Parallelism - 2

EECE695D: Efficient ML Systems

Spring 2025

Recap

- **Last class.**
  - Data parallelism
  - Model parallelism
    - Pipeline, Tensor, Expert
- **Today.** Advanced topics
  - Sequence parallelism
  - ZeRO, Gradient compression
  - Automated parallelism

Sequence parallelism

Motivation

- Training a transformer-based generative model
  - Want to generate high-dimensional data with an **extremely long context**
  - Example. High-resolution video generation
    - Spatio-temporal tokens as an input
- **Problem.** Not holdable on one device, even for a small batch

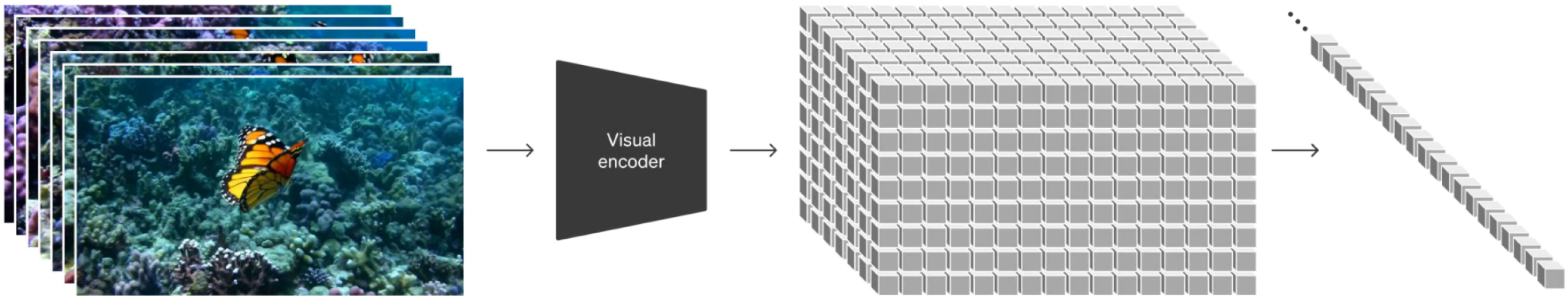
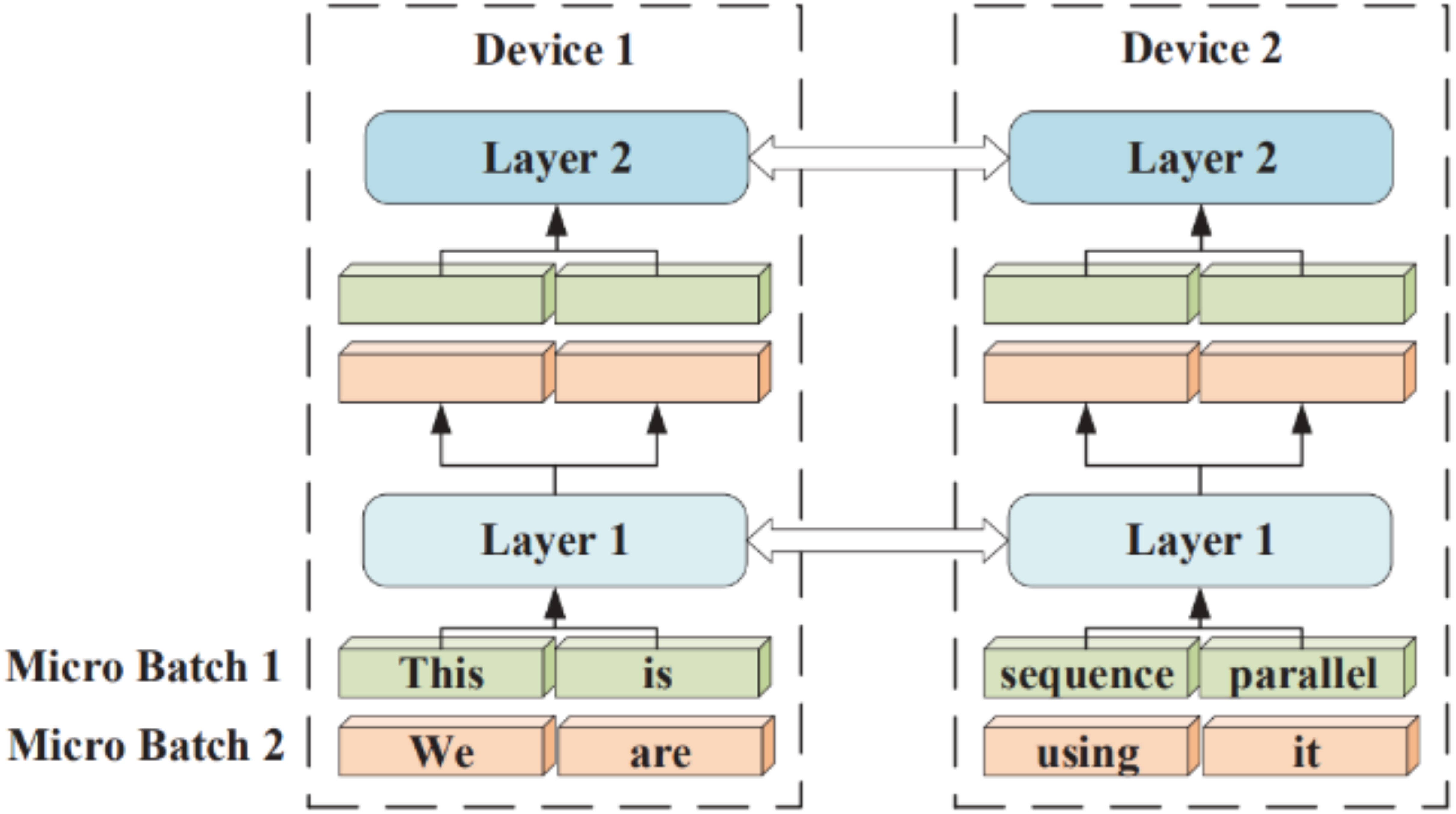


Image Source: OpenAI Sora

Basic idea

- **Solution.** Each GPU processes a **fraction of input tokens**
  - FFN. Easy, because tokens are handled separately anyways
  - MHSA. Requires additional communication



Li et al., "Sequence Parallelism: Long Sequence Training from System Perspective," ACL 2023

The case of MHSA

- **Goal.** Compute the output of each token:

$$\mathbf{o}_i = \sum_{i=1}^{L} \mathbf{s}_i \mathbf{v}_i$$

  - $\mathbf{s}_i$ are the attention scores:

$$\mathbf{s}_i = \mathrm{SoftMax}\left(\frac{1}{\sqrt{d}}[\mathbf{q}_i^\top \mathbf{k}_1, \mathbf{q}_i^\top \mathbf{k}_2, \ldots, \mathbf{q}_i^\top \mathbf{k}_L]\right)$$

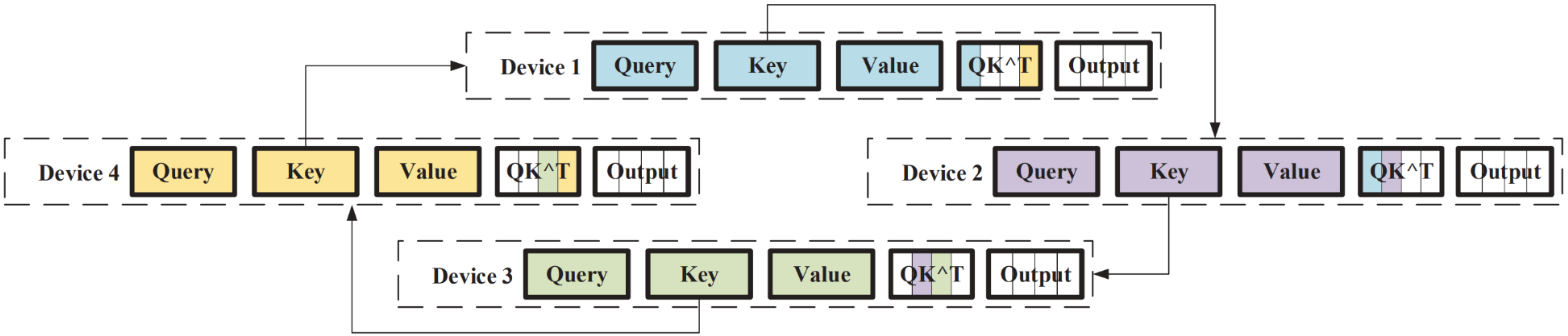
  - $\mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i$ are query/key/values.

- **Problem.** Tokens are distributed among devices (k&v, in particular)

Li et al., "Sequence Parallelism: Long Sequence Training from System Perspective," ACL 2023

Ring Self-Attention

- **Idea.** Transmit the key and value embeddings of the sequence
- Step 1. Compute and transmit keys
    - Each node can start computing the $\mathbf{q}_i^\top \mathbf{k}_j$, as soon as they receive any fraction of the key embeddings
    - After the full ring, can compute the softmax to get attention scores

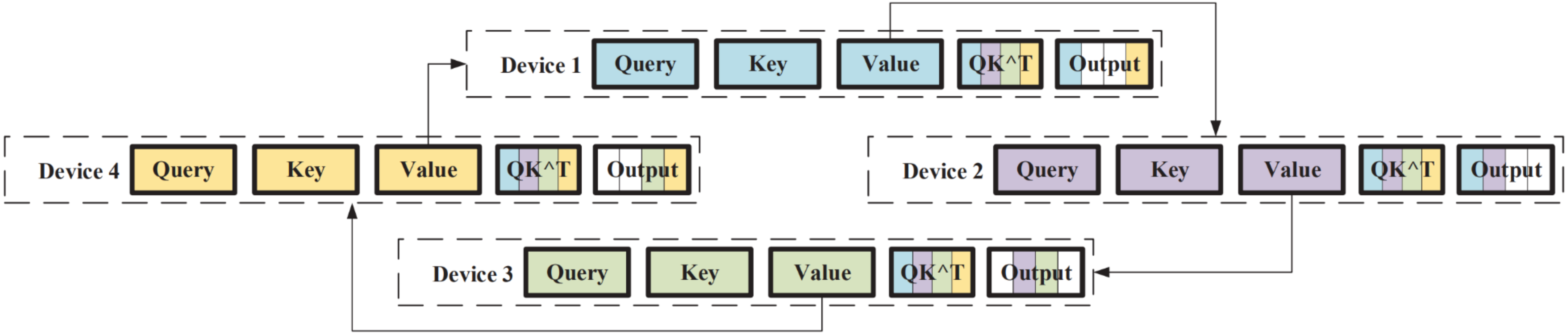


Li et al., "Sequence Parallelism: Long Sequence Training from System Perspective," ACL 2023

Ring Self-Attention

- Step 2. Compute and transmit values

- Step 3. Now everybody has the full KV, and can compute the full output



Li et al., “Sequence Parallelism: Long Sequence Training from System Perspective,” ACL 2023

