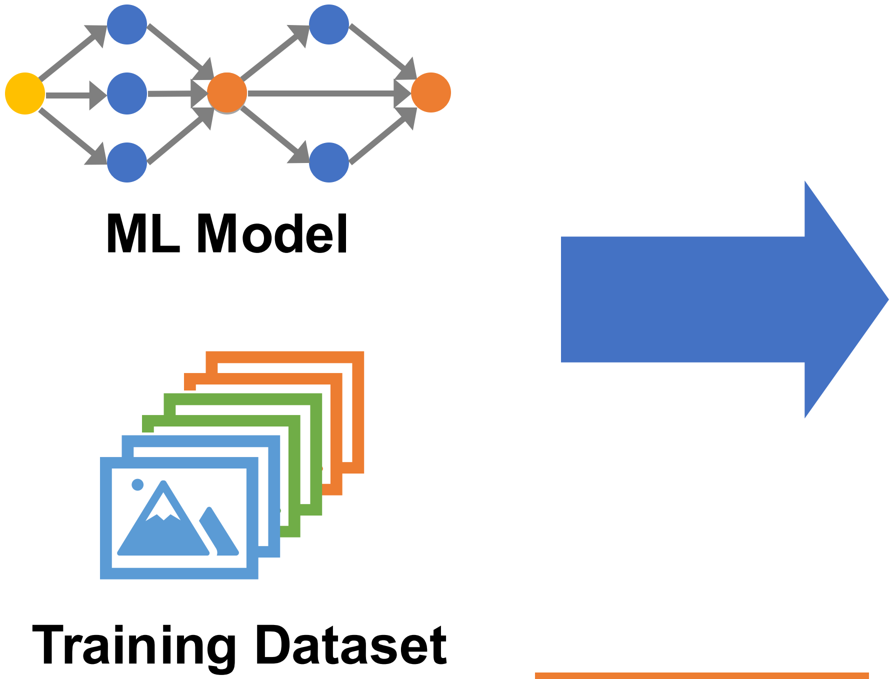
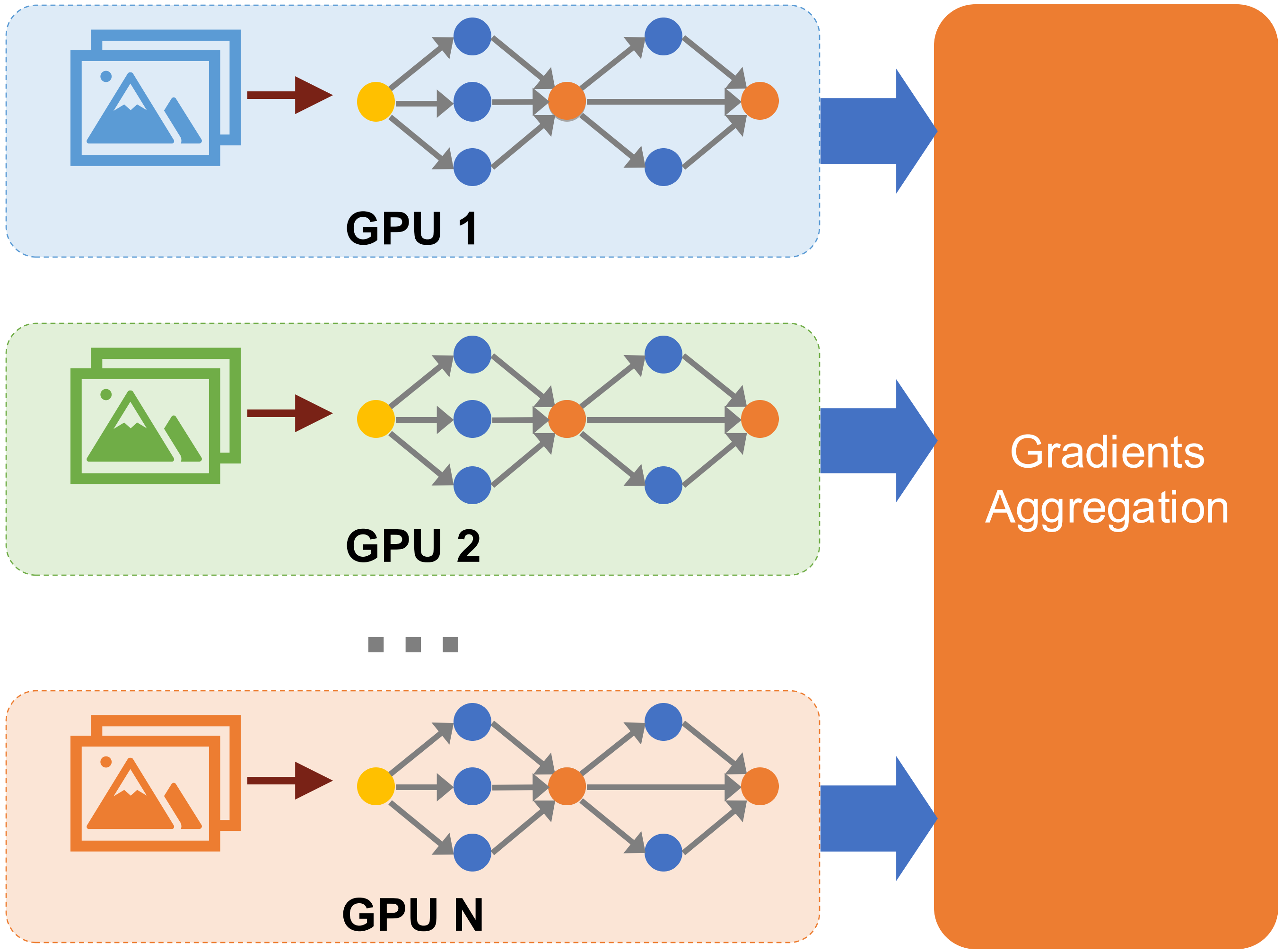
Lecture 25:
Parallel Deep Learning
(Model & Pipeline Parallelism)

Parallel Computer Architecture and Programming

CMU 15-418/15-618, Fall 2025

Recap: Data Parallelism





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

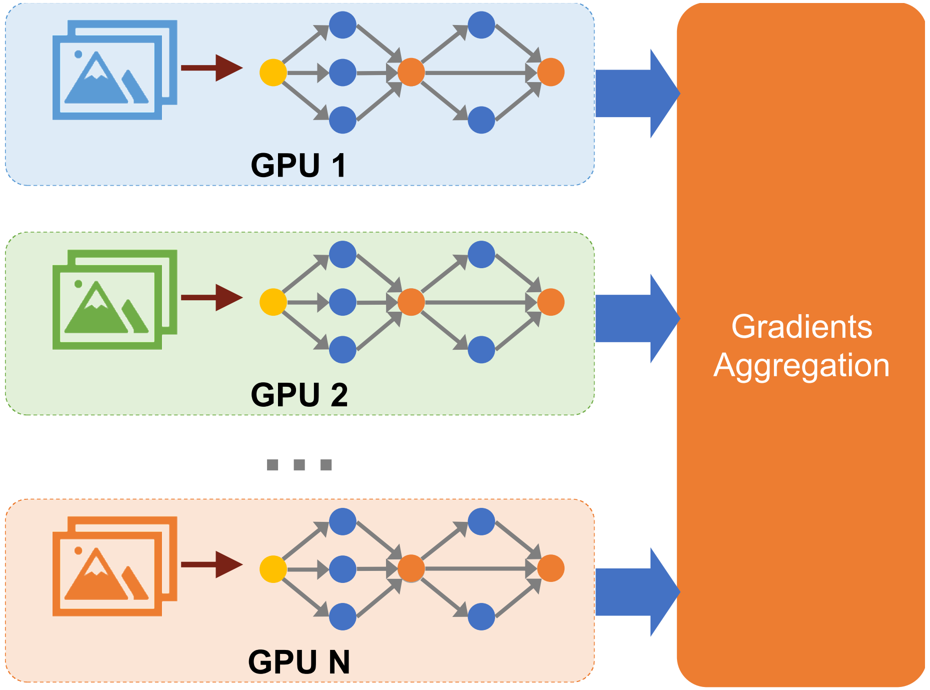
1. Partition training data into batches

2. Compute the gradients of each batch on a GPU

3. Aggregate gradients across GPUs

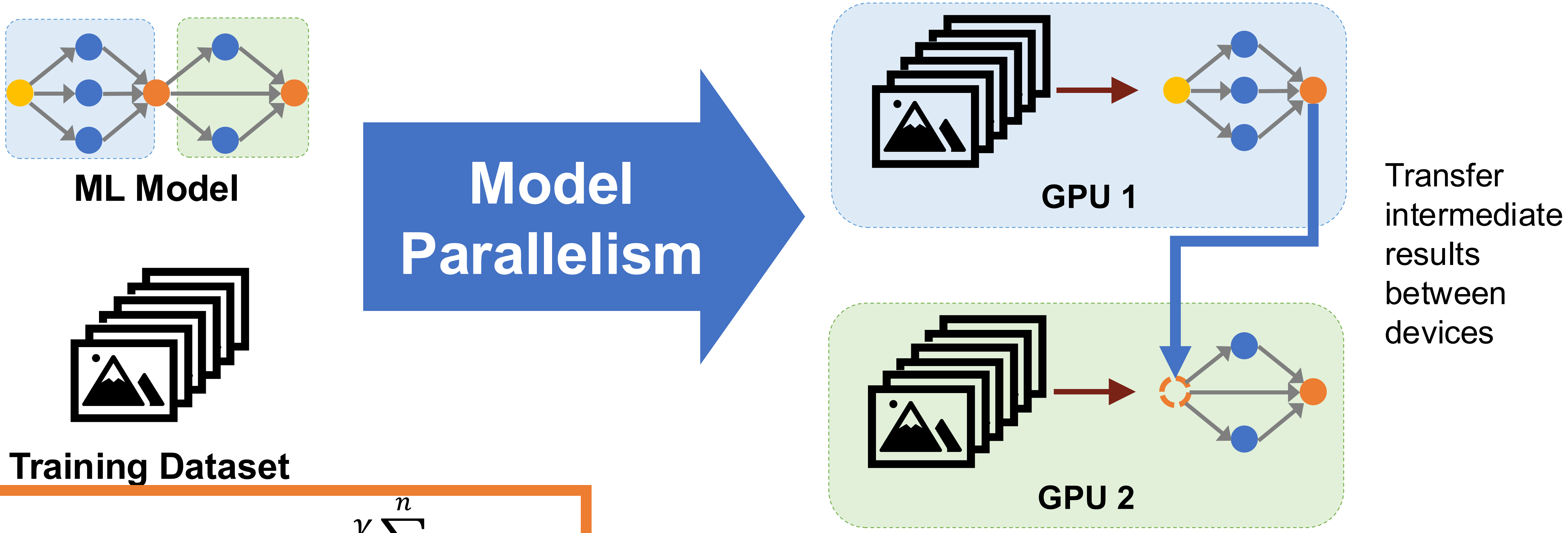
Recap: An Issue with Data Parallelism

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



Model Parallelism

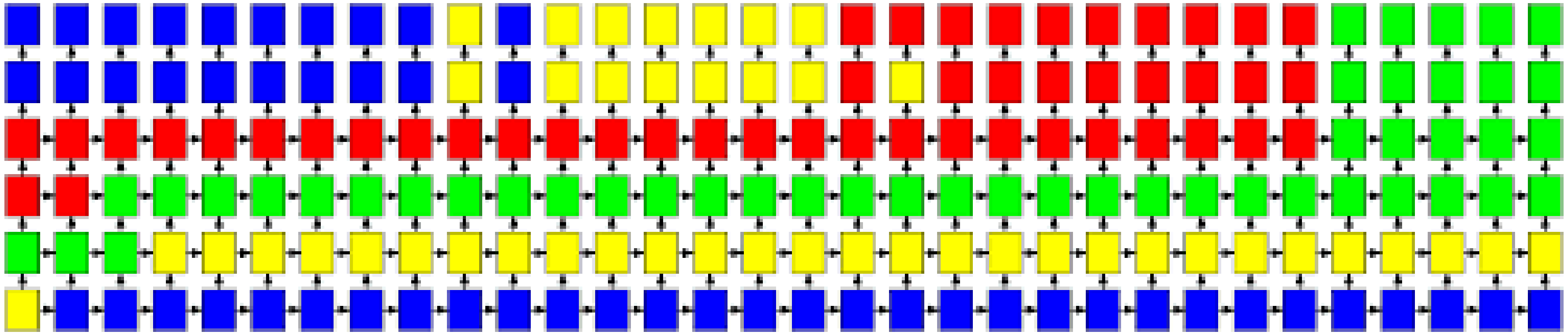
- Split a model into multiple subgraphs and assign them to different devices



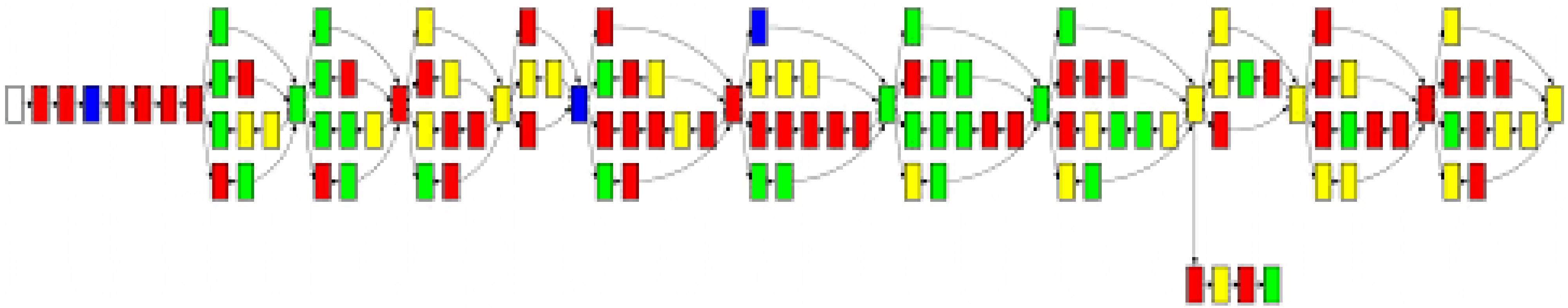
$$w_i := w_i - \gamma \nabla L(w_i) = w_i - \frac{\gamma}{n}\sum_{j=1}^{n} \nabla L_j(w_i)$$

