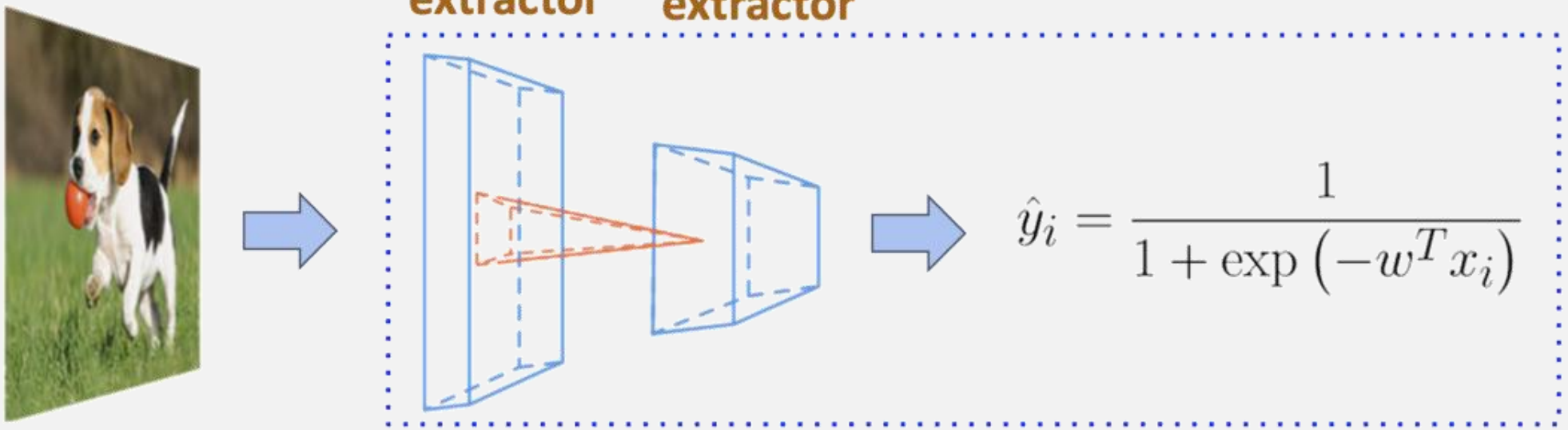
机器学习系统

第六章 数据并行与模型并行

张燕咏 讲席教授 yyz@ustc.edu.cn
张午阳 特任教授 wyz@ustc.edu.cn

中国科学技术大学
University of Science and Technology of China

回顾：DNN（深度神经网络）训练概览



目标

$$L(w) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \lambda \|w\|^2$$

训练

$$w \leftarrow w - \eta \nabla_w L(w)$$

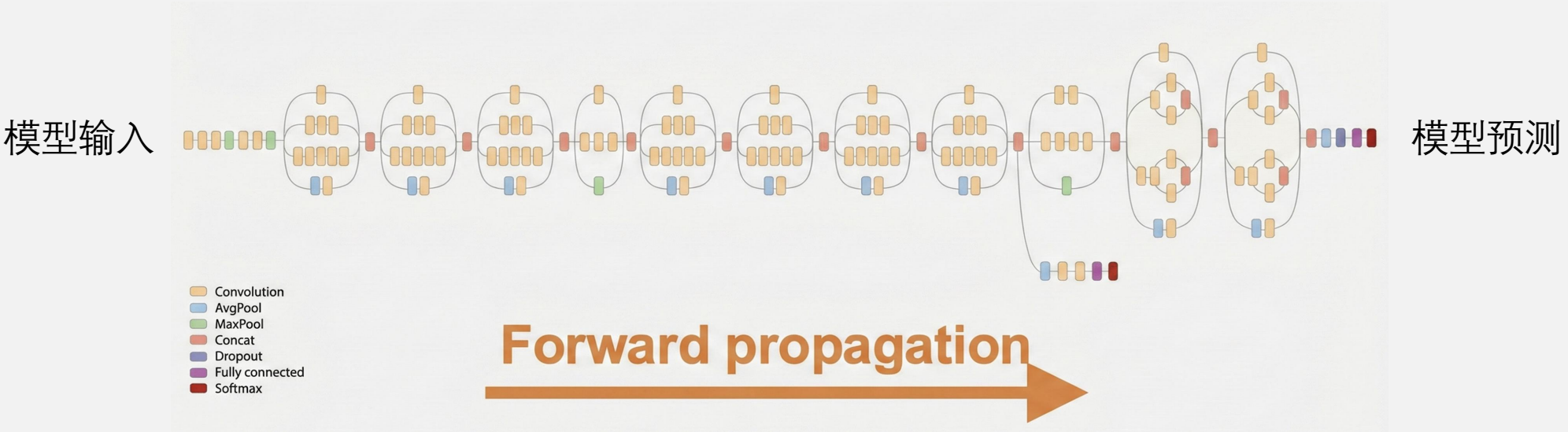
DNN 训练过程

通过多次迭代以下3个阶段来训练机器学习模型：

1.前向传播 ： 将模型应用于一批输入样本，并通过算子运行计算以产生预测结果

2.反向传播 ： 反向运行模型以生成每个可训练权重的误差

3.权重更新 ： 使用损失值来更新模型权重



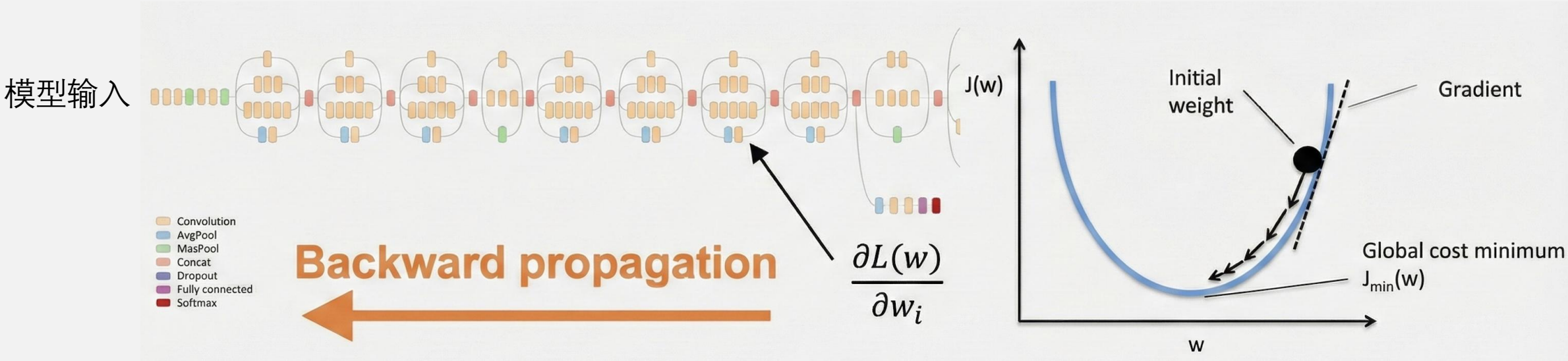
DNN 训练过程

通过多次迭代以下3个阶段来训练机器学习模型：

1.前向传播 ： 将模型应用于一批输入样本，并通过算子运行计算以产生预测结果

2.反向传播 ： 反向运行模型以生成每个可训练权重的误差

3.权重更新 ： 使用损失值来更新模型权重



DNN 训练过程

通过多次迭代以下3个阶段来训练机器学习模型：

1.前向传播 ： 将模型应用于一批输入样本，并通过算子运行计算以产生预测结果

2.反向传播 ： 反向运行模型以生成每个可训练权重的误差

3.权重更新 ： 使用损失值来更新模型权重

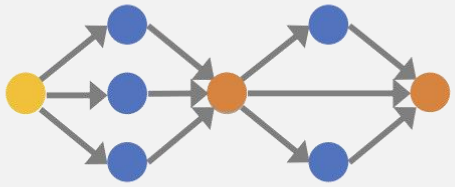
$$w_i := w_i - \gamma \frac{\partial L(w)}{\partial w_i} = w_i - \frac{\gamma}{n} \sum_{j=1}^{n} \frac{\partial l_i(w)}{\partial w_i}$$

个体样本梯度

如何实现深度神经网络训练的并行化？

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

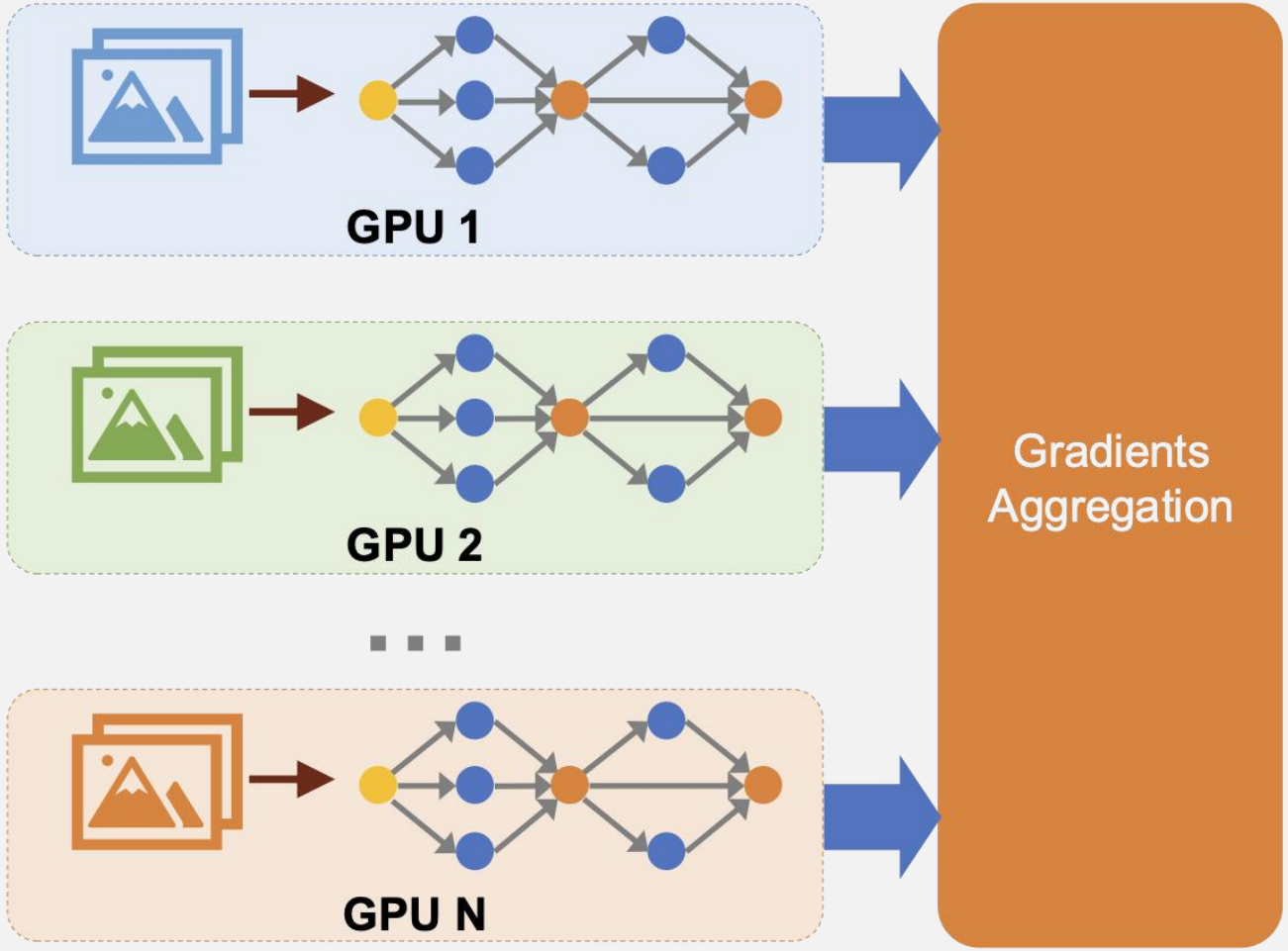
数据并行性



ML Model

Training Dataset

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

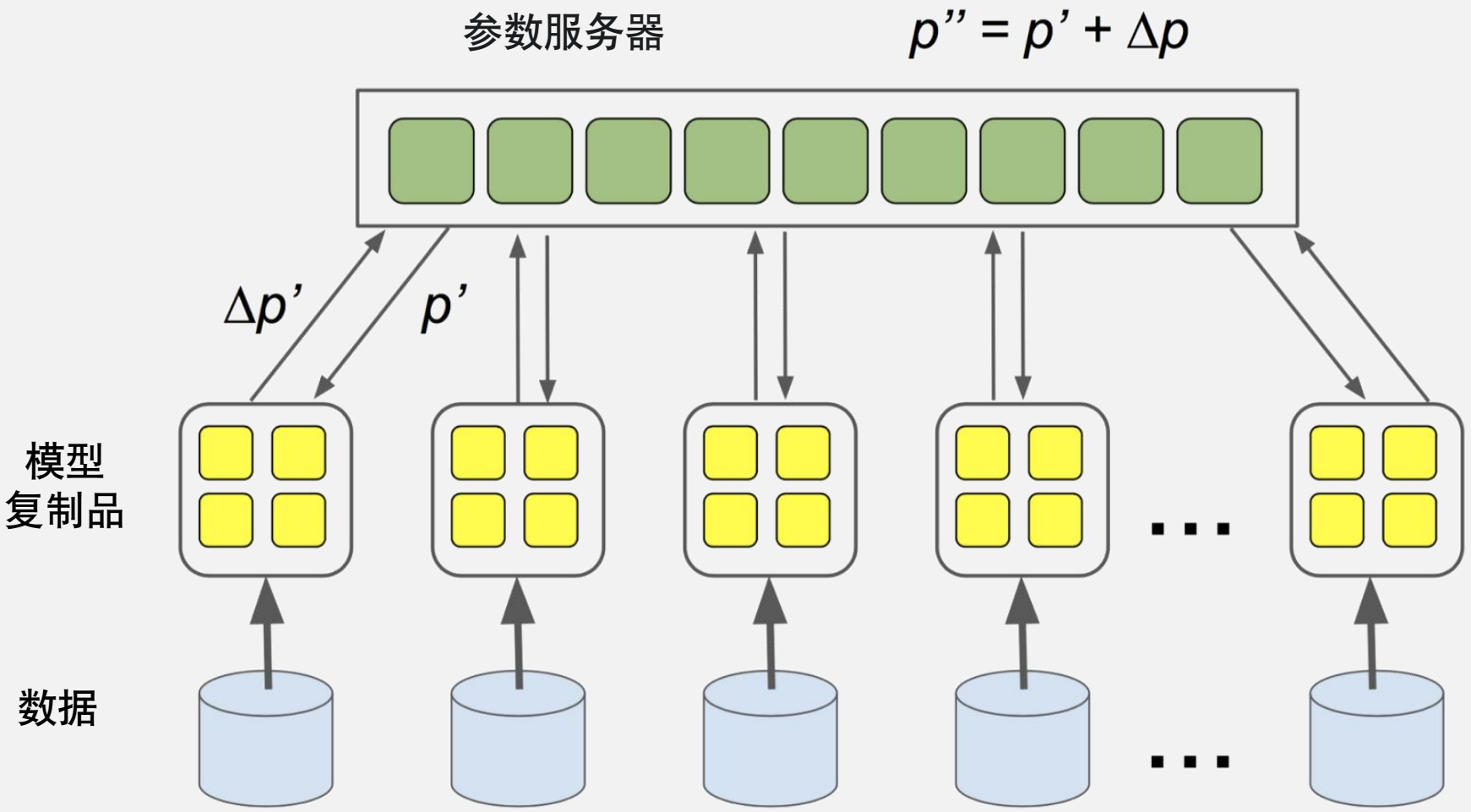


1.将训练数据划分为批次

2.在GPU上计算每批次的梯度

3.跨GPU聚合梯度

数据并行性：参数服务器

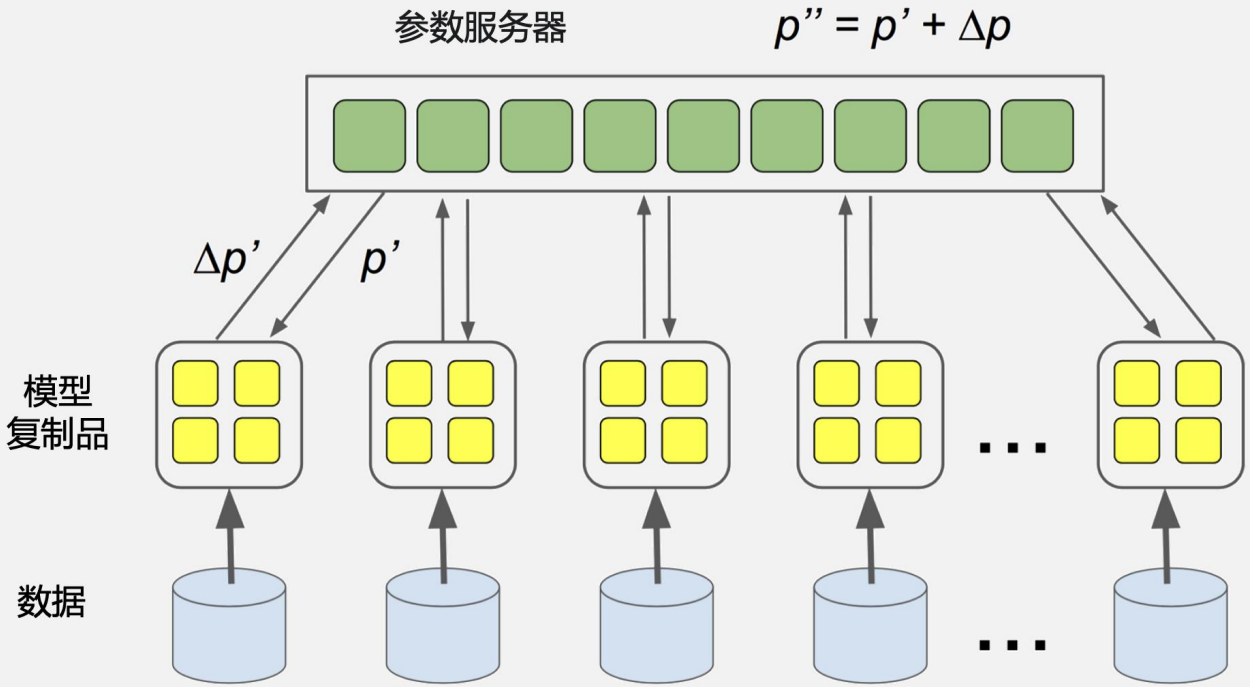


工作器将梯度推送到参数服务器，并拉取更新后的参数回来

参数服务器的低效性

•集中式通信：所有工作者均通过参数服务器进行权重更新；无法扩展至大量工作者

•如何在DNN训练中实现通信去中心化?

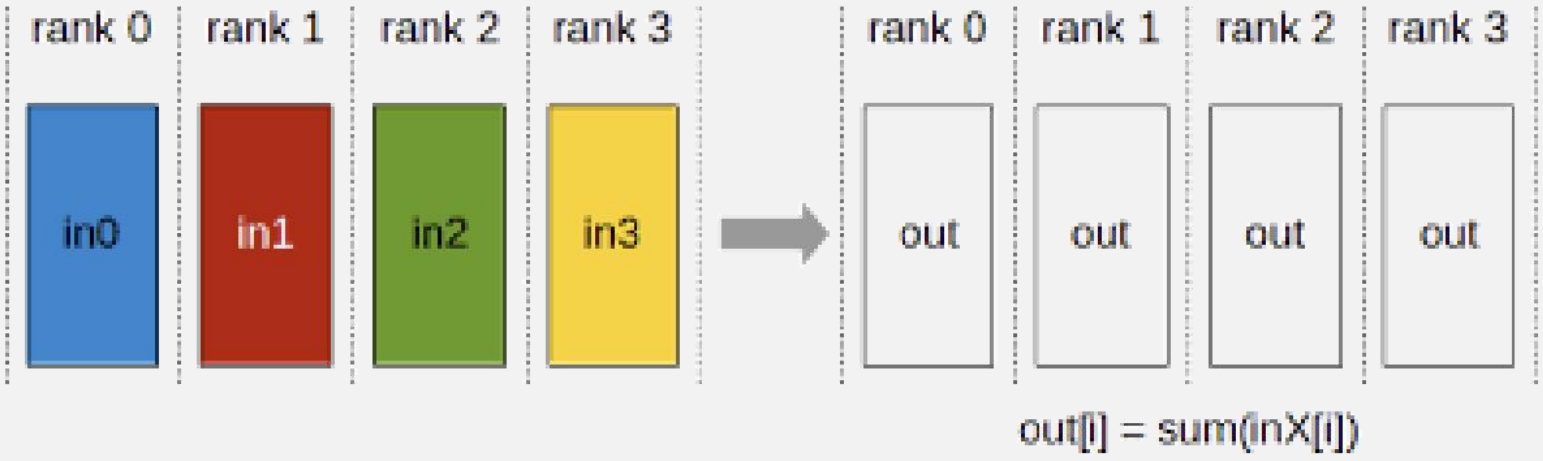


参数服务器的低效性

•集中式通信：所有工作者均通过参数服务器进行权重更新；无法扩展至大量工作者

•如何在DNN训练中实现通信去中心化?

• AllReduce：在多个设备上执行元素级归约操作

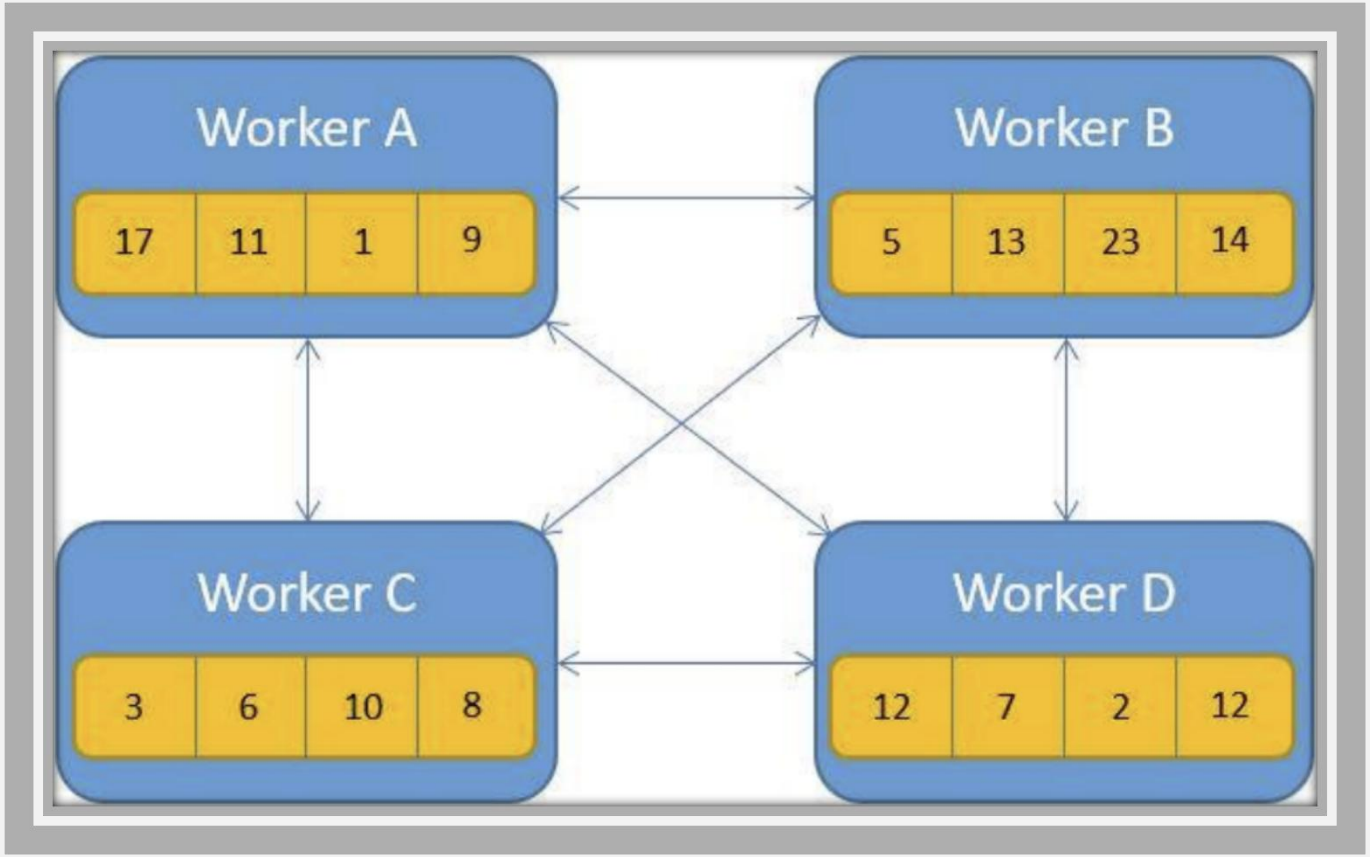


执行AllReduce(全缩减)的不同方式

- 朴素AllReduce
- 环形AllReduce
- 树形AllReduce
- 蝴蝶AllReduce

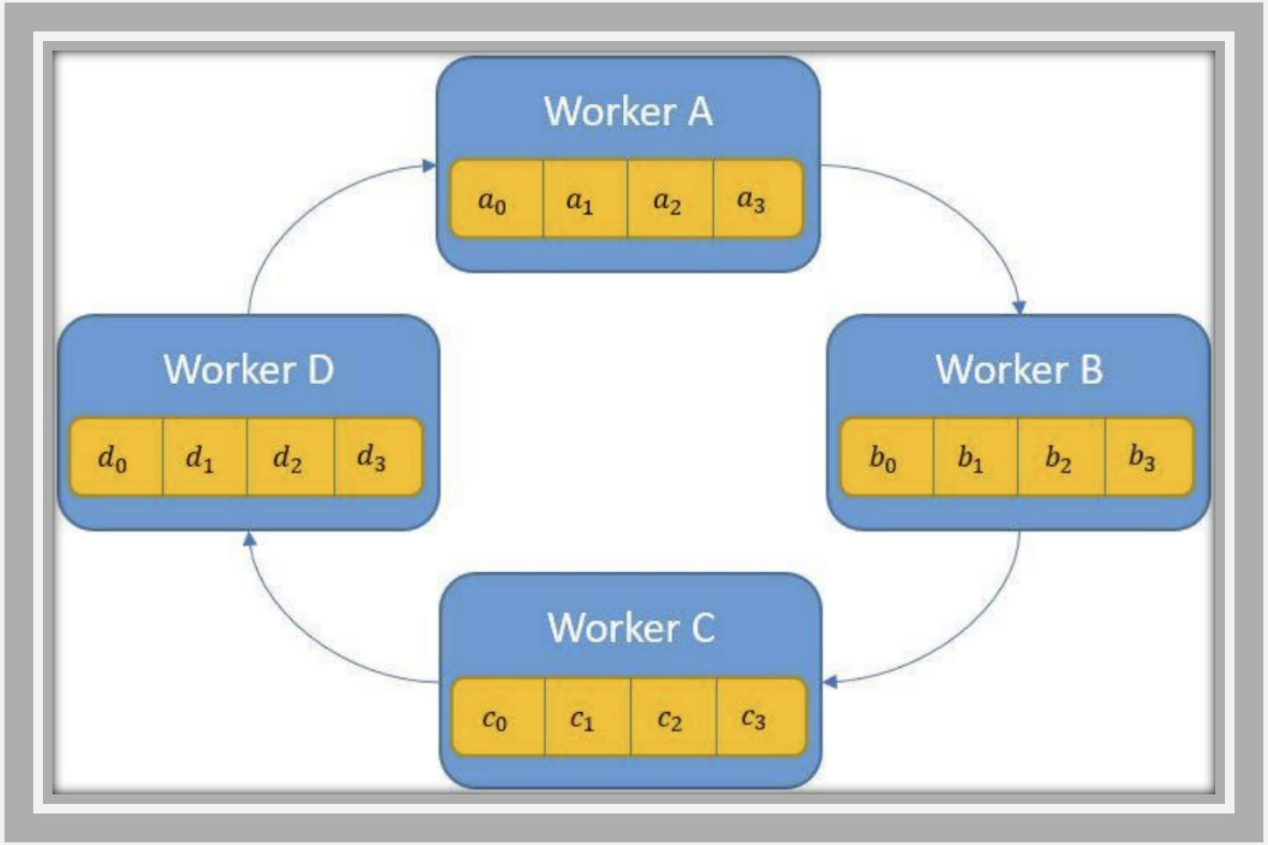
朴素AllReduce

- 每个工作节点可将其本地梯度发送给所有其他工作节点
- 若有 N 个工作节点，且每个节点包含 M 个参数
- 总通信量：N × (N-1) × M 个参数
- 问题：每个工作节点需与所有其他节点通信；存在与参数服务器相同的可扩展性问题



