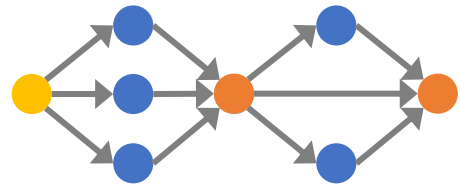


Lecture 25: Parallel Deep Learning (Model & Pipeline Parallelism)

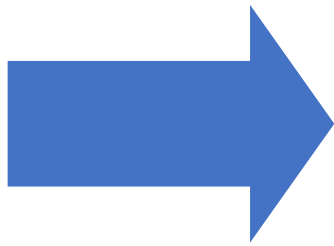
Parallel Computer Architecture and Programming

CMU 15-418/15-618, Fall 2023

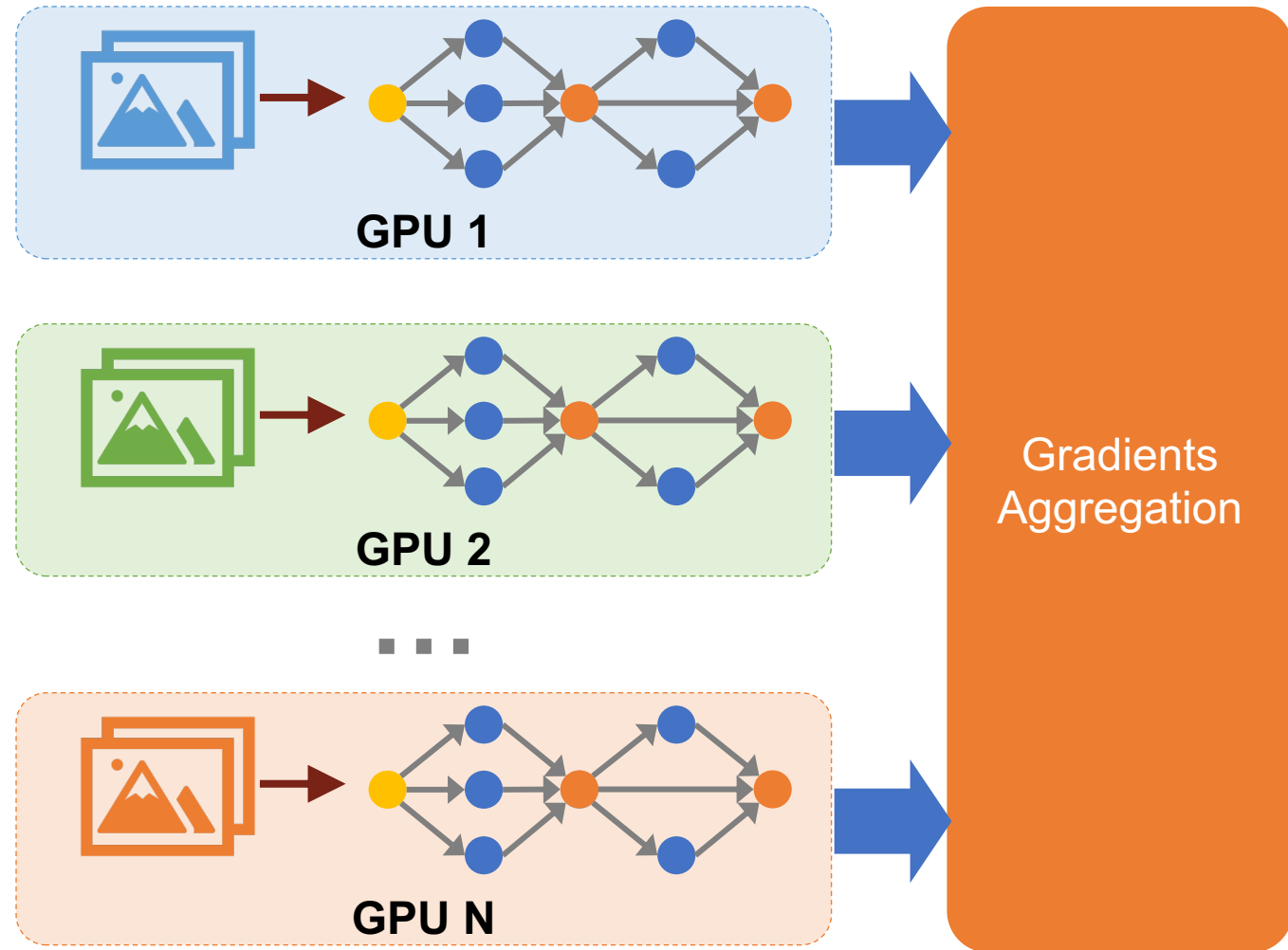
Recap: Data Parallelism



Training Dataset



$$w_i := w_i - \gamma \nabla L(w_i) = w_i - \frac{\gamma}{n} \sum_{j=1}^n \nabla L_j(w_i)$$



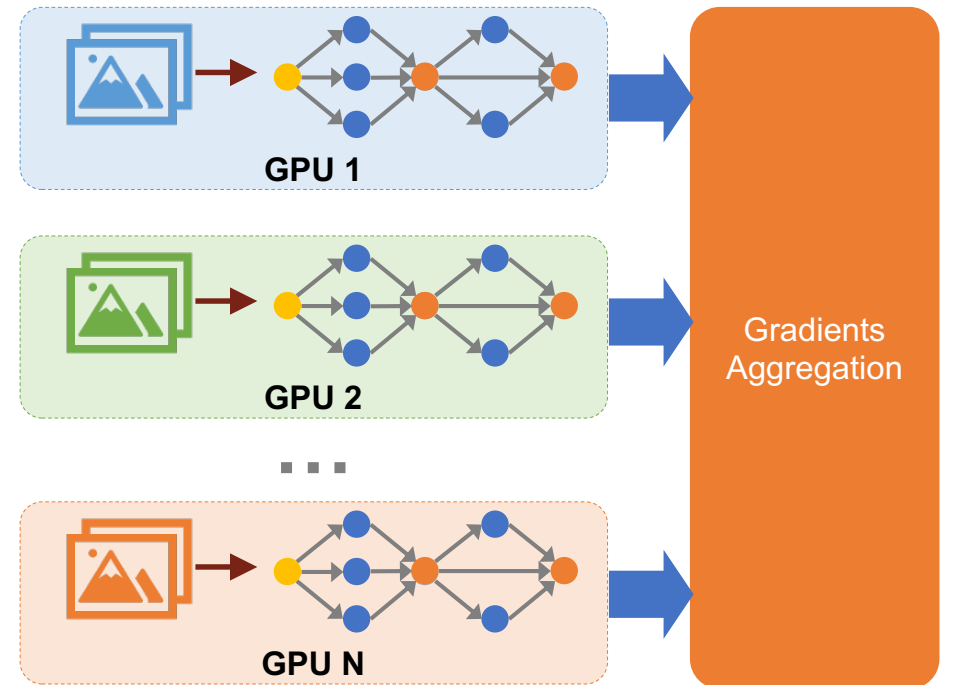
1. Partition training data into batches

2. Compute the gradients of each batch on a GPU

3. Aggregate gradients across GPUs

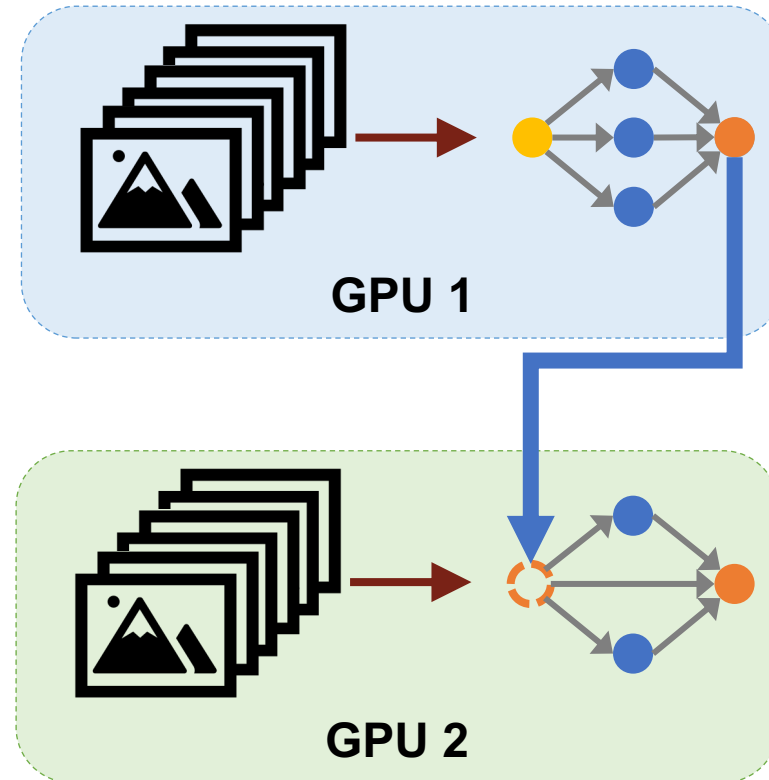
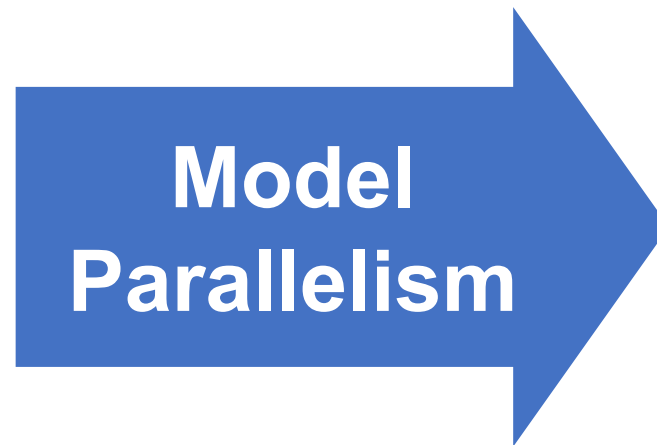
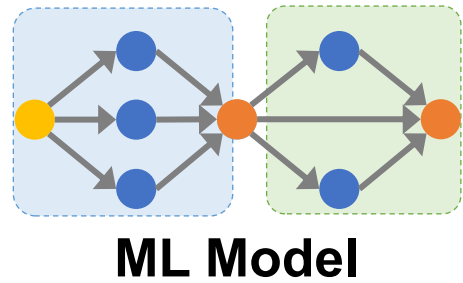
Recap: An Issue with Data Parallelism

- Each GPU saves a replica of the entire model
- Cannot train large models that exceed GPU device memory



Model Parallelism

- Split a model into multiple subgraphs and assign them to different devices

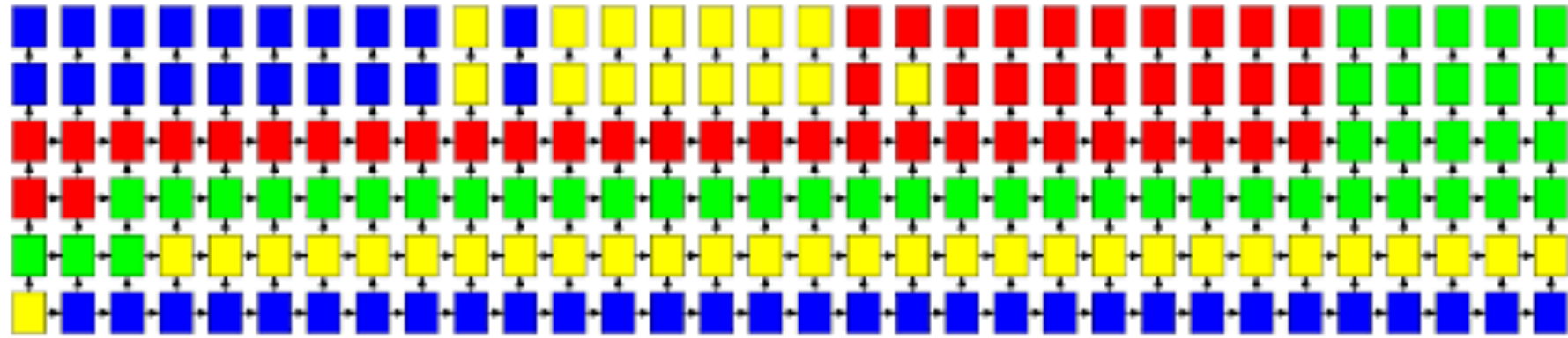


Transfer intermediate results between devices

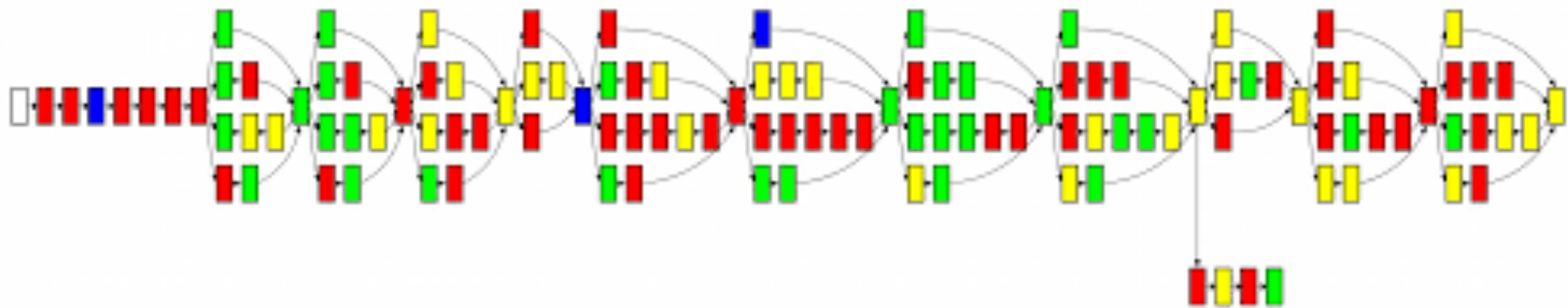


$$w_i := w_i - \gamma \nabla L(w_i) = w_i - \frac{\gamma}{n} \sum_{j=1}^n \nabla L_j(w_i)$$

