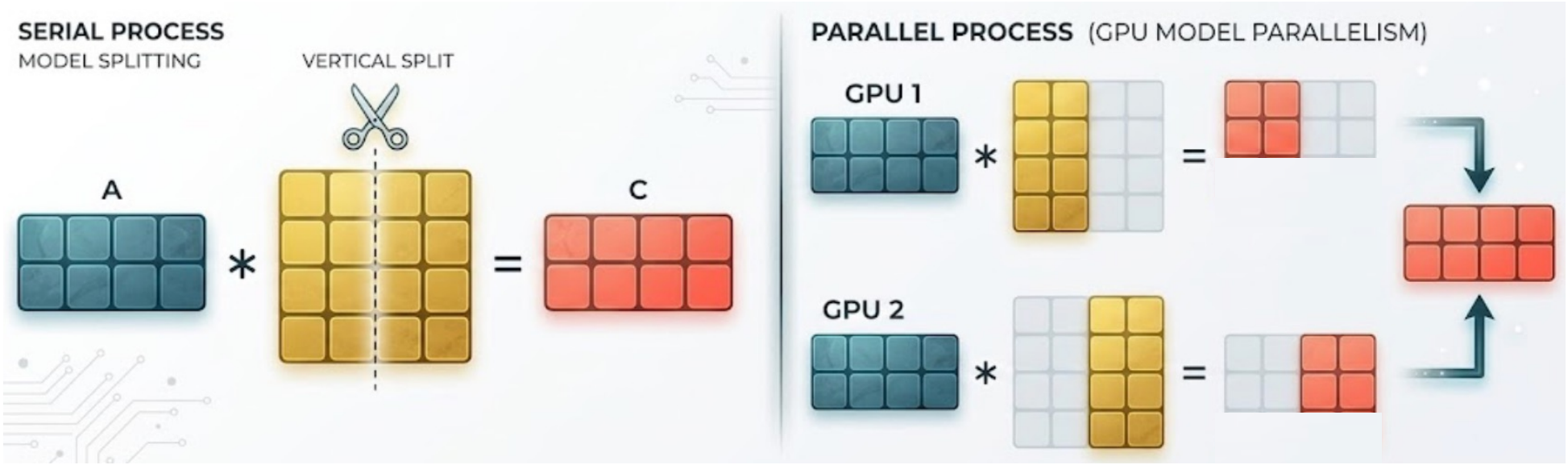


Easily understandable, but how?

《Demystifying Tensor Parallelism》 by *Josh Levy-Kramer*

Tensor Parallelism

- Mathematical principle of tensor parallelism (e.g. GEMM)
  - 1D: segmenting a matrix by **column** or row only
    - First linear layer $Y = XA$: $A = [A_1, A_2] \rightarrow Y = [XA_1, XA_2] = [Y_1, Y_2]$



Tensor Parallelism

- Mathematical principle of tensor parallelism (e.g. GEMM)
  - 1D: segmenting a matrix by **column** or row only
    - Second linear layer $Z = YB$:

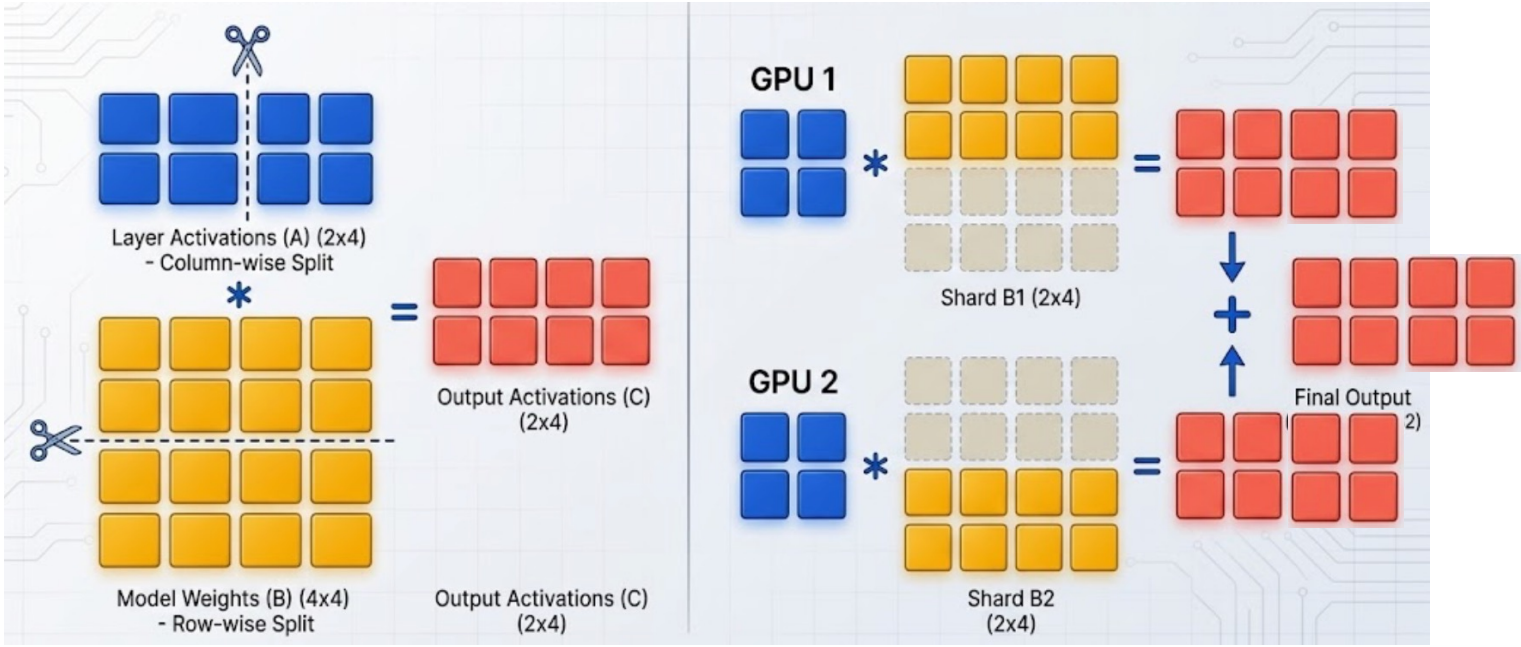
$$Z = [Y_1 \ Y_2] \cdot \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$$

      - Compute $Y_i \ B_i$ independently and aggregate $Z = Y_1 B_1 + Y_2 B_2$
    - Question: memory usage? communication load? end-to-end latency?
      - 1/P computation, $1/P$ parameter, Full activation, $2(P-1)/P$ communication load, 2(P-1) latency

P: computation

Tensor Parallelism

- Mathematical principle of tensor parallelism (e.g. GEMM)
  - 1D: segmenting a matrix by column or **row** only
    - *First linear layer* $Y = XA$: → $Y = [X_1 \ X_2] \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$



Tensor Parallelism

- Mathematical principle of tensor parallelism (e.g. GEMM)
  - 1D: segmenting a matrix by column or **row** only
    - *Second linear layer Z = YB:*

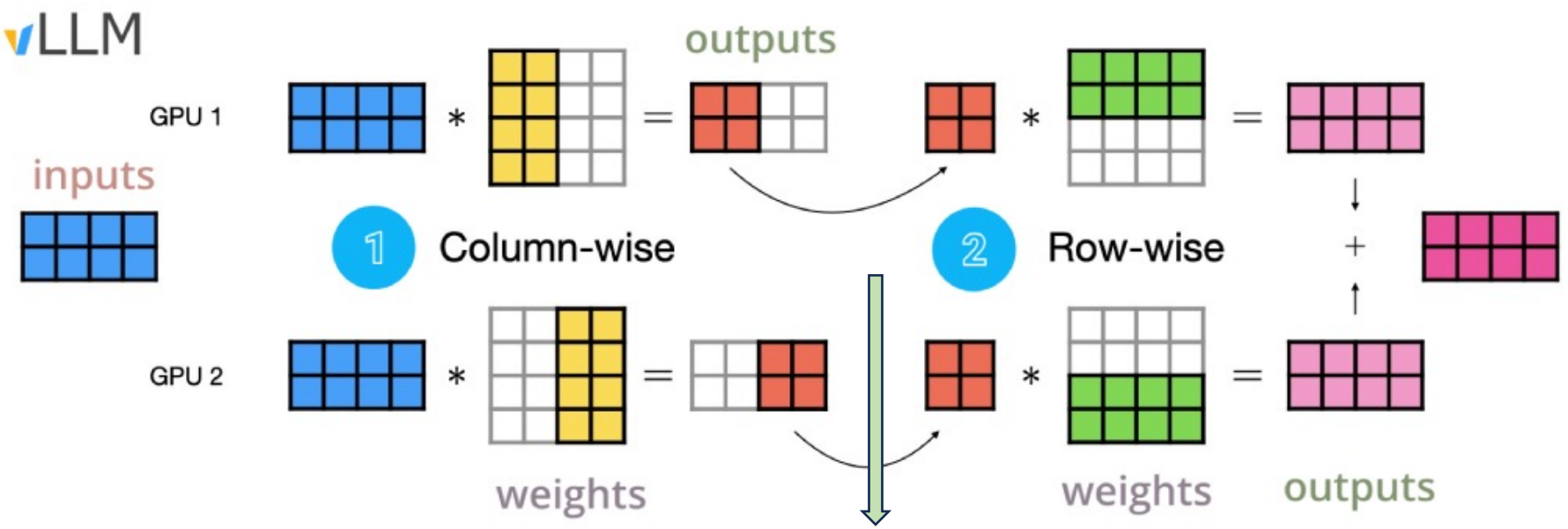
$$Z = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} \cdot \begin{bmatrix} B_1 & B_2 \end{bmatrix}$$

      - *Compute Y_i B_i independently and aggregate* $Z = Y_1 B_1 + Y_2 B_2$

    - Question: memory usage? communication load? end-to-end latency?
      - 1/P computation, 1/P parameter, Full activation, 2(P-1)/P communication load, 2(P-1) latency

Tensor Parallelism

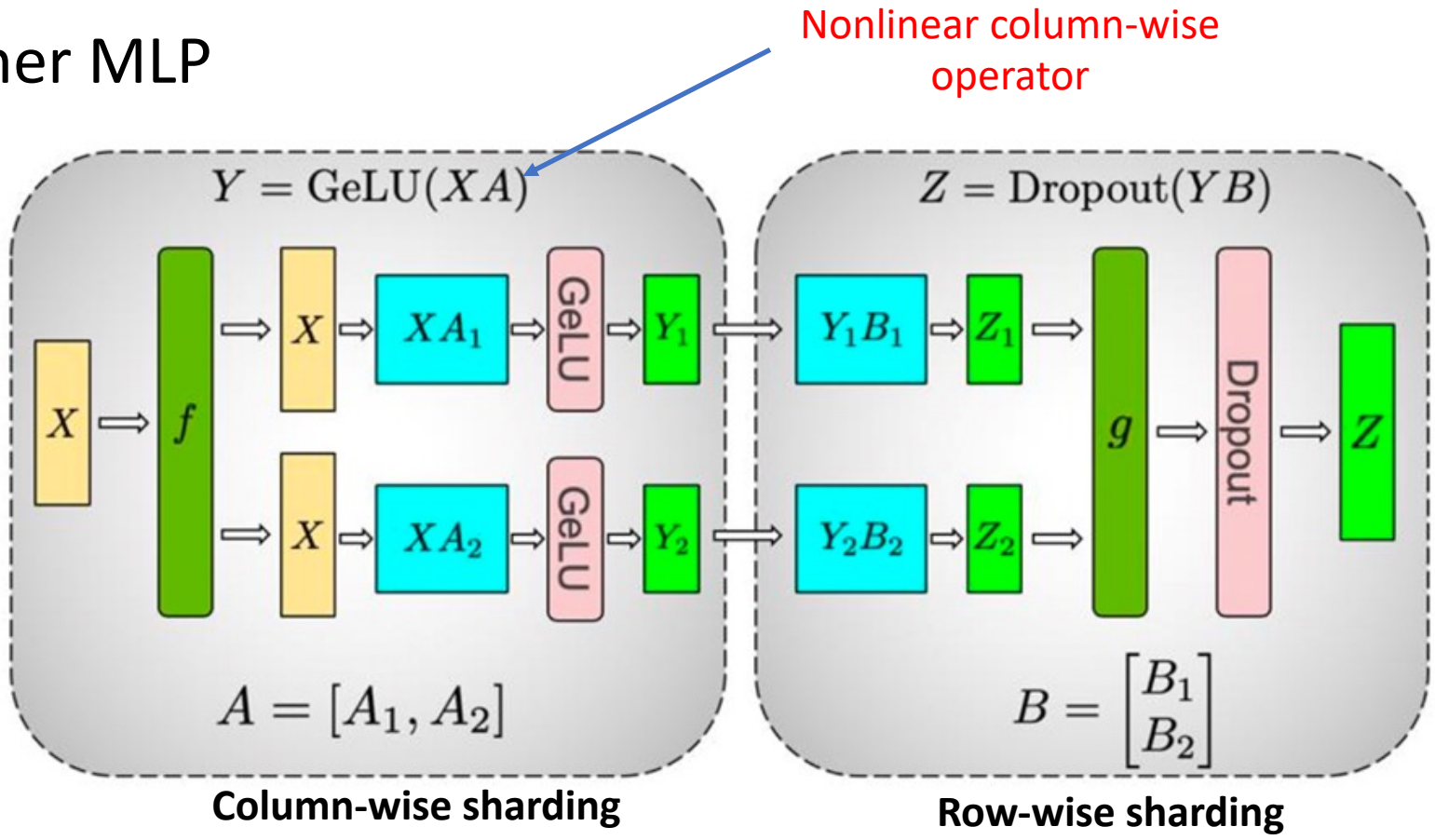
- How to split matrices
  - The column-wise and row-wise styles can be combined for multiple linear layers



Source: https://cloudthrill.ca/what-is-vllm-features

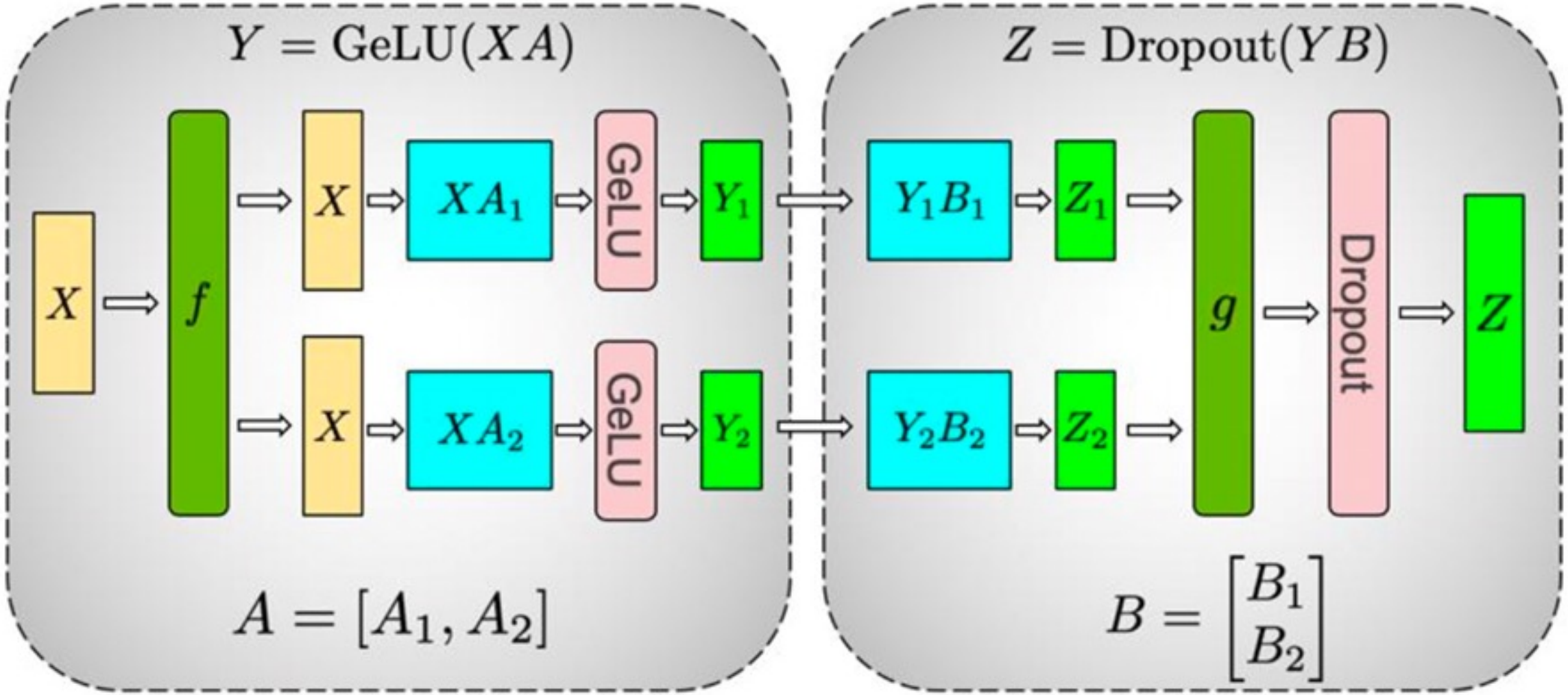
Tensor Parallelism

- Transformer MLP



f and ***g*** are **conjugate**, ***f*** is identity operator in the forward pass and all-reducein the backward pass while ***g*** is all-reduce in forward and identity in backward