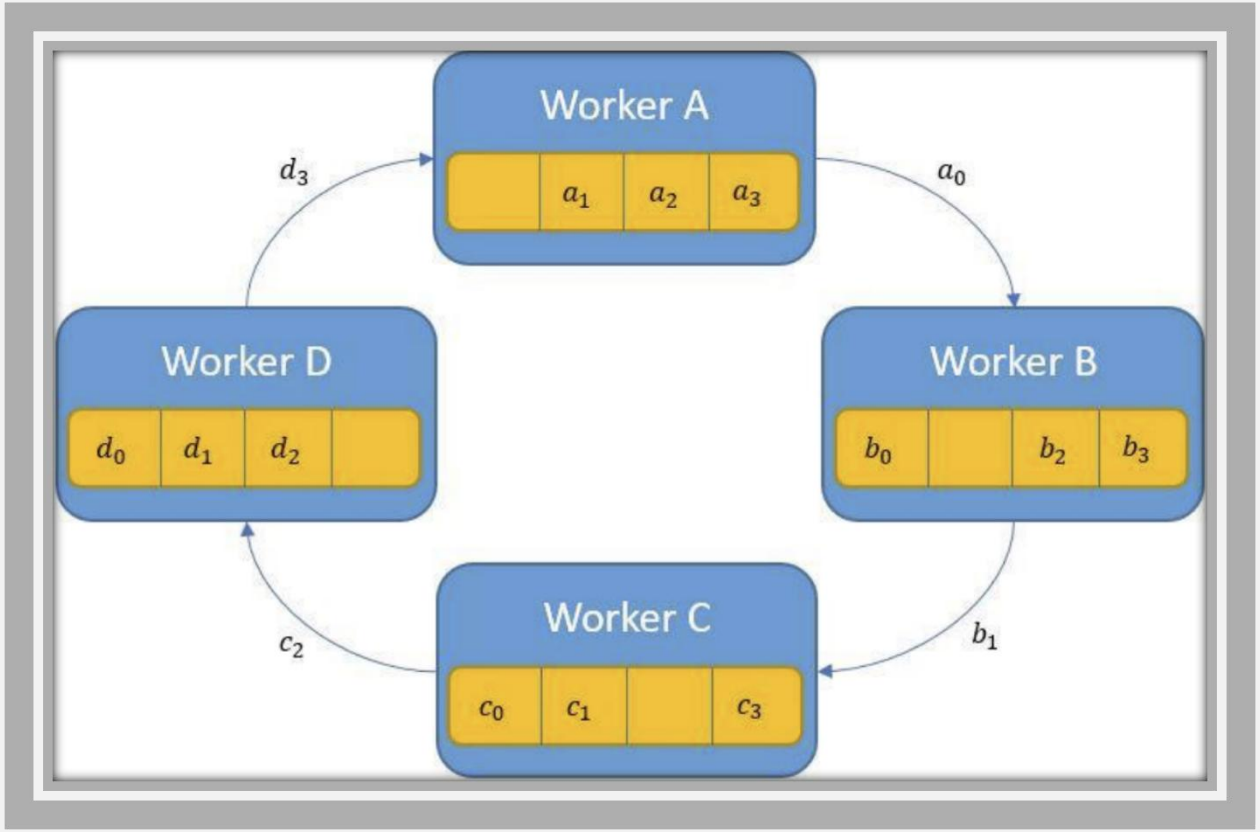
环形AllReduce

- 构建由 N 个工作节点组成的环形网络，将 M 个参数划分为 N 个片段
- 步骤 1（聚合）：每个工作节点将一个片段（M/N 个参数）发送给环形网络中的下一个节点；重复 N 次



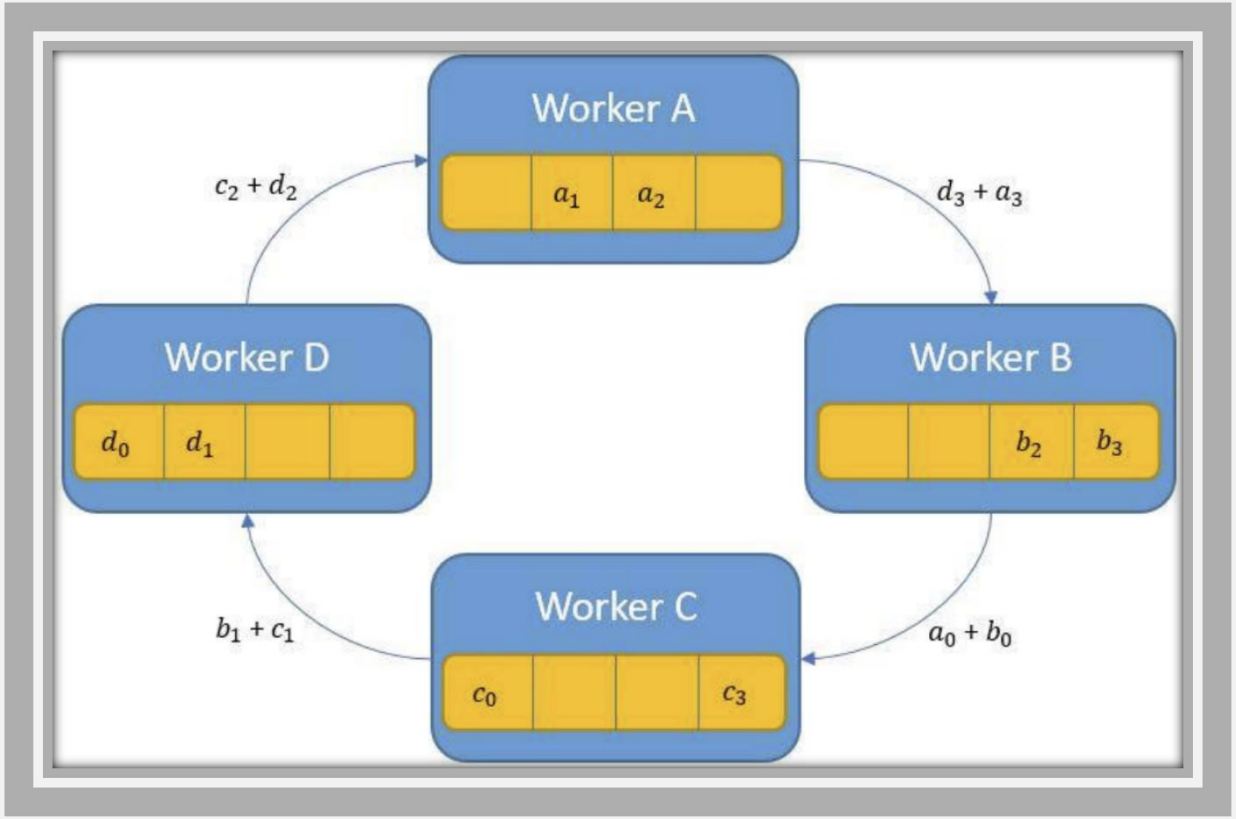
环形AllReduce

- 构建由 N 个工作节点组成的环形网络，将 M 个参数划分为 N 个片段
- 步骤 1（聚合）：每个工作节点将一个片段（M/N 个参数）发送给环形网络中的下一个节点；重复 N 次



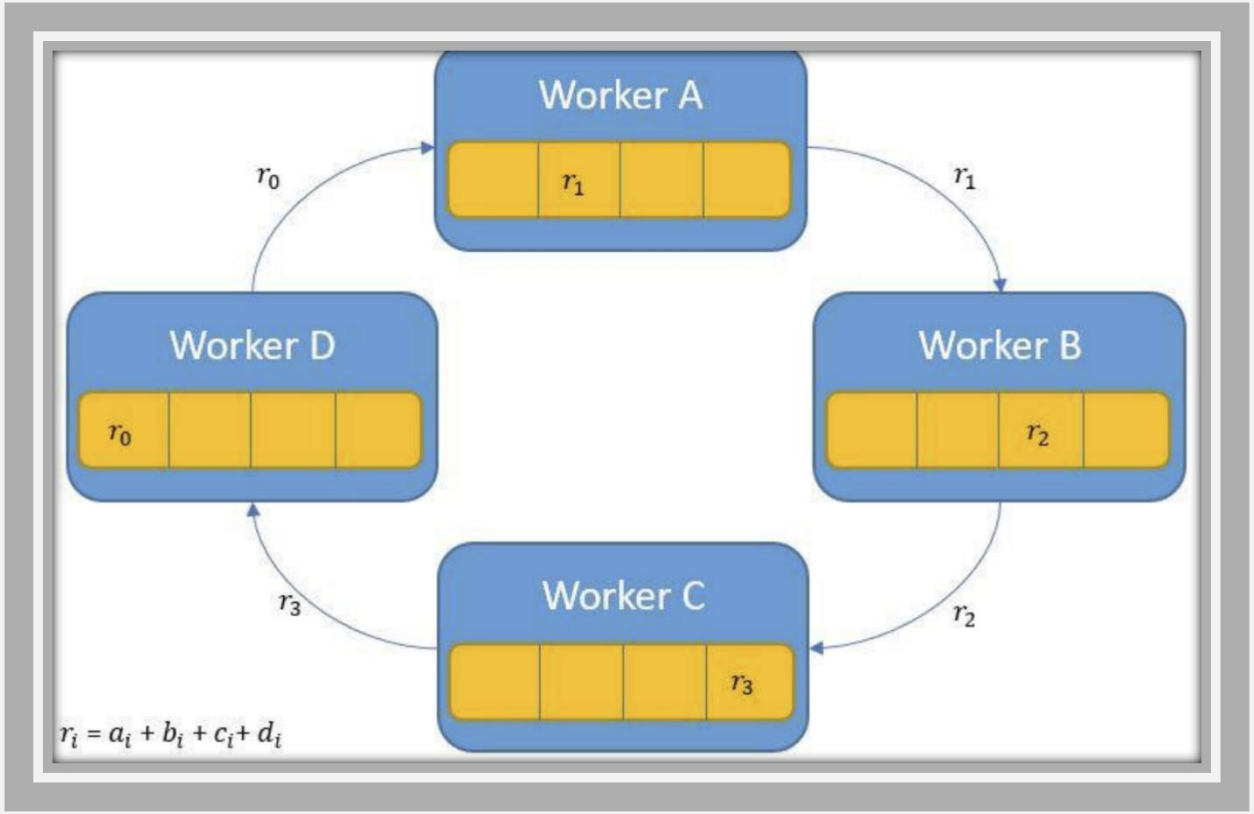
环形AllReduce

- 构建由 N 个工作节点组成的环形网络，将 M 个参数划分为 N 个片段
- 步骤 1（聚合）：每个工作节点将一个片段（M/N 个参数）发送给环形网络中的下一个节点；重复 N 次



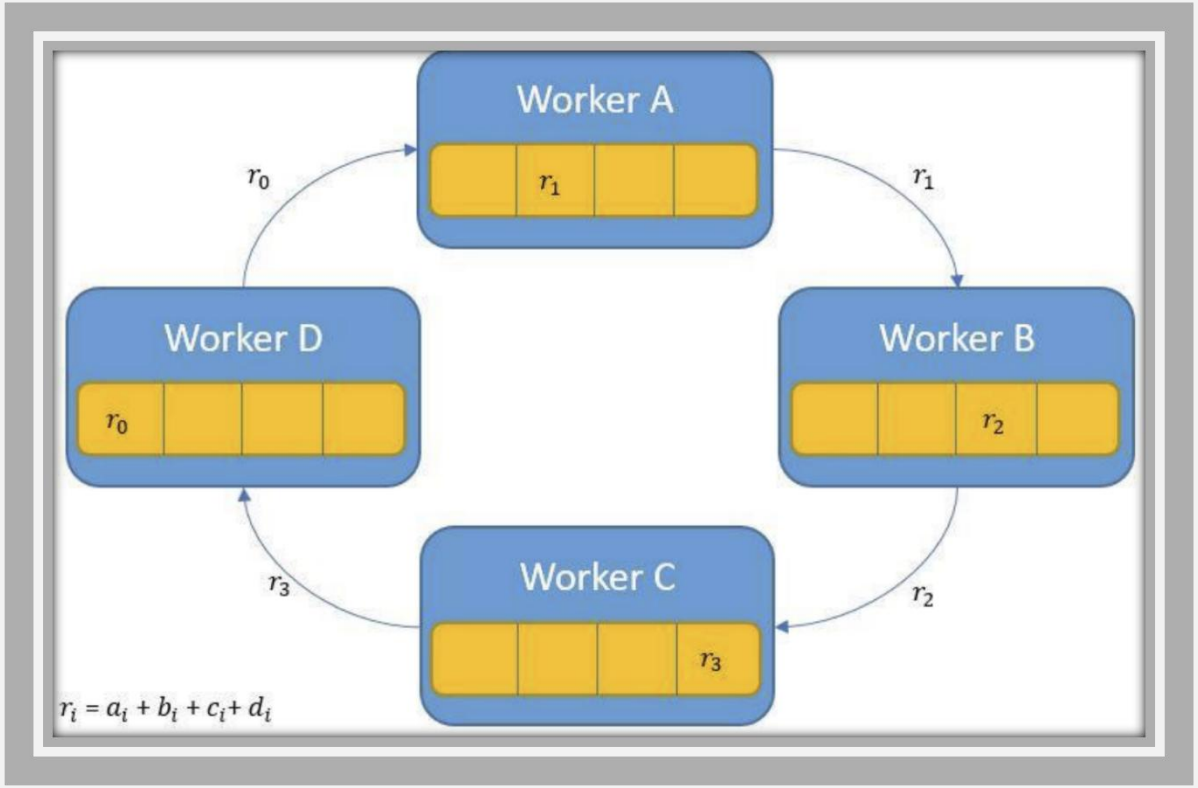
环形AllReduce

- 构建由 N 个工作节点组成的环形网络，将 M 个参数划分为 N 个片段
- 步骤 1（聚合）：每个工作节点将一个片段（M/N 个参数）发送给环形网络中的下一个节点；重复 N 次
- 完成步骤1后，每位工作者都拥有M/N参数的聚合版本



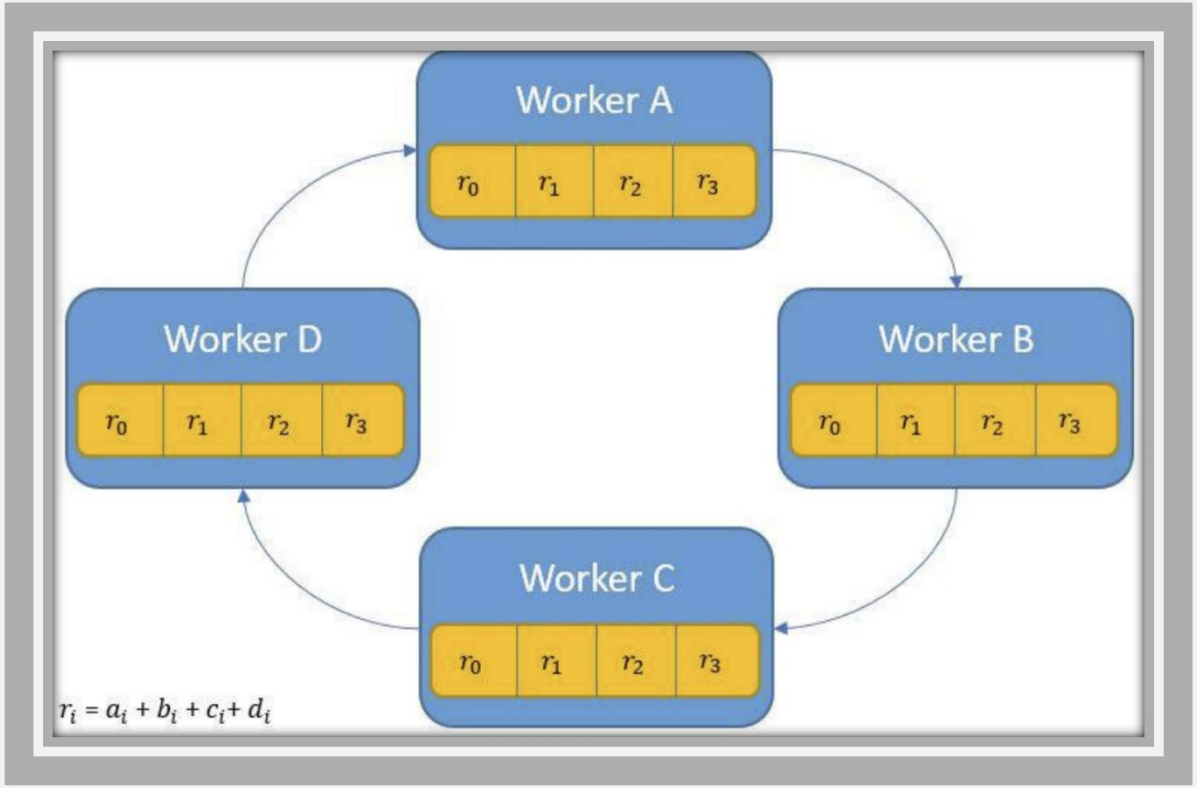
环形AllReduce

- 构建由 N 个工作节点组成的环形网络，将 M 个参数划分为 N 个片段
- 步骤 1（聚合）：每个工作节点将一个片段（M/N 个参数）发送给环形网络中的下一个节点；重复 N 次
- 步骤 2（广播）：每个工作者将一组聚合参数片段发送给下一个工作者；重复 N 次



环形AllReduce

- 构建由 N 个工作节点组成的环形网络，将 M 个参数划分为 N 个片段
- 步骤 1（聚合）：每个工作节点将一个片段（M/N 个参数）发送给环形网络中的下一个节点；重复 N 次
- 步骤 2（广播）：每个工作者将一组聚合参数片段发送给下一个工作者；重复 N 次

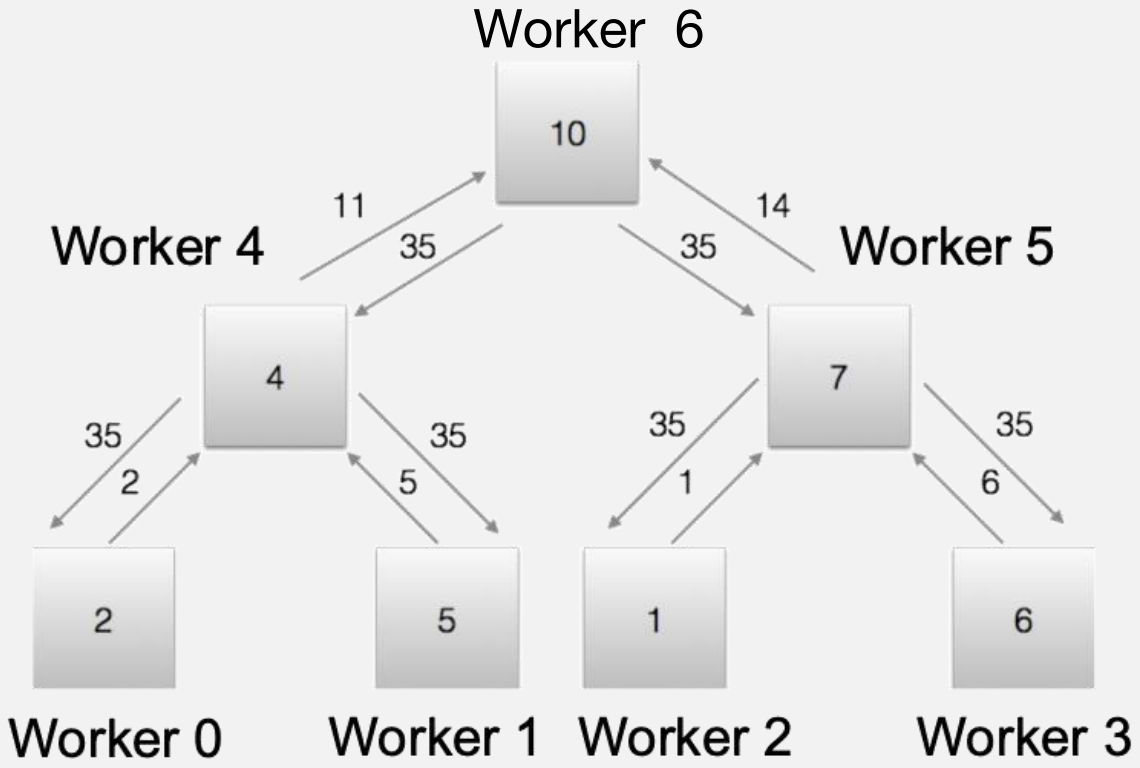


环形AllReduce

- 构建由 N 个工作节点组成的环形网络，将 M 个参数划分为 N 个片段
- 步骤 1（聚合）：每个工作节点将一个片段（M/N 个参数）发送给环形网络中的下一个节点；重复 N 次
- 步骤 2（广播）：每个工作者将一组聚合参数片段发送给下一个工作者；重复 N 次
- 整体通信：2 × M × N 个参数
  - 聚合：M × N 个参数
  - 广播：M × N 个参数

树形AllReduce

- 构建由 N 个工作节点组成的树结构；
- 步骤 1（聚合）：每个工作节点向其父节点发送 M 个参数；重复 log(N) 次
- 步骤 2（广播）：每个工作节点向其子节点发送 M 个参数；重复 log(N) 次



树形AllReduce

- 构建由 N 个工作节点组成的树结构；
- 步骤 1（聚合）：每个工作节点向其父节点发送 M 个参数；

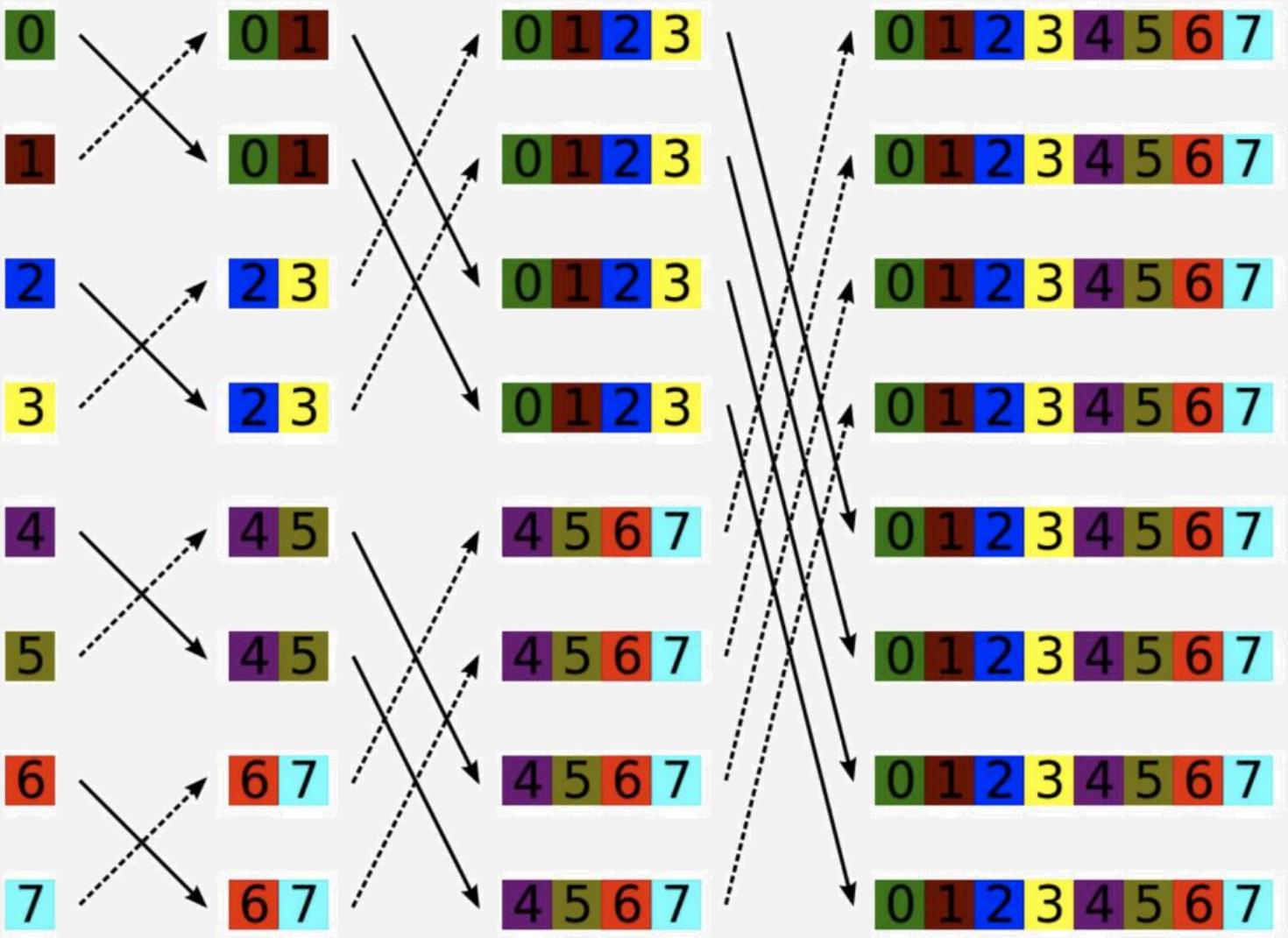
重复 log(N) 次

- 步骤 2（广播）：每个工作节点向其子节点发送 M 个参数；

重复 log(N) 次

- 总通信量：2 * N * M 个参数
- 聚合操作：M * N 个参数
- 广播操作：M * N 个参数

蝴蝶AllReduce

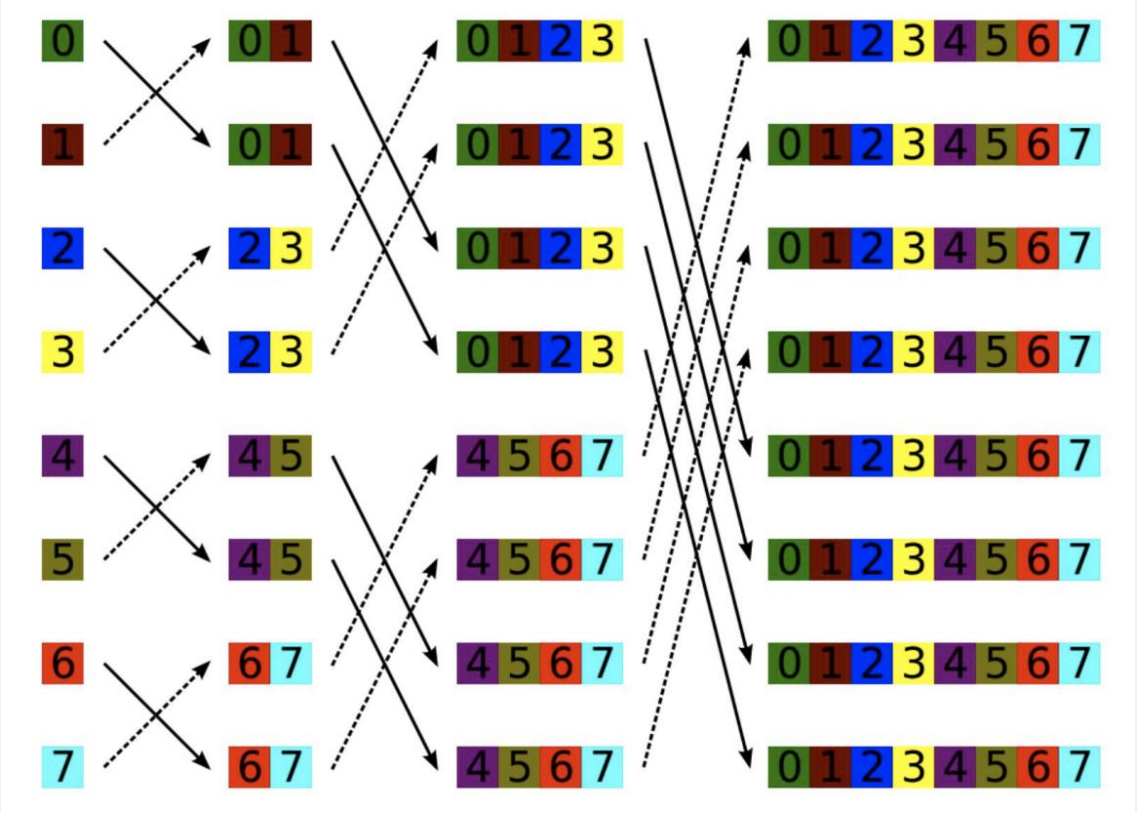


蝴蝶AllReduce

- 重复 log(N) 次：

1. 每个工作者向蝴蝶网络中的目标节点发送 M 个参数
2. 每个工作者本地聚合梯度

- 总通信量：N * M * log(N) 个参数

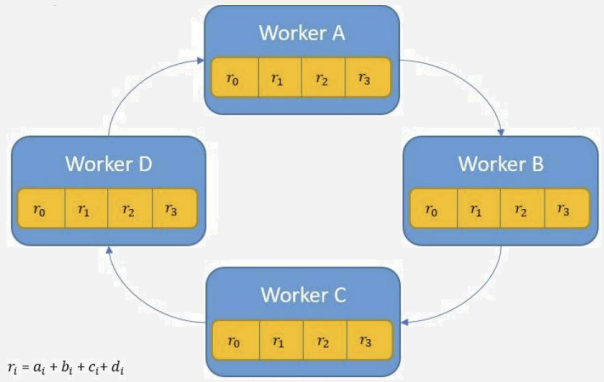


比较不同的AllReduce方法

	参数服务器	朴素AllReduce	环形朴素AllReduce	树形朴素AllReduce	蝴蝶朴素AllReduce
Overall communication	2 × N × M	N² × M×	2 × N × M	2 × N × M	N× M× log N

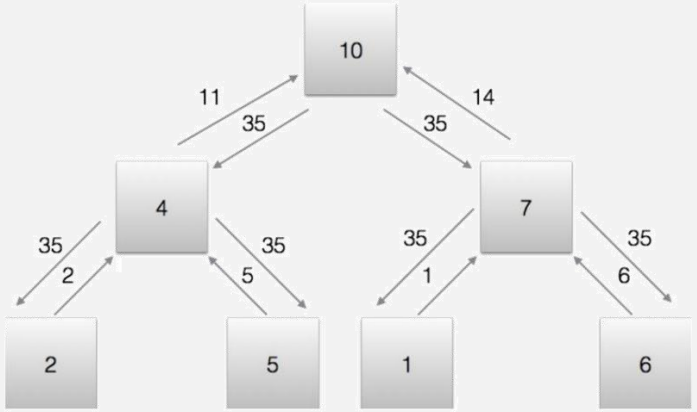
问题：环形AllReduce比树形AllReduce和参数服务器更高效且可扩展，为什么?