Device Placement for Model Parallelism is Challenging

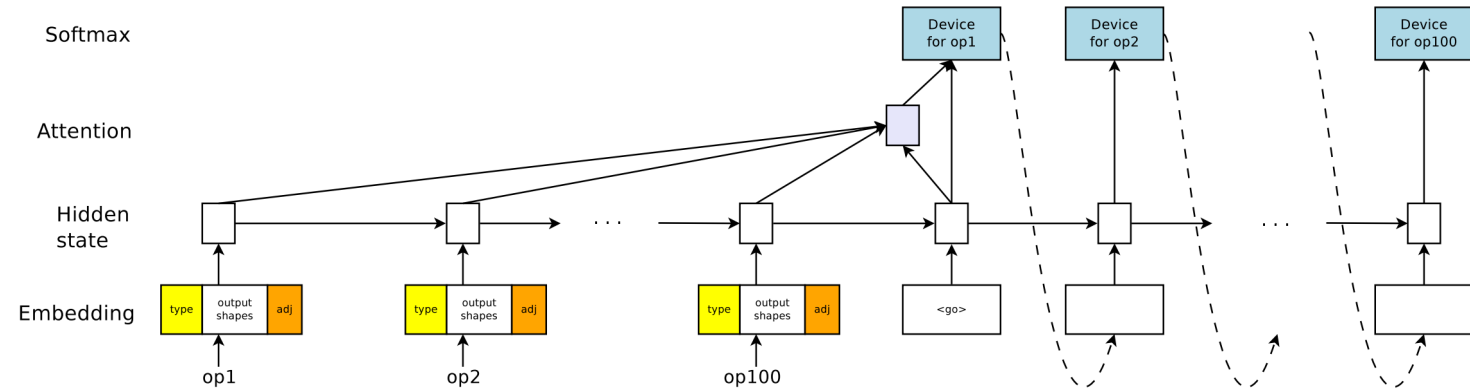
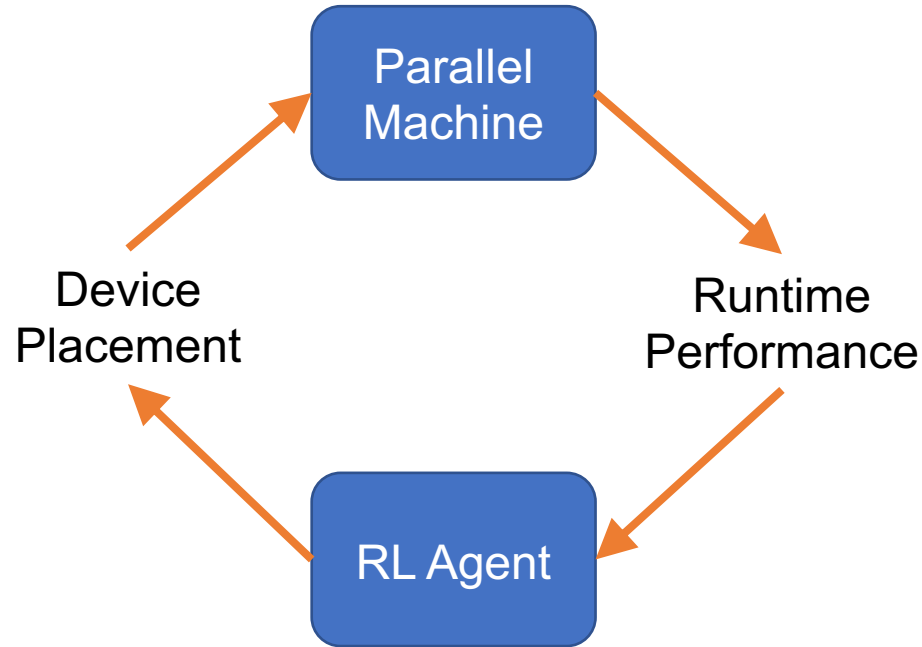


Model parallelism: training a recurrent neural network on 4 GPUs



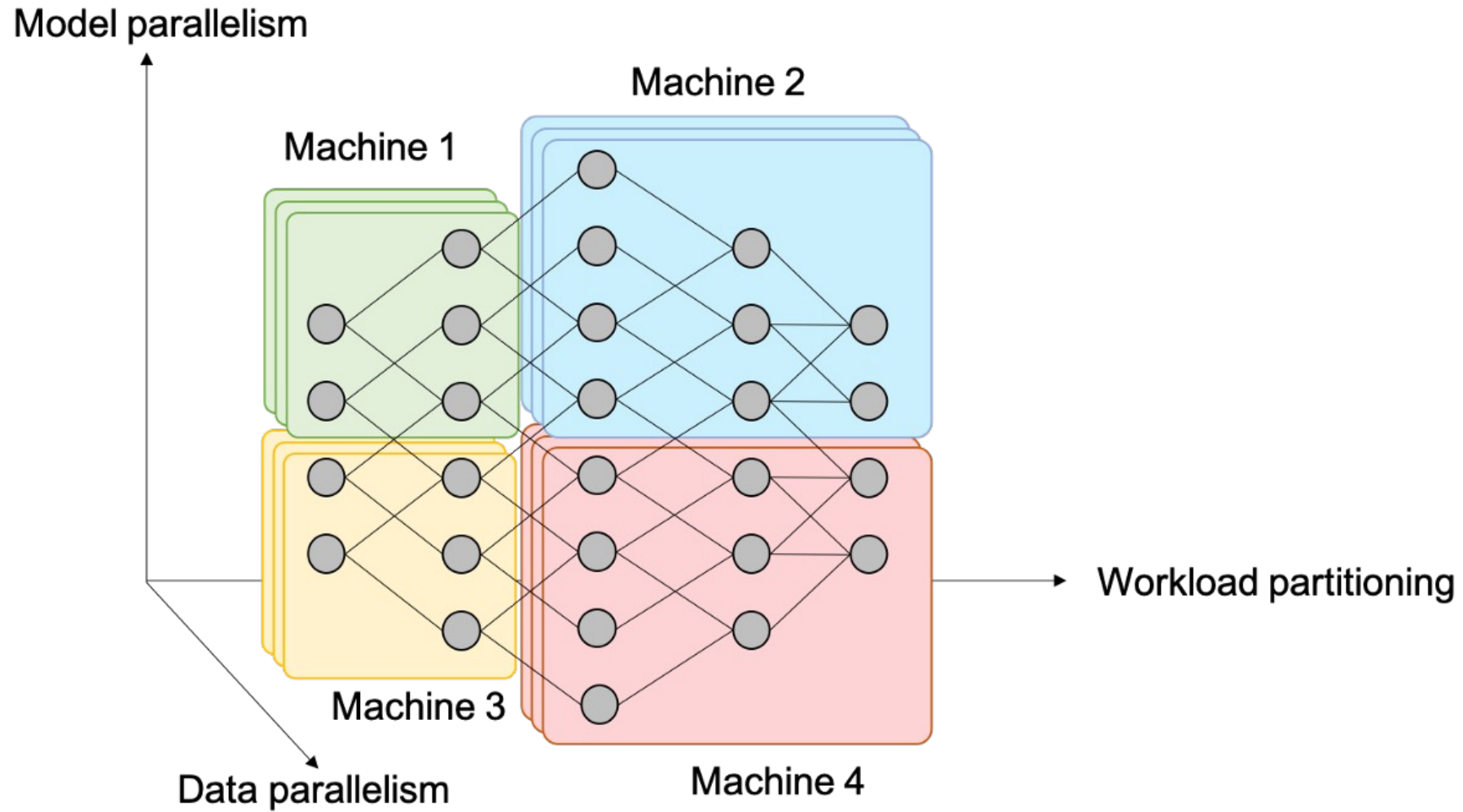
Model parallelism: training a conventional neural network on 4 GPUs

Using ML to Optimize Device Placement for ML



ColocRL's neural architecture

Combine Data and Model Parallelism

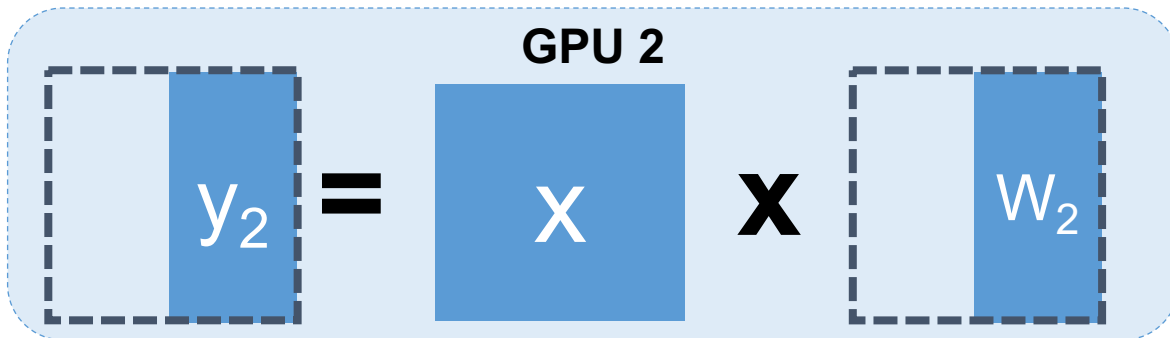
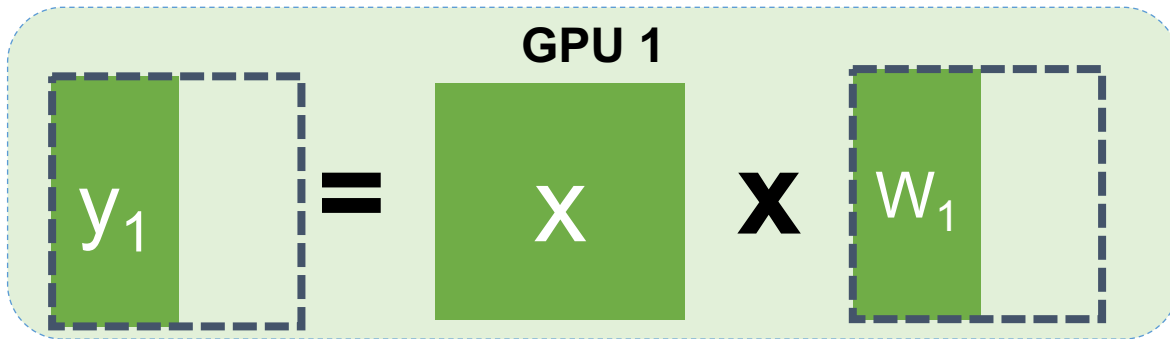


Tensor Model Parallelism

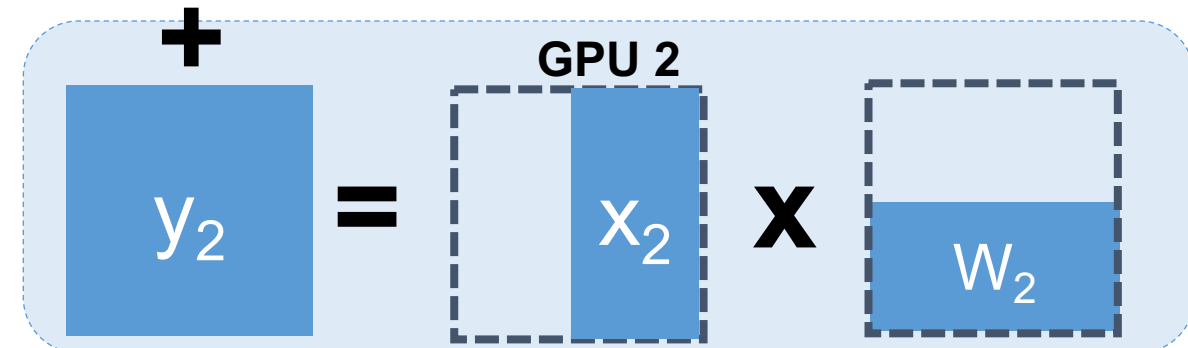
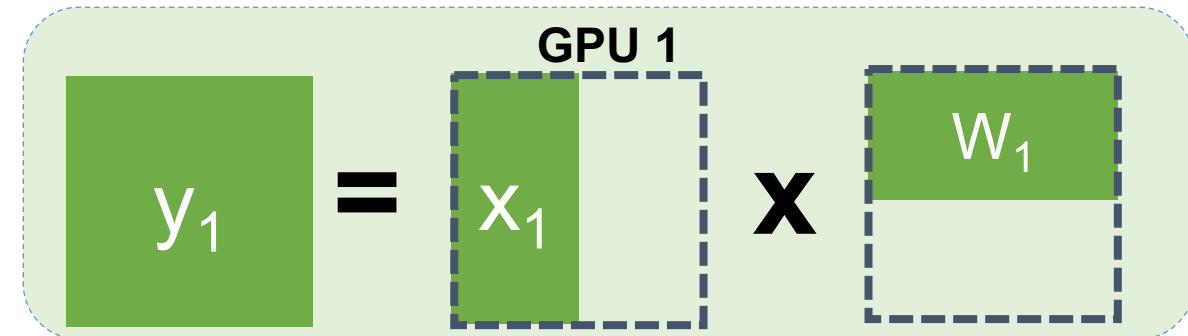
$$y = X \times W$$

output input parameters

- Partition parameters/gradients *within* a layer



Tensor Model Parallelism (partition output)

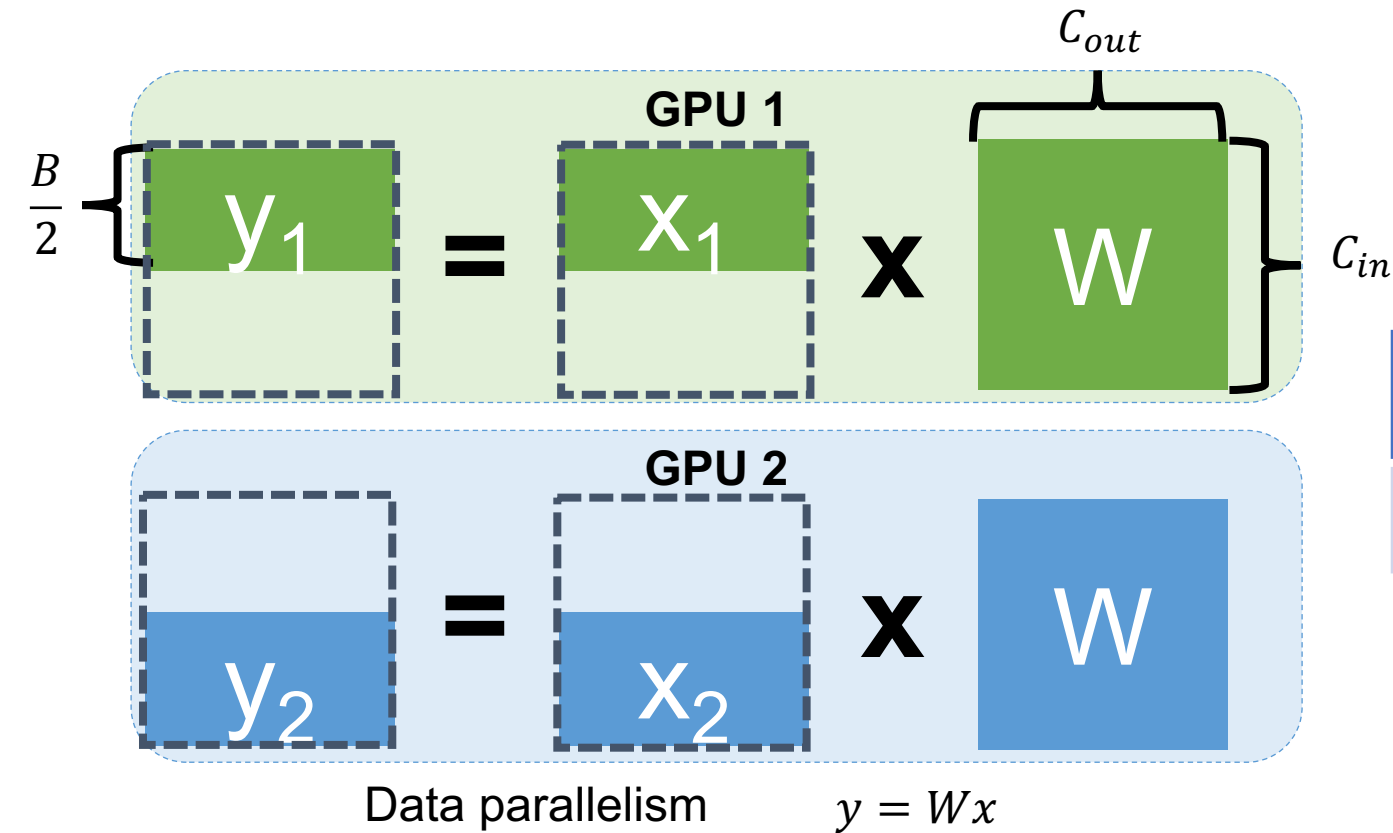


Tensor Model Parallelism (reduce output)

$$y = y_1 + y_2$$

Comparing Data and Tensor Model Parallelism

Diagram illustrating the matrix equation $y = x \cdot W$. The output vector y has dimension B . The input vector x and weight matrix W have dimensions C_{out} and C_{in} respectively.

