15-442/15-642: Machine Learning Systems

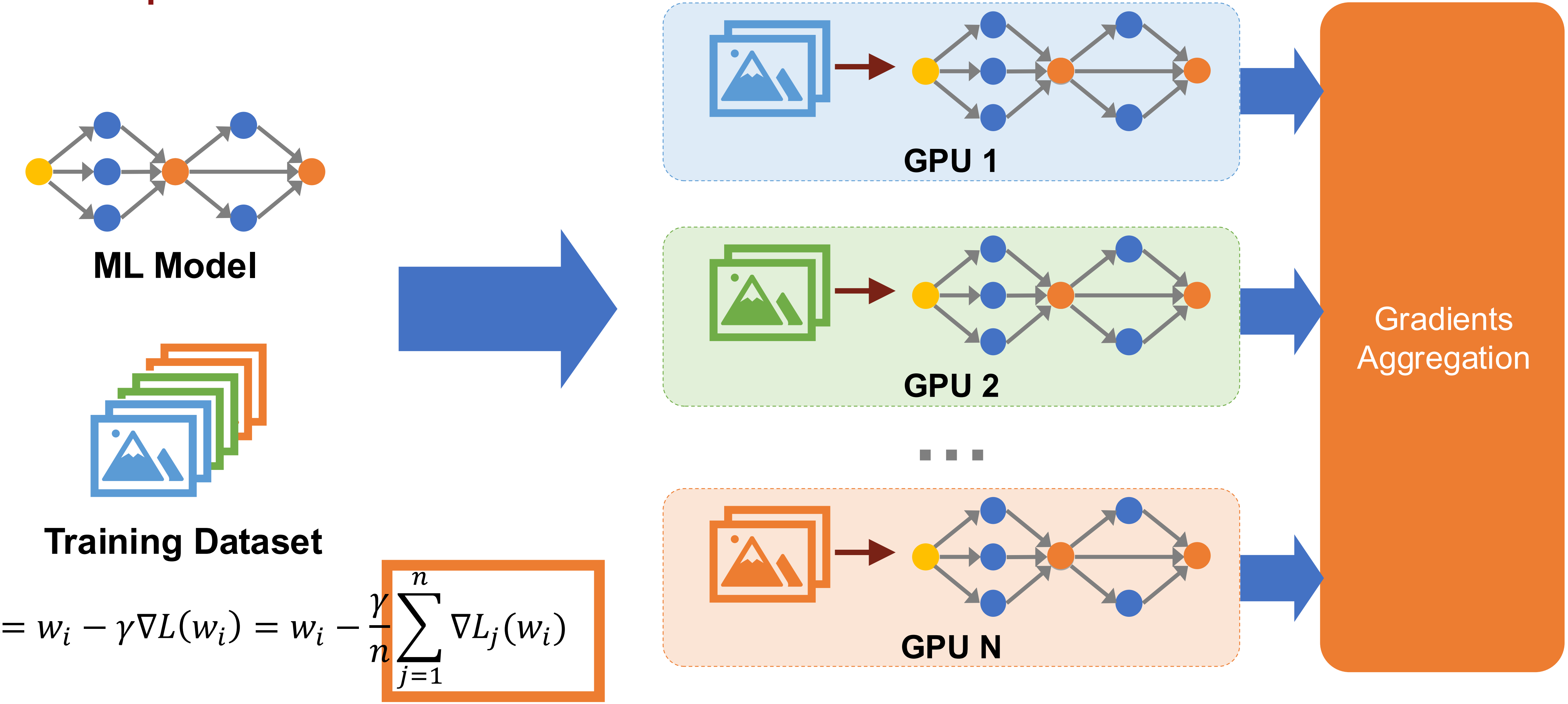
Parallelization Part 2 (Model and Pipeline Parallelism)

Tianqi Chen and Zhihao Jia

Carnegie Mellon University

2/17/2026

Recap: Data Parallelism



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

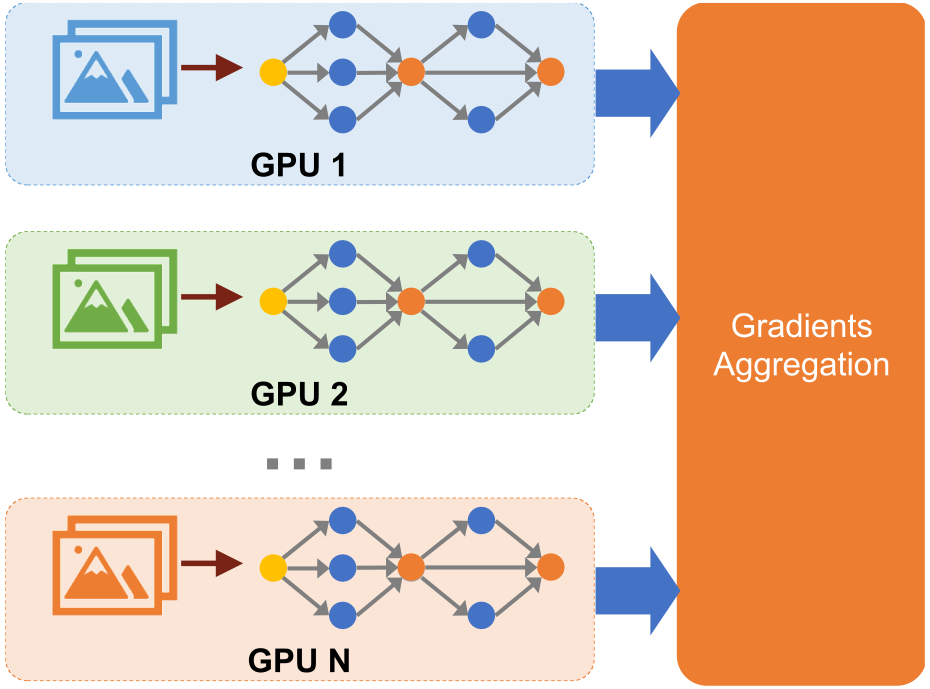
1. Partition training data into batches

2. Compute the gradients of each batch on a GPU

3. Aggregate gradients across GPUs

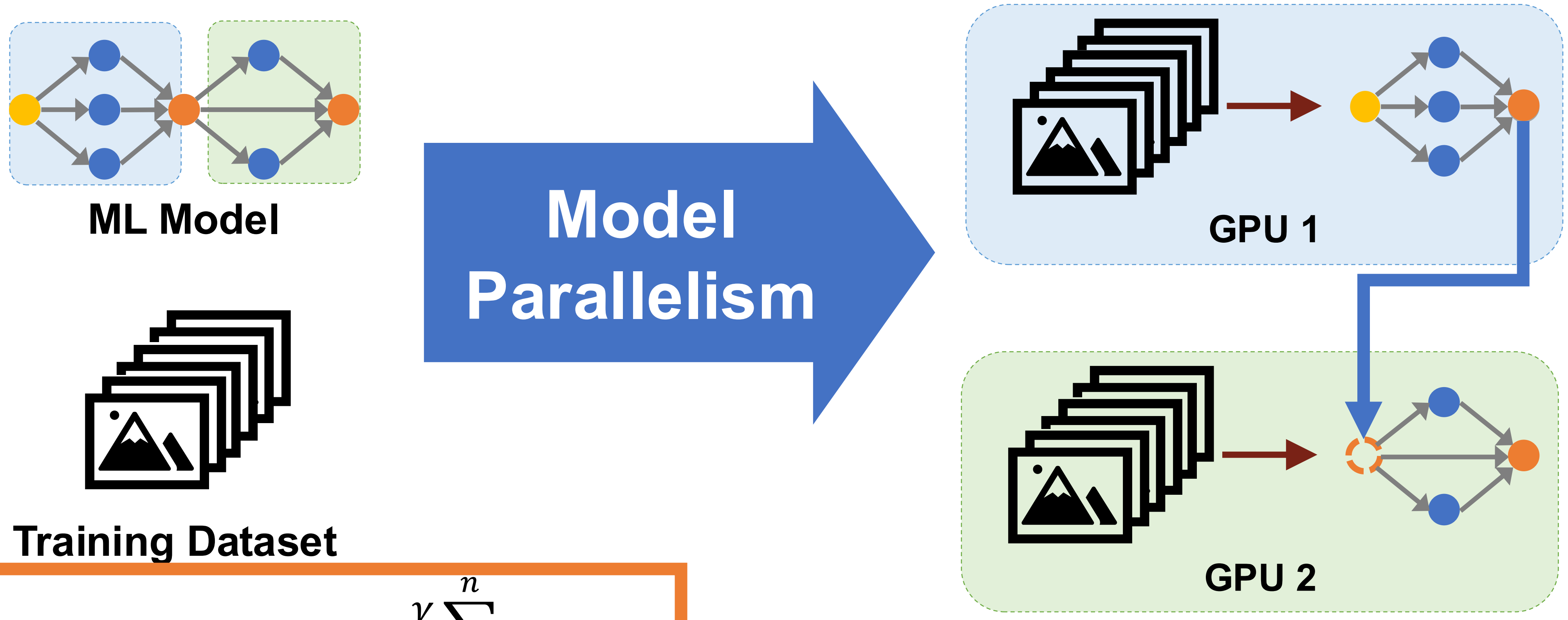
Recap: An Issue with Data Parallelism

- 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



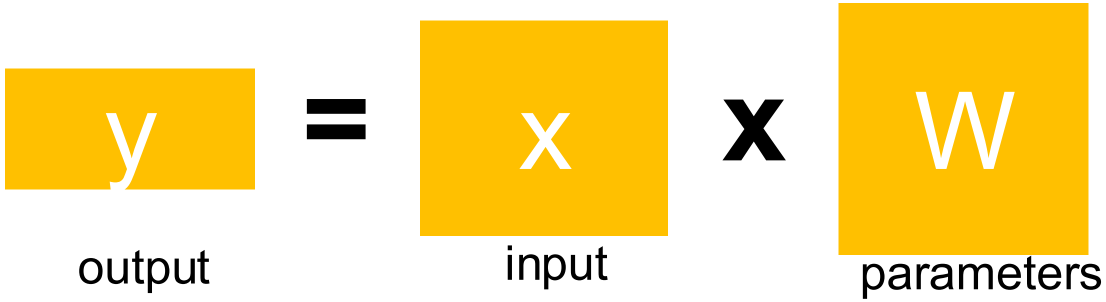
Transfer intermediate results between devices

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

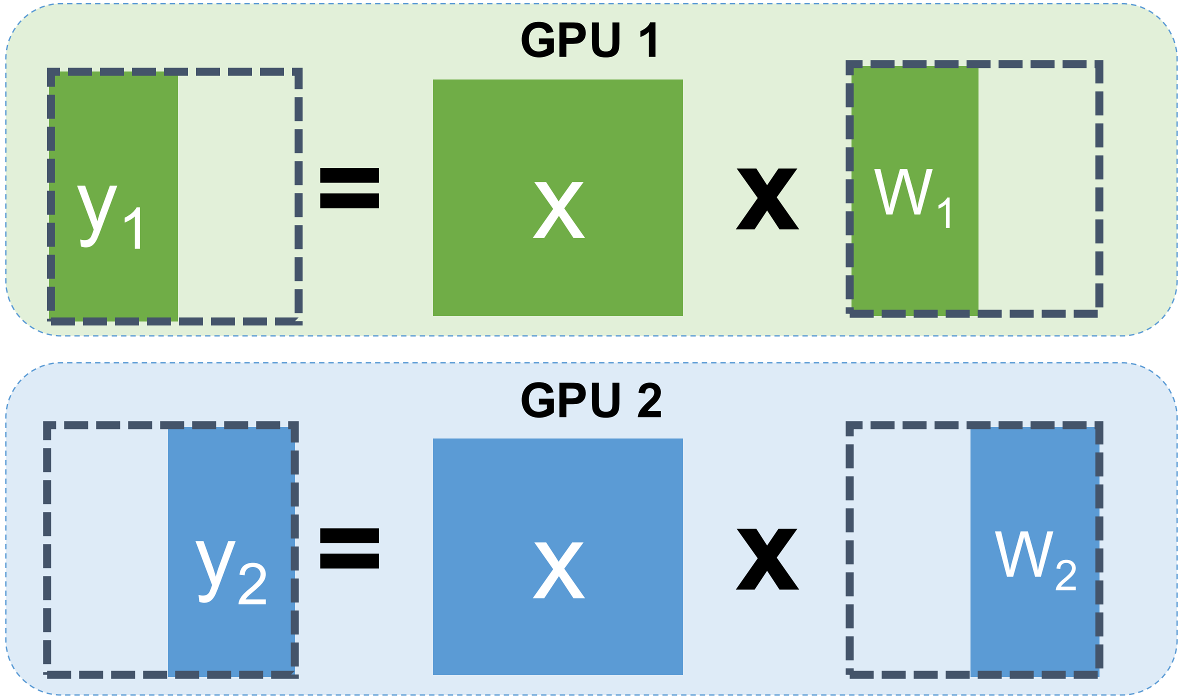
How to parallelize DNN Training?