环形AllReduce v.s. 树形AllReduce v.s. 参数服务器

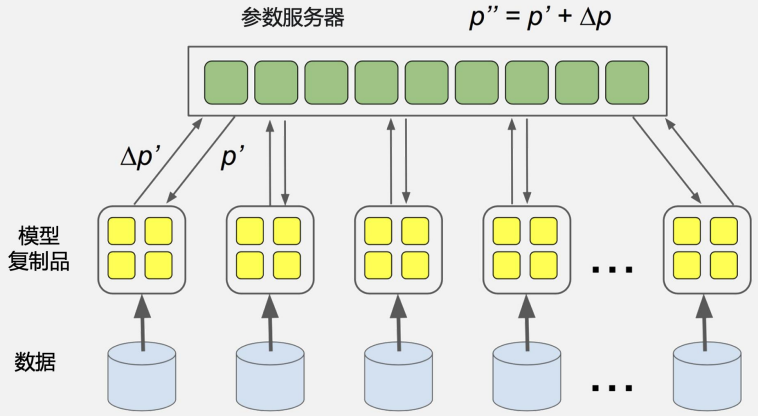


每个工作者每轮发送 M/N 个参数；
重复执行 2*N 轮迭代
延迟：M/N * (2*N) / 带宽

环形 AllReduce：
- 最佳延迟
- 工作负载在各工作节点间均衡分布
- 更具可扩展性，因每个工作节点发送2*M个参数（与工作节点数量无关）



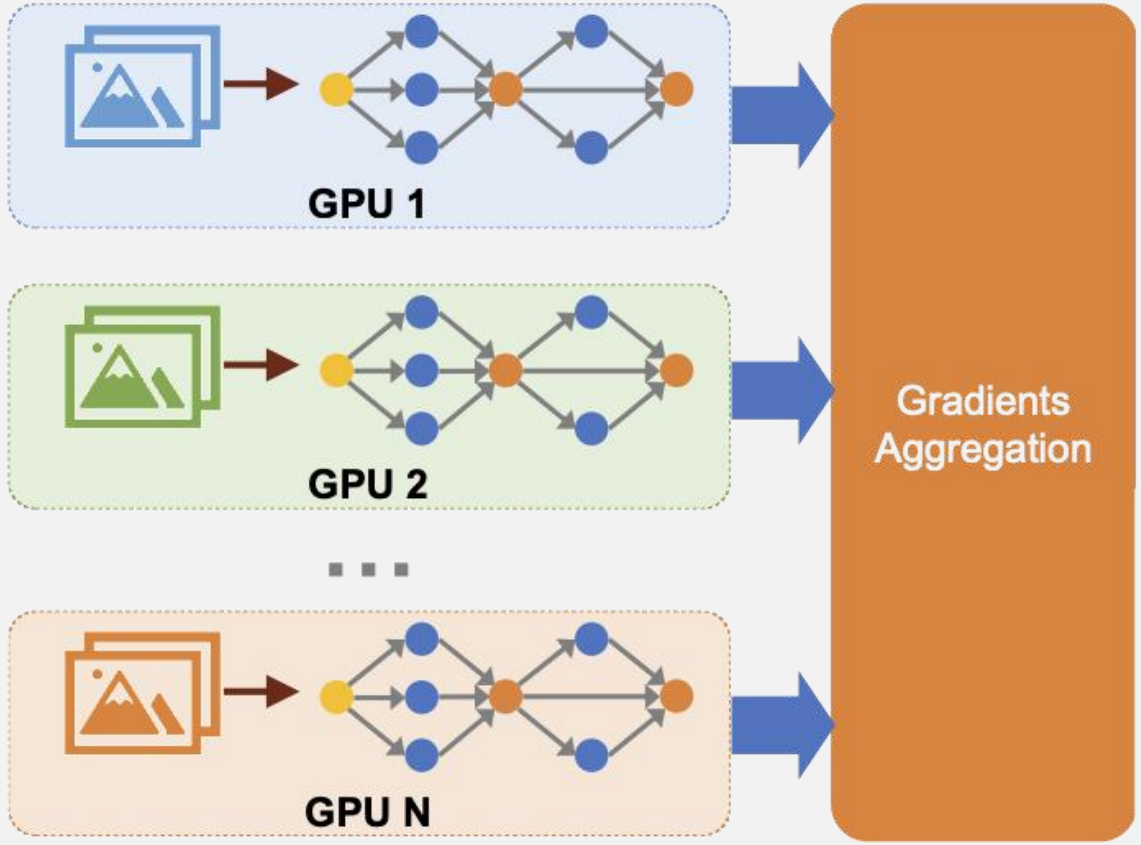
每个工作者每轮发送 M 个参数；
重复 2*log(N) 次迭代
延迟：M * 2 * log(N) / 带宽



所有工作者向参数服务器发送 M 个参数，并从服务器接收 M 个参数
延迟：M * N / 带宽

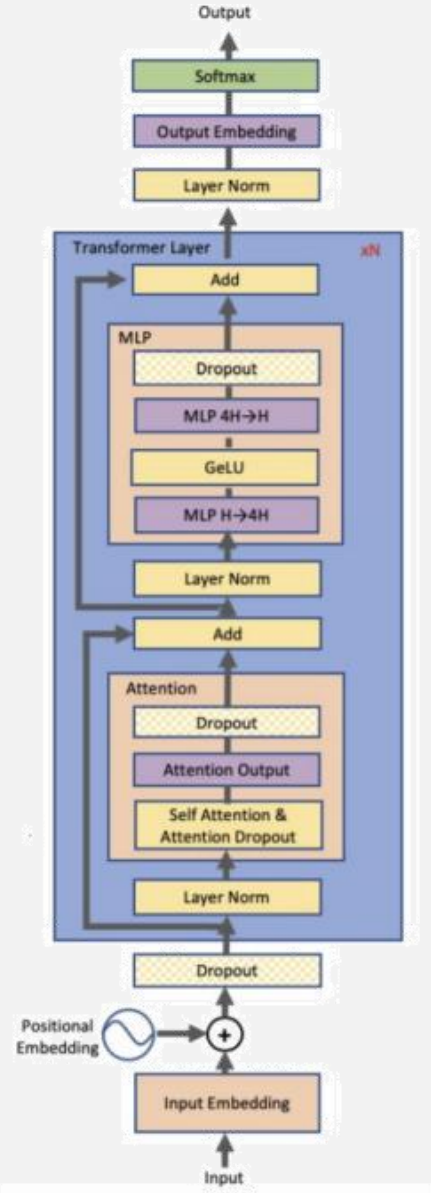
数据并行性问题

- 每块GPU都会保存整个模型的副本
- 无法训练超过GPU设备内存容量的大型模型



大型模型训练挑战

	Bert-Large	GPT-2	Turing 17.2 NLG	GPT-3
Parameters	0.32B	1.5B	17.2B	175B
Layers	24	48	78	96
Hidden Dimension	1024	1600	4256	12288
Relative Computation	1x	4.7x	**54x**	**547x**
Memory Footprint	5.12GB	24GB	**275GB**	**2800GB**

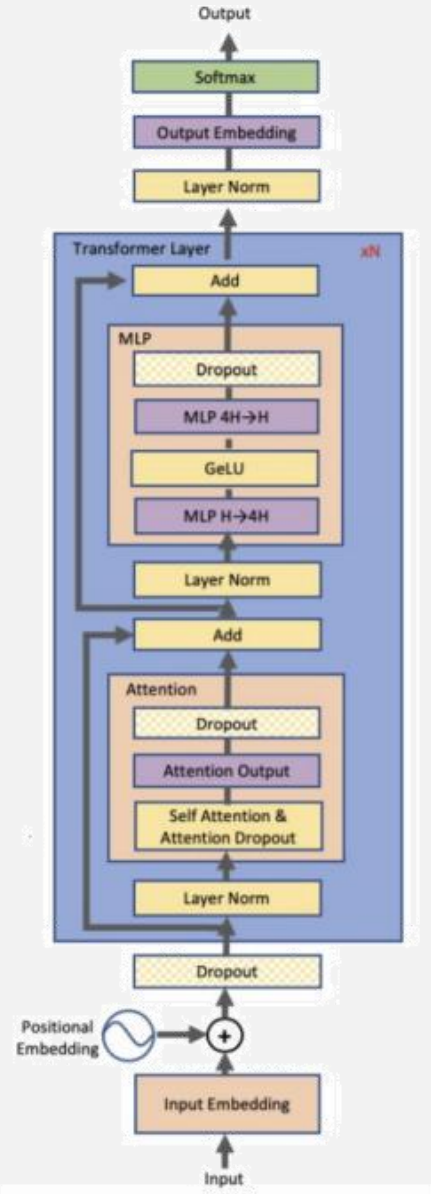


大型模型训练挑战

	Bert-Large	GPT-2	Turing 17.2 NLG	GPT-3
Parameters	0.32B	1.5B	17.2B	175B
Layers	24	48	78	96
Hidden Dimension	1024	1600	4256	12288
Relative Computation	1x	4.7x	**54x**	**547x**
Memory Footprint	5.12GB	24GB	**275GB**	**2800GB**

Out of Memory

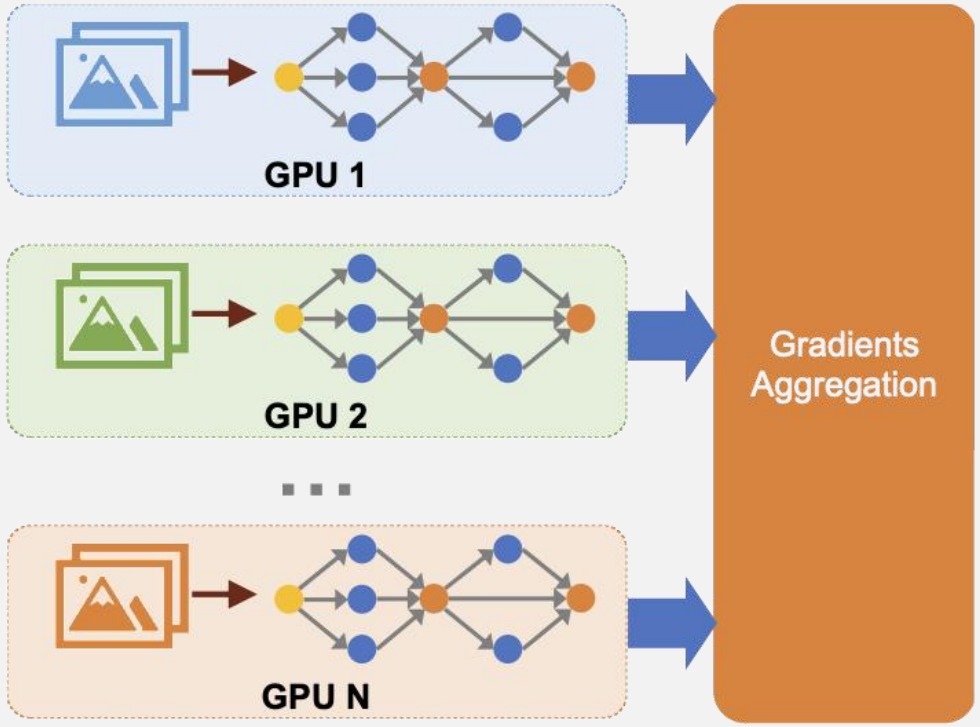
NVIDIA V100 GPU存储容量: 16G/32G
NVIDIA A100 GPU存储容量:
40G/80G



ZeRO: 零冗余优化器

deepspeed

- 消除数据并行训练中的数据冗余
- 用于大型模型数据并行训练的广泛采用技术



重温：随机梯度下降法

Backward pass

Forward pass

For t = 1 to T

$\Delta w = \eta \times \frac{1}{b} \sum_{i=1}^{b} \nabla \left(loss\left(f_w(x_i, y_i) \right) \right)$ // compute derivative and update

w -= Δw // apply update

End

Adapted from Minjia Zhang, DeepSpeed Presentation

自适应学习率（Adam）

For t = 1 to T

$g = \frac{1}{b}\sum_{i=1}^{b} \nabla \left(loss\left(f_w(x_i, y_i)\right)\right)$

$\Delta w = adam(g)$

$w \mathrel{-}= \Delta w$ // apply update

End

$$\nu_t = \beta_1 * \nu_{t-1} - (1 - \beta_1) * g_t$$

$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * g_t^2$$

$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$

g_t : *Gradient at time t along* ω^j

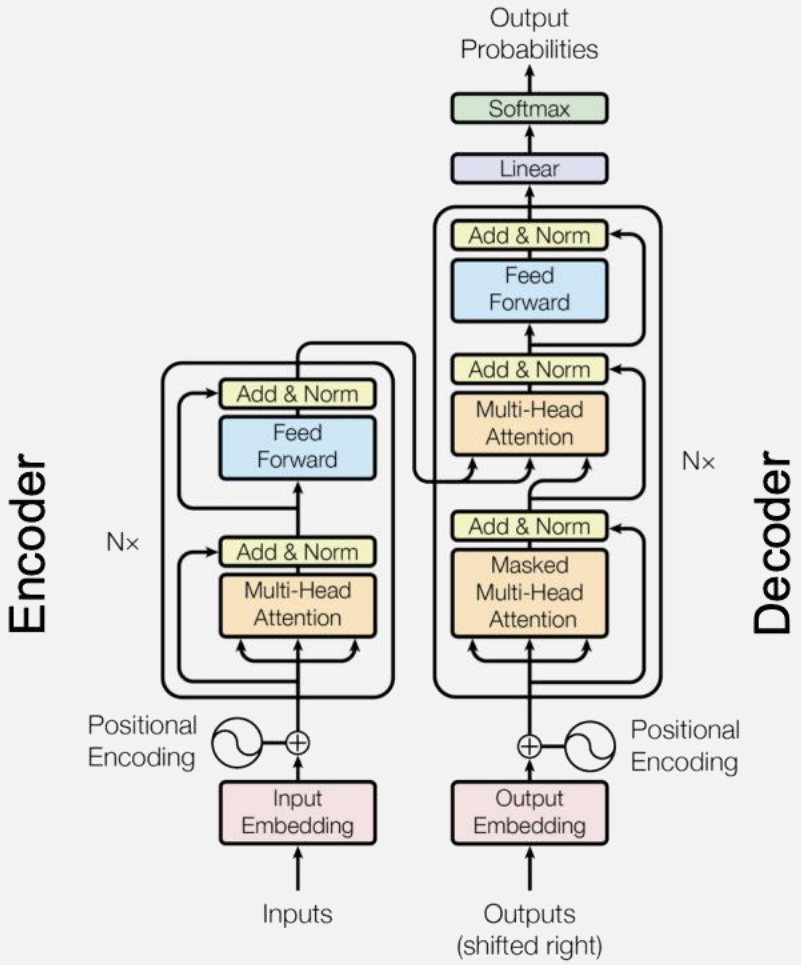
ν_t : *Exponential Average of gradients along* ω_j

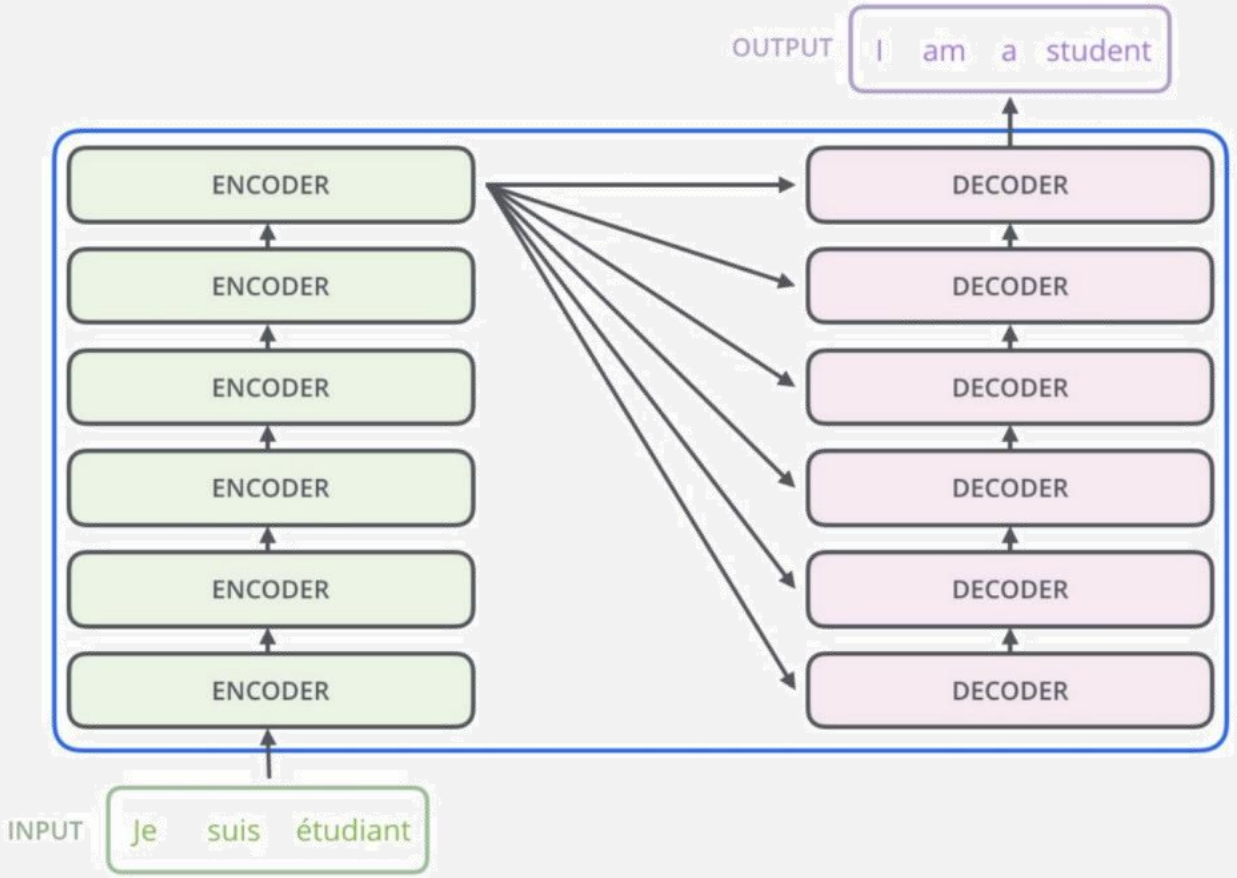
s_t : *Exponential Average of squares of gradients along* ω_j

β_1, β_2 : *Hyperparameters*

[1] Kingma and Ba, “Adam: A Method for Stochastic Optimization”, 2014, https://arxiv.org/abs/1412.6980

语言模型Transformer





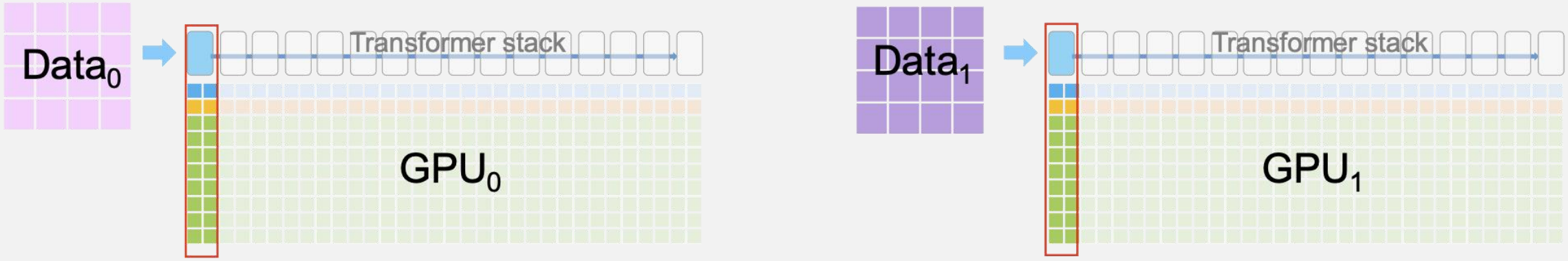
Ashish Vaswani et. al. Attention is all you need.

理解内存消耗

一个16层变压器模型 =1层

Adapted from Minjia Zhang, DeepSpeed Presentation

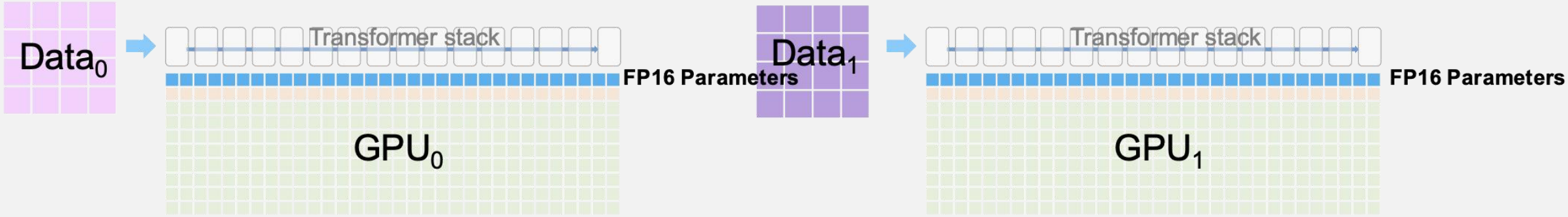
理解内存消耗



每个单元格 代表其对应的Transformer层所使用的GPU内

Adapted from Minjia Zhang, DeepSpeed Presentation

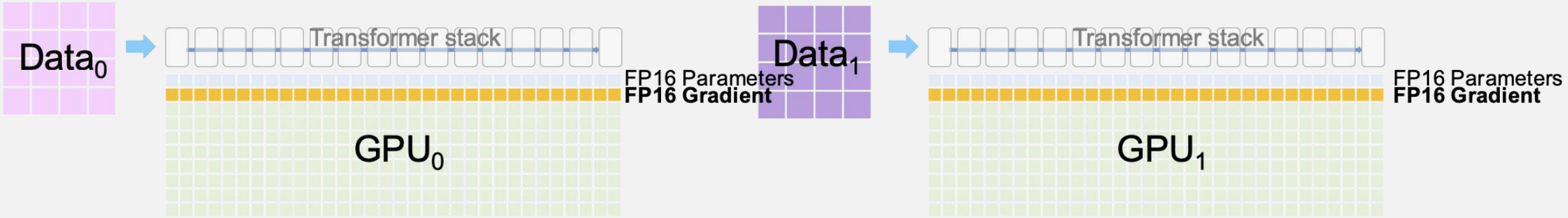
理解内存消耗



- FP16参数

Adapted from Minjia Zhang, DeepSpeed Presentation

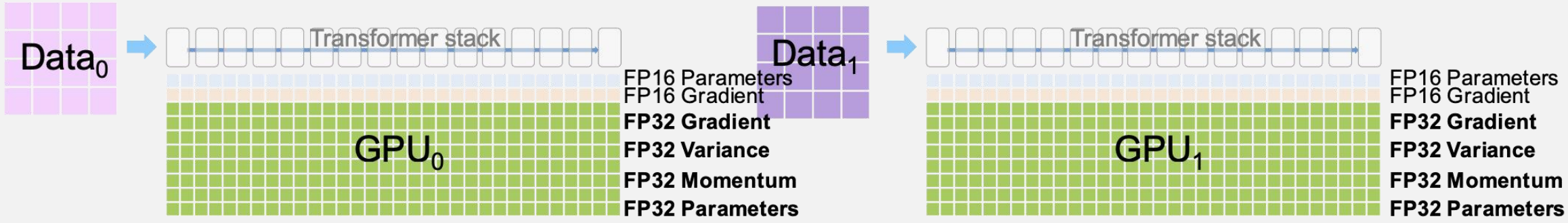
理解内存消耗



- FP16参数
- FP16梯度

Adapted from Minjia Zhang, DeepSpeed Presentation

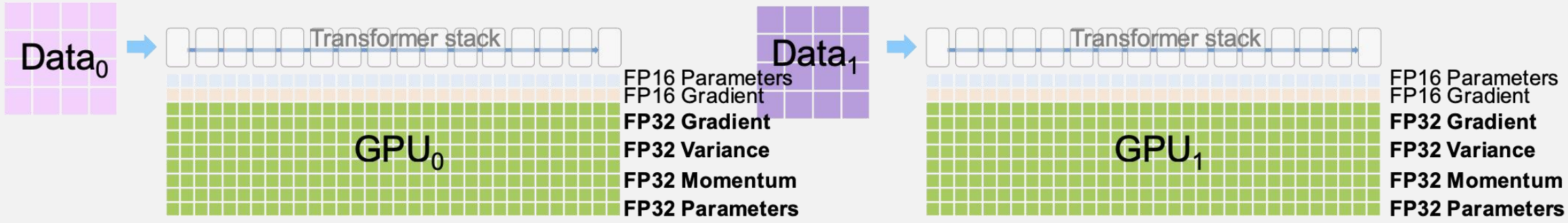
理解内存消耗



- FP16参数
- FP16梯度
- FP32优化器状态
  - 梯度、方差、动量、参数

Adapted from Minjia Zhang, DeepSpeed Presentation

理解内存消耗



- FP16参数
- FP16梯度
- FP32优化器状态
  - 梯度、方差、动量、参数

M = 模型中的参数数量

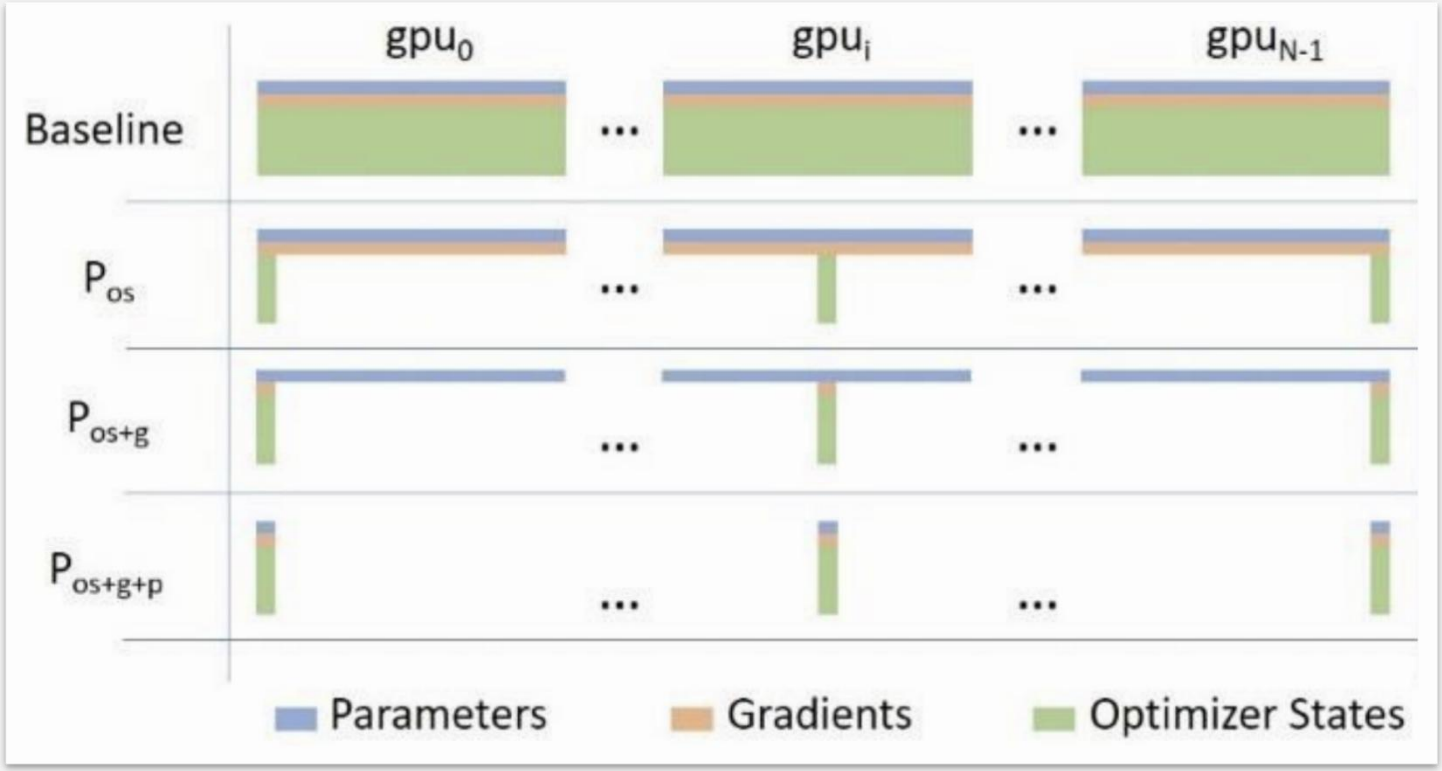
示例1B参数模型 -> 20GB/GPU

内存消耗不包括：
- 输入批次 + 激活值

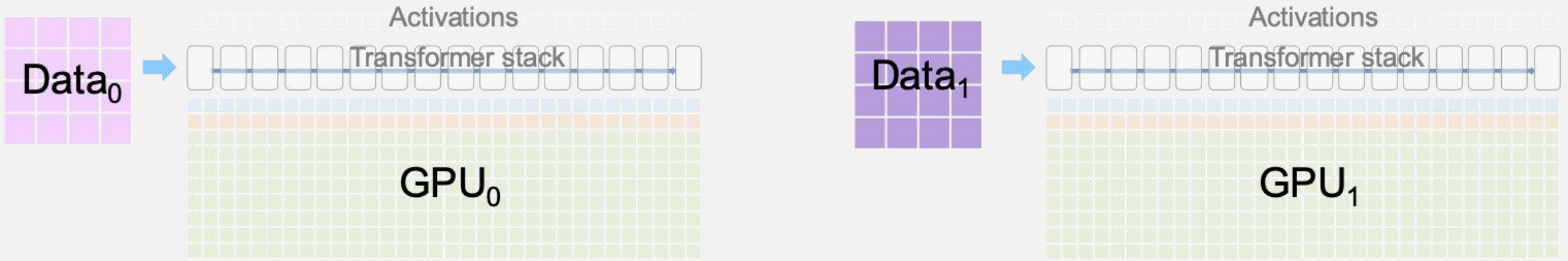
Adapted from Minjia Zhang, DeepSpeed Presentation