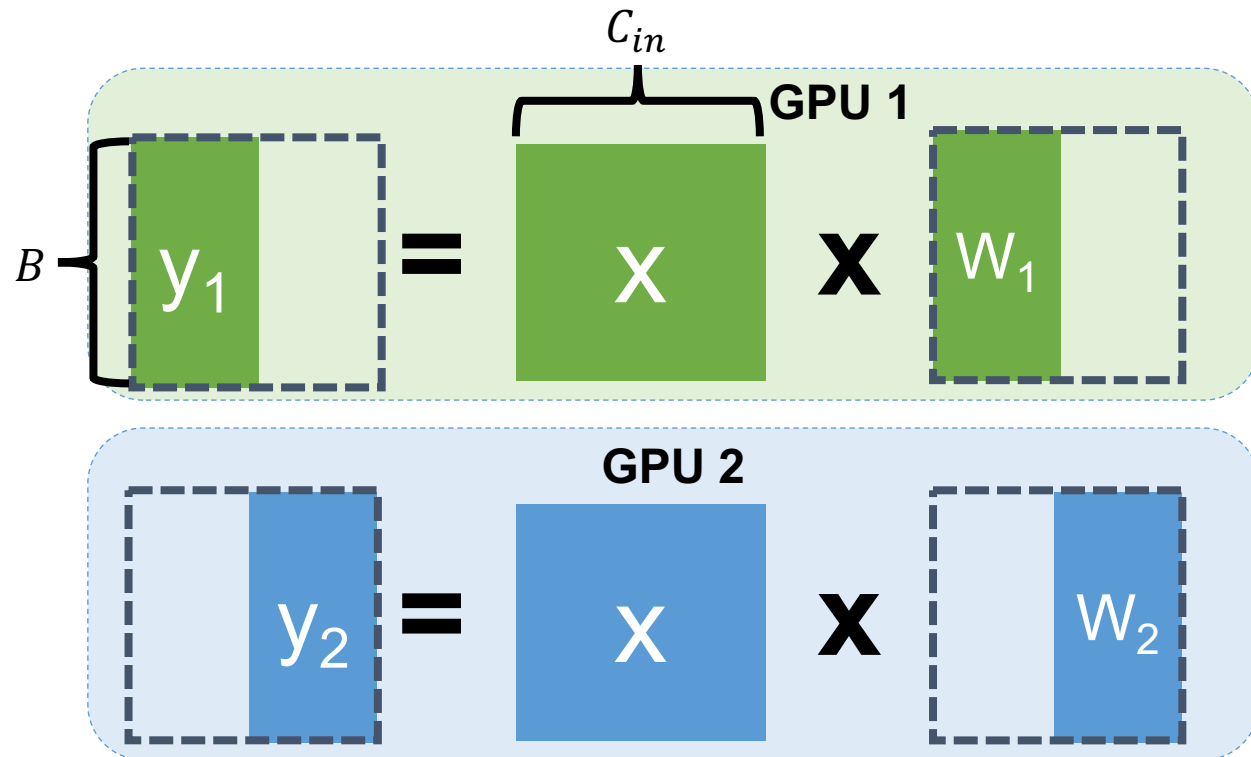


Forward Processing	Backward Propagation	Gradients Sync
0	0	$2 * C_{out} * C_{in}$

Communication Cost of Data Parallelism

Comparing Data and Tensor Model Parallelism

$$\begin{matrix} & & & C_{out} \\ & & & \overbrace{\hspace{2cm}} \\ B \left\{ \begin{matrix} \color{yellow} \square \\ y \end{matrix} \right. & = & \color{yellow} \square & \times & \color{yellow} \square & \left. \vphantom{\begin{matrix} \color{yellow} \square \\ y \end{matrix}} \right\} C_{in} \\ & & & & & \underbrace{\hspace{2cm}}
 \end{matrix}$$



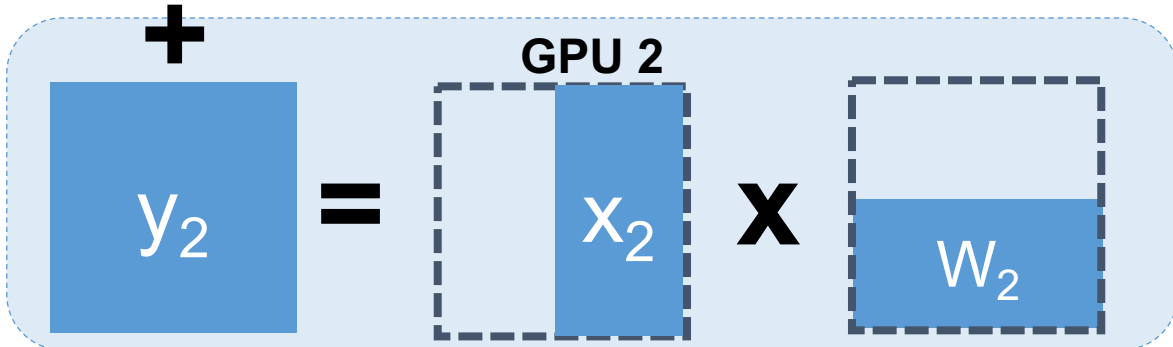
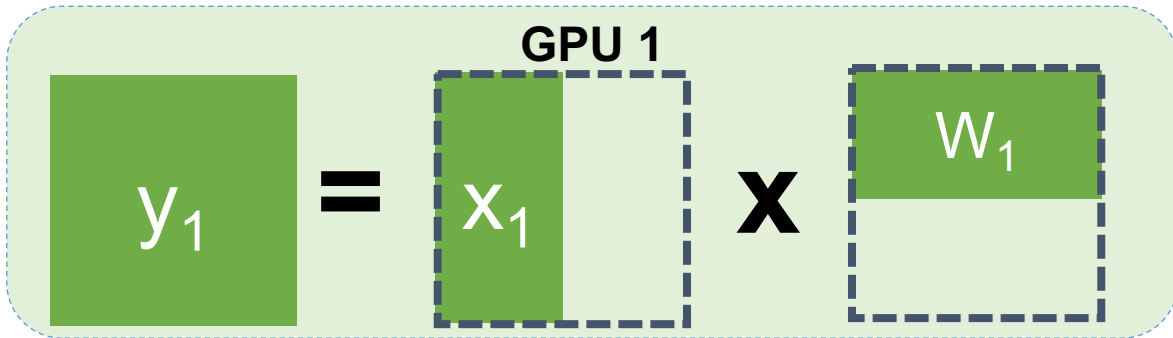
Tensor Model Parallelism (partition output)

Forward Processing	Backward Propagation	Gradients Sync
$B * C_{in}$	$B * C_{in}$	0

Communication Cost of Tensor Model Parallelism

Comparing Data and Tensor Model Parallelism

$$B \left[\begin{array}{c} y \end{array} \right] = X \times \begin{array}{c} C_{out} \\ \left[\begin{array}{c} W \end{array} \right] C_{in} \end{array}$$



Tensor Model Parallelism (Reduce output)

$$y = y_1 + y_2$$

Forward Processing	Backward Propagation	Gradients Sync
$2 * B * C_{out}$	0	0

Communication Cost of Tensor Model Parallelism

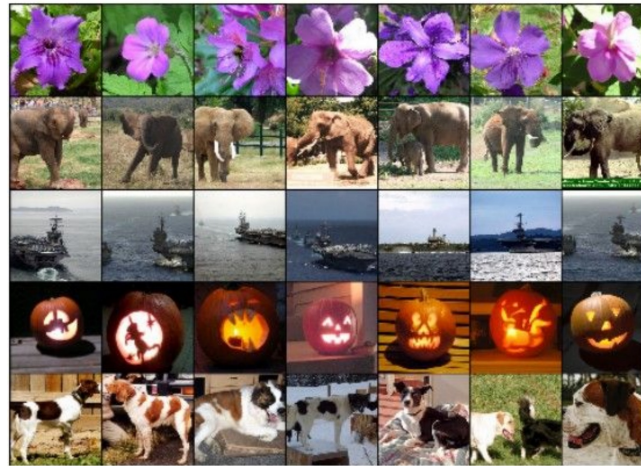
Comparing Data and Tensor Model Parallelism

- Data parallelism: $C_{out} * C_{in}$
- Tensor model parallelism (partition output): $B * C_{in}$
- Tensor model parallelism (reduce output): $B * C_{out}$
- **The best strategy depends on the model and underlying machine**