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

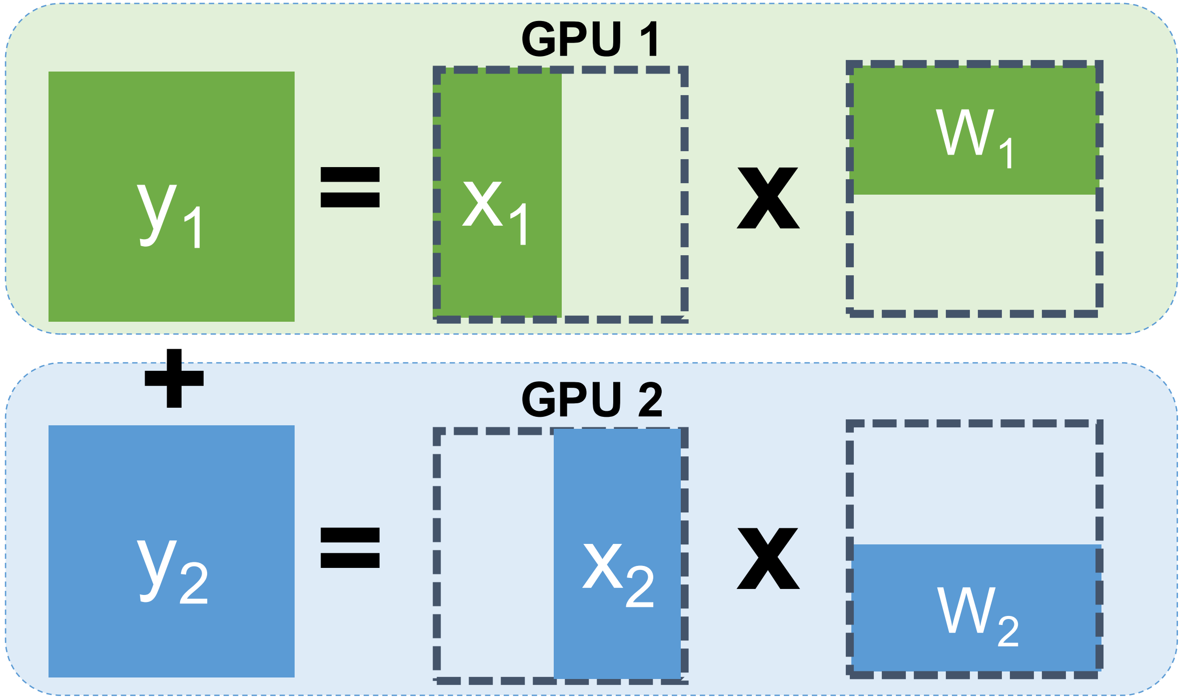
Tensor Model Parallelism



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



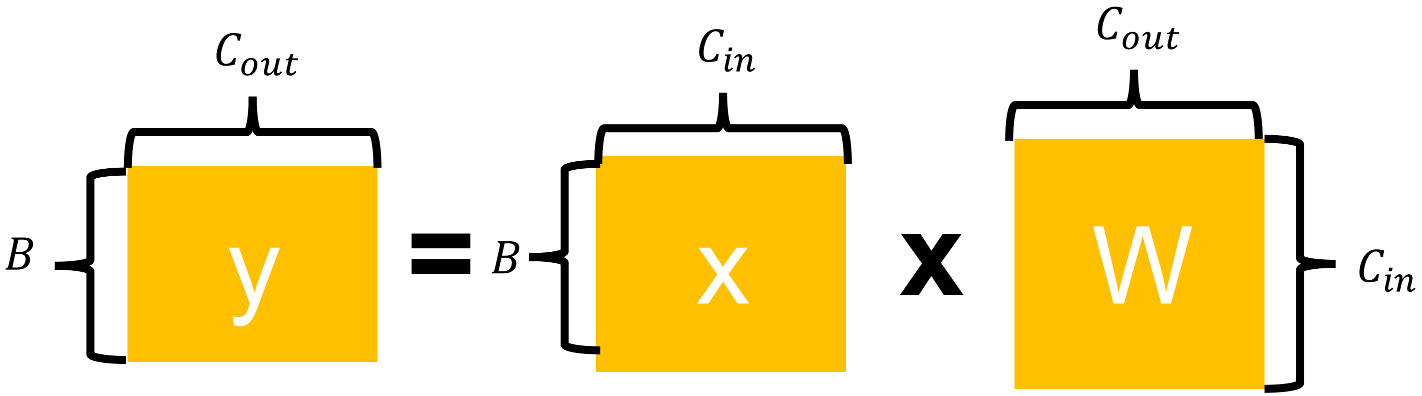
Tensor Model Parallelism (partition output)

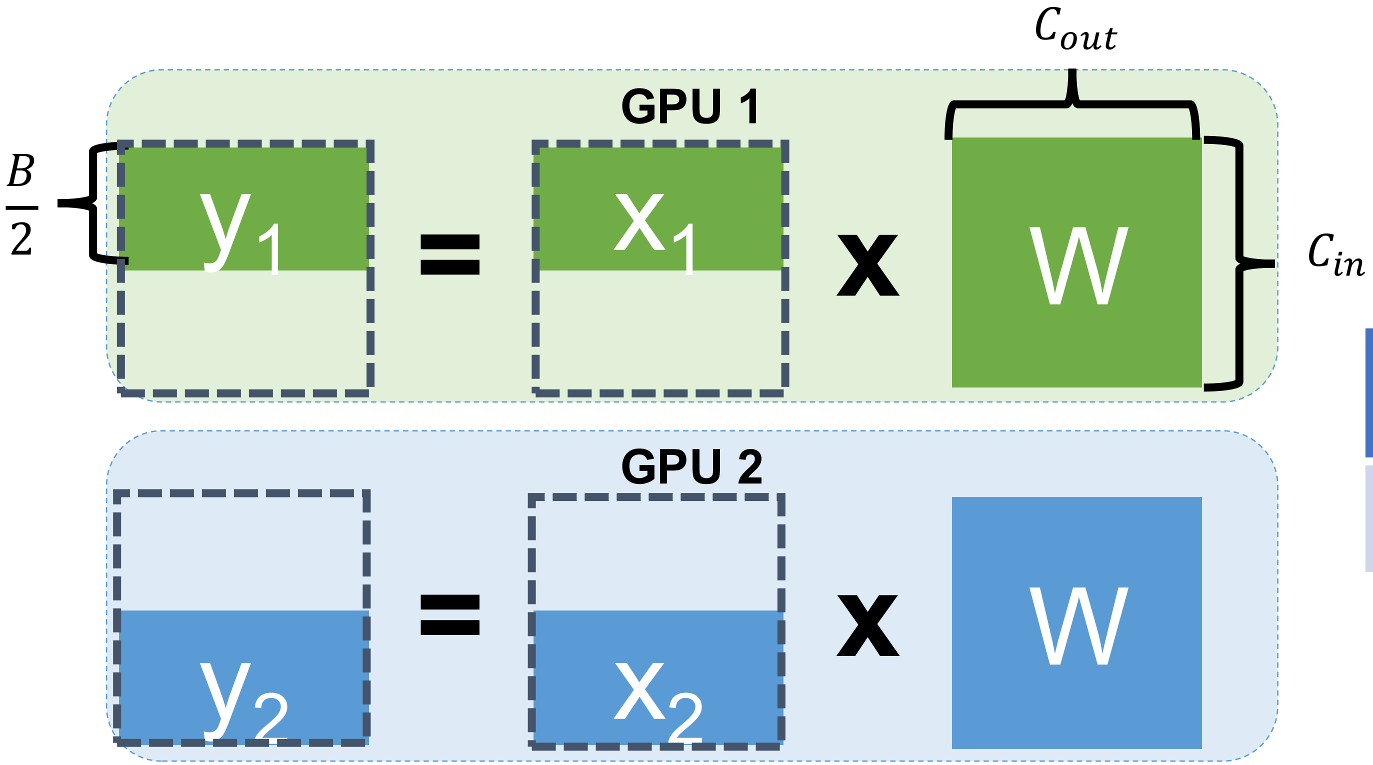


Tensor Model Parallelism (reduce output)