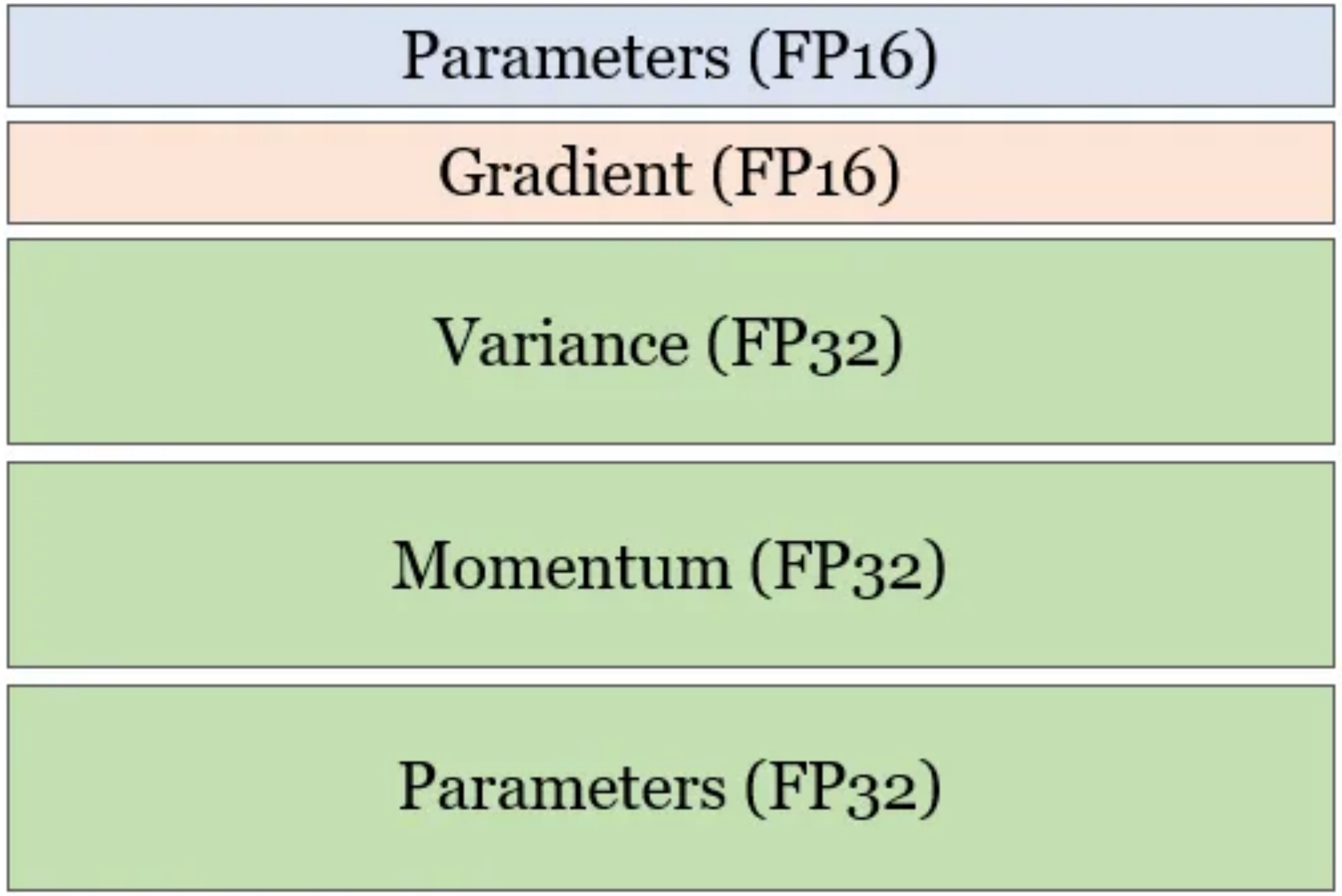
Further readings

- Combined framework with other notions of parallelism
  - Megatron-SP (NVIDIA)
    - Combines with tensor parallelism
    - https://arxiv.org/abs/2205.05198
  - DeepSpeed-Ulysses (Microsoft)
    - https://arxiv.org/abs/2309.14509

ZeRO

Motivation

- If we use optimizers like AdamW, we need to keep various **optimizer states**
- **Example.** Optimizing a model with M parameter with Adam, in FP16
  - Param. $2M$ bytes
  - Grad. $2M$ bytes
  - Variance. $4M$ bytes
  - Momentum. $4M$ bytes
  - FP32 Params. $4M$ bytes

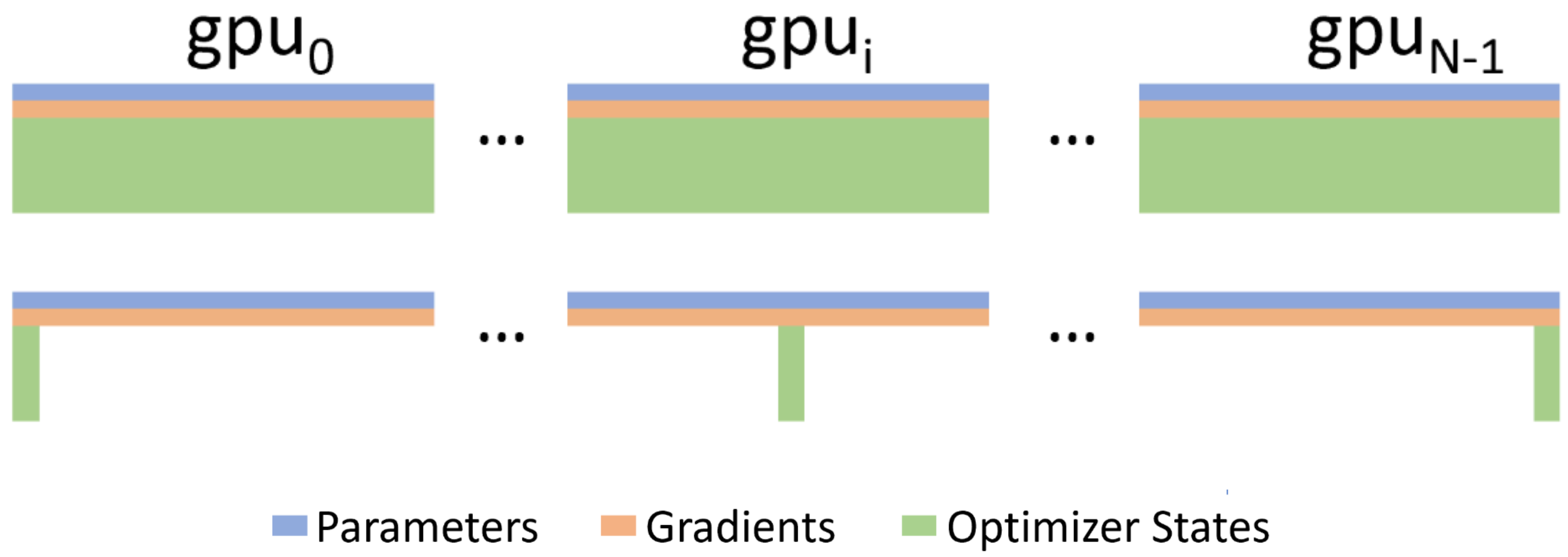


$\Rightarrow$ High redundancy in GPUs, when we do data-parallel

Rajbhandari et al., “ZeRO: Memory Optimizations Toward Training Trillion Parameter Models,” SC 2020

Idea

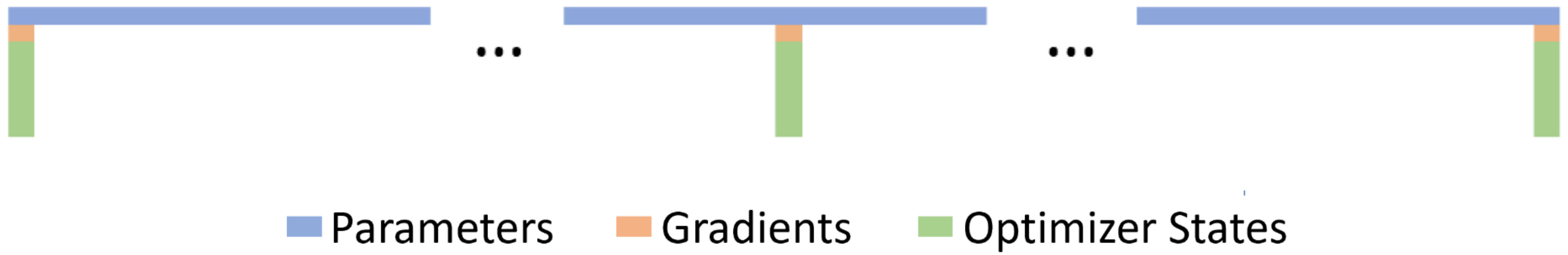
- Partition the states and gradients on many GPUs
- **ZeRO-1.** The optimizer states are distributed (~4x memory reduction)
  - Gradients for each GPU are partitioned and sent to corresponding GPUs
  - Updated parameters are sent to all GPUs



Rajbhandari et al., "ZeRO: Memory Optimizations Toward Training Trillion Parameter Models," SC 2020

Idea

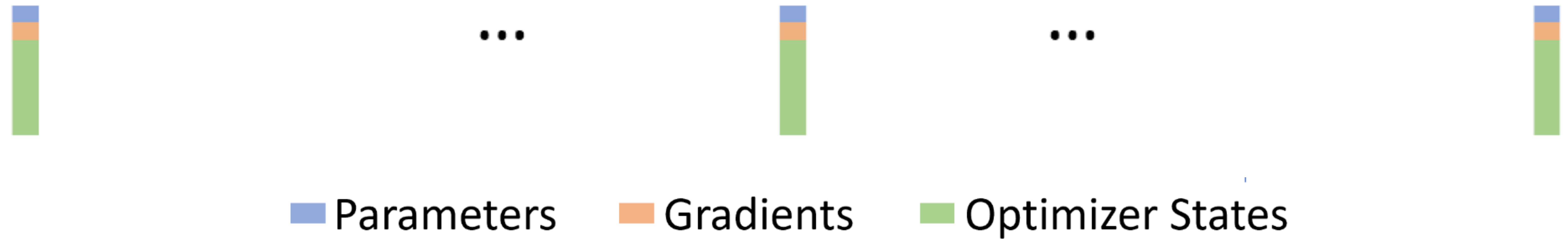
- **ZeRO-2.** Gradients are partitioned as well (~8x memory reduction)
  - On GPU i, the gradients for layer j is:
    - Kept, if the GPU i is responsible for layer j
    - Discarded, otherwise, after computing the gradient for layer j-1 and transmitting to the responsible GPU



Rajbhandari et al., "ZeRO: Memory Optimizations Toward Training Trillion Parameter Models," SC 2020

Idea

- **ZeRO-3.** Even parameters are partitioned
  - Significant communication load; use when extremely memory-poor



M is the number of parameters, N is the number of devices.