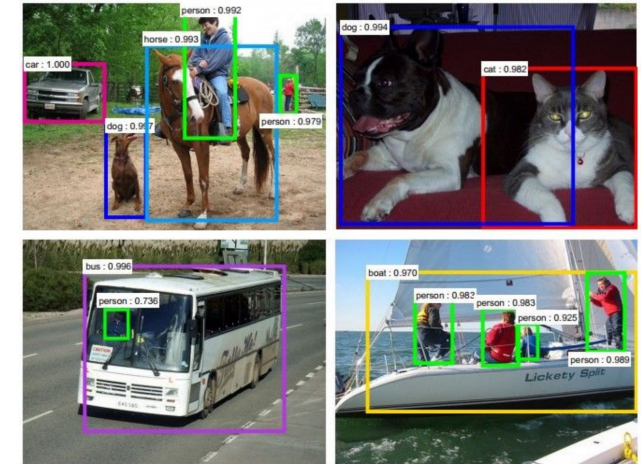
Example: Convolutional Neural Networks



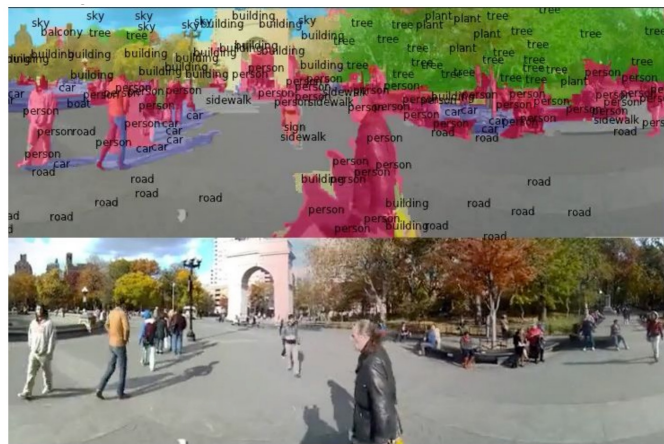
Classification



Retrieval



Detection



Segmentation



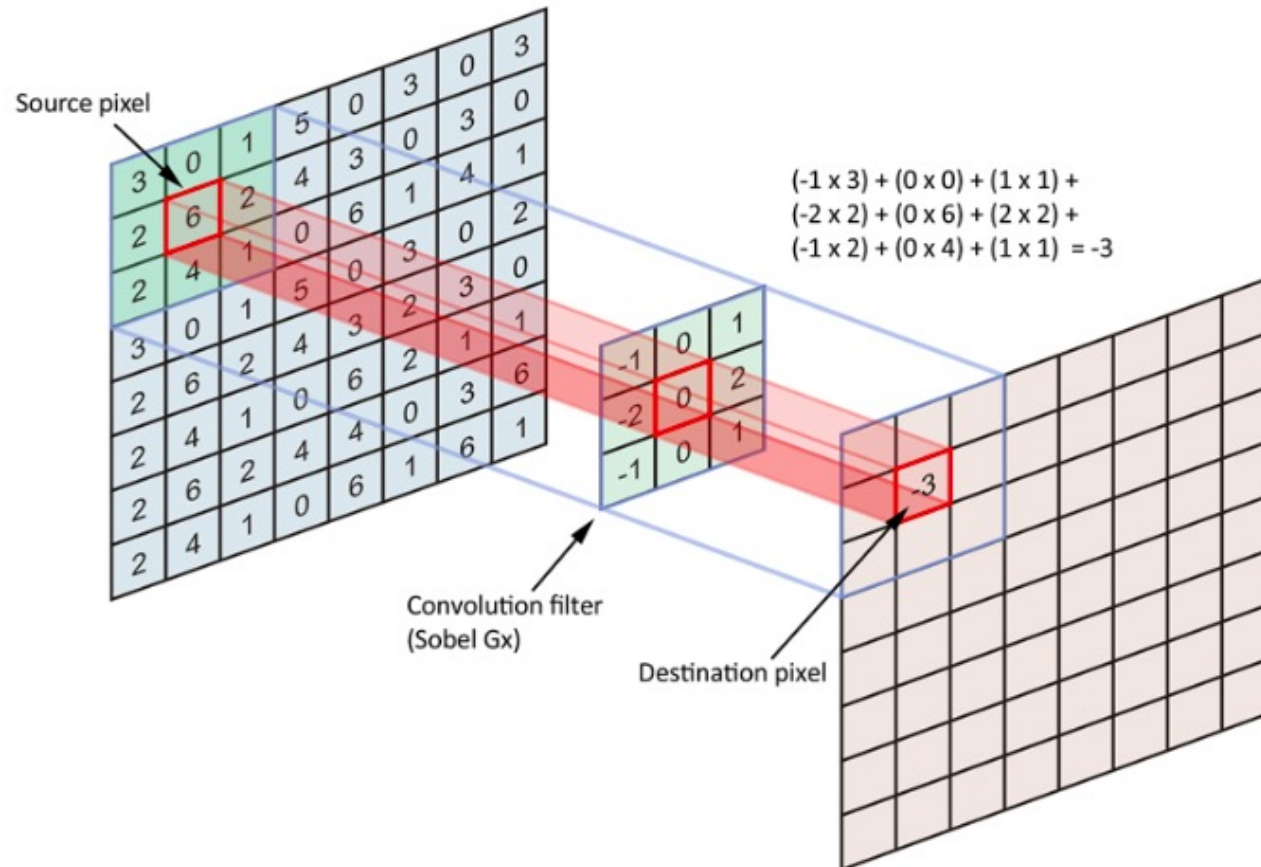
Self-Driving



Synthesis

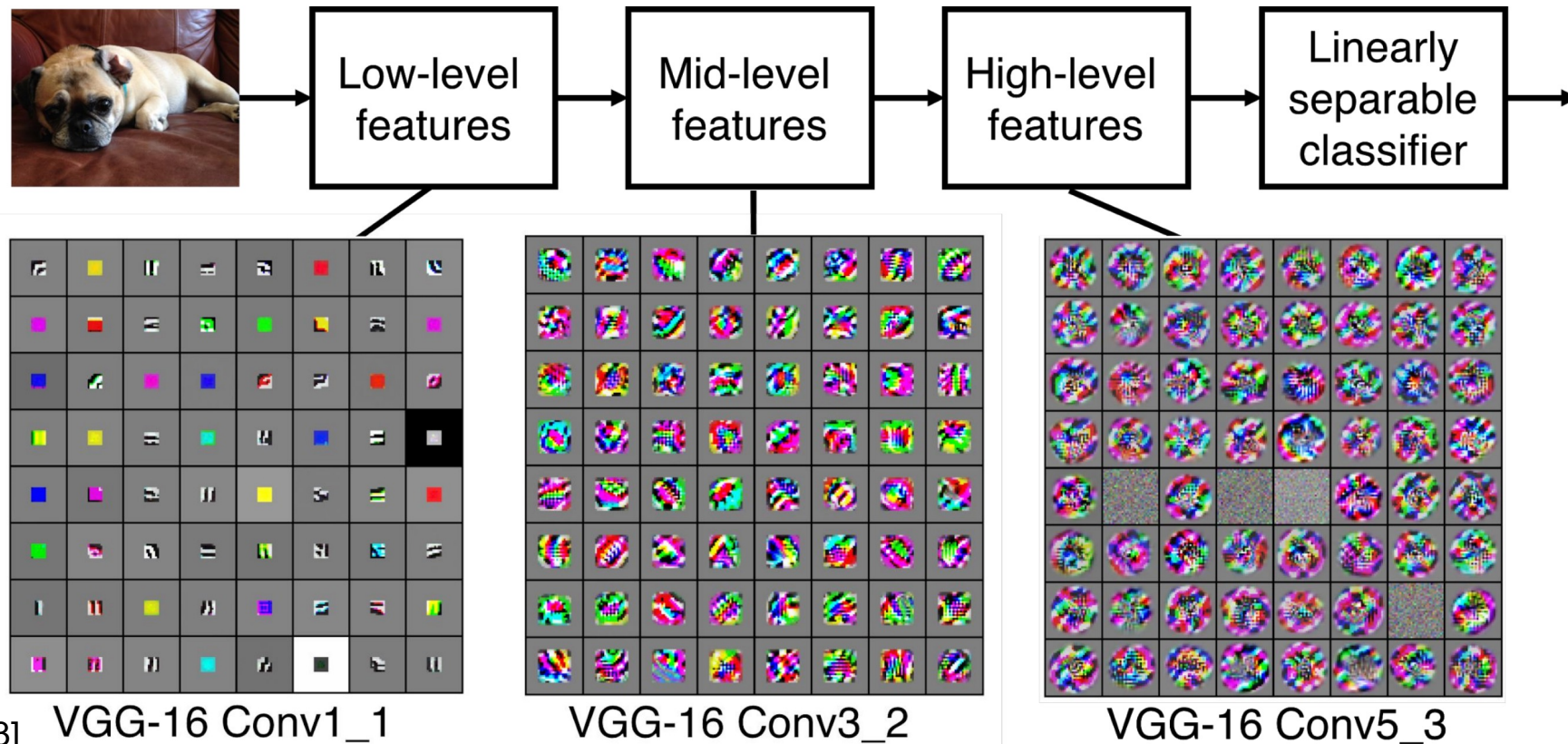
Convolution

- Convolve the filter with the image: slide over the image spatially and compute dot products



CNNs

- A sequence of convolutional layers, interspersed by pooling, normalization, and activation functions



Parallelizing Convolutional Neural Networks

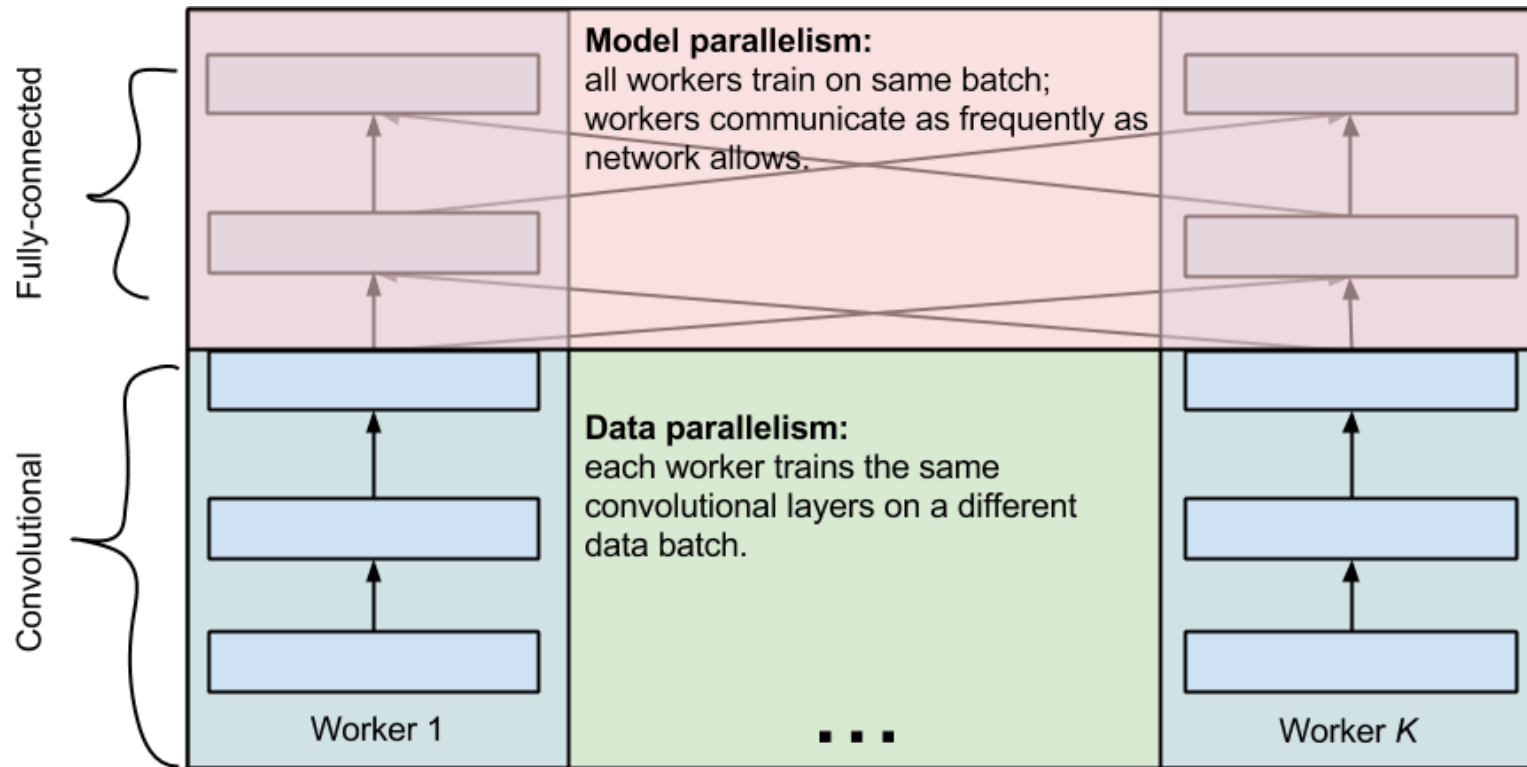
- Convolutional layers
 - 90-95% of the computation
 - 5% of the parameters
 - Very large intermediate activations
- Fully-connected layers
 - 5-10% of the computation
 - 95% of the parameters
 - Small intermediate activations
- **Discussion: how to parallelize CNNs?**

Data parallelism

Tensor model parallelism

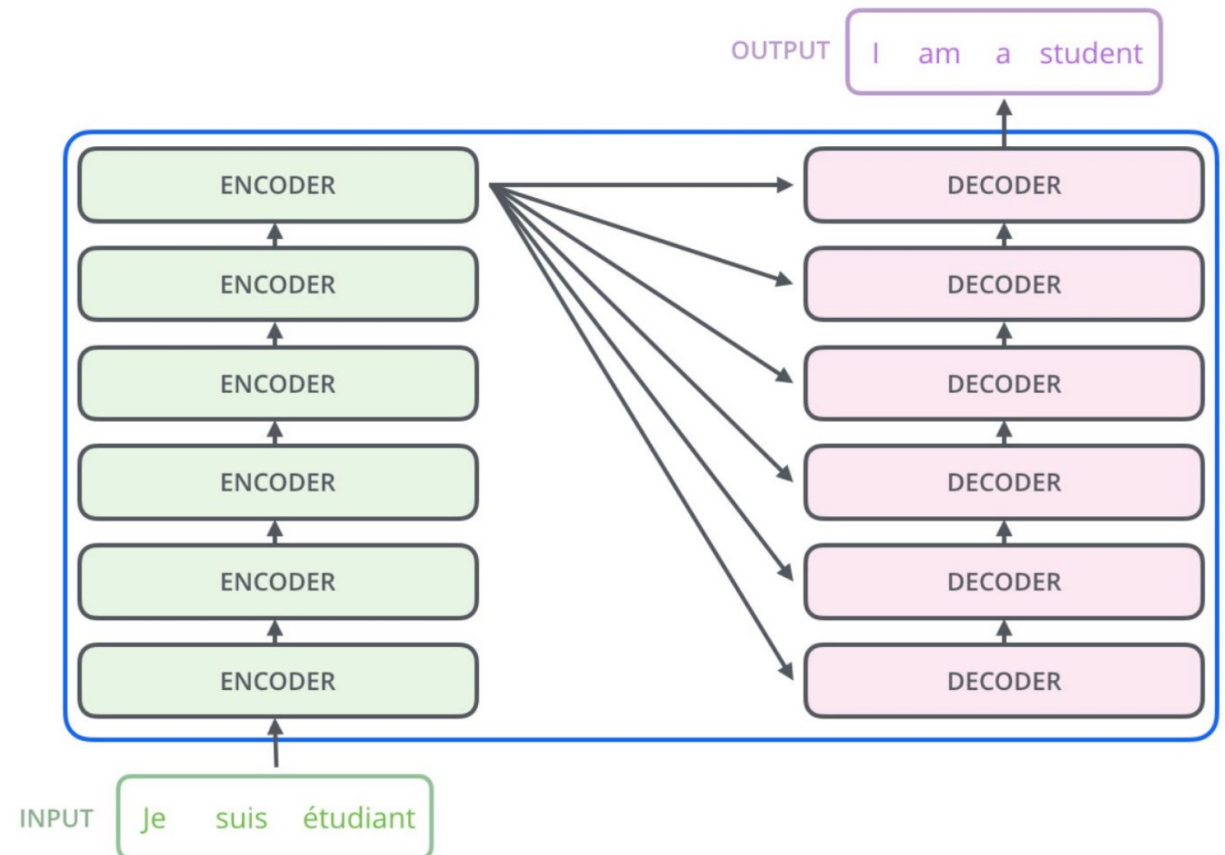
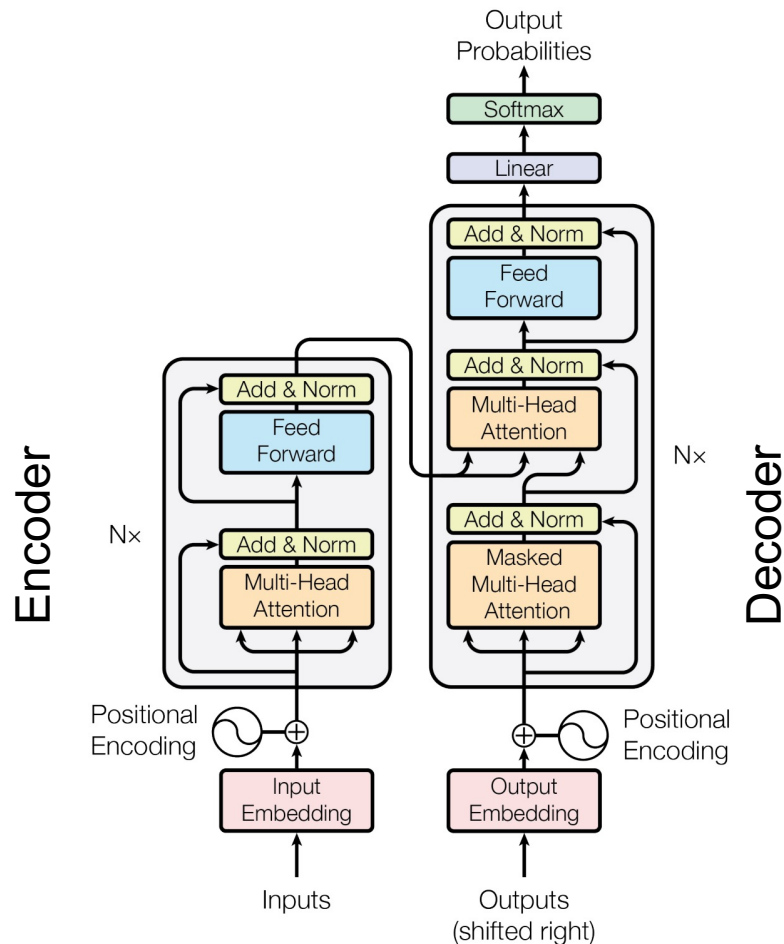
Parallelizing Convolutional Neural Networks

- Data parallelism for convolutional layers
- Tensor model parallelism for fully-connected layers