Tensor Parallelism



y rows

f

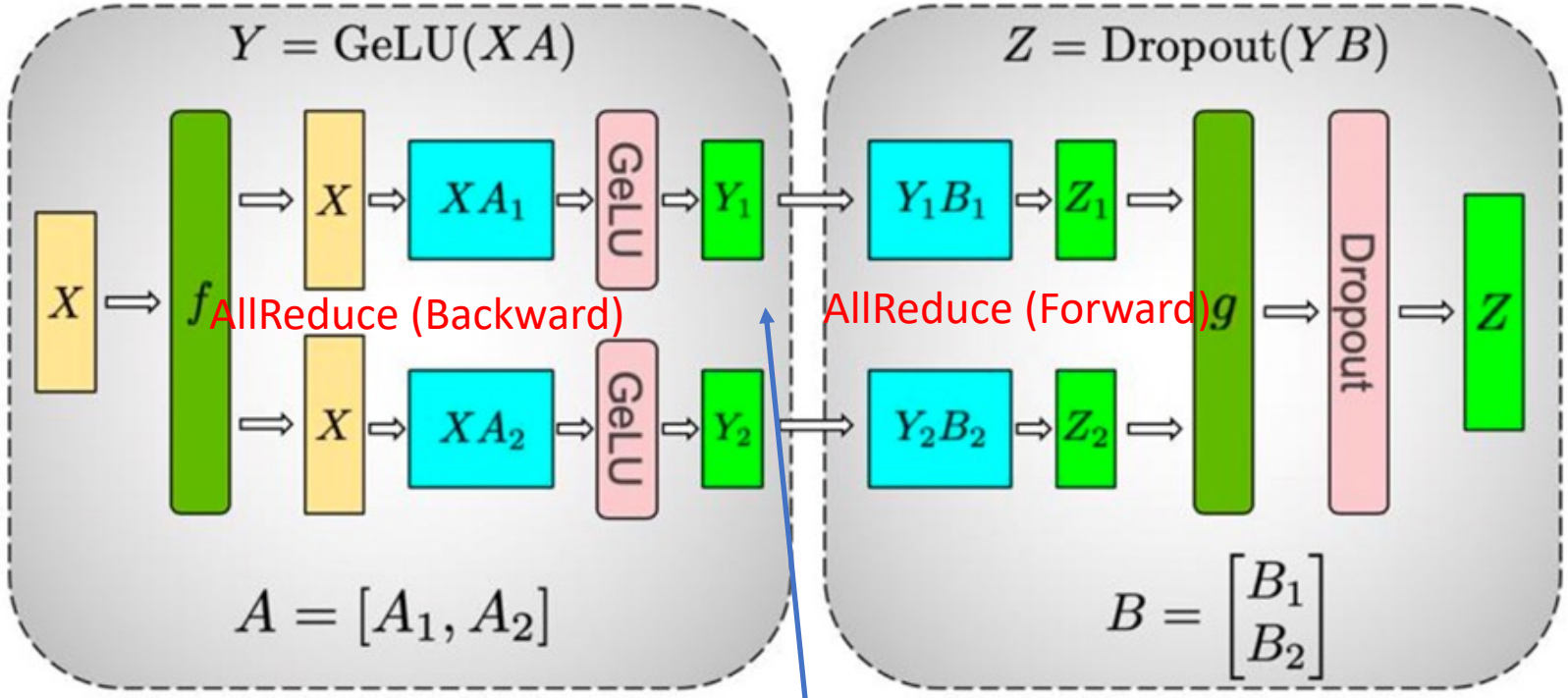
ete

s via ***g***

X on

Tensor Parallelism

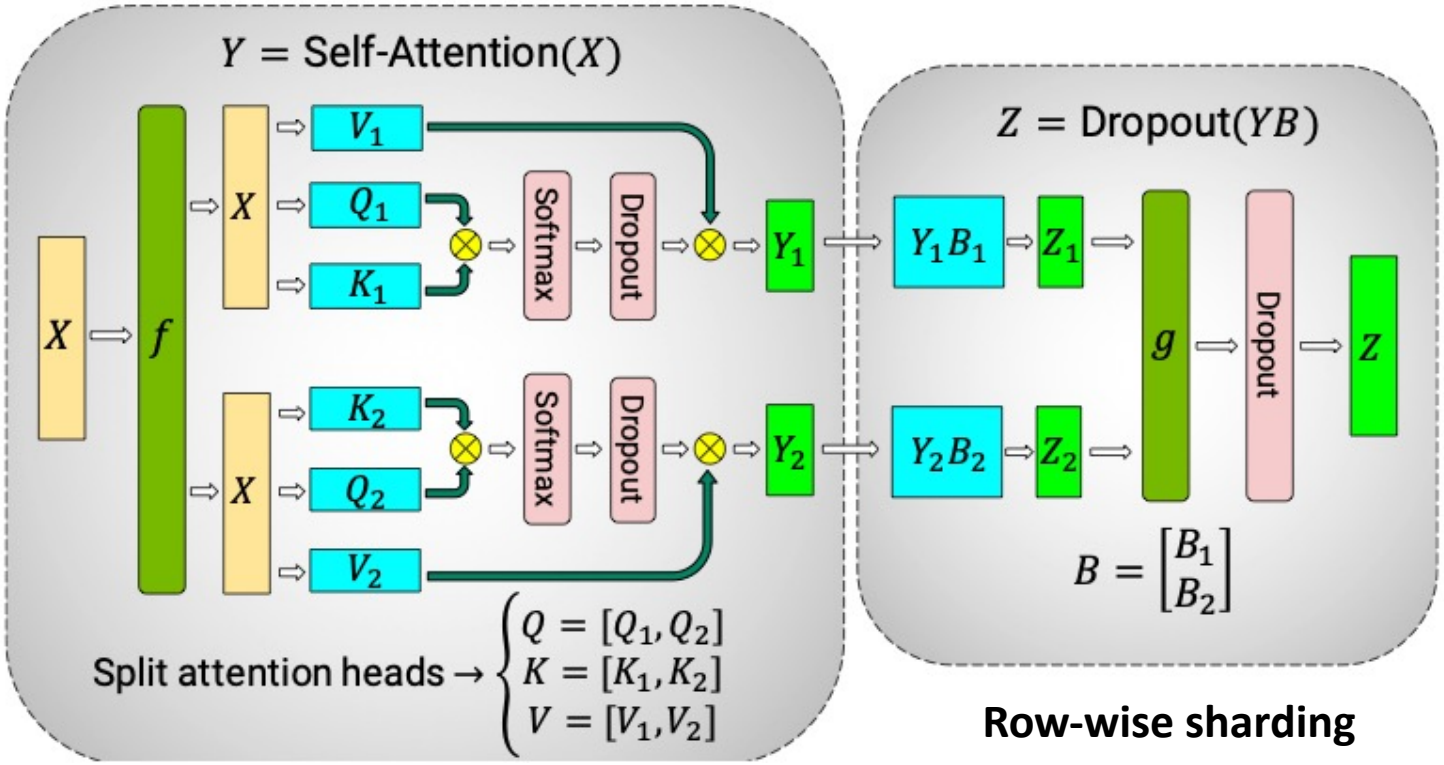
- Transformer MLP: *communication analysis*



f and ***g*** are **conjugate**, ***f*** is identity operator in the forward pass and all-reduce in the backward pass while ***g*** is all-reduce in forward and identity in backward

Tensor Parallelism

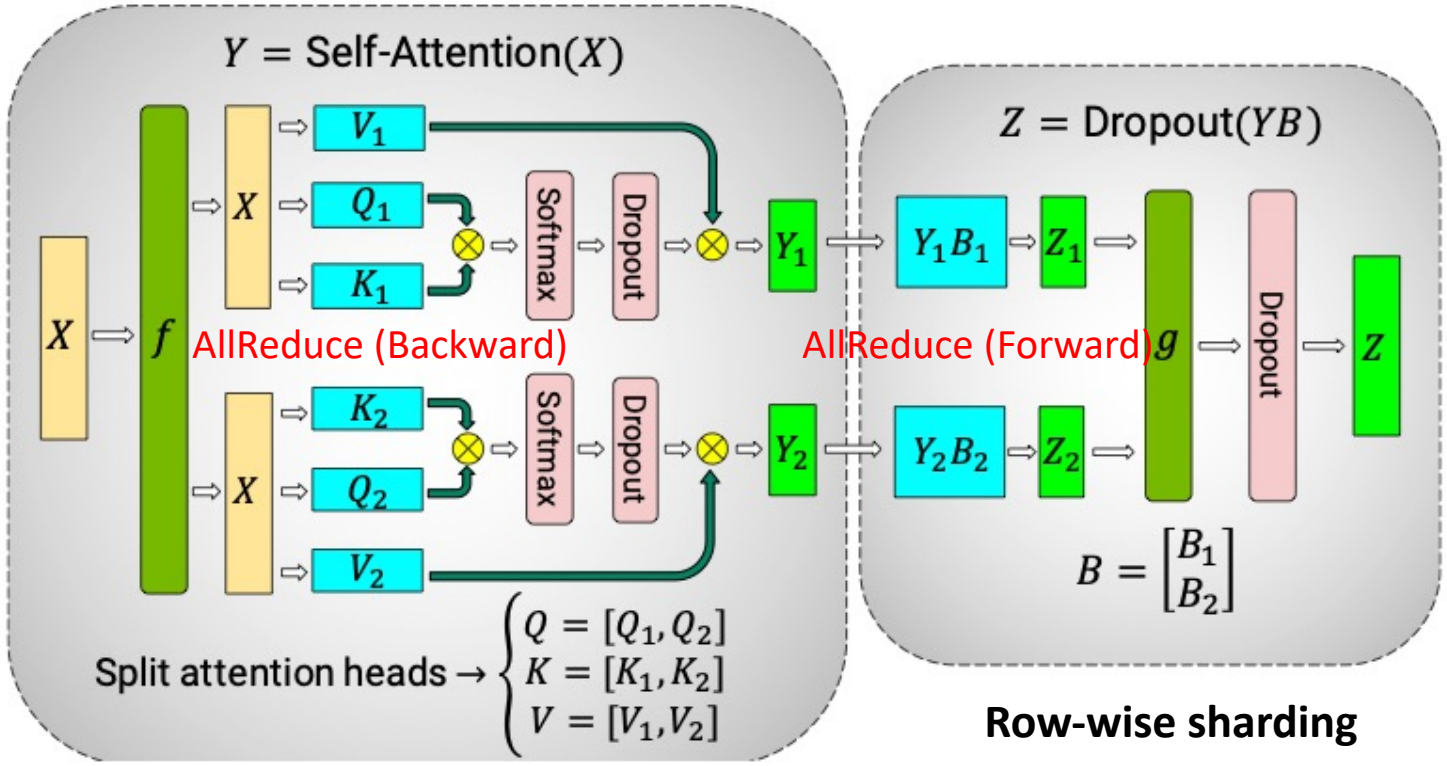
- Transformer Self-Attention



- Partition three parameter matrices W_Q, W_K and W_V by columns and partition the linear layer B by rows
- Forward pass
  - "Copy" X to both GPUs via ***f***
  - Execute math operations such as computing Q, K, V
  - "Add" Z_1 and Z_2 for a complete output Z via ***g***
- Backward pass
  - "copy" $\partial L/\partial Z$ to both GPUs via ***g***
  - Execute math operations
  - "Add" partial derivatives of X on two paths via ***f***

Tensor Parallelism

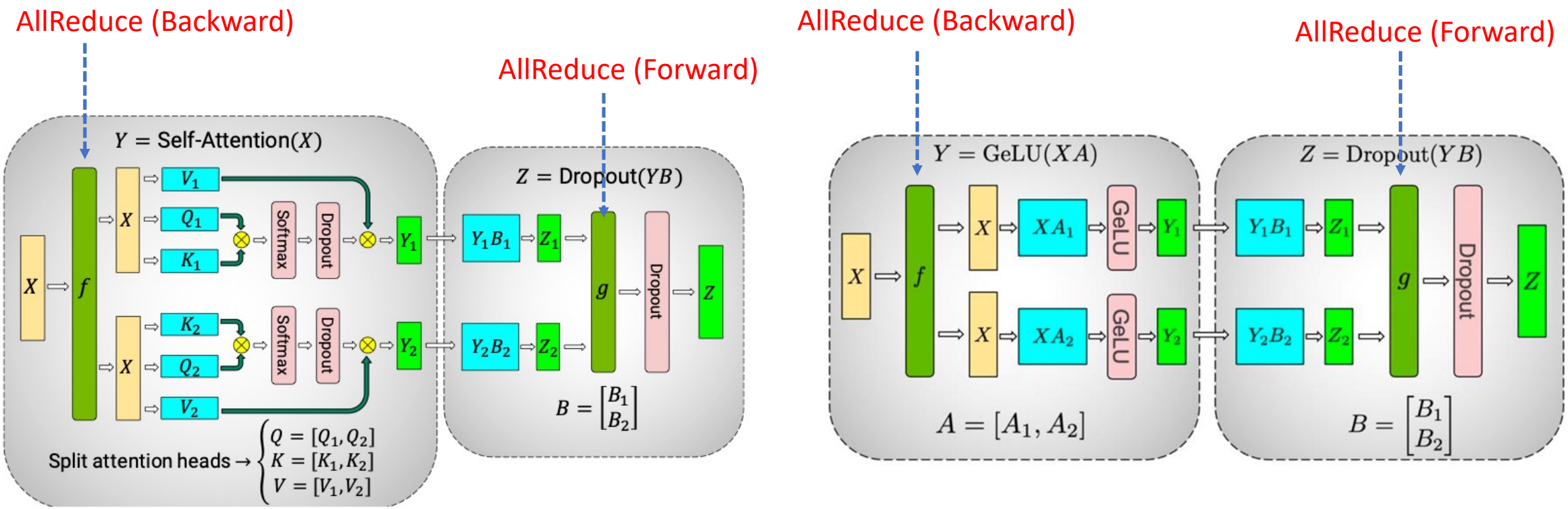
- Transformer Self-Attention



f and ***g*** are **conjugate**, ***f*** is identity operator in the forward pass and all-reduce in the backward pass while ***g*** is all-reduce in forward and identity operator in backward

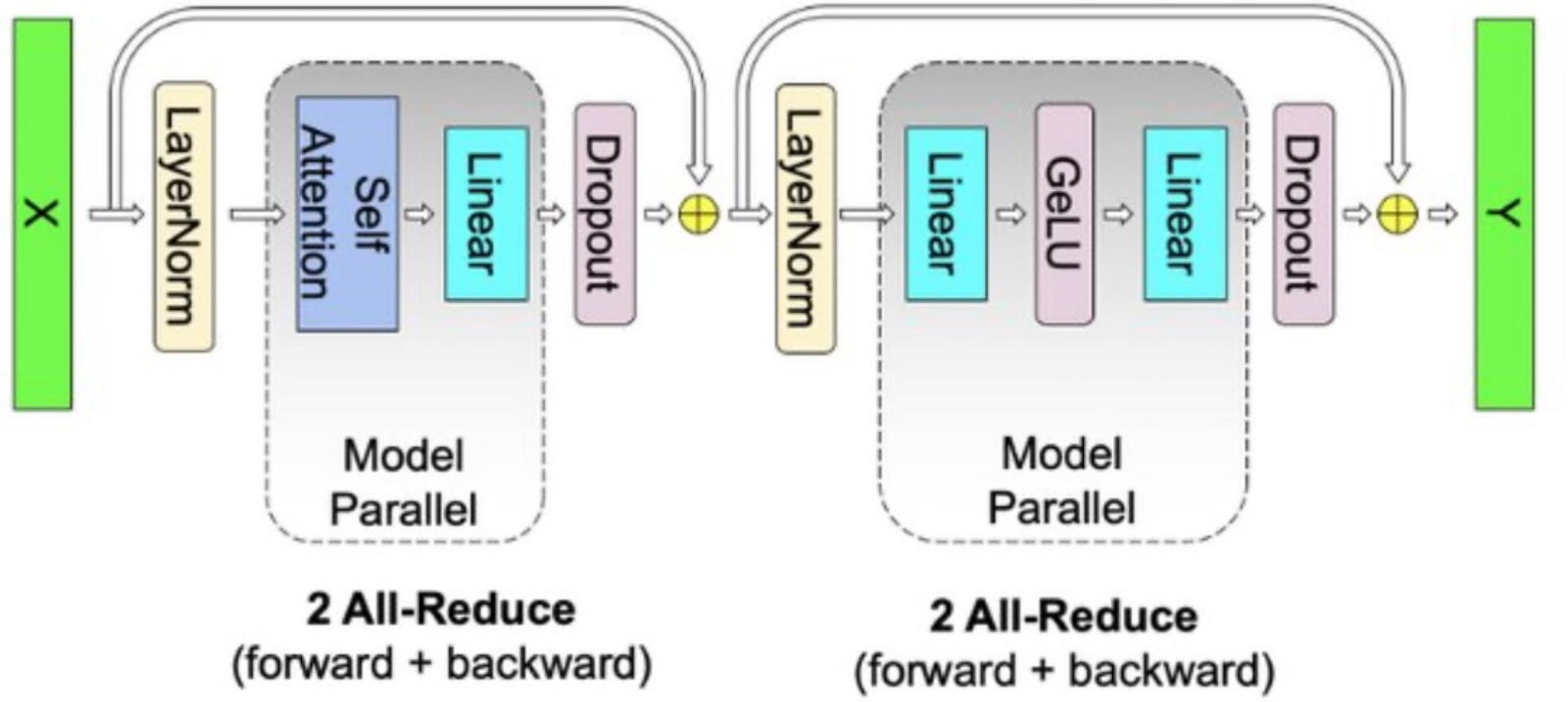
Tensor Parallelism

- TP communication analysis



Tensor Parallelism

- Transformer layer: communication analysis

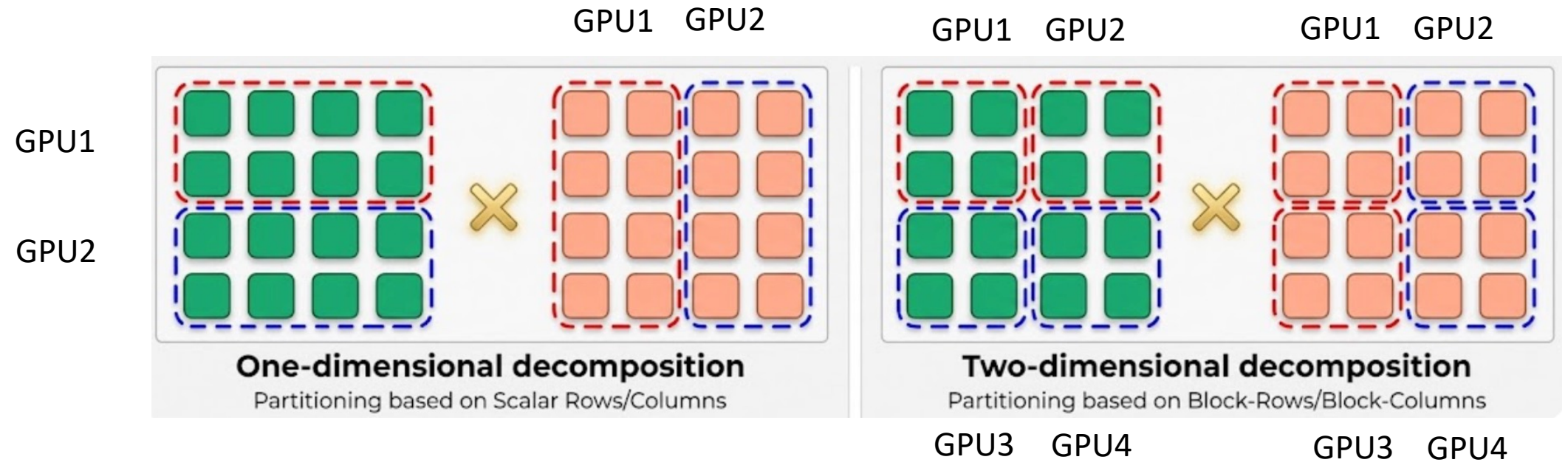


Total communication complexity: $8\ \Phi$. (Φ is the volume of the corresponding parameters)