	Optimizer States (12M)	Gradients (2M)	Model Weights (2M)	Memory Cost	Communication Cost
Data Parallelism	Replicated	Replicated	Replicated	$16M$	all-reduce(2M)
ZeRO Stage 1	Partitioned	Replicated	Replicated	$4M + \frac{12M}{N}$	all-reduce(2M)
ZeRO Stage 2	Partitioned	Partitioned	Replicated	$2M + \frac{14M}{N}$	all-reduce(2M)
ZeRO Stage 3	Partitioned	Partitioned	Partitioned	$\frac{16M}{N}$	1.5 all-reduce(2M)

Rajbhandari et al., "ZeRO: Memory Optimizations Toward Training Trillion Parameter Models," SC 2020

Further materials

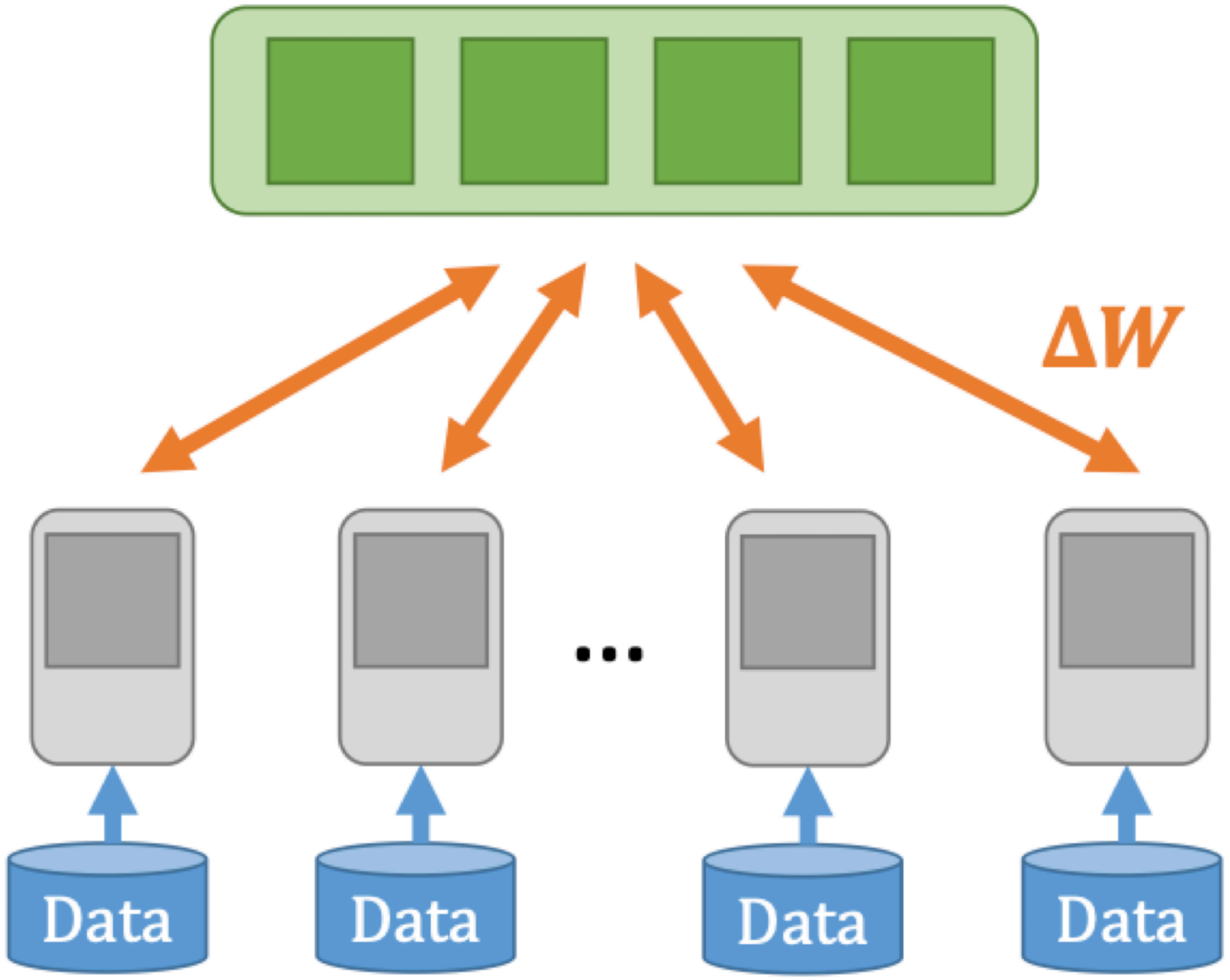
- Cool explanatory video
  - https://www.microsoft.com/en-us/research/blog/zero-deepspeed-new-system-optimizations-enable-training-models-with-over-100-billion-parameters/
- Other advances
  - Memory checkpointing, offloading, and so on
  - https://arxiv.org/abs/1910.02054

Rajbhandari et al., "ZeRO: Memory Optimizations Toward Training Trillion Parameter Models," SC 2020

Gradient compression

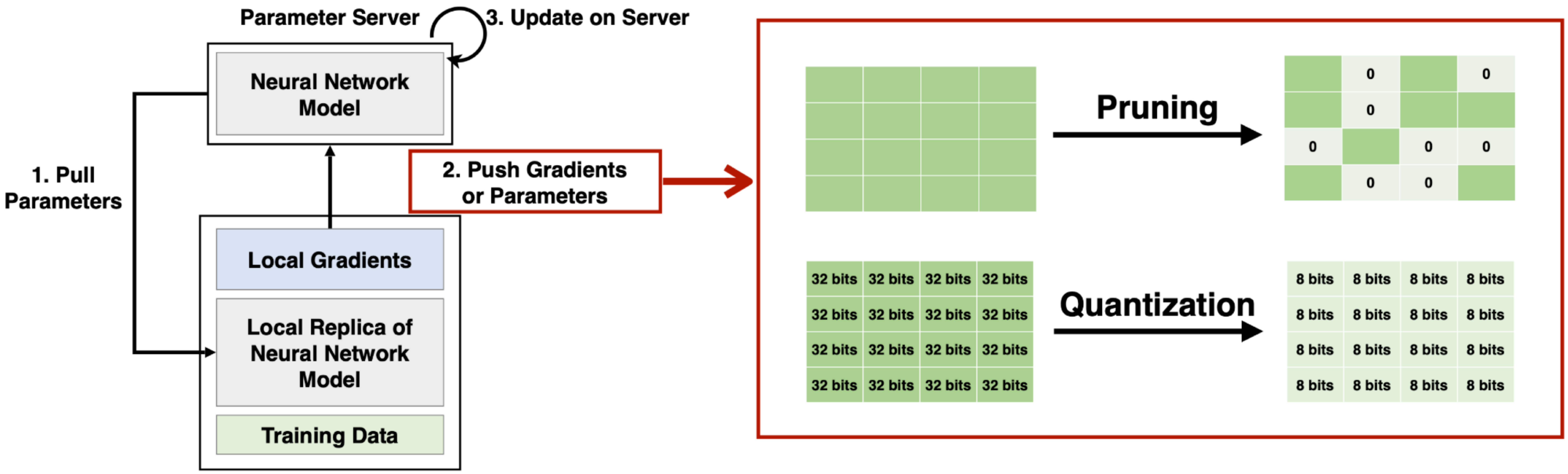
Motivation

- Recall that in DP, the key bottleneck is the communication bandwidth
  - Transmitting gradients & model updates



Basic idea

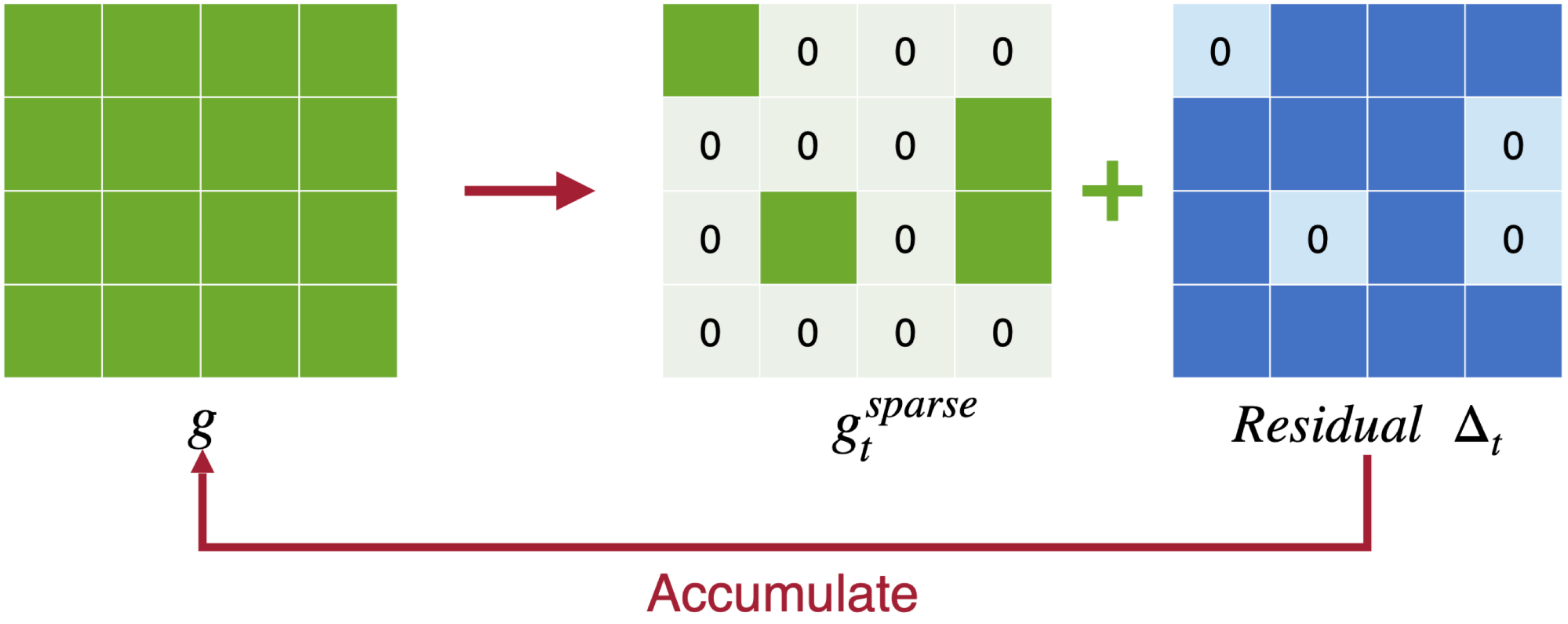
- Compress the gradients using model compression techniques
- **Remark.** No longer need to take "inference efficiency" into account
  (e.g., no stringent need for linear quantization)
  - Instead, encoding / decoding cost may be an issue