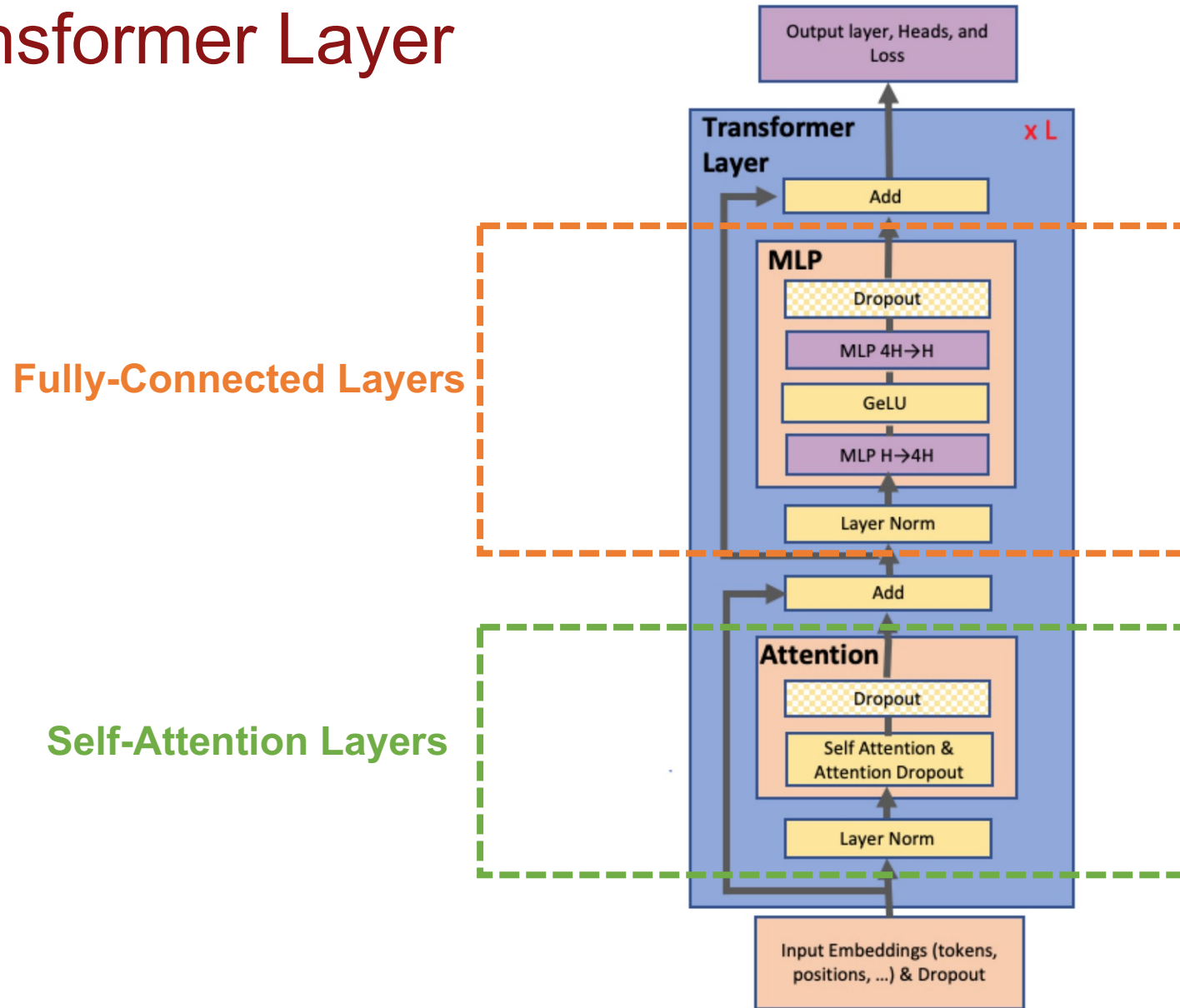


Example: Parallelizing Transformers

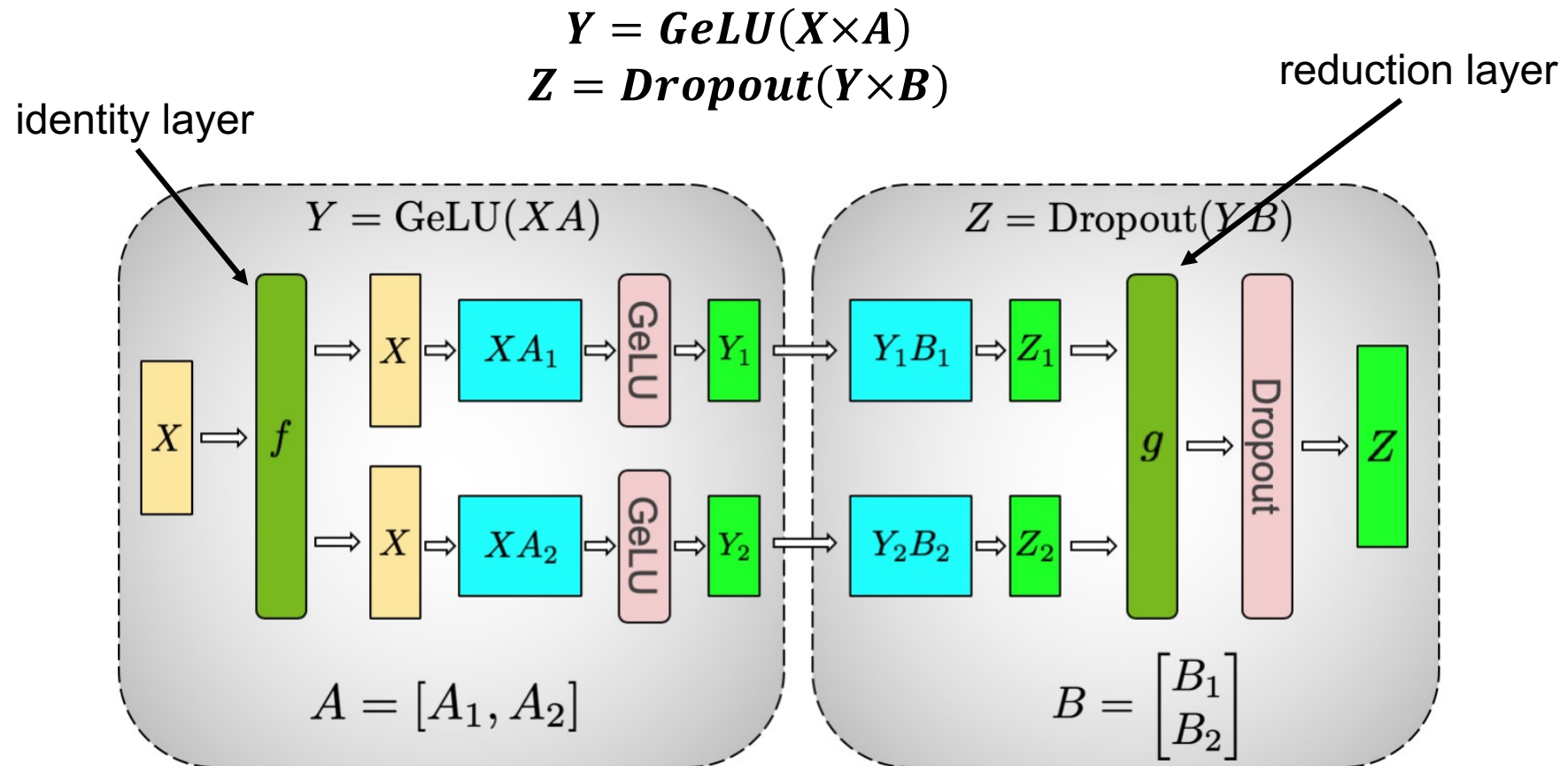
- Transformer: attention mechanism for language understanding



A Single Transformer Layer



Parallelizing Fully-Connected Layers in Transformers

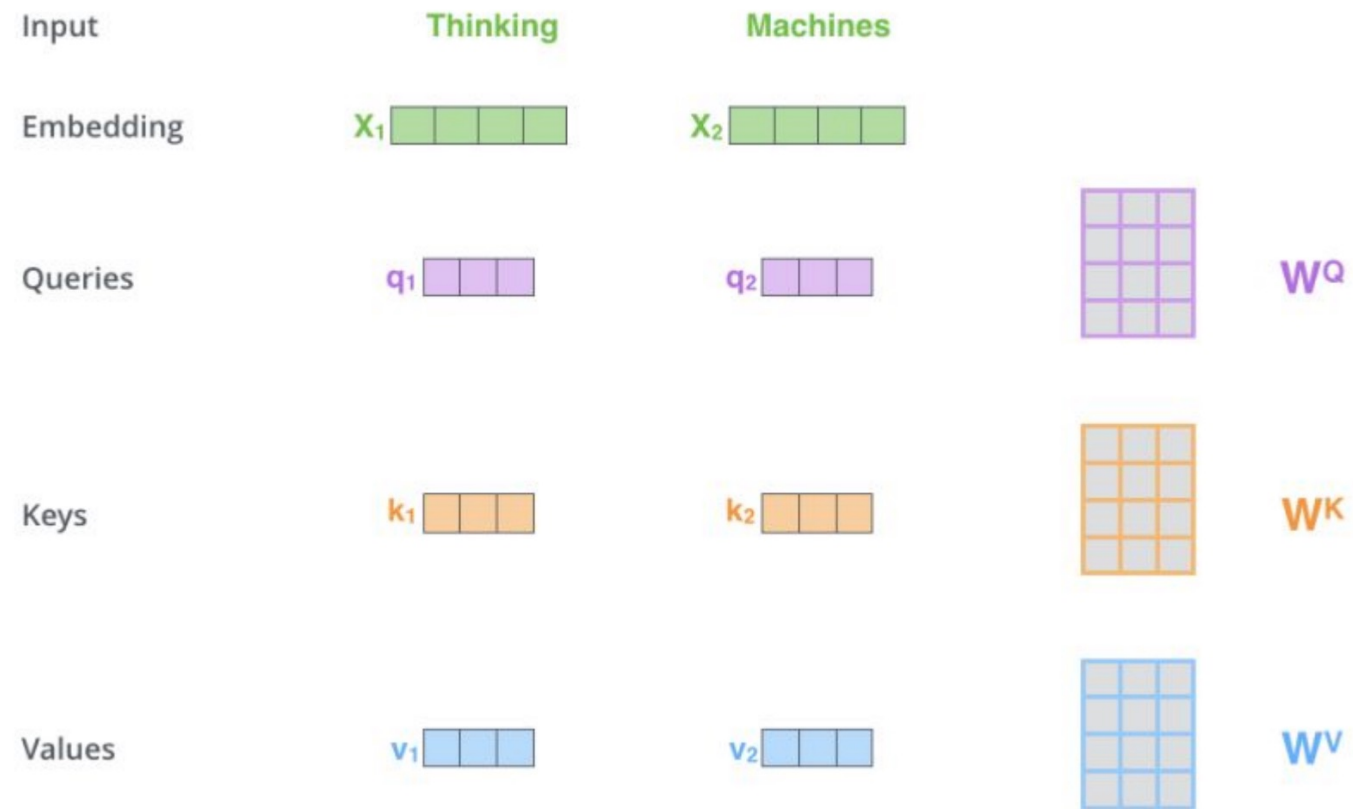


Tensor model parallelism
(partition output)

Tensor model parallelism
(reduce output)

