CSCI 544: Applied Natural Language Processing

Advanced Techniques in Large-Scale Pretraining

Xuezhe Ma (Max)

USC University of Southern California

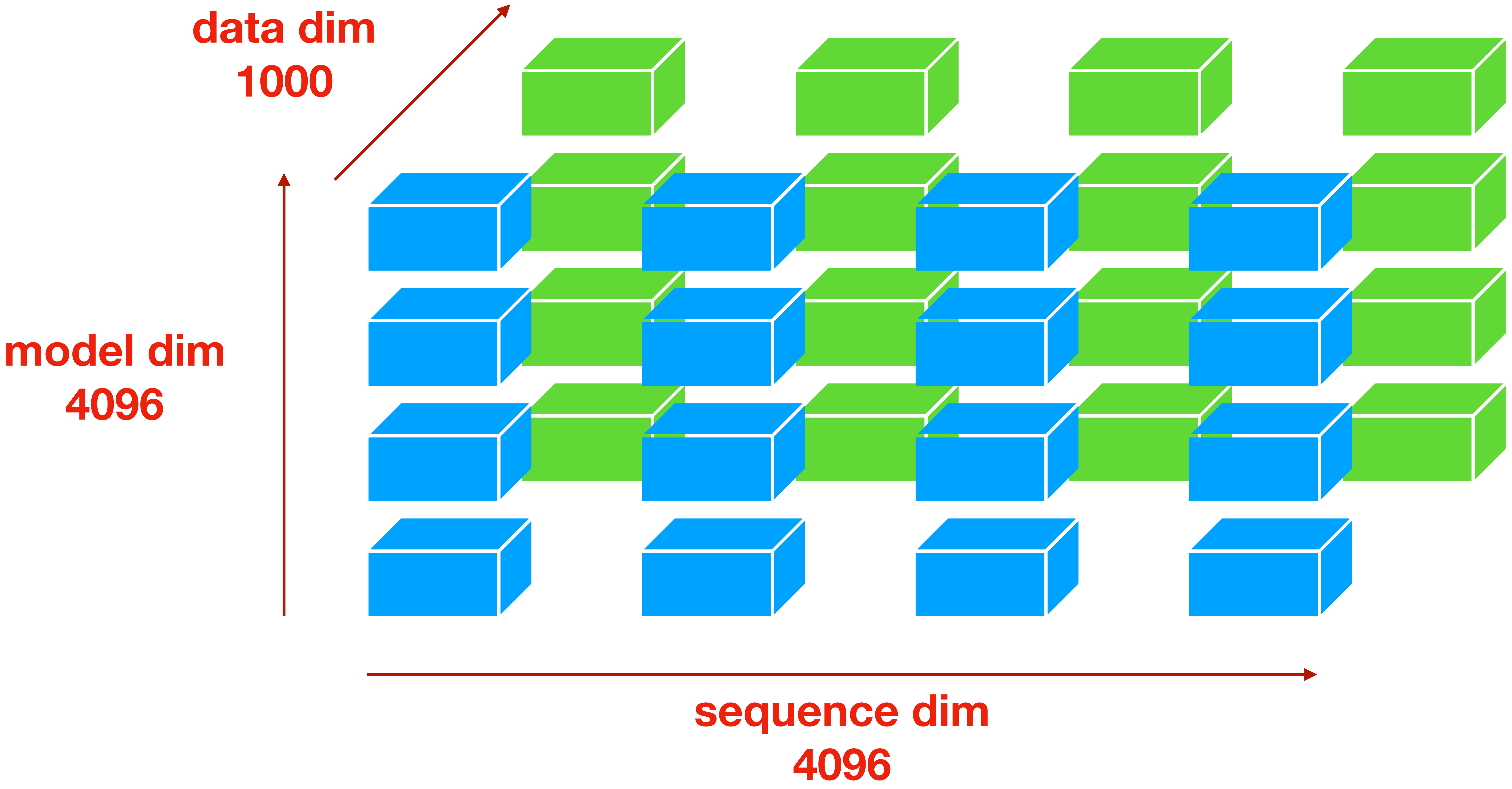
Large-Scale Pretraining

- **Model Architecture**
  - Transformer
- **Data**
  - Publicly available data
    - Common Crawl
    - Wikipedia
    - Books
    - …
- **Training Objective**
  - Autoregressive language modeling (next-token prediction)
- **Machines**
  - Sufficient amount of GPUs
    - A100 (80G memory per device)

Training Receipt

- **7B parameters**
  - 32 blocks of Transformer decoder
  - Model dimension size $d = 4096$
- **Training sequence length**
  - 4096 tokens
- **Training batch size**
  - 4 million tokens (1,000 sequences)

Challenges in Large-Scale Pretraining



16 billion elements (64G memory w. float32)

distributing the large tensor to multiple GPUs!

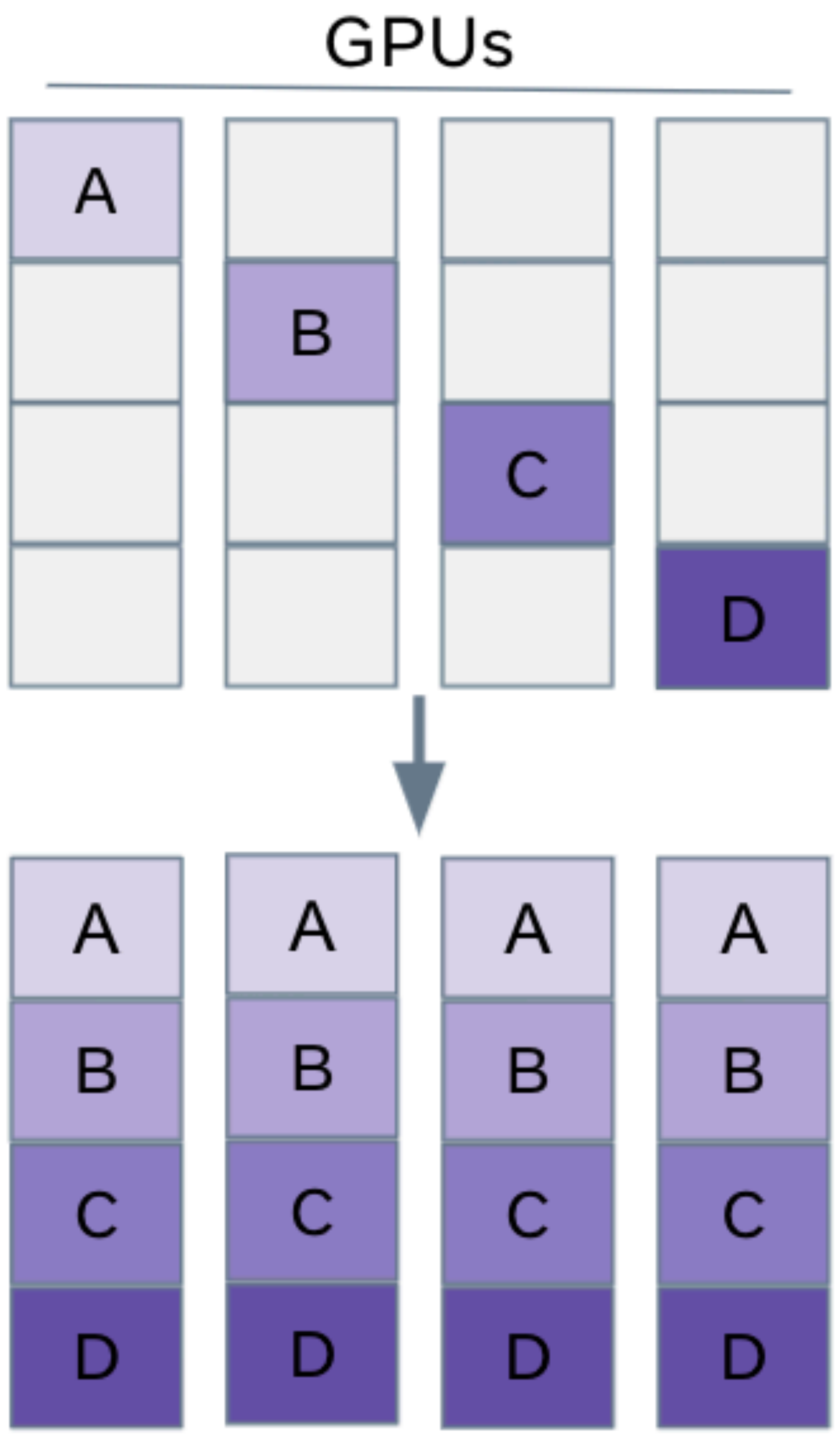
Communication between GPUs

- For distributed training, we need to synchronize training status
  - Model parameters
  - Gradients
  - Optimization states