No synchronization of gradients

Device Placement for Model Parallelism is Challenging

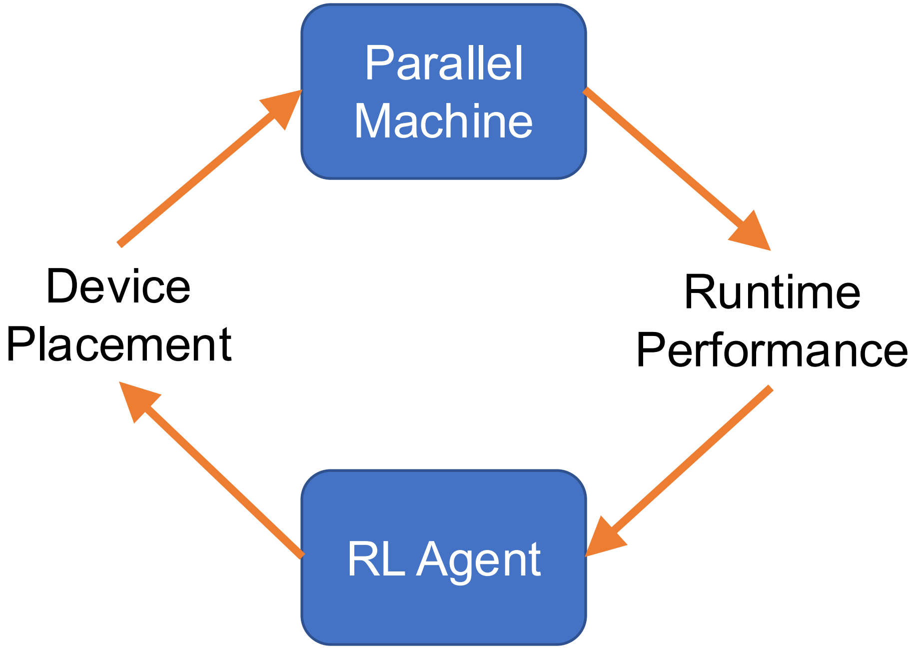


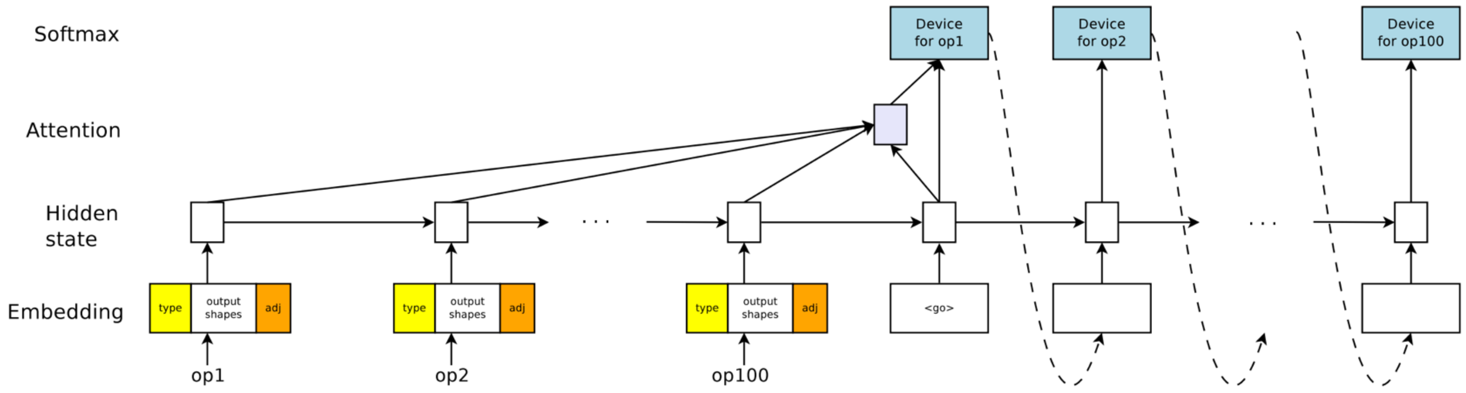
Model parallelism: training a recurrent neural network on 4 GPUs



Model parallelism: training a conventional neural network on 4 GPUs

Using ML to Optimize Device Placement for ML

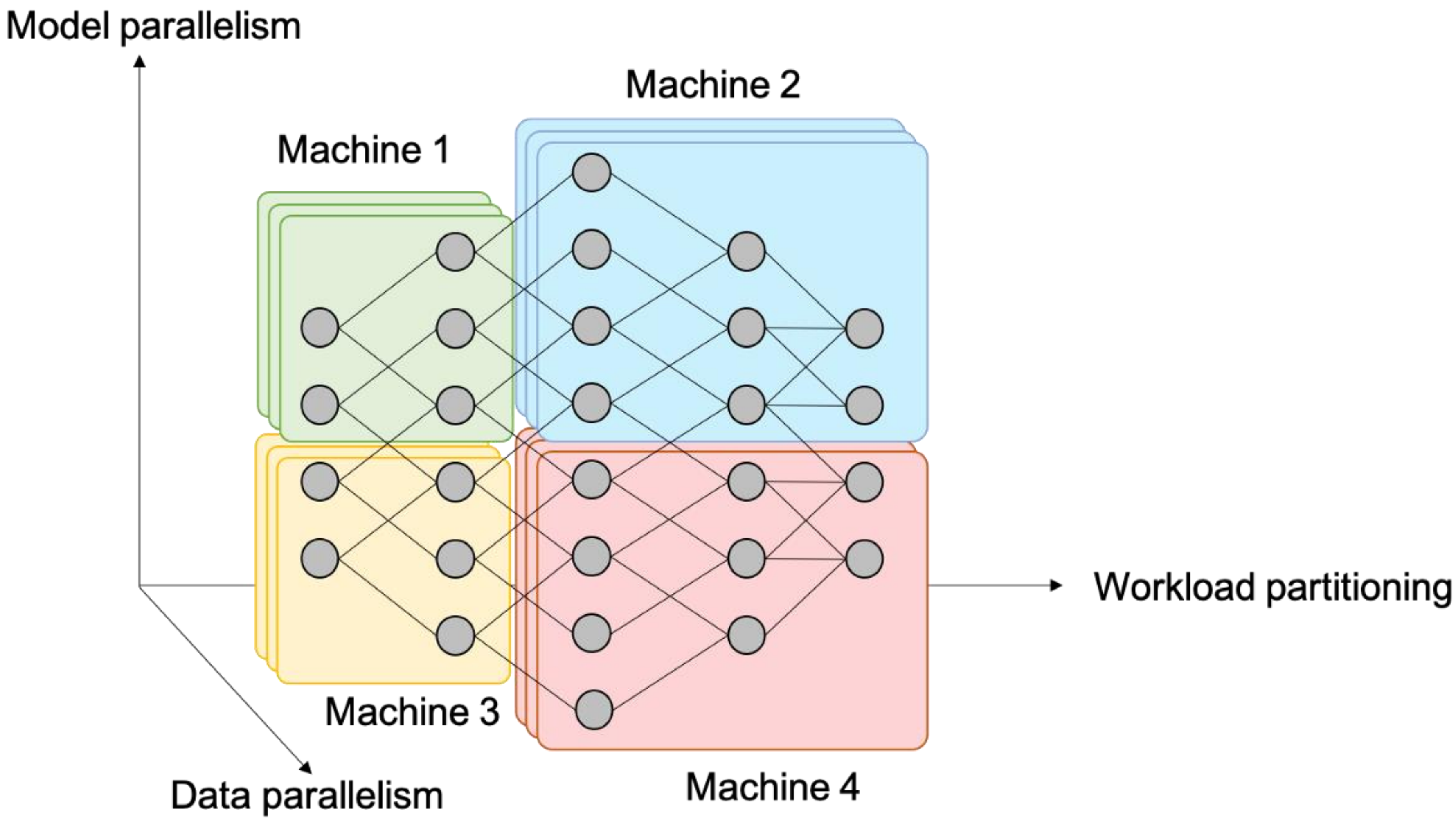




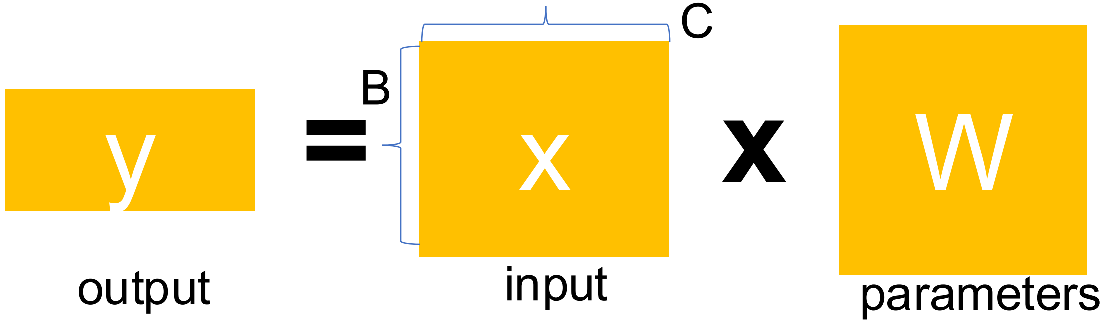
ColocRL's neural architecture

Device placement optimization with reinforcement learning. A Mirhoseini et al.

Combine Data and Model Parallelism

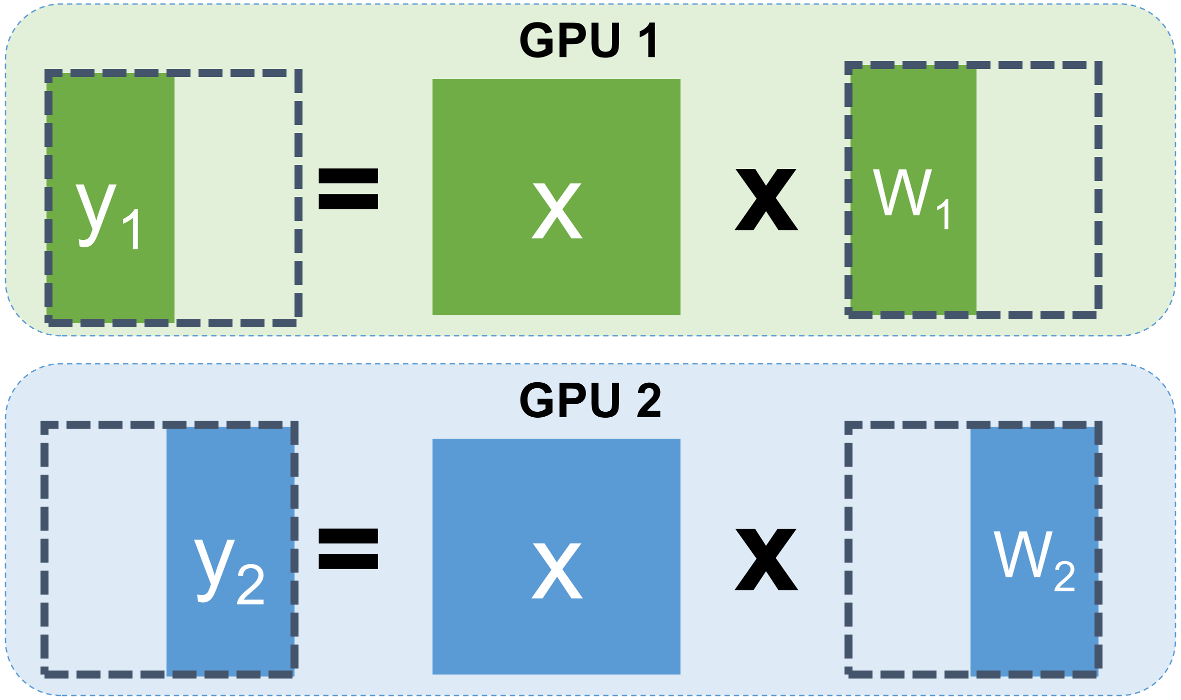


Tensor Model Parallelism

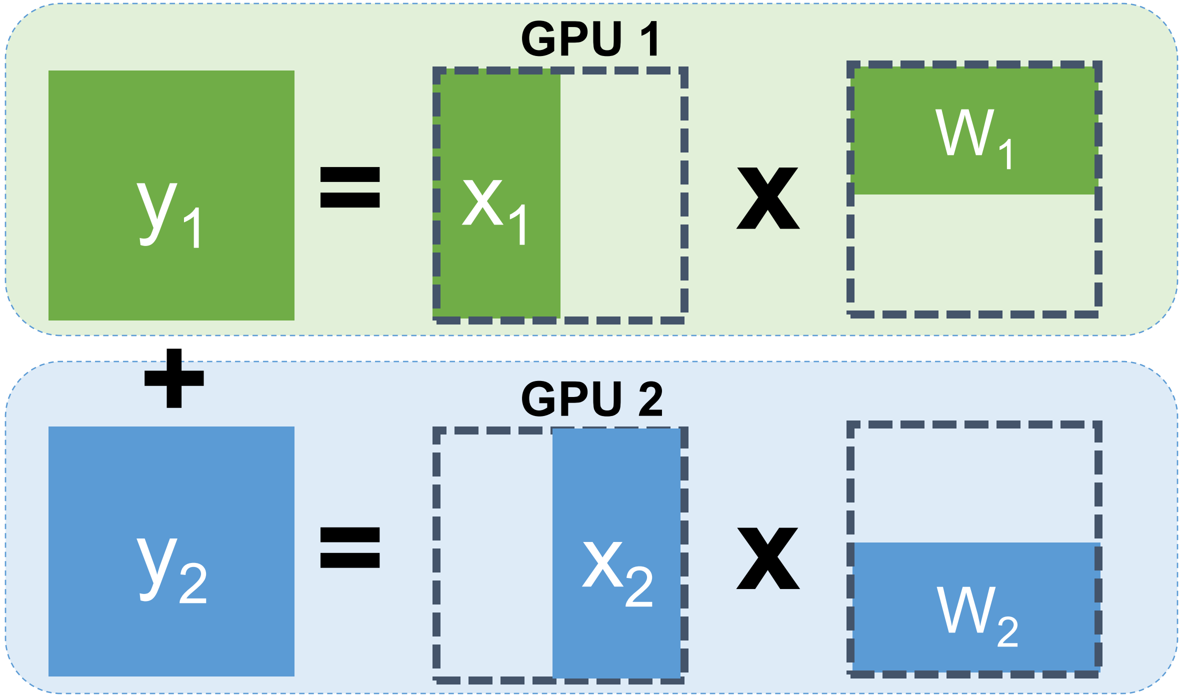


- Partition parameters/gradients ***within*** a layer/operator

Each GPU has the entire input



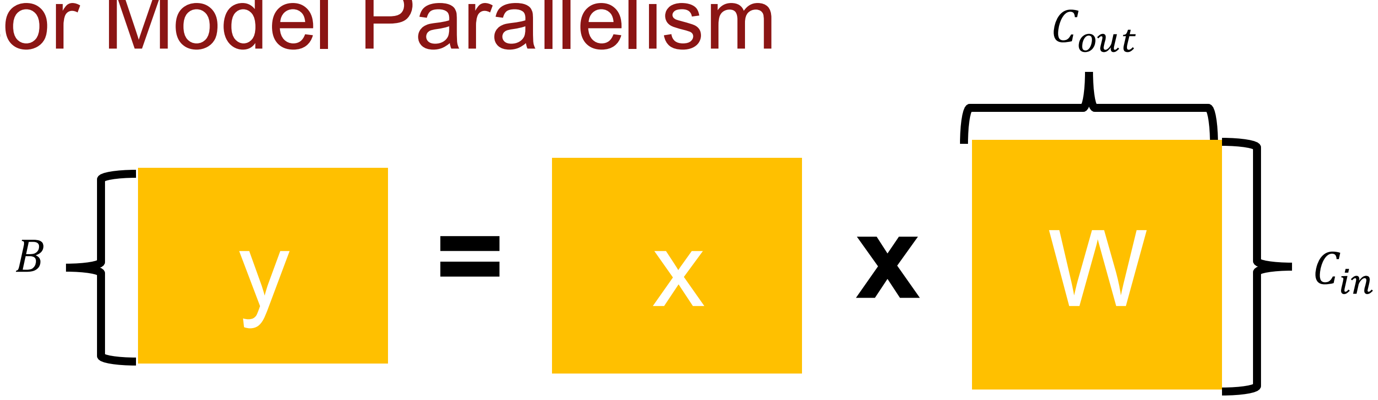
Tensor Model Parallelism (partition output)