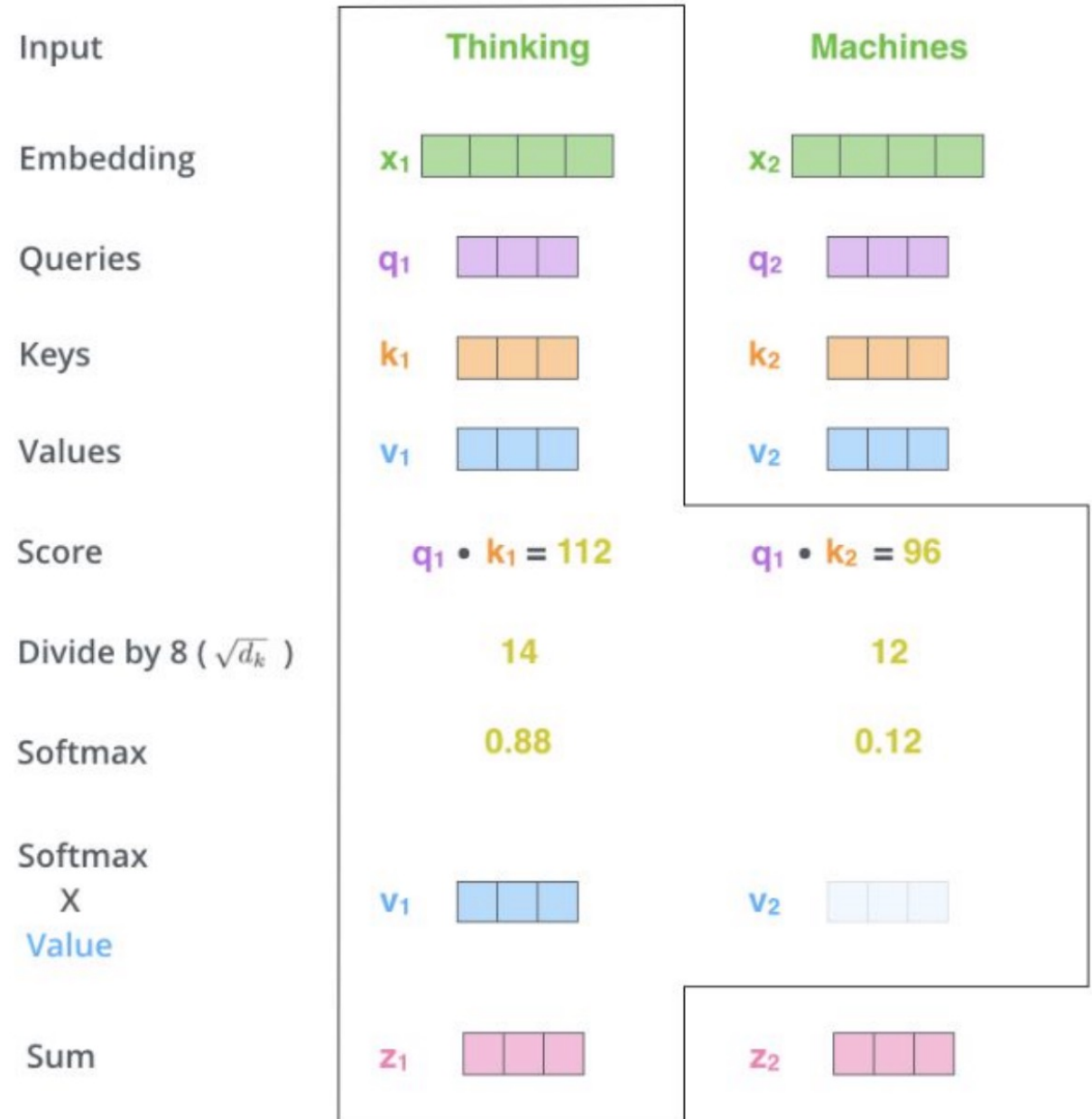
Self-Attention

- Mapping a query and a set of key-value pairs to an output

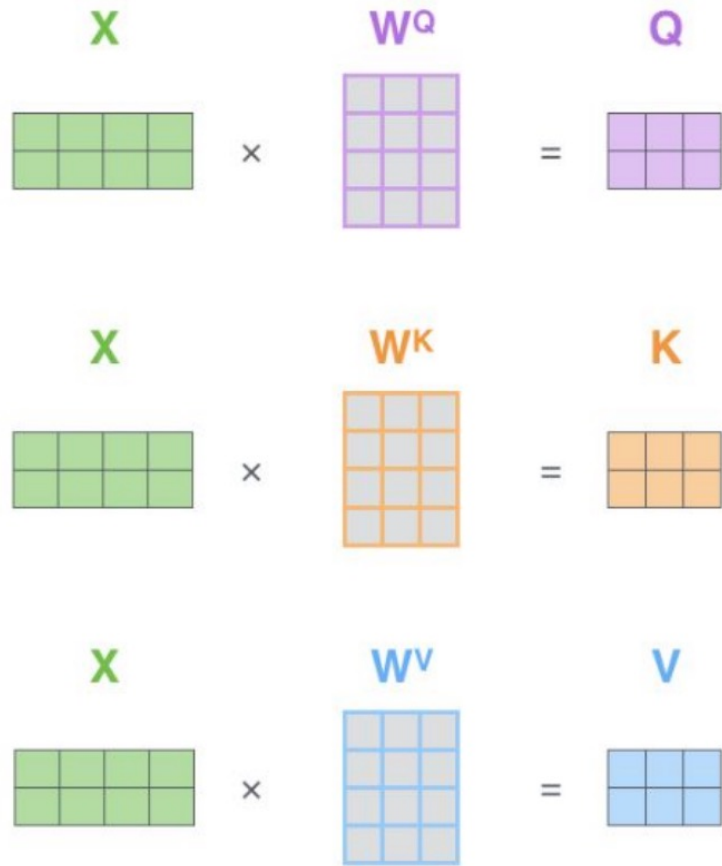


Self-Attention

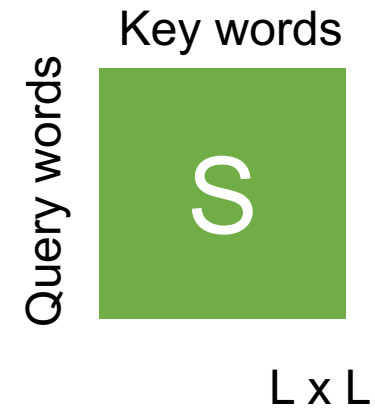
- Mapping a query and a set of key-value pairs to an output



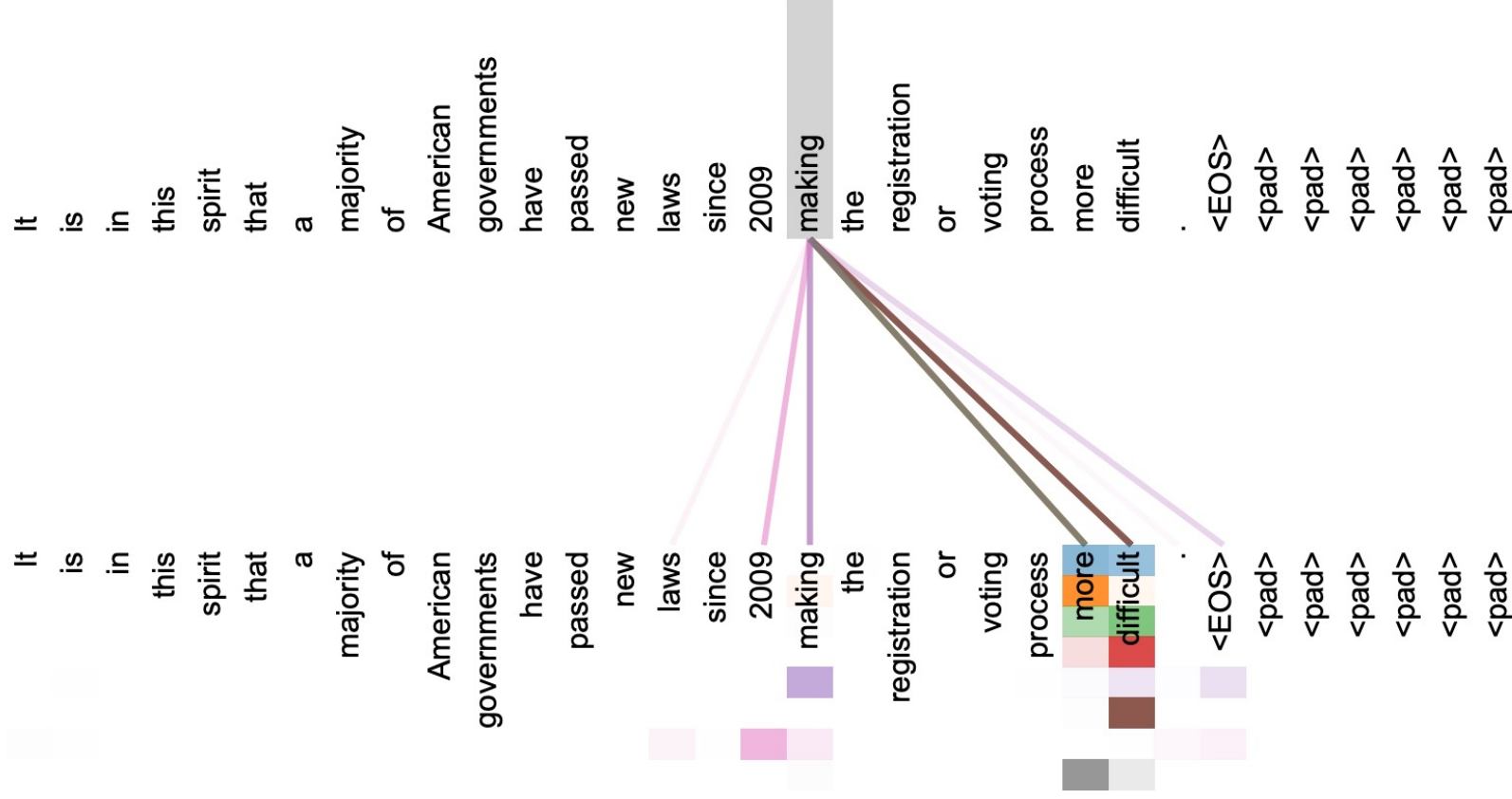
Self-Attention



$$A(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

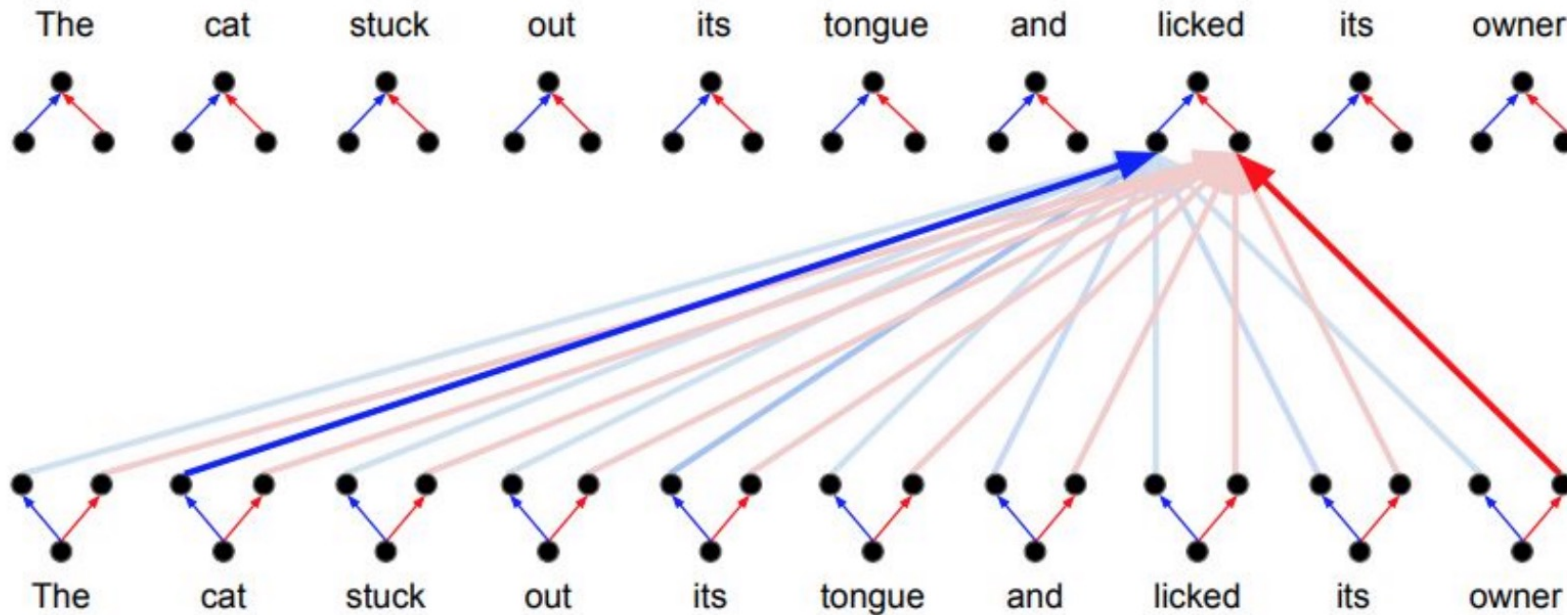


Self-Attention

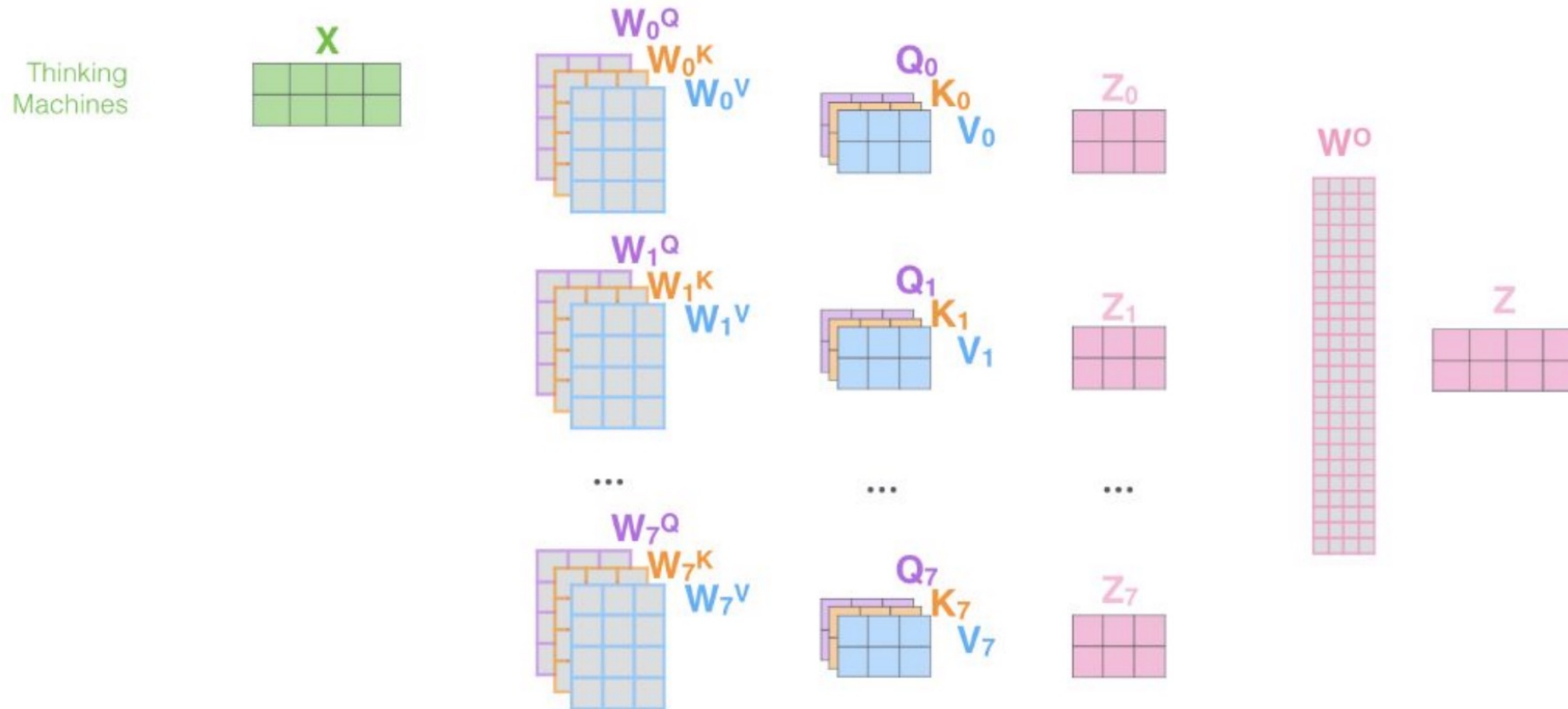


Multi-Head Self-Attention

- Parallelize attention layers with different linear transformations on input and output
- **Benefits: more parallelism, reduced computation cost**