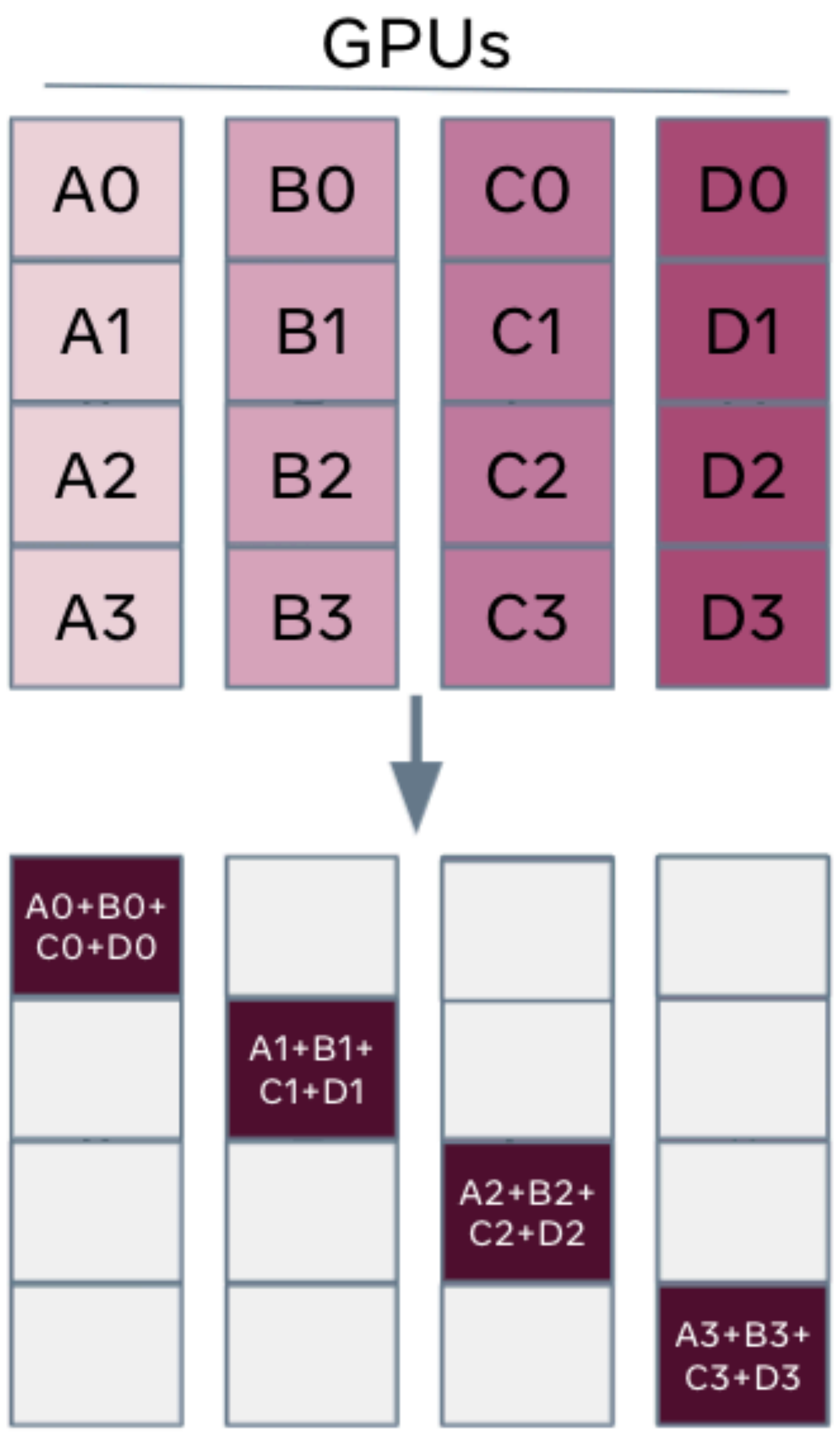
Communication between GPUs

- Some basic communication operations

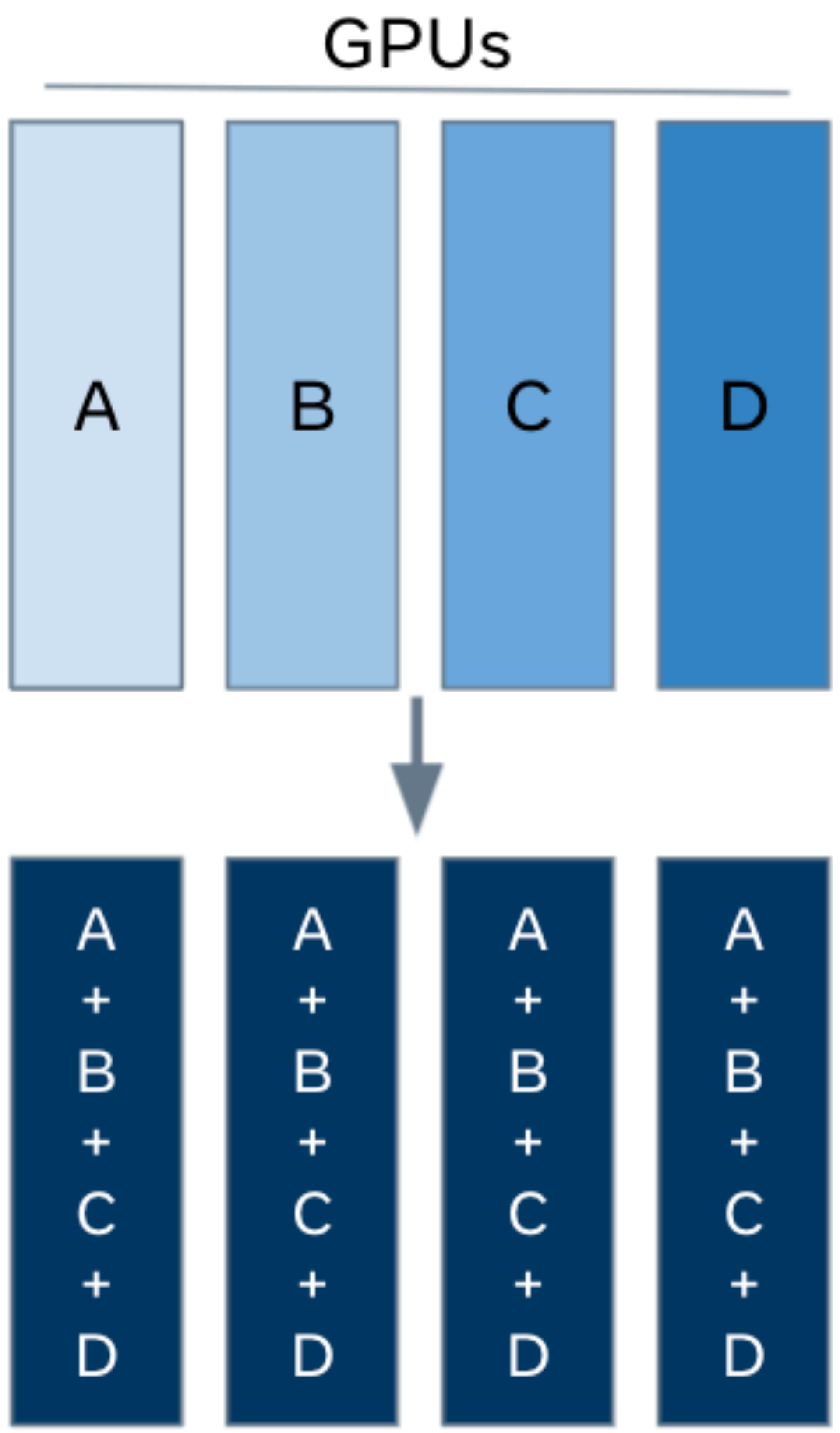
All-gather



Reduce-scatter

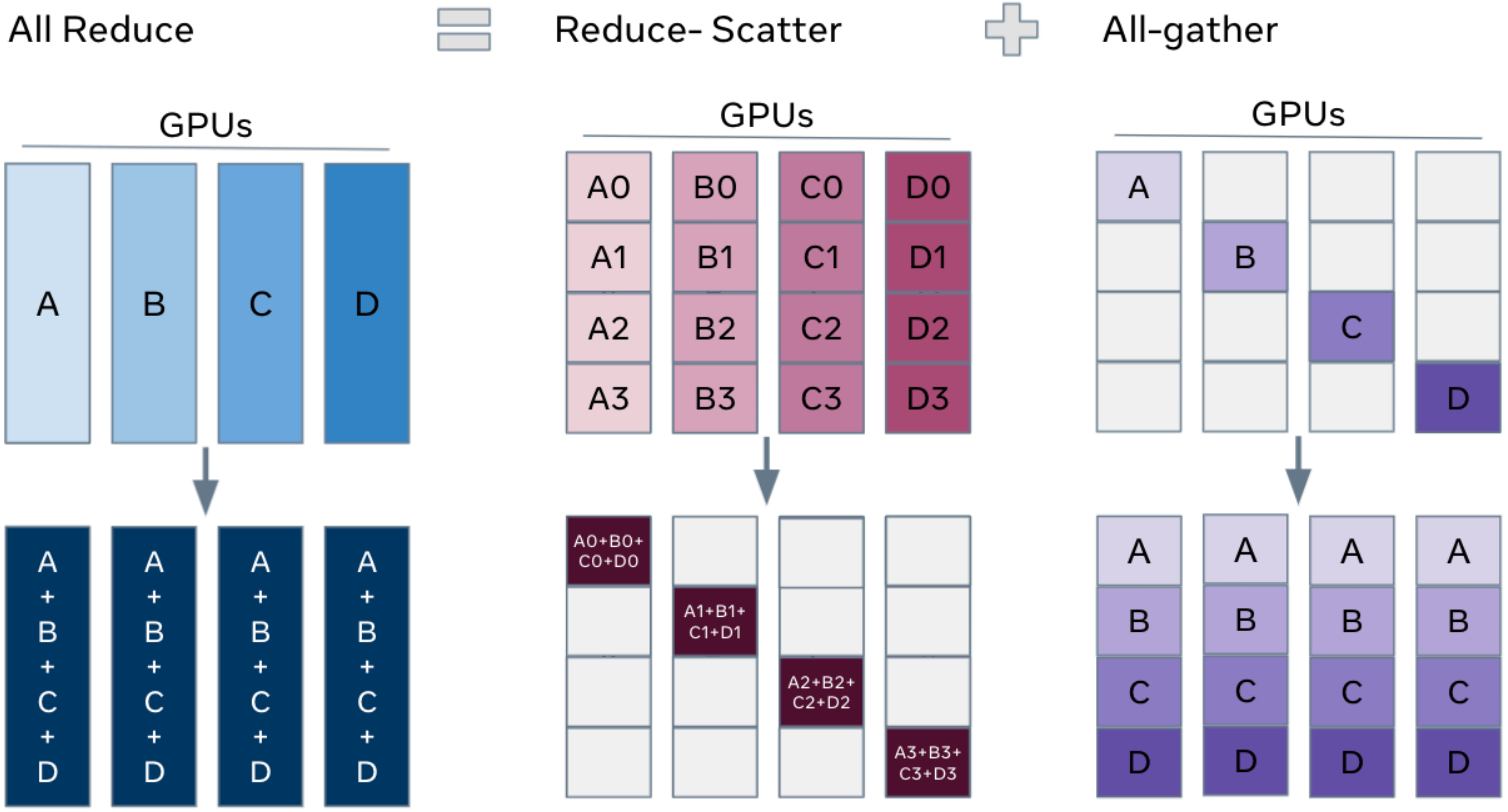


All-reduce



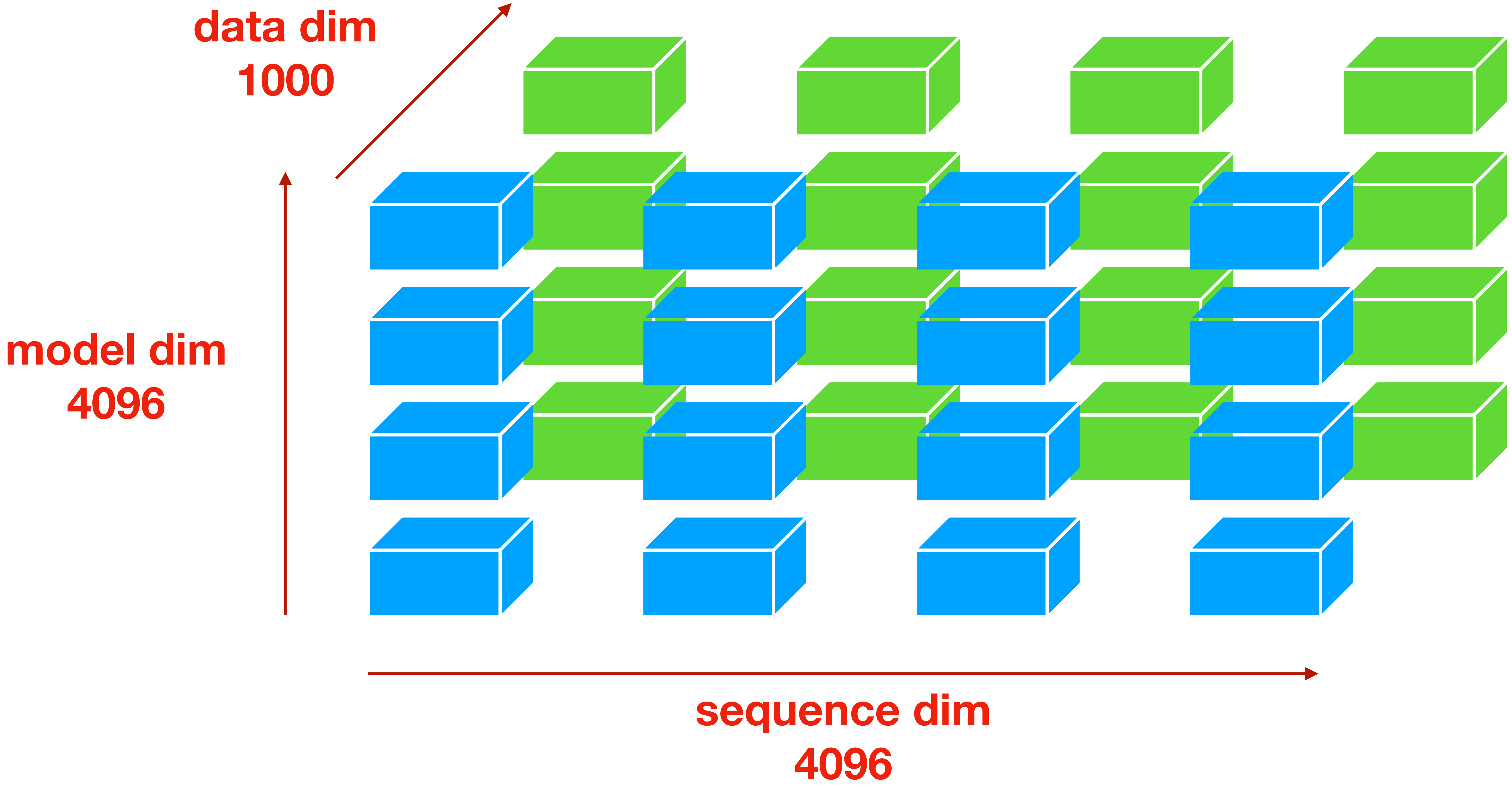
Communication between GPUs

- Some basic communication operations



Distributed Large-Scale Pretraining

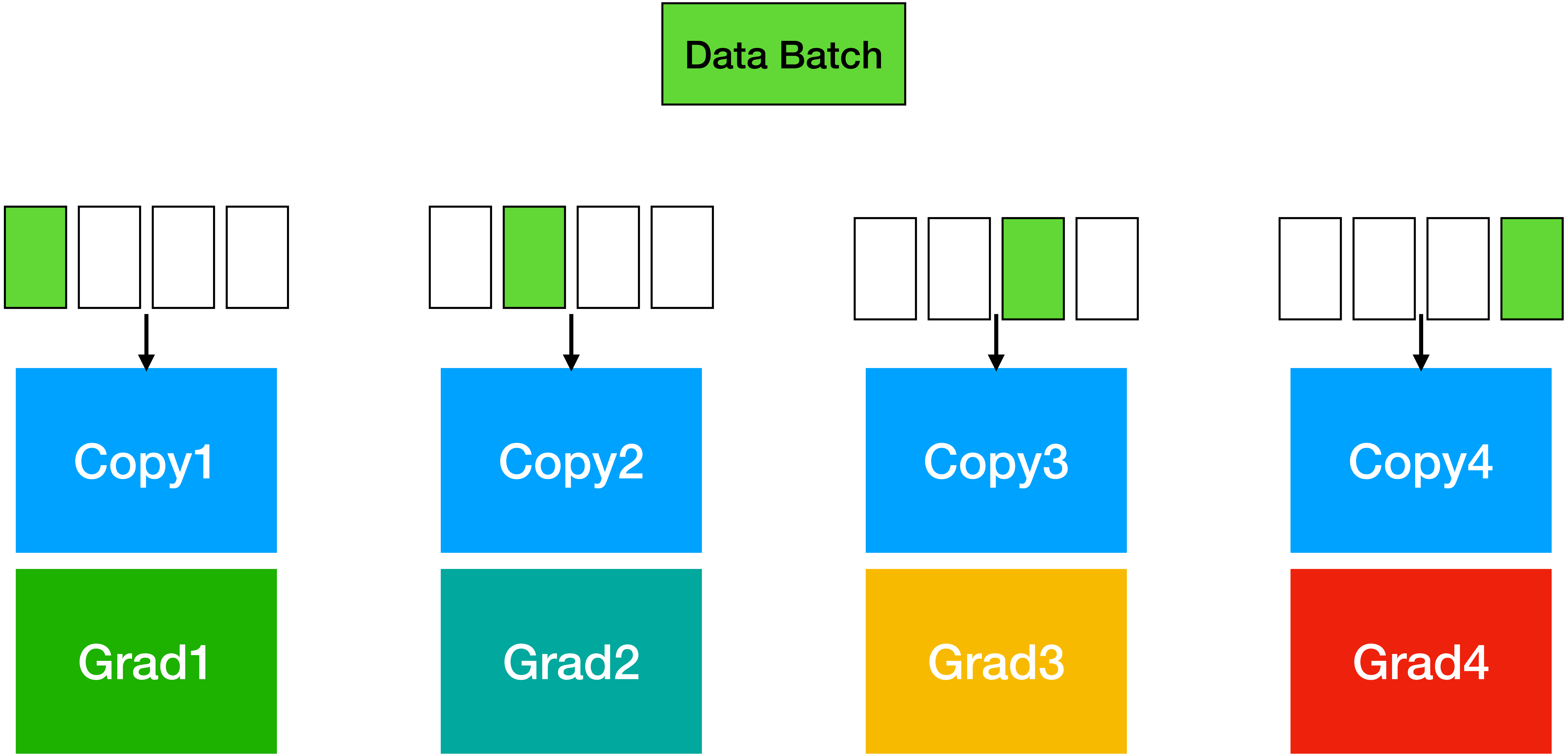
- **Three Criteria**
  - Minimal redundant computation
  - Minimal peak memory cost
  - Minimal communication overhead



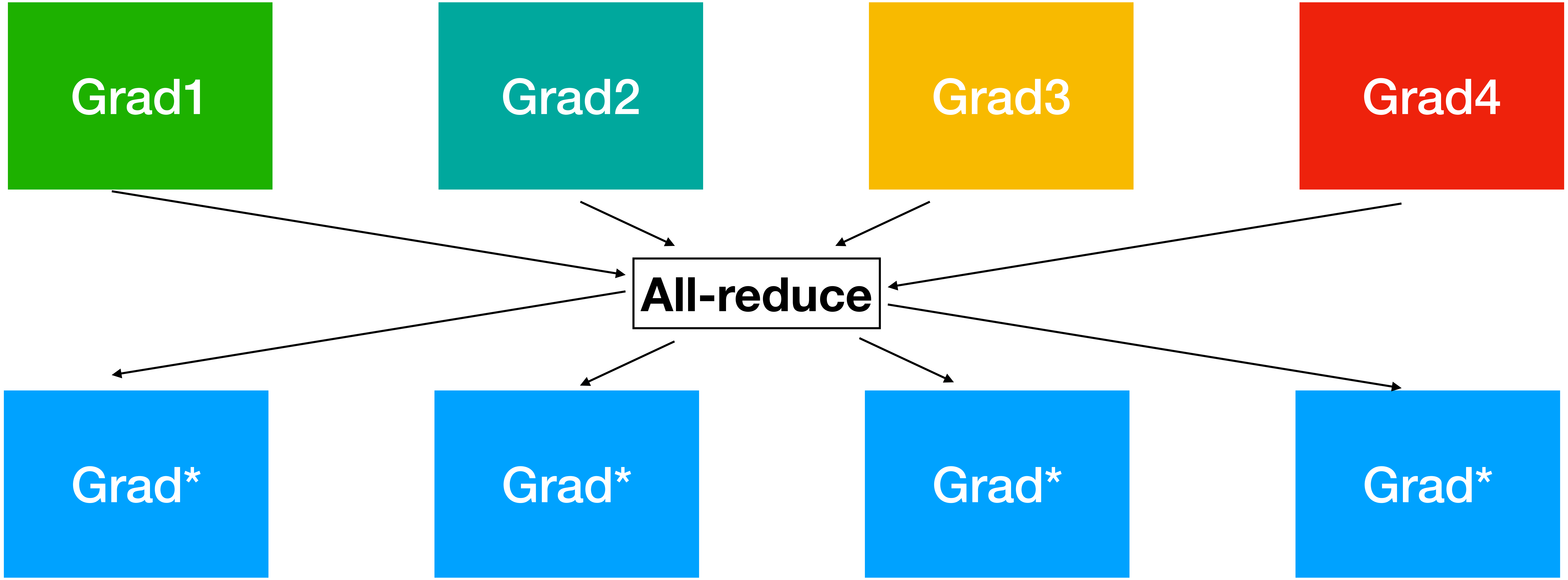