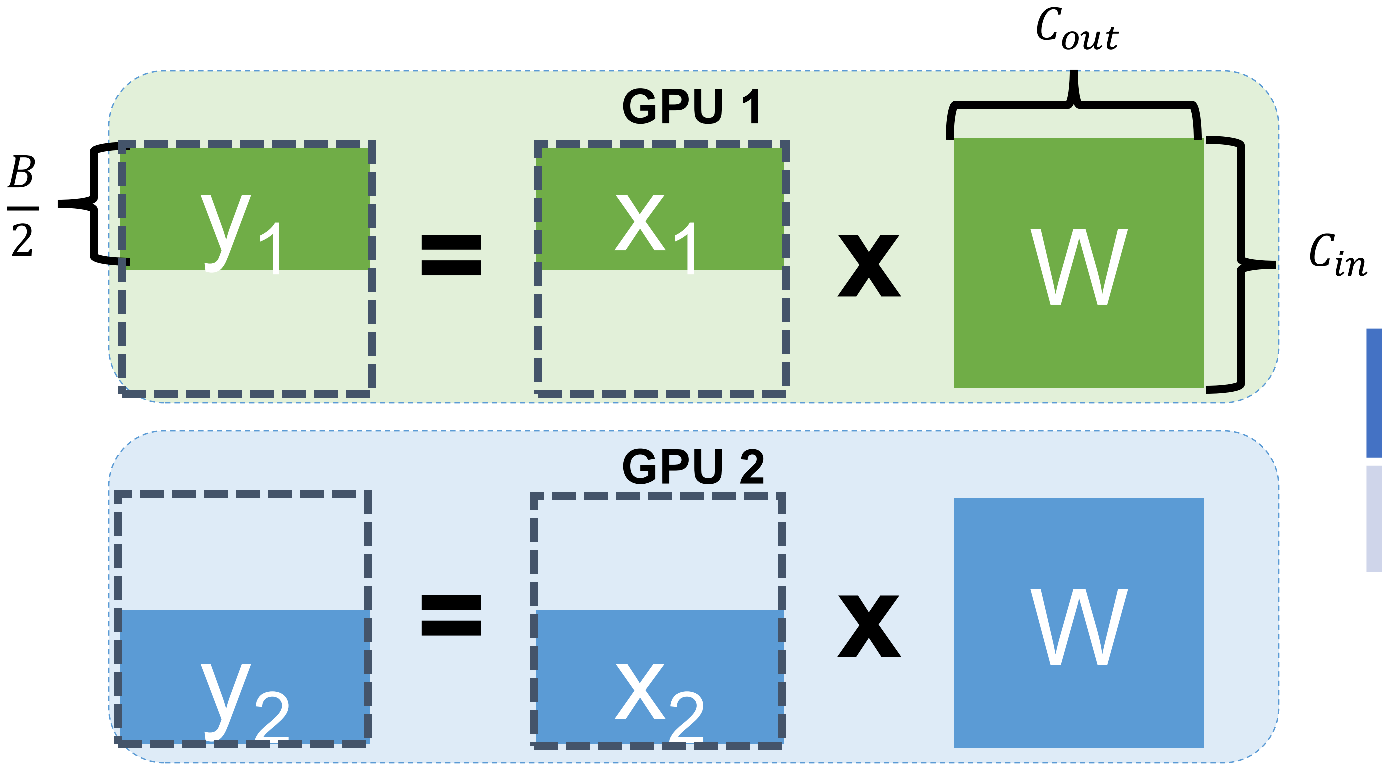


Tensor Model Parallelism (reduce output)

$y = y1 + y2$

Comparing Data and Tensor Model Parallelism



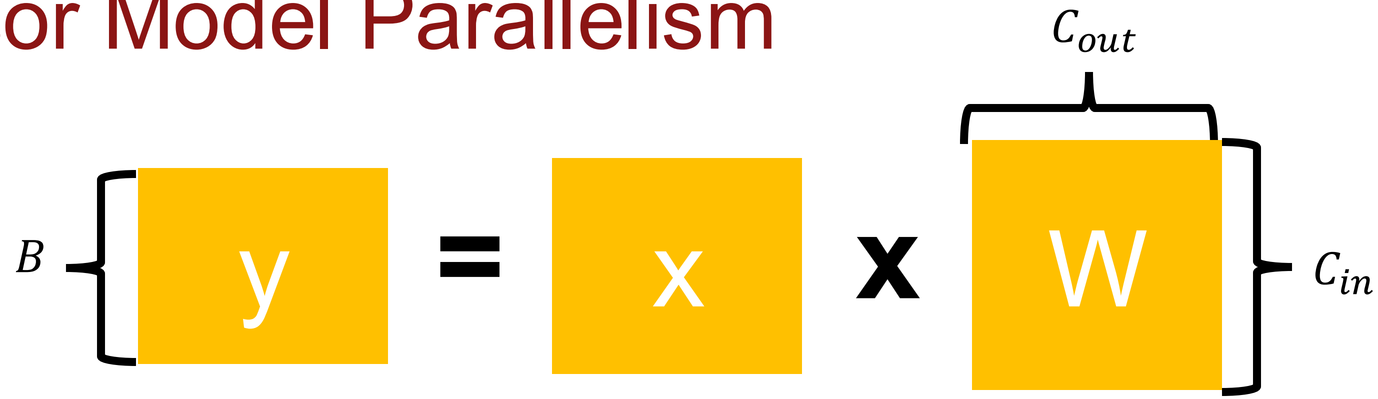


Forward Processing	Backward Propagation	Gradients Sync
0	0	$2 * C_{out} * C_{in}$

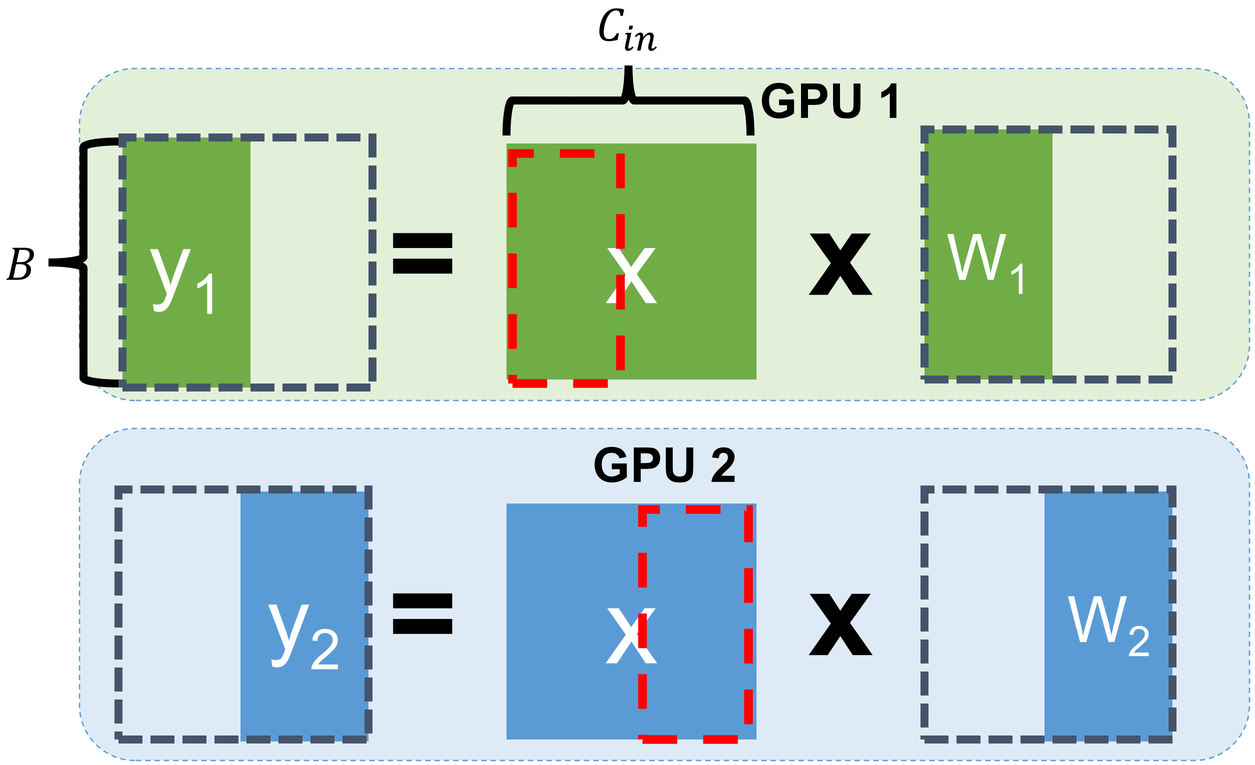
Communication Cost of Data Parallelism

 Data parallelism $y = Wx$

Comparing Data and Tensor Model Parallelism



F: Starting with partial Input



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

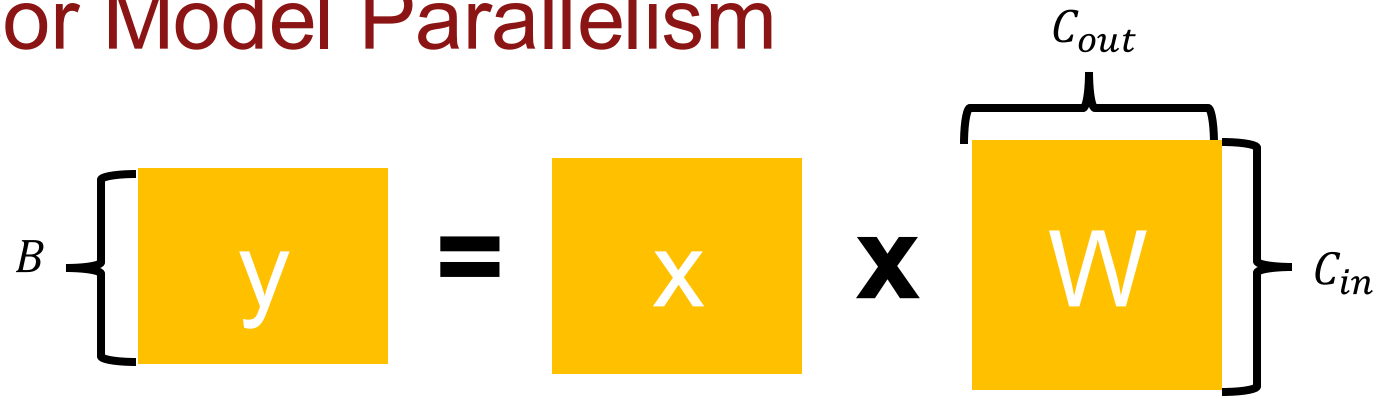
Communication Cost of Tensor Model Parallelism

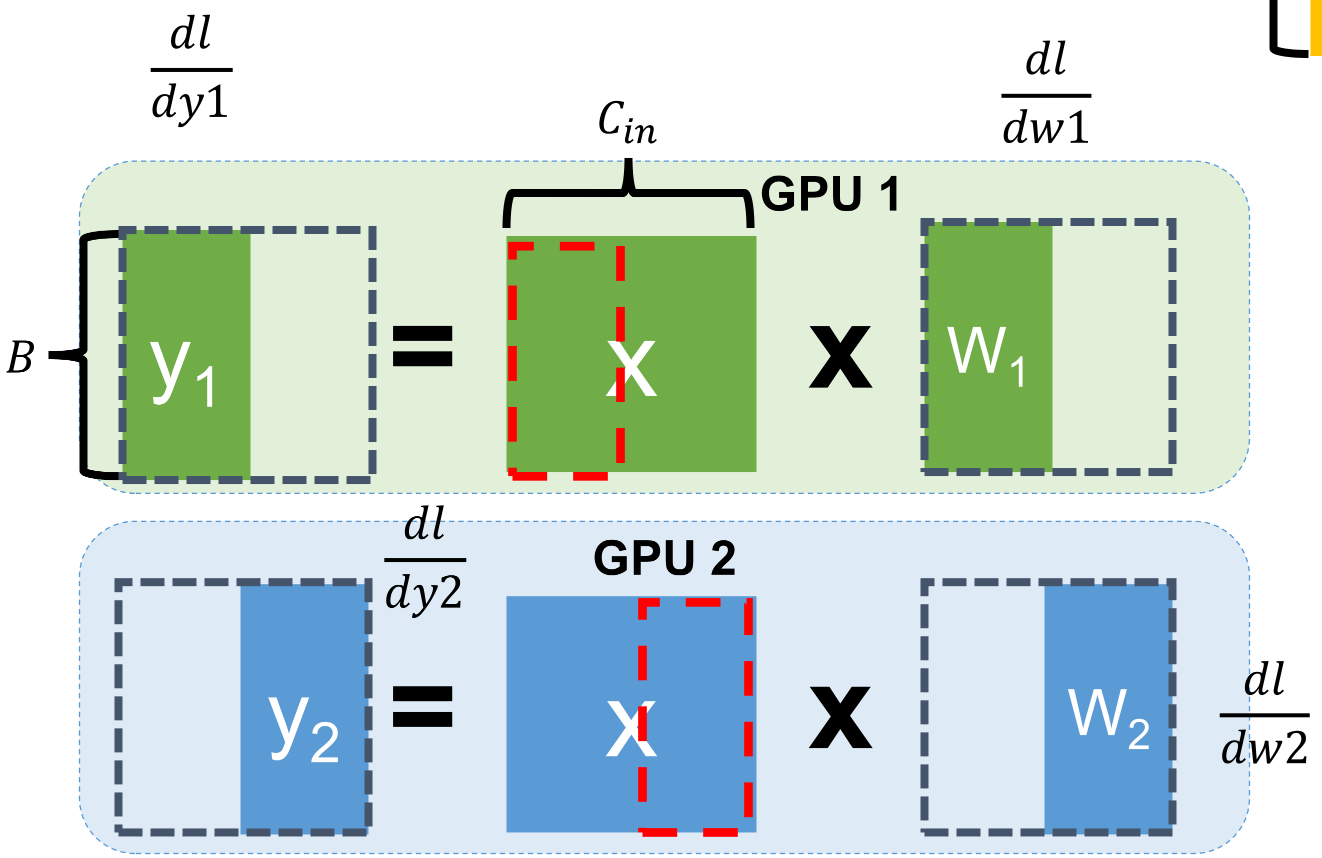
Tensor Model Parallelism (partition output)