ZeRO-DP：基于ZeRO的数据并行性

- ZeRO 消除了数据并行处理过程中的冗余
- 第一阶段：划分优化器状态
- 第二阶段：划分梯度
- 第三阶段：划分参数



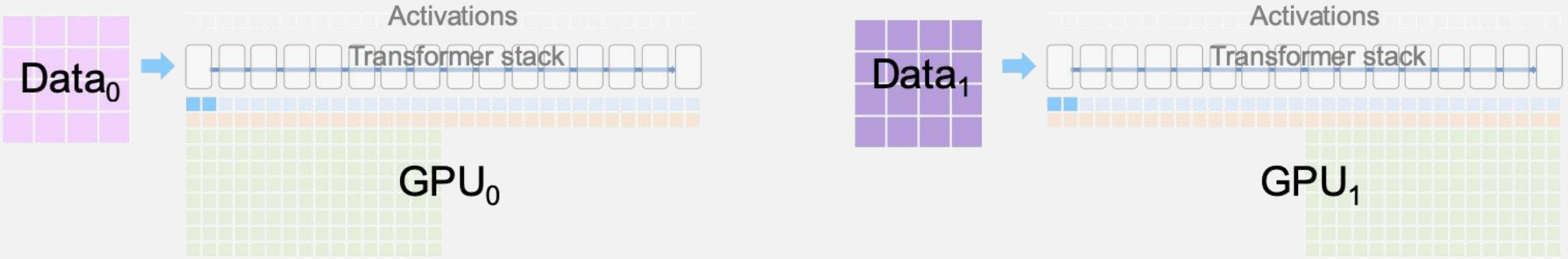
ZeRO 第一阶段：分区优化器状态



- ZeRO Stage 1

Adapted from Minjia Zhang, DeepSpeed Presentation

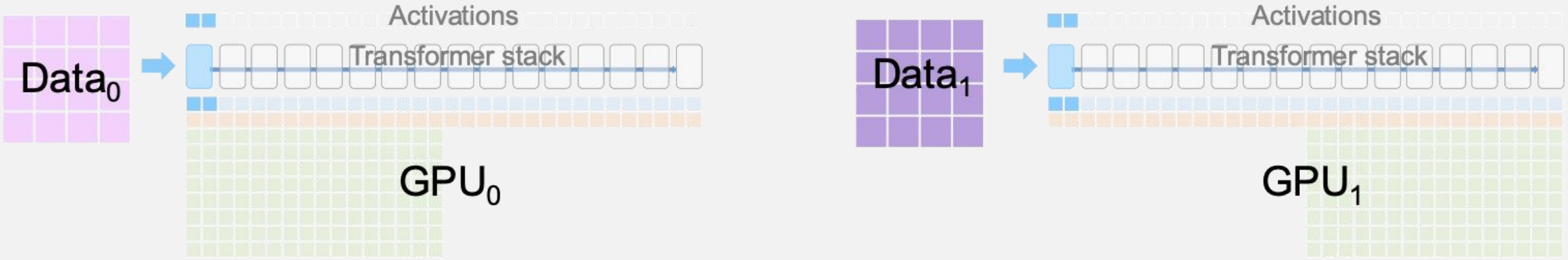
ZeRO 第一阶段：分区优化器状态



- ZeRO Stage 1
- 分区优化器状态跨GPU同步

Adapted from Minjia Zhang, DeepSpeed Presentation

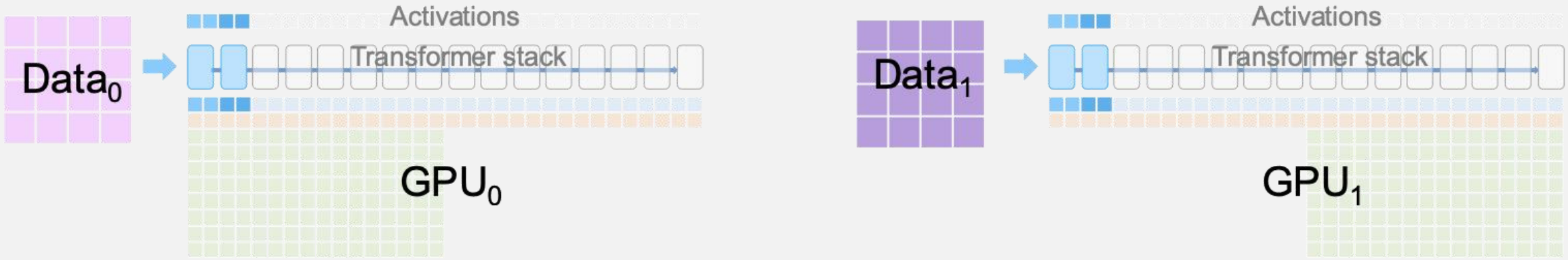
ZeRO 第一阶段：分区优化器状态



- ZeRO Stage 1
- 分区优化器状态跨GPU同步
- 在transformer块间执行前向传播

Adapted from Minjia Zhang, DeepSpeed Presentation

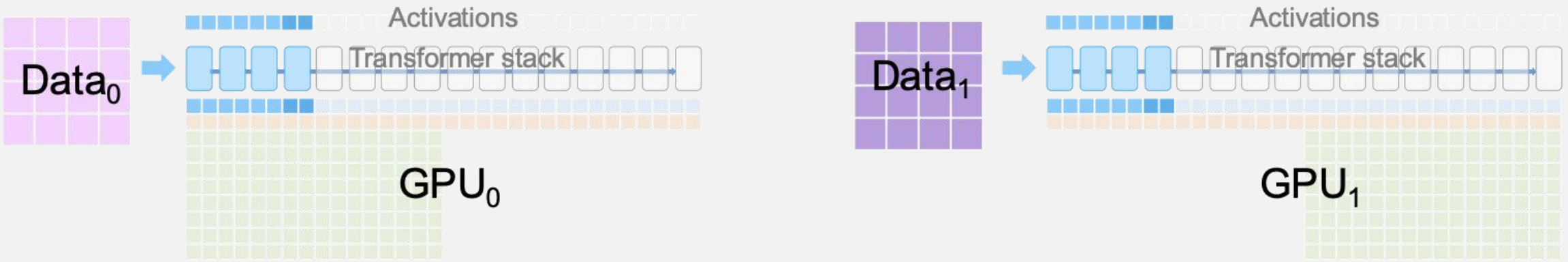
ZeRO 第一阶段：分区优化器状态



- ZeRO Stage 1
- 分区优化器状态跨GPU同步
- 在transformer块间执行前向传播

Adapted from Minjia Zhang, DeepSpeed Presentation

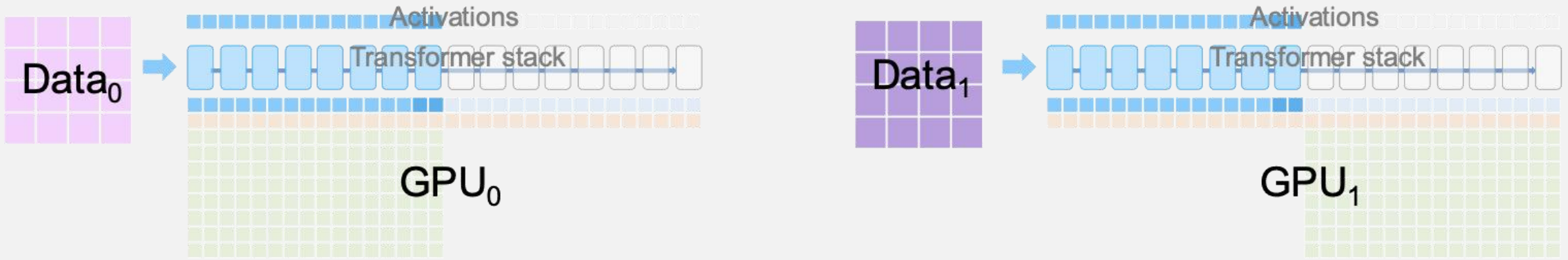
ZeRO 第一阶段：分区优化器状态



- ZeRO Stage 1
- 分区优化器状态跨GPU同步
- 在transformer块间执行前向传播

Adapted from Minjia Zhang, DeepSpeed Presentation

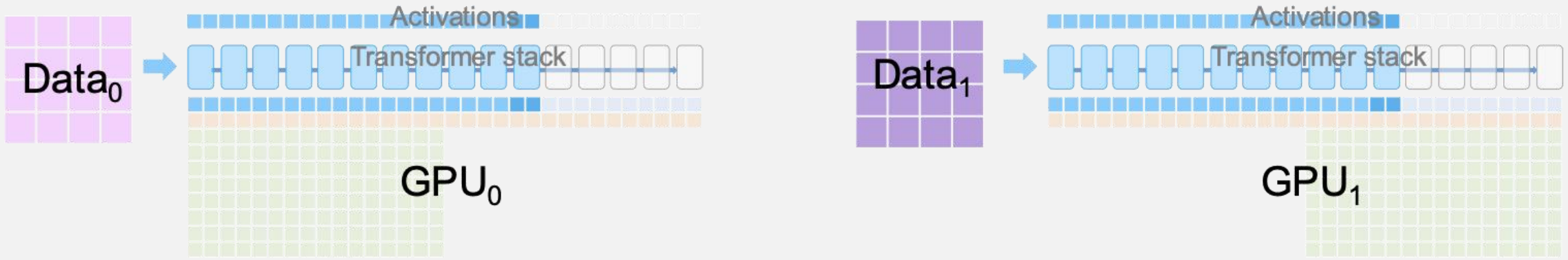
ZeRO 第一阶段：分区优化器状态



- ZeRO Stage 1
- 分区优化器状态跨GPU同步
- 在transformer块间执行前向传播

Adapted from Minjia Zhang, DeepSpeed Presentation

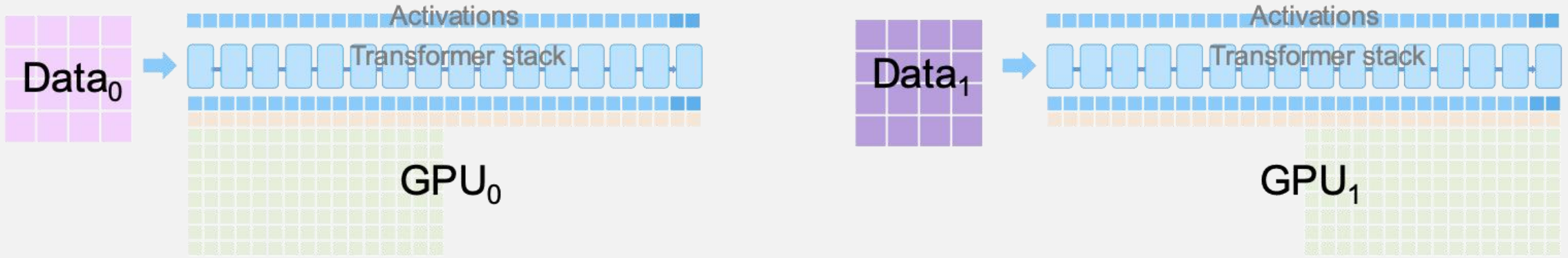
ZeRO 第一阶段：分区优化器状态



- ZeRO Stage 1
- 分区优化器状态跨GPU同步
- 在transformer块间执行前向传播

Adapted from Minjia Zhang, DeepSpeed Presentation

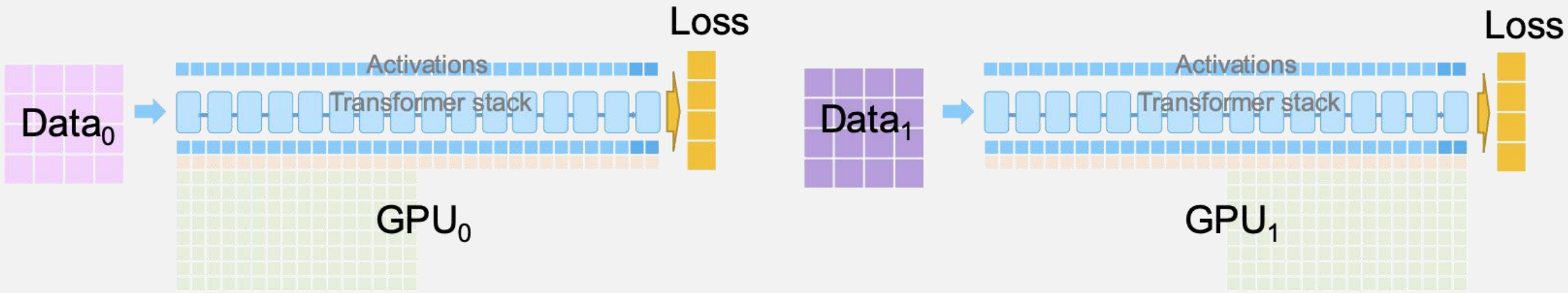
ZeRO 第一阶段：分区优化器状态



- ZeRO Stage 1
- 分区优化器状态跨GPU同步
- 在transformer块间执行前向传播

Adapted from Minjia Zhang, DeepSpeed Presentation

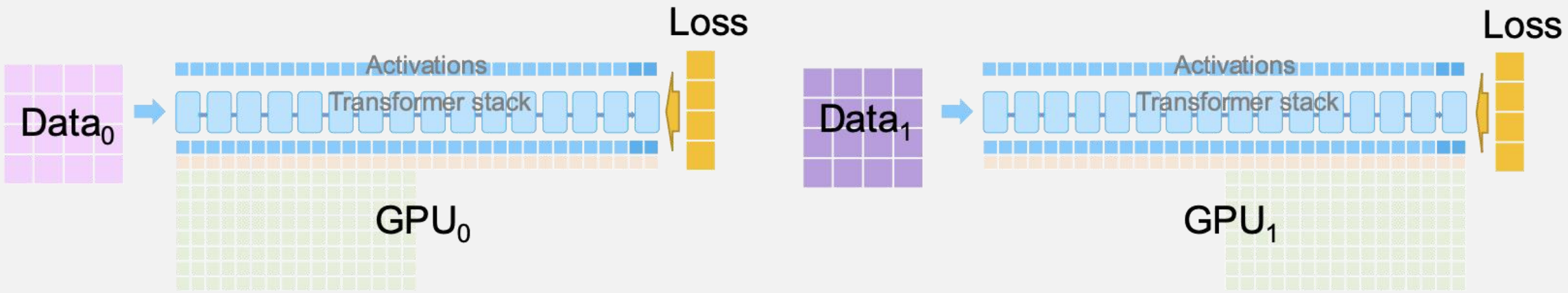
ZeRO 第一阶段：分区优化器状态



- ZeRO Stage 1
- 分区优化器状态跨GPU同步
- 在transformer块间执行前向传播

Adapted from Minjia Zhang, DeepSpeed Presentation

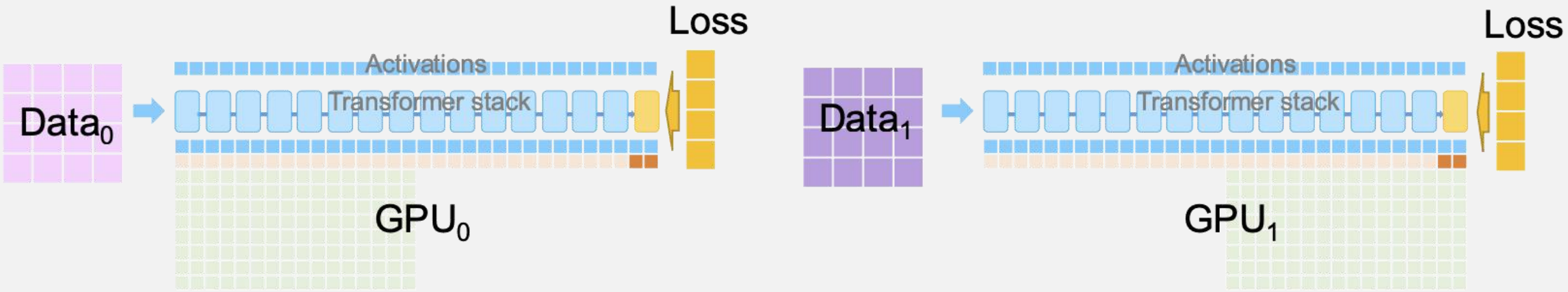
ZeRO 第一阶段：分区优化器状态



- ZeRO Stage 1
- 分区优化器状态跨GPU同步
- 在transformer块间执行前向传播
- 反向传播以生成FP16梯度

Adapted from Minjia Zhang, DeepSpeed Presentation

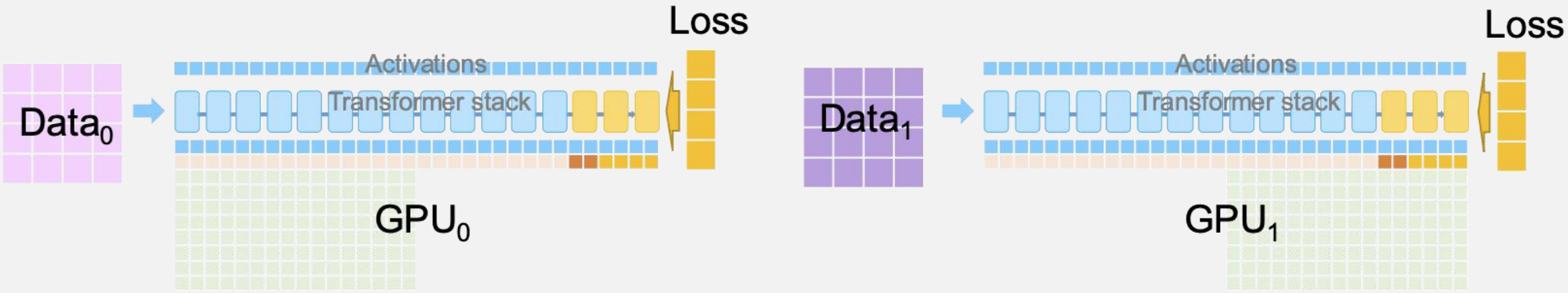
ZeRO 第一阶段：分区优化器状态



- ZeRO Stage 1
- 分区优化器状态跨GPU同步
- 在transformer块间执行前向传播
- 反向传播以生成FP16梯度

Adapted from Minjia Zhang, DeepSpeed Presentation

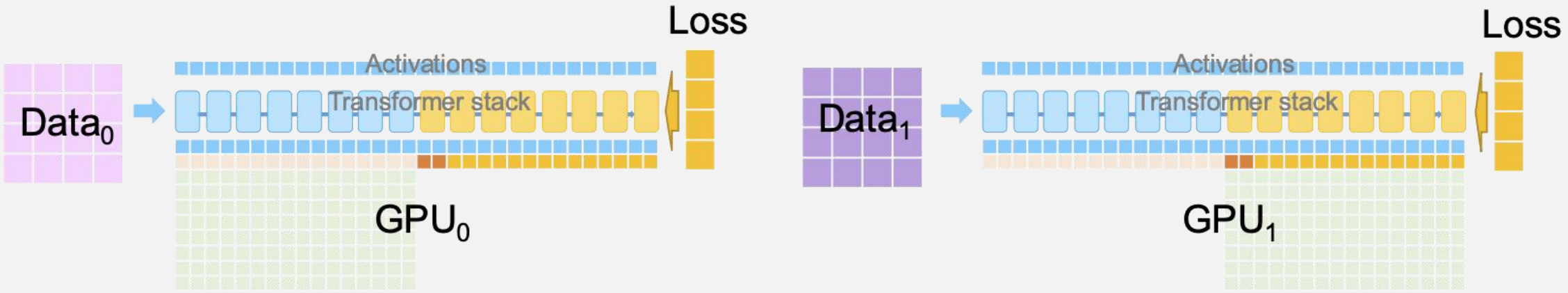
ZeRO 第一阶段：分区优化器状态



- ZeRO Stage 1
- 分区优化器状态跨GPU同步
- 在transformer块间执行前向传播
- 反向传播以生成FP16梯度

Adapted from Minjia Zhang, DeepSpeed Presentation

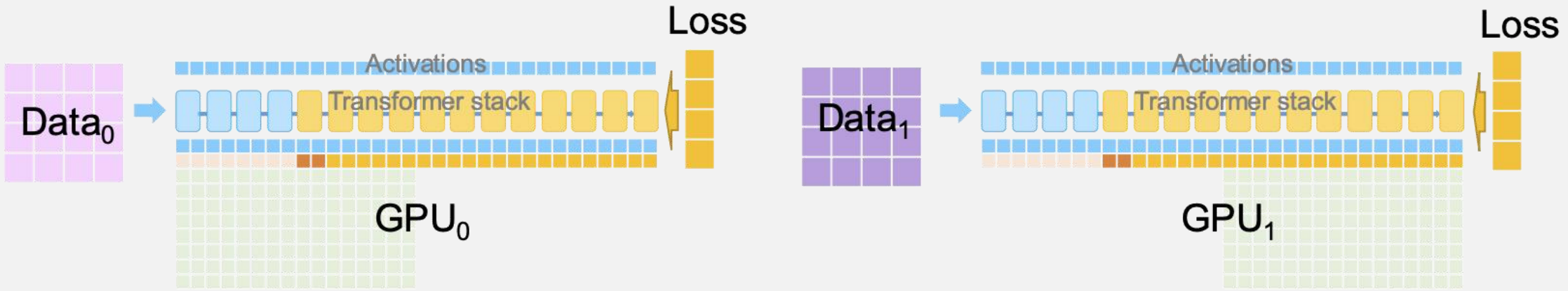
ZeRO 第一阶段：分区优化器状态



- ZeRO Stage 1
- 分区优化器状态跨GPU同步
- 在transformer块间执行前向传播
- 反向传播以生成FP16梯度

Adapted from Minjia Zhang, DeepSpeed Presentation

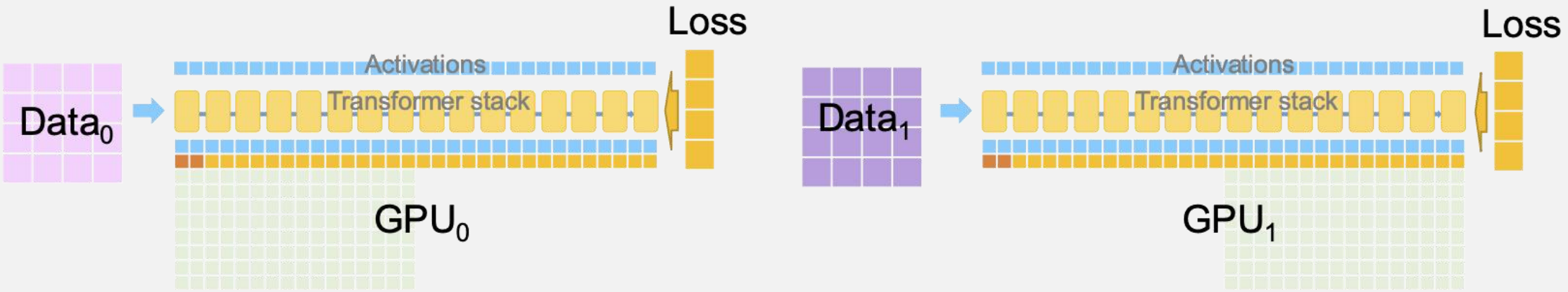
ZeRO 第一阶段：分区优化器状态



- ZeRO Stage 1
- 分区优化器状态跨GPU同步
- 在transformer块间执行前向传播
- 反向传播以生成FP16梯度

Adapted from Minjia Zhang, DeepSpeed Presentation

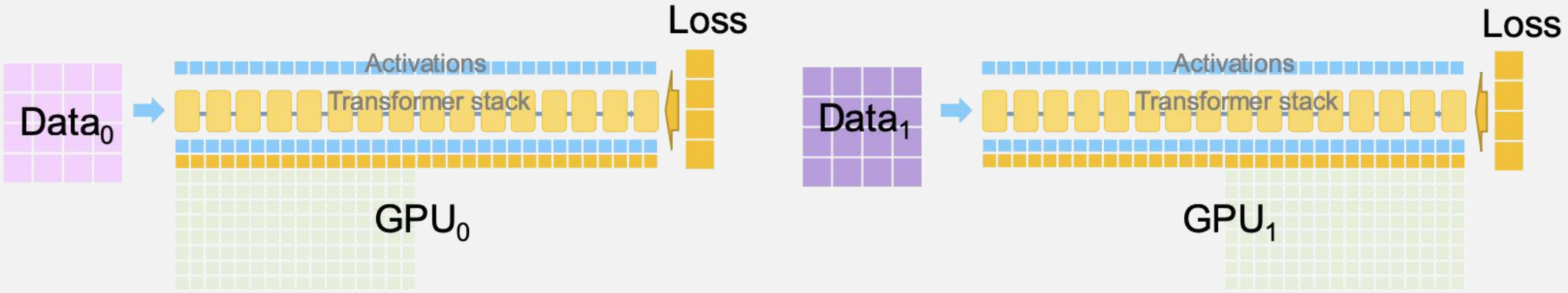
ZeRO 第一阶段：分区优化器状态



- ZeRO Stage 1
- 分区优化器状态跨GPU同步
- 在transformer块间执行前向传播
- 反向传播以生成FP16梯度

Adapted from Minjia Zhang, DeepSpeed Presentation

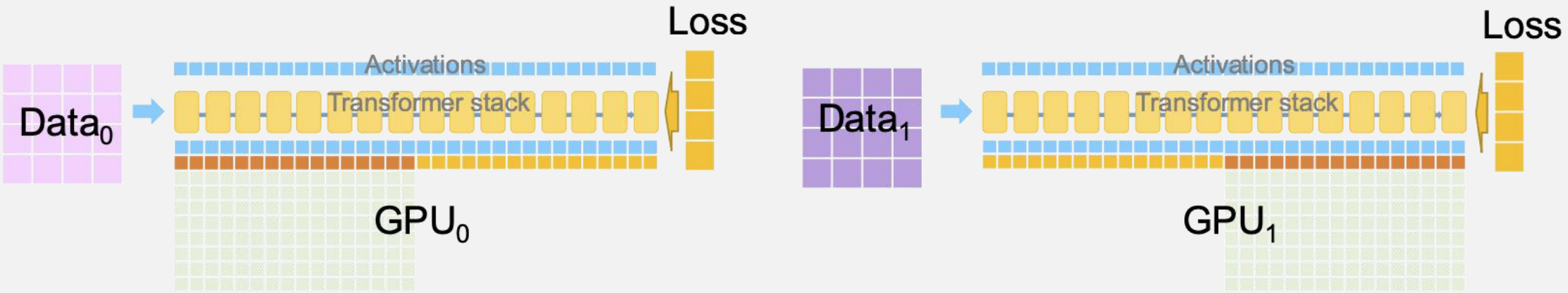
ZeRO 第一阶段：分区优化器状态



- ZeRO Stage 1
- 分区优化器状态跨GPU同步
- 在transformer块间执行前向传播
- 通过反向传播生成FP16梯度，并使用AllReduce进行平均

Adapted from Minjia Zhang, DeepSpeed Presentation

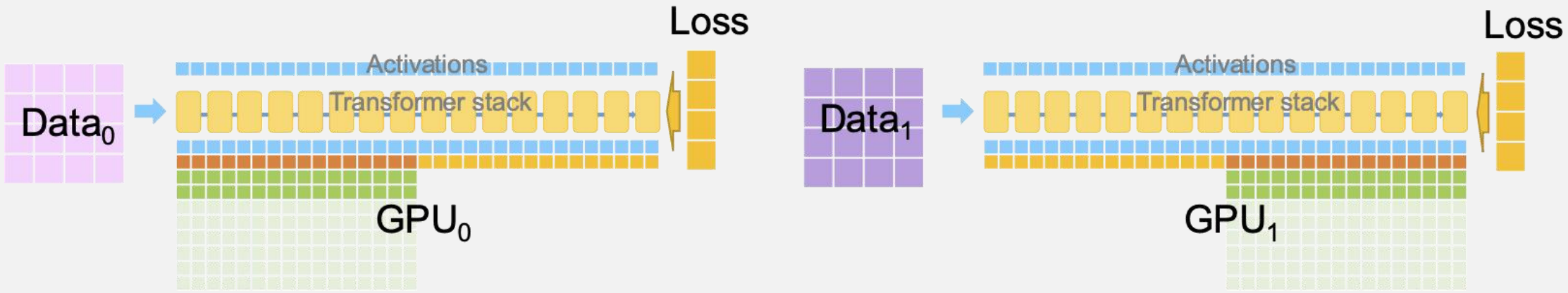
ZeRO 第一阶段：分区优化器状态



- ZeRO Stage 1
- 分区优化器状态跨GPU同步
- 在transformer块间执行前向传播
- 通过反向传播生成FP16梯度，并使用AllReduce进行平均

Adapted from Minjia Zhang, DeepSpeed Presentation

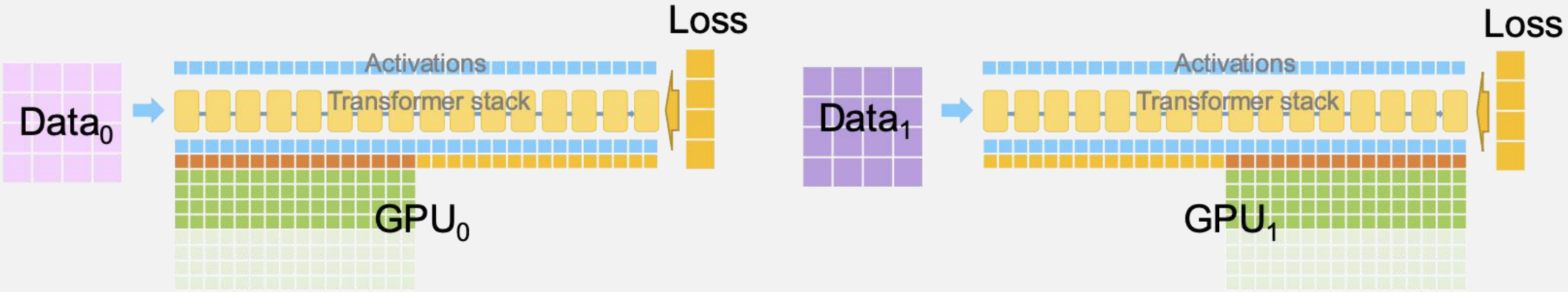
ZeRO 第一阶段：分区优化器状态



- ZeRO Stage 1
- 分区优化器状态跨GPU同步
- 在transformer块间执行前向传播
- 通过反向传播生成FP16梯度，并使用AllReduce进行平均
- 使用ADAM优化器更新FP32权重