Distributed Data Parallel (DDP)

- **Each GPU keeps a complete copy of the model**
- **Each GPU process a subset of a training batch**
  - No data overlaps between GPUs
- **Synchronize parameter gradients after each forward-backward pass**

Distributed Data Parallel (DDP)



Distributed Data Parallel (DDP)



$$L = \frac{1}{4}(L_1 + L_2 + L_3 + L_4)$$

Distributed Data Parallel (DDP)

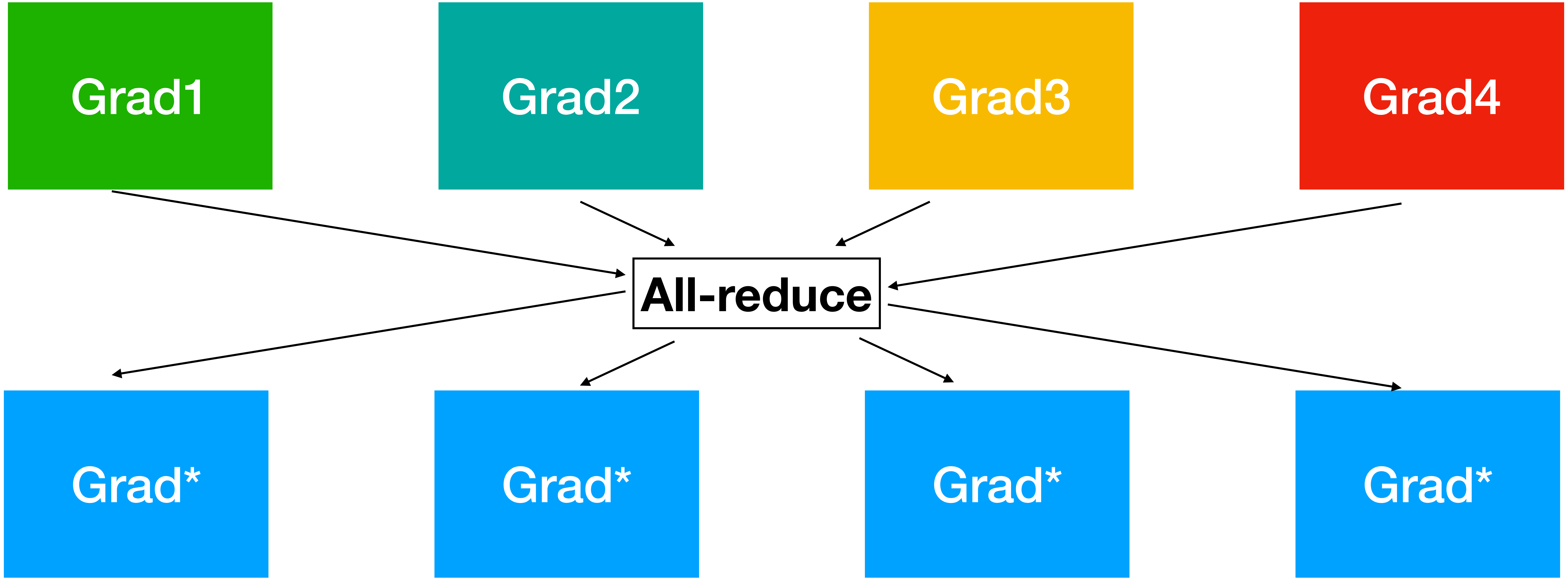
- No redundant computation in each forward-backward pass
- Communication only on gradients
  - One all-reduce operation

Is DDP optimal?