*Assume each GPU has part of the input

Comparing Data and Tensor Model Parallelism

$$\frac{dl}{dw1} = \textcolor{red}{X}\frac{dl}{dly}$$





Tensor Model Parallelism (partition output)

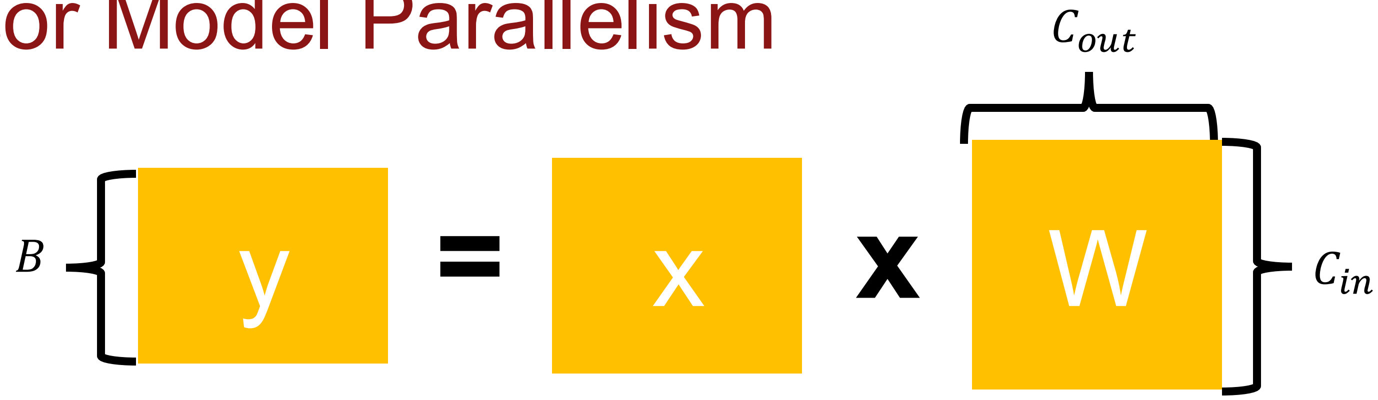
$$\frac{dl}{dx} = \frac{dl}{dlx1} + \frac{dl}{dlx2}$$

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

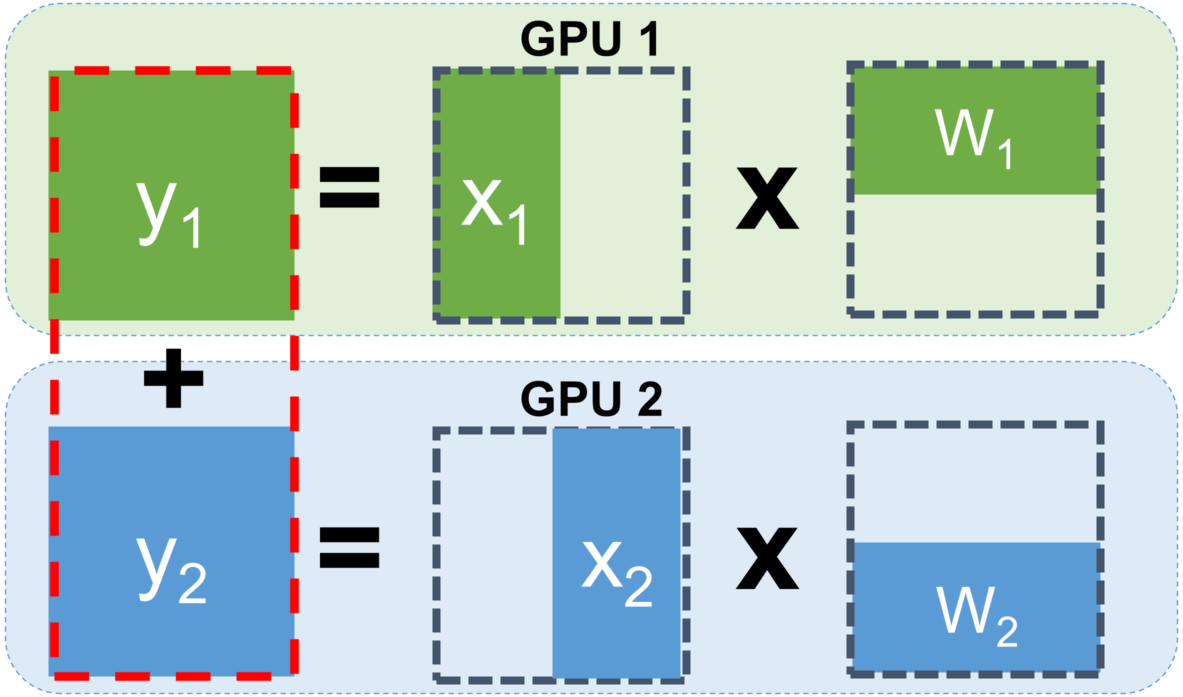
Communication Cost of Tensor Model Parallelism

*Assume each GPU has part of the input e.g., previous layer uses data parallelism

Comparing Data and Tensor Model Parallelism



Allreduce for y1 and y2



Tensor Model Parallelism (Reduce output)

$$y = y1 + y2$$

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: $C_{out} * C_{in}$ → Synchronize gradients
- Tensor model parallelism (partition output): $B * C_{in}$
- Tensor model parallelism (reduce output): $B * C_{out}$

→ transfer activations*

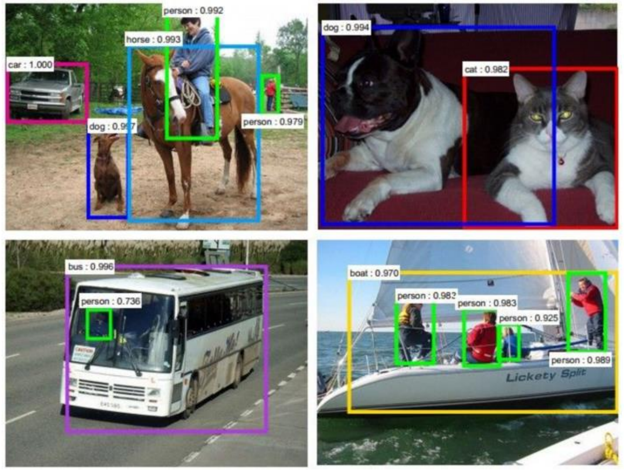
- **The best strategy depends on the model and underlying machine**

*intermediate results

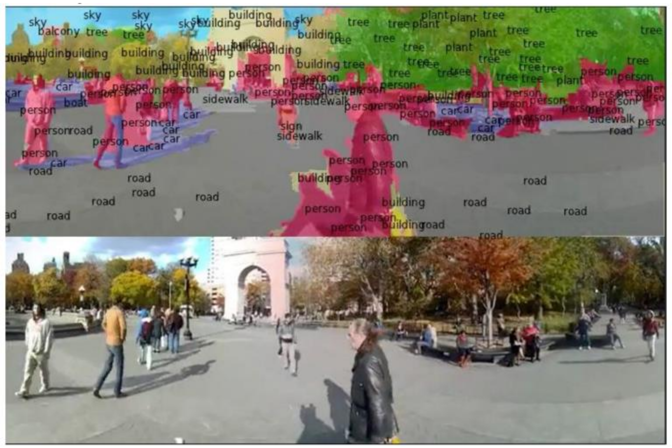
Example: Convolutional Neural Networks

Classification

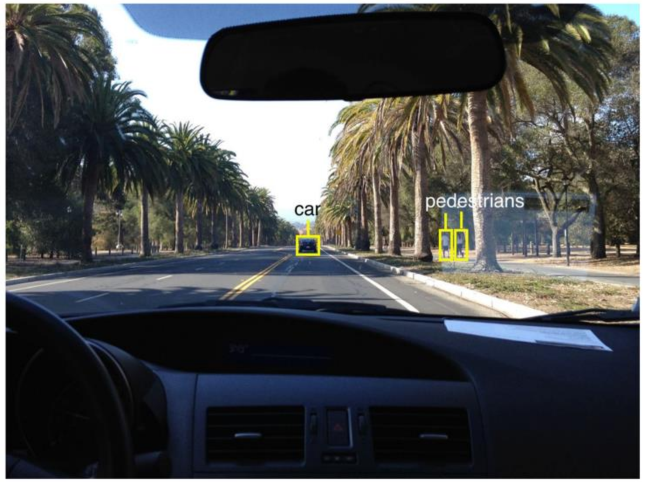
Retrieval



Detection



Segmentation

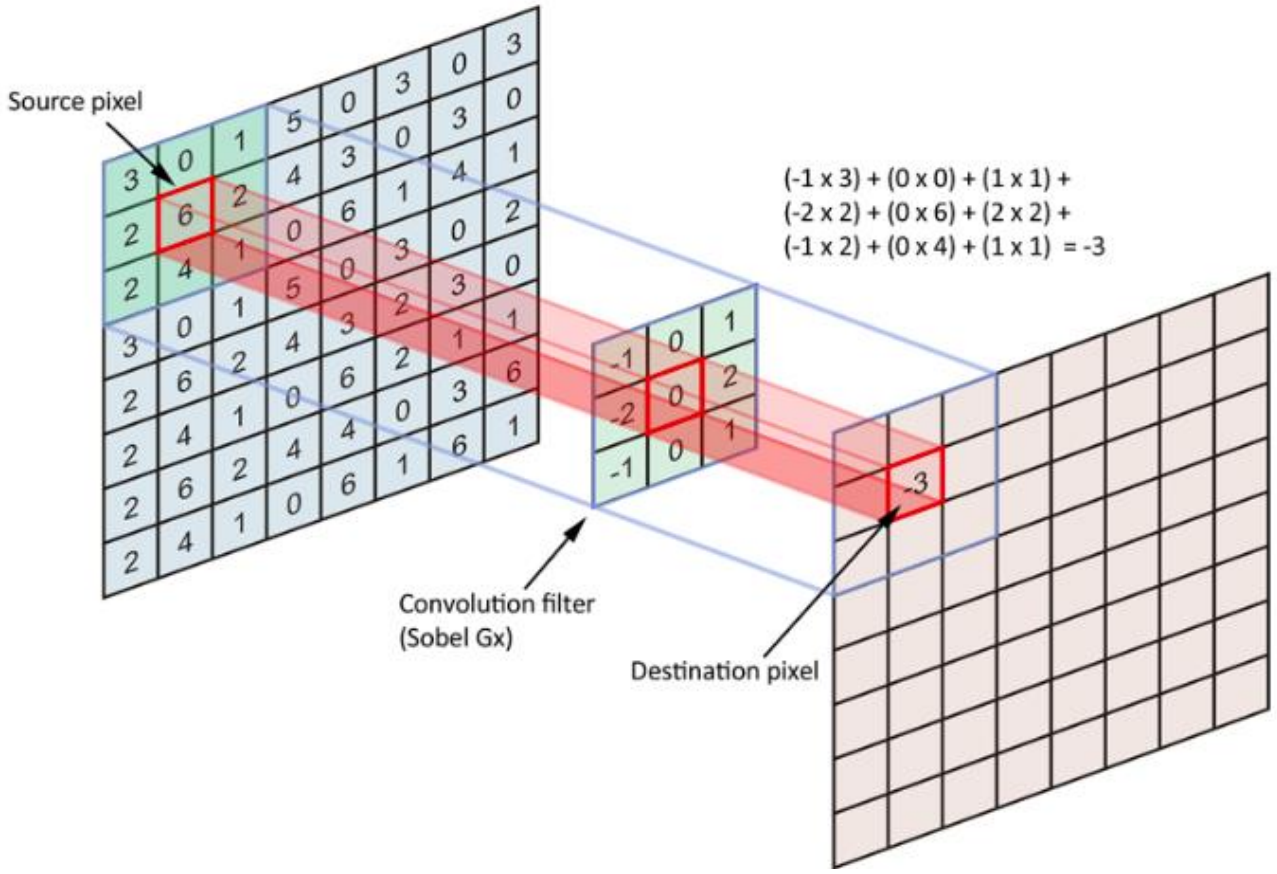


Self-Driving

Synthesis

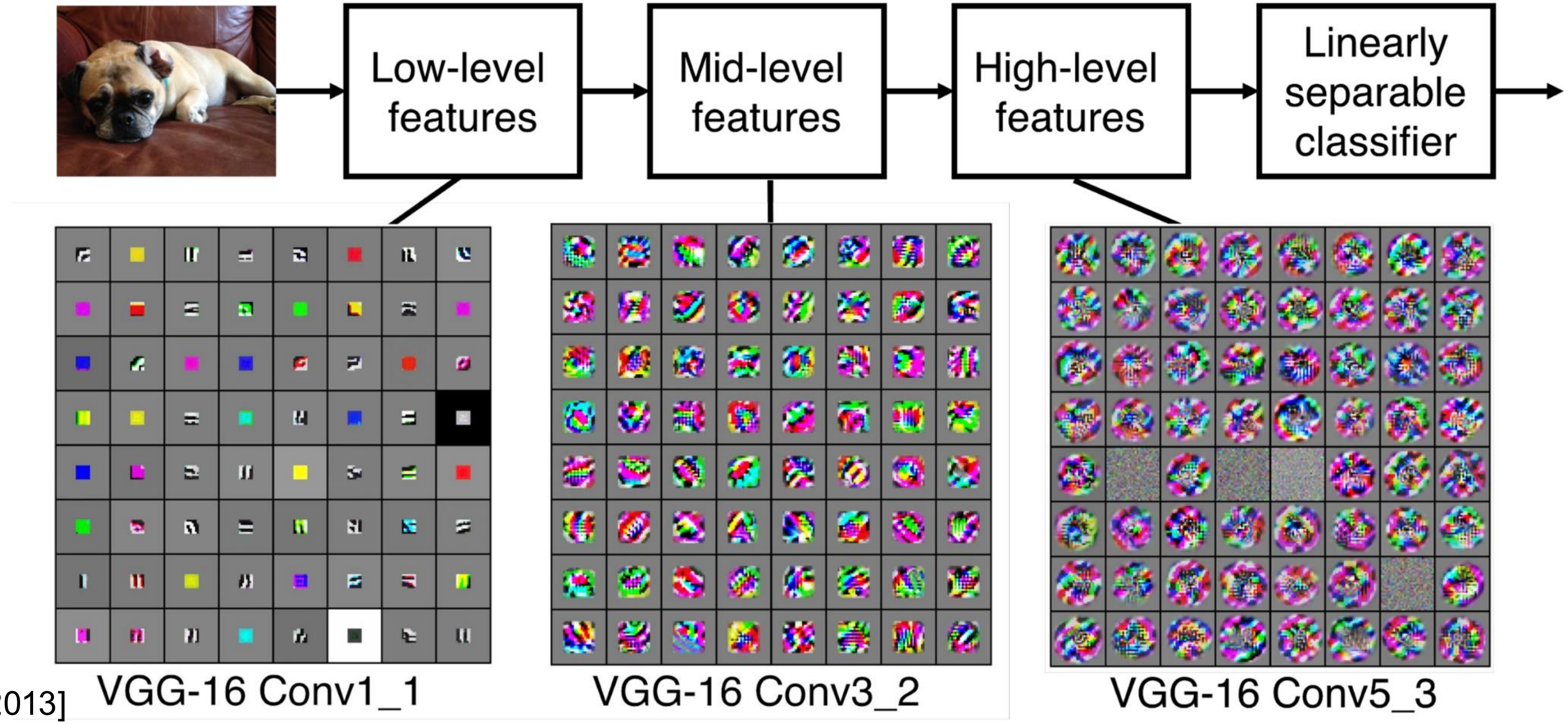
Convolution

- 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



[Zeiler and Fergus 2013]