Multi-Head Self-Attention



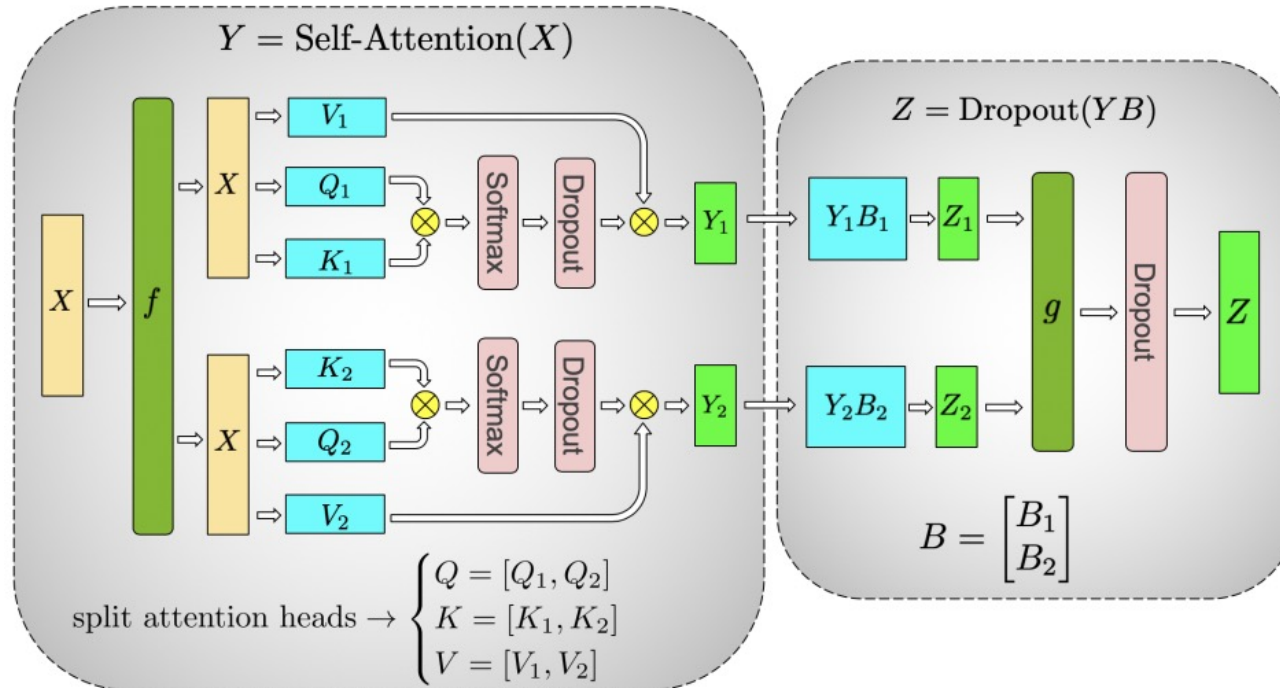
$$Z_i = A(Q_i, K_i, V_i) = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d}}\right) V_i$$

$$Z = \text{MultiHead}(Q, K, V) = \text{Concat}(Z_0, \dots, Z_7) W^O$$

Parallelizing Self-Attention Layers in Transformers

$$Y_i = A(Q_i, K_i, V_i) = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d}}\right) V_i$$

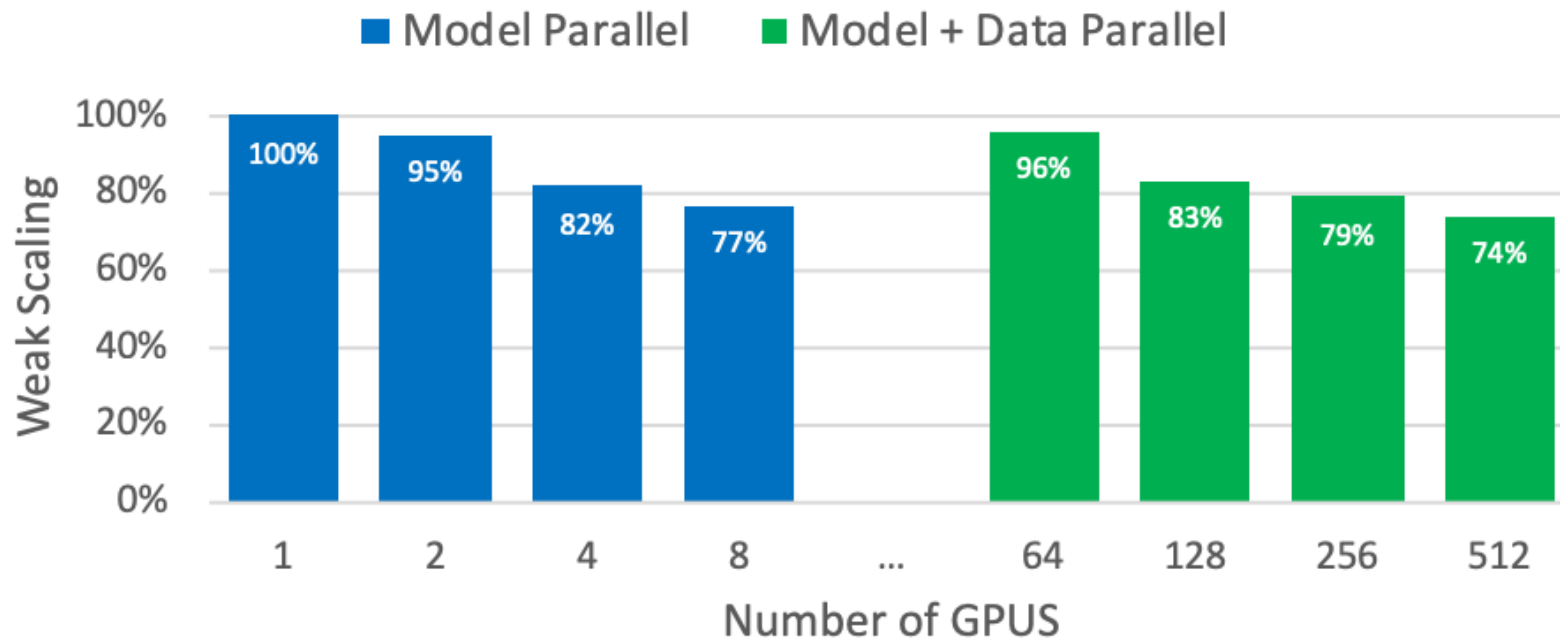
$$Z = \text{MultiHead}(Q, K, V) = \text{Concat}(Y_0, \dots, Y_h) W^o$$



Parallelizing across attention heads

Tensor model parallelism (reduce output)

Parallelizing Transformers



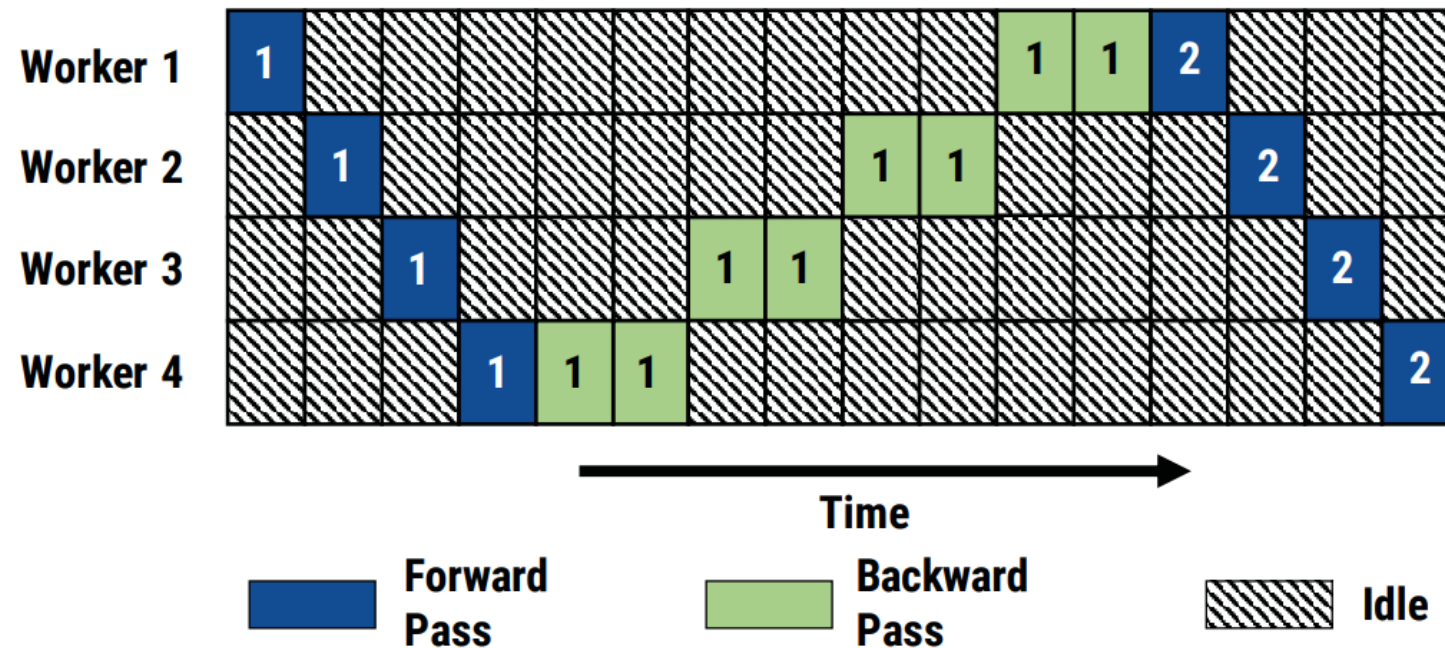
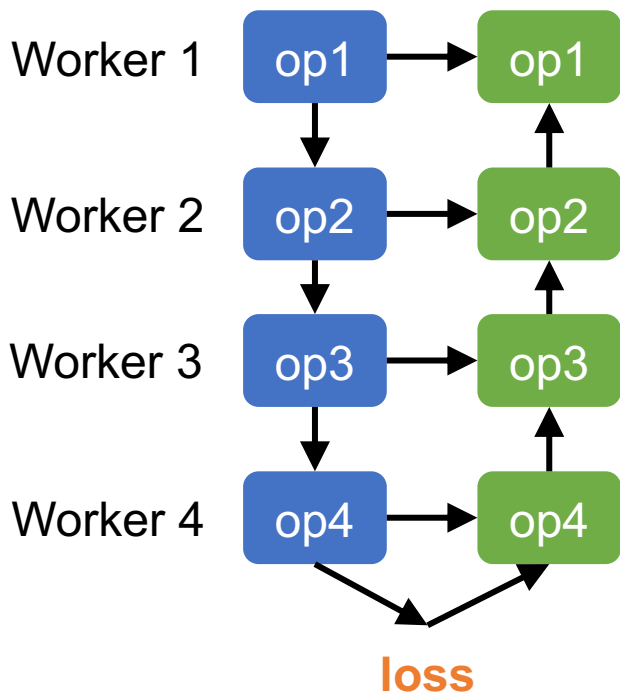
Scale to 512 GPUs by combining data and model parallelism

How to parallelize DNN Training?

- Data parallelism
- Model parallelism
- Tensor model parallelism
- **Pipeline model parallelism**

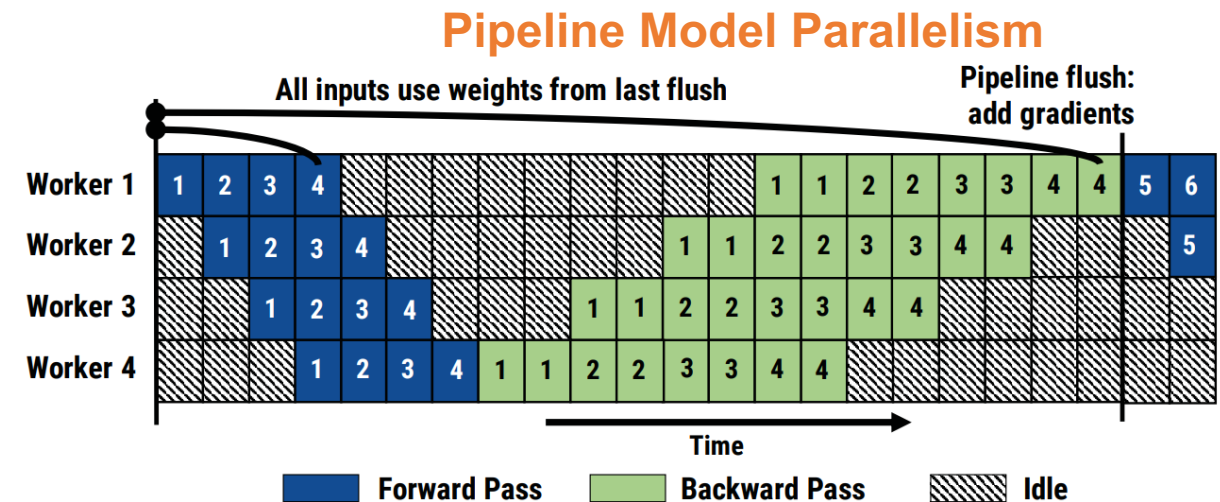
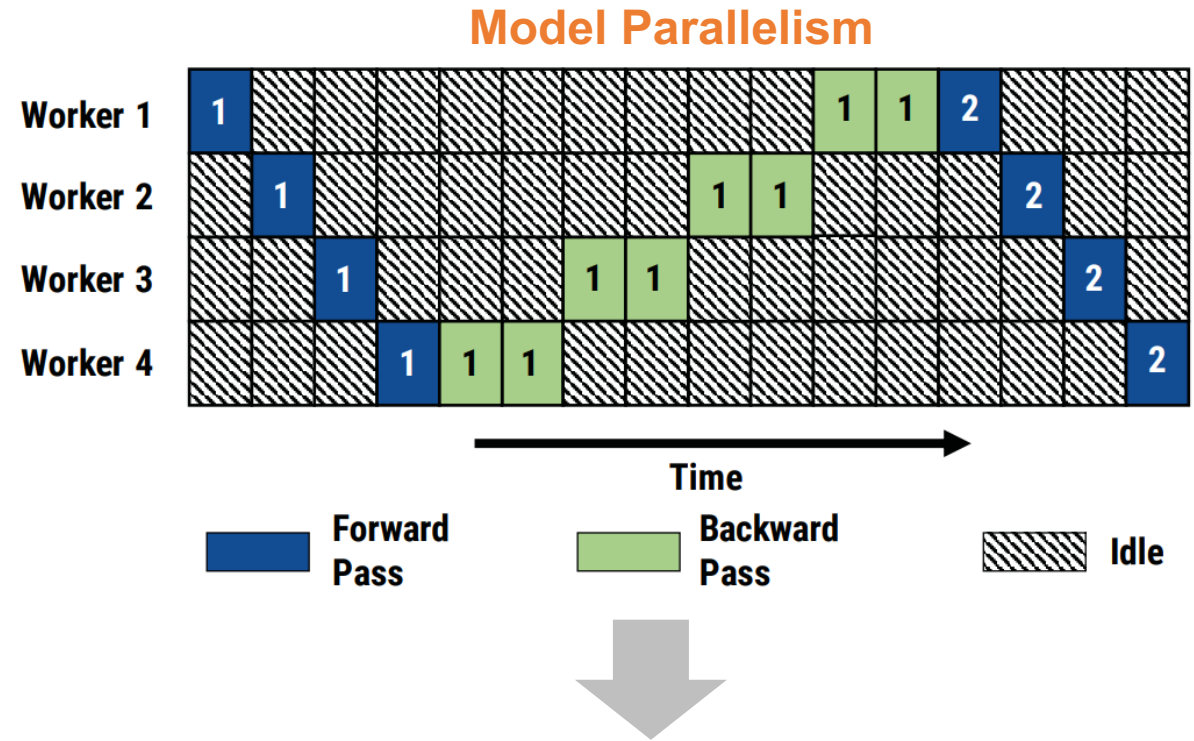
An Issue with Model Parallelism