$$y = y1 + y2$$

Comparing Data and Tensor Model Parallelism



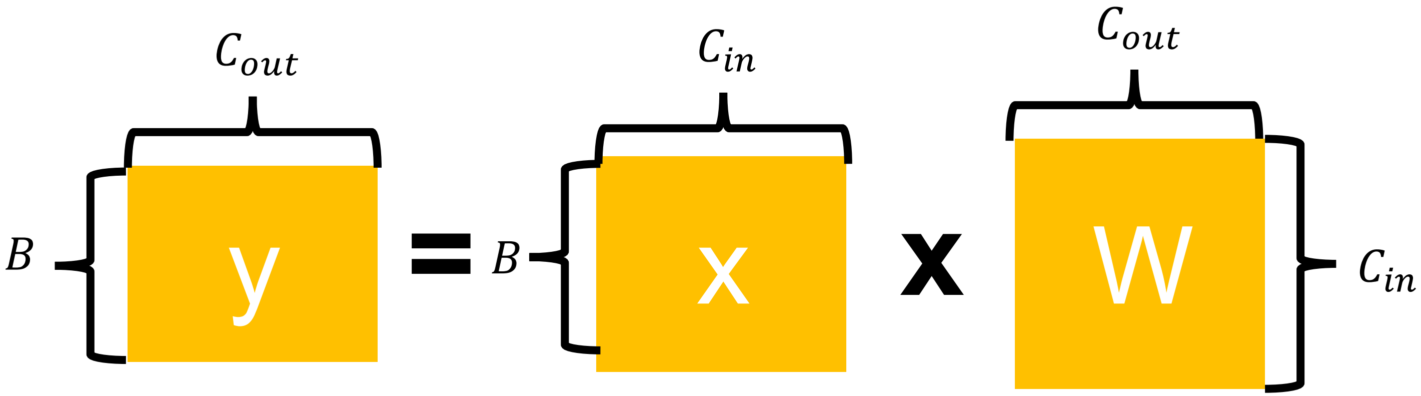


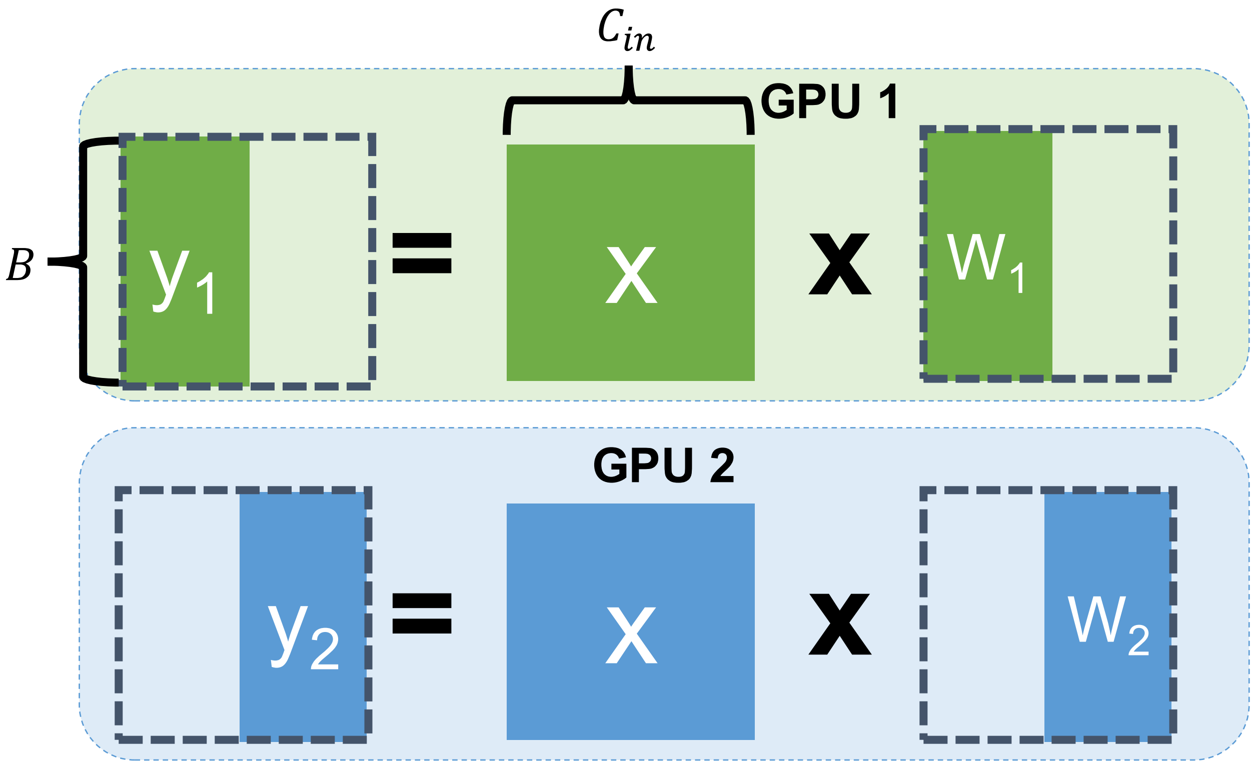
Data parallelism $y = Wx$

Forward Processing	Backward Propagation	Gradients Sync
0	0	$O(C_{out} * C_{in})$

Communication Cost of Data Parallelism

Comparing Data and Tensor Model Parallelism



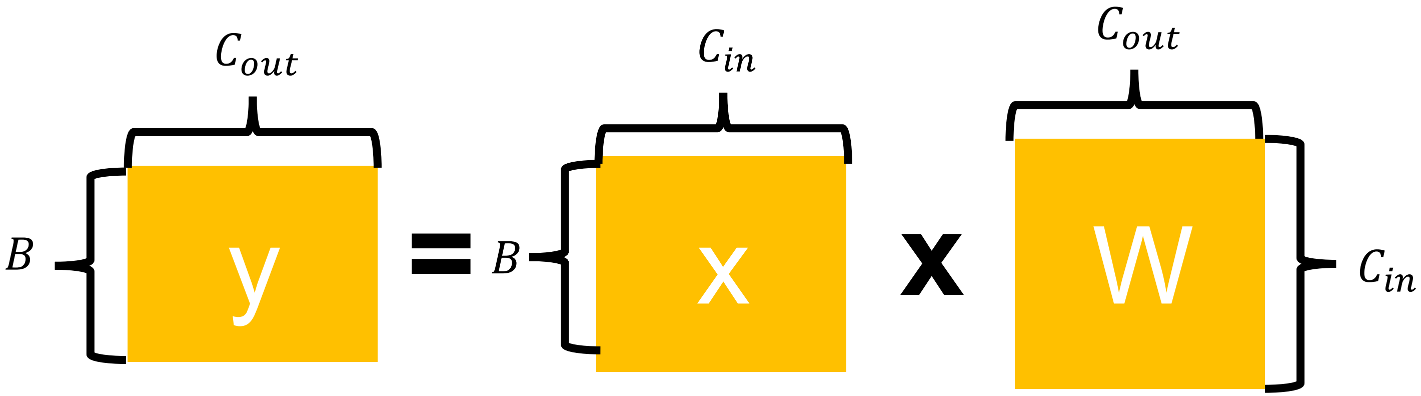


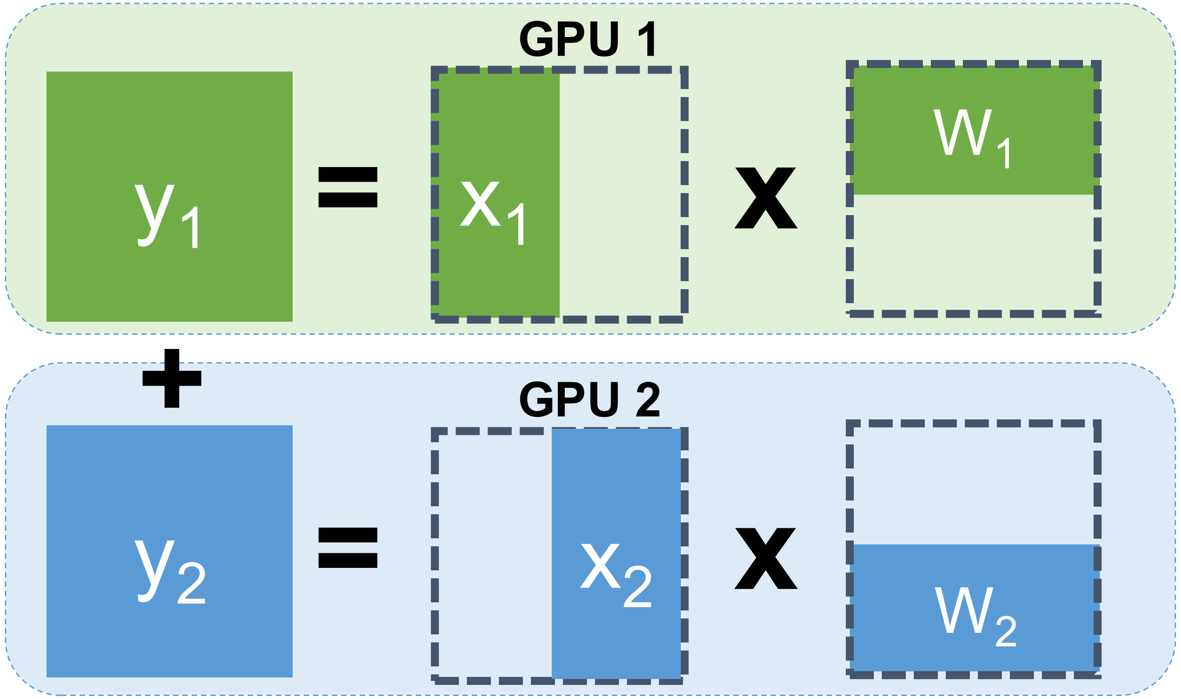
Tensor Model Parallelism (partition output)

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

Communication Cost of Tensor Model Parallelism

Comparing Data and Tensor Model Parallelism





Tensor Model Parallelism (Reduce output)

$$y = y1 + y2$$

Forward Processing	Backward Propagation	Gradients Sync
$O(B * C_{out})$	0	0

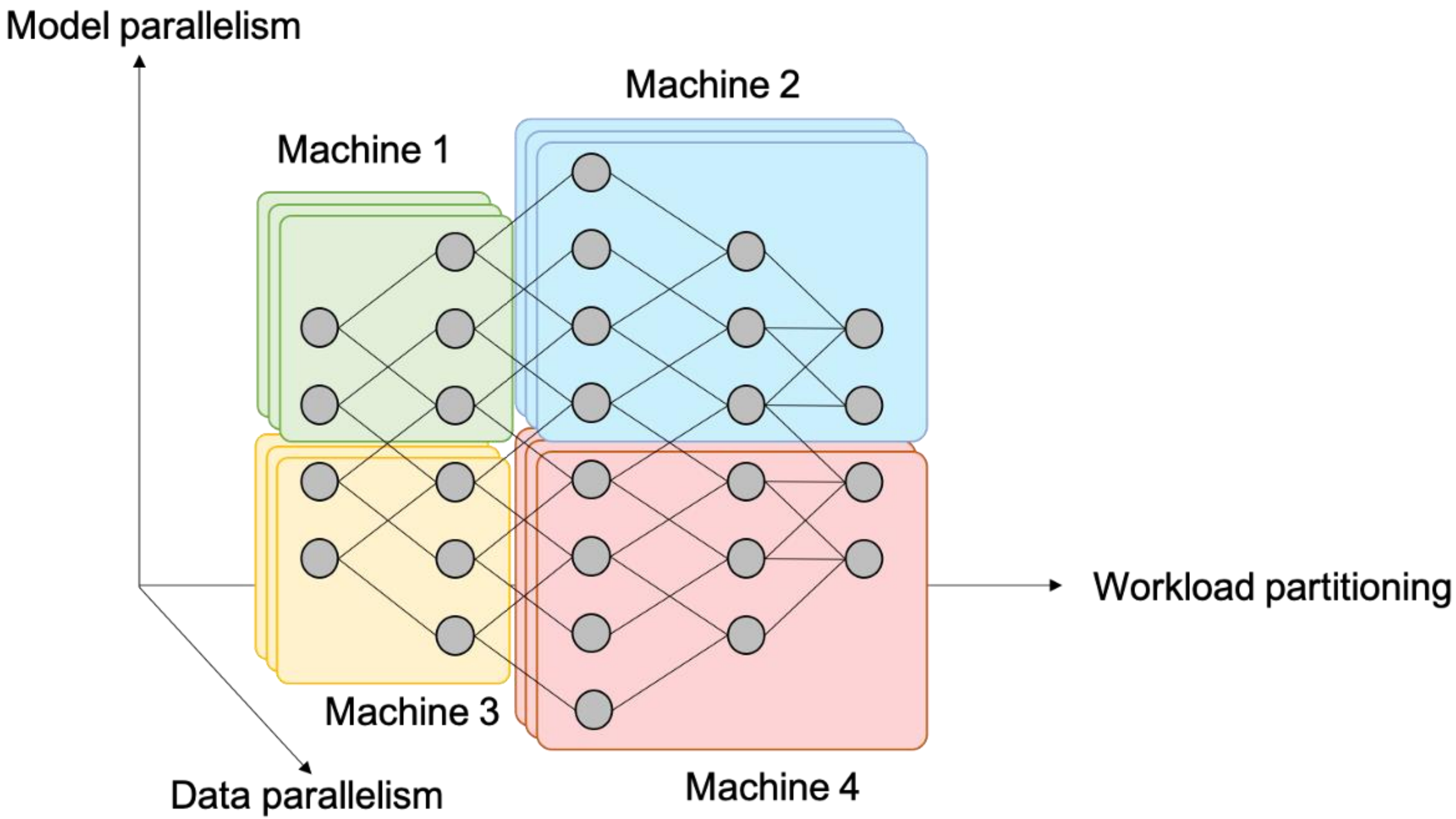
Communication Cost of Tensor Model Parallelism

Comparing Data and Tensor Model Parallelism

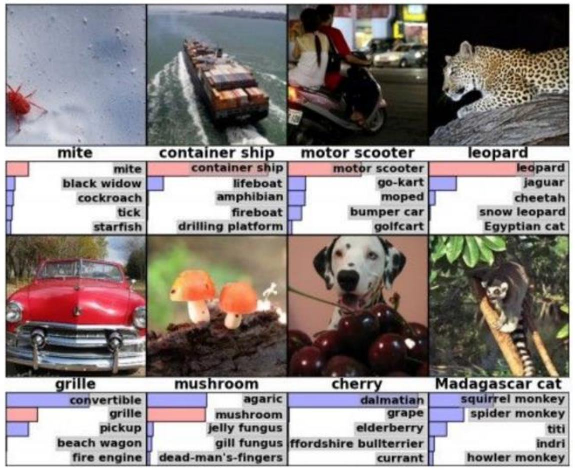
- Data parallelism: $O(C_{out} * C_{in})$
- Tensor model parallelism (partition output): $O(B * C_{in})$
- Tensor model parallelism (reduce output): $O(B * C_{out})$