Parallelizing Convolutional Neural Networks

- Convolutional layers
  - 90-95% of the computation
  - 5% of the parameters
  - Very large intermediate activations

Data parallelism

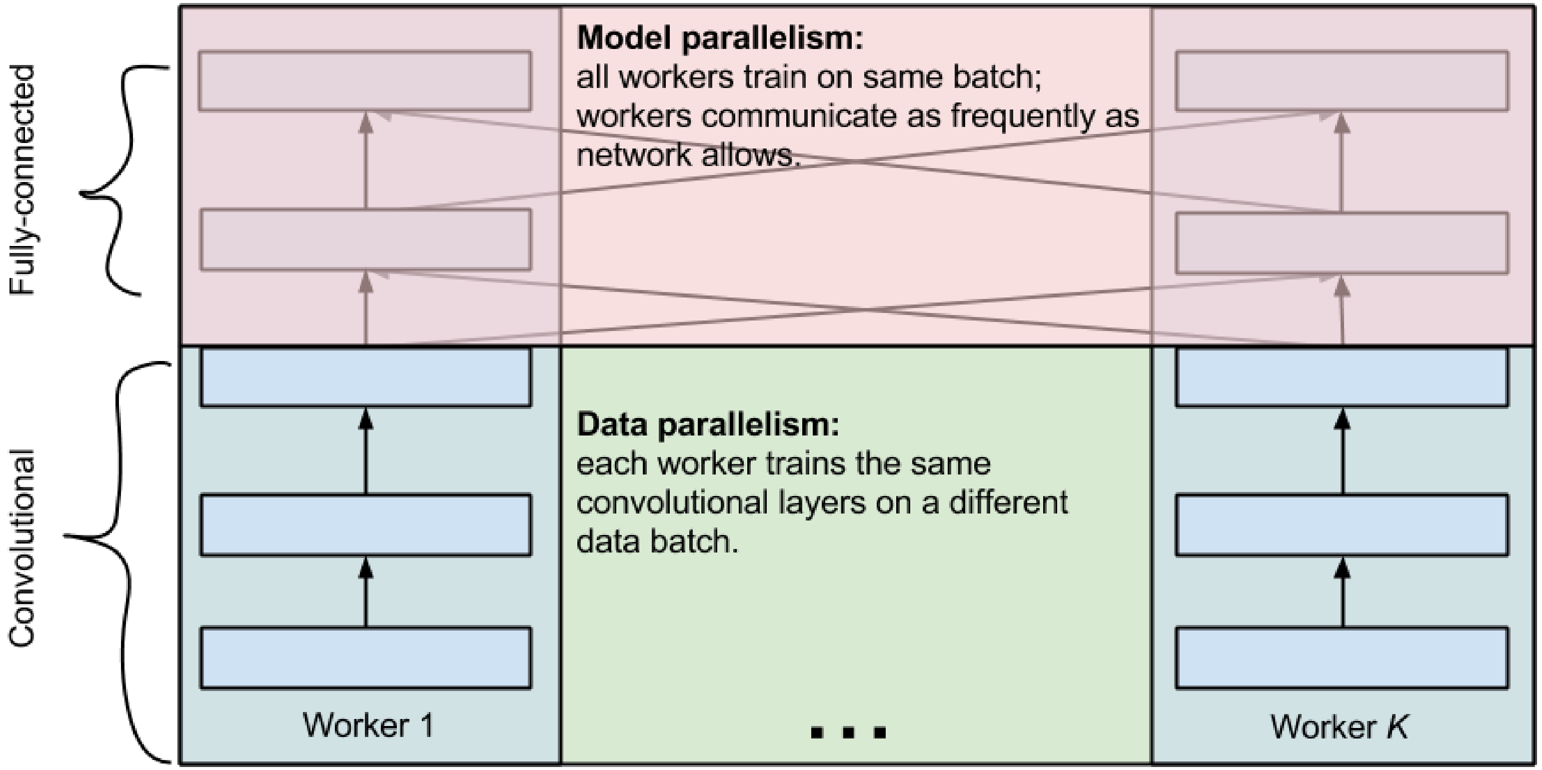
- Fully-connected layers
  - 5-10% of the computation
  - 95% of the parameters
  - Small intermediate activations

Tensor model parallelism

- **Discussion: how to parallelize CNNs?**

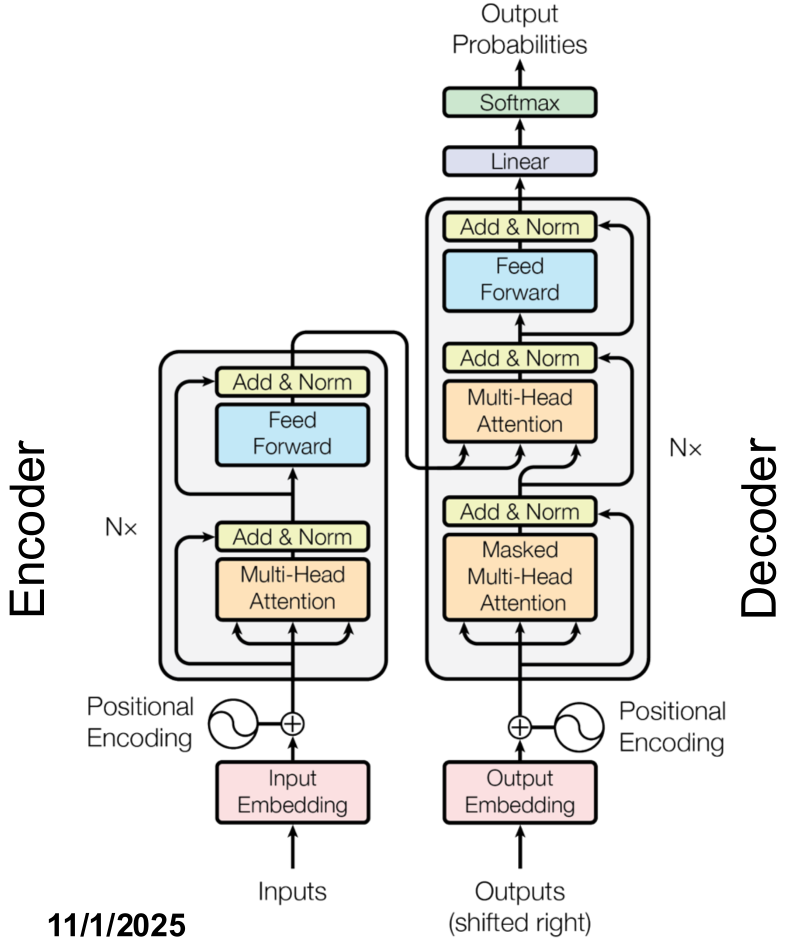
Parallelizing Convolutional Neural Networks

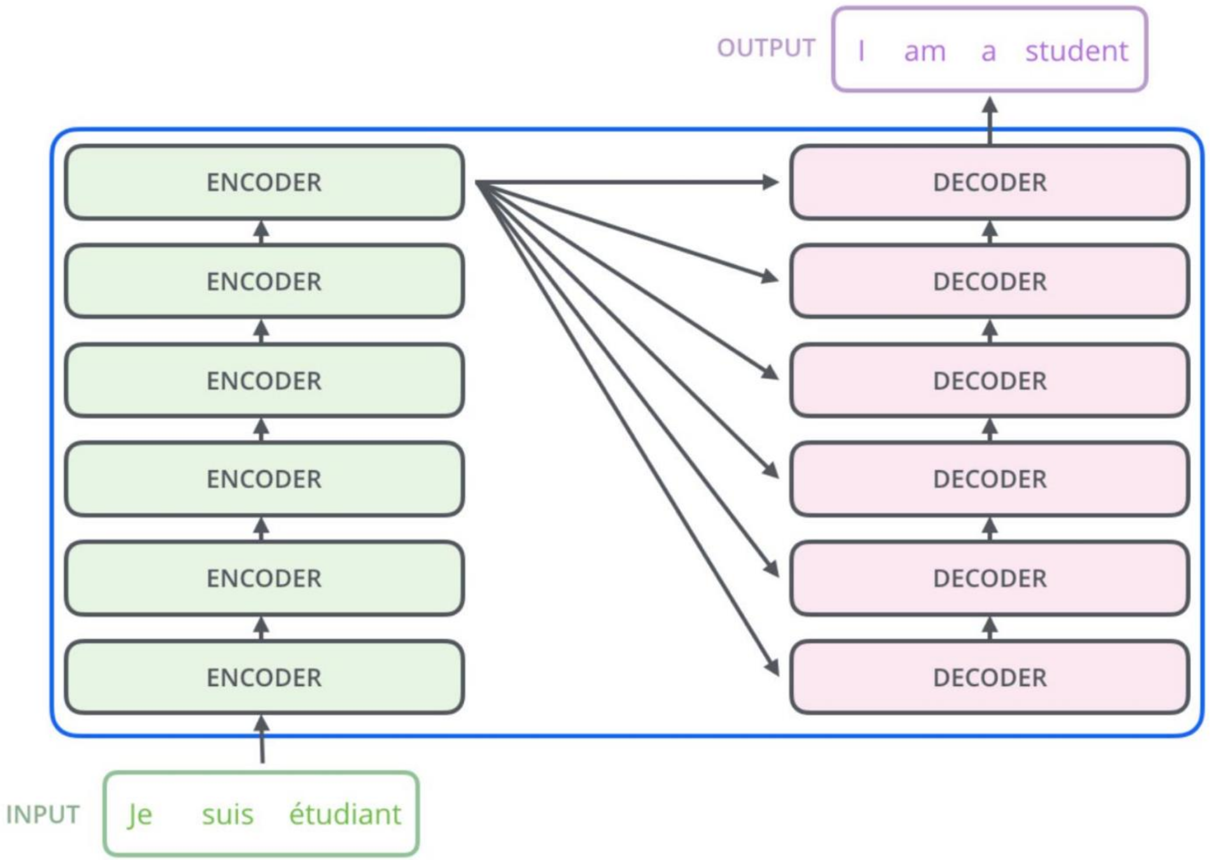
- Data parallelism for convolutional layers
- Tensor model parallelism for fully-connected layers



Example: Parallelizing Transformers

- Transformer: attention mechanism for language understanding



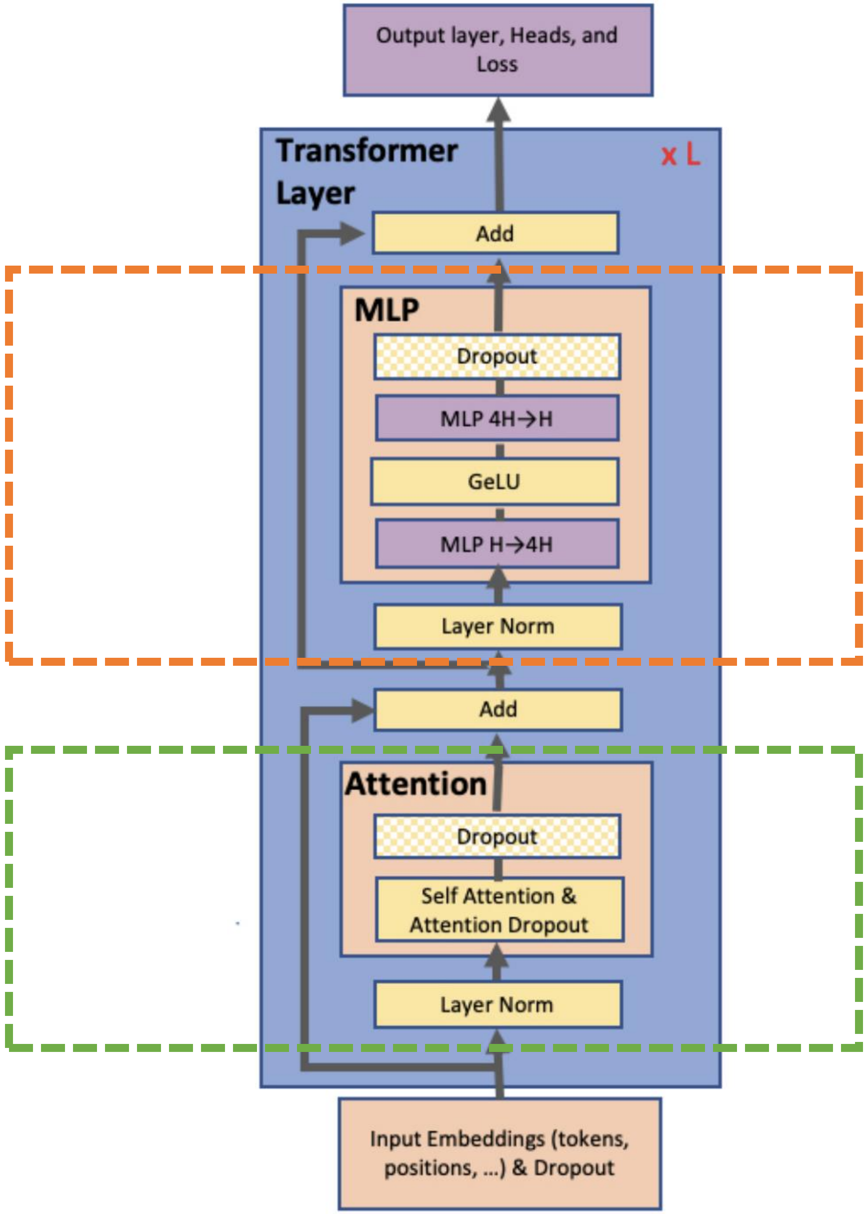


Ashish Vaswani et. al. Attention is all you need.