Adapted from Minjia Zhang, DeepSpeed Presentation

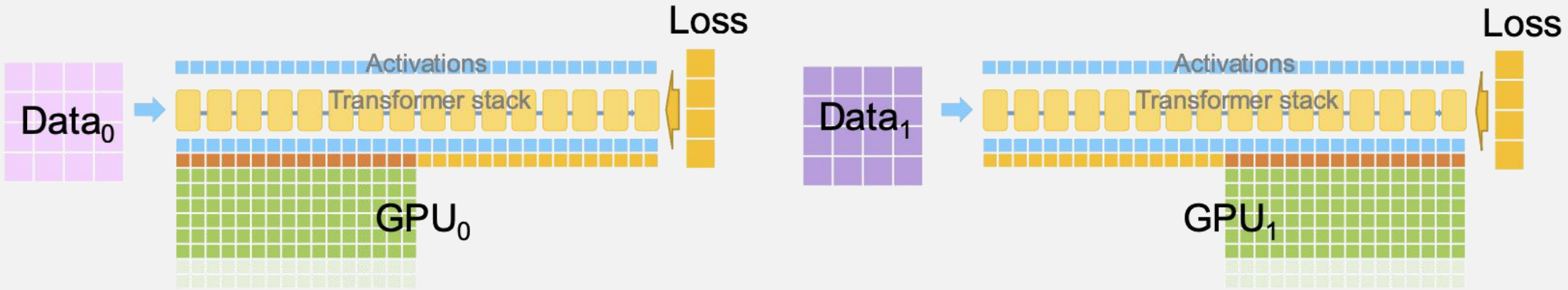
ZeRO 第一阶段：分区优化器状态



- 在transformer块间执行前向传播
- 通过反向传播生成FP16梯度，并使用AllReduce进行平均
- 使用ADAM优化器更新FP32权重

Adapted from Minjia Zhang, DeepSpeed Presentation

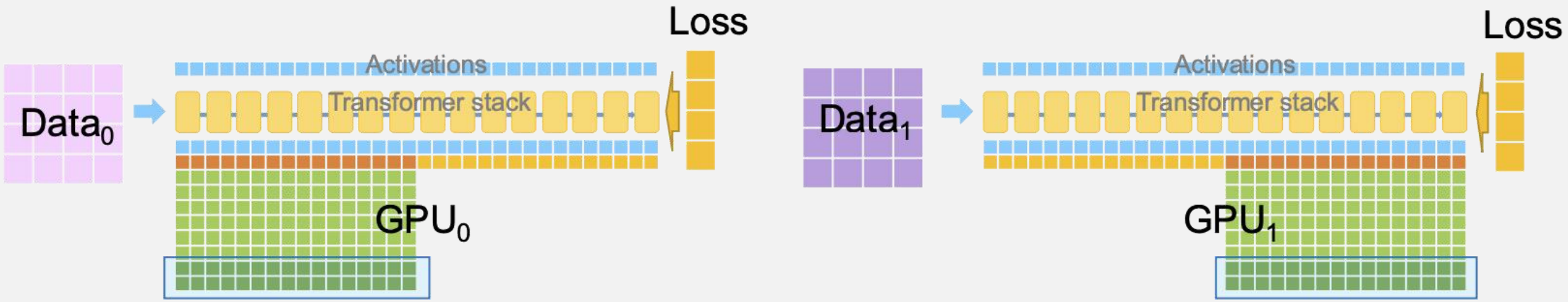
ZeRO 第一阶段：分区优化器状态



- 在transformer块间执行前向传播
- 通过反向传播生成FP16梯度，并使用AllReduce进行平均
- 使用ADAM优化器更新FP32权重

Adapted from Minjia Zhang, DeepSpeed Presentation

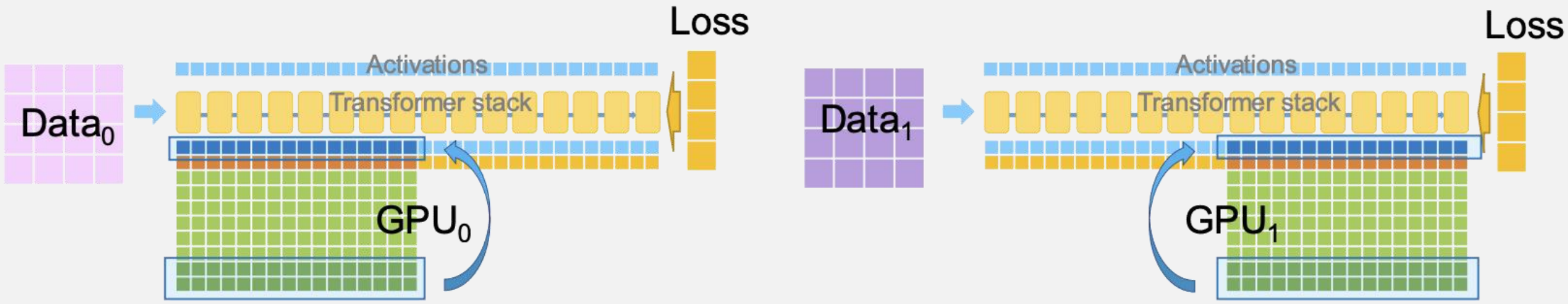
ZeRO 第一阶段：分区优化器状态



- 在transformer块间执行前向传播
- 通过反向传播生成FP16梯度，并使用AllReduce进行平均
- 使用ADAM优化器更新FP32权重

Adapted from Minjia Zhang, DeepSpeed Presentation

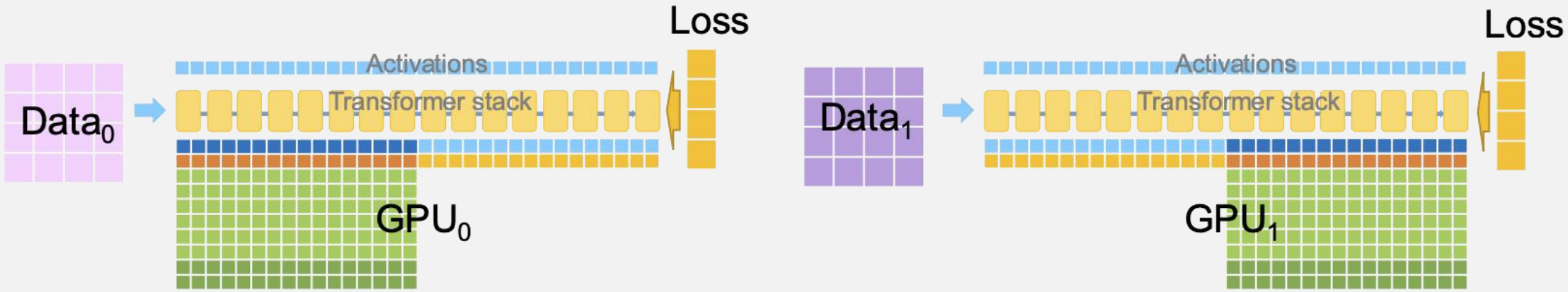
ZeRO 第一阶段：分区优化器状态



- 在transformer块间执行前向传播
- 通过反向传播生成FP16梯度，并使用AllReduce进行平均
- 使用ADAM优化器更新FP32权重
- 更新 FP16 权重

Adapted from Minjia Zhang, DeepSpeed Presentation

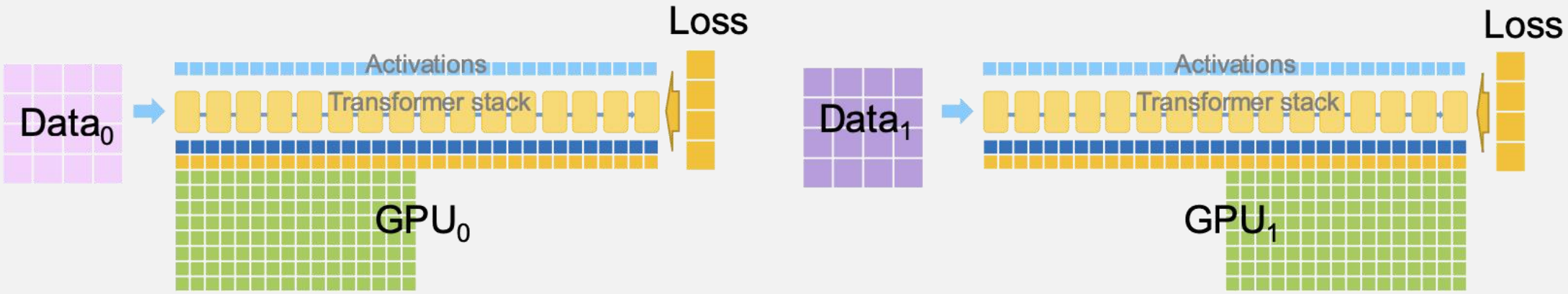
ZeRO 第一阶段：分区优化器状态



- 在transformer块间执行前向传播
- 通过反向传播生成FP16梯度，并使用AllReduce进行平均
- 使用ADAM优化器更新FP32权重
- 更新 FP16 权重
- 对所有 FP16 权重进行聚合以完成迭代

Adapted from Minjia Zhang, DeepSpeed Presentation

ZeRO 第一阶段：分区优化器状态

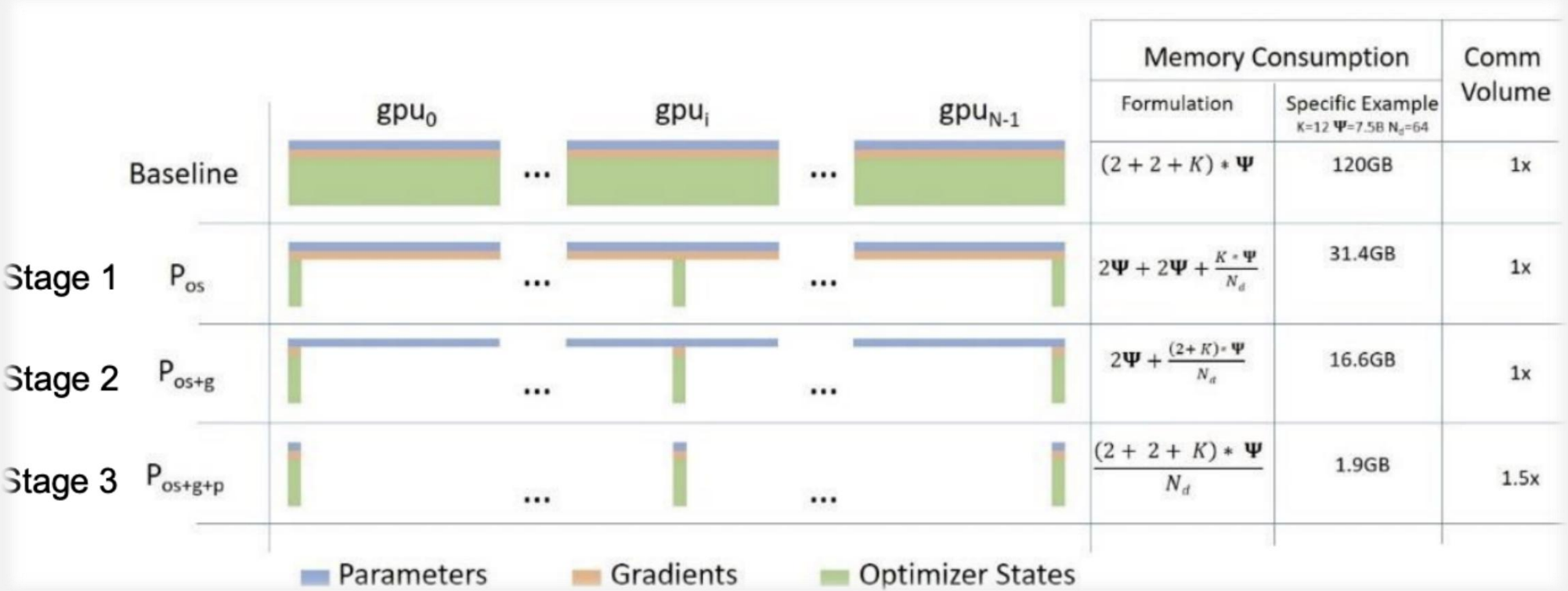


- 在transformer块间执行前向传播
- 通过反向传播生成FP16梯度，并使用AllReduce进行平均
- 使用ADAM优化器更新FP32权重
- 更新 FP16 权重
- 对所有 FP16 权重进行聚合以完成迭代

Adapted from Minjia Zhang, DeepSpeed Presentation

ZeRO：零冗余优化器

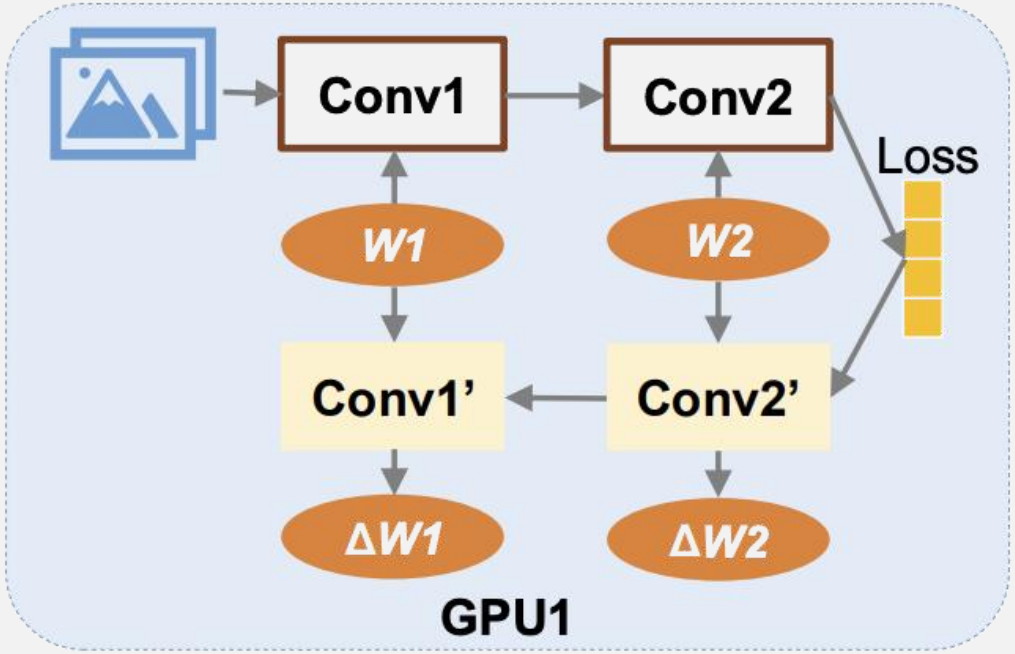
- 渐进式内存节省与通信量
- NLR 17.2B的转向由第一阶段和Megatron提供动力

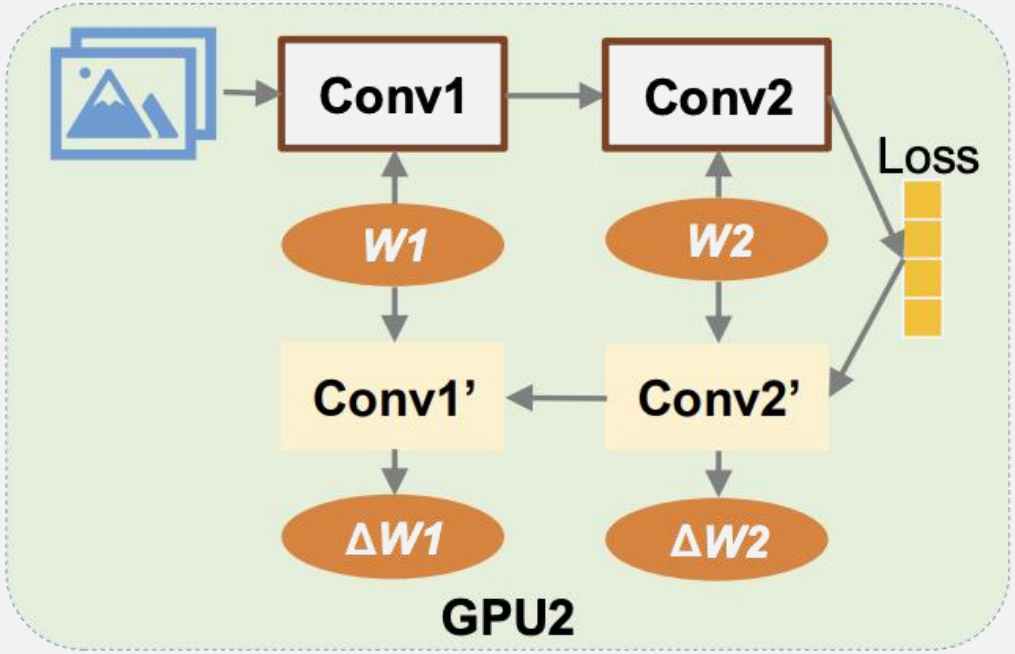


	Memory Consumption		Comm Volume
	Formulation	Specific Example K=12 Ψ=7.5B N_d=64	
Baseline	$(2+2+K)*\Psi$	120GB	1x
Stage 1 P_{os}	$2\Psi+2\Psi+\frac{K*\Psi}{N_d}$	31.4GB	1x
Stage 2 P_{os+g}	$2\Psi+\frac{(2+K)*\Psi}{N_d}$	16.6GB	1x
Stage 3 P_{os+g+p}	$\frac{(2+2+K)*\Psi}{N_d}$	1.9GB	1.5x

ZeRO 第三阶段：参数划分

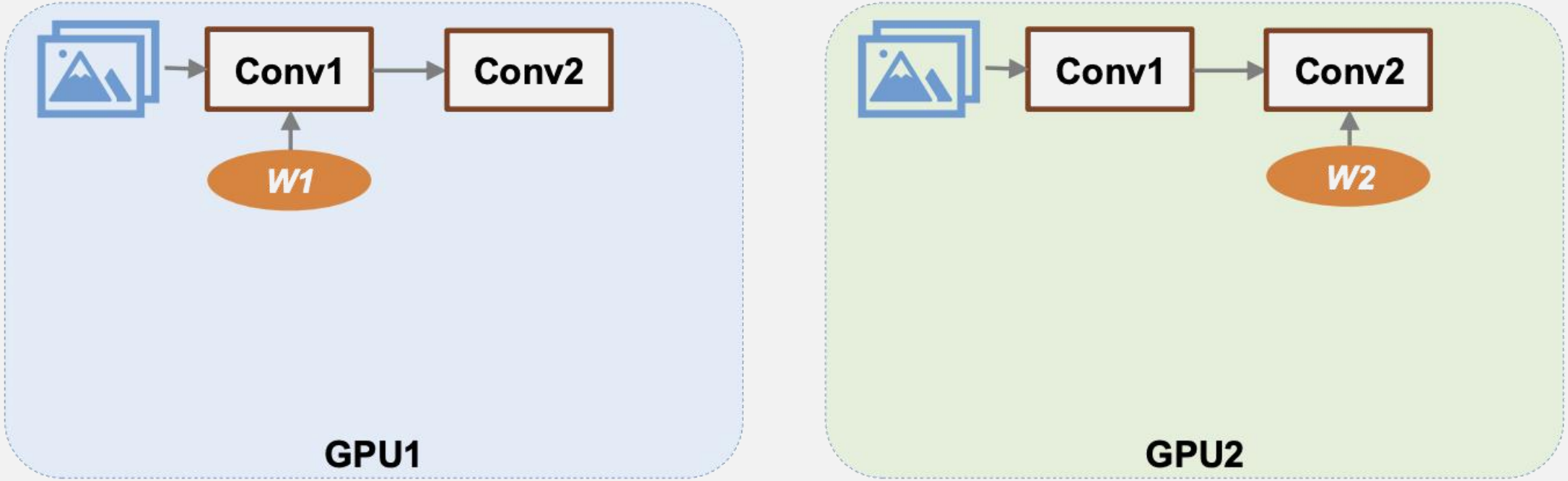
- 在数据并行训练中，所有GPU在训练期间均保留所有参数





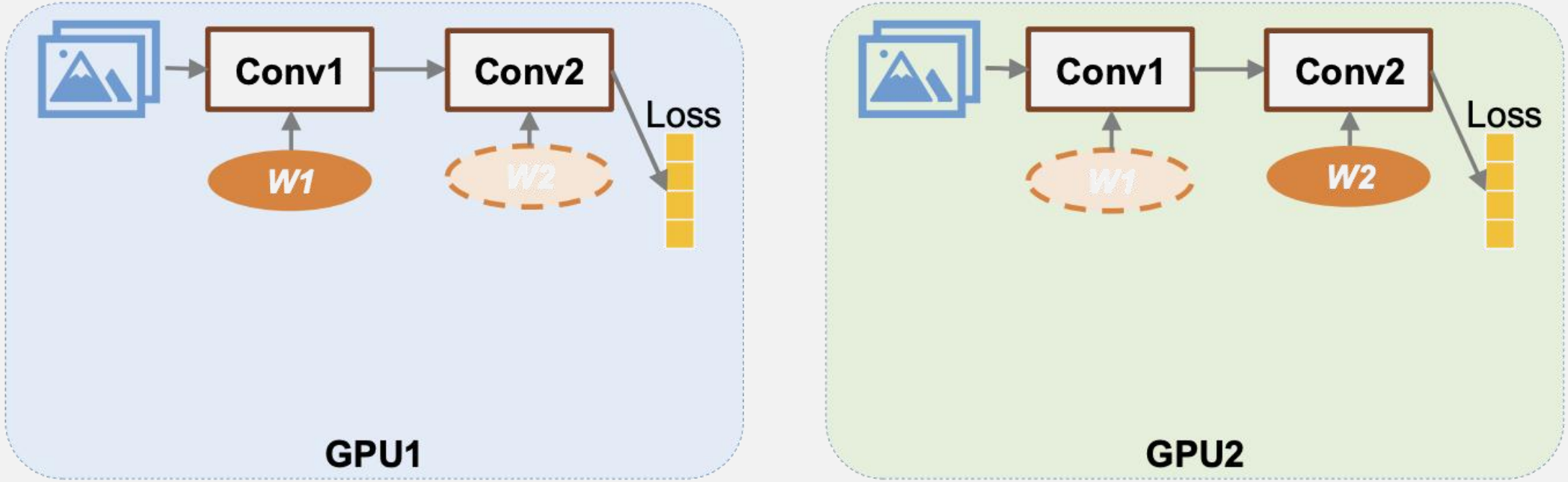
ZeRO 第三阶段：参数划分

- 在ZeRO中，模型参数被划分到多个GPU上



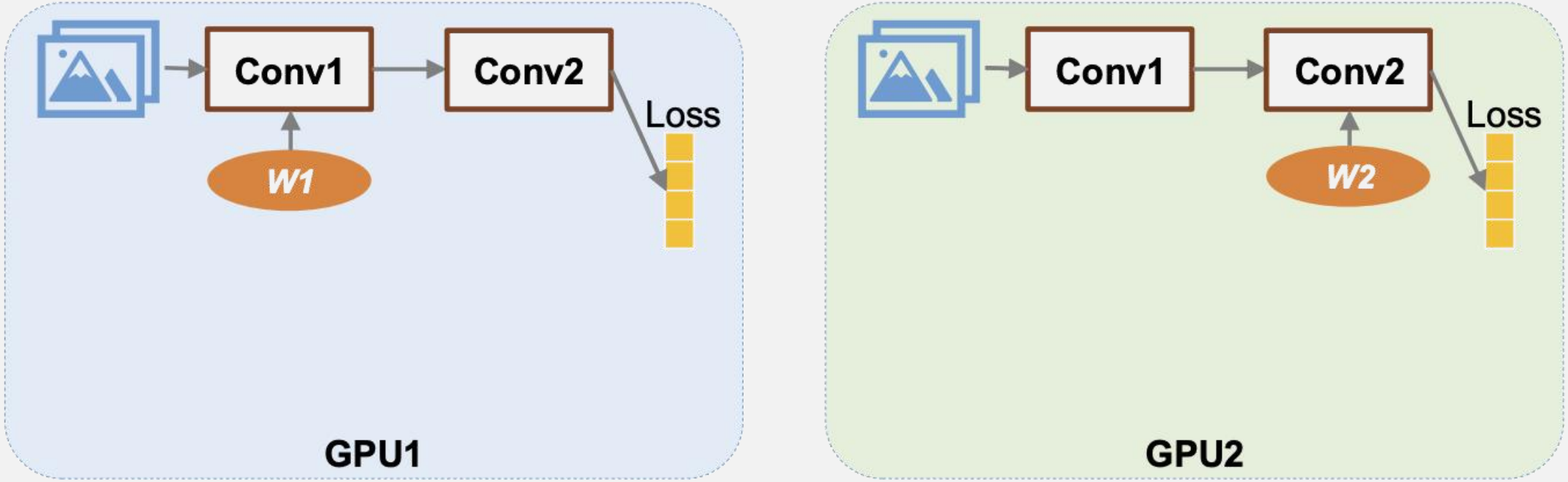
ZeRO 第三阶段：参数划分

- 在ZeRO中，模型参数被划分到多个GPU上
- GPU 在前向传播过程中广播其参数



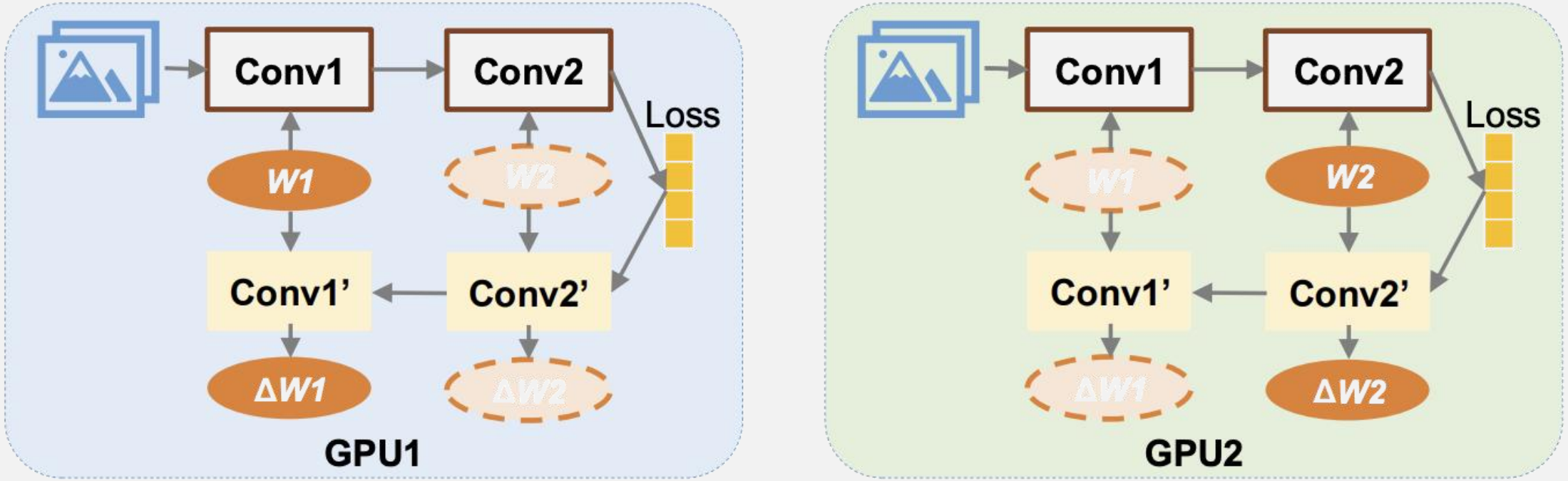
ZeRO 第三阶段：参数划分

- 在ZeRO中，模型参数被划分到多个GPU上
- 参数在使用后立即被丢弃



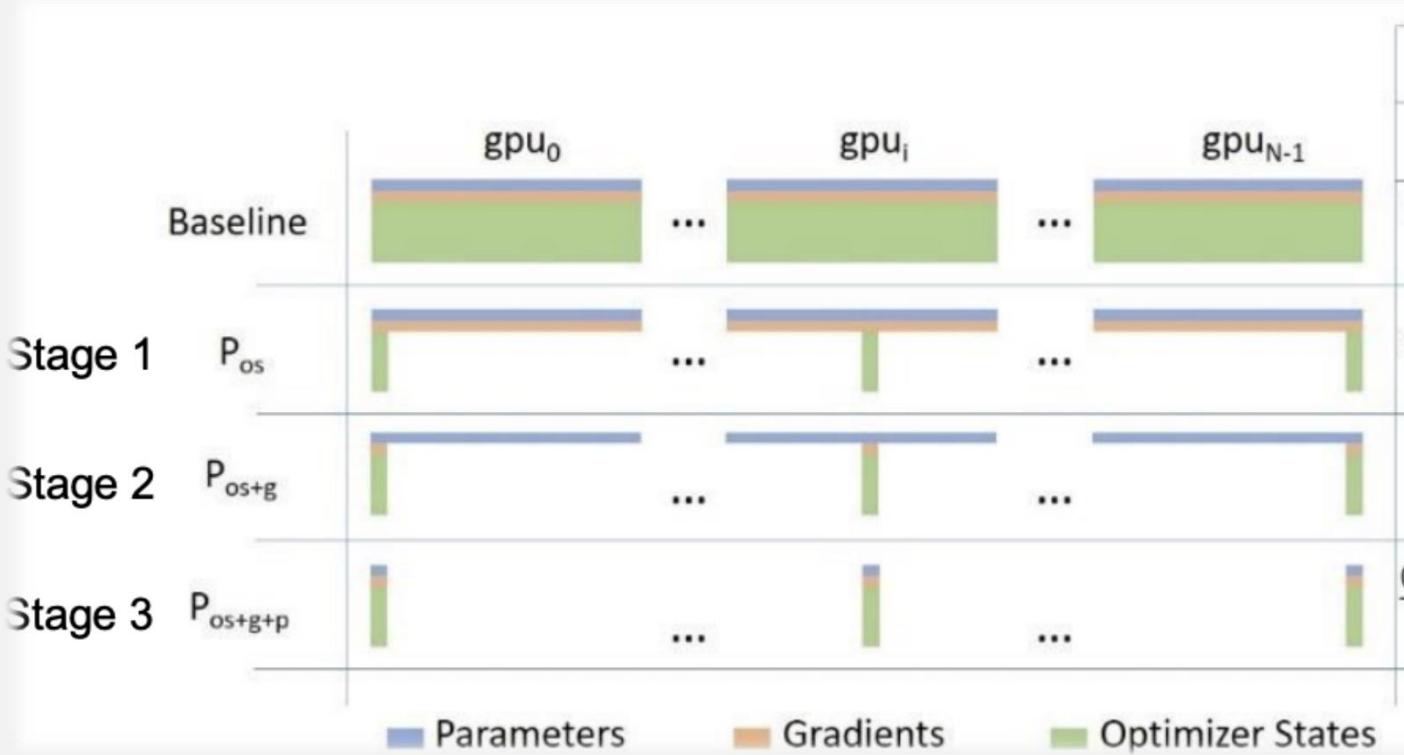
ZeRO 第三阶段：参数划分

- 在ZeRO中，模型参数被划分到多个GPU上
- GPU在反向传播过程中再次广播其参数



ZeRO：零冗余优化器

- ZeRO包含三个不同阶段
- 逐步提升内存节省与通信量



	Memory Consumption		Comm Volume
	Formulation	Specific Example K=12 Ψ=7.5B N_d=64	
Baseline	$(2+2+K)*\Psi$	120GB	1x
Stage 1 P_{os}	$2\Psi+2\Psi+\frac{K*\Psi}{N_d}$	31.4GB	1x
Stage 2 P_{os+g}	$2\Psi+\frac{(2+K)*\Psi}{N_d}$	16.6GB	1x
Stage 3 P_{os+g+p}	$\frac{(2+2+K)*\Psi}{N_d}$	1.9GB	1.5x