《Accelerating Large Language Model Training with Hybrid GPU-based Compression》

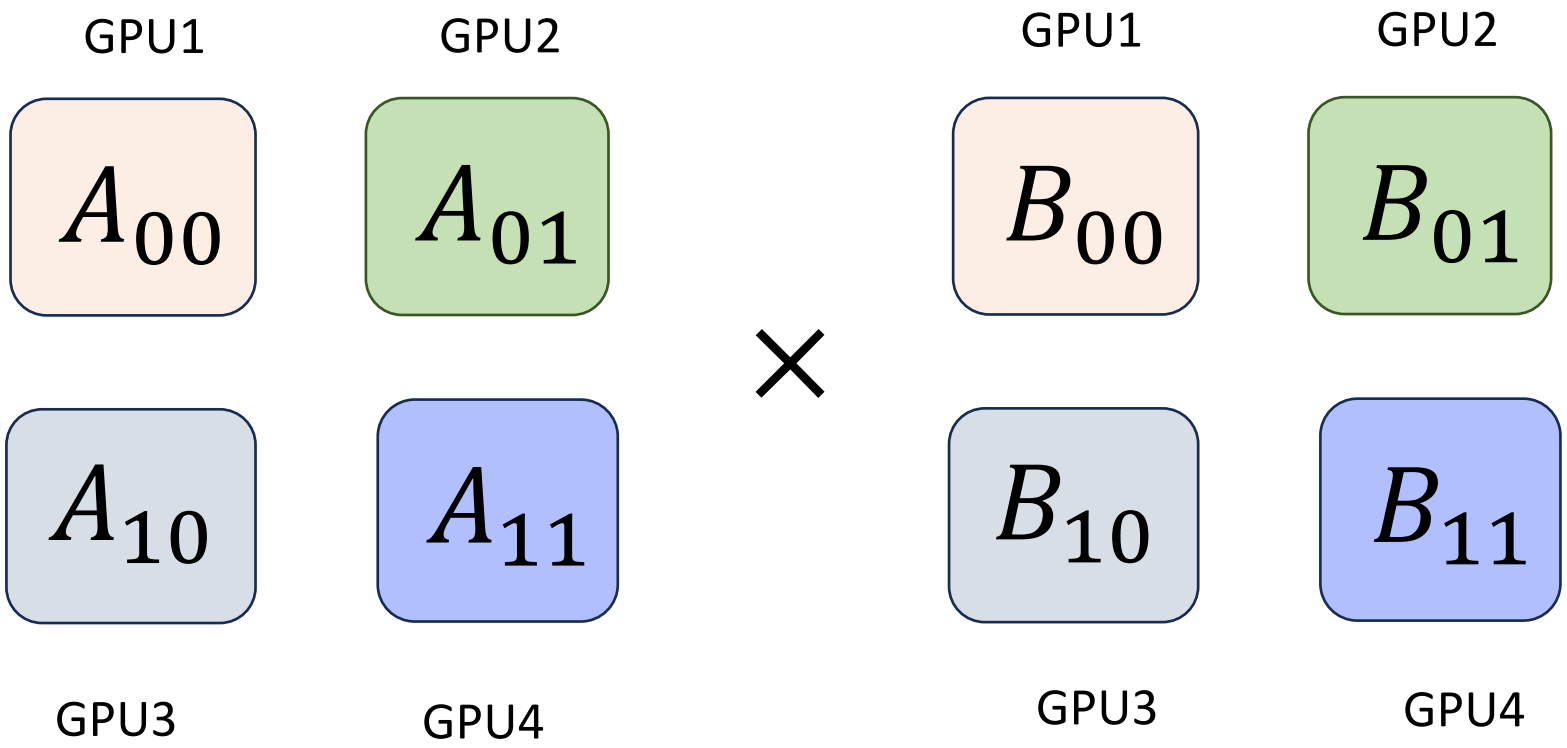
Tensor Parallelism

- Submatrix Multiplication
  - More GPUs working simultaneously

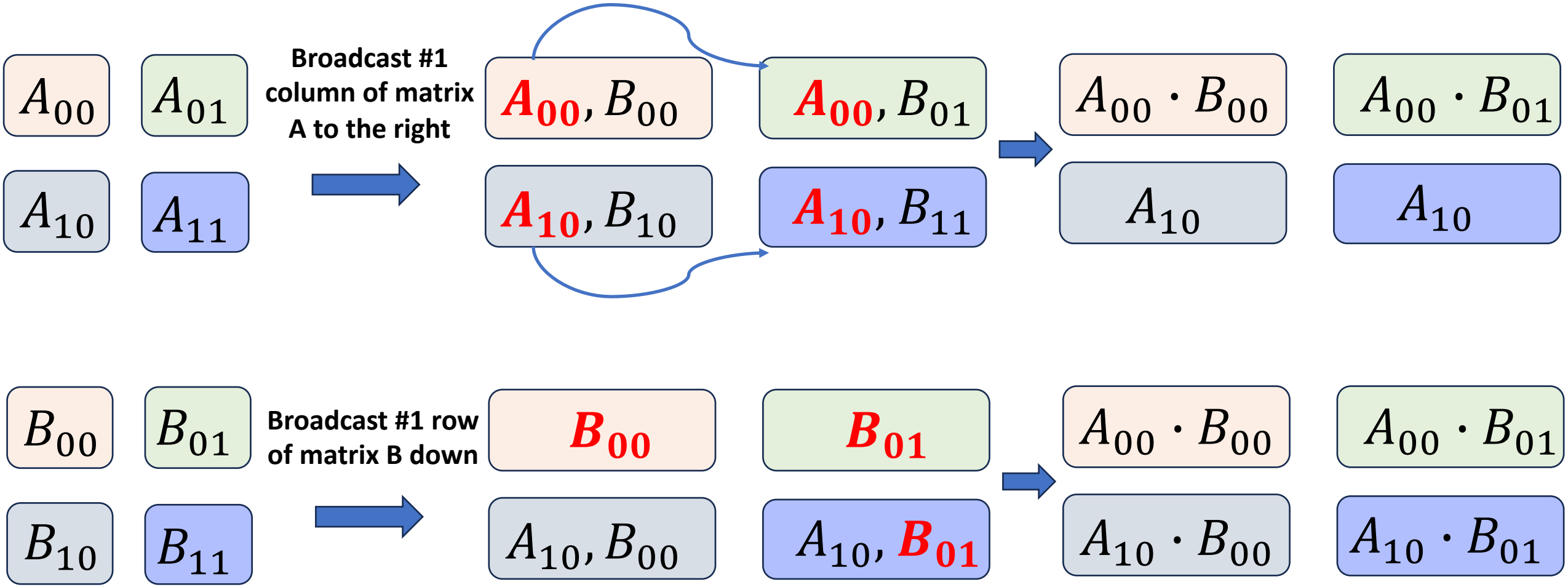


Tensor Parallelism

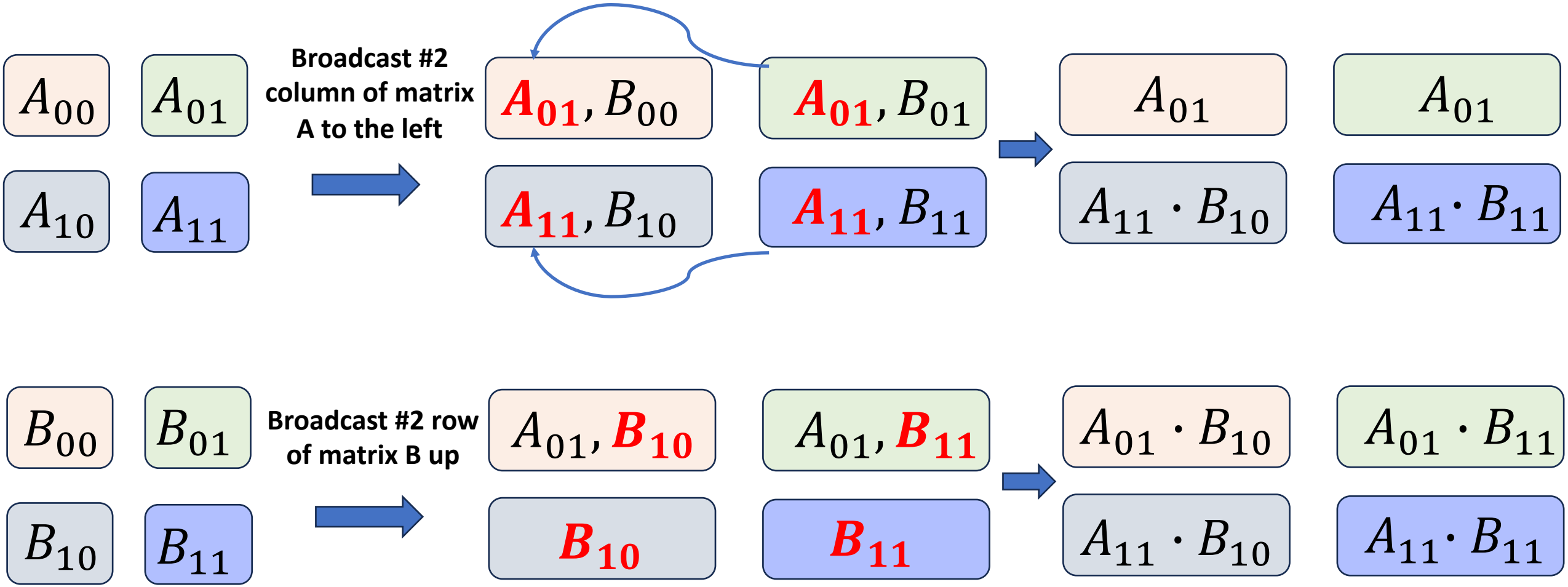
- 2D Tensor Parallelism: each color represents a different GPU



Tensor Parallelism



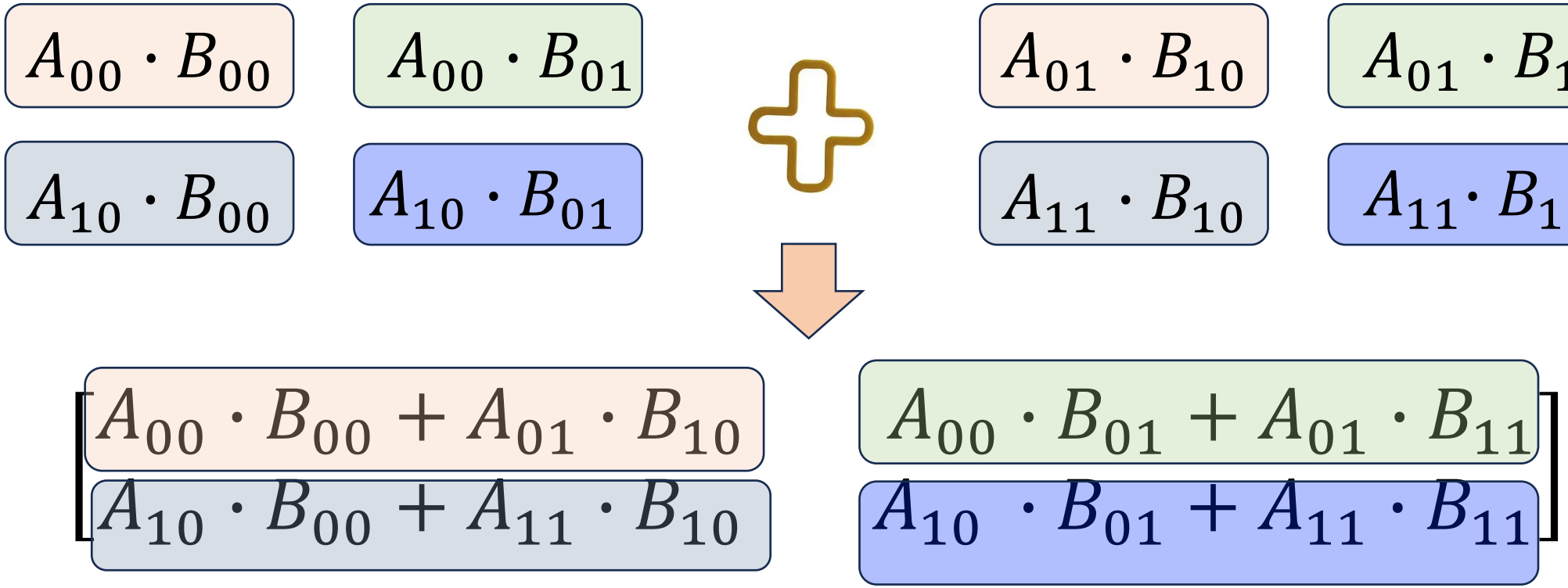
Tensor Parallelism



Tensor Parallelism

- 2D Tensor Parallelism

Sequential computation



- 2D Tensor Parallelism with more rows and columns?

Tensor Parallelism

- 2D Tensor Parallelism: general procedure on $Y = AB$
  - Demand $P = q^2$ GPUs, and partition both input and parameters on P GPUs

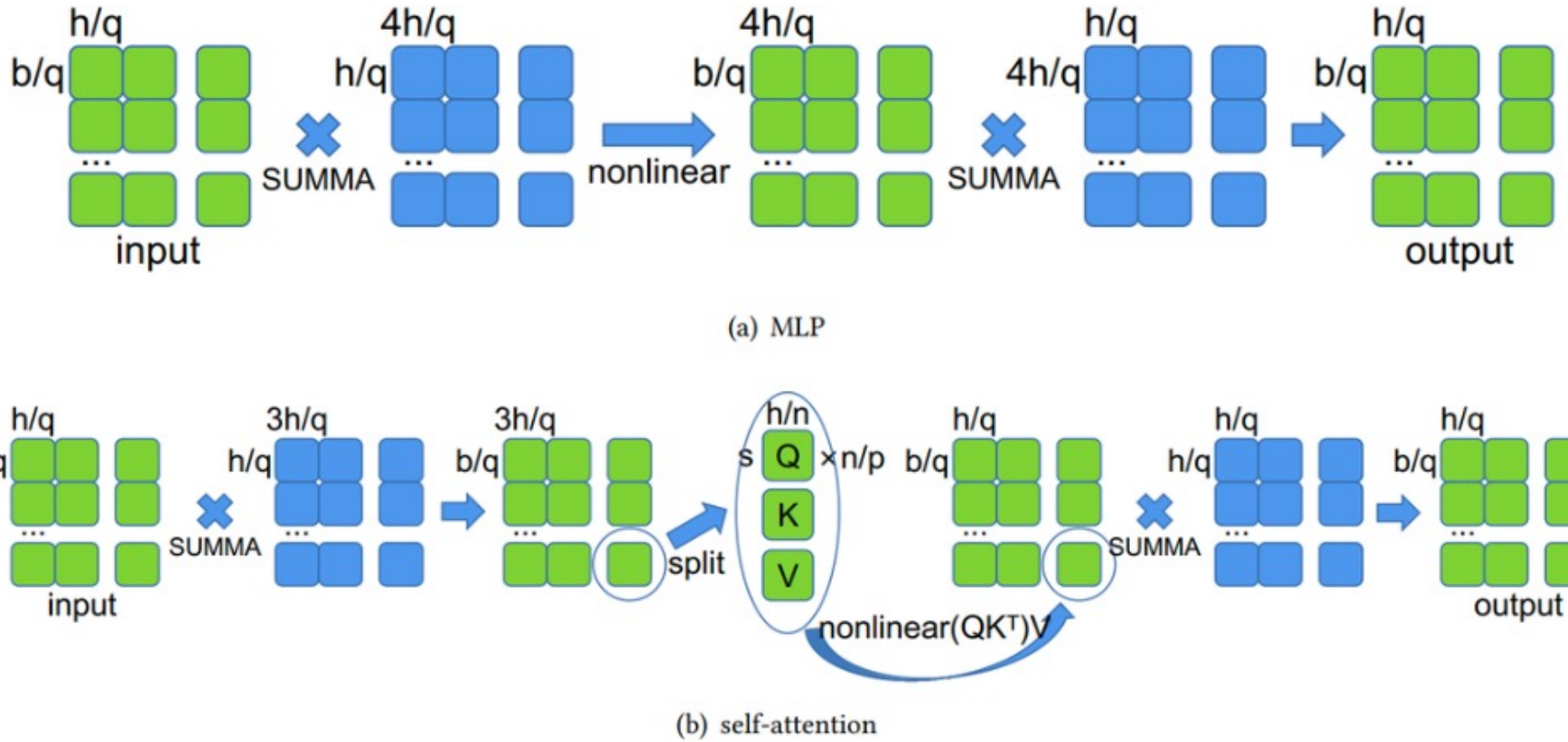
Less per-GPU communication

Computation, memory and communication complexity

- Computation: $O\left(\frac{1}{P}\right)$
- Memory (parameter and activation): $O\left(\frac{1}{P}\right)$
- Communication: $O\left(\frac{1}{q}\right)$ (note that broadcast is replaced by All-Gather)

Tensor Parallelism

- 2D Tensor Parallelism

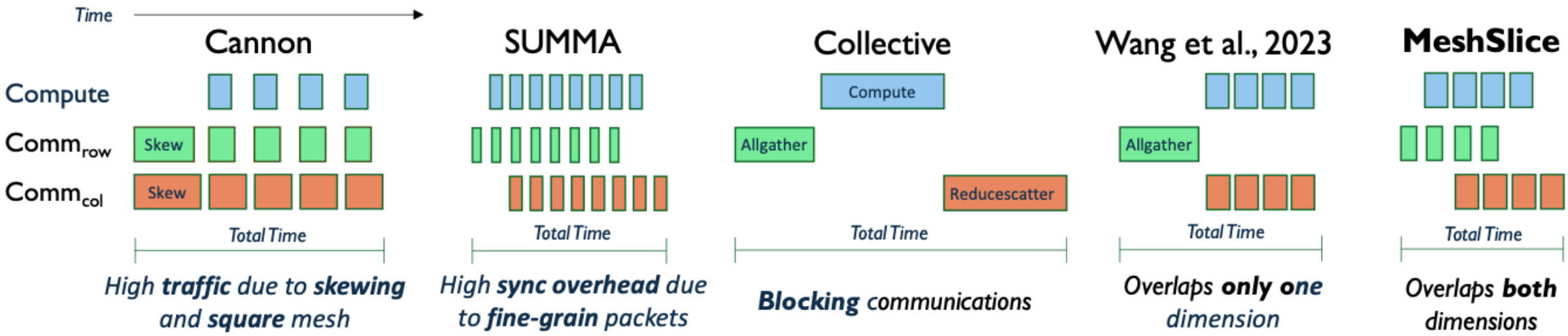


Green: activations, Blue: Transformer modules

《SUMMA: Scalable universal matrix multiplication algorithm》

Tensor Parallelism

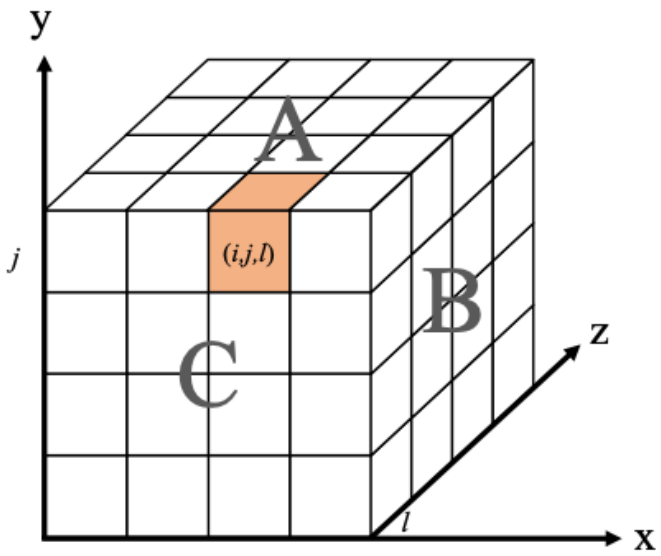
- 2D Tensor Parallelism: still many ongoing works, supplementary reading



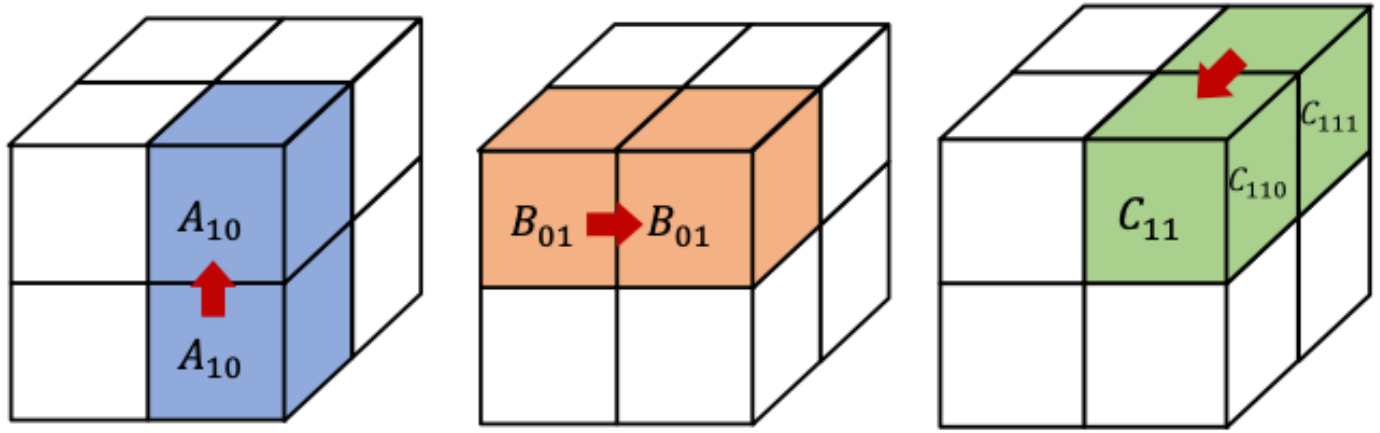
《MeshSlice: Efficient 2D Tensor Parallelism for Distributed DNN Training》

Tensor Parallelism

- 3D Tensor Parallelism: further segmentation for less memory usage
  - Supplementary reading



Colored block **(i,j,k)** represents an example of a single processor.