No!

Parameter optimization is entirely redundant

Distributed Data Parallel (DDP)



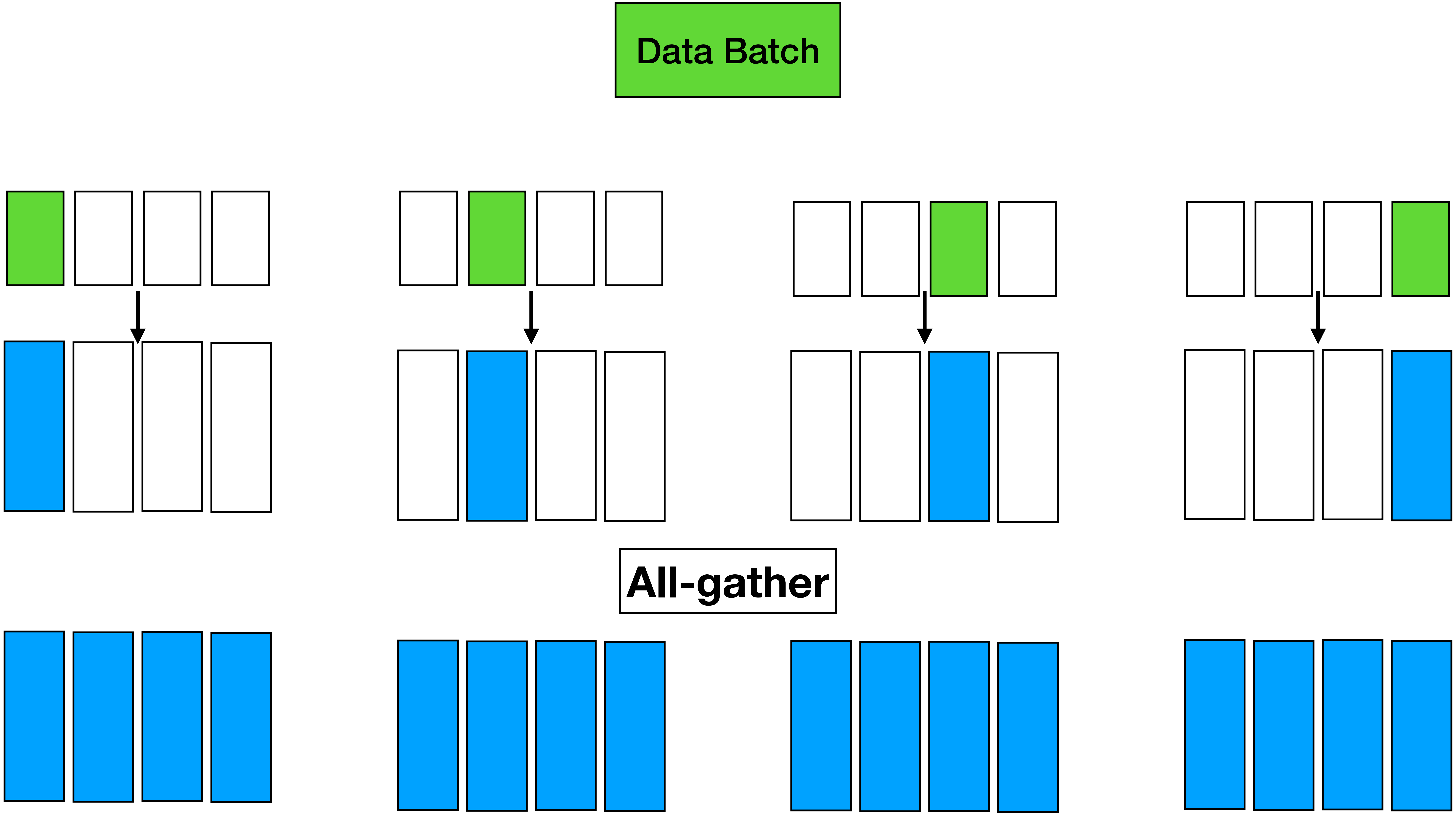
$$L = \frac{1}{4}(L_1 + L_2 + L_3 + L_4)$$

Distributed Data Parallel (DDP)

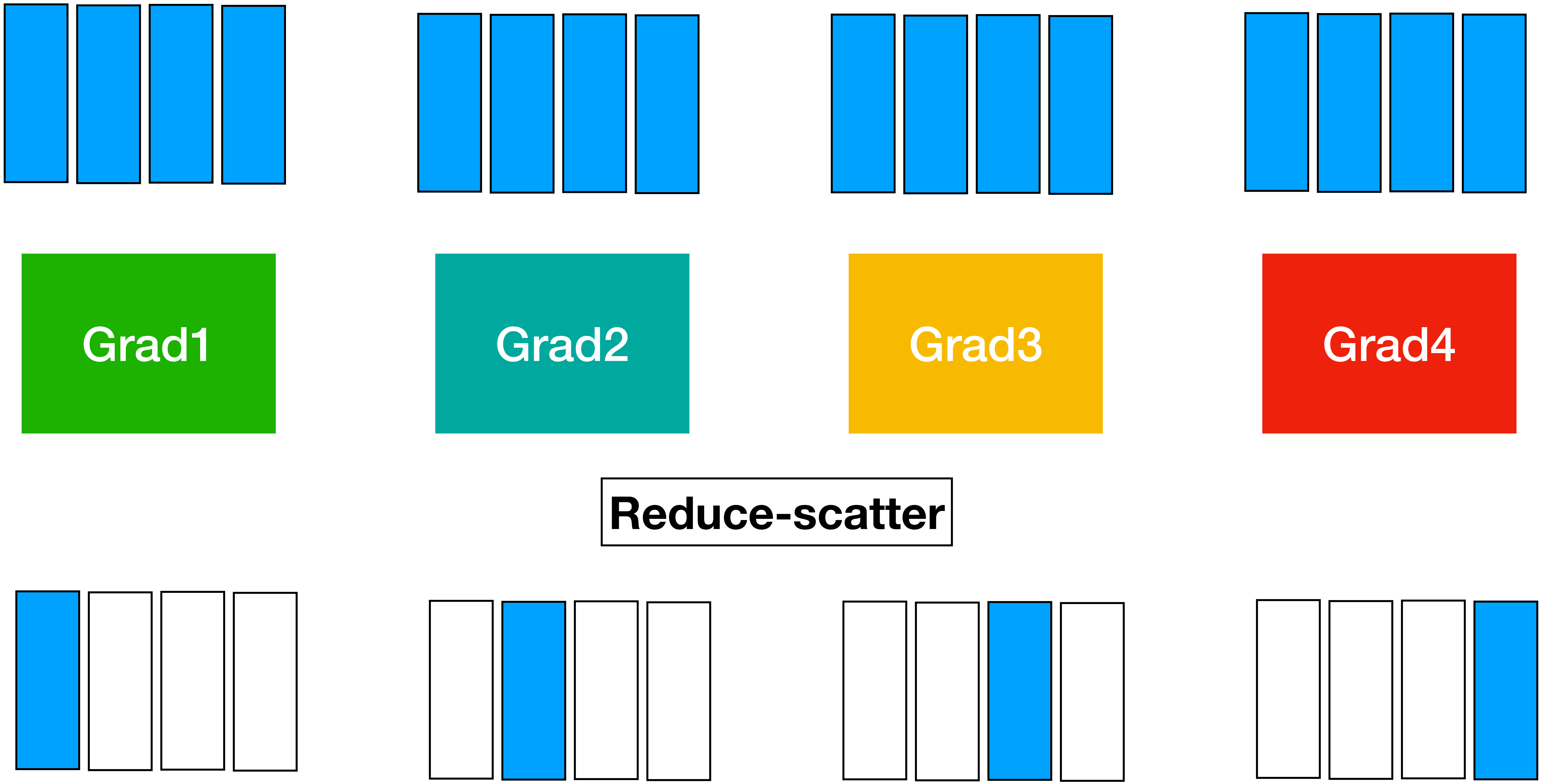
- **For large-scale pretraining, parameter optimization is expensive**
  - A whole copy of 7B model takes 28G memory
  - Gradient takes the same memory as parameters
  - Optimization states in Adam optimizer take two times of model parameters
  - 28G x 4 = 112G memory for only storing model and optimization states

How about splitting model parameters together with data?