Lin et al., "Deep gradient compression: Reducing the communication bandwidth for distributed training" ICLR 2018

Sparsity

- Select only **top-K gradients** (i.e., magnitude pruning)
  - What is not transmitted ("residuals") are stored, for the next communication round



Lin et al., "Deep gradient compression: Reducing the communication bandwidth for distributed training" ICLR 2018

Sparsity: Nitty-gritty details

- **Momentum.** Update the momentums based on the pruned gradient, not the original ones
- **Gradient Clipping.** Clip the gradients before adding the residuals
- **Warm-up.** Warm up both step size and sparsity

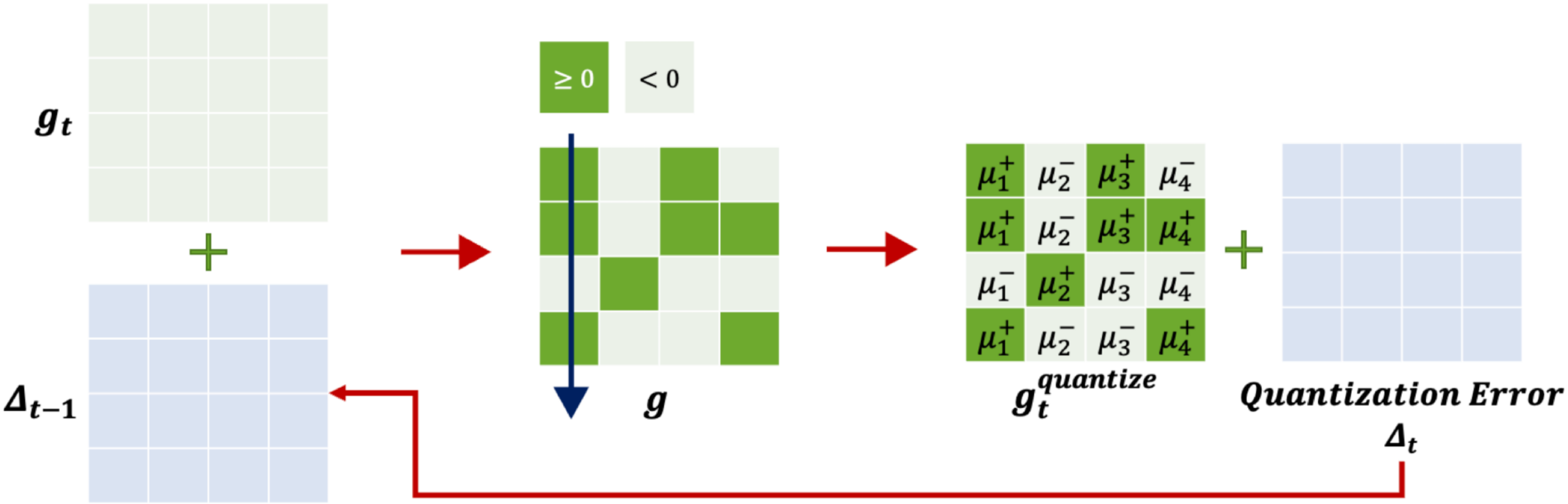
Lin et al., "Deep gradient compression: Reducing the communication bandwidth for distributed training" ICLR 2018

Task		Baseline	Deep Gradient Compression
ResNet-50 On ImageNet	Top-1 Accuracy	75.96%	**76.15% (+0.19%)**
	Top-5 Accuracy	92.91%	**92.97% (+0.06%)**
	Gradient Compression Ratio	1 ×	**277 ×**
5-Layer GRU On LibriSpeech	Word Error Rate (WER)	9.45%	**9.06% (-0.39%)**
	Word Error Rate (WER)	27.07%	**27.04% (-0.03%)**
	Gradient Compression Ratio	1 ×	**608 ×**
2-Layer LSTM Language Model On Penn Treebank	Perplexity	72.30	**72.24 (-0.06)**
	Gradient Compression Ratio	1 ×	**462 ×**

Lin et al., "Deep gradient compression: Reducing the communication bandwidth for distributed training" ICLR 2018

Quantization

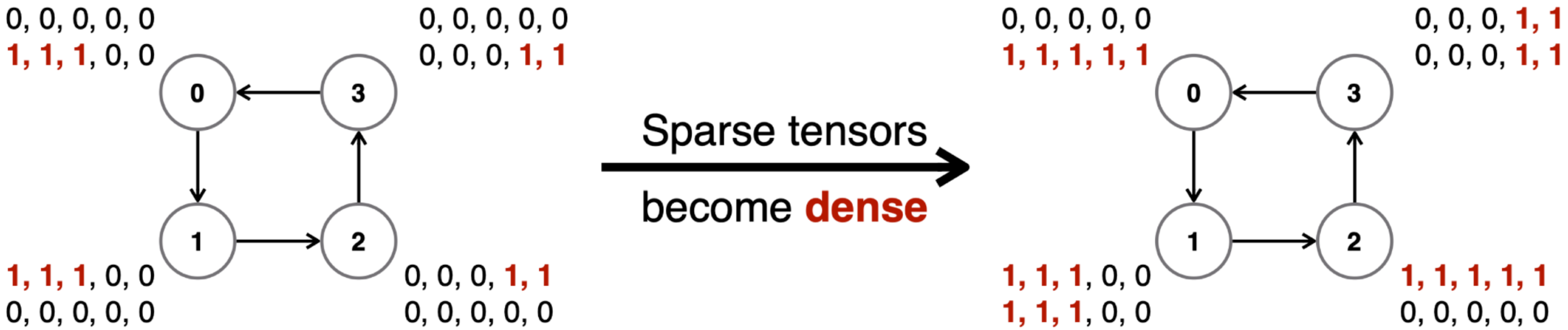
- In **1-bit SGD**, the gradients are quantized to binary values
  - Allocate column-wise scaling factors
  - Accumulate quantization errors



Seide et al., "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs" Interspeech 2014

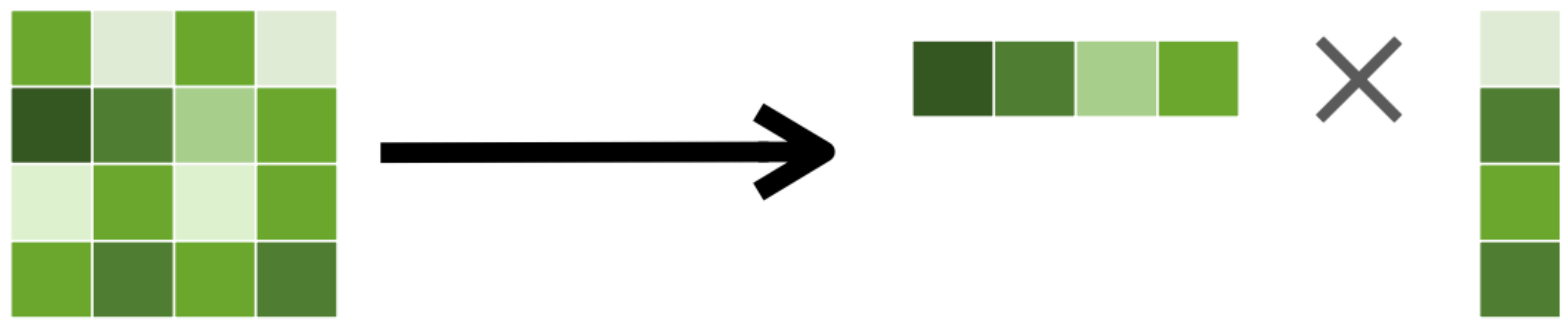
All-reducing compressed gradients

- **Problem.** Suppose that we use all-reduce to aggregate gradient signals
  - Sparsity. No longer sparse
  - Quantization. No longer low-bit
  - Repeated pruning/quantization leads to much noise / order-dependency



PowerSGD

- Apply low-rank approximation to gradients
  - Free of the order-dependency issue



Algorithm 1 Rank-r PowerSGD compression

The update vector Δ_w is treated as a list of tensors corresponding to individual model parameters. Vector-shaped parameters (biases) are aggregated uncompressed. Other parameters are reshaped into matrices. The functions below operate on such matrices independently. For each matrix $M \in \mathbb{R}^{n \times m}$, a corresponding $Q \in \mathbb{R}^{m \times r}$ is initialized from an i.i.d. standard normal distribution.

function COMPRESS+AGGREGATE(update matrix $M \in \mathbb{R}^{n \times m}$, previous $Q \in \mathbb{R}^{m \times r}$)

- $P \leftarrow MQ$
- $P \leftarrow$ ALL REDUCE MEAN(P) ▷ Now, $P = \frac{1}{W}(M_1 + \ldots + M_W)Q$
- $\hat{P} \leftarrow$ ORTHOGONALIZE(P) ▷ Orthonormal columns
- $Q \leftarrow M^\top \hat{P}$
- $Q \leftarrow$ ALL REDUCE MEAN(Q) ▷ Now, $Q = \frac{1}{W}(M_1 + \ldots + M_W)^\top \hat{P}$
- **return** the compressed representation $(\hat{P}, Q)$.

end function

function DECOMPRESS$(\hat{P} \in \mathbb{R}^{n \times r}, Q \in \mathbb{R}^{m \times r})$

- **return** $\hat{P}Q^\top$

end function

Automating model parallelism

Inter-op vs Intra-op

- Roughly, there are two ways to distribute operations:
- **Inter-op.** Assign different operators to different devices
  (e.g., Pipeline parallel)
  - Good. Less communication
  - Bad. Much idle time

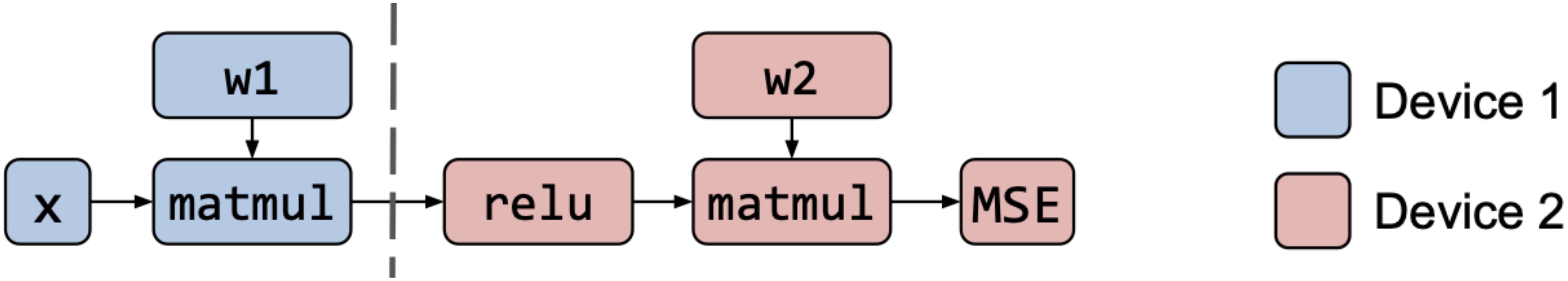


Image Source: https://sites.google.com/view/icml-2022-big-model

Inter-op vs Intra-op

- **Intra-op.** Assign different regions of one operator to different devices
  (e.g., tensor parallel, data parallel)
  - Good. Devices stay busy all the time
  - Bad. Much communication
    - Replication & all-reduce