A Single Transformer Layer

Fully-Connected Layers

Self-Attention Layers



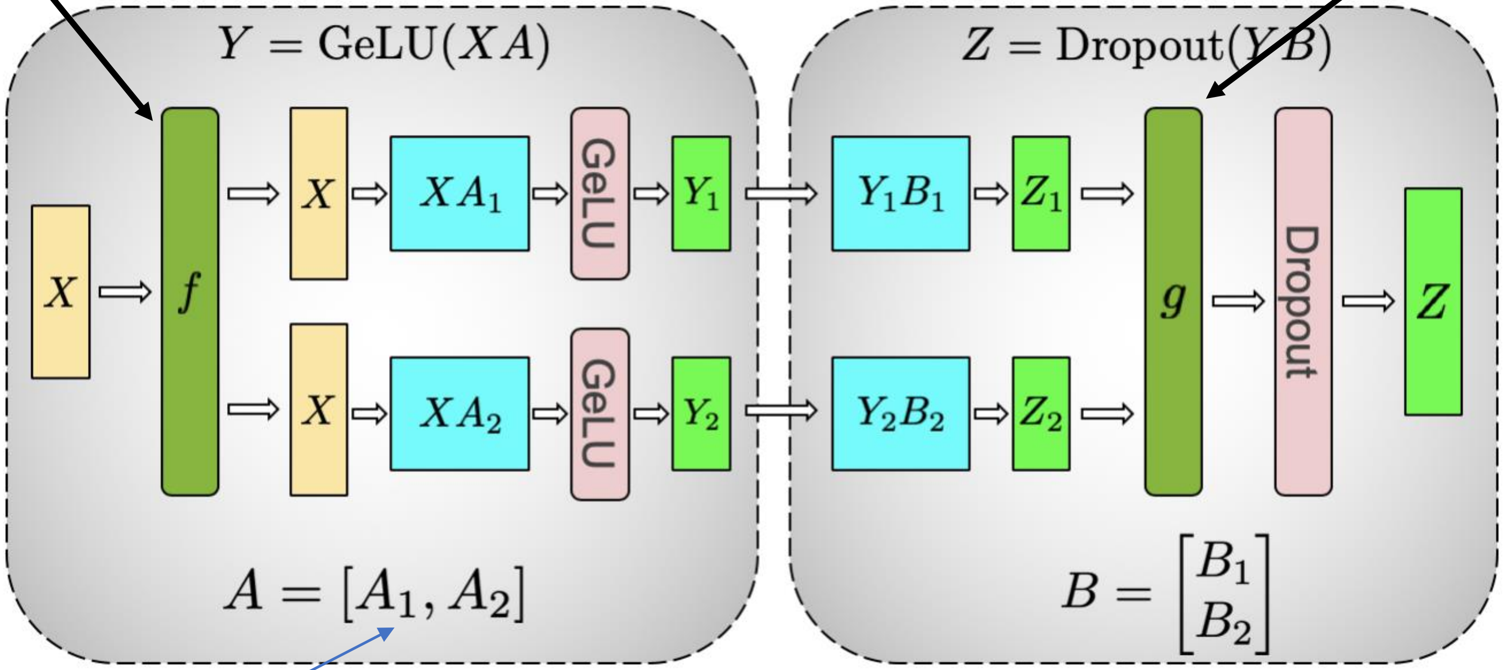
Parallelizing Fully-Connected Layers in Transformers

Weights

$$\boldsymbol{Y = GeLU(X \times A)}$$
$$\boldsymbol{Z = Dropout(Y \times B)}$$

identity layer → Replicate X

reduction layer



Z = Z1 + Z2

Partition on Col dim

Y = [Y1 Y2]

Tensor model parallelism
(partition output)

Tensor model parallelism
(reduce output)

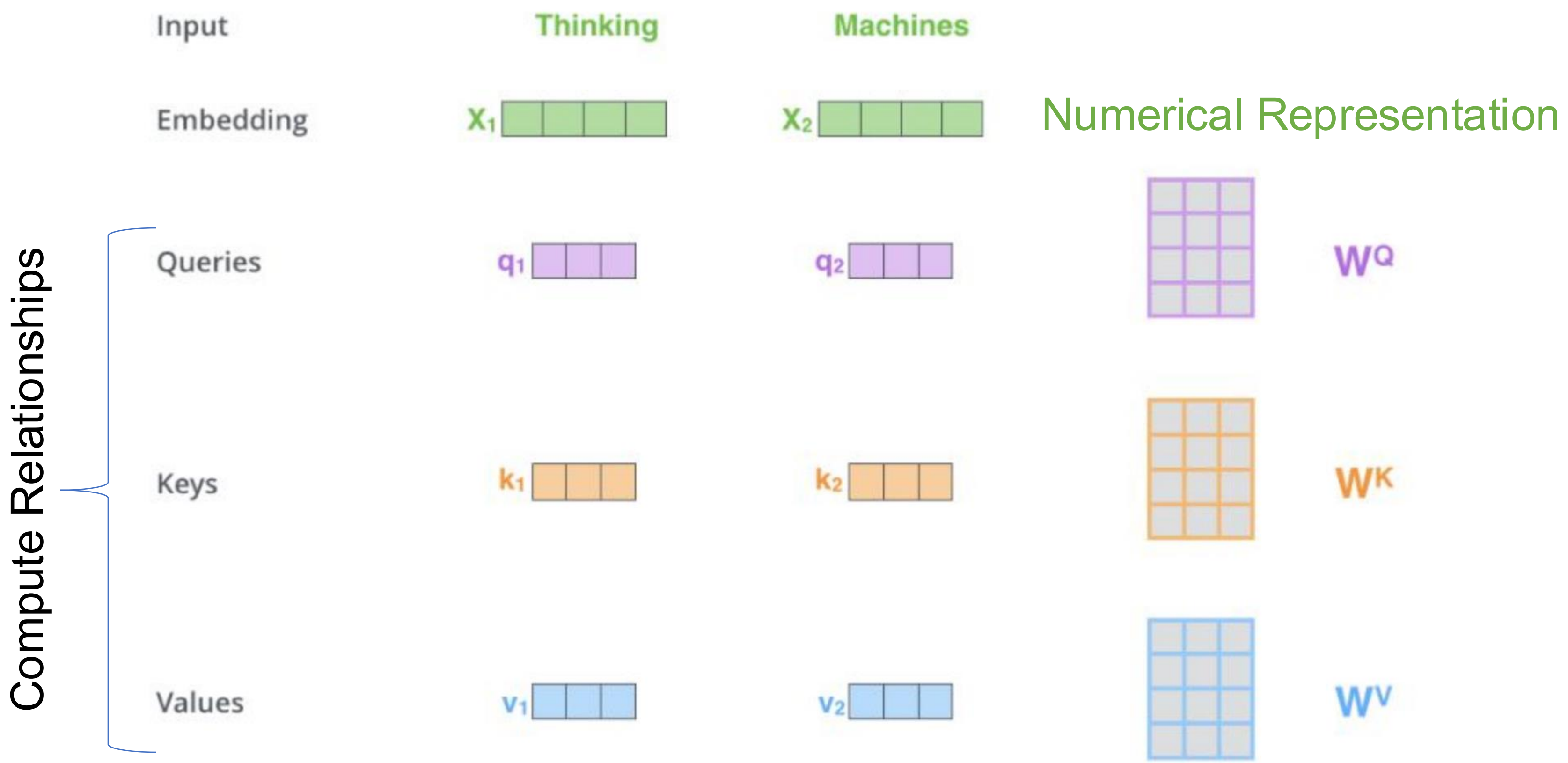
Why switch?

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism.

Self-Attention

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

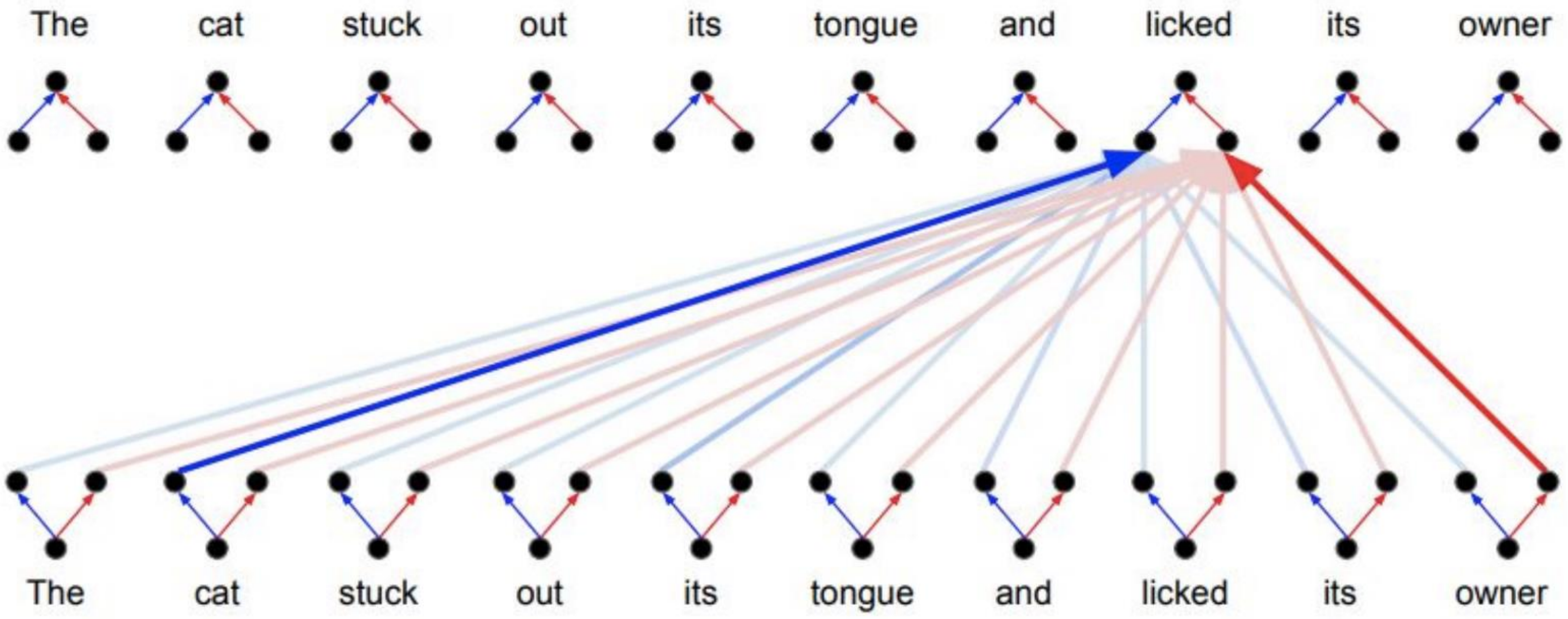
- Mapping a query and a set of key-value pairs to an output



Slide credit: Jay Allamar

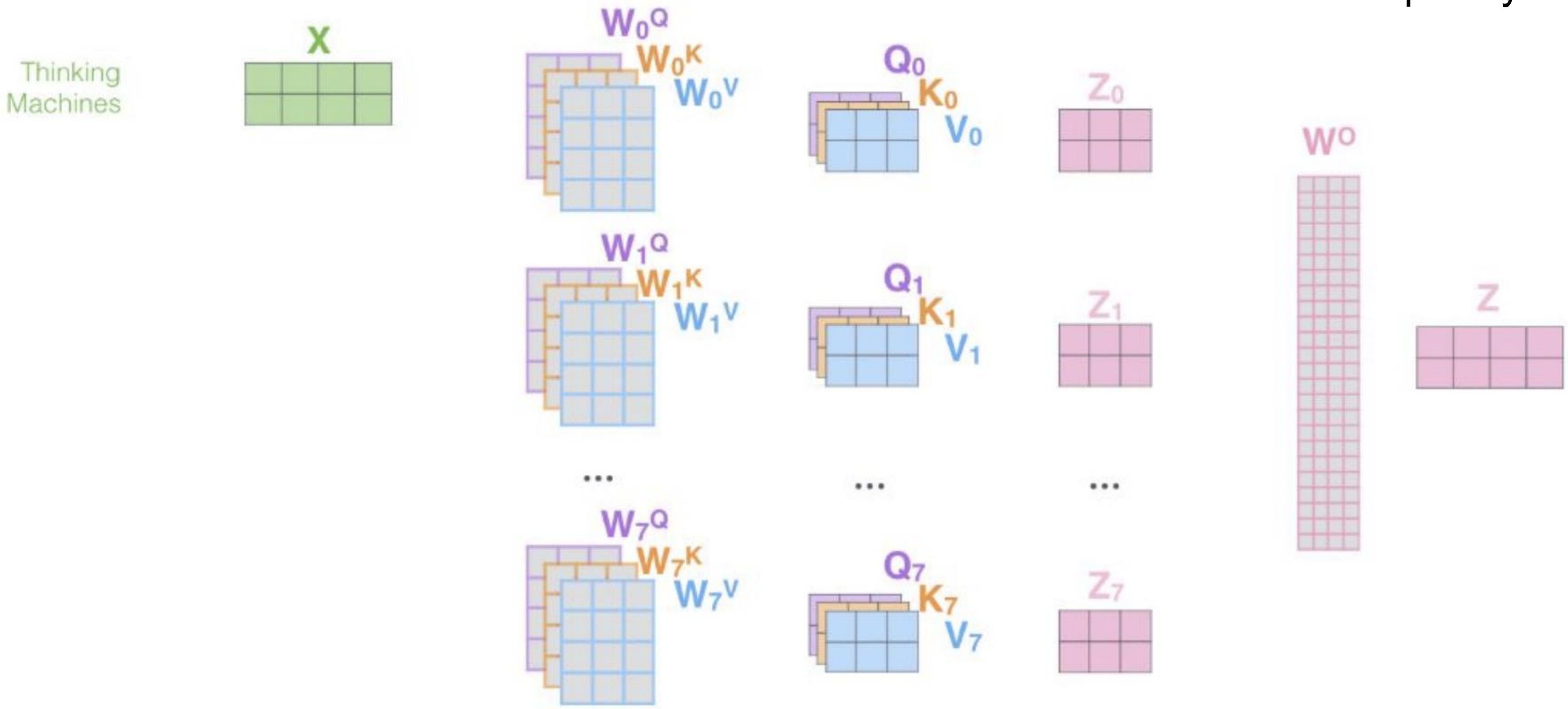
Multi-Head Self-Attention

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



Multi-Head Self-Attention

Why multi-head attention?
→ Reduce complexity of matrix multiply!