Distributed Data Parallel (DDP)



Distributed Data Parallel (DDP)

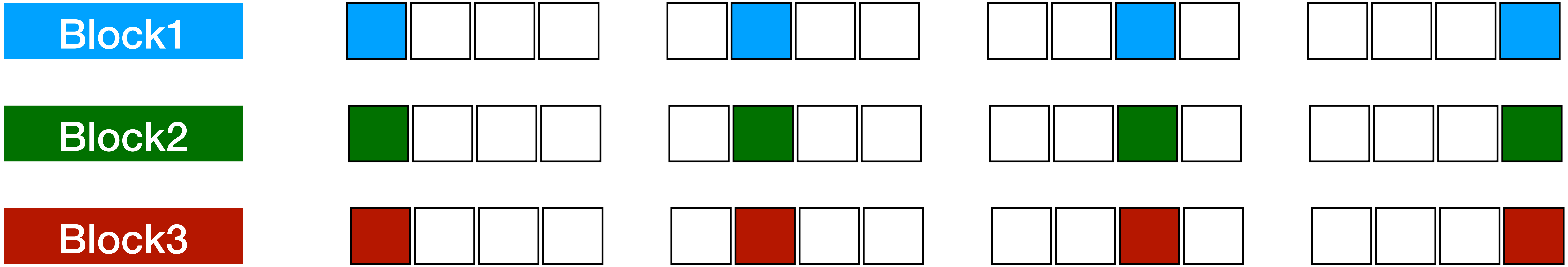


Distributed Data Parallel (DDP)

- No redundant computation in optimizing parameters
- No redundant memory cost for the two optimization states in Adam
- No more communication overhead
  - All-reduce = Reduce-Scatter + All-gather
- At one moment, still need to store the whole parameters and gradients

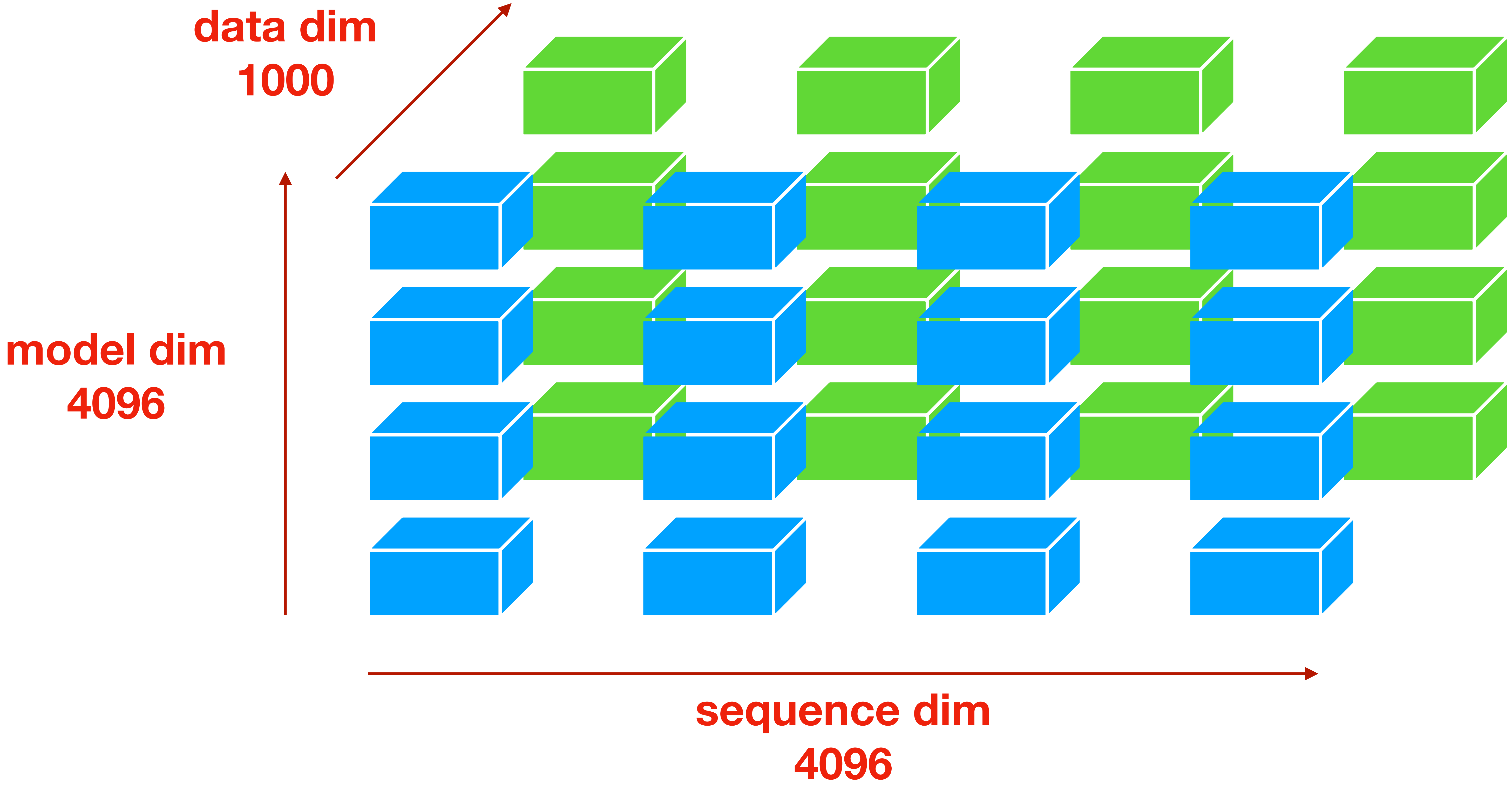
Fully Sharded Data Parallel (FSDP)

- One Transformer model consists of multiple blocks
- Split each block parameters individually
- Gather the parameters of one layer only when we need to compute the output of that layer



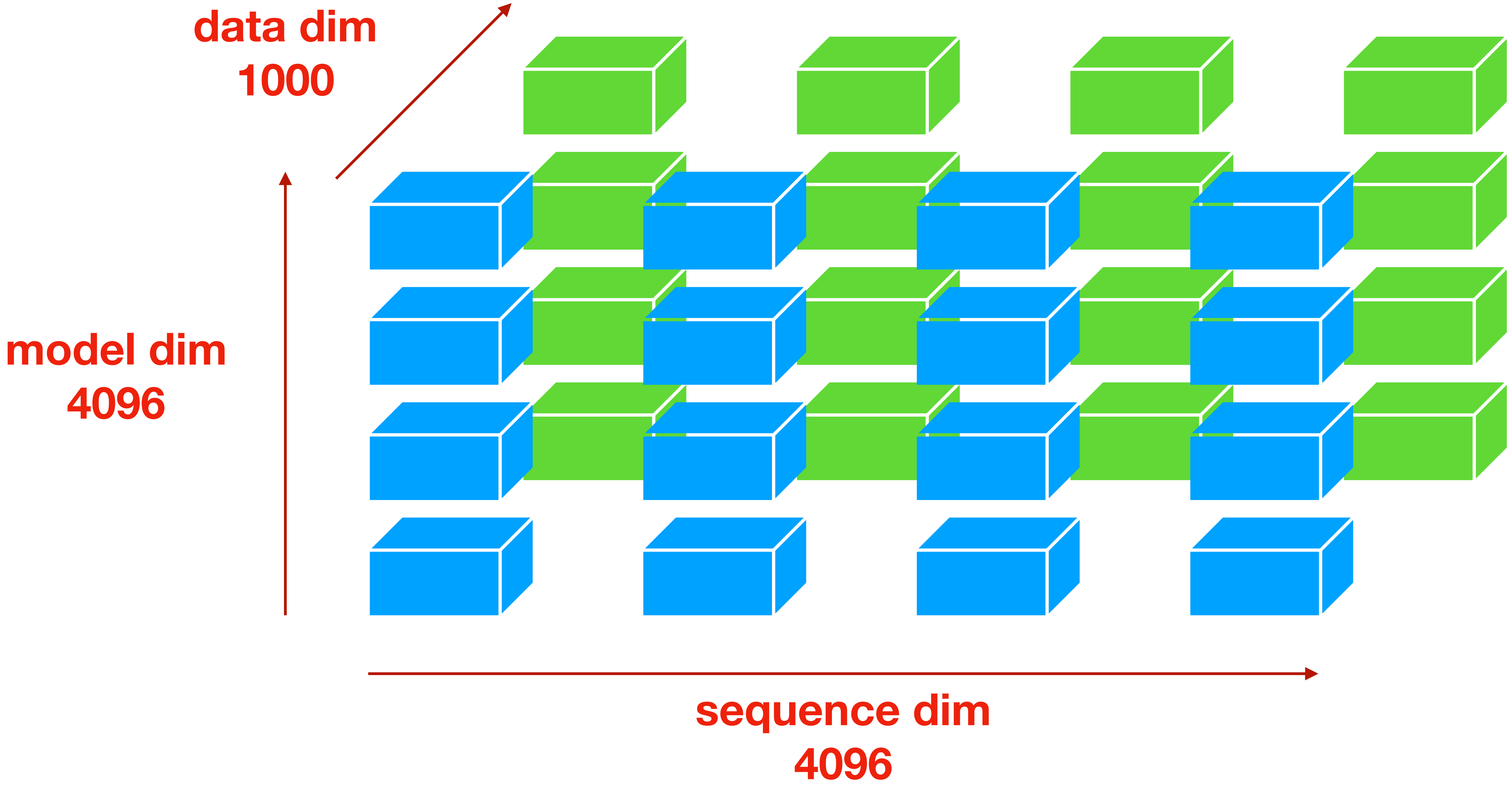
Distributed Large-Scale Pretraining

- **Three Criteria**
  - Minimal redundant computation
  - Minimal peak memory cost
  - Minimal communication overhead

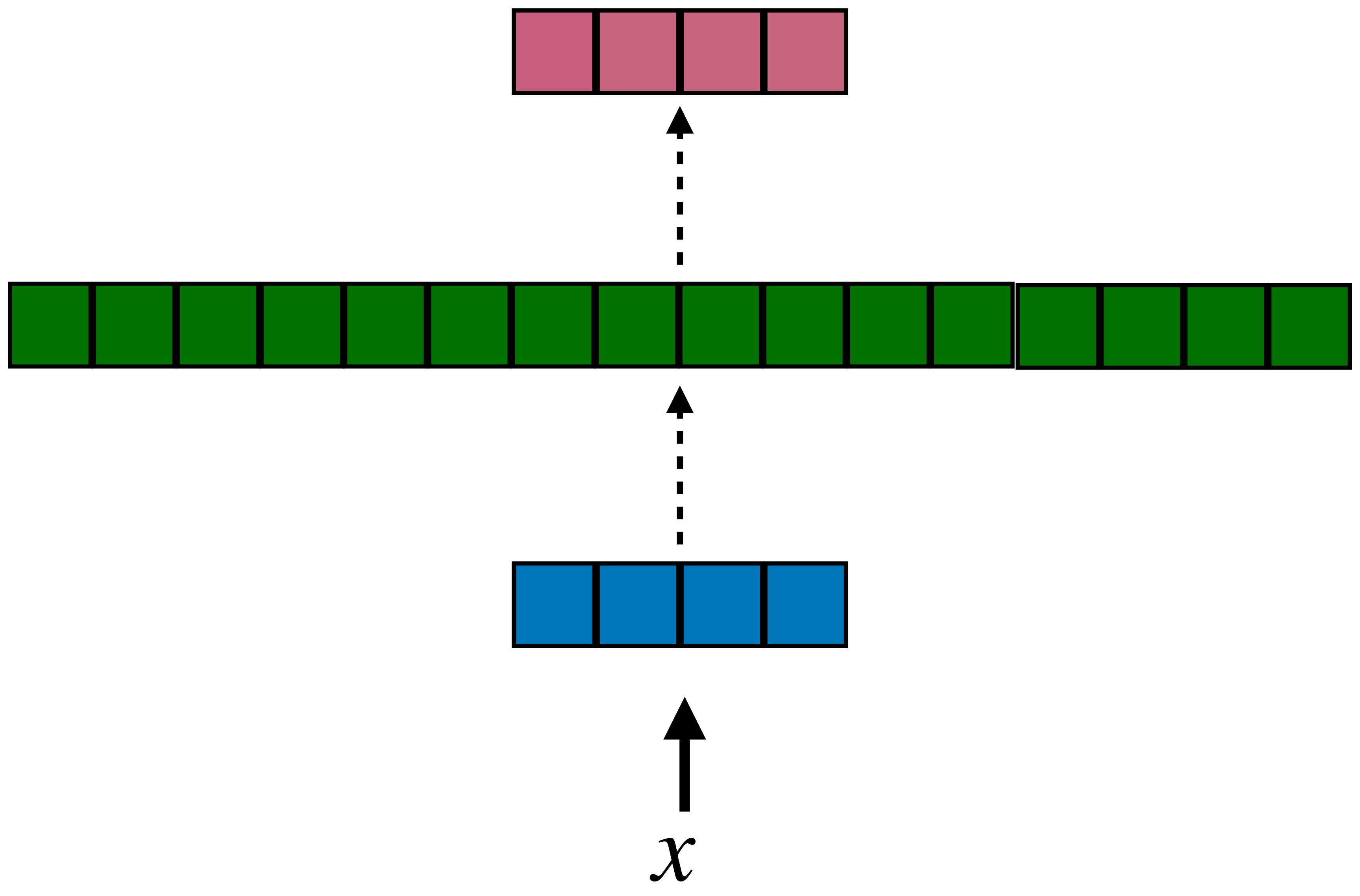


Model/Tensor Parallel

- **Specifically designed for linear layers**
- **Partition both model parameters and hidden states along model dimension**
  - Can be applied together with data parallel