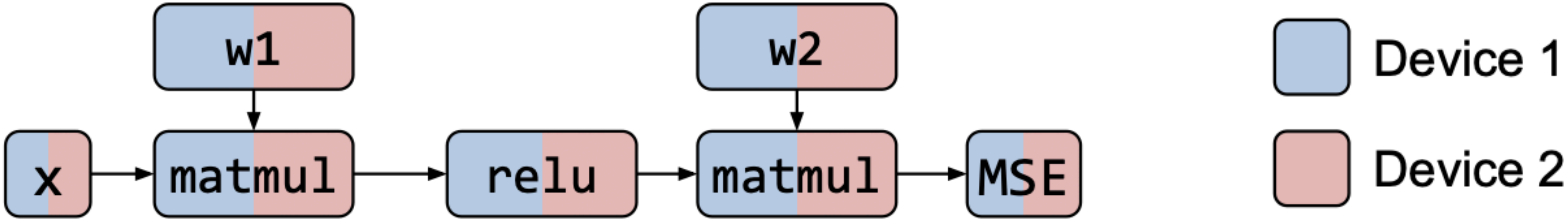


Image Source: https://sites.google.com/view/icml-2022-big-model

Motivation

- **Question.** Which parallelism should I adopt, for my own model & cluster?

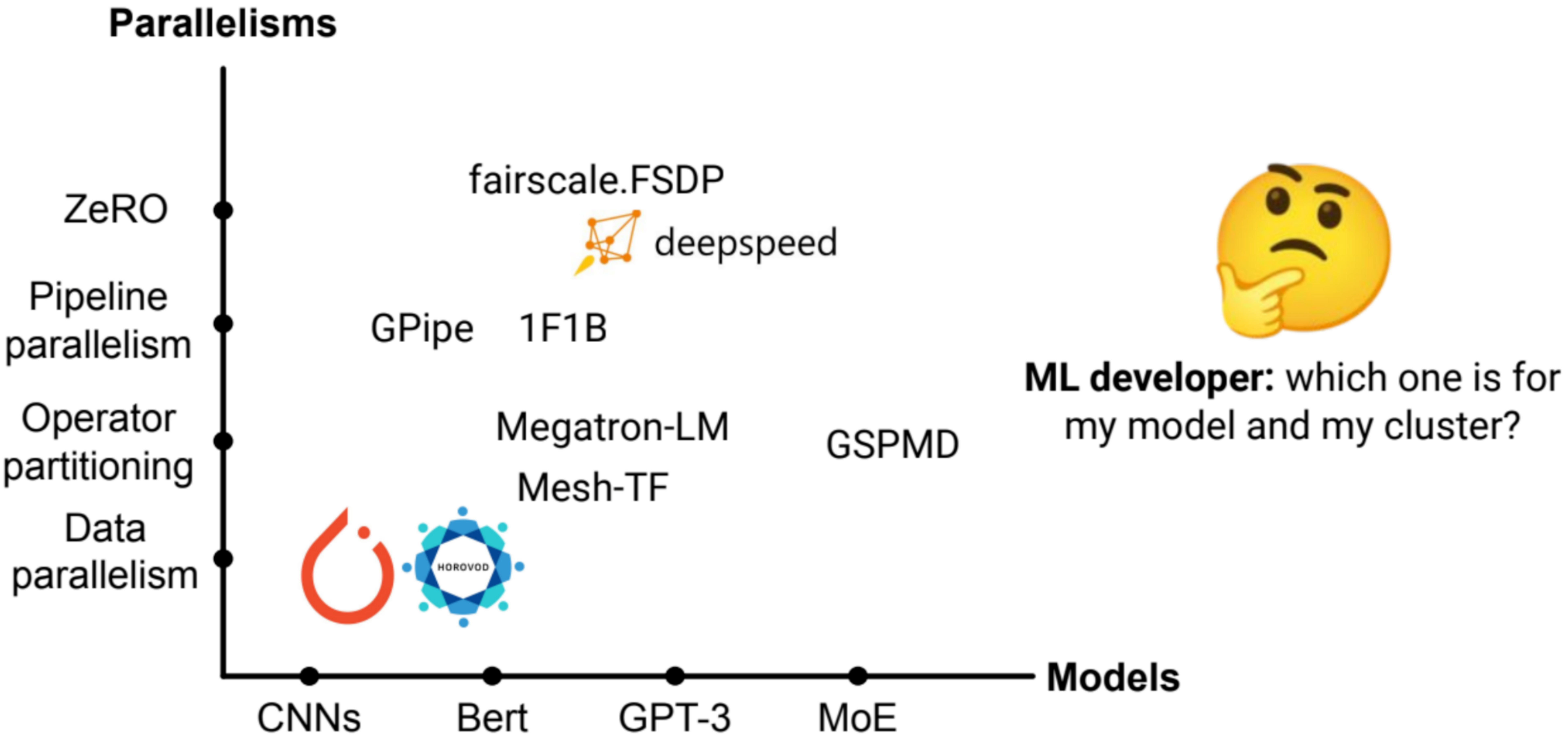


Image Source: https://sites.google.com/view/icml-2022-big-model

Formulation

- Abstractly put, we want to solve:

$$\min_{\text{strategy}} \; \text{Cost}(\text{model}, \text{cluster}; \text{strategy})$$

  - strategy is any possible combination of inter-op & intra-op parallelism

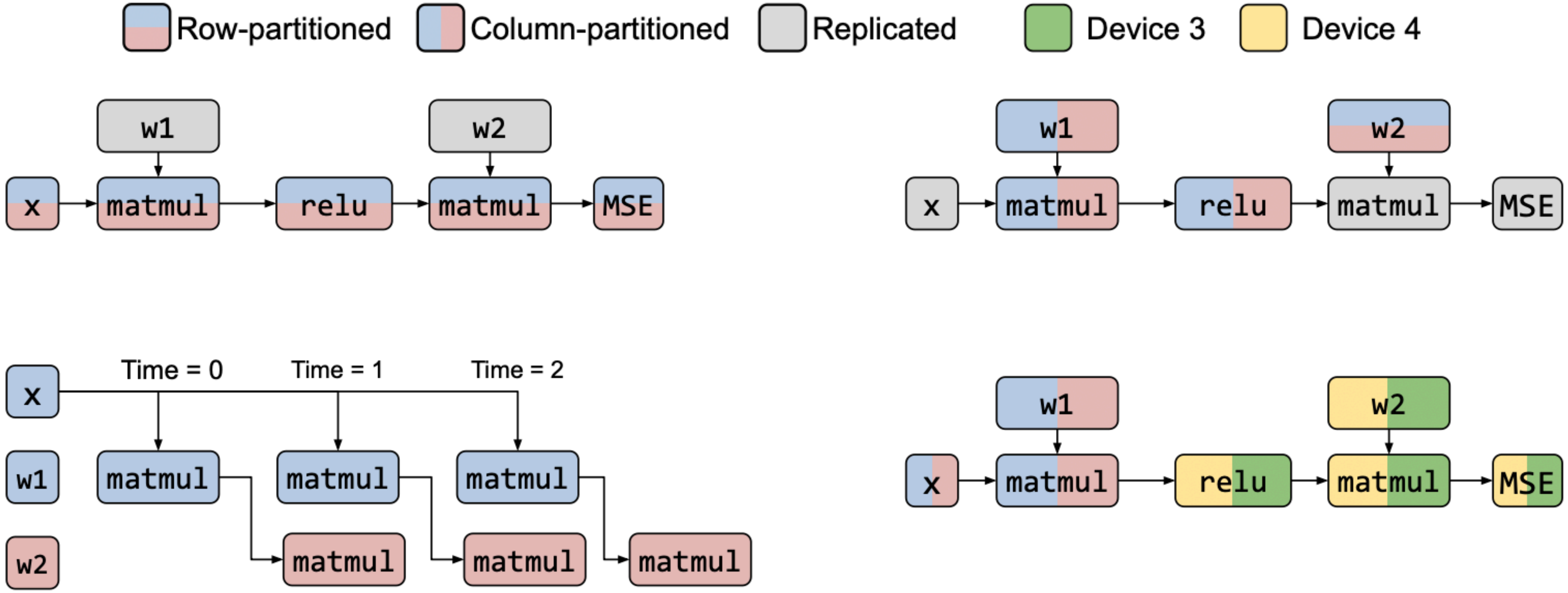


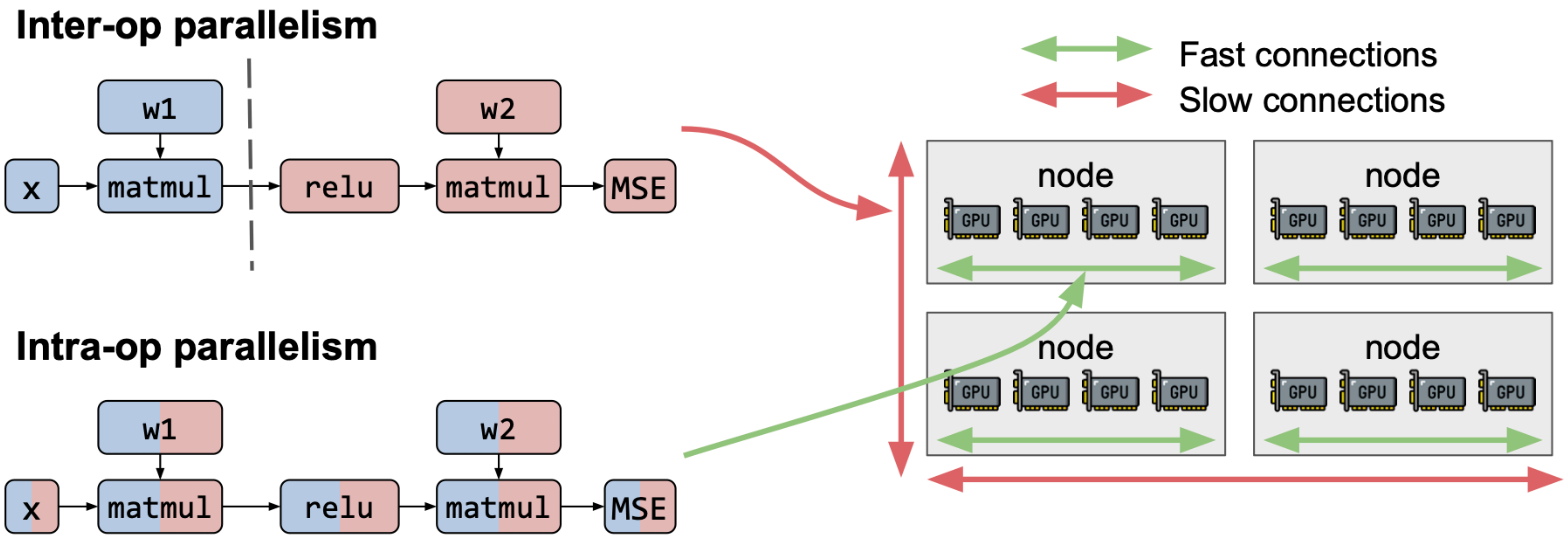
Image Source: https://sites.google.com/view/icml-2022-big-model