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

Combine Data and Model Parallelism

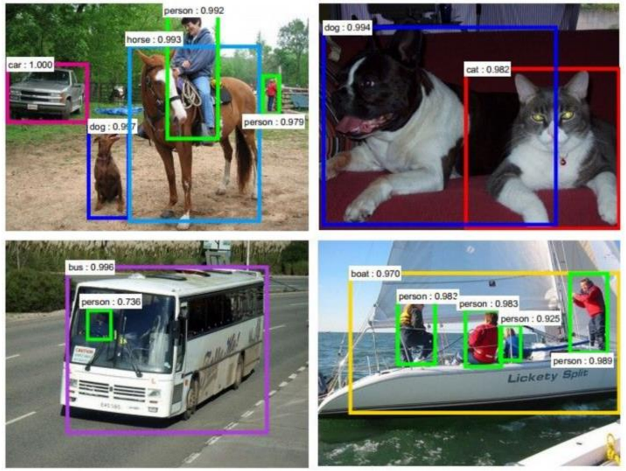


Example: Convolutional Neural Networks

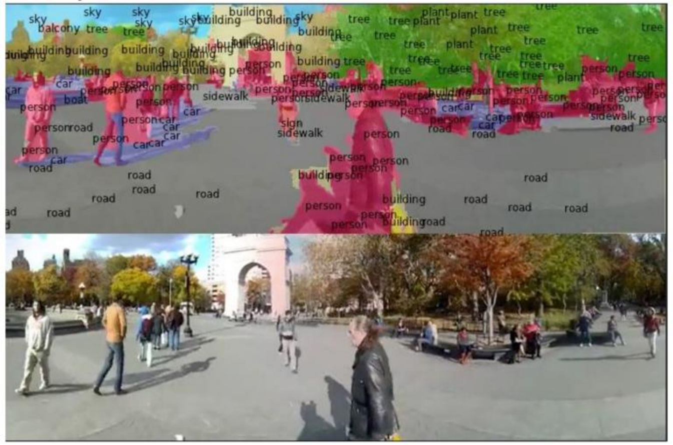


Classification

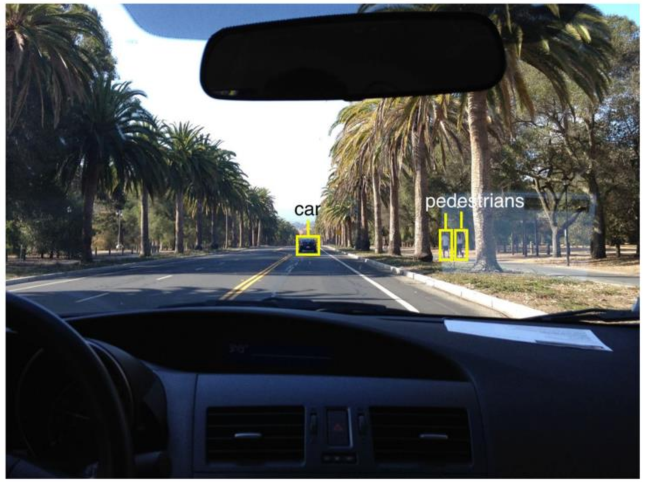
Retrieval



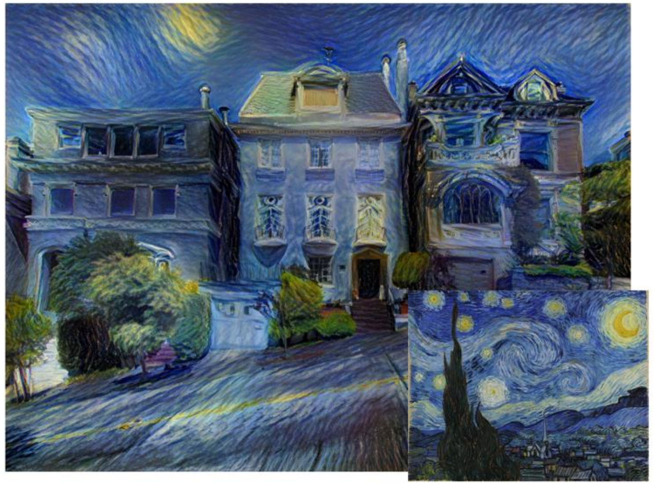
Detection



Segmentation



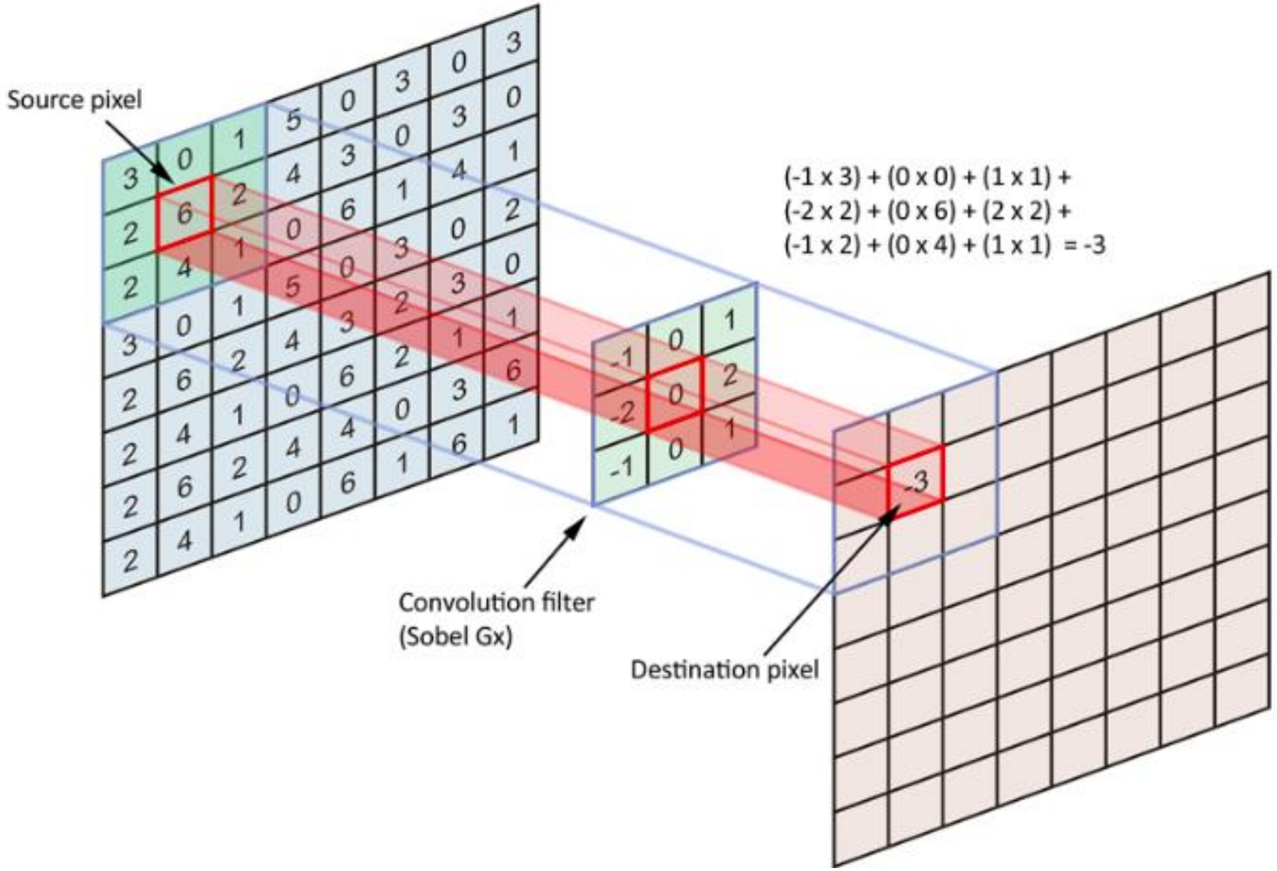
Self-Driving



Synthesis

Convolution

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



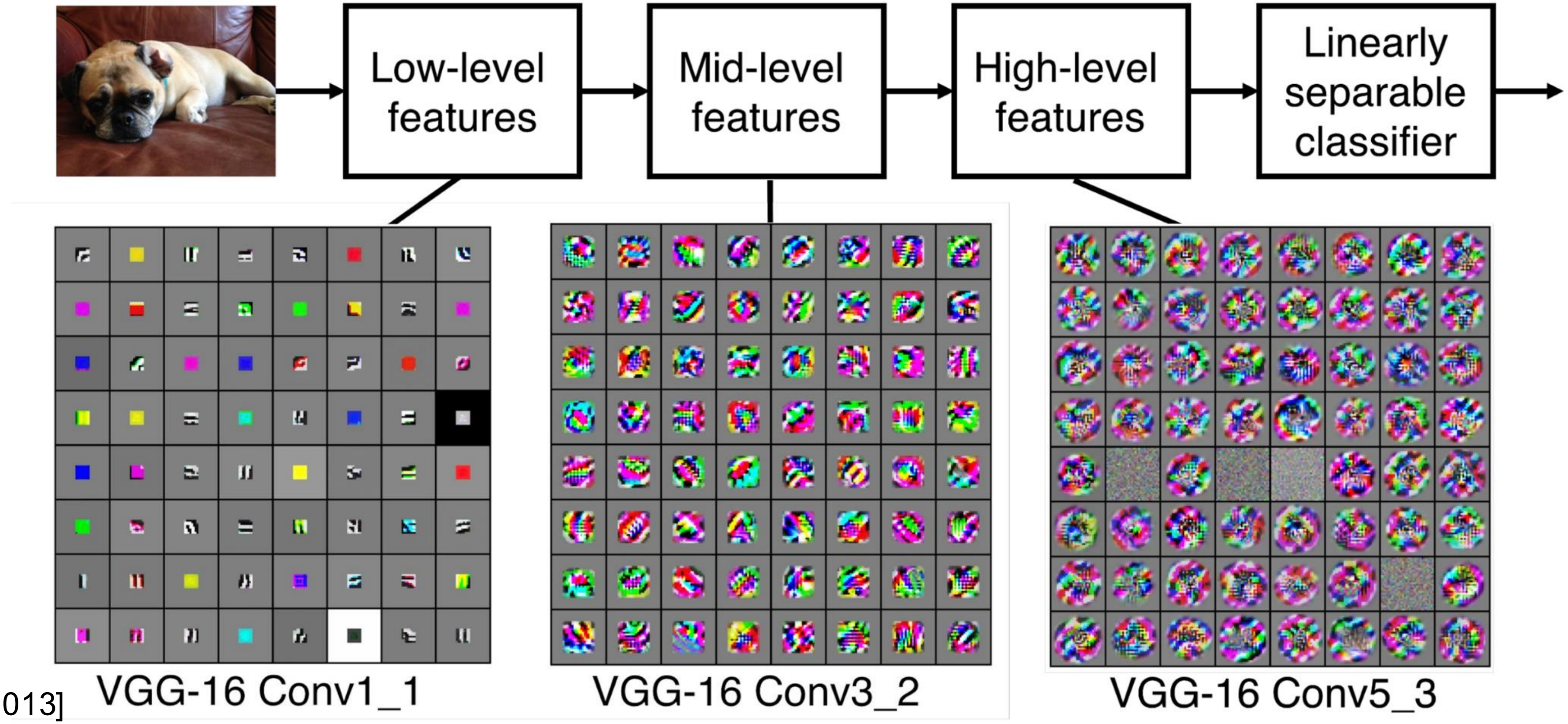
3	0	1	5	0	3	0	3
2	6	2	4	3	0	3	0
2	4	1	0	6	1	4	1
3	0	1	5	0	3	0	2
2	6	2	4	3	2	3	0
2	4	1	0	6	2	1	1
2	6	2	4	4	0	3	6
2	4	1	0	6	1	6	1

-1	0	1
-2	0	2
-1	0	1

(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3

CNNs

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



Low-level features

Mid-level features

High-level features

Linearly separable classifier

VGG-16 Conv1_1

VGG-16 Conv3_2

VGG-16 Conv5_3

[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

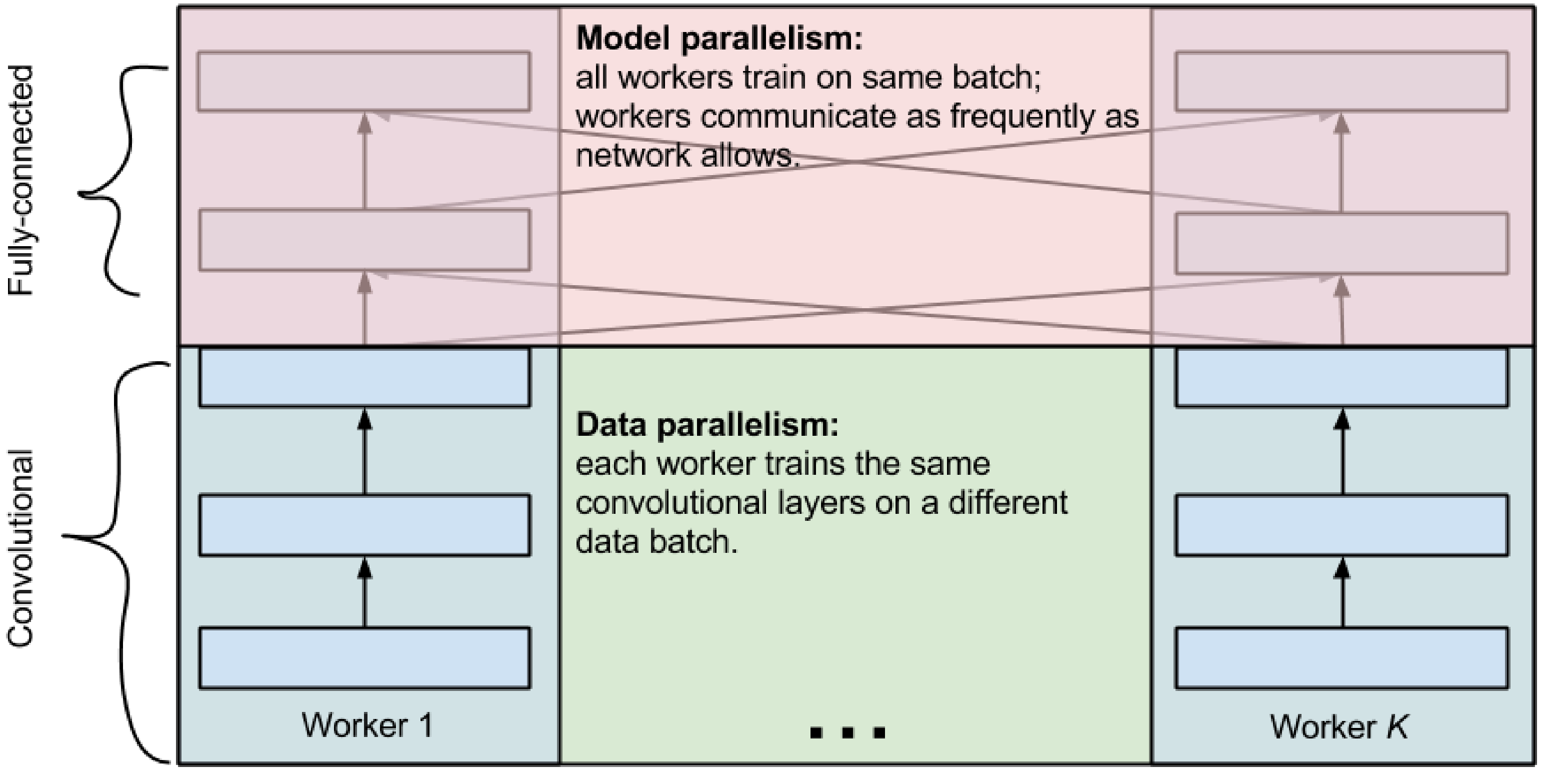
- Fully-connected layers
  - 5-10% of the computation
  - 95% of the parameters
  - Small intermediate activations