(a) Broadcast A_{il} (b) Broadcast B_{lj} (c) Reduce C_{ij}

An example of the 3-D parallel matrix multiplication

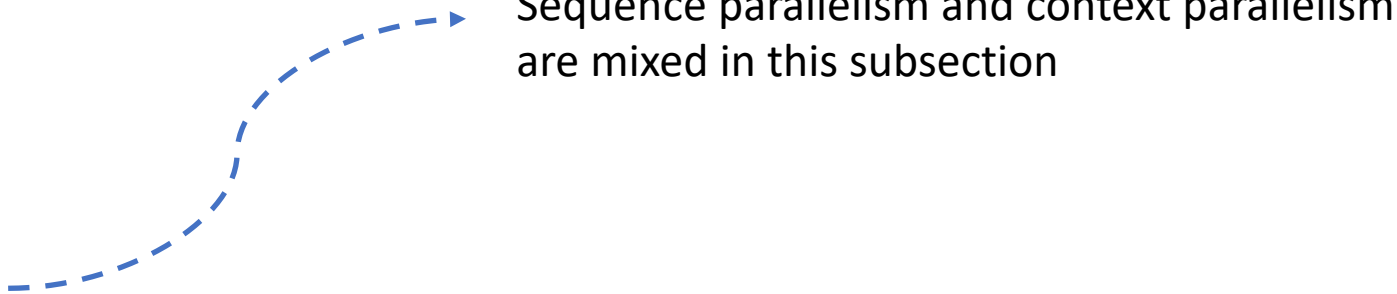
《Maximizing Parallelism in Distributed Training for Huge Neural Networks》

Tensor Parallelism

- Recap
  - Tensor parallelism segments matrix on **feature dimensions** (i.e. hidden dimension and out dimension) that addresses the multiplication of large weight matrices
  - Tensor parallelism introduces **multiple AllReduce operations** that demand nontrivial communication burdens, and is thus constrained by the high bandwidth domain
  - Only for Attention and MLP
  - Keeping the entire input and output to the Attention and MLP layers

Distributed LLM Training: Outline

- Data Parallelism
  - Parameter-Server
  - All-Reduce
  - Memory Optimization
- Model Parallelism
  - Pipeline Parallelism
  - Tensor Parallelism
  - **Sequence Parallelism**
- Mixture of Experts



Sequence Parallelism

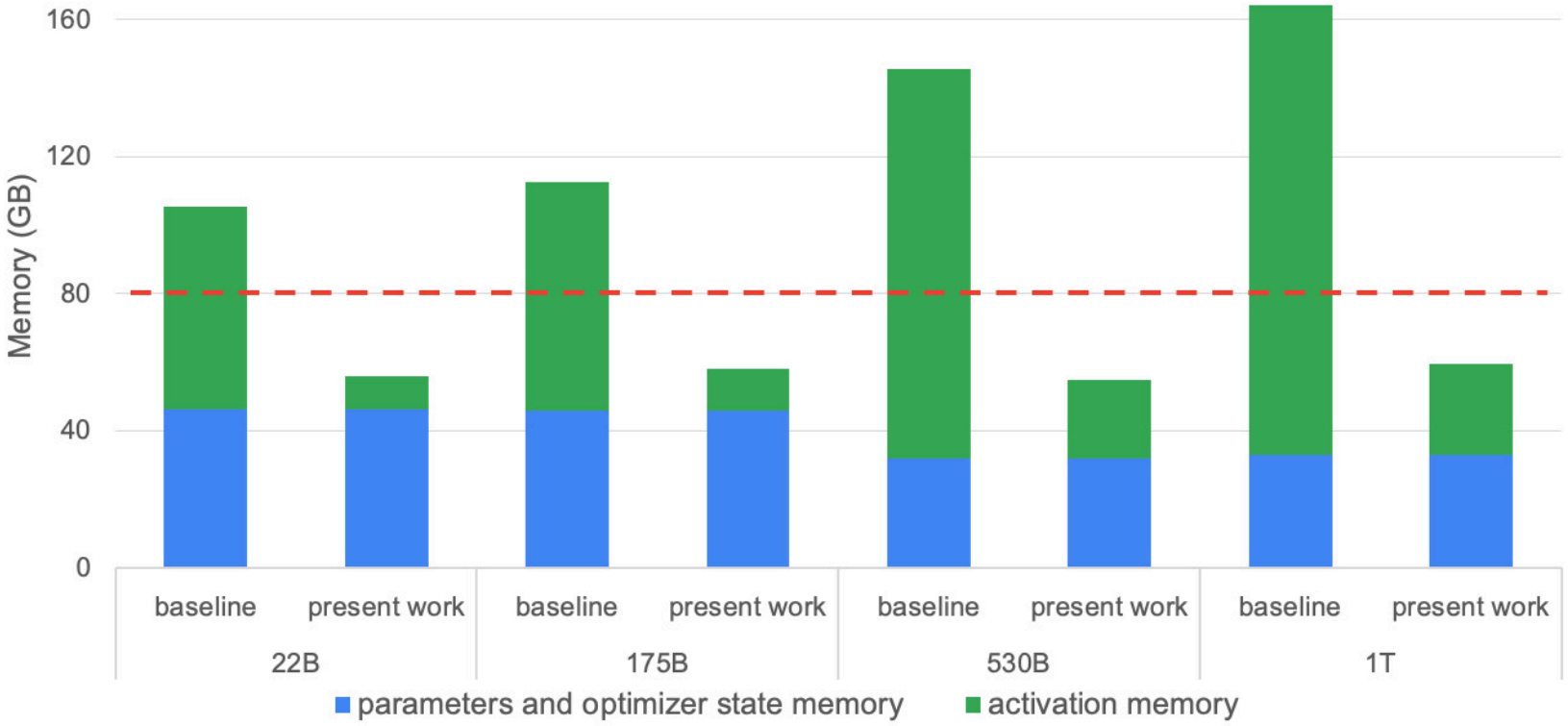
- Sequence Parallelism
  - Tensor parallelism reduces the memory footprint of **model data**, how about **non-model data**?
  - Activation memory is $O(n^2)$ in *self-attention* with *n* being the sequence length
    - Attention scores not stored in HBM
    - Activations include ***Q***, ***K***, ***V*** matrices

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

  - *Some layers have not been parallelized: layernorm and dropout*

Sequence Parallelism

- Sequence Parallelism



Parameters, optimizer state, and activations memory

《Reducing Activation Recomputation in Large Transformer Models》

Sequence Parallelism (Recap)

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

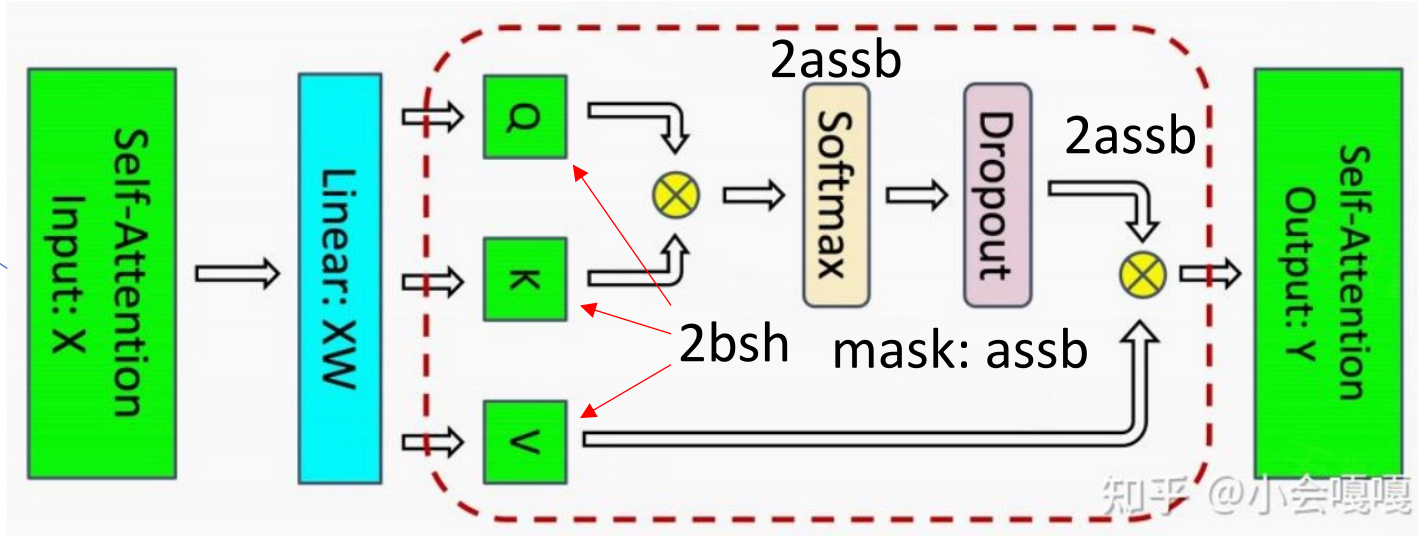
Output of a previous layer, intermediate variable

$$y = Wx + b$$

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

Cannot be discarded after FP

bsh * 2 Bytes



In-total: 5abss + 8sbh

Sequence Parallelism (Recap)

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

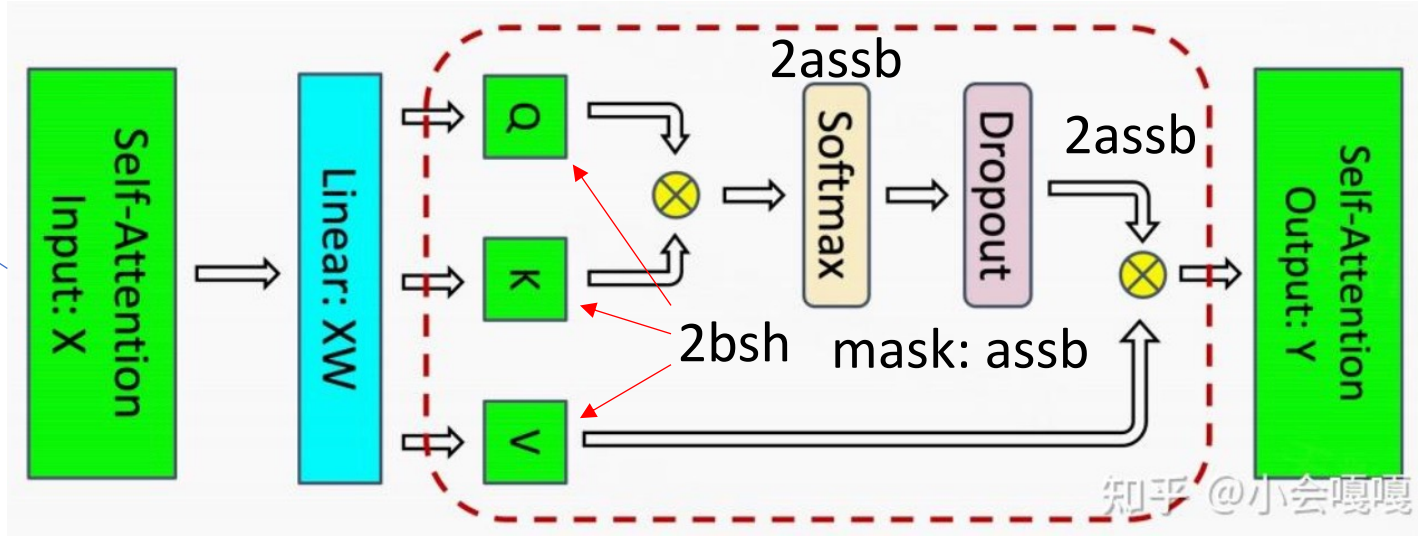
Output of a previous layer, intermediate variable

$$y = Wx + b$$

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

Cannot be discarded after FP

bsh * 2 Bytes

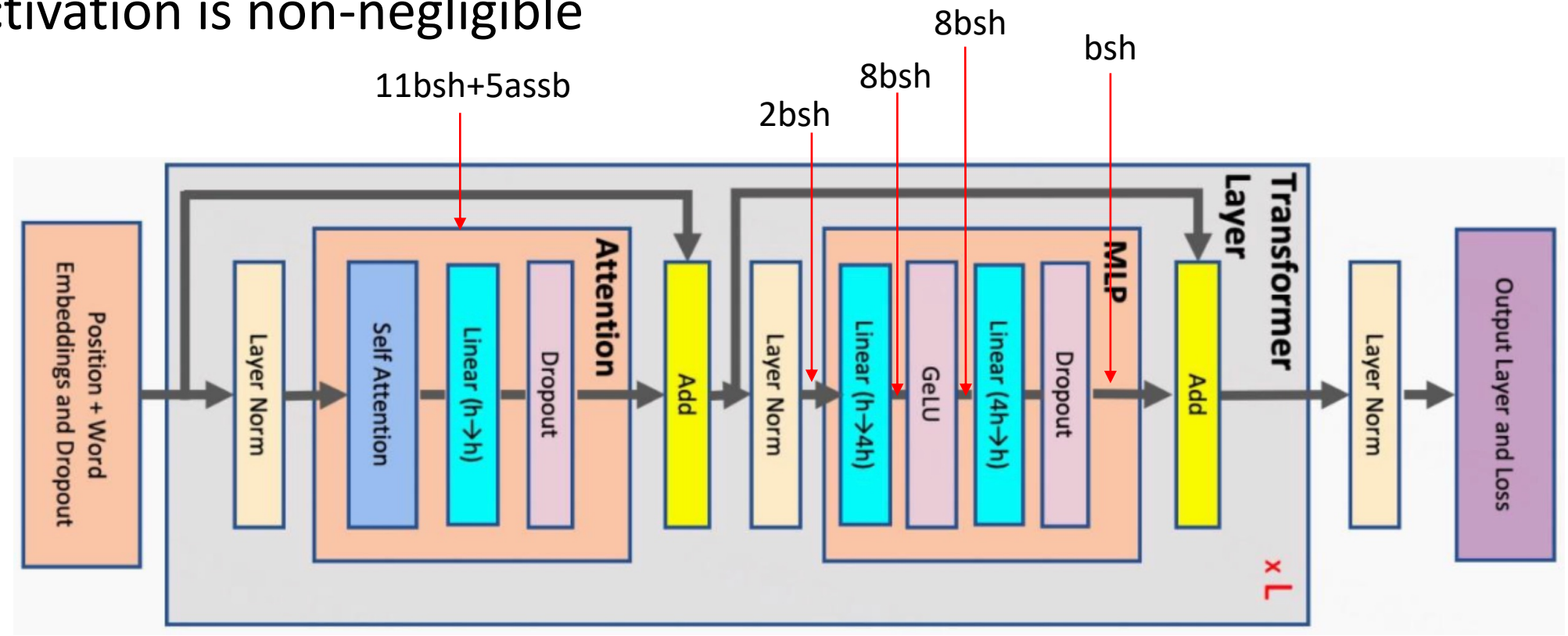


3sbh: Linear layer + dropout layer

In-total: 5abss + 11sbh

Sequence Parallelism (Recap)

- Activation is non-negligible



In-total: 5abss + 34sbh

Sequence Parallelism

- What to keep for a Transformer block in HBM?

- MLP
  - Input to MLP ✓
  - Input of Dropout ✓
  - Input of GeLU
  - Inputs of linear matrices ✓

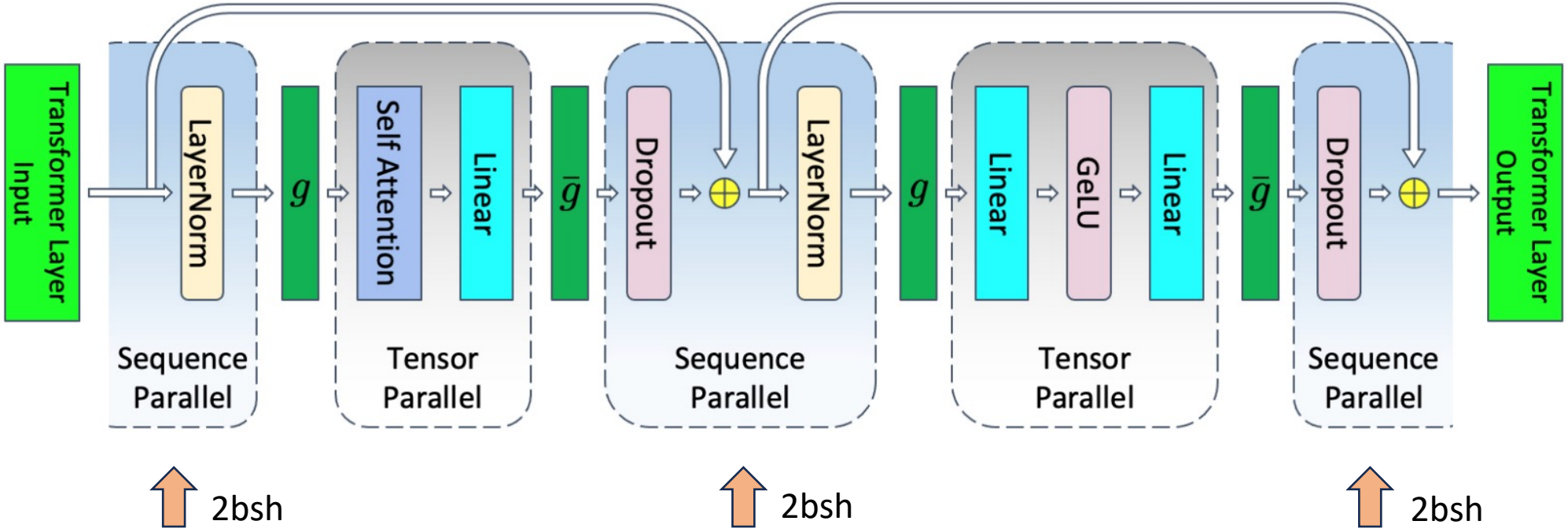
- Attention
  - Input to Attention ✓
  - Q, K and V ✓
  - QK^T
  - Softmax(QK^T) (removed via flashattention) ✓
  - Dropout Mask (removed via flashattention) ✓

Sequence Parallelism

- Representative SP frameworks
  - Megatron-LM SP
    - 《Reducing Activation Recomputation in Large Transformer Models》
    - Deemed as a further optimization of Tensor Parallelism
  - DeepSpeed Ulysses
    - 《DeepSpeed Ulysses: System Optimizations for Enabling Training of Extreme Long Sequence Transformer Models》
  - Colossol AI SP
    - 《Colossal-AI: A Unified Deep Learning System For Large-Scale Parallel Training》
  - Megatron Context Parallelism
    - https://docs.nvidia.com/megatron-core/developer-guide/latest/api-guide/context_parallel.html