Approaches

- There are quite many approaches:
    - **MCMC.** FlexFlow (2018)
    - **RL.** ColocRL (2017)
    - (…)
- A popular approach is called **Alpa**
    - Hierarchical optimization-based method

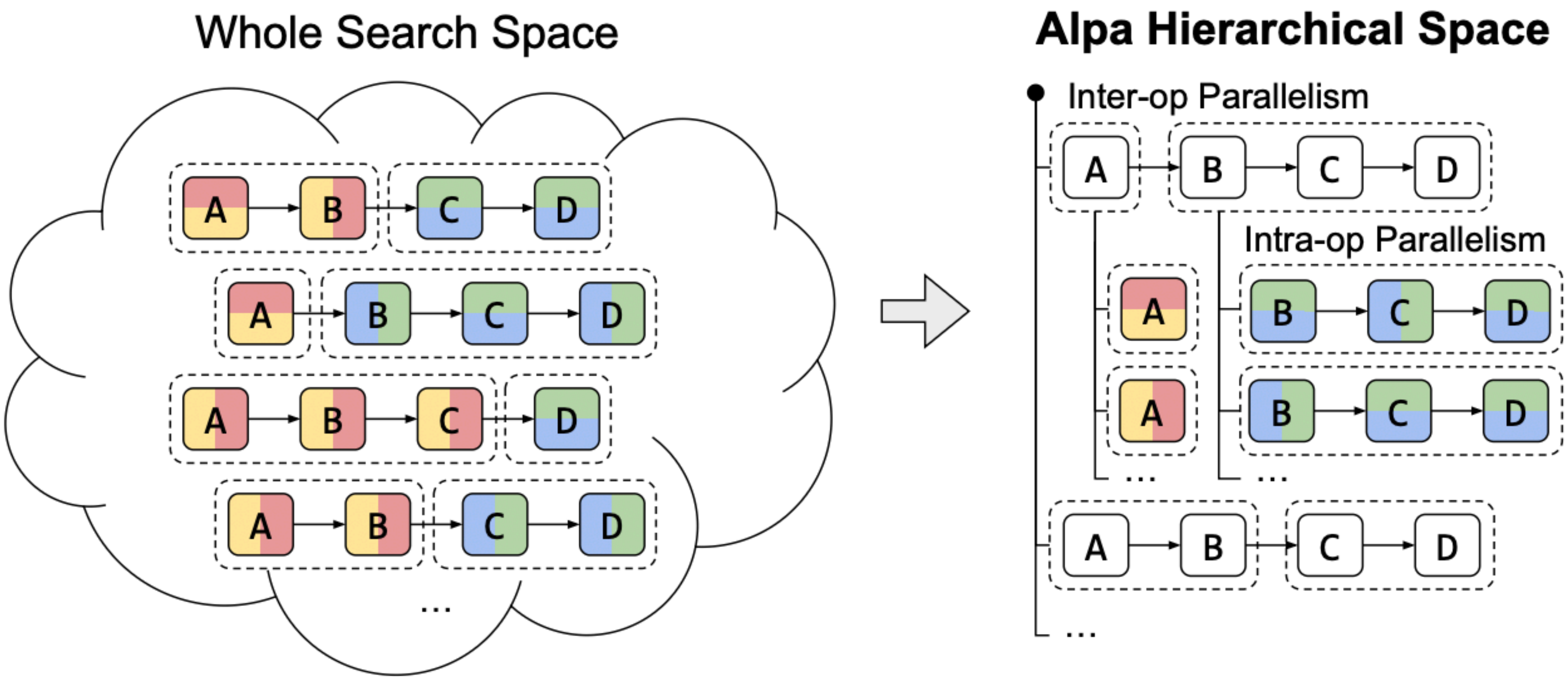
Alpa

- Prioritize performing:
  - **Inter-op.** Between nodes (as it requires less comm)
  - **Intra-op.** Between devices, inside a node (as it requires more comm)



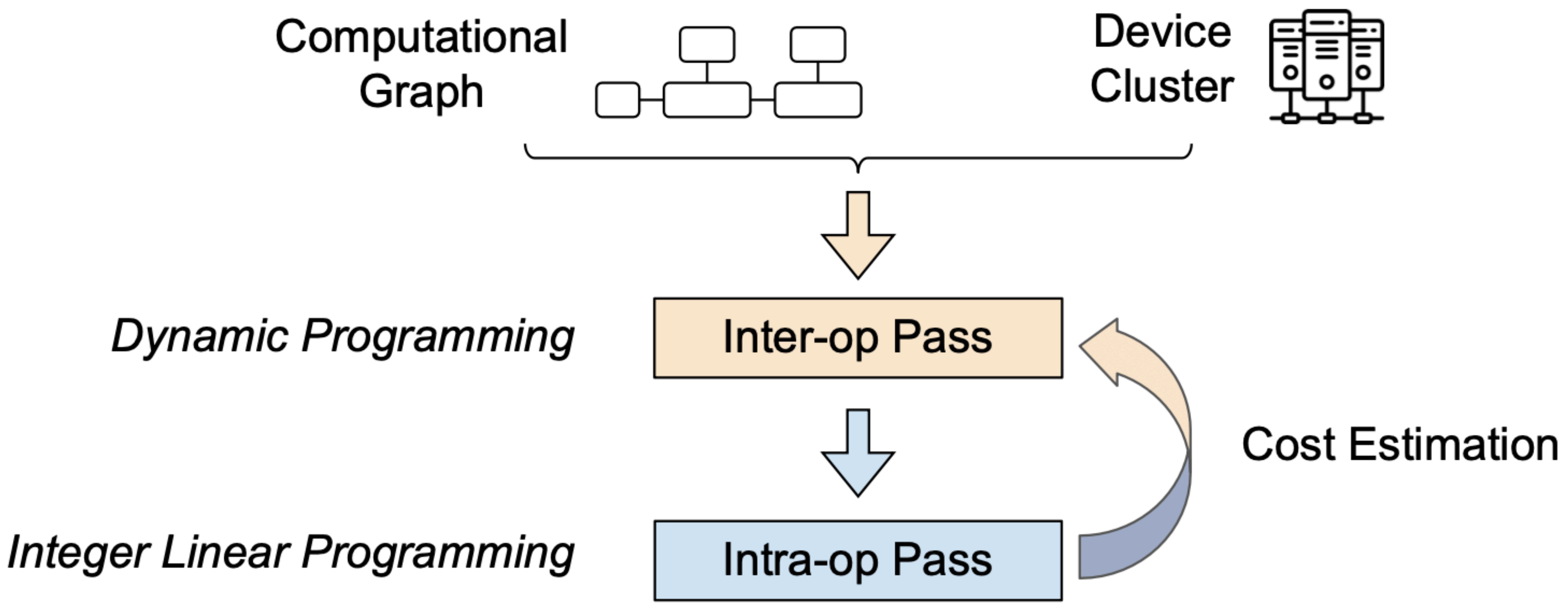
Alpa

- The search space thus becomes smaller and structured



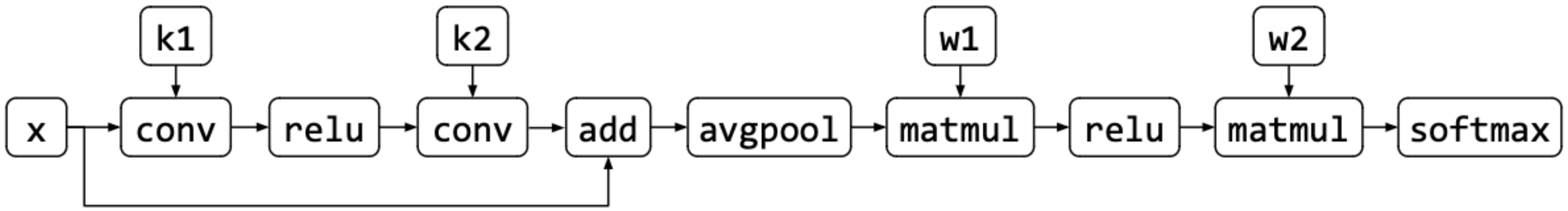
Alpa

- Roughly, the search is done by a two-stage iterative optimization
  - **Inter-op.** Determine the group of ops to be done in a node
  - **Intra-op.** How to conduct tensor/data parallel inside a node