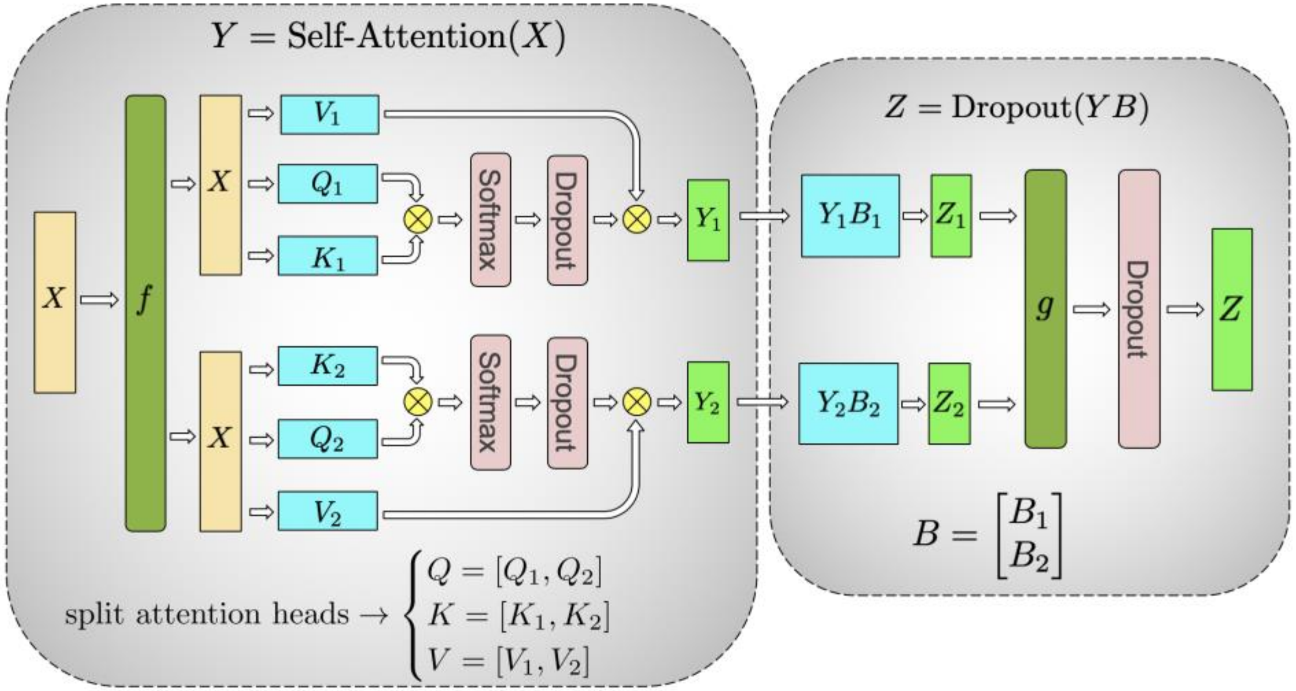
$$Z_i = A(Q_i, K_i, V_i) = softmax\left(\frac{Q_i {K_i}^T}{\sqrt{d}}\right) V_i$$

$$Z = MultiHead(Q, K, V) = Concat(Z_0, \dots, Z_7) W^o$$

Parallelizing Self-Attention Layers in Transformers

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

$$Z = MultiHead(Q, K, V) = Concat(Y_0, \ldots, Y_h) W^o$$

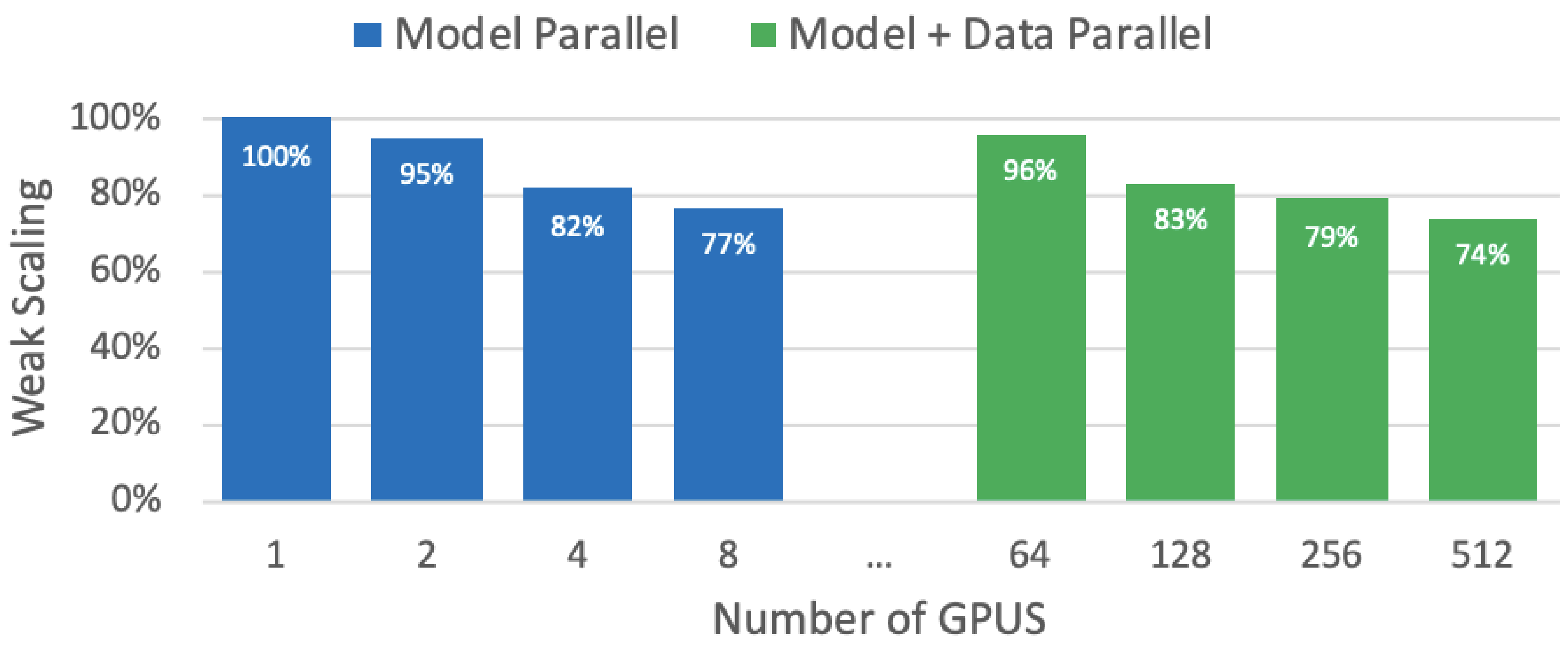


Parallelizing across attention heads

Tensor model parallelism (reduce output)

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism.

Parallelizing Transformers



Scale to 512 GPUs by combining data and model parallelism

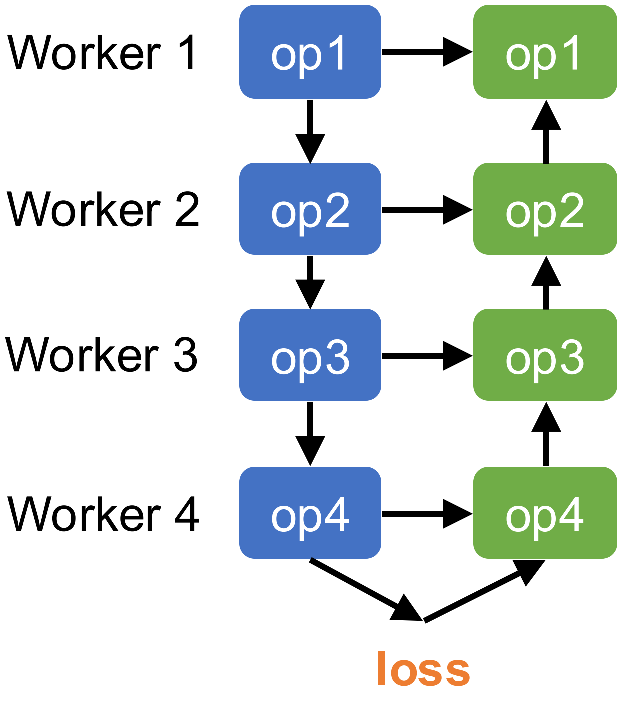
Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism.

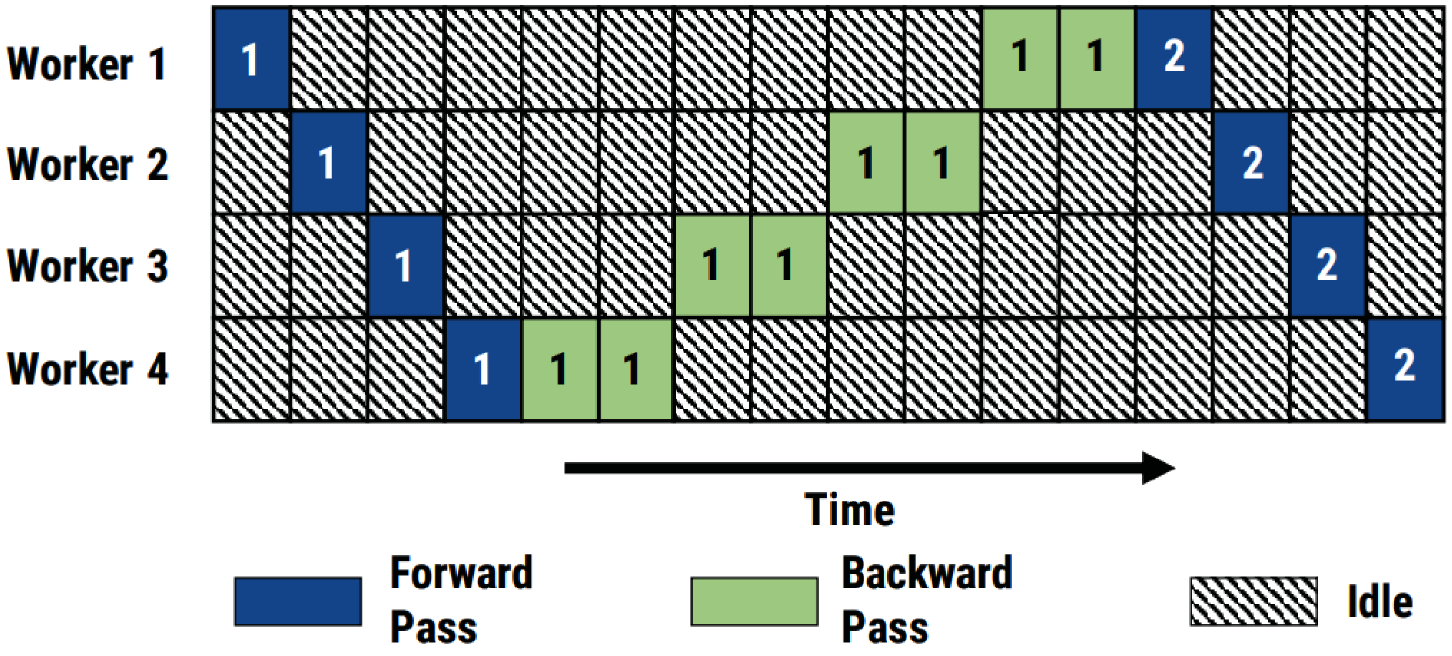
How to parallelize DNN Training?

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

An Issue with Model Parallelism

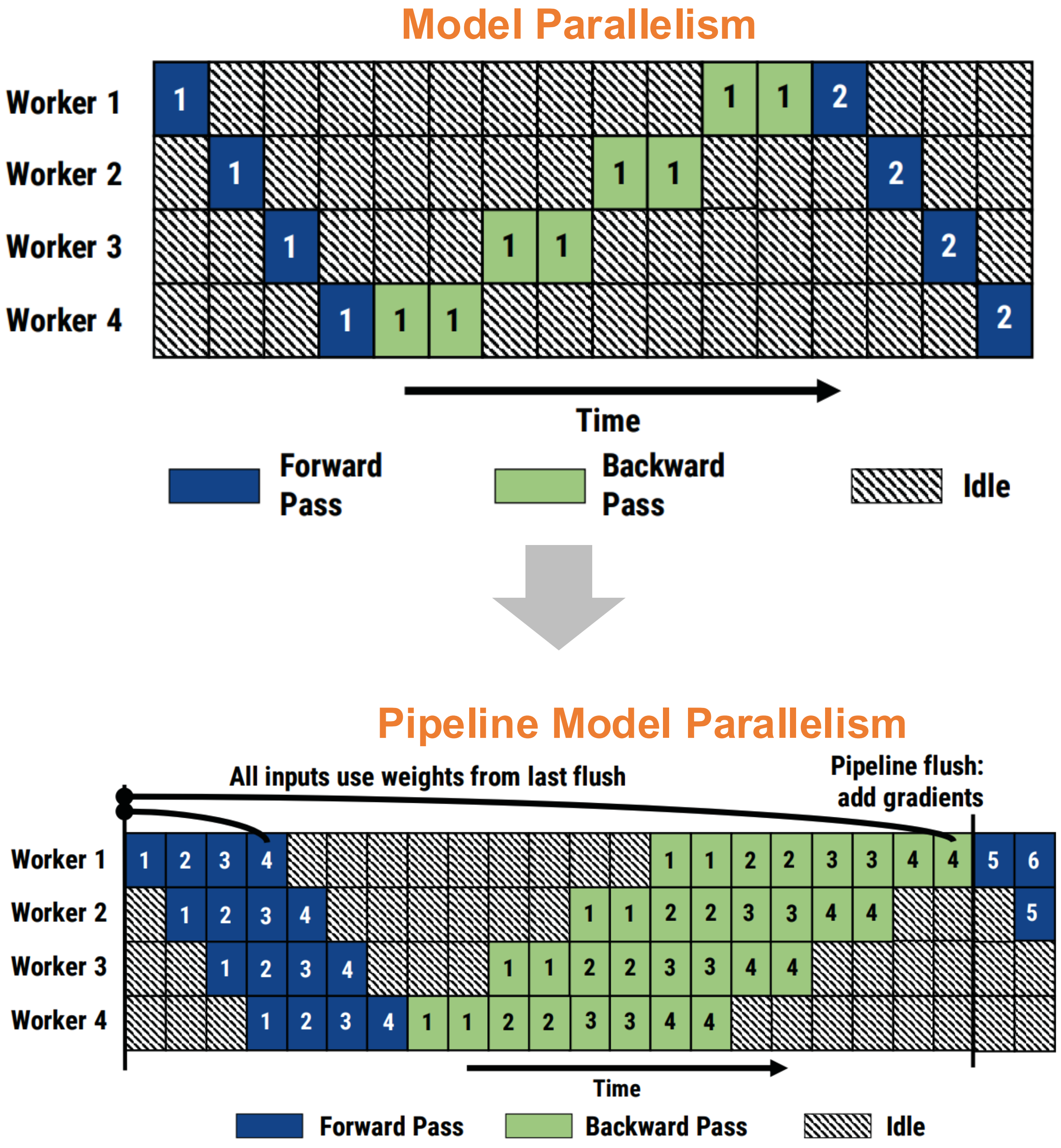
- Under-utilization of compute resources
- Low overall throughput due to resource utilization