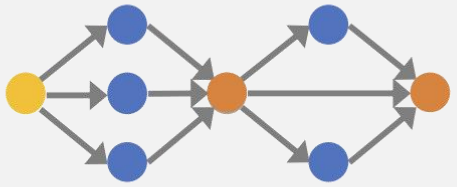
Summary

数据并行训练

- 参数服务器
- 环形AllReduce
- 树形AllReduce
- 蝴蝶AllReduce

- ZeRO：零冗余优化器

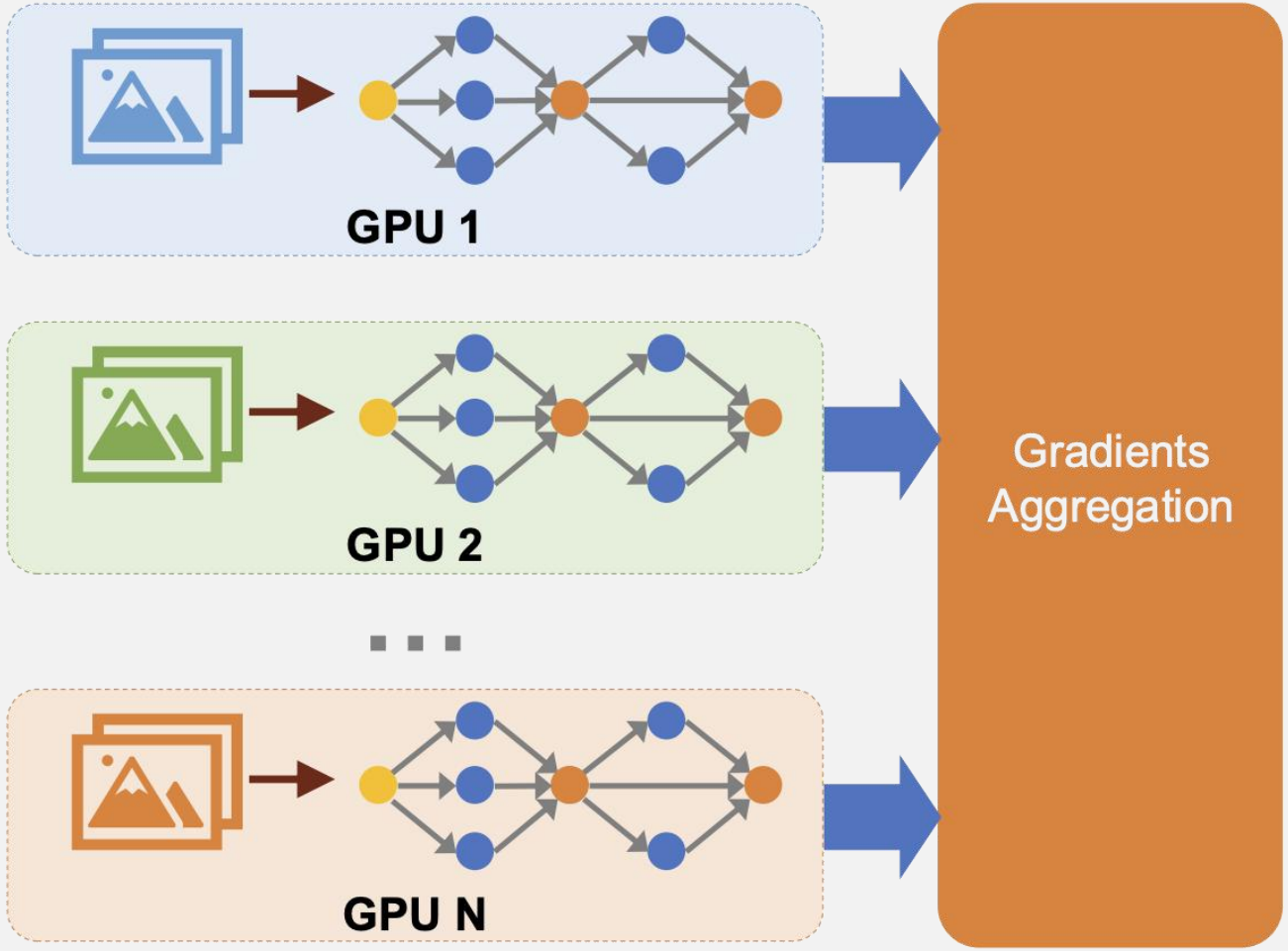
回顾：数据并行性



ML Model

Training Dataset

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



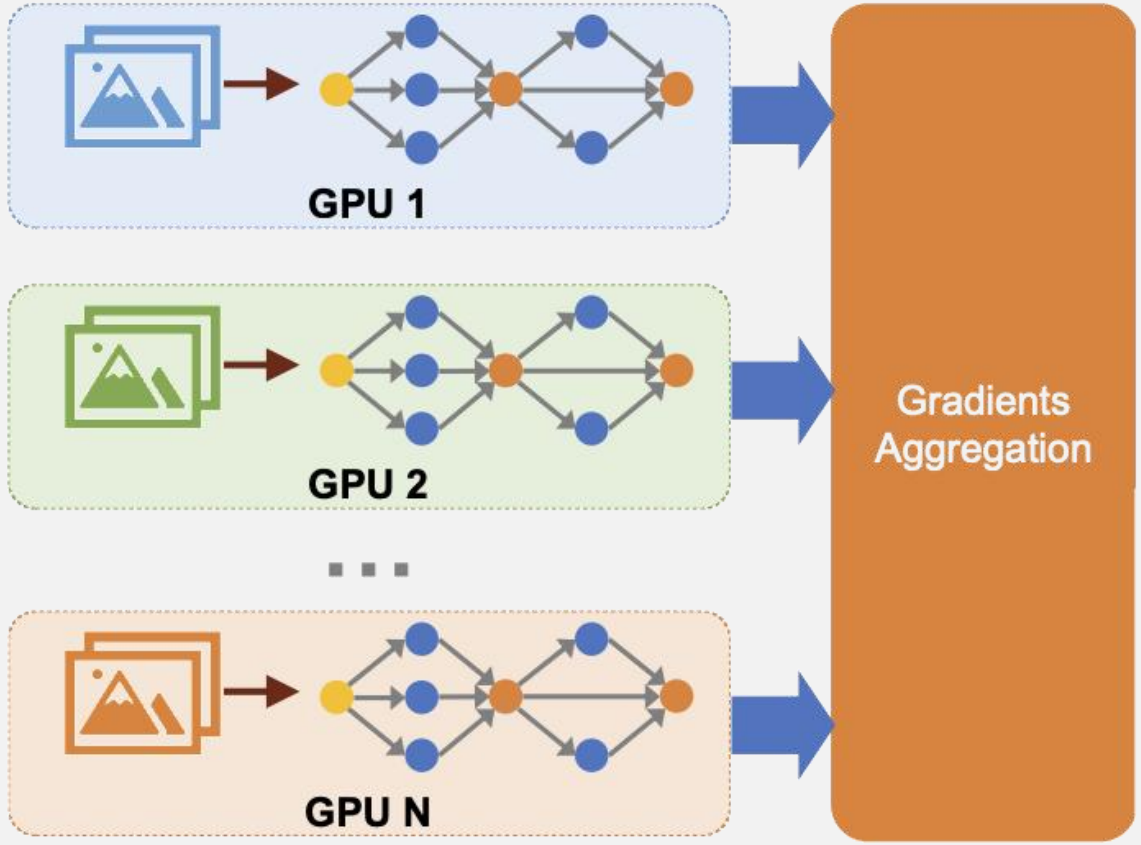
1.将训练数据划分为批次

2.在GPU上计算每批次的梯度

3.跨GPU聚合梯度

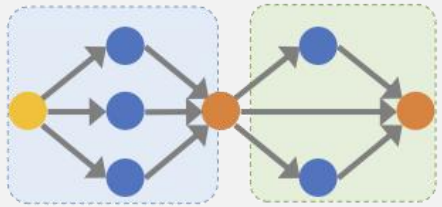
回顾：数据并行性问题

- 每块GPU都会保存整个模型的副本
- 无法训练超过GPU设备内存容量的大型模型



模型并行性

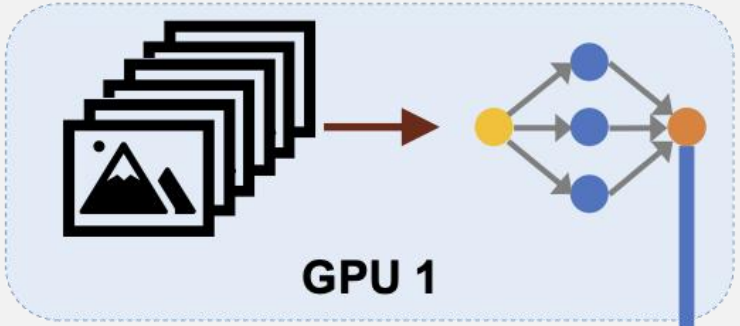
- 将模型拆分为多个子图，并将其分配给不同的设备

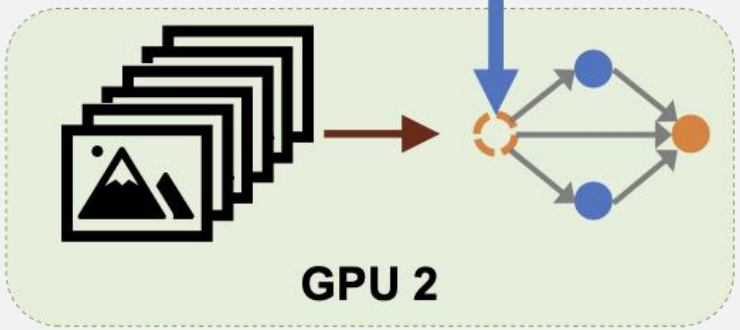


ML Model

Training Dataset

Model Parallelism





在设备之间传输中间结果

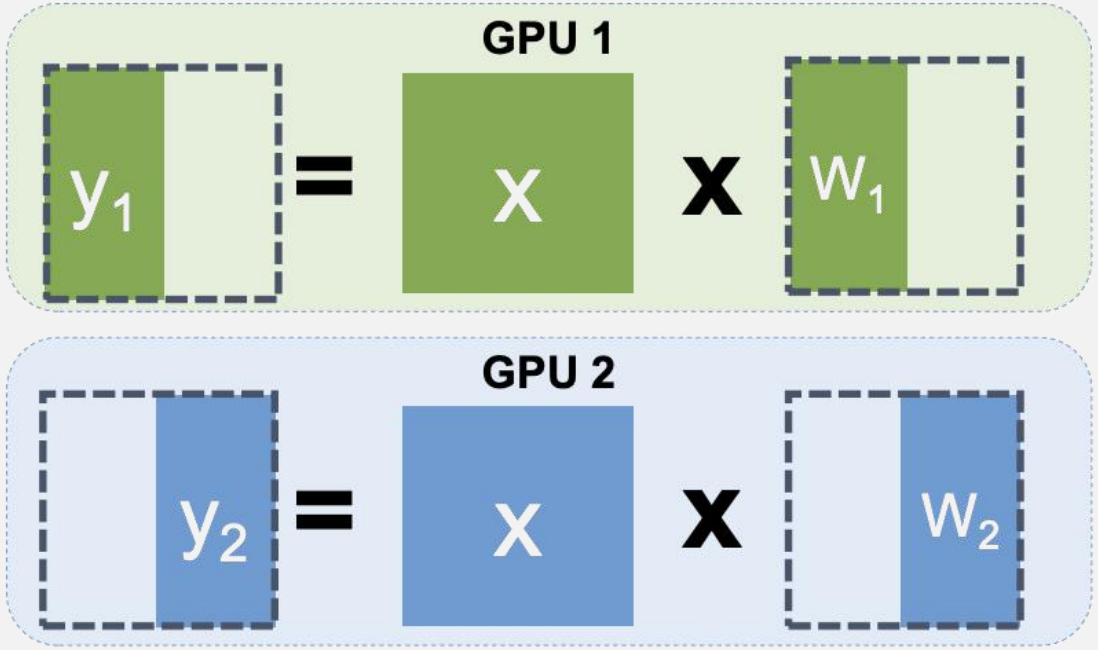
$$w_i := w_i - \gamma \nabla L(w_i) = w_i - \frac{\gamma}{n} \sum_{j=1}^{n} \nabla L_j(w_i)$$

如何实现DNN训练的并行化？

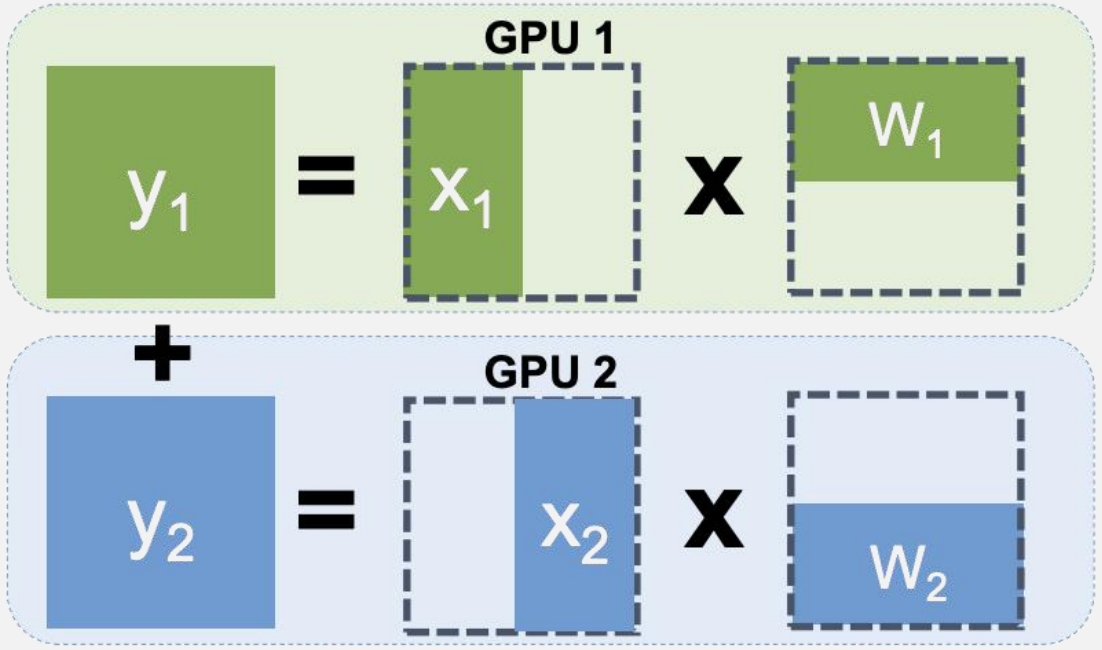
- 数据并行
- 模型并行
  - 张量模型并行
  - 流水线模型并行

张量模型并行性

- 层内分区参数/梯度



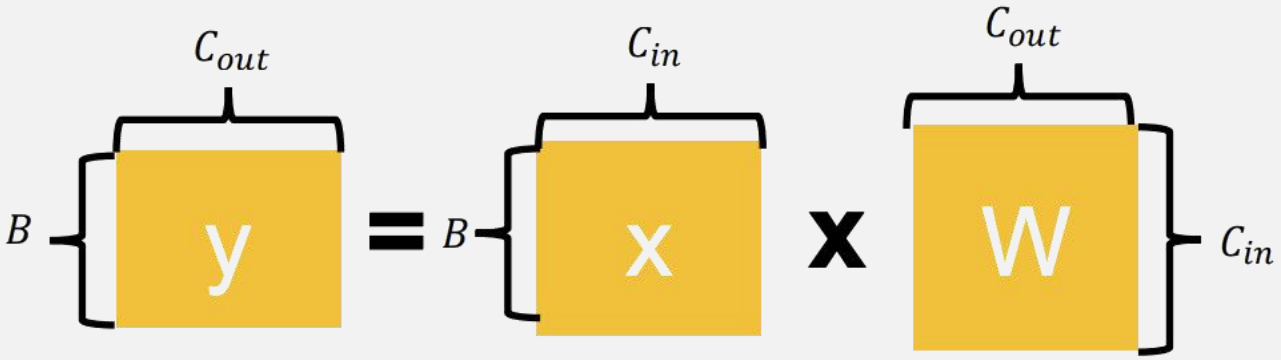
张量模型并行（输出分区）

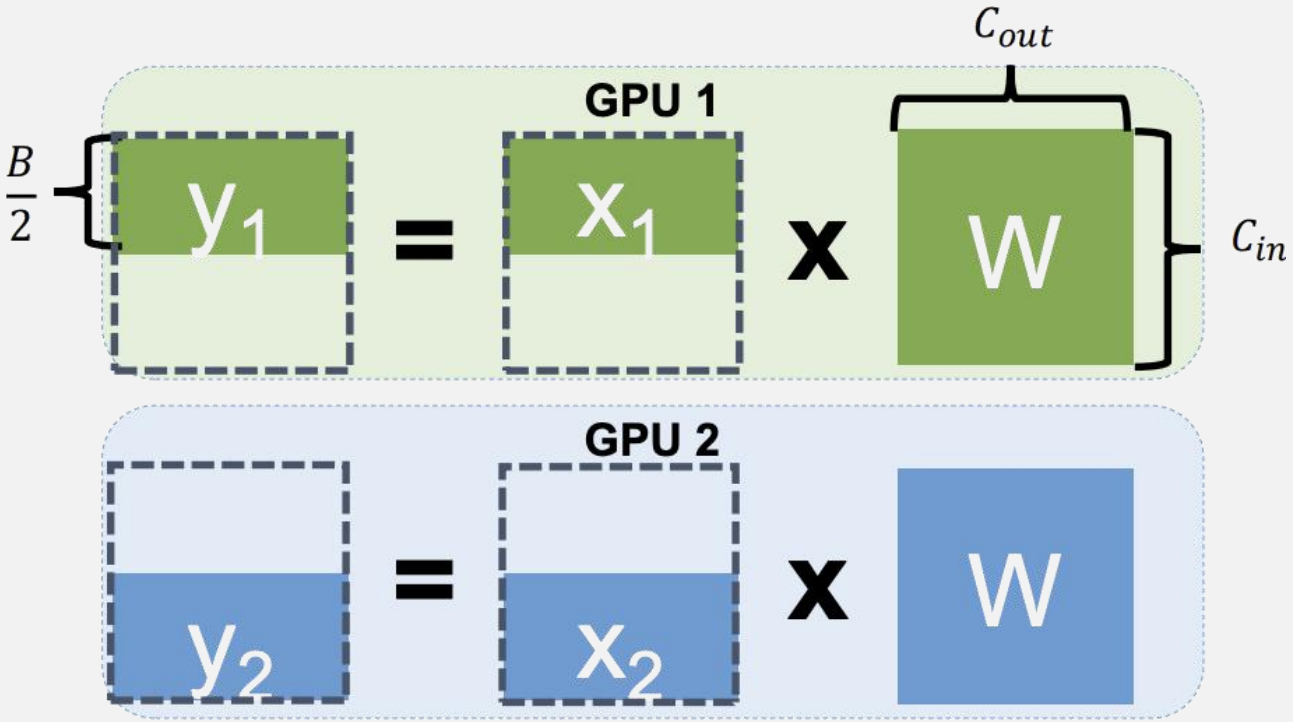


张量模型并行（减少输出）

$y = y_1 + y_2$

数据与张量模型并行性的比较



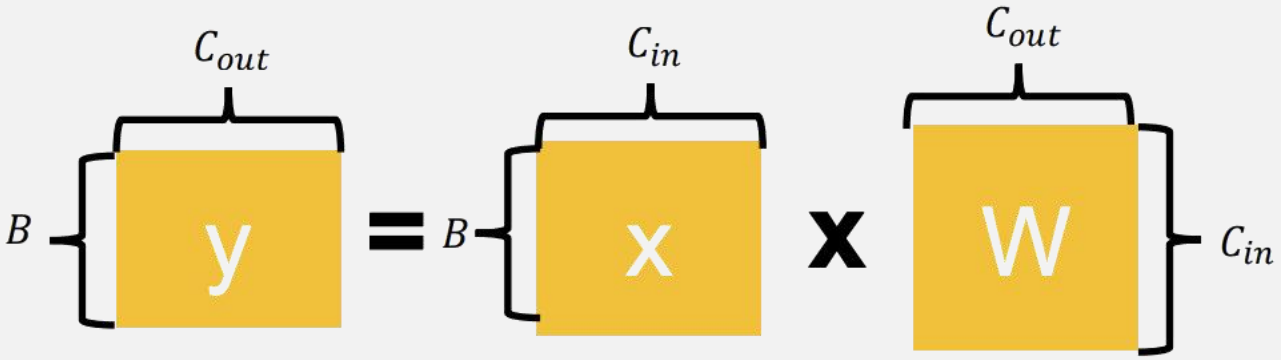


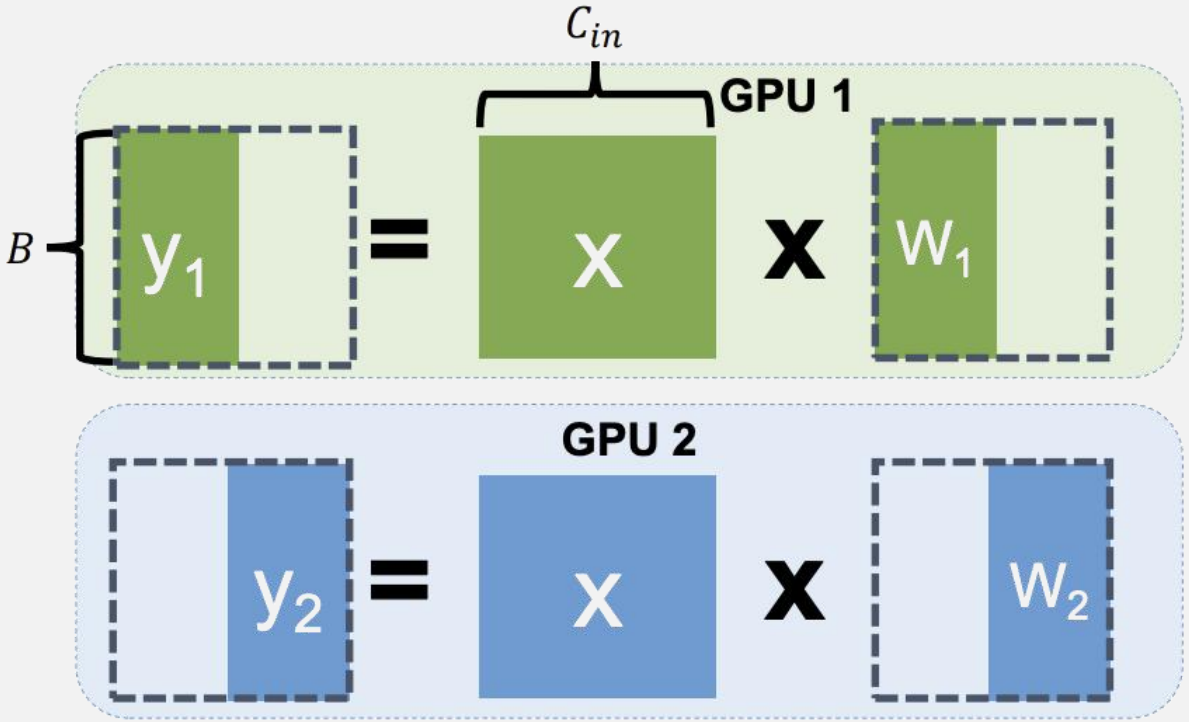
数据并行性　　$y = Wx$

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

数据并行通信开销

数据与张量模型并行性的比较



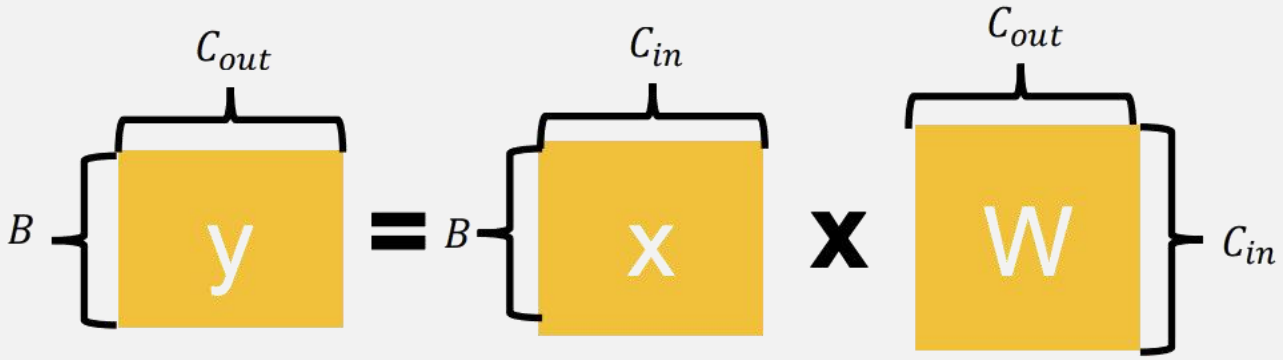


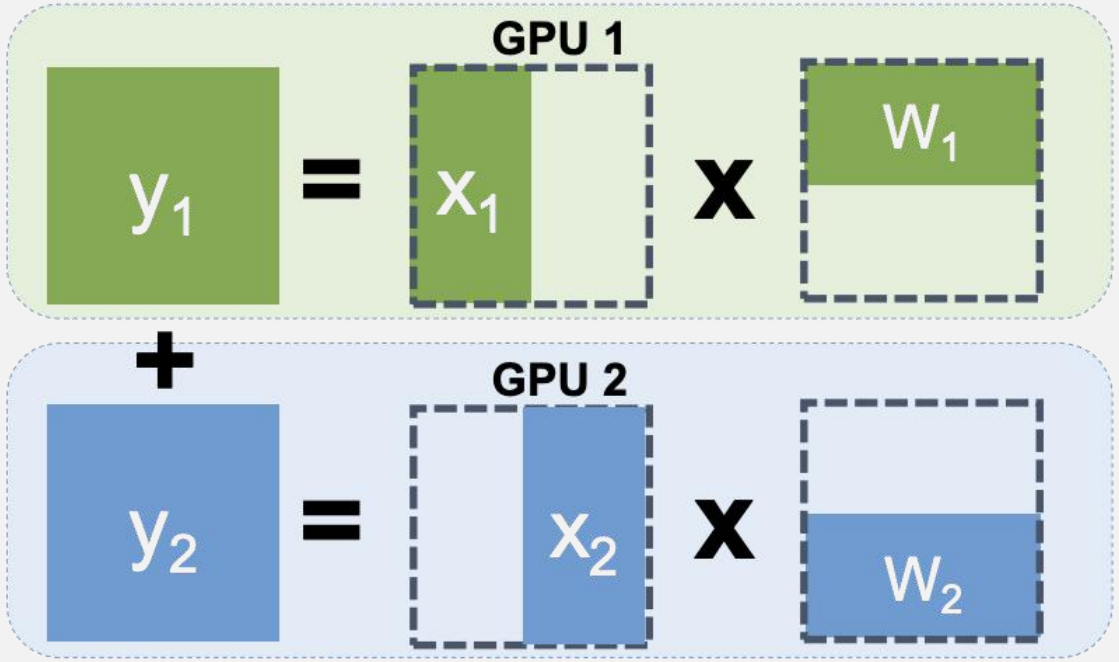
张量模型并行（输出分区）

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

数据并行通信开销

数据与张量模型并行性的比较





张量模型并行（减少输出）

$y = y_1 + y_2$

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

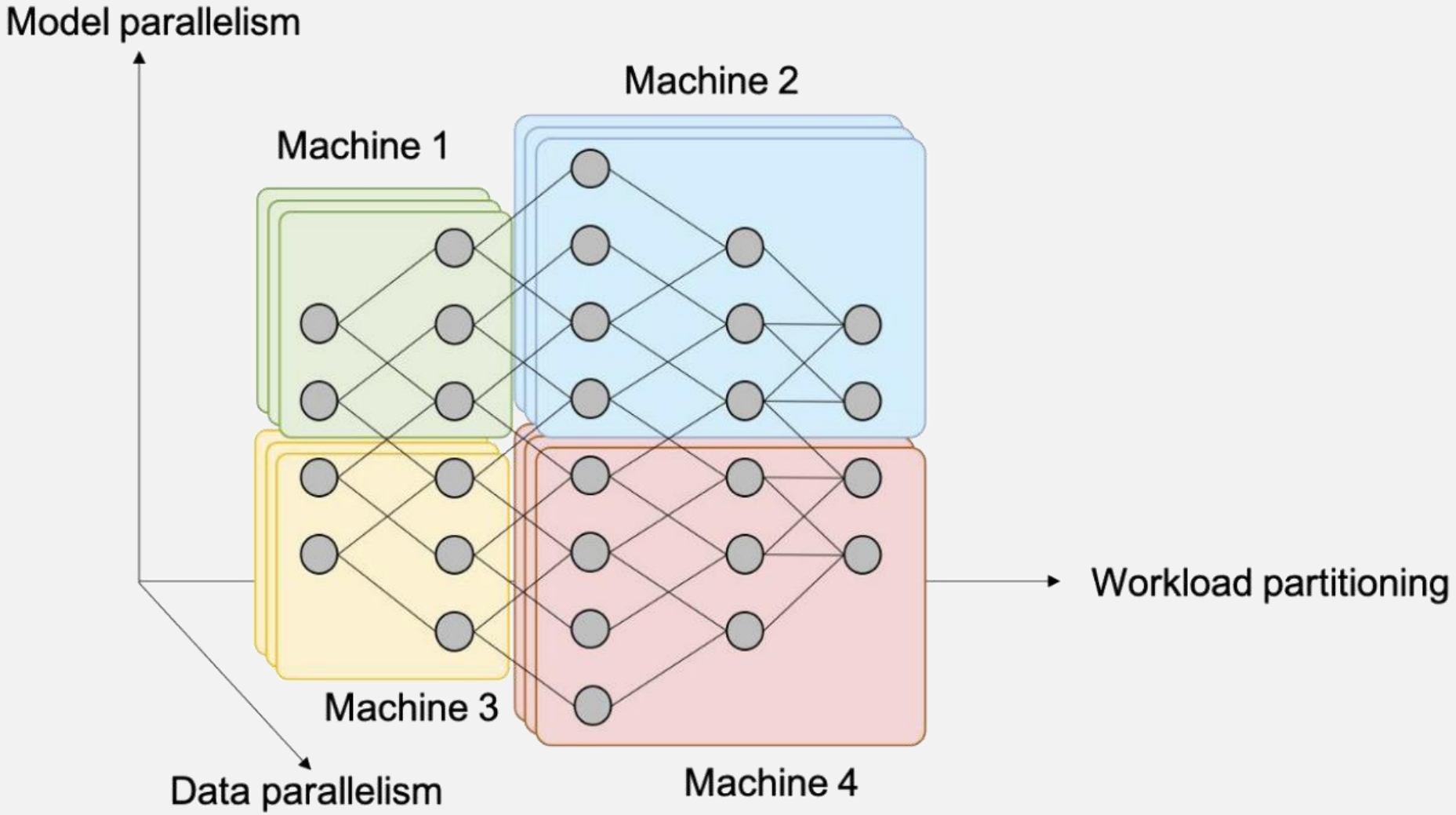
张量模型并行通信开销

数据与张量模型并行性的比较

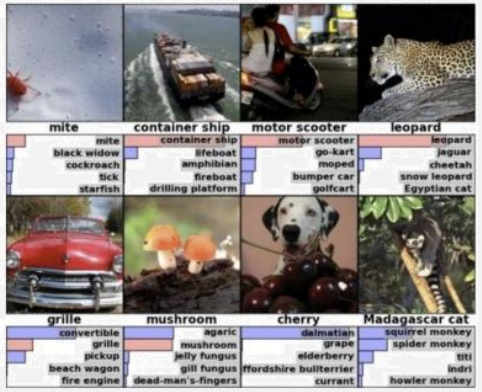
- 数据并行：$O(Cat * Cin)$
- 张量模型并行（输出分区）：$O(B * Cin)$
- 张量模型并行（输出归约）：$O(B * Cat)$
- 最佳策略取决于模型和底层机器