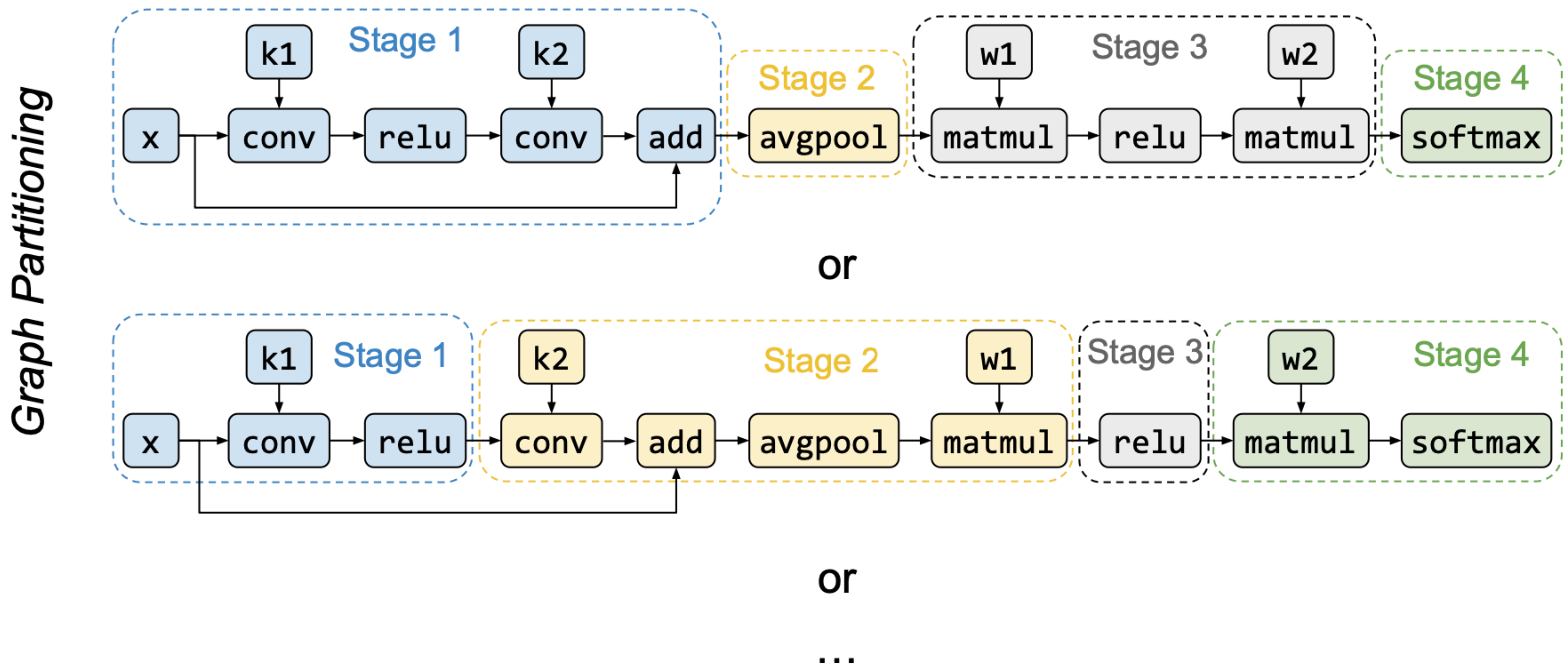


Inter-op

- Given a computational graph,

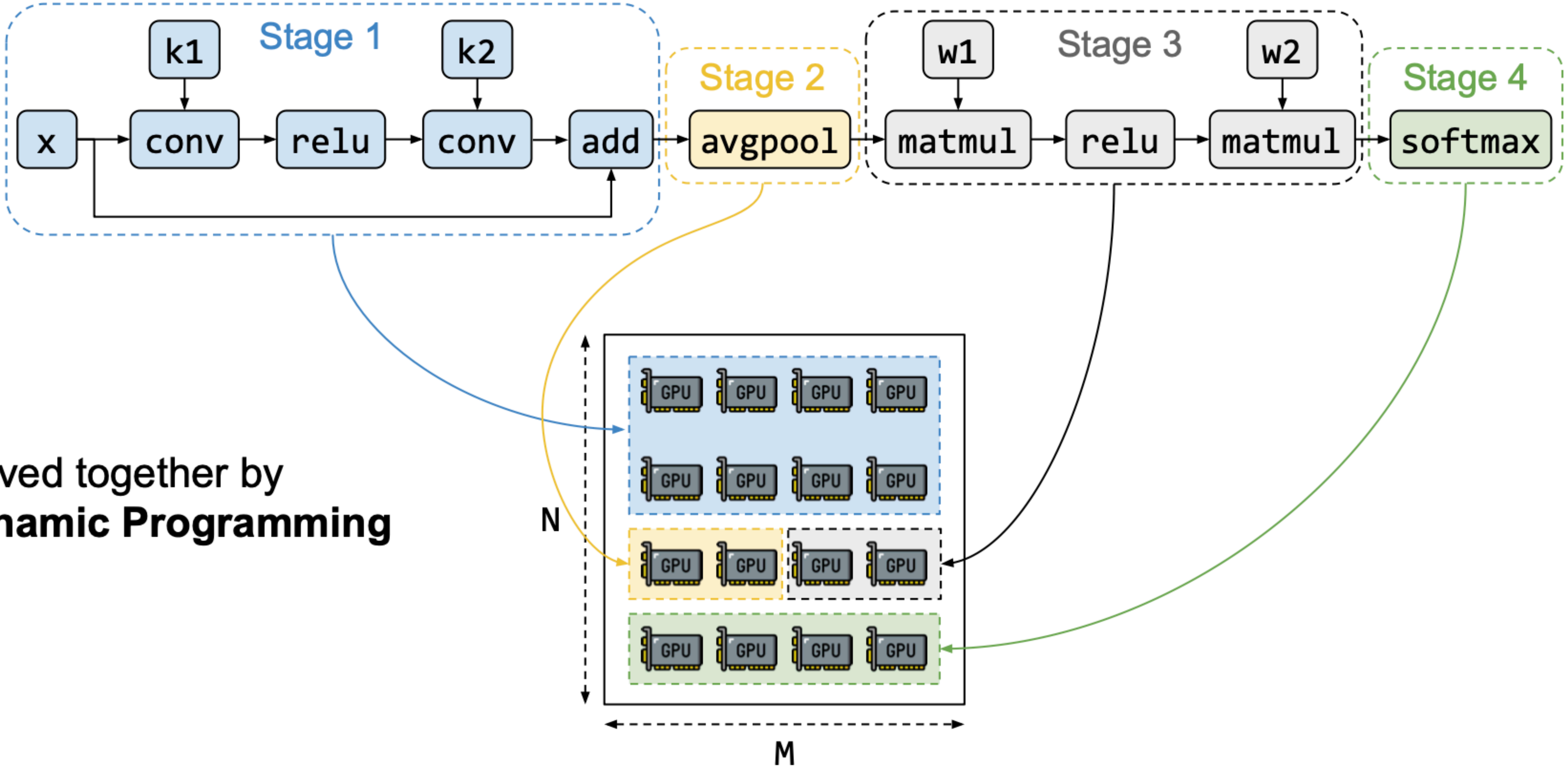


- Determine the partition of the graph



Inter-op

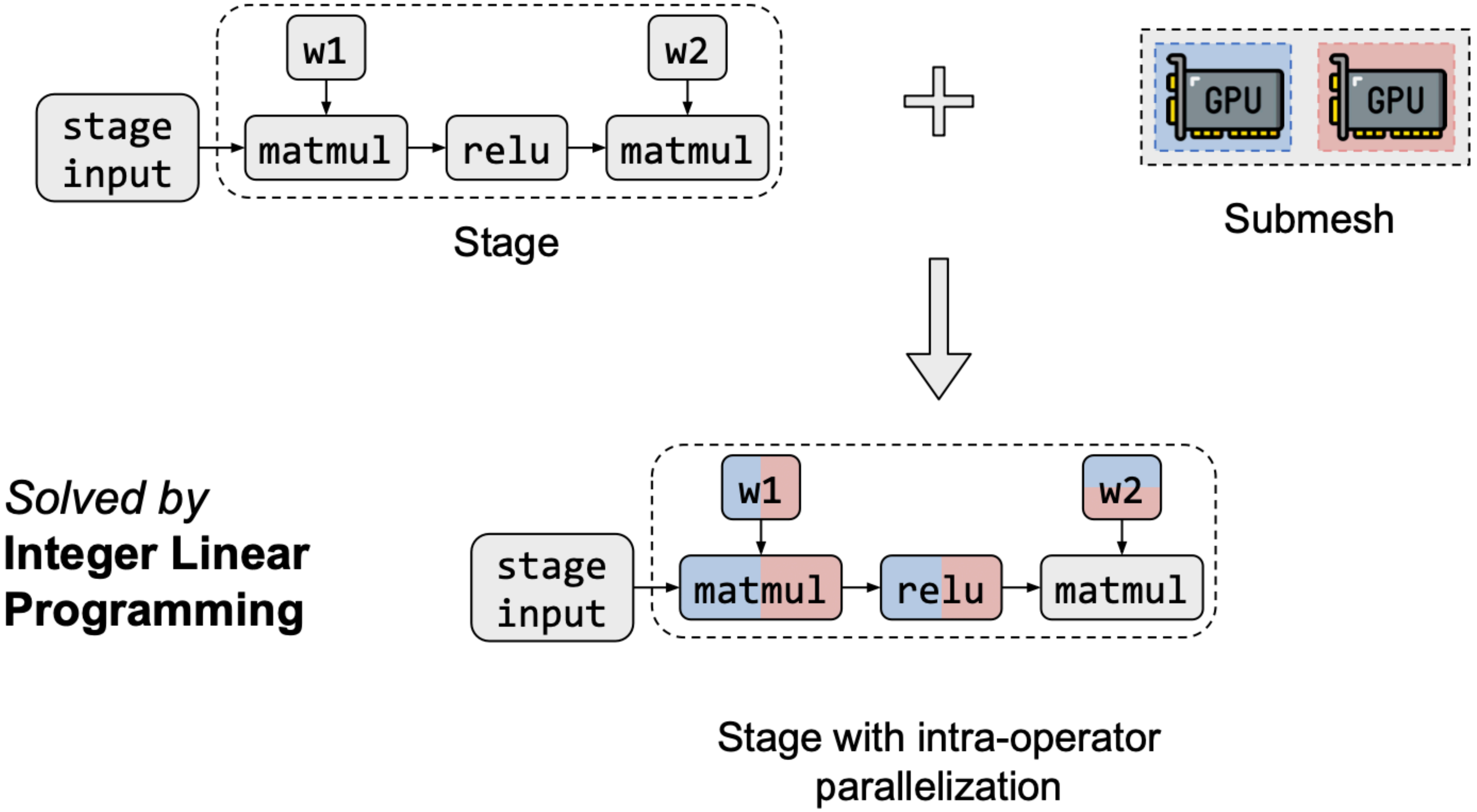
- Then, assign the nodes for each partition, via dynamic programming
  - **Required.** For this to be accurate, need a good latency estimate of each partition on the nodes