Tensor model parallelism

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

Parallelizing Convolutional Neural Networks

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

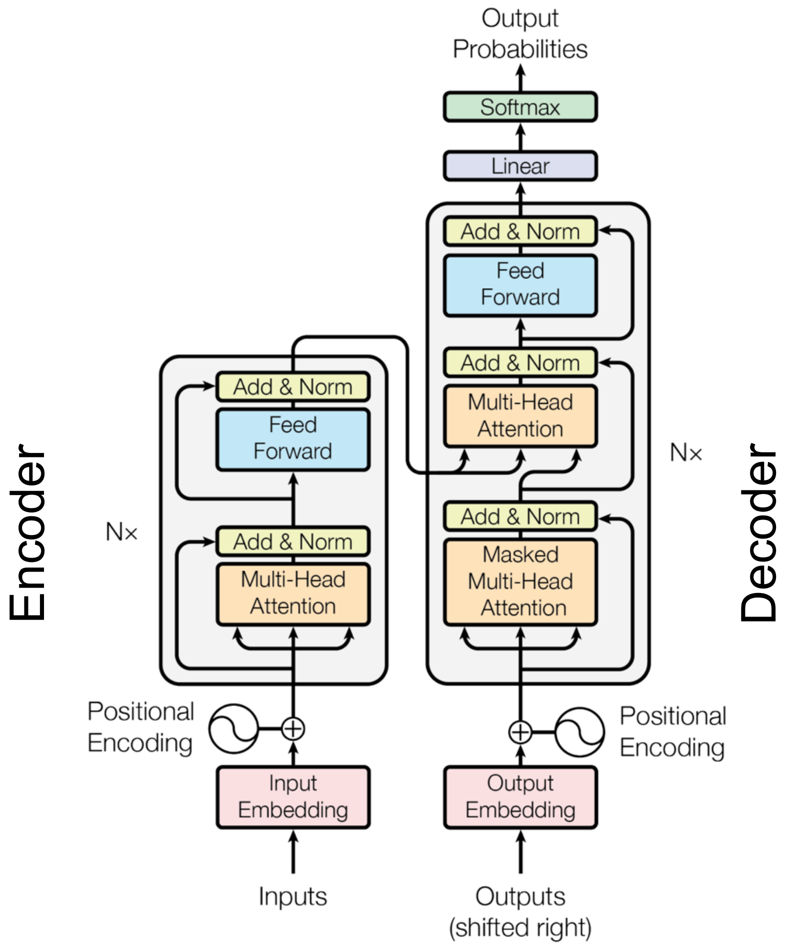


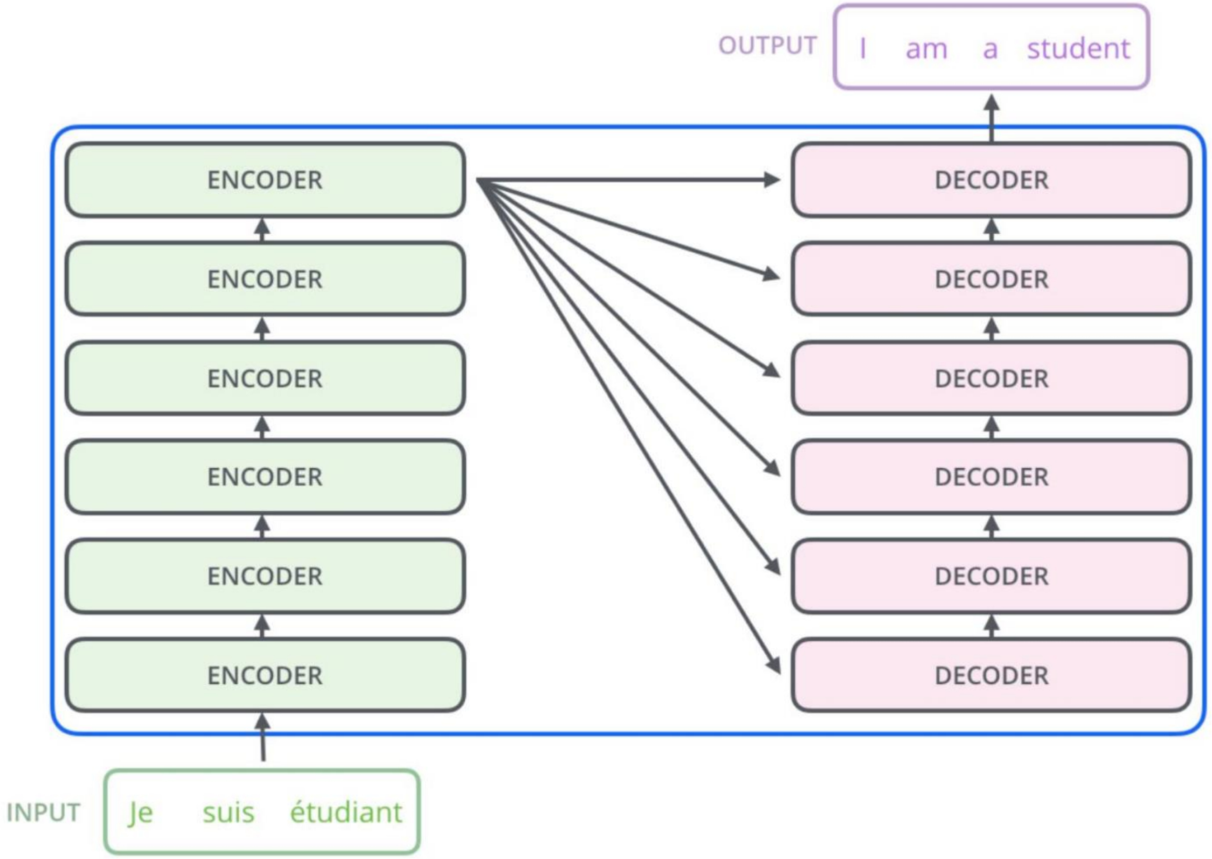
Model parallelism:
all workers train on same batch;
workers communicate as frequently as
network allows.

Data parallelism:
each worker trains the same
convolutional layers on a different
data batch.

Example: Parallelizing Transformers

- Transformer: attention mechanism for language understanding



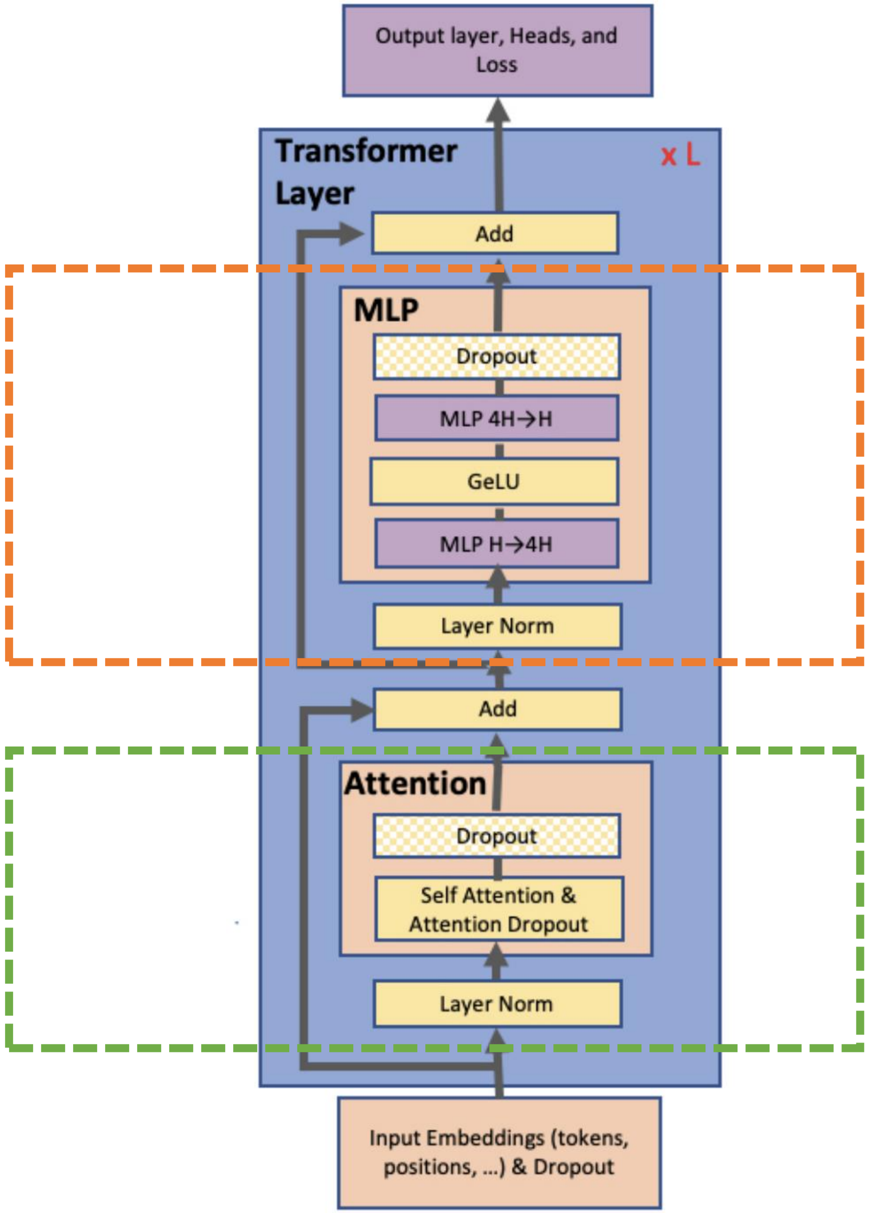


Ashish Vaswani et. al. Attention is all you need.

A Single Transformer Layer

Fully-Connected Layers

Self-Attention Layers

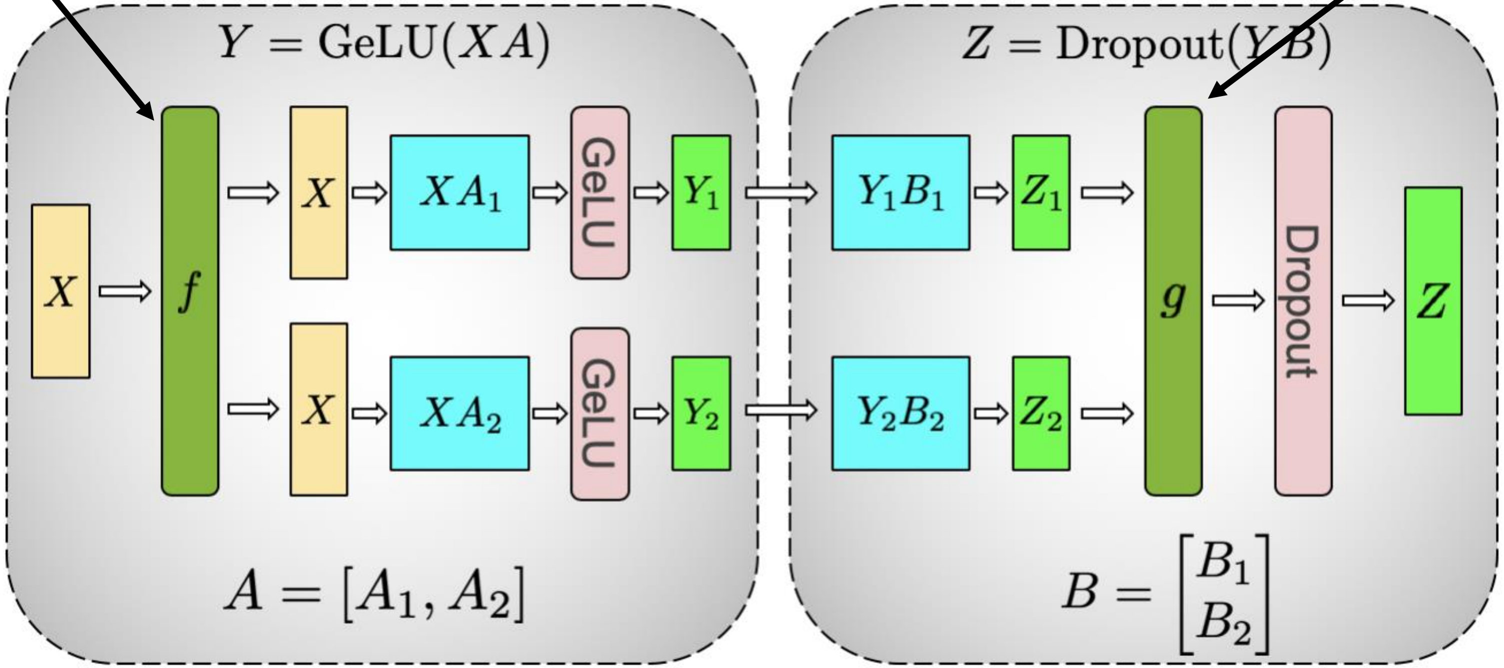

Output layer, Heads, and Loss
Transformer Layer
x L
Add
MLP
Dropout
MLP 4H→H
GeLU
MLP H→4H
Layer Norm
Add
Attention
Dropout
Self Attention & Attention Dropout
Layer Norm
Input Embeddings (tokens, positions, ...) & Dropout

Parallelizing Fully-Connected Layers in Transformers

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

identity layer

reduction layer

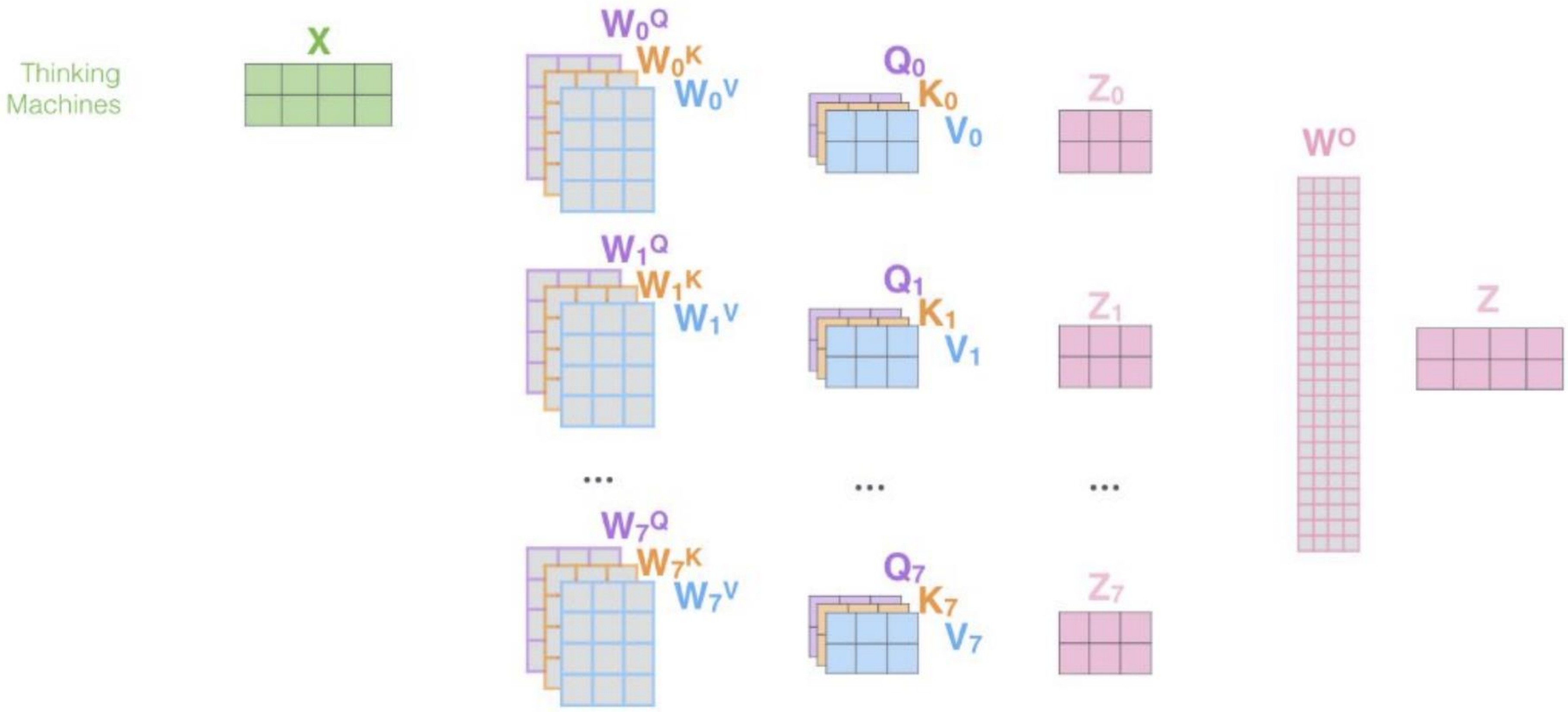


Tensor model parallelism
(partition output)

Tensor model parallelism
(reduce output)