Sequence Parallelism

- Megatron-LM sequence parallelism



Tensor parallelism only segments inputs and outputs of ***self-attention*** and ***MLP***

Sequence Parallelism

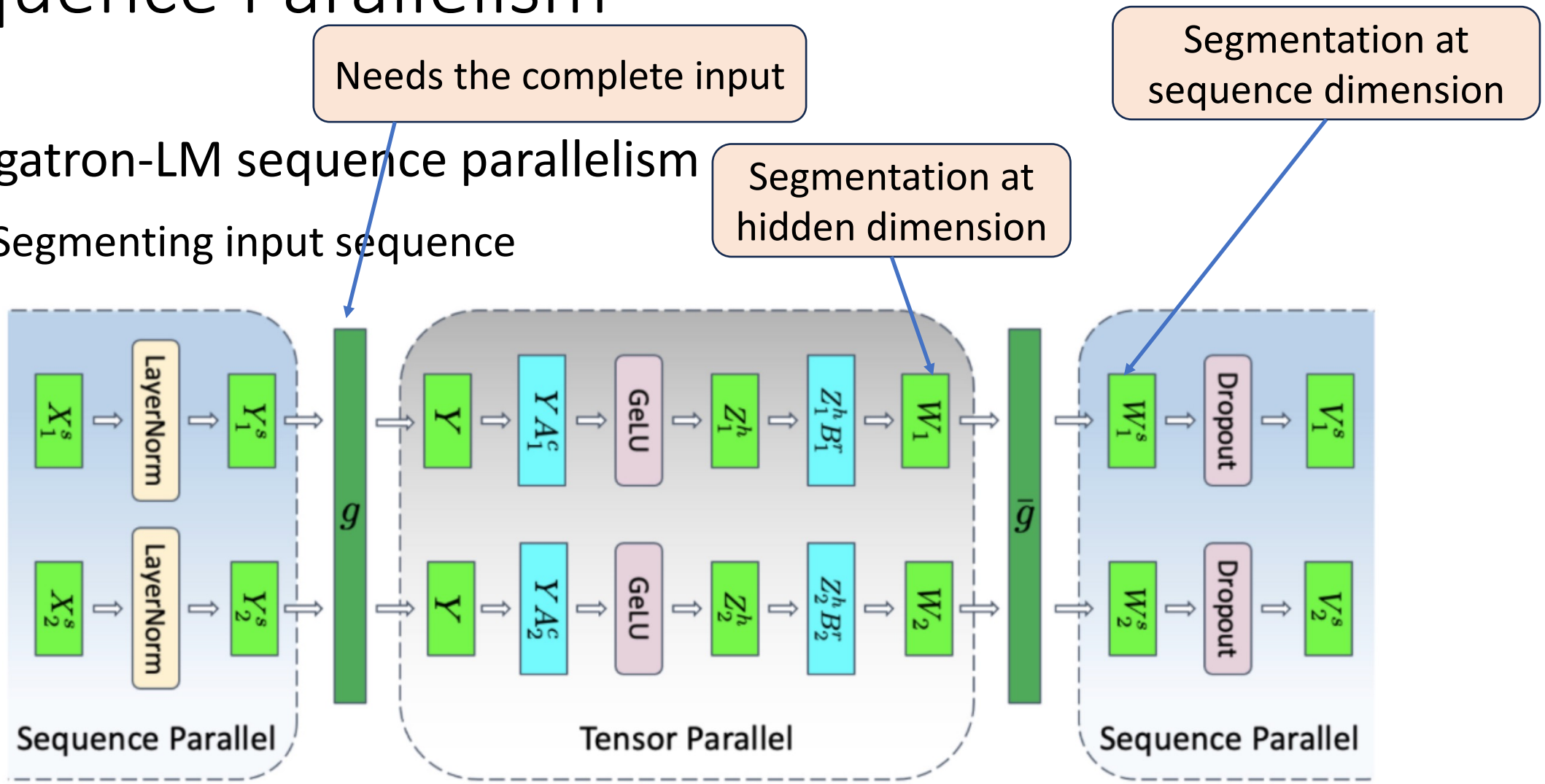
- Megatron-LM sequence parallelism
  - *layernorm* operation
    - The shape of an input matrix: [batch_size, seq_len, hidden_dim]
    - Compute mean, variance and a learnable affine transformation for each ***TOKEN***
    - Can be naturally partitioned by tokens without any cross-device communication
  - *Dropout* operation
    - Each GPU compute dropout independently

Activations (input/output of layernorm) on each GPU: $\frac{2bsh}{t}$

Activations (dropout) on each GPU: $\frac{bsh}{t}$

Sequence Parallelism

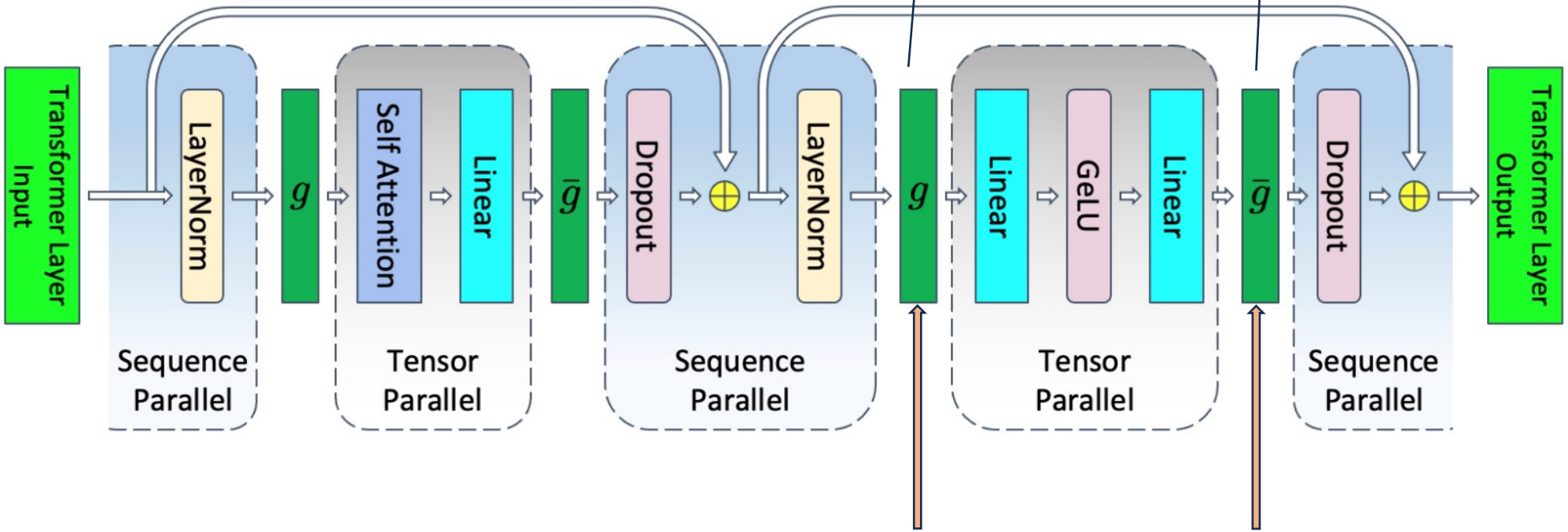
- Megatron-LM sequence parallelism
  - Segmenting input sequence



MLP layer with tensor and sequence parallelism. $\boldsymbol{g}$ and $\bar{\boldsymbol{g}}$ are conjugate. $\boldsymbol{g}$ is all-gather in forward pass and reduce-scatter in backward pass. $\bar{\boldsymbol{g}}$ is reduce-scatter in forward pass and all-gather in backward pass.

Sequence Parallelism

- Megatron-LM sequence parallelism
  - MLP layer (forward pass)

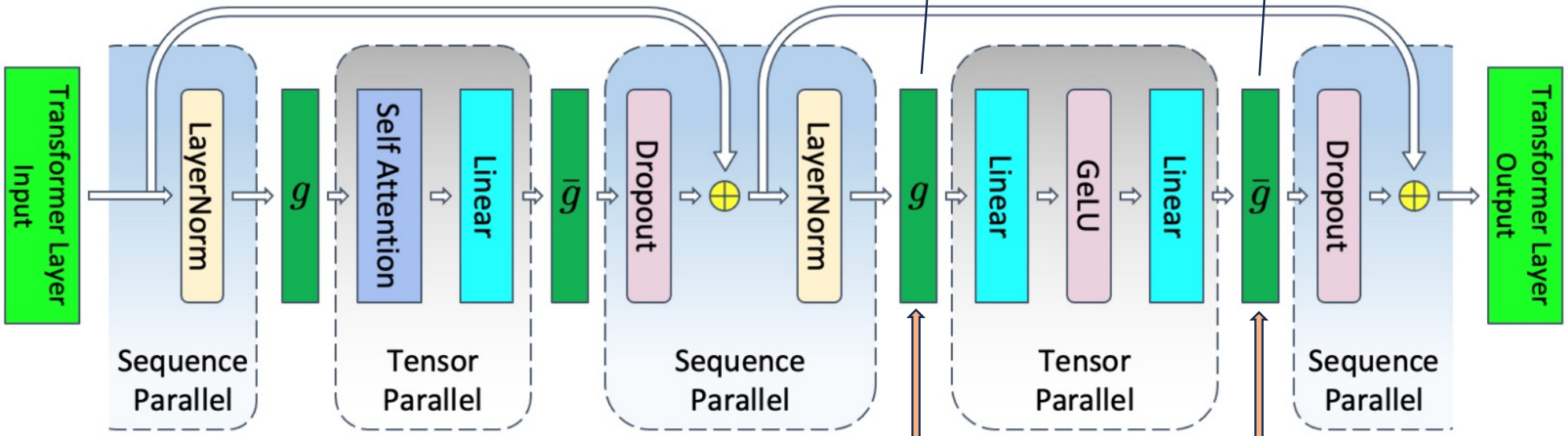


Sequence Parallelism

- Megatron-LM sequence parallelism
  - MLP layer (backward pass)

The currently buffered output is not what you need to proceed to Dropout

Matrix multiplication needs the complete input



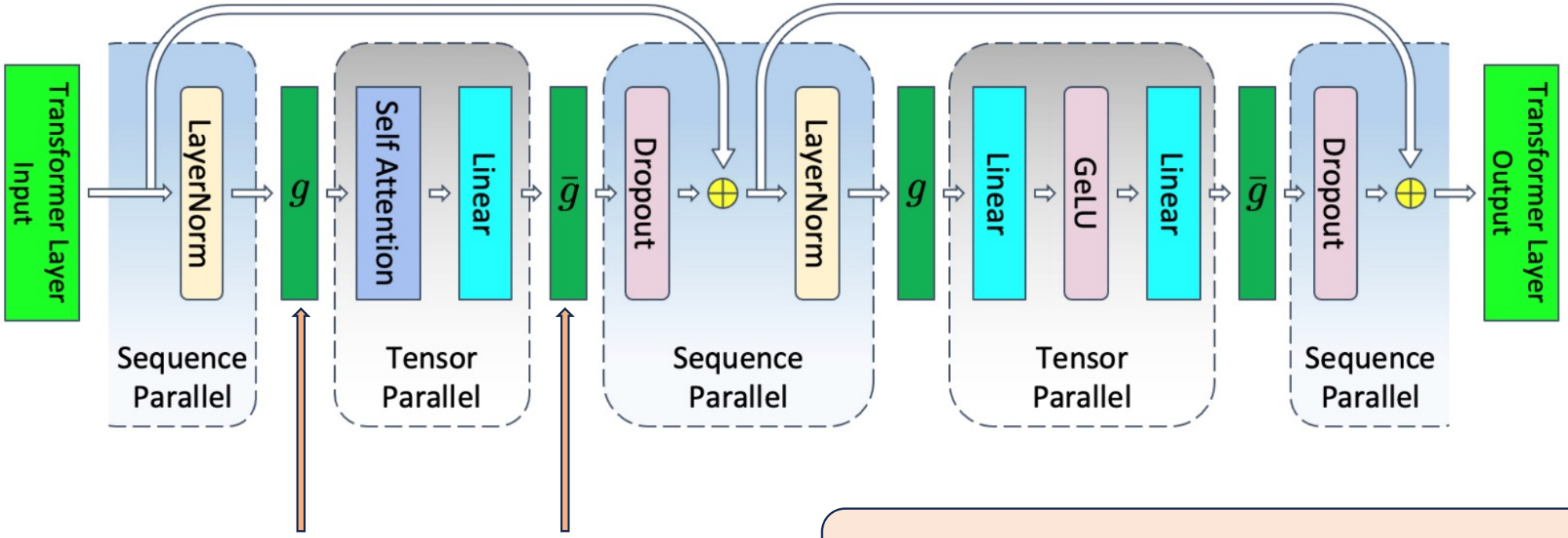
Finally: $5\text{bsh} + \frac{16bsh}{t}$ becomes $\frac{21bsh}{t}$

ReduceScatter to obtain the output fraction

AllGather to obtain the complete input

Sequence Parallelism

- Megatron-LM sequence parallelism
  - Attention (forward pass, backward pass omitted)



AllGather to obtain the complete input

ReduceScatter to obtain the output fraction

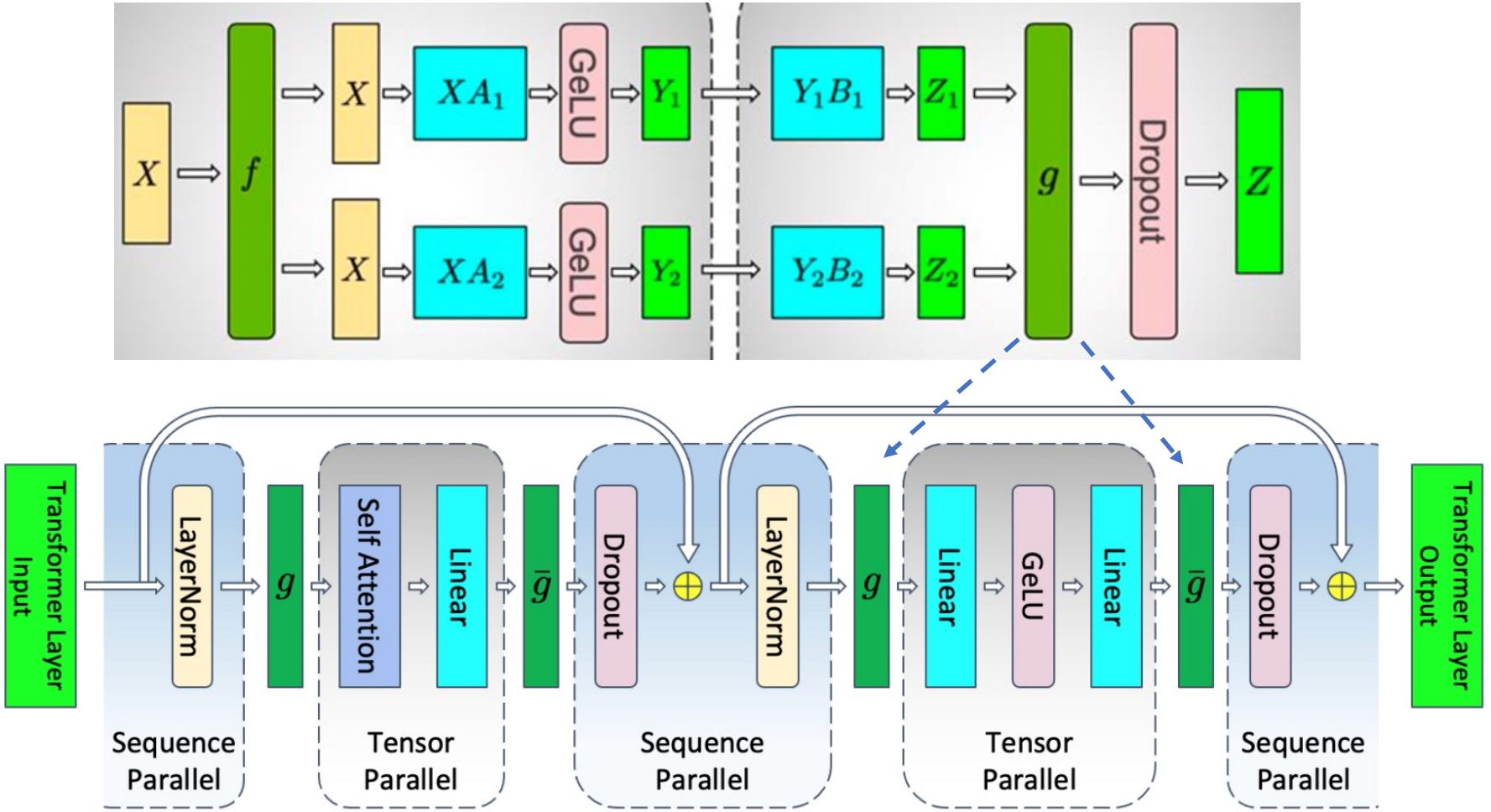
Finally: $5\text{bsh} + \frac{5bass+8bsh}{t}$ becomes $\frac{5bass+13bsh}{t}$

Sequence Parallelism

- Megatron-LM sequence parallelism
  - Slicing input/output of layernorm, and input of dropout
  - Changing AllReduce into ReduceScatter and AllGather
    - Almost **ZERO** extra communication overhead
  - Changing their memory consumption from $10bsh$ to $\frac{10bsh}{t}$
  - An example: batch_size = 64, hidden_size = 4096, seq_len = 32768, sp = 8
    - Memory consumption per-block changes from 86 GB to 10.7 GB

Sequence Parallelism

- **ZERO** extra communication overhead



Sequence Parallelism

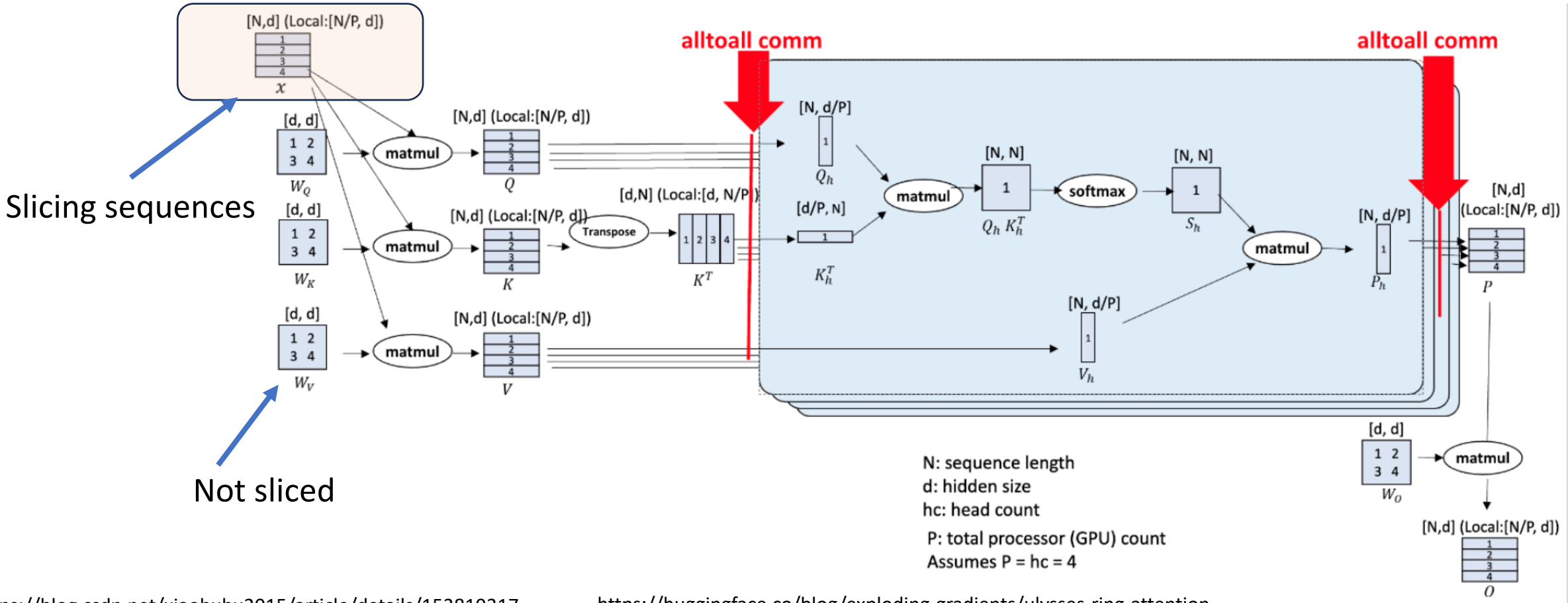
- DeepSpeed Sequence Parallelism
  - Megatron SP in fact slice activations at the token (sequence) dimension

How about slicing sequences in the very beginning?