- 最佳策略取决于模型和底层机器

结合数据并行与模型并行



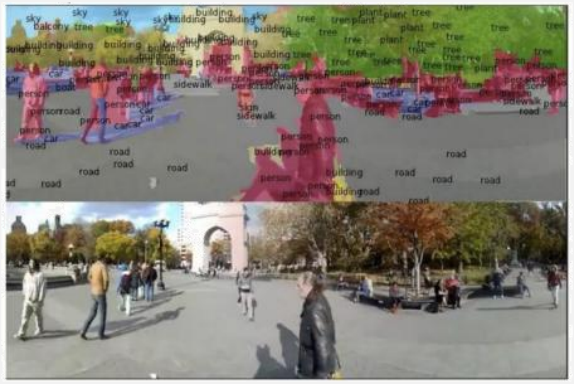
示例：卷积神经网络



分类

检索

检测



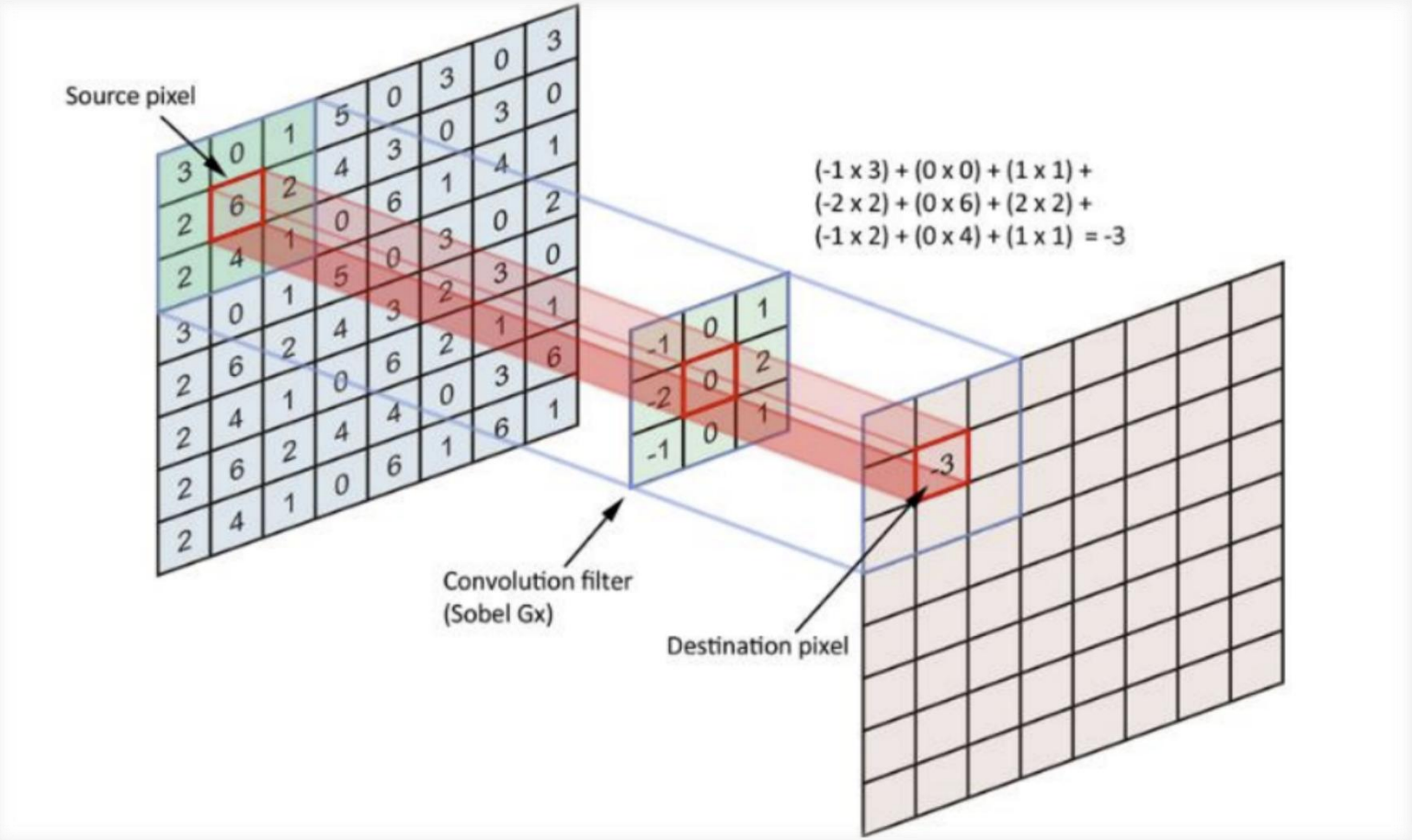
分割

自动驾驶

合成

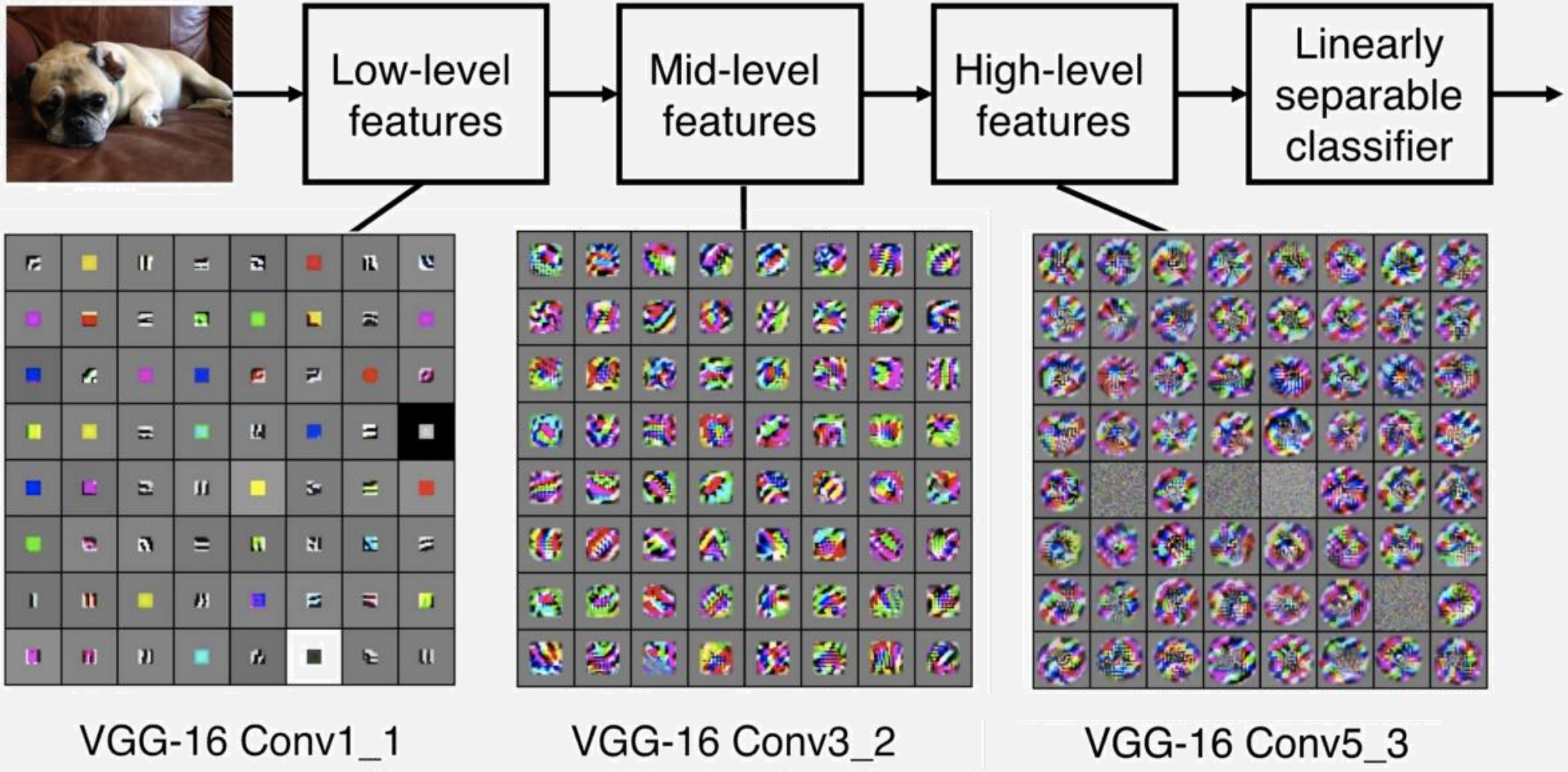
卷积

- 对滤波器与图像进行卷积：在图像上进行空间滑动，并计算点积



CNNs

- 一系列卷积层，其间穿插着池化、归一化及激活函数



卷积神经网络的并行化

- 卷积层
  - 占计算量的90-95%
  - 占参数量的5%
  - 中间激活值非常大

数据并行化

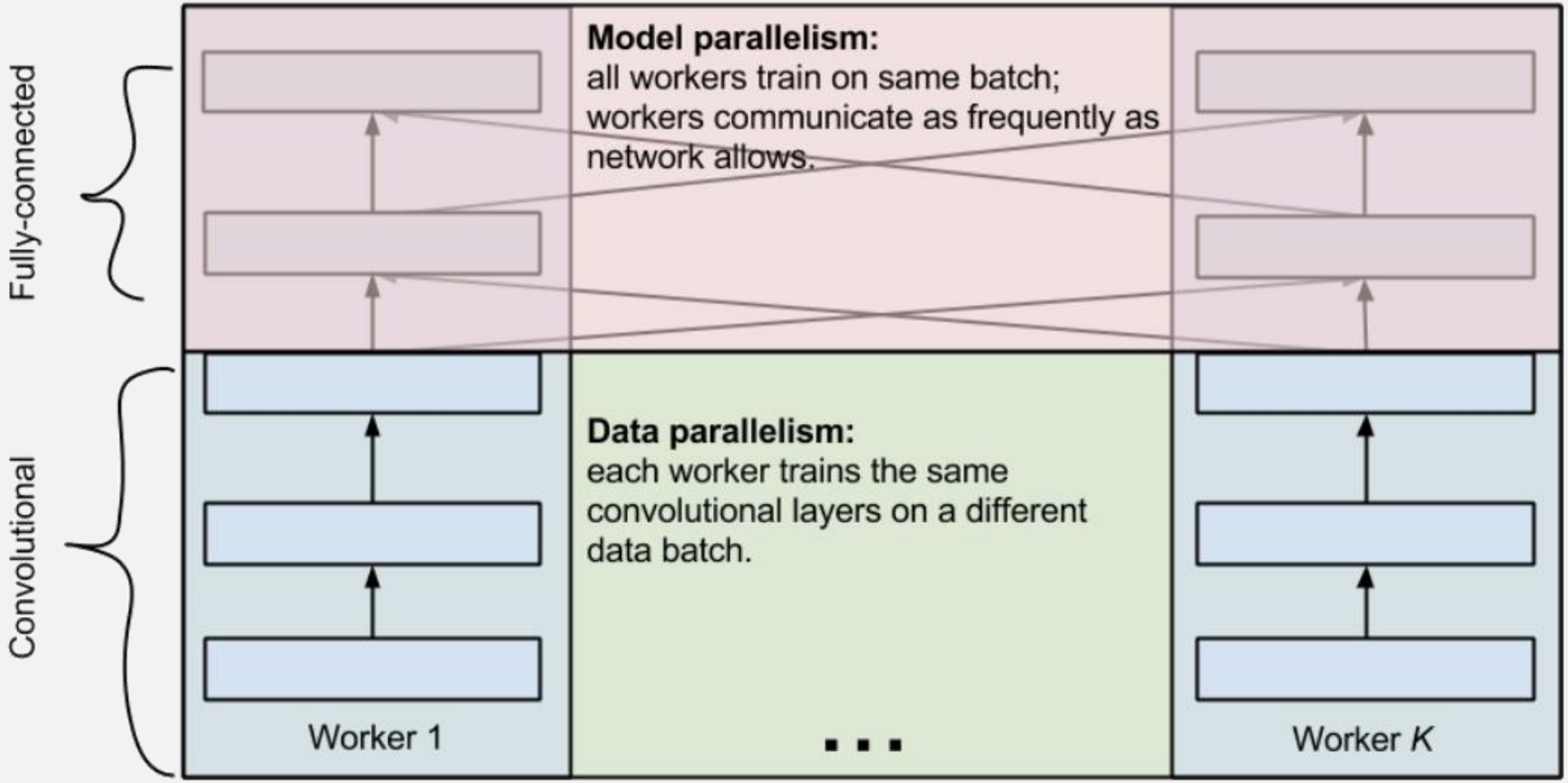
- 全连接层
  - 占计算量的5-10%
  - 占参数量的95%
  - 中间激活值较小

张量模型并行性

- 讨论：如何实现卷积神经网络的并行化?

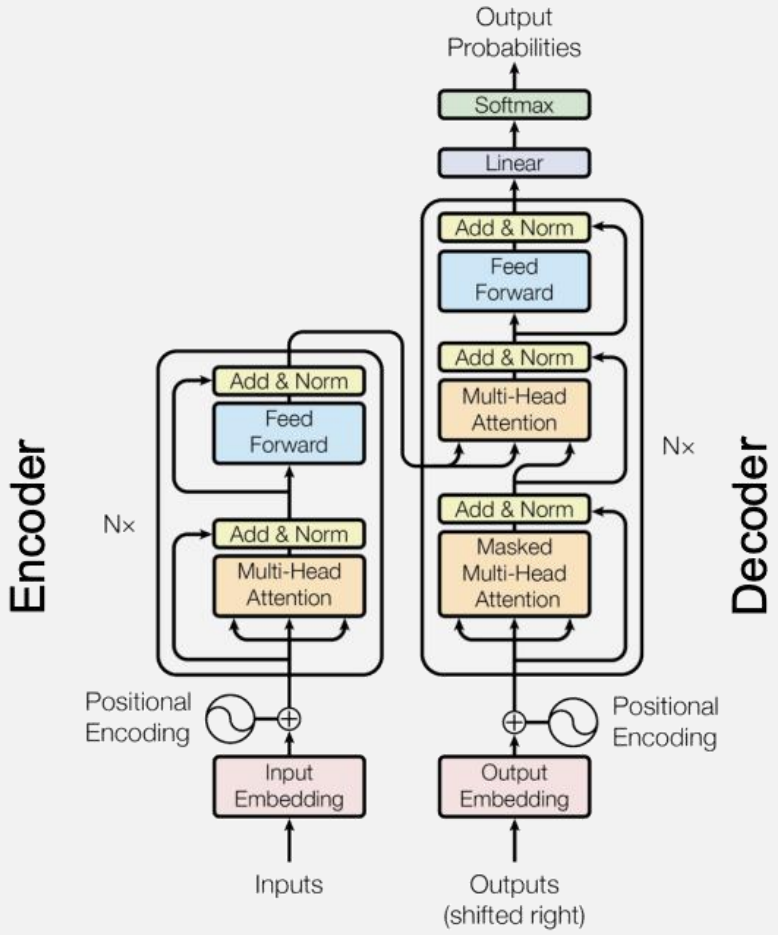
卷积神经网络的并行化

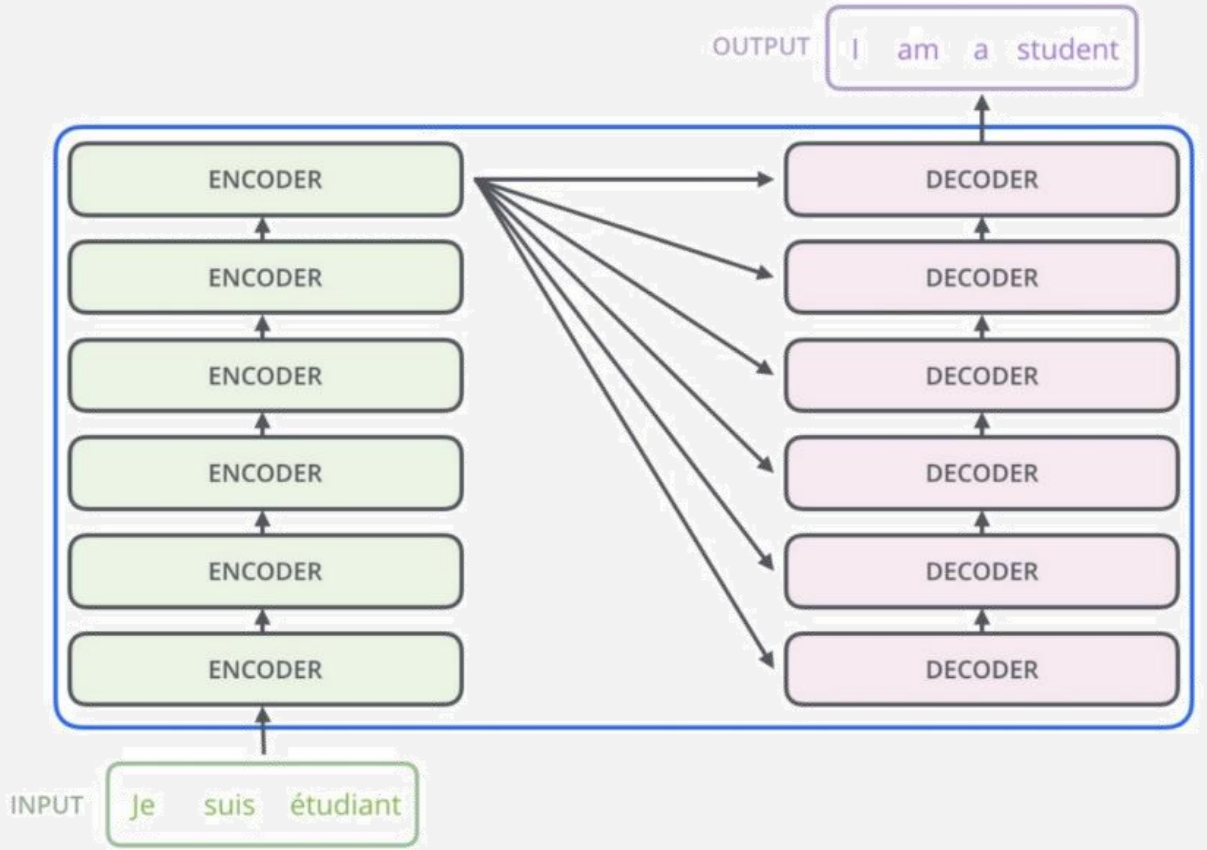
- 卷积层的数据并行性
- 全连接层的张量模型并行性



示例：Transformer的并行化

- Transformer：用于语言理解的注意力机制



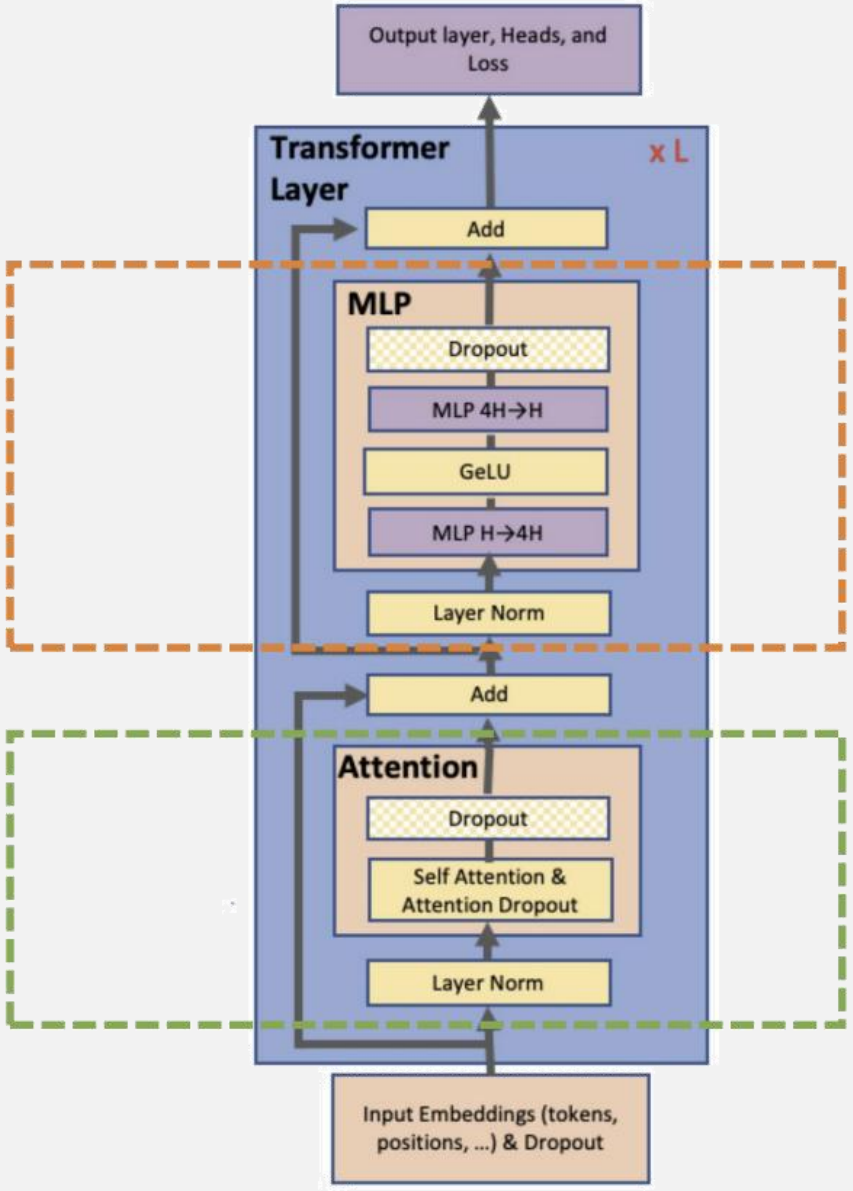


Ashish Vaswani et. al. Attention is all you need.

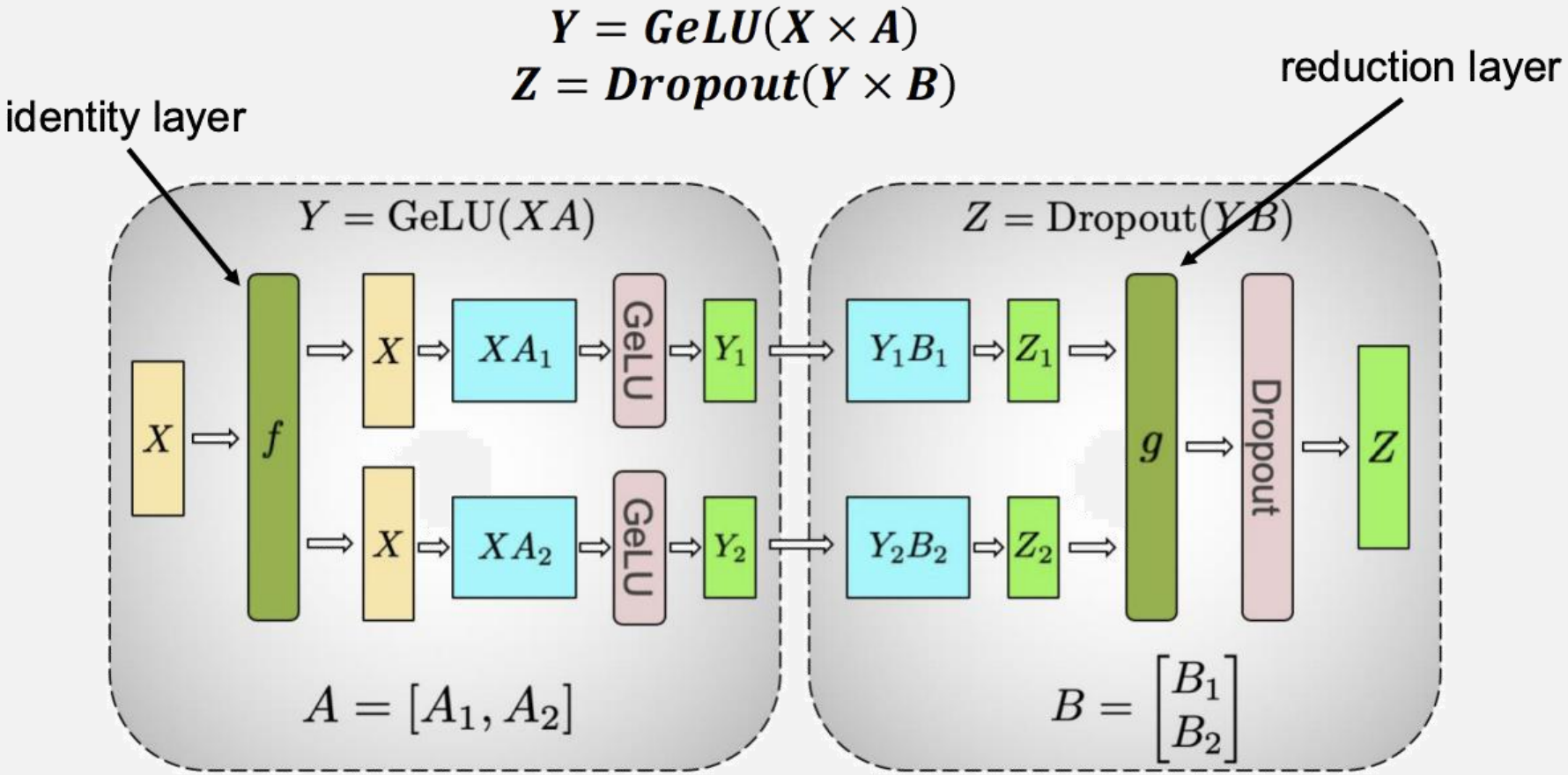
单层Transformer

全连接层

自注意力层



在Transformer模型中并行化全连接层

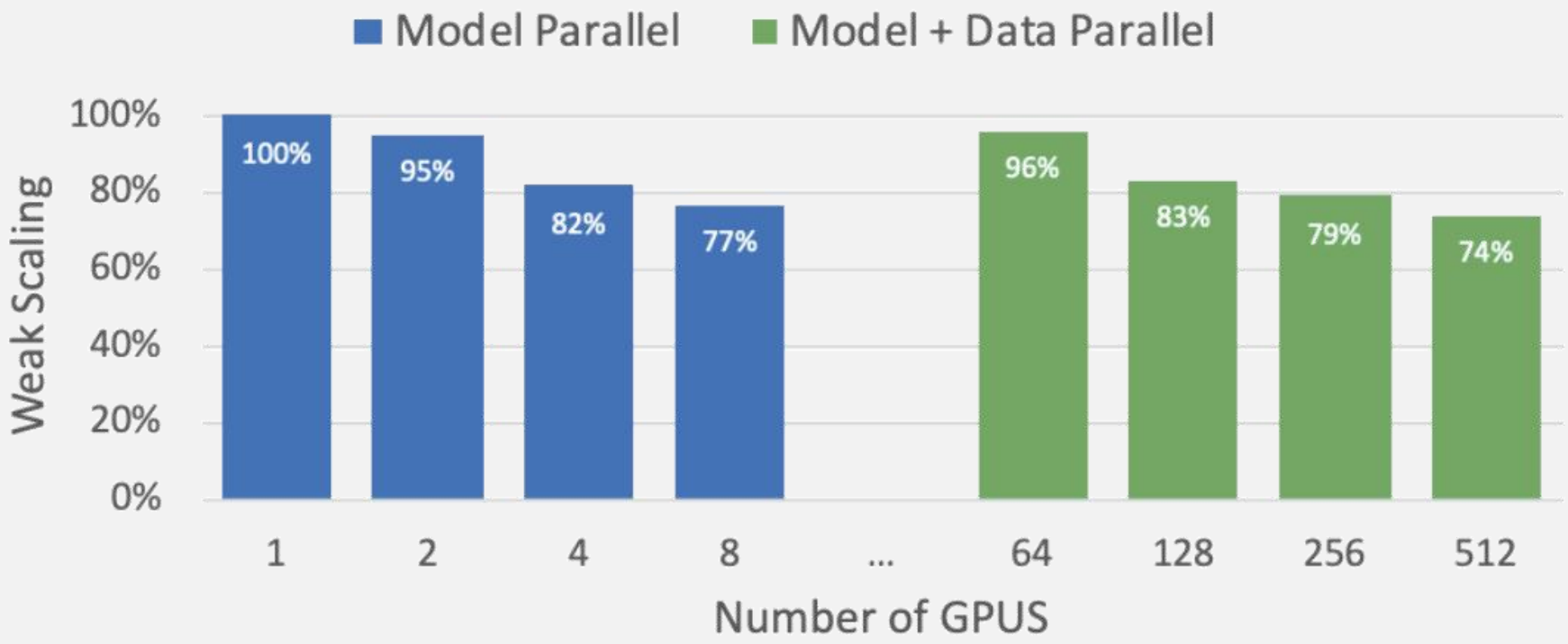


张量模型并行
（输出分区）

张量模型并行
（减少输出）

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

Transformer的并行化



通过结合数据并行和模型并行实现扩展至512个GPU

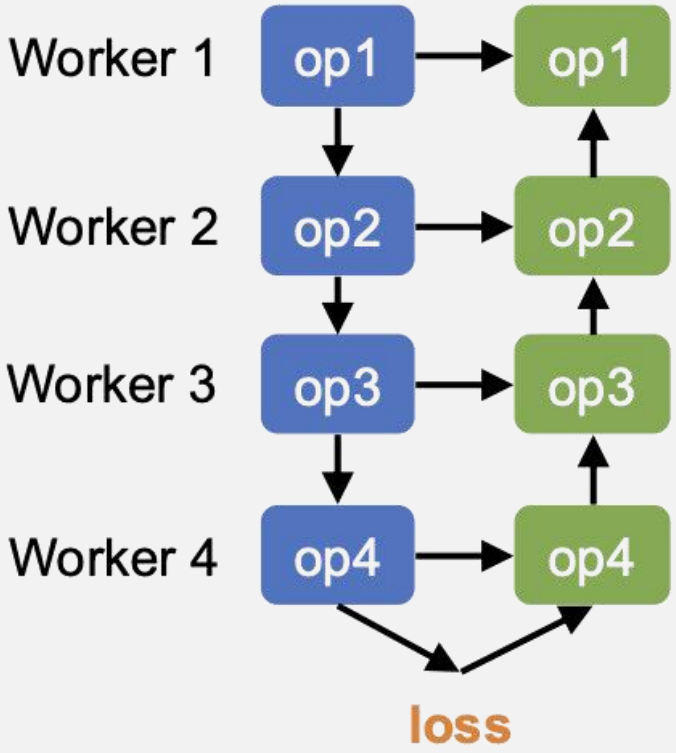
Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism.