FFN



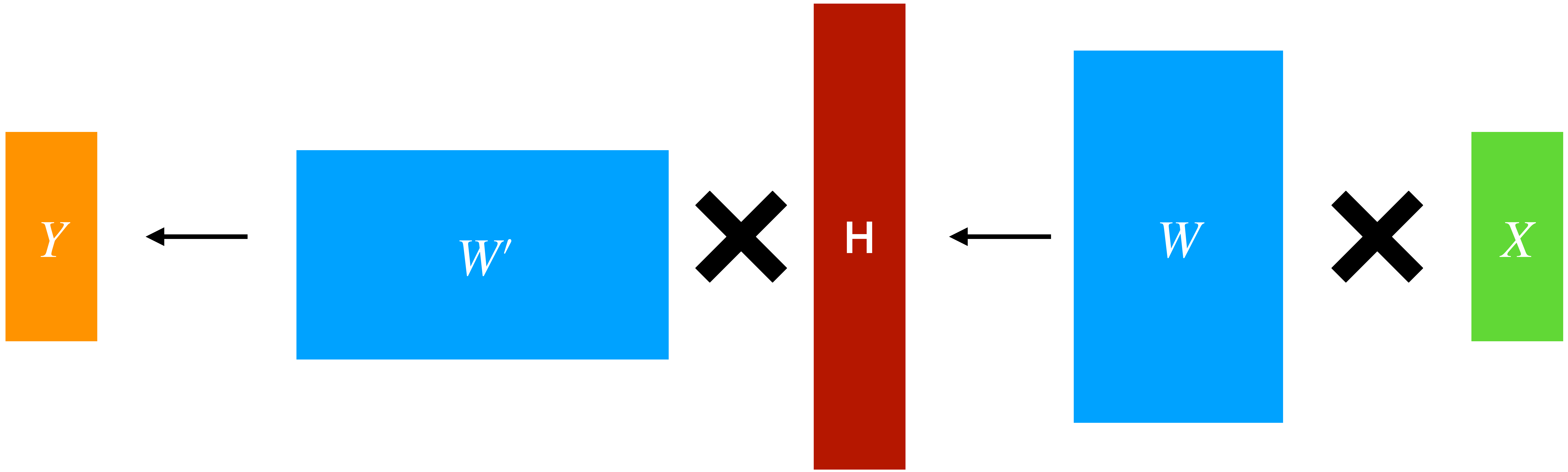
$\in \mathrm{R}^d$

$\in \mathrm{R}^{d_{ff}}$

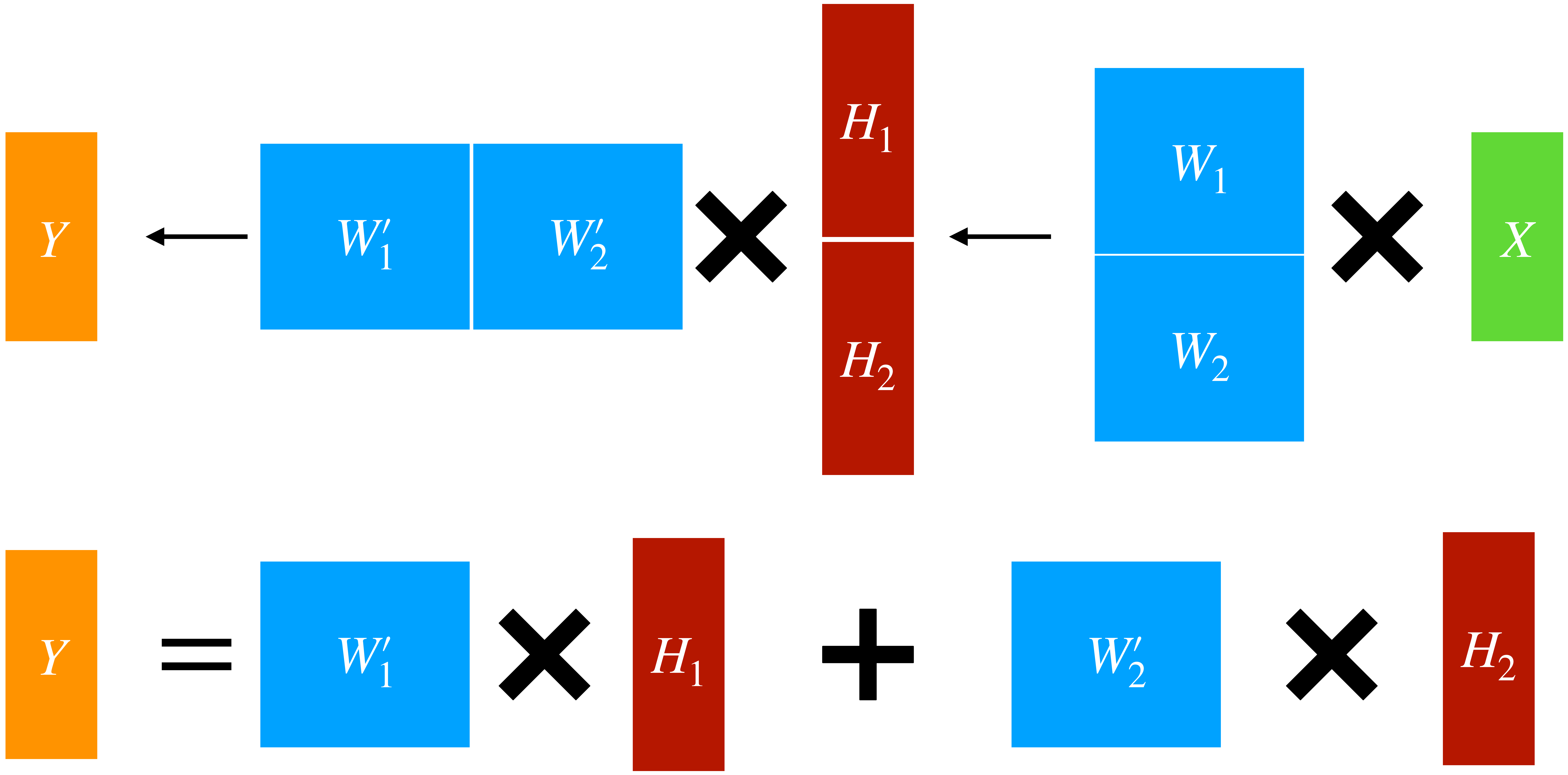
$\in \mathrm{R}^d$

$$d_{ff} = 4d$$

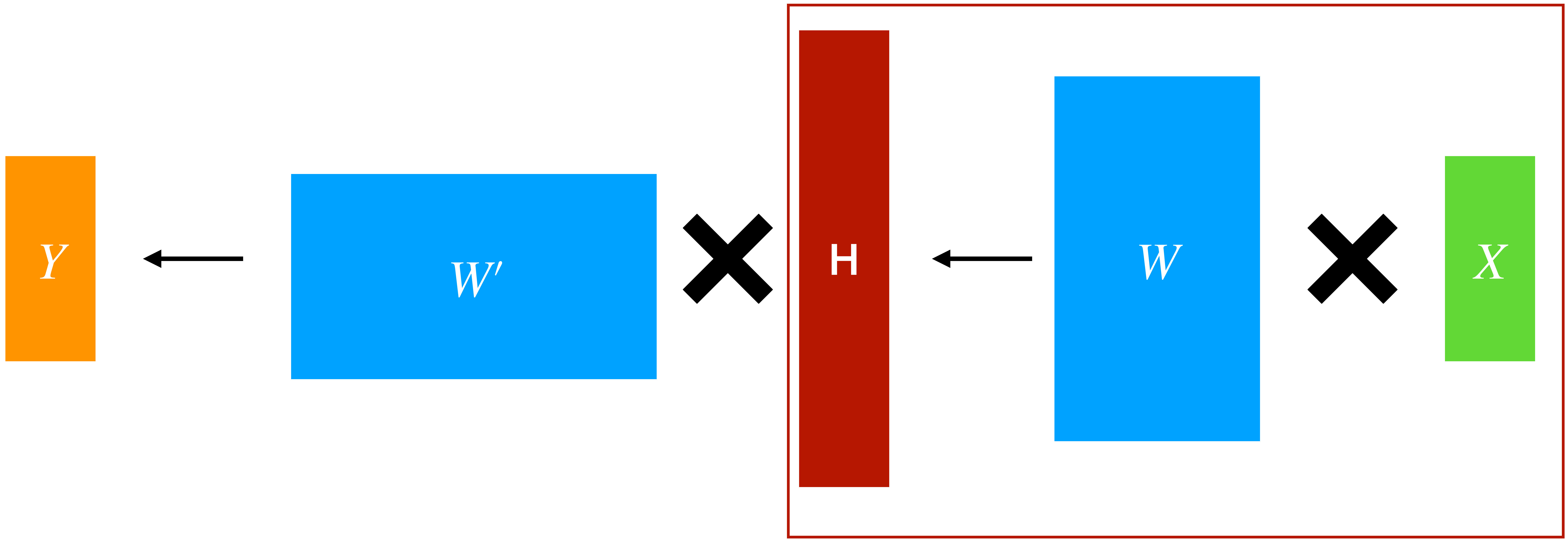
Model/Tensor Parallel