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

Multi-Head Self-Attention



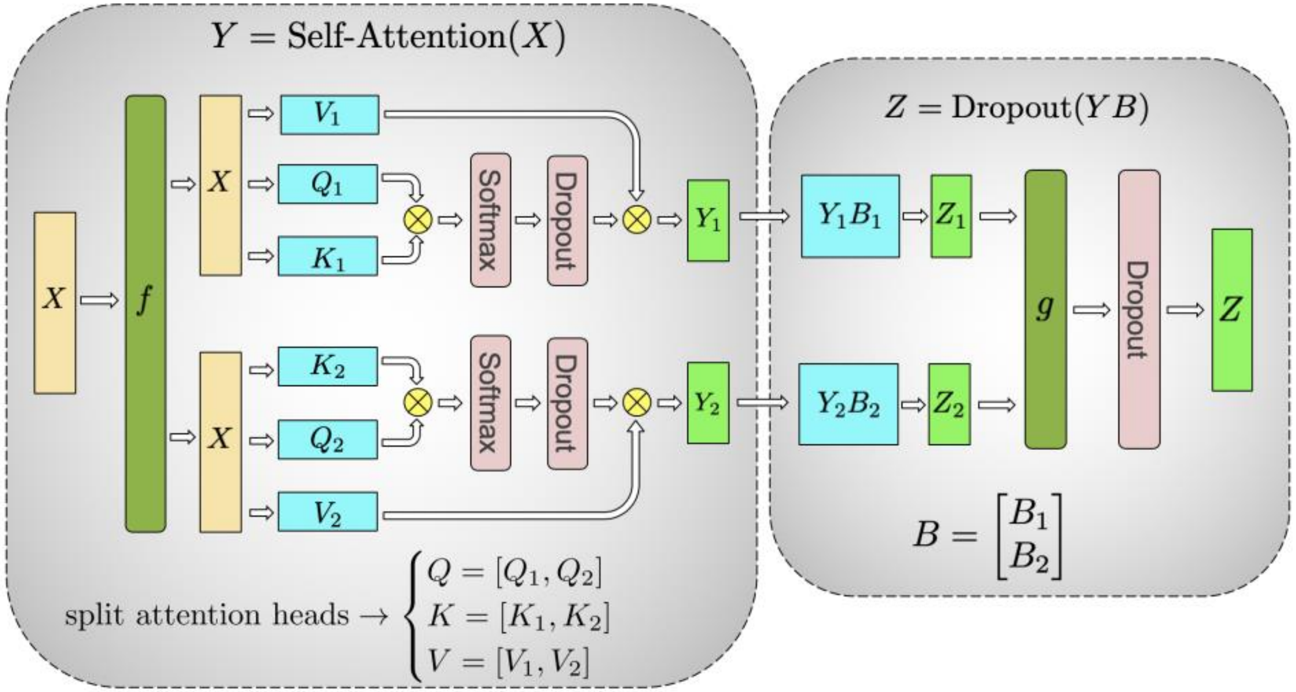
$$Z_i = A(Q_i, K_i, V_i) = softmax\left(\frac{Q_i {K_i}^T}{\sqrt{d}}\right) V_i$$

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

Parallelizing Self-Attention Layers in Transformers

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

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

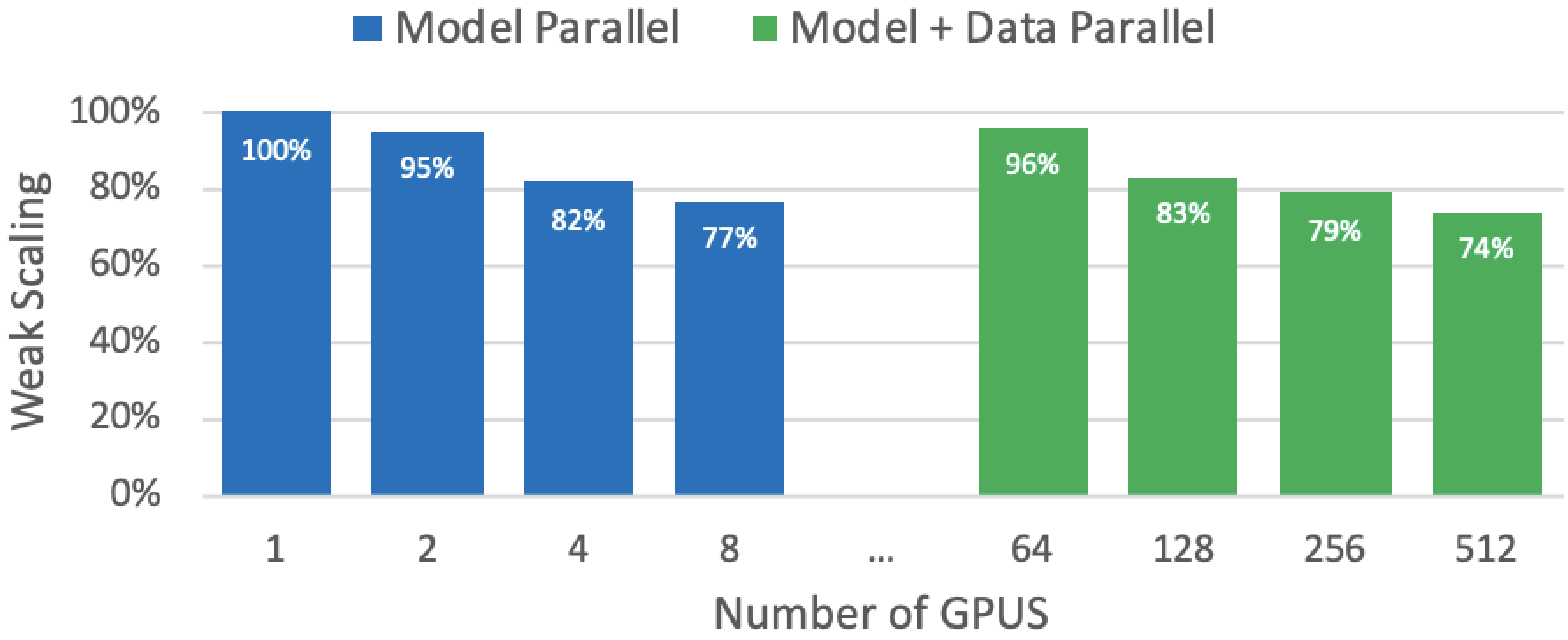


Parallelizing across attention heads

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

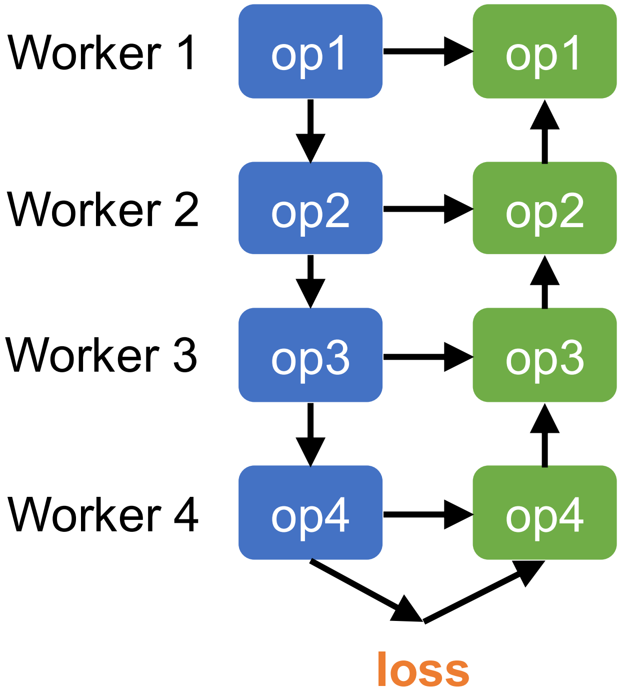
Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism.

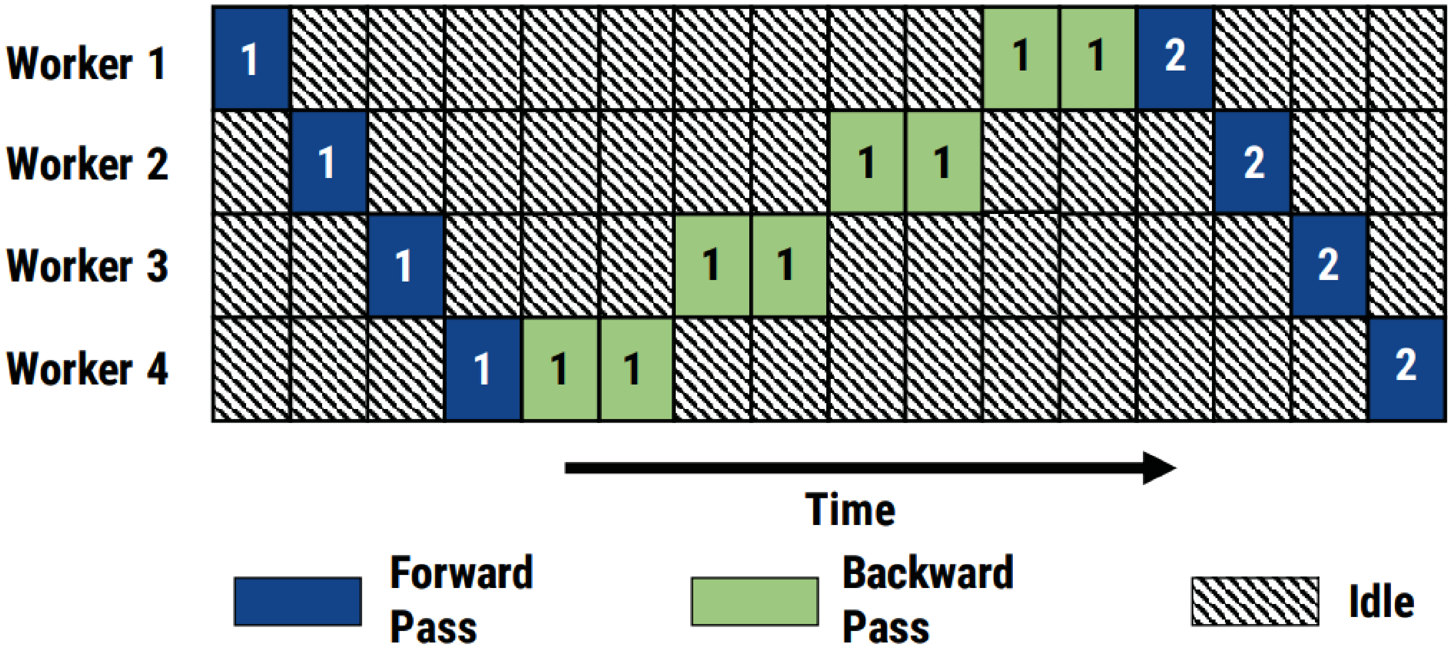
How to parallelize DNN Training?

- 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





Worker 1	🟦 1	▨	▨	▨	▨	▨	▨	▨	▨	▨	🟩 1	🟩 1	🟦 2	▨	▨	▨
Worker 2	▨	🟦 1	▨	▨	▨	▨	▨	▨	🟩 1	🟩 1	▨	▨	▨	🟦 2	▨	▨
Worker 3	▨	▨	🟦 1	▨	▨	🟩 1	🟩 1	▨	▨	▨	▨	▨	▨	▨	🟦 2	▨
Worker 4	▨	▨	▨	🟦 1	🟩 1	🟩 1	▨	▨	▨	▨	▨	▨	▨	▨	▨	🟦 2

🟦 Forward Pass 🟩 Backward Pass ▨ Idle

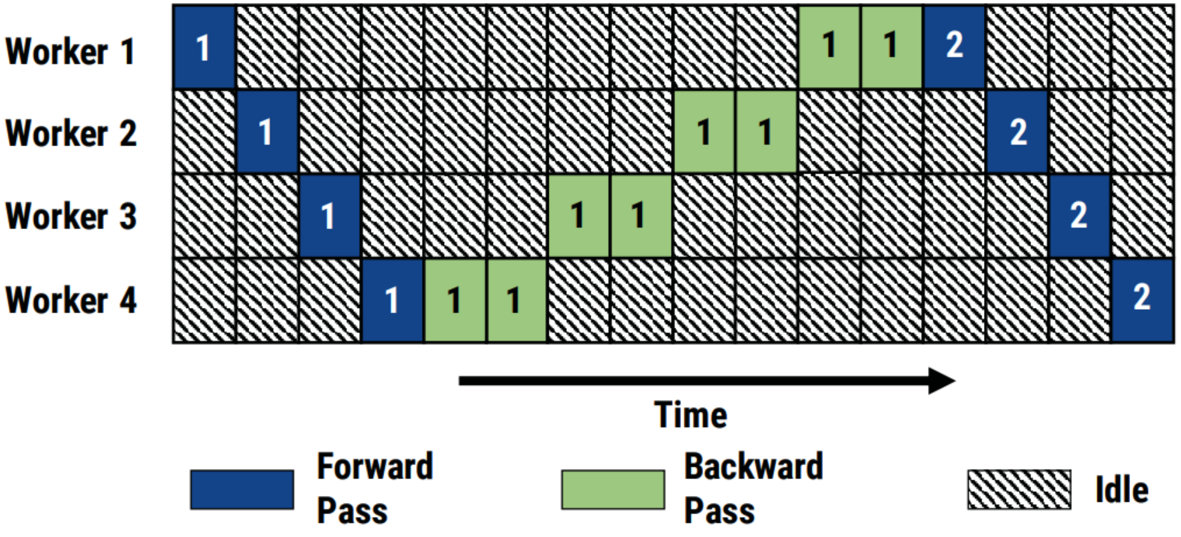
Pipeline Model Parallelism

- **Mini-batch**: the number of samples processed in each iteration

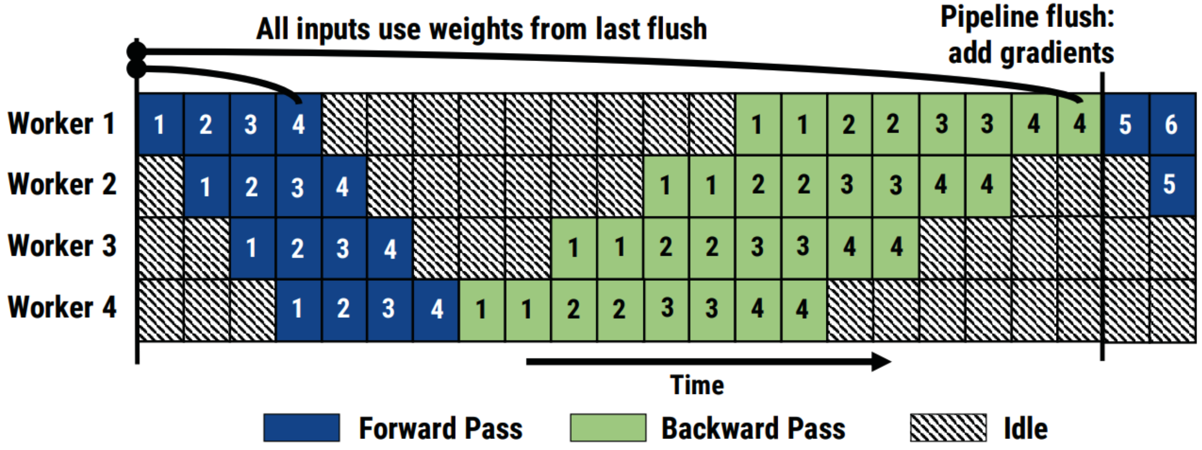
- Divide a mini-batch into multiple **micro-batches**