Sequence Parallelism

Mode not partitioned!

- DeepSpeed Ulysses Sequence Parallelism

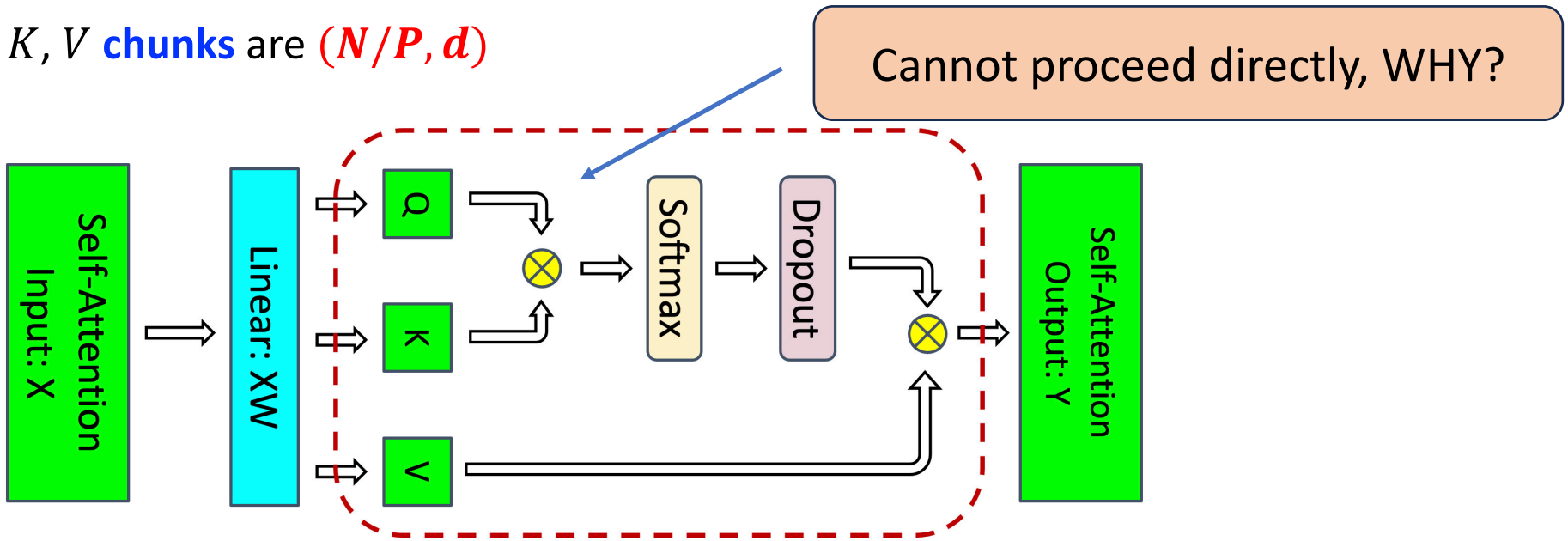


https://blog.csdn.net/xiaobubu2015/article/details/153819317

https://huggingface.co/blog/exploding-gradients/ulysses-ring-attention

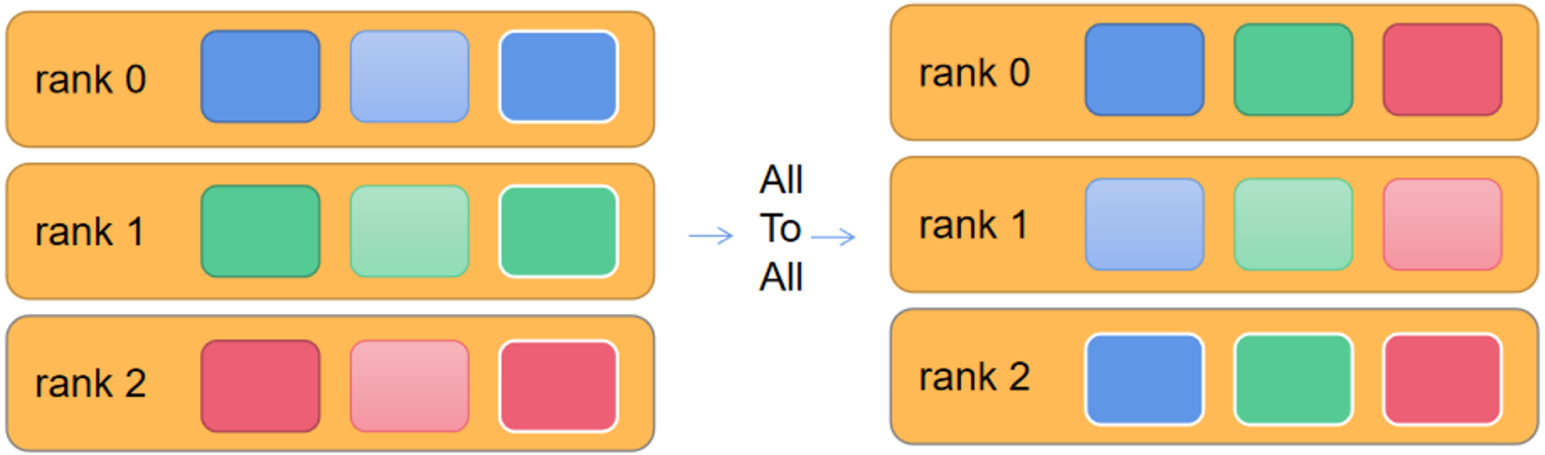
Sequence Parallelism

- DeepSpeed Ulysses Sequence Parallelism
  - Slicing input *X=(N, d)*, with *N* for sequence length and *d* for hidden dimension
    - Each GPU has an input ***(N/P, d)***, with *d* for # of GPUs
  - Attention weight matrices W_Q, W_K and $W_V \subseteq \mathbb{R}^{d \times d}$ not partitioned
    - Sizes of Q, K, V **chunks** are $(N/P, d)$



Sequence Parallelism

- DeepSpeed Ulysses Sequence Parallelism
  - All-to-All communication
    - Before all-to-all, each GPU holds the SEQUENCE CHUNK of **ALL HEADs** (i.e. $(N/P, d)$)
    - After all-to-all, each GPU holds ALL SEQUENCES on a specific HEAD (i.e. $(N, d/P)$)

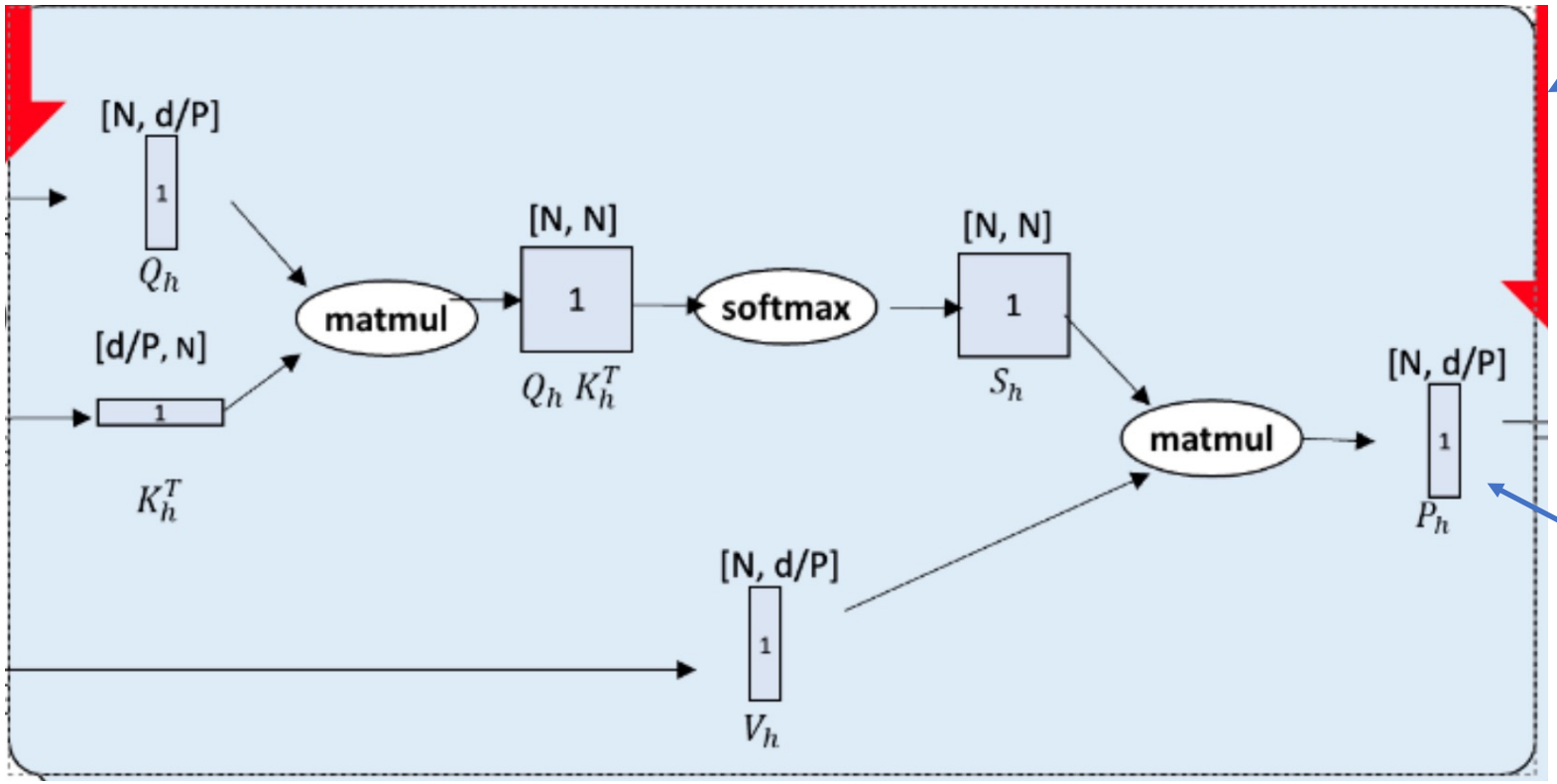


Looks like a transpose operation!

https://mdnice.com/writing/974038f2cc3d46239377b0928e8b9eea

Sequence Parallelism

- DeepSpeed Ulysses Sequence Parallelism
  - All-to-All communication

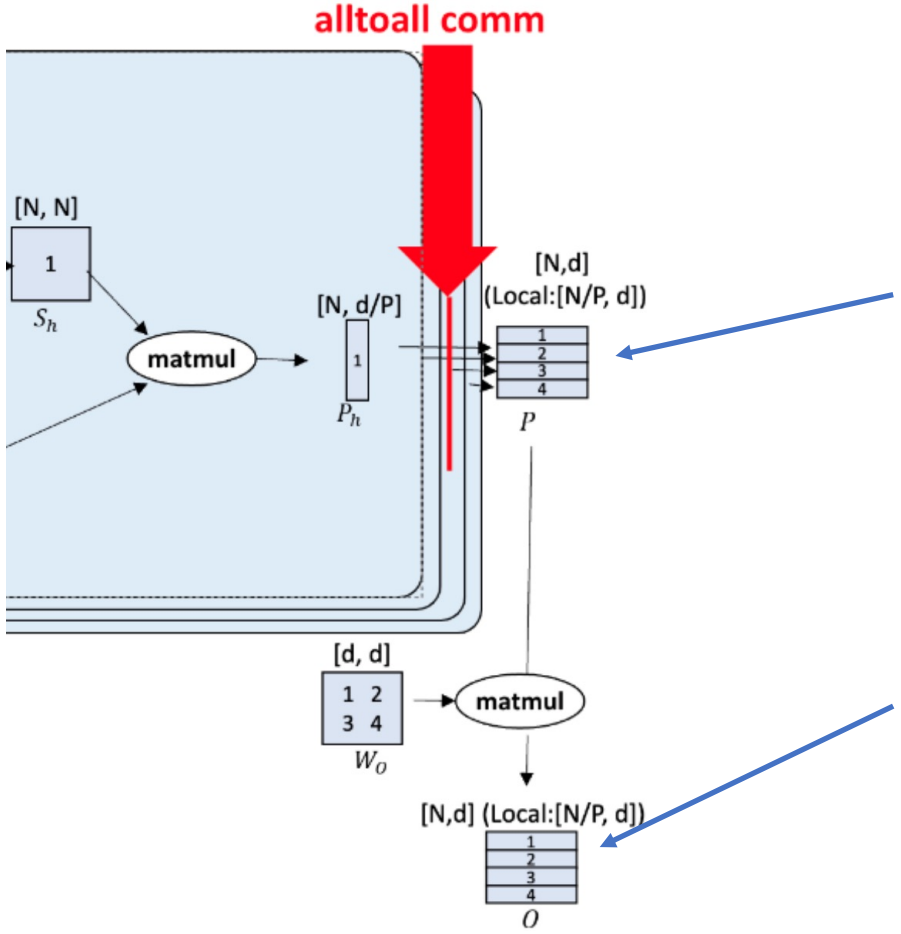


Multi-head Attention Computation

Output dimension: $(\boldsymbol{N}, \boldsymbol{d/P})$

Sequence Parallelism

- DeepSpeed Ulysses Sequence Parallelism
- MLP can compute at each sequence chunk granularity
- DeepSpeed Ulysses backward pass



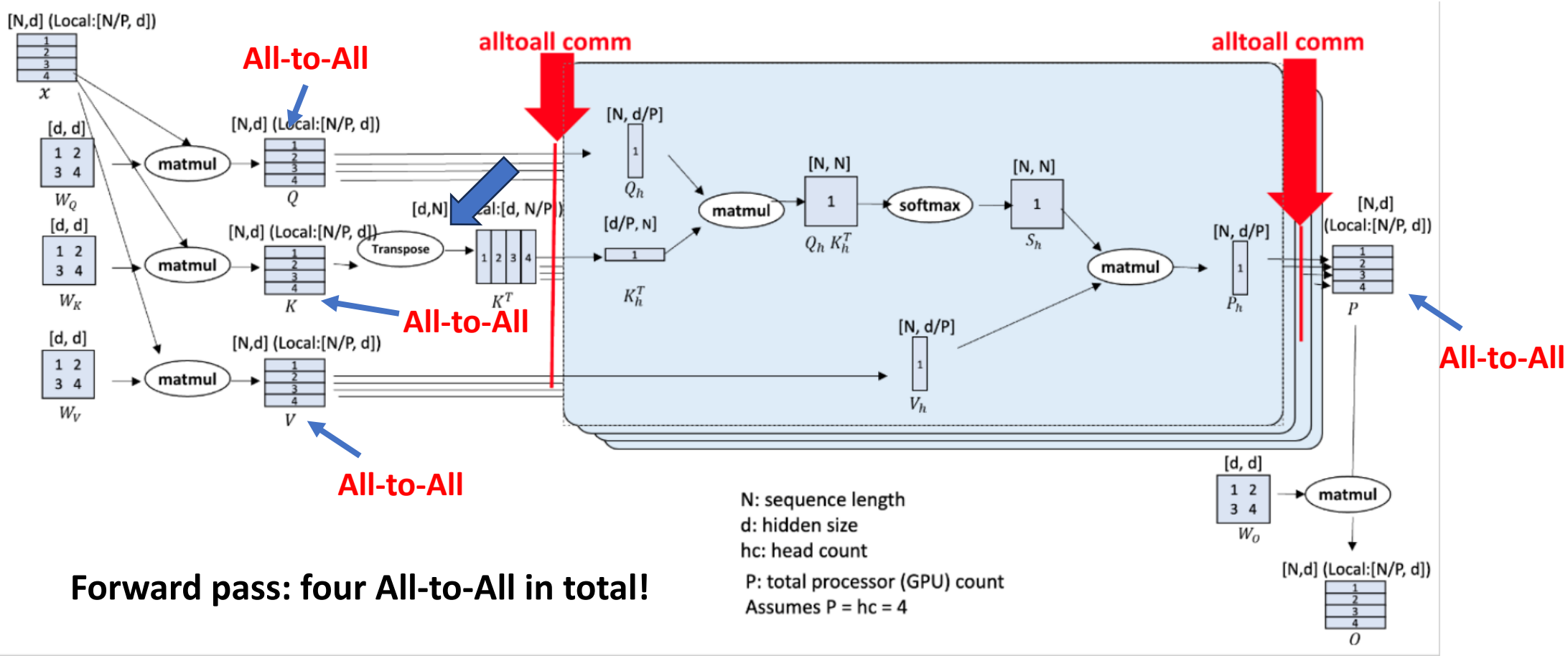
Output dimension: $\left(\frac{N}{P}, d\right)$

after ALL-to-ALL

W_O dimension: $\left(\frac{N}{P}, d\right)$

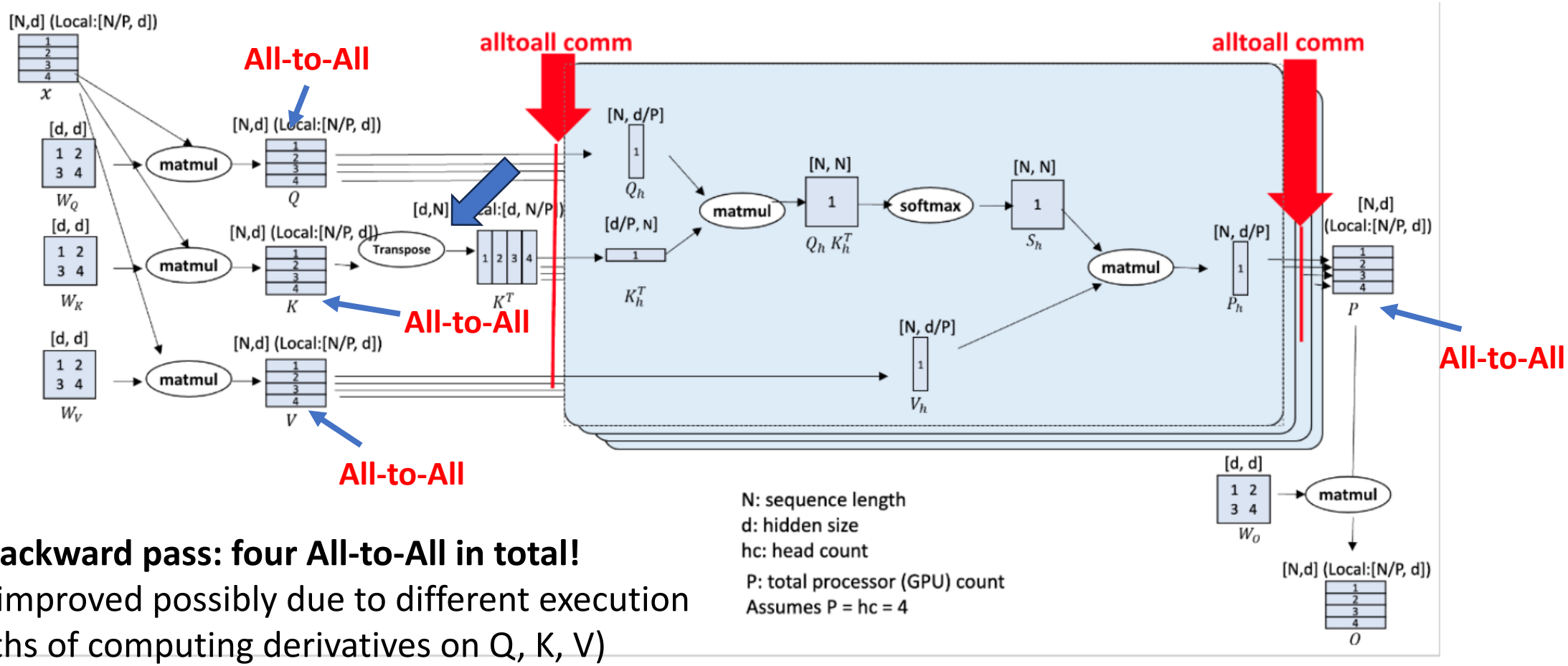
Sequence Parallelism

- DeepSpeed Ulysses Sequence Parallelism: Communication Analysis



Sequence Parallelism

- DeepSpeed Ulysses Sequence Parallelism: Communication Analysis



Backward pass: four All-to-All in total!