Model/Tensor Parallel

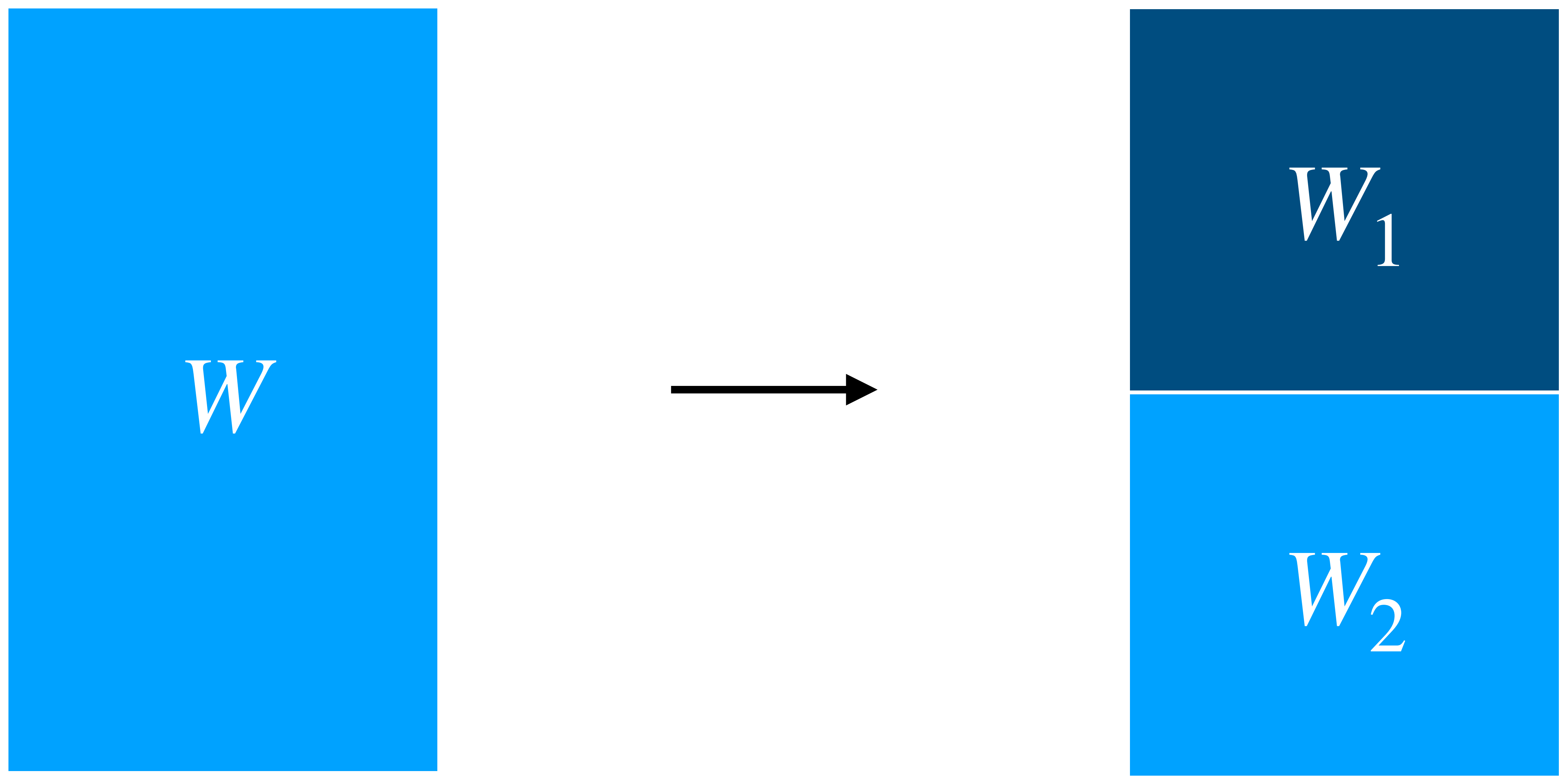


Model/Tensor Parallel

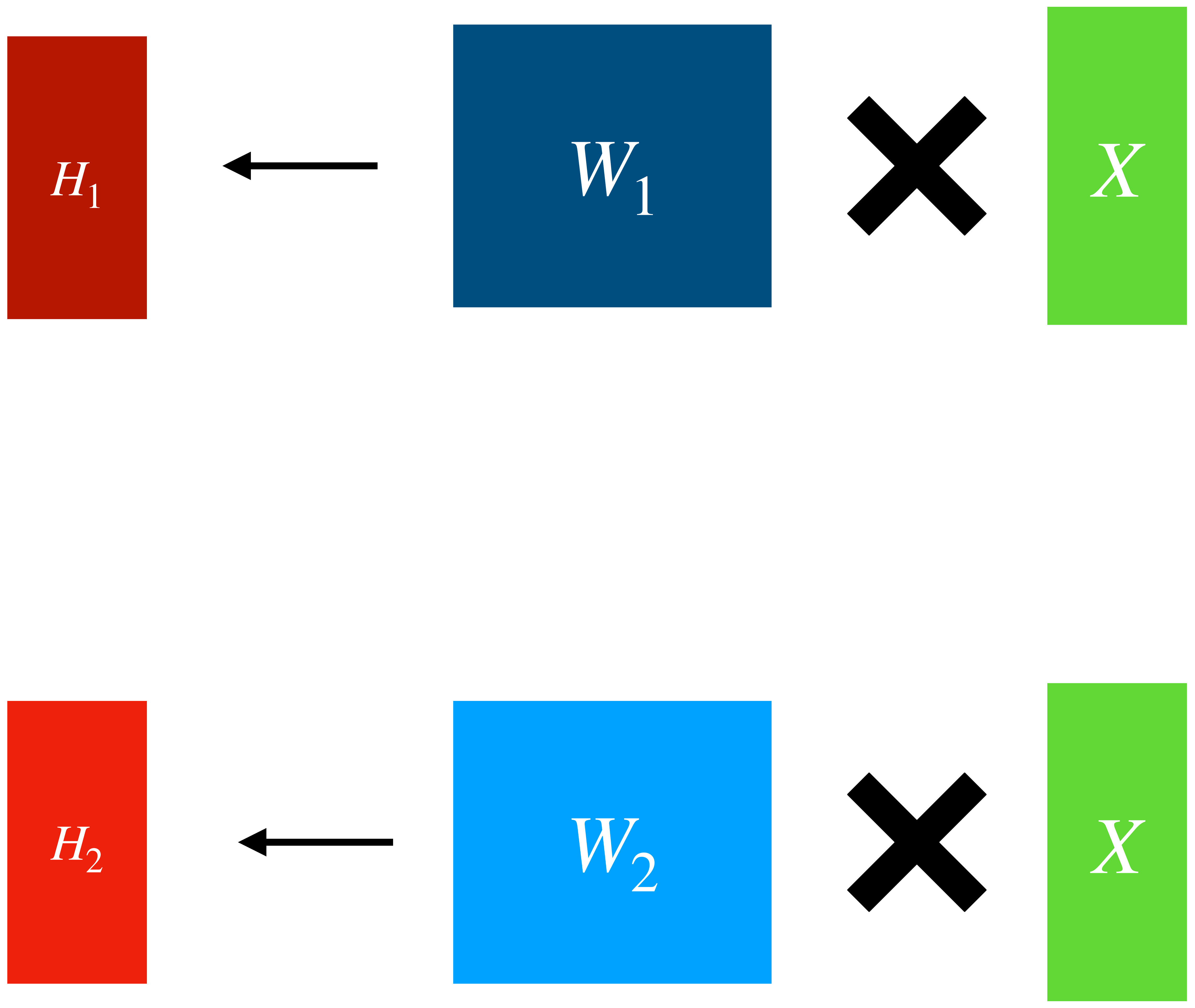


Column Parallel

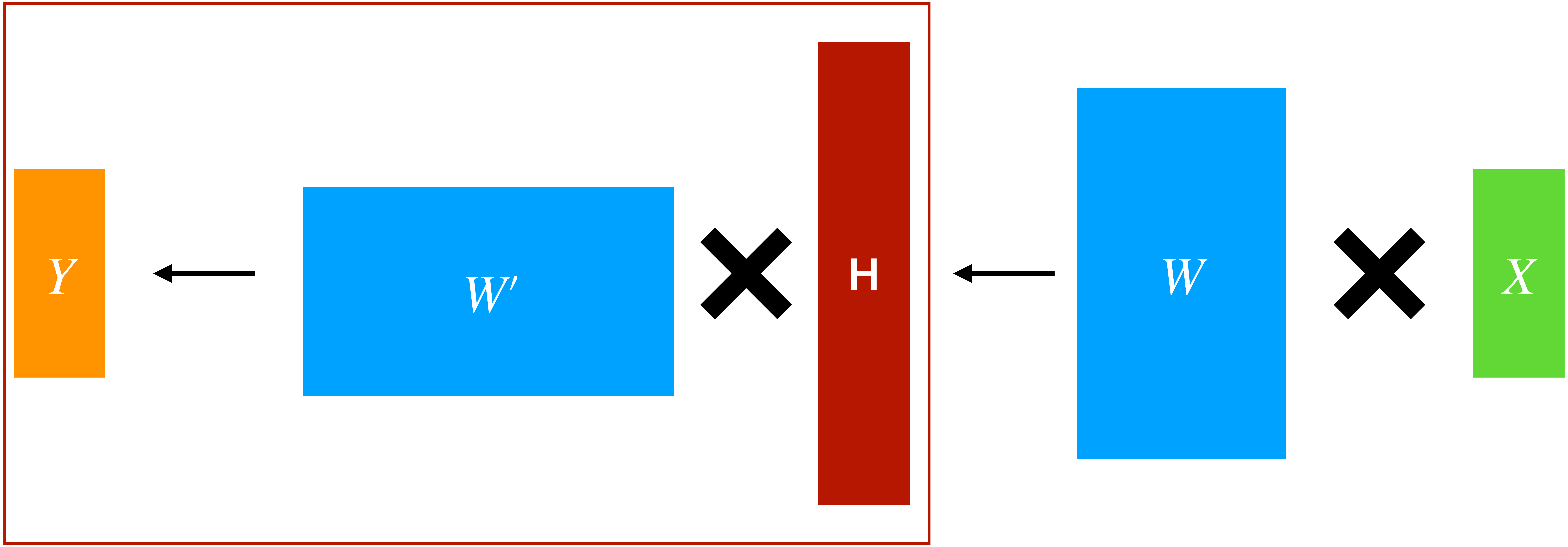
- **Split the weight matrix along the column axis**