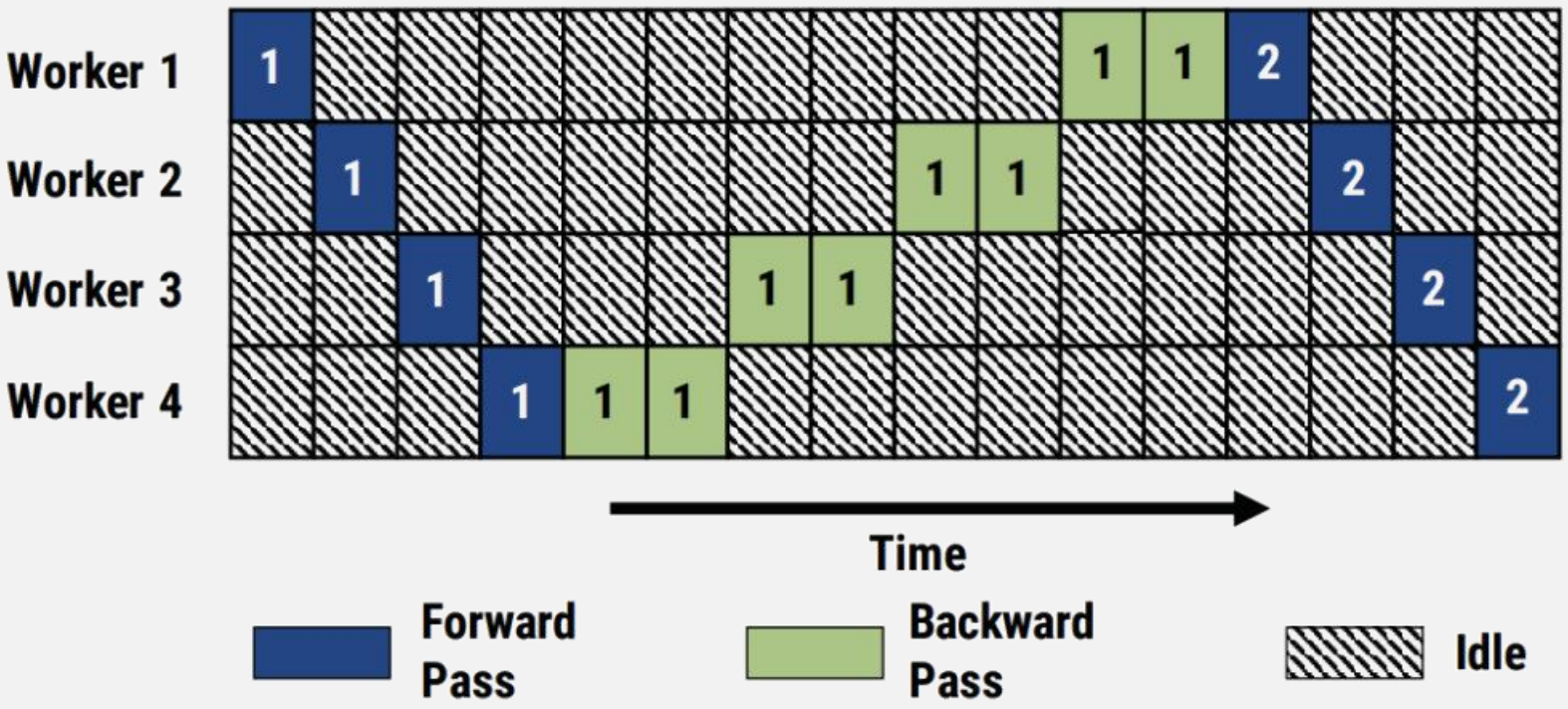
如何实现DNN训练的并行化？

- 数据并行
- 模型并行
  - 张量模型并行
  - 流水线模型并行

模型并行性问题

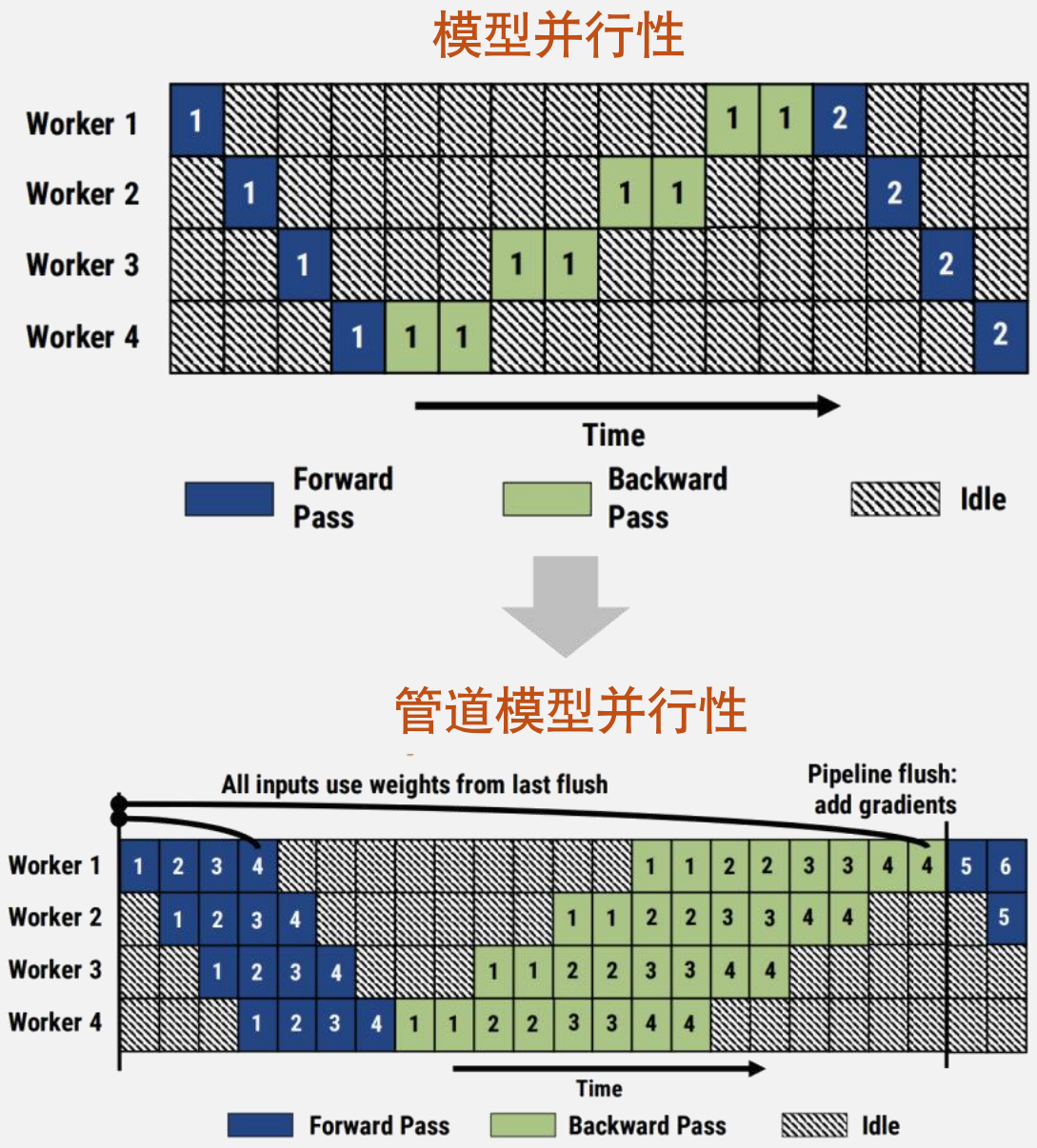
- 计算资源利用率不足
- 因资源利用率导致整体吞吐量偏低





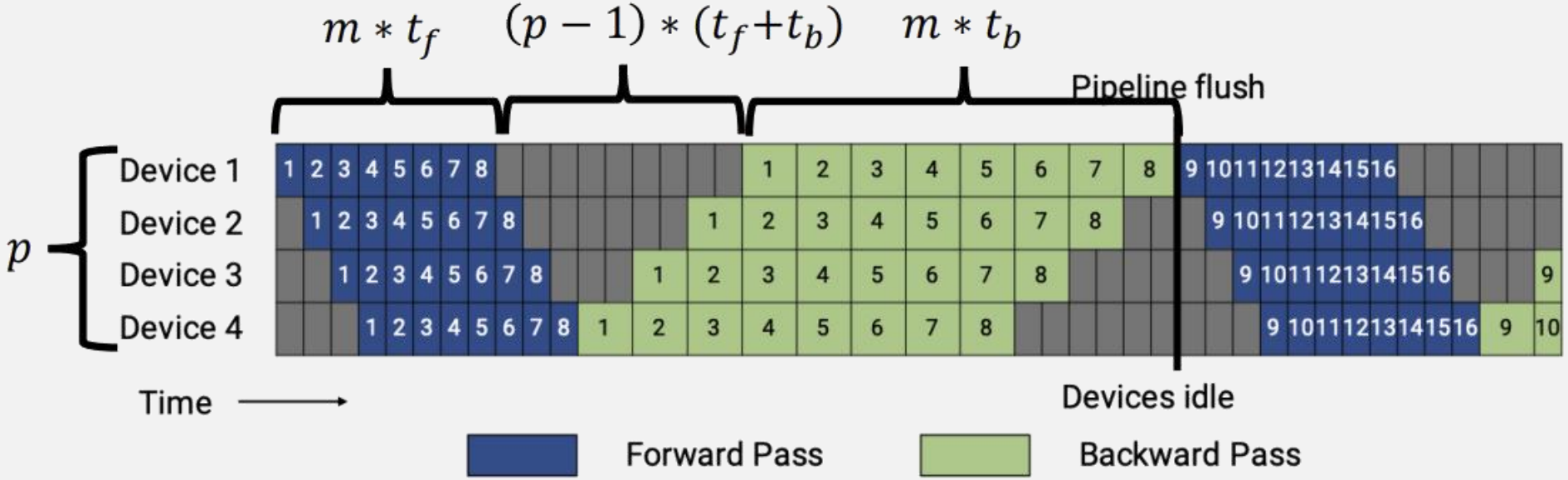
管道模型并行性

- 小批量：每次迭代处理的样本数量
- 将小批量划分为多个微批次
- 在微批次间串行化正向与反向计算



管道模型并行性：设备利用率

- m 微批次中的子批次数量
- p 管道处理阶段数
- 所有阶段处理一个正向（反向）微批次所需时间为 t_f/t_b

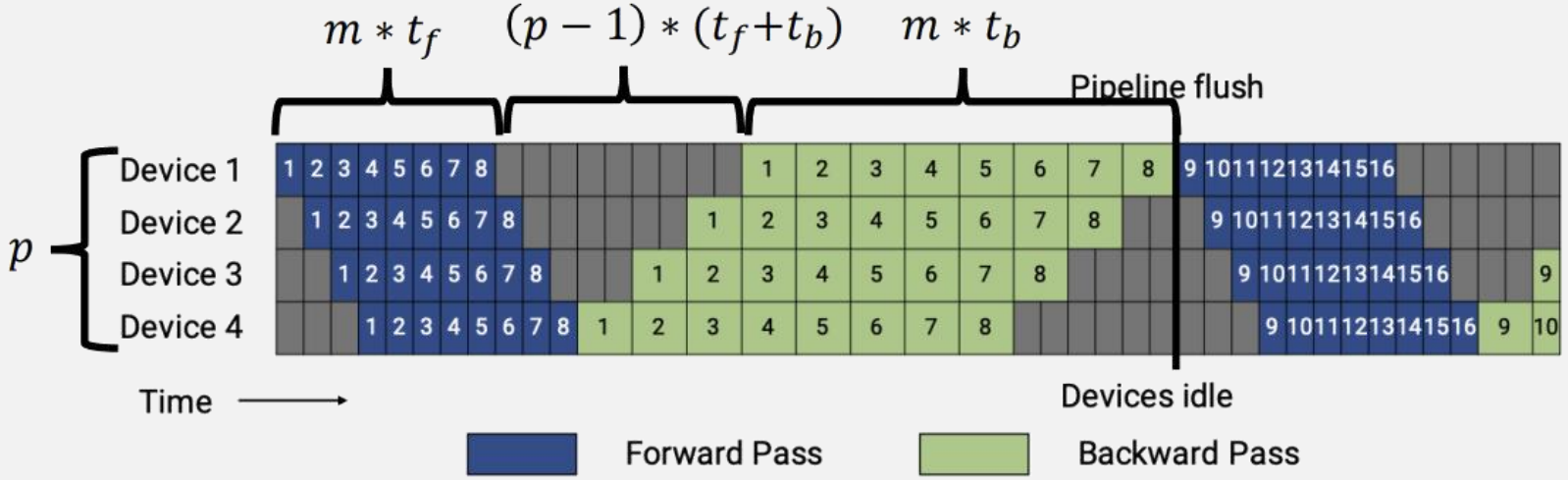


$$\boldsymbol{BubbleFraction} = \frac{(\boldsymbol{p}-\boldsymbol{1}) * (\boldsymbol{t_f}+\boldsymbol{t_b})}{\boldsymbol{m} * \boldsymbol{t_f} + \boldsymbol{m} * \boldsymbol{t_b}} = \frac{\boldsymbol{p}-\boldsymbol{1}}{\boldsymbol{m}}$$

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism

提升管道并行效率

- m 迷你批次中的微批次数量
  - 增加迷你批次大小或减少微批次数量
  - 注意事项：过大的迷你批次可能导致准确率下降；过小的微批次会降低GPU利用率
- p 管道阶段数量
  - 减少管道深度
  - 注意事项：增加阶段规模

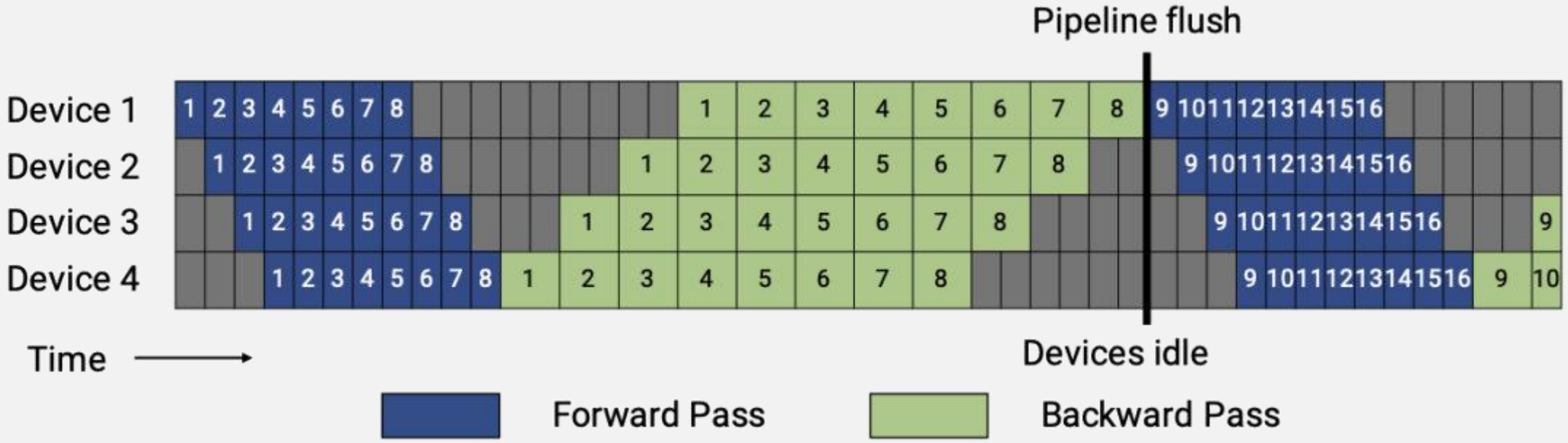


$$\boldsymbol{BubbleFraction} = \frac{(\boldsymbol{p}-\boldsymbol{1}) * (\boldsymbol{t_f}+\boldsymbol{t_b})}{\boldsymbol{m} * \boldsymbol{t_f} + \boldsymbol{m} * \boldsymbol{t_b}} = \frac{\boldsymbol{p}-\boldsymbol{1}}{\boldsymbol{m}}$$

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism

管道模型并行性：内存需求

- 问题：在反向传播之前，我们需要保留所有微批次的中间激活值



能否优化管道调度方案以降低内存需求?

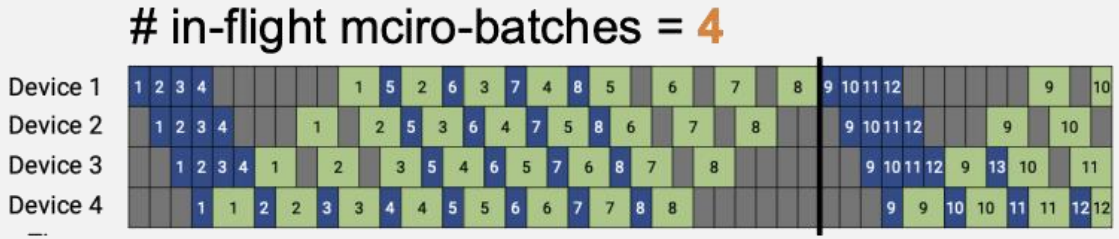
GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism

采用1F1B调度方案的管道并行性

- 稳态下采用一进一退机制
- 将飞行中的微批次数量限制在管道深度范围内
- 降低管道并行处理的内存占用
- 不减少管道气泡

我们能否减少管道泡沫?

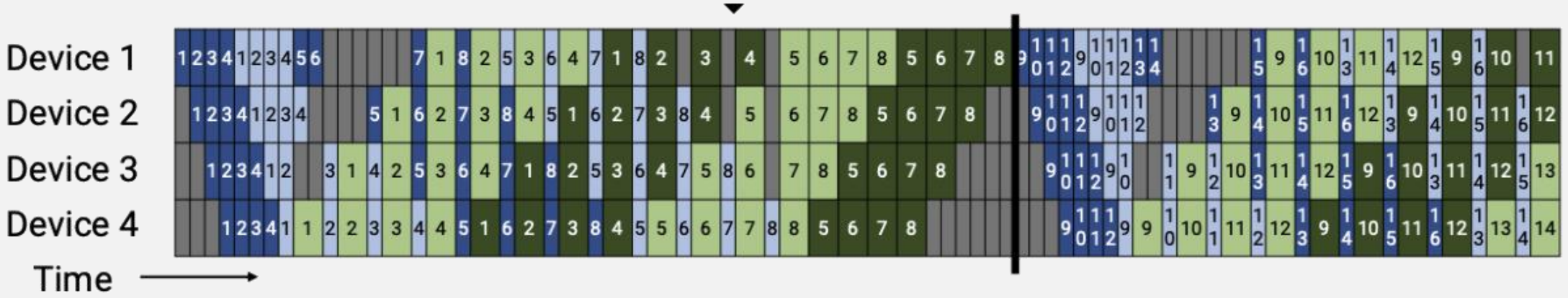
基于GPipe调度的管道并行性



采用1F1B调度方案的流水线并行性

采用交错1F1B调度方案的管道并行性

- 将每个阶段进一步划分为v个子阶段
- 每个子阶段的正向（反向）时间为 $\frac{t_f}{v}$ ($\frac{t_b}{v}$)



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

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

缩短泡沫时间，代价是增加沟通

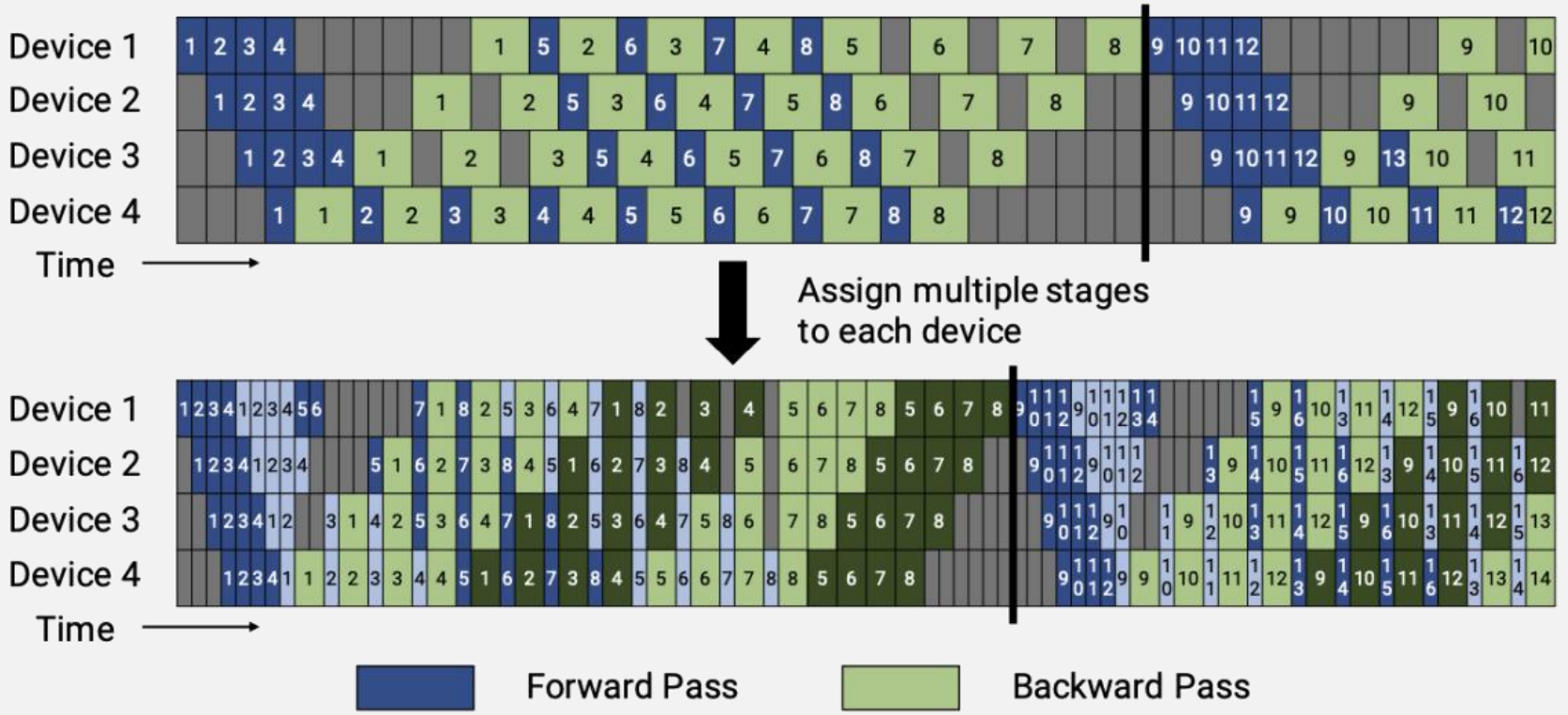
采用交错1F1B调度方案的管道并行性

采用1F1B调度方案的管道并行性

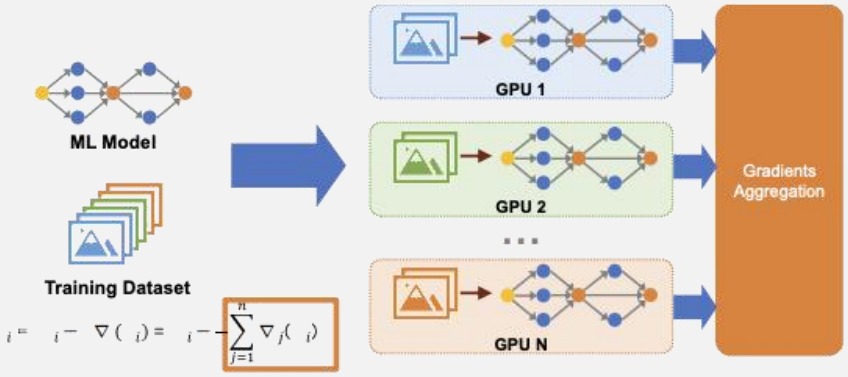
$$BubbleFraction = \frac{p-1}{m}$$

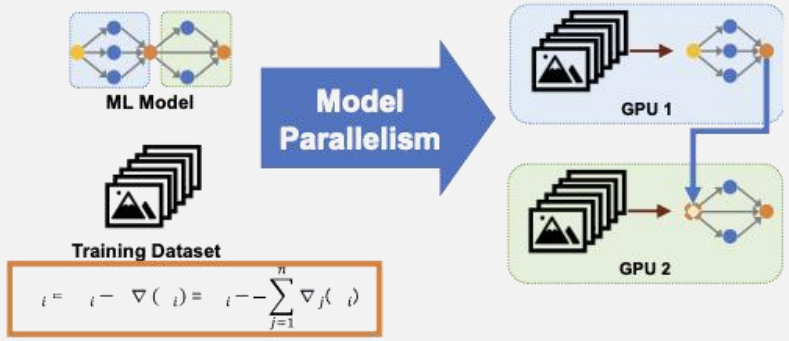
采用交错1F1B调度方案的管道并行性

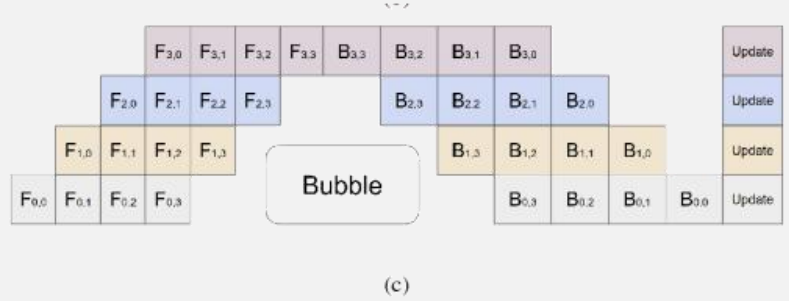
$$BubbleFraction = \frac{1}{v} * \frac{p-1}{m}$$



总结：对比数据/张量模型/管道模型并行性

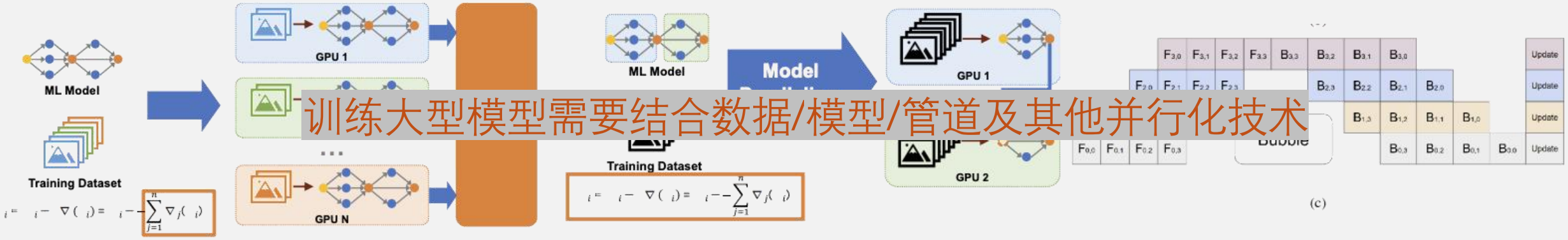






	数据并行性	张量模型并行性	管道模型并行性
pros	√ 可大规模并行化 √ 在正向/反向推导过程中无需通信	√ 支持训练大型模型 √ 适用于参数数量庞大的模型	√ 支持大规模批量训练 √ 适用于深度学习模型的高效训练
cons	❖ 不要为无法装入GPU的模型工作 ❖ 不要为参数数量庞大的模型进行缩放	❖ 并行化能力有限；无法扩展至大量GPU ❖ 需要在正向/反向传播中传输中间结果	❖ 有限利用率：气泡在前进/后退

总结：对比数据/张量模型/管道模型并行性



训练大型模型需要结合数据/模型/管道及其他并行化技术

	数据并行性	张量模型并行性	管道模型并行性
pros	√ 可大规模并行化 √ 在正向/反向推导过程中无需通信	√ 支持训练大型模型 √ 适用于参数数量庞大的模型	√ 支持大规模批量训练 √ 适用于深度学习模型的高效训练
cons	❖ 不要为无法装入GPU的模型工作 ❖ 不要为参数数量庞大的模型进行缩放	❖ 并行化能力有限；无法扩展至大量GPU ❖ 需要在正向/反向传播中传输中间结果	❖ 有限利用率：气泡在前进/后退

示例：DeepSpeed中的3D并行性