- Pipeline the forward and backward computations across micro-batches

Model Parallelism

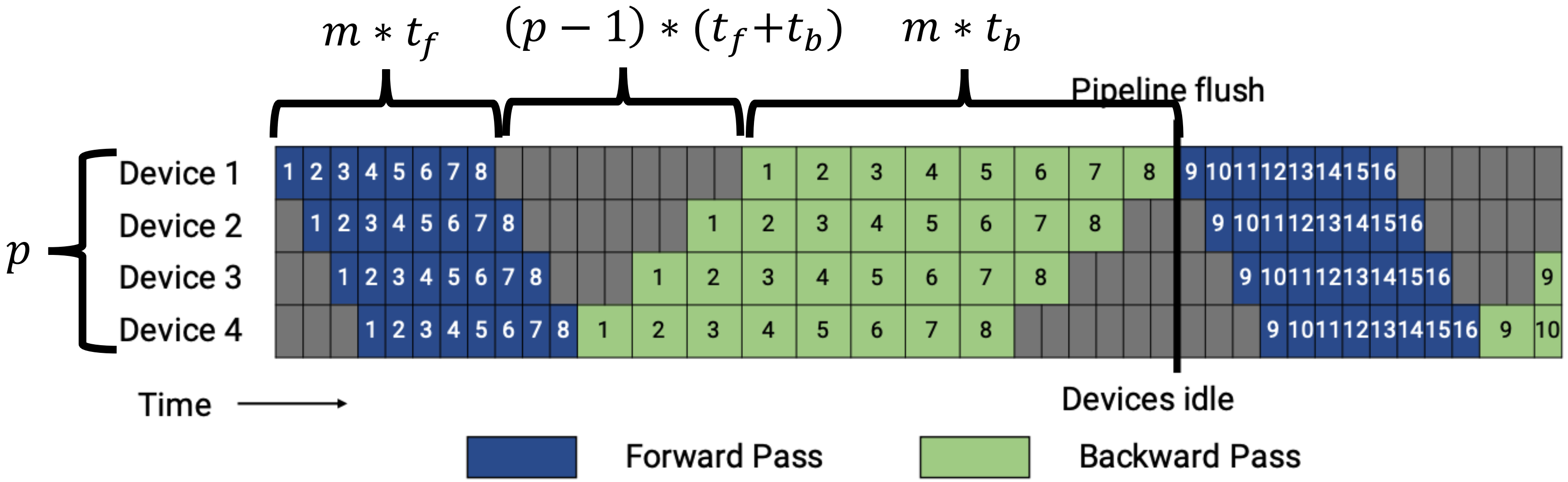


Pipeline Model Parallelism



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

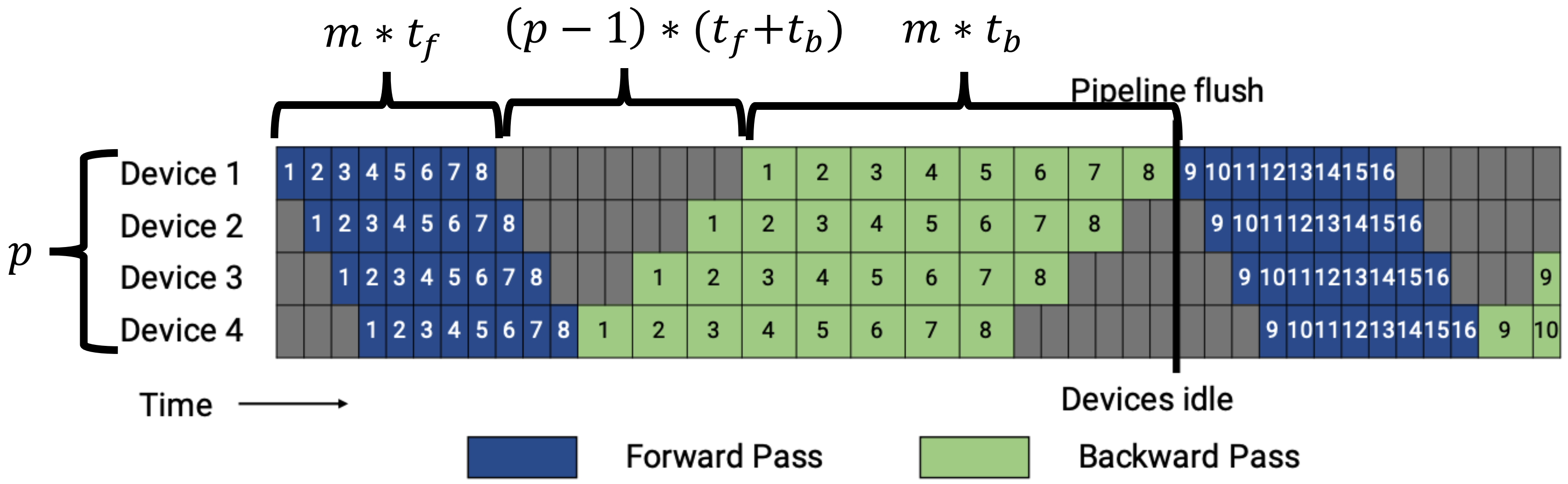


$$\boldsymbol{BubbleFraction = \frac{(p-1)*(t_f+t_b)}{m*t_f\ + m*t_b} = \frac{p-1}{m}}$$

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism

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

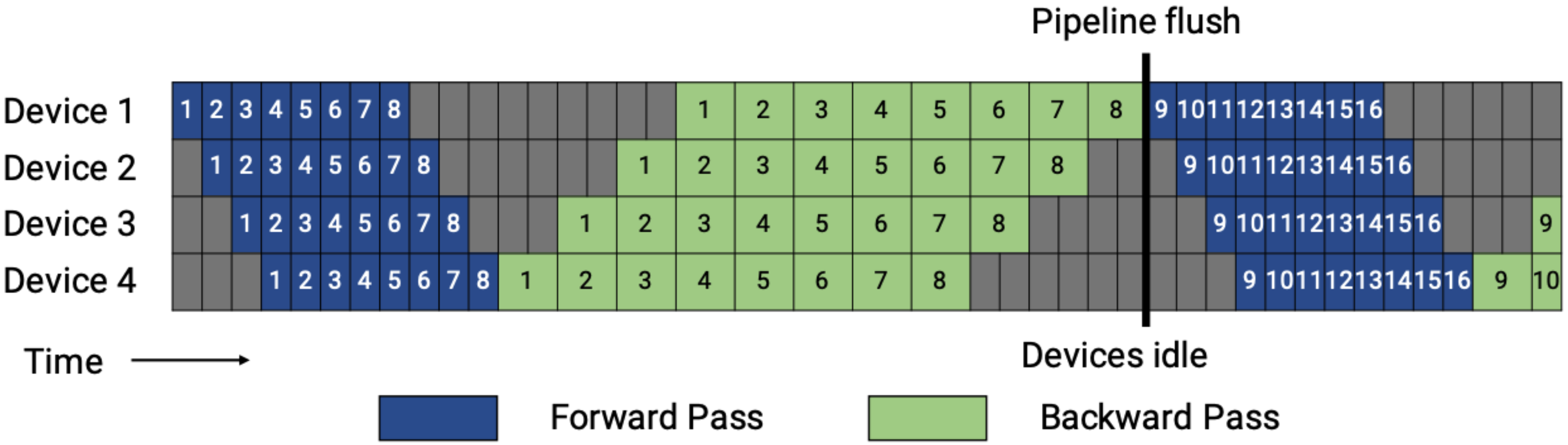


$$\boldsymbol{BubbleFraction} = \frac{\boldsymbol{(p-1) * (t_f + t_b)}}{\boldsymbol{m * t_f + m * t_b}} = \frac{\boldsymbol{p-1}}{\boldsymbol{m}}$$

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism

Pipeline Model Parallelism: Memory Requirement

- An issue: we need to keep the intermediate activations of **all micro-batches** before back propagation



Can we improve the pipeline schedule to reduce memory requirement?

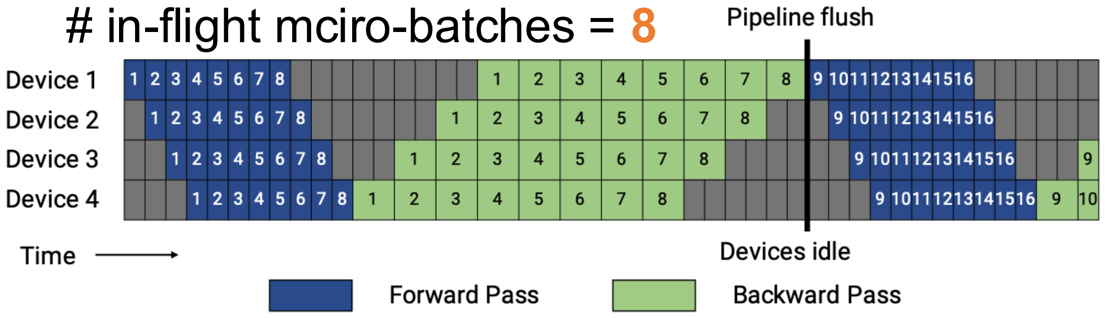
GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism

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**

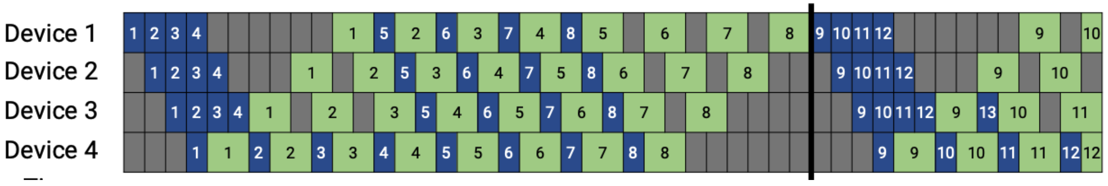
Can we reduce pipeline bubble?

# in-flight mciro-batches = 8



Pipeline parallelism with GPipe's schedule

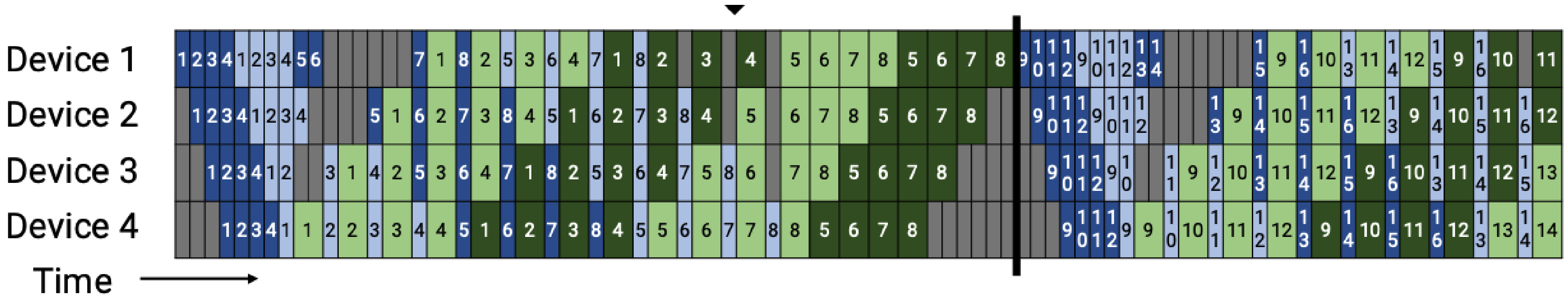
# in-flight mciro-batches = 4



Pipeline parallelism with 1F1B schedule

Pipeline Parallelism with Interleaved 1F1B Schedule

- Further divide each stage 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.