(Can be improved possibly due to different execution paths of computing derivatives on Q, K, V)

Sequence Parallelism

- DeepSpeed Ulysses versus NVIDIA Megatron
  - Megatron SP **splits attention and MLP layer parameters** by using tensor parallelism, and splits layernorm and dropout inputs accordingly. The computations are carried out on the heads at different GPUs.
  - Ulysses splits attention and MLP by segmenting the input sequences, and the computations are carried out on the heads at different GPUs. **No segmentation** on the attention layer parameters.

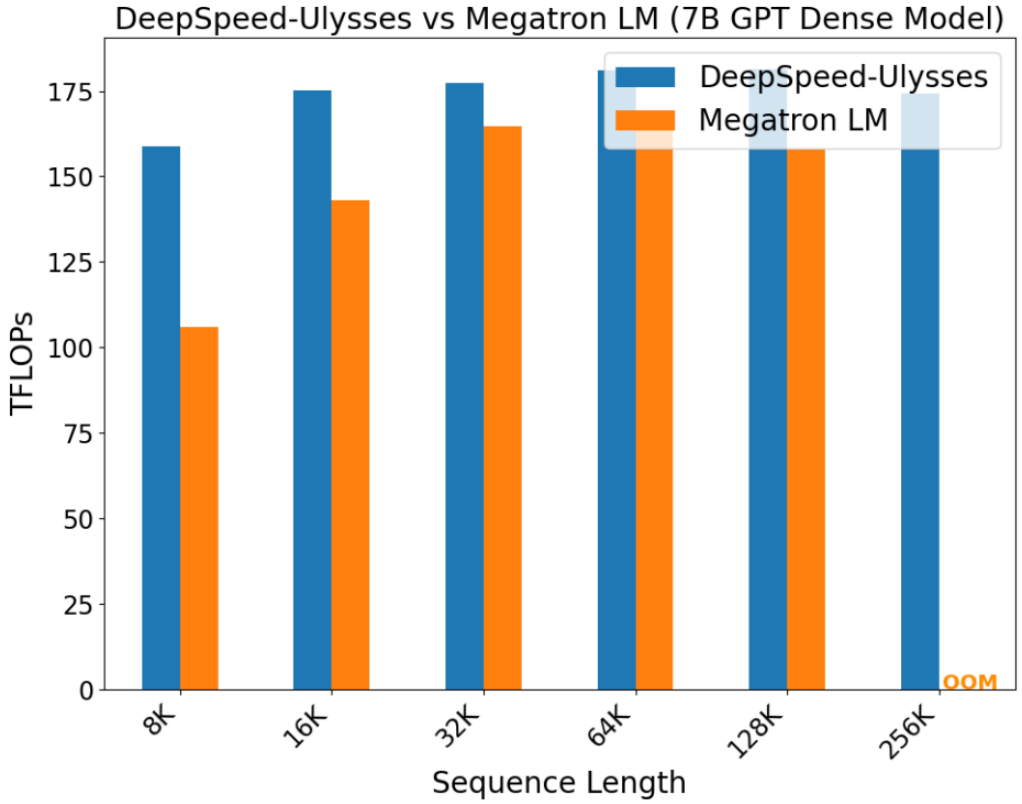
Sequence Parallelism

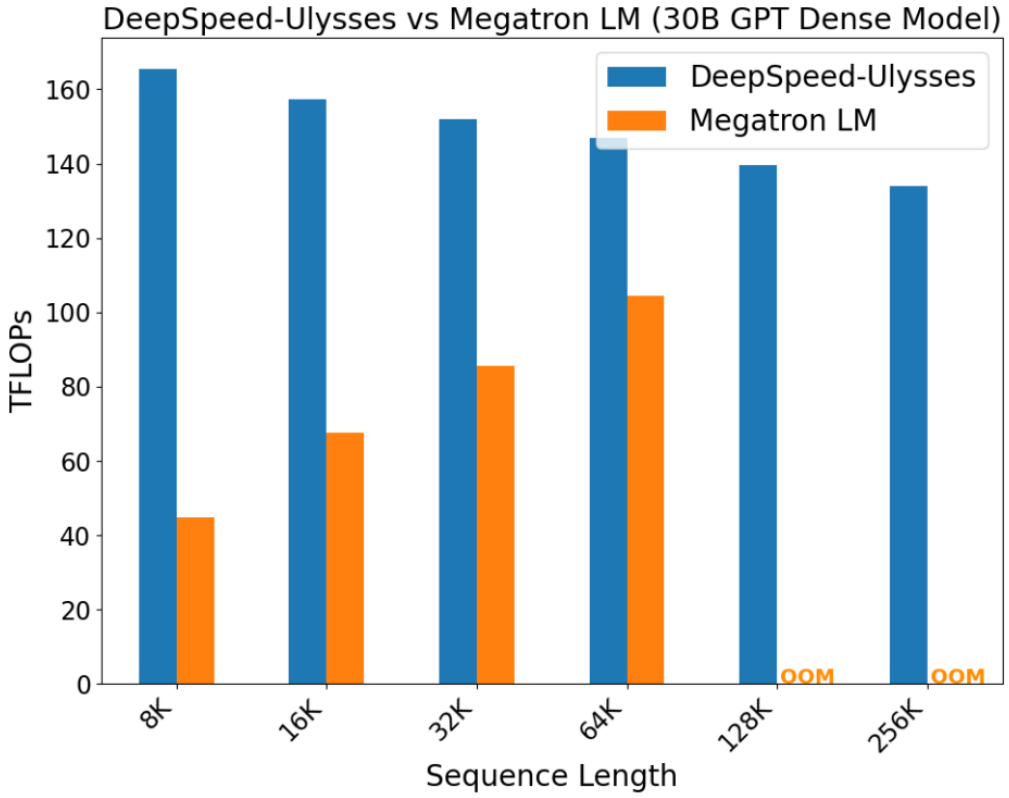
- DeepSpeed Ulysses versus NVIDIA Megatron

	Attention	**MLP**
Megatron-LM (combined with TP)	***4 N d*** Forward: ReduceScatter + AllGather, Backward: ReduceScatter + AllGather	***4 N d*** Forward: ReduceScatter + AllGather, Backward: ReduceScatter + AllGather
DeepSpeed Ulysses	***8 N d/P*** Forward: 4 All-to-All, Backward: 4 All-to-All	***0*** MLP is element-wise, naturally fitting SP

Sequence Parallelism

- DeepSpeed Ulysses versus NVIDIA Megatron





Sequence Parallelism

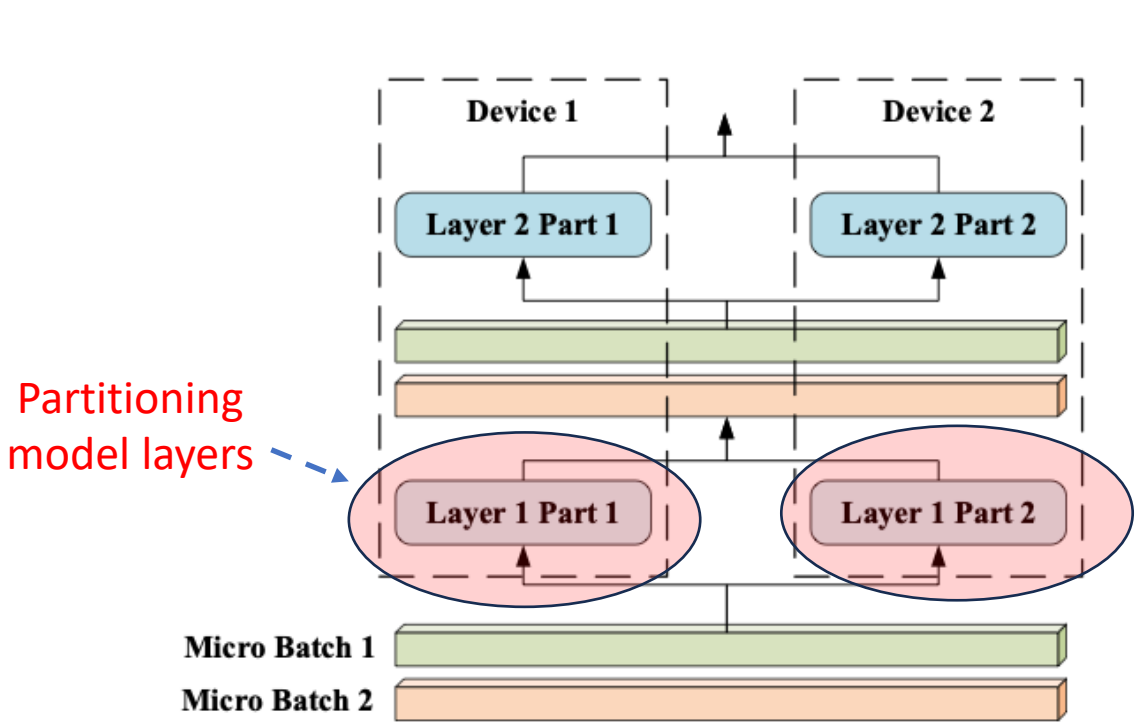
- DeepSpeed Ulysses versus NVIDIA Megatron
  - N = 1M tokens, d = 4096, P = 128, FP16
  - Megatron: $8 \times N \times d = 8 \times 1e6 \times 4096 \times 2 = 65.536$ GB
  - Ulysses: $8 \times N \times d / P = 65.536$ GB / 128 = 0.512 GB

"Ulysses reduces communication volume by eliminating All-Gather in MLP and using All-to-All only in attention, achieving near-linear scaling for sequence lengths up to 1M."

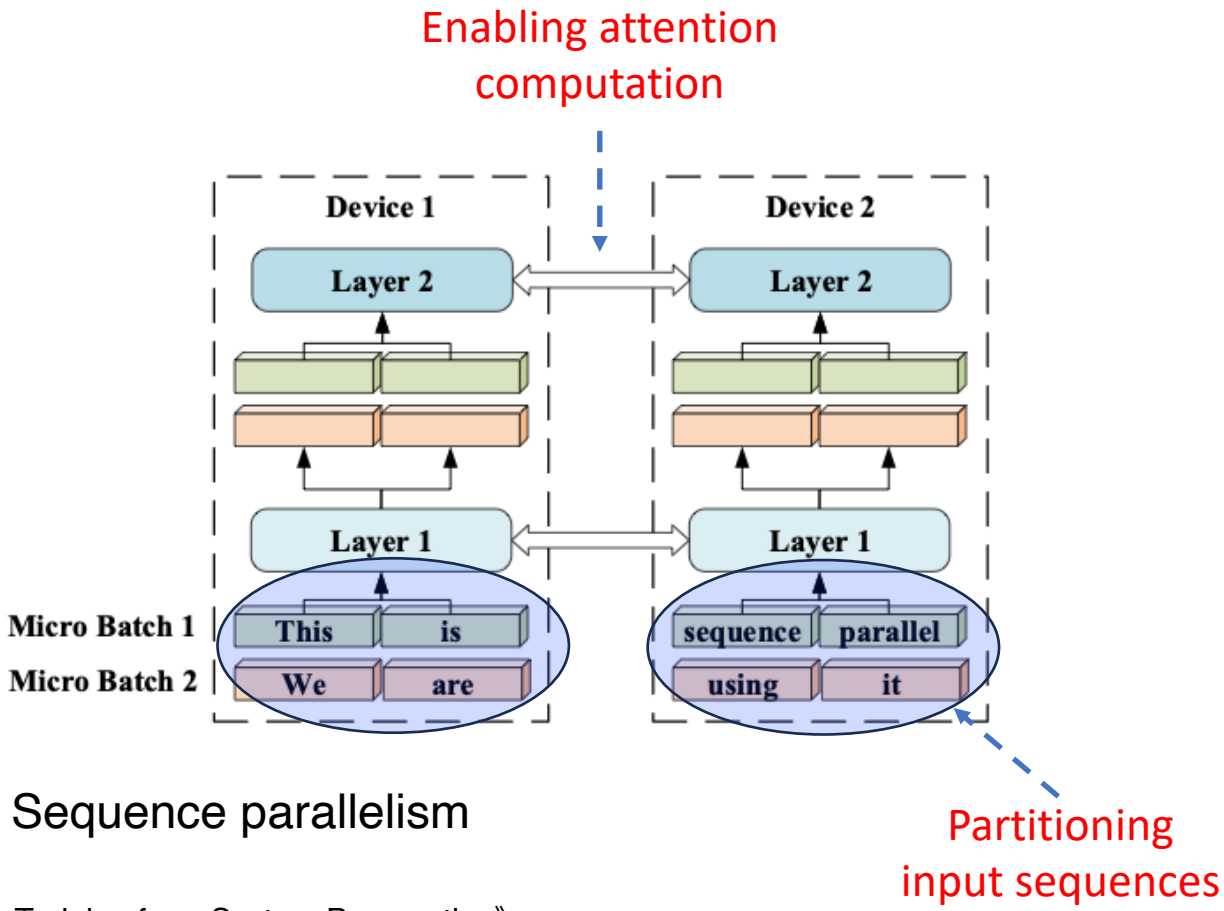
"Megatron-LM sequence parallelism incurs a communication volume per link of 4Nh which is P times larger than that for DeepSpeed sequence parallelism."

Sequence Parallelism

- Colossal AI Sequence Parallelism



Tensor parallelism



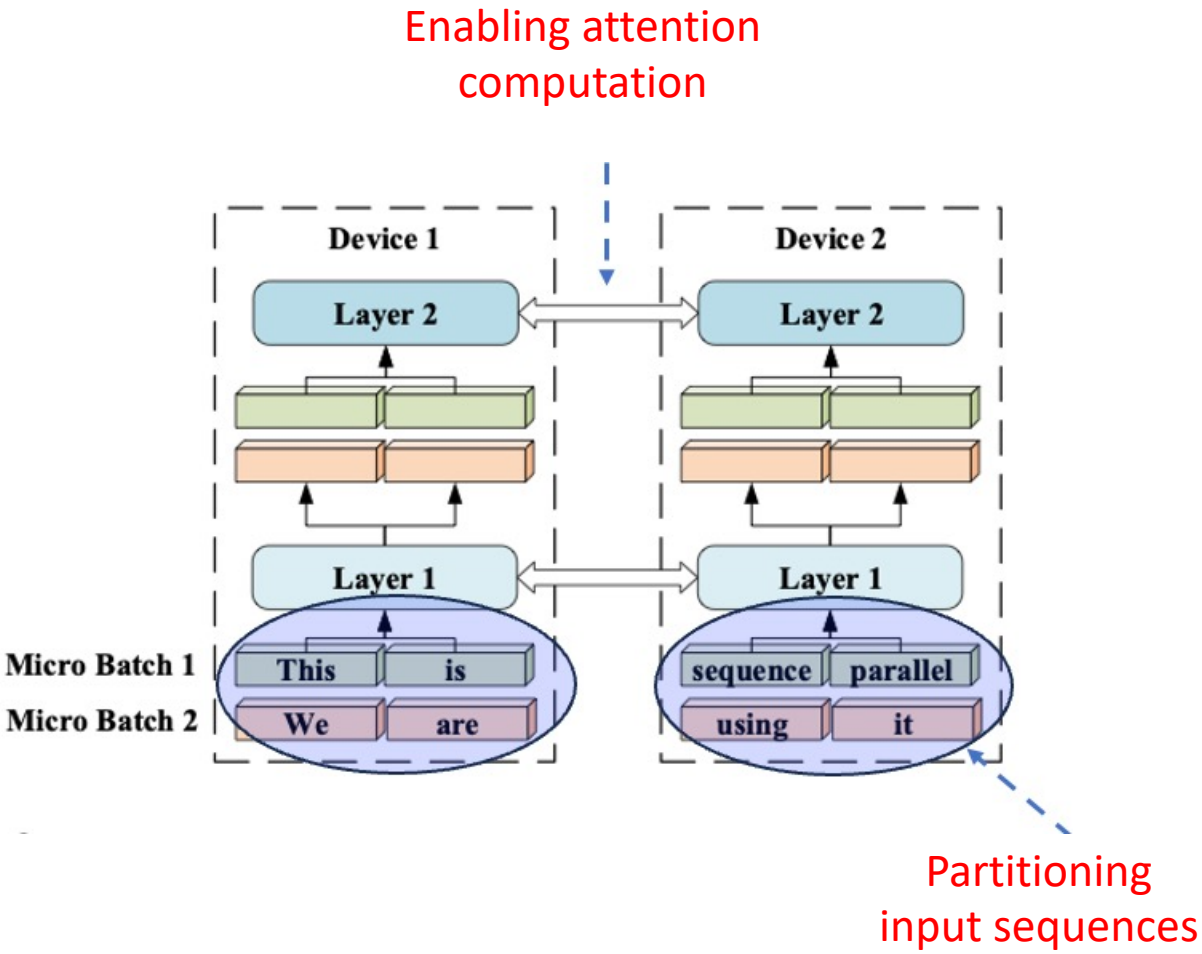
Sequence parallelism

《Sequence Parallelism: Long Sequence Training from System Perspective》

Sequence Parallelism

- Colossal AI Sequence Parallelism
  - Q, K, V partitioned on the **sequence**
  - Computing attention scores need full Q, K, V on all tokens, in which communication is inevitable