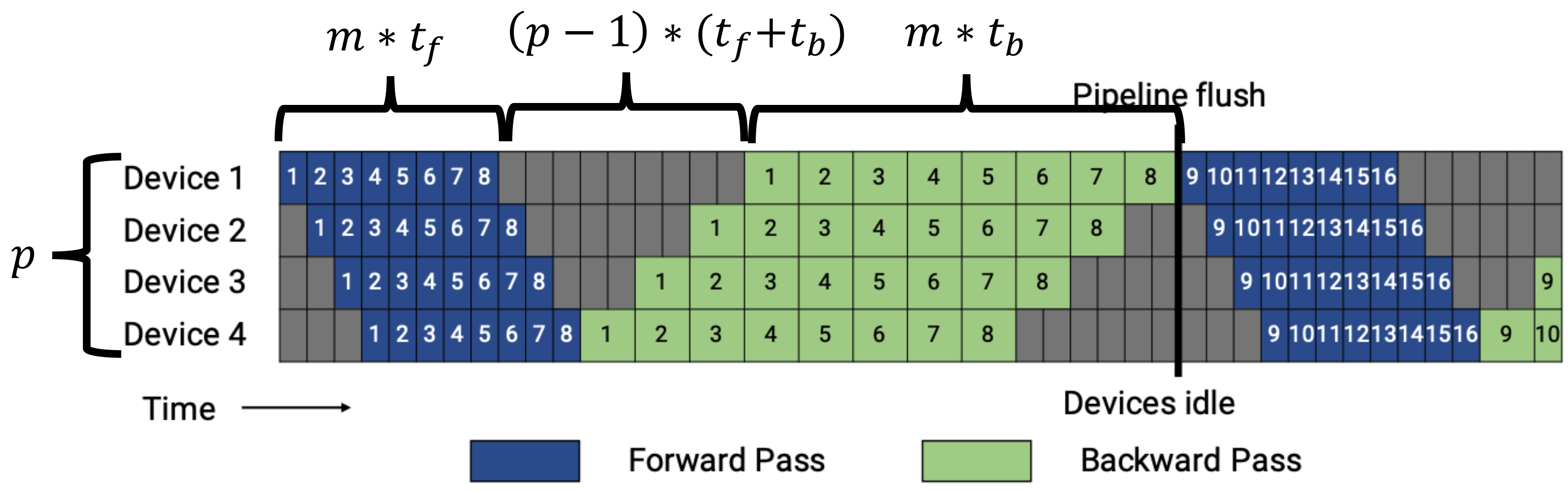
Pipeline Model Parallelism

- **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



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

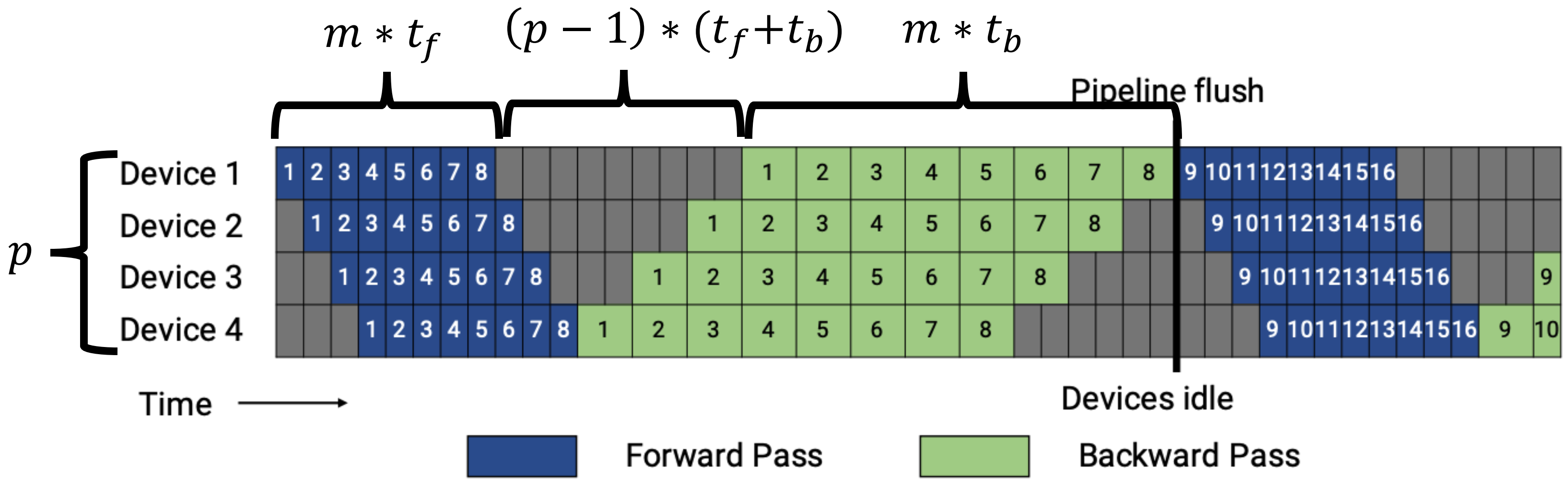
GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism

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

$$m = \frac{minibatch}{microbatch}$$

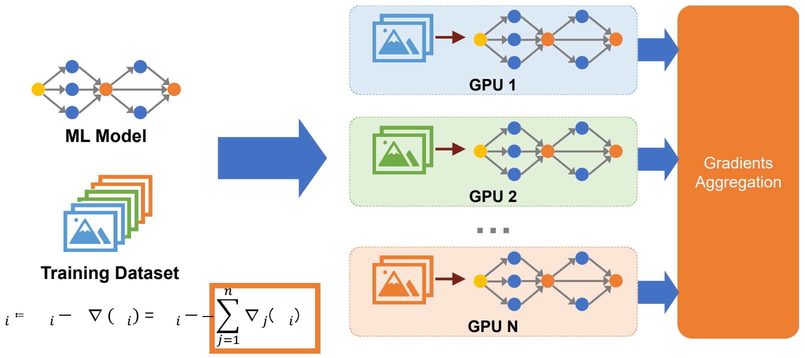
- p: number of pipeline stages
  - Decrease pipeline depth
  - Caveat: increase stage size

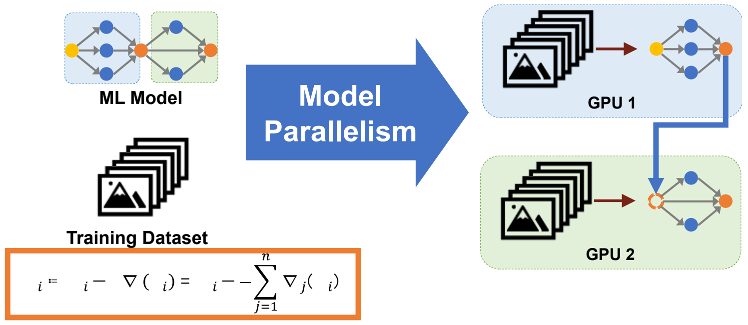


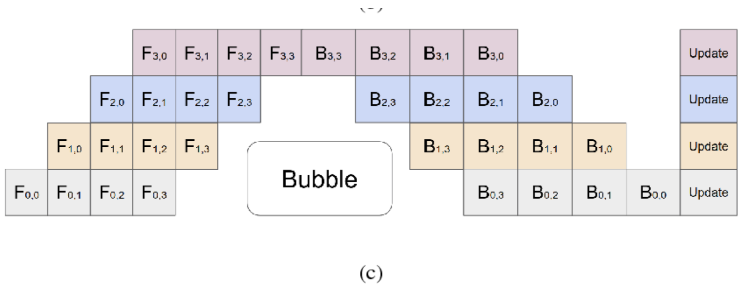
$$\boldsymbol{BubbleFraction = \frac{(p-1) * (t_f + t_b)}{m * t_f \ + m * t_b} = \frac{p-1}{m}}$$

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism

Summary: Comparing Data/Model/Pipeline Parallelism

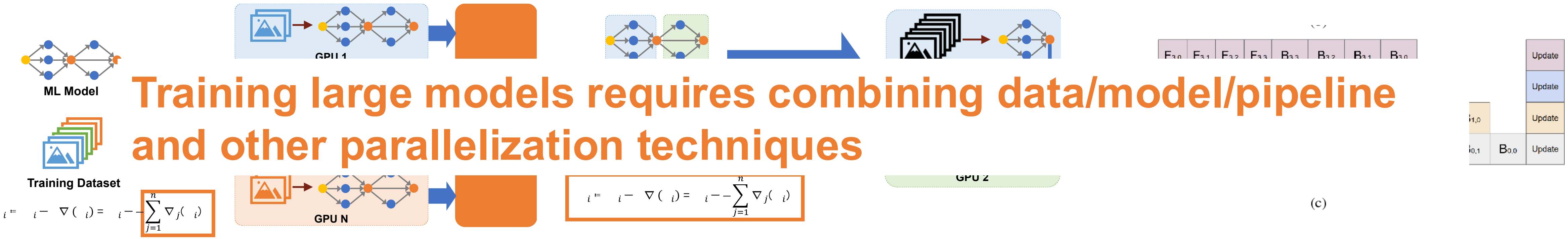






	Data Parallelism	Model Parallelism	Pipeline Parallelism
Pros	✓ Massively parallelizable ✓ Require no communication during forward/backward	✓ Support training large models ✓ Efficient for models with large numbers of parameters	✓ Support large-batch training ✓ Efficient for deep models
Cons	❖ Do not work for models that cannot fit on a GPU ❖ Do not scale for models with large numbers of parameters	❖ Limited parallelizability; cannot scale to large numbers of GPUs ❖ Need to transfer intermediate results in forward/backward	❖ Limited utilization: bubbles in forward/backward

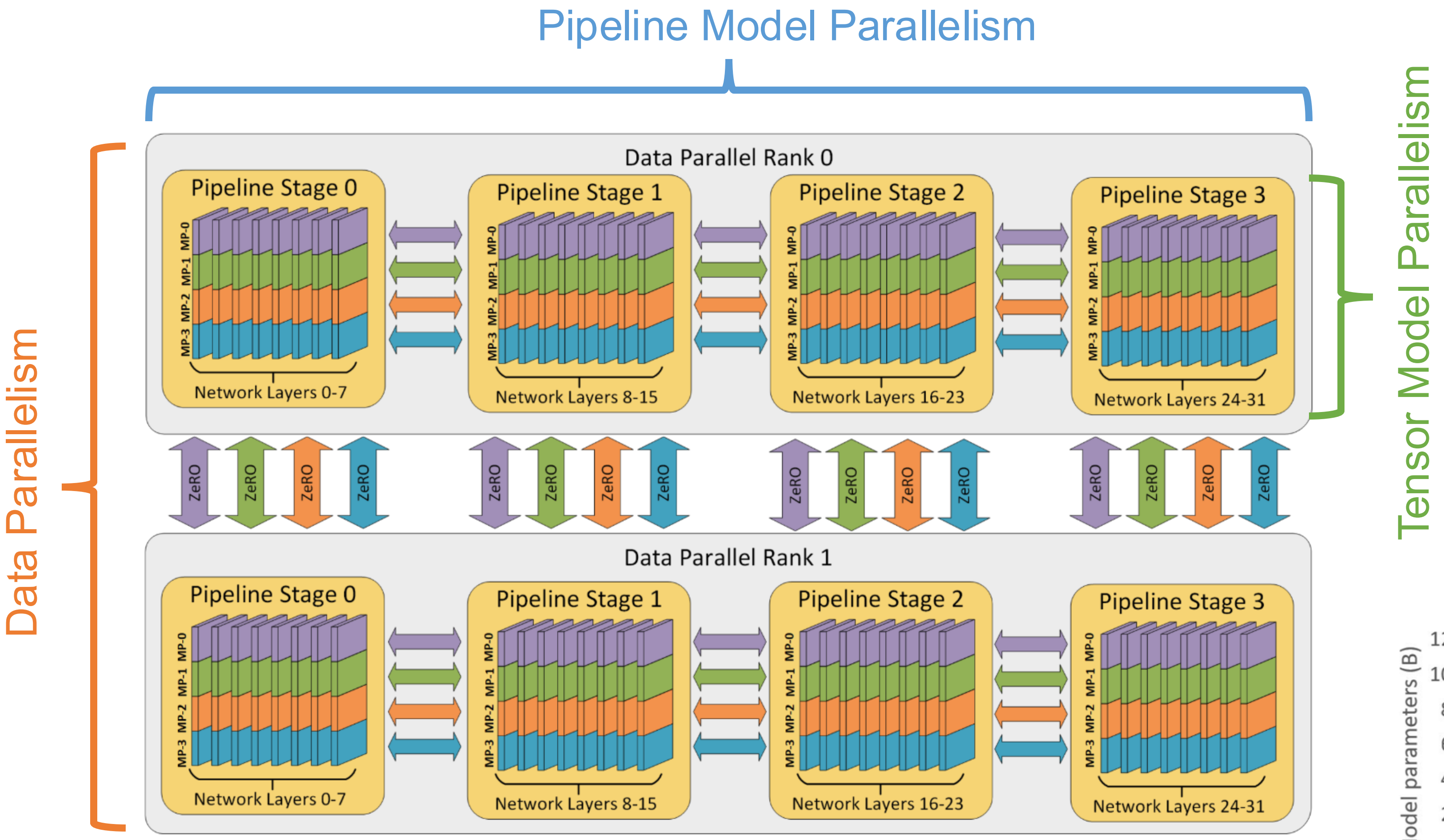
Summary: Data/Model/Pipeline Parallelism

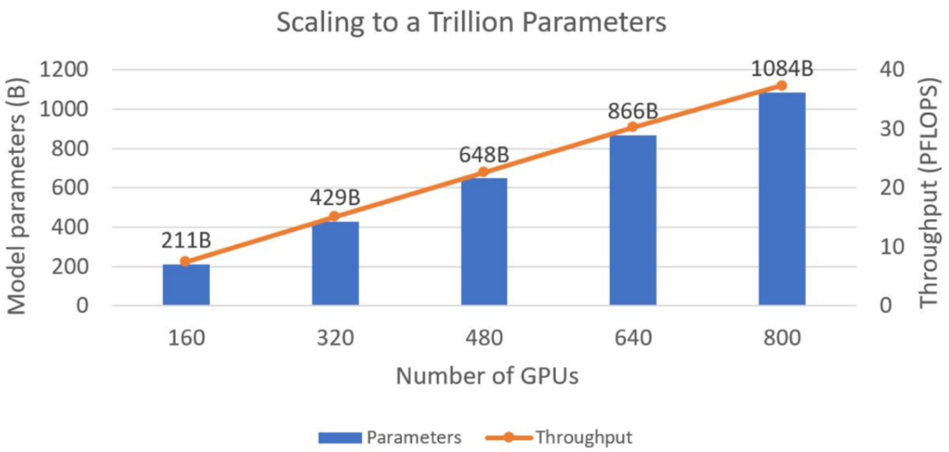


Training large models requires combining data/model/pipeline and other parallelization techniques

	Data Parallelism	Model Parallelism	Pipeline Parallelism
Pros	✓ Massively parallelizable ✓ Require no communication during forward/backward	✓ Support training large models ✓ Efficient for models with large numbers of parameters	✓ Support large-batch training ✓ Efficient for deep models
Cons	❖ Do not work for models that cannot fit on a GPU ❖ Do not scale for models with large numbers of parameters	❖ Limited parallelizability; cannot scale to large numbers of GPUs ❖ Need to transfer intermediate results in forward/backward	❖ Limited utilization: bubbles in forward/backward

Example: 3D parallelism in DeepSpeed





https://www.microsoft.com/en-us/research/blog/deepspeed-extreme-scale-model-training-for-everyone/