- Under-utilization of compute resources
- Low overall throughput due to resource utilization



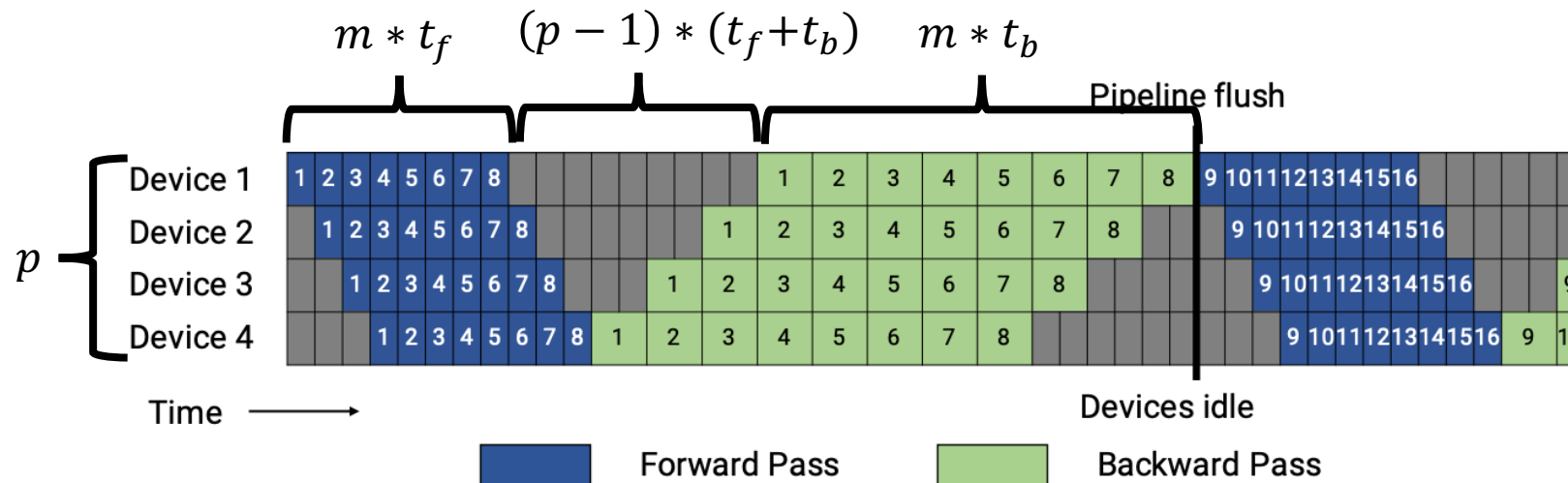
Pipeline Model Parallelism

- **Mini-batch**: the number of samples processed in each iteration
- Divide a mini-batch into multiple **micro-batches**
- Pipeline the forward and backward computations across micro-batches



Pipeline Model Parallelism: Device Utilization

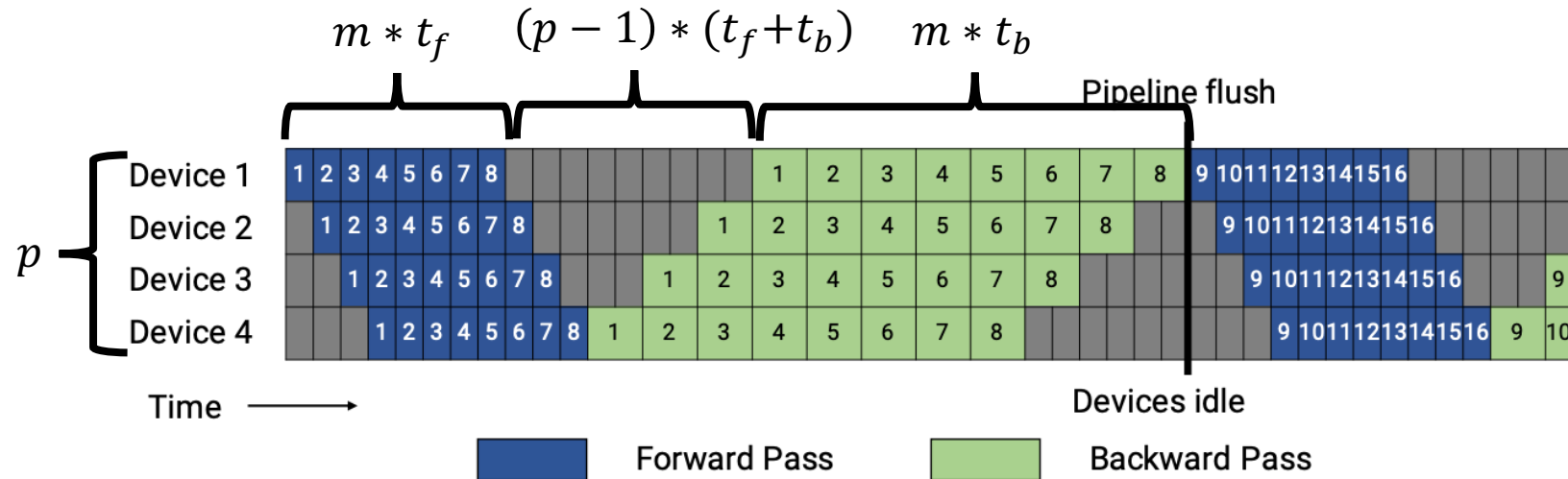
- m : micro-batches in a mini-batch
- p : number of pipeline stages
- All stages take t_f/ t_b to process a forward (backward) micro-batch



$$\mathbf{BubbleFraction} = \frac{(p - 1) * (t_f + t_b)}{m * t_f + m * t_b} = \frac{p - 1}{m}$$

Improving Pipeline Parallelism Efficiency

- m : number of micro-batches in a mini-batch
 - Increase mini-batch size or reduce micro-batch size
 - Caveat: large mini-batch sizes can lead to accuracy loss; small micro-batch sizes reduce GPU utilization
- p : number of pipeline stages
 - Decrease pipeline depth
 - Caveat: increase stage size

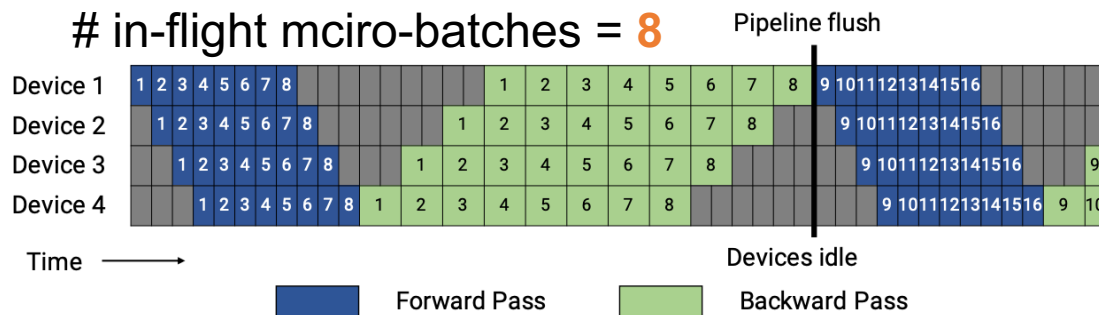


$$BubbleFraction = \frac{(p - 1) * (t_f + t_b)}{m * t_f + m * t_b} = \frac{p - 1}{m}$$

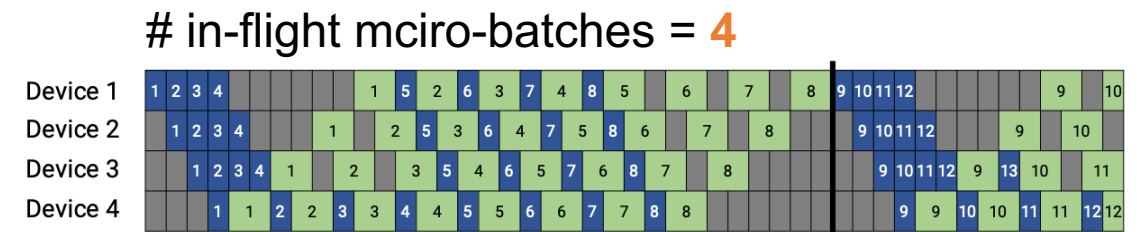
Pipeline Parallelism with 1F1B Schedule

- One-Forward-One-Backward in the steady state
- Limit the number of in-flight micro-batches to the pipeline depth
- Reduce memory footprint of pipeline parallelism
- Doesn't reduce pipeline bubble

Can we reduce pipeline bubble?



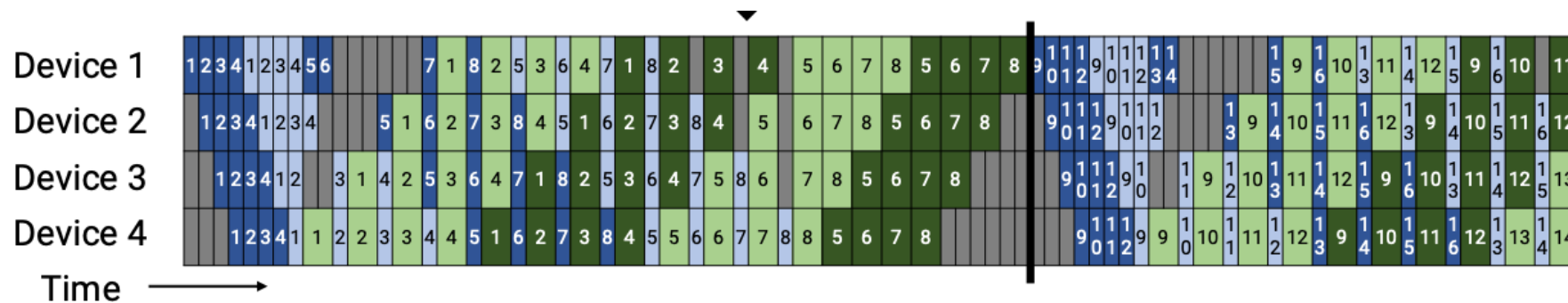
Pipeline parallelism with GPipe's schedule



Pipeline parallelism with 1F1B schedule

Pipeline Parallelism with Interleaved 1F1B Schedule

- Further divide each stages into v sub-stages
- The forward (backward) time of each sub-stage is $\frac{t_f}{v}$ ($\frac{t_b}{v}$)

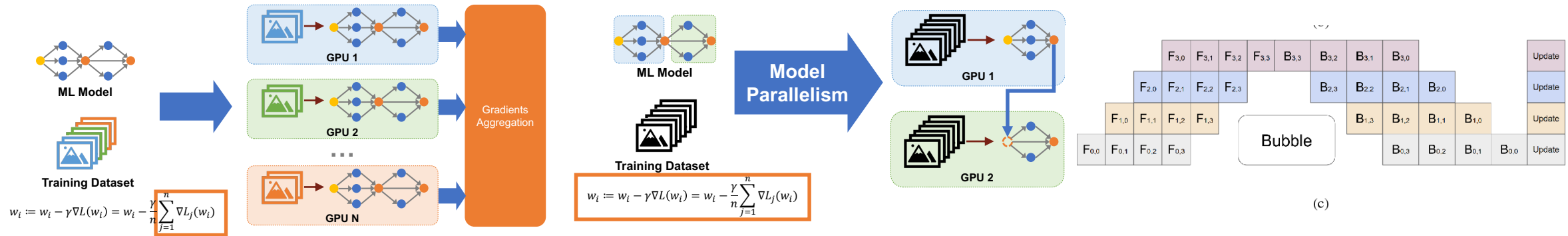


Each device is assigned two chunks. Dark colors show the first chunk and light colors show the second chunk.

$$BubbleFraction = \frac{(p - 1) * \frac{(t_f + t_b)}{v}}{m * t_f + m * t_b} = \frac{1}{v} * \frac{p - 1}{m}$$

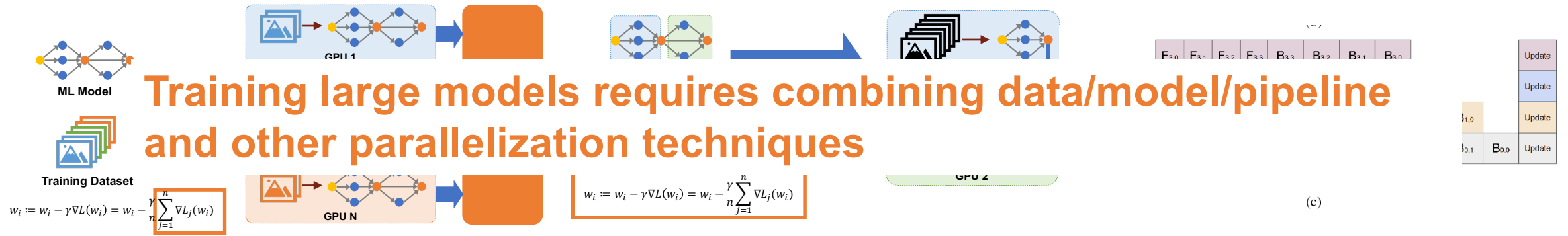
Reduce bubble time at the cost increased communication

Summary: Comparing Data/Model/Pipeline Parallelism



	Data Parallelism	Model Parallelism	Pipeline Parallelism
Pros	<ul style="list-style-type: none"> ✓ Massively parallelizable ✓ Require no communication during forward/backward 	<ul style="list-style-type: none"> ✓ Support training large models ✓ Efficient for models with large numbers of parameters 	<ul style="list-style-type: none"> ✓ Support large-batch training ✓ Efficient for deep models
Cons	<ul style="list-style-type: none"> ❖ Do not work for models that cannot fit on a GPU ❖ Do not scale for models with large numbers of parameters 	<ul style="list-style-type: none"> ❖ Limited parallelizability; cannot scale to large numbers of GPUs ❖ Need to transfer intermediate results in forward/backward 	<ul style="list-style-type: none"> ❖ Limited utilization: bubbles in forward/backward

Summary: Data/Model/Pipeline Parallelism



	Data Parallelism	Model Parallelism	Pipeline Parallelism
Pros	<ul style="list-style-type: none"> ✓ Massively parallelizable ✓ Require no communication during forward/backward 	<ul style="list-style-type: none"> ✓ Support training large models ✓ Efficient for models with large numbers of parameters 	<ul style="list-style-type: none"> ✓ Support large-batch training
Cons	<ul style="list-style-type: none"> ❖ Do not work for models that cannot fit on a GPU ❖ Do not scale for models with large numbers of parameters 	<ul style="list-style-type: none"> ❖ Limited parallelizability; cannot scale to large numbers of GPUs ❖ Need to transfer intermediate results in forward/backward 	<ul style="list-style-type: none"> ❖ Limited utilization: bubbles in forward/backward

Example: 3D parallelism in DeepSpeed

Pipeline Model Parallelism