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

Reduce bubble time at the cost increased communication

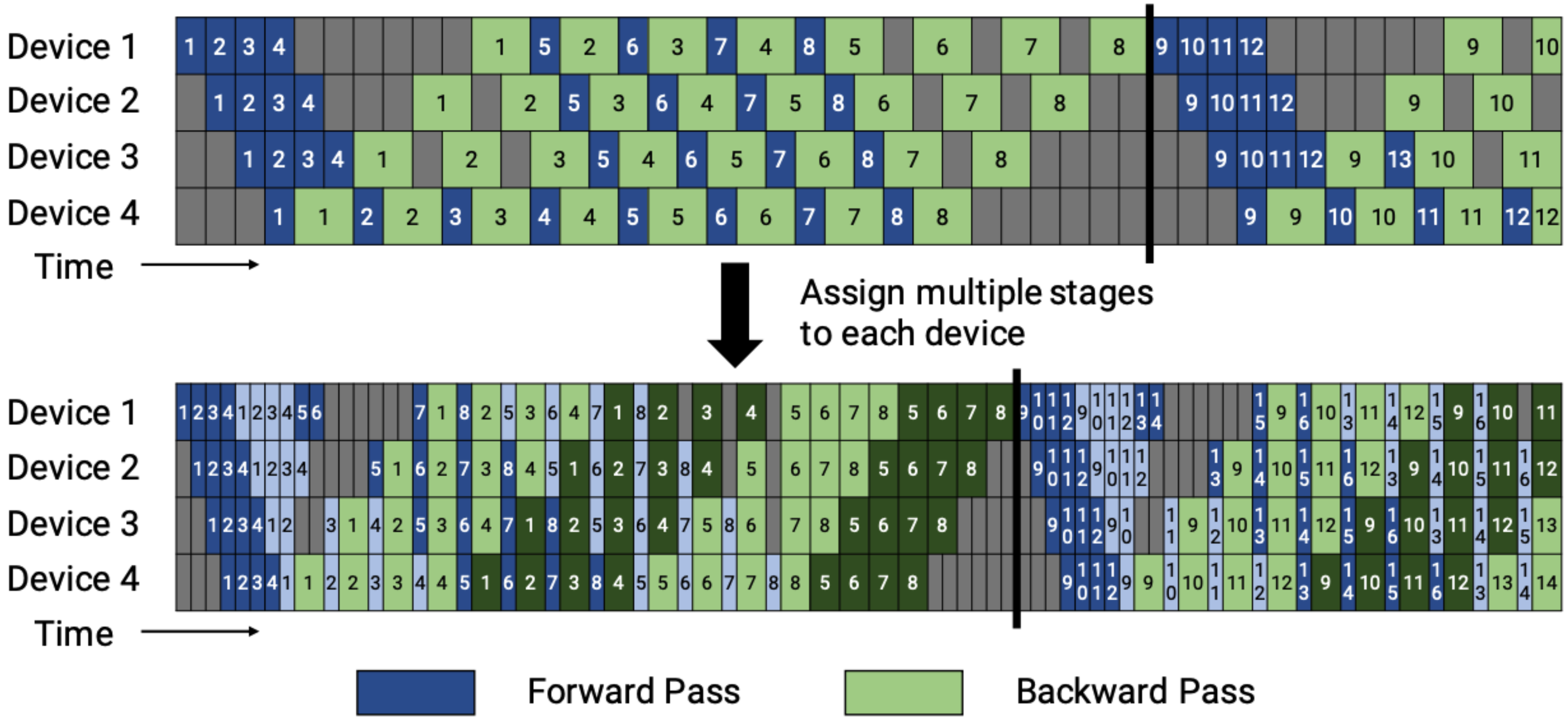
Pipeline Parallelism with Interleaved 1F1B Schedule

Pipeline parallelism with 1F1B Schedule

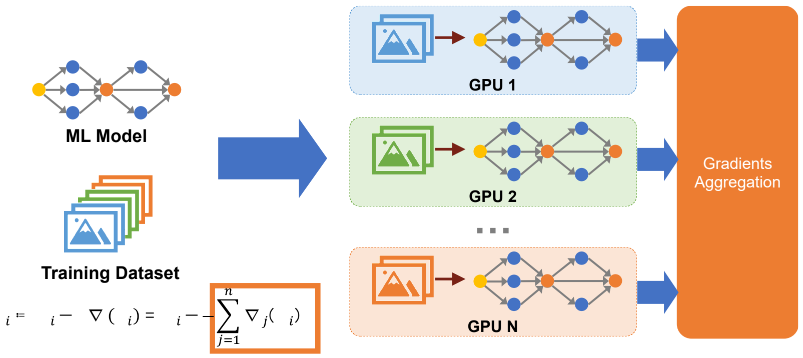
$$BubbleFraction = \frac{p-1}{m}$$

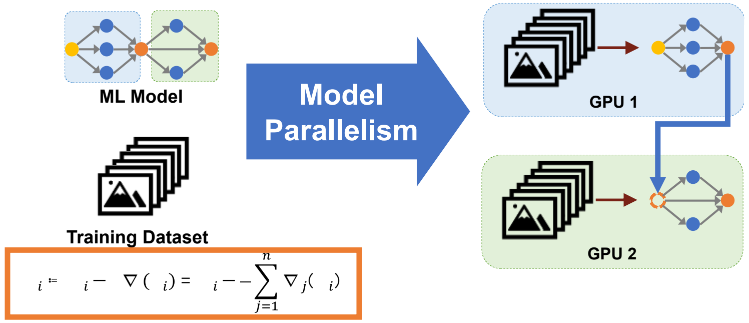
Pipeline parallelism with interleaved 1F1B Schedule

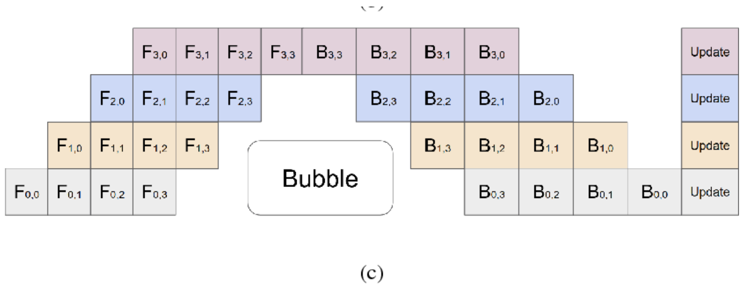
$$BubbleFraction = \frac{1}{v} * \frac{p-1}{m}$$



Summary: Comparing Data/Tensor Model/Pipeline Model Parallelism

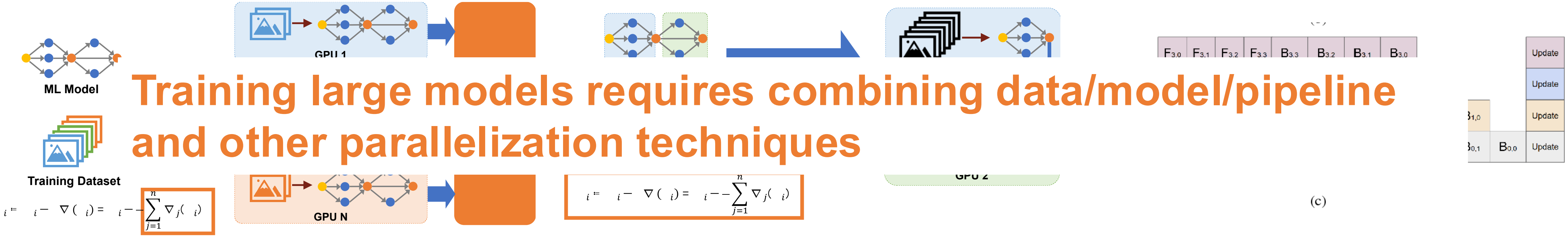

ML Model
Training Dataset
GPU 1
GPU 2
. . .
GPU N
Gradients Aggregation
$_i = \; _i - \nabla(\; _i) = \; _i - -\sum_{j=1}^{n} \nabla_j(\; _i)$


ML Model
Training Dataset
Model Parallelism
GPU 1
GPU 2
$_i = \; _i - \nabla(\; _i) = \; _i - -\sum_{j=1}^{n} \nabla_j(\; _i)$


$F_{3,0}$ $F_{3,1}$ $F_{3,2}$ $F_{3,3}$ $B_{3,3}$ $B_{3,2}$ $B_{3,1}$ $B_{3,0}$ Update
$F_{2,0}$ $F_{2,1}$ $F_{2,2}$ $F_{2,3}$ $B_{2,3}$ $B_{2,2}$ $B_{2,1}$ $B_{2,0}$ Update
$F_{1,0}$ $F_{1,1}$ $F_{1,2}$ $F_{1,3}$ Bubble $B_{1,3}$ $B_{1,2}$ $B_{1,1}$ $B_{1,0}$ Update
$F_{0,0}$ $F_{0,1}$ $F_{0,2}$ $F_{0,3}$ $B_{0,3}$ $B_{0,2}$ $B_{0,1}$ $B_{0,0}$ Update
(c)

	Data Parallelism	Tensor Model Parallelism	Pipeline Model 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

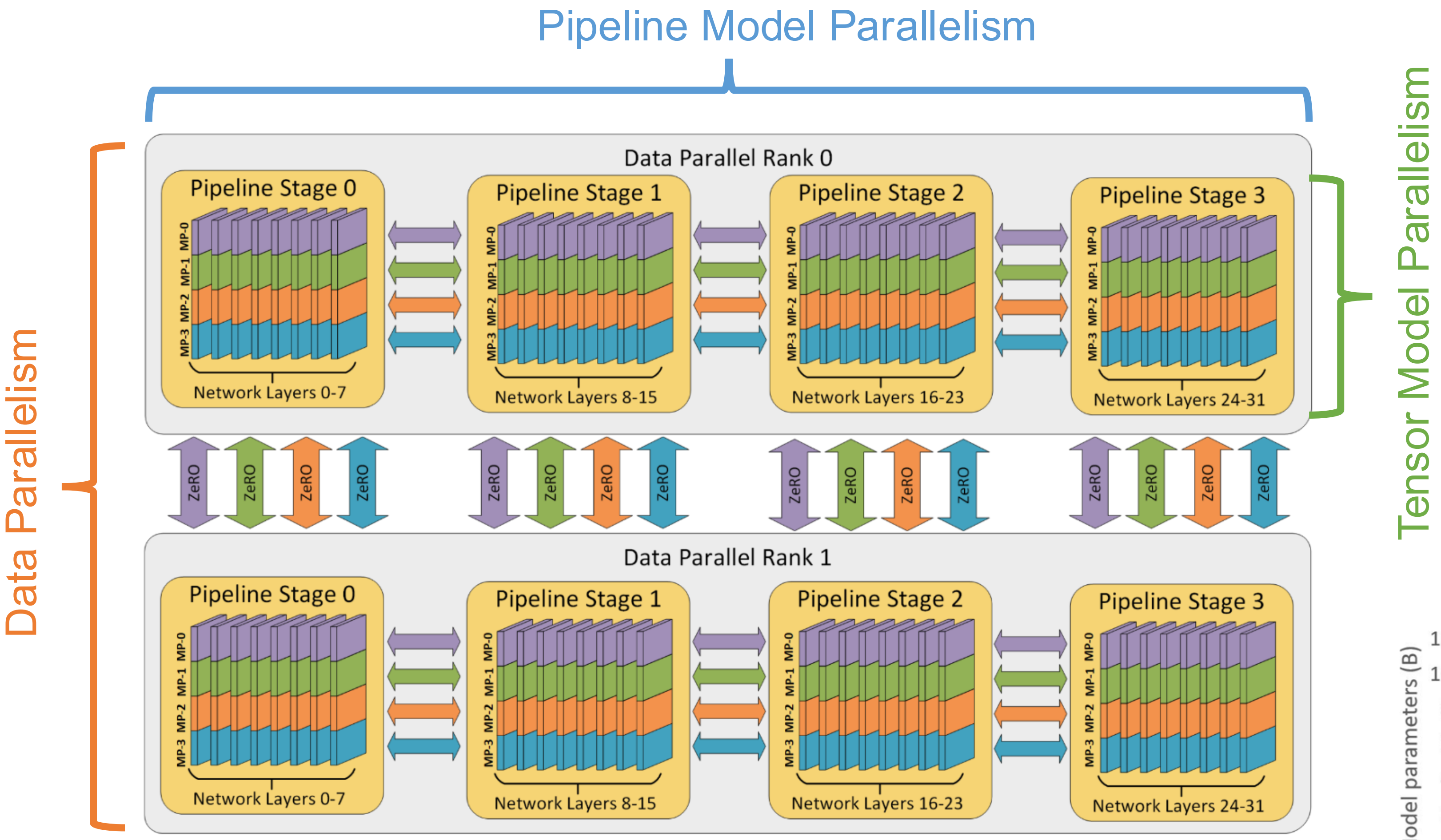
Summary: Comparing Data/Tensor Model/Pipeline Model Parallelism



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

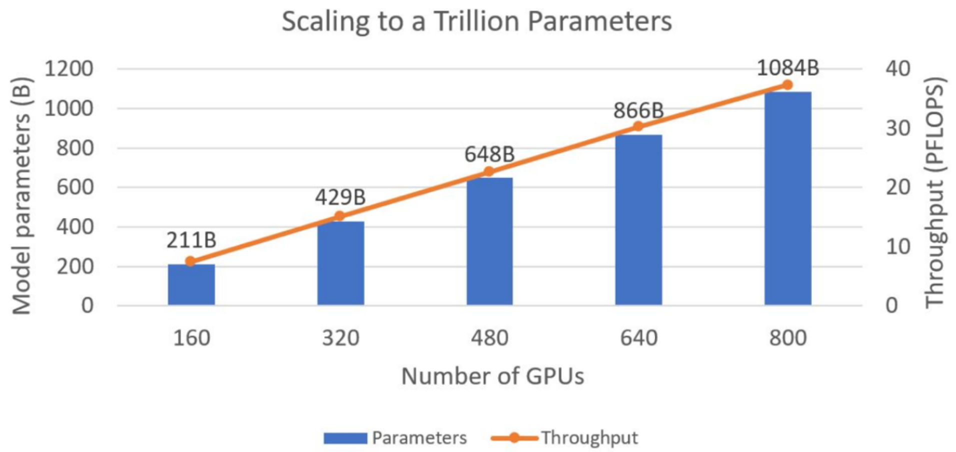
Example: 3D parallelism in DeepSpeed



Pipeline Model Parallelism

Data Parallelism

Tensor Model Parallelism



Scaling to a Trillion Parameters

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