Solved together by
Dynamic Programming

Intra-op

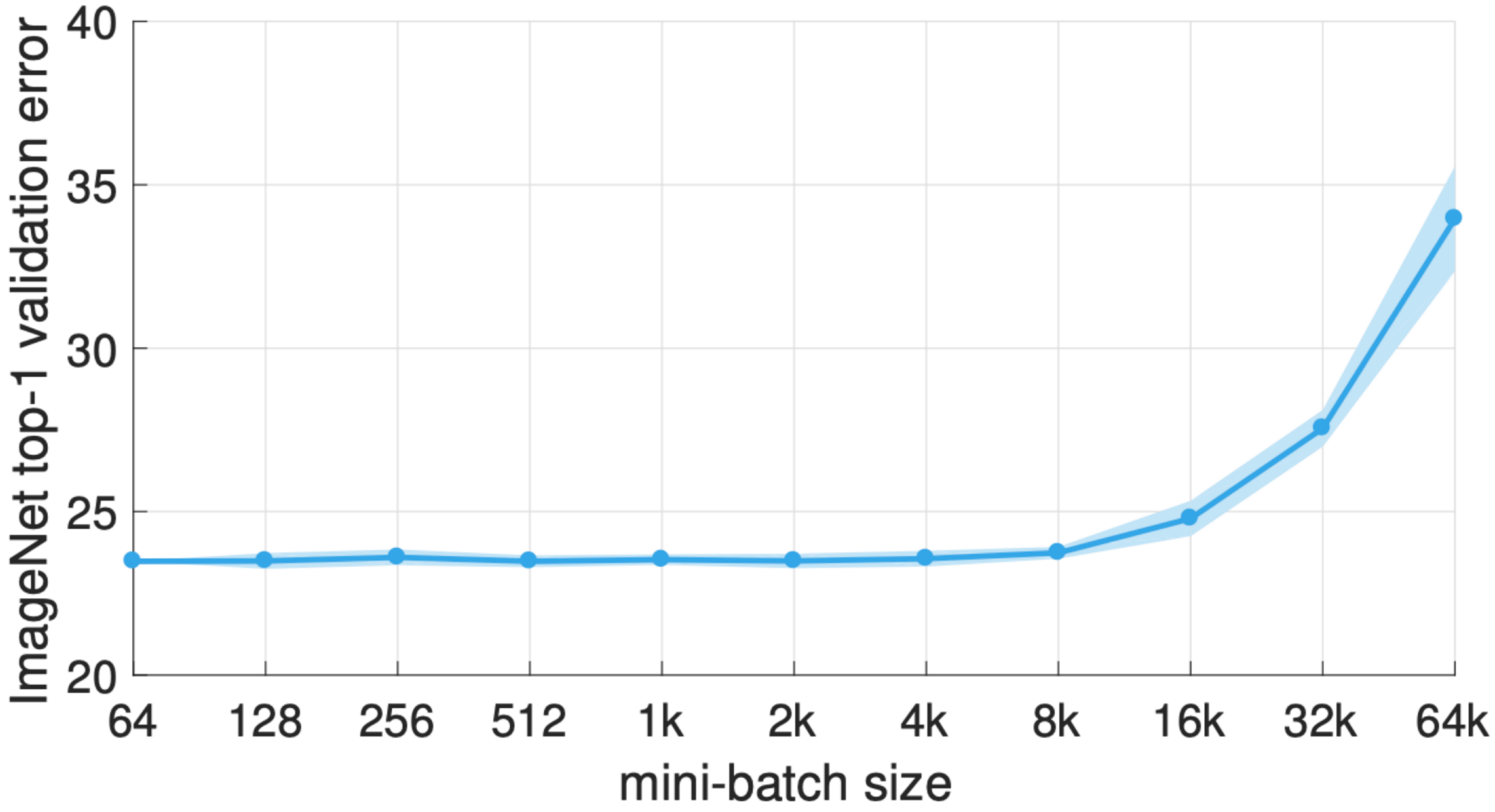
- In each intra-op pass, we solve an optimization problem
  - Assignment problem (discrete decisions) with linear costs
    - a **mixed integer-linear programming**!



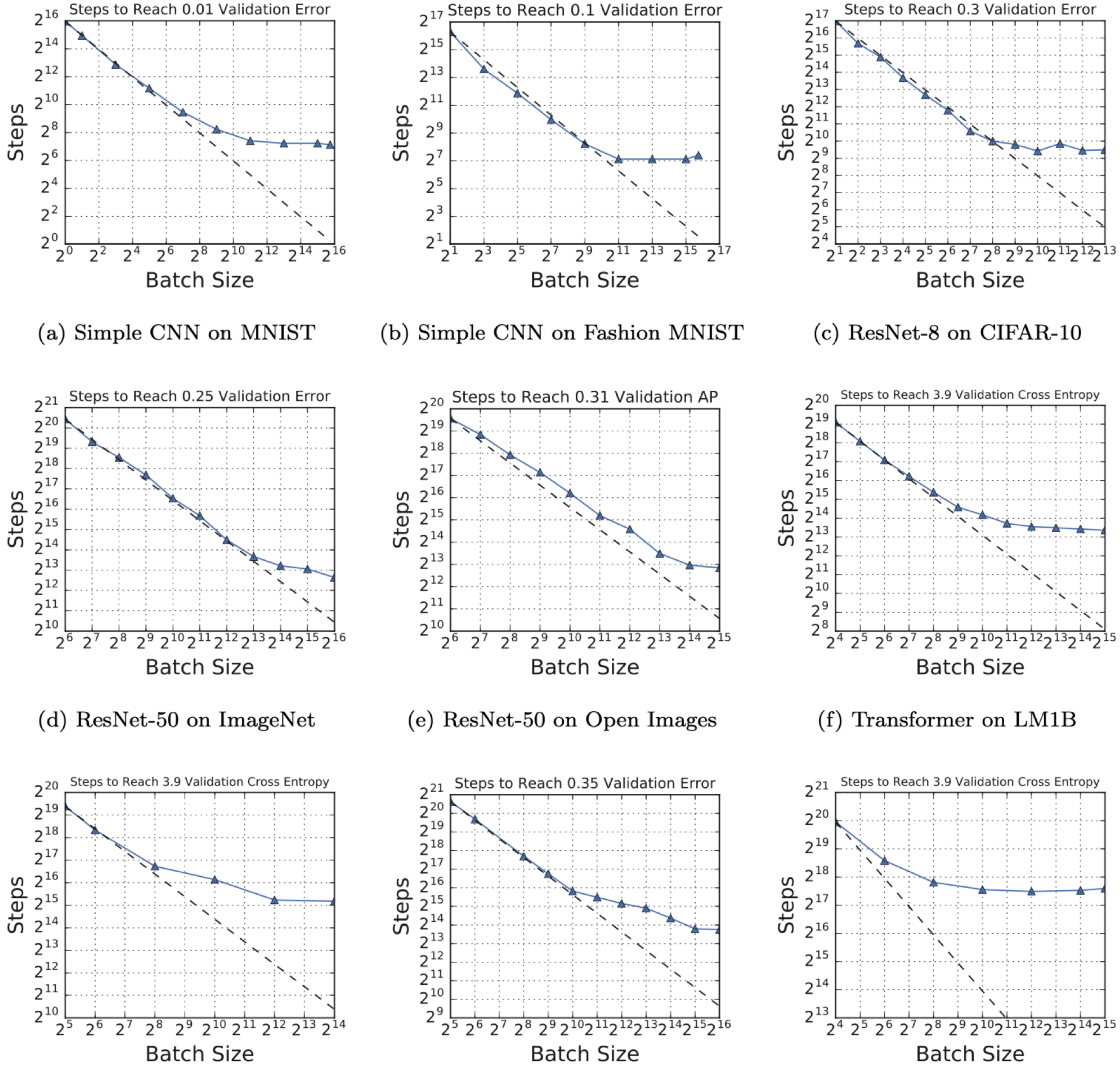
Remarks

Can we parallelize to infinity?

- Suppose that we can use infinite amount of GPUs
- **Question.** Can we make the batch size infinity, and finish training in seconds?
  - Answer. Unfortunately, no. We lose generalizability



Goyal et al., "Accurate, large mini-batch SGD: Training ImageNet in 1 hour" arXiv 2017

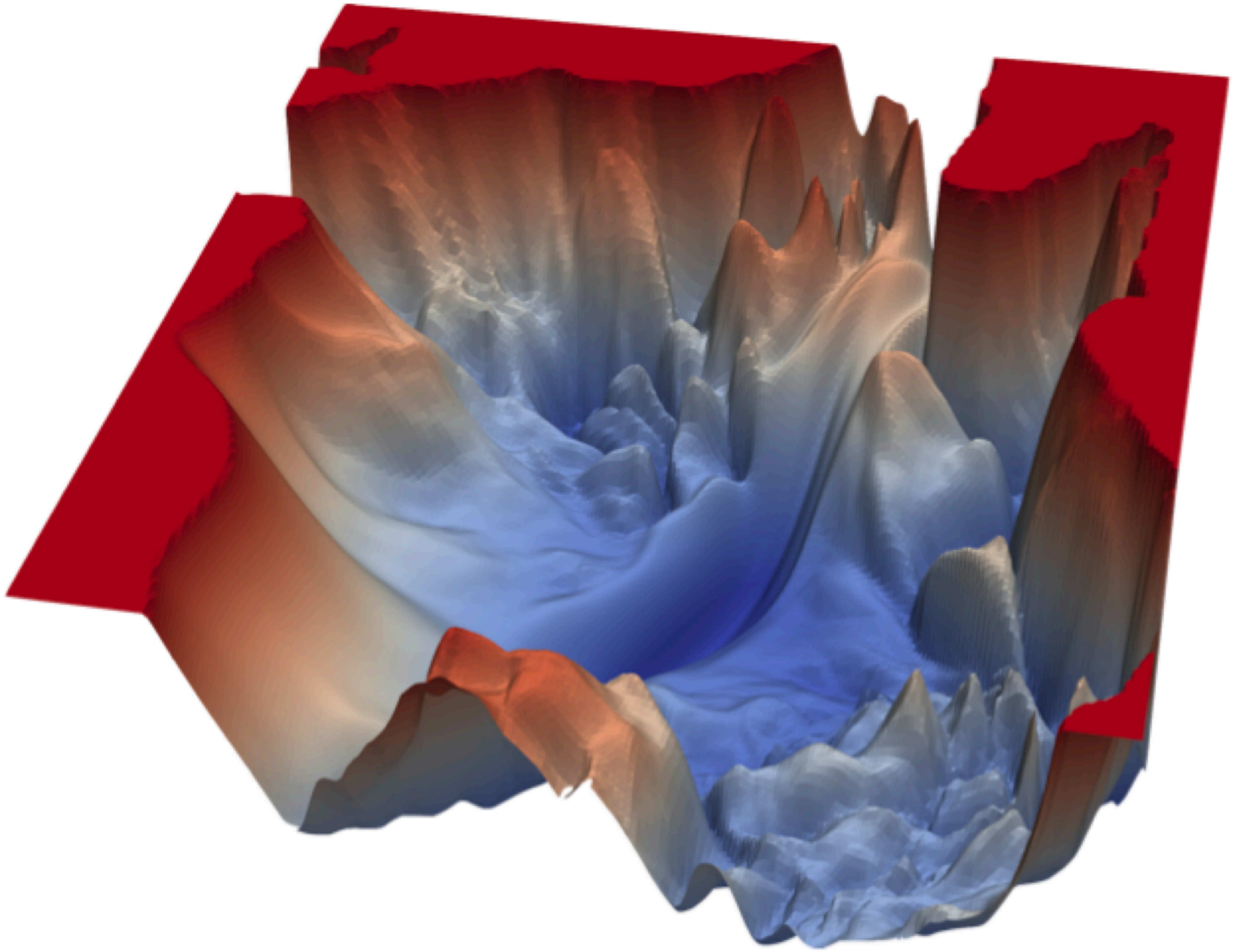


(a) Simple CNN on MNIST
(b) Simple CNN on Fashion MNIST
(c) ResNet-8 on CIFAR-10
(d) ResNet-50 on ImageNet
(e) ResNet-50 on Open Images
(f) Transformer on LM1B
(g) Transformer on Common Crawl
(h) VGG-11 on ImageNet
(i) LSTM on LM1B

Shallue et al., “Measuring the effects of data parallelism on neural network training” JMLR 2019.

Why?

- No complete answer, but some speculations…
  - Large batch —> Small SGD noise —> Trapped in local minima (narrow valley)



That’s it for today 🙌