Column Parallel

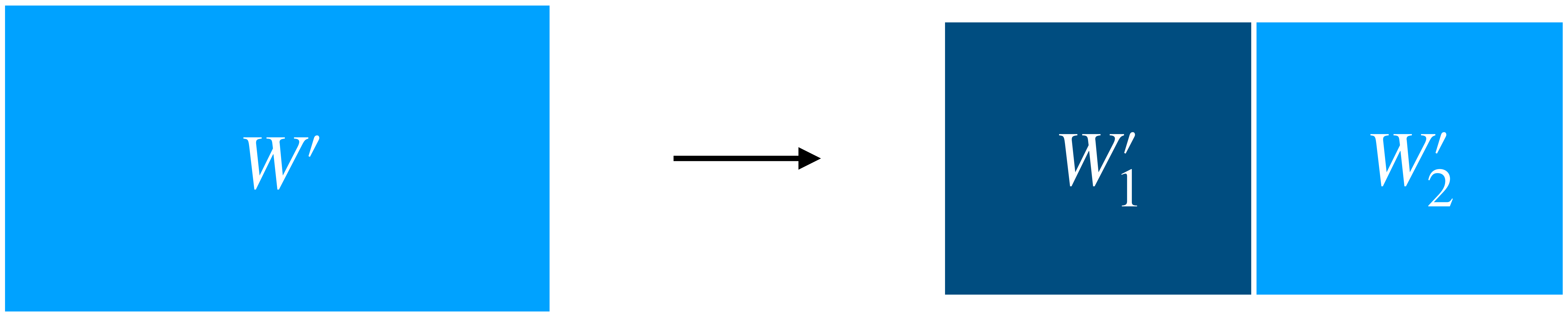


Model/Tensor Parallel

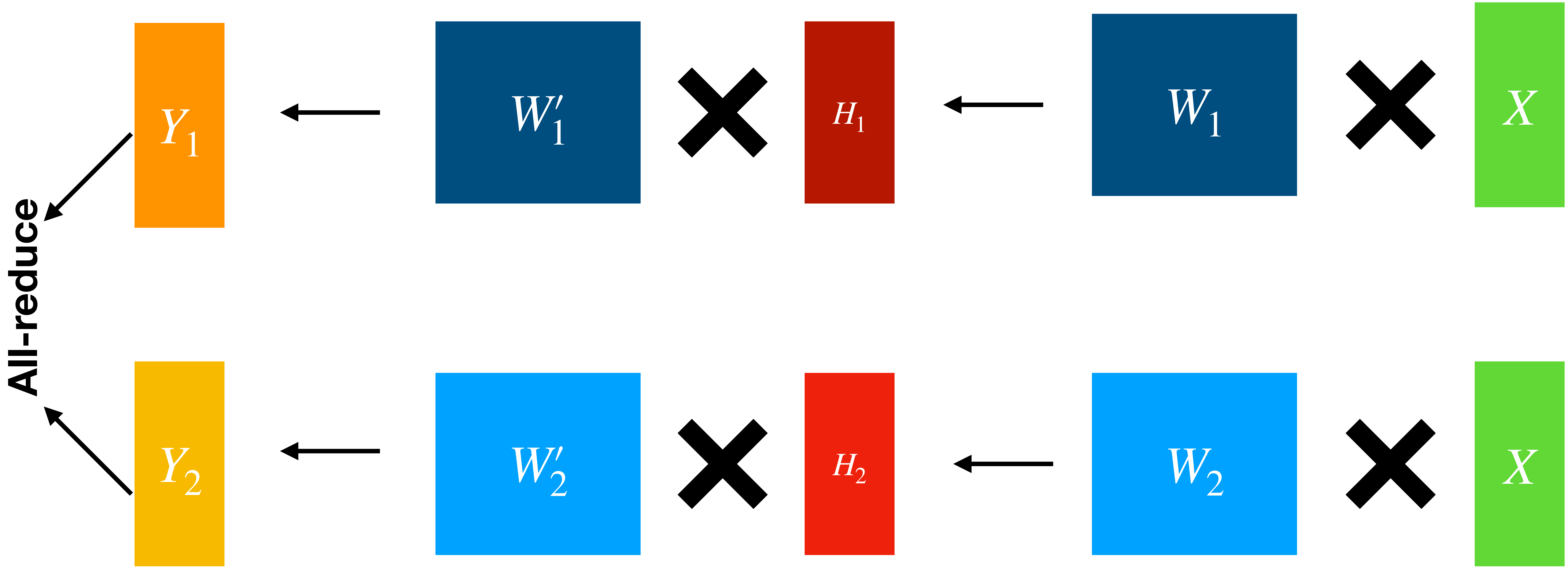


Raw Parallel

- Split the weight matrix along the raw axis



Raw Parallel



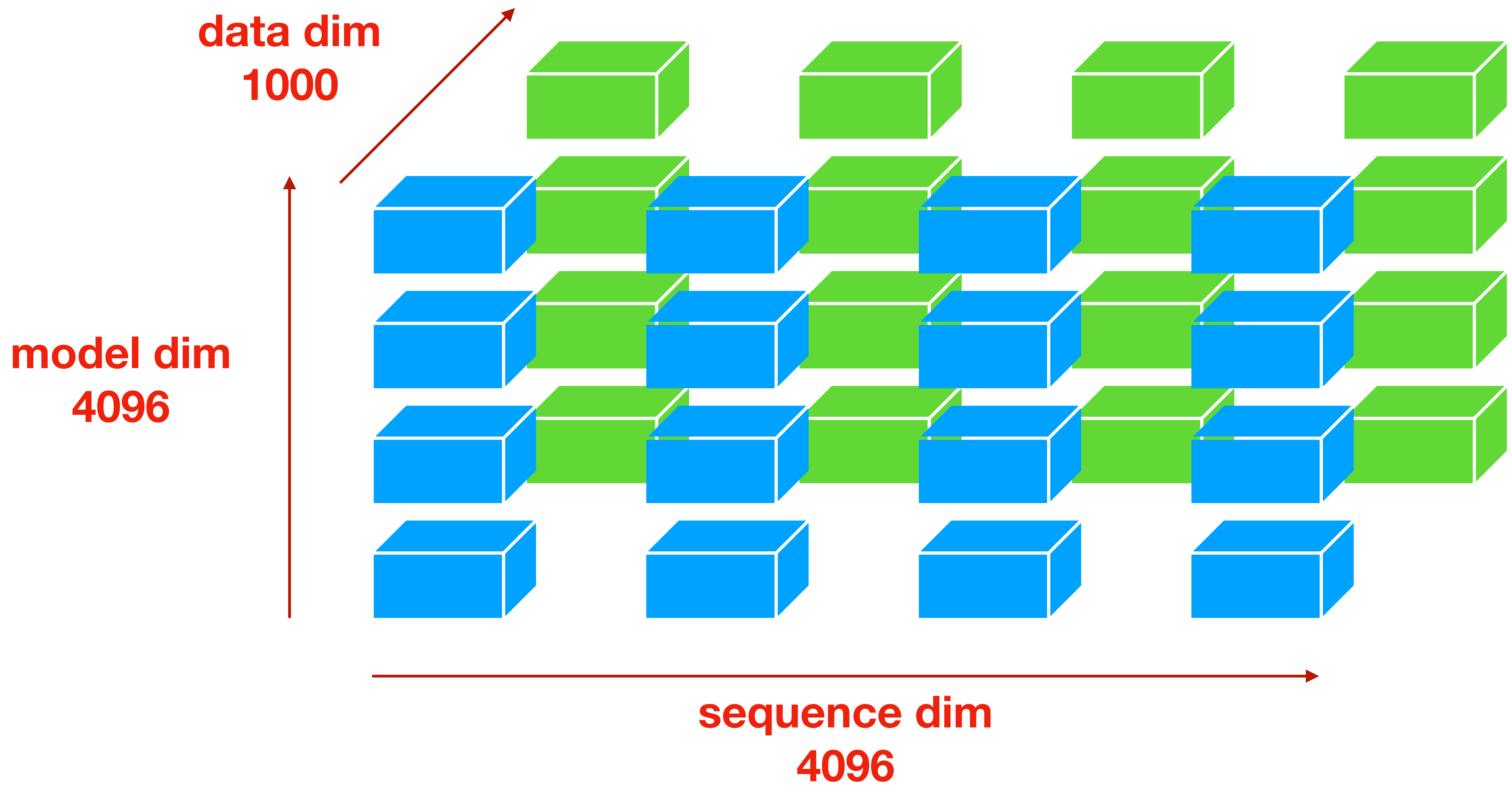
$$Y = Y_1 + Y_2$$

Model/Tensor Parallel

- **Communication is heavy**
  - Only applied to GPUs in the same computing node
- **How to apply tensor parallel to the attention layer?**
  - First apply column parallel to QKV matrices
  - Compute attentions of different heads in different GPUs
  - Apply raw parallel to the output matrix to get the final attention output

Distributed Large-Scale Pretraining

- **Three Criteria**
  - Minimal redundant computation
  - Minimal peak memory cost
  - Minimal communication overhead



Other Techniques in Large-Scale Pretraining

- **Activation Checkpointing**
  - https://pytorch.org/docs/stable/checkpoint.html

- **Half-Precision Training**

FP32

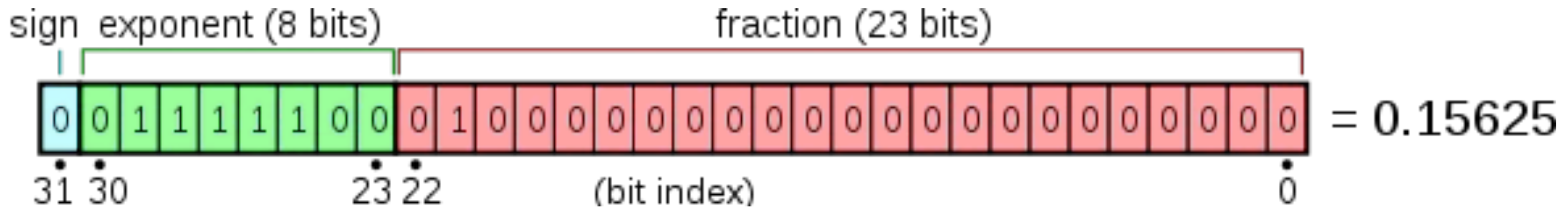
- 1 bit sign
- 8 bits exponent
- 23 bits fraction
- 1e38 range

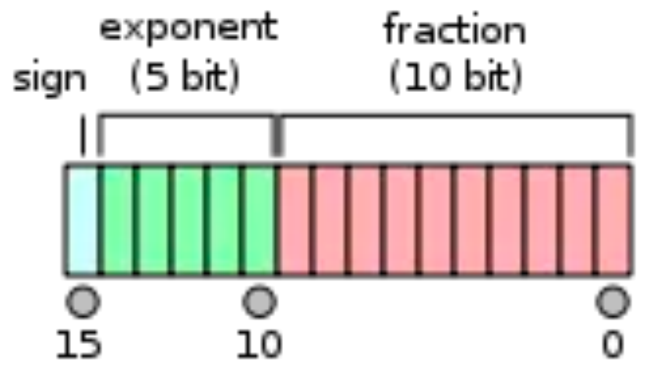
FP16

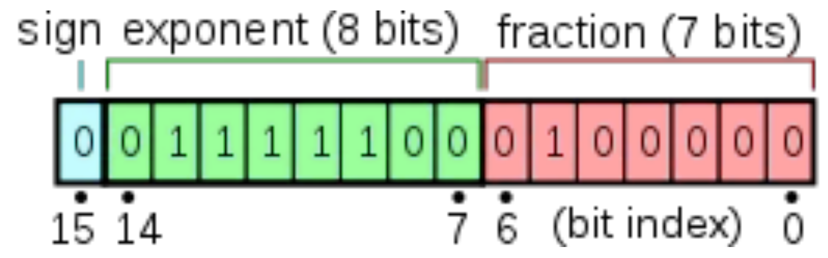
- 1 bit sign
- 5 bits exponent
- 10 bits fraction
- 65504 range

BF16

- 1 bit sign
- 8 bits exponent
- 7 bits fraction
- 1e38 range







Thanks!

Q&A