- **Ring Attention** adopted in Colossal-AI



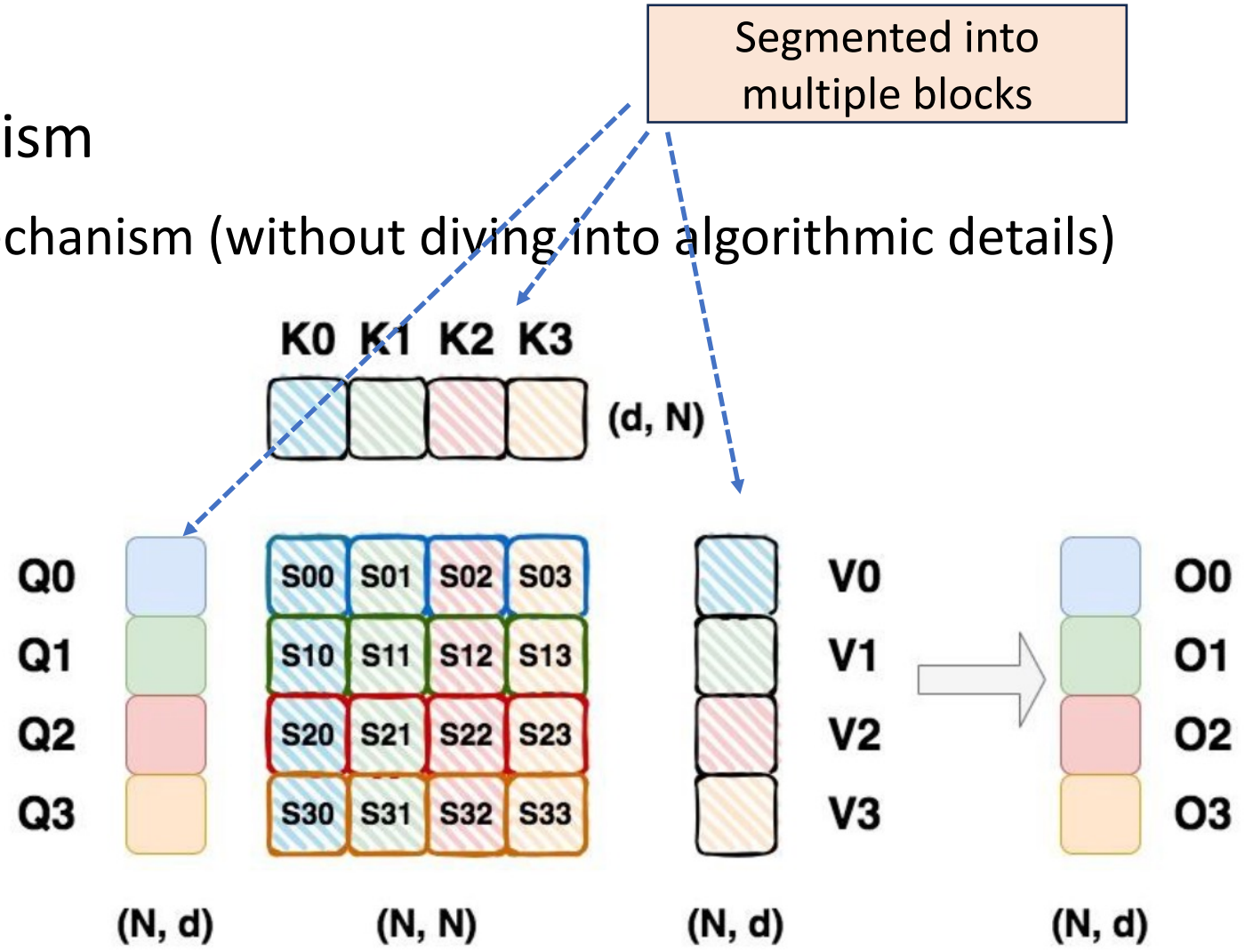
Sequence Parallelism

- Colossal AI Sequence Parallelism
  - Convoluted with Attention mechanism (without diving into algorithmic details)

Step 1: for each Q_i, compute attention score by feeding K_j and V_j, and obtain O_{ij}

Step 2: for each K_j and V_j, repeat Step 1 and update O_{ij}, and eventually obtain O_i

Step 3: change Q_i to Q_{i+1}, repeat Step 1 and Step 2 until Q_C

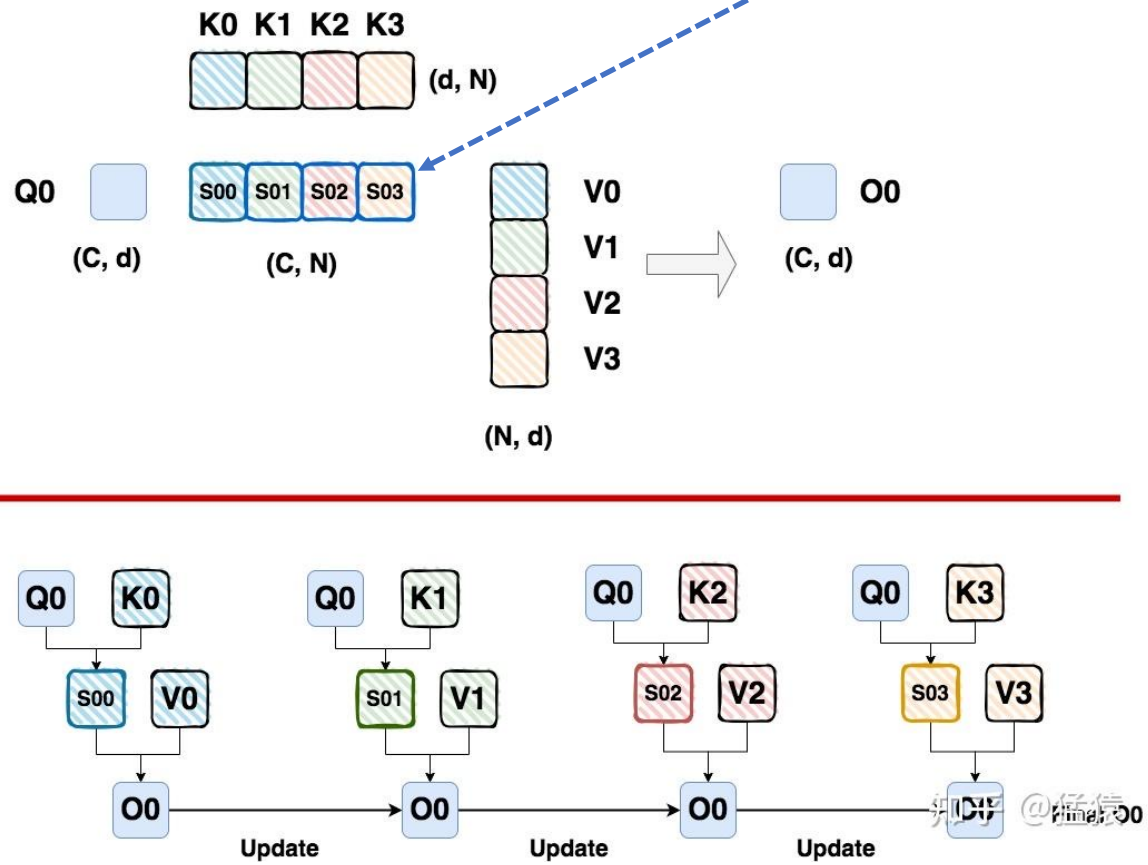


From https://zhuanlan.zhihu.com/p/4963530231

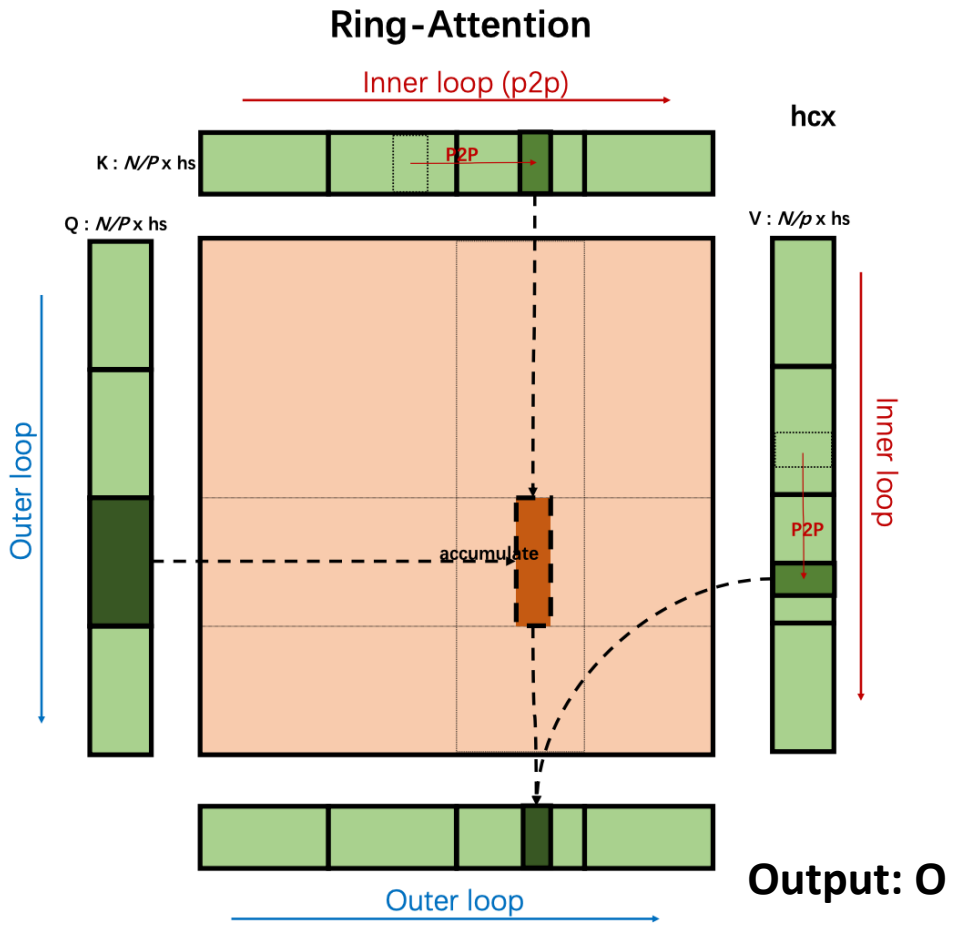
Sequence Parallelism

Mathematical deduction shows that S_{00}, S_{01}, S_{02}, and S_{03} can be computed in an arbitrary order!

- Colossal AI Sequence Parallelism



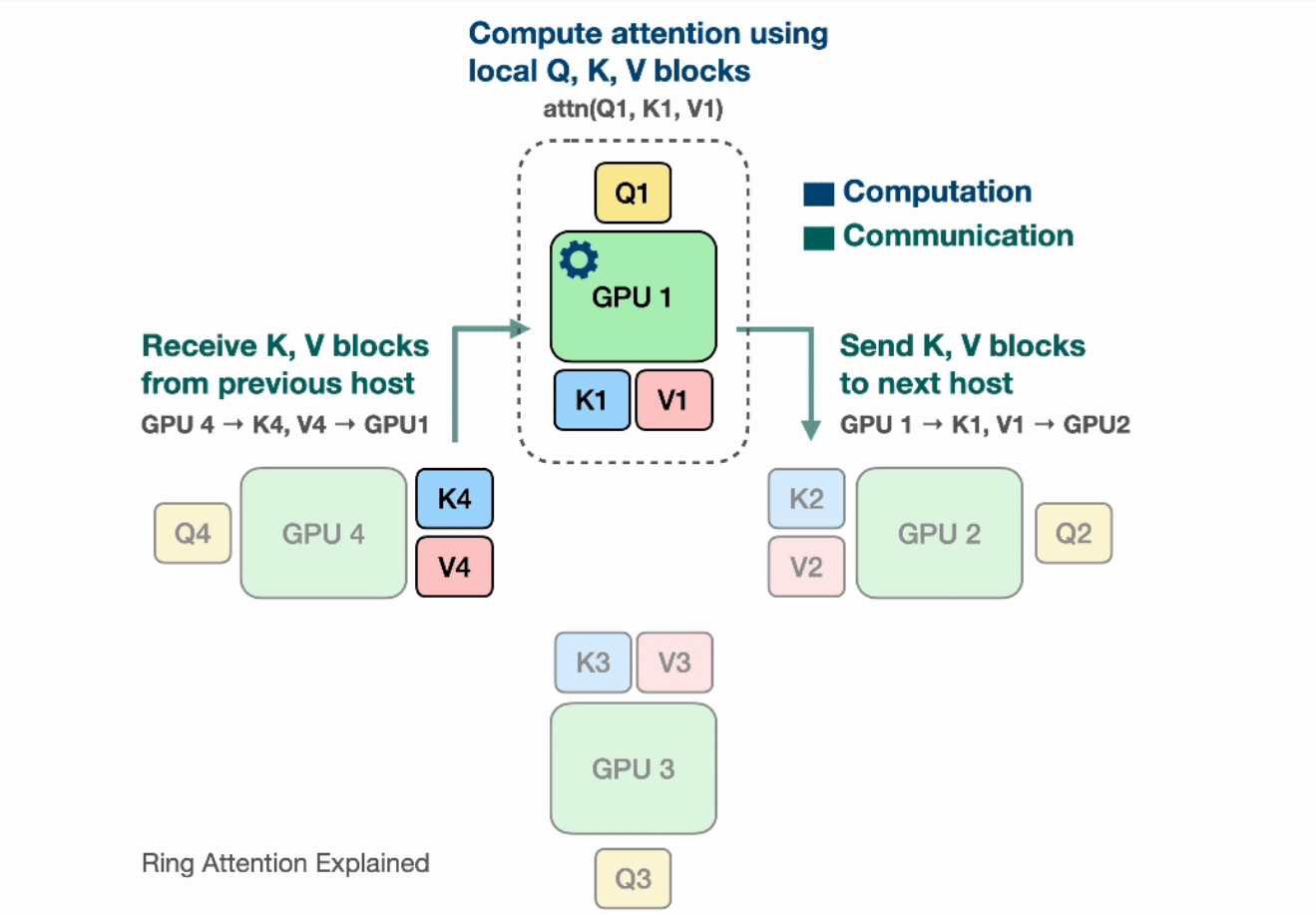
https://www.themoonlight.io/en/review/usp-a-unified-sequence-parallelism-approach-for-long-context-generative-ai



From https://zhuanlan.zhihu.com/p/496530231

Sequence Parallelism

- Colossal AI Sequence Parallelism



Step 1: *GPU1 computes O1 using Q1, K1 and V1 (other GPUs do the same thing)*

Step 2: *GPU1 sends K1 and V1 to GPU2, and receives K4 and V4 from GPU4; compute and update O1 using Q1, K4 and V4*

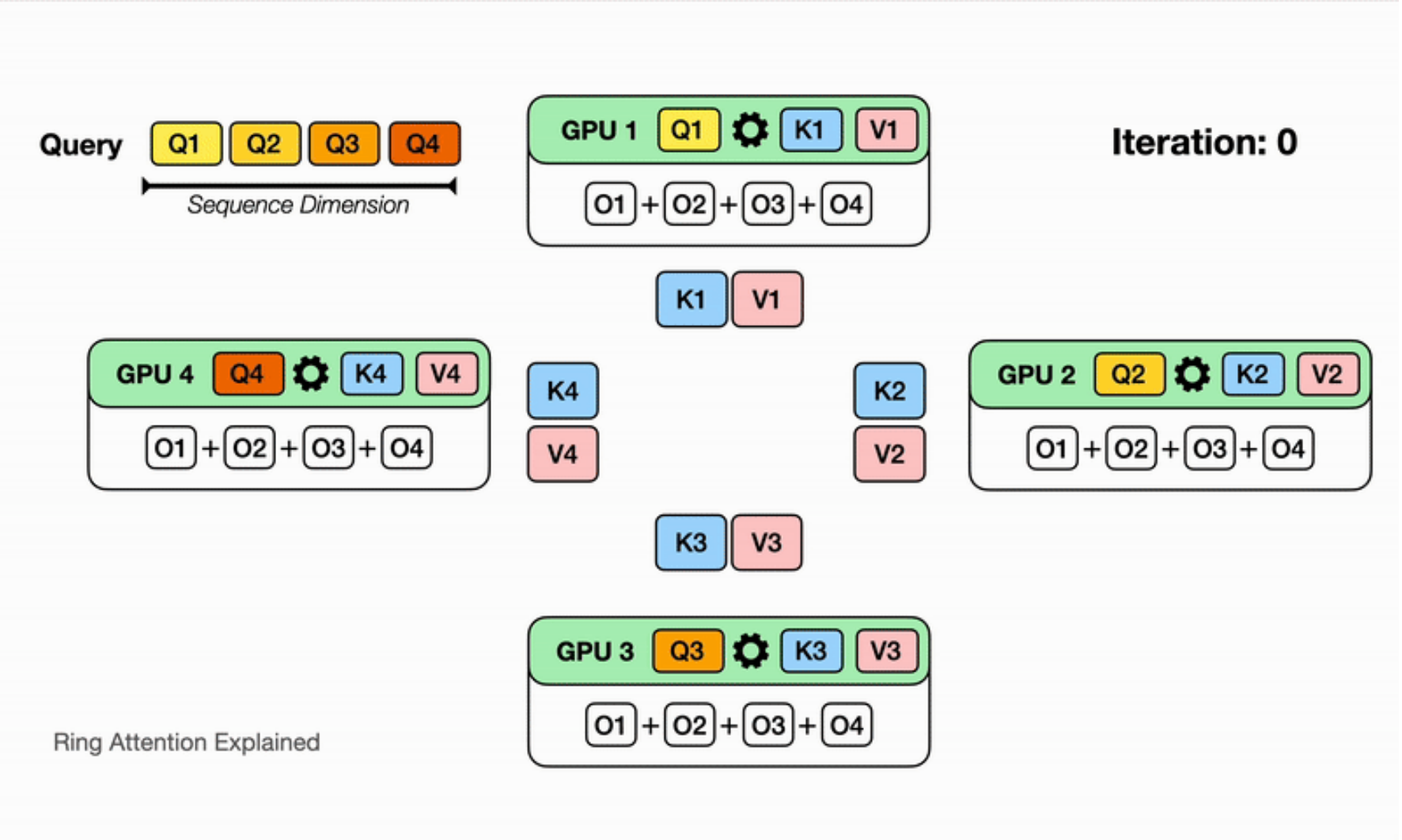
Step 3: *GPU1 sends K4 and V4 to GPU2, and receives K3 and V3 from GPU4; compute and update O1 using Q1, K3 and V3*

Step 4: *GPU1 sends K3 and V3 to GPU2, and receives K2 and V2 from GPU4; compute and update O1 using Q1, K2 and V2*

The network communications are illustrated by the blinking arrows.

From https://coconut-mode.com/posts/ring-attention/

Sequence Parallelism



Illustrate for N devices, it will take N iterations to finish the whole output comptutation. Watch for each iteration on each device, different partial sum of the output is computed with the K, V block it currently has, and it eventually sees all the K, V blocks and has all the partial sum for the output!

From https://coconut-mode.com/posts/ring-attention/

Sequence Parallelism

d: hidden dimension
c: sequence length after slicing

- Colossal AI Sequence Parallelism
  - Memory Complexity
    - Storing current K and V blocks (in FP16) needs $2dc$ floats or $4dc$ Bytes
    - Receiving new K and V blocks needs $2dc$ floats or $4dc$ Bytes
    - Storing Q_i block needs $2dc$ Bytes
    - Computing the output needs at least $2dc$ Bytes
    - In total at least $12dc$ Bytes, where c is the chunk size
  - Communication Complexity
    - Each step sends $2dc$ floats or $4dc$ Bytes, and the total number of rotations is around $\frac{s}{c}$
    - At each inner loop a GPU sends $4sd$ Bytes, and the outer loop has p rounds
    - Total communication load is around $O(sdp)$, which is much higher than Megatron and Ulysses

Sequence Parallelism

- Recap
  - Megatron CP: sharding weights and some inputs on the hidden dimension
  - Ulysses CP: sharding input token sequence
  - Ring Attention CP: letting KV blocks flow on the ring

Thanks!